C++ now

Our Other C++ Interfaces

Mistakes to Avoid When Writing C++ Projects

Bret Brown

2024

Our Other C++ Interfaces

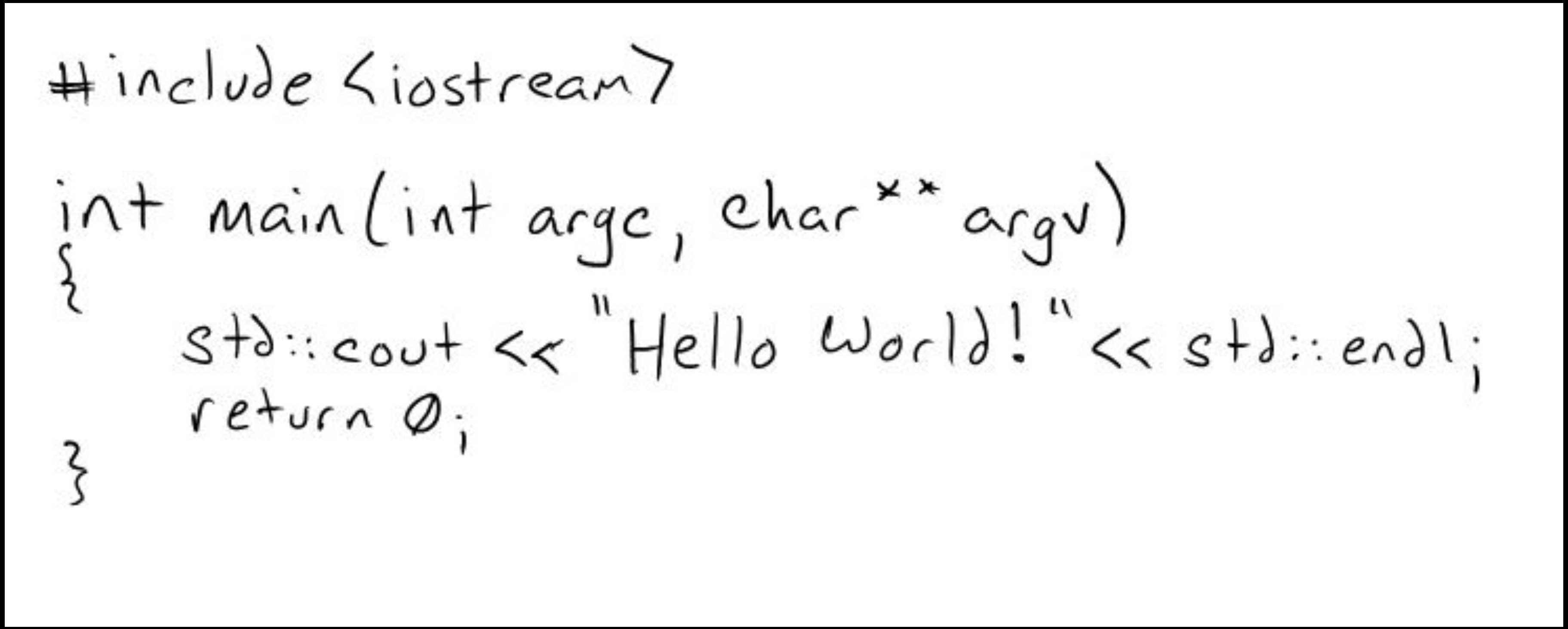
Mistakes to Avoid When Writing C++ Projects

C++Now
May 2, 2024

Bret Brown
Lead, C++ Infrastructure
Developer Experience

TechAtBloomberg.com

Why Doesn't This Compile?



```
#include <iostream>

int main(int argc, char** argv)
{
    std::cout << "Hello World!" << std::endl;
    return Ø;
}
```

Source code by James McNellis via StackOverflow
https://stackoverflow.com/questions/5508110/why-is-this-program-erroneously-rejected-by-three-c-compilers

Bloomberg

Engineering

Needed: Good *Project* Design

- Users don’t consume ISO-specified C++ text
- Users consume projects more than C++ code
- Interfaces of **projects** need good design too!

Bloomberg

Engineering

C++ Design? Yes, Please!

- SOLID
- Design by Contract
- Category Theory
  - Lambda calculus, monads, functors, etc.
- YAGNI
- DRY

Do we apply our design expertise to our projects?

Challenge: Lots of Workflows!

- Development environments
- Analysis builds
- Package builds
- License scanning tools
- Static analysis tools

So:

- X Supporting all explicitly
- ✔ Considered project design

Agenda

- Concepts and Theory
- Pragmatic Recommendations
- Positive Example

Concepts and Theory

TechAtBloomberg.com

Bloomberg

Engineering

Illustration:
Project Design by Contract

TechAtBloomberg.com

Bloomberg

Engineering

What is a “Project”?

For this talk, a *project* is a *source release*

Generally, a snapshot of a repo:

- Source code
- Build rules
- Tests
- Docs

….and more!

