CS197c: Programming in C++

Lecture 2

Marc Cartright

http://ciir.cs.umass.edu/~irmarc/cs197c/index.html

Syllabus

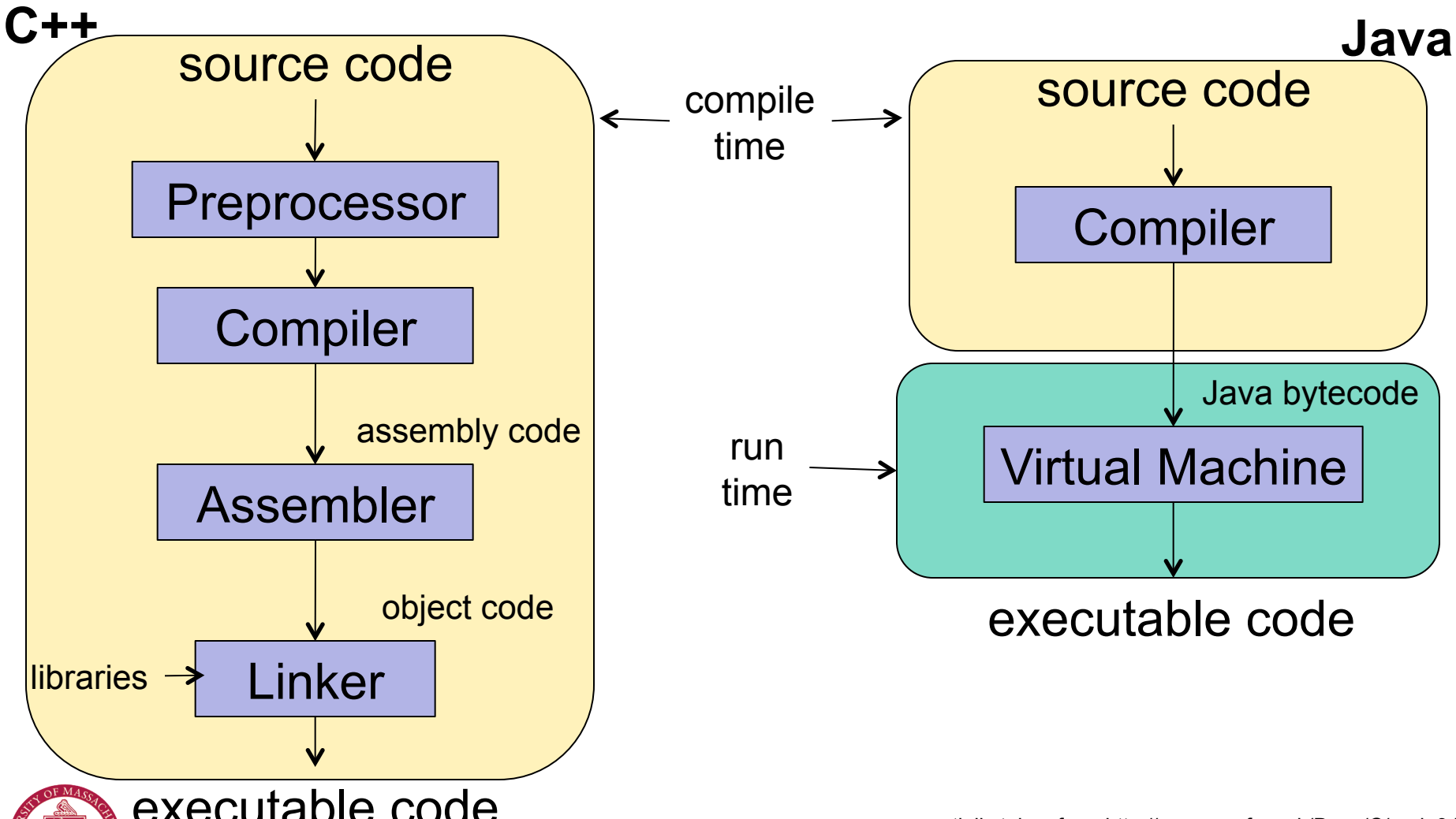
- ~~Lecture 1 : C/C++ basics, tools, Makefiles, C data types, ADTs~~
- Lecture 2 : C libraries
- Lecture 3 : Classes in C++, C++ I/O
- Lecture 4 : Memory & Pointers
- Lecture 5 : More Pointers
- Lecture 6 : Templates and the STL
- Lecture 7 : Reflection, Exceptions, C++11
- Lecture 8 : Adv. Topics: Boost and OpenGL

