arm

Tutorial: BOLT on AArch64 and how it compares with other PGOs

Paschalis Mpeis
29/10/2025

What this talk covers

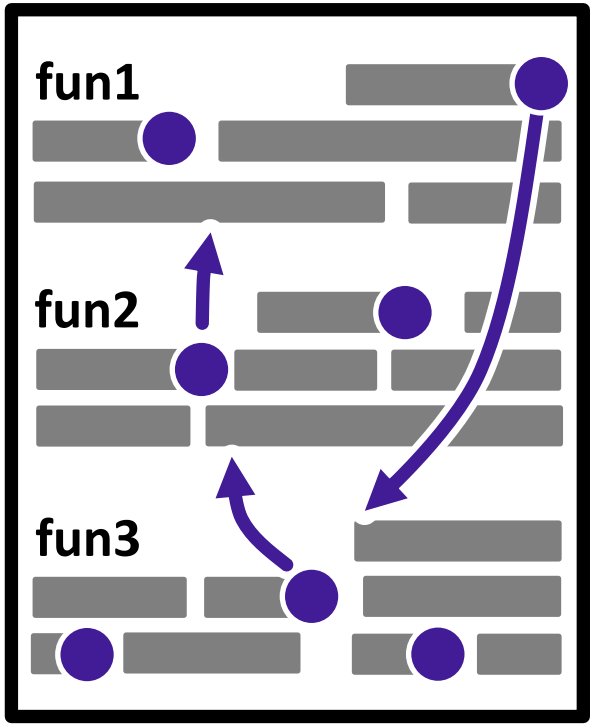
Packed Talk; Q/A at the end!

- Motivation
- Solution
- BOLT
- LLVM PGOs
- BOLT: Quick Demo

- **Motivation**
- Solution
- BOLT
- LLVM PGOs
- BOLT: Quick Demo

What is the problem?

- binary



hot: executed often
cold: not executed often

CPU

instruction cache

hot path travels long distances

==

bad spatial locality!

How we know we have it?

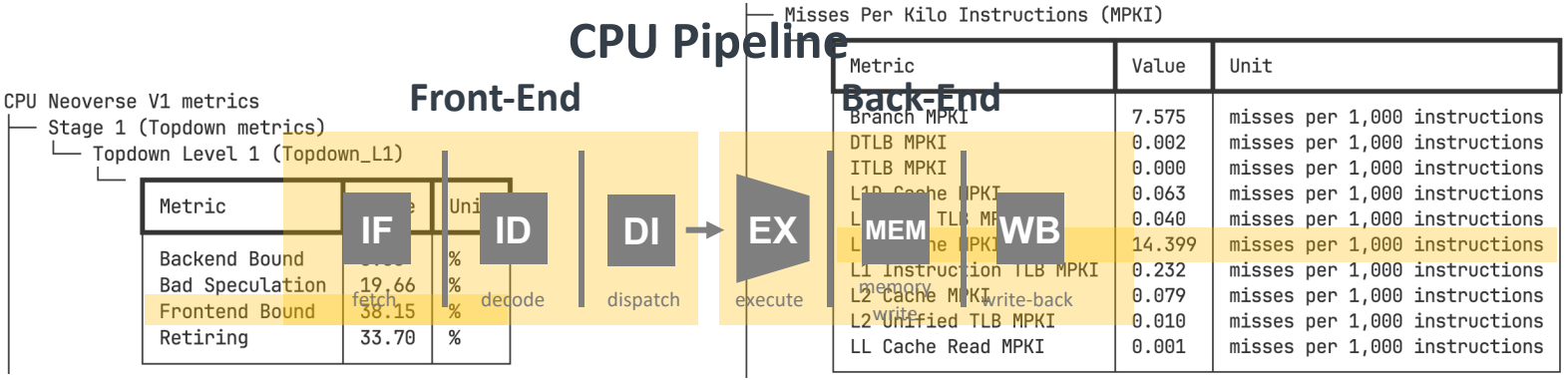
Hardware Metrics

Top-Down Methodology

- Front-End Bound Workloads (**>10% FE**)
- Back-end waits for *uOps* to come in
- Misses on i-cache, i-TLB, branch-prediction

Micro-Architecture Exploration

- BOLT's rule-of-thumb: **>10 L1i MPKI**



arm

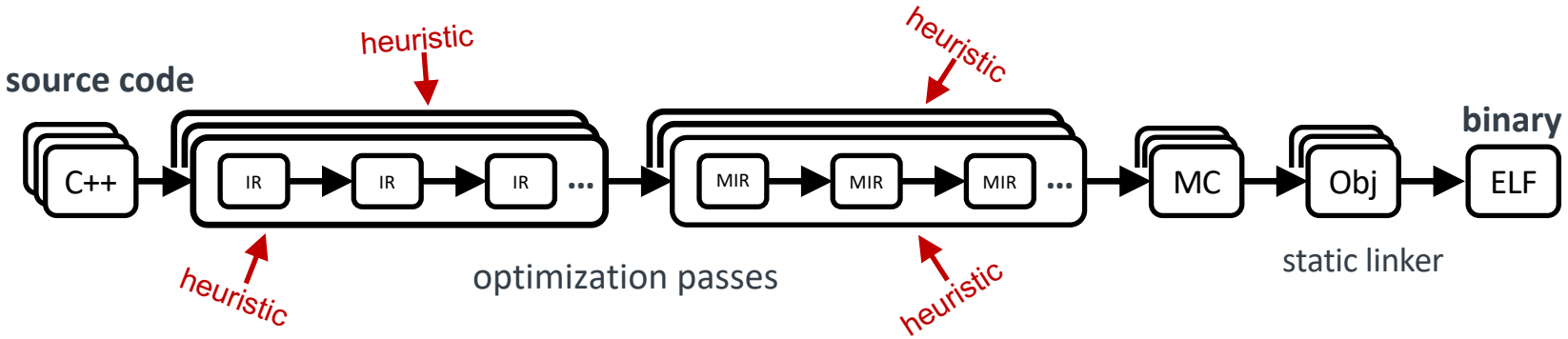
arm ATP: Arm Total Performance

~~topdown-tool~~ ./binary

perf stat -e instructions,L1-icache-misses -- binary

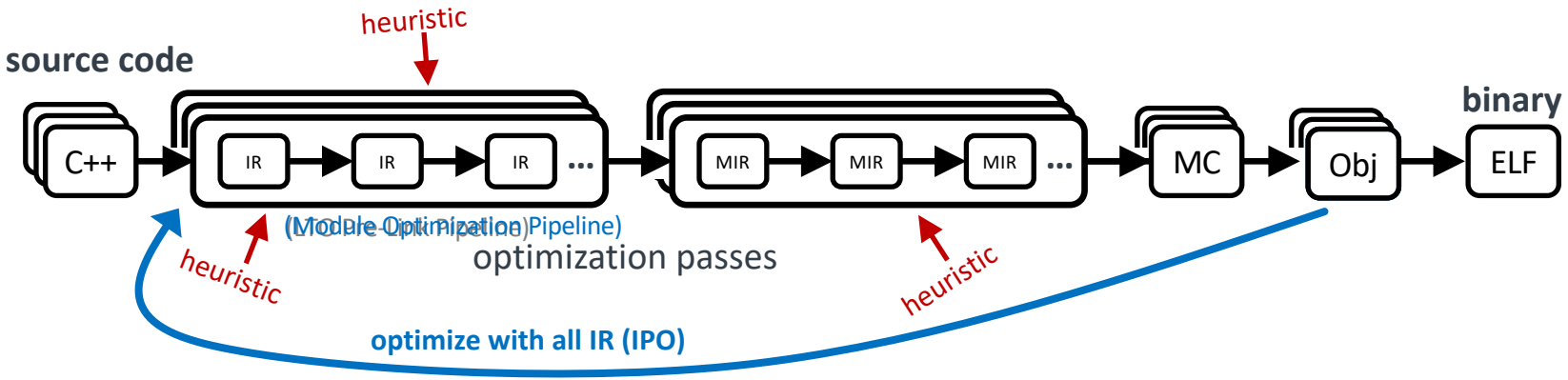
Why compilers alone fall short

Compilation flow in Clang/LLVM:



Why compilers alone fall short

LTO flow in Clang/LLVM:



(1) Reorder code

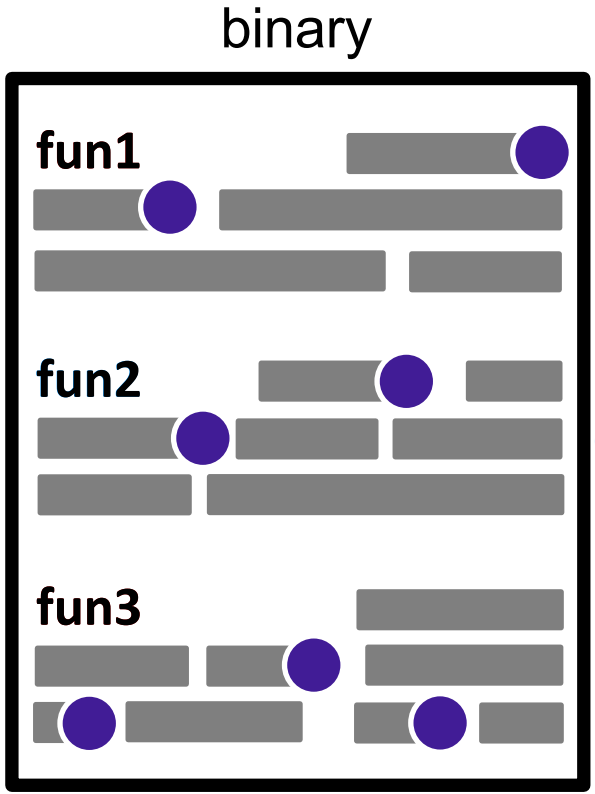
(2) Get profile data!

