Fast distributed compilation and testing of large C++ projects

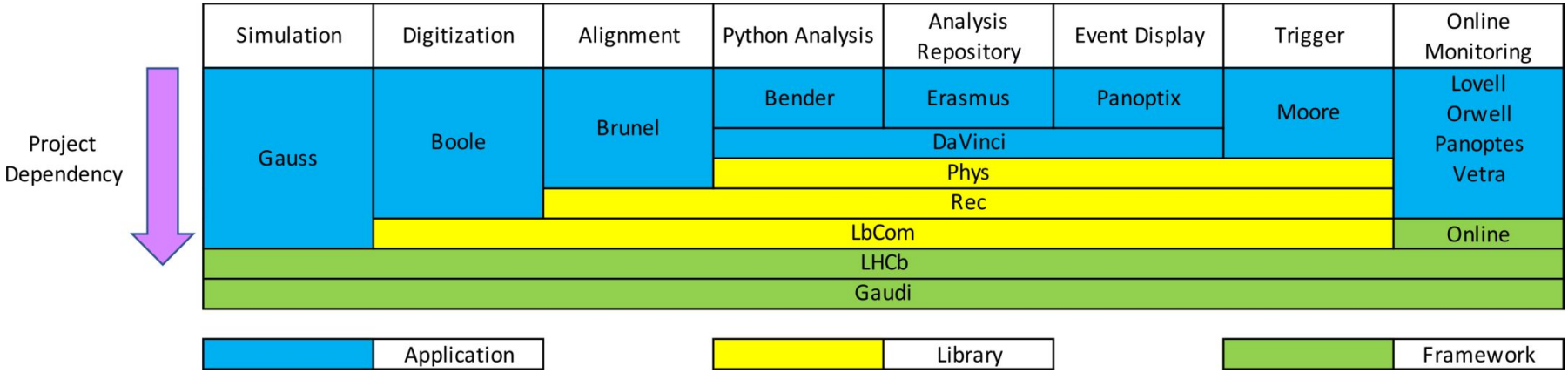
Rosen Matev (CERN)
on behalf of the LHCb RTA project

CHEP 2019, Adelaide
4-8 November 2019

How large are our projects?

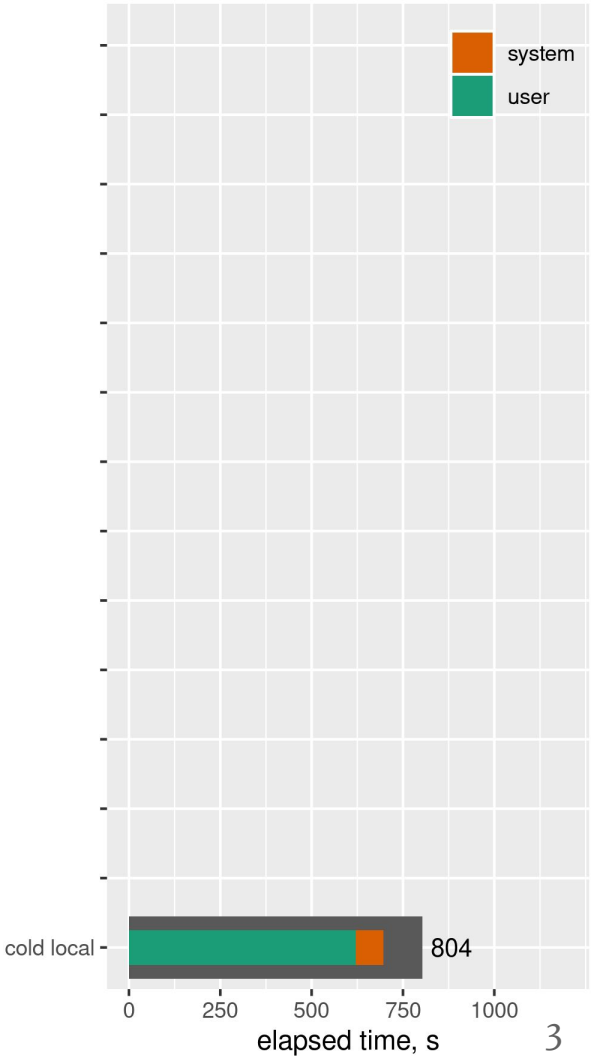
Gaudi: ~600 targets out of which ~450 object files

Moore (LHCb trigger) + deps: ~4800 targets (~3500 object files)



Iterating is slow!

