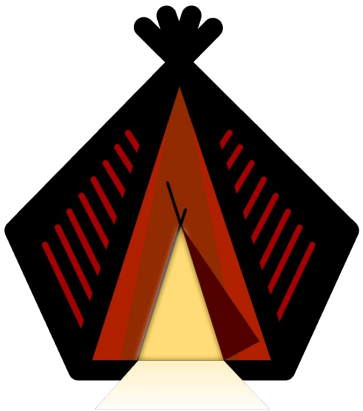


Tipi.build

tipi.build

by EngFlow

tipi.build by EngFlow

How CMake Works

EngFlow Summit, Salzburg

Damien Buhl
Tuesday April 1st, 2025

Today’s Talk

CMake vs Bazel

BAZEL

```
### BUILD FILES
# MODULE.bazel
bazel_dep(name = "com_google_googletest")

# BUILD.bazel
cc_library(
    name = "foo",
    srcs = ["foo.cc"],
    hdrs = ["foo.h"],
)
cc_binary(
    name = "bar",
    srcs = ["bar.cc"],
    deps = [":foo"],
)
cc_test(
    name = "foo_test",
    srcs = ["foo_test.cc"],
    deps = [
        ":foo",
        "@com_google_googletest//:gtest_main",
    ],
)
```

CMake

```
# CMakeLists.txt
FetchContent_Declare(
  GTest
  URL https://...
)
FetchContent_MakeAvailable(GTest)

add_library(foo foo.cc)
add_executable(bar bar.cc)
target_link_libraries(bar foo)

add_executable(foo_test foo_test.cc)
target_link_libraries(foo_test
  foo
  GTest::gtest_main)

include(GoogleTest)
gtest_discover_tests(foo_test)
```

CMake RE

Cross Platform Development Today

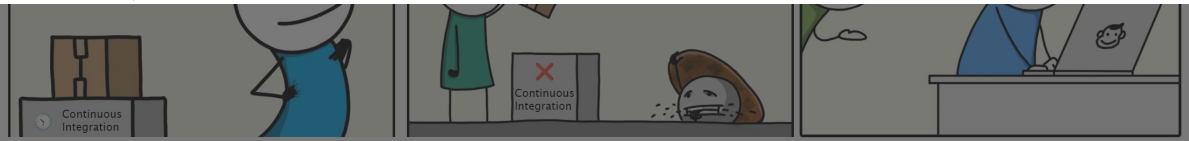
Cross Platform Development Today

Cross Platform Development Today

Commits on Feb 5, 2021

🔧 test if it works on windows (again) daminetreg committed on 5 Feb ✓	Verified	0f9343c
🔧 test if the fix for macos works on windows too daminetreg committed on 5 Feb ✗	Verified	fbd7ce3
😉 the linux fix just broke the code on macos, fix that should work for macos daminetreg committed on 5 Feb ✓	Verified	58ddf8a
🔧 fixed the code to also work on linux, paths are different on windows daminetreg committed on 5 Feb ✗		bf77f08
🔧 linux build fixes actually breek windows, not the same API daminetreg committed on 5 Feb ✗	Verified	da27496
🔧 fix build on linux daminetreg committed on 5 Feb ✗	Verified	0897690
FEATURE: New configuration UI and save daminetreg committed on 5 Feb ✗	Verified	f461ac9