- **Motivation**
- Solution
- BOLT
- LLVM PGOs
- BOLT: Quick Demo

- Motivation
- **Solution**
- BOLT
- LLVM PGOs
- BOLT: Quick Demo

Reorder Code

Code Layout Trinity: (1) Function Reordering

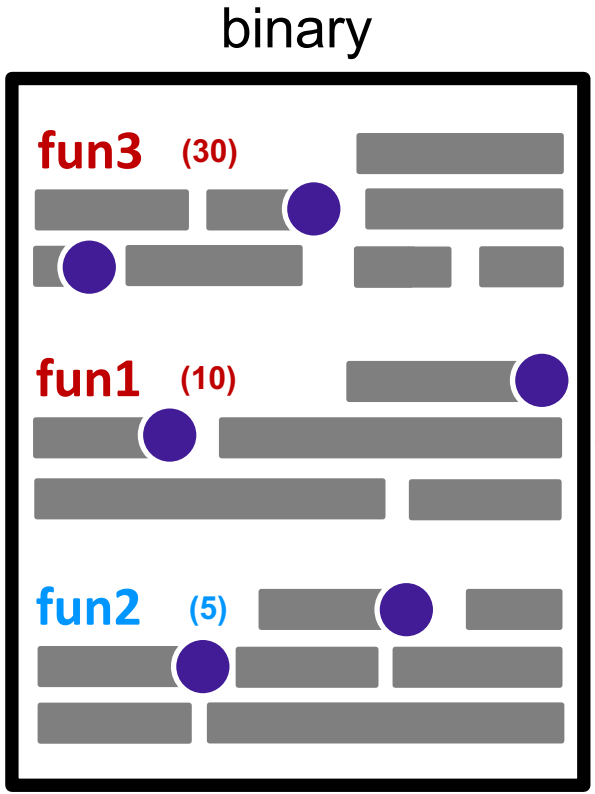


hot: executed often

cold: not executed often

Reorder Code

Code Layout Trinity: (1) Function Reordering

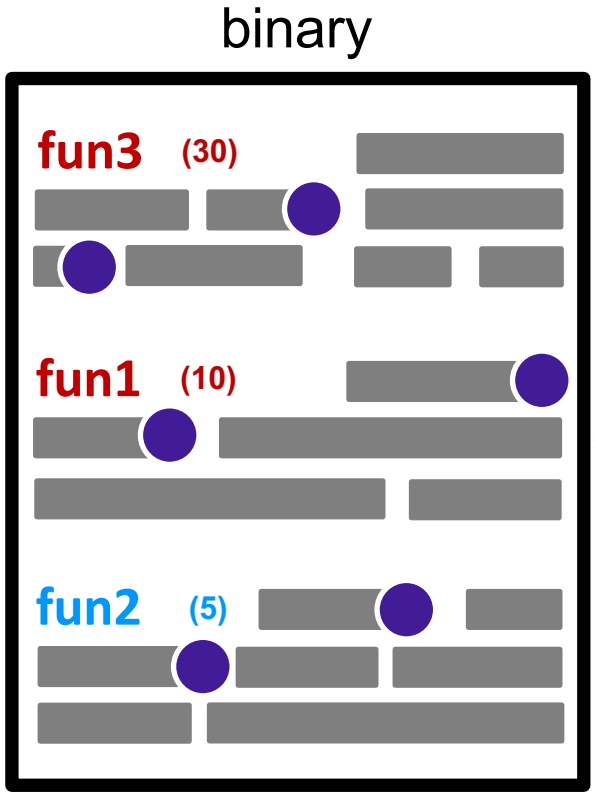


hot: executed often

cold: not executed often

Reorder Code

Code Layout Trinity: (2) Block Reordering

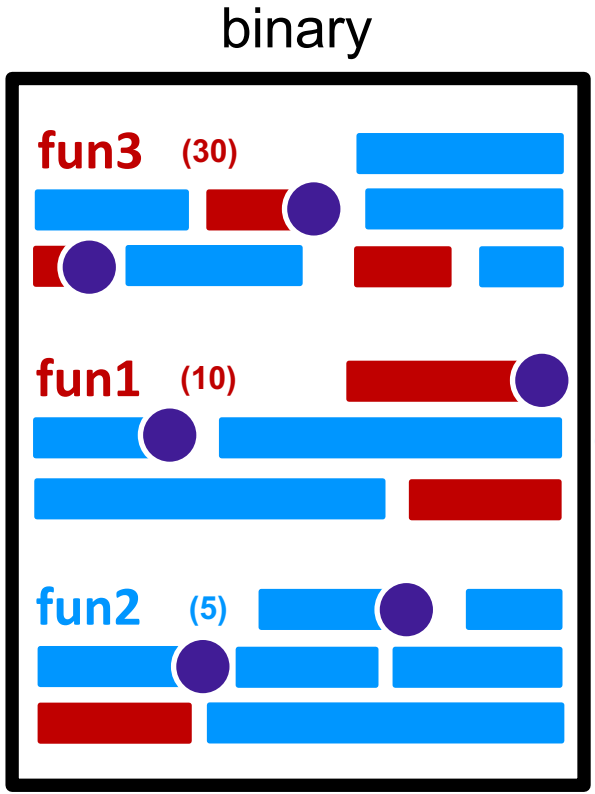


hot: executed often

cold: not executed often

Reorder Code

Code Layout Trinity: (2) Block Reordering



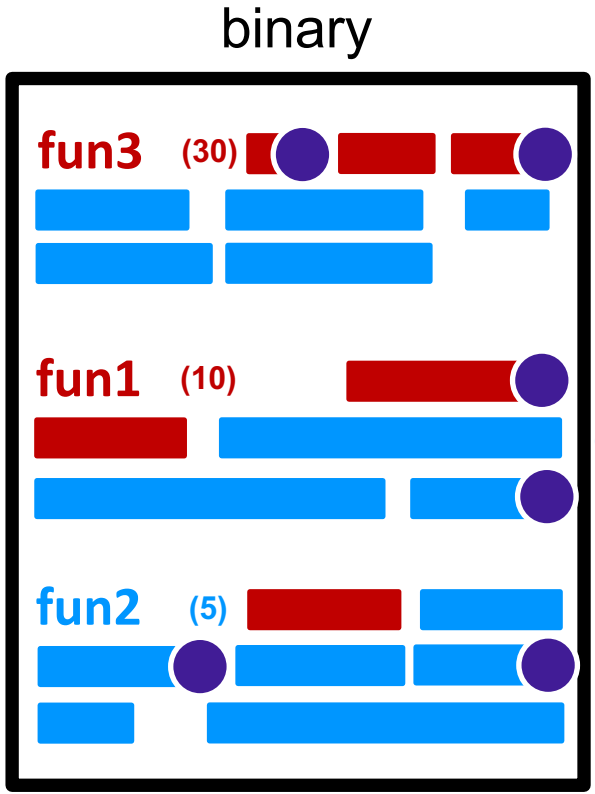
hot: executed often

cold: not executed often

Reorder Code

Code Layout Trinity: (2) Block Reordering

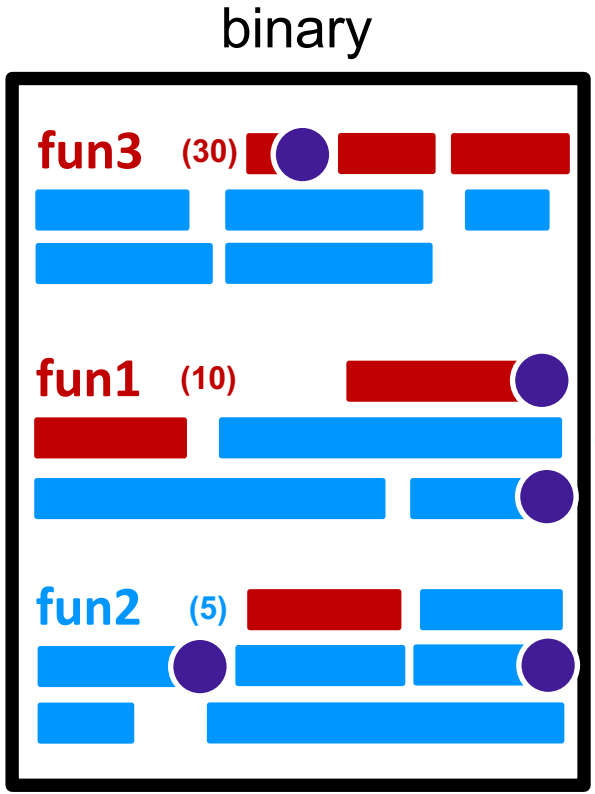
- fall-through branches cost less
- Travelling Salesman Problem



hot: executed often
cold: not executed often

Reorder Code

Code Layout Trinity: (3) Function Splitting

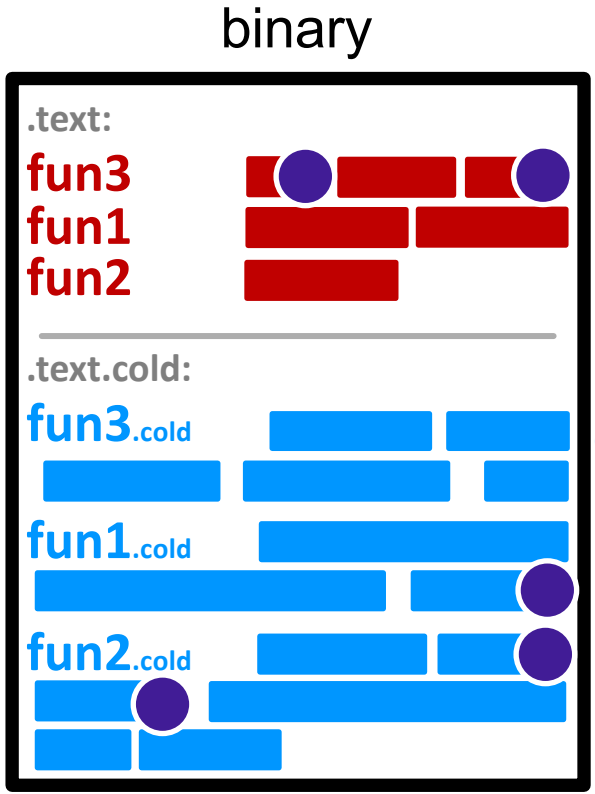


hot: executed often

cold: not executed often

Reorder Code

Code Layout Trinity: (3) Function Splitting



hot: executed often

cold: not executed often

- **minimal section** of hot code
- usually, a small subset of code **dominates execution**

Profiling

Profile Guided Optimization

- PGO, FDO (Feedback-Driven)
- Way more than Code-Layout

rg "BlockFrequencyInfo" -l | grep Reg

```
llvm/lib/CodeGen/MLRegAllocEvictAdvisor.cpp
llvm/lib/CodeGen/RegAllocScore.cpp
llvm/lib/CodeGen/GlobalISel/RegBankSelect.cpp
llvm/lib/CodeGen/RegAllocBasic.cpp
llvm/lib/CodeGen/RegAllocPBQP.cpp
llvm/lib/CodeGen/MLRegAllocPriorityAdvisor.cpp
llvm/lib/CodeGen/RegAllocGreedy.cpp
llvm/lib/CodeGen/RegAllocScore.h
llvm/lib/CodeGen/RegAllocGreedy.h
llvm/lib/Target/WebAssembly/WebAssemblyRegColoring.cpp
llvm/lib/Target/WebAssembly/WebAssemblyRegStackify.cpp
llvm/include/llvm/Passes/MachinePassRegistry.def
llvm/include/llvm/CodeGen/GlobalISel/RegBankSelect.h
llvm/include/llvm/CodeGen/RegAllocPBQP.h
```

rg "BranchProbabilityInfo" -l | grep Reg

```
llvm/lib/CodeGen/GlobalISel/RegBankSelect.cpp
llvm/include/llvm/CodeGen/GlobalISel/RegBankSelect.h
```

LLVM: rg "ProfileSummaryInfo" -l

Analysis:
OptimizationRemarkEmitter
Inlining: • InlineOrder • InlineAdvisor • InlineCostMLInlineAdvisor

Transformations:
InstructionCombining
LoopVectorize
InlineFunction
Scalar:
ConstantHoisting
Loops: • SimpleLoopUnswitch • LoopLoadElimination • LoopUnrollPass
CodeGen:
IfConversion
BranchFolding

Transformations (cont..)
MachineFunctionSplitter
SelectOptimize
MachineCombiner
ExpandMemCmp
MachineBlockPlacement
TailDuplication
MachineSizeOpts
Instruction Selection:
InstructionSelect (GlobalISel)
SelectionDAGISel (SelectionDAG)

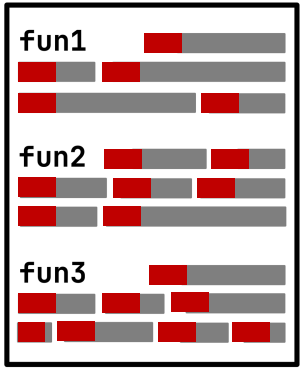
Inter-Procedural:
Inlining: • Inliner • ModuleInliner • PartialInlining • AlwaysInliner
HotColdSplitting

Profiling

Methods

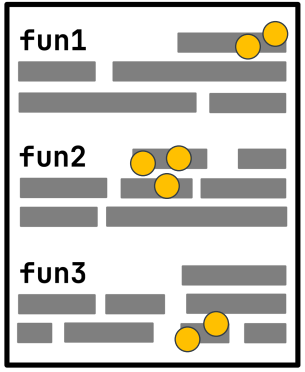
Instrumentation

- Modifies input binary to record **accurate counts**
- Operates at different granularity
- Theoretical optimal quality
- Comes at a **high cost**!



