Principles of Operating Systems

CS 446/646

2. Processes

René Doursat

Department of Computer Science & Engineering
University of Nevada, Reno

Fall 2005

Principles of Operating Systems
CS 446/646

0. Course Presentation
1. Introduction to Operating Systems
2. **Processes**
3. Memory Management
4. CPU Scheduling
5. Input/Output
6. File System
7. Case Studies

Principles of Operating Systems
CS 446/646

2. Processes

a. Process Description & Control

b. Threads

c. Concurrency

d. Deadlocks

Principles of Operating Systems

CS 446/646

2. Processes

a. **Process Description & Control**
   - ✓ What is a process?
   - ✓ Process states
   - ✓ Process description
   - ✓ Process control

b. **Threads**

c. **Concurrency**

d. **Deadlocks**

2.a Process Description & Control

What is a process?

- A process is the <u>activity</u> of executing a program

<u>Pasta for six</u>

- boil 1 quart salty water
- stir in the pasta
- cook on medium until "al dente"
- serve

thread of execution

Program

CPU

input data

Process

2.a Process Description & Control

What is a process?

1. Given that a computer system is organized into
   - ✓ hardware resources (CPU, memory, I/O, timer, disks, etc.)
   - ✓ operating system software
   - ✓ user application software
2. Given the O/S responsibility of executing applications
   - ✓ resources be made available to multiple applications
   - ✓ the CPU, in particular, be switched among multiple applications
   - ✓ the CPU and I/O devices be utilized efficiently

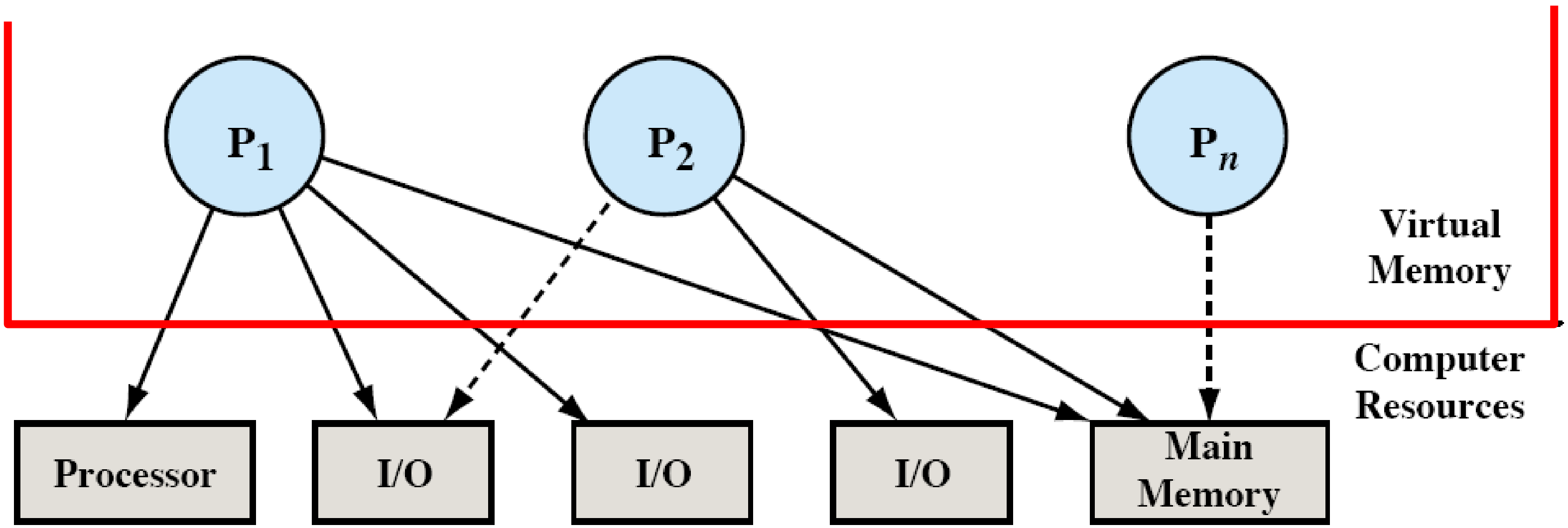
➢ . . . the approach taken by modern O/S is the “process”

- ✓ modern O/S rely on a model in which the execution of an application is abstracted into one or more **processes**

2.a Process Description & Control

What is a process?

- The O/S has to multiplex resources to the processes
  - ✓ a number of processes have been created
  - ✓ each process during the course of its execution needs access to system resources: CPU, main memory, I/O devices



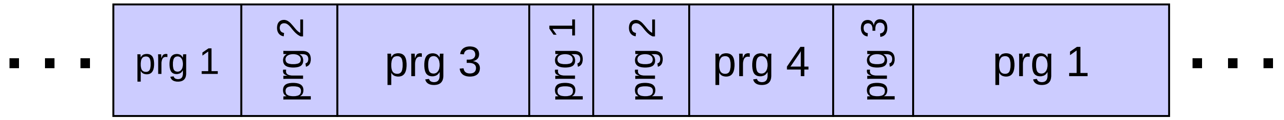
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Resource allocation for processes (one snapshot in time)

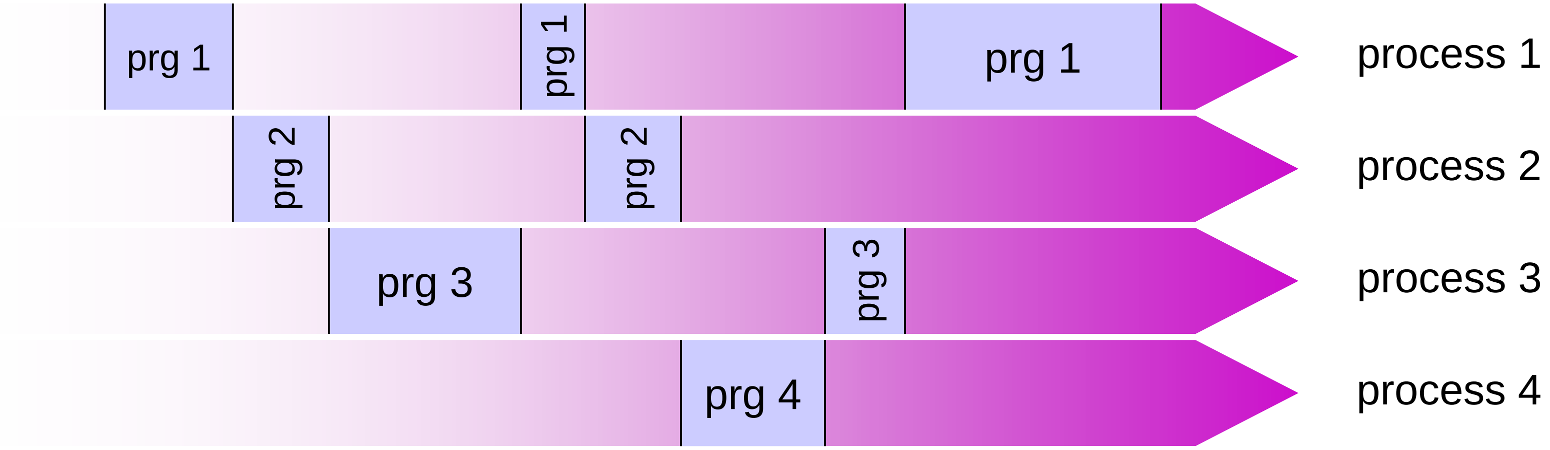
2.a Process Description & Control

What is a process?

- Multitasking can be conveniently described in terms of multiple processes running in (pseudo)parallel



(a) Multitasking from the CPU's viewpoint



(b) Multitasking from the processes' viewpoint = 4 virtual program counters

Pseudoparallelism in multitasking

2.a Process Description & Control

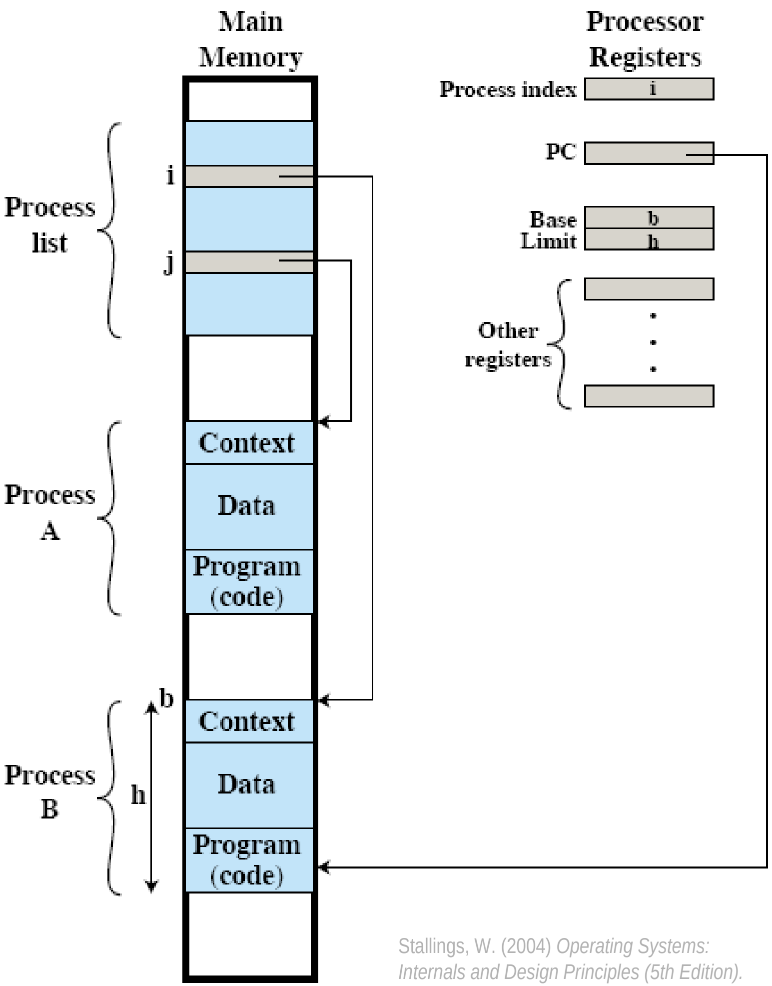
What is a process?

- A process image consists of three components

user address space:

1. an executable program
2. the associated data needed by the program

3. the execution context of the process, which contains all information the O/S needs to manage the process (ID, state, CPU registers, stack, etc.)