- Local development
  - small machines (here 4 core openstack VM)
  - often incremental builds work
  - until needing to work on the “core” (full checkouts needed)
- Automated testing (e.g. nightly builds)
  - not much bigger machines (8 core VMs)
  - always configure & builds from scratch
  - building & testing *everything* in *many* flavours
- Faster feedback helps across the board!



Iterating is slow!

- Local development
  - small machines (here 4 core openstack VM)
  - often incremental builds work
  - until needing to work on the “core” (full checkouts needed)
- Automated testing (e.g. nightly builds)
  - not much bigger machines (8 core VMs)
  - always configure & builds from scratch
  - building & testing *everything* in *many* flavours
- Faster feedback helps across the board!

Caching build products

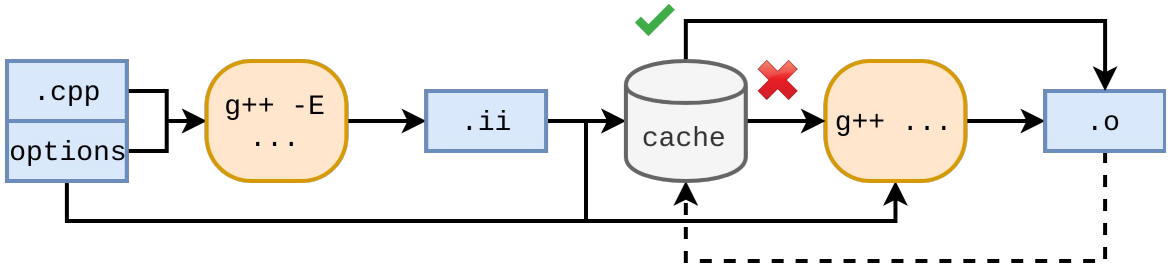
- “clean” or start over sometimes needed
- nice to not redo identical work

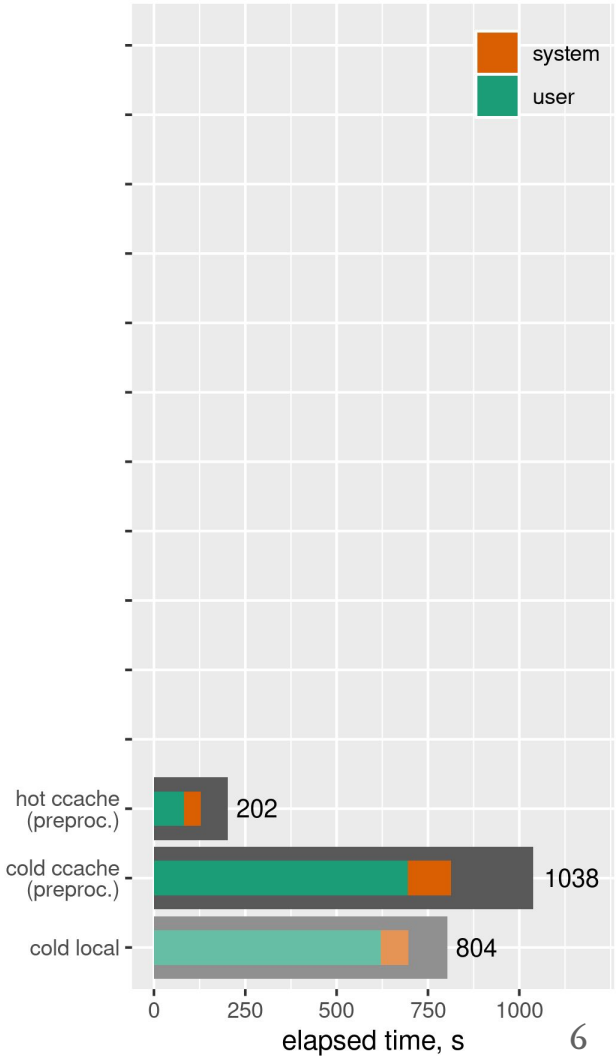
<u>ccache</u>

- widely used and in ~active development
- aims for speed
- simple to use
- nice debugging features
- used in the Gaudi CI, LHCb nightlies

ccache (1)

