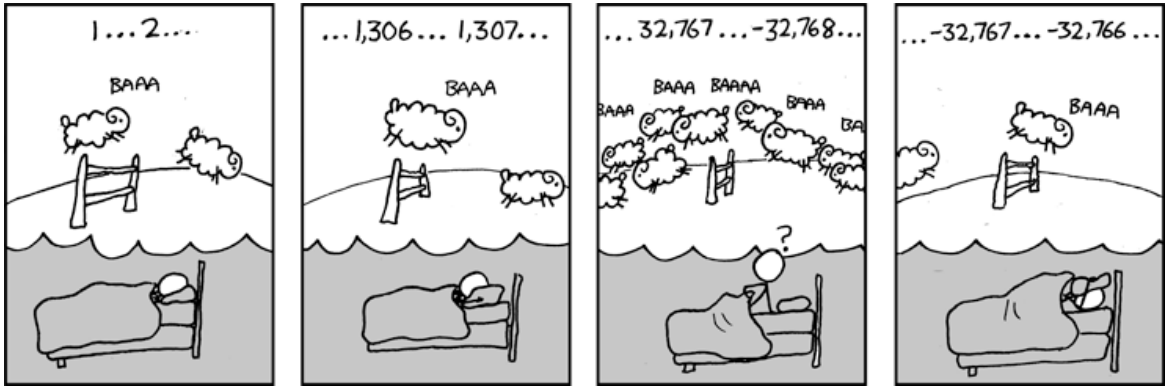
Lecture 0: Introduction to Computer Systems

CS 105

Abstraction

Correctness

- **Example 1: Is $x^2 \geq 0$?**
  - Floats: Yes!



  - Ints:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??
- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Ints: Yes!
  - Floats:
    - (2^30 + -2^30) + 3.14 → 3.14
    - 2^30 + (-2^30 + 3.14) → ??

Performance

```
void copyij(int src[2048][2048],
            int dst[2048][2048]){
  int i,j;
  for (i = 0; i < 2048; i++){
    for (j = 0; j < 2048; j++){
      dst[i][j] = src[i][j];
    }
  }
}
```

4.3ms

```
void copyji(int src[2048][2048],
            int dst[2048][2048]){
  int i,j;
  for (j = 0; j < 2048; j++){
    for (i = 0; i < 2048; i++){
      dst[i][j] = src[i][j];
    }
  }
}
```

81.8ms

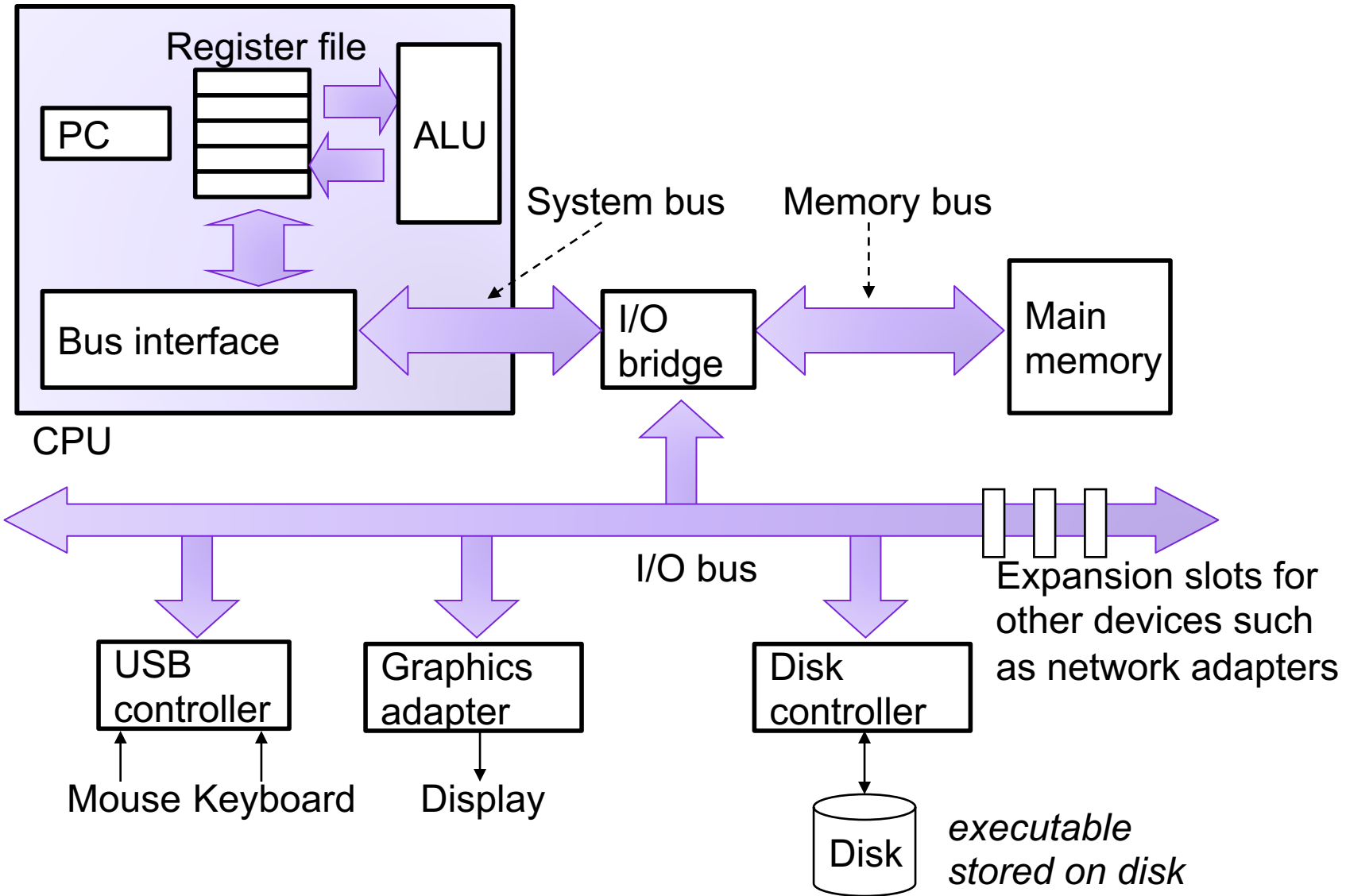
- Hierarchical memory organization
- Performance depends on access patterns
  - Including how step through multi-dimensional array

