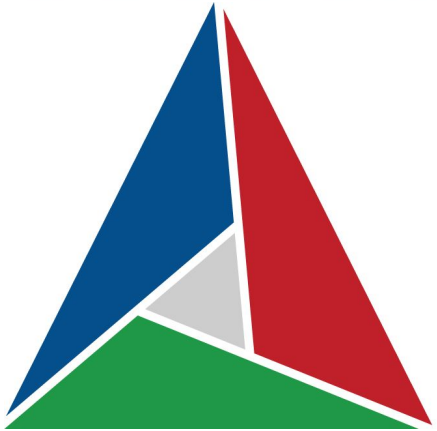
CMake: One Tool To Build Them All

Bill Hoffman

C++ now

2021
MAY 2-7
Aspen, Colorado, USA



CMake

One Tool To Build Them All

Bill Hoffman, CTO @ Kitware creator of CMake

Kitware

Thank You C++ Now

Wish we were in Aspen!

Kitware

Kitware Blog

kitware.com Blog Events Courses Demos

CMake 5.0: One Tool to Build Them All

Kyle Edwards on April 1, 2021

Kitware

Tell them what you're going to say

- Kitware, Open Source, and how CMake came to be
- A High Level Tour of what CMake has to offer
- C++ Modules
- How to Learn CMake
- Packaging C++

We advance science and technology to empower global innovation and meet the world's challenges.

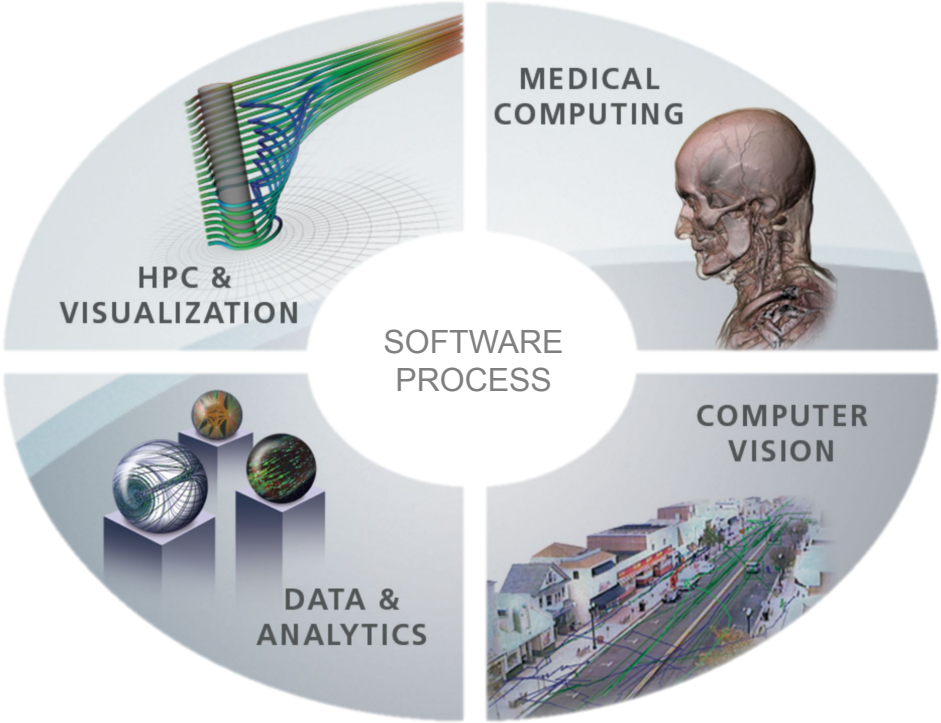
Kitware

This is not what Kitware Does

Kitware

Collaborative software R&D

- Technical computing
- Algorithms & applications
- Software process & infrastructure
- Support & training
- Open source leadership



Supporting all sectors

- Industry, government & academia

Kitware

Kitware's customers & collaborators

Over 75 **academic** institutions…

Harvard
Massachusetts Institute of Technology
University of California, Berkeley
Stanford University
California Institute of Technology
Imperial College London
Johns Hopkins University
Cornell University
Columbia University
Robarts Research Institute
University of Pennsylvania
Rensselaer Polytechnic Institute
University of Utah
University of North Carolina

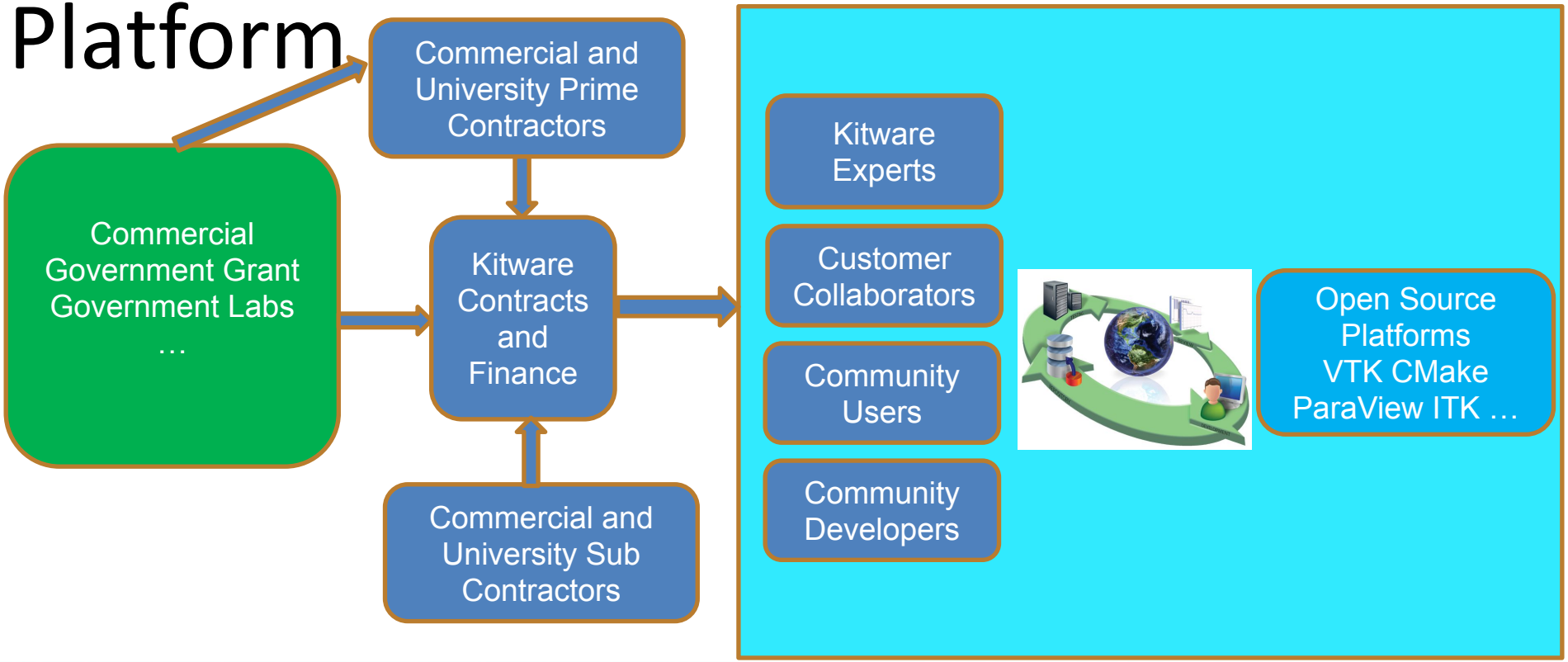
Over 50 **government** agencies and labs…

National Institutes of Health (NIH)
National Science Foundation (NSF)
National Library of Medicine (NLM)
Department of Defense (DOD)
Department of Energy (DOE)
Defense Advanced Research Projects Agency (DARPA)
Army Research Lab (ARL)
Air Force Research Lab (AFRL)
Sandia (SNL)
Los Alamos National Labs (LANL)
Argonne (ANL)
Oak Ridge (ORNL)
Lawrence Livermore (LLNL)

Over 100 **commercial** companies…

Automotive
Aircraft
Defense
Energy technology
Environmental sciences
Finance
Industrial inspection
Oil & gas
Pharmaceuticals
Publishing
3D Mapping
Medical devices
Security
Simulation

Care and Feeding of an Open Source Platform



Open source platforms

VTK & ParaView interactive visualization and analysis for scientific data

ITK & 3D Slicer medical image analysis and personalized medicine research

CMake cross-platform build system

- CDash, CTest, CPack, software process tools

Resonant informatics and infovis

KWIVER computer vision image and video analysis

Simulation, ultrasound, physiology, information security, materials science, …

CMake Offerings

- Courses
- Adding features
- Develop/audit build systems
- Selling printed documentation

Kitware

Unlikely, but way cool contributors

Minecraft CMake Collaboration

Zack Galbreath, Kyle Edwards, Brad King, Robert Maynard and Bill Hoffman on November 4, 2020

Tags: CMake , Software Process

CMakePresets.json CMakeUserPresets.json

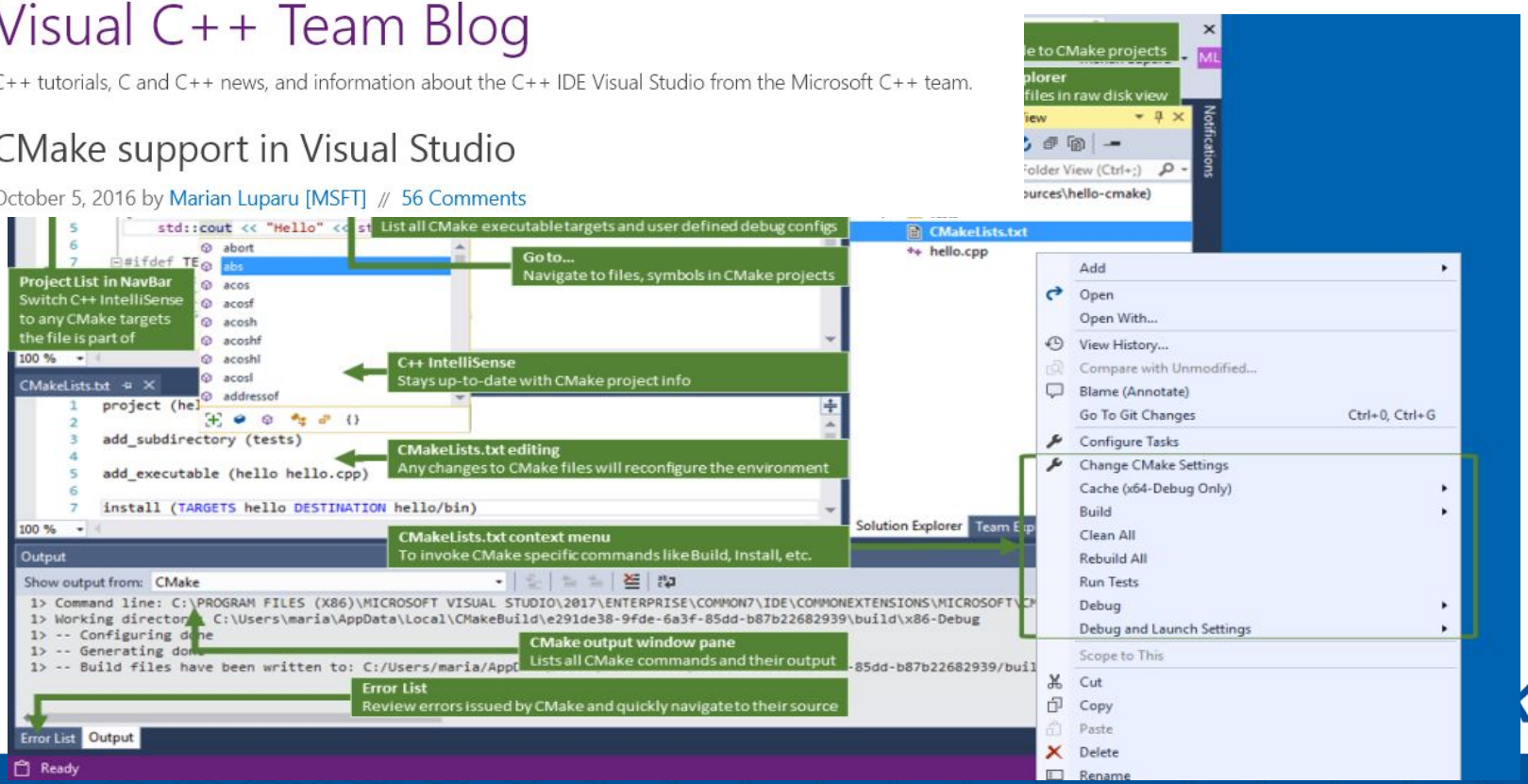
Kitware

Visual C++ Team Blog

C++ tutorials, C and C++ news, and information about the C++ IDE Visual Studio from the Microsoft C++ team.

