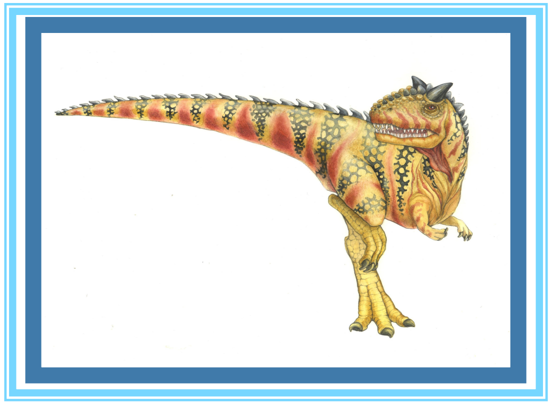
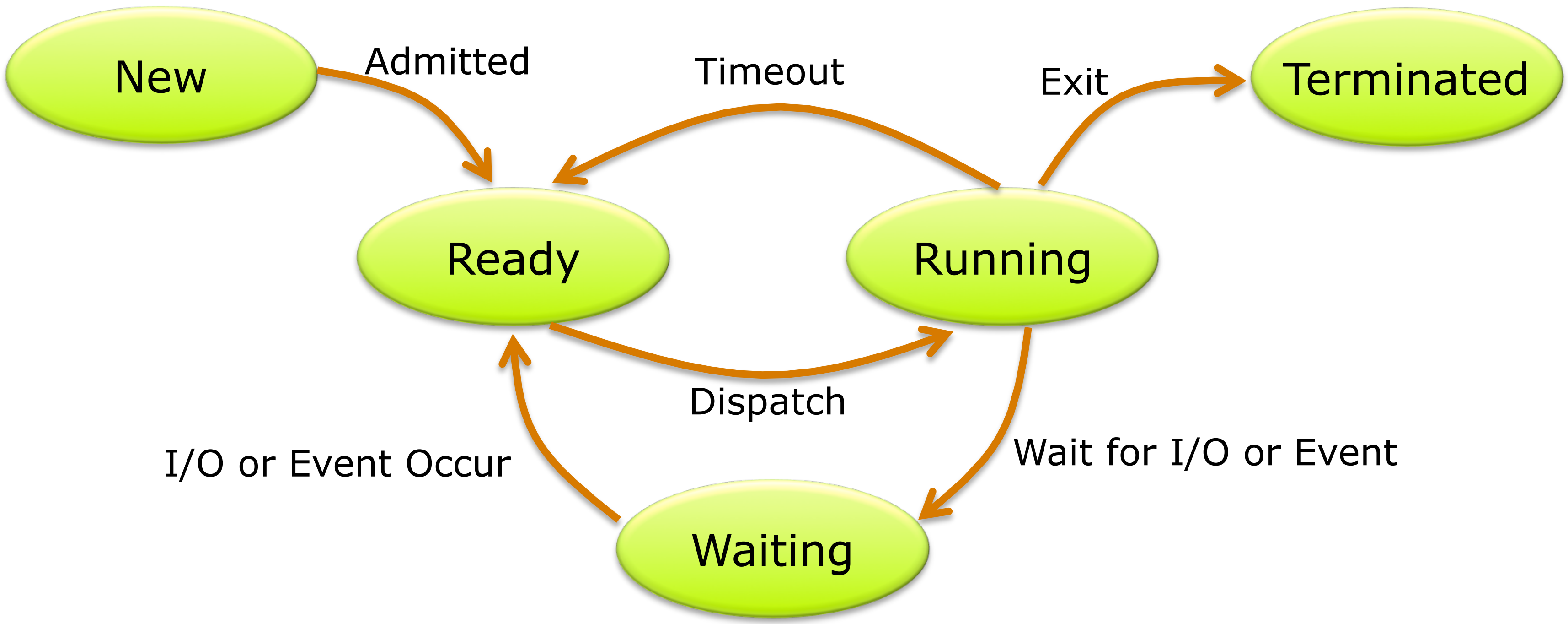
Chapter 3: Processes



