GIT, CMAKE, CONAN

HOW TO SHIP AND REUSE OUR C++ PROJECTS?

Mateusz Pusz
May 8, 2018

ACCU 2018 TALK SUBMISSION FEEDBACK

Maybe prefer "Mercurial, Meson, Conan"?

ACCU 2018 TALK SUBMISSION FEEDBACK

Maybe prefer "Mercurial, Meson, Conan"?

Please do not differentiate our development environment even more

THE MOST COMMON C++ TOOLSET

VERSION CONTROL SYSTEM	git
BUILDING	CMake
PACKAGE MANAGEMENT	None

THE MOST COMMON C++ TOOLSET

VERSION CONTROL SYSTEM	git
BUILDING	CMake
PACKAGE MANAGEMENT	None

- **Conan** is a strong contender to become *the Package Manager for C++*

TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

1. *`external` or `3rdparty` subdirs* with external projects' source code + CMake **`add_subdirectory()`**

TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

1. *`external` or `3rdparty` subdirs* with external projects' source code + CMake **`add_subdirectory()`**
2. External source code as *git submodules* + CMake **`add_subdirectory()`**

TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

1. *`external` or `3rdparty` subdirs* with external projects' source code + CMake **`add_subdirectory()`**
2. External source code as *git submodules* + CMake **`add_subdirectory()`**
3. CMake *`ExternalProject_Add()`* + CMake **`add_subdirectory()`**

TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

1. *`external` or `3rdparty` subdirs* with external projects' source code + CMake **`add_subdirectory()`**
2. External source code as *git submodules* + CMake **`add_subdirectory()`**
3. CMake *`ExternalProject_Add()`* + CMake **`add_subdirectory()`**
4. *Downloading and installing* each dependency + CMake **`find_package()`**

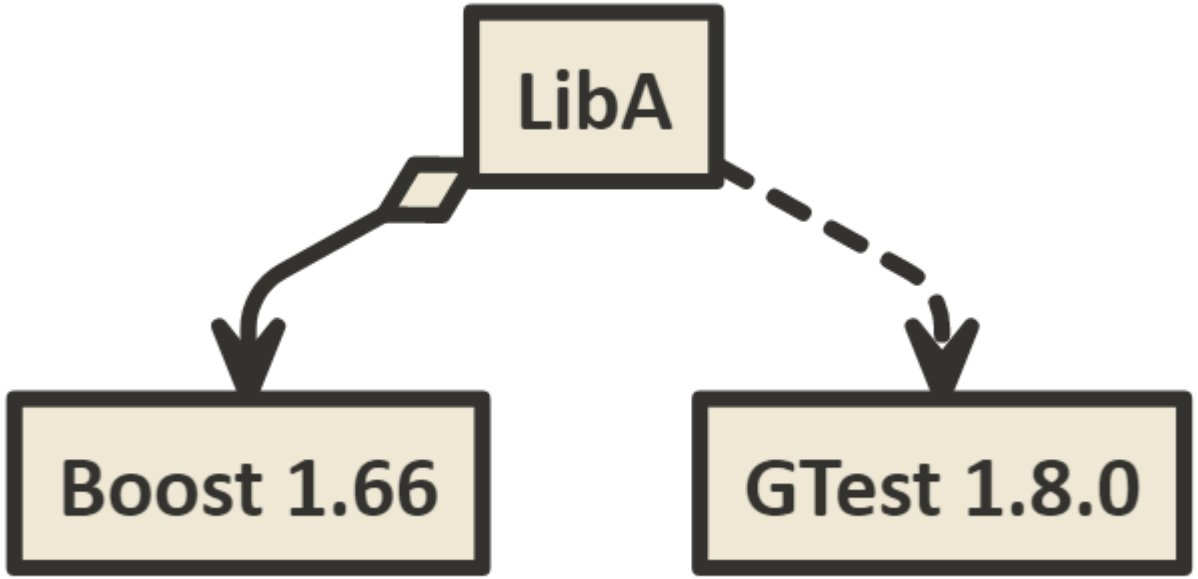
TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

1. *`external` or `3rdparty` subdirs* with external projects' source code + CMake **`add_subdirectory()`**
2. External source code as *git submodules* + CMake **`add_subdirectory()`**
3. CMake *`ExternalProject_Add()`* + CMake **`add_subdirectory()`**
4. *Downloading and installing* each dependency + CMake **`find_package()`**
5. Usage of *other languages' toolsets* (i.e. maven, NuGet, ...)

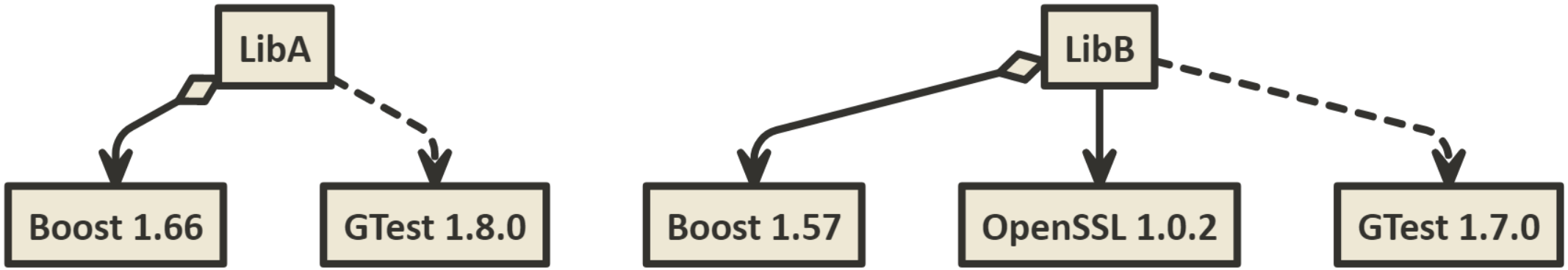
TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

1. *`external` or `3rdparty` subdirs* with external projects' source code + CMake **`add_subdirectory()`**
2. External source code as *git submodules* + CMake **`add_subdirectory()`**
3. CMake *`ExternalProject_Add()`* + CMake **`add_subdirectory()`**
4. *Downloading and installing* each dependency + CMake **`find_package()`**
5. Usage of *other languages' toolsets* (i.e. maven, NuGet, ...)
6. *Dedicated C++ package managers* (i.e. Conan)

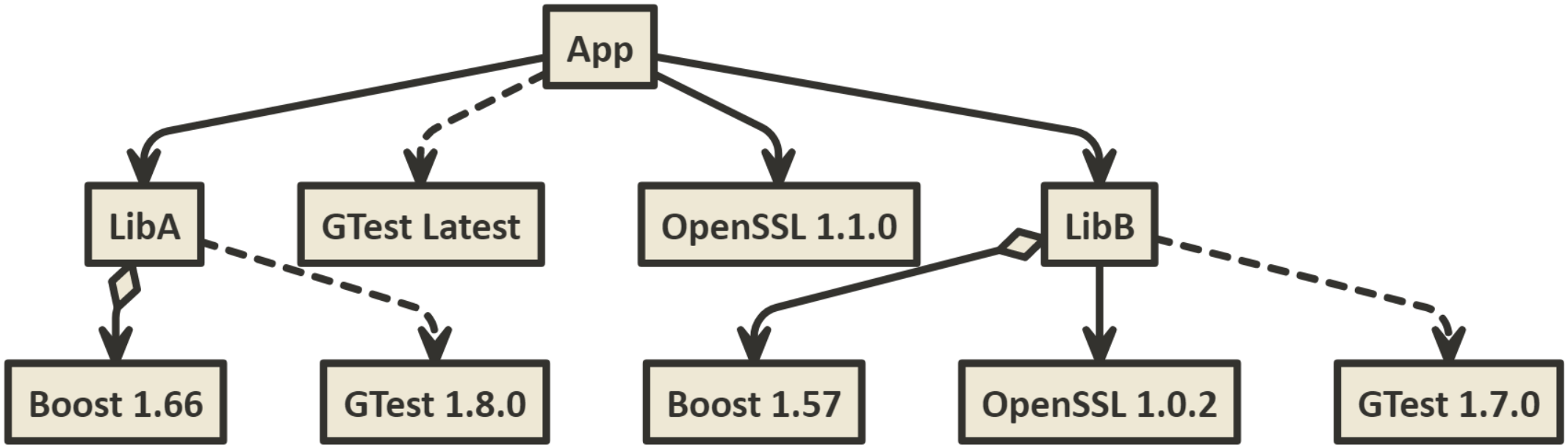
`add_subdirectory()` FOR DEPS?



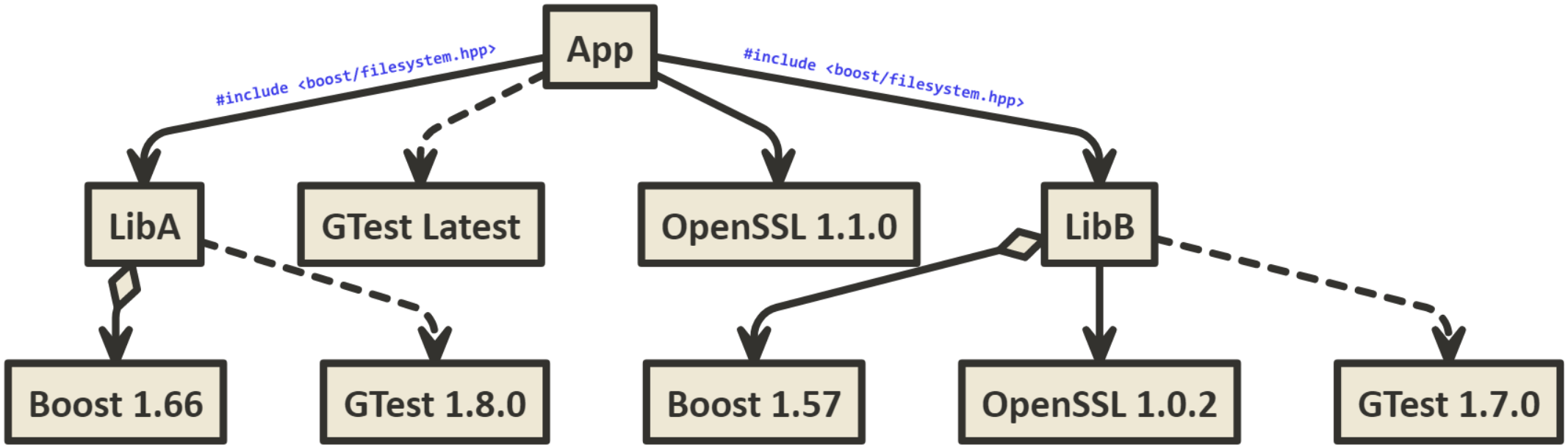
add_subdirectory() FOR DEPS?



