Standard C++ toolset

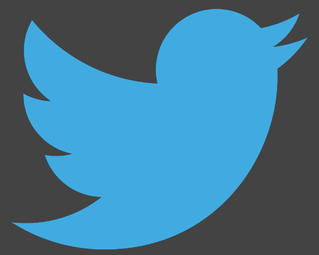
Anastasia Kazakova, JetBrains
@anastasiak2512

Meeting C++ 2022

About me
–

C++: Embedded,
Telecom, 4G/LTE

C++ Tools PMM



@anastasiak2512

What is language toolset?

–

Essential tools

- Compiler + standard library
- Project model / Build system
- Dependency management
- Debugger

Complementary

- Analyzer
- Unit testing frameworks and other tools

Java

1. javac, ECJ
2. Java Class Library: concurrency, networking, AWT / Swing, generics, internationalization
3. Maven (67-72%), Gradle (20-49%), Ant (8-11%)
4. JUnit (85%)

Ruby

1. “Compiler”:
   a. Interpreter: MRI (CRuby), YARV
   b. JIT: JRuby (JVM), Rubinius (LLVM)
   c. IronRuby (.NET), new mruby interpreter, …
2. RubyGems (Bundler 90%)
3. RSpec (77%)

Swift

—

1. Swift toolchain from Apple
2. Swift compiler (LLVM)
3. SourceKit (LSP)
4. LLDB, Read Eval Print Loop (REPL), Playground
5. SPM, CocoaPods
6. XCTest (87%)

Rust

1. rustc
2. Cargo build + packages
3. Rustfmt
4. rust-analyzer (LSP)
5. 60% debug via `println!` or `dbg!` macros
6. test module / cargo test

Does the toolset help?

–

1. Quicker start
2. Easier development environments maintenance
3. More code unification
4. Easier onboarding and code maintenance
5. Better adoption of good practices like unit testing
6. Less issues with packages versioning

Conflicts and inconsistencies!

Example: syntax style inconsistencies in C++

—

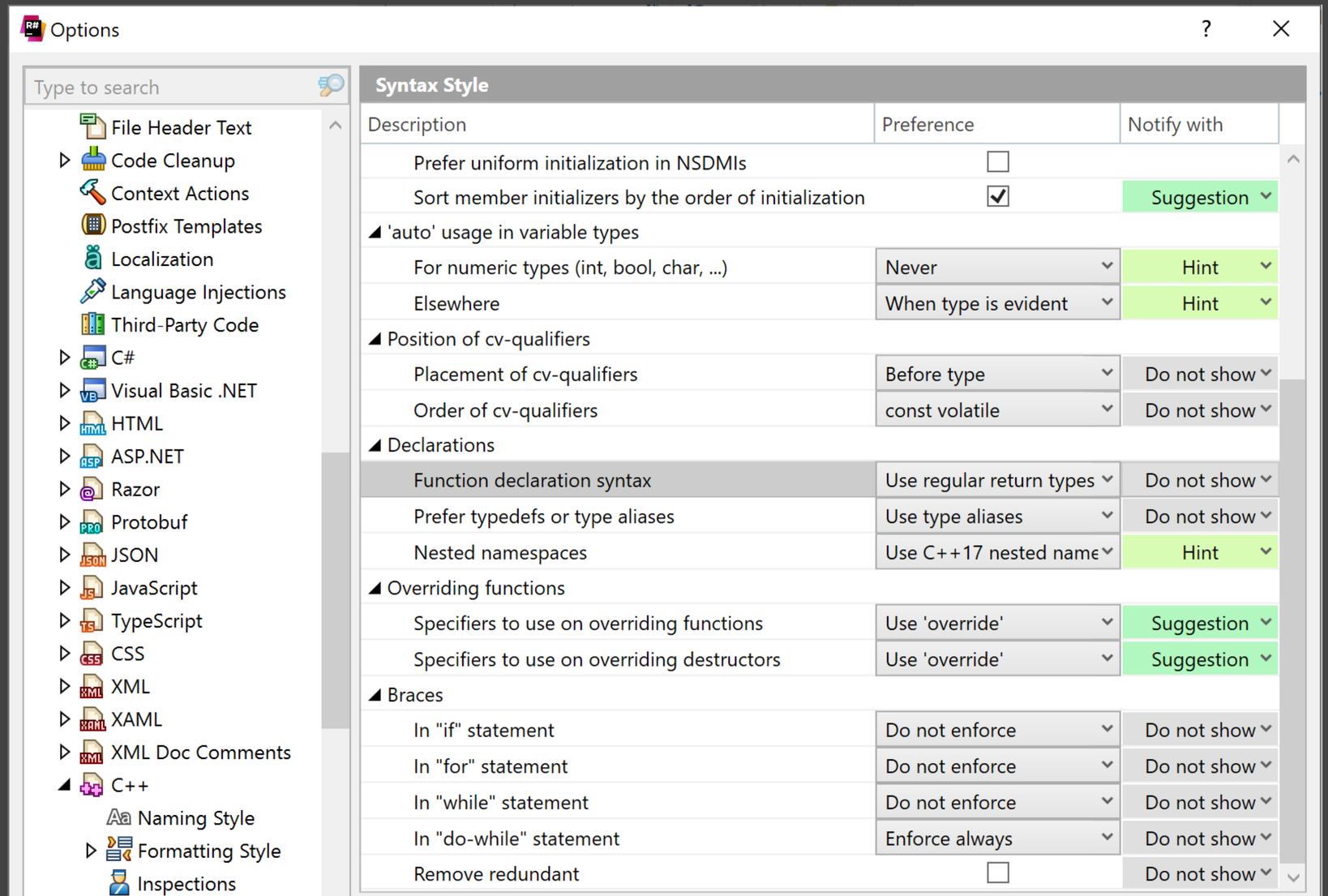
Almost Always Auto or when type is evident or never for numeric types.

Const before or after the type it applies to.

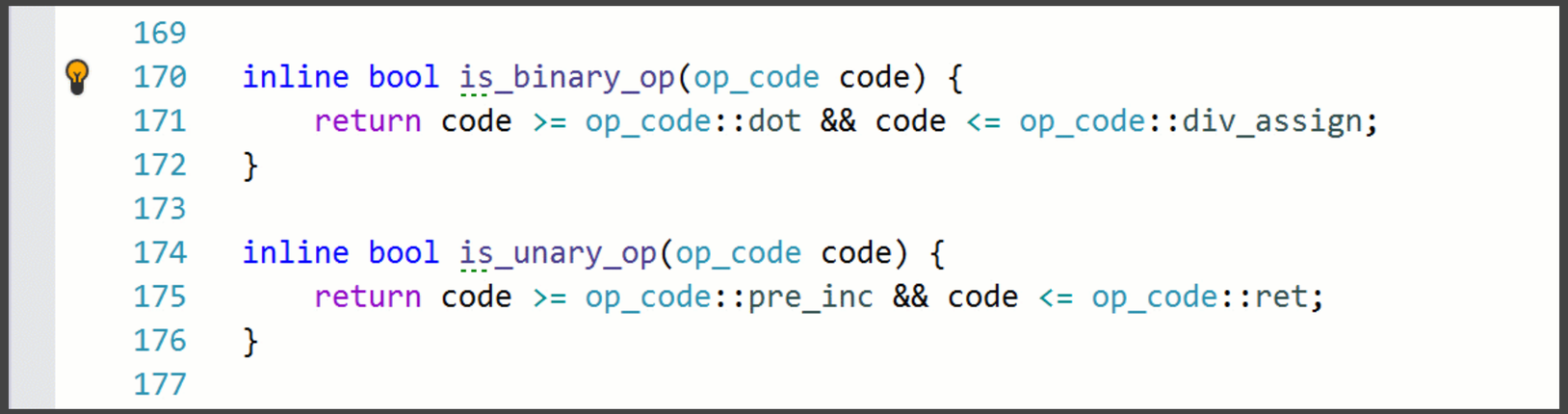
Trailing return type for lambdas or always.

Virtual explicitly (UE) or override/final and no virtual (C++ Core Guidelines).

