

**Department of Physics,  
Computer Science & Engineering**

CPSC 410 – Operating Systems I

# Operating System Overview

Keith Perkins

Original slides by Dr. Roberto A. Flores

# Topics

---

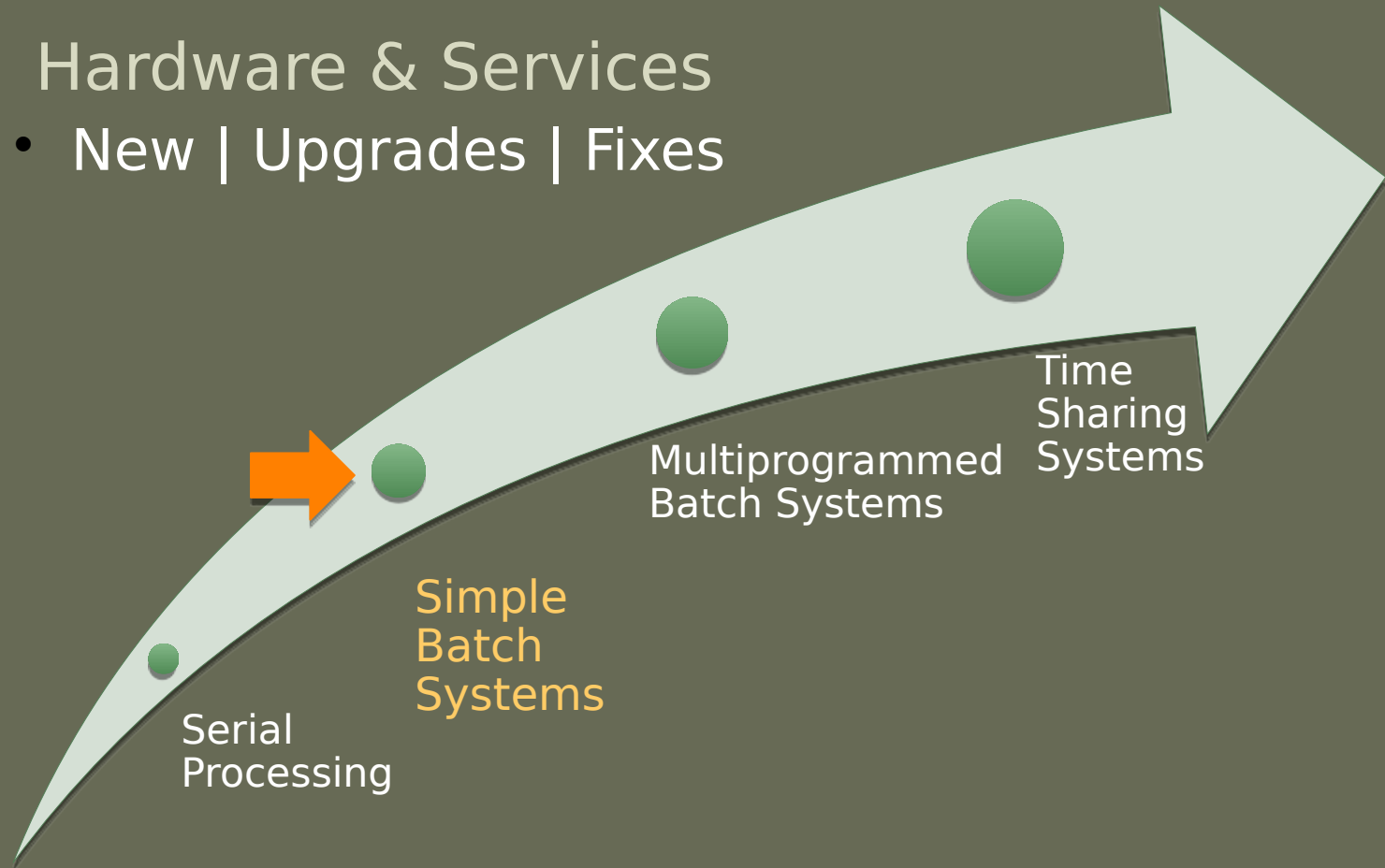
## • OS evolution

- Batch, Multi-programming, Time sharing
- Achievements
  - Process, Memory management, Scheduling, System structure