CMake support in Visual Studio

October 5, 2016 by Marian Luparu [MSFT] // 56 Comments



Bloomberg

Engineering

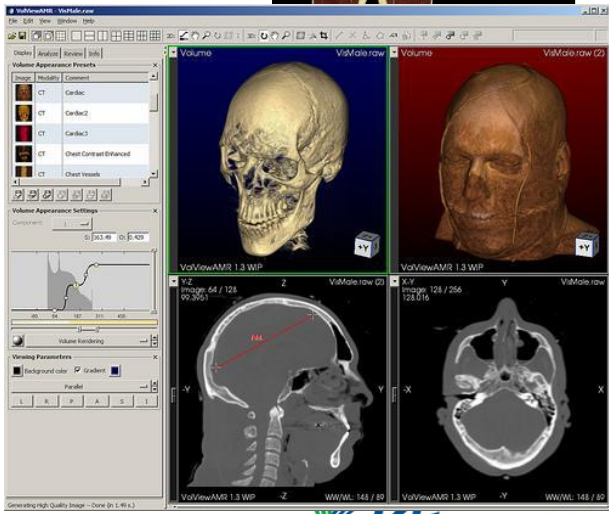
- Improved XCOFF support in CMake
- Improved CMake's file-based API to include more information about installed components
- Both of these enhancements are available in CMake 20

Kitware

Where did CMake come from?

- Kitware was the lead engineering team for the Insight Segmentation and Registration Toolkit (ITK) http://www.itk.org

- Funded by National Library of Medicine (NLM): part of the Visible Human Project
  - Data CT/MR/Slice 1994/1995
  - Code (ITK) 1999
    - Cmake Release-1-0 branch created in 2001



Kitware

How CMake Changes The Way We Build C++

- Boost aims to give C++ a set of useful libraries like Java, Python, and C#
- CMake aims to give C++ compile portability like the compile once and run everywhere of Java, Python, and C#
  - Same build tool and files for all platforms
  - Easy to mix both large and small libraries





Base Rules of CMake

- Only depend on a C++ compiler
- Let developers use the IDE and tools they are most familiar with

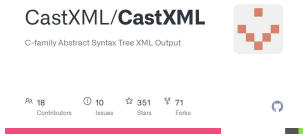
Kitware

ITKWrapping success story

I Installed:


```
git clone https://github.com/InsightSoftwareConsortium/ITK.git
cd ITK
mkdir build
cd build
cmake -DITK_WRAP_PYTHON=ON ..
ninja
```

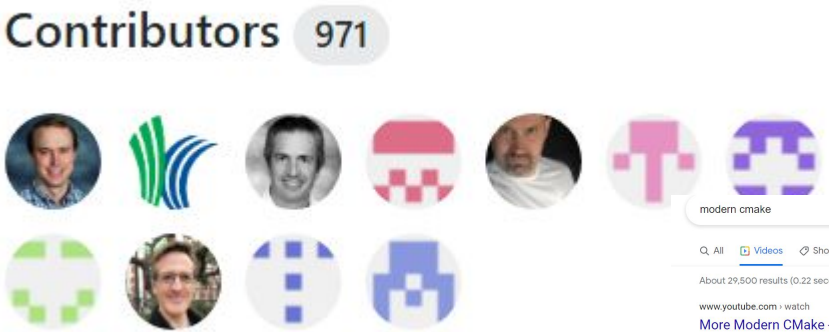
Installed during build:

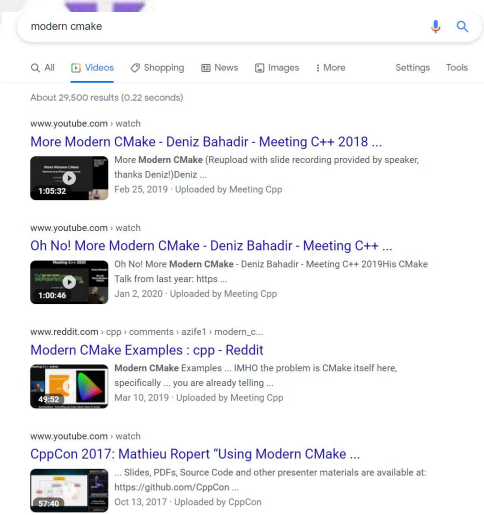


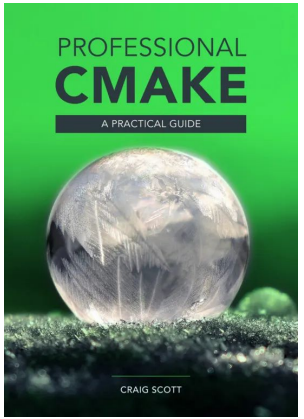
pygccxml

Kitware

CMake is a Community Effort



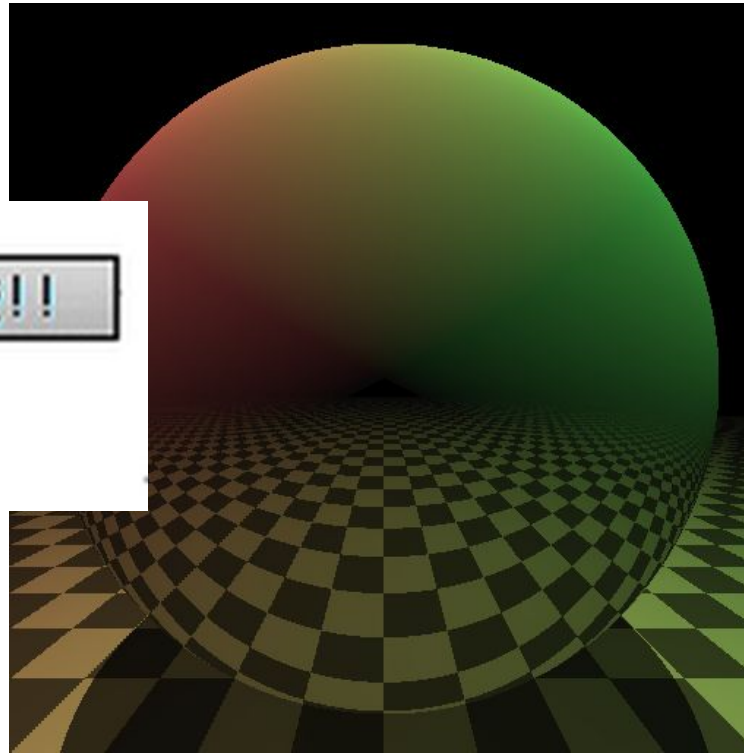




Kitware

CMake raytracer

- Plan to add NVIDIA OPTIX™ RAY TRACING ENGIN to CMake!



Kitware & NVIDIA Collaborations

- Kitware and NVIDIA have collaborated on multiple projects in the last several years in areas of SW dev & Visualization:
- CMake improvements such as “FindCUDA”, and CUDA as a first class language
- ParaView & VTK support for NVIDIA IndeX & Optix
- Coming soon, tighter integration with NVIDIA HPC SDK

Kitware

CMake and CUDA

```
cmake_minimum_required(VERSION 3.18...3.20 FATAL_ERROR)

project(GTC CXX)
option(GTC_ENABLE_CUDA “Enable CUDA” OFF)

if(GTC_ENABLE_CUDA)
  enable_language(CUDA)
  set(CMAKE_CUDA_ARCHITECTURES 70-real 75-real 80)
endif()
target_compile_features(gtc_compiler_flags
                INTERFACE cuda_std_14 cxx_std_14)
set(CMAKE_CXX_EXTENSIONS OFF)
set(CMAKE_CUDA_EXTENSIONS OFF)
```

Add our Library

```
add_library(gtc_lib STATIC serial.cxx)

if(GTC_ENABLE_CUDA)
  target_sources(gtc_lib PRIVATE parallel.cu)
endif()
```

Kitware

Add our Library

```
add_library(gtc_lib STATIC serial.cxx)

if(GTC_ENABLE_CUDA)
  target_sources(gtc_lib PRIVATE parallel.cu)
endif()

target_link_libraries(gtc_lib PUBLIC gtc_compiler_flags)
target_include_directories(gtc_lib
  PRIVATE “${CMAKE_CURRENT_SOURCE_DIR}”
  INTERFACE $<INSTALL_INTERFACE:include/gtc>)
```

Kitware

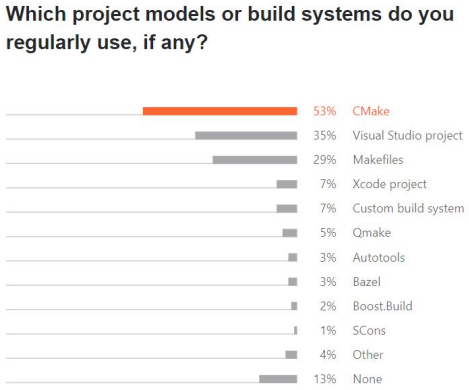
Add our Executable

```
add_executable(gtc)
target_sources(gtc PRIVATE main.cxx)
target_link_libraries(gtc PRIVATE gtc_lib)
```

```
c++  -I/presentations/S9444 -std=c++14 -o <...> -c /presentations/S9444/serial.cxx
nvcc -I/presentations/S9444 -std=c++14 -x cu -c /presentations/S9444/parallel.cu
     -o  <...>
<...>
c++ -std=c++14 -o <...> -c /presentations/S9444/main.cxx
c++  main.cxx.o -o gtc -L/usr/local/cuda/lib64/stubs  -L/usr/local/cuda/lib64
     libgtc_lib.a -lcudadevrt -lcudart_static -lrt -lpthread -ldl
```

Kitware

Jetbrains IDE- CMake is the most popular build tool at 53%



Up from up from 42% the previous year.

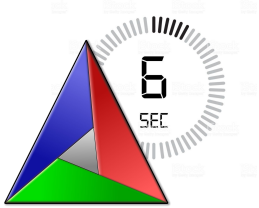
- Job openings requiring CMake experience, March, 2019 Indeed.com, 464 jobs, at Tesla Motors, DCS Corp, Mindsource, Quanergy, ...LinkedIn.com, 486 jobs, at Samsung, Johnson Controls, Apple, Uber, Toyota, Microsoft ...

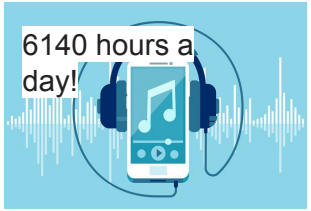
Kitware

CMake

5 million CMake downloads so far this year for total dl size greater than 100 TiB

13809 downloads per day, for a daily rate of 307.3GiB





The Qt Company Decides To Deprecate The Qbs Build System, Will Focus On CMake & QMake

Written by Michael Larabel in Qt on 29 October 2018 at 08:23 AM EDT. 62 Comments

While Qt's Qbs build system was once planned as the default build system for Qt6 and shaping up to be the de facto successor to QMake, there is a change of course with The Qt Company now announcing they are deprecating this custom build system.

In recent months Qbs for Qt 6 began looking less certain and now The Qt Company has announced they are going to deprecate Qbs. From talking with their customers, they decided to focus on QMake and CMake.

Kitware

Why is CMake Popular

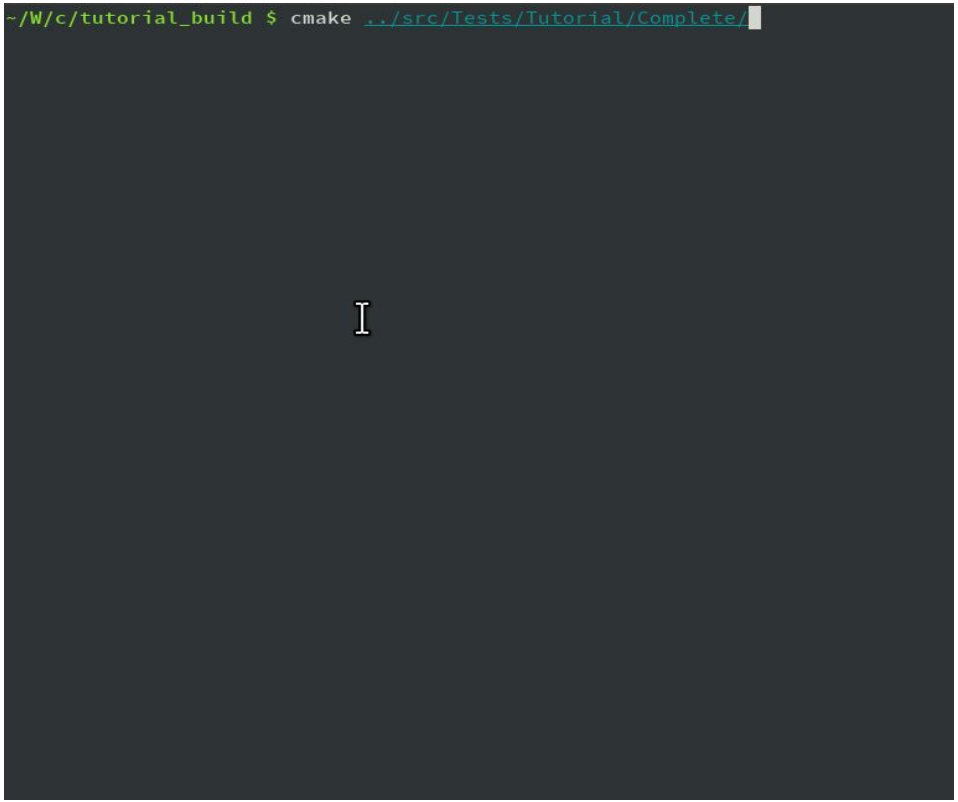
It does a lot with a little!