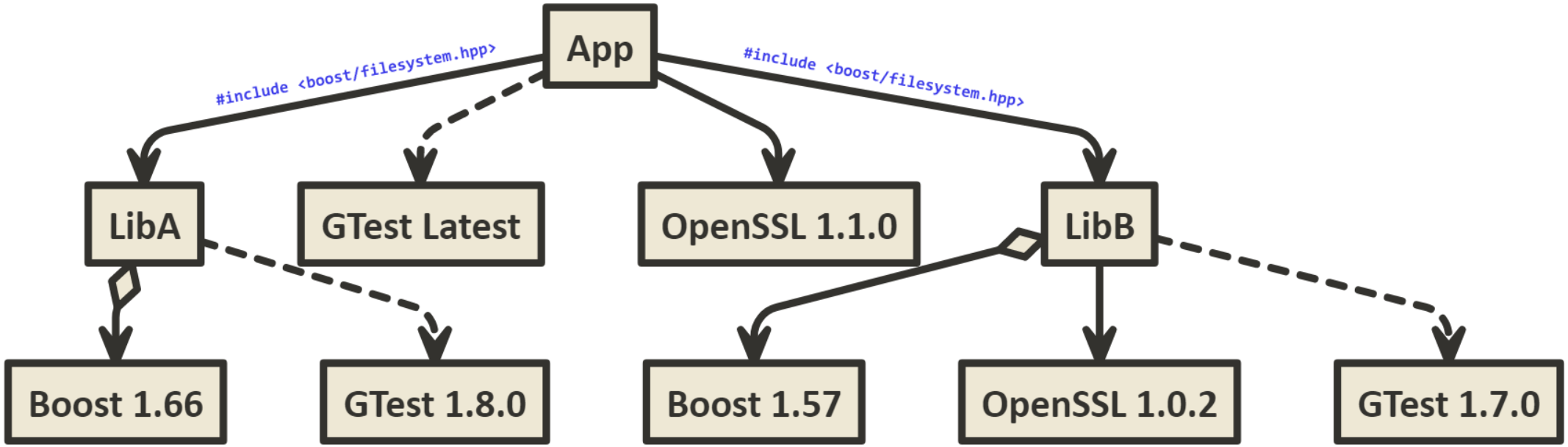
add_subdirectory() FOR DEPS?



add_subdirectory() FOR DEPS? PLEASE DON'T!



add_subdirectory() FOR DEPS? PLEASE DON'T!



Handling dependencies as subdirectories does not scale!

CMake

Not a build system!

Cross-platform C++ build generator

TYPICAL CMAKE WORKFLOW

GCC

```
cmake -DCMAKE_BULD_TYPE=Release -DCMAKE_INSTALL_PREFIX=~/.local ..
cmake --build .
ctest -VV
cmake --build . --target install
```

VISUAL STUDIO

```
cmake -G "Visual Studio 15 2017 Win64" -DCMAKE_INSTALL_PREFIX=~/.local ..
cmake --build . --config Release
ctest -VV -C Release
cmake --build . --target install --config Release
```

MODERN CMake

C++Now 2017: Daniel Pfeifer “Effective CMake"

Build flags don't scale!

- Every change in public flags *has to be propagated* upwards
- Not possible to maintain flags on a large scale
- Different targets may require *conflicting flag values*

MODERN CMake

Build flags don't scale!

- Every change in public flags *has to be propagated* upwards
- Not possible to maintain flags on a large scale
- Different targets may require *conflicting flag values*

Modern CMake is about Targets and Properties

CMake **TARGET TYPES**

EXECUTABLES	`add_executable`
SHARED LIBRARIES	`add_library(SHARED)`
STATIC LIBRARIES	`add_library(STATIC)`
OBJECT LIBRARIES	`add_library(OBJECT)`
INTERFACE LIBRARIES	`add_library(INTERFACE)`
ALIAS LIBRARIES	`add_library(ALIAS)`
IMPORTED LIBRARIES	`add_library(IMPORTED [GLOBAL])`

CMake **TARGET TYPES**

EXECUTABLES	`add_executable`
SHARED LIBRARIES	`add_library(SHARED)`
STATIC LIBRARIES	`add_library(STATIC)`
OBJECT LIBRARIES	`add_library(OBJECT)`
INTERFACE LIBRARIES	`add_library(INTERFACE)`
ALIAS LIBRARIES	`add_library(ALIAS)`
IMPORTED LIBRARIES	`add_library(IMPORTED [GLOBAL])`

- *If no type is given explicitly* to **`add_library()`** the type is ***`STATIC`*** *or* ***`SHARED`*** *based on* whether the current value of the variable ***`BUILD_SHARED_LIBS`*** *is* ***`ON`***

MODERN CMake: MODULAR DESIGN

- Available since version 2.8.12 (Oct 2013)
- Specified by **`target_xxx()`** commands

```
target_link_libraries(<target>
                      <PRIVATE|PUBLIC|INTERFACE> <item>...
                     [<PRIVATE|PUBLIC|INTERFACE> <item>...]...)
```

MODERN CMake: MODULAR DESIGN

- Available since version 2.8.12 (Oct 2013)
- Specified by **`target_xxx()`** commands

```
target_link_libraries(<target>
                      <PRIVATE|PUBLIC|INTERFACE> <item>...
                     [<PRIVATE|PUBLIC|INTERFACE> <item>...]...)
```

	NEEDED BY ME	NOT NEEDED BY ME
NEEDED BY DEPENDERS	PUBLIC	INTERFACE
NOT NEEDED BY DEPENDERS	PRIVATE	:-)

MODERN CMake: MODULAR DESIGN

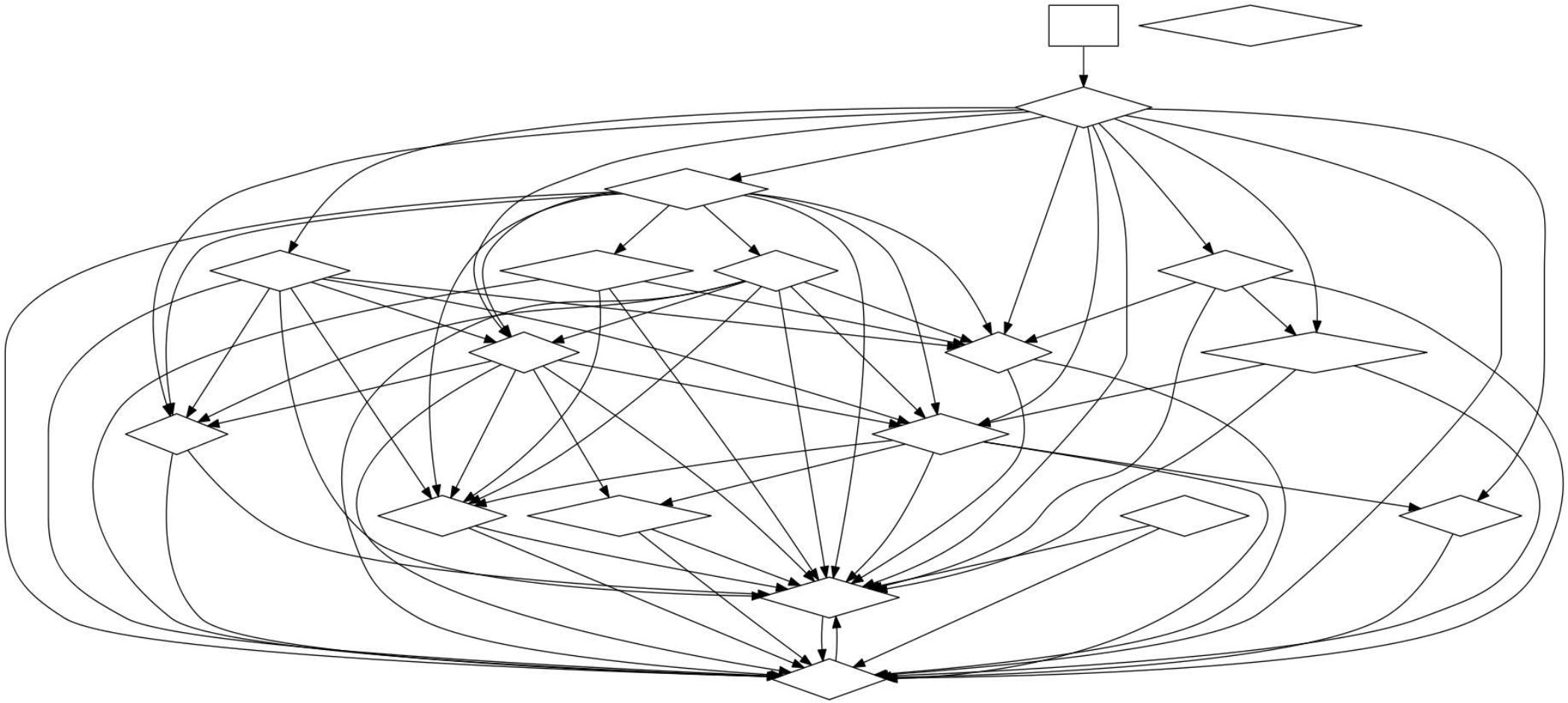
- Available since version 2.8.12 (Oct 2013)
- Specified by **`target_xxx()`** commands

```
target_link_libraries(<target>
                      <PRIVATE|PUBLIC|INTERFACE> <item>...
                     [<PRIVATE|PUBLIC|INTERFACE> <item>...]...)
```