Outline of a Contract

- Precondition
- Operation
- Postcondition

Contract Preconditions

- Machine is Ubuntu between 23.04 and 24.04
- Build requires Ubuntu packages: `cmake`, `ninja-build`, `g++`
- `$PWD` is a working copy of https://github.com/bretbrownjr/zerocode.git

Project Operation

Operation “Build with CMake”:

- `cmake -B build -S . -G Ninja`
- `cmake --build build`

-
  ```
  ctest --test-dir build \
    --output-junit build/xunit/results.xml
  ```

-
  ```
  DESTDIR=staging cmake --install build
  --component libzerocode-dev
  ```

Contract Postconditions

- All commands exit non-zero
- Test results: `build/xunit/results.xml`
- Files will exist:
  - `staging/usr/local/include/zerocode.hxx`
  - `staging/usr/local/lib/libzerocode.a`
  - `staging/usr/local/lib/pkgconfig/zerocode.pc`
  - `staging/usr/local/lib/cmake/zerocode/zerocode-config.cmake`

Contract Specification

- How do we communicate that contract?
- How do we support that contract?
- When is a problem a bug? A user error?

Breaking Contracts

Contract changes break code:

- CMake files
- Dockerfiles
- CI configurations
- etc.

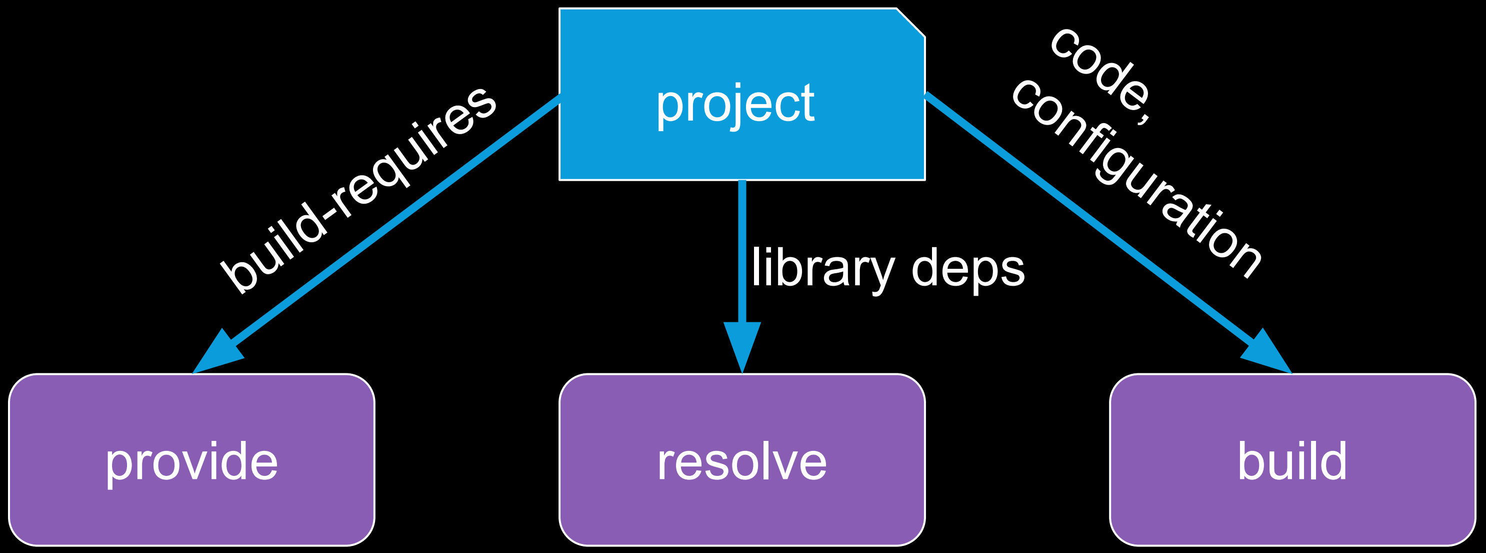
C++ projects are unavoidably polyglot!

