Chapter 3
Processes

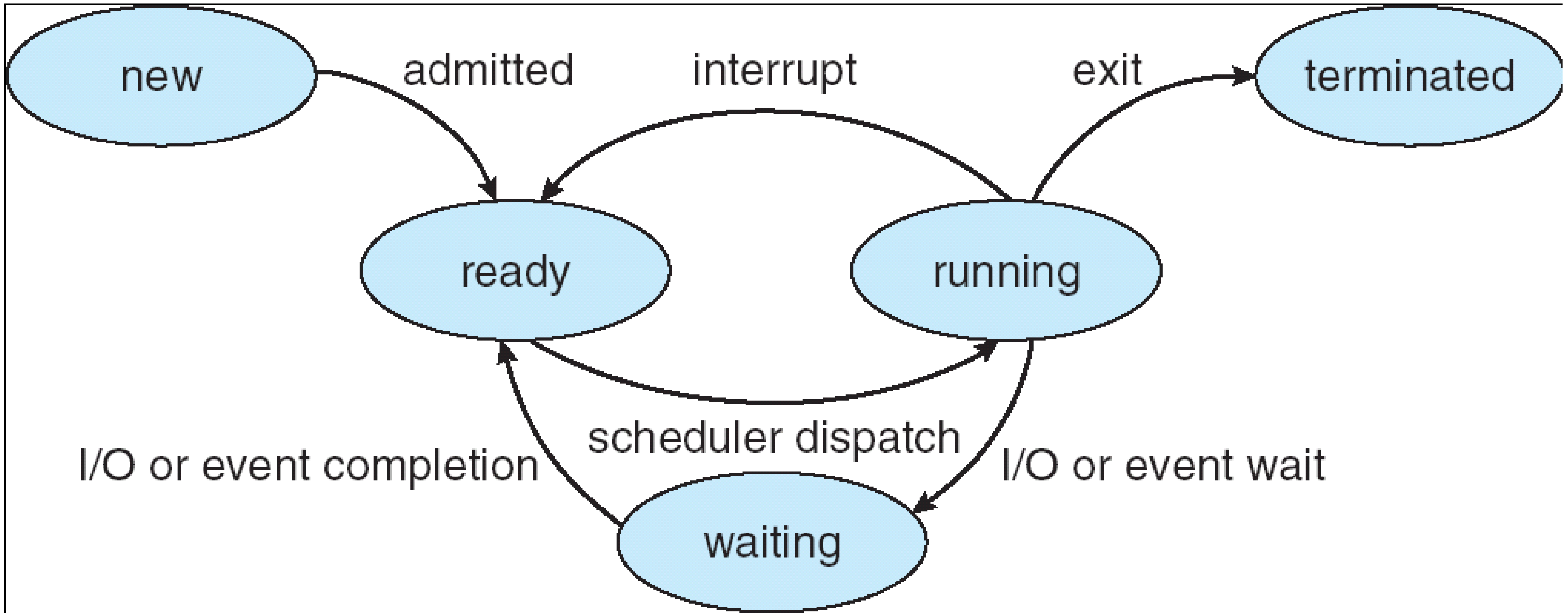
we will completely ignore threads today

Images from Silberschatz

Process

- Define:
- Memory Regions:
- Loaded from executable file:
  - ELF: Executable and Linkable Format
    - Linux
    - What does this contain?

