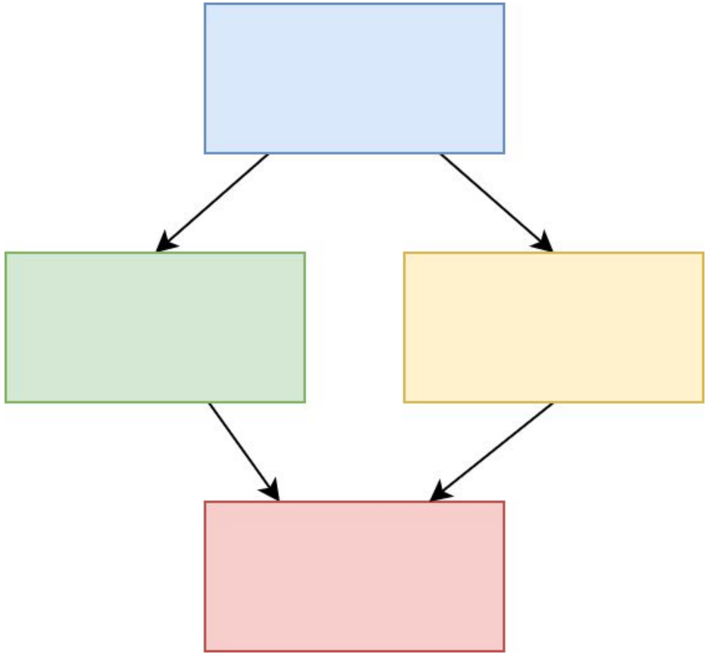
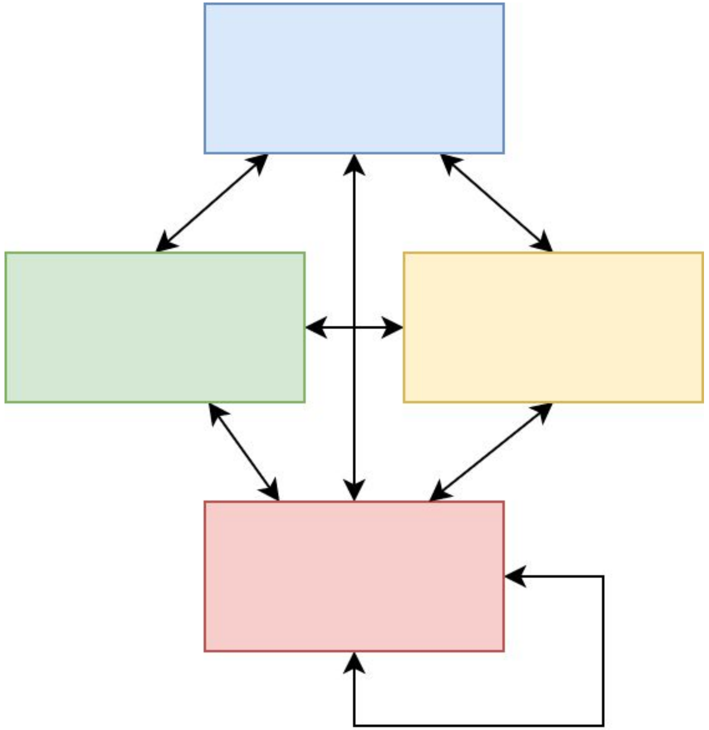
Modern CMake for modular design





Did you ever...

About this talk

- Modular design
- Build systems (CMake in particular)
- … and how to combine that to improve your codeline

Hello!

I am Mathieu Ropert

I'm a senior developer at Murex, a contributor to Conan, and I love portable C++.

You can reach me at:

- ✉ mro@puchiko.net
- @MatRopert
- @mropert

Let’s talk about CMake

- Not a build system!
- “Cross-platform C++ build generator”
- First released in 2000
- Used by many projects

Let’s talk about CMake

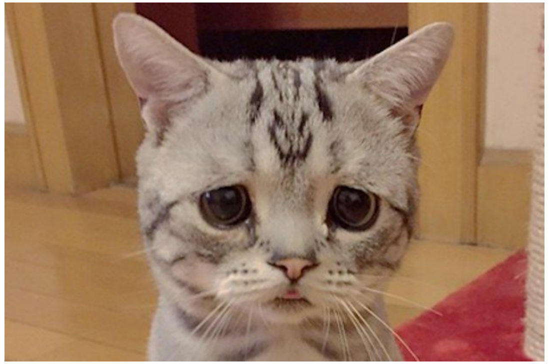
```
> cmake -G “Visual Studio 15 2017 Win64” .
```

```
$ cmake -DCMAKE_CXX_COMPILER=”clang++”
      -DCMAKE_CXX_FLAGS=”-stdlib=libc++ -m64” .
```

Modern CMake?

- Available since version 2.8.12 (Oct 2013)
- In practice, version 3.0.0 (June 2014)

```
cmake_minimum_required(VERSION 2.8)
```



