INTRODUCTION TO COMPUTER SYSTEMS

Tanzir Ahmed

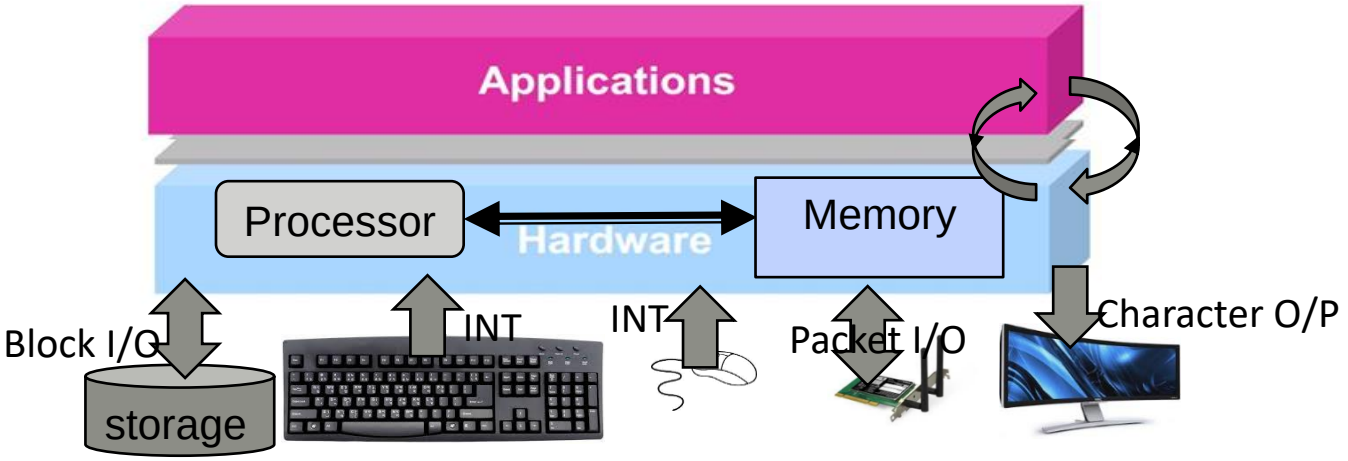
CSCE 313 Spring 2021

Lecture Outline

- **Part1**
  - ***Course Objectives and Outcome***
  - ***Logistics***
- Part2
  - *Course Context*
  - *Intro to OS*

