CS202 (003): Operating Systems Intro

Instructor: Jocelyn Chen

Most of the materials covered in this slide come from the lecture notes of Mike Walfish's CS202

Course Staff

- Instructor: Jocelyn Chen
- TA: Bob Yao
- Course webpage: See your Brightspace course page
- Course webpage contains syllabus, important information about HW policy, handouts, reading, etc.

About this Course

Learn how operating systems work

Learn key abstractions and concepts in operating systems

These materials will be useful beyond OSes!

Understanding resource management, abstractions, design tradeoffs in large-scale systems

Textbooks

- *Operating Systems: Three Easy Pieces*, by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau.
- *Operating Systems and Middleware: Supporting Controlled Interaction*, by Max Hailperin.
- *Computer Systems: A Programmer's Perspective*, Third Edition, Randal E. Bryant and David R. O'Hallaron.
- *The C programming language (second edition)*, Brian W. Kernighan and Dennis M. Ritchie.

- Textbooks are not substitute for lectures!
- Class presentation may not follow the books
- Skip many chapters and cover extra materials

Campuswire

- We will be using Campuswire for all course-related discussions
- **Make sure you can access Campuswire! (link available on webpage)**
- Please use Campuswire (**with posts, not DMs**) for class-related questions. Please use common sense when posting questions: hints/ideas ok, but **cannot post full solutions**. If you include **code**, please mark your question **private**.
- Please use **DMs** (through Campuswire) for more personal questions. We will **ignore** emails about course administration and other class-related questions.
- We will response in 12-24 hours. Don’t expect us to answer questions minutes before the due time.

Office Hours

Name	Time	Location
Bob	Tue 5-6 PM	60FA Room 527
Jocelyn	Wed 4-5 PM	60FA Room 446
Bob	Fri 3:30-4:30 PM	60FA Room 446

Periodically check course website and Campuswire for OH update!

Course Grades

- Exam: collectively 60% of final grade
- Homework: 5% of final grade
- Labs: 35% of final grade
- Final grades **will be curved**
- You can submit any graded item for a regrade, under the following conditions:
  - You submit a clear, written statement that explains the request
  - You submit your request within one week of when the graded work was returned
  - We will regrade the entire exam, homework, etc. (so a regrade might decrease your grade)

Homework Policy

- Homework is intended to reinforce the course material
- Submit homework in a **typed pdf** generated by LaTex or a markup language. Hand-written assignment will receive **0** automatically.
- Homework must be submitted by **5pm** on the due date
- Late homework **will not be accepted**. We will drop your lowest two homework scores
- Homework will be graded loosely. To receive credit, you must take a credible effort to solve the problem. Minor mistakes will not be penalized, in general.

Lab Policy

- Lab must be submitted by **7pm** on the due date
- Late labs will be accepted until **7pm** a **week after the due date**
- Late labs incur penalties. However, you have **5** slack days that forestall the penalty clock. Please view the detailed policy here.
- We will **ignore all extension requests** for reasons such as job interviews, work on research publications, etc. unless it is explicitly stated here
- You will get a **0** if you either do not hand in the assignment, or hand in a blank assignment.

Collaboration Policy

- All assignments (homework, labs) must be **done on your own**. That means,
  - **Not allowed** to do assignments in groups
  - **Not allowed** to check solutions with each other
  - **Not allowed** to discuss problems with each other through other channels that course staff does not have access to. You can discuss questions (in general terms) through Campuswire
  - **Not allowed** to discuss/show/debug code with any person other than the instructor and the TA
  - Collaboration with other students on assignments is considered **cheating**

Collaboration Policy, cont.

- You **may not** use any AI-Assisted code writing tools (such as Copilot) in this course.
- You **may not** use any AI-Assisted tools for written assignments and non-coding lab questions.
- You **may** use AI-Assisted chatbots for Lab 2 and onwards (more on this next slide)
- You **may not** look at, or use, (similar) solutions from prior years on the web, or seek assistant from the Internet.
- You **must** acknowledge your influences (either from any person you discussed with, websites, AI assistants, or any other sources). You **must** declare what ideas are borrowed from the source
- You **must** take reasonable steps to protect your work. You **must not publicize** your solutions in this semester or any future semester

What you **can** do with AI-assisted Chatbots

- **General Programming Language Queries:**
  - “How to declare a struct in C?”
  - “How do I traverse a linked list?”
- **Standard Library Functions:**
  - “What is the string length function in <string.h>?”

