GIT, CMAKE, CONAN

HOW TO SHIP AND REUSE OUR C++ PROJECTS?

Mateusz Pusz
September 24, 2018

THE MOST COMMON C++ TOOLSET

THE MOST COMMON C++ TOOLSET

VERSION CONTROL SYSTEM	git
BUILDING	
PACKAGE MANAGEMENT	

THE MOST COMMON C++ TOOLSET

VERSION CONTROL SYSTEM	git
BUILDING	CMake
PACKAGE MANAGEMENT	

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

- Please *do not partition* our C++ development environment even more
- Other tools may sometimes be better at one aspect but probably are worse at others
- Until a strong game-changer appears on the market please *use above tools*

THE MOST COMMON C++ TOOLSET

VERSION CONTROL SYSTEM	git
BUILDING	CMake
PACKAGE MANAGEMENT	None

- Please *do not partition* our C++ development environment even more
- Other tools may sometimes be better at one aspect but probably are worse at others
- Until a strong game-changer appears on the market please *use above tools*

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

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()`**

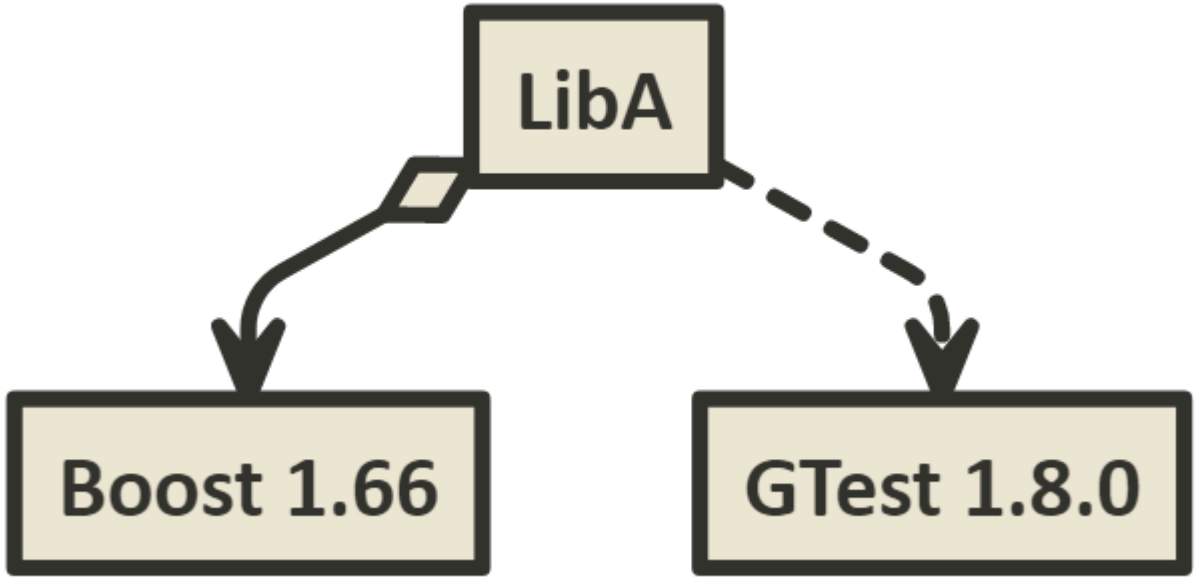
TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

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, ...)

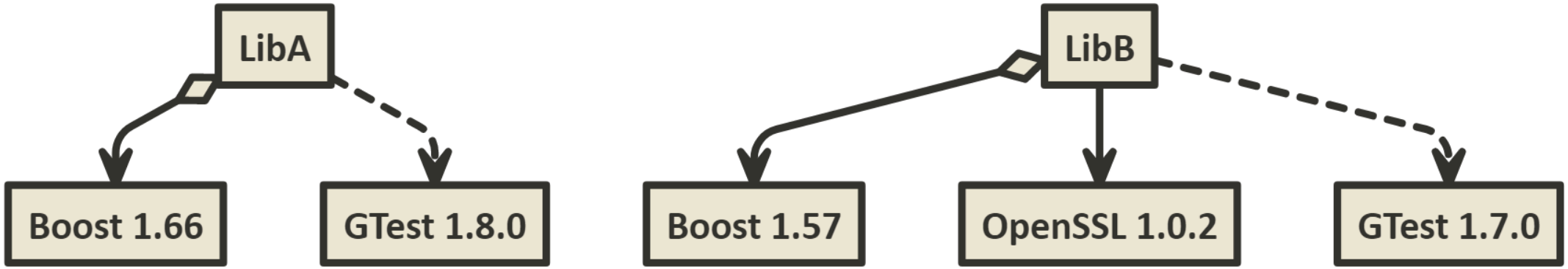
TYPICAL WAYS TO HANDLE DEPENDENCIES IN C++ PROJECTS

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)

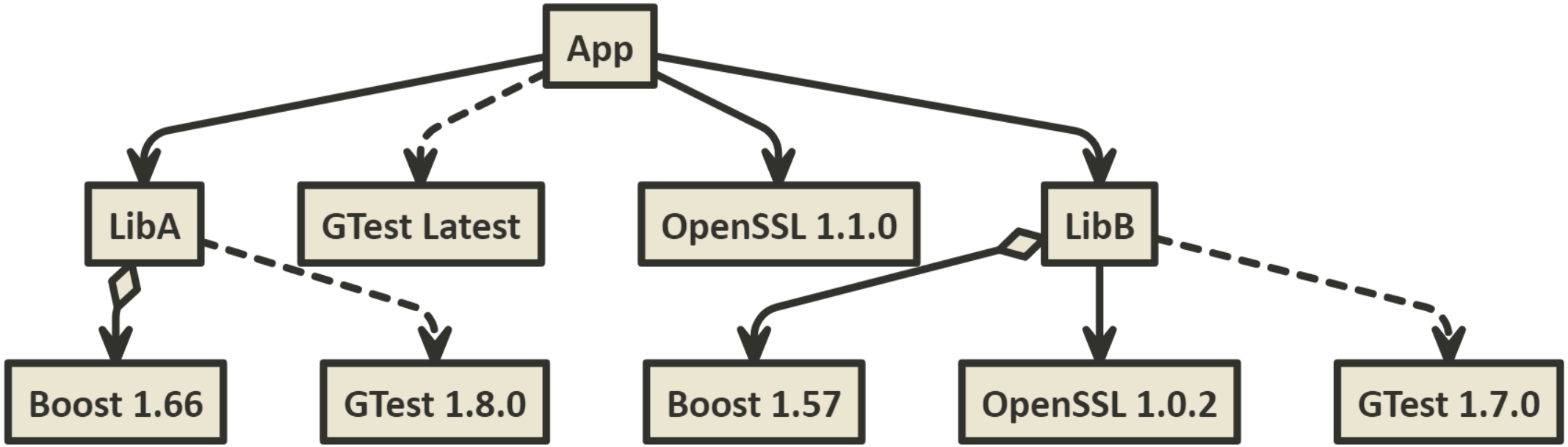
`add_subdirectory()` FOR DEPS?



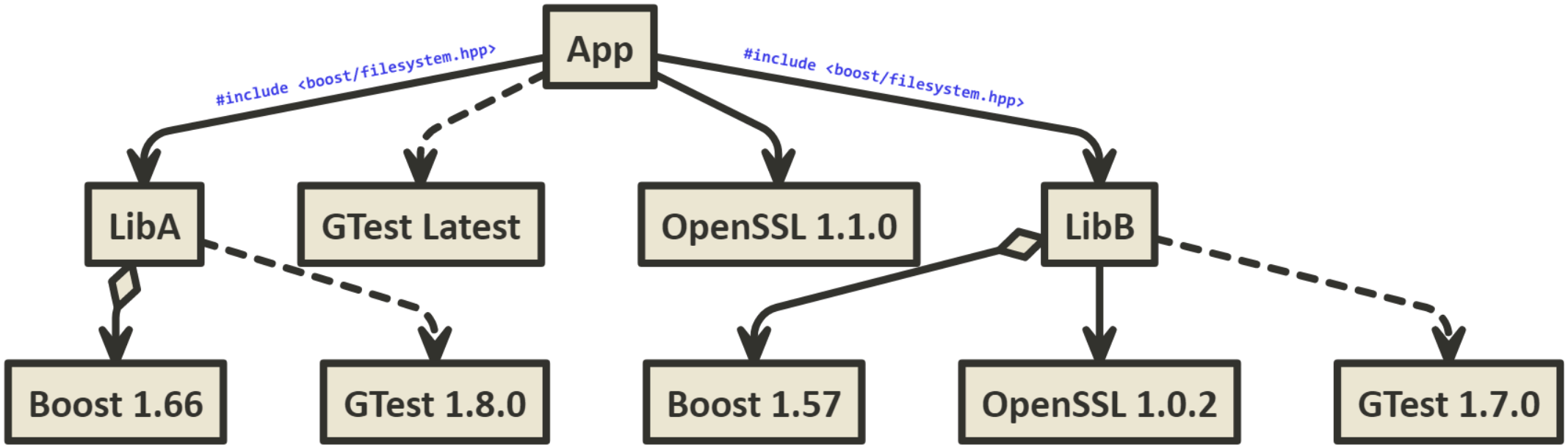
add_subdirectory() FOR DEPS?



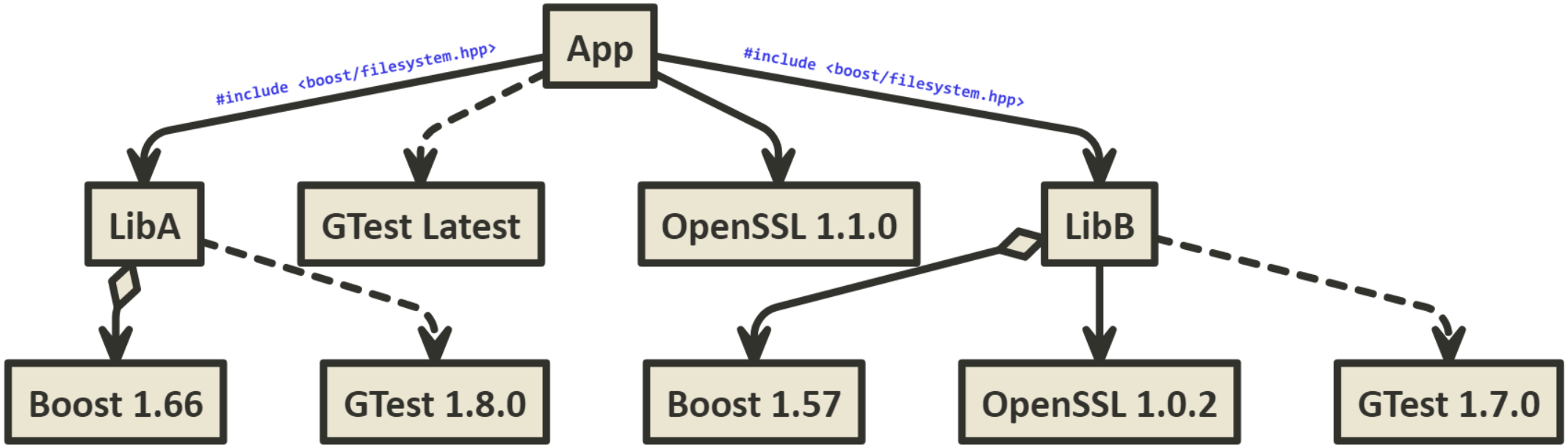
add_subdirectory() FOR DEPS?



