Process Description and Control

Chapter 3

Each process has its own **execution state**

OS should keep track of these execution states and manages the **transitions**

Data structures are designed to describe each process

(Classical) Processes
(经典)进程

Processes

- **Processes support the ability to have (pseudo) concurrent operation**
  - even when there is only one CPU available
    - The CPU switches back and forth from process to process quickly -- called "multiprogramming"
- **Processes turn a single CPU into multiple virtual CPUs.**
  - They make a modern computer do several things at the same time

Process Execution

- **Types of *Concurrency* (并发性)**
  - On a uniprocessor: the execution of multiple processes can be interleaved in time (交替执行)→ *pseudoparallelism* (伪并行)
  - On a multiprocessor: interleaved execution + *simultaneous/parallel execution* (并行)
- **Concurrency leads to a host of difficult problems**

Multiprogramming OSes

- **Three major lines (三条主线) of computer system development**
  - **Multiprogramming batch system**
  - **Time-sharing system**
  - **Real-time transaction processing system**
- **They all contributed to the development of the concept of the process.**

Multiprogramming Batch System

- **Multiprogramming is designed to keep the processor and I/O devices simultaneously busy to achieve *maximum efficiency***

Time-Sharing Systems

- **The key design objective is to:**
  - **Be responsive to the needs of the individual user**
  - **Be able to support many users simultaneously, for the cost reasons.**
- **These two goals are compatible, because of the relatively slow reaction time of the user.**

Real-Time Transaction Processing System

- **A number of users are entering queries or updates against a database**
  - Example: airline reservation system
- **Difference between transaction processing system and time-sharing system:**
  - The former is limited to one or a few applications, while the latter can engage in various applications
- **System response time is paramount in both cases**

Definitions

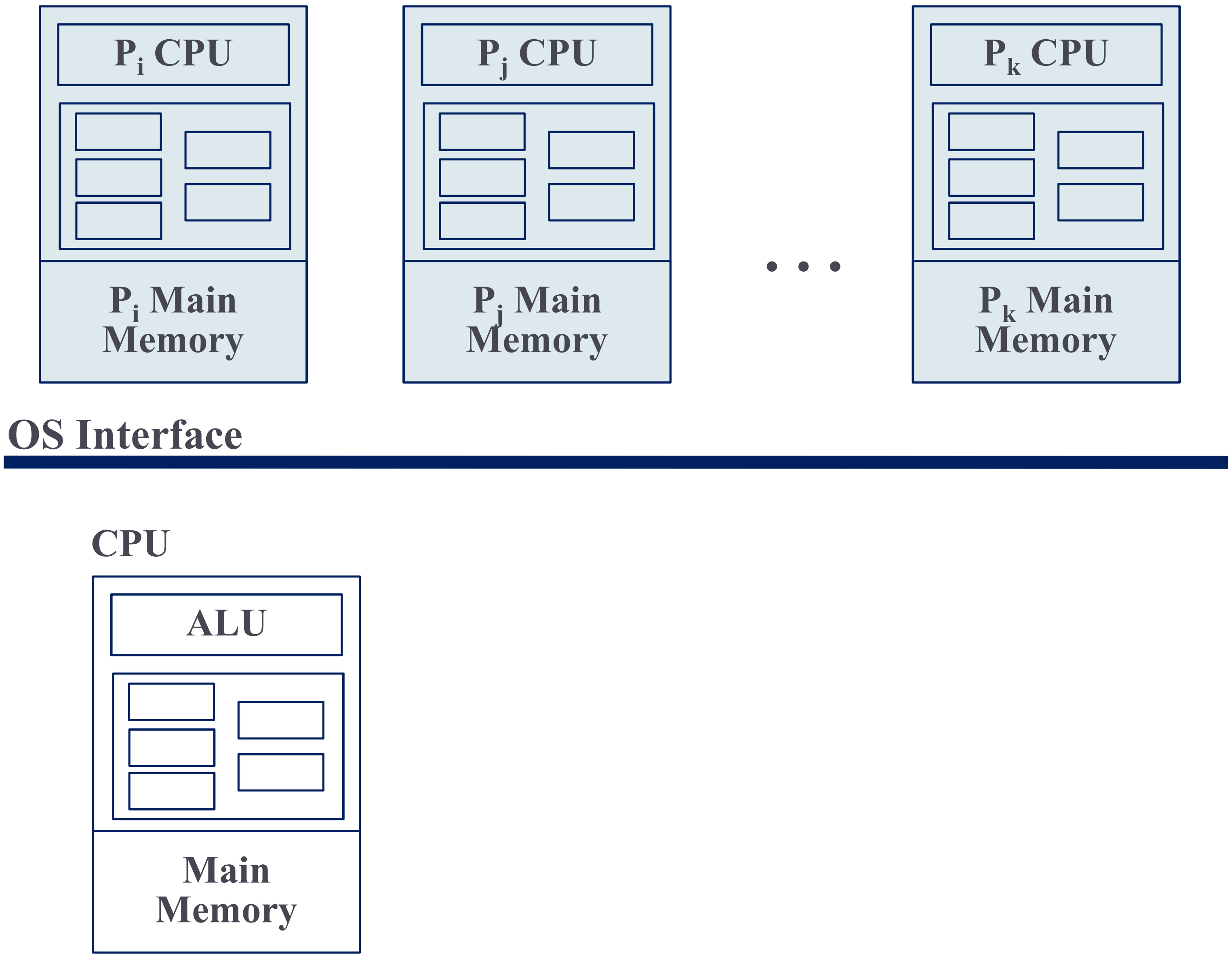
What are the differences between processes and programs?

- **Several Definitions have been given for the term "(classical) process":**
  - *A program in execution*
  - *An instance of a program running on a computer*
  - *The entity that can be assigned to and executed on a processor*
  - *A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources*

进程的特性：动态性、并发性、独立性、异步性

The Abstract Machine for Classic Processes

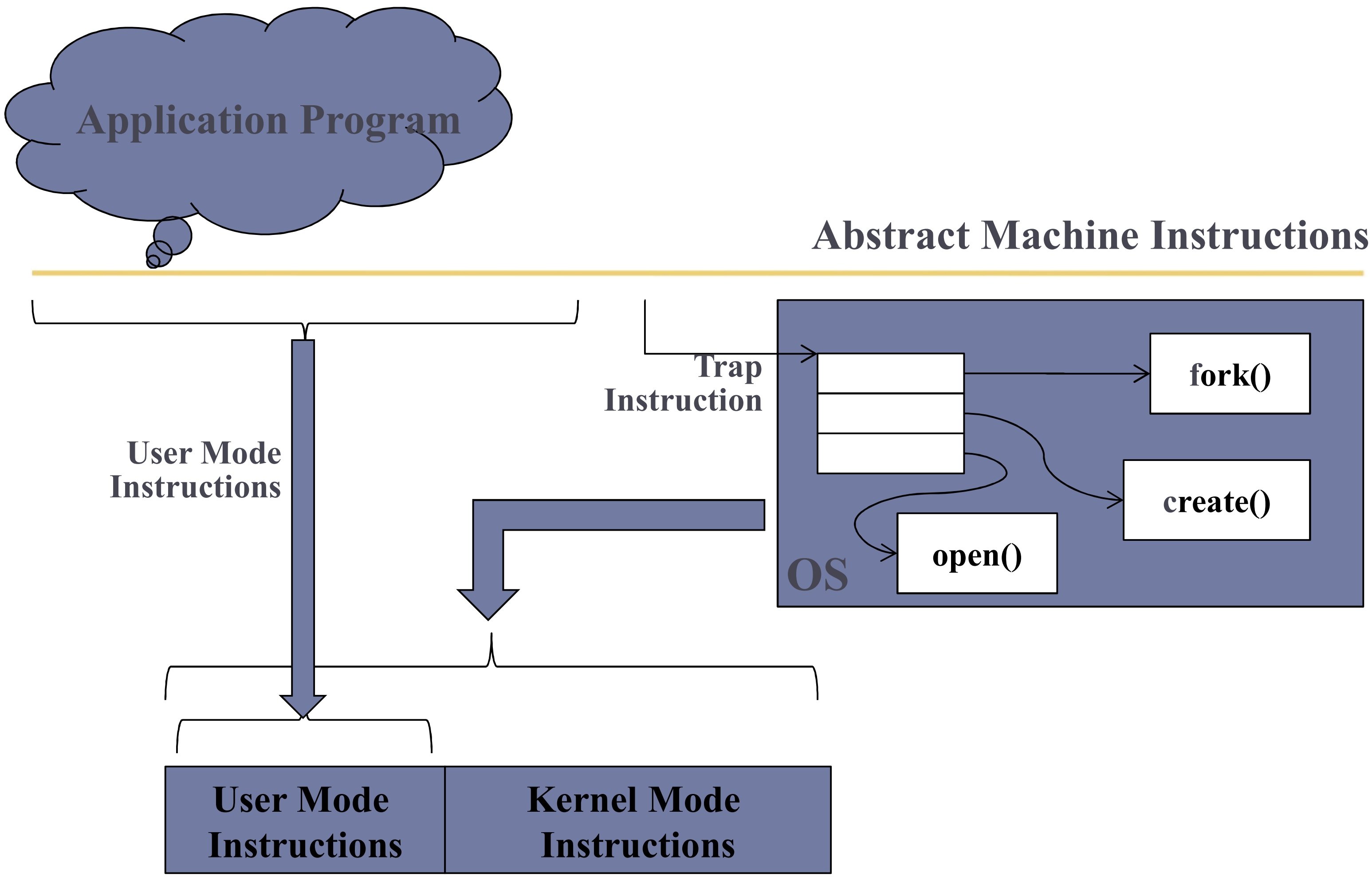
- **Multiprogramming OSes make the illusion that application programs each have their own exclusive machine where to execute their codes**
  - The OS allocates blocks of main memory using a ***space-multiplexing* (空分复用)** appoach
  - It allocates the processor different classical processes using ***time-multiplexing* (时分复用)**


P_i CPU
P_i Main Memory
P_j CPU
P_j Main Memory
P_k CPU
P_k Main Memory
OS Interface
CPU
ALU
Main Memory

Abstract Machine Interface

- **The abstract machine interface consists of**
  - the *OS system calls*, and
  - the *user mode instructions* (including `trap` instruction)
- **The user mode `trap` instruction**
  - switches the CPU to kernel mode, then branches to an OS function entry point.

The Abstract Machine Interface



Instruction Trace
指令轨迹

- ***Instruction trace* is used to characterize the behavior of an individual process.**
  - *Instruction trace for a process is* ***the sequence of instructions that are executed*** *for that process.*
- ***The traces of the various processes are interleaved, from the processor's point of view.***

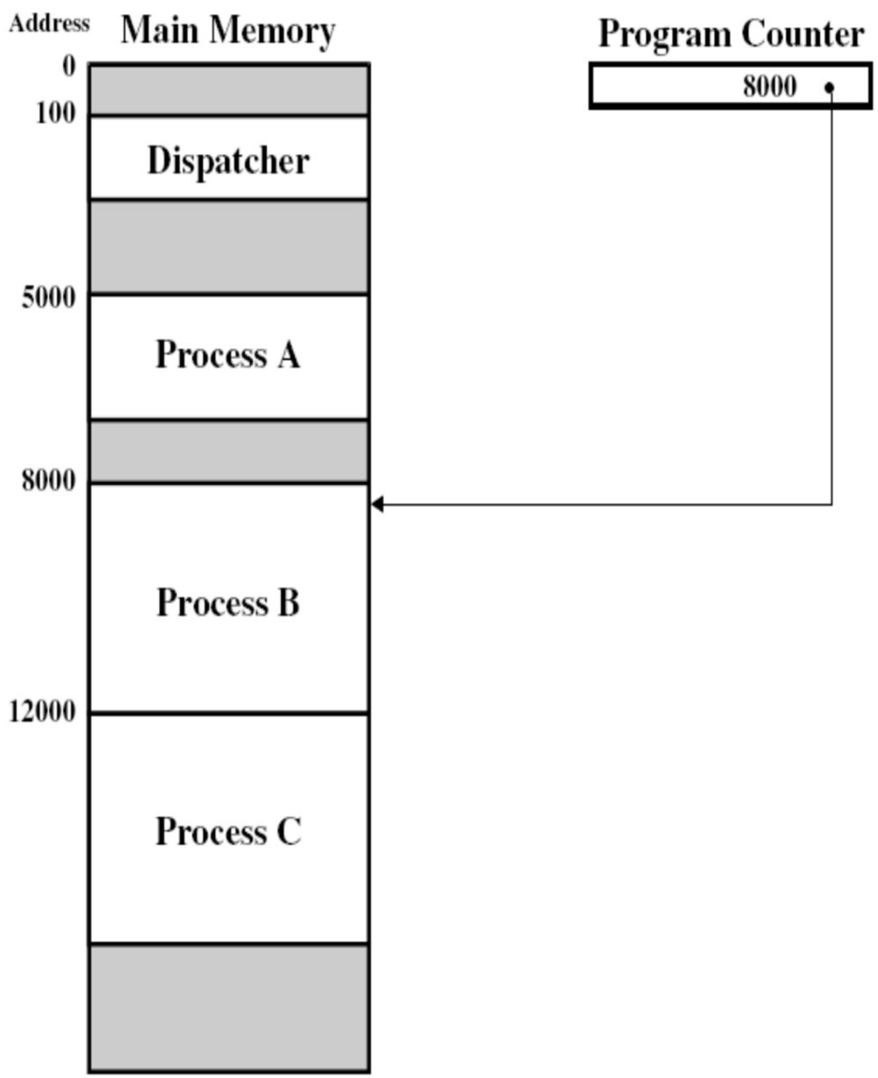


Figure 3.1 Snapshot of Example Execution (Figure 3.3) at Instruction Cycle 13

There are three processes in system: Process A, Process B, Process C.

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

an I/O operation

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

The OS only allows a process to continue execution for a maximum of 6 instructions

Figure 3.2 Traces of Processes of Figure 3.1

Assume that the OS only allows a process to continue execution for a **maximum of six instruction cycles**

1	5000	27	12004
2	5001	28	12005
3	5002	------------------------Time out	
4	5003	29	100
5	5004	30	101
6	5005	31	102
------------------------Time out		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	------------------------Time out	
16	8003	41	100
--------------------I/O request		42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		------------------------Time out	

The execution of some code in the dispatcher (分派程序/分派器)

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.3 Combined Trace of Processes of Figure 3.1

Process States
进程状态

Transparent to the user or application program

Process Creation: When?

- *Initialization of a batch job (in batch environment)*
- *User logs on or requests to create a process(in interactive environment)*
- *Created by OS to provide a service (such as printing)*
- *A process can make an explicit request to creates (or spawns) another process.*
  - *The spawned process is referred to as the child process, while the former process is the parent process*

When OS creates a process at the explicit request of another process, the action is referred to as process spawning (进程派生)

Process Termination: When?

- *A batch job issues an explicit operating system service call to indicate its completion (voluntary) – exit() in UNIX and ExitProcess() in Windows*
- *For an interactive application, the action of the user will indicate when the process is completed:*
  - *In a timesharing system: User logs off or turns off the terminal*
  - *On a personal computer or workstation, user may quit an application*
- *A number of error and fault conditions*
- *In some operating systems, a process may be terminated by its parent process or when the parent process is terminated. (involuntary)*

Reasons for Process Termination (1)

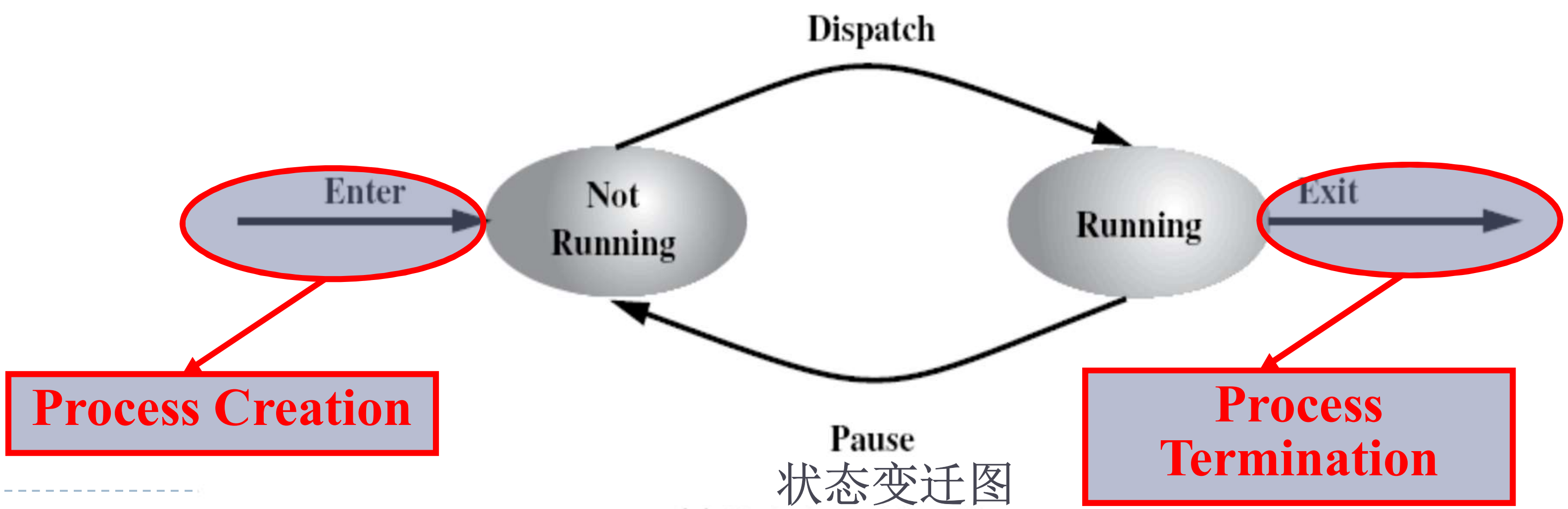
- Normal completion (正常完成)
- Time limit exceeded (超过时限)
- Memory unavailable (无可用内存)
- Bounds violation (越界)
- Protection error (保护错误)
  - For example: write to read-only file
- Arithmetic error (算数错误，例如除以0)
- Time overrun (时间超出)
  - process waited longer than a specified maximum for an event

Reasons for Process Termination (2)

- I/O failure (I/O失败)
- Invalid instruction (无效指令)
  - happens when try to execute data
- Privileged instruction (特权指令)
- Operator or OS intervention (操作员或操作系统干预)
  - such as when deadlock occurs
- Parent termination (父进程终止)
- Parent request (父进程请求)

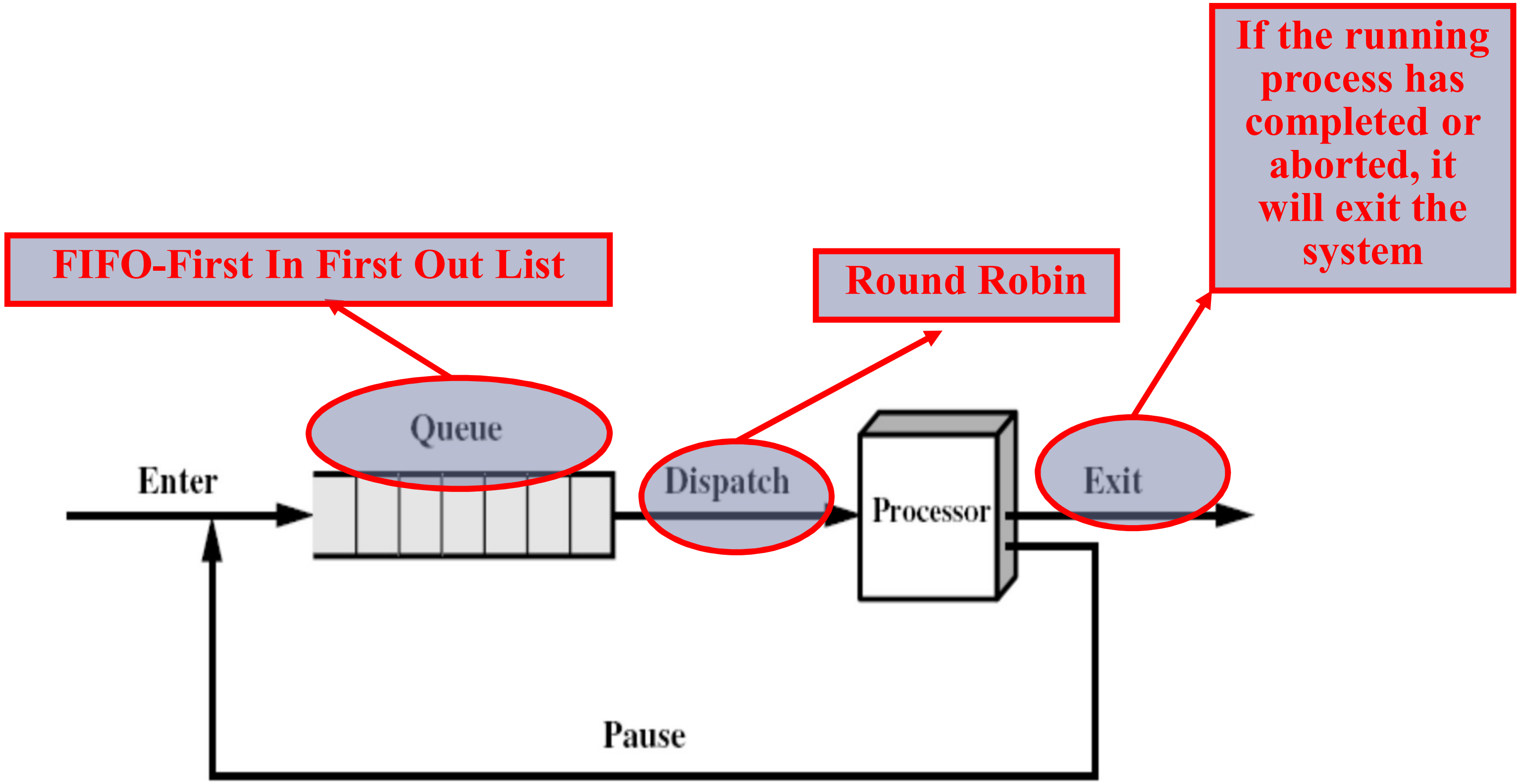
A Two-State Process Model

- Process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram

Not-Running Process in a Queue



(b) Queuing diagram

排队图

Dispatch: 分派

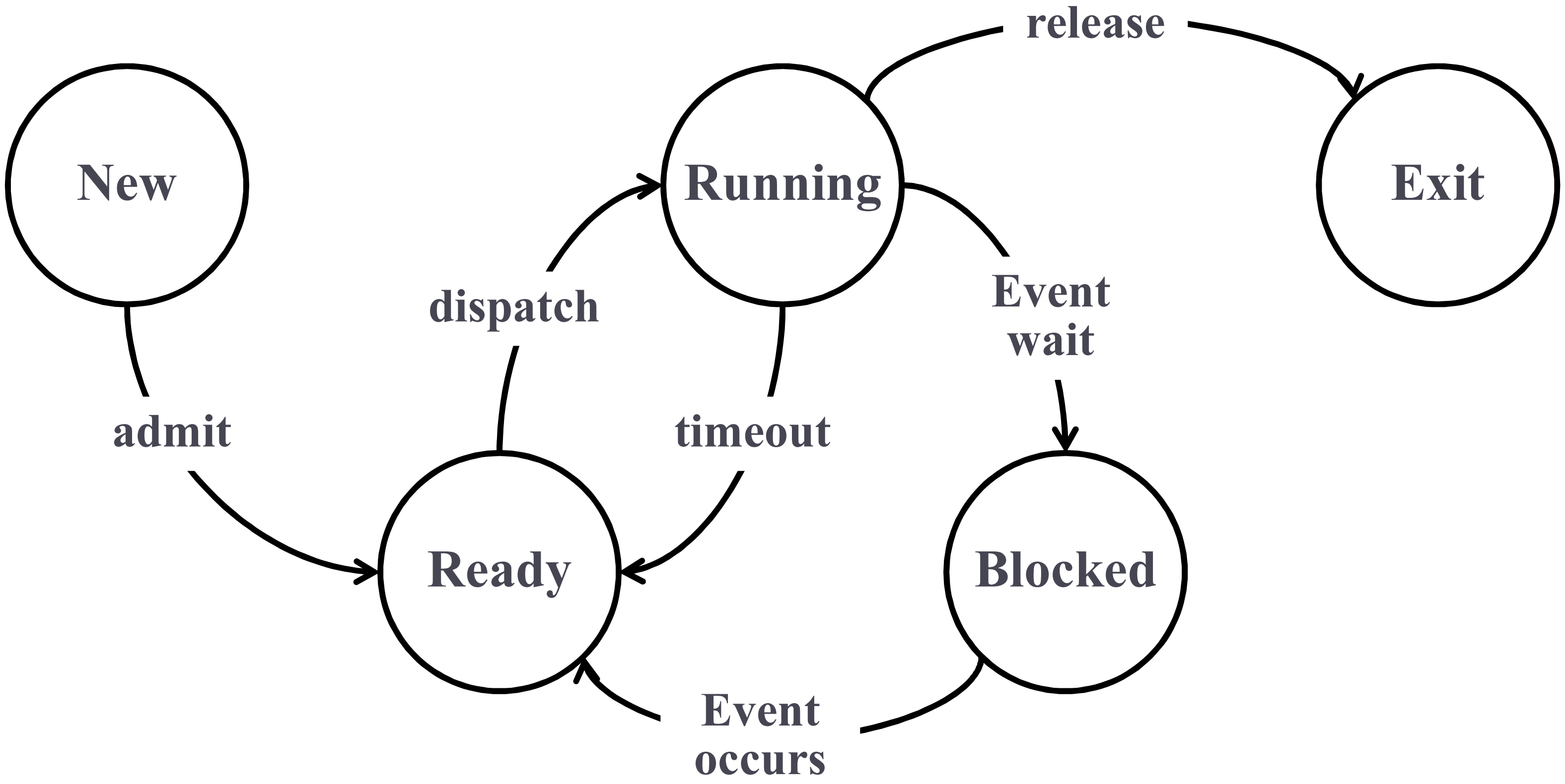
Dispatcher：分派程序/分派器

Two-State Model is Not Enough! Why?

- **With only one queue of not-running, dispatcher would have to look for the process that is not blocked**
  - cannot just select the process that has been in the queue the longest because it may be blocked
- **Solution: Split the Not-Running state into**
  - Ready State
    - ready to execute
  - Blocked State
    - waiting for some event to happen

A Five-State Model

- **Running (运行)**
  - *The process that is currently being executed*
- **Ready (就绪)**
  - *The process that is prepared to execute when given the opportunity*
- **Blocked (阻塞)**
  - *A process that cannot execute until some event occurs.*
- **New (新建)**
  - *Typically a new process has not yet been loaded into main memory*
- **Exit (退出)**



Five-state process model

Why Use the “New” State

- At New state, the necessary housekeeping chores (辅助工作) has been performed by the OS
  - An identifier is associated with the process
  - Any tables for managing the process are allocated and built
- The OS may limit the number of processes that may be in the system ***for reasons of performance or main memory limitation***.
  - The process itself is not in main memory (the program is still in secondary storage)
  - In systems that support virtual memory, the program and data are loaded into virtual memory when a process moves from New to Ready

Why Use "Exit" State?

- **Exit State: The tables and other information associated with the process is temporarily preserved by OS**
  - which provides time for auxiliary or support programs to extract any needed information
    - An accounting program → processor time and other resource utilization → billing purposes
    - A utility program → information about the history of the process →performance or utilization analysis

Possible State-Transitions (1)

- Null→New: A process is created
- New→Ready: The OS moves a process from New state to Ready state, when it is prepared to take on an additional process
- Ready→Running: When it is time to select a new process to run
- Running→Exit: normal termination or aborts
- "Ready→Exit" or "Blocked→Exit": These two transitions are not shown on the state diagram.
  - In some systems: A parent can terminate a child process at any time (or the termination of a parent process will cause the termination of all its child processes)

Possible State-Transitions (2)

- Running→Ready
  - The running process has reached the maximum allowable time
  - A running process is preempted by another process of higher priority level
  - A process may voluntarily release control of the processor
- Running→Blocked
  - A process is put in the blocked state if it requests something for which it must wait. (waiting for I/O operation to compeletion or a message from another process)
- Blocked→Ready
  - When the event for which a process in the Blocked state is waiting occurs, the process is moved to the Ready state.

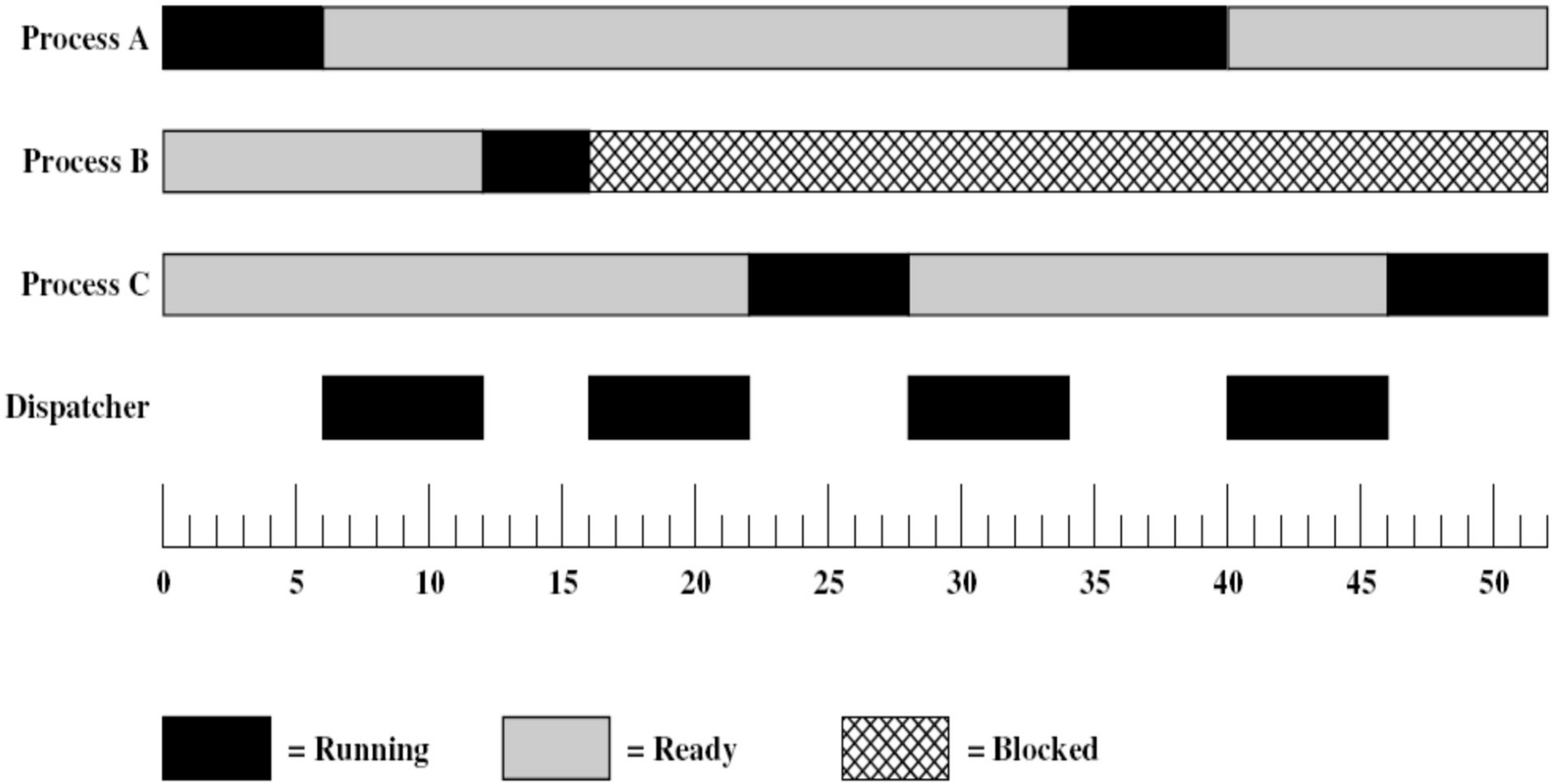
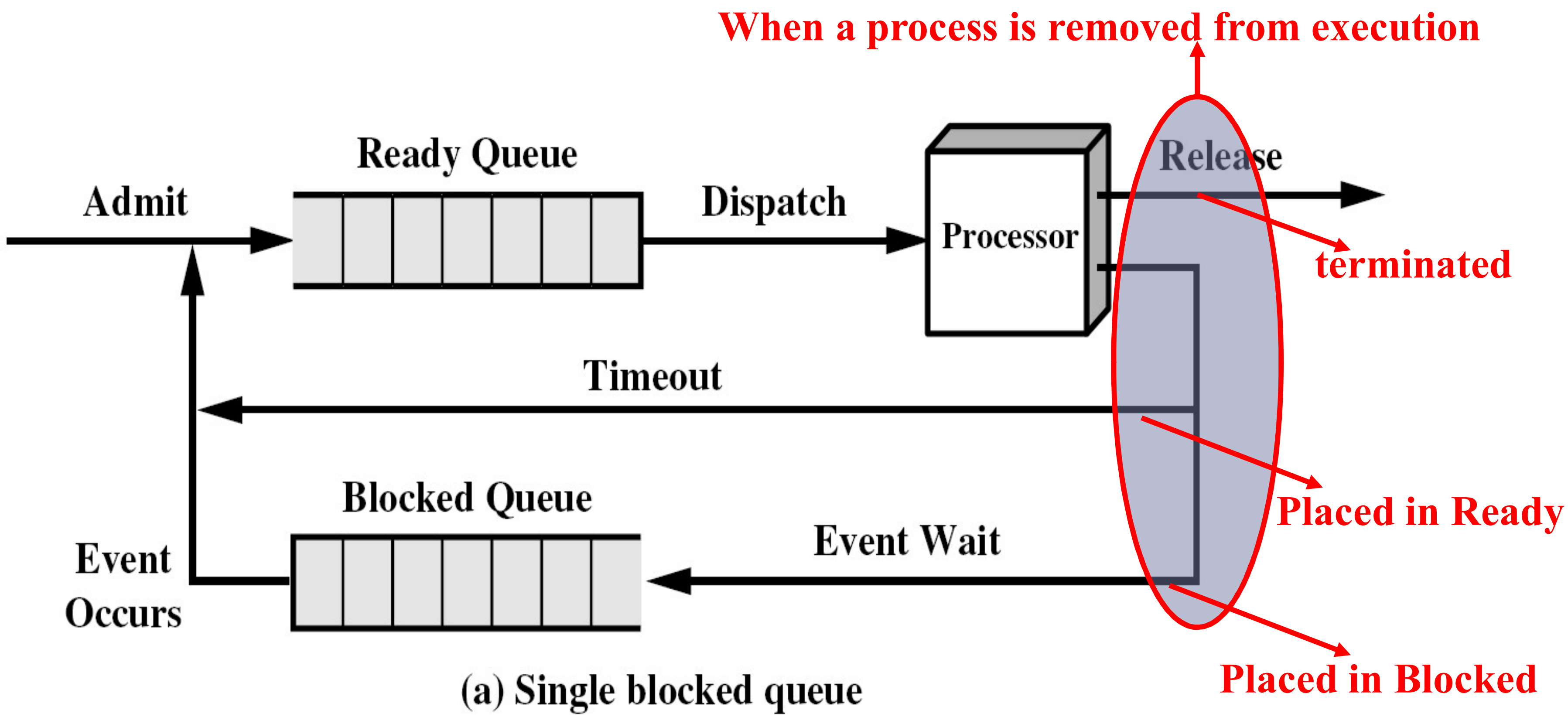


Figure 3.6 Process States for Trace of Figure 3.3

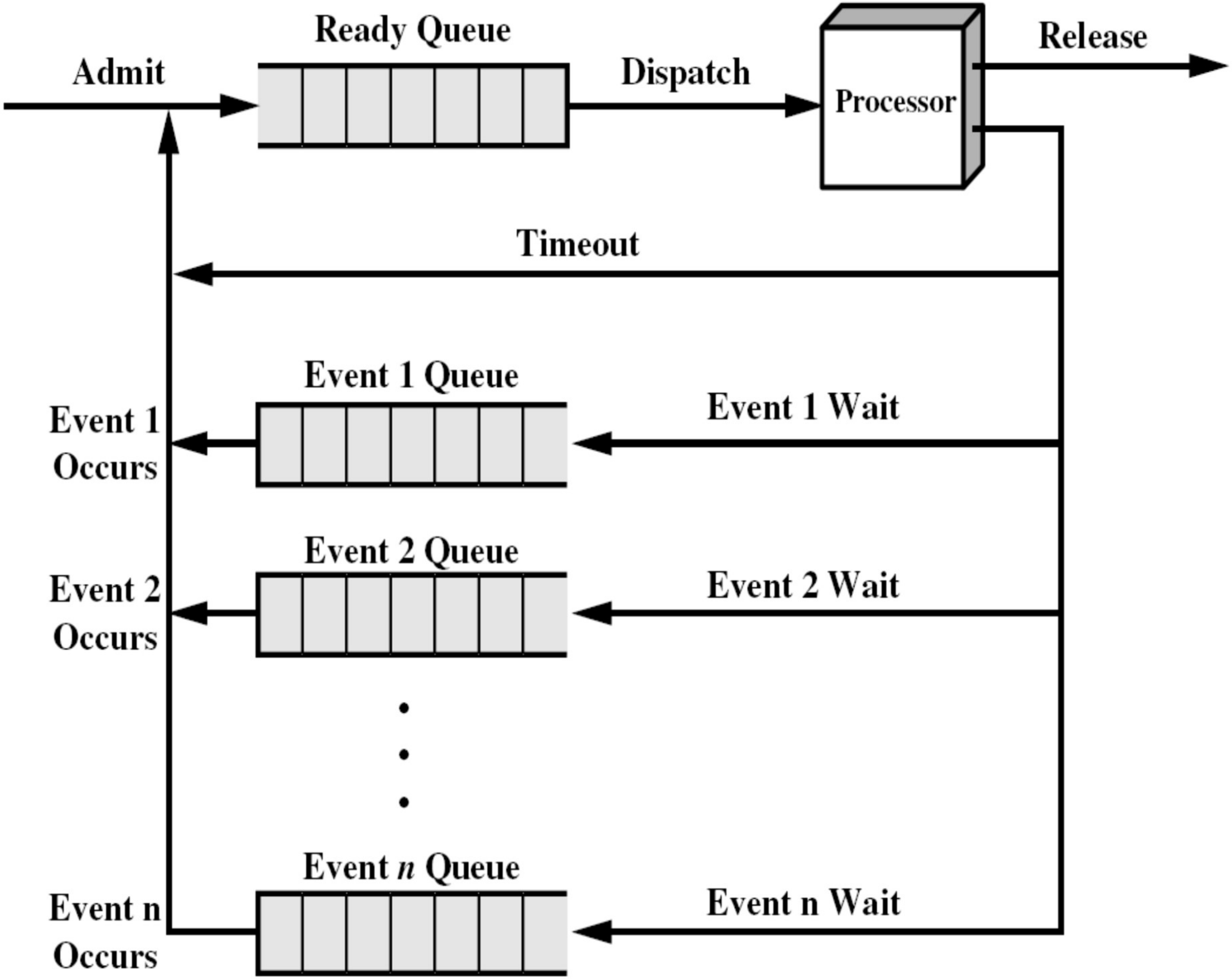
Queuing Models
Single Blocked Queue



(a) Single blocked queue

Queuing Model
Multiple Blocked Queues

- What is the problem with a single Blocked queue?
  - When an event occurs, the OS must scan the entire Blocked queue to search for those purposes waiting on that event.
  - In a larger operating system, there could be hundreds or even thousands of processes in that queue→inefficiency!!
- How about having a number of queues, one for each event?



(b) Multiple blocked queues

Final Refinement
Multiple Ready Queues

- If the dispatching of processes is dictated by a priority scheme, then it would be convenient to have a number of Ready queues!
  - The operating system could easily determine which is the highest-priority ready process that has been waiting the longest.
  - **Why????**

Suspended Processes (1)
挂起的进程

It should be blamed !!! For not only the Suspended States but also the Blocked State

- Let us consider **a system without employing virtual memory**.
  - Each process to be executed must be loaded fully into main memory.
- Processor is so much faster than I/O that the processor could be still idle most of the time!! **How to improve the utilization of the processor**??
  - To **expand the main memory** to accommodate more processes????
  - To **Swap these processes to disk** to free up more memory

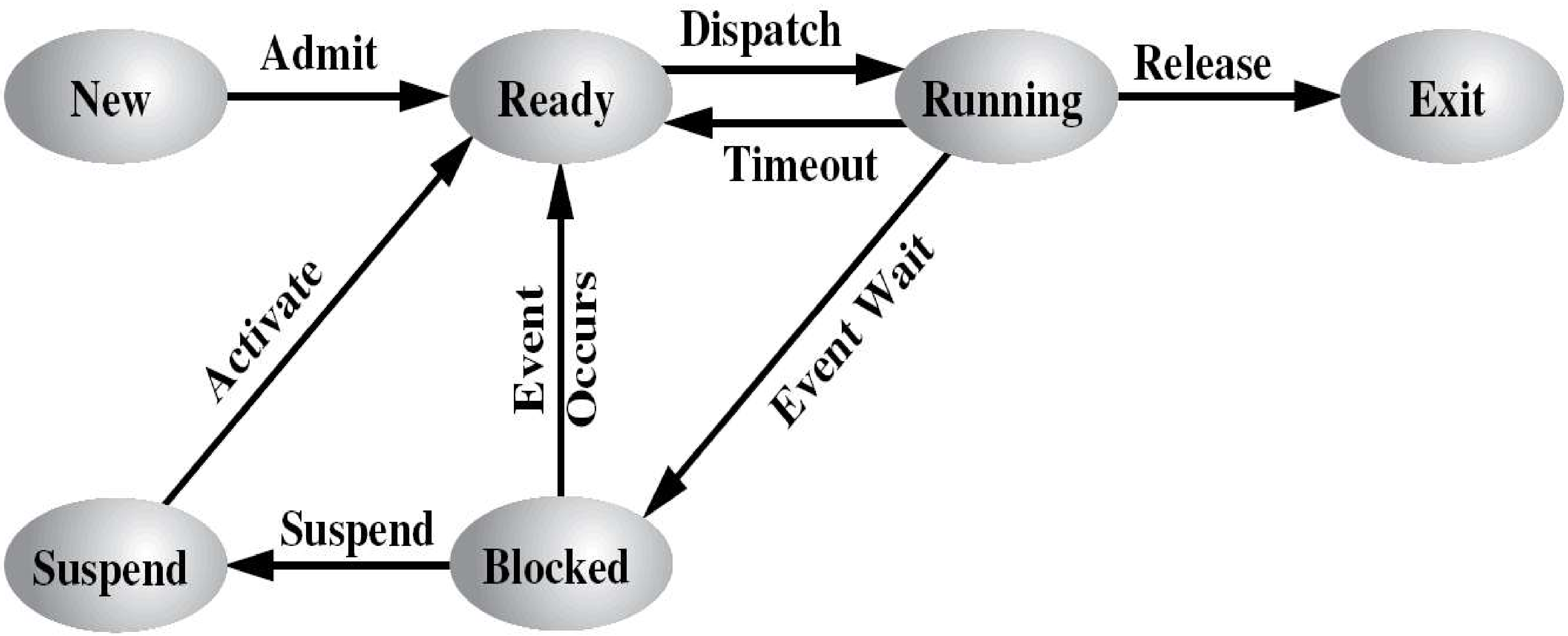
Suspended Processes (2)
Swapping (交換)

- When none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out onto disk (**into a suspend state**)
  - Correspondingly, there is a suspend queue.
- The OS then brings in another process from the suspend queue, or accepts a new-process request.
- Swapping is also an I/O operation. Why it does not make the situation worse?
  - Disk I/O is the fastest I/O on a system!!

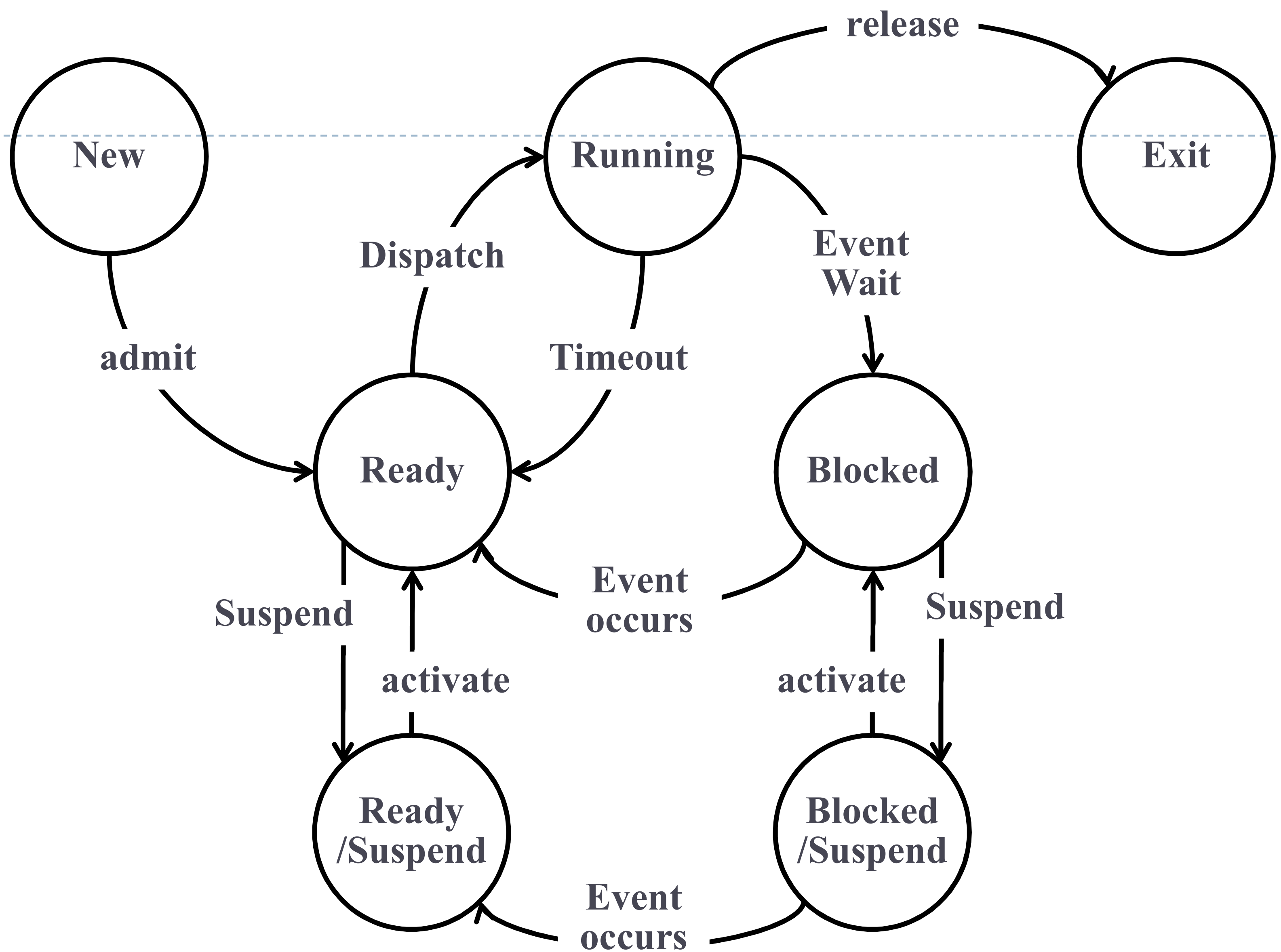
The Design of Suspend States

- One Suspend State
  - The difficulty is:
    - All of the processes that have been suspended were in the Blocked state. (**It does no good to bring a blocked process back into main memory**)
  - However, blocked process is waiting for an event. (if that event occurs, the process is not blocked.)
- Two Suspend States
  - Blocked/Suspend: The process is in secondary memory and awaiting an event.
  - Ready/Suspend: The process is in secondary memory and is available for execution if loaded into main memory.

One Suspend State



(a) With One Suspend State



Two Suspend States

Important New Transitions (1)

- Blocked→Blocked/Suspend:
  - No ready processes
  - OS determines that the process to be dispatched requires more main memory.
- Blocked/Suspend→Ready/Suspend:
  - The event that the process is waiting for occurs
- Ready/Suspend→Ready
  - When there are no ready processes in main memory
  - A process in the Ready/Suspend has higher priority than any processes in the Ready state.

Important New Transitions (2)

- Ready→Ready/Suspend: (Normally, OS prefers to suspend a Blocked process)
  - It may be necessary to suspend a ready process if it is the only way to free up a sufficiently large block of main memory.
  - The OS may choose to suspend a lower-priority ready process rather than a higher-priority blocked process.

Other Transitions

- **New→Ready/Suspend and New→Ready:**
- **Blocked/Suspend→Blocked:**
- **Running→Ready/Suspend**
- **Various→Exit:**

Reasons for Process Suspension

Swapping	The operating system needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The operating system may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendents.

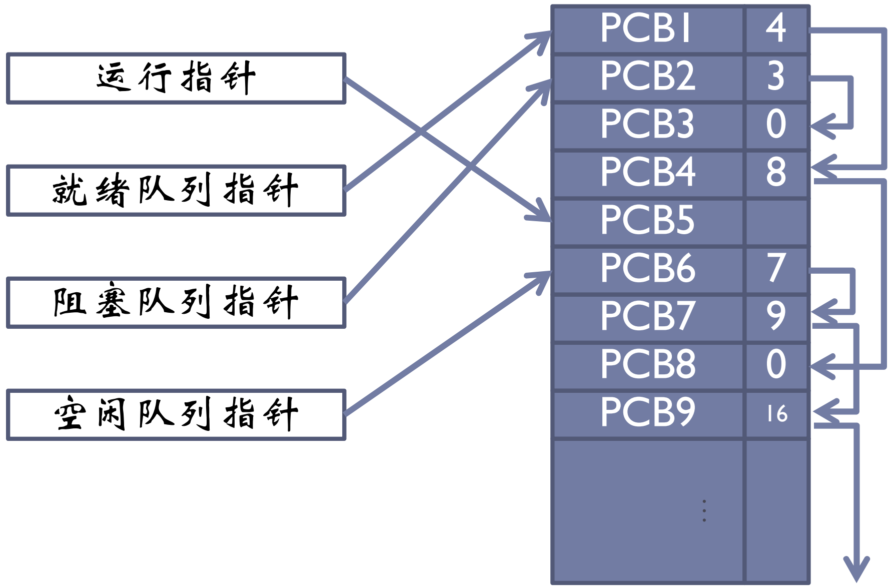
Process Description

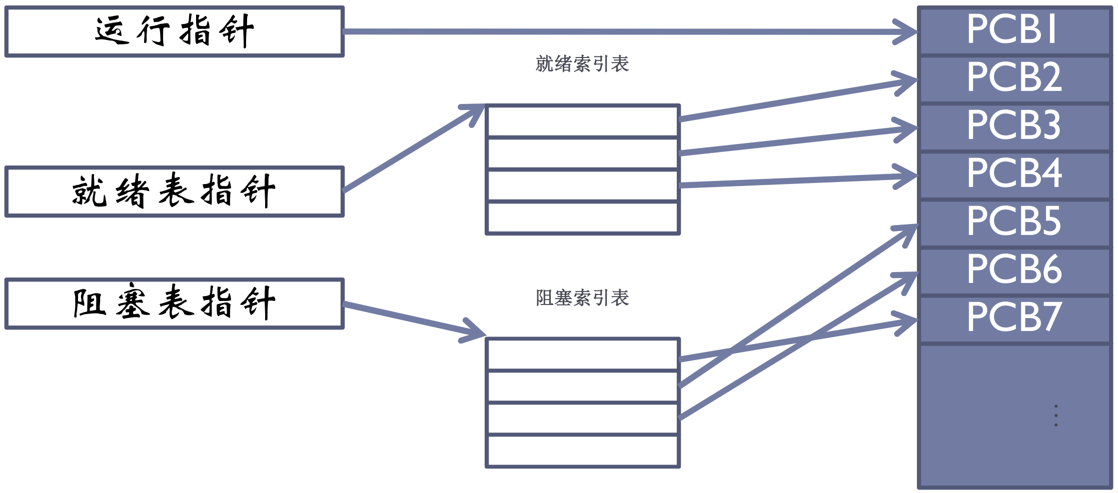
Process Tables

- **The OS maintains a table (an array of structures), called the process table, with one entry per process**
- **The entries are called process control block or process descriptor.**
  - Where process is located
  - The attributes of the process that are necessary for its management, such as process ID and process state

The Organization of PCBs

- Two common way to manage the PCBs
  - 链接方式:
    - 把具有同一状态的PCB，用其中的链接字链接成一个队列，从而形成就绪队列、阻塞队列和空白队列等
  - 索引方式:
    - 系统根据进程的状态创建几张索引表，索引表在内存的首地址记录在内存的一些专用单元中。
    - 索引表的每个表目指向具有相应状态的某个PCB


运行指针
就绪队列指针
阻塞队列指针
空闲队列指针
PCB1
4
PCB2
3
PCB3
0
PCB4
8
PCB5
PCB6
7
PCB7
9
PCB8
0
PCB9
16


运行指针
就绪索引表
就绪表指针
阻塞表指针
阻塞索引表
PCB1
PCB2
PCB3
PCB4
PCB5
PCB6
PCB7

Process Image
进程映像

- Process includes set of programs to be executed
  - Data locations for local and global variables
  - Any defined constants
  - Stack
- Process control block
  - Collection of attributes about a specific process
- Process image
  - Collection of program, data, stack, and attributes

Process Location

- The location of a process image ***depends on the memory management scheme being used***.
  - When the process image is maintained as a contiguous block of memory.
    - The operating system must know the location of each process on disk and the location of that process in main memory.
  - When a process image consists of a set of blocks (page or segment or a combined one) that need not be stored contiguously.
    - The process table must show the location of each block

Process Control Block Information Categories

- *Process control block contains many pieces of information about a specific process*
  - *Also called the process descriptor*
- *The information stored in PCB are usually grouped into three categories:*
  - *Process Identification (进程标识信息)*
  - *Processor State Information (处理器状态信息)*
  - *Process Control Information (进程控制信息)*

PCB Information: Process Identification

- **Process Identifiers (进程标识号)**
  - Each process is assigned a unique numeric identifier
  - Many other tables may use process identifiers to cross reference process tables
  - In process communication, the process identifier indicates the destination of a particular communication.
  - When process is allow to create other process, identifiers indicate the parent and descendents of each process
- **User identifier (用户标识号)**
  - Indicates the user responsible for the job

PCB Information: Processor State Information (1)

- **Processor state information consists of the contents of processor registers.**
  - When a process is running, the information is in the registers
  - When a process is interrupted, all of this register information must be saved, so that it can be restored when the process resumes execution
- **Typical register set includes:**
  - User-visible registers
  - Control and status registers
  - Stack pointers.

PCB Information: Processor State Information (2)

- **User-Visible Registers**
  - A user-visible register is one that may be referenced by means of the machine language that the processor executes. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

PCB Information: Processor State Information (3)

- **Control and Status Registers**

  These are a variety of processor registers that are employed to control the operation of the processor. These include

  - *Program counter:* Contains the address of the next instruction to be fetched
  - *Condition codes:* Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
  - *Status information:* Includes interrupt enabled/disabled flags, execution mode

Example: the EFLAGS register on Pentium machines

Pentium II EFLAGS Register System Flags

ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

Figure 3.11 Pentium II EFLAGS Register

PCB Information: Processor State Information (4)

- **Stack Pointers**
  - Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

PCB Information: Process Control Information (1)

- *Scheduling and State Information*: This information is needed by the OS to perform its scheduling function. Typical items of information:
  - *Process state:* (e.g., running, ready, waiting).
  - *Priority:* describes the scheduling priority of the process
  - *Scheduling-related information:* This will depend on the scheduling algorithm used. (e.g. the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.
  - *Event:* Identity of event the process is awaiting before it can be resumed

PCB Information: Process Control Information (2)

- *Data Structuring:* A process may be linked to other process in a queue, ring, or some other structure. For example,
  - all processes in a ready state for a particular priority level may be linked in a ***queue***.
  - A process may exhibit a parent-child (creator-created) relationship with another process. → ***Tree structure***
  - The process control block may contain pointers to other processes to support these structures.

PCB Information: Process Control Information (3)

- *Interprocess Communication*
  - Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.
- *Process Privileges*
  - Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

PCB Information: Process Control Information (4)

- *Memory Management (存储管理)*
  - include pointers to segment and/or page tables that describe the virtual memory assigned to this process.
- *Resource Ownership and Utilization*
  - Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

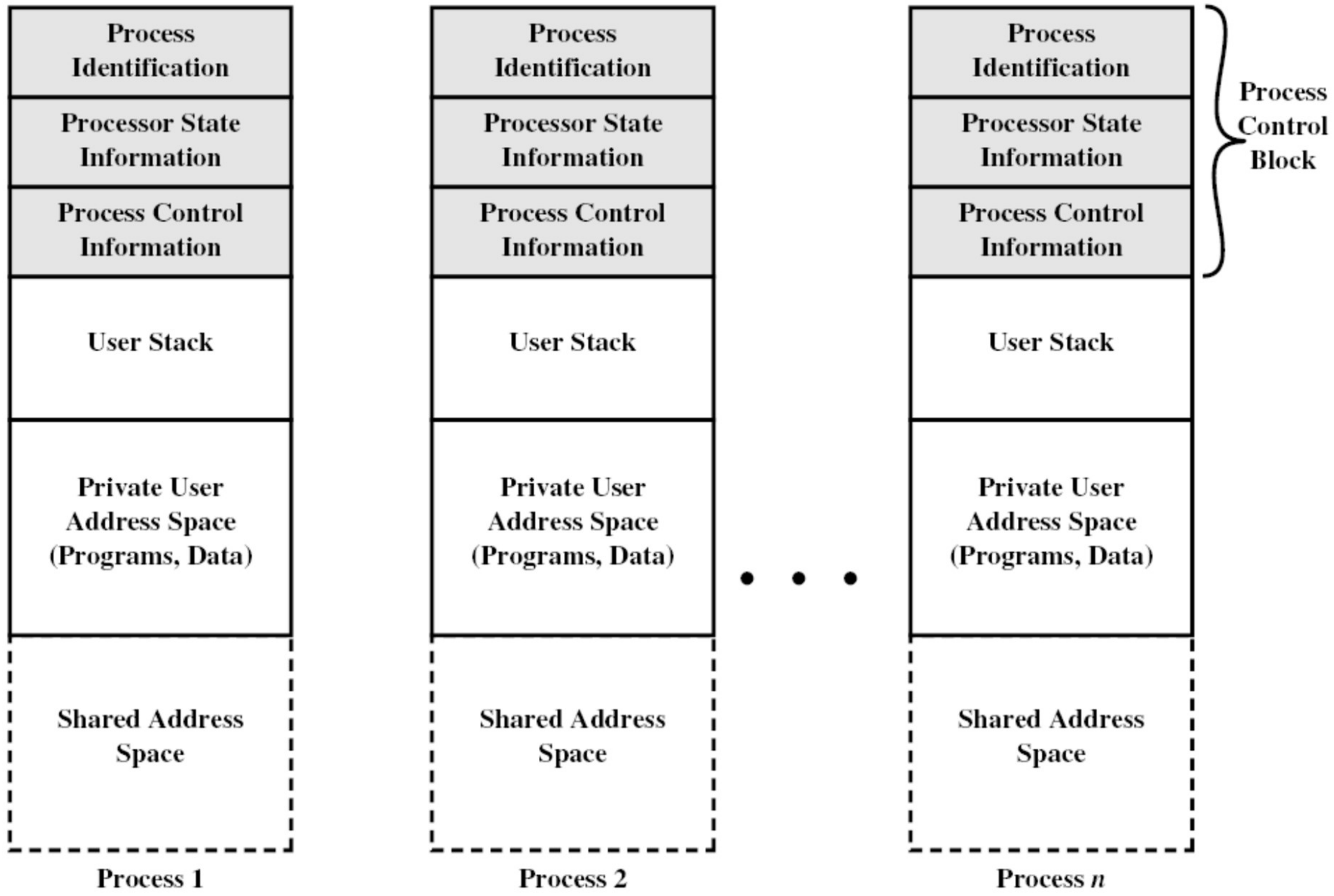
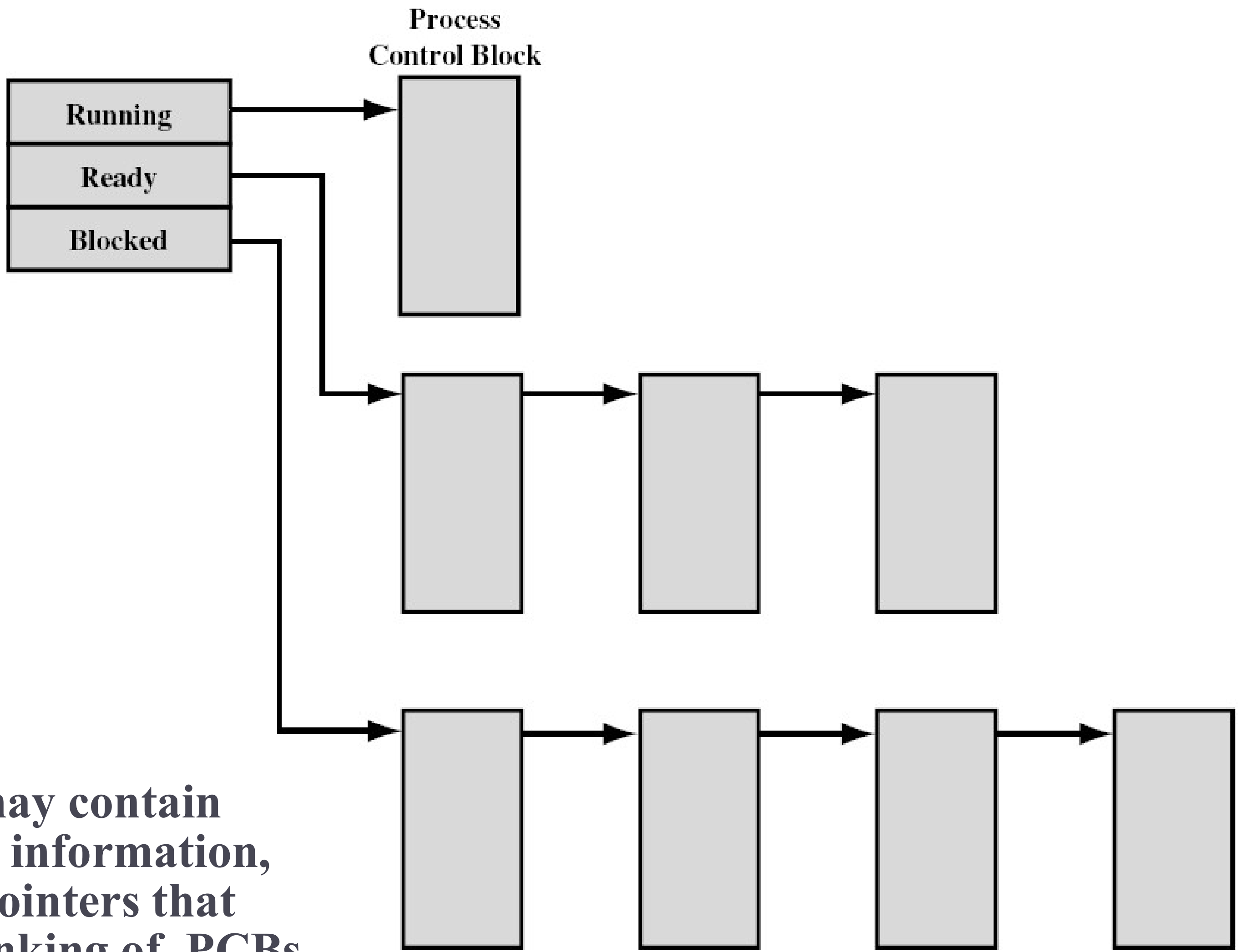