Five-state Process Model



UNIX Process State Transition Diagram

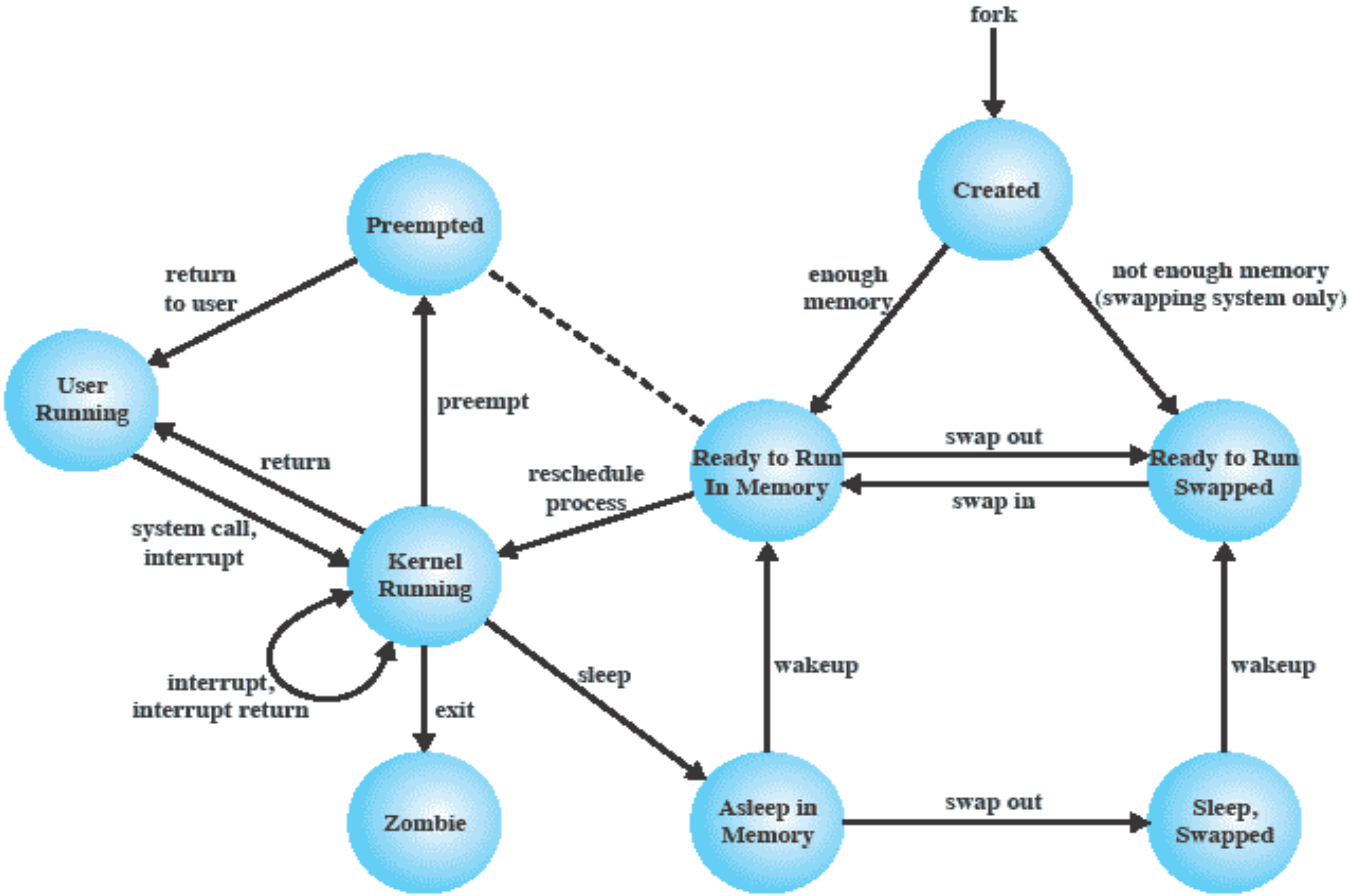
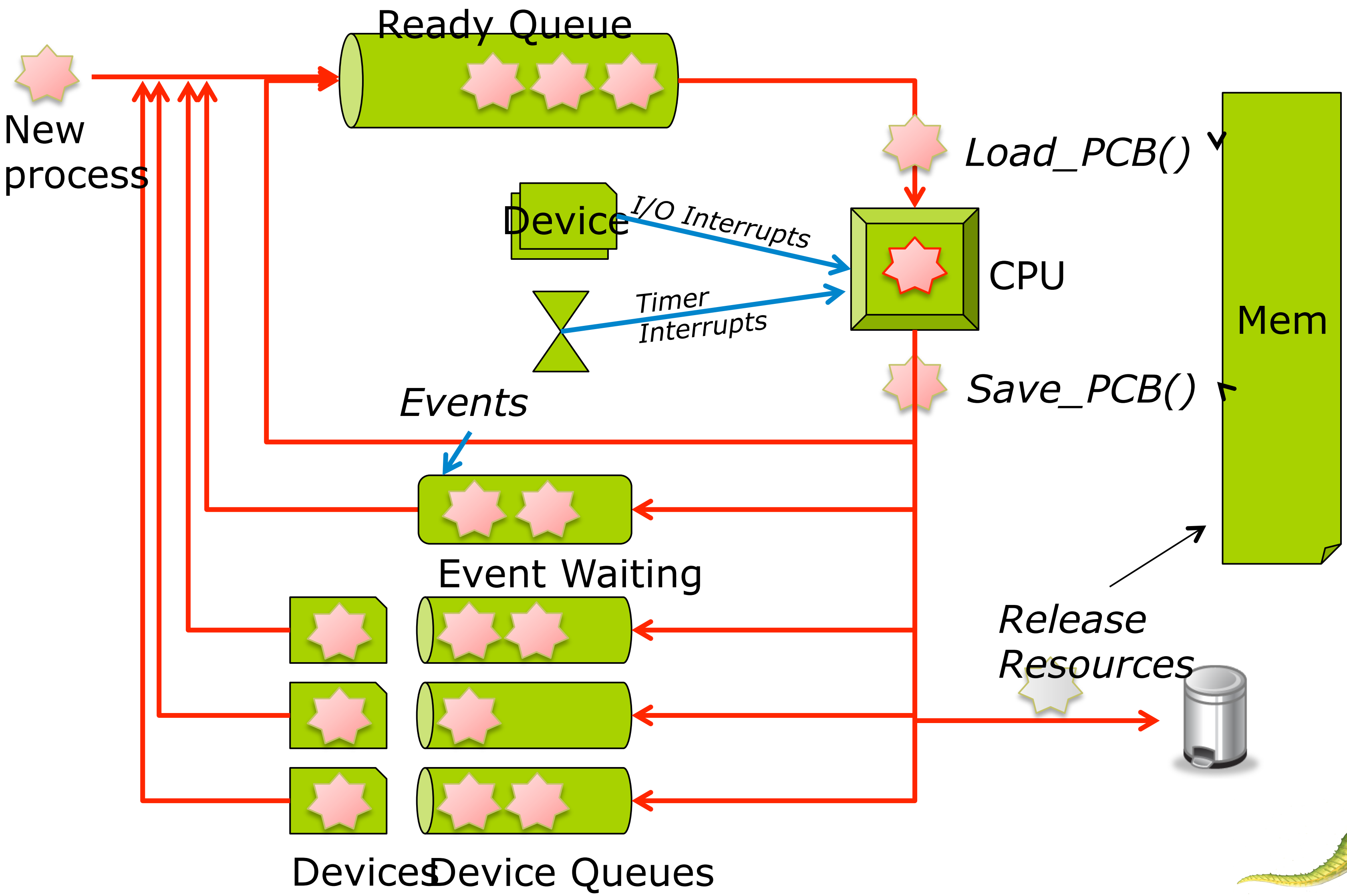