Commit signature



Shifting CI left

CI at developers' fingertips

✅

CI at developers' fingertips

Commits on Oct 4, 2023

FEATURE: New configuration UI for windows, linux and macos. ···

daminetreg committed 2 weeks ago ✓

Verified 72bef4e

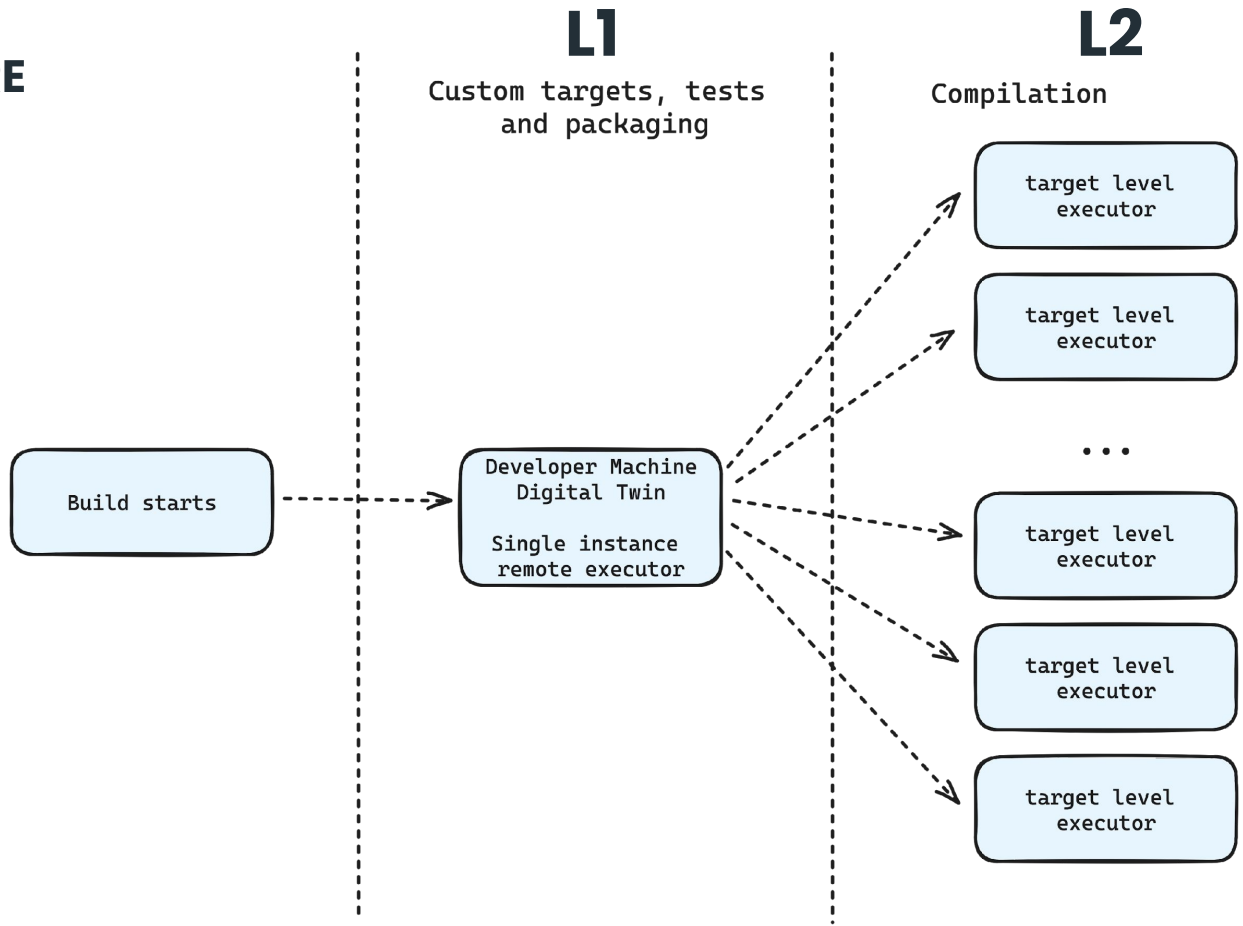
How we do it

- → Content addressable, multi-layer build caching
- → Cloud based build & test parallelization
- → Build graph based auto-scaling

