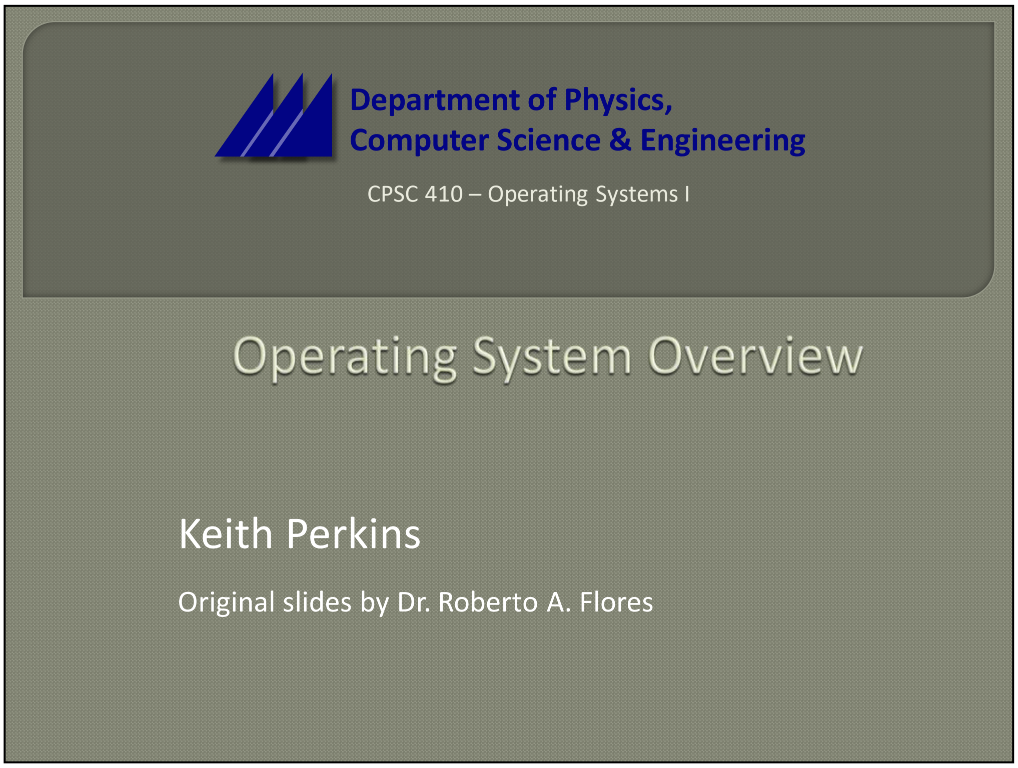


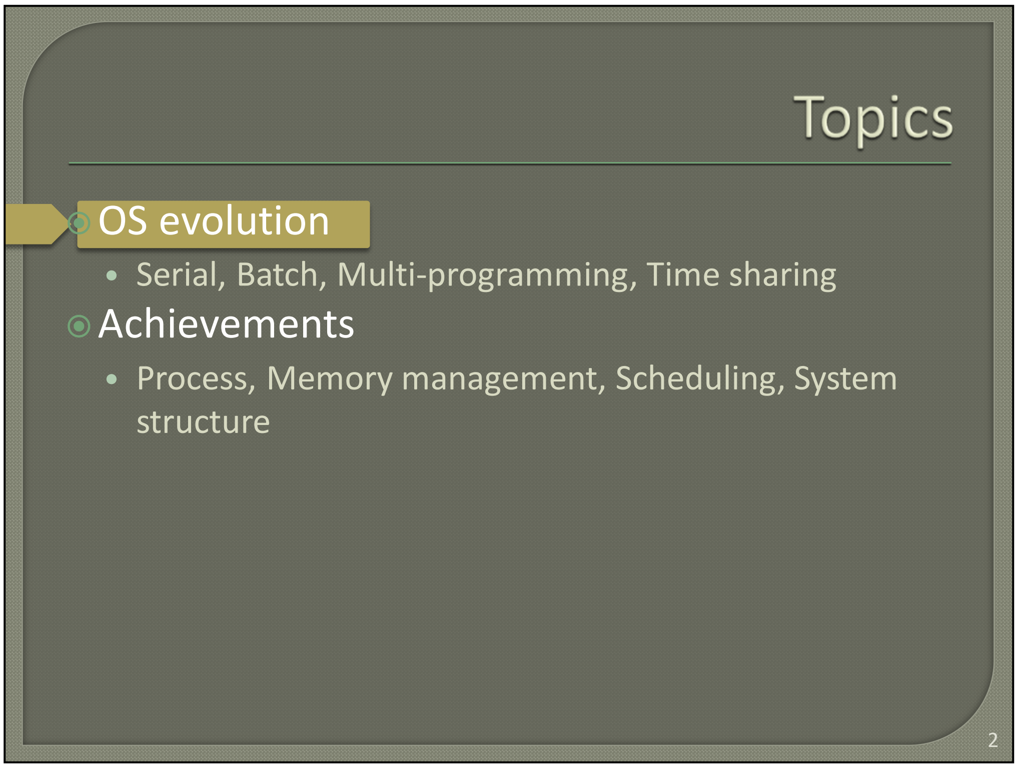
Department of Physics,
Computer Science & Engineering

CPSC 410 – Operating Systems I

Operating System Overview

Keith Perkins

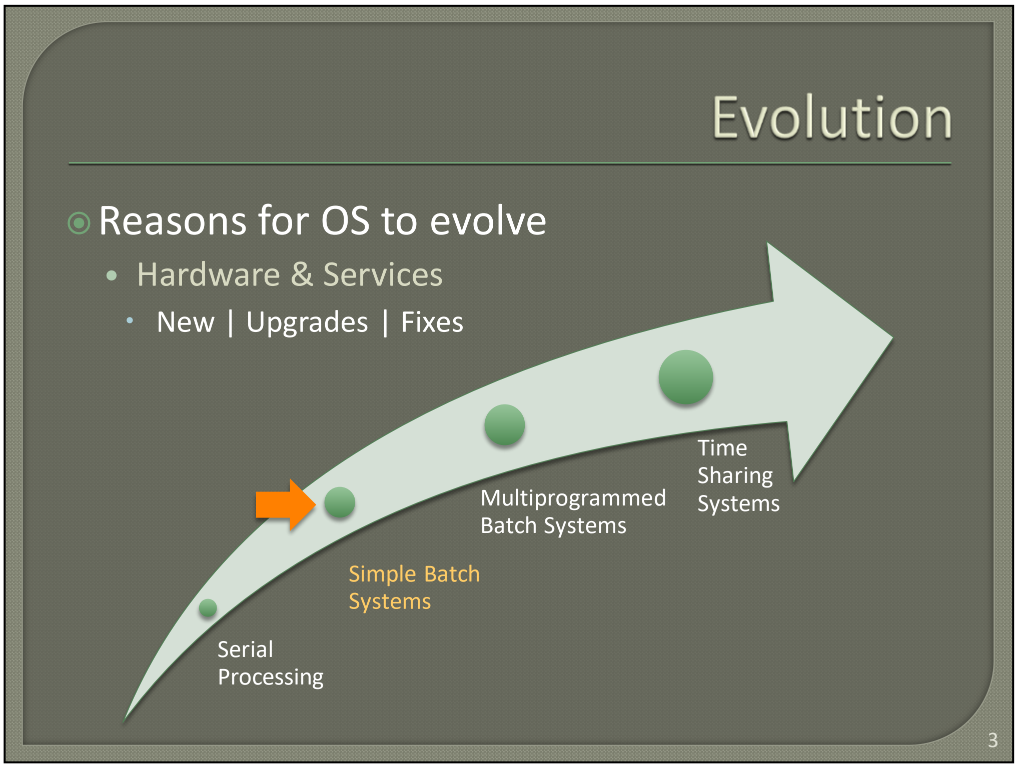
Original slides by Dr. Roberto A. Flores