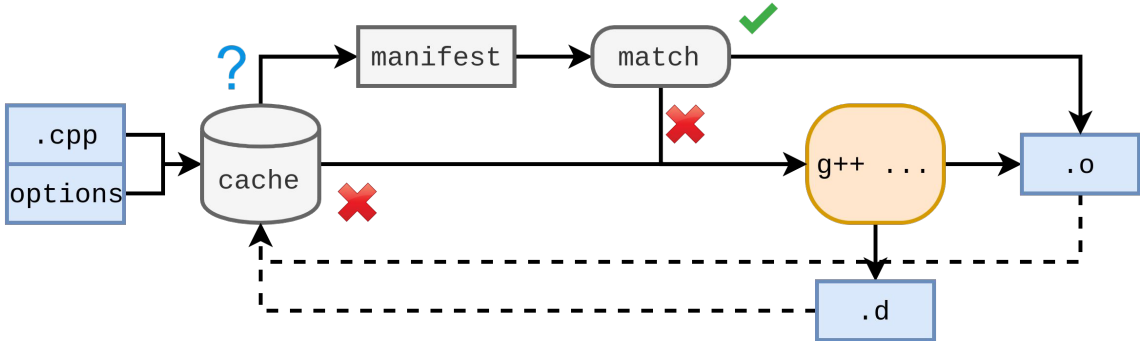
- **preprocessor mode:** use the CPP to determine the transitive header dependencies
- preprocessing is slow!

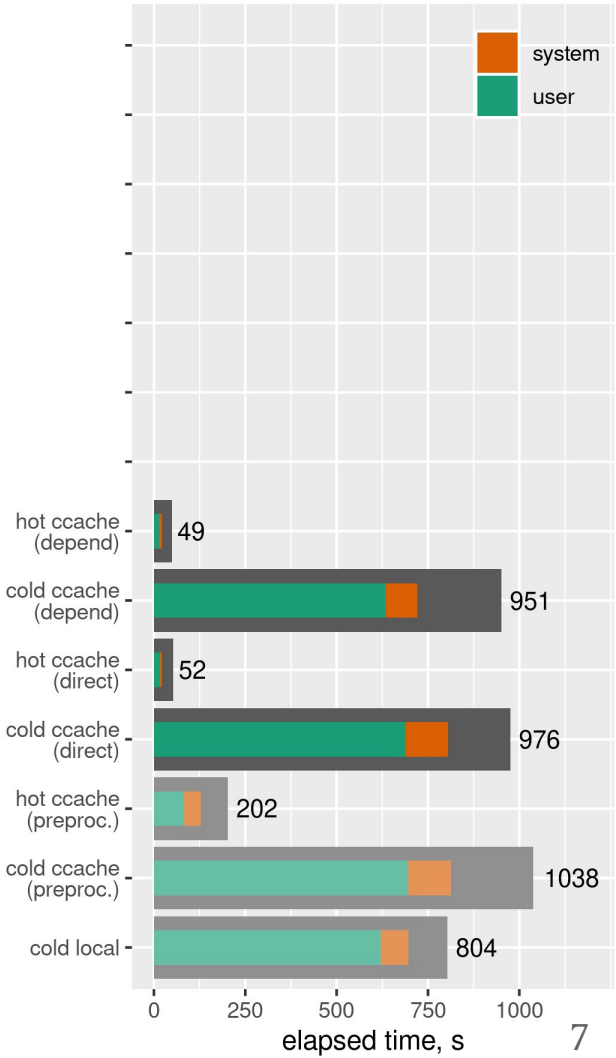




ccache (2)

