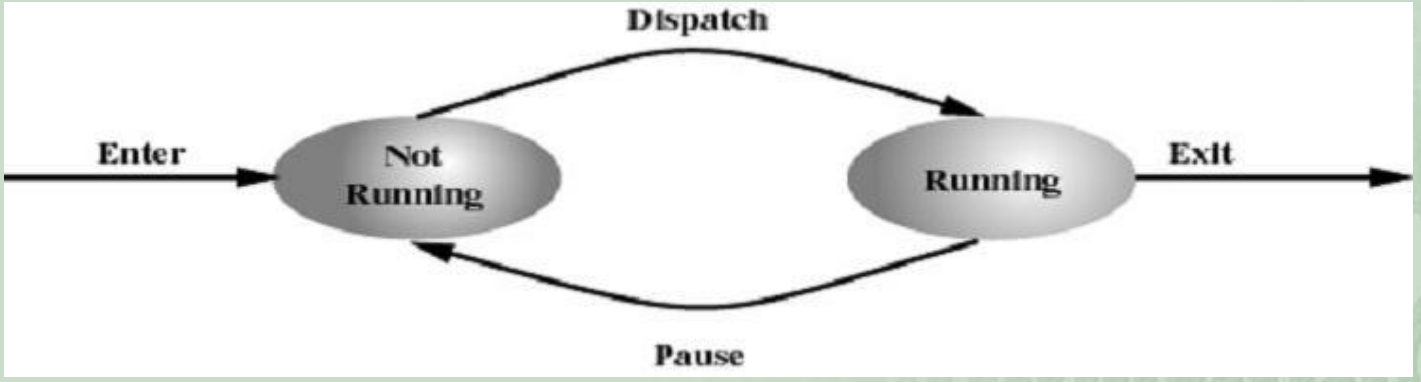
Process

Process Concepts

- Process Definition
- Process Creation
- Process States and State Transitions
- Important Information Associated with Processes

Two-State Process Model

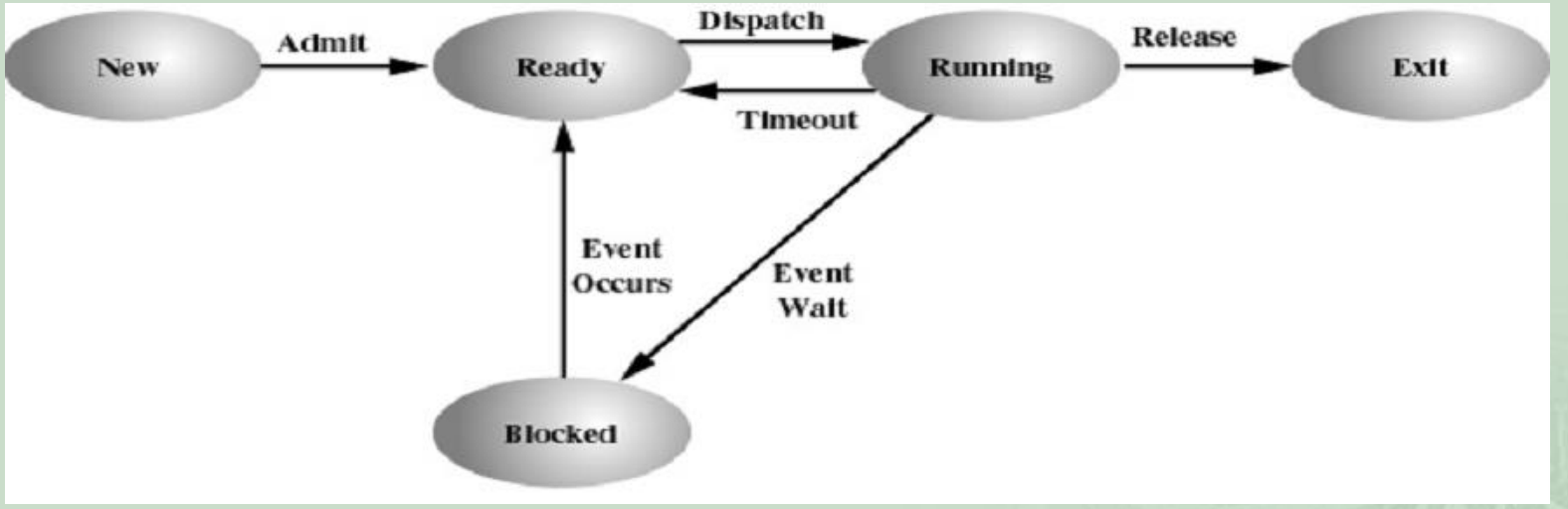
- Process may be in one of two states
  - Running
  - Not-running