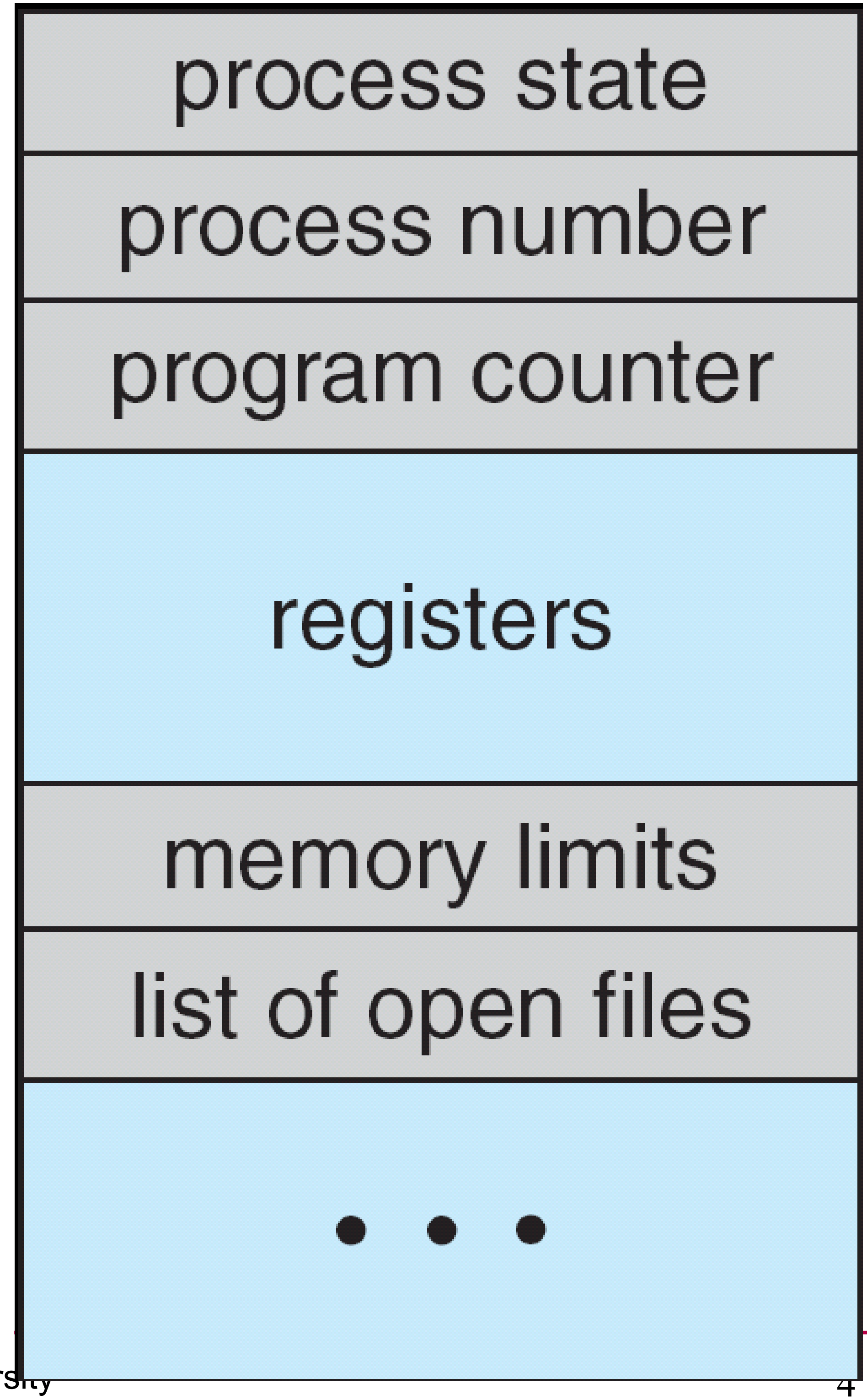
State Machine



- While a process is active it is in a particular state
- How many processes can be in each state?
- Data Structures? Where? Which kind? Why?

Process Control Block

- Who owns this data structure?
- CPU Scheduling data
- Memory Management data
- Accounting data



