Using Content-Addressable Storage in Clang for Caching Computations and Eliminating Redundancy

Steven Wu, Ben Langmuir

LLVM Developers' Meeting 2022 | Apple, Inc. | November 9, 2022

Vision

Compilers are getting more and more complex

- Using lots of computation, memory and storage

… But much of the work is redundant

- Parsing the same sources, optimizing the same functions, etc.

Vision

Compilers are getting more and more complex

- Using lots of computation, memory and storage

... But much of the work is redundant

- Parsing the same sources, optimizing the same functions, etc.

Our solution to the problem is: **Content Addressable Storage**!

RFC: https://discourse.llvm.org/t/rfc-add-an-llvm-cas-library-and-experiment-with-fine-grained-caching-for-builds/59864

Agenda

What is a CAS?

Clang Caching

CAS ObjectFile Storage

Potential and Future Work

CAS: Content Addressable Storage

Content stored is assigned a unique address based on its content

Widely used, including many build systems

A new concept to introduce to compilers

CAS Characteristics

Uniqueness

- Identical data stored into CAS will be assigned the same address

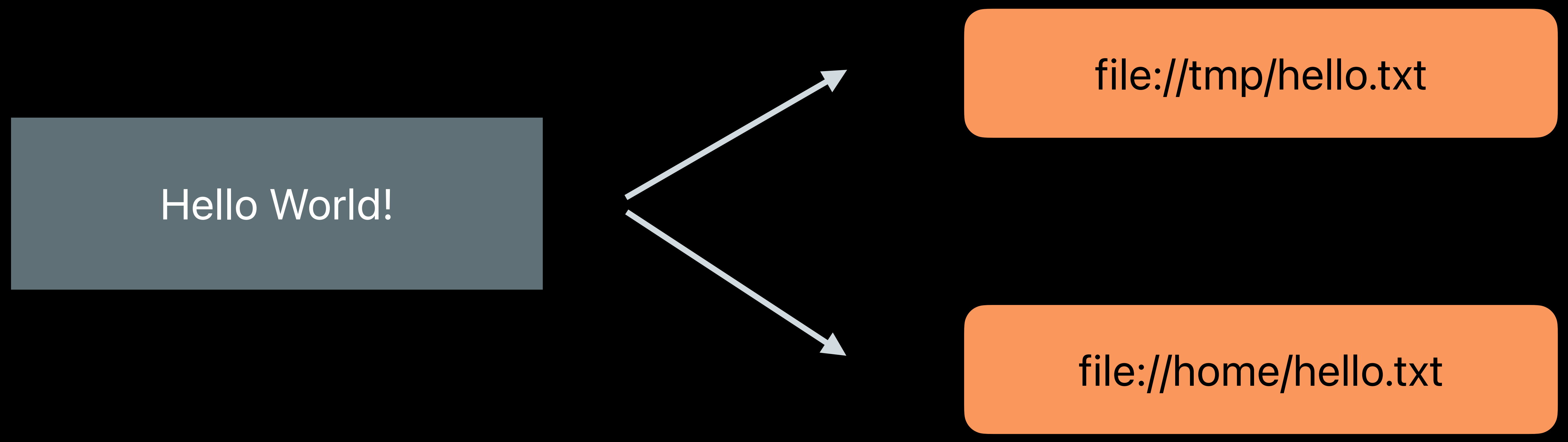
Immutable

- The content assigned to the address cannot be changed

CAS: Data Uniqueness

Uniqueness

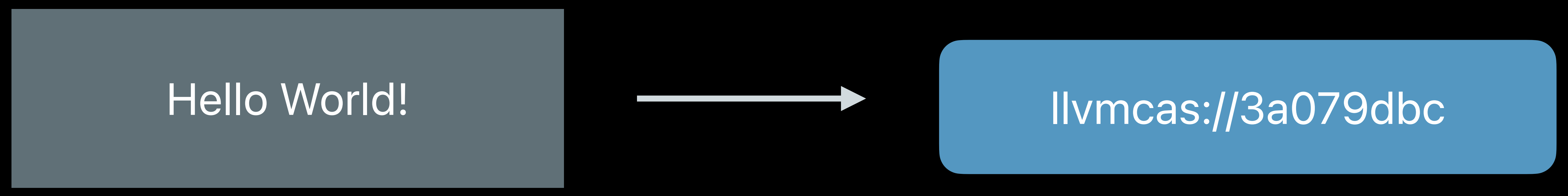
- Identical data stored into CAS will be assigned the same address



CAS: Data Uniqueness

Uniqueness

- Identical data stored into CAS will be assigned the same address



CAS: Immutable Data

Immutable

- The content assigned to an address cannot be changed

Proposed LLVM CAS Library

CAS APIs

- Thread Safe, easy to use for compiler integration
- Extensible for different CAS implementation (e.g. RemoteCAS)
- ObjectStore: Content Addressable Storage
- ActionCache: KeyValue storage to associate inputs and outputs

BuiltinCAS

- A default CAS implementation within LLVM
- Good performance for production usage

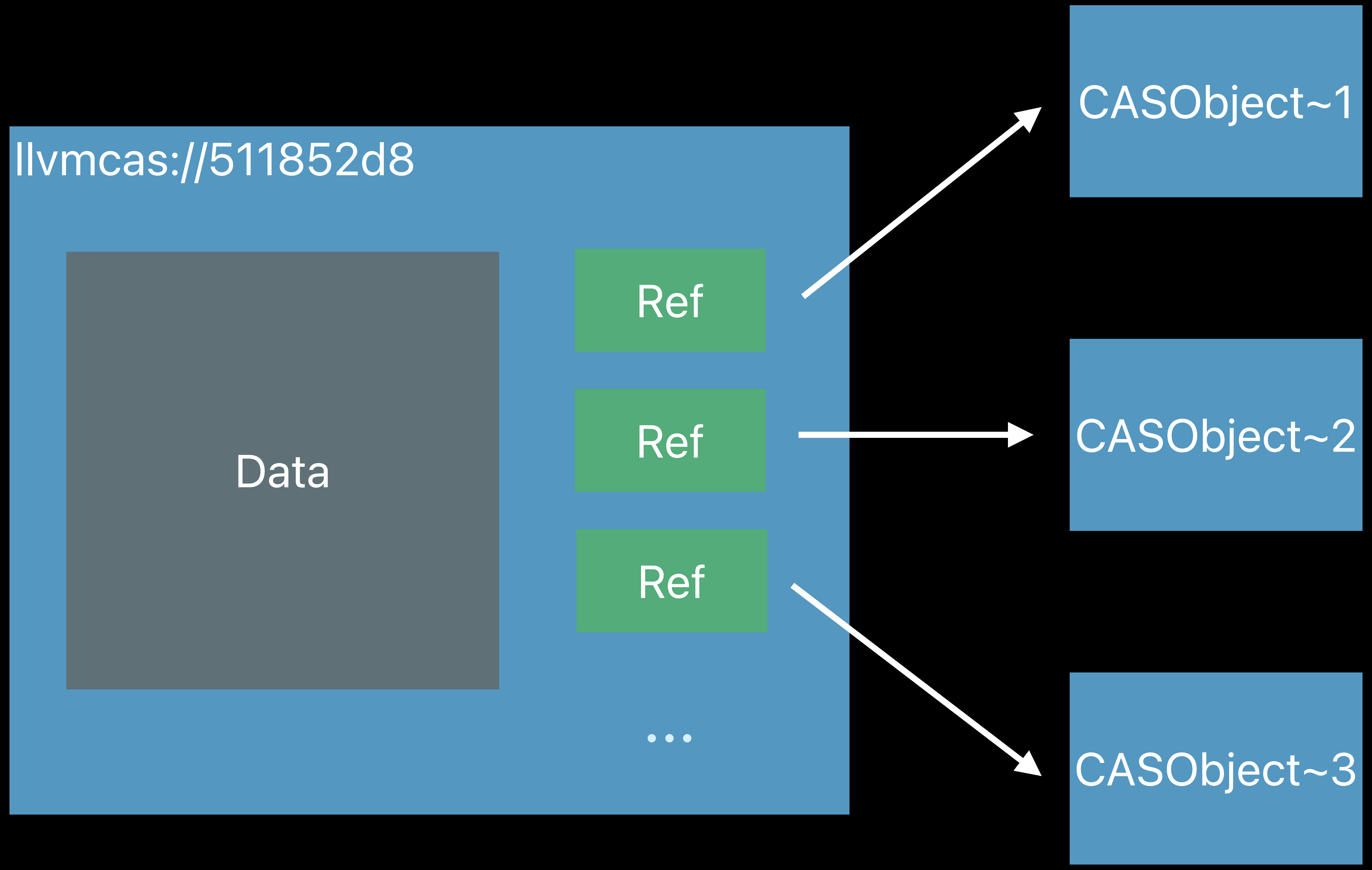
CASObject Model

CASObject contains:

- Data blob
- [Ref]

