

# Standard C++ toolset

Anastasia Kazakova, JetBrains  
@anastasiak2512

Meeting C++ 2022

# About me

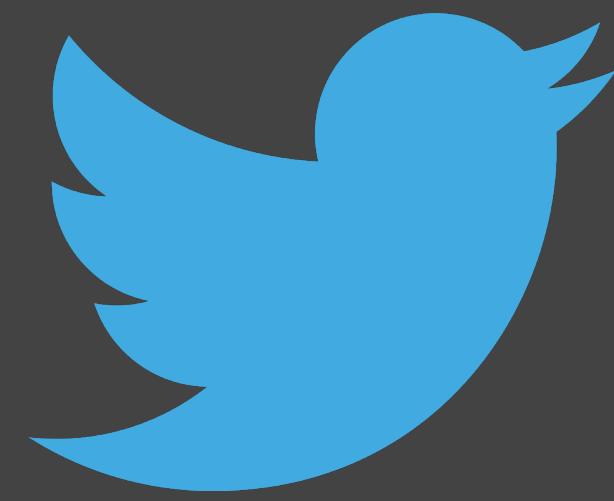
---



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++

---

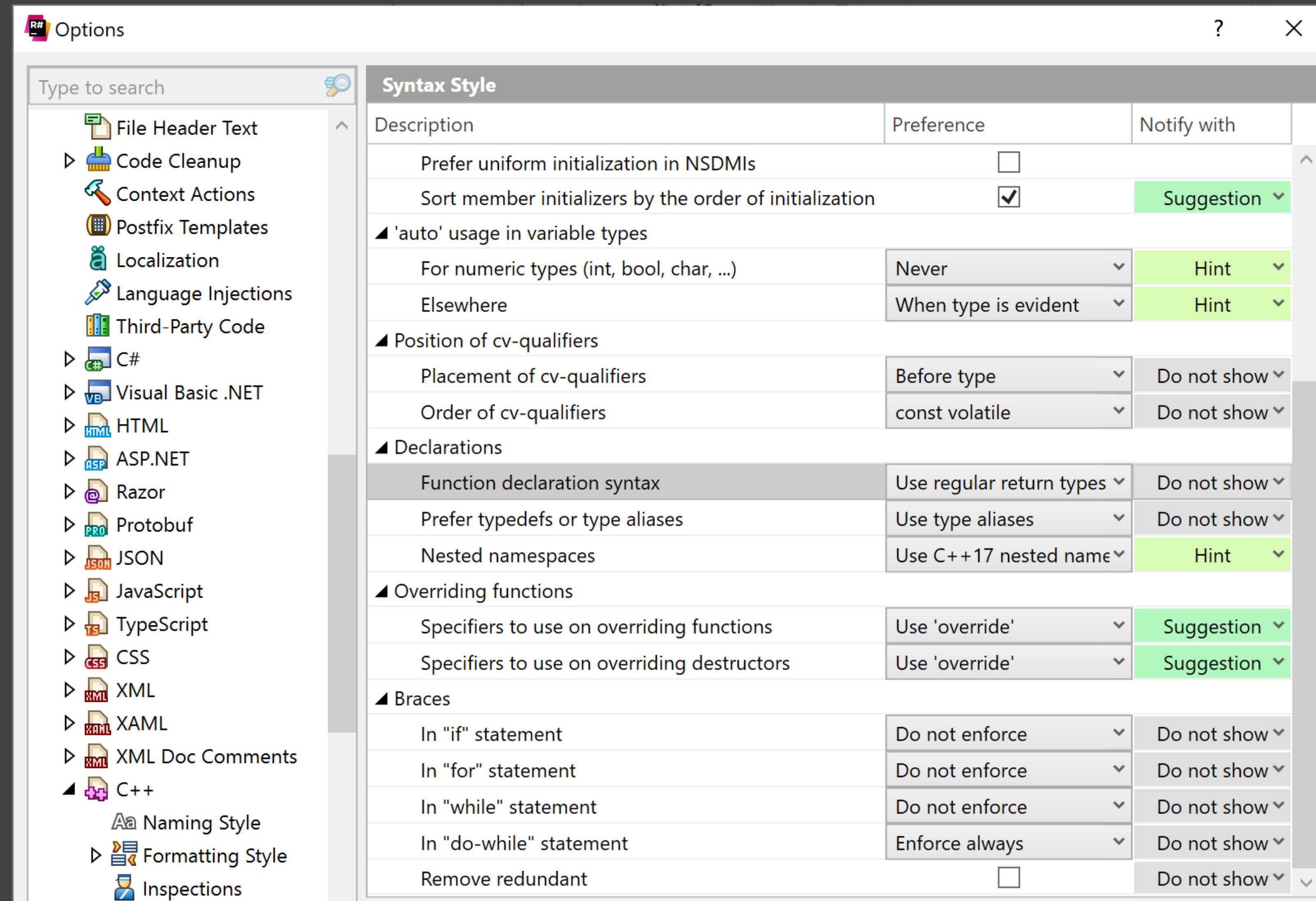
Almost Always Auto or when type is evident or never for numeric types.