Types of Processes

- I/O Bound

- CPU Bound

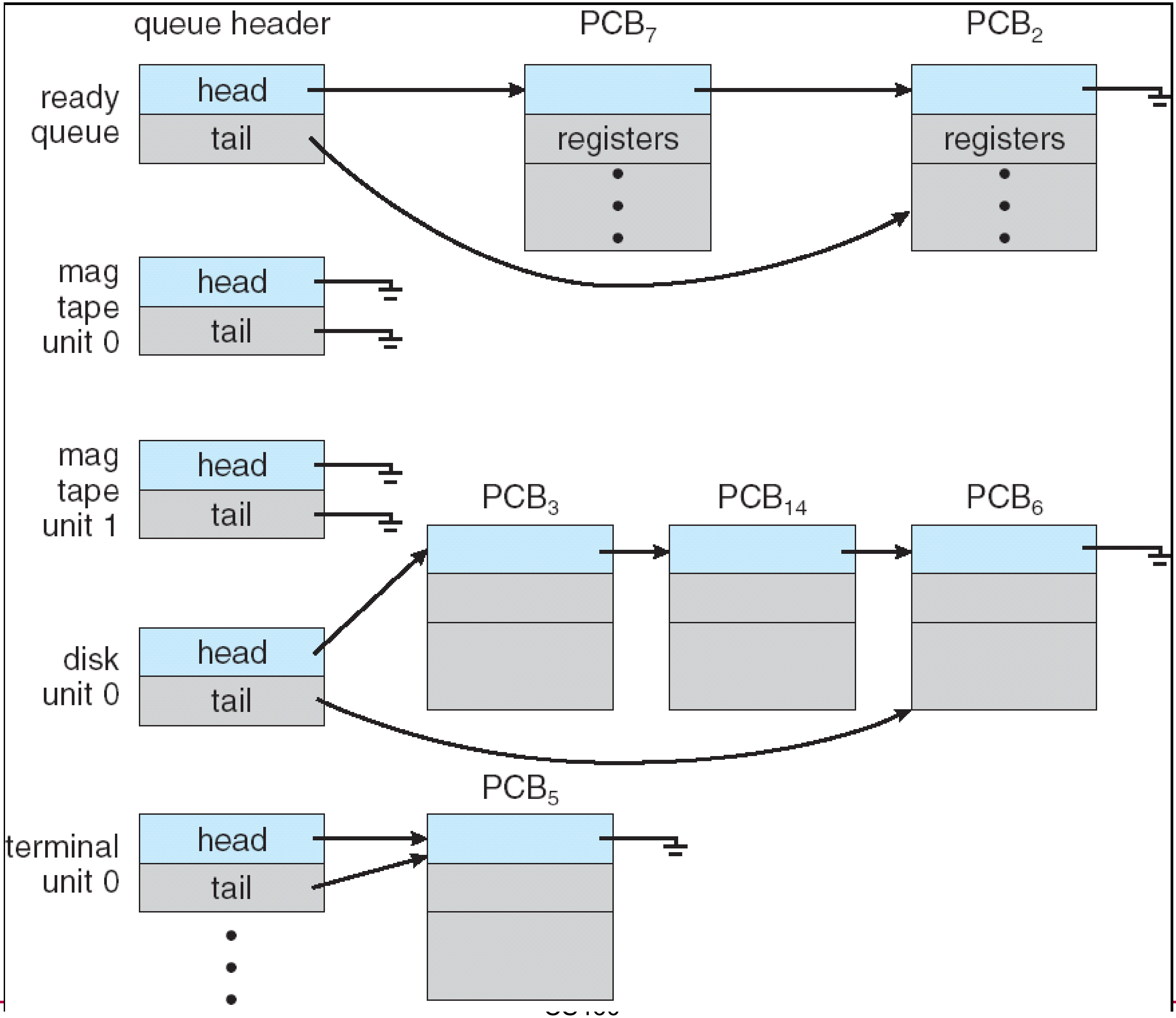
- How does this affect the OS?