Figure 3.12 User Processes in Virtual Memory



The PCB may contain structuring information, including pointers that allow the linking of PCBs

Figure 3.13 Process List Structures

Process Control

Modes of Execution (1)
执行模式

- **Most processors support at least two modes of processor execution**
  - **User-mode (用户模式)**: the less-privileged mode
  - **Supervisor Mode, System Mode, Control Mode, or Kernel Mode (监督模式、系统模式、控制模式或者内核模式)**: the more-privileged mode
- **What is the difference?**
  - **Certain instructions** can only be executed in the more-privileged mode.
    - Reading or altering a control register, such as the program status word
    - Primitive I/O instructions
    - Instructions related to memory management.
  - **Certain memory regions** can only be accessed in the more-privileged mode

These instructions are called Supervisor, privileged, or protected instruction

Modes of Execution (2)

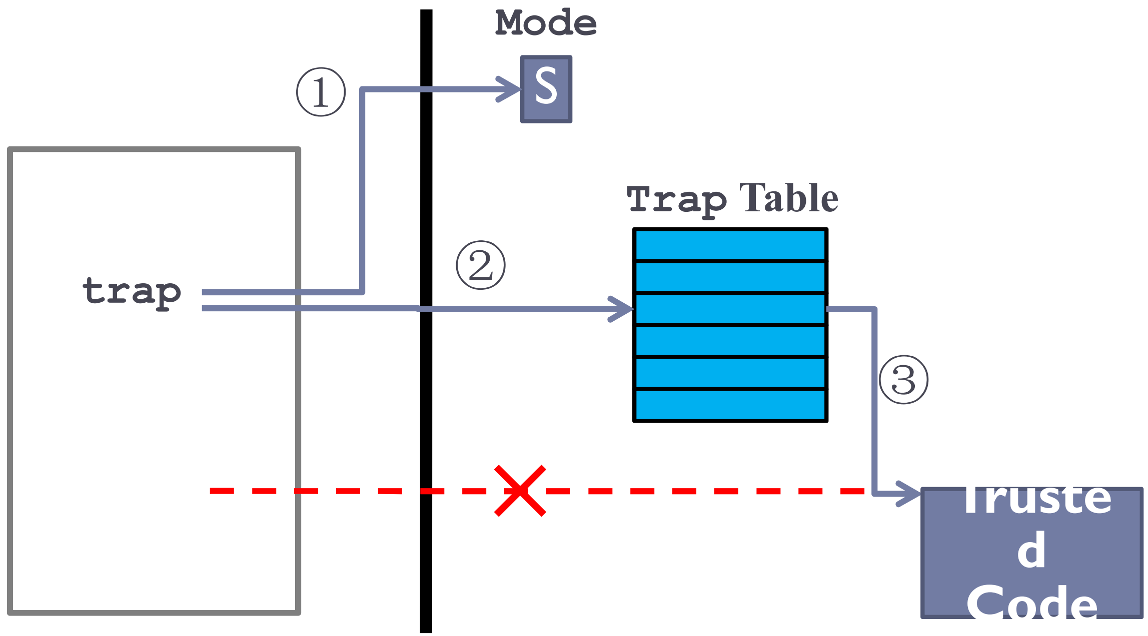
- **Why use two modes?**
  - It is necessary to protect the operating system and key operating system tables from interference by user programs
  - It is also necessary to prevent unauthorized access of resources by user programs.

Modes of Execution (3)

- A process executing in user mode should be able to switch the processor to supervisor mode, in order to begin executing OS code
- Whenever the processor switches to supervisor mode, the computer should only execute OS code
- Whenever user mode software calls the OS, the processor should be switched into supervisor mode

Modes of Execution (3)
Choices of Execution Modes

- Computer boots up in supervisor mode.
  - Used by bootstrap and OS to initialize the system
- Applications run in user mode
  - Operating system changes to user mode before running user code
  - Applications have no way to get into supervisor mode
    - Instructions that change the processor status register are privileged.
- Reentering supervisor mode is strictly controlled.
  - Only happens in response to traps and interrupts


Mode
S
①
trap
②
Trap Table
③
Truste
d
Code

Process Creation

- Assign a unique process identifier
- Allocate space for the process
  - Private user address space (program and data) and user stack
  - Whether to share any existing address space? If yes, set up the appropriate linkages.
  - Allocate the space for a process control block.
- Initialize process control block
- Set up appropriate linkages
  - Ex: add new process to linked list used for scheduling queue
- Create or expand other data structures
  - Ex: maintain an accounting file (for billing and/or performance assessment)

Process Switching

- Definition
  - A running process is interrupted and the operating system assigns another process to the Running state and turns control over to that process
- Three issues
  - *What events trigger a process switch?*
  - *What is the difference between mode switching and process switching?*
  - *What must the OS do to achieve a process switch?*

When to Switch a Process (1)

- Possible events that may give control to OS. Then a process switch may occur
  - Two kinds of system interrupts:
    - Interrupt: An interrupt is due to some sort of event that is external to and independent of the currently running process
    - Trap: A trap relates to an error or exception condition generated within the currently running process
  - System calls/Supervisor calls
    - The program being executed can make a supervisor call explicitly, to request a system function (such as a file open)

When to Switch a Process (2) Examples of Interrupts

- Clock interrupt
  - Whether the currently running process has executed for the maximum allowable time slice (**if yes, doing process switch**)
- I/O interrupt
  - What I/O action has occurred? If it constitutes an event, then the OS moves the corresponding blocked processes to the Ready state. The OS has to determine whether to continue executing the currently running process, or **to preempt that process for a higher-priority Ready process.**
- Memory fault
  - A virtual memory address reference is not in main memory, so the corresponding block must be brought into main memory (**It is a I/O request**)

When to Switch a Process (3) Traps and Supervisor Calls

- Trap
  - If the error or exception condition is fatal, the OS move the currently running process to Exit state.
  - If not fatal, the action of OS will depend on the error and the design of the OS.
- Supervisor call
  - The use of a system call may result in placing the user process in the Blocked state

Mode Switching (1)

- In the interrupt cycle, the processor checks to see if any interrupts have occurred
  - If no interrupts are pending, the processor proceeds to fetch cycle.
  - If an interrupt is pending, the processor do the following:
    - Saving the context of the currently running program
    - Setting the program counter to the starting address of an interrupt-handler program
    - Switching from user mode to kernel mode so that the interrupt processing code may include privileged instructions

Mode Switching (2)

- What constitutes the context that is saved?
  - It must include any information that may be altered by the execution of the interrupt handler and that will be needed to resume the program that was interrupted.
- In most operating systems, ***the occurrence of an interrupt does not necessarily mean a process switch.***

Steps of a Process Switch

1) Save the context of processor including program counter and other registers
2) Update the process control block of the process that is currently running
3) Move the process control block to appropriate queue (Ready, Blocked on Event *i*; or Ready/Suspend)
4) Select another process for execution
5) Update the process control block of the process selected
6) Update memory-management data structures (depending on how address translation is managed)
7) Restore the context of the selected process