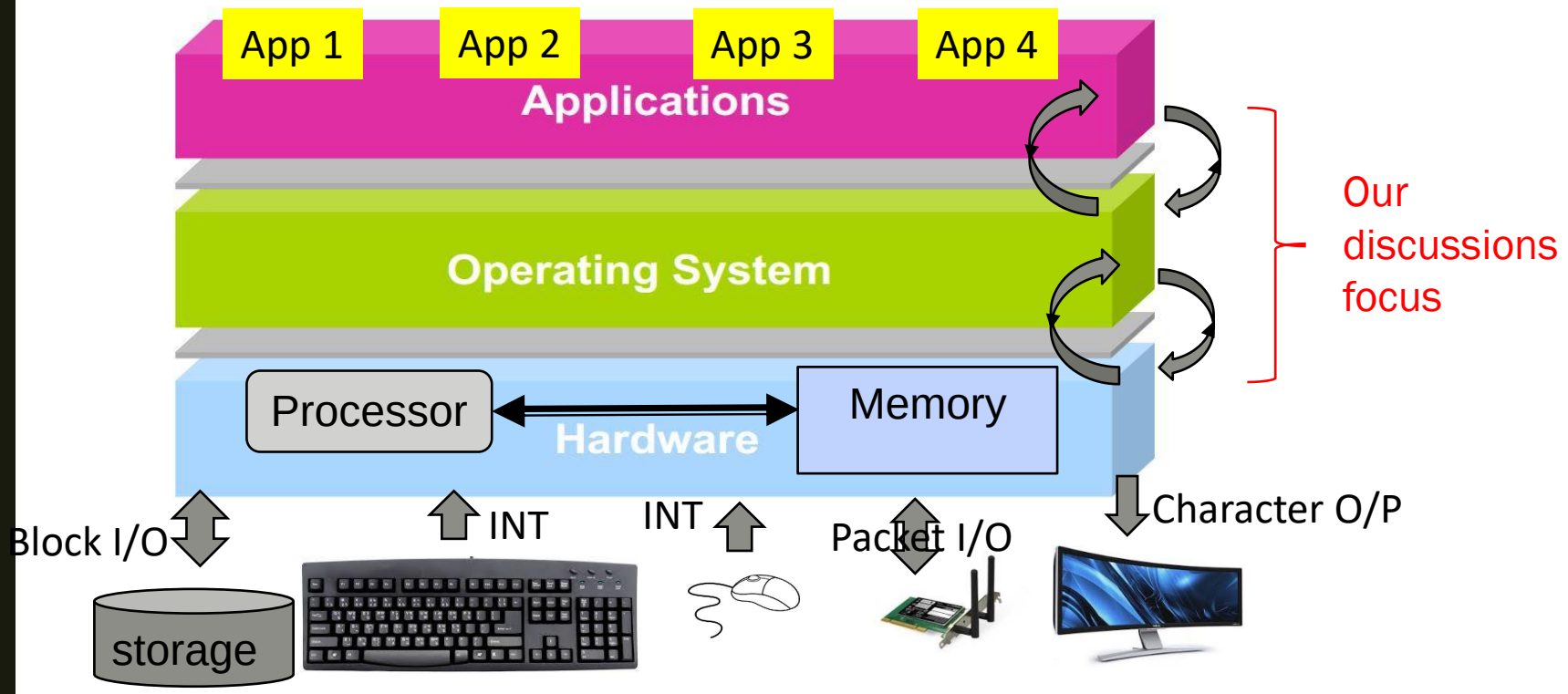
Essence of the Course

- To learn the software interface to the Operating Systems
  - *Why interface with OS? Or Why cannot the application software do everything by itself?*
  - *What is an OS anyways?*
- The following shows one program running in isolation (i.e., no other programs running) that uses CPU, memory, and other devices as it wishes
  - *This is what we have been taught so far*



Essence of the Course

- However, the reality is more complicated because of many application programs running at the same time:
  - *There must be an agent that controls sharing of resources and makes peace between*



Essence of the Course

- This is NOT an Operating System (OS) course
  - *Can take CSCE 410: Operating Systems*
- However, the content will be about OS quite a bit from a programmer's perspective, because:
  - *We want to harness OS basics to write high-performance, inter-dependent, and hardware and OS-aware applications*
  - *We can also borrow techniques used in OS as a huge software and apply those to everyday software development practices*
    - For instance, the concept of **timers** in OS is crucial for Asynchronous programming required in real-time systems

Learning Outcome

- In this course, you will learn how to use different OS services and features from our application programs:
  - *What is an operating system; its components; how it works*
  - *Execution of a program; function calls; interrupts; system calls; process control*
  - *File system*
  - *Concurrency and Synchronization*
  - *Inter-Process Communication (IPC)*
  - *Network/socket programming*
  - *Network Threats/Security Basics*

CSCE 313 Instruction Team

- Instructor: Dr. Sarker Tanzir Ahmed
  - *Instructional Assistant Professor*
  - *Background: PhD from TAMU CSE in High Performance Computing, Big Data, Web Crawling*
- Teaching Assistants:
  - Wenxuan Wu ww6726@tamu.edu
  - Di Xiao xiaodi_sean@tamu.edu
  - Rahul Parasa rsparasa@tamu.edu
- See Class Website->Logistics page for detailed contacts, office hours and Zoom meeting links

Textbook, Reference Books

- **Text**:
  - **T1:** *Computer Systems: A Programmer's Perspective,* Randal E. Bryant and David R. O'Hallaron, Prentice Hall, 2nd / 3rd edition
  - **T2:** *Operating Systems: Three Easy Pieces,* Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, available online at: http://pages.cs.wisc.edu/~remzi/OSTEP/
- **Reference**:
  - **R1** ***(Main):*** *Operating Systems – Internals and Design Principles, William Stallings, Ninth Edition*

How Success will be Measured

- 4-5 Quizzes (15%)
  - *Mixture of analytic and small programming problems*
- 2 Exams (35%): Midterm (15%) and Final (20%), both open-note
  - *Both exams will be through ecampus testing system*
  - *The midterm is 50 mins during class time; the final is registrar-allocated time*
- 6 to 7 Programming Assignments worth 50 points total
  - *The first one PA0 is tiny one, worth just 2%.*
  - *Rest 48% is equally divided in PA1-PAx*
  - *You will get Rubric and Grading instructions for each PA*
  - *Unless otherwise specified, each PA needs to be accompanied by a report and a demonstration video*
- Late penalty for Quizzes and PA's is 12%/day or 0.5%/hour
  - *100% penalty for exams (i.e., late exams not accepted)*

