INTRODUCTION TO COMPUTER SYSTEMS

Tanzir Ahmed

CSCE 313 Spring 2020

Lecture Outline

- **Part1**
  - ***Course Objectives and Outcome***
  - ***Logistics***
- Part2
  - *Course Context*

Essence of the Course

- To learn the software interface to the Operating Systems with a view to becoming more educated and efficient programmers
- This is, NOT an Operating System (OS) course
  - *Can take CSCE 410: Operating Systems*
- However, the content will discuss OS quite a bit from a programmer's perspective, because:
  - *We want to <u>harness OS basics</u> to write high-performance, inter-dependent, and hardware and OS-aware applications*
  - *We can also <u>borrow techniques</u> 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

Why do we need OS in 313?

- **Some** of you will **design** and **build** OS's or components of them
  - *Perhaps more now than ever, because there are many different OS's targeted for specific devices (e.g., robots, servers, signal lights, cameras, vehicle infotainment systems)*
- **Many** of you will create systems that utilize the core concepts in operating systems
  - *The concept of "<u>hierarchy</u>" and "<u>abstraction</u>" are all too imp*
  - *Whether you build software or hardware*
  - *The concepts and design patterns appear at many levels*
- **All of you** will build applications, etc. that utilize operating systems
  - *The better you understand their design and implementation, the better use you'll make of them.*

Learning Outcome

- In this course, you will learn how to use different OS services and features:
  - *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 Assistant:
  - Minhwan Oh (501-4)
  - Di Xiao (505-8)
  - Ananya Bothra (509-12)

Textbook, Reference Books

- Text:
  - *Operating Systems: Principles and Practice, Second Edition, Thomas Anderson and Michael Dahlin, Recursive Books, 2014.*
  - *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:
  - ***Main:** Operating Systems – Internals and Design Principles, William Stallings, Ninth Edition*
  - ***Secondary**: Understanding Unix/Linux Programming A Guide to Theory and Practice, Bruce Molay, Pearson Education Inc., 2003*
- Other Interesting Readings
  - *Computer Systems: A Programmer's Perspective, Randal E. Bryant and David R. O'Hallaron, Prentice Hall, 2011*

How Success will be Measured

- The course will have several quizzes/mini assignments, two exams, and a series of Programming Assignments (PA). The grade allocation is as follows:
- Total 100 points
  - *Quizzes (4-5): 15 points*
  - *2 Exams: 35 points*
  - *Programming Assignments (5-6): 48 points*
  - *Attendance (lab + lecture): 2 points*
- The grading scale is as follows:
  - *90 – 100: A*
  - *80 – 89: B*
  - *70 – 79: C*
  - *60 – 69: D*
  - *59 or below: F*

Exams

- Two exams: midterm 15%, final 20%
- Exams are closed book
- Cheat-sheets are allowed
  - *A4 page, double-sided, hand-written*
- Final Exam is NOT cumulative
  - *Will be based on material covered after the midterm*
- The exams are relatively “hard”
  - *Sample exams from previous semesters will ease them significantly*

Programming Assignments

- There will be roughly 7 (+/- 1) Programming Assignments (PA) assigned weekly or bi-weekly basis.
- Each PA will be individual effort
  - *You are not allowed to work in teams*
  - *You are not supposed to collaborate with other students on any peic*
- Worth 48% of your grade
- PAs are posted and submitted through Ecampus