These two are the **only usages** allowed!

What you **cannot** do with AI-assisted Chatbots (include but not limited to)

- **Lab-Specific Code Completion**: "[lab instructions] [code snippet] Can you help me complete the code/give me a skeleton of the implementation/pseudocode?"
- **Code Style Improvement**: "Please improve the coding style of the following code using the guideline [coding style rubric]"
- **Lab Question Answers**: "[lab question]"
- **Debugging Assistance**: "[your code] I get the following error [error], what is going on?"
- **Algorithm Explanation**: "[code snippet from the lab] What is this part of the code doing?"

Honor Code

- Failing to adhere to the collaboration and integrity policy is **a violation of the NYU honor code**
- We take the honor code **extremely seriously**! If you cheat, you will own the consequences
- If you are unsure whether a particular source of external information is permitted, **contact the instructor** before looking at it
- Please make sure you read the policy page and understand **everything** in it. **ASK US IF YOU HAVE ANY QUESTIONS**. **Do not** wait until you violate the policy and then say “I thought this sentence means …”.

More on Labs

- Lab is a **crucial component** of the operating system course. You will spend substantial time programming
- **Please start early**! The labs will take more than you think
- We are eager to help you but **please make sure you are making good use of our time** (i.e. please think a while before asking any questions, make sure there is no similar questions on Campuswire)
- Make sure you check out Setup, *The Missing Semester of your CS education*, and *Unix dev tools* before working on the labs

Let’s get started!

What is an operating system?

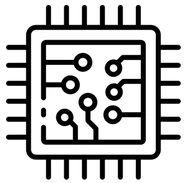
"Operating System is a program that abstracts and manages hardware resources for user programs."

Application

Web Browsers
Games
Office Softwares
…

Operating System

File System
Processes
Virtual Memory
…



Hardware

Disk
CPU
RAM
…

What is an operating system?

Job 1: managing the resources of the machine

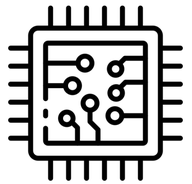
Scheduling: give each process some of the CPU

Virtual Memory: give each process some of the physical memory

Make sure one program won't screw up another (through multiplexing, isolation, protection, sharing)

Operating System

File System
Processes
Virtual Memory
…



Hardware

Disk
CPU
RAM
…

What is an operating system?

Job 2: Abstracting the hardware

(You do not want to program on hardware!)

Hide details of hardware for convenience and portability

Allow programs to run unaware of the hardware details

Application

Web Browsers
Games
Office Softwares
…

File System
Processes
Virtual Memory
…

Examples of resource management and abstraction

File Systems

Abstraction

"File is a continuous array of bytes" is a false illusion

Data in a file can be fragmented across different locations on the disk.

```
fd = open("/tmp/foo", WR_ONLY)
rc = write(fd, "abc...z", 26)
```

*"**tmp/fo**" abstracts actual location on the storage device*

*"**WR_ONLY**" simplifies access control*

*The **write** call abstracts the entire process of writing data to storage.*

Isolation

User program cannot write to a file unless it has permission

Examples of resource management and abstraction

Text Input

Abstraction

"Input from any source (a soft keyboard displayed on a touch screen, a physical keyboard, etc.) act the same" is a false illusion

There are significant differences in how different inputs are processed.

Programs are not aware of your input method

Isolation

Ensure that keystrokes go to a single program

Otherwise, would other application know your passwords?

Examples of resource management and abstraction

Memory

Abstraction

```
movl 0x1248 %rdx
```

“It is reading from memory address 0x1248”
is a false illusion
“0x1248” is a virtual address!

“the computer has a linear contiguous address space”is a false illusion
Physical memory can be fragmented and spread across different locations

Isolation

User program can’t write to another user’s memory

Examples of resource management and abstraction

Scheduling

Abstraction	Isolation
“this process is running continuously” is a false illusion In fact, processes are actually being rapidly started and stopped by the operating system.	User program that is hogging CPU gets switched out in favor of another user’s program

In this course

What is a operating system?