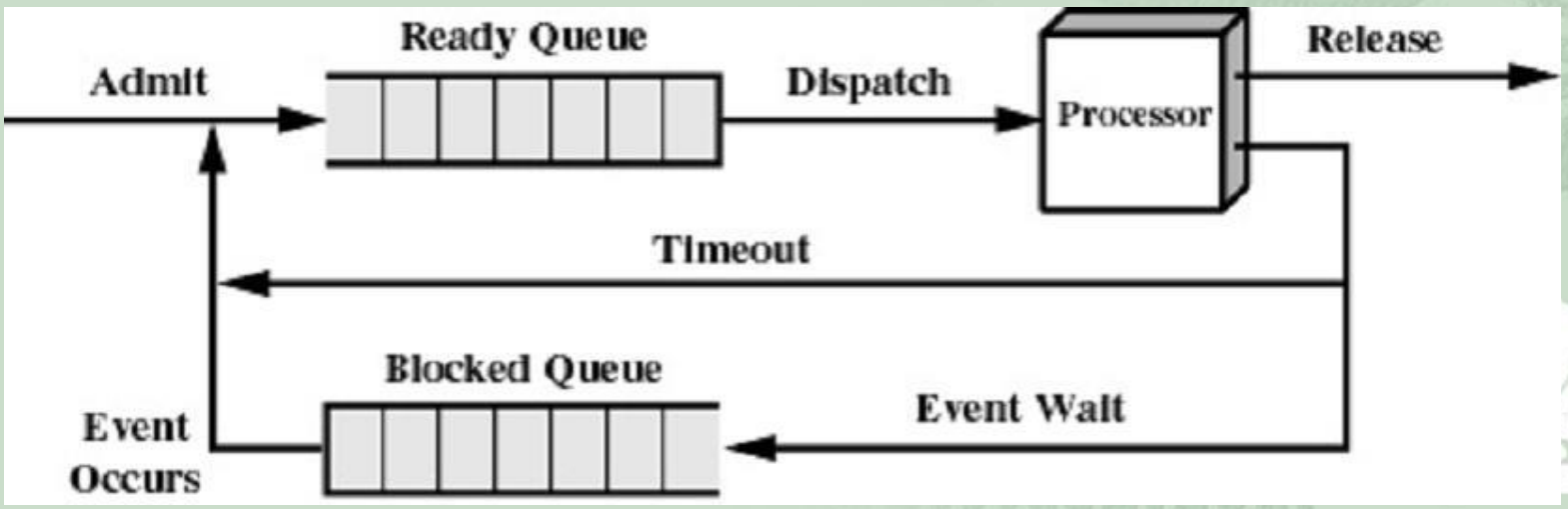


Two-state Model Problems

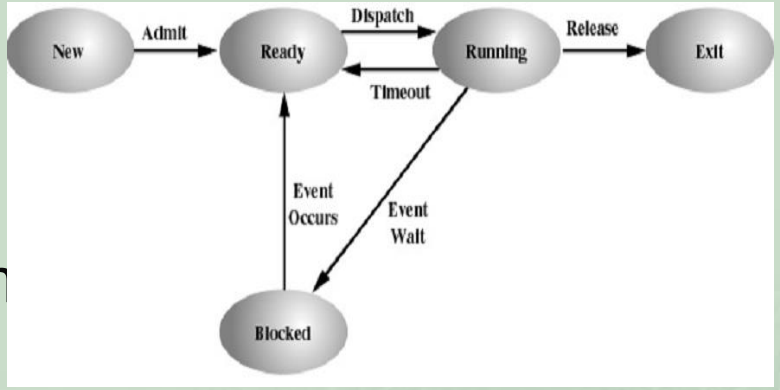
- Not-running
  - ready to execute
- Blocked
  - waiting for I/O, semaphore
- Dispatcher cannot just select the process that has been in the queue the longest because it may be blocked