# Evolution

- Reasons for OS to evolve

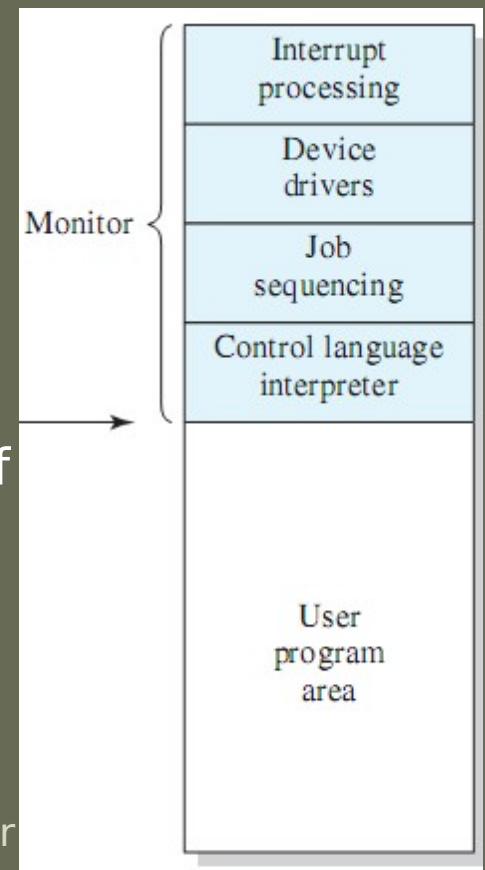
- Hardware & Services
  - New | Upgrades | Fixes



# OS Evolution

## ■ Simple Batch Systems

- improving computer utilization
  - programmer has no direct access to computer
  - operator batches jobs, feeds them to an input device, then...
- Monitor (aka Batch OS)
  - program controlling the execution of jobs
  - 1. monitor reads next job & yields control of CPU to the job
    - “control is passed to a job” : CPU starts running user program
  - 2. user program ends & monitor continues running again
    - “control is returned to the monitor” : CPU runs monitor



# OS Evolution

---

- Simple Batch Systems (II)
  - Job Control Language (JCL)
    - Instructions meant for the monitor (like pre-processing)
      - `$JOB <job info>$DD <data>$EXEC<source code>`
    - Memory protection
      - Memory where monitor resides is out-of-bounds for jobs
    - Timer
      - Notifies when jobs run longer than anticipated
    - Privileged instructions
      - Instructions that only the monitor can execute (e.g., load job)
    - Interrupts
      - Signals giving CPU a degree of flexibility

# OS Evolution

- Simple Batch Systems (II)

- Job Control Language (JCL)

- Instructions meant for the monitor (like pre-processing)
  - `$JOB <job info>$DD <data>$EXEC<source code>`

- Memory protection

- Memory where monitor runs

- Timer


- Notifies when jobs finish

- Privileged instructions

- Instructions that only monitor can execute (on job)

