OPERATING SYSTEMS

Introduction to Process

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

中山大學
SUN YAT-SEN UNIVERSITY

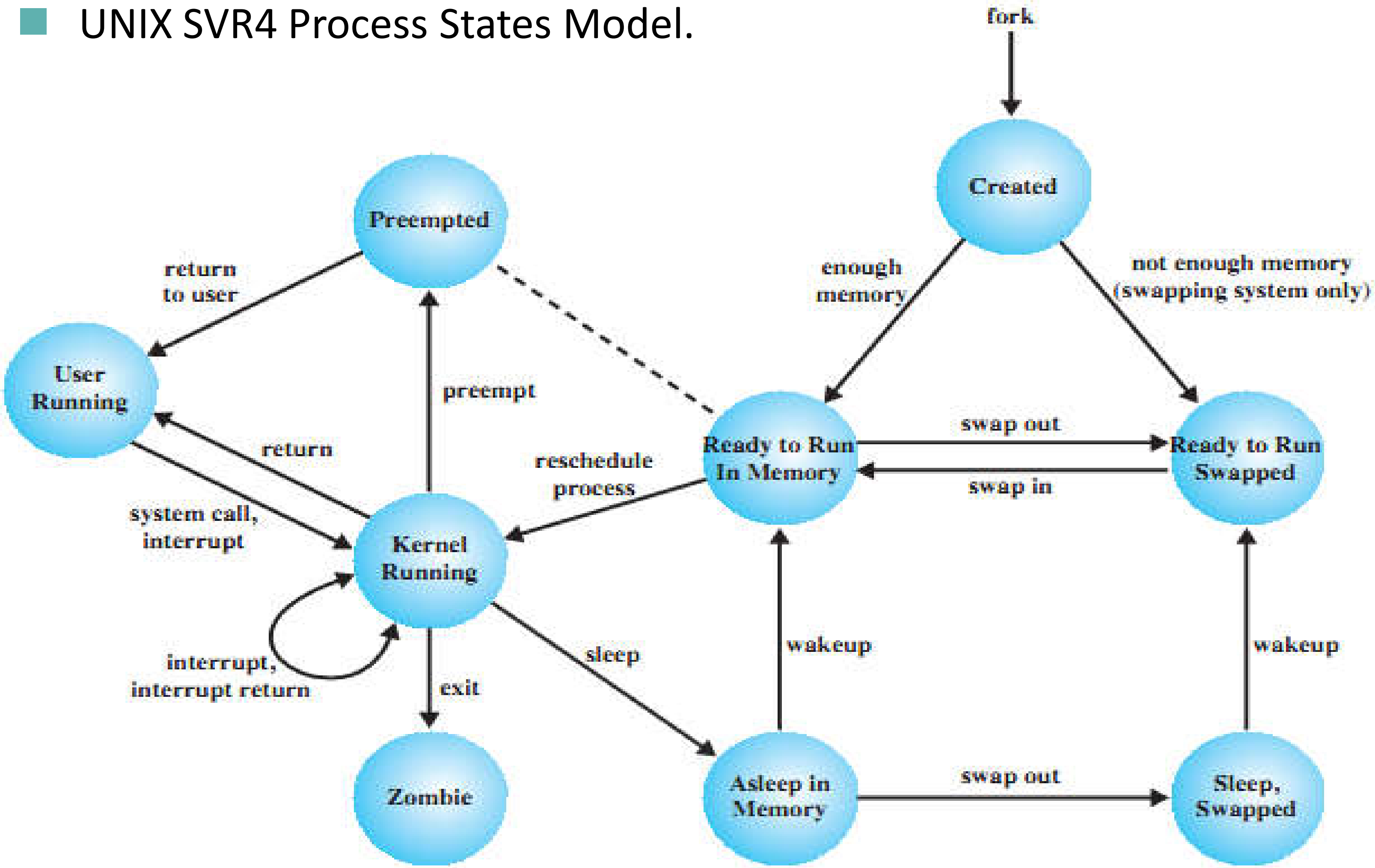
Contents

UNIX SVR4 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.

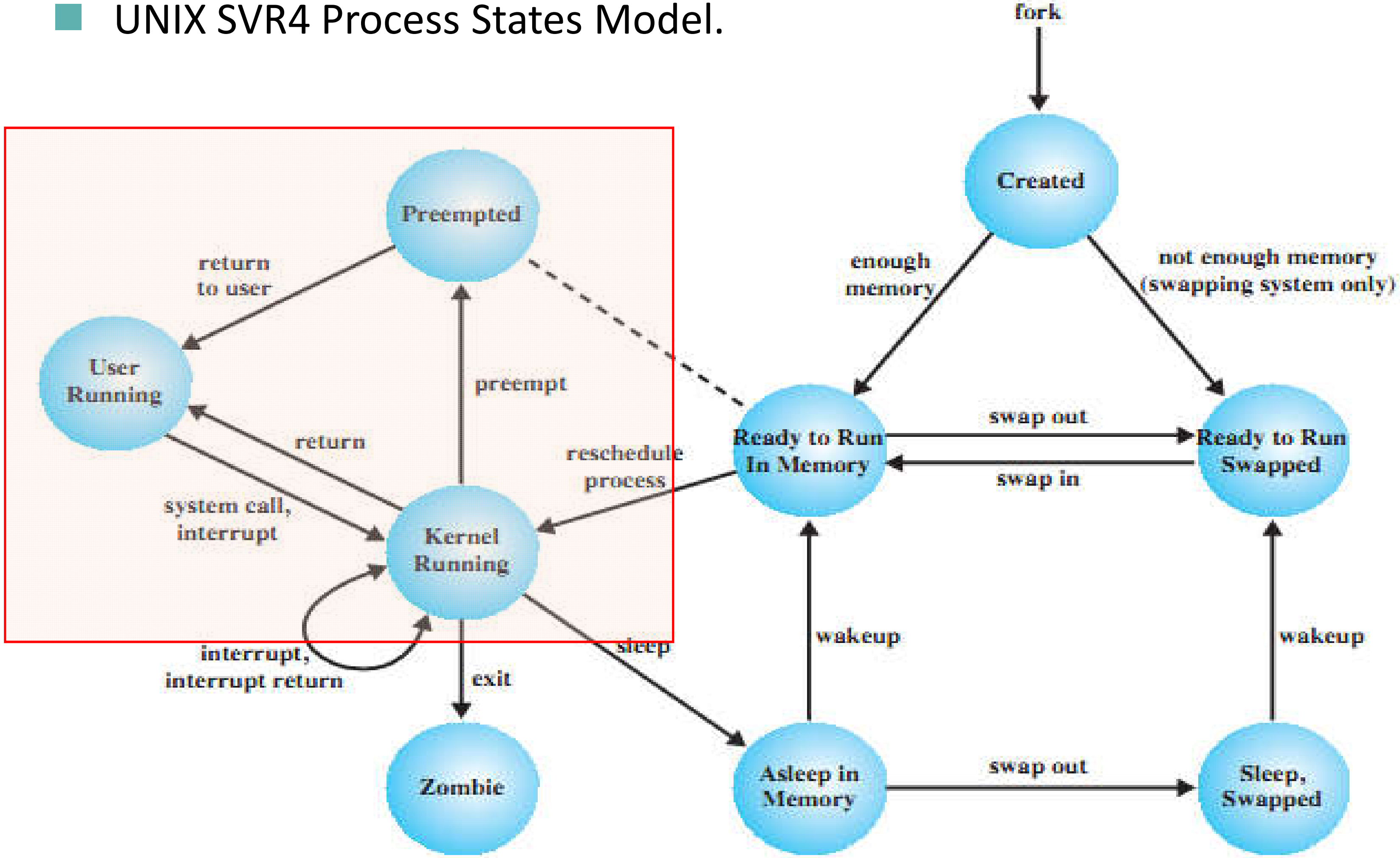
UNIX SVR4 Process States

- UNIX SVR4 Process States Model.



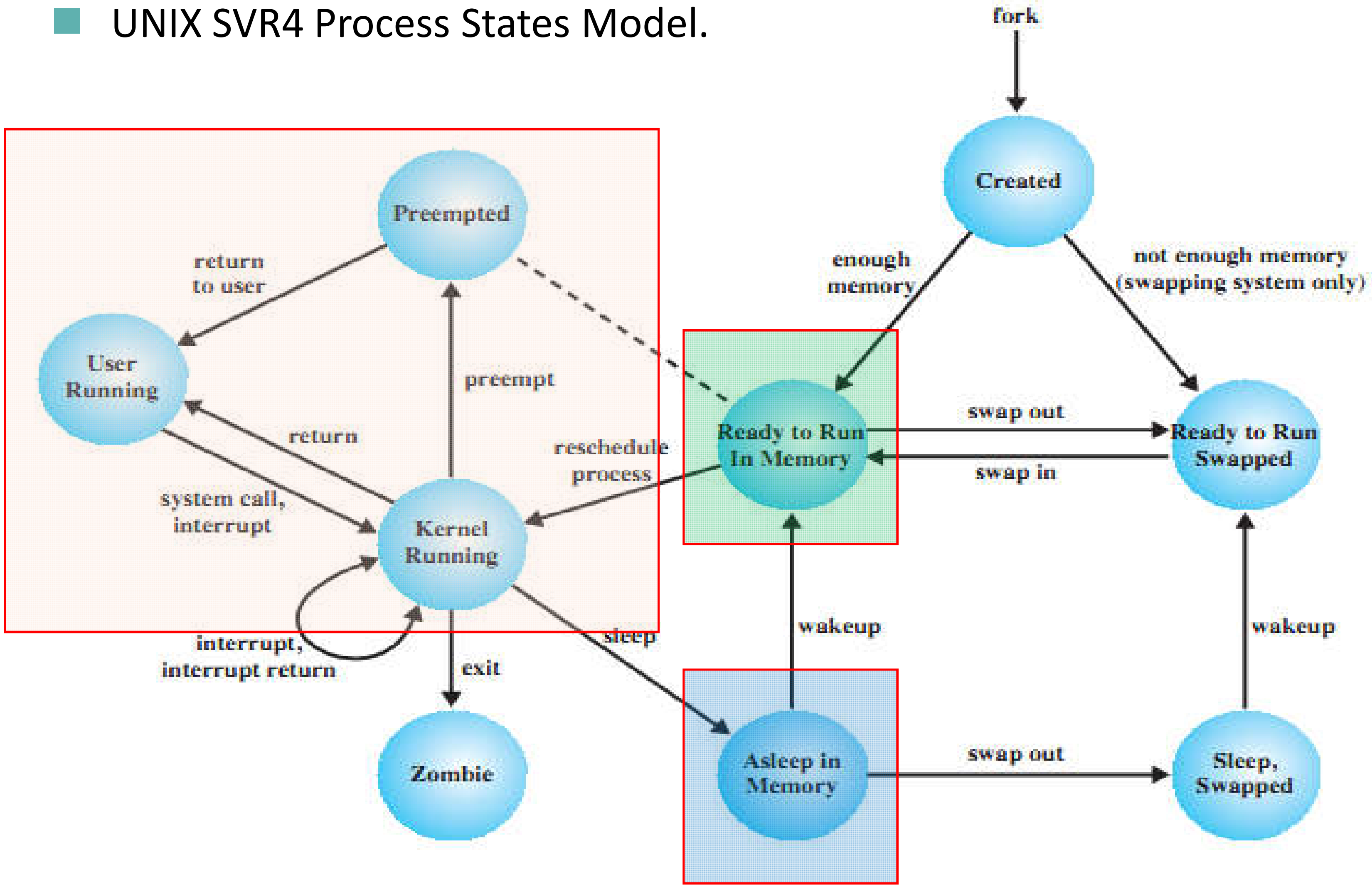
UNIX SVR4 Process States

- UNIX SVR4 Process States Model.