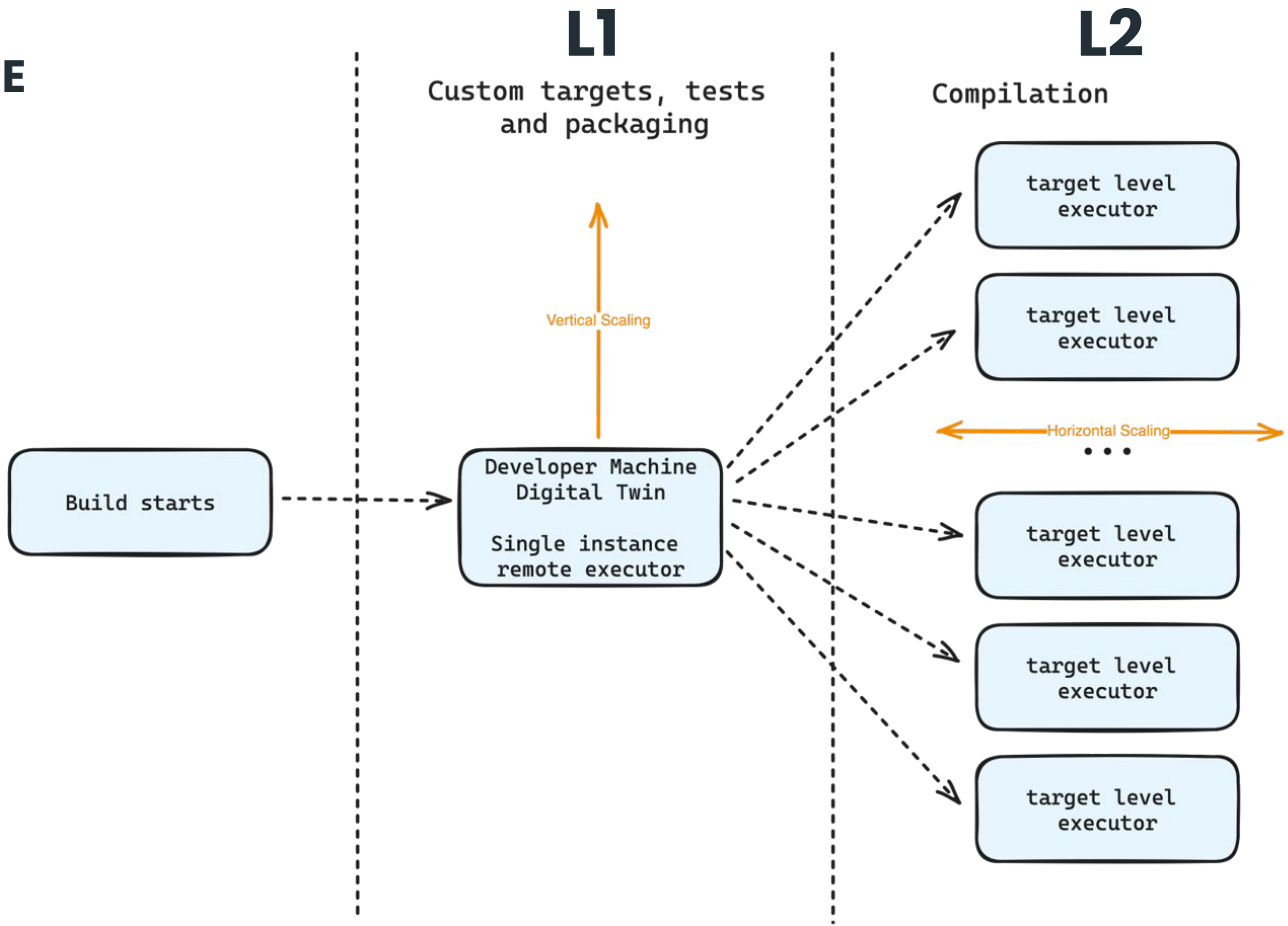
Open Standards and Protocols

- → CMake
- → Bazel RE-API
- → Git connected build caching

CMake RE

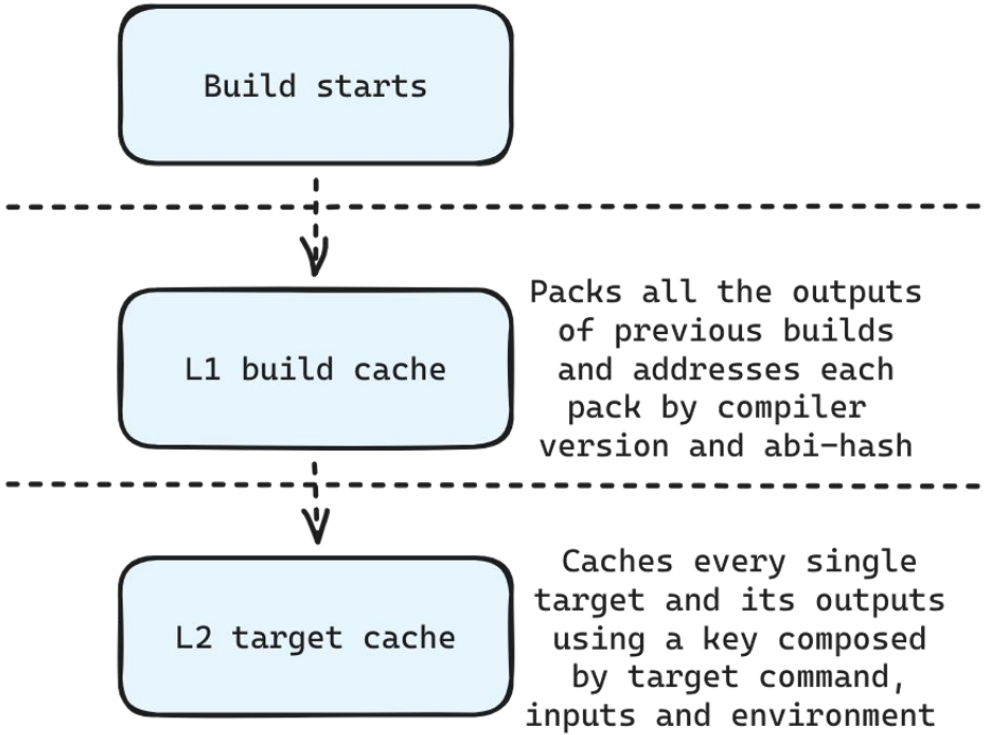

L1
Custom targets, tests and packaging
L2
Compilation
Build starts
Developer Machine Digital Twin
Single instance remote executor
target level executor
target level executor
target level executor
target level executor
target level executor

