

# CS420: Operating Systems

## Processes

---

James Moscola  
Department of Physical Sciences  
York College of Pennsylvania



# Process Concept

---

- **Process – a program in execution; process execution must progress in sequential fashion**
- **Textbook uses the terms *job* and *process* almost interchangeably**
- **A process includes:**
  - program counter
  - stack
  - data section

# The Process

- **Multiple parts**

- The program code, also called text section
- Program counter & processor registers
- Stack containing temporary data
  - **Function parameters, return addresses, local variables**
- Data section containing constants and global variables
- Heap containing memory dynamically allocated during run time

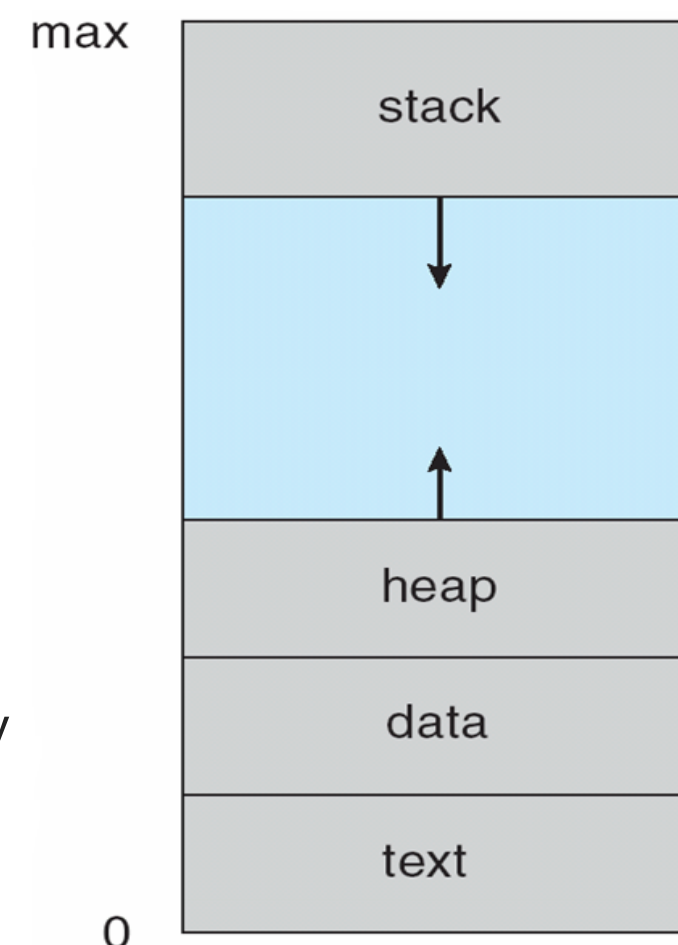
- **A program is passive entity, process is active**

- Program becomes process when executable file loaded into memory

- **One program can consist of several processes**

- **Multiple processes of the same type may run concurrently**

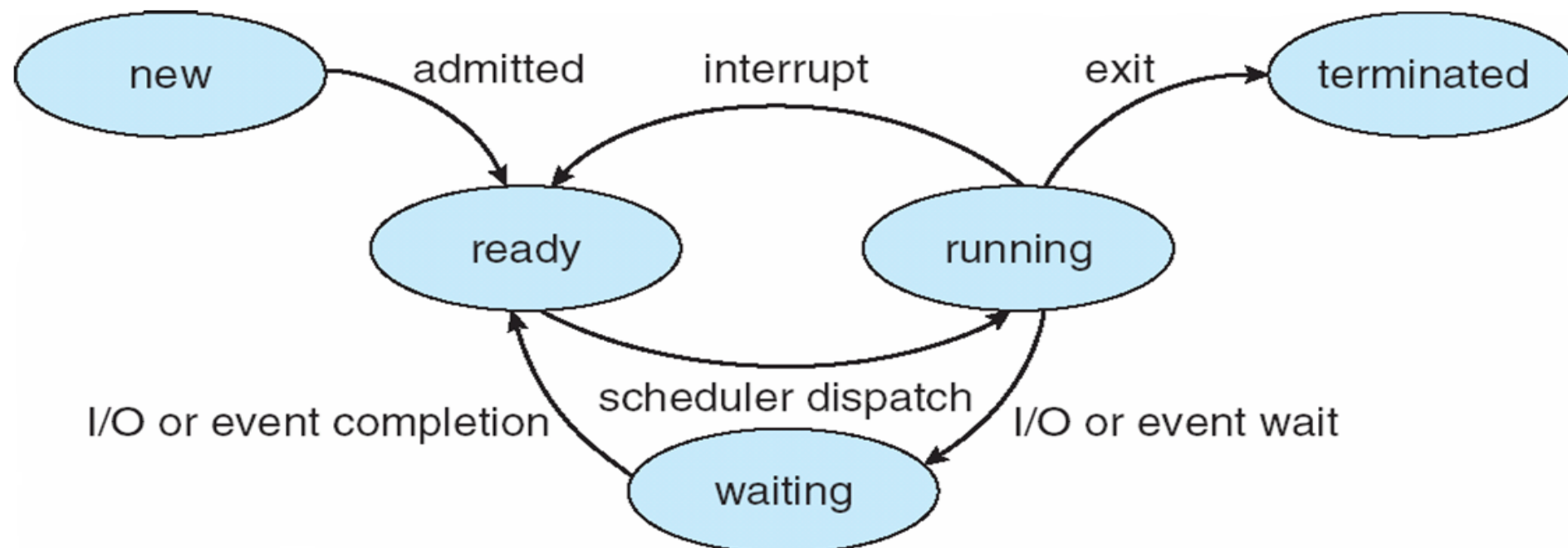
- Consider multiple users executing the same program
- Each has its own memory space



**A Process in Memory**

# Process State

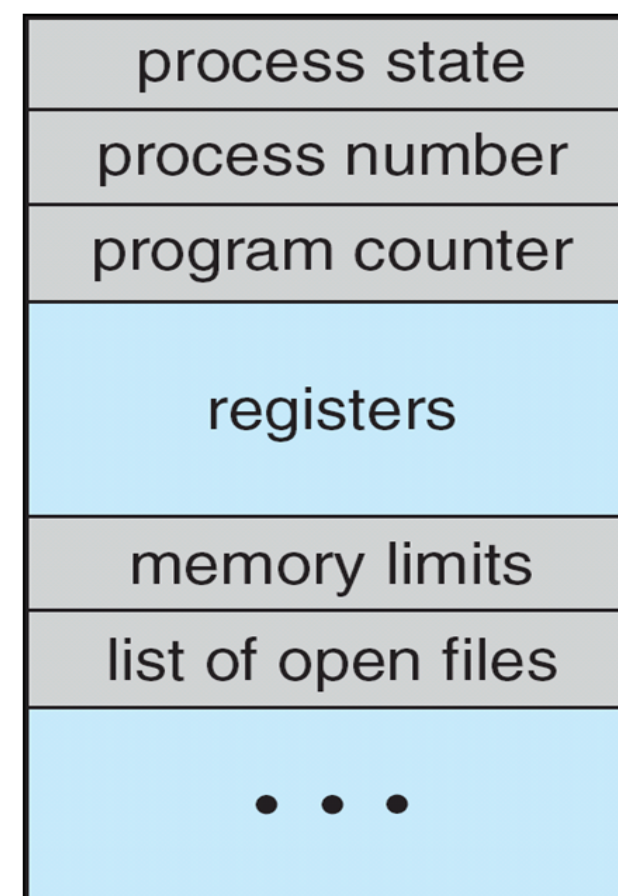
- **As a process executes, it changes state**
  - **new**: The process is being created
  - **ready**: The process is waiting to be assigned to a processor
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **terminated**: The process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



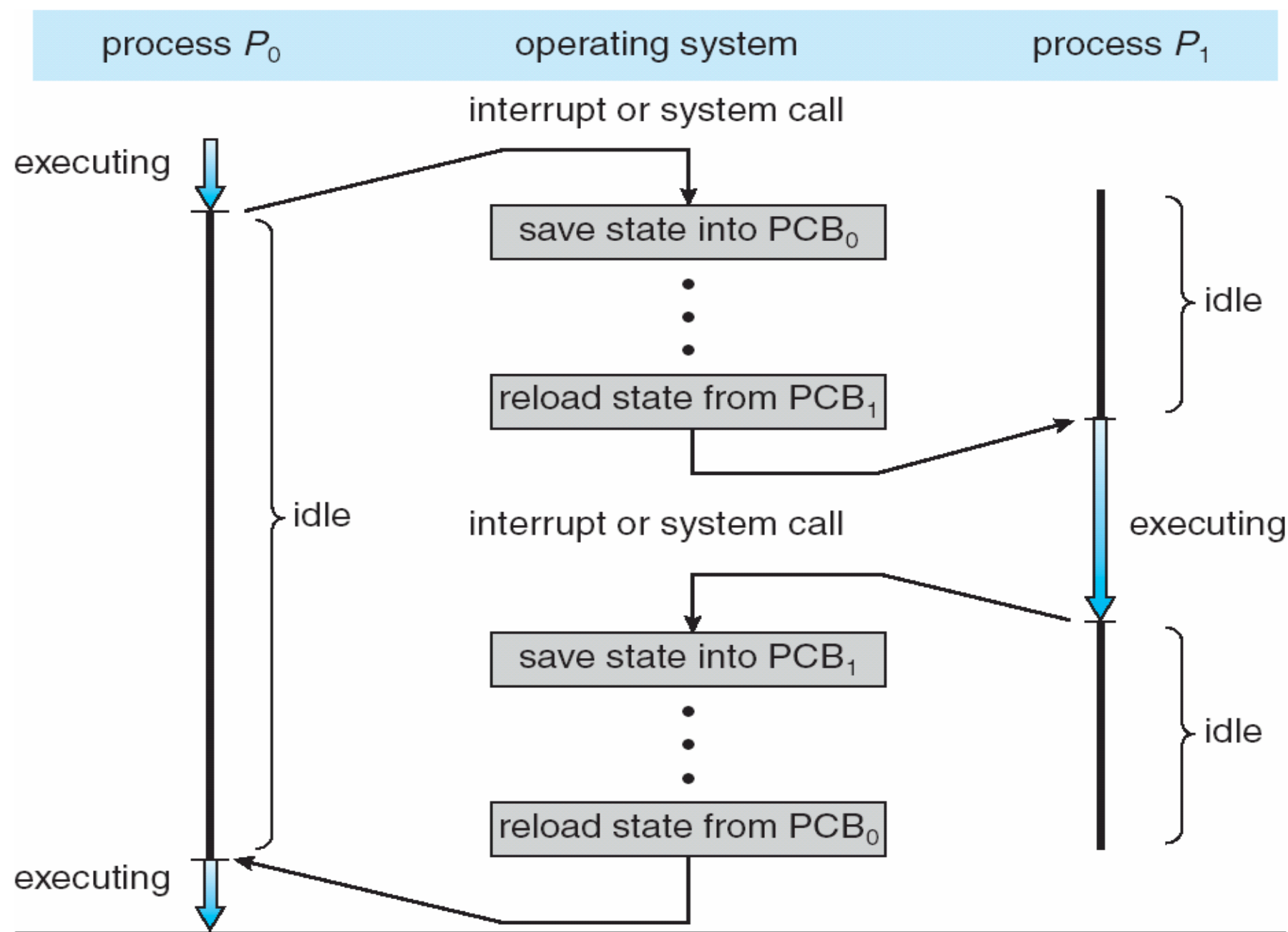
# Process Control Block (PCB)

---

- Each process is represented in the operating system by a **process control block (PCB)**
- The process control block contains information associated with each process including:
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

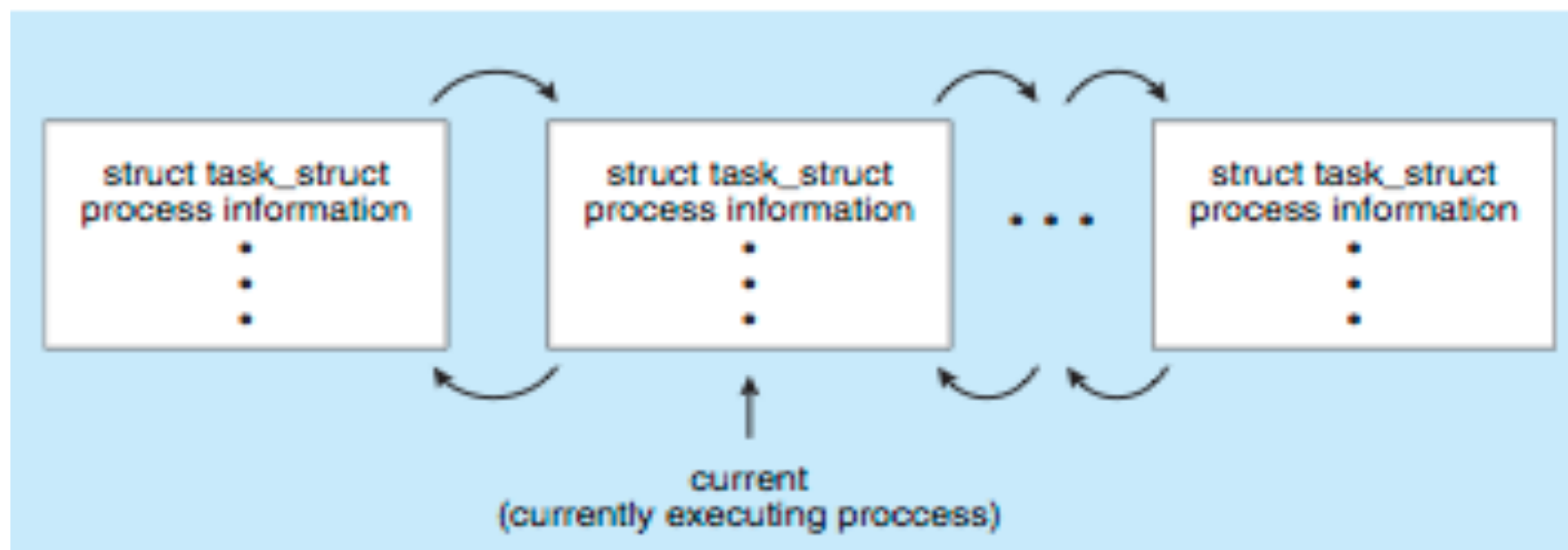


# CPU Switch From Process to Process



# Process Representation in Linux

```
struct task_struct
{
    pid_t pid;                /* process identifier */
    long state;               /* state of the process */
    unsigned int time_slice;  /* scheduling information */
    struct task_struct *parent; /* this process's parent */
    struct list_head children; /* this process's children */
    struct files_struct *files; /* list of open files */
    struct mm_struct *mm;      /* address space of process */
}
```



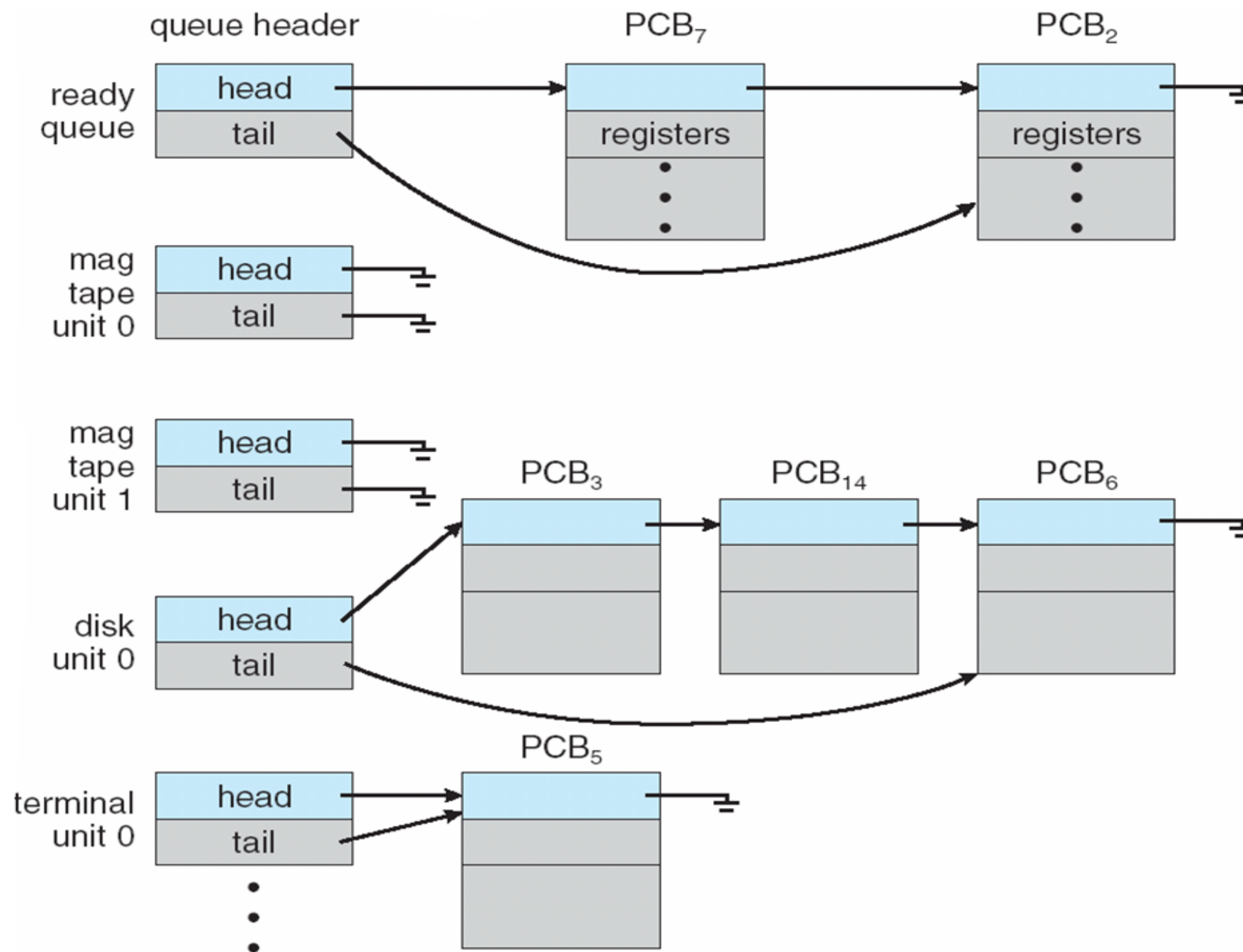
# Process Scheduling

---

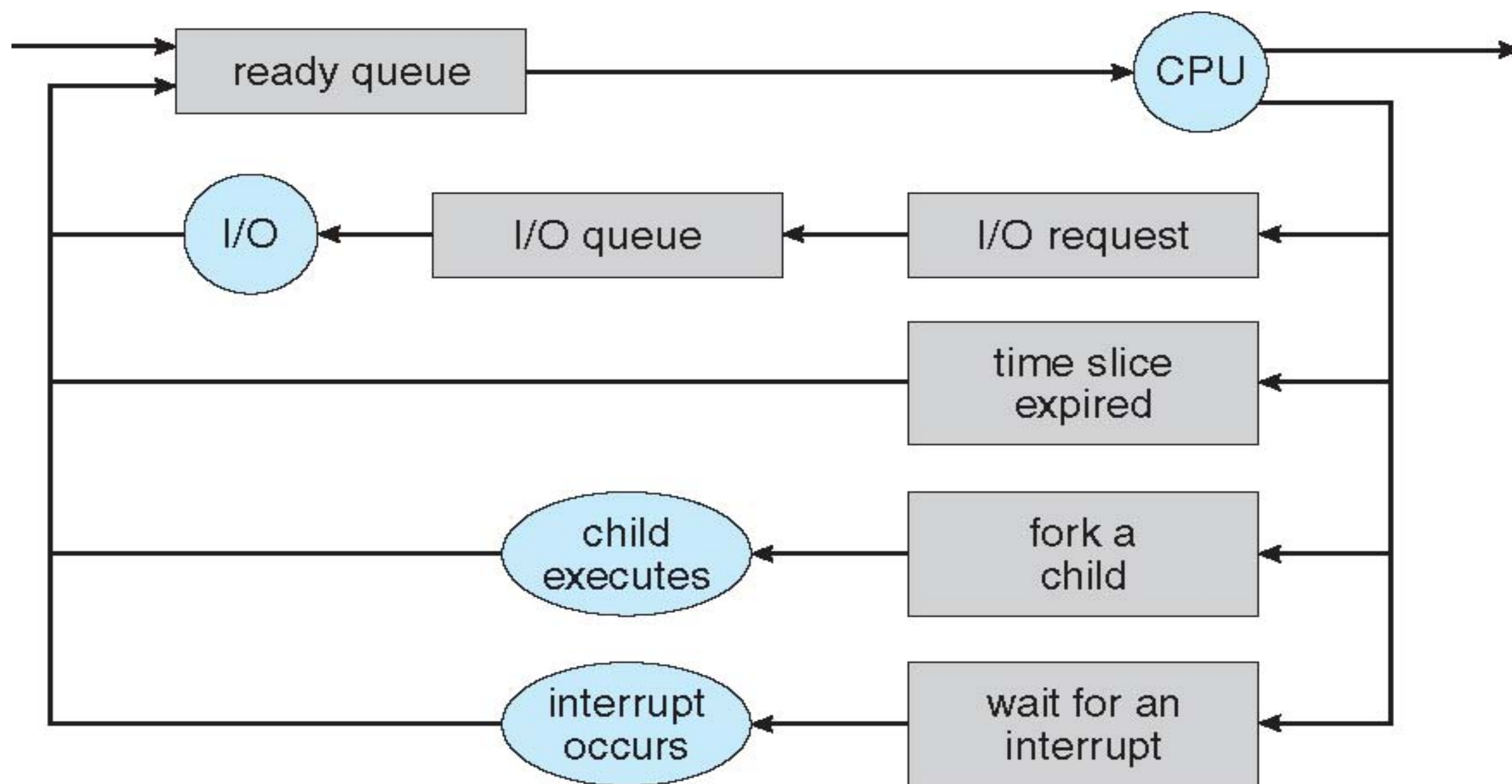
- **Want to maximize CPU use**
  - Quickly switch processes onto CPU for time sharing
- **Process scheduler selects among available processes for next execution on CPU**
- **Maintains scheduling queues of processes**
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of 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



# Representation of Process Scheduling



# Schedulers

---

- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is **invoked very infrequently** (seconds, minutes)  $\Rightarrow$  (may be slow)
  - The long-term scheduler controls the degree of multiprogramming (i.e. the number of processes in memory)
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is **invoked very frequently** (milliseconds)  $\Rightarrow$  (must be fast)

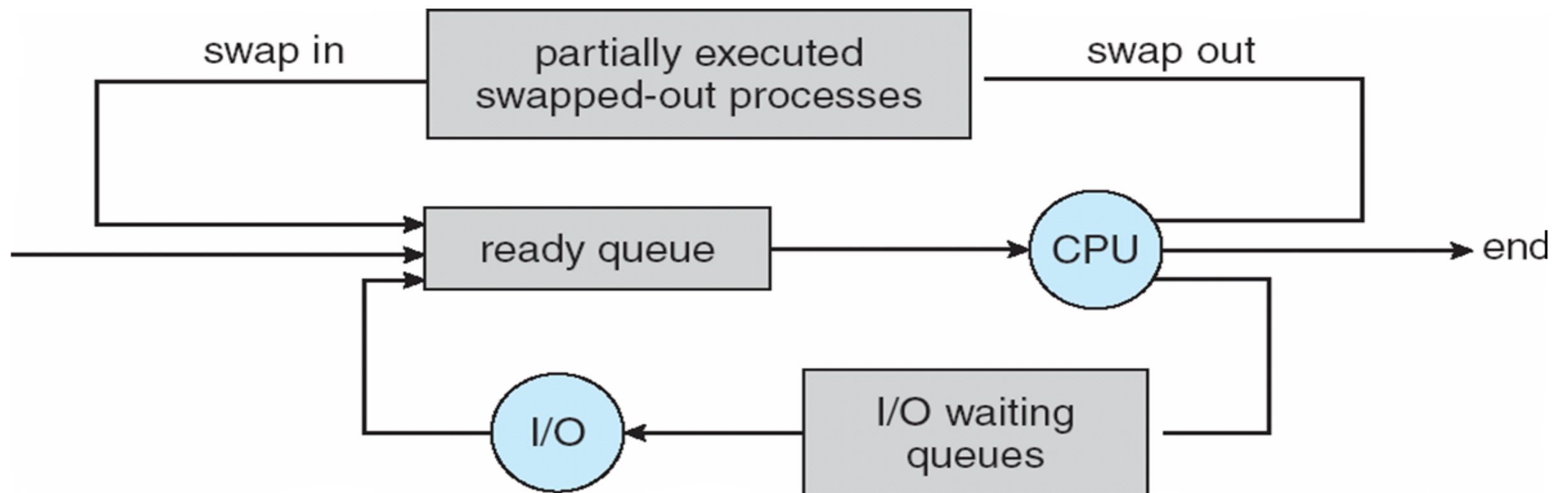
## Schedulers (Cont.)

---

- **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
- **Desirable to achieve a balance of I/O bound processes and CPU-bound processes**

# Addition of Medium Term Scheduling

- Sometimes it is useful to remove processes from memory and return them to memory later
  - This is called **swapping**



# 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 (e.g. Intel's hyperthreading)

# Process Creation

---

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

# Process Creation (Cont.)

---

- **Address space options**

- Child duplicate of parent
- Child has a program loaded into it

- **UNIX examples**

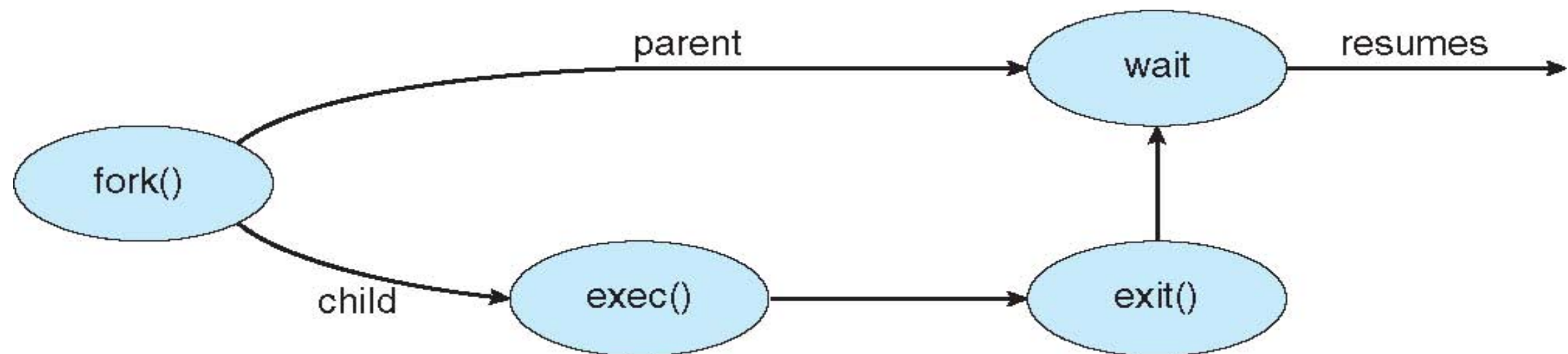
- `fork ( )` system call creates new process
- `exec ( )` system call used after a fork to replace the process' memory space with a new program



# Process Creation

---

- Parent waits for child to exit before resuming



# 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");

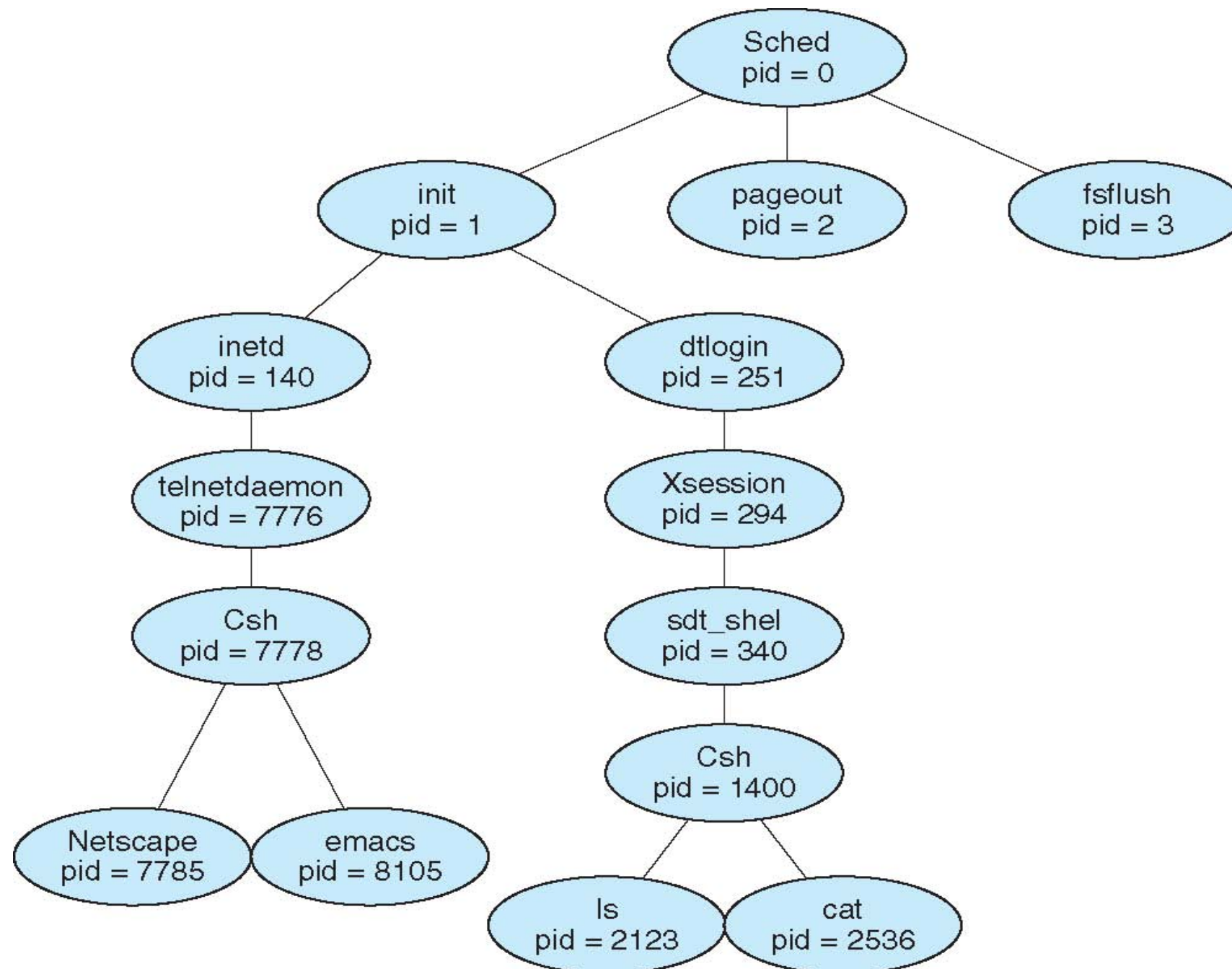
    }

    return 0;

}
```

# A Tree of Processes on Solaris

---



# Process Termination Options

---

- **Process executes last statement and asks the operating system to delete it (exit)**
  - Termination code output 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