ABI compatibility
is not a **MAJOR** problem

MINOR
PATCH

Javier G. Sogo

jgsogo@gmail.com
javierg@jfrog.com
@jgsogo

C++ Russia 2019

April 20th, 2019
Moscow, RU

what is this talk about?

- **ABI**, not API

API (Application Programming Interface): *A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.*

oxforddictionaries.com

It's a **contract**: language/compiler + documentation

what is this talk about?

API is defined by

- Names
- Signatures
- Declarations location
- Functionality (docs)

std::apply

Defined in header <tuple>

```
template <class F, class Tuple>
constexpr decltype(auto) apply(F&& f, Tuple&& t);
```

(since C++17)

Invoke the *Callable* object f with a tuple of arguments.

Parameters

f - *Callable* object to be invoked

t - tuple whose elements to be used as arguments to f

Return value

The value returned by f.

cppreference.com

what is this talk about?

- **ABI**, not API

ABI (Application Binary Interface): *an interface between two binary program modules. An ABI defines how data structures or computational routines are accessed in machine code, which is a low-level, hardware-dependent format.*

en.wikipedia.org

Not covered by the C++ Standard (implementation detail)

what is this talk about?

ABI can be affected by

Infrastructure (DevOps, CI, Sysadmin)

- Calling convention
- Exception handling
- Name mangling
- C++ runtime

Pablo Ruíz Picasso. *Guernica* (fragment). 1937

what is this talk about?

Pablo Ruíz Picasso. *Guernica* (fragment). 1937

ABI can be affected by

Code (Developer)

- Binary representation of types
- vtable layout
- Inheritance, namespaces, overloading,...

what is this talk about?

- **ABI compatibility**

A library is **binary compatible** if another module linked dynamically with a former version of that library continues running with newer versions without the need of recompiling.

Source compatibility: a program need to be recompiled against a new version of the library, but no further changes are needed.


Software Engineer @ JFrog
WE'RE HIRING
https://join.jfrog.com/
JFrog
JFrog

CONAN C/C++ package manager

FOSS (MIT), including in-house server

Decentralized/distributed, git-like

Build system agnostic

>2700 stars

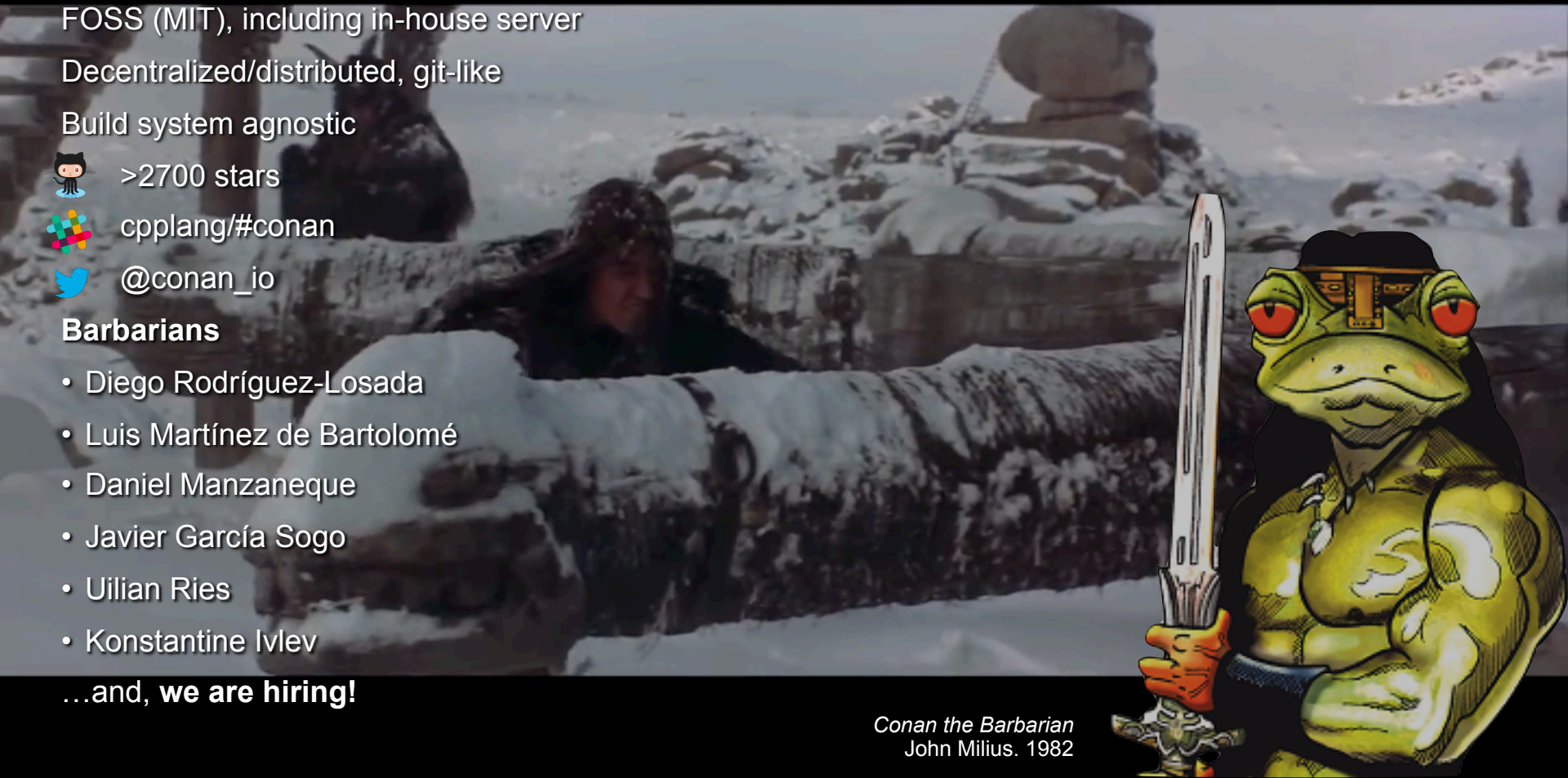
cpplang/#conan

@conan_io

Barbarians

- Diego Rodríguez-Losada
- Luis Martínez de Bartolomé
- Daniel Manzaneque
- Javier García Sogo
- Uilian Ries
- Konstantine Ivlev

…and, **we are hiring!**



Conan the Barbarian
John Milius. 1982

Madrid C/C++ User Group

@madridccppug

what is this talk about?

Pablo Ruíz Picasso. *Guernica*. 1937
Museo Nacional Centro de Arte Reina Sofía, Madrid, Spain

ABI compatibility

is ~~not~~ a **MAJOR** problem

HARD

COMPLEX

ABI compatibility, should I care?

- If you build an **application**, and
  - All your code is in the same repo, or
  - You compile always everything from sources, and
  - Do not allow plugins

- If you build a **library**
  - you DO care about ABI compatibility
  - …unless you are writing a header-only

ABI compatibility, should I care?

- If you build an **application**, and
  - All your code is in the same repo, or
  - You compile always everything from sources, and
  - Do not allow plugins

😥
😢
😭

- If you build a **library**
  - you DO care about ABI compatibility
  - …~~unless~~ also if you are writing a header-only


Challenger #1

The runtime

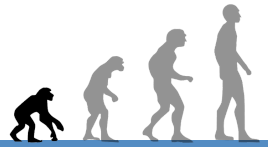
Challenger #1 - The runtime 

```c
int main( int argc, const char* argv[] )
{
    printf( "Scenario #1" );

    struct dirent **namelist;
    int n;

    n = scandir(".", &namelist, NULL, alphasort);

    if (n < 0)
        perror("scandir");
    else {
        while (n--) {
            printf("%s\n", namelist[n]->d_name);
            free(namelist[n]);
        }
        free(namelist);
    }
}
```

C plain application (C89 standard)

C-plain-app

```
gcc main.c -o c-plain-app
./c-plain-app
```



Challenger #1 - The runtime 

C standard library (libc)

- Many implementations: glibc, musl, uClibc, dietlibc,...