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

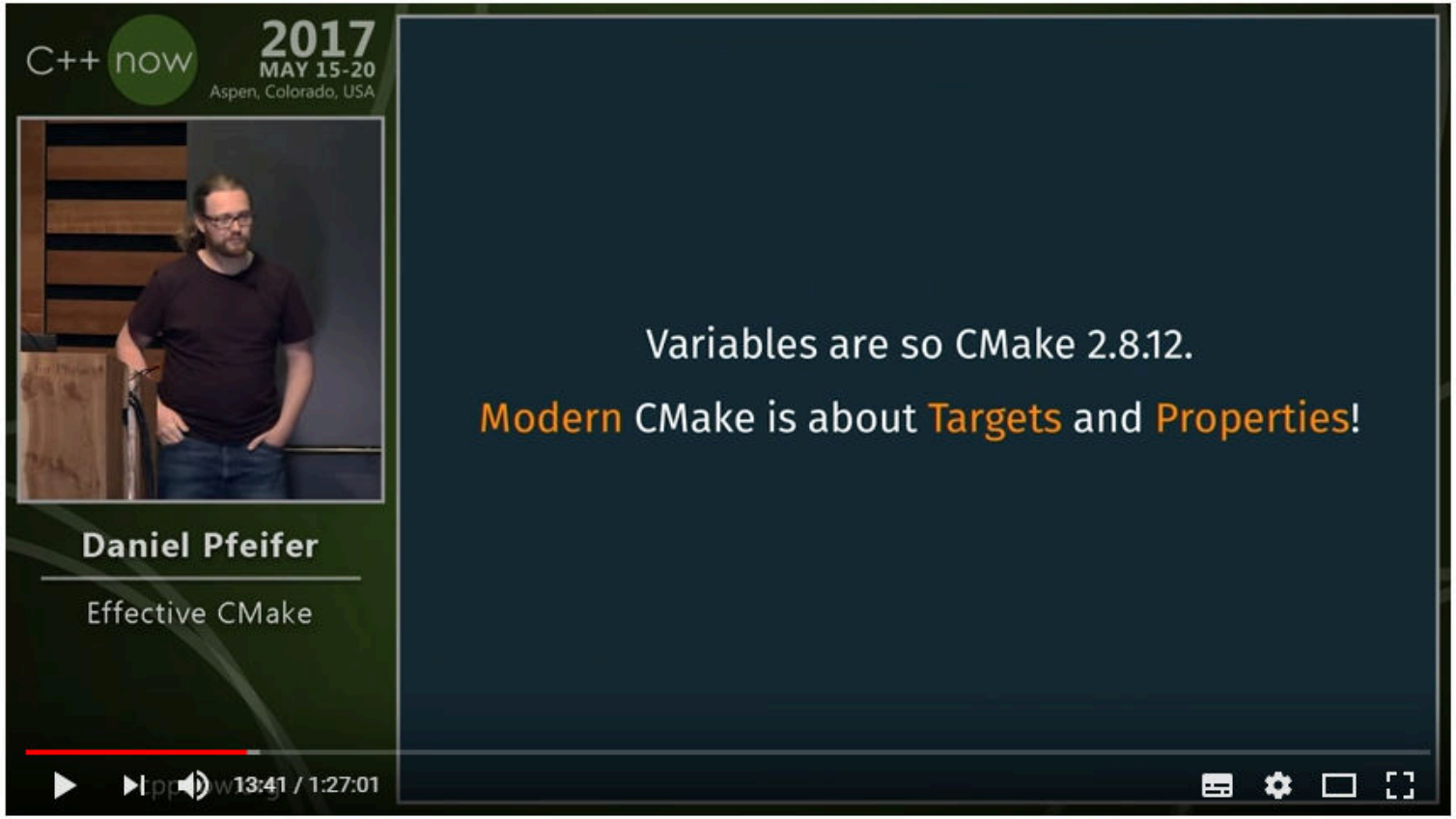
GCC

```
cmake .. -DCMAKE_BUILD_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

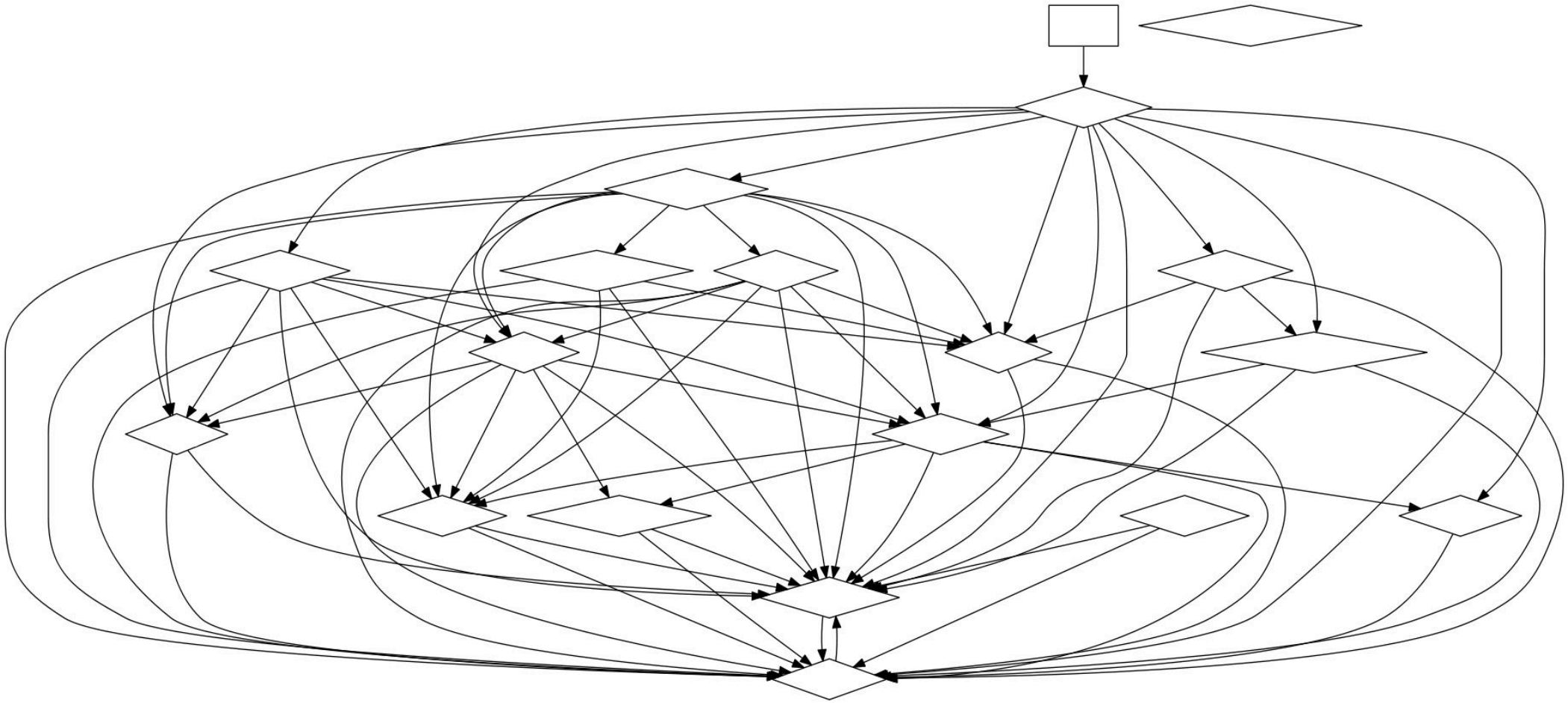
- Available since version 2.8.12 (Oct 2013)
- Specified by **`target_xxx()`** commands

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

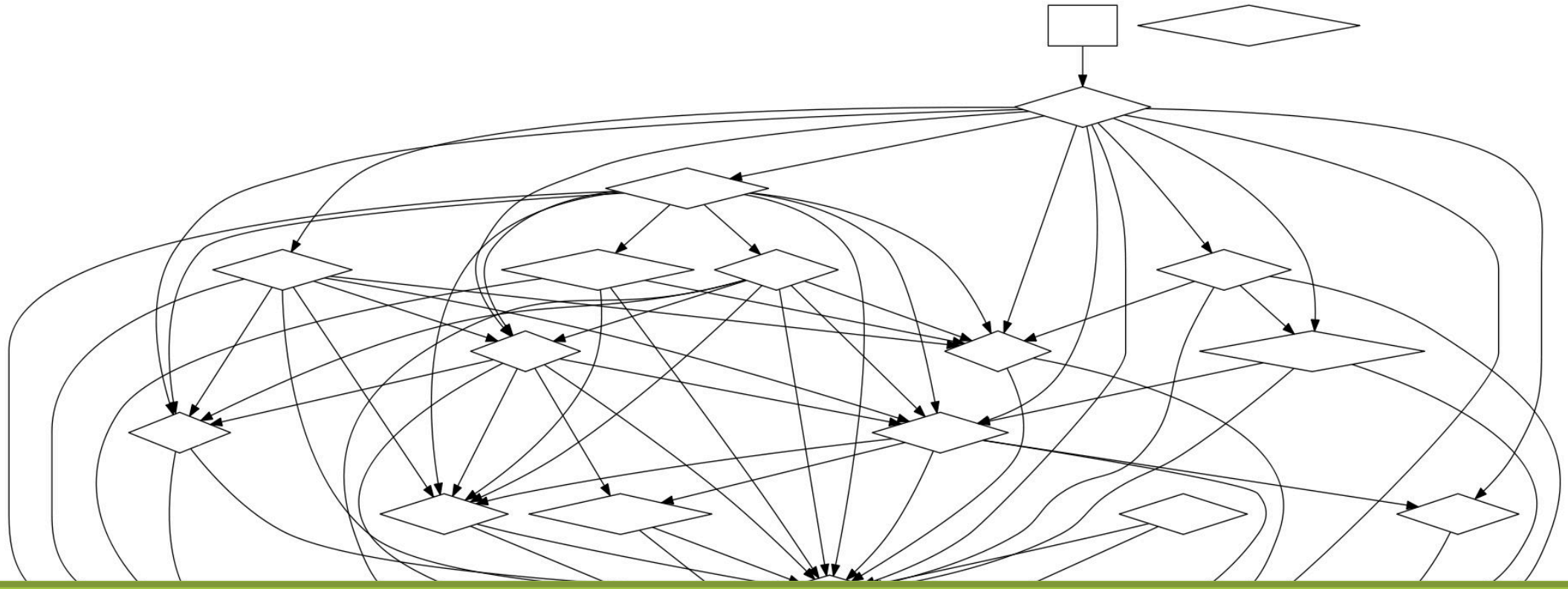
	NEEDED BY ME	NOT NEEDED BY ME
NEEDED BY DEPENDERS	PUBLIC	INTERFACE
NOT NEEDED BY DEPENDERS	PRIVATE	:-)

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

PHYSICAL DESIGN: OLD CMAKE STYLE



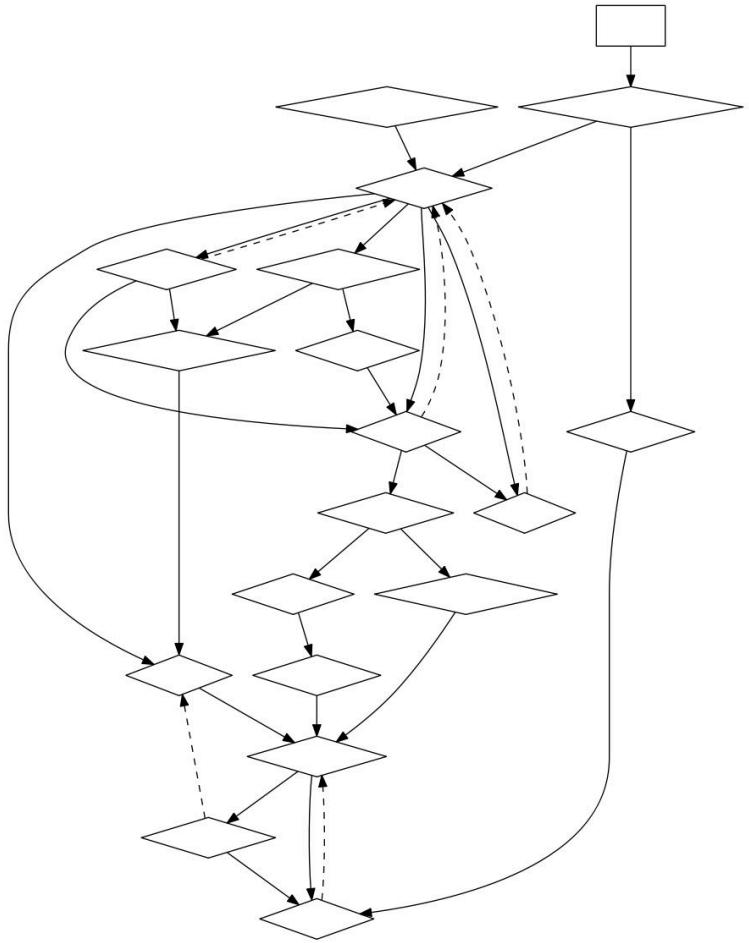
PHYSICAL DESIGN: OLD CMAKE STYLE



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

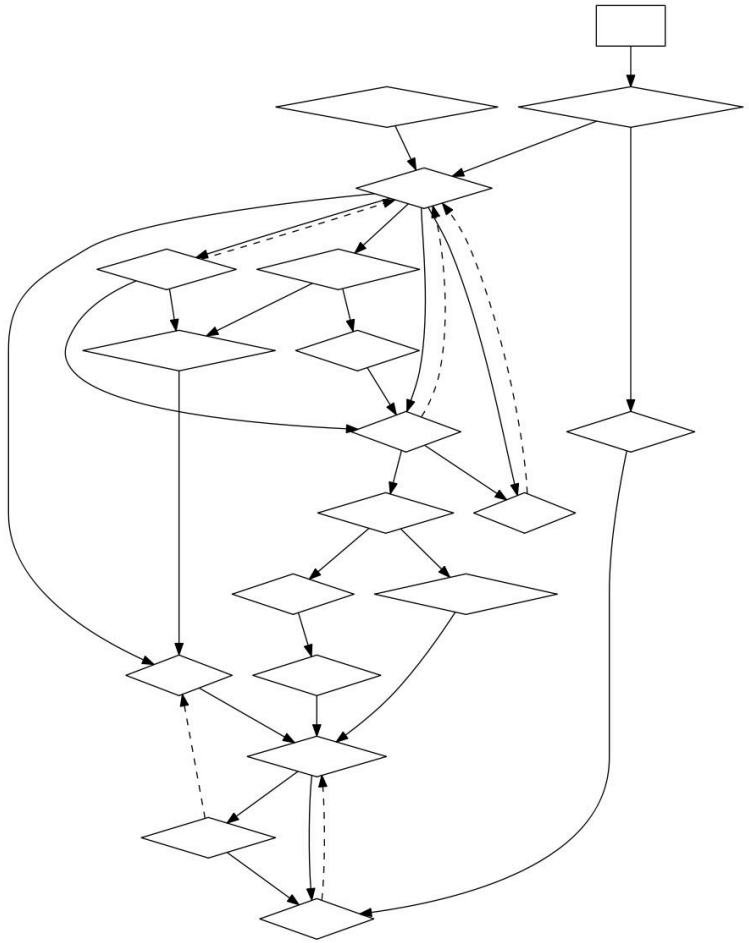
PHYSICAL DESIGN: MODERN CMAKE

- 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 lib_source.h include/MyLibrary/lib_header.h)
add_library(MyCompany::MyLibrary ALIAS MyLibrary)
```

