



Session 3

Tristan Brindle

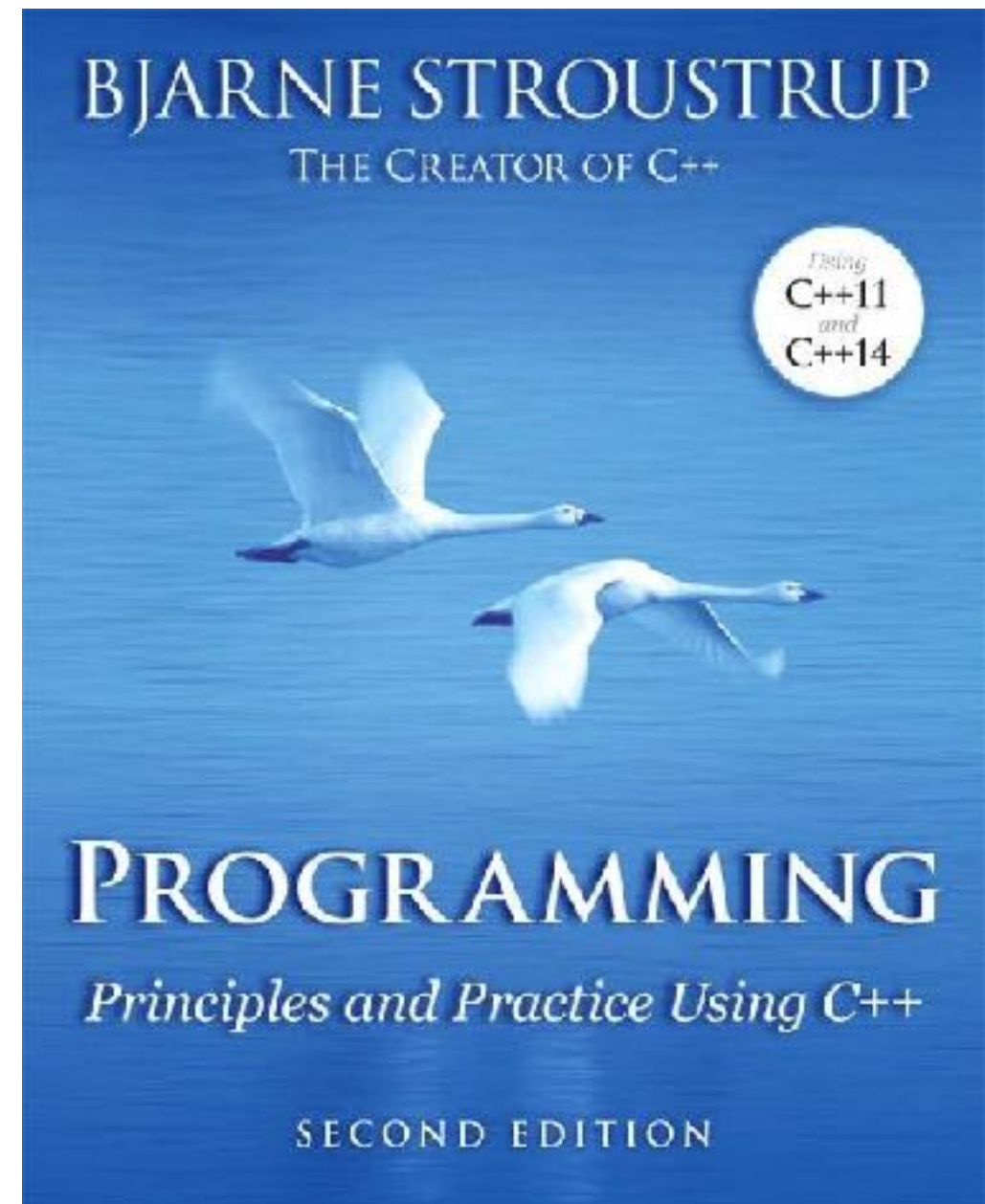
Feedback



- We'd love to hear from you!
- The easiest way is via the *cpplang* channel on Slack — we have our own chatroom, *#ug_uk_cpplondonuni*
- Go to <https://cpplang.now.sh/> for an “invitation”

Textbook

- We'll be using
*"Programming Principles
and Practice Using C++"* by
Bjarne Stroustrup
- Please pick up a copy if you
haven't already



Last week...



- Further intro to C++
- Commonly used types (integer types, `std::string`, `std::vector`)
- Declaring variables
- The `const` keyword
- Type deduction with the `auto` keyword

This week...



- Getting set up with a compiler and CLion
- The C++ compilation model
 - What compilers do
 - Using the compiler from the command line
 - The C++ compilation process
 - Separate compilation
 - Using libraries
 - Header files and `#include`

Getting set up for C++

- So far we have only used an online compiler to write examples
- Today we're going to make sure everyone is set up with a working compiler and IDE (integrated development environment)
- In this course we'll use the CLion IDE and either the GCC or Clang compiler

Installing a compiler



- Windows:
 - Go to <https://nuwen.net/mingw.html> and select either of the mingw.exe download links
 - Run the executable and select the install directory (e.g. C:\MinGW)
- Mac:
 - Open a Terminal window and enter
 - `xcode-select --install`
 - This will download and install the command-line development tools
- Linux:
 - Try running `CC --version` in a terminal window. If this works, you already have a C++ compiler installed
 - Otherwise, please install g++ using your system package manager

Installing CLion



- Please go to <https://www.jetbrains.com/clion/> and download CLion via the “start a 30 day trial” link
- Run the installer and then launch CLion
- The wizard should guide you through the process of creating a “toolchain”
 - Windows: when prompted, tell it that you want to use MinGW, and enter the directory you selected earlier (e.g. C:\MinGW)
 - Mac and Linux: CLion should automatically detect your compiler

Running CLion



- Live demo

Exercise



- In CLion, follow the steps you've just seen to create a new C++ executable project
- Ensure that you can compile and run this program successfully
- If you have any problems, please let one of us know

**Any questions before
we move on?**

Compilers for C++



- There are many C++ compilers out there, but the “big three” are:
 - The GNU Compiler Collection (GCC), called MinGW on Windows
 - Clang, part of the LLVM project
 - Microsoft Visual C++ (MSVC), part of Visual Studio
- In this course we will be using GCC/MinGW or Clang

What is a compiler?



- A *compiler* is a program which takes source code (text) and processes it into some other (lower-level) form
- For example, a Java compiler will accept Java source code, and process it into Java bytecode
- In C++, the compiler will usually generate *machine code*, that is, instructions to be executed directly on the CPU
- Typically, this machine code is specific to a particular CPU architecture and operating system combination

What does a compiler do?



- It's the compiler's job to translate your source code into machine code
- It does this by first checking the *syntax* of your source code: whether you have used symbols and keywords in a way it can understand.
- It then checks the *semantics* of your program: whether it "makes sense"

What does a compiler do?



- If the compiler cannot understand your program, it will terminate with an error, usually called a *compiler error* or *compile-time error*
- In C++ we generally **strongly** prefer compile-time errors to run-time errors (those that occur when the program executes)
- Once it has checked that it understands your program, the compiler will proceed to translate it into machine code, typically applying *optimisations* to make it execute faster

Invoking the compiler

- On Unix systems, we can run the C++ compiler on the command line, passing it the name of the file(s) we want to compile, and the output name, e.g.
 - `CC my_file.cpp -o my_exe`
- This can be very useful for small test programs
- (This is also possible on Windows with MinGW: use `g++ .exe` rather than `CC`)

Exercise



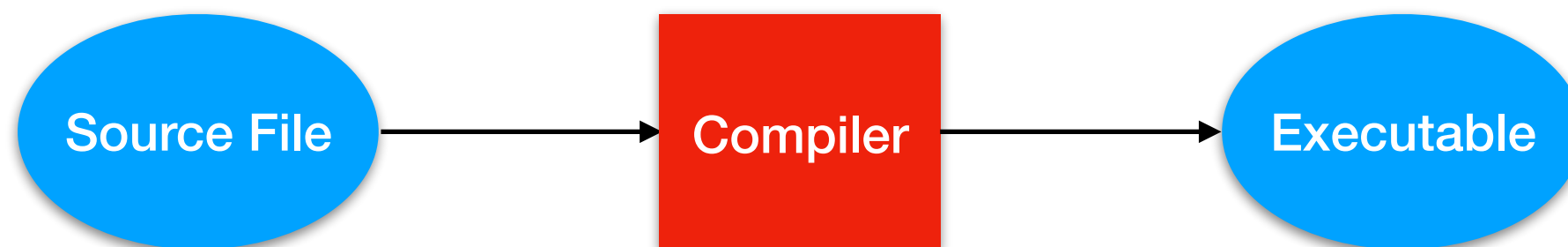
- Open a text editor and enter the “Hello World” program on the right
- Save this file as “hello_world.cpp”
- Compile this file using the command line

```
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

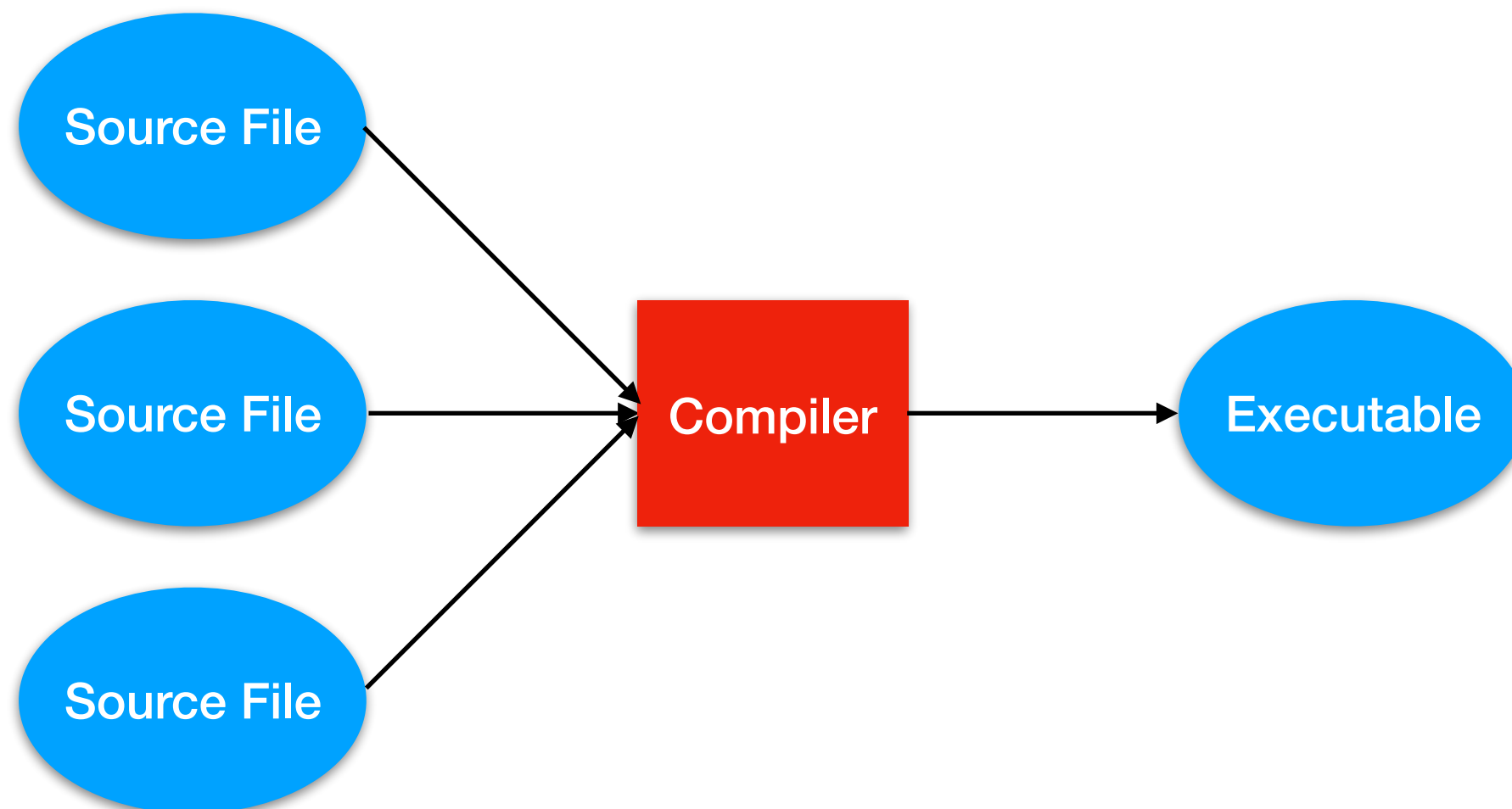
The compilation process

- Schematically, the process we have just used looks like this:



The compilation process

- We can also pass multiple files to the compiler at once

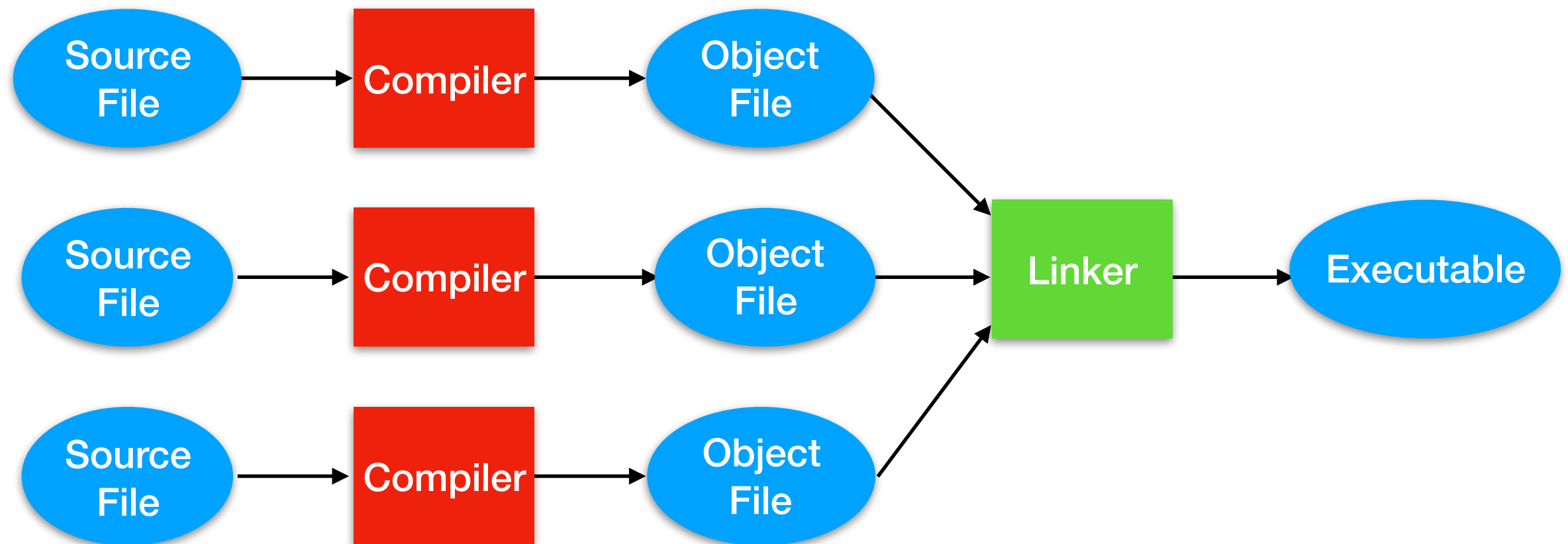


Separate compilation



- In real-world programs however, we usually don't want to compile all our files at once
- Instead, we send each file individually to the compiler, which produces object files (typically with a `.o` or `.obj` extension)
- Another program called the *linker* collects these object files and links them together into an executable

Separate compilation

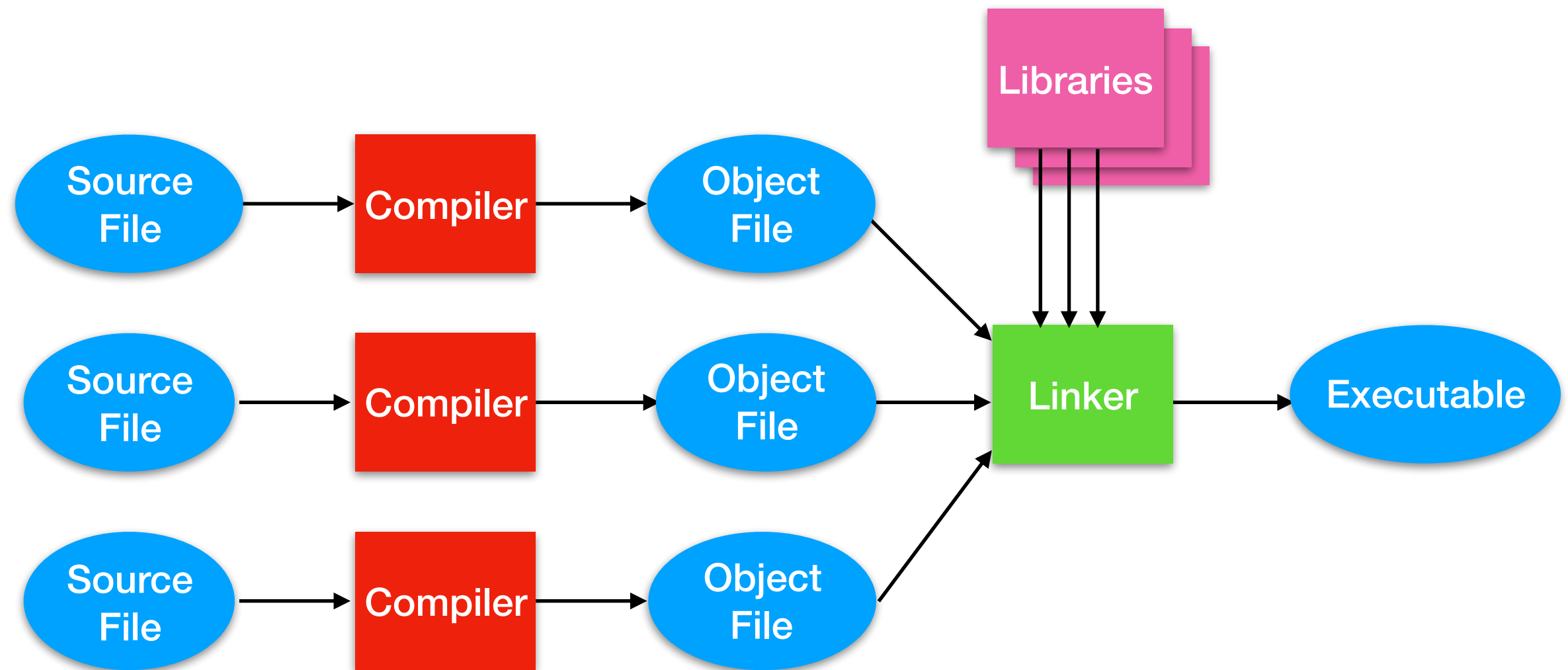


Using libraries



- Real-world programs are usually built using software from different sources, called *libraries*
- Libraries compiled into object files, which are provided to the *linker*
- We've already used one! The *C++ standard library* is bundled with your compiler, and is automatically linked to C++ programs

Separate compilation



Build systems

- Managing the compilation process with multiple source files and libraries is usually automated by build tools or build systems
- These take care of noticing which files have been edited, and recompiling only what is necessary
- There are many, many different build systems out there (MSBuild, Automake, waf, Scons, Meson, etc etc)

Build systems



- The most common build system for C++ projects is CMake, which is supported directly by CLion and most other C++ IDEs
- In general, we'll be relying on CLion to manage our CMake configuration...
- ...but if you're curious, you can take a look at CMakeLists.txt in a CMake project

Headers and implementation files



- As we've seen, each source file is processed individually by the compiler
- But we generally want to re-use types and functions between different source files!
- In C++ this is accomplished using *header files*, or simply *headers*
- Because headers also contains source code, we will sometimes use the term *implementation file* to refer to files that are passed directly to the compiler
- Typically, implementation files have a `.cpp` extension. Header files usually use `.hpp` or `.h`

Headers

- Generally, we place function *declarations* into header files, and the function *implementations* into implementation files
- If the compiler has seen the declaration of a function, it knows how to call it
- Later, the linker matches up function *calls* in one object file to function *implementations* in another
- Lots of other things also go in header files (type definitions, templates, inline functions etc) which we'll cover later

Using header files



- We can include a header file in an implementation file using the `#include` command
- For headers in our own project we need to say

```
#include "header.hpp"
```

- For headers from other libraries we need to say

```
#include <header.hpp>
```

Using header files



- `#include` performs a literal copy and paste of header file text wherever it's called
- Header files can (and usually do) `#include` other headers that they rely on
- We need to be careful to ensure that a header is `#include`-d only once in each implementation file
- We normally use include guards to prevent this, or compiler-specific mechanisms like `#pragma once`
- CLion will generate include guards for you

Creating headers and source files in CLion



- Live demo

Exercise



- In CLion, create a new C++ executable project if you have not already done so
- Add a source file `example.cpp`, with a matching header
- In `example.cpp`, write a function `add(int a, int b)` which returns `a + b`.
- Add a *declaration* of this function to the `example` header
- In `main.cpp`, use this function print the result of `3 + 4`.

Next time

- Value semantics and object lifetime fundamentals
- Basic control flow
 - If statements
 - Loops

Online resources



- <https://isocpp.org/get-started>
- cppreference.com — The bible, but aimed at experts
- cplusplus.com — Another reference site, also has a tutorial section
- learncpp.com — Free online tutorial, very up-to-date
- <https://www.pluralsight.com/authors/kate-gregory> - Comprehensive set of courses from an experienced C++ trainer (free trial)
- reddit.com/r/cpp_questions
- Cpplang Slack channel — <https://cpplang.now.sh/> for an “invite”
- StackOverflow (but...)

Thanks for coming!



C++ London Uni:

- Website: cpplondonuni.com
- Github: github.com/CPPLondonUni

Where to find Tom Breza:

- On Slack: [#learn #ug_uk_cpplondonuni](https://cpplang.slack.com)
- E-mail: tom@PCServiceGroup.co.uk
- Mobile: 07947451167

My stuff:

- Website: tristanbrindle.com
- Twitter: @tristanbrindle
- Github: github.com/tcbrindle

See you next time! 😊