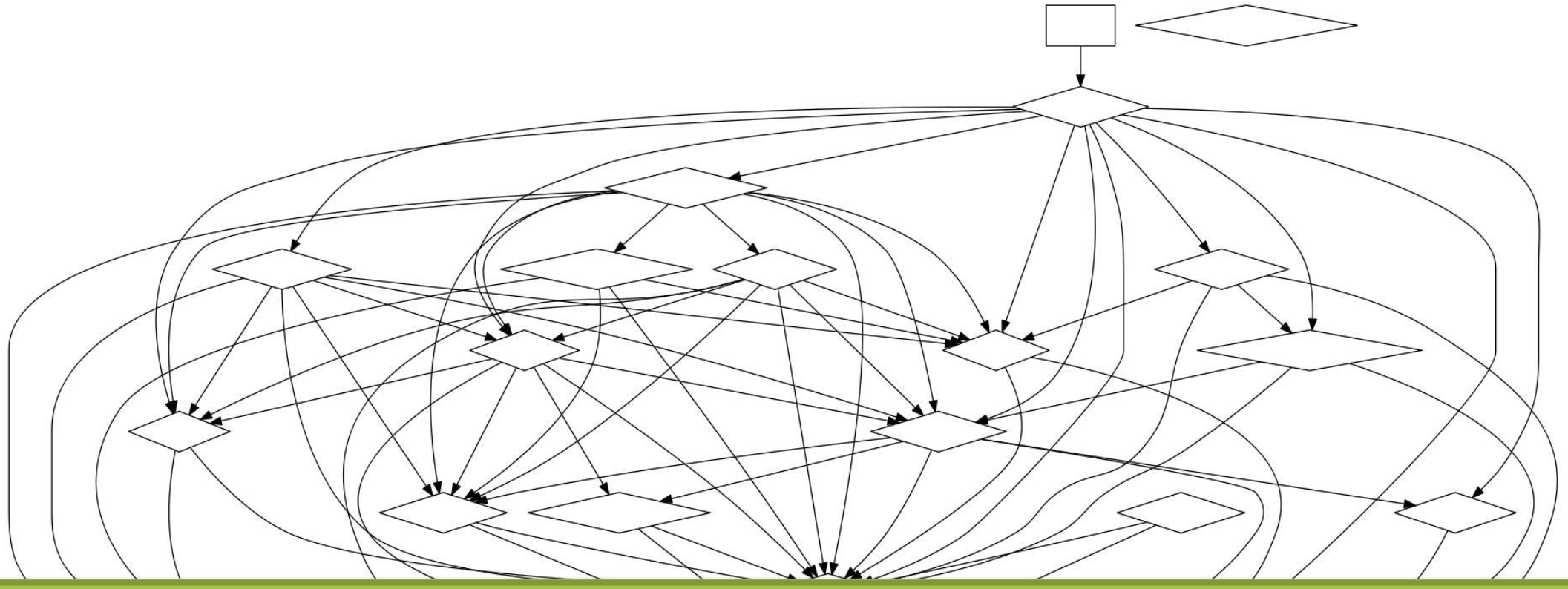
	NEEDED BY ME	NOT NEEDED BY ME
NEEDED BY DEPENDERS	PUBLIC	INTERFACE
NOT NEEDED BY DEPENDERS	PRIVATE	:-)

INTERFACE and **PUBLIC** dependencies are **transitive** while **PRIVATE** are not

PHYSICAL DESIGN: OLD CMAKE STYLE



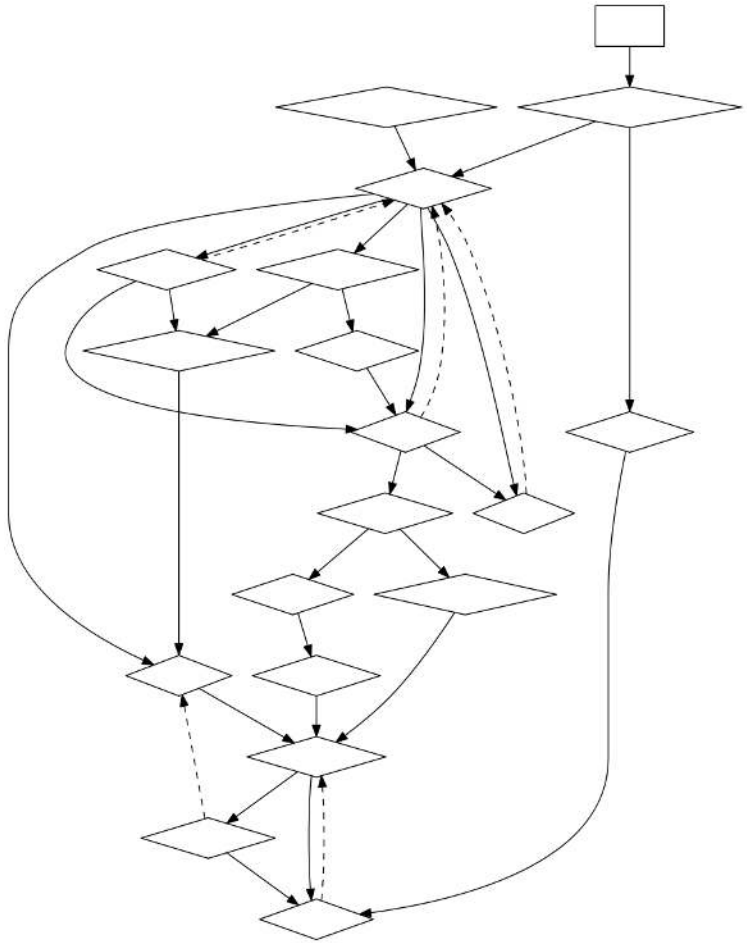
PHYSICAL DESIGN: OLD CMAKE STYLE



Linking diagram only. Not useful to reveal real design problems.

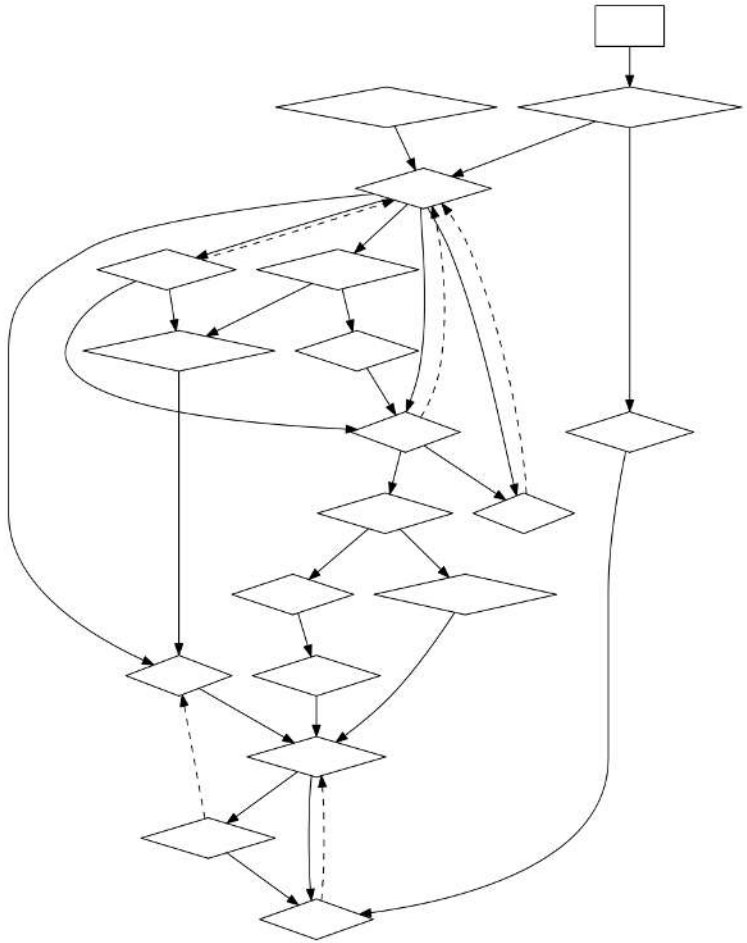
PHYSICAL DESIGN: MODERN CMAKE

- More than a linking diagram
- *Logical dependencies* included
- *PRIVATE* and *PUBLIC* dependencies make difference



PHYSICAL DESIGN: MODERN CMAKE

- More than a linking diagram
- *Logical dependencies* included
- *PRIVATE* and *PUBLIC* dependencies make difference
- Now we see real design problems...



NO CYCLIC PHYSICAL DEPENDENCIES!

http://cyber-dojo.org C++ Modules and Large-Scale Development - John Lakos [ACCU 2018]

ALIAS TARGETS

```
add_library(MyLibrary lib_source.cpp)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

ALIAS TARGETS

```
add_library(MyLibrary lib_source.cpp)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

```
# find_package(MyLibrary CONFIG REQUIRED)

target_link_libraries(MyLibraryTest
    PUBLIC
        MyCompany::MyLibrary
        GTest::Main
)
```

- Unifies with **`find_package()`** target naming
- **`::`** has to be followed with Target name (prevents typos)

GENERATOR EXPRESSIONS

```
target_compile_definitions(foo
    PRIVATE
        "VERBOSITY=$<IF:$<BOOL:${VERBOSE}>,30,10>")
```

- Use the **$<>** syntax
- *Not evaluated* by the command interpreter
- **Evaluated during build system generation**

GENERATOR EXPRESSIONS

BAD

```
add_executable(hello main.cpp)
if(CMAKE_BUILD_TYPE STREQUAL DEBUG)
    target_sources(hello PRIVATE helper_debug.cpp)
else()
    target_sources(hello PRIVATE helper_release.cpp)
endif()
```

GENERATOR EXPRESSIONS

BAD

```
add_executable(hello main.cpp)
if(CMAKE_BUILD_TYPE STREQUAL DEBUG)
    target_sources(hello PRIVATE helper_debug.cpp)
else()
    target_sources(hello PRIVATE helper_release.cpp)
endif()
```

GOOD

```
add_executable(hello main.cpp
    $<IF:$<CONFIG:Debug>:helper_debug.cpp,helper_release.cpp>)
```

GENERATOR EXPRESSIONS

BAD

```
add_executable(hello main.cpp)
if(CMAKE_BUILD_TYPE STREQUAL DEBUG)
    target_sources(hello PRIVATE helper_debug.cpp)
else()
    target_sources(hello PRIVATE helper_release.cpp)
endif()
```

GOOD

```
add_executable(hello main.cpp
    $<IF:$<CONFIG:Debug>:helper_debug.cpp,helper_release.cpp>)
```

Never use **CMAKE_BUILD_TYPE** in **if()**

GENERATOR EXPRESSIONS

The library interface may change during installation. Use **`BUILD_INTERFACE`** and **`INSTALL_INTERFACE`** generator expression filters.