Security

```
void admin_stuff(int authenticated){
  if(authenticated){
    // do admin stuff
    // should only happen if user is authenticated
    printf("The answer is 42\n");
  }
}

int dontTryThisAtHome(char * user_input, int size) {
  char data[size];
  int ret = memcpy(*user_input, data);
  return ret;
}
```

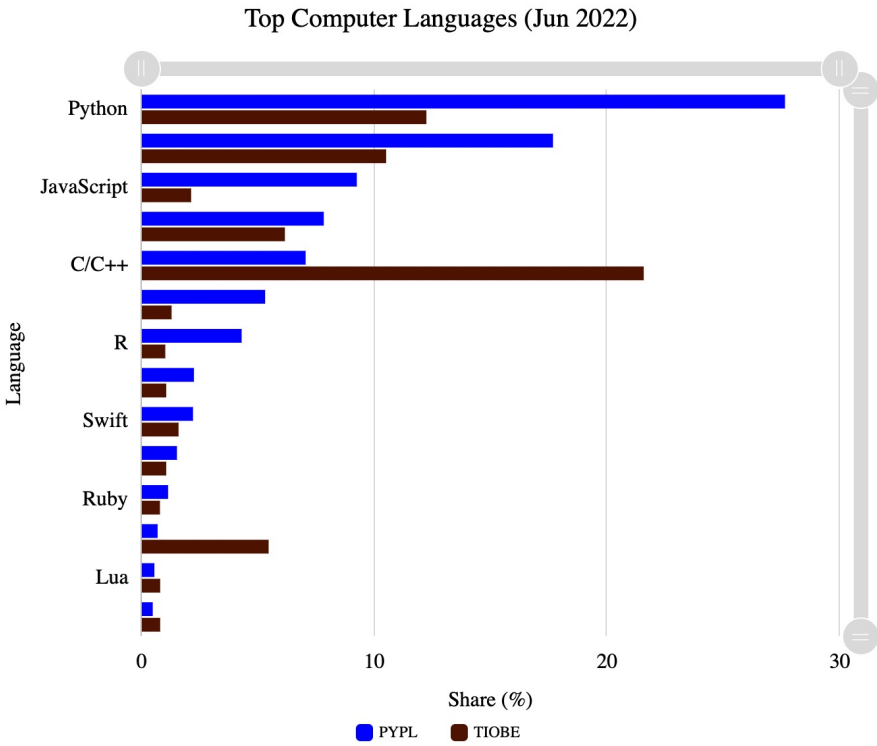
A Computer System



C

- imperative language that provides low-level access to memory
- low overhead, high performance