Example: syntax style inconsistencies in C++ –



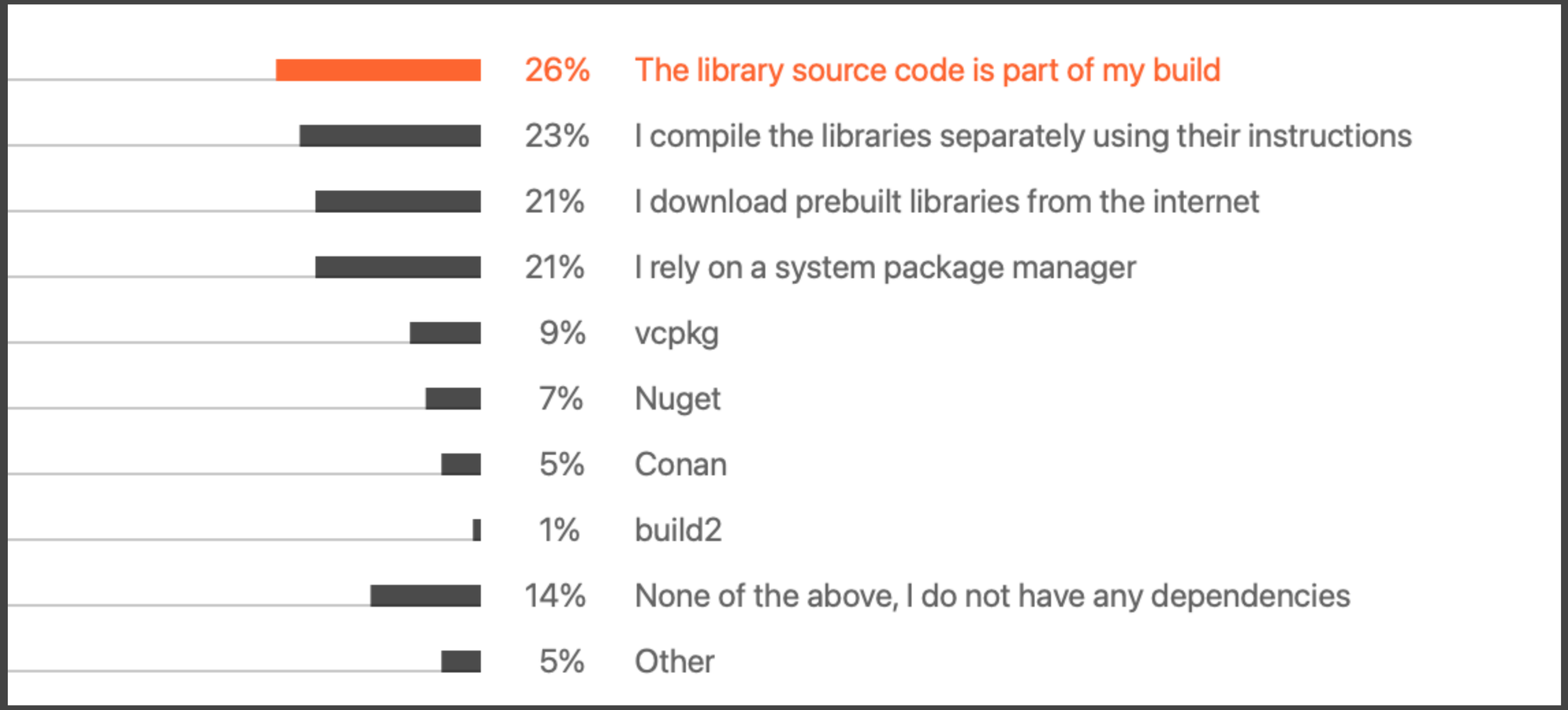
Example: syntax style inconsistencies in C++ –



```
169
170  inline bool is_binary_op(op_code code) {
171      return code >= op_code::dot && code <= op_code::div_assign;
172  }
173
174  inline bool is_unary_op(op_code code) {
175      return code >= op_code::pre_inc && code <= op_code::ret;
176  }
177
```

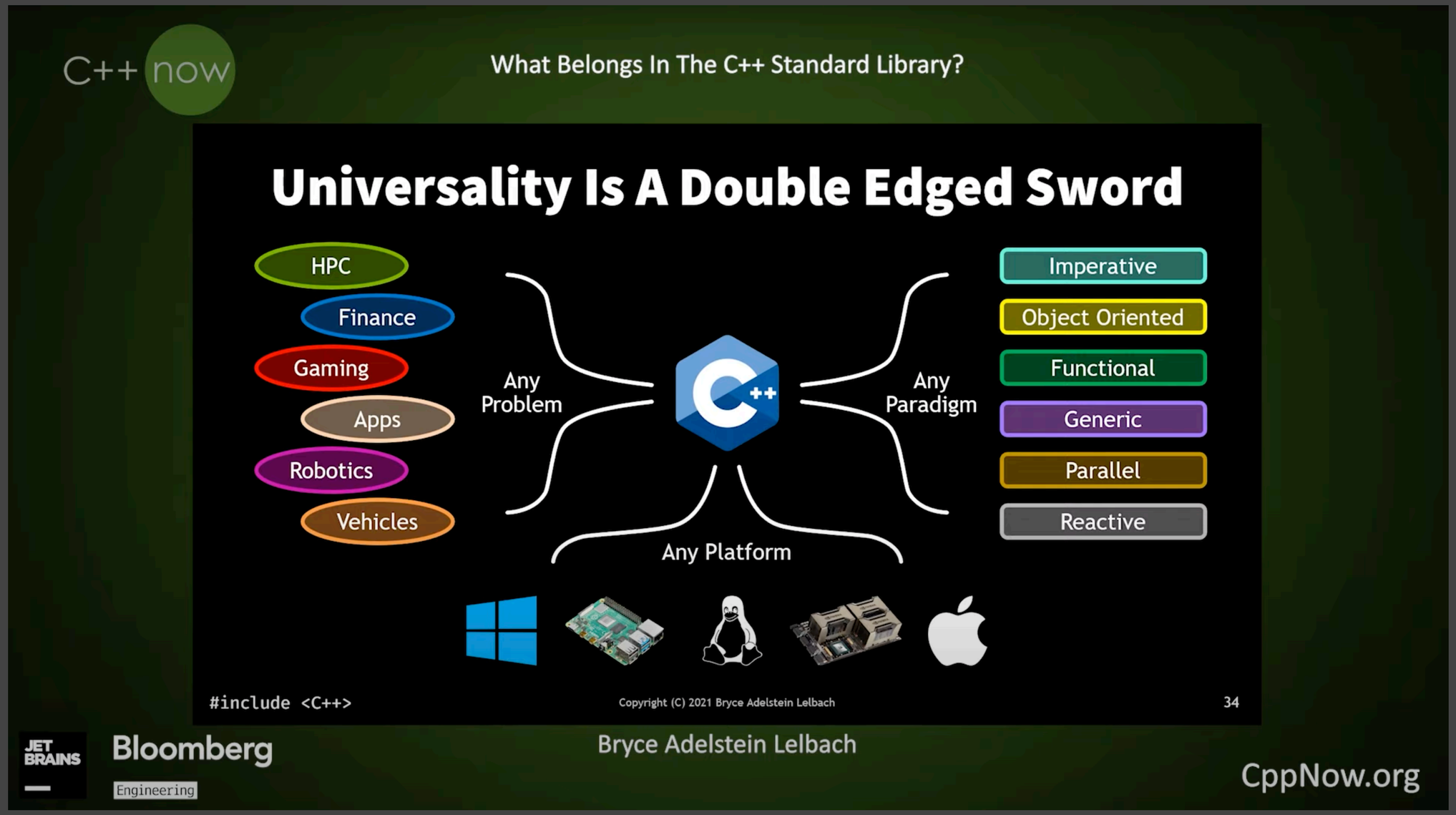
Example: package management struggle

How do you manage your third party libraries in C++?