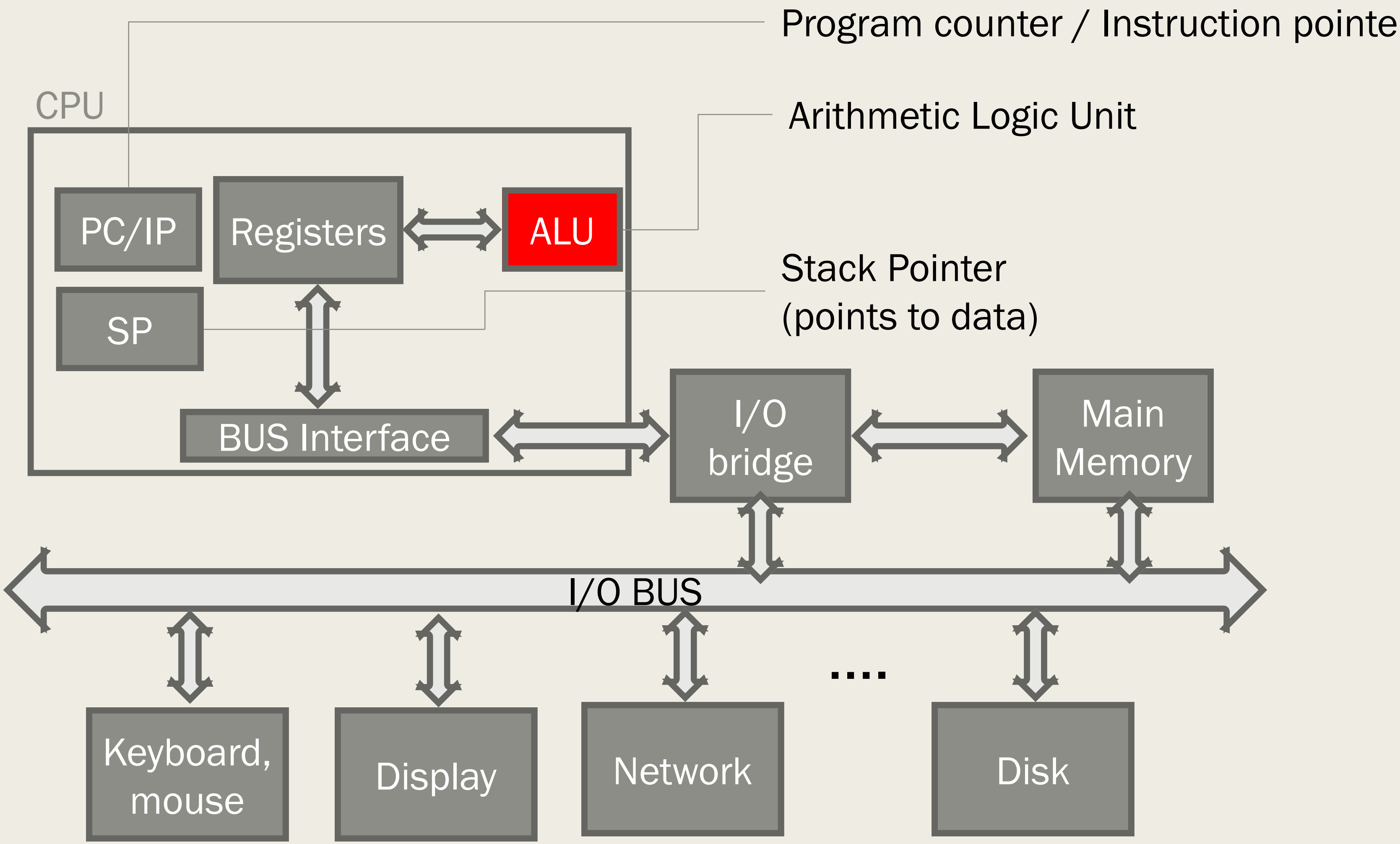
Late Policy for Programming Assignments

- **Penalty:** Unless stated otherwise, lateness is penalized as a simple linear function of hours late
  - ***0.5% /hour*** *of your score*
- As a result on late submissions you loose:
  - ***12% / day***
  - *100% if more than 8 days late*