- Interrupts

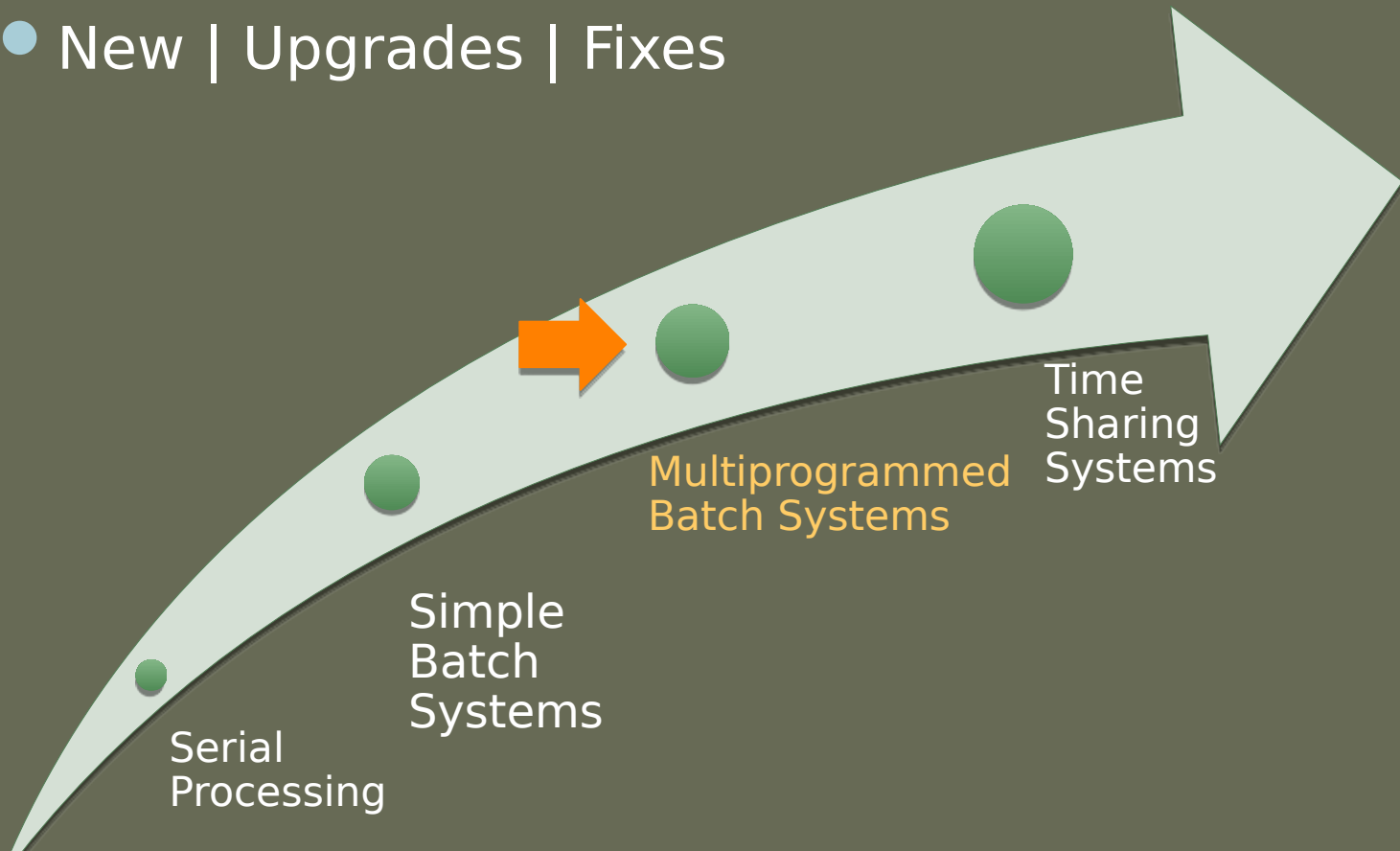
- Signals giving CPU a degree of flexibility



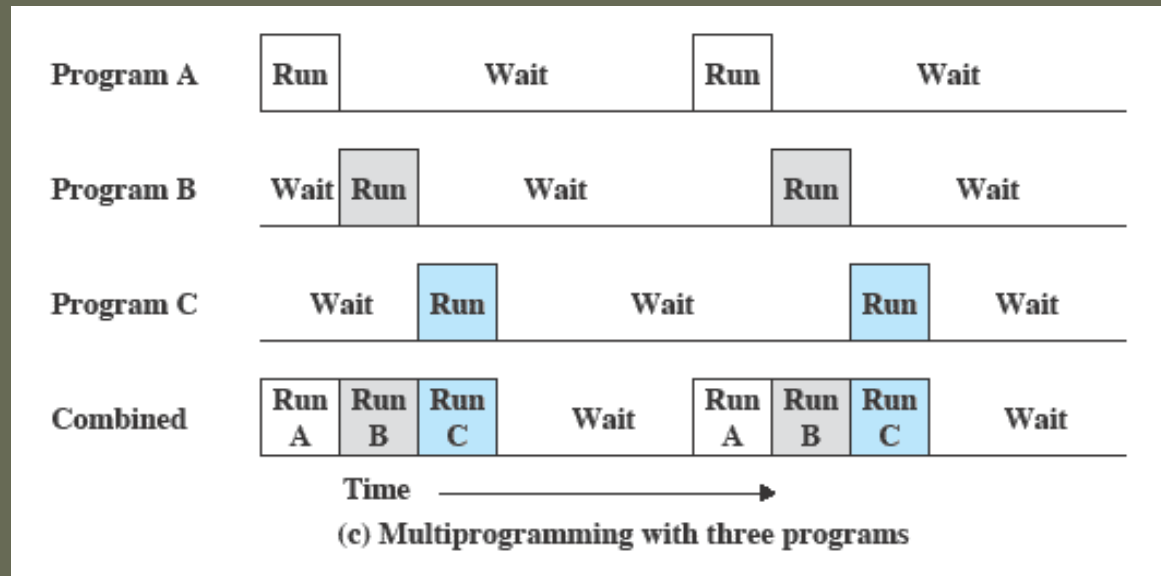
	User Mode	Kernel Mode
Applies to...	User programs	Monitor
Memory access	Restricted	Unrestricted
Instructions	Limited	Unlimited

# Evolution

- Reasons for OS to evolve
  - Hardware & Services
    - New | Upgrades | Fixes



# Multiprogramming



- Multiprogramming
  - also known as multitasking
  - memory is expanded to hold three, four, or more programs and switch among all of them