GENERATOR EXPRESSIONS

The library interface may change during installation. Use **BUILD_INTERFACE** and **INSTALL_INTERFACE** generator expression filters.

```
target_include_directories(Foo PUBLIC
        $<BUILD_INTERFACE:${Foo_BINARY_DIR}/include>
        $<BUILD_INTERFACE:${Foo_SOURCE_DIR}/include>
        $<INSTALL_INTERFACE:include>)
```

MODERN LIBRARY EXAMPLE

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)

# dependencies
find_package(Foo 1.0 REQUIRED)

# library definition
add_library(MyLibrary lib_source.cpp)
target_compile_features(MyLibrary PUBLIC cxx_std_17)
target_include_directories(MyLibrary PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>
)
target_link_libraries(MyLibrary PRIVATE Foo::Foo)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

MODERN LIBRARY EXAMPLE

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)

# dependencies
find_package(Foo 1.0 REQUIRED)

# library definition
add_library(MyLibrary lib_source.cpp)
target_compile_features(MyLibrary PUBLIC cxx_std_17)
target_include_directories(MyLibrary PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>
)
target_link_libraries(MyLibrary PRIVATE Foo::Foo)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

MODERN LIBRARY EXAMPLE

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)

# dependencies
find_package(Foo 1.0 REQUIRED)

# library definition
add_library(MyLibrary lib_source.cpp)
target_compile_features(MyLibrary PUBLIC cxx_std_17)
target_include_directories(MyLibrary PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>
)
target_link_libraries(MyLibrary PRIVATE Foo::Foo)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

MODERN LIBRARY EXAMPLE

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)

# dependencies
find_package(Foo 1.0 REQUIRED)

# library definition
add_library(MyLibrary lib_source.cpp)
target_compile_features(MyLibrary PUBLIC cxx_std_17)
target_include_directories(MyLibrary PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>
)
target_link_libraries(MyLibrary PRIVATE Foo::Foo)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

MODERN LIBRARY EXAMPLE

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)

# dependencies
find_package(Foo 1.0 REQUIRED)

# library definition
add_library(MyLibrary lib_source.cpp)
target_compile_features(MyLibrary PUBLIC cxx_std_17)
target_include_directories(MyLibrary PUBLIC
    $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
    $<INSTALL_INTERFACE:include>
)
target_link_libraries(MyLibrary PRIVATE Foo::Foo)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

Avoid *custom* variables in the arguments of project commands

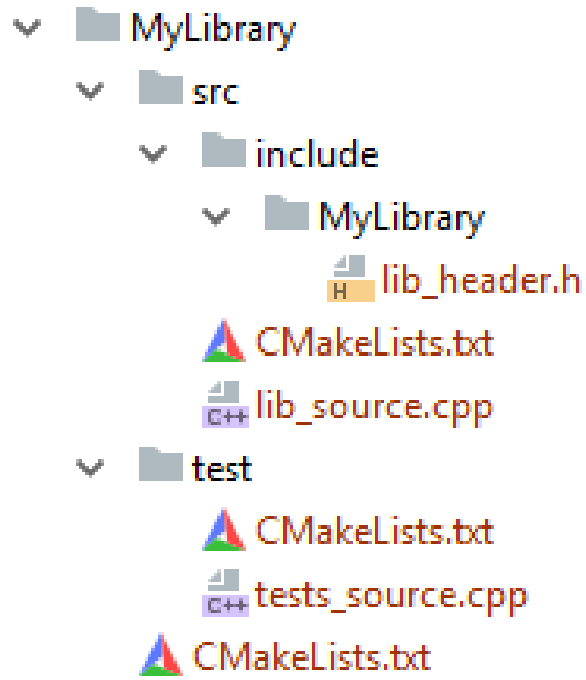
MODERN LIBRARY USAGE

```
cmake_minimum_required(VERSION 3.8)
project(MyLibraryTests)

# dependencies
enable_testing()
find_package(GTest MODULE REQUIRED)
if(NOT TARGET MyCompany::MyLibrary)
    find_package(MyLibrary CONFIG REQUIRED)
endif()

# target definition
add_executable(MyLibraryTests tests_source.cpp)
target_link_libraries(MyLibraryTests
    PRIVATE
        MyCompany::MyLibrary
        GTest::Main
)
add_test(NAME MyLibrary.UnitTests
    COMMAND MyLibraryTests
)
```

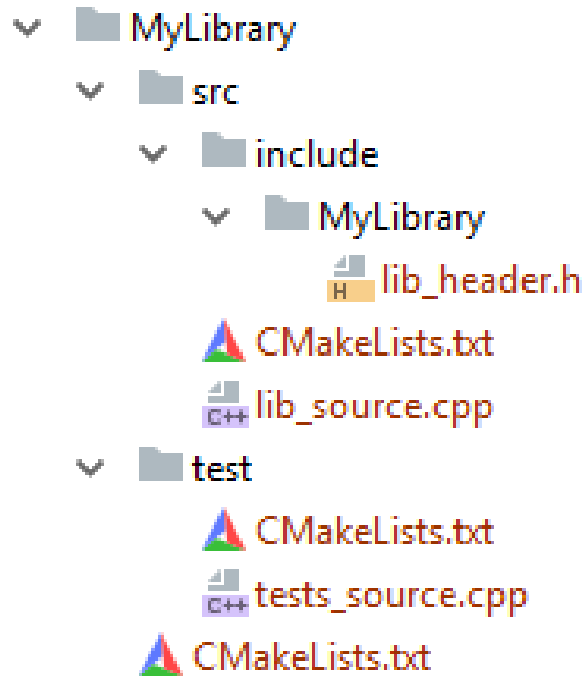
FILES ORGANIZATION



FILES ORGANIZATION

MYLIBRARY/SRC/CMAKELISTS.TXT

- *Standalone* library definition and installation
- Does not change compiler's warnings!



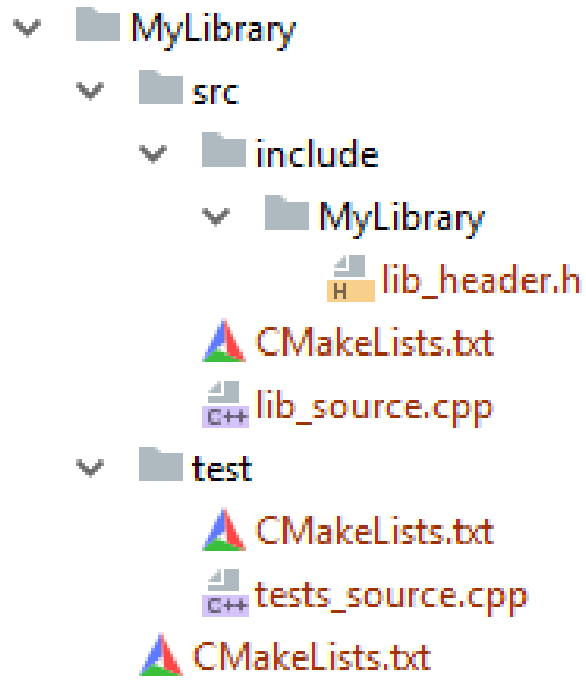
FILES ORGANIZATION

MYLIBRARY/SRC/CMAKELISTS.TXT

- *Standalone* library definition and installation
- Does not change compiler's warnings!

MYLIBRARY/TEST/CMAKELISTS.TXT

- *Standalone* unit tests definition



FILES ORGANIZATION

MYLIBRARY/SRC/CMAKELISTS.TXT

- *Standalone* library definition and installation
- Does not change compiler's warnings!