- add_library()
- add_executable()
- add_test()

A lot happens with those simple commands! (shared libraries, static libraries, many compilers/IDEs, and more)

Kitware

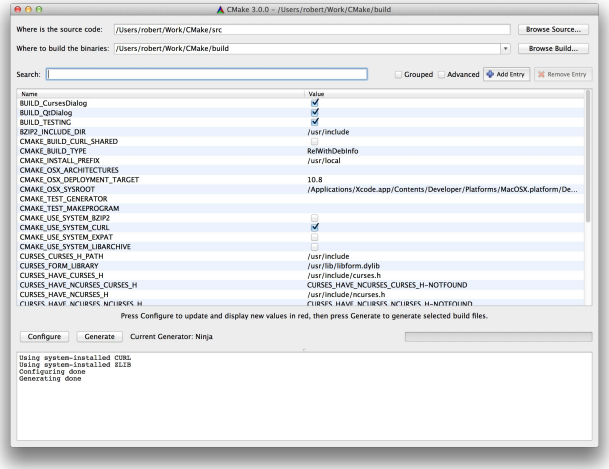
CMake Workflow

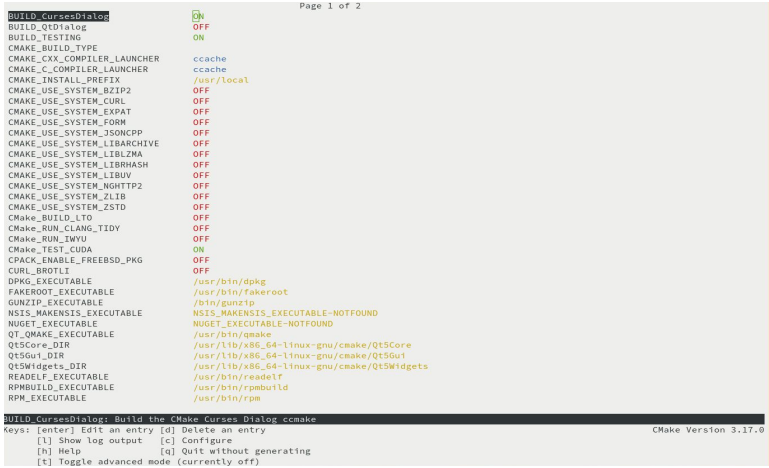


Kitware

Running CMake

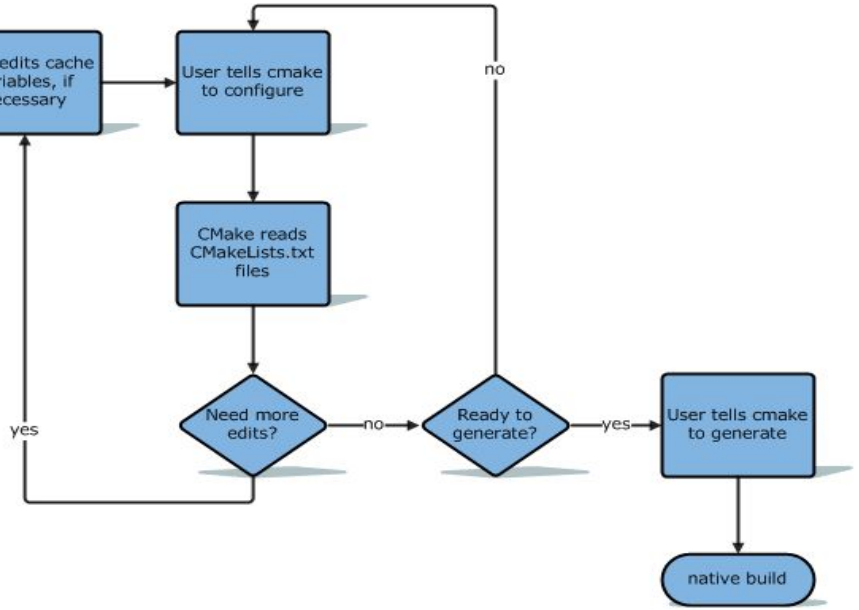
- cmake-gui (the Qt gui)
- ccmake (the terminal cli)
- cmake (non-interactive command line)





Kitware

Running CMake



Kitware

Quickly adapt to new technologies

- New build IDE's and compilers
  - Visual Studio releases supported weeks after beta comes out
  - Xcode releases supported weeks after beta comes out
  - ninja (command line build tool from Google) support contributed to CMake as ninja matured
  - Apple Silicon
- New compiler support
  - clang
  - gcc versions

“Usage Requirements” aka Modern CMake

Modern style: target-centric

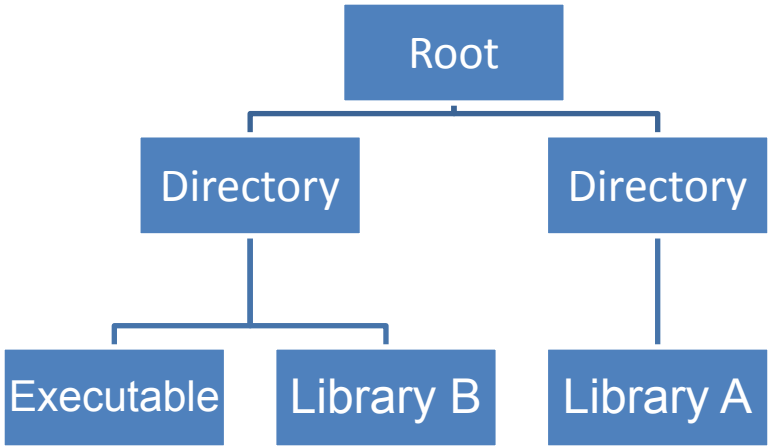
```
target_include_directories(example PUBLIC "inc")
```

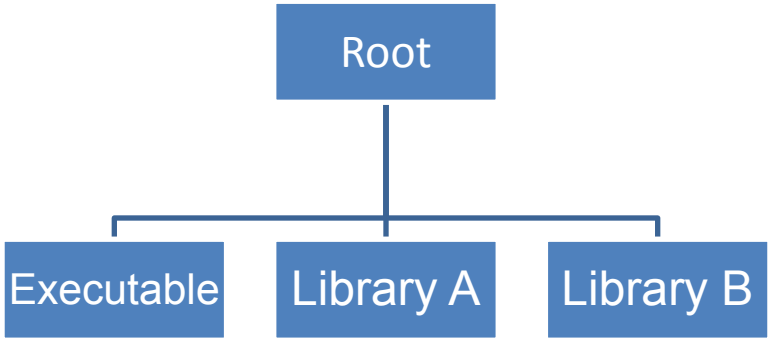
example and anything that links to gets `-Iinc`

- Each target should fully describe how to properly use it
- No difference between external and internal targets

Kitware

Modern CMake





Kitware

Usage Requirements

`PRIVATE:`	Only the given target will use it
`INTERFACE:`	Only consuming targets use it
`PUBLIC:`	**PRIVATE + INTERFACE**
`$<BUILD_INTERFACE>:`	Used by consumers from this project or use the build directory
`$<INSTALL_INTERFACE>:`	Used by consumers after this target has been installed

Kitware

Usage Requirements

```
target_link_libraries(trunk PUBLIC root)
target_link_libraries(leaf PUBLIC trunk)
```

```
/usr/bin/c++ -fPIC -shared -Wl,-soname,libleaf.so
             -o libleaf.so leaf.cxx.o libtrunk.so libroot.so
```

```
target_link_libraries(trunk PRIVATE root)
target_link_libraries(leaf PUBLIC trunk)
```

```
/usr/bin/c++ -fPIC -shared -Wl,-soname,libleaf.so
             -o libleaf.so leaf.cxx.o libtrunk.so
```

Kitware

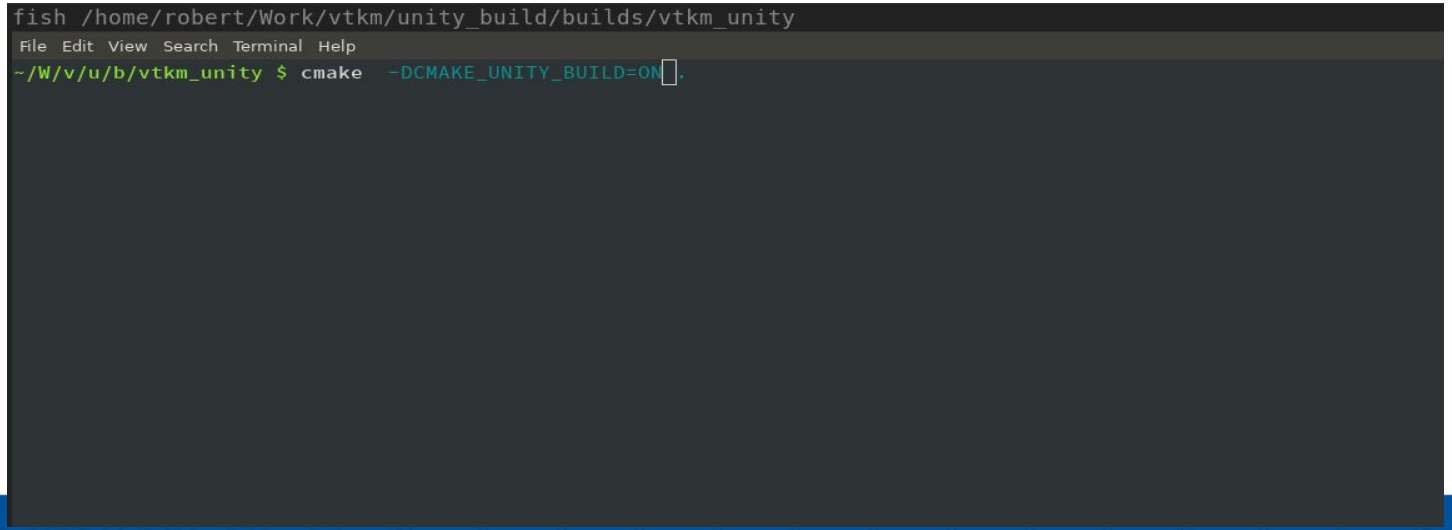
TLL (target link libraries)

- TLL can propagate dependencies when using:
  - `target_include_directories`
  - `target_compile_definitions`
  - `target_compile_options`
  - `target_sources`
  - `target_link_options`

Kitware

Unity / Jumbo Builds

- Control grouping size with CMAKE_UNITY_BUILD_BATCH_SIZE



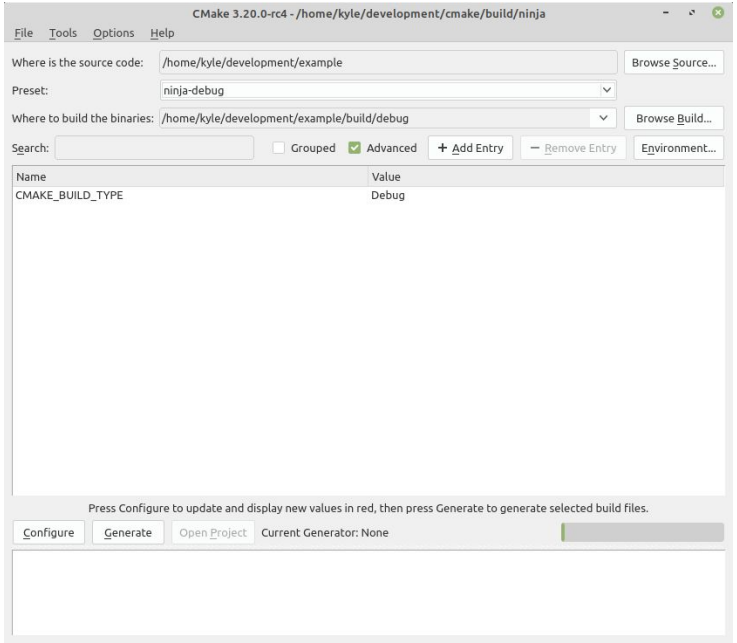
Kitware

Presets

- Allows common configuration flags (variables, build directory, generator, etc.) for a project to be stored in a JSON file for reuse
  - CMakePresets.json - version controlled, for sharing between users
  - CMakeUserPresets.json - not version controlled, for local machine-specific or user-specific use