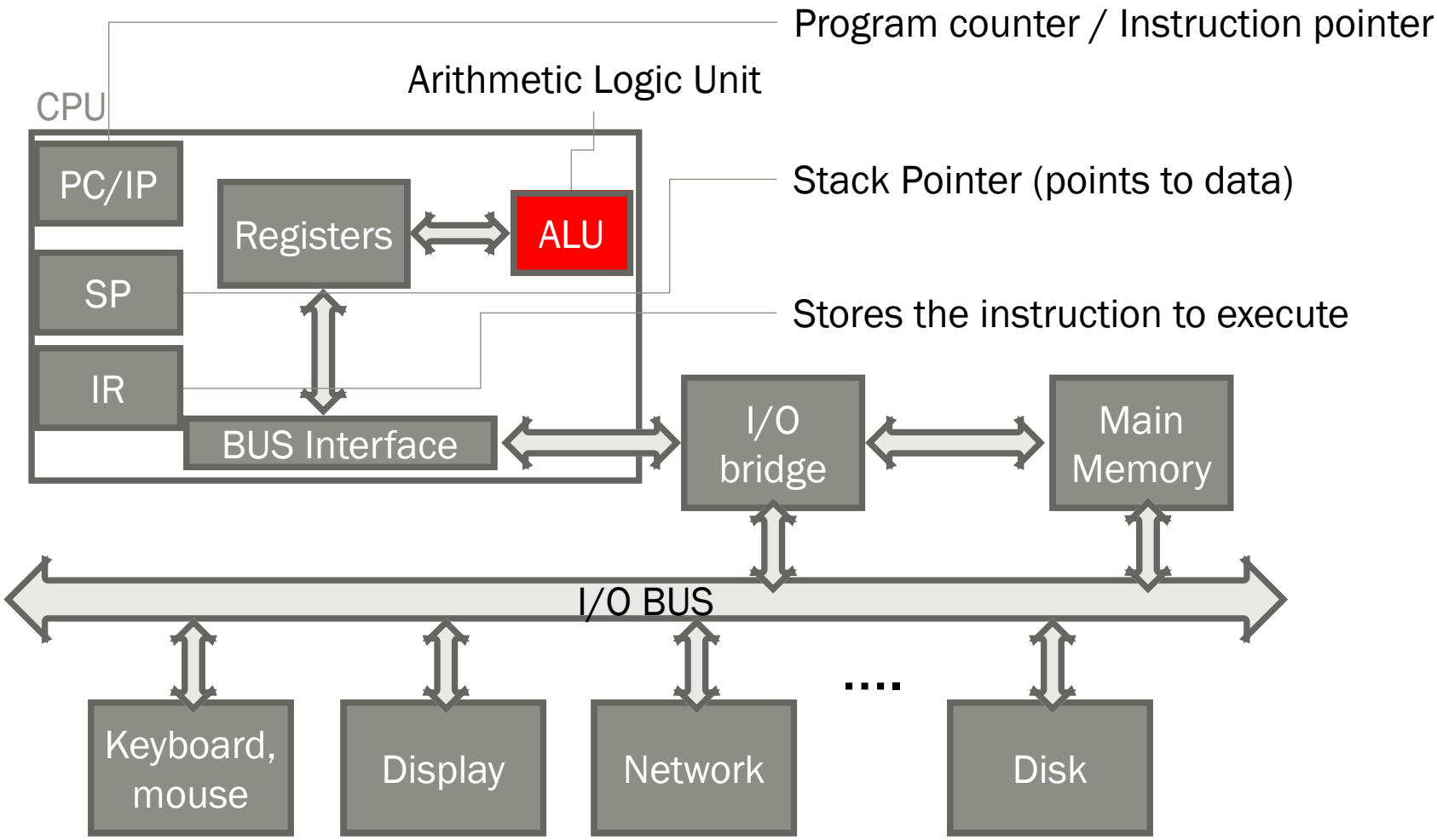
Method of Teaching

- Lectures and labs will happen synchronously during the howdy-schedule via Zoom
  - *See class website->Logistics for the Zoom links*
- However, there will be an asynchronous component to the lectures only
  - *Some slide sets and recorded video lectures will be made available prior to the lecture meetings*
  - *Some lecture meetings will be like flipped classes where you need prior preparation for class activities. These special lectures will be announced beforehand*
  - *The other lectures will be in the conventional format over Zoom*
- Labs are synchronous only
  - *You must physically or virtually attend the labs*
  - *Sessions may not be recorded, i.e., only the materials (e.g., slide sets, discussed code) will be available*
- Attendance in both is critical for success in this course
  - *However, we will not record attendance*

Lecture Outline

- Part1
  - *Course Objectives and Outcome*
  - *Logistics*
- Part2
  - *Introduction to Computer Systems*
    - Background – What we know already
    - What we are going to learn
  - *OS Intro*