Project Design *Domains*

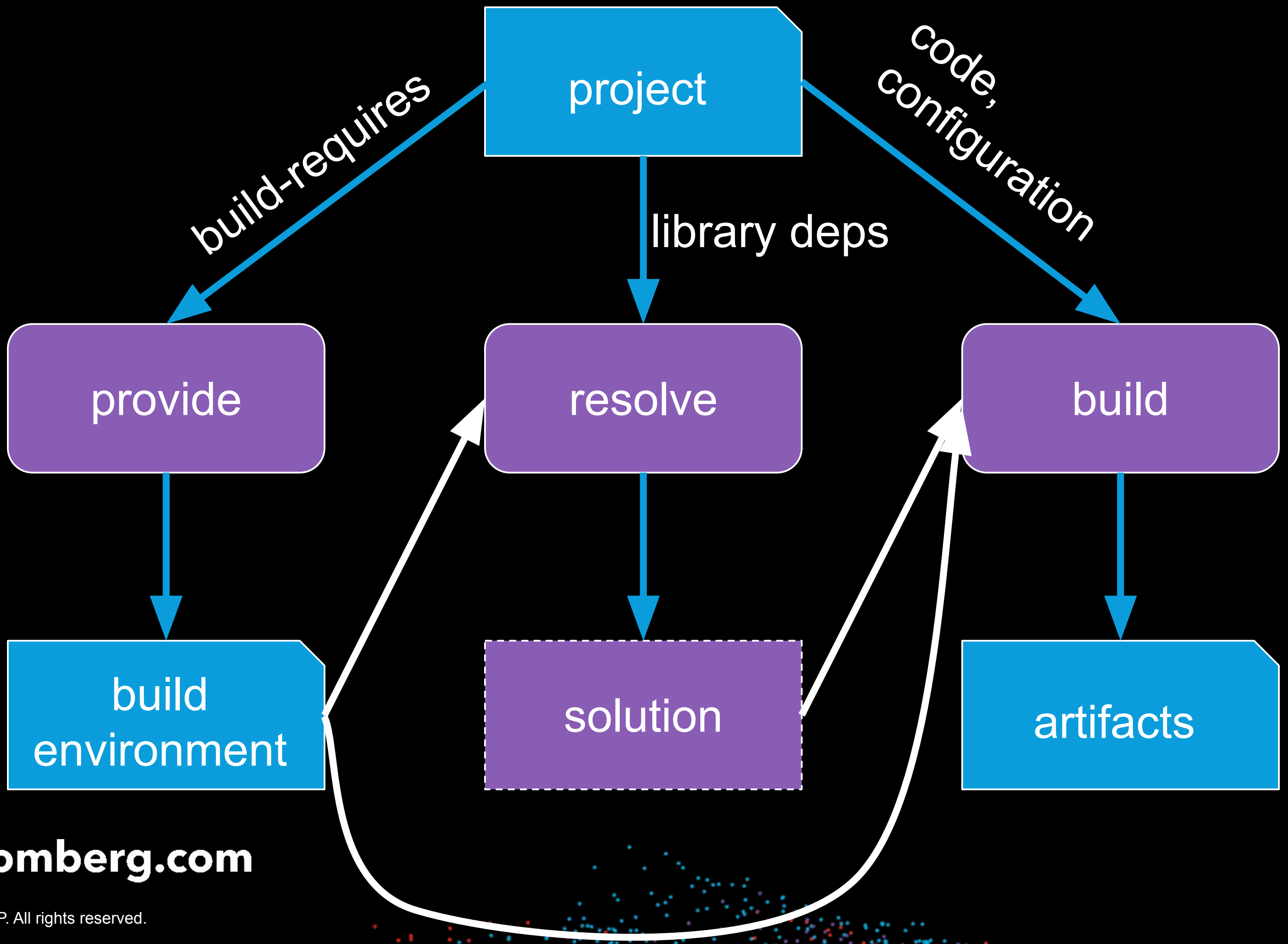
Domains for Build Workflows

- Provision
- Resolution
- Build
- Package

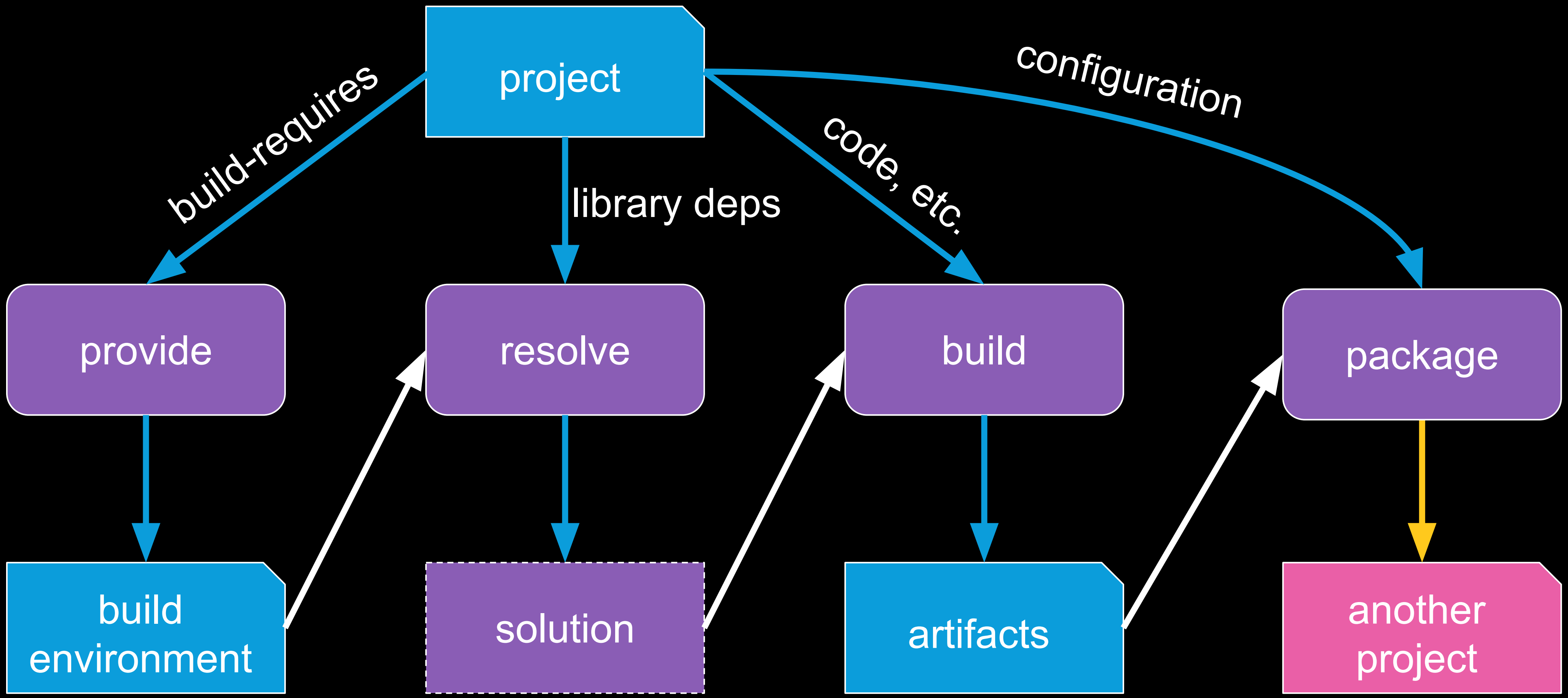
Project Interfaces: Build Workflows



Build Workflow Interconnection



Build Workflow and Consumers



Build Workflows: Node Identities

Lots of transitive graph building!

What are the nodes?

- Provision: projects
  - `cmake`, `libzerocode-dev`
- Resolution: logical dependencies
  - `zerocode`, `zerocode::zerocode`, `//zerocode:zerocode`
- Build: inputs, commands, outputs
  - `/usr/bin/cmake`, `zerocode.hxx`, `libzerocode.a`

Note: Steps validate outputs of previous steps

Interfaces for Other Domains

- Symbol tables for linkers
- License scanning
- Static analysis and security scanning
- Filesystems

Not typically graph-oriented, but still interfaces!

When Changing

- We'll be discussing stable interfaces
  - Generally, these are what we want

- When changing:
  - Provide both legacy and new interfaces
  - Communicate, warn, allow opt-outs
  - Give users time to adjust
  - Remove legacy interface (when possible)

Pragmatic Recommendations

TechAtBloomberg.com

Bloomberg

Engineering

Mistake: Unclear Project “Contracts”

- Is this a “real” project?
- What is it for?
- How does it compare to `{other project}`?
- Is this project supposed to build this way?

Instead: Have a README