Figure 3.17 UNIX Process State Transition Diagram

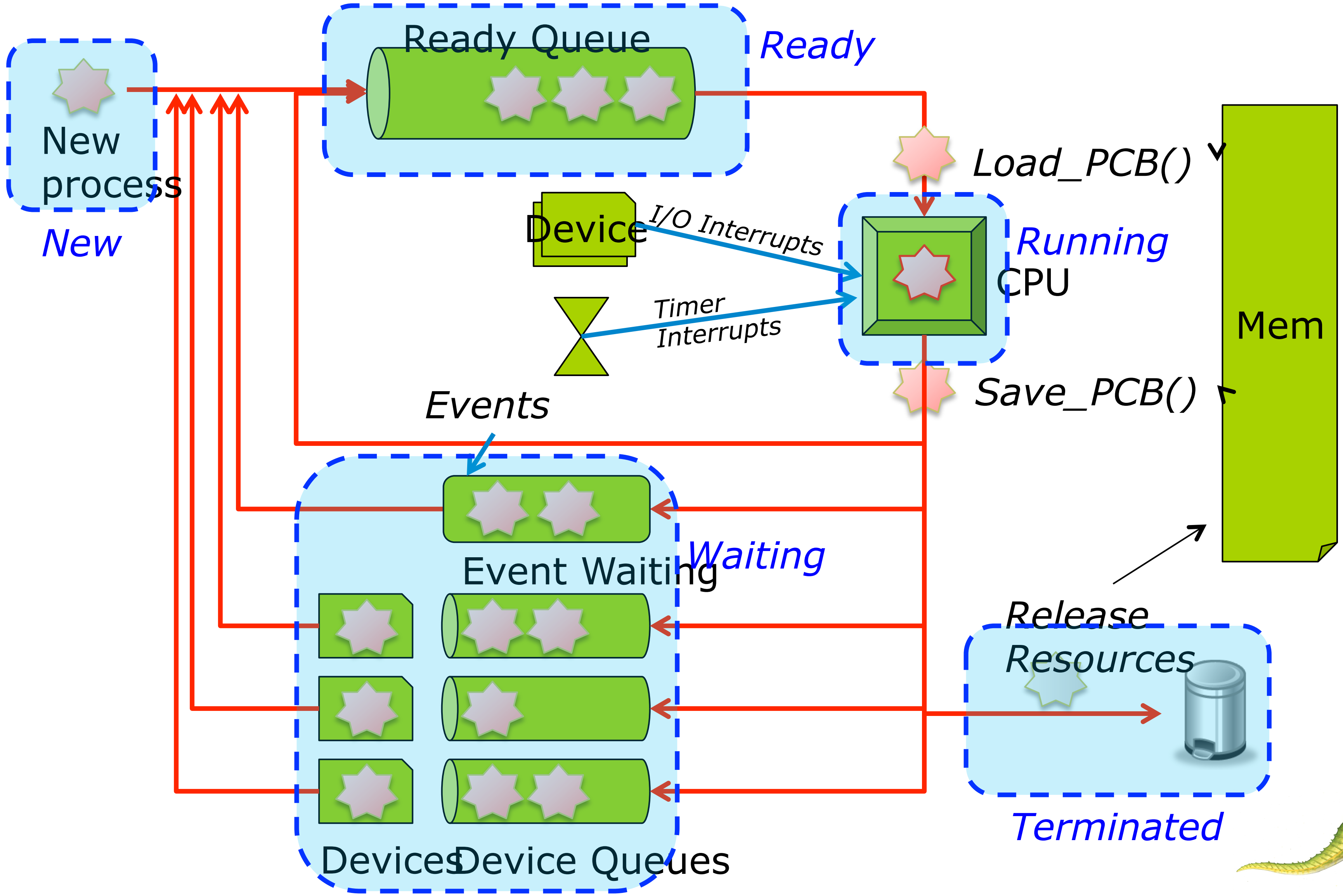
Process Scheduling

- 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

Lifecycle of Processes

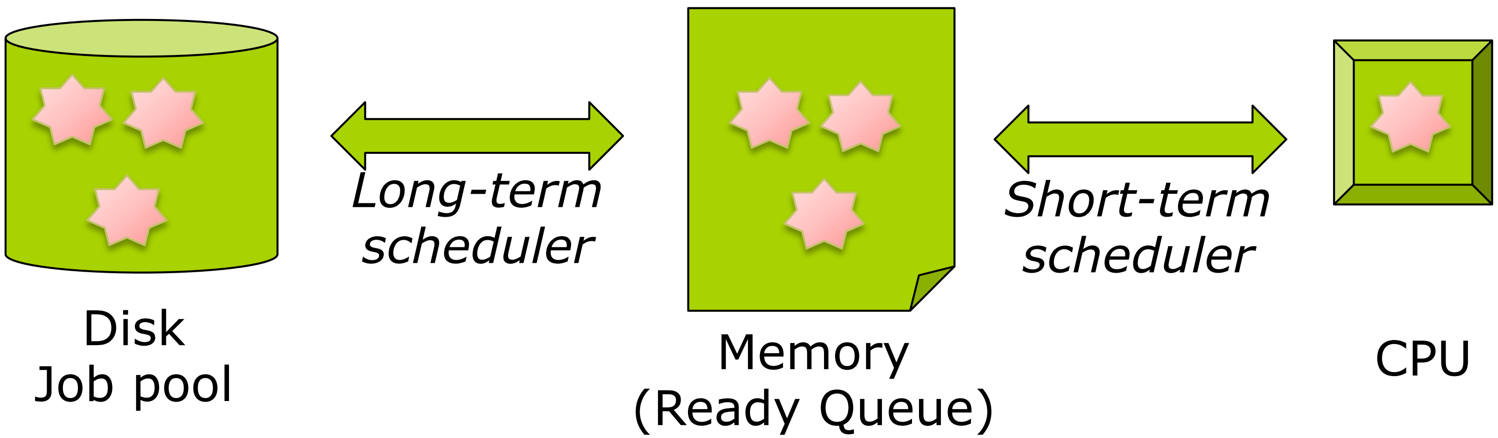


States of Processes



Schedulers

- **Scheduler:** determines the change of process state
- **long-term scheduler** (or job scheduler)
- **Short-term scheduler** (or CPU scheduler)
  - Sometimes the only scheduler in a system

