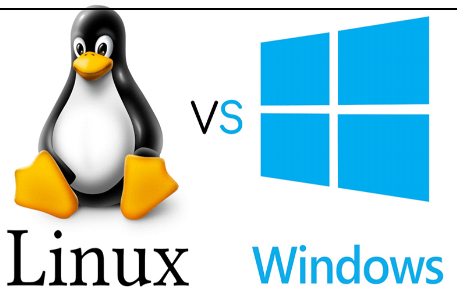
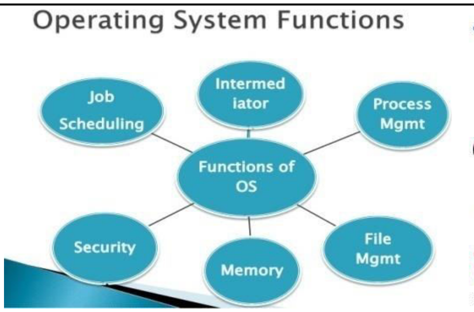
CIT210 OP. SYS. MANAG.

CSUN
COLLEGE OF
ENGINEERING AND
COMPUTER SCIENCE

Week7Day1 : Process Management, Process Attributes, Interrupt, Context Switch, Job Scheduler, Process Scheduler, Process State, Inter-process Communication





Objectives

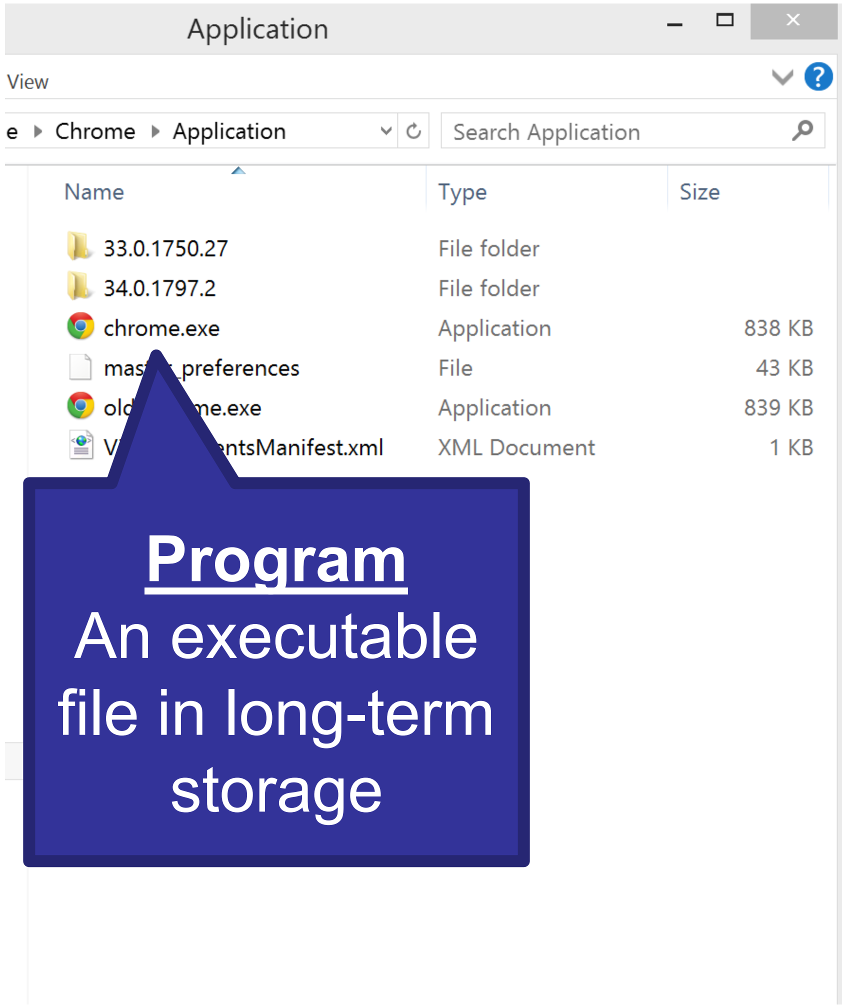
- Process Concept
- Process Scheduling
- Operations on Processes
- Inter-process Communication
- IPC in Shared-Memory Systems
- IPC in Message-Passing Systems
- Examples of IPC Systems
- Communication in Client-Server Systems

What is a process?

- A process is a program in execution
- Components:
  - Object program to be executed (called the *program text* in Unix)
  - The input data to be processed (obtained from a file, interactively or by other means) and the results it produces
  - Resources required by the program (i.e. files, etc..)
  - PCB – Process Control Block, containing the state of the process in execution.
- Clear distinction between a program and a process:
  - Program – static entity made up of source program language statements that define process behavior when executed on a set of data
  - Process – dynamic entity that executes a program on a particular set of data using resources allocated by the system; two or more processes could execute the same program, each using its own data and resources

PROF. HARJIT DHILLON

Programs and Processes



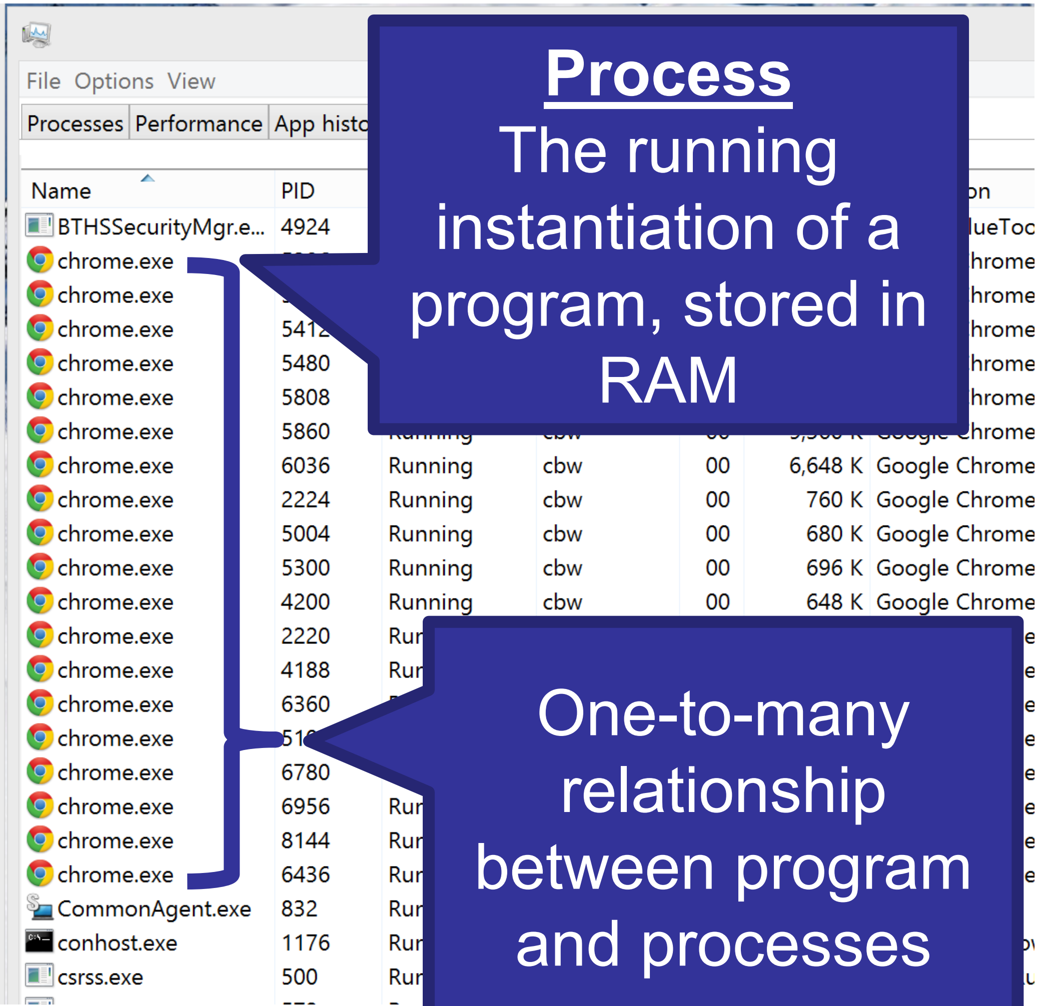
Application

View

e ▸ Chrome ▸ Application | Search Application

Name	Type	Size
33.0.1750.27	File folder	
34.0.1797.2	File folder	
chrome.exe	Application	838 KB
mas preferences	File	43 KB
old me.exe	Application	839 KB
V ntsManifest.xml	XML Document	1 KB

Program
An executable file in long-term storage



File Options View

Processes | Performance | App histo

Name	PID					on
BTHSSecurityMgr.e...	4924					lueToo
chrome.exe						hrome
chrome.exe						hrome
chrome.exe	541					hrome
chrome.exe	5480					hrome
chrome.exe	5808					hrome
chrome.exe	5860	Running	cbw	00	9,360 K	Google Chrome
chrome.exe	6036	Running	cbw	00	6,648 K	Google Chrome
chrome.exe	2224	Running	cbw	00	760 K	Google Chrome
chrome.exe	5004	Running	cbw	00	680 K	Google Chrome
chrome.exe	5300	Running	cbw	00	696 K	Google Chrome
chrome.exe	4200	Running	cbw	00	648 K	Google Chrome
chrome.exe	2220	Rur				e
chrome.exe	4188	Rur				e
chrome.exe	6360					e
chrome.exe	51					e
chrome.exe	6780					e
chrome.exe	6956	Rur				e
chrome.exe	8144	Rur				e
chrome.exe	6436	Rur				e
CommonAgent.exe	832	Rur				
conhost.exe	1176	Rur				o
csrss.exe	500	Rur				u

Process
The running instantiation of a program, stored in RAM

One-to-many relationship between program and processes

PROF. HARJIT DHILLON

CSUN COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Process Concept

- An operating system executes a variety of programs that run as a process.
- Process – a program in execution; process execution progress in sequential fashion
- Multiple parts
  - The program code, also called text section
  - Current activity including program counter, processor registers
  - Stack containing temporary data
    - Function parameters, return addresses, local variables
  - Data section containing global variables
  - Heap containing memory dynamically allocated during run time

Process and Program Concept

- Process
  - is a program in execution.
  - is an active entity.
  - forms the basis of all computation.
  - process execution must progress in sequential fashion.
- Program
  - is a passive entity stored on disk (executable file),
  - becomes a process when executable file is loaded into memory.
- Execution of program is started via CLI entry of its name, GUI mouse clicks, etc.
- A process is an instance of a running program; it can be assigned to, and executed on, a processor.
- Related terms for Process: Job, Step, Load Module, Task, Thread.