- Describe the interfaces to your project there

- While you’re at it, add the other basics
  - Introduction
  - Goals and Scope
  - Developer documentation
  - Contributing guide

Mistake: Inconsistent/Claimed Project Name

- Your project needs to be well-identified
  - When downloaded, released, signed
- ⚠️ Names like util and db
- ⚠️ Three-letter-acronyms!
- X Incomplete forking
- X Ad hoc vendoring
- ⚠️ Changing names =~ forking

Instead: Do Some Homework

Looking for “zerocode” in arbitrary C++ packaging ecosystems:

- ArchLinux AUR: none
- vcpkg: none
- ConanCenter: none

Verdict: Seems unclaimed!

Other languages?

- crates.io: none
- PyPI: none

Mistake: Neglecting Library Filenames

What: `zerocode` given `libzerocode.a` or `libzerocode.so`

- ⚠️ `-L/usr/lib -lzerocode.debug`
- ✗ `-L/usr/lib -lzero/code`
- Header-only libraries claim their filenames!

Instead: Name your library files after your project!

- ✔ `-L/usr/lib -lzero-code`
- ✔ `-L/usr/lib -lzero_code`
- ✔ `-L/usr/lib -lzerocode`
- ✔ `-L/usr/lib/debug -lzerocode`
- ✔ `/usr/lib/libzerocode.a`

Assume case insensitive filesystems!

Mistake: Ignoring Users with Build Systems

- Hi! We’re almost everyone!
- We reference your project in our build configurations
- ಠ_ಠ No ecosystem-wide interop yet

Instead: Define a Build System Identity

How do build systems describe you as a dependency?