Ref points to another CASObject

The address of CASObject is computed based on both Data and Refs



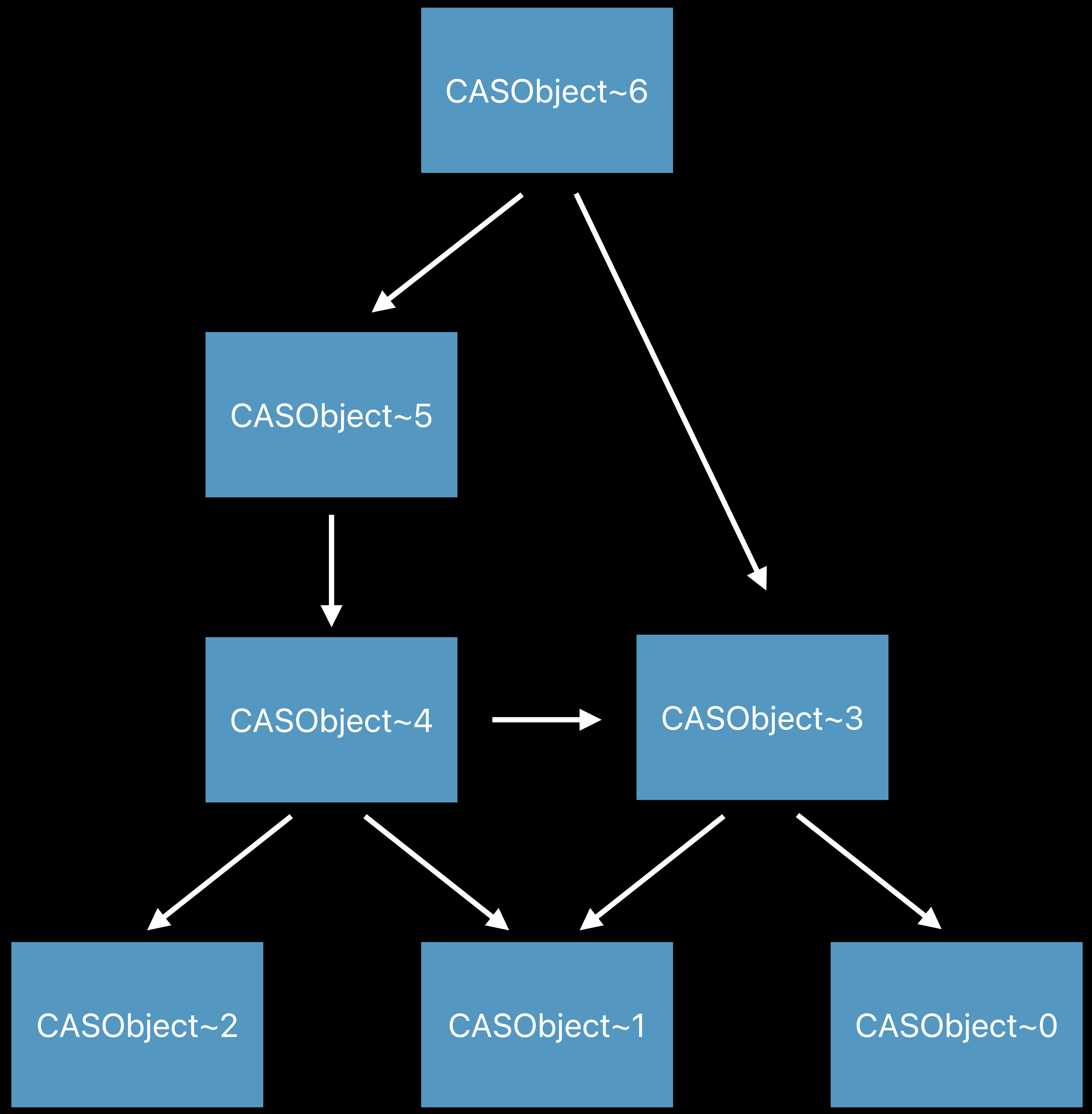
CASObject

CASObject Graphs

Build CASObject graphs from bottom up from leaf nodes

Properties of CASObject graphs

- Direct Acyclic Graph: no cycles
- Easily composable
- Infer equality from the address of the root node

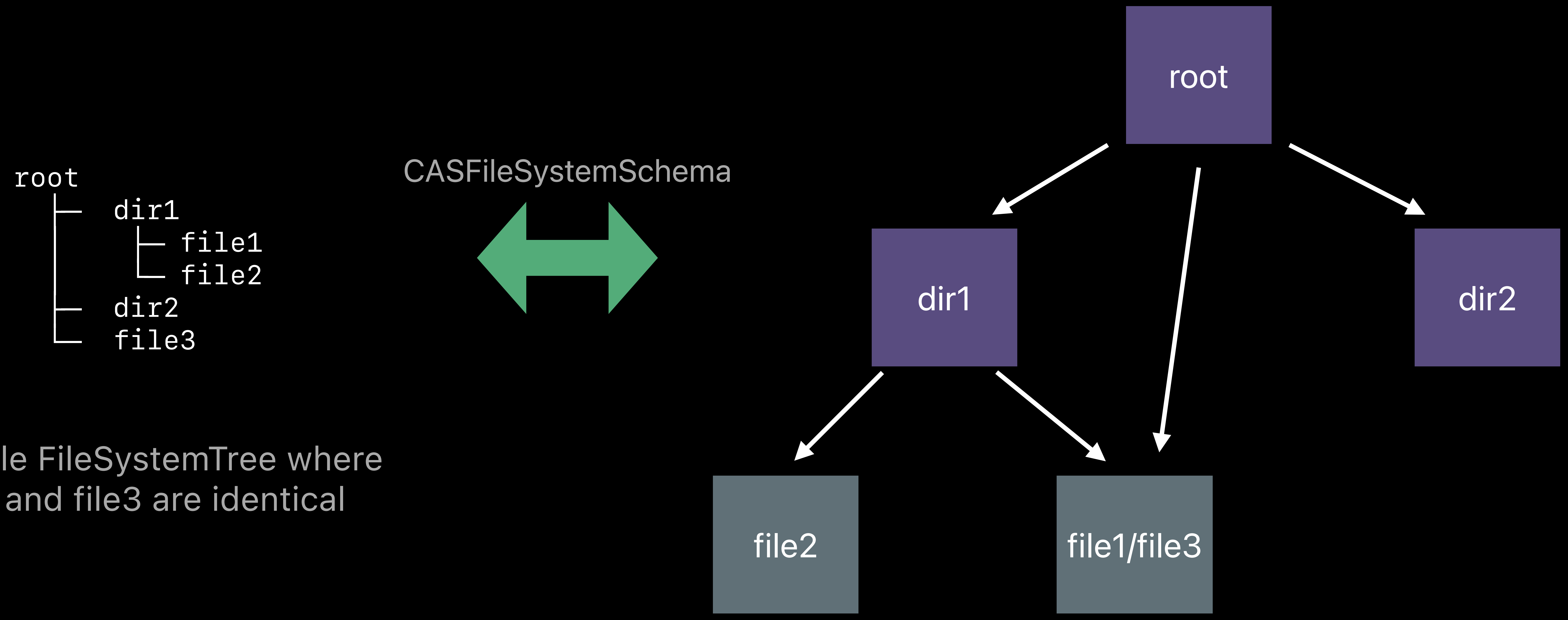


CASSchema

CASSchema: a model used to serialize and deserialize high-level objects from CAS

CASSchema

CASSchema: a model used to serialize and deserialize high-level objects from CAS



Example FileSystemTree where file1 and file3 are identical

Clang Caching

Motivation

Compilers perform redundant work

- Within individual builds
- Across builds

Motivation

Compilers perform redundant work

- Within individual builds
- Across builds

Want to cache computations, but...

- Inputs and outputs not well-specified
- Mutable filesystem

Motivation

Build against persistent CAS

- Isolate pure computations
- Sound and efficient caching by design

Motivation

Build against persistent CAS

- Isolate pure computations
- Sound and efficient caching by design

Example: whole compilation caching

Clang Compilation Caching

Cache compilation of each translation unit

Like "ccache", but with compiler integration

- Sound and efficient caching by design
- Designed to work with modules

Implemented at

- https://github.com/apple/llvm-project/tree/experimental/cas/main

Clang Compilation Caching: Quick Tour

test.h
```
void printHello();
```

test.cpp
```
#include "test.h"
int main() {
  printHello();
  return 0;
}
```

```
clang-cache clang++ -c test.cpp -Rcompile-job-cache
```

Enable Caching

Enable cache-related remarks

Clang Compilation Caching: Quick Tour

test.h

```
void printHello();
```

test.cpp

```
#include "test.h"
int main() {
  printHello();
  return 0;
}
```

```
clang-cache clang++ -c test.cpp -Rcompile-job-cache
remark: compile job cache miss for 'llvmcas://983e09ad7717df57dbcc6906c977e11c769936f5a740aeb46f40a57743abb2ad' [-Rcompile-job-cache-miss]
```

```
clang-cache clang++ -c test.cpp -Rcompile-job-cache
remark: compile job cache hit for 'llvmcas://983e09ad7717df57dbcc6906c977e11c769936f5a740aeb46f40a57743abb2ad' =>
    'llvmcas://a6b70c60e0165e98295f6f2ee40a70e0233748c2bf09ca2d1de0a869d99cca7b' [-Rcompile-job-cache-hit]
```

How does it work?