Topics

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

2



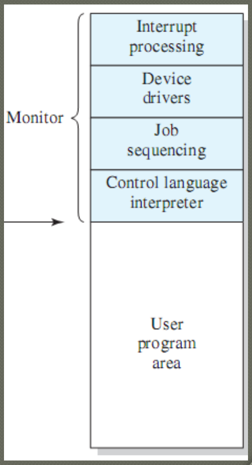
Evolution

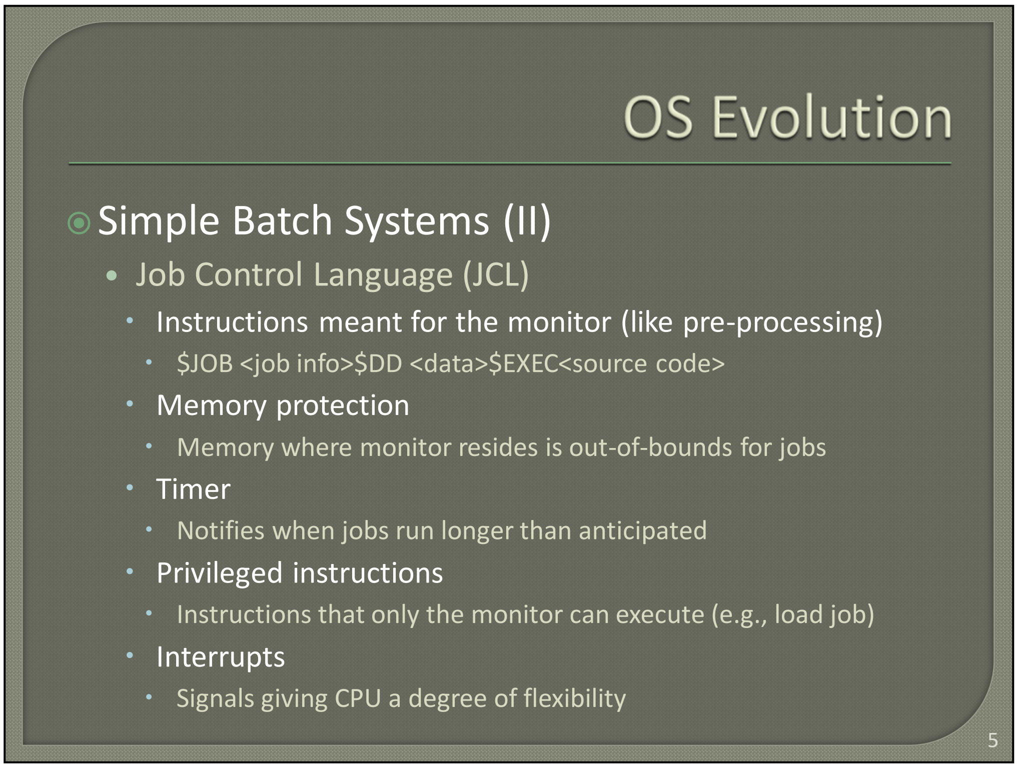
- Reasons for OS to evolve
  - Hardware & Services
    - New | Upgrades | Fixes

Serial Processing
Simple Batch Systems
Multiprogrammed Batch Systems
Time Sharing Systems

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

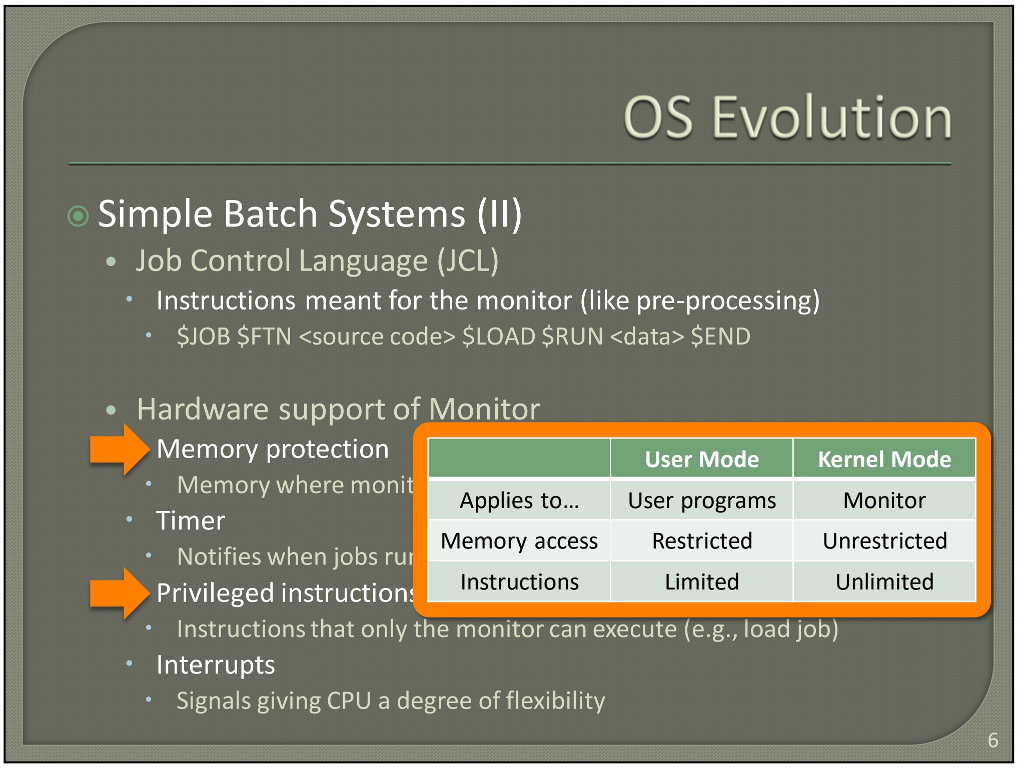




OS Evolution

- 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

5



OS Evolution

- Simple Batch Systems (II)
  - Job Control Language (JCL)
    - Instructions meant for the monitor (like pre-processing)
      - $JOB $FTN <source code> $LOAD $RUN <data> $END
  - Hardware support of Monitor
    - Memory protection
      - Memory where monit
    - Timer
      - Notifies when jobs ru
    - Privileged instructions
      - Instructions that only the monitor can execute (e.g., load job)
    - Interrupts
      - Signals giving CPU a degree of flexibility

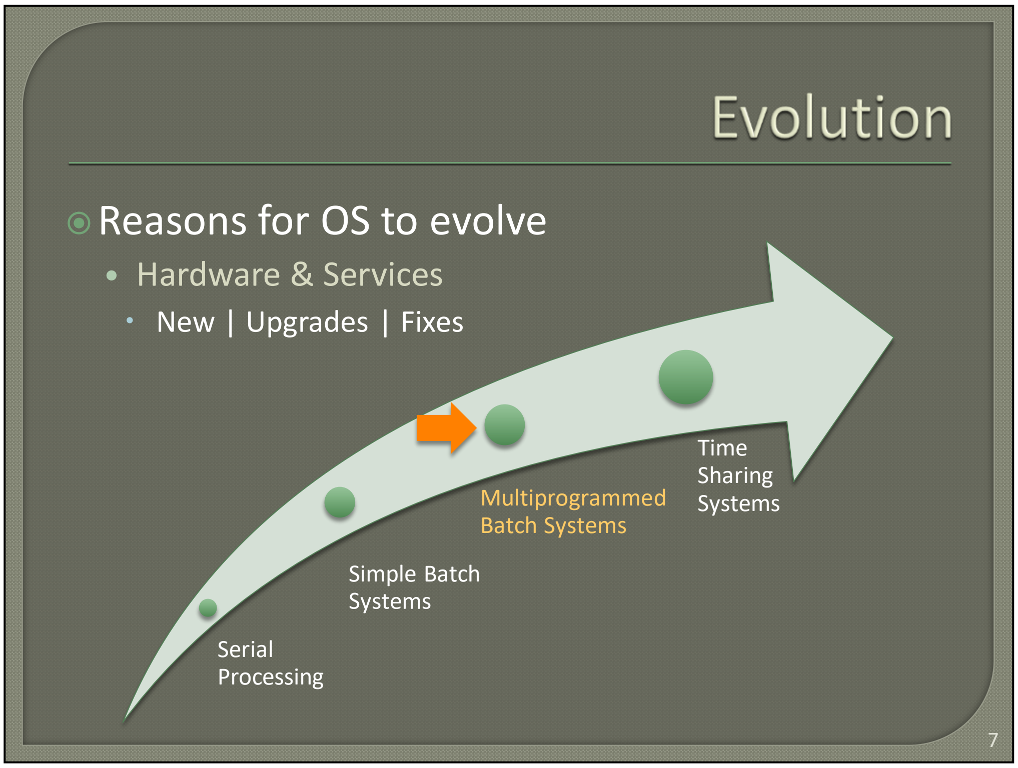
	User Mode	Kernel Mode
Applies to…	User programs	Monitor
Memory access	Restricted	Unrestricted
Instructions	Limited	Unlimited

6

User mode certain areas of memory are protected from users user (monitor mem)

Priv inst – some instructions are off limits to user

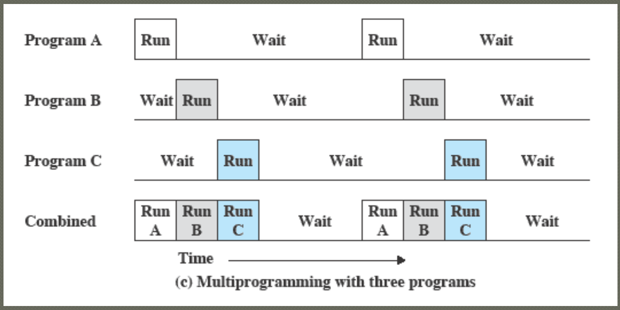
Monitor takes some mem and processor time