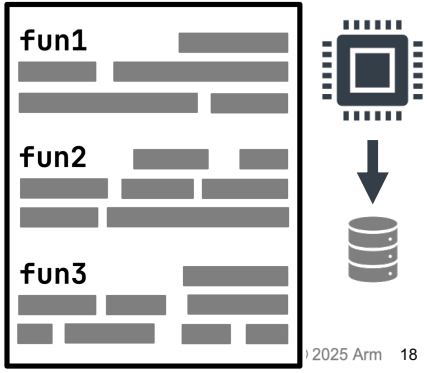
Sampled-Based

- Hot paths show in samples
- **Approximate counts** by periodically observing code
- Less data → similar conclusions
- Different HW units: **BRBE**, **SPE**
- **Faster**; quality impact varies



Tracing

- Records flow and data traces
- HW Units: ETM / ETE
- **Big volumes** of events
- Strobing can help



Profiling

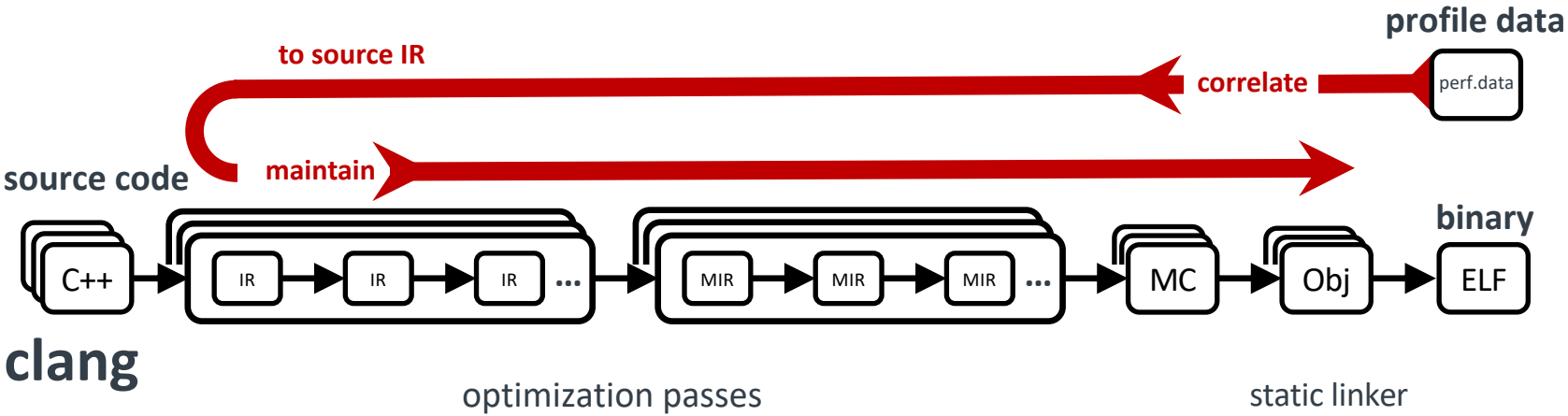
Challenges

- Environment restrictions (mobile, server, production)
- Hardware restrictions
- Trade-off: Profile Quality vs Overheads
- Workloads and Code keeps changing
- Profile Correlation and Maintenance

```
3529      if (match(Cond, m_Not(m_Value(X))) && !isa<Constant>(X)) {
3530        // Swap Destinations and condition...
3531        BI.swapSuccessors();
3532  +     if (BPI)
3533  +       BPI->swapSuccEdgesProbabilities(BI.getParent());
3534      return replaceOperand(BI, 0, X);
3535    }
```

```
9   - ; CHECK-NEXT:   edge %entry -> %bb2 probability is 0x06186186 / 0x80000000 = 4.76%
10  - ; CHECK-NEXT:   edge %entry -> %bb1 probability is 0x79e79e7a / 0x80000000 = 95.24% [HOT edge]
```

```
9   + ; CHECK-NEXT:   edge %entry -> %bb2 probability is 0x79e79e7a / 0x80000000 = 95.24% [HOT edge]
10  + ; CHECK-NEXT:   edge %entry -> %bb1 probability is 0x06186186 / 0x80000000 = 4.76%
```

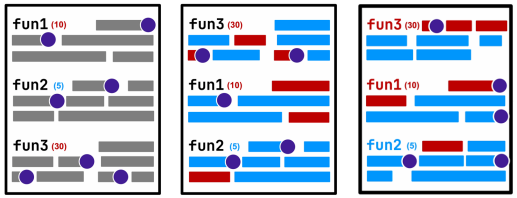


- Motivation
- **Solution**
- BOLT
- LLVM PGOs
- BOLT: Quick Demo

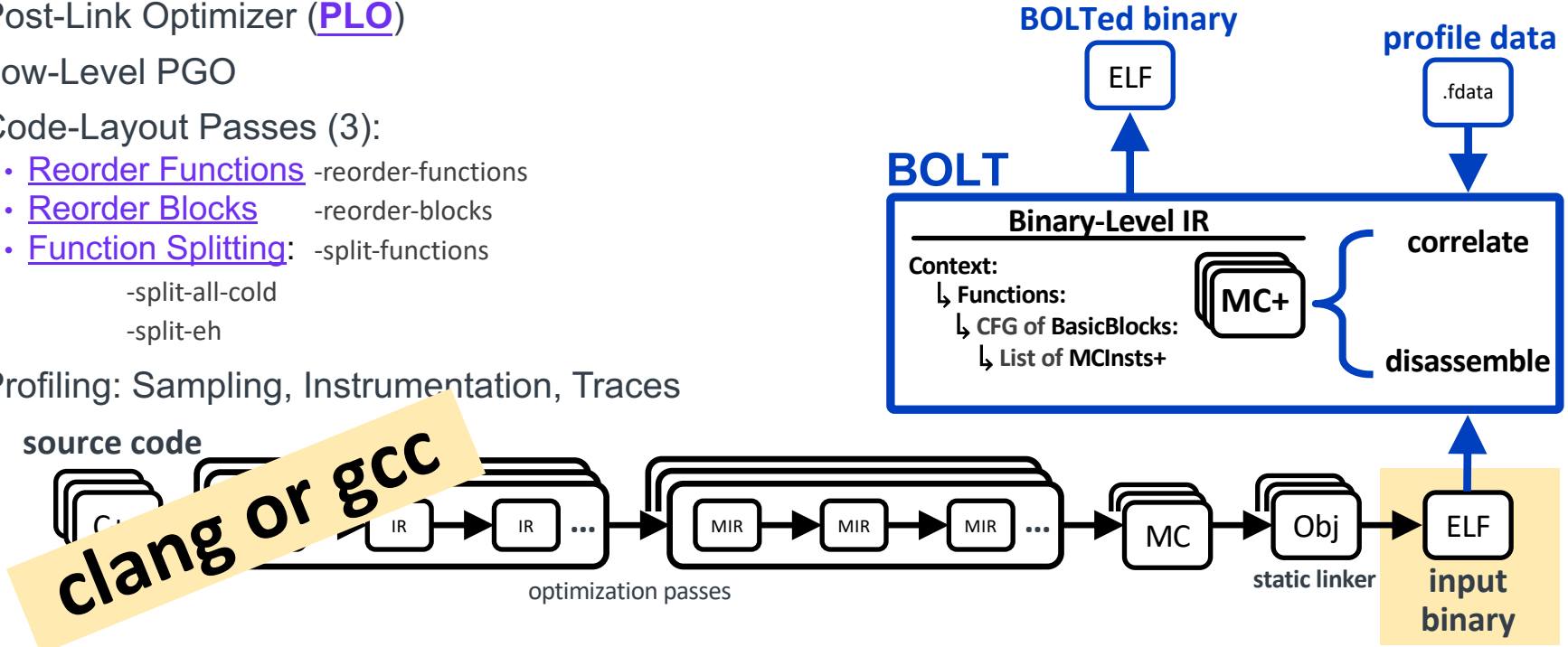
- Motivation
- Solution
- ➢ **BOLT**
- LLVM PGOs
- BOLT: Quick Demo

BOLT Introduction

Binary Optimization and Layout Tool



- Created by **Meta**; Used >10 years in production
- Post-Link Optimizer (**PLO**)
- Low-Level PGO
- Code-Layout Passes (3):
  - Reorder Functions -reorder-functions
  - Reorder Blocks -reorder-blocks
  - Function Splitting: -split-functions
    -split-all-cold
    -split-eh
- Profiling: Sampling, Instrumentation, Traces