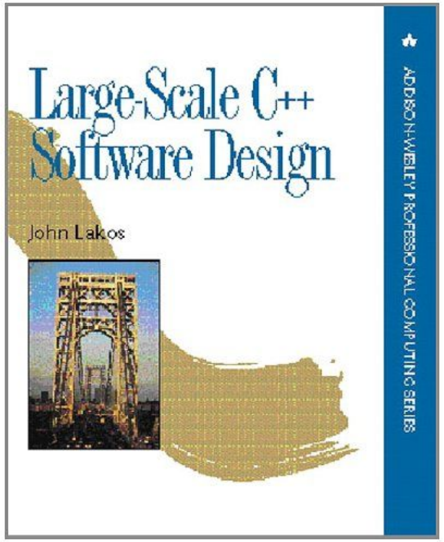
1

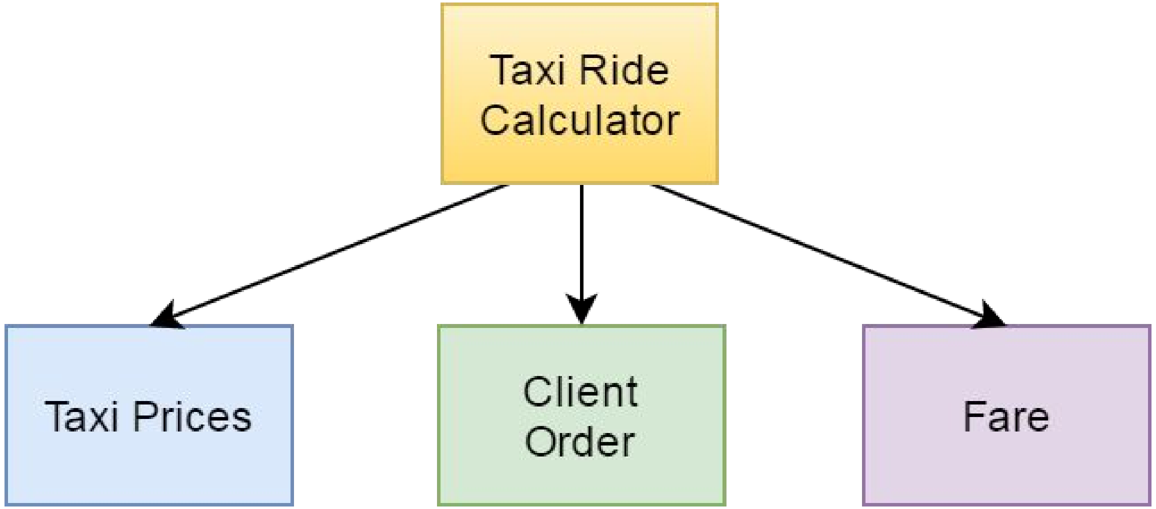
Modular Design

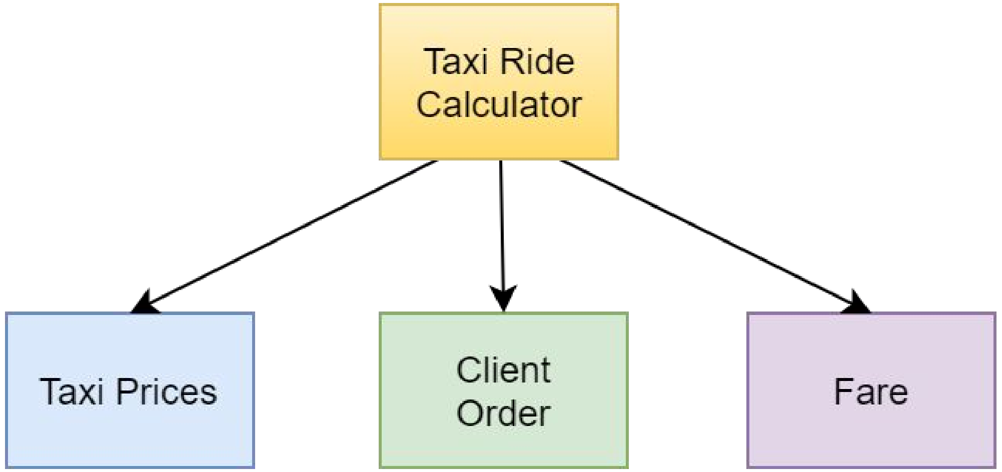
A brief recap of the philosophy

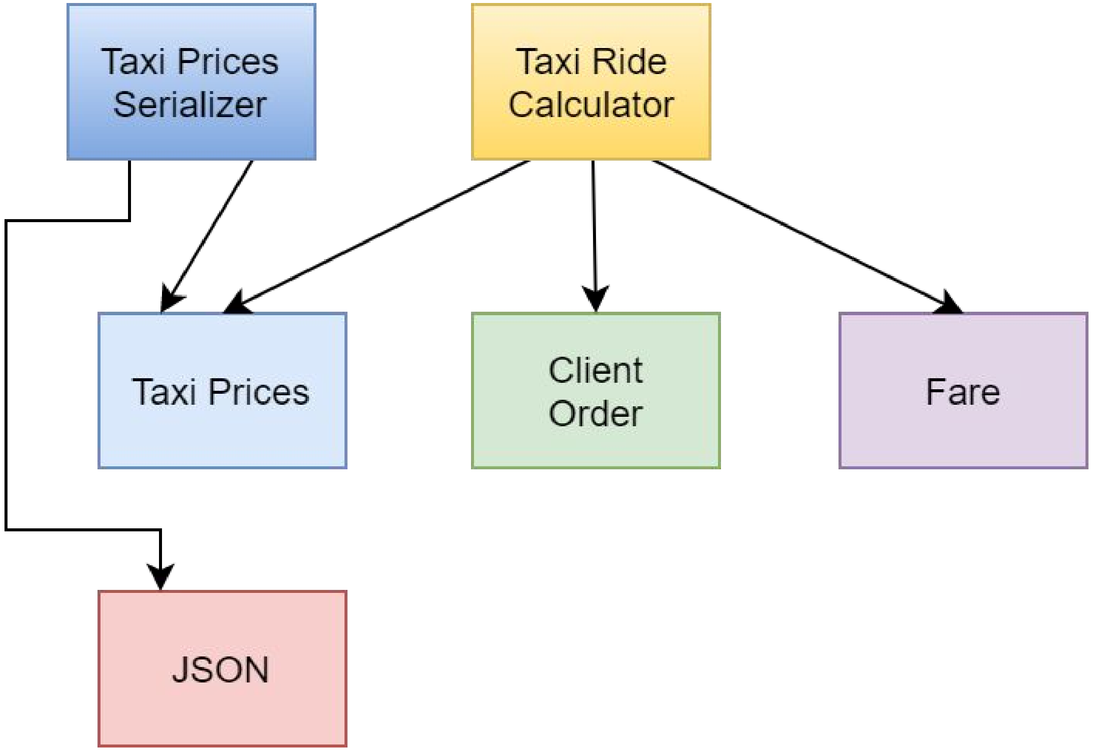
Modular Design at scale

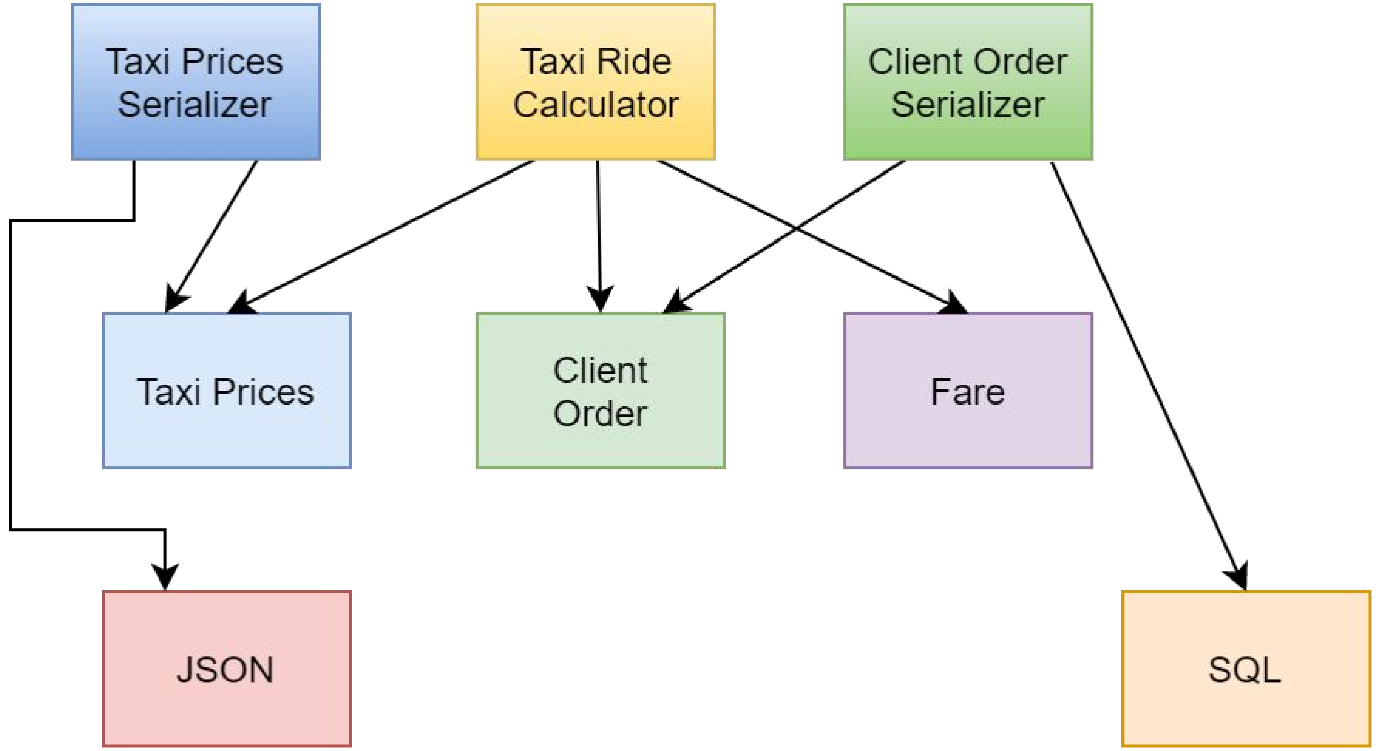
- *Large-Scale C++ Software Design (1996)*
- *Advanced Levelization techniques* talk series *(CppCon 2016)*