ALIAS TARGETS

```
add_library(MyLibrary lib_source.cpp lib_source.h include/MyLibrary/lib_header.h)
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 lib_source.h include/MyLibrary/lib_header.h)
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 lib_source.h include/MyLibrary/lib_header.h)
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 lib_source.h include/MyLibrary/lib_header.h)
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 lib_source.h include/MyLibrary/lib_header.h)
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

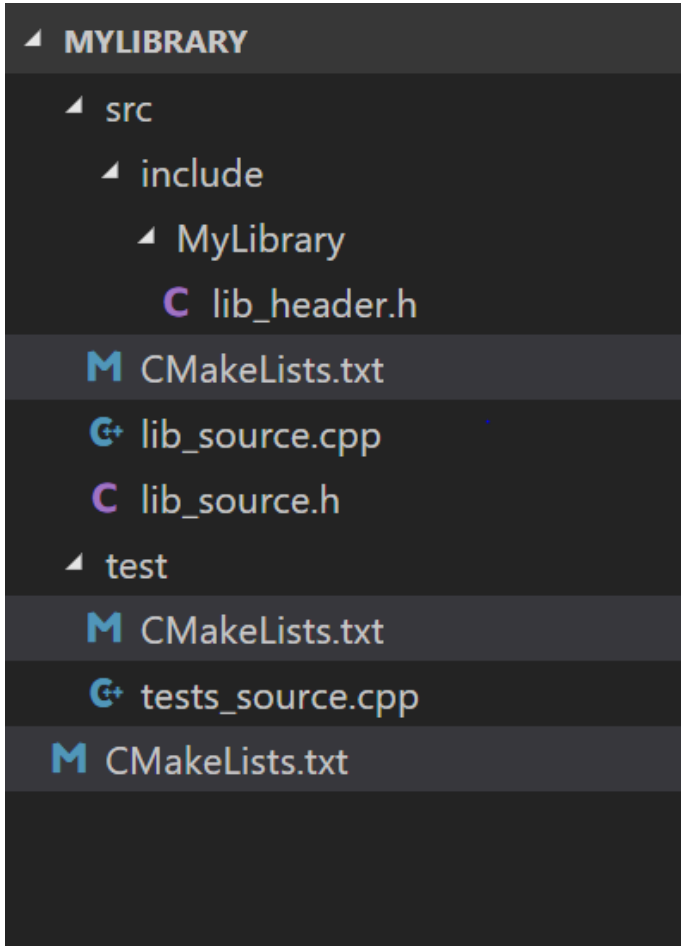
```
cmake_minimum_required(VERSION 3.8)
project(MyLibrary VERSION 0.0.1)

# dependencies
find_package(Foo 1.0 REQUIRED)

# library definition
add_library(MyLibrary lib_source.cpp lib_source.h include/MyLibrary/lib_header.h)
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

FILES ORGANIZATION

MYLIBRARY/SRC/CMAKELISTS.TXT

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



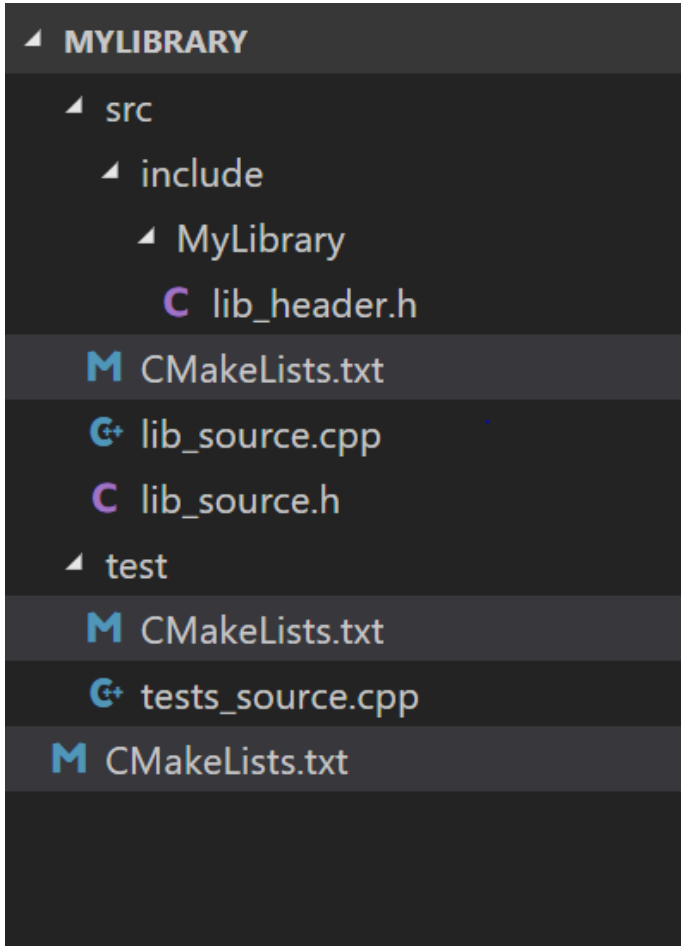
FILES ORGANIZATION

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!

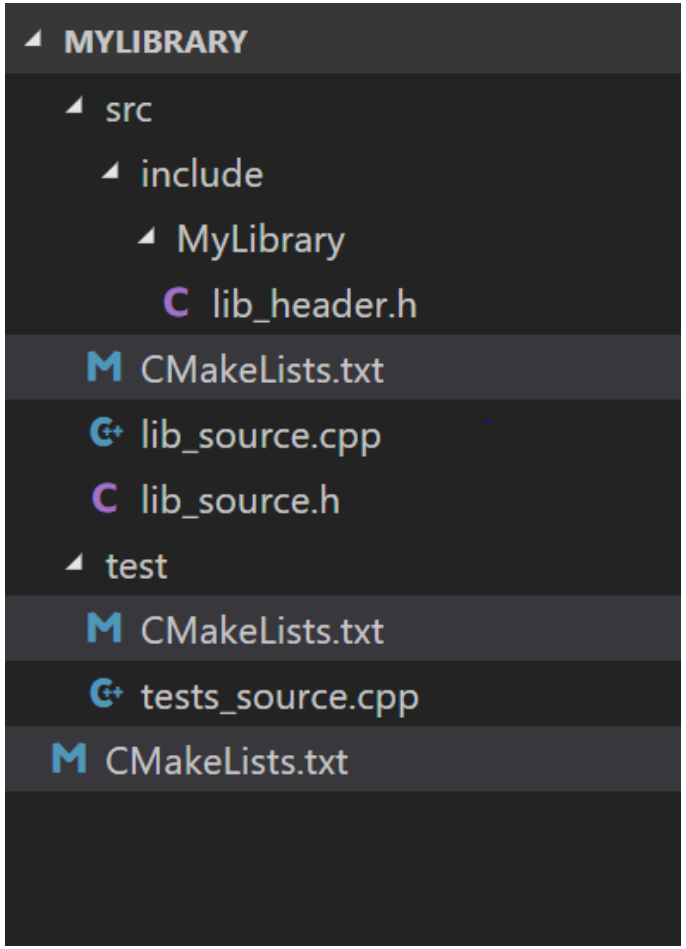
MYLIBRARY/TEST/CMAKELISTS.TXT

- *Standalone* unit tests definition

MYLIBRARY/CMAKELISTS.TXT

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

  `add_subdirectory()`



WRAPPER: LOCAL DEVELOPMENT 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 LIBRARY USAGE: TESTS

```
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
)
```

