S J P N Trust's

Hirasugar Institute of Technology, Nidasoshi.

Inculcating Values, Promoting Prosperity

Accredited at 'A' Grade by NAAC

Programmes Accredited by NBA: CSE, ECE, EEE & ME

Subject: Operating System (18CS43)

Process Management

By: Prof. Prasanna Patil
Asst. Prof. , Dept. of Computer Science & Engg.,
Hirasugar Institute of Technology, Nidasoshi

Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Inter Process Communication

Prasanna Patil, Dept of CSE, HIT Nidasoshi

OBJECTIVES

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication
- To describe communication in client-server systems

Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably
- Process – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section

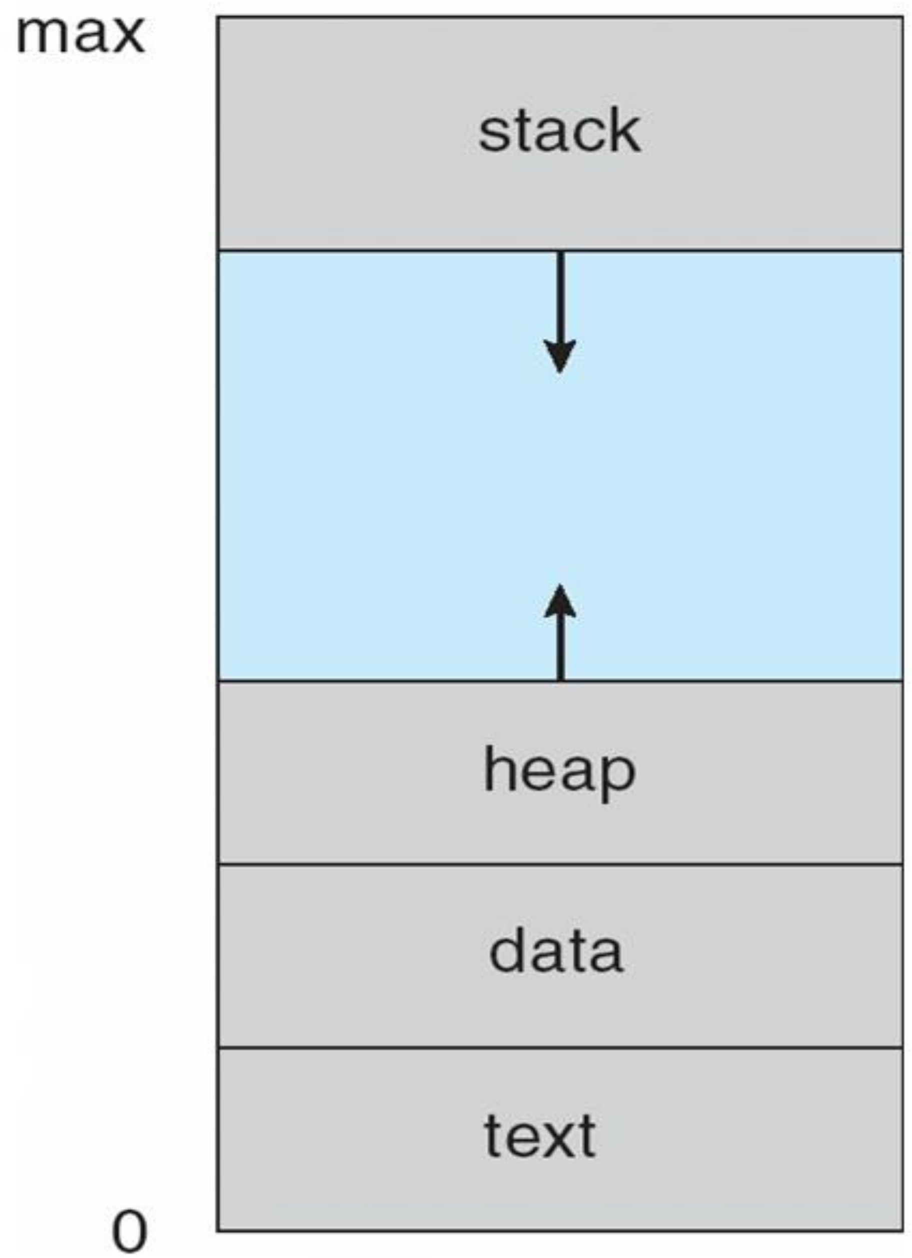
Prasanna Patil, Dept of CSE, HIT Nidasoshi

THE PROCESS

- 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
- Program is passive entity, process is active
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