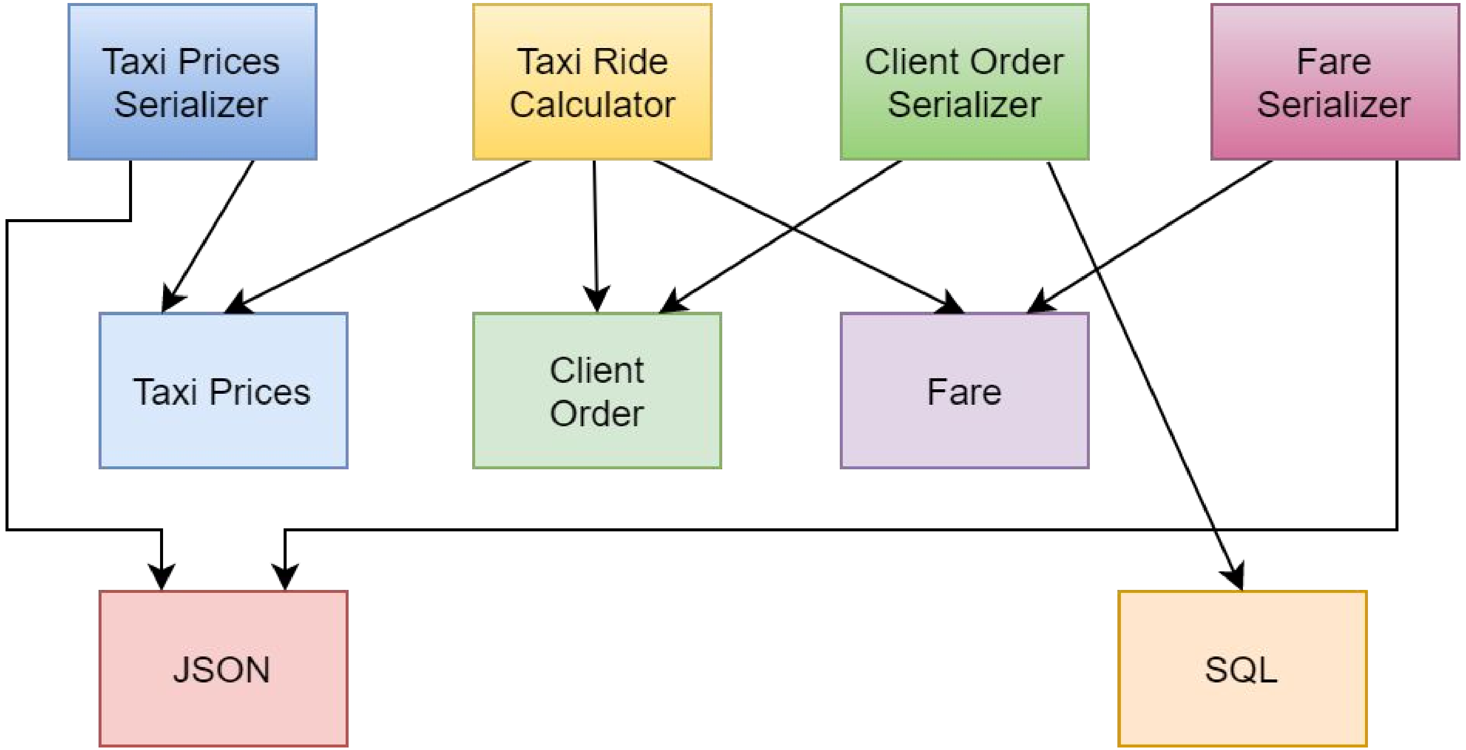

Large-Scale C++ Software Design
John Lakos
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