How does OS abstract hardware resources?

Why does OS provide these abstractions?

Let’s go through an example

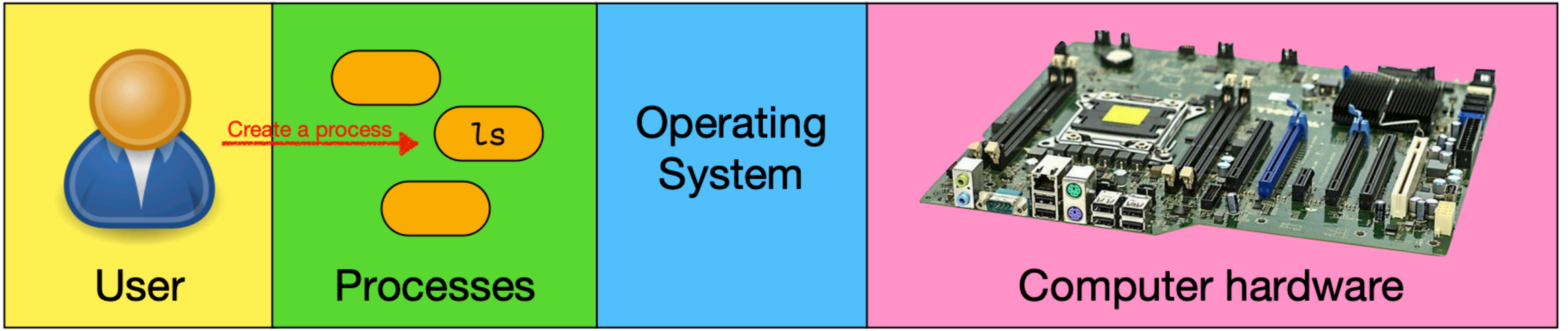
The following 5 slides are borrowed from Yang Tang’s lecture notes for CS202

What is an operating system?

Example

Step 1

Most commands you type in the shell will start a new process.



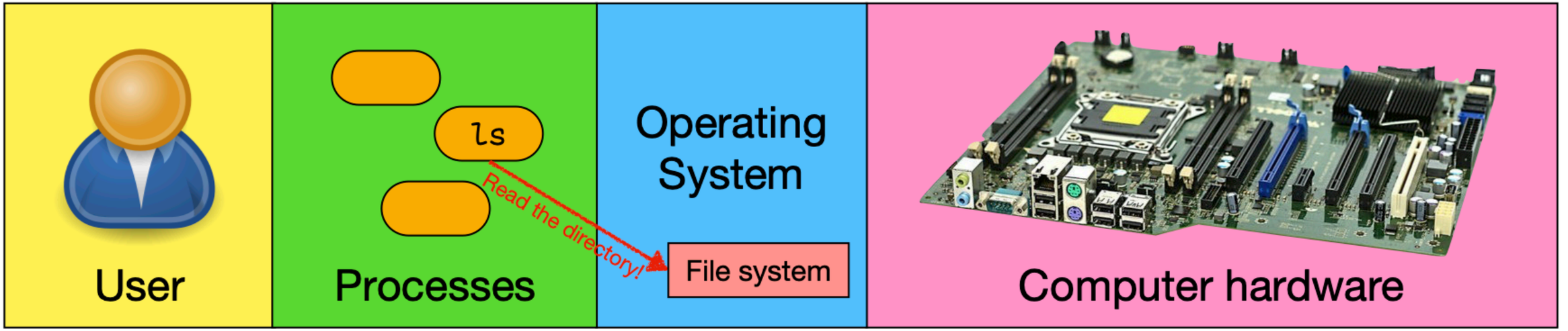
What is an operating system?

Example

Step 2

The operating system contains code that is needed to work with the file system.

Such code in the OS is called the kernel.



