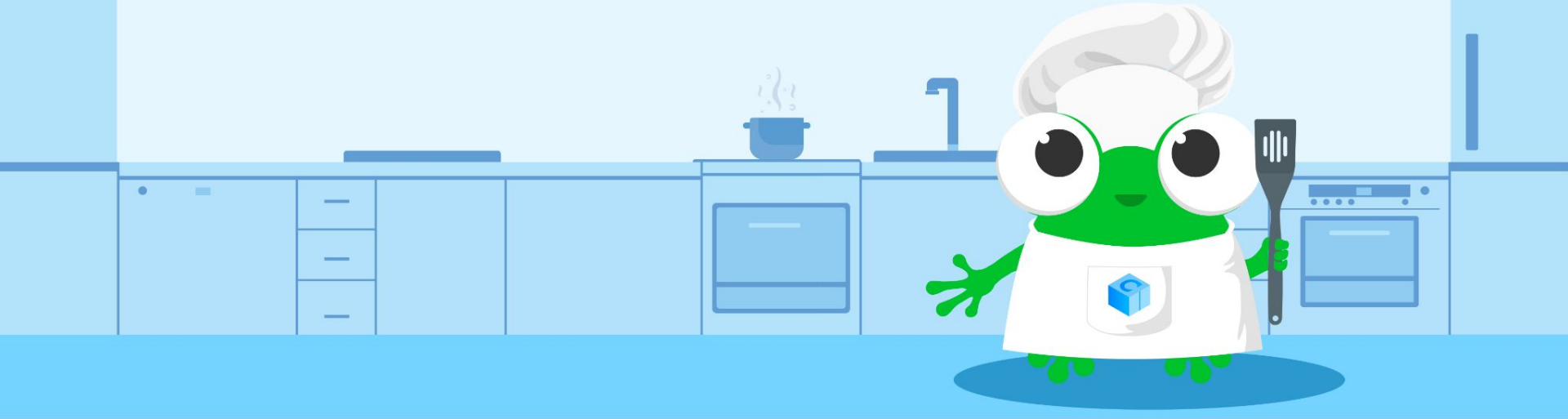
CONAN 2.0: LESSONS LEARNED FROM THE C++ ECOSYSTEM

LUIS CARO CAMPOS



Developer envy

Does C++ have a package manager like npm, pip, gem, etc? [closed]

Asked 8 years, 3 months ago Modified 7 months ago Viewed 71k times

Developer envy

Does C++ have a package manager like npm, pip, gem, etc? [closed]

Asked 8 years, 3 months ago Modified 7 months ago Viewed 71k times

Posted by u/pistacchio 9 years ago

95

What bothers me about c++

Recently I started relearning C++. What Srautsroup says, that it feels like a new language, is true. If you start with a fresh mentality and follow all the idioms, C++11 is really an amazing and powerful language.

Working daily with Python, Ruby and Node, what really bothers me is C++ primitive dependency management environment.

Developer envy

Does C++ have a package manager like npm, pip, gem, etc? [closed]

Asked 8 years, 3 months ago Modified 7 months ago Viewed 71k times

Posted by u/pistacchio 9 years ago

95

What bothers me about c++

Recently I started relearning C++. What Srautsroup says, that it feels like a new language, is true. If you start with a fresh mentality and follow all the idioms, C++11 is really an amazing and powerful language.

Working daily with Python, Ruby and Node, what rea
management environment.

Posted by u/antoninj 8 years ago

70

Is there a C++ package manager? If not, how do you handle dependencies?

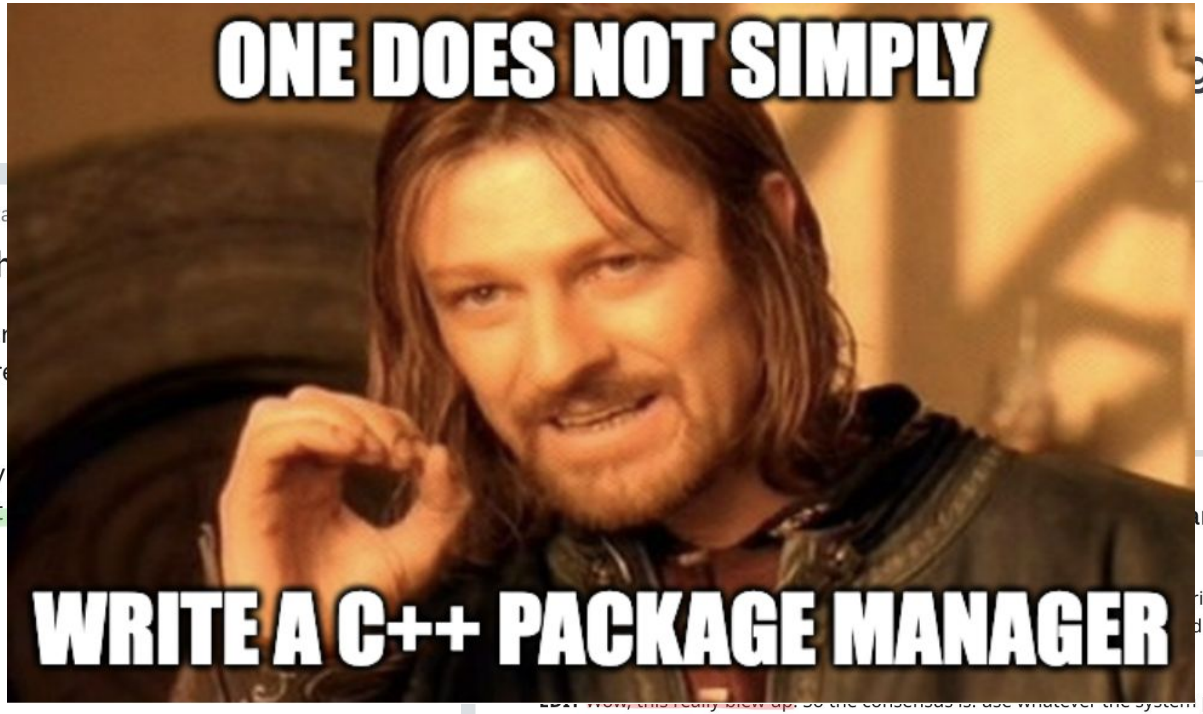
Coming from the PHP and Javascript world, I'm used to using packages to bring down dependencies and implement them into my project. I'm only starting with C++ but I wanted to be aware if there is any tooling around dependencies.

EDIT Wow, this really blew up. So the consensus is: use whatever the system provides you with or resort to git submodules.

101 Comments Award Share Save ...

JFrog | CONAN C/C++ Package Manager

Developer envy



gem, etc? [closed]

Posted by u/pista

95

What both

Recently I star
start with a fre
language.

Working daily
management

ndle

ring down dependencies
d to be aware if there is

provides you with or
resort to git submodules.

101 Comments Award Share Save ...

JFrog CONAN C/C++ Package Manager

Case study: pip and a (pure) Python library

hola.py

pip-example > hola.py > ...

```
1  import urllib3
2  http = urllib3.PoolManager()
3  r = http.request('GET', 'https://conan.io')
4
5  print(f"HTTP GET status: {r.status}")
```

```
$ pip install urllib3
```

Case study: pip and a (pure) Python library

hola.py

```
pip-example > hola.py > ...
1  import urllib3
2  http = urllib3.PoolManager()
3  r = http.request('GET', 'https://conan.io')
4
5  print(f"HTTP GET status: {r.status}")
```

```
$ pip install urllib3
```

Built Distribution

urllib3-1.26.15-py2.py3-none-any.whl (140.9 kB view hashes)

Uploaded Mar 11, 2023 py2 py3

Case study: pip and a (pure) Python library

hola.py

```
pip-example > hola.py > ...
1   import urllib3
2   http = urllib3.PoolManager()
3   r = http.request('GET', 'https://conan.io')
4
5   print(f"HTTP GET status: {r.status}")
```

```
$ pip install urllib3
```

Built Distribution

urllib3-1.26.15-py2.py3-none-any.whl (140.9 kB view hashes)

Uploaded Mar 11, 2023 py2 py3

Same file, same contents, irrespective of:

- Operating system
- CPU architecture
- Version of Python

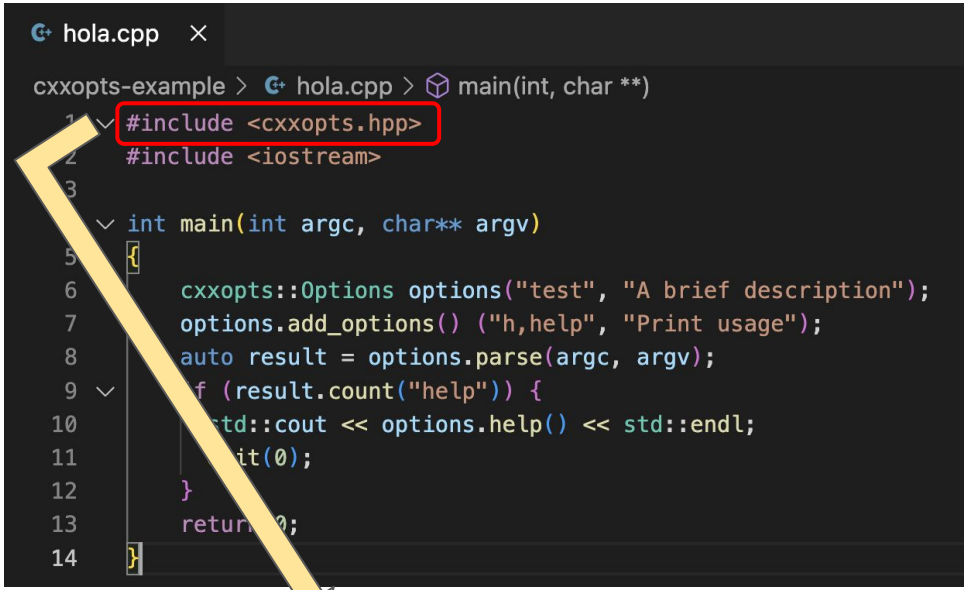
Case study: header only library in C++

hola.cpp

cxxopts-example > hola.cpp > main(int, char **)

```cpp
#include <cxxopts.hpp>
#include <iostream>

int main(int argc, char** argv)
{
    cxxopts::Options options("test", "A brief description");
    options.add_options() ("h,help", "Print usage");
    auto result = options.parse(argc, argv);
    if (result.count("help")) {
      std::cout << options.help() << std::endl;
      exit(0);
    }
    return 0;
}
```

Case study: header only library in C++

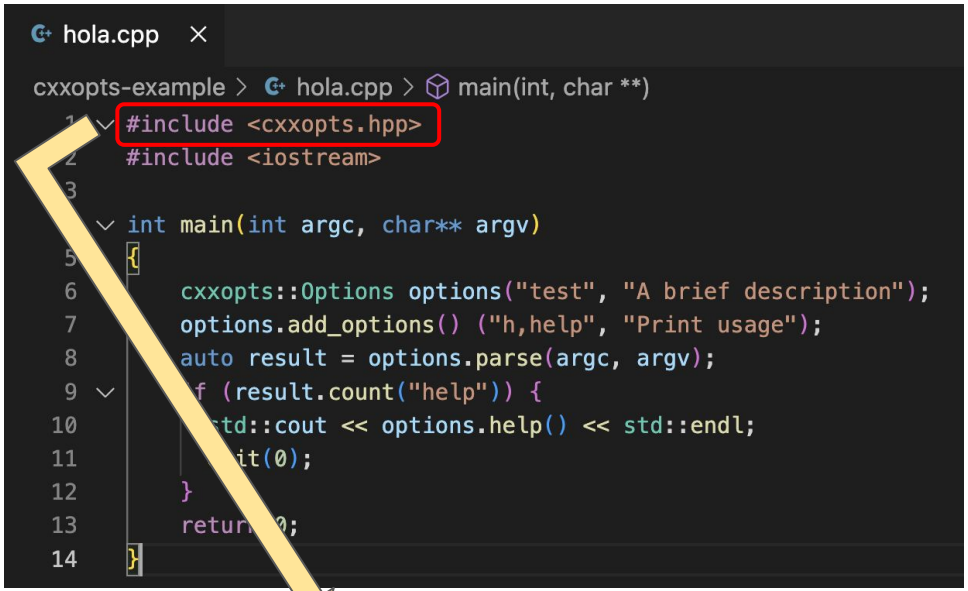


Same file, same contents, irrespective of:

- Operating system
- CPU architecture
- Compiler or compiler version

JFrog CONAN C/C++ Package Manager

Case study: header only library in C++



Same file, same contents, irrespective of:

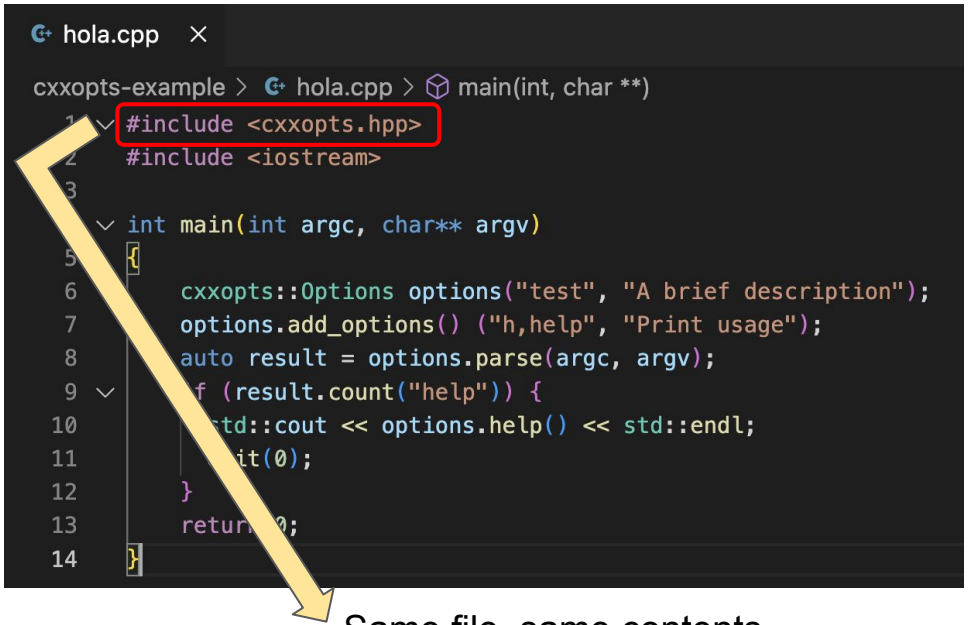
- Operating system
- CPU architecture
- Compiler or compiler version

```
$ ???? install cxxopts
```

- conan
- apt
- brew
- vcpkg
- …

JFrog CONAN C/C++ Package Manager

Case study: header only library in C++



Same file, same contents, irrespective of:

- Operating system
- CPU architecture
- Compiler or compiler version

```
$ ???? install cxxopts
```

- conan
- apt
- brew
- vcpkg
- …

First hurdle: not all of these will install it in a way that the compiler will “**just find it**”.

```
clang++ -I[/path/to/cxxopts/include] hola.cpp
-o hola
```

This library does not cause anything to be passed to the linker at build time or runtime - only a single compiler flag

JFrog CONAN C/C++ Package Manager

Binary libraries: Python

- Some interpreted languages like Python can make calls to C/C++ binary code
- The CPython interpreter can call C code
- This complicates the package manager story: Python package managers are also exposed to binary compatibility

```
$ pip install lief
```

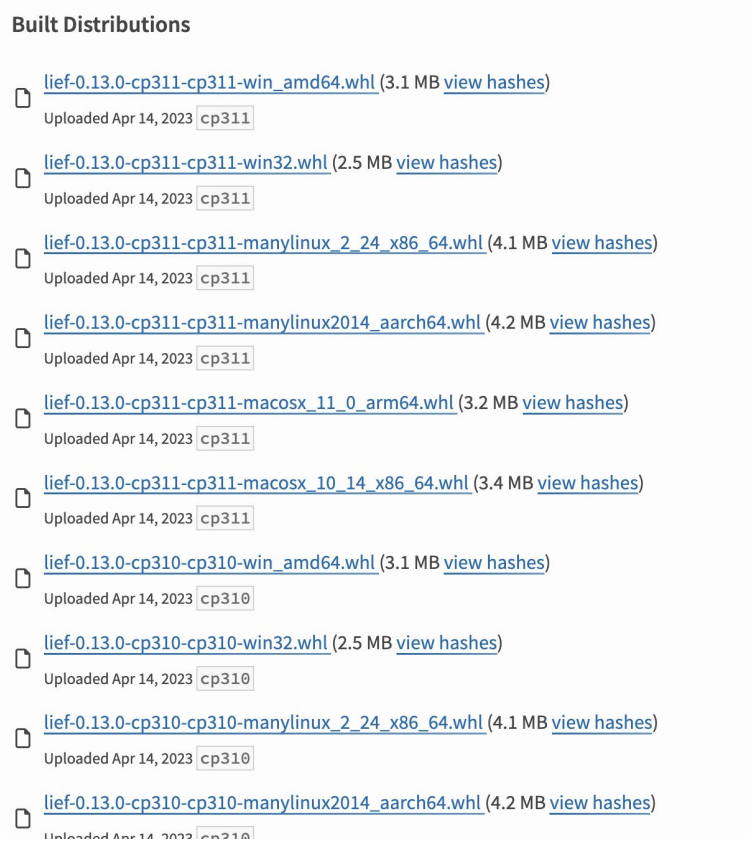
Binary libraries: Python

- Some interpreted languages like Python can make calls to C/C++ binary code
- The CPython interpreter can call C code
- This complicates the package manager story: Python package managers are also exposed to binary compatibility

```
$ pip install lief
```

Variability modeled with:

- CPython version
- **OS and Version**
- Architecture



JFrog | CONAN C/C++ Package Manager

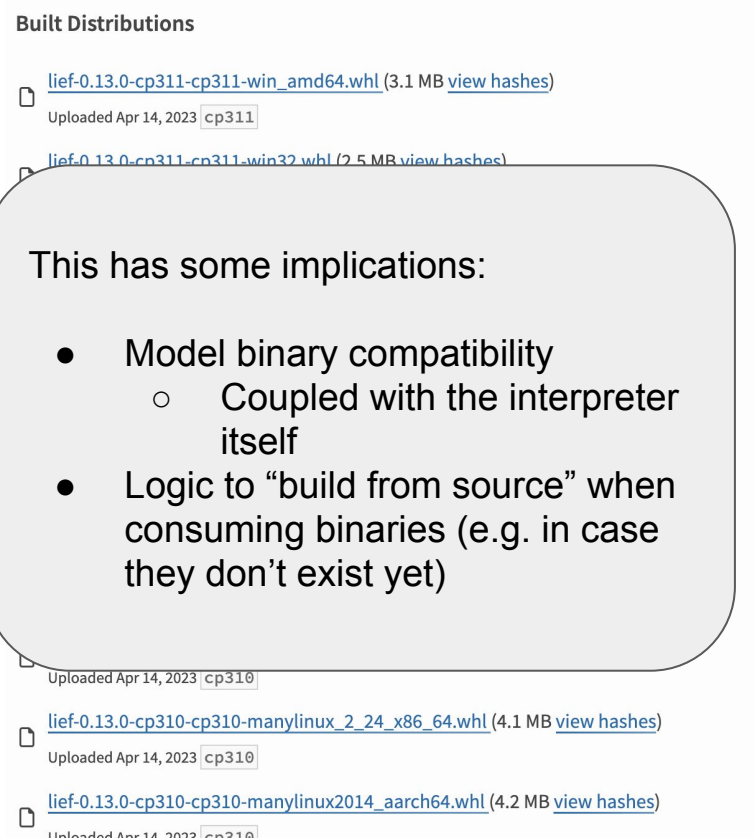
Binary libraries: Python

- Some interpreted languages like Python can make calls to C/C++ binary code
- The CPython interpreter can call C code
- This complicates the package manager story: Python package managers are also exposed to binary compatibility

```
$ pip install lief
```

Variability modeled with:

- CPython version
- **OS and Version**
- Architecture



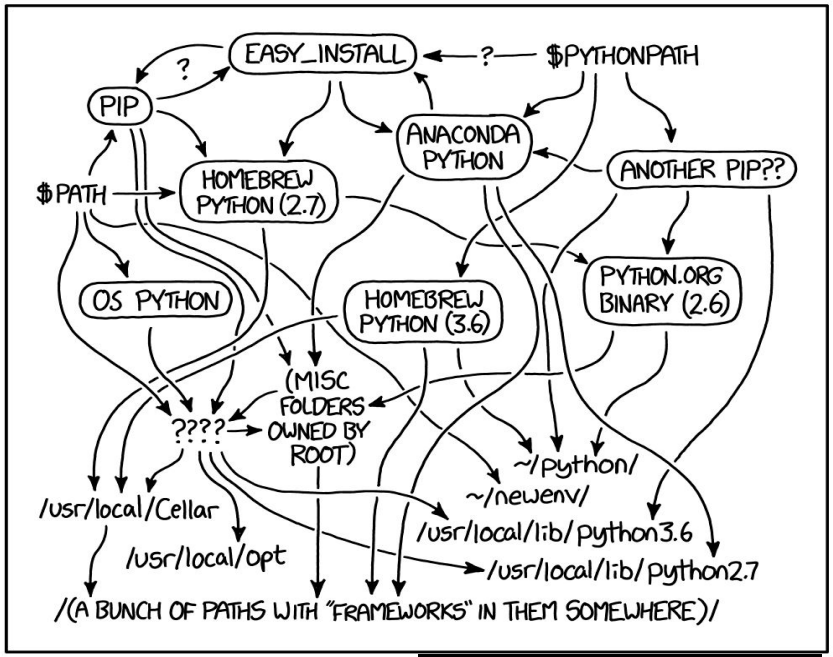
This has some implications:

- Model binary compatibility
  - Coupled with the interpreter itself
- Logic to “build from source” when consuming binaries (e.g. in case they don’t exist yet)

Binary libraries: Python

- Some interpre
  can make calls
- The CPython i
- This complicat
  story: Python p
  also exposed t

```
$ pip install lie
```



https://xkcd.com/1987/

- **OS and V**
- Architecture

Built Distributions

_amd64.whl (3.1 MB view hashes)

32.whl (2.5 MB view hashes)

implications:

ary compatibility
pled with the interpreter
lf
build from source" when
g binaries (e.g. in case
t exist yet)

ylinux_2_24_x86_64.whl (4.1 MB view hashes)

lief-0.13.0-cp310-cp310-manylinux2014_aarch64.whl (4.2 MB view hashes)

Uploaded Apr 14 2023 cp310

JFrog | CONAN C/C++ Package Manager

Binary librar

- Some interp
  can make ca
- The CPython
- This complic
  story: Python
  also exposed

```
$ pip install l
```

- Architecture

implications:

ary compatibility
upled with the interpreter
lf
build from source" when
g binaries (e.g. in case
t exist yet)

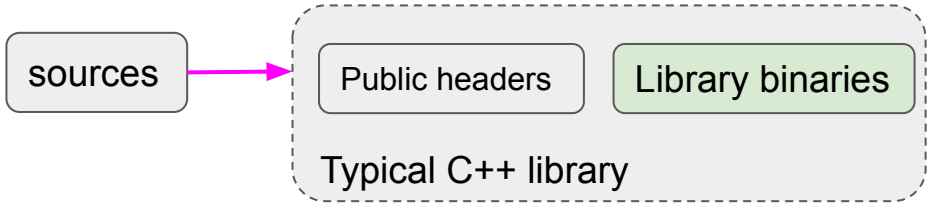
amd64.whl (3.1 MB view hashes)

ylinux_2_24_x86_64.whl (4.1 MB view hashes)

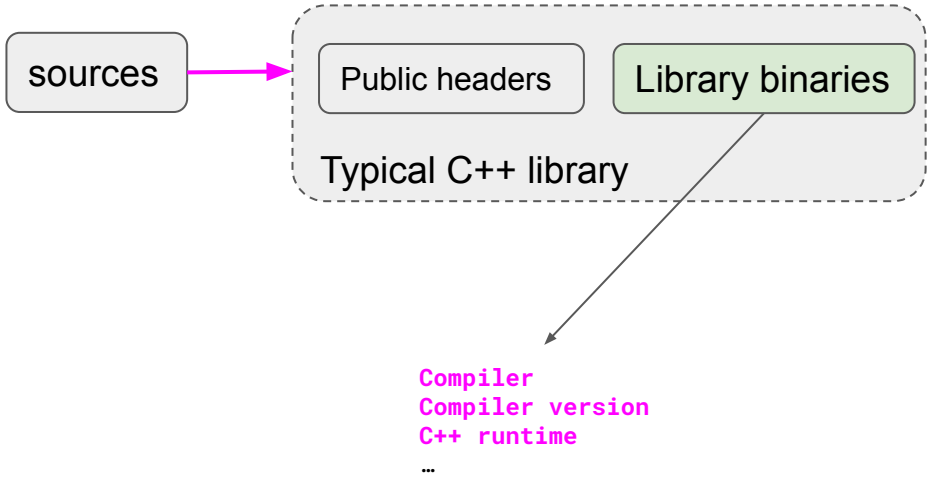
lief-0.13.0-cp310-cp310-manylinux2014_aarch64.whl (4.2 MB view hashes)