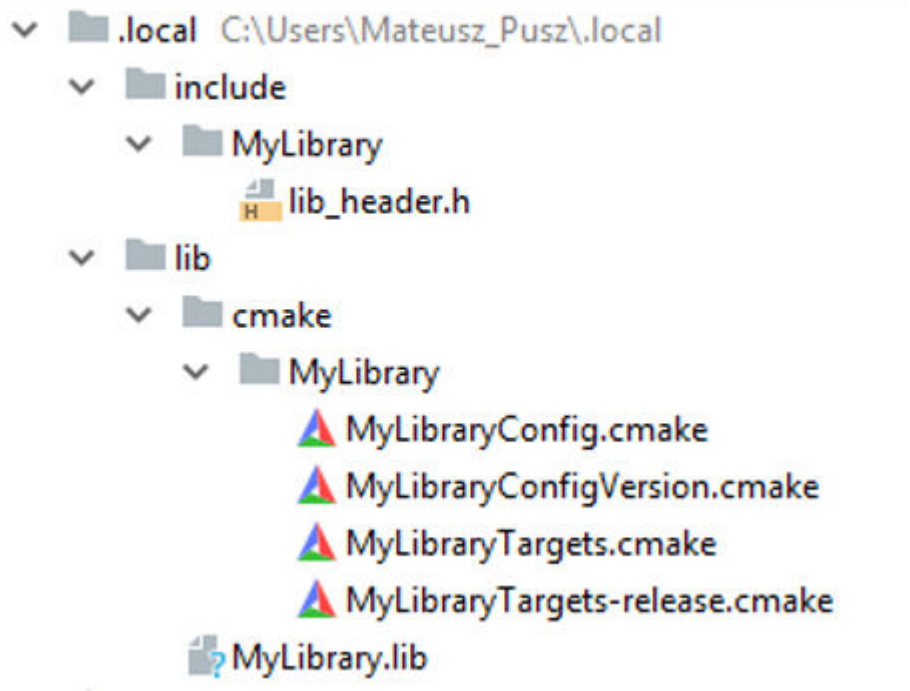
COMPILED, BUILT, TESTED

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 lib_source.h include/MyLibrary/lib_header.h)
```

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 lib_source.h include/MyLibrary/lib_header.h)
```

```
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 lib_source.h include/MyLibrary/lib_header.h)
```

```
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 lib_source.h include/MyLibrary/lib_header.h)
```

```
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 lib_source.h include/MyLibrary/lib_header.h)
```

```
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_BUILD_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_BUILD_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_BUILD_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

WHAT I WANT NOW AND I DID NOT REALIZE THAT EARLIER?

WHAT I WANT NOW AND I DID NOT REALIZE THAT EARLIER?

- *Decentralized* servers
  - OSS, corporate, private, etc

WHAT I WANT NOW AND I DID NOT REALIZE THAT EARLIER?

- *Decentralized* servers
  - OSS, corporate, private, etc
- *Central cache* on local PC
  - sharing of the dependencies data among different projects

WHAT I WANT NOW AND I DID NOT REALIZE THAT EARLIER?

- *Decentralized* servers
  - OSS, corporate, private, etc
- *Central cache* on local PC
  - sharing of the dependencies data among different projects
- Support for
  - *sources only* (do not use prebuilt binaries even if available)
  - *sources + prebuilt binaries* (improves build performance)
  - *binaries only* (for closed source projects and development tools)

WHAT I WANT NOW AND I DID NOT REALIZE THAT EARLIER?

- *Decentralized* servers
  - OSS, corporate, private, etc
- *Central cache* on local PC
  - sharing of the dependencies data among different projects
- Support for
  - *sources only* (do not use prebuilt binaries even if available)
  - *sources + prebuilt binaries* (improves build performance)
  - *binaries only* (for closed source projects and development tools)
- *Offline* use
  - so I can start a new project with popular dependencies in a plane flying to CppCon ;-)

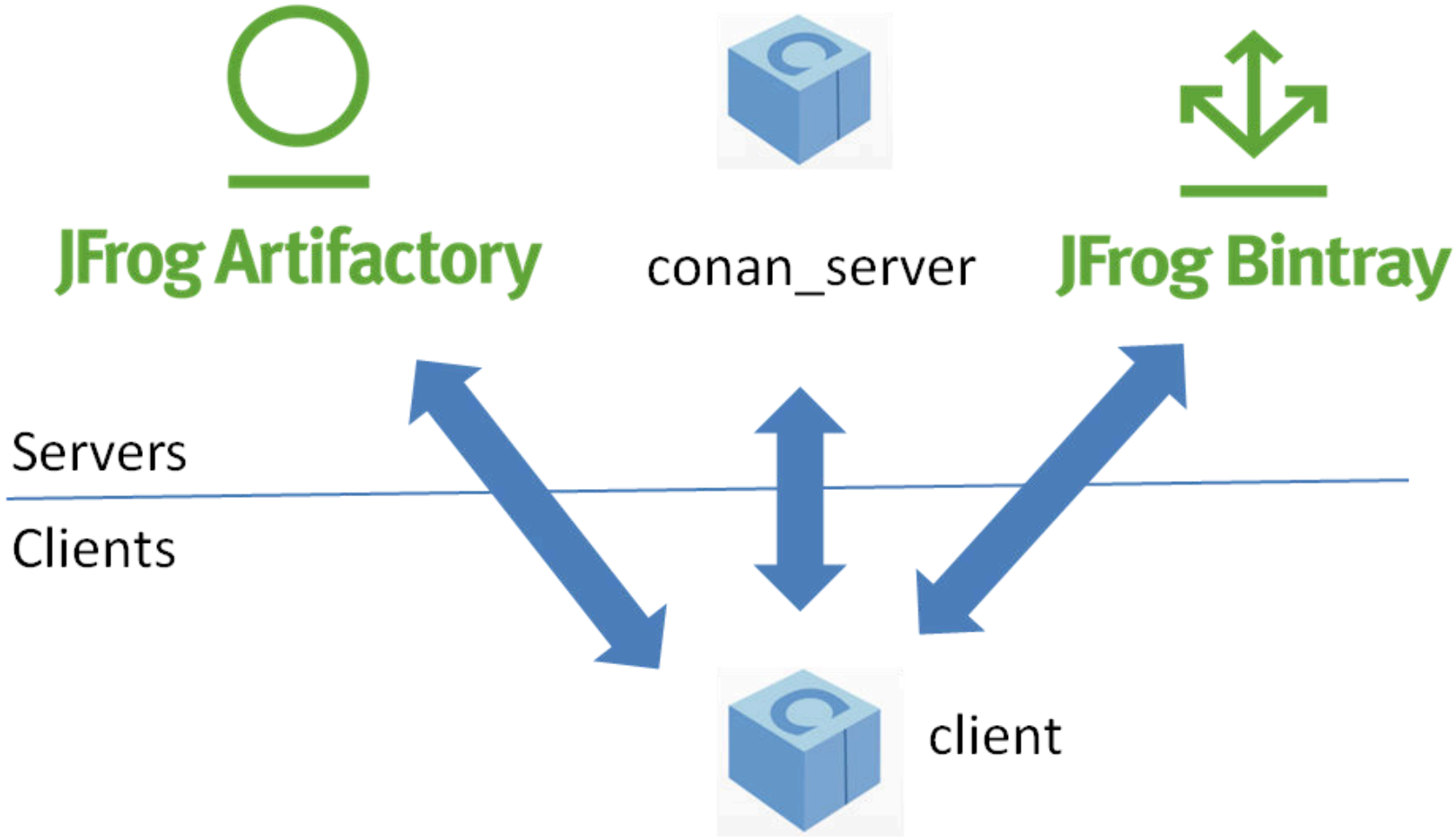
CONAN

- Conan is **OSS**, with a MIT license
- **Decentralized** package manager with
  a client-server architecture
- *The servers are just a 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 CLIENT-SERVER ARCHITECTURE

CONAN CLIENT

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

JFROG BINTRAY

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

CONAN CLIENT-SERVER ARCHITECTURE

CONAN CLIENT

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

JFROG BINTRAY

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

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 BINTRAY

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

JFROG ARTIFACTORY

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

POPULAR CONAN REMOTES ON BINTRAY

CONAN-CENTER

- *Moderated, curated and well-maintained packages*
- Place in which you can share your packages with the community

POPULAR CONAN REMOTES ON BINTRAY

CONAN-CENTER

- *Moderated, curated and well-maintained packages*
- Place in which you can share your packages with the community

BINCRAFTERS

- The Bincrafters team builds binary software packages for the OSS community
- Contains a wide and growing variety of Conan *packages from contributors*

POPULAR CONAN REMOTES ON BINTRAY

CONAN-CENTER

- *Moderated, curated and well-maintained packages*
- Place in which you can share your packages with the community

BINCRAFTERS

- The Bincrafters team builds binary software packages for the OSS community
- Contains a wide and growing variety of Conan *packages from contributors*

CONAN-COMMUNITY

- Created by Conan developers
- An *incubator* for maturing packages

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

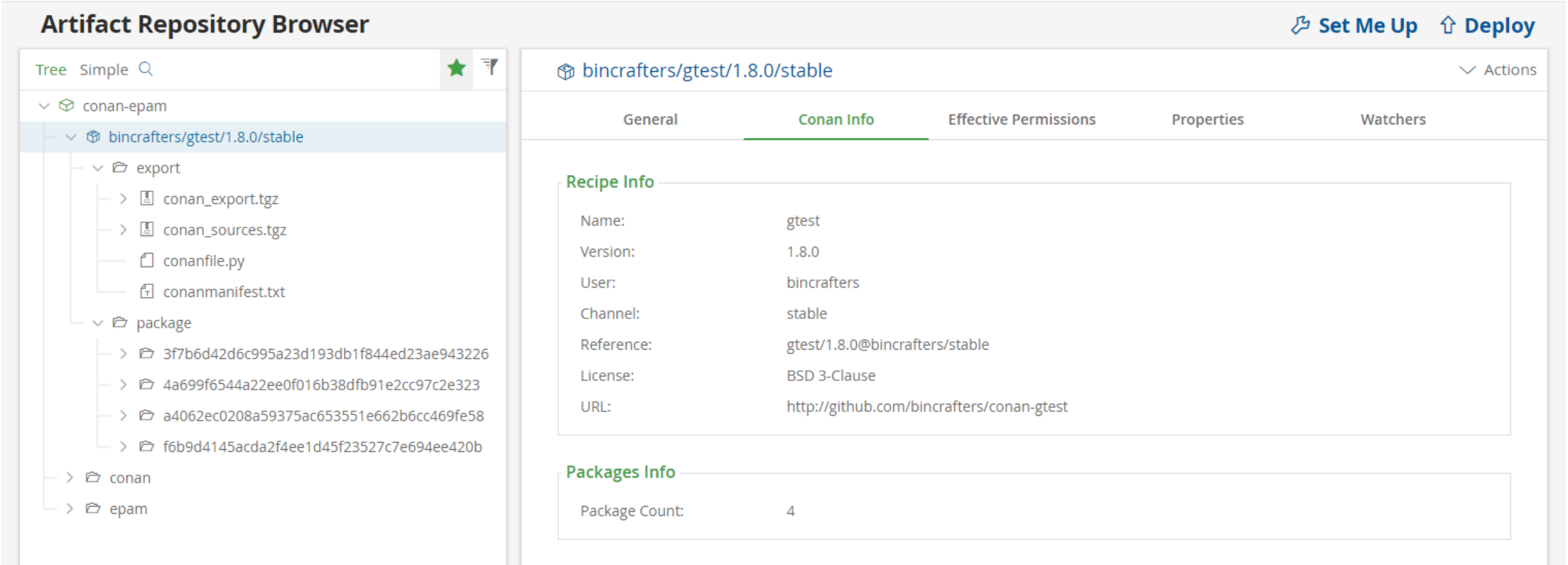
- Any string

CHANNEL

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

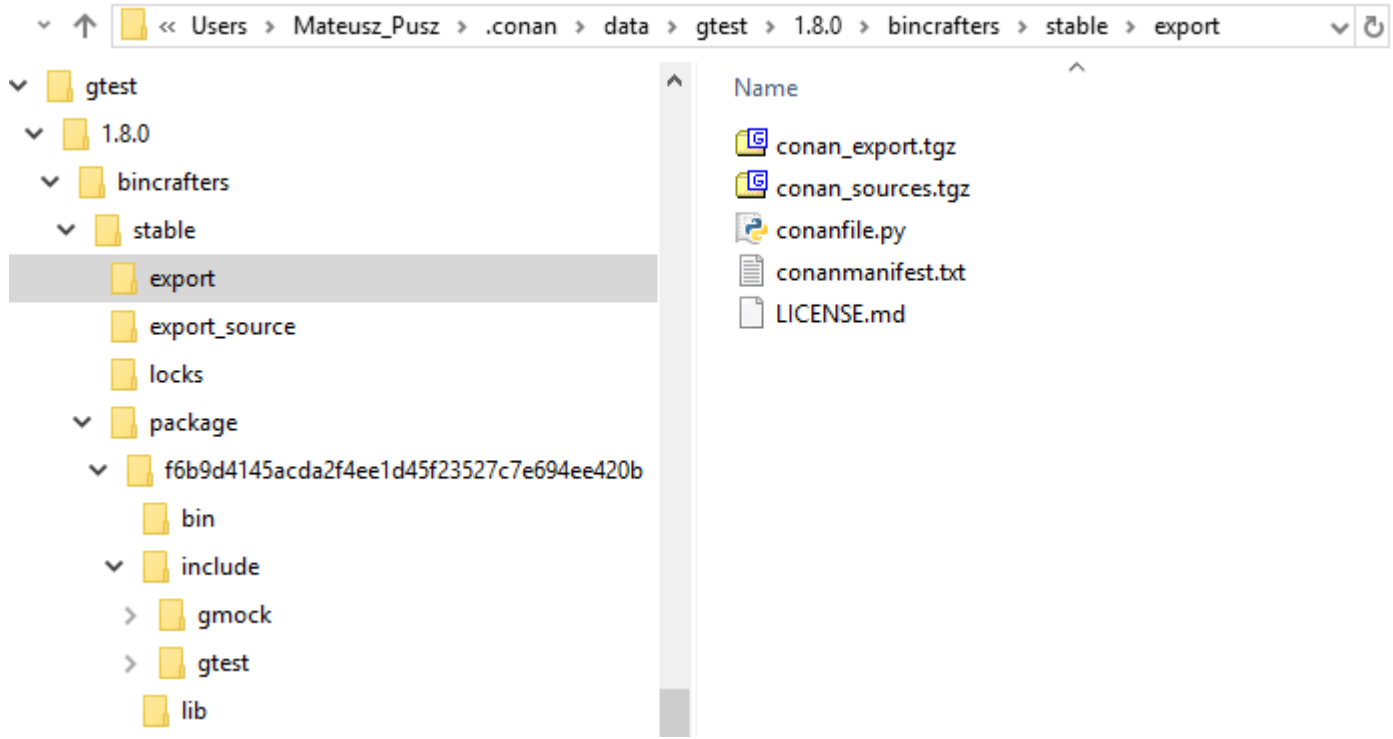
CONAN PACKAGES

- gtest/1.8.0@bincrafters/stable



CONAN PACKAGES

- gtest/1.8.0@bincrafters/stable



SEARCHING FOR PACKAGES

- *List* packages in the *local cache*

```
conan search
```

SEARCHING FOR PACKAGES

- *List* packages in the *local cache*

```
conan search
```

- *List* all **gtest** packages in the *conan-center*

```
conan search gtest -r conan-center
```

SEARCHING FOR PACKAGES

- *List* packages in the *local cache*

```
conan search
```

- *List* all **gtest** packages in the *conan-center*

```
conan search gtest -r conan-center
```

- Inspect *binary package details*

```
conan search gtest/1.8.1@bincrafters/stable
```

SEARCHING FOR PACKAGES

- *List* packages in the *local cache*

```
conan search
```

- *List* all **gtest** packages in the *conan-center*

```
conan search gtest -r conan-center
```

- Inspect *binary package details*

```
conan search gtest/1.8.1@bincrafters/stable
```

- Generate *table of binary packages* of **gtest/1.8.1@bincrafters/stable** on *conan-center*

```
conan search gtest/1.8.1@bincrafters/stable -r conan-center --table=conan_matrix.html
```

PACKAGE DETAILS

```
conan search gtest/1.8.1@bincrafters/stable
```

```
Existing packages for recipe gtest/1.8.1@bincrafters/stable:

    Package_ID: a4062ec0208a59375ac653551e662b6cc469fe58
        [options]
            build_gmock: True
            fPIC: True
            shared: False
        [settings]
            arch: x86_64
            build_type: Release
            compiler: gcc
            compiler.libcxx: libstdc++11
            compiler.version: 7
            os: Linux
        Outdated from recipe: False
```

INSPECTING DEPENDENCIES

- Inspect your current *project's dependencies*

```
conan info ..
```

INSPECTING DEPENDENCIES

- Inspect your current *project's dependencies*

```
conan info ..
```

- Inspect dependencies of ***OpenSSL/1.1.0i@conan/stable***

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

INSPECTING DEPENDENCIES

- Inspect your current *project's dependencies*

```
conan info ..
```

- Inspect dependencies of ***OpenSSL/1.1.0i@conan/stable***

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

- Generates a *graph of dependencies* of current project

```
conan info .. --graph=graph.html
```

PACKAGE DEPENDENCIES

```
OpenSSL/1.1.0i@conan/stable
    ID: 75b541b6f810787a4fcee8cec9cfdbf95959366f
    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
    Recipe: Downloaded
    Binary: Download
    Binary remote: conan-center
    Creation date: 2018-08-28 09:02:07
    Required by:
        PROJECT
    Requires:
        zlib/1.2.11@conan/stable
zlib/1.2.11@conan/stable
    ID: 6ae331b72e7e265ca2a3d1d8246faf73aa030238
    BuildID: None
    Remote: conan-center=https://conan.bintray.com
    URL: http://github.com/lasote/conan-zlib
    License: http://www.zlib.net/zlib_license.html
    Recipe: Cache
    Binary: Cache
    Binary remote: conan-center
    Creation date: 2018-08-30 15:46:32
    Required by:
        OpenSSL/1.1.0i@conan/stable
```

INSTALLING DEPENDENCIES

- Installing specific package **by hand**

```
conan install gtest/1.8.1@bincrafters/stable -g cmake
```

INSTALLING DEPENDENCIES

- Installing specific package **by hand**

```
conan install gtest/1.8.1@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/1.8.1@bincrafters/stable -g cmake
```

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

```
conan install ..
```

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

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

INSTALLING DEPENDENCIES

- Installing specific package **by hand**

```
conan install gtest/1.8.1@bincrafters/stable -g cmake
```

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

```
conan install ..
```

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

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

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

```
conan install .. -b missing
```

CONAN FILES

CONANFILE.TXT

```
[requires]
Foo/1.0@my_channel/stable
gtest/1.8.1@bincrafters/stable

[options]

[generators]
cmake
```

- **cmake generator** creates **conanbuildinfo.cmake** file that defines
  - CMake variables (module paths include paths, library names, ...)
  - CMake 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/1.8.1@bincrafters/stable

[options]

[generators]
cmake
```

CONANFILE.PY

```
from conans import ConanFile

class MyLibraryConan(ConanFile):
    requires = (
        "Foo/1.0@my_channel/stable",
        "gtest/1.8.1@bincrafters/stable"
    )
    generators = "cmake"
```

- **cmake generator** creates **`conanbuildinfo.cmake`** file that defines
  - CMake variables (module paths include paths, library names, ...)
  - CMake 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/1.8.1@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/1.8.1@bincrafters/stable"
    )
    default_options = "gtest:shared=True"
    generators = "cmake"
```

SETTING PACKAGE OPTIONS

CONANFILE.TXT

```
[requires]
Foo/1.0@my_channel/stable
gtest/1.8.1@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/1.8.1@bincrafters/stable"
    )
    default_options = "gtest:shared=True"
    generators = "cmake"
```

- Command line override

```
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/1.8.1@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/1.8.1@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/1.8.1@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/1.8.1@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/1.8.1@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/1.8.1@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
```

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

CONAN PROFILES

```
conan profile show clang6
```

```
Configuration for profile clang6:

[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=clang
compiler.version=6.0
compiler.libcxx=libstdc++11
build_type=Release
[options]
[build_requires]
[env]
CC=/usr/bin/clang
CXX=/usr/bin/clang++
```

CONAN PROFILES

```
conan profile show clang6
```

```
Configuration for profile clang6:

[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=clang
compiler.version=6.0
compiler.libcxx=libstdc++11
build_type=Release
[options]
[build_requires]
[env]
CC=/usr/bin/clang
CXX=/usr/bin/clang++
```

- Easy to override or extend the profile

```
conan install .. -pr clang6 -s build_type=Debug
```

CONAN PROFILES

- Example of a Conan *profile file distributed with the project source code*

DEBUG_SHARED

```
include(default)

[settings]
build_type=Debug

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

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

CONAN PACKAGES

1. The package recipe *independent* of the project source code repository
   - packaging 3rd party libraries

CONAN PACKAGES

1. The package recipe *independent* of the project source code repository
2. The package recipe *inside* the project source code repository
   - packaging own library
   - 2 approaches
     - exporting *snapshot of the source code together* with the recipe
     - exporting *only the recipe* and obtaining the source code from the project repository