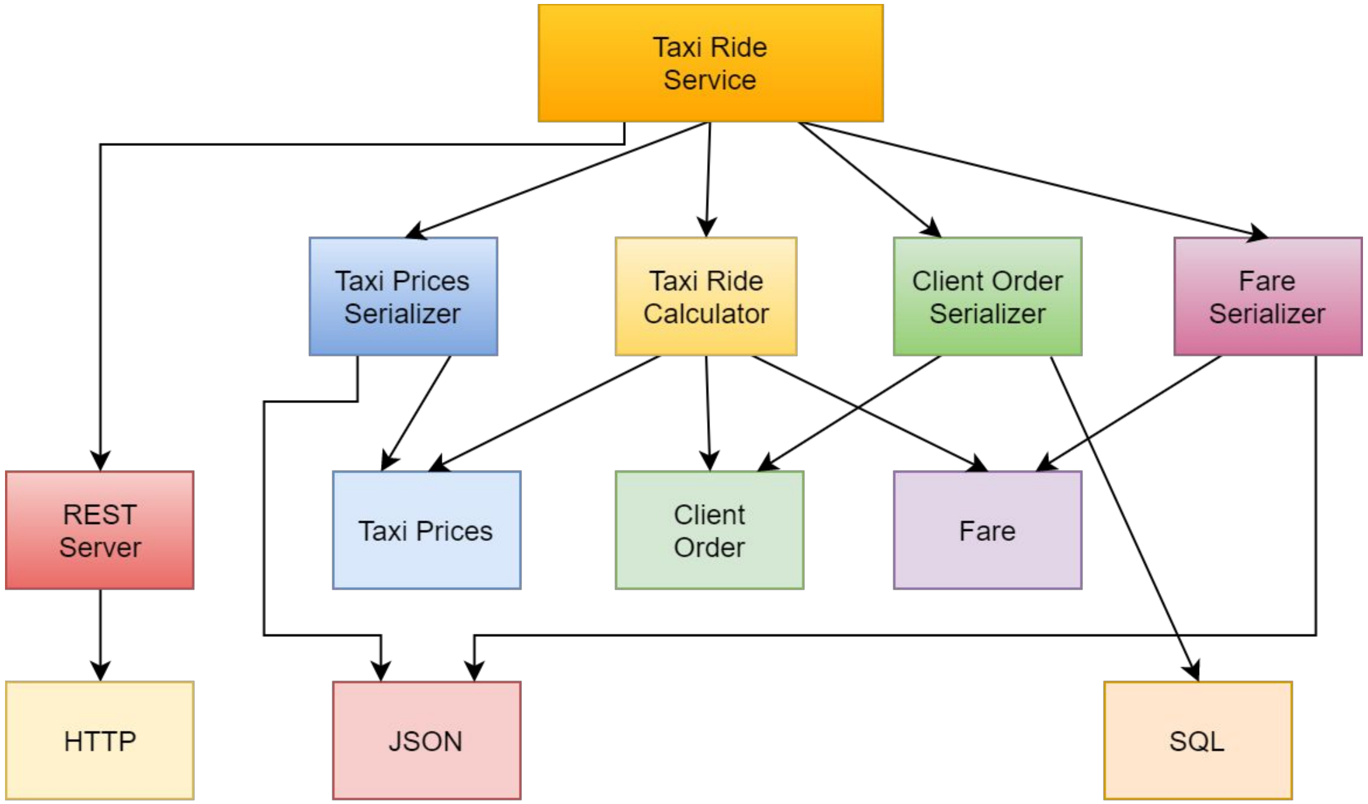

Taxi Ride Calculator
Taxi Prices
Client Order
Fare

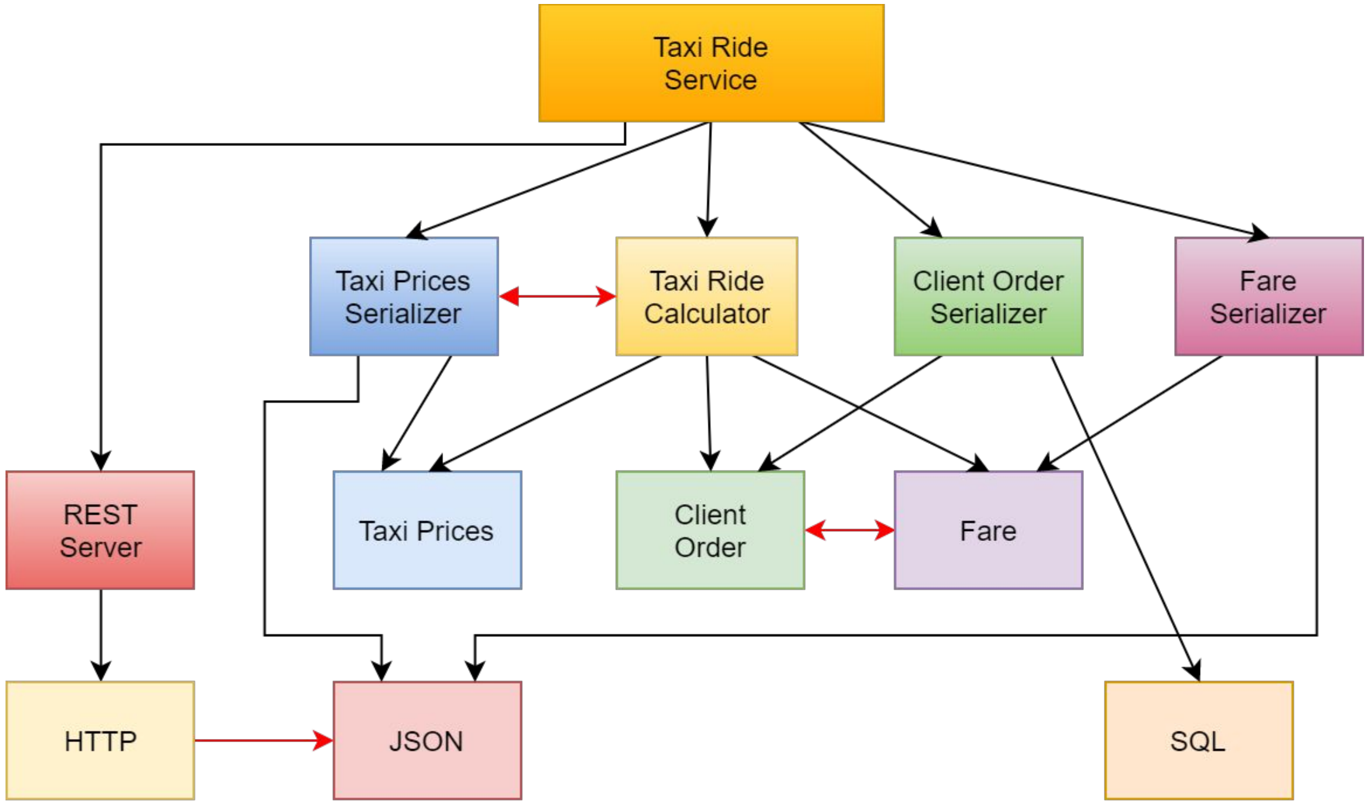

Taxi Ride Calculator
Taxi Prices
Client Order
Fare

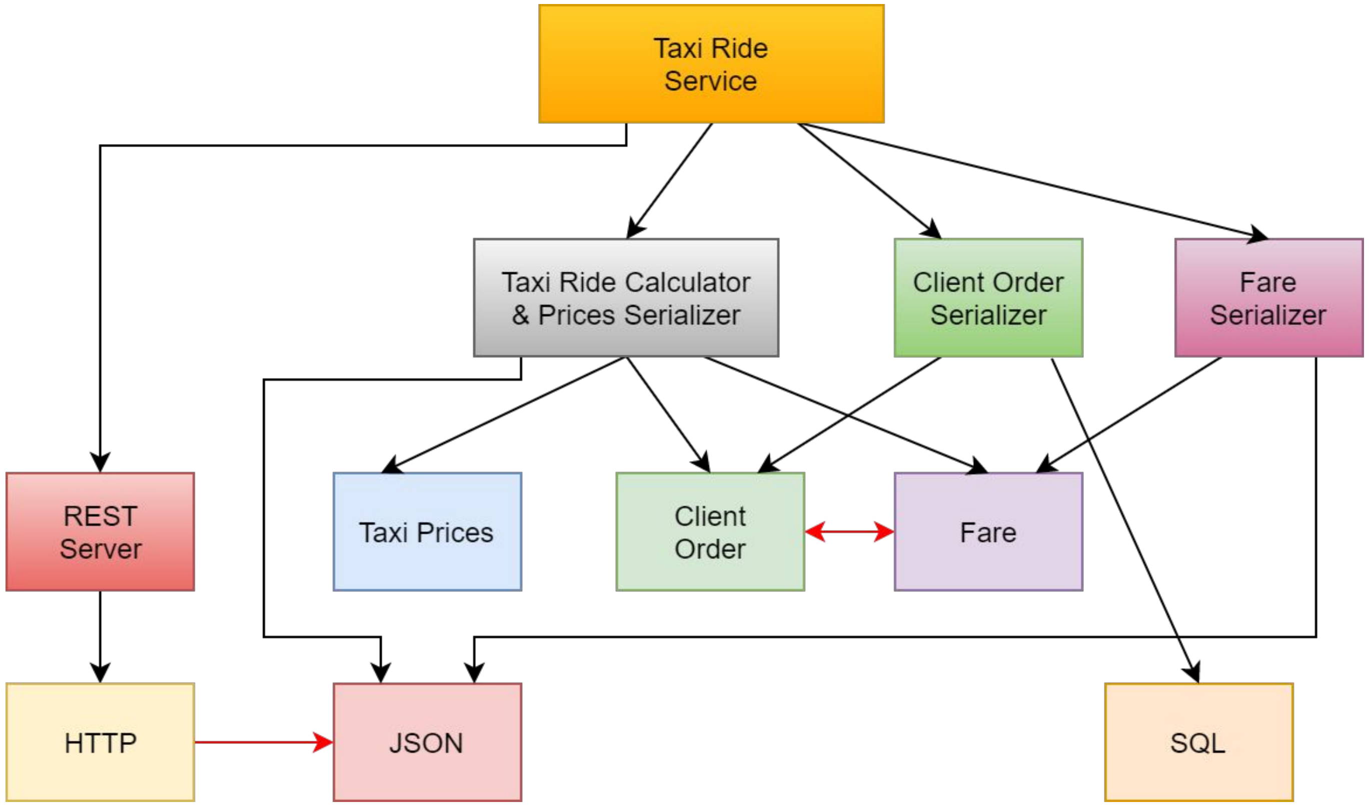

Taxi Prices Serializer
Taxi Ride Calculator
Taxi Prices
Client Order
Fare
JSON

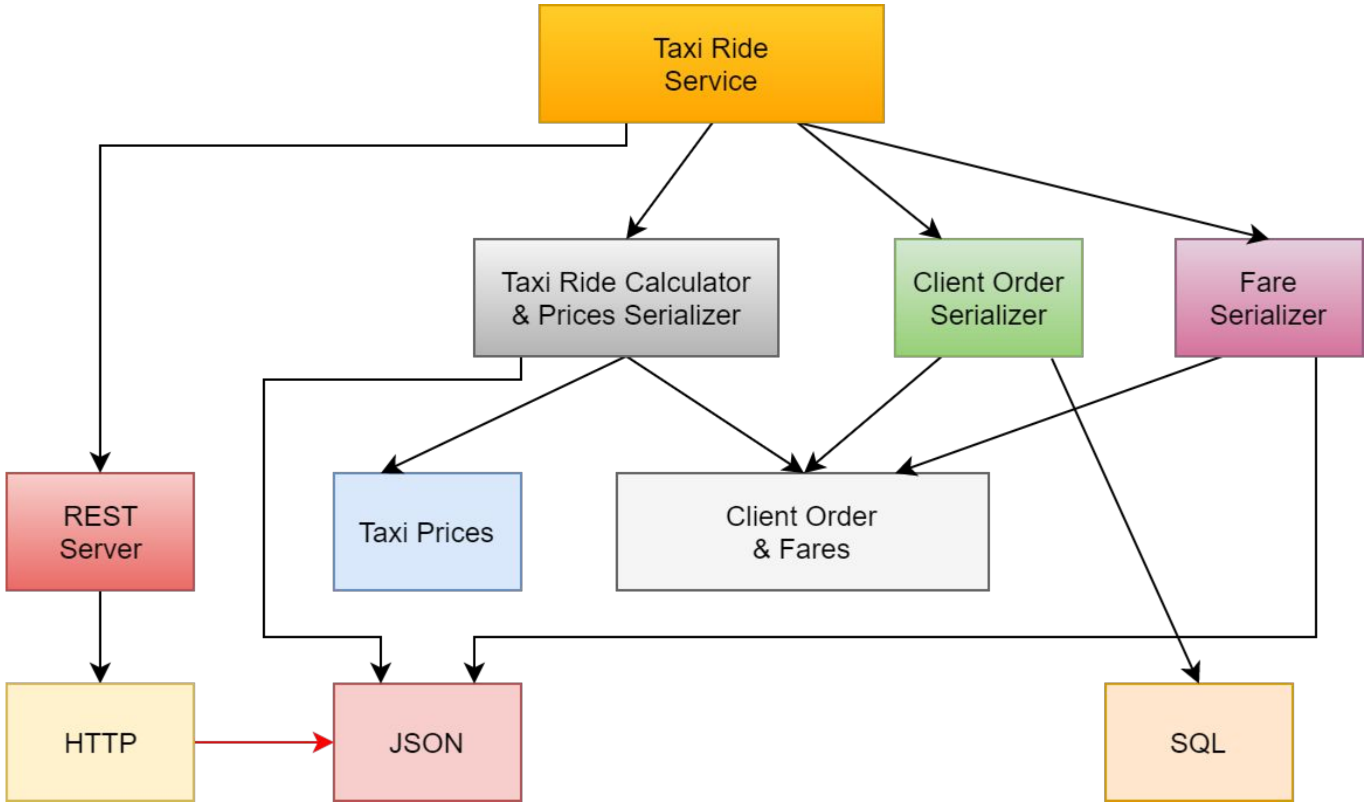

Taxi Prices Serializer
Taxi Ride Calculator
Client Order Serializer
Taxi Prices
Client Order
Fare
JSON
SQL


Taxi Prices Serializer
Taxi Ride Calculator
Client Order Serializer
Fare Serializer
Taxi Prices