- `zerocode::zerocode` from `zerocode-config.cmake`
- `zerocode` maps to `zerocode.pc`

If you use CMake:

- `install(EXPORT ... NAMESPACE zerocode::)`

If you don't:

- Consider shipping generated CMake anyway
- Ship pkg-config metadata if you target POSIX
- Help drive convergence: https://github.com/cps-org/cps

Mistake: Unconsidered Header Identity

These are all “valid” references to one header

- `/usr/include/zerocode/core.hxx`
  - Contents of `libzerocode-dev`!
- `#include <zerocode/core.hxx>`
- `#include <core.hxx>`

Instead: Namespace Your Headers

- Each inclusion target should be unique
  - ✔ `#include <zerocode/core.hxx>`
  - ✔ `#include "zerocode.h"`
  - ⚠️ `#include "config.h"`
  - ⚠️ `#include "core/utils.h"`
  - ✗ `#include <utils>`
- Ensure `zerocode/core.hxx` exists in your repo
- Be consistent in your codebase

Mistake: Invented/Ambiguous File Extension

- Q: How do you identify a file as Java?
- Q: How do you identify a file as Python?
- Q: How do you identify a file as C++?
- A: Ask the build system? Best-effort lexxing?

Why? Common IDE, editor, and code awareness workflows

Instead: Use a Language-Specific File Extension

Define your files as being implemented in a specific language:

- ✔ `zerocode.cxx`
- ✔ `zerocode.hpp`
- ✔ `zerocode.c`
- ⚠️ `zerocode.h`
- X `zerocode`
- X `zerocode.codegen`

Mistake: No Correctness Contracts

- Useful projects will get ported, patched
- We need support helping determine correctness
  - Users
  - Package maintainers
  - Contributors
- Modern build systems have standard test hooks!

Instead: Provide Tests

- Some accurate, reliable tests are better than nothing
- Define at least the contracts you can commit to
- If someone patches your project, did anything break?

Mistake: Little/No Build Support

For instance:

- ✗ No build instructions
- ✗ Source files and a README
  - Looking at you, header-only projects
- ⚠️ Bespoke build systems
  - Makefiles generally qualify

Instead: Have a Build System

- If you don’t have a strong opinion, use CMake
  - ✔ Portable
  - ✔ Minimal dependency list
  - ✔ Test workflow
  - ✔ Install workflow
  - ✔ Packaging integrations

- If you’re disrupting CMake, best of luck!
  - Then, please have a simple project structure

Mistake: Overspecifying Build Rules

- Many choices must be made before environment provision
  - Architecture tuning
  - Dependency pinning
  - Compilation toolchain
  - Standard version
  - Thread sanitizer
  - See also: Hyrum’s Law

Mistake: Overspecifying Build Rules – CMake Edition

- ❌ Hardcoding CMAKE_* variables
  - `CMAKE_CXX_FLAGS`
  - `CMAKE_TOOLCHAIN_FILE`
  - `CMAKE_BUILD_TYPE`

- ⚠️ Fiddling with build types in CMakeLists.txt
  - Is everyone fiddling compatibly?

Instead: Defer to “Higher Level” Contexts

- Invest in dependency management tools
  - Monorepo
  - Packaging system

- Inject more into your build
  - `CXXFLAGS` and `LDFLAGS`
  - CMake toolchain files
    - Conan and vcpkg know how to leverage these

- Analogous: Inversion of Control

Needed: An interoperability standard for “build flavors”

Mistake: Treating Warnings as Errors

- ⚠️ What does "all warnings are errors" mean?
  - e.g., `/WX` or `-Werror`
- ⚠️ What about flaky `-Wall` warnings?
- ❌ Have you tested that specifically?
  - What about for GCC 15 and Clang 20?

Instead: Allow Choice Per Workflow

- ✔ Support in build system instead
  - See CMake's COMPILE_WARNING_AS_ERROR
  - Other build and packaging systems please note
- ✔ Use `CXXFLAGS` to match `CXX`
  - `-Werror=all -Wno-error=deprecated-declarations`
- ✔ Drive diagnostics from `compile_commands.json`

Demonstration: `zerocode`

zerocode

- https://github.com/bretbrownjr/zerocode
- `zerocode` is a C++ library with zero code[1]
- An experiment in project structure through “negative space”

[1] Actually it has code in `CMakeLists.txt` and Dockerfiles. See footnote 1 in `README.md`.

zerocode: Project Size

```
$ ./some_command | wc -l | sort
    2 ./src/zerocode/zerocode.cxx
    2 ./src/zerocode/zerocode.hxx
    5 ./test/zerocode/zerocode.test.cxx
    8 ./test/zerocode/CMakeLists.txt
   11 ./src/zerocode/zerocode.pc
   13 ./CMakeLists.txt
   17 ./LICENSE
   21 ./src/zerocode/zerocode.cps
   28 ./.ci/docker/ubuntu.Dockerfile
   29 ./.ci/docker/rockylinux.Dockerfile
   53 ./src/zerocode/CMakeLists.txt
  275 ./README.md
  464 total
```

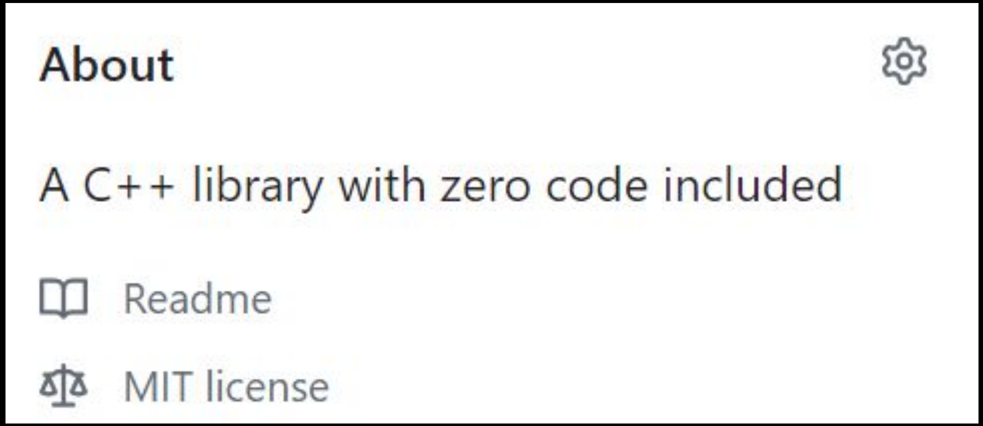
[1] Actually it has code in `CMakeLists.txt` and Dockerfiles. See footnote 1 in `README.md`.

zerocode: Top-Level Directory

```
├── CMakeLists.txt
├── LICENSE
├── README.md
├── src/
└── test/
```

zerocode: README and LICENSE

```
$ grep "^## " README.md
## About
## Building
## Usage
## Contributing
## Inspiration
```



About

A C++ library with zero code included

Readme

MIT license

TechAtBloomberg.com

Bloomberg Engineering

zerocode: src Directory

```
src/zerocode/
├── CMakeLists.txt
├── zerocode.cps
├── zerocode.cxx
├── zerocode.hxx
└── zerocode.pc
```

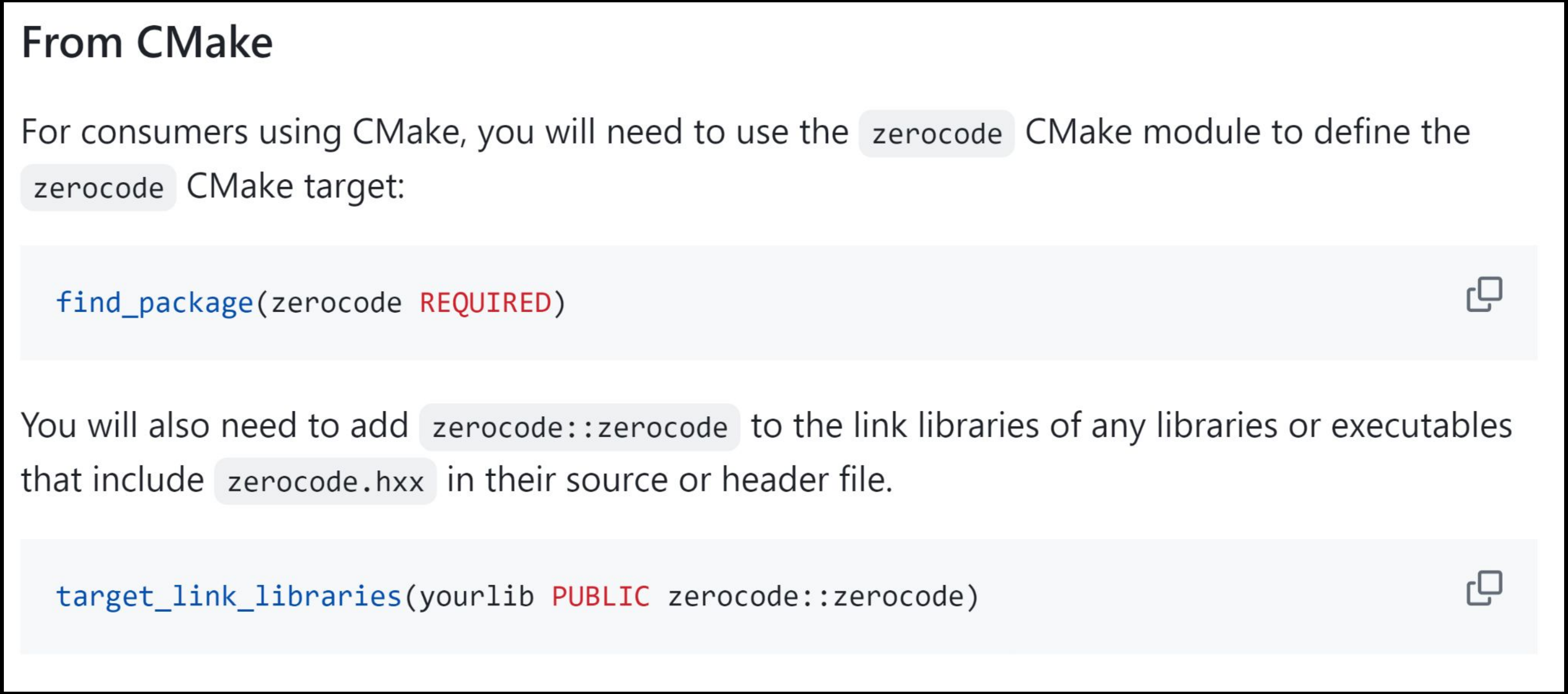
zerocode: All the Code

zerocode: All the Code (Unpreprocessed)

```
$ cat src/zerocode/zerocode.hxx
// Copyright © 2024 Bret Brown
// SPDX-License-Identifier: MIT

$ cat src/zerocode/zerocode.cxx
// Copyright © 2024 Bret Brown
// SPDX-License-Identifier: MIT
```