JFrog CONAN C/C++ Package Manager

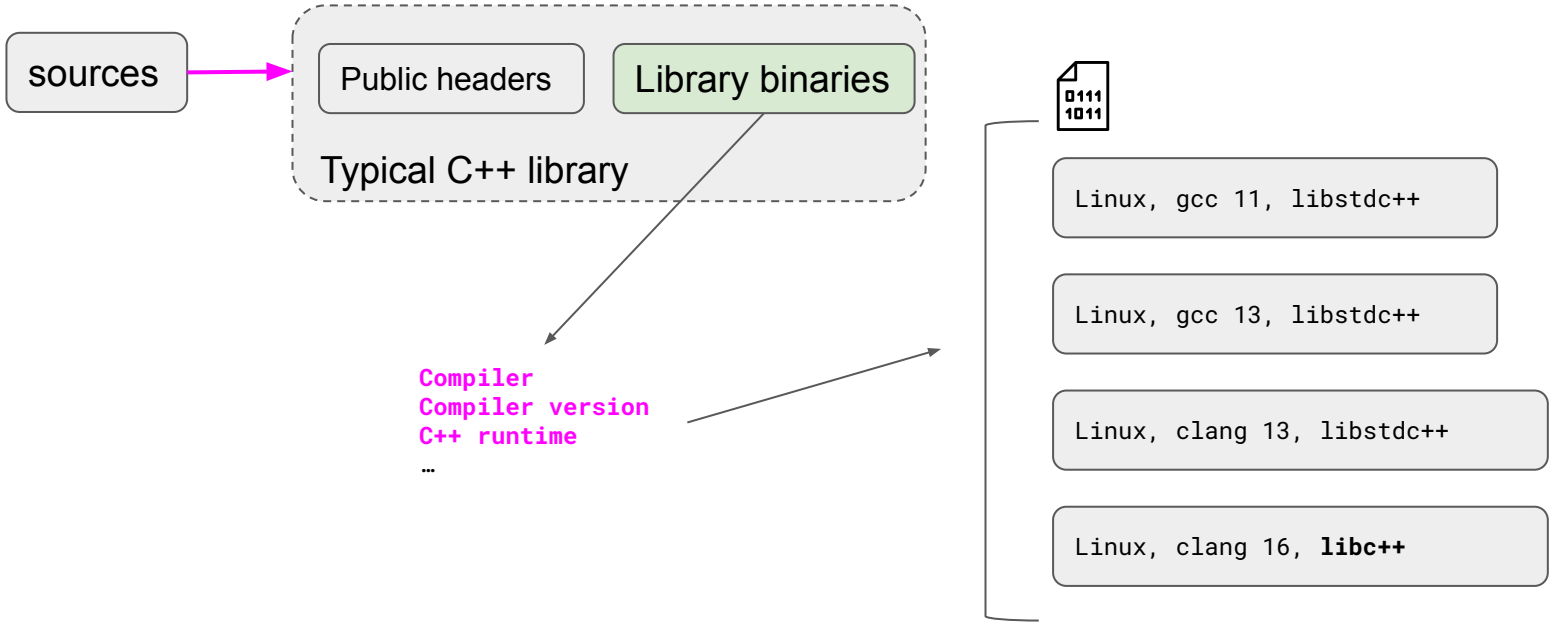
Binary libraries: C++



Binary libraries: C++

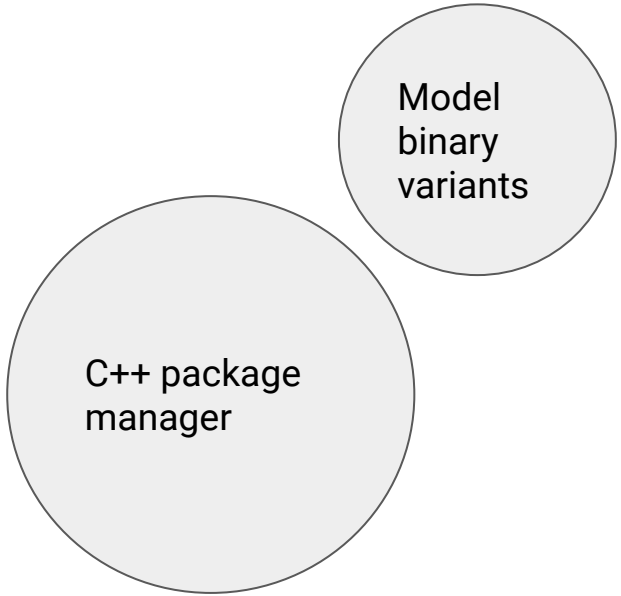


Binary libraries: C++

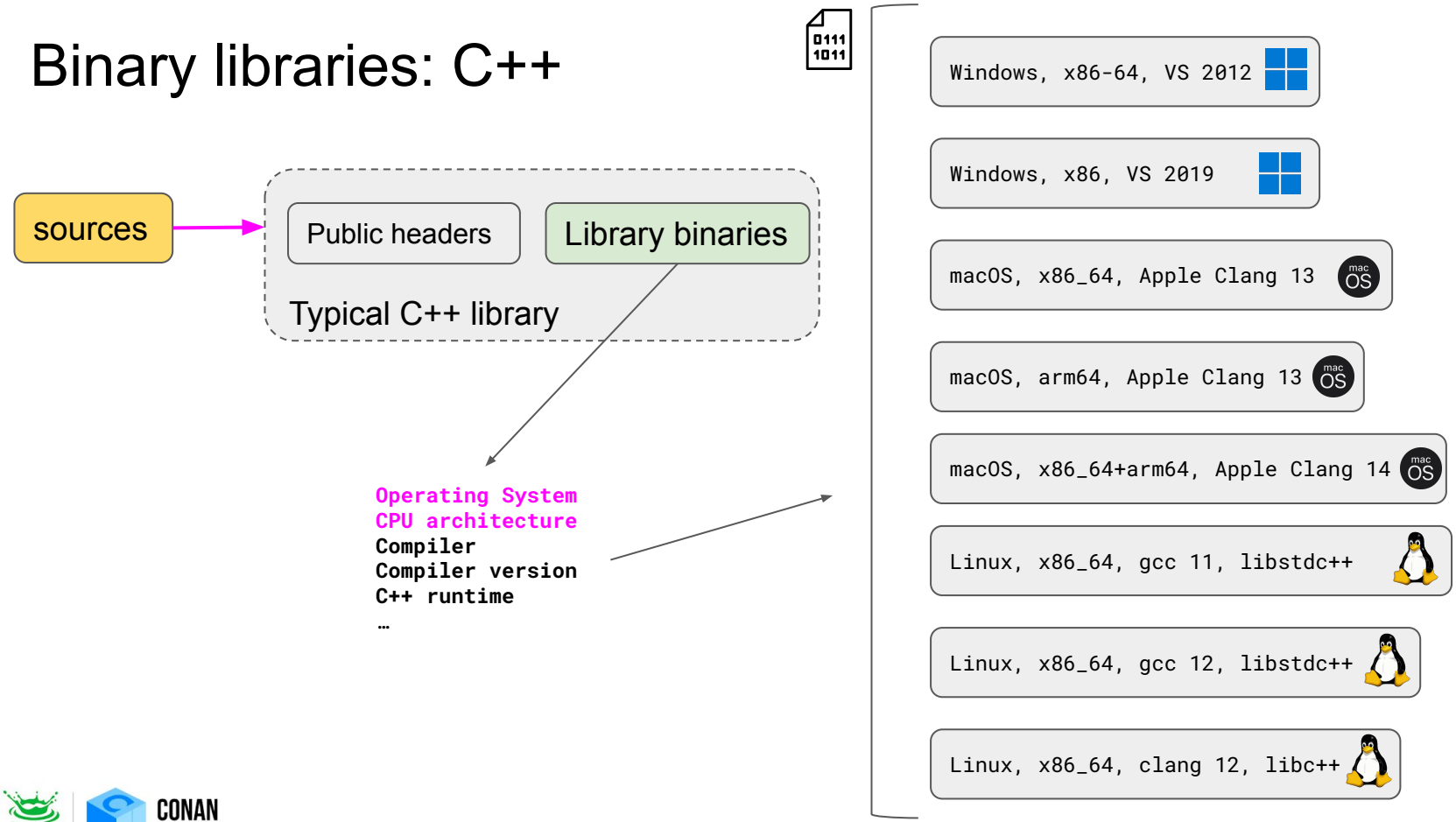


A C++ package manager

Wishlist

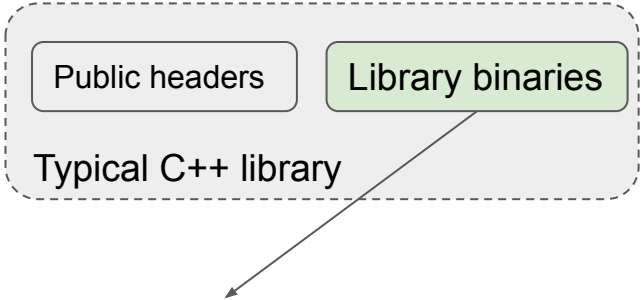


Binary libraries: C++



JFrog CONAN C/C++ Package Manager

Binary libraries: C++ Release and Debug



Operating system
CPU architecture
Compiler+version
C++ runtime
"Build type"
…

Typical example:

- Release: `-O3 -DNDEBUG`
- Debug: `-g`

Windows, x86-64, VS 2012 Release	Windows, x86-64, VS 2012 Debug
Windows, x86, VS 2019 Release	Windows, x86, VS 2019 Debug
macOS, x86_64, AppleClang 13 Release	macOS, x86_64, AppleClang 13 Debug
macOS, arm64, AppleClang 13 Release	macOS, arm64, AppleClang 13 Debug
macOS, x86_64+arm64, AppleClang 14 Release	macOS, x86_64+arm64, AppleClang 14 Debug
Linux, x86_64, gcc 11, libstdc++ Release	Linux, x86_64, gcc 11, libstdc++ Debug
Linux, x86_64, gcc 12, libstdc++ Release	Linux, x86_64, gcc 12, libstdc++ Debug
Linux, x86_64, clang 12, libc++ Release	Linux, x86_64, clang 12, libc++ Debug

