Peter Bindels (@dascandy42) TomTom International

How to understand *million-line* C++ projects

```
#include <cstdio>


int main() {
  printf("%d\n", __LINE__);
  … copy-pasted 6999998 times …
  printf("%d\n", __LINE__);
}
```

Long runtime / leadtime

Many disjoint domains, or a very large domain

Many developers, multi-team

(Often) Multi-site development

Complicated code

Culture

Lots of legacy.

Long time buildup of hard to find minor issues

- Accidentally including too much
  - (we measured 2600 headers indirectly including 100kLOC+ - not counting platform and STL headers)
- Stale dependency
- Code that’s no longer used anywhere

Code that conforms to outdated standards

Lots of code not understood by current team(s)

- Causes duplicate code
- Causes similar but subtly different behavior

The HR/Management solution

- Keep people onboard longer
- Make people responsible for code
- Make people document their code more
- Have standard ways to do things
- Adhere to coding standards
- Have people approve commits by seniors
- Do lots of code reviews
- Use COTS solutions


THIS IS FINE.

It's not just you

Dare to make a change and revert it

Code is a *cost*

Less code is less future cost

```
float f (float x){
    float xhalf = 0.5f*x;
    int i = *(int*)&x;
    i = 0x5f3759df - (i>>1);
    x = *(float*)&i;
    x = x*(1.5f - xhalf*x*x);
    return x;
}
```

Test coverage only means

"This code was run while running these tests”

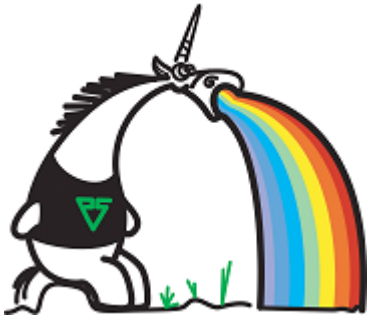
Tools are your *friends*

Compile time errors

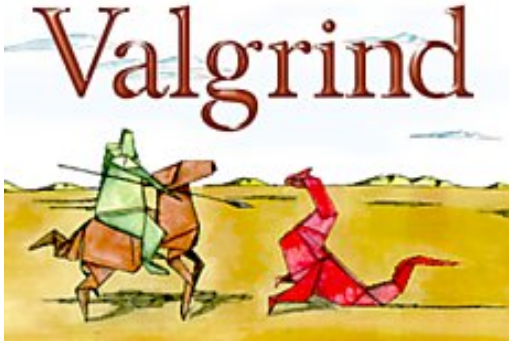
Static analysis

Warnings

Runtime checks



coverity™



Let’s make things better

The things that helped you to understand code are the things you should be making it now.

Have less code

Replace custom code with standard c++

Remove little-used libraries

Find and kill dead code / headers

Test your requirements → What does the rest do?

Have navigable code

Given a logical function → Only one place it could be

Given the code → You know what component it is

Given the name → You know what it is and does

Have readable code

People spend more time on your code

Than your compiler ever will

Have stable code

DRY

- **Don’t Repeat Yourself**
- **Anything will get out of sync**
- **Remove duplicates**
  - Templates
  - Reuse
  - Code generation
- **This is *much* worse in large projects**

Have well-defined components

Small

Have well-defined components

Easy to use

Have well-defined components

Responsibilities

- Talk to colleagues
- Read documentation
- Reverse design the component
- Guess

Have well-defined components

Dependencies

- Every include goes somewhere
- You also tell your compiler
- You also tell your build tools