BOLT vs Other PLOs

A lightning comparison

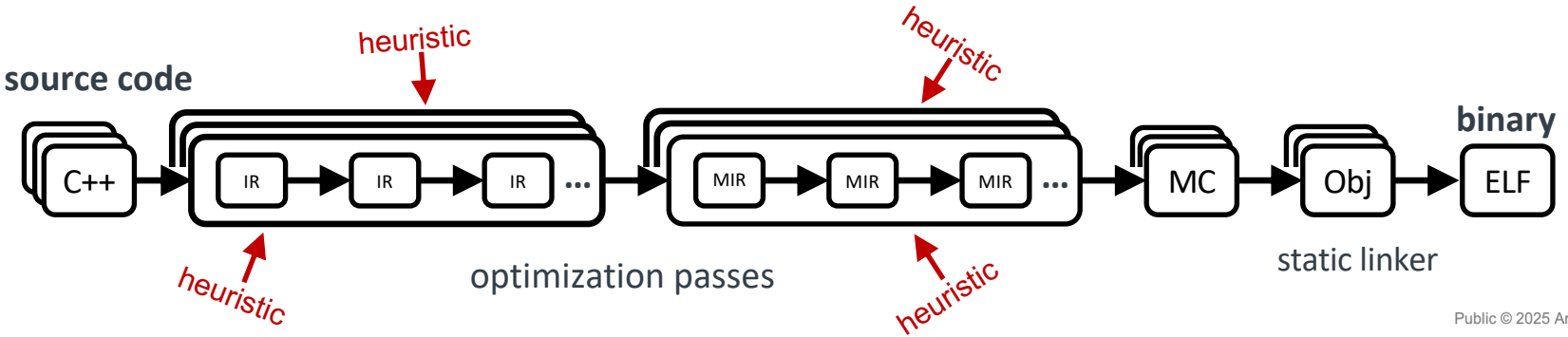
	BOLT	CodeLocality	Propeller
Input Binary Requirements			
Code-Layout Reordering			
Compiler Support			
Binary Rewriting			
	-Wl,--emit-relocs -plt –icf -hugify -inline-memcpy …	-ffunction-sections -symbol-ordering-file	-fbasic-block-sections

- Motivation
- Solution
- ➢ **BOLT**
- LLVM PGOs
- BOLT: Quick Demo

- Motivation
- Solution
- BOLT
- ➢ **LLVM PGOs**
- BOLT: Quick Demo

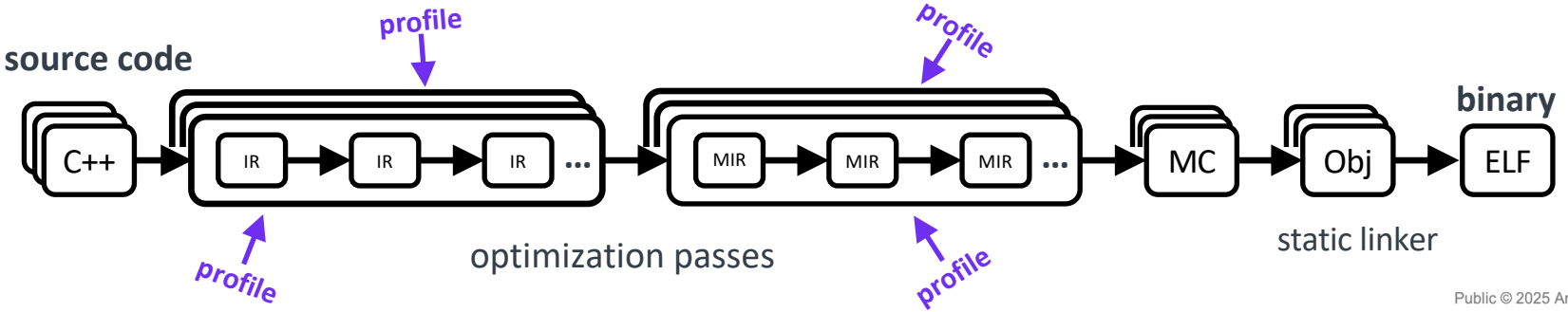
LLVM PGO

Clang



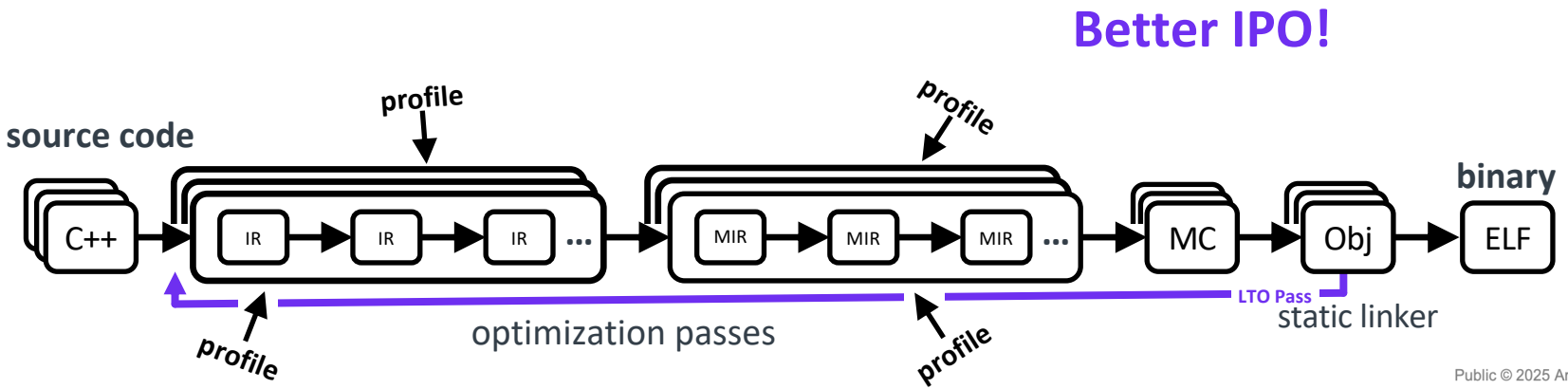
LLVM PGO

Clang+PGO



LLVM PGO

Clang+PGO+LTO



LLVM PGO

Profiling Techniques

- Sampling: S-PGO (A-FDO) **brstack**
- Instrumentation

BRBE: 0xA00 0xB00 0xC00 0xC01 0xC02 0xC03 0xD00 ...

taken branches sample control-flow and call graphs

cond (taken)

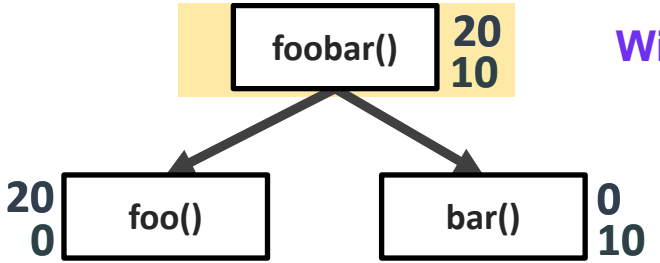
fall-throughs

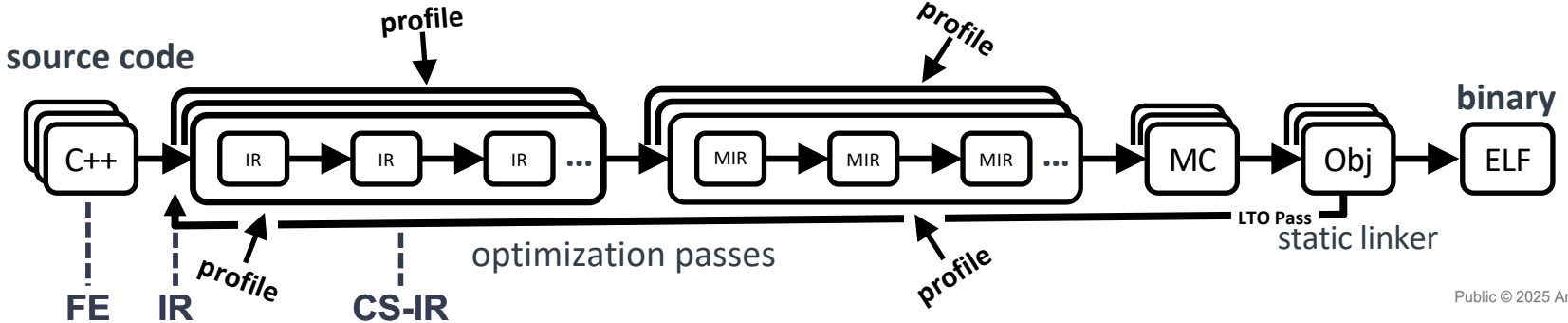
-fprofile-instr-generate
-fprofile-generate
-fcs-profile-generate