JFrog CONAN C/C++ Package Manager

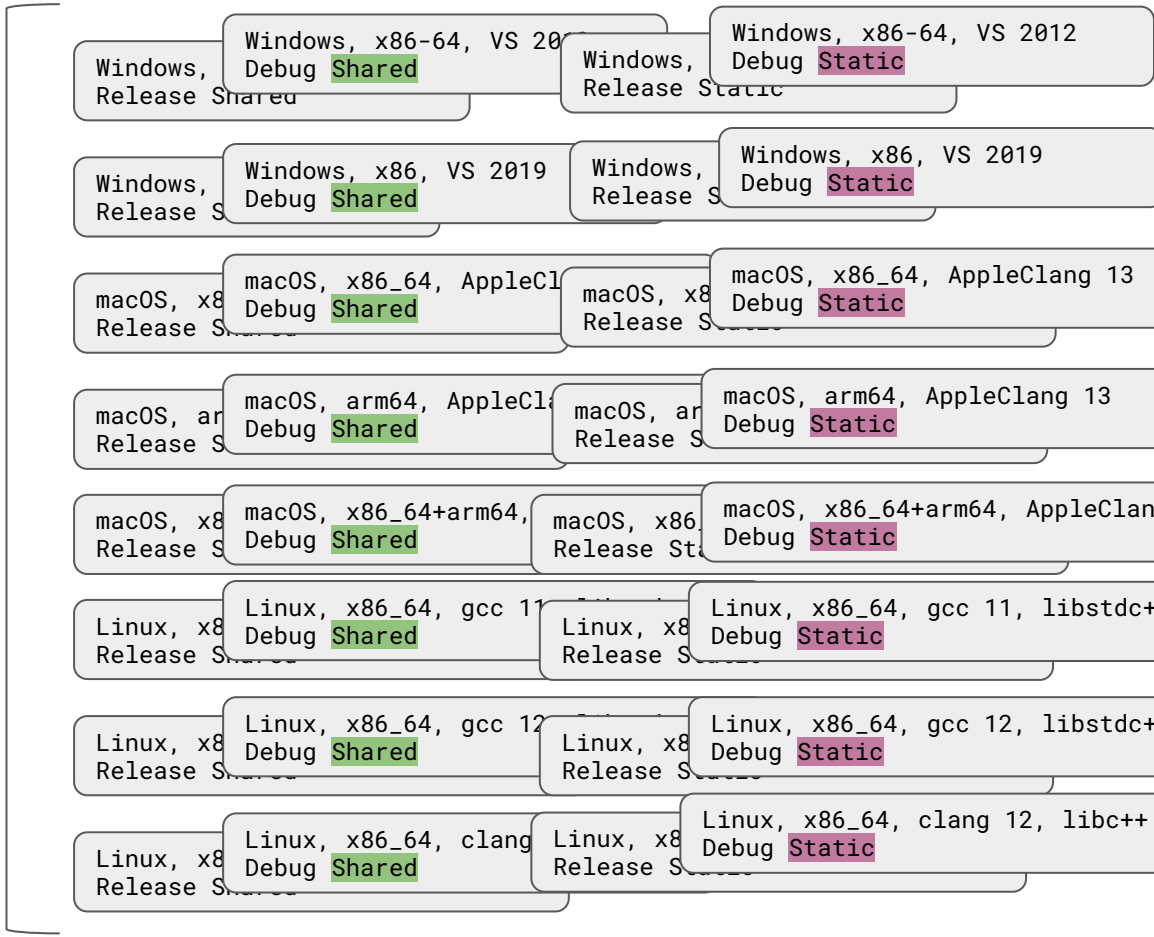
Binary libraries: C++ Static and Shared

Public headers

Library binaries

Typical C++ library

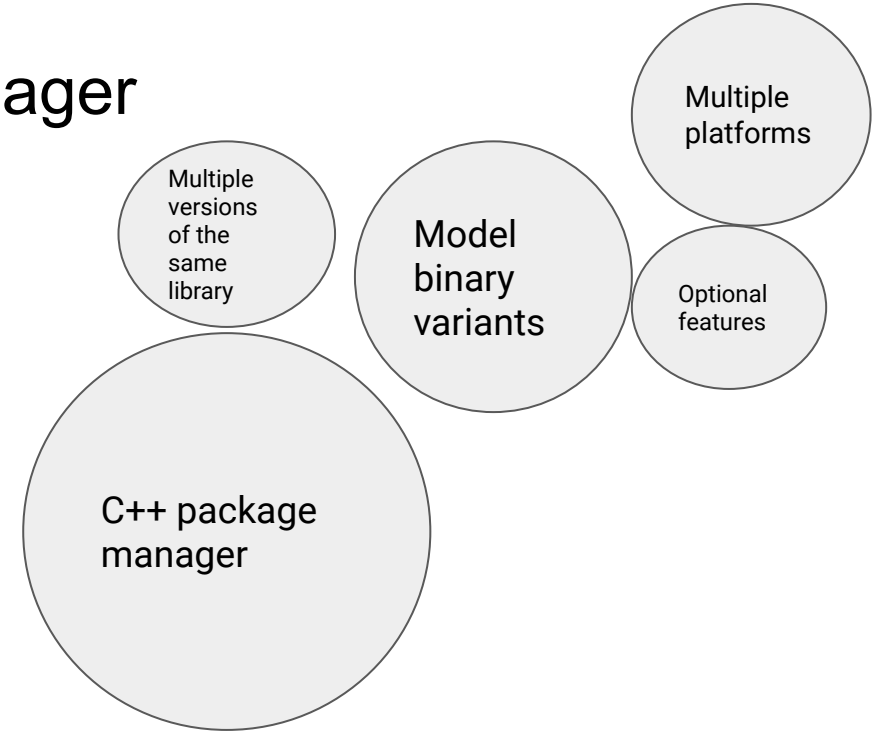
Operating system
CPU architecture
Compiler+version
C++ runtime
Debug/Release
"Static/Shared"
…



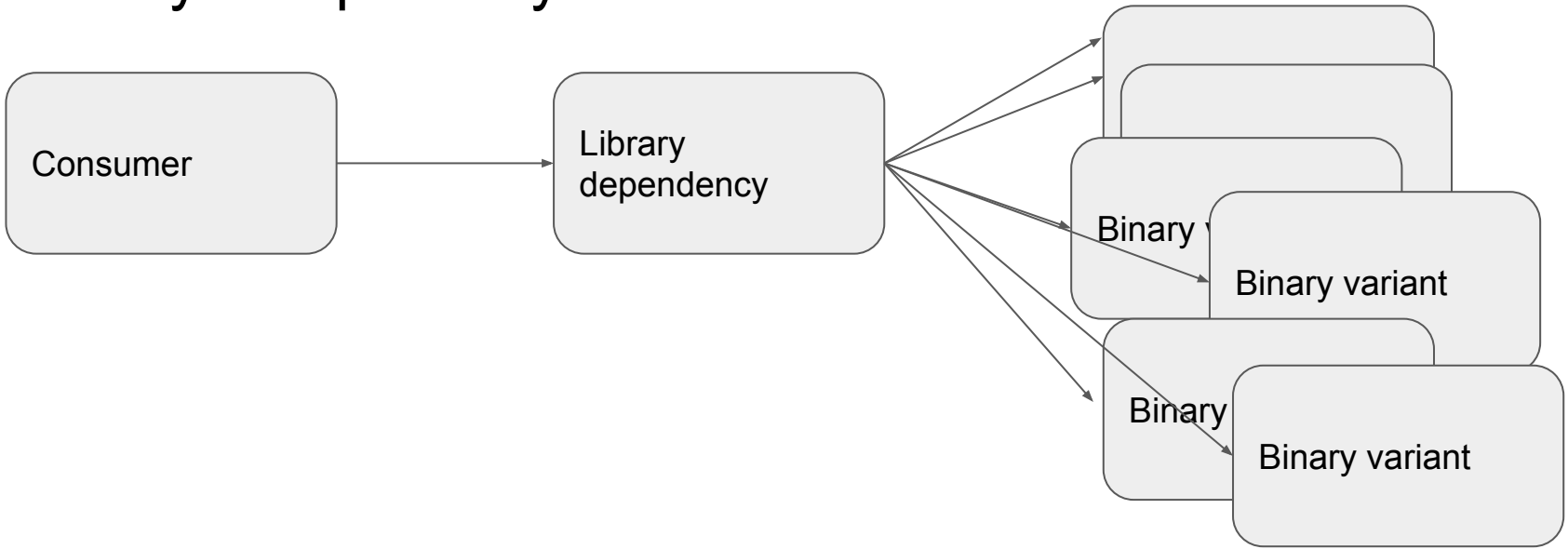
JFrog | CONAN C/C++ Package Manager

A C++ package manager

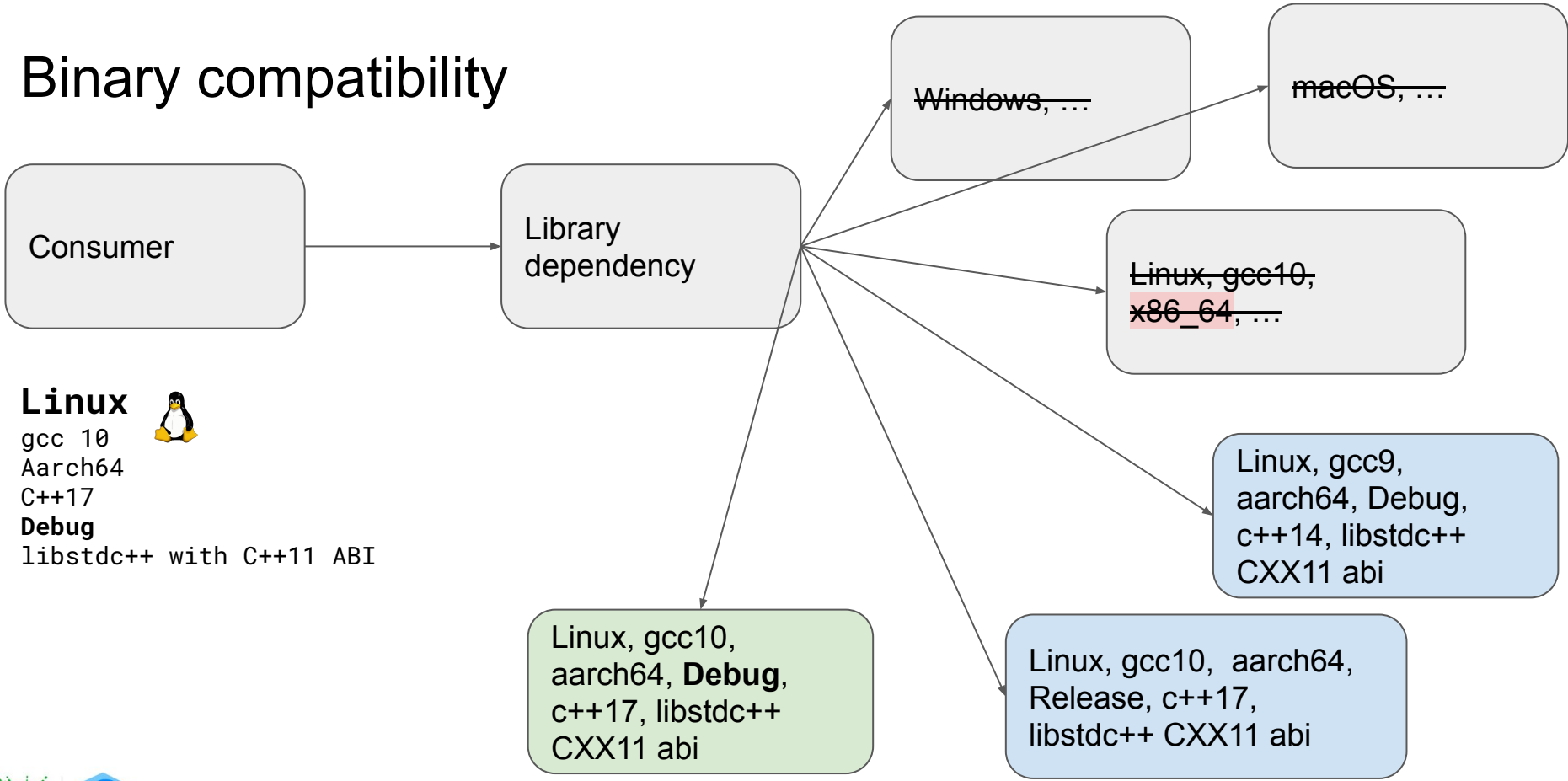
Wishlist



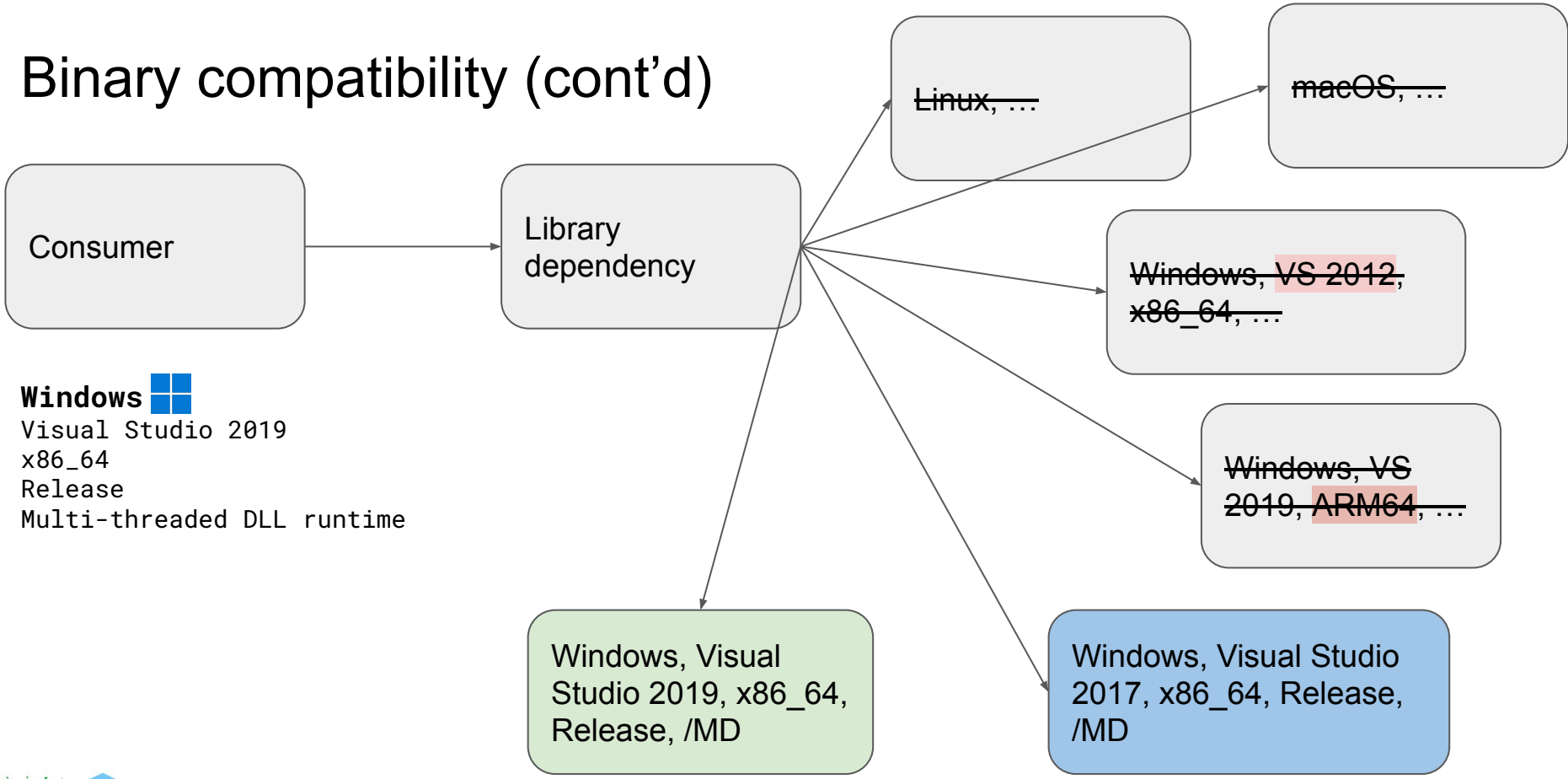
Binary compatibility



Binary compatibility



Binary compatibility (cont'd)



Windows

```
Visual Studio 2019
x86_64
Release
Multi-threaded DLL runtime
```

No one-size-fits-all solution

Binary compatibility - some use cases

- Sanitizer builds

Using instrumented libraries

It is critical that you should build all the code in your program (including libraries it uses, in particular, C++ standard library) with MSan. See MemorySanitizerLibcxxHowTo for more details.