Presets Example

```
{
  "version": 3,
  "configurePresets": [
    {
      "name": "ninja-debug",
      "generator": "Ninja",
      "binaryDir": "${sourceDir}/build/debug",
      "cacheVariables": {
        "CMAKE_BUILD_TYPE": "Debug"
      }
    }
  ]
}
```



Kitware

Precompiled Headers

- Speed up compilation by creating partially processed header files
- Use those when compiling instead of repeatedly parsing header files

Precompiled Headers

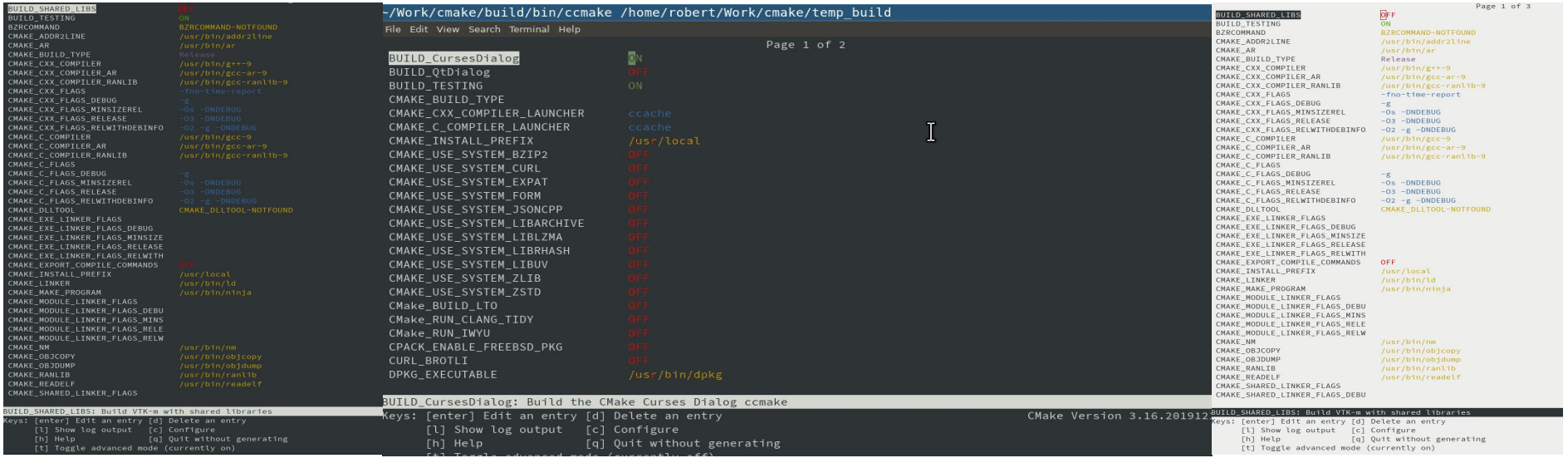
```
add_library(leaf SHARED leaf.cxx)
target_precompile_headers(leaf
  PRIVATE
    <iostream>
    <vector>
    <unordered_map>
  INTERFACE
    "leaf.h")
```

- Multi Config Ninja
- CMAKE_<LANG>_COMPILER_LAUNCHER via ENV

```
export CMAKE_CXX_COMPILER_LAUNCHER=ccache
cmake -G “Ninja Multi-Config” -S ./src -B ./build
...
cmake --build ./build --config Debug -j10 -v
...
/usr/bin/ccache gcc ...
...
ctest --build-config Debug -j10
```

Kitware

ccmake in living color



Kitware

In progress

- Integrate file(GET_RUNTIME_DEPENDENCIES) with install targets

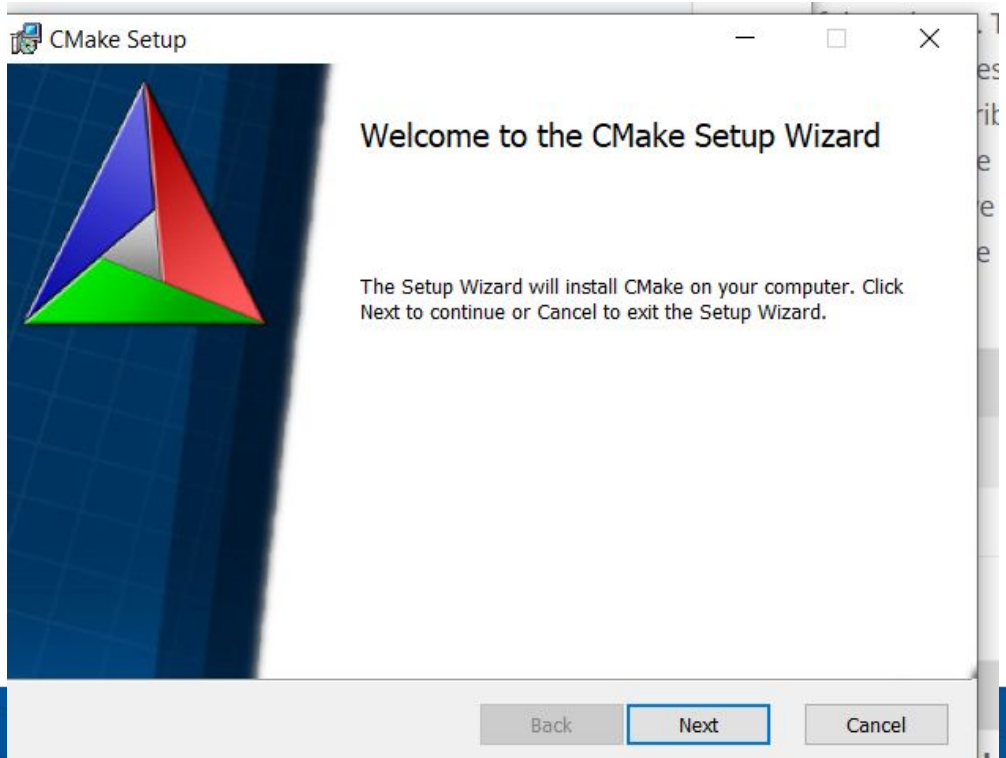
Full cross platform install system

- Specify rules to run at install time
- Can install targets, files, or directories
- Provides default install locations

```
add_library(leaf SHARED leaf.cxx)
install(TARGETS root trunk leaf parasite)
```


Kitware

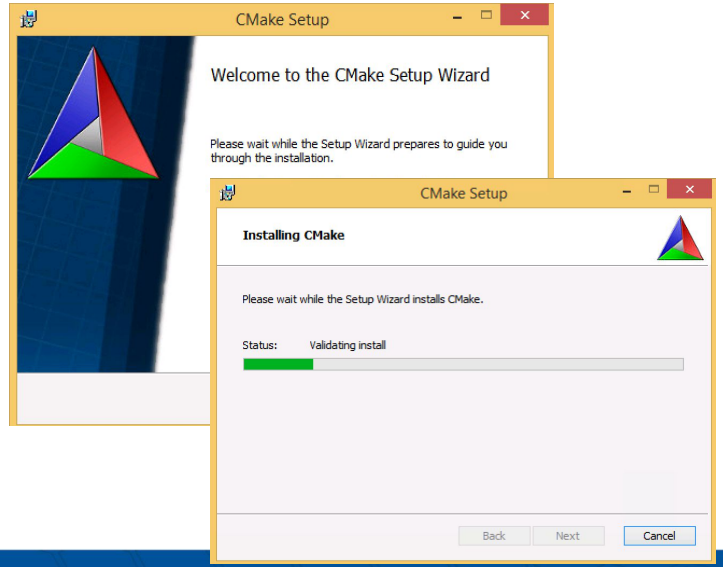
Packaging (dark art… or simple CMake command)

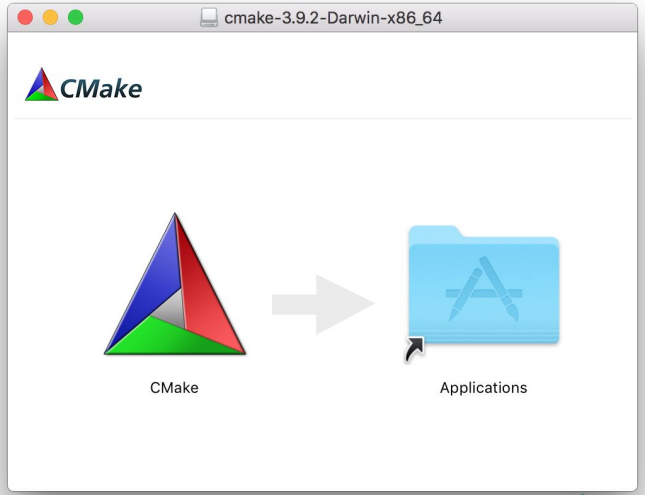


Kitware

CPack

- CPack is bundled with CMake
- Creates professional platform specific installers





Kitware

CPack Features

- Supports CMake-based and non-CMake-based projects
- Unix
  - TGZ and self-extracting TGZ (STGZ)
- Windows
  - WiX – MSI installers
  - NullSoft Scriptable Install System (NSIS / NSIS64)
- Mac OSX
  - DragNDrop
  - PackageMaker
- Deb
  - Debian packages
- RPM
  - RPM package manager

Using CPack

- On Windows install command line ZIP program, NSIS and WiX
- Setup your project to work with cpack
  - Get `make install` to work
    - install(…)
    - make sure your executables work with relative paths and can work from any directory
  - Set cpack option variables if needed
  - `include(CPack)`

Kitware

Testing with CMake

- Testing needs to be enabled by calling `include(CTest)` or `enable_testing()`

```
add_test(NAME testname
         COMMAND exename arg1 arg2 ...)
```

  - Executable should return 0 for a test that passes

- ctest – an executable that is distributed with cmake that can run tests in a project

Kitware

Running CTest

- Run ctest at the top of a binary directory to run all tests

```
$ ctest
Test project /tmp/example/bin
    Start 1: case1
1/1 Test #1: case1 .............................   Passed    0.00 sec
    Start 2: case2
2/2 Test #2: case2 .............................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.01 sec
```

Running CTest

- **`-j`** option allows you to run tests in parallel
- **`-R`** option allows you to choose a test
- **`-VV`** for more verbose output
- **`--rerun-failed`** to repeat failed tests

- **`ctest --help`** for more information

Kitware

GoogleTest integration

```
include(GoogleTest)
add_executable(tests tests.cpp)
target_link_libraries(tests GTest::GTest)
```

- gtest_discover_tests: added in CMake 3.10.
  - CMake asks the test executable to list its tests.
    Finds new tests without rerunning CMake.

```
gtest_discover_tests(tests)
```


Kitware

CTest and multi core tests

- PROCESSOR_AFFINITY - when supported ties processes to specific processors

```
set_tests_properties(myTest PROPERTIES
                     PROCESSOR_AFFINITY ON
                     PROCESSORS 4)
```


Kitware

CDash Example

Scan Build — 1 build

Site	Build Name	Update Revision	Configure Error	Configure Warn	Build Error	Build Warn	Start Time
elysium-linux.kitware	Linux-clang-scanbuild	8c3071	0	0	0	0	10 hours ago

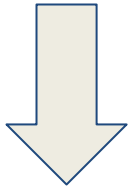
Nightly Expected — 96 of 108 builds

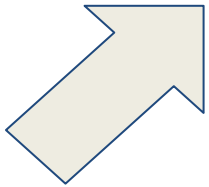
Site	Build Name	Update Revision	Configure Error	Configure Warn	Build Error	Build Warn	Test Not Run	Test Fail	Test Pass	Start Time
trinsic.kitware	vs14-64-ninja	8c3071	0	0	0	0	0	2 +2	479 -2	13 hours ago
trinsic.kitware	mingw64-make	8c3071	0	0	0	0	0	1 +1	449 -1	6 hours ago
hythloth.kitware	Linux-ninja-gcov	8c3071	0	0	0	0	0	1	519	7 hours ago
trinsic.kitware	vs14-64-ide-v90	8c3071	0	0	0	0	0	1 +1 -1	387 -1	8 hours ago
vesper.kitware	watcom	8c3071	0	0	0	0	0	1 +1	368 -1	8 hours ago
trinsic.kitware	vs9-64-ide	8c3071	0	0	0	0	0	1 +1	381 -1	9 hours ago
vesper.kitware	bcc55	8c3071	0	0	0	0	0	1 +1	370 -1	9 hours ago
dash3win7.kitware	vs14-64	8c3071	0	0	0	0	0	1 +1 -1	460 +1 -1	9 hours ago
dashsun1.kitware.com	Solaris-10-8.11_Oracle-12.3	8c3071	0	0	0	0	0	0	415	1 hour ago