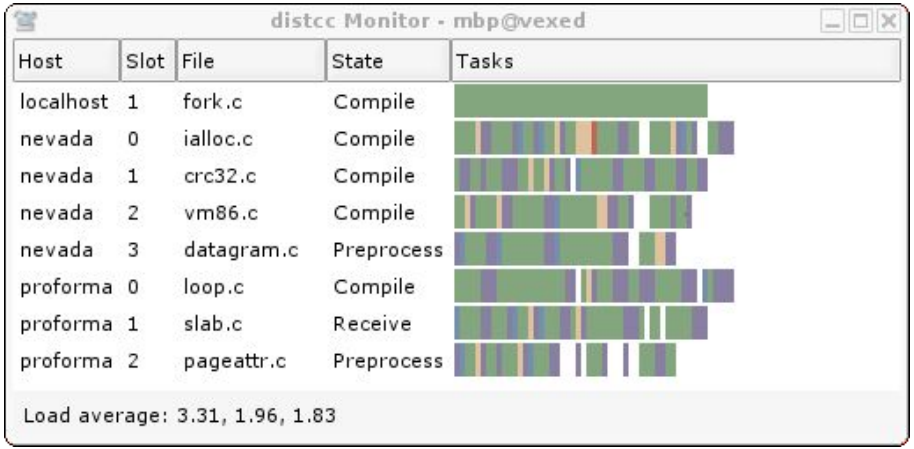
- direct mode: preprocess on cache miss
- **depend mode:** rely on -MD to gather deps
- when hot, ccache hits could be parallelized
  - but always overcommitting is a bad idea





A fast, free distributed C/C++ compiler

- distcc – almost as old as Gaudi
- have your compilation offloaded to some voluntary or dedicated resources, not necessarily having the same OS / CPU
- seems like there is support but no major new developments

