manyclangs: Fast bisection with a small storage cost

Veselin Karaganev

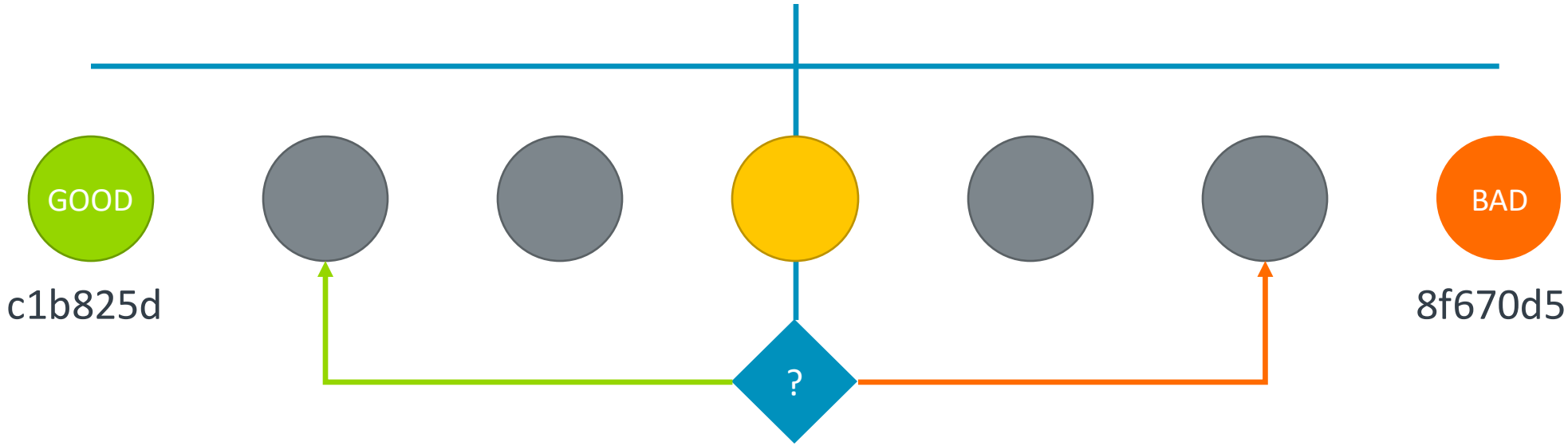
Peter Waller

veselin.karaganev@arm.com, peter.waller@arm.com

2021 LLVM Developers' Meeting

What is Bisection?

- Bisection is applying binary search for searching through commit history
- Given a good and bad commit, by testing a commit between the two, we can narrow down the search space by half




```
$ cmake ... && ninja
$ [ test ]
$ git bisect good/bad
```

LLVM Build Times

- Clean build: 3m58s on a 64-core Neoverse CPU
- Ccache rebuild 100 commits later: 2m45s

- With the normal build process, bisection can take a while!

- Single build ~ 310 MiB (executable binaries only)
- At a rate of 1800 changes pushed in a month (300 MiB * 1800) = 540 GiB to have builds ready to run

