ThinLTO

A Framework for Scalable and Incremental Link-Time Optimization

Teresa Johnson   Mehdi Amini   Xinliang David Li

ThinLTO

Towards Always-Enabled LTO

A Framework for Scalable and Incremental Link-Time Optimization

Teresa Johnson   Mehdi Amini   Xinliang David Li

LLVM LTO: in a Nutshell

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

LLVM LTO: in a Nutshell

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

clang -cc1

main.o

ELF / MachO / COFF

1010110101010101110101101011
0101010101110101101011010101
0101110101101011010101010111
0101101011010101010111010110
1011010101010111101101011011
1011010101010111101101011011

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

clang -cc1

test1.o

ELF / MachO / COFF

1010110101010101110101101011
0101010101110101101011010101
0101110101101011010101010111
0101101011010101010111010110
1011010101010111101101011011
1011010101010111101101011011

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1

test2.o

ELF / MachO / COFF

1010110101010101110101101011
0101010101110101101011010101
0101110101101011010101010111
0101101011010101010111010110
1011010101010111101101011011
1011010101010111101101011011

LLVM LTO: in a Nutshell

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

clang -cc1 **-flto**

main.o

ELF / MachO / COFF

1010110101010101110101101011
0101010101110101101011010101
0101110101101011010101010111
0101101011010101010111010110
1011010101010111101101011011
1011010101010111101101011011

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

clang -cc1 **-flto**

test1.o

ELF / MachO / COFF

1010110101010101110101101011
0101010101110101101011010101
0101110101101011010101010111
0101101011010101010111010110
1011010101010111101101011011
1011010101010111101101011011

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 **-flto**

test2.o

ELF / MachO / COFF

1010110101010101110101101011
0101010101110101101011010101
0101110101101011010101010111
0101101011010101010111010110
1011010101010111101101011011
1011010101010111101101011011

LLVM LTO: in a Nutshell

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 **-flto**

clang -cc1 **-flto**

clang -cc1 **-flto**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

test2.o

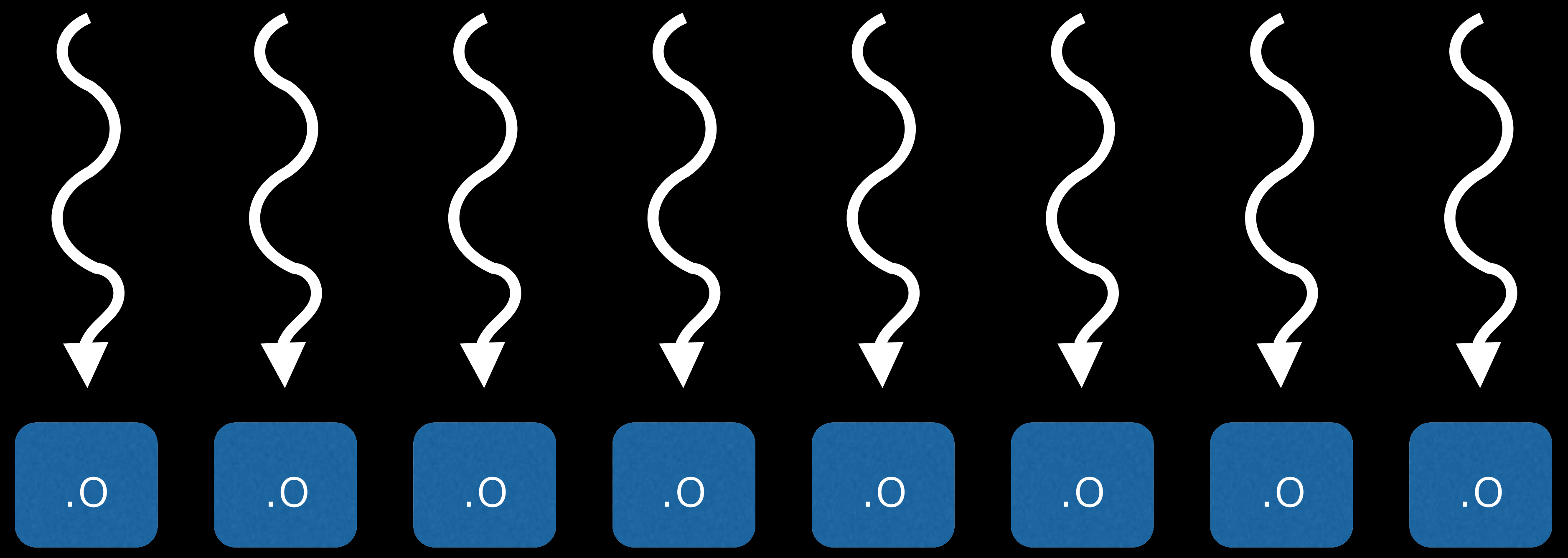
```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

.o files are generated, but they are actually raw bitcode files

static archive will contain these bitcode files

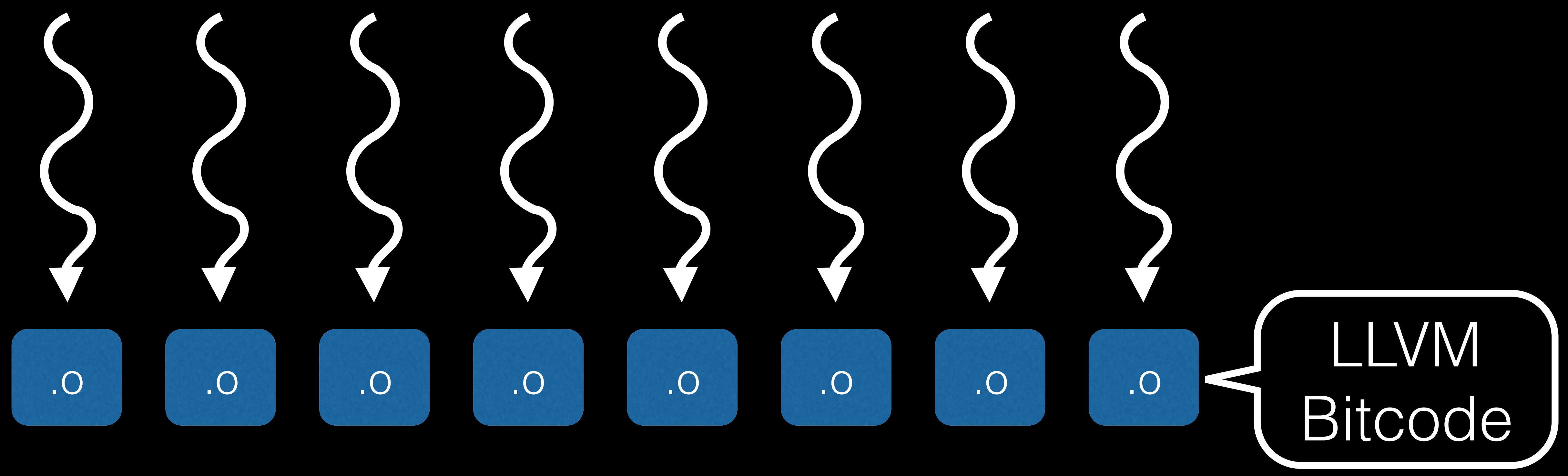
LLVM LTO: in a Nutshell

LLVM LTO: in a Nutshell



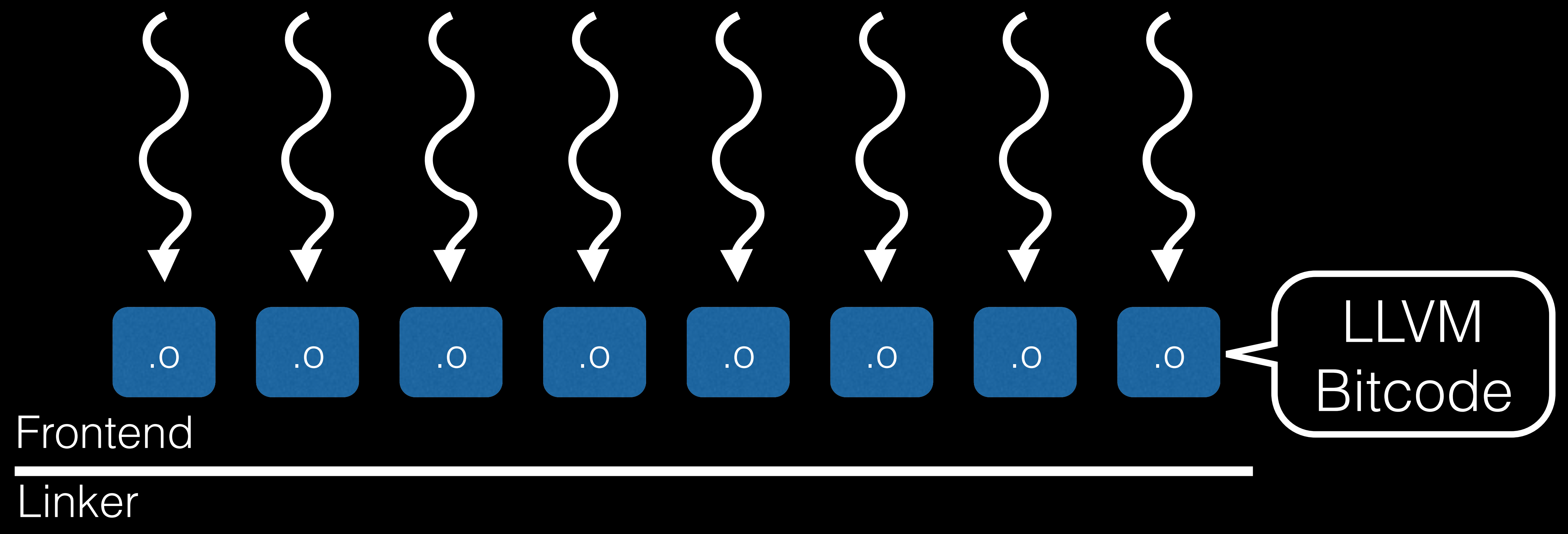
Highly parallel
frontend processing
+ initial optimizations

LLVM LTO: in a Nutshell



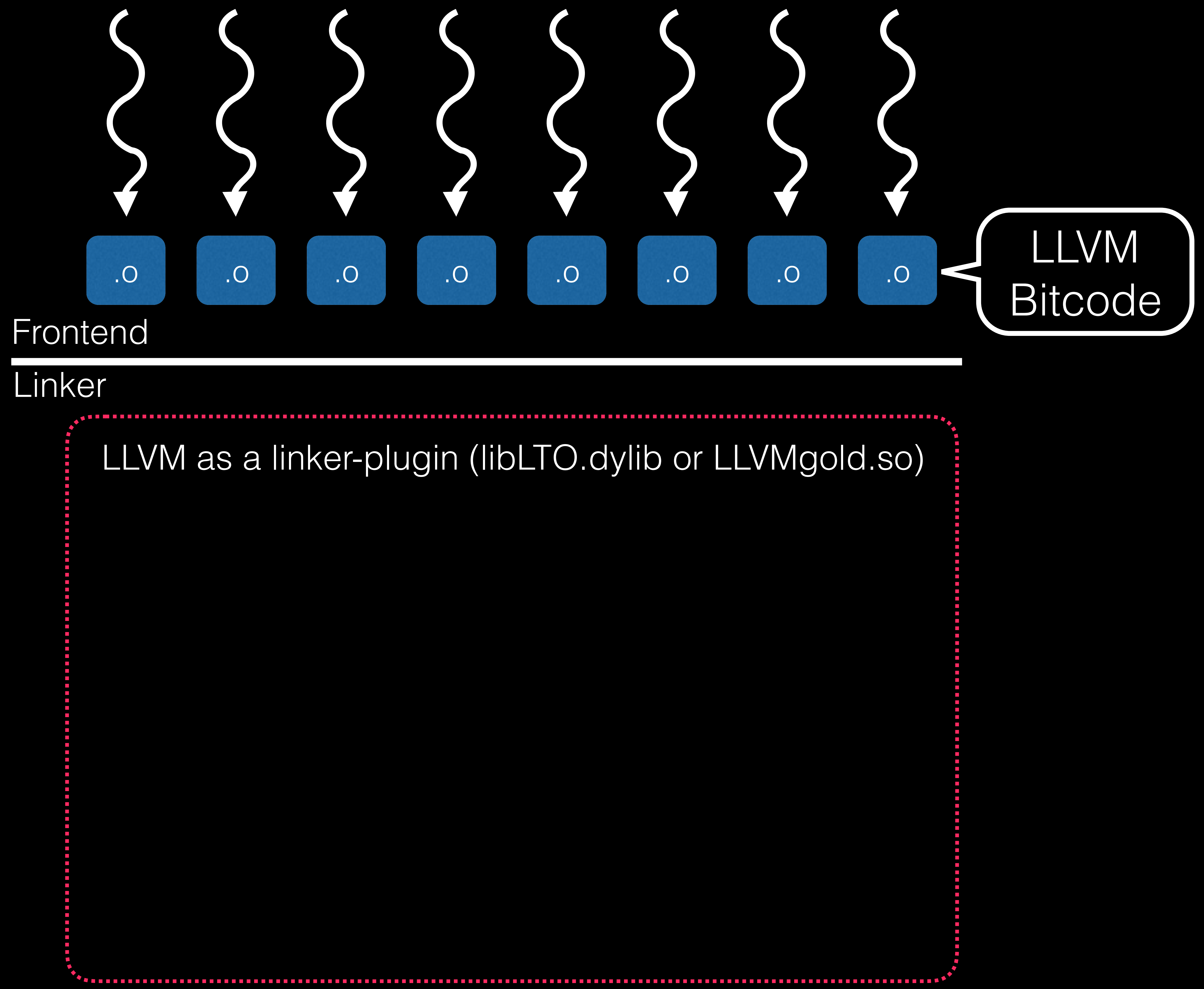
Highly parallel
frontend processing
+ initial optimizations

LLVM LTO: in a Nutshell



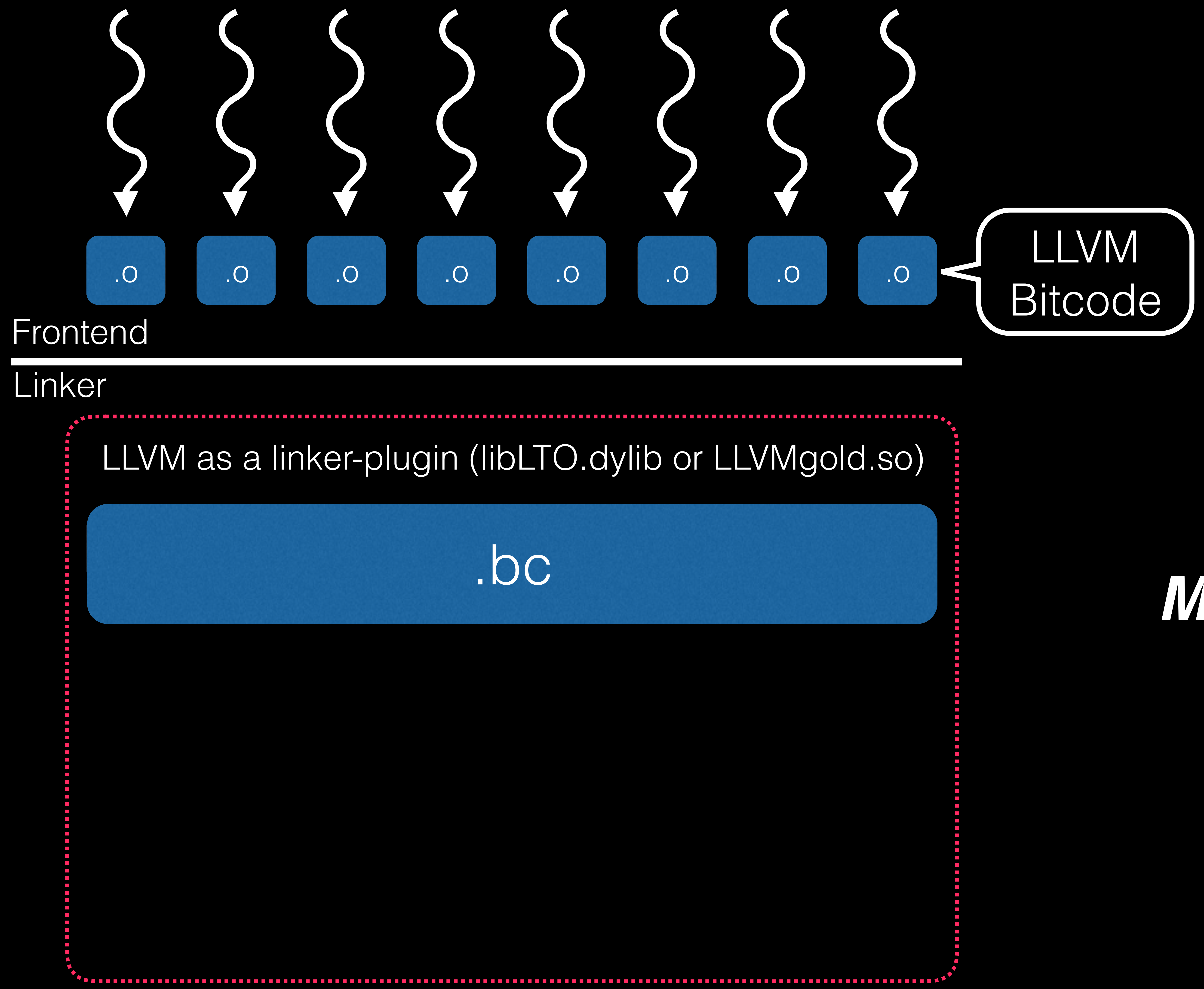
Highly parallel
frontend processing
+ initial optimizations

LLVM LTO: in a Nutshell



Highly parallel
frontend processing
+ initial optimizations

LLVM LTO: in a Nutshell

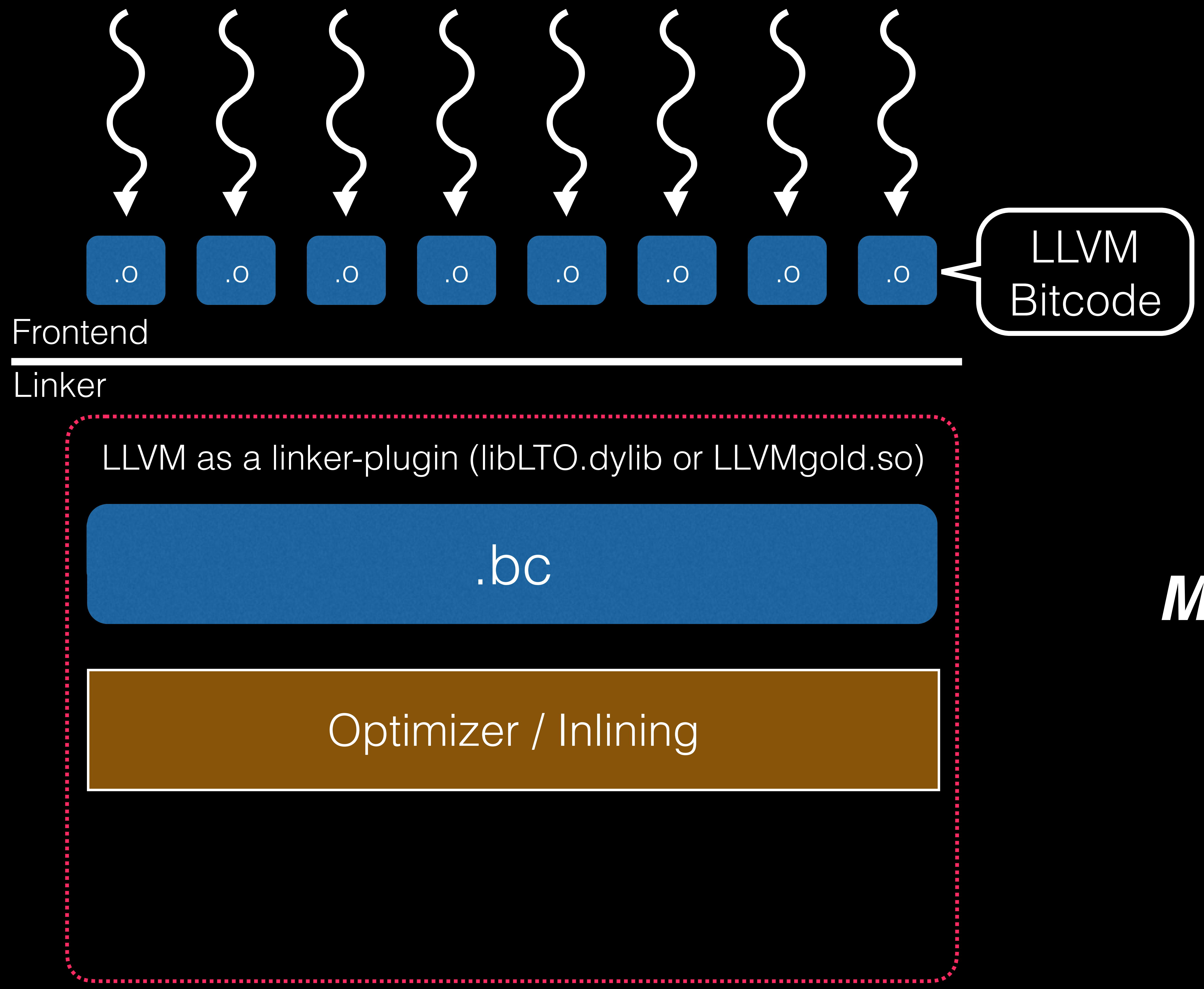


Highly parallel
frontend processing
+ initial optimizations

Link all bitcode in
one single Module

***Monolithic** LTO Implementation*

LLVM LTO: in a Nutshell



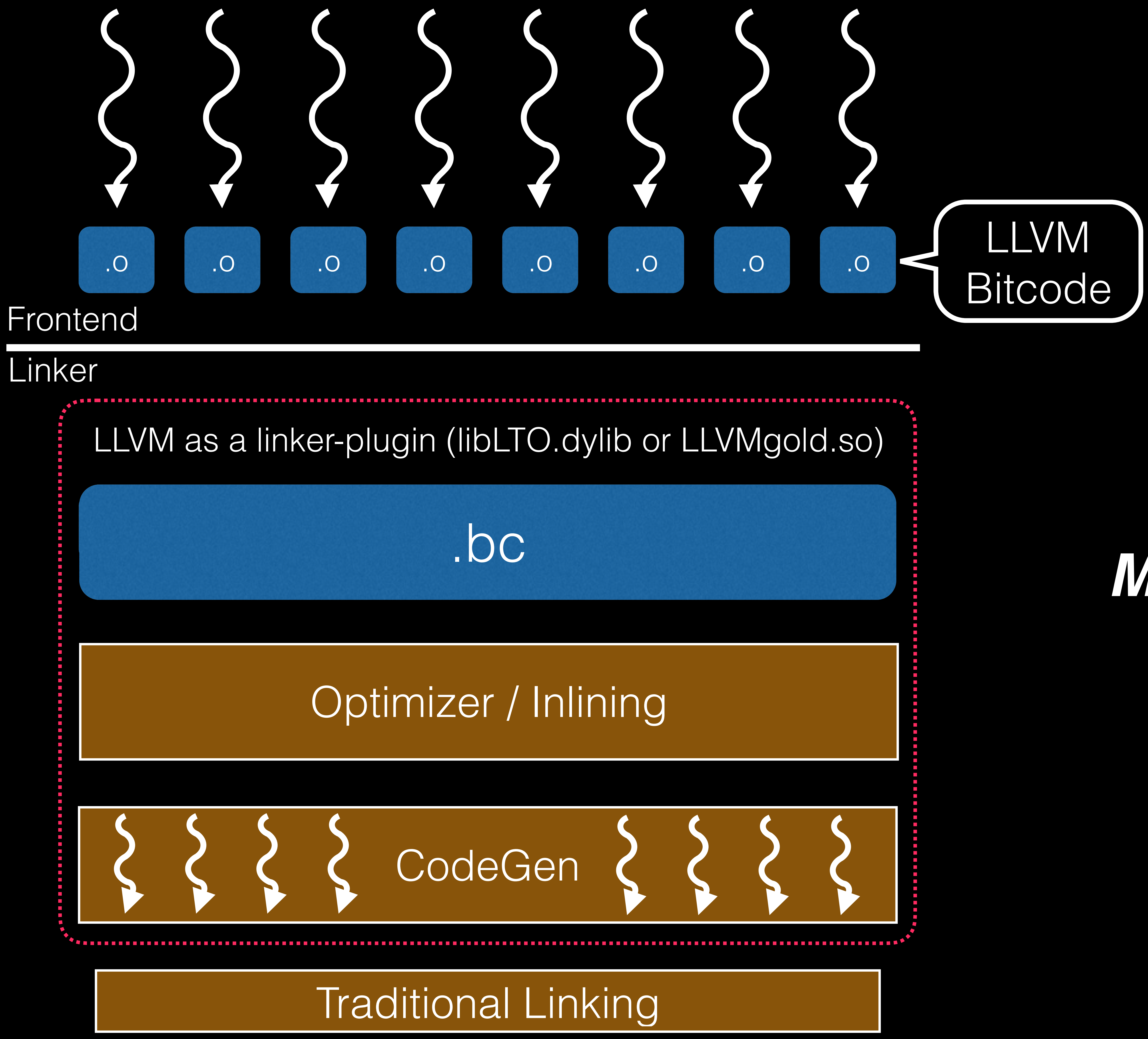
Highly parallel
frontend processing
+ initial optimizations

Link all bitcode in
one single Module

Monolithic *LTO Implementation*

Single-threaded very boring
usual optimizations

LLVM LTO: in a Nutshell



Highly parallel frontend processing + initial optimizations

Link all bitcode in one single Module

Monolithic *LTO Implementation*

Single-threaded very boring usual optimizations

Potentially threaded CodeGen

Why LTO?

Benefits

Why LTO?

Benefits

- Binary size: inherent dead-stripping and auto-hidden visibility *via* internalization.

Why LTO?

Benefits

- Binary size: inherent dead-stripping and auto-hidden visibility *via* internalization.
- Performance! 10% boost is common.

→ Removes module optimization boundaries via Cross-Module Optimization (CMO)

→ Most of benefit comes from cross-module inlining

Why LTO?

Benefits

- Binary size: inherent dead-stripping and auto-hidden visibility *via* internalization.
- Performance! 10% boost is common.

→ Removes module optimization boundaries via Cross-Module Optimization (CMO)

→ Most of benefit comes from cross-module inlining

Performance Improvements - Execution Time	Δ	Previous	Current	σ
SingleSource/Benchmarks/Shootout-C++/objinst	-96.80%	0.0937	0.0030	-
SingleSource/Benchmarks/Dhrystone/fldry	-93.17%	0.0893	0.0061	0.0002
SingleSource/Benchmarks/Dhrystone/dry	-79.21%	0.0534	0.0111	0.0000
SingleSource/Benchmarks/Misc/matmul_f64_4x4	-52.76%	0.0199	0.0094	0.0008
SingleSource/UnitTests/Vector/build2	-46.56%	0.0421	0.0225	0.0002
MultiSource/Benchmarks/tramp3d-v4/tramp3d-v4	-35.45%	0.1839	0.1187	0.0026
SingleSource/Benchmarks/Misc/flops-6	-33.50%	0.0403	0.0268	0.0001
MultiSource/Benchmarks/Olden/bh/bh	-31.09%	0.1412	0.0973	0.0002
MultiSource/Benchmarks/Olden/power/power	-30.45%	0.0867	0.0603	0.0000

Why LTO?

Benefits

- Binary size: inherent dead-stripping and auto-hidden visibility *via* internalization.
- Performance! 10% boost is common.

➔ Removes module optimization boundaries via Cross-Module Optimization (CMO)

➔ Most of benefit comes from cross-module inlining

Performance Improvements - Execution Time	Δ	Previous	Current	σ
SingleSource/Benchmarks/Shootout-C++/objinst	-96.80%	0.0937	0.0030	-
SingleSource/Benchmarks/Dhrystone/fldry	-93.17%	0.0893	0.0061	0.0002
SingleSource/Benchmarks/Dhrystone/dry	-79.21%	0.0534	0.0111	0.0000
SingleSource/Benchmarks/Misc/matmul_f64_4x4	-52.76%	0.0199	0.0094	0.0008
SingleSource/UnitTests/Vector/build2	-46.56%	0.0421	0.0225	0.0002
MultiSource/Benchmarks/tramp3d-v4/tramp3d-v4	-35.45%	0.1839	0.1187	0.0026
SingleSource/Benchmarks/Misc/flops-6	-33.50%	0.0403	0.0268	0.0001
MultiSource/Benchmarks/Olden/bh/bh	-31.09%	0.1412	0.0973	0.0002
MultiSource/Benchmarks/Olden/power/power	-30.45%	0.0867	0.0603	0.0000

Why LTO?

Benefits

- Binary size: inherent dead-stripping and auto-hidden visibility *via* internalization.
- Performance! 10% boost is common.

➔ Removes module optimization boundaries via Cross-Module Optimization (CMO)

➔ Most of benefit comes from cross-module inlining

Performance Improvements - Execution Time	Δ	Previous	Current	σ
SingleSource/Benchmarks/Shootout-C++/objinst	-96.80%	0.0937	0.0030	-
SingleSource/Benchmarks/Dhrystone/fldry	-93.17%	0.0893	0.0061	0.0002
SingleSource/Benchmarks/Dhrystone/dry	-79.21%	0.0534	0.0111	0.0000
SingleSource/Benchmarks/Misc/matmul_f64_4x4	-52.76%	0.0199	0.0094	0.0008
SingleSource/UnitTests/Vector/build2	-46.56%	0.0421	0.0225	0.0002
MultiSource/Benchmarks/tramp3d-v4/tramp3d-v4	-35.45%	0.1839	0.1187	0.0026
SingleSource/Benchmarks/Misc/flops-6	-33.50%	0.0403	0.0268	0.0001
MultiSource/Benchmarks/Olden/bh/bh	-31.09%	0.1412	0.0973	0.0002
MultiSource/Benchmarks/Olden/power/power	-30.45%	0.0867	0.0603	0.0000

Single source improvements because global variables can be internalized (better alias analysis, etc.).

➔ LTO is more powerful than “Unity Build” because of **Linker supplied information.**

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```cpp
static signed int i = 0;
void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c
static signed int i = 0;
static void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
static int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c
static signed int i = 0;
static void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
static int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c
static signed int i = 0;
static void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
static int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c
static signed int i = 0;
static void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
static int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c
static signed int i = 0;



static int foo3() {
  foo4();
  return 10;
}
static int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c
static int foo3() {
  foo4();
  return 10;
}
static int foo1(void) {
  int data = 0;


  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```cpp




static int foo3() {
  foo4();
  return 10;
}
static int foo1(void) {
  int data = 0;


  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c
static int foo1(void) {
  int data = 0;


  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c








static int foo1(void) {
  int data = 0;


  data = data + 42;
  return data;
}
static void foo4(void) {
  printf("Hi\n");
}
int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c








static int foo1(void) {
  int data = 0;


  data = data + 42;
  return data;
}



int main() {
  return foo1();
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```c








static int foo1(void) {
  int data = 0;


  data = data + 42;
  return data;
}



int main() {
  return 42;
}
```

Example

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3();
  data = data + 42;
  return data;
}
```

lto.cpp 〉 No Selection