CONAN PACKAGES

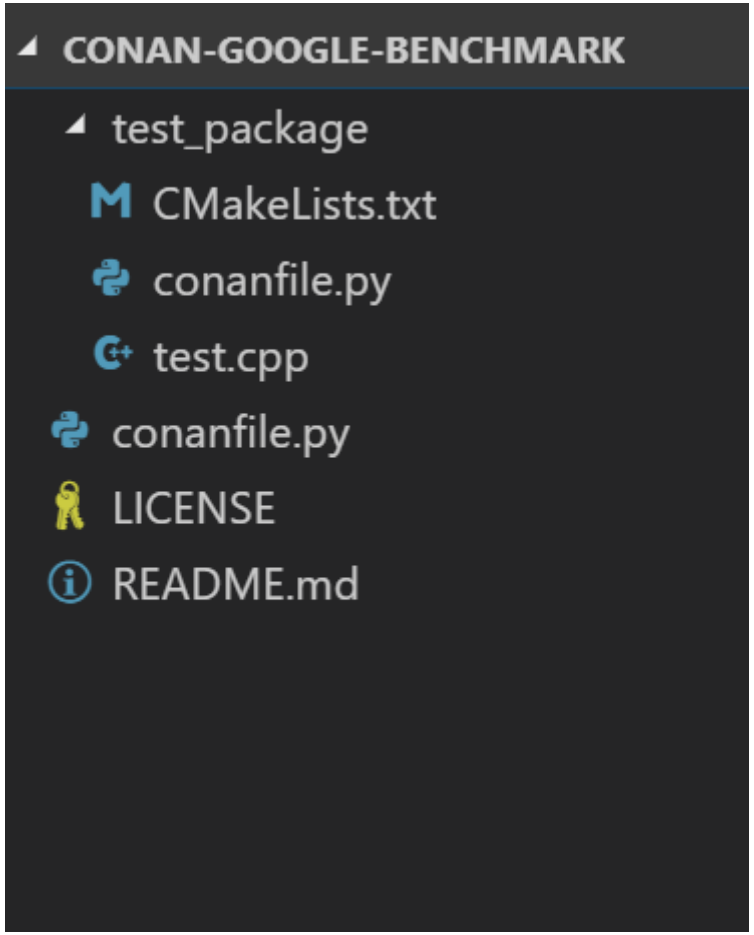
1. The package recipe *independent* of the project source code repository
2. The package recipe *inside* the project source code repository
3. Packaging *existing binaries*
   - only prebuilt 3rd party binaries are available
   - artifacts already built and no need to rebuild

CONAN PACKAGES

1. The package recipe *independent* of the project source code repository
2. The package recipe *inside* the project source code repository
3. Packaging *existing binaries*
4. Packaging of the *development tools*
   - distribution of the development tools needed to build the project (i.e. CMake, nasm)

TYPICAL CONAN PACKAGE CONTENTS

- **`./conanfile.py`**
  - main recipe file used to create a package
- **`./LICENSE`**
  - licence of the Conan package source code
  - *not to be confused with a license of a packaged project*
- **`./test_package`**
  - simple package consumer project
- **`./test_package/conanfile.py`**
  - recipe of a testing project



PACKAGE RECIPE ATTRIBUTES

```
class GoogleBenchmarkConan(ConanFile):
    name = "google-benchmark"
    version = "1.4.1"
    description = "A microbenchmark support library"









    # ...
```

- Basic package information

PACKAGE RECIPE ATTRIBUTES

```
class GoogleBenchmarkConan(ConanFile):
    name = "google-benchmark"
    version = "1.4.1"
    description = "A microbenchmark support library"
    license = "https://github.com/google/benchmark/blob/master/LICENSE"

    # ...
```

- The license of *the packaged project*

PACKAGE RECIPE ATTRIBUTES

```
class GoogleBenchmarkConan(ConanFile):
    name = "google-benchmark"
    version = "1.4.1"
    description = "A microbenchmark support library"
    license = "https://github.com/google/benchmark/blob/master/LICENSE"
    url = "https://github.com/mpusz/conan-google-benchmark"









    # ...
```

- Link to *the Conan package repository* (not the packaged project)

PACKAGE RECIPE ATTRIBUTES

```
class GoogleBenchmarkConan(ConanFile):
    name = "google-benchmark"
    version = "1.4.1"
    description = "A microbenchmark support library"
    license = "https://github.com/google/benchmark/blob/master/LICENSE"
    url = "https://github.com/mpusz/conan-google-benchmark"
    exports = ["LICENSE"]

    # ...
```

- Copies a *packaging code license* to be stored within a package

PACKAGE RECIPE ATTRIBUTES

```
class GoogleBenchmarkConan(ConanFile):
    name = "google-benchmark"
    version = "1.4.1"
    description = "A microbenchmark support library"
    license = "https://github.com/google/benchmark/blob/master/LICENSE"
    url = "https://github.com/mpusz/conan-google-benchmark"
    exports = ["LICENSE"]
    settings = "os", "arch", "compiler", "build_type"

    # ...
```

- The *configuration* of the different binary packages
- *Any change in those parameters will generate* a different binary package

PACKAGE RECIPE ATTRIBUTES

```
class GoogleBenchmarkConan(ConanFile):
    name = "google-benchmark"
    version = "1.4.1"
    description = "A microbenchmark support library"
    license = "https://github.com/google/benchmark/blob/master/LICENSE"
    url = "https://github.com/mpusz/conan-google-benchmark"
    exports = ["LICENSE"]
    settings = "os", "arch", "compiler", "build_type"
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    default_options = ("shared=False", "exceptions=True", "lto=False")




    # ...
```

- Package *options* and their *default values*
- *Any change in those parameters will generate* a different binary package

PACKAGE RECIPE ATTRIBUTES

```
class GoogleBenchmarkConan(ConanFile):
    name = "google-benchmark"
    version = "1.4.1"
    description = "A microbenchmark support library"
    license = "https://github.com/google/benchmark/blob/master/LICENSE"
    url = "https://github.com/mpusz/conan-google-benchmark"
    exports = ["LICENSE"]
    settings = "os", "arch", "compiler", "build_type"
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    default_options = ("shared=False", "exceptions=True", "lto=False")
    scm = {
        "type": "git",
        "url": "https://github.com/google/benchmark.git",
        "revision": "v%s" % version
    }

    # ...
```

- Packaged project *repository information*
- Used to *obtain project source code* to build and pack

SETTINGS VS OPTIONS

SETTINGS

- Project-wide and *common to all the dependencies* in order for them to link together without issues with the project being built
- *Provided by the environment* and *cannot be defaulted* in the recipe

SETTINGS VS OPTIONS

SETTINGS

- Project-wide and *common to all the dependencies* in order for them to link together without issues with the project being built
- *Provided by the environment* and *cannot be defaulted* in the recipe

OPTIONS

- *Package-specific configuration* and each dependency can provide different set of options to fulfill their configuration needs
- *Default option values* provided in the recipe file *can be overridden* by the user

OTHER USEFUL ATTRIBUTES

- `generators` - not needed only if a package does not have any dependencies

OTHER USEFUL ATTRIBUTES

- **`generators`** - not needed only if a package does not have any dependencies
- **`requires`** - package *dependencies*

```
class MyLibConan(ConanFile):
    requires = (("Hello/[>1.0,<2.0]@user/testing"),
                ("Bye/2.1@coder/beta", "private"),
                ("Say/0.2@dummy/stable", "override"))
```

  - **`private`** - not exposed to clients, *non-transitive*
  - **`override`** - *does not introduce a new dependency*, overrides a version if it is a dependency already

OTHER USEFUL ATTRIBUTES

- **`generators`** - not needed only if a package does not have any dependencies
- **`requires`** - package *dependencies*

```
class MyLibConan(ConanFile):
    requires = (("Hello/[>1.0,<2.0]@user/testing"),
                ("Bye/2.1@coder/beta", "private"),
                ("Say/0.2@dummy/stable", "override"))
```

  - **`private`** - not exposed to clients, *non-transitive*
  - **`override`** - *does not introduce a new dependency*, overrides a version if it is a dependency already
  - *version ranges*

```
>1.1,<2.1     # in such range
2.8           # equivalent to =2.8
~=3.0         # compatible, according to semver
>1.1 || 0.8   # conditions can be OR'ed
```

OTHER USEFUL ATTRIBUTES

- `build_requires` - build-time only dependencies
  - *only needed to build a package from sources*
  - dev tools, compilers, build systems, code analyzers, testing libraries, etc.
  - **non-transitive**

OTHER USEFUL ATTRIBUTES

- `build_requires` - build-time only dependencies
  - *only needed to build a package from sources*
  - dev tools, compilers, build systems, code analyzers, testing libraries, etc.
  - **non-transitive**
- `exports` - *copies provided files* to be stored in a package along with the recipe

OTHER USEFUL ATTRIBUTES

- **`build_requires`** - build-time only dependencies
  - *only needed to build a package from sources*
  - dev tools, compilers, build systems, code analyzers, testing libraries, etc.
  - **non-transitive**
- **`exports`** - *copies provided files* to be stored in a package along with the recipe
- **`exports_sources`** - creates a *snapshot of the packaged project source code* in the package

OTHER USEFUL ATTRIBUTES

- **`build_requires`** - build-time only dependencies
  - *only needed to build a package from sources*
  - dev tools, compilers, build systems, code analyzers, testing libraries, etc.
  - **non-transitive**
- **`exports`** - *copies provided files* to be stored in a package along with the recipe
- **`exports_sources`** - creates a *snapshot of the packaged project source code* in the package
- **`homepage`** - the home *web page of the library being packaged*

PACKAGE RECIPE METHODS

```python
class GoogleBenchmarkConan(ConanFile):
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    # ...

    def _configure_cmake(self):
        cmake = CMake(self)
        cmake.definitions["BENCHMARK_ENABLE_EXCEPTIONS"] = "ON" if self.options.exceptions else "OFF"
        cmake.definitions["BENCHMARK_ENABLE_LTO"] = "ON" if self.options.lto else "OFF"
        cmake.definitions["BENCHMARK_ENABLE_TESTING"] = "OFF"
        cmake.configure()
        return cmake

    # ...
```

- *Helper method* to configure CMake

PACKAGE RECIPE METHODS

```
class GoogleBenchmarkConan(ConanFile):
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    # ...

    def _configure_cmake(self):
        cmake = CMake(self)
        cmake.definitions["BENCHMARK_ENABLE_EXCEPTIONS"] = "ON" if self.options.exceptions else "OFF"
        cmake.definitions["BENCHMARK_ENABLE_LTO"] = "ON" if self.options.lto else "OFF"
        cmake.definitions["BENCHMARK_ENABLE_TESTING"] = "OFF"
        cmake.configure()
        return cmake

    # ...
```

- CMake helper *automatically appends* some definitions based on popular settings

PACKAGE RECIPE METHODS

```python
class GoogleBenchmarkConan(ConanFile):
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    # ...

    def _configure_cmake(self):
        cmake = CMake(self)
        cmake.definitions["BENCHMARK_ENABLE_EXCEPTIONS"] = "ON" if self.options.exceptions else "OFF"
        cmake.definitions["BENCHMARK_ENABLE_LTO"] = "ON" if self.options.lto else "OFF"
        cmake.definitions["BENCHMARK_ENABLE_TESTING"] = "OFF"
        cmake.configure()
        return cmake

    # ...
```

- Compiling unit tests makes the *packaging longer* and *does not influence the binary* package content

PACKAGE RECIPE METHODS

```
class GoogleBenchmarkConan(ConanFile):
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    # ...

    def _configure_cmake(self):
        # ...

    def build(self):
        cmake = self._configure_cmake()
        cmake.build()









    # ...
```

- *Configures and builds* the project

PACKAGE RECIPE METHODS

```python
class GoogleBenchmarkConan(ConanFile):
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    # ...

    def _configure_cmake(self):
        # ...

    def build(self):
        cmake = self._configure_cmake()
        cmake.build()

    def package(self):
        cmake = self._configure_cmake()
        cmake.install()
        self.copy("license*", dst="licenses",  ignore_case=True, keep_path=False)



    # ...
```

- Copies *packaged project binaries*, *CMake configuration files*, and *its license* to the package

PACKAGE RECIPE METHODS

```
class GoogleBenchmarkConan(ConanFile):
    options = {
        "shared": [True, False],
        "exceptions": [True, False],
        "lto": [True, False],
    }
    # ...

    def _configure_cmake(self):
        # ...

    def build(self):
        cmake = self._configure_cmake()
        cmake.build()

    def package(self):
        cmake = self._configure_cmake()
        cmake.install()
        self.copy("license*", dst="licenses",  ignore_case=True, keep_path=False)

    def package_info(self):
        self.cpp_info.libs = ["benchmark"]
    # ...
```

- Defines *the target that the consumer must link with* when using this package

OTHER USEFUL METHODS

- `source()` - *retrieves the source code* from external origin (git, **`*.tgz`**, ...)