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++

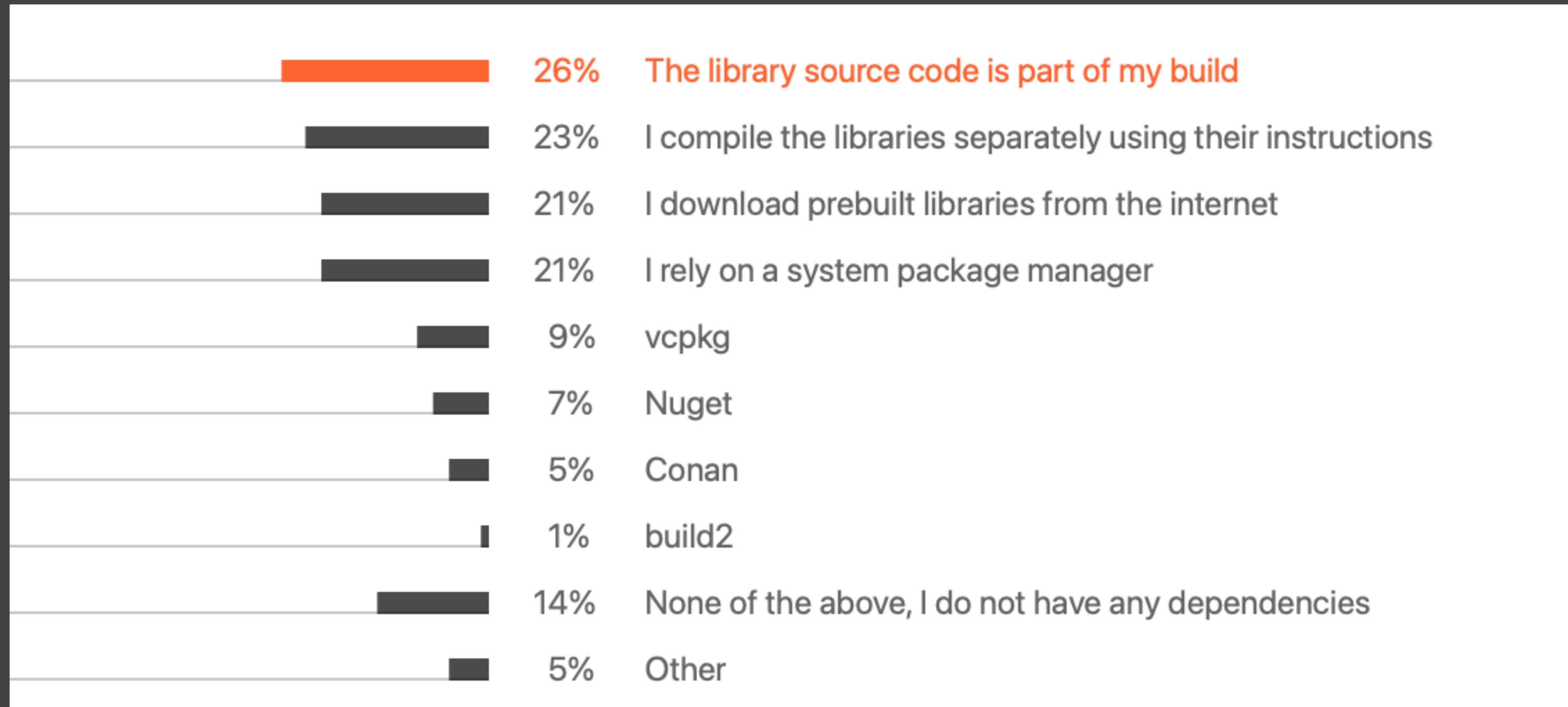
---

```
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++ now

What Belongs In The C++ Standard Library?

## Universality Is A Double Edged Sword

The diagram illustrates the universality of C++ through a central blue hexagonal icon containing a white 'C++' symbol. Four curved lines extend from this center to four groups of colored boxes:

- Any Problem:** HPC (light green), Finance (light blue), Gaming (red), Apps (orange), Robotics (purple), Vehicles (brown).