Context Sensitivity

With CS

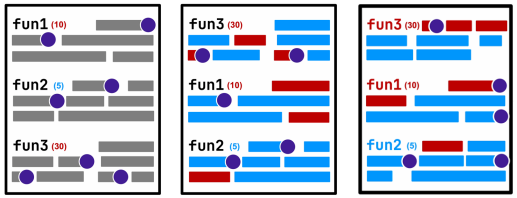
Without CS



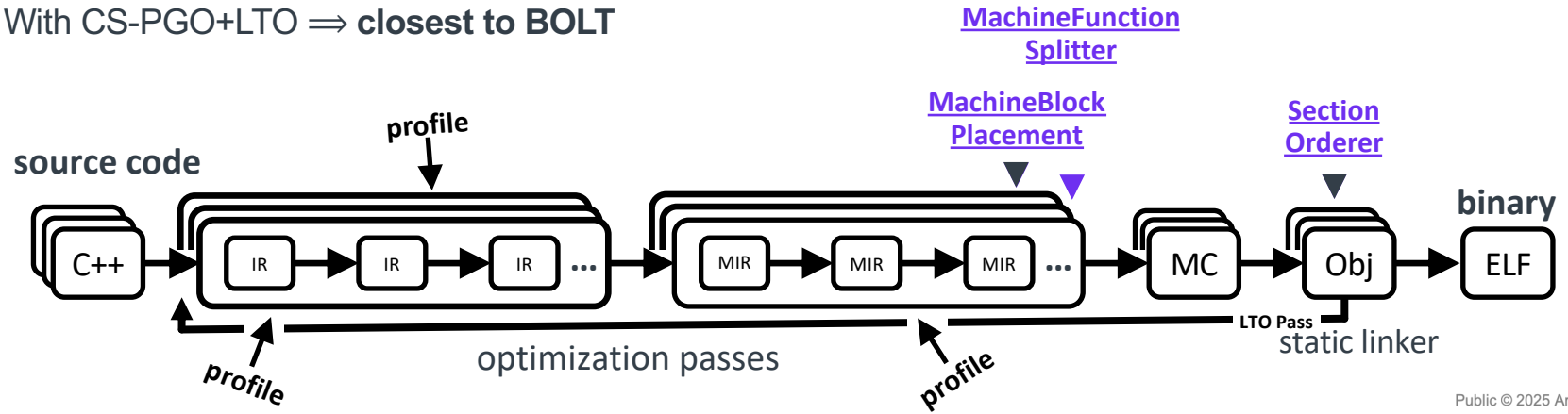


LLVM PGO

Code Layout



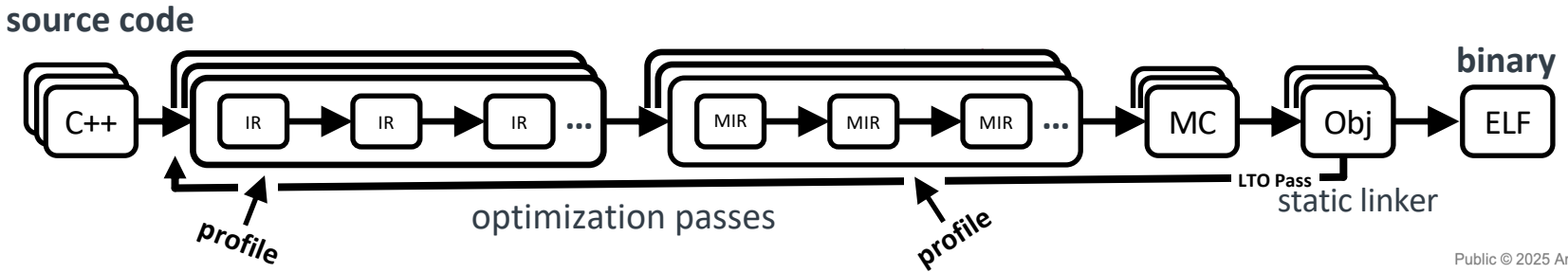
- Function Reordering
  - IR-PGO: default=ON
  - Can manually pass order to LLD -symbol-ordering-file
- Block Reordering
  - We need some layout anyway
  - CS-PGO runs Ext-TSP ordering algorithm -enable-ext-tsp-block-placement
- Function Splitting
  - Default is OFF -fsplit-machine-functions
  - With CS-PGO+LTO ⟹ **closest to BOLT**