[Memory sanitizer documentation]

[Visual C++ documentation]

Binary compatibility - some use cases

- Sanitizer builds

Using instrumented libraries

It is critical that you should build all the code in your program (including libraries it uses, in particular, C++ standard library) with MSan. See MemorySanitizerLibcxxHowTo for more details.

[Memory sanitizer documentation]

- C++ runtime

All modules passed to a given invocation of the linker must have been compiled with the same run-time library compiler option (**/MD**, **/MT**, **/LD**).

[Visual C++ documentation]

Binary compatibility - some use cases (cont'd)

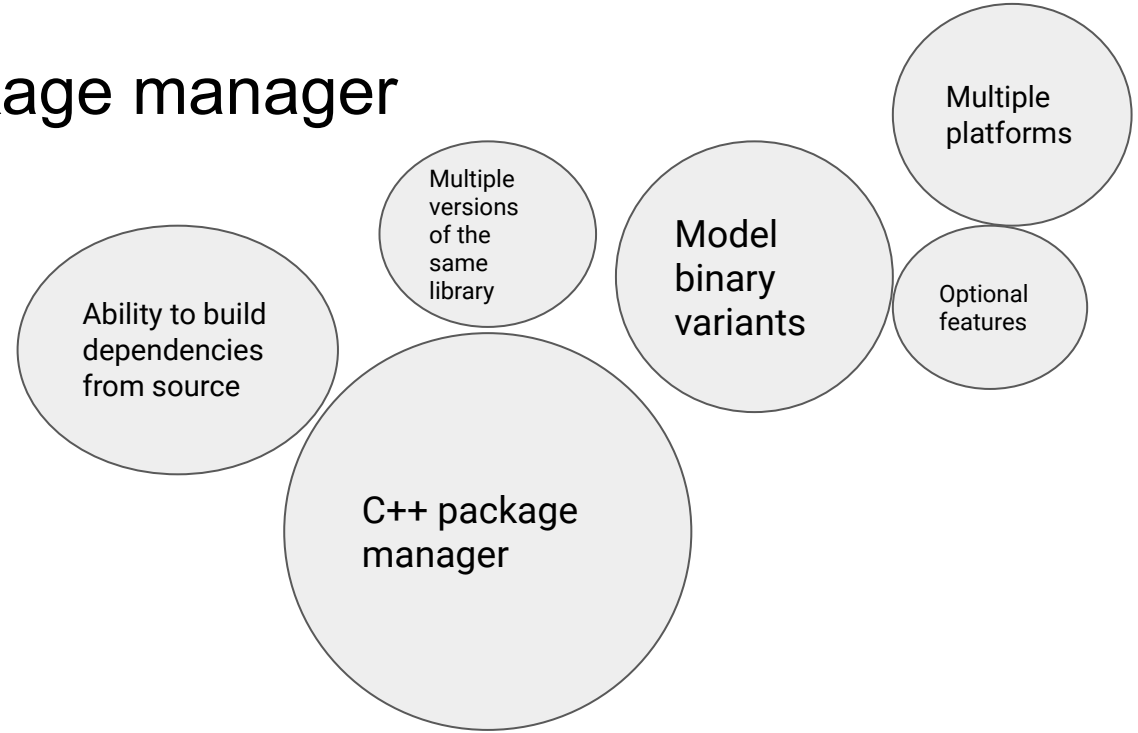
- libstdc++ C++11 ABI update

Troubleshooting

If you get linker errors about undefined references to symbols that involve types in the `std::__cxx11` namespace or the tag `[abi:cxx11]` then it probably indicates that you are trying to link together object files that were compiled with different values for the _GLIBCXX_USE_CXX11_ABI macro. This commonly happens when linking to a third-party library that was compiled with an older version of GCC. If the third-party library cannot be rebuilt with the new ABI then you will need to recompile your code with the old ABI.

A C++ package manager

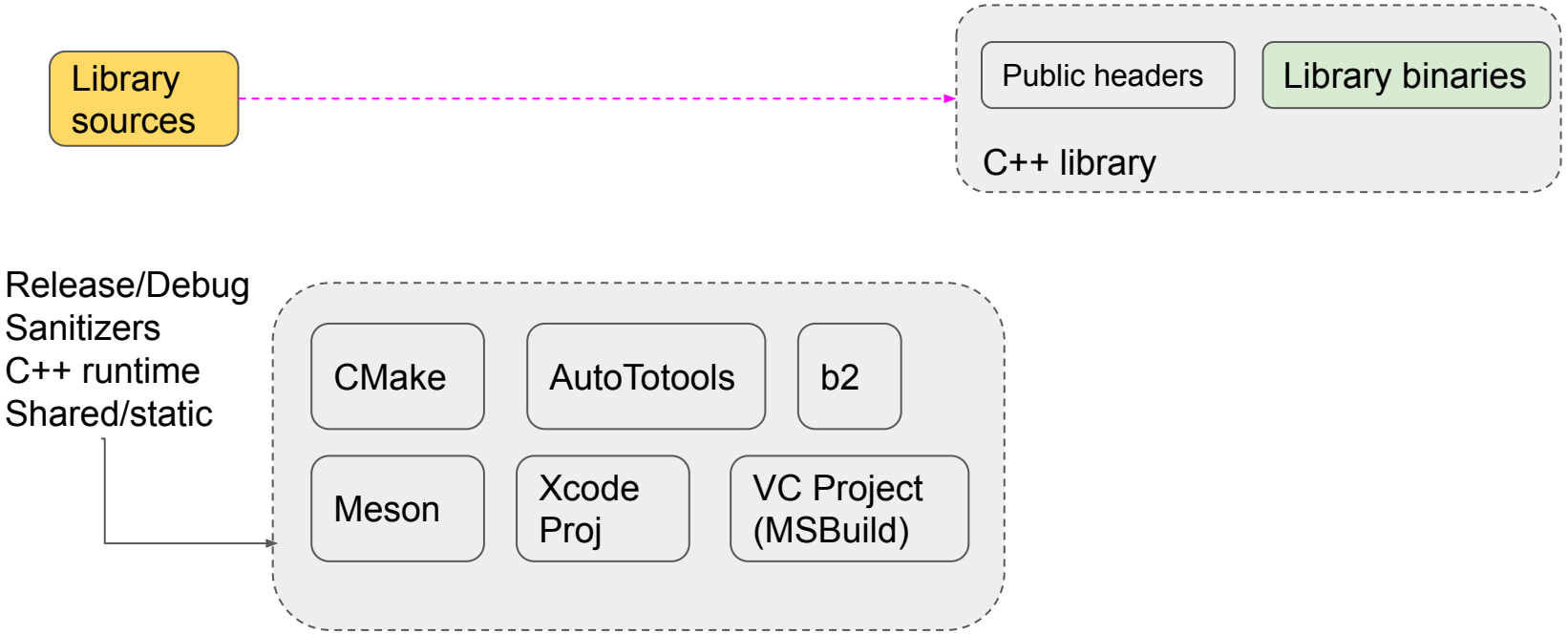
Wishlist



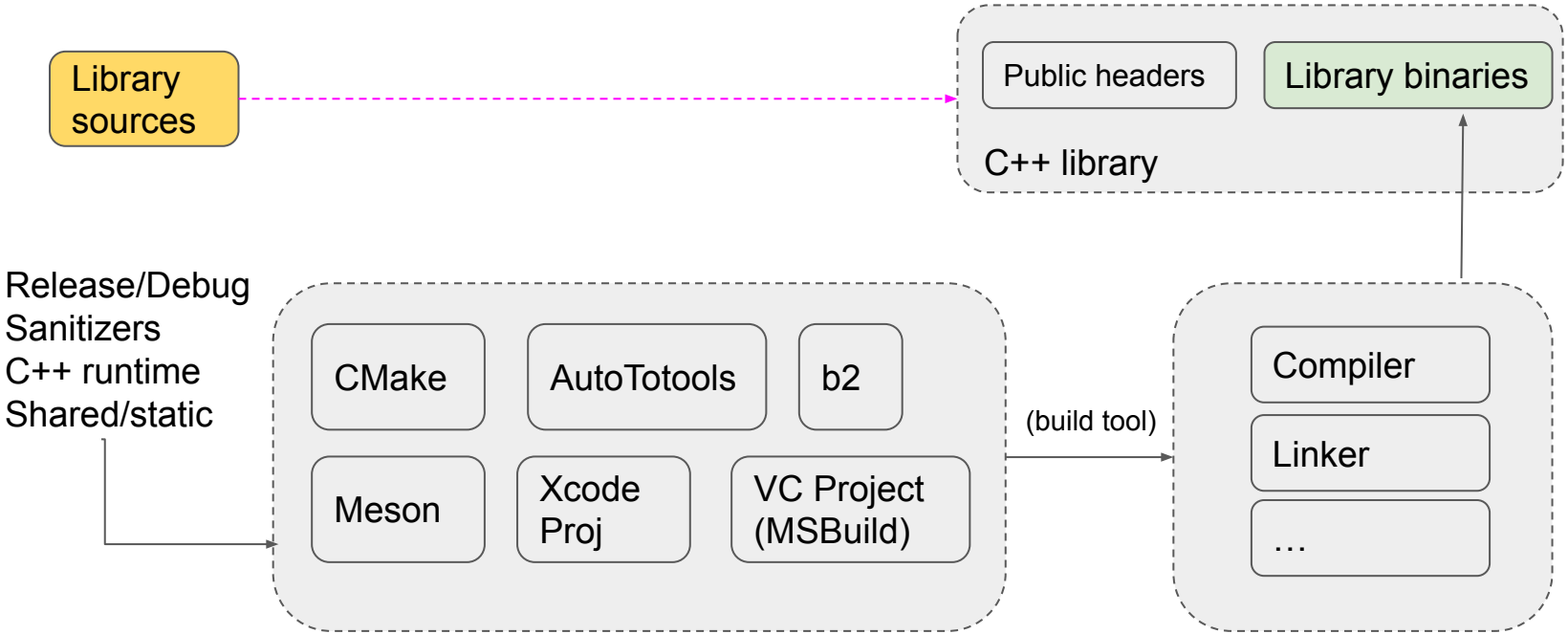
Building from source

Release/Debug
Sanitizers
C++ runtime
Shared/static

Building from source



Building from source



Building from source - some conventions

- Release vs Debug

```
cmake -DCMAKE_BUILD_TYPE=”Release”
cmake -DCMAKE_BUILD_TYPE=”Debug”

CFLAGS=”-O3 -DNDEBUG” CXXFLAGS=”-O3 -DNDEBUG” ./configure
CFLAGS=”-g” CXXFLAGS=”-g” ./configure
```