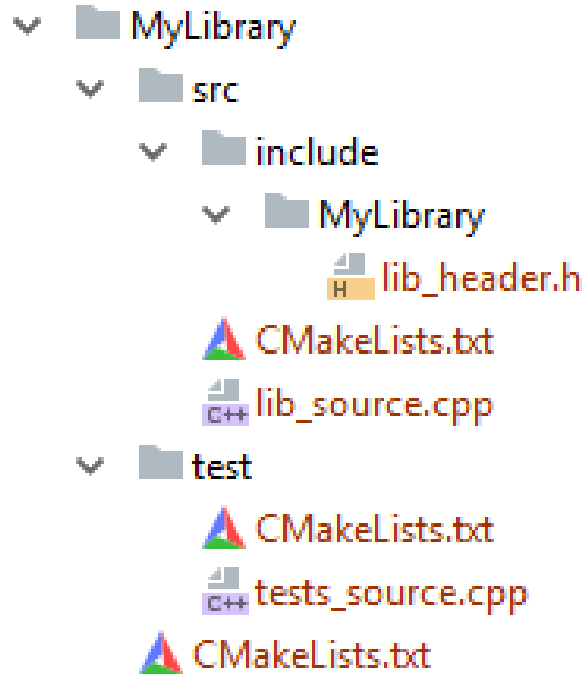
MYLIBRARY/TEST/CMAKELISTS.TXT

- *Standalone* unit tests definition

MYLIBRARY/CMAKELISTS.TXT

- Simple project *wrapper*
- Entry point for development
- **`src`** and **`test`** subdirectories added with

  `add_subdirectory()`



WRAPPER / ENTRY POINT

```
cmake_minimum_required(VERSION 3.5)
project(MyLibrary)

# add project code
add_subdirectory(src)

# add unit tests
enable_testing()
add_subdirectory(test)
```

- Useful for *project development*
  - no need to install the library in order to run unit tests
- Entry point for IDEs like CLion or Visual Studio

MODERN CMAKE CODE OF CONDUCT

- Don't use macros that affect all targets
  - `add_definitions()`
  - `add_compile_options()`
  - `include_directories()`
  - `link_directories()`
  - `link_libraries()`

MODERN CMAKE CODE OF CONDUCT

- Don't use macros that affect all targets
  - `add_definitions()`
  - `add_compile_options()`
  - `include_directories()`
  - `link_directories()`
  - `link_libraries()`
- Don't use `file(GLOB)` in projects

MODERN CMAKE CODE OF CONDUCT

- Don't use macros that affect all targets
  - `add_definitions()`
  - `add_compile_options()`
  - `include_directories()`
  - `link_directories()`
  - `link_libraries()`
- Don't use `file(GLOB)` in projects
- Avoid unnecessary variables

MODERN CMAKE CODE OF CONDUCT

- Don't use macros that affect all targets
  - `add_definitions()`
  - `add_compile_options()`
  - `include_directories()`
  - `link_directories()`
  - `link_libraries()`
- Don't use `file(GLOB)` in projects
- Avoid unnecessary variables
- Keep your hands out of `CXX_FLAGS`

MODERN CMAKE CODE OF CONDUCT

- Don't use macros that affect all targets
  - `add_definitions()`
  - `add_compile_options()`
  - `include_directories()`
  - `link_directories()`
  - `link_libraries()`
- Don't use `file(GLOB)` in projects
- Avoid unnecessary variables
- Keep your hands out of `CXX_FLAGS`
- Don't use `target_include_directories()` with a path outside of your module

MODERN CMAKE CODE OF CONDUCT

- Don't use macros that affect all targets
  - `add_definitions()`
  - `add_compile_options()`
  - `include_directories()`
  - `link_directories()`
  - `link_libraries()`
- Don't use `file(GLOB)` in projects
- Avoid unnecessary variables
- Keep your hands out of `CXX_FLAGS`
- Don't use `target_include_directories()` with a path outside of your module
- Don't use `target_link_libraries()` without specifying `PRIVATE`, `PUBLIC` or `INTERFACE`

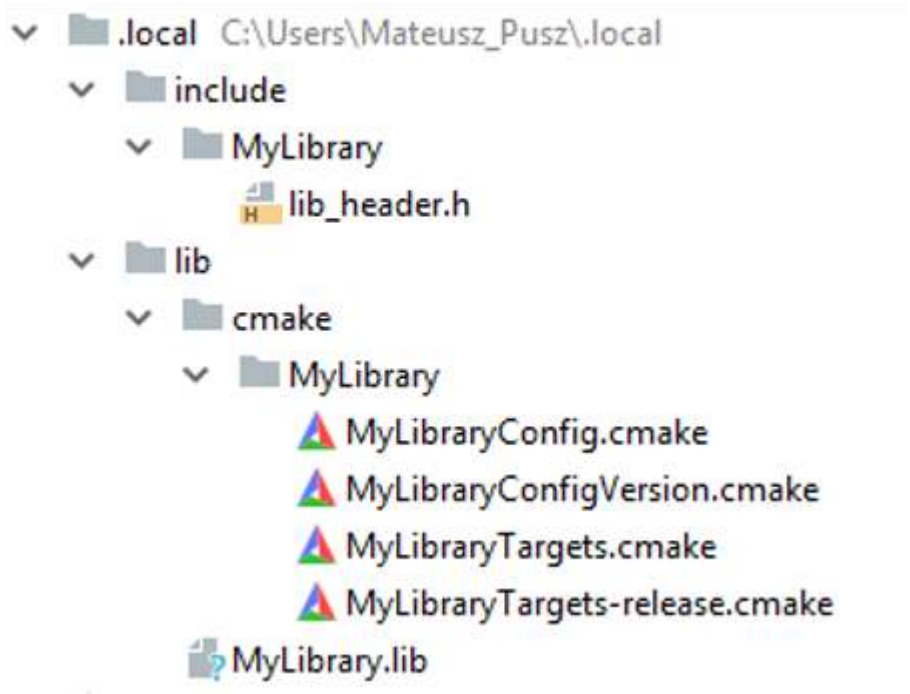
COMPILED, BUILT, TESTED

Done?

DON'T BE ASOCIAL!

David Sankel: Big Projects, and CMake, and Git, Oh My!

EXPORT YOUR LIBRARY INTERFACE



EXPORT YOUR LIBRARY INTERFACE

CMAKELISTS.TXT

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)
find_package(Foo 1.0 REQUIRED)
add_library(MyLibrary lib_source.cpp)
```

EXPORT YOUR LIBRARY INTERFACE

CMAKELISTS.TXT

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)
find_package(Foo 1.0 REQUIRED)
add_library(MyLibrary lib_source.cpp)
```

```
install(TARGETS MyLibrary EXPORT MyLibraryTargets
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib
        RUNTIME DESTINATION bin
        INCLUDES DESTINATION include)
install(EXPORT MyLibraryTargets
        DESTINATION lib/cmake/MyLibrary
        FILE MyLibraryTargets.cmake
        NAMESPACE MyCompany::)
```

EXPORT YOUR LIBRARY INTERFACE

CMAKELISTS.TXT

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)
find_package(Foo 1.0 REQUIRED)
add_library(MyLibrary lib_source.cpp)
```

```
install(TARGETS MyLibrary EXPORT MyLibraryTargets
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib
        RUNTIME DESTINATION bin
        INCLUDES DESTINATION include)
install(EXPORT MyLibraryTargets
        DESTINATION lib/cmake/MyLibrary
        FILE MyLibraryTargets.cmake
        NAMESPACE MyCompany::)
```

```
install(DIRECTORY include/MyLibrary
        DESTINATION include)
```

EXPORT YOUR LIBRARY INTERFACE

CMAKELISTS.TXT

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)
find_package(Foo 1.0 REQUIRED)
add_library(MyLibrary lib_source.cpp)
```

```
include(CMakePackageConfigHelpers)
write_basic_package_version_file(MyLibraryConfigVersion.cmake
        COMPATIBILITY SameMajorVersion)
install(FILES MyLibraryConfig.cmake ${CMAKE_CURRENT_BINARY_DIR}/MyLibraryConfigVersion.cmake
        DESTINATION lib/cmake/MyLibrary)
```

EXPORT YOUR LIBRARY INTERFACE

CMAKELISTS.TXT

```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)
find_package(Foo 1.0 REQUIRED)
add_library(MyLibrary lib_source.cpp)
```

```
include(CMakePackageConfigHelpers)
write_basic_package_version_file(MyLibraryConfigVersion.cmake
        COMPATIBILITY SameMajorVersion)
install(FILES MyLibraryConfig.cmake ${CMAKE_CURRENT_BINARY_DIR}/MyLibraryConfigVersion.cmake
        DESTINATION lib/cmake/MyLibrary)
```

MYLIBRARYCONFIG.CMAKE

```
include(CMakeFindDependencyMacro)
find_dependency(Foo 1.0)
include("${CMAKE_CURRENT_LIST_DIR}/MyLibraryTargets.cmake")
```

PACKAGE TESTING WORKFLOW

- Create and **install** the package

```
mkdir src/build
cd src/build
cmake -DCMAKE_BULD_TYPE=Release -DCMAKE_INSTALL_PREFIX=~/.local ..
cmake --build . --target install
```

PACKAGE TESTING WORKFLOW

- Create and **install** the package

```
mkdir src/build
cd src/build
cmake -DCMAKE_BULD_TYPE=Release -DCMAKE_INSTALL_PREFIX=~/.local ..
cmake --build . --target install
```

- Compile and **run tests** using the installed library

```
mkdir test/build
cd test/build
cmake -DCMAKE_BULD_TYPE=Release -DCMAKE_INSTALL_PREFIX=~/.local ..
ctest -VV
```

PURE CMAKE: DEPENDENCIES THE WRONG WAY

PROCESS

- *Build* each repository *in isolation*, *generate* and *install* its binaries along with a CMake config file
- For each project that has dependencies, use ***`find_package()`*** *to load* the config file and use the library

PURE CMAKE: DEPENDENCIES THE WRONG WAY

PROCESS

- *Build* each repository *in isolation*, *generate* and *install* its binaries along with a CMake config file
- For each project that has dependencies, use ***`find_package()`*** *to load* the config file and use the library

PROBLEMS

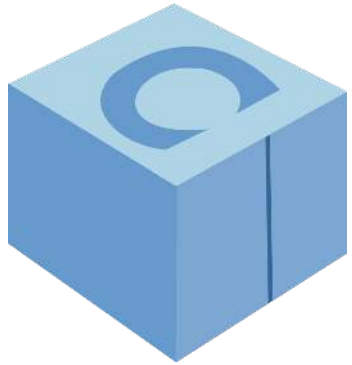
- Updating a program implies *recompiling* its package and then every one of its dependers *manually*
- *It doesn't scale* (many levels of dependencies, many configurations, ...)
- What about *supporting different versions?*
  - Release, Debug, RelWithDebInfo, ...
  - different compilers (gcc, clang, ...), compiler versions, C++ libraries (libc++ vs libstdc++)
  - different runtime libraries
  - different package configurations (no exceptions, shared lib, ...)

WHAT WE WANT?