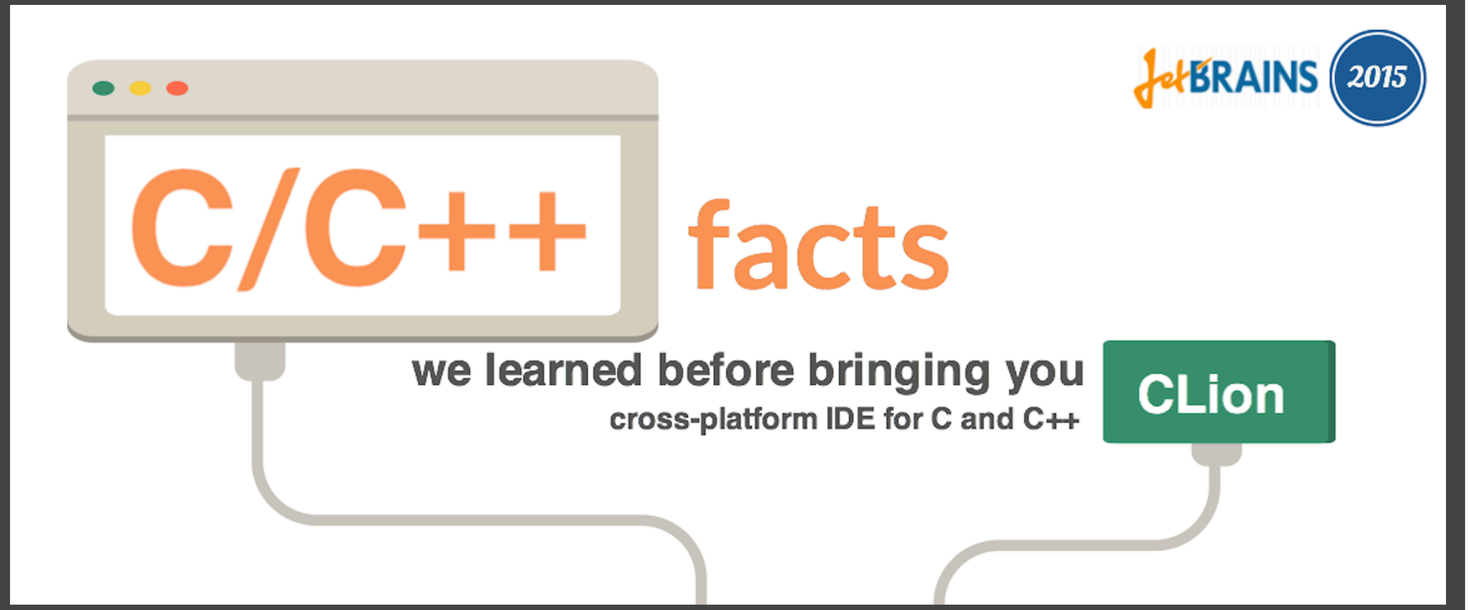


C++ toolset. Where are we now?

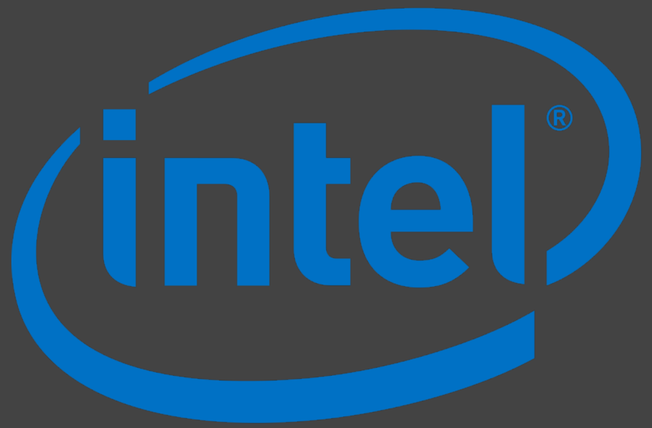
C++ toolset. Where are we now?



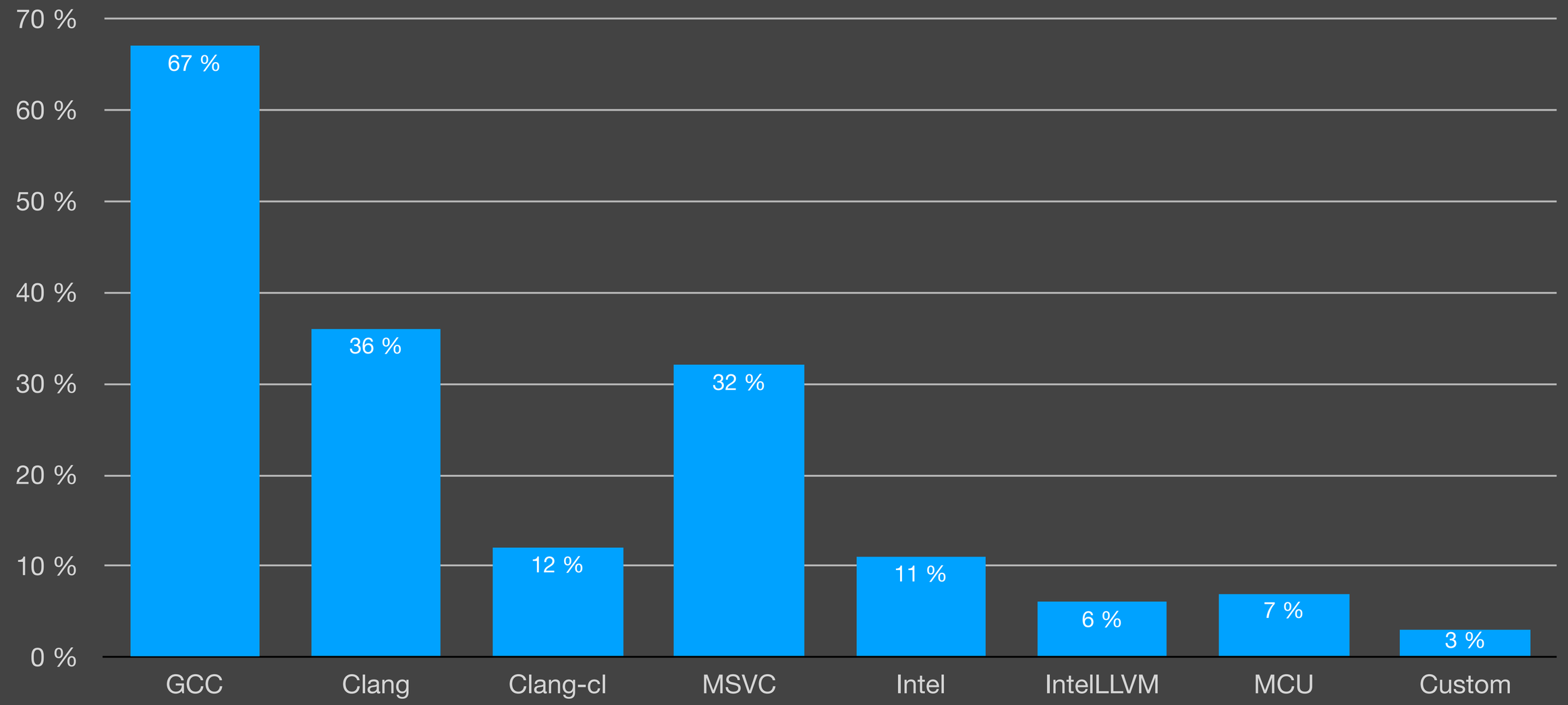
C++ toolset. Back in 2015
–



C++ compilers
–



C++ compilers: what do developers use?