```cpp
int main() {
  return 42;
}
```

Optimization across *module boundaries!*

+

Linker Information

Monolithic LTO: No Free Lunch!

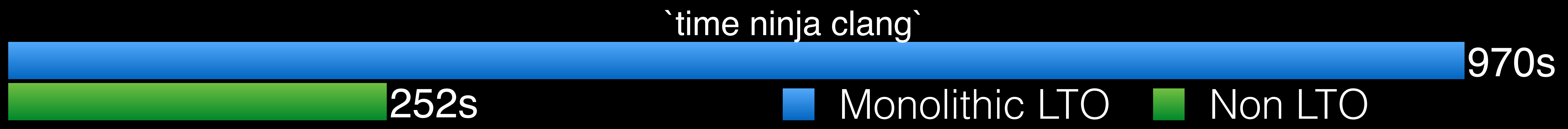
Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

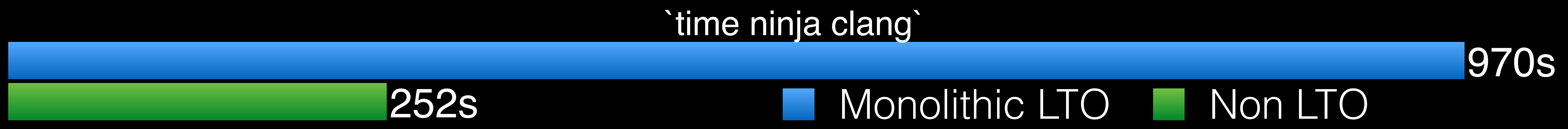
- Slow: inherently serial / can’t be distributed



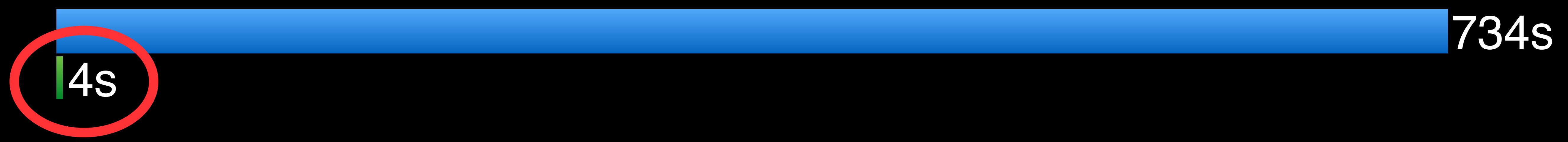
Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

- Slow: inherently serial / can’t be distributed



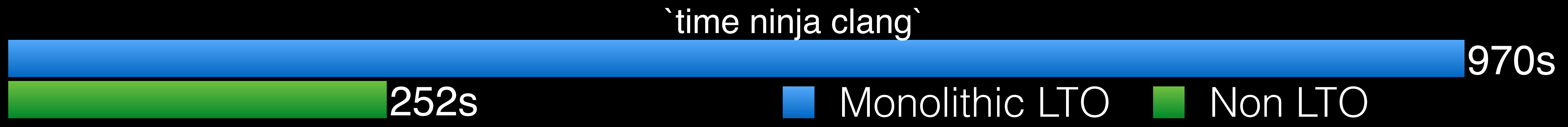
- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.



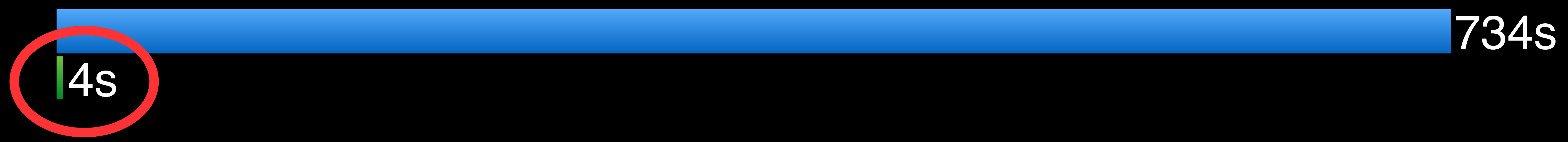
Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

- Slow: inherently serial / can’t be distributed



- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.

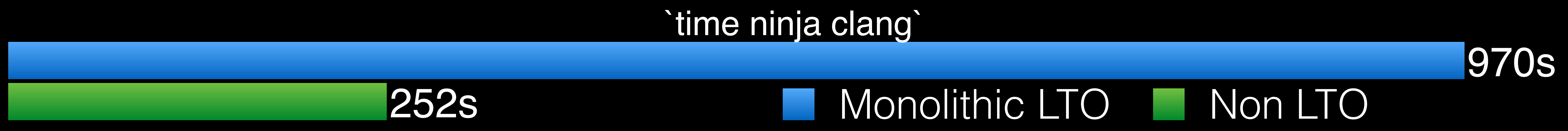


https://www.xkcd.com/303/

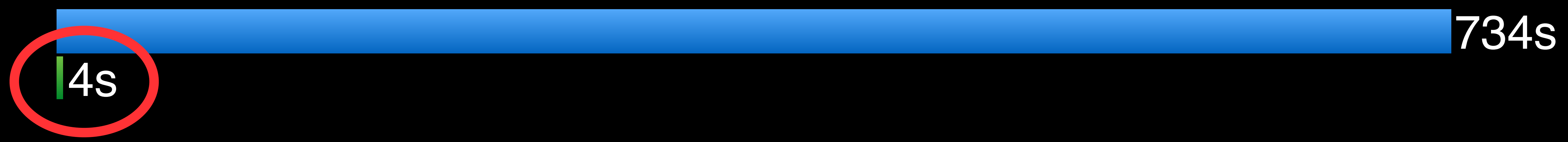
Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

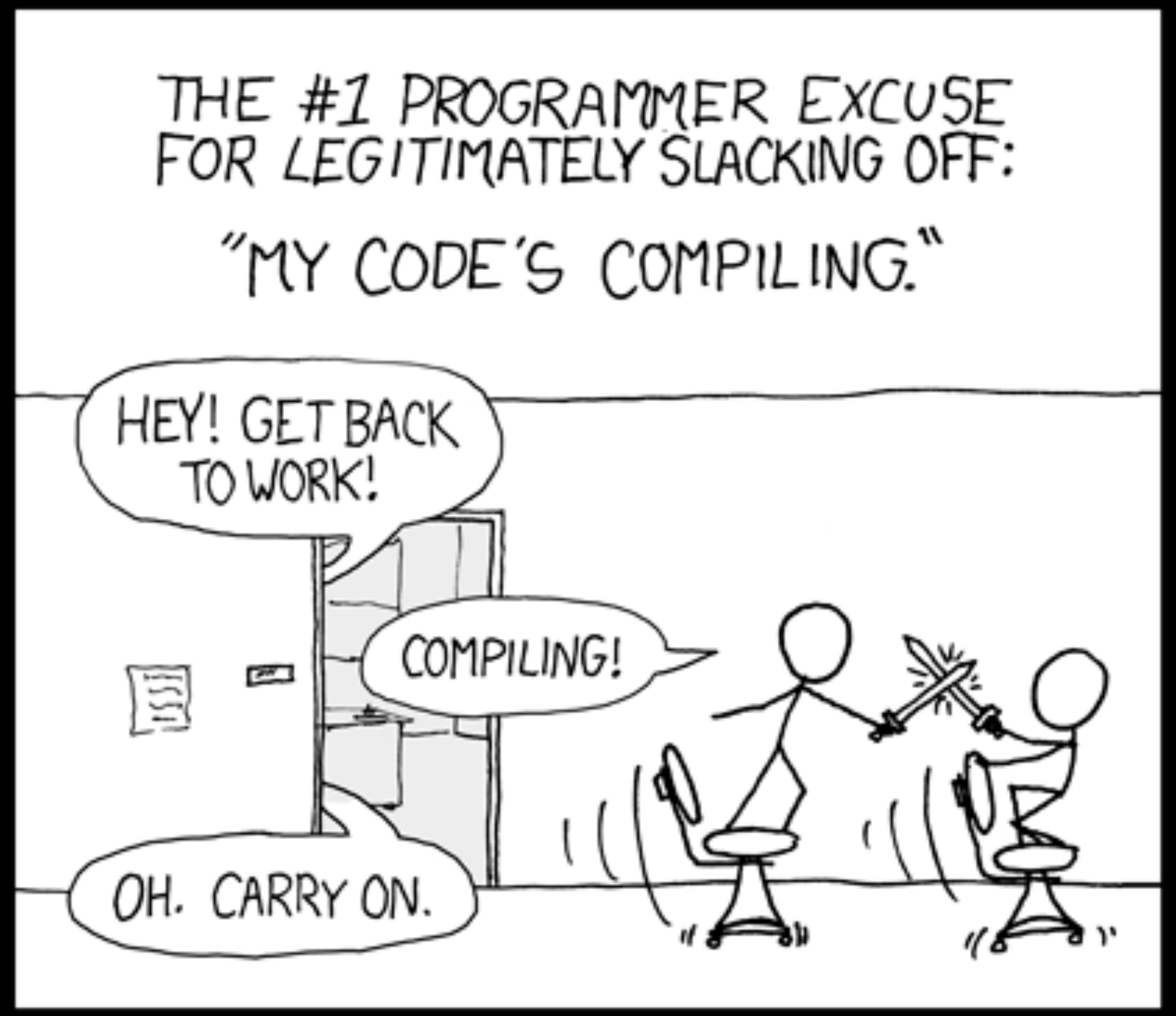
- Slow: inherently serial / can't be distributed



- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.



- Memory hungry: all the program in a single bitcode file in memory.

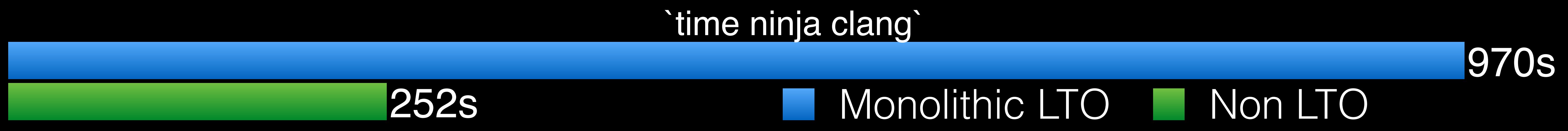


https://www.xkcd.com/303/

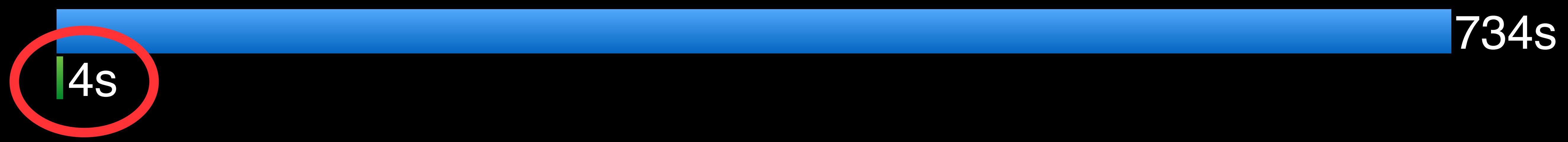
Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

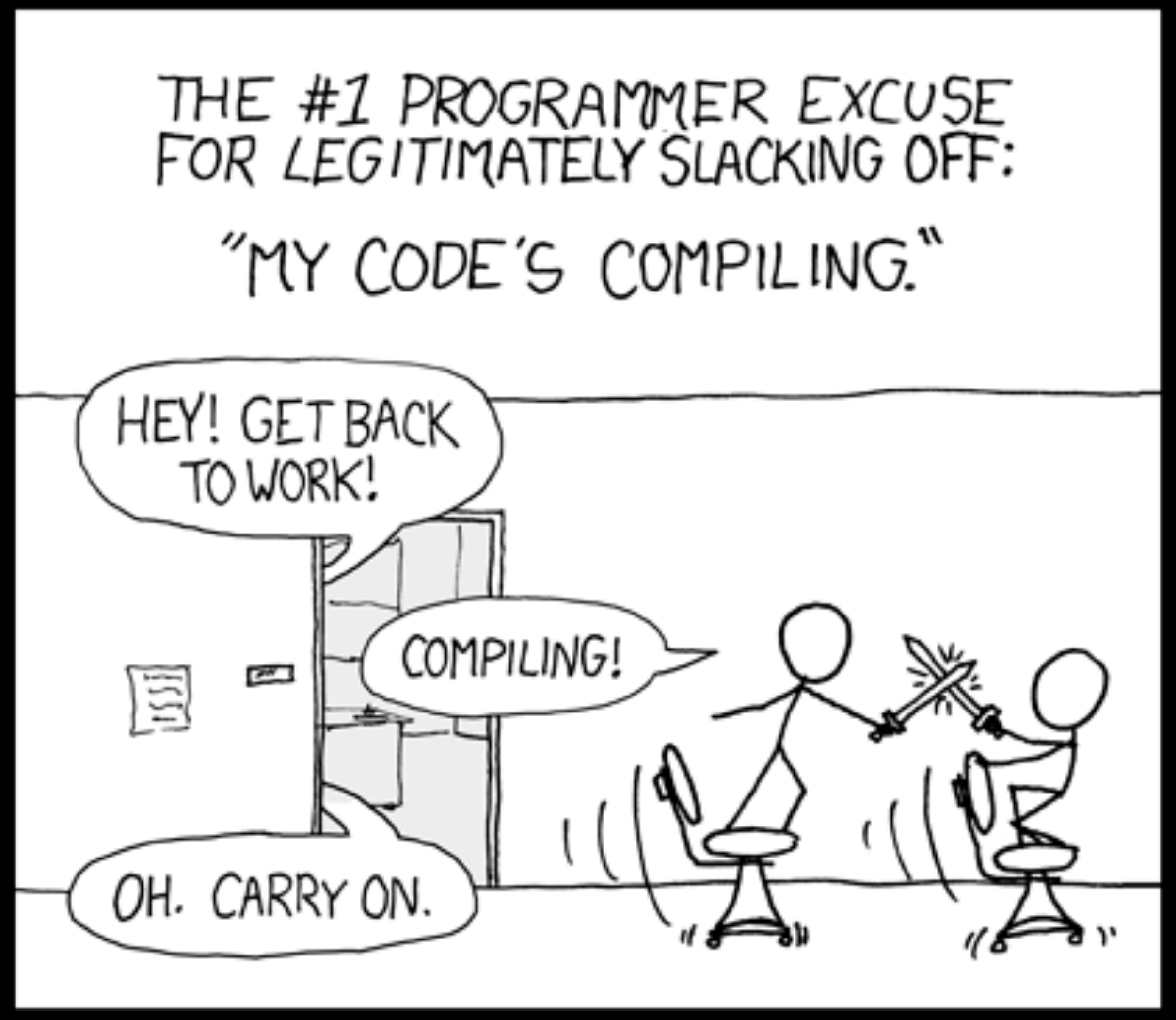
- Slow: inherently serial / can't be distributed



- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.



- Memory hungry: all the program in a single bitcode file in memory.



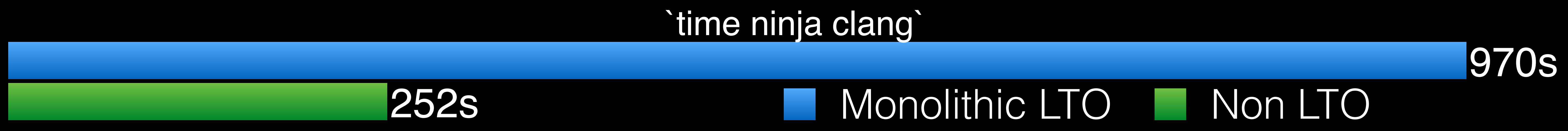
https://www.xkcd.com/303/

Malloc Peak for the LTO Link of the `clang-3.6` binary, with only the X86 backend configured

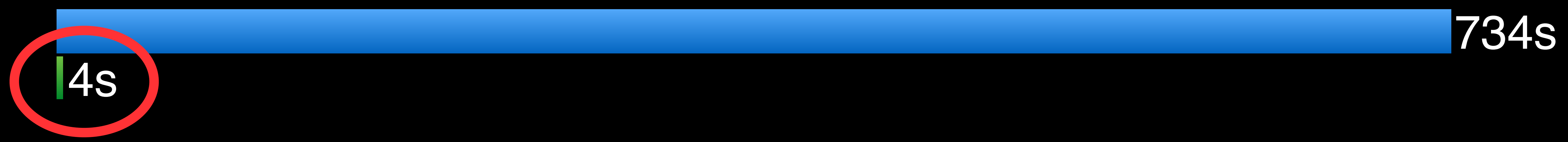
Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

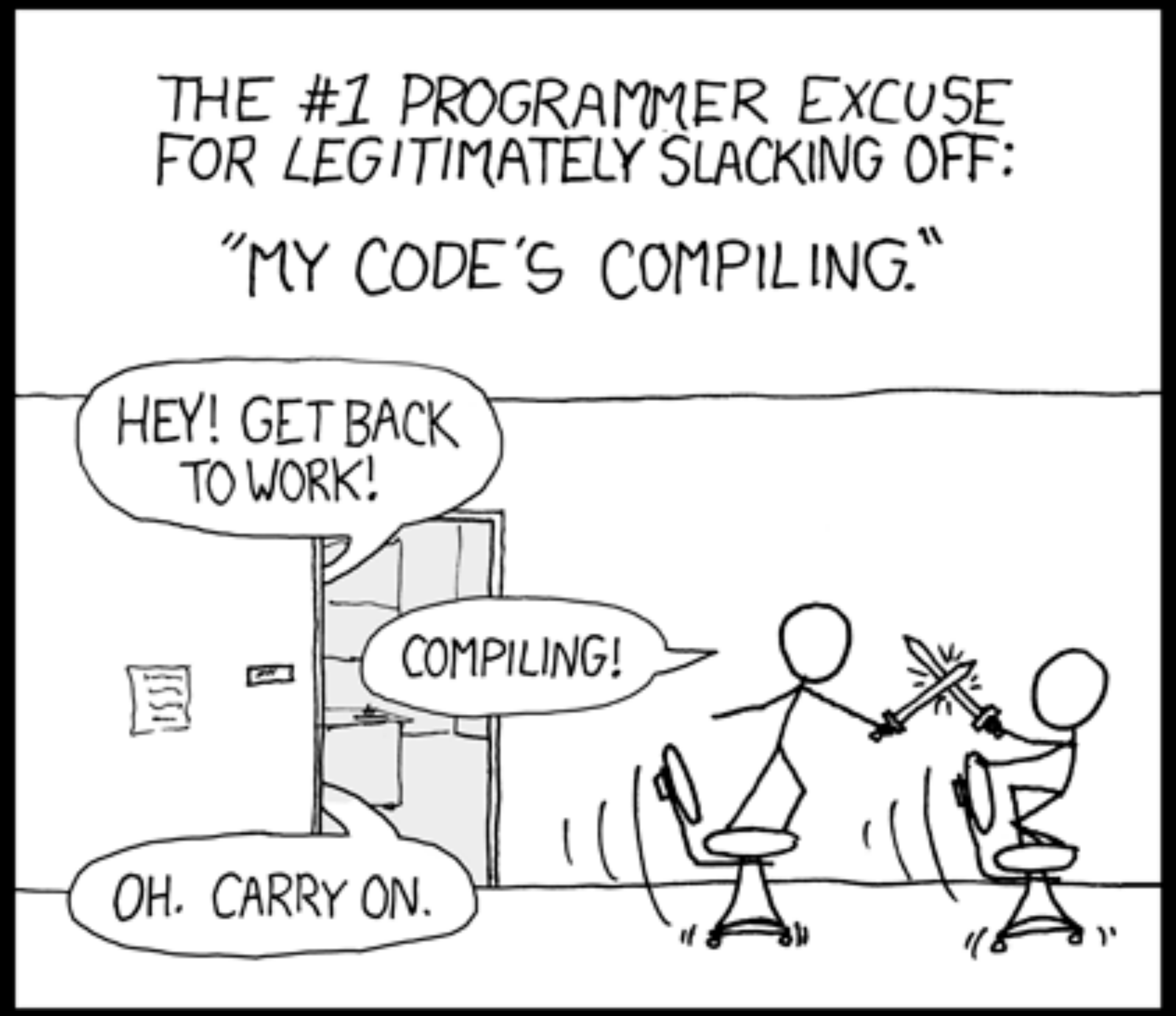
- Slow: inherently serial / can't be distributed



- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.

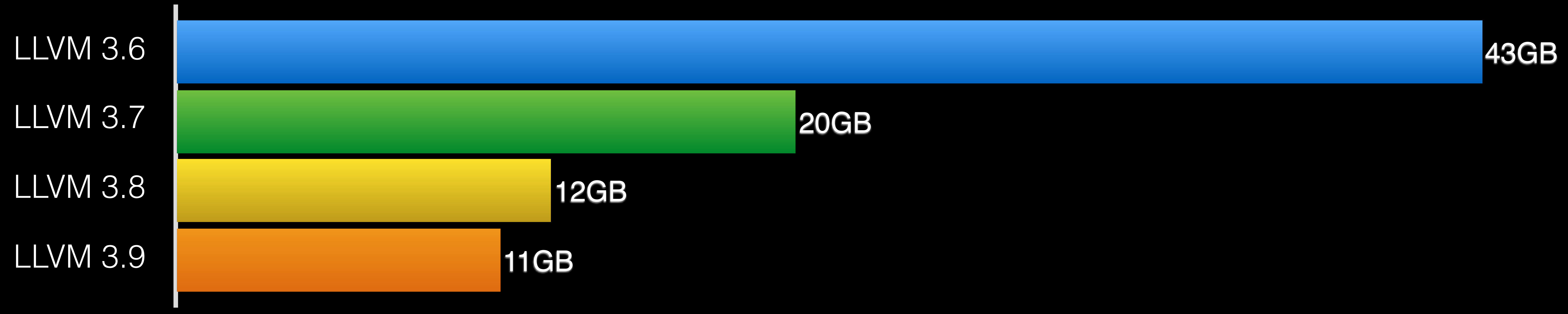


- Memory hungry: all the program in a single bitcode file in memory.



https://www.xkcd.com/303/

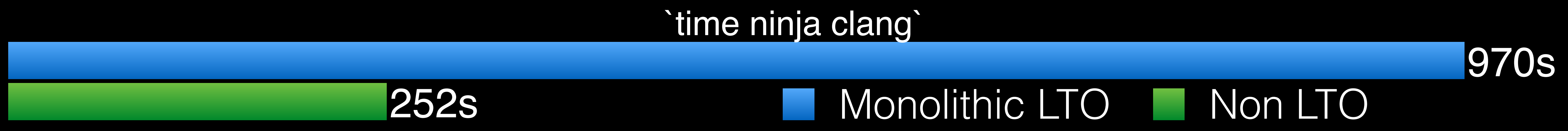
Malloc Peak for the LTO Link of the `clang-3.6` binary, with only the X86 backend configured



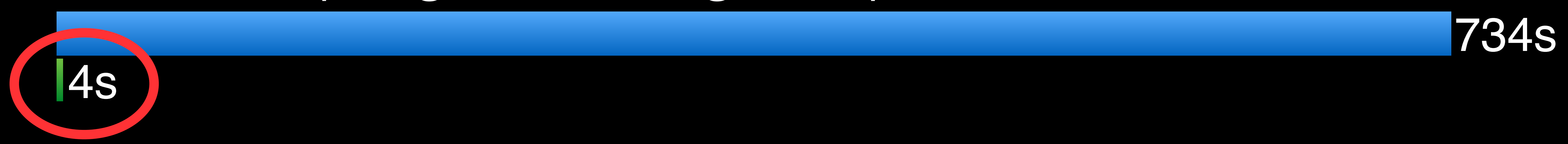
Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: **why?**

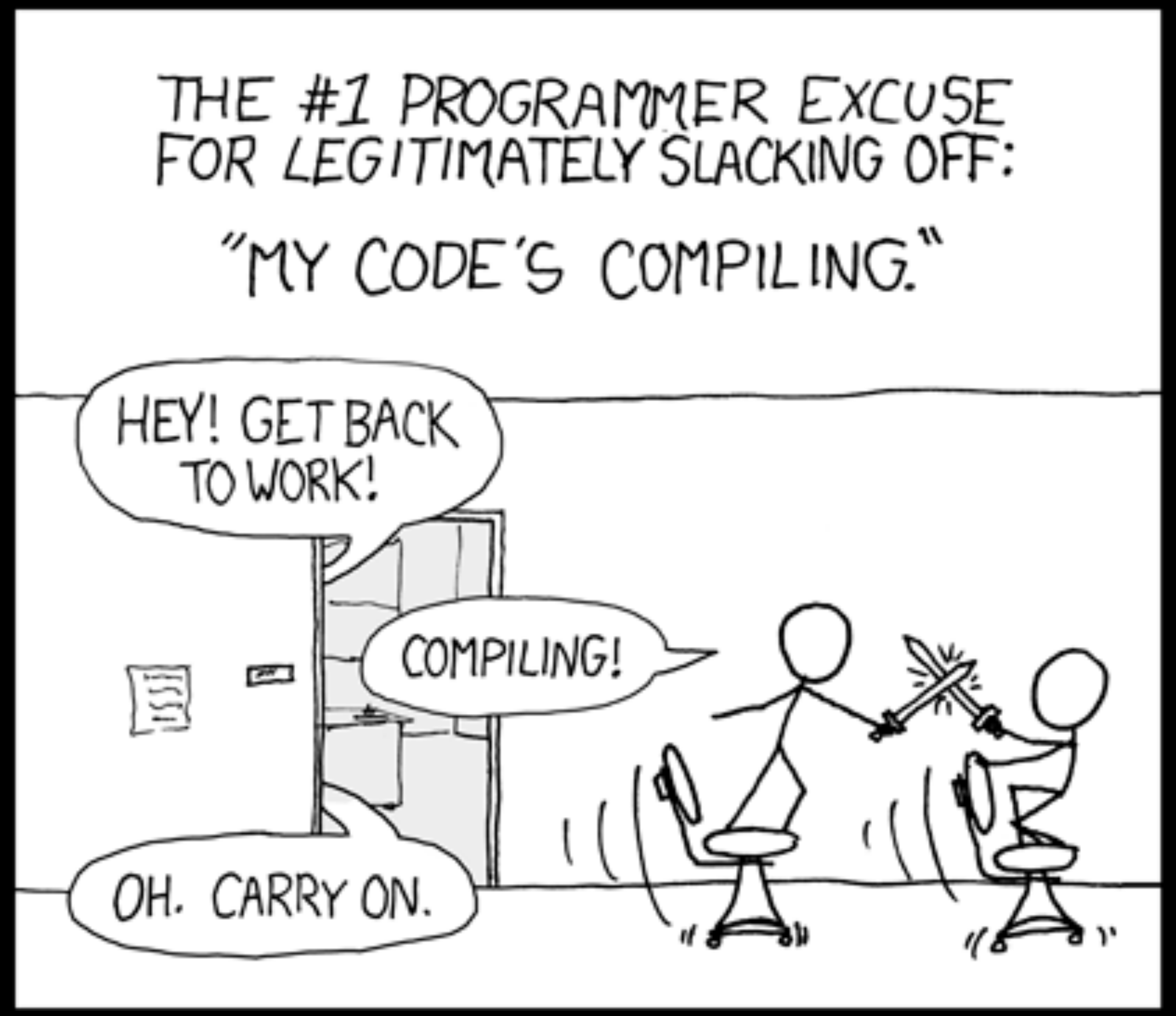
- Slow: inherently serial / can't be distributed



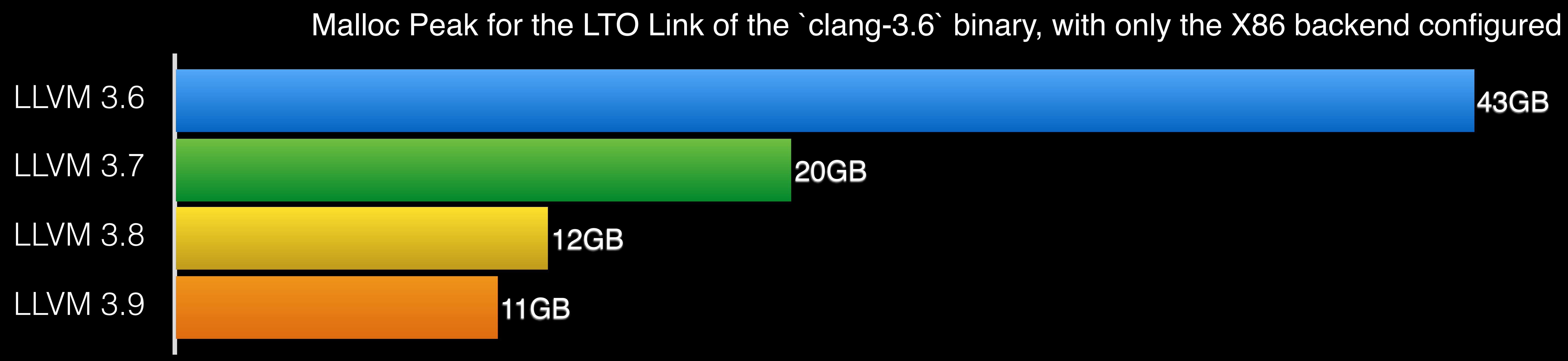
- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.



- Memory hungry: all the program in a single bitcode file in memory.



https://www.xkcd.com/303/



Dead end: linking Chromium with debug info still crashes now after >2h and >50GB mem

Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: why?

- Slow: inherently serial / can’t be distributed

- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.

- Memory hungry: all the program in a single bitcode file in memory.

Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: why?

- Slow: inherently serial / can't be distributed

→ Parallel

- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.

- Memory hungry: all the program in a single bitcode file in memory.

Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: why?

- Slow: inherently serial / can’t be distributed

→ Parallel

- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.

→ Incremental

- Memory hungry: all the program in a single bitcode file in memory.

Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: why?

- Slow: inherently serial / can't be distributed

➔ Parallel

- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.

➔ Incremental

- Memory hungry: all the program in a single bitcode file in memory.

➔ Memory lean

Monolithic LTO: No Free Lunch!

LTO adoption is still low after >10 years of existence: why?

- Slow: inherently serial / can't be distributed

→ Parallel

- Not friendly with incremental build: fix a typo, and see the full program being re-optimized as a whole.

→ Incremental

- Memory hungry: all the program in a single bitcode file in memory.

→ Memory lean

To fulfill this goal of LTO always enabled,

we need a solution designed around these three key features!

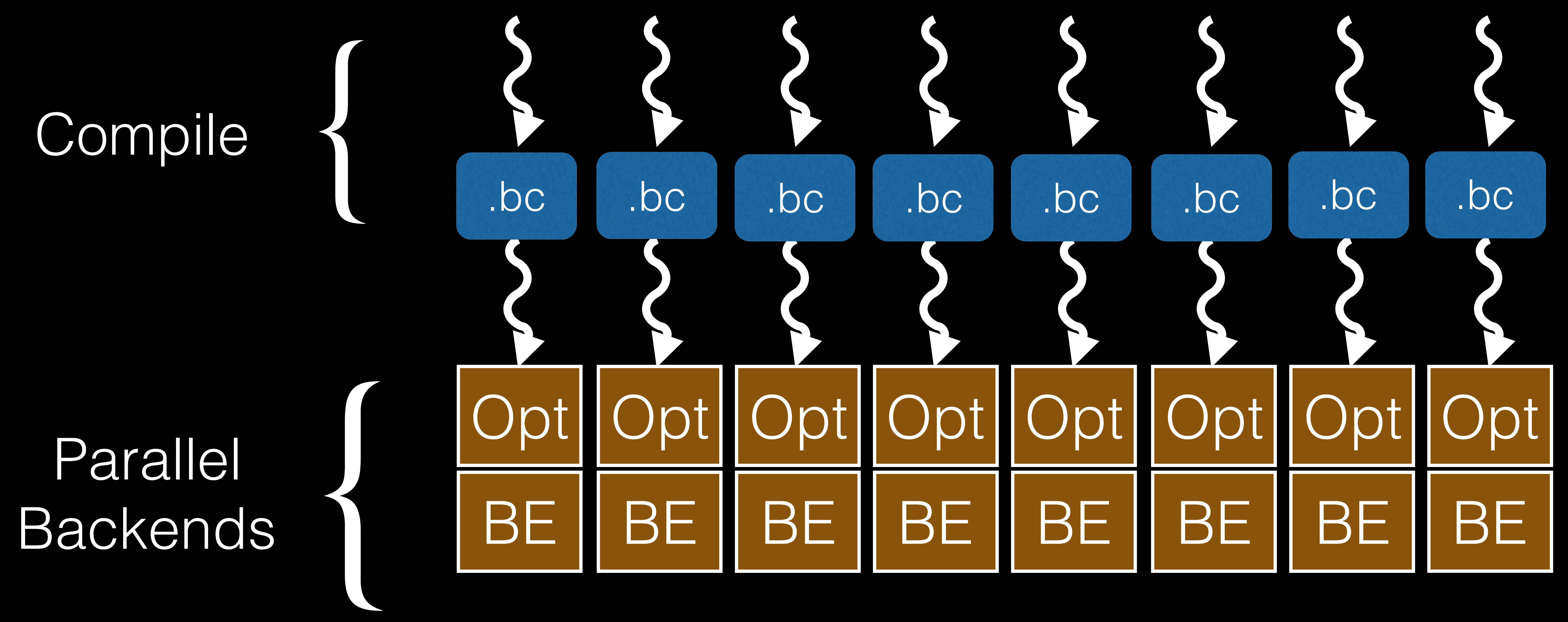
ThinLTO: Design

ThinLTO: Design

- Designed from the start for huge (Google scale) applications

ThinLTO: Design

- Designed from the start for huge (Google scale) applications
- Fully parallel compile step and backends → enables distributed builds

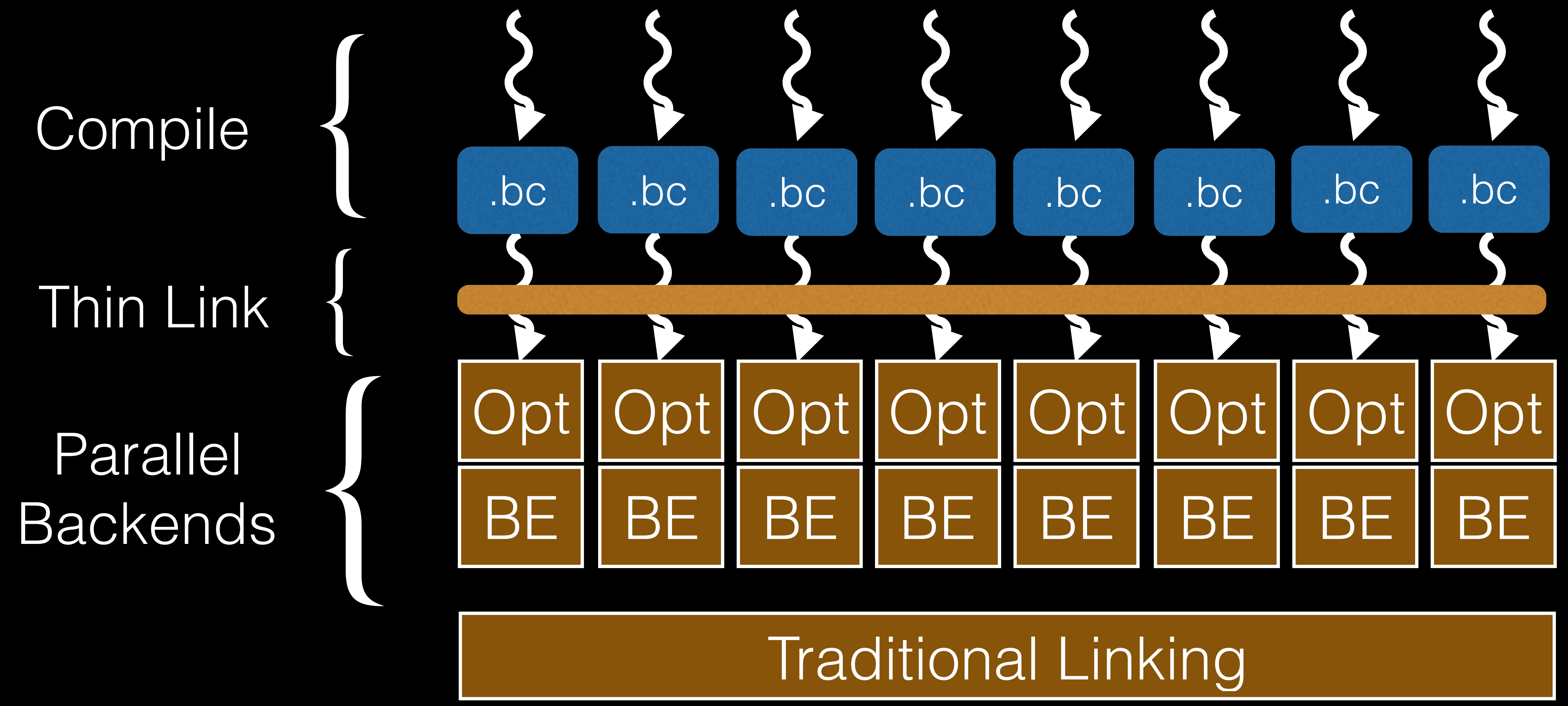


Fully-parallel frontend processing
+ initial optimizations

Fully-parallel (very boring) usual
optimizations and CodeGen

ThinLTO: Design

- Designed from the start for huge (Google scale) applications
- Fully parallel compile step and backends ➔ enables distributed builds



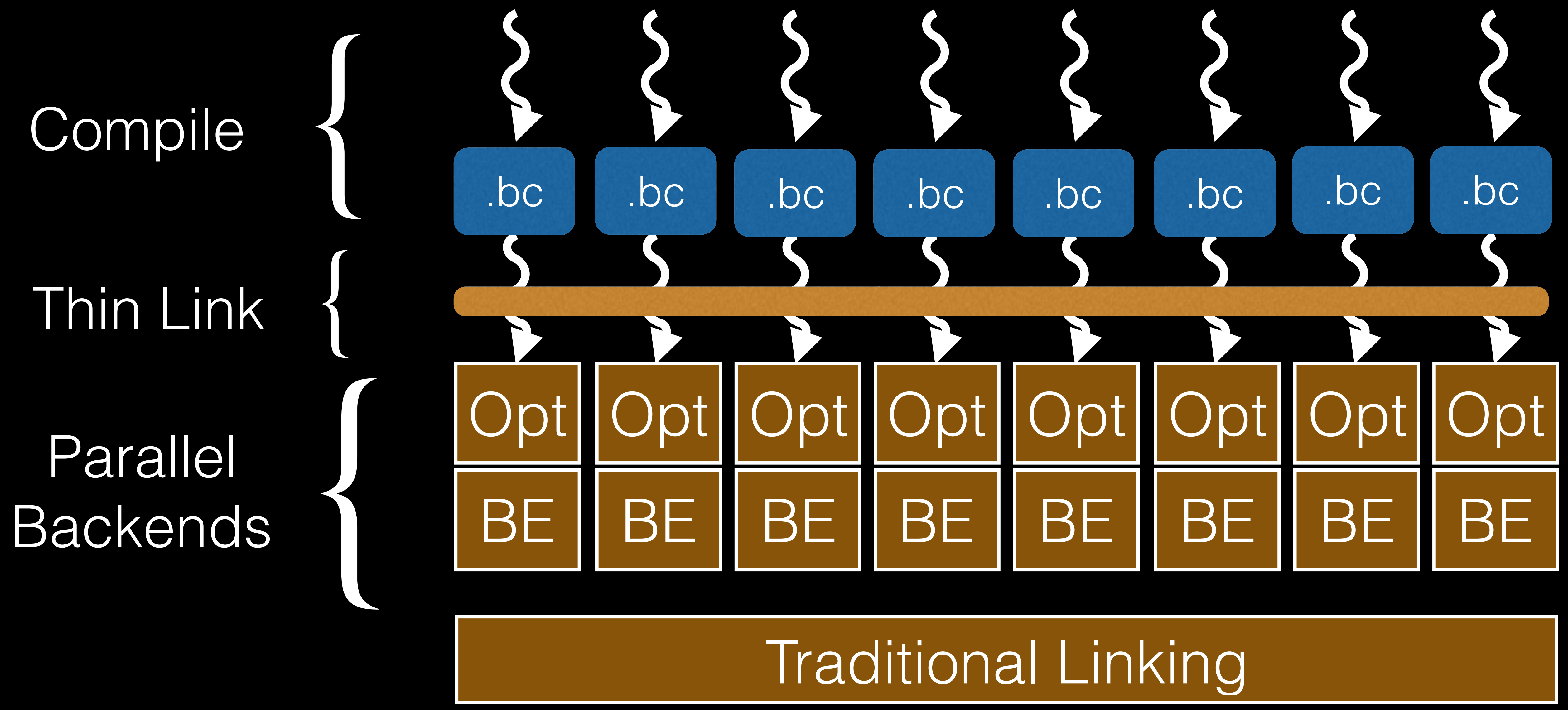
Fully-parallel frontend processing + initial optimizations

Thin serial synchronization step

Fully-parallel (very boring) usual optimizations and CodeGen

ThinLTO: Design

- Designed from the start for huge (Google scale) applications
- Fully parallel compile step and backends → enables distributed builds
- Module is unit of compilation → enables incremental builds



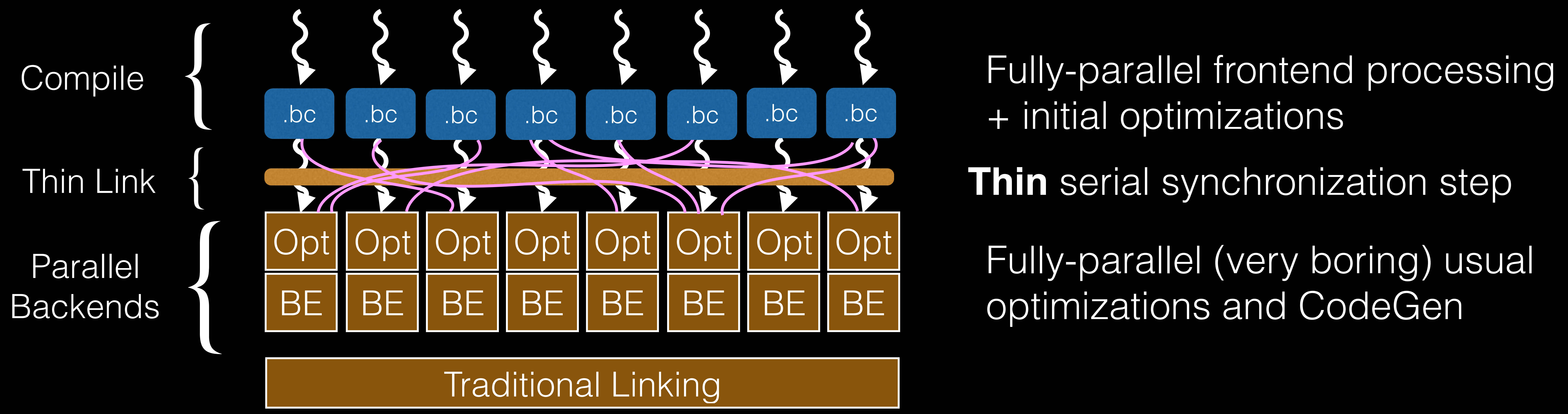
Fully-parallel frontend processing + initial optimizations

Thin serial synchronization step

Fully-parallel (very boring) usual optimizations and CodeGen

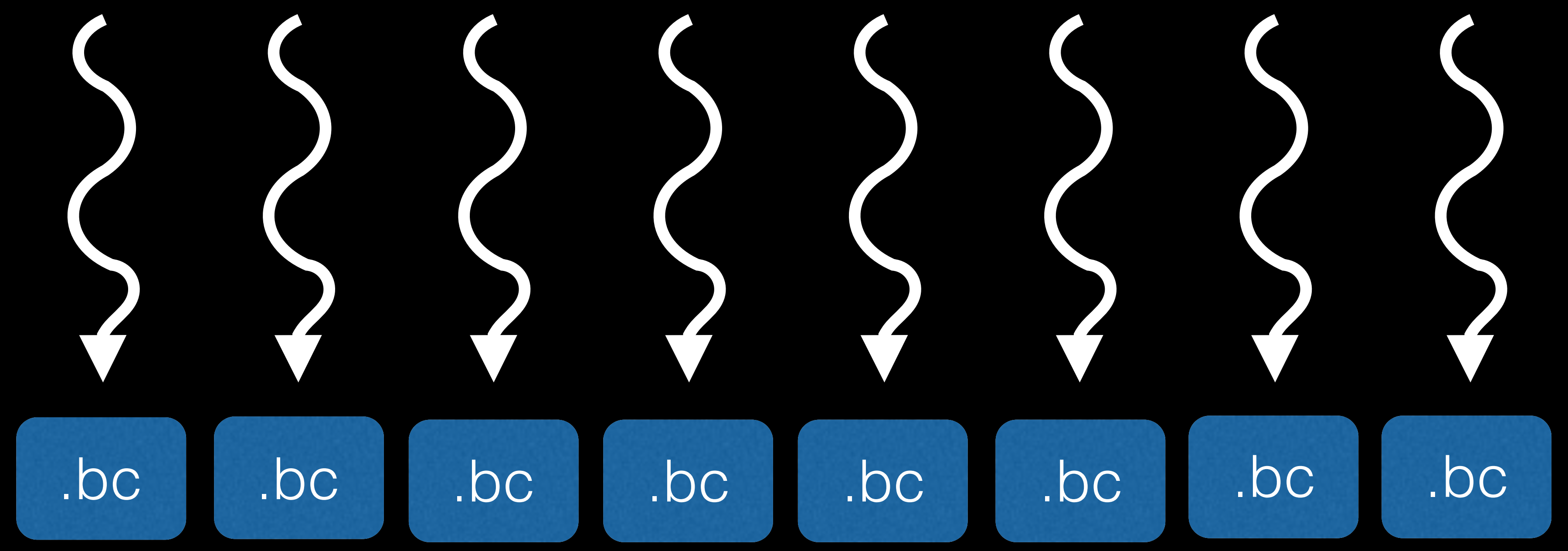
ThinLTO: Design

- Designed from the start for huge (Google scale) applications
- Fully parallel compile step and backends → enables distributed builds
- Module is unit of compilation → enables incremental builds
- Only perform profitable cross module optimization into each module
  → memory scaling



ThinLTO: Overview

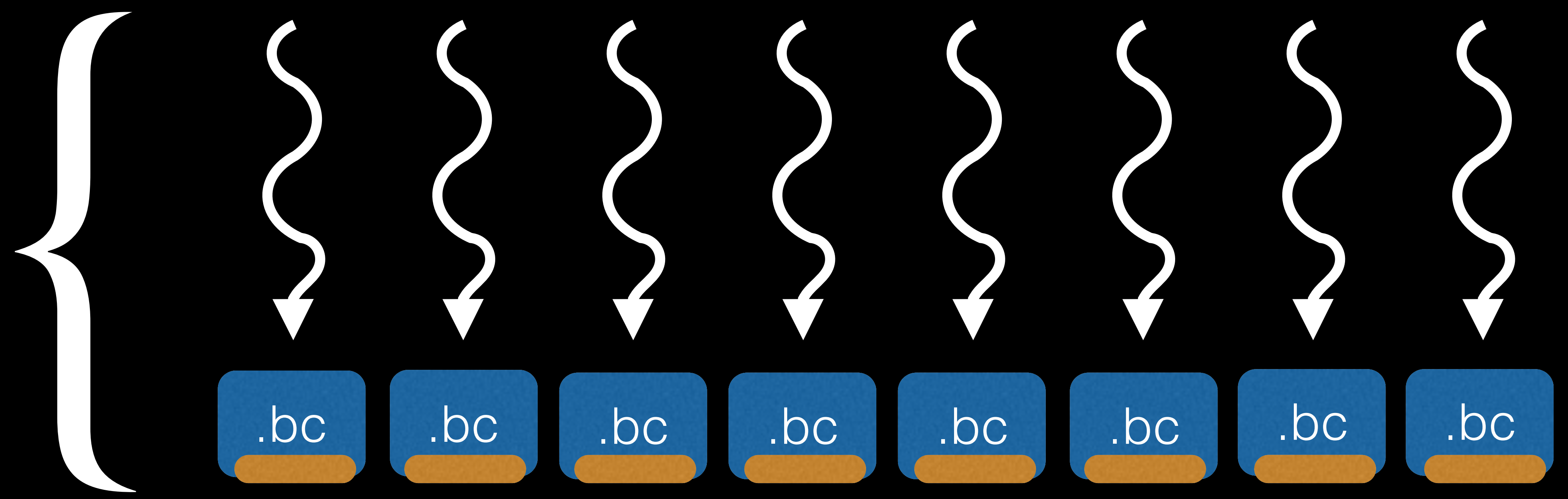
ThinLTO: Overview



Fully-parallel frontend processing
+ initial optimizations

ThinLTO: Overview

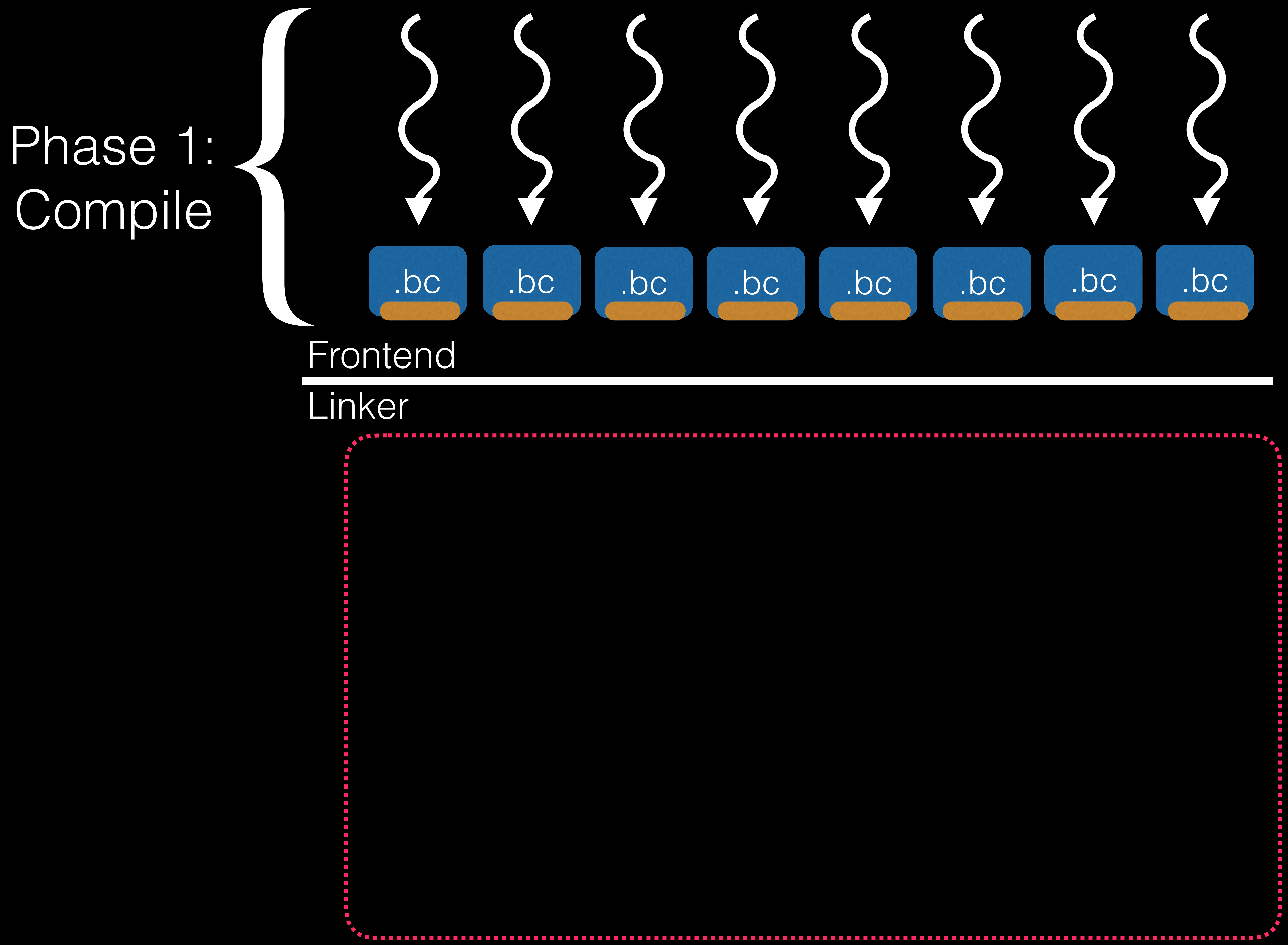
Phase 1: Compile



Fully-parallel frontend processing
+ initial optimizations

Extra per-function summary information
are generated *"on the side"*

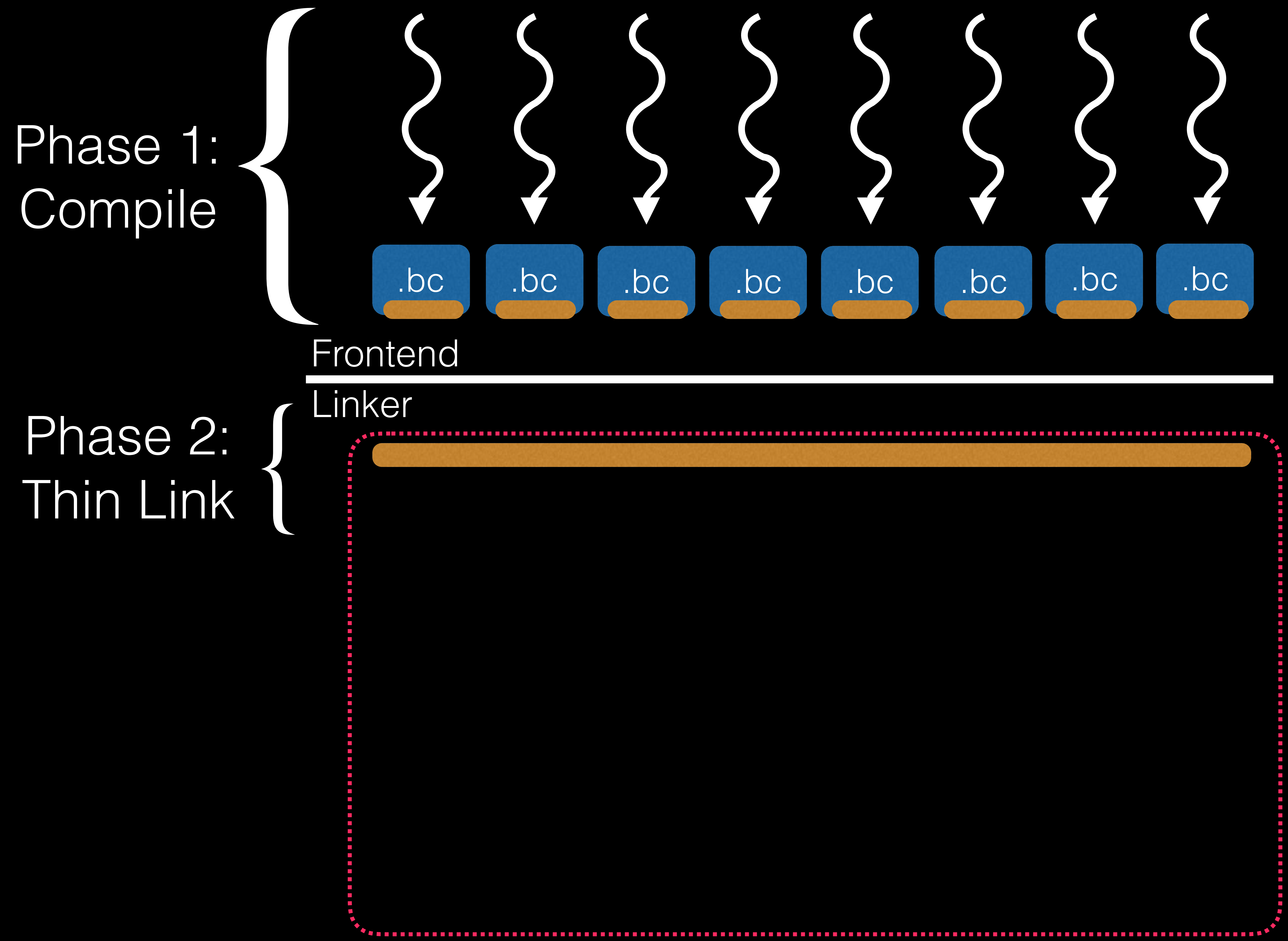
ThinLTO: Overview



Fully-parallel frontend processing
+ initial optimizations

Extra per-function summary information
are generated *"on the side"*

ThinLTO: Overview

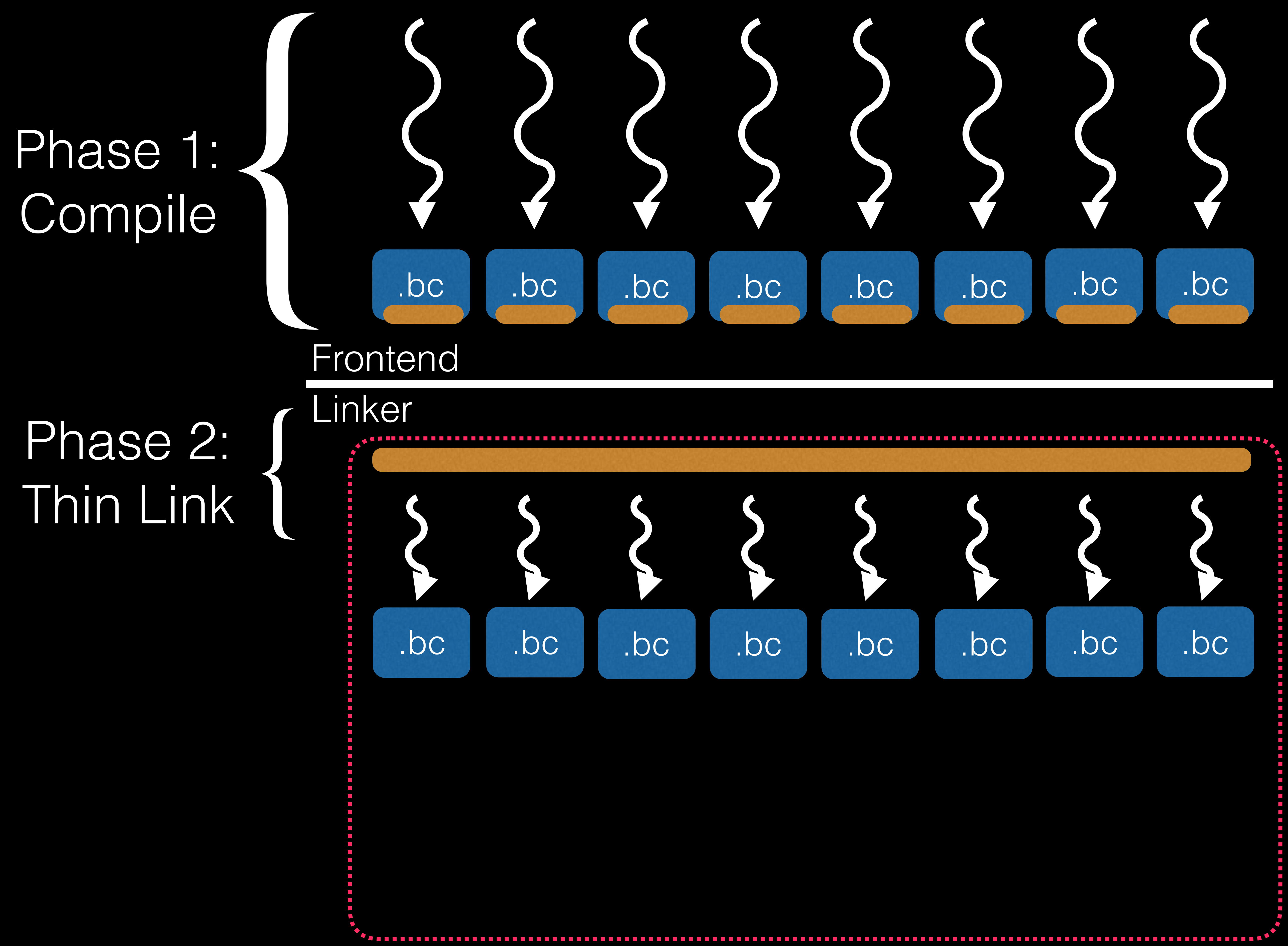


Fully-parallel frontend processing
+ initial optimizations

Extra per-function summary information
are generated *"on the side"*

Link only the summary info
in a giant index: *thin-link*.
No need to parse the IR

ThinLTO: Overview



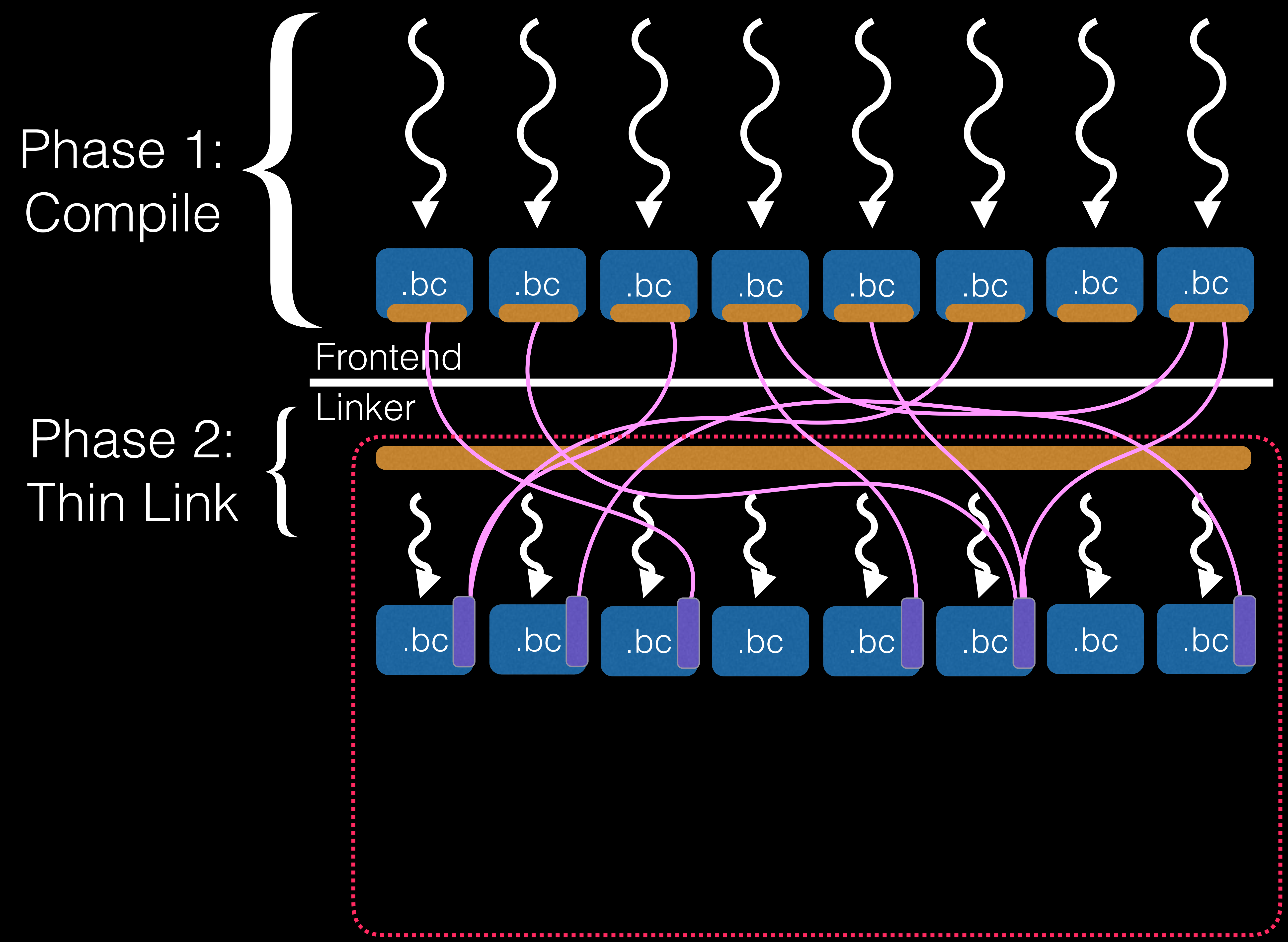
Fully-parallel frontend processing
+ initial optimizations

Extra per-function summary information
are generated *"on the side"*

Link only the summary info
in a giant index: *thin-link*.
No need to parse the IR

Fully-parallel cross-module function
Importing based on summary.

ThinLTO: Overview



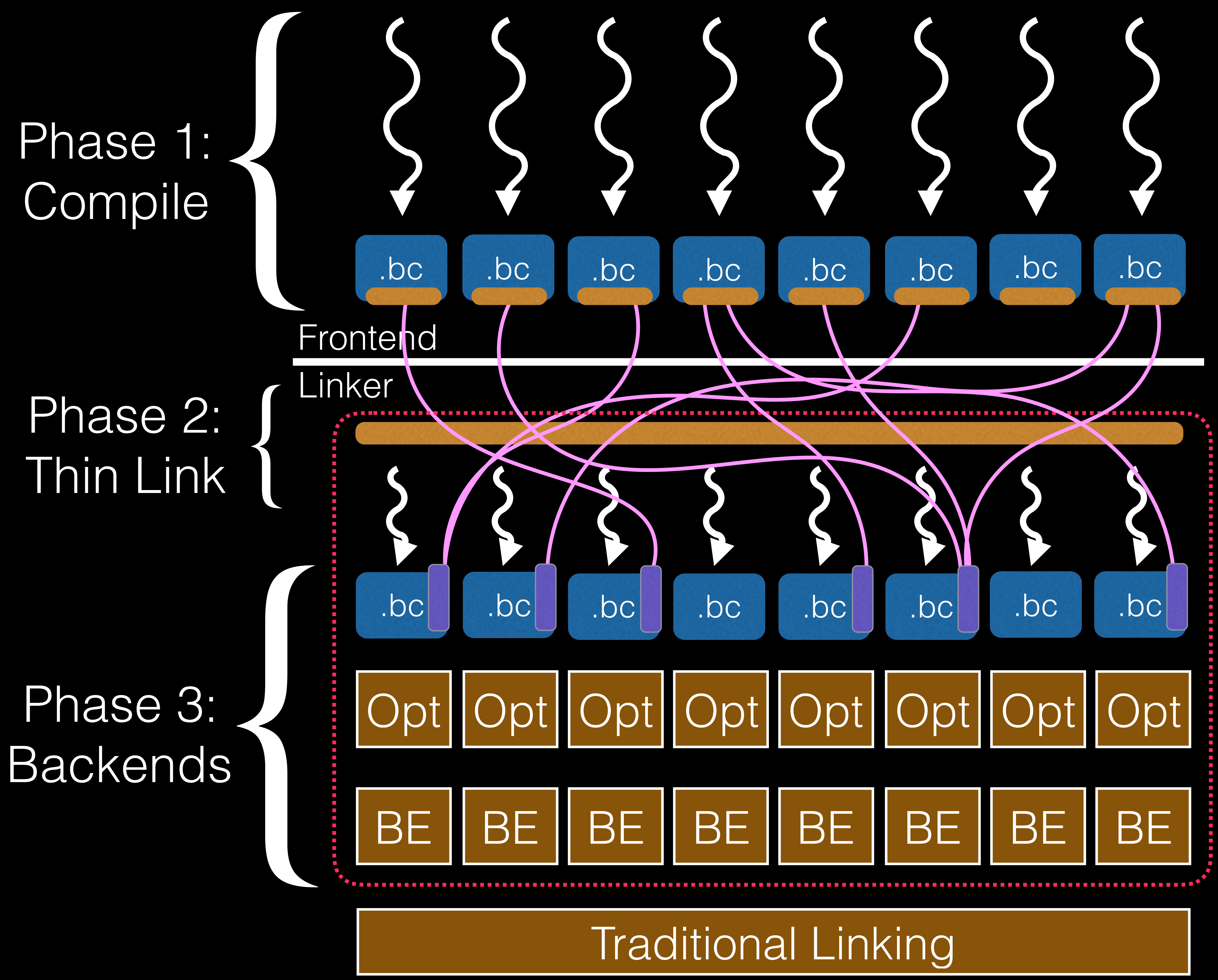
Fully-parallel frontend processing
+ initial optimizations

Extra per-function summary information
are generated *"on the side"*

Link only the summary info
in a giant index: *thin-link*.
No need to parse the IR

Fully-parallel cross-module function
Importing based on summary.
Imported functions are dropped after inlining.

ThinLTO: Overview



Fully-parallel frontend processing
+ initial optimizations

Extra per-function summary information
are generated *"on the side"*

Link only the summary info
in a giant index: *thin-link*.
No need to parse the IR

Fully-parallel cross-module function
Importing based on summary.
Imported functions are dropped after inlining.

Fully-parallel (very boring) usual
optimizations and CodeGen

ThinLTO Model: Summary Generation

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

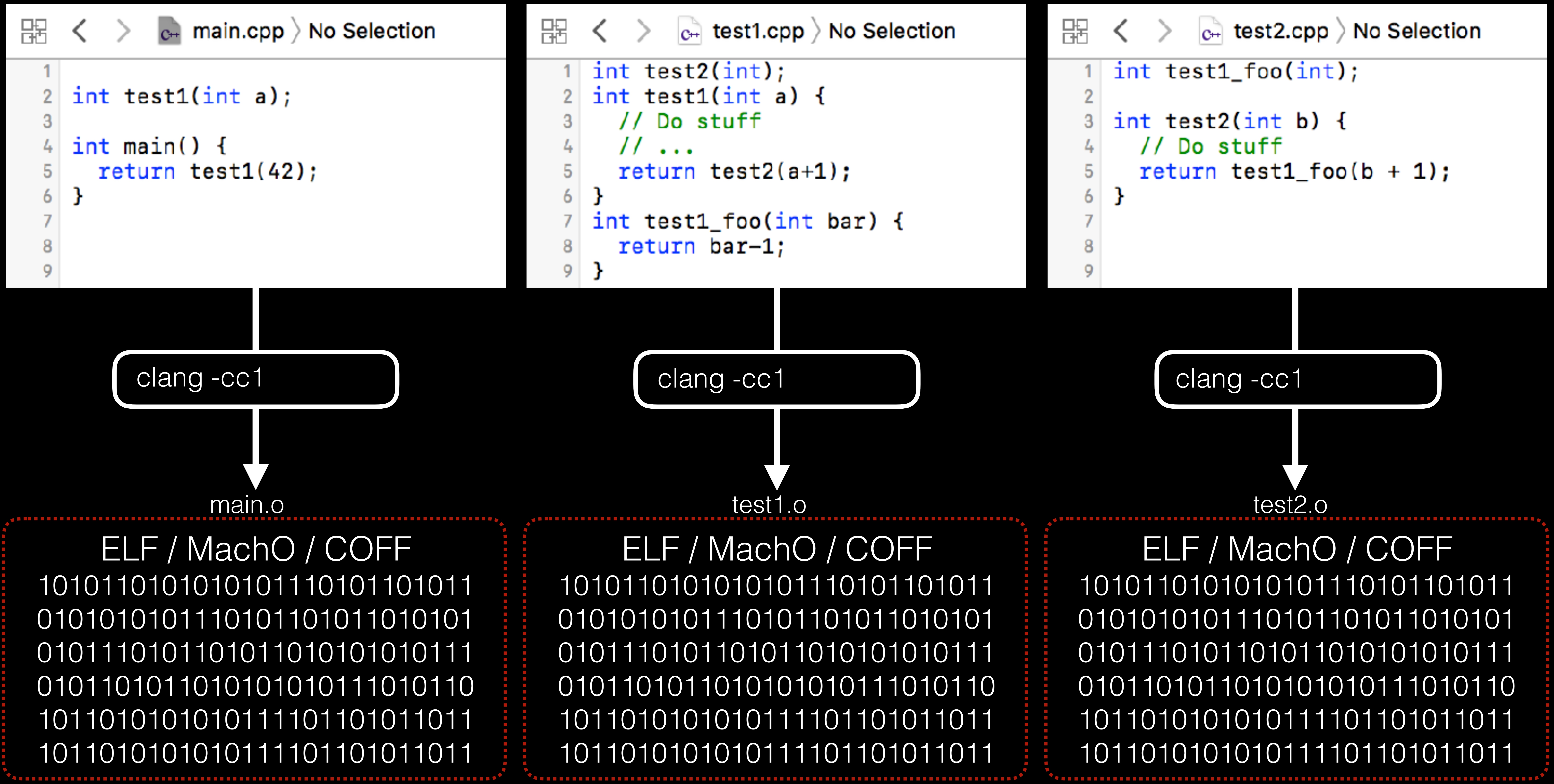
test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

ThinLTO Model: Summary Generation



ThinLTO Model: Summary Generation

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto

clang -cc1 -flto

clang -cc1 -flto

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

ThinLTO Model: Summary Generation

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

ThinLTO Model: Summary Generation

main.cpp 〉 No Selection

```
int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Summaries can contain *profile data* (PGO) data as well, or be extended with other attributes

ThinLTO Model: Summary Generation

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

clang -cc1 -flto=**thin**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

clang -cc1 -flto=**thin**

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Summaries can contain *profile data* (PGO) as well, or be extended with other attributes

ThinLTO Model: Summary Generation

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

ThinLTO Model: Thin-Link Phase

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o

ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o
→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Combined Index

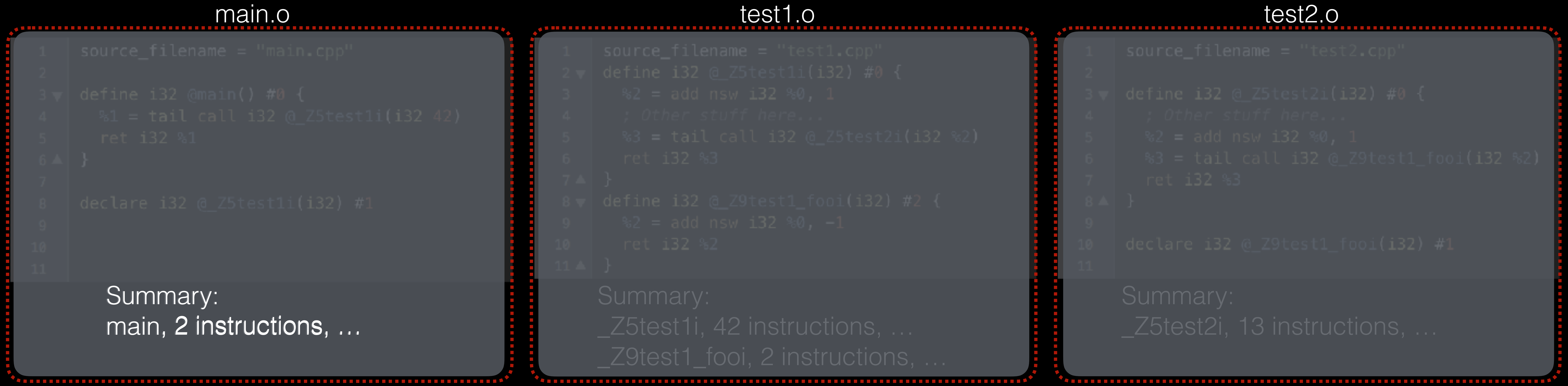
Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o
→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin



Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)

Combined Index

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)

Combined Index

main →

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)

Combined Index

main → main.o , 2 instructions, …

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Combined Index

7c897c21 → main.o , 2 instructions, …

Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)
- Original function names are lost!

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Combined Index

7c897c21 → main.o , 2 instructions, …

Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)
- Original function names are lost!

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o
→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Combined Index

7c897c21	→	main.o	, 2 instructions, …
3597eb0	→	test1.o	, 42 instructions, …
768595e	→	test1.o	, 2 instructions, …

Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)
- Original function names are lost!

ThinLTO Model: Thin-Link Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Combined Index

7c897c21 →	main.o	, 2 instructions, …
3597eb0 →	test1.o	, 42 instructions, …
768595e →	test1.o	, 2 instructions, …
77c4a42 →	test2.o	, 13 instructions, …

Thin-link:

- Produce the combined index
- Sequential step, but **fast** (<1s for clang)
- Original function names are lost!

ThinLTO ***Original*** *Model: Backend Phase*

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

Parallel backend:

- Cross-Module Importing
- Optimization pipeline
- Code generation

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

*ThinLTO **Original** Model: Backend Phase*

clang -o test -flto main.o test1.o test2.o

ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Combined Index

- 7c897c21 → main.o, 2 instructions, …
- 3597eb0 → test1.o, 42 instructions, …
- 768595e → test1.o, 2 instructions, …
- 77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** *Model: Backend Phase*

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

call i32 @_Z5test1i(i32 42) → 3597eb0

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

*ThinLTO **Original** Model: Backend Phase*

clang -o test -flto main.o test1.o test2.o

ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

→ 3597eb0

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

→ 3597eb0

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** *Model: Backend Phase*

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

→ 3597eb0

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO **Original** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {




}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

*ThinLTO **Original** Model: Backend Phase*

clang -o test -flto main.o test1.o test2.o

ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {




}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
```

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO **Original** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
```

call i32 @_Z5test2i(i32 %2) → 77c4a42

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
```

→ 77c4a42

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
```

→ 77c4a42

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
```

Combined Index

7c897c21	→	main.o, 2 instructions, …
3597eb0	→	test1.o, 42 instructions, …
768595e	→	test1.o, 2 instructions, …
77c4a42	→	test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1

```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {




}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {




}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO **Original** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)   → 768595e
  ret i32 %3
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO ***Original*** Model: Backend Phase

clang -o test -flto main.o test1.o test2.o

→ ld […] -o test main.o test1.o test2.o

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions, …

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Combined Index

7c897c21 → main.o, 2 instructions, …

3597eb0 → test1.o, 42 instructions, …

768595e → test1.o, 2 instructions, …

77c4a42 → test2.o, 13 instructions, …

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel Incremental

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel	Incremental	Memory lean

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel | Incremental | Memory lean

- January: link time on single machine was *terrible:*

→ cross-module importing alone >3x the total time of Monolithic LTO

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel | Incremental | Memory lean

- January: link time on single machine was *terrible:*

  ➔ cross-module importing alone >3x the total time of Monolithic LTO

  ➔ linking *opt:* 1036 inputs files, but >30000 IR loads

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel | Incremental | Memory lean

- January: link time on single machine was *terrible:*

➔ cross-module importing alone >3x the total time of Monolithic LTO

➔ linking *opt:* 1036 inputs files, but >30000 IR loads

➔ Loading IR is slow! Loading IR with debug-info is **insanely** slow!

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel | Incremental | Memory lean

- January: link time on single machine was *terrible:*

➞ cross-module importing alone >3x the total time of Monolithic LTO

➞ linking *opt:* 1036 inputs files, but >30000 IR loads

➞ Loading IR is slow! Loading IR with debug-info is **insanely** slow!

"none of the subsystems in LLVM are really good until they have been rewritten at least once."

➞ This is what we did between January and March!

ThinLTO

Euro-LLVM 2015: A Fine-Grained Demand-Driven Infrastructure

- December 2015: clang bootstrap on MacOS \o/

Parallel | Incremental | Memory lean

- January: link time on single machine was *terrible:*

➞ cross-module importing alone >3x the total time of Monolithic LTO

➞ linking *opt:* 1036 inputs files, but >30000 IR loads

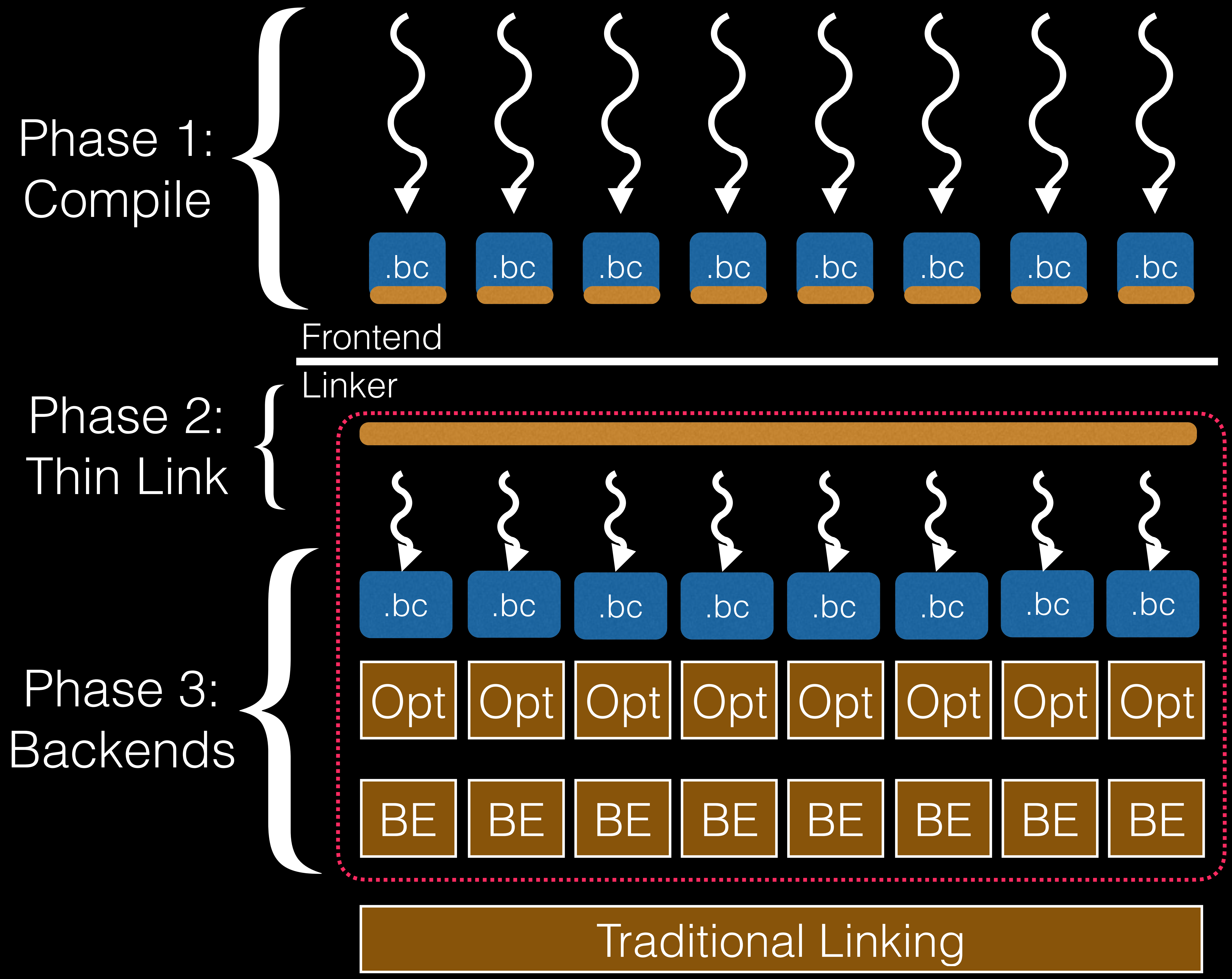
➞ Loading IR is slow! Loading IR with debug-info is **insanely** slow!

"none of the subsystems in LLVM are really good until they have been rewritten at least once."