UNIX SVR4 Process States

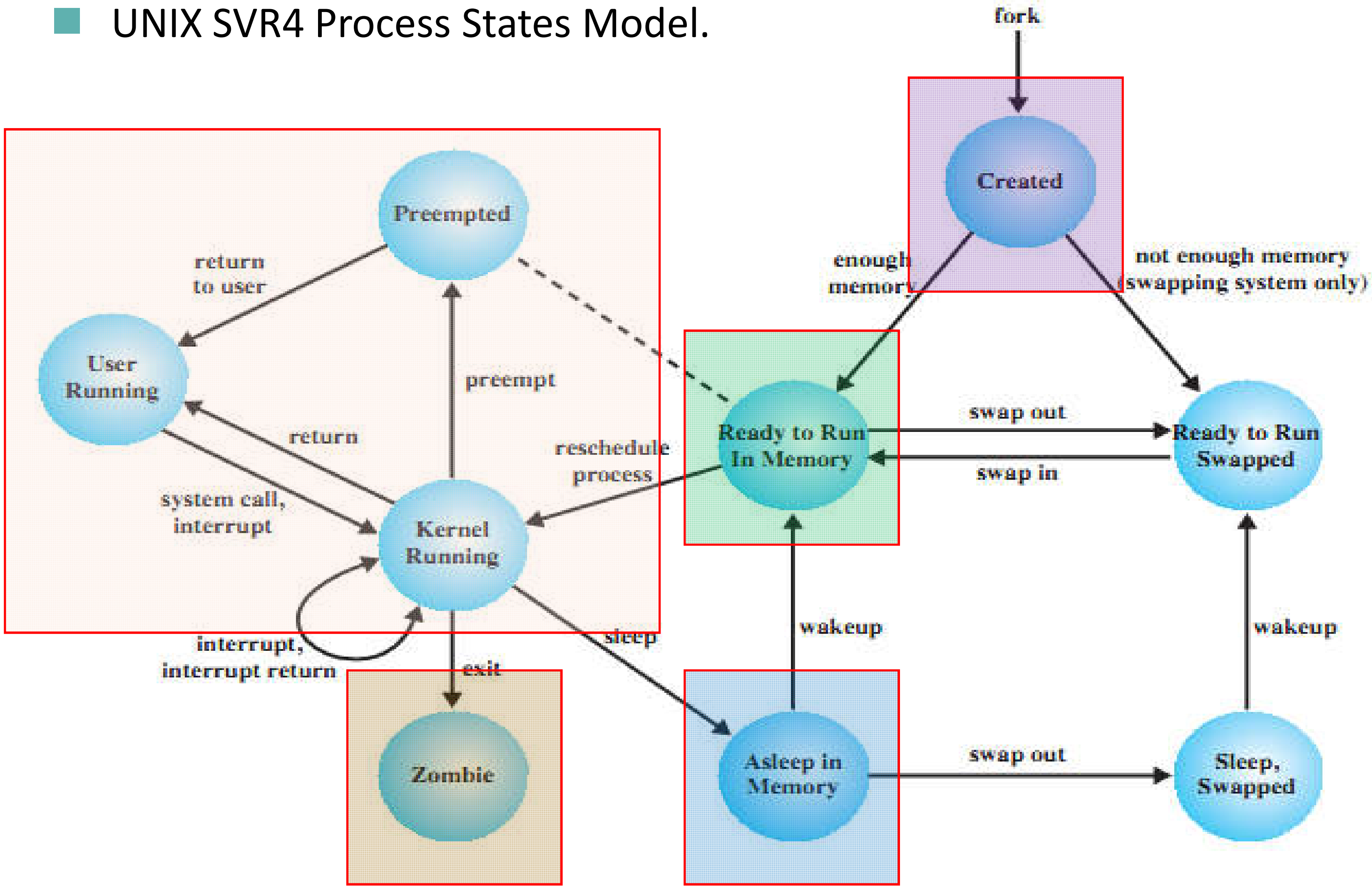
- UNIX SVR4 Process States Model.



Three-state Transition

UNIX SVR4 Process States

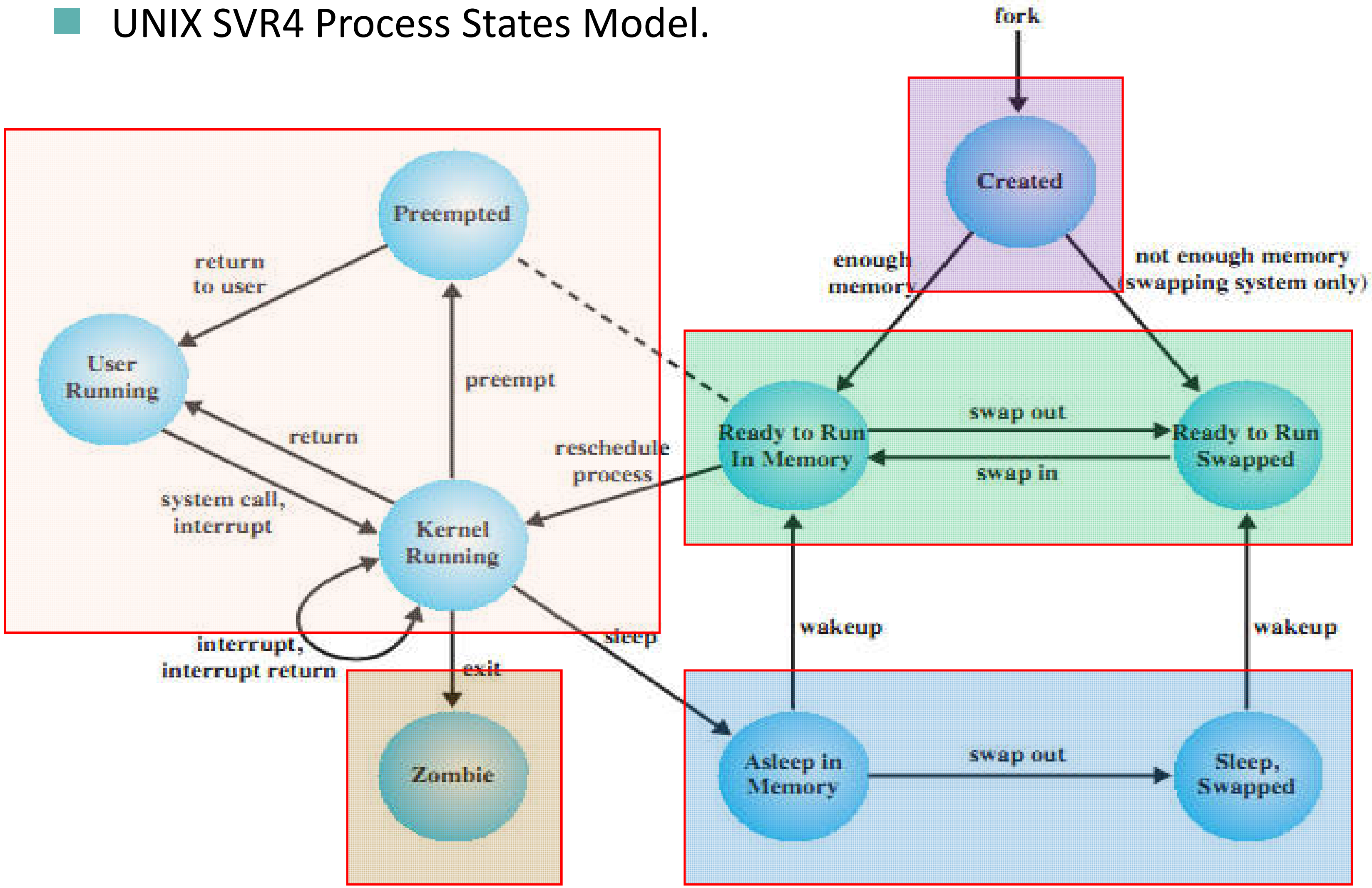
- UNIX SVR4 Process States Model.



Five-state Transition

UNIX SVR4 Process States

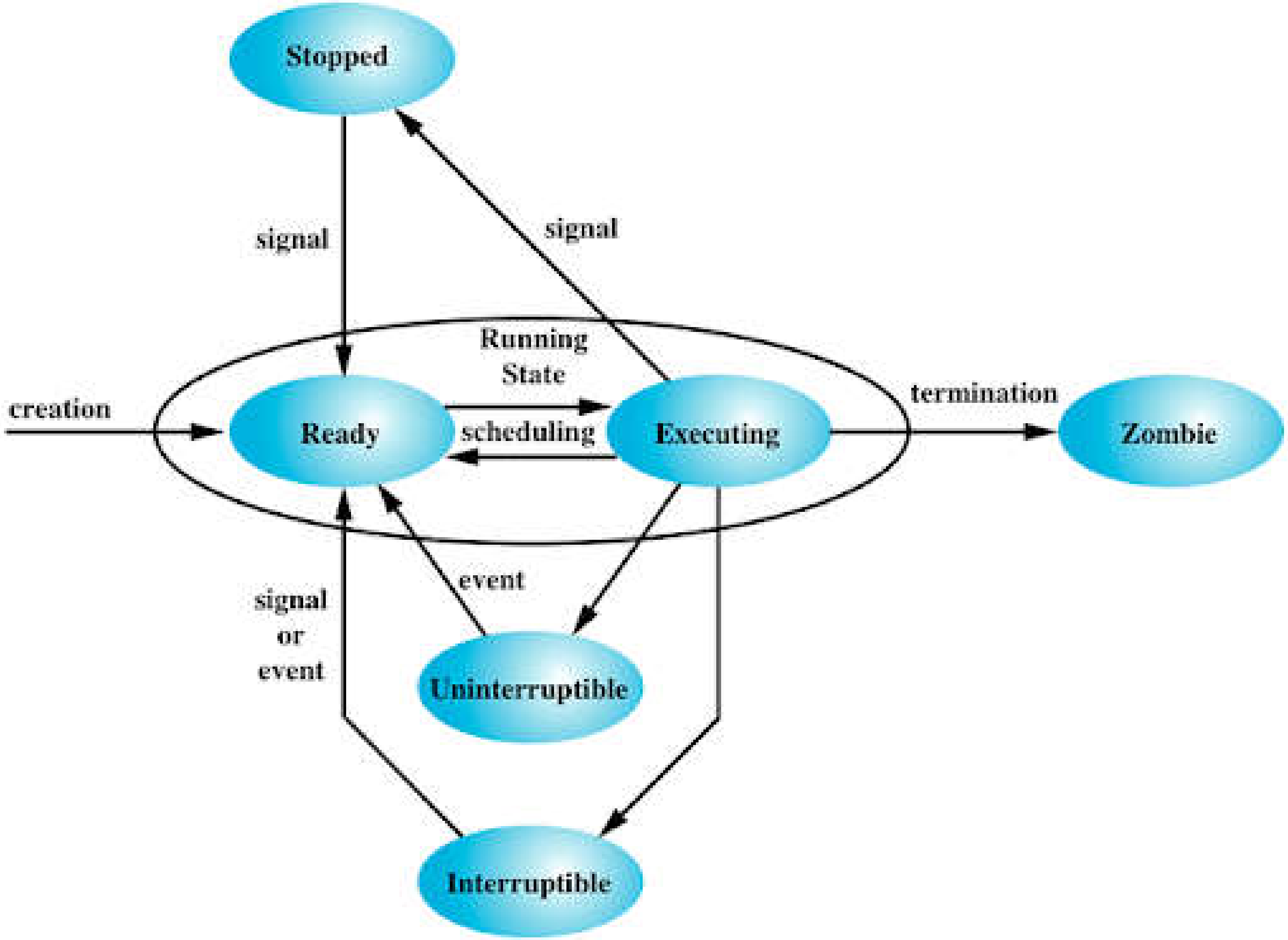
- UNIX SVR4 Process States Model.



Five-state Transition with Swapping

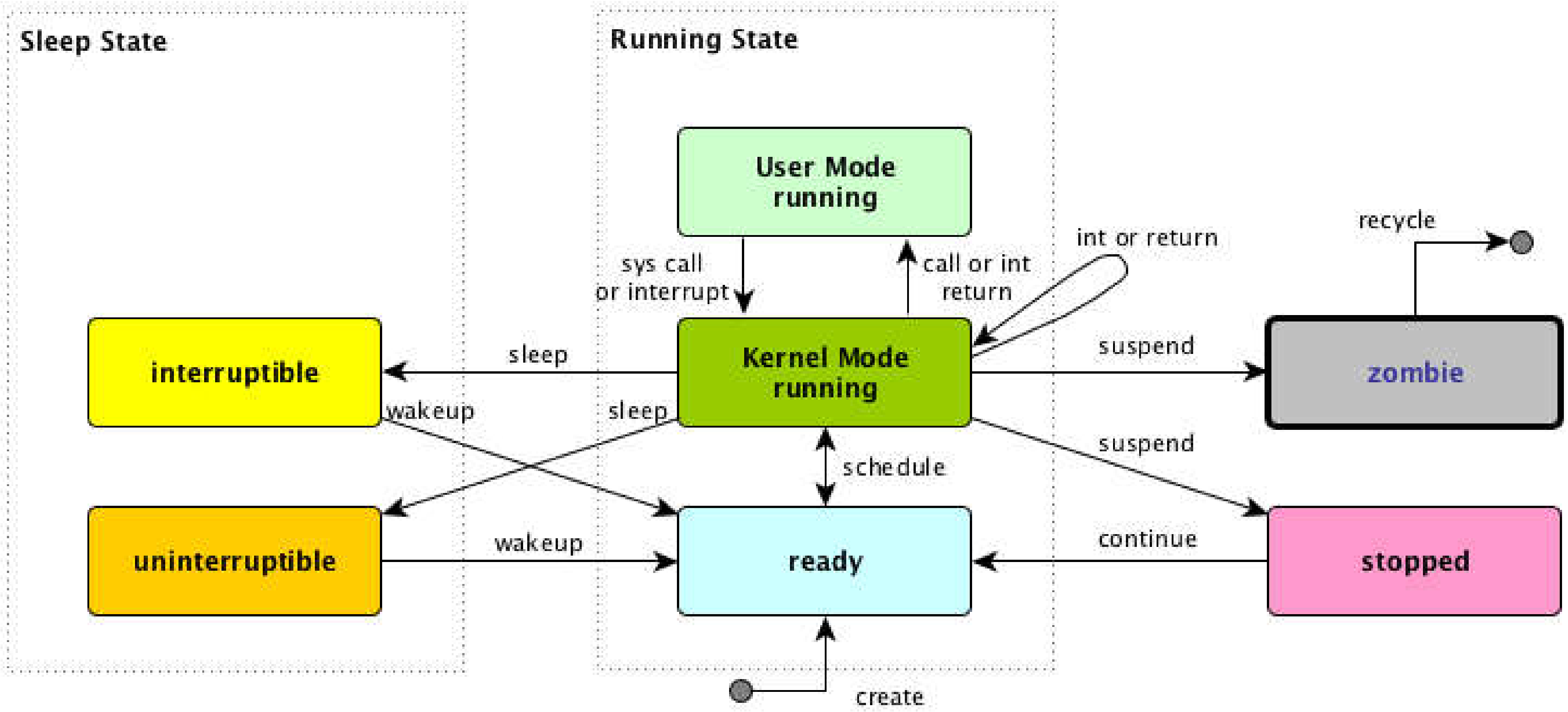
Process Representation in Linux

- Linux Process State.