Client Order
Fare
JSON
SQL

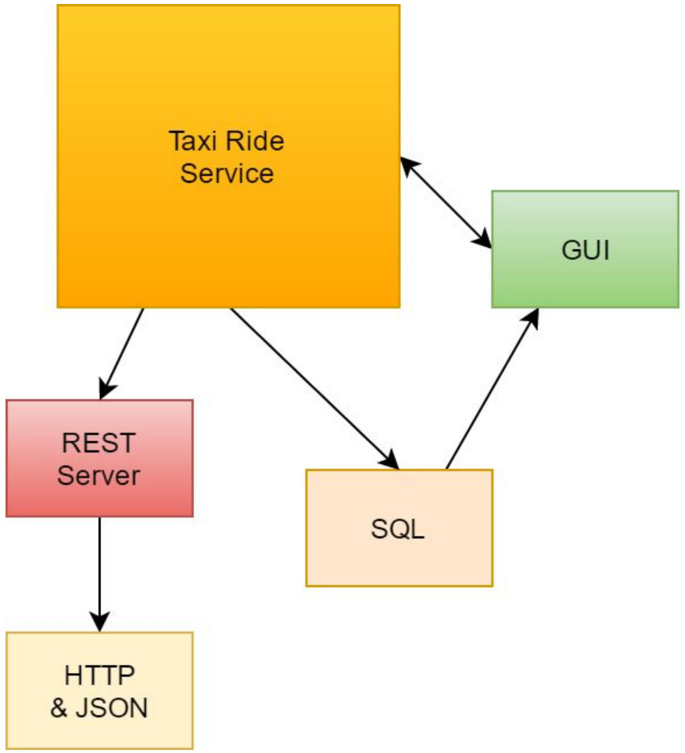

Taxi Ride Service
Taxi Prices Serializer
Taxi Ride Calculator
Client Order Serializer
Fare Serializer
REST Server
Taxi Prices
Client Order
Fare
HTTP
JSON
SQL


Taxi Ride Service
Taxi Prices Serializer
Taxi Ride Calculator
Client Order Serializer
Fare Serializer
REST Server
Taxi Prices
Client Order
Fare
HTTP
JSON
SQL


Taxi Ride Service
Taxi Ride Calculator & Prices Serializer
Client Order Serializer
Fare Serializer
REST Server
Taxi Prices
Client Order
Fare
HTTP
JSON
SQL