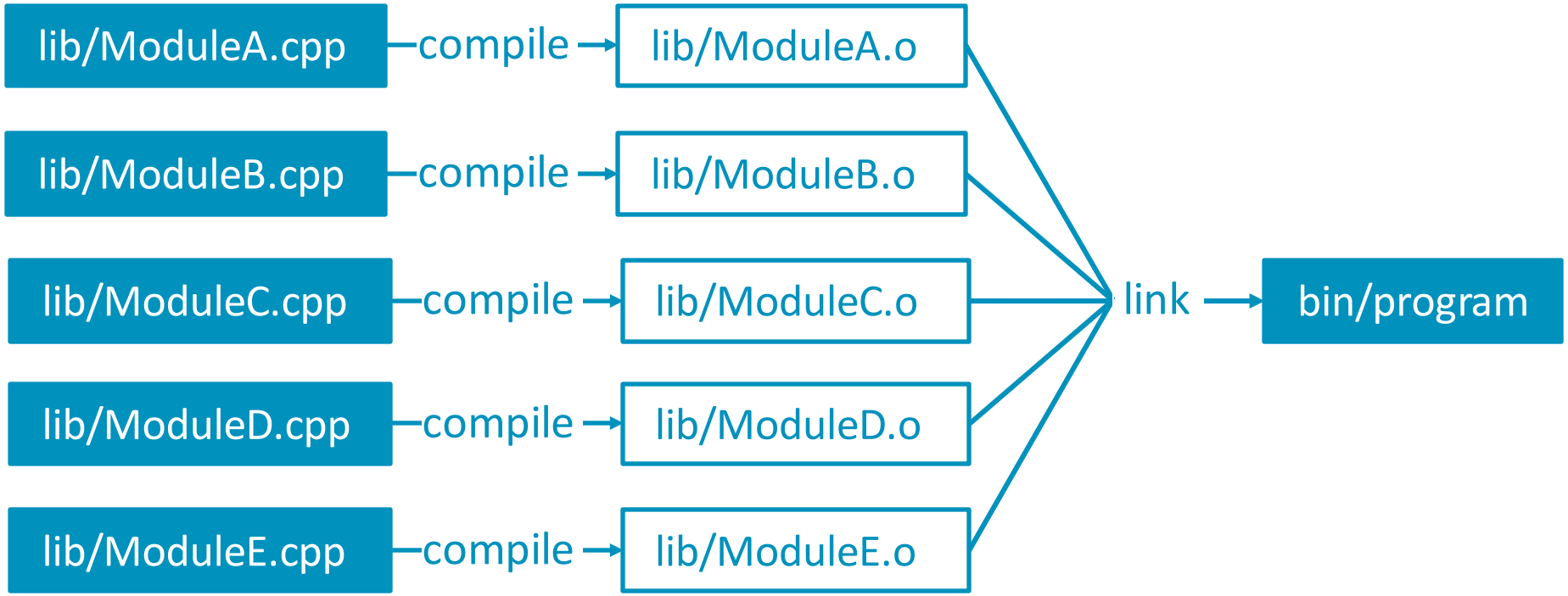
Considered Alternatives

- Ccache?
  - Speeds-up invocations of the compiler using a cache
  - Performs well with small changes to source files, but header changes can mean long build times
  - Our solution stores entire builds and can be used outside of a build environment
- Store on cloud storage?
  - non-local, high costs, requires a decent network link
- Regular compression?
  - Can compress 300 GiB -> 30 GiB
  - No random-access to data in the archive, need to decompress everything