ABI and versioning comparison	musl	uClibc	dietlibc	glibc
Stable ABI	yes	no	unofficially	yes
LSB-compatible ABI	incomplete	no	no	yes
Backwards compatibility	yes	no	unofficially	yes
Forwards compatibility	yes	no	unofficially	no
Atomic upgrades	yes	no	no	no
Symbol versioning	no	no	no	yes

Table from musl libc (https://www.musl-libc.org/)

- “Provided” by the running distro

```
bash: ./c-plain-app: /lib/ld-musl-x86_64.so.1: bad ELF interpreter: No such file or directory
bash: ./c-plain-app: No such file or directory
./c-plain-app: /lib64/libc.so.6: version `GLIBC_2.15' not found (required by ./c-plain-app)
```

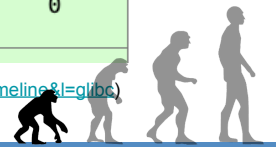
Challenger #1 - The runtime 

glibc: GNU C library

- Backwards compatible
- Compile your executables against a really old distro (centos6)
- Be careful even if it works

Version	Date	Soname	Change Log	Backward Compat.	Added Symbols	Removed Symbols
2.29	2019-01-31	0/1/2/6	changelog	99.96%	2 new	0
2.28	2018-08-01	0/1/2/6	changelog	99.92%	27 new	0
2.27	2018-02-01	0/1/2/6	changelog	98.49%	15 new	0
2.26	2017-08-02	0/1/2/6	changelog	99.54%	33 new	5 removed
2.25	2017-02-05	0/1/2/6	changelog	99.70%	22 new	0
2.24	2016-08-02	0/1/2/6	changelog	99.76%	3 new	0
2.23	2016-02-18	0/1/2/6	changelog	99.96%	3 new	0

Table from ABI Laboratory (https://abi-laboratory.pro/?view=timeline&l=glibc)



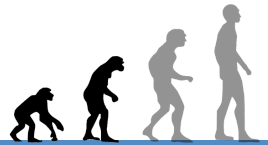
Challenger #2

The tools

Challenger #2 - Tools 

Compiler — Shit happens!

- GCC 4.7.0 and GCC 4.7.1
  - a data member was added to `std::list` (change size)
  - `std::pair`'s move constructor was not trivial (change calling convention)
- GCC has some defaults
  - C++ dialect `-fabi-version`
  - Dual ABI: `_GLIBCXX_USE_CXX11_ABI`
  - ...



Challenger #2 - Tools 

`_GLIBCXX_USE_CXX11_ABI` and GCC > 5.x

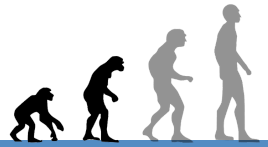
```
#define _GLIBCXX_USE_CXX11_ABI 1

#include <string>

void function(std::string input) {}
```

GCC < 5.0 function(std::basic_string<char, std::char_traits<char>, std::allocator<char> >)

GCC > 5.x function(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)

```
#define _GLIBCXX_USE_CXX11_ABI 0
```

GCC > 5.x function(std::basic_string<char, std::char_traits<char>, std::allocator<char> >)



Challenger #2 - Tools 

`_GLIBCXX_USE_CXX11_ABI` and GCC > 5.x

```cpp
#define _GLIBCXX_USE_CXX11_ABI 1

#include <string>

void function(std::string input) {}
```

```
$> cat /etc/os-release
NAME="Ubuntu"
VERSION="14.04.6 LTS, Trusty Tahr"
```

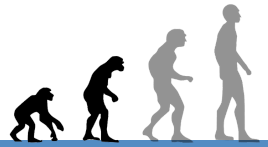
Old distro (no dual ABI)

```
$> g++ --version
g++ (Ubuntu 6.5.0-2ubuntu1~14.04.1) 6.5.0 20181026
```

New compiler

```
$> nm -gC lib.o
0000000000000000 T function(std::string)
```

No C++11 ABI



Challenger #2 - Tools 

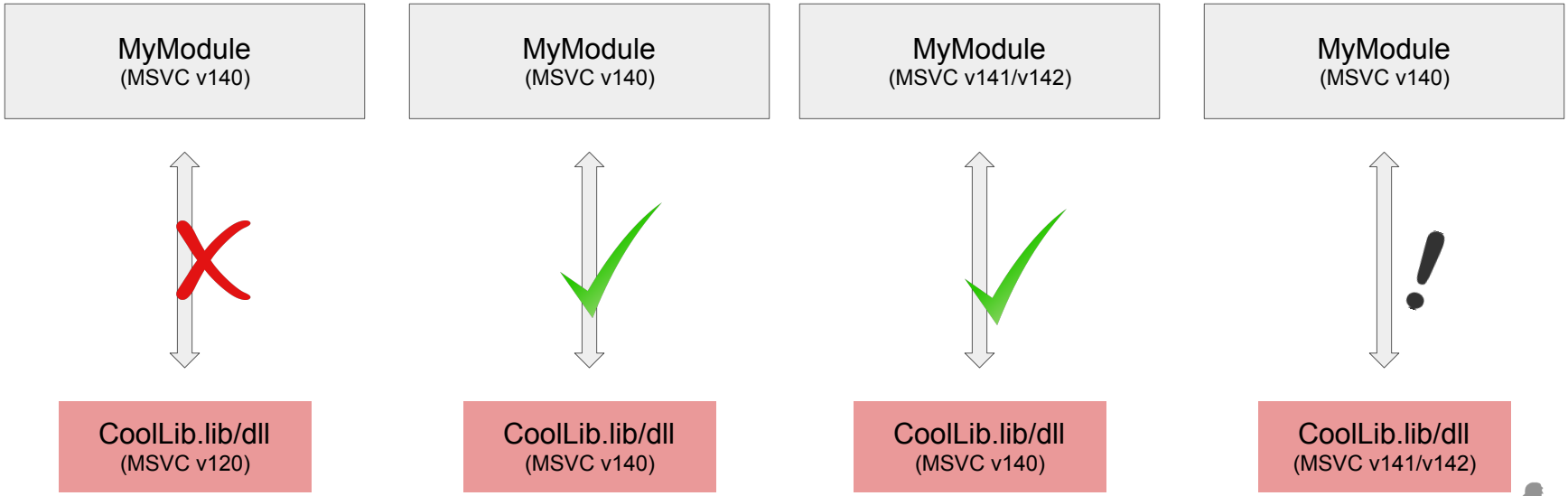
Visual Studio: *"VC Runtime in the latest MSVC v142 toolset is binary compatible with v140 and v141"*

MyModule (MSVC v140)	MyModule (MSVC v140)	MyModule (MSVC v141/v142)	MyModule (MSVC v140)
CoolLib.lib/dll (MSVC v120)	CoolLib.lib/dll (MSVC v140)	CoolLib.lib/dll (MSVC v140)	CoolLib.lib/dll (MSVC v141/v142)

Quoted: https://devblogs.microsoft.com/cppblog/cpp-binary-compatibility-and-pain-free-upgrades-to-visual-studio-2019/

Challenger #2 - Tools

Visual Studio: *"... linking all of it together (with the latest linker)..."*
"... VCRedist can't be older than any of the toolset versions used to build…"



Quoted: https://devblogs.microsoft.com/cppblog/cpp-binary-compatibility-and-pain-free-upgrades-to-visual-studio-2019/

Francisco de Goya. *El Aquelarre* o *El gran Cabrón*. 1823
Museo Nacional del Prado, Madrid (Spain)

Challenger #3

The sources

Challenger #3 - The sources 

lib.h

```
int add_1(int a = 0);

int add_2(int a = 0) {
    return a + 2;
}
```

lib.cpp

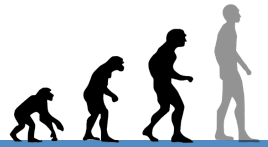
```
#include "lib.h"

int add_1(int a) {
    return a + 1;
}
```

main.cpp

```
int main(int argc, const char* argv[]) {
    std::cout << "add_1: " << add_1() << "\n";
    std::cout << "add_2: " << add_2() << "\n";
```

Compile and link shared:

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
g++ -o executable bin/main.cpp -lfoo

./executable
add_1: 1
add_2: 2
```



Challenger #3 - The sources 

lib.cpp

```cpp
#include "lib.h"

int add_1(int a) {
    return a + 1;
}
```

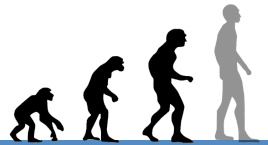
lib.cpp