Kitware

Dynamic Analysis

Dynamic Analysis			
Site	Build Name	Checker	Defect Count
localhost	my_build	AddressSanitizer	1





Site Name: localhost		
Build Name: my_build		
Name	Status	heap-use-after-free
my_test	Failed	1

==========
==========
heap-use-after-free ==24634==ERROR: AddressSanitizer: heap-use-after-free on address 0x607
READ of size 1 at 0x607000000025 thread T0
    #0 0x4f192a in main /tmp/asan/main.c:5:10
    #1 0x7f4fba4bdf44 in __libc_start_main /build/eglibc-ripdx6/eglibc-2.19/csu/libc-start.c:287
    #2 0x41a918 in _start (/tmp/asan/bin/main+0x41a918)

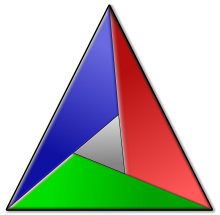
0x607000000025 is located 5 bytes inside of 80-byte region [0x607000000020,0x607000000070)
freed by thread T0 here:
    #0 0x4c23f2 in free (/tmp/asan/bin/main+0x4c23f2)
    #1 0x4f18fa in main /tmp/asan/main.c:4:3
    #2 0x7f4fba4bdf44 in __libc_start_main /build/eglibc-ripdx6/eglibc-2.19/csu/libc-start.c:287

previously allocated by thread T0 here:
    #0 0x4c2773 in __interceptor_malloc (/projects/ctest_sanitizer_examples/asan/bin/use-after-free+0x4c2773)
    #1 0x4f18ef in main /tmp/asan/main.c:3:20
    #2 0x7f4fba4bdf44 in __libc_start_main /build/eglibc-ripdx6/eglibc-2.19/csu/libc-start.c:287

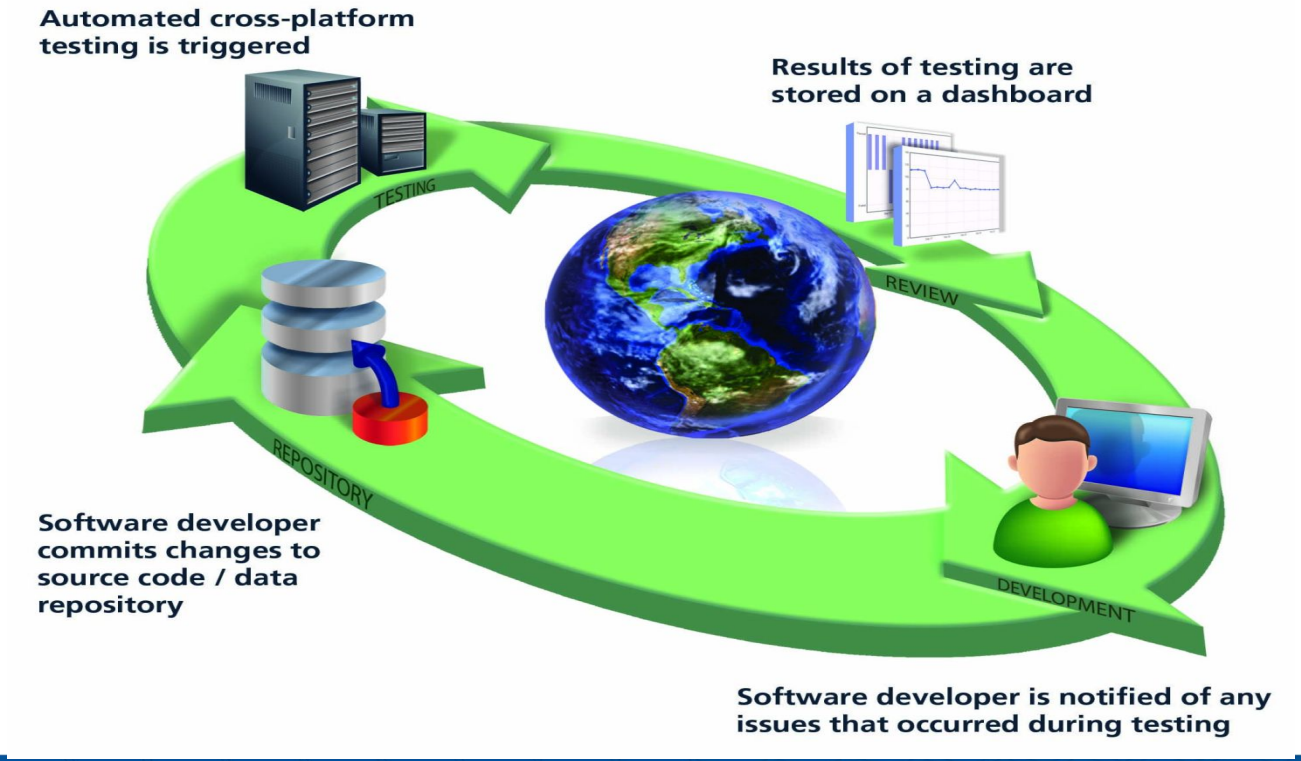
Kitware

Improve CI turnaround with sccache

- Kitware hosted gitlab CI
- Redis server - linux machine with lots of memory
- sccache - integrated into our CI builds with CMAKE_<LANG>_COMPILER_LAUNCHER



Software Process Dashboard



Kitware

C++20 Modules

CMake and the future of C++

```
// helloworld.cpp
export module helloworld;  // module declaration
import <iostream>;         // import declaration

export void hello() {      // export declaration
    std::cout << "Hello world!\n";
}
```

```
// main.cpp
import helloworld;  // import declaration

int main() {
    hello();
}
```

CC -o main.cpp; CC -o helloworld.cpp is now an Error!

Kitware

But wait, CMake has 16 years experience with modules!

- 2005 Initial makefile support for modules added to CMake:

commit 19f977bad7261d9e8f8d6c5d2764c079d35cc014
Author: Brad King <brad.king@kitware.com>
Date: Wed Jan 26 15:33:38 2005 -0500
ENH: Added Fortran dependency scanner implementation.

- Added support to ninja in 2015 for Fortran dep file depends - funded by the Trilinos project forked ninja
- May 2019 ninja merged all of the changes to support Fortran because of C++ modules!

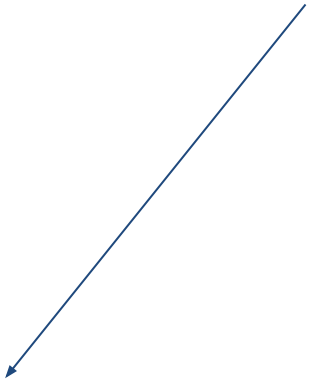
Kitware

How Fortran Modules Work

r

```
module math
contains
  function add(a,b)
    real :: add
    real, intent(in) :: a
    real, intent(in) :: b
    add = a + b
  end function
end module
```

math.o and math.mod

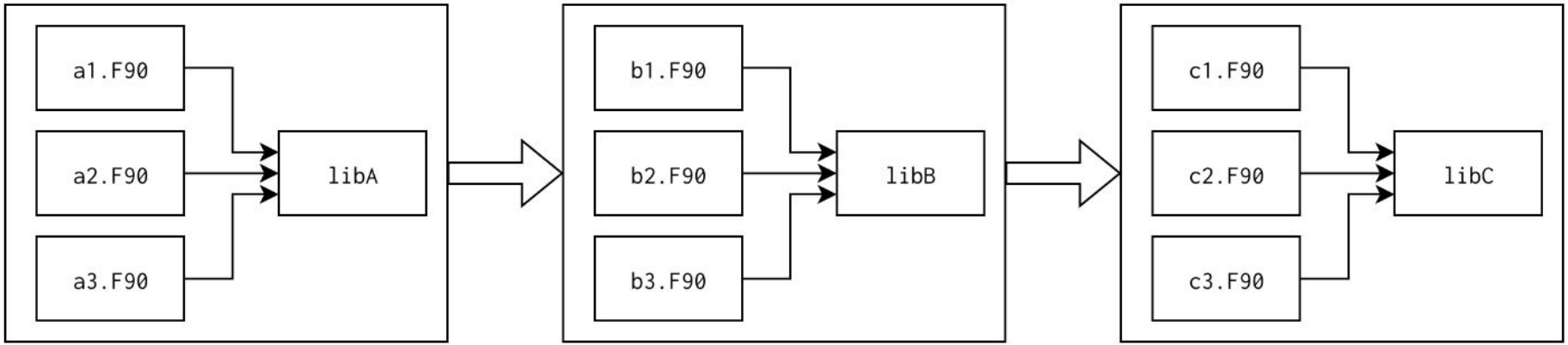


```
program main
  use math
  print *,'sum is',add(1., 2.)
end program
```

main.o needs math.mod

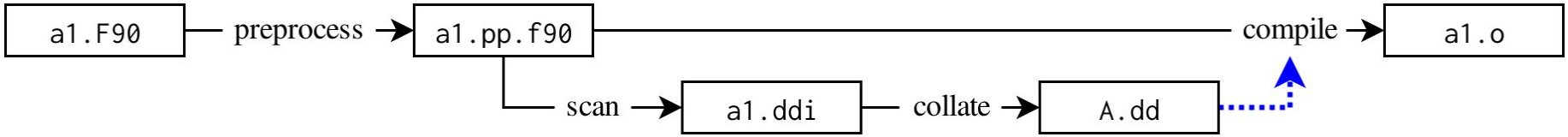
Kitware

Input to CMake



Kitware

Build graph single source



.F90 - original source

.pp.f90 - preprocessed source

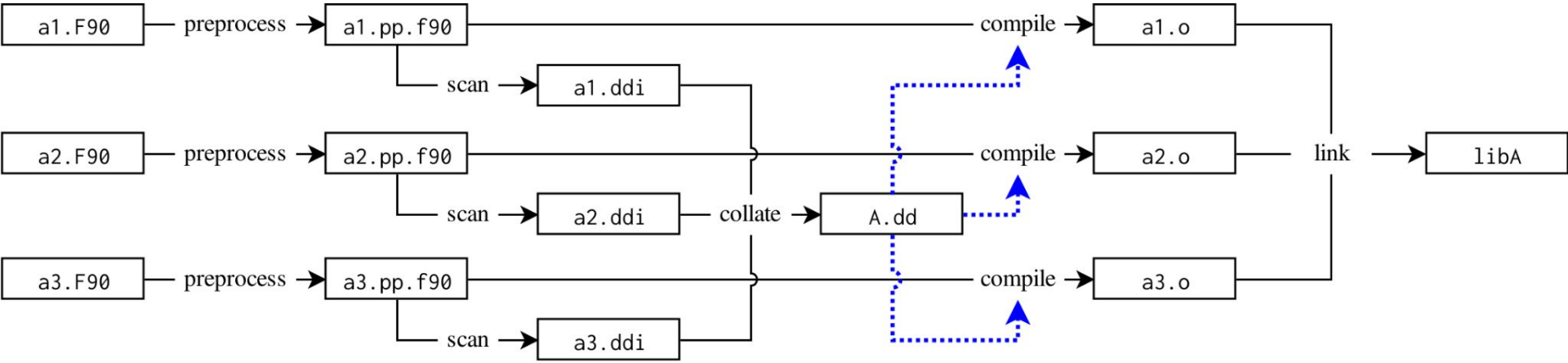
.ddi - record provides and requires

.dd - dynamic dependency file

.o - compiled object file

Kitware

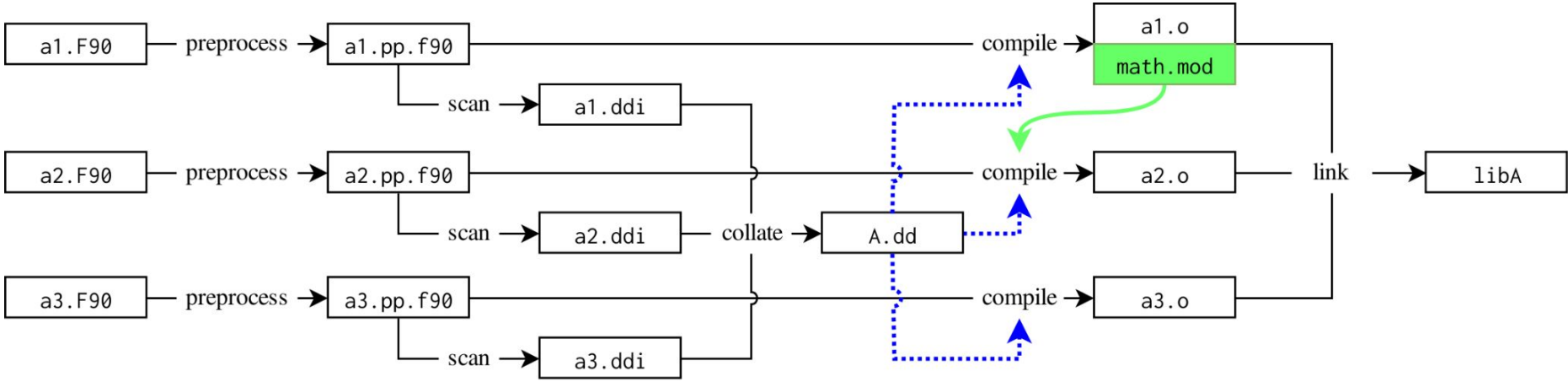
Build graph for single target



interdependencies between compilation outputs will be discovered during the build.

ninja was modified to handle dynamic dependencies.