The Secret to sub-1% Compression

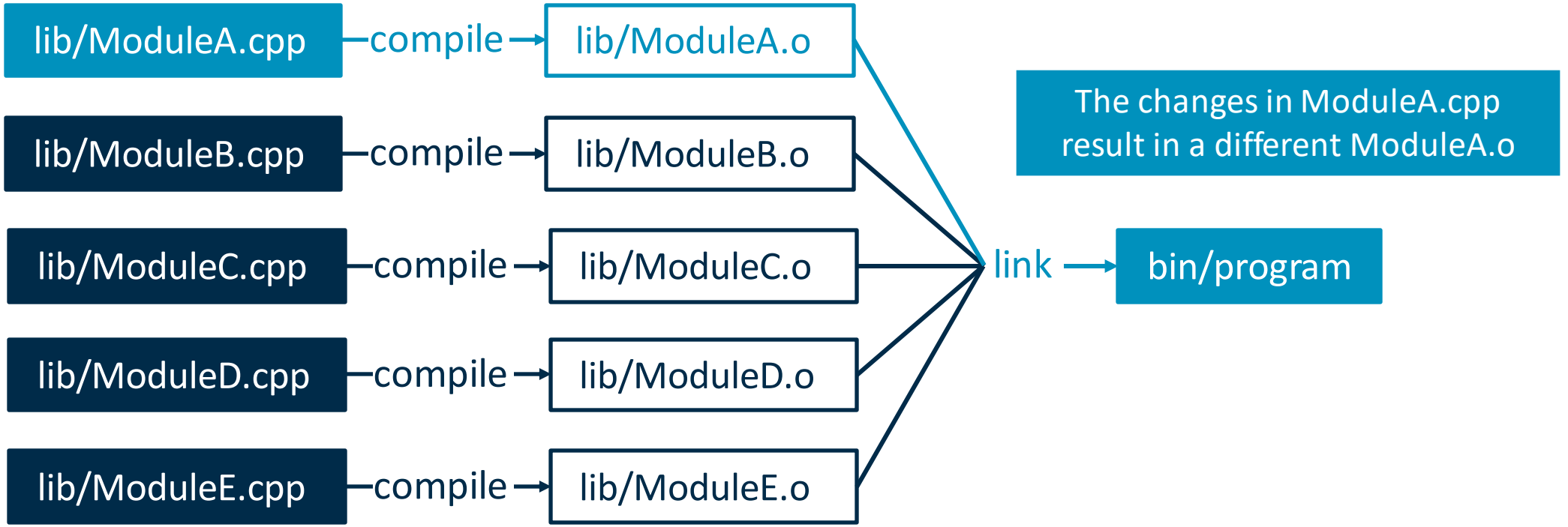
Compilation Process

Compile sources into object code, then link into executable



Make Changes, Recompile

Recompilation after source changes



Make Changes, Recompile

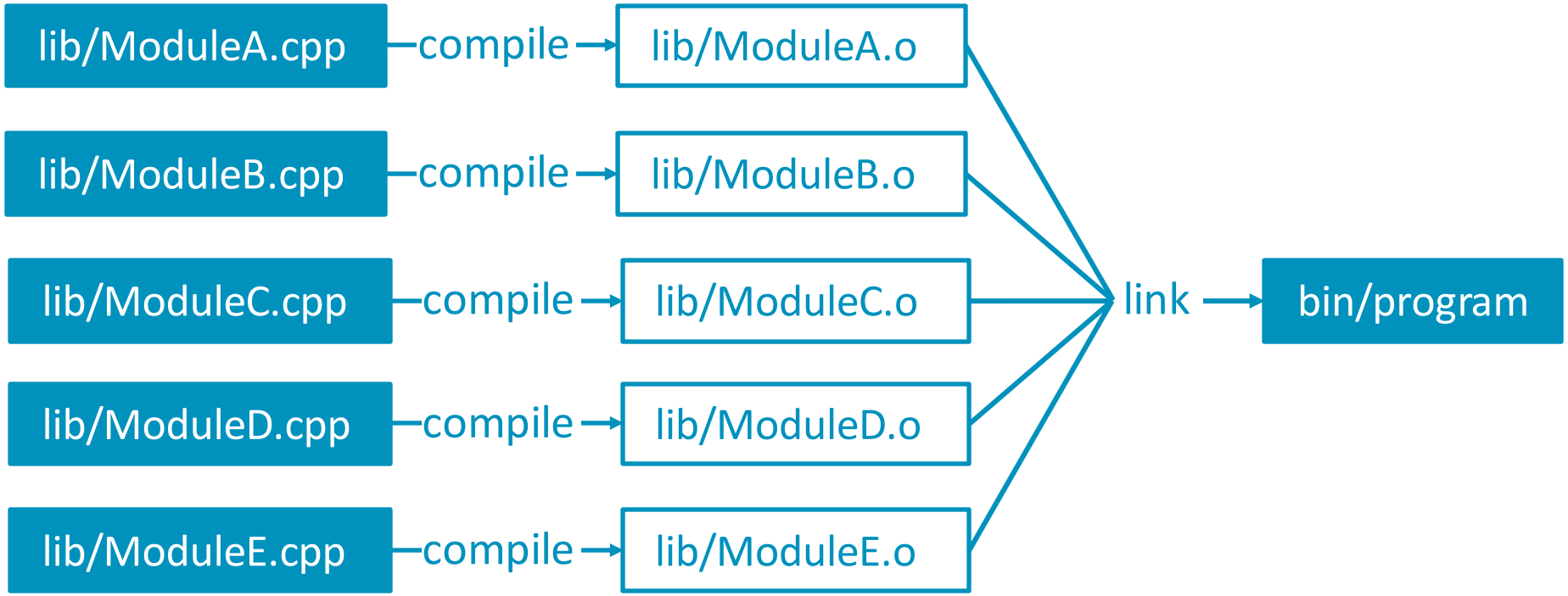
Effects of adding a single call to printf()