Compatibility: compiler options

—

```
if (MSVC)
    add_compile_options(/W4 /WX)
else()
   add_compile_options(-Wall -Wextra -Werror)
endif()
```

Compatibility: C++ standard support

–

C++20 core language features

C++20 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	IBM XLC++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc	[Collapse]
Allow lambda-capture `[=, this]`	P0409R2	8	6	19.22*	10.0.0*	5.1	2021.1					20.7		
`__VA_OPT__`	P0306R4 P1042R1	8 (partial)* 10 (partial)* 12	9	19.25*	11.0.3*	5.1	2021.1					20.7		
Designated initializers	P0329R4	4.7 (partial)* 8	3.0 (partial)* 10	19.21*	12.0.0*	5.1	2021.1					20.7		
template-parameter-list for generic lambdas	P0428R2	8	9	19.22*	11.0.0*	5.1	2021.1					20.7		
Default member initializers for bit-fields	P0683R1	8	6	19.25*	10.0.0*	5.1	2021.1					20.7		
Initializer list constructors in class template argument deduction	P0702R1	8	6	19.14*	Yes	5.0	2021.1					20.7		
const&-qualified pointers to members	P0704R1	8	6	19.0 (2015)*	10.0.0*	5.1	2021.1					20.7		
Concepts	P0734R0	6 (TS only) 10	10	19.23* (partial)* 19.30*	12.0.0* (partial)	6.1	2021.5					20.11		
Lambdas in unevaluated contexts	P0315R4	9	13 (partial)* 14 (partial)*	19.28 (16.8)*	13.1.6* (partial)									
Three-way comparison operator	P0515R3	10	8 (partial) 10	19.20*		5.1	2021.1					20.7		
DR: Simplifying implicit lambda capture	P0588R1	8		19.24*		5.1	2021.1					20.7		

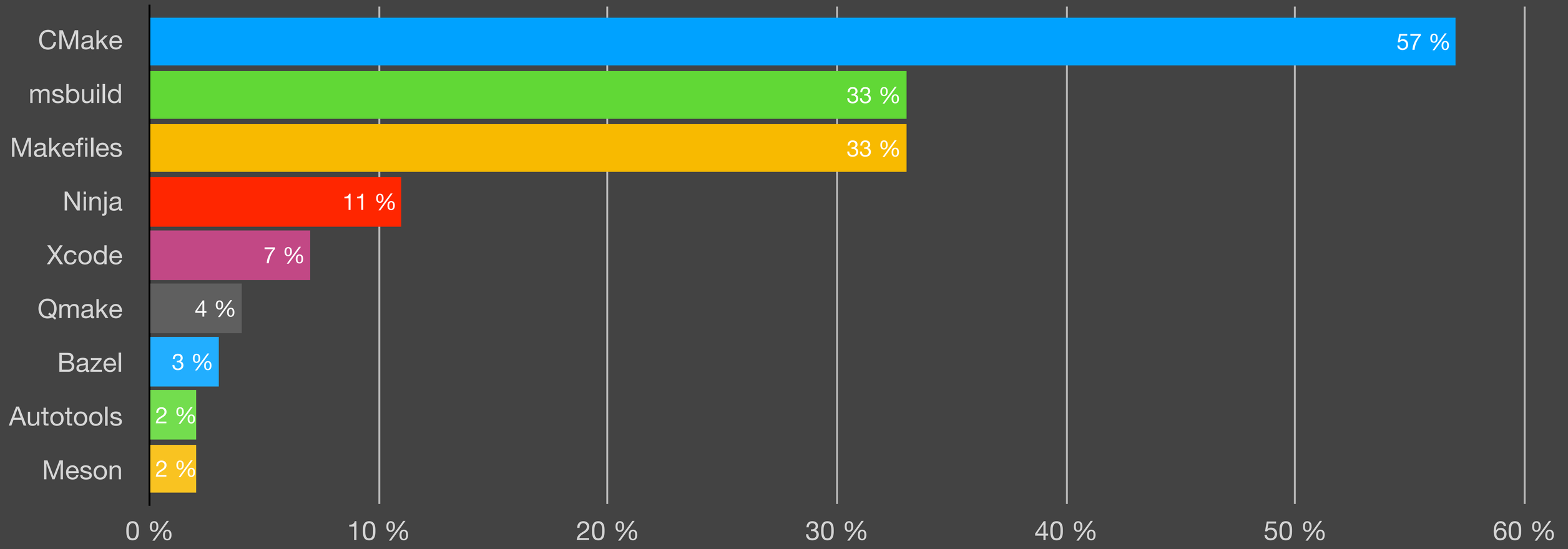
Clang - beyond the compiler

-

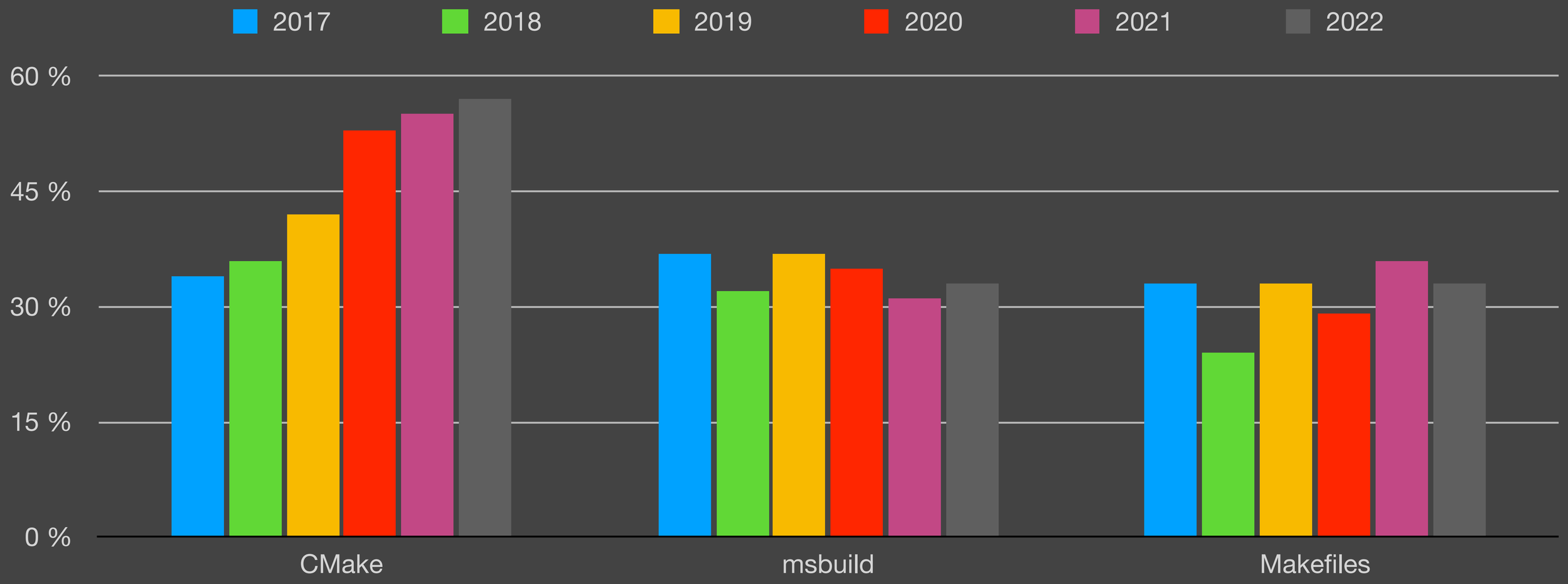
1. Clang, libclang, clangd, LSP
2. Clang Analyzer, Clang-Tidy
3. ClangFormat
4. IDEs
5. Every tool that needs AST

Clang - basis for C++ tools

Build system / project model



Build system / project model



Build system / project model

—

You wanna standard C++ Build System?

You got One! It’s called CMake!