```cpp
#include "lib.h"

int add_1(int a) {
    return a + 10;
}
```

Compile only the shared library:

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
~~g++ -o executable bin/main.cpp -lfoo~~
```

```
./executable
add_1: 10
add_2: 2
```



Challenger #3 - The sources 👩🏻

lib.h

```
int add_1(int a = 0);

int add_2(int a = 0) {
    return a + 2;
}
```

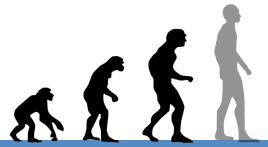
lib.h

```
int add_1(int a = 100);

int add_200(int a = 0) {
    return a + 200;
}
```

Compile only the shared library:

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
~~g++ -o executable bin/main.cpp -lfoo~~

./executable
add_1: 10
add_2: 2
```

😒



Challenger #3 - The sources 👩🏻‍💻

lib.h

```
int add_1(int a = 0);

int add_2(int a = 0) {
    return a + 2;
}
```

lib.h

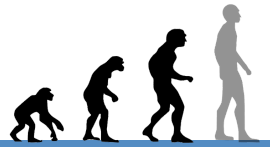
```
int add_100(int a = 0);

int add_2(int a = 0) {
    return a + 2;
}
```

Compile only the shared library:

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
~~g++ -o executable bin/main.cpp -lfoo~~
```

☺️

```
./executable
./executable: symbol lookup error: ./executable: undefined symbol: _Z5add_1i
```



Challenger #3 - The sources

lib.h

```
struct Sum {
    int a;
    int b;
};

int sum(const Sum&);

int sum_inline(const Sum& s) {
    return s.a + s.b;
}
```

lib.cpp

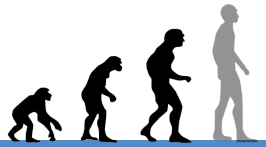
```
int sum(const Sum& s) {
    return s.a + s.b;
}
```

main.cpp

```
int main(int argc, const char* argv[]) {
    Sum s{10, 10};
    std::cout << "sum: " << sum(s) << "\n";
    std::cout << "sum_inline: " << sum_inline(s) << "\n";
```

Compile and link shared:

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
g++ -o executable bin/main.cpp -lfoo

./executable
sum: 20
sum_inline: 20
```



Challenger #3 - The sources 

lib.h

```
struct Sum {
    int a;
    int b;
};
```

```
int sum_inline(const Sum& s) {
    return s.a + s.b;
}
```

lib.cpp

```
int sum(const Sum& s) {
    return s.a + s.b;
}
```

lib.h

```
struct Sum {
    int a;
    int b;
    int c;
};
```

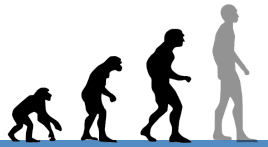
```
int sum_inline(const Sum& s) {
    return s.a + s.b + s.c;
}
```

lib.cpp

```
int sum(const Sum& s) {
    return s.a + s.b + s.c;
}
```

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
~~g++ -o executable bin/main.cpp -lfoo~~

./executable
sum: -1743113196
sum_inline: 20
```

😟



Challenger #3 - The sources 

lib.h

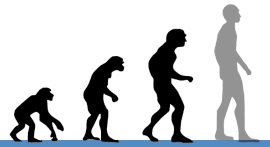
```
class Person {
public:
    Person(const std::string&);
    ~Person();

    std::string name;
};
```

main.cpp

```
void greet(const std::string& name) {
    std::cout << "Hello " << name << "!\n";
}

int main(int argc, const char* argv[]) {
    Person s{"Manuel"};
    if (argc == 1) { greet(s.name); }
    else { greet(argv[1]); }
}
```

Compile and link shared:

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
g++ -o executable bin/main.cpp -lfoo

./executable
Hello Manuel!
./executable Javier
Hello Javier!
```



Challenger #3 - The sources 

lib.h

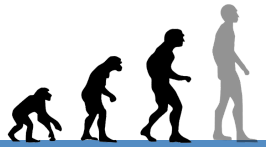
```
class Person {
public:
    Person(const std::string&);
    ~Person();

    std::string name;
};
```

lib.h

```
class Person {
public:
    Person(const std::string&);
    ~Person();
};
```

Compile and link shared:

```
g++ -c -fpic lib/lib.cpp
g++ -shared -o libfoo.so lib.o
g++ -o executable bin/main.cpp -lfoo