- One build process *builds the project and all dependencies*
- *Only required* dependencies are being *rebuilt*
  - reuse of prebuilt binaries if available and up-to-date
- *No need to manually* download, build and install the dependencies
- Possibility to use *our own versions of dependencies* (ZLib, Boost, etc) instead of using system versions

CONAN

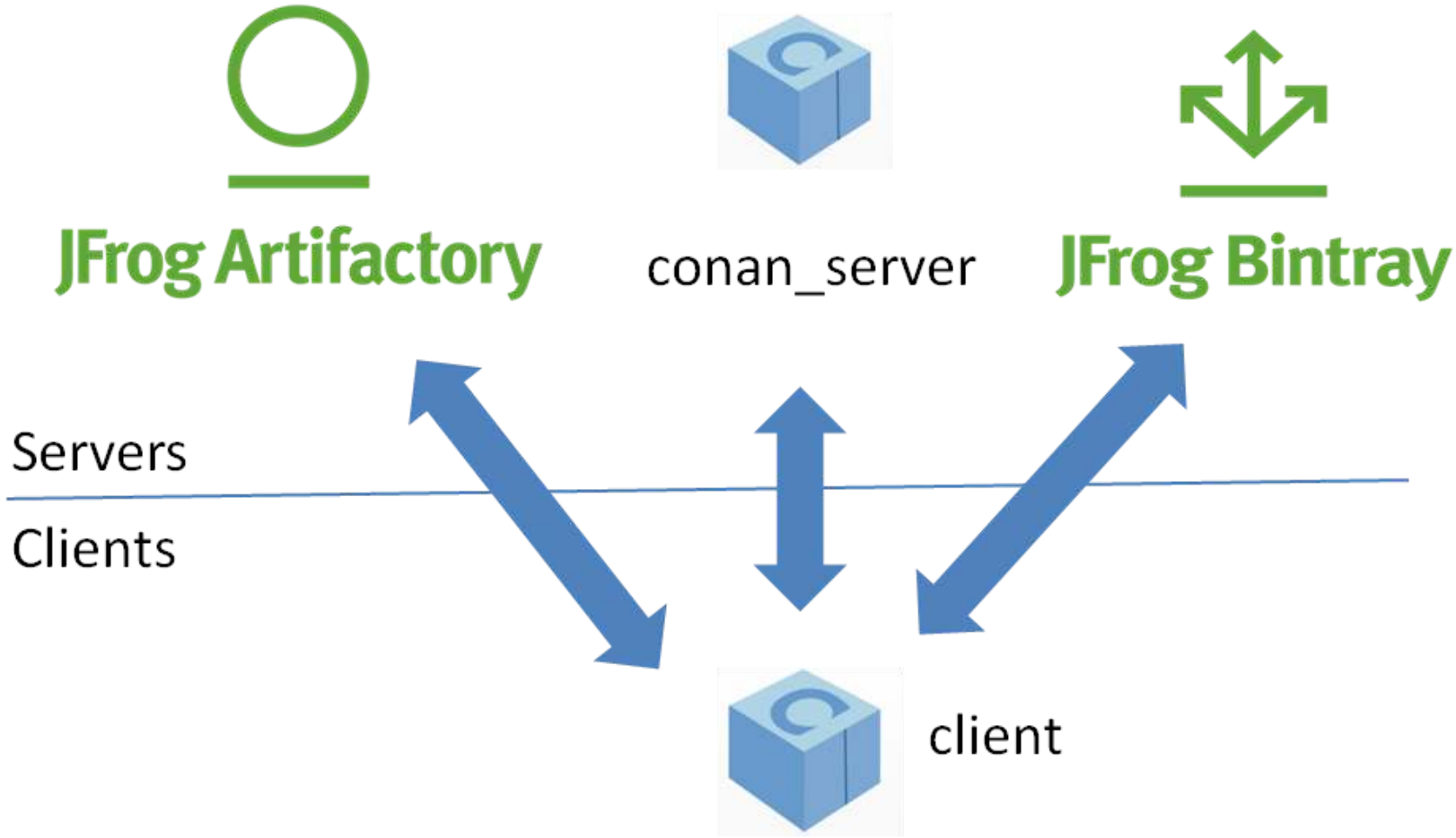
- Conan is **OSS**, with a MIT license
- **Decentralized** package manager with
  a client-server architecture
- The servers are just package storage
  - they do not build nor create the packages
- *The packages are created by the client*
  - packaging of **prebuilt** binaries
  - building from **sources** if needed
- **Portable** to any platform supporting Python
- *Works with any build system*
- Uses **Python** as its scripting language
- Easy hosting in a cloud or on a local server



conan.io

C++ package manager

CONAN CLIENT-SERVER ARCHITECTURE



CONAN CLIENT-SERVER ARCHITECTURE

CONAN CLIENT

- Console/terminal application
- Package creation and consumption
- Local cache for package storage
  - allows offline work

CONAN SERVER

- Quite simple TCP server
- User can run it as a daemon or a service

CONAN CLIENT-SERVER ARCHITECTURE

CONAN CLIENT

- Console/terminal application
- Package creation and consumption
- Local cache for package storage
  - allows offline work

CONAN SERVER

- Quite simple TCP server
- User can run it as a daemon or a service

JFROG ARTIFACTORY

- Offers conan repositories
- More powerful than conan server (WebUI, multiple auth protocols, High Availability, ...)
- JFrog Artifactory Community Edition for C/C++

JFROG BINTRAY

- Provides public and free hosting service for OSS conan packages
- Account is only needed to upload packages (anonymous read access)
- conan-center - moderated official repository

CONAN PACKAGE IDENTIFIER

```
package_name/package_version@user/channel
```

PACKAGE_NAME

- Usually project/library name

USER

- Owner of the package version
- Namespace that allows different users to have their *own packages for the same library* with the same name

PACKAGE_VERSION

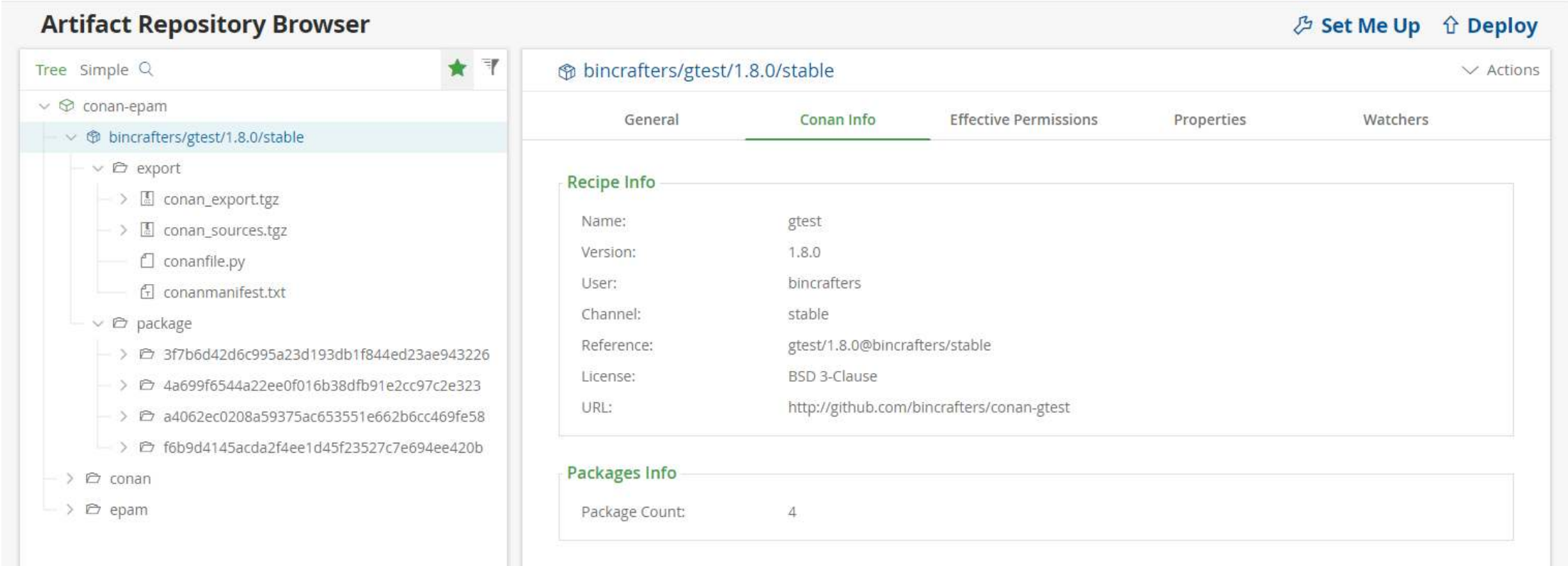
- Any string

CHANNEL

- Allows *different packages* for the same library
- Usually denote the maturity of a package ("stable", "testing")