CMake adoption

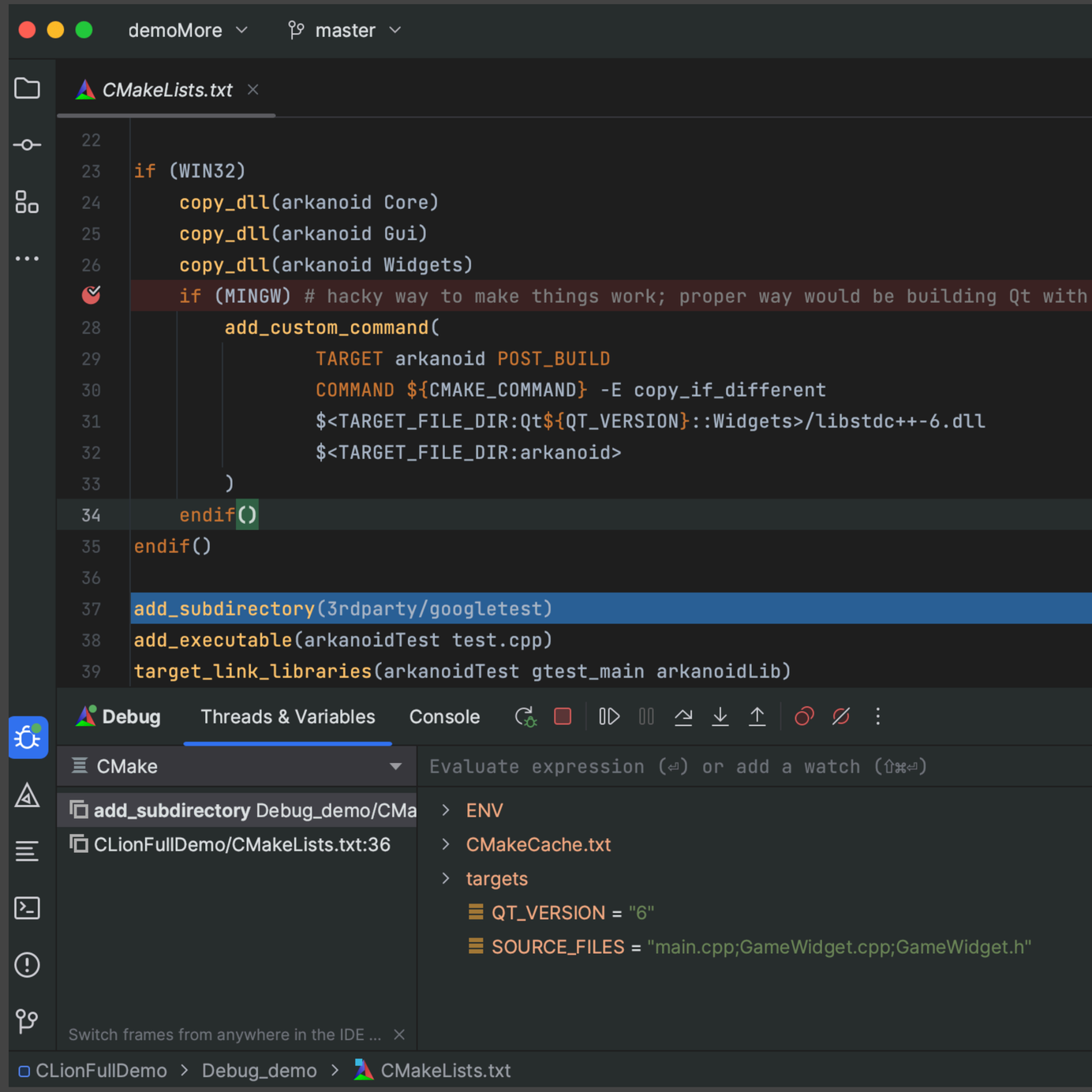
–

1. Libraries, package managers, IDEs
2. Qt6: qmake → CMake
3. Embedded: Zephyr RTOS, Nordic’s nRF5 SDK
4. CMake-buildable Boost effort
5. CMake File API
6. CMake Presets

CMake as a language

—

1. Good for quizzes
   - CMake + Conan: 3 Years Later - Mateusz Pusz, CppNow 2021
2. Needs a debugger



CMake and C++20 modules

1. CMake + VS generator

```
add_executable(ModuleSample a.ixx main.cpp)
```

CMake and C++20 modules
–

2. CMake + GCC/Clang compiler flags

```
function(add_module name)
    file(MAKE_DIRECTORY ${PREBUILT_MODULE_PATH})
    add_custom_target(${name}.pcm
            COMMAND
            ${CMAKE_CXX_COMPILER}
            -std=gnu++20
            -x c++
            -fmodules
            -c
            ${CMAKE_CURRENT_SOURCE_DIR}/${ARGN}
            -Xclang -emit-module-interface
            -o ${PREBUILT_MODULE_PATH}/${name}.pcm
            )
endfunction()
```

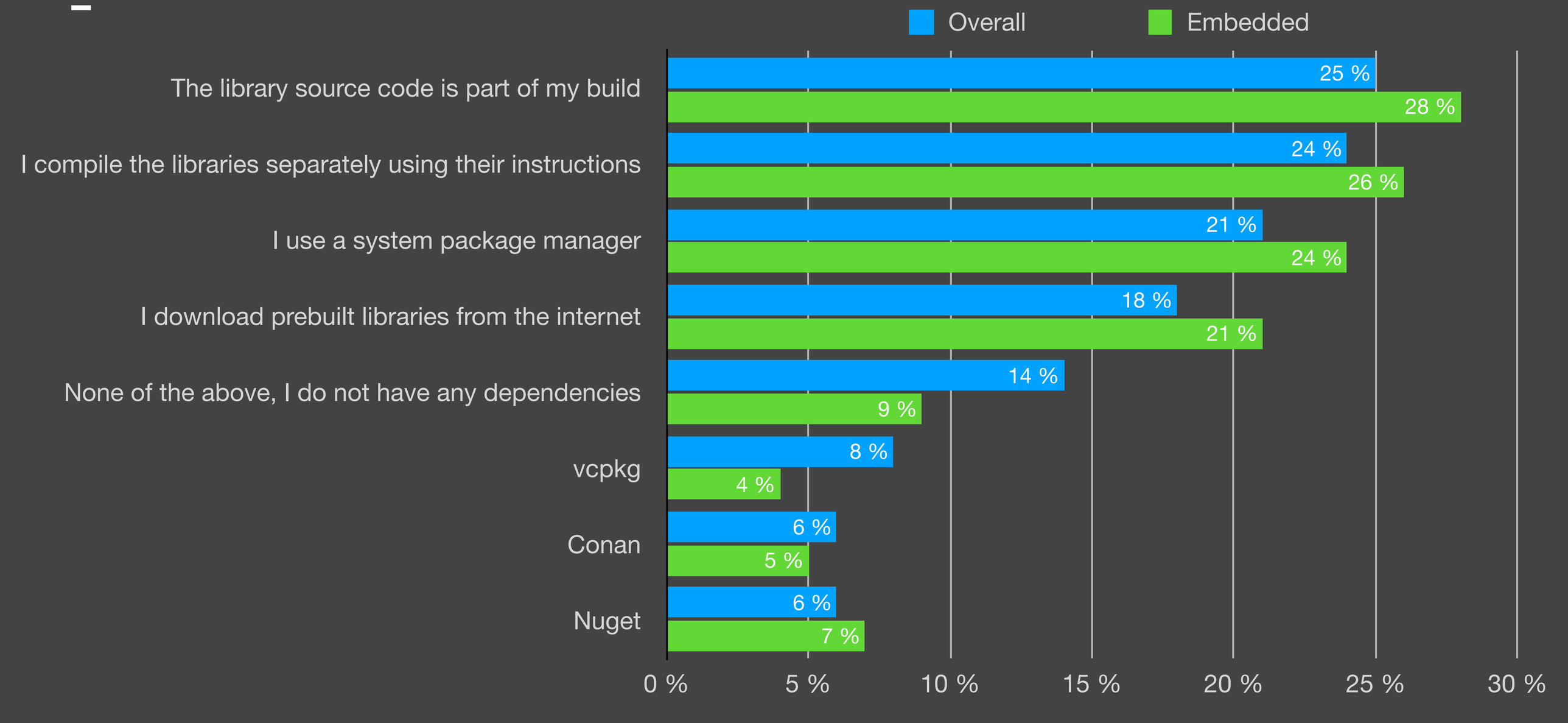
CMake and C++20 modules
–

3. Natively in CMake 3.25 (CMake examples)

```
add_executable(simple)
target_sources(simple
        PRIVATE
          main.cxx
        PRIVATE
         FILE_SET CXX_MODULES
          BASE_DIRS
            "${CMAKE_CURRENT_SOURCE_DIR}"
          FILES
            importable.cxx)
target_compile_features(simple PUBLIC cxx_std_20)

add_test(NAME simple COMMAND simple)
```