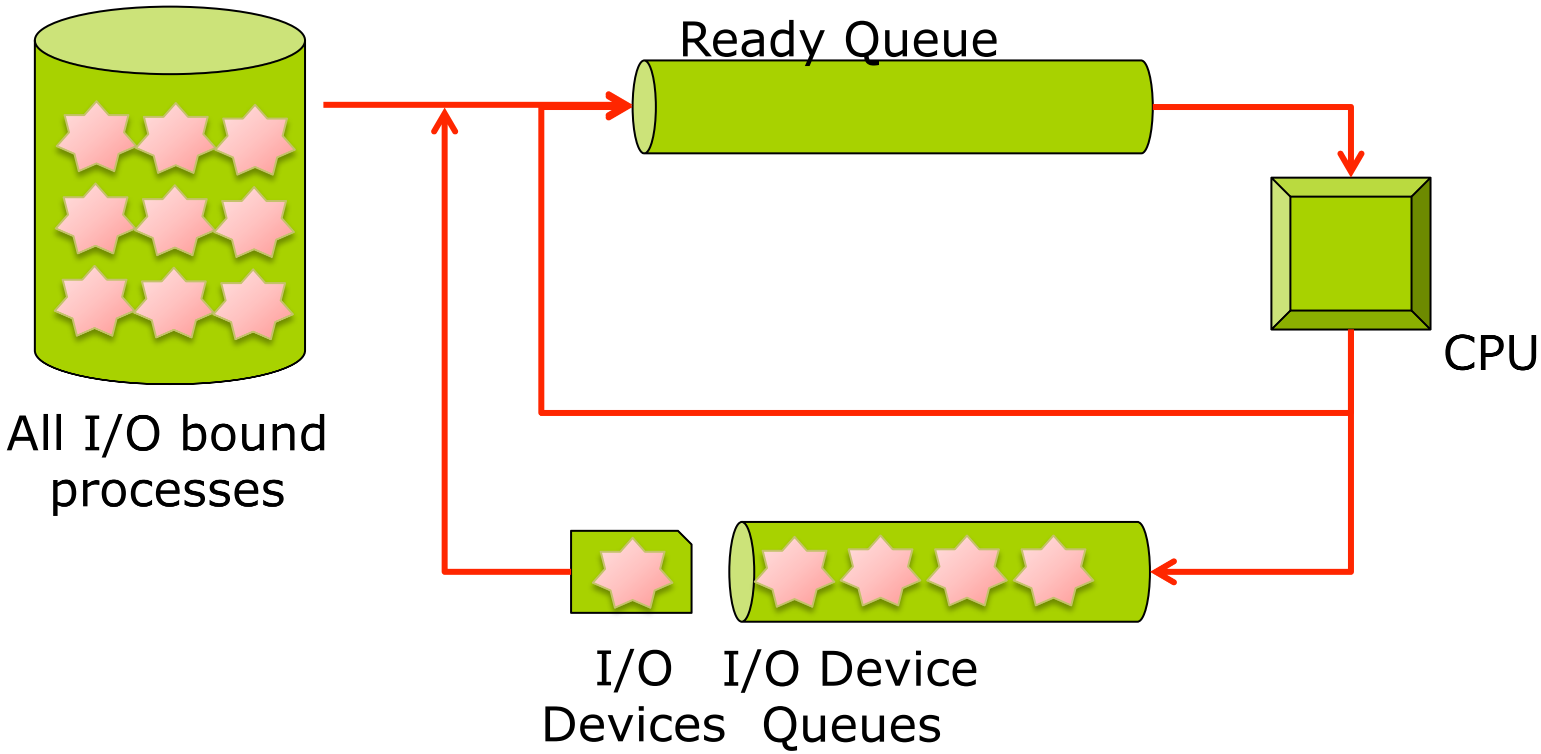


Schedulers (Cont.)

- Short-term scheduler is invoked very frequently
  - When a process leaves CPU
  - in milliseconds
  - must be fast
- Long-term scheduler is invoked very infrequently
  - When a process leaves memory
  - in seconds/ minutes
  - may be slow
- Types of processes
  - **I/O-bound process** – spends more time doing I/O
  - **CPU-bound process** – spends more time doing computations