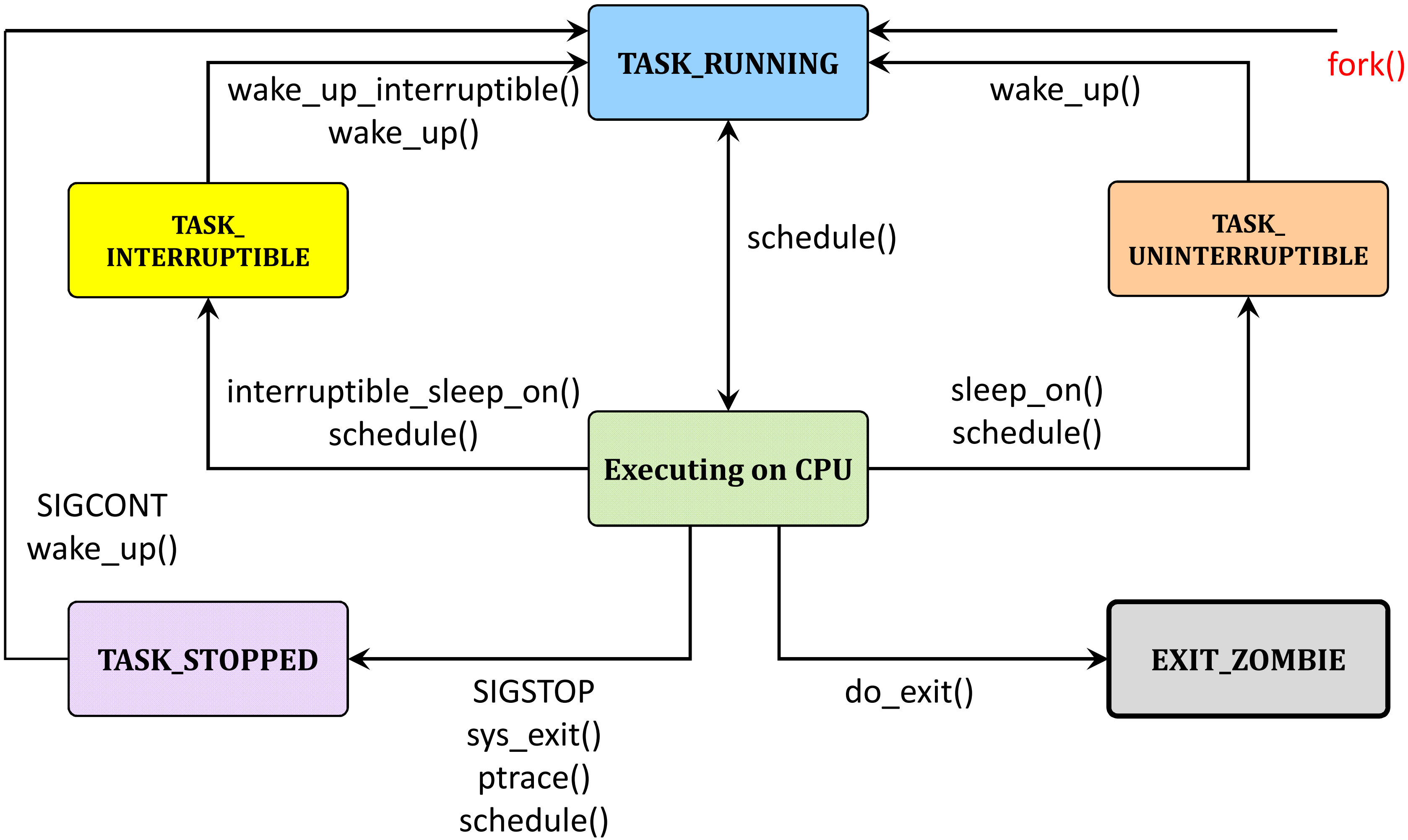
Process Representation in Linux

- Linux Process State.



Process Representation in Linux

- Linux Process State.

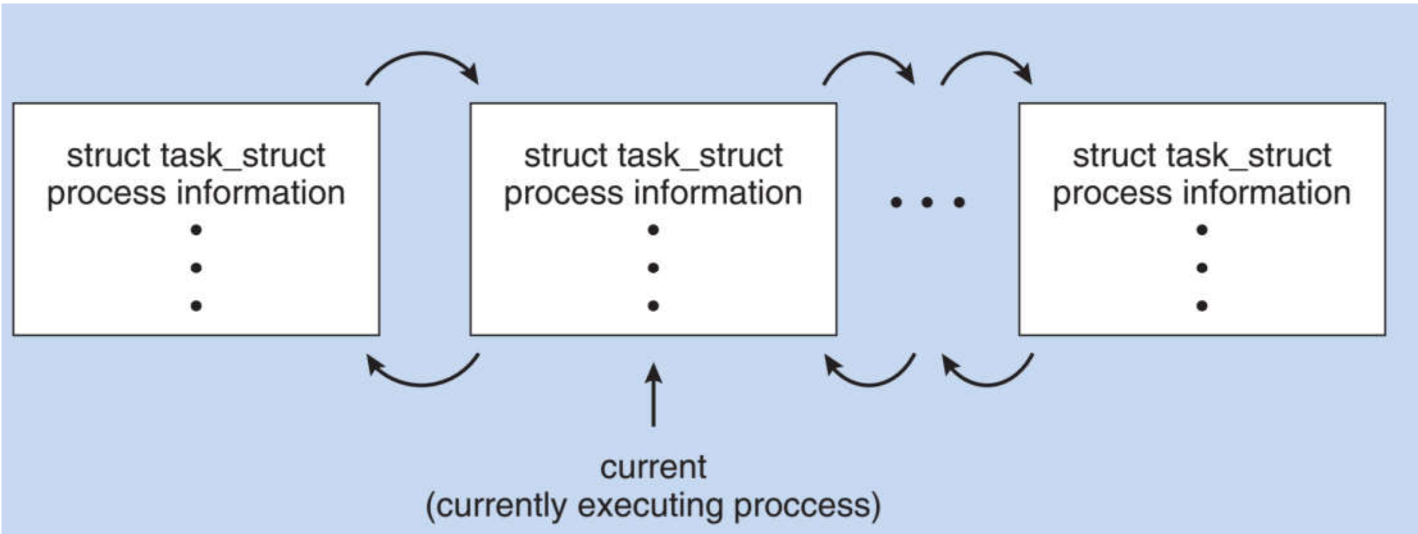


Process Representation in Linux

- The PCB in Linux is represented by the C structure task_struct included in <linux/sched.h>.
  - task_struct is declared in linux/sched.h.

```
$ sudo apt-get install linux-source
$ tar …
$ vim /usr/src/linux-source-X.XX.X/include/linux/sched.h
```

- The Linux kernel uses a *circular doubly-linked list* of struct task_struct to store these process descriptors.



Process Representation in Linux

- task_struct contains all the necessary information for representing a process, including
  - PID
  - the state of the process
  - processor registers
  - scheduling and memory-management information
  - list of open files
  - pointers to the process's parent and a list of its children and siblings.
- Some of these fields include:

```
long state; /* state of the process */
struct sched_entity se; /* 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 this process */
```

Process Representation in Linux

- Here are a few fields from kernel 2.6.15-1.2054_FC5, starting at line 701: (what about these fields in your current Ubuntu version?)

```
701.struct task_struct {
702.    volatile long state; /*-1 unrunnable,0 runnable,>0 stopped */
703.    struct thread_info *thread_info;
          . . .
767.      /* PID/PID hash table linkage. */
768.    struct pid pids[PIDTYPE_MAX];
          . . .
798.    char comm[TASK_COMM_LEN]; /* executable name excluding path
```

- Type definition of volatile long.

```
#define TASK_RUNNING 0
#define TASK_INTERRUPTIBLE 1
#define TASK_UNINTERRUPTIBLE 2
#define TASK_STOPPED 4
#define TASK_TRACED 8 /* in tsk->exit_state */
#define EXIT_ZOMBIE 16
#define EXIT_DEAD 32 /* in tsk->state_again */
#define TASK_NONINTERACTIVE 64
```

Overview

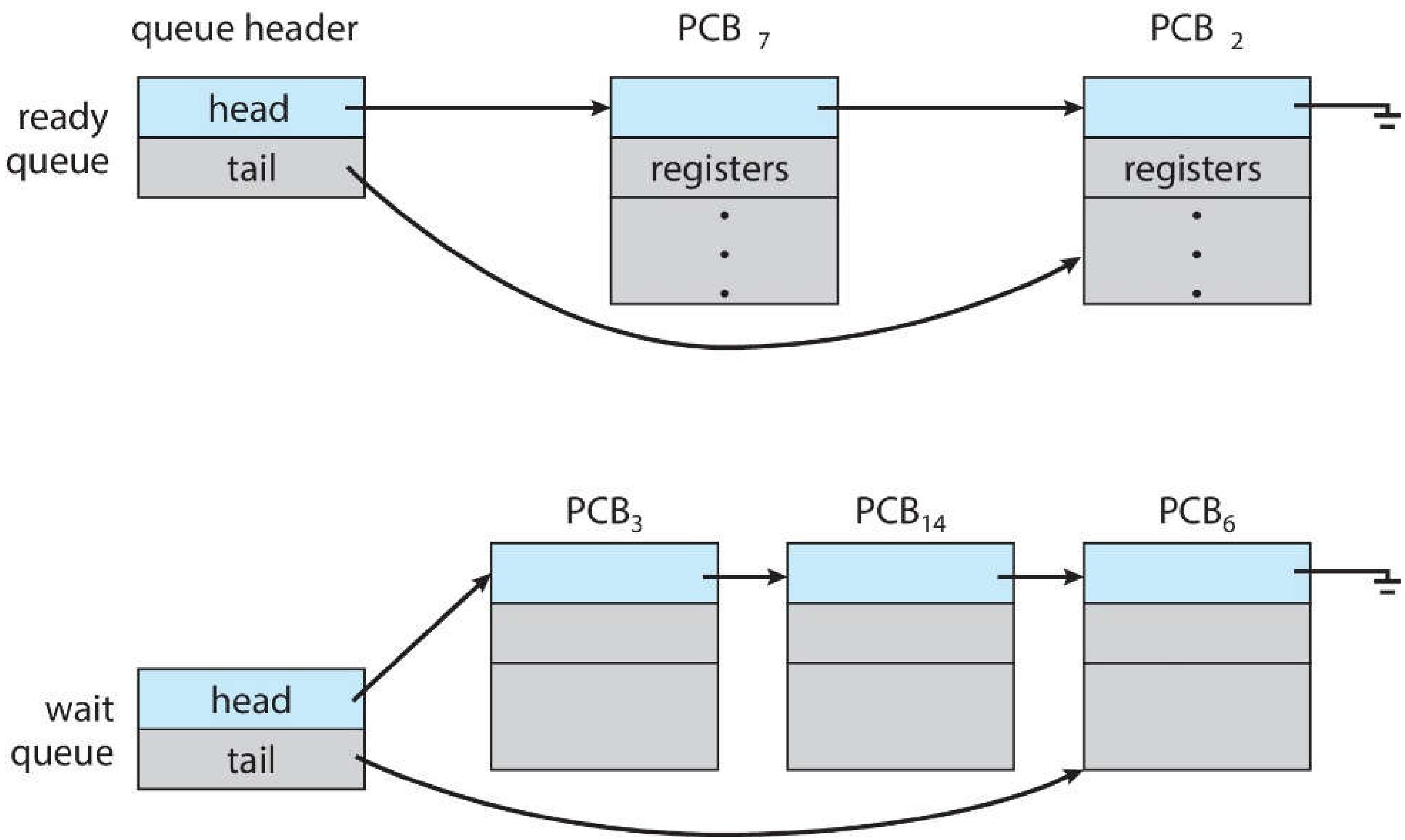
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
  - The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- *Process scheduler* selects among available processes for next execution on CPU.
  - For a single-processor system, there will be only one running process and the rest will have to wait until the CPU is free and can be rescheduled.

Overview

- *Scheduling queues* of processes are to be maintained:
  - 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/Waiting Queues
    - set of processes waiting for an I/O device
- Processes *migrate* among the various queues.

Overview

- The Ready and wait Queues.



Overview

- Most processes can be described as either I/O-bound or CPU-bound:
  - I/O-bound process
    - spends more time doing I/O than computations
      - short CPU bursts
  - CPU-bound process
    - spends more time doing computations
      - long CPU bursts

Types of Process Schedulers

- There are three types/levels of Process Schedulers
  - Long-term Scheduler
    - High-level Scheduler, or
    - Jobs Scheduler.
  - Medium-term Scheduler
    - Medium-level Scheduler,
    - Swapping Scheduler, or
    - Emergency Scheduler.
  - Short-term Scheduler
    - Low-level Scheduler,
    - CPU Scheduler,
    - Micro Scheduler, or
    - Process/Thread Scheduler on narrow sense (狭义上的进程/线程调度).

Long-term Schedulers

- Long-term process scheduler selects which programs/processes should be brought into the *ready queue*.
  - determines which programs are admitted to the system for processing
  - controls the degree of multiprogramming
  - strives for good process mix of I/O-bound and CPU-bound processes
  - Long-term scheduler is invoked infrequently.
    - seconds, minutes
    - may be slow
- If more processes are admitted for processing:
  - less likely that all processes will be blocked
    - bringing better CPU usage
  - each process has less fraction of the CPU time

Short-term Schedulers

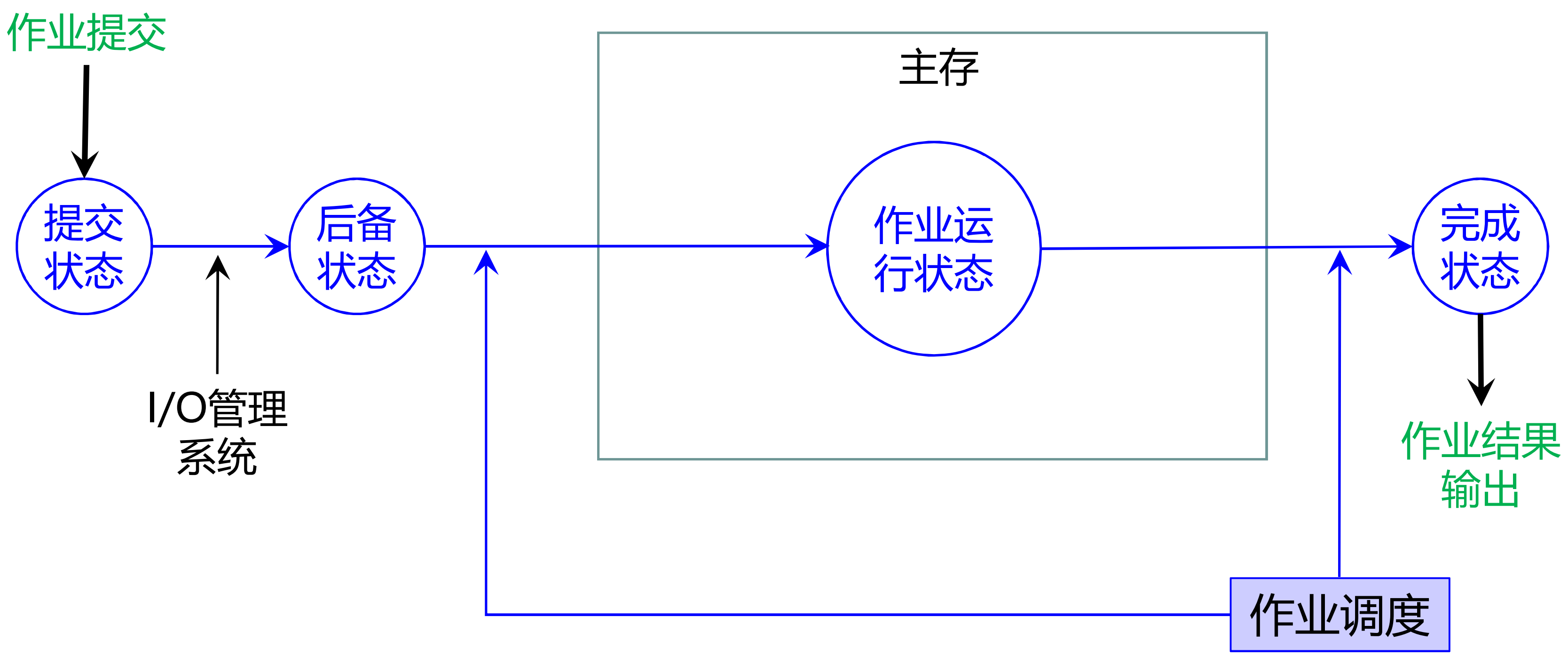
- Short-term process scheduler selects which process should be *executed* next and allocates CPU — also called CPU scheduling (处理机调度).
  - CPU scheduling determines which process is going to execute next according to a scheduling algorithm.
  - Short-term scheduler is also known as the *dispatcher* (which is part of it) that moves the processor from one process to another, and prevents a single process from monopolizing (独占) processor time.
  - Short-term scheduler is invoked on an event that may lead to choose another process for execution:
    - Clock interrupts
    - I/O interrupts
    - Operating system calls and traps
    - Signals.
  - Short-term scheduler is invoked very frequently
    - milliseconds
    - must be fast.

Medium-term Schedulers

- Medium-term process scheduler selects which job/process should be *swapped* out if system is overloaded.
  - 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.
  - OS may need to swap out some processes to disk, and then later swap them back in.
  - Swapping decisions is based on the need of multiprogramming management.

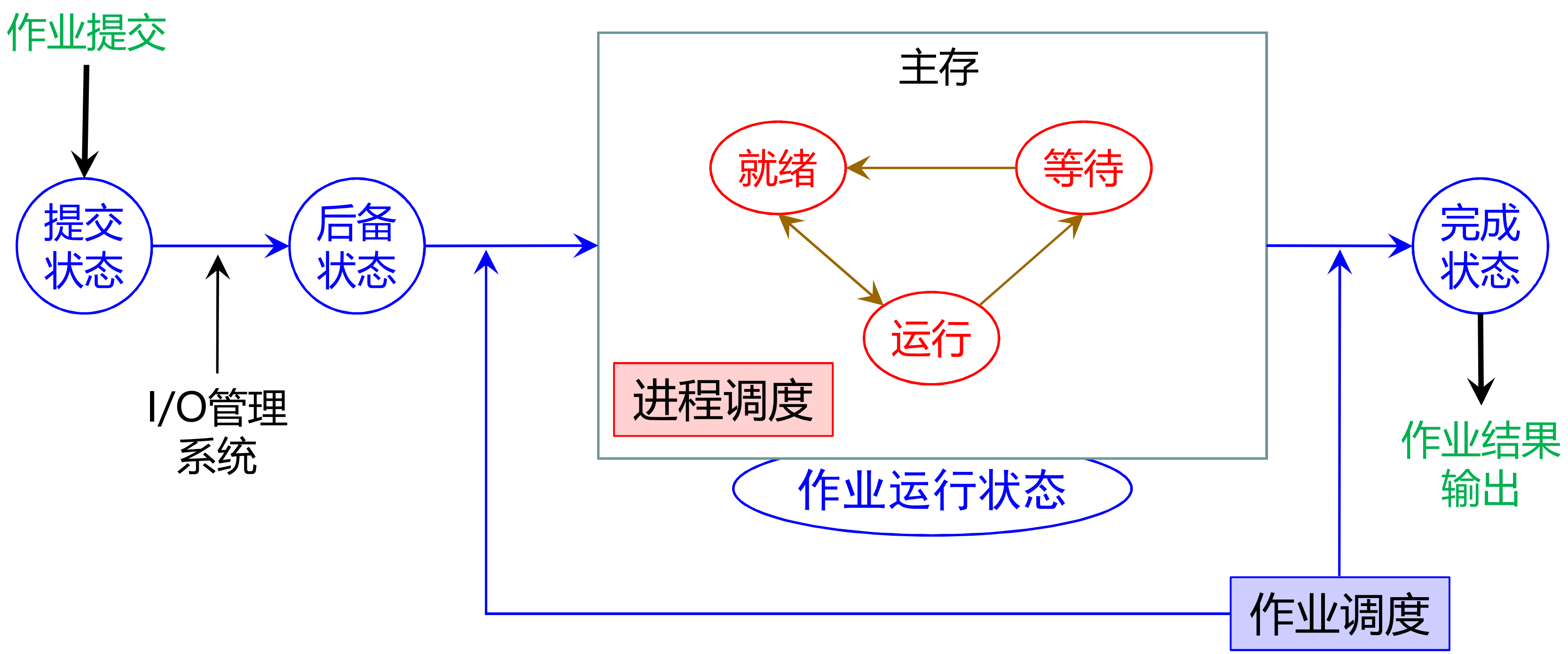
Schematic View of Schedulers

- Job scheduling, Swapping scheduling and Process/Thread scheduling.



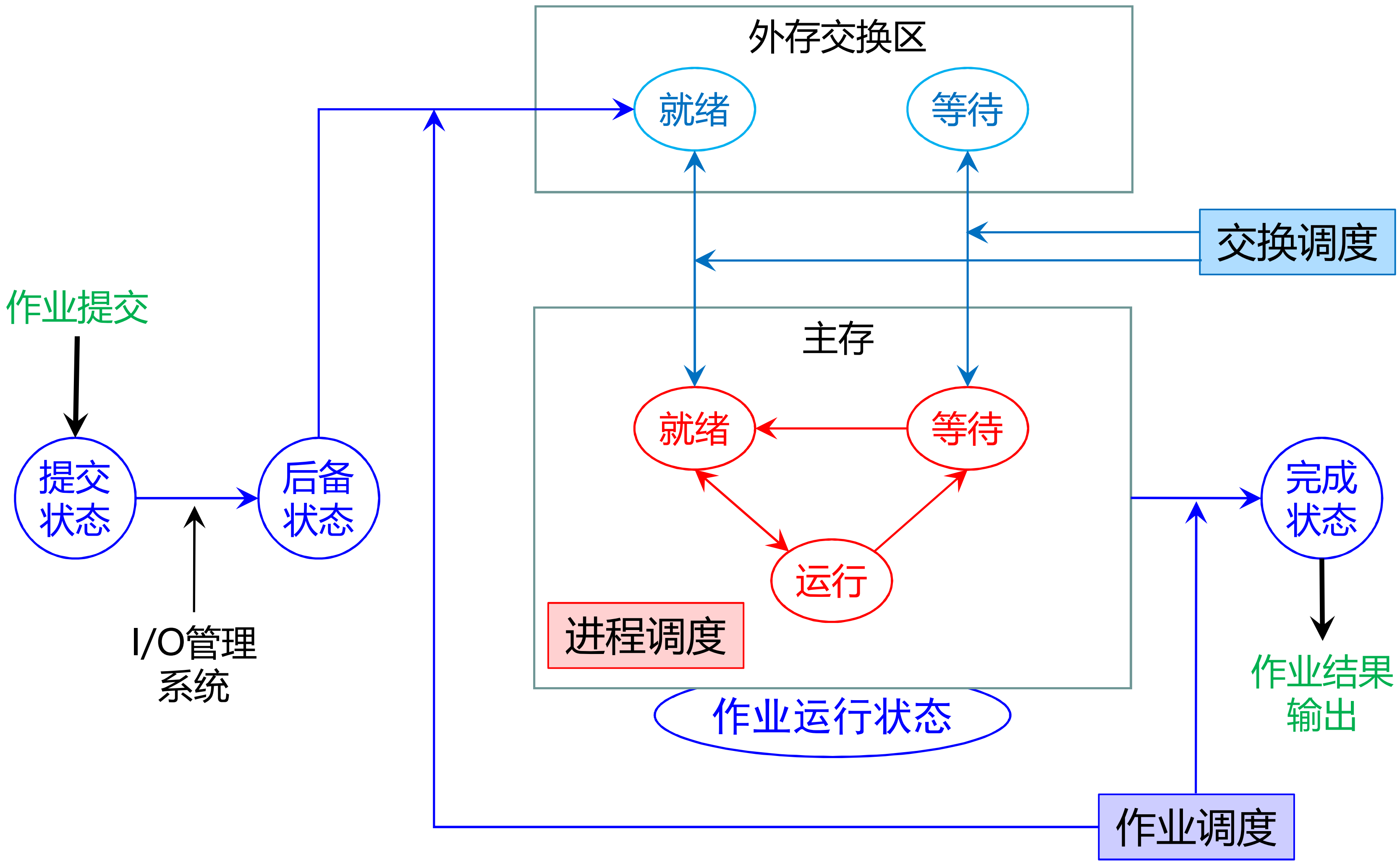
Schematic View of Schedulers

- Job scheduling, Swapping scheduling and Process/Thread scheduling.



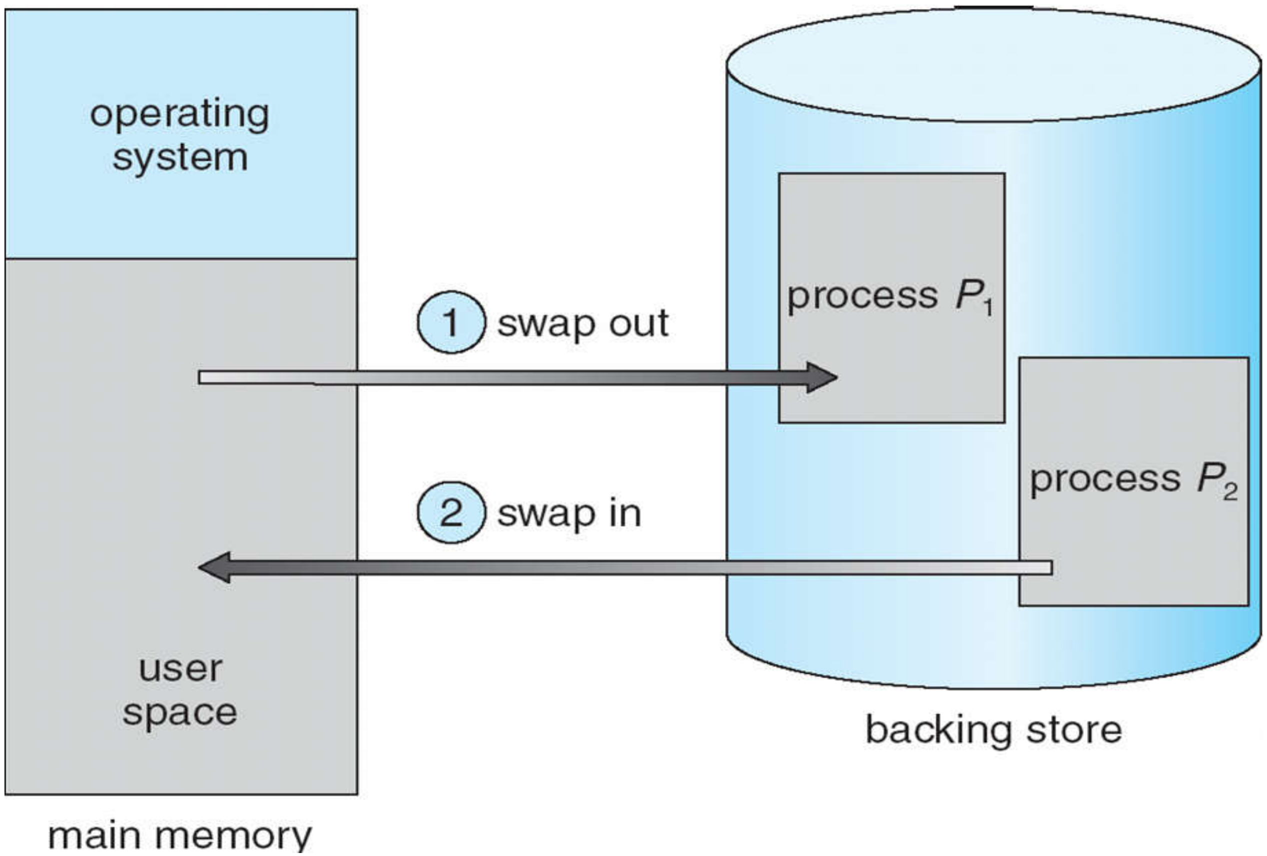
Schematic View of Schedulers

- Job scheduling, Swapping scheduling and Process/Thread scheduling.



Process Swapping

- Schematic View of Swapping.

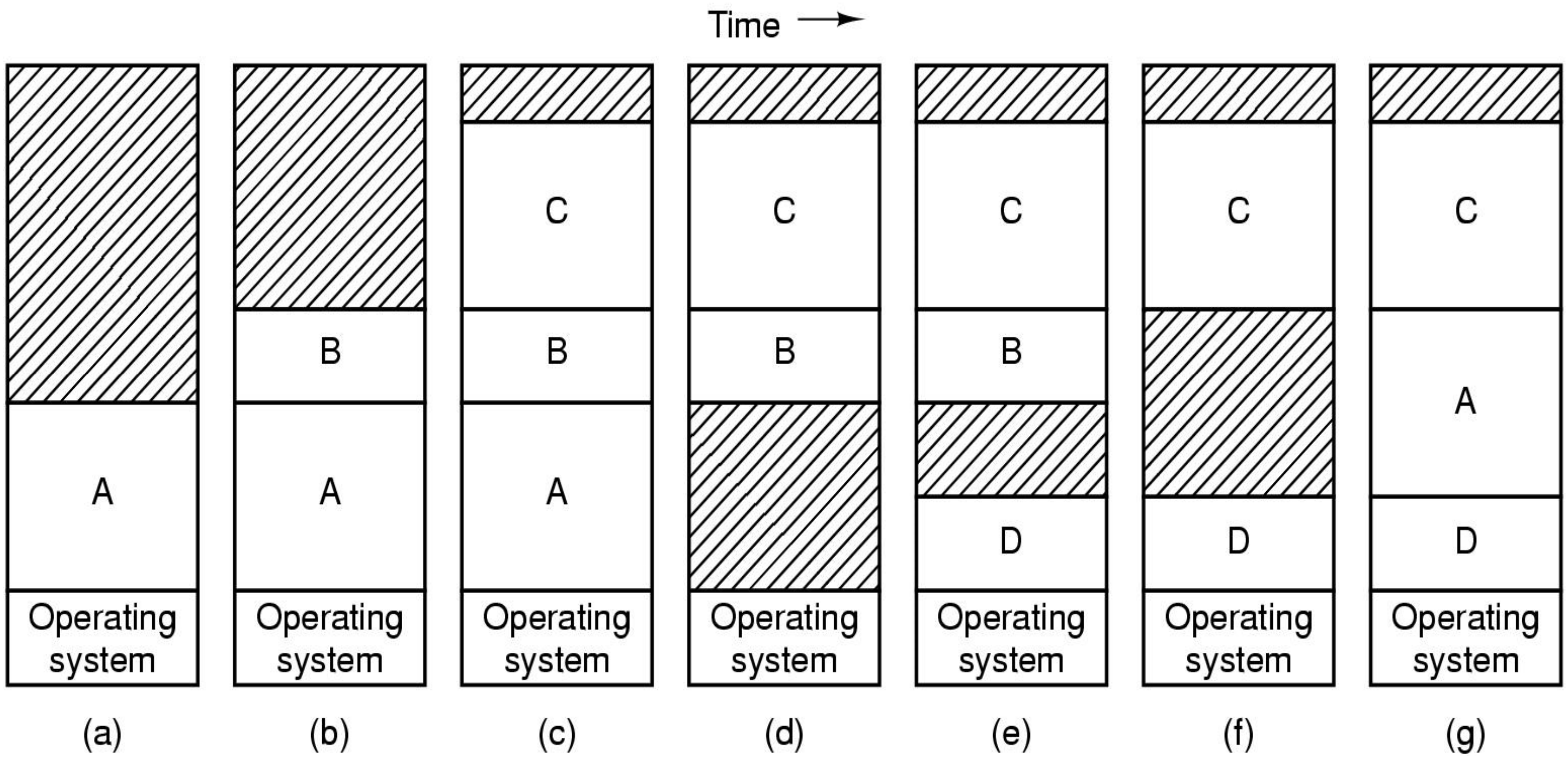


Process Swapping

- Dynamics of Swapping
  - A process can be swapped temporarily out of memory to a backing storage, and then brought back into memory for continued execution.
  - Backing storage
    - 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。
    - e.g., UNIX, Linux, and Windows.
  - System maintains a ready queue of ready-to-run processes which have memory images on disk.

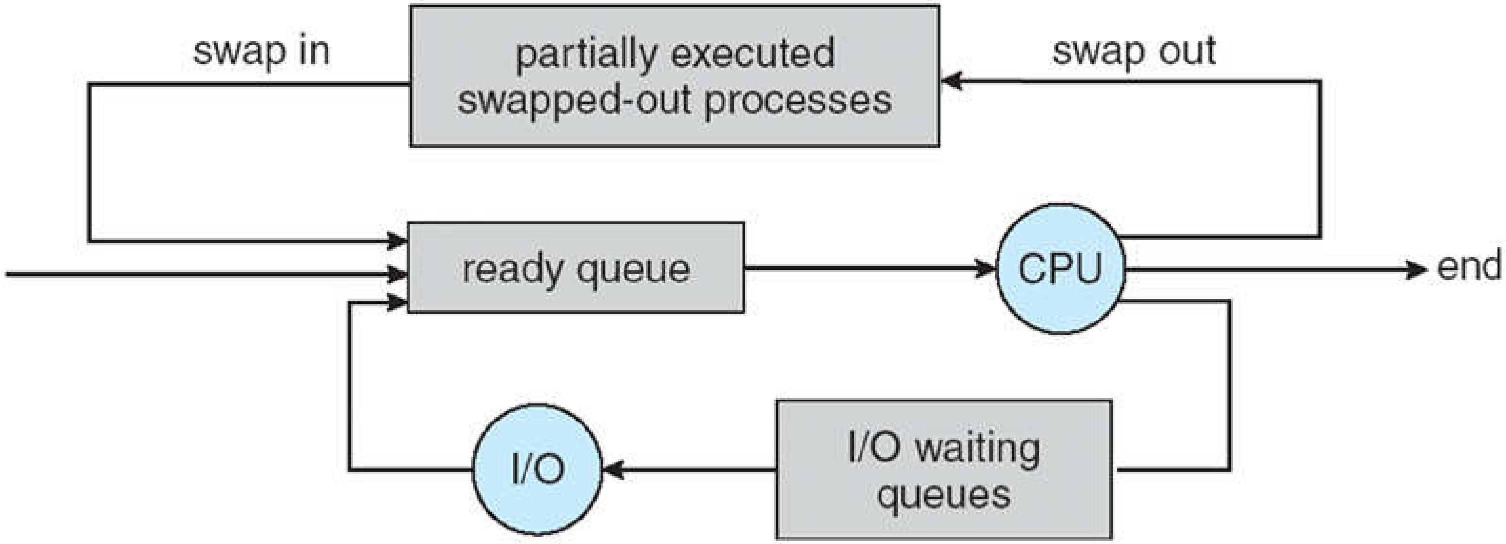
Process Swapping

- Swapping Example.



Process Swapping

- Addition of Medium-term Scheduling.

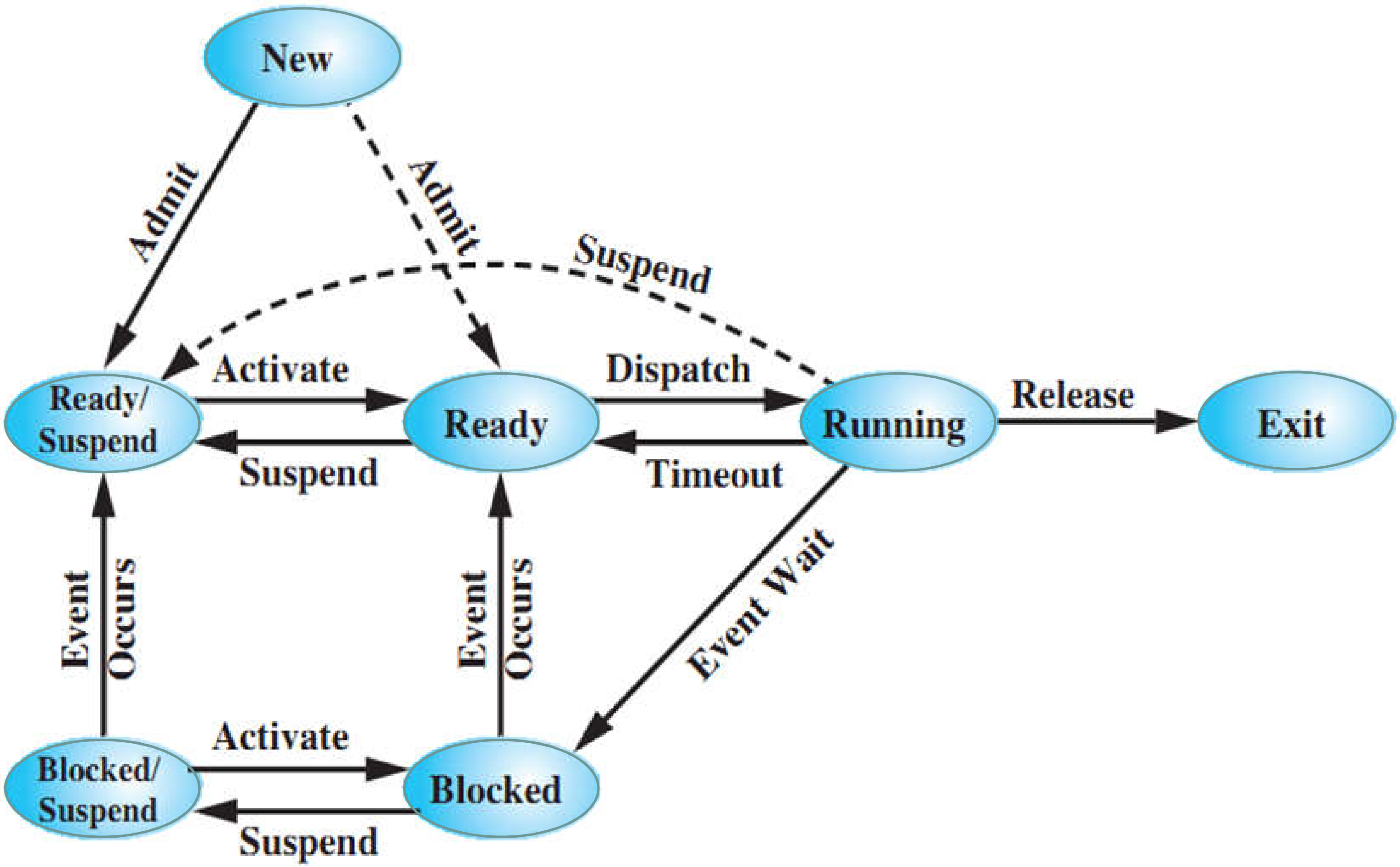


Process Swapping

- Support of Swapping
  - OS may need to suspend some processes, i.e., to swap them out to disk and then swap them back in.
  - Two new states might need to be added:
    - Blocked Suspend: blocked processes which have been swapped out to disk
    - Ready Suspend: ready processes which have been swapped out to disk

Process Scheduling Activities

- A Seven-state Process Model.

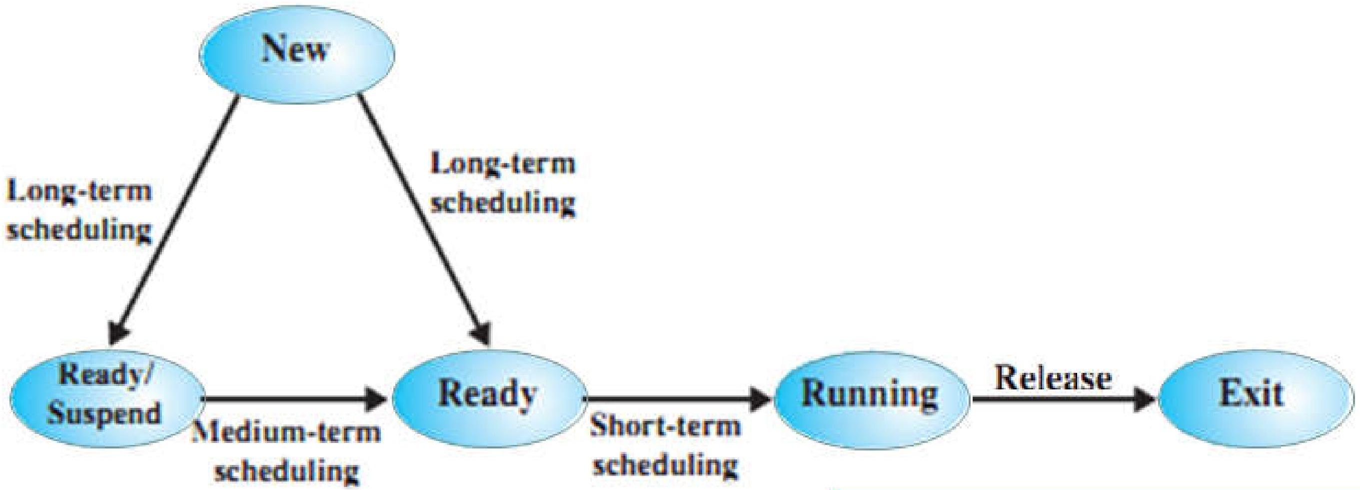


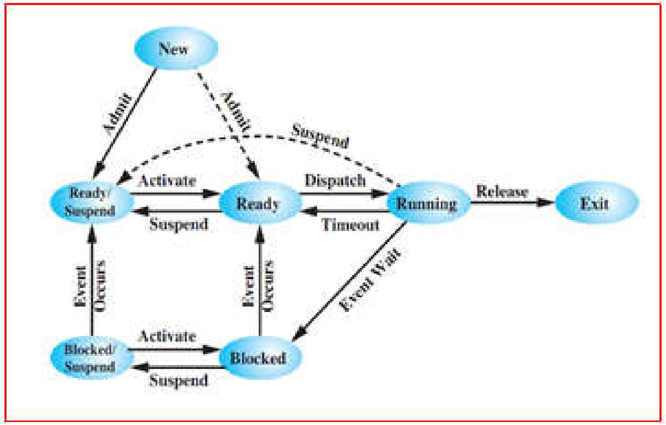
Process Scheduling Activities

- A Seven-state Process Model
  - Additional state transitions
    - Blocked $\rightarrow$ Blocked Suspend
      - When all processes are blocked, the OS will make room to bring a ready process in memory.
    - Blocked Suspend $\rightarrow$ Ready Suspend
      - when the event for which it has been waiting occurs (state info is available to OS).
    - Ready Suspend $\rightarrow$ Ready
      - when no more ready processes in main memory
    - Ready $\rightarrow$ Ready Suspend (unlikely)
      - when there are no blocked processes and memory must to be freed for adequate performance

Process Scheduling Activities

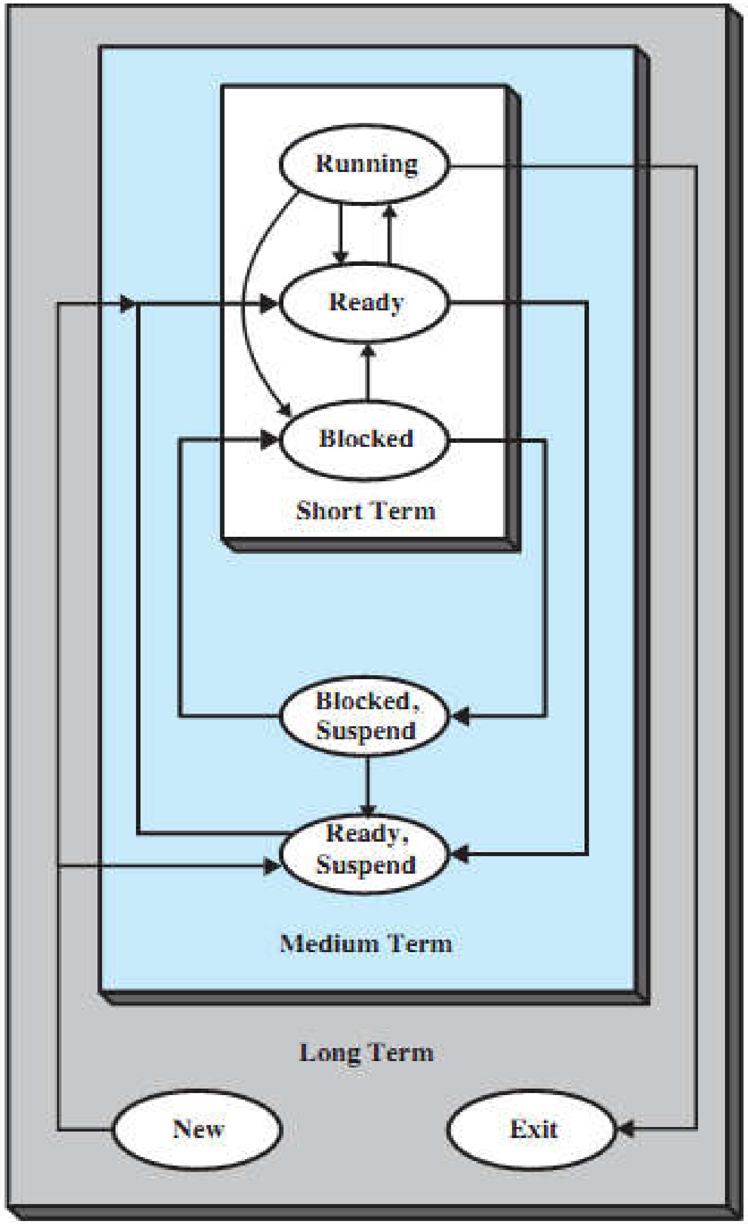
- A Seven-state Process Model
  - Scheduling Levels and Scheduling Activities.





Process Scheduling Activities

- A Seven-state Process Model.



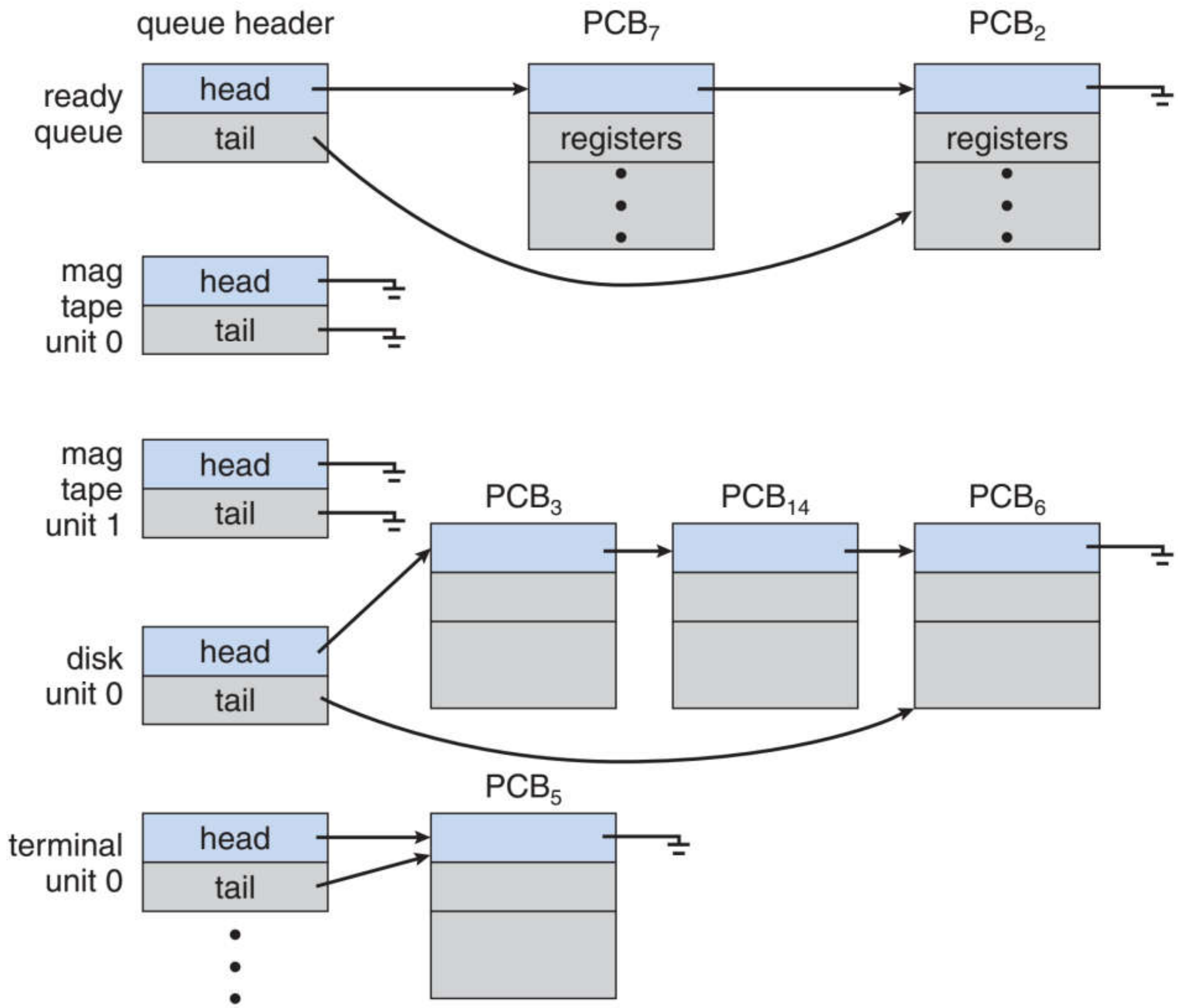
- Another view of the three levels of scheduling.

Process Scheduling Queues

- Job Queue, Ready Queue and Device Queue
  - As processes enter the system, they are put into a *job queue*, which consists of all processes in the system.
  - The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the *ready queue*.
    - The ready queue is generally stored as a linked list. The header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.
  - The list of processes waiting for a particular I/O device is called a *device queue*. Each device has its own device queue.
    - Suppose A process makes an I/O request to a shared device such as a disk, but the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk.
    - A device queue is not a queue of devices, but a queue of process waiting for a particular I/O device.

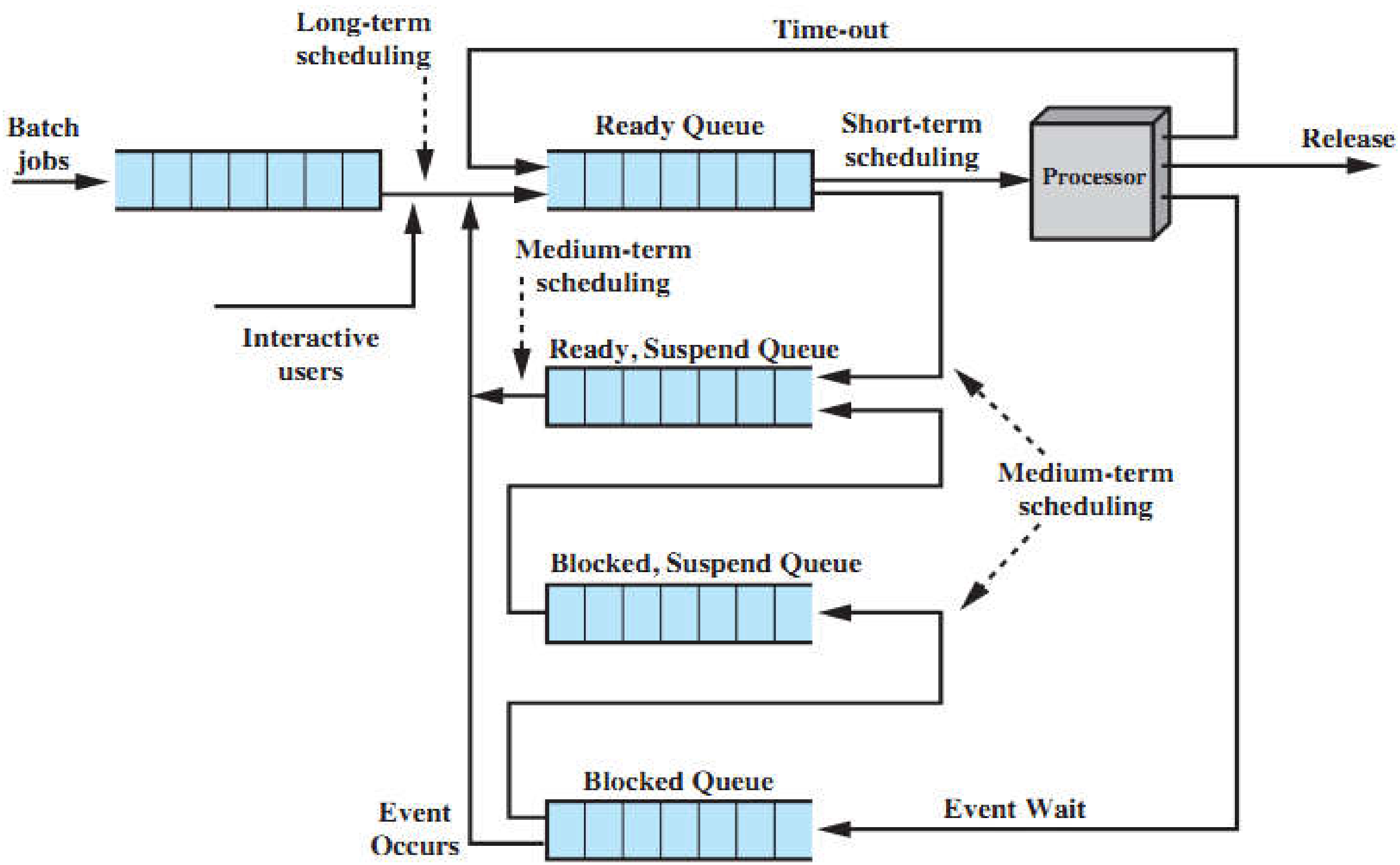
Process Scheduling Queues

- The ready queue and various I/O device queues.



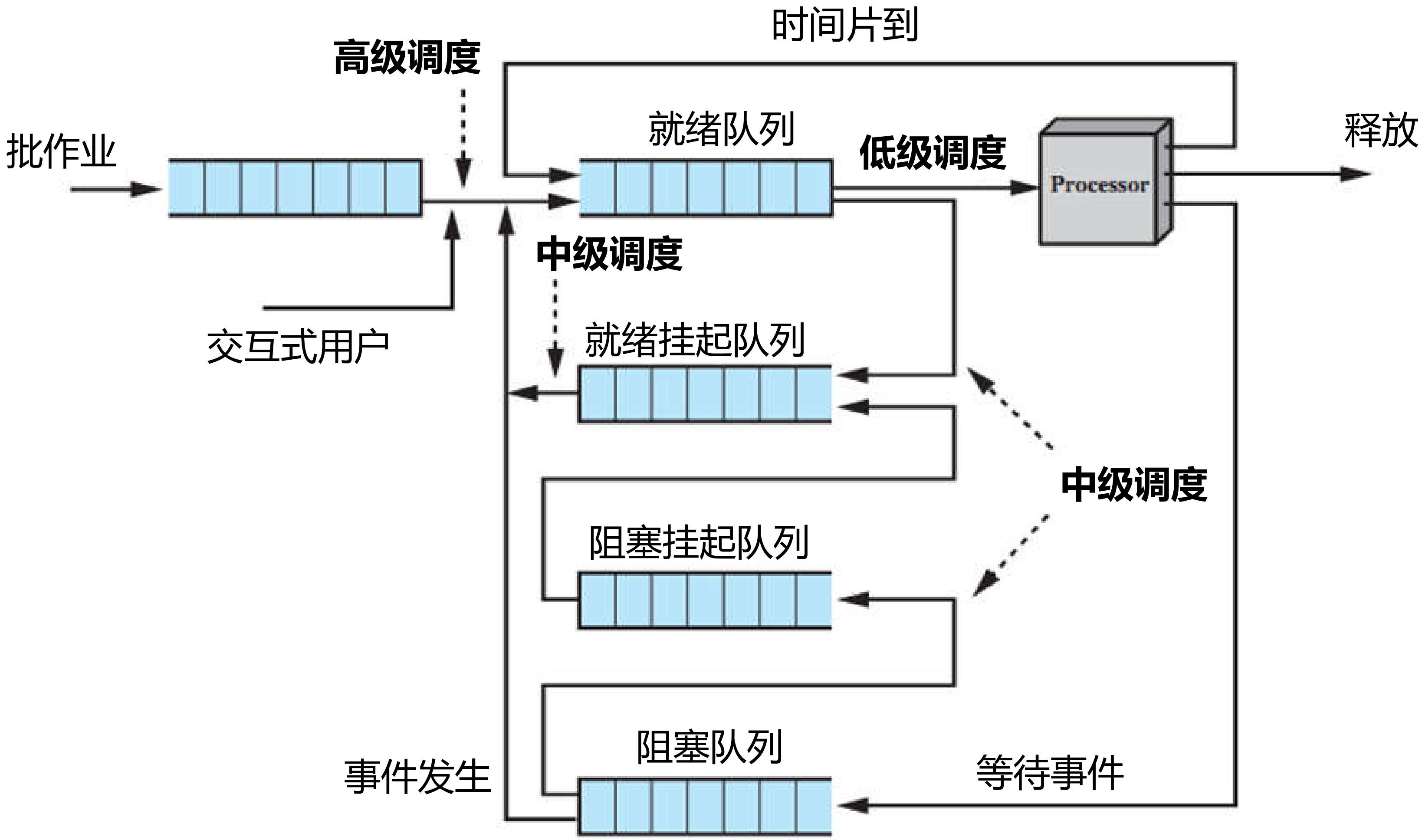
Process Scheduling Queues

- Queuing diagram for process scheduling
  - Processes migrate among the various queues.



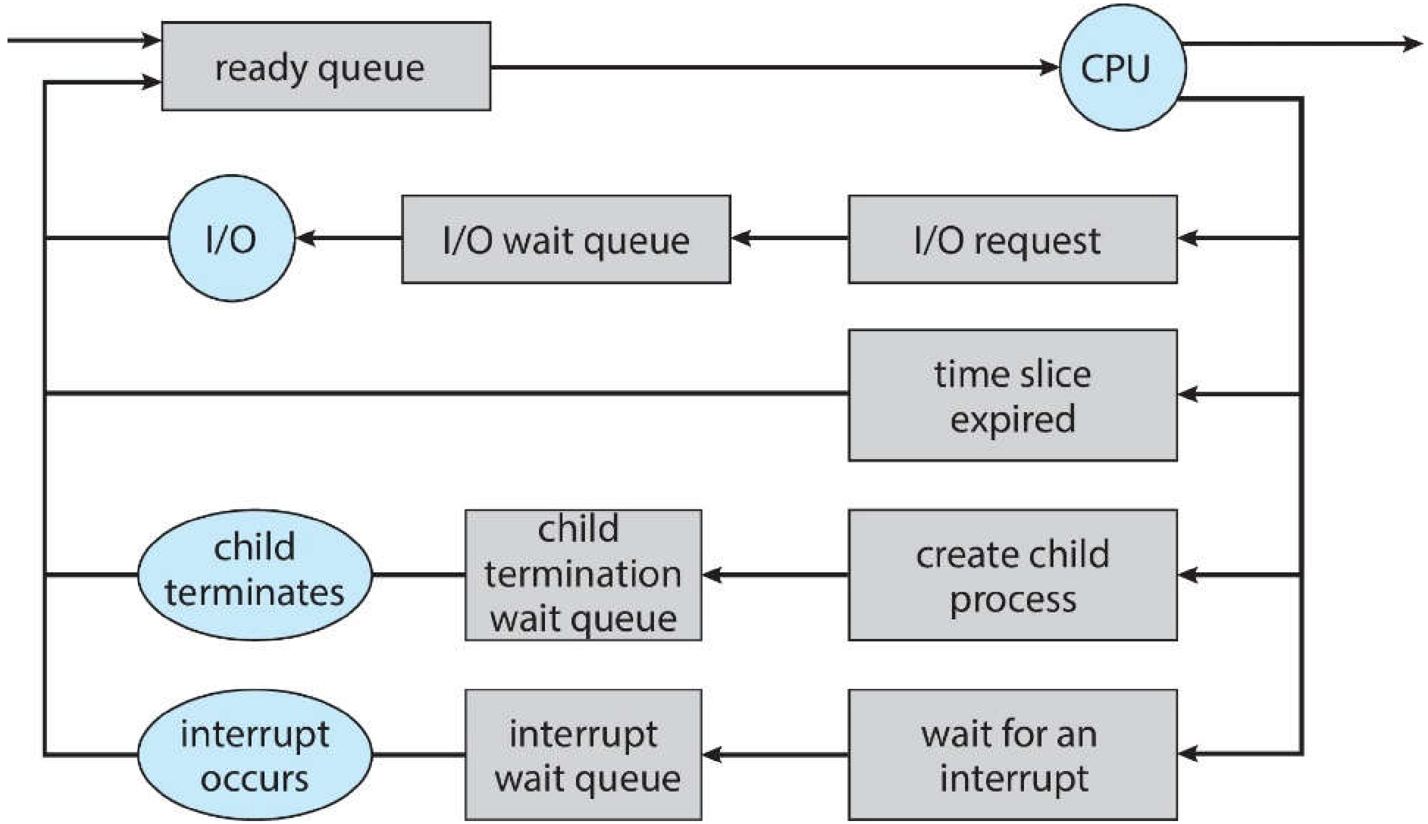
Process Scheduling Queues

- Queuing diagram for process scheduling
  - Processes migrate among the various queues.