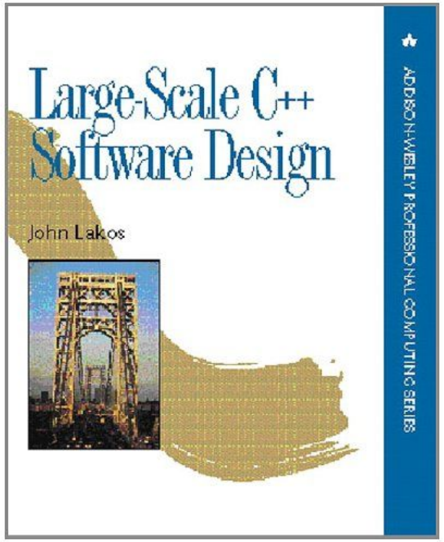


Taxi Ride Service
Taxi Ride Calculator & Prices Serializer
Client Order Serializer
Fare Serializer
REST Server
Taxi Prices
Client Order & Fares
HTTP
JSON
SQL


Taxi Ride Service
GUI
REST Server
SQL
HTTP & JSON

Modular Design at scale

- Retain control of your dependency graph
- Keep concerns separated
- Make modules reusable in other contexts at minimal cost


Large-Scale C++ Software Design
John Lakos
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

2

Modern build systems

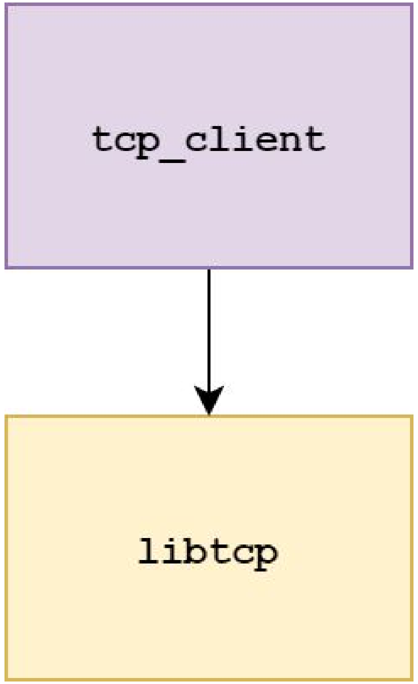
Theory & practice

Modern build system

Facilitate large scale modular design
& protect against antipatterns

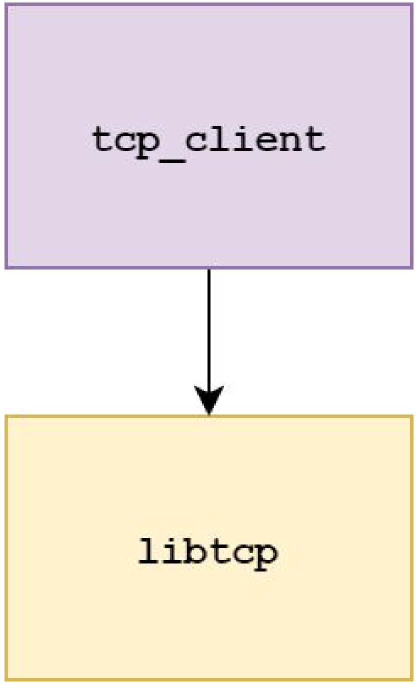
Build before

```
ADD_SUBDIRECTORY(libtcp)

ADD_EXECUTABLE(tcp_client
     tcp_client.cpp)

INCLUDE_DIRECTORIES(tcp/include)
ADD_DEFINITIONS(IPV6)

TARGET_LINK_LIBRARIES(tcp_client
                      libtcp)
```



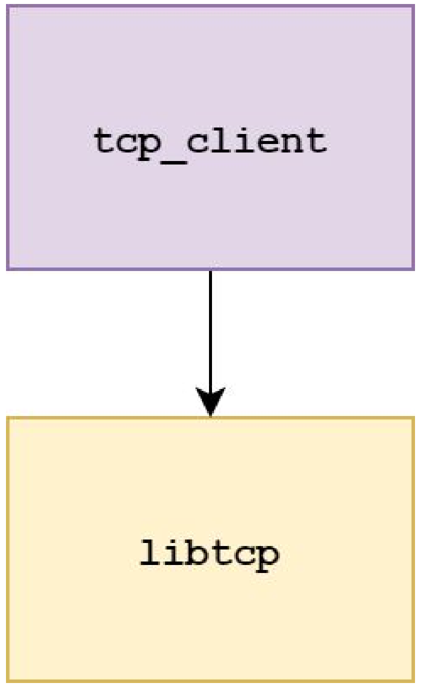
Build before

```
ADD_SUBDIRECTORY(libtcp)

ADD_EXECUTABLE(tcp_client
     tcp_client.cpp)

INCLUDE_DIRECTORIES(tcp/include)
ADD_DEFINITIONS(IPV6)

TARGET_LINK_LIBRARIES(tcp_client
                      libtcp)
```



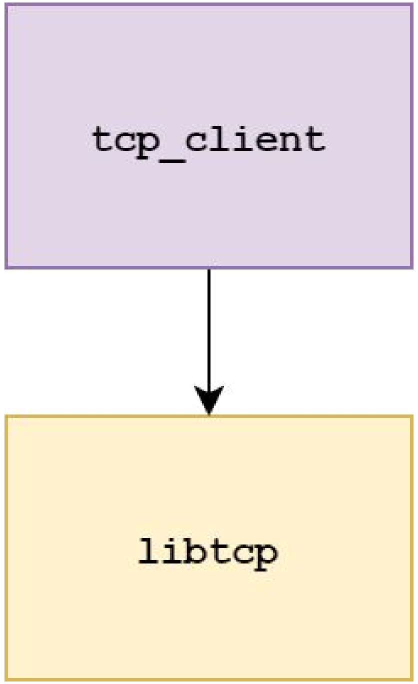
Build before

```
ADD_SUBDIRECTORY(libtcp)

ADD_EXECUTABLE(tcp_client
      tcp_client.cpp)

INCLUDE_DIRECTORIES(tcp/include)
ADD_DEFINITIONS(IPV6)

TARGET_LINK_LIBRARIES(tcp_client
                                 libtcp)
```



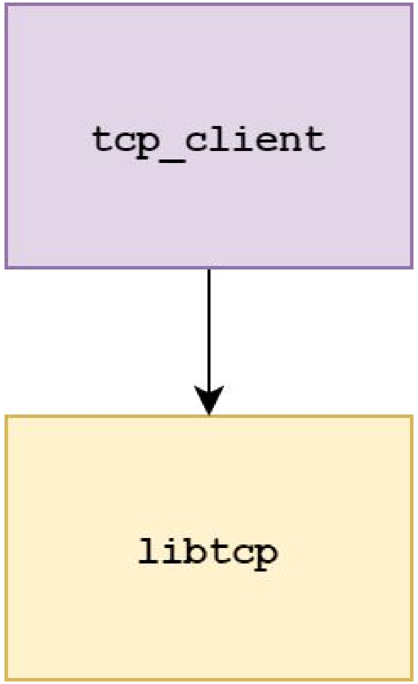
Build before

```
ADD_SUBDIRECTORY(libtcp)

ADD_EXECUTABLE(tcp_client
     tcp_client.cpp)

INCLUDE_DIRECTORIES(tcp/include)
ADD_DEFINITIONS(IPV6)

TARGET_LINK_LIBRARIES(tcp_client
                           libtcp)
```