Lecture Outline

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

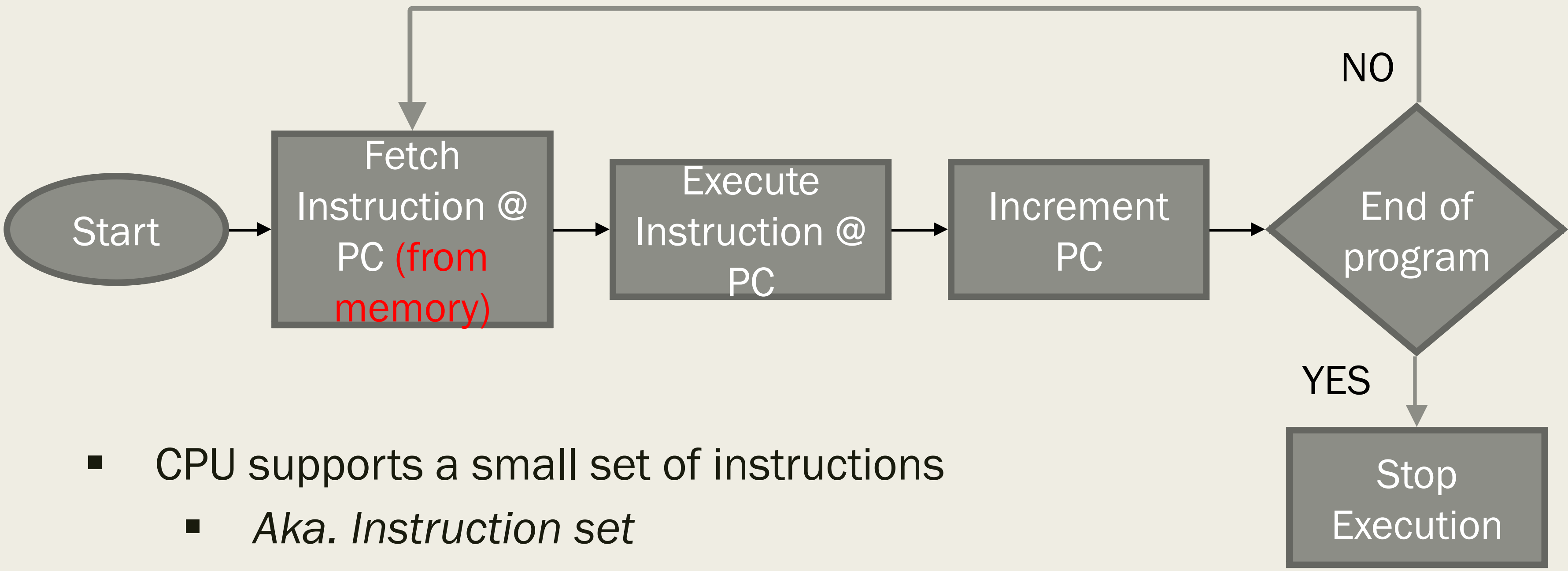
Background – From 312



Simplified hardware organization of a typical system

Instruction Execution

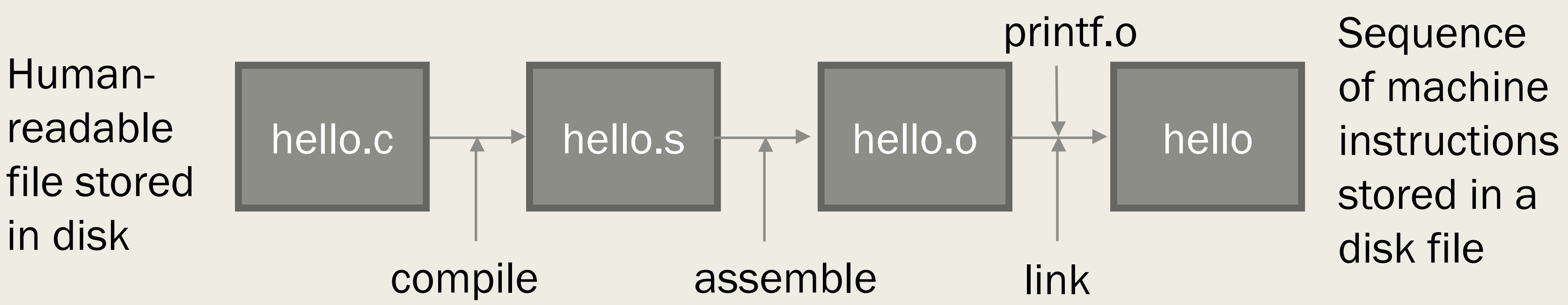
- Simple Instruction execution model



- CPU supports a small set of instructions
  - *Aka. Instruction set*
- Now, we must think about how to run a program using this model