1. clang-scan-deps discovers inputs; ingest into CAS
2. Produce a `-cc1` command that only accesses the CAS
3. Capture outputs in `VirtualOutputBackend`
4. Fearlessly cache results

clang-scan-deps discovers inputs

clang-scan-deps is a library to discover dependencies

- Finds file and module dependencies
- Much faster than preprocessing

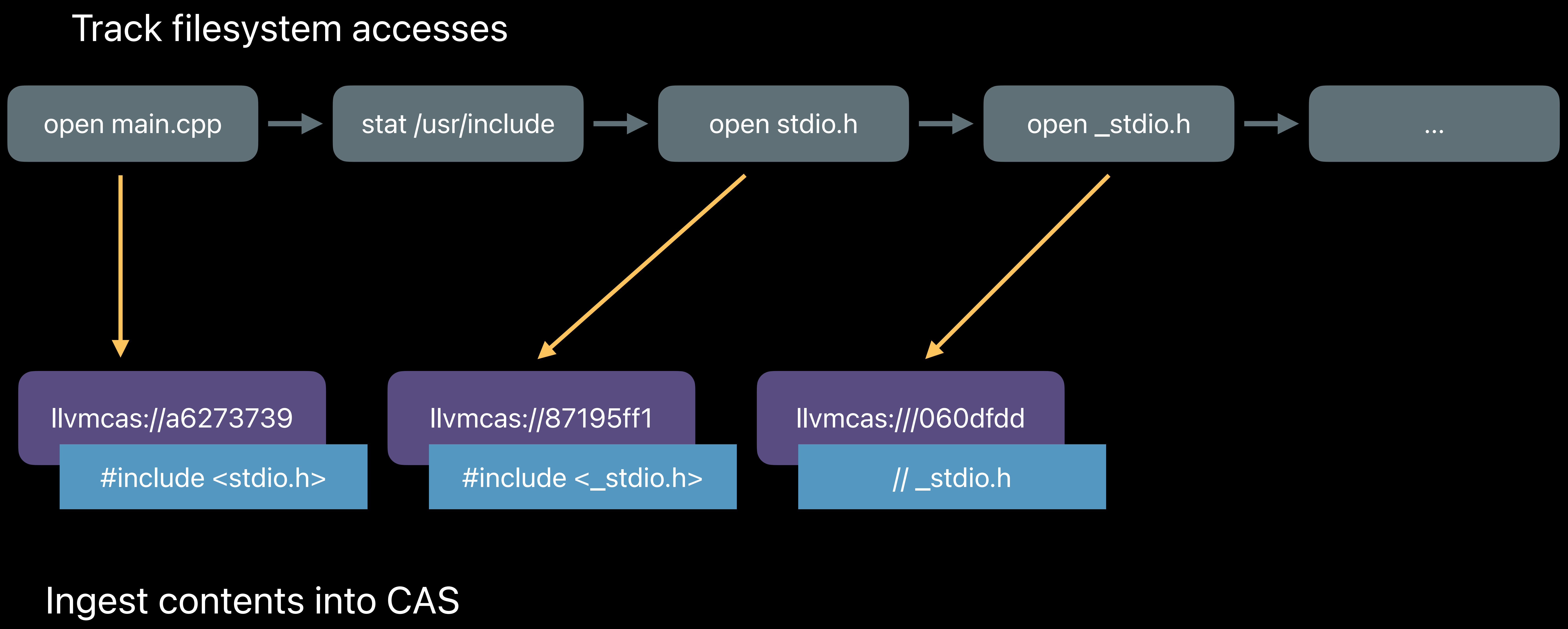
For more information

- clang-scan-deps: Fast Dependency Scanning for Explict Modules EuroLLVM 2019
- Implicitly discovered, explicitly built Clang modules EuroLLVM 2022

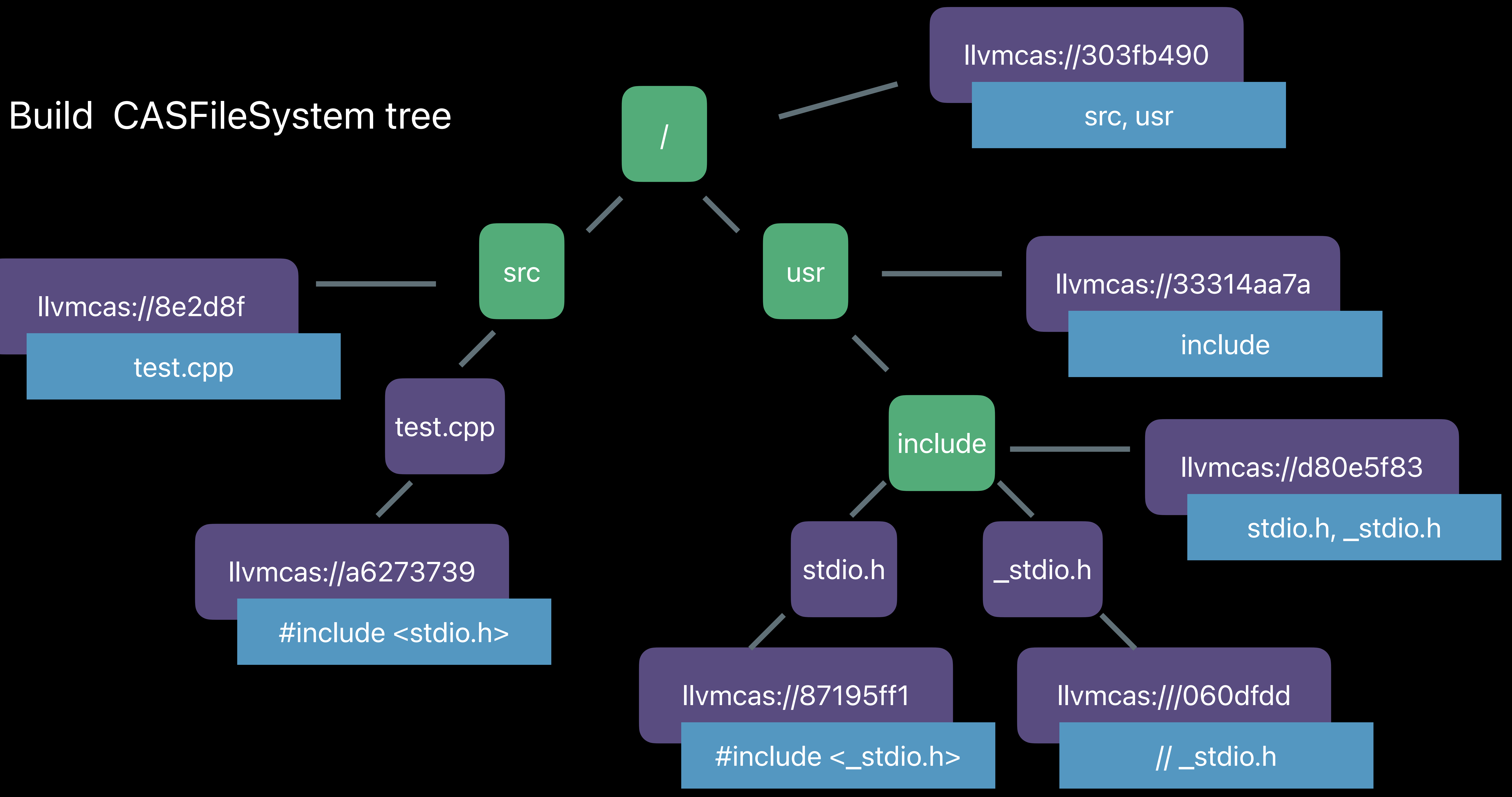
clang-scan-deps discovers inputs: Ingest into CAS

Ingest input files using `CachingOnDiskFileSystem` (VFS)

Simple input discovery with CachingOnDiskFileSystem



Simple input discovery with `CachingOnDiskFileSystem`



Simple input discovery with `CachingOnDiskFileSystem`

`CASFileSystem` identified by root CAS ID

/

llvmcas://303fb490

```
-fcas-fs llvmcas://303fb490
```

Fearlessly cache results

Cache key:

- `-cc1` command
- CAS inputs
- Compiler version

Fearlessly cache results

```
clang-cache clang++ -c test.cpp -Rcompile-job-cache
remark: compile job cache hit for 'llvmcas://983e09ad7717df57dbcc6906c977e11c769936f5a740aeb46f40a57743abb2ad' =>
    'llvmcas://a6b70c60e0165e98295f6f2ee40a70e0233748c2bf09ca2d1de0a869d99cca7b' [-Rcompile-job-cache-hit]
```

```
command-line:
  -cc1 \
  -fcas-path llvm.cas.builtin.v2[BLAKE3] \
  -fcas-fs llvmcas://89bafd50702f574967b246b94ed4da43c7bdefb9af3d69b8087d34029e6113af \
  -fcas-fs-working-directory /Users/blangmuir/src/cas/build \
  -o \
  - \
  -emit-obj \
  -x c++ test.cpp \
...

filesystem: llvmcas://89bafd50702f574967b246b94ed4da43c7bdefb9af3d69b8087d34029e6113af
  tree llvmcas://f0945... /Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/System/Library/Frameworks/
  tree llvmcas://f0945... /Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/c++/v1/
  tree llvmcas://f0945... /Users/blangmuir/src/cas/build/lib/clang/16.0.0/include/
  file llvmcas://31a68... /Users/blangmuir/src/cas/build/test.cpp
  file llvmcas://f1748... /Users/blangmuir/src/cas/build/test.h

version:
  clang version 16.0.0 (git@github.com:apple/llvm-project.git 8f121b691067418a8ca9d926512950ca0b75b47e)
```