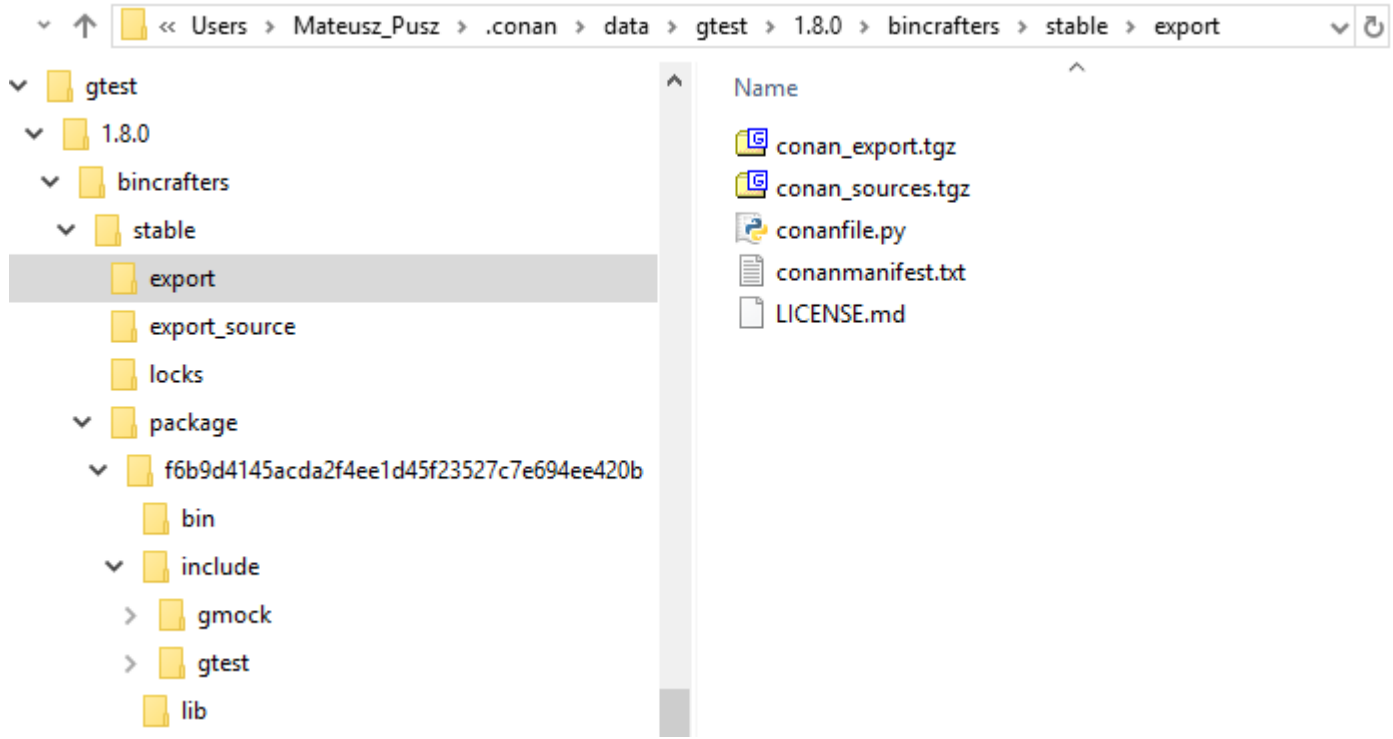
CONAN PACKAGES

- gtest/1.8.0@bincrafters/stable



CONAN PACKAGES

- gtest/1.8.0@bincrafters/stable



INSPECTING DEPENDENCIES

- *List* packages in the *local cache*

```
conan search
```

INSPECTING DEPENDENCIES

- *List* packages in the *local cache*

```
conan search
```

- Inspect binary *package details*

```
conan search gtest/1.8.0@bincrafters/stable
```

INSPECTING DEPENDENCIES

- *List* packages in the *local cache*

```
conan search
```

- Inspect binary *package details*

```
conan search gtest/1.8.0@bincrafters/stable
```

- Inspect your current *project's dependencies*

```
conan info ..
```

INSPECTING DEPENDENCIES

- *List* packages in the *local cache*

```
conan search
```

- Inspect binary *package details*

```
conan search gtest/1.8.0@bincrafters/stable
```

- Inspect your current *project's dependencies*

```
conan info ..
```

- Possibility to generate table of all binary packages and graph of all dependencies

```
conan search gtest/1.8.0@bincrafters/stable --table=conan_matrix.html
conan info .. --graph=graph.html
```

PACKAGE DETAILS

```
conan info OpenSSL/1.1.0g@conan/stable
```

PACKAGE DETAILS

```
conan info OpenSSL/1.1.0g@conan/stable
```

```
  OpenSSL/1.1.0g@conan/stable
      ID: 606fdb601e335c2001bdf31d478826b644747077
      BuildID: None
      Remote: conan-center=https://conan.bintray.com
      URL: http://github.com/lasote/conan-openssl
      License: The current OpenSSL licence is an 'Apache style' license: https://www.openssl.org/source/license.html
      Updates: Version not checked
      Creation date: 2018-03-29 18:34:10
      Required by:
          None
      Requires:
          zlib/1.2.11@conan/stable
  zlib/1.2.11@conan/stable
      ID: 6cc50b139b9c3d27b3e9042d5f5372d327b3a9f7
      BuildID: None
      Remote: conan-center=https://conan.bintray.com
      URL: http://github.com/lasote/conan-zlib
      License: http://www.zlib.net/zlib_license.html
      Updates: Version not checked
      Creation date: 2018-04-02 15:25:03
      Required by:
          OpenSSL/1.1.0g@conan/stable
```

PACKAGE DETAILS

```
conan search gtest/1.8.0@bincrafters/stable
```

PACKAGE DETAILS

```
conan search gtest/1.8.0@bincrafters/stable
```

```
Existing packages for recipe gtest/1.8.0@bincrafters/stable:

    Package_ID: f6b9d4145acda2f4ee1d45f23527c7e694ee420b
        [options]
            build_gmock: True
            shared: False
        [settings]
            arch: x86_64
            build_type: Debug
            compiler: Visual Studio
            compiler.runtime: MDd
            compiler.version: 15
            os: Windows
        Outdated from recipe: False
```

INSTALLING DEPENDENCIES

- Installing specific package **by hand**

```
conan install gtest/1.8.0@bincrafters/stable -g cmake
```

INSTALLING DEPENDENCIES

- Installing specific package **by hand**

```
conan install gtest/1.8.0@bincrafters/stable -g cmake
```

- Usage of project-specific **`conanfile.txt`** or **`conanfile.py`** files

```
conan install ..
```

INSTALLING DEPENDENCIES

- Installing specific package **by hand**

```
conan install gtest/1.8.0@bincrafters/stable -g cmake
```

- Usage of project-specific **conanfile.txt** or **conanfile.py** files

```
conan install ..
```

- Optionally a **profile** name can be provided (otherwise **default** is used)

```
conan install .. -pr vs2017
```

INSTALLING DEPENDENCIES

- Installing specific package **by hand**

```
conan install gtest/1.8.0@bincrafters/stable -g cmake
```

- Usage of project-specific **`conanfile.txt`** or **`conanfile.py`** files

```
conan install ..
```

- Optionally a **profile** name can be provided (otherwise **`default`** is used)

```
conan install .. -pr vs2017
```

- Build packages **from sources** if prebuilt ones are not available

```
conan install .. --build missing
```

CONAN FILES

CONANFILE.TXT

```
[requires]
Foo/1.0@my_channel/stable
gtest/1.8.0@bincrafters/stable

[options]

[generators]
cmake
```

- **`cmake`** generator creates **`conanbuildinfo.cmake`** file that defines CMake
  - variables (module paths include paths, library names, ...)
  - helper functions (defining targets, configuring CMake environment, ...)
- Many different generators provided to support any build system

CONAN FILES

CONANFILE.TXT

```
[requires]
Foo/1.0@my_channel/stable
gtest/1.8.0@bincrafters/stable

[options]

[generators]
cmake
```

CONANFILE.PY

```
from conans import ConanFile

class MyLibraryConan(ConanFile):
    requires = (
        "Foo/1.0@my_channel/stable",
        "gtest/1.8.0@bincrafters/stable"
    )
    generators = "cmake"
```

- **`cmake`** generator creates **`conanbuildinfo.cmake`** file that defines CMake
  - variables (module paths include paths, library names, ...)
  - helper functions (defining targets, configuring CMake environment, ...)
- Many different generators provided to support any build system

SETTING PACKAGE OPTIONS

CONANFILE.TXT

```
[requires]
Foo/1.0@my_channel/stable
gtest/1.8.0@bincrafters/stable

[options]
gtest:shared=True

[generators]
cmake
```

CONANFILE.PY

```
from conans import ConanFile

class MyLibraryConan(ConanFile):
    requires = (
        "Foo/1.0@my_channel/stable",
        "gtest/1.8.0@bincrafters/stable"
    )
    default_options = "gtest:shared=True"
    generators = "cmake"
```

SETTING PACKAGE OPTIONS

CONANFILE.TXT

```
[requires]
Foo/1.0@my_channel/stable
gtest/1.8.0@bincrafters/stable

[options]
gtest:shared=True

[generators]
cmake
```

CONANFILE.PY

```
from conans import ConanFile

class MyLibraryConan(ConanFile):
    requires = (
        "Foo/1.0@my_channel/stable",
        "gtest/1.8.0@bincrafters/stable"
    )
    default_options = "gtest:shared=True"
    generators = "cmake"
```

- Command line

```
conan install .. -o gtest:shared=True
```

```
conan install .. -o *:shared=True
```

FIXING IMPORTS FOR SHARED LIBRARIES

CONANFILE.TXT

```
[requires]
Foo/1.0@my_channel/stable
gtest/1.8.0@bincrafters/stable

[options]
gtest:shared=True

[generators]
cmake

[imports]
bin, *.dll -> ./bin
lib, *.dylib* -> ./bin
```

CONANFILE.PY

```
from conans import ConanFile

class MyLibraryConan(ConanFile):
    requires = (
        "Foo/1.0@my_channel/stable",
        "gtest/1.8.0@bincrafters/stable"
    )
    default_options = "gtest:shared=True"
    generators = "cmake"

    def imports(self):
        self.copy("*.dll", dst="bin", src="bin")
        self.copy("*.dylib*", dst="bin", src="lib")
```

- Copies all **`*.dll`** files from packages **`bin`** folder to my **`./bin`** folder
- Copies all **`*.dylib*`** files from packages **`lib`** folder to my **`./bin`** folder

MORE POWER WITH CONANFILE.PY

```
from conans import ConanFile, CMake

class MyLibraryConan(ConanFile):
    settings = "os", "compiler", "build_type", "arch"
    requires = (
        "Foo/1.0@my_channel/stable",
        "gtest/1.8.0@bincrafters/stable"
    )
    generators = "cmake"

    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()
        cmake.install()
```

MORE POWER WITH CONANFILE.PY

```
from conans import ConanFile, CMake

class MyLibraryConan(ConanFile):
    settings = "os", "compiler", "build_type", "arch"
    requires = (
        "Foo/1.0@my_channel/stable",
        "gtest/1.8.0@bincrafters/stable"
    )
    generators = "cmake"

    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()
        cmake.install()
```

```
conan install . -pr vs2017 --install-folder build
conan build . --build-folder build
```

MORE POWER WITH CONANFILE.PY

```
from conans import ConanFile

class MyLibraryConan(ConanFile):
    requires = (
        "gtest/1.8.0@bincrafters/stable"
    )
    options = { "testing": [True, False] }
    default_options = "testing=False"
    generators = "cmake"

    def requirements(self):
        if self.options.testing:
           self.requires("Foo/2.0@my_channel/testing")
        else:
           self.requires("Foo/1.0@my_channel/stable")
```

MORE POWER WITH CONANFILE.PY

```
from conans import ConanFile

class MyLibraryConan(ConanFile):
    requires = (
        "gtest/1.8.0@bincrafters/stable"
    )
    options = { "testing": [True, False] }
    default_options = "testing=False"
    generators = "cmake"

    def requirements(self):
        if self.options.testing:
           self.requires("Foo/2.0@my_channel/testing")
        else:
           self.requires("Foo/1.0@my_channel/stable")
```

And many more features...