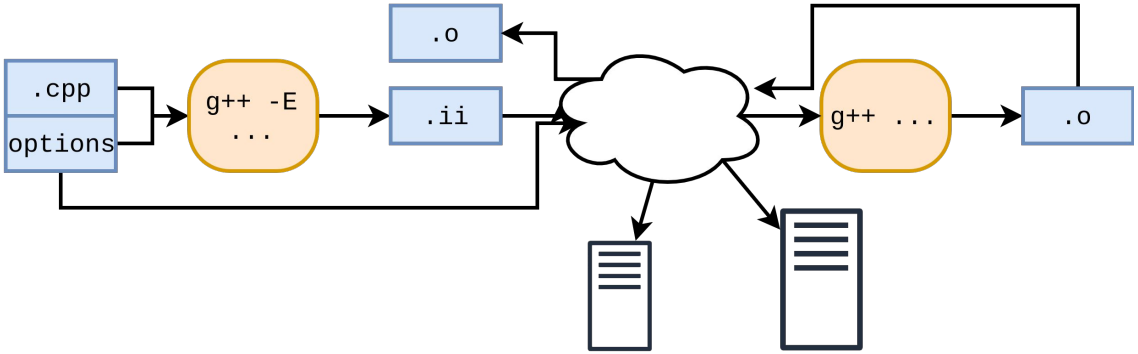


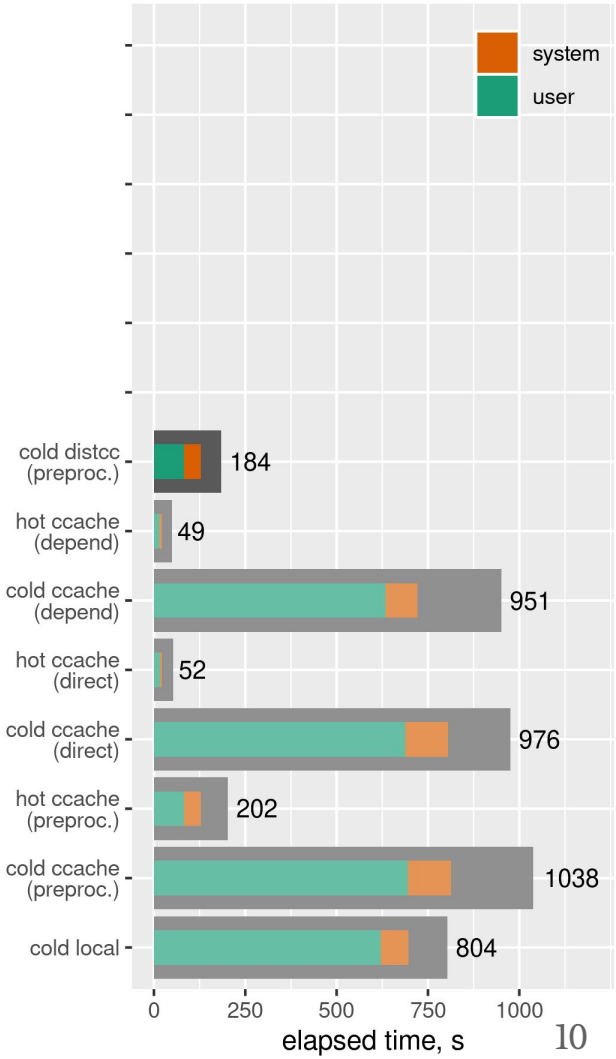
distcc setup

- distccd 3.2rc1 compiled with GSSAPI authentication (CERN)
- servers running CentOS 7 (matches target OS)
  - two dual socket Intel® Xeon® E5-2630 v4 @ 2.20GHz, 80 cores in total
- use authenticated TCP connections
  - lowest overhead but mutual trust between client and server
  - the server of each host registered CERN's central database (KDC)
  - whitelist of users
- slightly modified client
- limit parallelism for non distributed work with ninja pools
  - e.g. linking, ROOT dictionary generation

distcc

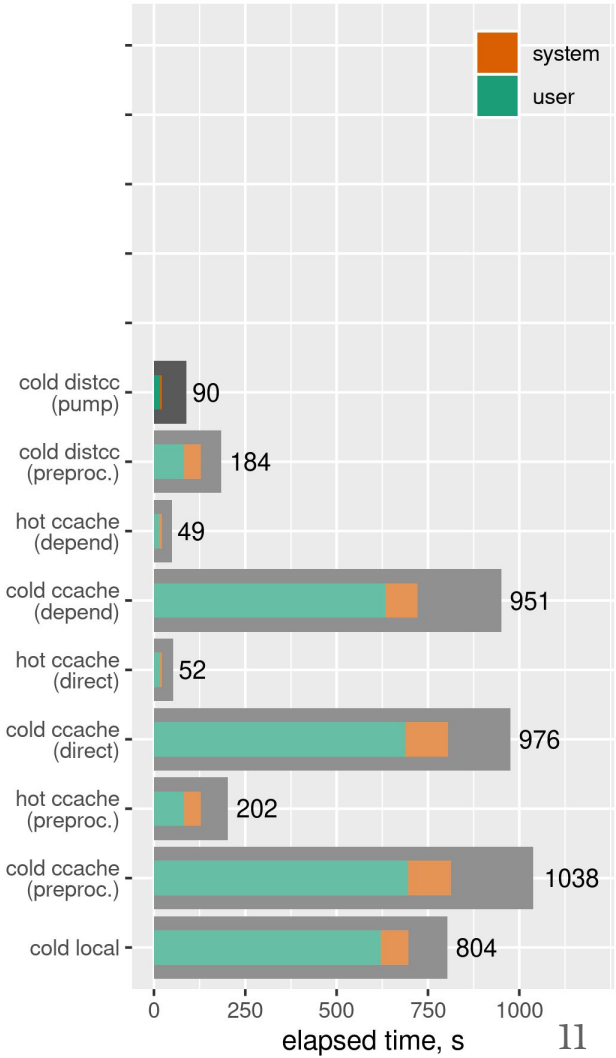
- combine with any ccache mode
  - limit preprocessing with GNU parallel
- on miss, send preprocessed file to server
- preprocessessing is local ⇒ bottleneck
  - can't fully load 80 cores



