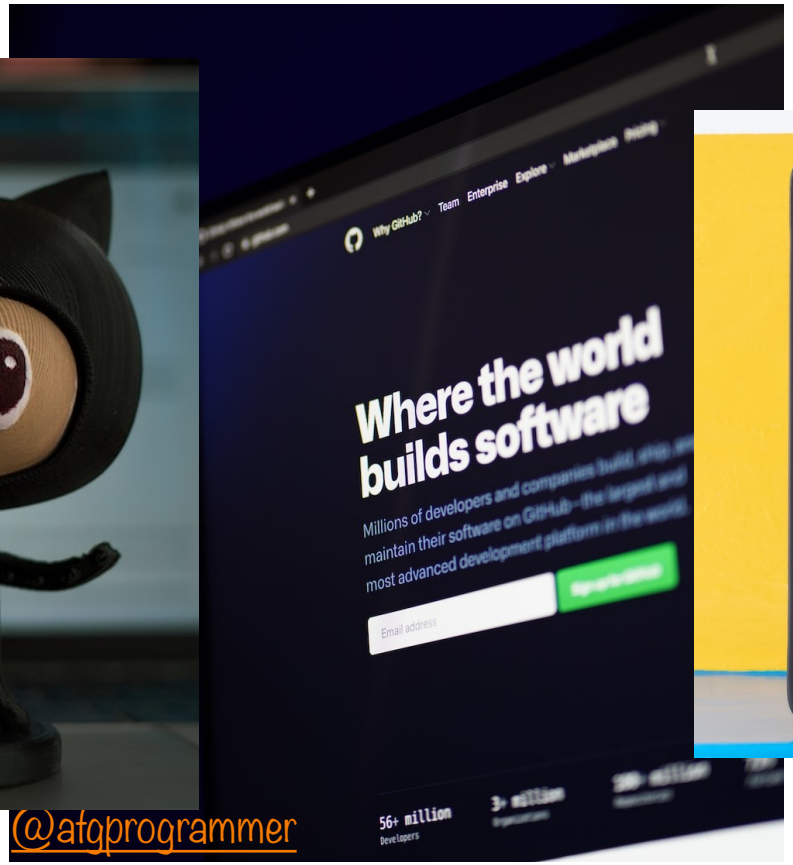
Git and GitHub … then C



Agenda

Our computing environment

- Precept: Linux and Bash
- Today: git and GitHub

A taste of C (compared with Java)

- History of C
- Building and running C programs
- Characteristics of C

Revision Control Systems

Problems often faced by programmers:

- I've deleted my code! How do I get it back?
- Try one way of writing these functions, and go back to the other if it doesn't work
- What things have I changed since the last working version?
- How do I work with source code on multiple computers?

- How do I work with others (e.g., a COS 217 partner) on the same program?
- What changes did my partner just make?
- My partner and I make changes to different parts of a program; how do we merge those changes?

All solved by revision control tools, e.g. `git`

Repository vs. Working Copy

WORKING COPY

- Represents one version of the code
- Plain files (e.g, .c)
- Make a coherent set of modifications, then *commit* this version of code to the repository
- Best practice: write a meaningful *commit message*

`git commit`

`git checkout`‡

REPOSITORY (or “repo”)

- Contains all checked-in versions of the code
- Specialized format, located in .git directory
- Can view commit history
- Can diff any versions
- Can *check out* any version, by default the most recent (known as HEAD)

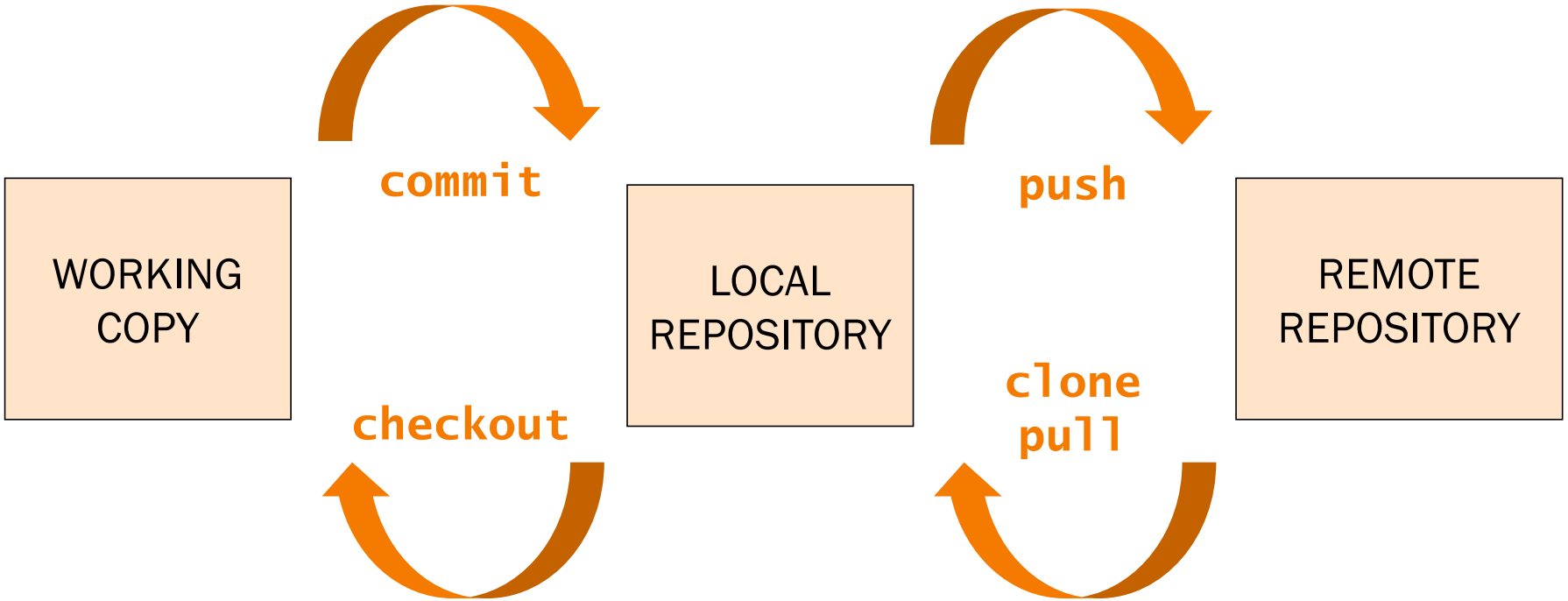
‡ We'll rarely use checkout except to throw away local changes (see slide 6)

Information Content of Git Comments

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

https://xkcd.com/1296/

Another Level of Indirection



Local vs. Remote Repositories

LOCAL REPOSITORY

- Located in .git directory
- Only accessible from the computer where it lives
- Commit early, commit often: you can only go back to versions you've committed
- Can *push* current state (i.e., complete committed history) of a local repo to remote repo

REMOTE REPOSITORY

- Located in the cloud
  E.g., github.com
- Can *clone* remote repo into local repo + working copy on multiple machines
- Any clone can *pull* the current state from remote repo

COS 217 🧡 GitHub

We distribute assignment code through a github.com repo

But we're not very nice

- You can't push code to our repo
- You should create your own (private) repo for each assignment
  - Two methods for this in git primer handout
- Then create clones of that repo for yourself
  - Create one clone on armlab, to test and submit
  - If developing on your own machine, create another clone there
  - Commit from working copy to local dev clone, push from it "up" to github, then pull "down" onto armlab, before testing and submitting

Agenda

Our computing environment

- Lecture 1 and Precepts 1 and 2: Linux and Bash
- Lecture 2: git

A taste of C (compared with Java)

- History of C
- Building and running programs
- Characteristics of C

The C Programming Language

Who? Dennis Ritchie

When? ~1972

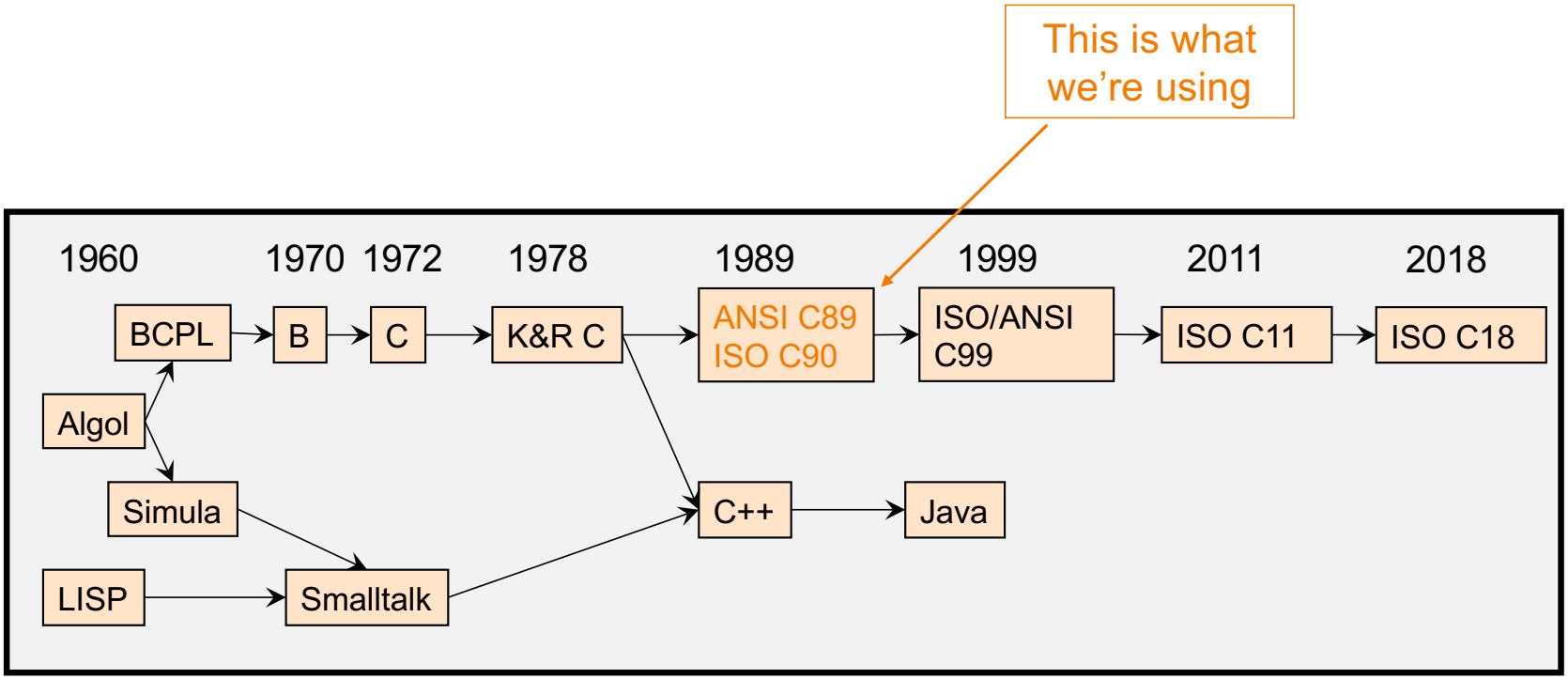
Where? Bell Labs

Why? Build the Unix OS

Read more history:

https://www.bell-labs.com/usr/dmr/www/chist.html

History of C and Java



C vs. Java: Design Goals

C Design Goals (1972)	Java Design Goals (1995)
Build the Unix OS	Language of the Internet
Low-level; close to HW and OS	High-level; insulated from hardware and OS
Good for system-level programming	Good for application-level programming
Support structured programming	Support object-oriented programming
Unsafe: don't restrict the programmer much	*Safe*: can't step "outside the sandbox"
	Look like C

Agenda

Our computing environment

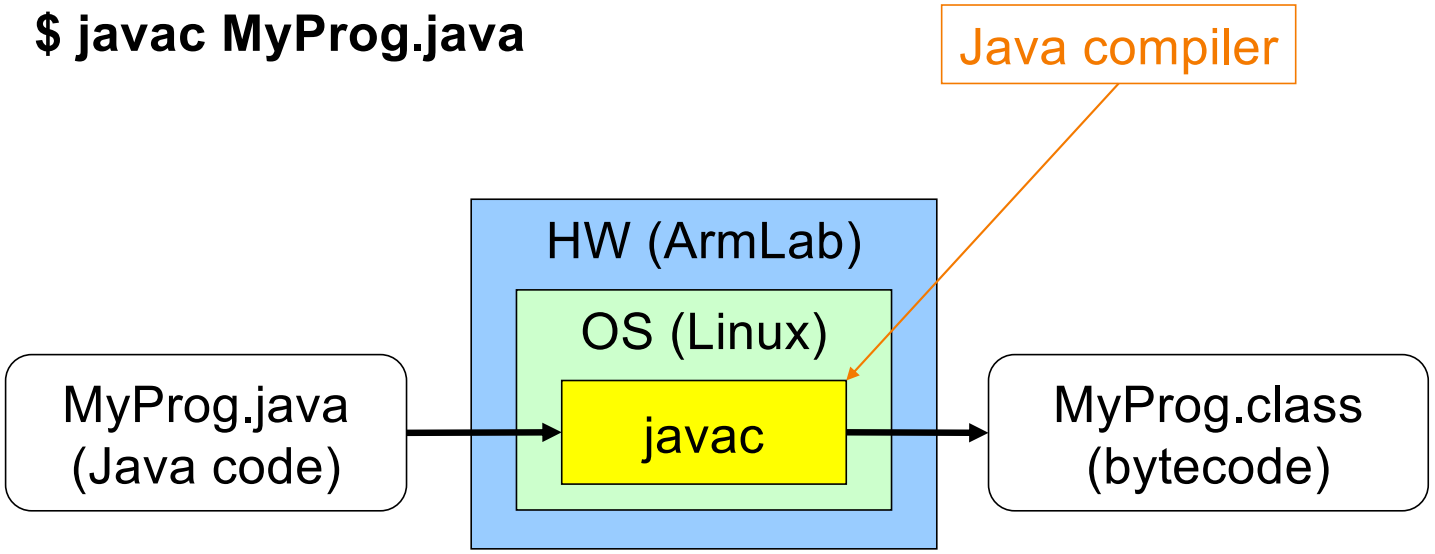
- Lecture 1 and Precepts 1 and 2: Linux and Bash
- Lecture 2: git

A taste of C (compared with Java)

- History of C
- Building and running C programs
- Characteristics of C

Building Java Programs

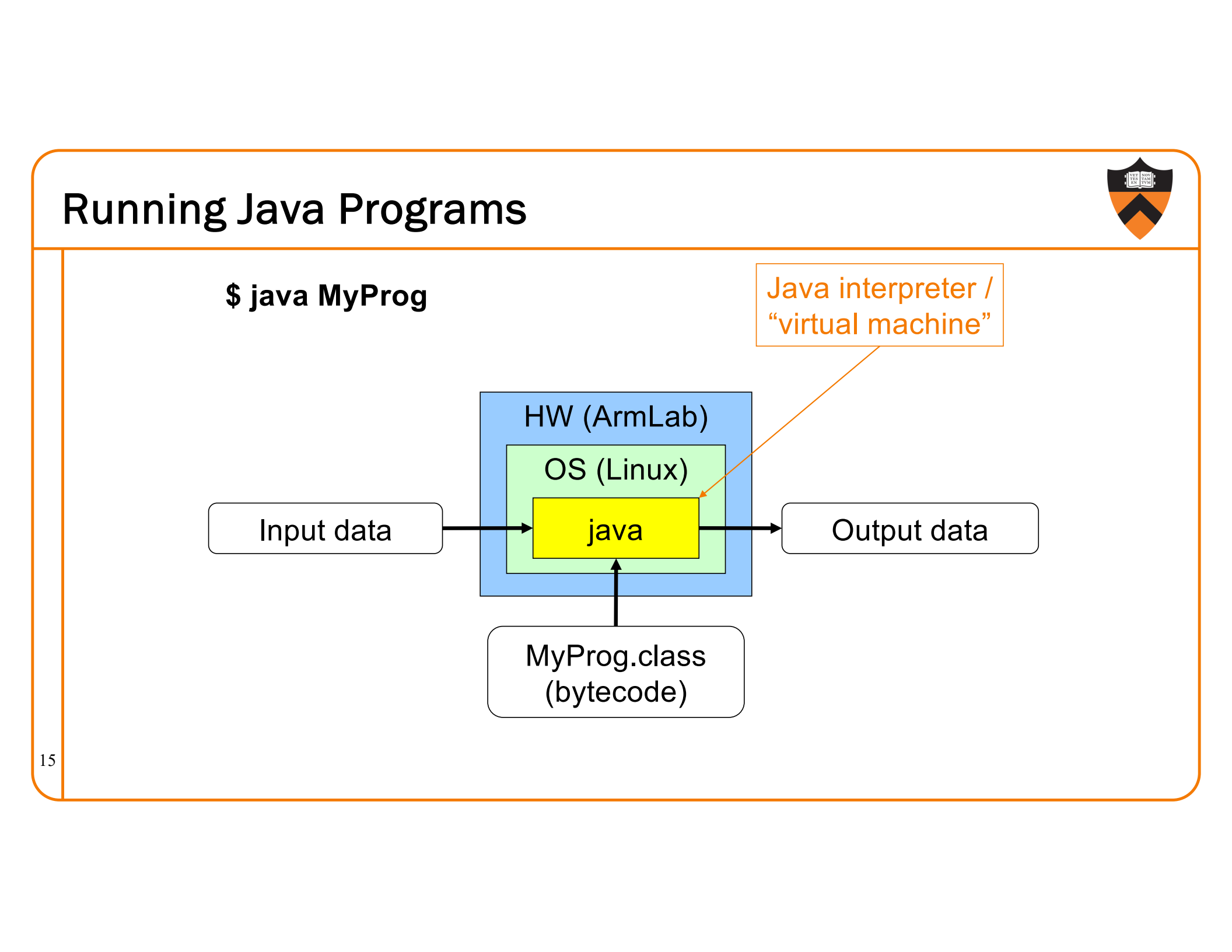
$ javac MyProg.java



Myprog.class is bytecode, which is machine independent and runs on an interpreter, which is a software layer that runs on the hardware

- The bytecode is an input to the interpreter software

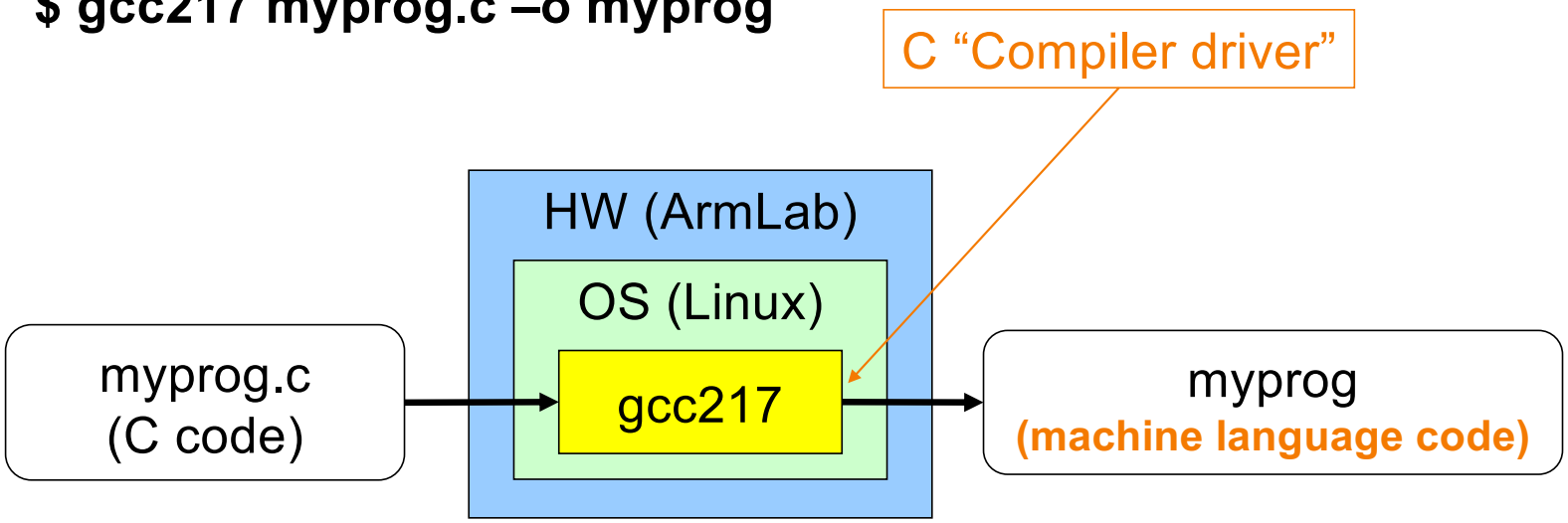
Running Java Programs



$ java MyProg

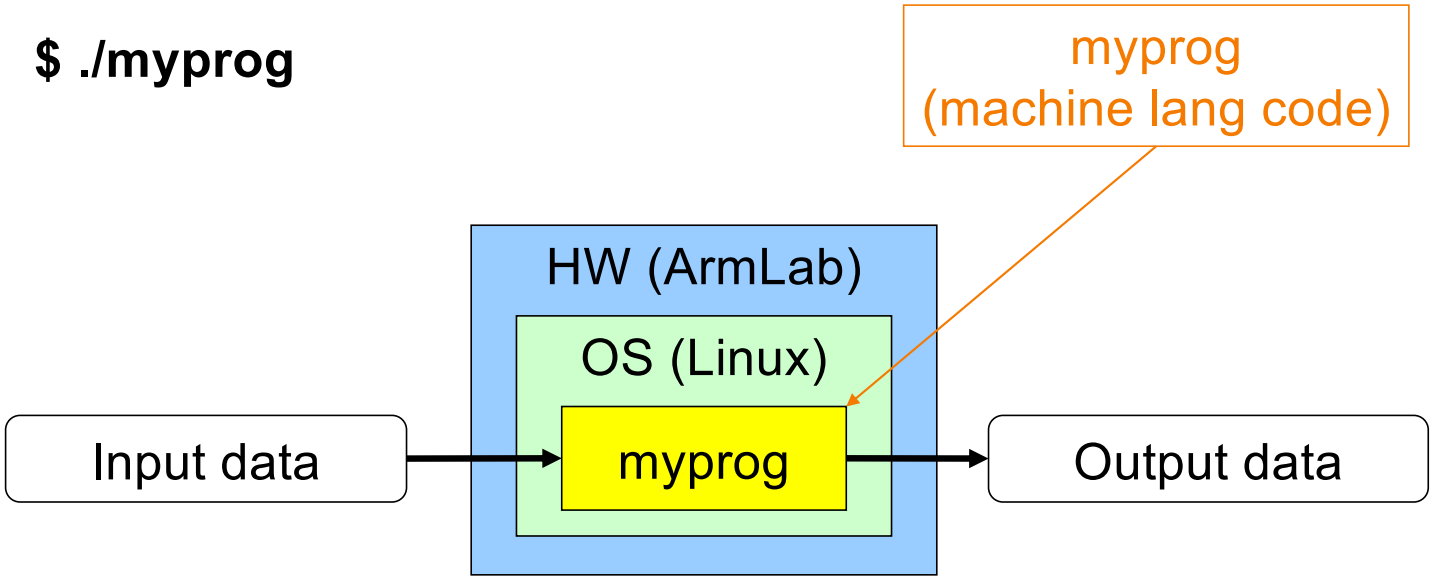
Building C Programs

$ gcc217 myprog.c –o myprog



Myprog is machine language, which is machine dependent and runs on the raw hardware (unlike Java’s bytecode)

Running C Programs



Agenda

Our computing environment

- Lecture 1 and Precepts 1 and 2: Linux and Bash
- Lecture 2: git

A taste of C (compared with Java)

- History of C
- Building and running C programs
- Characteristics of C

Java vs. C: Portability

Program	Code Type	Portable?
MyProg.java	Java source code	Yes
myprog.c	C source code	Mostly*
MyProg.class	Bytecode	Yes**
myprog	Machine lang code	No

Conclusion: Java programs are more portable

* COS 217 has used *many* architectures over the years, and every time we've switched, all our programs have had to be recompiled)

** Java interpreter provides a consistent interface across OSes and hardware, though its implementation is different across them

Java vs. C: Safety & Efficiency

Java does a lot more checking

- `null` reference checking (crash)
- Automatic array-bounds checking (crash)
- Automatic memory management (garbage collection)
- Other safety features

C

- NULL pointer checking (crash)
- Manual bounds checking (keep going ...)
- Manual memory management

Conclusion 1: Java is often safer than C

Conclusion 2: Java is often slower than C

Java vs. C: Details

Next 7 slides show C details through comparisons with the Java you know.

We're going to mostly skip them in lecture. You can read them later

- This is largely syntax mapping. But it prepares us for the more significant differences later

But we will look at an example program

Java vs. C: Details

	Java	C
Overall Program Structure	Hello.java: public class Hello { public static void main (String[] args) { System.out.println("hello, world"); } }	hello.c: #include <stdio.h> int main(void) { printf("hello, world\n"); return 0; }
Building	$ javac Hello.java	$ gcc217 hello.c –o hello
Running	$ java Hello hello, world $	$ ./hello hello, world $

Java vs. C: Details

	Java	C
Character type	`char  // 16-bit Unicode`	`char /* 8 bits */`

Integral types	`byte  // 8 bits`
	`short  // 16 bits`
	`int  // 32 bits`
	`long  // 64 bits`
		`(unsigned, signed) char`
		`(unsigned, signed) short`
		`(unsigned, signed) int`
		`(unsigned, signed) long`

Floating point types	`float  // 32 bits`
	`double  // 64 bits`
		`float`
		`double`
		`long double`

Logical type	`boolean`	`/* no equivalent */` `/* use 0 and non-0 */`
Generic pointer type	`Object`	`void*`

Constants	`final int MAX = 1000;`	`#define MAX 1000`
		`const int MAX = 1000;`
		`enum {MAX = 1000};`

Java vs. C: Details

	Java	C

Arrays	`int [] a = new int [10];`
	`float [][] b =`
	`  new float [5][20];`
		`int a[10];`
		`float b[5][20];`

Array bound checking	`// run-time check`	`/* no run-time check */`
Pointer type	`// Object reference is an` `// implicit pointer`	`int *p;`

Record type	`class Mine`
	`{  int x;`
	`  float y;`
	`}`
		`struct Mine`
		`{  int x;`
		`  float y;`
		`};`

Java vs. C: Details

	Java	C

Strings	`String s1 = "Hello";`
	`String s2 =`
	`  new String("hello");`
		`char *s1 = "Hello";`
		`char s2[6];`
		`strcpy(s2, "hello");`

String concatenation	`s1 + s2` `s1 += s2`	`#include <string.h>` `strcat(s1, s2);`
Logical ops *	`&&, ||, !`	`&&, ||, !`
Relational ops *	`==, !=, <, >, <=, >=`	`==, !=, <, >, <=, >=`
Arithmetic ops *	`+, -, *, /, %, unary -`	`+, -, *, /, %, unary -`
Bitwise ops	`<<, >>, >>>, &, ^, |, ~`	`<<, >>, &, ^, |, ~`
Assignment ops	`=, +=, -=, *=, /=, %=,` `<<=, >>=, >>>=, &=, ^=, |=`	`=, +=, -=, *=, /=, %=,` `<<=, >>=, &=, ^=, |=`

* Essentially the same in the two languages

Java vs. C: Details

	Java	C

if stmt *	`if (i < 0)`
	`  statement1;`
	`else`
	`  statement2;`
		`if (i < 0)`
		`  statement1;`
		`else`
		`  statement2;`

switch stmt *	`switch (i)`
	`{  case 1:`
	`  ...`
	`  break;`
	`  case 2:`
	`  ...`
	`  break;`
	`  default:`
	`  ...`
	`}`
		`switch (i)`
		`{  case 1:`
		`  ...`
		`  break;`
		`  case 2:`
		`  ...`
		`  break;`
		`  default:`
		`  ...`
		`}`

goto stmt	`// no equivalent`	`goto someLabel;`

* Essentially the same in the two languages

Java vs. C: Details

	Java	C

for stmt	`for (int i=0; i<10; i++)`
	`  statement;`
		`int i;`
		`for (i=0; i<10; i++)`
		`  statement;`

while stmt *	`while (i < 0)` `  statement;`	`while (i < 0)` `  statement;`

do-while stmt *	`do`
	`  statement;`
	`while (i < 0)`
		`do`
		`  statement;`
		`while (i < 0);`

continue stmt *	`continue;`	`continue;`
labeled continue stmt	`continue someLabel;`	`/* no equivalent */`
break stmt *	`break;`	`break;`
labeled break stmt	`break someLabel;`	`/* no equivalent */`

* Essentially the same in the two languages

Java vs. C: Details

	Java	C
return stmt *	`return 5;` `return;`	`return 5;` `return;`

Compound stmt (alias block) *	`{`
	`  statement1;`
	`  statement2;`
	`}`
		`{`
		`  statement1;`
		`  statement2;`
		`}`

Exceptions	`throw, try-catch-finally`	`/* no equivalent */`
Comments	`/* comment */` `// another kind`	`/* comment */`

Method / function call	`f(x, y, z);`
	`someObject.f(x, y, z);`
	`SomeClass.f(x, y, z);`
		`f(x, y, z);`

* Essentially the same in the two languages

Coming up next …

Character processing, structured exactly how we'll want you to design your Assignment 1 solution

Read the A1 specs soon: you'll be ready to start after Lecture 3