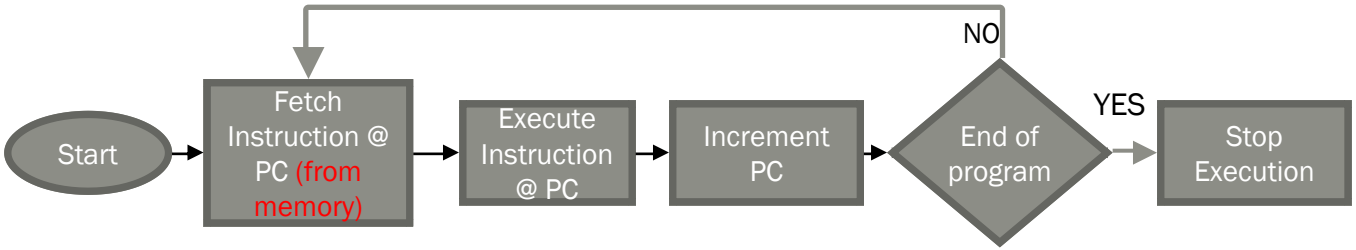
Background – From 312



Simplified hardware organization of a typical system

Program Execution

- A program is a collection of instructions
- Simple Instruction execution model:

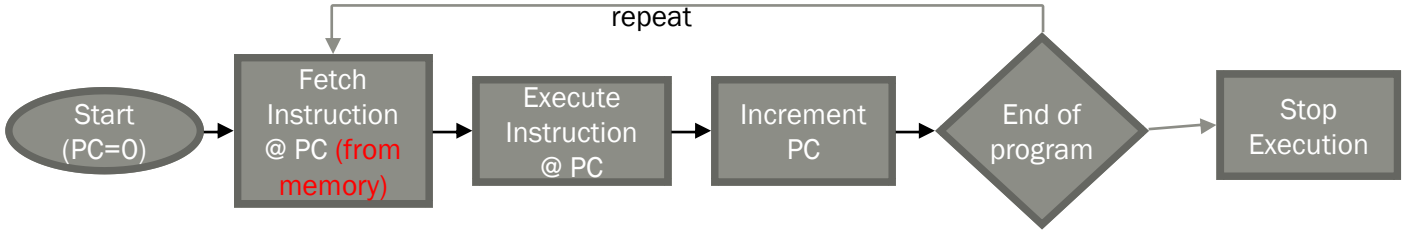


- CPU supports a small set of instructions (aka. Instruction set)
- To run a single program, load that program at addr=0 of the memory, and then unleash the above model
- We must improve this model with the ability to run multiple programs simultaneously
  - *Let us start with a little bit history first*

A Short Historical Tour

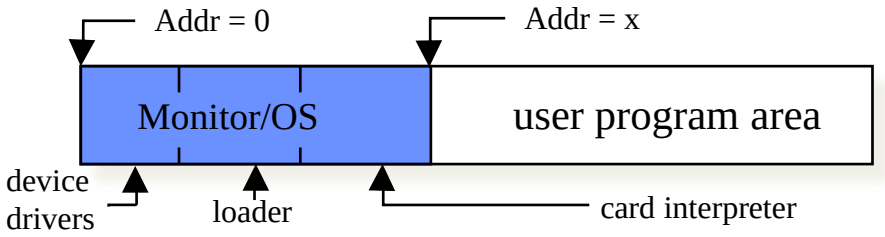
First Generation Computer Systems (1949-1956):
Single user: writes program, operates computer through console or card reader / printer

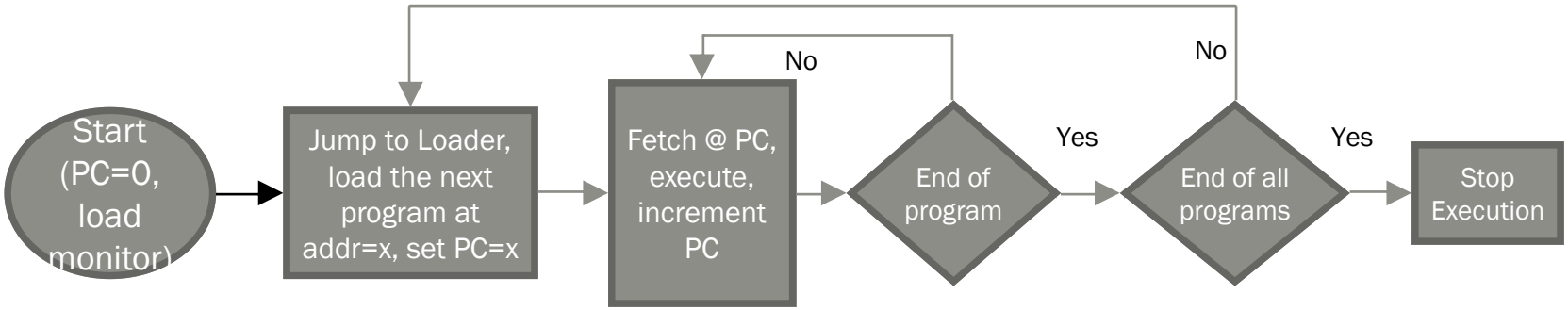
- *Absolute machine language*
- *No concept of an Operating Systems*
- *Hardware was simple, but not the programs written on it*
  - Compilation in head and loading by hand (later separate card readers were used for loading)



Second-Generation Computers (1956-1963)

- Automation of Load/Translate/Load/Execute
  - *Aka, Batch systems*
  - *Monitor programs*

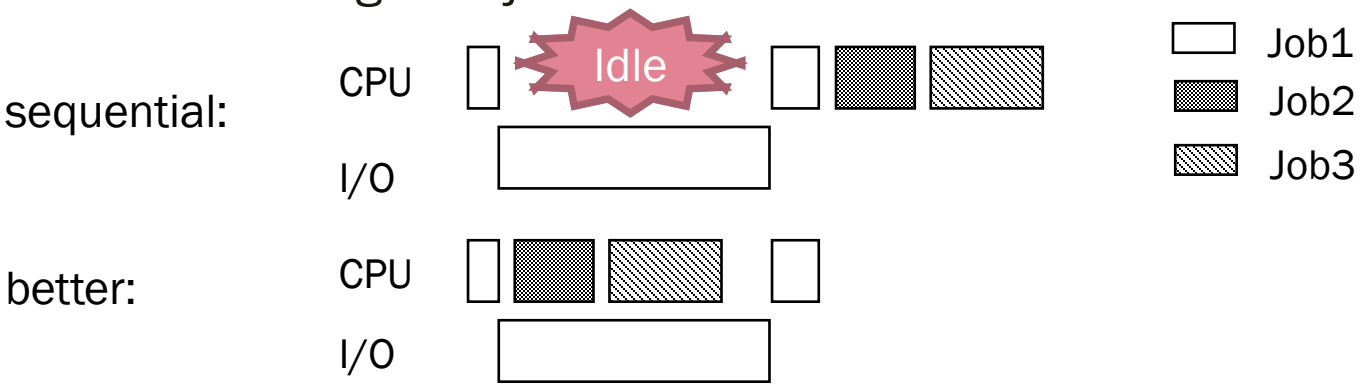




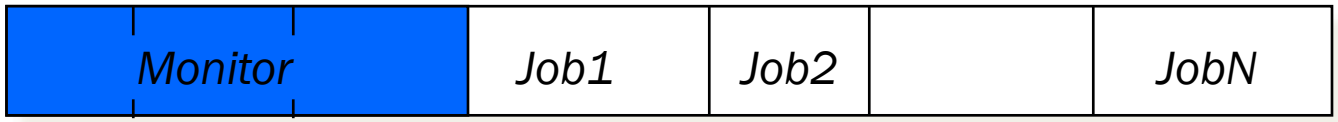
- Problem: Resource management and I/O still under control of programmer. Issues??
  - *Memory protection, security etc.*
  - *What if one program crashes?*

Third-Generation Computer Systems (1964-1975)

- Problem with batching: one-job-at-a-time



- Solution: **Multiprogramming**
  - *Job pools: have several programs ready to execute*
  - *Keep several programs in memory*



- New issues:
  - *Job scheduling,*
  - *Memory management*
  - *Protection*

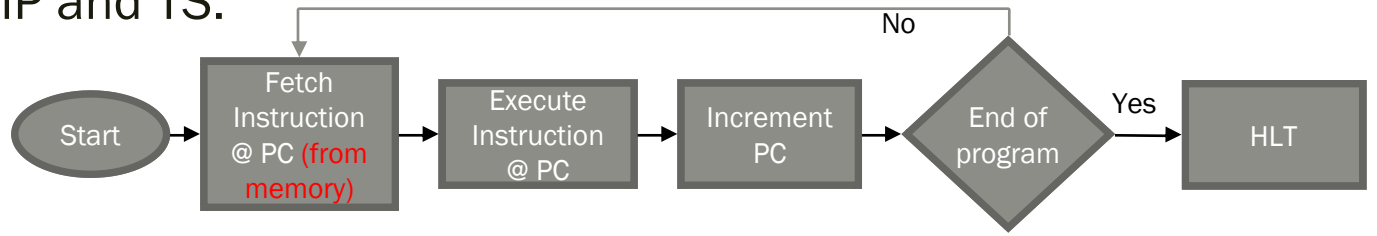
*Prerequisite for multiprogramming is Direct Memory Access (DMA): a sub-component of CPU that can drive I/O operations w/o involving CPU

Time Sharing (mid 1960s on)

- OS interleaves execution of multiple user programs with time quantum
  - *CTSS (1961): time quantum 0.2 sec*
  - *User owns the machine during the time quant.*
  - *Once time is up, user must wait in line behind other users*
- New aspects and issues:
  - *On-line file systems*
  - *resource protection*
  - *virtual memory*
  - *sophisticated process scheduling*
- Advent of systematic techniques for designing and analyzing OSs.

Multiprogramming(MP)/Time Sharing (TS) Implementation?