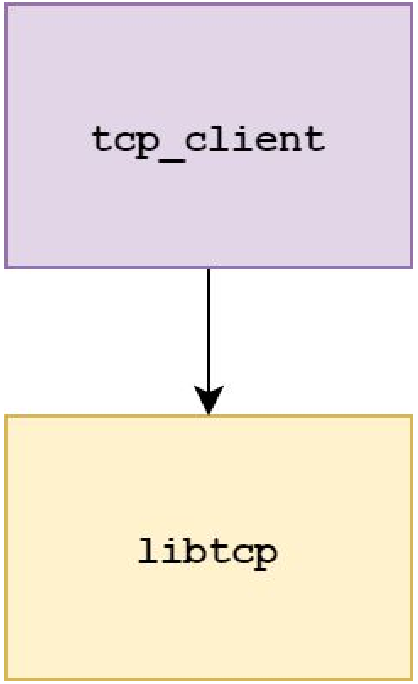
Build before

```
ADD_SUBDIRECTORY(libtcp)

ADD_EXECUTABLE(tcp_client
     tcp_client.cpp)

INCLUDE_DIRECTORIES(tcp/include)
ADD_DEFINITIONS(IPV6)

TARGET_LINK_LIBRARIES(tcp_client
                       libtcp)
```



Build before

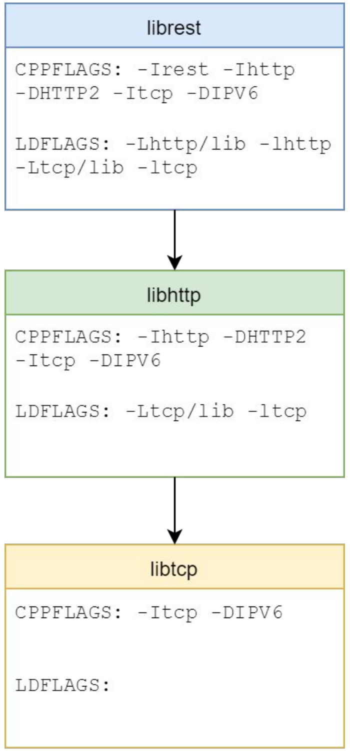
```
ADD_SUBDIRECTORY(libtcp)

ADD_EXECUTABLE(tcp_client
     tcp_client.cpp)

INCLUDE_DIRECTORIES(tcp/include)
ADD_DEFINITIONS(IPV6)

TARGET_LINK_LIBRARIES(tcp_client
                              libtcp)
```



Build flags don't scale

- Every change in public flags has to be propagated upwards

- Most people usually give up and put every include directory in a common/root build file



```
librest
CPPFLAGS: -Irest -Ihttp
-DHTTP2 -Itcp -DIPV6

LDFLAGS: -Lhttp/lib -lhttp
-Ltcp/lib -ltcp
```

```
libhttp
CPPFLAGS: -Ihttp -DHTTP2
-Itcp -DIPV6

LDFLAGS: -Ltcp/lib -ltcp
```

```
libtcp
CPPFLAGS: -Itcp -DIPV6

LDFLAGS:
```

Help the build system help you

- It's not easy to detect bad code architecture patterns when looking at build flags
- In contrast, defining build in term of modules depending on other modules makes the problem trivial

Modern build systems

- Forbid/report circular and hidden dependencies
- Help developer reason at module level
- Do more than build as you are told!

Modern build, in practice

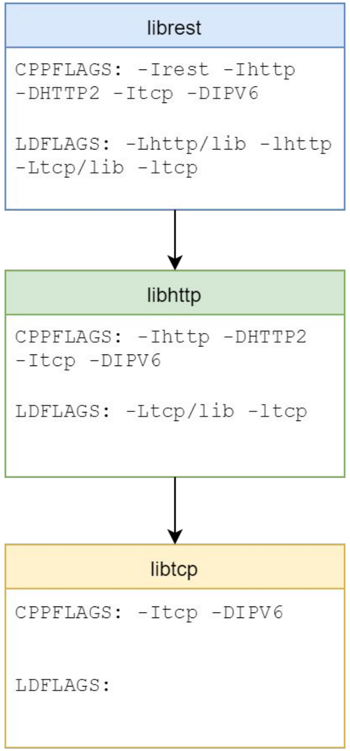
- Define your module build flags
- Define your module dependencies
- Keep out of other modules internals

Modern build, in practice

- Each module has a set of private flags (required to build its implementation)
- Each module has a set of public flags (required to build against its interface)
- Build interfaces are transitive

Public/private dependencies

- Dependencies are either public or private
- Public dependencies are transitive and will be passed down to clients



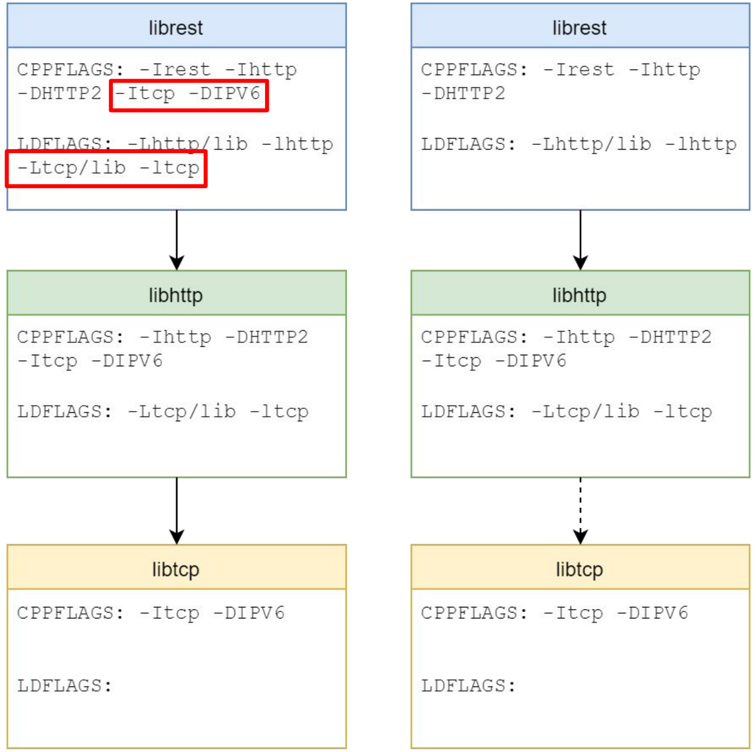
```
librest
CPPFLAGS: -Irest -Ihttp
-DHTTP2 -Itcp -DIPV6

LDFLAGS: -Lhttp/lib -lhttp
-Ltcp/lib -ltcp
```

```
libhttp
CPPFLAGS: -Ihttp -DHTTP2
-Itcp -DIPV6

LDFLAGS: -Ltcp/lib -ltcp
```