./executable
Segmentation fault
./executable Javier
Hello Javier!
```

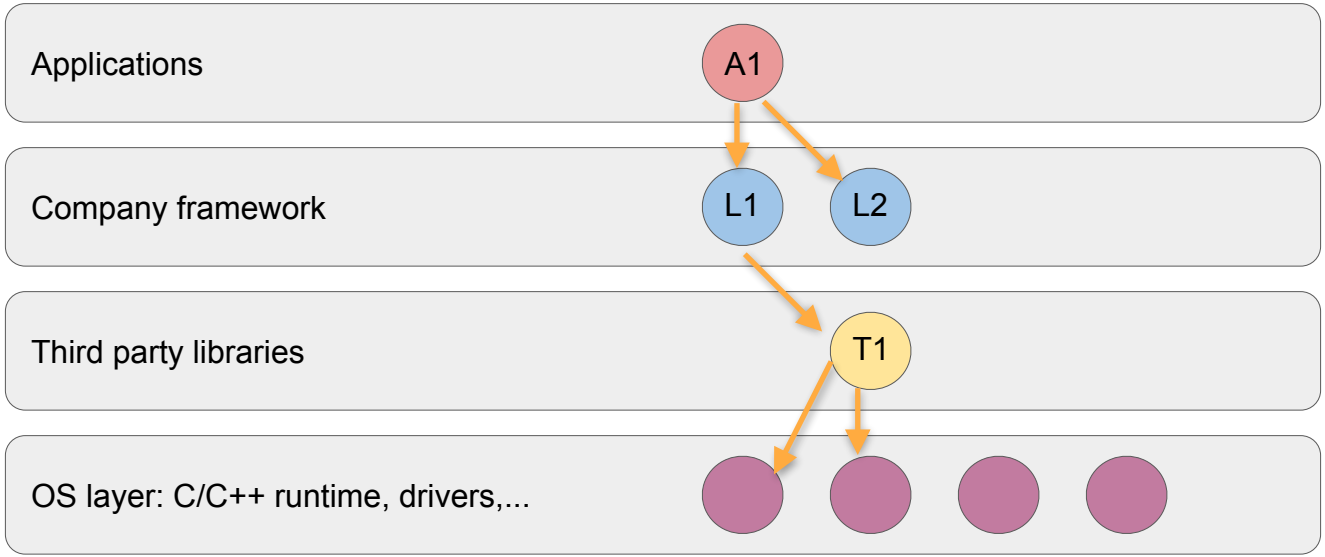


Challenger #4

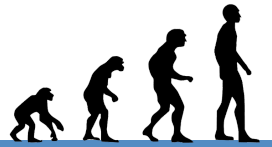
The environment

Challenger #4 - The environment 

When shared libraries enter the game...

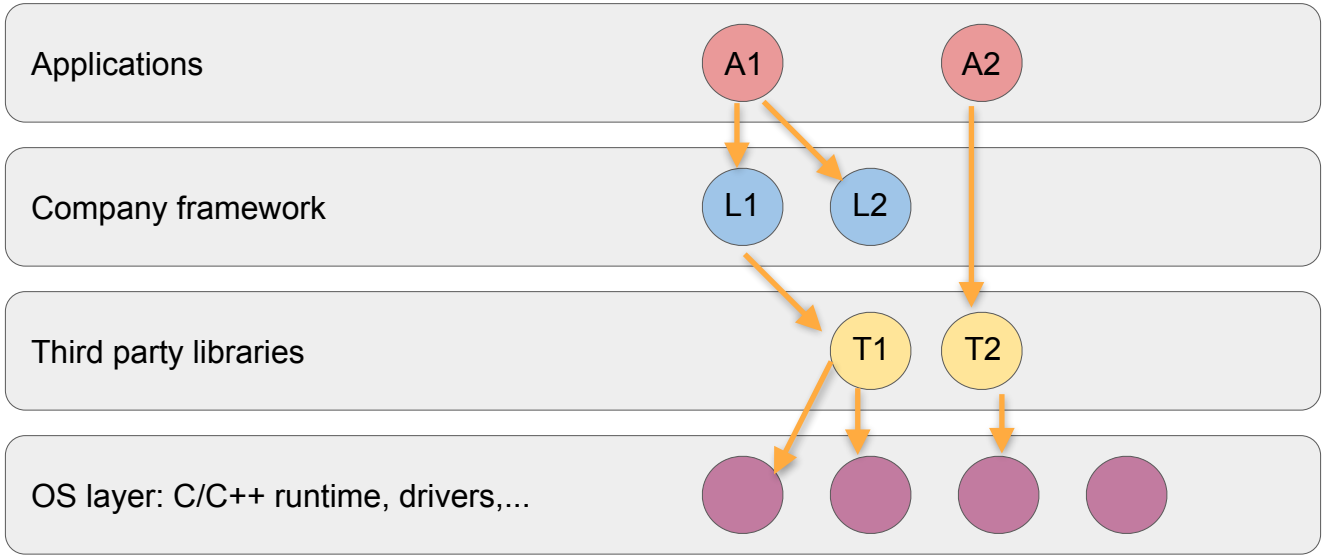


- `n` deploy environments

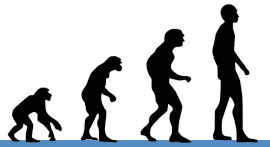


Challenger #4 - The environment

When shared libraries enter the game...

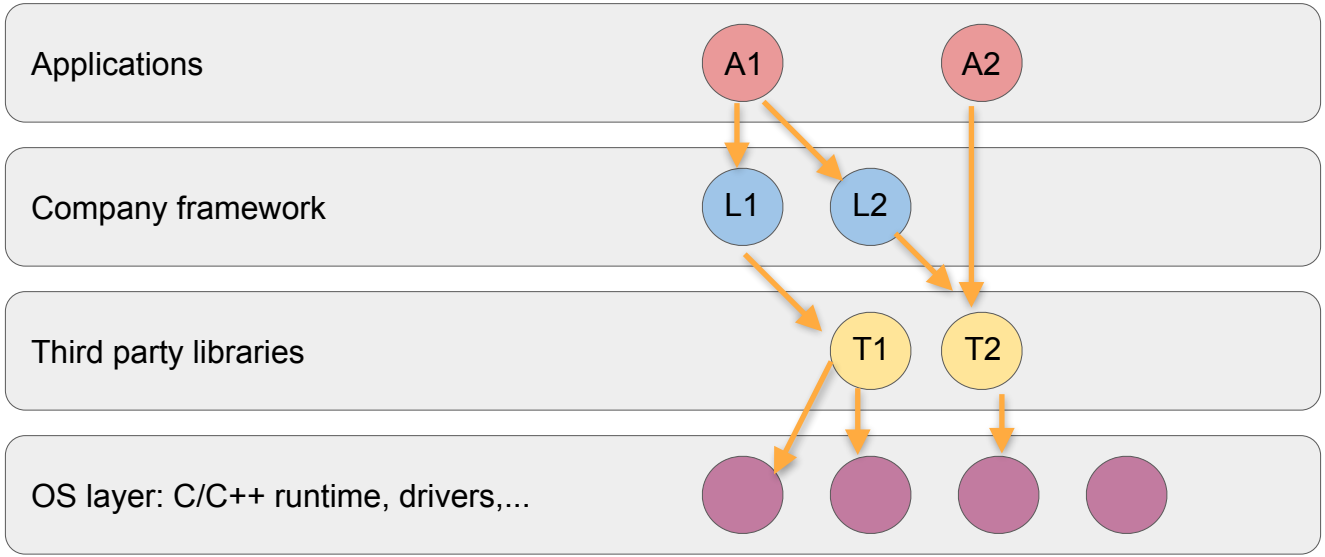


- `n` deploy environments

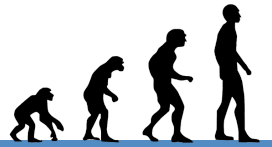


Challenger #4 - The environment 

When shared libraries enter the game...

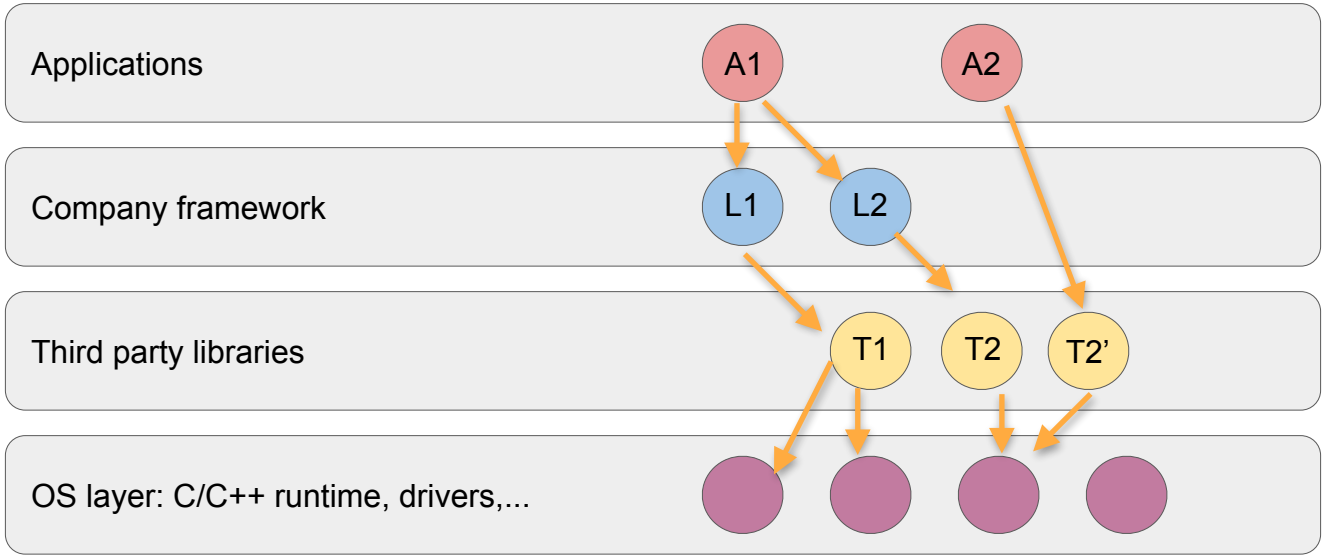


- `n` deploy environments

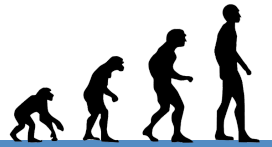


Challenger #4 - The environment

When shared libraries enter the game...

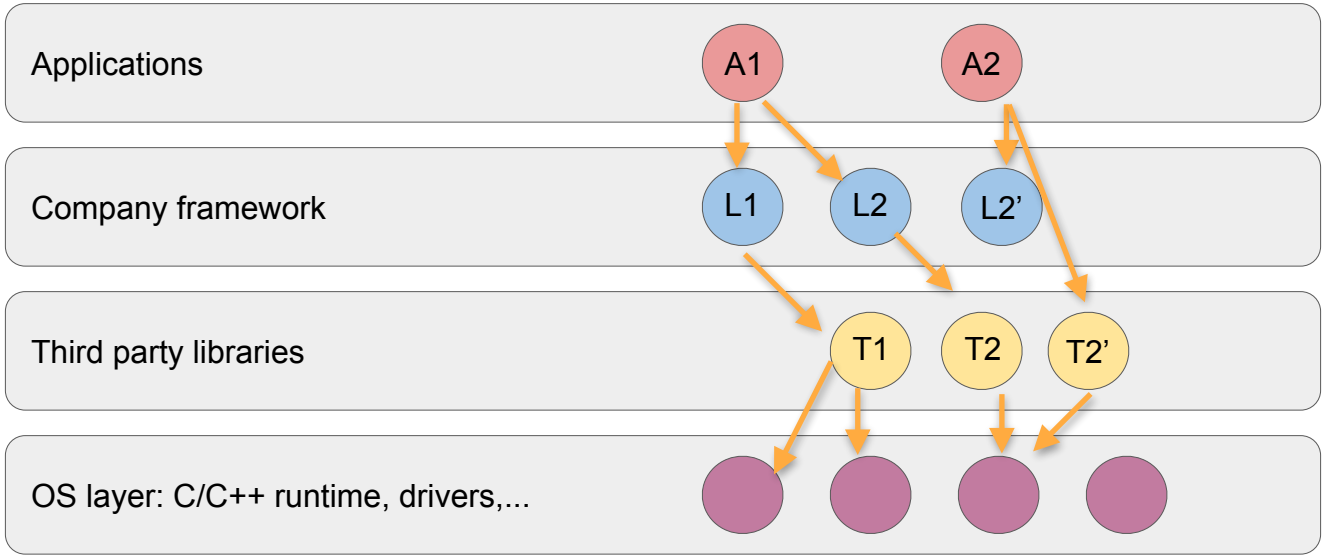


- `n` deploy environments

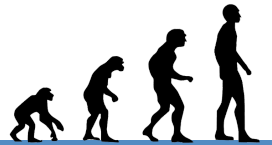


Challenger #4 - The environment

When shared libraries enter the game...



- `n` deploy environments

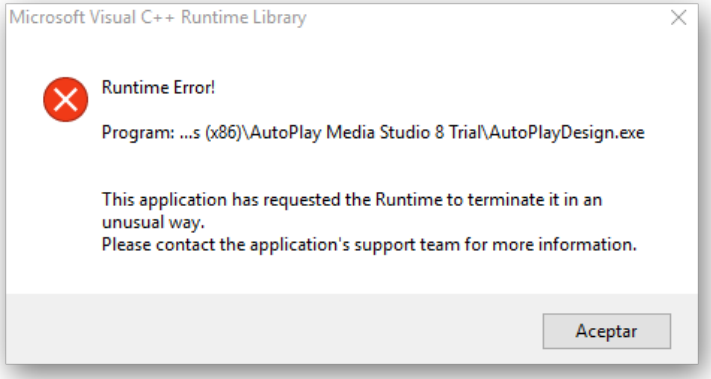


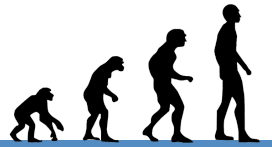
Challenger #4 - The environment 

Runtime error!