LLVM PGO

Comparing with BOLT

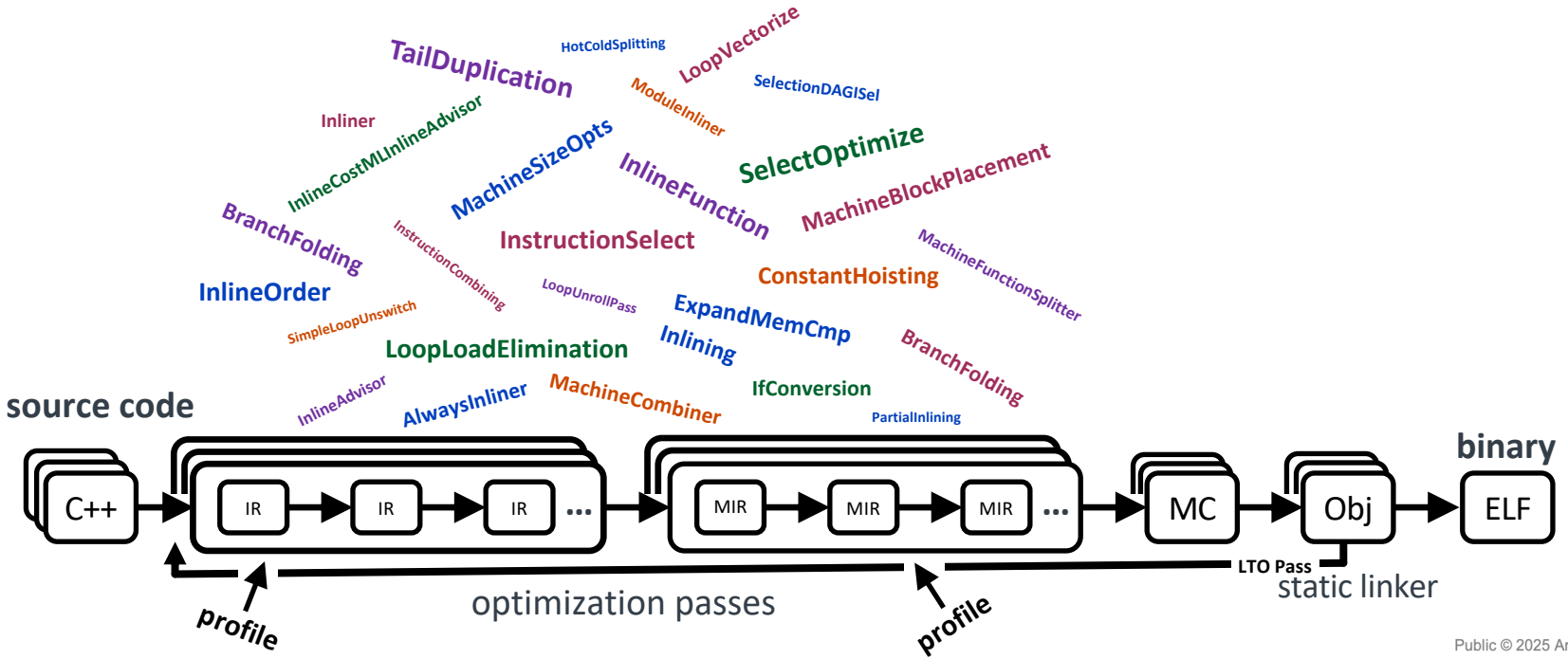
- High-Level PGO → More Powerful



LLVM PGO

Comparing with BOLT

- High-Level PGO → More Powerful



LLVM PGO

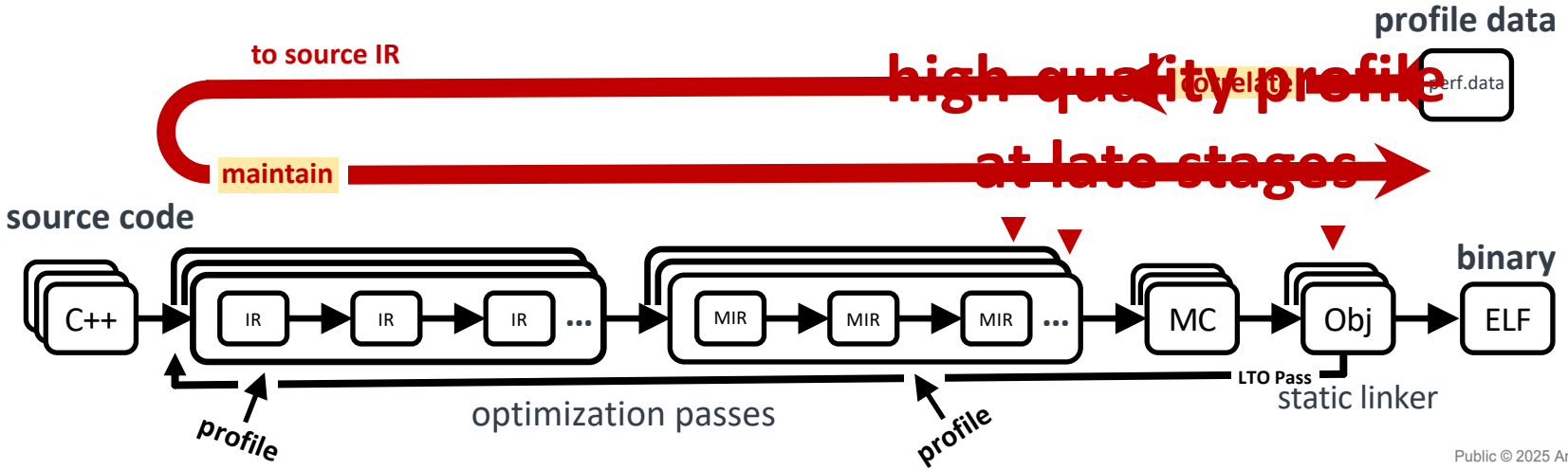
Comparing with BOLT

- High-Level PGO → More Powerful
- Profile Correlation and Maintenance
- Which is better?
- Who should we use?

High-Level PGO vs Low-Level PGO

BOTH

IR-PGO + LTO + BOLT



LLVM PGO

Overview of more PGOs!

- CSS-PGO:
  - Context-Sensitive **Sampling**
  - Combines branch-stacks with call-stacks for CS
  - Uses pseudo-probes to improve correlation
- HW-PGO:
  - Extends S-PGO with data from more HW-Units
    - ifConversion using branch miss/hits information
    - Optimize vtable loads, using memory profiles
- Data Locality
  - Split Data sections (.rodata.hot)
- Temporal-PGO:
  - Mobile apps care about loading time
  - Not CPU-Bound workloads
  - Decrease startup page-faults

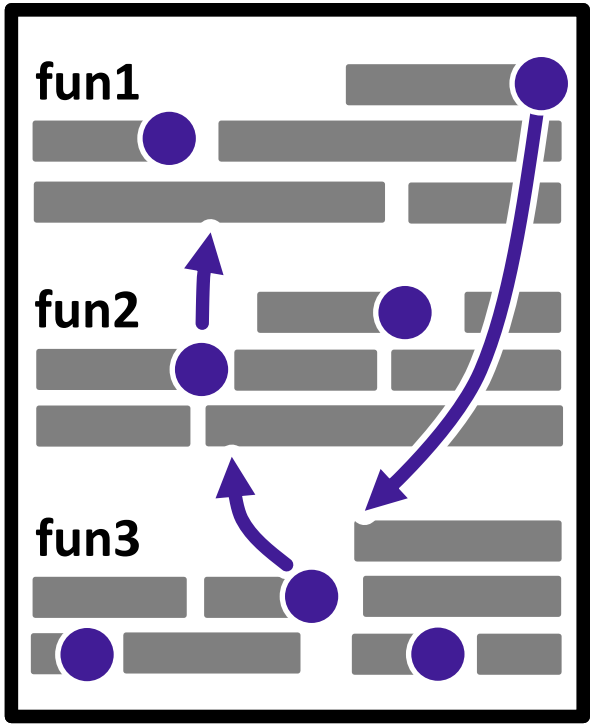
- Motivation
- Solution
- BOLT
- ➢ **LLVM PGOs**
- BOLT: Quick Demo

- Motivation
- Solution
- BOLT
- LLVM PGOs

➢ **BOLT: Quick Demo**

BOLT: Quick Demo

Optimize a pathological case



BOLT: Quick Demo

Bubble Sort

```
while (change):

  for (i: 1-N):

    if (a[i] < a[i-1]):
          t = a[i];
          a[i] = a[i - 1];
          a[i - 1] = t;
          change = T;
```


BOLT: Quick Demo

Bubble Sort

```
while (change):
  for (i: 1-N):
    if (a[i] < a[i-1]):
          swap(a,change);
```

```
swap(a,change):
    t = a[i];
    a[i] = a[i - 1];
    a[i - 1] = t;
    change = T;
```


BOLT: Quick Demo

Bubble Sort

```
while (change):
  for (i: 1-N):
    if (a[i] < a[i-1]):
          swap(a,change);
```

```
swap(a,change):
    if (COND) NOPS(N) else      t = a[i];
    if (COND) NOPS(N) else      a[i] = a[i - 1];
    if (COND) NOPS(N) else      a[i - 1] = t;
    if (COND) NOPS(N) else      change = T;
```


BOLT: Quick Demo

Bubble Sort

```
while (change):
  for (i: 1-N):
    if (a[i] < a[i-1]):
      swap(a,change);
```

```
swap(a,change):
  if (COND) NOPS(N) else        t = a[i];
  if (COND) NOPS(N) else        a[i] = a[i - 1];
  if (COND) NOPS(N) else        a[i - 1] = t;
  if (COND) NOPS(N) else        change = T;
```

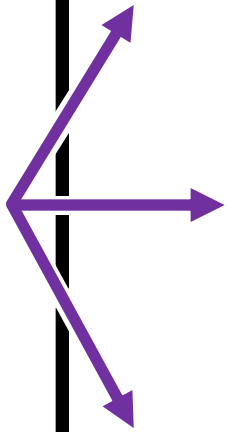
```
swap(a,change):
  if (COND) NOPS(N) else        t = a[i];
  if (COND) NOPS(N) else        a[i] = a[i - 1];
  if (COND) NOPS(N) else        a[i - 1] = t;
  if (COND) NOPS(N) else        change = T;
```

```
swap(a,change):
  if (COND) NOPS(N) else        t = a[i];
  if (COND) NOPS(N) else        a[i] = a[i - 1];
  if (COND) NOPS(N) else        a[i - 1] = t;
  if (COND) NOPS(N) else        change = T;
```