Today

C Libraries

Recap: Compilation Models

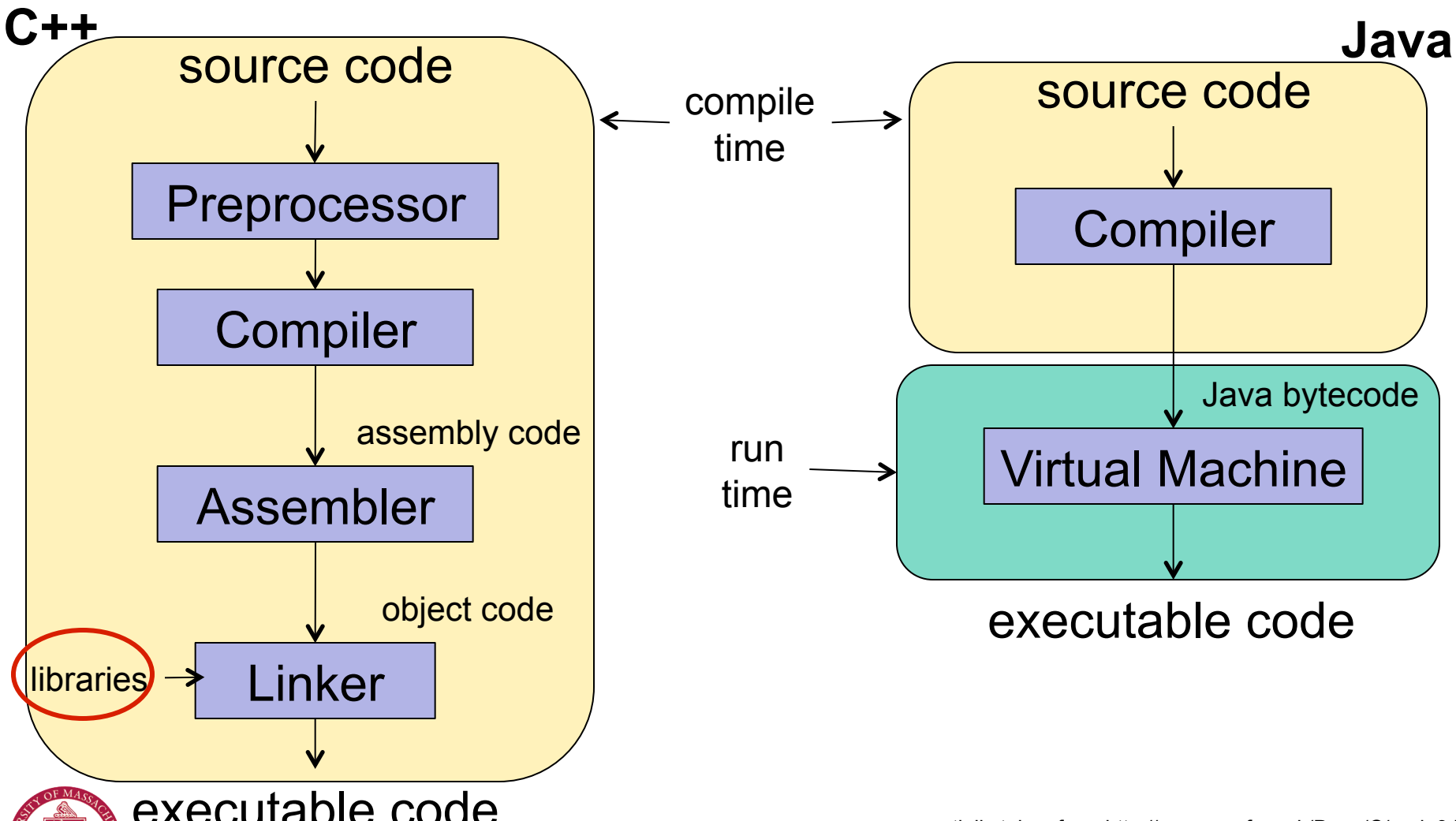
- Source code → machine executable code



partially taken from http://www.cs.cf.ac.uk/Dave/C/node3.html

Recap: Compilation Models

- Source code → machine executable code