CMake RE

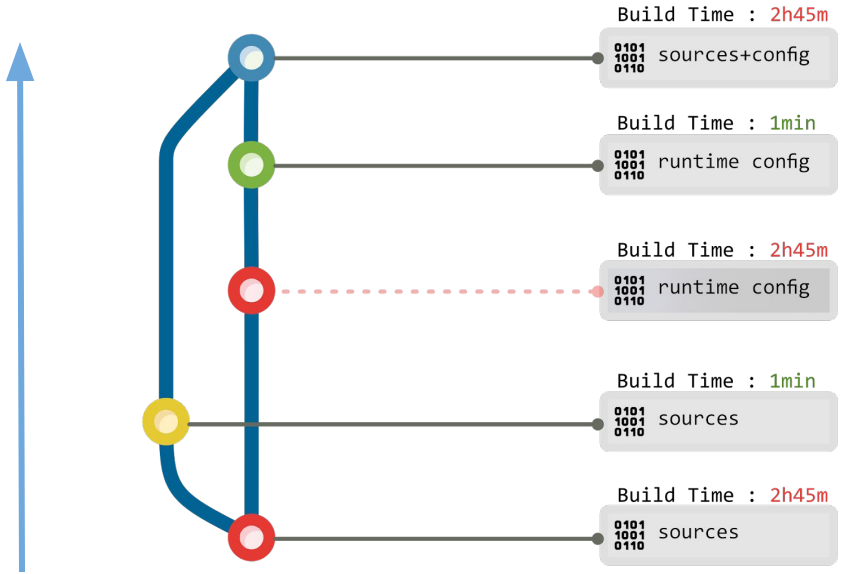

L1
Custom targets, tests and packaging
L2
Compilation
Vertical Scaling
Horizontal Scaling
Build starts
Developer Machine Digital Twin
Single instance remote executor
target level executor
target level executor
target level executor
target level executor
target level executor

Why 2 Layers of Caching ?

CMake RE Caches

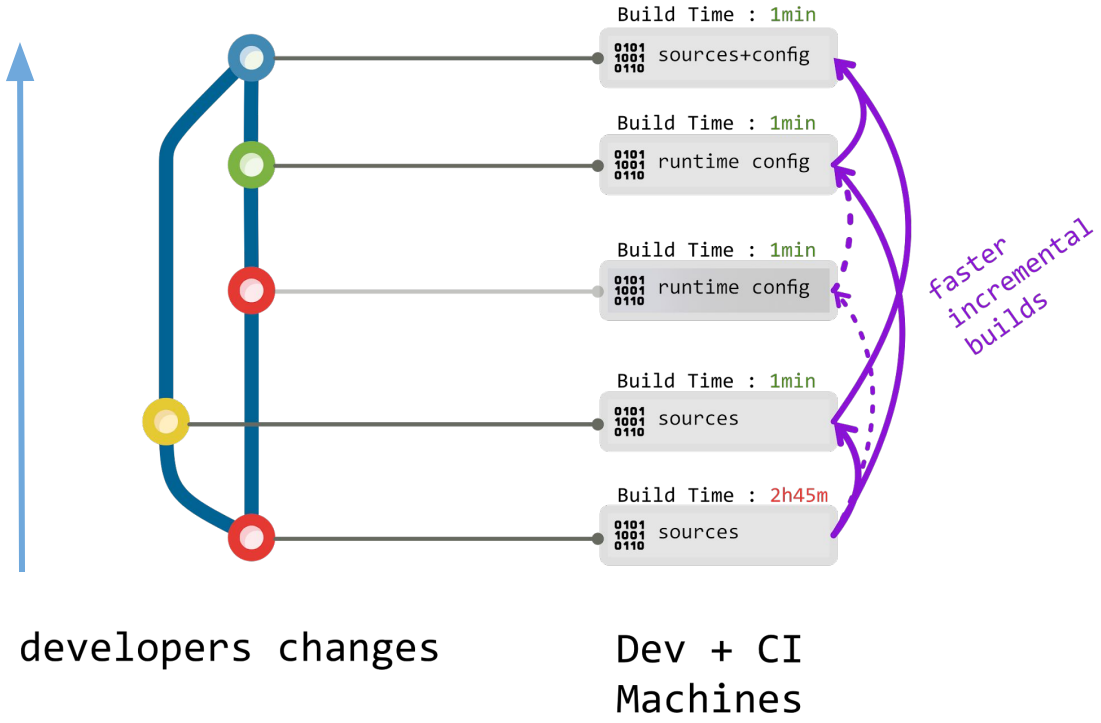


Why CMake RE L1 Build Cache ?



→ **Branching for an hotfix**
→ **Without** CMake RE Cache **2h45min** again, sources get rebuilt due to the change represented by the yellow dot which invalidated the build tree

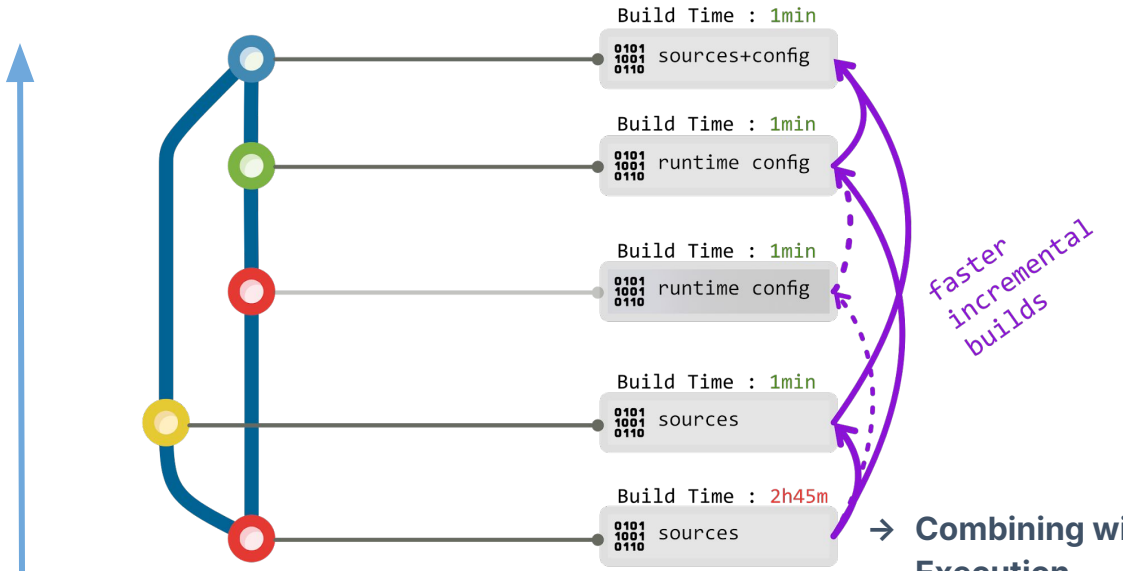
Why CMake RE L1 Build Cache ?



→ **Branching for an hotfix**
→ **With** CMake RE Cache
1min because the first build snapshot is restored by cmake-re

tipi.build by EngFlow

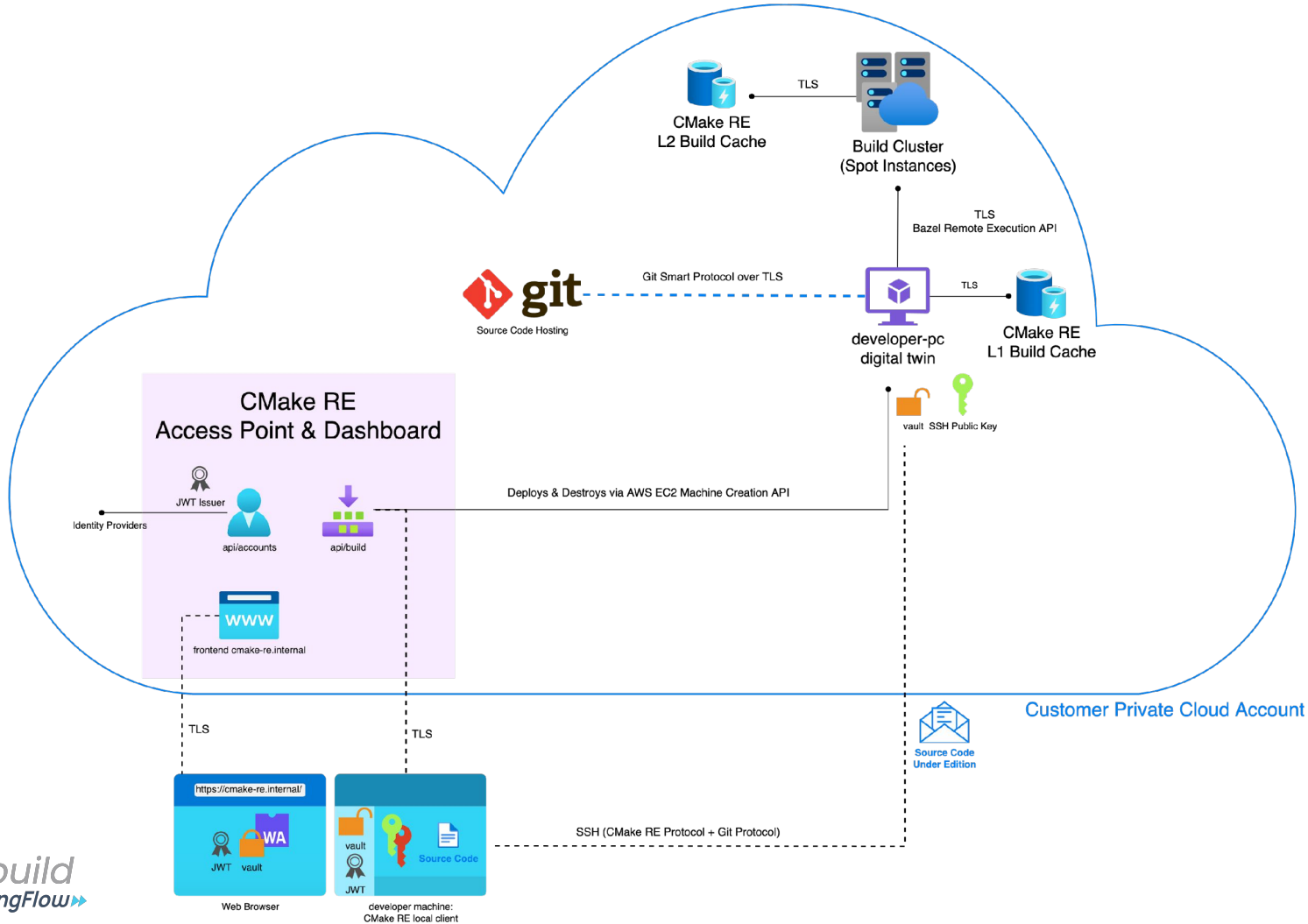
Why CMake RE L1 Build Cache ?