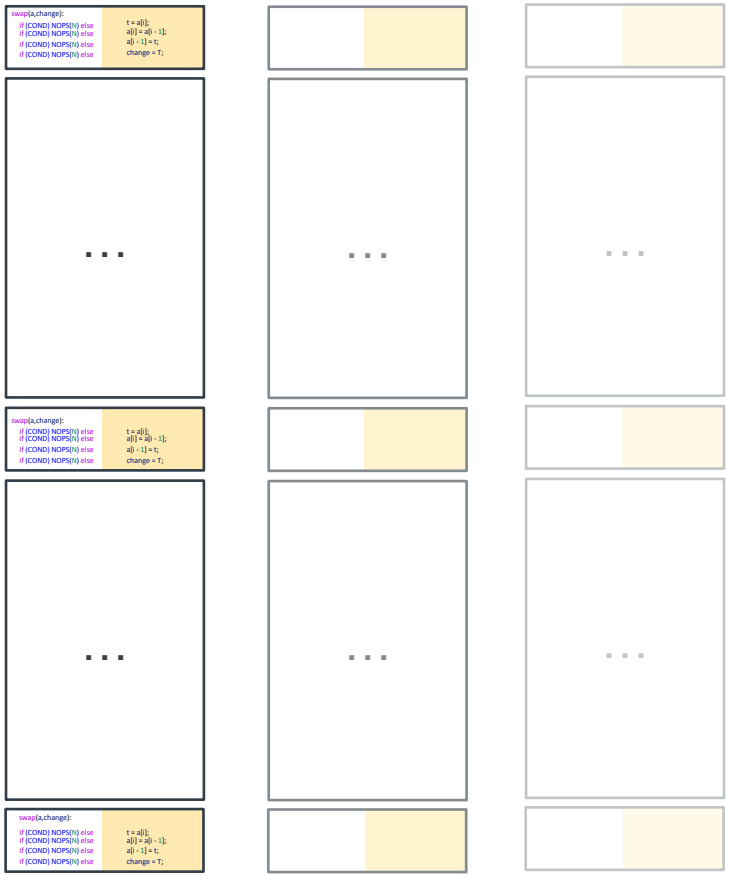
BOLT: Quick Demo

Bubble Sort

```
while (change):
  for (i: 1-N):
    if (a[i] < a[i-1]):
          swap(a,change);
```



Demo Usage & ways of profiling;
Don't focus much on numbers



Verify Effectiveness

Profile Quality

Code Density (based on llvm-profgen):

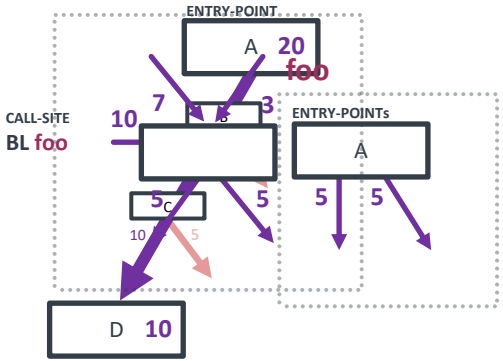
- Warn on insufficient samples

$$\frac{dynamically\ executed\ instructions}{static\ code\ size}$$

Flow Continuity:

- How well-connected profiles are

1. CFG Continuity
2. CG flow conservation
3. CFG flow conservation

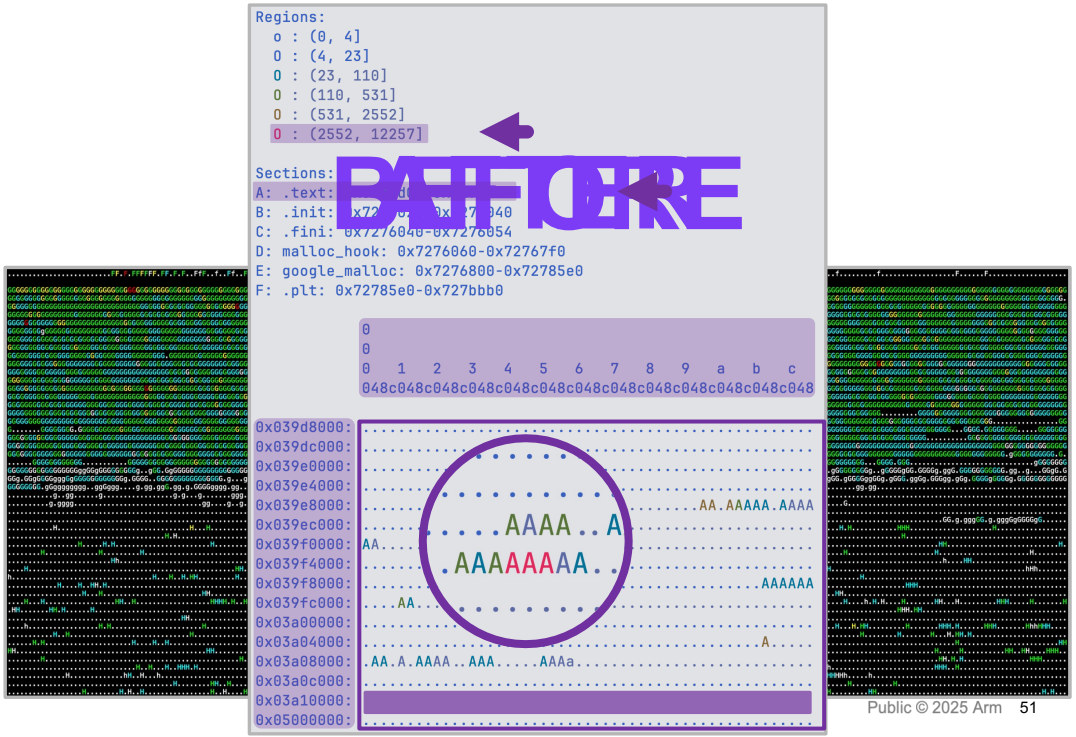


Verify Effectiveness

Heatmap Tool

Visualize samples: [llvm-bolt-heatmap]

- Sample the binary (before / after)
- Run tool
- Address range
- Empty lines: bigger gap
- Character block of code
- Letter when there is a sample
- Color is hotness

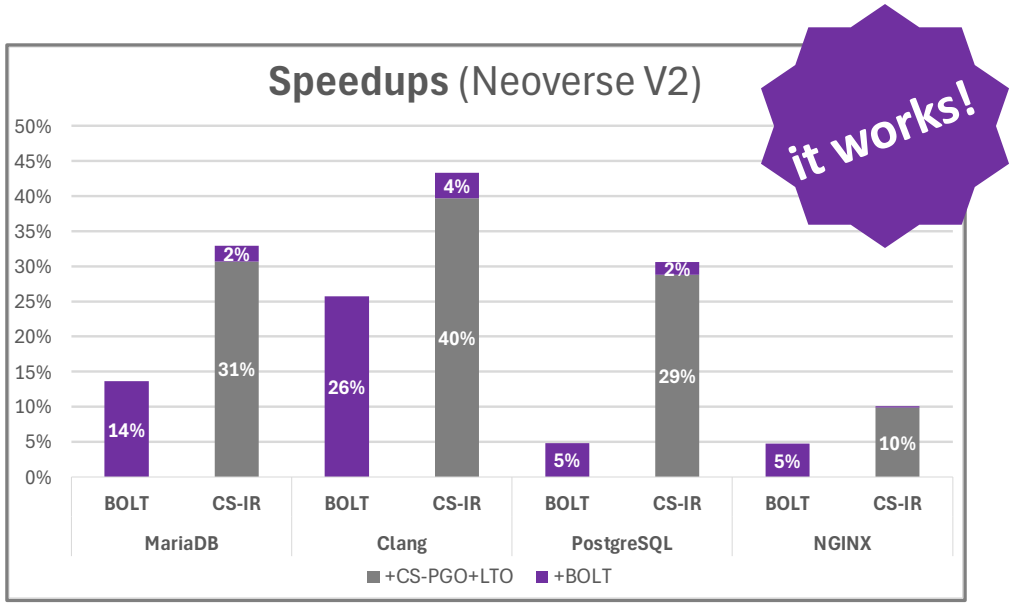


Bonus slide!

- Arm Team, working on BOLT
- Maintainer of AArch64 Backend
- Contribute: Patches, Reviews, Testing
- Active community on LLVM

Try BOLT!

- Discord: **#bolt**
- Office Hours
  - Monthly, 2nd Wednesday
  - **11:00am PT**, 30 mins, **ZOOM**
- **roundtable**



References

- **Tutorial: HelloBOLT: Getting started with BOLT**
- The many faces of LLVM PGO and FDO, Amir, Meta
- LLVM BOLT, Meta
- Arm Top-Down Methodology (Overview), Arm
- Arm Telementry Solution (topdown-tool), Arm
- Understanding Performance, Arm
- Speedup of clang build with PGO and BOLT on AArch64, Arm
- Call-Site Aware PGO (CS-PGO), Huawei
- Context-Sensitive-Sampling (CSS-PGO), Meta
- Control-Flow Sensitive AutoFDO (FS-AFDO), Google
- Optimizing Function Layout for Mobile Application (Temporal PGO), Meta
- HW-PGO extension for S-PGO, Intel
- Profile Guided Static Data Partitioning, Google


arm
Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm