10 Problems Large Companies Have With Managing C++ Dependencies and How to Solve Them

AUGUSTIN POPA

Introduction

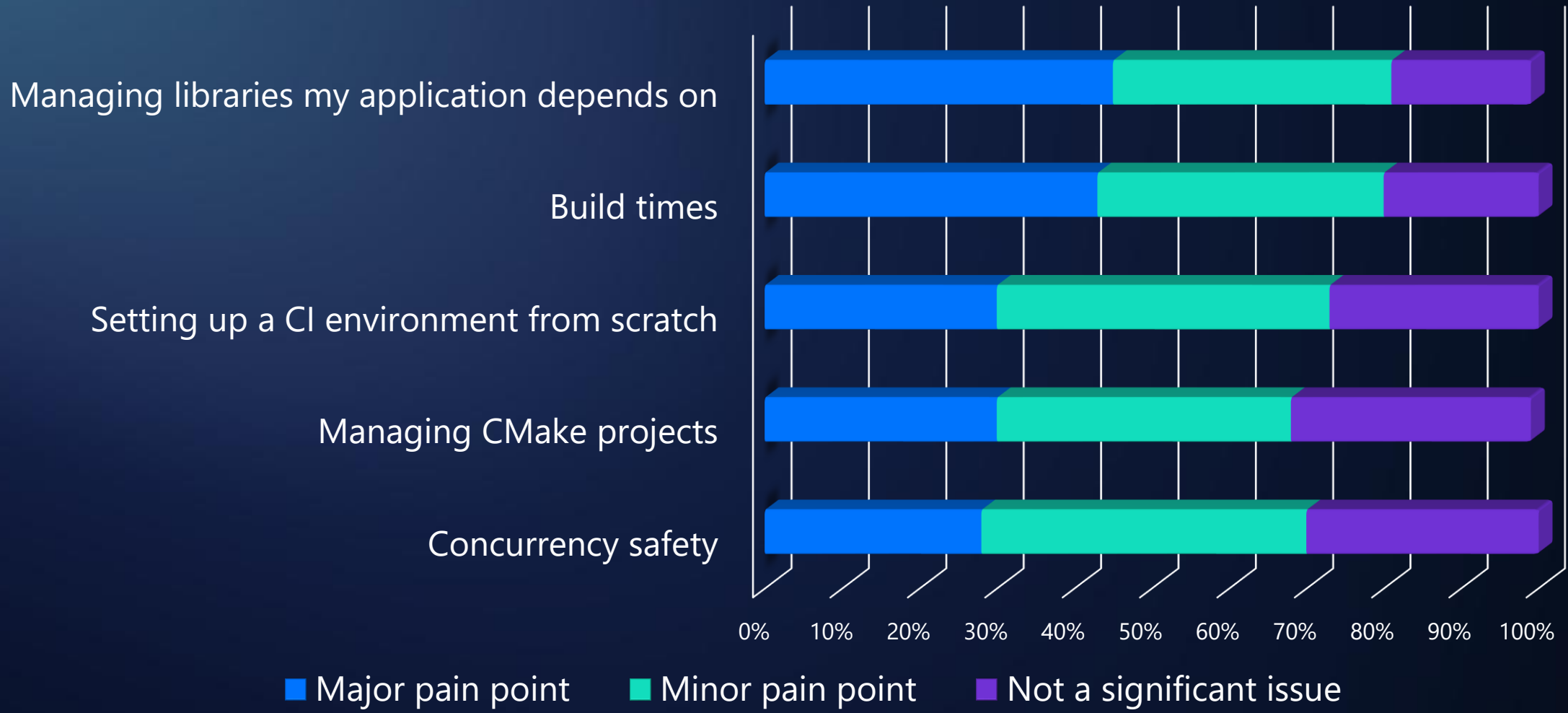
Augustin Popa

Senior Product Manager, vcpkg

Microsoft C++ Team

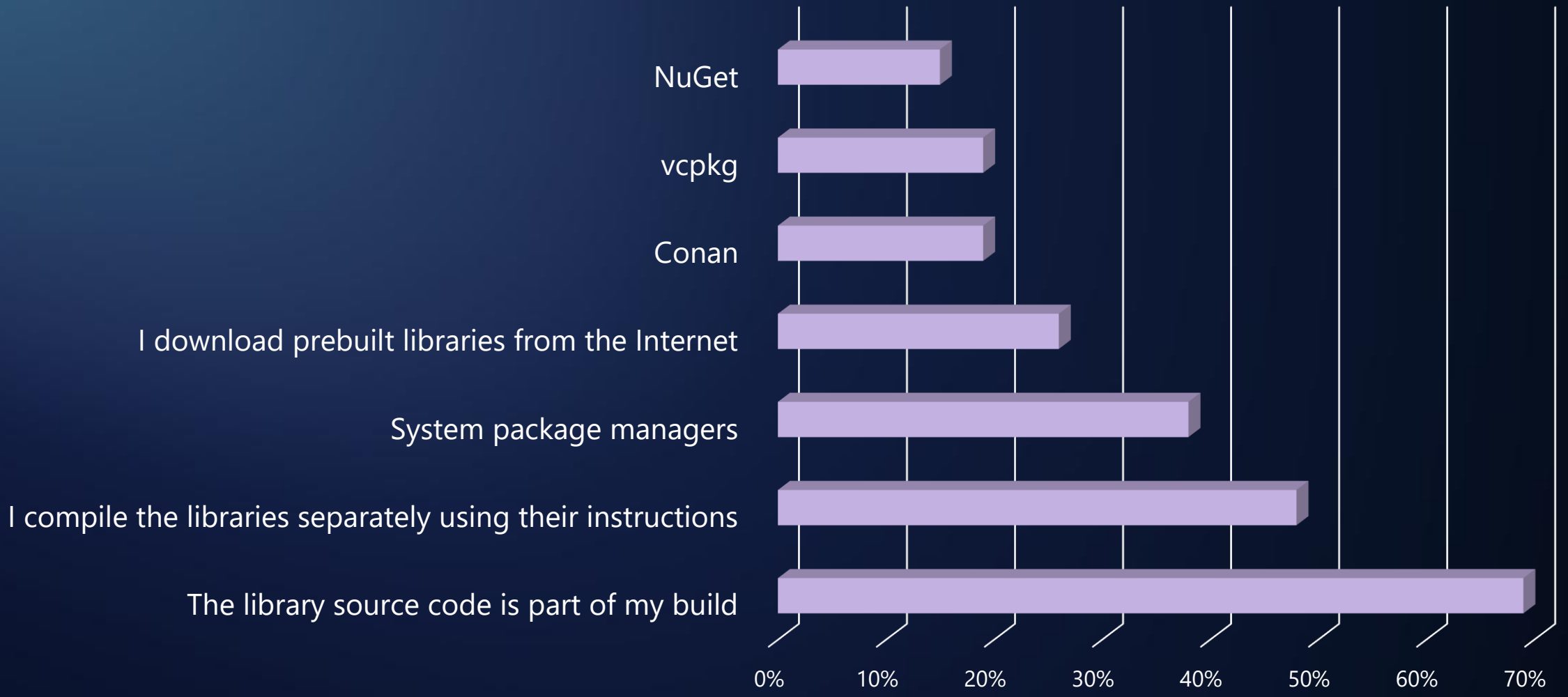
@augustin_popa on Twitter/X

Which of these do you find frustrating about C++ development?



Source: ISO C++ 2024 survey

How do you manage your C++ 1st and 3rd party libraries?

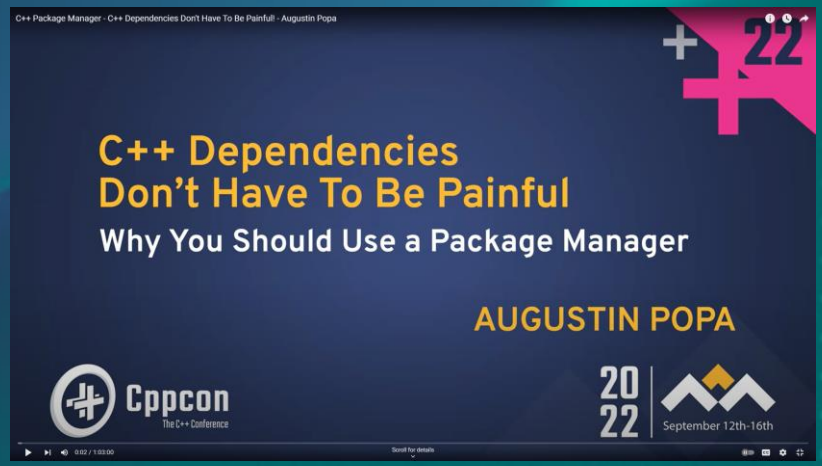


Source: ISO C++ 2024 survey

Conan vcpkg NuGet Custom solution

Most popular package managers for enterprises

For more on package managers, see:



Preview – 10 conclusions

1. Support building from source
2. Build a verified binary cache
3. Version using baselines
4. Build open-source with a package manager
5. Cache build assets internally
6. Monitor, prevent, and respond to vulnerabilities
7. Centralize common tasks
8. Produce SBOMs
9. Global, reproducible builds
10. Break large migrations down into smaller milestones

Demo Intro: Cataclysm: Dark Days Ahead

vcpkg + CMake + Visual Studio

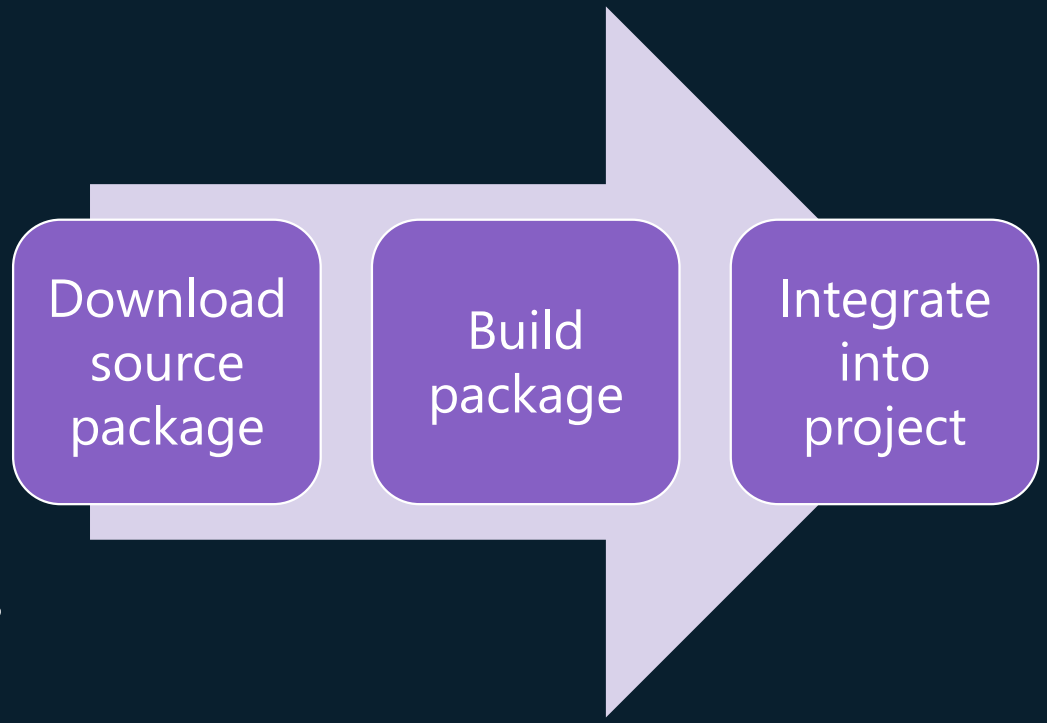
Problem 1: ABI incompatible C++ binaries

Compiler	Compiler version	Target OS
Target architecture	Optional library features	Other compiler flags

Problem 1: ABI incompatible C++ binaries
Solution 1: Build C++ dependencies from source

- Instead of consuming prebuilt libraries, establish a way to build them from source.

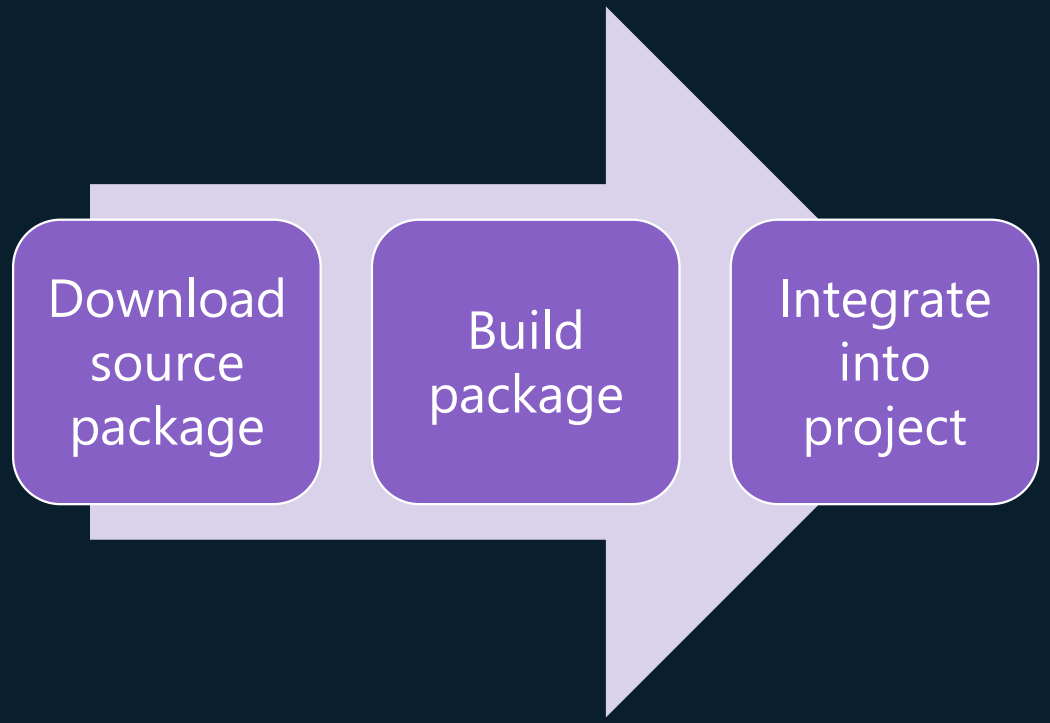
- Some approaches:
  - "Build from source" package manager like vcpkg or Conan
  - Leverage CI to build dependencies and use the binaries
  - Teams that own libraries share build instructions
  - Monorepo: build everything together



Problem 1: ABI incompatible C++ binaries
Solution 1: Build C++ dependencies from source

Benefits of building from source:

- No waiting for library owner to update binary
- Able to view, edit, and debug source code
- Easier and cheaper to distribute source packages than all possible binaries for all users

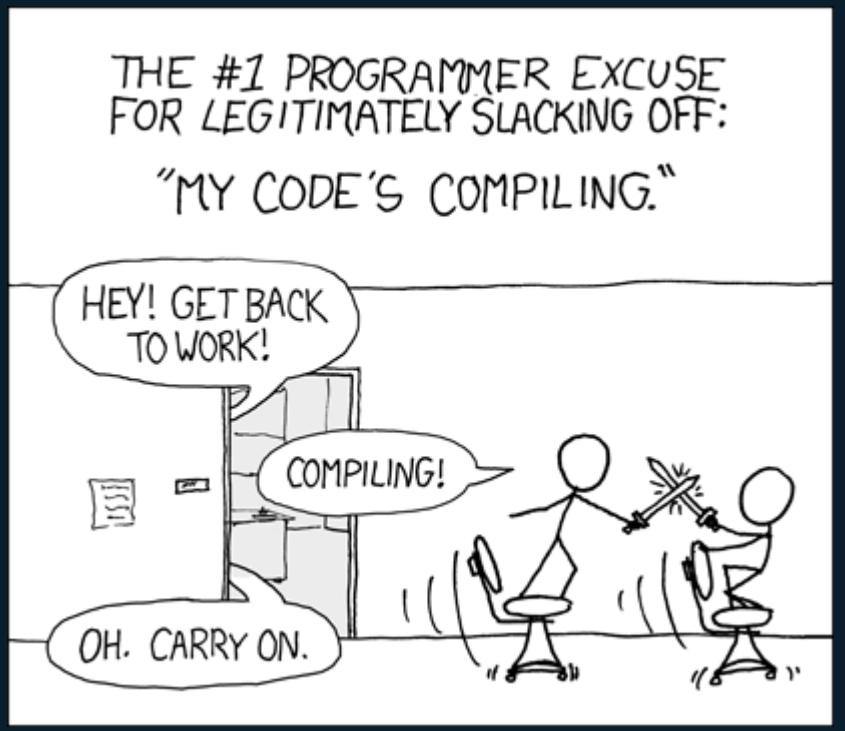


Cataclysm: Dark Days Ahead

Building dependencies from source

Problem 2: Build times are too long when building from source

- Source packages take time to produce usable binaries
- Build times: 2nd biggest pain point for C++ (ISO C++ surveys)
- Scales poorly with large projects or many distributed teams

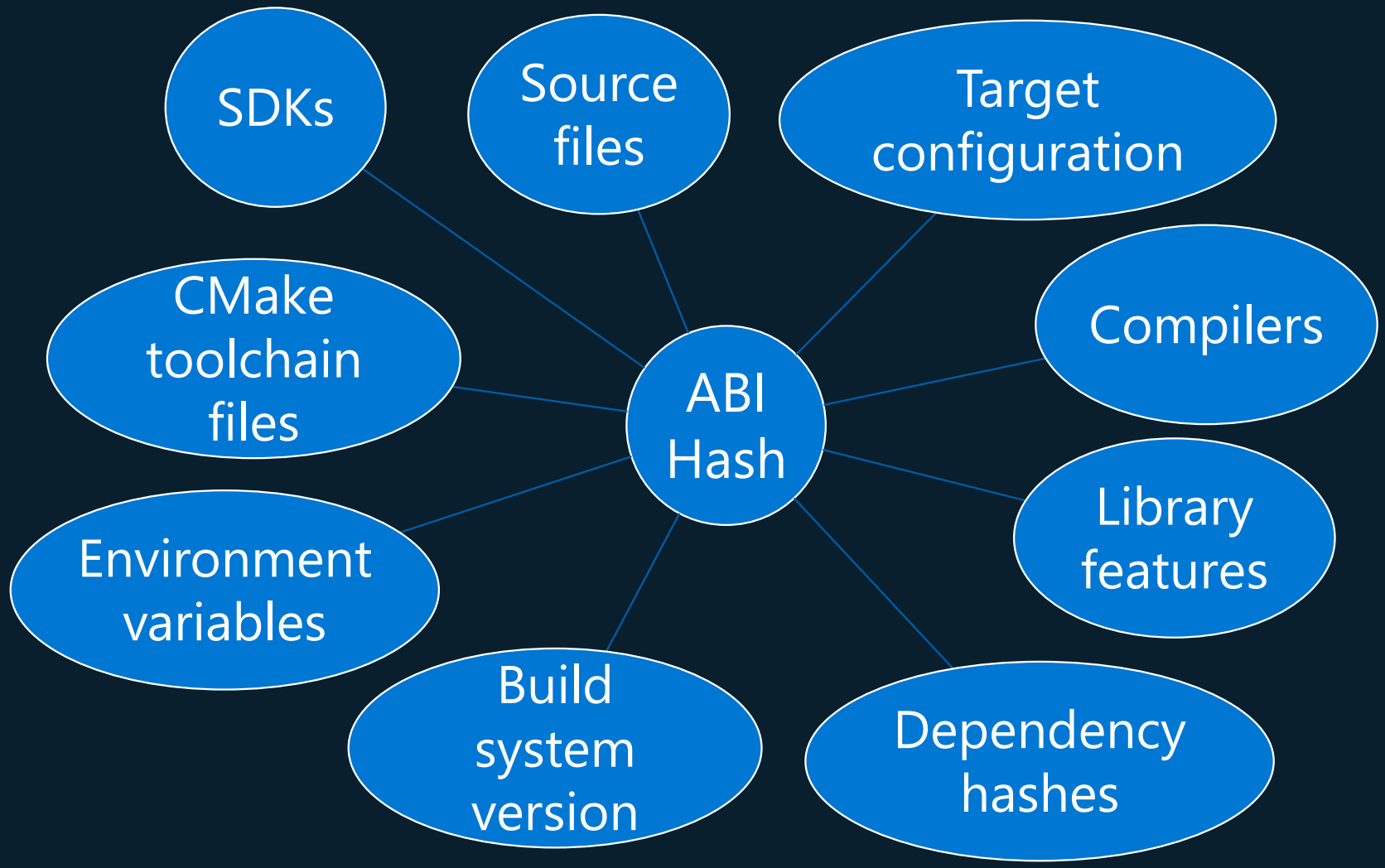


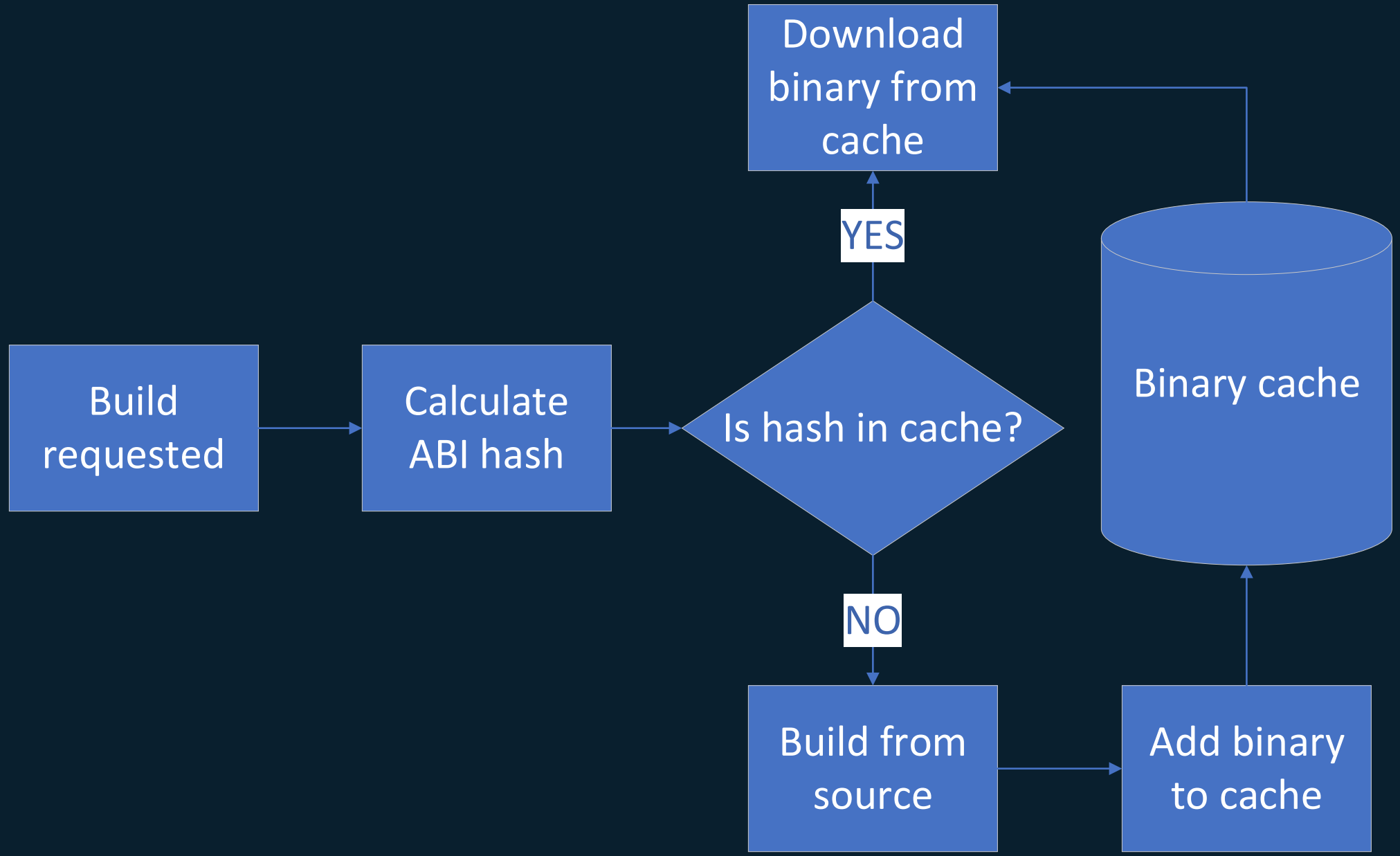
Problem 2: Build times are too long when building from source
Solution 2: Establish a verified binary cache

- Establish a binary cache to store library binaries, shared with all developers and CI builds

- For every build, calculate an **ABI hash** to uniquely identify it.

An ABI hash calculation with binary caching in vcpkg



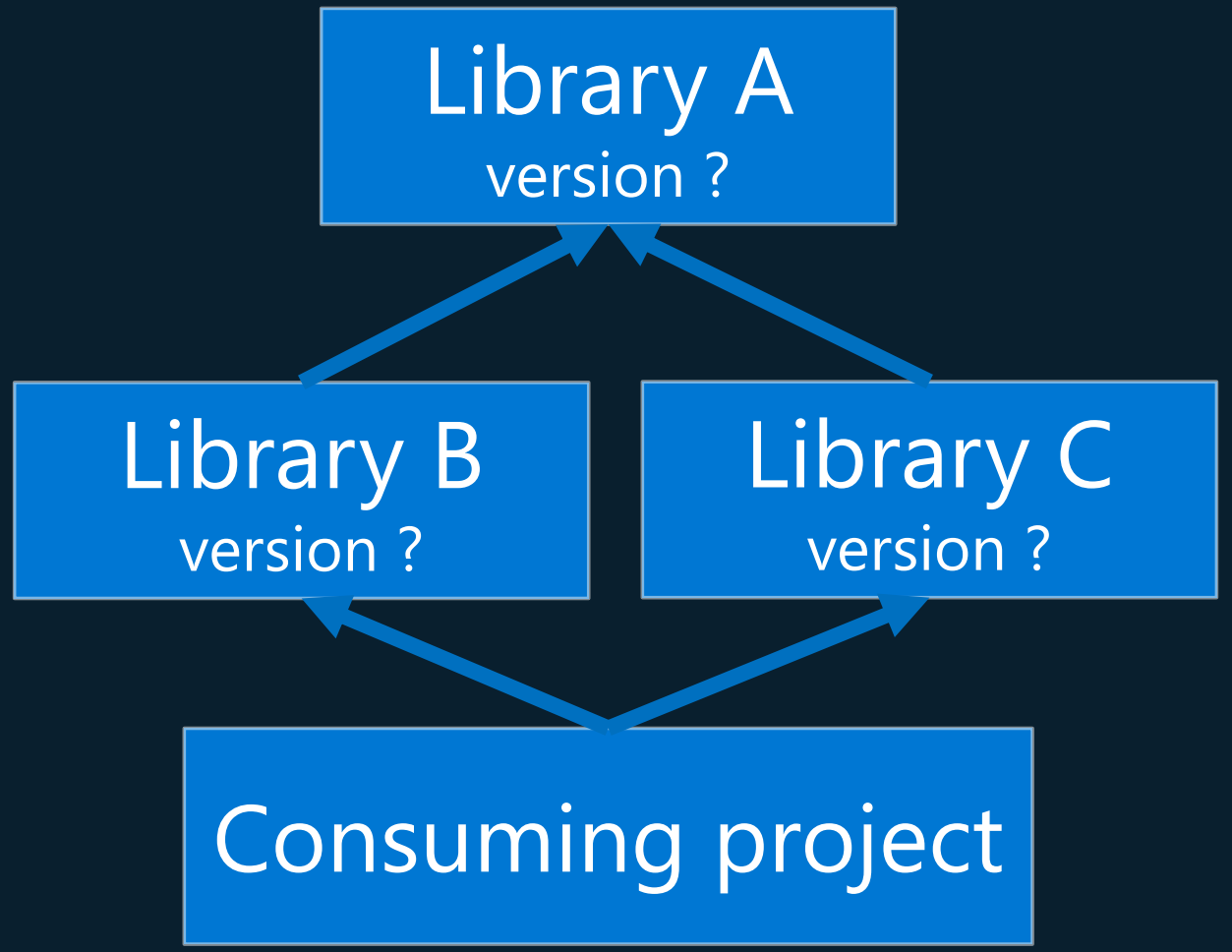

Download binary from cache
YES
Binary cache
Build requested
Calculate ABI hash
Is hash in cache?
NO
Build from source
Add binary to cache

Cataclysm: Dark Days Ahead

Install dependencies with binary caching

Problem 3: Version conflicts – The Diamond problem

- Sometimes multiple dependencies share a common transitive dependency
- But only one copy of that library should exist in the build
- If consumers of the library disagree about its API version, a Diamond problem occurs
- Very common with open-source, but also possible with 1st party libraries at large companies

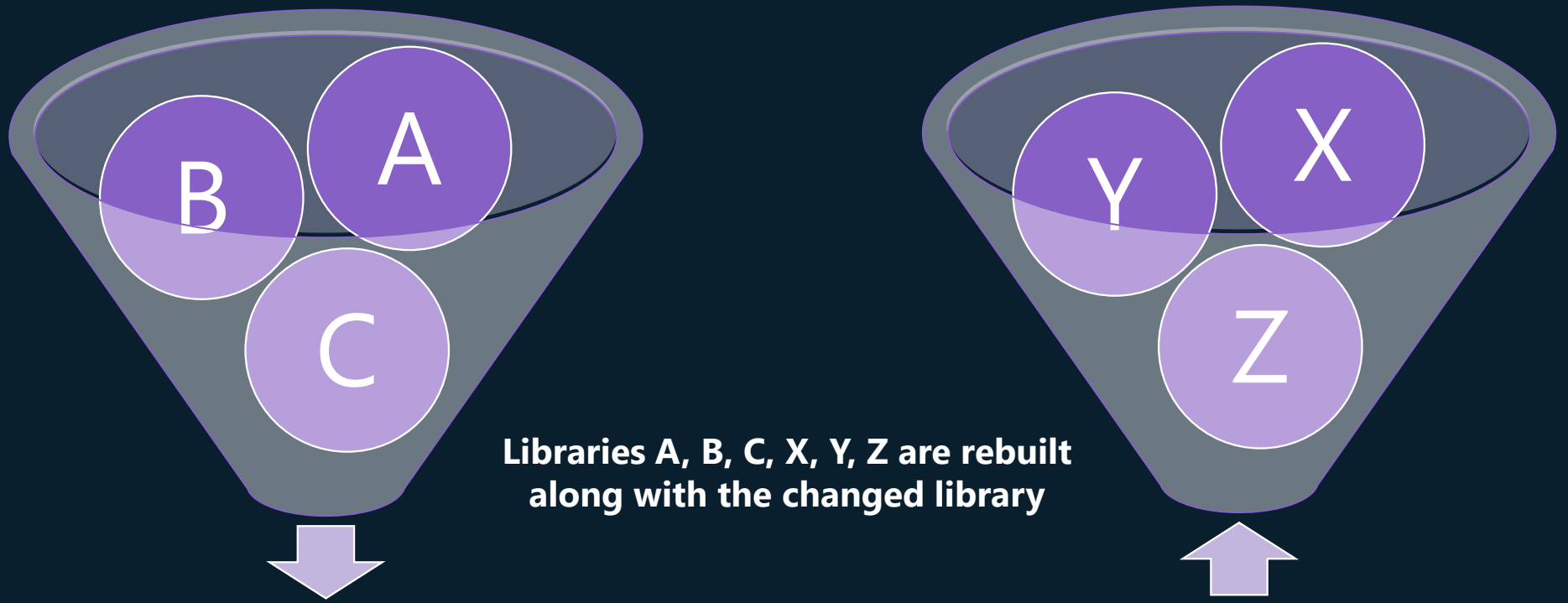


Problem 3: Version conflicts – The Diamond problem
Solution 3: Manage baselines, not individual libraries

Also see this CppCon 2021 talk from Bloomberg

Problem 3: Version conflicts – The Diamond problem
Solution 3: Manage baselines, not individual libraries

Test version updates using the “cones of destruction” method



Cataclysm: Dark Days Ahead

Versioning baselines

Problem 4: Building open-source dependencies is hard

- Trade-off: writing your own libraries is more initial work, but easier to maintain
- Almost every large company I spoke to consumes some open-source, but most consumed packages are not open-source
- Concerns with consuming open-source:
  - Time spent learning how to build and integrate into the consuming project
  - Lots of transitive dependencies; building them all manually is a pain
  - Licensing issues; need to minimize legal risk (covered later in this talk)
  - Fears of potential security vulnerabilities (covered later in this talk)

Problem 4: Building open-source dependencies is hard
Solution 4: Use a package manager with a large open-source catalog

- Best options: vcpkg or Conan (thousands of popular open-source libraries)
- Observation: companies are hesitant to use more open-source, package managers help close some (but not all) of the gaps
- Package managers can build open-source libraries and resolve their transitive dependencies automatically

Problem 5: Organization restricts access to online downloads

- Scenario A: Developers use a completely segregated, offline, air-gapped build environment
- Scenario B: Developers must get approval before downloading open-source code
- Scenario C: Continuity of business – 3rd party servers may go offline or there could be a network outage

Problem 5: Organization restricts access to online downloads
Solution 5: Cache assets needed for builds

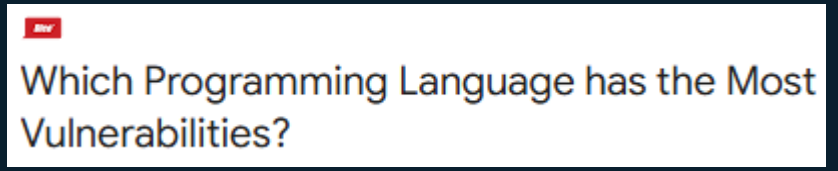
- Companies must maintain their own internally-controlled copies of any software needed to operate their business
  - Download everything you need once outside of your “controlled” environment.
  - Verify that all downloaded assets are valid and compliant with your org’s rules.
  - Move cache to shared location accessible to all build environments by authorized users.