- Due to the widespread use of offsets in the ELF format, a small change in the sources can result in large deltas in the binaries
- These different binaries compress together poorly
- Only one .o files was recompiled
- The average .o file size is 120 KiB

```
+00002230: 3814 a003 0000 0000 dc02 0000 0000 0000  8...............
+00002240: 1f8d 0200 1200 0e00 b402 1302 0000 0000  ................
 00002250: 9000 0000 0000 0000 a747 2600 1200 0e00  .........G&.....
-00002260: 1888 ca04 0000 0000 a800 0000 0000 0000  ................
-00002270: 88f6 2700 1200 0e00 8cf4 0405 0000 0000  ..'.............
+00002260: fc87 ca04 0000 0000 a800 0000 0000 0000  ................
+00002270: 88f6 2700 1200 0e00 70f4 0405 0000 0000  ..'.....p.......
 00002280: 6000 0000 0000 0000 e9ae 0500 1200 0e00  `...............
-00002290: 6444 8902 0000 0000 d001 0000 0000 0000  dD..............
-000022a0: cc60 0b00 2200 0e00 04b8 8102 0000 0000  .`.."...........
+00002290: 5444 8902 0000 0000 d001 0000 0000 0000  TD..............
+000022a0: cc60 0b00 2200 0e00 f4b7 8102 0000 0000  .`.."...........
 000022b0: 1000 0000 0000 0000 3711 1600 1200 0e00  ........7.......
-000022c0: 646f ed03 0000 0000 5001 0000 0000 0000  do......P.......
-000022d0: 36b9 2a00 1200 0e00 14e1 0805 0000 0000  6.*.............
+000022c0: 486f ed03 0000 0000 5001 0000 0000 0000  Ho......P.......
+000022d0: 36b9 2a00 1200 0e00 f8e0 0805 0000 0000  6.*.............
 000022e0: 6000 0000 0000 0000 3fdd 3200 1200 0e00  `.......?.2.....
-000022f0: 9cdb 2105 0000 0000 c000 0000 0000 0000  ..!.............
-00002300: e48b 0b00 1200 0e00 fc75 8502 0000 0000  .........u......
+000022f0: 80db 2105 0000 0000 c000 0000 0000 0000  ..!.............
+00002300: e48b 0b00 1200 0e00 ec75 8502 0000 0000  .........u......
```