PROF. HARJIT DHILLON

When is a process created?

- Processes can be created in two ways
  - System initialization: one or more processes created when the OS starts up
  - Execution of a process creation system call: something explicitly asks for a new process
- System calls can come from
  - User request to create a new process (system call executed from user shell)
  - Already running processes
    - User programs
    - System daemons

Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - `fork()` system call creates new process
  - `exec()` system call used after a `fork()` to replace the process' memory space with a new program

PROF. HARJIT DHILLON

When do processes end?

- Conditions that terminate processes can be
  - Voluntary
  - Involuntary
- Voluntary
  - Normal exit
  - Error exit
- Involuntary
  - Fatal error (only sort of involuntary)
  - Killed by another process

PROF. HARJIT DHILLON

Process Termination

- Process executes last statement and then asks the operating system to delete it using the `exit()` system call.
  - Returns status data from child to parent (via `wait()`)
  - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the `abort()` system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

PROF. HARJIT DHILLON

Process Termination

- Some operating systems do not allow a child to exist if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - cascading termination. All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the `wait()` system call. The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If no parent waiting (did not invoke `wait()`) process is a zombie
- If parent terminated without invoking `wait`, process is an orphan

Process hierarchies

- Parent creates a child process
  - Child processes can create their own children
- Forms a hierarchy
  - UNIX calls this a “process group”
  - If a process exits, its children are “inherited” by the exiting process’s parent
- Windows has no concept of process hierarchy
  - All processes are created equal

PROF. HARJIT DHILLON

Process Parts

- A process includes three segments/sections:
  1. Program: code/text.
  2. Data: global variables and heap
     - Heap contains memory dynamically allocated during run time.
  3. Stack: temporary data
     - Procedure/Function parameters, return addresses, local variables.
- Current activity of a program includes its Context: program counter, state, processor registers, etc.
- One program can be several processes:
  - Multiple users executing the same Sequential program.
  - Concurrent program running several process.

PROF. HARJIT DHILLON

Process Elements

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data
- I/O status information
- Accounting information

PROF. HARJIT DHILLON

Process Context

- The (constantly changing) state of a running program at any point in time.
- Process context includes the following parts:
  - The scheduling context is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process.
  - The kernel maintains accounting information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far.
  - The file table is an array of pointers to kernel file structures.
  - When making file I/O system calls, processes refer to files by their index into this table.

PROF. HARJIT DHILLON

Process Context (Cont.)

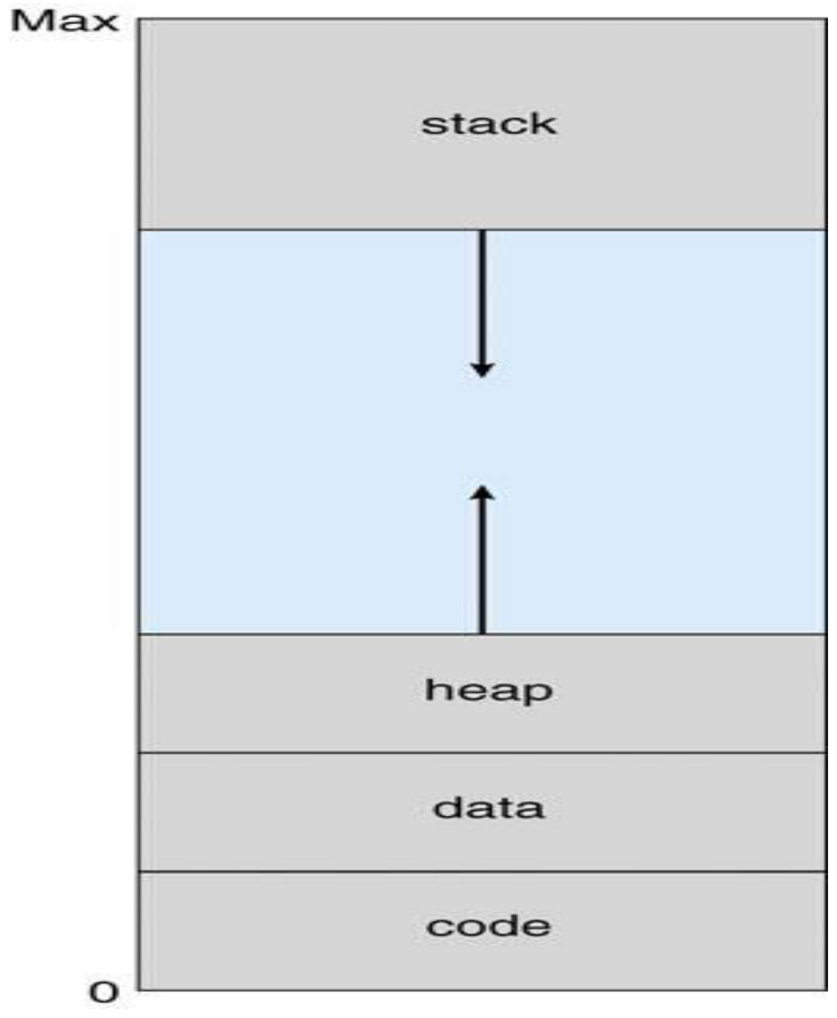
- Whereas the file table lists the existing open files, the file-system context applies to requests to open new files.
  - The current root and default directories to be used for new file searches are stored here.
- The signal-handler table defines the routine in the process's address space to be called when specific signals arrive.
- The virtual-memory context of a process describes the full contents of the its private address space.

Main OS Process-related Goals

- Main OS Process-related Goals
  - Interleave the execution of existing processes to maximize <u>processor utilization</u>
  - Provide reasonable <u>response times</u>
  - Allocate resources to processes
  - Support inter-process communication (and synchronization) and user creation of processes
- How are these goals achieved?
  - *Schedule* and *dispatch* processes for execution by the processor
  - Implement a safe and fair policy for *resource allocation* to processes
  - Respond to requests by user programs
  - *Construct* and *maintain tables* for each process managed by the operating system

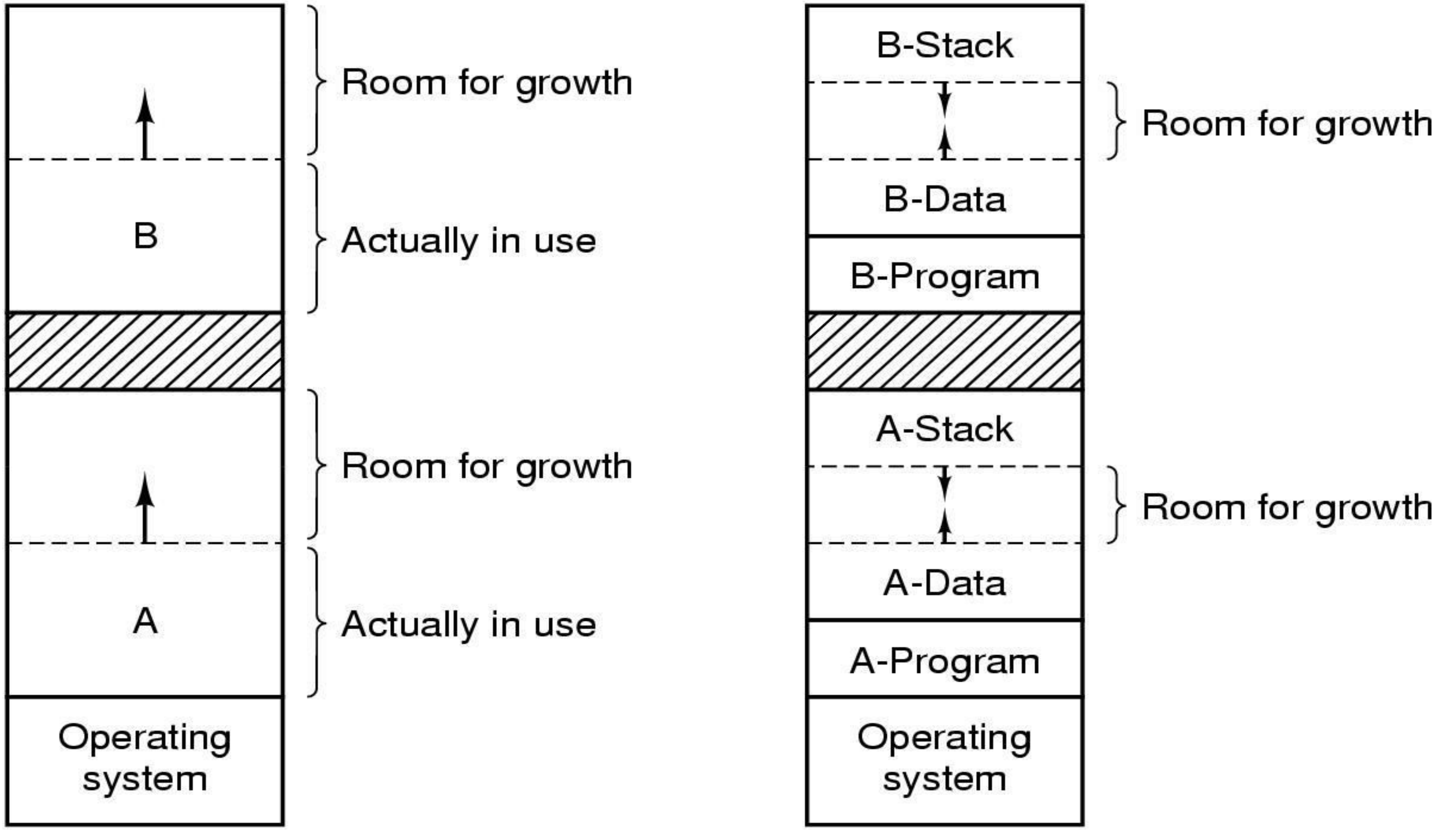
PROF. HARJIT DHILLON

Process in Memory (1)