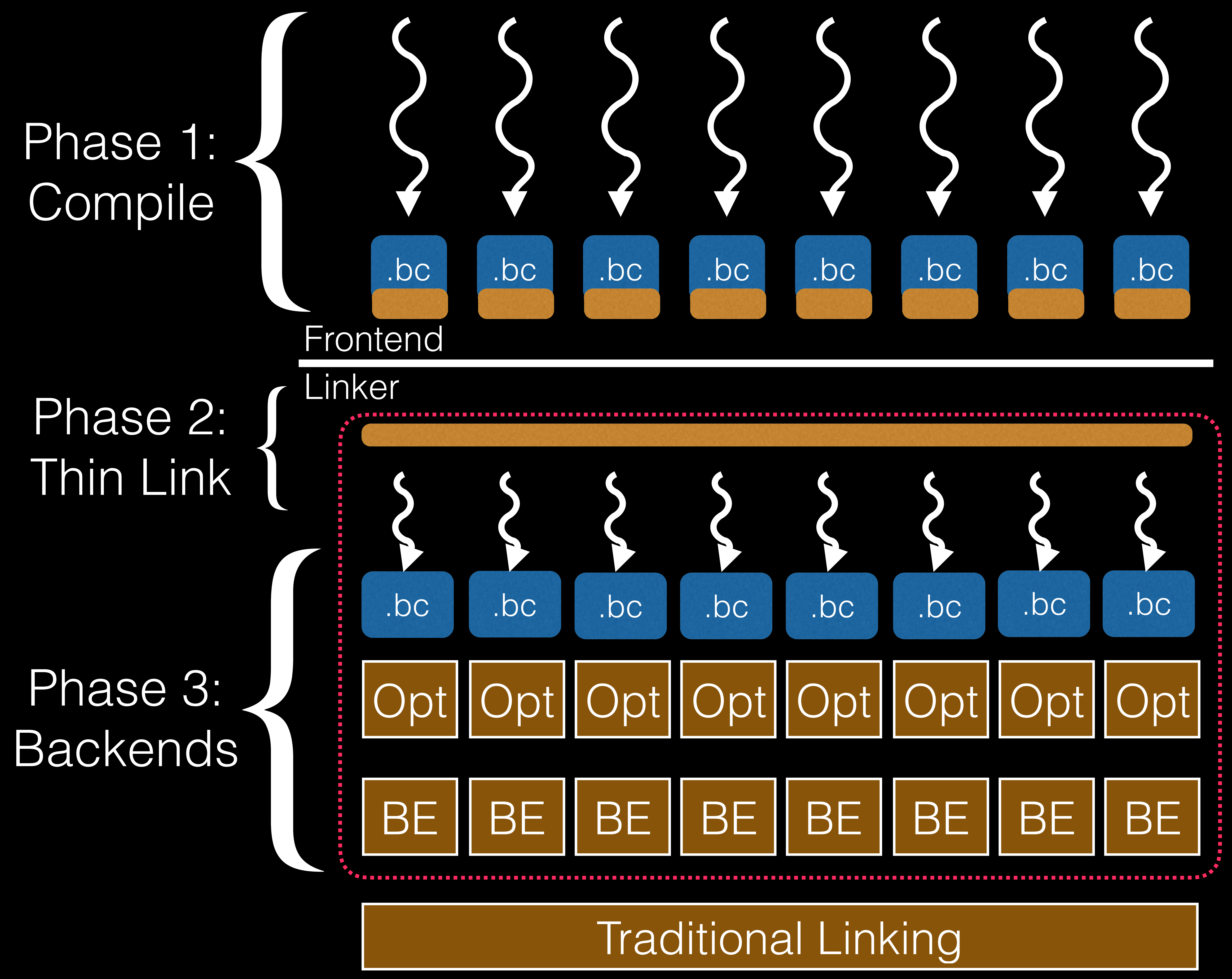
(Chris Lattner, The Architecture of Open Source Applications: LLVM, 2011)

➞ This is what we did between January and March!

ThinLTO: Overview

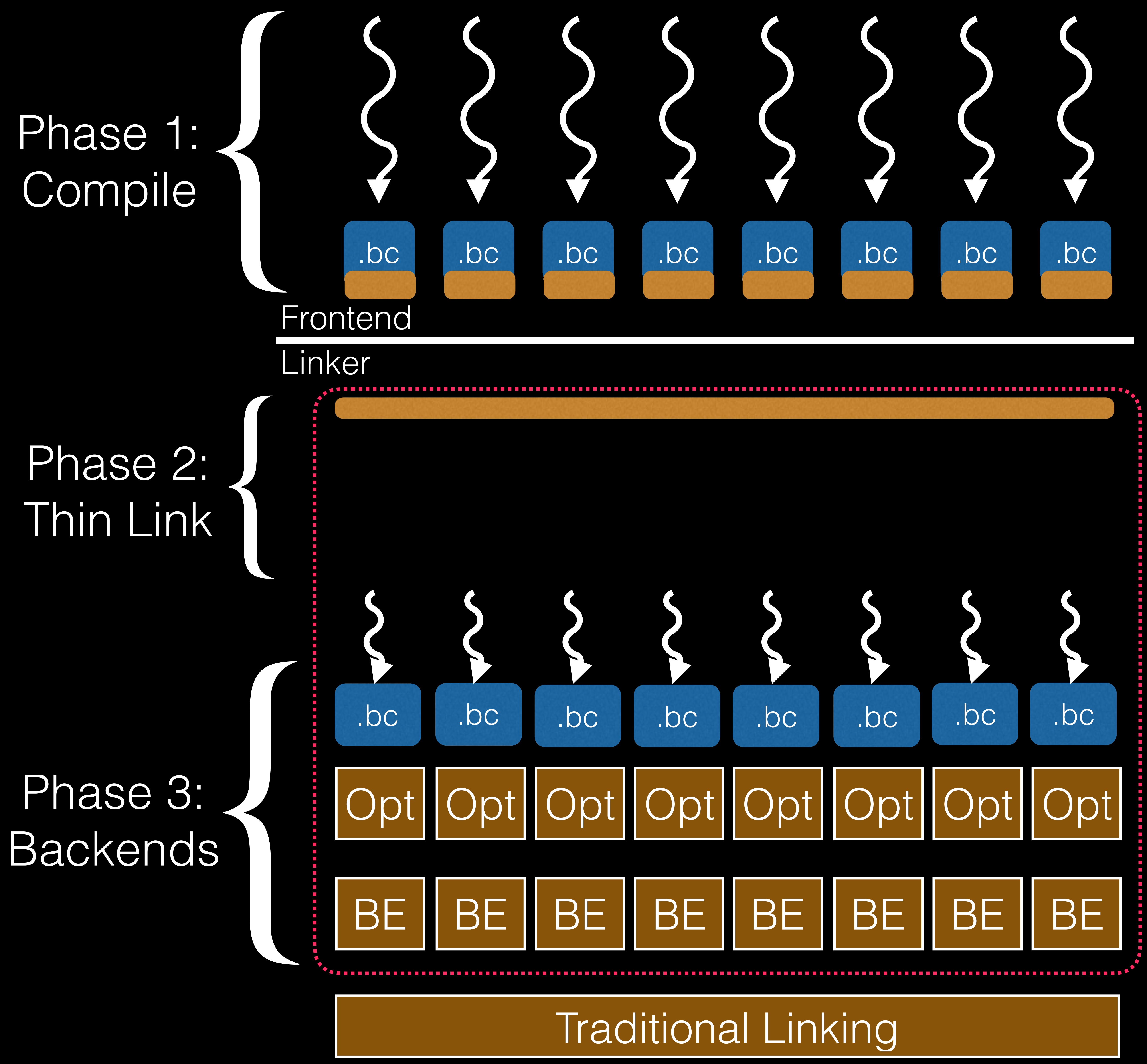


ThinLTO: Overview



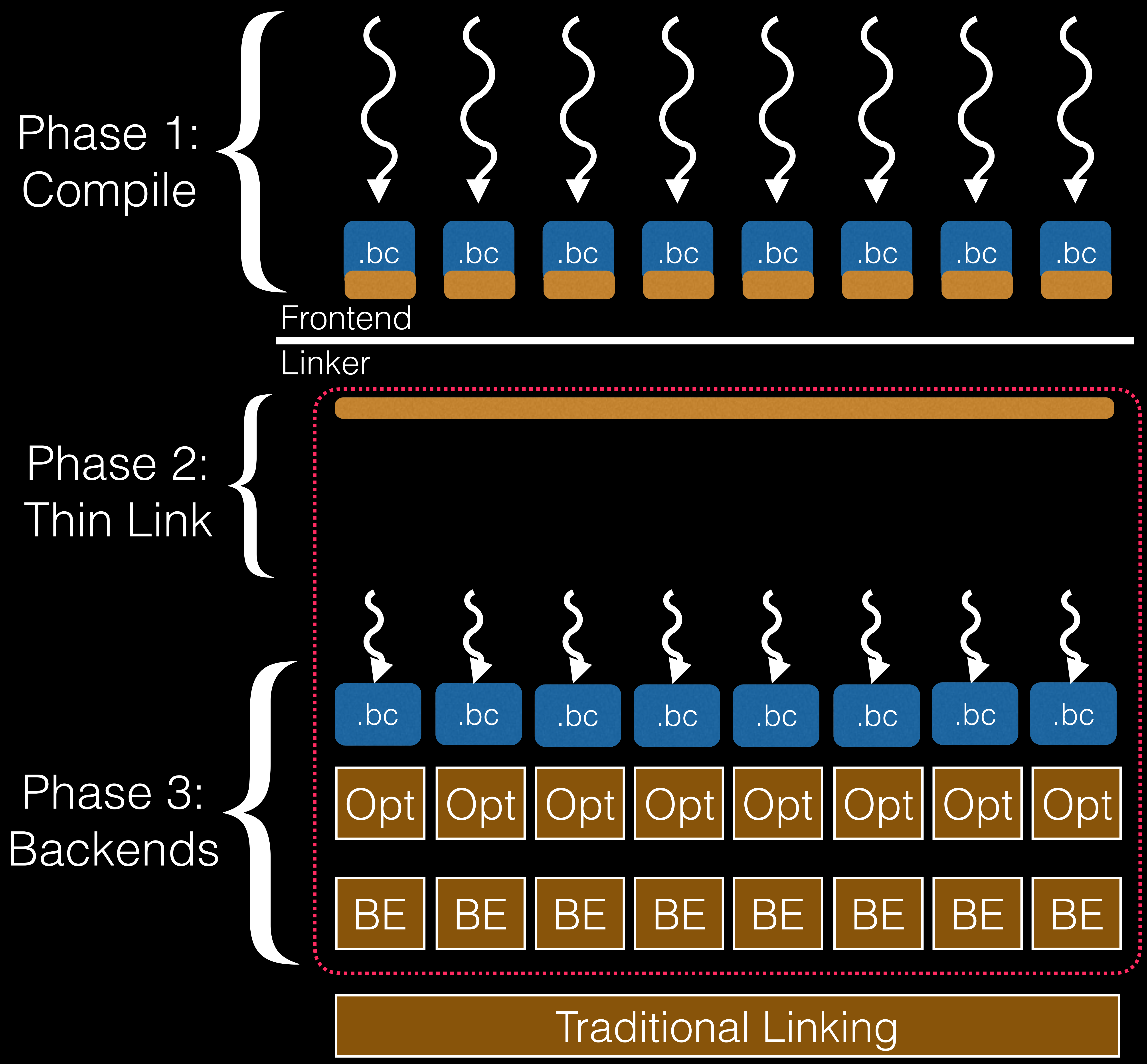
+ Local reference/call graph
+ Module Hash (for incremental build)

ThinLTO: Overview



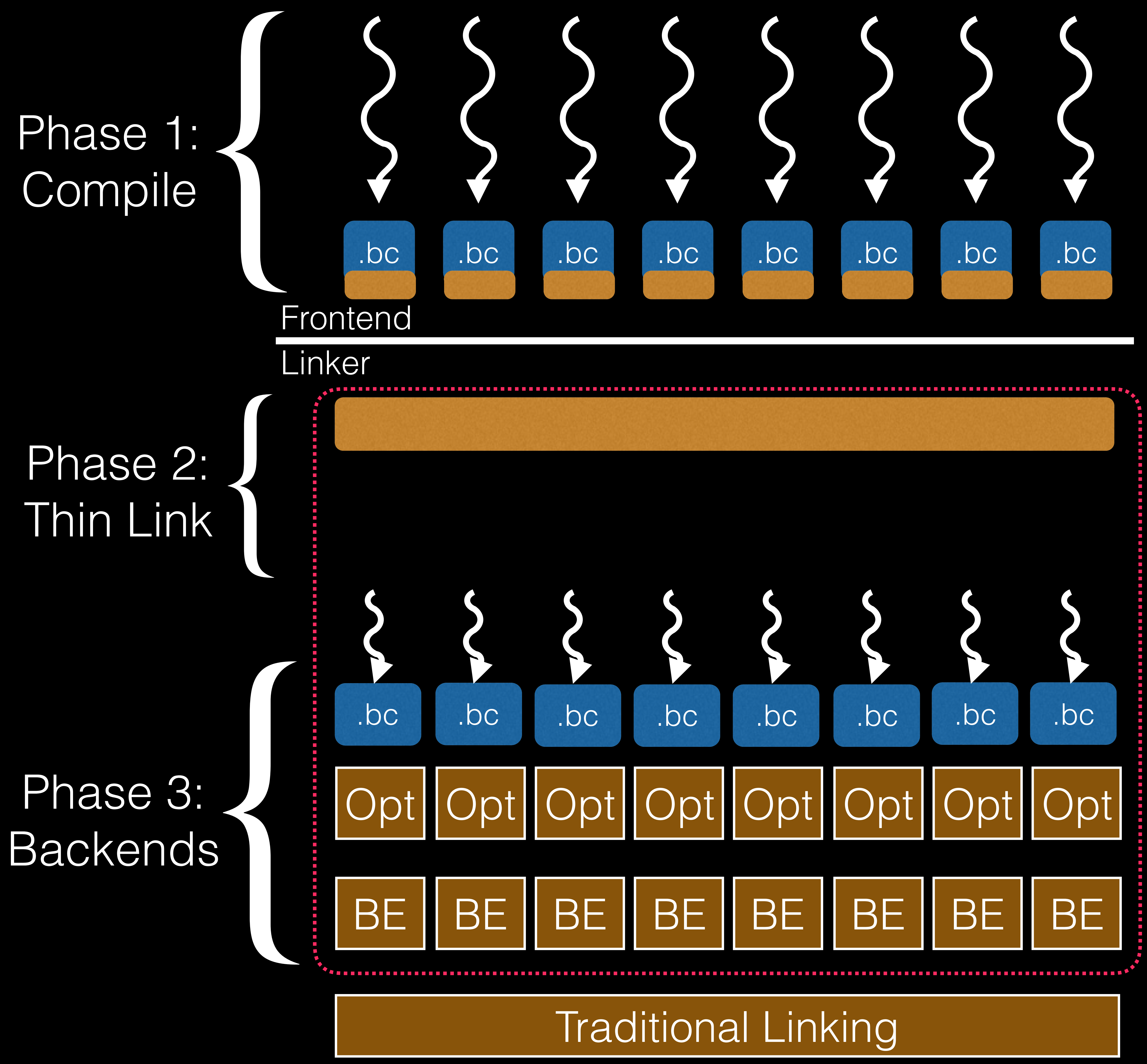
+ Local reference/call graph
+ Module Hash (for incremental build)

ThinLTO: Overview



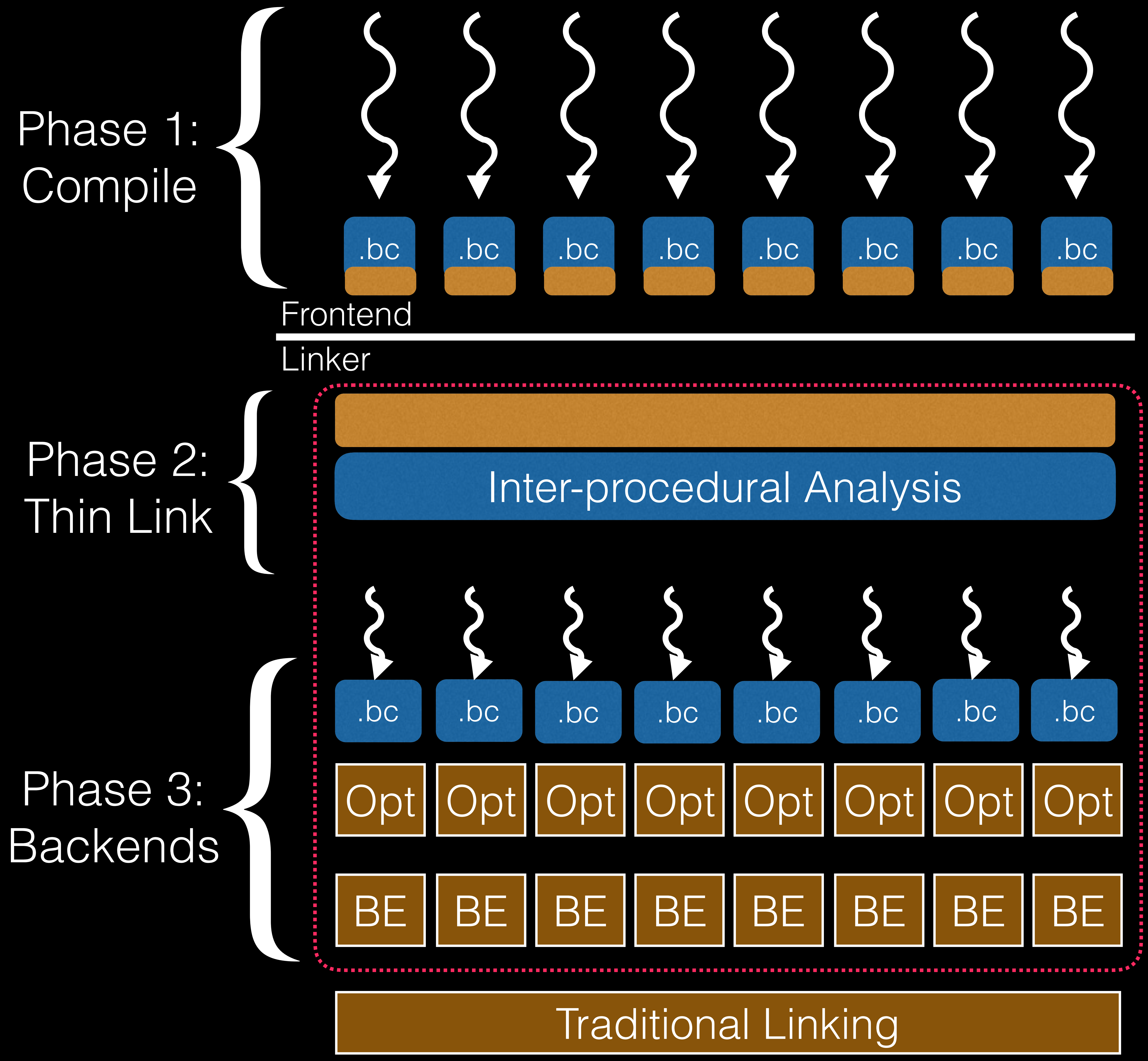
+ Local reference/call graph
+ Module Hash (for incremental build)

ThinLTO: Overview



+ Local reference/call graph
+ Module Hash (for incremental build)
+ Full reference/call graph

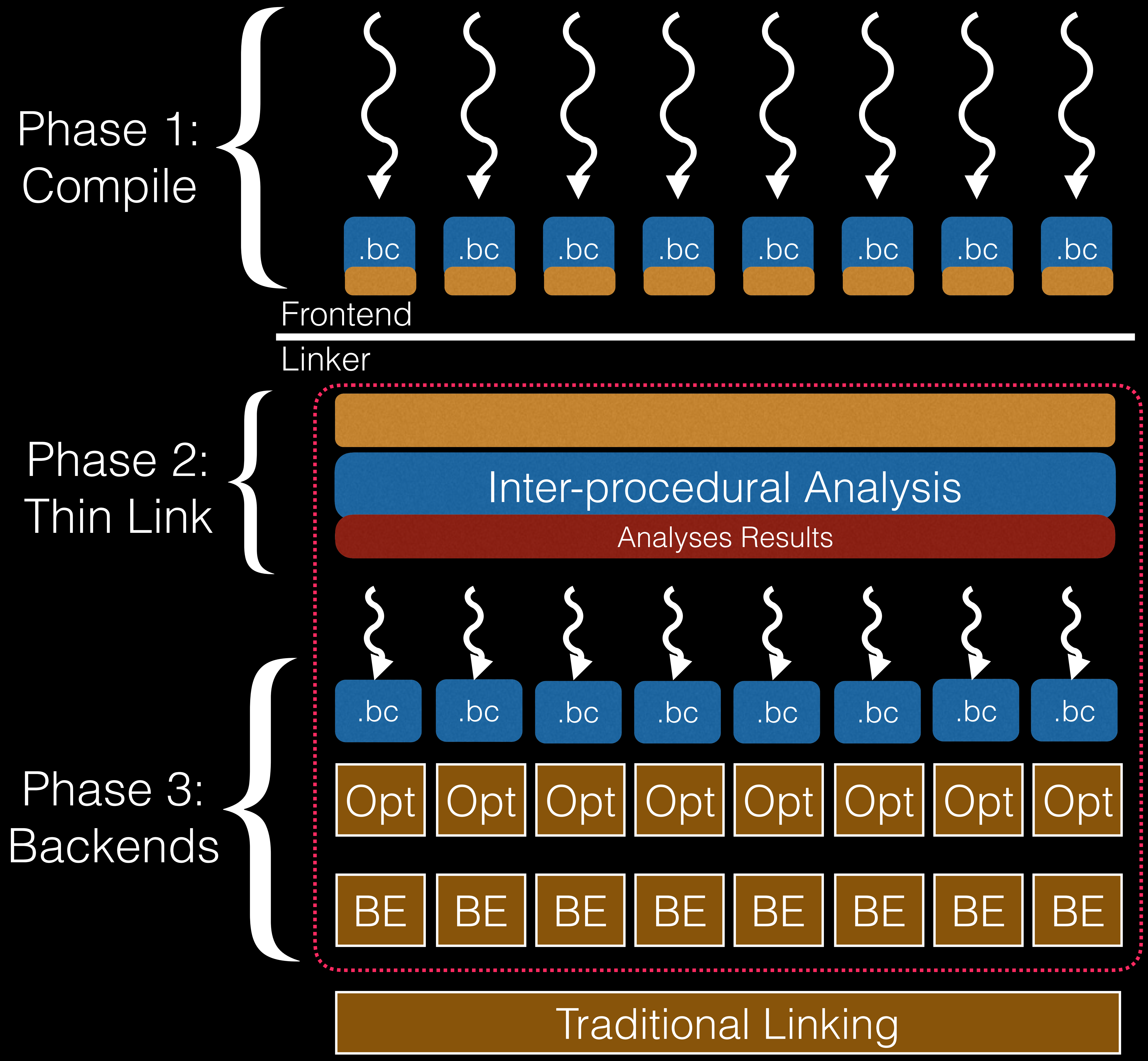
ThinLTO: Overview



+ Local reference/call graph
+ Module Hash (for incremental build)

+ Full reference/call graph

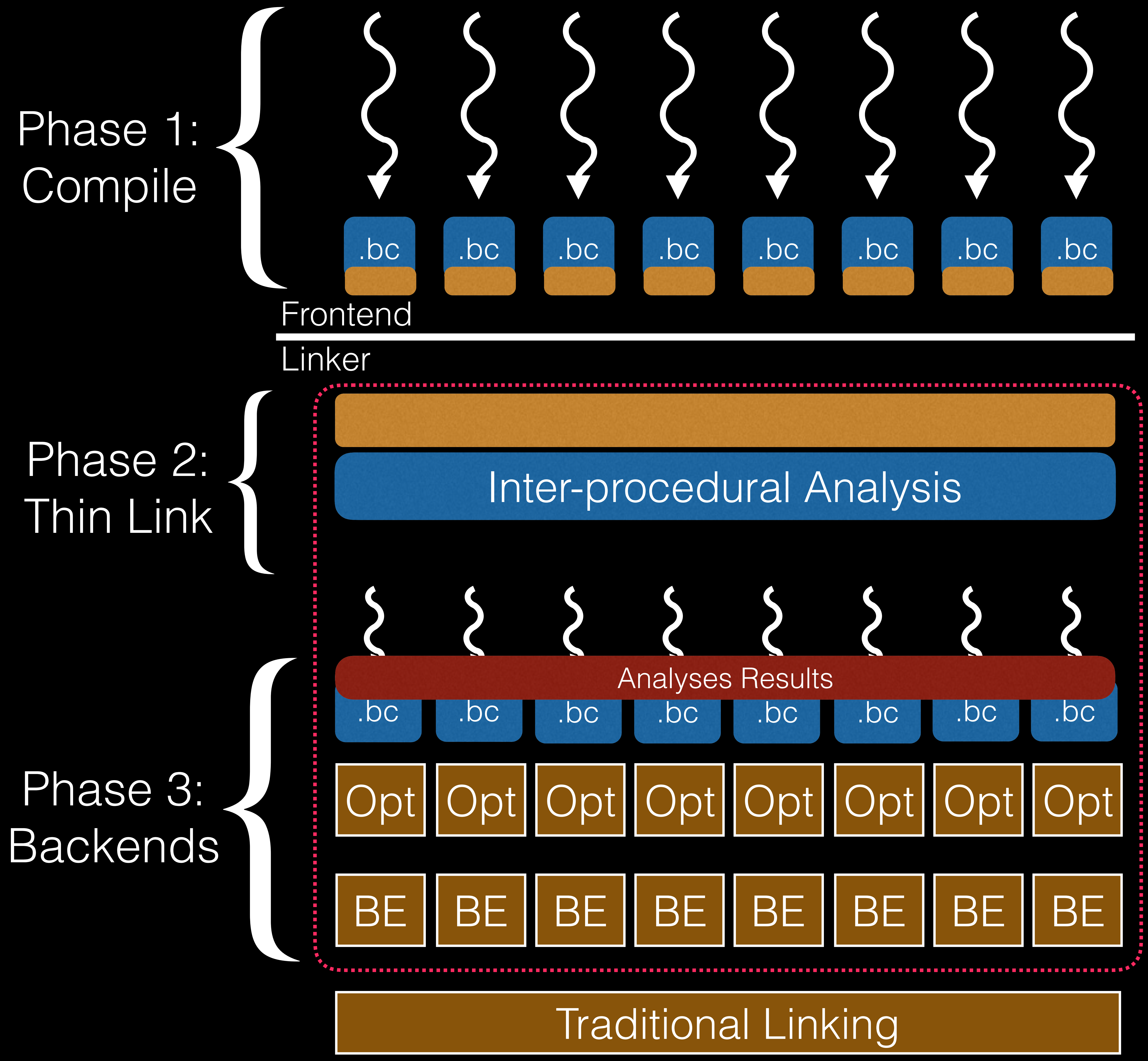
ThinLTO: Overview



+ Local reference/call graph
+ Module Hash (for incremental build)

+ Full reference/call graph

ThinLTO: Overview



+ Local reference/call graph
+ Module Hash (for incremental build)
+ Full reference/call graph

+ Parallel Inter-procedural transformation
based on the analyses results

ThinLTO Revisited: Compile Phase

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

clang -cc1 -flto=**thin**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

clang -cc1 -flto=**thin**

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions

_Z9test1_fooi, 2 instructions

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions

ThinLTO Revisited: Compile Phase

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

ThinLTO Revisited: Compile Phase

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

ThinLTO Revisited: Compile Phase

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

ThinLTO Revisited: Thin Link

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

ThinLTO Revisited: Thin Link

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Thin-link:

- Produce the combined index
- Sequential step, but fast (<2s for clang)
- Original function names are lost!

Combined Index

7c897c21 → main.o, 2 instructions

3597eb0 → test1.o, 42 instructions

768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions

LLVM Linker Plugin

ThinLTO Revisited: Thin Link

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Thin-link:

- Produce the combined index
- Sequential step, but fast (<2s for clang)
- Original function names are lost!

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0, global linkage
3597eb0 → test1.o, 42 instructions
call 77c4a42, global linkage
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e, global linkage

LLVM Linker Plugin

ThinLTO Revisited: Thin Link

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Thin-link:

- Produce the combined index
- Sequential step, but fast (<2s for clang)
- Original function names are lost!

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0, global linkage
3597eb0 → test1.o, 42 instructions
call 77c4a42, global linkage
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e, global linkage

LLVM Linker Plugin

ThinLTO Revisited: Thin Link

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Thin-link:

- Produce the combined index
- Sequential step, but fast (<2s for clang)
- Original function names are lost!
- Run inter-procedural analyses:

→ Transformations: Global DCE, Linkage, …

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0, global linkage
3597eb0 → test1.o, 42 instructions
call 77c4a42, global linkage
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e, global linkage

LLVM Linker Plugin

ThinLTO Revisited: Thin Link

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Thin-link:

- Produce the combined index
- Sequential step, but fast (<2s for clang)
- Original function names are lost!
- Run inter-procedural analyses:

  → Transformations: Global DCE, Linkage, …

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0, global linkage
3597eb0 → test1.o, 42 instructions
call 77c4a42, global linkage
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e, global linkage

No need to load any IR! This is still a very fast serial phase.

LLVM Linker Plugin

ThinLTO Revisited: Thin Link

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

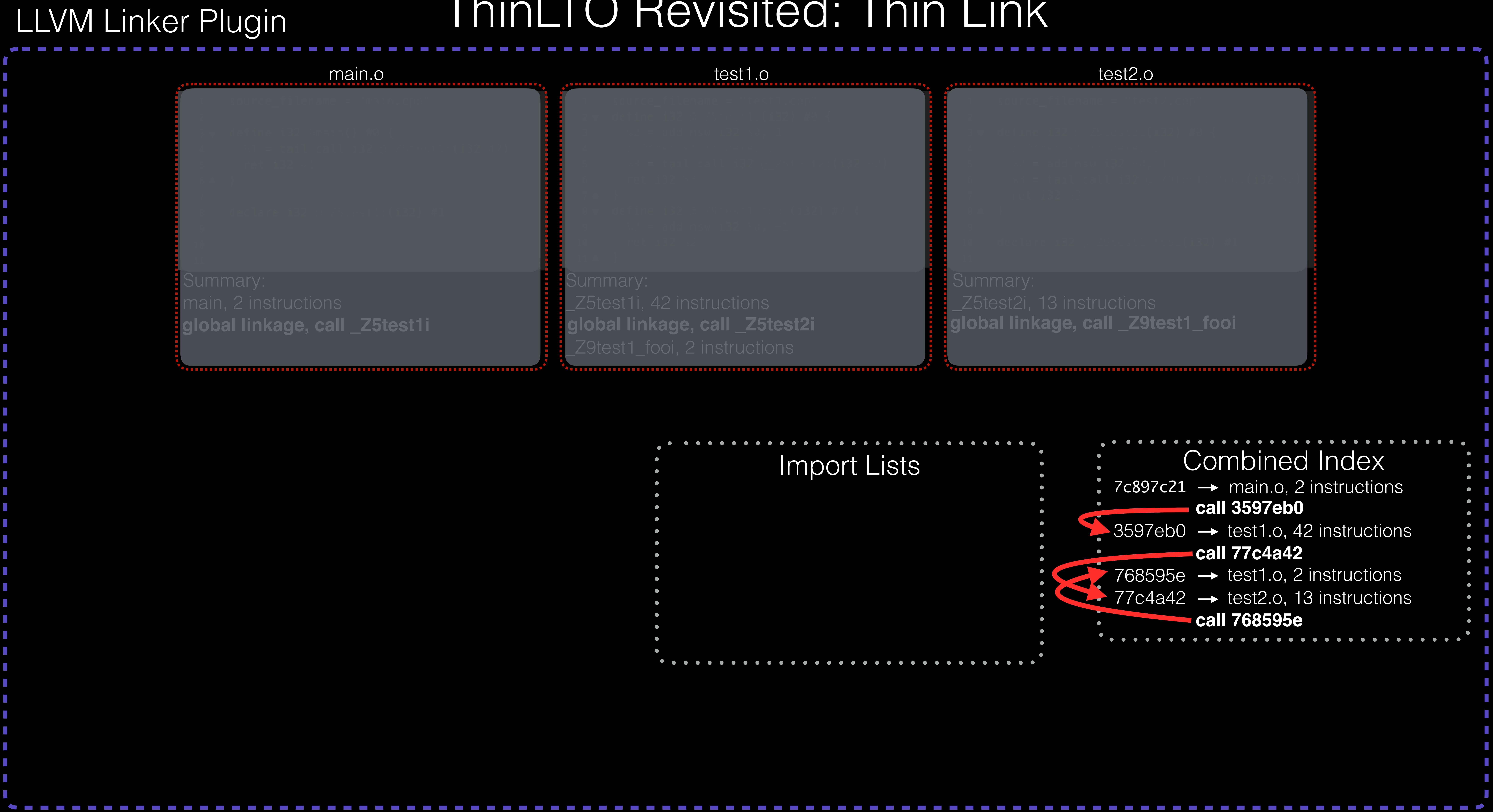
test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

ThinLTO Revisited: Thin Link



LLVM Linker Plugin

ThinLTO Revisited: Thin Link

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Import Lists

main.o
3597eb0
77c4a42
768595e

Combined Index

7c897c21 ⟶ main.o, 2 instructions
call 3597eb0
3597eb0 ⟶ test1.o, 42 instructions
call 77c4a42
768595e ⟶ test1.o, 2 instructions
77c4a42 ⟶ test2.o, 13 instructions
call 768595e

ThinLTO Revisited: Thin Link

LLVM Linker Plugin

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Import Lists

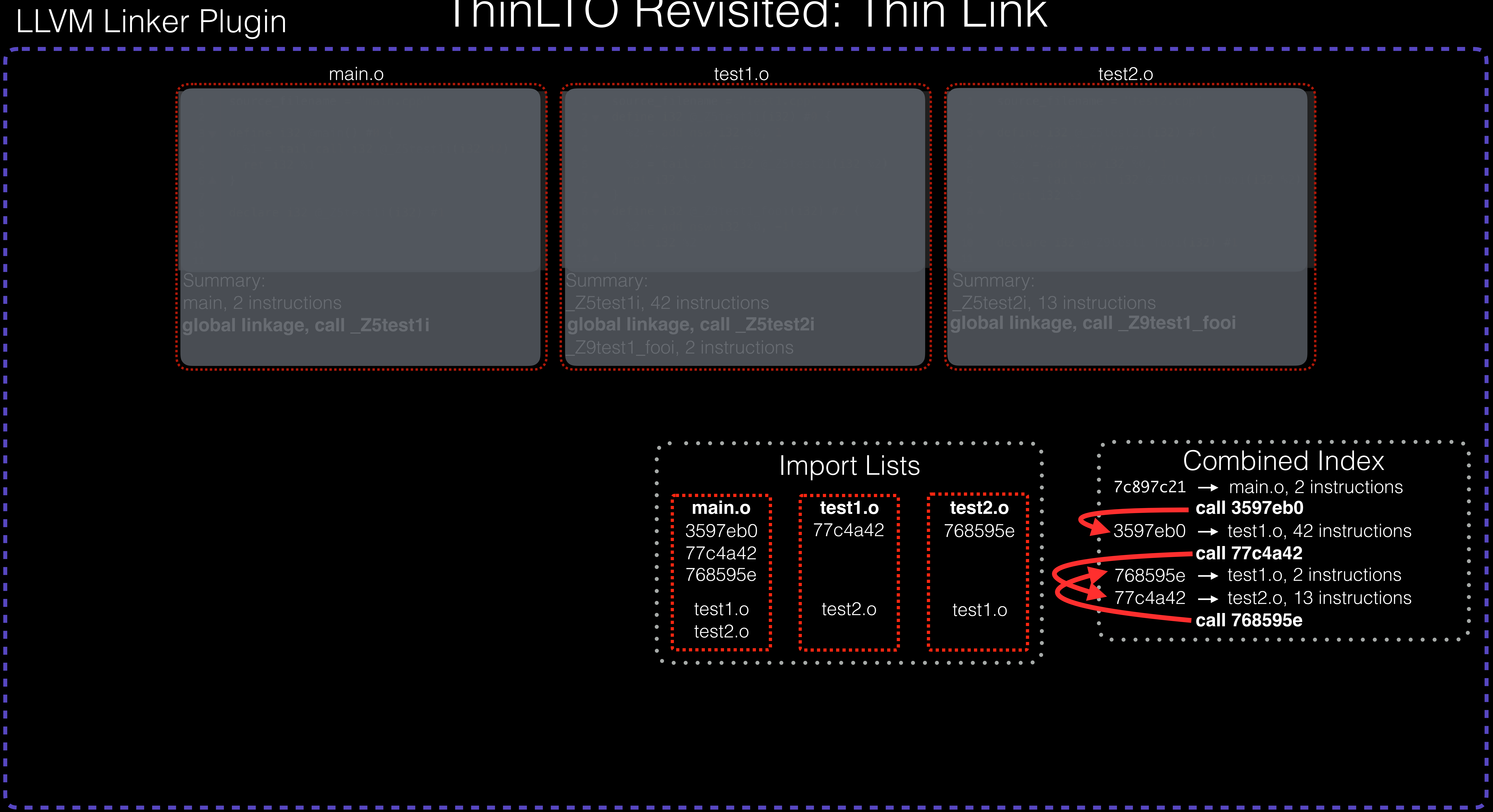
main.o
3597eb0
77c4a42
768595e

test1.o
test2.o

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

ThinLTO Revisited: Thin Link



ThinLTO Revisited: Thin Link

LLVM Linker Plugin

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Import Lists

main.o	**test1.o**	**test2.o**
3597eb0 77c4a42 768595e	77c4a42	768595e
test1.o test2.o	test2.o	test1.o

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

ThinLTO Revisited: Thin Link

LLVM Linker Plugin

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Import Lists

main.o	**test1.o**	**test2.o**
3597eb0 77c4a42 768595e	77c4a42	768595e
test1.o test2.o	test2.o	test1.o

ThinLTO Revisited: Parallel Backends

LLVM Linker Plugin

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Import Lists

main.o
3597eb0
77c4a42
768595e

test1.o
test2.o

test1.o
77c4a42

test2.o

test2.o
768595e

test1.o

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Parallel ThinLTO Backends:

- Read the import lists

Import Lists

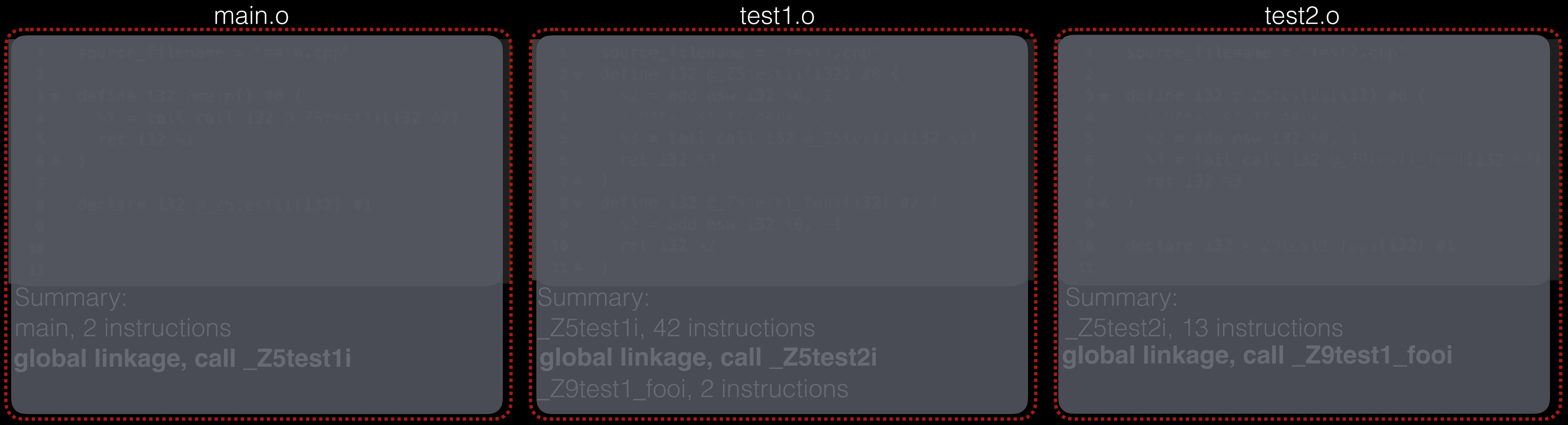
main.o
3597eb0
77c4a42
768595e
test1.o
test2.o

test1.o
77c4a42
test2.o

test2.o
768595e
test1.o

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends



Parallel ThinLTO Backends:

- Read the import lists
- Process all imports from one module after the other

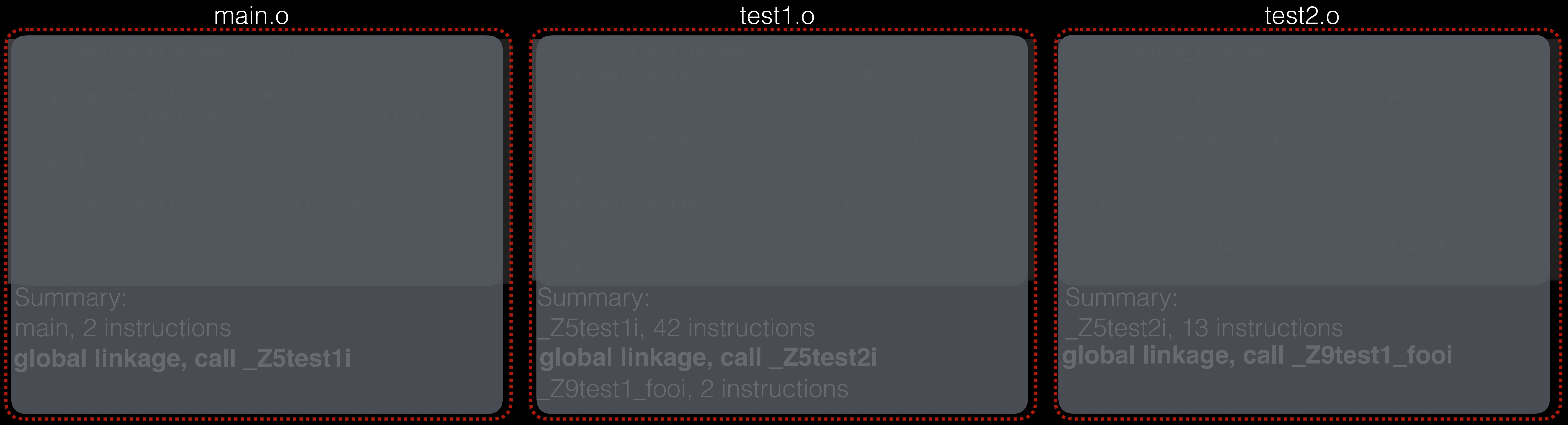
Import Lists

main.o
3597eb0
77c4a42
768595e
test1.o
test2.o

test1.o
77c4a42
test2.o

test2.o
768595e
test1.o

ThinLTO Revisited: Parallel Backends



Parallel ThinLTO Backends:

- Read the import lists
- Process all imports from one module after the other

➔ Knowing upfront what to import allows to limit the number of IR loading

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

ThinLTO Revisited: Parallel Backends

LLVM Linker Plugin

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

2) Inlining

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

2) Inlining
3) “Simplification” passes

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

4) Inlining

2) Inlining
3) “Simplification” passes

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

4) Inlining
5) “Simplification” passes

2) Inlining
3) “Simplification” passes

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

ThinLTO Revisited: Parallel Backends

LLVM Linker Plugin

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

6) Inlining

4) Inlining
5) “Simplification” passes

2) Inlining
3) “Simplification” passes

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

ThinLTO Revisited: Parallel Backends

LLVM Linker Plugin

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

6) Inlining
7) “Simplification” passes

4) Inlining
5) “Simplification” passes

2) Inlining
3) “Simplification” passes

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

6) Inlining
7) “Simplification” passes

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}
```

8) “Eliminate Available Externally” Pass

4) Inlining
5) “Simplification” passes

2) Inlining
3) “Simplification” passes

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

6) Inlining
7) "Simplification" passes

4) Inlining
5) "Simplification" passes

2) Inlining
3) "Simplification" passes

1) "Simplification" passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}
```

8) "Eliminate Available Externally" Pass

9) "Optimization Passes": 43 passes including vectorization

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

define available_externally i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
define available_externally i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

CGSCC Simplification and Inlining

6) Inlining
7) “Simplification” passes

4) Inlining
5) “Simplification” passes

2) Inlining
3) “Simplification” passes

1) “Simplification” passes: 47 passes including SROA - JumpThreading - InstCombine - LICM - … (and many others*)

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}
```

10) Codegen

main.o

8) “Eliminate Available Externally” Pass

9) “Optimization Passes”: 43 passes including vectorization

*See *PassManagerBuilder::addFunctionSimplificationPasses()*

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}


define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
```

main.o

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}


define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
```

CGSCC Simplification and Inlining

4) Inlining
5) “Simplification” passes

1) “Simplification” passes

2) Inlining
3) “Simplification” passes

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

6) “Eliminate Available Externally” Pass

7) “Optimization Passes”: 43 passes including vectorization

8) Codegen

main.o

test1.o

LLVM Linker Plugin

ThinLTO Revisited: Parallel Backends

main.o

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}


define available_externally i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}
```

CGSCC Simplification and Inlining

4) Inlining
5) “Simplification” passes

1) “Simplification” passes

2) Inlining
3) “Simplification” passes

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

8) Codegen

main.o

test1.o

6) “Eliminate Available Externally” Pass

7) “Optimization Passes”: 43 passes including vectorization

Parallelism is obtained thanks to redundant optimizations over the same IR!

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉No Selection

```cpp
int test1_foo(int);



int test2(int b) {
  // Do stuff

  return test1_foo(b + 1);
}
```

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);
static void bar() {
  // Important stuff here...
}
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);
static void bar() {
  // Important stuff here...
}
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
__attribute__((available_externally))
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);
static void bar() {
  // Important stuff here...
}
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  bar();
  return test1_foo(a + 1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);
static void bar() {
  // Important stuff here...
}
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  bar();
  return test1_foo(a + 1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);
static void bar() {
  // Important stuff here...
}
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  bar();
  return test1_foo(a + 1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

Introduction of an external reference (due to importing)!

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉 No Selection

```
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```
int test1_foo(int);
void bar.llvm.77c4a42() {
  // Important stuff here...
}
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

test1.cpp 〉 No Selection

```
int test2(int);
int test1(int a) {
  // Do stuff
  bar.llvm.77c4a42();
  return test1_foo(a + 1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

Introduction of an external reference (due to importing)!

→ requires promotion/rename of local linkage functions

Thin-Link IPA: Correctness
Global Scope Promotion

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);
void bar.llvm.77c4a42() {
  // Important stuff here...
}
int test2(int b) {
  // Do stuff
  bar();
  return test1_foo(b + 1);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  bar.llvm.77c4a42();
  return test1_foo(a + 1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

Introduction of an external reference (due to importing)!

- ➔ requires promotion/rename of local linkage functions
- ➔ similar technique for exported “discardable” linkage references

Thin-Link IPA: Compile Time Optimization
Weak Linkage Resolution

vector1.cpp 〉 No Selection

```cpp
#include <vector>

void foo(std::vector<int> &V) {
  V.push_back(1);
  V.push_back(2);
  V.push_back(3);
  V.push_back(4);
}
```

vector2.cpp 〉 No Selection

```cpp
#include <vector>

void bar(std::vector<int> &V) {
  V.push_back(11);
  V.push_back(22);
  V.push_back(33);
  V.push_back(44);
}
```

Thin-Link IPA: Compile Time Optimization
Weak Linkage Resolution

vector1.cpp 〉 No Selection

```cpp
#include <vector>

void foo(std::vector<int> &V) {
  V.push_back(1);
  V.push_back(2);
  V.push_back(3);
  V.push_back(4);
}
```

vector2.cpp 〉 No Selection

```cpp
#include <vector>

void bar(std::vector<int> &V) {
  V.push_back(11);
  V.push_back(22);
  V.push_back(33);
  V.push_back(44);
}
```

```llvm
source_filename = "vector2.cpp"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.12.0"

define void @_Z3fooRNSt3__16vectorIiNS_9allocatorIiEEEE(%"class.std::__1::vector"* dereferenceable(24)) #0 {
    ;   Lot
    ;   of
    ; stuff
  ret void
}

define linkonce_odr void @_ZNSt3__16vectorIiNS_9allocatorIiEEE21__push_back_slow_pathIiEEvOT_(%"class.std::__1::vector"*,
        i32* nocapture readonly dereferenceable(4)) #0 align 2 personality i8* bitcast (i32 (...)* @__gxx_personality_v0 to i8*) {
  %3 = getelementptr inbounds %"class.std::__1::vector", %"class.std::__1::vector"* %0, i64 0, i32 0, i32 1
  %4 = bitcast i32** %3 to i64*
  %5 = load i64, i64* %4, align 8, !tbaa !6
  %6 = bitcast %"class.std::__1::vector"* %0 to i64*
  %7 = load i64, i64* %6, align 8, !tbaa !11
  %8 = sub i64 %5, %7
  %9 = ashr exact i64 %8, 2
  %10 = add nsw i64 %9, 1
  %11 = icmp ugt i64 %10, 4611686018427387903
  br i1 %11, label %12, label %15

; <label>:12                                        ; preds = %2
```

… ~100 instructions

C++ template generates a lot of redundant code!

Thin-Link IPA: Compile Time Optimization
Weak Linkage Resolution

vector1.cpp 〉 No Selection

```cpp
#include <vector>

void foo(std::vector<int> &V) {
  V.push_back(1);
  V.push_back(2);
  V.push_back(3);
  V.push_back(4);
}
```

vector2.cpp 〉 No Selection

```cpp
#include <vector>

void bar(std::vector<int> &V) {
  V.push_back(11);
  V.push_back(22);
  V.push_back(33);
  V.push_back(44);
}
```

```llvm
source_filename = "vector2.cpp"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.12.0"

define void @_Z3fooRNSt3__16vectorIiNS_9allocatorIiEEEE(%"class.std::__1::vector"* dereferenceable(24)) #0 {
    ;   Lot
    ;   of
    ; stuff
  ret void
}

define linkonce_odr void @_ZNSt3__16vectorIiNS_9allocatorIiEEE21__push_back_slow_pathIiEEvOT_(%"class.std::__1::vector"*,
        i32* nocapture readonly dereferenceable(4)) #0 align 2 personality i8* bitcast (i32 (...)* @__gxx_personality_v0 to i8*) {
  %3 = getelementptr inbounds %"class.std::__1::vector", %"class.std::__1::vector"* %0, i64 0, i32 0, i32 1
  %4 = bitcast i32** %3 to i64*
  %5 = load i64, i64* %4, align 8, !tbaa !6
  %6 = bitcast %"class.std::__1::vector"* %0 to i64*
  %7 = load i64, i64* %6, align 8, !tbaa !11
  %8 = sub i64 %5, %7
  %9 = ashr exact i64 %8, 2
  %10 = add nsw i64 %9, 1
  %11 = icmp ugt i64 %10, 4611686018427387903
  br i1 %11, label %12, label %15

; <label>:12                                          ; preds = %2
```

… ~100 instructions

C++ template generates a lot of redundant code!

- Monolithic LTO will *merge* these and codegen only one naturally

Thin-Link IPA: Compile Time Optimization
Weak Linkage Resolution

vector1.cpp 〉 No Selection

```cpp
#include <vector>

void foo(std::vector<int> &V) {
  V.push_back(1);
  V.push_back(2);
  V.push_back(3);
  V.push_back(4);
}
```

vector2.cpp 〉 No Selection

```cpp
#include <vector>

void bar(std::vector<int> &V) {
  V.push_back(11);
  V.push_back(22);
  V.push_back(33);
  V.push_back(44);
}
```

```llvm
source_filename = "vector2.cpp"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.12.0"

define void @_Z3fooRNSt3__16vectorIiNS_9allocatorIiEEEE(%"class.std::__1::vector"* dereferenceable(24)) #0 {
    ;   Lot
    ;   of
    ; stuff
  ret void
}

define linkonce_odr void @_ZNSt3__16vectorIiNS_9allocatorIiEEE21__push_back_slow_pathIiEEvOT_(%"class.std::__1::vector"*,
        i32* nocapture readonly dereferenceable(4)) #0 align 2 personality i8* bitcast (i32 (...)* @__gxx_personality_v0 to i8*) {
  %3 = getelementptr inbounds %"class.std::__1::vector", %"class.std::__1::vector"* %0, i64 0, i32 0, i32 1
  %4 = bitcast i32** %3 to i64*
  %5 = load i64, i64* %4, align 8, !tbaa !6
  %6 = bitcast %"class.std::__1::vector"* %0 to i64*
  %7 = load i64, i64* %6, align 8, !tbaa !11
  %8 = sub i64 %5, %7
  %9 = ashr exact i64 %8, 2
  %10 = add nsw i64 %9, 1
  %11 = icmp ugt i64 %10, 4611686018427387903
  br i1 %11, label %12, label %15

; <label>:12                                        ; preds = %2
```

… ~100 instructions

C++ template generates a lot of redundant code!

- Monolithic LTO will *merge* these and codegen only one naturally
- ThinLTO selects one at Thin Link time ➔ other copies marked for drop after inlining

Thin-Link IPA: Compile Time Optimization
Weak Linkage Resolution

vector1.cpp 〉 No Selection

```cpp
#include <vector>

void foo(std::vector<int> &V) {
  V.push_back(1);
  V.push_back(2);
  V.push_back(3);
  V.push_back(4);
}
```

vector2.cpp 〉 No Selection

```cpp
#include <vector>

void bar(std::vector<int> &V) {
  V.push_back(11);
  V.push_back(22);
  V.push_back(33);
  V.push_back(44);
}
```

```llvm
source_filename = "vector2.cpp"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.12.0"

define void @_Z3fooRNSt3__16vectorIiNS_9allocatorIiEEEE(%"class.std::__1::vector"* dereferenceable(24)) #0 {
    ;   Lot
    ;   of
    ; stuff
  ret void
}

define linkonce_odr void @_ZNSt3__16vectorIiNS_9allocatorIiEEE21__push_back_slow_pathIiEEvOT_(%"class.std::__1::vector"*,
        i32* nocapture readonly dereferenceable(4)) #0 align 2 personality i8* bitcast (i32 (...)* @__gxx_personality_v0 to i8*) {
  %3 = getelementptr inbounds %"class.std::__1::vector", %"class.std::__1::vector"* %0, i64 0, i32 0, i32 1
  %4 = bitcast i32** %3 to i64*
  %5 = load i64, i64* %4, align 8, !tbaa !6
  %6 = bitcast %"class.std::__1::vector"* %0 to i64*
  %7 = load i64, i64* %6, align 8, !tbaa !11
  %8 = sub i64 %5, %7
  %9 = ashr exact i64 %8, 2
  %10 = add nsw i64 %9, 1
  %11 = icmp ugt i64 %10, 4611686018427387903
  br i1 %11, label %12, label %15

; <label>:12                                      ; preds = %2
```

… ~100 instructions

C++ template generates a lot of redundant code!

- Monolithic LTO will *merge* these and codegen only one naturally
- ThinLTO selects one at Thin Link time ➔ other copies marked for drop after inlining

Linking clang: **~25%** less function that are codegen!

Thin Link IPA: Dead Global Pruning

```
int Option = 42;

int getGlobalOption() {
  return Option;
}
```

```
extern int Option;

void setOption(int Value) {
  Option = Value;
}
```

Thin Link IPA: Dead Global Pruning

```c
int Option = 42;

int getGlobalOption() {
  return Option;
}
```

```c
extern int Option;

void setOption(int Value) {
  Option = Value;
}
```

```c
int Option
```

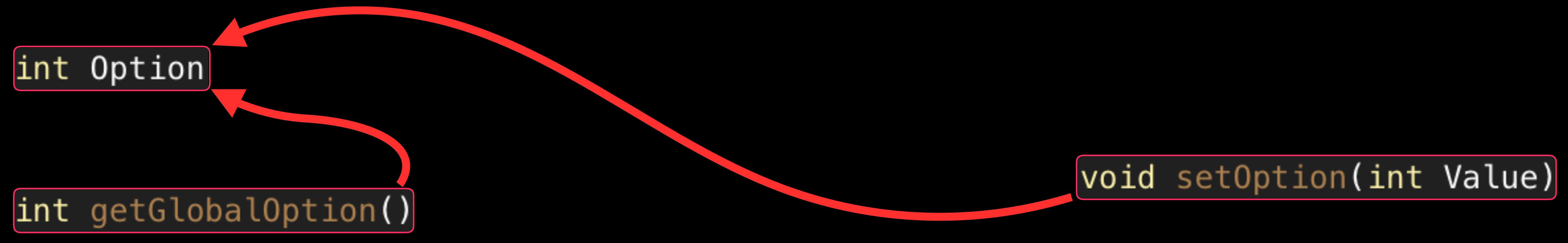
```c
int getGlobalOption()
```

```c
void setOption(int Value)
```

Thin Link IPA: Dead Global Pruning

```c
int Option = 42;

int getGlobalOption() {
  return Option;
}
```

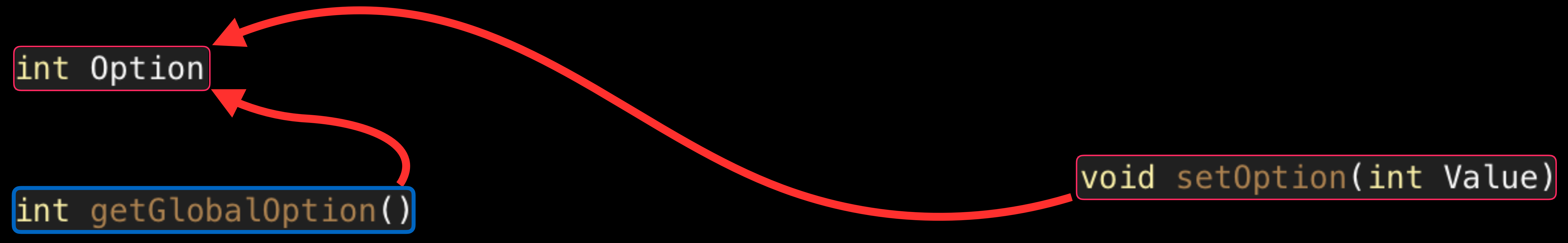
```c
extern int Option;

void setOption(int Value) {
  Option = Value;
}
```



Thin Link IPA: Dead Global Pruning

```c
int Option = 42;

int getGlobalOption() {
  return Option;
}
```

```c
extern int Option;

void setOption(int Value) {
  Option = Value;
}
```



- Linker identifies external reference to getGlobalOption()

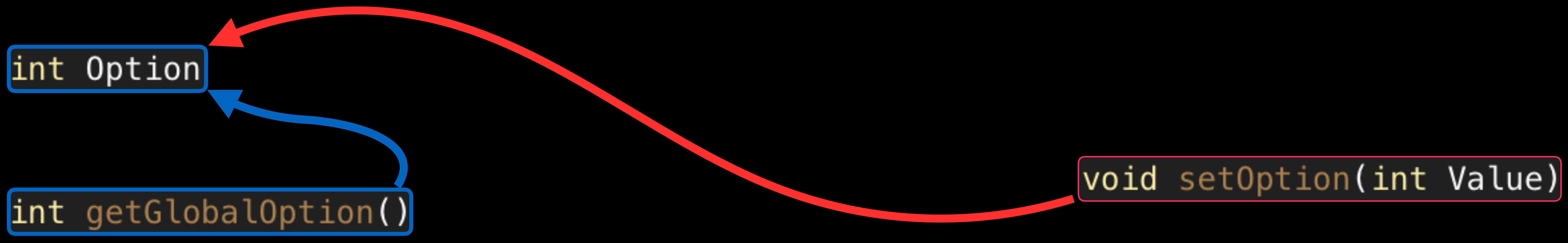
Linker Info:
external ref.

getGlobalOption

Thin Link IPA: Dead Global Pruning

```c
int Option = 42;

int getGlobalOption() {
  return Option;
}
```

```c
extern int Option;

void setOption(int Value) {
  Option = Value;
}
```



- Linker identifies external reference to getGlobalOption()
- Compute reachability to externally referenced nodes in index

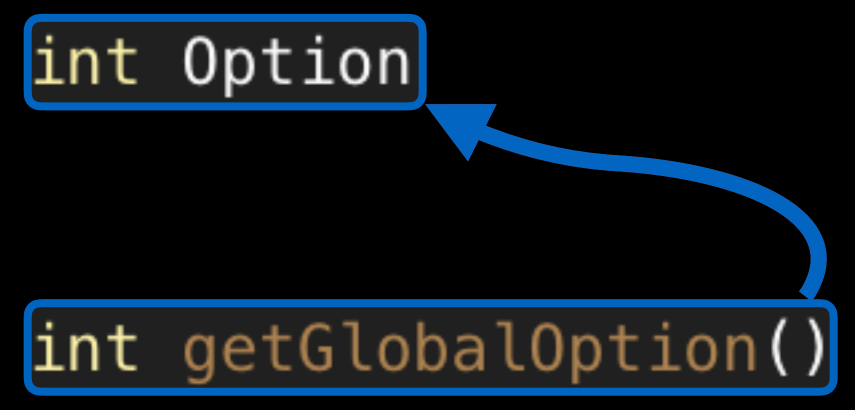
Linker Info:
external ref.

getGlobalOption

Thin Link IPA: Dead Global Pruning

```c
int Option = 42;

int getGlobalOption() {
  return Option;
}
```

```c
extern int Option;

void setOption(int Value) {
  Option = Value;
}
```



- Linker identifies external reference to getGlobalOption()
- Compute reachability to externally referenced nodes in index
- Prune unreachable nodes from the graph

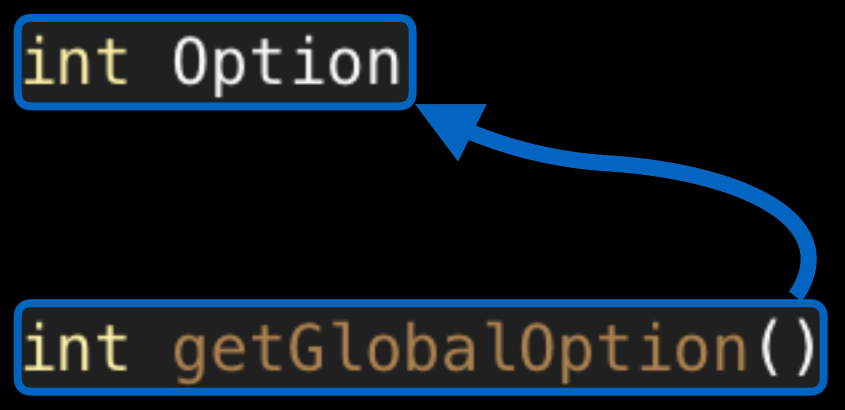
Linker Info:
external ref.

getGlobalOption

Thin Link IPA: Dead Global Pruning

```c
int Option = 42;

int getGlobalOption() {
  return Option;
}
```

```c
extern int Option;

void setOption(int Value) {
  Option = Value;
}
```



- Linker identifies external reference to getGlobalOption()
- Compute reachability to externally referenced nodes in index
- Prune unreachable nodes from the graph

Linker Info:
external ref.

getGlobalOption

→ **Enabler** for better subsequent analyses

- *Option* can be *internalized* and later constant folded.
- The function-importing will generate a smaller list → save CPU cycles!