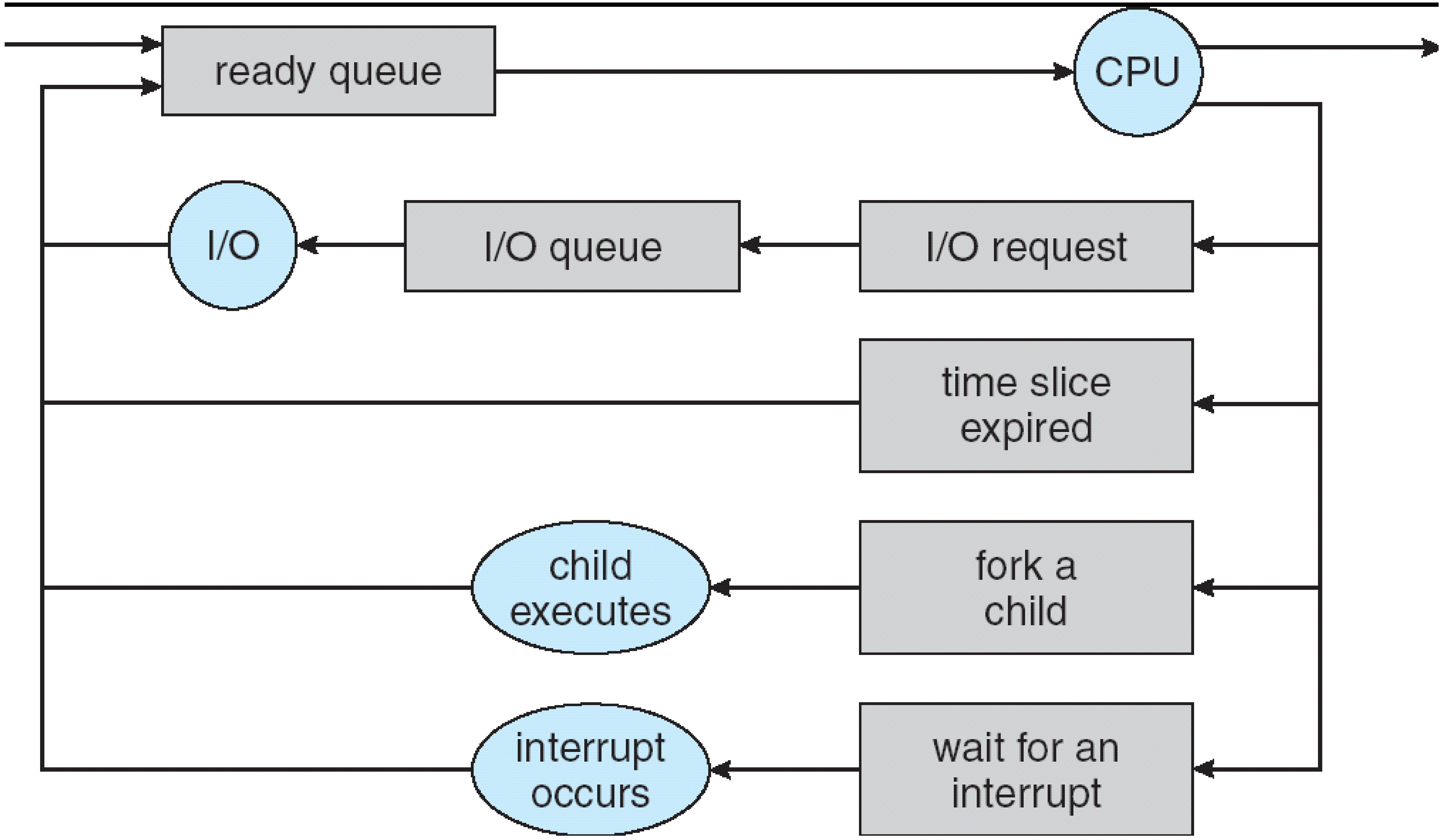
Process Scheduling

- Process Scheduler
  - Purpose:
  - Data structures:
  - Dispatched:

Schedulers

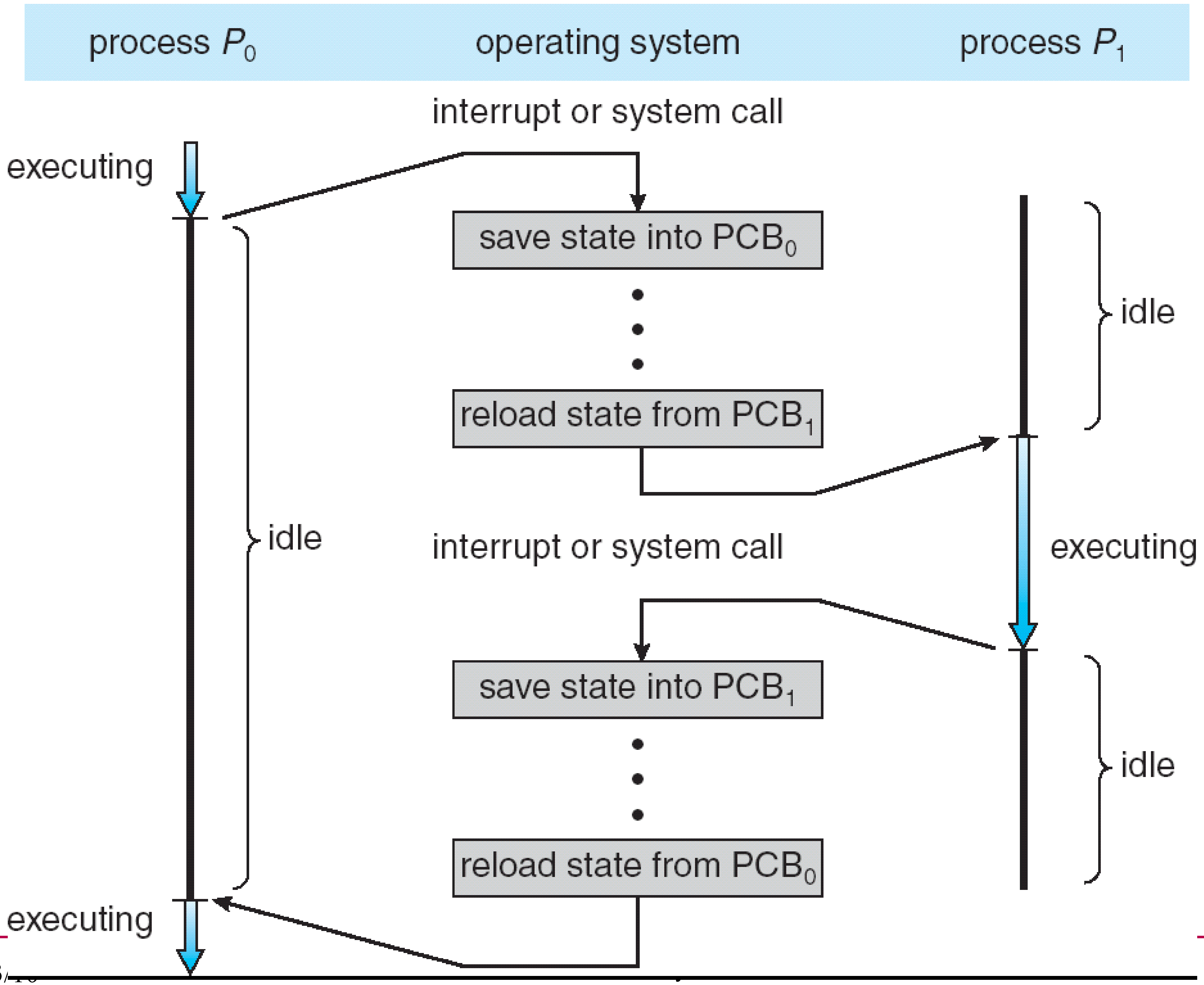
- Job Scheduler
  - Long term
  - Why is this important?
- CPU Scheduler
  - Short term
  - Constraints?
- Many OSes (Unix/Windows) don't really have a Job Scheduler


