Why Is My Build So Slow?

Compilation Profiling and Visualization

SAMUEL PRIVETT

Disclaimer: The views and opinions expressed in this presentation are solely those of the presenters and do not necessarily represent the views, positions, or policies of Johnson & Johnson MedTech.

Table of Contents

Introduction

High Level Concepts

About Me

- Sam Privett (He/Him)
- Robotics Software Engineer @ JnJ
- Curious about compilers

What is this talk?

Visualizing Compilation with Ninja and Clang ✅

Speed up compilers ❌

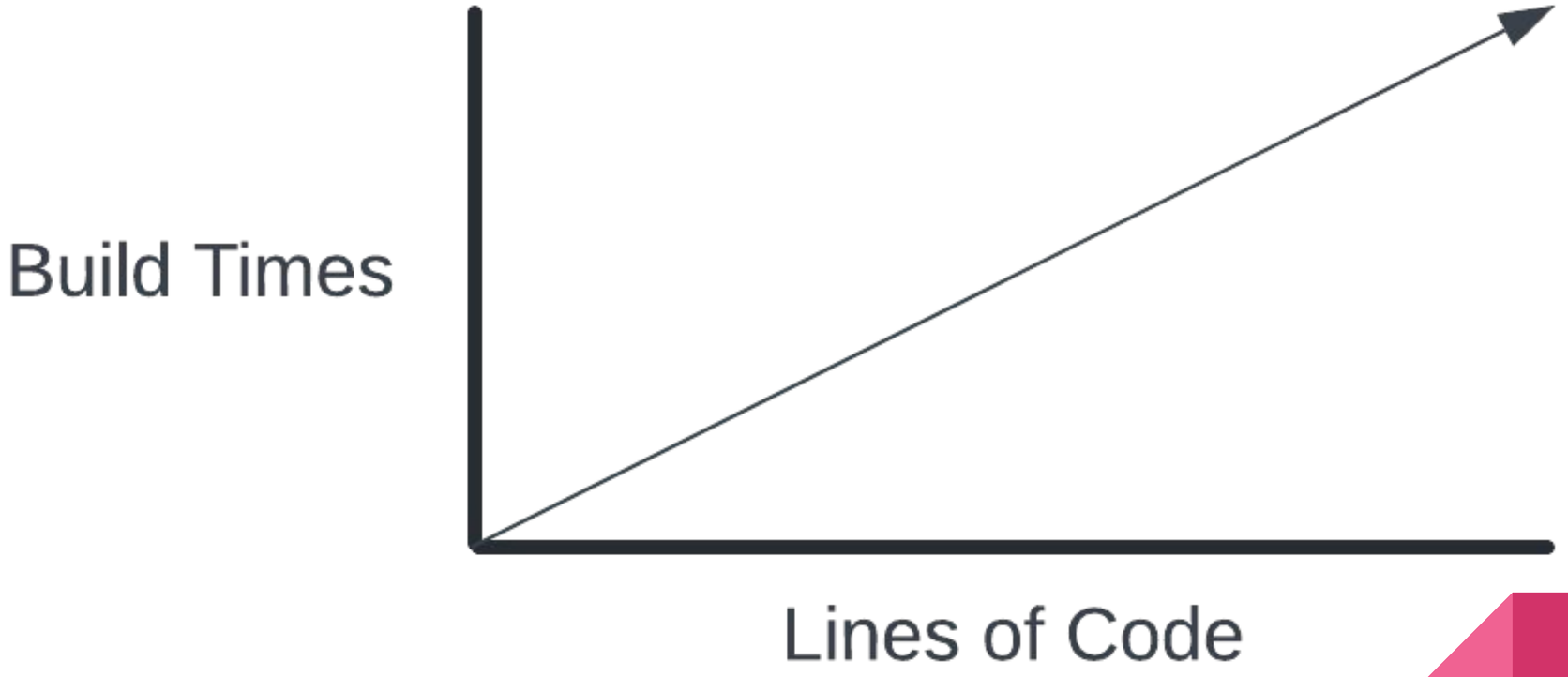
Write code that compiles faster ✅

Introduction

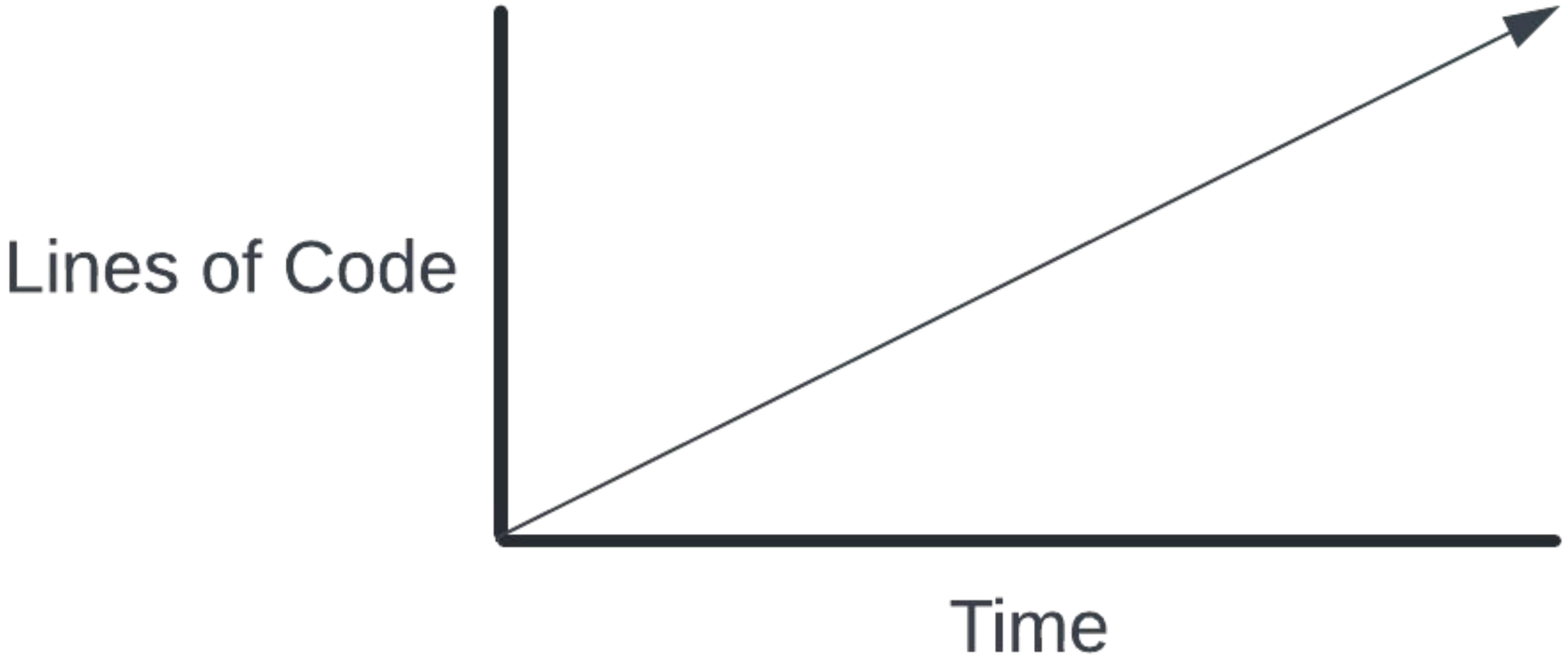
Assumptions

Visualizing Compilation

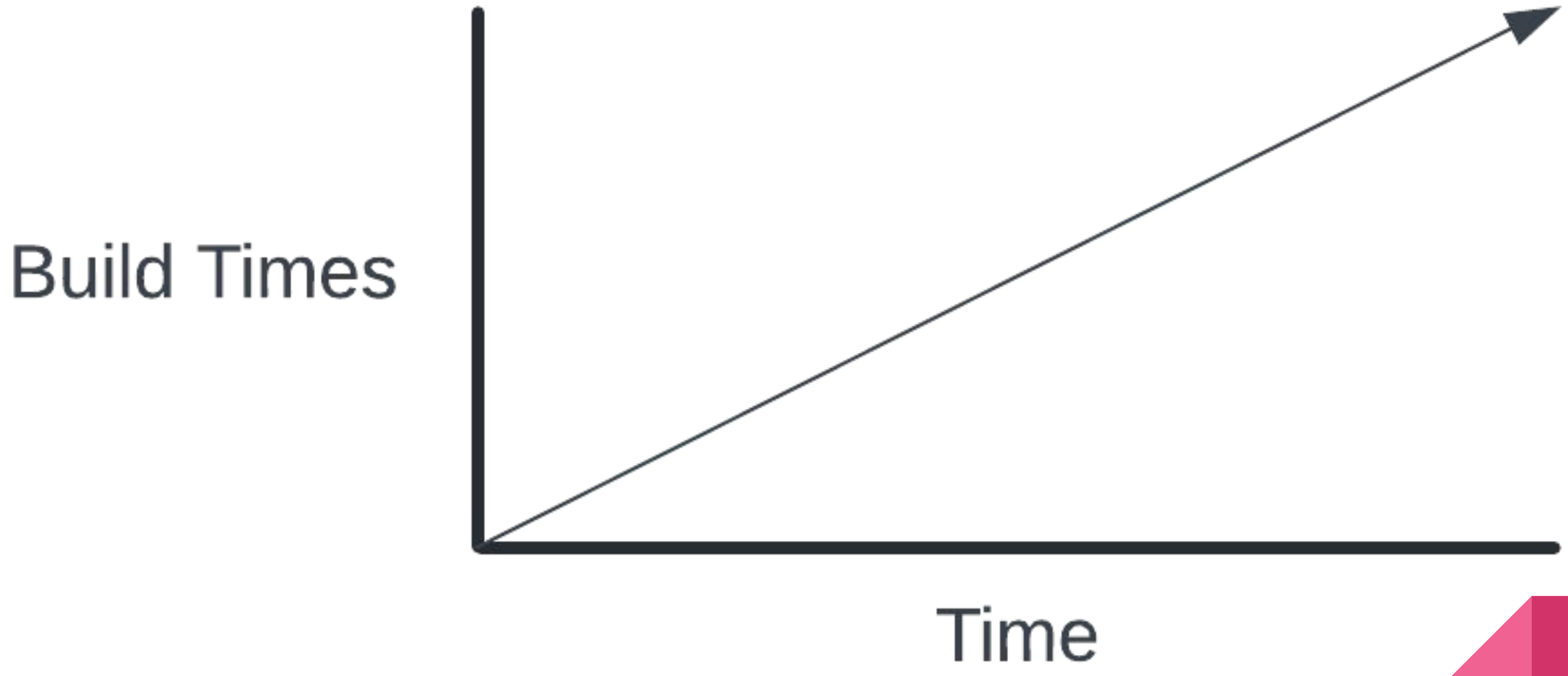
Lines of Code ≈ Longer Build Times



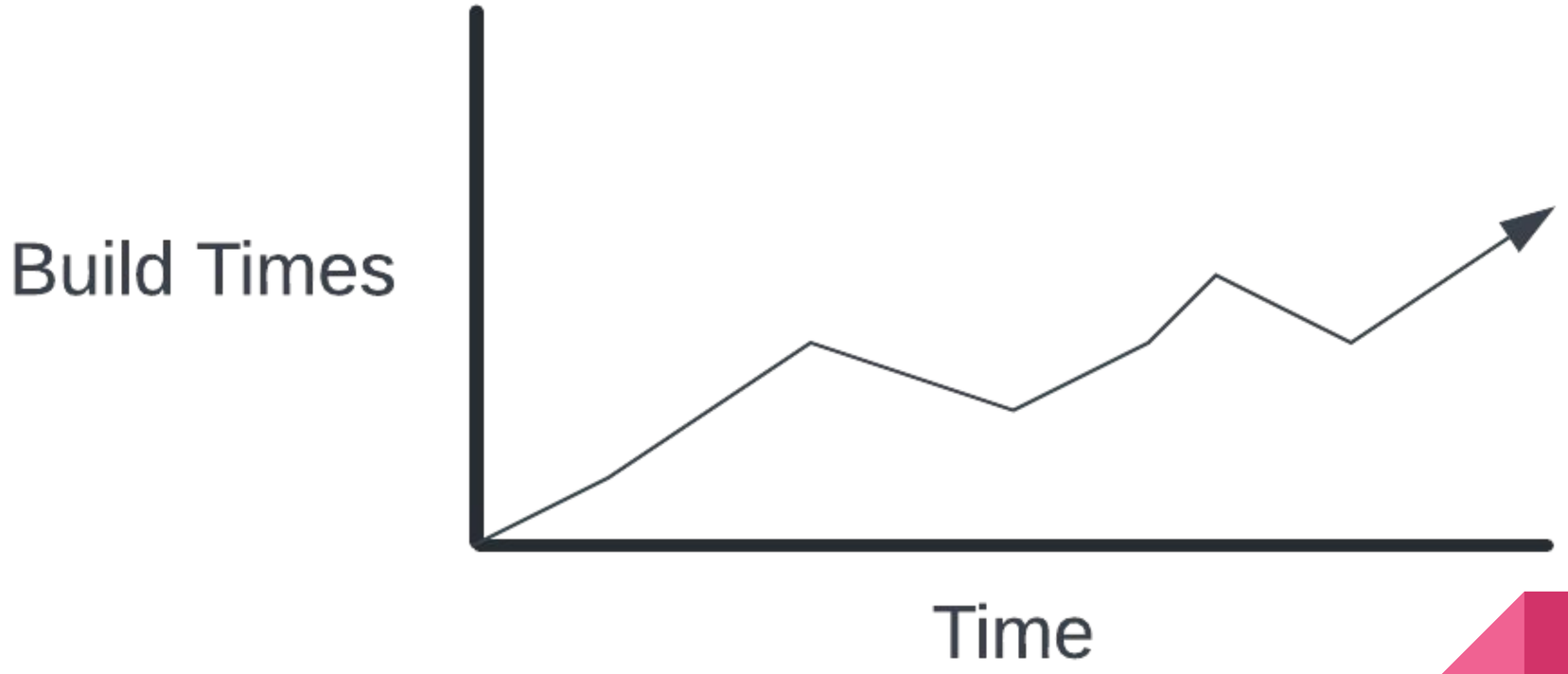
Lines of Code 📈



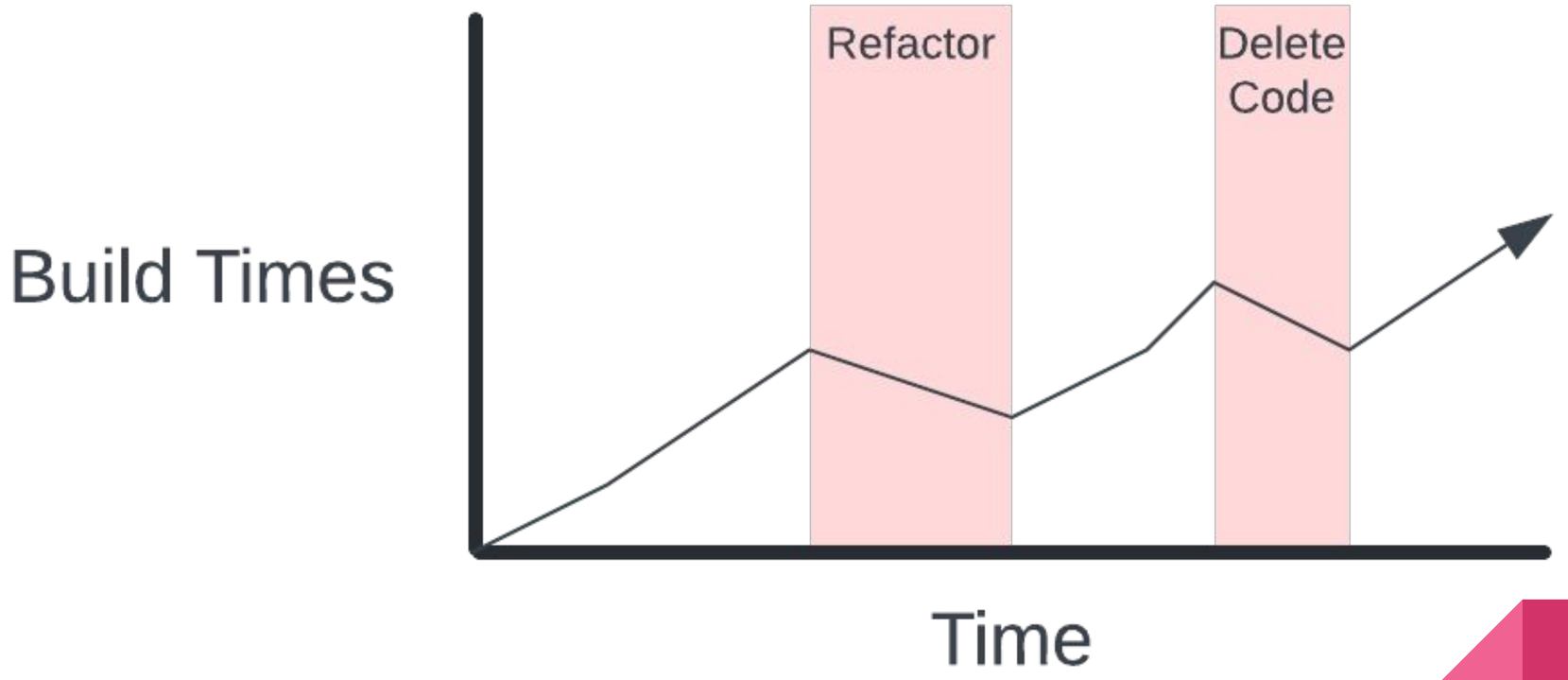
Build Times 📈



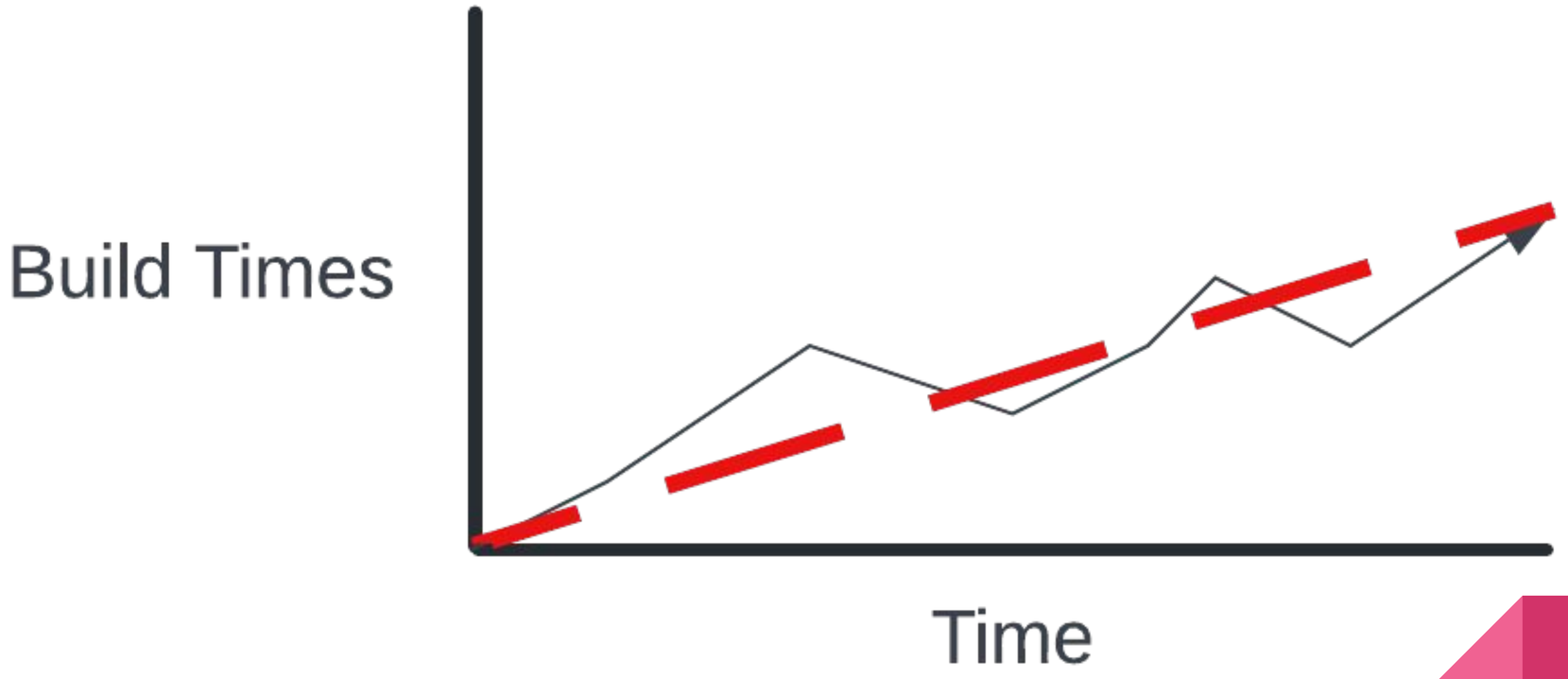
Build Times 📈



Build Times 📈



Build Times 📈



Boiling Frog

Developer Productivity

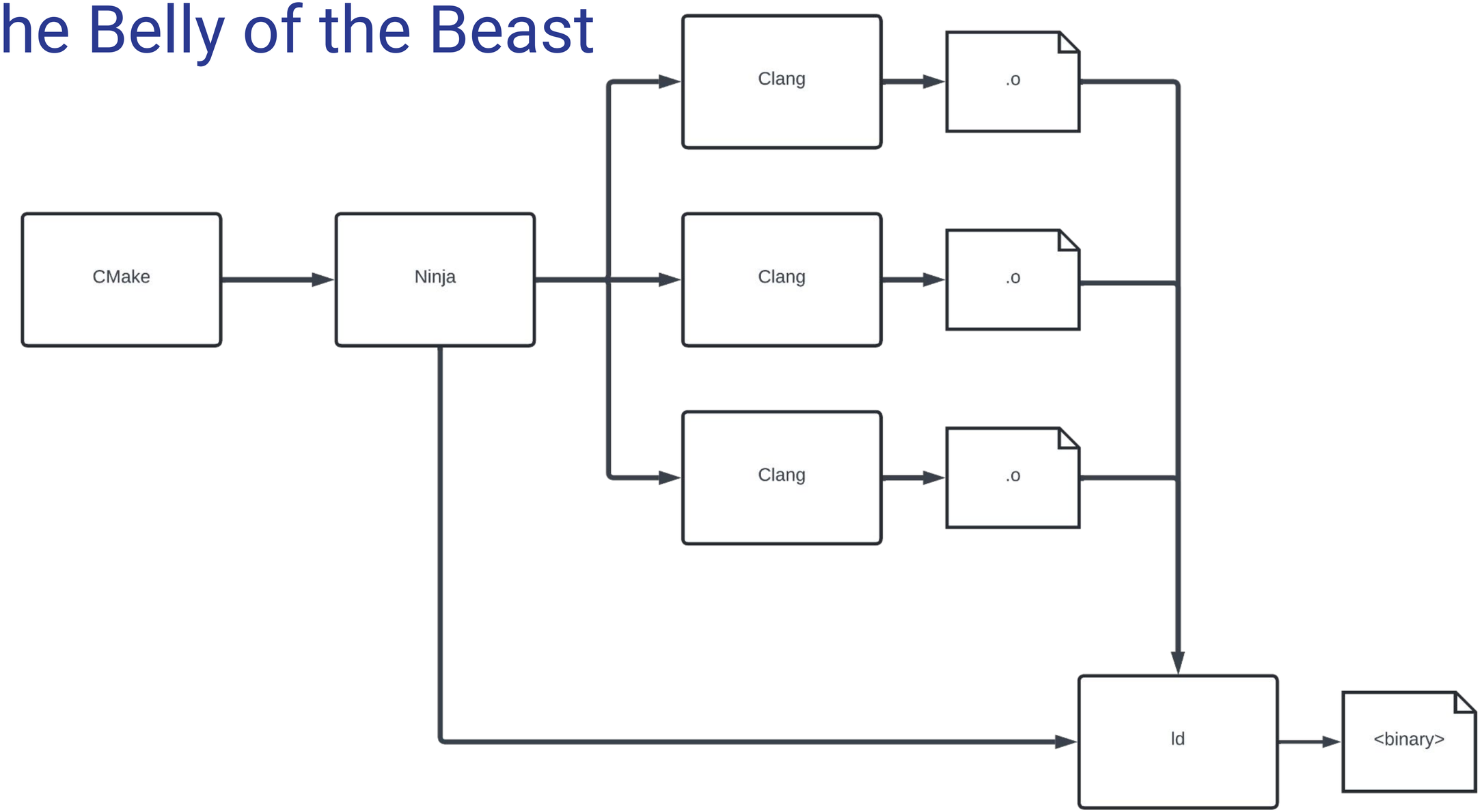
https://xkcd.com/303/

Assumptions

Visualizing Compilation

Single File Gotchas

Into the Belly of the Beast

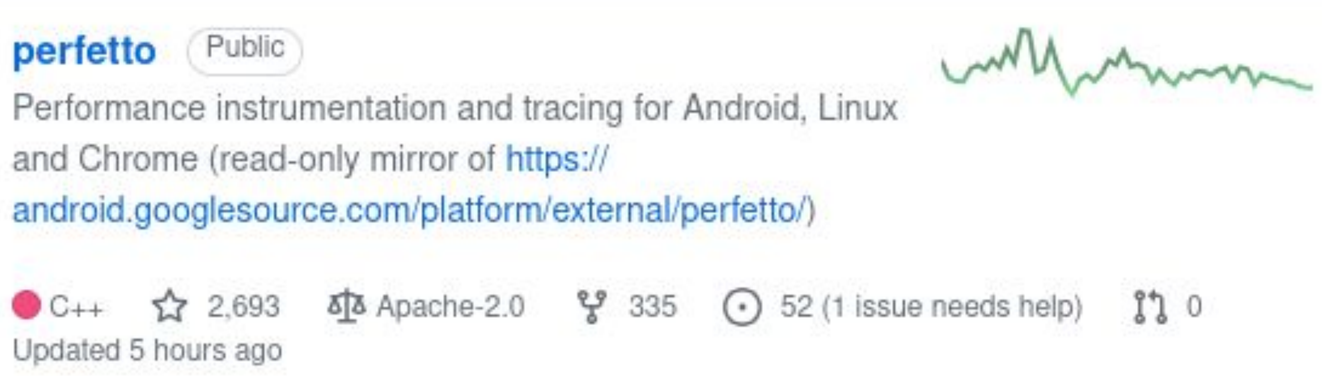


.ninja_log

- Tracks start/stop time in ms
- Name of the output file
- Hash of the command used (compiler, linker, etc)

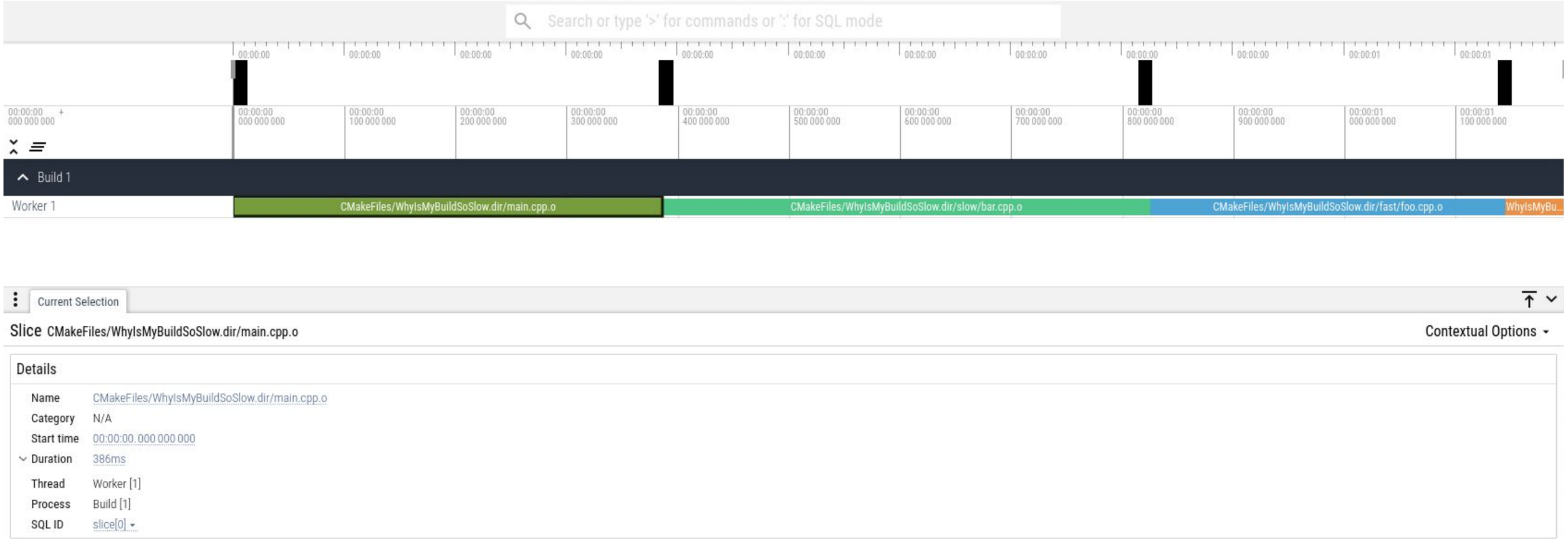
```
# ninja log v5
0       386     1724180262104460299     CMakeFiles/WhyIsMyBuildSoSlow.dir/main.cpp.o    696e8180c4325128
386     825     1724180262544460313     CMakeFiles/WhyIsMyBuildSoSlow.dir/slow/bar.cpp.o        f80e0b2e43465a47
825     1144    1724180262864460323     CMakeFiles/WhyIsMyBuildSoSlow.dir/fast/foo.cpp.o        7c134e49e6045f83
1144    1198    1724180262920460324     WhyIsMyBuildSoSlow      605a48889fdb150e
```

Visualization



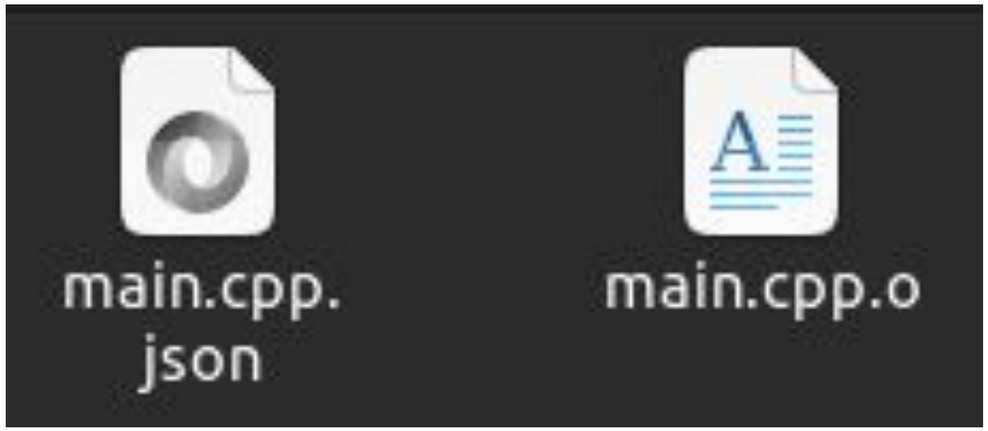
- Interactive Trace Viewer
- https://ui.perfetto.dev/
  - Can build and run server locally as well
- Chrome Event Tracing Format JSON
- … and more!

Visualization

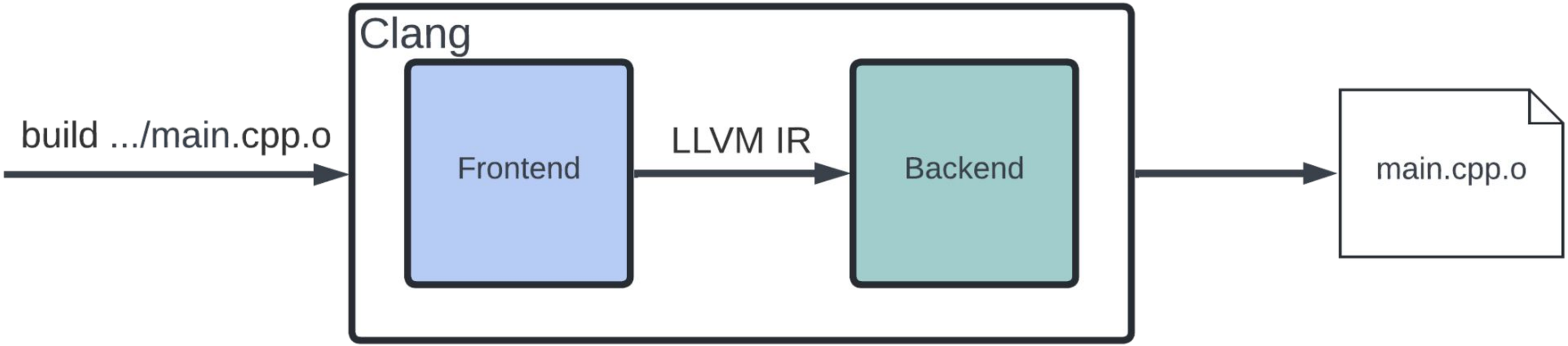


Clang -ftime-trace

- Generates JSON file based on output filename
- Detailed information on where the compiler spent time



We Need to Go Deeper



Visualization

Current Selection

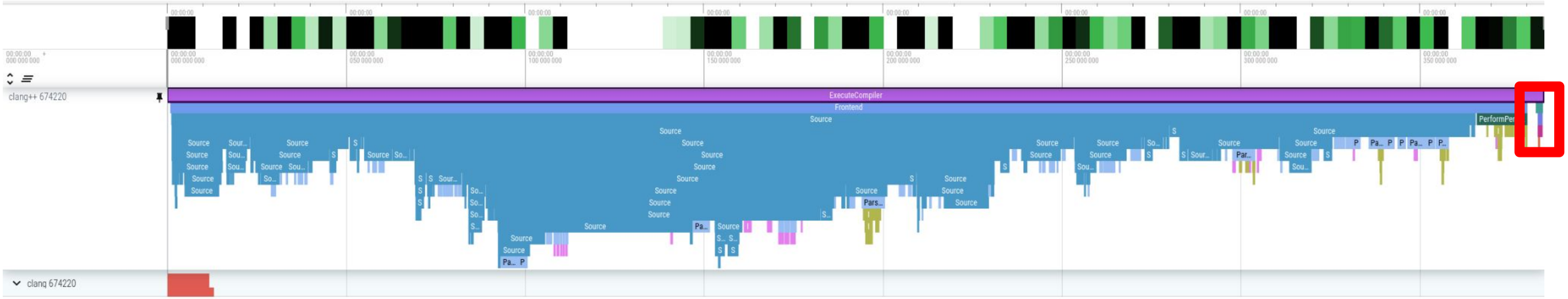
Slice ExecuteCompiler

Contextual Options

Details

Name	ExecuteCompiler
Category	N/A
Start time	00:00:00.000 011 000
Duration	384ms 718us
Thread	clang++ [674220]
Process	clang [674220]
SQL ID	slice[31]

Visualization



Current Selection

Slice ExecuteCompiler

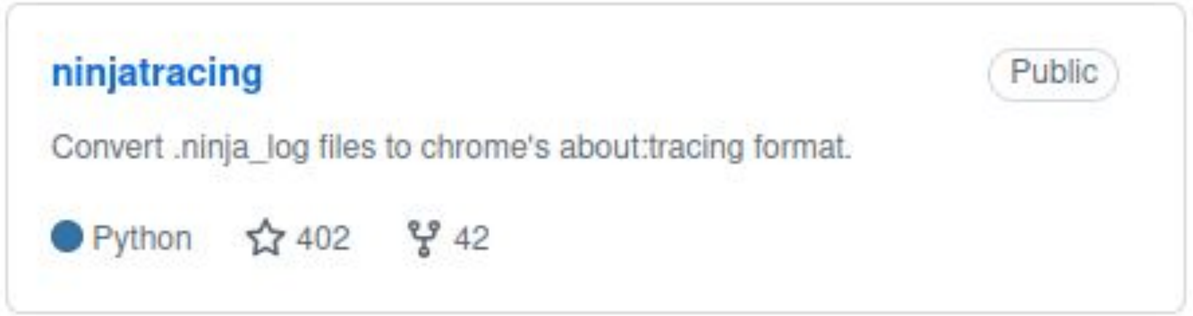
Contextual Options

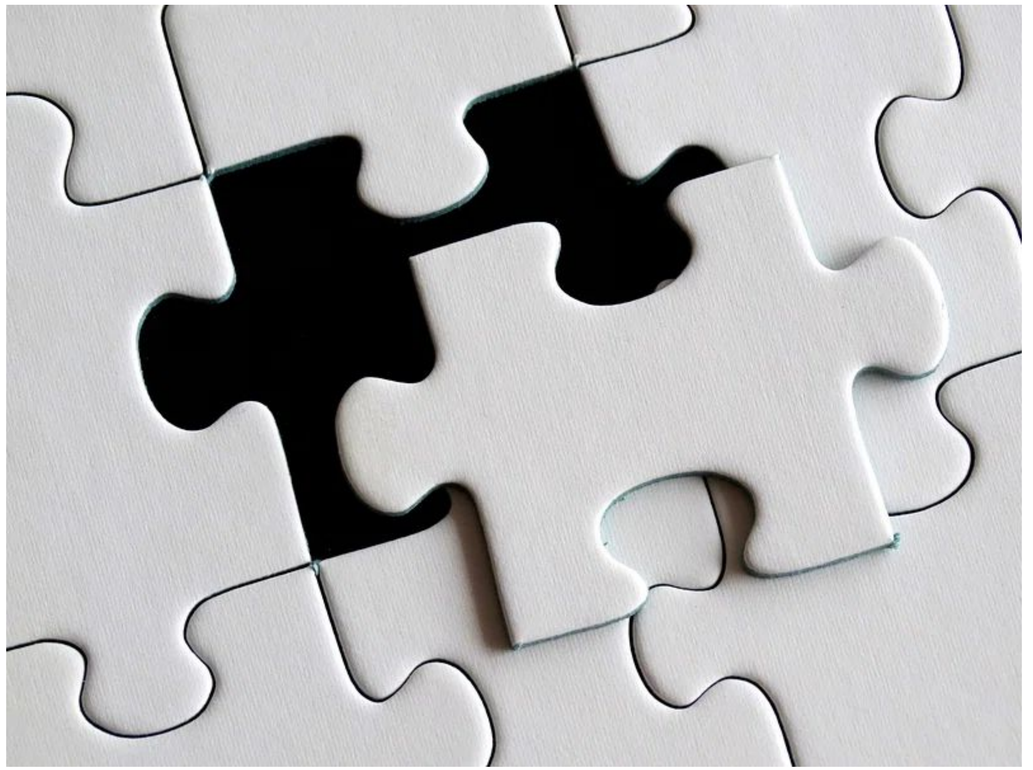
Details

Name	ExecuteCompiler
Category	N/A
Start time	00:00:00.000 011 000
Duration	384ms 718us
Thread	clang++ [674220]
Process	clang [674220]
SQL ID	slice[31]

ninjatracing

- Convert .ninja_log files to chrome tracing format
- **Embed -ftime-trace .json files**





Combining Visualizations

Visualizing Compilation

Single File

Project Level

Includes

```
#include <iostream>

int main() {
	std::cout << "Hello, world!" << std::endl;
}
```

Includes

```
#include <iostream>
```

~69,000 Lines!

```
int main() {
        std::cout << "Hello, world!" << std::endl;
}
```

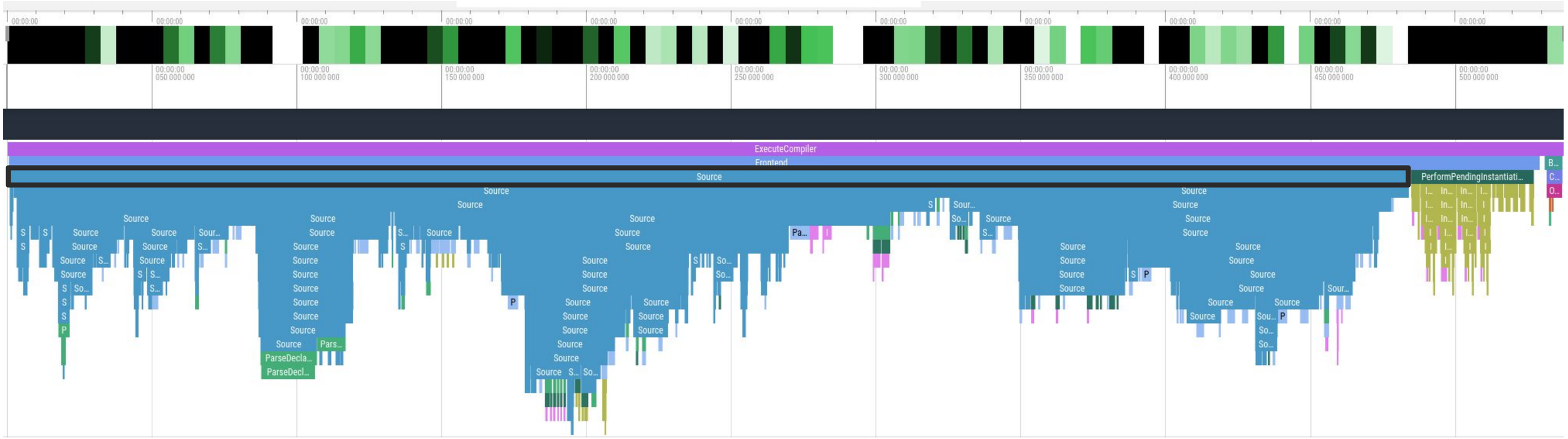
Includes

```
clang main.cpp -stdlib=libc++ -E &> prep_main.cpp
```

```
$ du -bh main.cpp
82 main.cpp
$ du -bh prep_main.cpp
2.9M prep_main.cpp
```

Includes

```
clang main.cpp -stdlib=libc++ -ftime-trace
```



Includes - Recommendations

- Refactor massive header files
  - Having smaller header files gives consumers a better chance at including only what’s strictly necessary for them
- Forward declarations
  - Requires no external tooling
  - Frowned upon for entities defined in another project
  - Can obfuscate dependencies
- Include What You Use
  - IWYU Project

Templates

```
template <int N>
struct Sum {
    static const int value = N + Sum<N - 1>::value;
};

template <>
struct Sum<0> {
    static const int value = 0;
};
```

Templates

```
#include <iostream>

template <int N>
struct Sum {
    static const int value = N + Sum<N - 1>::value;
};

template <>
struct Sum<0> {
    static const int value = 0;
};

int main()
{
    std::cout << Sum<5>::value << std::endl;
}
```

Templates

```
#include <iostream>

template <int N>
struct Sum {
    static const int value = N + Sum<N - 1>::value;
};

template <>
struct Sum<0> {
    static const int value = 0;
};

int main()
{
    std::cout << Sum<5>::value << std::endl;
}
```


```
#include <iostream>

template<int N>
struct Sum
{
  static const int value = N + Sum<N - 1>::value;
};

/* First instantiated from: insights.cpp:5 */
#ifdef INSIGHTS_USE_TEMPLATE
template<>
struct Sum<4>
{
  static const int value = 4 + Sum<3>::value;
};

#endif
/* First instantiated from: insights.cpp:5 */
#ifdef INSIGHTS_USE_TEMPLATE
template<>
struct Sum<3>
{
  static const int value = 3 + Sum<2>::value;
};

// Sum<2>, Sum<1>, Sum<0>

int main()
{
  std::cout.operator<<(Sum<5>::value).operator<<(std::endl);
  return 0;
}
```

Templates

Sum<**8192**>::value → ~33,000 Lines!

Templates

- Template Instantiation is a type of monomorphization
- Trade build time (and binary size) for faster run times

Templates - Recommendations

- Consider whether you need to use Template Meta Programming
  - **constexpr** and **consteval** can go a long way
  - Is there a way to more directly express intent to the compiler?

- Re-evaluate your API
  - Do you need to be generic over that extra type?
  - Can you eliminate recursion?

Single File

Project Level

Higher Order

ClangBuildAnalyzer

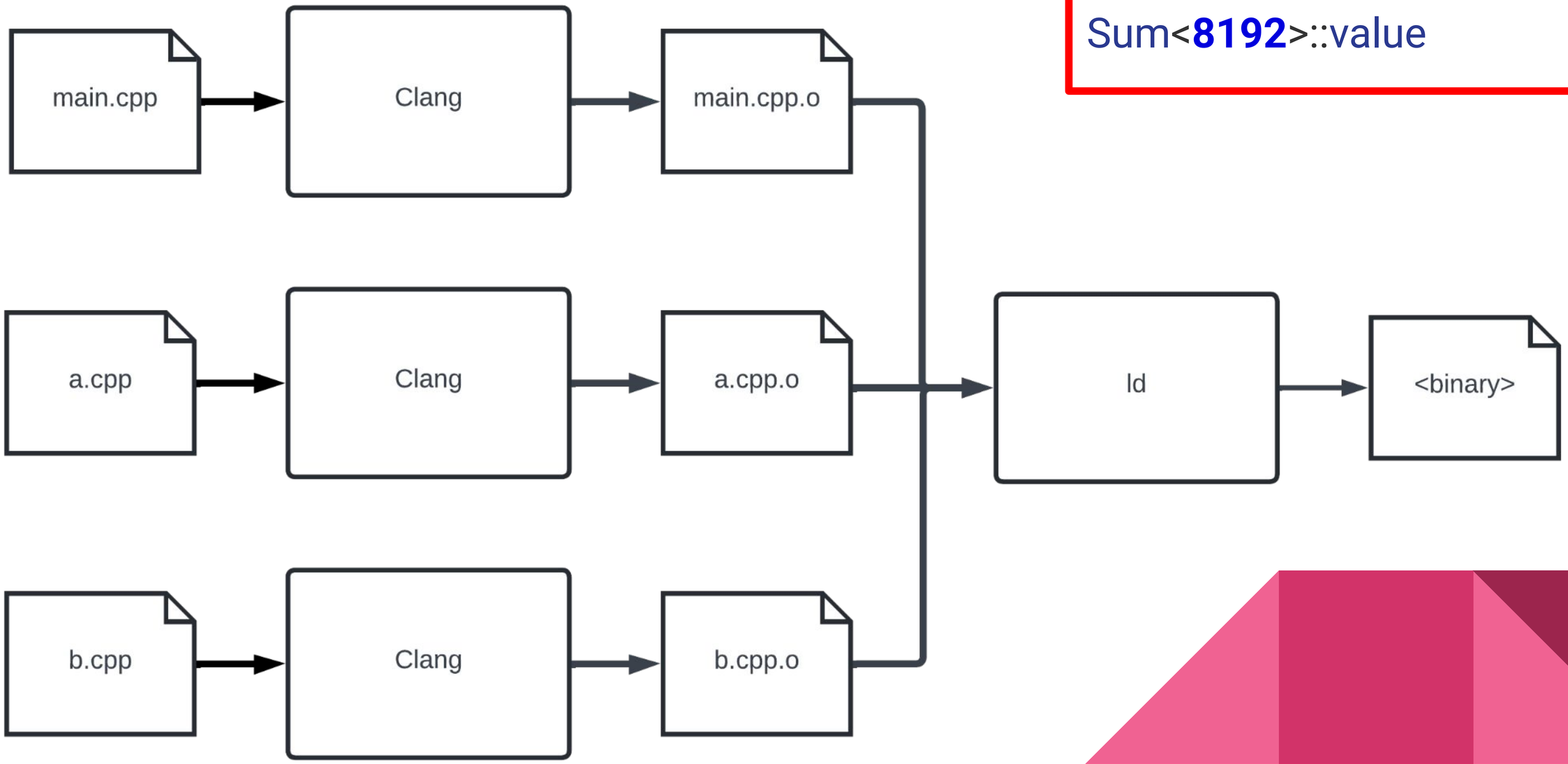
- High level overview of hot spots
  - Files that took longest to parse
  - Templates that took longest to instantiate
  - Functions that took longest to compile
  - **Expensive headers (with include chains!)**
- Support for incremental builds

Translation Units

```
// templates.hpp

#include <iostream>

template <int N>
struct Sum {
    static const int value = N + Sum<N - 1>::value;
};

template <>
struct Sum<0> {
    static const int value = 0;
};
```

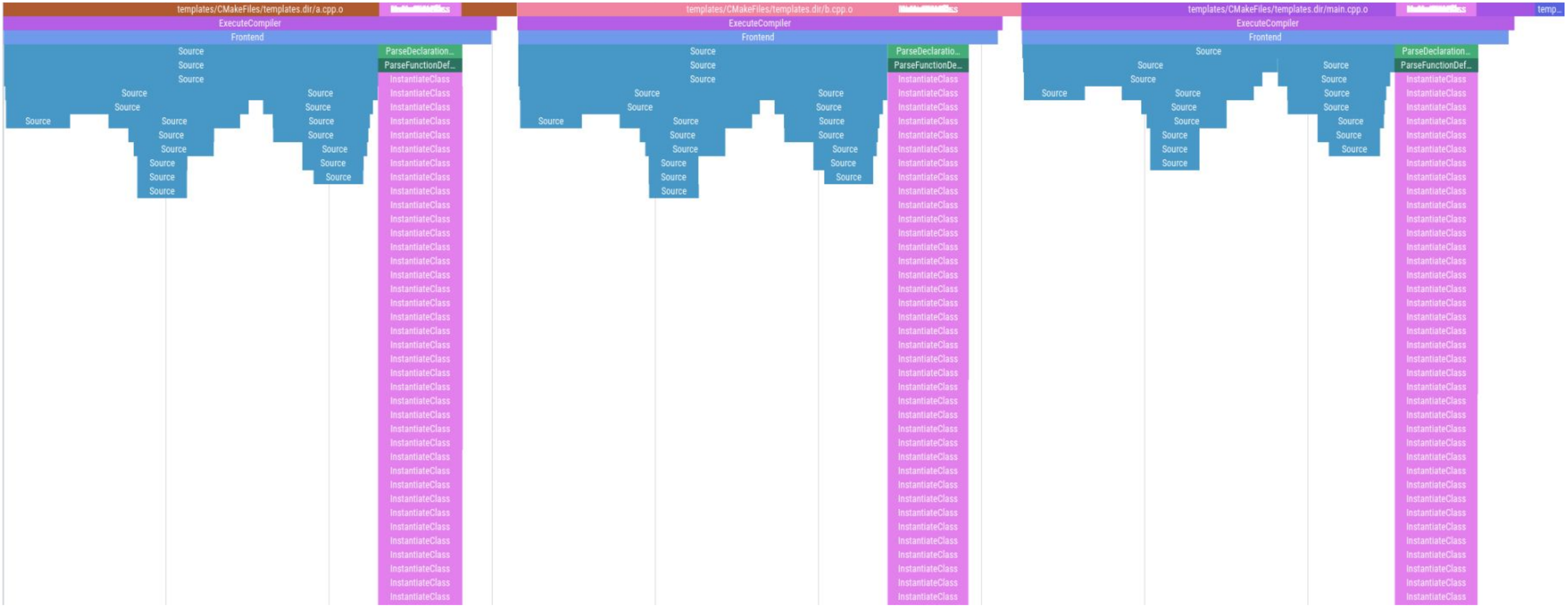
Translation Units

```
#include “templates.hpp”

Sum<8192>::value
```



Translation Units

≈1.91 seconds

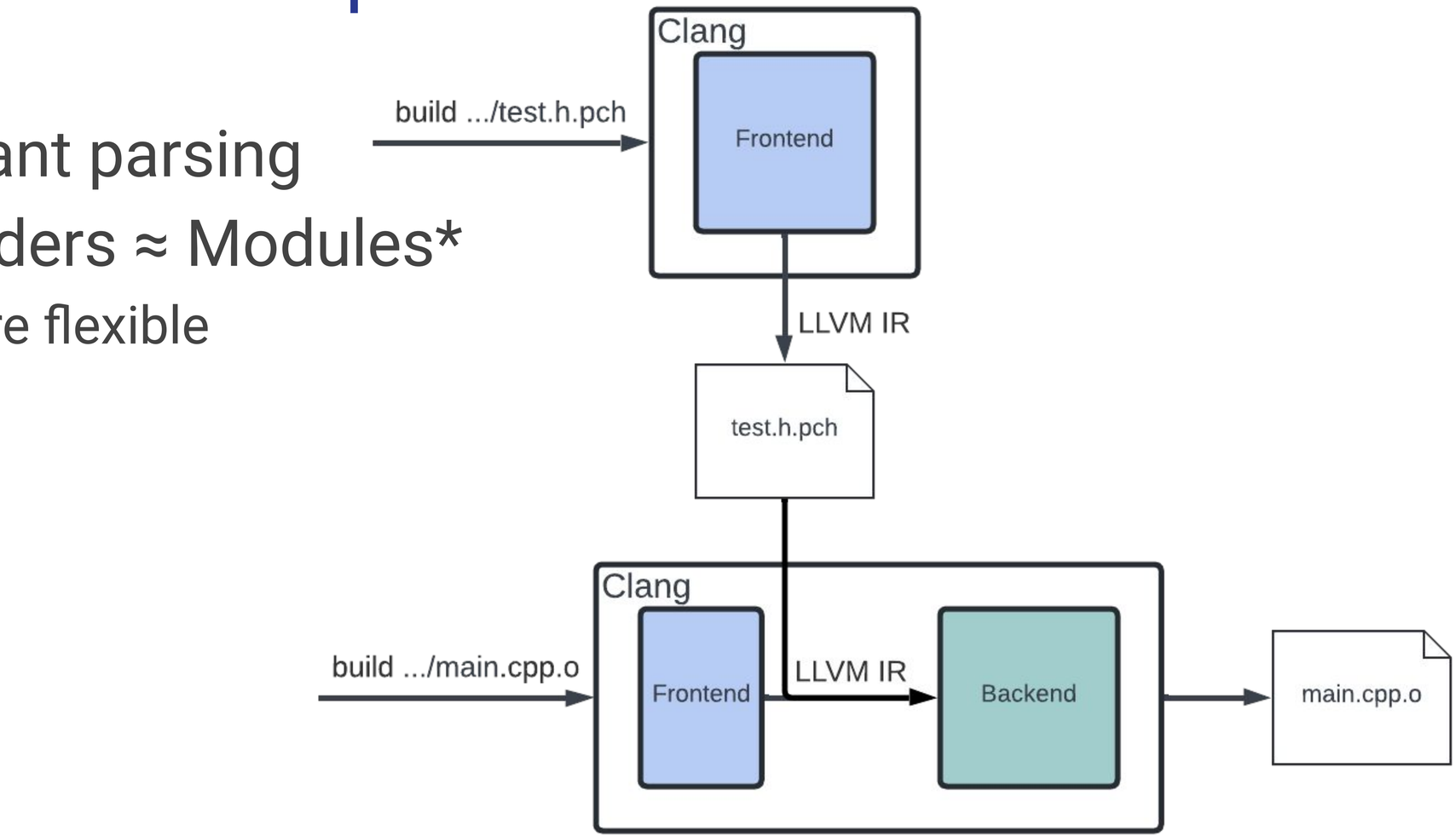


Translation Units

- #include and **template** costs are per translation unit
  - #pragma once and header guards won’t save you

Wait, Why Do Modules Help?

- Eliminate redundant parsing
- Precompiled Headers ≈ Modules*
  - Modules are more flexible



Using Precompiled Headers

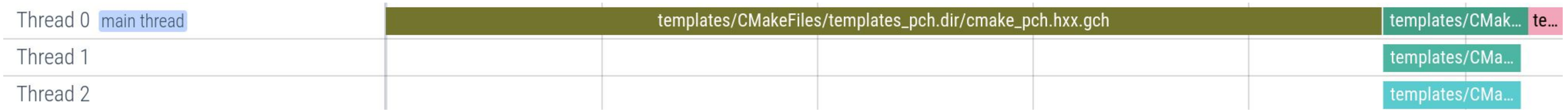
≈1.16 seconds

Translation Units - Recommendations

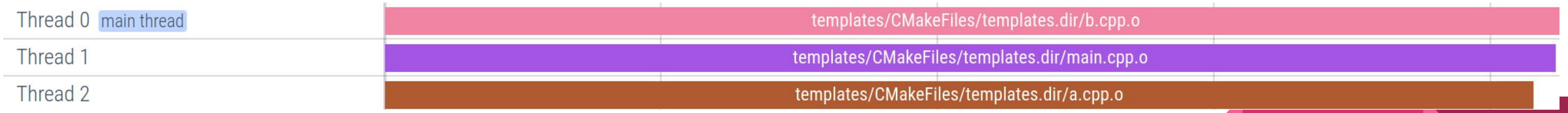
- Precompiled headers
  - Not a standard feature
  - target_precompile_headers
- Modules
  - Standard feature (as of C++20)!
  - Few compilers have full support
  - cmake-cxxmodules

Sometimes, Duplicating Work is Faster

1089ms

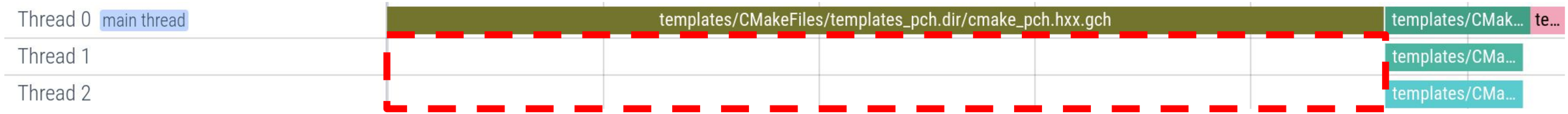


890ms



But…

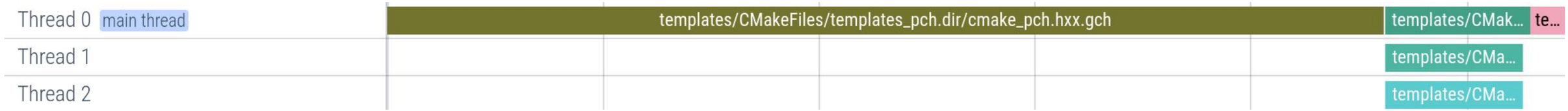
1089ms


Thread 0
main thread
Thread 1
Thread 2
templates/CMakeFiles/templates_pch.dir/cmake_pch.hxx.gch
templates/CMak…
te…
templates/CMa…
templates/CMa…


It's Free Real Estate

Free House for You, Jim!

1089ms



1137ms

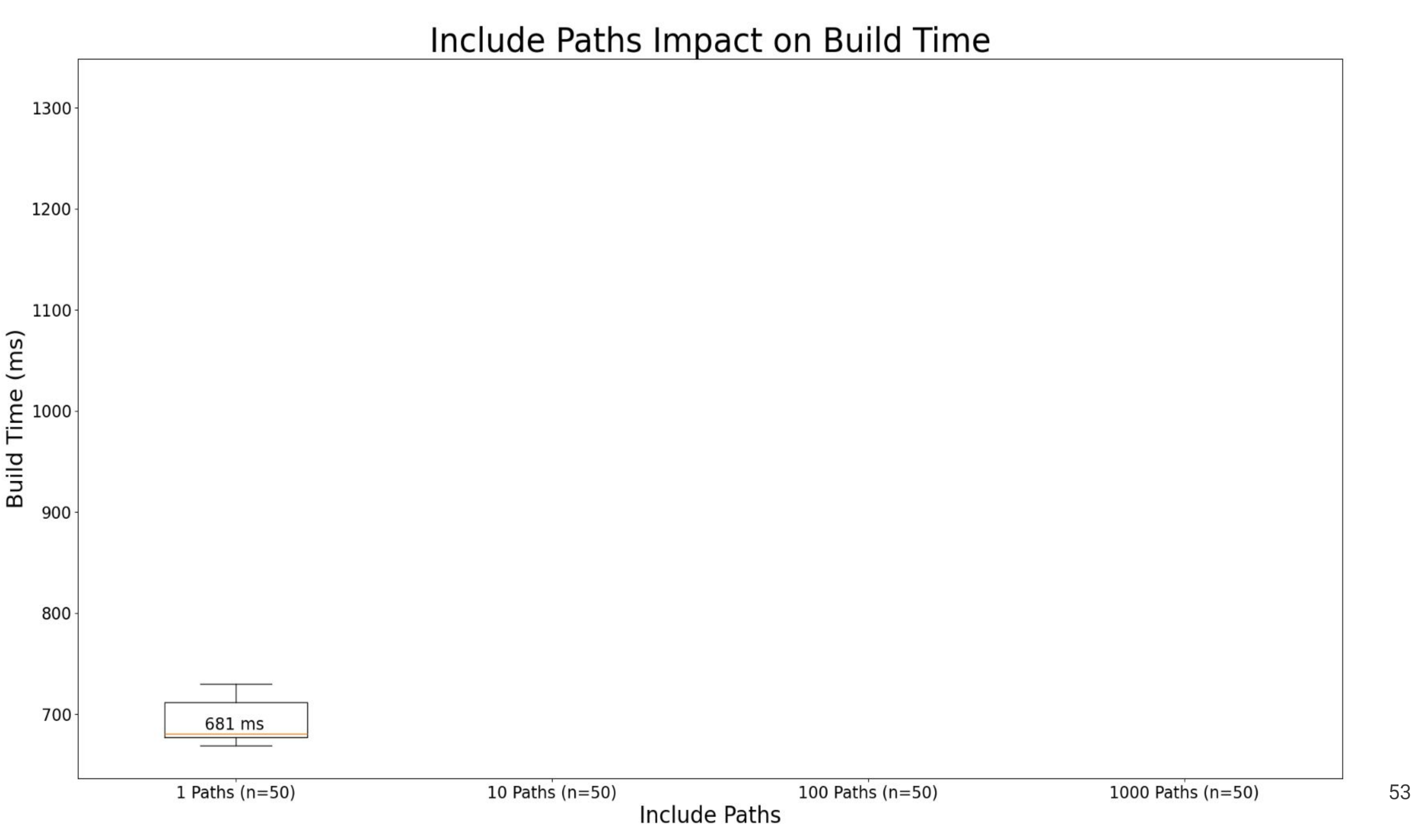
Poor Dependency Management

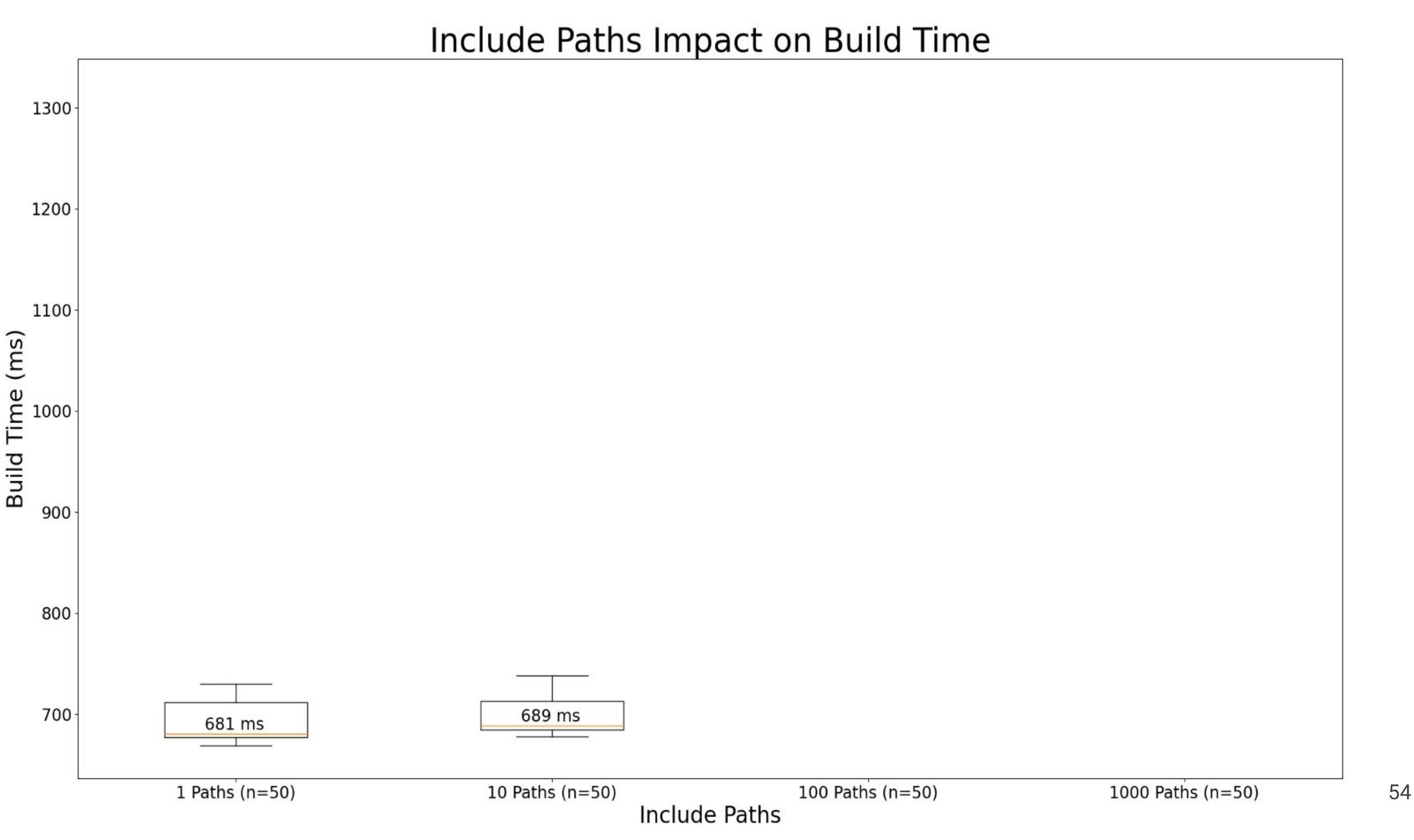
- Builds should be purely functional
  - Single-core and parallel builds should just work
- Avoid large dependency bottlenecks
  - Can force the build to be synchronous
  - Especially important when generating code (i.e., Protobuf)
- Prefer smaller targets
- **Explicitly expressed dependencies enables efficient hardware utilization while maintaining build correctness**

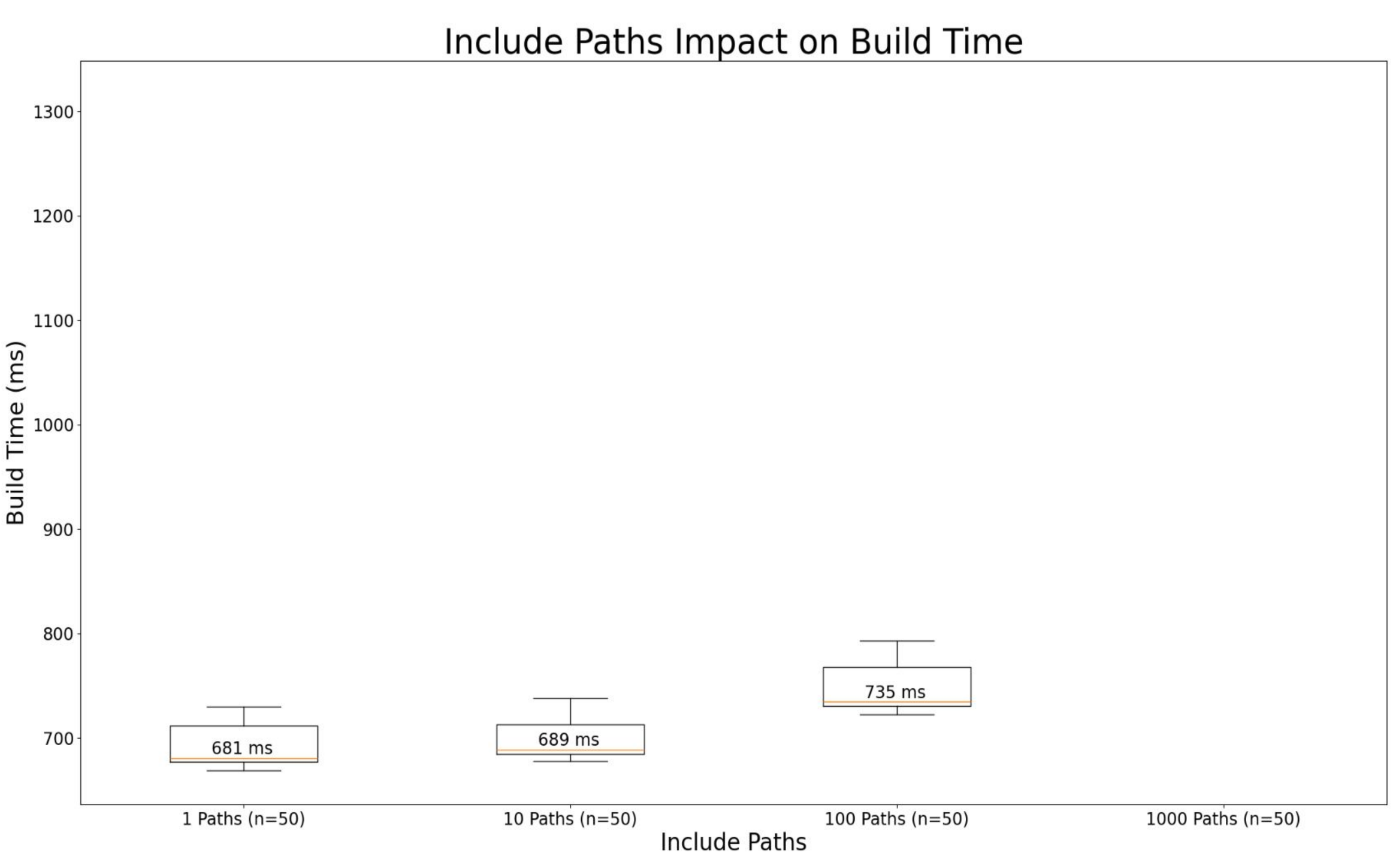
Compiler Include Paths

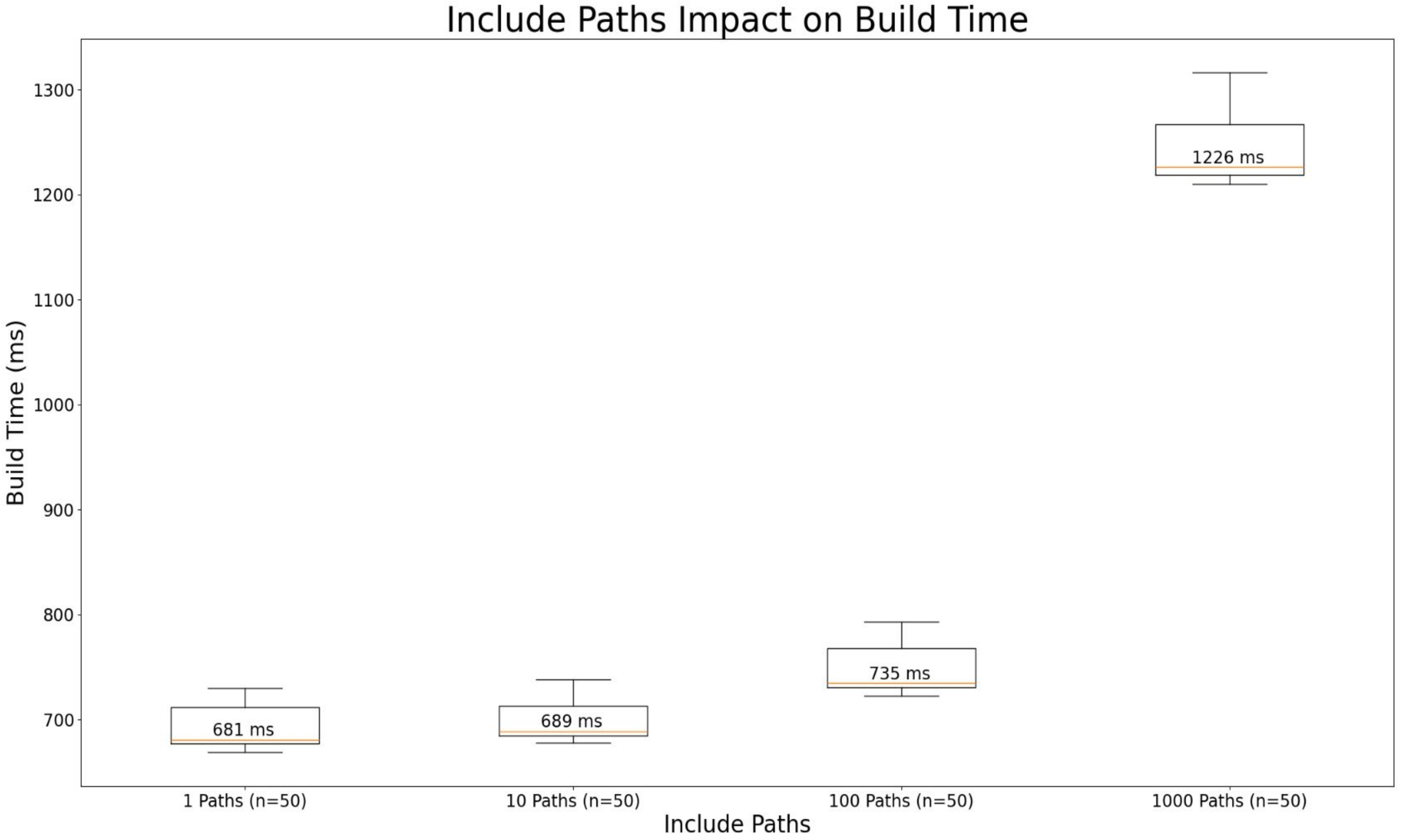
```
#include <iostream>

int main() {
        std::cout << "Hello, world!" << std::endl;
}
```

```
clang -I… -stdlib=libc++ main.cpp
```


Include Paths Impact on Build Time
Build Time (ms)
1300
1200
1100
1000
900
800
700
681 ms
1 Paths (n=50)
10 Paths (n=50)
100 Paths (n=50)
1000 Paths (n=50)
Include Paths

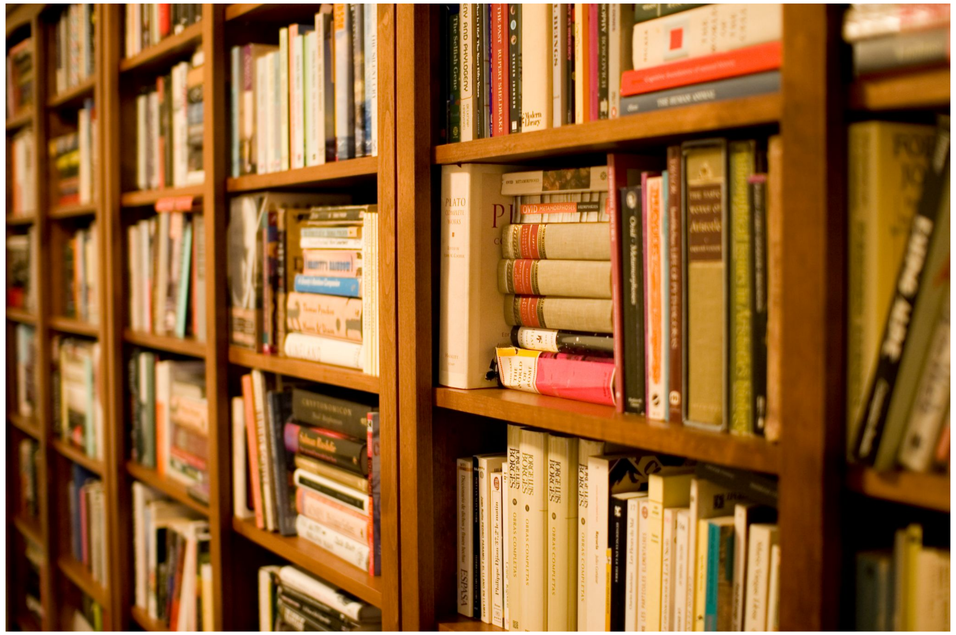

Include Paths Impact on Build Time
Build Time (ms)
1300
1200
1100
1000
900
800
700
681 ms
689 ms
1 Paths (n=50)
10 Paths (n=50)
100 Paths (n=50)
1000 Paths (n=50)
Include Paths


Include Paths Impact on Build Time
Build Time (ms)
1300
1200
1100
1000
900
800
700
681 ms
689 ms
735 ms
1 Paths (n=50)
10 Paths (n=50)
100 Paths (n=50)
1000 Paths (n=50)
Include Paths


Include Paths Impact on Build Time
Build Time (ms)
1300
1200
1100
1000
900
800
700
1226 ms
735 ms
689 ms
681 ms
1 Paths (n=50)
10 Paths (n=50)
100 Paths (n=50)
1000 Paths (n=50)
Include Paths

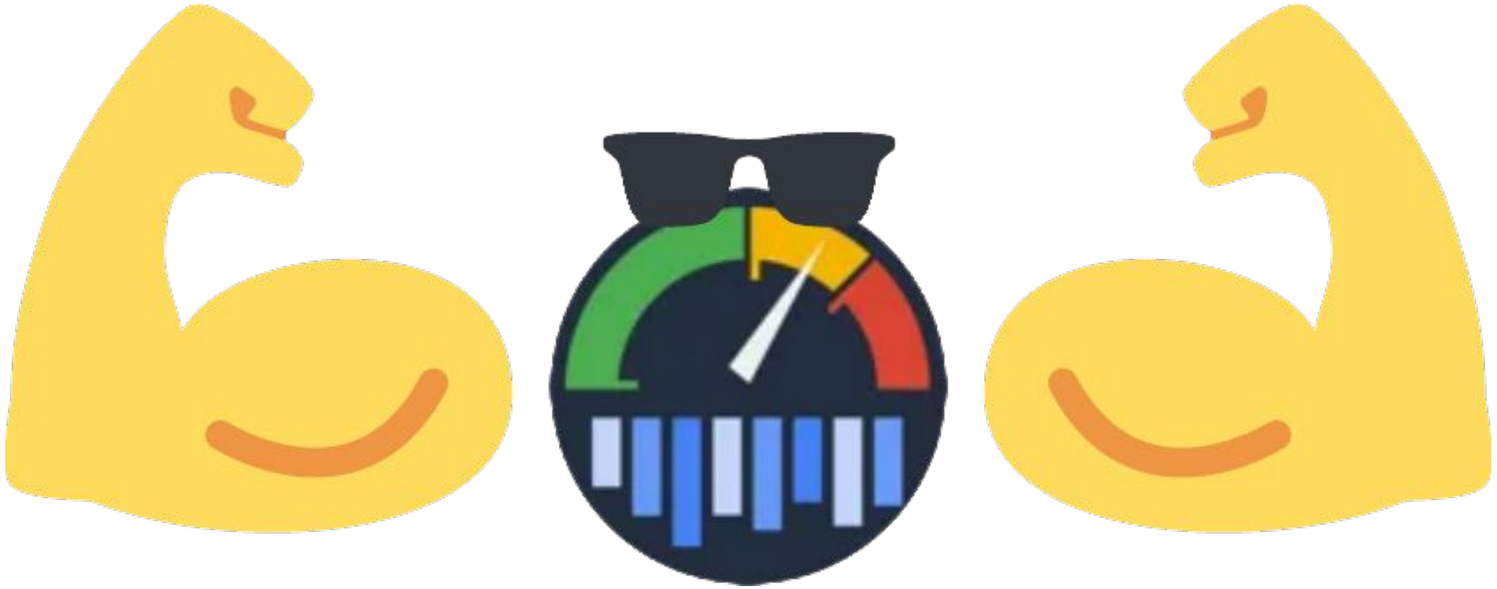
Compiler Include Paths

- Interacting with the filesystem is not free
- Caches abound but still expensive at scale
- Scales per translation unit



Querying Traces

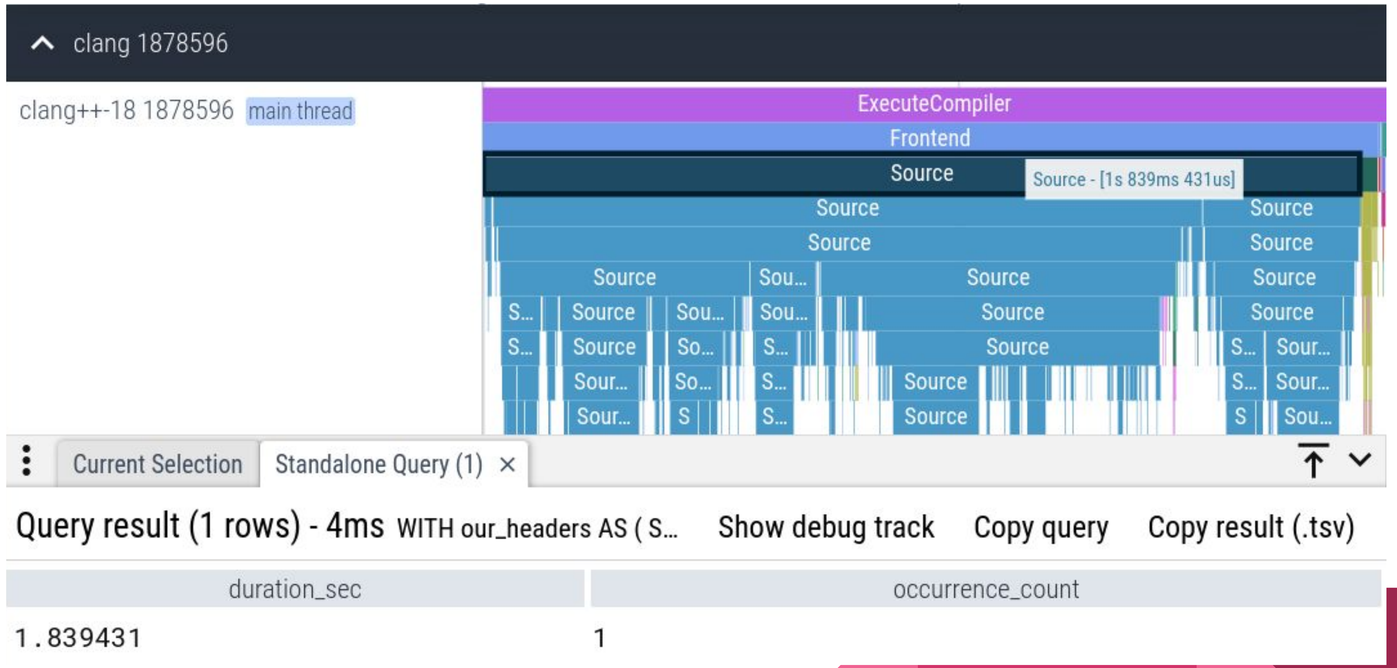
- Perfetto’s superpower
- Answer complex questions like
  - “How much of the build is spent including this specific header?”
  - “What is the total impact on our build by using this external library?”



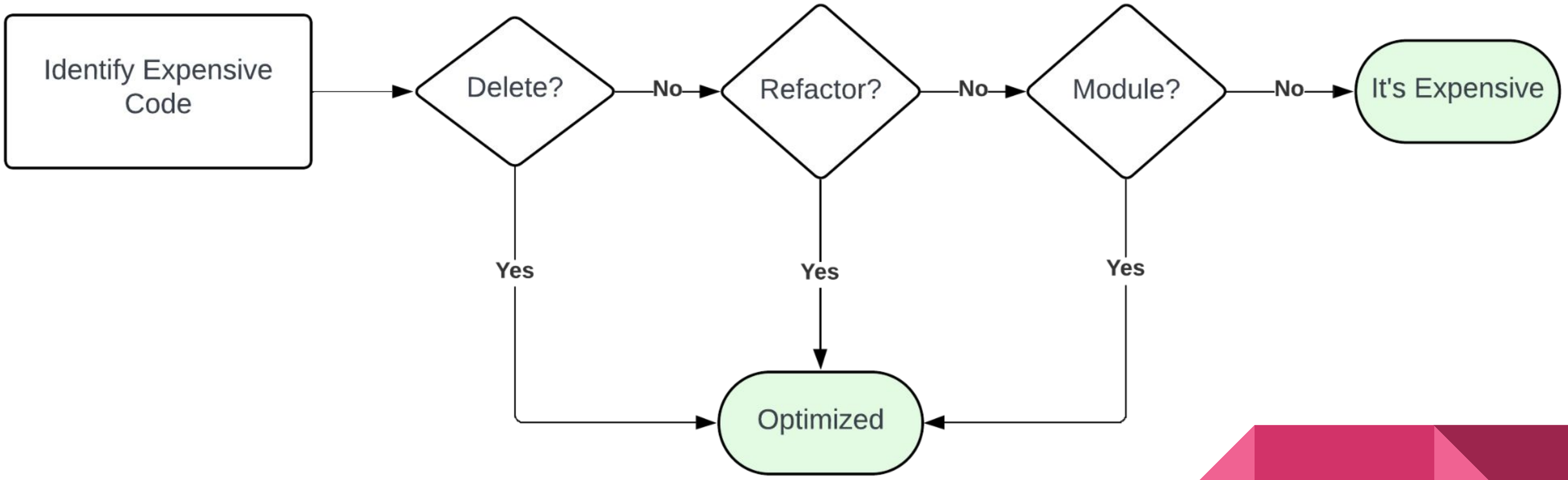
Querying Traces

“How much of the build is spent including this specific header?”

```
WITH our_headers AS (
    SELECT
        DISTINCT arg_set_id,
        display_value
      FROM
        args
      WHERE
        KEY = 'args.detail'
        AND (display_value LIKE 'my_header.h')
) SELECT
      SUM (slice.dur) / 1e+9 AS duration_sec,
      COUNT (*) AS occurrence_count
    FROM
      our_headers JOIN slice
        ON our_headers.arg_set_id = slice.arg_set_id
```



Project Level Analysis



Is the Juice Worth the Squeeze?

- Optimizing build times is a lot of work
- Focus on low hanging fruit

Project Level

Higher Order

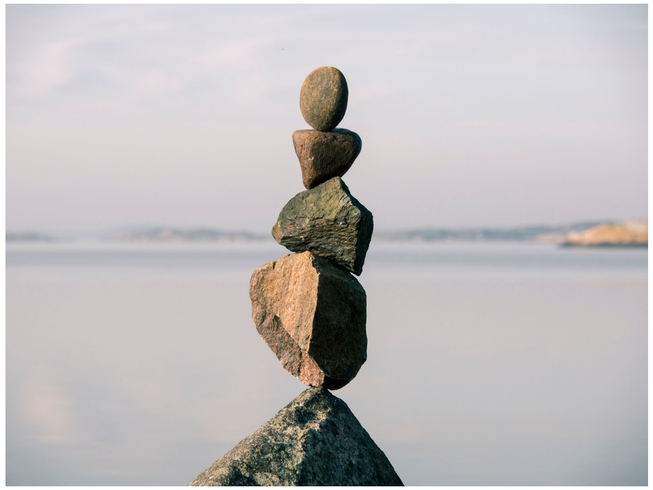
Takeaways

The Easy Stuff Is Gone, Now What?

- Do less work
  - Incremental build
  - Use a package manager
- Compiler caching
- Throw hardware at it
- Distributed builds

What Andy Giveth, Bill Taketh Away

- Andy and Bill’s Law
  - New software will always consume any increase in computing power that new hardware can provide
- Wirth's law
  - Software is getting slower more rapidly than hardware is becoming faster



Higher Order

Takeaways

Questions?

Takeaways

- Profile your builds
- Compilation requires parsing which isn’t free
- Better hardware won’t always save us from ourselves
- We are all stewards of our build times
- Give a compiler engineer a hug

Thank You!

https://github.com/maspe36/WhyIsMyBuildSoSlow

https://www.linkedin.com/in/sam-privett/

careers.jnj.com

Takeaways

Questions?

GCC Profiling

- Externis, GCC Plugin “similar” to -ftime-trace
- Write a script to parse -ftime-report

Profile Memory Consumption

Really nice presentation on this from an LLVM Performance Workshop

https://llvm.org/devmtg/2023-02-25/slides/basic-memory-profiler.pdf

TL;DR: Nothing yet, but maybe soon