Before Even That: Preparing a Program to Run

- Write the program in a high-level language e.g., C
- Convert the human-readable C program to an assembly program (i.e., `hello.as`) by ***Compilation***
    - *Assembly program still human-readable*
- ***Assemble*** the program into relocatable machine code (`hello.o`)
- ***Link*** the program to other required libraries that are precompiled and make a single executable (`hello`)



Background – Loading the hello program

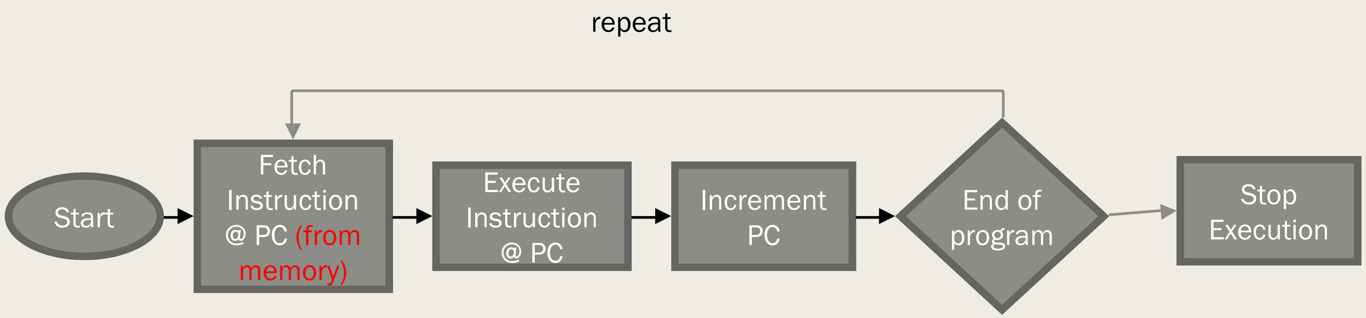
- A program although stored in the disk, cannot run while in the disk
  - *Why?? Disks are million times slower than a CPU speed*
  - *It must be loaded to the main memory before running*
- Load the executable program `hello` (its code and data) into main memory
- Set the PC to the first byte of the first instruction of `hello`
- Let the CPU do what it does – execute an instruction, proceed to the next one and repeat

Background – A Few Questions

- Who loads a program into main memory?
  - *A part of the Operating System called **loader***
  - *But what if you like another faster loader from another vendor*
- Alternatives??
  - *One solution, each programmer write his own loader. But is that feasible?*
  - *How about compiler, linker, assembler etc.?*
  - *How about other common utilities (e.g., file explorer, shell)*
- Seems like all would be best IF these utilities can be obtained from third party easily, but they go through a supervisory program called the Operating System (OS)
  - *The OS makes sure that everything is working safely and securely*
- That is the whole point of an OS

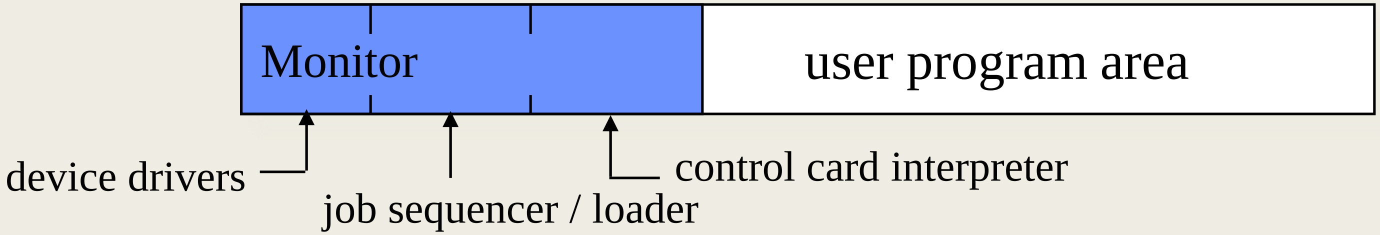
A Short Historical Tour

- **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 simple, but not the programs written on it*
    - Programmer must compile, load, and run program