Building from source - some conventions

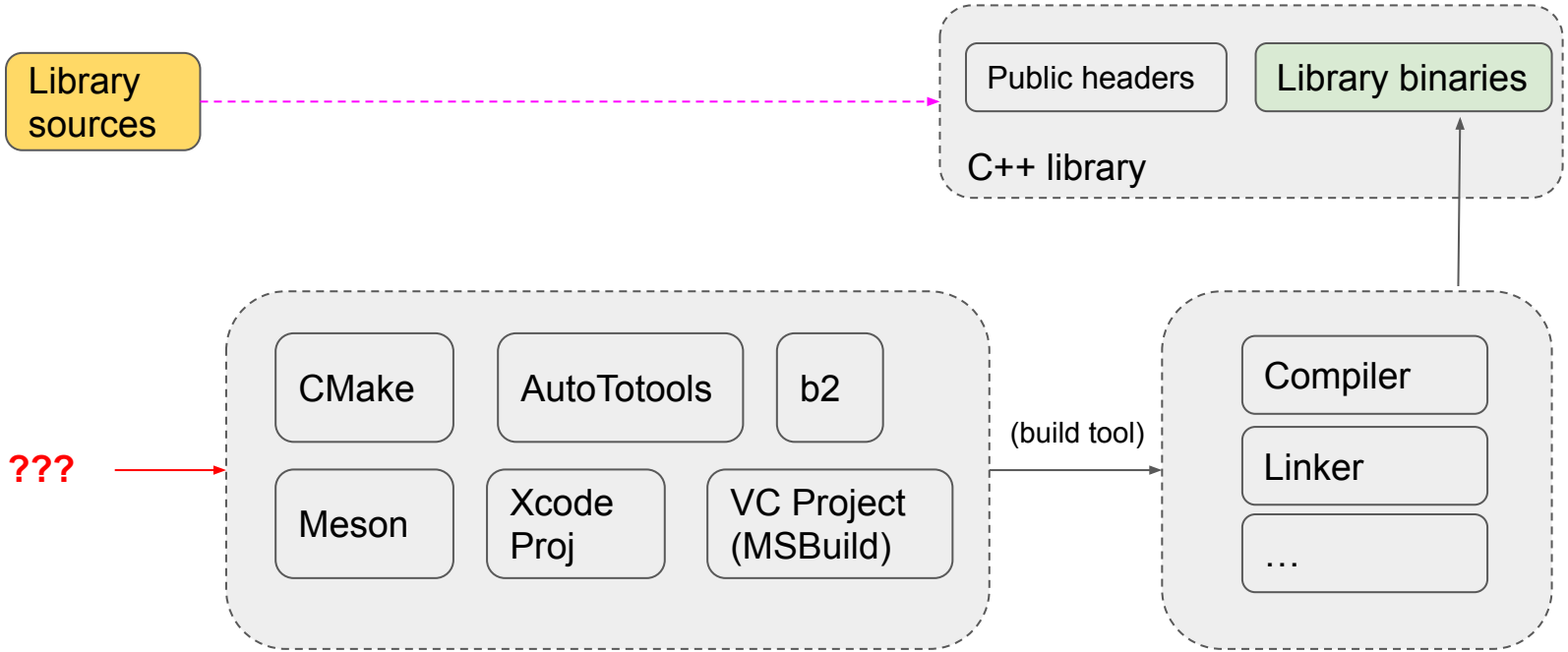
- Release vs Debug

```
cmake -DCMAKE_BUILD_TYPE=”Release”
cmake -DCMAKE_BUILD_TYPE=”Debug”

CFLAGS=”-O3 -DNDEBUG” CXXFLAGS=”-O3 -DNDEBUG” ./configure
CFLAGS=”-g” CXXFLAGS=”-g” ./configure
```

- Static vs Shared

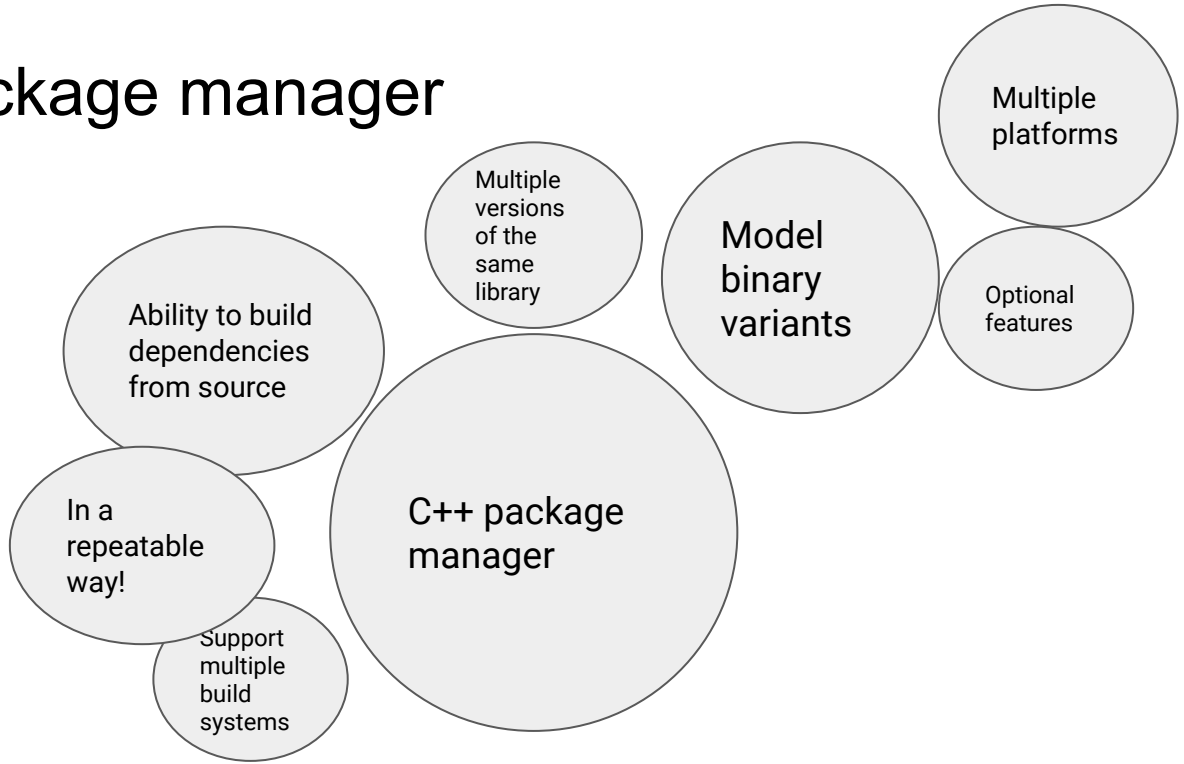
```
cmake –DBUILD_SHARED_LIBS=OFF
cmake –DBUILD_SHARED_LIBS=ON

./configure --enable-static --disable-shared
./configure --enable-shared --disable-static
```

Building from source: challenges



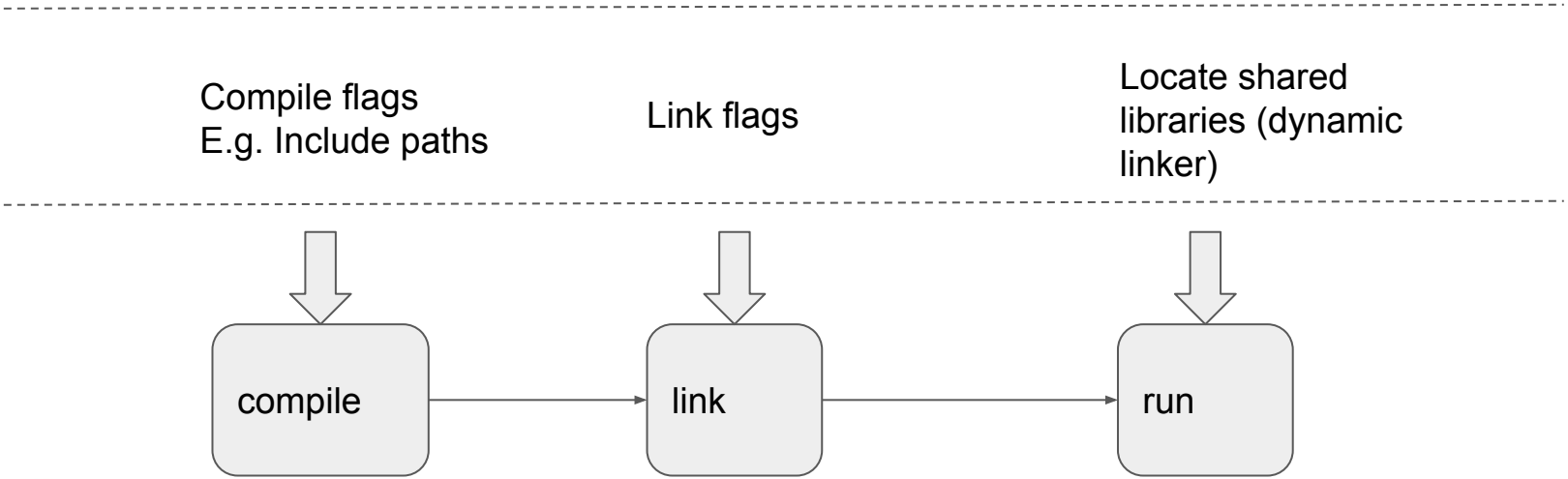
A C++ package manager

Wishlist



Usage requirements

- When consuming the libraries, the correct flags need to be propagated to the Compiler and Linker at build time, and runtime/dynamic linker needs to be able to find libraries
  - These can depend on the binary variants



Usage requirements - Link flags

	EXECUTABLE	STATIC LIBRARY	STATIC LIBRARY
Link time	`-lfoo -lbar -L/path/to/libfoo -L/path/to/libbar`	`--`	`--`
Runtime	`--`		

JFrog | CONAN C/C++ Package Manager

Usage requirements - Link flags (cont'd)

Link time	`-lfoo`
	`-L/path/to/libfoo`
	`-rpath-link`
	`/path/to/libbar`
		`-lbar`
		`-L/path/to/libbar`
			`--`

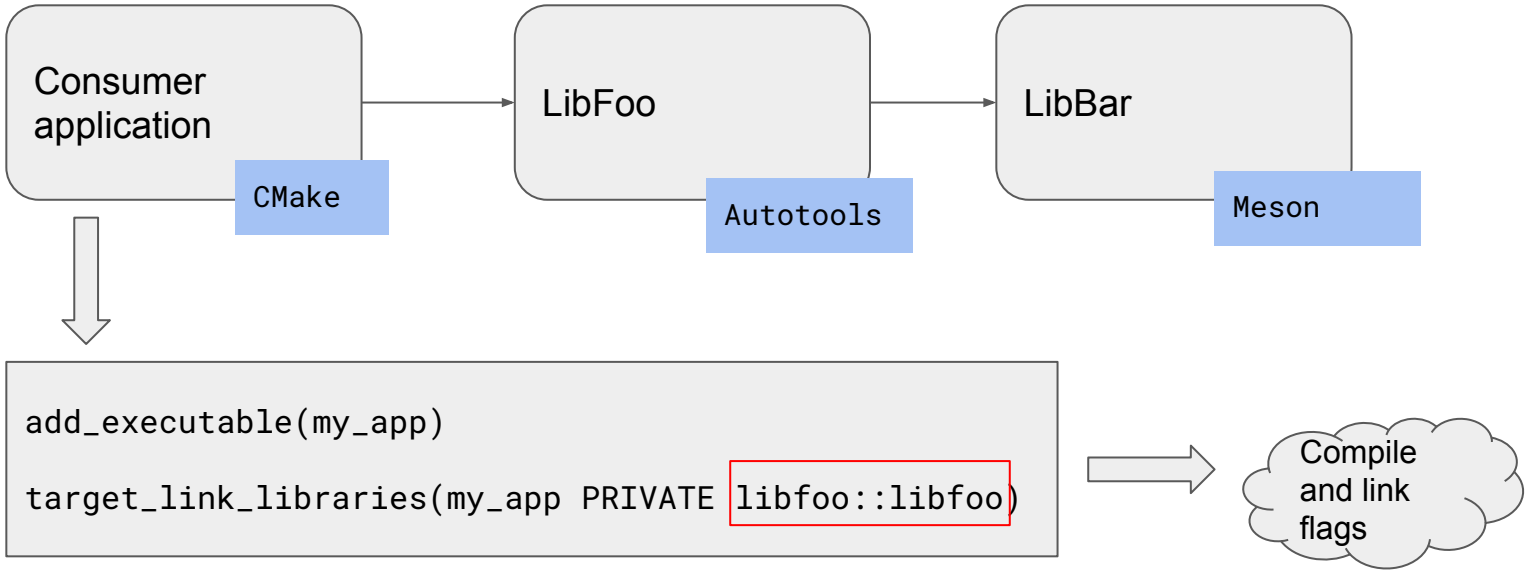
Runtime	`Locate libfoo` `Locate libbar`		

JFrog | CONAN C/C++ Package Manager

Usage requirements - Link flags (cont'd)

	EXECUTABLE	SHARED LIBRARY	STATIC LIBRARY
Link time	`-lfoo` `-L/path/to/libfoo`	`-lbar` `-L/path/to/libbar`	`--`
Runtime	`Locate foo` ~~`Locate bar`~~		

Usage requirements - flag propagation

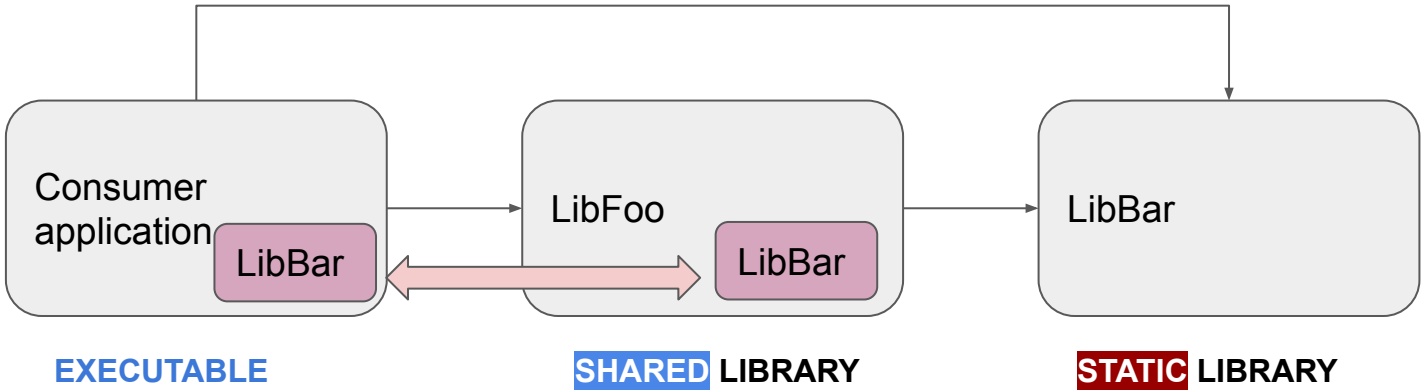


Usage requirements - Symbol visibility

	EXECUTABLE	SHARED LIBRARY	STATIC LIBRARY
Link time	`-lfoo -lbar`	`-lbar`	`--`
Runtime	`Locate foo` ~~`Locate bar`~~		