- Any Platform:** Windows logo, Raspberry Pi, Linux logo, Server hardware, Apple logo.
- Any Paradigm:** Imperative (teal), Object Oriented (yellow), Functional (green), Generic (purple), Parallel (gold), Reactive (grey).

#include <C++>

Copyright (C) 2021 Bryce Adelstein Lelbach

34

JET BRAINS

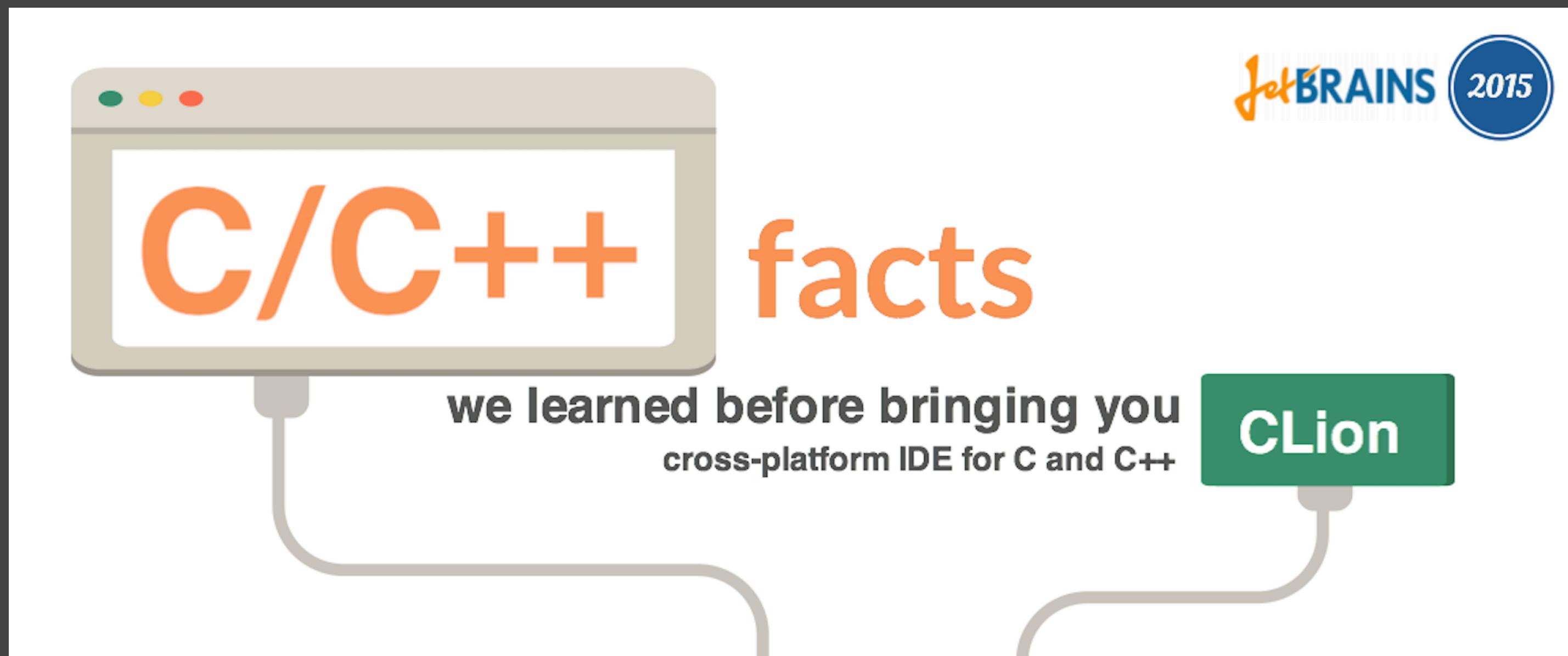
Bloomberg

Engineering

Bryce Adelstein Lelbach

CppNow.org

# C++ toolset. Back in 2015



## Compiler + Build tool + Debugger

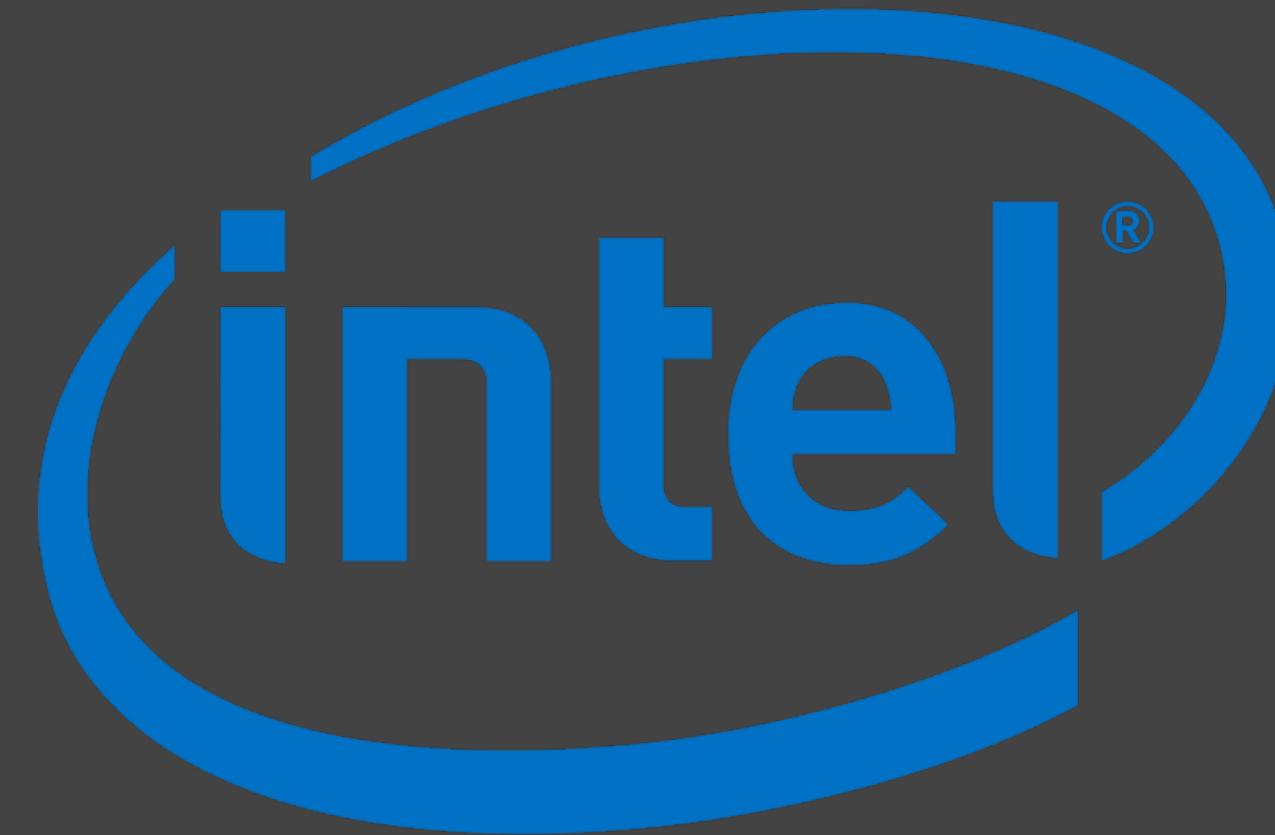
GCC + CMake + GDB7 toolchain takes the top spot for all C++ developers.