OTHER USEFUL METHODS

- `source()` - *retrieves the source code* from external origin (git, **`*.tgz`**, ...)
- `requirements()` - defines more *advanced requirements logic*

OTHER USEFUL METHODS

- `source()` - *retrieves the source code* from external origin (git, `*.tgz`, ...)
- `requirements()` - defines more *advanced requirements logic*
- `build_requirements()` - defines dependencies used when the package is *built from sources*

OTHER USEFUL METHODS

- `source()` - *retrieves the source code* from external origin (git, `*.tgz`, ...)
- `requirements()` - defines more *advanced requirements logic*
- `build_requirements()` - defines dependencies used when the package is *built from sources*
- `imports()` - copies files *from the local store to client's project*

OTHER USEFUL METHODS

- `source()` - *retrieves the source code* from external origin (git, `*.tgz`, ...)
- `requirements()` - defines more *advanced requirements logic*
- `build_requirements()` - defines dependencies used when the package is *built from sources*
- `imports()` - copies files *from the local store to client's project*
- `package_id()` - creates a *unique ID* for the package

```
def package_id(self):
    self.info.header_only()
```

OTHER USEFUL METHODS

- `source()` - *retrieves the source code* from external origin (git, `*.tgz`, ...)
- `requirements()` - defines more *advanced requirements logic*
- `build_requirements()` - defines dependencies used when the package is *built from sources*
- `imports()` - copies files *from the local store to client's project*
- `package_id()` - creates a *unique ID* for the package

```
def package_id(self):
    self.info.header_only()
```

- `configure()` - configures or constrains/validates the available options in a package

OTHER USEFUL METHODS

- `source()` - *retrieves the source code* from external origin (git, `*.tgz`, ...)
- `requirements()` - defines more *advanced requirements logic*
- `build_requirements()` - defines dependencies used when the package is *built from sources*
- `imports()` - copies files *from the local store to client's project*
- `package_id()` - creates a *unique ID* for the package

```
def package_id(self):
    self.info.header_only()
```

- `configure()` - configures or constrains/validates the available options in a package
- `config_options()` - constrains the available options in a package, before they are given a value

```
def config_options(self):
    if self.settings.os == 'Windows':
        del self.options.fPIC
```

THE `test_package` DIRECTORY

- **`test_package`** *differs from the library unit or integration tests* (which should be more comprehensive)
- *Packaging tests* to verify that
  - the package is *properly created*
  - the package consumers *will be able to link against it* and reuse it

THE test_package RECIPE

```
from conans import ConanFile, CMake

class GoogleBenchmarkTestConan(ConanFile):
    settings = "os", "arch", "compiler", "build_type"
    generators = "cmake_paths"

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

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

    def test(self):
        cmake = CMake(self)
        self.run("ctest -VV -C %s" % cmake.build_type)
```

THE test_package RECIPE

```
from conans import ConanFile, CMake

class GoogleBenchmarkTestConan(ConanFile):
    settings = "os", "arch", "compiler", "build_type"
    generators = "cmake_paths"

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

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

    def test(self):
        cmake = CMake(self)
        self.run("ctest -VV -C %s" % cmake.build_type)
```

- **`requires`** attribute *skipped* as all needed information (e.g. project version) is automatically *injected by Conan* during the package creation

CONAN PACKAGE CREATION

```
conan create . google-benchark/testing
```

CONAN PACKAGE CREATION

```
conan create . google-benchark/testing
```

- *Exports* the **conanfile.py** and files specified by the **exports_sources** field *into the local cache*

CONAN PACKAGE CREATION

```
conan create . google-benchark/testing
```

- *Exports* the **conanfile.py** and files specified by the **exports_sources** field *into the local cache*
- *Installs* the package, forcing it to be *built from the sources*

CONAN PACKAGE CREATION

```
conan create . google-benchark/testing
```

- *Exports* the **`conanfile.py`** and files specified by the **`exports_sources`** field *into the local cache*
- *Installs* the package, forcing it to be *built from the sources*
- Creates a temporary ***`./test_package/build`*** *directory*

CONAN PACKAGE CREATION

```
conan create . google-benchark/testing
```

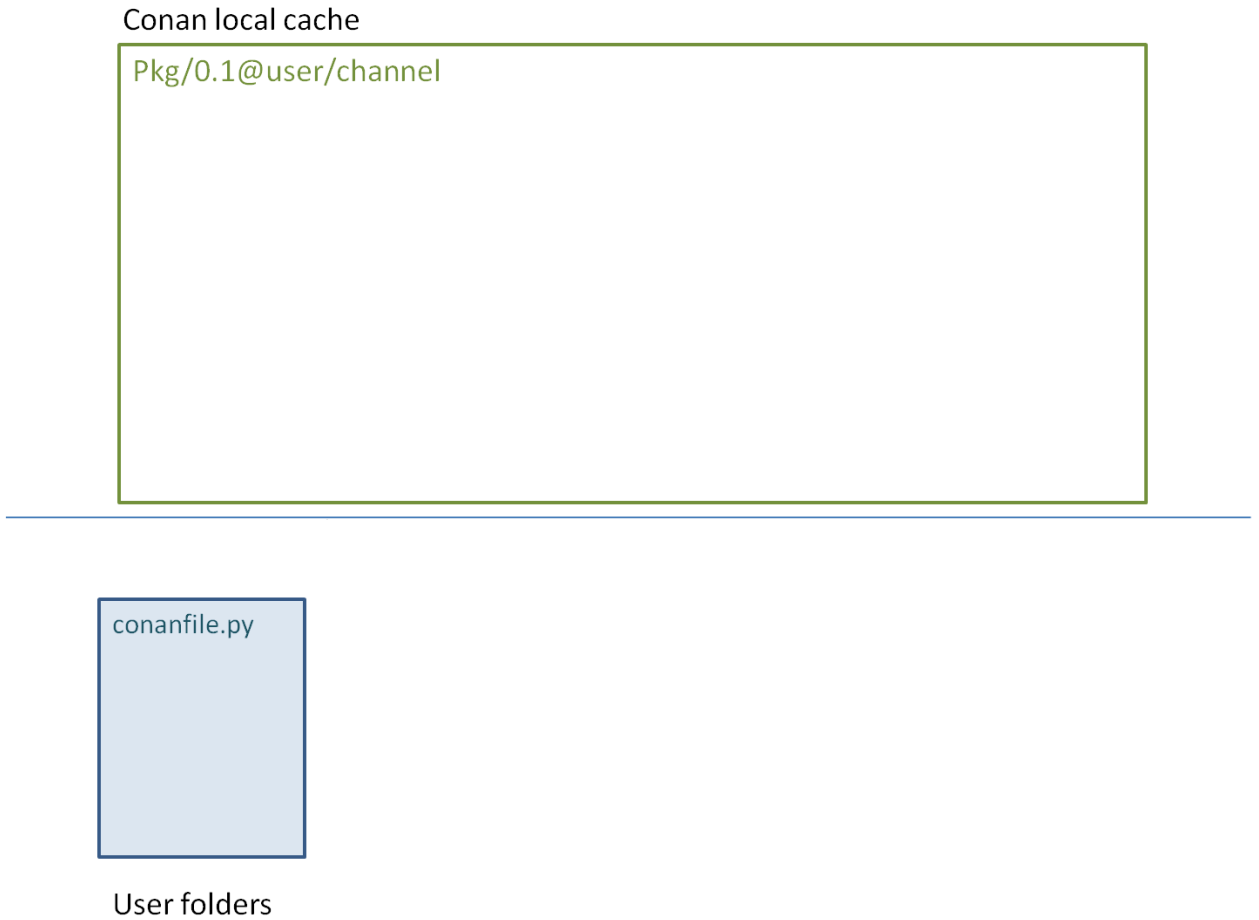
- *Exports* the **conanfile.py** and files specified by the **exports_sources** field *into the local cache*
- *Installs* the package, forcing it to be *built from the sources*
- Creates a temporary ***./test_package/build*** *directory*
- Executes the ***conan install ..***, to install the requirements of the **./test_package/conanfile.py** and the packaged project

CONAN PACKAGE CREATION

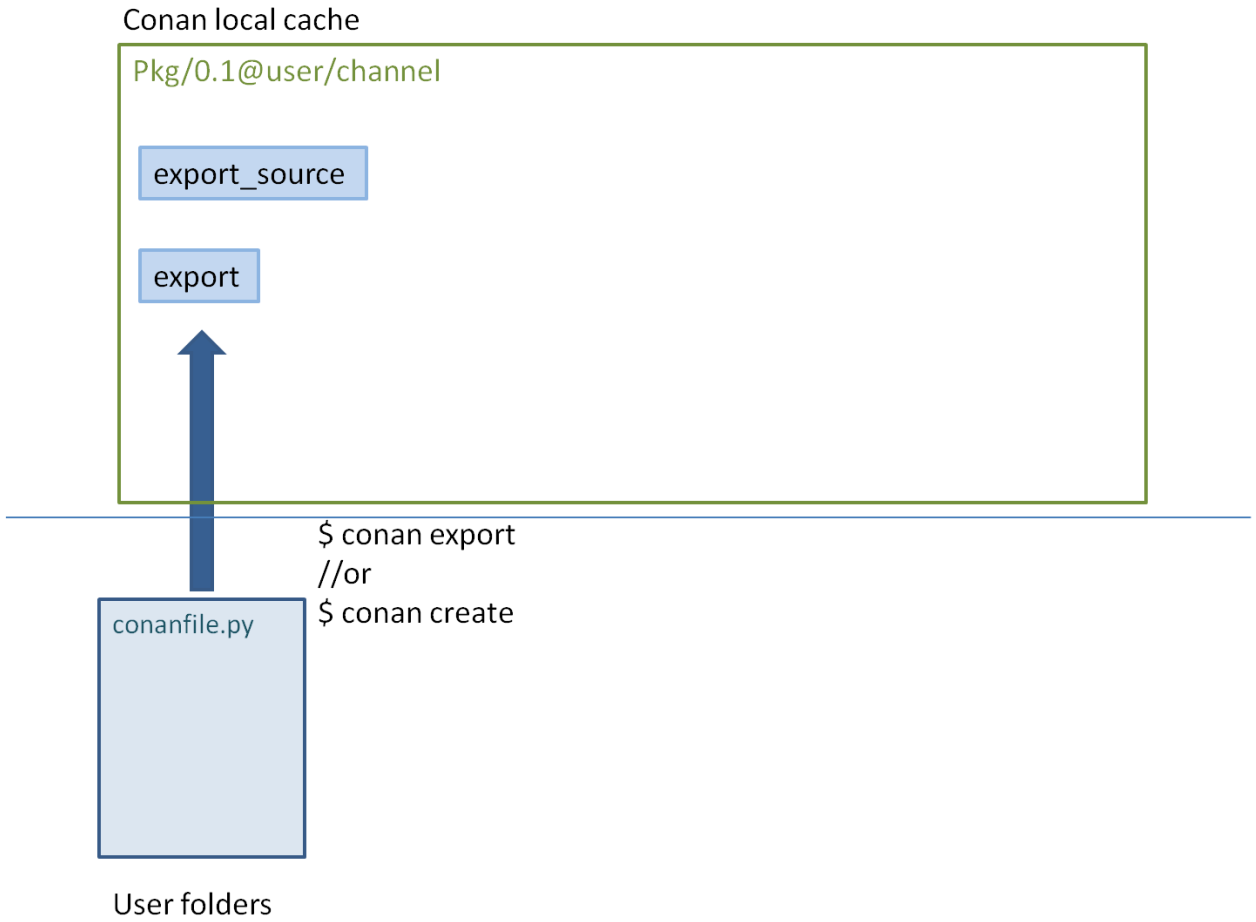
```
conan create . google-benchark/testing
```

- *Exports* the **conanfile.py** and files specified by the **exports_sources** field *into the local cache*
- *Installs* the package, forcing it to be *built from the sources*
- Creates a temporary *./test_package/build directory*
- Executes the *conan install ..*, to install the requirements of the **./test_package/conanfile.py** and the packaged project
- *Builds and launches the example consuming application*, calling the **./test_package/conanfile.py** **build()** and **test()** methods