Evolution

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

Multiprogramming



(c) Multiprogramming with three programs

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

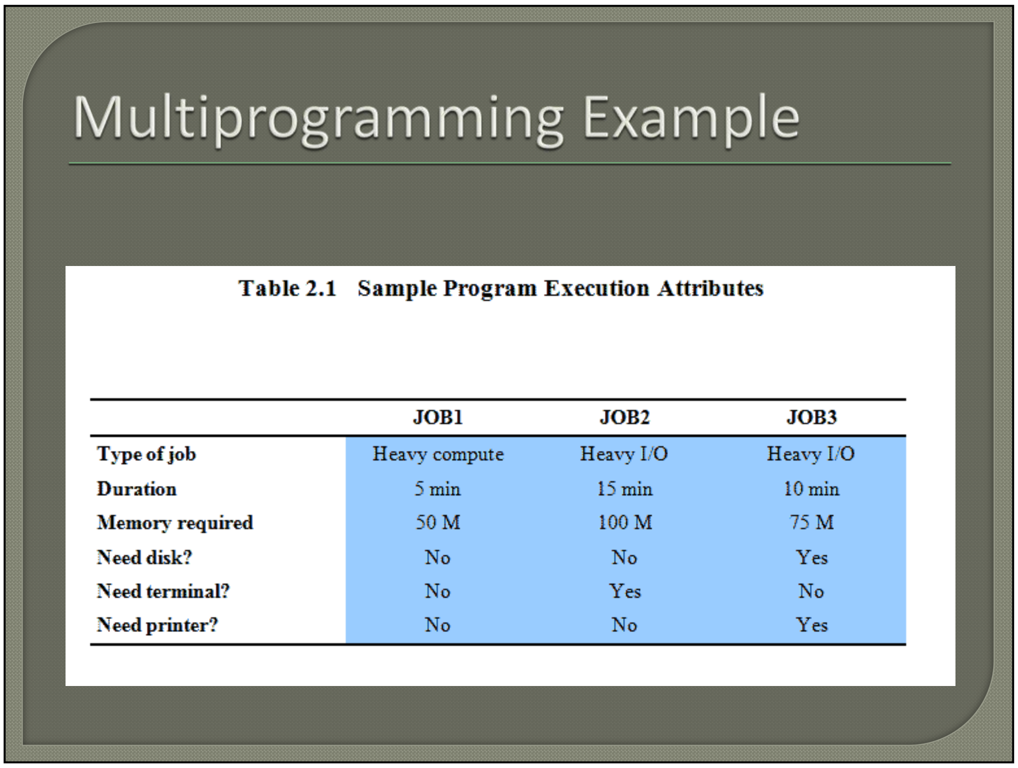
Furthermore, we might expand memory to hold three, four, or more

programs and switch among all of them (Figure 2.5c). The approach is known as

multiprogramming , or multitasking . It is the central theme of modern operating

systems.

Notice switch when waiting for I/O   or time slicing



Multiprogramming Example

Table 2.1 Sample Program Execution Attributes

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

To illustrate the benefit of multiprogramming, we give a simple example.

Consider a computer with 250 Mbytes of available memory (not used by the OS),

a disk, a terminal, and a printer. Three programs, JOB1, JOB2, and JOB3, are

submitted for execution at the same time, with the attributes listed in Table 2.1 .

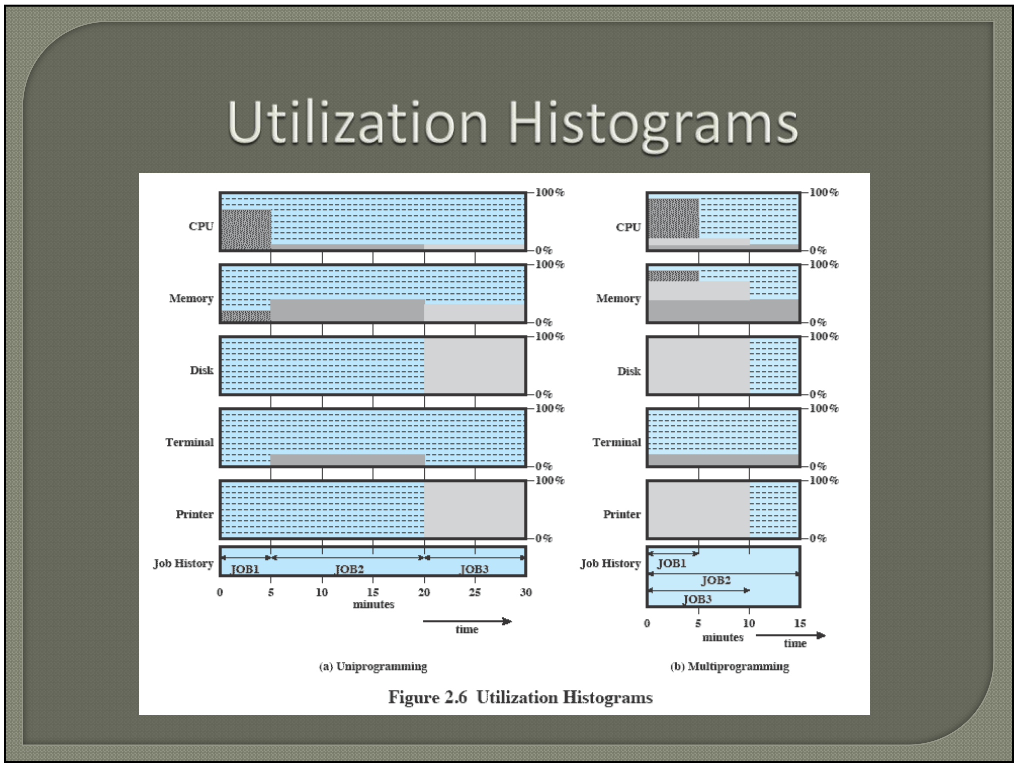
We assume minimal processor requirements for JOB2 and JOB3 and continuous

disk and printer use by JOB3. For a simple batch environment, these jobs will be

executed in sequence. Thus, JOB1 completes in 5 minutes. JOB2 must wait until

the 5 minutes are over and then completes 15 minutes after that. JOB3 begins after

20 minutes and completes at 30 minutes from the time it was initially submitted.



Utilization Histograms

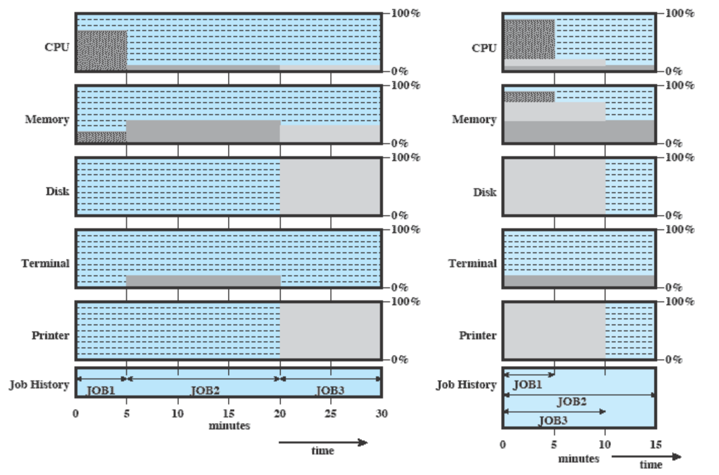


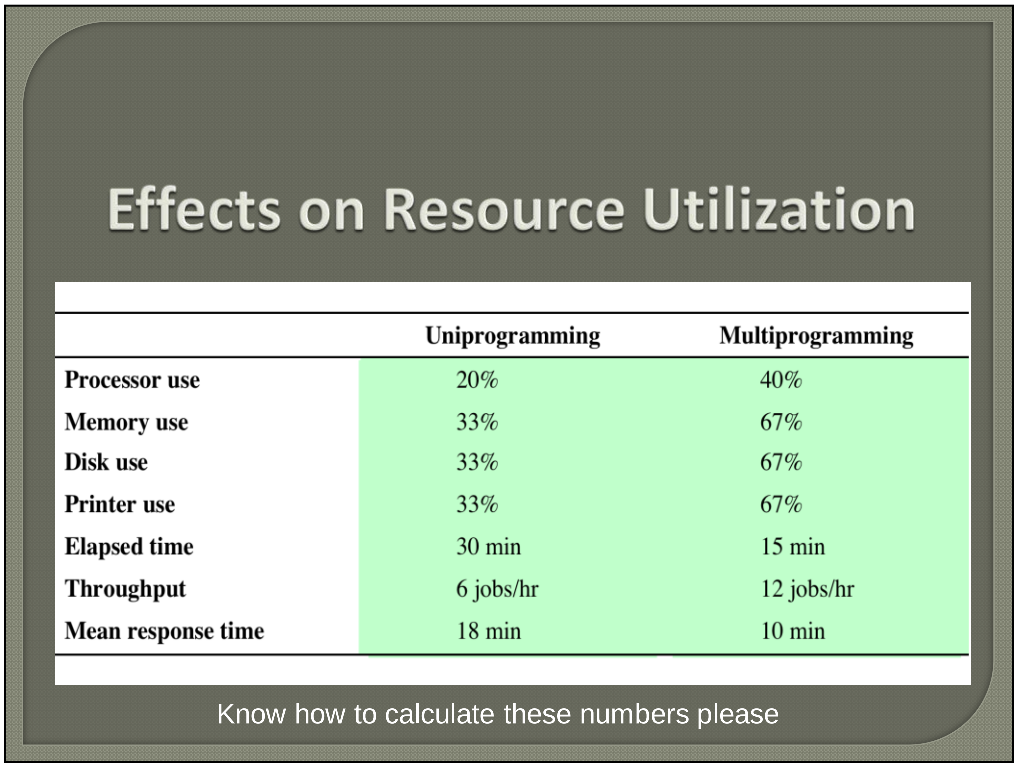
Figure 2.6 Utilization Histograms

Job1 uses 70%CPU, J2 and 3 10 %

Uni=(.7*5 + .1*25)/30 = 20%

Multi = (.9*5 + .2*5 +.1*5)/15 = 40%

Know how to calculate utilization!