Build graph for single target

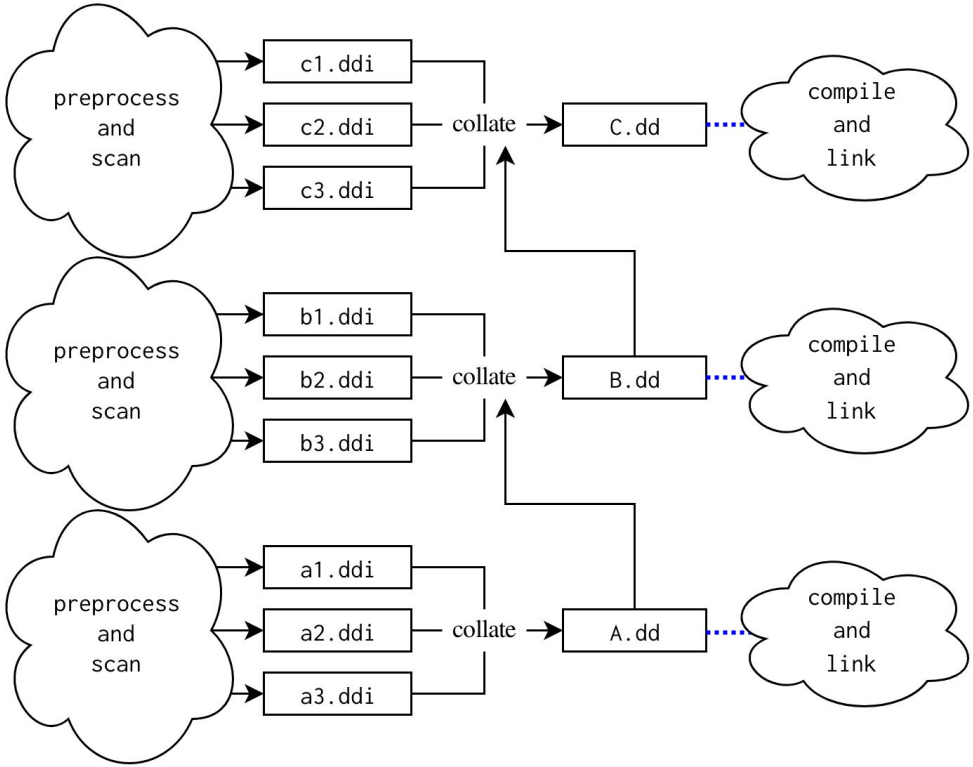


When `A.dd` is up-to-date and read, `ninja` updates its build graph and it learns:

- `a3.F90` and `a1.F90` may be compiled in parallel
- `a2.F90` must wait for the compilation of `a1.F90` for `math.mod` to created.
- If `a1.F90` changes, `a2.F90` may need recompiled, but `a3.F90` will never need to be.
- If `a1.F90` is modified and recompiled, if `math.mod` is not changed, then `a2.F90` is still considered up-to-date and not recompiled unnecessarily.
- If `a3.F90` is changed to gain a dependency on `math.mod`, the requirement on the `A.dd` generation means that the new dependency will be discovered and that the `a1.F90` must be checked for changes as well.

Kitware

Build graph multiple targets



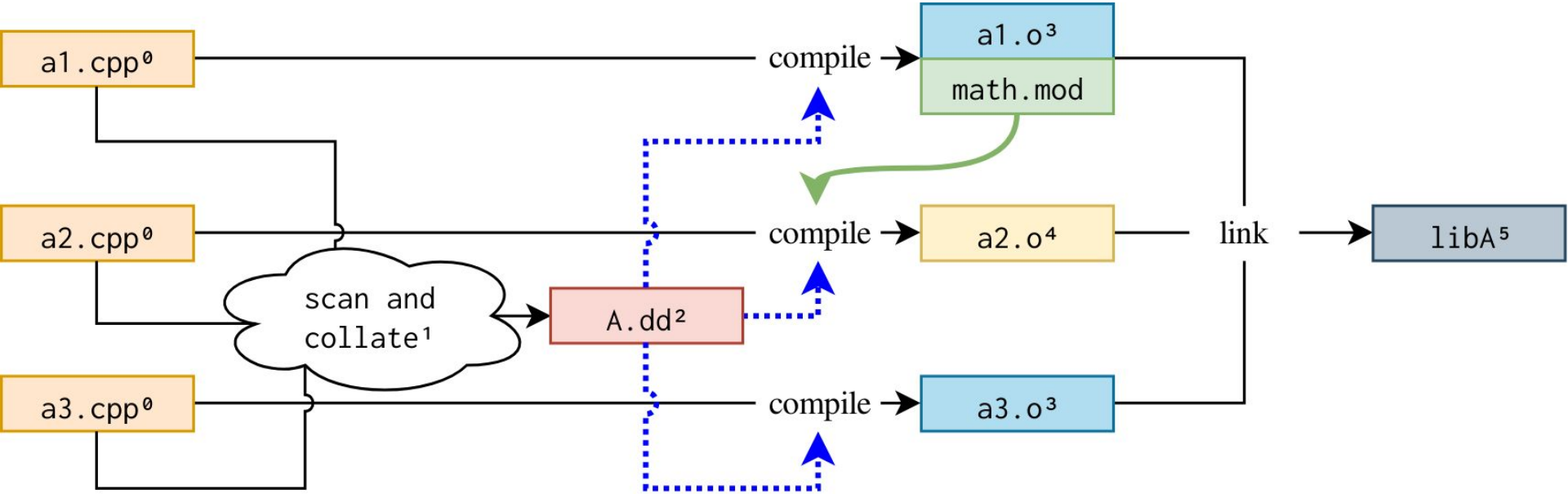
The `.dd` files for direct dependent targets are additional inputs to the target's `.dd` file. This allows modules contained within a source file in library `B` to be used in a source file in library `C` while still getting dependencies correct.

Kitware

How does CMake do this

- A Fortran parser based off of makedepf90
- Patches made to ninja build tool now upstreamed
- The dynamic dependency collator inside CMake

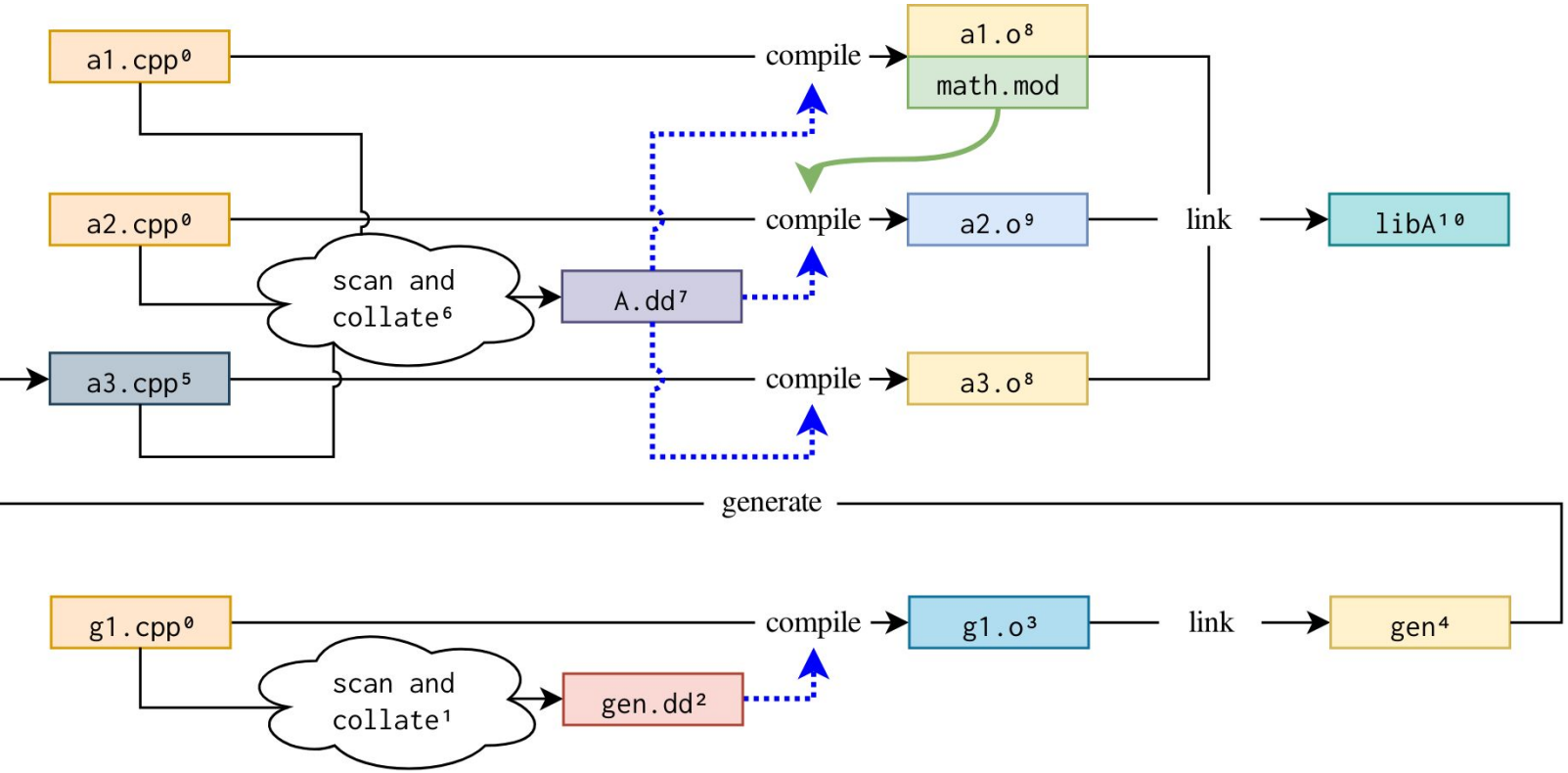
Now for C++ (C++ now?)



With compiler help C++'s build graph will be able to skip the separate preprocess rule within the build and instead use a single scan rule.