On OS X, however, that honor goes to Clang + CMake + LLDB.

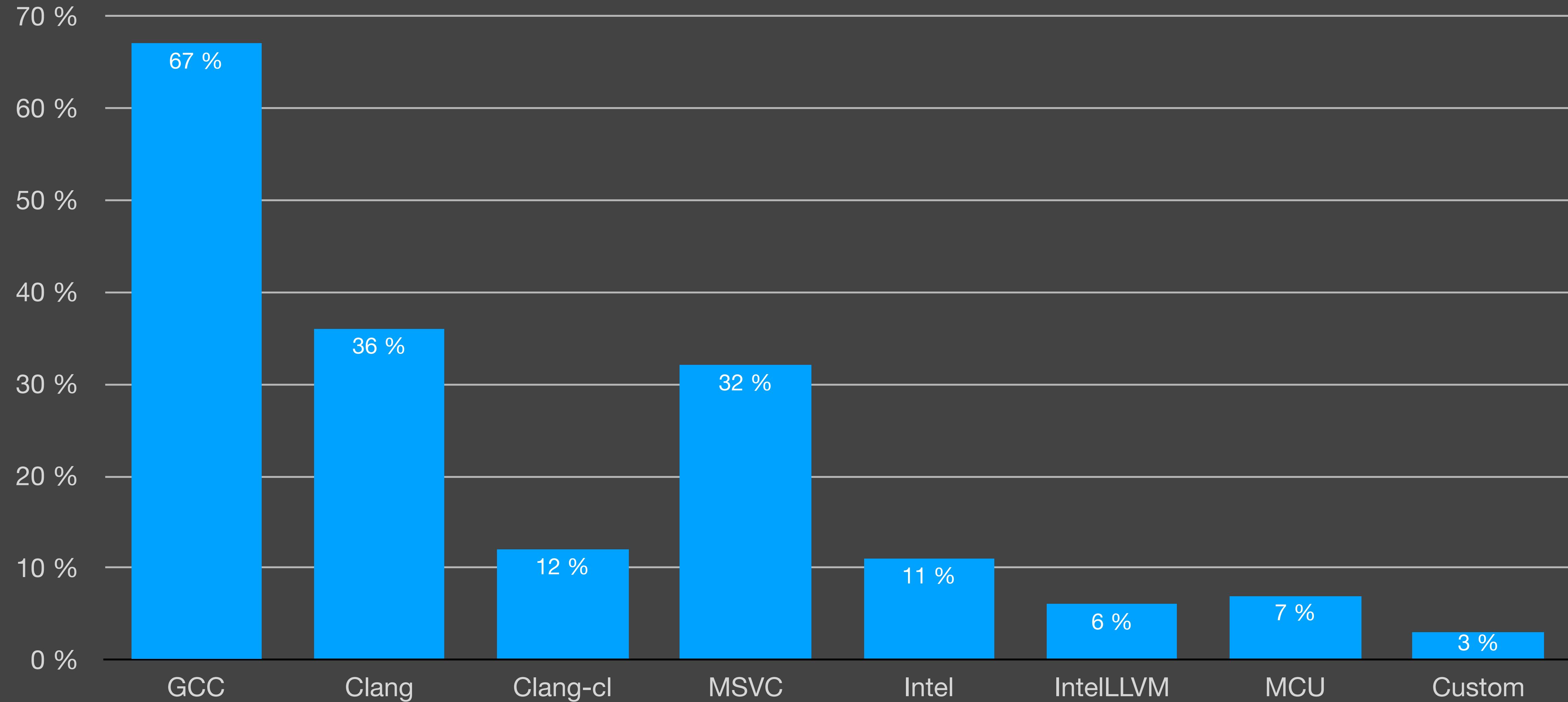


## C++ compilers

---



# 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 XL C++	Sun/Oracle C++	Embarcadero C++ Builder	Nvidia HPC C++ (ex Portland Group/PGI)	Cray	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

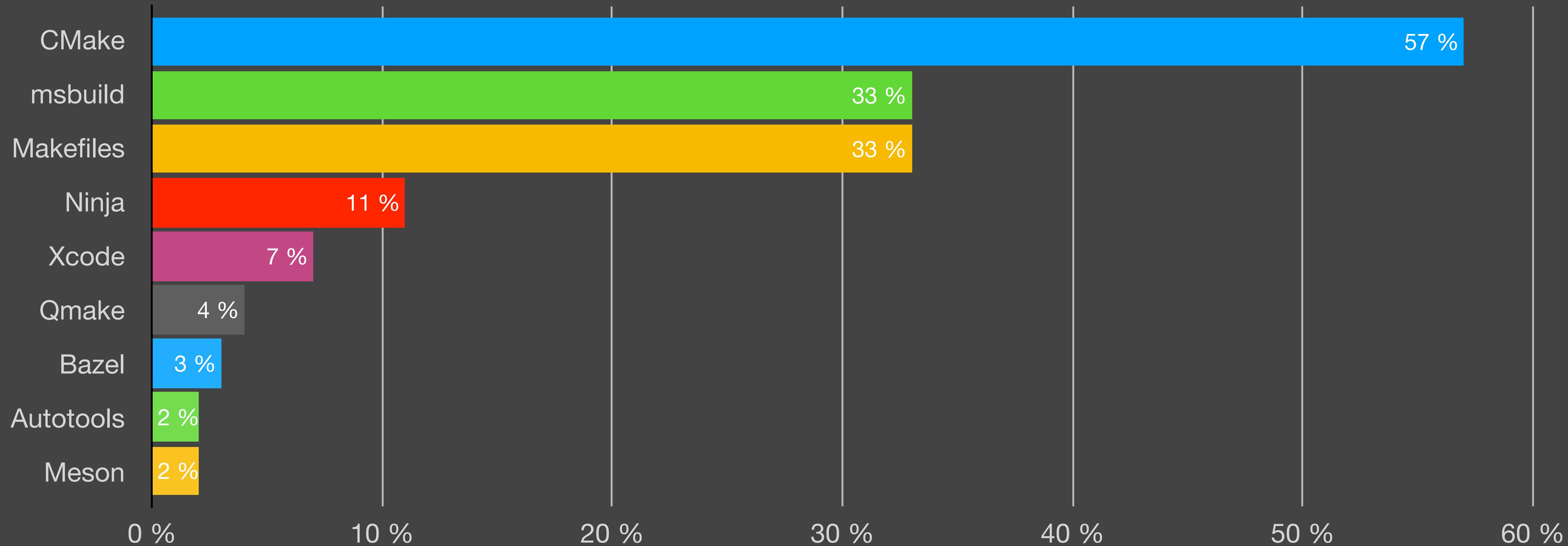
---

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

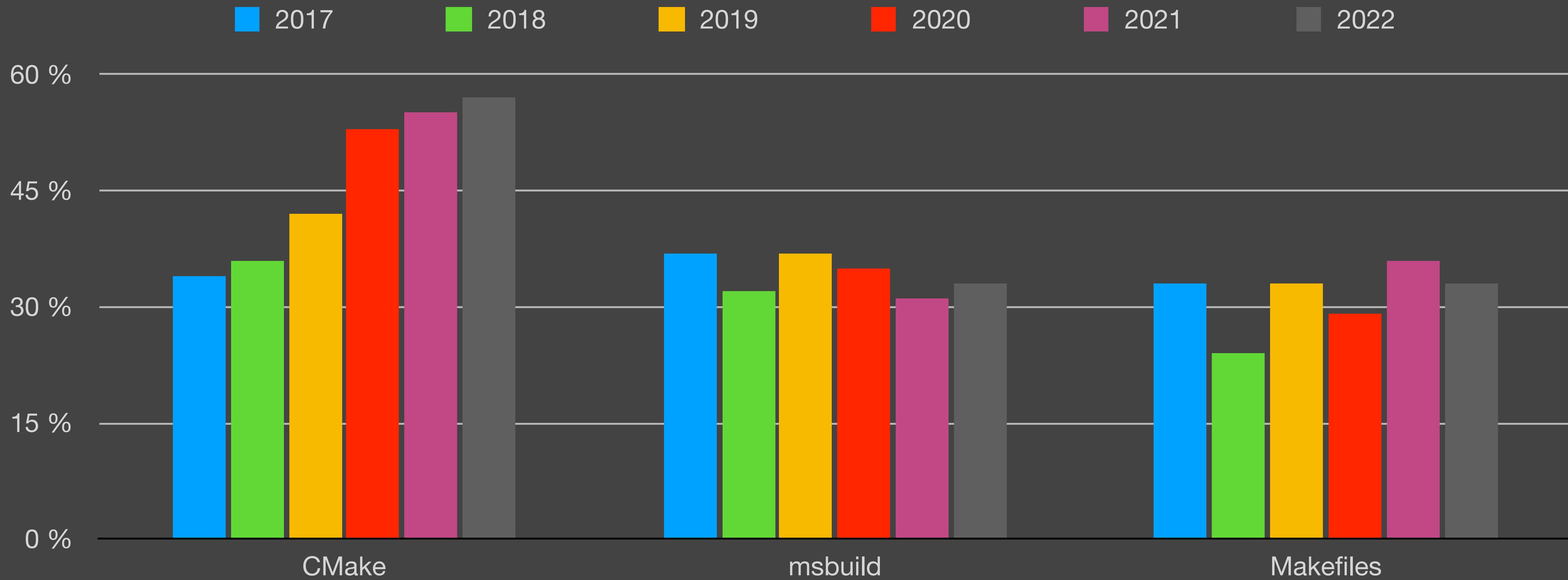
Clang – basis for C++ tools

# 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!

© Bryce Adelstein Lelbach, CppNow 2021

# 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

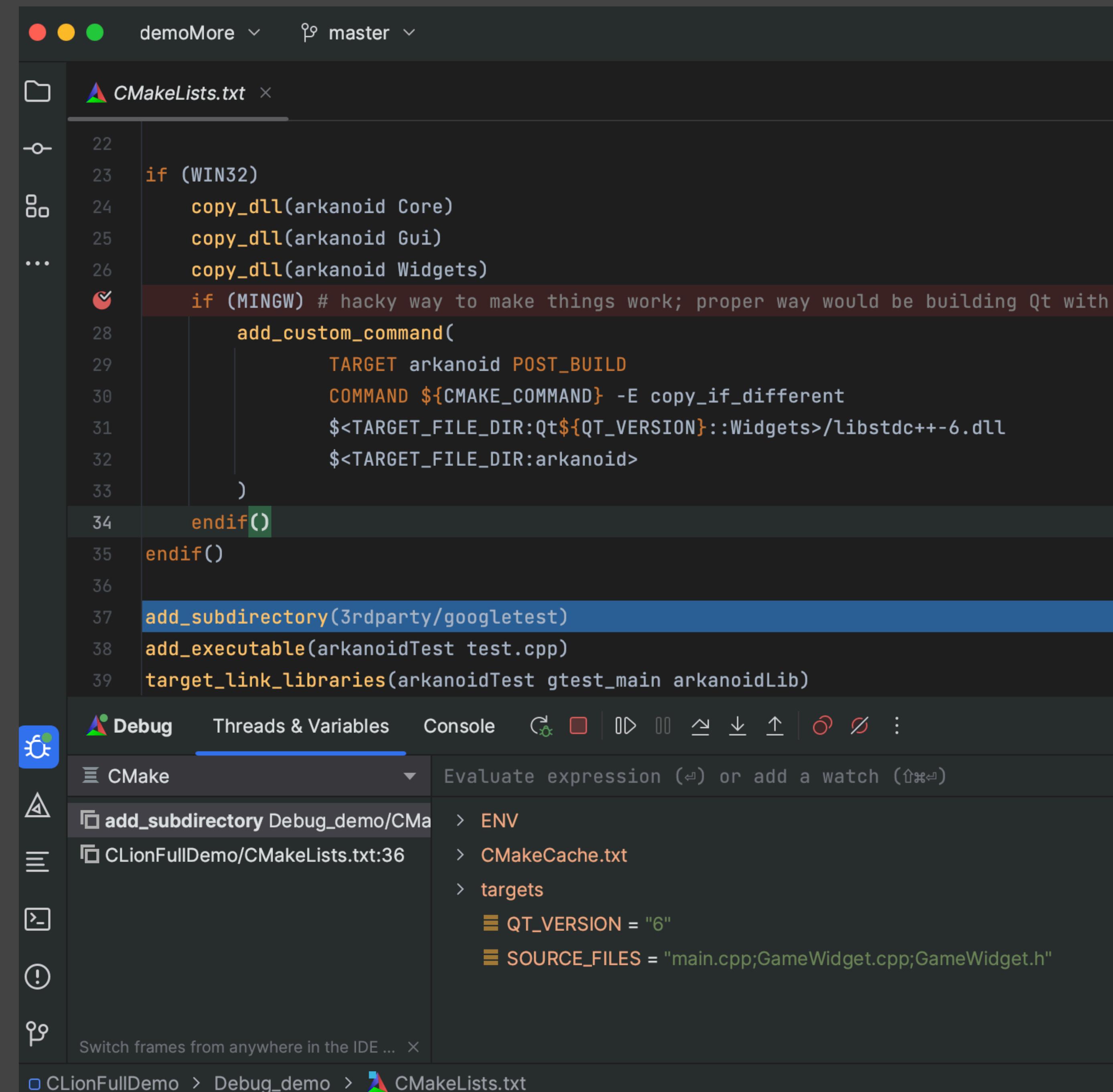
## 1. Good for quizzes

- CMake + Conan: 3 Years

Later - Mateusz Pusz,

CppNow 2021

## 2. Needs a debugger



The screenshot shows the CLion IDE interface. The top window displays the file `CMakeLists.txt` with code related to copying DLL files and adding custom commands for MinGW builds. The bottom window shows the "Variables" tab of the debugger, which lists environment variables like `QT_VERSION` and `SOURCE_FILES`, and CMake cache variables like `CMakeCache.txt`.

```
22
23 if (WIN32)
24     copy_dll(arkanoid Core)
25     copy_dll(arkanoid Gui)
26     copy_dll(arkanoid Widgets)
27
28     if (MINGW) # hacky way to make things work; proper way would be building Qt with
29         add_custom_command(
30             TARGET arkanoid POST_BUILD
31             COMMAND ${CMAKE_COMMAND} -E copy_if_different
32             ${TARGET_FILE_DIR:Qt${QT_VERSION}:Widgets}/libstdc++-6.dll
33             ${TARGET_FILE_DIR:arkanoid}
34     )
35 endif()
36
37 add_subdirectory(3rdparty/googletest)
38 add_executable(arkanoidTest test.cpp)
39 target_link_libraries(arkanoidTest gtest_main arkanoidLib)
```

Debug  
Threads & Variables  
Console  
CMake  
add\_subdirectory Debug\_demo/CMakeLists.txt  
CLionFullDemo/CMakeLists.txt:36  
ENV  
CMakeCache.txt  
targets  
QT\_VERSION = "6"  
SOURCE\_FILES = "main.cpp;GameWidget.cpp;GameWidget.h"

# CMake and C++20 modules

---

## 1. CMake + VS generator

```
add_executable(ModuleSample a.hxx 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

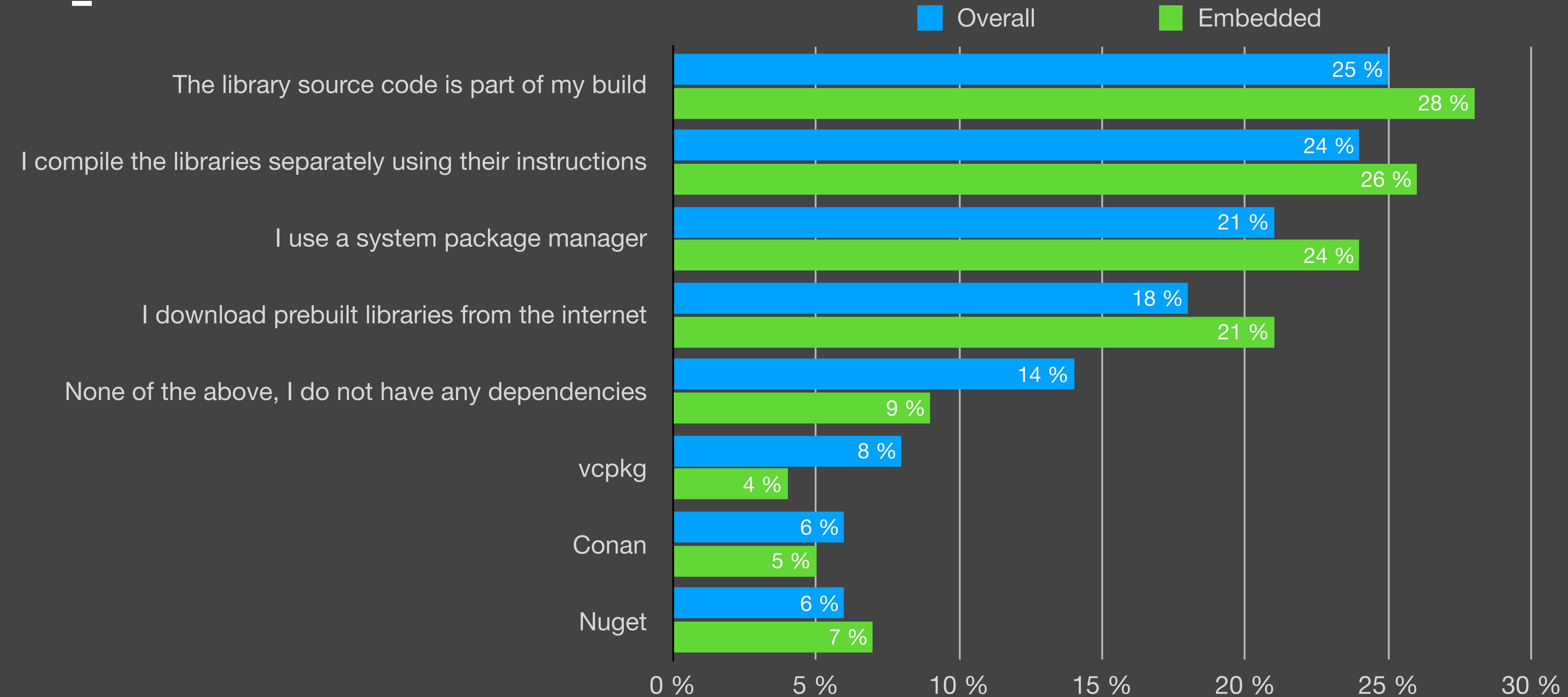
---

## 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



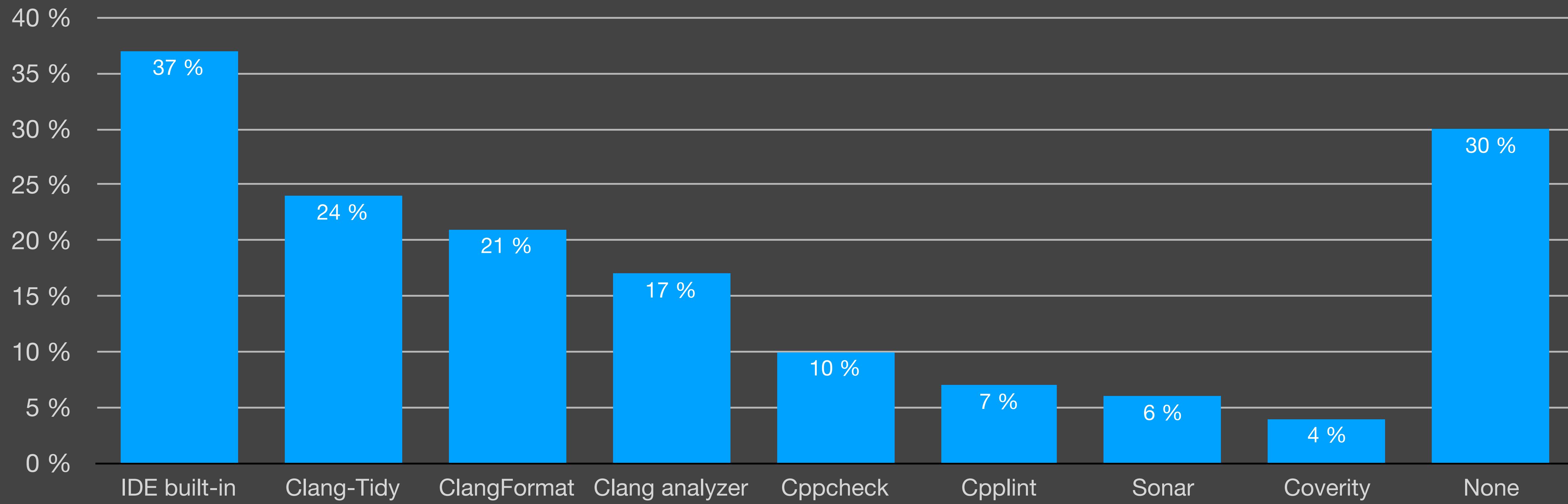
# ClangFormat

---

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

# Code Analysis

---



# Clang-Tidy

---

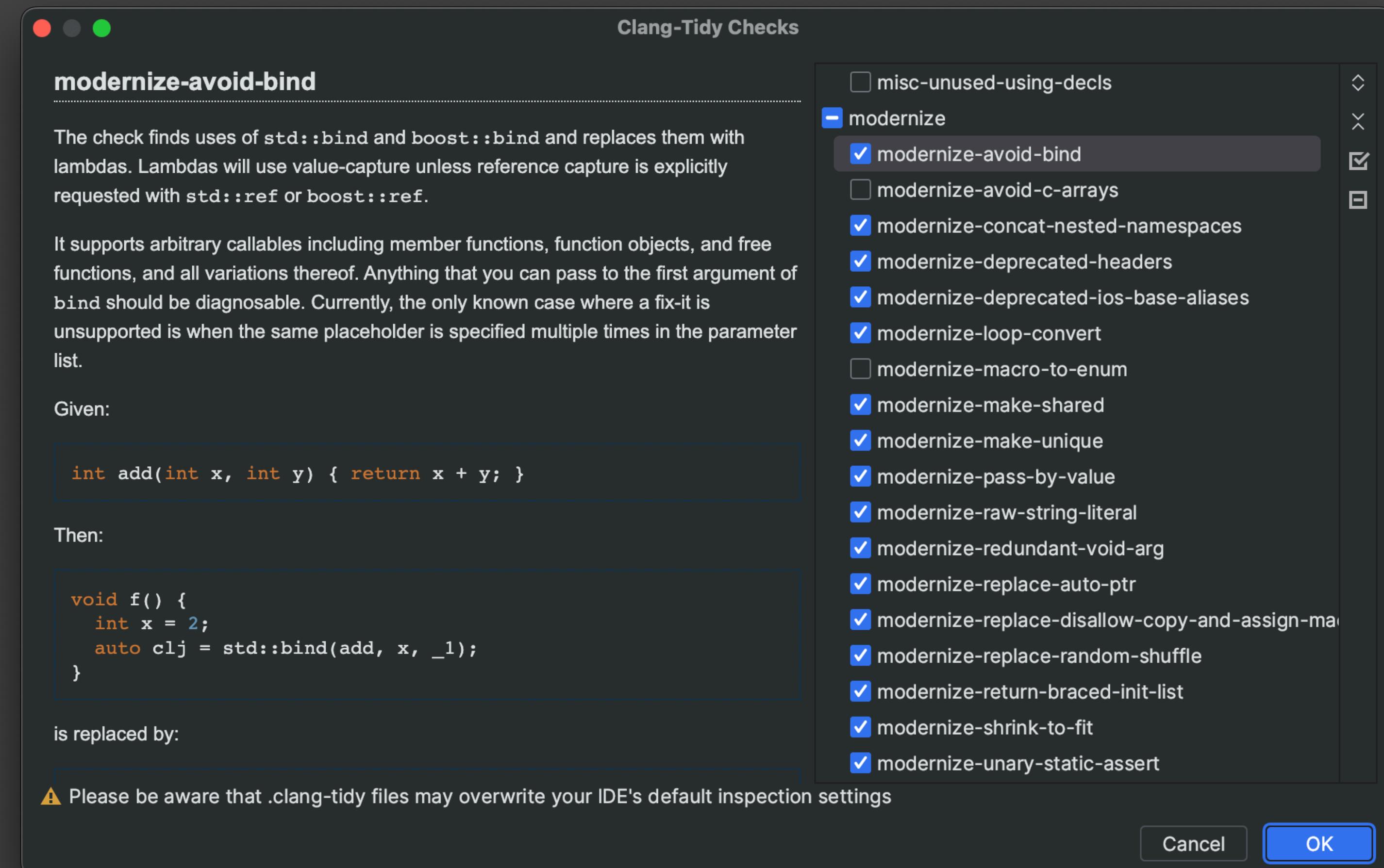
1. Baseline

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.

```
class Deref {  
    int* foo() {  
        return nullptr;  
    }  
  
public:  
    void bar() {  
        int* buffer = foo();  
        buffer[0] = 0;  
    }  
};
```

Pointer may be null

```
int *Deref::buffer = foo();
```

# 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

The screenshot shows a CI pipeline interface with the following details:

- Top Bar:** Conversation 0, Commits 4, Checks 1, Files changed 6, 1 fail.
- SonarQube Analysis:** Failed — 14 days ago. Includes a link to "Add Foo".
- SonarQube Code Analysis:** Failure: ran 10 days ago in less than 5 seconds by @johndoe, feature/johndoe/test.
- Quality Gate failed:** Failed. Issues: Reliability Rating on New Code (is worse than A), Security Rating on New Code (is worse than A), 68.2% Coverage on New Code (is less than 80%).
- Analysis details:** 15 Issues: 1 Bug, 1 Vulnerability, 13 Code Smells. See issues details on SonarQube.
- Coverage and Duplications:** 68.2% Coverage (72.4% Estimated after merge), 17.7% Duplication (5.3% Estimated after merge). See coverage & duplications details on SonarQube.
- Code Inspection Summary:** Problems: 8 032, Checks: 1 666. A circular chart shows the distribution of issues by severity: Critical (red), Major (orange), Moderate (yellow), Minor (light green), and Info (blue).
- Filter Options:** Files and folders: All, Tool: All, Severity: All, Category: All, Type: All. Save as... button.
- Details View:** Problems 8032, Files 381. A list of issues under "Functions.php":
  - Method call is provided 1 parameters, but the method signature uses 0 parameters (Code smell)
  - Method call is provided 1 parameters, but the method signature uses 0 parameters (Code smell)
- Code Snippet:** project/src/Framework/Assert/Functions.php:

```
2375     function isTrue(): IsTrue
2376     {
2377         return Assert::isTrue(...func_get_args());
2378     }
2379 }
```

A callout highlights line 2377 with the note: "Reports the function/method calls that take more parameters than specified in their declaration."
- Actions:** Open file in IDE, More actions.

# Code Analysis++

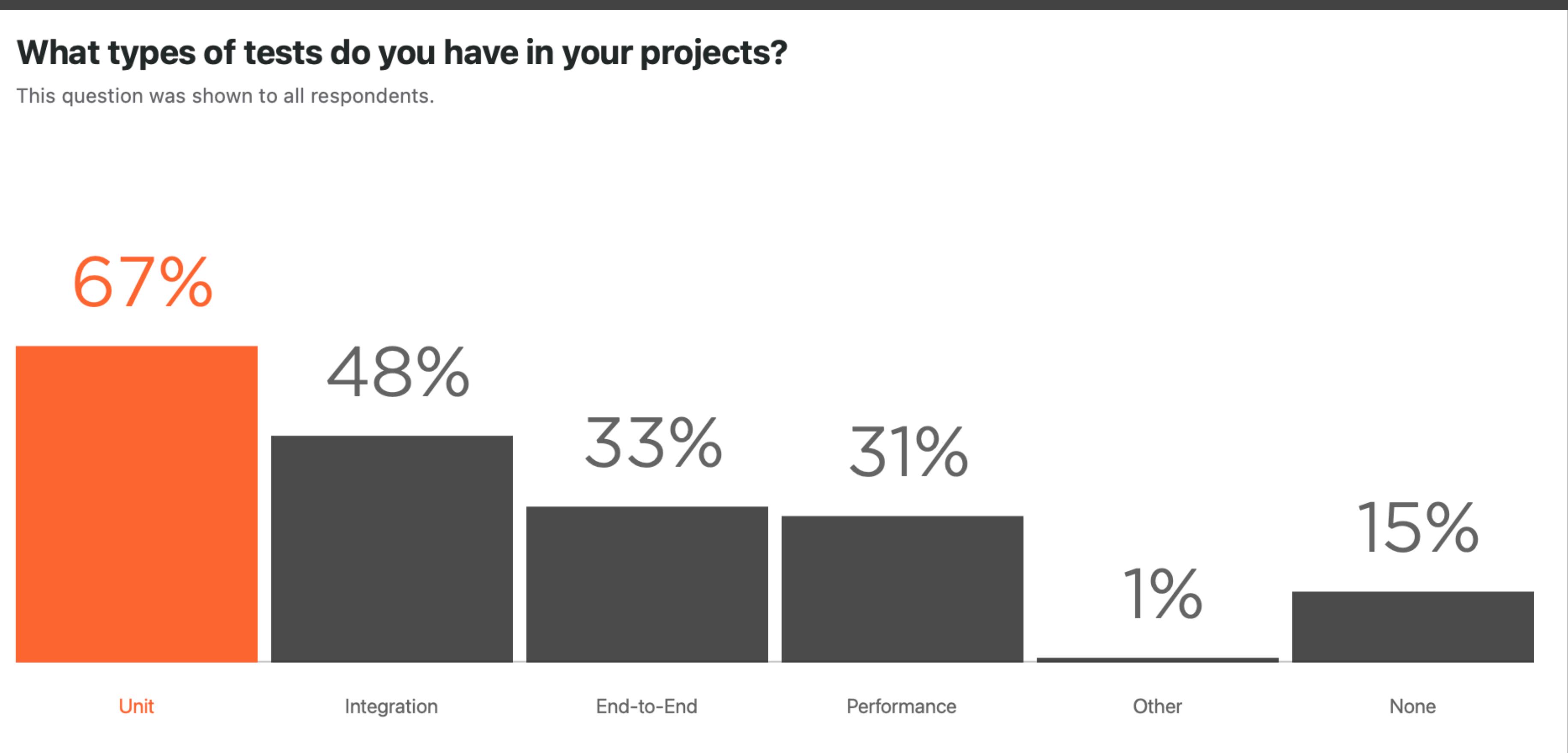
---

Find my talk from [C++Now 2019](#) or NDC TechTown 2022

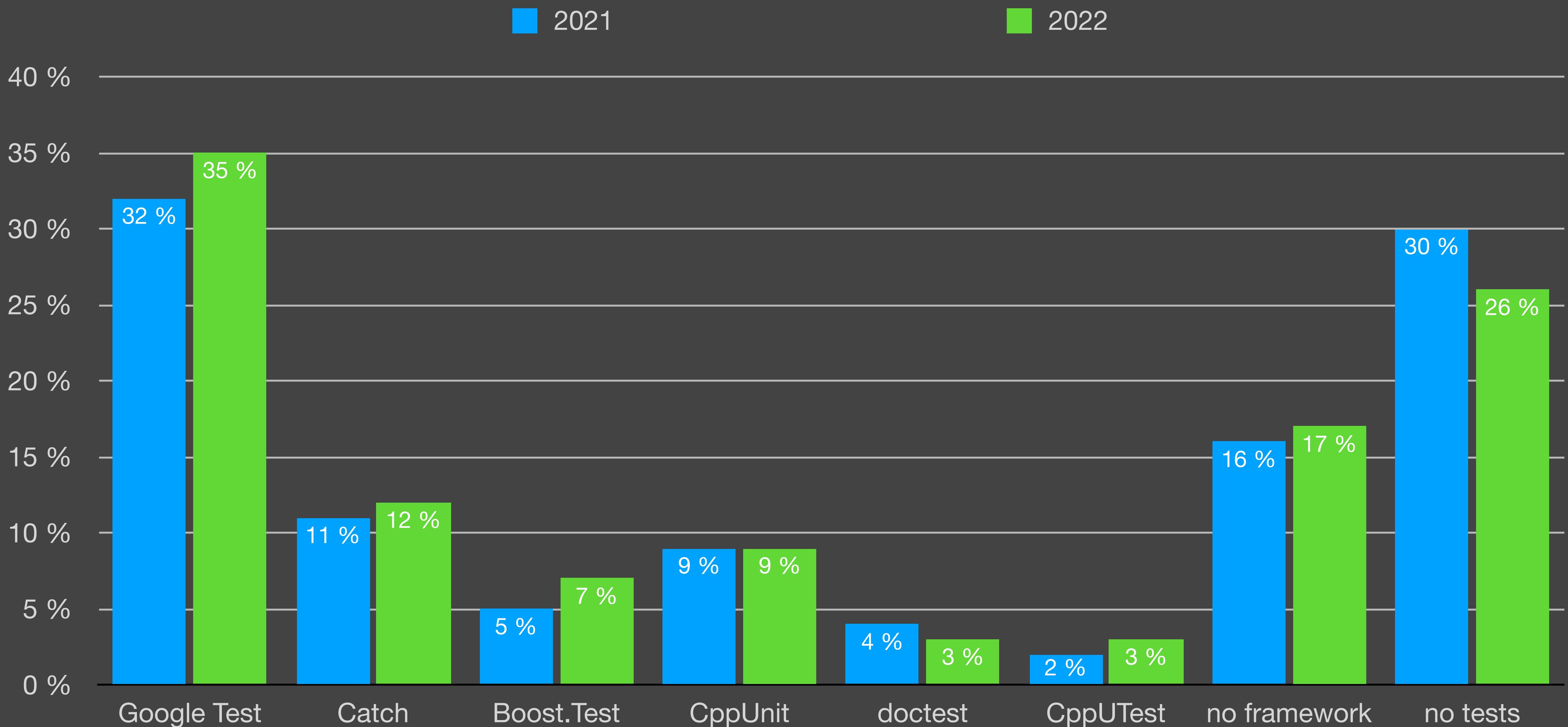
# Unit Testing

---

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



# Unit Testing



# Code Coverage

---

1. Line coverage
2. Statement coverage
3. Branch coverage

# Code Coverage

---

1. llvm-com <https://llvm.org/docs/CommandGuide/llvm-cov.html>
  - Clang
  - `-fprofile-instr-generate -fcov-coverage-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/devcosystem-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=0gM0MYb4DqE>)
5. [Code Analysis++, C++Now 2019] (<https://www.youtube.com/watch?v=qUmG61aQyQE>)