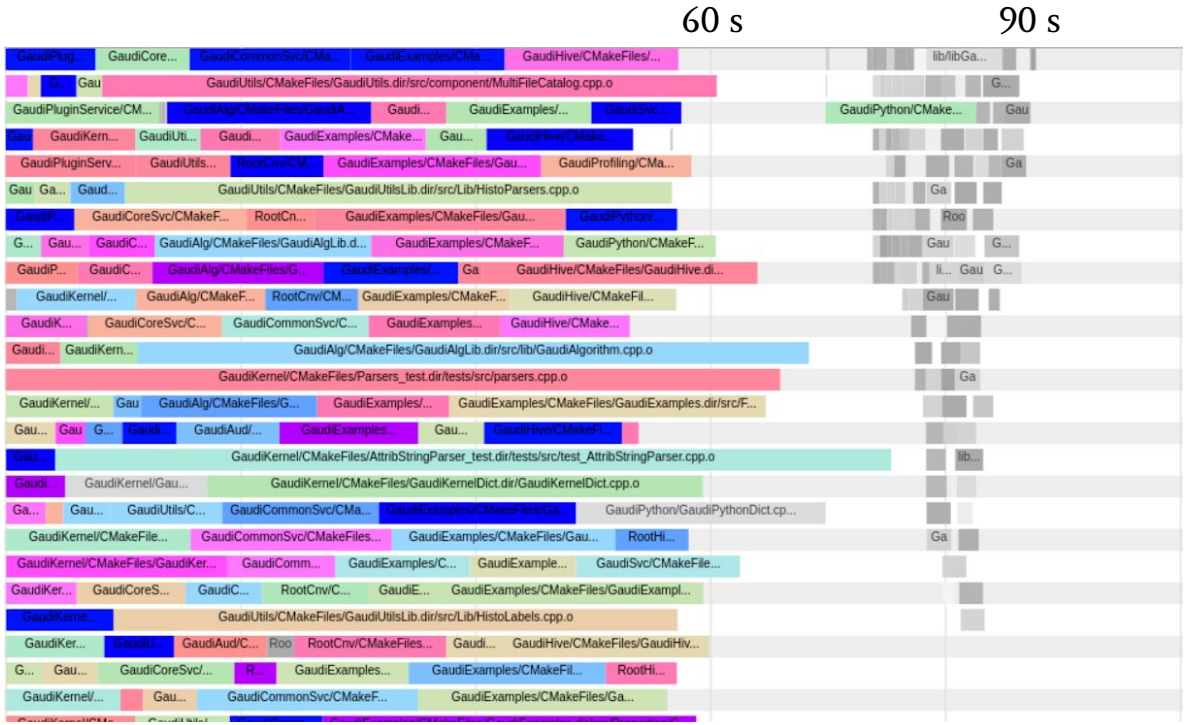


distcc pump mode

- couple with ccache depend mode
- no local preprocessing
- send source and headers to server
  - an “include server” statically analyzes sources to find transitive header dependencies
  - do **not** send headers on /cvmfs
- get object (and dep) file back, ccache it
- moderate network traffic (opt build)
  - ⇑ 40 MiB (1.5 MiB/s peak)
  - ⇓ 75 MiB (4 MiB/s peak)



Amdahl's law for Gaudi builds

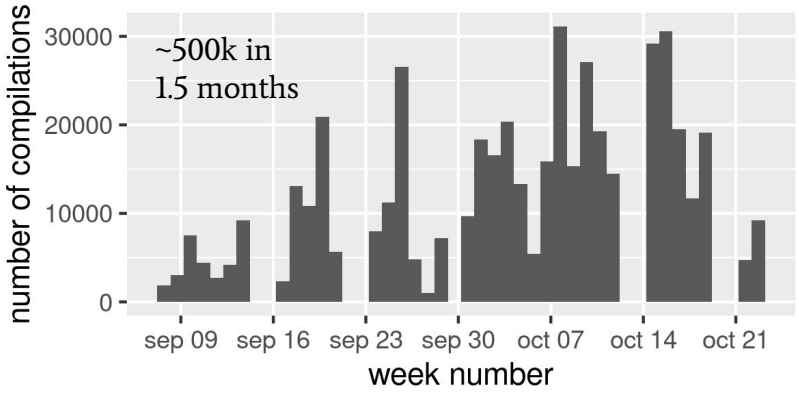


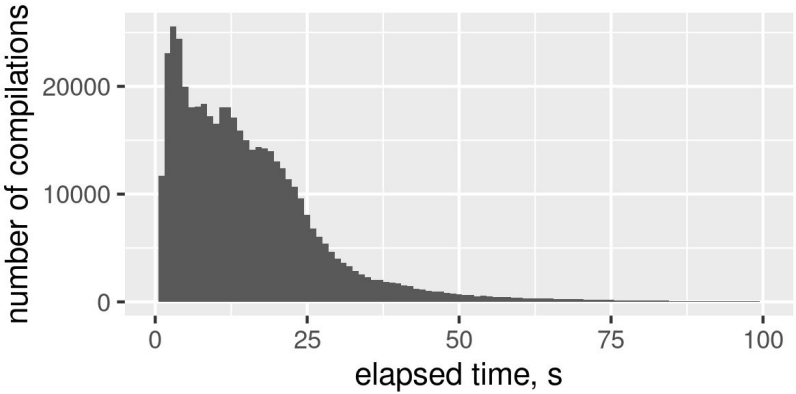
100

...