Second-Generation Computers (1956-1963)

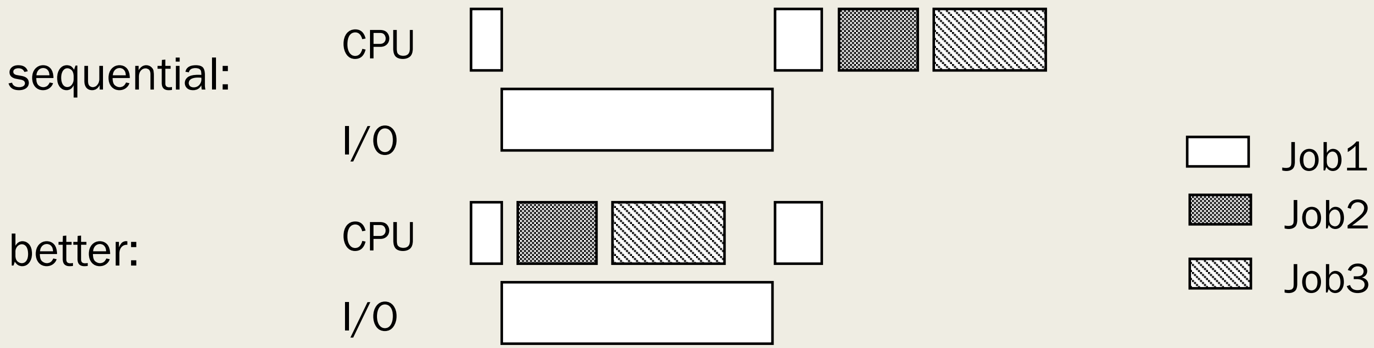
- Automation of Load/Translate/Load/Execute
  - *Batch systems*
  - *Monitor programs*



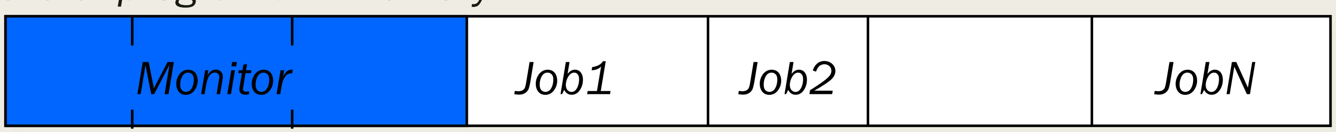
- Problem: Resource management and I/O still under control of programmer. Issues??
  - *Memory protection, security etc.*
  - *What if one program crashes*

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*

I/O Does not Involve CPU?? Why and How?

- At early stages of computing, CPU was in fact in charge of I/O operations. Let us consider an example
- A 1GHz CPU trying to print a page in the printer (an I/O device), which can print 1 page/sec
- If the CPU had to print 1 page at a time, its rate of operation would reduce from 1B instructions/sec to 1 instruction/sec
  - *A 1B-fold slowdown*
- To read a byte from disk, which requires >1ms, the CPU would go to 1000 instructions/s
  - *A 1M times slowdown*
- Clearly, there is room for improvement

Direct Memory Access (DMA)