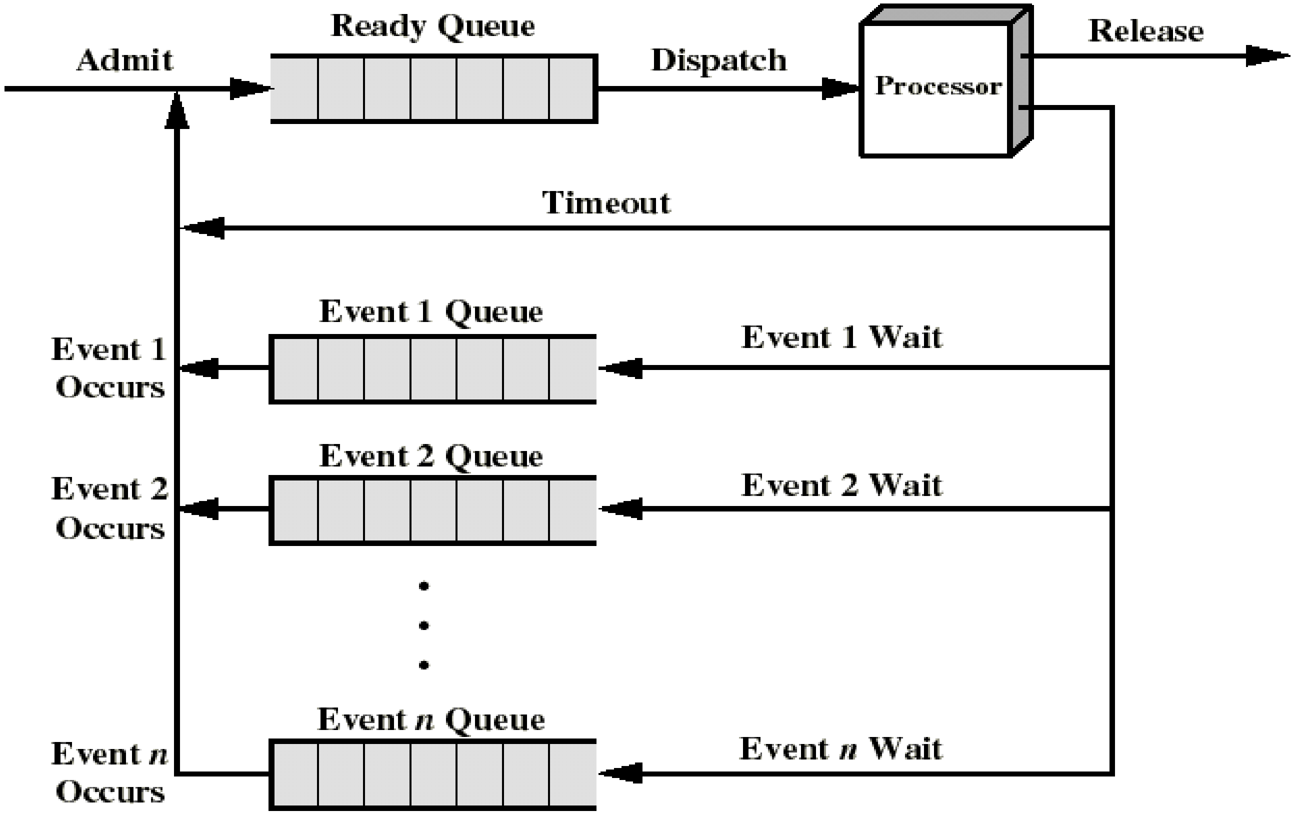
Process Scheduling Queues

- Queuing diagram for process scheduling.



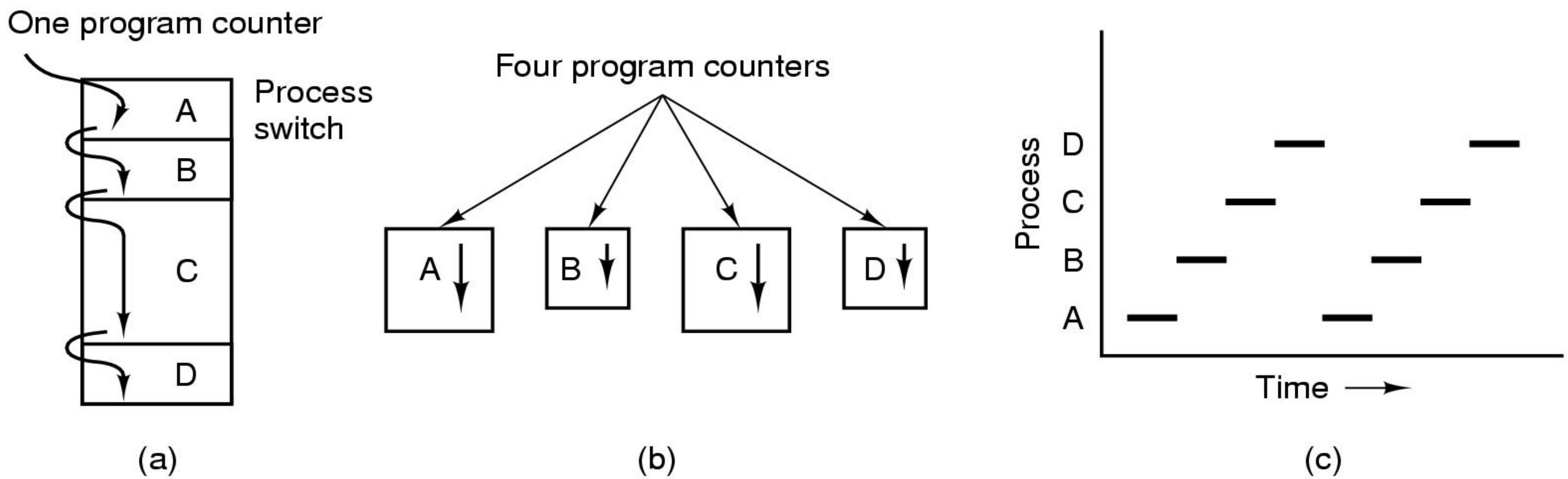
Process Scheduling Queues

- A Queuing Discipline
  - When some event occurs, the corresponding process is moved into the ready queue.



Process Switching

- Multiprogramming Scheduling.



Process Switching

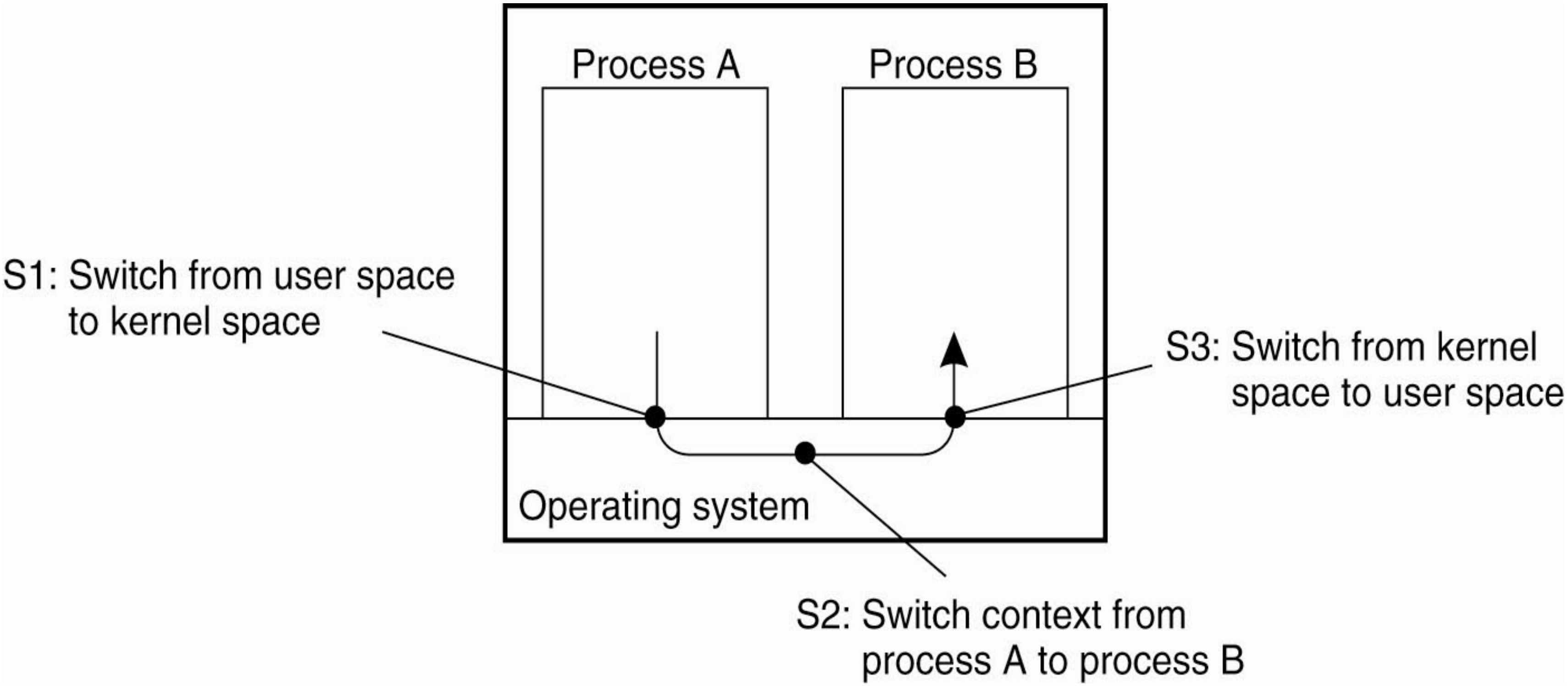
- A process switch (进程切换) may occur whenever the OS has gained control of CPU. i.e., when:
  - Supervisor Call
    - explicit request by the program (example: file open)
      - The process will probably be blocked.
  - Trap
    - an error resulted from the last instruction
      - It may cause the process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

Context Switching

- When CPU switches to another process, the system must save the state of the rolling out process and load the saved state of the rolling in process.
  - This is called *Context Switch* (上下文切换).
  - Context of a process is represented in the PCB.
- The time it takes is dependent on hardware support.
- Context-switch time is overhead (开销).
  - The system does no useful work while switching.

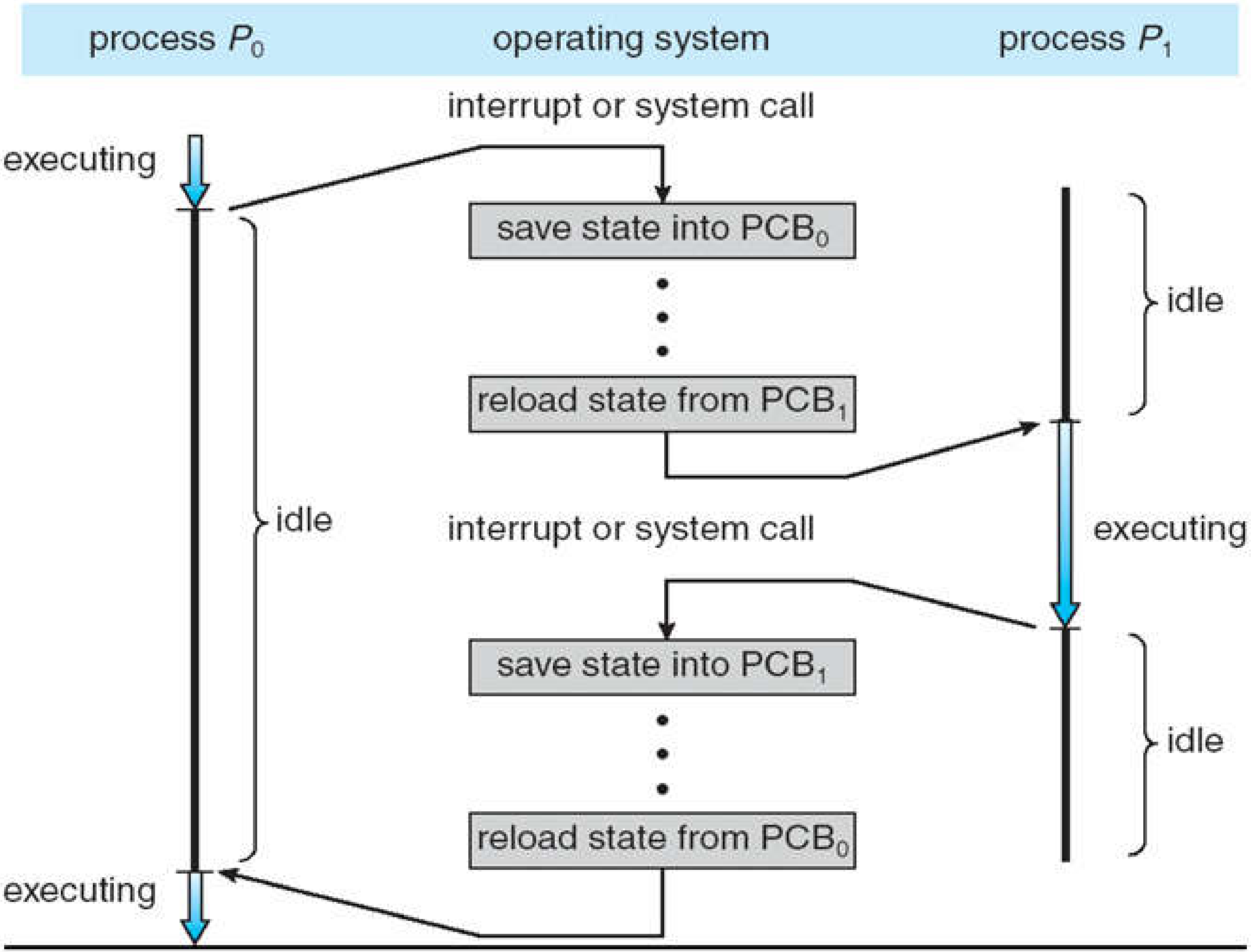
Context Switching

- Context Switch between Processes.



Context Switching

- Context Switch between Processes.



- **Context Switching**
  - Steps in Context Switch
    - Save CPU context including program counter and other registers.
    - Update the PCB of the running process with its new state and other associate information.
    - Move PCB to an appropriate queue – ready, blocked,
    - Select another process (for next execution).
    - Update PCB of the selected process.
    - Restore CPU context from that of the selected process.

Mode Switching

- An interrupt may 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* (模式切换)
  - user mode to kernel mode when going into Interrupt Handler.
- Less overhead
  - no need to update the PCB like for context switch.