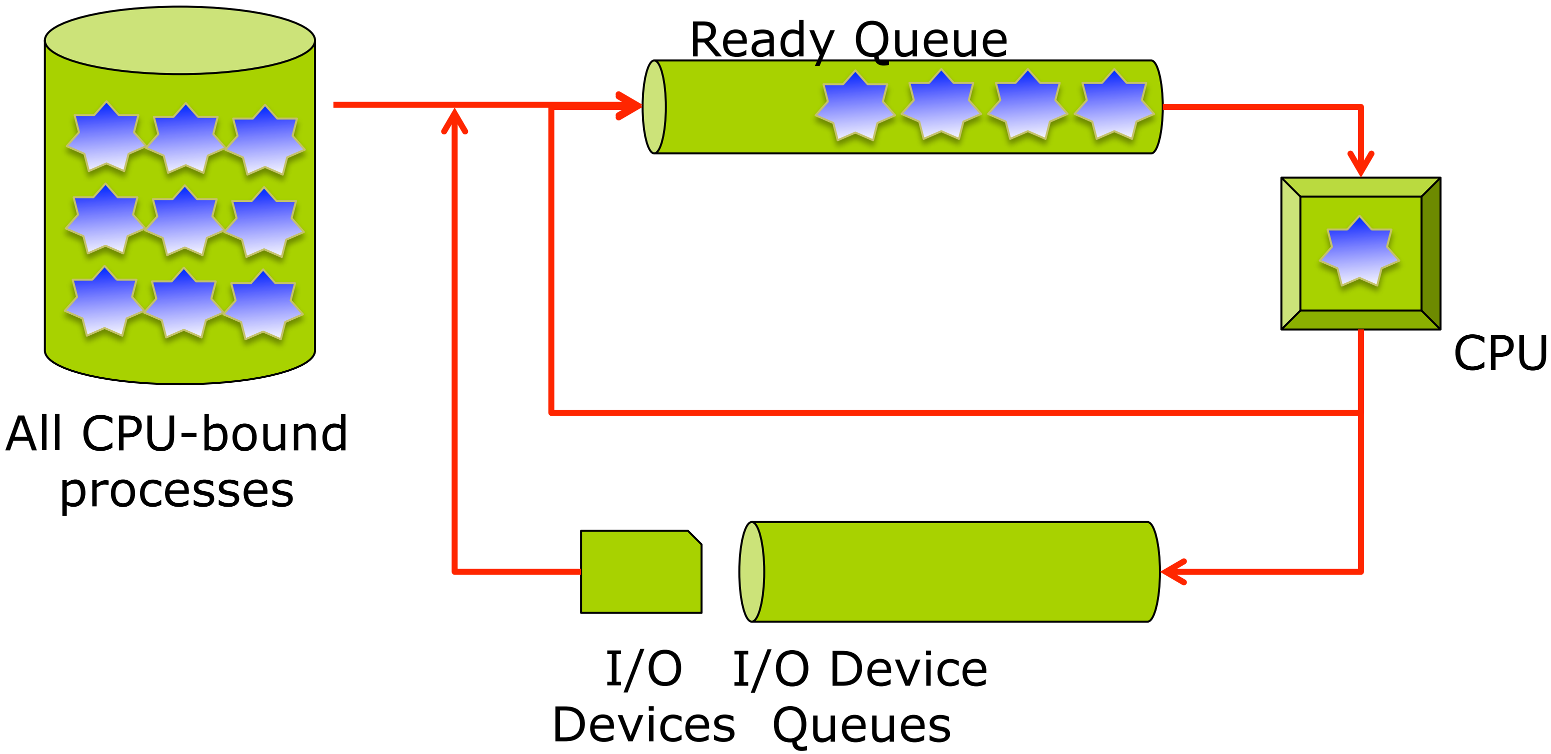
Challenge of LT-scheduler

- I/O-bound processes: fills up device queues



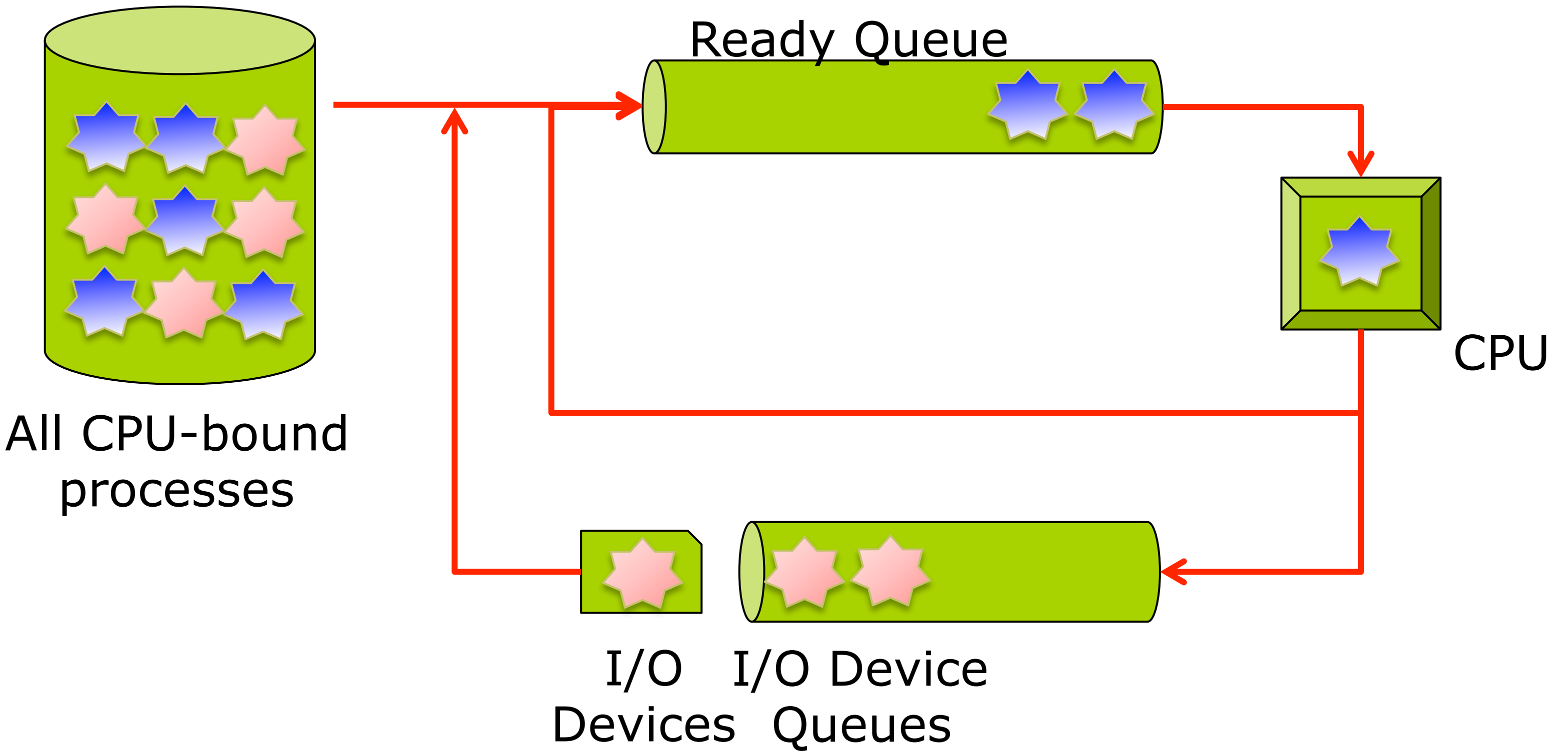
Challenge of LT-scheduler

- CPU-bound processes: fills up ready queue



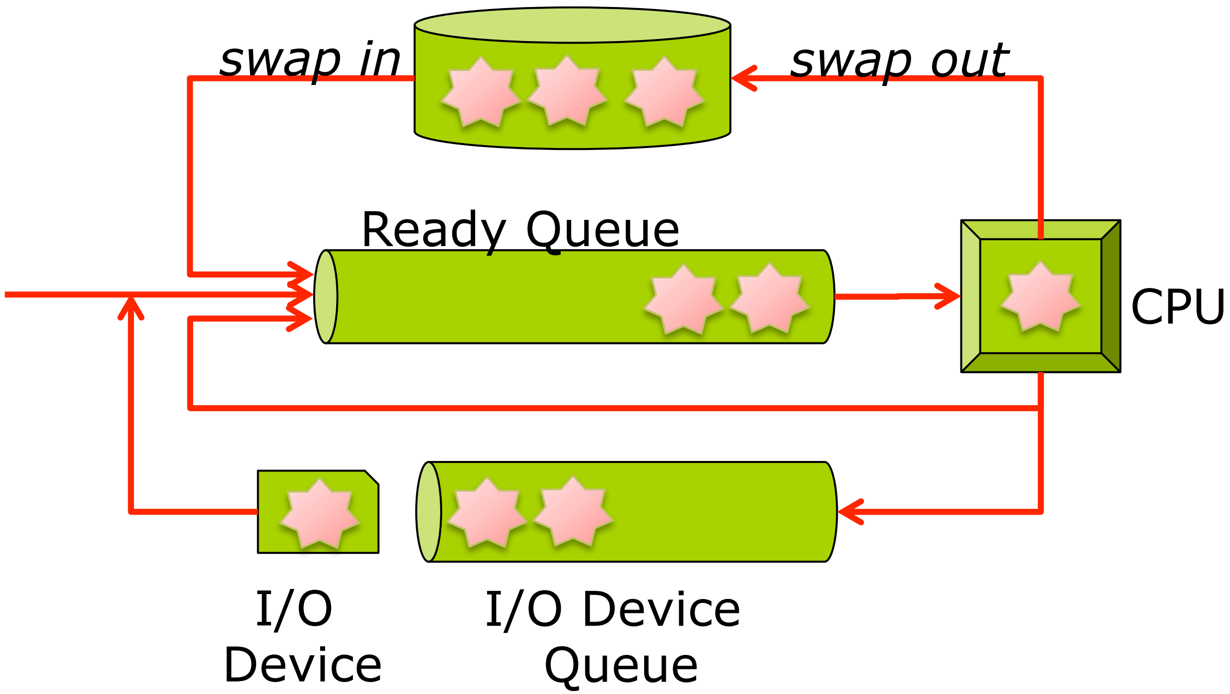
Challenge of LT-scheduler

- LT-scheduler: mix I/O-bound and CPU-bound processes
  - Good system utility



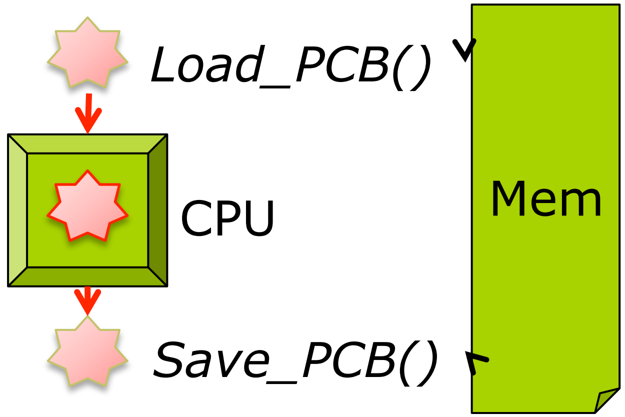
Medium Scheduling: Swapping

- Move some process from memory into disk temporarily
  - Swap out
- Later, reloads the process from disk to memory
  - Swap in