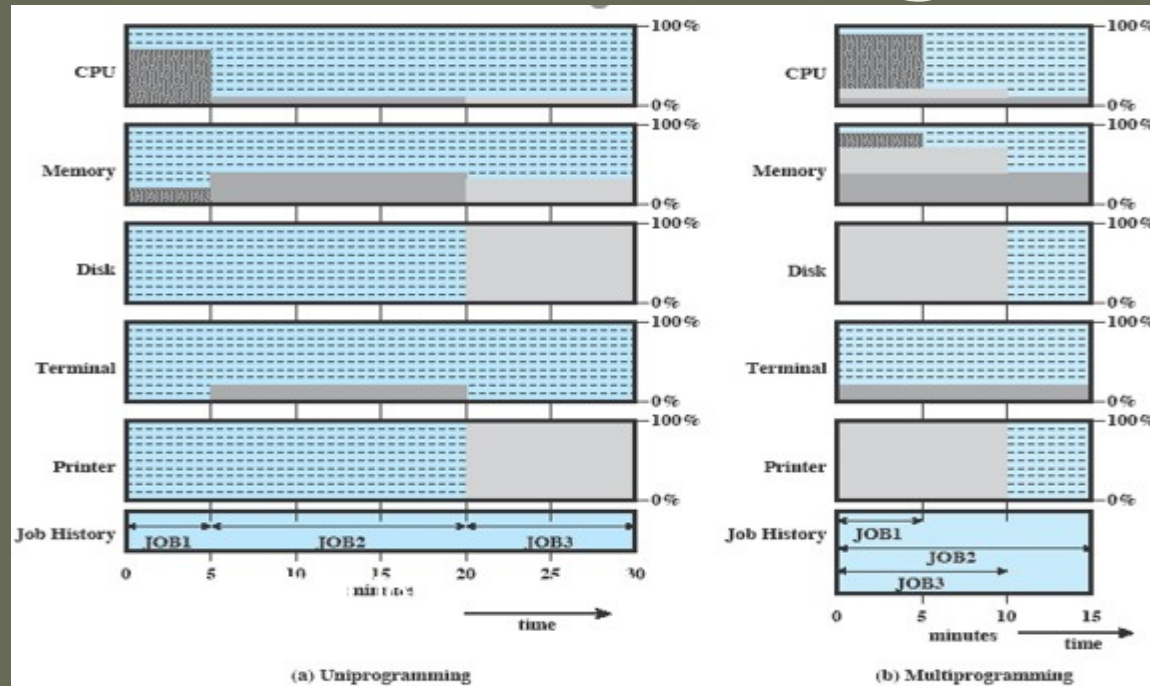


# Multiprogramming Example

**Table 2.1 Sample Program Execution Attributes**

	<b>JOB1</b>	<b>JOB2</b>	<b>JOB3</b>
<b>Type of job</b>	Heavy compute	Heavy I/O	Heavy I/O
<b>Duration</b>	5 min	15 min	10 min
<b>Memory required</b>	50 M	100 M	75 M
<b>Need disk?</b>	No	No	Yes
<b>Need terminal?</b>	No	Yes	No
<b>Need printer?</b>	No	No	Yes

# Utilization Histograms



Job1 uses 70%CPU, Job2 and Job3 use 10 %

CPU Utilization:

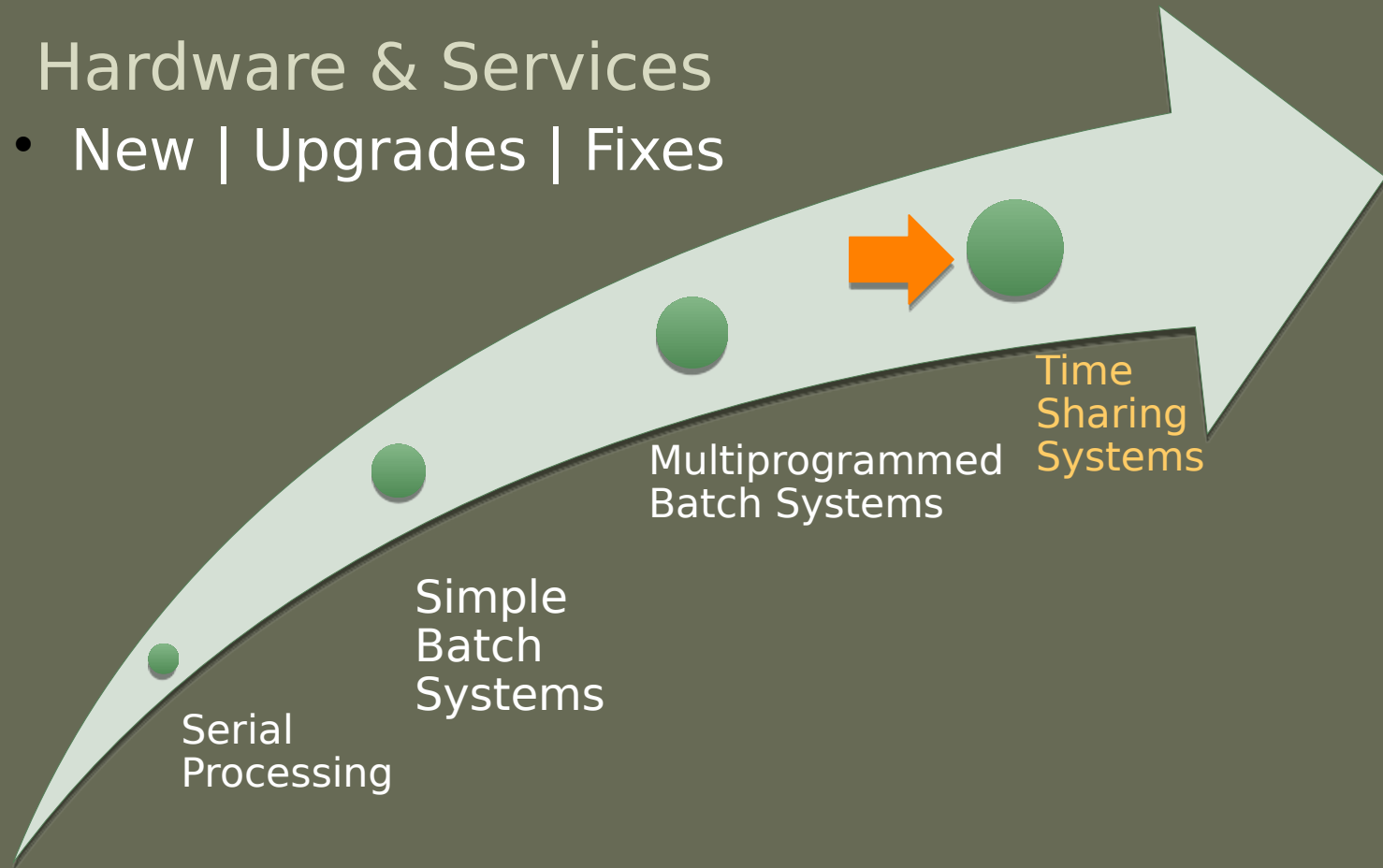
Uniprogramming= $(.7*5 + .1*25)/30 = 20\%$