```
./executable: symbol lookup error: ./executable: undefined symbol:
_Z9get_helloRSt6vectorINSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEESaIS5_EE
```

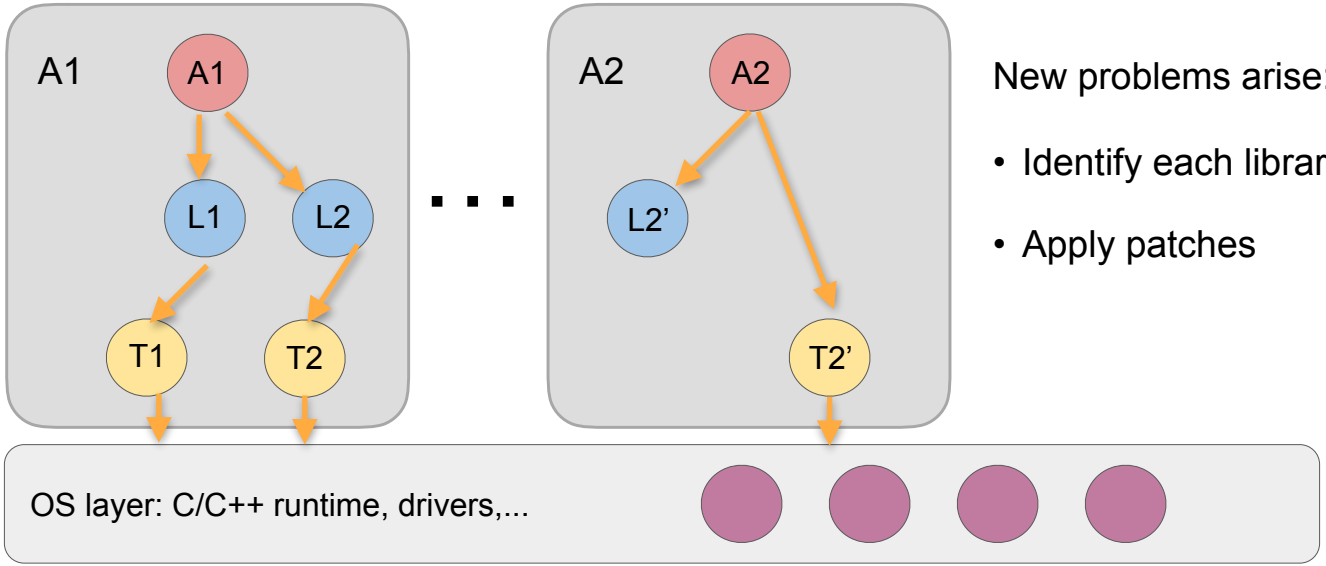
```
./executable: error while loading shared libraries: libfoo.so: cannot open shared object file: No such
file or directory
```





Challenger #4 - The environment

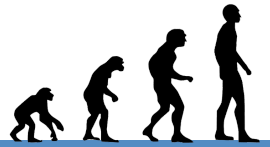
Final layout: each application with all its dependencies

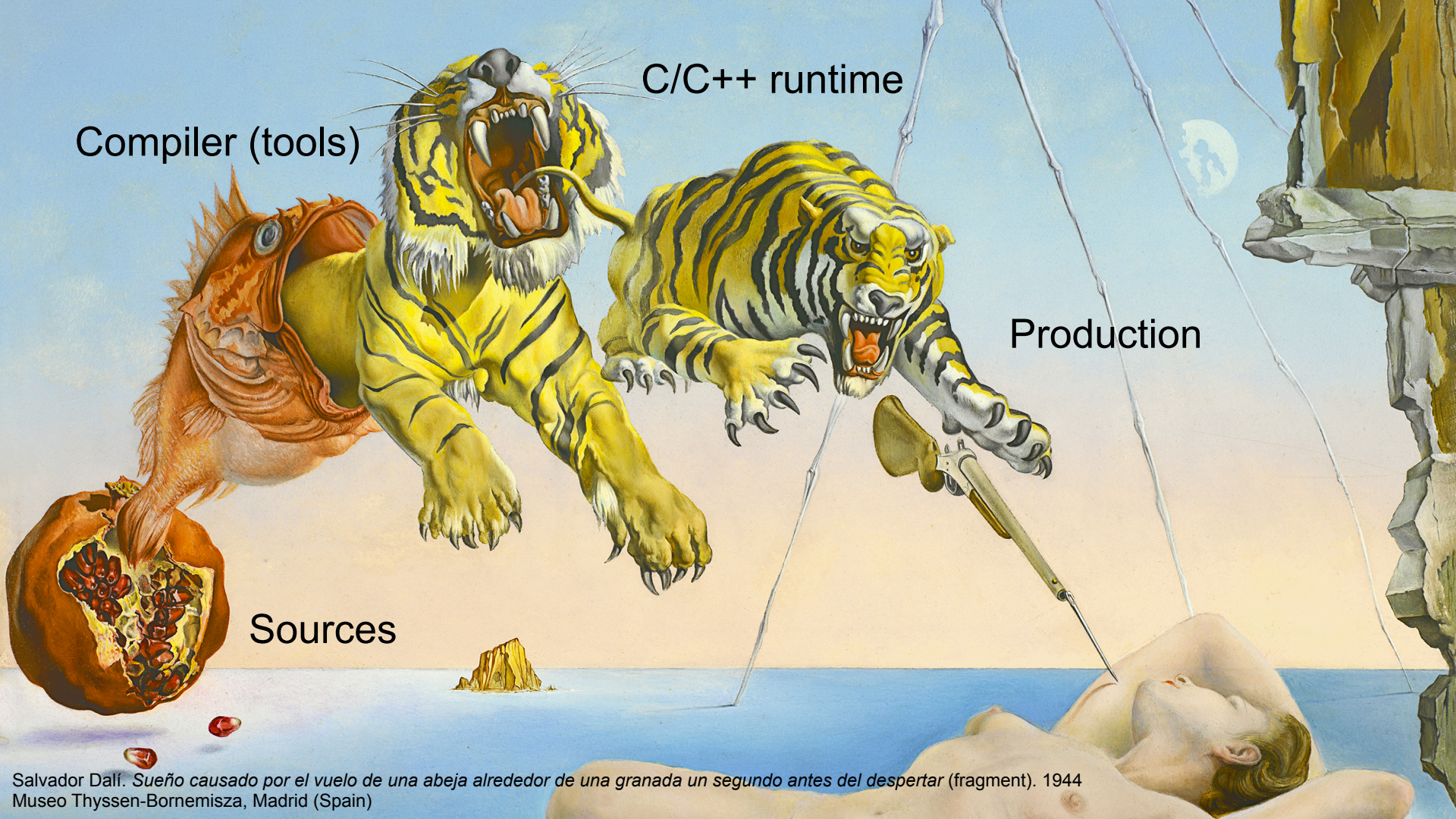


New problems arise:

- Identify each library
- Apply patches

· `n` deploy environments





Salvador Dalí. *Sueño causado por el vuelo de una abeja alrededor de una granada un segundo antes del despertar* (fragment). 1944
Museo Thyssen-Bornemisza, Madrid (Spain)

ABI compatibility

is ~~not~~ a ~~**MAJOR**~~ problem

MANAGEABLE

TRACTABLE

javierg@jfrog.com | @jgsogo | JFrog Ltd.

ABI compatibility, handle the problem

- Define everything that affects your ABI:
  - Sources:
    - name
    - version: semver
    - user: fork
    - channel: branch