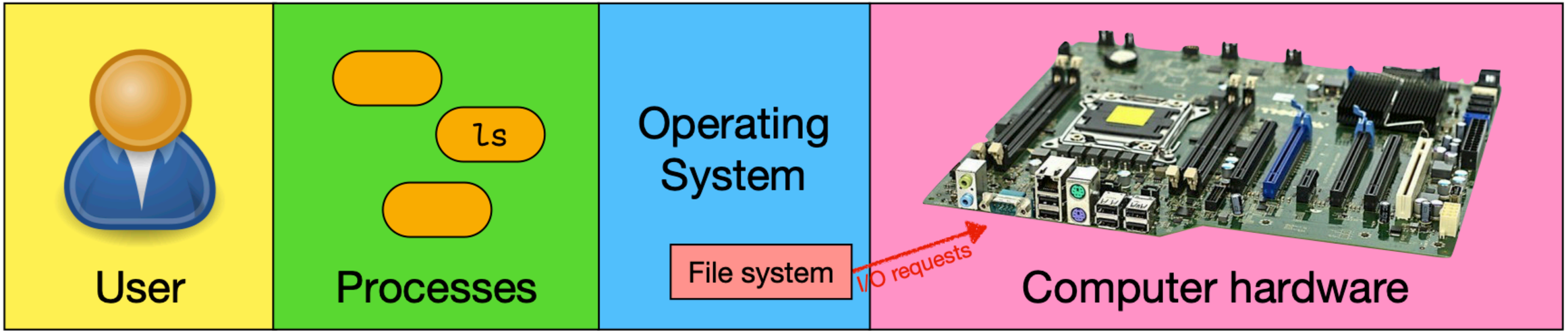
What is an operating system?

Example

```
$ ls
mail  public_html
$ _
```

Step 3

The file system module inside the operating system knows how to work with devices, using device drivers.



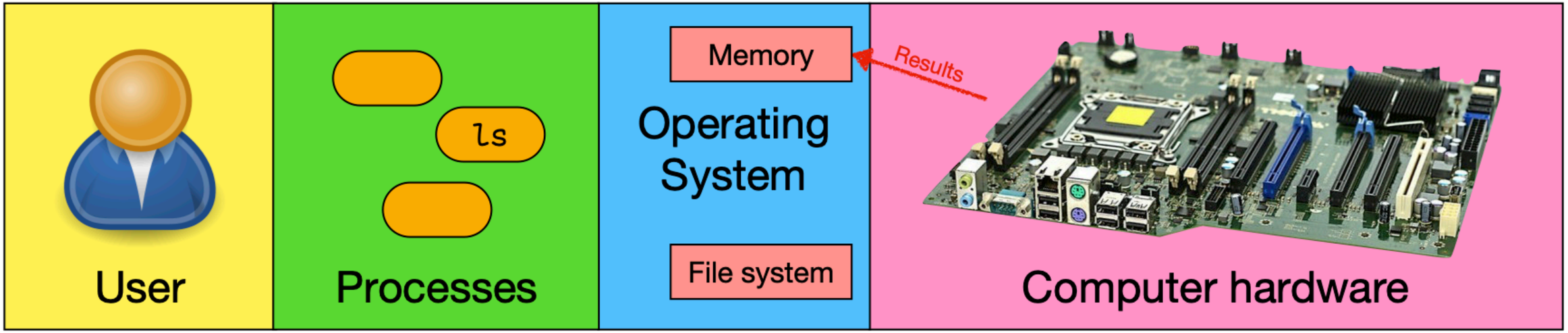
What is an operating system?

Example

```
$ ls
mail  public_html
$ _
```

Step 4

Of course, the operating system will allocate memory for the results.



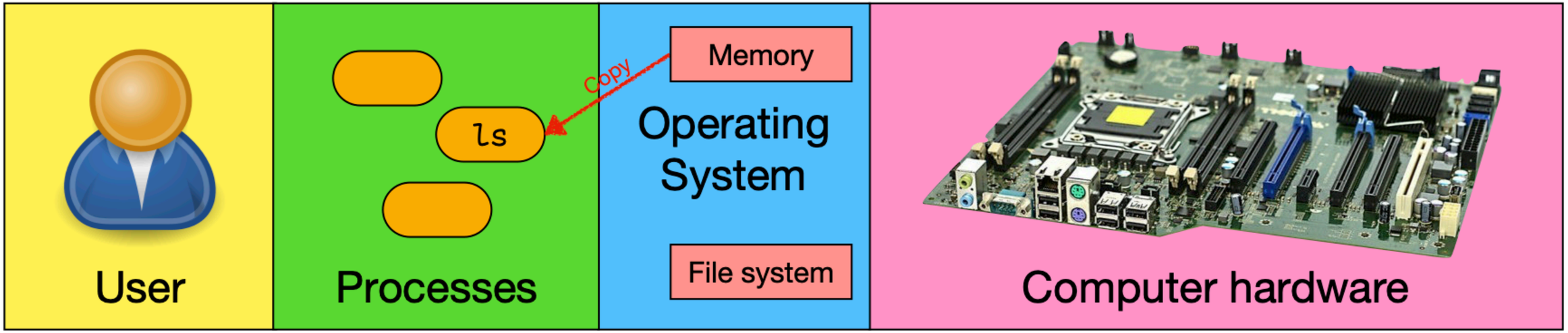
What is an operating system?