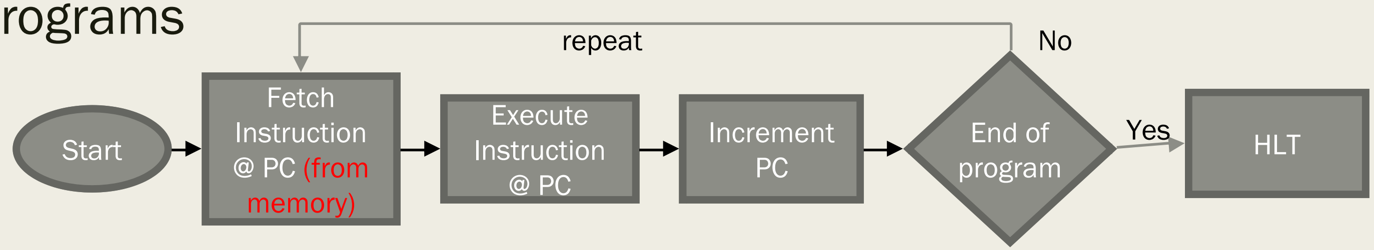
- Instead of involving CPU in every step of I/O, another hardware component called the DMA controller drives the device
  - *The CPU only starts the transfer by sending a high-level instruction to DMA controller*
  - *The DMA controller does the I/O by copying data to/from the given location*
  - *The device controller speaks to the DMA controller along the way, and does not bother the CPU*
  - *At the end, the DMA controller notifies CPU that the transfer is done*
- This saves a lot of CPU cycles
- This is called **Direct Memory Access (DMA)**
- This is the primary enabler for Multiprogramming

Time Sharing (mid 1960s on)

- Remote interactive access to computer: "Computing as Utility"
- OS interleaves execution of multiple user programs with time quantum
  - *CTSS (1961): time quantum 0.2 sec*
- User returns to own the machine
- 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?