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

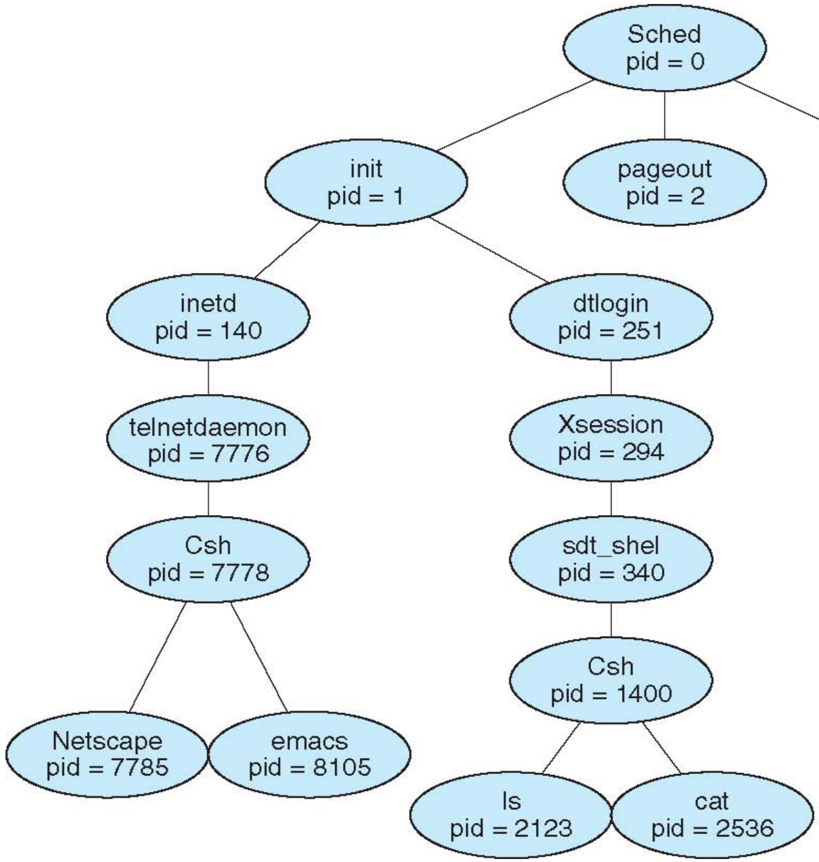


Context Switch

- Context-switch time is pure overhead
  - memory speed
  - number of registers
  - special instruction for context switch
  - a few milliseconds
  - Some hardware provides multiple sets of registers per CPU → no register copy needed

Process Creation

- **Parent** process creates **children** processes, which, in turn create other processes, forming a tree of processes



Process Creation

- Generally, process identified and managed via **a process identifier** (**pid**)
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - 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 in Unix

```
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;
}
```

Process Creation in Unix

```
int main() {
    …
    pid = fork();
```

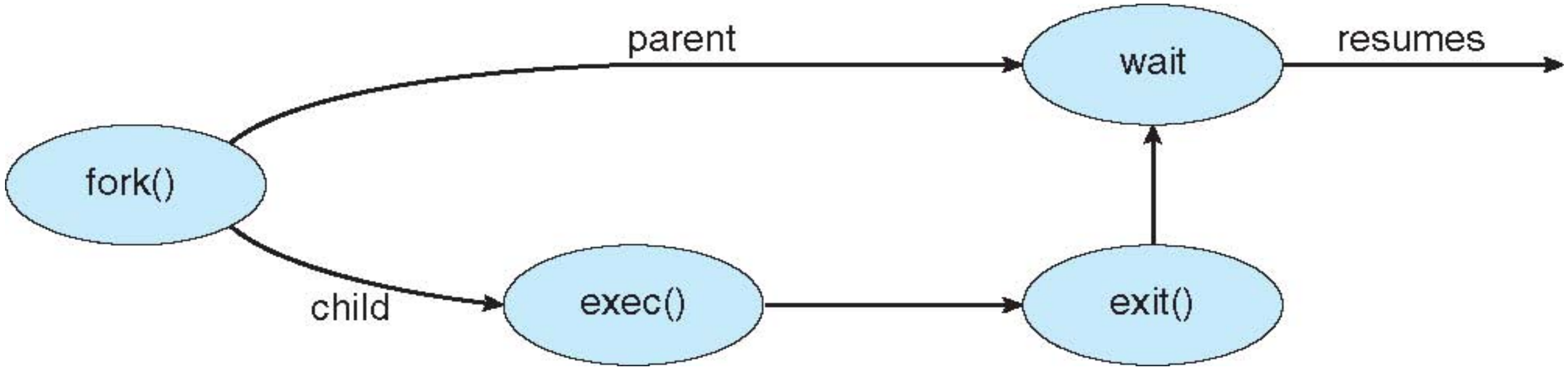
parent process

new child process
process id = 1234

```
// pid is 1234
if (pid < 0) {
        fprintf(stderr, "Fork 
Failed");
        return 1;
}
else if (pid == 0) {
        execlp("/bin/ls", "ls", 
NULL);
}
else {
        wait (NULL);
        printf ("Child 
Complete");
}
return 0;
}
```

```
// pid is 0
if (pid < 0) {
        fprintf(stderr, "Fork 
Failed");
        return 1;
}
else if (pid == 0) {
        execlp("/bin/ls", "ls", 
NULL);
}
else {
        wait (NULL);
        printf ("Child 
Complete");
}
return 0;
}
```

Process Creation



Process Creation in Win32

```
int main(VOID) {
        //...
        // create child process
        if (!CreateProcess(NULL, // use command line
                "C:\\WINDOWS\\system32\\mspaint.exe”
                NULL, //inherit process handle
                NULL, //don't inherit thread handle
                FALSE, //disable handle inheritance
                0, // no creation flags
                NULL, //use parent's environment block
                NULL, //use parent's existing directory
                &si, &pi)) {
                        fprintf(stderr, "Create Process Failed");
                        return -1;
          }
        WaitForSingleObject(pi.hProcess, INFINITE);
        printf("Child Complete");
}
```