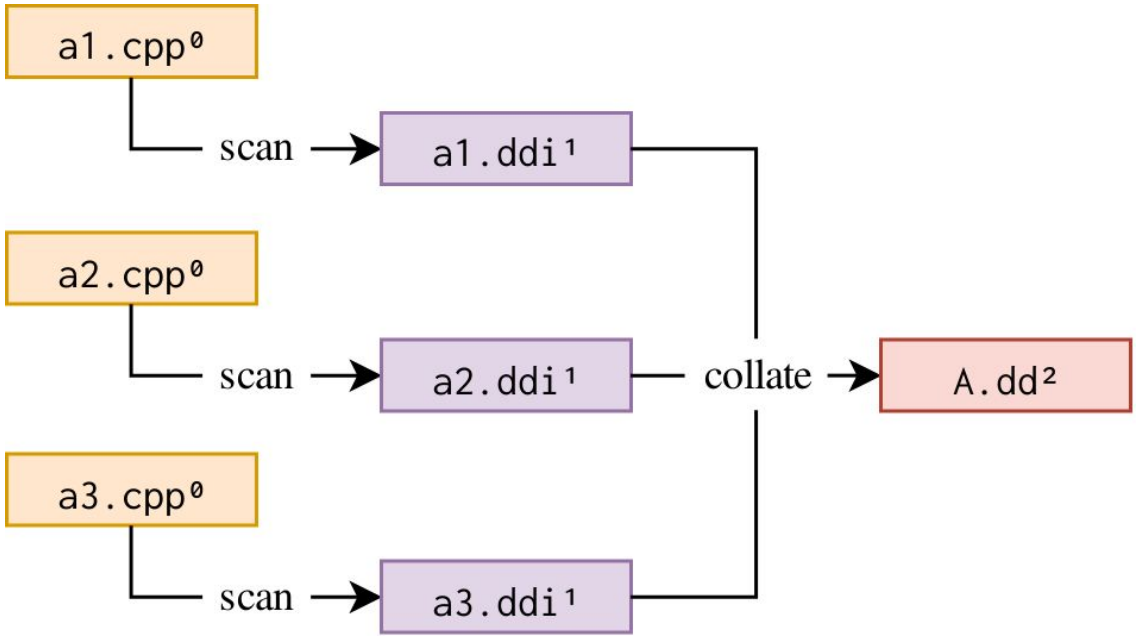
Kitware

Compiled code generators



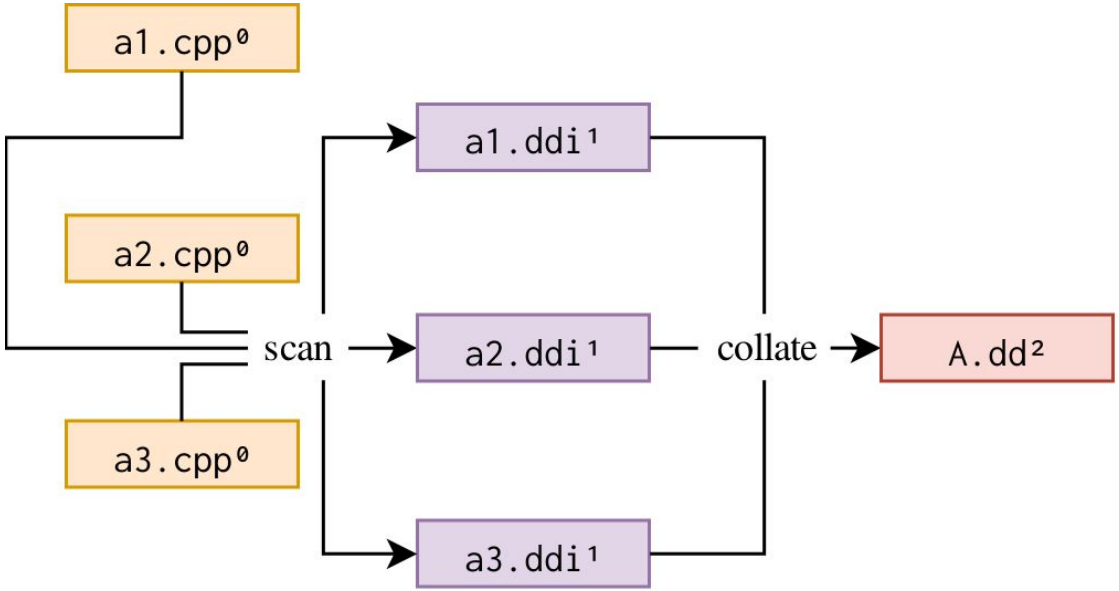
Kitware

option 1 - Scan sources independently then collate

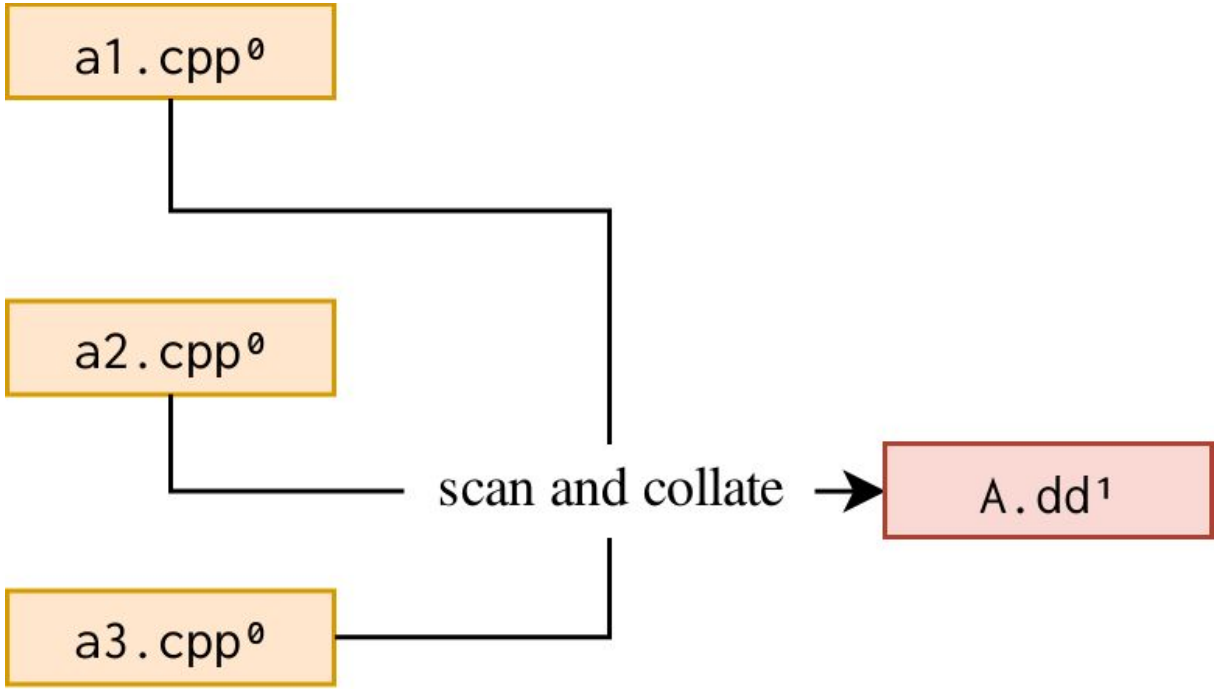


Kitware

option 2 - Scan sources all-at-once then collate



Options 3 - Scan sources and collate all-at-once



Reference Papers

How CMake supports Fortran Modules:
https://mathstuf.fedorapeople.org/fortran-modules/fortran-modules.html

Proposed file format:
https://mathstuf.fedorapeople.org/dep-format/dep-format-r3.html

Build systems need compiler support for C++ Modules!

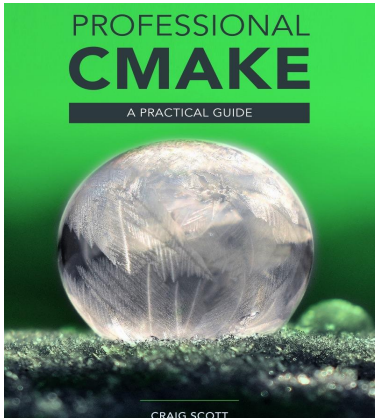
Please do not make CMake have to include a C++ parser!

Kitware

Learning CMake

For help or more information see:

- ~~Copy some CMake code from 2010..~~
- Watch a talk on “Modern CMake”
- Professional CMake by Craig Scott
- Discourse Forum
  - https://discourse.cmake.org
- Documentation
  - https://www.cmake.org/cmake/help/latest/
- Tutorial
  - https://cmake.org/cmake/help/latest/guide/tutorial/index.html



Kitware

CMake Guides

cmake.org/cmake/help/latest/index.html

- cmake-toolchains(7)
- cmake-variables(7)
- cpack-generators(7)

Guides

- CMake Tutorial
- User Interaction Guide
- Using Dependencies Guide
- Importing and Exporting Guide
- IDE Integration Guide

Kitware

Testing Doc Code

```
Specify the C++ Standard
------------------------

Next let's add some C++11 features to our project by replacing ``atof`` with
``std::stod`` in ``tutorial.cxx``.  At the same time, remove
``#include <cstdlib>``.

.. literalinclude:: Step2/tutorial.cxx
  :language: c++
  :start-after: // convert input to double
  :end-before: // calculate square root

We will need to explicitly state in the CMake
correct flags. The easiest way to enable suppo
in CMake is by using the :variable:`CMAKE_CXX_
tutorial, set the :variable:`CMAKE_CXX_STANDAR
``CMakeLists.txt`` file to 11 and :variable:`C
True. Make sure to add the ``CMAKE_CXX_STANDAR
to ``add_executable``.

.. literalinclude:: Step2/CMakeLists.txt
  :language: cmake
  :end-before: # configure a header file to pa
```

```
#include <string>

#include "TutorialConfig.h"

int main(int argc, char* argv[])
{
  if (argc < 2) {
    // report version
    std::cout << argv[0] << " Version " << Tutorial_VERSION_MAJOR << "."
              << Tutorial_VERSION_MINOR << std::endl;
    std::cout << "Usage: " << argv[0] << " number" << std::endl;
    return 1;
  }

  // convert input to double
  const double inputValue = std::stod(argv[1]);

  // calculate square root
  const double outputValue = sqrt(inputValue);
  std::cout << "The square root of " << inputValue << " is " << outputValue
            << std::endl;
  return 0;
}
```

Specify the C++ Standard

Next let's add some C++11 features to our project by replacing `atof` with `std::stod` in `tutorial.cxx`. At the same time, remove `#include <cstdlib>`.

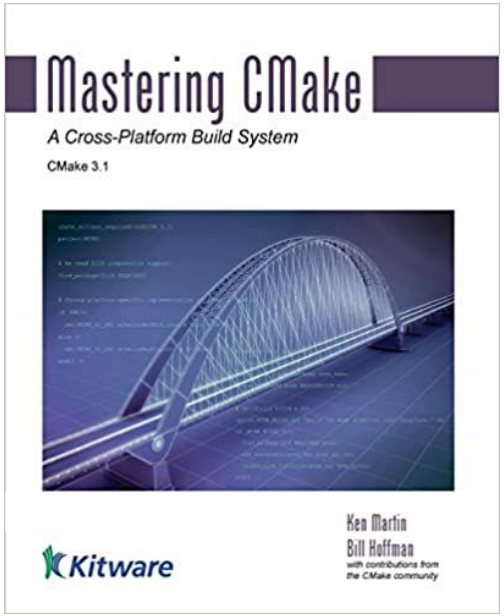
```
const double inputValue = std::stod(argv[1]);
```

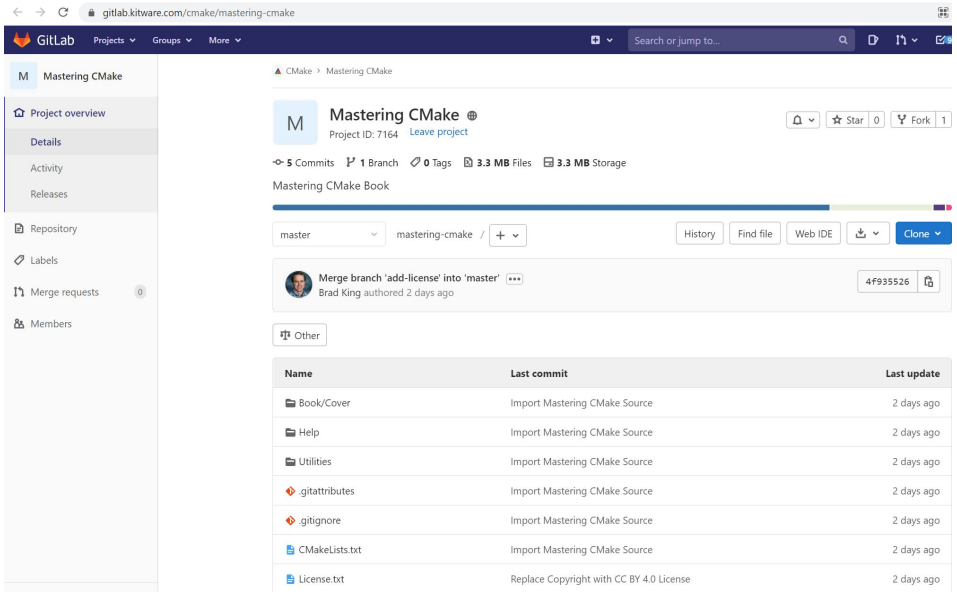
We will need to explicitly state in the CMake code that it should use the correct flags. The easiest way to enable support for a specific C++ standard in CMake is by using the `CMAKE_CXX_STANDARD` variable. For this tutorial, set the `CMAKE_CXX_STANDARD` variable in the `CMakeLists.txt` file to 11 and `CMAKE_CXX_STANDARD_REQUIRED` to True. Make sure to add the `CMAKE_CXX_STANDARD` declarations above the call to `add_executable`.

```
cmake_minimum_required(VERSION 3.10)

# set the project name and version
project(Tutorial VERSION 1.0)

# specify the C++ standard
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)
```

Kitware

Mastering CMake Now Open Source!