CAS configuration

CAS filesystem contains all used files

`CachingOnDiskFileSystem` has room for improvement

Conservatively models all filesystem accesses

Some dependencies too coarse-grained, causing high cache misses

- Consider headermaps: contain a list of every header in target/project
- Add a new header => 100% cache misses since the headermap is modified

Run header search twice - already done in clang-scan-deps

Include Tree

Introducing "Include Tree"

Save which file is included at each `#include` directive

Header search is only run in clang-scan-deps

Only files actually needed by compilation are included

- No more search paths, headermaps, ivfsoverlay in cached `-cc1` command

Include Tree: Example

```
test.cpp
#include "H1.h"
...
```

```
H1.h
#include "H2.h"
#include "H3.h"
```

```
H2.h
#include "H3.h"
```

```
H3.h
// empty
```

```
CLANG_CACHE_ENABLE_INCLUDE_TREE=1 clang-cache clang++ -c test.cpp -Rcompile-job-cache
remark: compile job cache miss for 'llvmcas://6c83ab8b4fd6a2652778dd2dc2fdc88bb2d65cd16d666ea220a60390dd6a5543' [-Rcompile-job-cache-miss]
```

```
command-line:
  -cc1 \
  -fcas-path llvm.cas.builtin.v2[BLAKE3] \
  -fcas-include-tree llvmcas://2cfc99...
...

include-tree: llvmcas://2cfc99...
  test.cpp llvmcas://c53da...
  1:1 <built-in> llvmcas://deaec...
  2:1 ./H1.h llvmcas://ccb75...
    2:1 ./H2.h llvmcas://3e709...
      2:1 ./H3.h llvmcas://7ff63...
    2:16 ./H3.h llvmcas://7ff63...
```

Include Tree: Example

```
include-tree: llvmcas://2cfc99...
  test.cpp llvmcas://c53da...
  1:1 <built-in> llvmcas://deaec...
  2:1 ./H1.h llvmcas://ccb75...
    2:1 ./H2.h llvmcas://3e709...
      2:1 ./H3.h llvmcas://7ff63...
    2:16 ./H3.h llvmcas://7ff63...
```

Building clang with caching

```
cmake -G Ninja \
  -DCMAKE_C_COMPILER_LAUNCHER:PATH=$(which clang-cache) \
  -DCMAKE_CXX_COMPILER_LAUNCHER:PATH=$(which clang-cache) \
  -DLLVM_ENABLE_PROJECTS="clang" \
  -DLLVM_TARGETS_TO_BUILD="X86;ARM;AArch64" \
  -DCMAKE_BUILD_TYPE:STRING=Release \
  -DLLVM_ENABLE_ASSERTIONS:BOOL=ON \
  -DCMAKE_CXX_FLAGS="-Rcompile-job-cache" \
  ../llvm-project/llvm
```

Enable Caching

Building clang with caching

```
time ninja clang
```

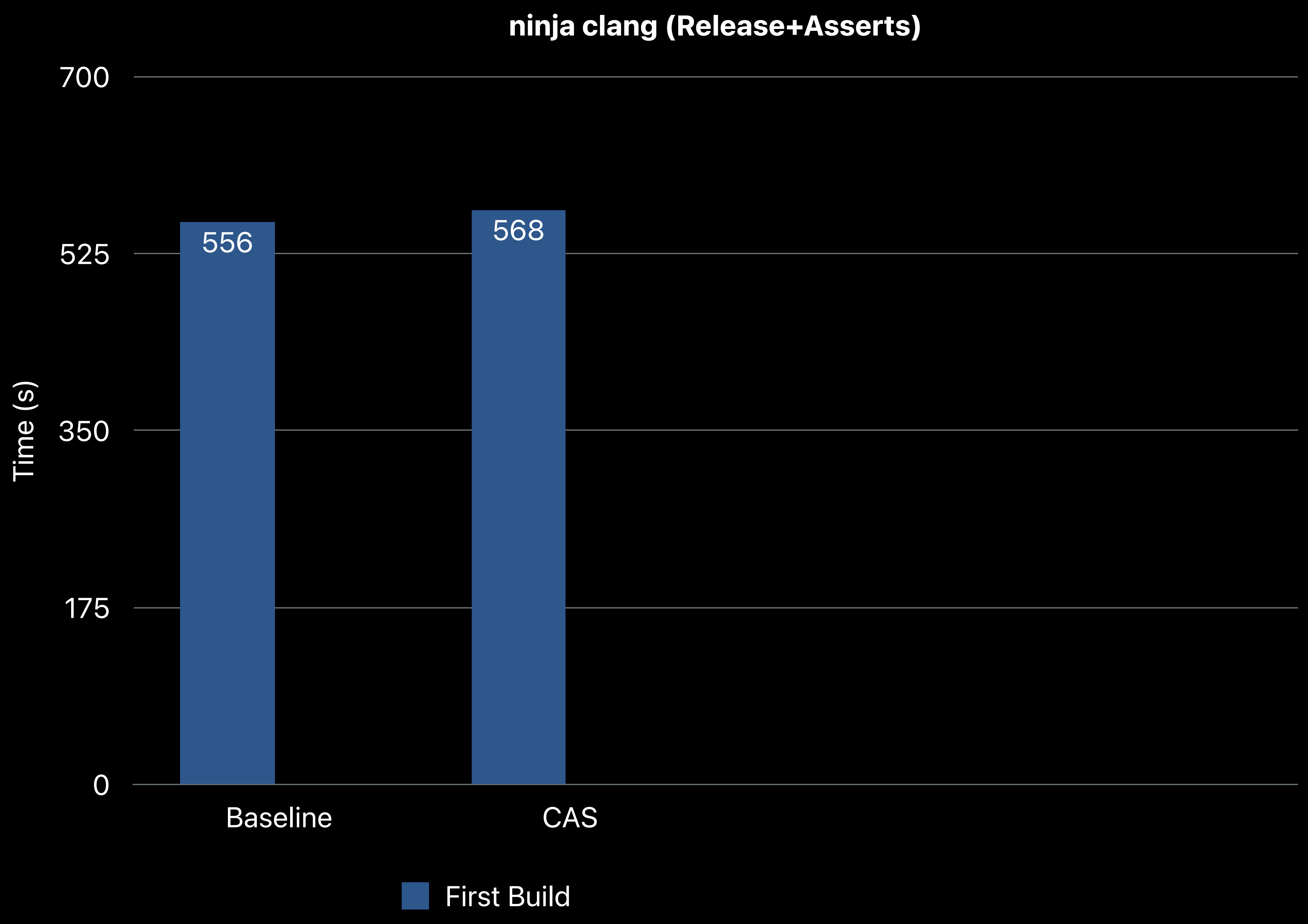
```
5004.98s user 238.51s system 922% cpu 9:28.23 total
```

```
ninja clean
```

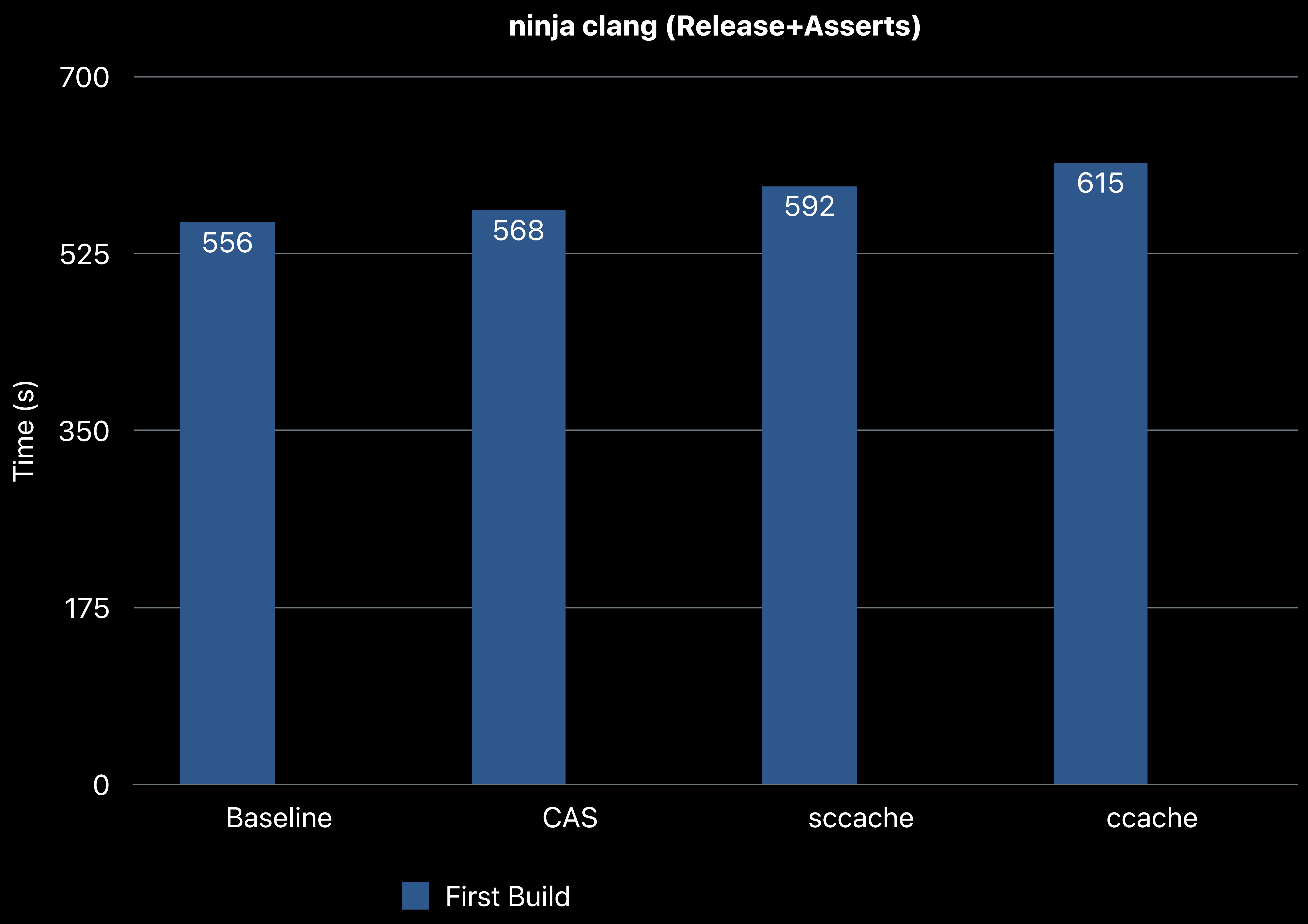
```
time ninja clang
```

```
66.26s user 26.42s system 526% cpu 17.615 total
```