Idea: Store Object Code, Link On-Demand

And reduce storage costs by deduplicating object files

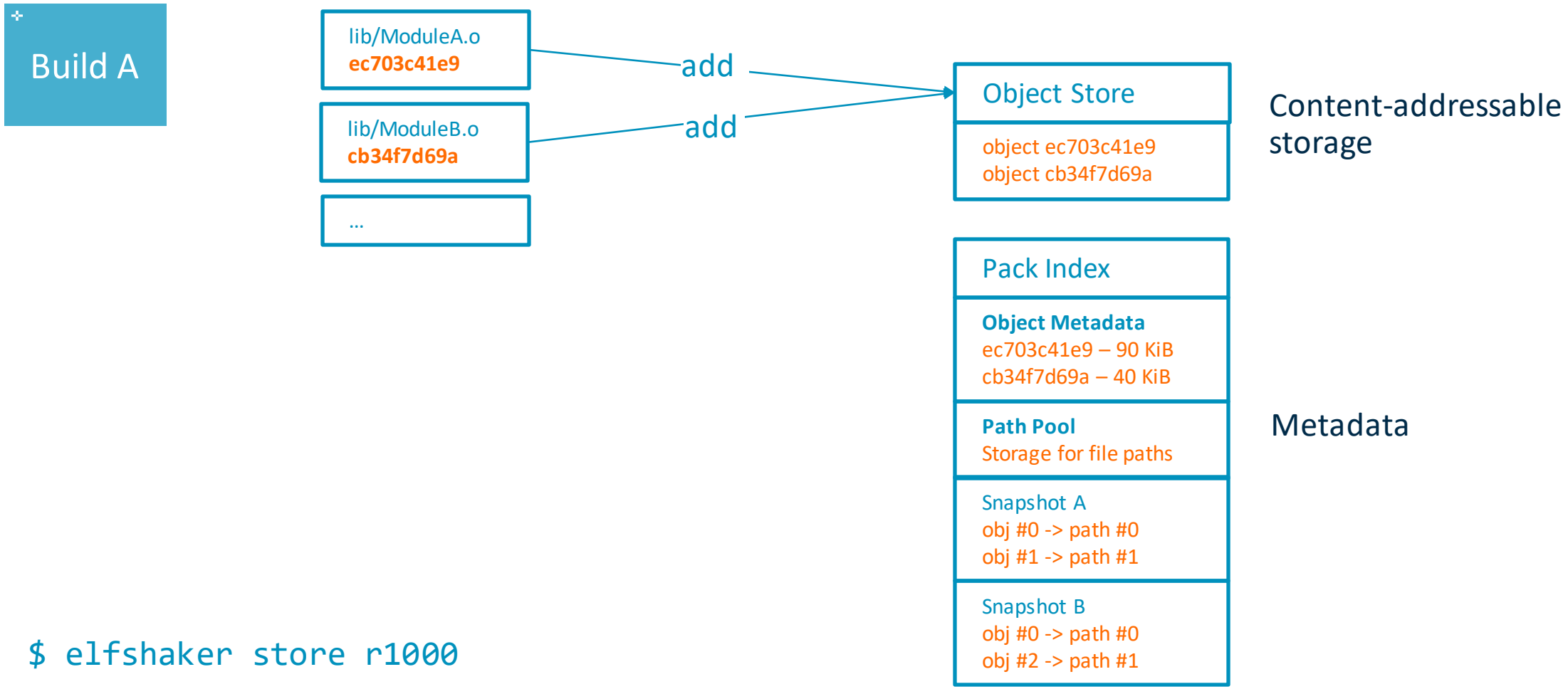


arm

elfshaker

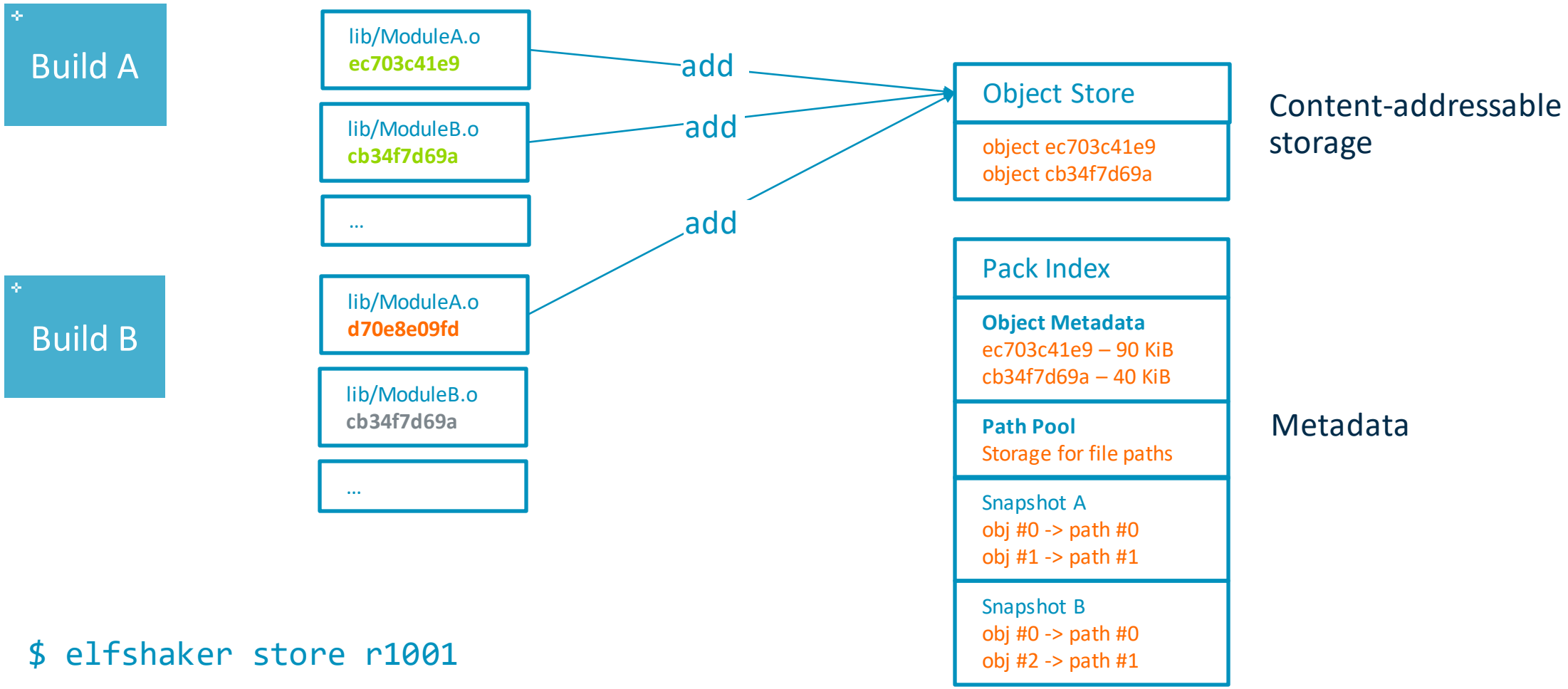
A Storage System Optimised for Storing Object Code