Experience so far

- Generally happy users, as no need to work on the big server where they have little control / influence or no access at all
- Load balancing by randomization not always fine
- Timeouts when overloaded
- About 15% errors (timeouts, auth failures, user interruptions)



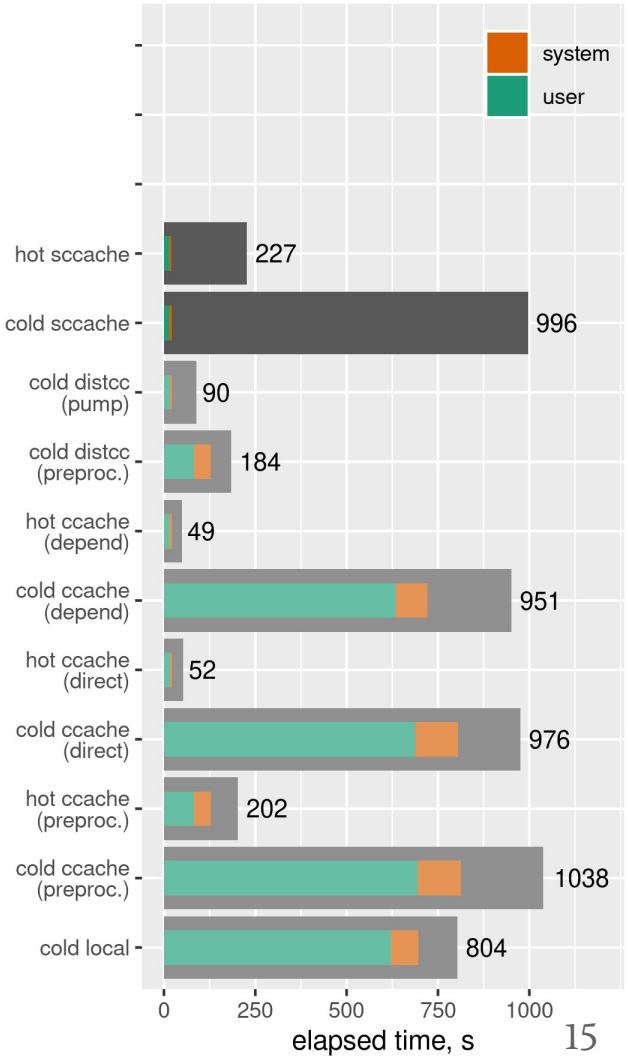


Sharing a cache is difficult

- Ultimately we want to share compilation cache between devs, but
  - Developers typically use different directories
    - Some information included in the hash can contain absolute paths (passed with -I, as a result of including symbols with -g)
  - You need to be smart with which flags can escape the hash
- ccache covers this but it relies on a local filesystem (for now)
  - many use NFS, but performance not evident and hard to do at CERN
- Is there something else on the market?

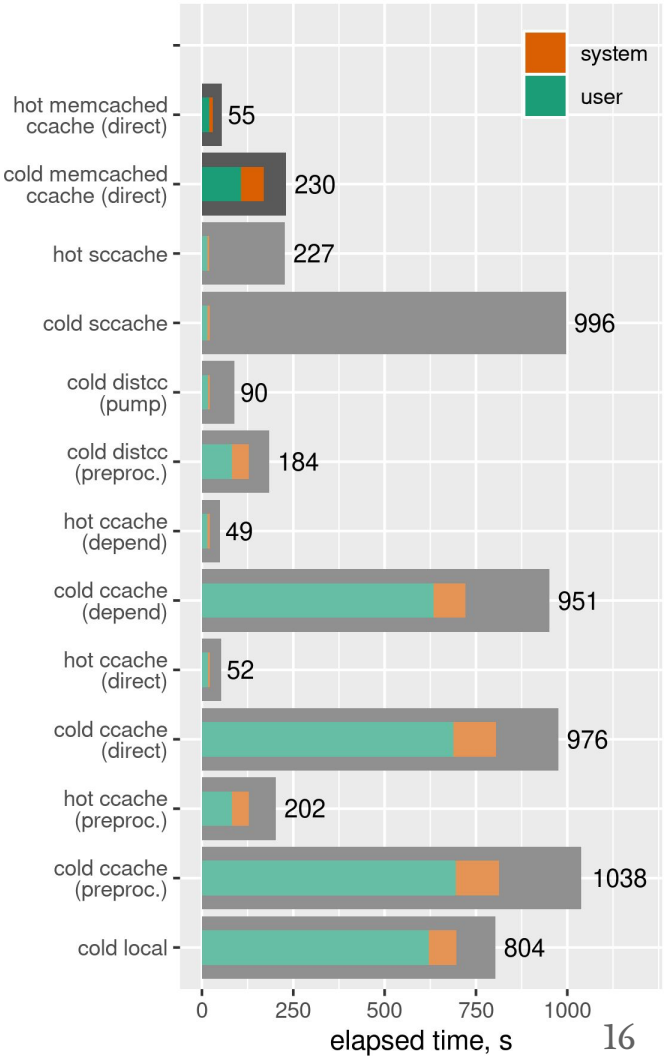
sccache (Mozilla)