```
name/version@user/channel
```

(aka. Conan reference)

ABI compatibility, handle the problem

- Define everything that affects your ABI:
  - Sources: name/version@user/channel
  - Operating system (distro, version,…), architecture,…

```
os:
    Windows:
        subsystem: [None, cygwin, msys, msys2, wsl]
    WindowsStore:
        version: ["8.1", "10.0"]
    Linux:
    Macos:
        version: [None, "10.6", "10.7", "10.8", "10.9", "10.10", "10.11", "10.12", "10.13", "10
    Android:
        api_level: ANY
    iOS:
        version: ["7.0", "7.1", "8.0", "8.1", "8.2", "8.3", "9.0", "9.1", "9.2", "9.3", "10.0",
    watchOS:
        version: ["4.0", "4.1", "4.2", "4.3", "5.0", "5.1"]
    tvOS:
```

ABI compatibility, handle the problem

- Define everything that affects your ABI:
  - Sources: name/version@user/channel
  - Operating system (distro, version,…), architecture,…
  - Tooling: compiler, linker options,…

```
compiler:
    gcc:
        version: ["4.1", "4.4", "4.5", "4.6", "4.7", "4.8", "4.9",
                  "5", "5.1", "5.2", "5.3", "5.4", "5.5",
                  "6", "6.1", "6.2", "6.3", "6.4",
                  "7", "7.1", "7.2", "7.3",
                  "8", "8.1", "8.2"]
        libcxx: [libstdc++, libstdc++11]
    Visual Studio:
        runtime: [MD, MT, MTd, MDd]
        version: ["8", "9", "10", "11", "12", "14", "15", "16"]
        toolset: [None, v90, v100, v110, v110_xp, v120, v120_xp,
                  v140, v140_xp, v140_clang_c2, LLVM-vs2012, LLVM-vs2012_xp,
                  LLVM-vs2013, LLVM-vs2013_xp, LLVM-vs2014, LLVM-vs2014_xp,
```

ABI compatibility, handle the problem

- Define everything that affects your ABI:
  - Sources: name/version@user/channel
  - Operating system (distro, version,…), architecture,…
  - Tooling: compiler, library options,…
  - Other settings or flags

```
build_type: [None, Debug, Release, RelWithDebInfo, MinSizeRel]
cppstd: [None, 98, gnu98, 11, gnu11, 14, gnu14, 17, gnu17, 20, gnu20]
```

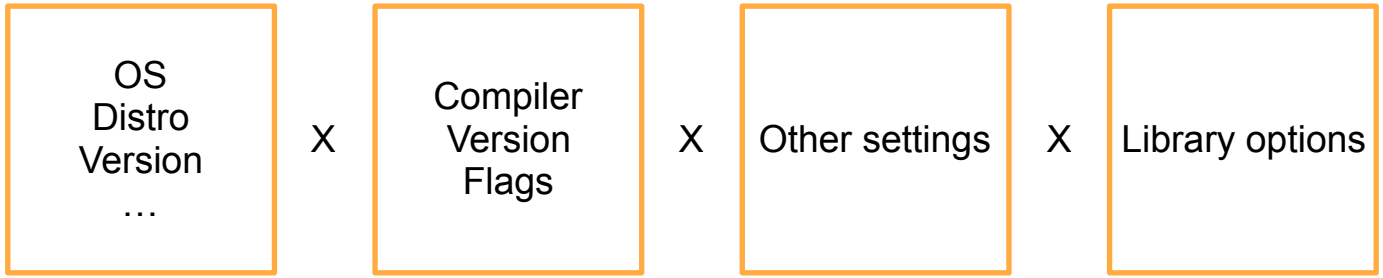
ABI compatibility, handle the problem

- Define everything that affects your ABI:
  - Sources: name/version@user/channel
  - Operating system (distro, version,…), architecture,…
  - Tooling: compiler, library options,…
  - Other settings or flags
  - … specific options

```
options = {"shared": [True, False], "fPIC": [True, False],      "contrib": [True, False],
           "jpeg": [True, False],   "tiff": [True, False],      "webp": [True, False],
           "png": [True, False],    "jasper": [True, False],    "gtk": [None, 2, 3]}
```

ABI compatibility, handle the problem

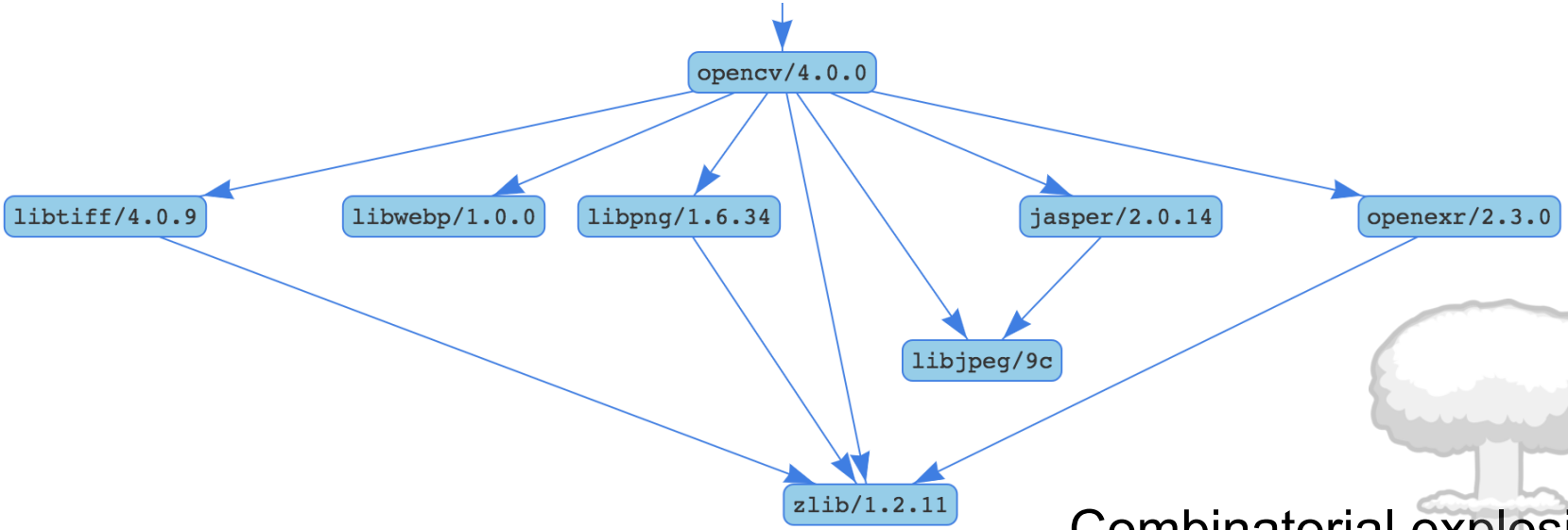
- Define everything that affects your ABI
- Assign a different ID to each combination



Combinatorial explosion!

ABI compatibility, handle the problem

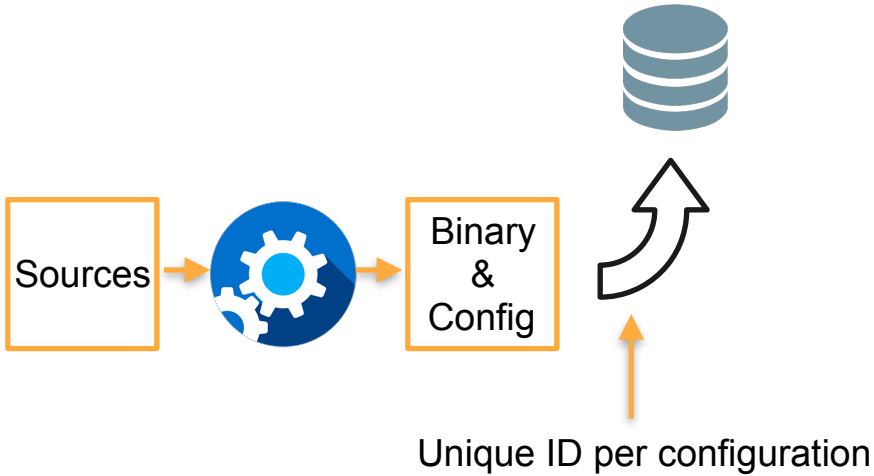
- Define everything that affects your ABI
- Assign a different ID to each combination



Combinatorial explosion!

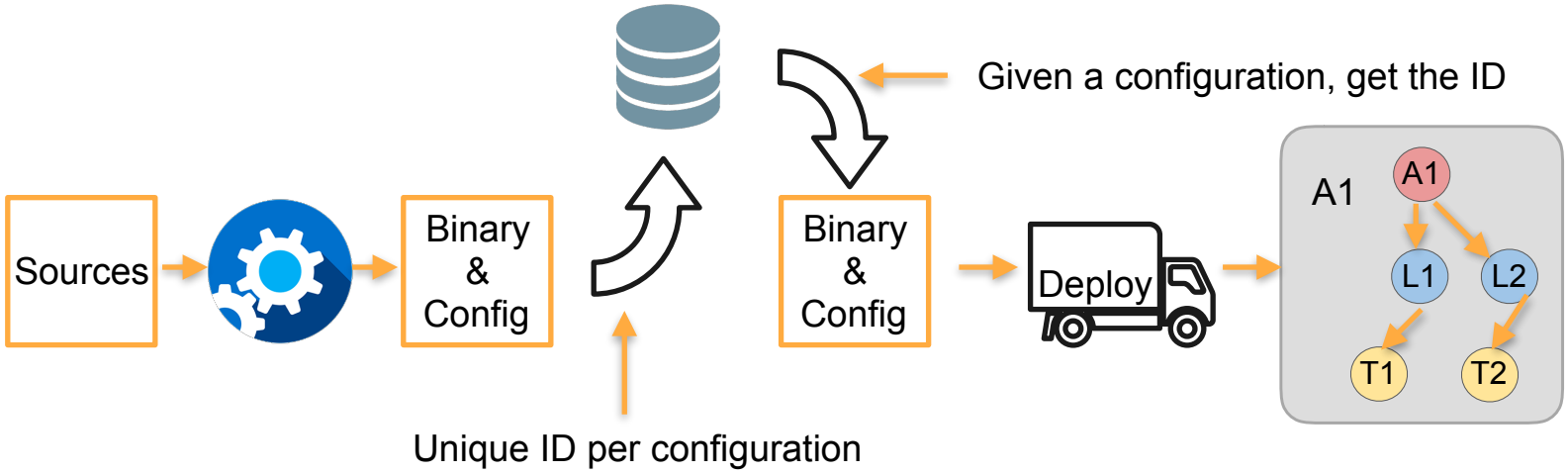
ABI compatibility, handle the problem

- Define everything that affects your ABI
- Assign a different ID to each combination



ABI compatibility, handle the problem

- Define everything that affects your ABI
- Assign a different ID to each combination



Diego Velázquez. *La fragua de Vulcano* (fragment). 1630
Museo Nacional del Prado, Madrid (Spain)

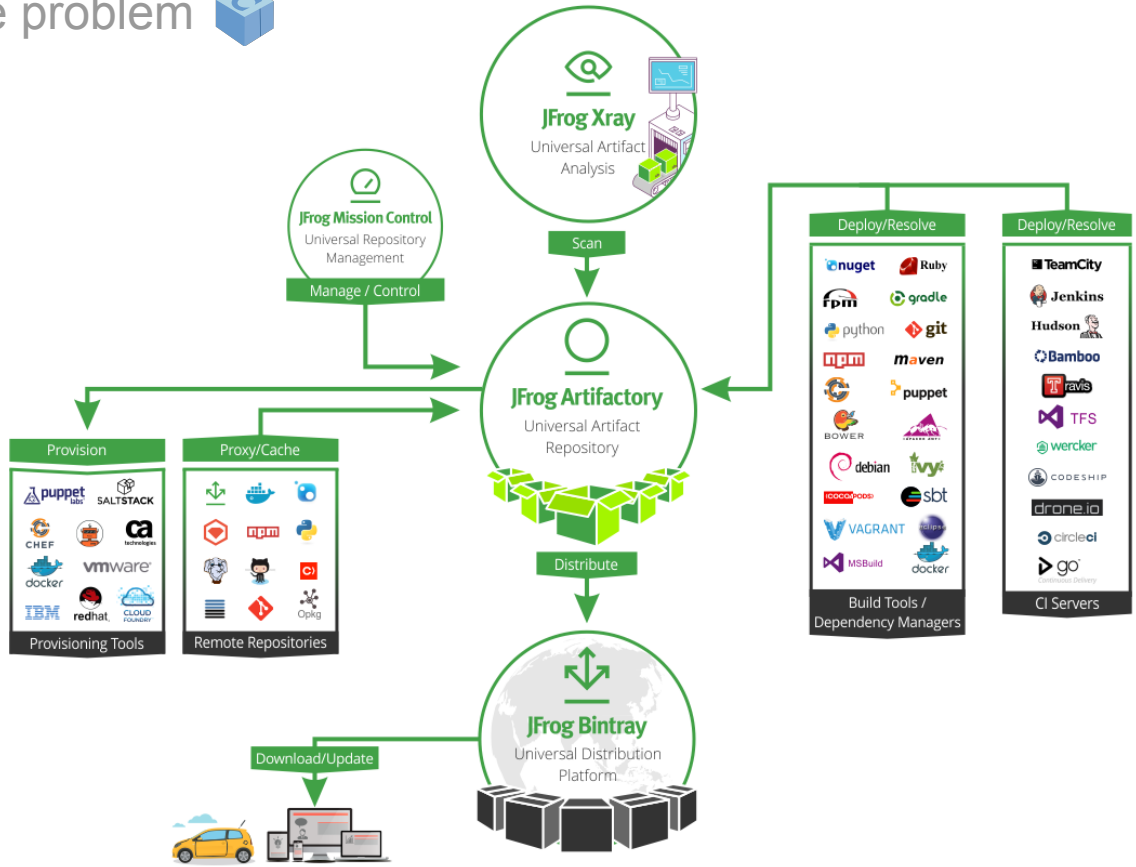
ABI compatibility, handle the problem

Artifactory:

> Binary database

Conan

~ SQL



ABI compatibility, handle the problem

CONAN: reference + settings/options => ID

```
conan info .  [default configuration]
ID: f8bda7f0751e4bc3beaa6c3b2eb02d455291c8a2

conan info . -s build_type=Debug
ID: f51350faf9be5480f35c35c17122e2cb6bfb617f

conan info . -s build_type=Debug -s os=Windows
ID: cf18caf7722d01e5ebf09c08a264d2fa7f5e1a58

conan info . -s os=Windows
ID: 98daf1be085c523736063391741a5fde90344a04

conan info . -s os=Windows -o api=v2
ID: ffa526e575e6def1cd97cbda3f211d5f5d650870
```

```python
class Library(ConanFile):
    name = "library"
    version = "1.0.0"

    settings = "os", "arch", "compiler", "build_type"
    options = {'shared': [True, False],
               'api': ['v1', 'v2', 'v3',] }
    default_options = {'shared': True, 'api': 'v1'}

    def source(self):
        # Instructions to get the sources
        pass

    def build(self):
        # Instructions to build
        pass

    def package(self):
        # Collect files that belong to the package
        pass
```

ABI compatibility, handle the problem

CONAN: reference + ID => settings/options + recipe

```
conan search zlib/1.2.8@conan/stable  [query expression]
Existing packages for recipe zlib/1.2.8@conan/stable:

    Package_ID: 1513b3452ef7e2a2dd5f931247c5e02edeb98cc9
        [options]
            fPIC: True
            shared: False
        [settings]
            arch: x86_64
            build_type: Debug
            compiler: apple-clang
            compiler.version: 10.0
            os: Macos
        Outdated from recipe: False

    Package_ID: 534dcc368c999e07e81f146b3466b8f656ef1f55
        [options]
            fPIC: True
            shared: False
```

More about **CONAN?**

- **Core C++ (Tel-Aviv)** May 14-17
- **Italian C++ (Milan)** Jun 15
- **SwampUp (S. Francisco)** Jun 17-19
- **CppCon (Colorado)** Sept 15-20
- **Meeting C++ (Berlin)** Nov 14.16
- … discussion zone (C++ Russia)

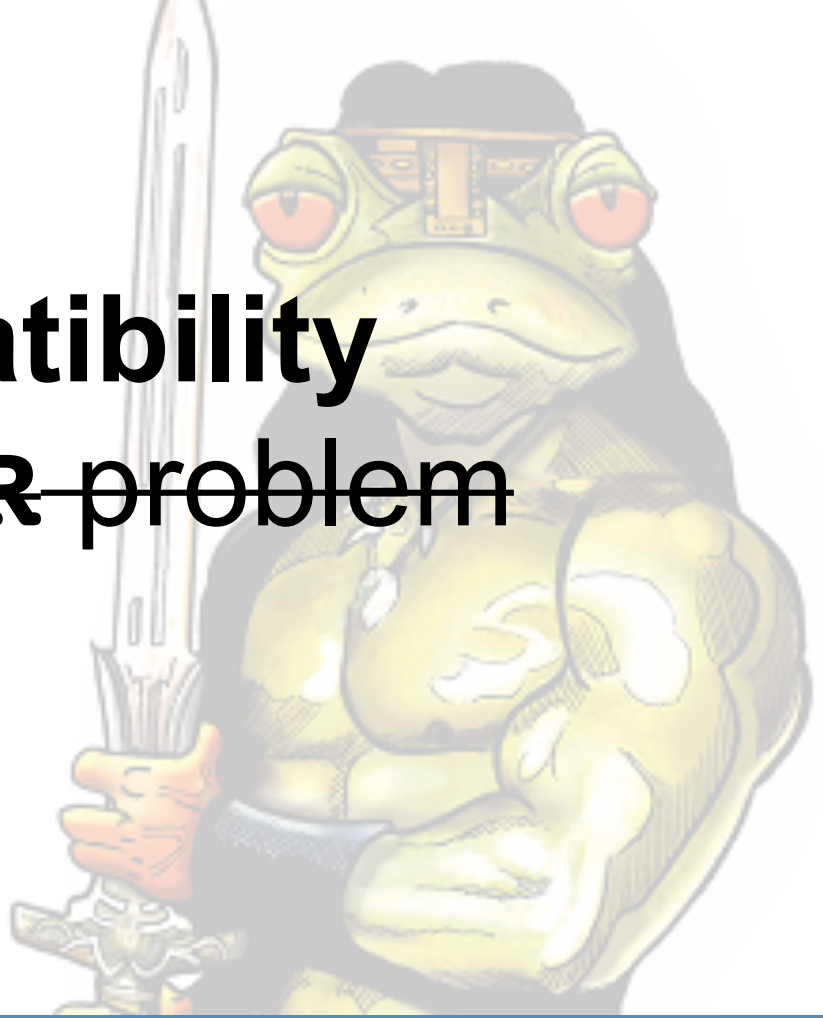
And everywhere thanks to our amazing community, you are awesome!

https://conan.io

El Greco. *Pentecostés* (fragment). 1597-1600
Museo Nacional del Prado, Madrid (Spain)

ABI compatibility

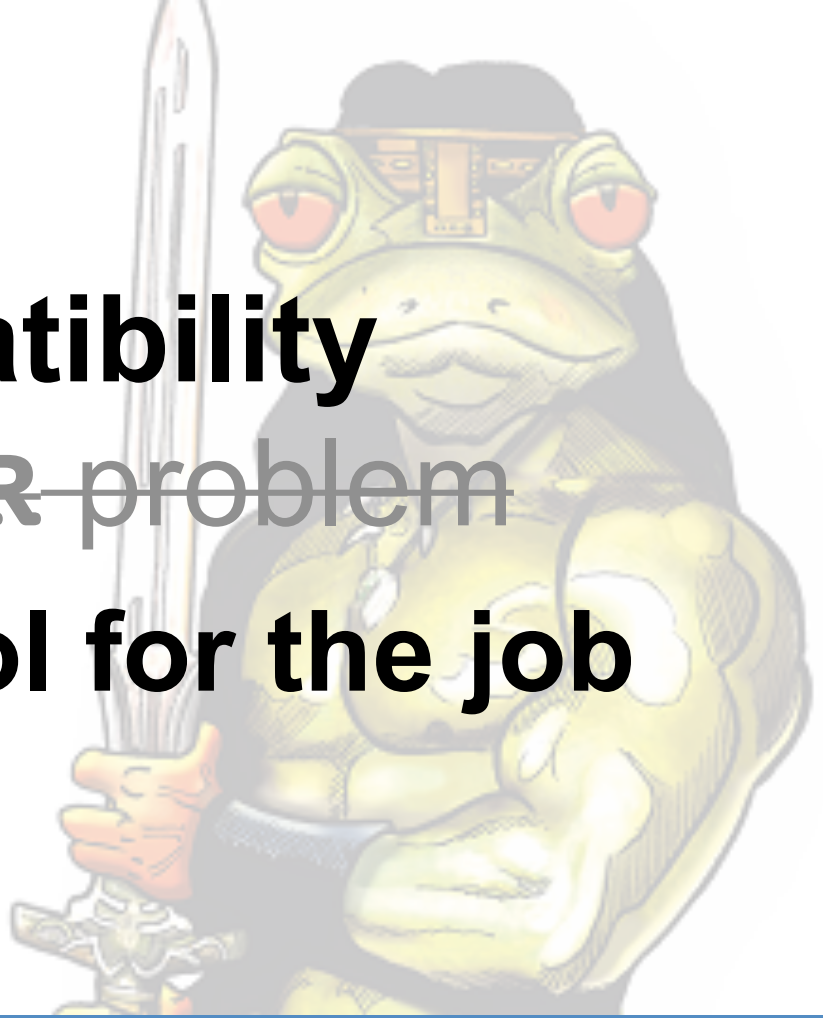
~~is not a **MAJOR** problem~~



ABI compatibility

~~is not a **MAJOR** problem~~

Use the right tool for the job



References

- GCC 5 in Fedora (What's an ABI, and what happens when we change it?)
  Link: https://fedoramagazine.org/gcc-5-in-fedora-whats-an-abi-and-what-happens-when-we-change-it/
- Mathieu Ropert. "ABI & API versioning" CppCon 2017
  Video: https://www.youtube.com/watch?v=Ia3IDPjA-d0
- Thiago Macieira. "Binary compatibility for library developers" C++Now 2013
  Video: https://www.youtube.com/watch?v=PHrXGHDd9no
- Conan documentation
  Link: http://docs.conan.io/
- C++ Exception handling internals (An infinite monkey, Nico Brailovsky's blog)
  Link: https://monoinfinito.wordpress.com/series/exception-handling-in-c/
- KDE: Policies/Binary compatibility issues with C++
  Link: https://community.kde.org/Policies/Binary_Compatibility_Issues_With_C%2B%2B
- Oracle: Stability of the C++ ABI: Evolution of a Programming Language
  Link: https://www.oracle.com/technetwork/systems/stablecplusplusabi-333927.html
- Libabigail manual
  Link: https://sourceware.org/libabigail/manual/index.html
- Examining binary files in Linux
  Link: http://blog.vinceliu.com/2009/06/examining-binary-files-in-linux.html
- What's an ABI and why is it so complicated?
  Link: https://accu.org/content/conf2015/JonathanWakely-What Is An ABI And Why Is It So Complicated.pdf

Thanks!

conan.io
@conan_io

Javier G. Sogo | javierg@jfrog.com | @jgsogo | JFrog Ltd.