What's New in Conan 2.0

Lessons Learned from the C++ Ecosystem

DIEGO RODRIGUEZ-LOSADA

Everything is new!

1.0

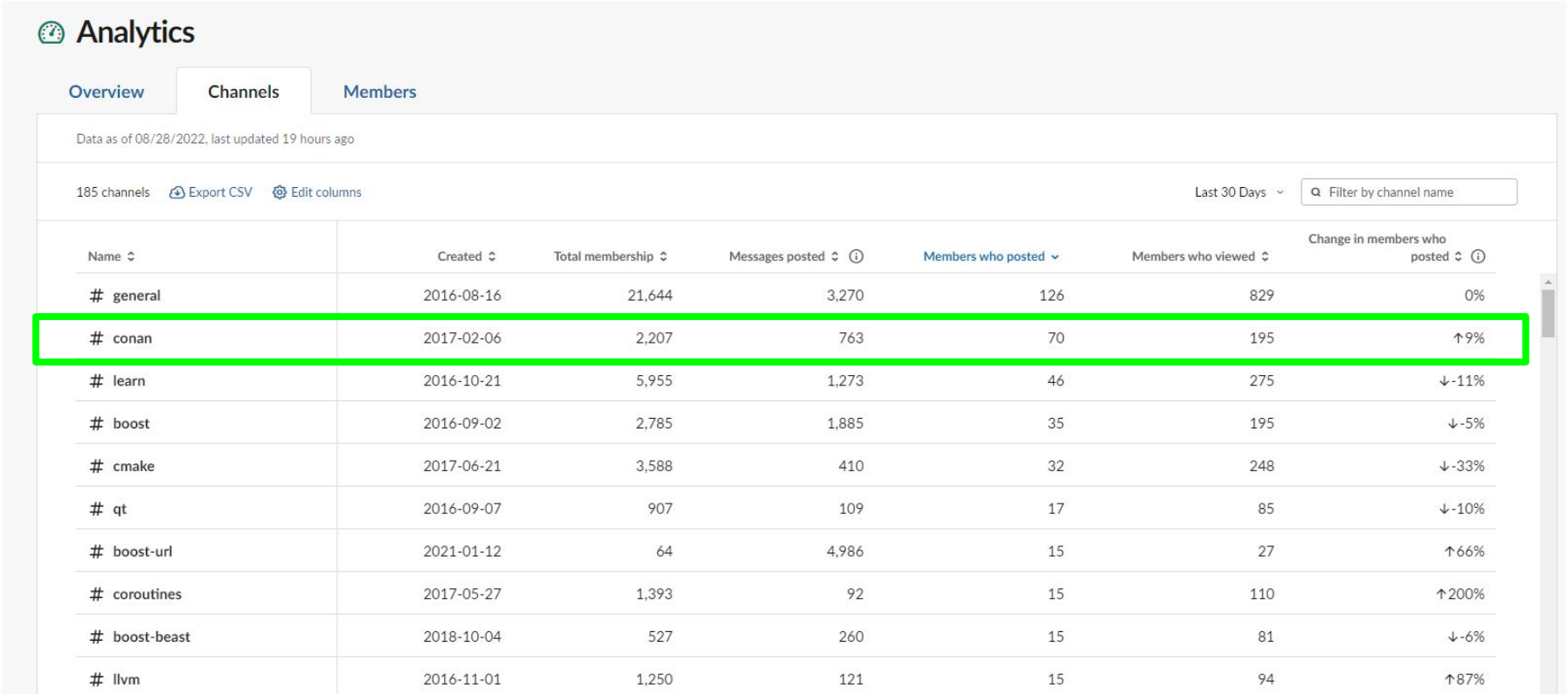
5 years, without breaking

60% new code, 20% backports

1.X ⇔ 2.0 compatible syntax subset

2.0

CppLang #conan slack



Analytics

Overview | Channels | Members

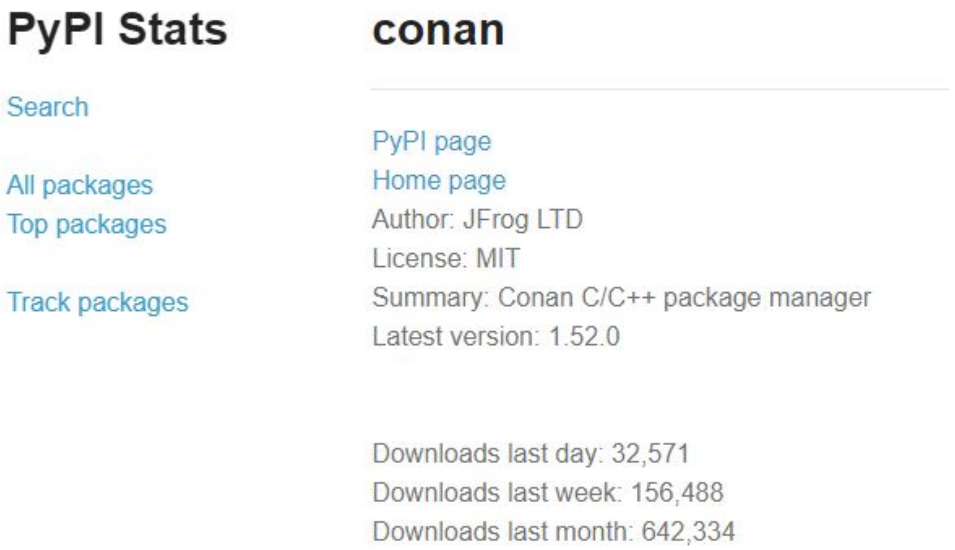
Data as of 08/28/2022, last updated 19 hours ago

185 channels | Export CSV | Edit columns | Last 30 Days | Filter by channel name

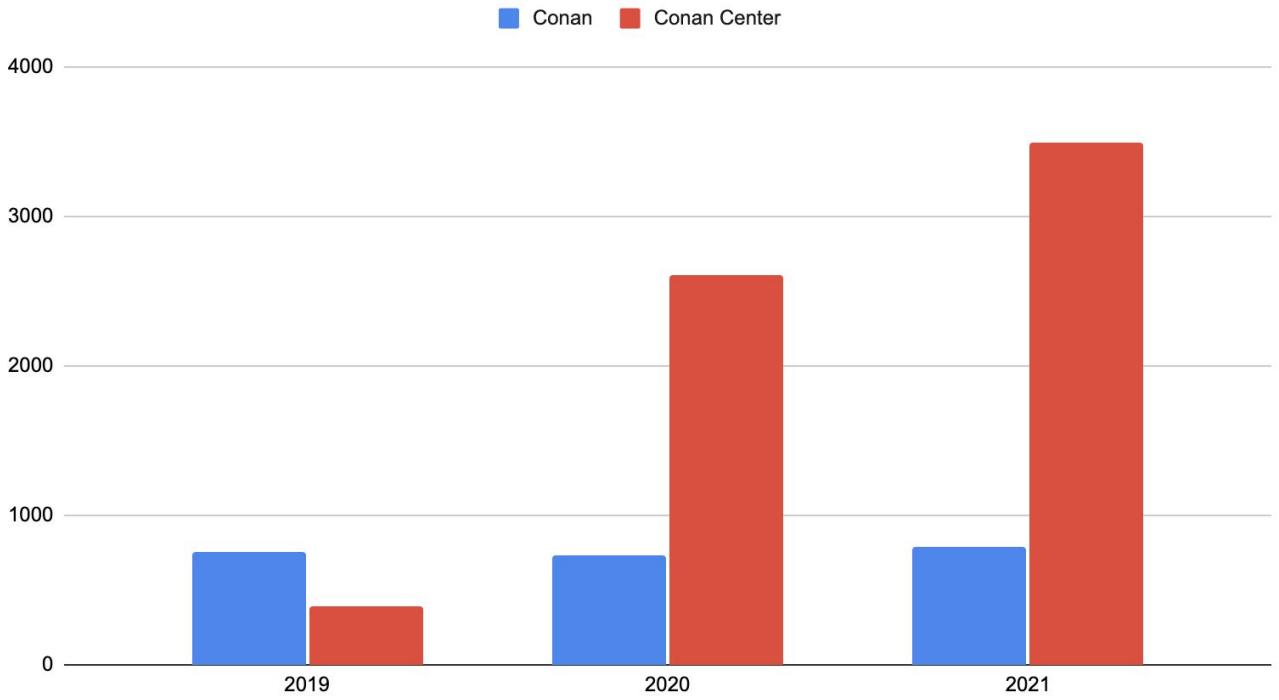
Name	Created	Total membership	Messages posted	Members who posted	Members who viewed	Change in members who posted
# general	2016-08-16	21,644	3,270	126	829	0%
# conan	2017-02-06	2,207	763	70	195	↑9%
# learn	2016-10-21	5,955	1,273	46	275	↓-11%
# boost	2016-09-02	2,785	1,885	35	195	↓-5%
# cmake	2017-06-21	3,588	410	32	248	↓-33%
# qt	2016-09-07	907	109	17	85	↓-10%
# boost-url	2021-01-12	64	4,986	15	27	↑66%
# coroutines	2017-05-27	1,393	92	15	110	↑200%
# boost-beast	2018-10-04	527	260	15	81	↓-6%
# llvm	2016-11-01	1,250	121	15	94	↑87%

PyPI downloads (Conan tool)

- 642K downloads/month from PyPI
- Designated as PyPI critical project (1% of most downloaded in whole PyPI)



Github PRs

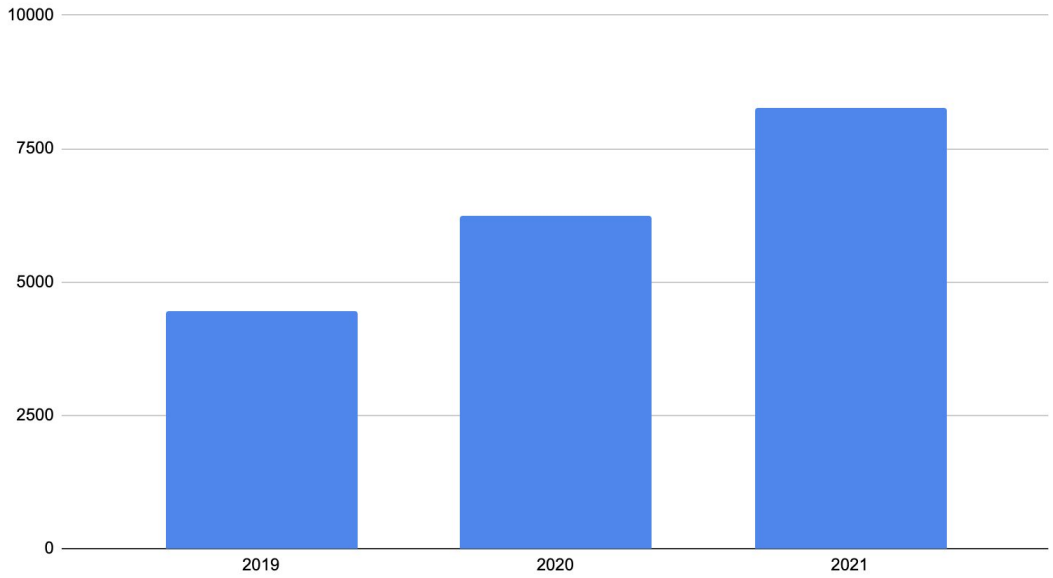


Support

+2000 Github issues / year

100 hr/year user videocalls

Direct support (slack, almost daily)



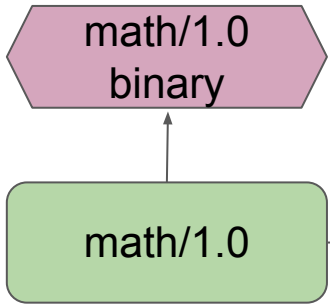
Artifactory servers running Conan in production and telemetry enabled (no firewalls)

Overview

- 5 lessons:
  - Learning to fly
  - Cats are a good default (or not?)
  - Building a dam
  - Repeating yourself
  - Dying of a thousand bites
- Conclusions

1. Learning to fly

Conanfile: A package “recipe”

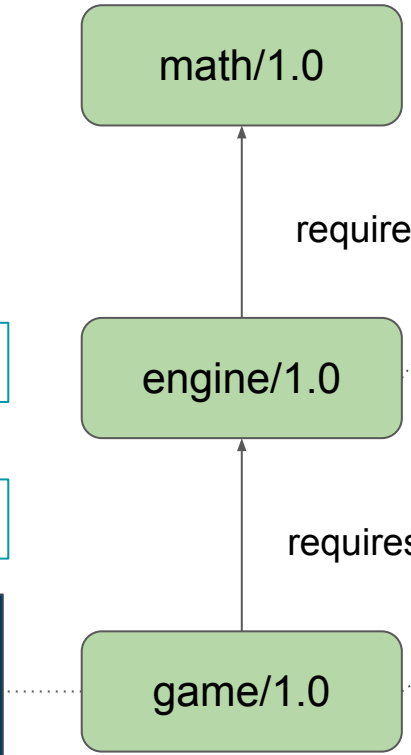



```
$ git clone … math && cd math
$ conan create .
```

math/conanfile.py

```python
from conan import ConanFile

class Math(ConanFile):
    name = "math"
    version = "1.0"

    def source(self): ...
    def build(self): ...
    def package(self): ...
```

Conan 1.X dependency model: Transitive deps



math-config.cmake

```
set_property(TARGET math::math ...)
```

engine-config.cmake

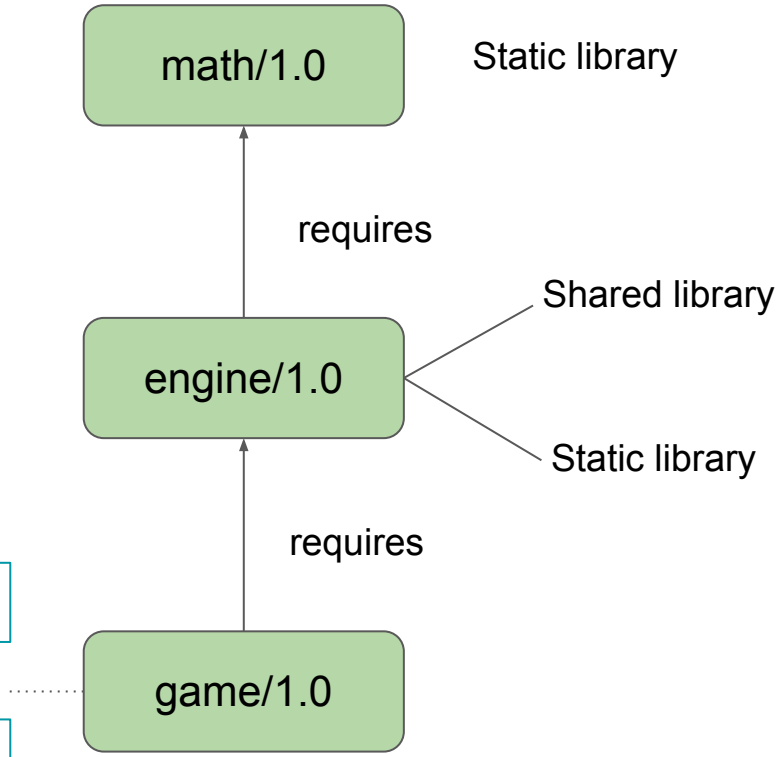
```
set_property(TARGET engine::engine ...)
```

```
$ git clone … game && cd game
$ conan install .
$ cmake …
```

```
from conan import ConanFile

class Engine(ConanFile):
    requires = "math/1.0"
```

```
from conan import ConanFile

class Game(ConanFile):
    requires = "engine/1.0"
    generators = "CMakeDeps"
```

Conan 1.X dependency model: Transitive deps



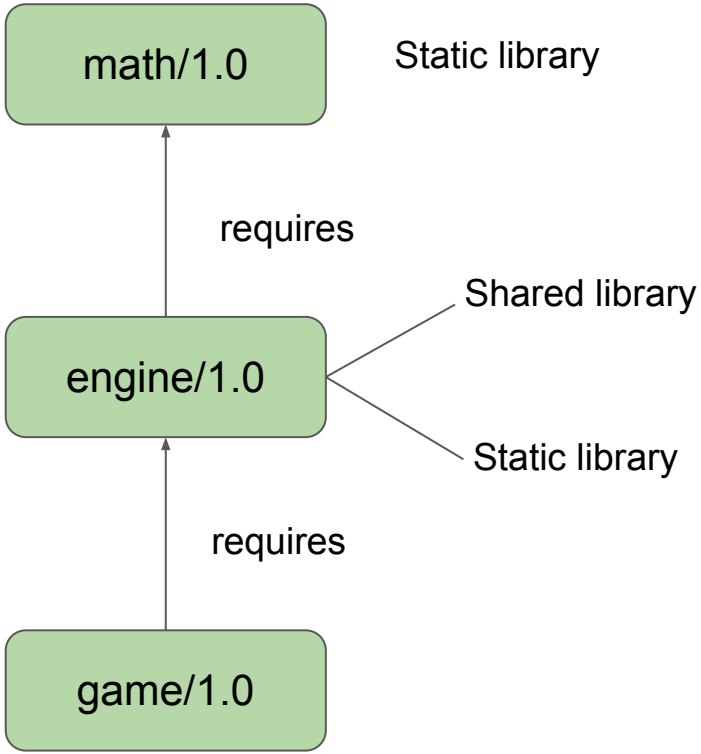
math-config.cmake

```
set_property(TARGET math::math ...)
```

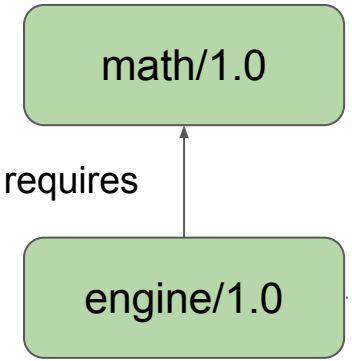
engine-config.cmake

```
set_property(TARGET engine::engine ...)
```

Learning to fly



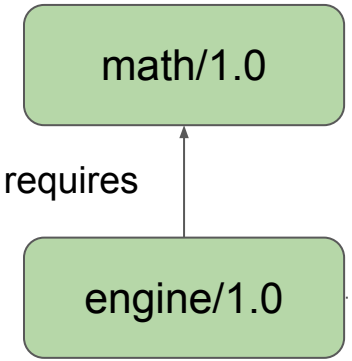
Conan 2.0 proposal



engine/conanfile.py

```
from conan import ConanFile

class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0")
```

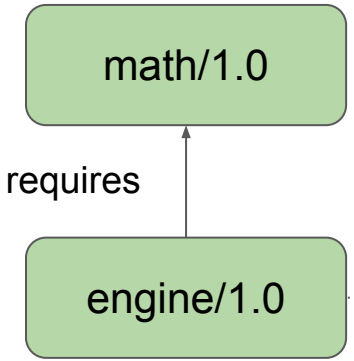
Conan 2.0 proposal: Requirement traits



engine/conanfile.py

```
from conan import ConanFile

class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                      headers=True, libs=True)
```

Conan 2.0 proposal: Requirement traits



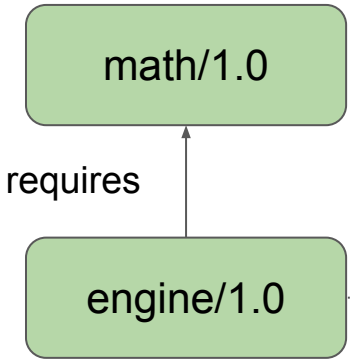
engine/conanfile.py

```
from conan import ConanFile

class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                      headers=True, libs=True)
```

math-config.cmake

```
set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)
set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES ...)
```

Conan 2.0 proposal: Requirement traits



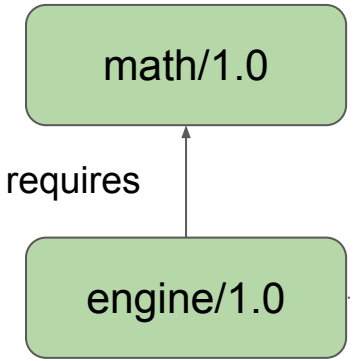
engine/conanfile.py

```python
from conan import ConanFile

class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                      headers=False, libs=True)
```

math-config.cmake

```
set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)
set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES ...)
```

Conan 2.0 proposal: Requirement traits



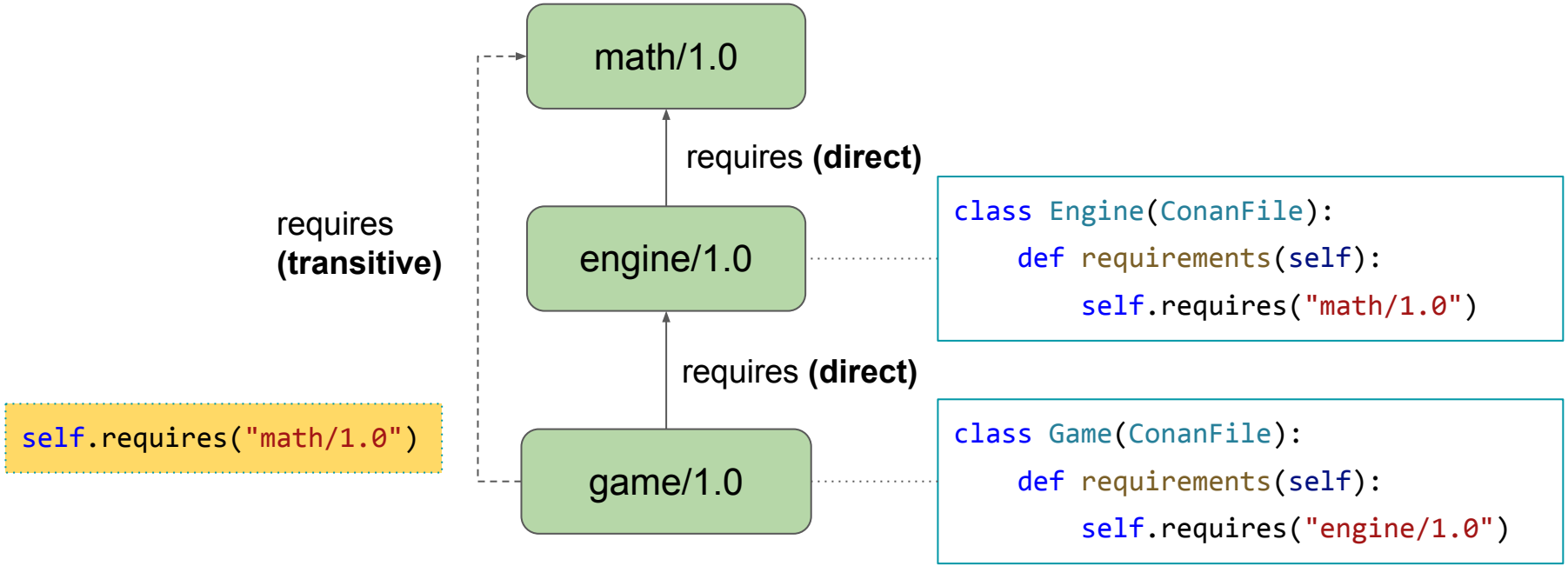
engine/conanfile.py

```python
from conan import ConanFile

class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                      headers=True, libs=False)
```

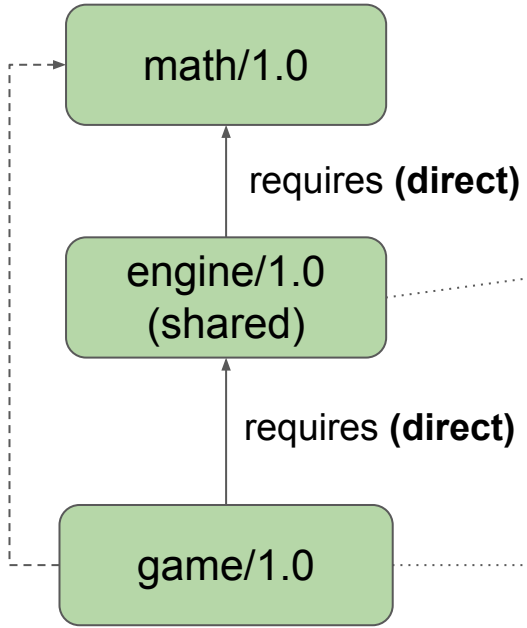
math-config.cmake

```
set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)
set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES ...)
```

Conan 2.0 proposal: Direct vs. transitive dependencies



Linkage requirements propagation



requires
(transitive)

```
class Engine(ConanFile):
    def requirements(self):
        self.requires("math/1.0",
            headers=True, libs=True,
            transitive_libs=False)
```

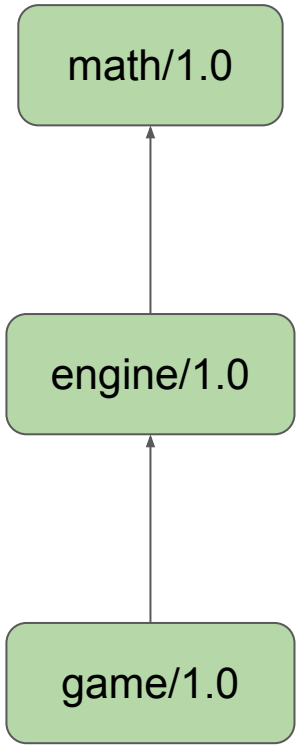
```
class Game(ConanFile):
    def requirements(self):
        self.requires("engine/1.0",
            headers=True, libs=True)
```

```
self.requires("math/1.0",
              headers=False,
              libs=False)
```

math-config.cmake

```
set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)
set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES ...)
```

Package Types



math/conanfile.py

```
class Math(ConanFile):
    name = "math"
    version = "1.0"
    package_type = "static-library"
    # OR options = {"shared": [True, False]}
```

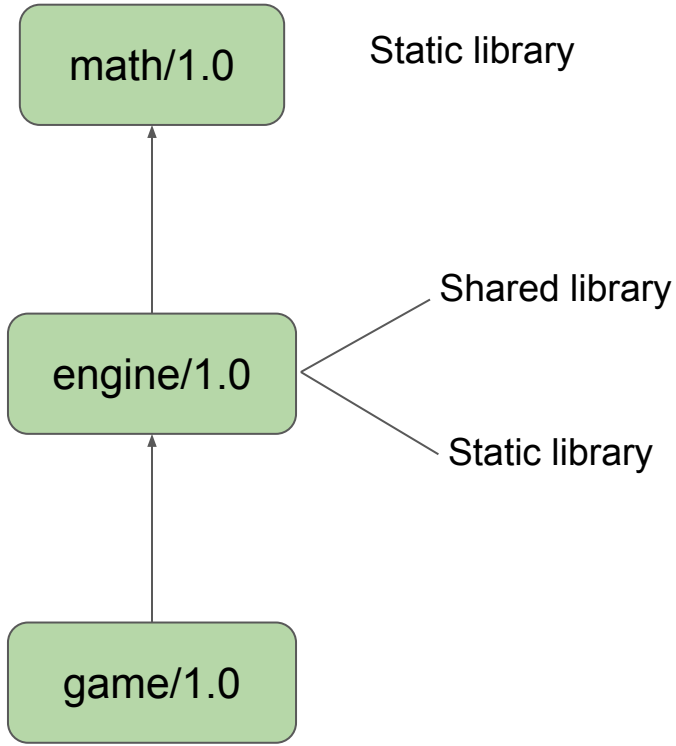
engine/conanfile.py

```
class Engine(ConanFile):
  package_type = "shared-library"
  # OR options = {"shared": [True, False]}
  def requirements(self):
    self.requires("math/1.0")
```

game/conanfile.py

```
class Game(ConanFile):
  package_type = "application"
  def requirements(self):
    self.requires("engine/1.0")
```

Demo


math/1.0
Static library
engine/1.0
Shared library
Static library
game/1.0

Dependency graph 2.0

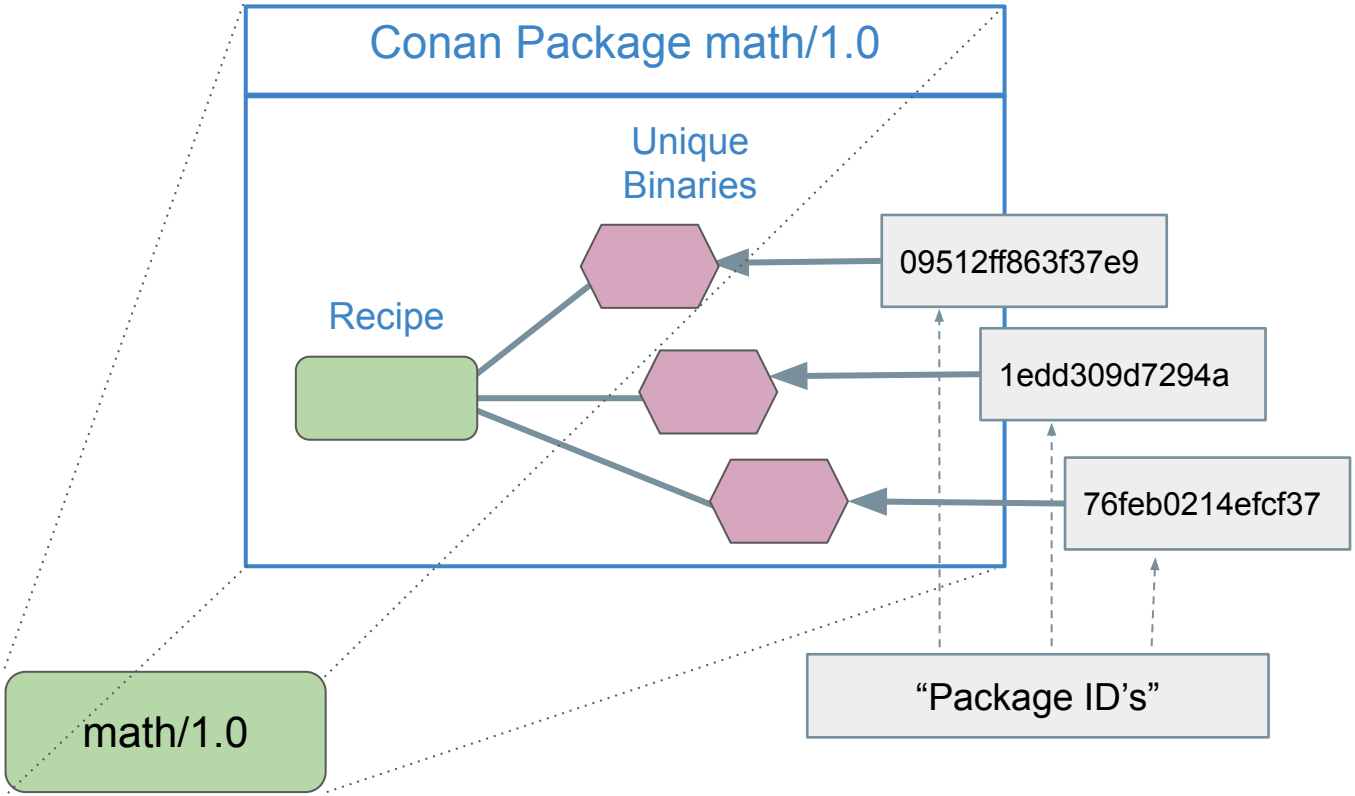
- Correct linkage requirements
- Correct header visibility
- Possible hidden/private dependencies
- and many more (ACCU 2022)

Among different build systems!

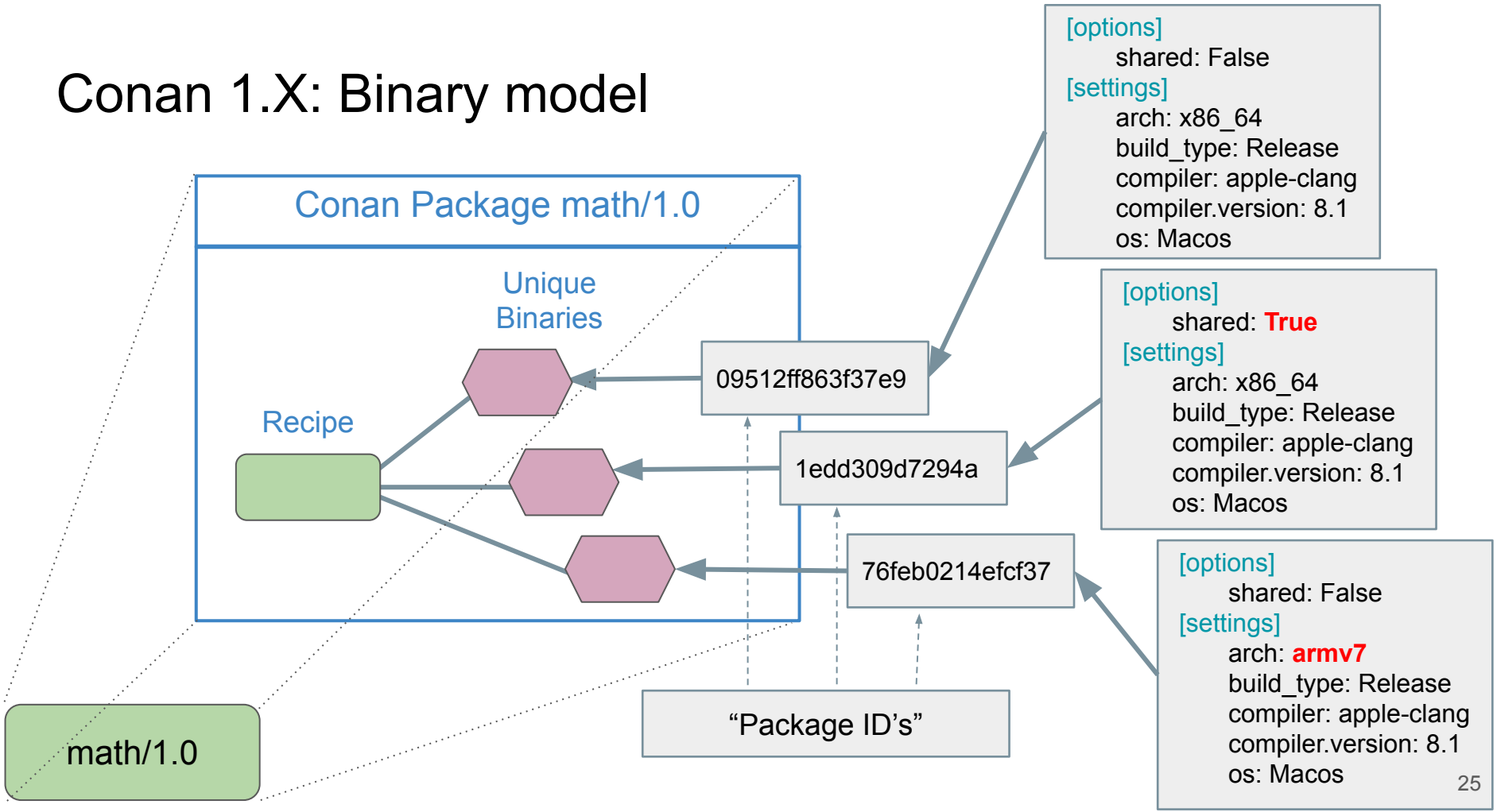
Compatible “requires” syntax with 1.X

2. Cats are a good default (or not?)

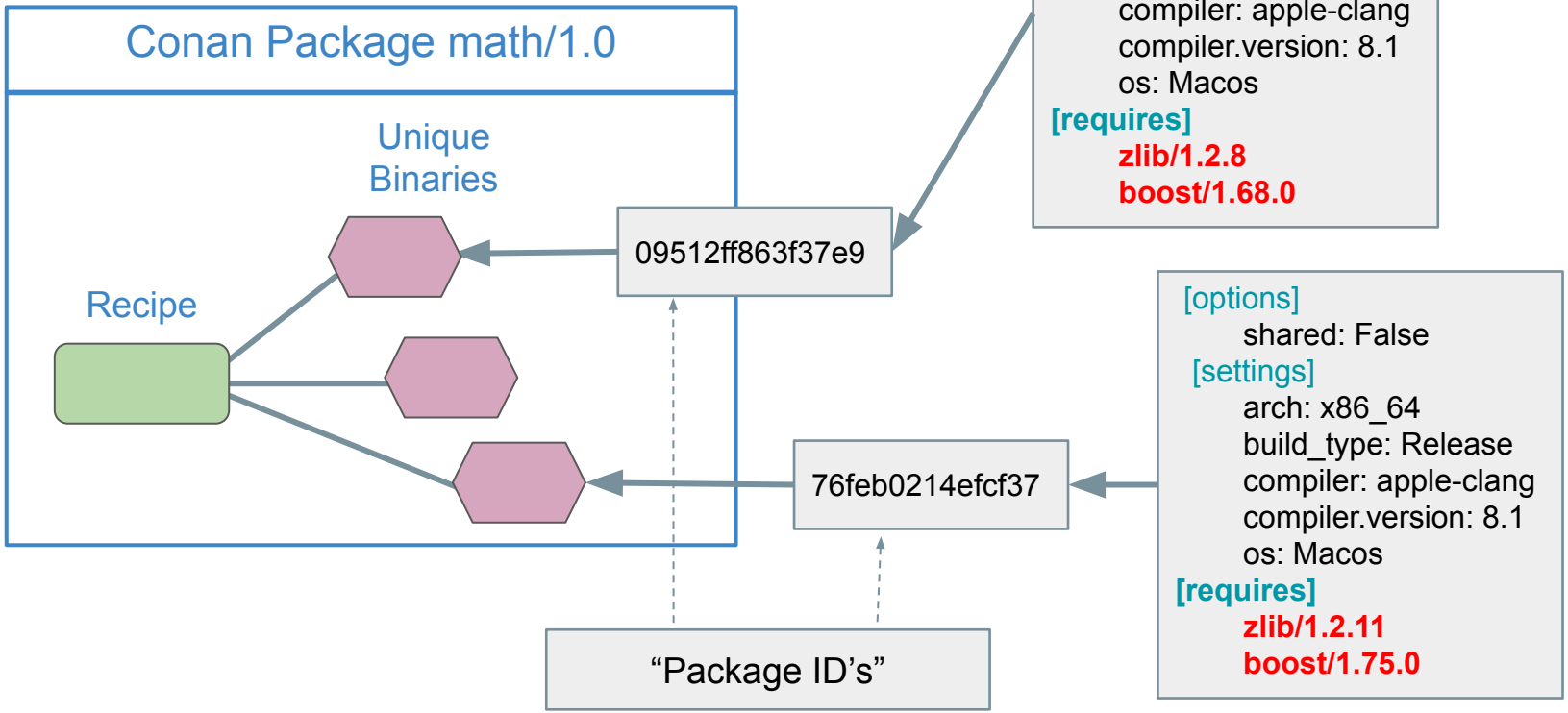
Conan 1.X: Binary model



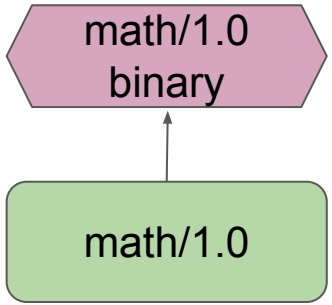
Conan 1.X: Binary model



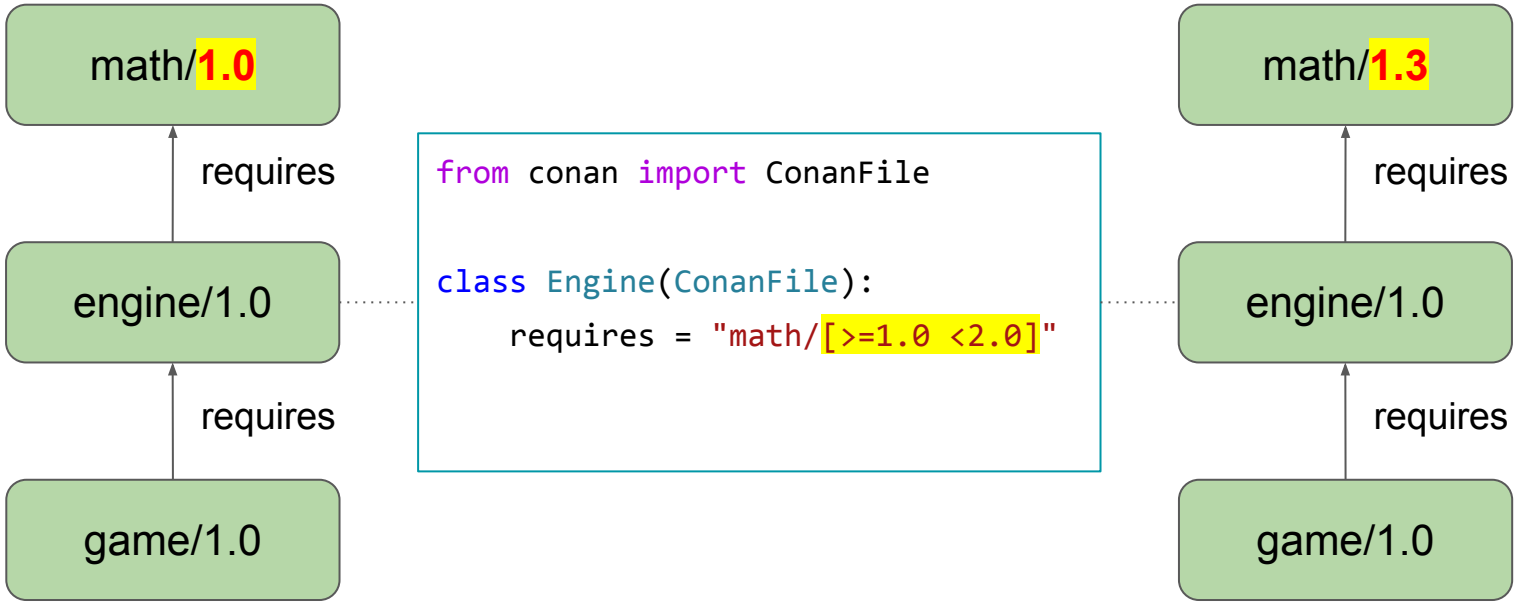
Conan 1.X: Full Binary model



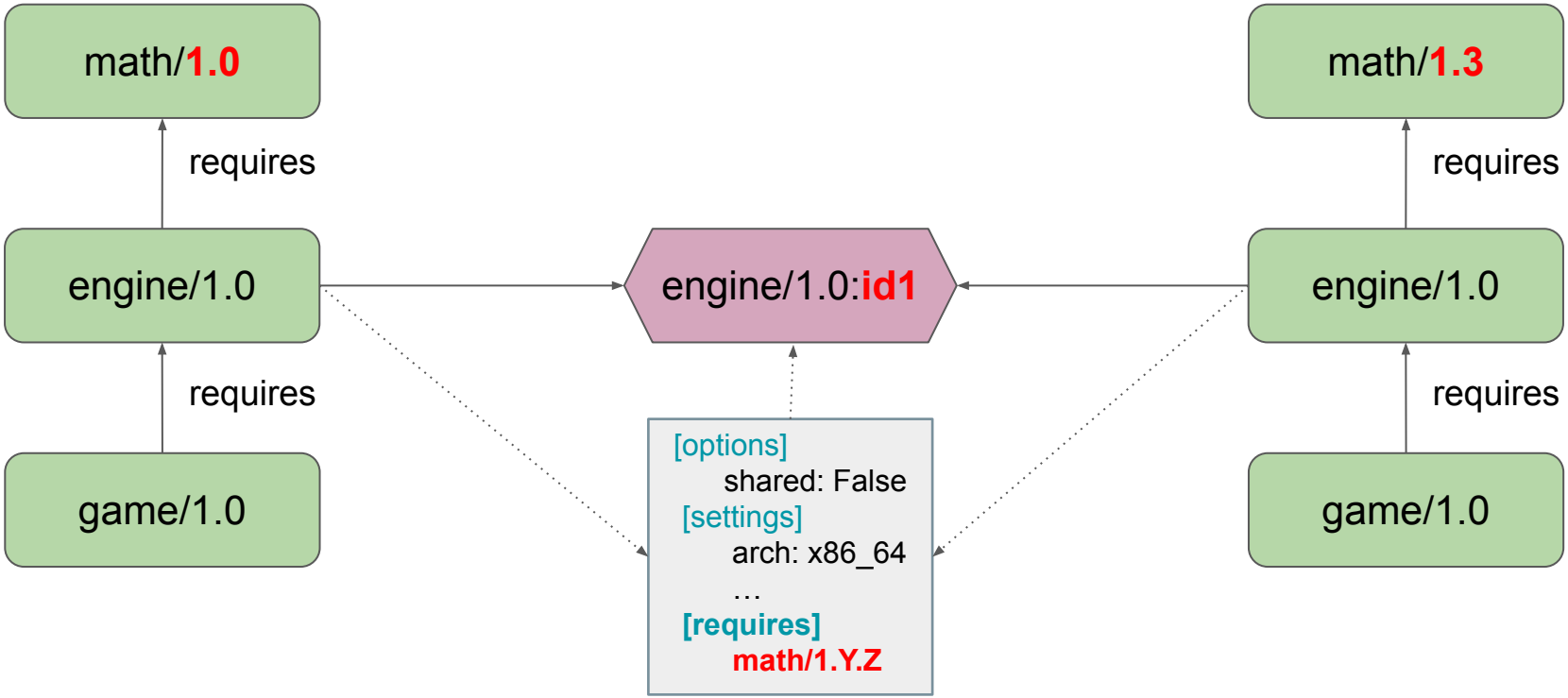
Can’t be wrong with semver



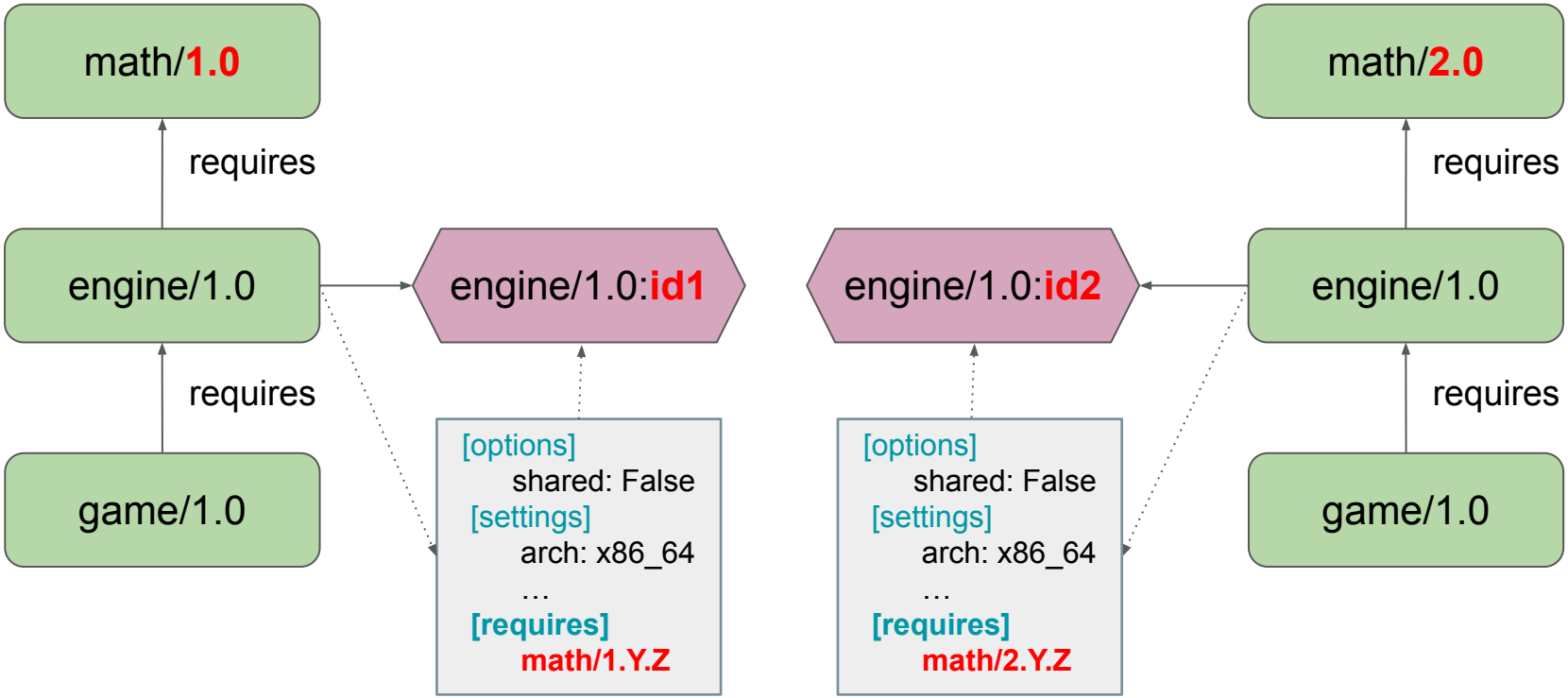
Conan 1.X default package_id_mode = semver



Conan 1.X default package_id_mode = semver

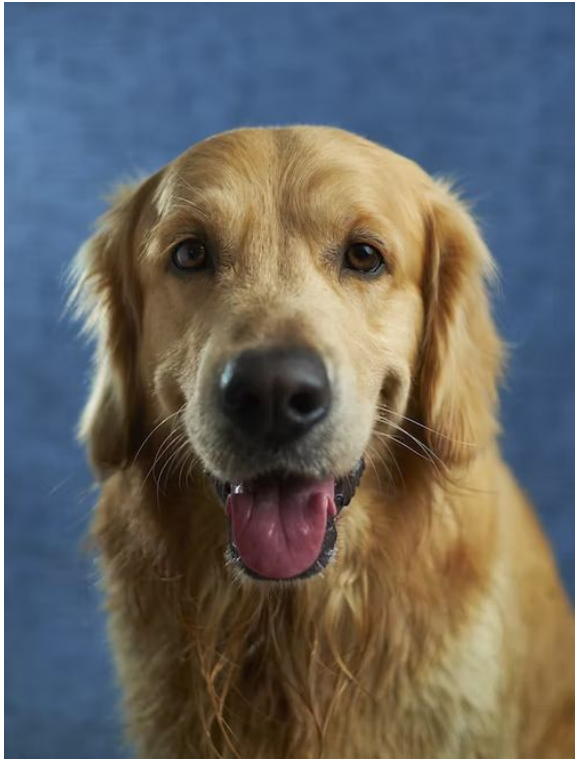


Conan 1.X default package_id_mode = semver



Semver in C and C++ for binaries?

- When boost last bumped the major version?
- GENERATION.MAJOR.MINOR.(PATCH) (OpenSource)
- Major differences when changing shared/static/header package type (native compilation model with inlining and embedding)



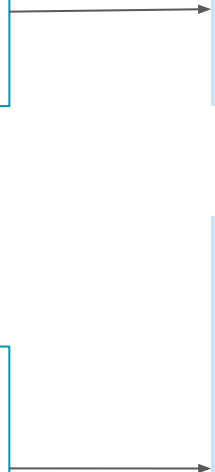
Shared library linking static

math2.cpp

```
int add(int a, int b){
  return a + b;
}
```

engine.cpp

```
#include "math2.h"
int move3d(int x, int y, int z){
    return add(x, add(y, z));
}
```



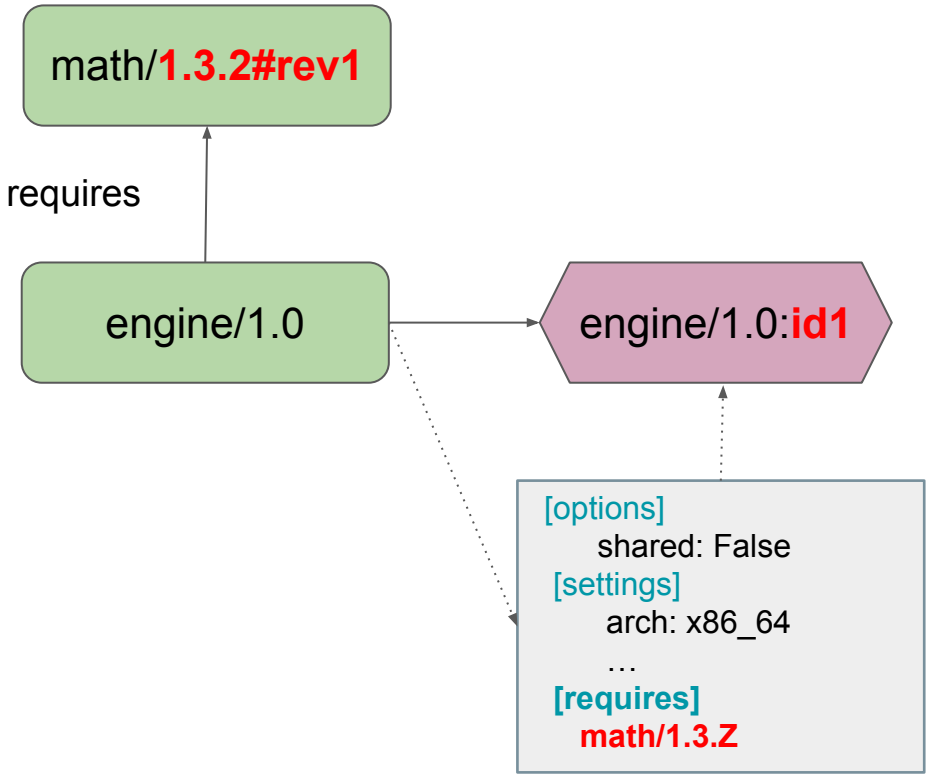
math2.lib

```
?add@@YAHHH@Z (int __cdecl add(int,int)):
  ...
  ...00011: 03 C8              add         ecx,eax
  ...
```

engine.dll

```
?move3d@@YAHHHH@Z (int __cdecl move3d(int,int,int)):
  ...
  ...00021: 8B 4C 24 30        mov         ecx,dword ptr [rsp+30h]
  ...00025: 8B 54 24 40        mov         edx,dword ptr [rsp+40h]
  ...00029: 8B 4C 24 38        mov         ecx,dword ptr [rsp+38h]
  ...0002D: E8 00 00 00 00     call        ?add@@YAHHH@Z
  ...00032: 8B D0              mov         edx,eax
  ...00034: 8B 4C 24 30        mov         ecx,dword ptr [rsp+30h]
  ...00038: E8 00 00 00 00     call        ?add@@YAHHH@Z
?add@@YAHHH@Z (int __cdecl add(int,int)):
  ...
  ...00011: 03 C8              add         ecx,eax
```

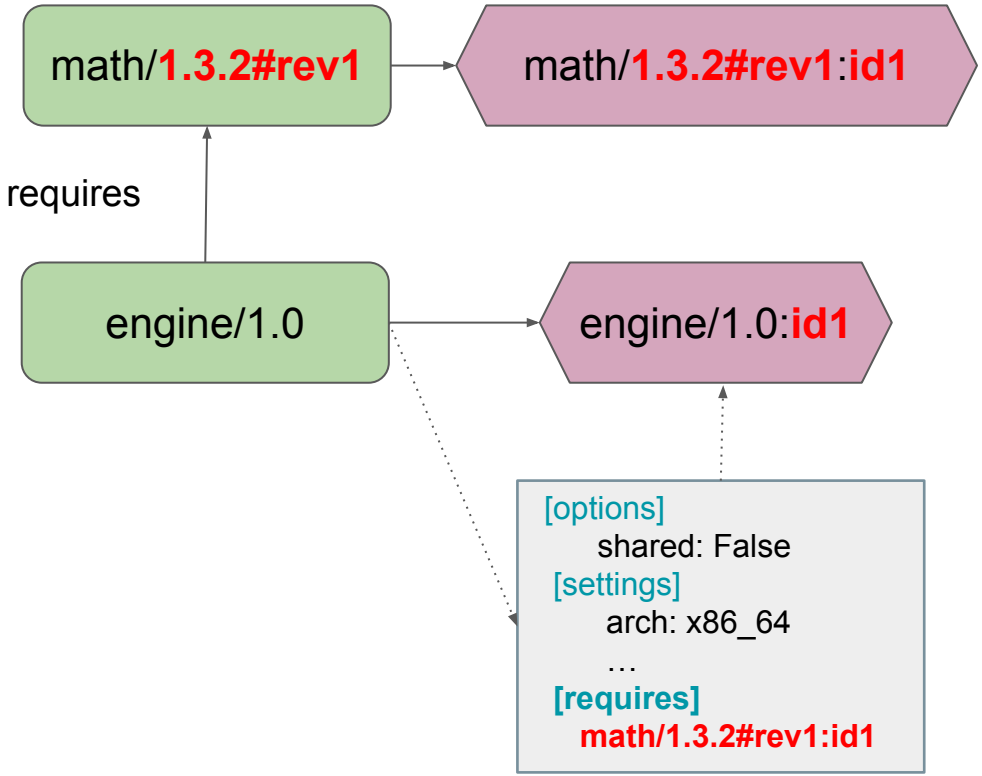
Conan 2.X default package_id modes: Non embed



non_embed_mode (=minor)

- app→ shared
- shared → shared
- static → static

Conan 2.X default package_id modes: Embed



embed_mode (=full)

- app→ static
- app → header
- shared → static
- shared → header
- static → header

Configuring package_id

global.conf

```
core.package_id:default_unknown_mode = semver_mode
core.package_id:default_non_embed_mode = minor_mode
core.package_id:default_embed_mode = full_mode
core.package_id:default_python_mode = minor_mode
core.package_id:default_build_mode = None
```

Demo

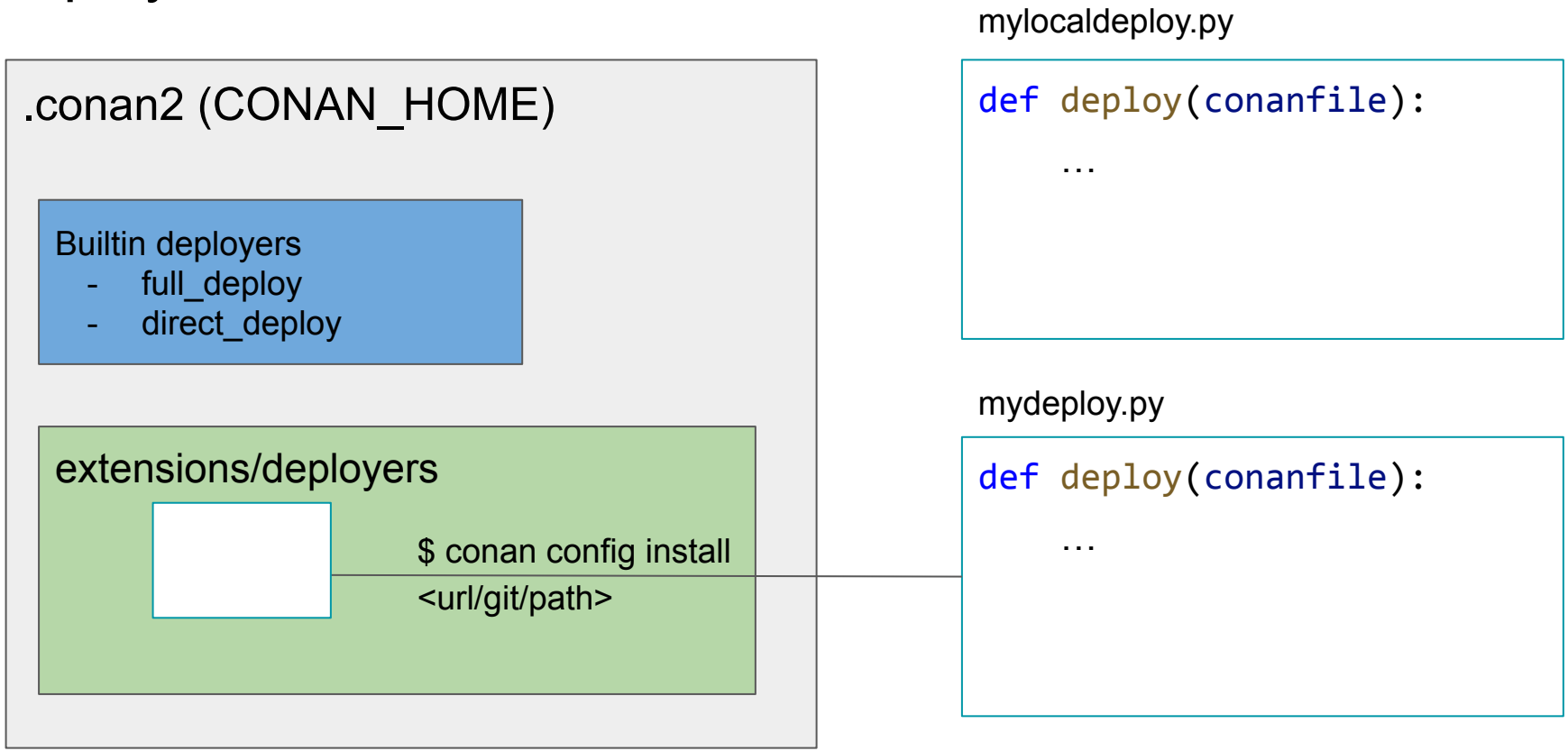
C and C++ right model

- Embed: full-mode (down to package-id)
- Non-embed: minor-mode
  - 1.2.PATCH => No build
  - 1.MINOR => Build
  - MAJOR => Breaking
- Fully customizable
- Major enabler for scale

3. Building a dam

App build & runs: great job!

Deployers


.conan2 (CONAN_HOME)
Builtin deployers
- full_deploy
- direct_deploy
extensions/deployers
$ conan config install
<url/git/path>
mylocaldeploy.py
def deploy(conanfile):
...
mydeploy.py
def deploy(conanfile):
...

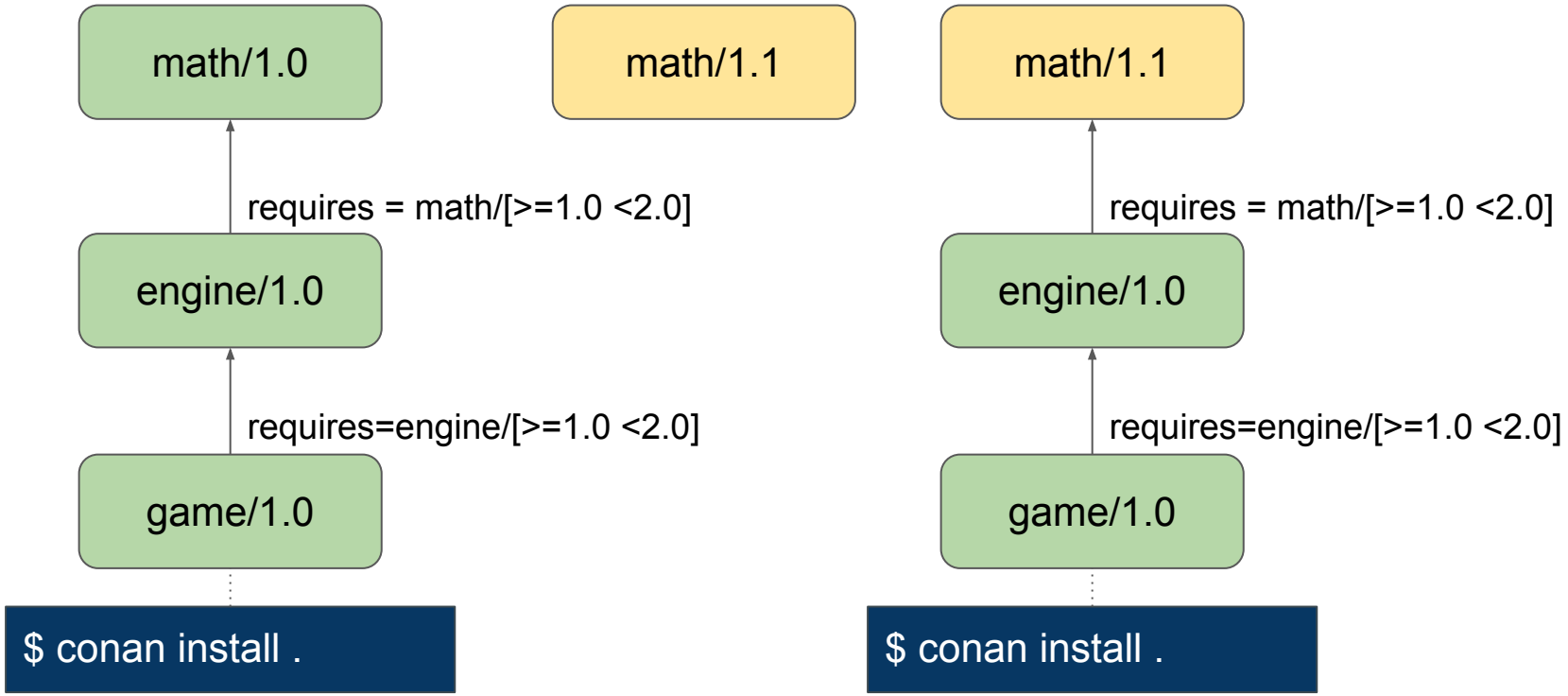
Demo

Deployers

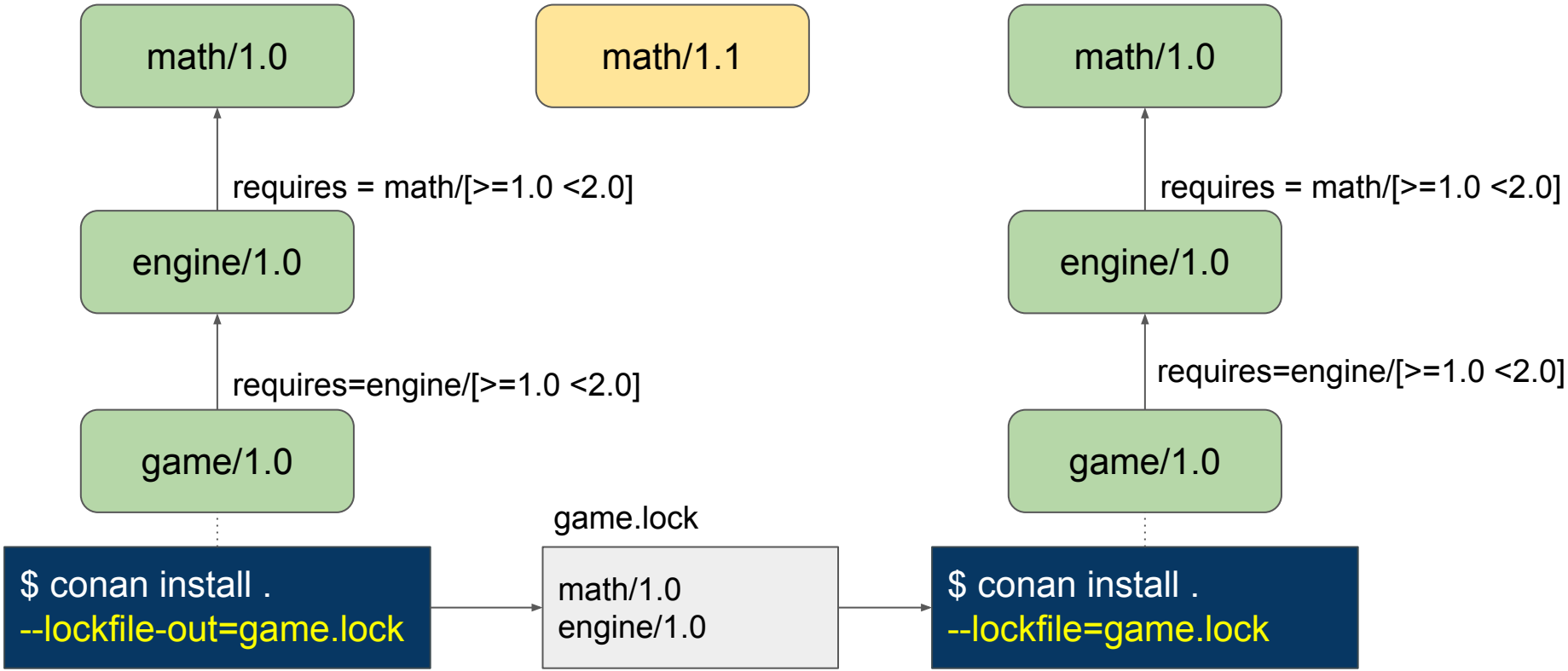
- Flexible way to extract artifacts from cache
- Automate post-conan tasks
- Not in recipes, scale
- User customizable, “conan config install” installable

4. Repeating yourself

Reproducible dependencies: the problem



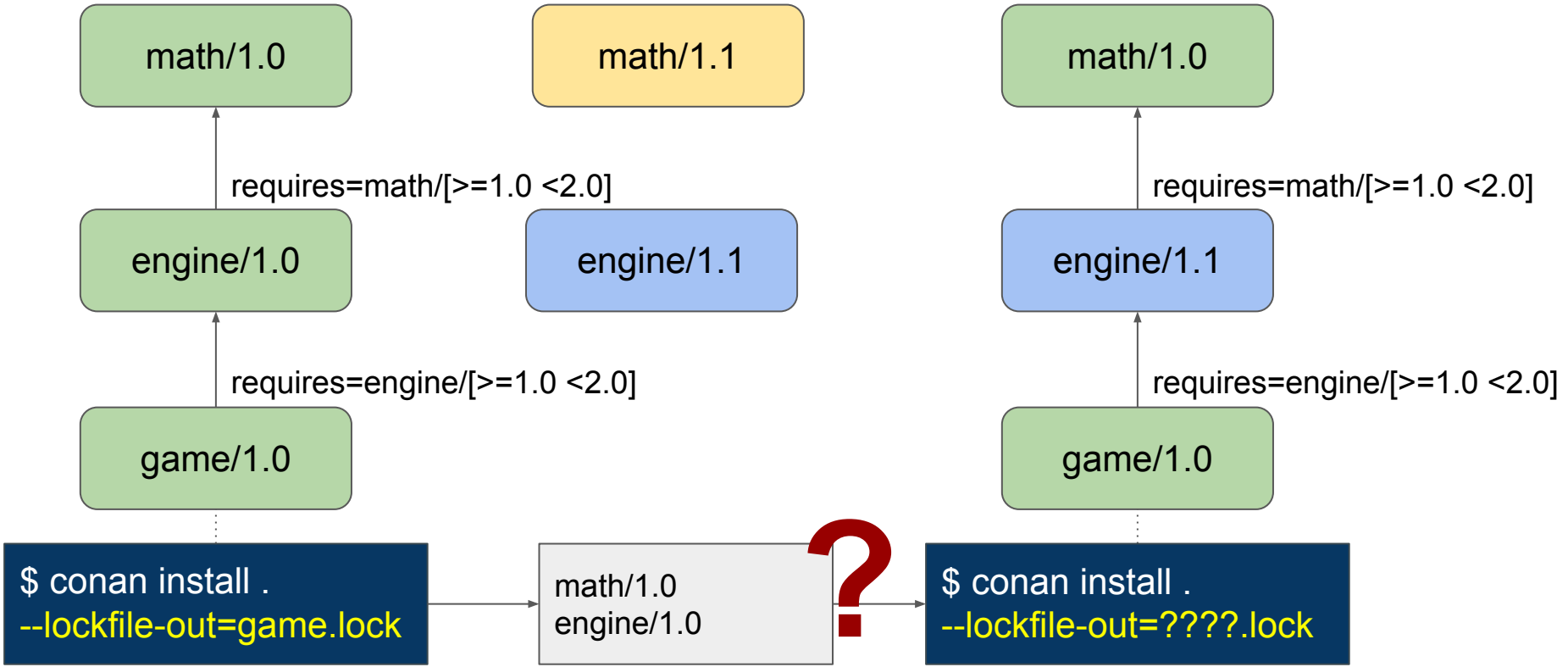
Reproducible dependencies: Lockfiles



Used feature

- 10% of issues last 2.5 years are lockfile related
- Decision tree:
  - Bump “requires = dep/xxx” versions of consumers
  - Use version ranges or revisions
    - Move forward aggressively
    - Lockfiles
- Estimate:
  - 25-40% use lockfiles
  - Demand > 75%

Unleashing the lockfiles for CI



Welcome Enterprise Devops for C and C++

- Enterprise escale can be high
- Enterprise/domain requirements can be challenging
- Continuous Integration at scale is critical
- Thinking beyond package and dependency management
  - Programming over time => SW engineering (T. Winters)
  - Dependency and Package management over time => Devops

Lockfiles 2.0

1.X

```
"0": {
      "ref": "engine/1.0#fd66..93b7",
      "requires": ["1"],
       },
"1": {"ref": "math/1.0#02fc..3729",
     }
```

2.0

```
"requires": [
  "math/1.0#02fc..3729",
  "engine/1.0#fd66..93b7"
],
"build_requires": [],
"python_requires": []
```

Demo

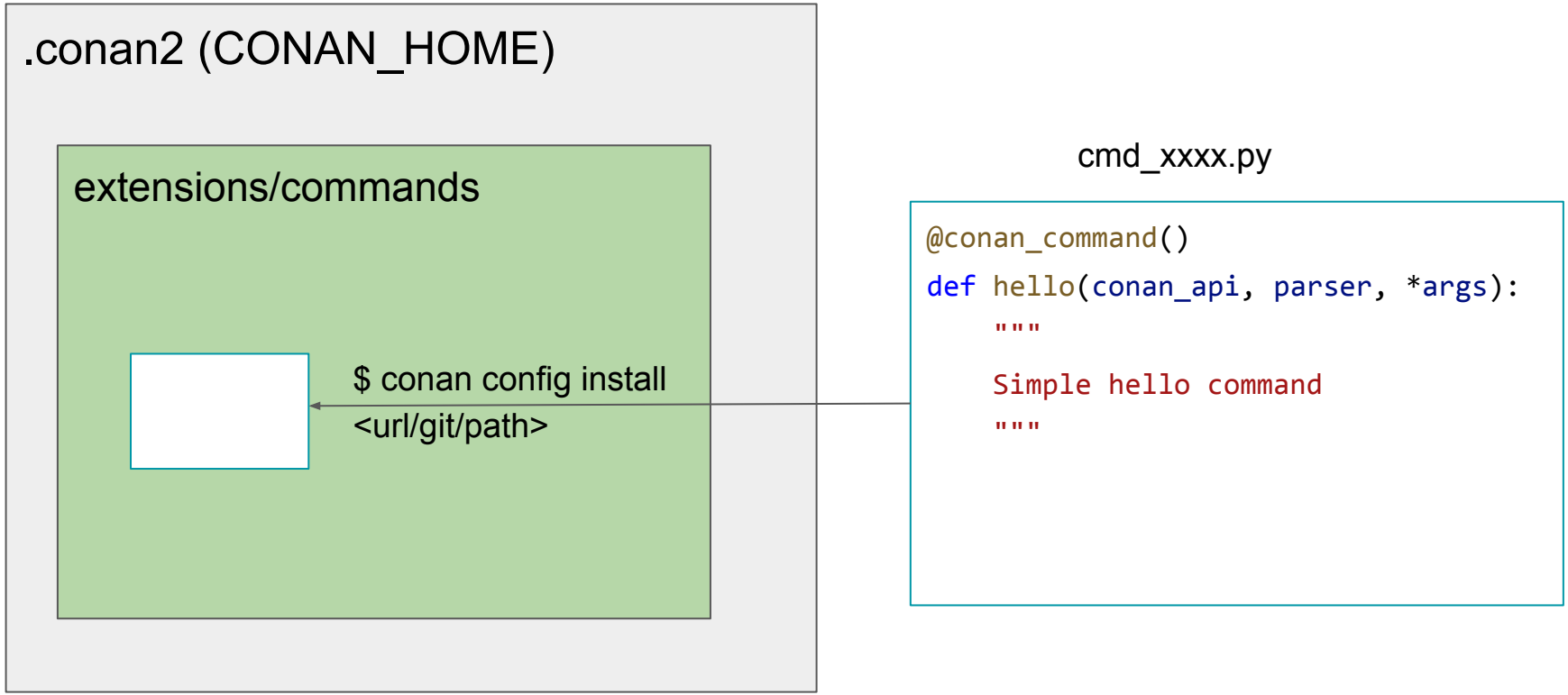
Lockfiles 2.0

- One lockfile for all configurations
- Easily mutable
- Easily understandable
- Fully strict and partial modes
- Easily mergeable
- Manual commands to modify (override)
- Possible to use multi-project
- Code in codebase 10x shorter
- Game changer for CI at scale

5. Dying of a thousand bites

A framework, not a tool

User custom commands



Custom command

```python
import json

from conan.api.output import ConanOutput
from conan.cli.command import conan_command

def output_json(msg):
    return json.dumps({"greet": msg})


@conan_command(group="My own commands", formatters={"json": output_json})
def hello(conan_api, parser, *args):
    """
    Simple command to print "Hello World!" line
    """
    msg = "Hello World!"
    ConanOutput().info(msg)
    return msg
```

Python API

```
# Conan 1.X API

class API:
    def install(path, profile_path, ...)
    def create(path, profile, ...)
    def lock(path, profile, ...)
    def search( ...)
    def remove(...)
    def upload(patterns, remote)
```

```
# Conan 2.0 API

class API:
    class RemoteAPI:
    class SearchAPI:
    class ProfileAPI:
        def get_profile(path, settings, options, conf)
    class GraphAPI:
        def graph(path, remotes, profiles, ...)
    class UploadAPI:
        def get_bundle(patterns, ..)
        def check_upstream(bundle, remote)
        def prepare(bundle)
        def check_integrity(bundle)
        def upload(bundle, remote)
```

Demo

Custom commands + Python API

- Very powerful automation extension, convenient for both developers and CI/devops
- Full Python API
  - Real building blocks
  - Not stabilized in 2.0, but 2.X
- conan config install https://…git/cmd-repo.git

Conclusions

New graph

New package_id

New deployers

New lockfiles

New custom commands + PythonAPI

Multi-revision cache

Package signing

Command wrapping

Profile checkers

Custom binary compatibility

New configuration and environment

Package immutability optimizations

… and many more

Conclusion

pip install conan==2.0-beta.3

https://conan.io