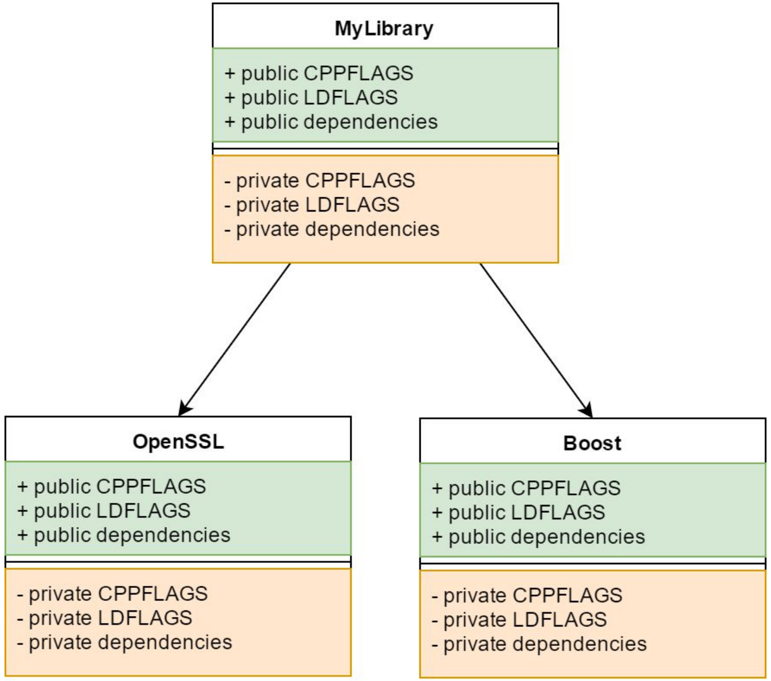
```
libtcp
CPPFLAGS: -Itcp -DIPV6

LDFLAGS:
```

Public/private dependencies

- Dependencies are either public or private
- Public dependencies are transitive and will be passed down to clients
- Private dependencies are not


librest
CPPFLAGS: -Irest -Ihttp -DHTTP2 -Itcp -DIPV6
LDFLAGS: -Lhttp/lib -lhttp -Ltcp/lib -ltcp
libhttp
CPPFLAGS: -Ihttp -DHTTP2 -Itcp -DIPV6
LDFLAGS: -Ltcp/lib -ltcp
libtcp
CPPFLAGS: -Itcp -DIPV6
LDFLAGS:
librest
CPPFLAGS: -Irest -Ihttp -DHTTP2
LDFLAGS: -Lhttp/lib -lhttp
libhttp
CPPFLAGS: -Ihttp -DHTTP2 -Itcp -DIPV6
LDFLAGS: -Ltcp/lib -ltcp
libtcp
CPPFLAGS: -Itcp -DIPV6
LDFLAGS:

Keep calm and focus



- Build flags aren't gone, only encapsulated
- You can still go crazy with CPPFLAGS, CXXFLAGS and LDFLAGS in your module
- But external flags aren't your concern anymore

3

Modern CMake

Let’s see some code!

Modern CMake in a nutshell

- Declare your module with ADD_LIBRARY or ADD_EXECUTABLE
- Declare your build flags with TARGET_xxx()
- Declare your dependencies with TARGET_LINK_LIBRARIES
- Specify what is PUBLIC and what is PRIVATE

Global setup

```
cmake_minimum_required(VERSION 3.0)

if(MSVC)
    add_compile_options(/W3 /WX)
else()
    add_compile_options(-W -Wall -Werror)
endif()
```

Declare your module

```
add_library(mylib
    src/file1.cpp
    src/file2.cpp
    ...)
```

Declare your flags

```
target_include_directories(mylib PUBLIC include)
target_include_directories(mylib PRIVATE src)

if (SOME_SETTING)
    target_compile_definitions(mylib
                    PUBLIC WITH_SOME_SETTING)
endif()
```

If the setting only affects implementation, use `PRIVATE` instead

Declare your dependencies

```cmake
# Public (interface) dependencies
target_link_libraries(mylib PUBLIC abc)

# Private (implementation) dependencies
target_link_libraries(mylib PRIVATE xyz)
```

Header-only libraries

```
add_library(mylib INTERFACE)
```

Nothing to build so it must be INTERFACE

```
target_include_directories(mylib INTERFACE include)

target_link_libraries(mylib INTERFACE Boost::Boost)
```

Recognize antipatterns

- Don’t use macros that affect all targets
  - INCLUDE_DIRECTORIES()
  - ADD_DEFINITIONS()
  - LINK_LIBRARIES
- Don’t use TARGET_INCLUDE_DIRECTORIES() with a path outside your module

Recognize antipatterns

- Don’t use TARGET_LINK_LIBRARIES() without specifying PUBLIC, PRIVATE or INTERFACE
- Don’t use TARGET_COMPILE_OPTIONS() to set flags that affect the ABI

That’s it!

Remember this
and you know 90% of Modern CMake

4

Beyond CMake

How to interact with the rest of the world

External projects

- Require external packages

```
find_package(GTest)
find_package(Threads)

add_executable(foo ...)

target_include_directories(foo
      PRIVATE ${GTEST_INCLUDE_DIRS})

target_link_libraries(foo
      PRIVATE ${GTEST_BOTH_LIBRARIES}
              Threads::Threads)
```

External projects

- Require external packages
- Don’t fall back to the old “flags” approach!

```
find_package(GTest)
find_package(Threads)

add_executable(foo ...)

target_include_directories(foo
    PRIVATE ${GTEST_INCLUDE_DIRS})

target_link_libraries(foo
    PRIVATE ${GTEST_BOTH_LIBRARIES}
            Threads::Threads)
```

External projects

- External packages should be targets too
- CMake built-in finders have undergone an effort in that sense

```
cmake_minimum_required(VERSION 3.5)

find_package(GTest)

add_executable(foo ...)

target_link_libraries(foo
                   GTest::GTest GTest::Main)
```

External projects

- Modern finders provide targets instead of flags
  - 3.4: OpenSSL
  - 3.5: Boost, GTest, GTK, PNG, TIFF
  - 3.6: PkgConfig
  - 3.7: Bzip2, OpenCL
  - 3.8: OpenGL

- Another reason to upgrade your CMake version!

Hand-made finder

- Creating your own target finder isn’t hard
- You should provide one with your public libraries
- CMake can even generate it for you!

Finder expectations

```
find_library(BAR_LIB bar HINTS ${BAR_DIR}/lib)
add_library(bar SHARED IMPORTED
            LOCATION ${BAR_LIB})


target_include_directories (bar INTERFACE ${BAR_DIR}/include)


target_link_libraries (bar INTERFACE Boost::boost)
```

Finder reality

Finder reality

```
find_library(BAR_LIB bar HINTS ${BAR_DIR}/lib)
add_library(bar SHARED IMPORTED)
set_target_properties(bar PROPERTIES
                      LOCATION ${BAR_LIB})
```

```
set_target_properties (bar PROPERTIES
                 INTERFACE_INCLUDE_DIRECTORIES  ${BAR_DIR}/include)
                 INTERFACE_LINK_LIBRARIES  Boost::boost)
```

Easier alternative(s)?

- There are a few but…
- As Fermat famously said: “it wouldn’t fit in the margin of this talk”
- Check-out Daniel Pfeifer’s talk *Effective CMake*

5 Wrapping up

Modern CMake in three slides

Modern build

- Keep your flags to yourself
- Think in terms of modules
- Let the build system handle transitivity

Modern CMake

- Switch to CMake 3.X
- Use the TARGET_xxx version of macros
- Specify if a property is PUBLIC, PRIVATE or INTERFACE
- Link against targets to get their build flags

External Packages

- Use modern finders that declare targets
- Generate them with CMake, from your actual project definition
- Use a package manager

Thanks!

Any questions ?

You can find me at

- mro@puchiko.net
- @MatRopert
- @mropert