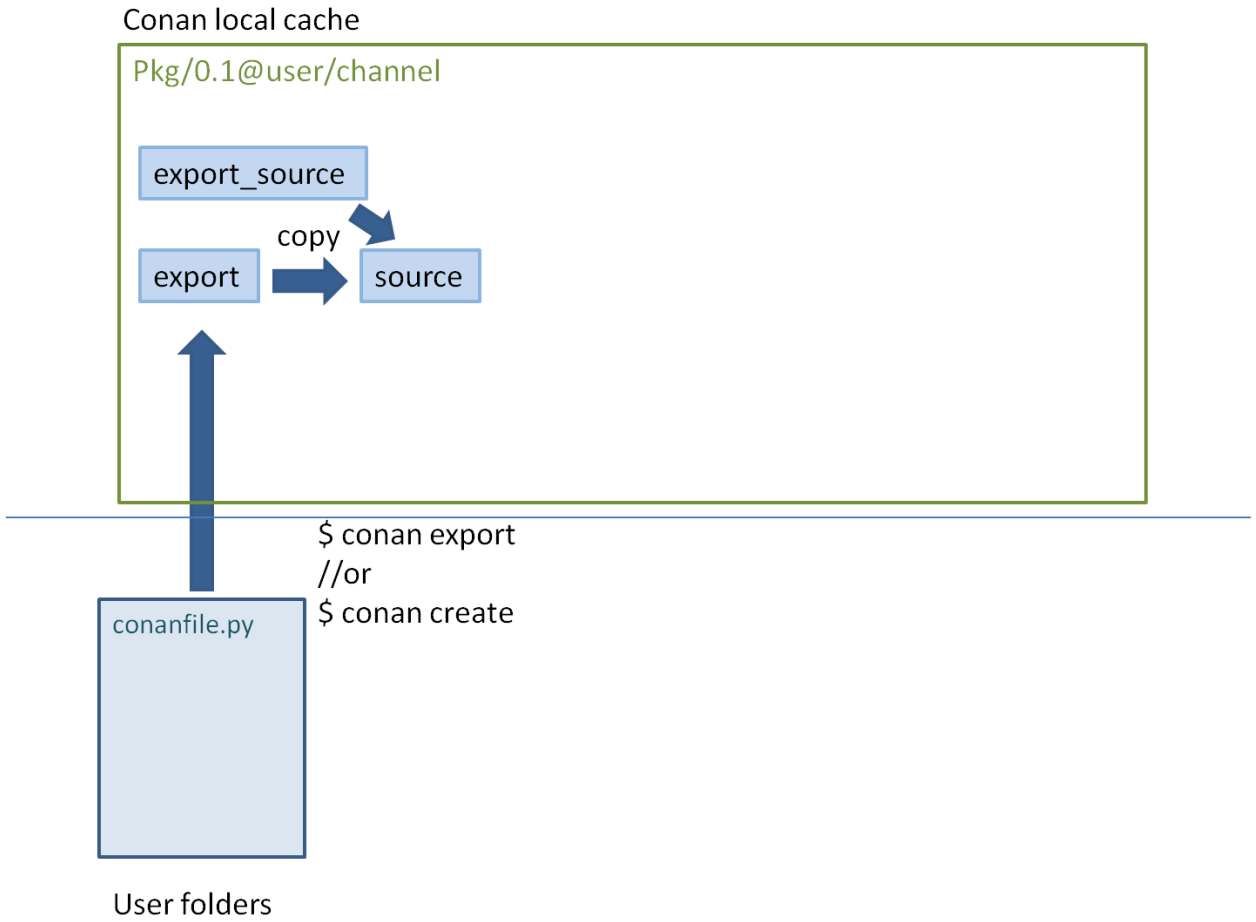
PACKAGING STEPS



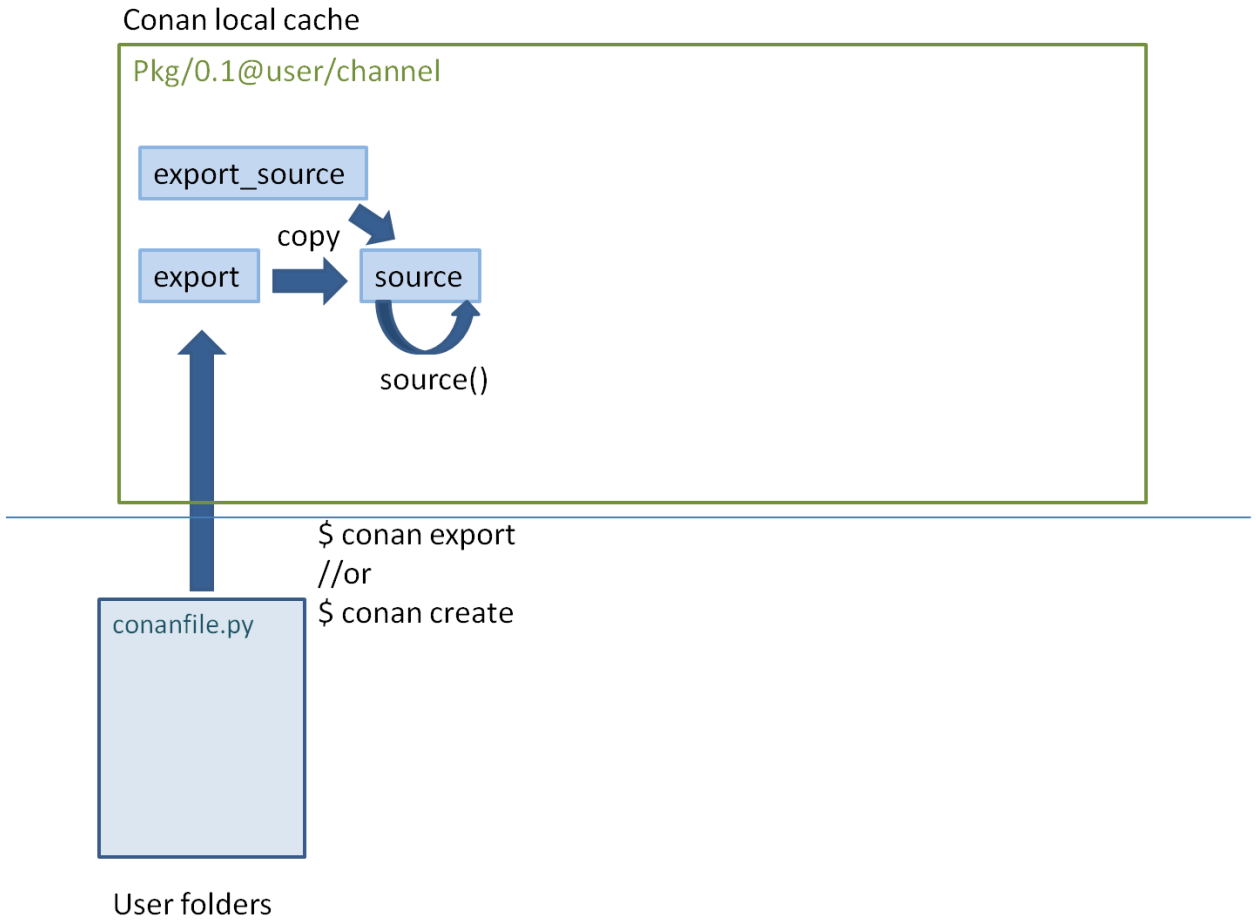
PACKAGING STEPS



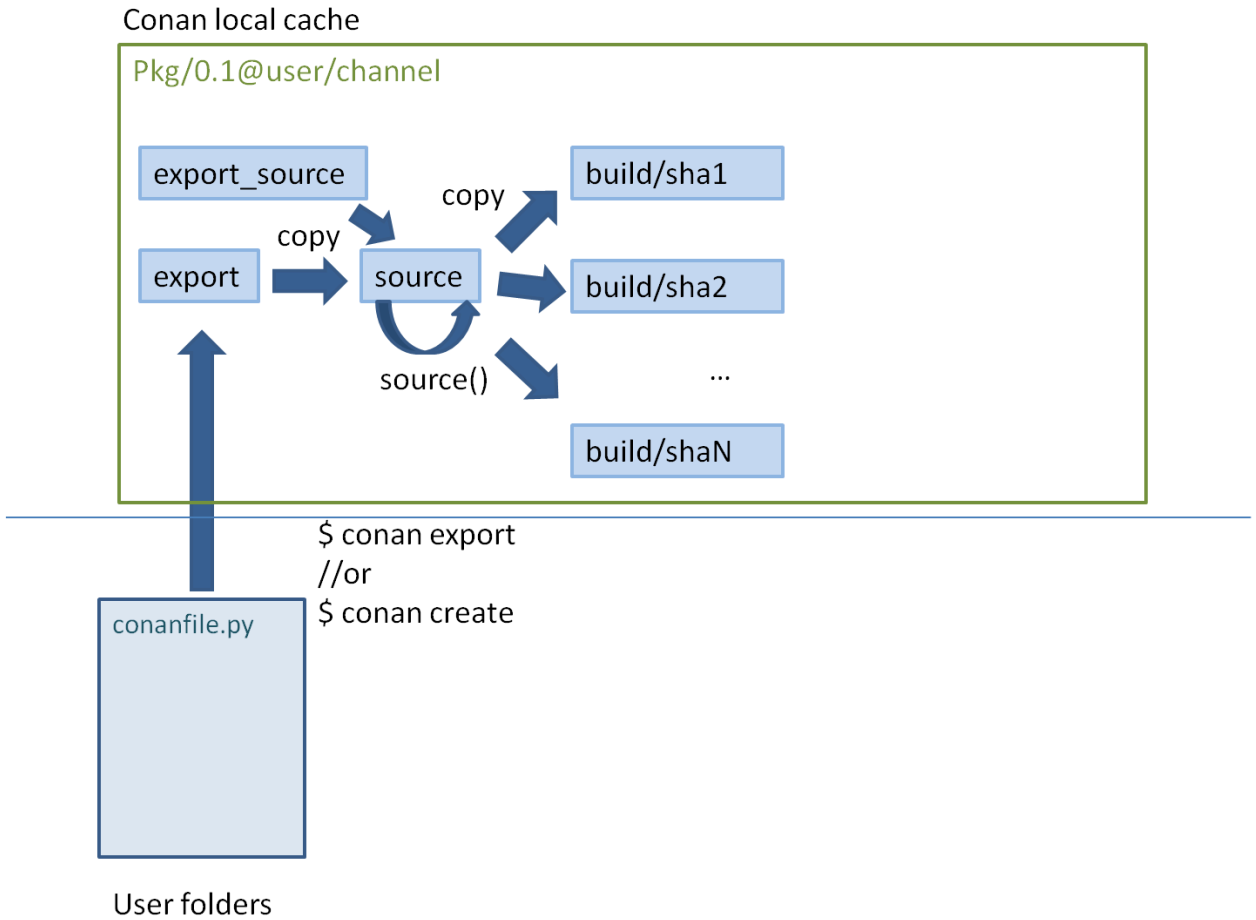
PACKAGING STEPS



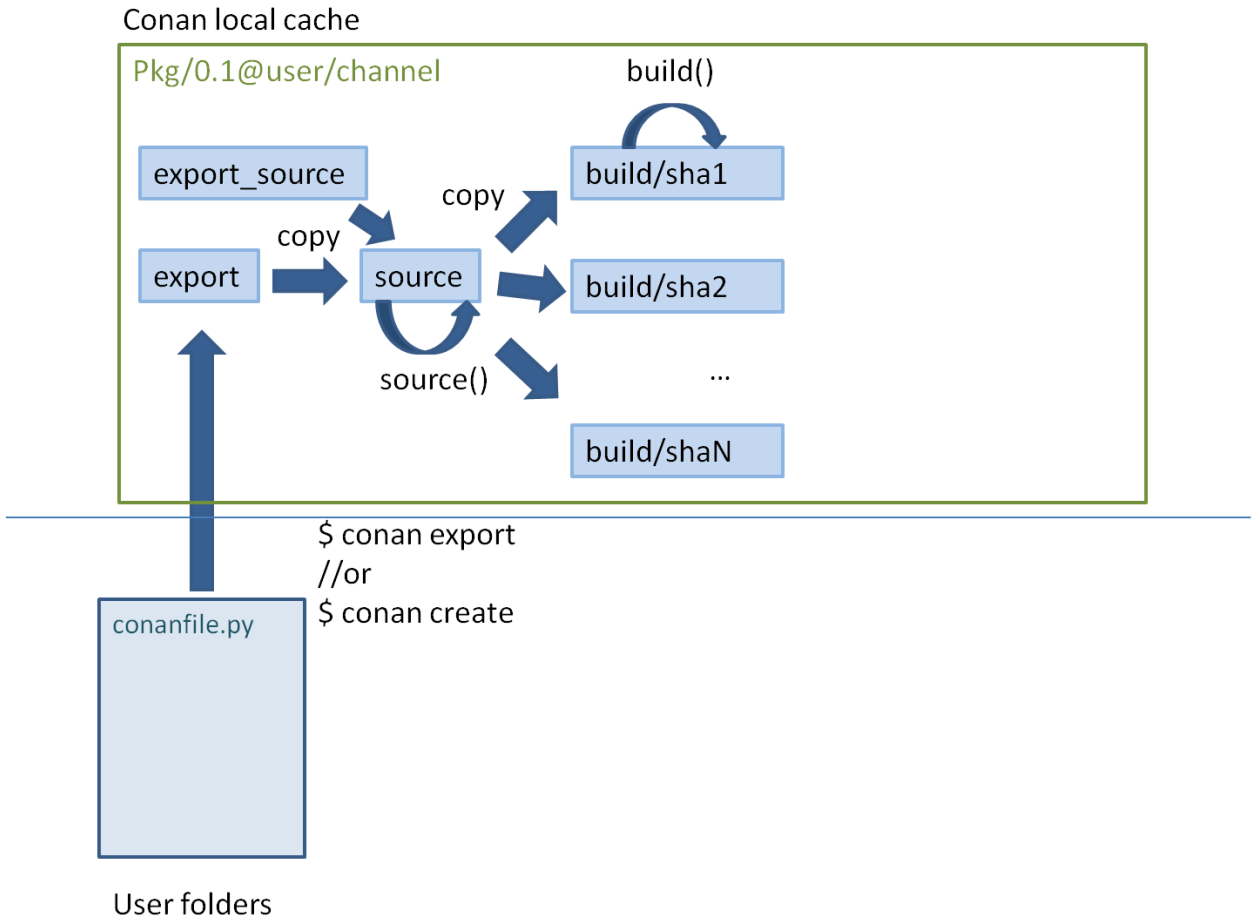
PACKAGING STEPS



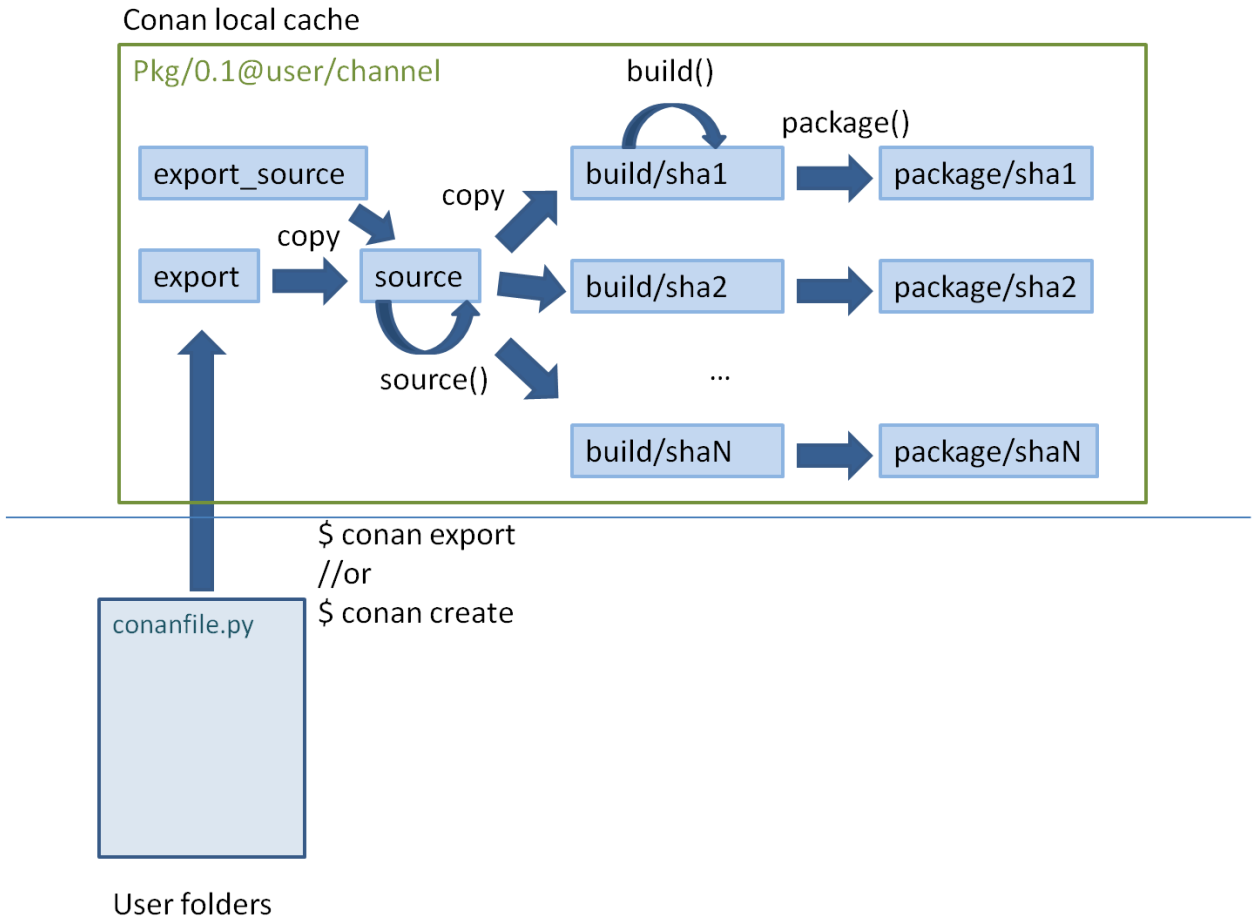
PACKAGING STEPS



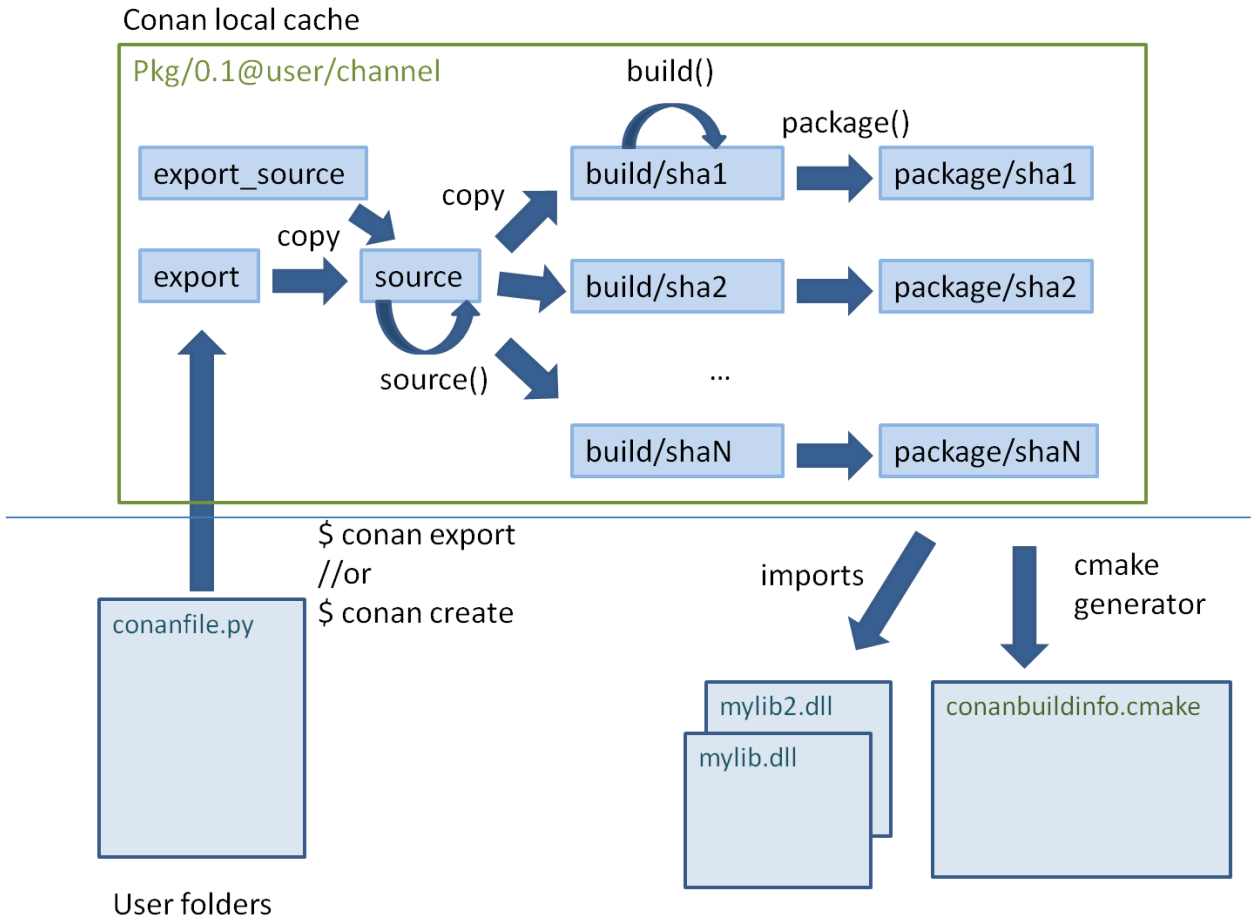
PACKAGING STEPS



PACKAGING STEPS



PACKAGING STEPS



CONAN PACKAGE DEVELOPMENT FLOW: OUT-OF-SOURCE

```
conan create . user/channel
```

CONAN PACKAGE DEVELOPMENT FLOW: OUT-OF-SOURCE

```
conan create . user/channel
```

```
conan create . user/channel --keep-source
```

CONAN PACKAGE DEVELOPMENT FLOW: OUT-OF-SOURCE

```
conan create . user/channel
```

```
conan create . user/channel --keep-source
```

```
conan create . user/channel --keep-build
```

CONAN PACKAGE DEVELOPMENT FLOW: OUT-OF-SOURCE

```
conan create . user/channel
```

```
conan create . user/channel --keep-source
```

```
conan create . user/channel --keep-build
```

```
conan test test_package MyLibrary/0.1@user/channel
```

CONAN PACKAGE DEVELOPMENT FLOW: IN-SOURCE

```
conan source . --source-folder=tmp/source
```

```
conan install . --install-folder=tmp/build [--profile XXXX]
```

```
conan build . --source-folder=tmp/source --build-folder=tmp/build
```

```
conan package . --source-folder=tmp/source --build-folder=tmp/build --package-folder=tmp/package
```

```
conan export-pkg . user/testing --package-folder=tmp/package
```

```
conan test test_package MyLibrary/0.1@user/channel
```

CONAN PACKAGE DEVELOPMENT FLOW: IN-SOURCE

```
conan source . --source-folder=tmp/source
```

```
conan install . --install-folder=tmp/build [--profile XXXX]
```

```
conan build . --source-folder=tmp/source --build-folder=tmp/build
```

```
conan package . --source-folder=tmp/source --build-folder=tmp/build --package-folder=tmp/package
```

```
conan export-pkg . user/testing --package-folder=tmp/package
```

```
conan test test_package MyLibrary/0.1@user/channel
```

Even smaller steps are possible with **`--configure`**, **`--build`**, **`--install`** options of **`conan build`** command

PACKAGE QUALITY

- Recipe *should include* **`description`**, **`license`**, and **`url`** attributes
- Use only *lowercase letters* in package names
- *Include* **`test_package`**
- *Raise errors* on invalid configurations
- *Do not use* version ranges
- Distribute *packaged software license files* in binary packages
- Export *packaging code license*
- Use clean python code and the latest Conan features

UPLOADING A PACKAGE

- If not done already, *add the Conan remote*

```
conan remote add conan-mpusz https://bintray.com/mpusz/conan-mpusz
```

UPLOADING A PACKAGE

- If not done already, *add the Conan remote*

```
conan remote add conan-mpusz https://bintray.com/mpusz/conan-mpusz
```

- *Upload the package recipe and all binaries* to the Conan remote

```
conan upload google-benchmark/1.4.1@mpusz/stable -r conan_mpusz --all
```

USING CONAN WITH CMAKE: GENERATORS

1. `cmake`
2. `cmake_multi`
3. `cmake_paths`
4. `cmake_find_package`

cmake **GENERATOR**

- Creates **`conanbuildinfo.cmake`** file that defines CMake
  - *variables* (module paths include paths, library names, ...)
  - *helper functions* (defining targets, configuring CMake environment, ...)
- Changes to **`CMakeLists.txt`** file are required

```
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()

# ...
```

cmake_multi GENERATOR

- Creates **conanbuildinfo_multi.cmake** file
- Used with *multi-configuration* environments like Visual Studio and Xcode
- Does not configure for a specific **build_type**, like Debug or Release
- *Cannot be used to create packages and issues with **find_package()** usage*
- Changes to **CMakeLists.txt** file are required

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

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

# ...
```

cmake_paths **GENERATOR**

- Creates **conan_paths.cmake** file that defines
  - **CMAKE_MODULE_PATH** used by *find_package()* to find CMake configuration and **FindXXX.cmake** files
  - **CMAKE_PREFIX_PATH** used by *find_library()* to locate library files in packages

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

include(${CMAKE_BINARY_DIR}/conan_paths.cmake OPTIONAL)

# ...
```

cmake_paths **GENERATOR**

- Creates **conan_paths.cmake** file that defines
  - **CMAKE_MODULE_PATH** used by *find_package()* to find CMake configuration and **FindXXX.cmake** files
  - **CMAKE_PREFIX_PATH** used by *find_library()* to locate library files in packages

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

include(${CMAKE_BINARY_DIR}/conan_paths.cmake OPTIONAL)

# ...
```

- *Can work without doing any changes to CMake files*

```
cmake .. -DCMAKE_PROJECT_MyLibrary_INCLUDE=conan_paths.cmake -DCMAKE_BUILD_TYPE=Release
```

`cmake_find_package` **GENERATOR**

- Creates **`Find<package_name>.cmake`** file for each requirement specified in the conanfile
- When used with **`CMake`** helper in **`build()`** method of ***`conanfile.py`*** *no changes to CMake files needed*

cmake_find_package GENERATOR

- Creates **Find<package_name>.cmake** file for each requirement specified in the conanfile