Step 1: Capture Snapshot




```
$ elfshaker store r1000
```

Step 1: Capture Snapshot




```
$ elfshaker store r1001
```

Step 2: Create a Pack

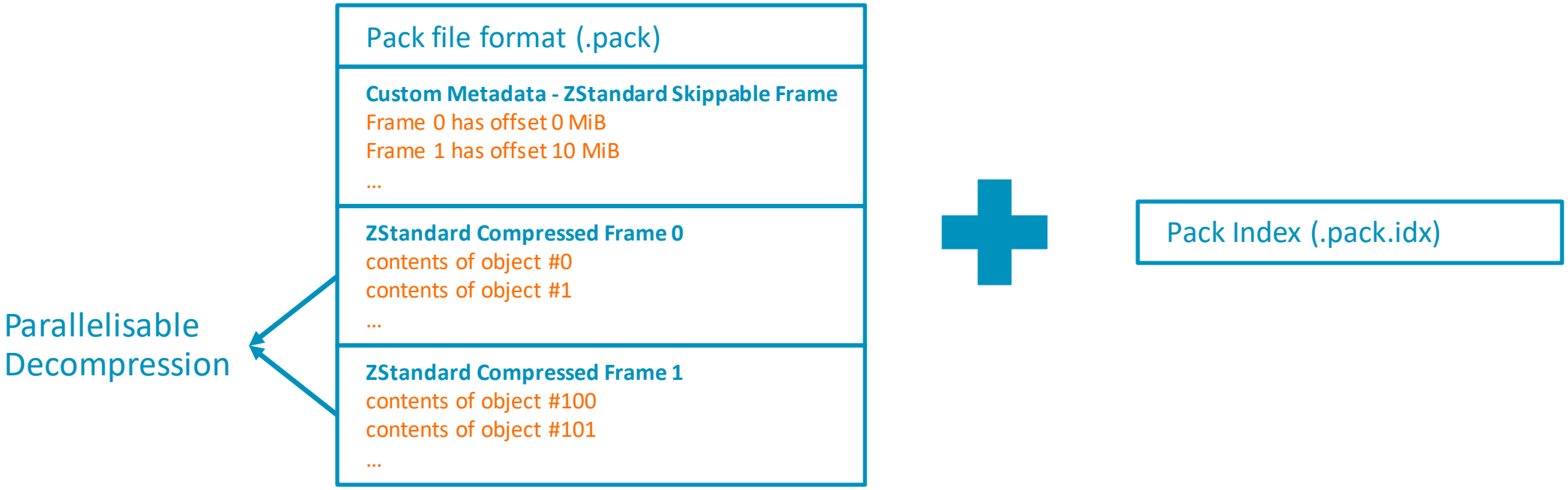
The stored builds are then compressed and packed into a .pack file in a single step

The pack stores the object data, the pack index contains metadata needed to make sense of the decompressed stream.