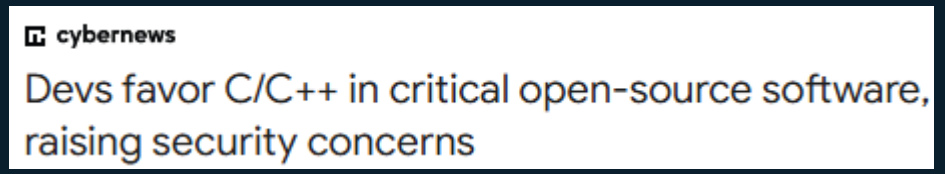
Cataclysm: Dark Days Ahead

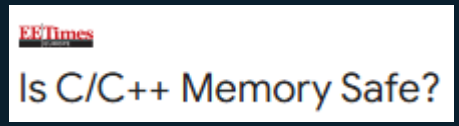
Asset caching

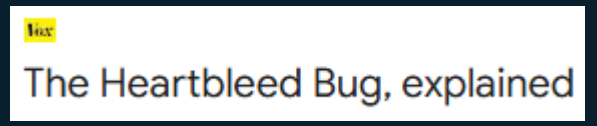
Problem 6: Security vulnerabilities in open-source code

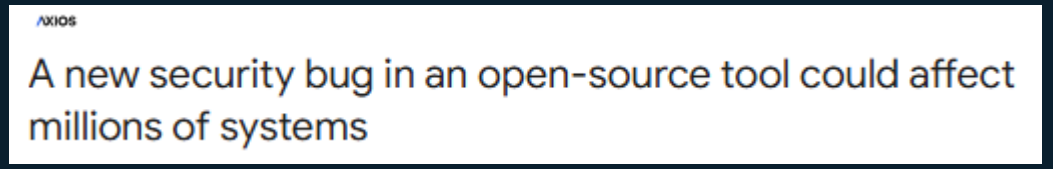
- Introduction of security vulnerabilities is a risk of consuming open-source
  - OpenSSL Heartbleed (2014)
  - OpenSSL email address buffer overflow (2022)
  - xz Utils backdoor (2024)
  - …
- Memory safety issues are common
  - Also see: The Urgent Need for Memory Safety in Software Products | CISA (2023)
- Other CppCon talks on these topics too











Problem 6: Security vulnerabilities in open-source code
Solution 6: Vulnerability monitoring, prevention, and response

- Review public CVE databases (e.g. GitHub Advisory Database)
- Run static analysis + dynamic analysis tools (e.g. Address Sanitizer) on all code, including open-source
- Incorporate fuzzing in some CI runs
- Scan code for references to vulnerable packages
- Produce scripts to scan company code for matches against known open-source code

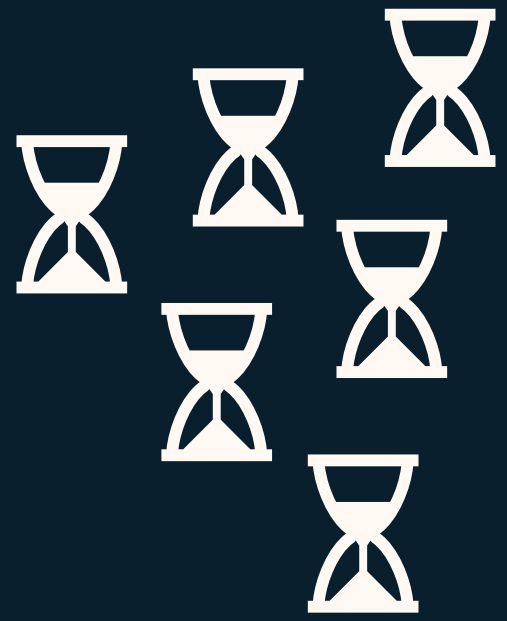
Problem 6: Security vulnerabilities in open-source code
Solution 6: Vulnerability monitoring, prevention, and response

- Make it trivial to update packages so you can respond quickly
  - Track 3rd party code as explicit, separately linked packages
  - Create scripts that automatically generate PRs to update vulnerable packages
  - Require all devs to use a consistent dependency acquisition experience

- Block vulnerable package versions from being accessible

Problem 7: Duplicated engineering cost to maintain dependencies

- As organizations grow, it is harder to share and maintain dependencies
- Communication breakdowns, inconsistent processes
- Many teams doing similar work:
  - Producing and distributing libraries
  - Establishing compliance processes for dependencies
  - Writing documentation for their internal customers
  - Cross-organizational ABI instability and version conflicts

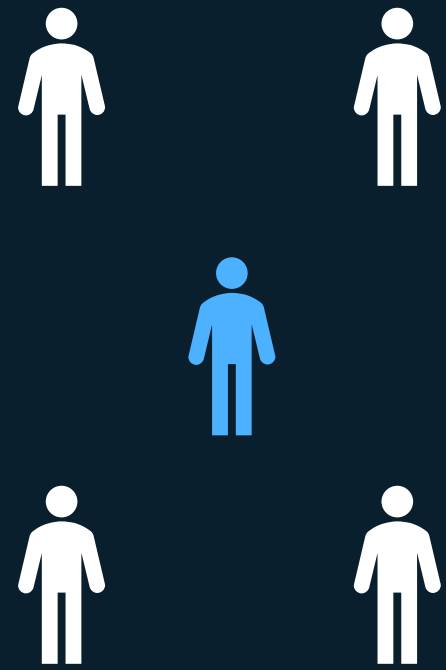


Problem 7: Duplicated engineering cost to maintain dependencies
Solution 7: Centralize dependency management processes