C/C++ Package Manager

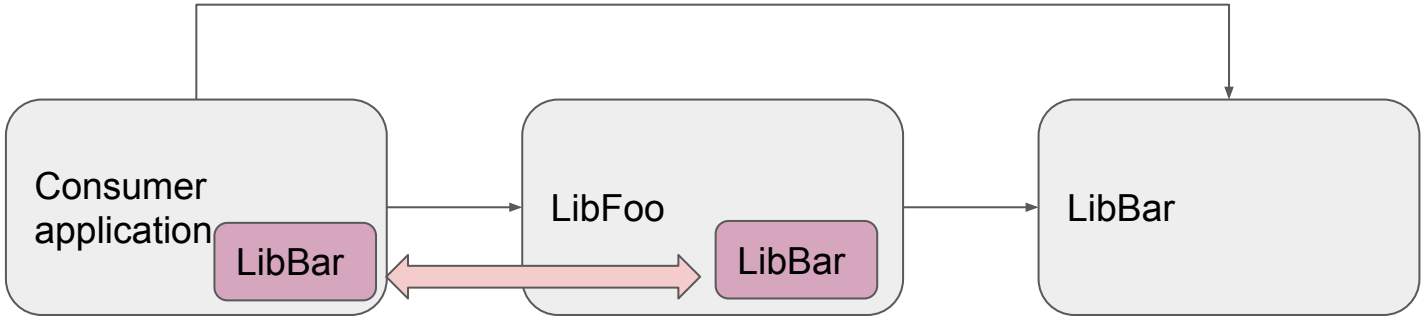
Usage requirements - Symbol visibility



	EXECUTABLE	SHARED LIBRARY	STATIC LIBRARY
Link time	`-lfoo -lbar`	`-lbar`	`--`
Runtime	`Locate foo` ~~`Locate bar`~~		

JFrog | C/C++ Package Manager

Usage requirements - Symbol visibility



A great number of compiler and linker errors experienced at the consumer side have a lot to do with dependency management.

A good dependency manager avoid these situations altogether or fail with a meaningful error

Link time `-lfoo -lbar`

Runtime `Locate foo` ~~`Locate bar`~~

C/C++ Package Manager

Dependencies and developer experience

A great number of compiler, linker and runtime issues experienced by users have more to do with handling dependencies than with the own C++ code.

Dependencies and developer experience

A great nu … by users have more to d… de.

Cannot use MySQL connector/C++ in Clion

Asked 2 days ago Modified 2 days ago Viewed 27 times

0

After configure my cmakelist file, and try to run my code to connect mysql, the error occurs:

```
erver | Debug ]================================
.ion\ch-0\231.9011.31\bin\cmake\win\x64\bin\cmake.exe --build D:\zjcfile\web
Files/server.dir/main.cpp.obj
rver.exe

\mingw64\bin\c++.exe -g  CMakeFiles/server.dir/main.cpp.obj -o server.exe -W
obj: In function `check_lib':
nnector C++ 8.0/include/jdbc/cppconn/driver.h:82: undefined reference to `ch
nnector C++ 8.0/include/jdbc/cppconn/driver.h:83: undefined reference to `ch
obj: In function `get_driver_instance_by_name':
nnector C++ 8.0/include/jdbc/mysql_driver.h:116: undefined reference to `sql
d 1 exit status
d failed.
```

Here is my `CMakeLists.txt` file:

JFrog | CONAN C/C++ Package Manager

Dependencies and developer experience (cont'd)

```
[2/2] Linking CXX executable server.exe
FAILED: server.exe
cmd.exe /C "cd . && D:\SOFTWARE\mingw64\bin\c++.exe -g  CMakeFiles/server.dir/main.cpp.obj -o server.exe
-Wl,--out-implib,libserver.dll.a -Wl,--major-image-version,0,--minor-image-version,0
-LC:/PROGRA~1/MySQL/MYSQLC~1.0/lib64/vs14 -lmysqlcppconn  -lkernel32 -luser32 -lgdi32 -lwinspool -lshell32
-lole32 -loleaut32 -luuid -lcomdlg32 -ladvapi32 && cd ."
CMakeFiles/server.dir/main.cpp.obj: In function `check_lib':
C:/Program Files/MySQL/MySQL Connector C++ 8.0/include/jdbc/cppconn/driver.h:82: undefined reference to
`check(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)'
C:/Program Files/MySQL/MySQL Connector C++ 8.0/include/jdbc/cppconn/driver.h:83: undefined reference to
`check(std::map<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
std::less<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >,
std::allocator<std::pair<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
const, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > > const&)'
CMakeFiles/server.dir/main.cpp.obj: In function `get_driver_instance_by_name':
C:/Program Files/MySQL/MySQL Connector C++ 8.0/include/jdbc/mysql_driver.h:116: undefined reference to
`sql::mysql::_get_driver_instance_by_name(char const*)'
collect2.exe: error: ld returned 1 exit status
ninja: build stopped: subcommand failed.
```

Dependencies and developer experience (cont'd)

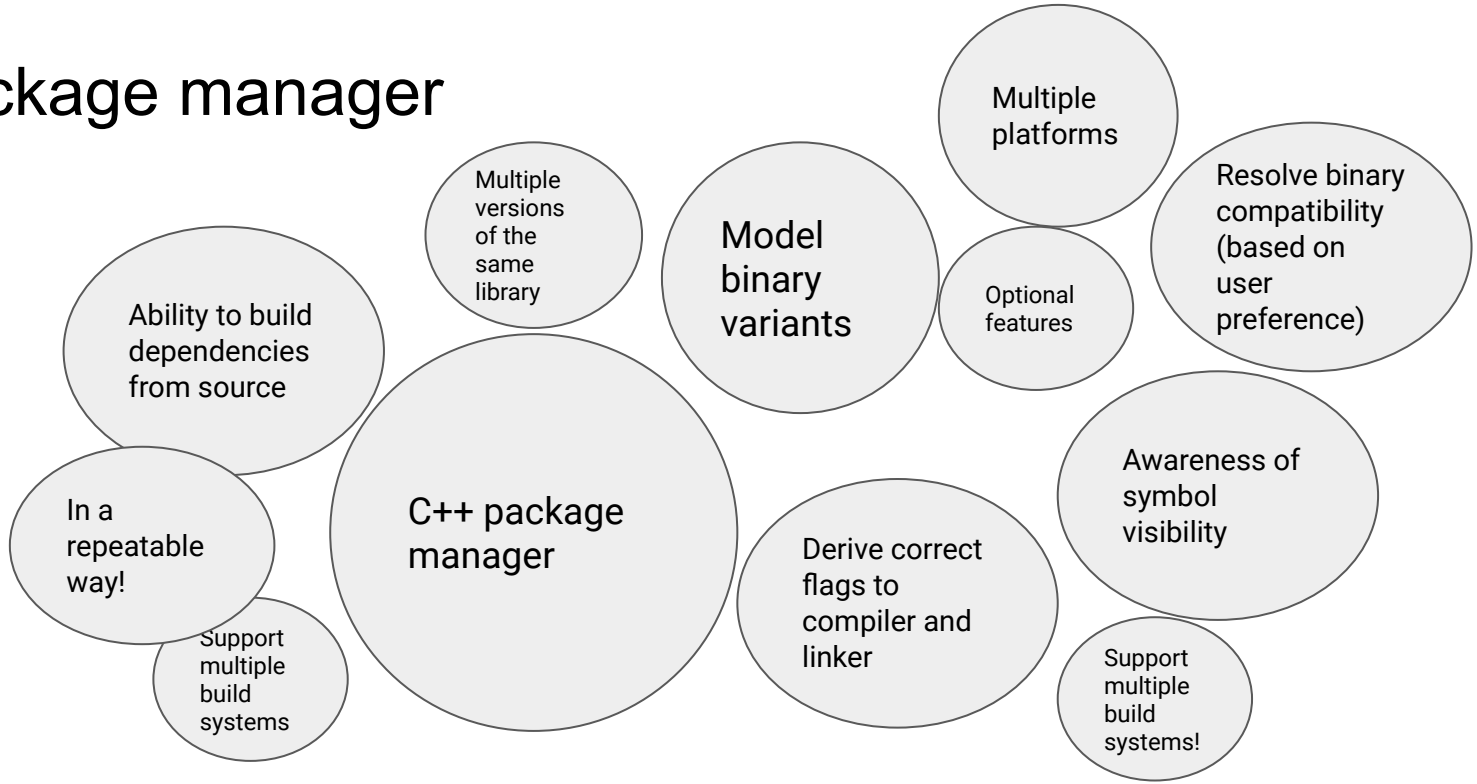
The actual error:

> "You are using a dependency built for a different C++ runtime (msvc14) than the one you are using (mingw64)"

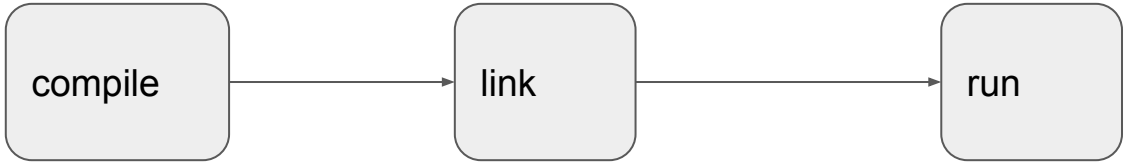
What the user sees:

```
undefined reference to `check(std::map<std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> >, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
>, std::less<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >,
std::allocator<std::pair<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
> const, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > > > >
const&)'
```

A C++ package manager



Locating shared libraries … at runtime




```
./myapp: error while loading shared libraries: libsomething.so.1: cannot
open shared object file: No such file or directory
```

Locating shared libraries … at runtime

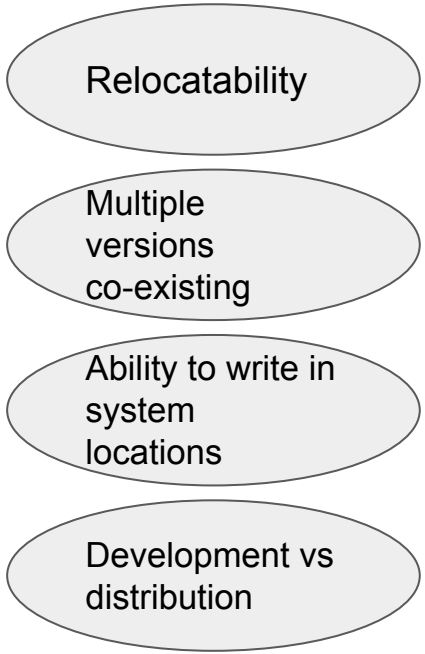
Runtime / Dynamic linker library search

Linux / macOS / other *nix:

- Pre-determined system locations, e.g. `/usr/lib`, `/usr/local/lib`
- `LD_LIBRARY_PATH / DYLD_LIBRARY_PATH`
- `RPATH/RUNPATH`
  - Embedded in the binaries
  - Absolute or relative (to `$ORIGIN, @loader_path, …`)
- Other, e.g. ld.conf

Windows:

- Relative to executable
- `PATH` environment variable
- System locations



Locating shared libraries … at runtime (cont'd)

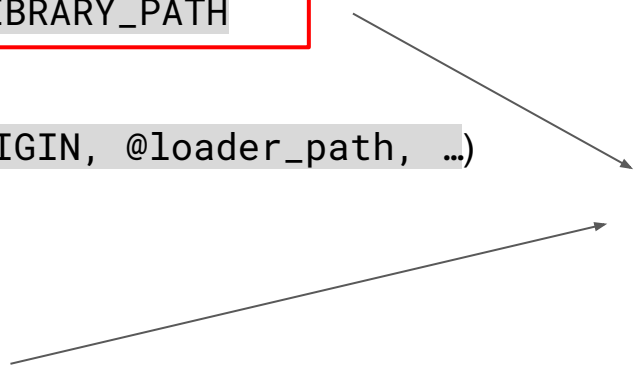
Runtime / Dynamic linker library search

Linux / macOS / other *nix:

- Pre-determined system locations, e.g. `/usr/lib`, `/usr/local/lib`
- `LD_LIBRARY_PATH / DYLD_LIBRARY_PATH`
- `RPATH/RUNPATH`
  - Embedded in the binaries
  - Absolute or relative (to `$ORIGIN`, `@loader_path`, …)
- Other, e.g. ld.conf

Windows:

- Relative to executable
- `PATH` environment variable
- System locations



Approach:

Tell the linker at runtime where to find the libraries

Locating shared libraries … at runtime (cont'd)

Runtime / Dynamic linker library search

Linux / macOS / other *nix:

- Pre-determined system locations, e.g. `/usr/lib`, `/usr/local/lib`
- `LD_LIBRARY_PATH / DYLD_LIBRARY_PATH`
- `RPATH/RUNPATH`
  - Embedded in the binaries
  - Absolute or relative (to `$ORIGIN, @loader_path, …`)
- Other, e.g. ld.conf

Windows:

- Relative to executable
- `PATH` environment variable
- System locations

Approach:

Embed in the top-level executable where to locate libraries

(your mileage may vary)

Locating shared libraries … at runtime (cont'd)

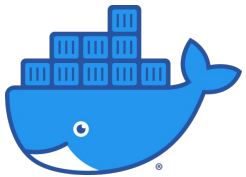
Runtime / Dynamic linker library search

Linux / macOS / other *nix:

- Pre-determined system locations, e.g. `/usr/lib`, `/usr/local/lib`
- `LD_LIBRARY_PATH / DYLD_LIBRARY_PATH`
- `RPATH/RUNPATH`
  - Embedded in the binaries
  - Absolute or relative (to `$ORIGIN, @loader_path, …`)
- Other, e.g. ld.conf

Windows:

- Relative to executable
- `PATH` environment variable
- System locations



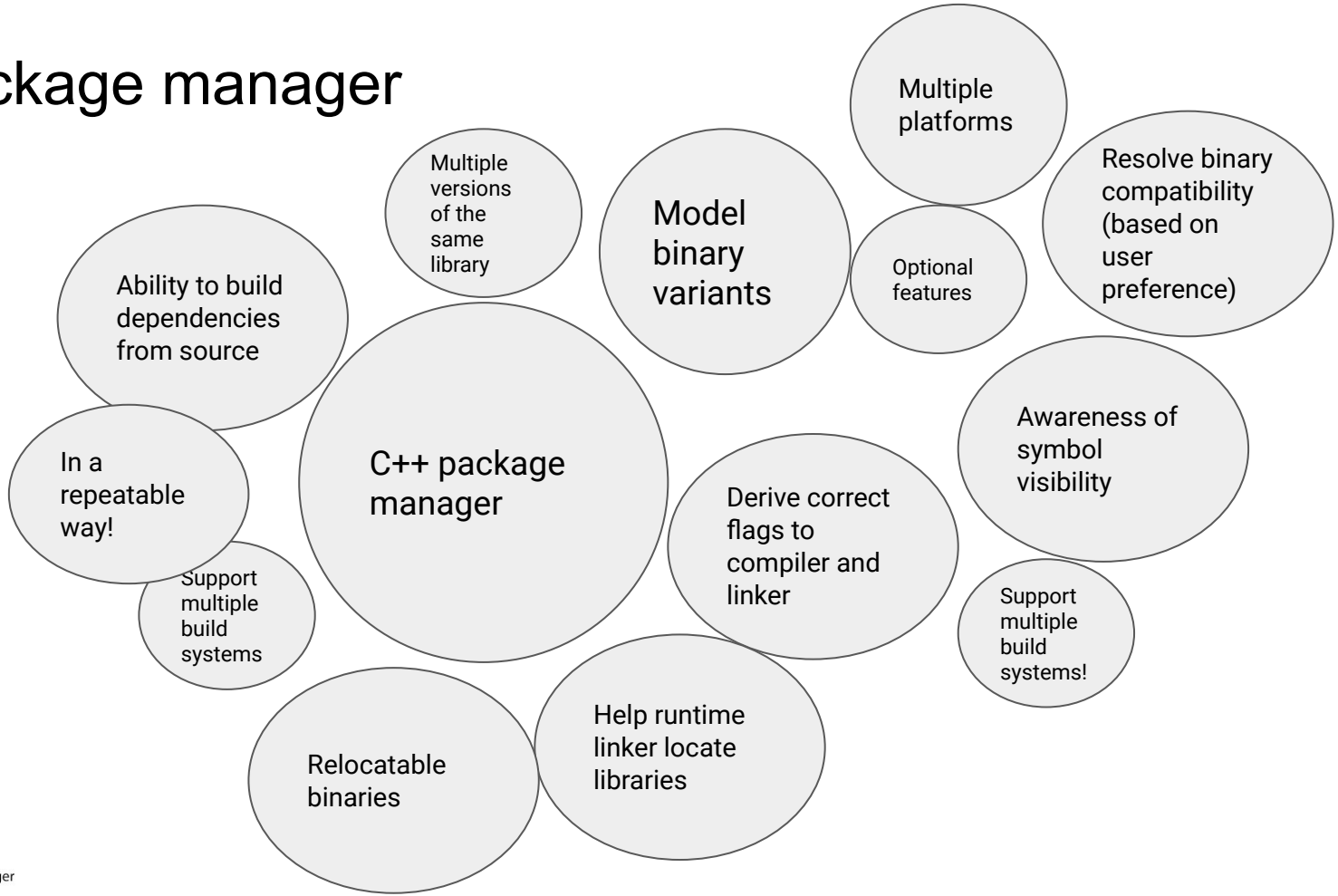
Approach:

Use Linux kernel capabilities to expose a different filesystem to the processes we launch

So that the libraries in these system locations are different…

… And throw an entire Linux distro filesystem at it

A C++ package manager



JFrog | CONAN C/C++ Package Manager

Prebuilt binaries approach

Good example: Linux distro package managers

```
# cat /etc/apt/sources.list
# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://ports.ubuntu.com/ubuntu-ports/ jammy main restricted
# deb-src http://ports.ubuntu.com/ubuntu-ports/ jammy main restricted

# dpkg --print-architecture
arm64
```

Linux
arm64
GCC 11
Libstdc++

Libraries in /usr/lib/aarch64-linux-gnu
(the /usr prefix)

Prebuilt binaries approach (cont'd)

⭐ Easy to install packages

⭐ Readily available compiled binaries

⭐ Guaranteed compatibility with same version of gcc installed in the system

⭐ Versions of libraries known to work together

🟠 Not trivial to build different variants from source (e.g. sanitizers)

🟠 Not trivial (but not impossible!) to get different (newer) versions of libraries

🟠 If we want to support other OSs (macOS, Windows) we need a different workflow for each platform

🟠 Limited to specific compiler and version and its compatibility constraints (depends on distro)

Building from source

We may wish to only deliver the “instructions” to retrieve sources and build libraries locally on the machine that need them (formulas, recipes, ports, source packages, ..)

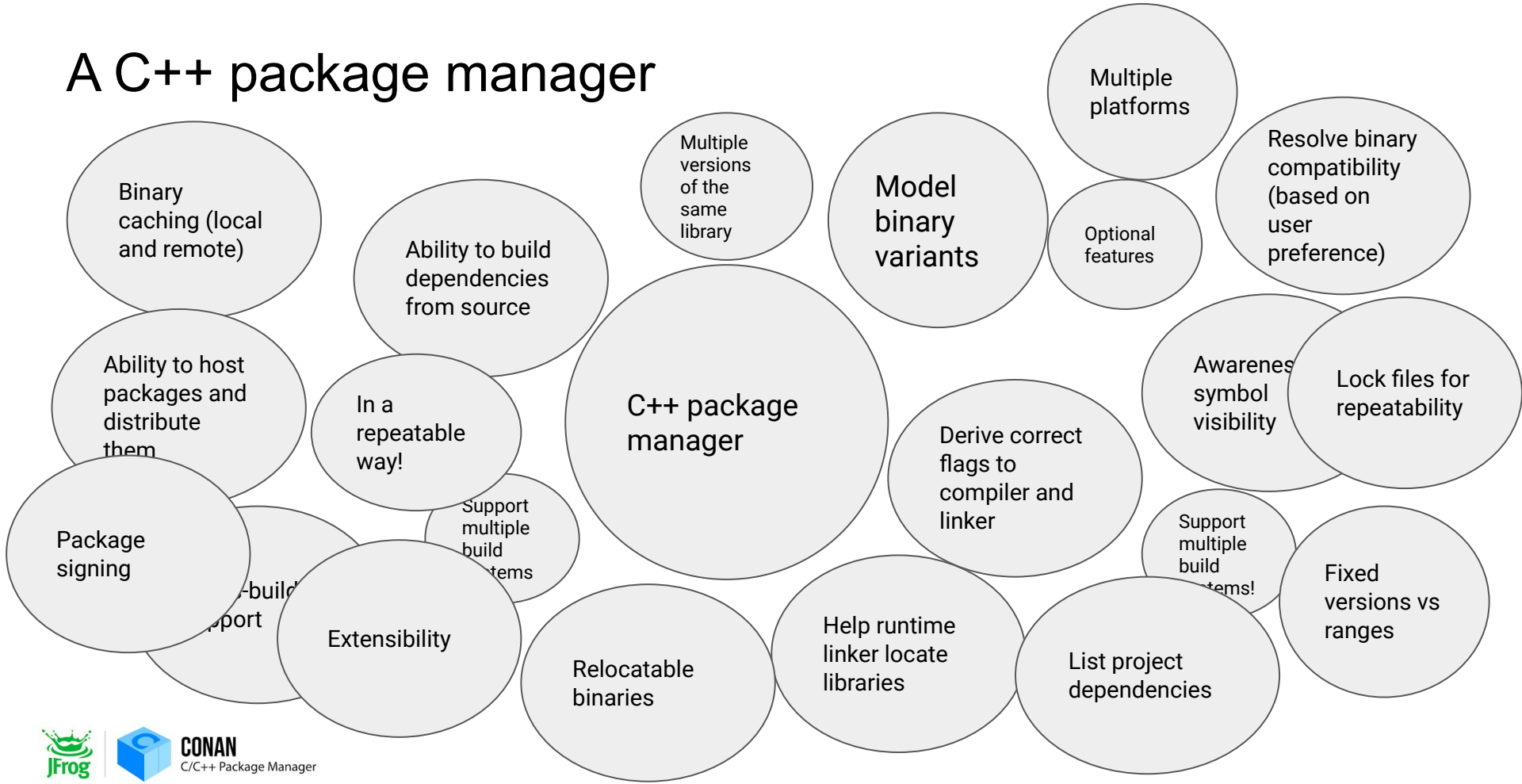
⭐ Support for multiple platforms (different distros, Linux/macOS/Windows, ..)

⭐ Built binaries should always match the configuration (compiler, version, runtime) that the **developer wants on the consumer side**

🟠 Downsides: build times

🟠 Risk of different results on different systems (non-hermetic builds)

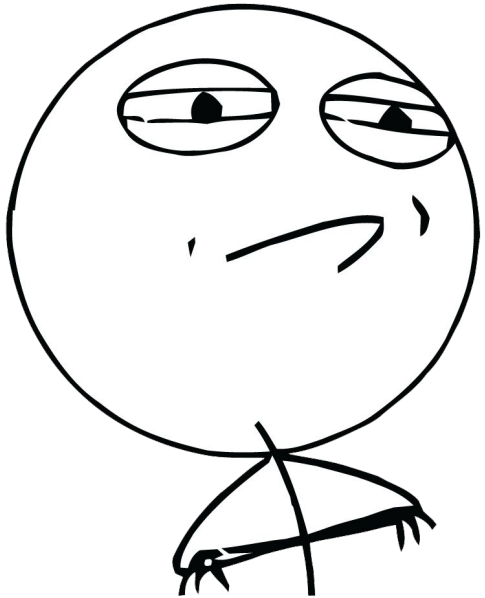
A C++ package manager



JFrog | CONAN C/C++ Package Manager


WHAT DO WE WANT
A C++ PACKAGE MANAGER
WHAT DO WE WANT IT TO DO
ALL THE THINGS

CHALLENGE ACCEPTED



The C++ package manager

Binary caching (local and remote)

Ability to build dependencies from source

Multiple versions of the same library

Model binary variants

Multiple platforms

Optional features

Resolve binary compatibility (based on user preference)

Ability to host packages and distribute them

In a repeatable way!

Awarenes symbol visibility

Lock files for repeatability

Derive correct flags to compiler and linker

Support multiple build tems

Support multiple build tems!

Package signing

-build pport

Extensibility

Relocatable binaries

Help runtime linker locate libraries

List project dependencies

Fixed versions vs ranges

JFrog

CONAN
C/C++ Package Manager

THANK YOU

Questions?

 @conan_io

 @luisc_cpp

 @jcar87