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

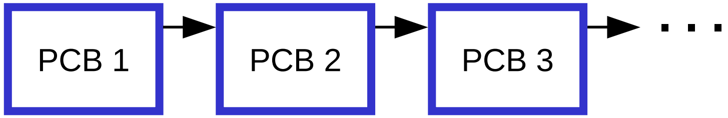
Typical process image implementation

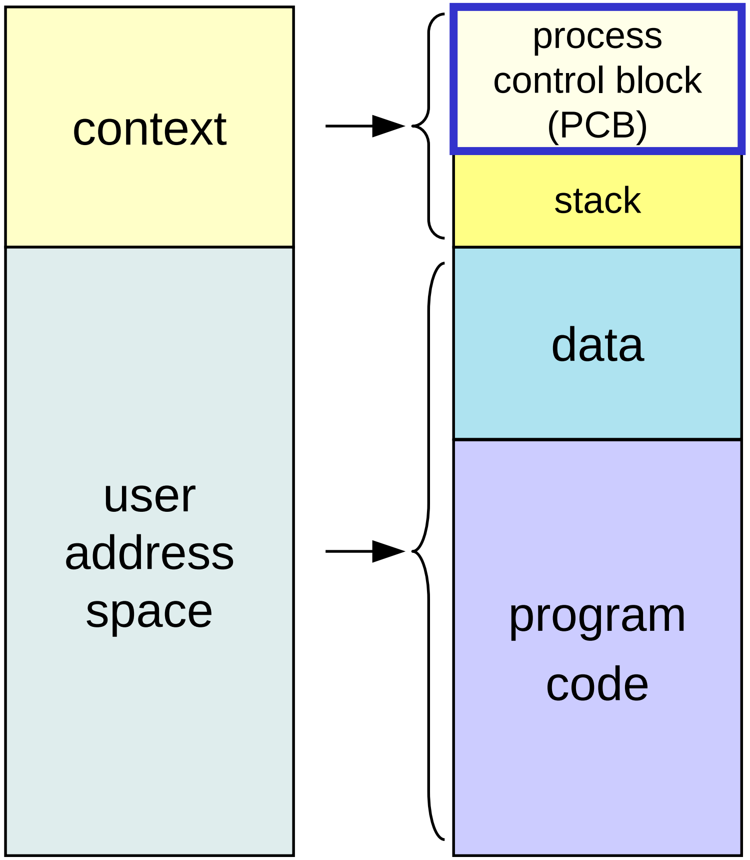
2.a Process Description & Control

What is a process?

➢ The Process Control Block (PCB)

- ✓ is included in the context, along with the stack
- ✓ is a "snapshot" that contains all necessary and sufficient data to restart a process where it left off (ID, state, CPU registers, etc.)
- ✓ is one entry in the operating system's **process table** (array or linked list)





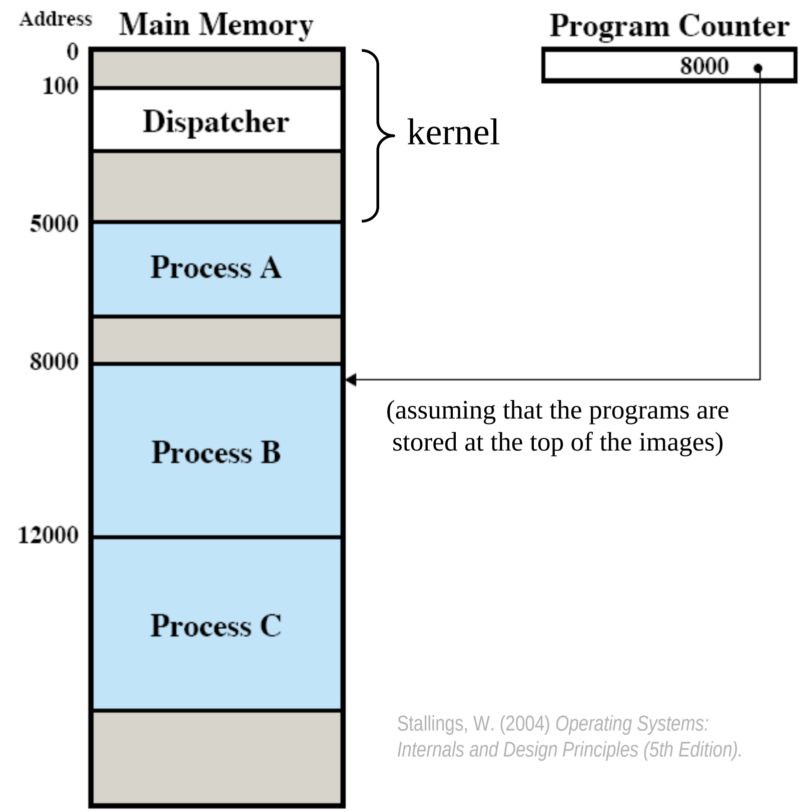
Typical process image implementation

2.a Process Description & Control

What is a process?

- A dispatcher switches the CPU between processes
  - ✓ the dispatcher is a routine program in kernel memory space

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



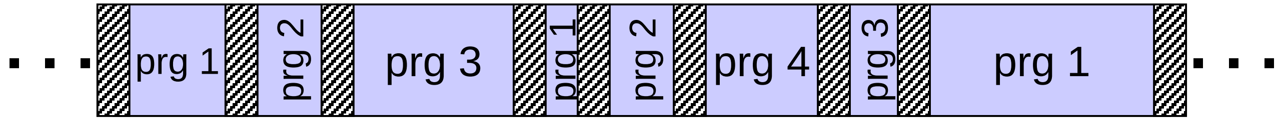
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Dispatching between three processes

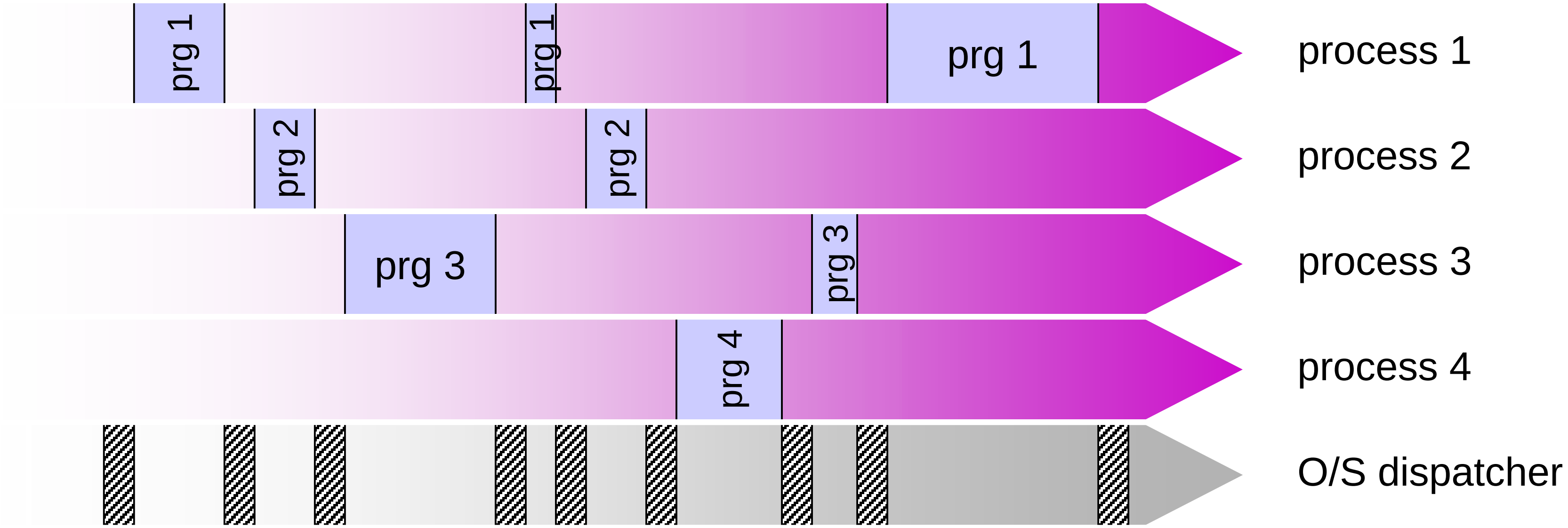
2.a Process Description & Control

What is a process?

- A dispatcher switches the CPU between processes
  - ✓ the dispatcher is a routine program in kernel memory space



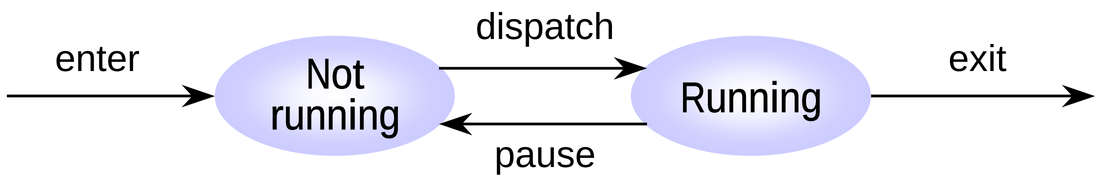
(a) Multitasking from the CPU's viewpoint



2.a Process Description & Control

Process states

- Deep truth: at any time, a given process is either being executed by the CPU or it is not
  - ✓ thus, a process can have two states: running or not running



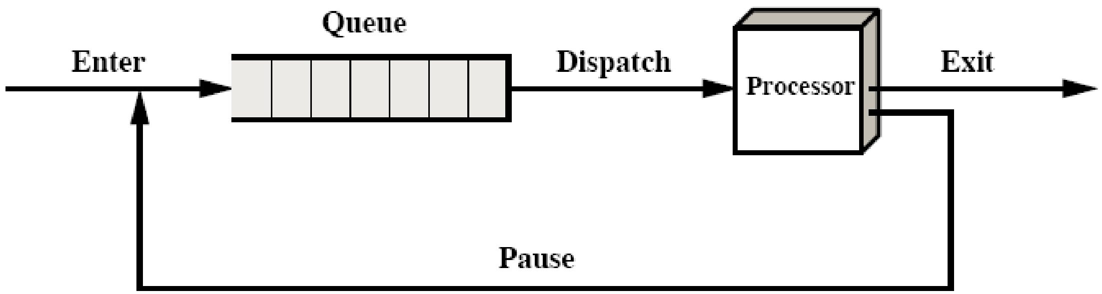
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Transition diagram of a two-state process model

2.a Process Description & Control

Process states

- ➢ How does the O/S keep track of processes and states?
  - ✓ by keeping a queue of pointers to the process control blocks



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

  - ✓ the queue can be implemented as a linked list if each PCB contains a pointer to the next PCB

Queuing diagram of a two-state process model

2.a Process Description & Control

Process states

- Some events that lead to process <u>creation</u> (enter)

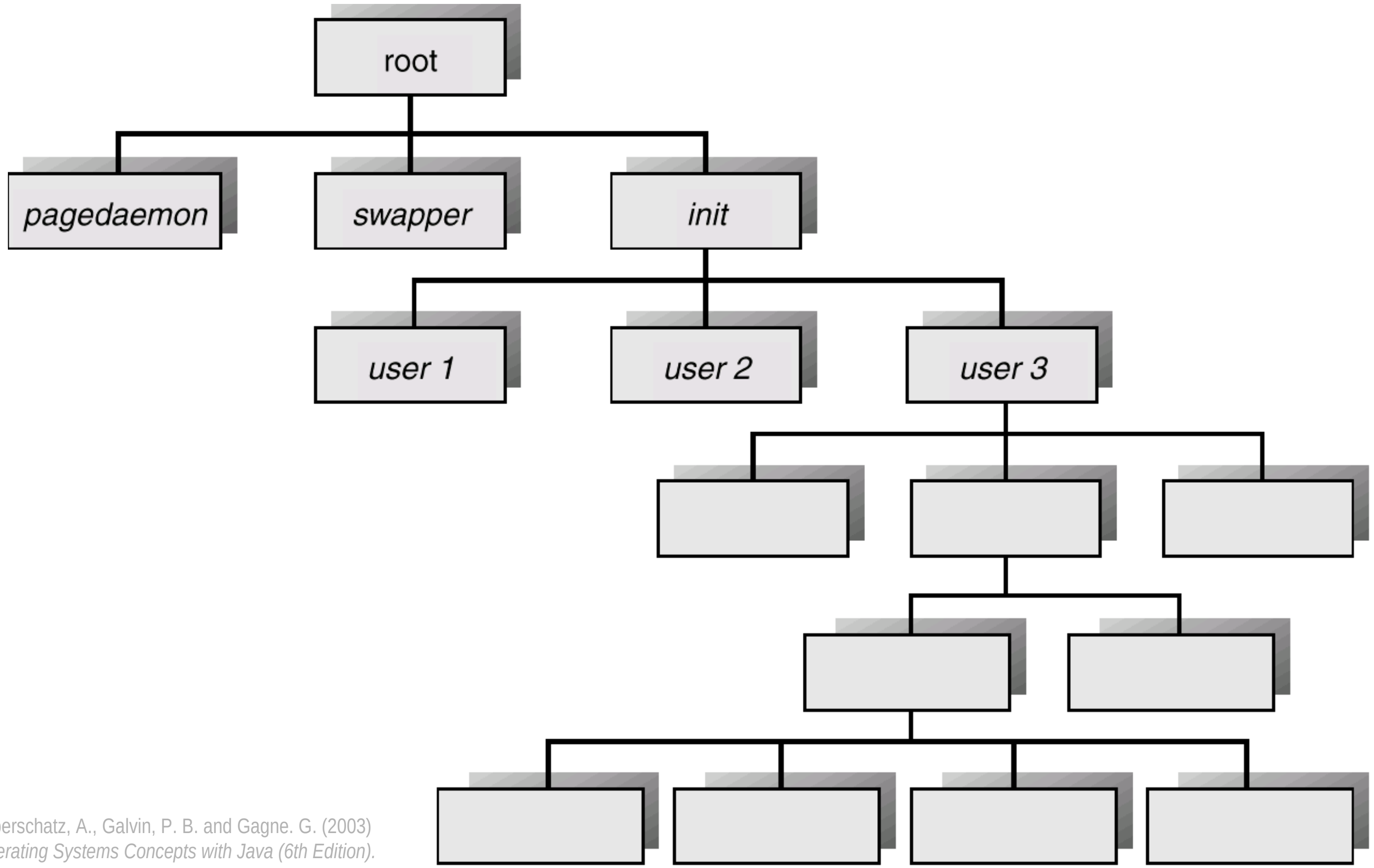
all cases of process spawning

- ✓ the system boots
  - when a system is initialized, several background processes or "daemons" are started (email, logon, etc.)
- ✓ a user requests to run an application
  - by typing a command in the CLI shell or double-clicking in the GUI shell, the user can launch a new process
- ✓ an existing process spawns a child process
  - for example, a server process (print, file) may create a new process for each request it handles
  - the *init* daemon waits for user login and spawns a shell
- ✓ a batch system takes on the next job in line

2.a Process Description & Control

Process states

- Process creation by spawning



Silberschatz, A., Galvin, P. B. and Gagne. G. (2003)
Operating Systems Concepts with Java (6th Edition).

A tree of processes on a typical UNIX system

2.a Process Description & Control

Process states

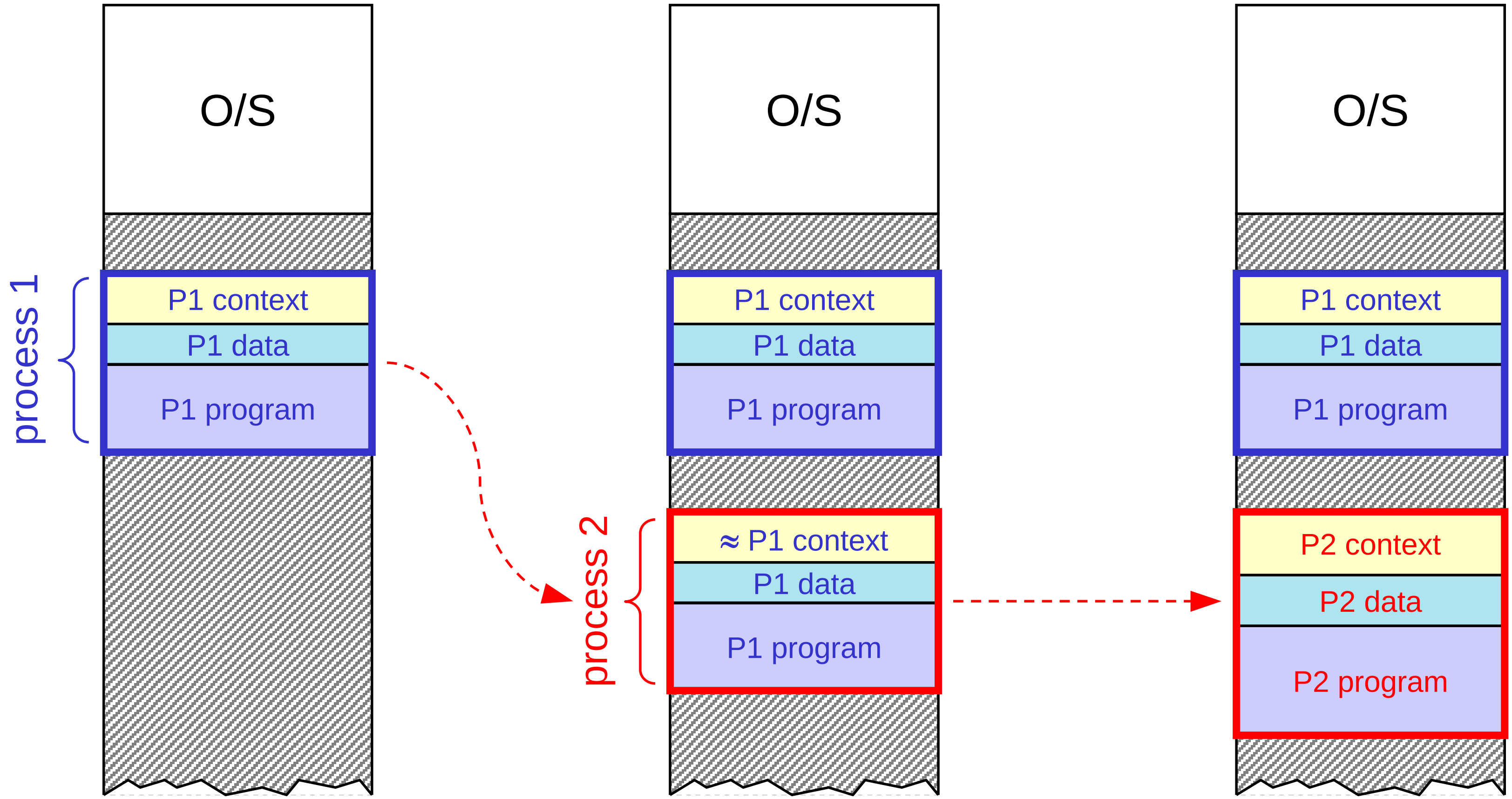
```
...
int main(...)
{
    ...
    if ((pid = fork()) == 0)                        // create a process
    {
        fprintf(stdout, "Child pid: %i\n", getpid());
        err = execvp(command, arguments);           // execute child
                                                    //   process
        fprintf(stderr, "Child error: %i\n", errno);
        exit(err);
    }
    else if (pid > 0)                               // we are in the
    {                                               //   parent process
        fprintf(stdout, "Parent pid: %i\n", getpid());
        pid2 = waitpid(pid, &status, 0);            // wait for child
        ...                                         //   process
    }
    ...

    return 0;
}
```

Implementing a shell command interpreter by process spawning

2.a Process Description & Control

Process states

1. Clone child process
   - ✓ `pid = fork()`
2. Replace child's image
   - ✓ `execve(name, ...)`



2.a Process Description & Control
Process states

- **Some events that lead to process termination (exit)**
  - ✓ regular completion, with or without error code
    - process-triggered: the process voluntarily executes an `exit(err)` system call to indicate to the O/S that it has finished
  - ✓ fatal error (uncatchable or uncaught)
    - O/S-triggered (following system call or preemption):
      - service errors: no memory left for allocation, I/O error, etc.
      - total time limit exceeded
    - hardware interrupt-triggered:
      - arithmetic error, out-of-bounds memory access, etc.
  - ✓ killed by another process via the kernel
    - software interrupt-triggered:
      - the process receives a `SIGKILL` signal
      - in some systems the parent takes down its children with it

2.a Process Description & Control

Process states

- Some events that lead to process <u>pause</u> / <u>dispatch</u>
  - ✓ I/O wait
    - O/S-triggered (following system call) ▪ a process invokes an I/O system call that blocks waiting for the I/O device: the O/S puts the process in “Not Running” mode and dispatches another process to the CPU
  - ✓ preemptive timeout
    - hardware interrupt-triggered (timer) ▪ the process receives a timer interrupt and relinquishes control back to the O/S dispatcher: the O/S puts the process in “Not Running” mode and dispatches another process to the CPU
    - ▪ not to be confused with “total time limit exceeded”, which leads to process termination

2.a Process Description & Control

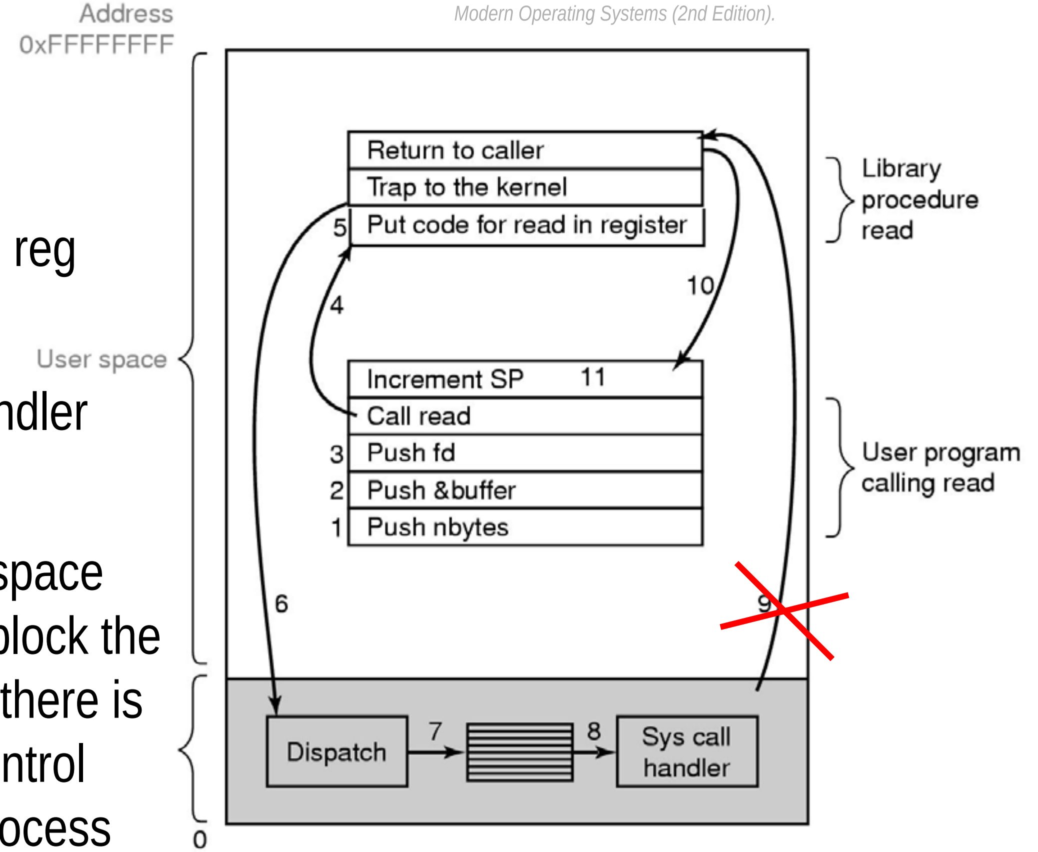
Process states

- Steps in making a system call that must wait for I/O

  1. – 3. . . . program prepares stack
  4. . . . program calls **read**
  5. . . . **read** stores **#read** in reg
  6. . . . **read** executes **TRAP**
  7. . . . kernel dispatches to call handler
  8. . . . system call handler runs
  9. control does not return to user space right away; the O/S decides to block the caller ("Not Running") because there is no input to read yet; instead, control eventually returns to another process

  → *not just mode switch: full process switch!*

Tanenbaum, A. S. (2001) *Modern Operating Systems (2nd Edition).*

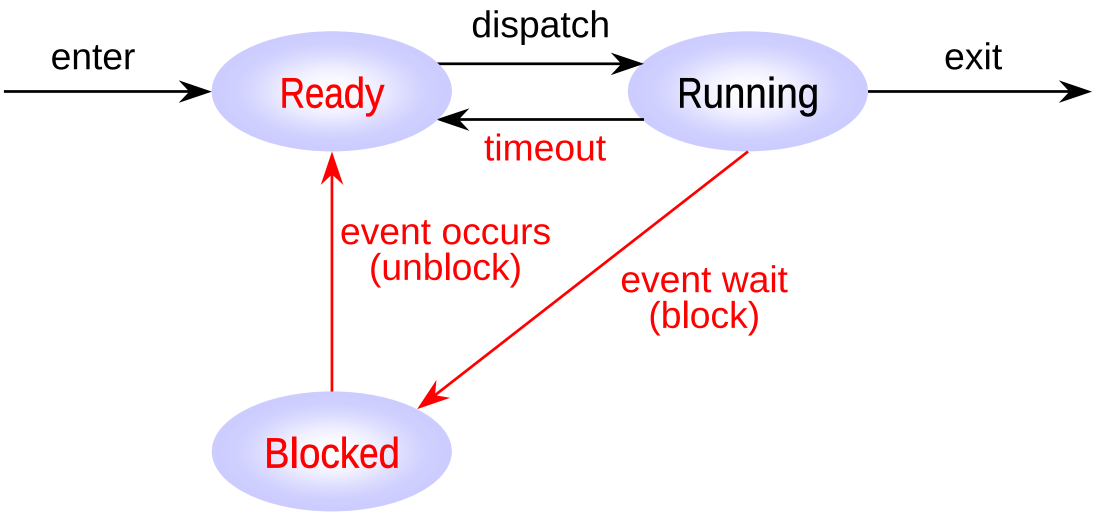


11 steps in making a system call

2.a Process Description & Control

Process states

- Problem with the two-state model
  - ✓ some “Not Running” processes are blocked (waiting for I/O, etc.)
  - ✓ the O/S wastes time scanning the queue for ready processes



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

→ solution: divide “Not Running” into “Ready” and “Blocked”

Transition diagram of a three-state (“Blocked/Ready”) process model

2.a Process Description & Control

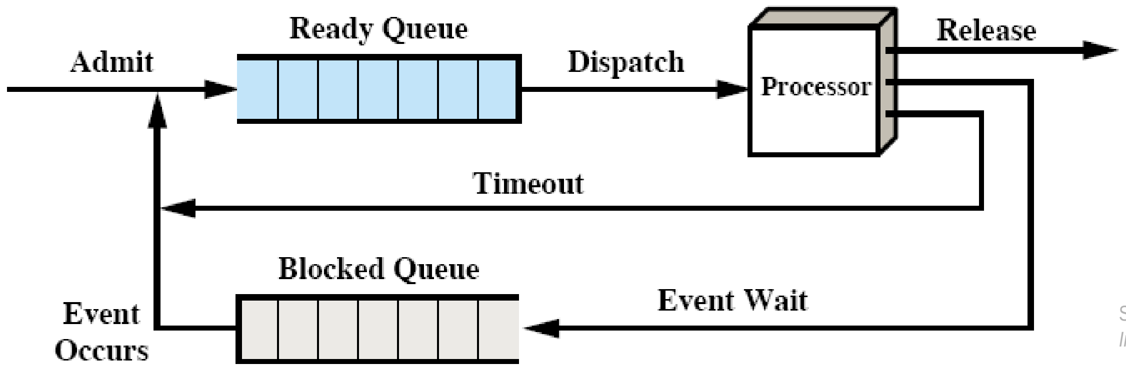
Process states

- Some events that lead to process <u>timeout</u> / <u>dispatch</u>
  <u>block</u> / <u>unblock</u>
  - ✓ I/O wait
    - O/S-triggered (following system call) ▪ a process invokes an I/O system call that blocks waiting for the I/O device: the O/S puts the process in "Blocked" mode and dispatches another process to the CPU
  - ✓ preemptive timeout
    - hardware interrupt-triggered (timer) ▪ the process receives a timer interrupt and relinquishes control back to the O/S dispatcher: the O/S puts the process in "Ready" mode and dispatches another process to the CPU
    - ▪ not to be confused with "total time limit exceeded", which leads to process termination

2.a Process Description & Control

Process states

- How does the O/S keep track of three process states?
  - ✓ by keeping an extra queue for blocked processes



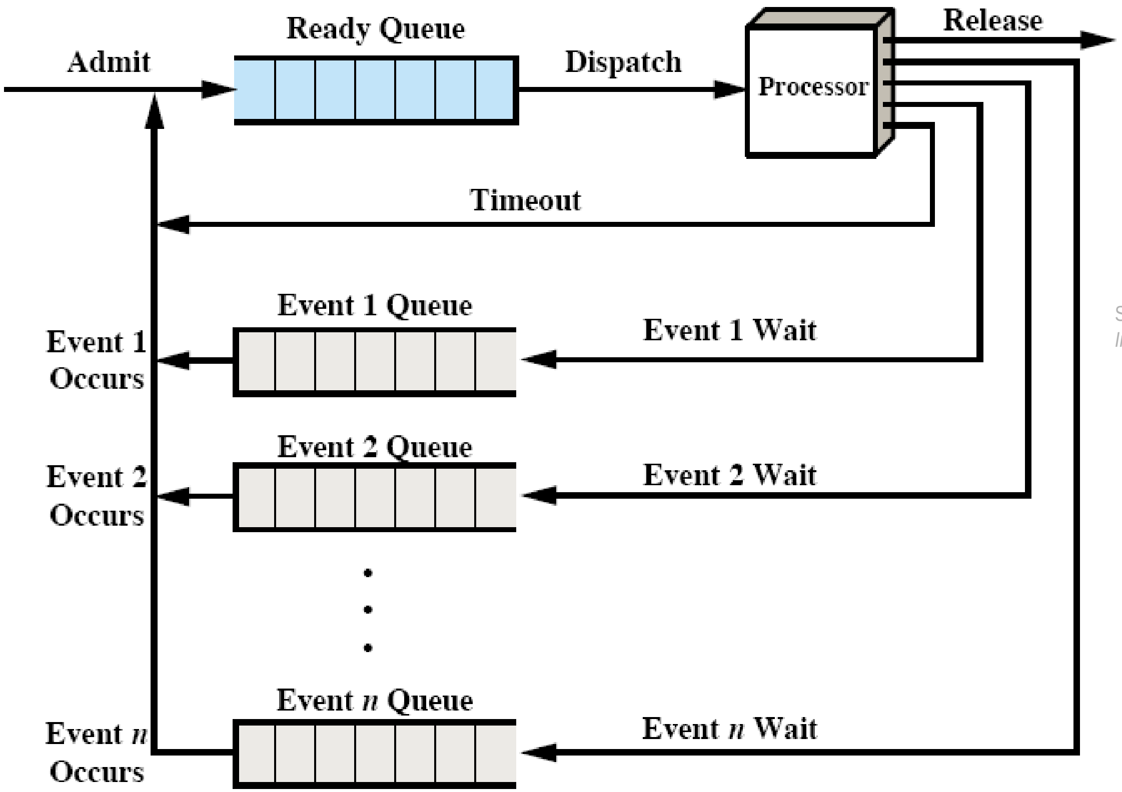
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Queuing diagram of a three-state ("Blocked/Ready") process model

2.a Process Description & Control

Process states

- To further reduce scanning, blocked processes can be placed in separate queues depending on the event type



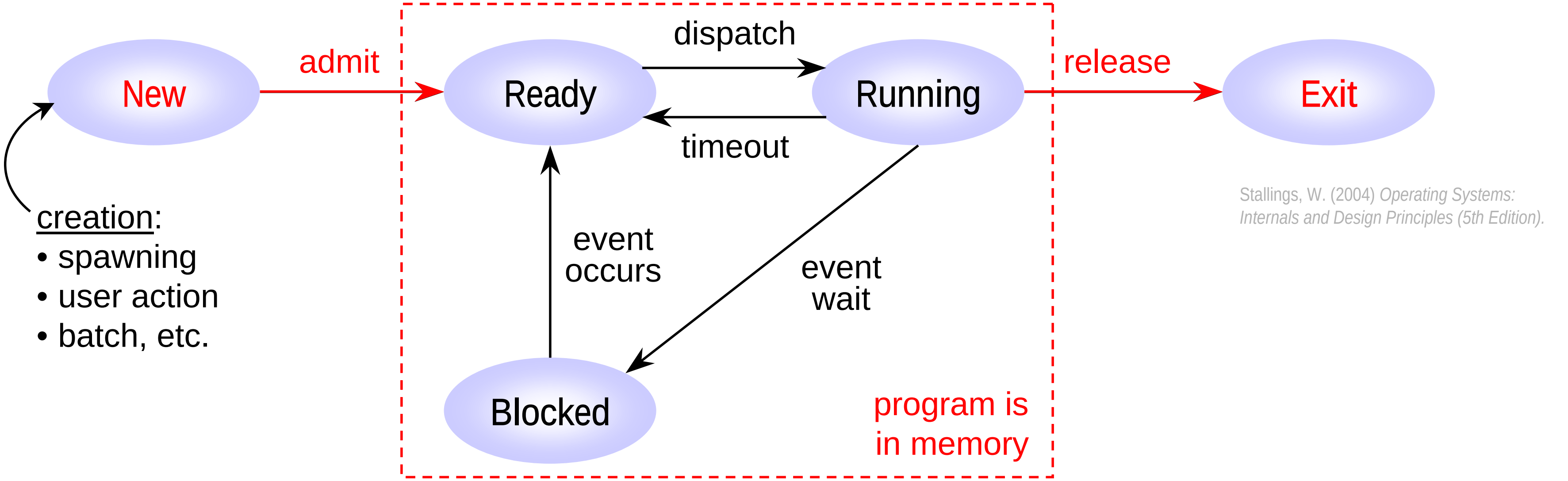
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Queuing diagram of a three-state ("Blocked/Ready") process model with multiple event queues

2.a Process Description & Control

Process states

- How is a process actually created (entered)?
  - ✓ in two steps: first the PCB is created and put in a "New" pool
  - ✓ then, program & data are loaded and the process is "Ready"



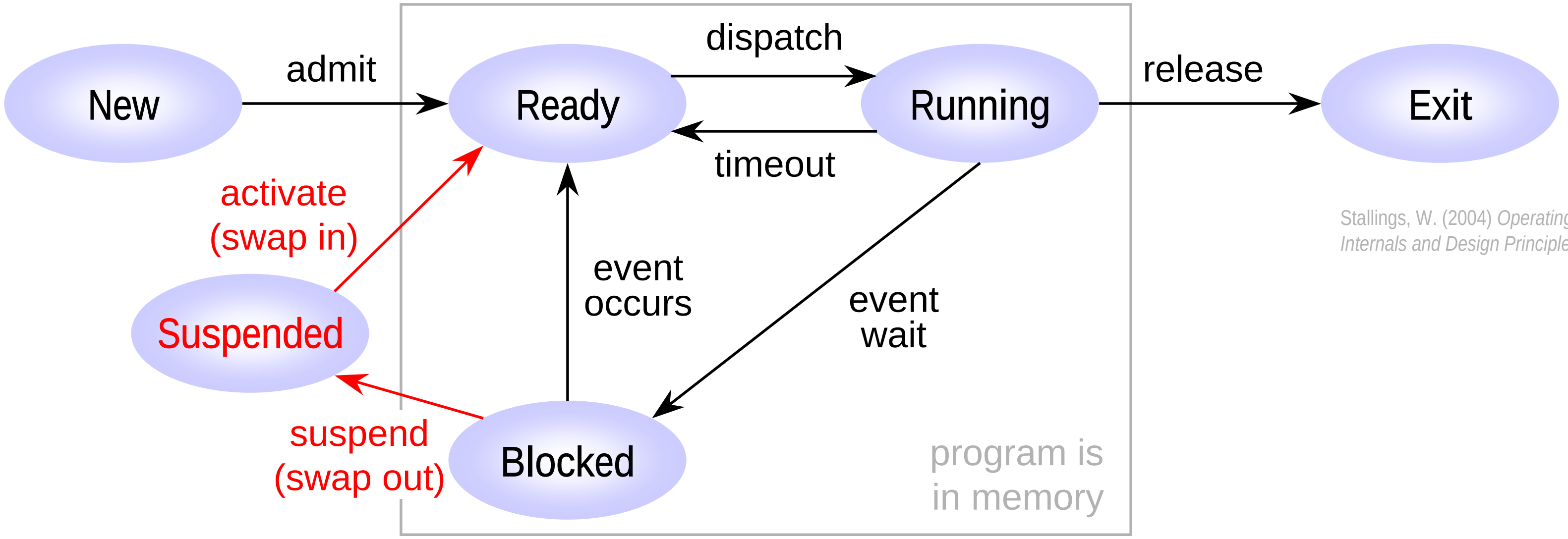
  - ✓ conversely with termination: first, program & data are swapped out, while the PCB is retained in an "Exit" pool, then removed

Transition diagram of a five-state (New/Exit) model

2.a Process Description & Control

Process states

- **Problems with the "Blocked/Ready" model**
  - ✓ blocked processes are taking up memory space
  - ✓ a hungry CPU might soon run out of ready processes in memory



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

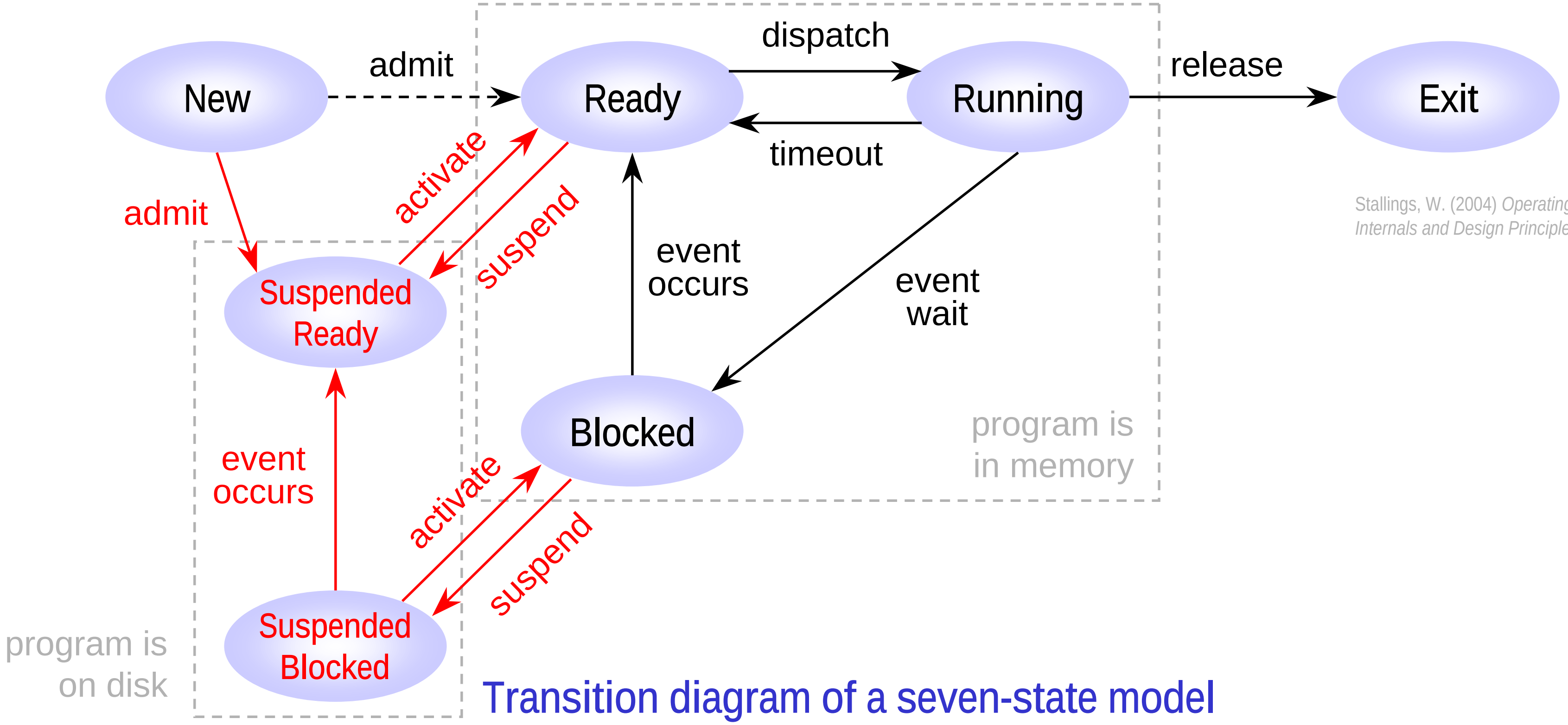
→ solution: swap processes out of memory and put them into a "Suspended" state

Transition diagram of a six-state ("Suspended") model

2.a Process Description & Control

Process states

- ➢ Last problem with the "Suspended" model
  - ✓ why swap in a suspended process that was blocked anyway?
  - → solution: add a "Suspended Ready" state



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Transition diagram of a seven-state model

2.a Process Description & Control

Process states

- Two independent concepts × two values each
  - ✓ whether a process is waiting on an event (is “Blocked”) or not
  - ✓ whether a process has been swapped out of main memory (is “Suspended”) or not
- = Four combined states
  - ✓ “Ready”: the process is in memory and available for execution
  - ✓ “Blocked”: the process is in main memory awaiting an event
  - ✓ “Suspended Blocked”: the process is in secondary memory and awaiting an event
  - ✓ “Suspended Ready”: the process is in secondary memory but is available for execution as soon as it is loaded into memory

2.a Process Description & Control

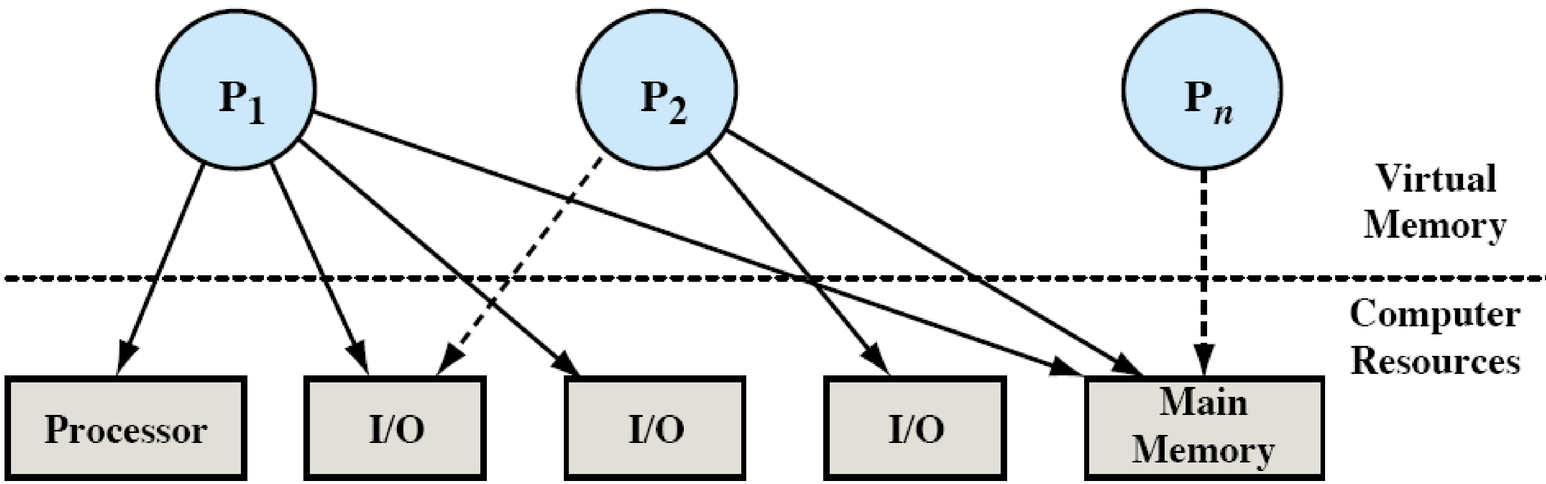
Process states

Note: Release of memory by swapping is not the only motivation for suspending processes. Various background processes may also be turned off and on, depending on CPU load, suspicion of a problem, some periodical timer or by user request.

2.a Process Description & Control

Process description

- The O/S has to multiplex resources to the processes
  - ✓ a number of processes have been created
  - ✓ each process during the course of its execution needs access to system resources: CPU, main memory, I/O devices



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Resource allocation for processes (one snapshot in time)

2.a Process Description & Control

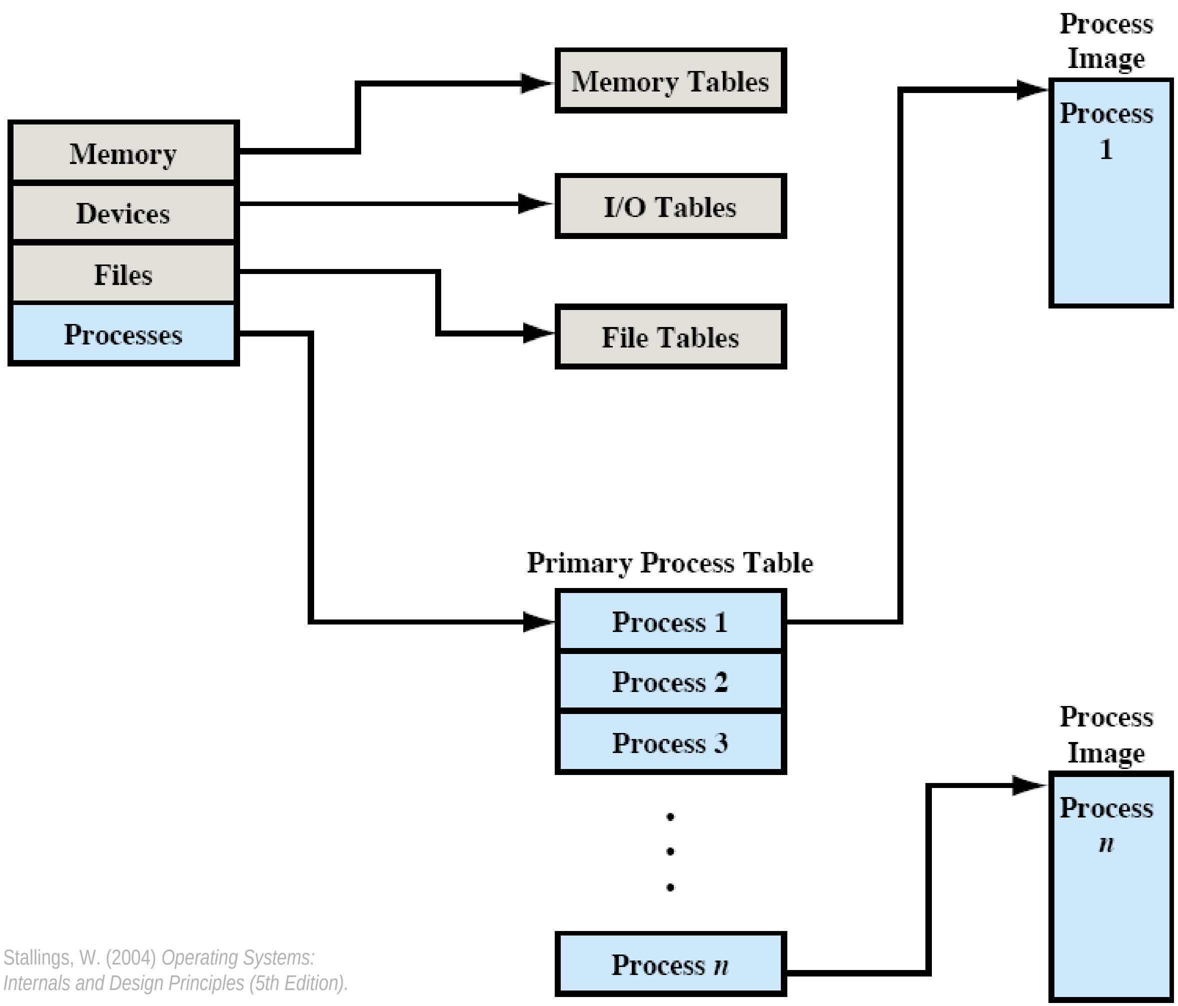
Process description

- To do this, the O/S must be a zealous bureaucrat keeping all sorts of tables
  - ✓ **memory tables** – what part of memory is currently reserved for what process
  - ✓ **I/O tables** – what I/O device is currently assigned to what process
  - ✓ **file tables** – what file is currently opened by what process
  - ✓ **process tables** – what are the processes running, blocked, suspended, etc.
- Naturally, these tables are cross-referenced in many ways

Carmen Tomfohrde - *Three-ring binders*

2.a Process Description & Control

Process description



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

General structure of an operating system's control tables

2.a Process Description & Control

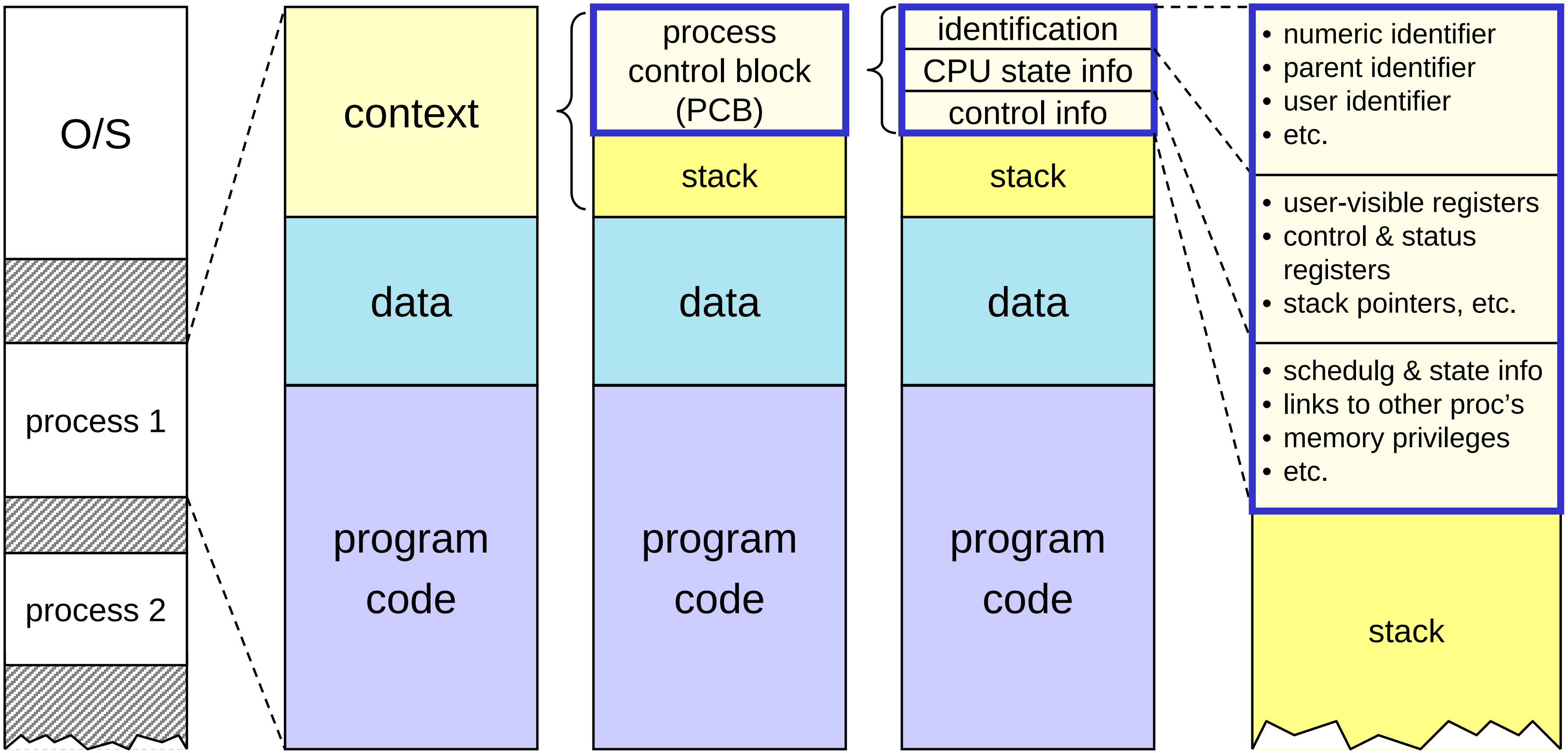
Process description

- In the process table, the O/S keeps one ID structure per process, the *Process Control Block* (PCB), containing:
  - ✓ process identification data
    - numeric identifiers of the process, the parent process, the user, etc.
  - ✓ CPU state information
    - user-visible, control & status registers
    - stack pointers
  - ✓ process control information
    - scheduling: state, priority, awaited event
    - used memory and I/O, opened files, etc.
    - pointer to next PCB

2.a Process Description & Control

Process description

- Example of process and PCB location in memory



Illustrative contents of a process image in (virtual) memory

2.a Process Description & Control

Process description

Note: In reality, depending on the specific O/S:

- PCB, stack, and user address space may be laid out in a different order
- within user space, data and program may be mixed.

Moreover:

- the process image may not be present in physical memory in its entirety
- the portion of process image in memory may not be contiguous, but distributed over disjoint address areas (“pages”).

We will meet the last two concepts again when we study **virtual memory**.

2.a Process Description & Control

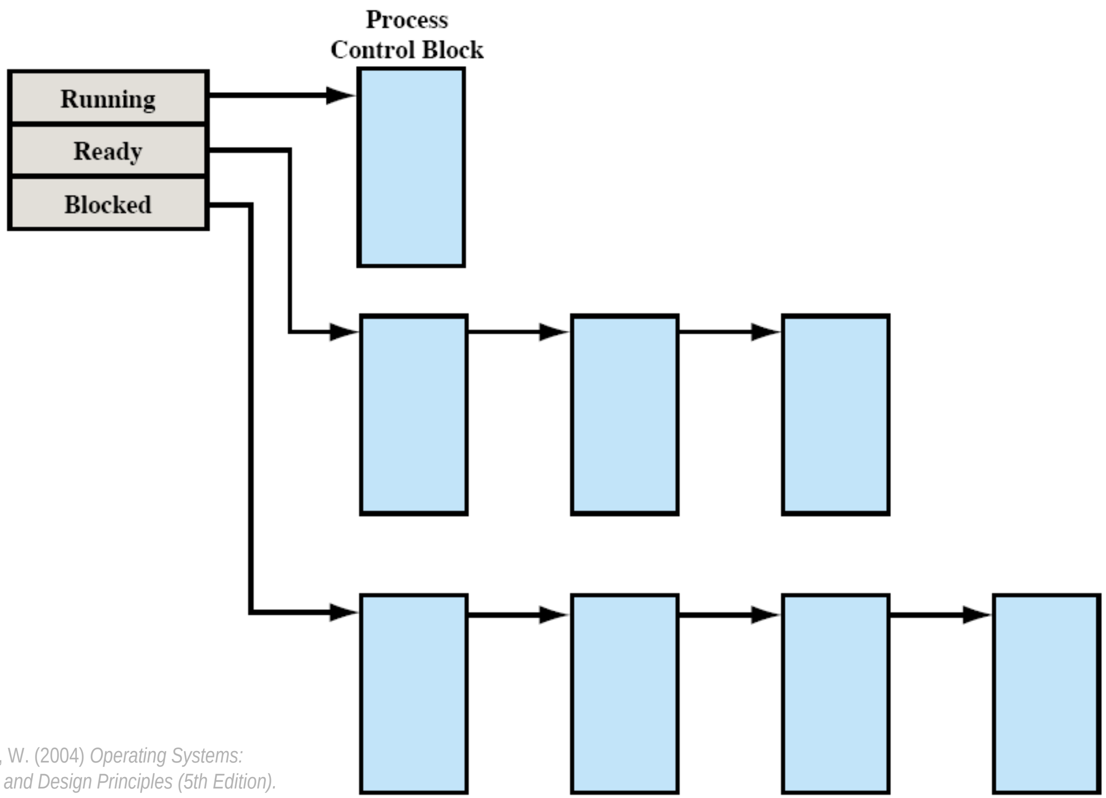
Process description

- The PCB is the most important O/S data structure
  - ✓ the set of PCBs (the process table) practically defines the state of the O/S
  - ✓ PCBs must be read/modified all the time by almost all modules in the O/S: scheduler, resource allocator, interrupt handler, performance monitor, etc.
  - ✓ therefore it is a good design practice to dedicate one low-level handler (“clerk”) to the protection of the process table; then, the modules must ask this handler for any read/write access
  - ✓ we have seen this design pattern before: encapsulate a critical resource in a service layer or module for better control and orderly access; this is the whole story of an O/S!

2.a Process Description & Control

Process description

- The process table can be split into per-state queues
  - ✓ PCBs can be linked together if they contain a pointer field



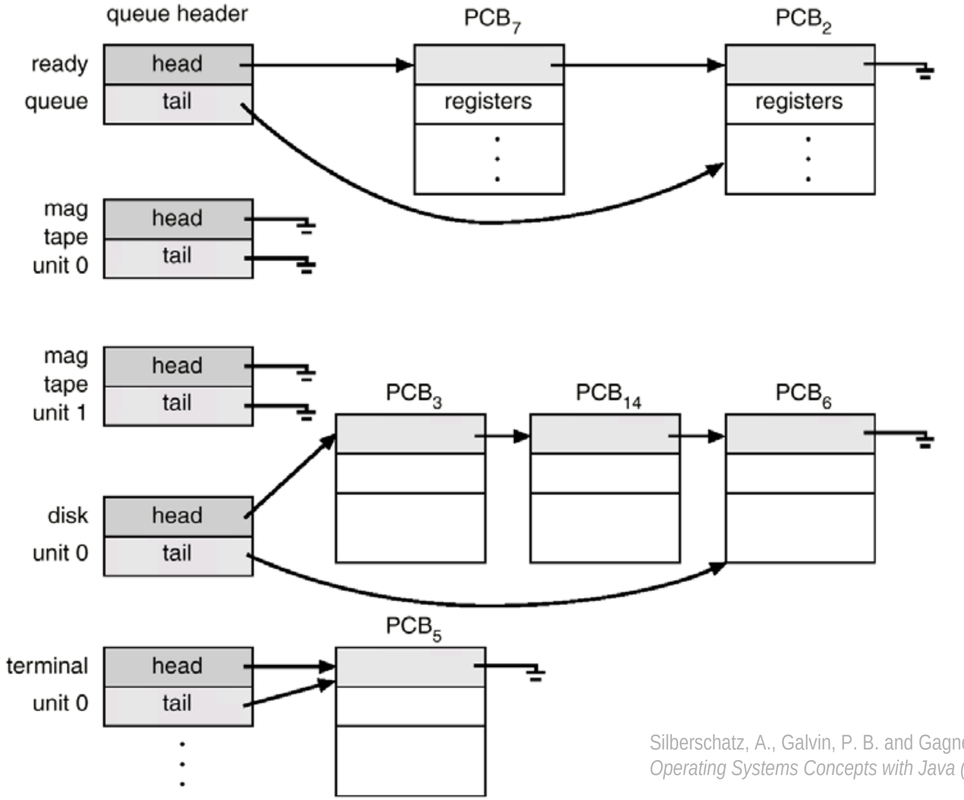
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Structure of process lists or queues

2.a Process Description & Control

Process description

- The blocked processes can themselves be split into device-specific queues



Silberschatz, A., Galvin, P. B. and Gagne. G. (2003)
Operating Systems Concepts with Java (6th Edition).

Various I/O device queues

2.a Process Description & Control

Process description

```
struct task_struct
{
    volatile long state;          /* -1 unrunnable, 0 runnable, >0 stopped */
    unsigned long flags;          /* per process flags, defined below */
    ...
    struct mm_struct *mm;         /* memory */
    ...
    struct task_struct *next_task, *prev_task;   /* linked list */
    ...
    struct linux_binfmt *binfmt;  /* task state */
    int exit_code, exit_signal;
    ...
    pid_t pid;                    /* process ID */
    pid_t pgrp;                   /* process group ID */
    ...
    /*
     * pointers to parent process, youngest child, younger sibling,
     * older sibling, respectively.
     */
    struct task_struct *p_opptr, *p_pptr, *p_cptr, *p_ysptr, *p_osptr;
    ...
    struct thread_struct thread;  /* CPU-specific state of this task */
    ...
    struct files_struct *files;   /* open file information */
    ...
}
```

Sample of the PCB data structure **task_struct** in Linux   http://lxr.linux.no

2.a Process Description & Control

Process control

➢ How is a process created by the O/S, step by step?

1. a unique identifier is assigned to the new process
   - one new entry is added to the primary process table
2. memory space is allocated for the process
   - this includes program (with linkages), data, stack and PCB
3. the PCB is constructed and initialized
   - ID, state = “Ready”, CPU state = empty, resources = none
4. the PCB is placed in the appropriate queue (linked list)
5. other O/S modules are notified about the new process
   - create or expand other data structures to accommodate info about the new process

2.a Process Description & Control

Process control

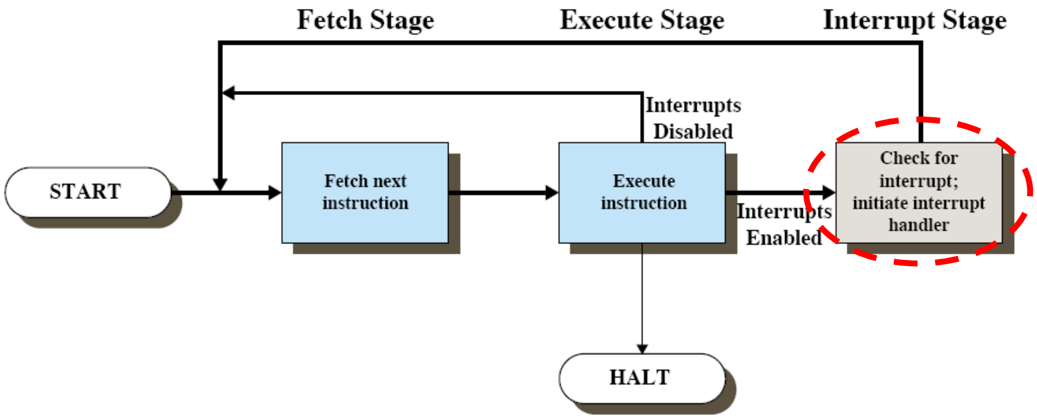
- What events trigger the O/S to switch processes?
  - ✓ **interrupts** — external, <u>asynchronous</u> events, independent of the currently executed process instructions
    - clock interrupt → O/S checks time and may block process
    - I/O interrupt → data has come, O/S may unblock process
    - memory fault → O/S may block process that must wait for a missing page in memory to be swapped in

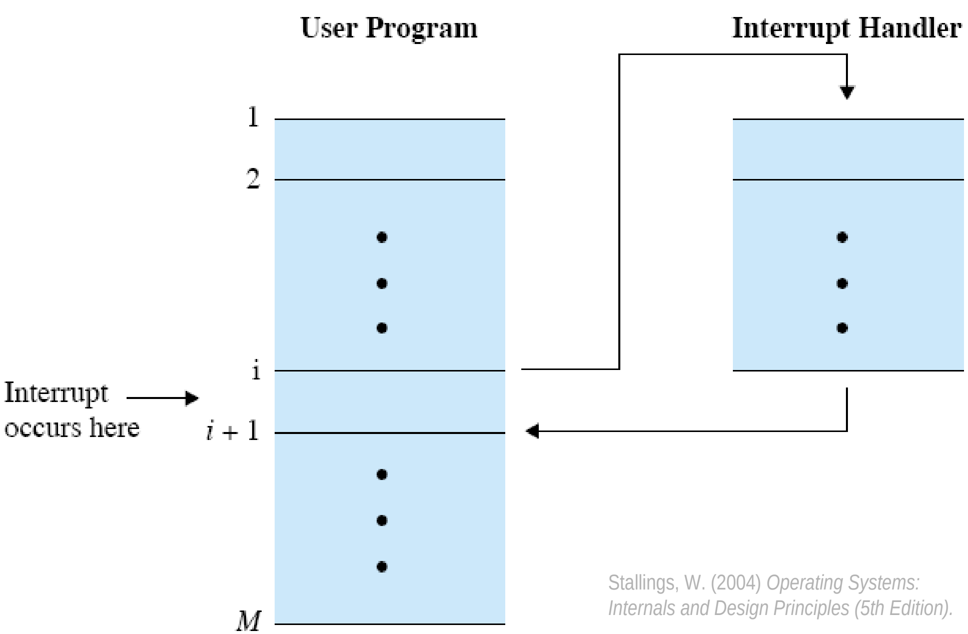
traps {
  - ✓ **exceptions** — internal, <u>synchronous</u> (but involuntary) events caused by instructions → O/S may terminate or recover process
  - ✓ **system calls** — voluntary <u>synchronous</u> events calling a specific O/S service → after service completed, O/S may either resume or block the calling process, depending on I/O, priorities, etc.

2.a Process Description & Control

Process control

- Interrupts or traps
  - ✓ are caught in a third stage of the fetch/ execute cycle and
  - ✓ transfer control (PC) to an interrupt handler in kernel space,
  - ✓ which branches to O/S routines specific to types of interrupts;
  - ✓ the CPU is eventually returned to this user program . . . or another



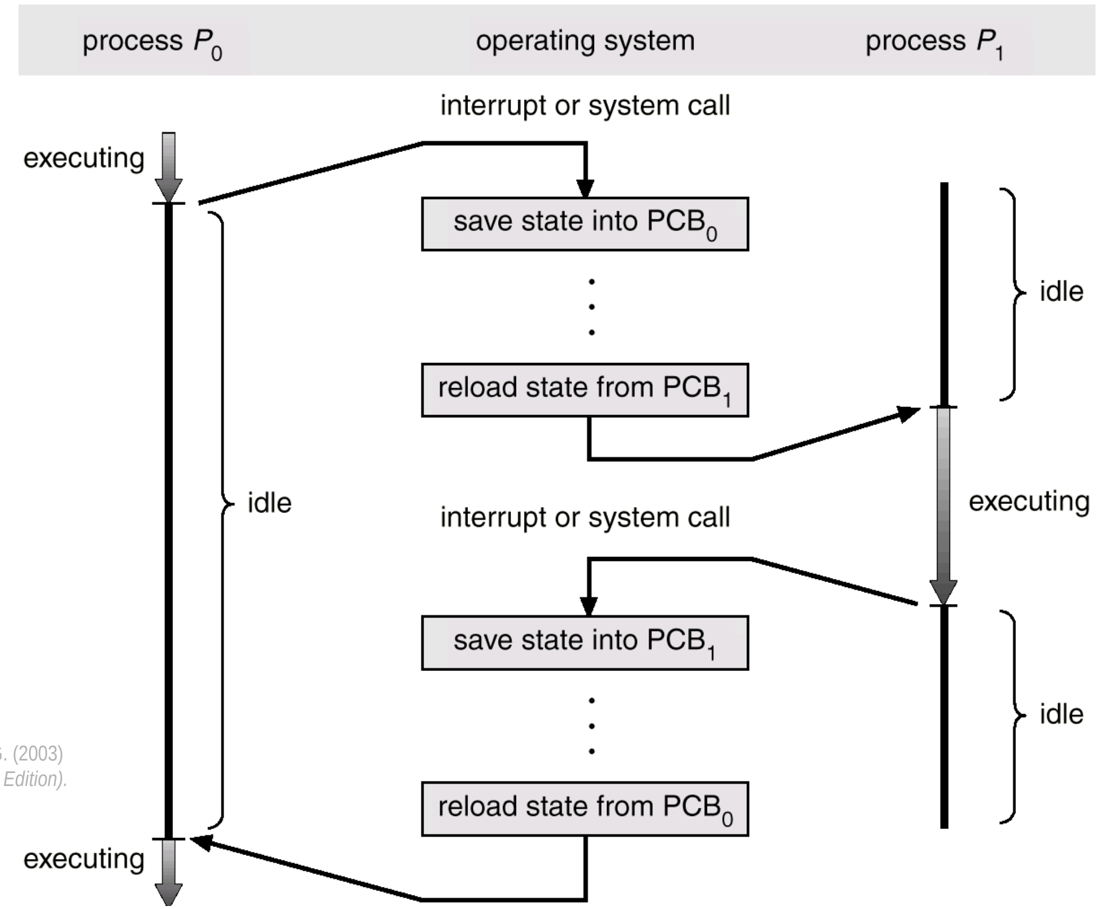


Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

2.a Process Description & Control

Process control

- Process switch



Silberschatz, A., Galvin, P. B. and Gagne. G. (2003)
Operating Systems Concepts with Java (6th Edition).

CPU switch from process to process

2.a Process Description & Control

Process control

- Mode switching ≠ process switching
  - ✓ when handling an interrupt, execution is always switched from user mode to kernel mode ("mode switch")
  - ✓ but this is independent from whether the O/S will return control to the interrupted process or another process ("process switch")

1. if control (execution) eventually returns to the interrupted process, for example after a nonblocking system call:
   - only the CPU state information (PC, registers, stack info) needed to be saved; this was initiated by the hardware
2. if control eventually passes to another process, for example after a blocking call, interrupt or trap:
   - the whole PCB is saved; this is done by the O/S scheduler

2.a Process Description & Control

Process control

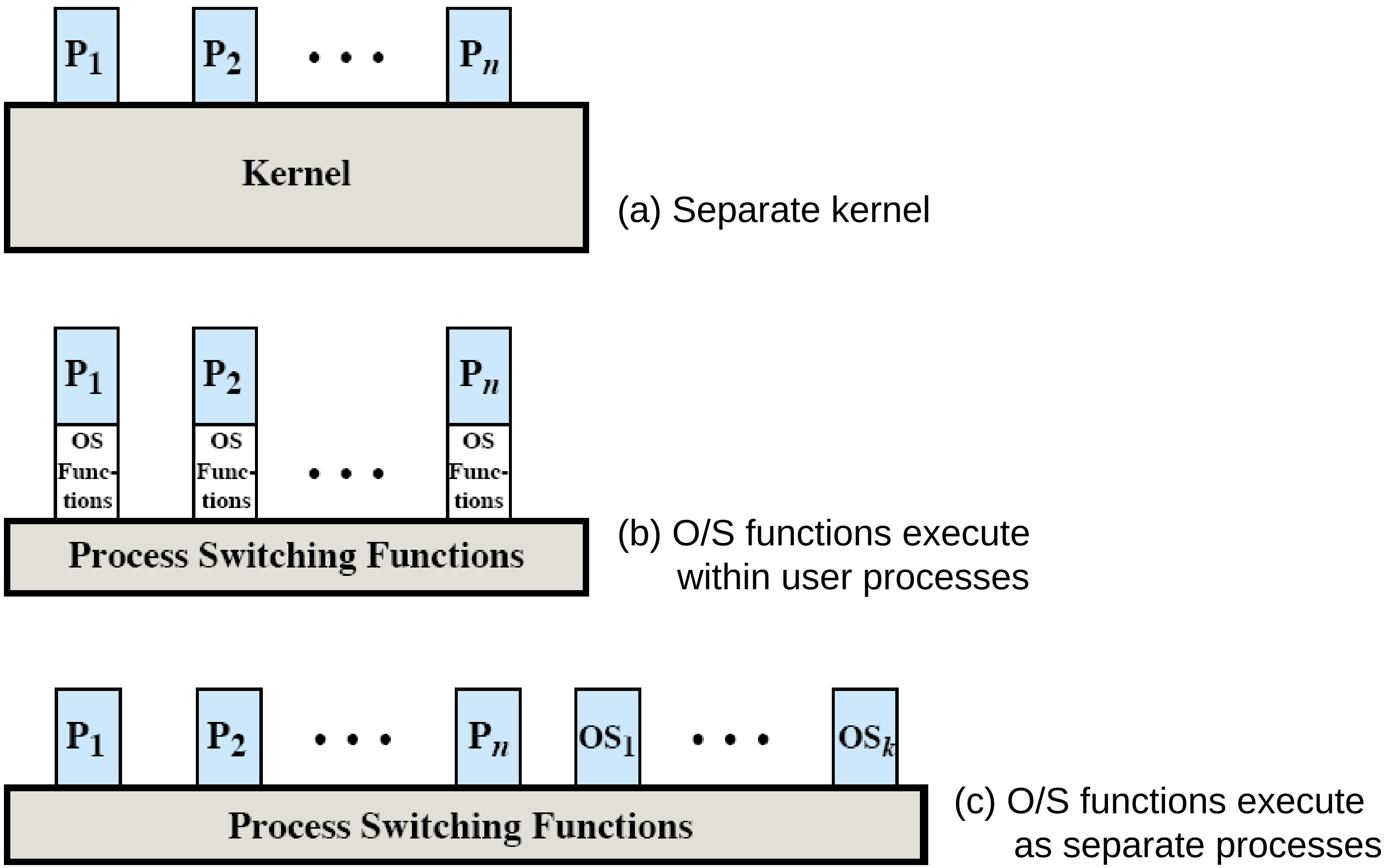
- How does a full process switch happen, step by step?

1. save CPU context, including PC and registers *(the only step needed in a simple mode switch)*
2. update process state (to “Ready”, “Blocked”, etc.) and other related fields of the PCB
3. move the PCB to the appropriate queue
4. select another process for execution: this decision is made by the CPU scheduling algorithm of the O/S
5. update the PCB of the selected process (state = “Running”)
6. update memory management structures
7. restore CPU context to the values contained in the new PCB

2.a Process Description & Control

Process control

- How is the O/S itself executed? Is it a process, too?



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Relationship between O/S execution and user processes

2.a Process Description & Control

Process control

- Possible designs for the execution of the O/S itself
  - ✓ nonprocess kernel (traditional approach in older O/S)
    - simple mode switch; kernel executes in own region of memory with own stack, outside of any process (i.e., no associated PCB); the only program that is not a “process”
  - ✓ O/S functions execute within each user process (most PCs)
    - the O/S is a collection of routines that can be “attached” to the processes in memory via shared address space
    - only the mode is switched, the current process (which executes user program + kernel program) continues to run
  - ✓ O/S functions execute as full, separate processes (microkernels)
    - modular O/S with clean, minimal interfaces

Principles of Operating Systems

CS 446/646

2. Processes

a. **Process Description & Control**
   - ✓ What is a process?
   - ✓ Process states
   - ✓ Process description
   - ✓ Process control

b. **Threads**

c. **Concurrency**

d. **Deadlocks**