→ **Combining with L2 EngFlow Remote Execution**

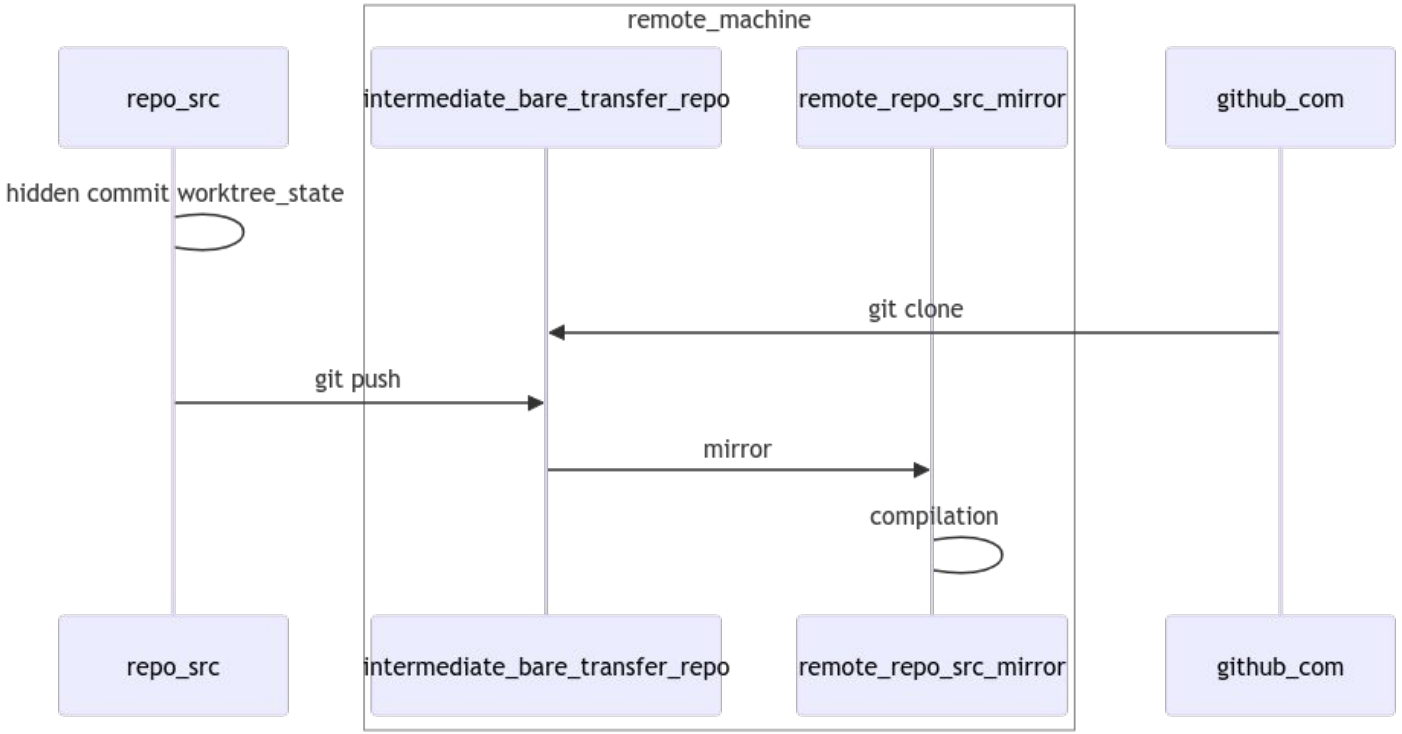
The initial **2h45min** gets to just **1min-5min !**

tipi.build by EngFlow



tipi.build
by EngFlow

CMake RE – Source Remoting to digital twin



We just started the CMake RE Public Beta

L1 Local Containerized Build Beta

→ Available in public for Download now

L1 + L2 Remote Build Beta

→ Available under request and special deployment

Hermetic FetchContent (HFC)

Why we built HermeticFetchContent (HFC)

→ FetchContent is the growing simple package manager in use in many CMake Projects

→ It add_subdirectory() dependencies which makes high build graph parallelism

BUT

→ Plain FetchContent only works with CMake and properly written CMake

→ Requires all dependencies sources to be downloaded ⏳

→ Each build tree need to redownload the sources 💾💾💾

What is Hermetic FetchContent (HFC)

- → Foreign build system support like *autotools*, *OpenSSL*
- → SBOM Generation
- → Source Caching for FetchConent
- → Dependency Install tree caching when run within cmake-re