- Companies are centralizing common dependency management tasks
- Maintaining 3rd party packages and providing support for 1st party ones
- Approving new open-source dependencies and validating compliance
- Maintaining a common library distribution system or package manager

Problem 7: Duplicated engineering cost to maintain dependencies
Solution 7: Centralize dependency management processes

- Enforce consistency across the organization:



- Common package format, storage, and distribution
- Authentication system to access/modify packages
- Verify ABI compatibility and establish versioning baselines
- Global documentation for dealing with dependencies
- Package consumers have one consistent way of consuming C++ dependencies

Problem 8: Difficult to track or report on all dependencies

- Large companies have many projects. Large projects have many dependencies. Can they track them all?
- Scenario A: Look up if your product has dependencies with known licensing or security risks.
- Scenario B: Important customer (e.g. US government) requires transparency about everything that goes into your software (like the ingredients list on a product at the grocery store)

Problem 8: Difficult to track or report on all dependencies
Solution 8: Produce a Software Bill of Materials (SBOM)

- Organize and list 3rd party dependencies as individual, named packages

- Produce Software Bill of Materials (SBOMs)

- Two common formats: SPDX and CycloneDX

A Software Bill of Materials (SBOM) is a formal record containing the details and supply chain relationships of various components used in building software. These components, including libraries and modules, can be open source or proprietary, free or paid, and the data can be widely available or access-restricted.

Source: SBOM FAQ (Cybersecurity and Infrastructure Security Agency)

Benefits of SBOMs accrue to both software suppliers and consumers — and are similar for both. They include:

- Identifying and avoiding known vulnerabilities
- Quantifying and managing licenses
- Identifying both security and license compliance requirements
- Enabling quantification of the risks inherent in a software package
- Managing mitigations for vulnerabilities (including patching and compensating controls for new vulnerabilities)
- Lower operating costs due to improved efficiencies and reduced unplanned and unscheduled work.

Source: SBOM FAQ (Cybersecurity and Infrastructure Security Agency)

Example spdx.json file

Problem 9: Build toolchain variations across the org

- Examples:

- Two teams contributing to a common application build with different versions of a compiler
- Developer tests a change locally, forgets target architecture is set to x86, and is surprised when the build later fails in CI on x64
- Developer forgets to turn on Address Sanitizer flag during testing
- A binary built with Debug configuration is deployed to production

Problem 9: Build toolchain variations across the org
Solution 9: Establish a global toolchain and build in containers

- Establish company-wide policy for build tools / versions
- Consider restricting access to other tools by distributing allowed tools from a trusted, internal location
- Start building in containers pre-configured for different environments (e.g. testing, production)
- Update entire organization at once to a new tool version (e.g. package manager, compiler toolset)
- Allow time for testing before updating tools

Problem 10: Moving to new solution is complex or too time consuming

- Changing how you manage your libraries is hard
- Especially for large organization with many projects / teams
- Fear of breaking builds and having to clean up the mess
- Timeline can feel extreme and unrealistic
- Or can be difficult to cost migration

Problem 10: Moving to new solution is complex or too time consuming
Solution 10: Establish incremental milestones with success at each step

- Strategy varies by company – general guidance

- Break down into smaller milestones, where you gain some benefit at each step, even if you're not all the way done yet

- Cost each milestone based on available capacity

Problem 10: Moving to new solution is complex or too time consuming
Solution 10: Establish incremental milestones with success at each step

Example: moving from checked-in + manually linked libs to package manager

Step	Action	Benefit	Cost
1	Develop tool to scan company repos/code for open-source	Identify legal & security risks	X months
2	Choose a project, migrate open-source code (proof of concept)	Project benefits from the new system; owning team can share learnings	X months
3	Start moving other projects' open-source to package manager	More teams gain benefits	X months
4	Centralize package storage, distribution, documentation	Reduce time spent by individual dev teams	X months
5	Start creating packages out of internal libraries	More libraries benefit from package manager features	X months
6	Require all libraries to move to package manager	Enforce consistency, easier maintenance	X months

Summary – 10 conclusions

1. Build C++ dependencies from source
2. Establish binary caching where possible
3. Organize dependencies + versions into baselines (fixed points in time)
4. Use an open-source package manager to save time / effort
5. Create an asset cache for sources needed to build dependencies
6. Develop a vulnerability monitoring, prevention, and response strategy and associated tools / workflows
7. Centralize common dependency management tasks, enforce consistency across organization at scale
8. Organize dependencies into coherent packages and start producing SBOMs
9. Establish a global toolchain policy and build in containers if possible
10. Break large migrations down into smaller milestones with a win at each step

Call to action: share your story!

- Do you work at a large company? How do you manage your dependencies? Have you developed other strategies to overcome some of these problems? Or do you have some other problems that weren't covered here?

- The community can benefit from your experience! Conference talks, blog posts, /r/cpp Reddit threads, etc.
  - Also, if you are a vcpkg user with an interesting story, we are publishing blog posts on the Microsoft C++ Team Blog ☺

- And feel free to chat with me at the Microsoft booth this week too!

Microsoft @ CppCon

- Check out other Microsoft sessions at CppCon this year!
  - Monday @ 14:00: So You Think You Can Hash – Victor Ciura
  - Thursday @15:15: What's New for Visual Studio Code: Performance, GitHub Copilot, and CMake Enhancements – Alexandra Kemper & Sinem Akinci
  - Friday @13:30: "What's New in Visual Studio for C++ Developers" – Mryam Girmay & Michael Price

Microsoft @ CppCon

Come by our booth in the main hallway, across from the stairs

Join #visual_studio channel on CppCon Discord https://aka.ms/cppcon/discord

- Meet the Microsoft C++ team
- Ask any questions
- Discuss the latest announcements

Take our survey

Win prizes

https://aka.ms/cppcon/dependencies