Thin-Link IPA Future Optimization Example: Global Variables

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3(); <- this call is dead
  data = data + 42;
  return data;
}
```

Thin-Link IPA Future Optimization Example: Global Variables

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3(); <- this call is dead
  data = data + 42;
  return data;
}
```

main.c 〉 No Selection

```c
#include <stdio.h>

extern int i.llvm.A570184;

int main() {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

int i.llvm.A570184 = 0;

int foo3.llvm.A570184() {
  printf("Hi\n");
  return 10;
}
int foo1(void) {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

Thin-Link IPA Future Optimization Example: Global Variables

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3(); <- this call is dead
  data = data + 42;
  return data;
}
```

main.c 〉 No Selection

```c
#include <stdio.h>

extern int i.llvm.A570184;

int main() {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

int i.llvm.A570184 = 0;

int foo3.llvm.A570184() {
  printf("Hi\n");
  return 10;
}
int foo1(void) {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

Knowing the range for i.llvm.A570184
(here it is even easier: it is a constant)
would enable folding the test as false

Thin-Link IPA Future Optimization Example: Global Variables

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3(); <- this call is dead
  data = data + 42;
  return data;
}
```

main.c 〉 No Selection

```c
#include <stdio.h>

extern int i.llvm.A570184;

int main() {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

int i.llvm.A570184 = 0;

int foo3.llvm.A570184() {
  printf("Hi\n");
  return 10;
}
int foo1(void) {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

Knowing the range for i.llvm.A570184 (here it is even easier: it is a constant) would enable folding the test as false

All this code can be dead-stripped by the linker, but takes time to optimize/codegen

Thin-Link IPA Future Optimization Example: Global Variables

main.c 〉 No Selection

```c
#include <stdio.h>

// Defined in a.c
int foo1(void);


void foo4(void) {
  printf("Hi\n");
}


int main() {
  return foo1();
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

static signed int i = 0;

void foo2(void) {
  i = -1;
}
static int foo3() {
  foo4();
  return 10;
}
int foo1(void) {
  int data = 0;
  if (i < 0)
    data = foo3(); <- this call is dead
  data = data + 42;
  return data;
}
```

main.c 〉 No Selection

```c
#include <stdio.h>

extern int i.llvm.A570184;

int main() {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

a.c 〉 No Selection

```c
void foo4(void); // Defined in main.c

int i.llvm.A570184 = 0;

int foo3.llvm.A570184() {
  printf("Hi\n");
  return 10;
}
int foo1(void) {
  if (i.llvm.A570184 < 0)
    return printf("Hi\n");
  return 42;
}
```

Knowing the range for i.llvm.A570184 (here it is even easier: it is a constant) would enable folding the test as false

All this code can be dead-stripped by the linker, but takes time to optimize/codegen

This is just one example of opportunity for better summary-based optimization!

Profile Guided Optimization (PGO): Indirect Call Promotion

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int bar_impl() {
  // do stuff
  // ...
  return value;
}

int foo() {
  // do stuff
  // ...


  return bar();
}
```

Profile Guided Optimization (PGO): Indirect Call Promotion

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int bar_impl() {
  // do stuff
  // ...
  return value;
}

int foo() {
  // do stuff
  // ...
  if (bar == &bar_impl)
    return bar_impl()
  return bar();
}
```

Profile Guided Optimization (PGO): Indirect Call Promotion

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int bar_impl() {
  // do stuff
  // ...
  return value;
}

int foo() {
  // do stuff
  // ...
  if (bar == &bar_impl)
    return bar_impl()
  return bar();
}
```

Indirect Call Promotion can only promote if the target is in the same module.

Profile Guided Optimization (PGO): Indirect Call Promotion

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int bar_impl() {
  // do stuff
  // ...
  return value;
}

int foo() {
  // do stuff
  // ...
  if (bar == &bar_impl)
    return bar_impl()
  return bar();
}
```

indirect_target.cpp 〉 No Selection

```cpp
int bar_impl() {
  // do stuff
  // ...
  return value;
}

int (*bar)() = &bar_impl;


```

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int foo() {
  // do stuff
  // ...


  return bar();
}
```

Indirect Call Promotion can only promote if the target is in the same module.

PGO Indirect Call Promotion

indirect_target.cpp 〉 No Selection

```cpp
int bar_impl() {
  // do stuff
  // ...
  return value;
}

int (*bar)() = &bar_impl;

```

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int foo() {
  // do stuff
  // ...


  return bar();
}
```

PGO Indirect Call Promotion

indirect_target.cpp 〉 No Selection

```cpp
int bar_impl() {
  // do stuff
  // ...
  return value;
}

int (*bar)() = &bar_impl;
```

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int foo() {
  // do stuff
  // ...


  return bar();
}
```

PGO Indirect Call Promotion

indirect_target.cpp 〉 No Selection

```cpp
int bar_impl() {
  // do stuff
  // ...
  return value;
}

int (*bar)() = &bar_impl;
```

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int foo() {
  // do stuff
  // ...


  return bar();
}
```

PGO Indirect Call Promotion

indirect_target.cpp 〉 No Selection

```cpp
int bar_impl() {
  // do stuff
  // ...
  return value;
}

int (*bar)() = &bar_impl;

```

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int foo() {
  // do stuff
  // ...


  return bar();
}
```

clang -cc1 -flto=thin

indirect_call.o

```llvm
source_filename = "indirect_call.cpp"

@bar = external global i32 ()*, align 8

define i32 @_Z3foov() #0 {
  %1 = load i32 ()*, i32 ()** @bar, align 8
  %2 = call i32 %1()
  ret i32 %2
}


```

Summary:
_Z5test2i, 13 instructions,
global linkage,

PGO Indirect Call Promotion

indirect_target.cpp 〉 No Selection

```cpp
int bar_impl() {
  // do stuff
  // ...
  return value;
}

int (*bar)() = &bar_impl;
```

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int foo() {
  // do stuff
  // ...


  return bar();
}
```

clang -cc1 -flto=thin **-fprofile-use**

indirect_call.o

```
source_filename = "indirect_call.cpp"

@bar = external global i32 ()*, align 8

define i32 @_Z3foov() #0 {
  %1 = load i32 ()*, i32 ()** @bar, align 8
  %2 = call i32 %1(), !prof !33
  ret i32 %2
}

!33 = !{!"VP", i32 0, i64 1, i64 124279134, i64 1}
```

Summary:
_Z5test2i, 13 instructions,
global linkage,

PGO Function name hash

PGO Indirect Call Promotion

indirect_target.cpp 〉 No Selection

```cpp
int bar_impl() {
  // do stuff
  // ...
  return value;
}

int (*bar)() = &bar_impl;

```

indirect_call.cpp 〉 No Selection

```cpp
extern int (*bar)();

int foo() {
  // do stuff
  // ...


  return bar();
}
```

clang -cc1 -flto=thin **-fprofile-use**

indirect_call.o

```
source_filename = "indirect_call.cpp"

@bar = external global i32 ()*, align 8

define i32 @_Z3foov() #0 {
  %1 = load i32 ()*, i32 ()** @bar, align 8
  %2 = call i32 %1(), !prof !33
  ret i32 %2
}

!33 = !{!"VP", i32 0, i64 1, i64 124279134, i64 1}
```

Summary:
_Z5test2i, 13 instructions,
global linkage, **call 768595e**

PGO Function name hash

Will also be used as key in the Combined Index

The summary records indirect calls possible targets as regular calls (speculative).

LLVM Linker Plugin

PGO Indirect Call Promotion

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

@foo = external local_unnamed_addr global i32 (i32)*
define i32 @_Z5test2i(i32 %b) #0 !prof !28 {
  ; Other stuff here
  %0 = load i32 (i32)*, i32 (i32)** @foo, align 8
  %add = add nsw i32 %b, 1
  %call = tail call i32 %0(i32 %add), !prof !33
  ret i32 %call
}
!33 = !{!"VP", i32 0, i64 1, i64 124279134, i64 1}
```

Summary:
_Z5test2i, 13 instructions,
global linkage, **call 768595e**

Thin-link:

LLVM Linker Plugin

PGO Indirect Call Promotion

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

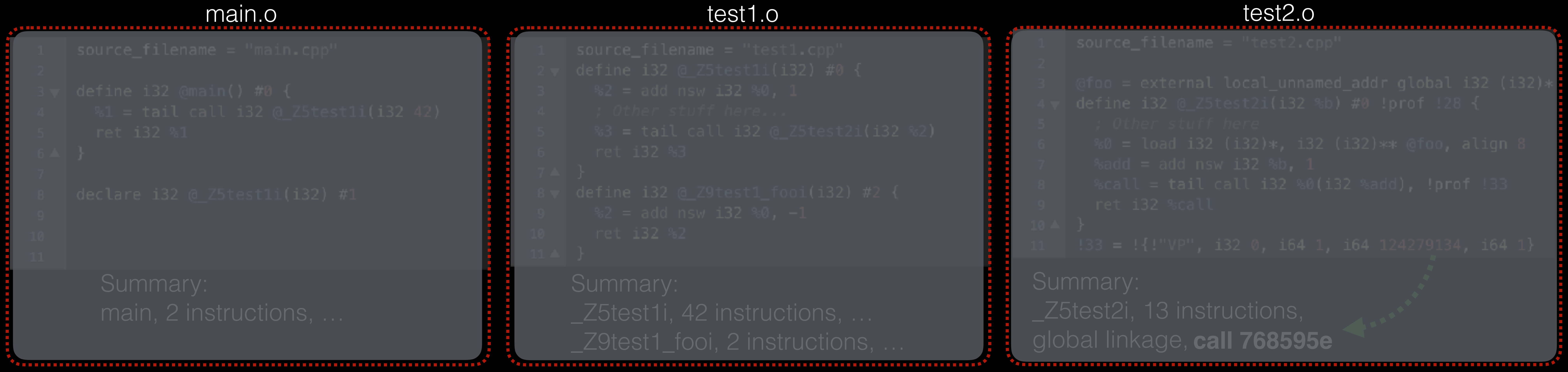
```
source_filename = "test2.cpp"

@foo = external local_unnamed_addr global i32 (i32)*
define i32 @_Z5test2i(i32 %b) #0 !prof !28 {
  ; Other stuff here
  %0 = load i32 (i32)*, i32 (i32)** @foo, align 8
  %add = add nsw i32 %b, 1
  %call = tail call i32 %0(i32 %add), !prof !33
  ret i32 %call
}
!33 = !{!"VP", i32 0, i64 1, i64 124279134, i64 1}
```

Summary:
_Z5test2i, 13 instructions,
global linkage, **call 768595e**

Thin-link:

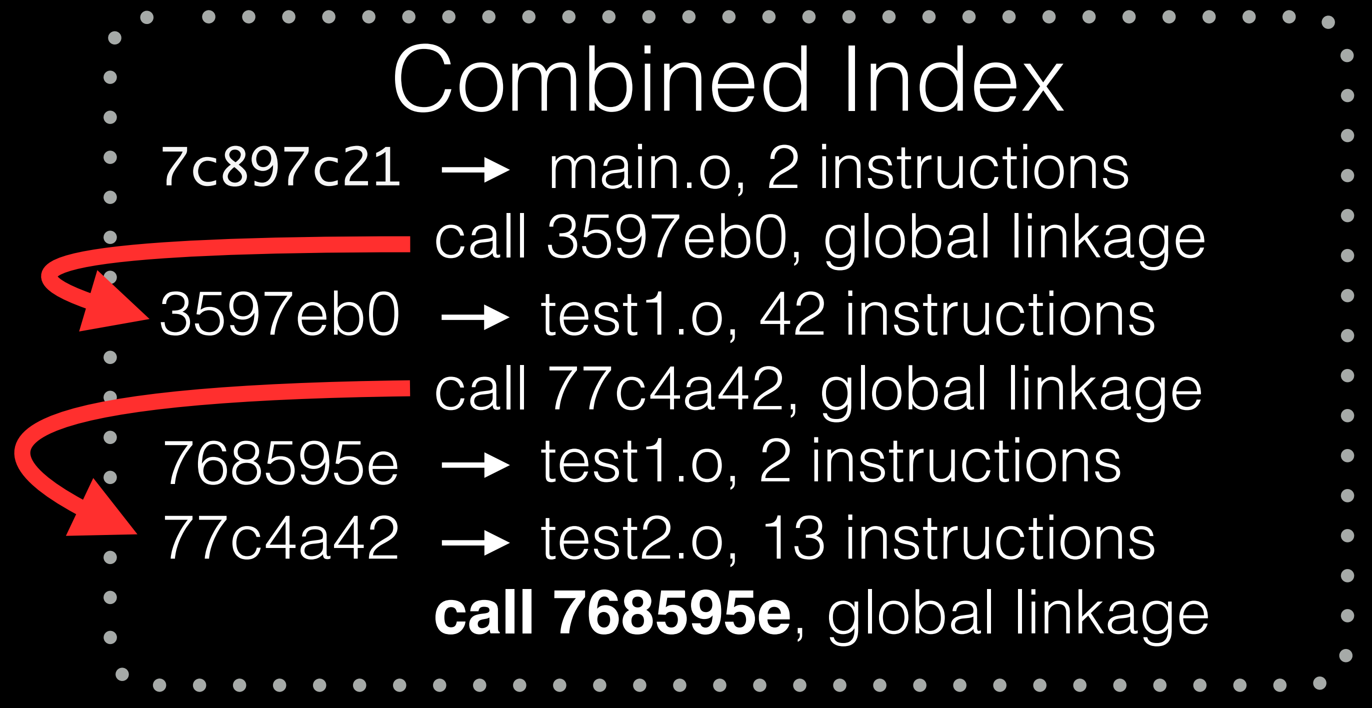
LLVM Linker Plugin

PGO Indirect Call Promotion



Thin-link:

- Produce the combined index as usual



PGO Indirect Call Promotion

LLVM Linker Plugin

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

@foo = external local_unnamed_addr global i32 (i32)*
define i32 @_Z5test2i(i32 %b) #0 !prof !28 {
  ; Other stuff here
  %0 = load i32 (i32)*, i32 (i32)** @foo, align 8
  %add = add nsw i32 %b, 1
  %call = tail call i32 %0(i32 %add), !prof !33
  ret i32 %call
}
!33 = !{!"VP", i32 0, i64 1, i64 124279134, i64 1}
```

Summary:
_Z5test2i, 13 instructions,
global linkage, **call 768595e**

Thin-link:

- Produce the combined index as usual
- Indirect profile-based edge points to target, if in index, the same way as direct edges.

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0, global linkage
3597eb0 → test1.o, 42 instructions
call 77c4a42, global linkage
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e, global linkage

LLVM Linker Plugin

PGO Indirect Call Promotion

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

@foo = external local_unnamed_addr global i32 (i32)*
define i32 @_Z5test2i(i32 %b) #0 !prof !28 {
  ; Other stuff here
  %0 = load i32 (i32)*, i32 (i32)** @foo, align 8
  %add = add nsw i32 %b, 1
  %call = tail call i32 %0(i32 %add), !prof !33
  ret i32 %call
}
!33 = !{!"VP", i32 0, i64 1, i64 124279134, i64 1}
```

Summary:
_Z5test2i, 13 instructions,
global linkage, **call 768595e**

Thin-link:

- Produce the combined index as usual
- Indirect profile-based edge points to target, if in index, the same way as direct edges.
- The indirect call target can be selected for import, just like targets of direct calls.

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0, global linkage
3597eb0 → test1.o, 42 instructions
call 77c4a42, global linkage
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e, global linkage

LLVM Linker Plugin

PGO Indirect Call Promotion

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions, …

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions, …
_Z9test1_fooi, 2 instructions, …

test2.o

```
source_filename = "test2.cpp"

@foo = external local_unnamed_addr global i32 (i32)*
define i32 @_Z5test2i(i32 %b) #0 !prof !28 {
  ; Other stuff here
  %0 = load i32 (i32)*, i32 (i32)** @foo, align 8
  %add = add nsw i32 %b, 1
  %call = tail call i32 %0(i32 %add), !prof !33
  ret i32 %call
}
!33 = !{!"VP", i32 0, i64 1, i64 124279134, i64 1}
```

Summary:
_Z5test2i, 13 instructions,
global linkage, **call 768595e**

Thin-link:

- Produce the combined index as usual
- Indirect profile-based edge points to target, if in index, the same way as direct edges.
- The indirect call target can be selected for import, just like targets of direct calls.
- Indirect calls are promoted after importing, but before the inliner

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0, global linkage
3597eb0 → test1.o, 42 instructions
call 77c4a42, global linkage
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e, global linkage

Profile Guided Optimization (PGO): Importing Heuristics

main.cpp 〉 No Selection

```cpp



void hot();
void cold();
void foo(bool TakeHotPath) {
  if (TakeHotPath)
    hot();
  else
    cold();
}
```

kernel.cpp 〉 No Selection

```cpp
void hot() {
  // do stuff...

  // 150 instructions here
}

void cold() {
  // do stuff...

  // 50 instructions here
}
```

Summary:
main, 10 instructions, global linkage,
call f9f84d51
call f46f05f7

Summary:

_Z3hotv, ***150 instructions***, global linkage
_Z4coldv, ***50 instructions***, global linkage

Profile Guided Optimization (PGO): Importing Heuristics

main.cpp 〉 No Selection

```cpp



void hot();
void cold();
void foo(bool TakeHotPath) {
  if (TakeHotPath)
    hot();
  else
    cold();
}
```

kernel.cpp 〉 No Selection

```cpp
void hot() {
  // do stuff...

  // 150 instructions here
}

void cold() {
  // do stuff...

  // 50 instructions here
}
```

Summary:
main, 10 instructions, global linkage,
  call f9f84d51
  call f46f05f7

Summary:

_Z3hotv, ***150 instructions***, global linkage
_Z4coldv, ***50 instructions***, global linkage

→ Only `cold()` would be inlined!

Profile Guided Optimization (PGO): Importing Heuristics

main.cpp 〉 No Selection

```cpp



void hot();
void cold();
void foo(bool TakeHotPath) {
  if (TakeHotPath)
    hot();
  else
    cold();
}
```

kernel.cpp 〉 No Selection

```cpp
void hot() {
  // do stuff...

  // 150 instructions here
}

void cold() {
  // do stuff...

  // 50 instructions here
}
```

Summary:
main, 10 instructions, global linkage,
call f9f84d51
call f46f05f7

Summary:

_Z3hotv, ***150 instructions***, global linkage
_Z4coldv, ***50 instructions***, global linkage

→ Only `cold()` would be inlined!

The importer would only import *cold()*!

Profile Guided Optimization (PGO): Importing Heuristics

main.cpp 〉 No Selection

```cpp
#define likely(x) \
  __builtin_expect(!!(x), 1)

void hot();
void cold();
void foo(bool TakeHotPath) {
  if (likely(TakeHotPath))
    hot();
  else
    cold();
}
```

kernel.cpp 〉 No Selection

```cpp
void hot() {
  // do stuff...

  // 150 instructions here
}

void cold() {
  // do stuff...

  // 50 instructions here
}
```

Summary:
main, 10 instructions, global linkage,
  call f9f84d51 → _Z3hotv, ***150 instructions***, global linkage
  call f46f05f7 → _Z4coldv, ***50 instructions***, global linkage

Summary:

→ Only `cold()` would be inlined! The importer would only import *cold()*!

With PGO data, `cold` will not be inlined, `hot` would be inlined if available

Profile Guided Optimization (PGO): Importing Heuristics

main.cpp 〉 No Selection

```cpp
#define likely(x) \
  __builtin_expect(!!(x), 1)

void hot();
void cold();
void foo(bool TakeHotPath) {
  if (likely(TakeHotPath))
    hot();
  else
    cold();
}
```

kernel.cpp 〉 No Selection

```cpp
void hot() {
  // do stuff...

  // 150 instructions here
}

void cold() {
  // do stuff...

  // 50 instructions here
}
```

Summary:
main, 10 instructions, global linkage,
call f9f84d51, ***hot***
call f46f05f7, ***cold***

Summary:
_Z3hotv, ***150 instructions***, global linkage
_Z4coldv, ***50 instructions***, global linkage

→ Only `cold()` would be inlined!

The importer would only import *cold()*!

With PGO data, `cold` will not be inlined, `hot` would be inlined if available

Profile Guided Optimization (PGO): Importing Heuristics

main.cpp 〉 No Selection

```cpp
#define likely(x) \
  __builtin_expect(!!(x), 1)

void hot();
void cold();
void foo(bool TakeHotPath) {
  if (likely(TakeHotPath))
    hot();
  else
    cold();
}
```

kernel.cpp 〉 No Selection

```cpp
void hot() {
  // do stuff...

  // 150 instructions here
}

void cold() {
  // do stuff...

  // 50 instructions here
}
```

Summary:
main, 10 instructions, global linkage,
call f9f84d51, ***hot*** → _Z3hotv, ***150 instructions***, global linkage
call f46f05f7, ***cold*** → _Z4coldv, ***50 instructions***, global linkage

Summary:

→ Only `cold()` would be inlined!   The importer would only import *cold()*!

With PGO data, `cold` will not be inlined, `hot` would be inlined if available

→ Mirror inliner heuristic by giving bonus for hot edges and a penalty for cold ones!

ThinLTO Revisited: Incremental Build

main.cpp 〉 No Selection

```
int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

ThinLTO Revisited: Incremental Build

main.cpp 〉 No Selection

```cpp

int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

ThinLTO Revisited: Incremental Build

main.cpp 〉 No Selection

```cpp
int test1(int a);

int main() {
  return test1(42);
}
```

test1.cpp 〉 No Selection

```cpp
int test2(int);
int test1(int a) {
  // Do stuff
  // ...
  return test2(a+1);
}
int test1_foo(int bar) {
  return bar-1;
}
```

test2.cpp 〉 No Selection

```cpp
int test1_foo(int);

int test2(int b) {
  // Do stuff
  return test1_foo(b + 1);
}
```

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

clang -cc1 -flto=**thin**

main.o

```llvm
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```llvm
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```llvm
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

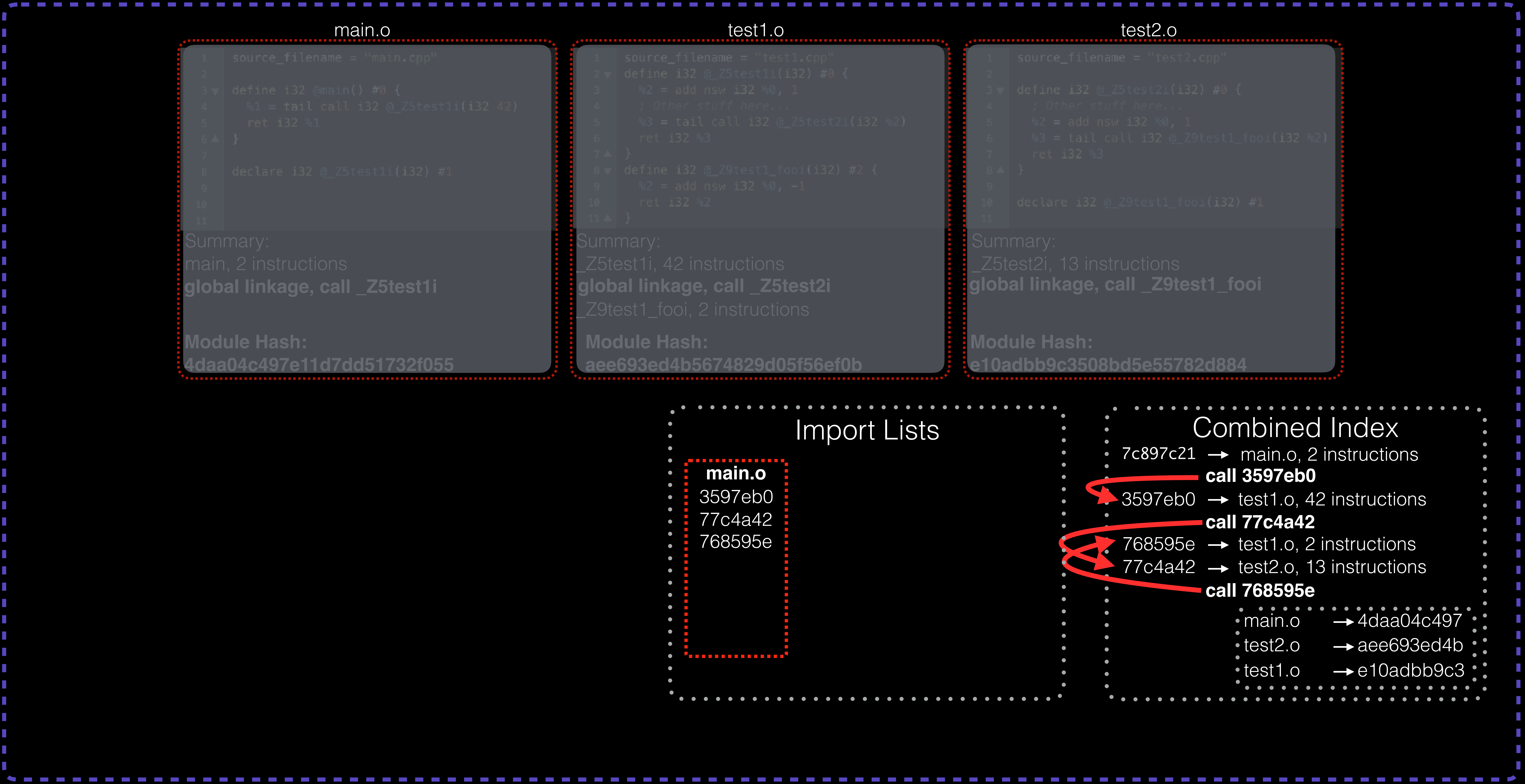
Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

main.o → 4daa04c497
test2.o → aee693ed4b
test1.o → e10adbb9c3

ThinLTO Revisited: Incremental Build

libLTO.dylib



libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Import Lists

main.o
3597eb0
77c4a42
768595e

test1.o
e10adbb
test2.o
aee693ed

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

main.o → 4daa04c497
test2.o → aee693ed4b
test1.o → e10adbb9c3

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Import Lists

main.o	**test1.o**	**test2.o**
3597eb0	77c4a42	768595e
77c4a42		
768595e		
test1.o	test2.o	test1.o
e10adbb	aee693ed	e10adbb
test2.o		
aee693ed		

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

main.o → 4daa04c497
test2.o → aee693ed4b
test1.o → e10adbb9c3

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Import Lists

main.o	**test1.o**	**test2.o**
3597eb0 77c4a42 768595e	77c4a42	768595e
test1.o e10adbb test2.o aee693ed	test2.o aee693ed	test1.o e10adbb

Combined Index

7c897c21 → main.o, 2 instructions
call 3597eb0
3597eb0 → test1.o, 42 instructions
call 77c4a42
768595e → test1.o, 2 instructions
77c4a42 → test2.o, 13 instructions
call 768595e

main.o → 4daa04c497
test2.o → aee693ed4b
test1.o → e10adbb9c3

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Import Lists

main.o
3597eb0
77c4a42
768595e

test1.o
e10adbb
test2.o
aee693ed

test1.o
77c4a42

test2.o
aee693ed

test2.o
768595e

test1.o
e10adbb

ThinLTO Revisited: Incremental Build

libLTO.dylib

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Import Lists

main.o	**test1.o**	**test2.o**
3597eb0 77c4a42 768595e	77c4a42	768595e
test1.o e10adbb test2.o aee693ed	test2.o aee693ed	test1.o e10adbb

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Parallel ThinLTO Backends:

- Read the import list

main.o
3597eb0
77c4a42
768595e

test1.o
e10adbb
test2.o
aee693ed

Import Lists

test1.o
77c4a42

test2.o
aee693ed

test2.o
768595e

test1.o
e10adbb

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Parallel ThinLTO Backends:

- Read the import list
- Compute a unique hash based on all the inputs: **without reading any IR**!

main.o
3597eb0
77c4a42
768595e

test1.o
e10adbb
test2.o
aee693ed

12f67fe8

Import Lists

test1.o
77c4a42

test2.o
aee693ed

test2.o
768595e

test1.o
e10adbb

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Parallel ThinLTO Backends:

- Read the import list
- Compute a unique hash based on all the inputs: **without reading any IR**!
- Query a persistent cache and early return on cache hit.

main.o
3597eb0
77c4a42
768595e

test1.o
e10adbb
test2.o
aee693ed

12f67fe8

Import Lists

test1.o
77c4a42

test2.o
aee693ed

test2.o
768595e

test1.o
e10adbb

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Parallel ThinLTO Backends:

- Read the import list
- Compute a unique hash based on all the inputs: **without reading any IR**!
- Query a persistent cache and early return on cache hit.
- Otherwise, process as normal, but commit the object to the cache before returning to the linker.

main.o
3597eb0
77c4a42
768595e

test1.o
e10adbb
test2.o
aee693ed

12f67fe8

Import Lists

test1.o
77c4a42

test2.o
aee693ed

test2.o
768595e

test1.o
e10adbb

libLTO.dylib

ThinLTO Revisited: Incremental Build

main.o

```
source_filename = "main.cpp"

define i32 @main() #0 {
  %1 = tail call i32 @_Z5test1i(i32 42)
  ret i32 %1
}

declare i32 @_Z5test1i(i32) #1
```

Summary:
main, 2 instructions
global linkage, call _Z5test1i

Module Hash:
4daa04c497e11d7dd51732f055

test1.o

```
source_filename = "test1.cpp"
define i32 @_Z5test1i(i32) #0 {
  %2 = add nsw i32 %0, 1
  ; Other stuff here...
  %3 = tail call i32 @_Z5test2i(i32 %2)
  ret i32 %3
}
define i32 @_Z9test1_fooi(i32) #2 {
  %2 = add nsw i32 %0, -1
  ret i32 %2
}
```

Summary:
_Z5test1i, 42 instructions
global linkage, call _Z5test2i
_Z9test1_fooi, 2 instructions

Module Hash:
aee693ed4b5674829d05f56ef0b

test2.o

```
source_filename = "test2.cpp"

define i32 @_Z5test2i(i32) #0 {
  ; Other stuff here...
  %2 = add nsw i32 %0, 1
  %3 = tail call i32 @_Z9test1_fooi(i32 %2)
  ret i32 %3
}

declare i32 @_Z9test1_fooi(i32) #1
```

Summary:
_Z5test2i, 13 instructions
global linkage, call _Z9test1_fooi

Module Hash:
e10adbb9c3508bd5e55782d884

Parallel ThinLTO Backends:

- Read the import list
- Compute a unique hash based on all the inputs: **without reading any IR**!
- Query a persistent cache and early return on cache hit.
- Otherwise, process as normal, but commit the object to the cache before returning to the linker.
- The final native link is not incremental: all the objects are re-linked
  → always bit-to-bit the same binary.

main.o
3597eb0
77c4a42
768595e

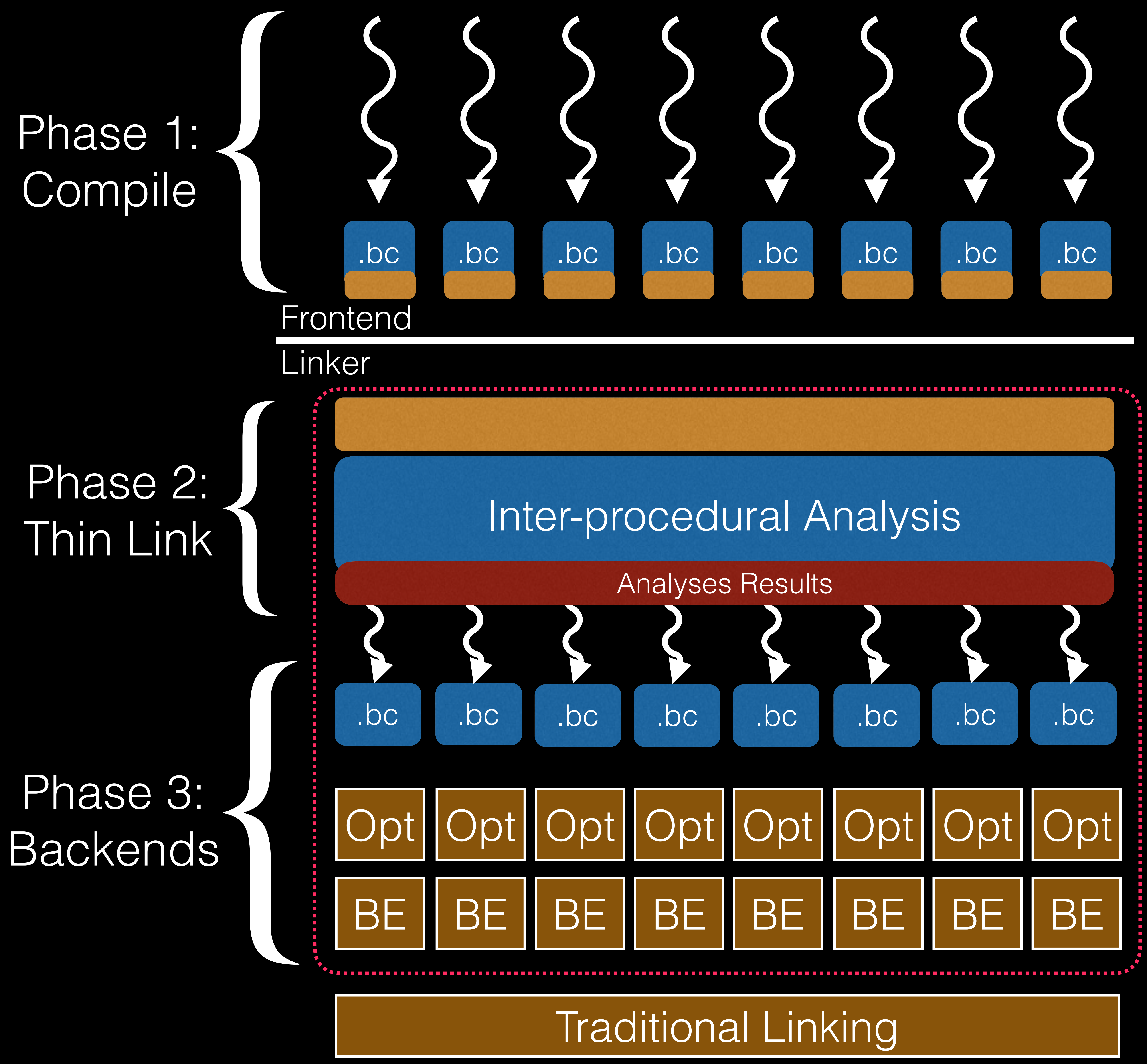
test1.o
e10adbb
test2.o
aee693ed

12f67fe8

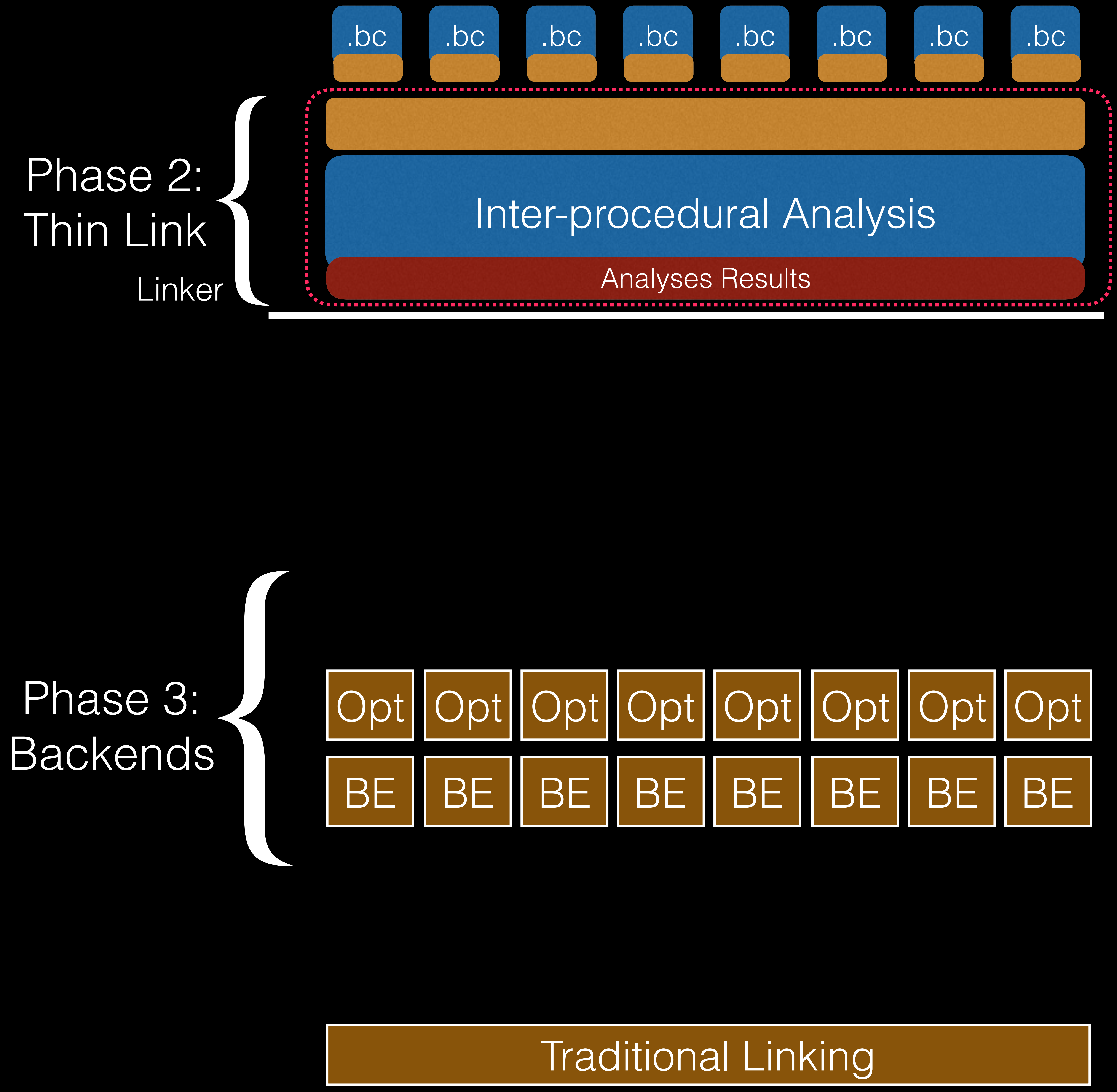
Import Lists

test1.o	**test2.o**
77c4a42	768595e
test2.o aee693ed	test1.o e10adbb

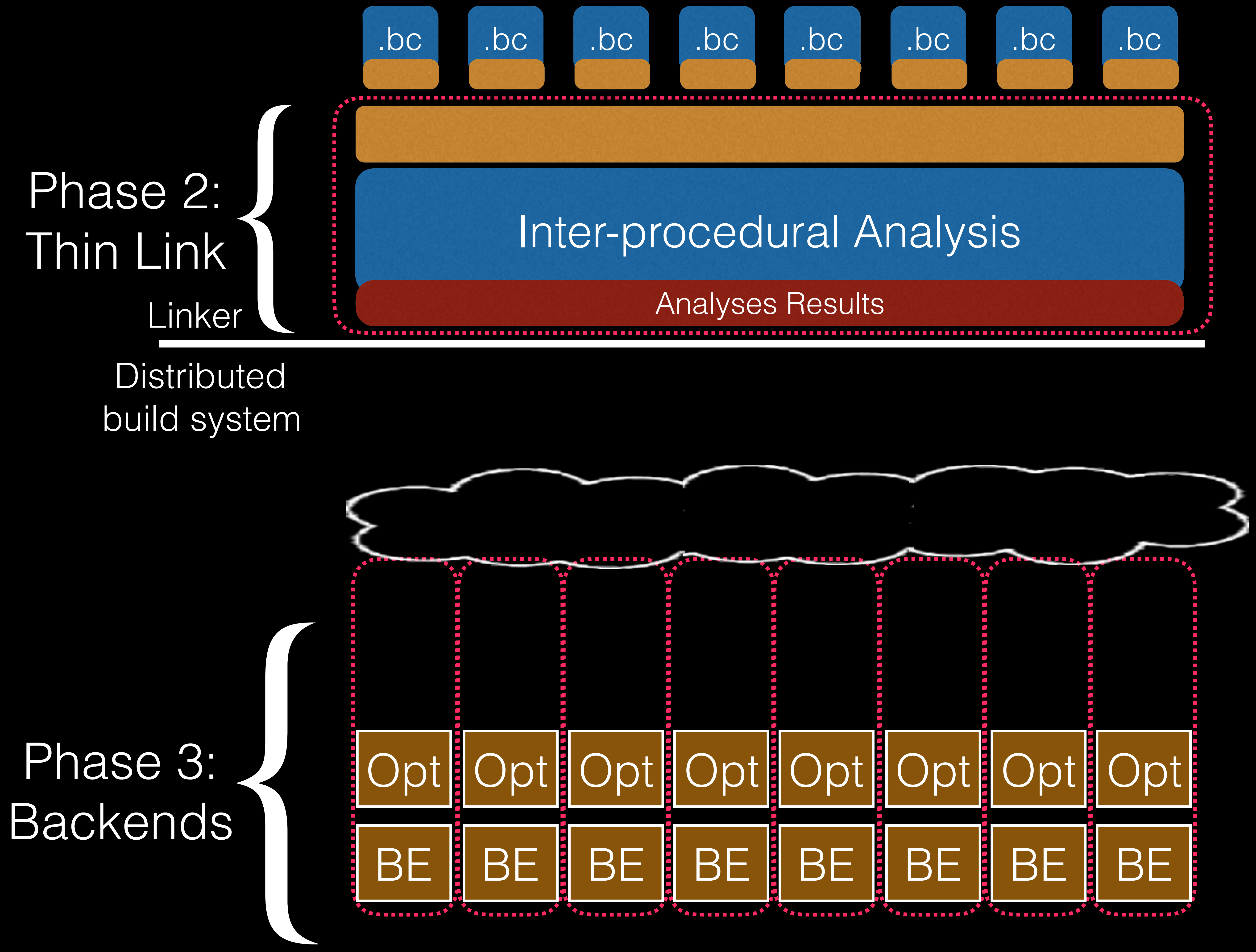
ThinLTO: Distributed Builds



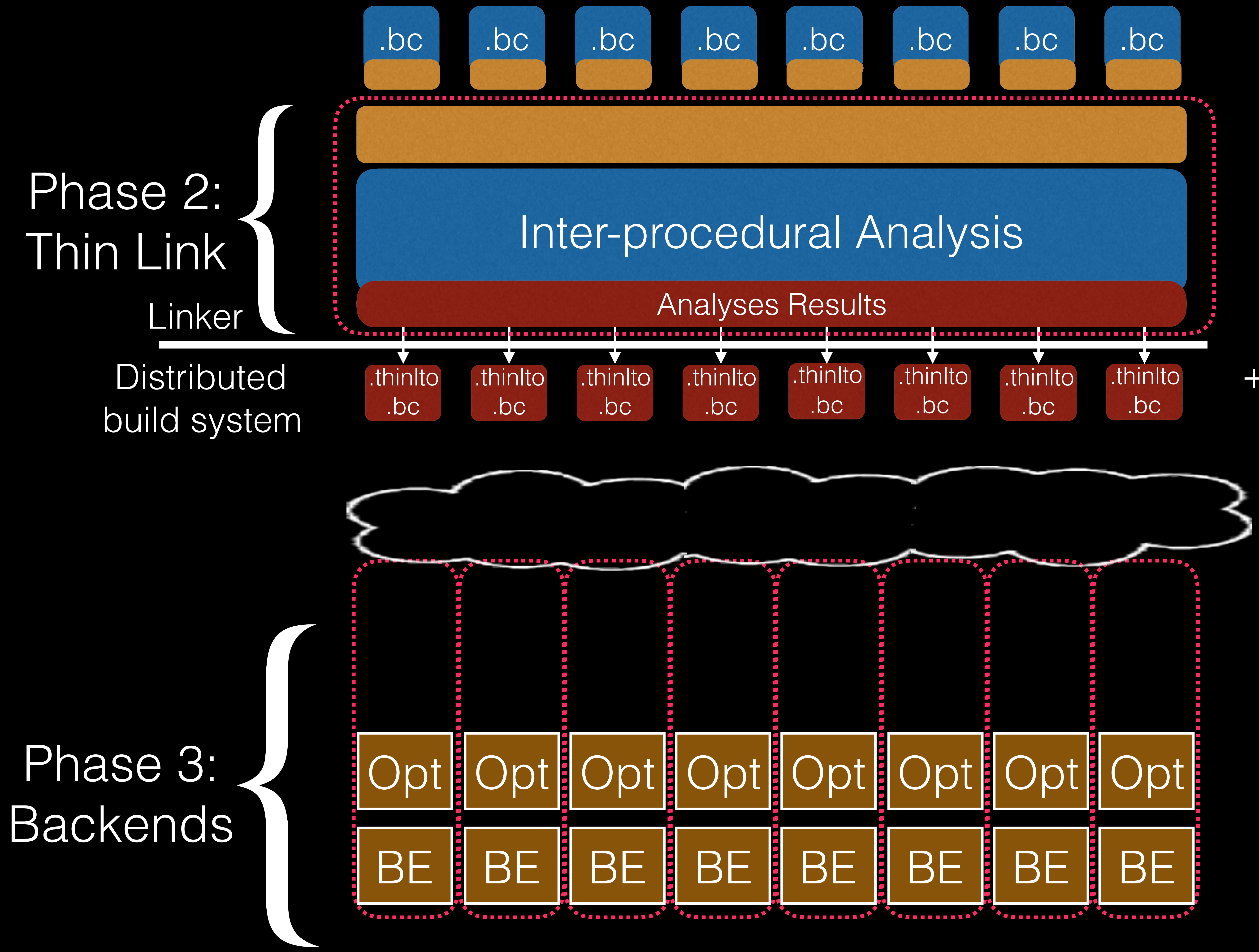
ThinLTO: Distributed Builds



ThinLTO: Distributed Builds

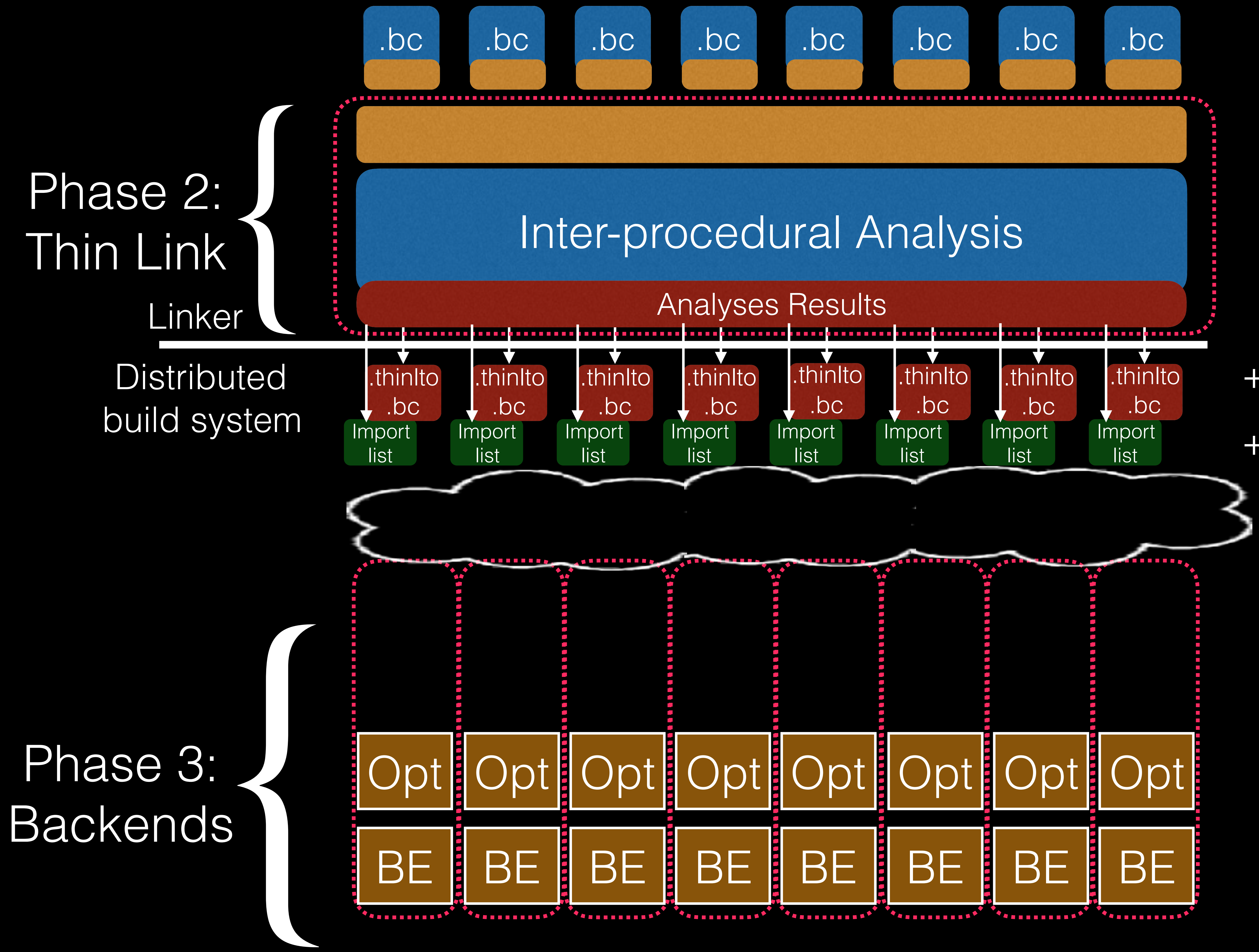


ThinLTO: Distributed Builds



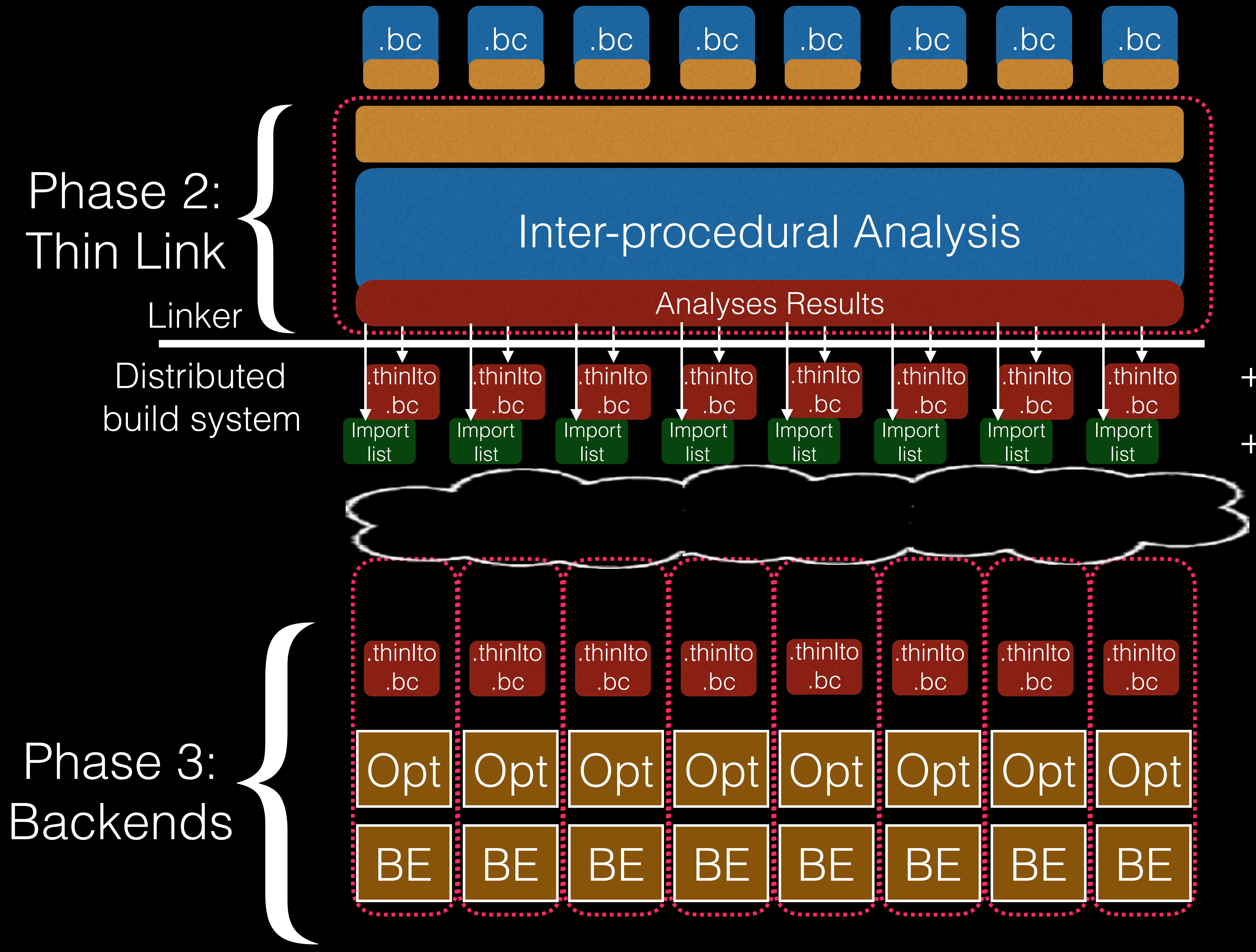
\+ Stream out analysis results for each backend job