FetchContent

```
# Pure CMake
FetchContent_Declare( Boost
  GIT_REPOSITORY "https://github.com/boostorg/boost.git"
  GIT_TAG "ab7968a0bbcf574a7859240d1d8443f58ed6f6cf"
)

FetchContent_MakeAvailable( Boost )
```

Hermetic FetchContent

```
# HFC
FetchContent_Declare( Boost
  GIT_REPOSITORY "https://github.com/boostorg/boost.git"
  GIT_TAG "ab7968a0bbcf574a7859240d1d8443f58ed6f6cf"
)

FetchContent_MakeHermetic( Boost )

# either:
HermeticFetchContent_MakeAvailableAt ConfigureTime(Boost)
# or:
HermeticFetchContent_MakeAvailableAt BuildTime(Boost)
```

tipi.build by EngFlow

Hermetic FetchContent

```
# HFC
FetchContent_MakeHermetic( <name>

  HERMETIC_BUILD_SYSTEM cmake|autotools|openssl

  HERMETIC_FIND_PACKAGES <list of exposed dependencies>

  HERMETIC_TOOLCHAIN_EXTENSION <cmake code>

  HERMETIC_CREATE_TARGET_ALIASES <cmake code>
  HERMETIC_CMAKE_EXPORT_LIBRARY_DECLARATION <cmake code>

  SBOM_LICENSE "<SPDX-Identifier>"
  SBOM_SUPPLIER "<Author>"

)
```

Docs : https://tipi-build.github.io/hfc/

HERMETIC_TOOLCHAIN_EXTENSION

→ **HERMETIC_TOOLCHAIN_EXTENSION** is CMake Code as a *bracket argument* that is appended to the CMAKE_TOOLCHAIN_FILE when building the dependency

Use Cases

→ Isolate dependency build with their own parameters
→ Avoid polluting project scope and other dependency variable scope
→ Allows conflicting parameters between project and between different dependencies.

HERMETIC_TOOLCHAIN_EXTENSION

```
FetchContent_Declare( Boost
 GIT_REPOSITORY "https://github.com/boostorg/boost.git"
 GIT_TAG "ab7968a0bbef574a7859a40d1d8448f98ed6f6cf"
)

FetchContent_MakeHermetic( Boost
 HERMETIC_TOOLCHAIN_EXTENSION [=[
  set(BUILD_TESTING OFF)
  set(BOOST_IOSTREAMS_ENABLE_ZLIB ON)
  set(BOOST_IOSTREAMS_ENABLE_BZIP2 ON)
 ]=]
)

HermeticFetchContent_MakeAvailableAtBuildTime( Boost )
```

HERMETIC_TOOLCHAIN_EXTENSION

```
FetchContent_Declare(  Boost
 GIT_REPOSITORY  "https://github.com/boostorg/boost.git"
 GIT_TAG "ab796cacbbcfb74a736594ed1d8445f3edcf0f"
)

FetchContent_MakeHermetic(  Boost
 HERMETIC_TOOLCHAIN_EXTENSION  [=[
  set(BUILD_TESTING OFF)
  set(BOOST_IOSTREAMS_ENABLE_ZLIB ON)
  set(BOOST_IOSTREAMS_ENABLE_BZIP2 ON)
 ]=]
)

HermeticFetchContent_MakeAvailableAtBuildTime(  Boost )
```

HFC: Leveraging Targets Data

→ The CPS datamodel is in most cases fully known as early as **configure time**

→ CPS describes **the useful artifacts of a software package**

→ Scanning targets, leveraging the CPS datamodel enables to predict how the **install tree** will be

It **forwards declare** the build system "API" of a project

```
add_library(lib src/lib.cpp)
target_include_directories(lib PUBLIC
 $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
 $<INSTALL_INTERFACE:include>
)
```