Prasanna Patil, Dept of CSE, HIT Nidasoshi

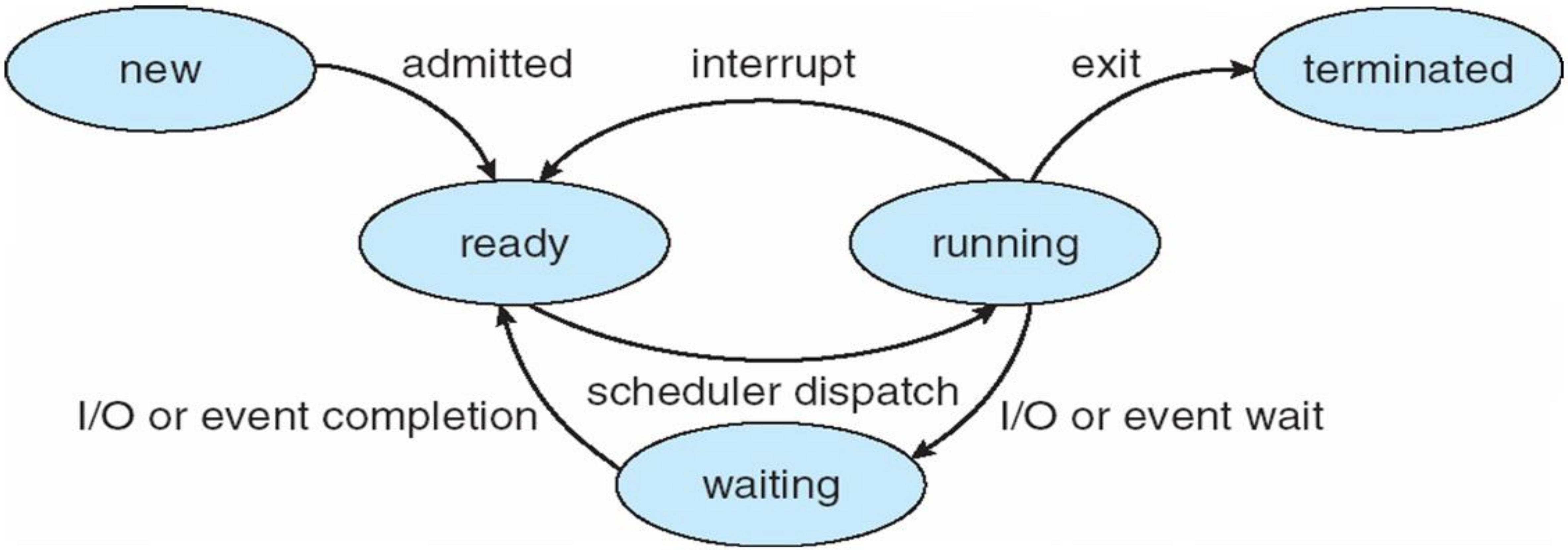
Process in Memory



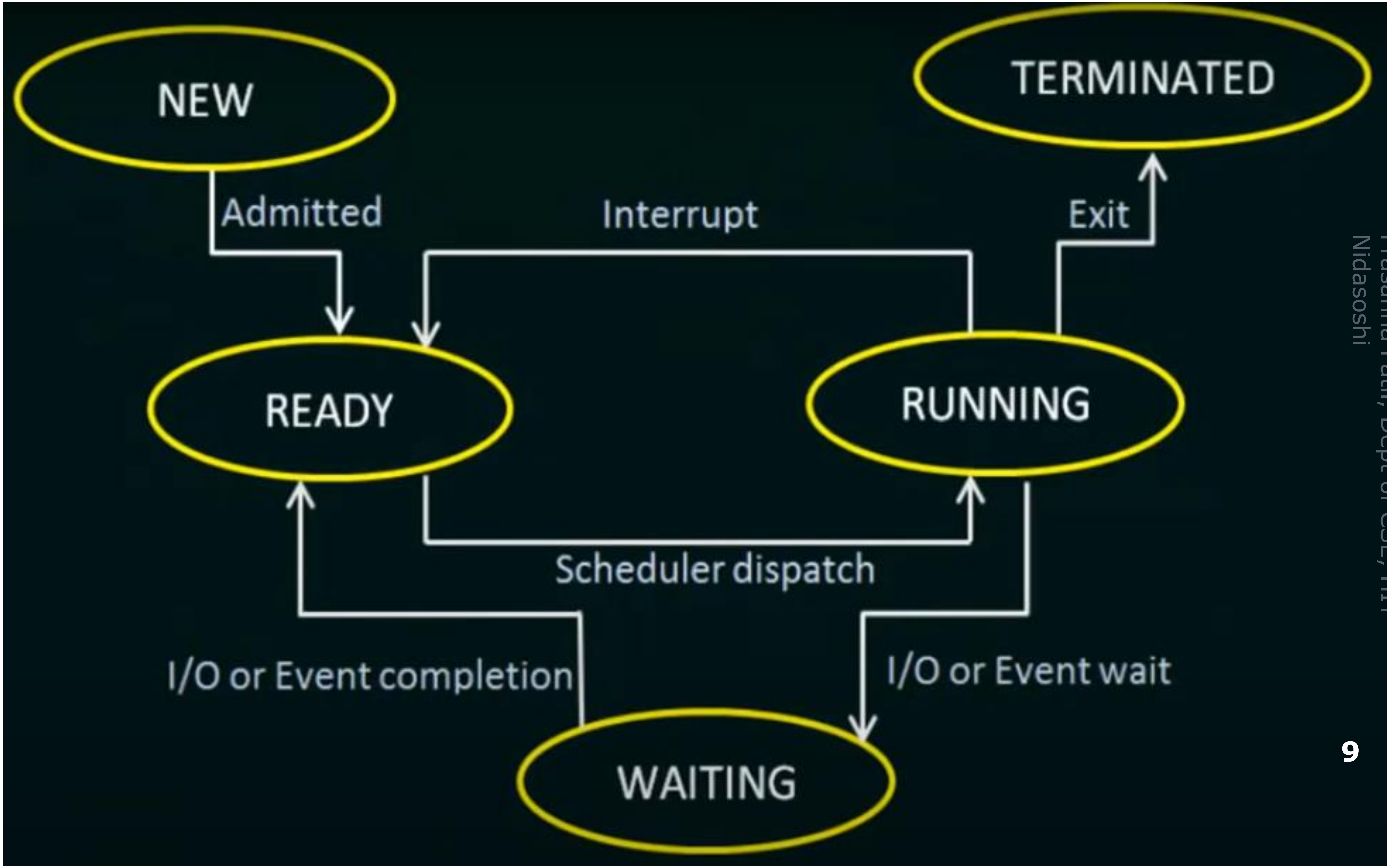
PROCESS STATE

- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

DIAGRAM OF PROCESS STATE



Prasanna Patil, Dept of CSE, HIT Nidasoshi



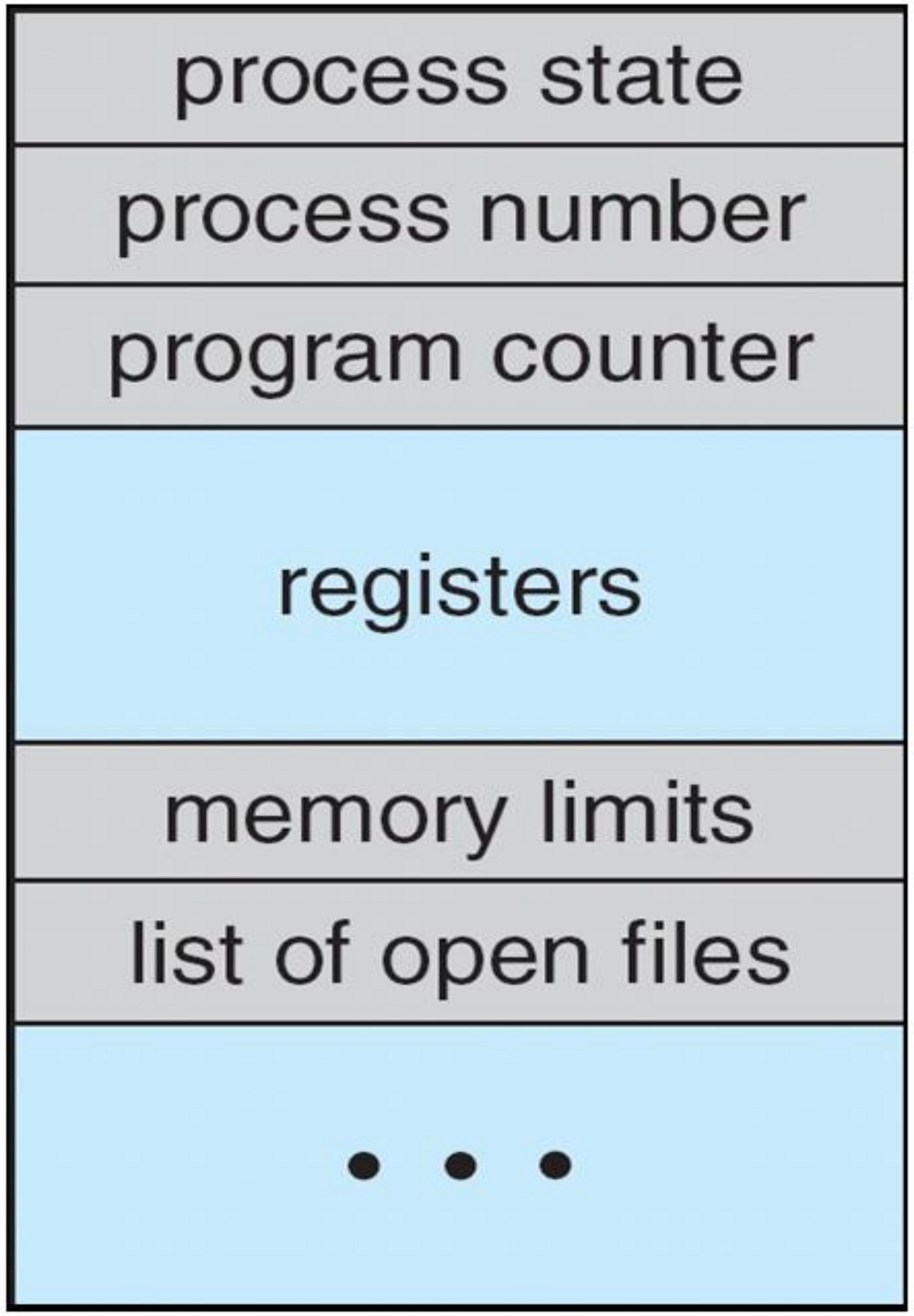
Prasanna Patil, Dept of CSE, HIT Nidasoshi

Process Control Block (PCB)

Information associated with each process

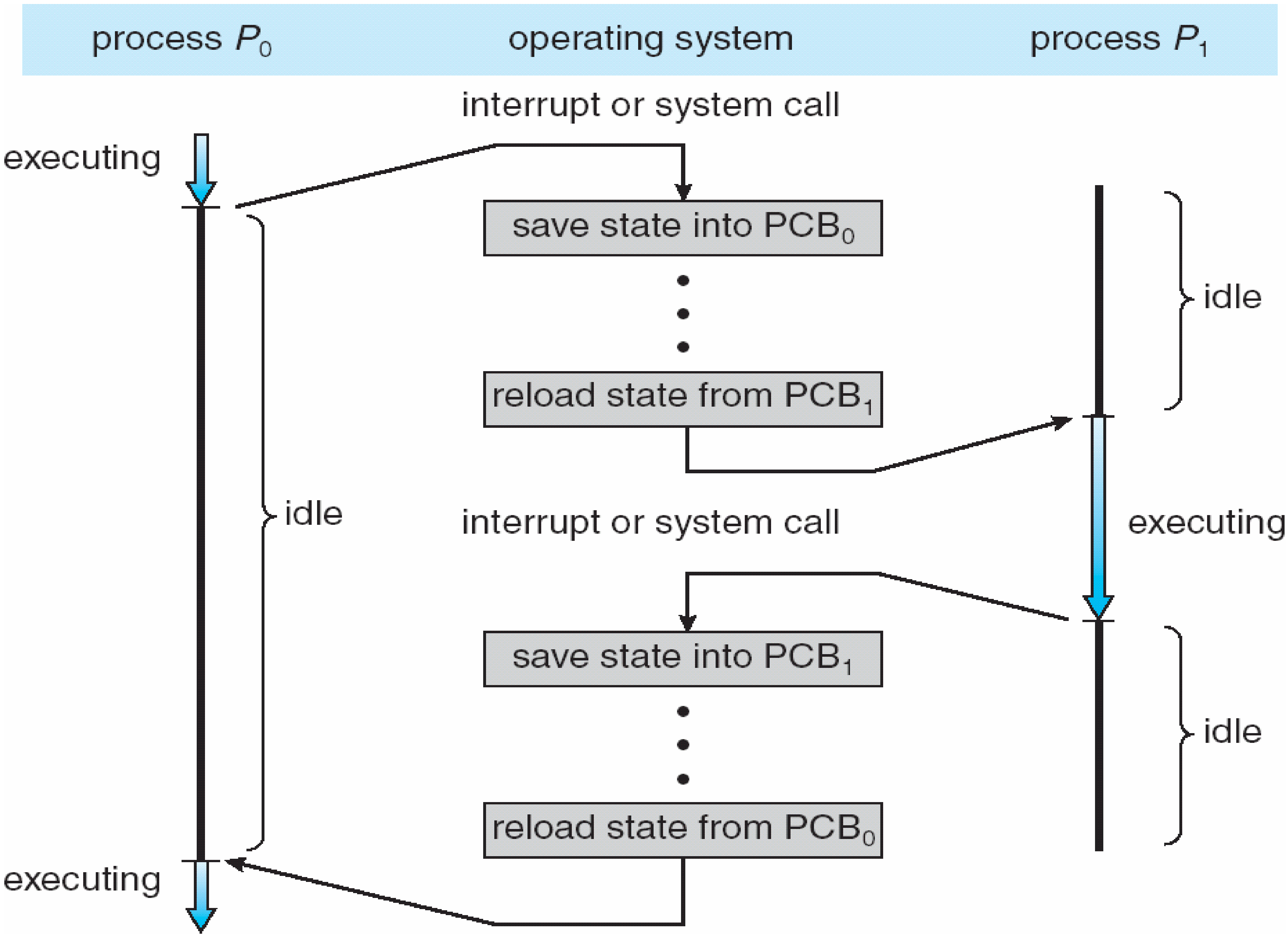
- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

PROCESS/ TASK CONTROL BLOCK (PCB)



Prasanna Patil, Dept of CSE, HIT Nidasoshi

CPU Switch From Process to Process

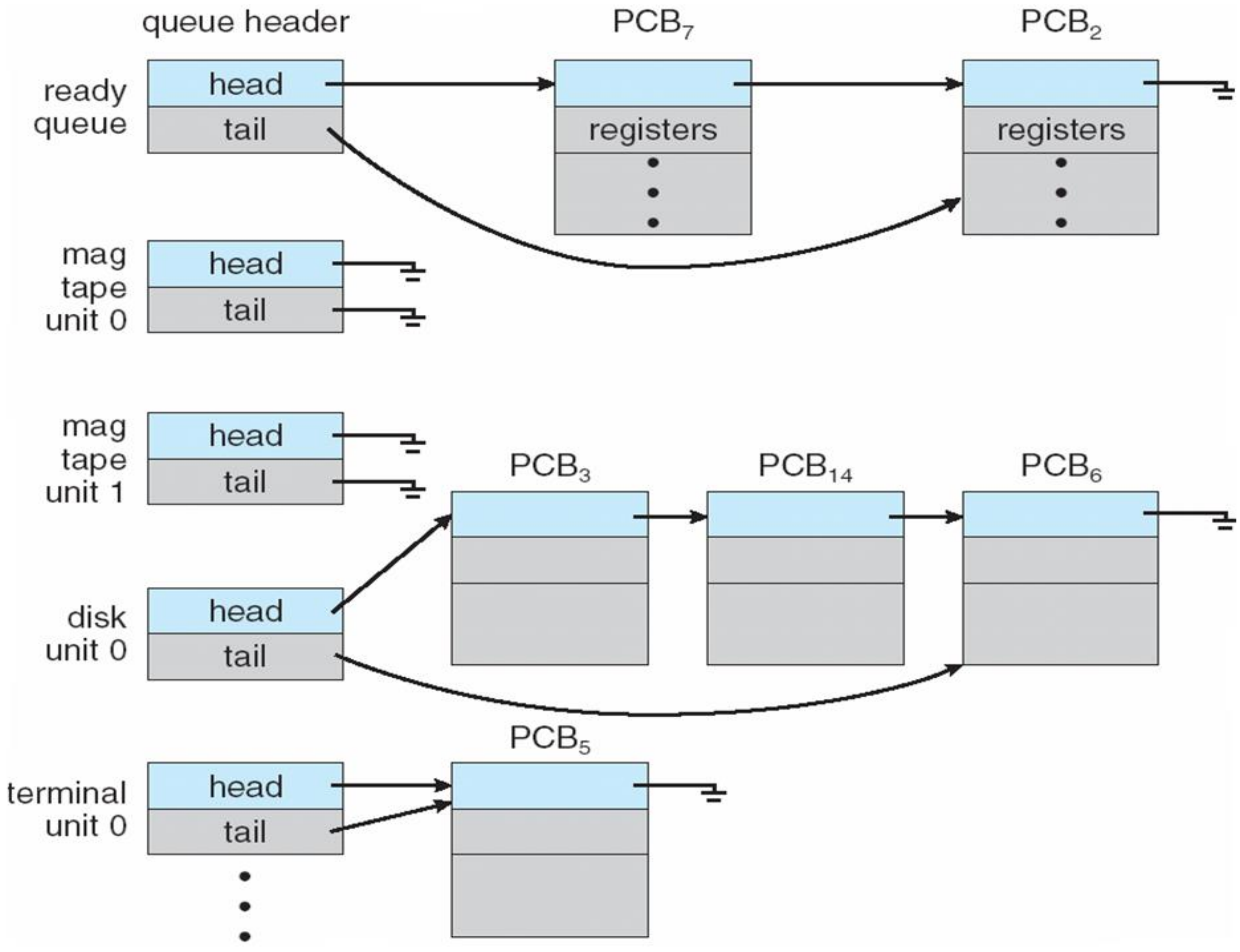


Prasanna Patil, Dept of CSE, HIT Nidasoshi

PROCESS SCHEDULING

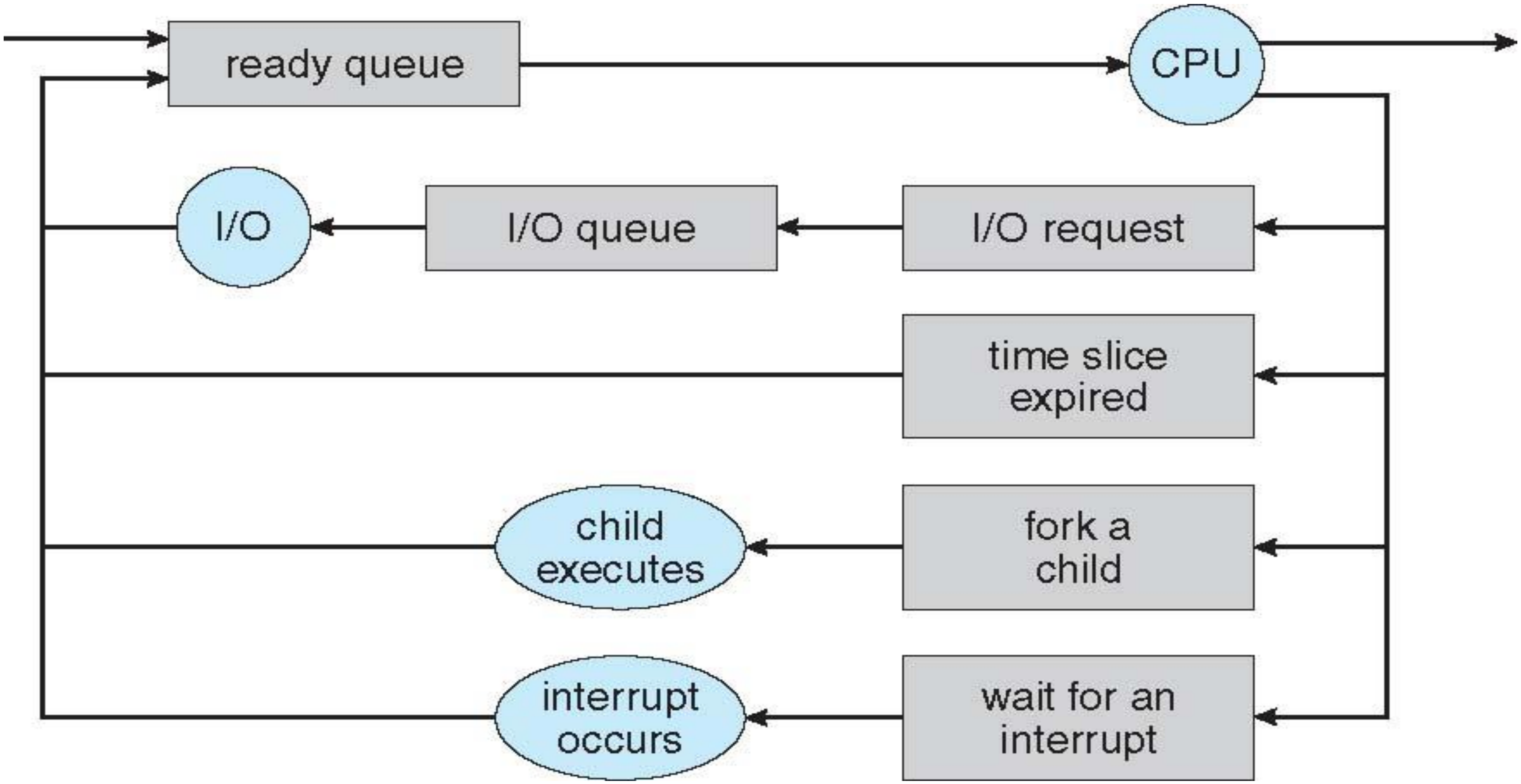
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing 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

Ready Queue And Various I/O Device Queues



Prasanna Patil, Dept of CSE, HIT Nidasoshi

Representation of Process Scheduling



Prasanna Patil, Dept of CSE, HIT Nidasoshi

SCHEDULERS

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system

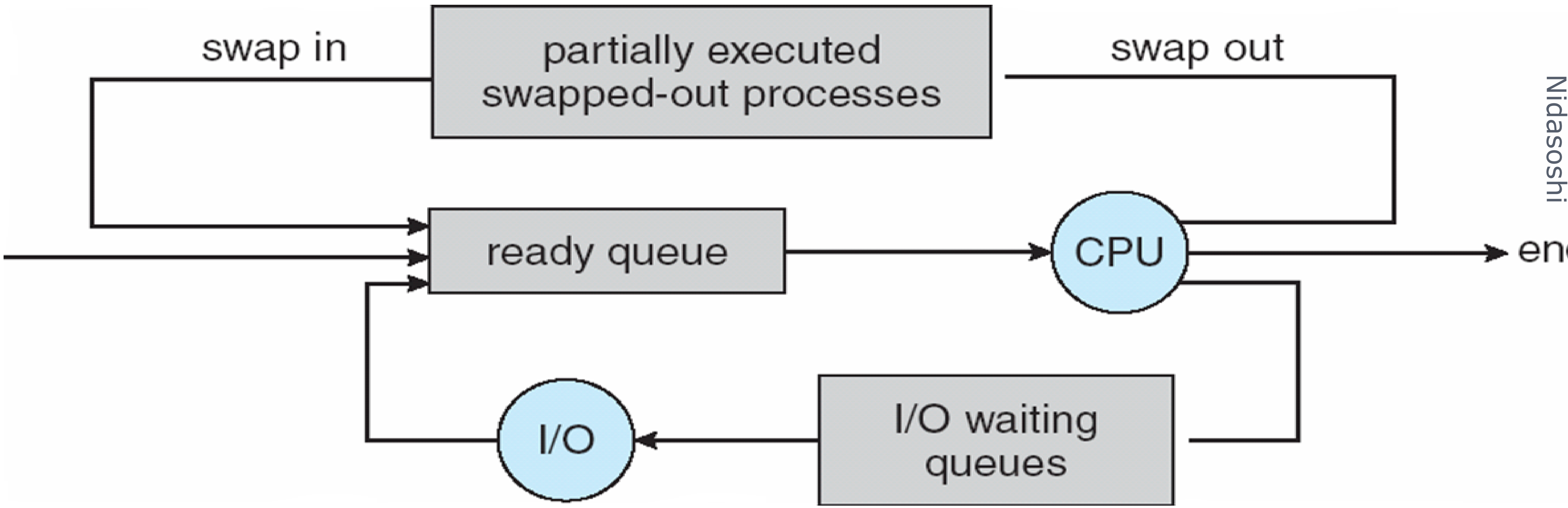
Prasanna Patil, Dept of CSE, HIT Nidasoshi

Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- 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

Prasanna Patil, Dept of CSE, HIT Nidasoshi

ADDITION OF MEDIUM TERM SCHEDULING



Prasanna Patil, Dept of CSE, HIT Nidasoshi

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 via a **context switch**.
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

Prasanna Patil, Dept of CSE, HIT Nidasoshi

Process Creation

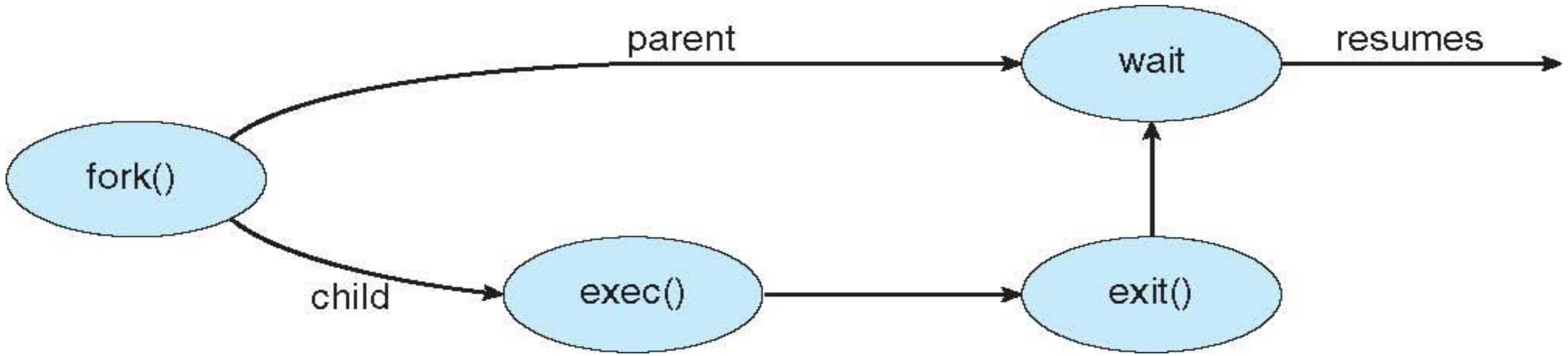
- **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via **a process identifier (pid)**
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources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

Prasanna Patil, Dept of CSE, HIT Nidasoshi

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child process 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

Process Creation



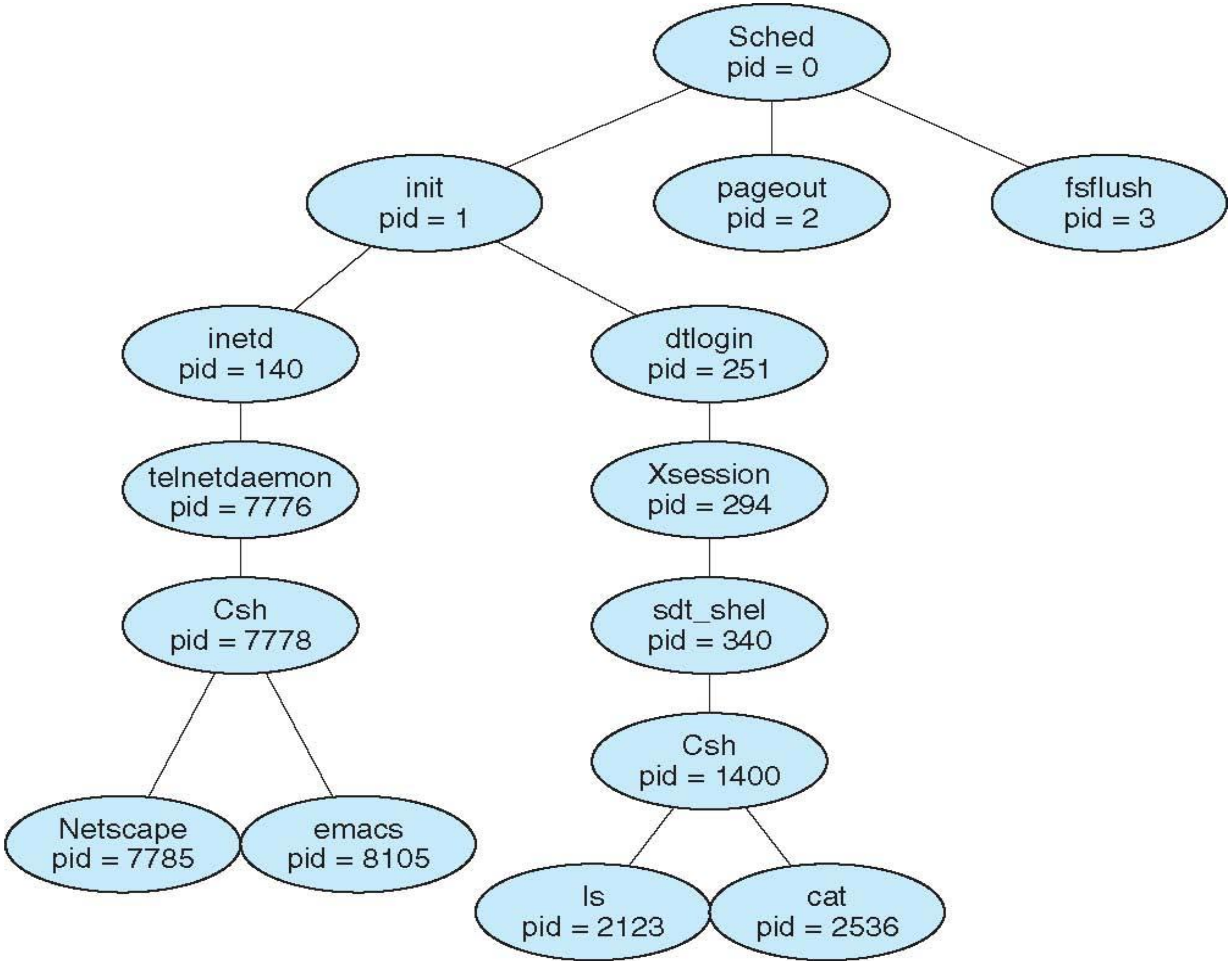
Prasanna Patil, Dept of CSE, HIT Nidasoshi

C Program Forking Separate Process

```
#include <sys/types.h>
#include <studio.h>
#include <unistd.h>
int main()
{
pid_t  pid;
      /* fork another process */
      pid = fork();
      if (pid < 0) { /* error occurred */
             fprintf(stderr, "Fork Failed");
             return 1;
      }
      else if (pid == 0) { /* child process */
             execlp("/bin/ls", "ls", NULL);
      }
      else { /* parent process */
             /* parent will wait for the child */
             wait (NULL);
             printf ("Child Complete");
      }
      return 0;
}
```

Prasanna Patil, Dept of CSE, HIT Nidasoshi

A Tree of Processes on Solaris



Prasanna Patil, Dept of CSE, HIT Nidasoshi

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system

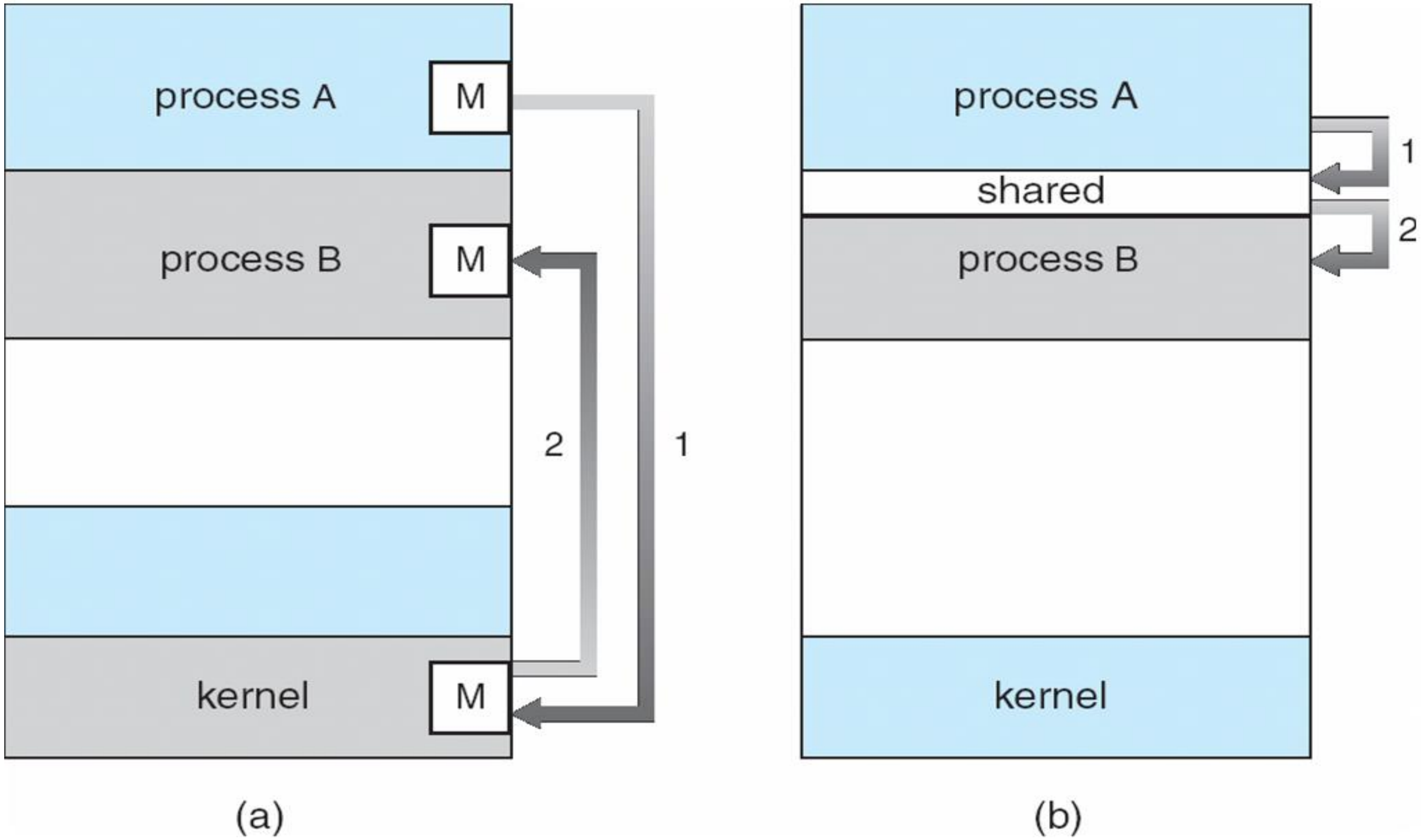
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating systems do not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**

INTERPROCESS COMMUNICATION

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication (IPC)**
- Two models of IPC
  - Shared memory
  - Message passing

Prasanna Patil, Dept of CSE, HIT Nidasoshi

Communications Models



Prasanna Patil, Dept of CSE, HIT Nidasoshi

Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process

- **Cooperating** process can affect or be affected by the execution of another process

- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - *unbounded-buffer* places no practical limit on the size of the buffer
  - *bounded-buffer* assumes that there is a fixed buffer size

Prasanna Patil, Dept of CSE, HIT Nidasoshi

BOUNDED-BUFFER – SHARED-MEMORY SOLUTION

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solution is correct, but can only use BUFFER_SIZE-1 elements

Prasanna Patil, Dept of CSE, HIT Nidasoshi

Bounded-Buffer – Producer

```
Item nextItemProduced;
while (true) {
      /* Produce an item */
      while (((in + 1) % BUFFERSIZE)  == out)
            ;   /* Buffer is full, do nothing*/
      buffer[in] = nextItemProduced;
      in = (in + 1) % BUFFER SIZE;
}
```

Prasanna Patil, Dept of CSE, HIT Nidasoshi

Bounded Buffer – Consumer

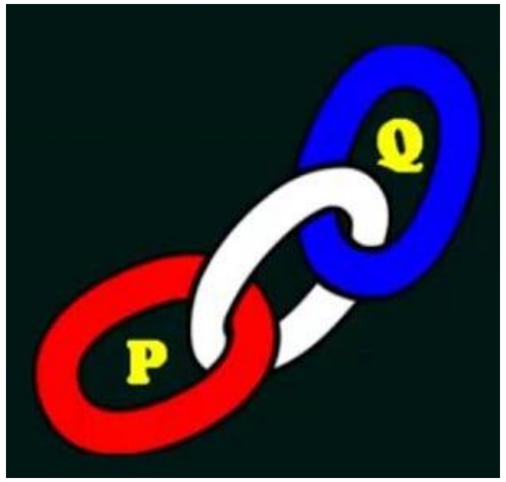
```
Item nextItemConsumed;
while (true) {
        while (in == out)
              ; // do nothing -- nothing to consume

        // remove an item from the buffer
        nextItemConsumed = buffer[out];
        out = (out + 1) % BUFFER SIZE;
    return item;
}
```

Prasanna Patil, Dept of CSE, HIT Nidasoshi

Interprocess Communication – Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)



Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Several methods for logically implementing a link and the send( ) & receive( ) operations:

- Direct or Indirect Communication
- Synchronous or Asynchronous Communication
- Automatic or Explicit Buffering

DIRECT COMMUNICATION

- Processes must name each other explicitly:
  - **send** (*P*, *message*) – send a message to process P
  - **receive**(*Q*, *message*) – receive a message from process Q

- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

receive(*id*, *message*) – receive a message to the id from any process

Prasanna Patil, Dept of CSE, HIT Nidasoshi

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Prasanna Patil, Dept of CSE, HIT Nidasoshi

INDIRECT COMMUNICATION

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox

- Primitives are defined as:

  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A

Prasanna Patil, Dept of CSE, HIT Nidasoshi

INDIRECT COMMUNICATION

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?

- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Synchronization

- Message passing may be either blocking or non-blocking

- **Blocking** is considered **synchronous**
  - **Blocking send:** the sending process is blocked until the message is received by receiving process or by the mailbox.
  - **Blocking receive** has the receiver block until a message is available

- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

Buffering

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits

End of Chapter 3