partially taken from http://www.cs.cf.ac.uk/Dave/C/node3.html

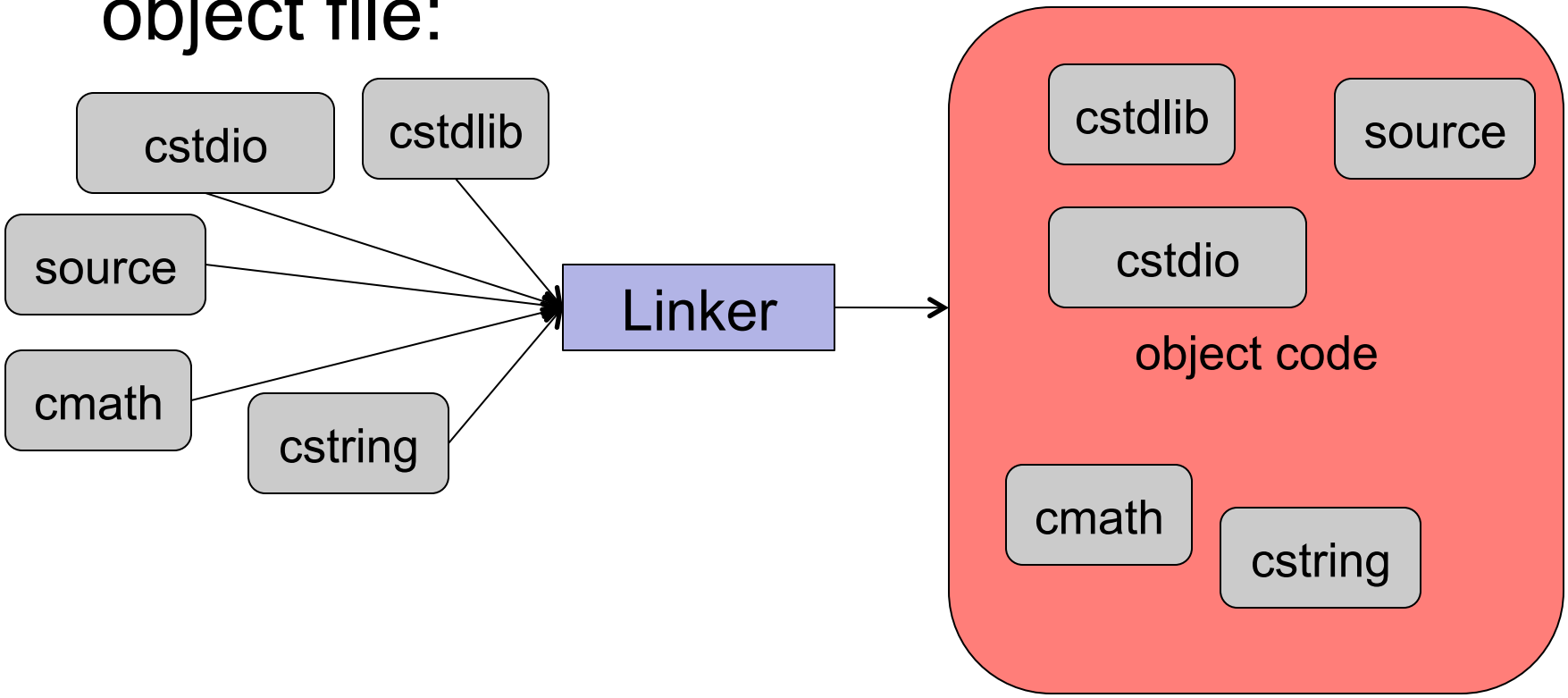
C++ Compilation Model: Linking

- "Linking" puts together separate object files to make final output (done by *link editor)*
  - program
  - another object file

- 2 types of linking: *static* and *dynamic*

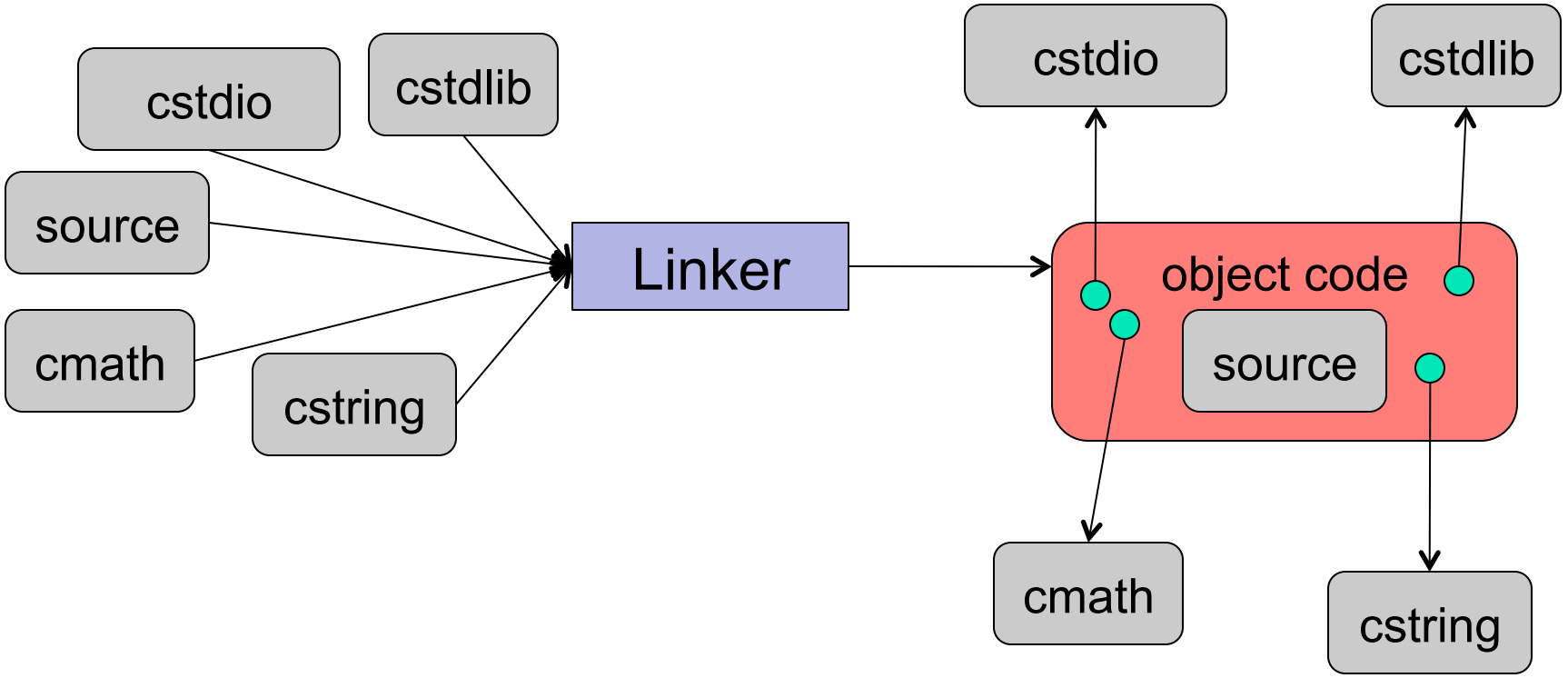
Static Linking

- All included libraries are copied into the object file:



Dynamic Linking

- Object file contains pointers to libraries, resolves at runtime:



Today's Goals:

- Understand
  - how to access libraries
  - use of headers to expose interfaces
  - what's available in the C library

Finally, some code

- Hello world !!

```
#include <cstdio>


int main(int argc, char * argv[]) {
        printf("Hello world!\n");
        return 0;
}
```

Finally, some code

- Hello world !!

Preprocessor includes header file

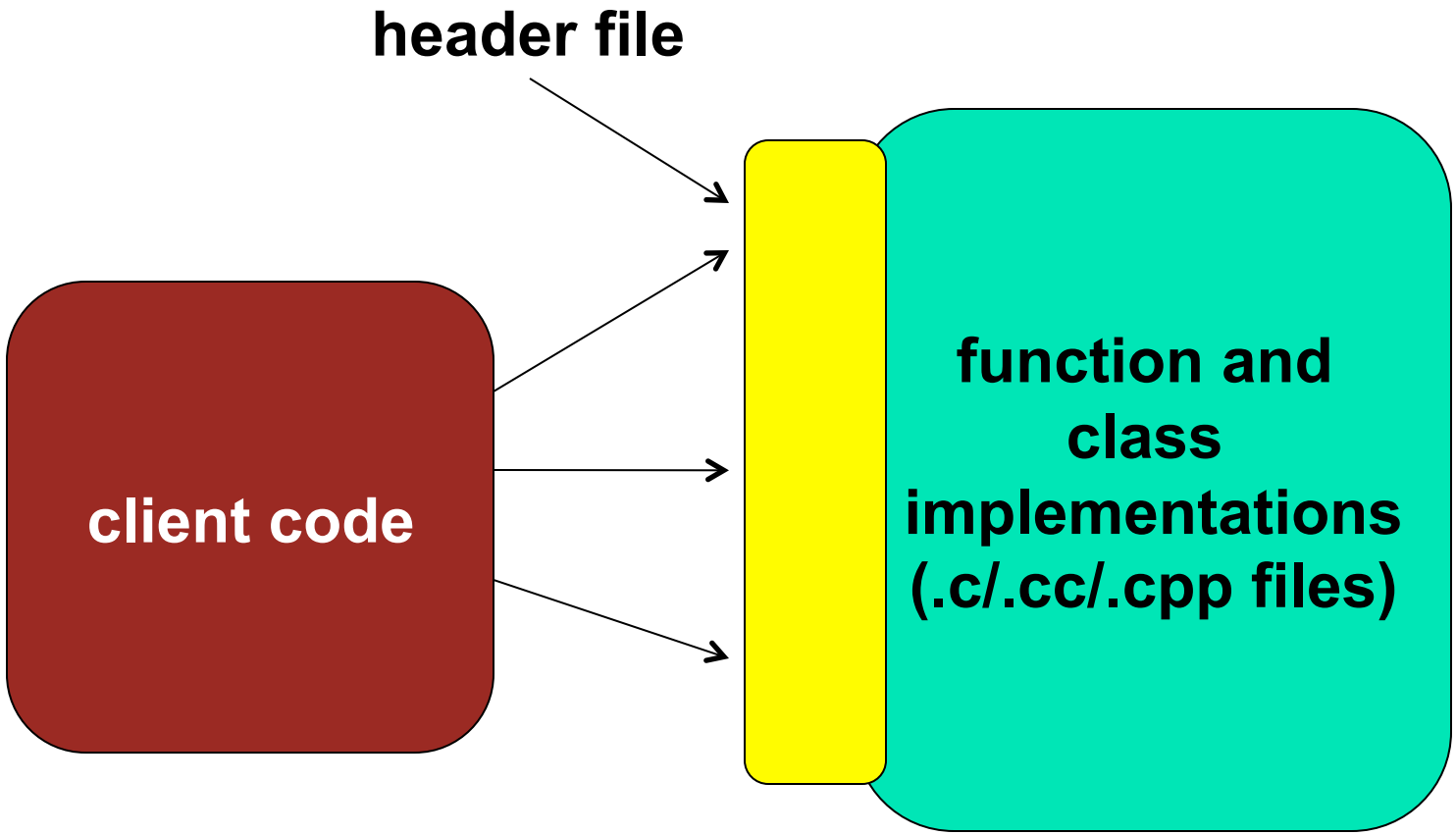
```
#include <cstdio>


int main(int argc, char * argv[]) {
        printf("Hello world!\n");
        return 0;
}
```

Components of C/C++ software

- Header files (.h in C, optional ext in C++)
  - Global declaration, class declarations
  - Included in other files
  - Preprocessor literally replaces `#include` with file contents
  - Should be guarded against multiple or recursive inclusions

- Implementation files (.c / .cc or .cpp)
  - All implementation of code elements

Header files and source files



Header files

- Sample header:

```
#ifndef MYHEADER
#define MYHEADER
float cosine(float x);

<function declarations>
<class declarations>

#endif
```

Protects against multiple inclusion

Finally, some code

- Hello world !!

```
#include <cstdio>

int main(int argc, char * argv[]) {
        printf("Hello world!\n");
        return 0;
}
```

Main function for C/C++

Main

```
void main()
int main()
int main(int argc, char * argv[])
```

- Serves as entry point to program
- 1st variant takes no arguments, returns no status
- 2nd variant takes no args but returns status
- 3rd accepts args and returns status
  - Most widely used, no harm in doing it

Finally, some code

- Hello world !!

```
#include <cstdio>


int main(int argc, char * argv[]) {
        printf("Hello world!\n");
        return 0;
}
```

Arguments from cmd line

Command-Line Arguments

- Similar to Java, but not quite
  - 1st argument is program call
  - Arguments come as C-strings

```
>> ./myprogram input1.txt 5 “holy cow!”

int main(int argc, char *argv[]) {
...
}
```

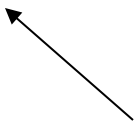
argc = 4

argv[0] = “./myprogram”
argv[1] = “input1.txt”
argv[2] = “5”
argv[3] = “holy cow!”

Finally, some code

- Hello world !!

```
#include <cstdio>


int main(int argc, char * argv[]) {
    printf("Hello world!\n");
    return 0;
}
```



prints to stdout

I/O in C

- C is function-based
  - printf(<format string>, <args>+);
  - scanf(<format string>, <receivers>+);
- Specify what the output should look like, then arguments are swapped in

```
printf("I'm only going to count to %d.\n", 3);

printf("My name is %s.\n", "HAL");
```

newline

Finally, some code

- Hello world !!

```
#include <cstdio>


int main(int argc, char * argv[]) {
    printf("Hello world!\n");
    return 0;
}
```

returns status value – simple way to communicate with calling process

The C library

- Popular libraries
  - <cstdio>
  - <cstdlib>
  - <cstring>
  - <cmath>
  - <climits>
- Obscure
  - all the rest

Reference: http://en.wikipedia.org/wiki/C_standard_library

<cstdio>

- Contains functions for input and output
  - printf/scanf probably most commonly used
  - fprintf is the file-based variant
  - sprintf prints TO a string (like a StringBuilder)

- When playing with files:
  - fopen/fclose
  - fflush, fseek ftell, feof, and so on

<cstdlib>

- Collection of utility functions and constants
  - defines the `NULL` value
- C-string ←→ numeric conversions
  - atoi, atol, strtol, atof, strtod
- Random numbers
  - rand, srand
- C-Memory handling (skipping)
- Process control
  - exit, abort, getenv, system

<cstring>

- Collection of string manipulation functions

- Popular functions
  - `strlen` : determine the length of a C string
  - `strcmp` : lexicographically compare two C strings

<cmath>

- Collection of mathematical functions and constants
  - Constants: `M_E`, `M_PI`, `M_SQRT2`
  - Functions: `cos`, `sin`, `log10`, `pow(x,y)`, etc.

<climits>

- Collection of limiting constants for the data types
  - `LONG_MIN`, `LONG,MAX`, `INT_MIN`, `INT_MAX`
- Necessary because different systems implement data types differently
  - `LONG_MAX` on 32-bit compiler: 2^{31}
  - `LONG_MAX` on 64-bit compiler: 2^{63}

Other libraries

- Provide other utility functions, worth at least reading over
- Wikipedia a good reference
- Every computer has the C libraries somewhere on it

Walkthrough

C Library example

Next lecture

Into C++: Classes