Dependency managers

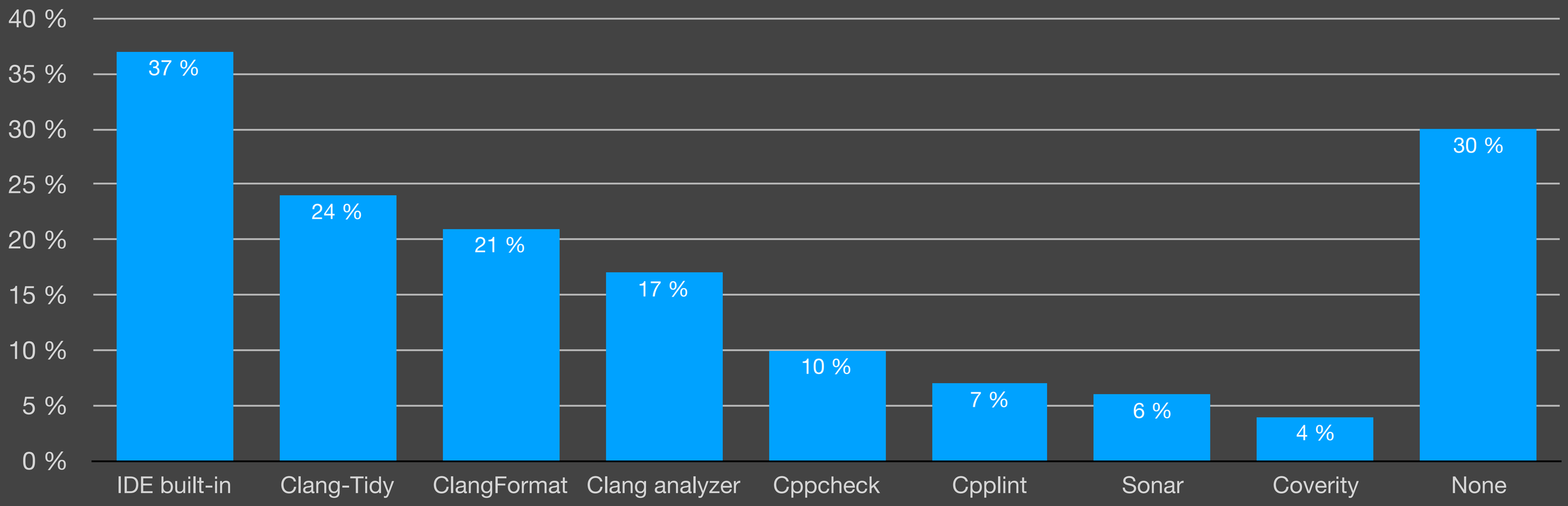


ClangFormat

-

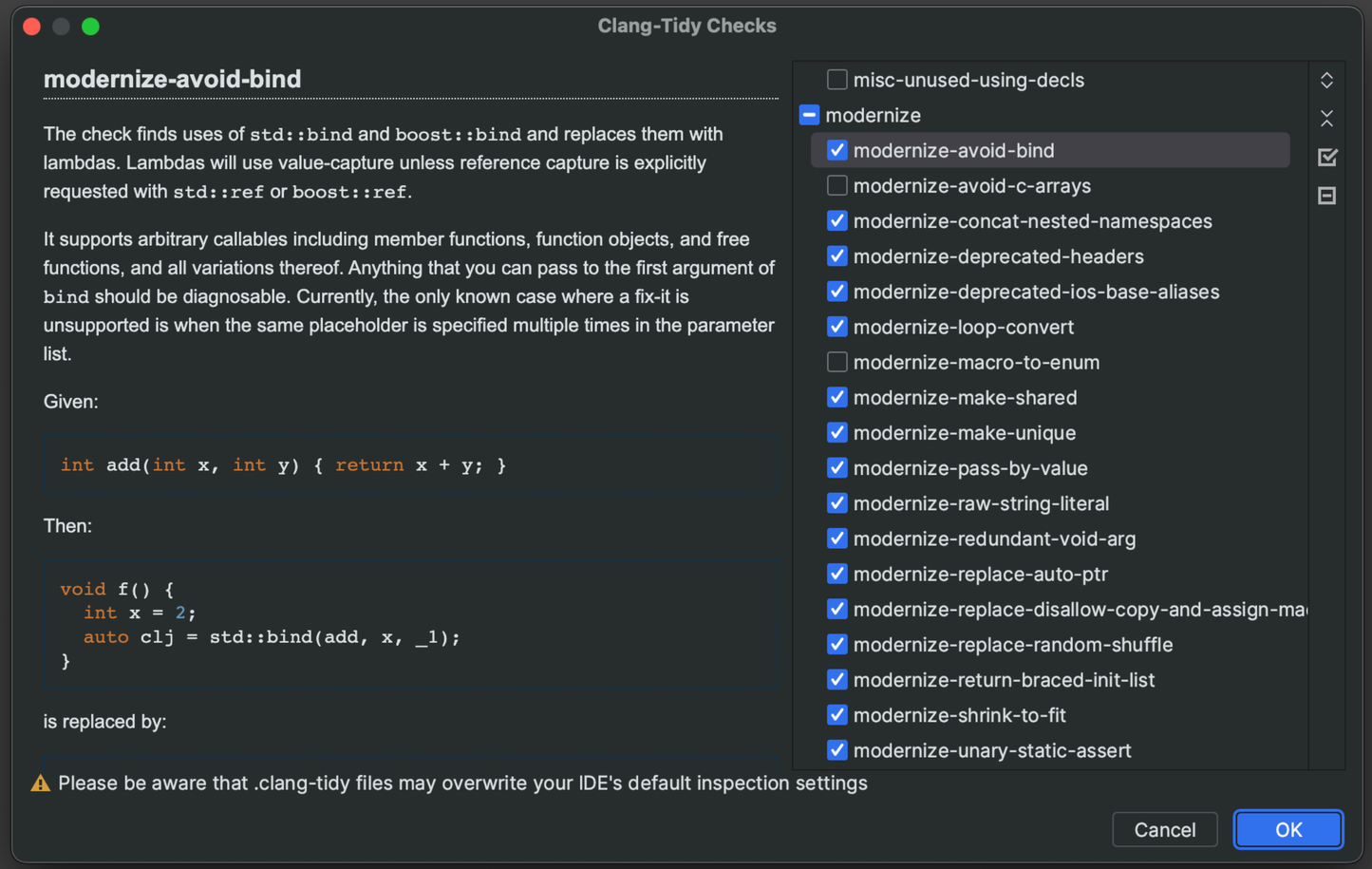
1. Standard tool
2. Fuzzy parser
3. Breaking compatibility between versions