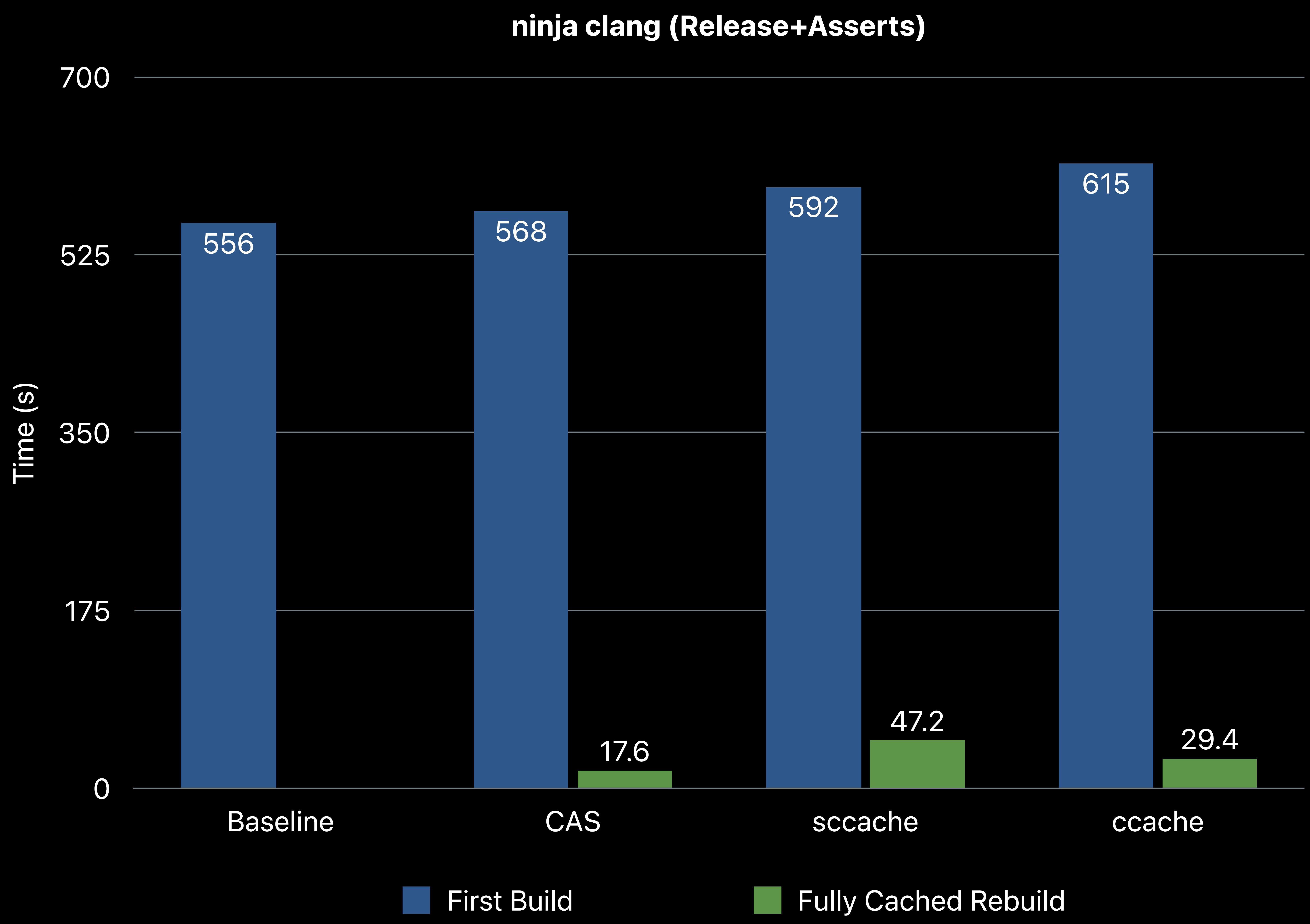
Building clang with caching



Building clang with caching



Building clang with caching



Clang Caching: Summary

Cache compilation of each translation unit

Compiler integration enables:

- Efficient and sound caching
  - Fast dependency scanning with clang-scan-deps
  - Isolation from the mutable filesystem using CAS
  - Domain-specific cache data structures (e.g. Include Tree)
- Designed to support modules (work in progress)

CAS ObjectFile Storage

Motivation

Build artifacts are large!

- Expensive to store
- Can be a bottleneck for object cache

Lots of information is duplicated

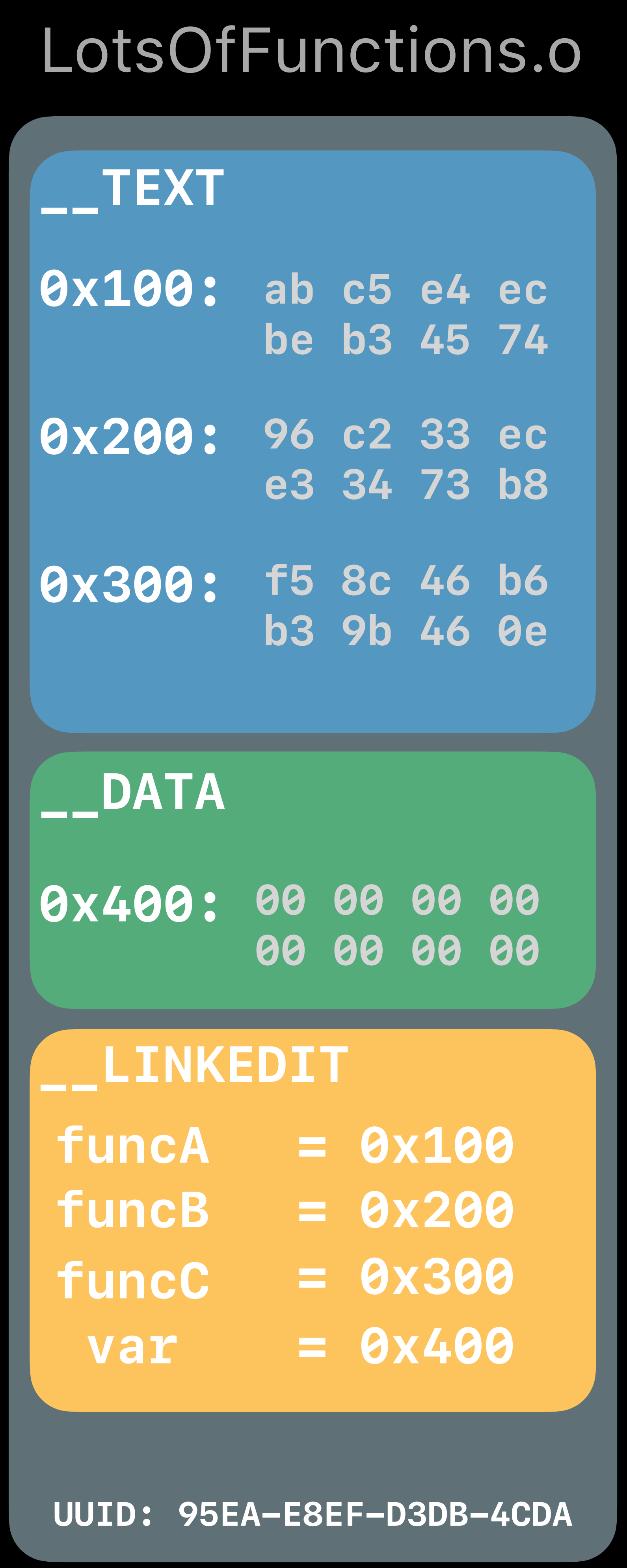
- Across incremental builds
- Within a single build