zerocode: Usage



From CMake

For consumers using CMake, you will need to use the `zerocode` CMake module to define the `zerocode` CMake target:

```
find_package(zerocode REQUIRED)
```

You will also need to add `zerocode::zerocode` to the link libraries of any libraries or executables that include `zerocode.hxx` in their source or header file.

```
target_link_libraries(yourlib PUBLIC zerocode::zerocode)
```

zerocode: pkg-config File

```
$ tail -5 src/zerocode/zerocode.pc
Name: zerocode
Description: A C++ library with no code
Version: 1.0.0
Cflags: -I${includedir}
Libs: -L${libdir} -lzerocode
```

- Aside: Note the `-lzerocode`… that’s how `pkg-config` works!

zerocode: CPS file

```
$ jq . src/zerocode/zerocode.cps
{
  "cps_version": "0.10.0",
  "name": "zerocode",
  "description": "A C++ project with no code",
  "license": "MIT",
  "version": "1.0.0",
  "default_components": [ "default" ],
  [...]
```

zerocode: CPS file, continued

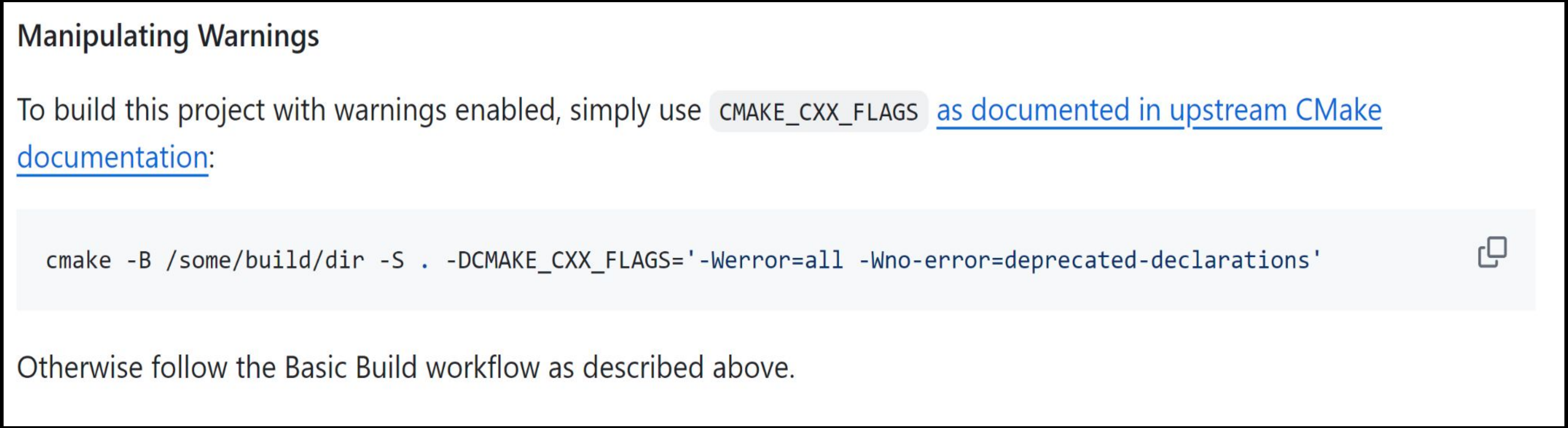
```
  [...]
  "components": {
    "default": {
      "type": "archive",
      "location": "@prefix@/lib/libzerocode.a",
      "includes": [ "@prefix@/include" ],
      "requires": []
    }
  }
}
```

TechAtBloomberg.com

Bloomberg
Engineering

zerocode: Warnings

- Yes, this is wired up in CI too



Manipulating Warnings

To build this project with warnings enabled, simply use `CMAKE_CXX_FLAGS` as documented in upstream CMake documentation:

```
cmake -B /some/build/dir -S . -DCMAKE_CXX_FLAGS='-Werror=all -Wno-error=deprecated-declarations'
```