- Some changes are needed in the following model to support MP and TS:



- How do we extend this to support either MP or TS?
  - *Clearly, we need a mechanism to "**kick**" a program out at some point from the CPU*
  - *And "**bring it back**" later into the CPU*
- In summary, we need "Asynchrony" in programs

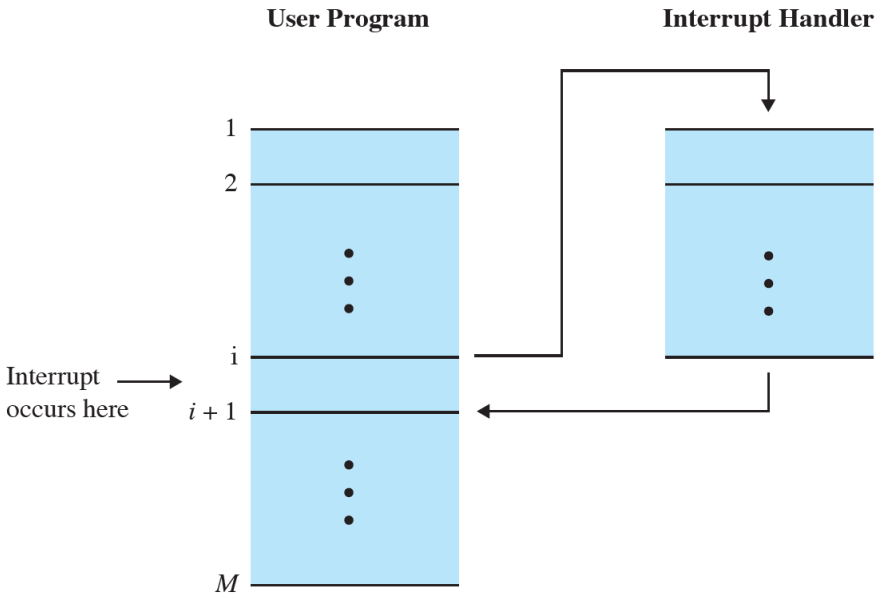
When to Kick-out a Program

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

For time sharing

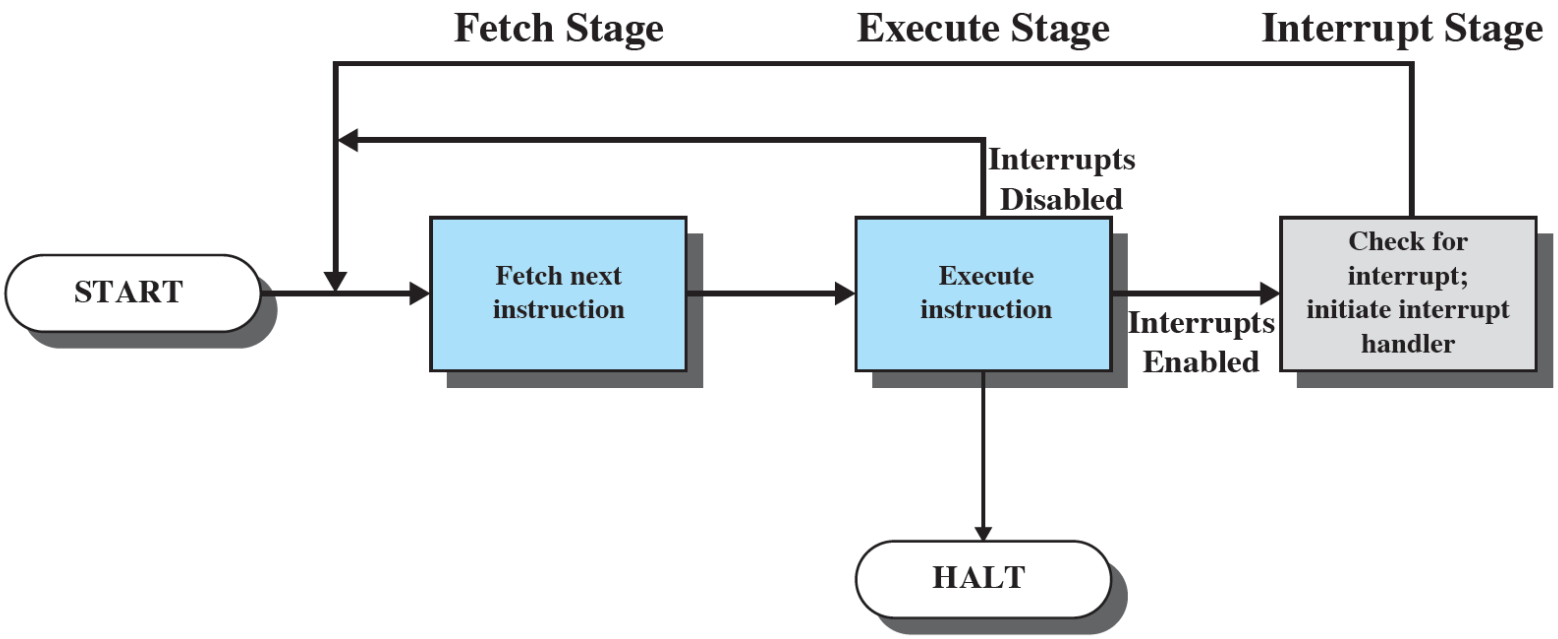
For multiprogramming

- Conceptually, what we need is "**Interrupts**"
- An interrupt "*pauses*" the current program and "*resumes*" it later