queue header
PCB7
PCB2
ready queue
head
tail
registers
registers
mag tape unit 0
head
tail
mag tape unit 1
head
tail
PCB3
PCB14
PCB6
disk unit 0
head
tail
PCB5
terminal unit 0
head
tail


ready queue
CPU
I/O
I/O queue
I/O request
time slice expired
child executes
fork a child
interrupt occurs
wait for an interrupt

Context Switch

- Context:
- What happens during a Context Switch?
- Speed?


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

Process Creation

```
/* This code works on Zeus! */
int main()
{
    pid_t  pid;
    int value = 0;
    value = 9;

    /* fork another process */
    pid = fork();
    fprintf(stderr,"The value: %d", value);

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
    }
}   /* page 92 of Silberschatz */
```

What happens if we put an fprintf() in side the block after the execlp()?

Process Termination

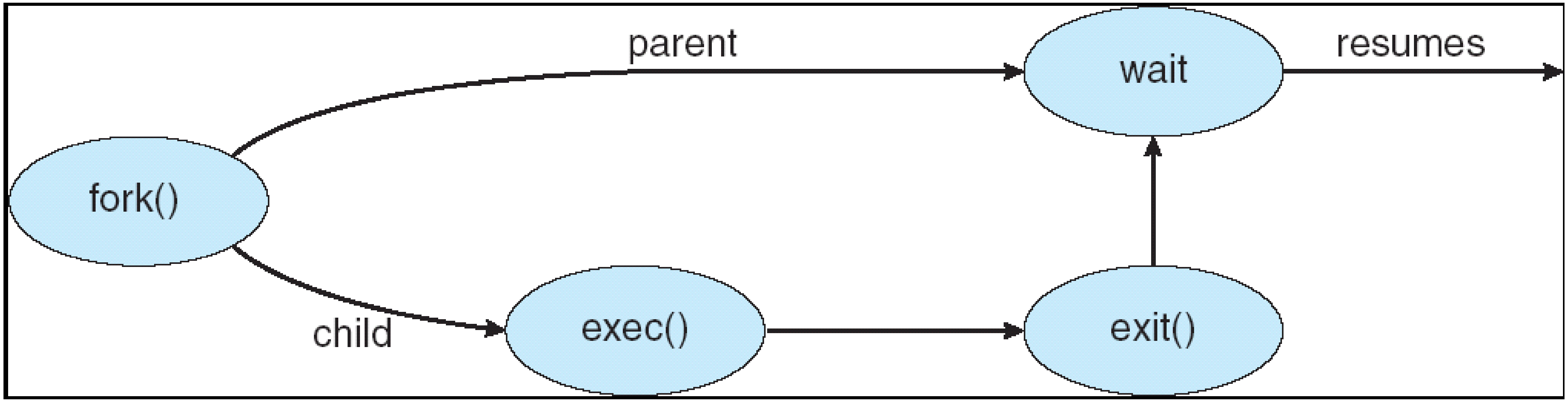
- kill(pid, signal)

```
$ man kill

$ ps u

$ kill -9 pid

$ man -s 2 kill

$ man -s 7 signal
```