ThinLTO: Distributed Builds



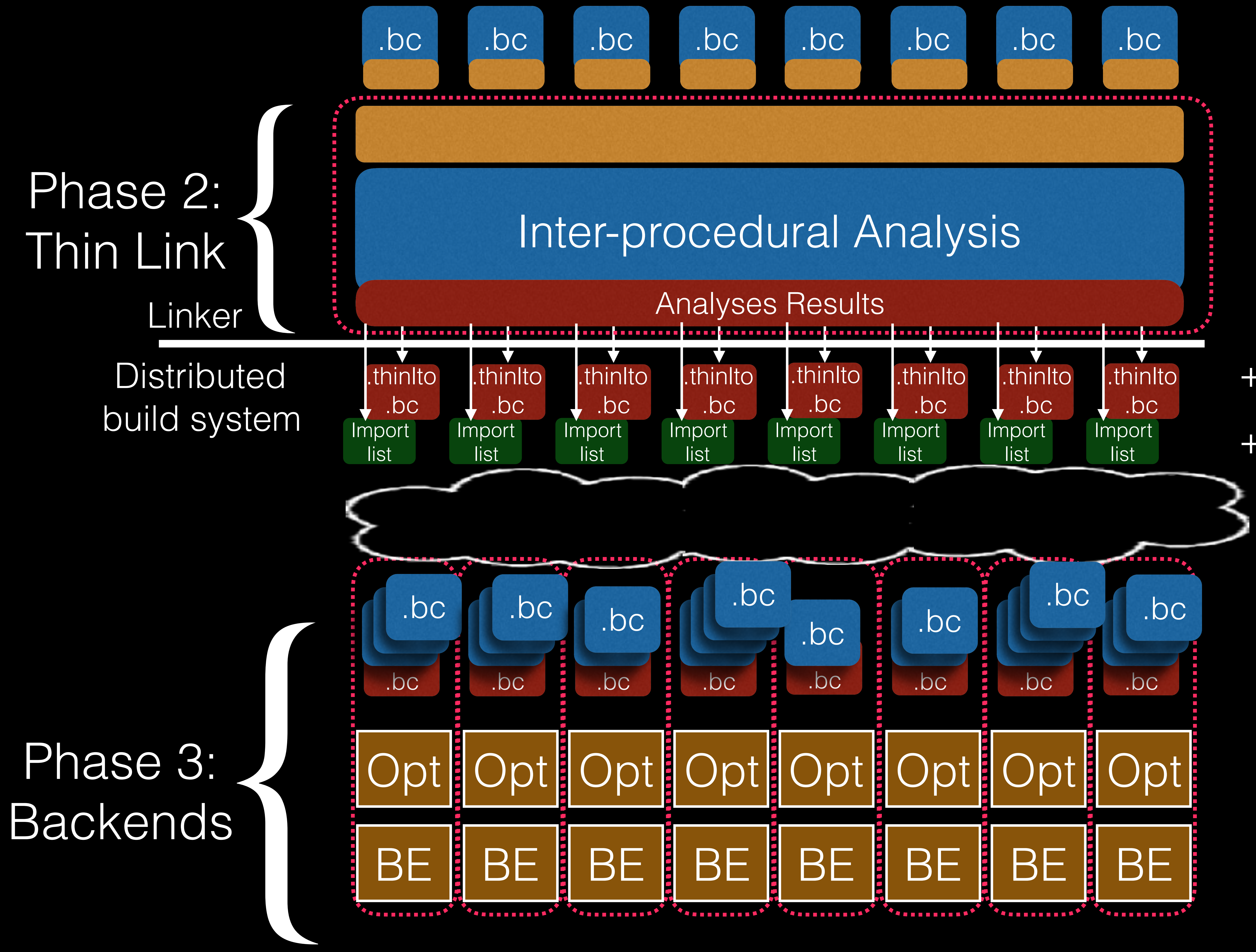
+ Stream out analysis results for each backend job
+ Produce dependent module lists:
  the build system bundles these for each job

ThinLTO: Distributed Builds



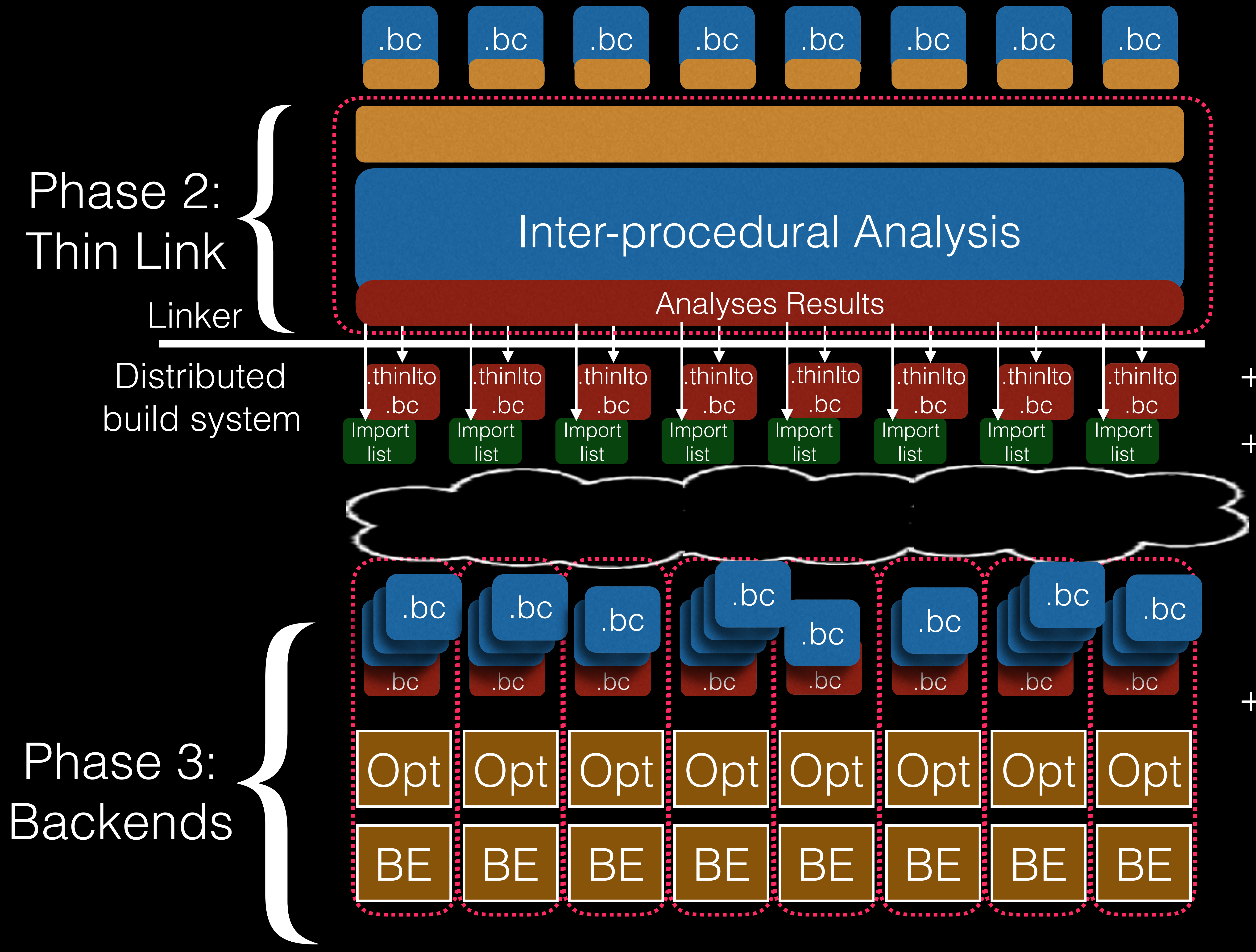
- Stream out analysis results for each backend job
- Produce dependent module lists:
  the build system bundles these for each job

ThinLTO: Distributed Builds



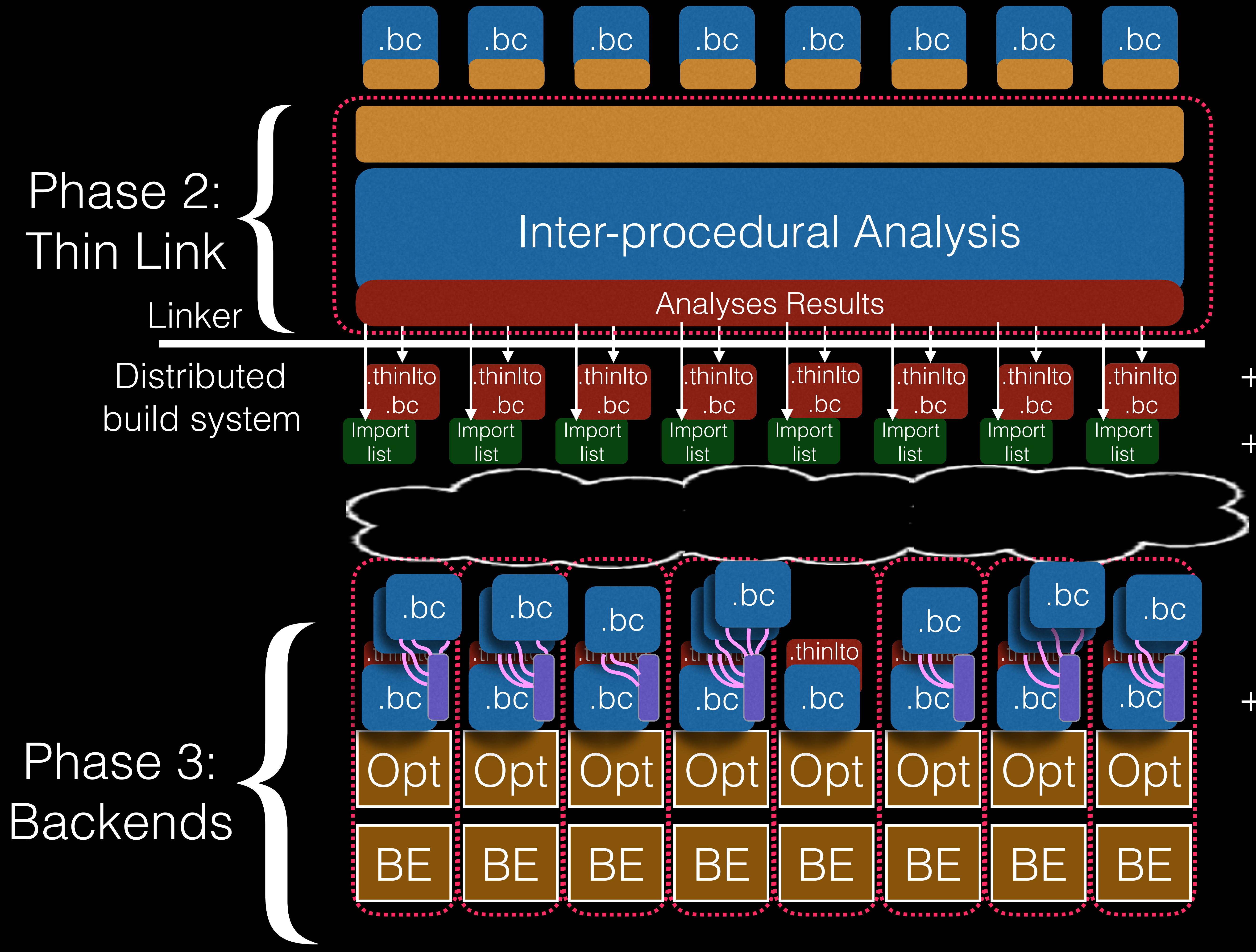
+ Stream out analysis results for each backend job
+ Produce dependent module lists:
  the build system bundles these for each job

ThinLTO: Distributed Builds



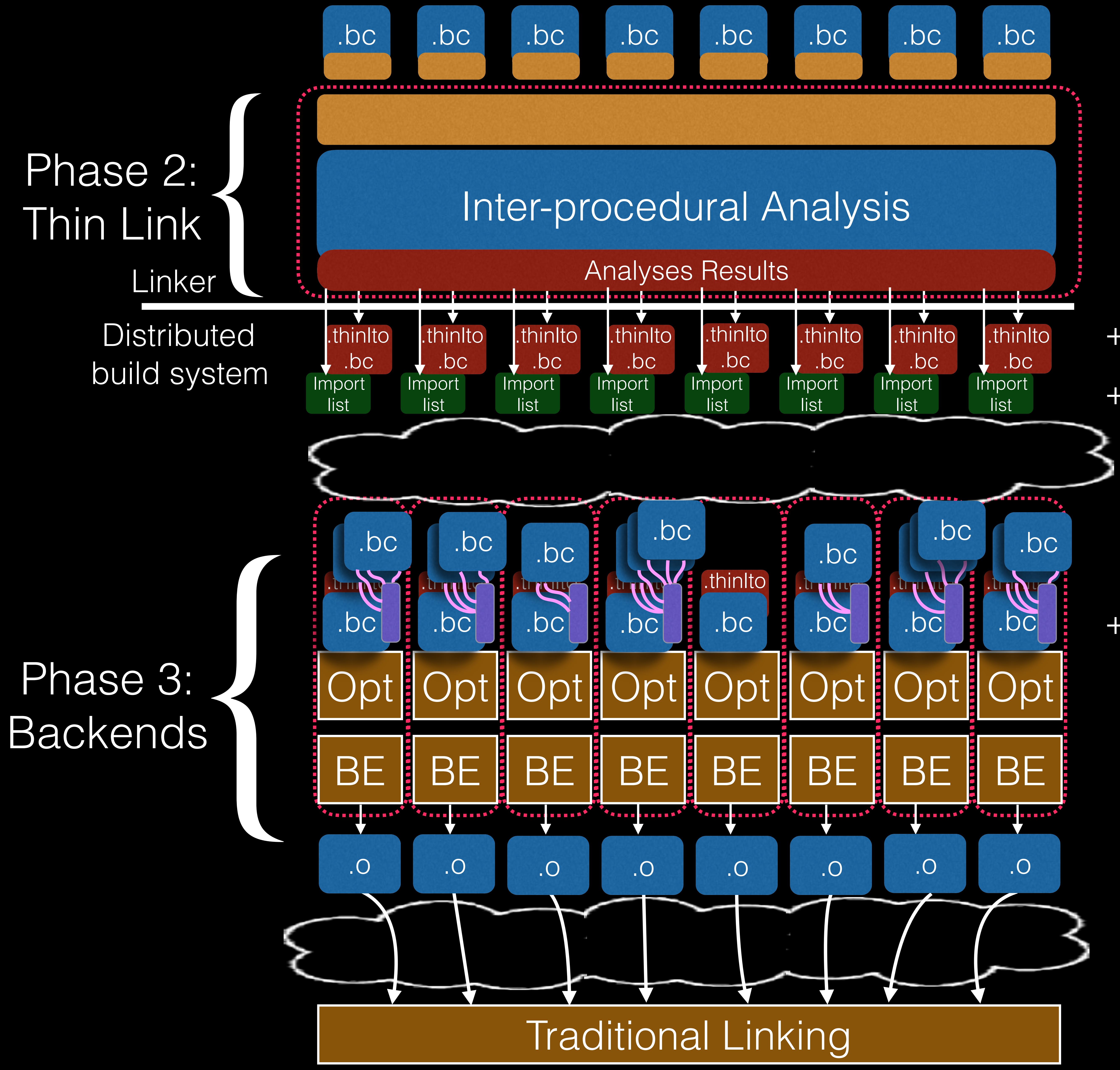
+ Stream out analysis results for each backend job
+ Produce dependent module lists: the build system bundles these for each job
+ ***Distributed*** Inter-procedural transformations based on the analyses results

ThinLTO: Distributed Builds



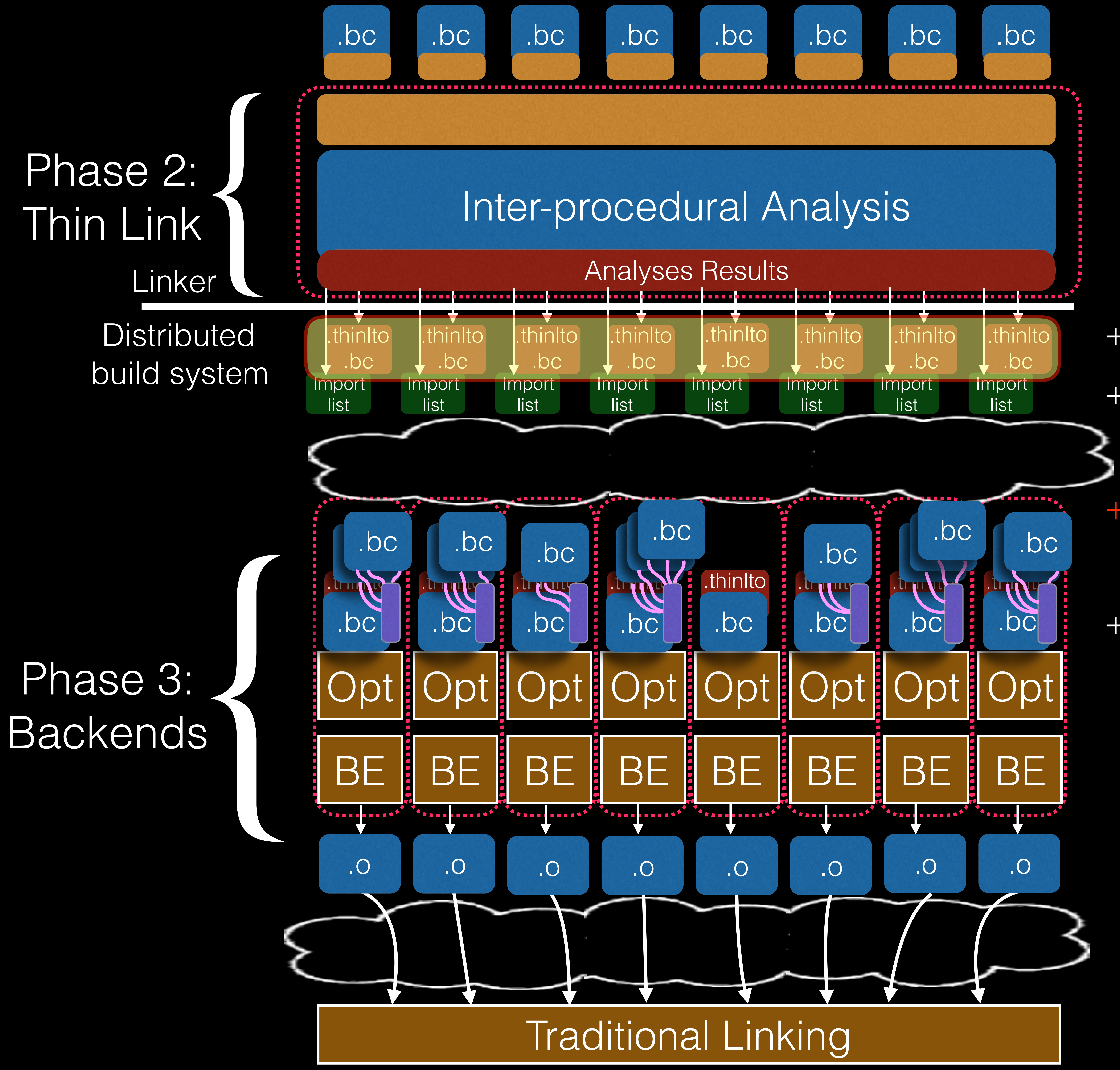
+ Stream out analysis results for each backend job
+ Produce dependent module lists: the build system bundles these for each job
+ ***Distributed*** Inter-procedural transformations based on the analyses results

ThinLTO: Distributed Builds



+ Stream out analysis results for each backend job
+ Produce dependent module lists:
  the build system bundles these for each job
+ ***Distributed*** Inter-procedural transformations
  based on the analyses results

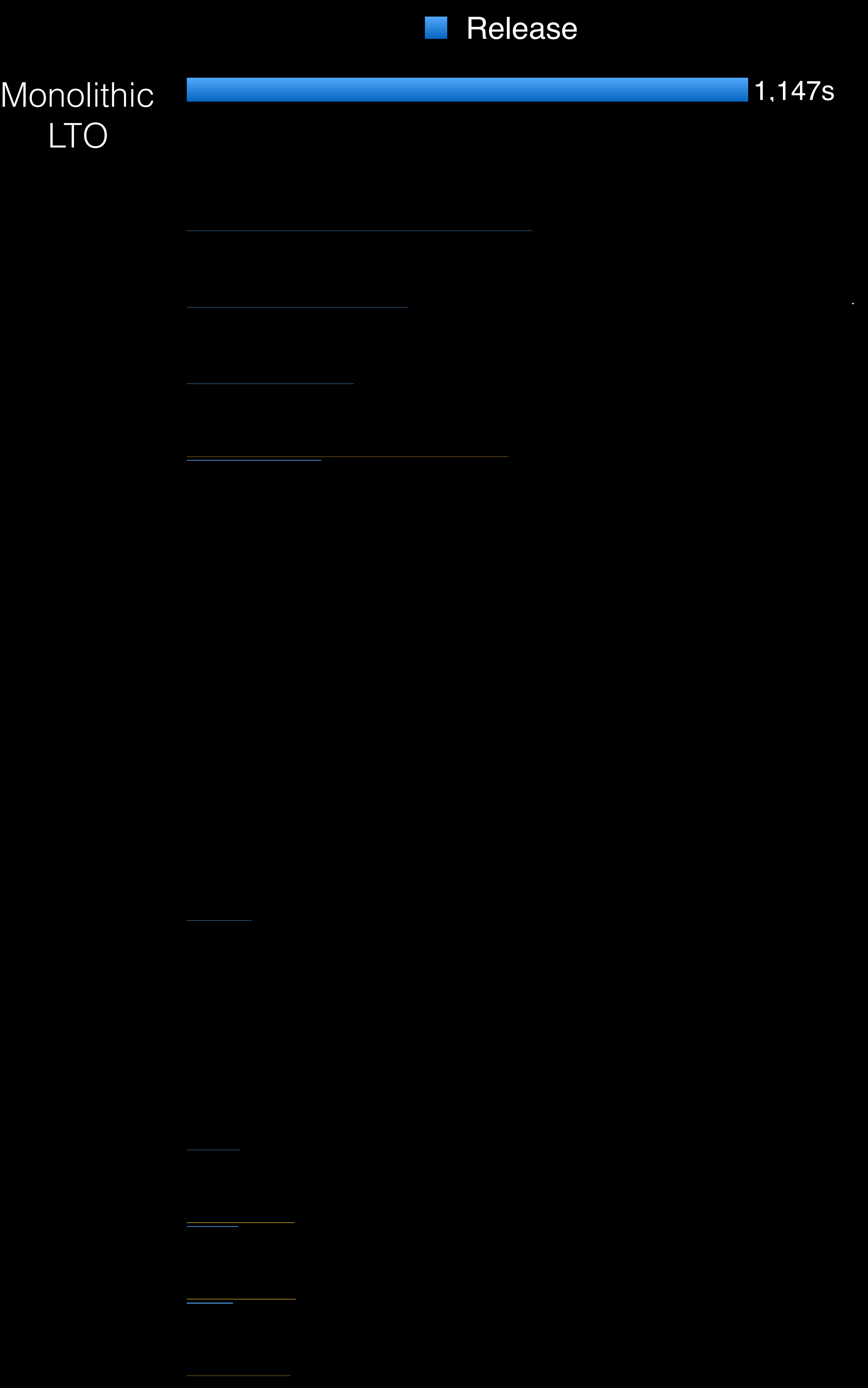
ThinLTO: Distributed Builds



+ Stream out analysis results for each backend job
+ Produce dependent module lists: the build system bundles these for each job
+ Individual index files encapsulate info necessary for incremental build decisions
+ ***Distributed*** Inter-procedural transformations based on the analyses results

Performance: Link Time for Clang

Performance: Link Time for Clang



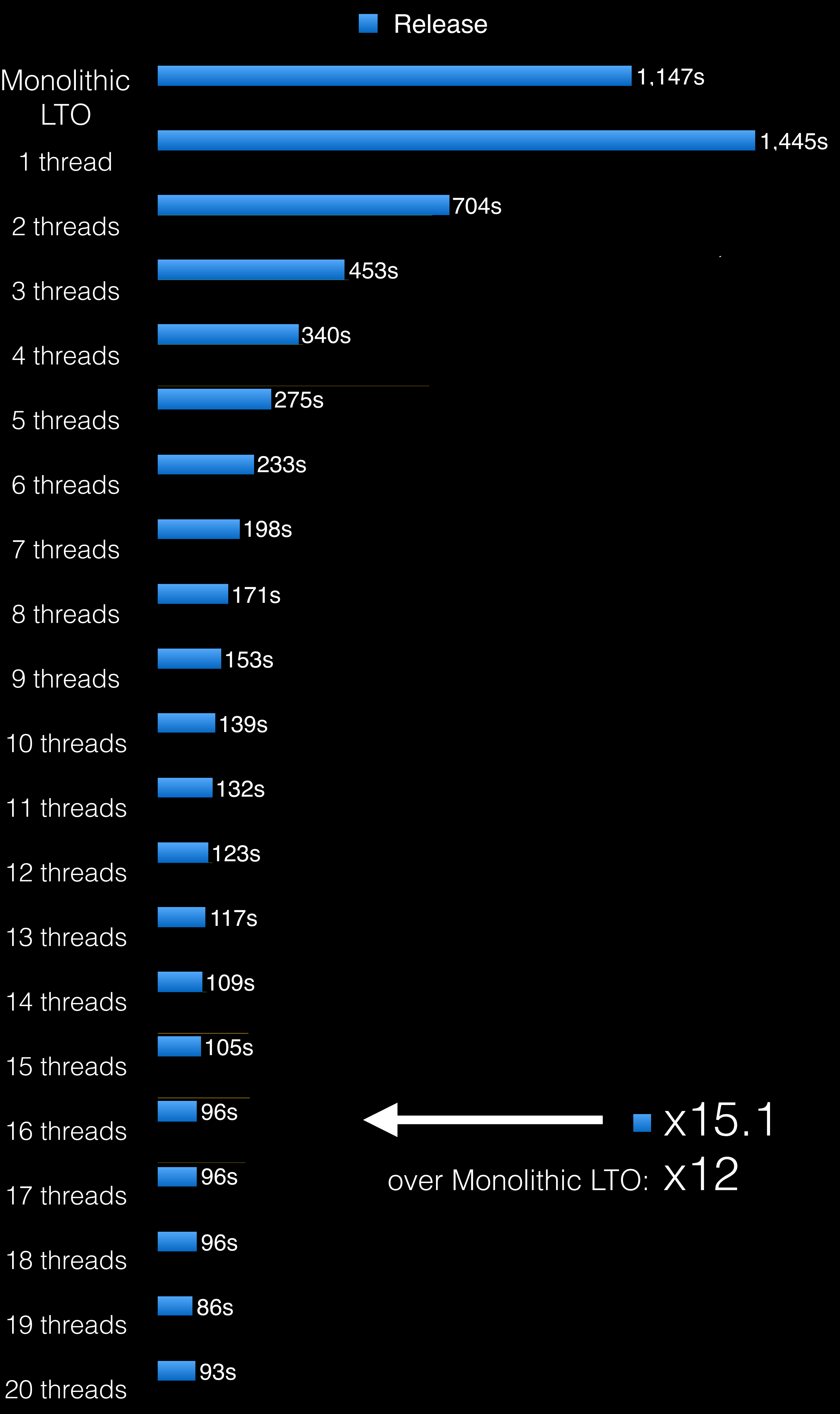
Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

Performance: Link Time for Clang

Half is the optimization pipeline
Other half is the ThinLTO overhead

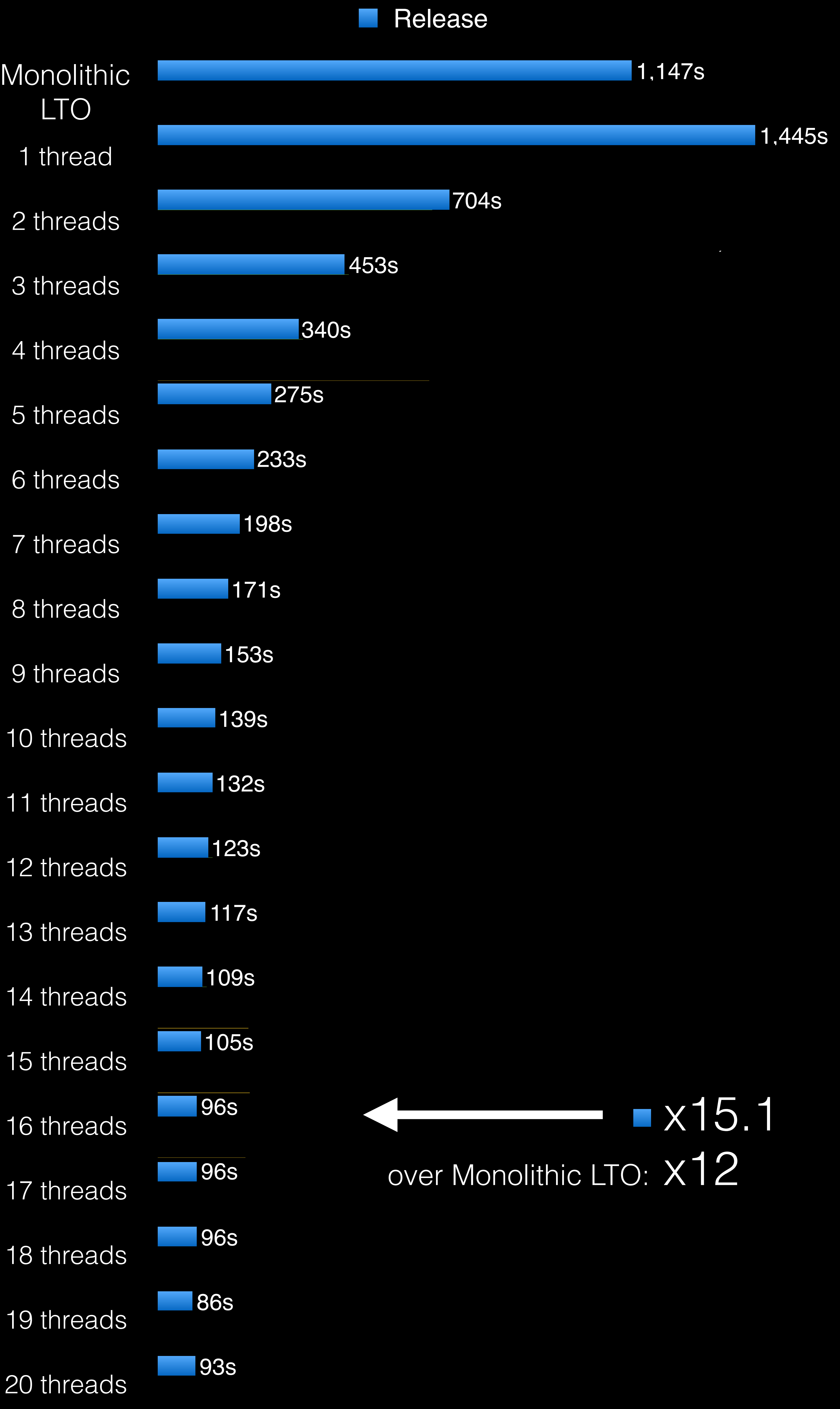
Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

Performance: Link Time for Clang



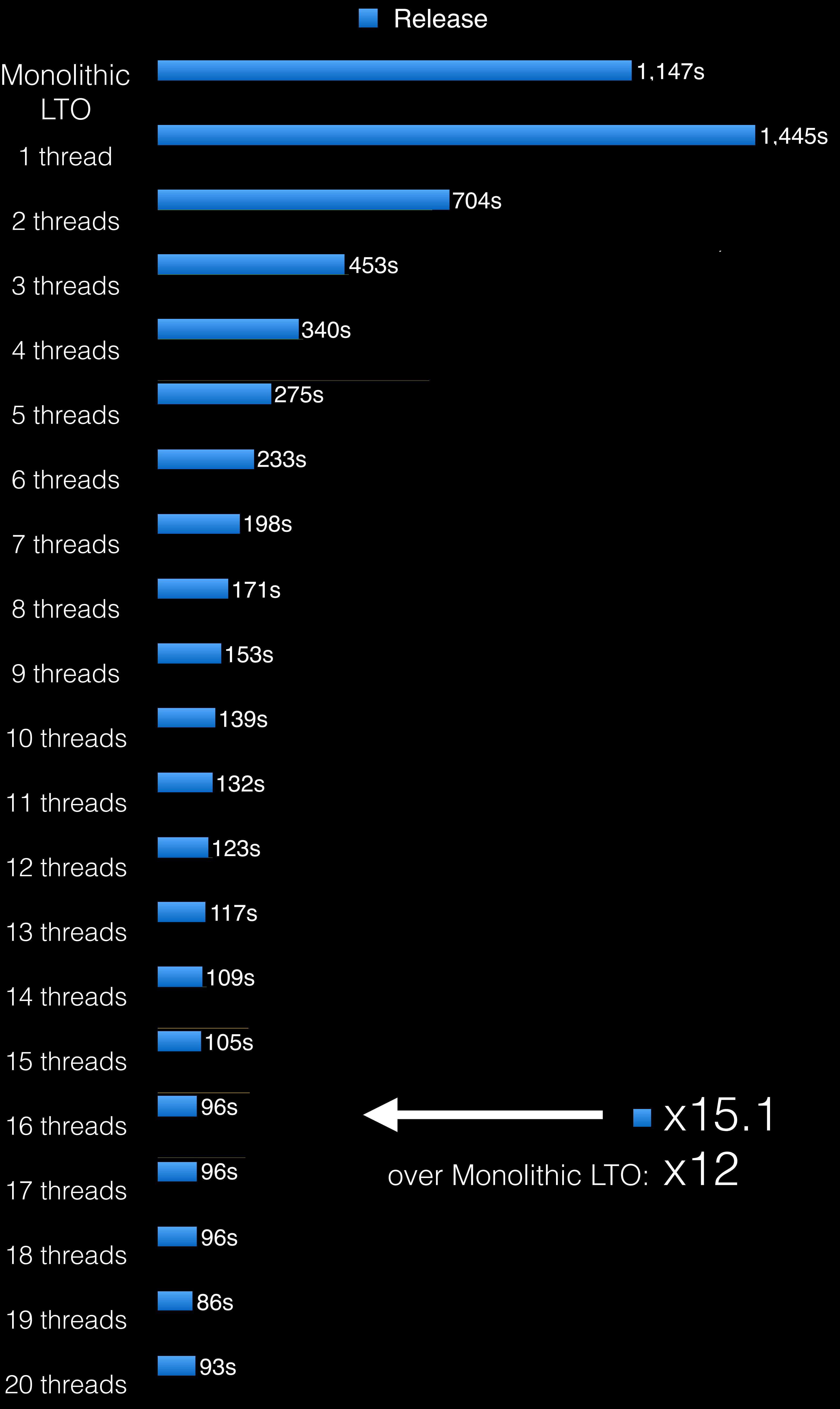
Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

Performance: Link Time for Clang



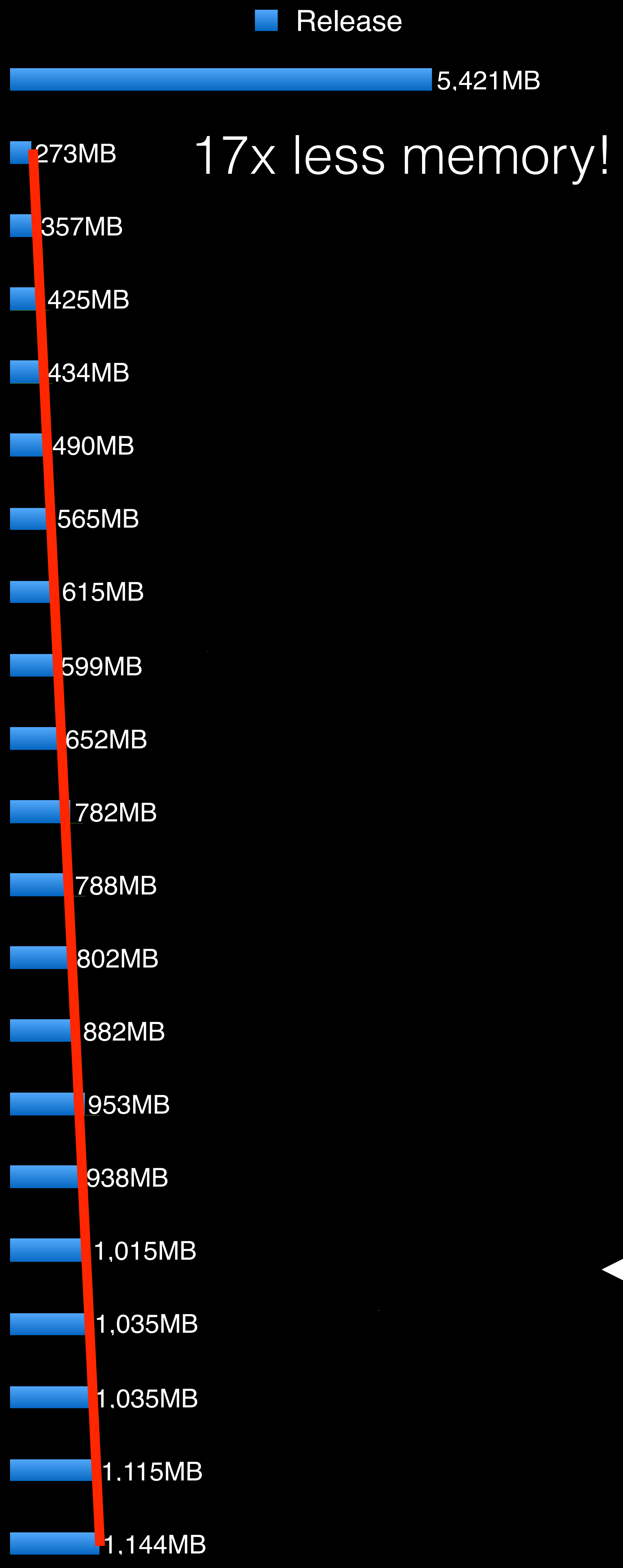
Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

Performance: Link Time for Clang



x15.1

over Monolithic LTO: x12



41MB per thread

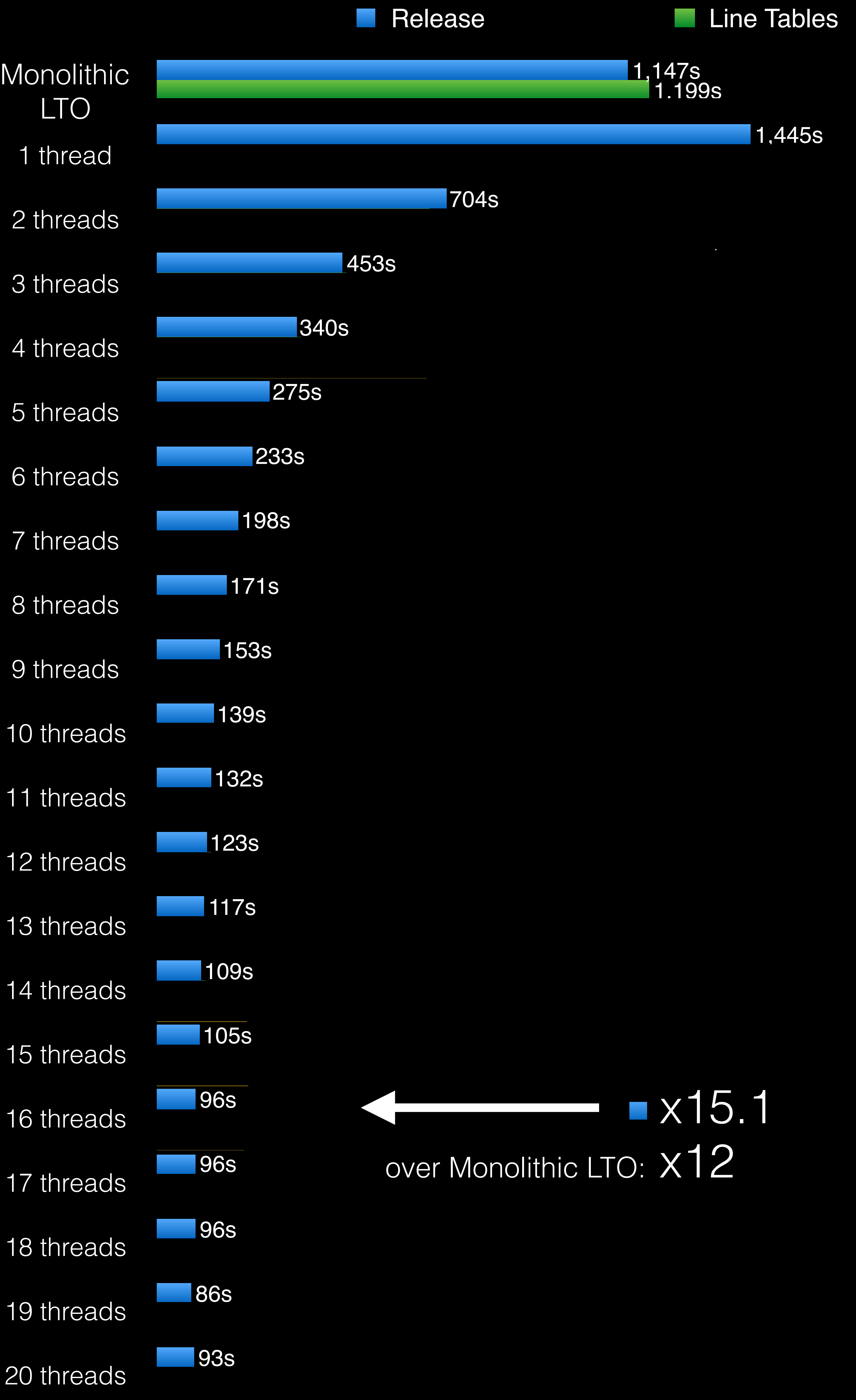
over Monolithic LTO:

÷5.3

Linux 16-core (32-logical)

Intel Xeon E5-2690 @ 2.90GHz

Performance: Link Time for Clang



- x15.1

over Monolithic LTO: x12

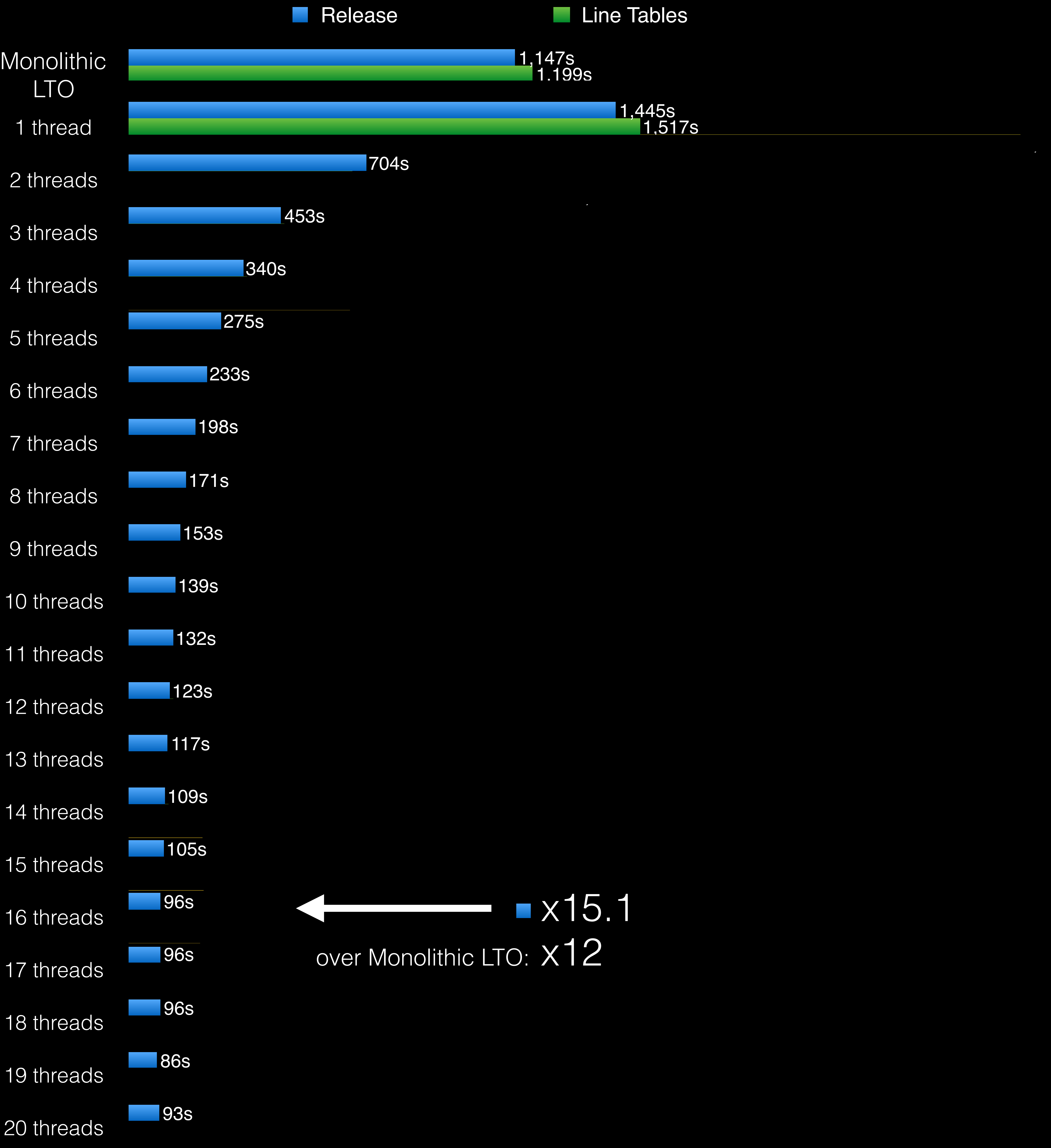
- 41MB per thread

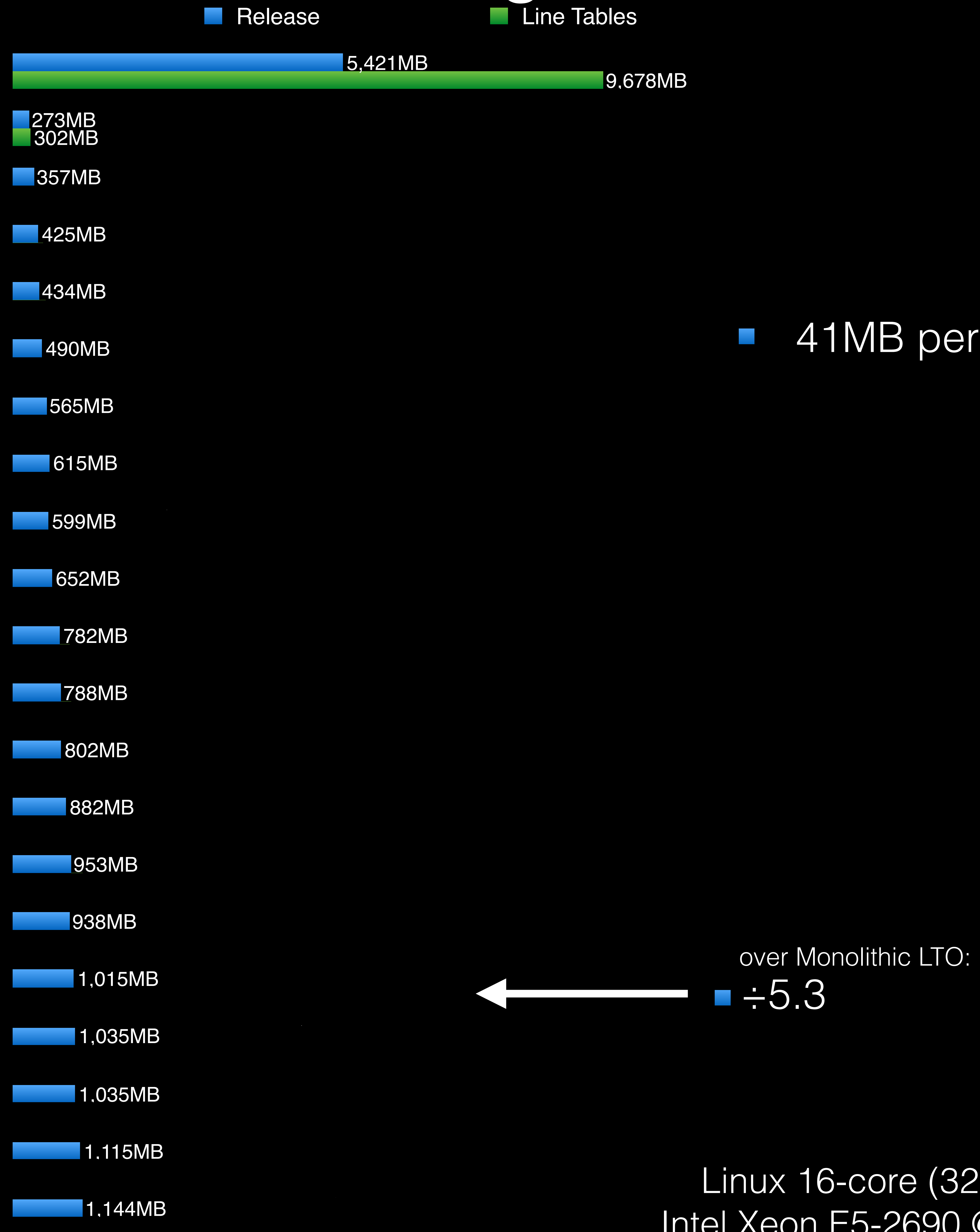
over Monolithic LTO:

- ÷5.3

Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

Performance: Link Time for Clang

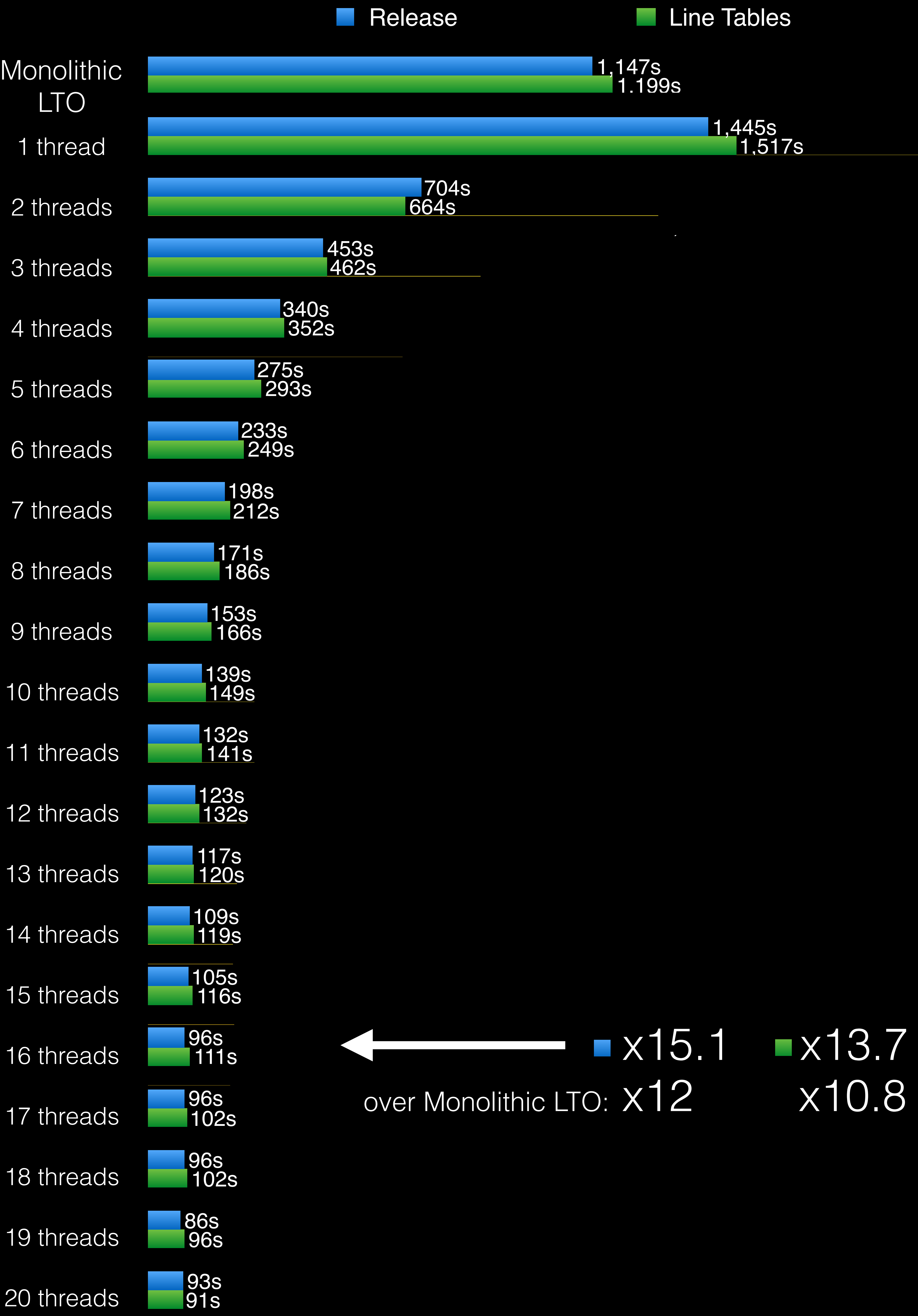


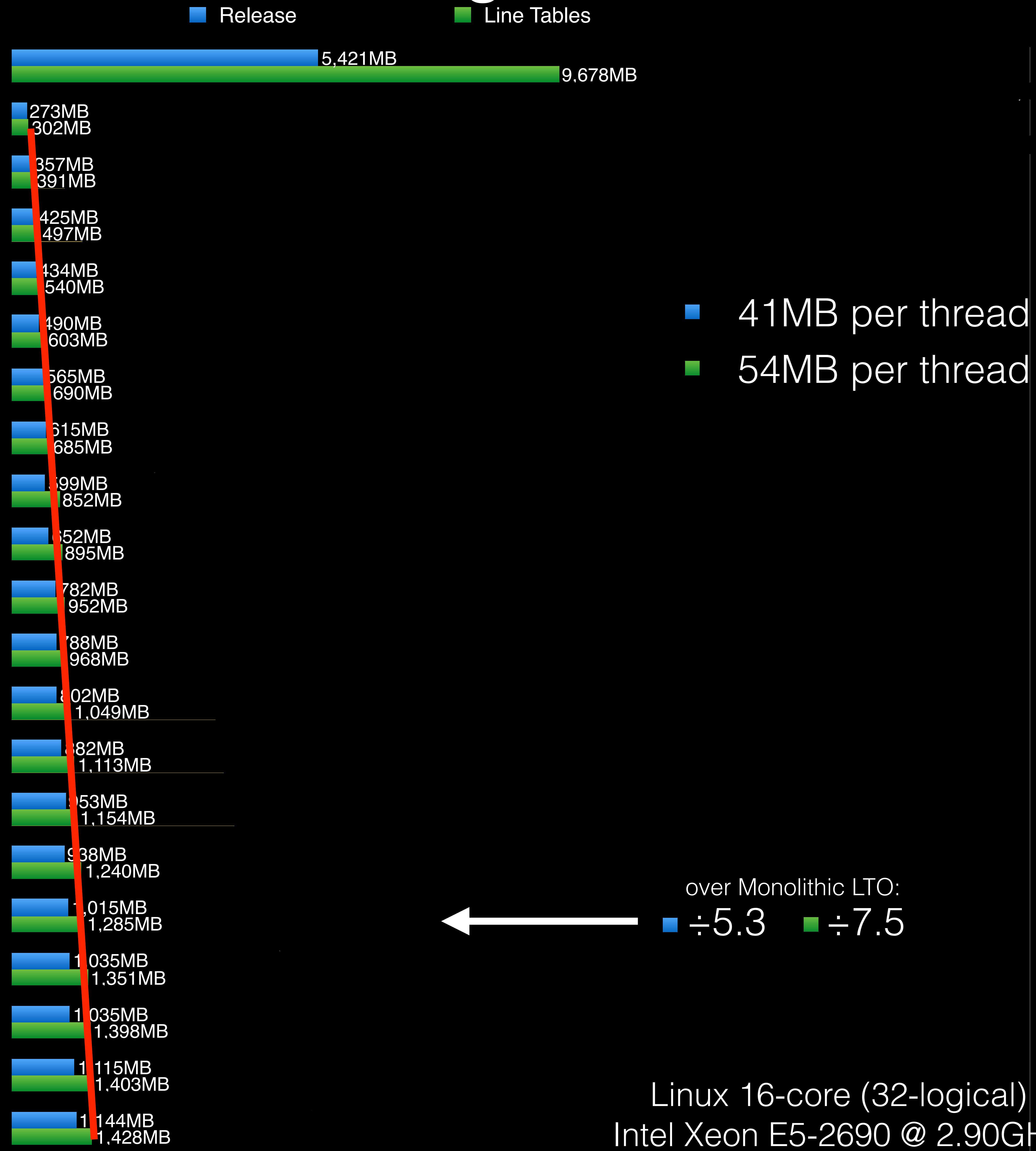


- 41MB per thread

Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

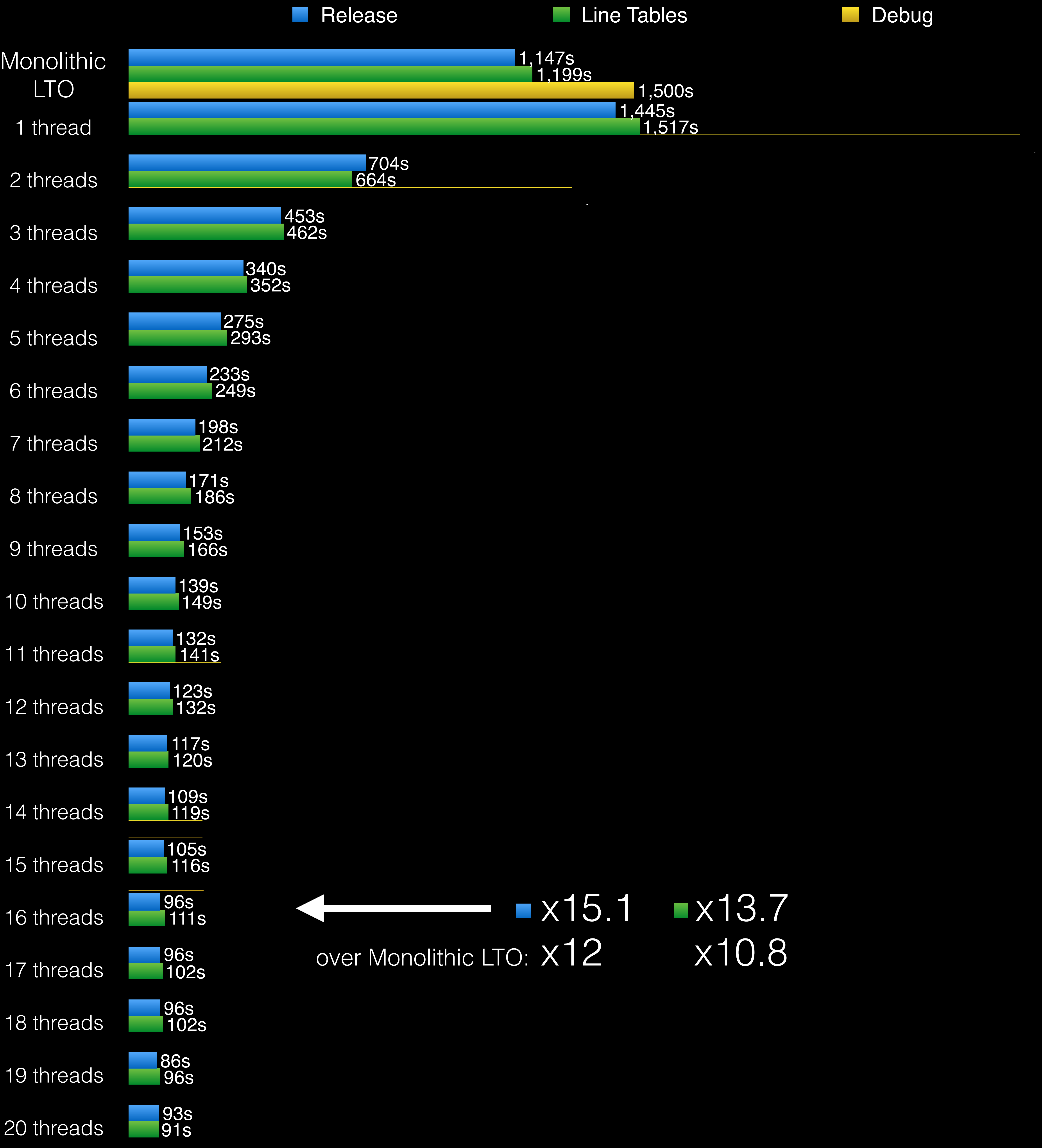
Performance: Link Time for Clang

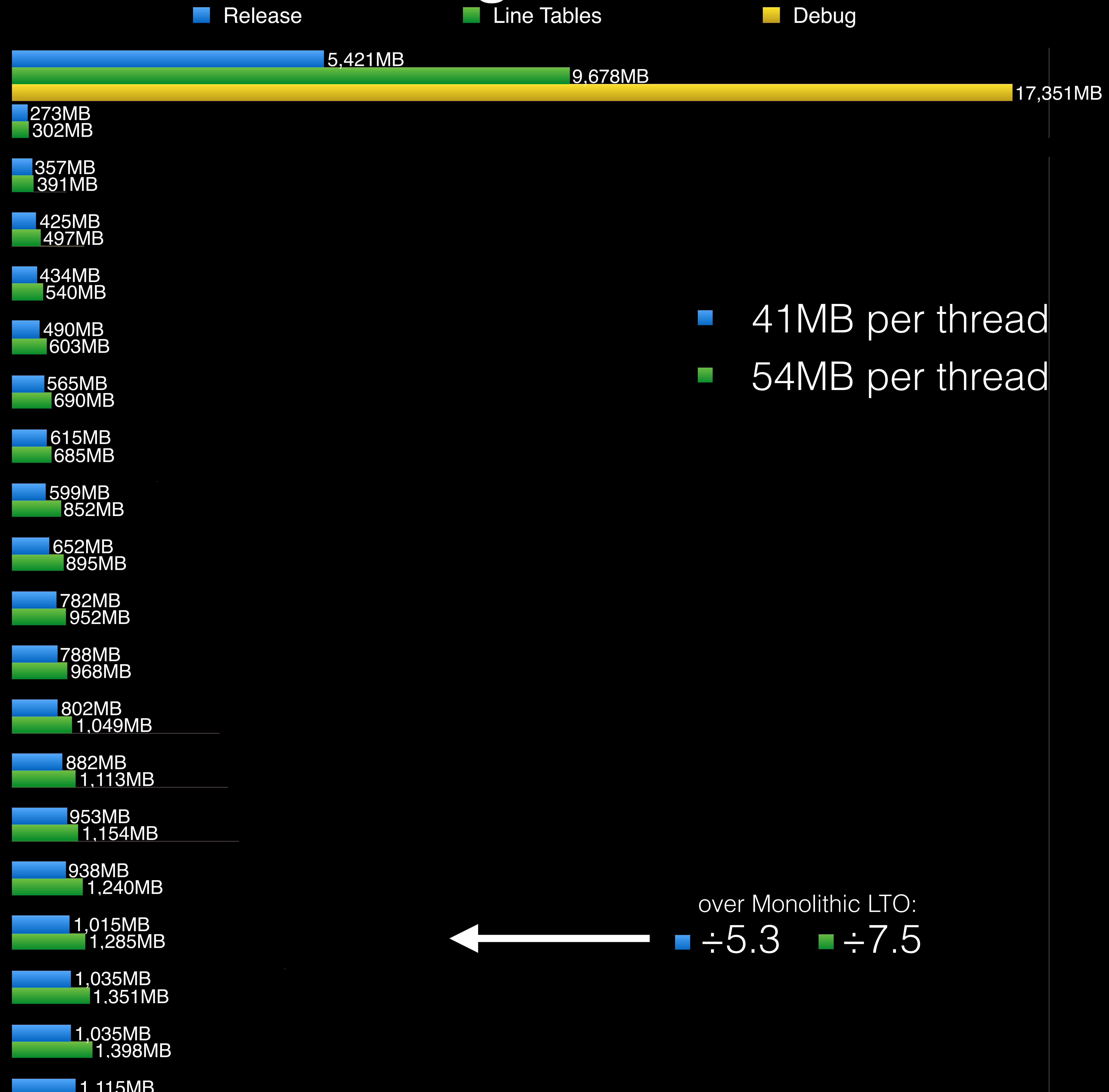




Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

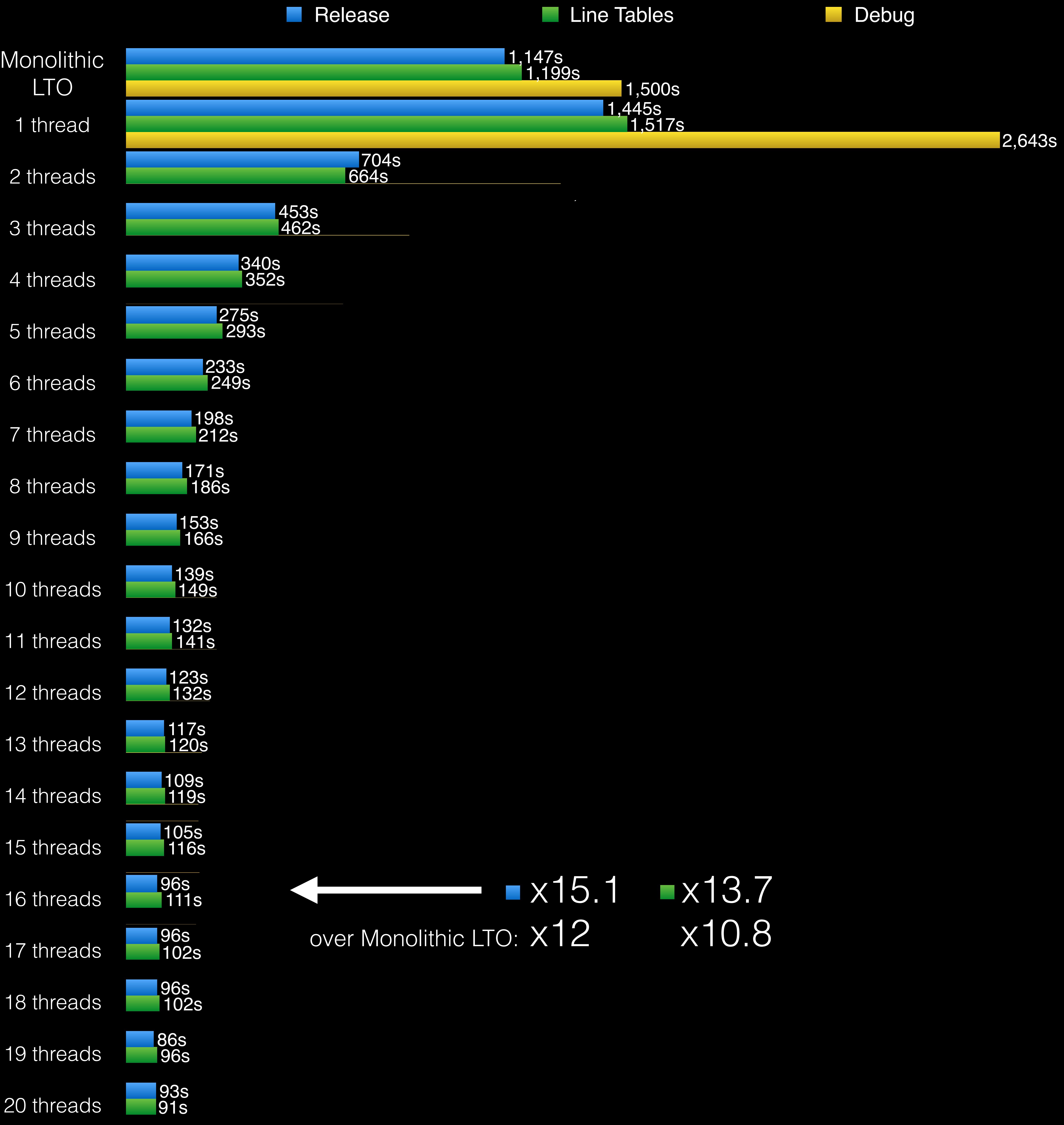
Performance: Link Time for Clang

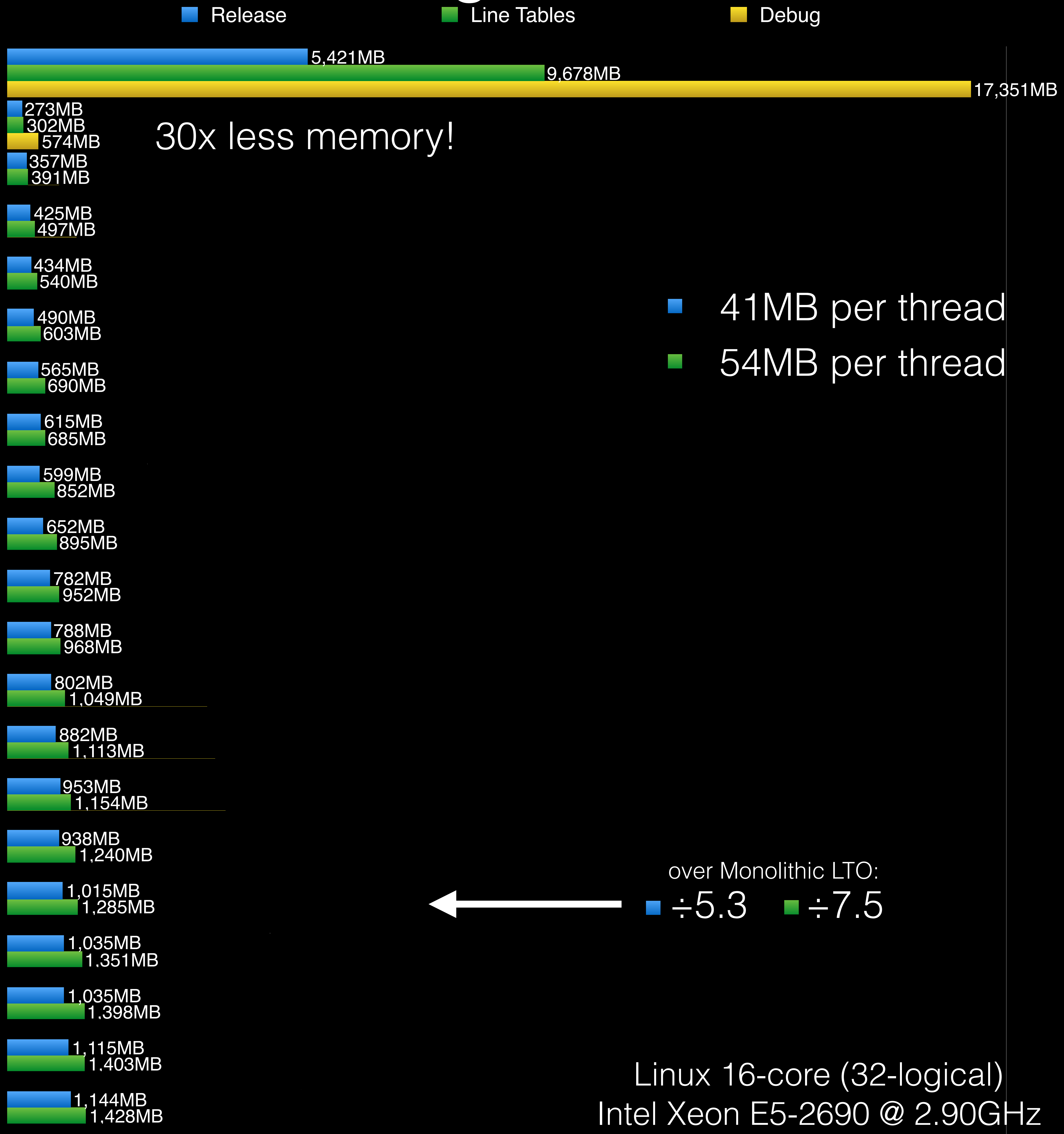




Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

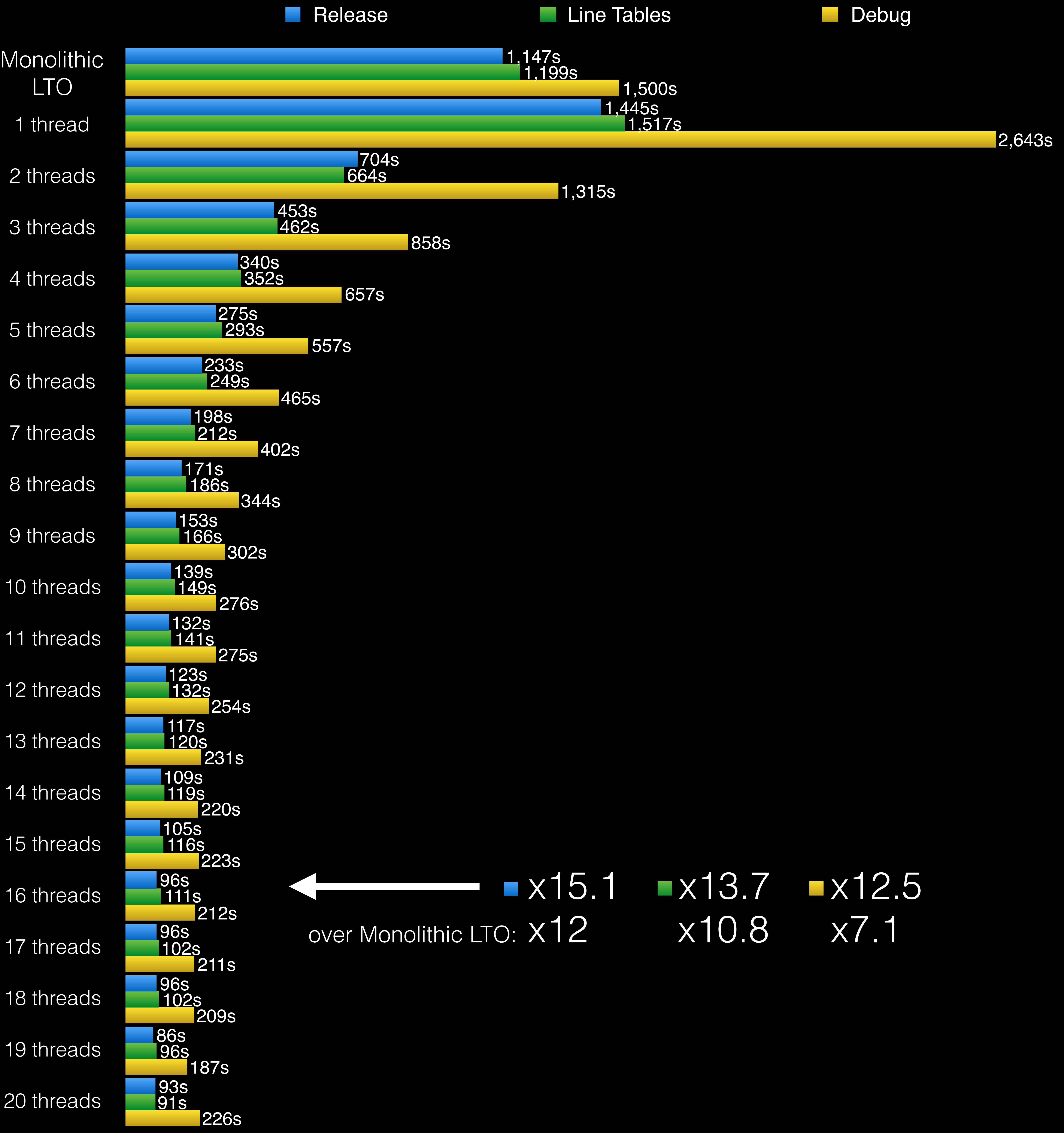
Performance: Link Time for Clang

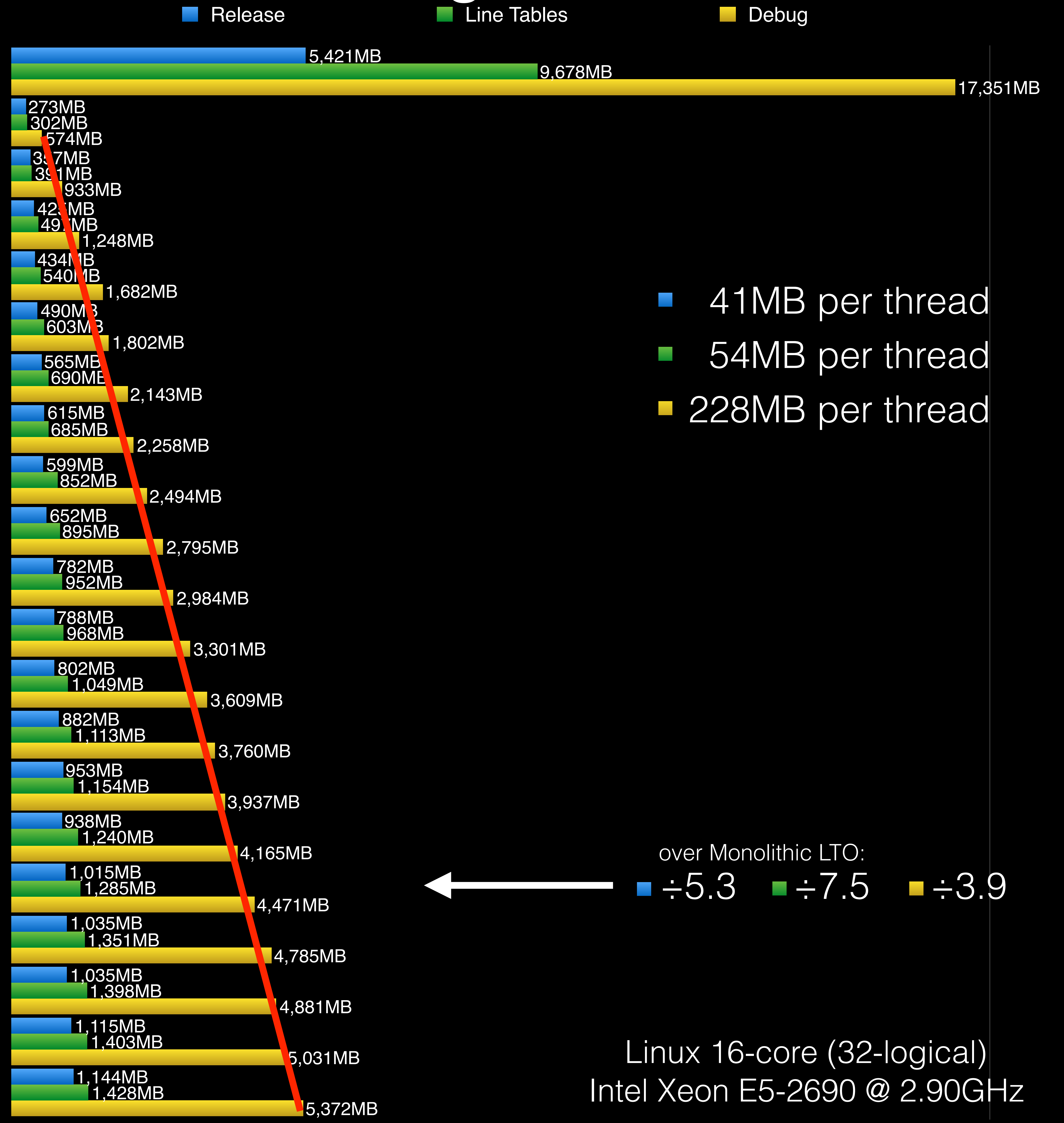




Linux 16-core (32-logical)
Intel Xeon E5-2690 @ 2.90GHz

Performance: Link Time for Clang





Performance: Link Time for Clang

Comparison with GCC LTO

Comparison with GCC LTO

GCC has a mature and well-tuned sophisticated LTO implementation (WHOPR), with two parts:

Comparison with GCC LTO

GCC has a mature and well-tuned sophisticated LTO implementation (WHOPR), with two parts:

1. WPA: Serial part that makes IPA and inlining decisions, rewrites partitioned IR

Comparison with GCC LTO

GCC has a mature and well-tuned sophisticated LTO implementation (WHOPR), with two parts:

1. WPA: Serial part that makes IPA and inlining decisions, rewrites partitioned IR

2. LTRANS: Parallel backends performing inlining within each partition, plus usual optimizations and code generation

Comparison with GCC LTO

GCC has a mature and well-tuned sophisticated LTO implementation (WHOPR), with two parts:

1. WPA: Serial part that makes IPA and inlining decisions, rewrites partitioned IR
    ➡ *Comparable Phase 2: Thin Link (both serial)*

2. LTRANS: Parallel backends performing inlining within each partition, plus usual optimizations and code generation
    ➡ *Comparable to Phase 3: ThinLTO Backends (both parallel)*

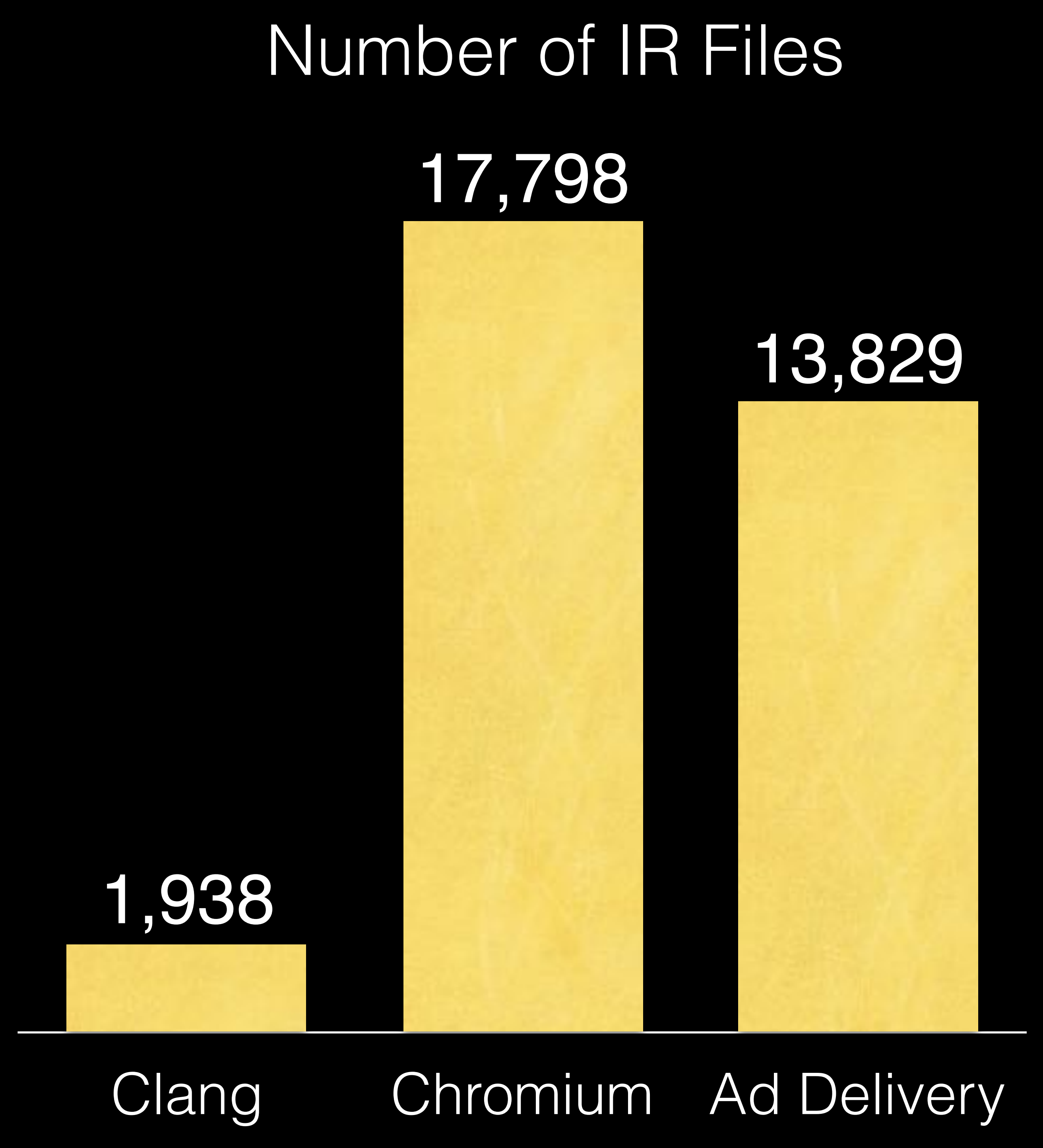
Scaling with the Input Size

Scaling with the Input Size

- LLVM/Clang - C/C++ Compiler
- Chromium - open-source web browser
- Ad Delivery - internal Google datacenter application

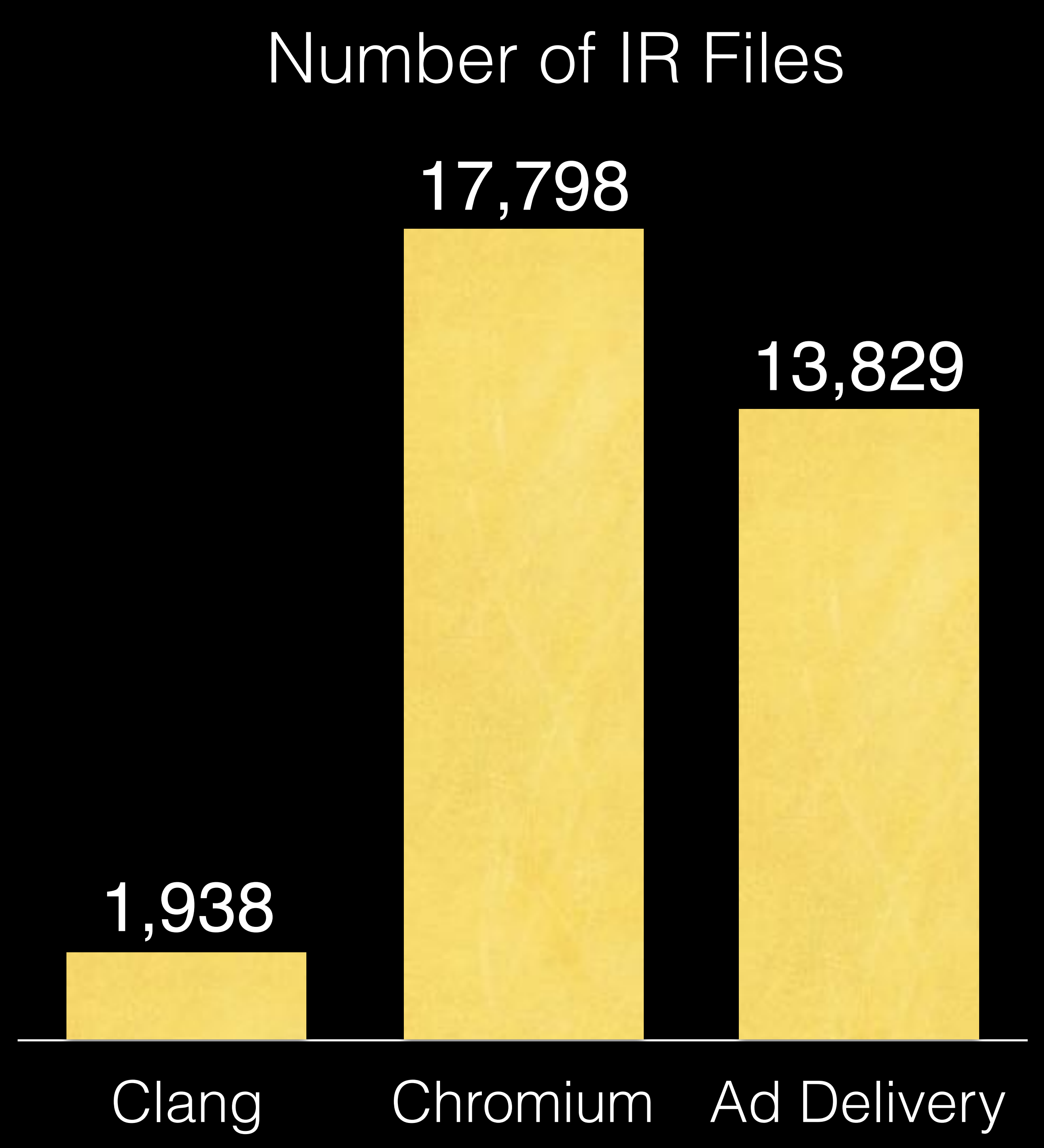
Scaling with the Input Size

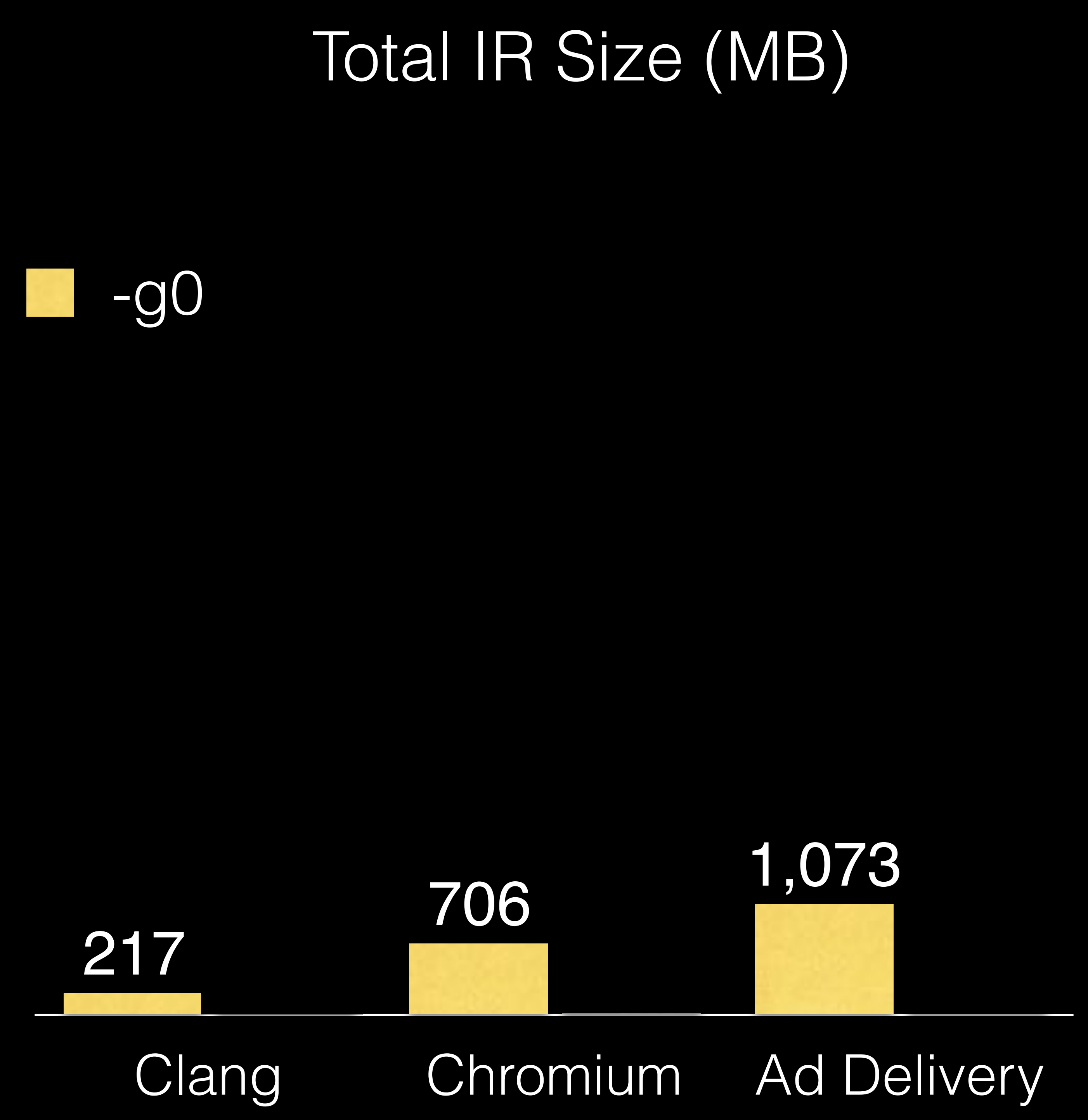
- LLVM/Clang - C/C++ Compiler
- Chromium - open-source web browser
- Ad Delivery - internal Google datacenter application



Scaling with the Input Size

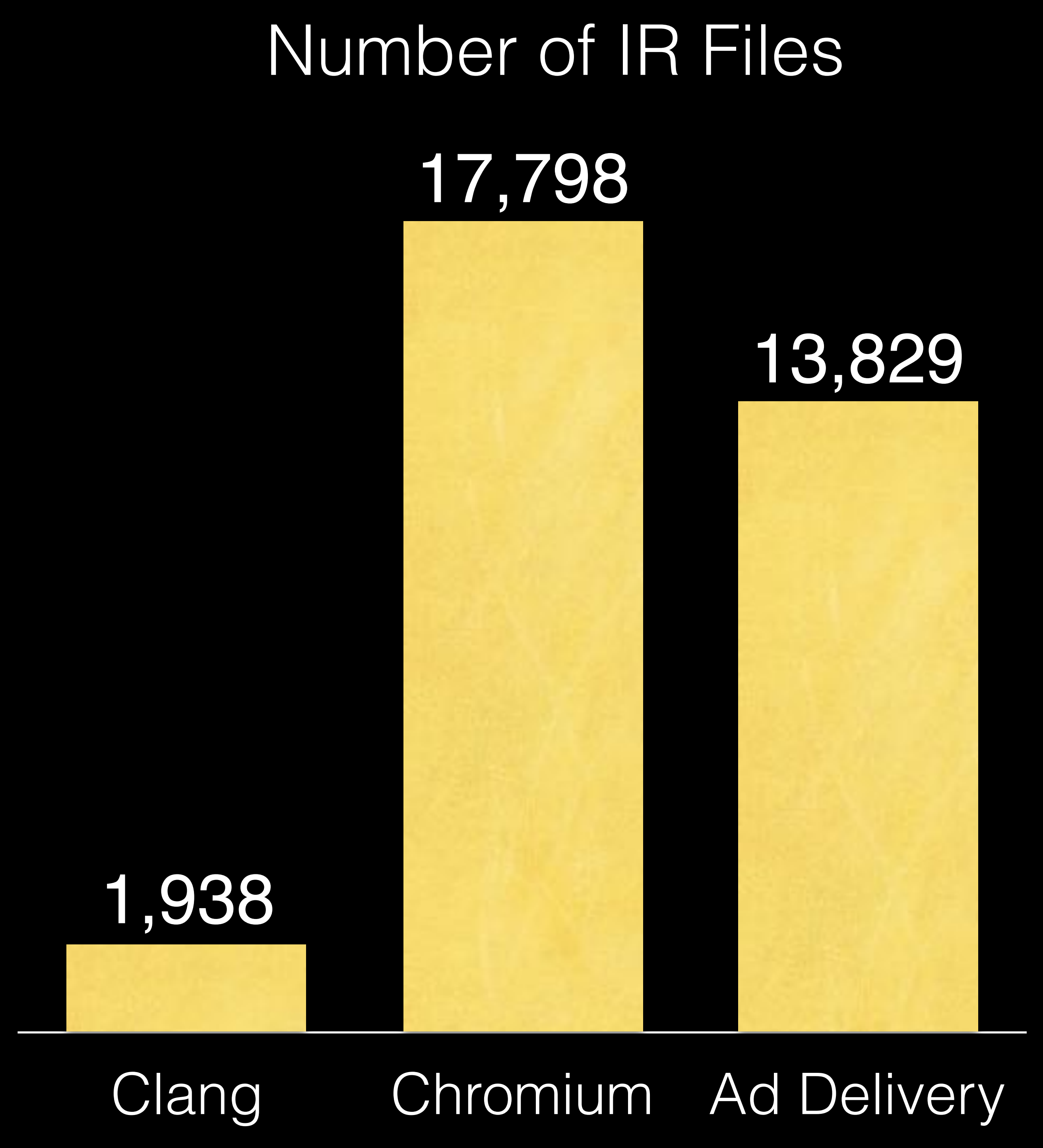
- LLVM/Clang - C/C++ Compiler
- Chromium - open-source web browser
- Ad Delivery - internal Google datacenter application

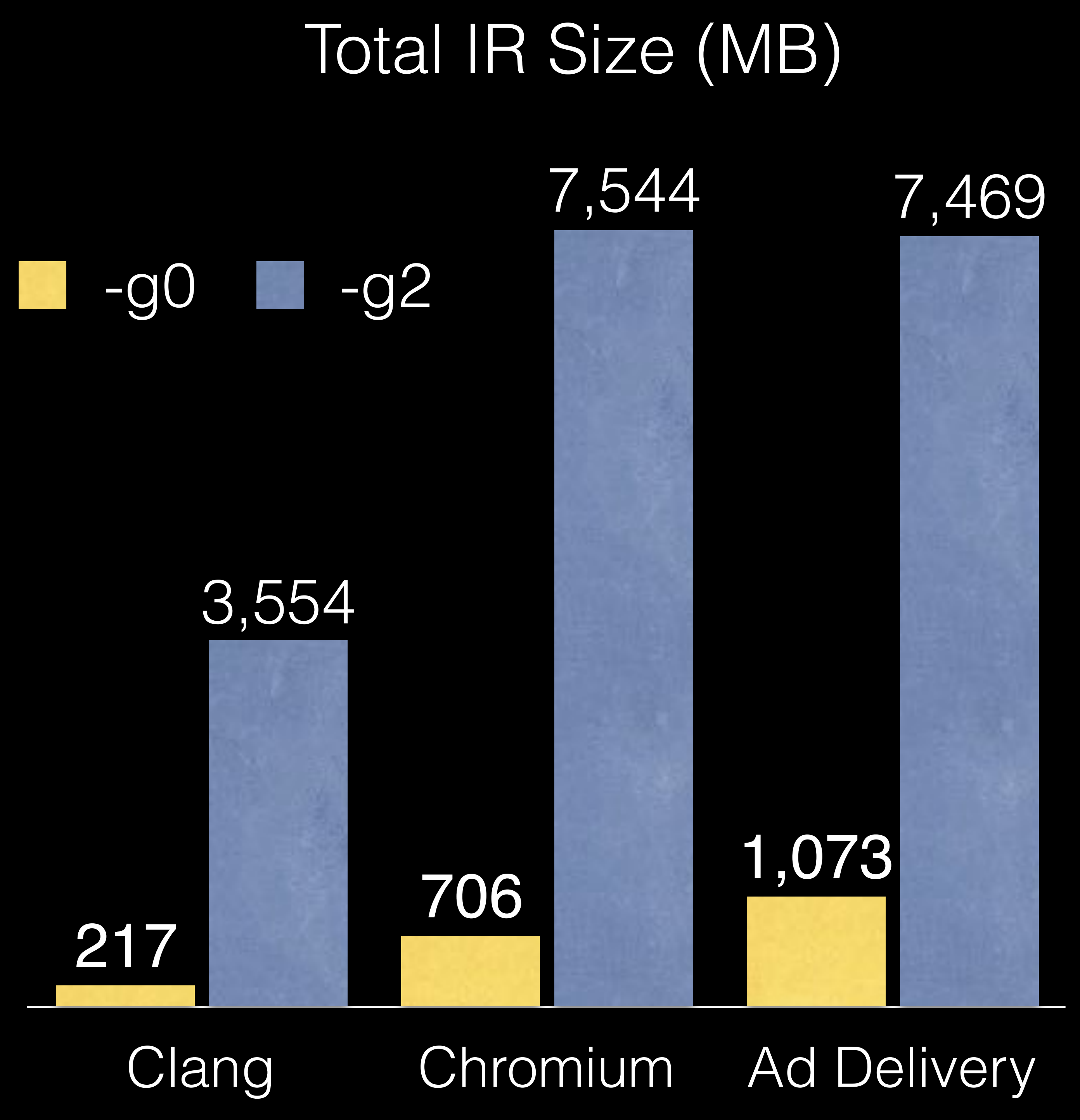




Scaling with the Input Size

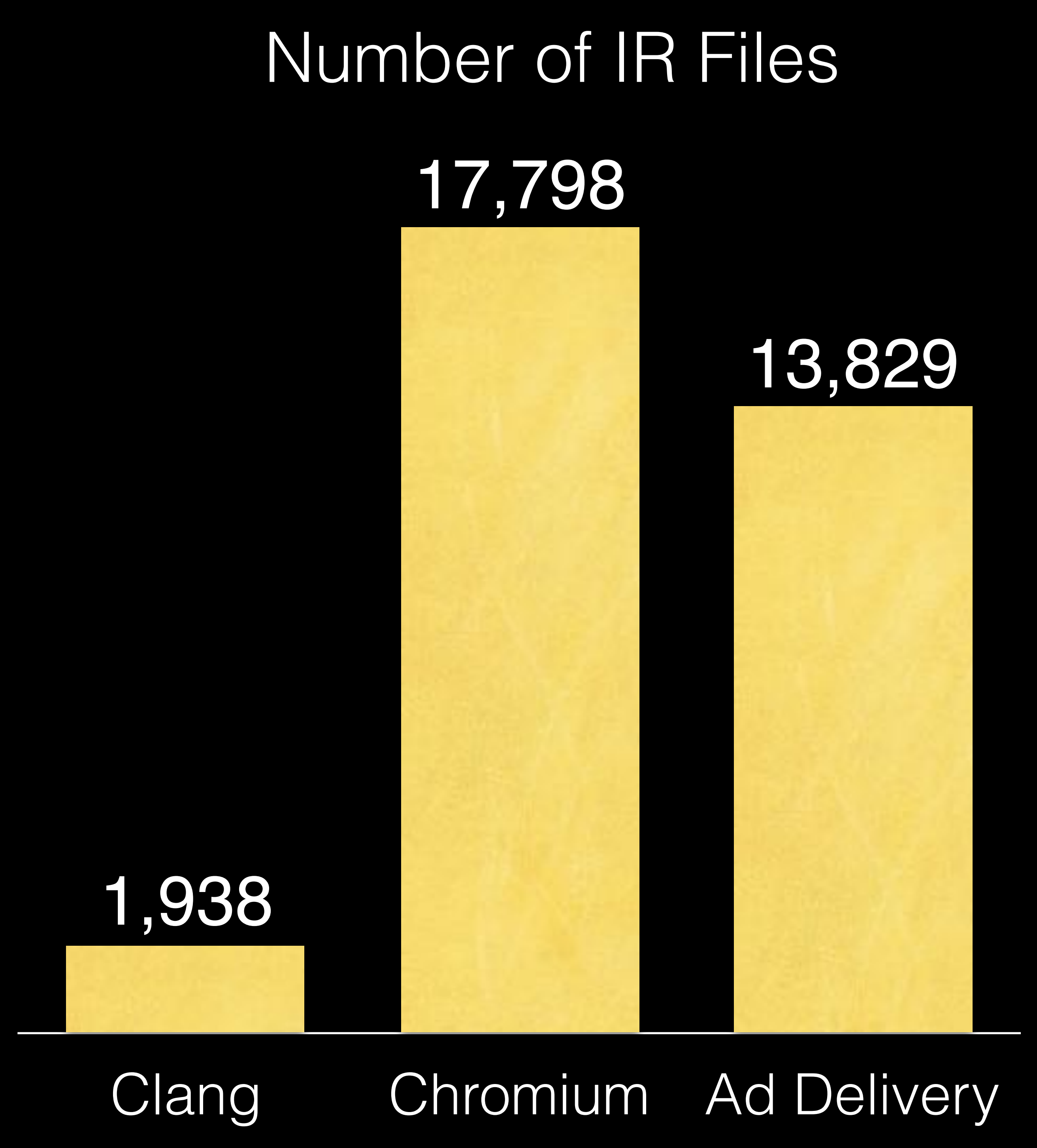
- LLVM/Clang - C/C++ Compiler
- Chromium - open-source web browser
- Ad Delivery - internal Google datacenter application

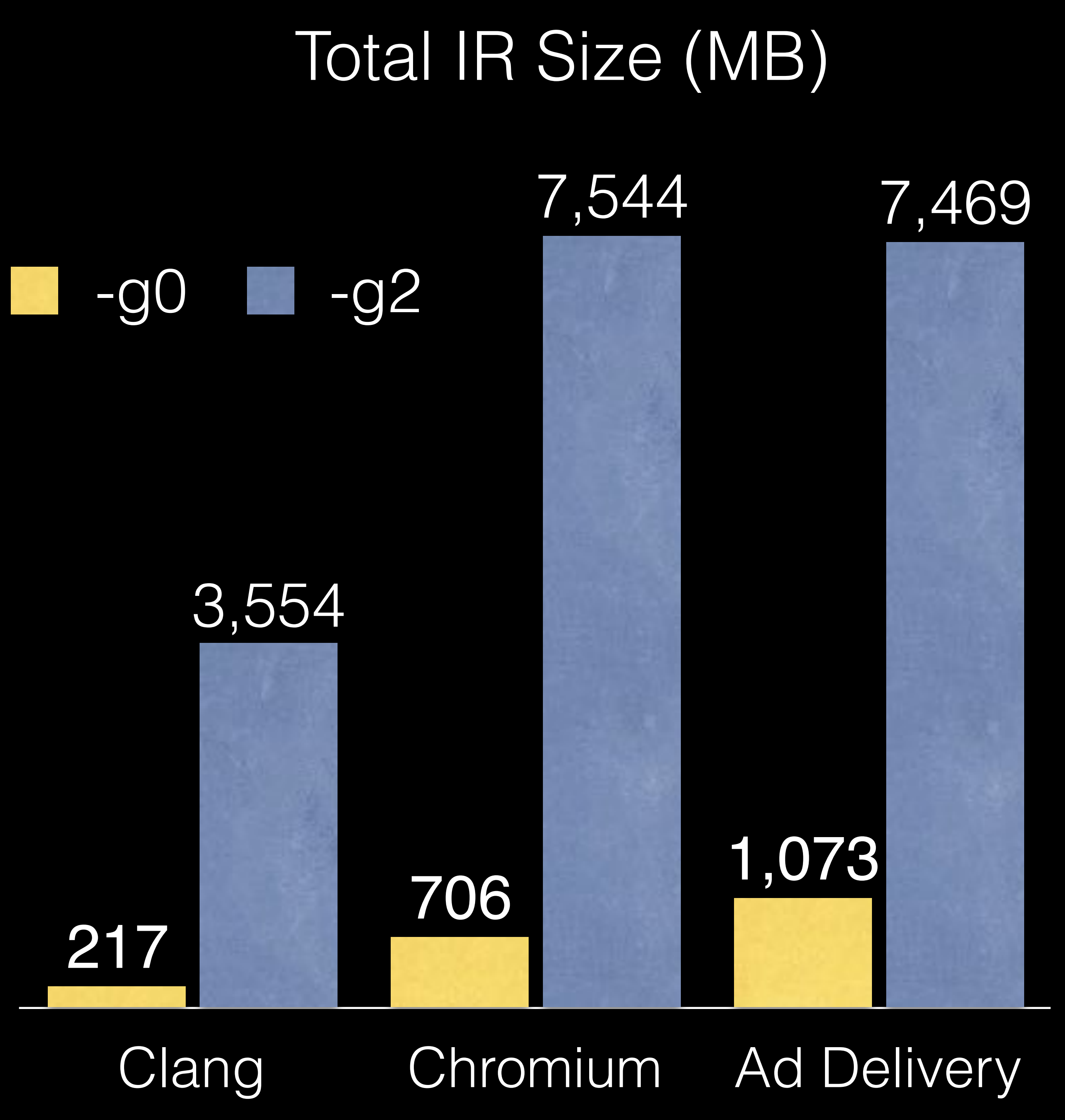


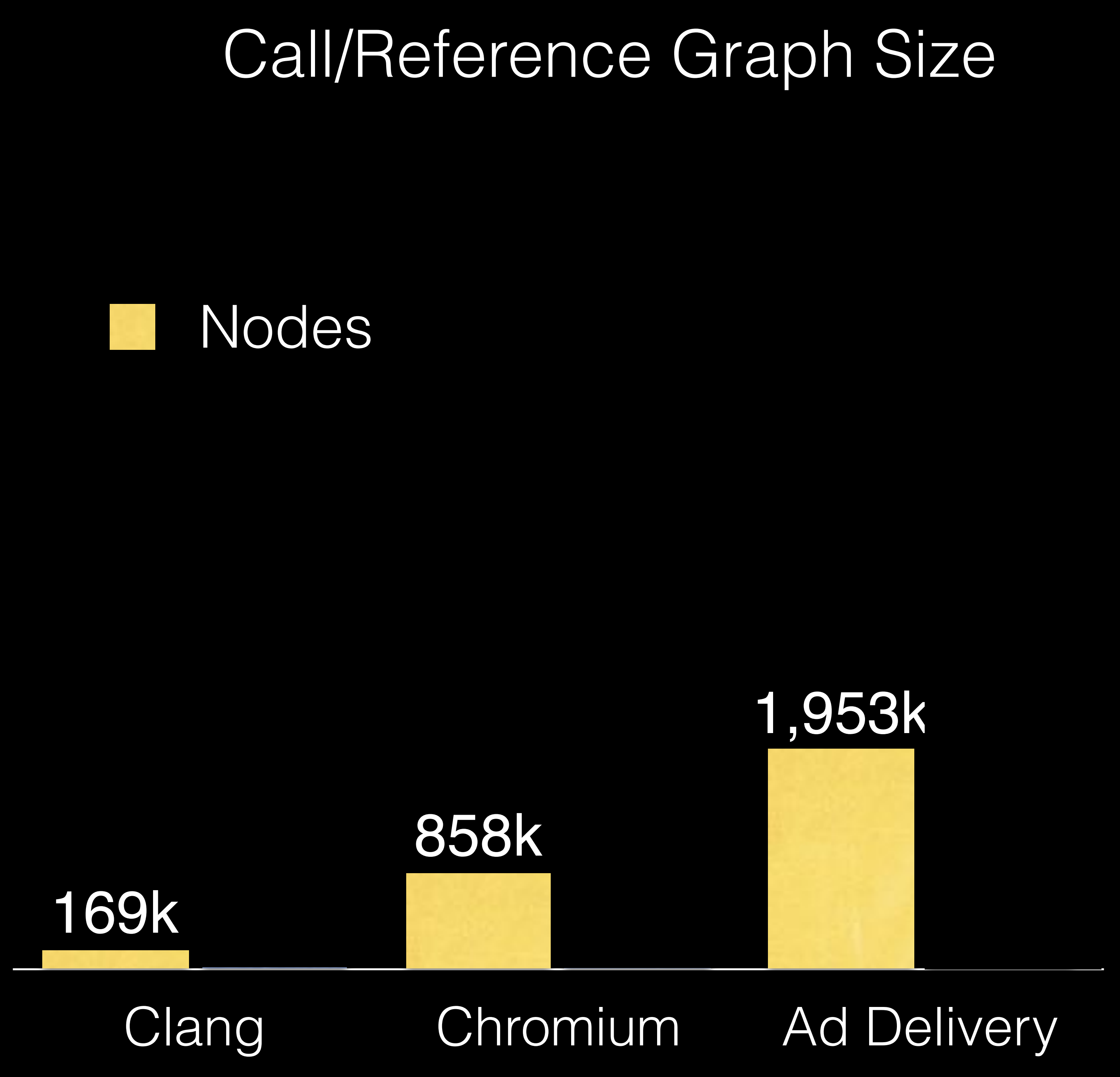


Scaling with the Input Size

- LLVM/Clang - C/C++ Compiler
- Chromium - open-source web browser
- Ad Delivery - internal Google datacenter application

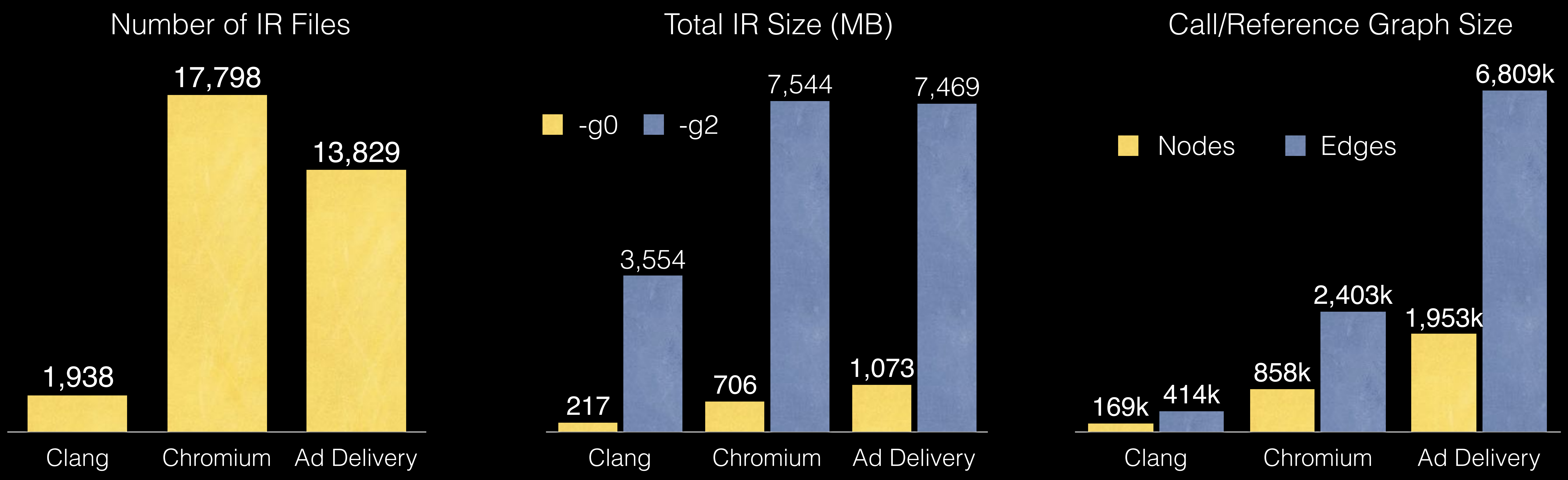






Scaling with the Input Size

- LLVM/Clang - C/C++ Compiler
- Chromium - open-source web browser
- Ad Delivery - internal Google datacenter application



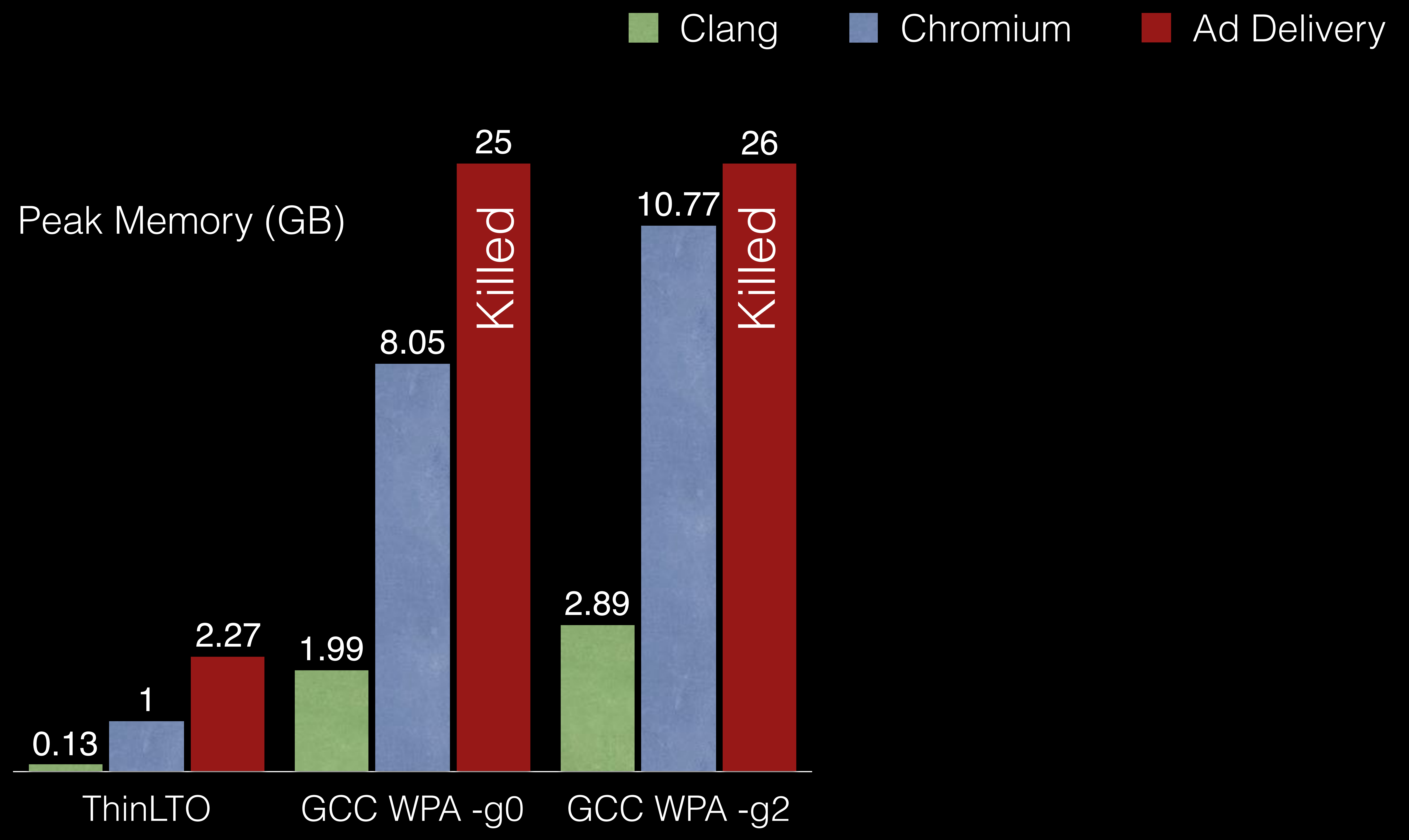
Serial Step Comparisons

Thin Link vs GCC WPA

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Serial Step Comparisons

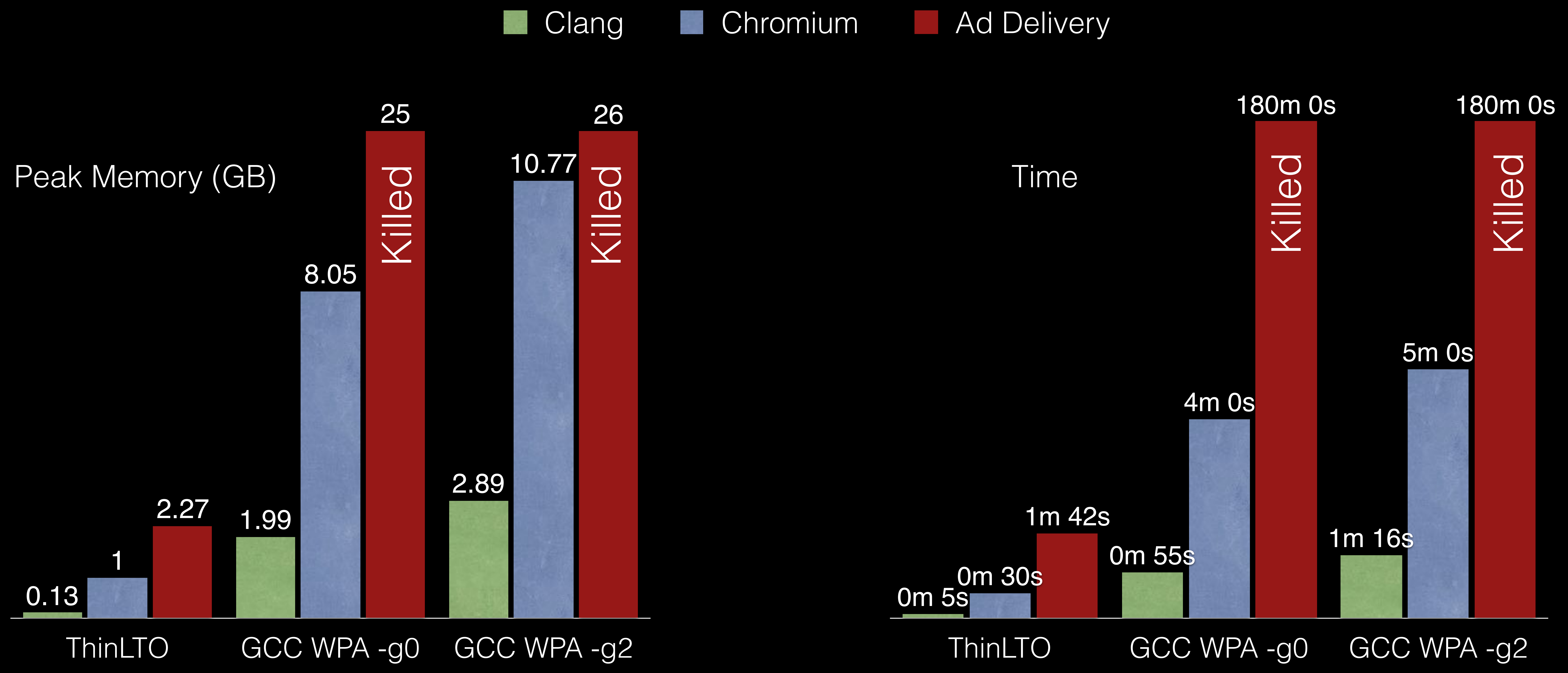
Thin Link vs GCC WPA



Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Serial Step Comparisons

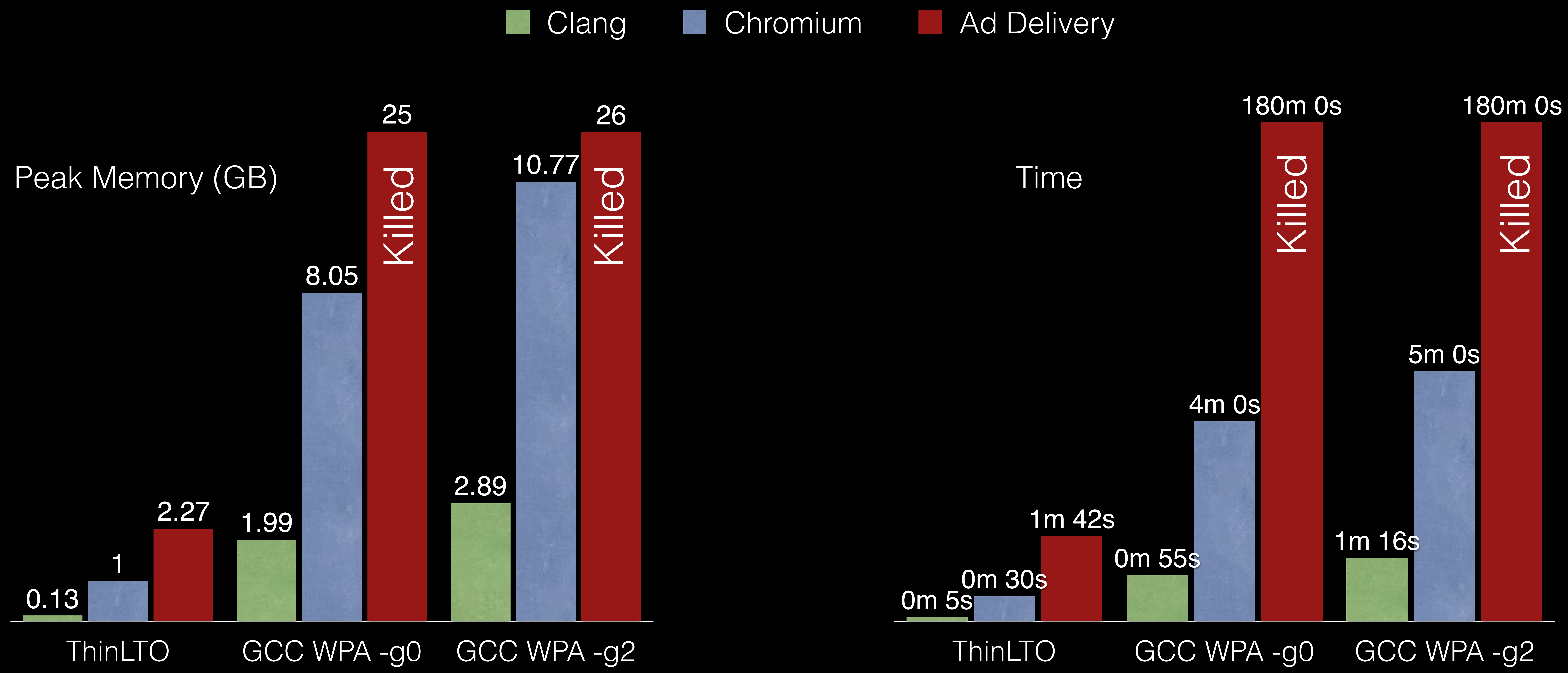
Thin Link vs GCC WPA



Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Serial Step Comparisons

Thin Link vs GCC WPA



LLVM LTO doesn't complete Ad Delivery even without debug (-g0), killed after 2 hours and > 12GB.

