TEST DRIVEN DEVELOPMENT MAKEFILES

Problem Solving with Computers-I

How difficult do you find the course so far?

A. Too easy

B. Easy, I sail through the labs with little effort

C. Moderately easy/difficult – I have to apply concepts and can complete the labs and homeworks with moderate effort

D. I understand the material but my partner does everything --- I don't really have the confidence to code.

E. I am really struggling and feel underprepared for this class

The compilation process

Source code

hello.cpp

Compiler

g++

Executable

a.out

Run Time

Hardware

Executable:
Program in machine code
+Data in binary

Source code:
Text file stored on computers hard disk or some secondary storage

```
1000110001100010000000000000000
1000110011110010000000000000100
1010110011110010000000000000000
1010110001100010000000000000100
```
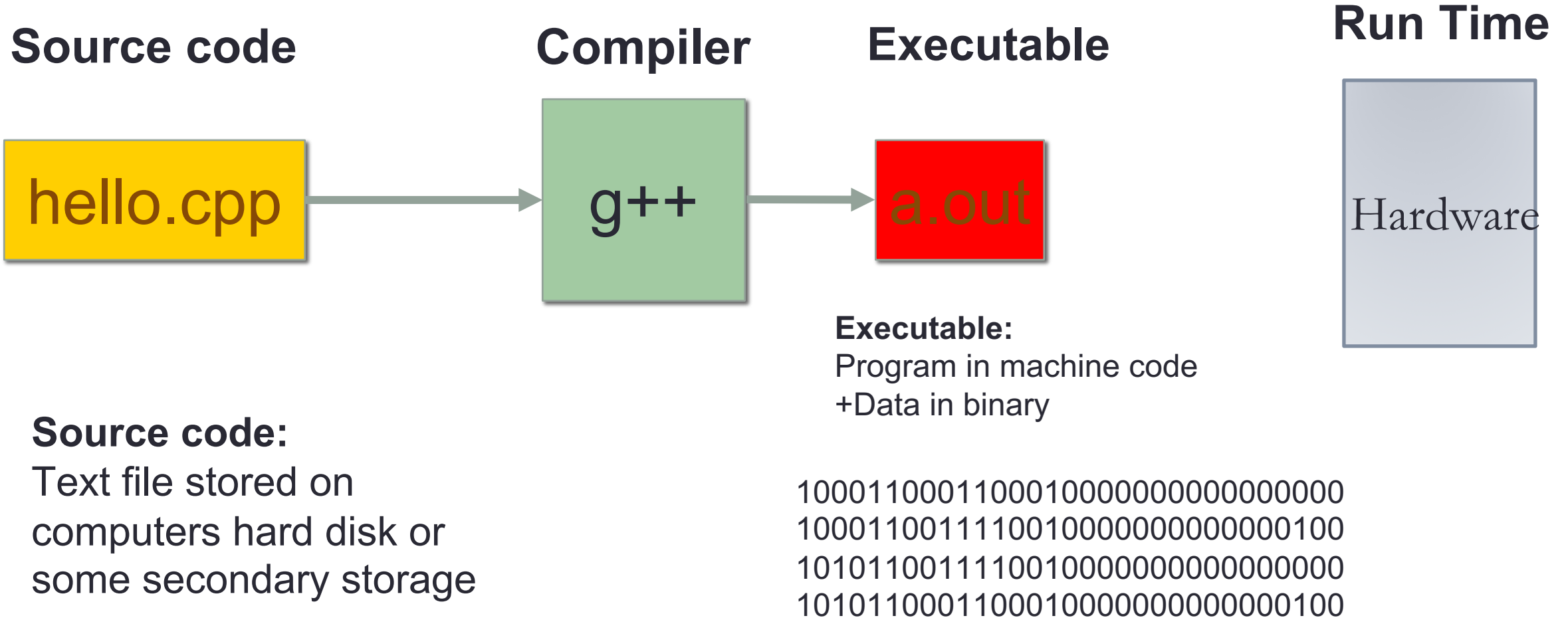
g++ is composed of a number of smaller programs

- Code written by others (libraries) can be included
- ld (linkage editor) merges one or more object files with the relevant libraries to produce a single executable

hello.cpp

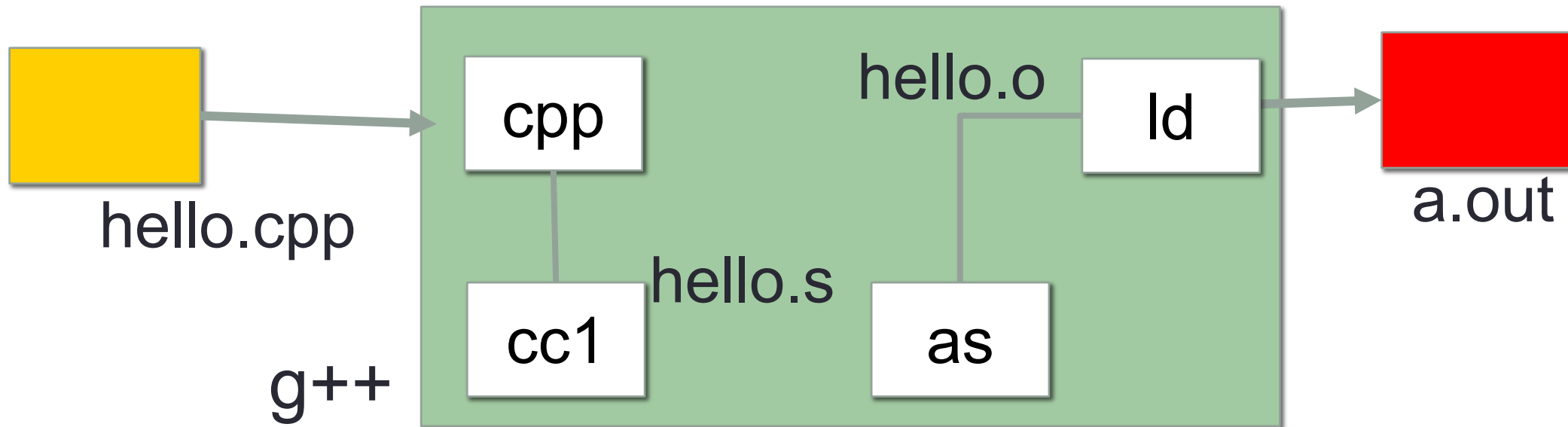cpp

cc1

hello.s

as

hello.o

ld

a.out

Library files e.g.math.o: the math library

g++
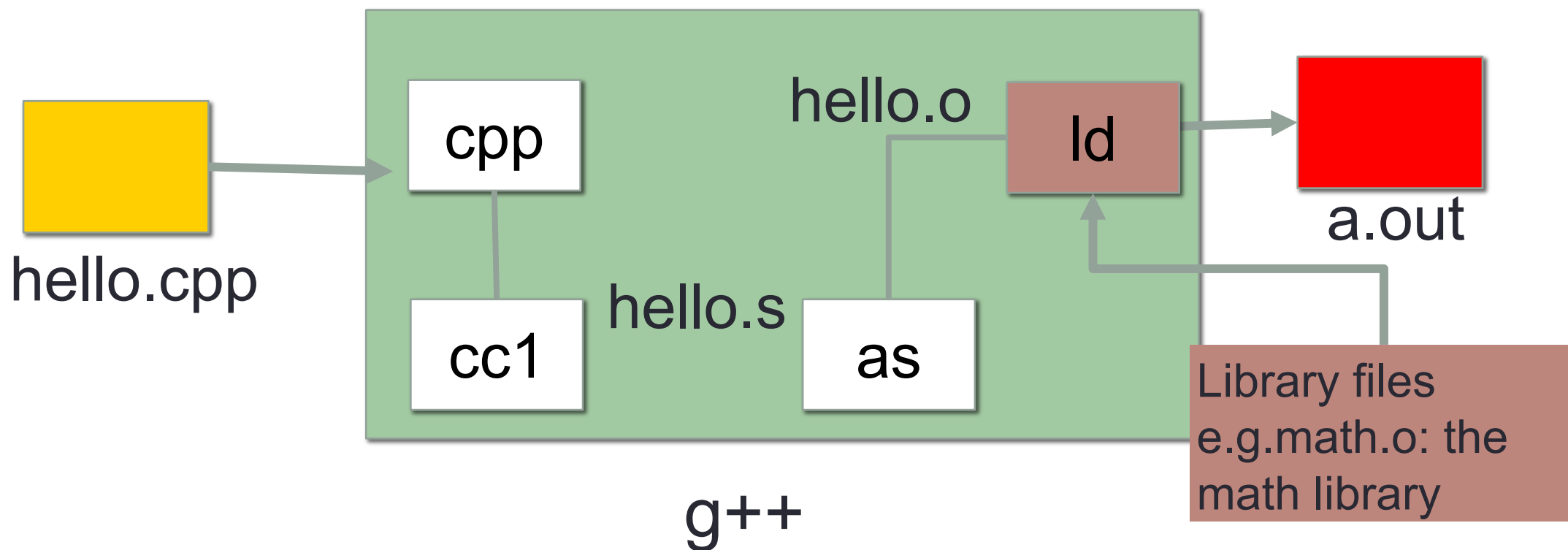
Steps in gcc

- Ask compiler to show temporary files:

$ g++ –S hello.cpp

$ g++ –c hello.o

$ g++ –o hello hello.cpp

$ g++ functions.o main.o –o myhello

hello.cpp

```
cpp        hello.o    ld
                              a.out
g++        hello.s
cc1        as
```

Make and makefiles

- The unix make program automates the compilation process as specified in a Makefile
- Specifies how the different pieces of a program in different files fit together to make a complete program
- In the makefile you provide a recipe for compilation
- When you run make it will use that recipe to compile the program

```
$ make
g++ testShapes.o shapes.o tdd.o –o testShapes
```

Specifying a recipe in the makefile

- **Comments** start with a #
- **Definitions** typically are a variable in all caps followed by an equals sign and a string, such as:

```
CXX=g++
CXXFLAGS=-Wall

BINARIES=proj1
```

```
# testShapes is the target - it is what we want to produce
# To produce the executable testShapes we need all the .o files
# Everything to the right of ":" is a dependency for testShapes

testShapes: testShapes.o shapes.o tdd.o
        #The recipe for producing the target (testshapes) is below
        g++ testShapes.o shapes.o tdd.o -o testShapes
```

Demo

- Basics of code compilation in C++ (review)
- Makefiles (used to automate compilation of medium to large projects) consisting of many files
- We will start by using a makefile to compile just a single program
- Extend to the case where your program is split between multiple files
- Understand what each of the following are and how they are used in program compilation
  - Header file (.h)
  - Source file (.cpp)
  - Object file (.o)
  - Executable
  - Makefile
  - Compile-time errors
  - Link-time errors

Writing code that works - its not magic :)

Write a function that RETURNS a string representing

an isosceles triangle with a given width

```
s = drawTriangle(5);
cout<<s;
```

```
   *
  ***
*****
```

Next time

- Files