- Cascading termination:

Windows (Win32 API)

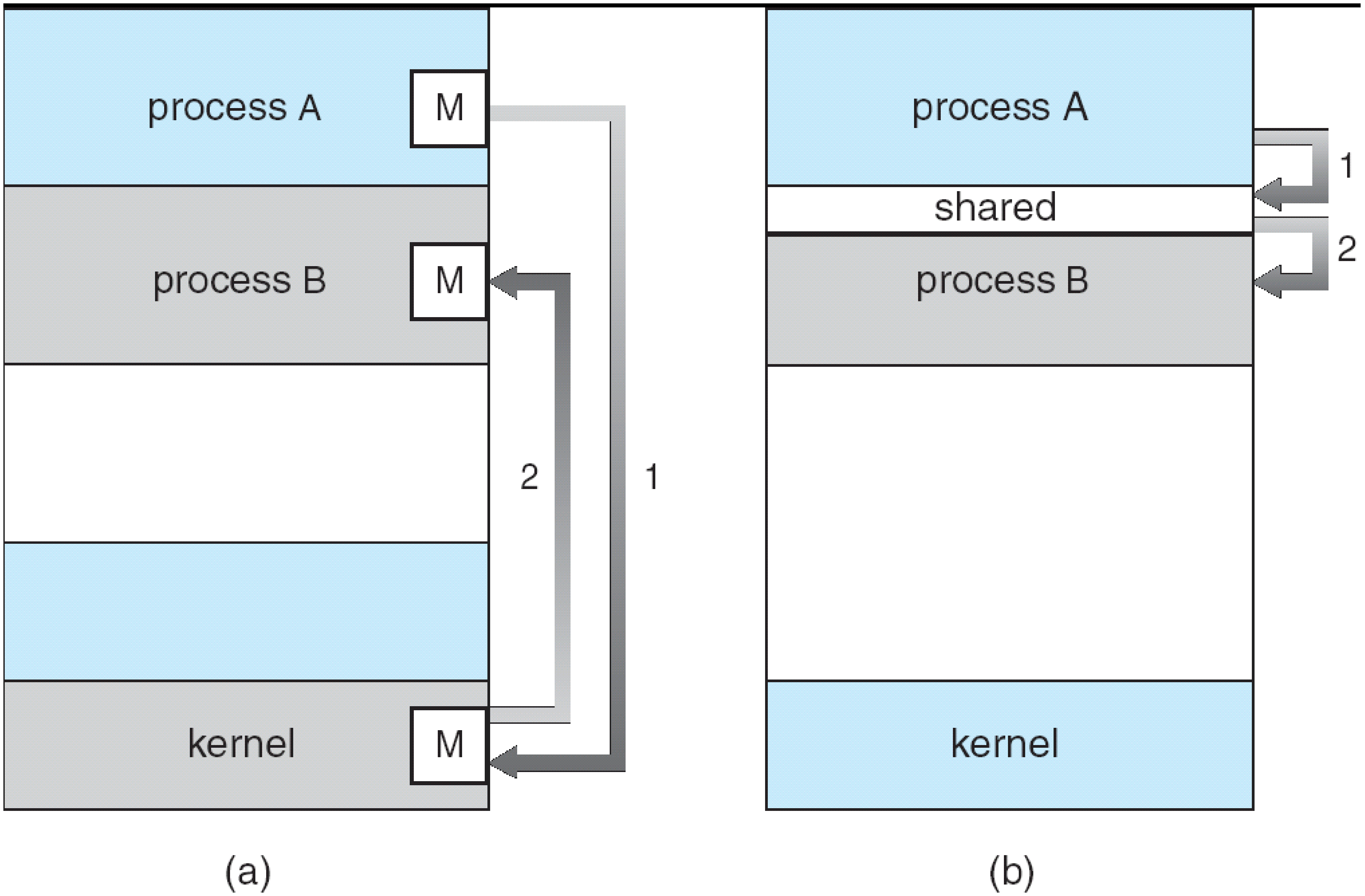
- CreateProcess()
  - fork() and exec() rolled into one
  - 10 parameters!
- WaitForSingleObject()
- TerminateProcess()