CONAN PROFILES

```
[settings]
setting=value

[options]
MyLib:shared=True

[env]
env_var=value

[build_requires]
Tool1/0.1@user/channel
Tool2/0.1@user/channel, Tool3/0.1@user/channel
```

- Stored in the default profile folder or anywhere in a project

CONAN PROFILES

```
conan profile show vs2017
```

CONAN PROFILES

```
conan profile show vs2017
```

```
Configuration for profile vs2017:

[settings]
os=Windows
arch=x86_64
compiler=Visual Studio
compiler.version=15
build_type=Release
arch_build=x86_64
os_build=Windows
[options]
[build_requires]
[env]
```

CONAN PROFILES

```
conan profile show vs2017
```

```
Configuration for profile vs2017:

[settings]
os=Windows
arch=x86_64
compiler=Visual Studio
compiler.version=15
build_type=Release
arch_build=x86_64
os_build=Windows
[options]
[build_requires]
[env]
```

- Easy to override or extend the profile

```
conan install .. -pr vs2017 -s build_type=Debug
```

CONAN PROFILES

- Example of a profile to install Poco dependencies as shared and in debug mode

DEBUG_SHARED

```
include(default)

[settings]
build_type=Debug

[options]
Poco:shared=True
Poco:enable_apacheconnector=False
OpenSSL:shared=True
```

```
conan install .. -pr=../debug_shared
```

USING CONAN WITH CMAKE

```
cmake_minimum_required(VERSION 3.5)
project(MyLibrary)

if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    conan_basic_setup(TARGETS)
endif()

# add project code
add_subdirectory(src)

# add unit tests
enable_testing()
add_subdirectory(unit_tests)
```

- Above code supports optional use of Conan
- **`if()`** statement can be skipped if Conan usage is mandatory in a project

USING CONAN WITH CMAKE

- Full Conan support with definition of **`CONAN_PKG::XXX`** packages

```
if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    conan_basic_setup(TARGETS)
endif()
```

USING CONAN WITH CMAKE

- Full Conan support with definition of **`CONAN_PKG::XXX`** packages

```
if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    conan_basic_setup(TARGETS)
endif()
```

- Enough to make **`find_package()`** work

```
if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    conan_set_find_paths()
endif()
```

USING CONAN WITH CMAKE

- Full Conan support with definition of **`CONAN_PKG::XXX`** packages

```
if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    conan_basic_setup(TARGETS)
endif()
```

- Enough to make **`find_package()`** work

```
if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    conan_set_find_paths()
endif()
```

```
if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    set(CMAKE_MODULE_PATH ${CONAN_CMAKE_MODULE_PATH} ${CMAKE_MODULE_PATH})
endif()
```

USING CONAN WITH CMAKE

- Running both **`find_package(GTest)`** and **`conan_basic_setup(TARGETS)`** duplicates targets
  - **`GTest::GTest`**, **`GTest::Main`**
  - **`CONAN_PKG::gtest`** (linking **`gmock_main`**, **`gmock`** and **`gtest`**)
- Which one to use?

USING CONAN WITH CMAKE

- Running both **`find_package(GTest)`** and **`conan_basic_setup(TARGETS)`** duplicates targets
  - **`GTest::GTest`**, **`GTest::Main`**
  - **`CONAN_PKG::gtest`** (linking **`gmock_main`**, **`gmock`** and **`gtest`**)
- Which one to use?
- It depends...

USING CONAN WITH CMAKE

- Running both **`find_package(GTest)`** and **`conan_basic_setup(TARGETS)`** duplicates targets
  - **`GTest::GTest`**, **`GTest::Main`**
  - **`CONAN_PKG::gtest`** (linking **`gmock_main`**, **`gmock`** and **`gtest`**)
- Which one to use?
- It depends...
- In general **`find_package(XXX)`** targets are more mature

USING CONAN WITH CMAKE

- Running both **`find_package(GTest)`** and **`conan_basic_setup(TARGETS)`** duplicates targets
  - **`GTest::GTest`**, **`GTest::Main`**
  - **`CONAN_PKG::gtest`** (linking **`gmock_main`**, **`gmock`** and **`gtest`**)
- Which one to use?
- It depends...
- In general **`find_package(XXX)`** targets are more mature
- But...
  - Difficult to make it properly address transitivity (WIP)
  - Does not support multi-configuration

USING CONAN WITH CMAKE

- Issues with **`find_package(XXX)`** are often patched by Conan recipe

```
googletest\include\gtest\gtest-printers.h(600): error C2220: warning treated as error - no 'object' file generated
                                                [googlemock\gtest\gtest.vcxproj]
     googletest\include\gtest\gtest-printers.h(600): warning C4996: 'std::tr1': warning STL4002: The non-Standard
             std::tr1 namespace and TR1-only machinery are deprecated and will be REMOVED. You can define
             _SILENCE_TR1_NAMESPACE_DEPRECATION_WARNING to acknowledge that you have received this warning.
                                                [googlemock\gtest\gtest.vcxproj]
```

USING CONAN WITH CMAKE

- Issues with **`find_package(XXX)`** are often patched by Conan recipe

```
googletest\include\gtest\gtest-printers.h(600): error C2220: warning treated as error - no 'object' file generated
                                                [googlemock\gtest\gtest.vcxproj]
     googletest\include\gtest\gtest-printers.h(600): warning C4996: 'std::tr1': warning STL4002: The non-Standard
             std::tr1 namespace and TR1-only machinery are deprecated and will be REMOVED. You can define
             _SILENCE_TR1_NAMESPACE_DEPRECATION_WARNING to acknowledge that you have received this warning.
                                                [googlemock\gtest\gtest.vcxproj]
```

CONANFILE.PY

```python
class GTestConan(ConanFile):

    def package_info(self):
        # ...
        if self.settings.compiler == "Visual Studio" and float(str(self.settings.compiler.version)) >= 15:
            self.cpp_info.defines.append("GTEST_LANG_CXX11=1")
            self.cpp_info.defines.append("GTEST_HAS_TR1_TUPLE=0")
```

USING CONAN WITH CMAKE

- Issues with **find_package(XXX)** are often patched by Conan recipe

```
googletest\include\gtest\gtest-printers.h(600): error C2220: warning treated as error - no 'object' file generated
                                                [googlemock\gtest\gtest.vcxproj]
     googletest\include\gtest\gtest-printers.h(600): warning C4996: 'std::tr1': warning STL4002: The non-Standard
             std::tr1 namespace and TR1-only machinery are deprecated and will be REMOVED. You can define
             _SILENCE_TR1_NAMESPACE_DEPRECATION_WARNING to acknowledge that you have received this warning.
                                                [googlemock\gtest\gtest.vcxproj]
```

CONANFILE.PY

```python
class GTestConan(ConanFile):

    def package_info(self):
        # ...
        if self.settings.compiler == "Visual Studio" and float(str(self.settings.compiler.version)) >= 15:
            self.cpp_info.defines.append("GTEST_LANG_CXX11=1")
            self.cpp_info.defines.append("GTEST_HAS_TR1_TUPLE=0")
```

FINDGTEST.CMAKE

```
set_property(TARGET GTest::Main PROPERTY INTERFACE_COMPILE_DEFINITIONS ${CONAN_COMPILE_DEFINITIONS_GTEST})
```

USING CONAN WITH CMAKE

- Choosing **`CONAN_PKG::XXX`**

```
if(CONAN)
    target_link_libraries(MyLibraryTests PRIVATE MyCompany::MyLibrary CONAN_PKG::gtest)
else()
    target_link_libraries(MyLibraryTests PRIVATE MyCompany::MyLibrary GTest::Main)
endif()
```

USING CONAN WITH CMAKE

- Choosing **CONAN_PKG::XXX**

```
if(CONAN)
    target_link_libraries(MyLibraryTests PRIVATE MyCompany::MyLibrary CONAN_PKG::gtest)
else()
    target_link_libraries(MyLibraryTests PRIVATE MyCompany::MyLibrary GTest::Main)
endif()
```

- Starting from CMake 3.11

```
if(EXISTS ${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
    conan_basic_setup(TARGETS)
    add_library(GTest::Main INTERFACE IMPORTED)
    target_link_libraries(GTest::Main INTERFACE CONAN_PKG::gtest)
else()
    find_package(GTest MODULE REQUIRED)
endif()
target_link_libraries(MyLibraryTests PRIVATE MyCompany::MyLibrary GTest::Main)
```

CONAN PACKAGE CREATION

- It is a complex subject and we are out of time
- For now please refer to Conan documentation
- Hopefully see you on CppCon 2018 ;-)

SUMMARY

SUMMARY

CMAKE

- Many projects still *do not use CMake at all*
- Many projects still *do not use CMake in a Modern way*
- Many projects still *do not provide installation option* with proper CMake configuration files generation

SUMMARY

CMAKE

- Many projects still *do not use CMake at all*
- Many projects still *do not use CMake in a Modern way*
- Many projects still *do not provide installation option* with proper CMake configuration files generation

CONAN

- Production quality Package Manager *designed with C++ in mind*
- For *free* and on MIT license
- Quite *easy to use*
- The *documentation* is really good
- **Give it a try!**


CAUTION
Programming
is addictive
(and too much fun)