- When used with **CMake** helper in **build()** method of ***conanfile.py*** *no changes to CMake files needed*
- When used with ***conanfile.txt***
  - can be used together with **cmake_paths** generator

```
[requires]
...

[generators]
cmake_find_package
cmake_paths
```

cmake_find_package **GENERATOR**

- Creates **Find<package_name>.cmake** file for each requirement specified in the conanfile
- When used with **CMake** helper in **build()** method of ***conanfile.py*** *no changes to CMake files needed*
- When used with ***conanfile.txt***
  - can be used together with **cmake_paths** generator

```
[requires]
...

[generators]
cmake_find_package
cmake_paths
```

  - **CMAKE_MODULE_PATH** can be updated manually

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

set(CMAKE_MODULE_PATH ${CMAKE_BINARY_DIR} ${CMAKE_MODULE_PATH})

# ...
```

BKM: AUTOMATION OF UPDATING CMakeLists.txt FILE

OPTION 1

```
class MyLibraryConan(ConanFile):
    # ...
    generators = "cmake"

    def source(self):
        # ...
        tools.replace_in_file("%s/CMakeLists.txt" % self.subfolder,
                              "project(MyLibrary)",
                              """project(MyLibrary)
include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup()""")
```

BKM: AUTOMATION OF UPDATING CMakeLists.txt FILE

OPTION 2

```
class MyLibraryConan(ConanFile):
    # ...
    exports_sources = ["CMakeLists.txt"]
    generators = "cmake"
    scm = {
        "type": "git",
        "subfolder": "source_subfolder",
        # ...
     }
```

```
project(MyLibraryWrapper)

include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup()

add_subdirectory("source_subfolder")
```

BKM: HOW TO RUN UNIT TESTS DURING PACKAGING PROCESS?

```
class MyLibraryConan(ConanFile):
    # ...
    generators = "cmake_paths"

    def build_requirements(self):
        if tools.get_env("CONAN_RUN_TESTS", False):
            self.build_requires("gtest/1.8.1@bincrafters/stable")

    def _configure_cmake(self):
        cmake = CMake(self)
        # ...
        cmake.definitions["BENCHMARK_ENABLE_TESTING"] = "ON" if tools.get_env("CONAN_RUN_TESTS", False) else "OFF"
        cmake.definitions["CMAKE_PROJECT_benchmark_INCLUDE"] = "conan_paths.cmake"
        cmake.configure()
        return cmake

    def build(self):
        cmake = self._configure_cmake()
        cmake.build()
        if tools.get_env("CONAN_RUN_TESTS", False):
            cmake.test()
```

BKM: HOW TO RUN UNIT TESTS DURING PACKAGING PROCESS?

```
class MyLibraryConan(ConanFile):
    # ...
    generators = "cmake_paths"

    def build_requirements(self):
        if tools.get_env("CONAN_RUN_TESTS", False):
            self.build_requires("gtest/1.8.1@bincrafters/stable")

    def _configure_cmake(self):
        cmake = CMake(self)
        # ...
        cmake.definitions["BENCHMARK_ENABLE_TESTING"] = "ON" if tools.get_env("CONAN_RUN_TESTS", False) else "OFF"
        cmake.definitions["CMAKE_PROJECT_benchmark_INCLUDE"] = "conan_paths.cmake"
        cmake.configure()
        return cmake

    def build(self):
        cmake = self._configure_cmake()
        cmake.build()
        if tools.get_env("CONAN_RUN_TESTS", False):
            cmake.test()
```

- To enable building and running tests define *environment variable*, set it in a *profile file*, or *run*

```
conan create . user/channel -e CONAN_RUN_TESTS=1
```

BKM: CUSTOM `FindXXX.cmake` FILE

REASON

- Package is *asocial* (does not have CMake configuration files or **`FindXXX.cmake`** file)
- Existing "official" **`FindXXX.cmake`** file is not able to find our libraries

BKM: CUSTOM FindXXX.cmake FILE

REASON

- Package is *asocial* (does not have CMake configuration files or **FindXXX.cmake** file)
- Existing "official" **FindXXX.cmake** file is not able to find our libraries

BKM

- Create a custom **FindXXX.cmake** file in package root folder and export it with the package

```
class MyLibraryConan(ConanFile):
    name = "my-library"
    version = "0.1"
    # ...
    exports_sources = ["FindMyLibrary.cmake"]

    def package(self):
        # ...
        self.copy("FindMyLibrary.cmake", ".", ".")
```

BKM: DO NOT USE CLASS VARIABLES TO SHARE STATE

```
class MyLibraryConan(ConanFile):
    name = "my-library"
    version = "0.1"
    something = ""
    some_dynamic_path = ""

    def source(self):
        self.something = "Hello"
        some_dynamic_path = find_some_dynamic_path()

    def build(self):
        print(self.something)
        use_some_dynamic_path(some_dynamic_path)

    def package(self):
        print(self.something)
        use_some_dynamic_path(some_dynamic_path)
```

- Works fine with **`conan create . user/channel`** because it *calls all the methods in sequence*
- *Problems with commands that execute only a part of the sequence*
  - i.e. the **`source()`** and **`build()`** methods won't be called at all when you run **`conan package`** command

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
- *Easy to use* and the *documentation* is really good

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
- *Easy to use* and the *documentation* is really good
- **Give it a try!**

```
pip install conan
```


CAUTION
Programming
is addictive
(and too much fun)