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Chromium Build Comparisons

No Debug (-g0)

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

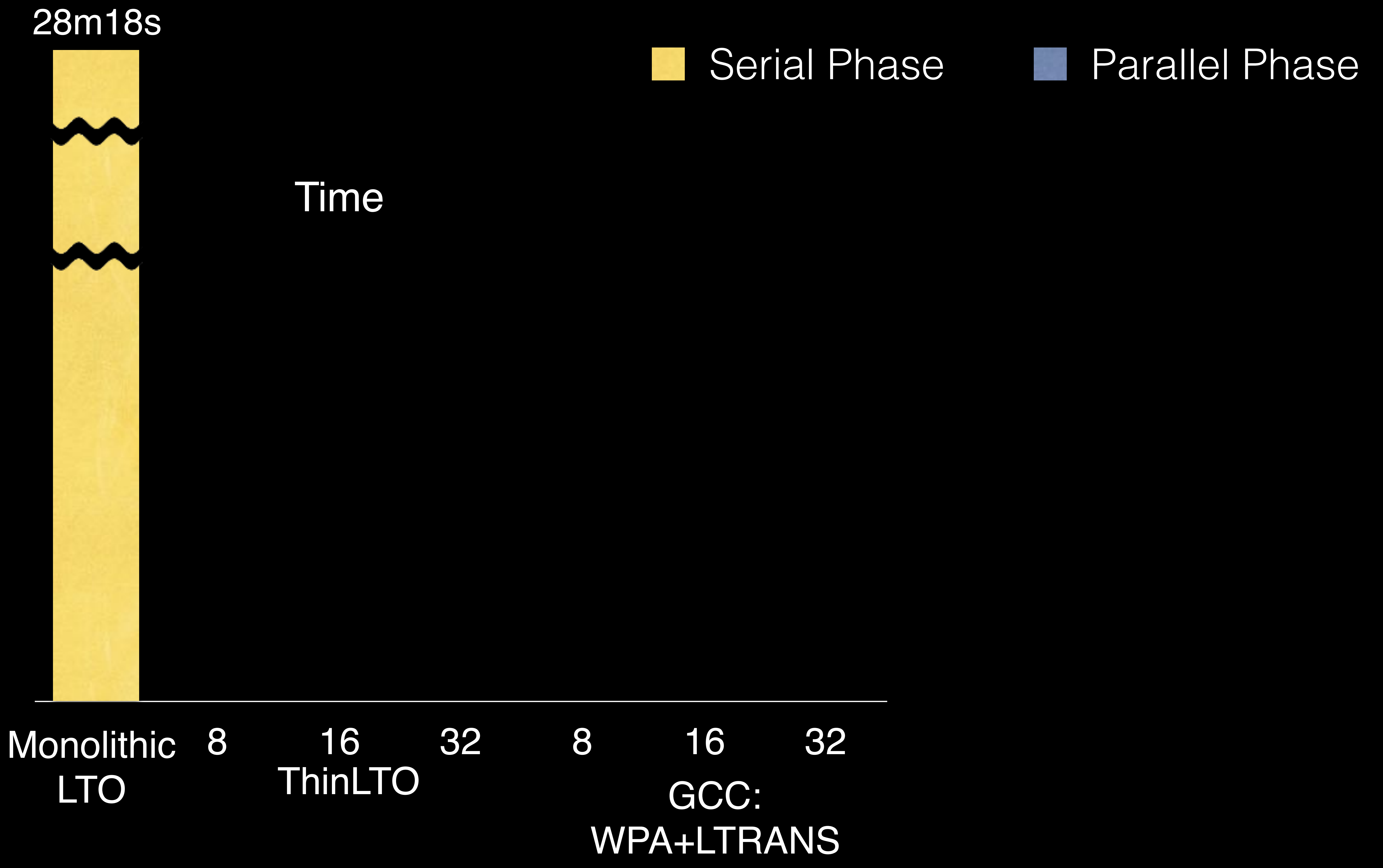
Chromium Build Comparisons

No Debug (-g0)

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Chromium Build Comparisons

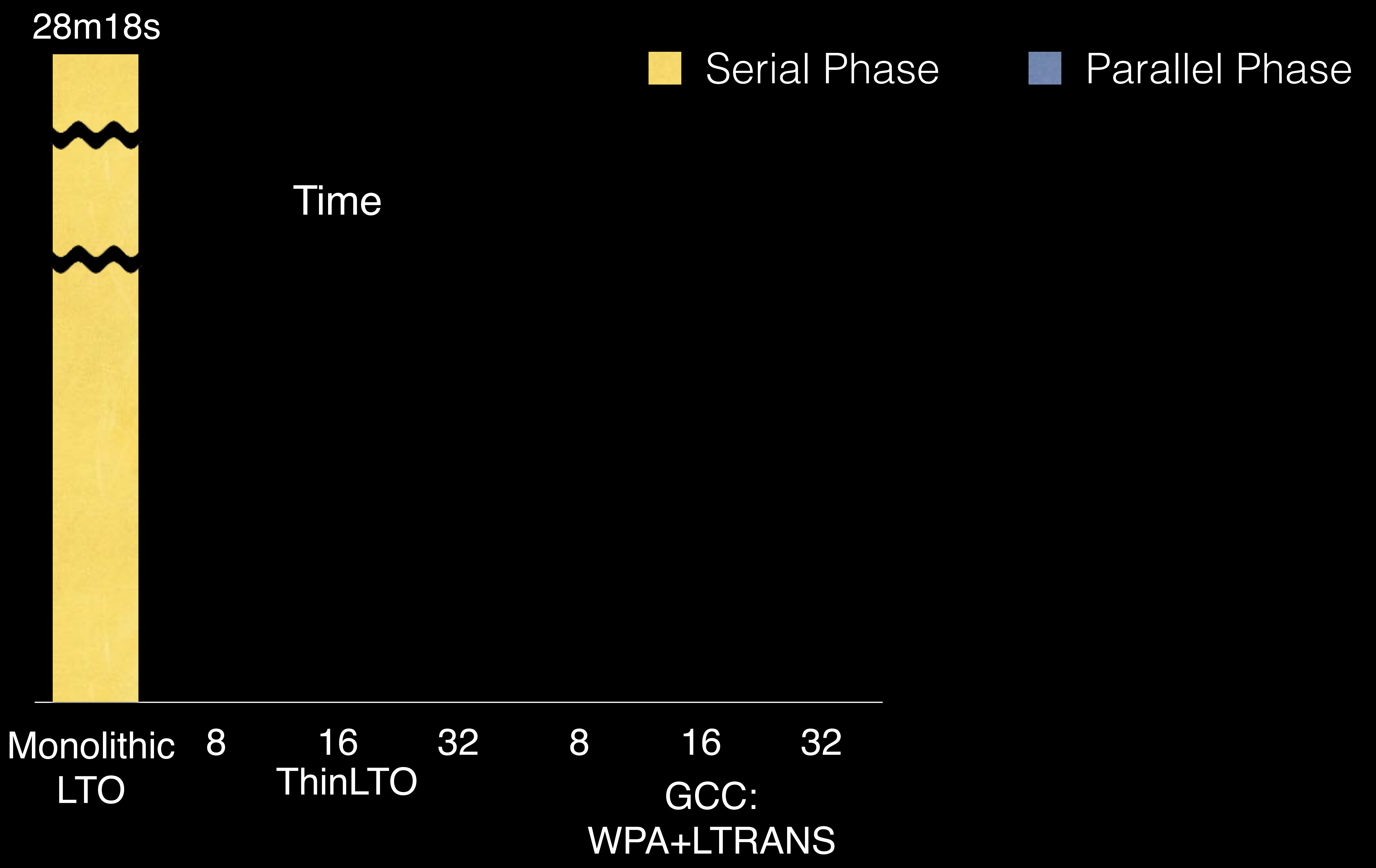
No Debug (-g0)



Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Chromium Build Comparisons

No Debug (-g0)

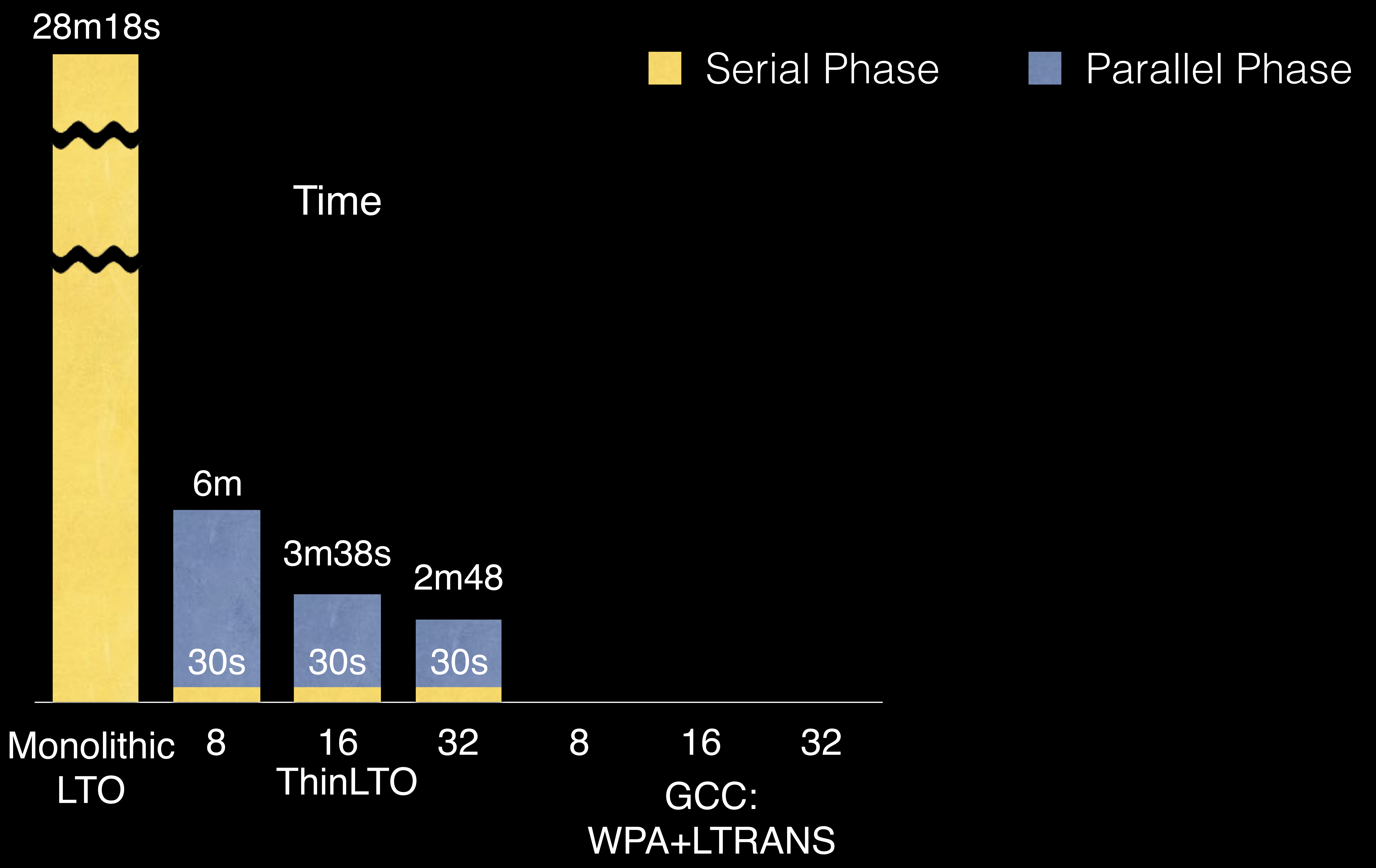


With Debug: Monolithic LTO crashes after >2h and >50GB mem

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Chromium Build Comparisons

No Debug (-g0)

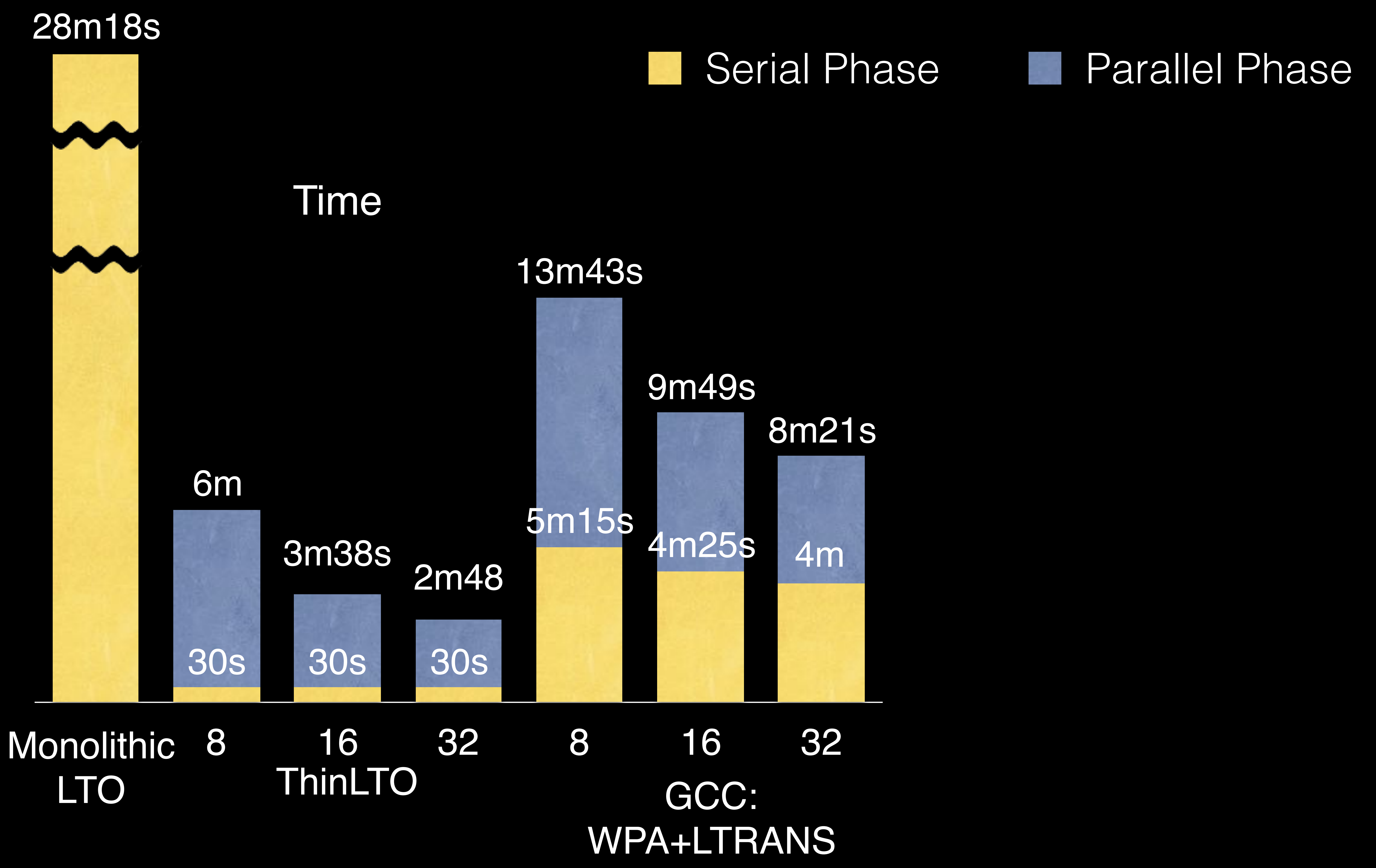


With Debug: Monolithic LTO crashes after >2h and >50GB mem

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Chromium Build Comparisons

No Debug (-g0)

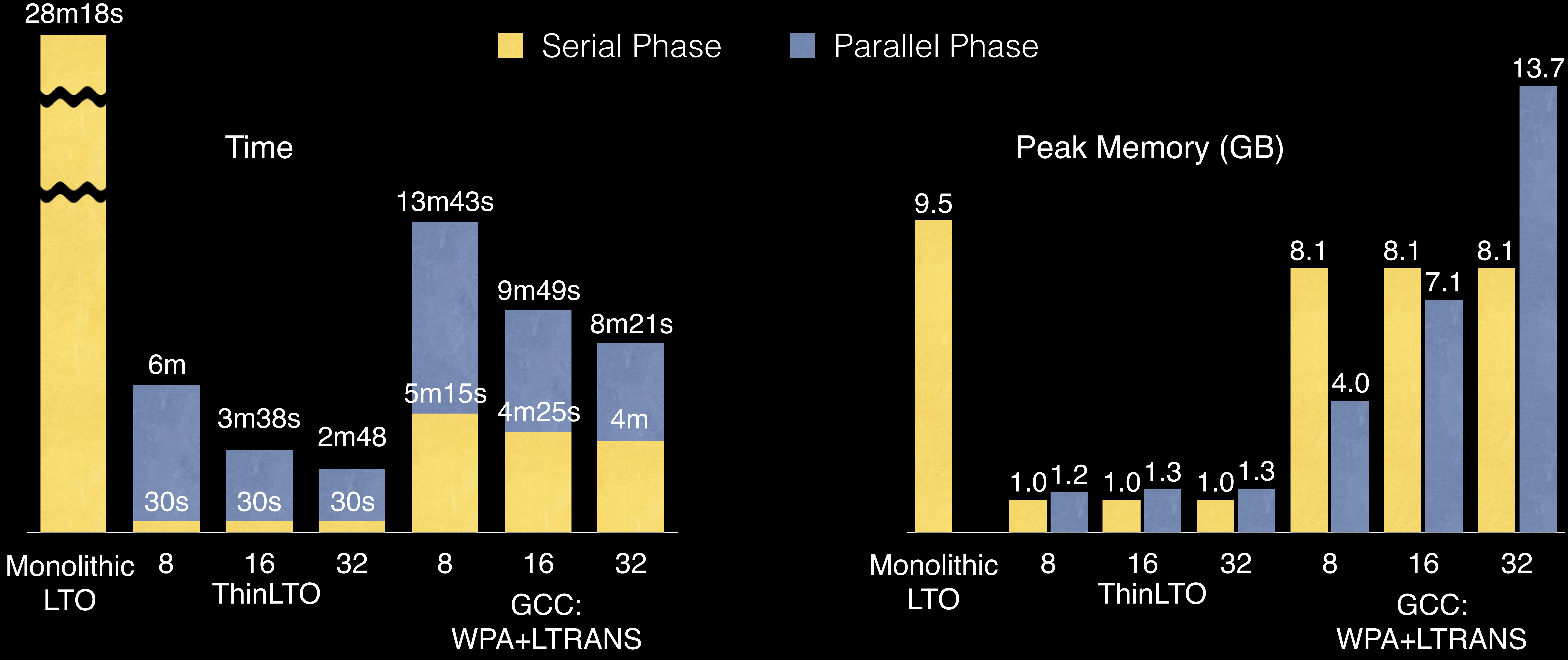


With Debug: Monolithic LTO crashes after >2h and >50GB mem

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Chromium Build Comparisons

No Debug (-g0)

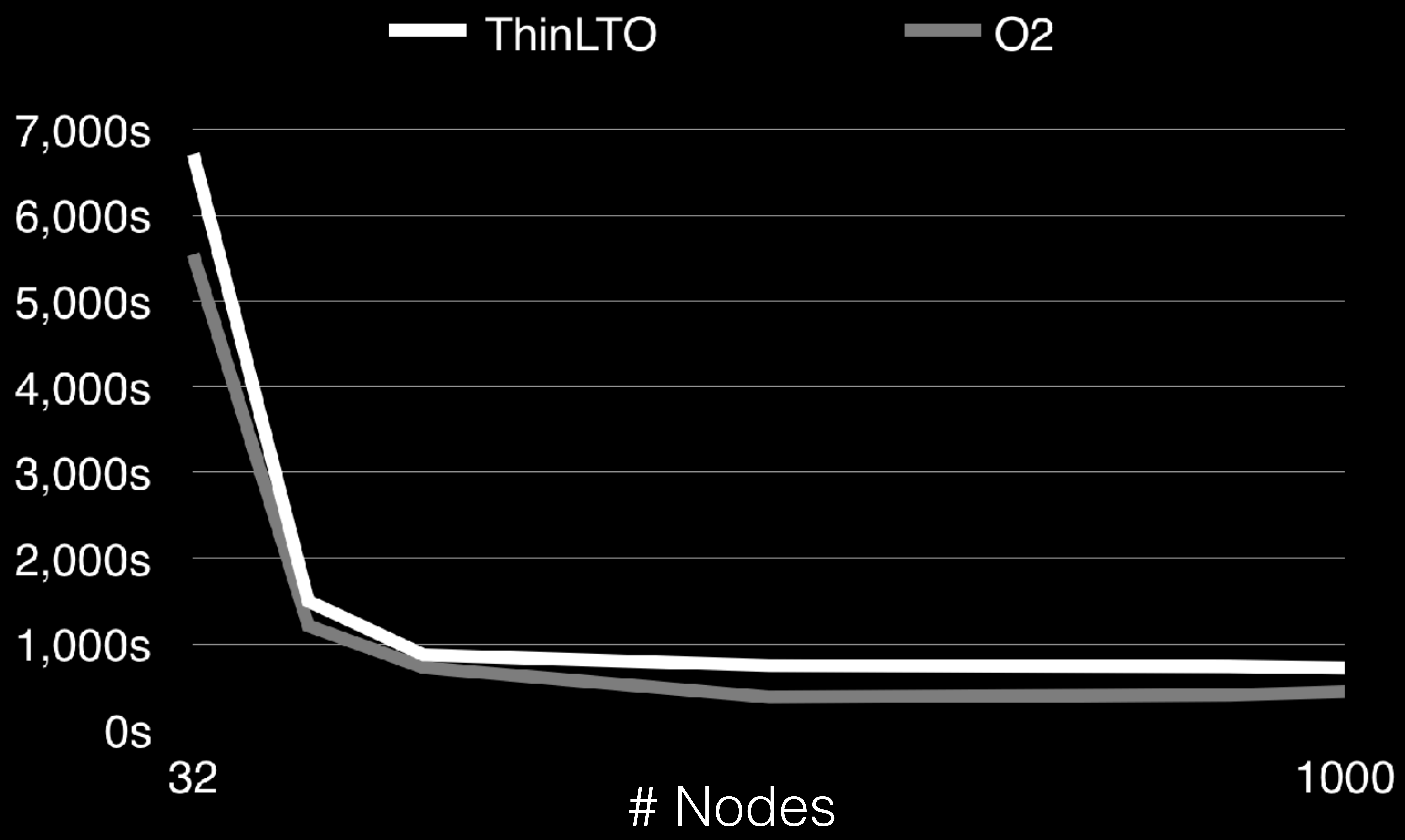


With Debug: Monolithic LTO crashes after >2h and >50GB mem

Linux 16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz

Distributed Build: Ad Delivery

Distributed Build: Ad Delivery



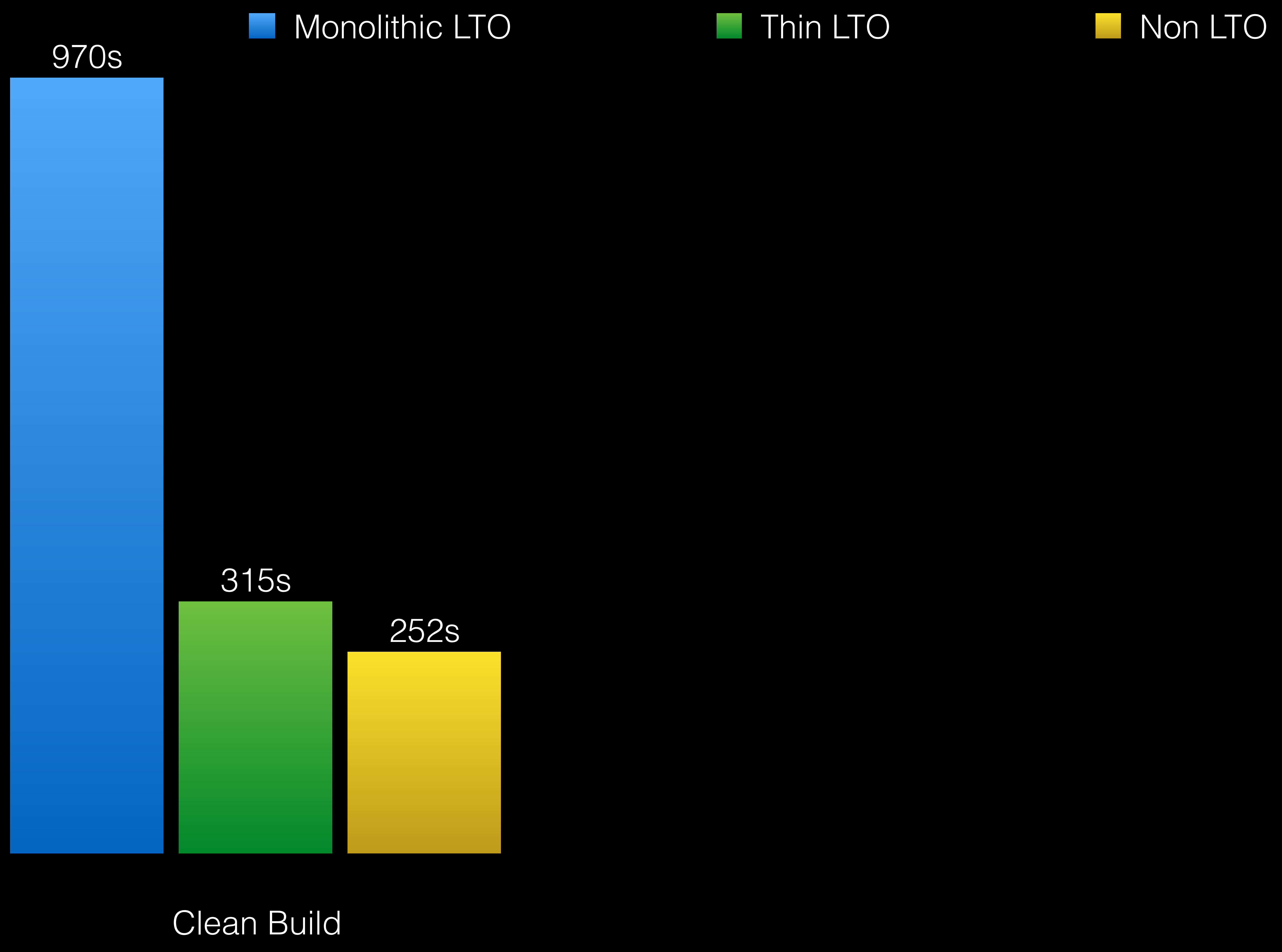
Performance: **Incremental** Build Time for Clang

Time to run `ninja clang`

2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

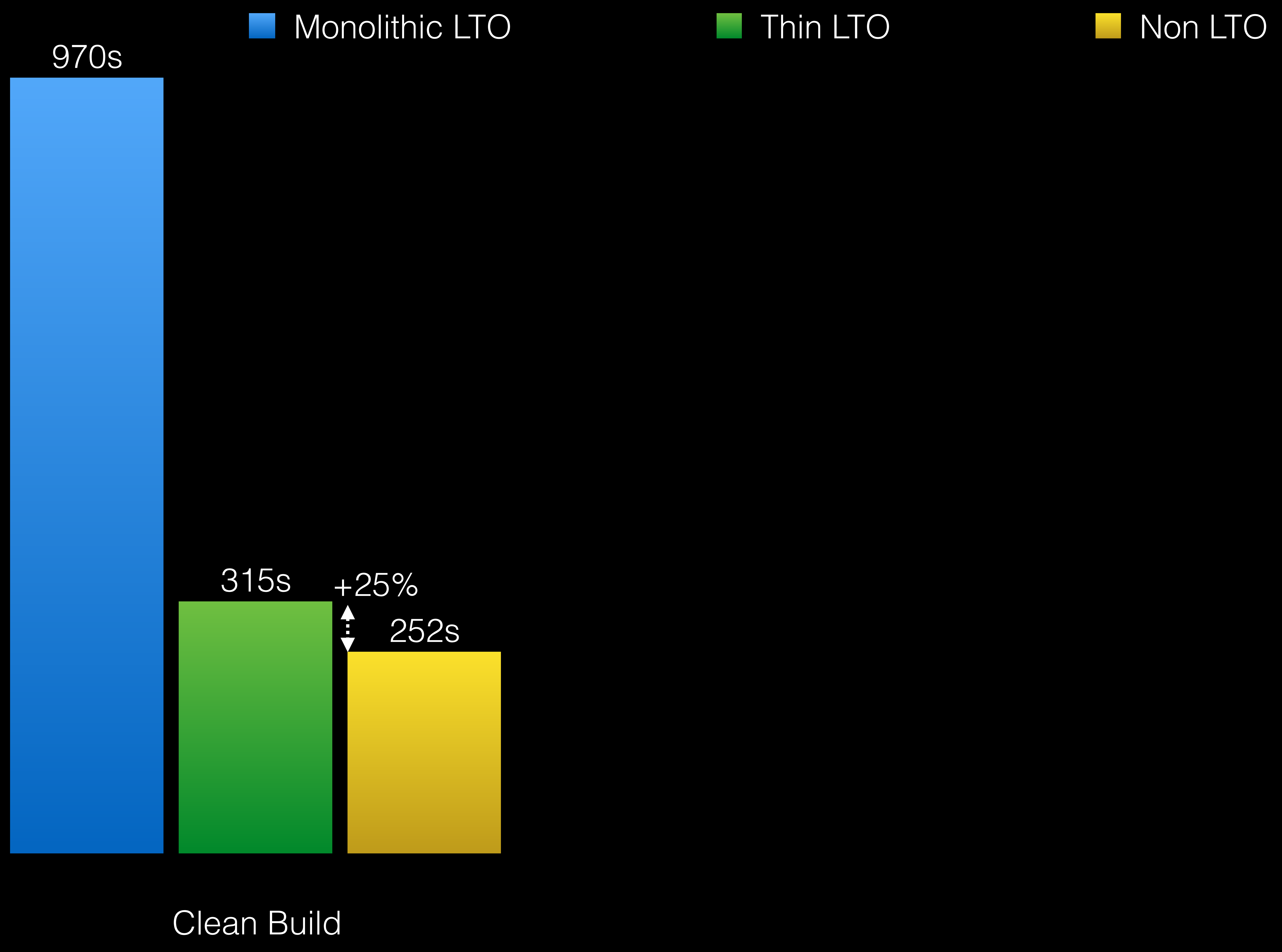
Time to run `ninja clang`



2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

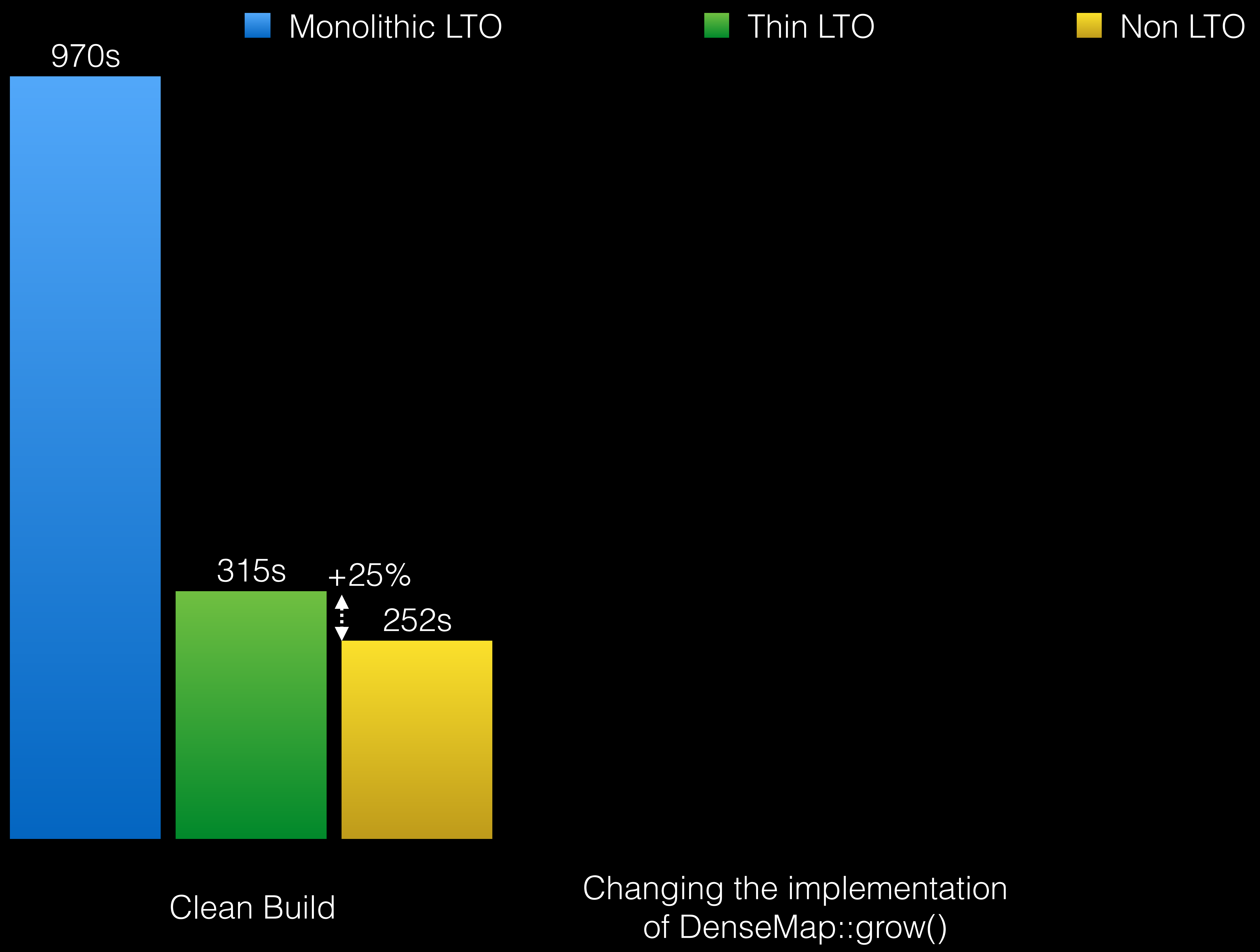
Time to run `ninja clang`



2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

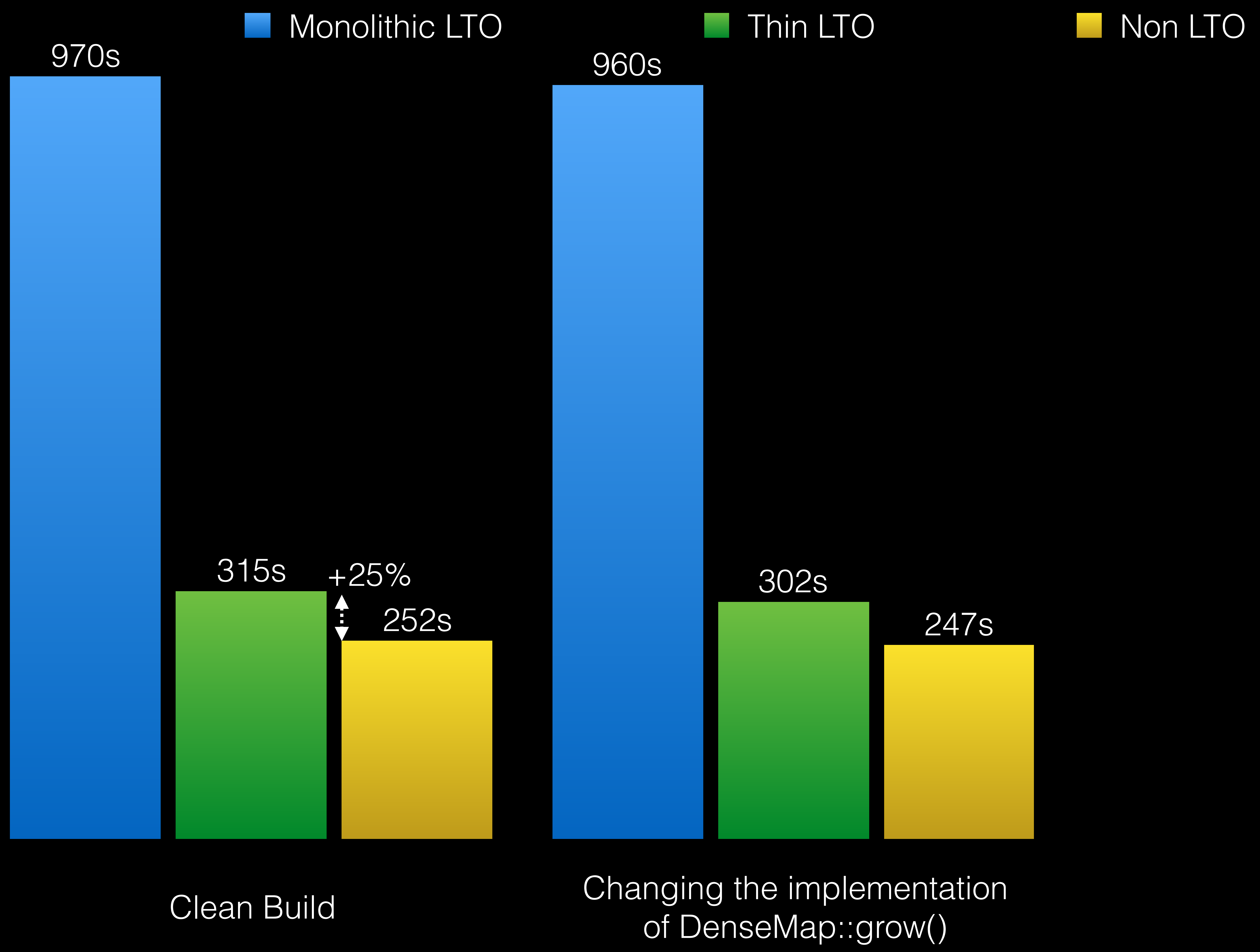
Time to run `ninja clang`



2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

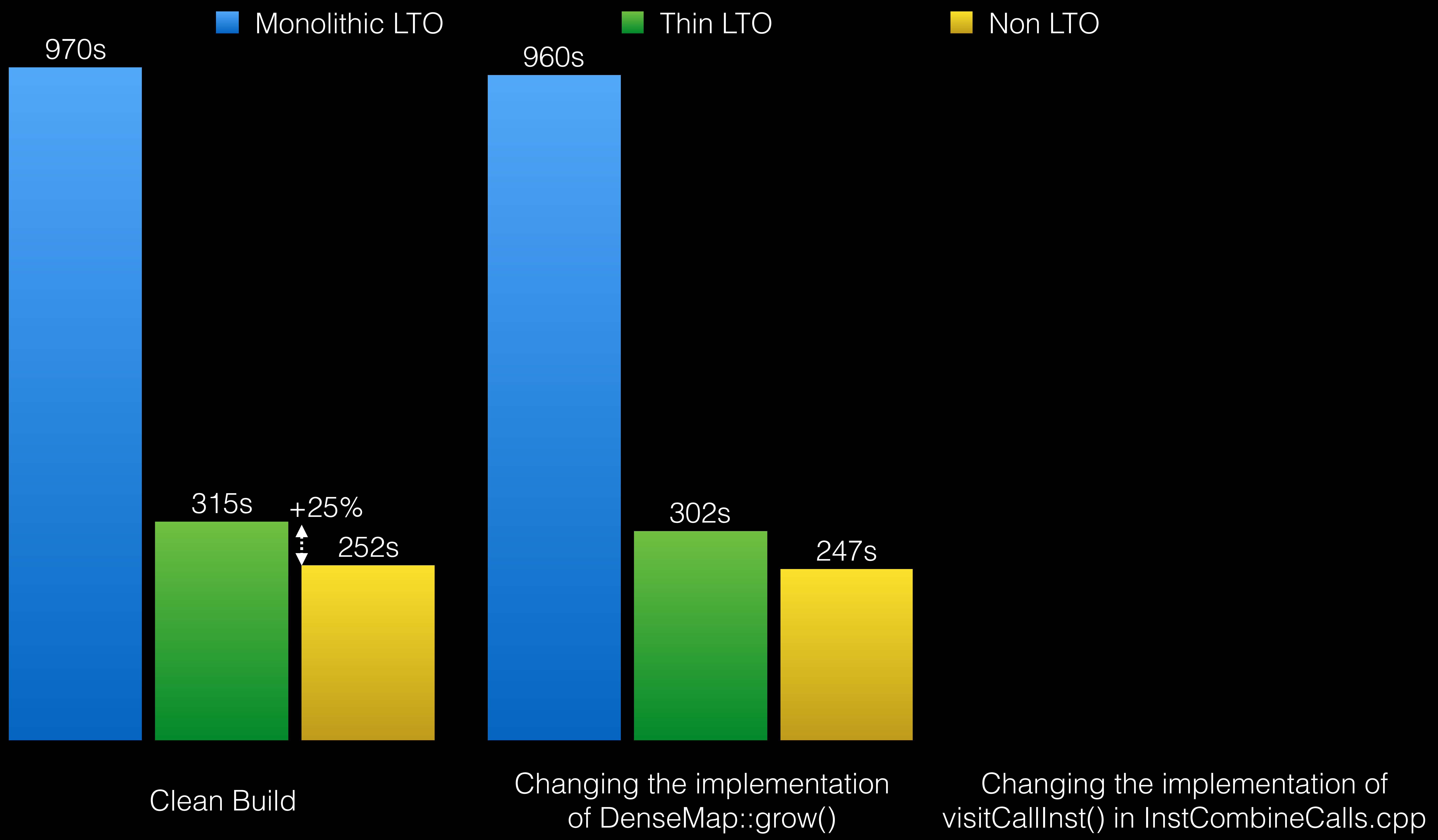
Time to run `ninja clang`



2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

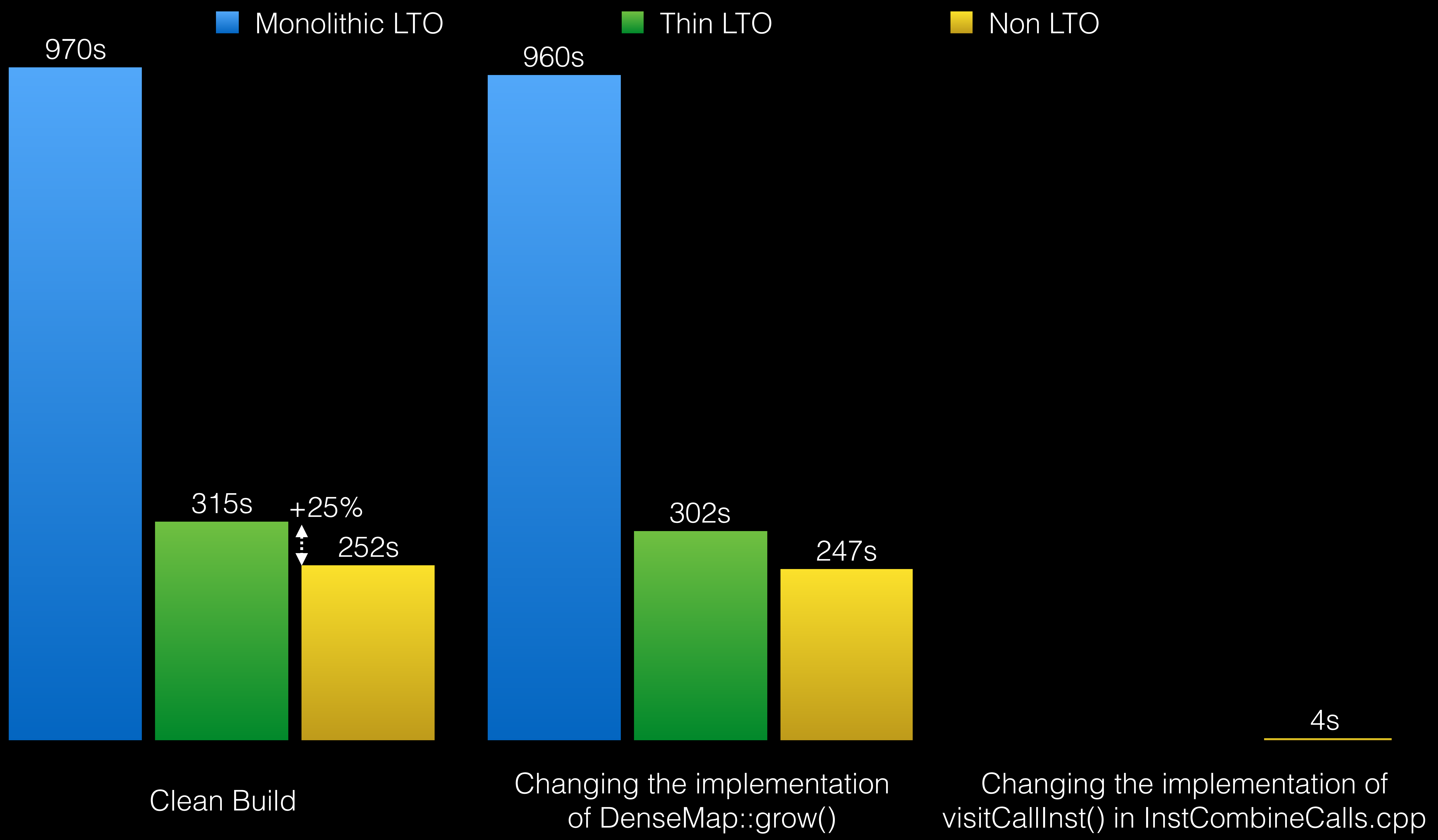
Time to run `ninja clang`



2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

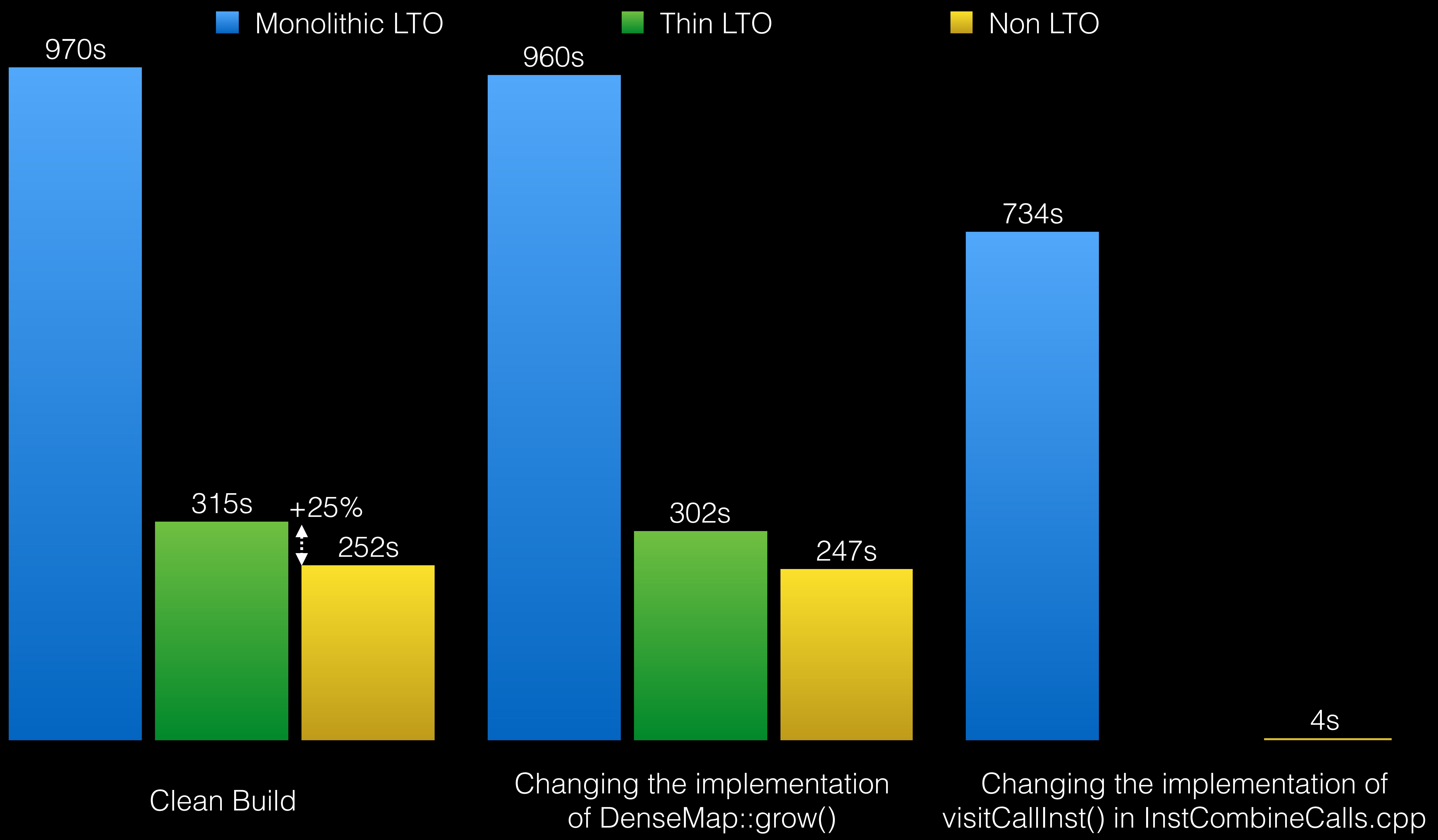
Time to run `ninja clang`



2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

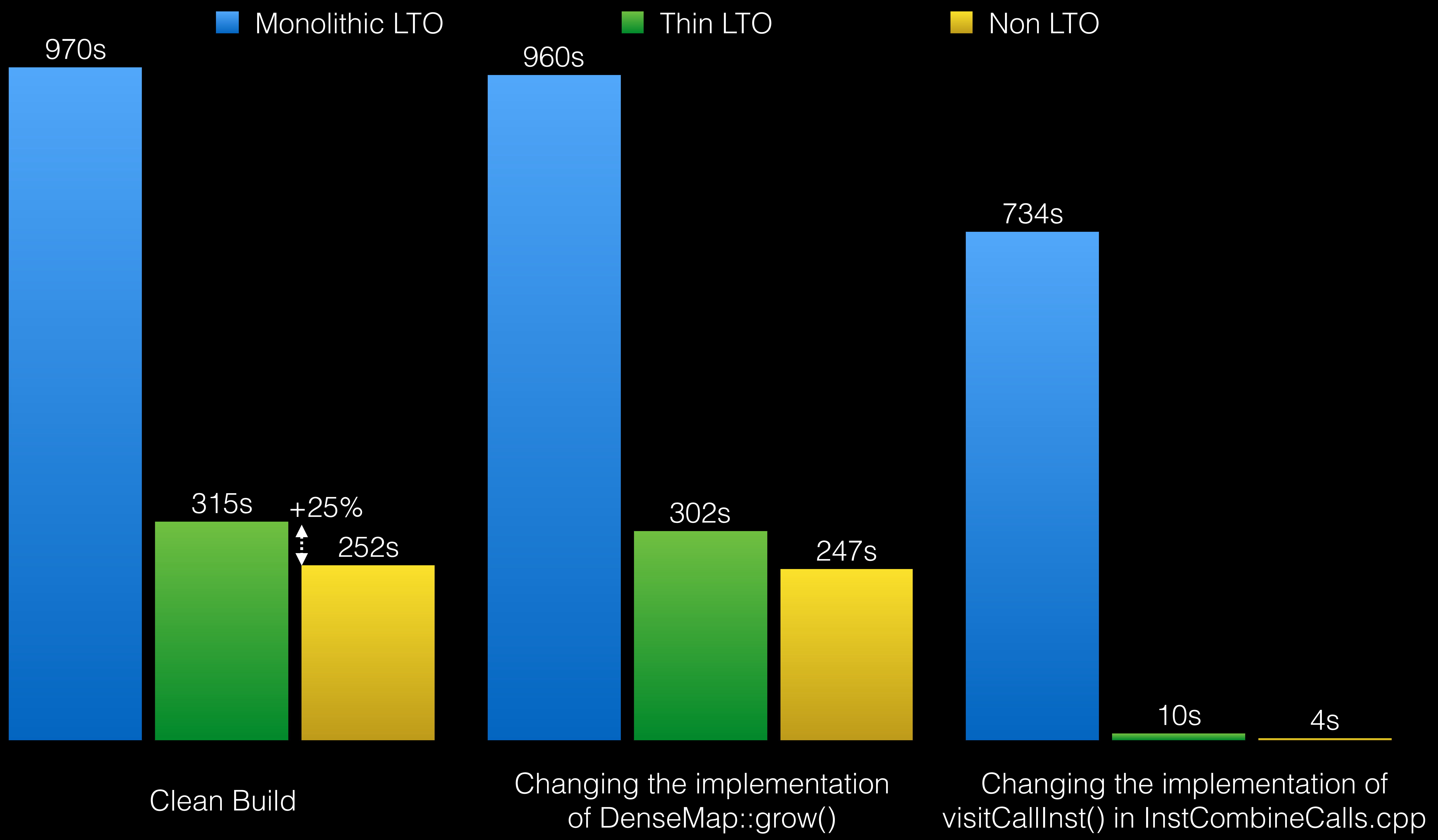
Time to run `ninja clang`



2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance: **Incremental** Build Time for Clang