Effects on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Know how to calculate these numbers please

The average resource utilization, throughput, and response times are shown in the

uniprogramming column of Table 2.2 .

Processor use:

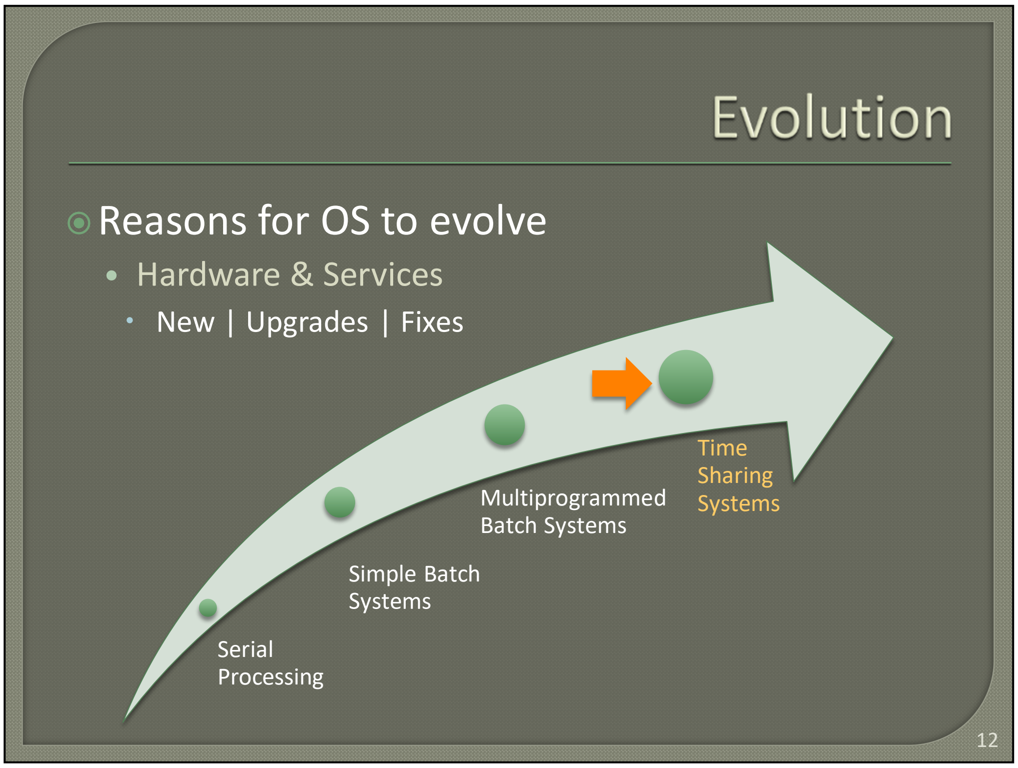
Uni=(.7*5 + .1*25)/30 = 20%

Multi = (.9*5 + .2*5 +.1*5)/15 = 40%

Mean response time – time between when a job is issued and when it completes

Uniprog ((0 +5) + (5 + 15) + (5+15 +10))/3jobs

Mult ((0+5) + (0+15) + (0 + 10))/3jobs



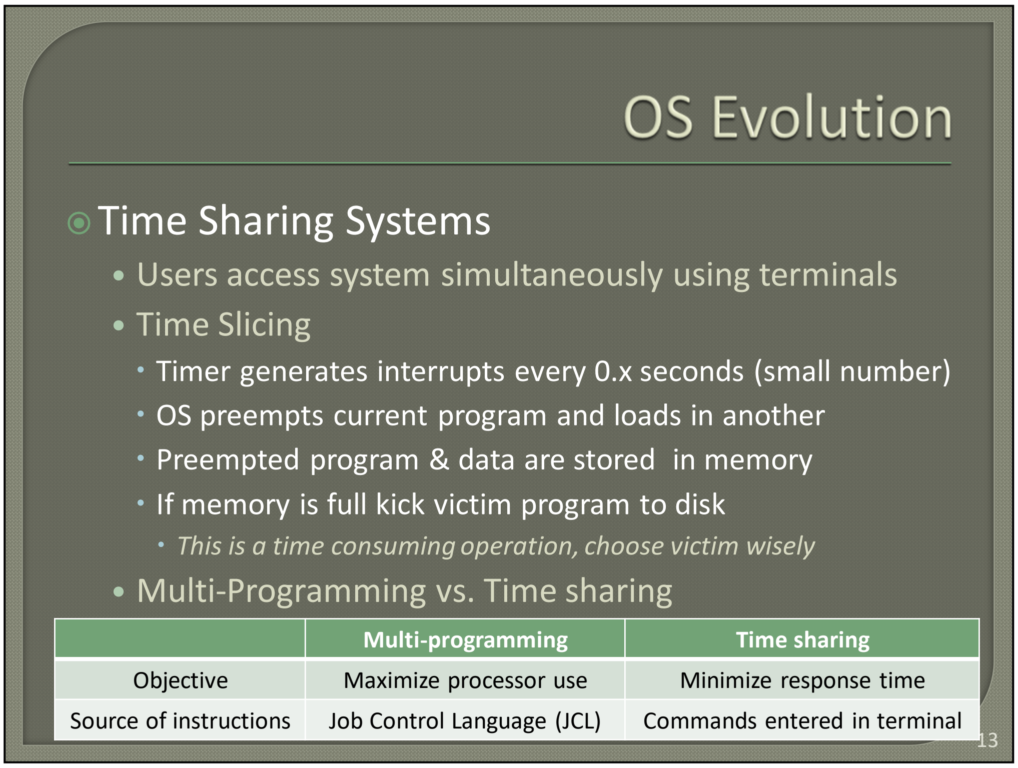
Evolution

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

Serial Processing
Simple Batch Systems
Multiprogrammed Batch Systems
Time Sharing Systems

12

9/11/18



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

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

13

Use the same mechanisms, interrupts, with different time slice lengths

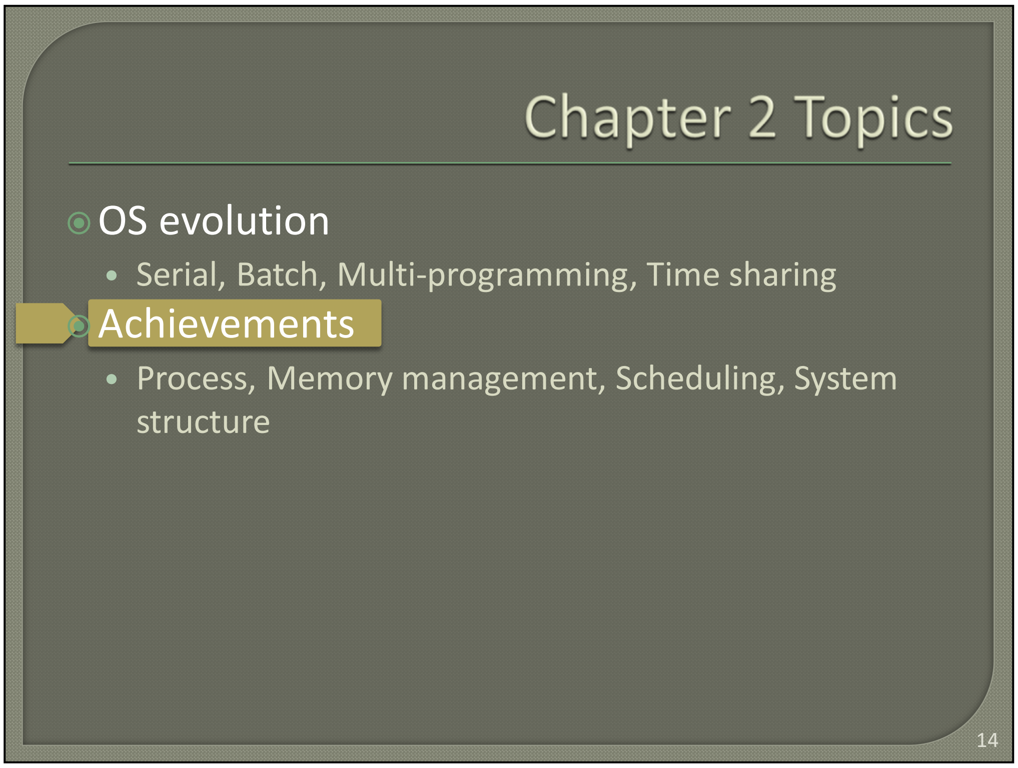
One is long enough to give process time to run

Other is much shorter to give impression of user servicing

Which would a webserver use?

Show multiprogramming on board with inconsistant time that each process runs (up to I/O need) so p1 may run for 20 p2 for 3

For time sharing, consistant switch time delta-t, 16ms or so

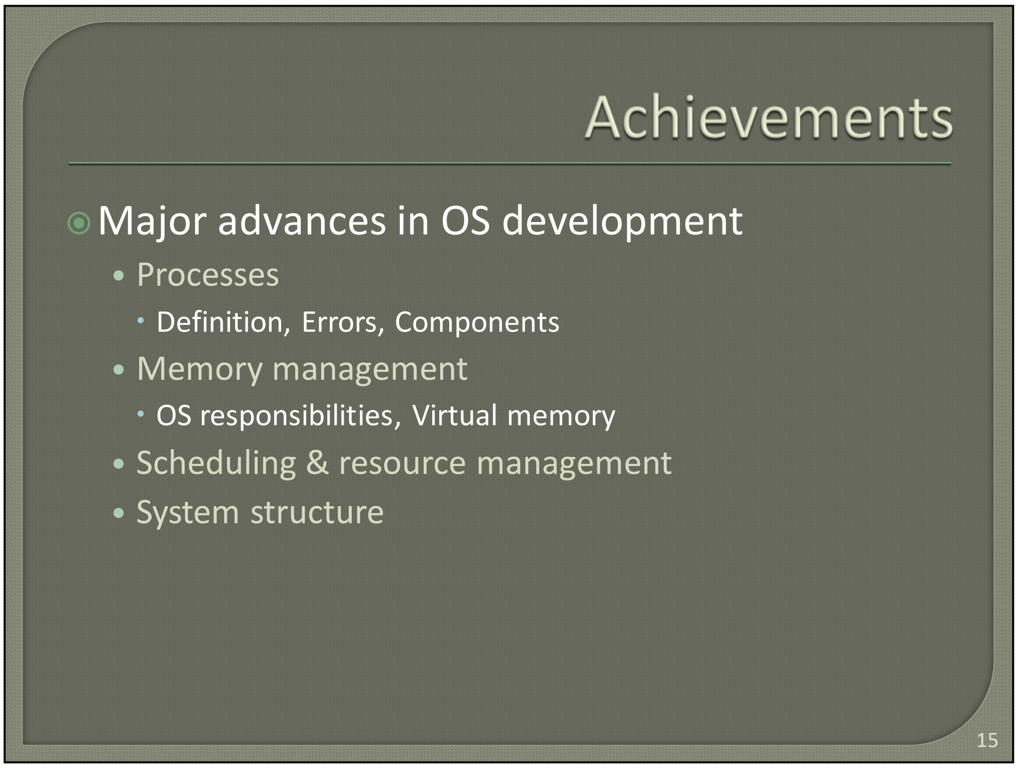


Chapter 2 Topics

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

14

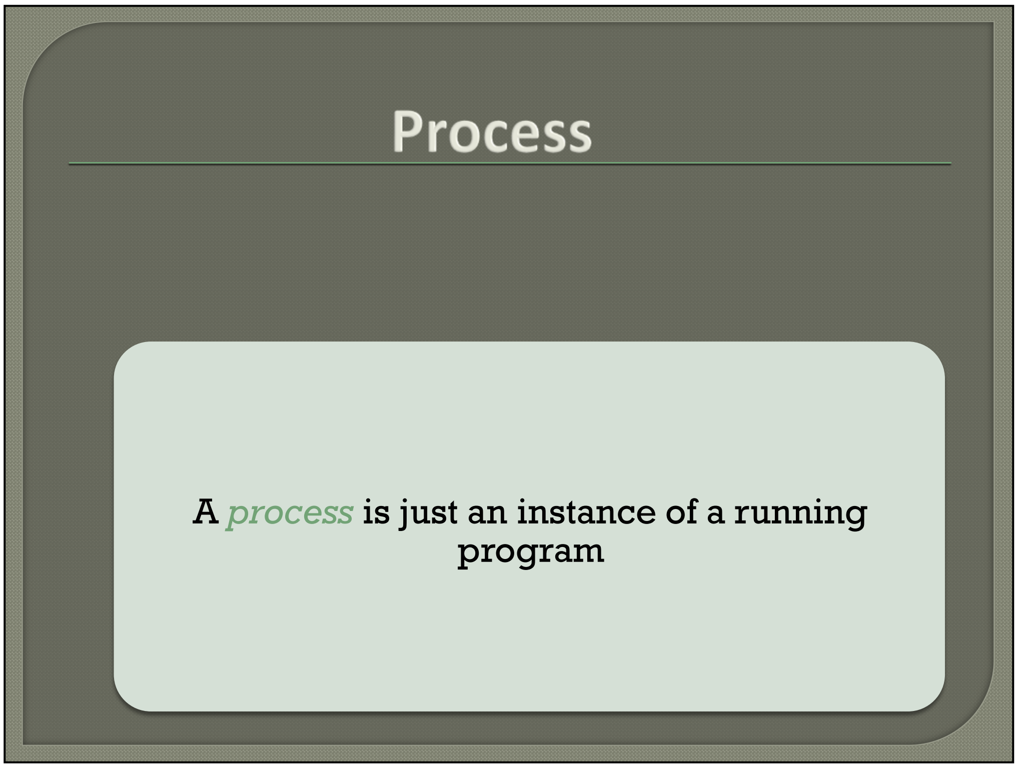
Stopped here 9/19/17



Achievements

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

15



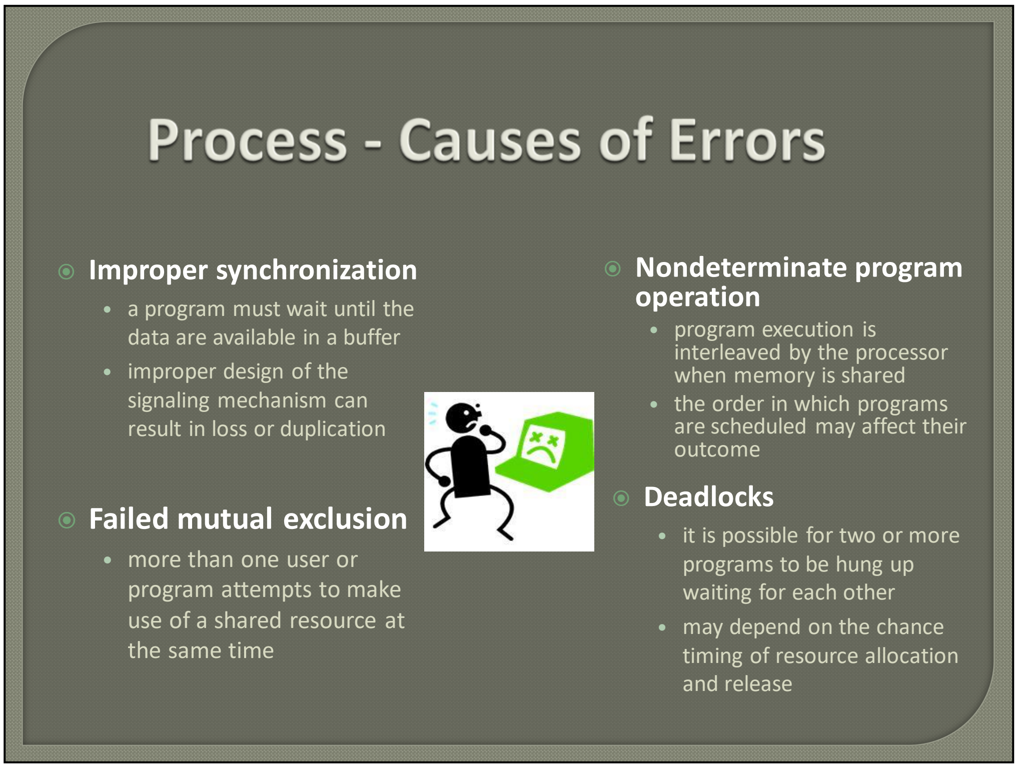
Process

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

Central to the design of operating systems is the concept of *process. This term was*

first used by the designers of Multics in the 1960s [DALE68]. It is a somewhat

more general term than *job.*



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

Efforts to design a system were vulnerable to subtle programming errors whose effects could be observed only when certain relatively rare sequences of actions occurred.

• These errors were difficult to diagnose because they needed to be distinguished from application software errors and hardware errors.

• Even when the error was detected, it was difficult to determine the cause, because the precise conditions under which the errors appeared were very hard to reproduce.

In general terms, there are four main causes of such errors:

Improper synchronization:

•Often a routine must be suspended awaiting an event elsewhere in the system.

• e.g. a program that initiates an I/O read must wait until the data are available in a buffer before proceeding.

• In such cases, a signal from some other routine is required.

•Improper design of the signalling mechanism can result in signals being lost or duplicate signals being received.

Failed mutual exclusion:

•e.g. two users may attempt to edit the same file at the same time.

• If these accesses are not controlled, an error can occur.

• There must be some sort of mutual exclusion mechanism that permits only one routine at a time to perform an update against the file.

Nondeterminate program operation:

• The results of a particular program normally should depend only on the input to that program and not on the activities of other programs in a shared system.

• But when programs share memory, and their execution is interleaved by the processor, they may interfere with each other by overwriting common memory areas in unpredictable ways.

• Thus, the order in which various programs are scheduled may affect the outcome of any particular program.

Deadlocks:

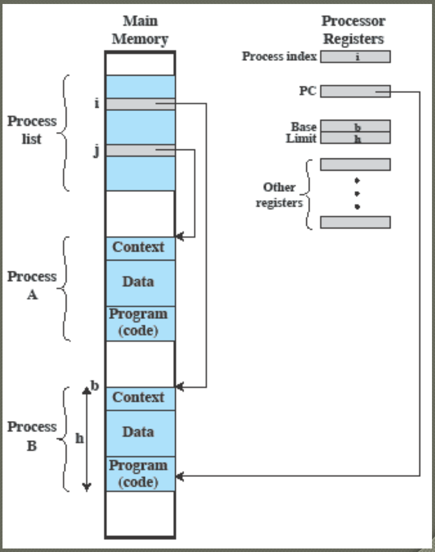
•Pa owns Lock1 and waits(Lock2)

•Pb owns Lock2 and waits(Lock1)

•Stopped here 1/30/18

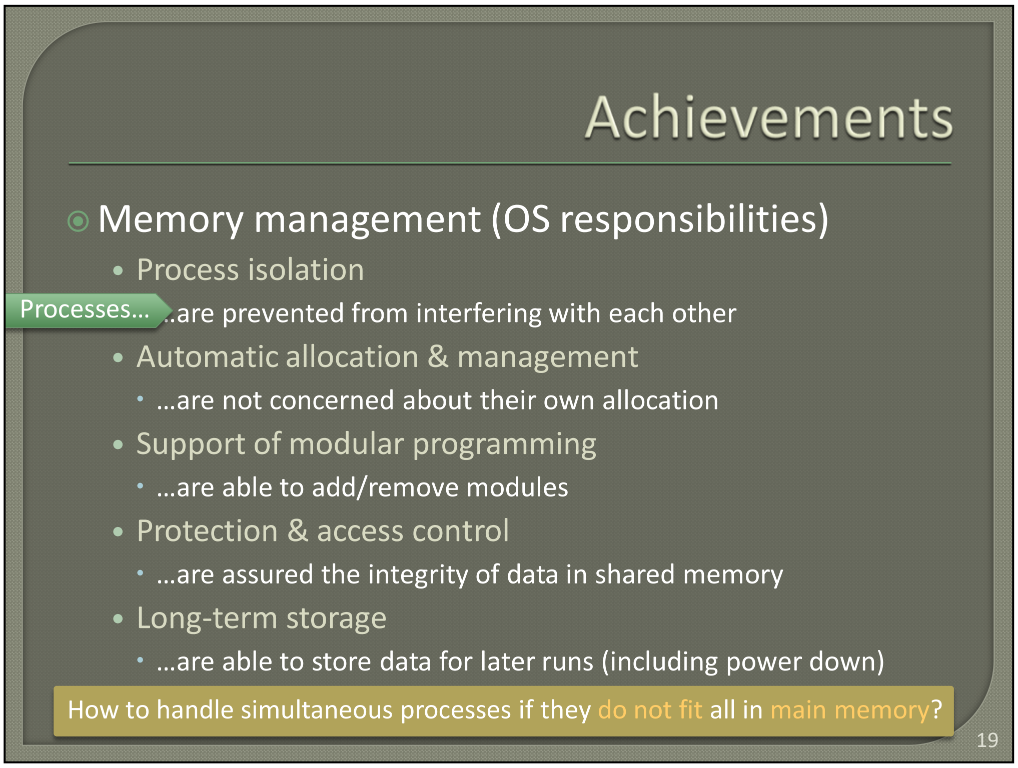
Process Management

- 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



18

Draw process layout in mem



Achievements

- Memory management (OS responsibilities)
  - Process isolation
    - Processes… …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?

19

The needs of users can be met best by a computing environment that supports modular programming and the flexible use of data. System managers need efficient and orderly control of storage allocation. The OS, to satisfy these requirements, has five principal storage management responsibilities:

• **Process isolation: The OS must prevent independent processes from interfering**
with each other's memory, both data and instructions.

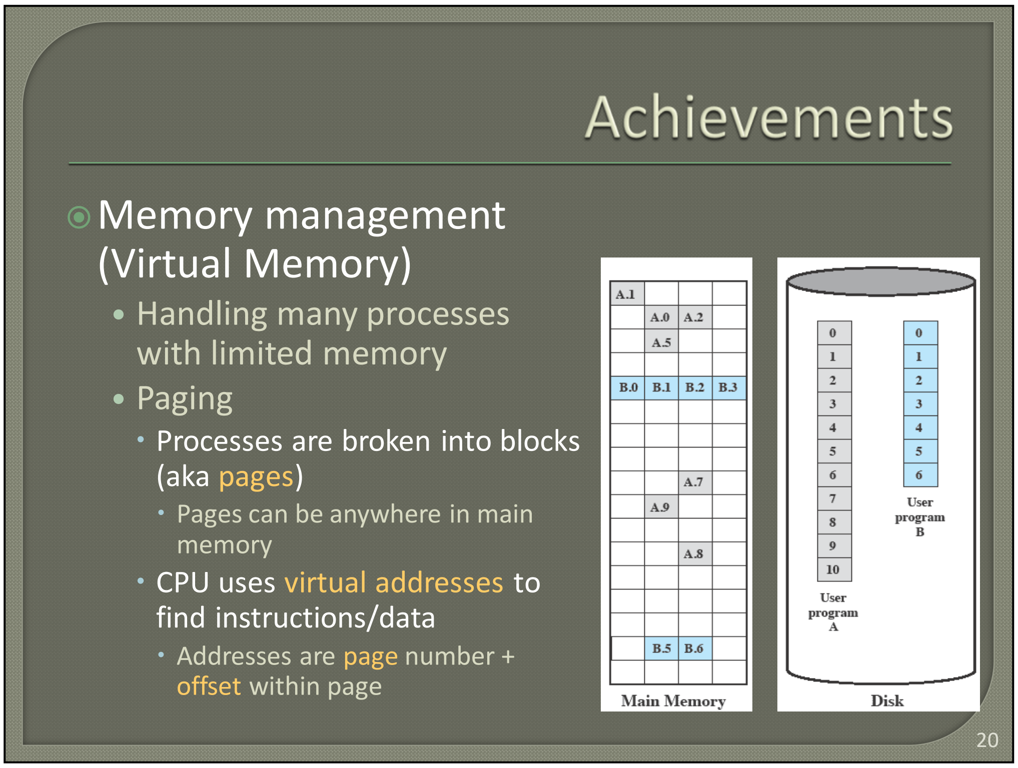
• **Automatic allocation and management: Programs should be dynamically**
allocated across the memory hierarchy as required. Allocation should be transparent to the programmer. Thus, the programmer is relieved of concerns relating to memory limitations, and the OS can achieve efficiency by assigning memory to jobs only as needed.

• **Support of modular programming: Programmers should be able to define program**
modules, and to create, destroy, and alter the size of modules dynamically.

• **Protection and access control: Sharing of memory, at any level of the memory**
hierarchy, creates the potential for one program to address the memory space of another. This is desirable when sharing is needed by particular applications.

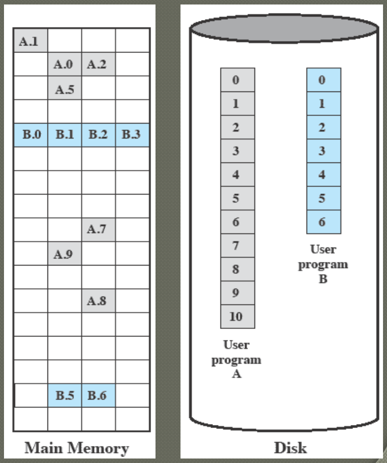
At other times, it threatens the integrity of programs and even of the OS itself. The OS must allow portions of memory to be accessible in various ways by various users.

• **Long-term storage: Many application programs require means for storing** information for extended periods of time, after the computer has been powered down.



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



20

Do simple example

2 progs a and b

Each start at loc 0

A is 63 bytes

B is 30 bytes

8 bits address, 3 bits VPN

2**3 = 8 blocks each 2**5 =32 bytes

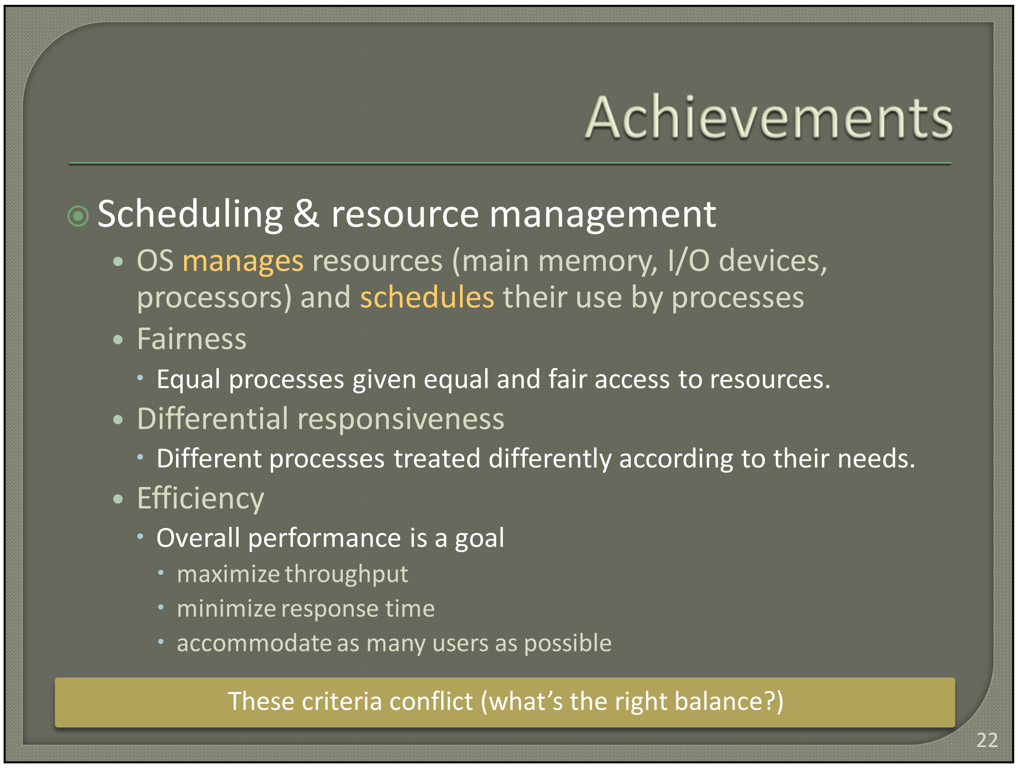
Draw mem on board

Demo how they each think they start at 0

How B is mapped somewhere by OS

How A is mapped somewhere by OS

Show instruction conversion Virtual to physical



Achievements

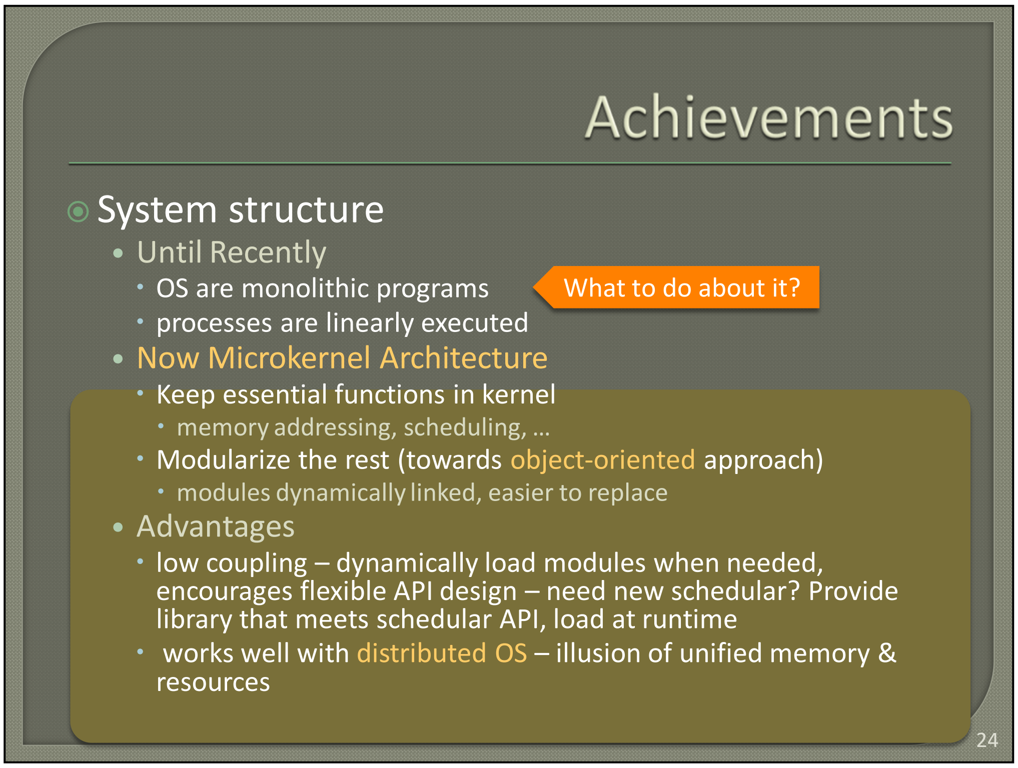
- Scheduling & resource management
  - OS manages resources (main memory, I/O devices, processors) and schedules their use by processes
  - Fairness
    - Equal processes given equal and fair access to resources.
  - Differential responsiveness
    - Different processes treated differently according 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?)

22

Scheduling dispatch

Schedular picks next job according to some criteria
Dispatch swaps it on to processor



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 schedular? Provide library that meets schedular API, load at runtime
    - works well with distributed OS – illusion of unified memory & resources

What to do about it?

24

Mem management,

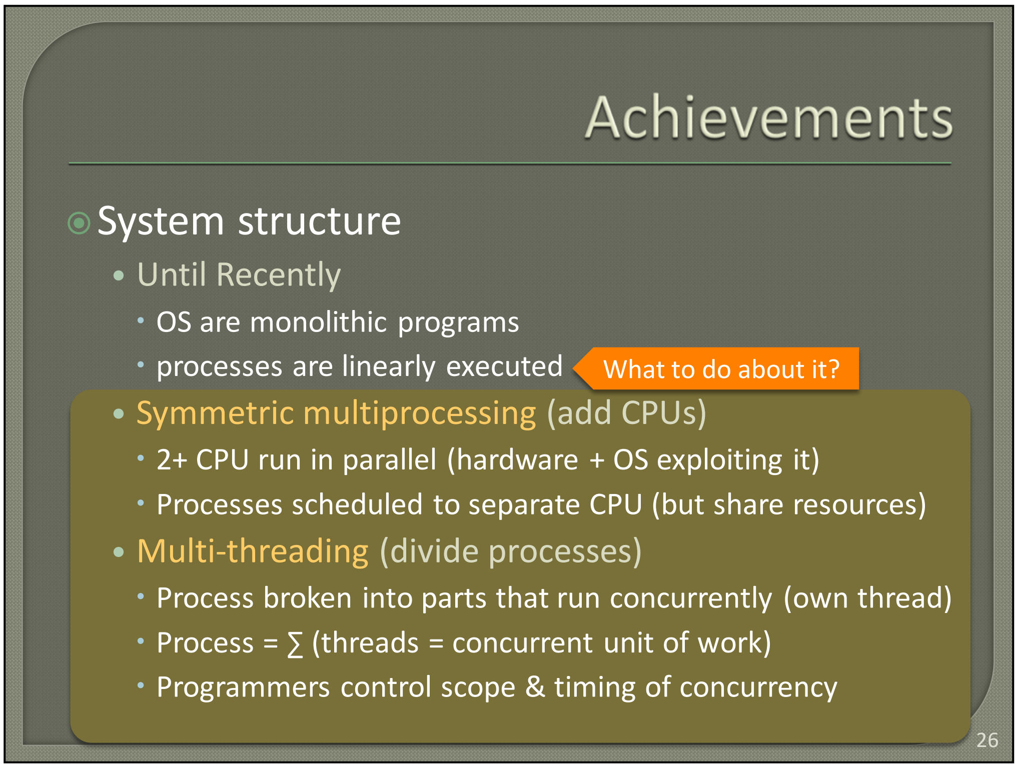
scheduling

Most operating systems, until recently, featured a large **monolithic kernel .** Most of what is thought of as OS functionality is provided in these large kernels, including scheduling, file system, networking, device drivers, memory management,

and more. Typically, a monolithic kernel is implemented as a single process, with all elements sharing the same address space. A **microkernel architecture assigns** only a few essential functions to the kernel, including address spaces, interprocess communication (IPC), and basic scheduling. Other OS services are provided by processes, sometimes called servers, that run in user mode and are treated like any other application by the microkernel. This approach decouples kernel and server development. Servers may be customized to specific application or environment requirements. The microkernel approach simplifies implementation, provides flexibility,

and is well suited to a distributed environment. In essence, a microkernel interacts with local and remote server processes in the same way, facilitating

construction
of distributed systems.



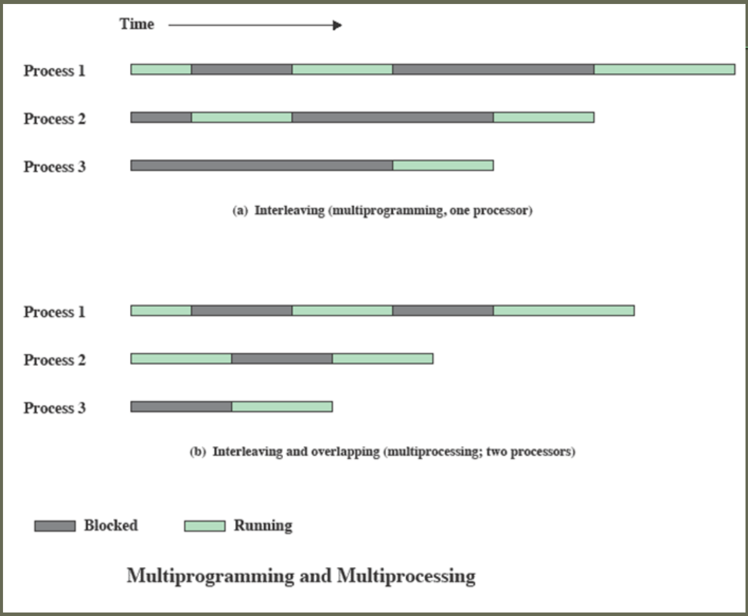
Achievements

- System structure
  - Until Recently
    - OS are monolithic programs
    - processes are linearly executed

What to do about it?

  - 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 = ∑ (threads = concurrent unit of work)
    - Programmers control scope & timing of concurrency

26


Time
Process 1
Process 2
Process 3
(a) Interleaving (multiprogramming, one processor)
Process 1
Process 2
Process 3
(b) Interleaving and overlapping (multiprocessing; two processors)
Blocked
Running
Multiprogramming and Multiprocessing

If the work to be done by a computer can be organized so that

some portions of the work can be done in parallel, then a system with multiple

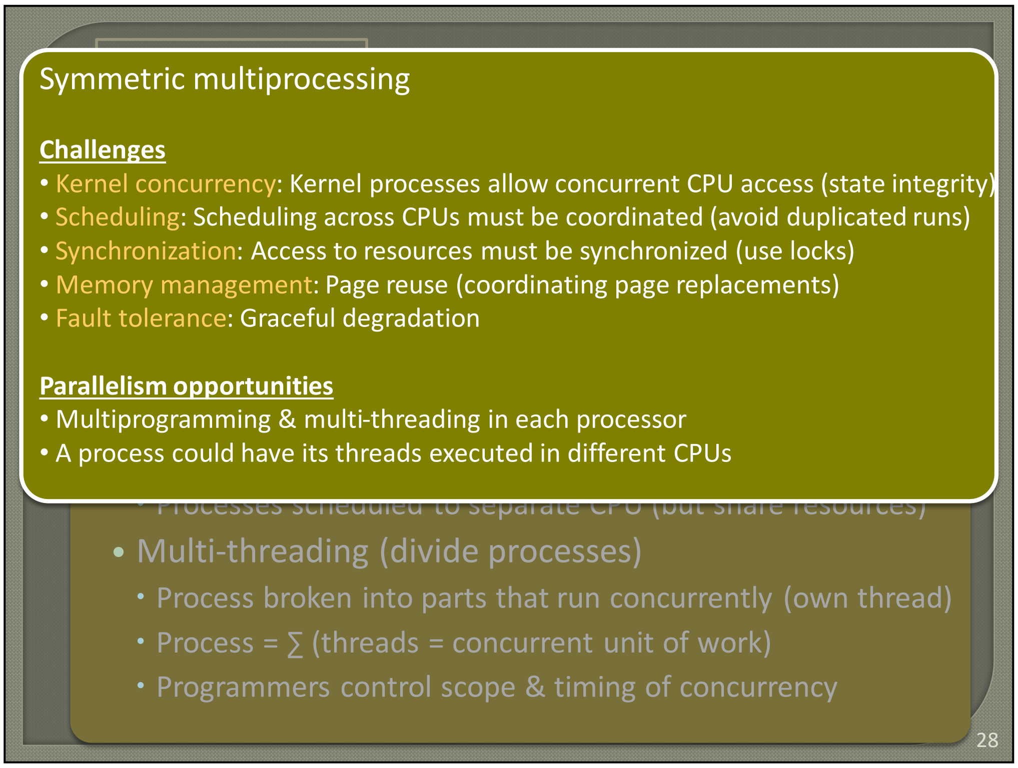
processors will yield greater performance than one with a single processor of

the same type. This is illustrated in Figure 2.12 . With multiprogramming, only

one process can execute at a time; meanwhile all other processes are waiting

for the processor. With multiprocessing, more than one process can be running

simultaneously, each on a different processor.



Symmetric multiprocessing

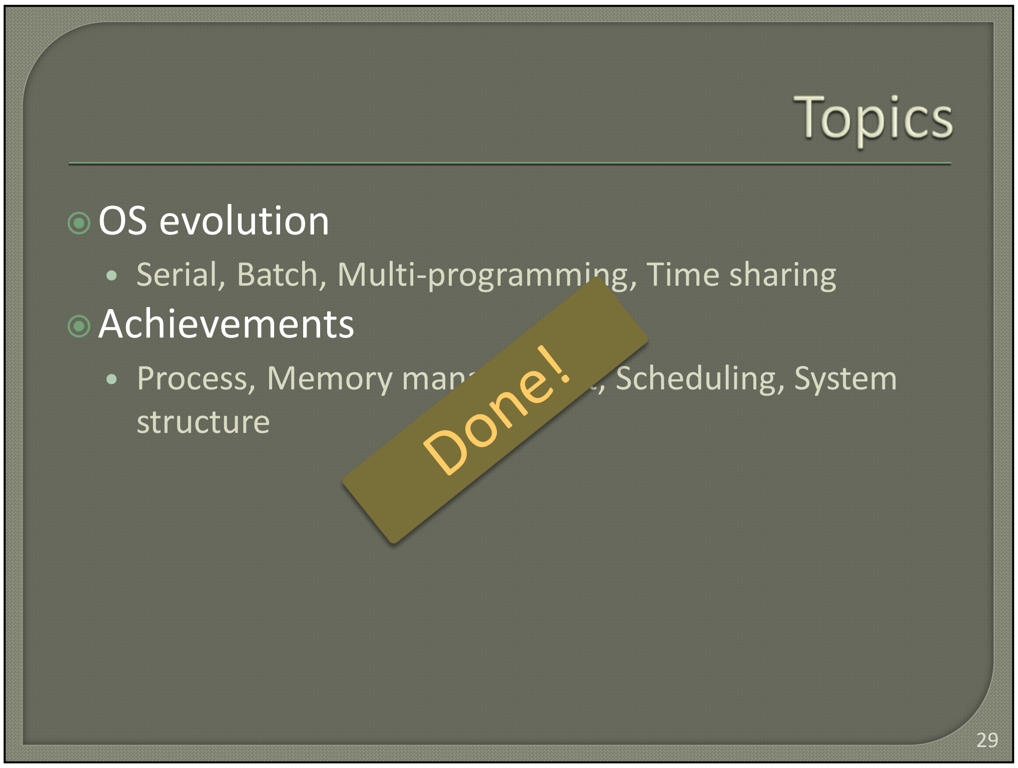
Challenges
- Kernel concurrency: Kernel processes allow concurrent CPU access (state integrity)
- Scheduling: Scheduling across CPUs must be coordinated (avoid duplicated runs)
- Synchronization: Access to resources must be synchronized (use locks)
- Memory management: Page reuse (coordinating page replacements)
- Fault tolerance: Graceful degradation

Parallelism opportunities
- Multiprogramming & multi-threading in each processor
- A process could have its threads executed in different CPUs

- Processes scheduled to separate CPU (but share resources)
- Multi-threading (divide processes)
  - Process broken into parts that run concurrently (own thread)
  - Process = ∑ (threads = concurrent unit of work)
  - Programmers control scope & timing of concurrency

28

Draw caches what if global here and here and back in memory? They snoop on global addressing, if global change in cache 1 then cache 2 knows and acts acordingly



Topics

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

29

Finish with a diagram on board of where OS resides in memory it

Loads, manages jobs

Allocates and manages memory

Shares disk space

Manages concurrency