Code Analysis



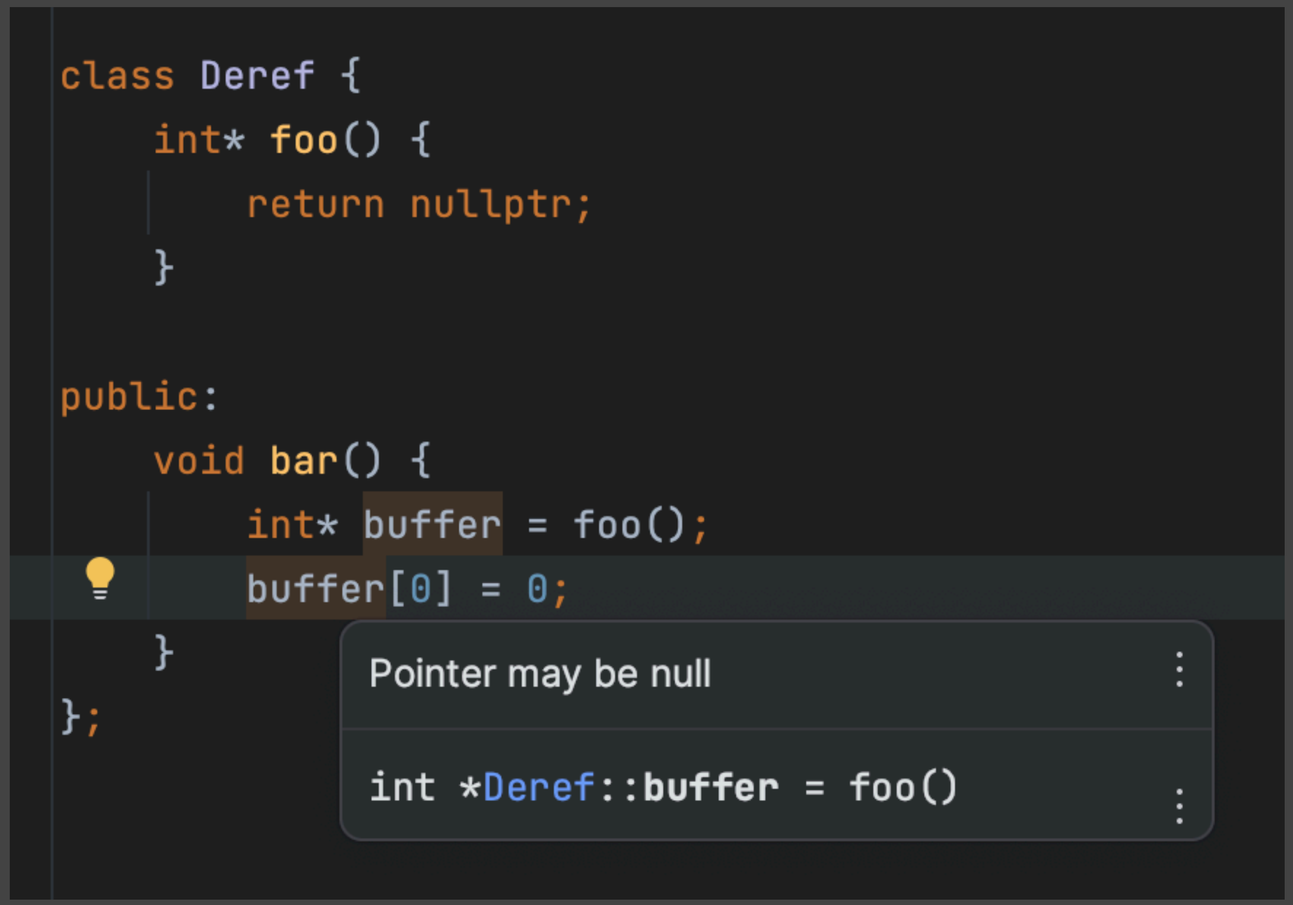
Clang-Tidy

1. Baseline
2. Modernize checks, C++ Core Guidelines checks, Google checks, etc.
3. Custom checks



Data Flow Analysis

1. Analyzes possible values
2. Function scope → TU → CTU
3. Dangling pointer, lifetimes, index out of bounds, unreachable code, endless loops, etc.

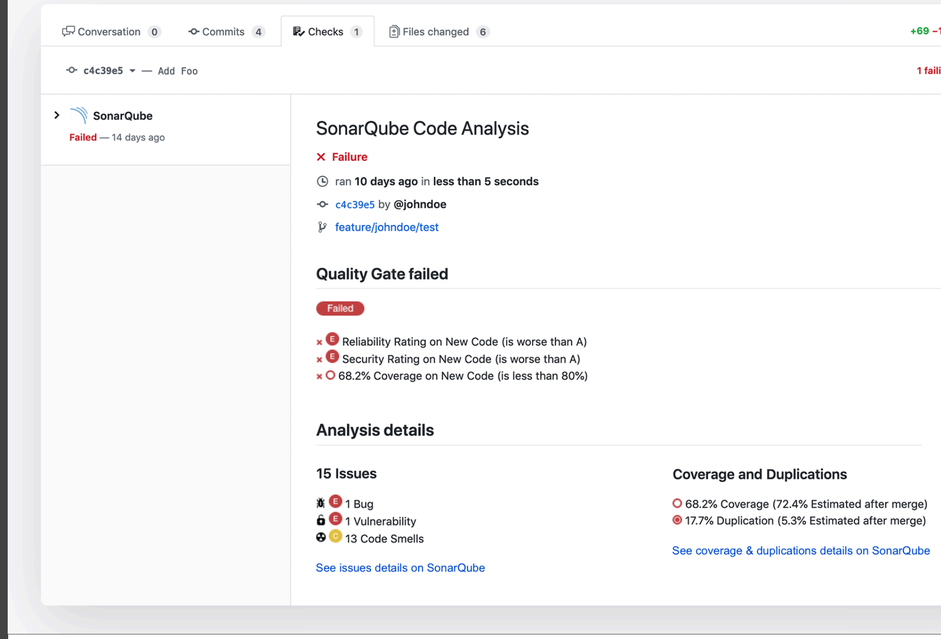


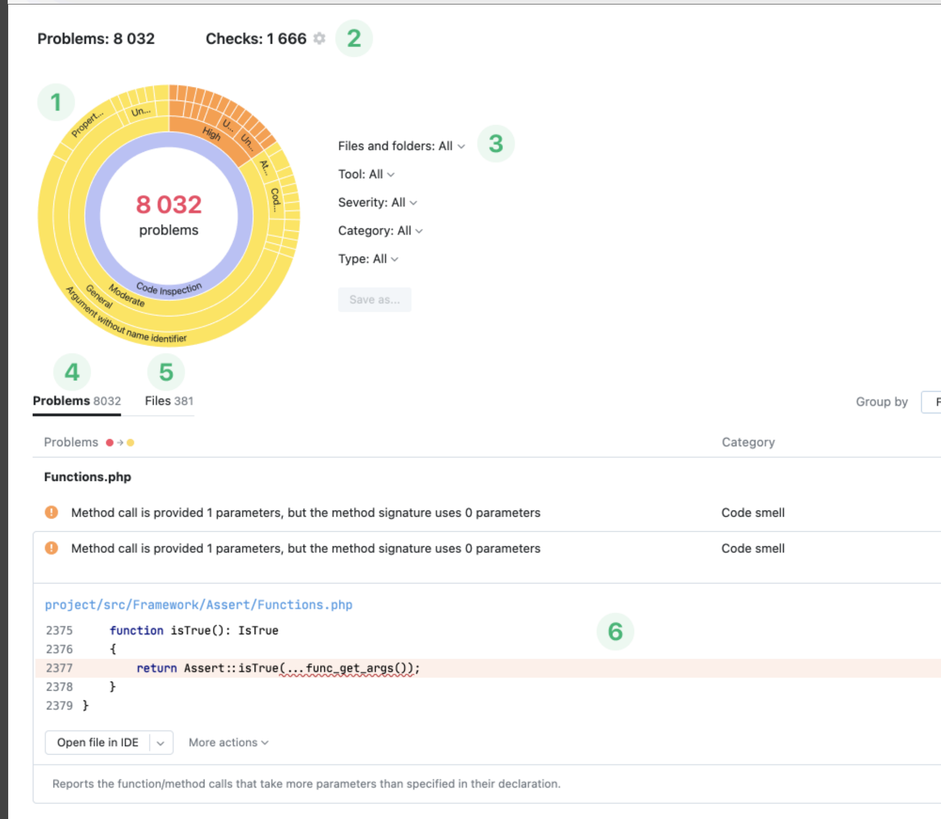
DSL analysis

1. Clazy for Qt
2. Unreal Header Tool for UE
3. Embedded checkers

Code Analysis on CI

1. For code reviews and pull requests
2. Target/platform-specific checks
3. Long/resource-greedy checks
4. Health check, timeline
5. Management tool
6. Good for open-source projects
7. SonarSource tools, JetBrains Qodana





