

# Dependency management in C++

Xavier Bonaventura

BMW AG.

 xbonaventurab

 limdor

 limdor

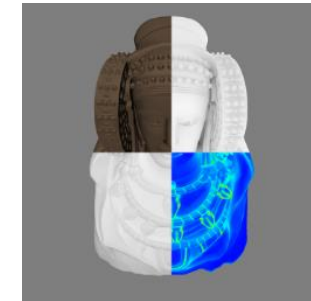
 xavierbonaventura

code::dive 2019 – Wrocław, Poland – 20<sup>th</sup> November 2019

# About me



- I...
  - studied software engineering
  - created a 3D model visualization tool in C++ and OpenGL during my PhD <https://github.com/limdor/quoniam> (~10.000 LOC) (2010 - 2015)
  - moved to Munich to work in 2015
  - started attending the C++ User Group Munich (MUC++) to realize that I knew nothing about C++
  - decided to do this presentation about dependencies after 4 years attending to C++ meetups every month
  - remembered that I developed the 3D model visualization tool when I knew nothing about C++



# Goal

Awareness and better understanding of the dependencies in your project

# What will we see?

- Basic dependency concepts
- Difference between declaration and definition
- Building process
- Execution sequence of a process
- Implications of the design of a library
- Examples with code

# What are dependencies?

From Longman Dictionary of Contemporary English

**de·pen·dence** /dɪ'pendəns/ ●●○ (also **dependency**) **noun** [uncountable] 🔊 🔊

1 when you depend on the help and support of someone or something else in order to exist or be successful **OPP** **independence**

**dependence on/upon**

🔊 our dependence on oil as a source of energy

🔊 the financial dependency of some women on men

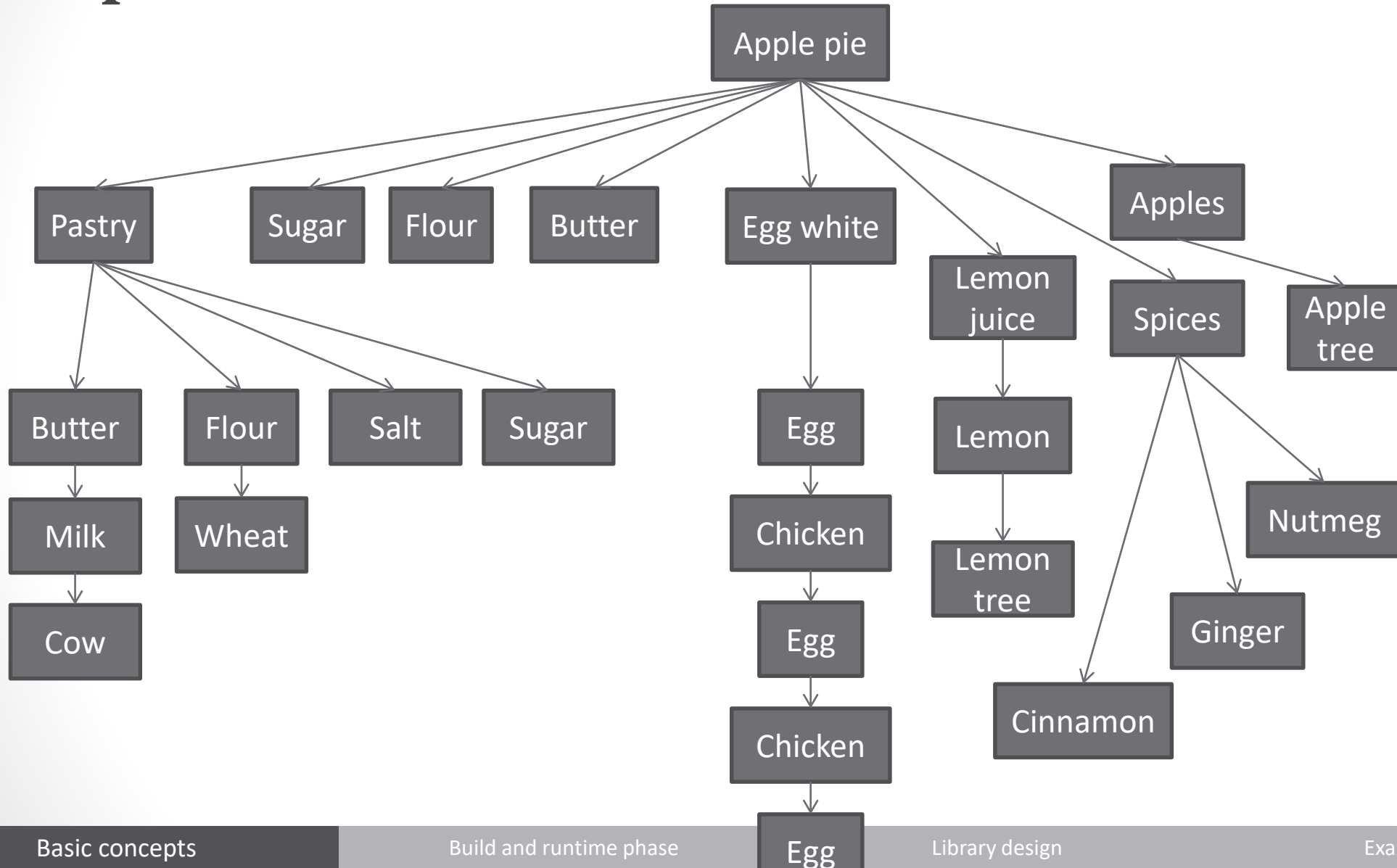
2 → **drug/alcohol dependence**

3 *technical* when one thing is strongly affected by another thing

**dependence of**

🔊 the mutual dependence of profit and growth

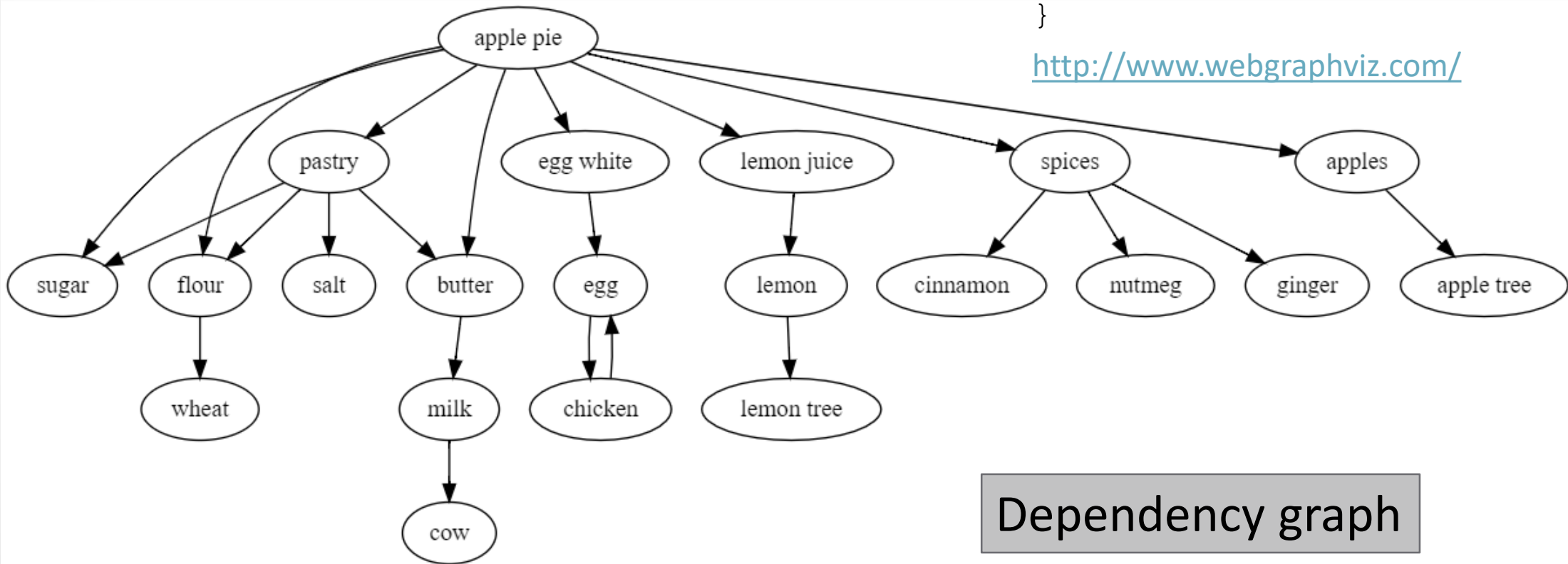
# Dependencies in real life



# Dependencies in real life

```
digraph G {  
  "apple pie" -> apples  
  apples -> "apple tree"  
  ...  
}
```

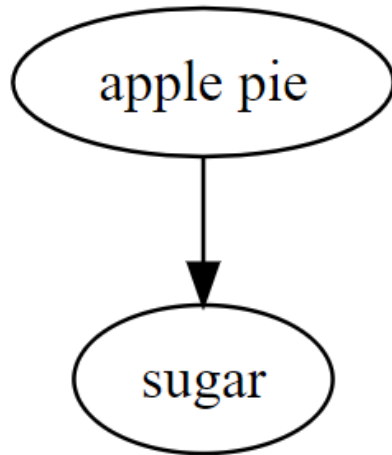
<http://www.webgraphviz.com/>



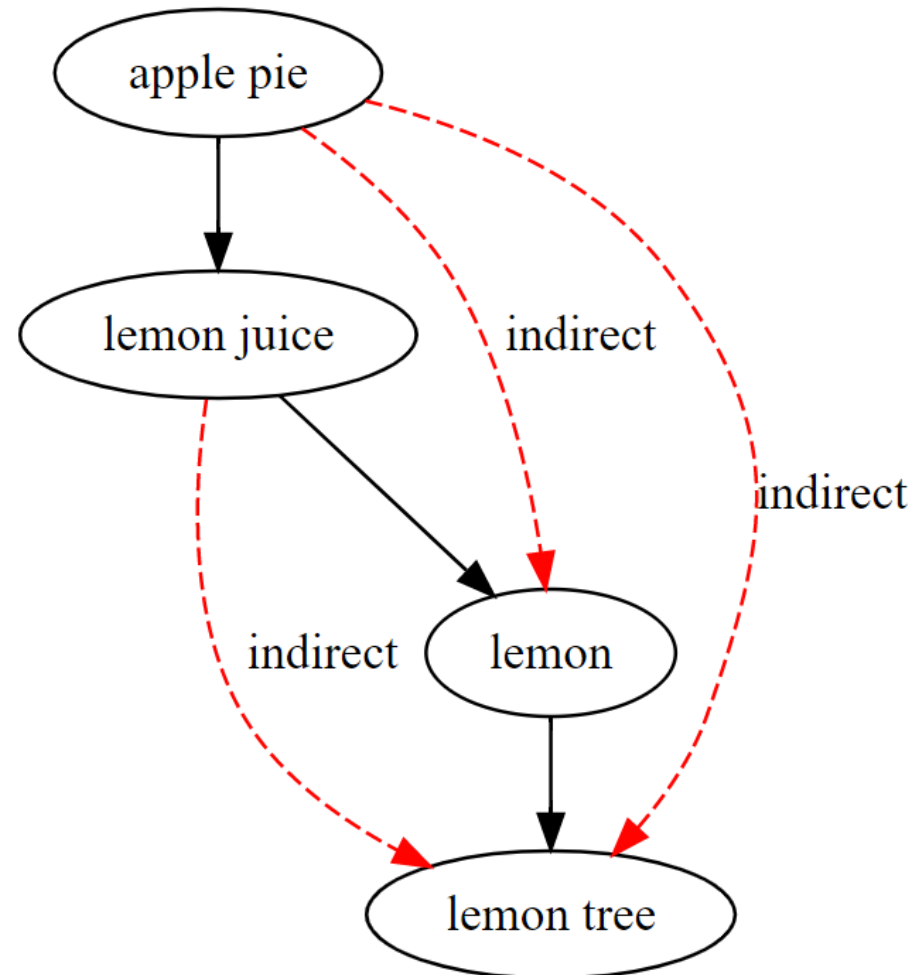
Dependency graph

# Direct vs indirect

- Direct dependency



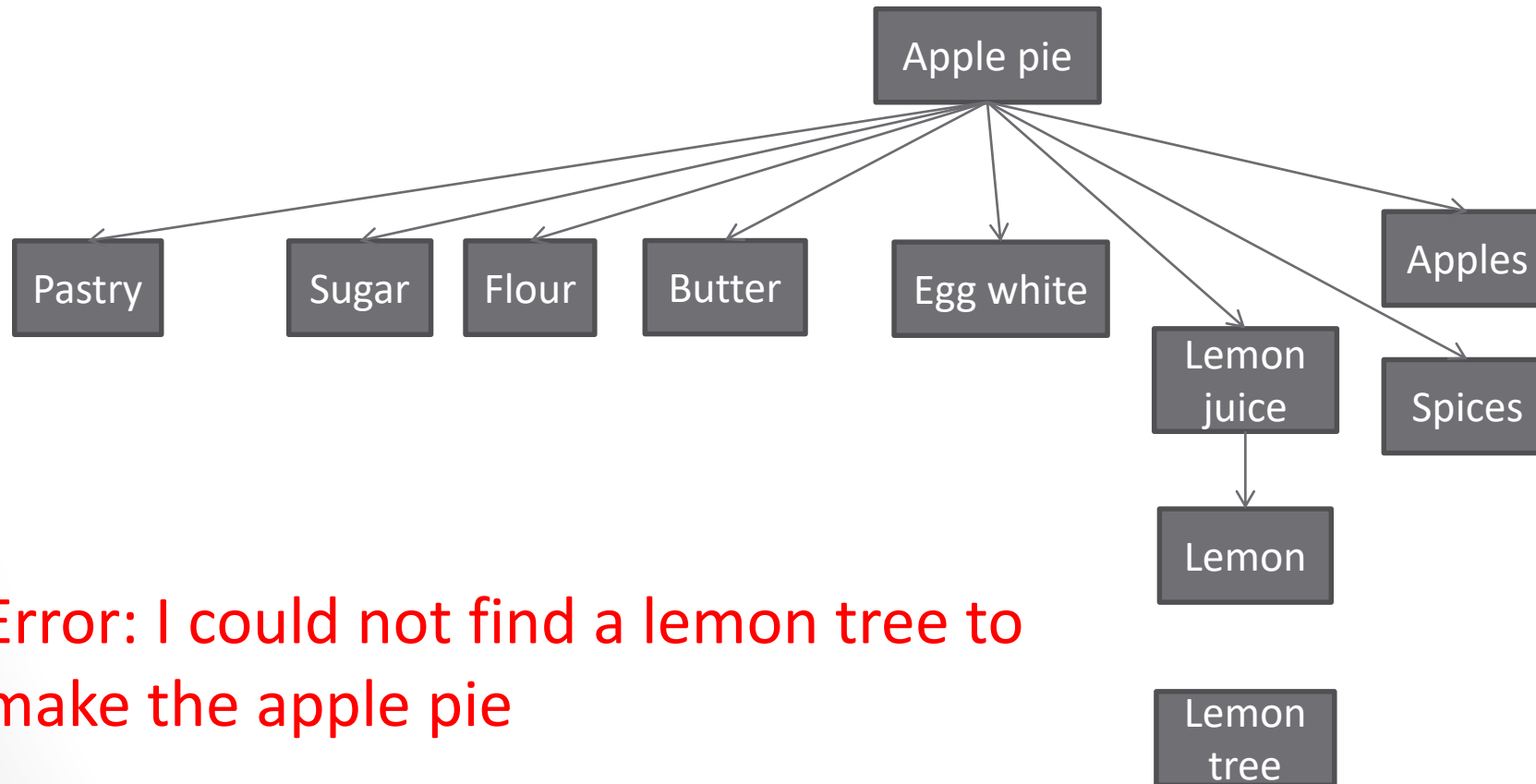
- Indirect or transitive dependency





# Direct vs indirect

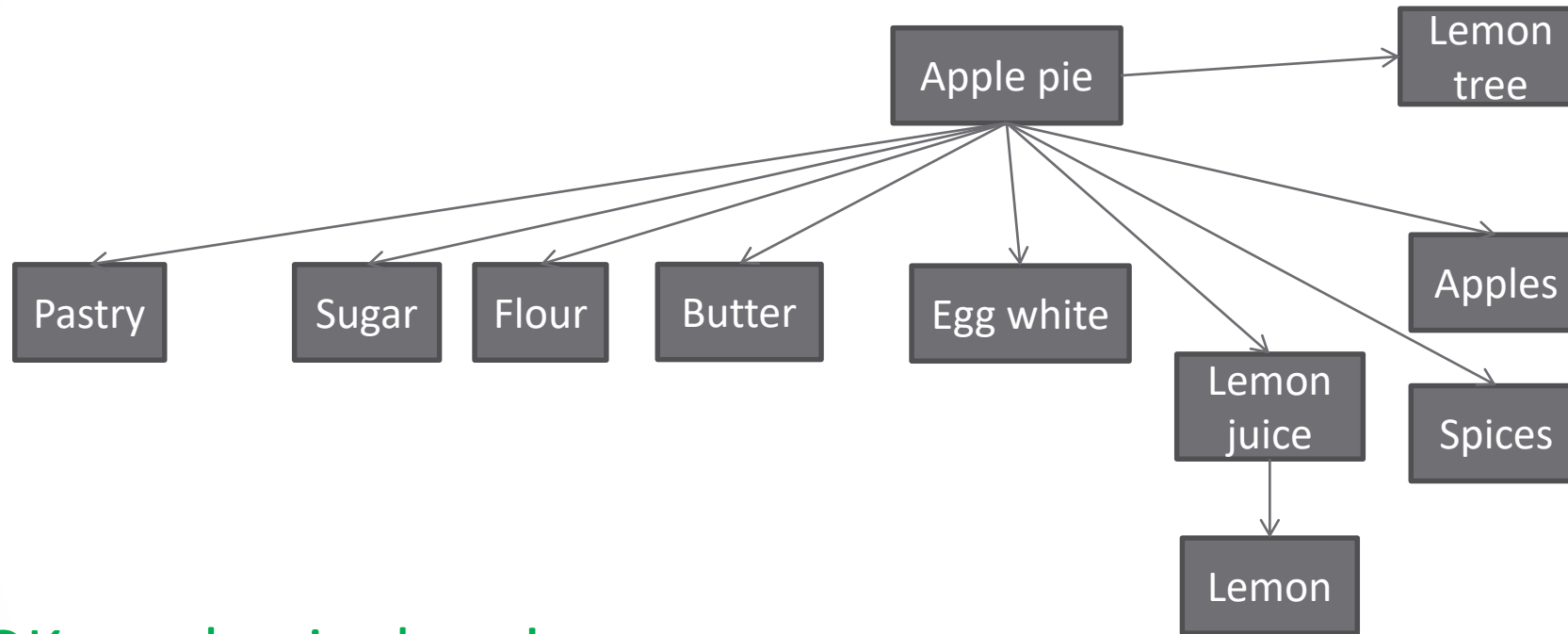
- What happens when a dependency is missing?



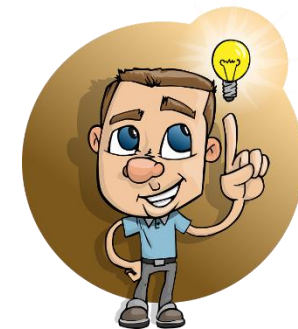
Error: I could not find a lemon tree to make the apple pie

# Direct vs indirect

- What happens when a dependency is missing?

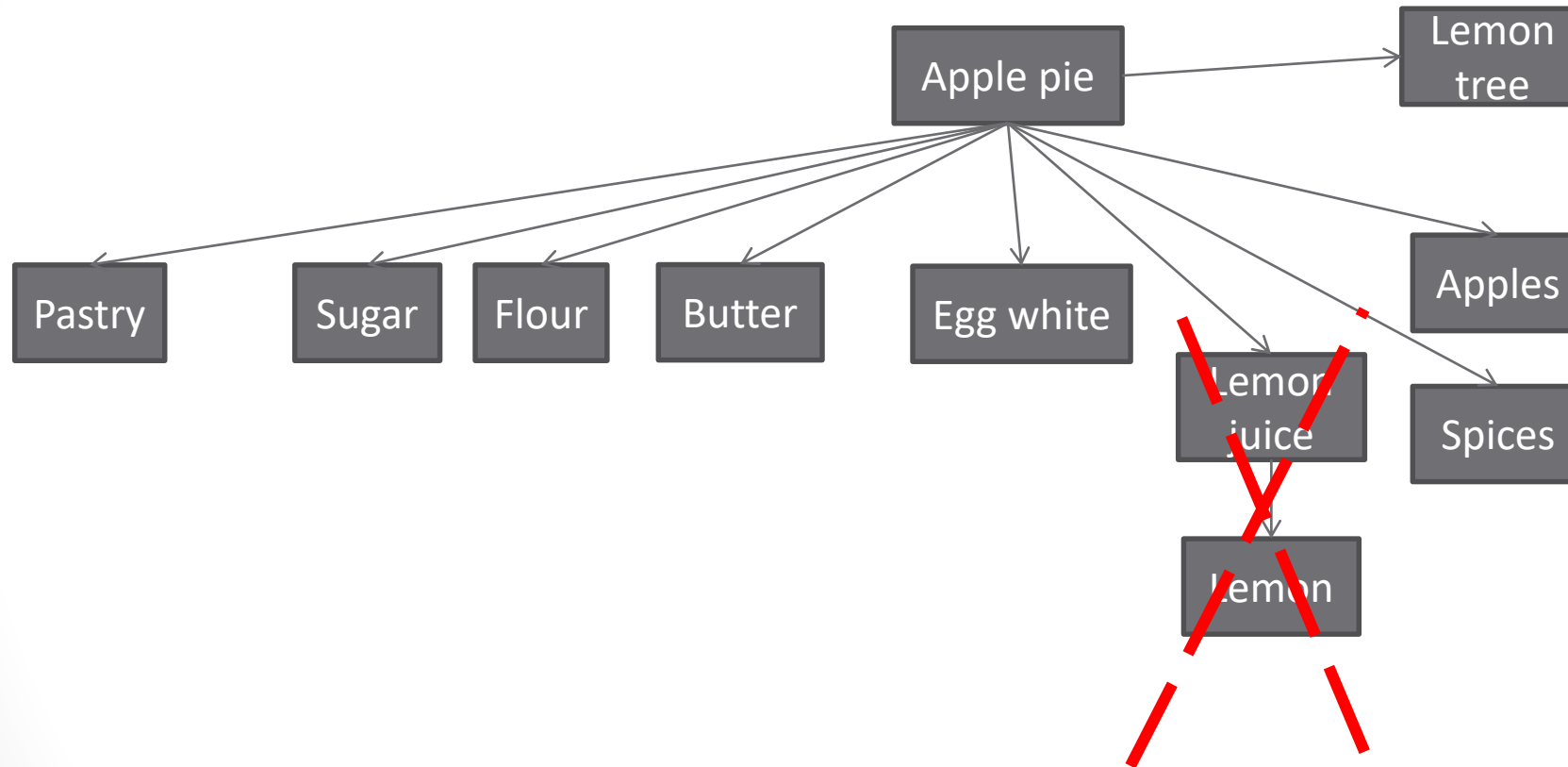


OK, apple pie done!



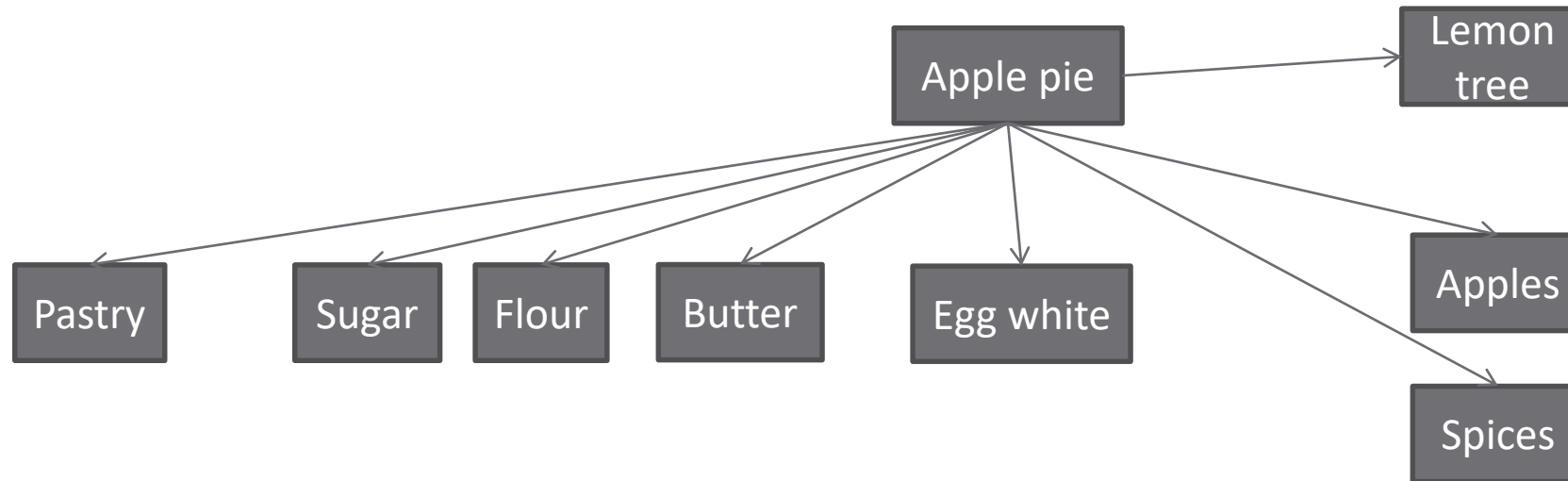
# Direct vs indirect

- What happens when a dependency is missing?



# Direct vs indirect

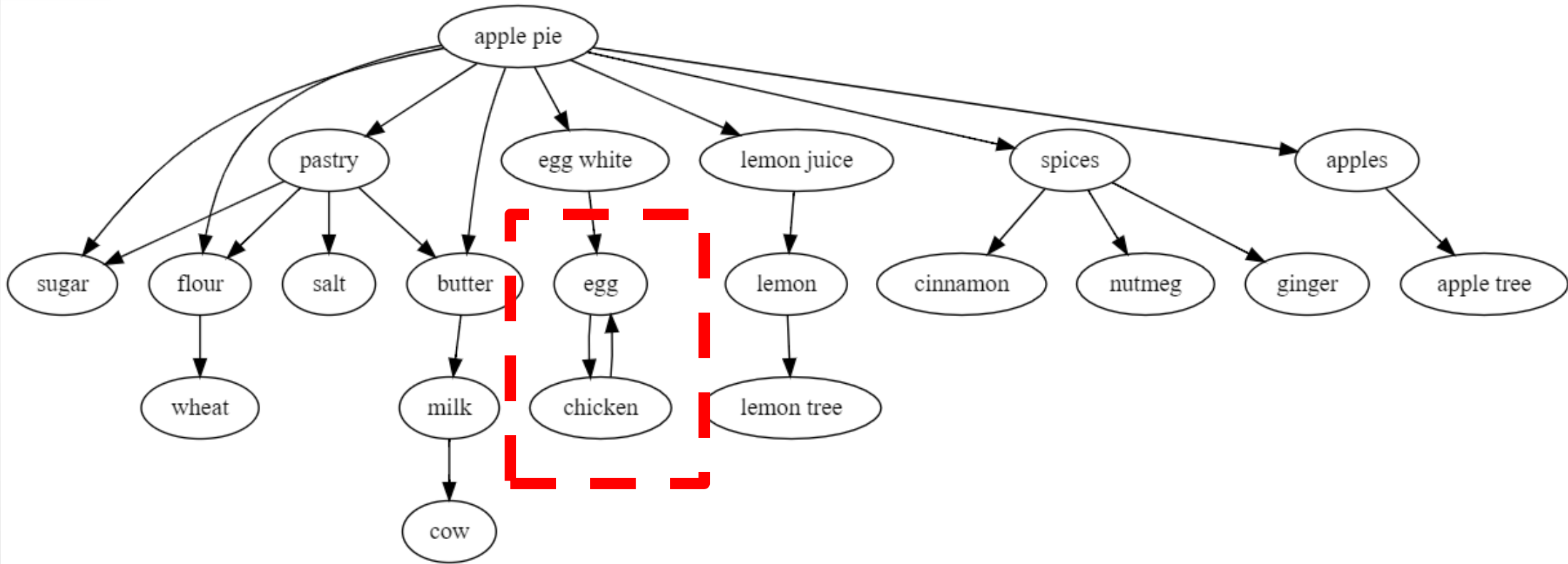
- What happens when a dependency is missing?



Why do I need a lemon tree for an apple pie?

Only define direct dependencies

# Cyclic dependency



# Declaration vs definition

- Declarations introduce (or re-introduce) names into the C++ program

<https://en.cppreference.com/w/cpp/language/declarations>

```
int add_values(int, int);
```

- Definitions are declarations that fully define the entity introduced by the declaration

<https://en.cppreference.com/w/cpp/language/definition>

```
int add_values(int a, int b) {  
    return a + b;  
}
```

One definition rule: Only one definition is allowed in one translation unit and in the entire program

# Declaration vs definition

```
extern int foo;
```

```
int add_values(int, int);
```

```
class Calculator;
```

```
template<typename T>  
T add_values(T a, T b);
```

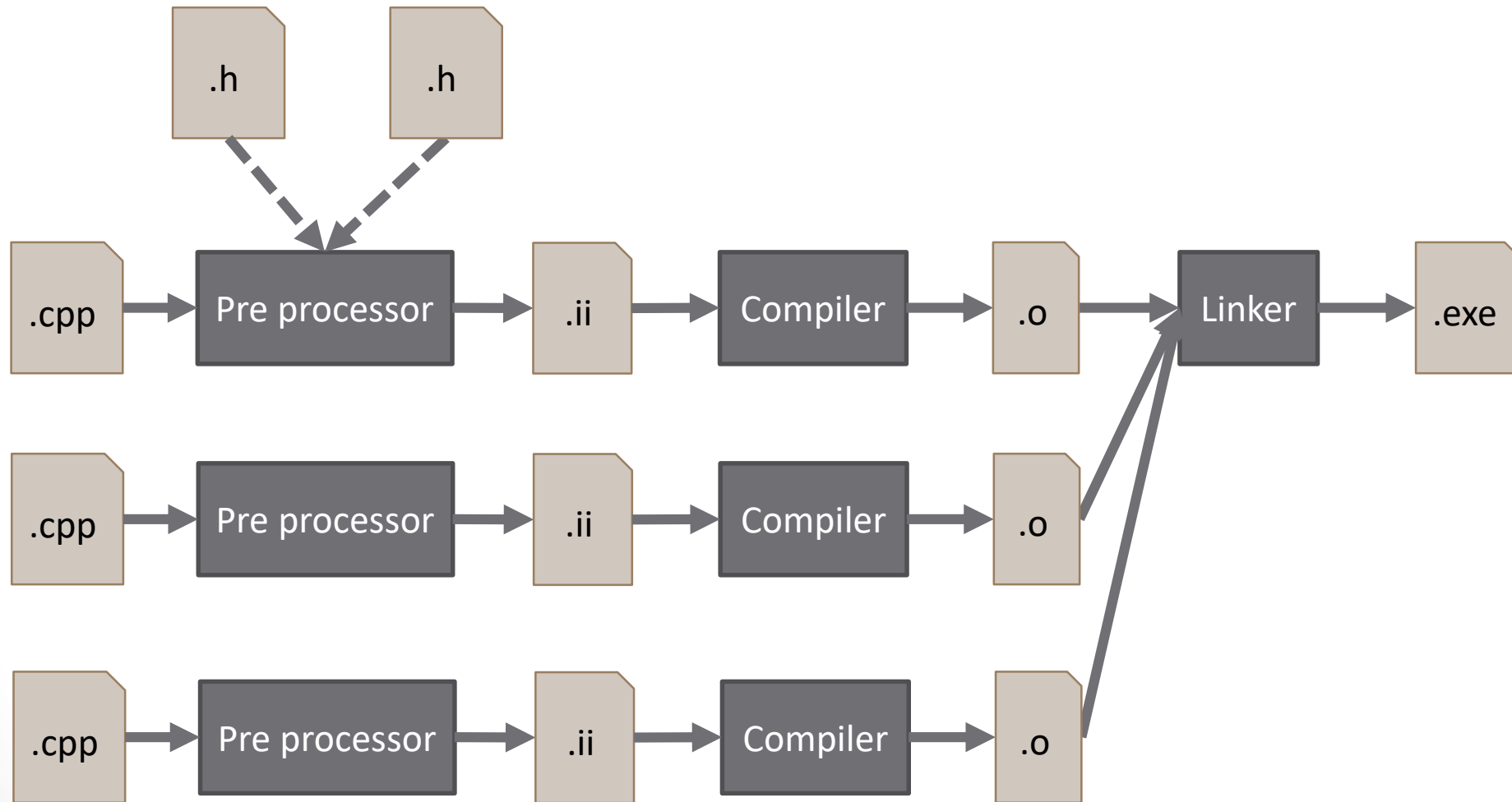
```
int foo;
```

```
int add_values(int a, int b) {  
    return a + b;  
}
```

```
class Calculator {  
    void add_value();  
};
```

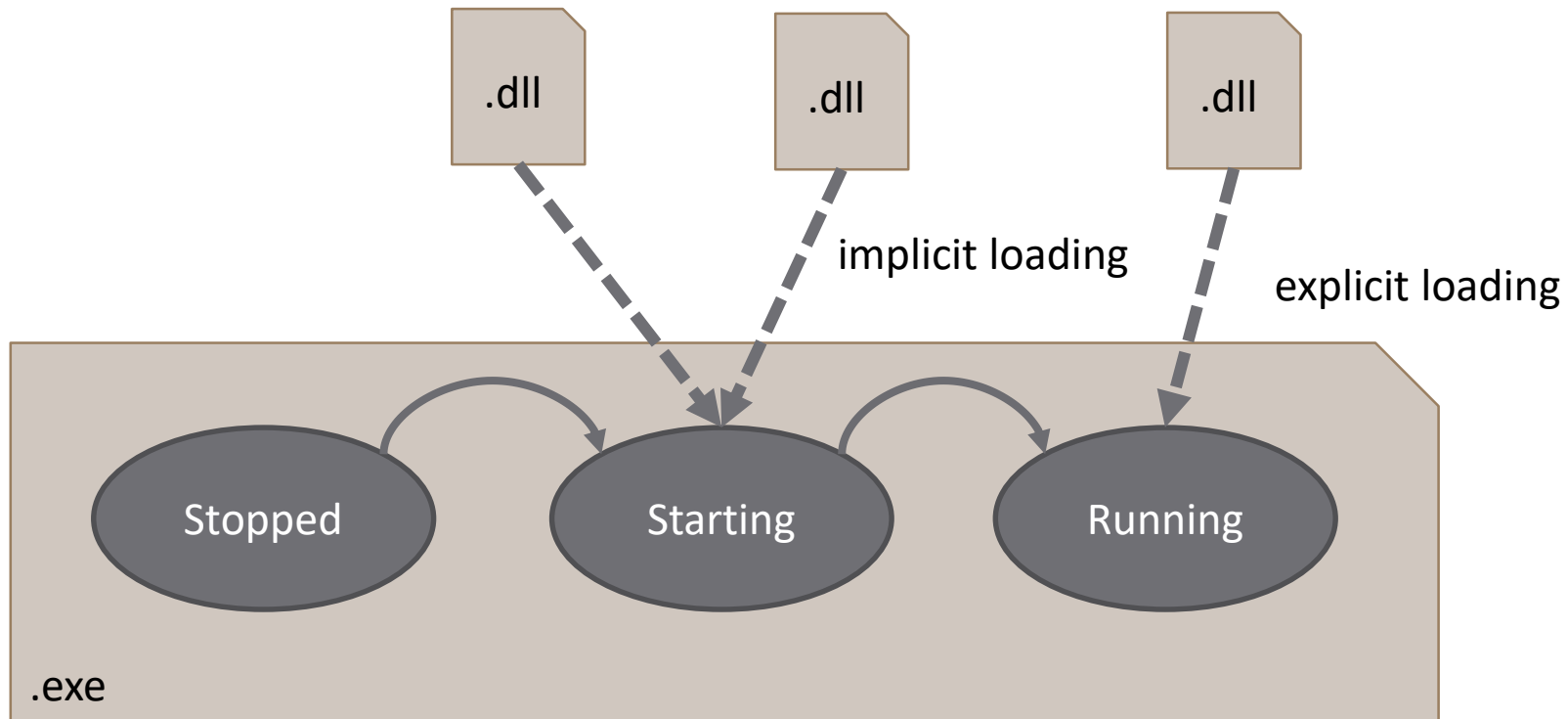
```
template<typename T>  
T add_values(T a, T b)  
{  
    return a + b;  
}
```

# Building phase



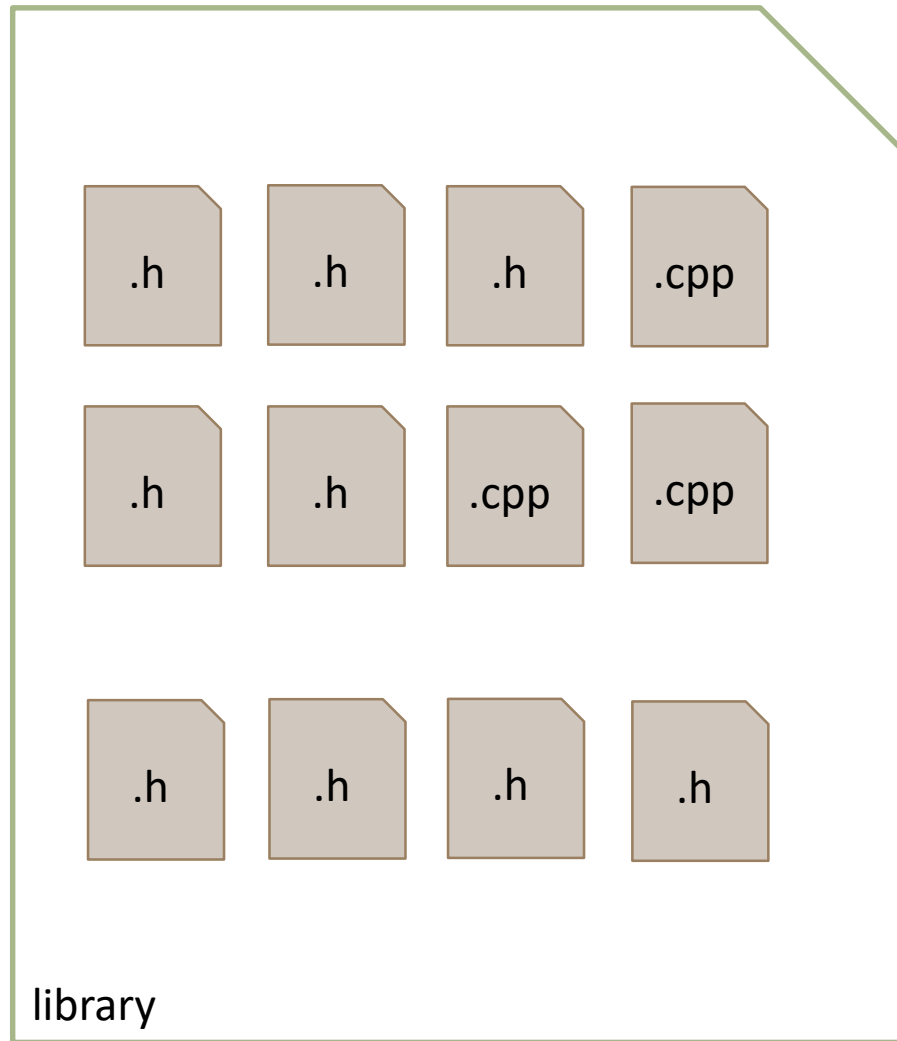


# Runtime

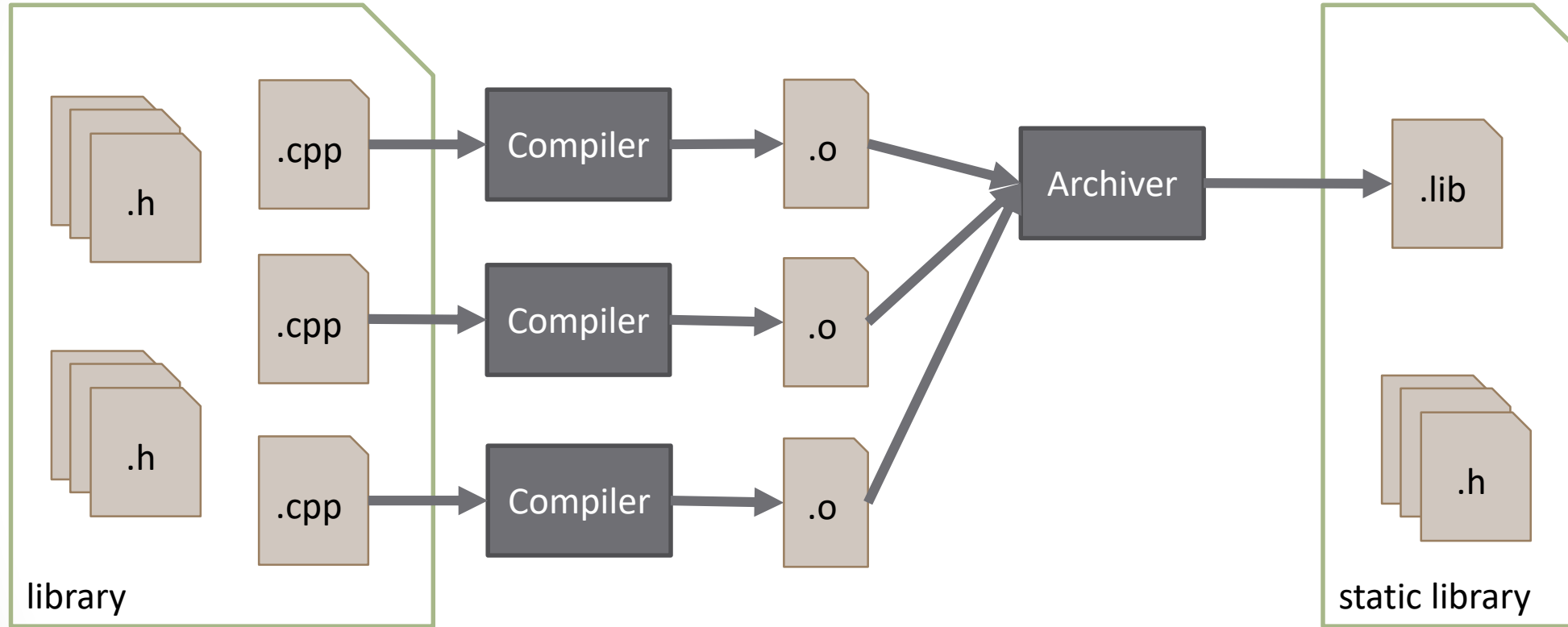


CppCon 2017: James McNellis “Everything You Ever Wanted to Know about DLLs”  
<https://www.youtube.com/watch?v=JPQWQfDhICA>

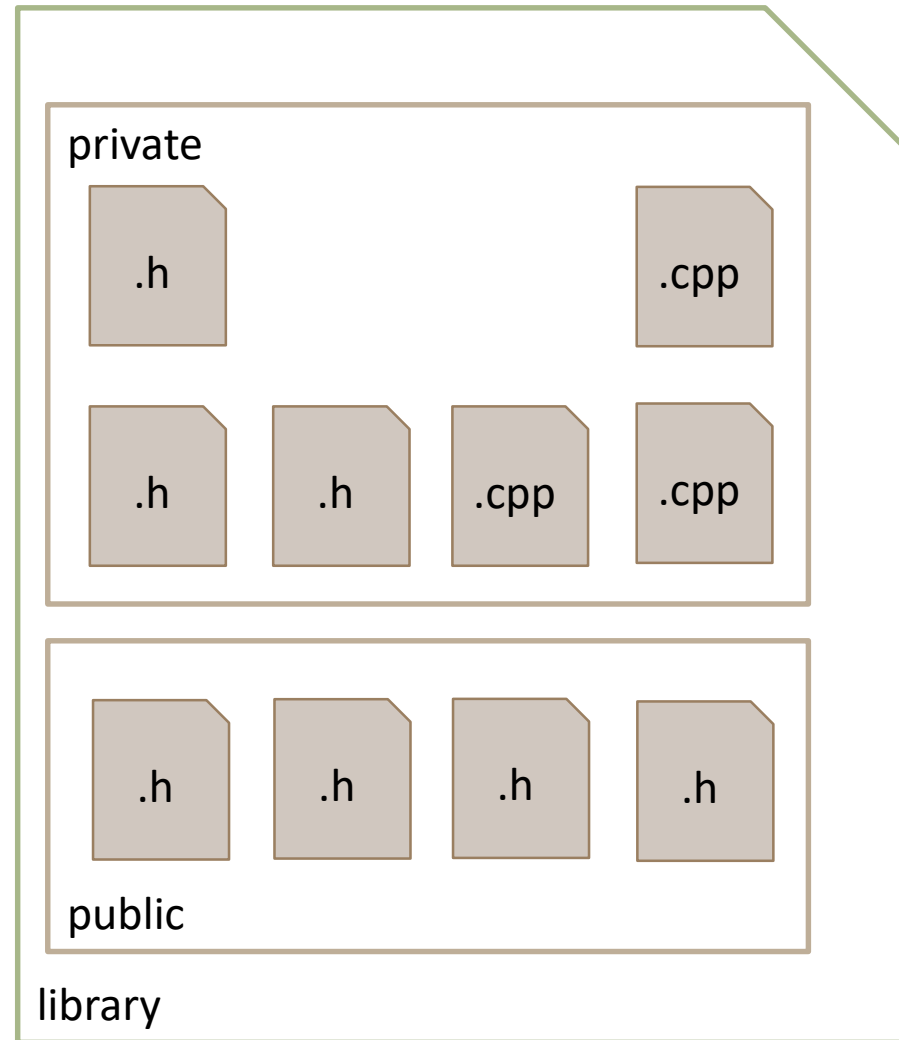
# Building a library



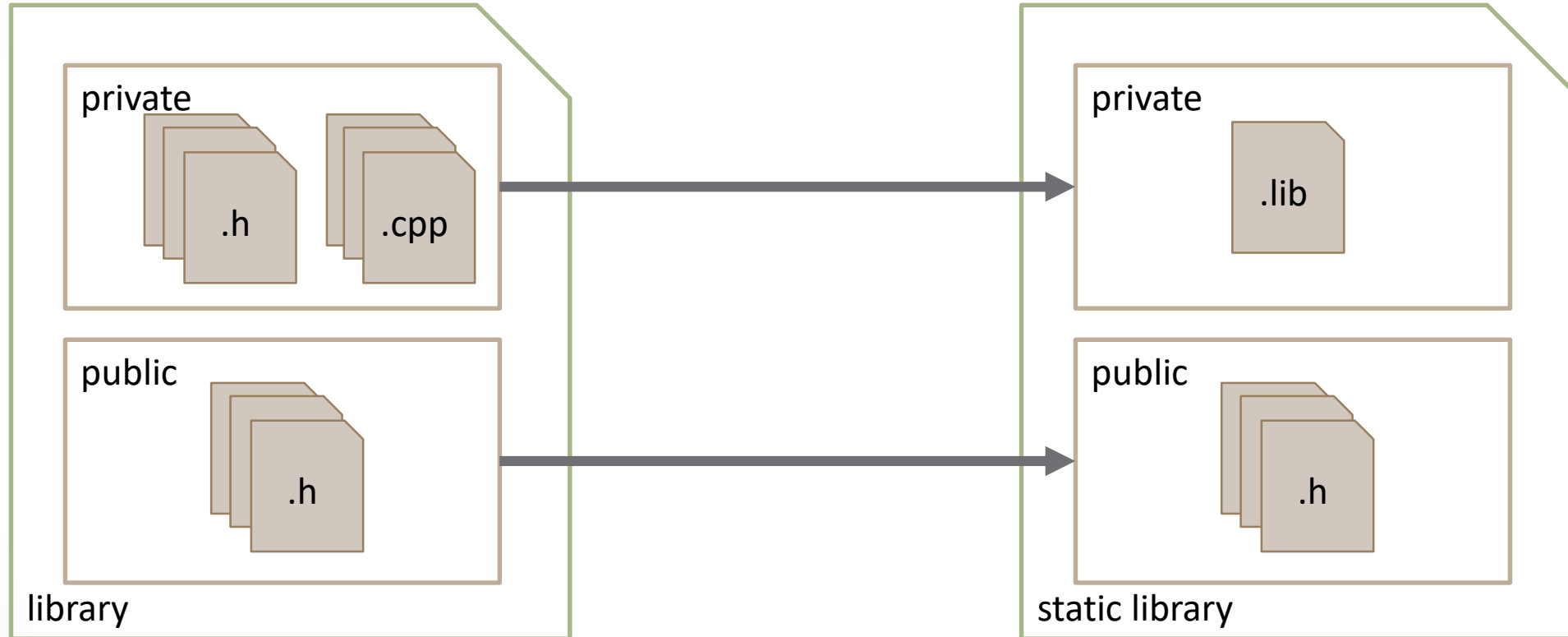
# Building a static library



# Building a static library

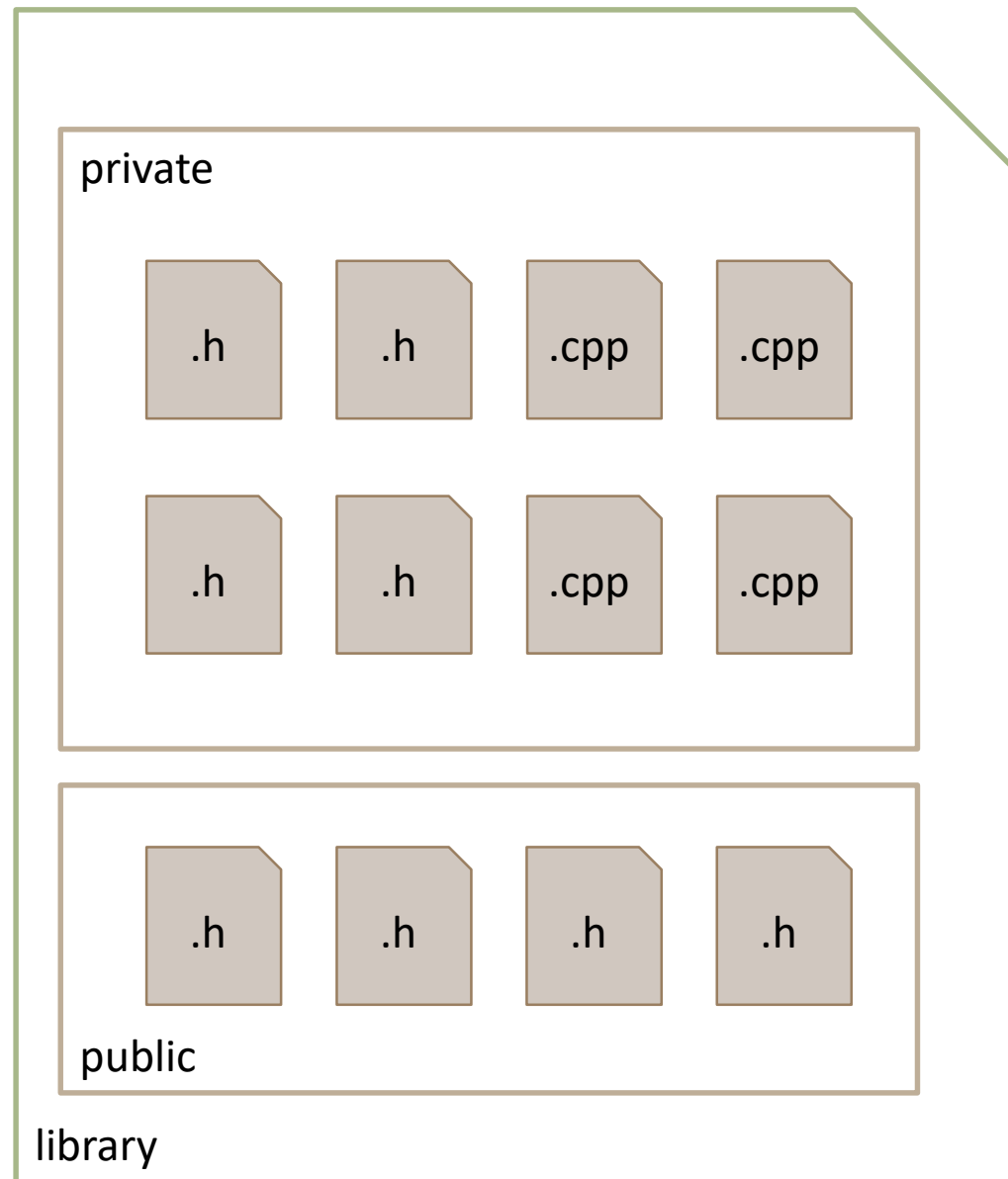
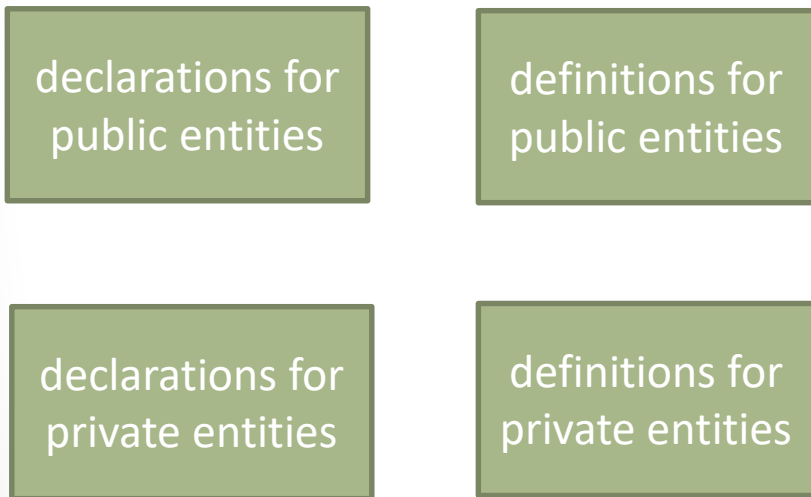


# Building a static library



# Building a library

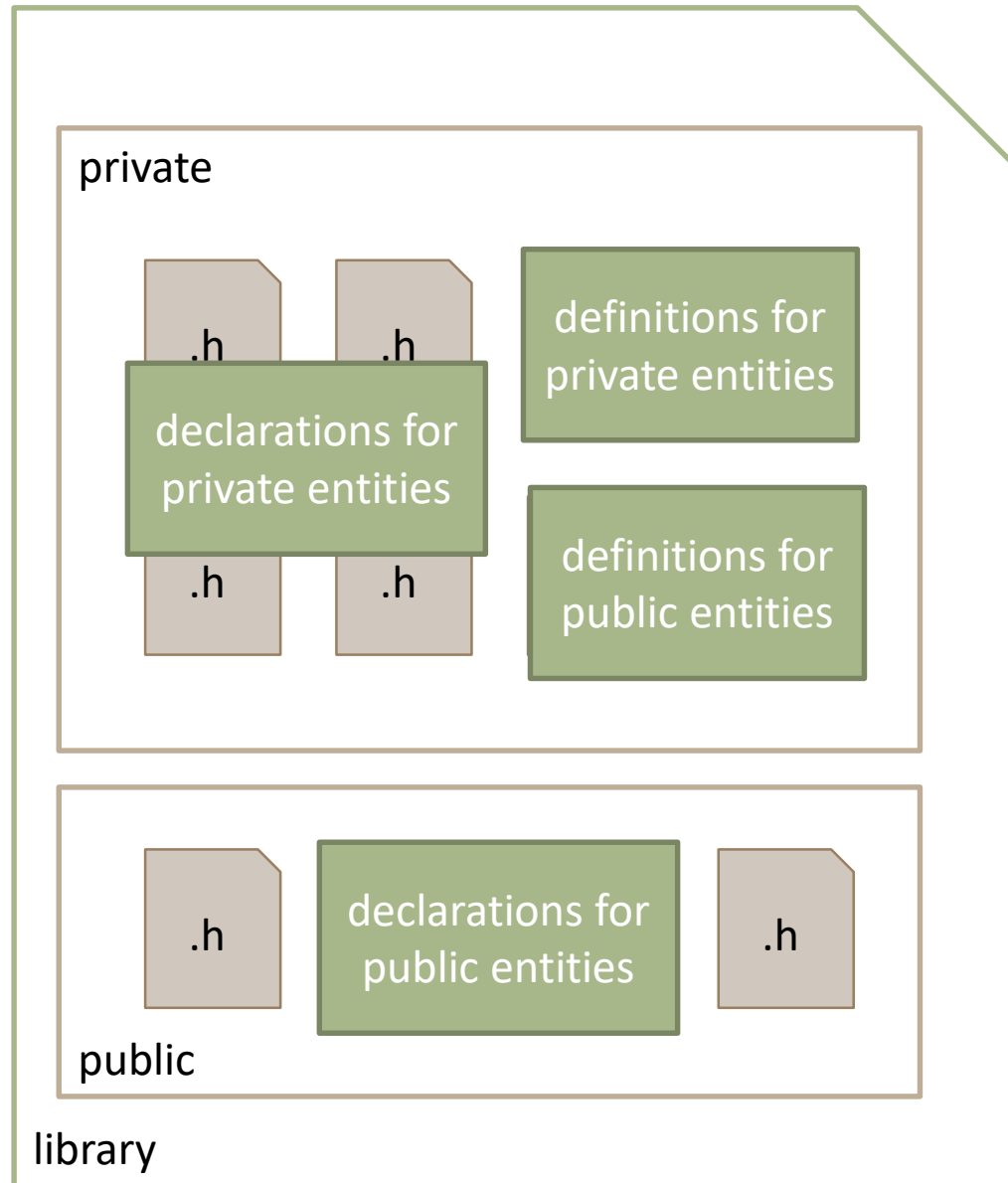
- Where to put declarations and definitions?



Forward declaration of classes help with the separation

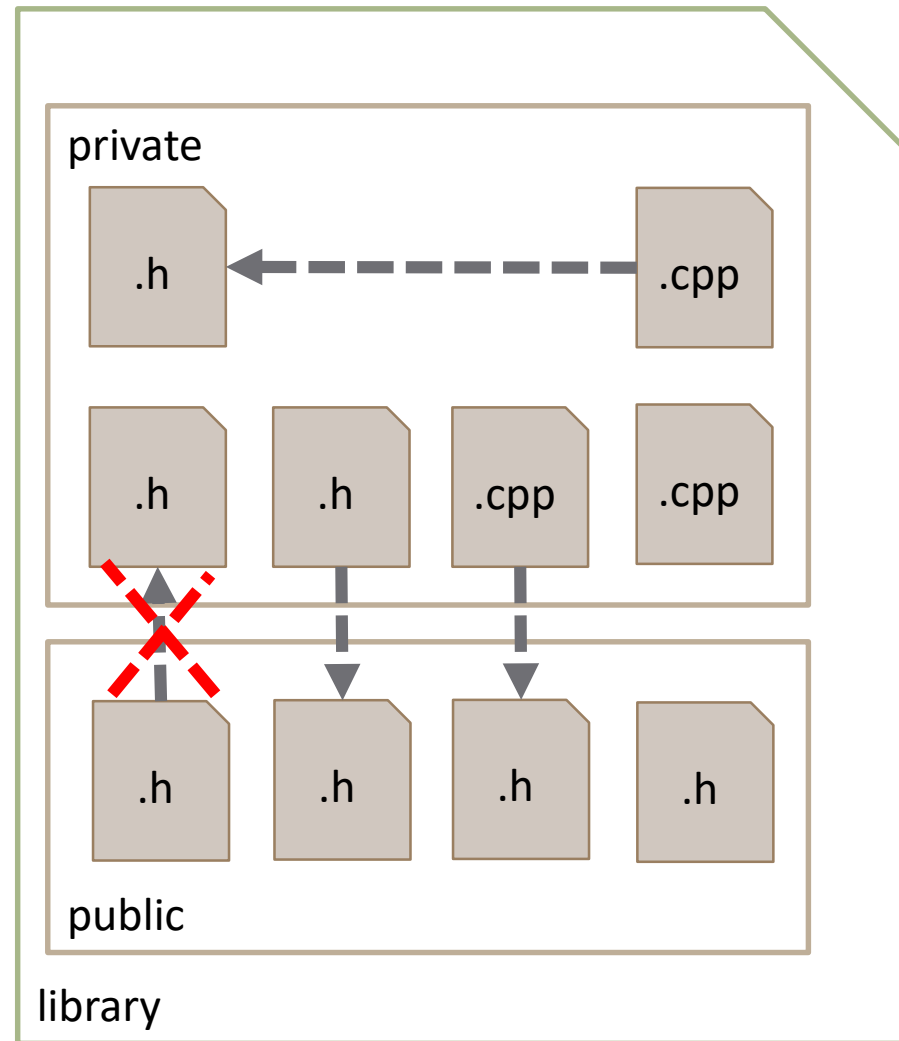
# Building a library

- Where to put declarations and definitions?



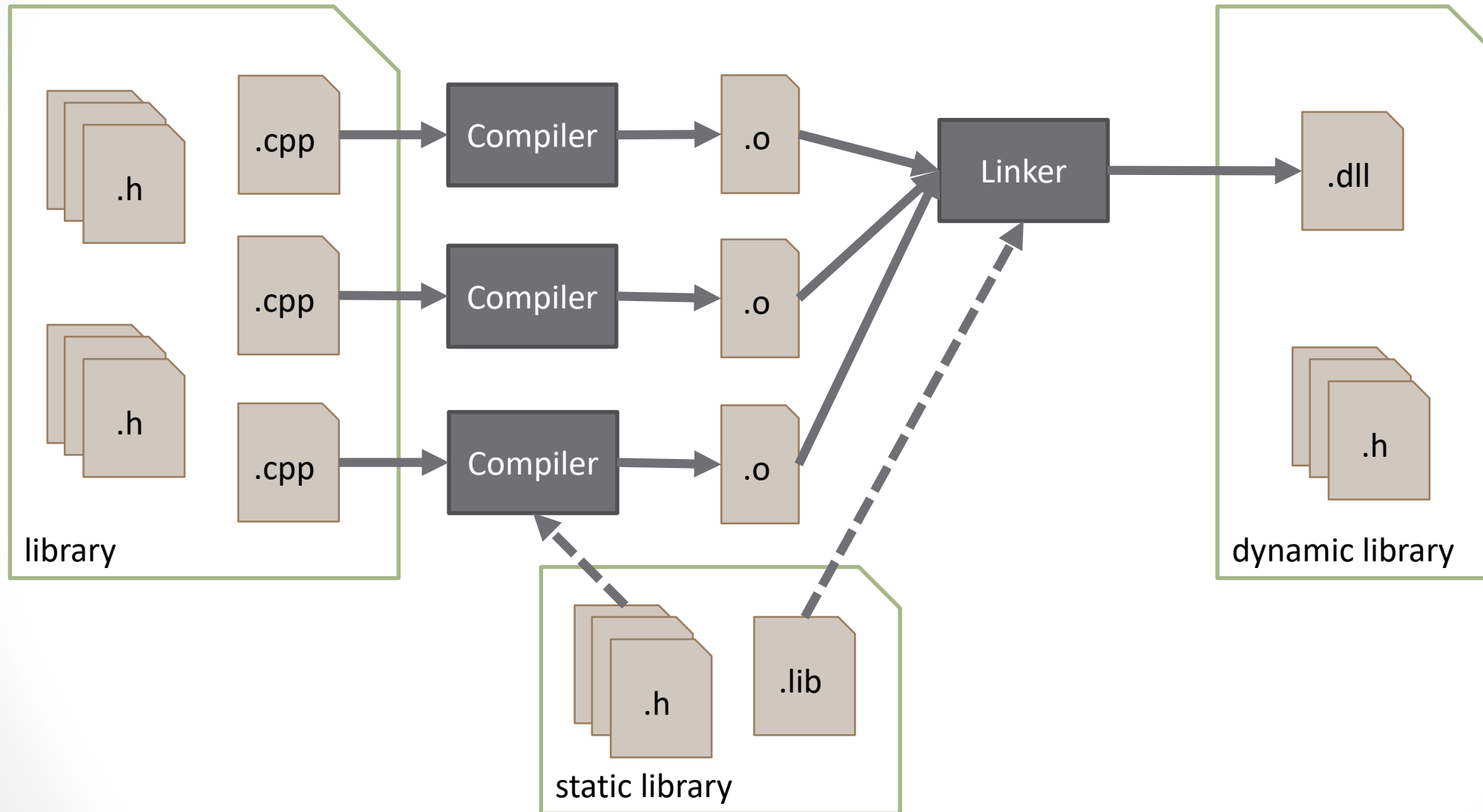
Forward declaration of classes help with the separation

# Building a static library

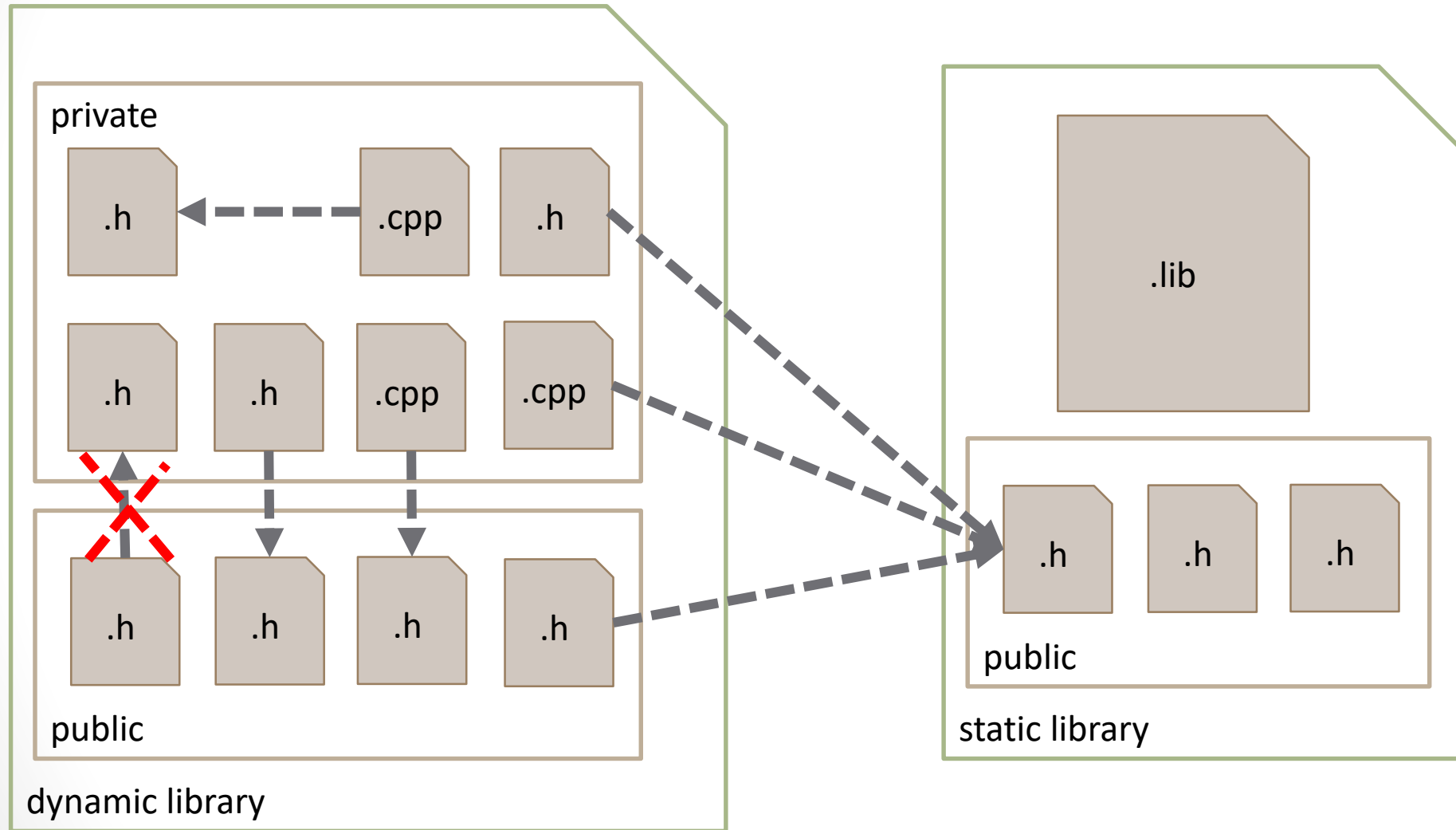




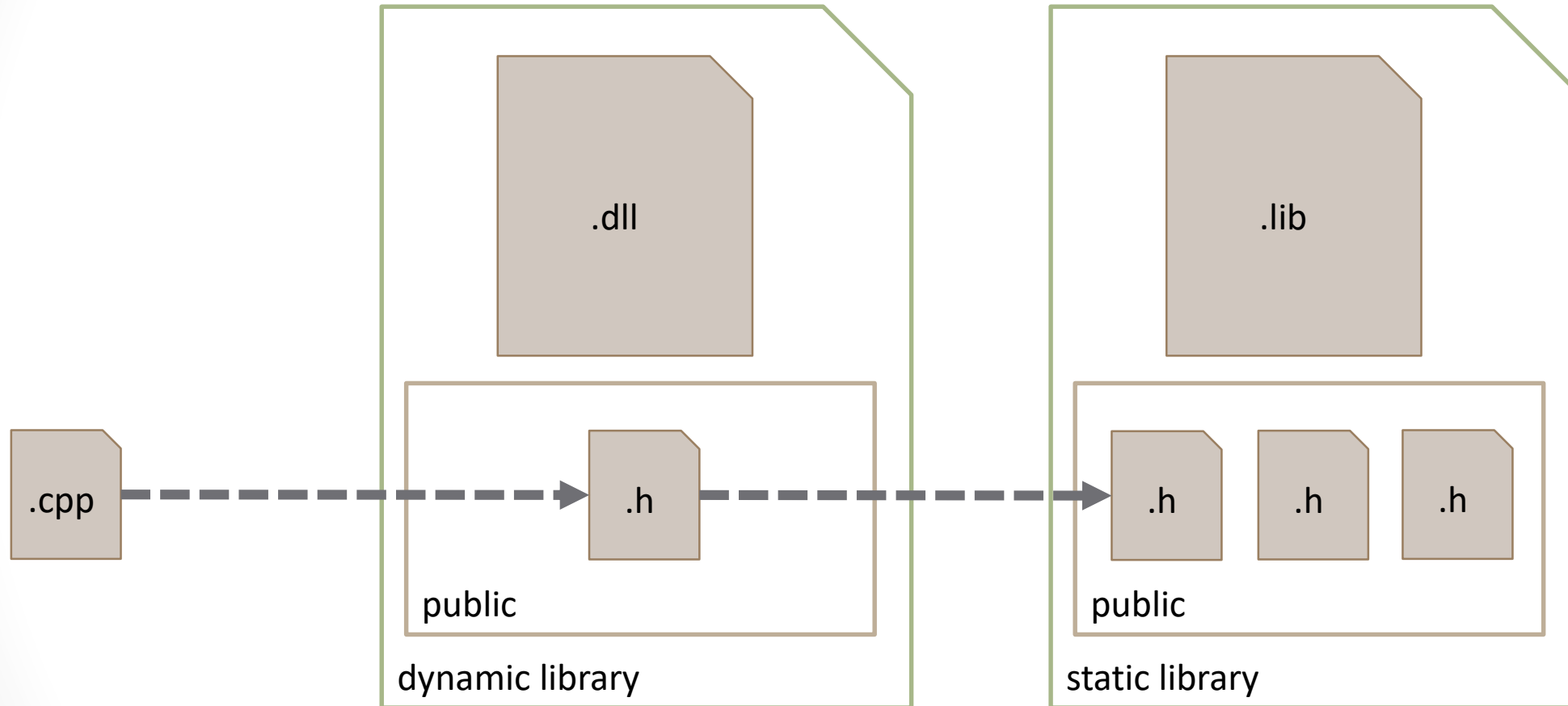
# Building a dynamic library



# Building a dynamic library



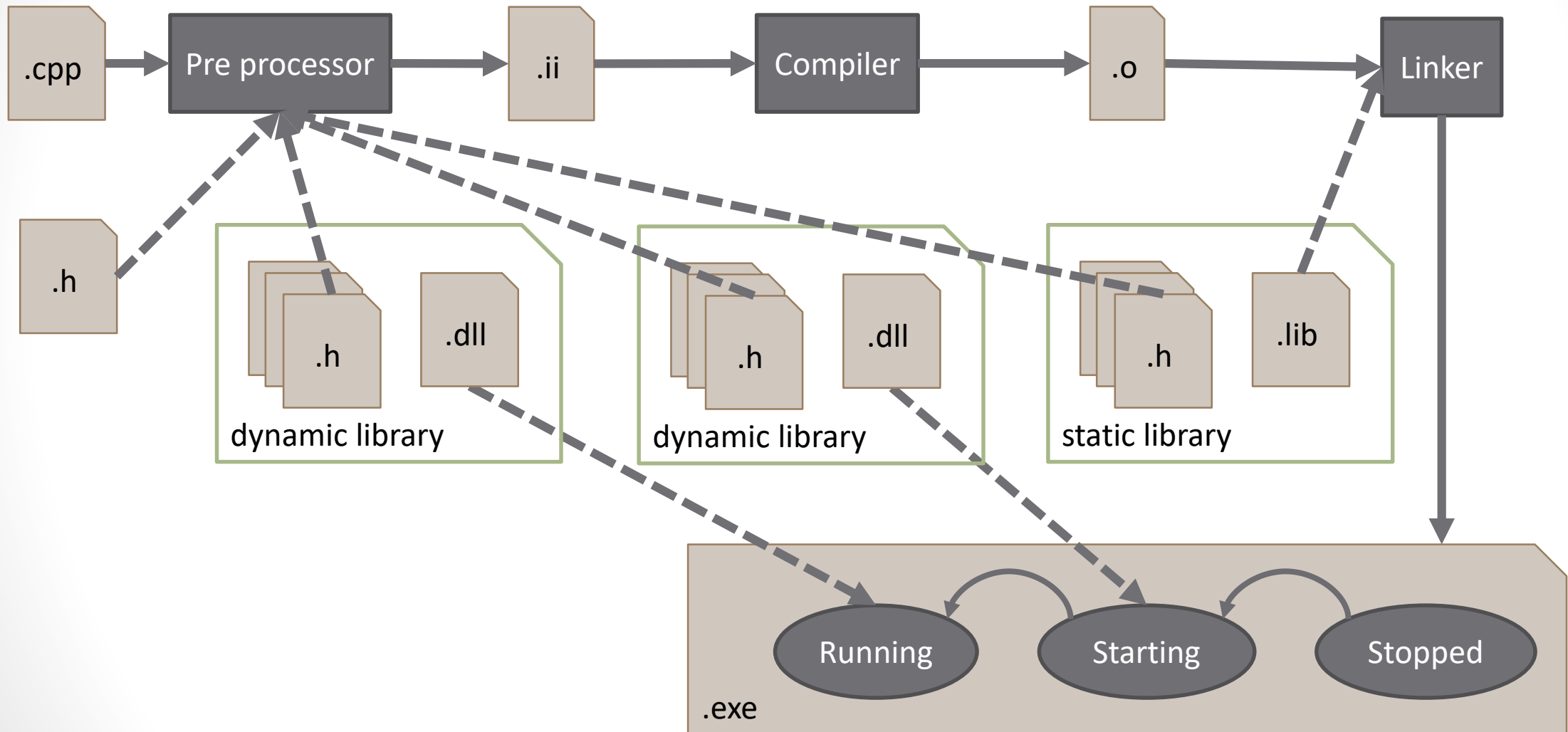
# Building a dynamic library



10 differences between static and dynamic libraries every C++ developer should know

<https://www.acodersjourney.com/cplusplus-static-vs-dynamic-libraries/>

# Library usage



# Installing clang

```
$ pacman -S mingw-w64-x86_64-clang  
resolving dependencies...  
looking for conflicting packages...
```

```
Packages (18) mingw-w64-x86_64-binutils-2.32-3 mingw-w64-x86_64-crt-git-7.0.0.5524.2346384e-1  
mingw-w64-x86_64-gcc-9.2.0-2 mingw-w64-x86_64-gcc-libs-9.2.0-2  
mingw-w64-x86_64-gmp-6.1.2-1 mingw-w64-x86_64-headers-git-7.0.0.5524.2346384e-1  
mingw-w64-x86_64-isl-0.21-1 mingw-w64-x86_64-libffi-3.2.1-4  
mingw-w64-x86_64-libiconv-1.16-1  
mingw-w64-x86_64-libwinpthread-git-7.0.0.5522.977a9720-1 mingw-w64-x86_64-llvm-8.0.1-3  
mingw-w64-x86_64-mpc-1.1.0-1 mingw-w64-x86_64-mpfr-4.0.2-2  
mingw-w64-x86_64-windows-default-manifest-6.4-3  
mingw-w64-x86_64-winpthreads-git-7.0.0.5522.977a9720-1 mingw-w64-x86_64-z3-4.8.6-1  
mingw-w64-x86_64-zlib-1.2.11-7 mingw-w64-x86_64-clang-8.0.1-3
```

```
Total Download Size: 436.85 MiB  
Total Installed Size: 2346.90 MiB
```

# Installing clang

```
$ pactree mingw-w64-x86_64-clang
mingw-w64-x86_64-clang
├── mingw-w64-x86_64-llvm provides mingw-w64-x86_64-llvm=8.0.1-3
│   ├── mingw-w64-x86_64-libffi
│   └── mingw-w64-x86_64-gcc-libs
│       ├── mingw-w64-x86_64-gmp
│       ├── mingw-w64-x86_64-mpc
│       │   ├── mingw-w64-x86_64-mpfr
│       │   └── mingw-w64-x86_64-gmp
│       └── mingw-w64-x86_64-mpfr
├── mingw-w64-x86_64-libwinpthread-git provides mingw-w64-x86_64-libwinpthread
├── mingw-w64-x86_64-gcc
│   ├── mingw-w64-x86_64-binutils
│   │   ├── mingw-w64-x86_64-libiconv
│   │   └── mingw-w64-x86_64-zlib
│   ├── mingw-w64-x86_64-crt-git provides mingw-w64-x86_64-crt
│   │   └── mingw-w64-x86_64-headers-git
│   ├── mingw-w64-x86_64-headers-git provides mingw-w64-x86_64-headers
│   ├── mingw-w64-x86_64-isl
│   ├── mingw-w64-x86_64-libiconv
│   ├── mingw-w64-x86_64-mpc
│   └── mingw-w64-x86_64-gcc-libs provides mingw-w64-x86_64-gcc-libs=9.2.0-2
├── mingw-w64-x86_64-windows-default-manifest
├── mingw-w64-x86_64-winpthreads-git provides mingw-w64-x86_64-winpthreads
│   ├── mingw-w64-x86_64-crt-git
│   └── mingw-w64-x86_64-libwinpthread-git provides mingw-w64-x86_64-libwinpthread-git=7.0.0.5522.977a9720
├── mingw-w64-x86_64-zlib
└── mingw-w64-x86_64-z3
```

Does llvm/clang still need MinGW gcc after built?

<https://stackoverflow.com/questions/9348197/does-llvm-clang-still-need-mingw-gcc-after-built>

# Dependencies in code

```
// hello.h
#ifdef WE_ARE_IN_DEBUG
#include "hello_debug.h"
#else
#include "hello_release.h"
#endif

// hello_debug.h
const char* HELLO_WORLD = "Hello slow world!";

// hello_release.h
const char* HELLO_WORLD = "Hello fast and optimized world!";

// main.cpp
#include "hello.h"
#include <iostream>

int main() {
    std::cout << HELLO_WORLD << "\n";
}
```

```
$ clang main.cpp -DWE_ARE_IN_DEBUG -o main -lstdc++ && ./main
Hello slow world!
```

Dependencies depending on macro definitions

# Dependencies in code

```
// config.h
```

```
#ifdef _DEBUG  
#define WE_ARE_IN_DEBUG  
#endif
```

```
// hello.h
```

```
#ifdef WE_ARE_IN_DEBUG  
#include "hello_debug.h"  
#else  
#include "hello_release.h"  
#endif
```

```
// hello_debug.h
```

```
const char* HELLO_WORLD = "Hello slow world!";
```

```
// hello_release.h
```

```
const char* HELLO_WORLD = "Hello fast and optimized world!";
```

```
// main.cpp
```

```
#include "hello.h"  
#include "config.h"  
#include <iostream>  
  
int main() {  
    std::cout << HELLO_WORLD << "\n";  
}
```

```
$ clang main.cpp -D_DEBUG -o main -lstdc++ && ./main  
Hello fast and optimized world!
```

Dependencies depending on include order



# Dependencies in code

```
// basics.h
#include <vector>
#include <iostream>
#include <string>
```

```
// hello.h
#include "basics.h"
const char* HELLO_WORLD = "Hello slow world!";

// main.cpp
#include "hello.h"

int main() {
    std::cout << HELLO_WORLD << "\n";
}
```

```
$ clang main.cpp -o main -lstdc++ && ./main
Hello slow world!
```

Missing includes go unnoticed

# Dependencies in code

- Order of includes affects our dependencies
  - One macro defined in one header affects another header
  - One missing include in a header file “will not be an issue” if it is included by another header file
- How can we mitigate it?
  - Include headers from local to global

```
#include "my_file.h"  
#include "another_lib.h"  
#include <vector>
```

```
#include <vector>  
#include "another_lib.h"  
#include "my_file.h"
```

- What happens if my\_file.h uses vector without including it?
- What happens if another\_lib.h includes vector without using it?

# Dependencies in code

- How can we mitigate it?
  - Automatic tool that shuffle headers and compiles?

```
$ python3 include_checker.py --folder "/my/source/folder/" --command "clang main.cpp -o main -lstdc++"
```

```
// main.cpp
#include "hello.h"
#include "config.h"
#include <iostream>
```

```
int main() {
    std::cout << HELLO_WORLD << "\n";
}
```



```
// main.cpp
#include "hello.h"
#include <iostream>
#include "config.h"
```

```
int main() {
    std::cout << HELLO_WORLD << "\n";
}
```

- Does it compile? If yes, repeat the process, if not, fix it!

# Dependencies in code

- Could we have it better in the future?

## C++20: Modules

```
// math.cppm
```

```
export module math;
```

```
export int add(int fir, int sec){  
    return fir + sec;  
}
```

```
// main.cpp
```

```
import math;
```

```
int main(){  
    add(2000, 20);  
}
```

- At least the import of the modules does not depend on the order

CppCon 2019: Boris Kolpackov “Practical C++ Modules”

<https://www.youtube.com/watch?v=szHV6RdQdg8>

Example from: <https://www.modernescpp.com/index.php/c-20-modules>

# Summary

- What is a dependency graph
- Only direct dependencies should be defined
- For each entity we can have multiple declarations but only one definition per translation unit
- How to structure the declarations and definitions inside a library
- Implications of bad dependency definition

# Dependency management in C++

Xavier Bonaventura

BMW AG.

 xbonaventurab

 limdor

 limdor

 xavierbonaventura

C++ User Group Munich – 17<sup>th</sup> October 2019