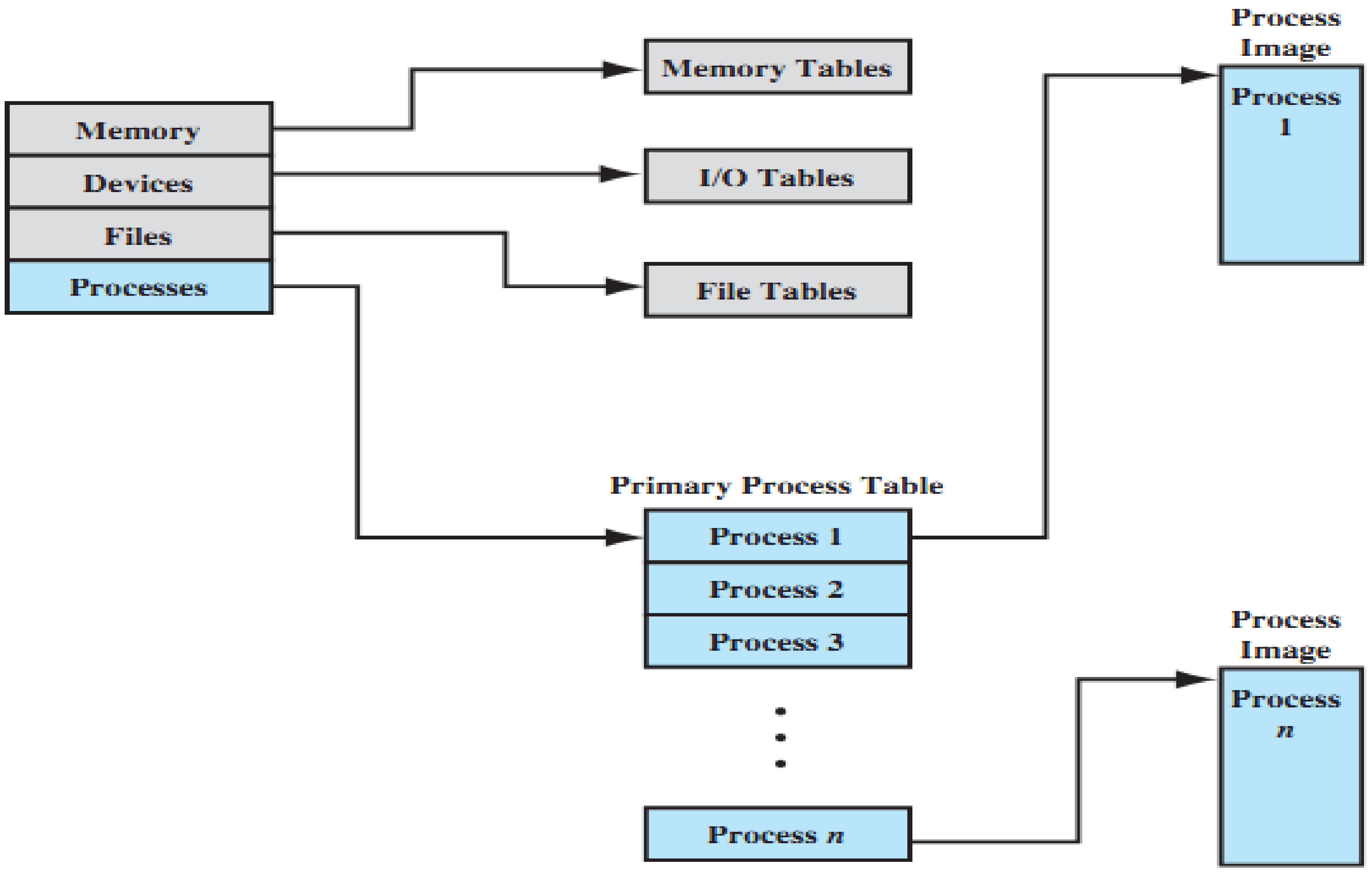
CSUN COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Processes in Memory



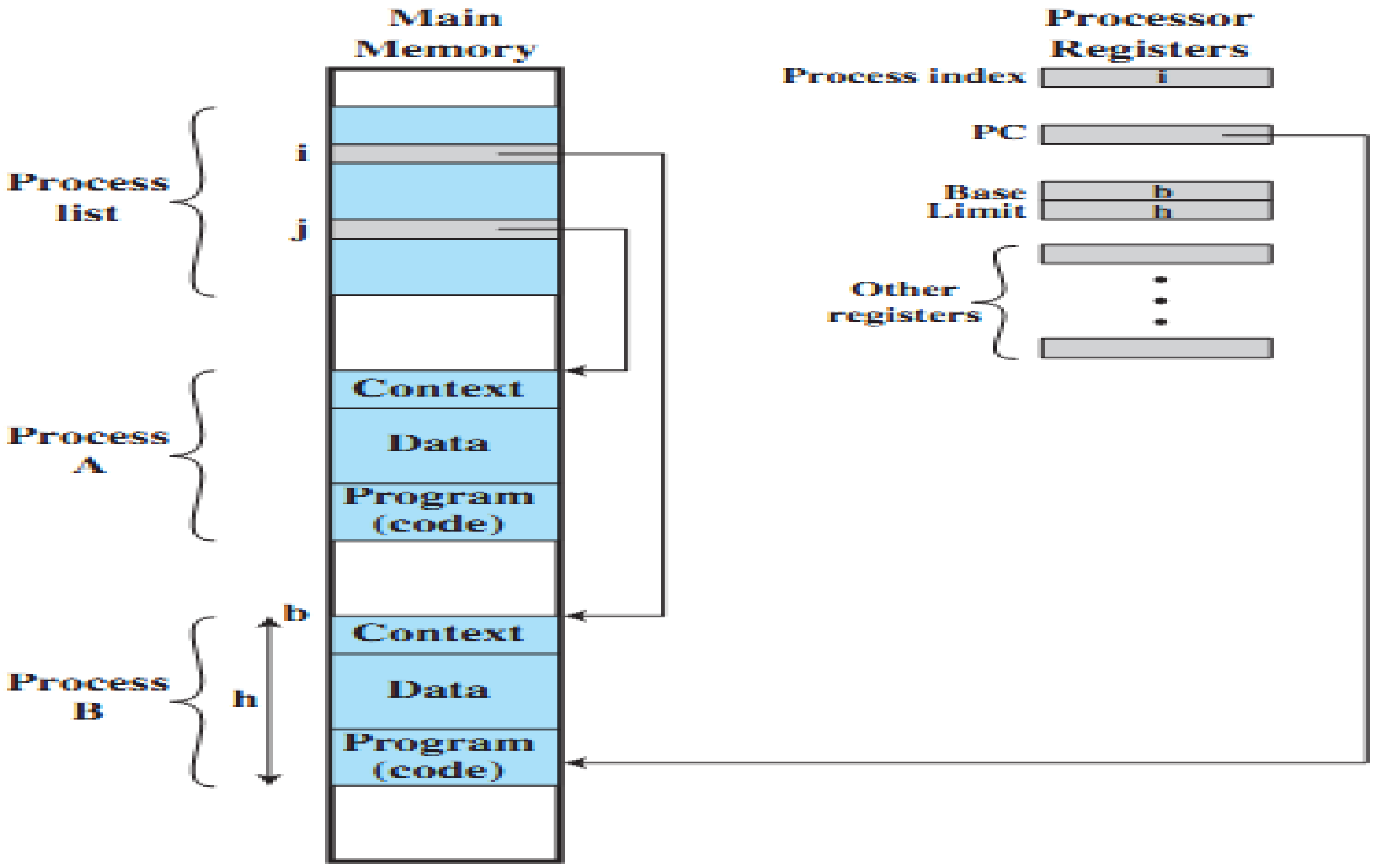
(a) (b)