- developed at Bell labs in the 1970s
- C (and related languages) still today



Variables

- Declaration

```
int myVariable;
```

type name semi-colon

- Assignment

```
myVariable = 47;
```

name value semi-colon

- Declaration and assignment

```
int myVariable = 47;
```

C Data Type	x86-64
char	1
unsigned short	2
unsigned int	4
unsigned long	8
short	2
int	4
long	8
float	4
double	8

Operations

- Arithmetic Operations: `+, -, *, /, %`

```
int x = 47;
int y = x + 13;
y = (x * y) % 5;
```

- Boolean Operators: `==, !=, >, >=, >, >=`

```
int x = (13 == 47);
```

- Bitwise Operations: `&, |, ^, ~`

```
int x = 47;
int y = ~x;
y = x & y;
```

- Logical Operations: `&&, ||, !`

```
int x = 47;
int y = !x;
y = x && y;
```

Functions

Declaring a Function

```
int myFunction(int x, int y){

  int z = x - 2*y;
  return z * x;

}
```

Calling a Function

```
int a;

a = myFunction(47, 13);
```

Exercise

- Define a function add3 that takes three integers as arguments and returns the sum of those three values

Control Flow

Conditionals

```
int x = 13;
int y;
if (x == 47){
  y = 1;
} else {
  y = 0;
}
```

While Loops

```
int x = 47;

while (x > 0){
  x = x - 1;
}
```

Do-While Loops

```
int x = 47;
do {
  x = x - 1;
} while (x > 0);
```

For Loops

```
int x = 0;
for (int i=0; i < 47; i++){
  x = x + i;
}
```

Exercise

- Define a function that takes two integers and returns an integer. If the second integer argument is greater than (or equal to) the first, it returns the sum of the integer values between those two numbers (inclusive). Otherwise it returns -1.

Main Functions

- By convention, main functions in C take two arguments:
  1. `int argc`
  2. `char ** argv`
- By convention, main functions in C return an int
  - 0 if program exited successfully

```
int main(int argc, char ** argv){
  // do stuff

  return 0;
}
```

Aside: Printing

```
printf("Hello world!\n");

printf("%d is a number\n", 13);

printf("%d is a number greater than %f\n”, 47, 3.14);
```

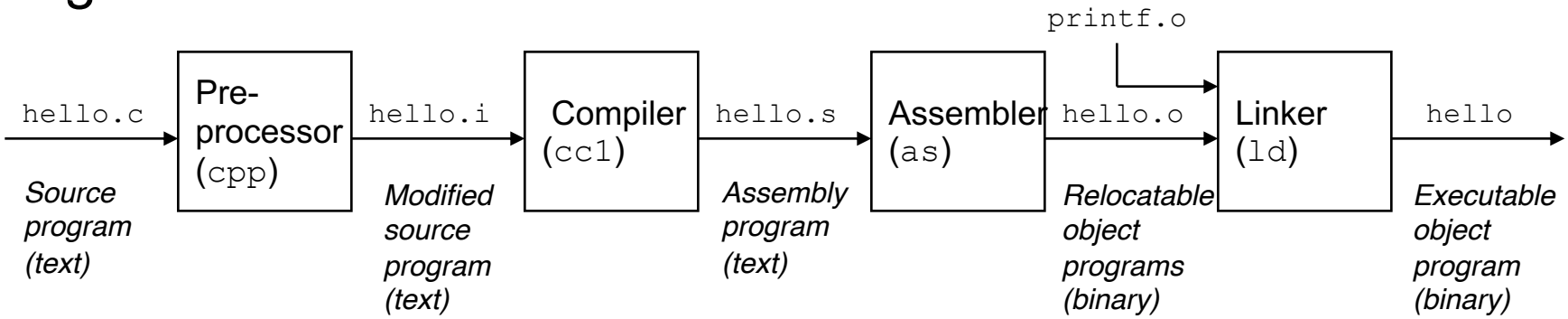
Exercise

- Define a main function that computes the sum of the integers between 13 and 47 and prints that value.