Code Analysis++

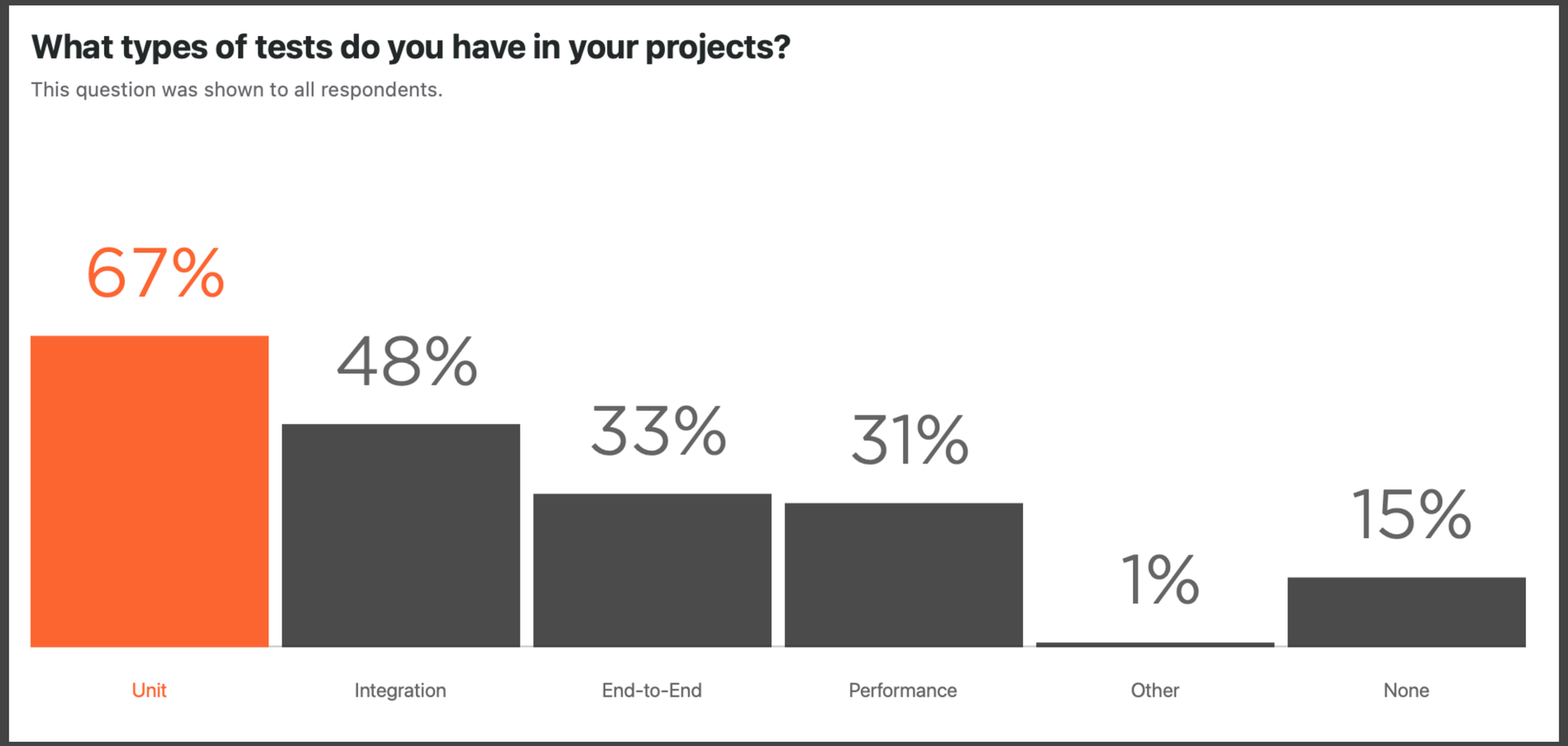
–

Find my talk from C++Now 2019 or NDC TechTown 2022

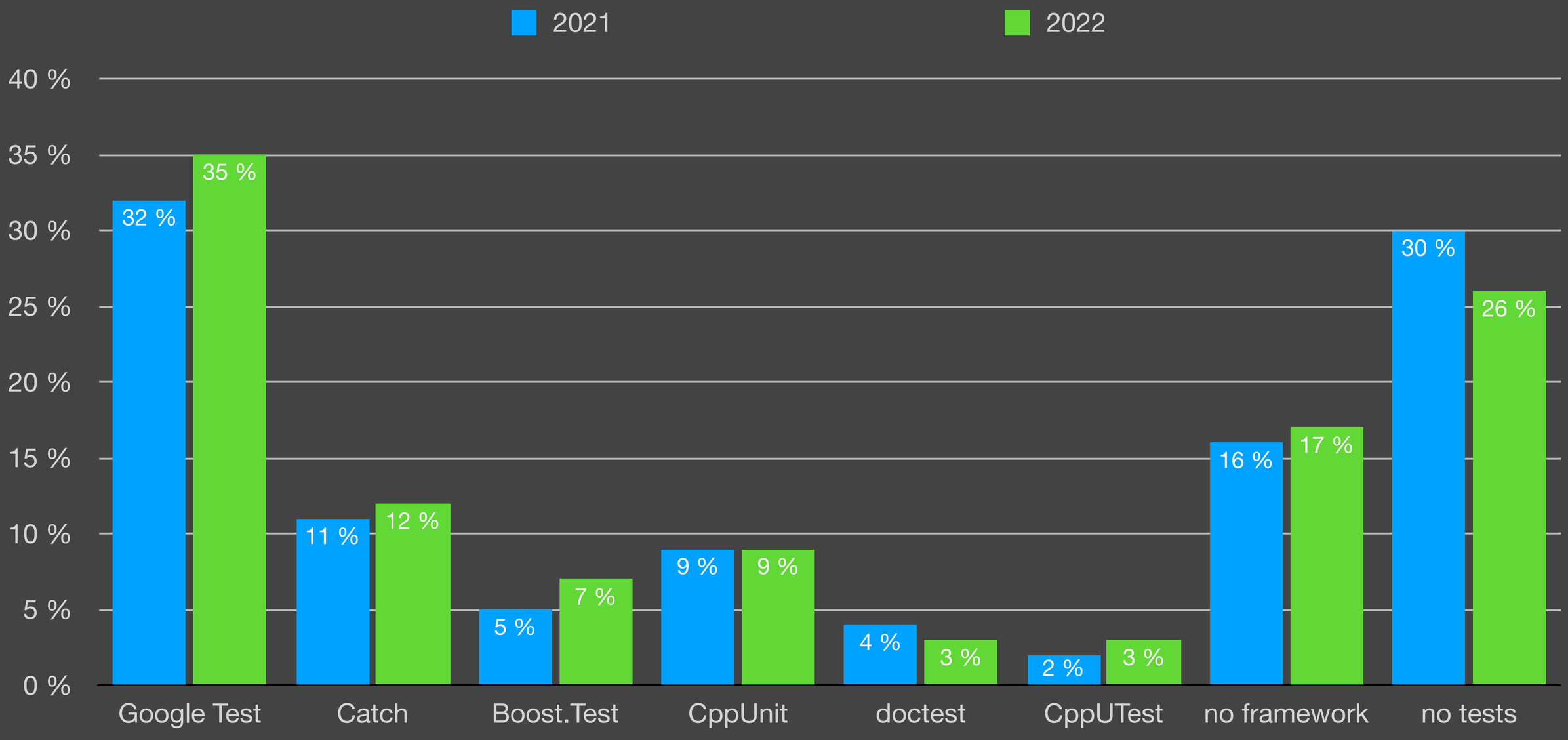
Unit Testing

–

75% of all respondents say testing plays an integral role in their development.



Unit Testing



Code Coverage
—

1. Line coverage
2. Statement coverage
3. Brach coverage

Code Coverage

1. llvm-com https://llvm.org/docs/CommandGuide/llvm-cov.html
   - Clang
   - -fprofile-instr-generate -fcoverage-mapping
2. gcov https://gcc.gnu.org/onlinedocs/gcc/Gcov.html#Gcov
   - GCC
   - -fprofile-arcs -ftest-coverage or an equivalent —coverage
   - llvm-cov gcov with Clang since v4.2

TL;DR

1. Do we have a standard C++ toolset? Not as Swift or Rust.
2. Do we have commonly used tools for various use cases? Yes.
3. Do we miss some tools? Yes.
4. Can we unify more? Yes.

References

1. [JetBrains Developer Ecosystem 2021](https://www.jetbrains.com/lp/devecosystem-2021)
2. [ReSharper C++ Syntax Style](https://blog.jetbrains.com/rscpp/2021/03/30/resharper-cpp-2021-1-syntax-style/)
3. [C++ Market Infographic, 2015](https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/)
4. [What Belongs In The C++ Standard Library? Bryce Adelstein Lelbach](https://www.youtube.com/watch?v=OgM0MYb4DqE)
5. [Code Analysis++, C++Now 2019](https://www.youtube.com/watch?v=qUmG61aQyQE)