General Structure of OS Control Tables



PROF. HARJIT DHILLON

Typical Process Table Implementation



PROF. HARJIT DHILLON

CSUN COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

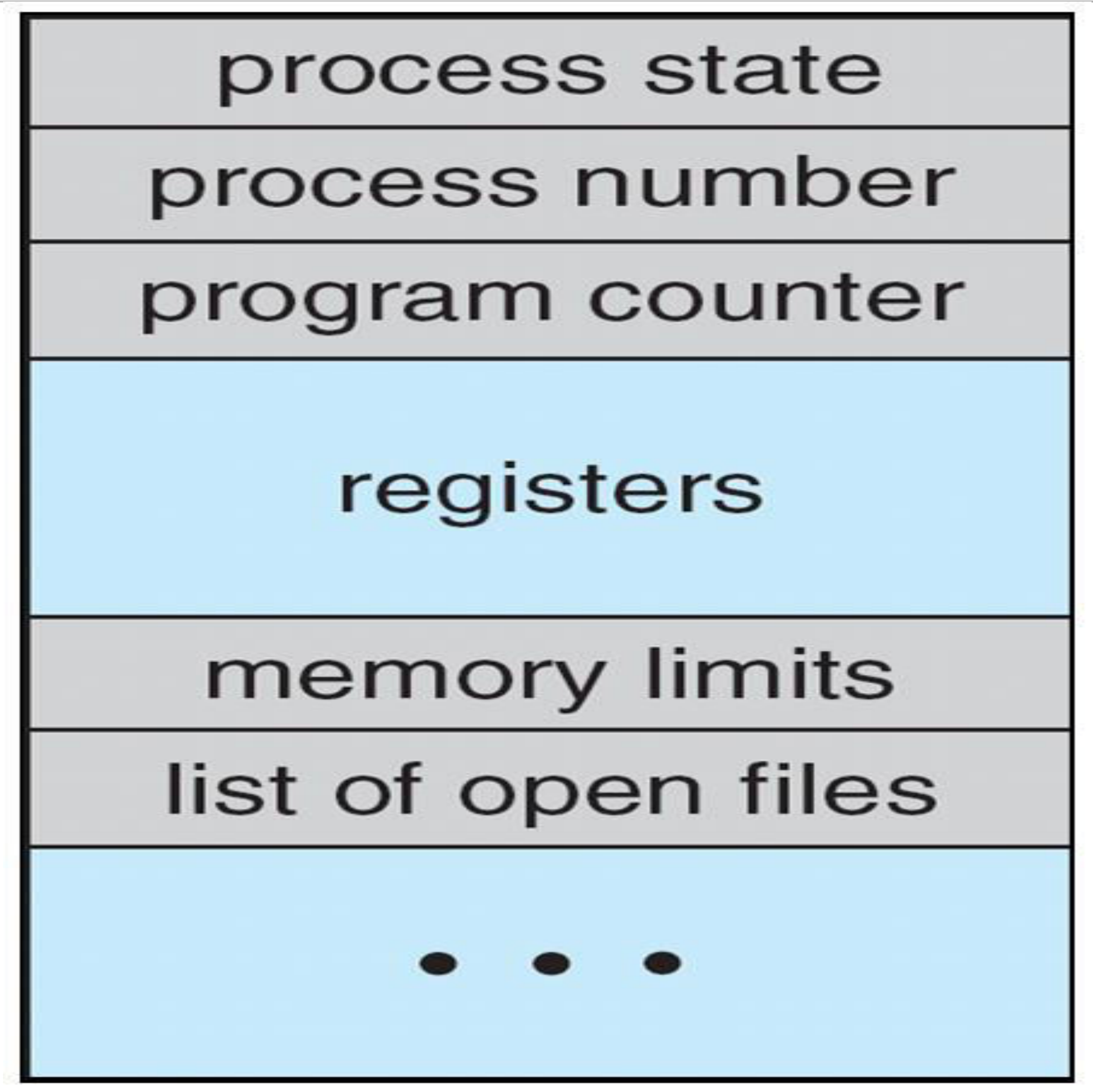
Program execution

- **Each execution of the program generates a process that is executed**
  - **Non reentrant**
    - The data and code (text) address space of each process (user) are different
  - **Reentrant**
    - The data address space are unique for each process
    - The code (text) is the same for all the processes
- **Inter-process relationships:**
  - Competition – processes are trying to get access to different resources of the system, therefore a protection between processes is necessary
  - Cooperation – sometime the processes need to communicate between themselves and exchange information – synchronization is needed

Process Attributes

- Process ID
- Parent process ID
- User ID
- Process state/priority
- Program counter
- CPU registers
- Memory management information
- I/O status information
- Access Control
- Accounting information

Typical process table entry



CSUN COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Fields of a typical process table entry

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

CSUN® COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Components of Process Control Block (PCB)

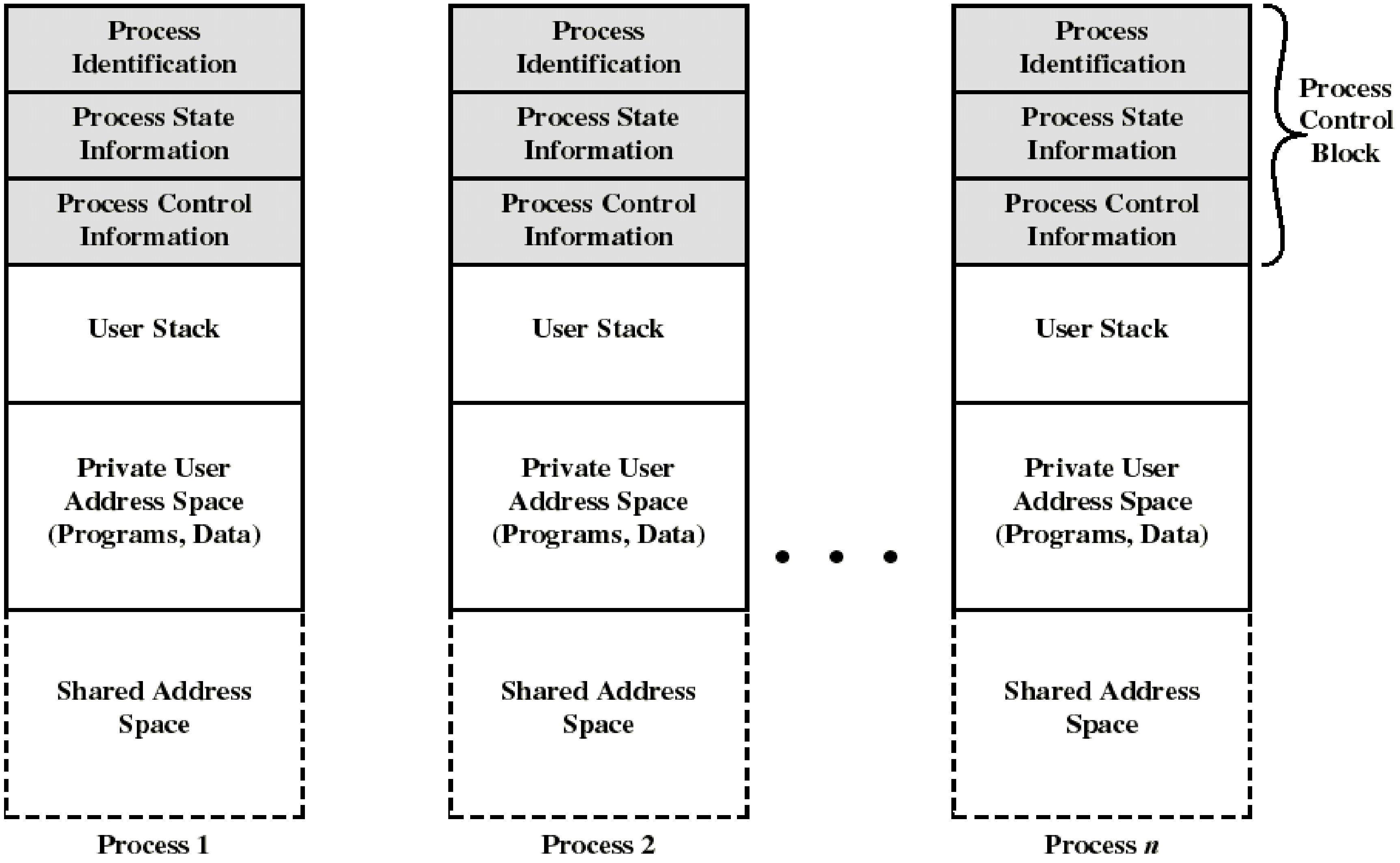
- **Process Control Block (PCB) – Information associated with each process – its context!**
- **PCB (execution context) is the data needed (process attributes) by OS to control process:**
  1. Process location information
  2. Process identification information
  3. Processor state information
  4. Process control information

PROF. HARJIT DHILLON

Process Location Information

- Each process image in memory:
  - may not occupy a contiguous range of addresses (depends on memory management scheme used).
  - both a private and shared memory address space can be used.
- The location if each process image is pointed to by an entry in the process table.
- For the OS to manage the process, at least part of its image must be brought into main memory.

Process Images in Memory



PROF. HARJIT DHILLON

Process Identification Information

- **A few numeric identifiers may be used:**
  - **Unique process identifier (PID) –**
    - indexes (directly or indirectly) into the process table.
  - **User identifier (UID) –**
    - the user who is responsible for the job.
  - **Identifier of the process that created this process (PPID).**
- **Maybe symbolic names that are related to numeric identifiers.**

Processor State Information

- Contents of processor registers:
  - User-visible registers
  - Control and status registers
  - Stack pointers
- Program Status Word (PSW)
  - contains status information
  - Example: the EFLAGS register on Pentium machines.

Process Control Information

- scheduling and state information:
  - Process state (i.e., running, ready, blocked...)
  - Priority of the process
  - Event for which the process is waiting (if blocked).
- data structuring information
  - may hold pointers to other PCBs for process queues, parent-child relationships and other structures.

Process Control Information

- Inter-process communication –
  - may hold flags and signals for IPC.
- Resource ownership and utilization –
  - resource in use: open files, I/O devices...
  - history of usage (of CPU time, I/O...).
- Process privileges (Access control) –
  - access to certain memory locations, to resources, etc…
- Memory management –
  - pointers to segment/page tables assigned to this process.

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

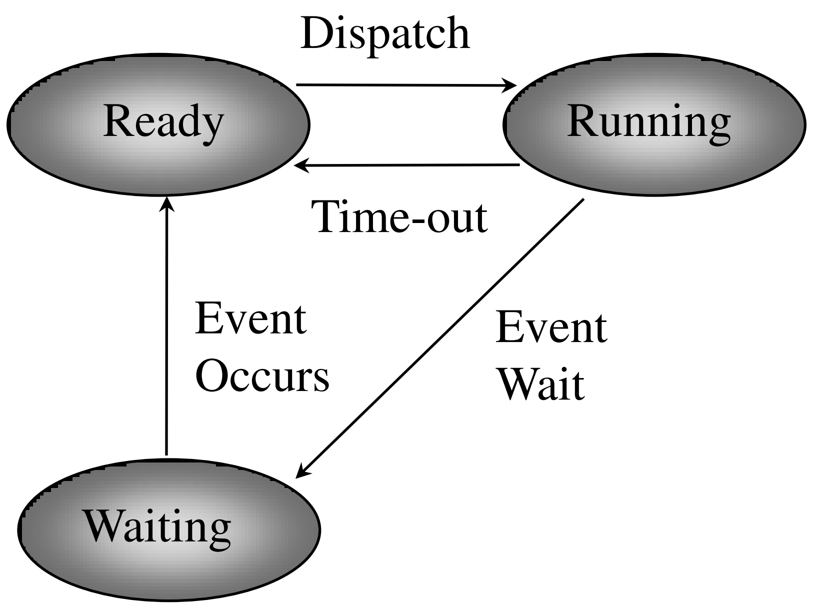
PROF. HARJIT DHILLON

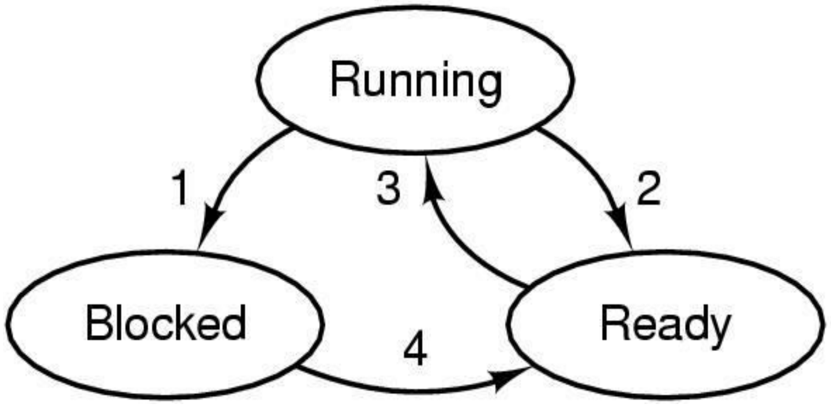
Process States

- **Let us start with three states:**

1) **Running state –**
   - the process that gets executed (single CPU); its instructions are being executed.
2) **Ready state –**
   - any process that is ready to be executed; the process is waiting to be assigned to a processor.
3) **Waiting/Blocked state –**
   - when a process cannot execute until its I/O completes or some other event occurs.

A Three-state Process Model





1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Process Transitions (1)

- Ready –> Running
  - When it is time, the dispatcher selects a new process to run.
- Running –> Ready
  - the running process has expired his time slot.
  - the running process gets interrupted because a higher priority process is in the ready state.

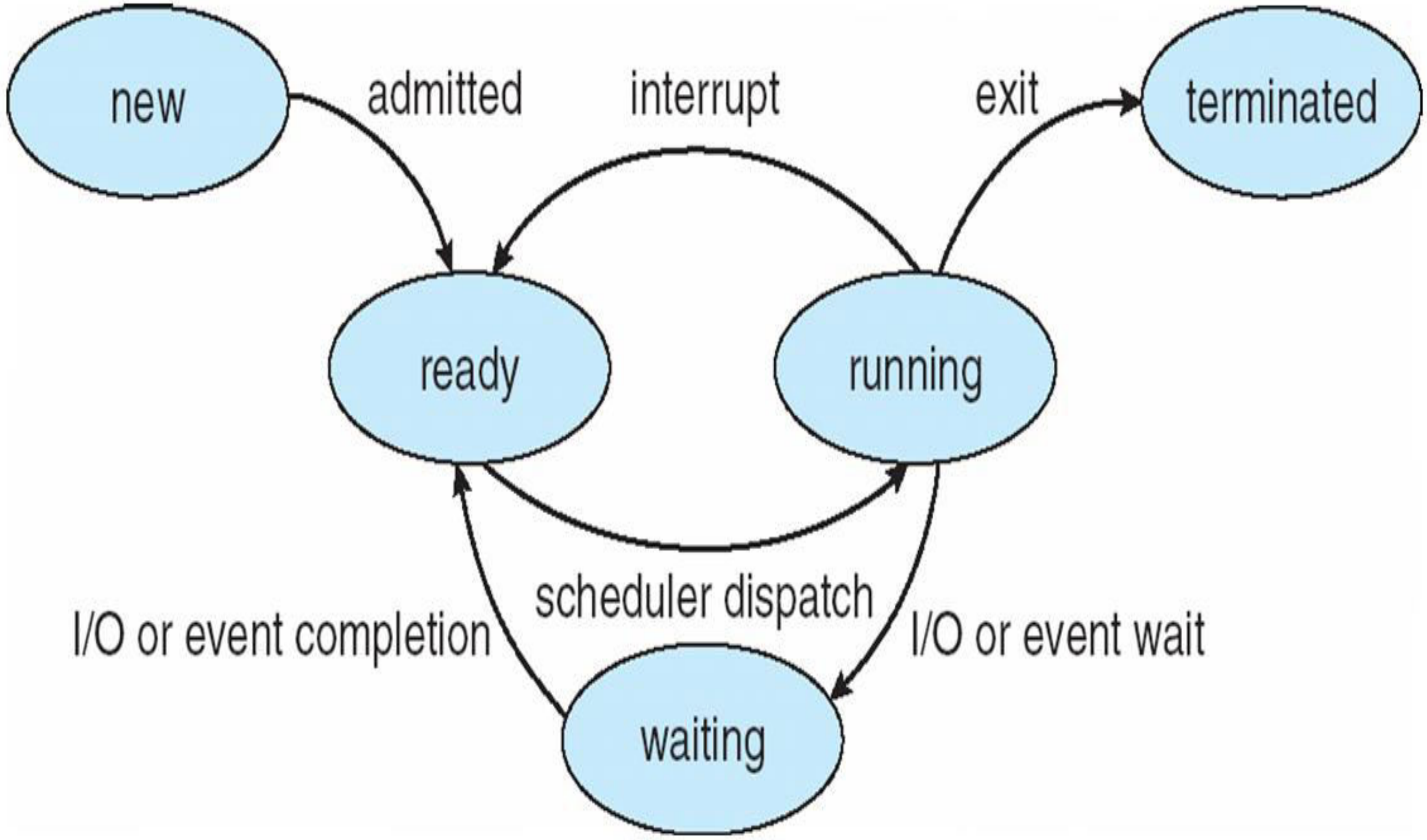
PROF. HARJIT DHILLON

Process Transitions (2)

- **Running –> Waiting**
  - **When a process requests something for which it must wait:**
    - a service that the OS is not ready to perform.
    - an access to a resource not yet available.
    - initiates I/O and must wait for the result.
    - waiting for a process to provide input.
- **Waiting –> Ready**
  - **When the event for which it was waiting occurs.**

PROF. HARJIT DHILLON

Five-state Process Model



Other Useful States (1)

- **New state –**
  - **OS has performed the necessary actions to create the process:**
    - has created a process identifier.
    - has created tables needed to manage the process.
  - **but has not yet committed to execute the process (not yet admitted):**
    - because resources are limited.

PROF. HARJIT DHILLON

Other Useful States (2)

- **Terminated state –**
  - **Program termination moves the process to this state.**
  - **It is no longer eligible for execution.**
  - **Tables and other info are temporarily preserved for auxiliary program –**
    - Example: accounting program that cumulates resource usage for billing the users.
- **The process (and its tables) gets deleted when the data is no more needed.**

CSUN® COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Reasons for Process Creation

- System initialization.
- Submission of a batch job.
- User logs on.
- Created by OS to provide a service to a user (e.g., printing a file).
- A user request to create a new process.
- Spawned by an existing process
  - a program can dictate the creation of a number of processes.

PROF. HARJIT DHILLON

Process Creation (1)

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Possible resource sharing:
  - Parent and children share all resources.
  - Children share subset of parent’s resources.
  - Parent and child share no resources.
- Possible execution:
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

CSUN® COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Process Creation (2)

- Assign a unique process identifier (PID).
- Allocate space for the process image.
- Initialize process control block
  - many default values (e.g., state is New, no I/O devices or files...).
- Set up appropriate linkages
  - Ex: add new process to linked list used for the scheduling queue.

Process Creation (3)

- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.
- Linux/UNIX examples
  - fork system call creates new process.
  - exec system call used after a fork to replace the process' memory space with a new program.

PROF. HARJIT DHILLON

When does a process get terminated?

- Batch job issues *Halt* instruction.
- User logs off.
- Process executes a service request to terminate.
- Parent kills child process.
- Error and fault conditions.

PROF. HARJIT DHILLON

Reasons for Process Termination (1)

- Normal/Error/Fatal exit.
- Time limit exceeded
- Memory unavailable
- Memory bounds violation
- Protection error
  - example: write to read-only file
- Arithmetic error
- Time overrun
  - process waited longer than a specified maximum for an event.

PROF. HARJIT DHILLON

Reasons for Process Termination (2)

- I/O failure
- Invalid instruction
  - happens when trying to execute data.
- Privileged instruction
- Operating system intervention
  - such as when deadlock occurs.
- Parent request to terminate one child.
- Parent terminates so child processes terminate.

PROF. HARJIT DHILLON

Process Termination

- **Process executes last statement and asks the operating system to terminate it (exit):**
  - **Output data from child to parent (via wait).**
  - **Process' resources are deallocated by operating system.**
- **Parent may terminate execution of child processes (abort):**
  - **Child has exceeded allocated resources.**
  - **Mission assigned to child is no longer required.**
  - **If Parent is exiting:**
    - Some OSs do not allow child to continue if its parent terminates.
    - Cascading termination – all children terminated.

PROF. HARJIT DHILLON

Process States

- *User Running* Executing in user mode.
- *Kernel Running* Executing in kernel mode.
- *Ready to Run*, in Memory Ready to run as soon as the kernel schedules it.
- *Asleep in Memory* Unable to execute until an event occurs; process is in main memory (a blocked state).
- *Ready to Run*, *Swapped* Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
- *Sleeping, Swapped* The process is awaiting an event and has been swapped to secondary storage (a blocked state).
- *Preempted* Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
- *Created* Process is newly created and not yet ready to run.
- *Zombie* Process no longer exists, but it leaves a record for its parent process to collect.

PROF. HARJIT DHILLON

Processor Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing.
- Process scheduler selects among available processes for next execution on CPU.
- Maintains scheduling queues of processes:
  - Job queue – set of all processes in the system.
  - Ready queue – set of all processes residing in main memory, ready and waiting to execute.
  - Device queues – set of processes waiting for an I/O device.
- Processes migrate among the various queues.

Types of Schedulers

1. Long-term scheduler (jobs scheduler) – selects which programs/processes should be brought into the ready queue.
2. Medium-term scheduler (emergency scheduler) – selects which job/process should be swapped out if system is loaded.
3. Short-term scheduler (CPU scheduler) – selects which process should be executed next and allocates CPU.

Long-Term Scheduling

- Determines which programs are admitted to the system for processing.
- Controls the degree of multiprogramming.
- If more processes are admitted:
  - less likely that all processes will be blocked – better CPU usage.
  - each process has less fraction of the CPU.
- Long-term scheduler strives for good process mix.

Short-Term Scheduling

- Determines which process is going to execute next (also called CPU scheduling).
- The short term scheduler is also known as the dispatcher (which is part of it).
- Is invoked on a event that may lead to choose another process for execution:
  - clock interrupts
  - I/O interrupts
  - operating system calls and traps
  - signals

Aspects of Schedulers

- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow).
- The long-term scheduler controls the degree of multiprogramming.
- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast).
- Processes can be described as either:
  - I/O-bound process – spends more time doing I/O than computations, many short CPU bursts.
  - CPU-bound process – spends more time doing computations; few very long CPU bursts.

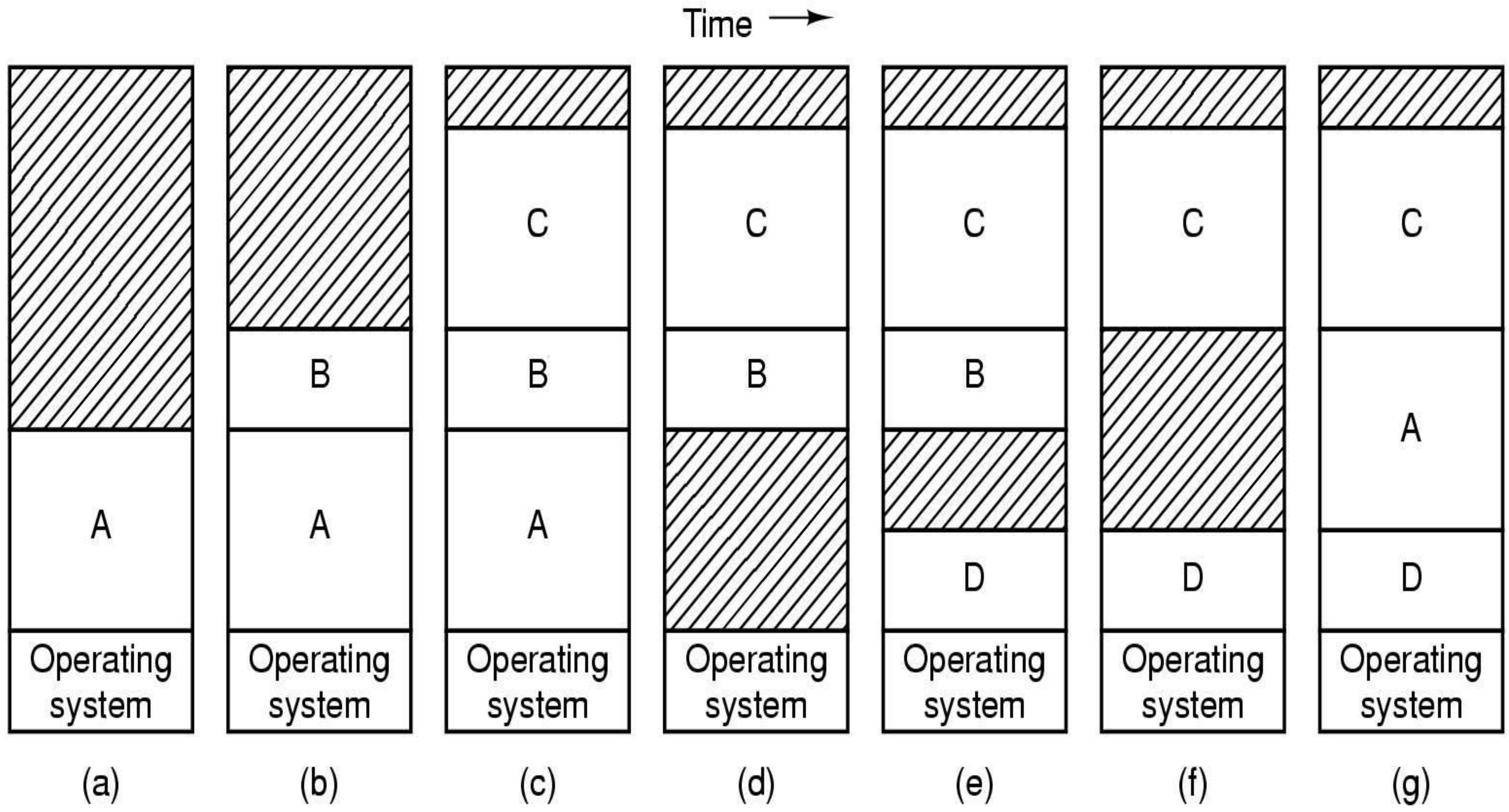
Medium-Term Scheduling

- So far, all processes have to be (at least partly) in main memory.
- Even with virtual memory, keeping too many processes in main memory will deteriorate the system’s performance.
- The OS may need to swap out some processes to disk, and then later swap them back in.
- Swapping decisions based on the need to manage multiprogramming.

Dynamics of Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows).
- System maintains a ready queue of ready-to-run processes which have memory images on disk

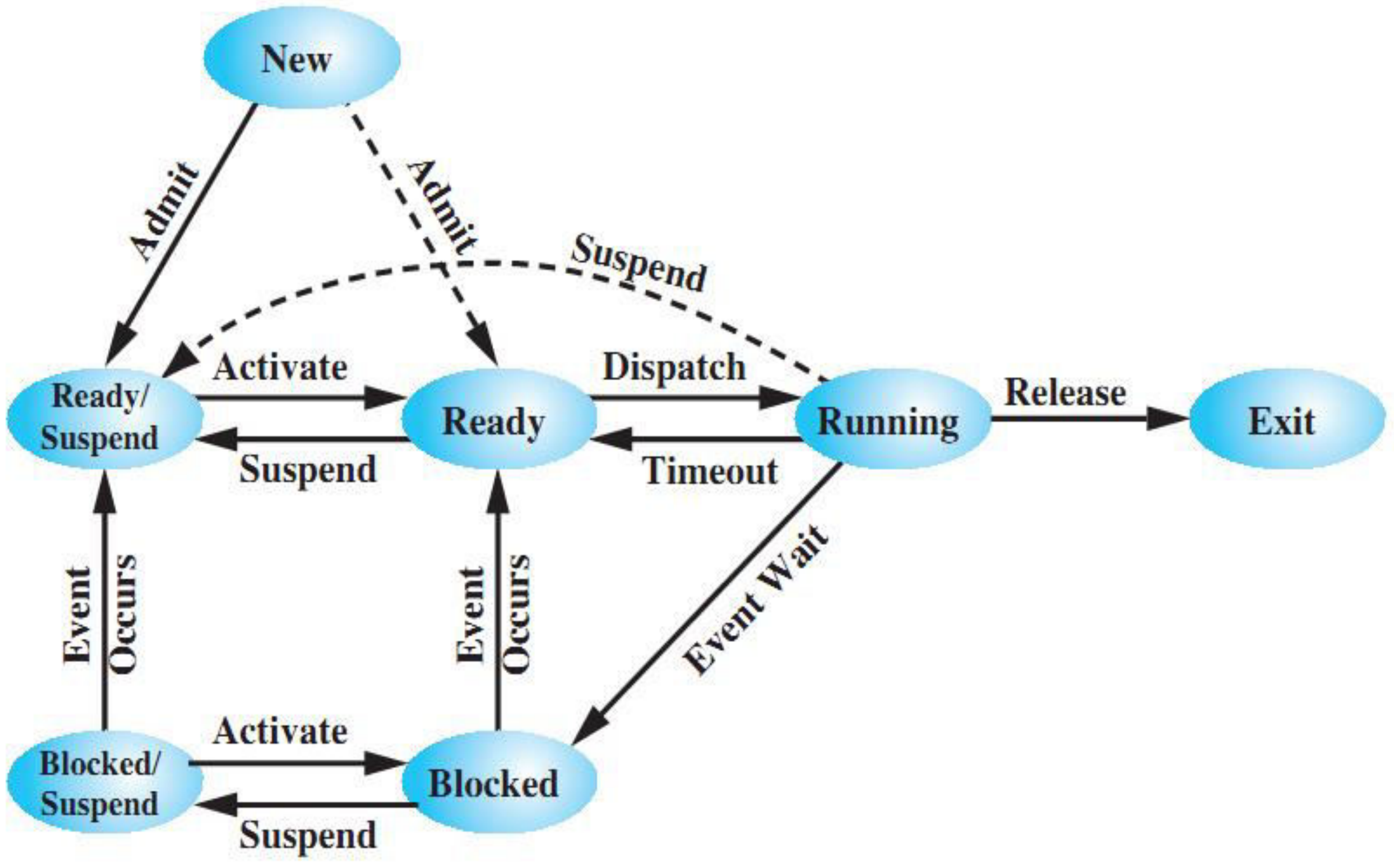
Swapping Example



Support for Swapping

- The OS may need to suspend some processes, i.e., to swap them out to disk and then swap them back in.
- We add 2 new states:
  - Blocked Suspend: blocked processes which have been swapped out to disk.
  - Ready Suspend: ready processes which have been swapped out to disk.

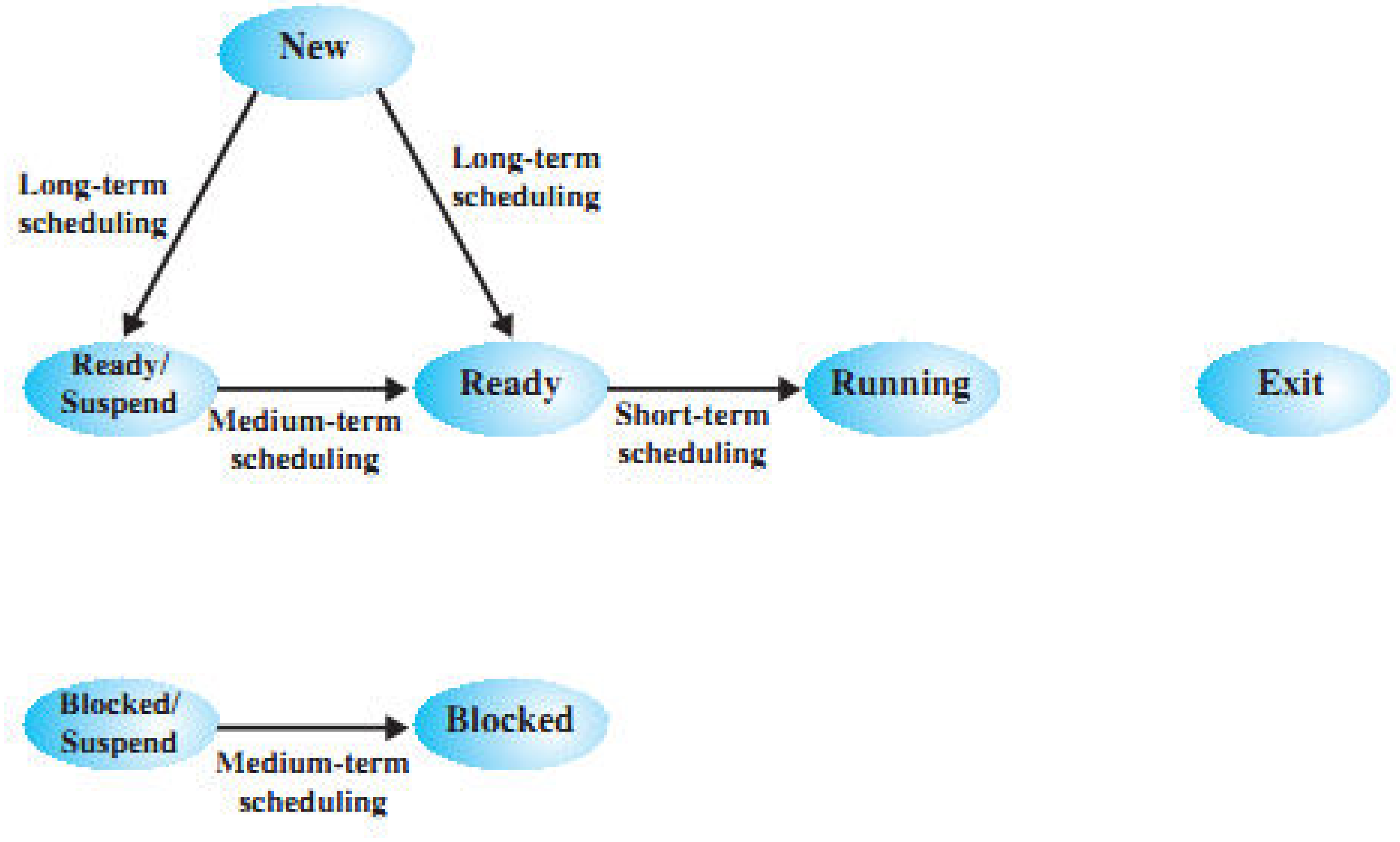
A Seven-state Process Model



New state transitions

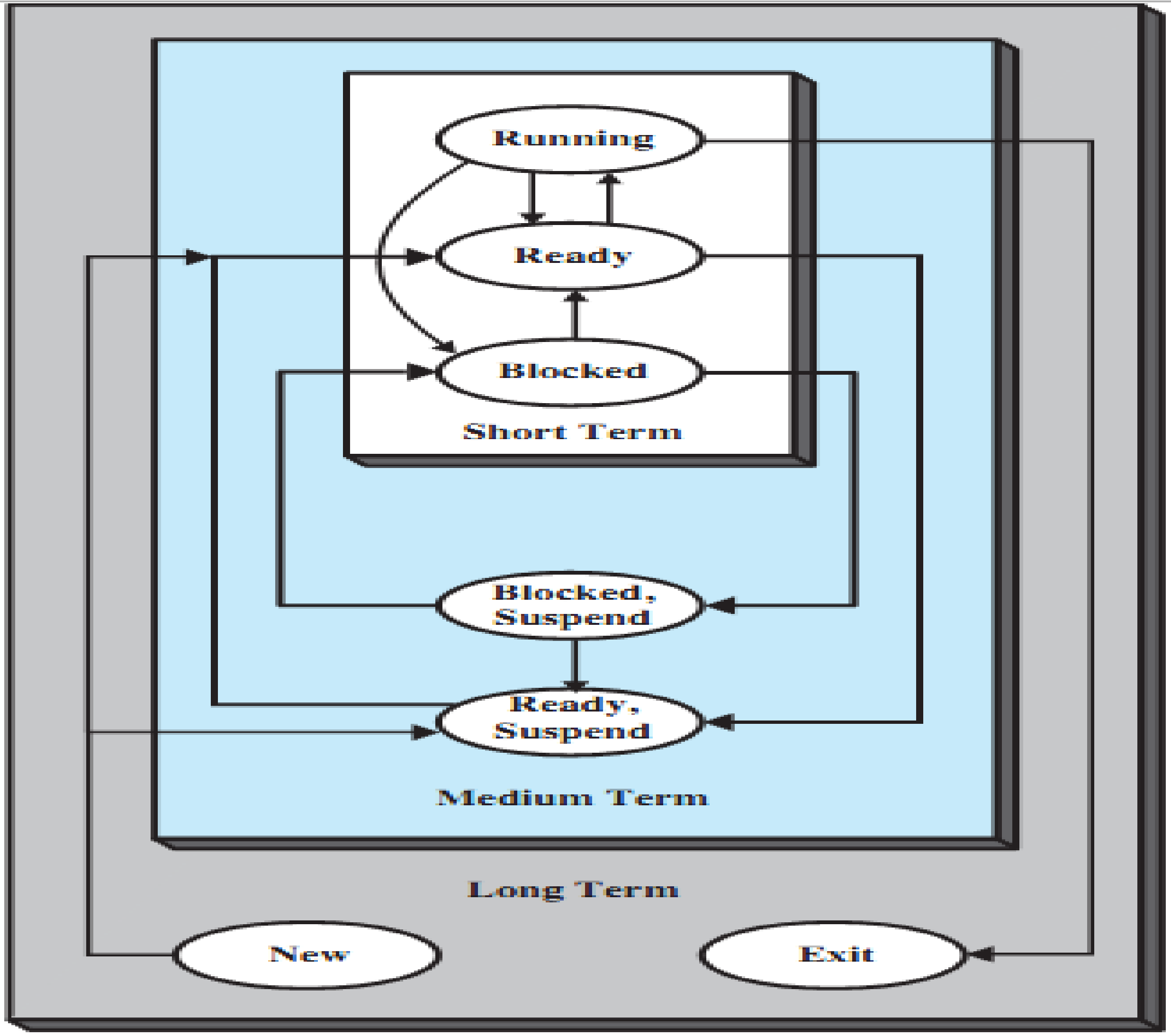
- Blocked –> Blocked Suspend
  - When all processes are blocked, the OS will make room to bring a ready process in memory.
- Blocked Suspend –> Ready Suspend
  - When the event for which it has been waiting occurs (state info is available to OS).
- Ready Suspend –> Ready
  - when no more ready processes in main memory.
- Ready –> Ready Suspend (unlikely)
  - When there are no blocked processes and must free memory for adequate performance.

Classification of Scheduling Activity



PROF. HARJIT DHILLON

Another view of the 3 levels of scheduling



PROF. HARJIT DHILLON

Dispatcher (short-term scheduler)

- Is an OS program that moves the processor from one process to another.
- It prevents a single process from monopolizing processor time.
- It decides who goes next according to a scheduling algorithm.
- The CPU will always execute instructions from the dispatcher while switching from process A to process B.

CSUN COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Process Scheduling Queues

- Process queue – set of *all* processes in the system.
- Ready queue – set of processes residing in main memory, ready and waiting to execute.
- Device queues – set of processes waiting for an I/O device.
- Processes migrate among the various queues.

When to Switch a Process?

- **A process switch may occur whenever the OS has gained control of CPU. i.e., when:**
  - **Supervisor Call**
    - explicit request by the program (example: file open) – the process will probably be blocked.
  - **Trap**
    - an error resulted from the last instruction –
      it may cause the process to be moved to terminated state.
  - **Interrupt**
    - the cause is external to the execution of the current instruction – control is transferred to Interrupt Handler.

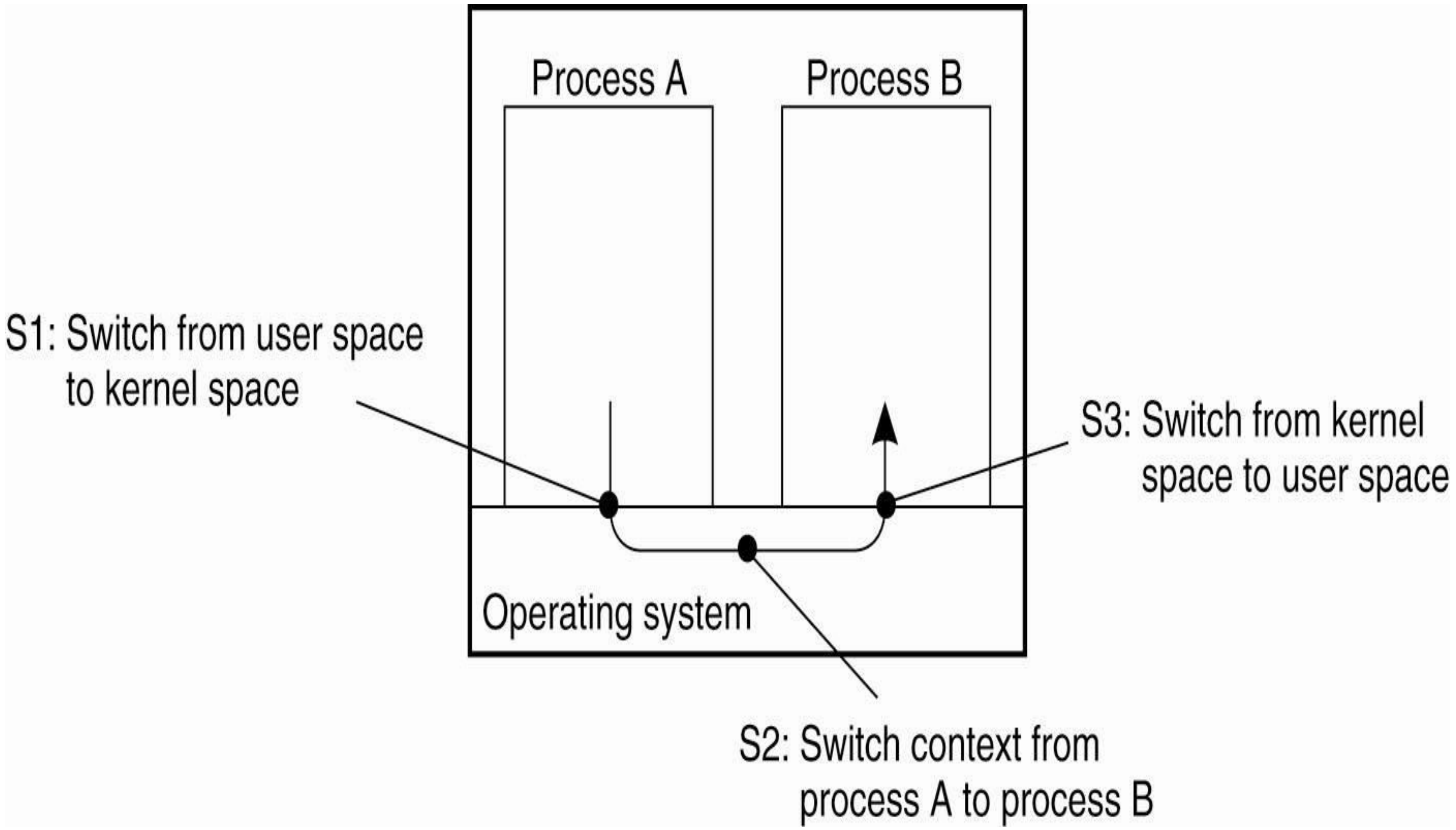
PROF. HARJIT DHILLON

Context Switch

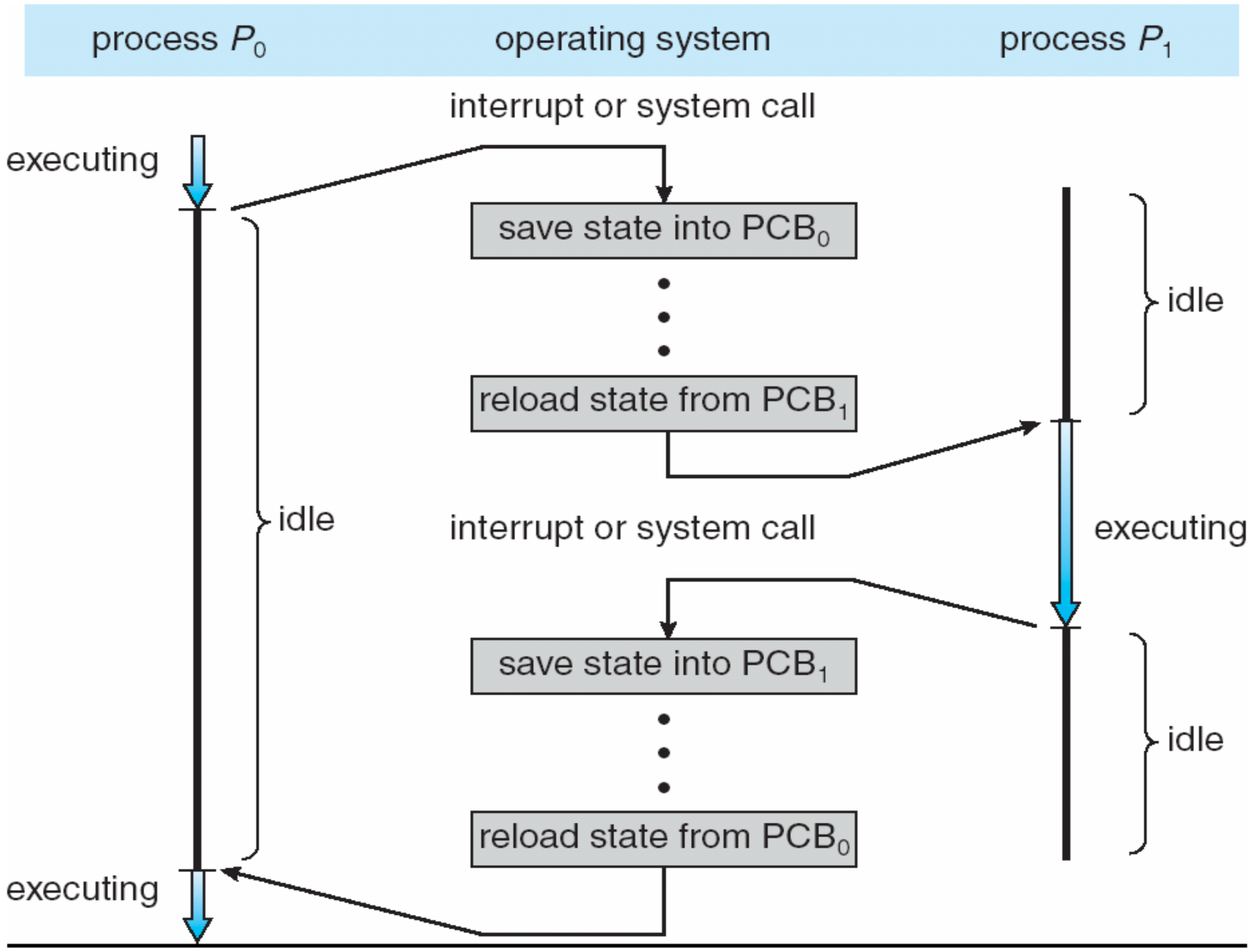
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- This is called context switch.
- Context of a process represented in the PCB.
- The time it takes is dependent on hardware support.
- Context-switch time is overhead; the system does no useful work while switching.

PROF. HARJIT DHILLON

Context switch between processes (1)


Process A
Process B
S1: Switch from user space to kernel space
S3: Switch from kernel space to user space
Operating system
S2: Switch context from process A to process B

Context switch between processes (2)



PROF. HARJIT DHILLON

Steps in Context Switch

- Save context of processor including program counter and other registers.
- Update the PCB of the running process with its new state and other associate information.
- Move PCB to appropriate queue – ready, blocked,
- Select another process for execution.
- Update PCB of the selected process.
- Restore CPU context from that of the selected process.

Mode Switch

- It may happen that an interrupt does not produce a context switch.
- The control can just return to the interrupted program.
- Then only the processor state information needs to be saved on stack.
- This is called mode switch (user to kernel mode when going into Interrupt Handler).
- Less overhead: no need to update the PCB like for context switch.

Interprocess Communication

A process is **independent** if it cannot affect or be affected by the other executing processes.

A process is **cooperating** if it can affect or be affected by other executing processes, e.g. sharing data with others.

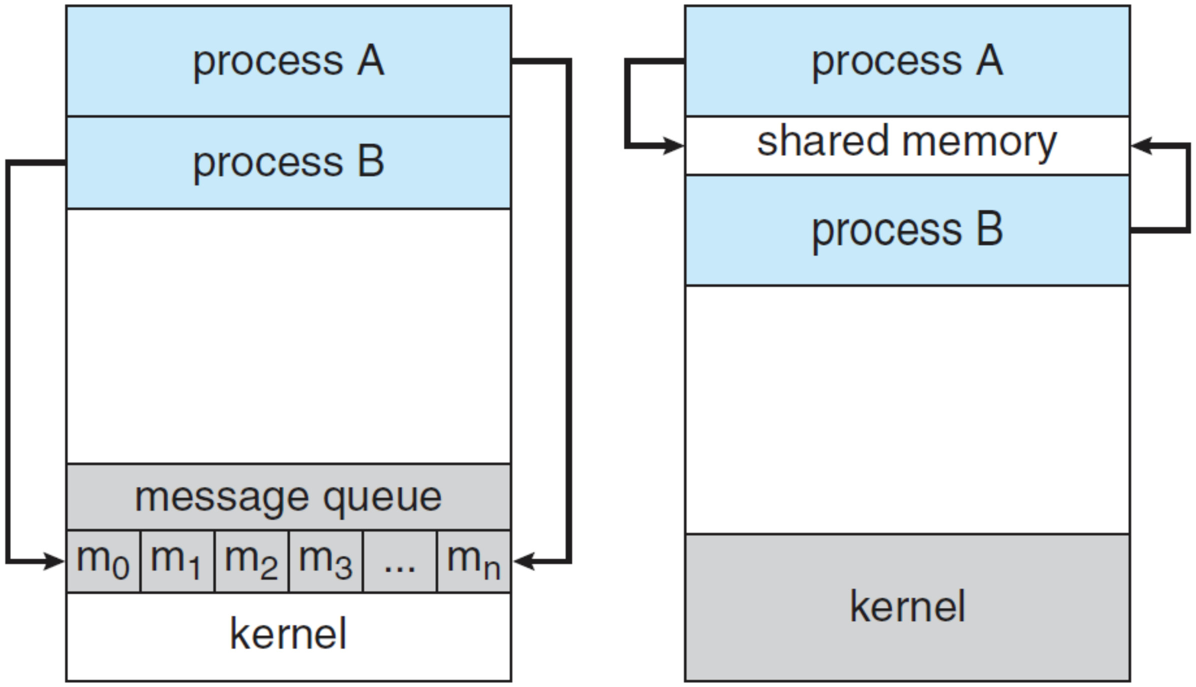
Reasons for process cooperation:

- **Information sharing**, e.g. concurrent access to file.
- **Speedup**, e.g. a task is broken into subtasks executed in parallel (in multiple processing cores).
- **Modularity**, e.g. dividing system functions into separate processes or threads.

Interprocess Communication

Cooperation requires an **interprocess communication (IPC)** mechanism to allow data and info exchange.

There are **shared memory** and **message passing** models. OS may implement both.



Interprocess Communication

Message passing is useful for exchanging smaller data, because no conflicts need be avoided.

Shared memory is faster than message passing, which implementation uses system calls. Shared-memory needs no kernel assistance.

Shared memory suffers from **cache coherency** issues, arising from shared data migration among few caches.

As the number of processing cores increases, message passing may take over shared-memory.

Shared-Memory Systems

•Shared-memory resides in the address space of the process creating it. The communicating processes attach it to their address space.

•Since OS normally prevents one process from accessing another process's memory, shared memory requires to remove this restriction.

•The form of the data and the location are determined by the processes and are not under the OS control.

•The processes are responsible to not writing to the same location simultaneously.

Message-Passing Systems

•Message-passing provides two operations: **send(message)** and **receive(message)**.

•A **communication link** must exist between processes.

•There are several methods for logically implementing a link and the **send()** / **receive()** operations:

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

PROF. HARJIT DHILLON

Process Characteristics (1)

- Unit of resource ownership – process is allocated:
  - an address space to hold the process image.
  - control of some resources (files, I/O devices...).
- Unit of dispatching – process is an execution path through one or more programs:
  - execution may be interleaved with other process.
  - the process has an execution state and a dispatching priority.

Process Characteristics (2)

- These two characteristics are treated independently by some recent OSs:
  1. The unit of resource ownership is usually referred to as a Task or (for historical reasons) also as a Process.
  2. The unit of dispatching is usually referred to a Thread or a Light-Weight Process (LWP).
- A traditional Heavy-Weight Process (HWP) is equal to a task with a single thread.
- Several threads can exist in the same task.
  - *Multithreading -* The ability of an OS to support multiple, concurrent paths of execution within a single process.