Observation

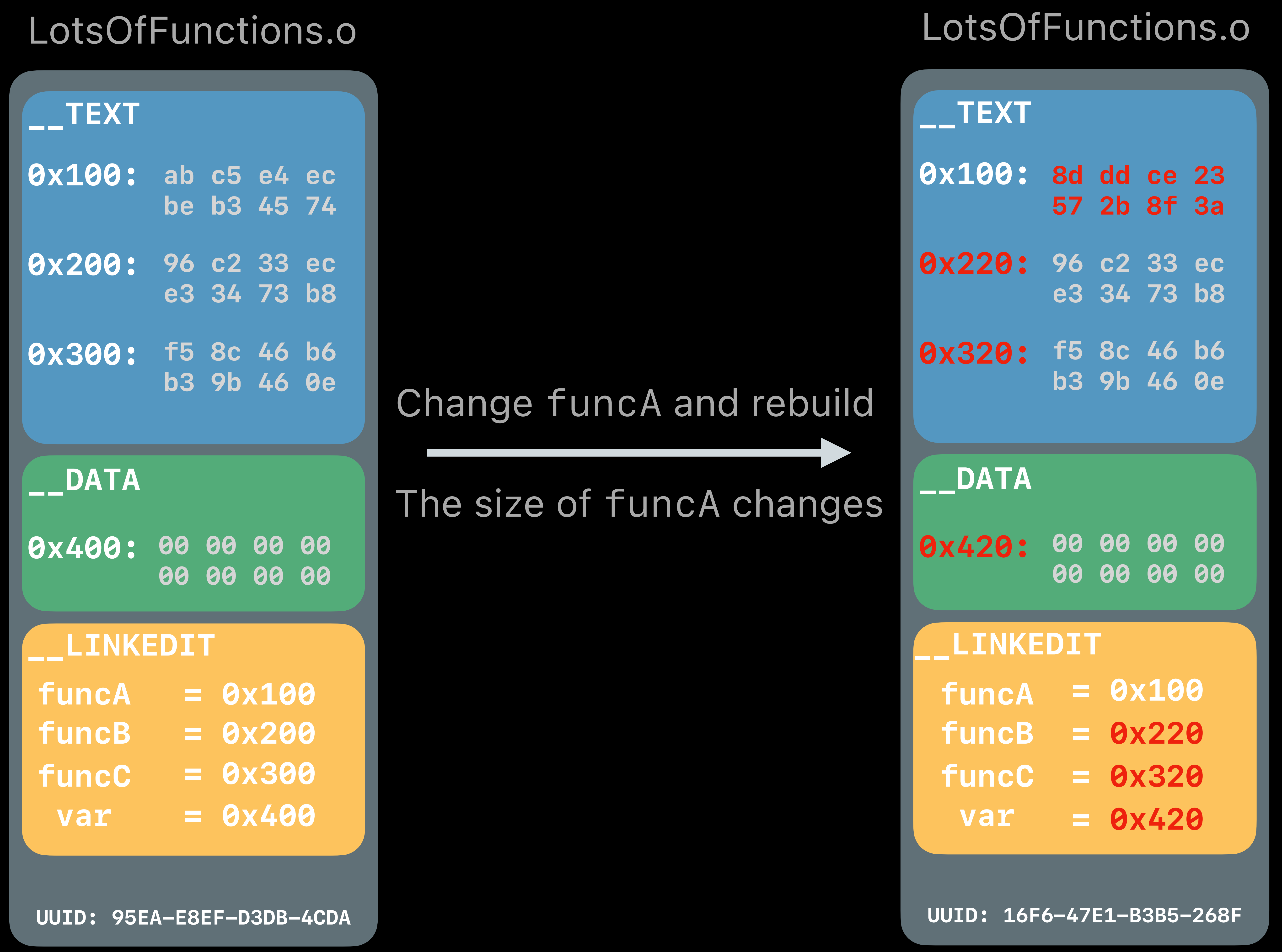
Within a build, duplicated information can be found in:

- Data: C-Strings
- ODR functions: C++ template functions
- Debug Info: Type info

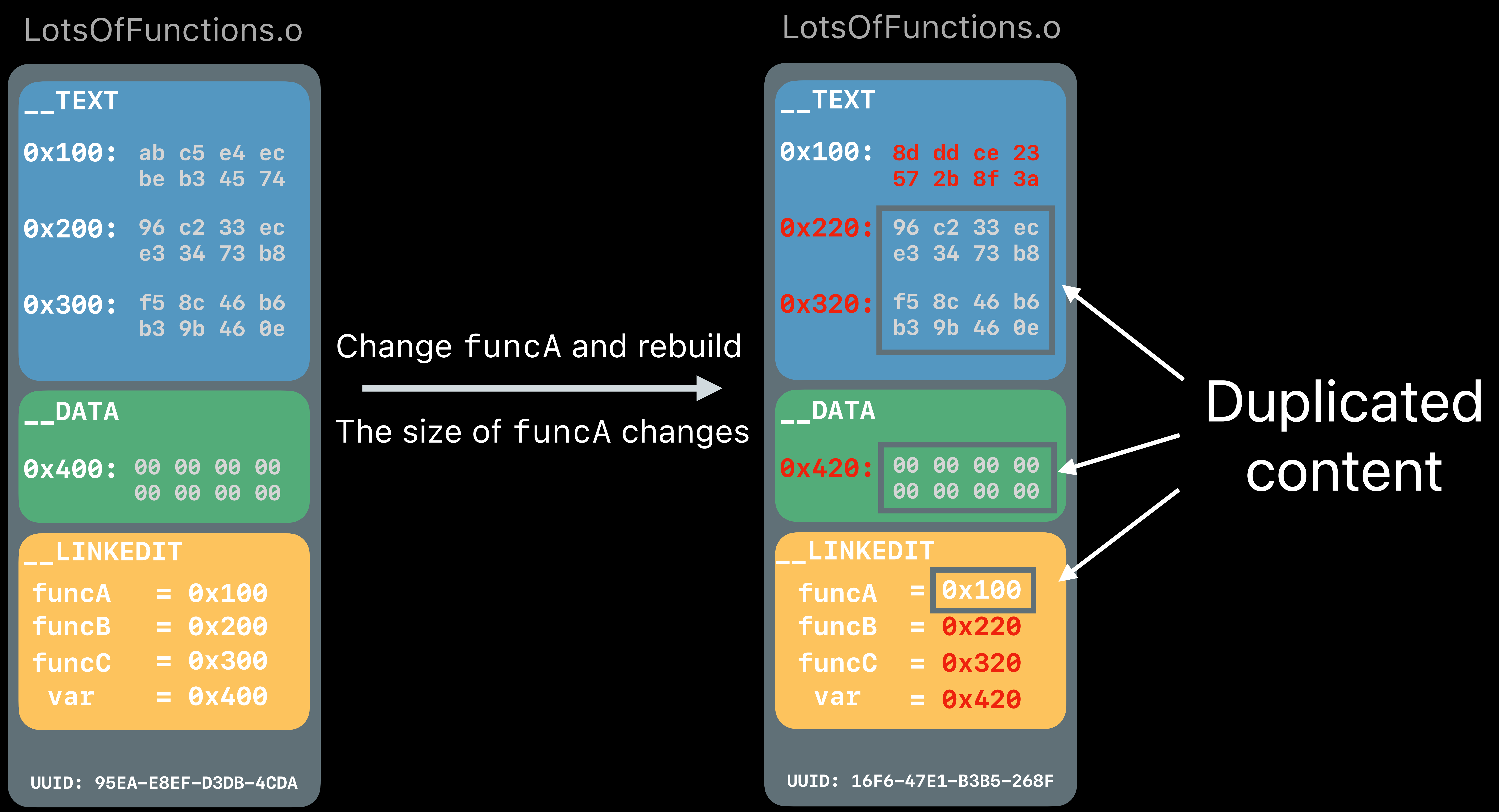
Observation



Observation



Observation



MCCASObjectFormat

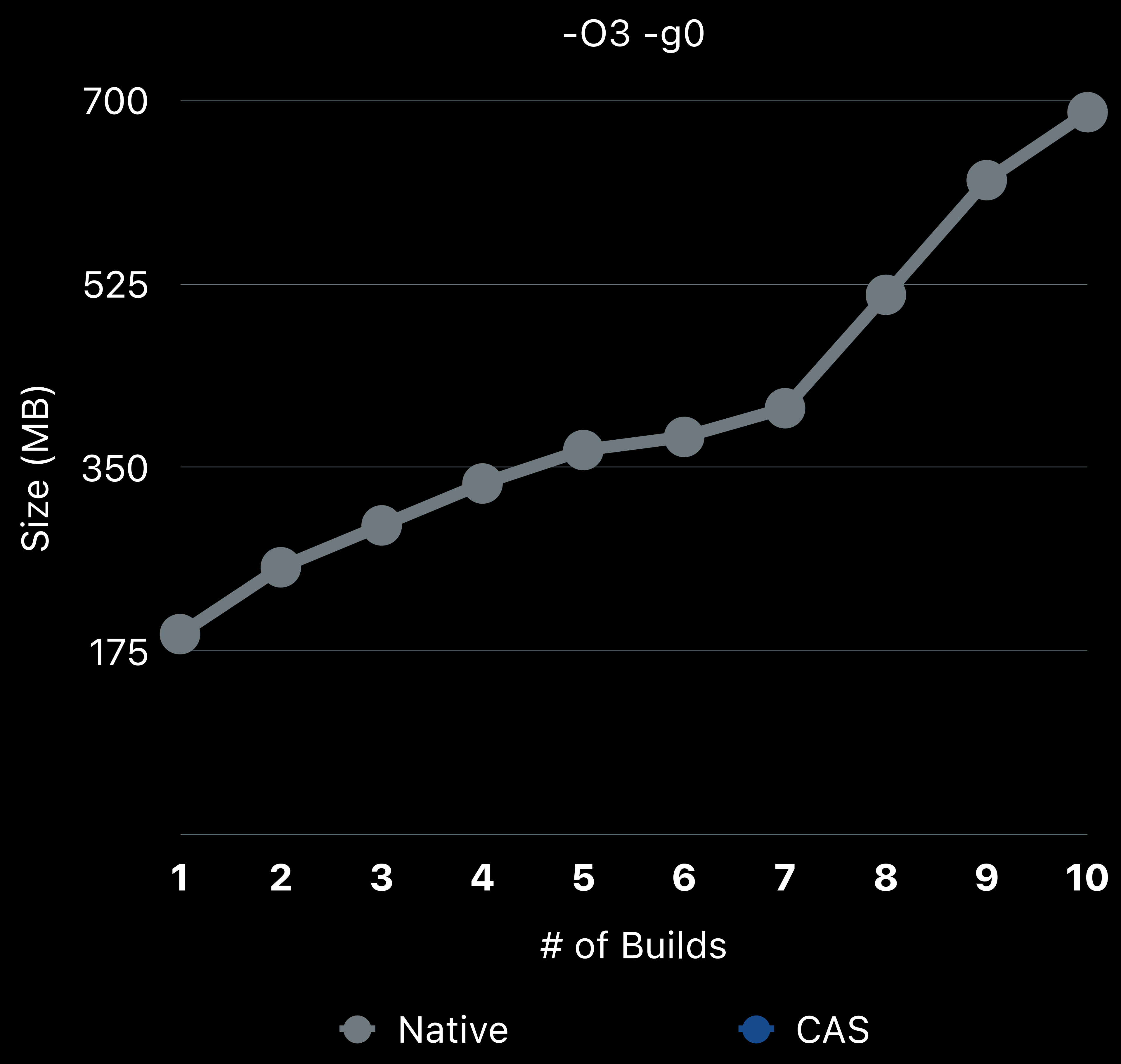
New CASObject representation for MachO object files