Have well-defined components

Dependencies

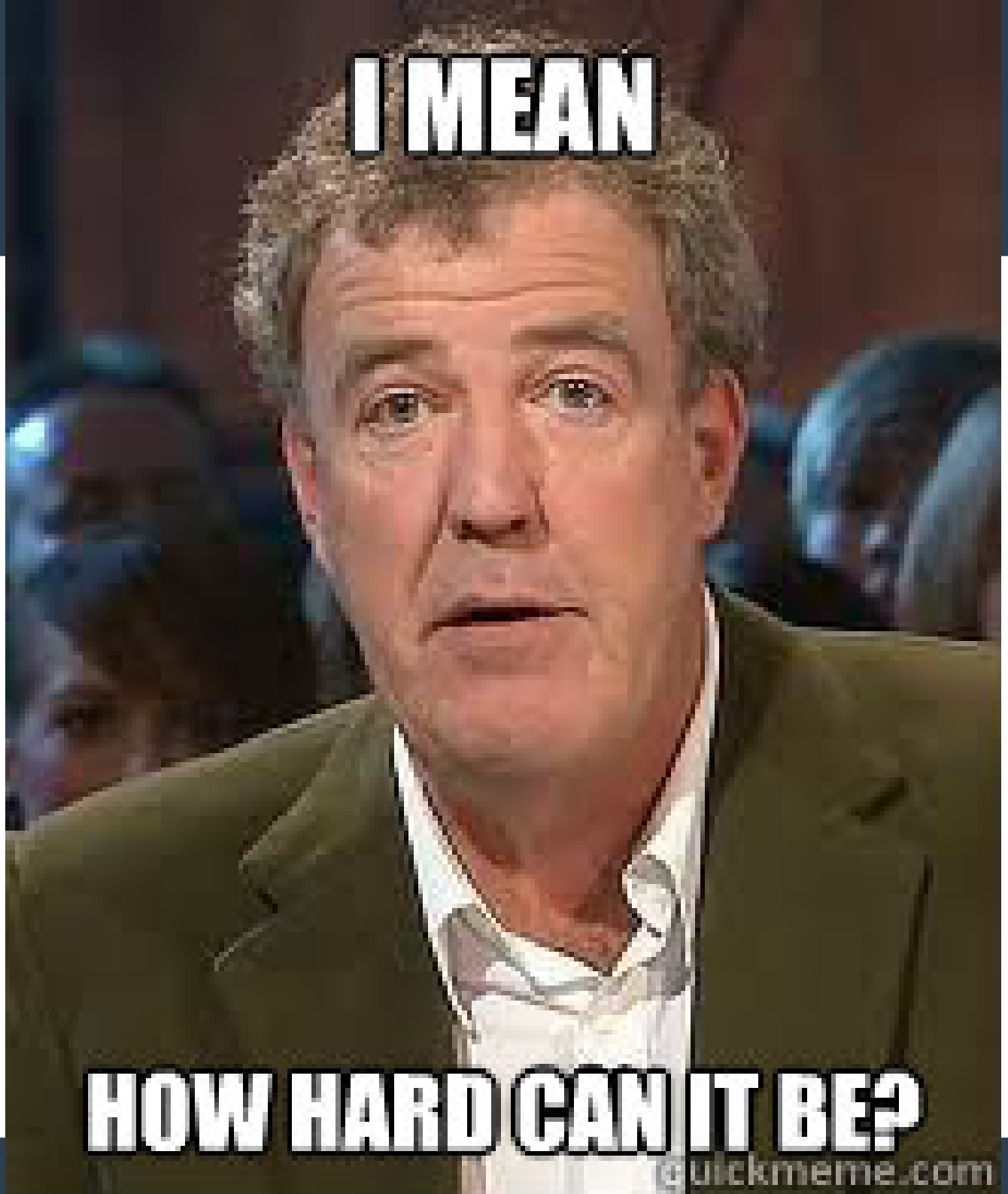
- Every include goes somewhere
- You **also** tell your compiler
- You **also** tell your build tools

Department of Redundancy Department

You *should* be able to just guess this

From source code.

Automatically.


I MEAN
HOW HARD CAN IT BE?
quickmeme.com

Test environment

- **TomTom proprietary code base**
  - 7M LoC
  - 500 components
  - Lot of test coverage
- **2012 4-core laptop with sufficient memory**
- **Expectancy: Should be able to match C preprocessor in speed**

Attempt #1: Shellscripts

See if it works

Create .dot file with dependencies

It works!

Runtime	7200s (2 hours)
Cycles per LoC	2880000

Attempt #1.5: Shellscripts + C

Make it usable

Create .dot file with dependencies

Create .dot file with only cycles

Create include lookup table for validation

It’s usable

Runtime	1200s (20 minutes)
Cycles per LoC	480000
Speedup	6x

Attempt #2: Python rewrite

- **Make it fast enough for CI use**
- **Output delta cycles**
- **It can go into CI**

Runtime	300s (5 minutes)
Cycles per LoC	120000
Speedup	24x

Attempt #3: C++ rewrite

- **Make it fast enough for interactive use**
- **Add interactive queries**
- **Add component info**

- **It’s much faster, but not quite there…**

Runtime	5s
Cycles per LoC	2000
Speedup	1440x

Attempt #3.1: C++ optimizing

- **Make it fast enough for interactive use**
- **Add generating CmakeLists.txt out of info**
- **Add more query commands**

- **Fast enough to just run every time**

Runtime	2s
Cycles per LoC	800
Speedup	3600x

Attempt #3.2: C++ optimizing

- See how fast it can become
- Just for fun, really

- Fast enough for any code base

Runtime	1.1s
Cycles per LoC	440
Speedup	7000x

What can you get out of #include?

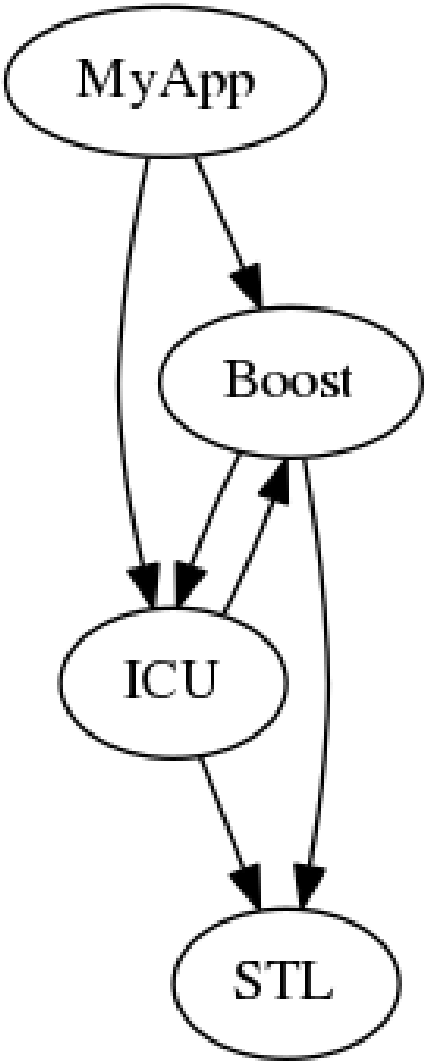
- **Your public interface**
- **Dependencies between components**
  - Public and private
- **Include paths**
  - So that all the includes that should end up here will end up working

- **Dependencies show “users” and “providers”**
  - Providers create its function
  - Users use its function
- **Responsibility is created from its providers, and used by its users**

  → Derive more responsibilities

What can you get out of dependencies?

- **Highlevel versus lowlevel components**
- **Layering (and layering violations)**
- **Cycles**
- **Numbers (for management)**



What’s wrong with a cycle?

- **Any cyclic dependency is a component, split along a line that results in co-dependent parts**
  - It behaves like a single component
  - This is **transitive**.
- **Small components with cycles are actually large components**
- **Large components are called “Monoliths”**


WENGER

Help in analyzing them

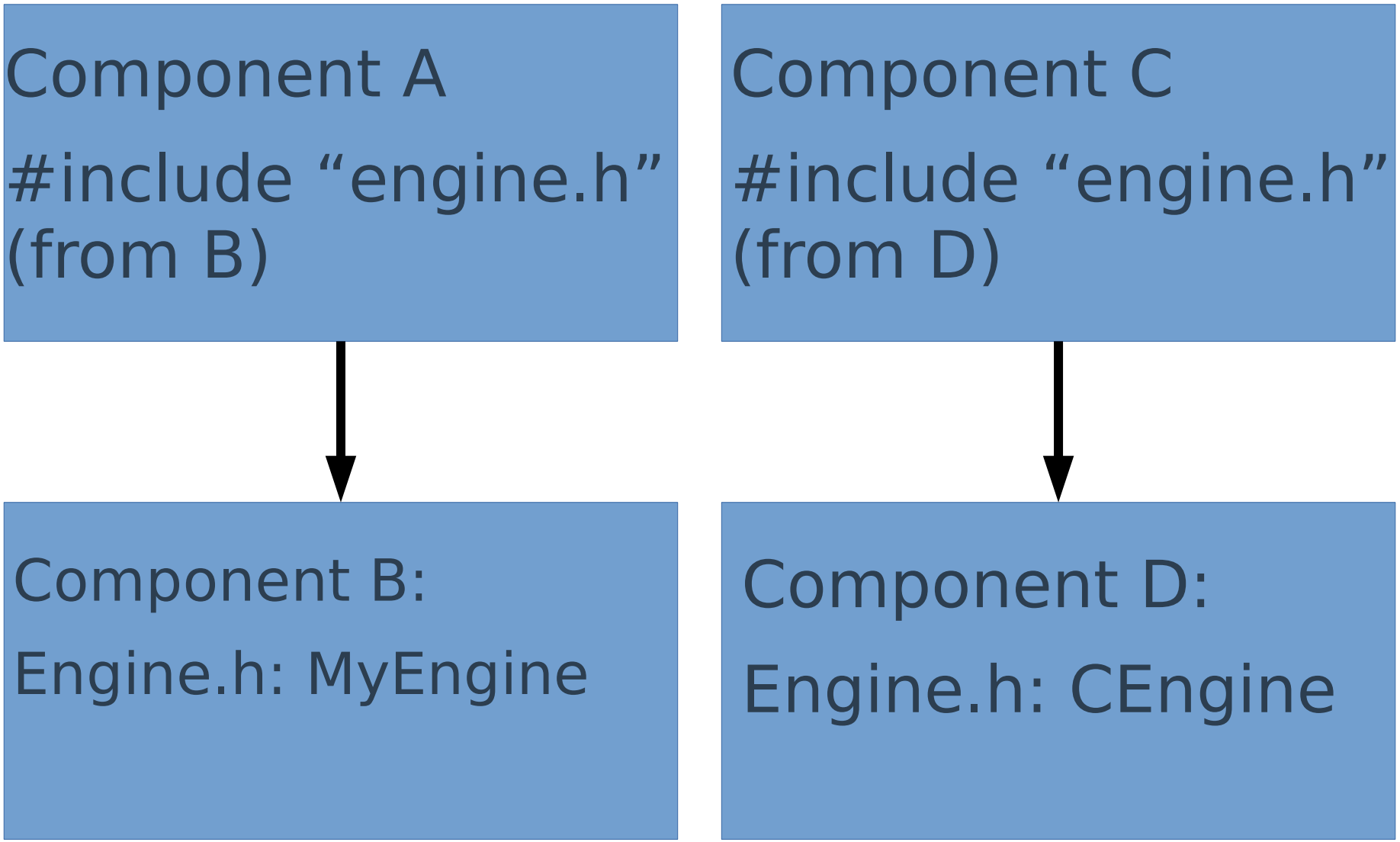
- **Get all cycles in the code base**
- **Find the shortest path from A to B and explain each link**
  - Or check that no path exists from A to B
- **Ignore one file, check what the effect is**
- **List “all the cycles”. Normally not useful**

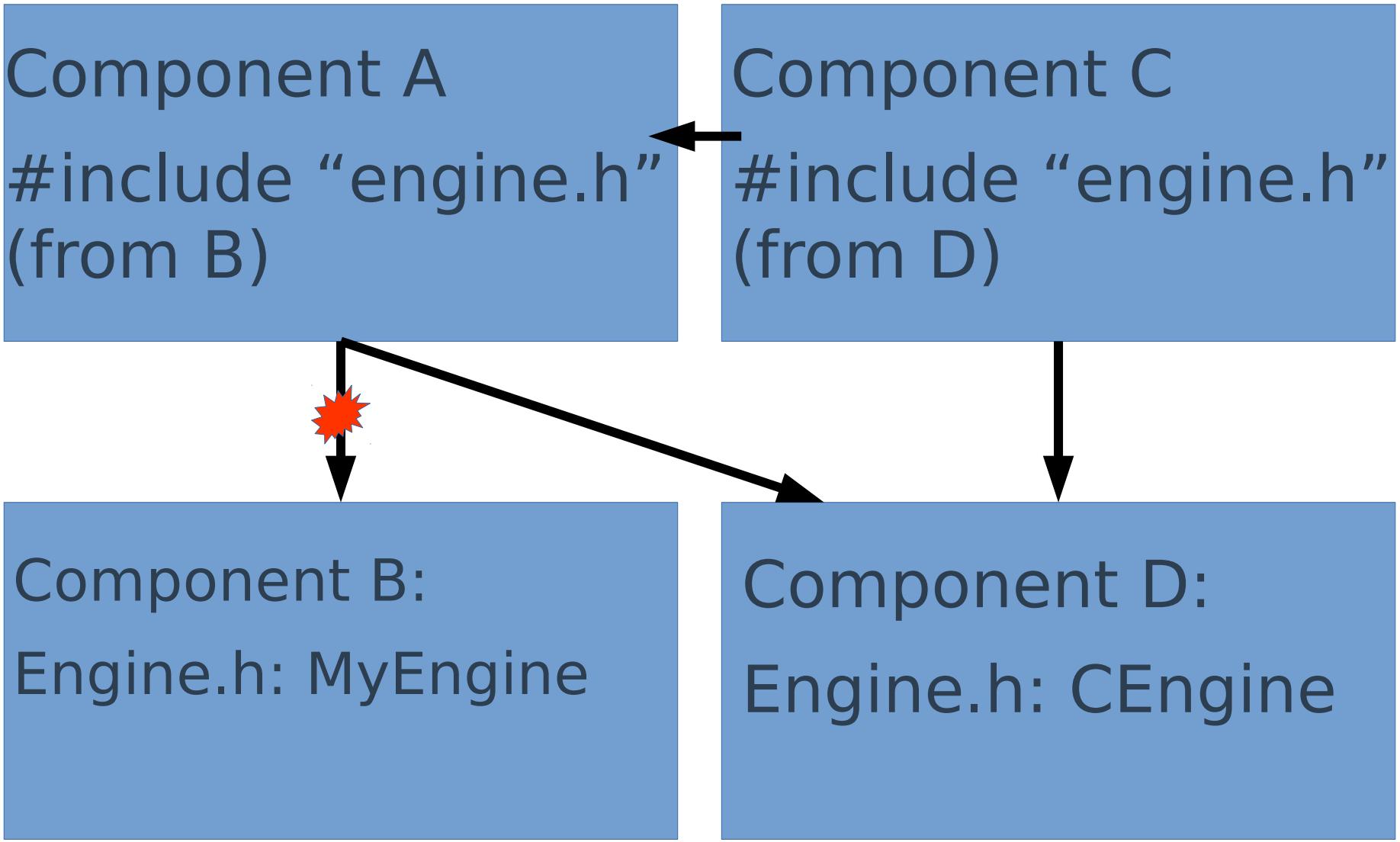
What else can you get?

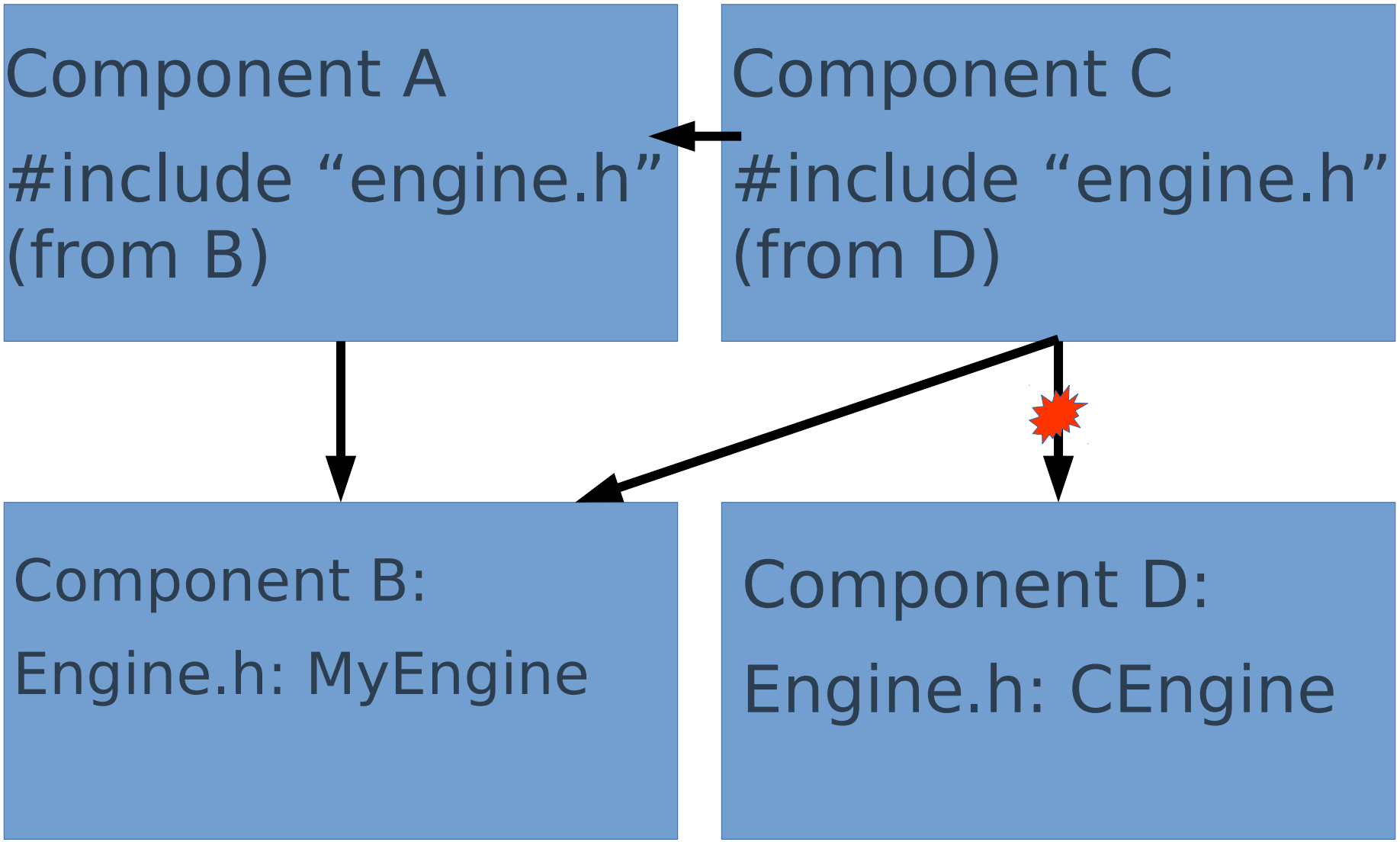
- **Headers that result in a lot of code being included**
  - Make your build faster
- **Very small or large files/components**
- **Automatically regenerate CMakeLists**
  - Insofar as possible

What else can you get?

- **Never-included headers**
- **Code that is not in any component**
- **Include statements that map to two or more headers**


Component A
#include “engine.h”
(from B)
Component C
#include “engine.h”
(from D)
Component B:
Engine.h: MyEngine
Component D:
Engine.h: CEngine


Component A
#include “engine.h”
(from B)
Component C
#include “engine.h”
(from D)
Component B:
Engine.h: MyEngine
Component D:
Engine.h: CEngine


Component A
#include “engine.h”
(from B)
Component C
#include “engine.h”
(from D)
Component B:
Engine.h: MyEngine
Component D:
Engine.h: CEngine

Coherent

Overdependent...

Encapsulation

Good public dependencies:

- **Value types**

Good private dependencies:

- **Implementation detail services**
- **Generic services (logging, caching)**

Limit the amount

Use at TomTom

- **On every checkin, we detect cycles**
- **Developers use the tool to autogenerate 55% of the cmakelists (out of 1400)**
  - We estimate that 80% could be, eventually
- **Tool is regularly used for health checking**
- **Many developers actively use it to counter dependency and maintainability problems**

TomTom International

- **Come to our stand!**
- **Actively hiring in Eindhoven, Berlin, Amsterdam and Łódź**

- **Thanks for listening!**
- **Feel free to try the tool:**

  https://github.com/tomtom-international/cpp-dependencies
- **Visit our stand, come have a chat**