HFC: Leveraging Targets Data

→ The CPS datamodel is in most cases fully known as early as **configure time**

→ CPS describes **the useful artifacts of a software package**

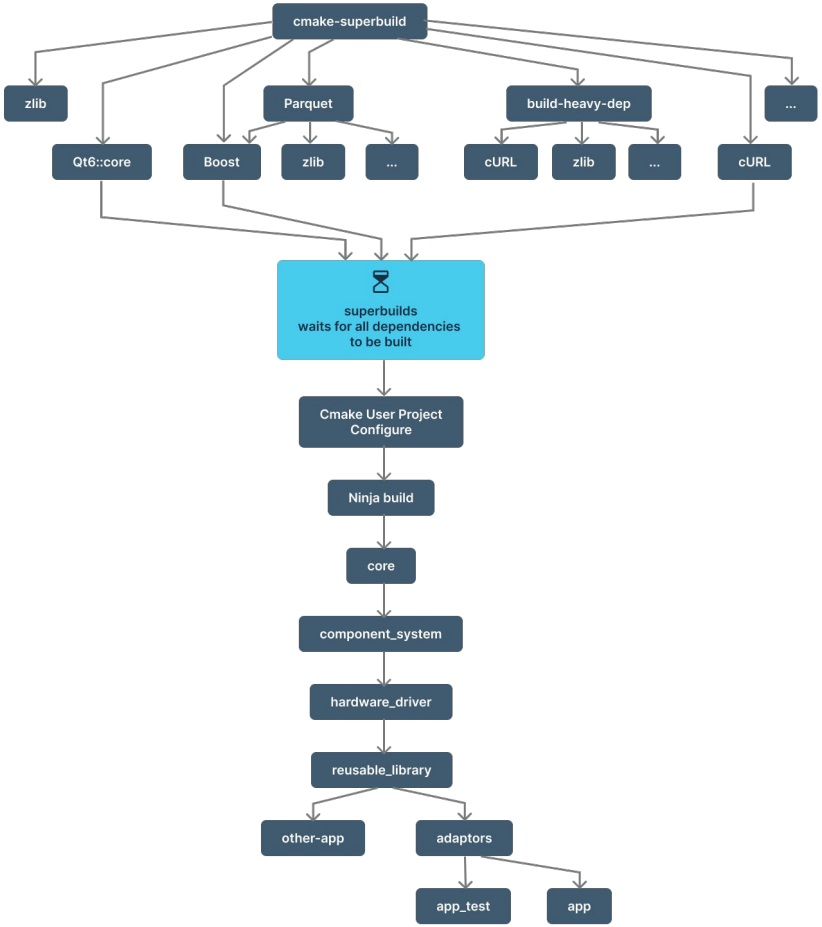
→ Scanning targets, leveraging the CPS datamodel enables to predict how the **install tree** will be

It **forwards declare** the build system "API" of a project

```
add_library(lib src/lib.cpp)
target_include_directories(lib PUBLIC
 $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
 $<INSTALL_INTERFACE:include>
)
```

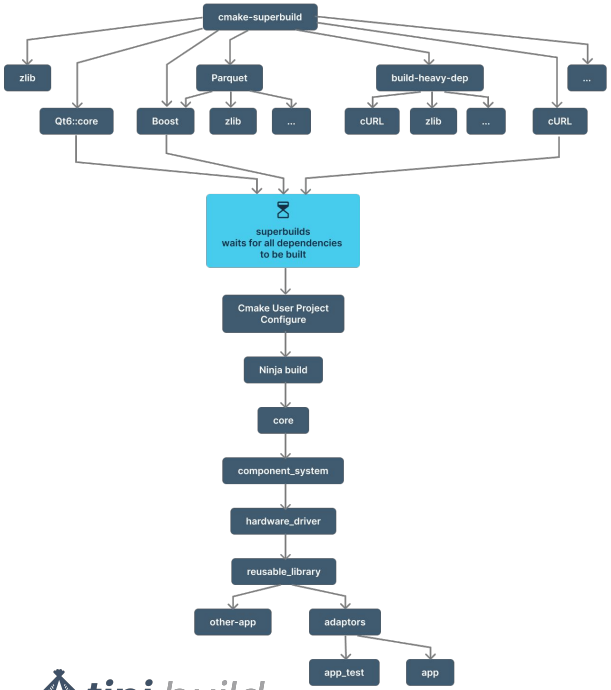
Leveraging this information early we could parallelize dependencies build with dependent code build.

Getting rid of the superbuild dependency barrier

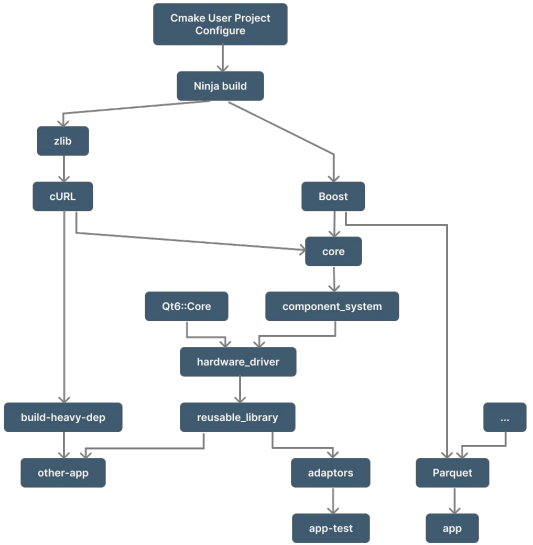

cmake-superbuild
zlib
Parquet
build-heavy-dep
...
Qt6::core
Boost
zlib
...
cURL
zlib
...
cURL
superbuilds
waits for all dependencies
to be built
Cmake User Project
Configure
Ninja build
core
component_system
hardware_driver
reusable_library
other-app
adaptors
app_test
app

Hermetic FetchContent

A CMake Module to get rid of the great dependency barrier



Hermetic FetchContent



tipi.build by EngFlow

Hermetic FetchContent + CMake RE L2

A CMake Module to get rid of the great dependency barrier

400 Cores

Hermetic FetchContent

400 Cores