Example

```
$ ls
mail  public_html
$ _
```

Step 5

The memory management subsystem will copy the result to the memory of the process.



HW 1 and Lab 1 are released today!

Notes on Lab 1

Lab 1 should be easier than other labs.

The lab will help you review C and teach you to use gdb

If you are unsure about your C skills, I recommend looking at K&R: The C programming language.

Revisiting C: Declaration & Initialization

Declaration
Informing the compiler about the variable name and type.
For stack variables, it reserves space, but doesn't allocate memory dynamically.

```
int x;    // Declaration of an integer (space reserved on stack)
int *ptr;  // Declaration of a pointer (space for the pointer itself, not what it points to)
```

Initialization
Assigning an initial value to the declared variable.

```
int y = 10;     // Declaration and initialization of an integer
int * x = &y;   // Declaration and initialization of a pointer to y
int * z = NULL; // Declaration and initialization to a null pointer
```

Important Note:
Always initialize your variables to avoid undefined behavior!

Revisiting C: Pointers

What are Pointers?
A pointer is an address within memory OR a variable whose value is the address of another variable.

- Essential for tasks like dynamic memory allocation (remember `malloc`?)
- Simplify certain programming tasks

Pointer Syntax
Format: `type * variable_name`

- The '*' denotes the pointer type
- Don't confuse this with accessing/dereferencing pointers
- '*' is also used to access contents (dereferencing)

```
int * myFavNum = 6;  // pointer to int 6
printf("%d\n", *myFavNum);  // '6' - *myFavNum accesses the value stored at myFavNum
```

Is this code correct?

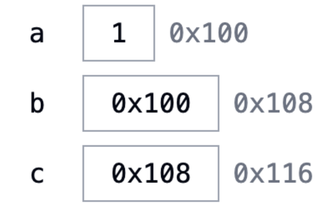
No, because int *myFavNum does not point to 6, it only assigns the point myFavNum with (address) 6

Revisiting C: Pointers

Visualizing pointers as boxes containing memory addresses can help in understanding how they work.

```
int a = 1;
int* b = &a;
int** c = &b;
```

Visual Representation:




```
int* x = *c;
int y = *x;
```

What is the value of y?

x	0x100	x stores the *c, and *c = 0x100
y	1	y stores *(x), which is equivalent to *(0x100), and that corresponds to the value of a

Revisiting C: Array and string

Strings in C
- C has no built-in string type
- Strings are represented as arrays of characters
- Character arrays are terminated with a null byte ('\0')

Accessing Array Elements
Two ways to access elements in an array:

- Subscript notation: array[index]
- Pointer arithmetic: *(array + index)

Examples:

```
// Incorrect: No space for null terminator
char name[5] = "Alice";

// Correct: Space for "Alice" + null terminator
char name[6] = "Alice";

// String constant (read-only)
const char * name = "Alice";
```

Important Notes:
- Always allocate an extra byte for the null terminator in character arrays
- String literals like "Alice" are stored in read-only memory
- Modifying string literals can lead to undefined behavior
- Use `const char *` for string literals to prevent accidental modification

Memory Layout:

```
char name[6] = "Alice";
```

Your TODO list after this lecture

Make sure you know where the course webpage is (for Section **003**)

Read Policies and grading **very very very carefully**

Checkout the Getting started section.
Setup Docker and Github repo as instructed

Make sure you have access to
Campuswire, Gradescope, and Brightspace for **this section**

Checkout the Schedule for readings be completed before class