- MachOCASObjectWriter: new object writer directly writes to CAS
- Utilize all Machine Code information (e.g. MCFragments)
- Break up object file into blocks that can be reused
- Can be serialized back into MachO Object that is identical to the original

Object Storage Benchmark

Object storage cost benchmark

- building llvm + clang (-03 -g0)
- 1 commit per day across 10 days
- only store unchanged native macho object files once

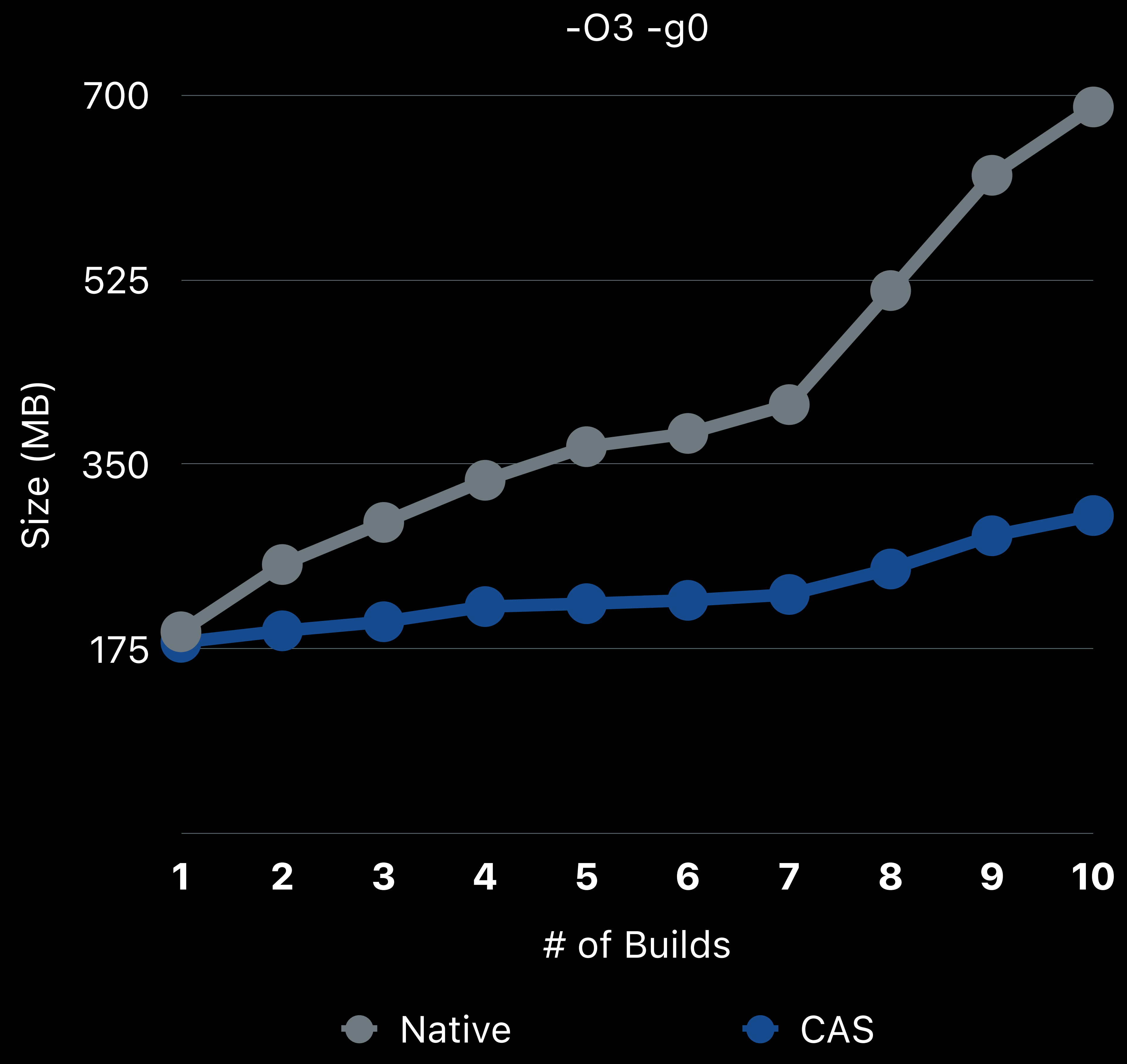


llvm-project from llvmorg-15-init, 1200 commits of changes in total for 10 days

Object Storage Benchmark

CASObjectFormat

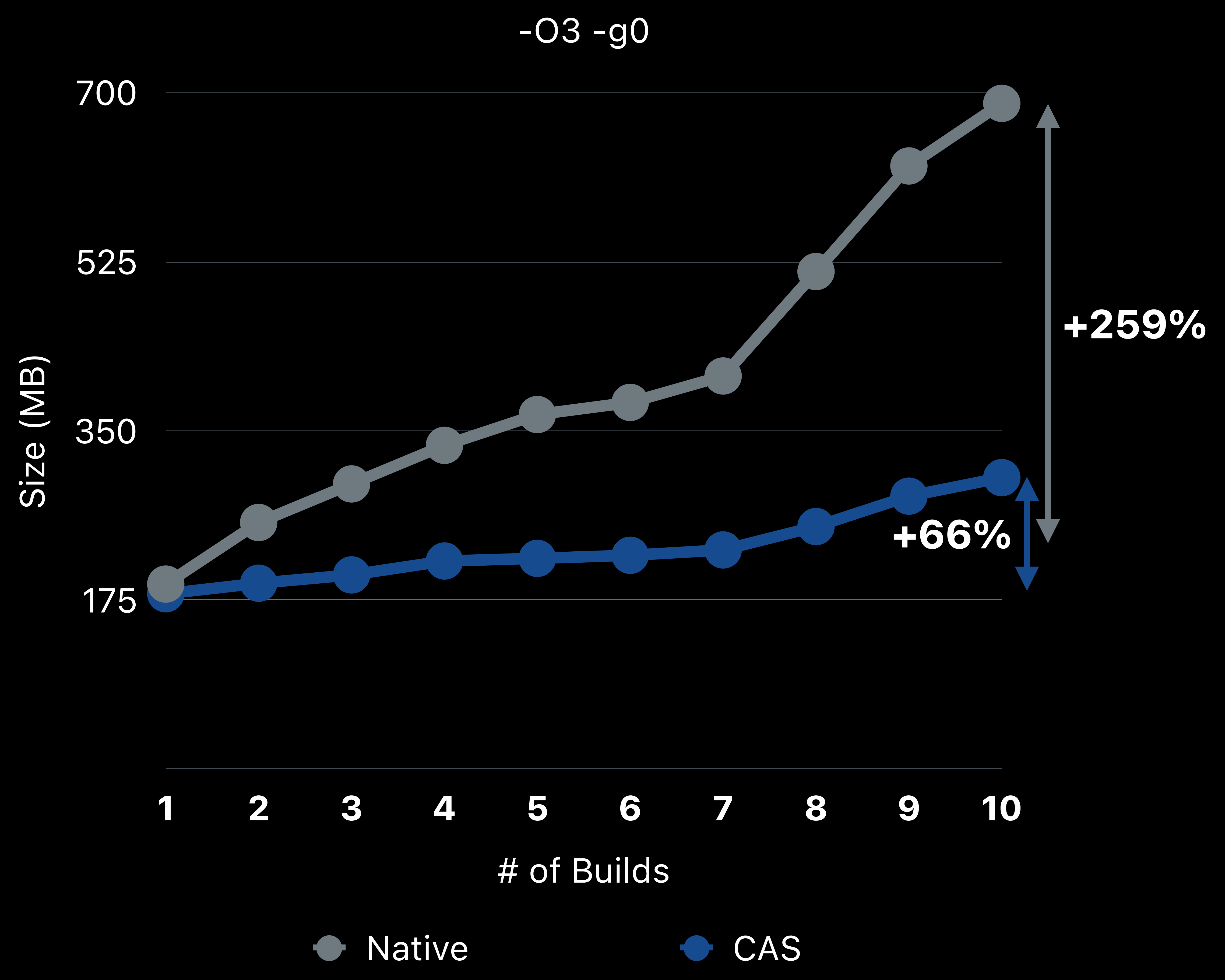
- space saving across the board



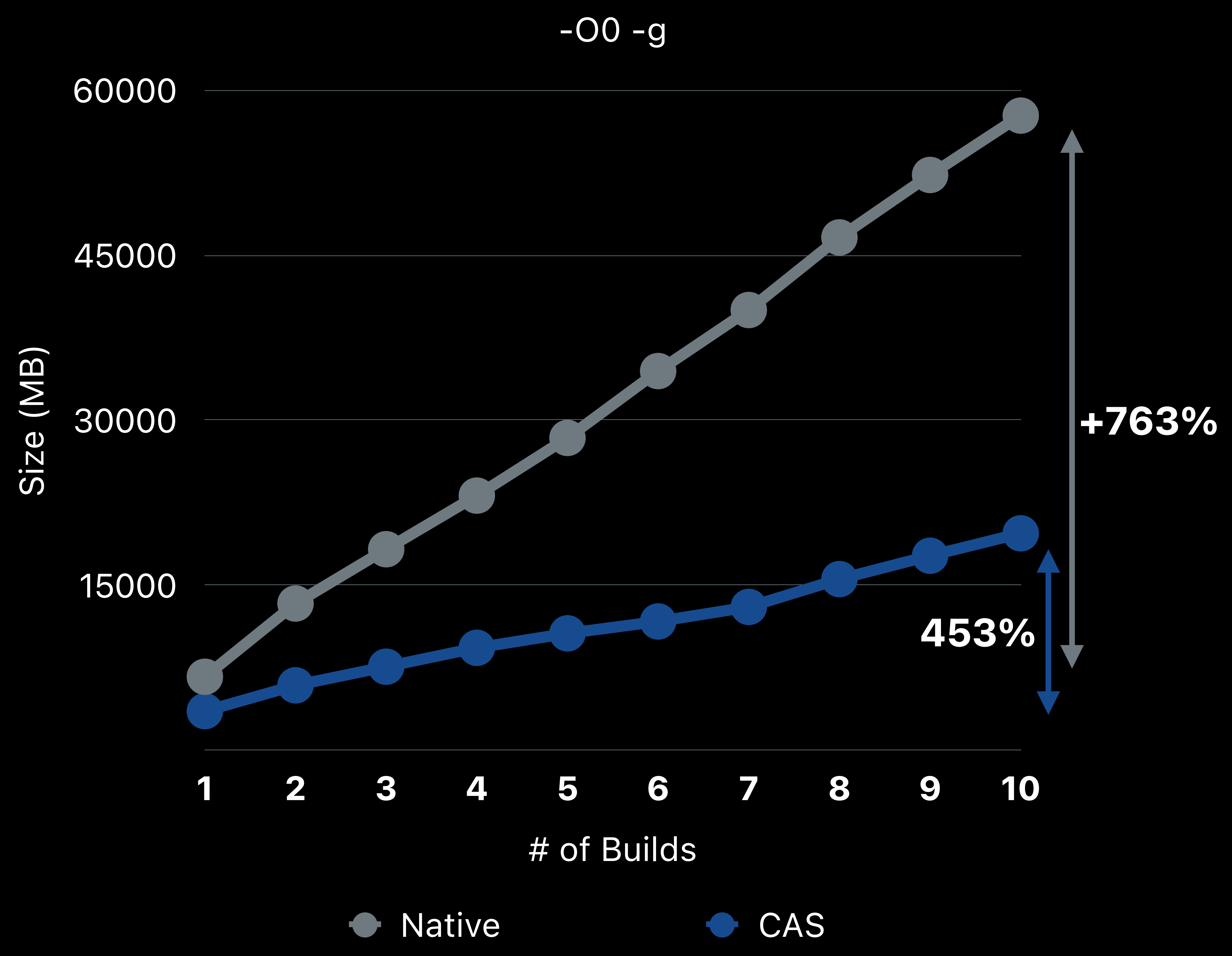
Object Storage Benchmark

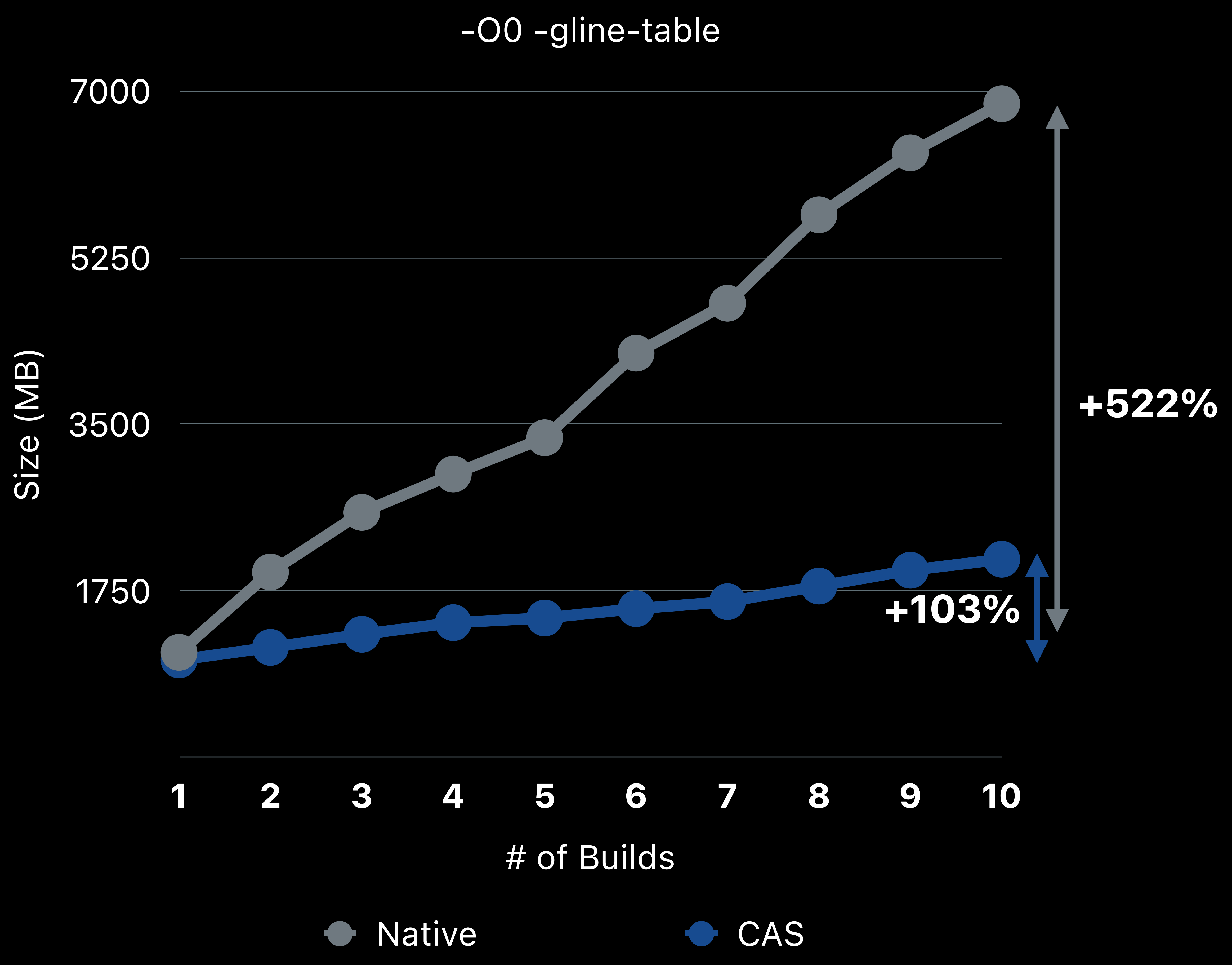
CASObjectFormat

- space saving across the board
- much slower growth in storage cost
- significant long term benefit!



Object Storage with DebugInfo





Future Opportunities

Fine-grained Caching

Finer-grained caching will expose more redundancies

- clang token cache (prototype available)
- Long term: refactor compilers for **fine-grained request-driven computations**

Other Opportunities

CAS-friendly Debug information

- Make line tables **invariant to code motion**
- Design **modular type information**

Add caching to more tools

- TableGen
- Linker

CAS-friendly build artifacts for less storage

Conclusion

RFC: *https://discourse.llvm.org/t/rfc-add-an-llvm-cas-library-and-experiment-with-fine-grained-caching-for-builds/59864*

Round Table for Content Addressable Storage (**Today Nov. 9 4:30pm**)

All examples and prototypes can be found at: *https://github.com/apple/llvm-project/tree/experimental/cas/main*

Pull Requests:

- VirtualOutputBackend: *https://reviews.llvm.org/D133504*
- CAS: *https://reviews.llvm.org/D133716*

Q&A