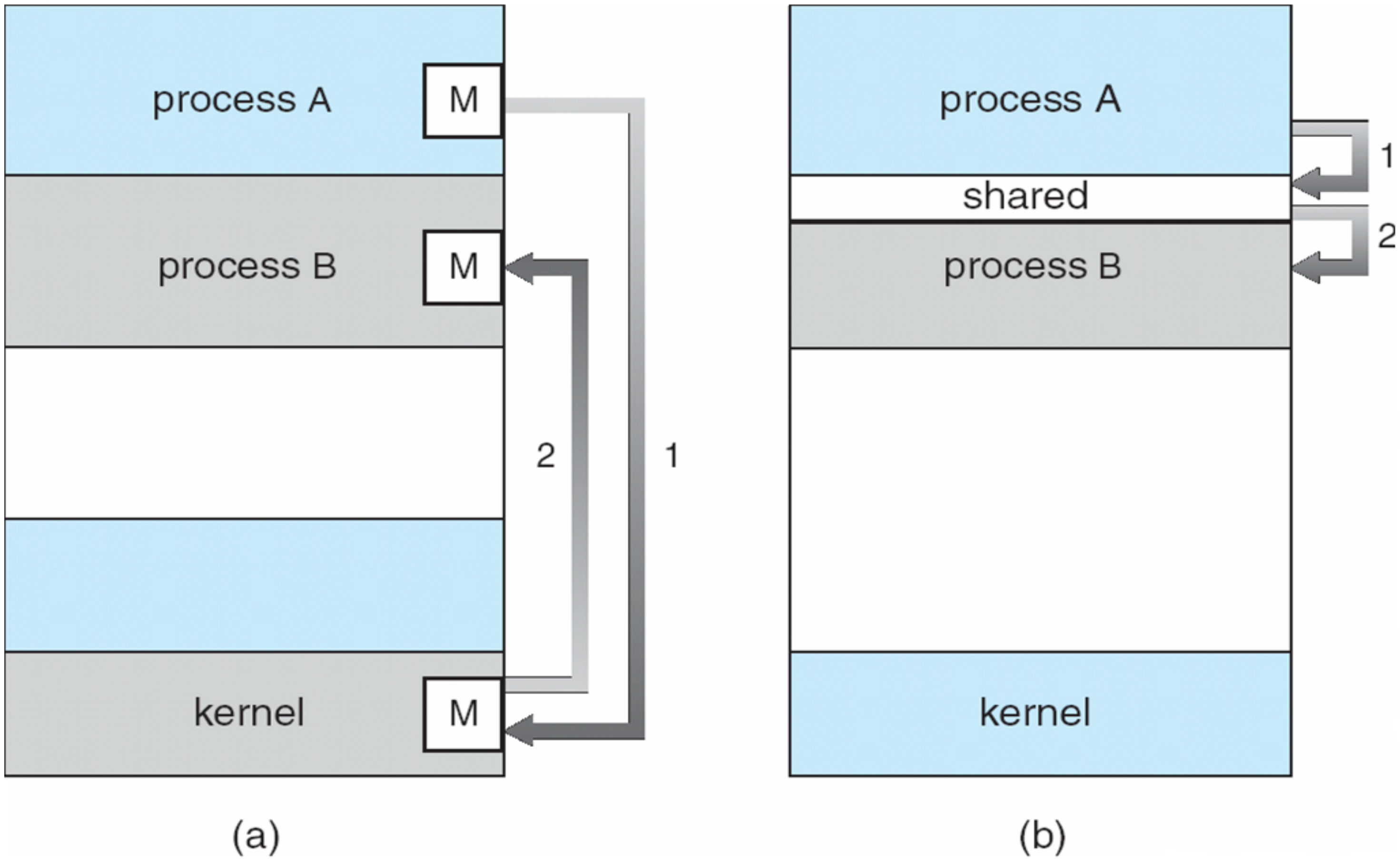
Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data 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**

Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

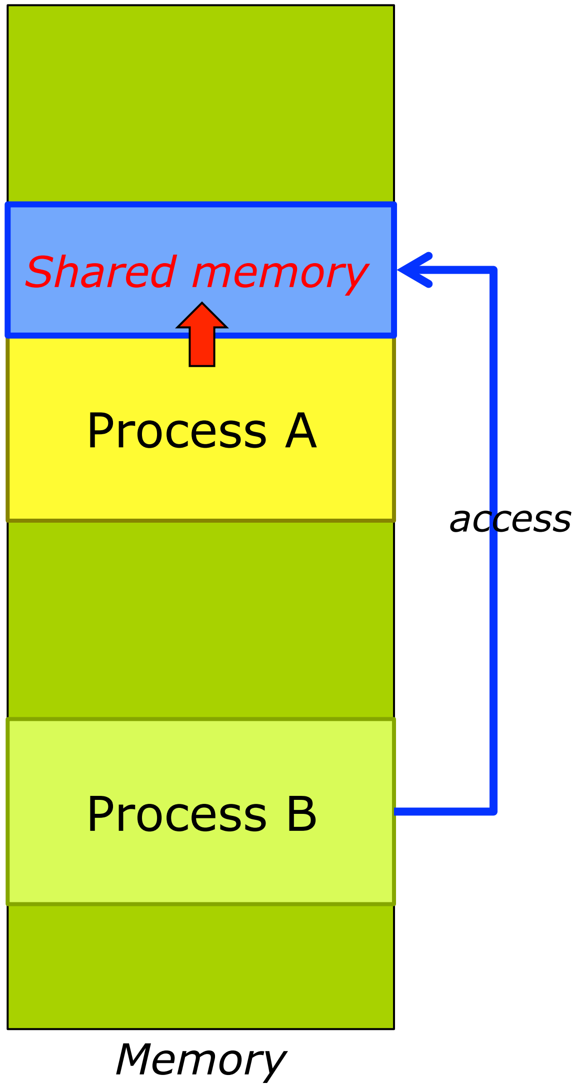
Communications Models



Shared Memory & Message Passing

	Message Passing	Shared Memory
Implementation	Easier	Difficult
Speed	Slower	Faster
Kernel intervention	A lot, via system calls	No system calls except setup
Data size	Good for small amount	Good for large amount

Shared Memory Systems



- Process-A creates a shared memory
  - Shared memory in Process-A's address space
- Allow Process B to access the shared memory
- No predefined data format