Compilation

compiler output name filename

- gcc –o hello hello.c




```
#include<stdio.h>

int main(int argc,
         char ** argv){

  printf("Hello
          world!\n");
  return 0;
}
```

```
…
int printf(const char *
           restrict,
           ...)
__attribute__((__format_
_ (__printf__, 1, 2)));
…
int main(int argc,
         char ** argv){

  printf("Hello
          world!\n");
  return 0;
}
```

```
pushq   %rbp
movq    %rsp, %rbp
subq  $32, %rsp
leaq  L_.str(%rip), %rax
movl  $0, -4(%rbp)
movl  %edi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rax, %rdi
movb  $0, %al
callq _printf
xorl  %ecx, %ecx
movl  %eax, -20(%rbp)
movl  %ecx, %eax
addq  $32, %rsp
popq  %rbp
retq
```

```
55
48 89 e5
48 83 ec 20
48 8d 05 25 00 00 00
c7 45 fc 00 00 00 00
89 7d f8
48 89 75 f0
48 89 c7
b0 00
e8 00 00 00 00
31 c9
89 45 ec
89 c8
48 83 c4 20
5d
c3
```

Running a Program

- ./hello

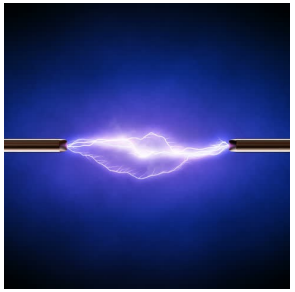
Bits

- a **bit** is a binary digit that can have two possible values

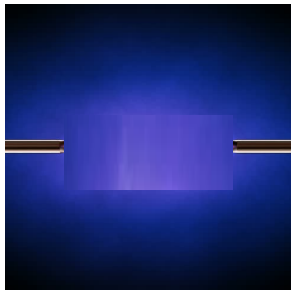
- can be physically represented with a two state device





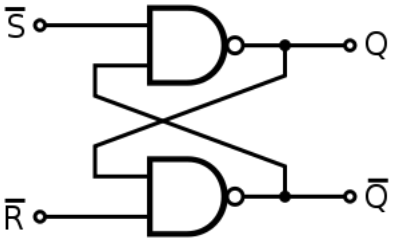




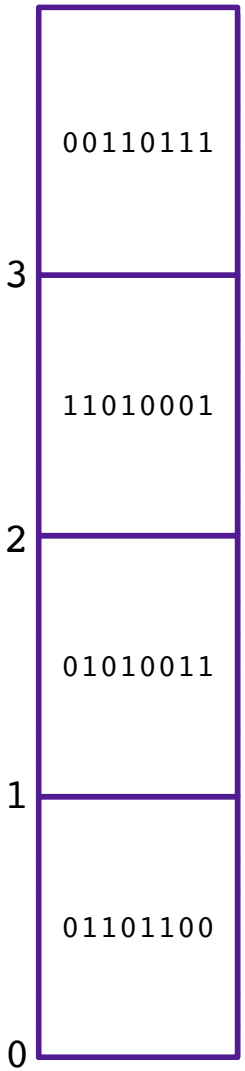
Storing bits

- Static random access memory (SRAM): stores each bit of data in a flip-flop, a circuit with two stable states
- Dynamic Memory (DRAM): stores each bit of data in a capacitor, which stores energy in an electric field (or not)
- Magnetic Disk: regions of the platter are magnetized with either N-S polarity or S-N polarity
- Optical Disk: stores bits as tiny indentations (pits) or not (lands) that reflect light differently
- Flash Disk: electrons are stored in one of two gates separated by oxide layers



Bytes and Memory

- **Memory** is an array of ~~bits~~ bytes
- A **byte** is a unit of eight bits
- An index into the array is an **address**, **location**, or **pointer**
  - Often expressed in hexadecimal
- We speak of the *value* in memory at an address
  - The value may be a single byte …
  - … or a multi-byte quantity starting at that address



Arrays

- Contiguous block of memory
- Random access by index
  - Indices start at zero

- Declaring an array:

```
int array1[5]; // array of 10 ints named array1

char array2[47]; // array of 47 chars named array2

int array3[7][4]; // two dimensional array
```

- Accessing an array:

```
int x = array1[0];
```

- The array variable stores the address of the first element in the array

Strings

- Strings are just arrays of characters
- End of string is denoted by null byte `\0`

Pointers

- Pointers are addresses in memory (i.e., indexes into the array of bytes)
- Most pointers declare how to interpret the value at (or starting at) that address
- Examples:

```
int * ptr = &myVariable;
char * ptr2 = (char *) ptr;
```

- Dereferencing pointers:

```
int var2 = *ptr
char c = *ptr2;
```

Pointer Types	x86-64
void *	8
int *	8
char *	8
⋮	8

& and * are inverses of one another

Pointer Arithmetic

```
int * ptr = &myVariable;
ptr += 1;

char * ptr2 = (char *) ptr;
ptr2 += 1;
```