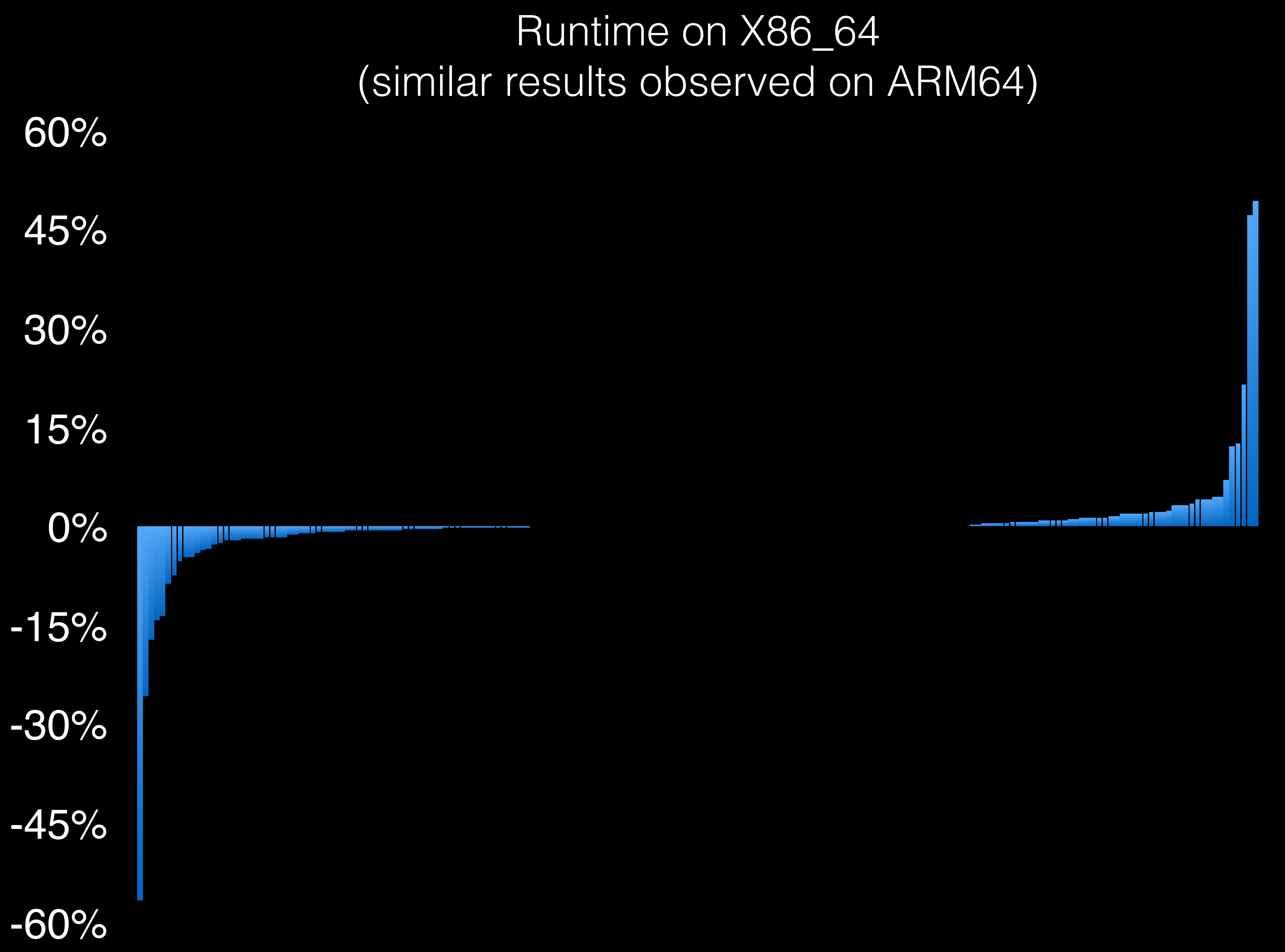
Time to run `ninja clang`



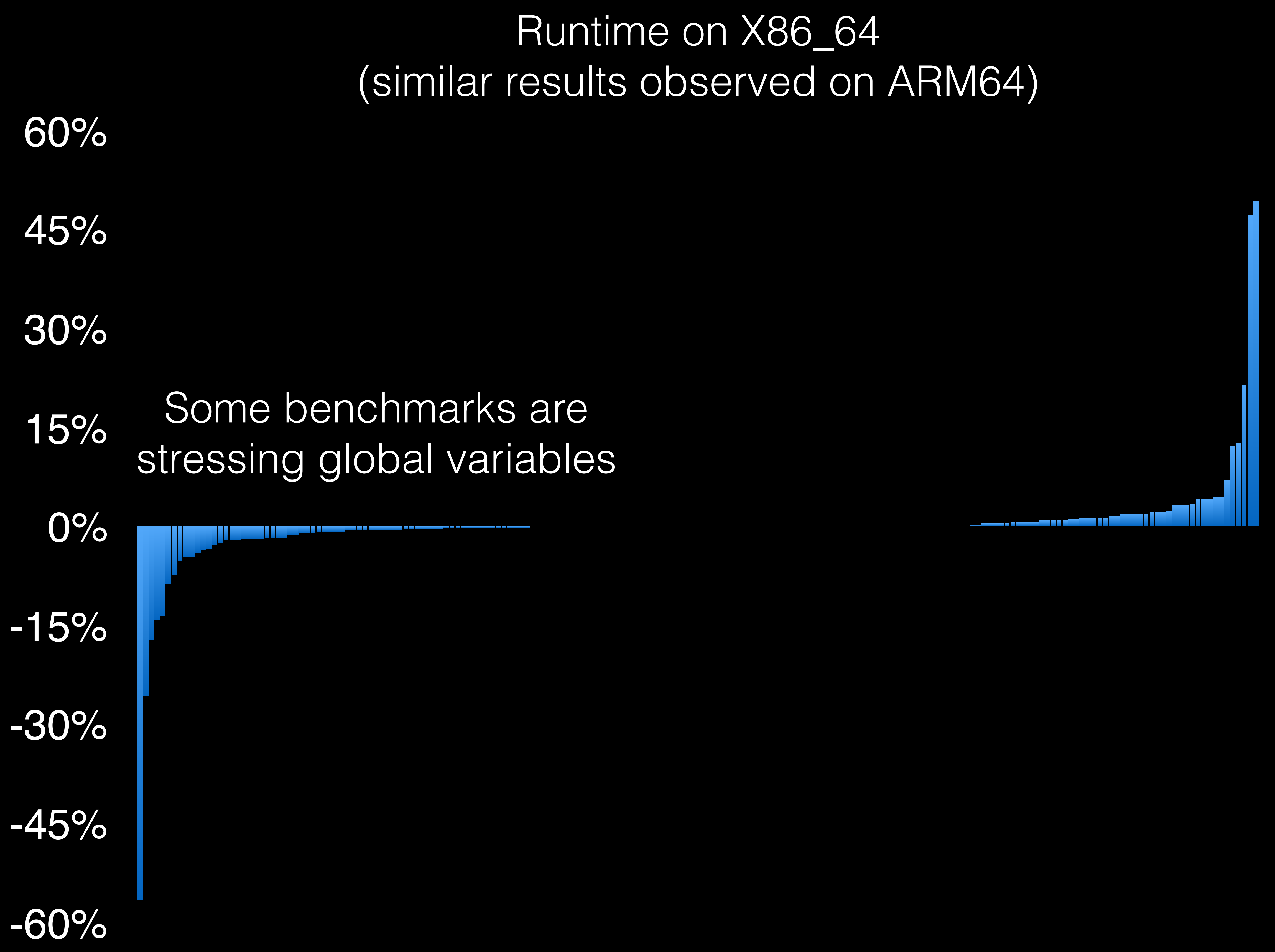
2013 MacPro - 12-cores E5-2697 @ 2.70GHz

Performance against Monolithic LTO: LLVM test-suite

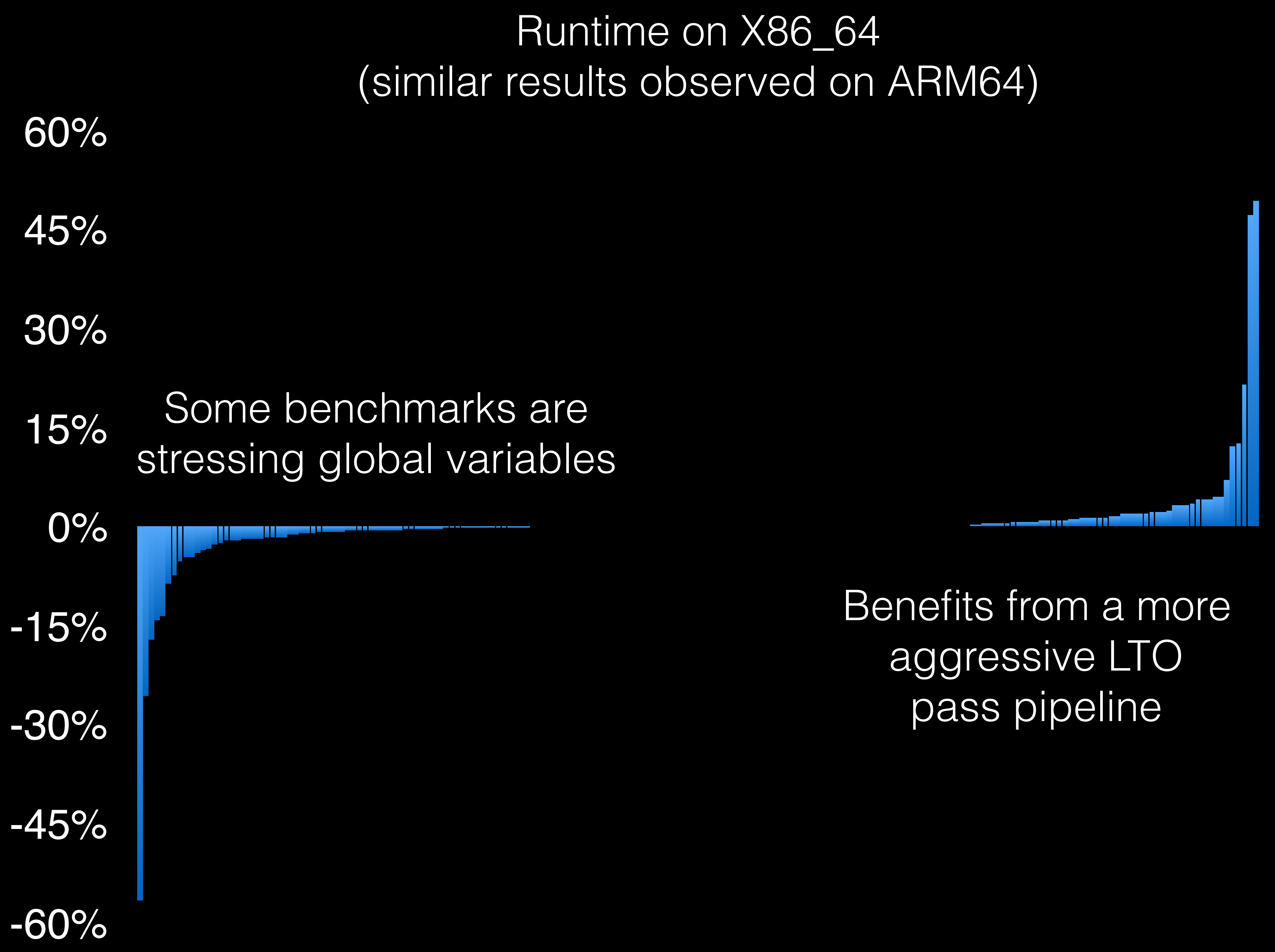
Performance against Monolithic LTO: LLVM test-suite



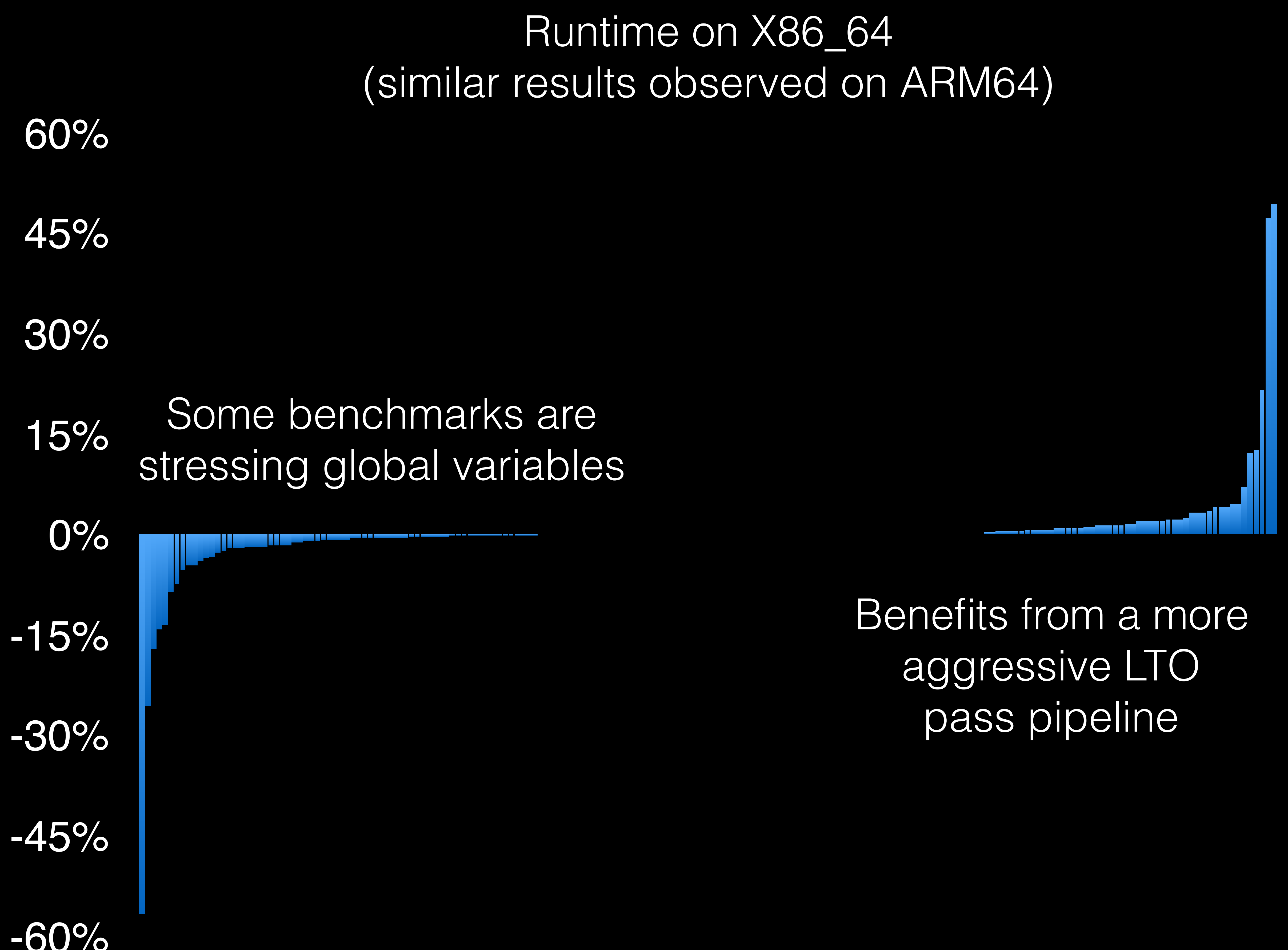
Performance against Monolithic LTO: LLVM test-suite

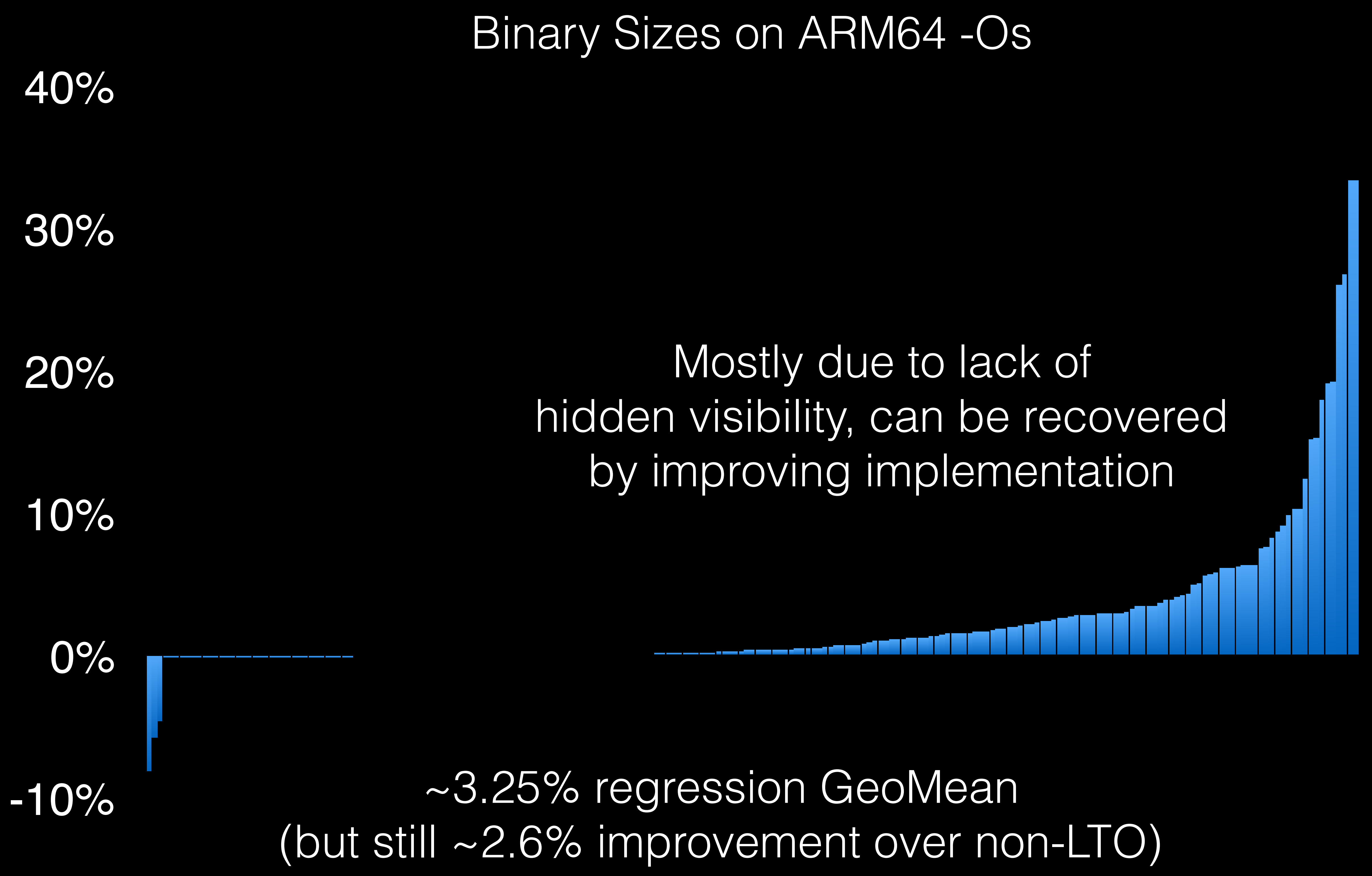


Performance against Monolithic LTO: LLVM test-suite



Performance against Monolithic LTO: LLVM test-suite





Run-time Performance: SPEC cpu2006

Improvement over -O2 (**all with PGO**)

Future Work

Future Work

- On-going new libLTO Interface: better linker information for better codegen

  Example: resolution of weak definition to strong definition.

Future Work

- On-going new libLTO Interface: better linker information for better codegen

  Example: resolution of weak definition to strong definition.
- On-going tuning for PGO

Future Work

- On-going new libLTO Interface: better linker information for better codegen

  Example: resolution of weak definition to strong definition.
- On-going tuning for PGO
- LTO Devirtualization and CFI (Control-Flow Integrity) integration.

Future Work

- On-going new libLTO Interface: better linker information for better codegen
  Example: resolution of weak definition to strong definition.
- On-going tuning for PGO
- LTO Devirtualization and CFI (Control-Flow Integrity) integration.
- Propagate function attributes across modules via summary.

Future Work

- On-going new libLTO Interface: better linker information for better codegen

  Example: resolution of weak definition to strong definition.
- On-going tuning for PGO
- LTO Devirtualization and CFI (Control-Flow Integrity) integration.
- Propagate function attributes across modules via summary.
- More bitcode format changes to improve link-time, especially with debug info

Future Work

- On-going new libLTO Interface: better linker information for better codegen
  Example: resolution of weak definition to strong definition.
- On-going tuning for PGO
- LTO Devirtualization and CFI (Control-Flow Integrity) integration.
- Propagate function attributes across modules via summary.
- More bitcode format changes to improve link-time, especially with debug info
- Ability to move function across modules instead of importing, especially useful when a single call site exists and we could internalize.

Future Work

- On-going new libLTO Interface: better linker information for better codegen
  Example: resolution of weak definition to strong definition.
- On-going tuning for PGO
- LTO Devirtualization and CFI (Control-Flow Integrity) integration.
- Propagate function attributes across modules via summary.
- More bitcode format changes to improve link-time, especially with debug info
- Ability to move function across modules instead of importing, especially useful when a single call site exists and we could internalize.
- *Augment* the edges in the call-graph with mod-ref, constant range, …

Special Thanks:
Akira Hatanaka and Bruno Cardoso Lopes (*performance testing/debunking*),
Adrian Prantl and Duncan Exon Smith (*major changes in debug info and bitcode format*),
Peter Collingbourne (new LTO API)
Rafael Ávila de Espíndola (Lot of support and help!)
Davide Italiano (lld support)
Piotr Padlewski (PGO import heuristics)

Conclusion

Special Thanks:
Akira Hatanaka and Bruno Cardoso Lopes (*performance testing/debunking*),
Adrian Prantl and Duncan Exon Smith (*major changes in debug info and bitcode format*),
Peter Collingbourne (new LTO API)
Rafael Ávila de Espíndola (Lot of support and help!)
Davide Italiano (lld support)
Piotr Padlewski (PGO import heuristics)

Conclusion

Scales as a regular non-LTO build

Special Thanks:
Akira Hatanaka and Bruno Cardoso Lopes (*performance testing/debunking*),
Adrian Prantl and Duncan Exon Smith (*major changes in debug info and bitcode format*),
Peter Collingbourne (new LTO API)
Rafael Ávila de Espíndola (Lot of support and help!)
Davide Italiano (lld support)
Piotr Padlewski (PGO import heuristics)

Conclusion

Scales as a regular non-LTO build

Performance close to Monolithic LTO

Special Thanks:
Akira Hatanaka and Bruno Cardoso Lopes (*performance testing/debunking*),
Adrian Prantl and Duncan Exon Smith (*major changes in debug info and bitcode format*),
Peter Collingbourne (new LTO API)
Rafael Ávila de Espíndola (Lot of support and help!)
Davide Italiano (lld support)
Piotr Padlewski (PGO import heuristics)

Conclusion

Scales as a regular non-LTO build

Performance close to Monolithic LTO

Incremental build is critical for day-to-day development!

Special Thanks:
Akira Hatanaka and Bruno Cardoso Lopes (*performance testing/debunking*),
Adrian Prantl and Duncan Exon Smith (*major changes in debug info and bitcode format*),
Peter Collingbourne (new LTO API)
Rafael Ávila de Espíndola (Lot of support and help!)
Davide Italiano (lld support)
Piotr Padlewski (PGO import heuristics)

Conclusion

Scales as a regular non-LTO build

Performance close to Monolithic LTO

Incremental build is critical for day-to-day development!

On the path of replacing O2/O3 by default in production build

Special Thanks:
Akira Hatanaka and Bruno Cardoso Lopes (*performance testing/debunking*),
Adrian Prantl and Duncan Exon Smith (*major changes in debug info and bitcode format*),
Peter Collingbourne (new LTO API)
Rafael Ávila de Espíndola (Lot of support and help!)
Davide Italiano (lld support)
Piotr Padlewski (PGO import heuristics)

Conclusion

Scales as a regular non-LTO build

Performance close to Monolithic LTO

Incremental build is critical for day-to-day development!

On the path of replacing O2/O3 by default in production build

Special Thanks:
Akira Hatanaka and Bruno Cardoso Lopes (*performance testing/debunking*),
Adrian Prantl and Duncan Exon Smith (*major changes in debug info and bitcode format*),
Peter Collingbourne (new LTO API)
Rafael Ávila de Espíndola (Lot of support and help!)
Davide Italiano (lld support)
Piotr Padlewski (PGO import heuristics)

Backup Slides

Different optimization pipelines

-O3 LTO

-O3 w/o LTO

-O3 ThinLTO

Early Opt

CGSCC

IR Simplification

Inliner

Drop Available Ext.

Aggressive Optimizations

Codegen

Different optimization pipelines

-O3 LTO

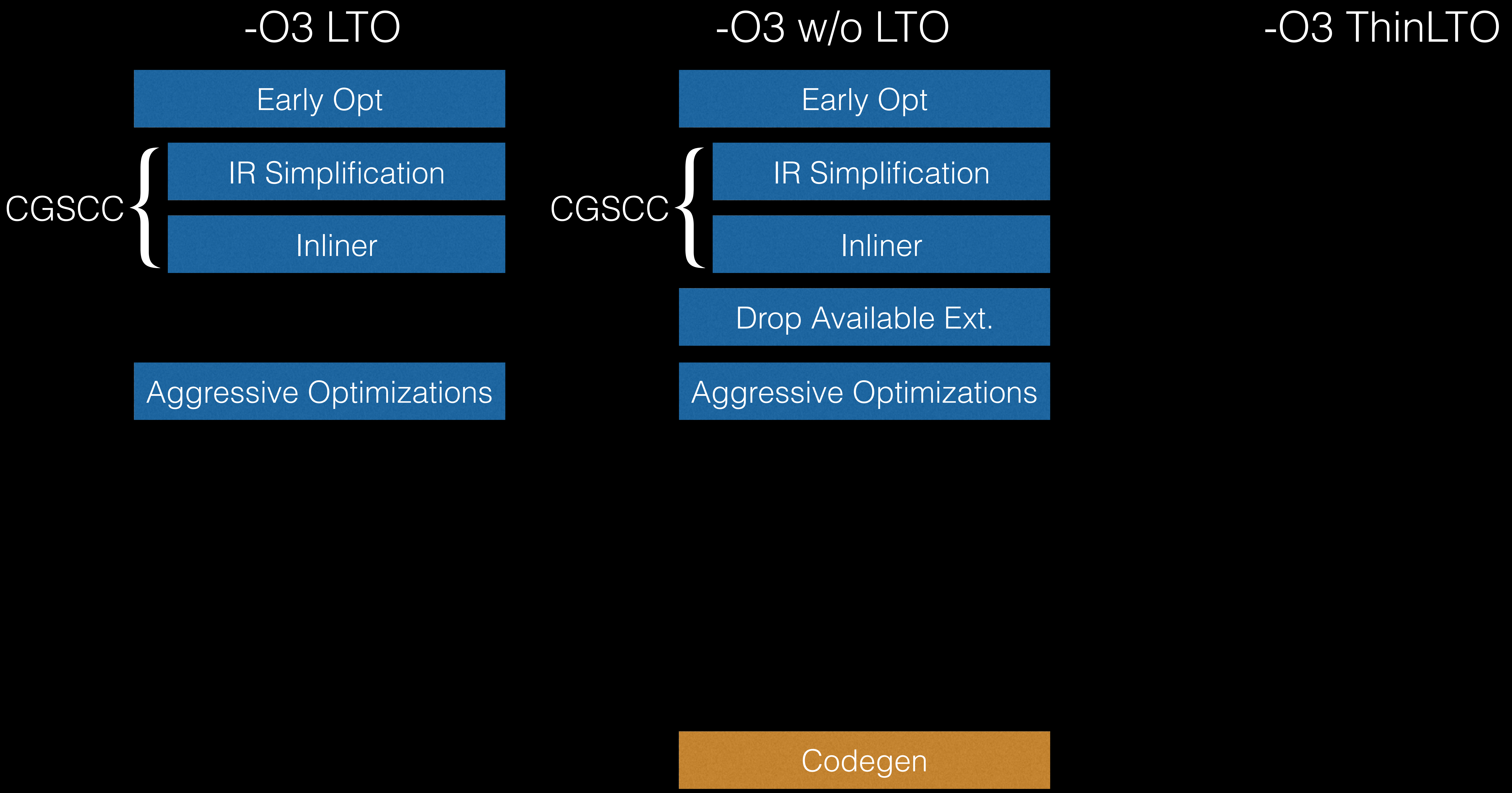
- Early Opt
- CGSCC: IR Simplification, Inliner
- Drop Available Ext.
- Aggressive Optimizations

-O3 w/o LTO

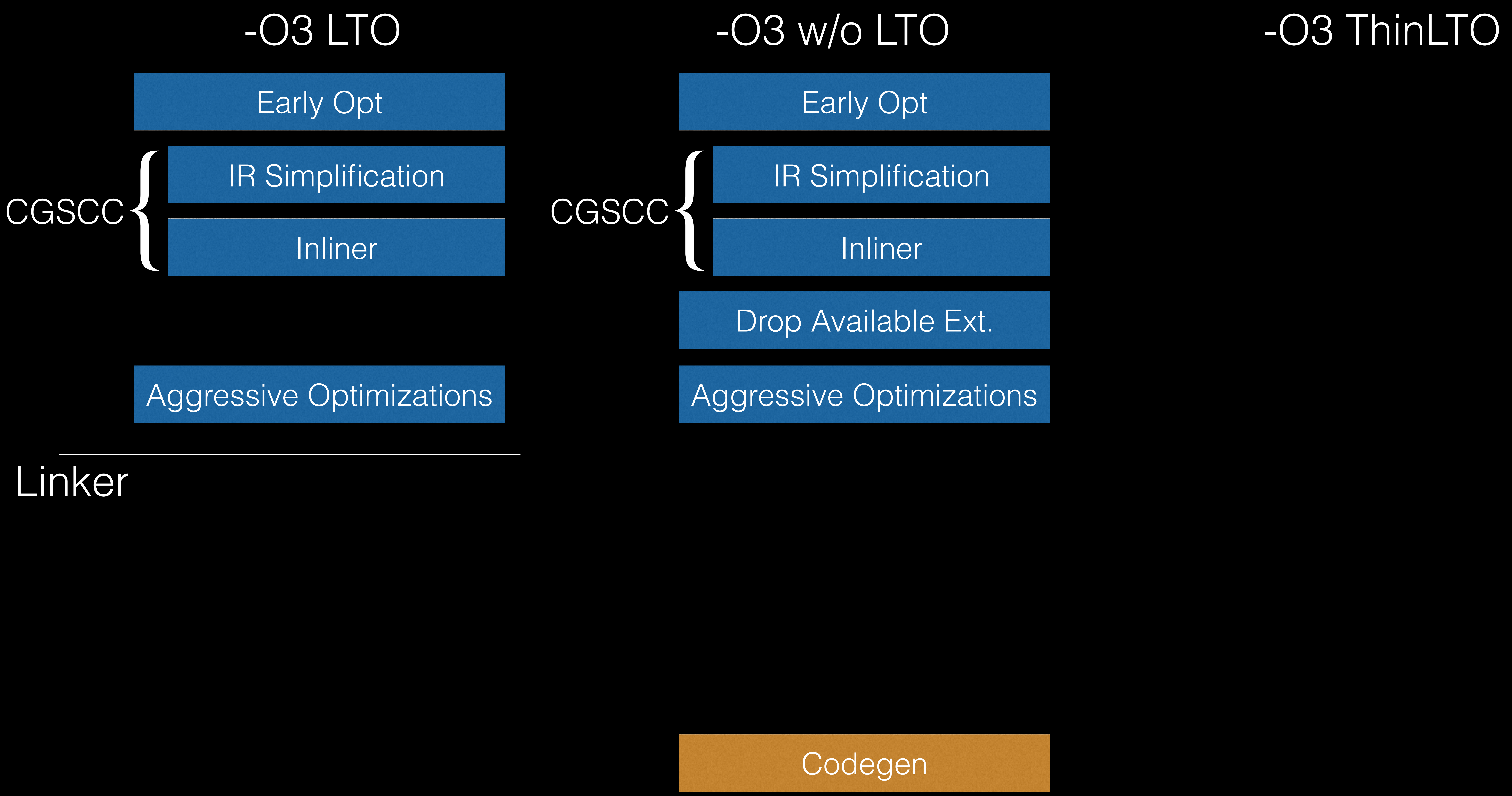
- Early Opt
- CGSCC: IR Simplification, Inliner
- Drop Available Ext.
- Aggressive Optimizations
- Codegen

-O3 ThinLTO

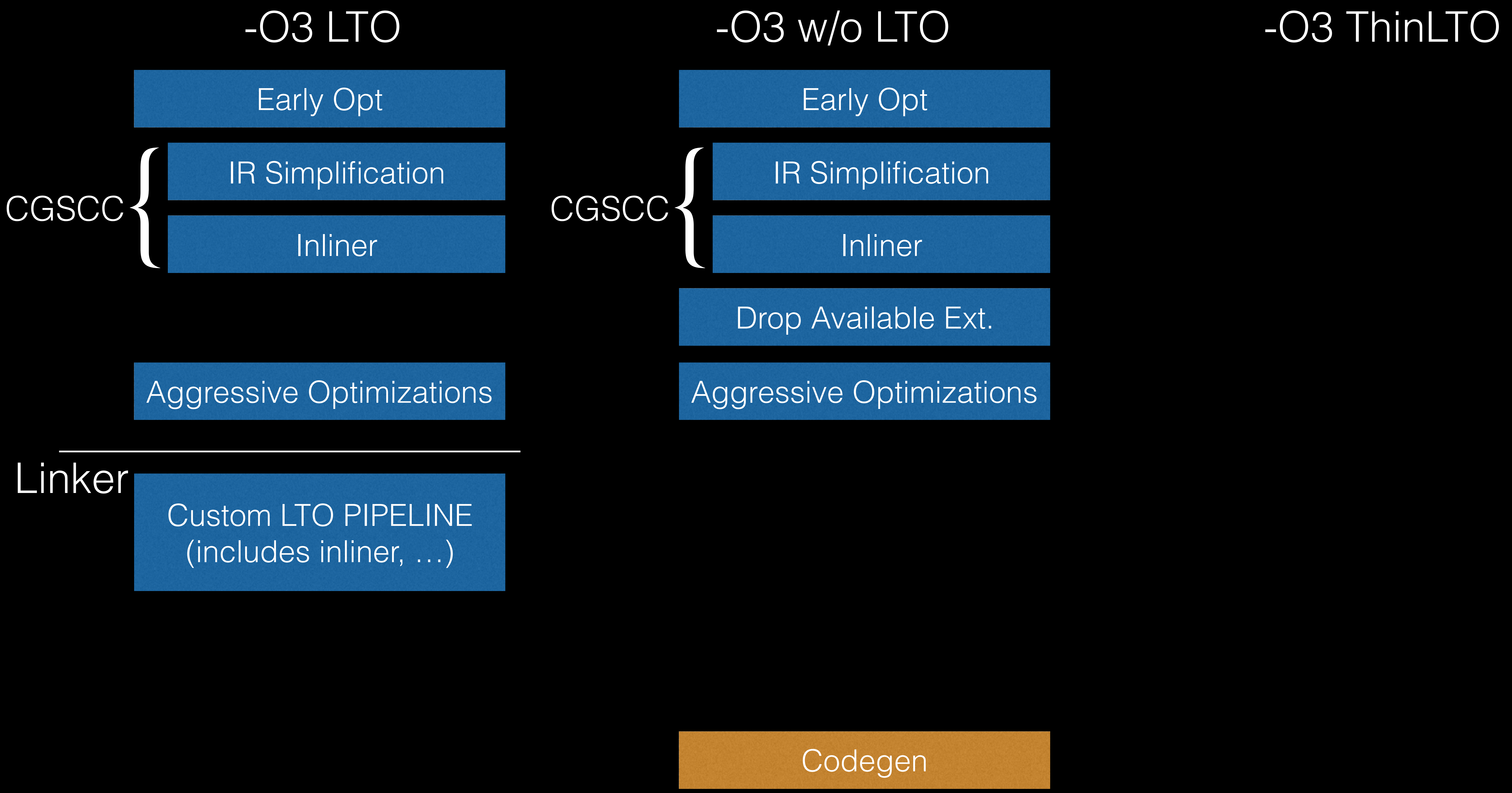
Different optimization pipelines



Different optimization pipelines



Different optimization pipelines



Different optimization pipelines

-O3 LTO

Early Opt

CGSCC:
- IR Simplification
- Inliner

Aggressive Optimizations

Linker

Custom LTO PIPELINE
(includes inliner, …)

Codegen

-O3 w/o LTO

Early Opt

CGSCC:
- IR Simplification
- Inliner

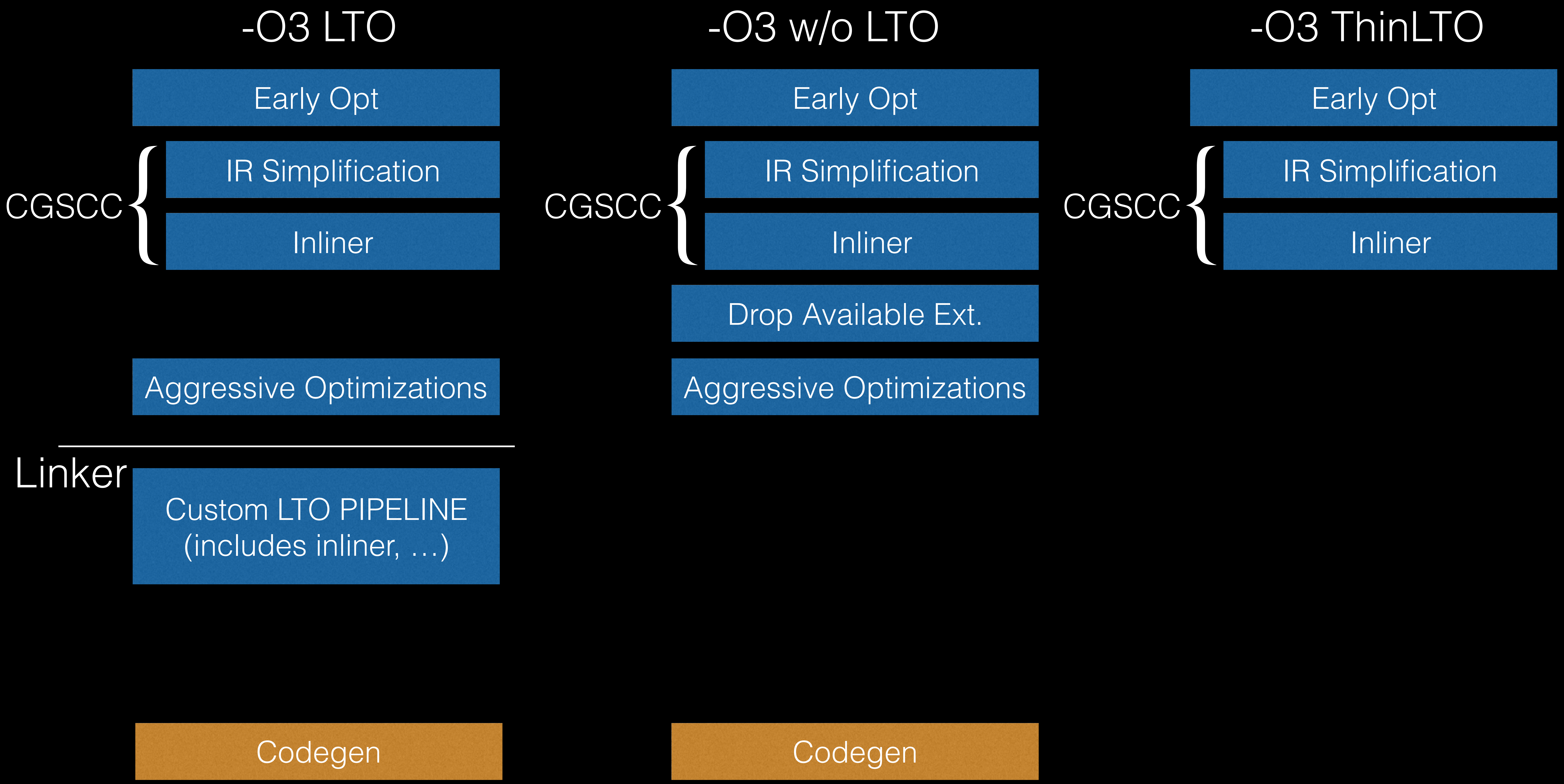
Drop Available Ext.

Aggressive Optimizations

Codegen

-O3 ThinLTO

Different optimization pipelines



Different optimization pipelines

Different optimization pipelines

Different optimization pipelines

-O3 LTO

Early Opt

CGSCC
- IR Simplification
- Inliner

Aggressive Optimizations

Linker

Custom LTO PIPELINE
(includes inliner, …)

Codegen

-O3 w/o LTO

Early Opt

CGSCC
- IR Simplification
- Inliner

Drop Available Ext.

Aggressive Optimizations

Codegen

-O3 ThinLTO

Early Opt

CGSCC
- IR Simplification
- Inliner

Linker

Early Opt

CGSCC
- IR Simplification
- Inliner

Drop Available Ext.

Aggressive Optimizations

Codegen

clang:

input files 217MB, and final binary is 52M

Summary blocks 0.85% on avg of IR .o sizes

Combined index total: 4M on-disk
(not stored on disk unless distributed though)

Chromium Build Comparisons

With Debug (-g2)

16-core (32-logical) Intel Xeon
E5-2690 @ 2.90GHz

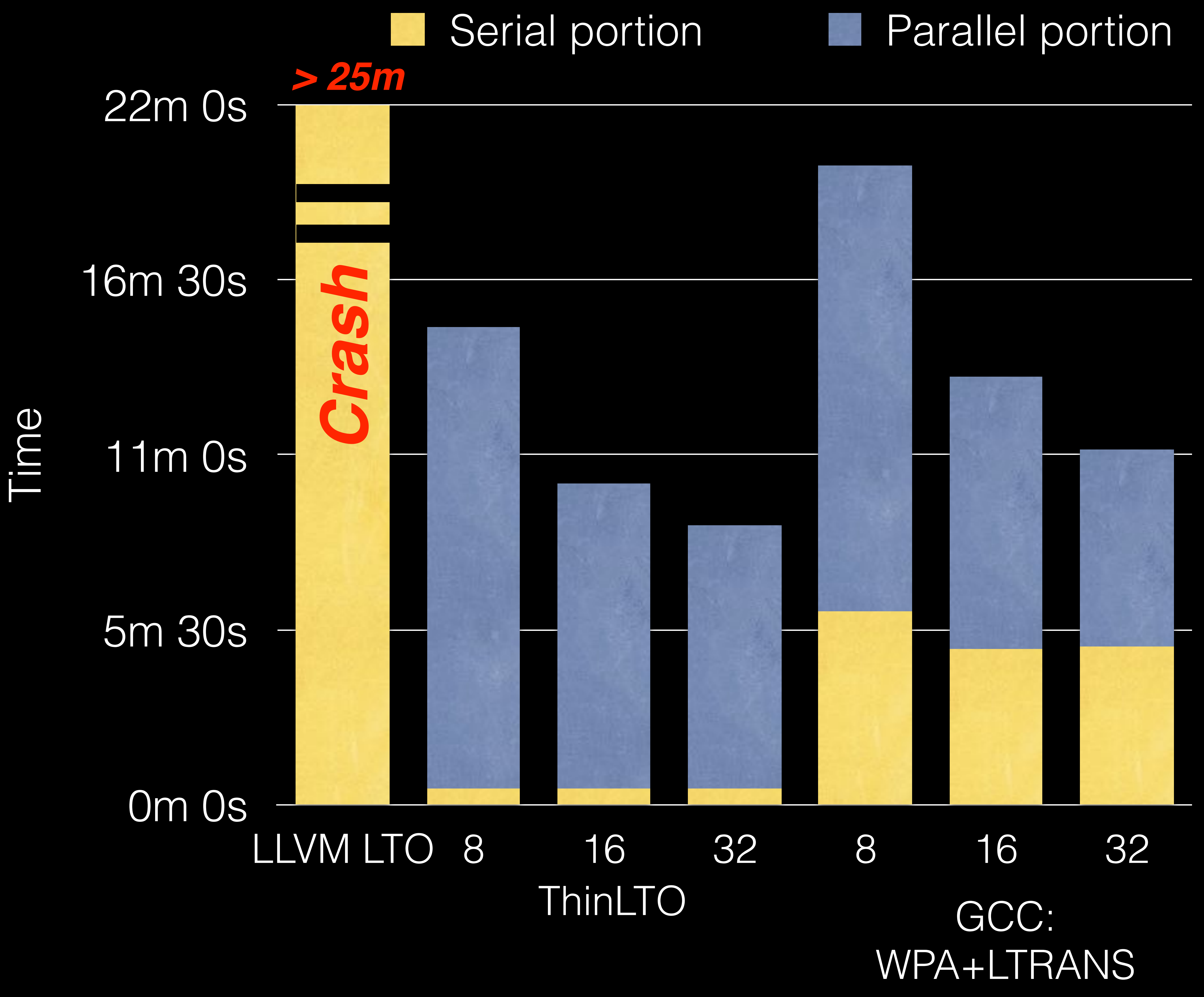
Chromium Build Comparisons

With Debug (-g2)

16-core (32-logical) Intel Xeon
E5-2690 @ 2.90GHz

Chromium Build Comparisons
With Debug (-g2)

16-core (32-logical) Intel Xeon
E5-2690 @ 2.90GHz



Chromium Build Comparisons

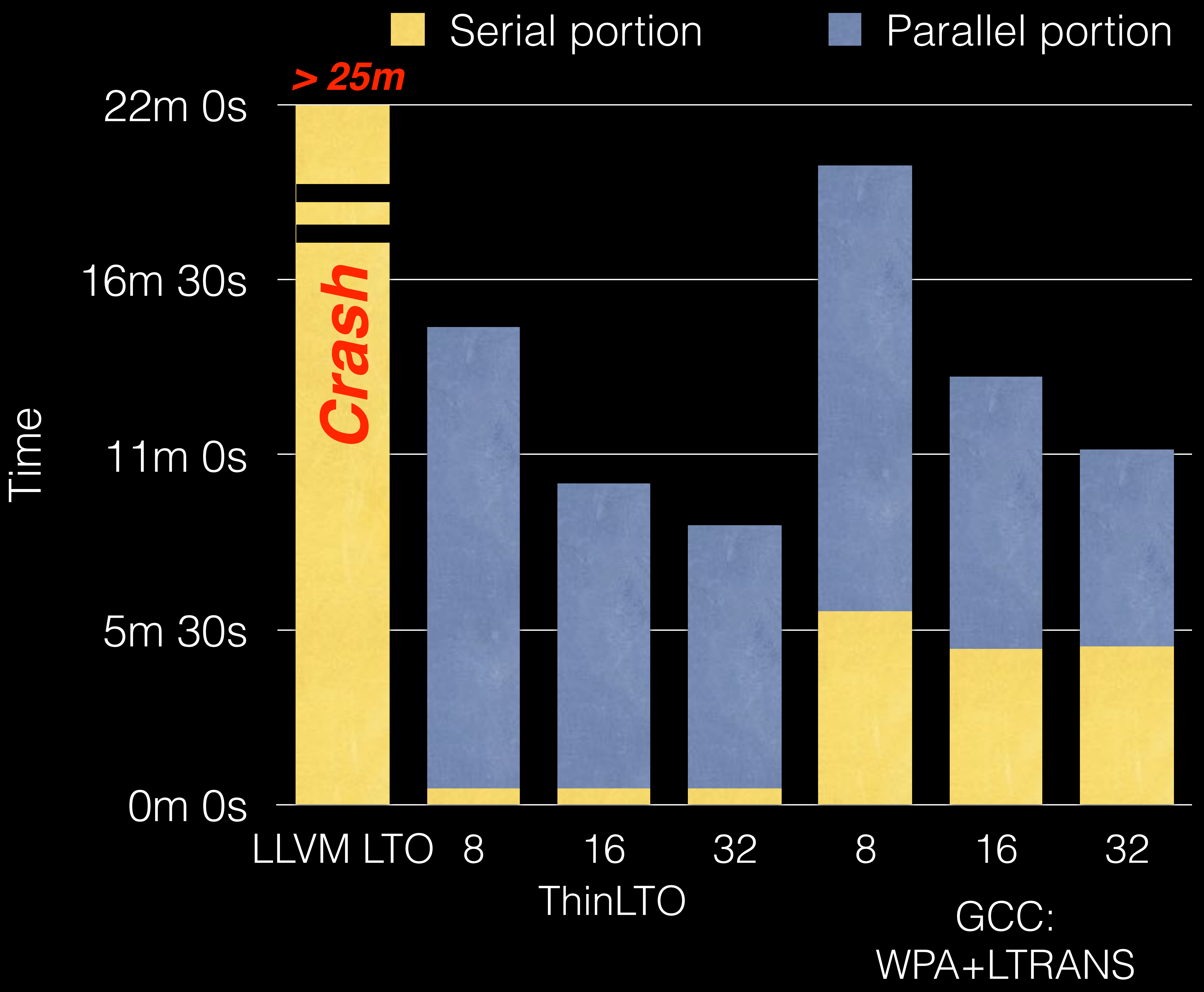
With Debug (-g2)

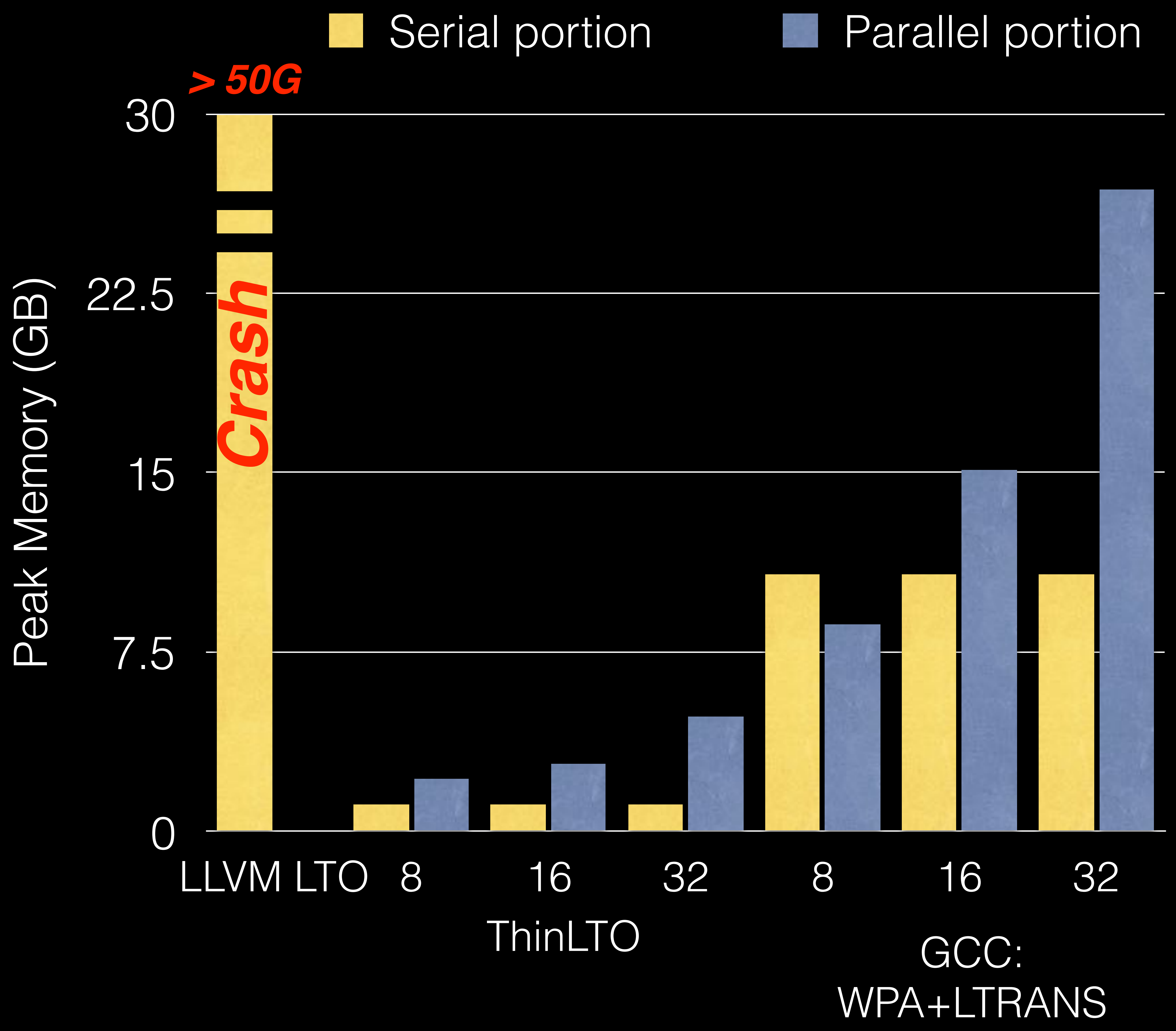
16-core (32-logical) Intel Xeon
E5-2690 @ 2.90GHz

Chromium Build Comparisons

With Debug (-g2)

16-core (32-logical) Intel Xeon E5-2690 @ 2.90GHz





-> *For GCC, time versus memory tradeoff is much more severe!*

Run-time Performance: SPEC cpu2006

Improvement over -O2 (no PGO)

Binary/Text Size: SPEC cpu2006 471.gcc

	Binary (MB)			Text (MB)		
Type	LTO	ThinLTO	*Increase*	LTO	ThinLTO	*Increase*
-g0	4276168	4438720	*3.80%*	4087383	4161320	*1.81%*
-g	12347112	20794136	*68.41%*	4087655	4161456	*1.81%*

Binary/Text Size: SPEC cpu2006 471.gcc

	Binary (MB)			Text (MB)		
Type	LTO	ThinLTO	*Increase*	LTO	ThinLTO	*Increase*
-g0	4276168	4438720	*3.80%*	4087383	4161320	*1.81%*
-g	12347112	20794136	*68.41%*	4087655	4161456	*1.81%*

- *Despite recent improvements, still too much debug metadata imported during ThinLTO importing!*
- *E.g. composite type definitions*

-> Area of future work