Kitware

Putting it all together

Most software is not an island and depends on other libraries

CMake Find Modules

Our library “trunk” needs PNG

```
find_package(PNG REQUIRED)
add_library(trunk SHARED trunk.cxx)
```

No different than if we built PNG as part of the project:

```
target_link_libraries(trunk PRIVATE PNG::PNG)
```

- CMake provides 150 find modules
  - cmake --help-module-list
  - https://cmake.org/cmake/help/latest/manual/cmake-modules.7.html

Exporting Targets

- Install rules can generate imported targets

```
add_library(parasite STATIC eat_leaf.cxx)
install(TARGETS parasite root trunk leaf
        EXPORT tree-targets)
install(EXPORT tree-targets
        DESTINATION lib/cmake/tree)
```

- Installs library and target import rules
  - `<prefix>/lib/libparasite.a`
  - `<prefix>/lib/cmake/tree/tree-targets.cmake`

Conan can create cmake config.cmake

CMakeDeps

Available since: 1.33.0

The `CMakeDeps` helper will generate one **xxxx-config.cmake** file per dependency, together with other necessary *.cmake* files like version, flags and directory data or configuration. It can be used like:

```
from conans import ConanFile

class App(ConanFile):
    settings = "os", "arch", "compiler", "build_type"
    requires = "hello/0.1"
    generators = "CMakeDeps"
```

Kitware

External Projects (build time)

```
ExternalProject_Add(foo
  GIT_REPOSITORY    git@github.com:FooCo/Foo.git
  GIT_TAG           origin/release/1.2.3
)

ExternalProject_Add(bar
  GIT_REPOSITORY    git@github.com:BarCo/Bar.git
  GIT_TAG           origin/release/2.3.4
  DEPENDS foo
)
```

FetchContent (Configure Time)

```
FetchContent_Declare(catch
  GIT_REPOSITORY https://github.com/catchorg/Catch2.git
  GIT_TAG        v2.2.1
)
FetchContent_GetProperties(catch)
if(NOT catch_POPULATED)
  FetchContent_Populate(catch)
  add_subdirectory(${catch_SOURCE_DIR} ${catch_BINARY_DIR})
endif()
```

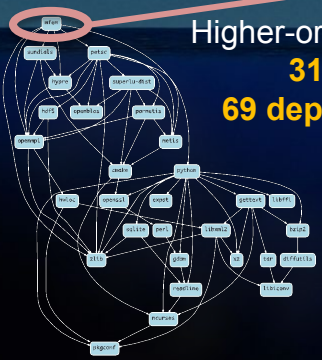
Kitware

Package Managers

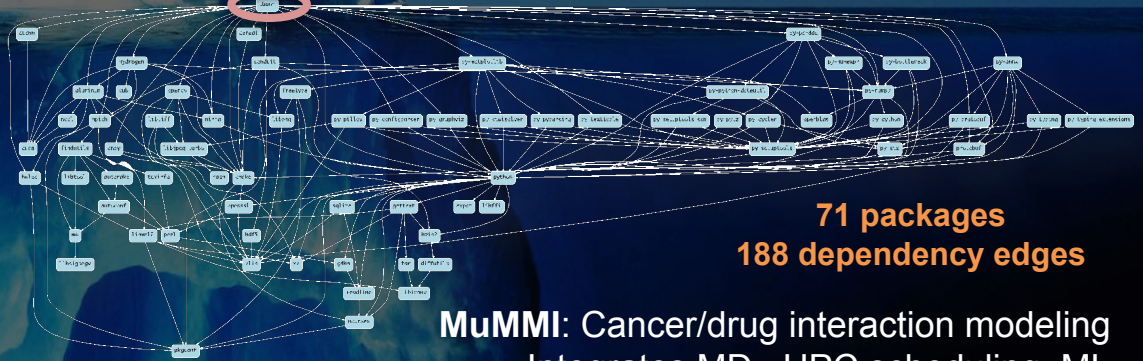
- Although CMake can build them all, CMake's ExternalProject and FetchContent, can only go so far.
- Complex software using many packages and languages are best handled with package managers
- Conan, vcpkg, and Spack are all good options for managing large C++ projects

Kitware

HPC simulations rely on icebergs of dependency libraries



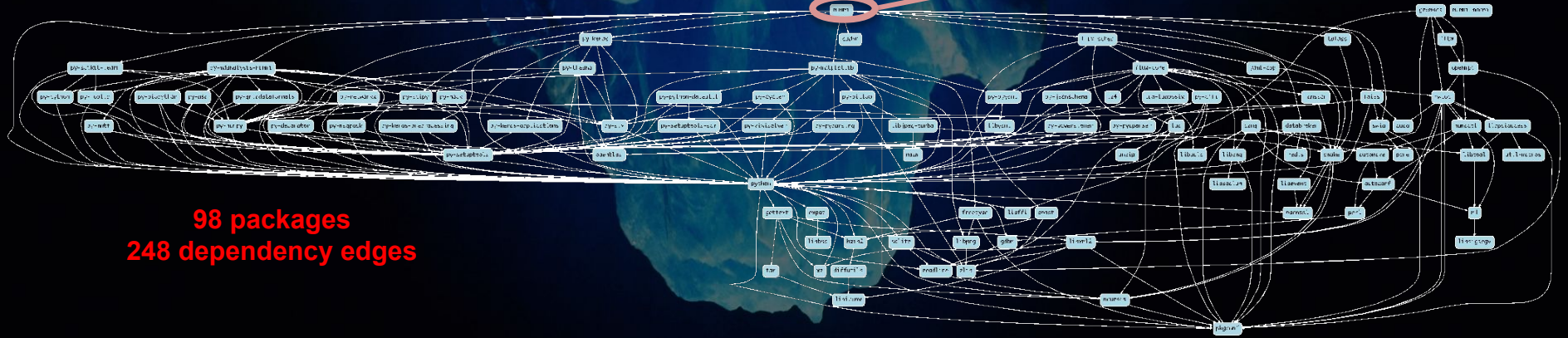
MFEM:
Higher-order finite elements
31 packages,
69 dependency edges



LBANN: Neural Nets for HPC

71 packages
188 dependency edges

MuMMI: Cancer/drug interaction modeling
Integrates MD , HPC scheduling, ML



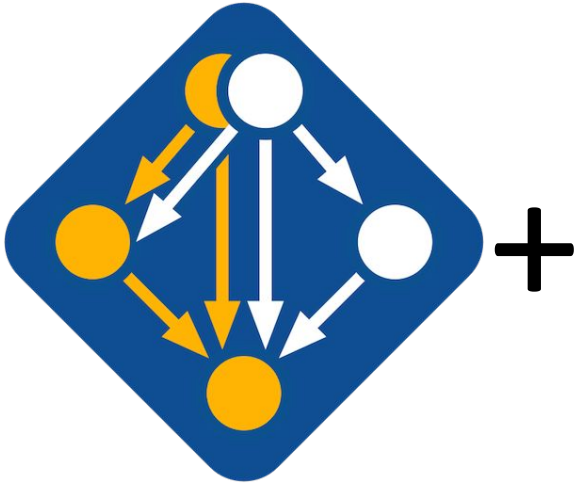
98 packages
248 dependency edges

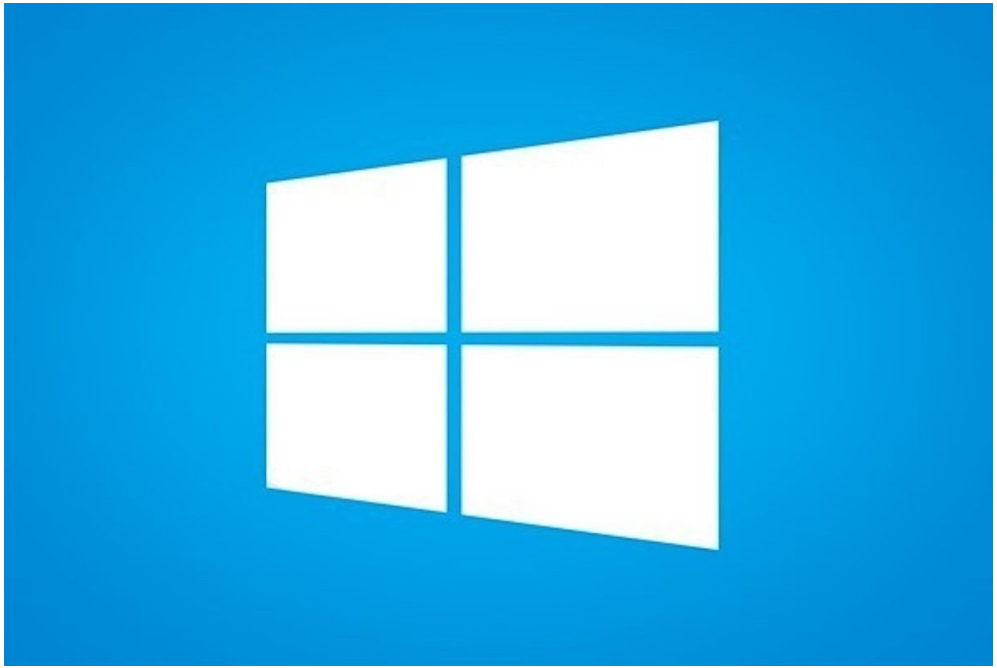
Spack helped streamline development environment for an LLNL C++ code team using PyTorch

- **Before Spack**
  - Integrating PyTorch with a C++ code for ppc64le and MPI was very painful
  - Everybody built their own python/pytorch from scratch
  - People wrote scripts and passed them around
    - Scripts slowly accumulated modifications and magic
  - **Days were spent trying to debug build differences**

- **After spack**
  - Versioned reproducible spack enviroments in a repo (spack.yaml / spack.lock)
  - Standard environments in a shared team directory
  - **Any team member can get a customizable working environment in ~20 minutes.**
    - Change python version, change pytorch version, etc.
    - Reuse prior builds via Spack buildcaches

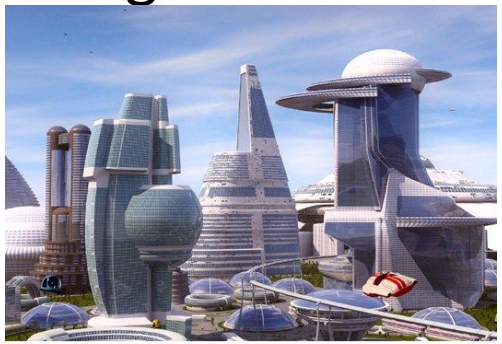
Kitware





Future I would like to see for C++/CMake

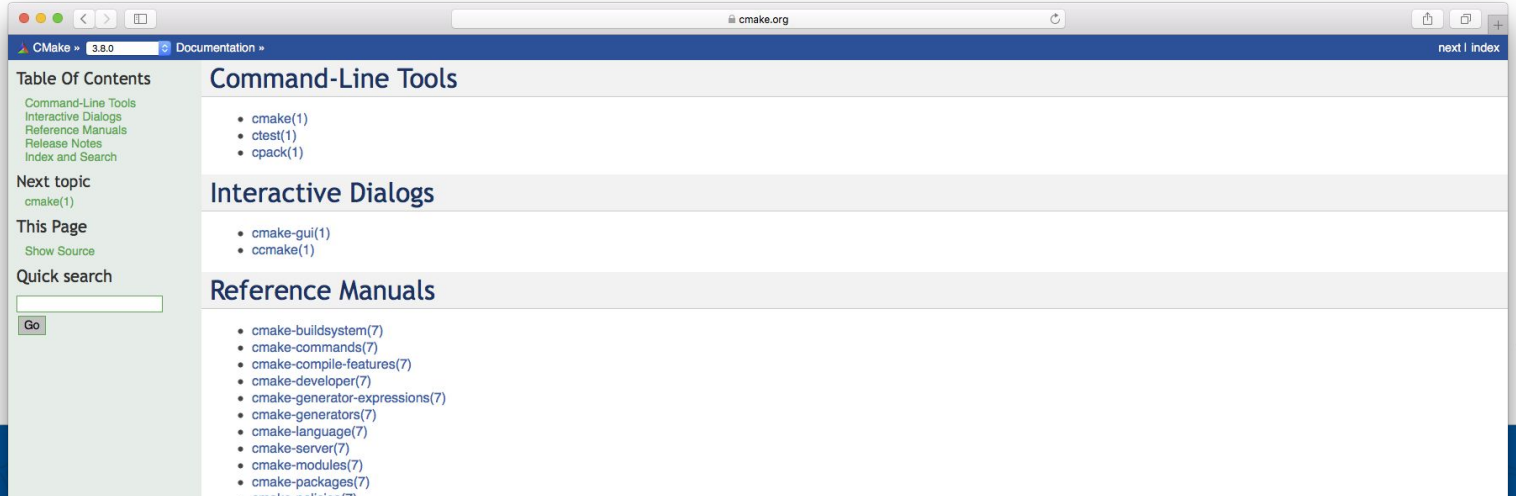
- All C++ compilers provide build system interfaces to collect C++20 module dependency information.
- A cross platform standard for the information found in cmake config files



Kitware

Thank You

- bill.hoffman@kitware.com
- Read “how to write a CMake buildsystem”
  - https://cmake.org/cmake/help/latest/manual/cmake-buildsystem.7.html/Explore the CMake documentation
- Explore the CMake documentation



Kitware