Multiprogramming =  $(.9*5 + .2*5 + .1*5)/15 = 40\%$

Know how to calculate utilization!

# Evolution

- Reasons for OS to evolve
  - Hardware & Services
  - New | Upgrades | Fixes



# OS Evolution

---

- Time Sharing Systems
  - Users access system simultaneously using terminals
  - Time Slicing
    - Timer generates interrupts every 0.x seconds (small number)
    - OS preempts current program and loads in another
    - Preempted program & data are stored in memory
    - If memory is full kick victim program to disk
      - *This is a time consuming operation, choose victim wisely*
  - Multi-Programming vs. Time sharing

# OS Evolution

- **Time Sharing Systems**

- Users access system simultaneously using terminals
- Time Slicing
  - Timer generates interrupts every 0.x seconds (small number)
  - OS preempts current program and loads in another
  - Preempted program & data are stored in memory
  - If memory is full kick victim program to disk
    - *This is a time consuming operation, choose victim*

	Multi-programming	Time sharing
Objective	Maximize processor use	Minimize response time
Source of instructions	Job Control Language (JCL)	Commands entered in terminal

# Achievements

---

- Major advances in OS development
  - Processes
    - Definition, Errors, Components
  - Memory management
    - OS responsibilities, Virtual memory
  - Scheduling & resource management
  - System structure

# Process

---

A *process* is just an instance of a running program

# Process - Causes of Errors

- **Improper synchronization**

- a program must wait until the data are available in a buffer
- improper design of the signaling mechanism can result in loss or duplication



- **Failed mutual exclusion**

- more than one user or program attempts to make use of a shared resource at the same time

- **Nondeterminate program operation**

- program execution is interleaved by the processor when memory is shared
- the order in which programs are scheduled may affect their outcome