Source: W. Stallings book

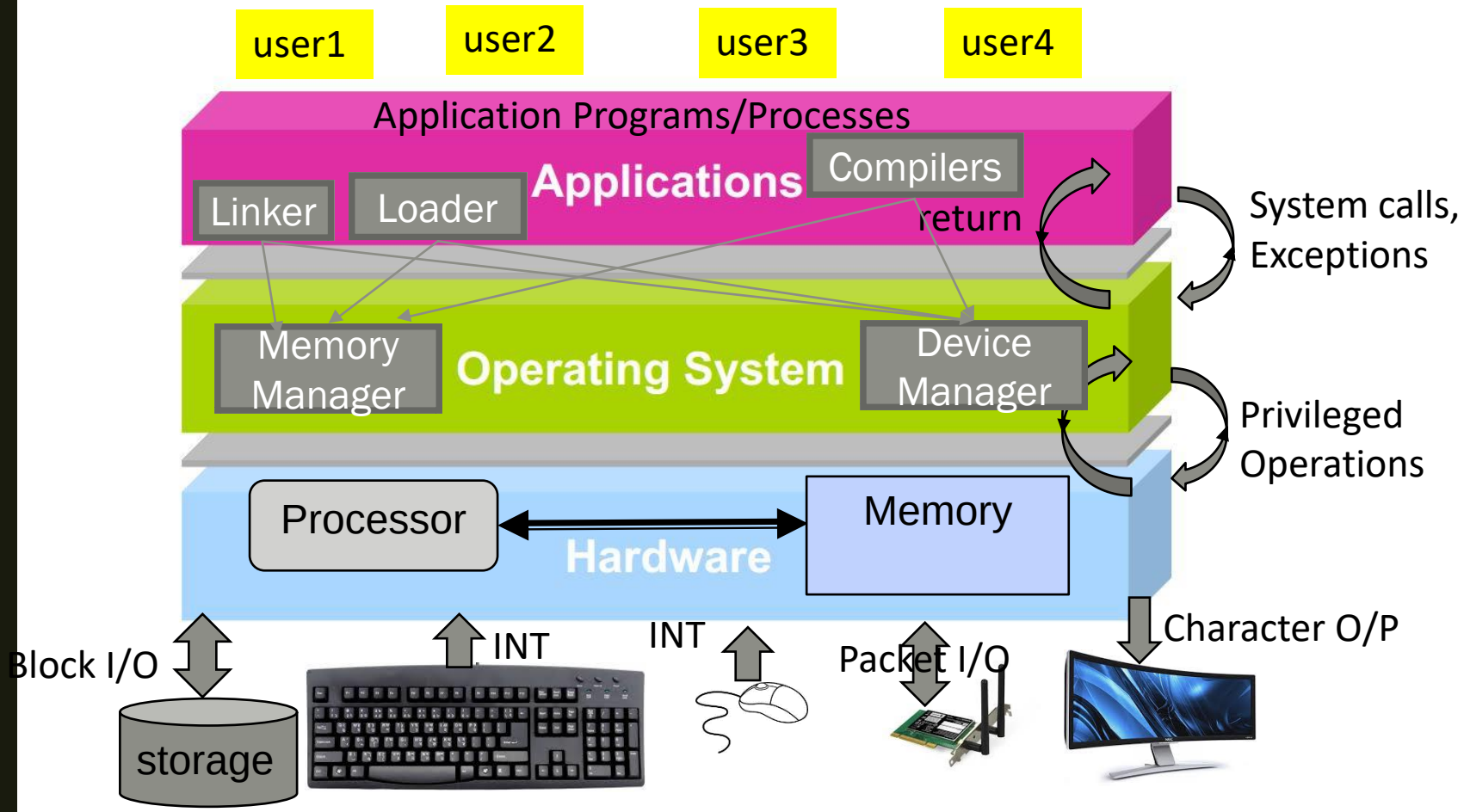
The Computing Model Changes



Computing Model to Support Interrupts

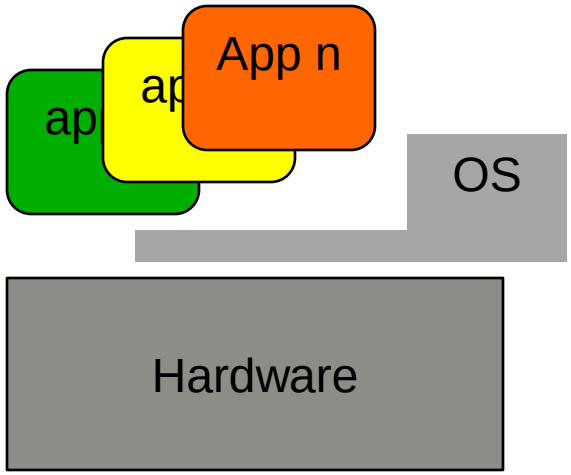
Source: W. Stallings book

A Modern Computer System



What is an operating system?