- So far, we only have this, which works perfect for sequential programs



- How do 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

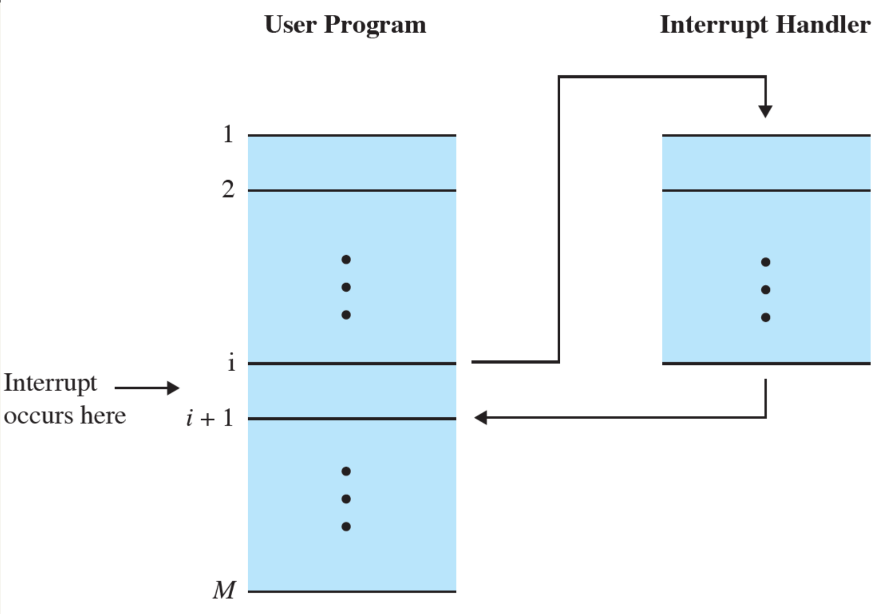
When to Kick-out a Program

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

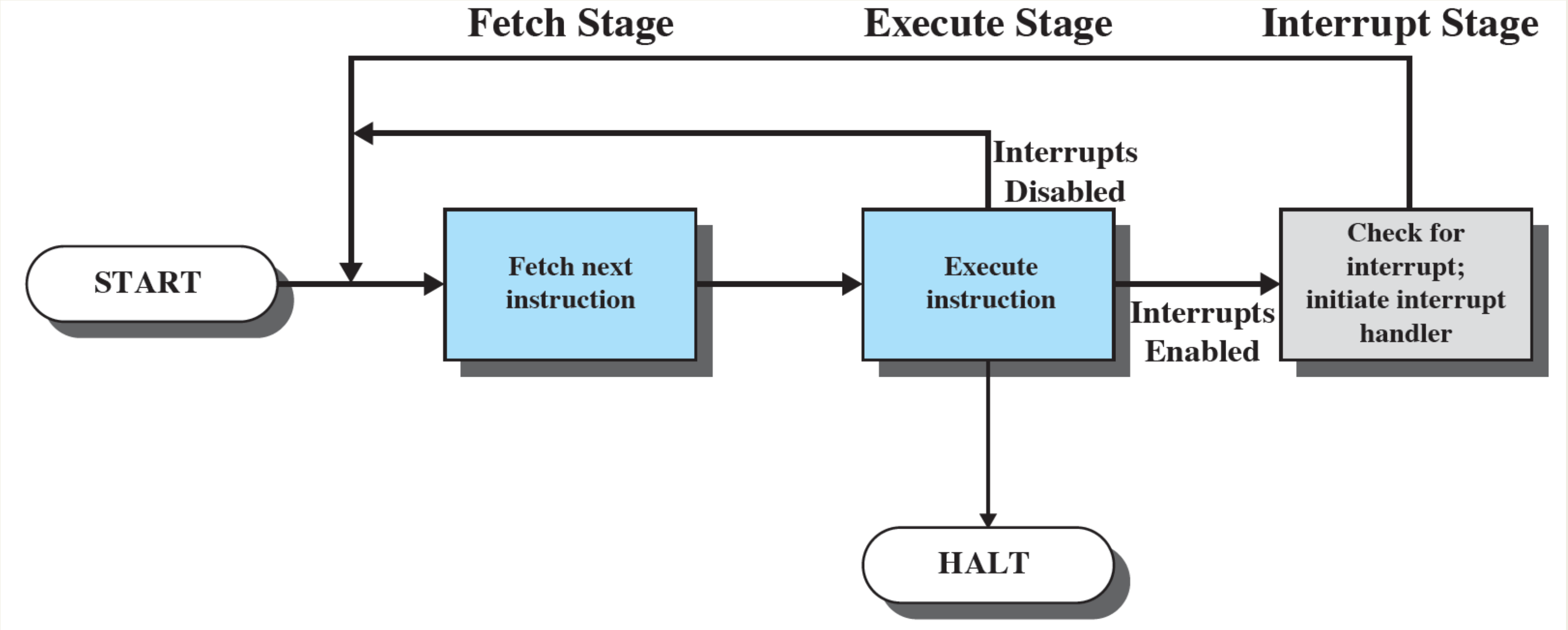
For multiprogramming

- Conceptually, what we need is this:
- Here come "**Interrupts**"