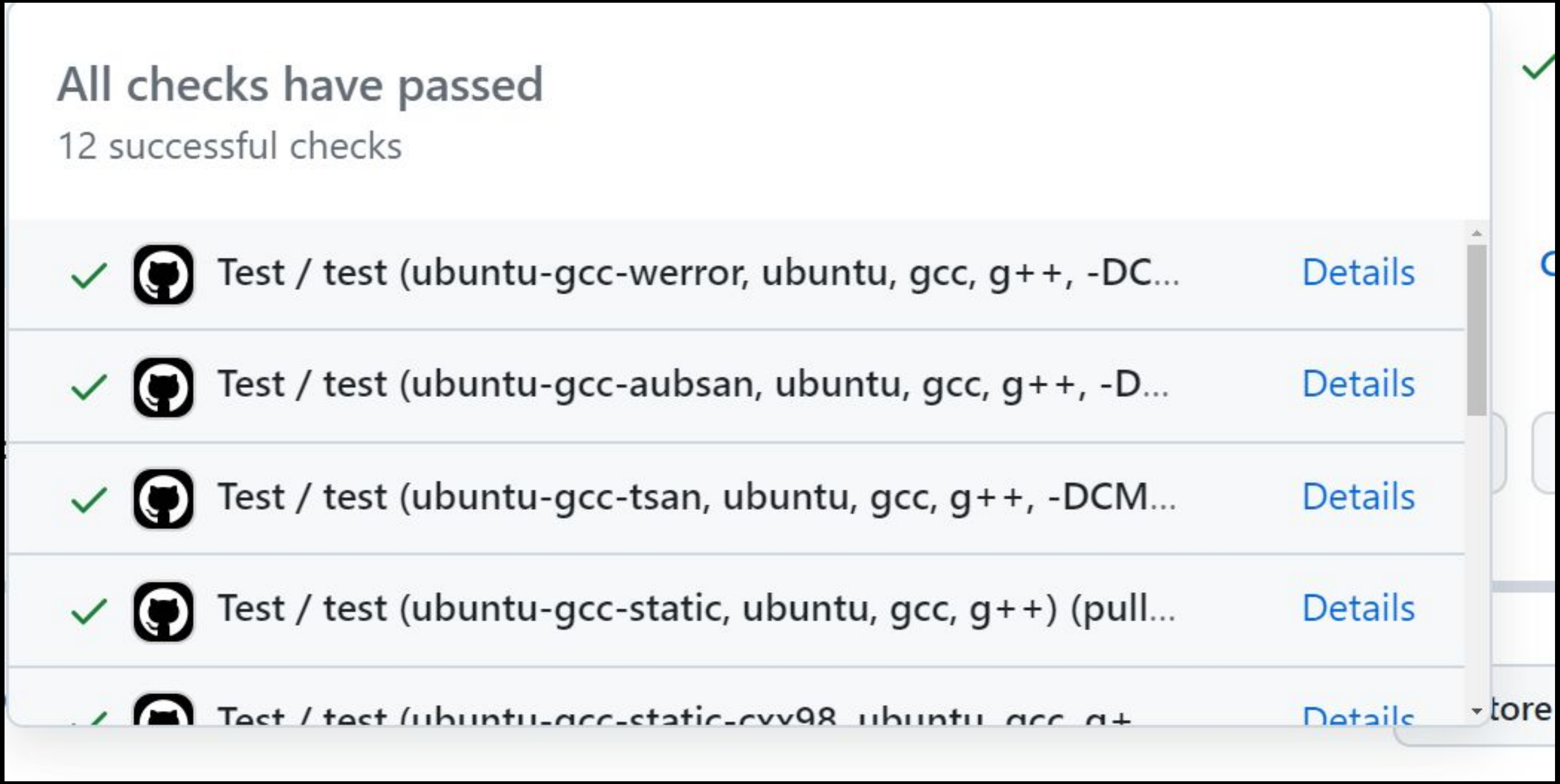
Otherwise follow the Basic Build workflow as described above.

TechAtBloomberg.com

Bloomberg Engineering

zerocode: CI-Tested Contracts

- GitHub Actions + Dockerfiles
- Your CI may look different



All checks have passed
12 successful checks

✓ Test / test (ubuntu-gcc-werror, ubuntu, gcc, g++, -DC... Details
✓ Test / test (ubuntu-gcc-aubsan, ubuntu, gcc, g++, -D... Details
✓ Test / test (ubuntu-gcc-tsan, ubuntu, gcc, g++, -DCM... Details
✓ Test / test (ubuntu-gcc-static, ubuntu, gcc, g++) (pull... Details
✓ Test / test (ubuntu-gcc-static-cxx98, ubuntu, gcc, g+ Details

Takeaways

- Let’s apply our design skills to our projects
- Projects interoperate to form connected ecosystems
- Example guidelines
- `zerocode`: application of principles and guidelines

Thank you!

https://TechAtBloomberg.com/cplusplus

https://www.bloomberg.com/careers

Contact me:
mail@bretbrownjr.com
https://github.com/bretbrownjr
https://x.com/bretbrownjr
https://mastodon.social/@bretbrownjr
https://linkedin.com/in/bretbrownjr
https://reddit.com/user/bretbrownjr
Bret Brown @ https://cpplang.slack.com

TechAtBloomberg.com

Bloomberg Engineering