- Location of **`ptr+k`** depends on the type of **`ptr`**
- adding 1 to a pointer **`p`** adds **`1*sizeof(*p)`** to the address

- **`array[k]`** is the same as **`*(array+k)`**

Strings

- Strings are just arrays of characters
- End of string is denoted by null byte `\0`

- generally declared as type `char *`

Exercise

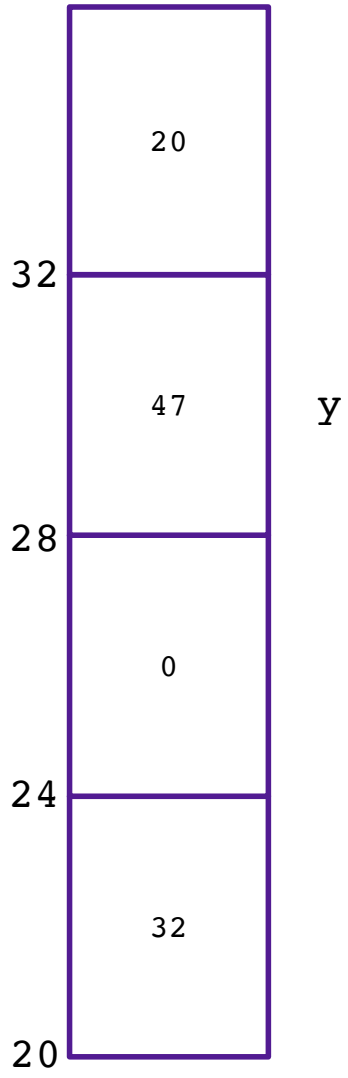
What does x evaluate to in each of the following?

1.
```
int * ptr = 32;
x = *ptr
```

2.
```
int y = 47; // assume at 28
x = &y
```

3.
```
int * ptr = 20;
x = *(*ptr)
```

4.
```
int * ptr = 24;
x = *(ptr+1)
```



Structs

- Heterogeneous records, like objects
- Typical linked list declaration:

```
typedef struct cell {
  int value;
  struct cell *next;
} cell_t;
```

- Usage:

```
cell_t c;
c.value = 42;
c.next = NULL;
```

- Usage with pointers:

```
cell_t *p;
p->value = 42;
p->next = NULL;
```

`p->next` is an abbreviation for `(*p).next`

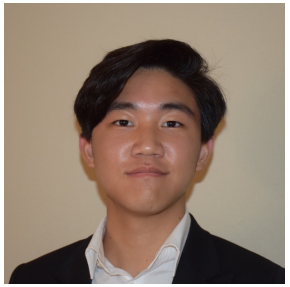
LOGISTICS

Course staff

Prof. Eleanor Birrell
Edmunds 221

Research in security and privacy
OH: M 7-9pm, T 2-4pm

Claire LeBlanc



Josh Yum

Pei Qin

Tonya Chivandire

Ziang Xue

The Course in a Nutshell

- Textbook
  - Bryant and O'Halloran, *Computer Systems: A Programmer's Perspective*, **third edition,** Pearson, 2016 (Recommended)
- Classes
  - Monday and Wednesday, 11am – 12:15pm in Edmunds 101
- Labs
  - Wednesdays 7-8:15 in Edmunds 229/219
  - **Starts Today!** Be sure to have an account and password

Mentor Session Schedule (Edmunds 227)						
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
4-6pm 7-9pm*	2-4pm* 7-9pm	LAB	7-9pm	1-3pm	2-4pm	3-5pm

Grading

- Assignments
  - Introduced during labs, Due Tuesdays at 11:59pm
  - Tremendous fun, work in pairs
  - must complete them all
  - Thirteen late days
- Check-ins
  - one-question exams at the start of lab next week
  - graded "Got it" / "Not yet"
  - Can improve from "Not yet" to "Got it" via one-on-one meeting or extra chance checkpoints
  - no limit on number of attempts to improve grade
- Grades
  - Must successfully complete all the assignments
  - Beyond that, grade determined by the number of "Got it" topics

Course website

https://www.cs.pomona.edu/classes/cs105

- All information is on the course website
- All course materials get posted on the course website
- Links from the course page:
  - Course materials (slides, demo code, videos, practice problems)
  - Slack (#cs105-2023sp), for questions and discussion
  - Gradescope, for submitting assignments and seeing grades