Source: W. Stallings book

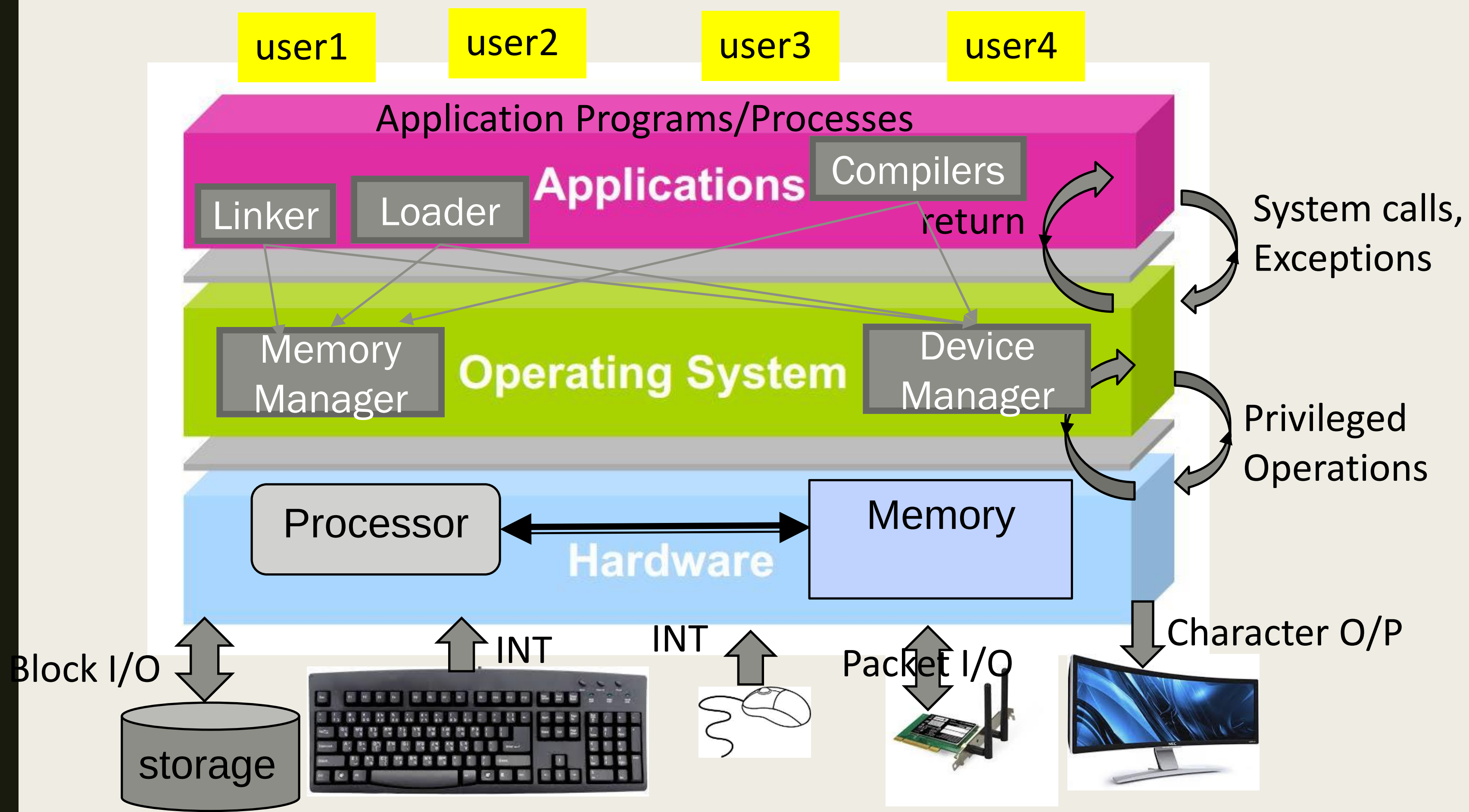
The Computing Model Changes



Computing Model to Support Interrupts

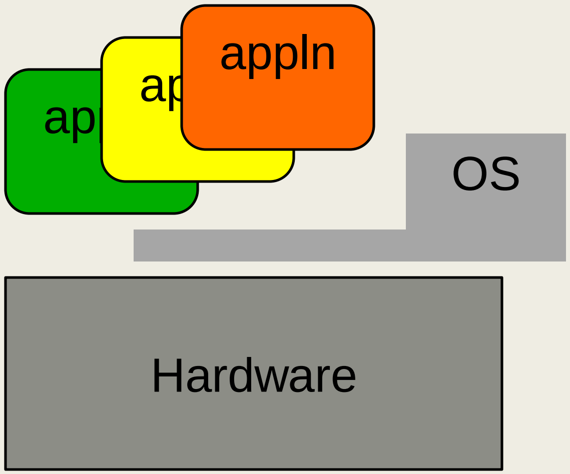
Source: W. Stallings book

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*



Credits

- Slides are based on Dr. Aakash Tyagi and Dr. Riccardo Bettati's lectures
- Programming Assignments are mostly taken from Dr. Bettati
- Other sources are credited along the way