- Special layer of software that provides application software access to hardware resources
  - *Convenient abstraction of complex hardware devices*
  - *Protected access to shared resources*
  - *Security and authentication*
  - *Communication amongst logical entities*



What is an OS? – Virtualization View

- Many programs share the computer simultaneously
- How to make this sharing easy?
  - *Or not a nightmare to the app programmers?*
- Let us consider the following example code

```
#include <iostream>
using namespace std;

int main (int ac, char** av){
   for (int i = 0; i < 10; i++){
     cout<<"hello from "<<av[1]<< endl;
     sleep (1);
   }
}
```

CPU Virtualization

When 3 instances are run together, the following happens:

```
tanzir@cse-buggatti:~$ ./a.out Tanzir & ./a.out Tamanna & ./a.out Maya
[1] 39
[2] 40
hello from Tanzir
hello from Tamanna
hello from Maya
hello from Tanzir
hello from Tamanna
hello from Maya
hello from Tanzir
hello from Tamanna
hello from Maya
hello from Tanzir
hello from Tamanna
hello from Maya
hello from Tanzir
hello from Tamanna
hello from Maya
[1]-  Done                    ./a.out Tanzir
[2]+  Done                    ./a.out Tamanna
```

Memory Virtualization

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main (){
    int p = getpid();
    sleep (1);
    printf ("Address: %p, Content: %d\n" , &p, p);
}
```

Every program gets its own ID, even the same program running multiple times

Sleeps for a sec

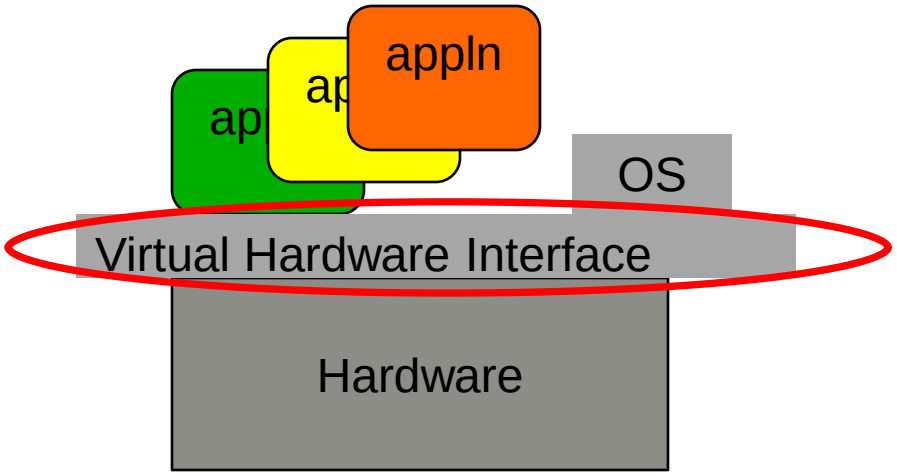
```
osboxes@osboxes:~/Documents$ ./a.out & ./a.out & ./a.out & ./a.out &
[1] 2570
[2] 2571
[3] 2572
[4] 2573
osboxes@osboxes:~/Documents$ Address: 0x7fffffffe084, Content: 2572
Address: 0x7fffffffe084, Content: 2570
Address: 0x7fffffffe084, Content: 2573
Address: 0x7fffffffe084, Content: 2571
```

These are coming from the shell, showing process IDs

Same address, different content

What's the OS's Role?

- In a single word: **Virtualization**
  - *The OS is hiding the physical machine and giving each program a **virtual machine** to play with*
  - *Each program can be agnostic of what else is going on in the System*
    - If apps wanted to know and manage all that, their job would be a lot more difficult



Goals of an Operating System?

- The OS controls and coordinates the use of system resources.

- Primary goal: Provide a convenient environment for a user to access the available resources (CPU, memory, I/O)
  - *Provide appropriate abstractions (files, processes, …)*
  - *"virtual machine"*

- Secondary goal: Efficient operation of the computer system
  - *Make sure that the OS does not kill performance too much*
  - *Efficient Resource Management*
    - Multiplexing: Create illusion of multiple resources from a single resource
    - Scheduling: "Who gets the resource when?"