Step 2: Create a Pack

The stored builds are then compressed and packed into a .pack file in a single step

The pack format uses ZStandard compression and supports parallel decompression



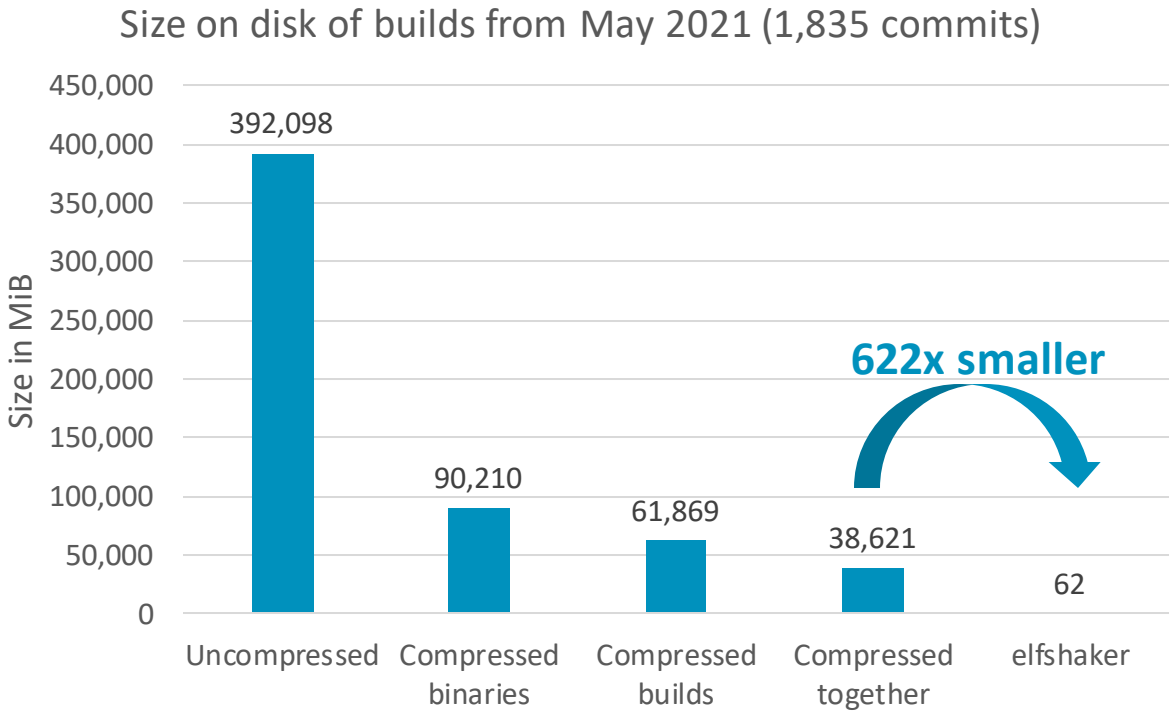
ManyClangs

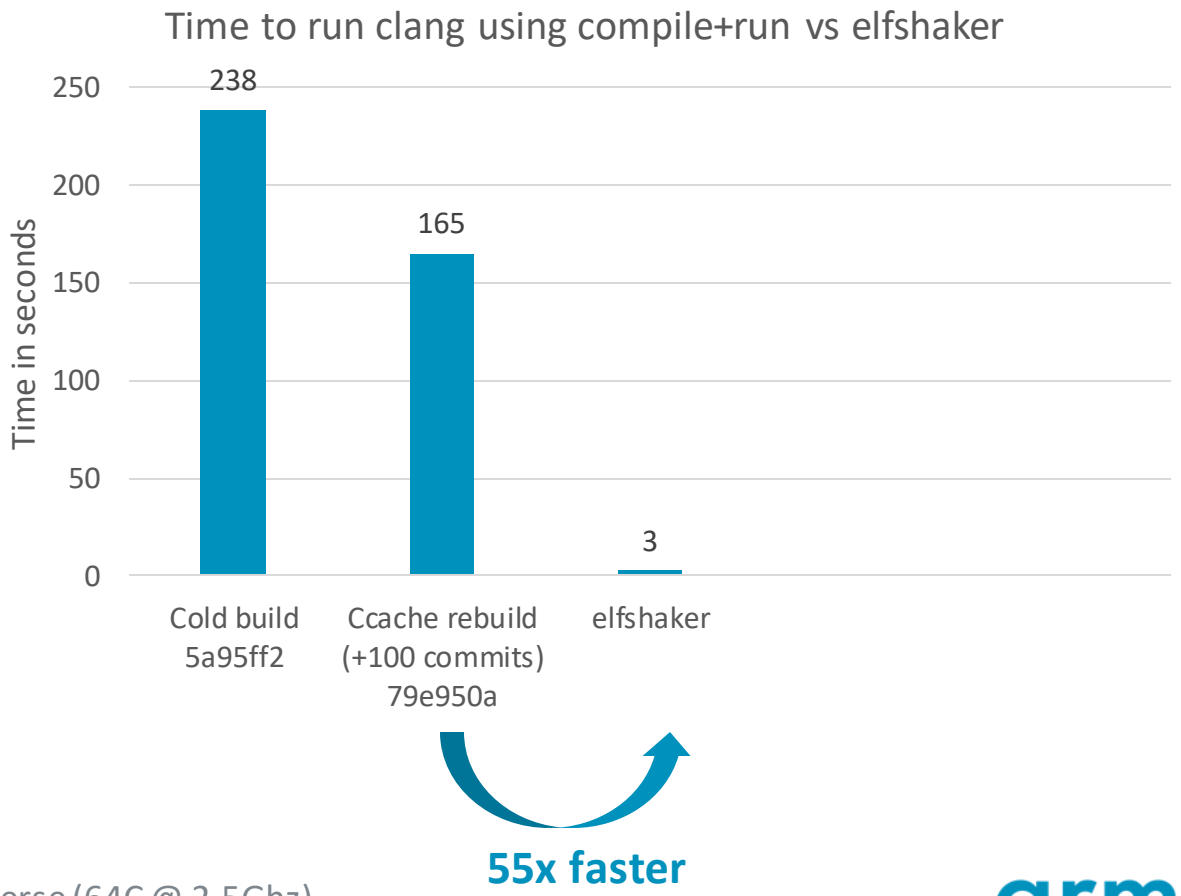
Packs of Monthly Builds of few 100 MiB Each

elfshaker & manyclangs

Results & Comparisons

35 KiB / build
Amortised storage cost





* Measured on Neoverse (64C @ 2.5Ghz)

Bisecting

git bisect start --no-checkout 8f670d5 c1b825d -- llvm clang

manyclangs-run /path/to/manyclangs BISECT_HEAD clang --version | grep '12.0.0'

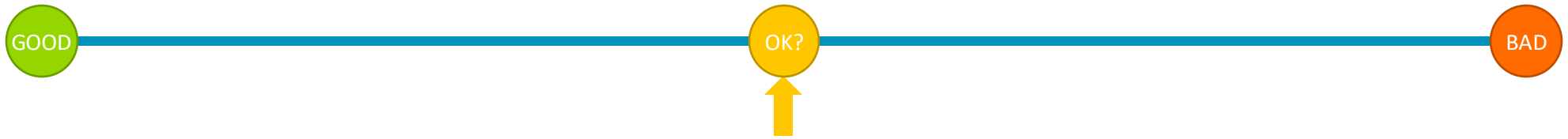
20s
with manyclangs

git bisect run ./automate.sh

23m59s
with Ccache

```
running ./automate.sh
...
elfshaker extracted 8e464dd76b in 1.517 seconds
clang version 12.0.0 (8e464dd76befbc4a39a1d21968a3d5f543e29312)
5369517d20dd362a178a1b2d6c398d8898ee4620 is the first bad commit
Date:   Tue Jan 26 19:37:08 2021 -0800

    Bump the trunk major version to 13
```



* Measured on Neoverse (64C @ 2.5Ghz)

Build Configuration

- Compiler flags to make binaries compress better and builds reproducible
- AArch64 binaries, but intend to provide a way to run these on other architectures
- Release w/ Assertions [-debug/diagnostics]
- LLVM and Clang with all stable targets
- Made some minor source changes (clang –version)
- Builds are reproducible and a clean build of a commit produces a bit-identical executable
  - This allows manyclangs builds to be validated by a bitwise comparison

Outlook

- elfshaker repo: `github.com/elfshaker/elfshaker`
- manyclangs packs: `github.com/elfshaker/manyclangs`

- Our future plans include:
  - Hooking up with Compiler Explorer, allowing people to build with any commit of LLVM
- What you can do:
  - Try using elfshaker/manyclangs in your workflow
  - Send bug reports, open issues on GitHub

arm

Thank You
Благодаря
Danke
Gracias
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה