CS 61C:
Great Ideas in Computer Architecture
Lecture 2: *Introduction to C, Part I*

Instructor: Sagar Karandikar

sagark@eecs.berkeley.edu

http://inst.eecs.berkeley.edu/~cs61c/

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

Note about textbooks

- Textbooks: Average 15 pages of reading/week (can rent!)
  - Patterson & Hennessey, *Computer Organization and Design*, 5/e
    (we'll also provide Revised 4th Ed pages, not Asian version 4th edition)
  - Kernighan & Ritchie, *The C Programming Language*, 2nd Edition
  - Barroso & Holzle, *The Datacenter as a Computer, 2nd Edition*

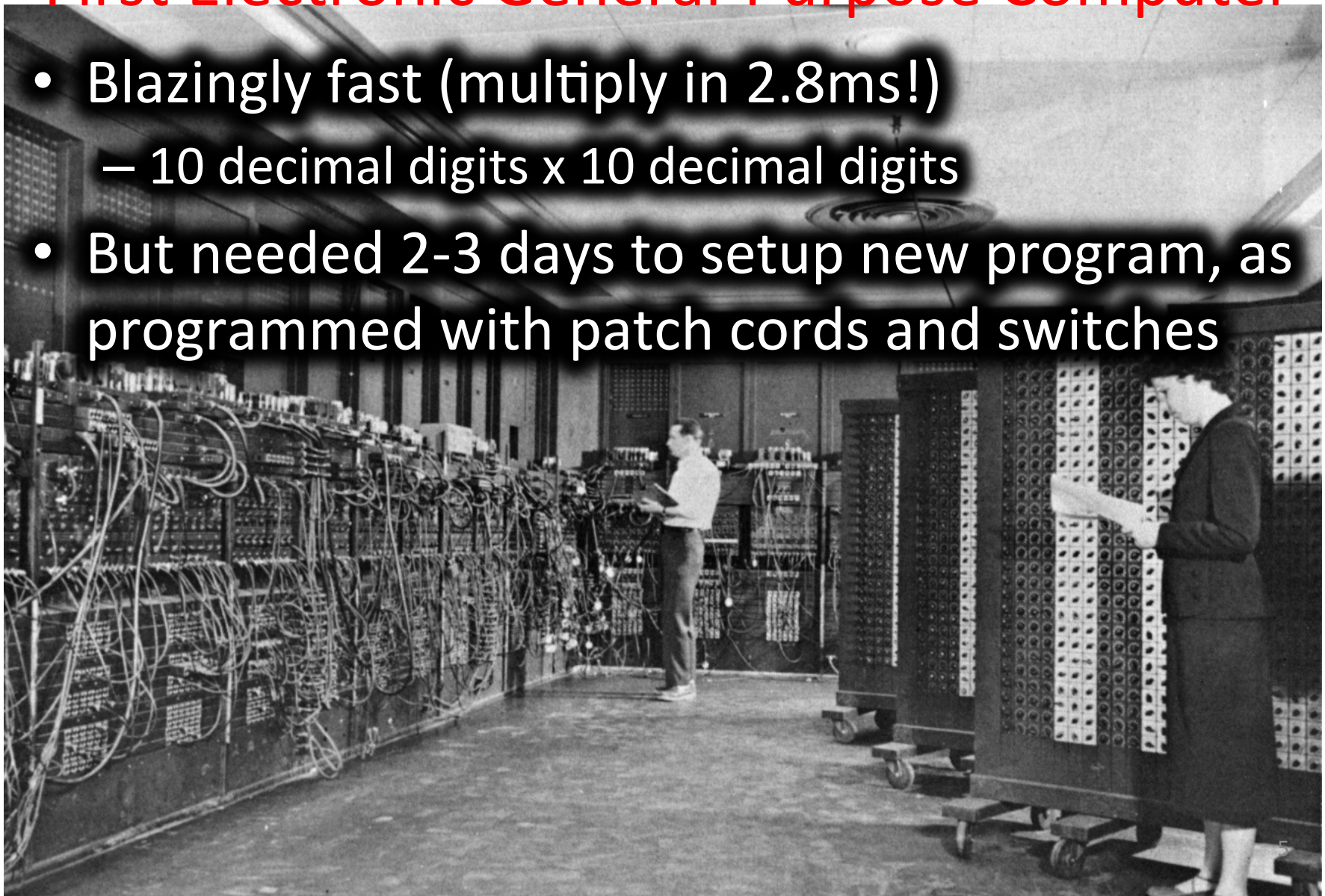
Agenda

- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

Agenda

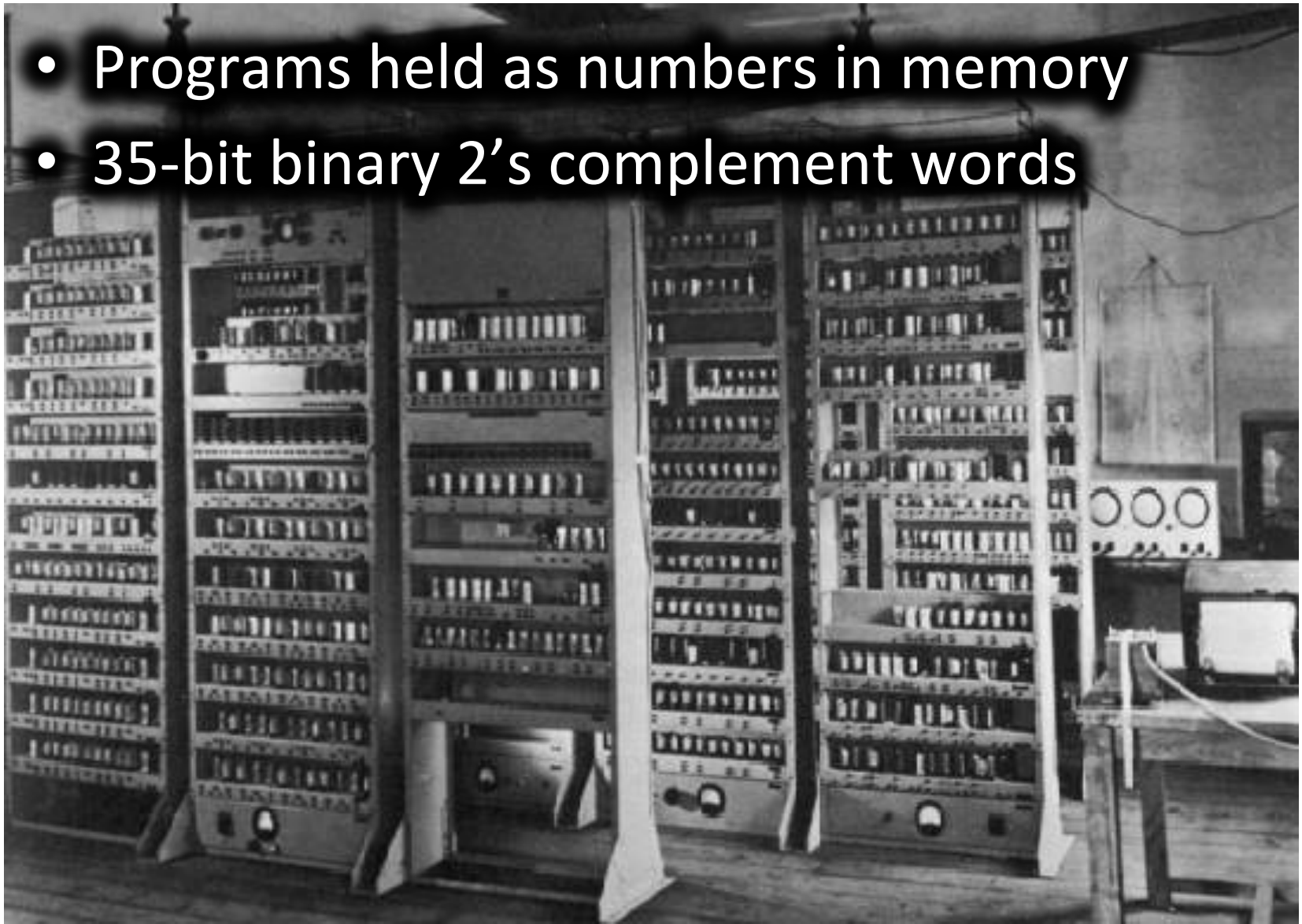
- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

ENIAC (U.Penn., 1946)
First Electronic General-Purpose Computer

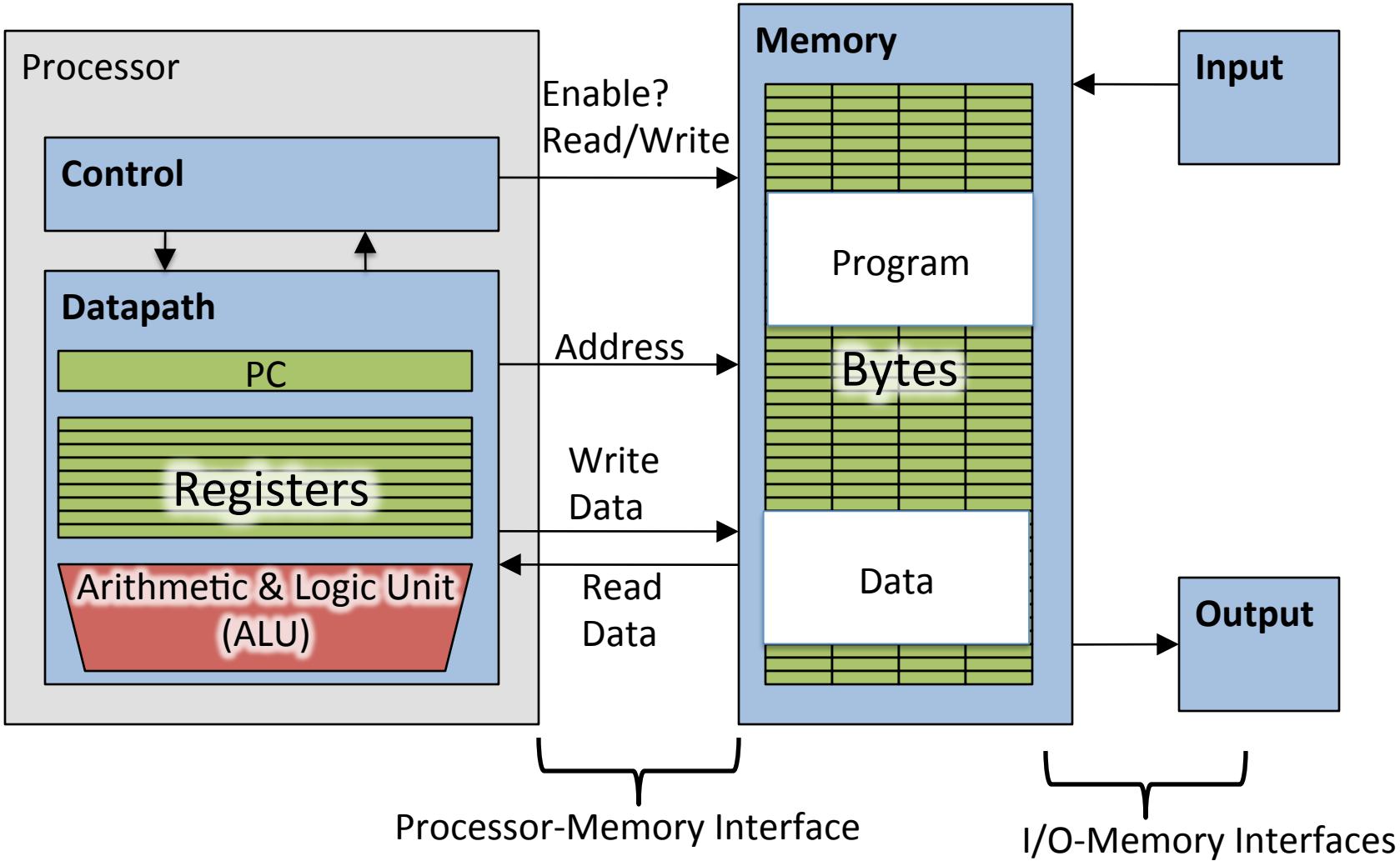


- Blazingly fast (multiply in 2.8ms!)
  - 10 decimal digits x 10 decimal digits
- But needed 2-3 days to setup new program, as programmed with patch cords and switches

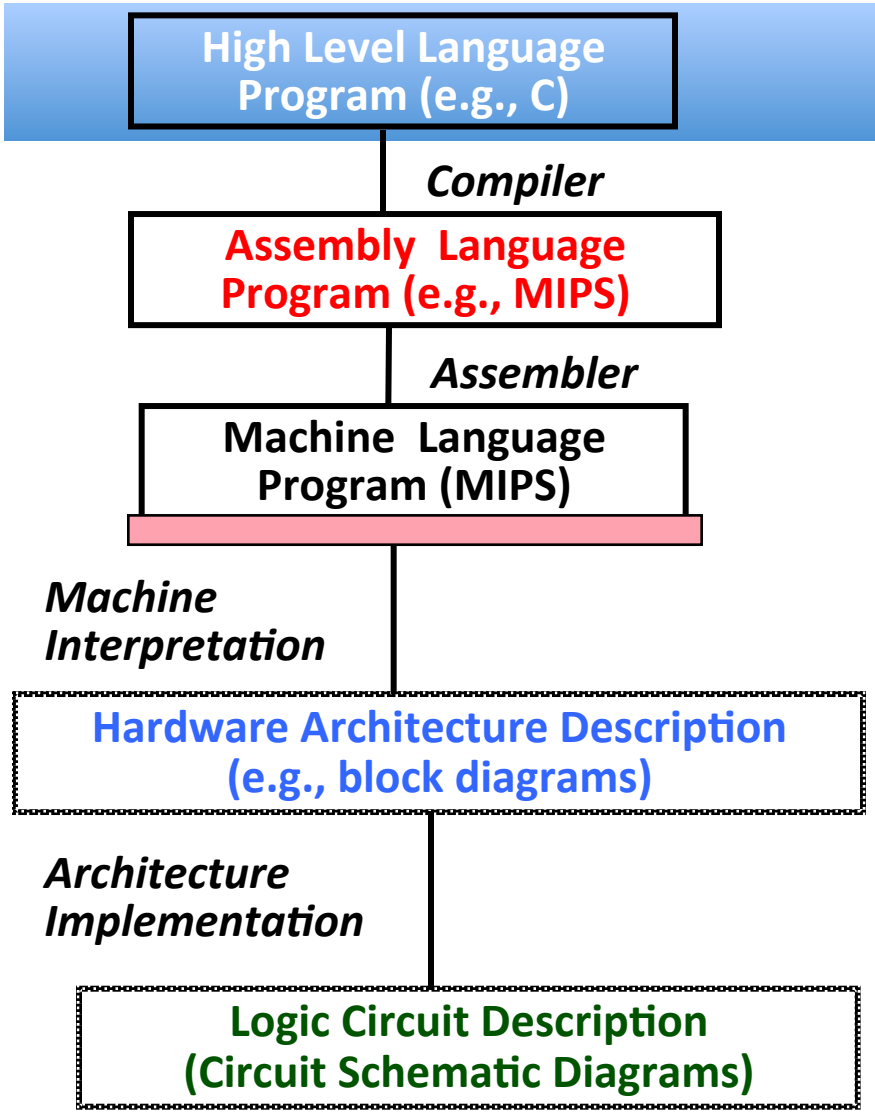
EDSAC (Cambridge, 1949)
First General Stored-Program Computer



Components of a Computer



Great Idea: Levels of Representation/ Interpretation



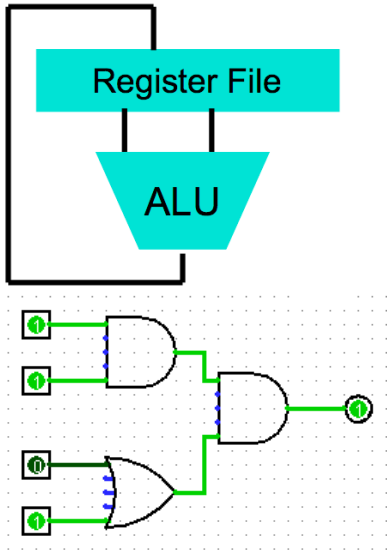

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

We are here!

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

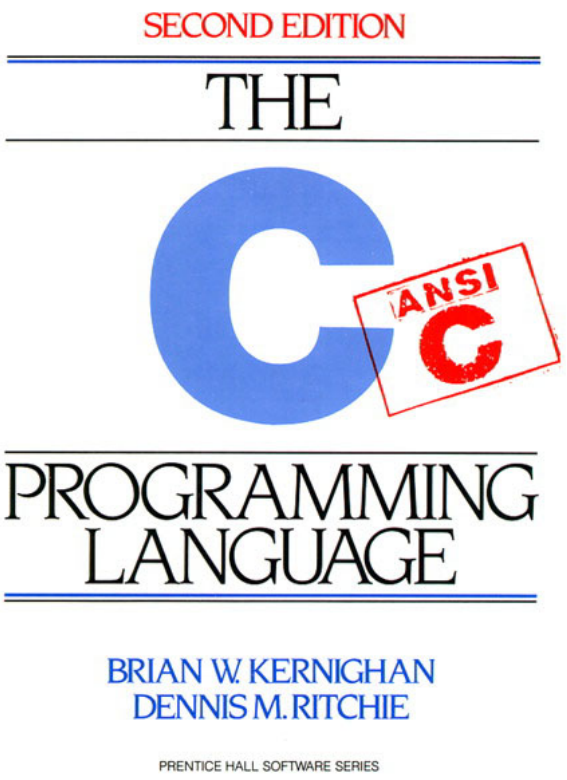
Anything can be represented as a *number*, i.e., data or instructions

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



Introduction to C
"The Universal Assembly Language"

- "Some" experience is required before CS61C
  C++ or Java OK



- Class pre-req included classes teaching Java
- Python used in two labs
- C used for everything else

Language Poll!

Please raise hand for *first* one of following you can say yes to

- ☐ I have programmed in C, C++, C#, or Objective-C
- ☐ I have programmed in Java
- ☐ I have programmed in FORTRAN, Cobol, Algol-68, Ada, Pascal, or Basic
- ☐ None of the above

Intro to C

- *C is not a "very high-level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.*

  – Kernighan and Ritchie

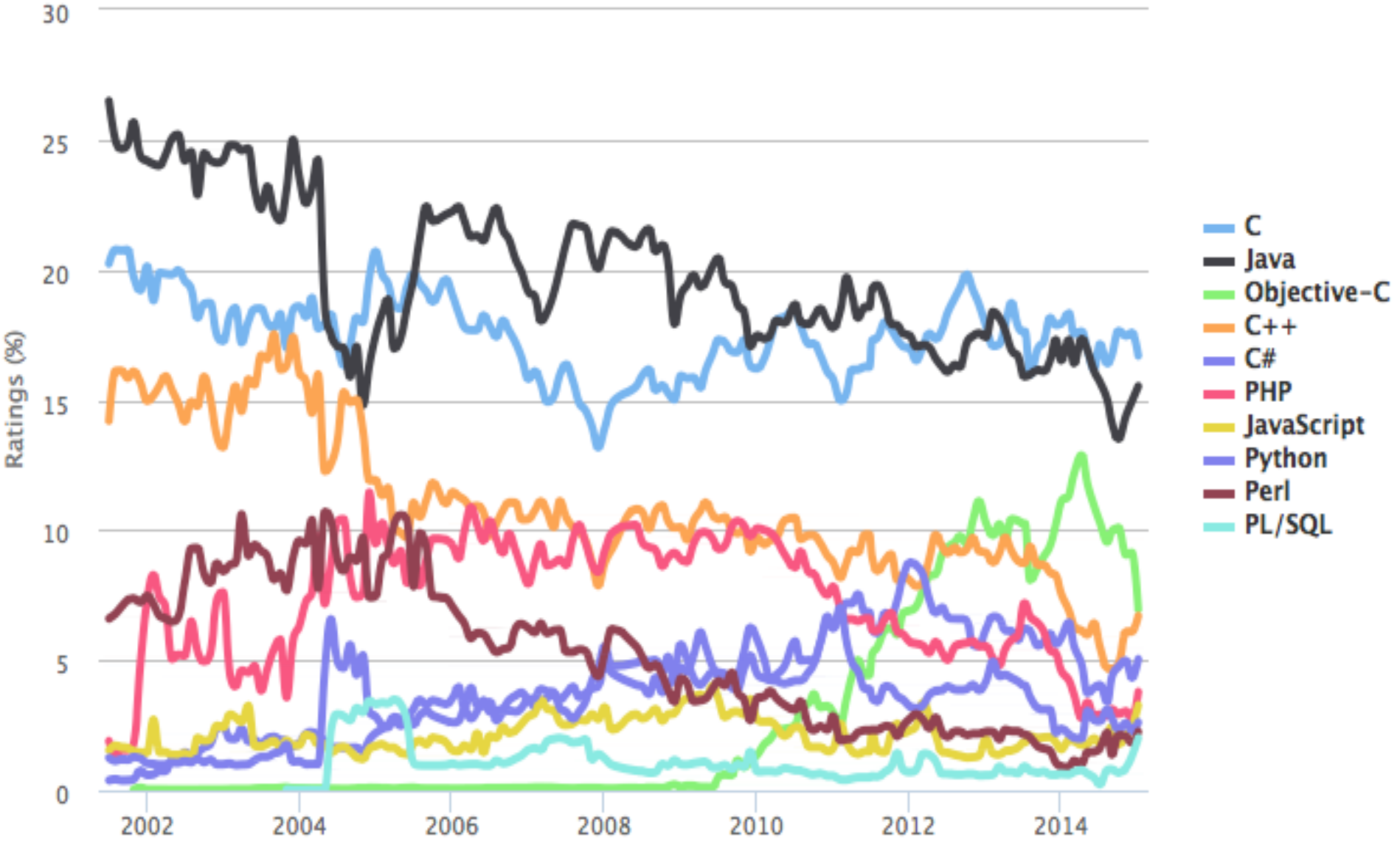
- Enabled first operating system not written in assembly language: *UNIX* - A portable OS!
- C and derivatives (C++/Obj-C/C#) still one of the most popular application programming languages after >40 years!

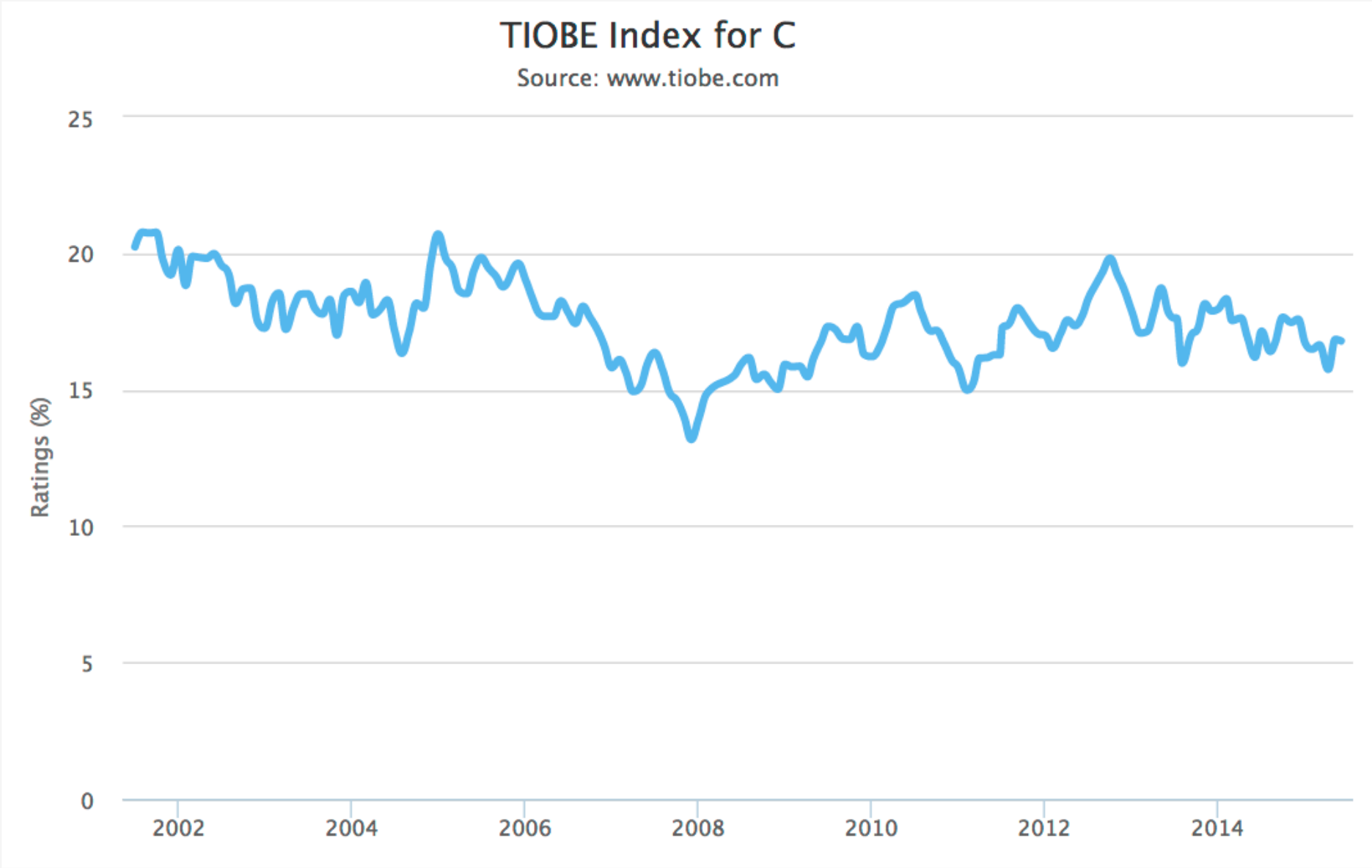
TIOBE Index of Language Popularity

Jan 2015	Jan 2014	Change	Programming Language	Ratings	Change
1	1		C	16.703%	-1.24%
2	2		Java	15.528%	-1.00%
3	**3**		**Objective-C**	**6.953%**	**-4.14%**
4	4		C++	6.705%	-0.86%
5	5		C#	5.045%	-0.80%
6	6		PHP	3.784%	-0.82%
7	9	^	JavaScript	3.274%	+1.70%
8	8		Python	2.613%	+0.24%
9	13	^^	Perl	2.256%	+1.33%
10	17	^^	PL/SQL	2.014%	+1.38%

http://www.tiobe.com

TIOBE Programming Community Index




TIOBE Index for C
Source: www.tiobe.com
Ratings (%)
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014

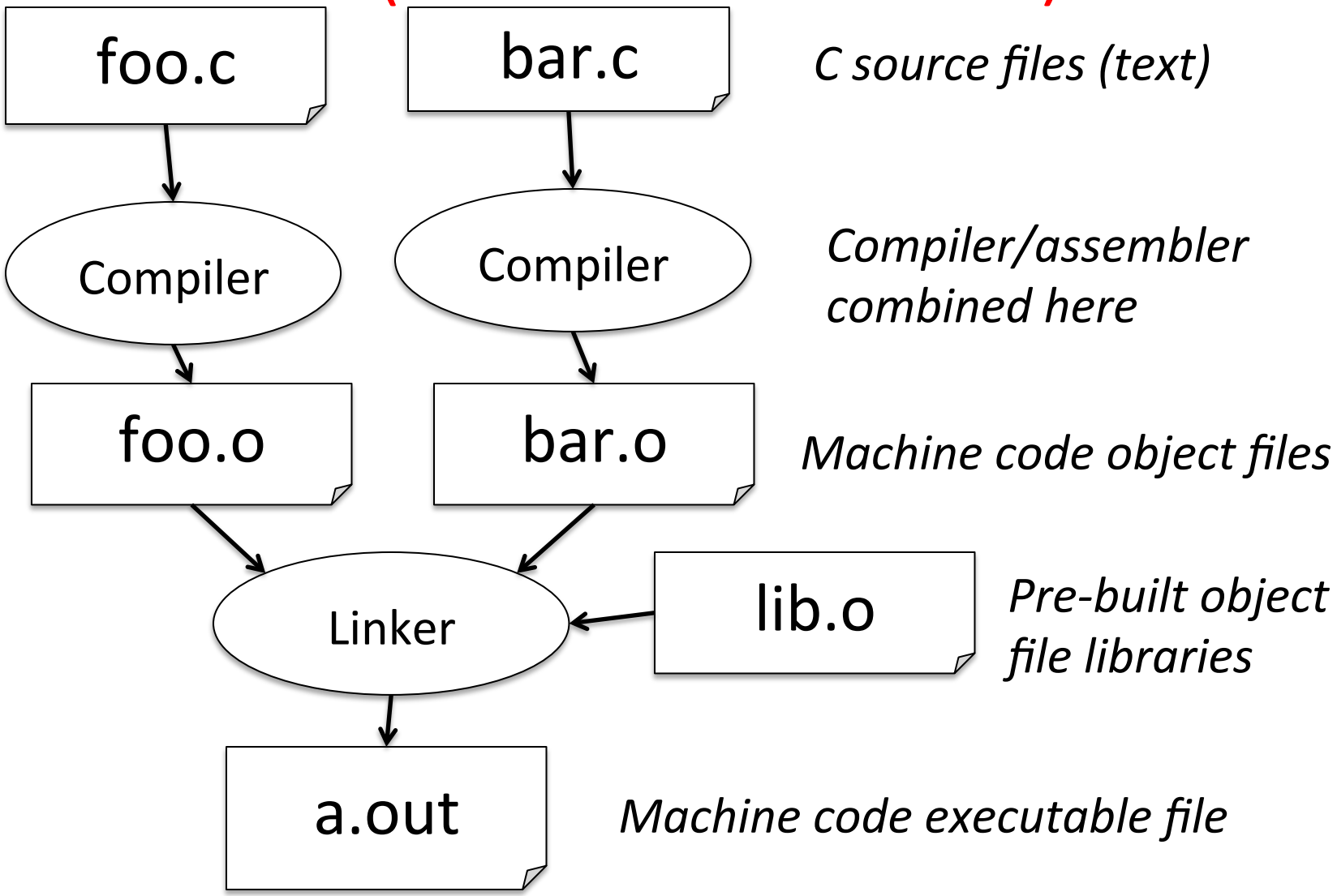
Disclaimer

- You will not learn how to fully code in C in these lectures! You'll still need your C reference for this course
  - K&R is a must-have
    - Check online for more sources
  - "JAVA in a Nutshell," O'Reilly
    - Chapter 2, "How Java Differs from C"
    - http://oreilly.com/catalog/javanut/excerpt/index.html
  - Brian Harvey's helpful transition notes
    - On CS61C class website: pages 3-19
    - http://inst.eecs.berkeley.edu/~cs61c/resources/HarveyNotesC1-3.pdf
- Key C concepts: Pointers, Arrays, Implications for Memory management

Compilation: Overview

- C *compilers* map C programs into architecture-specific machine code (string of 1s and 0s)
  - Unlike *Java*, which converts to architecture-independent *bytecode*
  - Unlike *Python* environments, which *interpret* the code
  - These differ mainly in exactly when your program is converted to low-level machine instructions ("levels of interpretation")
  - For C, generally a two part process of compiling .c files to .o files, then linking the .o files into executables;
  - Assembling is also done (but is hidden, i.e., done automatically, by default); we'll talk about that later

C Compilation Simplified Overview (more later in course)



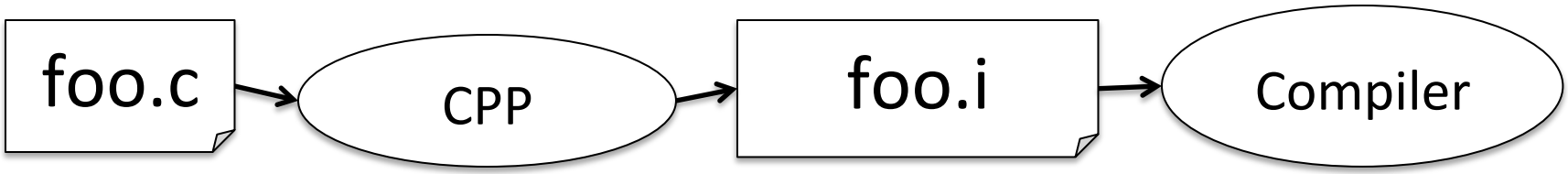
Compilation: Advantages

- Excellent run-time performance: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)
- Fair compilation time: enhancements in compilation procedure (Makefiles) allow only modified files to be recompiled
- Why C?: *we can write programs that allow us to exploit underlying features of the architecture – memory management, special instructions, parallelism*

Compilation: Disadvantages

- Compiled files, including the executable, are architecture-specific, depending on processor type (e.g., MIPS vs. RISC-V) and the operating system (e.g., Windows vs. Linux)
- Executable must be rebuilt on each new system
  - I.e., "porting your code" to a new architecture
- "Change → Compile → Run [repeat]" iteration cycle can be slow during development
  - but Make tool only rebuilds changed pieces, and can do compiles in parallel (linker is sequential though -> Amdahl's Law)

C Pre-Processor (CPP)



- C source files first pass through macro processor, CPP, before compiler sees code
- CPP replaces comments with a single space
- CPP commands begin with "#"
- #include "file.h" /* Inserts file.h into output */
- #include <stdio.h> /* Looks for file in standard location */
- #define M_PI (3.14159) /* Define constant */
- #if/#endif /* Conditional inclusion of text */
- Use –save-temps option to gcc to see result of preprocessing
- Full documentation at: **`http://gcc.gnu.org/onlinedocs/cpp/`**

Agenda

- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

C vs. Java

	C	Java
Type of Language	Function Oriented	Object Oriented
Program-ming Unit	Function	Class = Abstract Data Type
Compilation	gcc hello.c creates machine language code	javac Hello.java creates Java virtual machine language bytecode
Execution	a.out loads and executes program	java Hello interprets bytecodes

hello, world	`#include<stdio.h>`
	`int main(void) {`
	`  printf("Hello\n");`
	`  return 0;`
	`}`
		`public class HelloWorld {`
		`  public static void`
		`main(String[] args) {`
		`  System.out.println("Hello");`
		`  }`
		`}`

Storage	Manual (**`malloc`**, **`free`**)	Automatic (garbage collection)

From http://www.cs.princeton.edu/introcs/faq/c2java.html

C vs. Java

	C	Java
Comments	`/* … */`	`/* … */` or `// …` end of line
Constants	`#define, const`	`final`
Preprocessor	Yes	No
Variable declaration	At beginning of a block	Before you use it
Variable naming conventions	`sum_of_squares`	`sumOfSquares`
Accessing a library	`#include <stdio.h>`	`import java.io.File;`

From http://www.cs.princeton.edu/introcs/faq/c2java.html

Typed Variables in C

```
int   variable1   = 2;
float variable2   = 1.618;
char  variable3   = 'A';
```

- Must declare the type of data a variable will hold
  - Types can't change

Type	Description	Examples
int	integer numbers, including negatives	0, 78, -1400
unsigned int	integer numbers (no negatives)	0, 46, 900
float	floating point decimal numbers	0.0, 1.618, -1.4
char	single text character or symbol	'a', 'D', '?'
double	greater precision/big FP number	10E100
long	larger signed integer	6,000,000,000

Integers: Python vs. Java vs. C

Language	sizeof(int)
Python	>=32 bits (plain ints), infinite (long ints)
Java	32 bits
C	Depends on computer; 16 or 32 or 64

- C: `int` should be integer type that target processor works with most efficiently
- Only guarantee: sizeof(`long long`) ≥ sizeof(`long`) ≥ sizeof(`int`) ≥ sizeof(`short`)
  - Also, `short` >= 16 bits, `long` >= 32 bits
  - All could be 64 bits

Consts and Enums in C

- Constant is assigned a typed value once in the declaration; value can't change during entire execution of program

```
const float golden_ratio = 1.618;
const int days_in_week = 7;
```

- You can have a constant version of any of the standard C variable types
- Enums: a group of related integer constants used to parameterize libraries:

```
enum cardsuit {CLUBS,DIAMONDS,HEARTS,SPADES};
```

Clicker Test

- Clicker participation starting on Monday
- No web-based clickers or phone apps
- Participation only is recorded, not correctness of answers
- Register on bCourses

Clicker Test

A: I have an iClicker

B: I don’t have an iClicker

C: I don’t have an iClicker

D: I don’t have an iClicker

E: I don’t have an iClicker

Compare "`#define PI 3.14`" and "`const float pi=3.14`" – which is true?

A: Constants "PI" and "pi" have same type

B: Can assign to "PI" but not "pi"

C: Code runs at same speed using "PI" or "pi"

D: "pi" takes more memory space than "PI"

E: Both behave the same in all situations

Agenda

- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

Administrivia

- HW0 out, everyone should have been added to edX yesterday
  - Due: Sunday @ 11:59:59pm
- HW0-mini-bio posted on course website
  - Give paper copy to your TA in lab next Tuesday
- Labs start today
  - Meet people in your lab, think about proj1 teams
- Get Clickers!
- Let us know about exam conflicts by the end of this week

Agenda

- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

Typed Functions in C

```
int number_of_people ()
{
  return 3;
}

float dollars_and_cents ()
{
  return 10.33;
}

char first_letter ()
{
  return 'A';
}
```

- You have to *declare* the type of data you plan to return from a function
- Return type can be any C variable type, and is placed to the left of the function name
- You can also specify the return type as **`void`**
  - Just think of this as saying that no value will be returned
- Also necessary to declare types for values passed into a function
- Variables and functions MUST be declared before they are used

Structs in C

- Structs are structured groups of variables, e.g.,

```
typedef struct {
  int length_in_seconds;
  int year_recorded;
} Song;

Song song1;

song1.length_in_seconds =  213;
song1.year_recorded      = 1994;

Song song2;

song2.length_in_seconds =  248;
song2.year_recorded      = 1988;
```

Dot notation: **`x.y = value`**

A First C Program: Hello World

Original C:

```
main()
{
  printf("\nHello World\n");
}
```

ANSI Standard C:

```
#include <stdio.h>

int main(void)
{
  printf("\nHello World\n");
  return 0;
}
```

C Syntax: `main`

- When C program starts
  - C executable a.out is loaded into memory by operating system (OS)
  - OS sets up stack, then calls into C runtime library,
  - Runtime 1st initializes memory and other libraries,
  - then calls your procedure named main ()
- We'll see how to retrieve command-line arguments in main() later…

A Second C Program: Compute Table of Sines

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int    angle_degree;
    double angle_radian, pi, value;
    /* Print a header */
    printf("\nCompute a table of the
   sine function\n\n");

    /* obtain pi once for all       */
    /* or just use pi = M_PI, where */
    /* M_PI is defined in math.h    */
    pi = 4.0*atan(1.0);
    printf("Value of PI = %f \n\n",
   pi);
    printf("angle     Sine \n");

    angle_degree = 0;
    /* initial angle value */
    /* scan over angle     */
    while (angle_degree <= 360)
    /* loop until angle_degree > 360 */
       {
          angle_radian = pi*angle_degree/180.0;
          value = sin(angle_radian);
          printf (" %3d      %f \n ",
                   angle_degree, value);
          angle_degree = angle_degree + 10;
          /* increment the loop index */
       }
    return 0;
}
```

Second C Program Sample Output

```
Compute a table of the sine
   function

Value of PI = 3.141593

angle       Sine
   0      0.000000
  10      0.173648
  20      0.342020
  30      0.500000
  40      0.642788
  50      0.766044
  60      0.866025
  70      0.939693
  80      0.984808
  90      1.000000
 100      0.984808
 110      0.939693
 120      0.866025
 130      0.766044
 140      0.642788
 150      0.500000
 160      0.342020
 170      0.173648
 180      0.000000
 190     -0.173648
 200     -0.342020
 210     -0.500000
 220     -0.642788
 230     -0.766044
 240     -0.866025
 250     -0.939693
 260     -0.984808
 270     -1.000000
 280     -0.984808
 290     -0.939693
 300     -0.866025
 310     -0.766044
 320     -0.642788
 330     -0.500000
 340     -0.342020
 350     -0.173648
 360     -0.000000
```

C Syntax: Variable Declarations

- Similar to Java, but with a few minor but important differences
- *All* variable declarations must appear before they are used (e.g., at the beginning of the block)
- A variable may be initialized in its declaration; if not, it holds garbage!
- Examples of declarations:
  - Correct:
  - Incorrect:

```
{
        int a = 0, b = 10;
        ...
        for (int i = 0; i < 10; i++)
}
```

Newer C standards are more flexible about this, more later

C Syntax : Control Flow (1/2)

- Within a function, remarkably close to Java constructs (shows Java's legacy) in terms of control flow
  - **if-else**
    - **if (expression) statement**
    - **if (expression) statement1**
      else statement2
  - **while**
    - **while (expression)**
      statement
    - **do**
      statement
      while (expression);

C Syntax : Control Flow (2/2)

- **for**
  - **for (initialize; check; update) statement**
- **switch**
  - **switch (expression){**

```
case const1:    statements
case const2:    statements
default:        statements
}
```

  - **break**

C Syntax: True or False

- What evaluates to FALSE in C?
  - 0 (integer)
  - NULL (a special kind of *pointer*: more on this later)
  - *No explicit Boolean type*
- What evaluates to TRUE in C?
  - Anything that isn't false is true
  - Same idea as in Python: only 0s or empty sequences are false, anything else is true!

C and Java operators nearly identical

- arithmetic: +, -, *, /, %
- assignment: =
- augmented assignment: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
- bitwise logic: ~, &, |, ^
- bitwise shifts: <<, >>
- boolean logic: !, &&, ||
- equality testing: ==, !=
- subexpression grouping: ( )
- order relations: <, <=, >, >=
- increment and decrement: ++ and --
- member selection: ., ->
- conditional evaluation: ? :

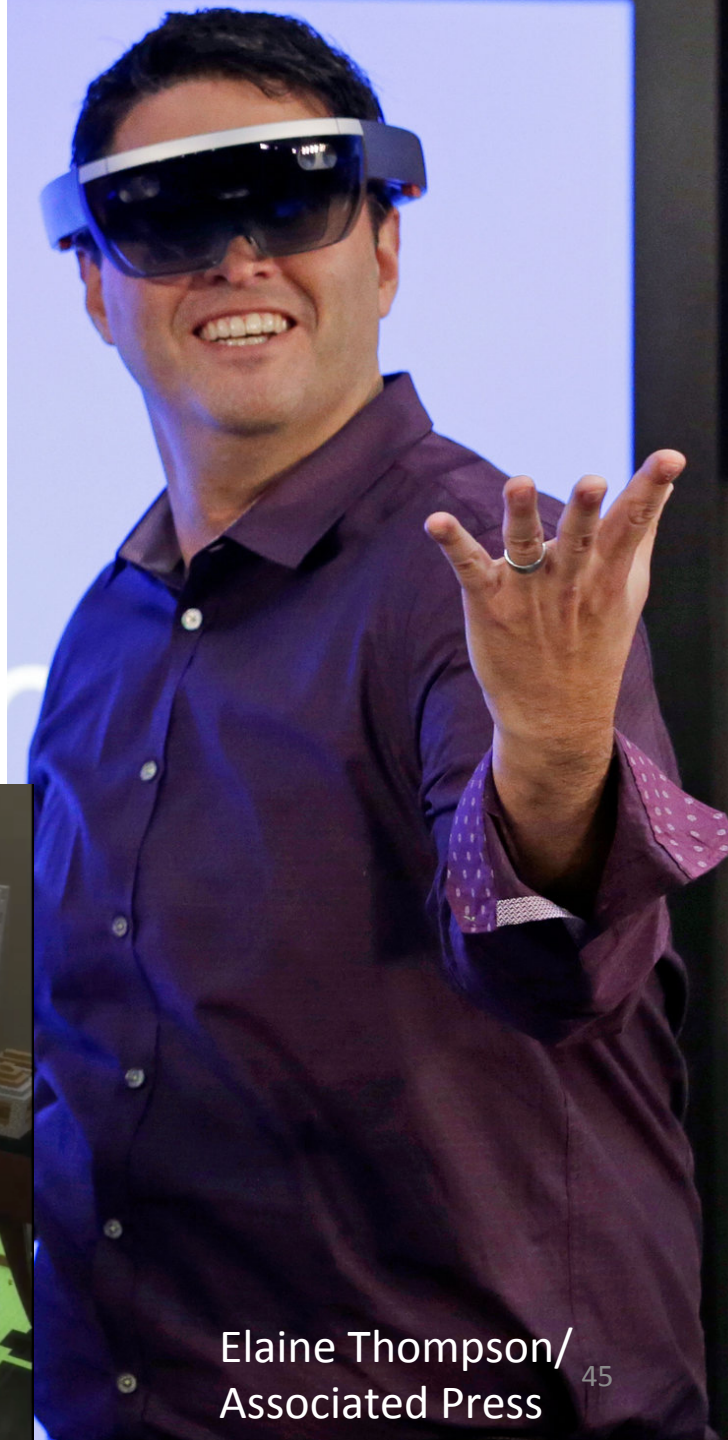
Agenda

- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

In the News

- Microsoft HoloLens
- Announced with Windows 10
- Virtual Reality headset
- Contains CPU and GPU, plus a holographic processing unit, or HPU!

Microsoft



Elaine Thompson/
Associated Press

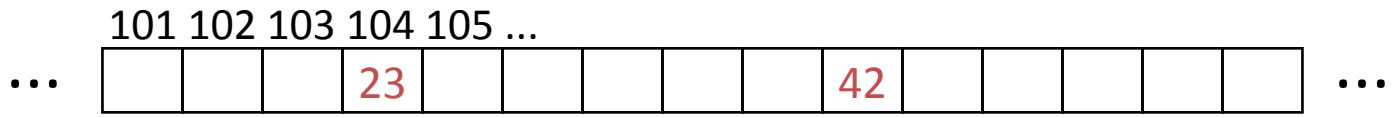
Break

Agenda

- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

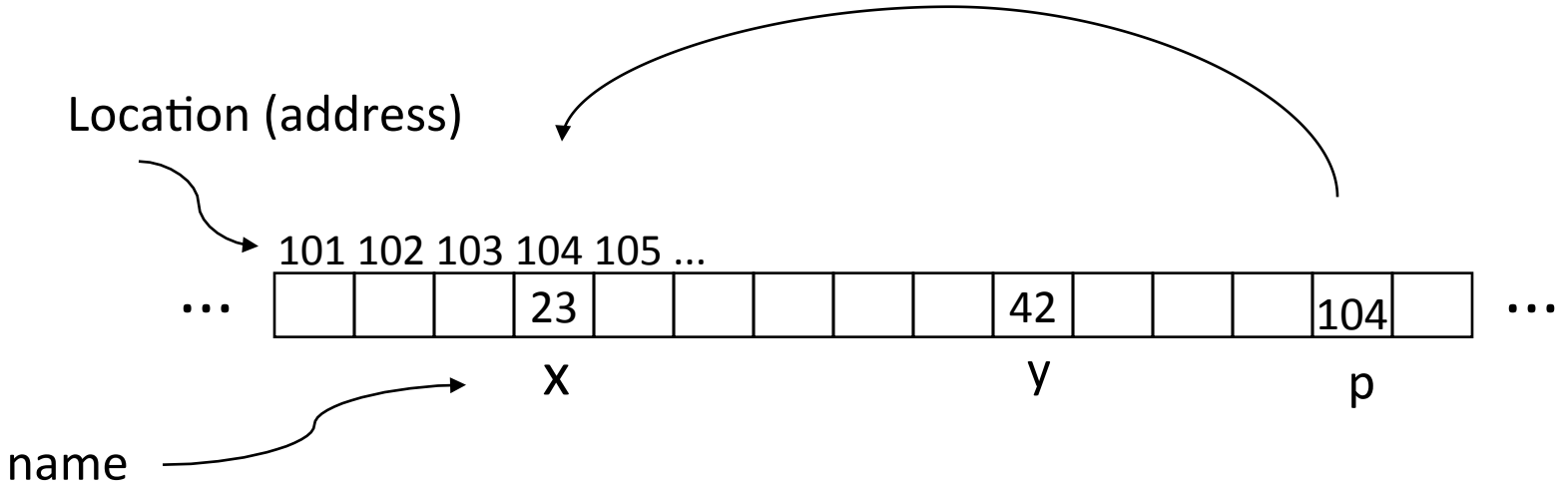
Address vs. Value

- Consider memory to be a single huge array
  - Each cell of the array has an address associated with it
  - Each cell also stores some value
  - Do you think they use signed or unsigned numbers? Negative address?!
- Don't confuse the address referring to a memory location with the value stored there



Pointers

- An *address* refers to a particular memory location; e.g., it points to a memory location
- *Pointer*: A variable that contains the address of a variable



Pointer Syntax

- `int *x;`
  - Tells compiler that variable x is address of an `int`
- `x = &y;`
  - Tells compiler to assign address of `y` to `x`
  - `&` called the "address operator" in this context
- `z = *x;`
  - Tells compiler to assign value at address in `x` to `z`
  - `*` called the "dereference operator" in this context

Creating and Using Pointers

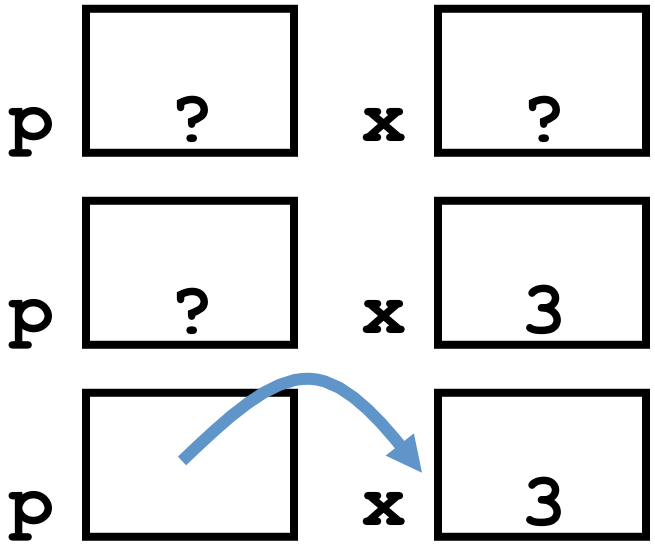
- How to create a pointer:

  & operator: get address of a variable

```
int *p, x;

   x = 3;

   p = &x;
```



Note the "*" gets used 2 different ways in this example. In the declaration to indicate that **p** is going to be a pointer, and in the **printf** to get the value pointed to by **p**.

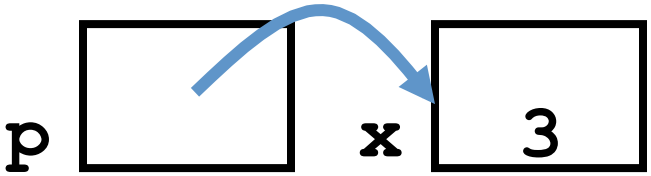
- How get a value pointed to?

  "*" (dereference operator): get the value that the pointer points to

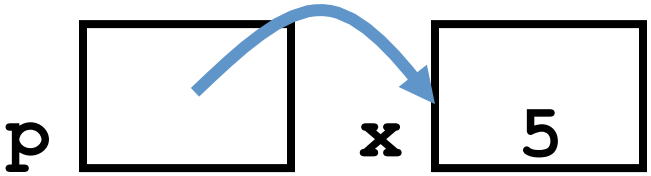
```
printf("p points to %d\n",*p);
```

Using Pointer for Writes

- How to change a variable pointed to?
  - Use the dereference operator ***** on left of assignment operator **=**




```
*p = 5;
```



Pointers and Parameter Passing

- Java and C pass parameters “by value”
  - Procedure/function/method gets a copy of the parameter, *so changing the copy cannot change the original*

```
void add_one (int x) {
    x = x + 1;
  }
int y = 3;
add_one(y);
```

y remains equal to 3

Pointers and Parameter Passing

- How can we get a function to change the value held in a variable?

```
void add_one (int *p) {
   *p = *p + 1;
  }
int y = 3;

add_one(&y);
```

y is now equal to 4

Types of Pointers

- Pointers are used to point to any kind of data (**`int`**, **`char`**, a **`struct`**, etc.)
- Normally a pointer only points to one type (**`int`**, **`char`**, a **`struct`**, etc.).
  - **`void *`** is a type that can point to anything (generic pointer)
  - Use **`void *`** sparingly to help avoid program bugs, and security issues, and other bad things!

More C Pointer Dangers

- Declaring a pointer just allocates space to hold the pointer – it does not allocate the thing being pointed to!
- Local variables in C are not initialized, they may contain anything (aka “garbage”)
- What does the following code do?

```
void f()
{
    int *ptr;
    *ptr = 5;
}
```

Pointers and Structures

```
tyepdef struct {
    int x;
    int y;
} Point;

Point p1;
Point p2;
Point *paddr;
```

```
/* dot notation */
int h = p1.x;
p2.y = p1.y;

/* arrow notation */
int h = paddr->x;
int h = (*paddr).x;

/* This works too */
p1 = p2;
```

Pointers in C

- Why use pointers?
  - If we want to pass a large struct or array, it's easier / faster / etc. to pass a pointer than the whole thing
  - In general, pointers allow cleaner, more compact code
- So what are the drawbacks?
  - Pointers are probably the single largest source of bugs in C, so be careful anytime you deal with them
    - Most problematic with dynamic memory management—coming up next week
    - *Dangling references* and *memory leaks*

Why Pointers in C?

- At time C was invented (early 1970s), compilers often didn't produce efficient code
  - Computers 25,000 times faster today, compilers better
- C designed to let programmer say what they want code to do without compiler getting in way
  - Even give compilers hints which registers to use!
- Today's compilers produce much better code, so may not need to use pointers in application code
- Low-level system code still needs low-level access via pointers

Video: Fun with Pointers

- https://www.youtube.com/watch?v=6pmWojisM_E

Agenda

- Compile vs. Interpret
- C vs. Java vs. Python
- Administrivia
- Quick Start Introduction to C
- News/Technology Break
- Pointers
- And in Conclusion, …

And In Conclusion, …

- All data is in memory
  - Each memory location has an address to use to refer to it and a value stored in it
- Pointer is a C version (abstraction) of a data address
  - `*` "follows" a pointer to its value
  - `&` gets the address of a value
  - Arrays and strings are implemented as variations on pointers
- C is an efficient language, but leaves safety to the programmer
  - Variables not automatically initialized
  - Use pointers with care: they are a common source of bugs in programs