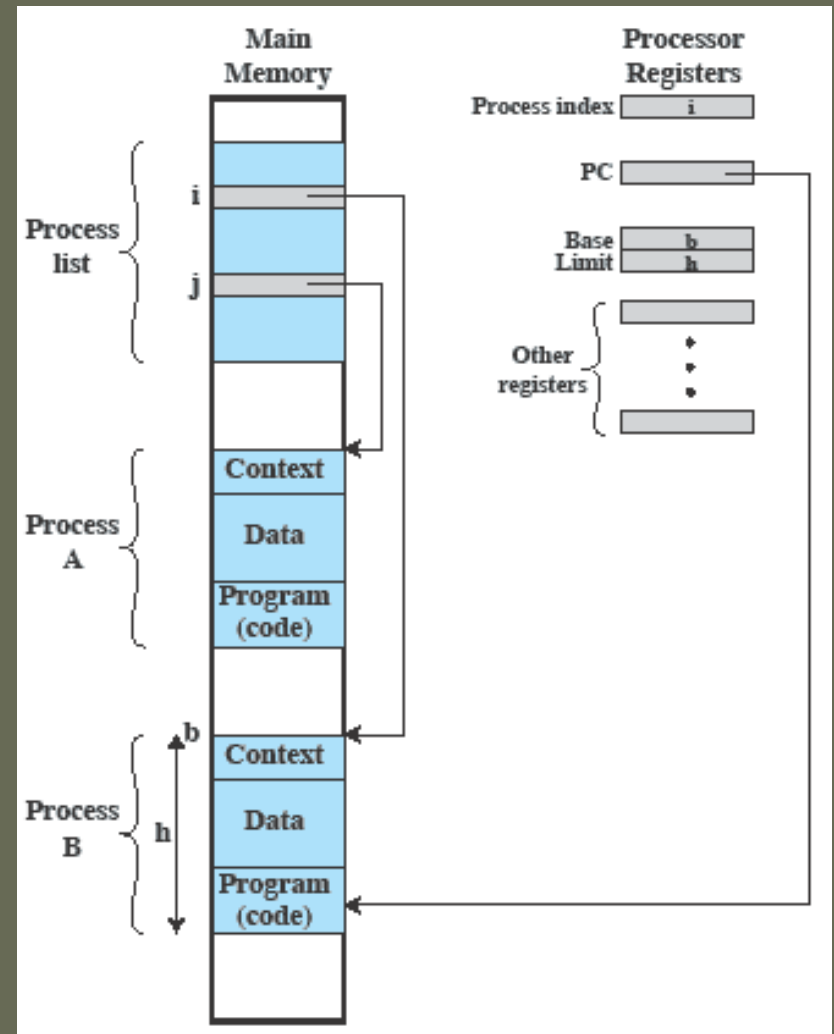
- **Deadlocks**

- it is possible for two or more programs to be hung up waiting for each other
- may depend on the chance timing of resource allocation and release



# Process Management

- Processes (components)
  - Executable code
  - Data
    - e.g., variables, buffers, ...
  - Execution context (aka “process state”)
    - internal data used by the OS to control the process
      - e.g., registers, priority, whether it is waiting for an I/O event



# Achievements

---

## Memory management (OS responsibilities)

Process isolation

...are prevented from interfering with each other

Automatic allocation & management

...are not concerned about their own allocation

Support of modular programming

...are able to add/remove modules

Protection & access control

...are assured the integrity of data in shared memory

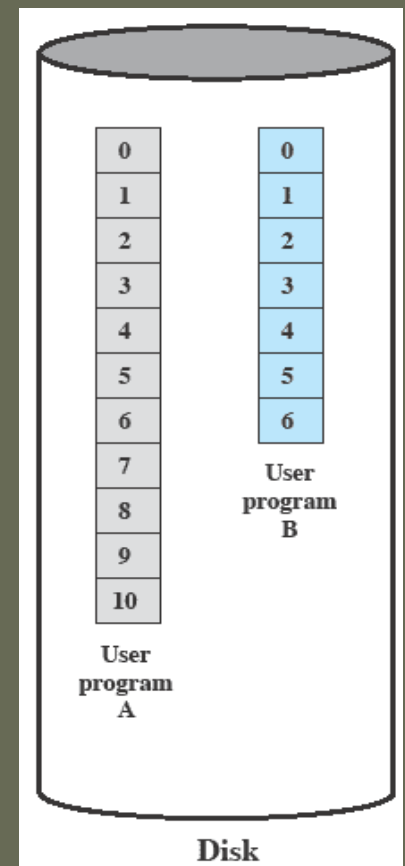
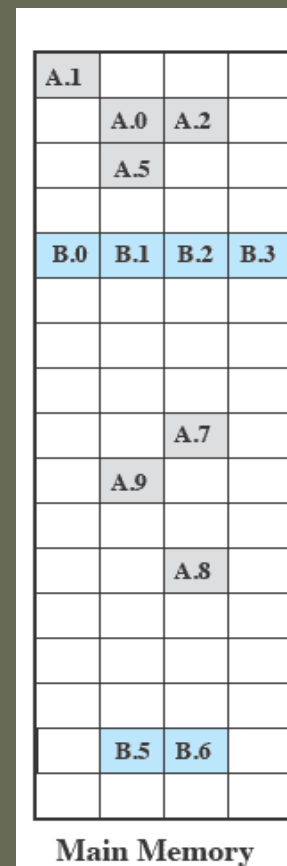
Long-term storage

...are able to store data for later runs (including power down)

How to handle simultaneous processes if they **do not fit** all in **main memory**?

# Achievements

- Memory management (Virtual Memory)
  - Handling many processes with limited memory
  - Paging
    - Processes are broken into blocks (aka **pages**)
      - Pages can be anywhere in main memory
    - CPU uses **virtual addresses** to find instructions/data
      - Addresses are **page** number + **offset** within page



# Achievements

---

- Scheduling & resource management
  - OS **manages** resources (main memory, I/O devices, processors) and **schedules** their use by processes
  - Fairness
    - Equal processes given equal and fair access to resources.
  - Differential responsiveness
    - Different processes treated differently according to their needs.
  - Efficiency
    - Overall performance is a goal
      - maximize throughput
      - minimize response time
      - accommodate as many users as possible

These criteria conflict (what's the right balance?)

# Achievements

---

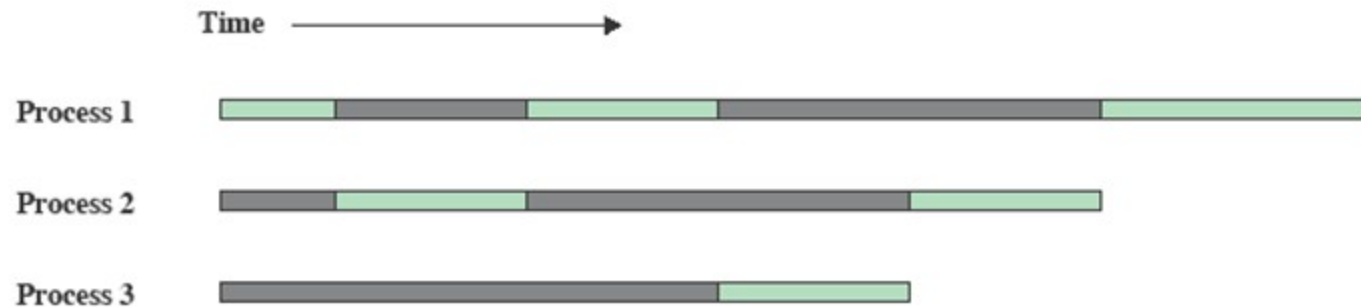
- System structure
  - Until Recently
    - OS are monolithic programs
    - processes are linearly executed
  - Now Microkernel Architecture
    - Keep essential functions in kernel
      - memory addressing, scheduling, ...
    - Modularize the rest (towards object-oriented approach)
      - modules dynamically linked, easier to replace
  - Advantages
    - low coupling – dynamically load modules when needed, encourages flexible API design – need new scheduler? Provide library that meets scheduler API, load at runtime
    - works well with distributed OS – illusion of unified memory & resources

What to do about it?

# Achievements

---



- System structure
  - Symmetric multiprocessing (add CPUs)
    - 2+ CPU run in parallel (hardware + OS exploiting it)
    - Processes scheduled to separate CPU (but share resources)
  - Multi-threading (divide processes)
    - Process broken into parts that run concurrently (own thread)
    - Process =  $\sum$  (threads = concurrent unit of work)
    - Programmers control scope & timing of concurrency



(a) Interleaving (multiprogramming, one processor)



(b) Interleaving and overlapping (multiprocessing; two processors)

 Blocked       Running

## Multiprogramming and Multiprocessing

# Achievements

---

Symmetric multiprocessing

## **Challenges**

- **Scheduling**: Scheduling across CPU cores must be coordinated
- **Synchronization**: Access to resources must be synchronized
- **Memory management**: Page reuse
- **Fault tolerance**: Graceful degradation

## **Parallelism opportunities**

- Multiprogramming & multi-threading in each processor
- A process could and probably does have its threads executed in different CPUs



# Topics

---

- OS evolution
  - Serial, Batch, Multi-programming, Time sharing
- Achievements
  - Process, Memory management, Scheduling, System structure

Done!