parent
wait
resumes
fork()
exec()
exit()
child

Interprocess Communication

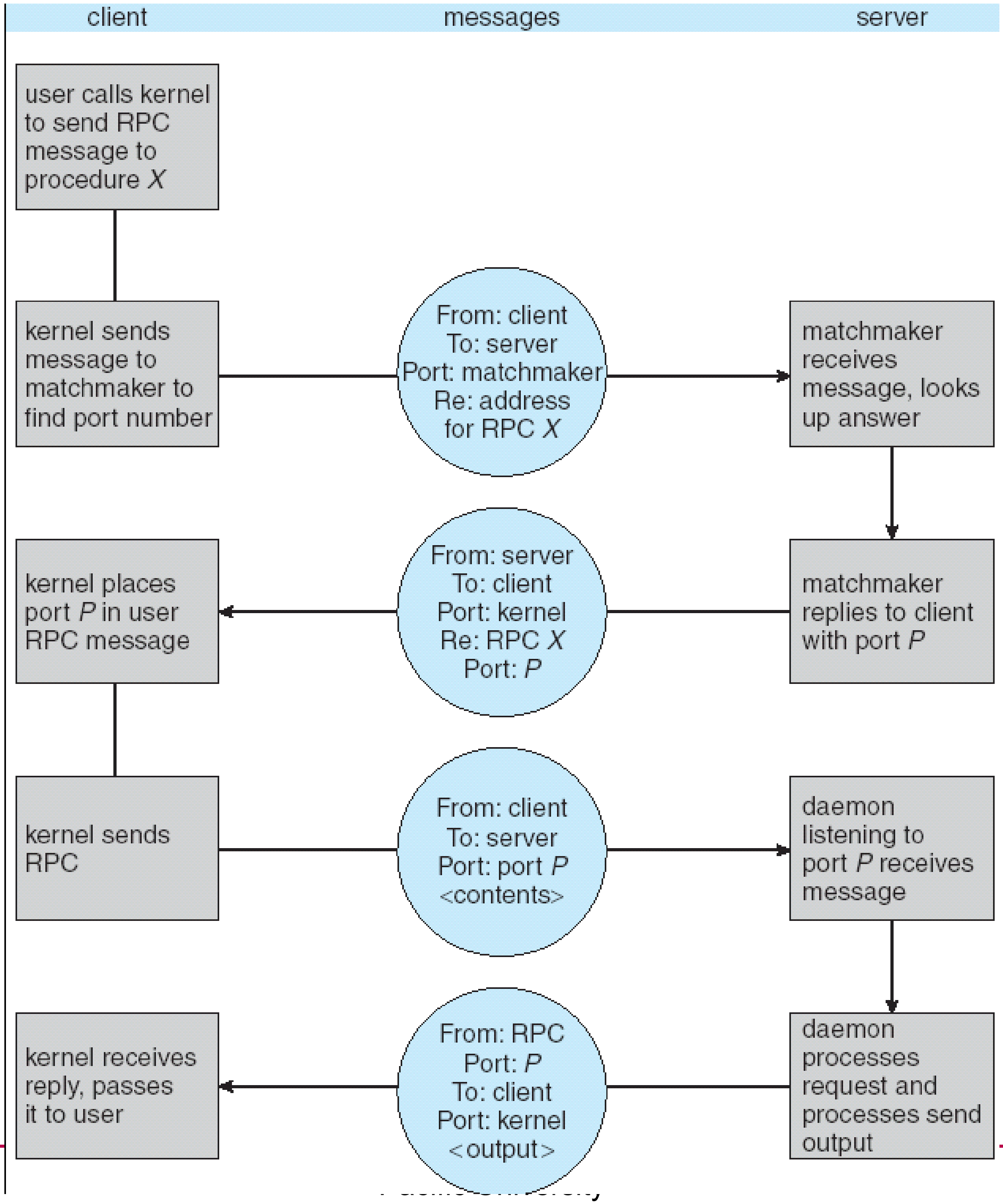
- Why do we want this?

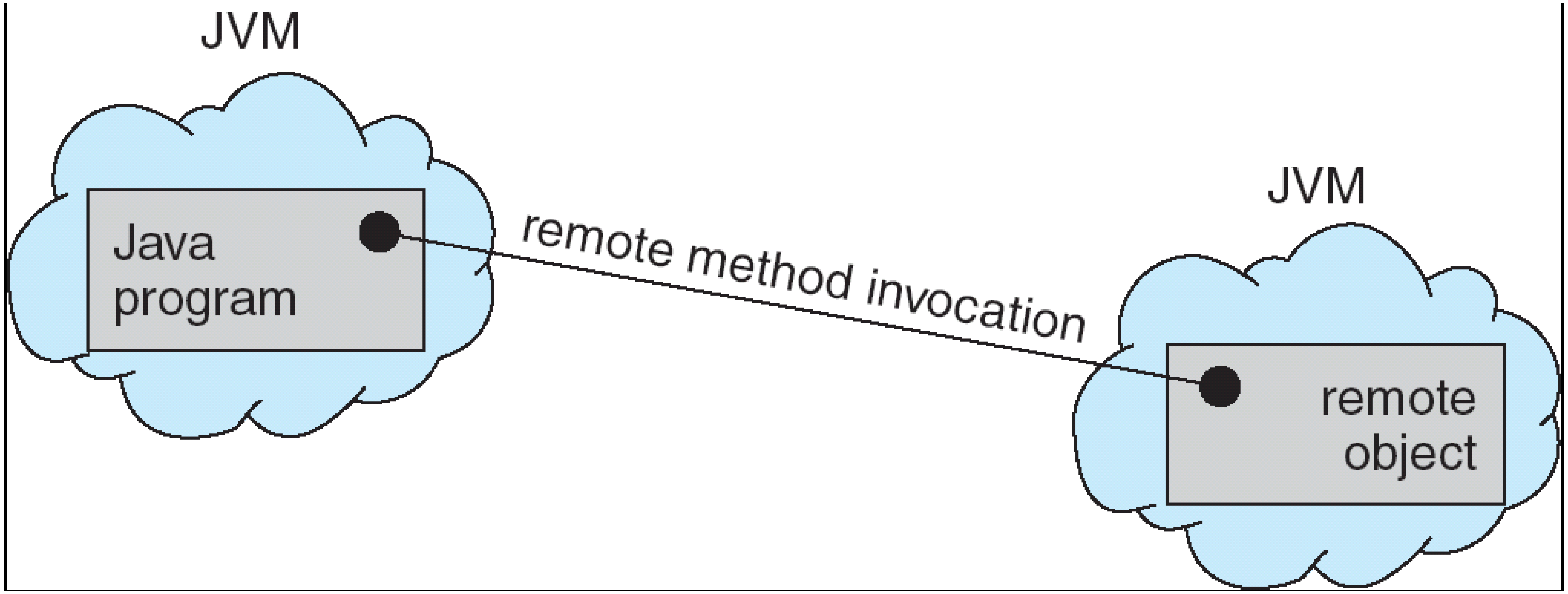
- Types:
  - Shared memory:
  - Message passing:


process A
M
process B
M
kernel
M
2
1
(a)
process A
1
shared
2
process B
kernel
(b)

Shared Memory

```
/* This code works on Zeus! */
int main()
{
   int segment_id;
   char *shared_memory;
   const int size = 4096;

   /* allocate shared memory segment */
   segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);

   /* attach the shared memory segment */
   shared_memory = (char*) shmat (segment_id, NULL, 0);

   /* write a message to the shared memory segment */
   sprintf(shared_memory, "Hi there!");

   /* now print out the string from shared memory */
   printf("*%s\n", shared_memory);

   /* now detach the shared memory segment */
   shmdt(shared_memory);

   /* now remove the shared memory segment */
   shmctl(segment_id, IPC_RMID, NULL);
}
/* page 104 of Silberschatz */
```


client
messages
server
user calls kernel to send RPC message to procedure X
kernel sends message to matchmaker to find port number
From: client To: server Port: matchmaker Re: address for RPC X
matchmaker receives message, looks up answer
matchmaker replies to client with port P
From: server To: client Port: kernel Re: RPC X Port: P
kernel places port P in user RPC message
kernel sends RPC
From: client To: server Port: port P <contents>
daemon listening to port P receives message
daemon processes request and processes send output
From: RPC Port: P To: client Port: kernel <output>
kernel receives reply, passes it to user


JVM
Java program
remote method invocation
JVM
remote object

Functions

```
int execl(const char *path, const char *arg, ...)

int execlp(const char *file, const char *arg, ...)

int execle(const char *file, const char *arg, ...,
   char const* envp[])

int execv(const char *path, char *const argv[])

int execvp(const char *file, char *const argv[])

int dup2(int oldfd, int newfd)

int pipe(int filedes[2])

pid_t waitpid(pid_t pid, int *status, int options)

char* strtok_r(char *str, const char* delim, char **saveptr)
```