- 🙂 Like ccache, used as a compiler wrapper
- 🙂 Stores cache in a remote storage using the S3 API, GCS API, or Redis
- 🙂 Client-server architecture for efficient communication with remote storage
- 🙂 Comes with nice distributed compilation
- ☹️ Unlike ccache, quite simple in what is hashed ⇒ hits might be hard to get
- ☹️ Always runs the preprocessor



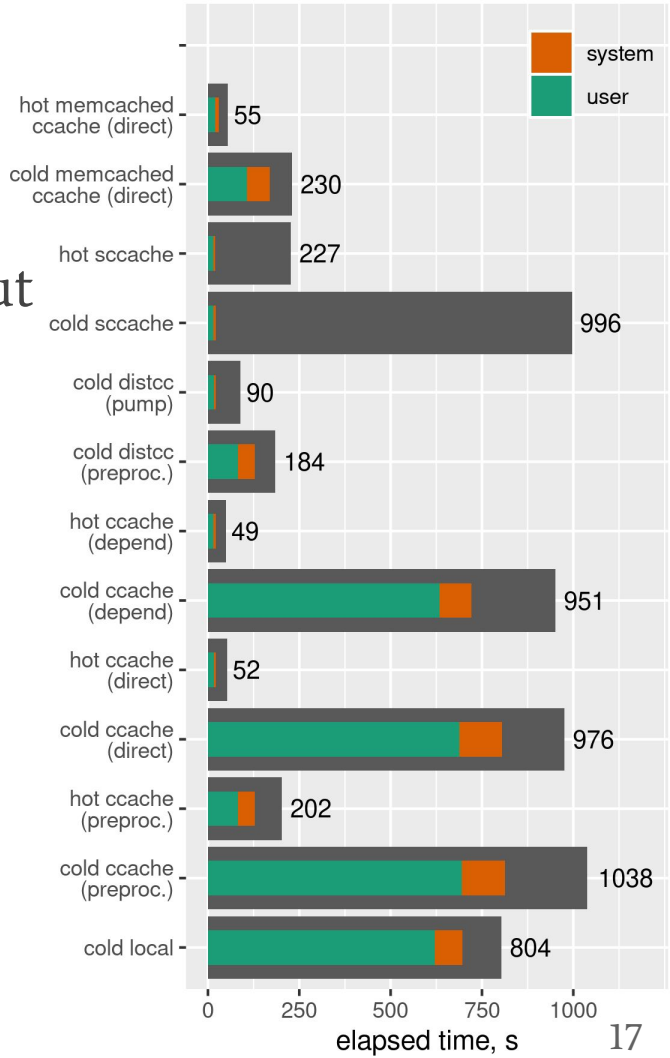
ccache with memcached

- memcached storage backend available in a fork
  - quite behind master (no depend mode)
- promising performance
  - but not used yet
- generic storage extension mechanism <u>planned</u> via plugins



Conclusion and outlook

- ccache and distcc are nearly 20 years old but still maintained and very useful
- proven utility in sharing compilation resources between developers
- 9-fold speedup of Gaudi compilation time
- potential for for automated builds (CI) where feedback speed is also essential
- explore Bazel to distribute *anything*
  - much harder to buy into, much more rewarding



Thank you