A Five-State Model



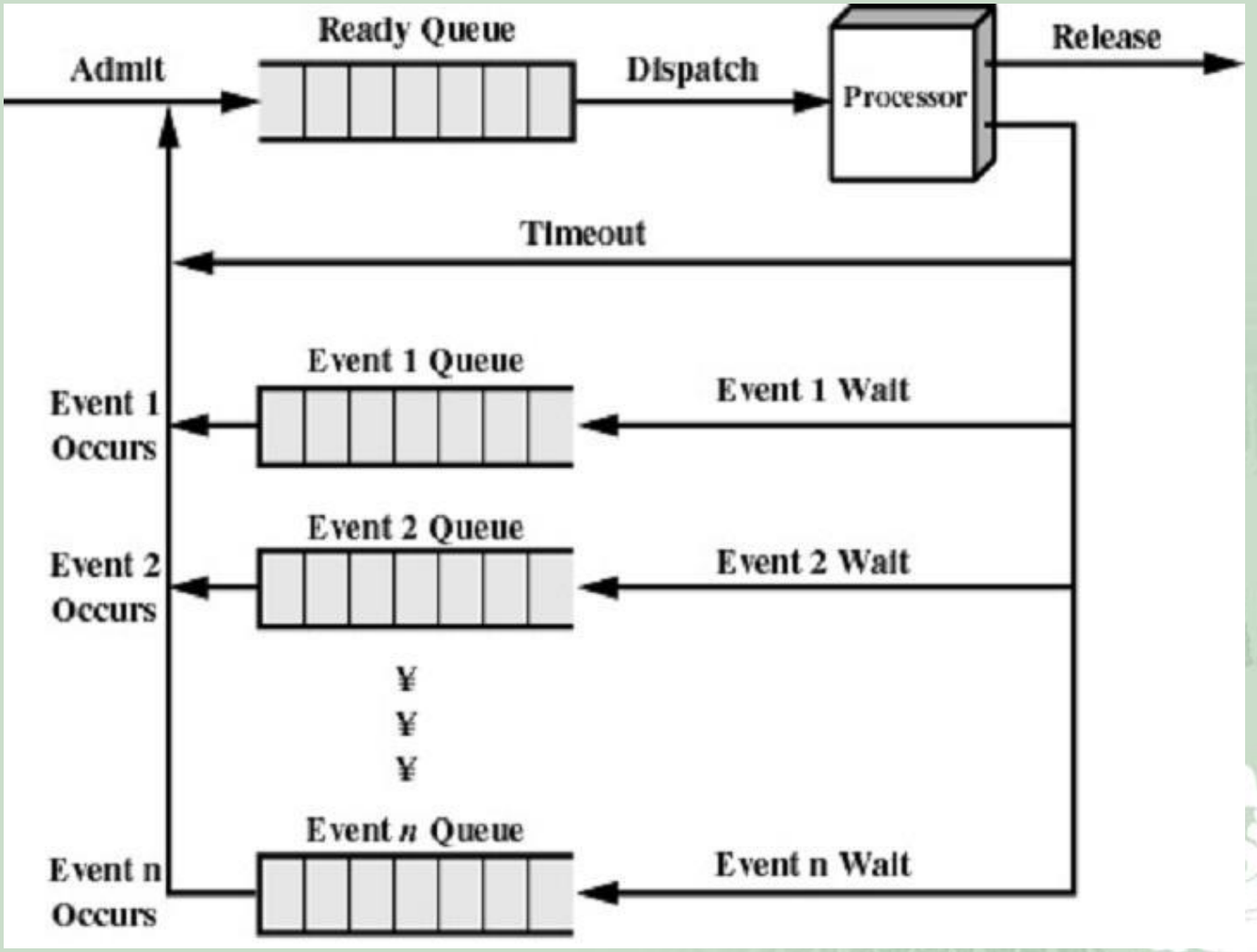


Five-state Model Transitions



- **Null → New:** Process is created
- **New → Ready:** O.S. is ready to handle another process
- **Ready → Running:** Select anoth process to run
- **Running → Exit:** Process has terminated
- **Running → Ready:** End of time slice or higher-priority process is ready
- **Running → Blocked:** Process is waiting for an event
- **Blocked → Ready:** The event a process is waiting for has occurred, can continue
- **Ready → Exit:** Process terminated by O.S. or parent
- **Blocked → Exit:** Same reasons

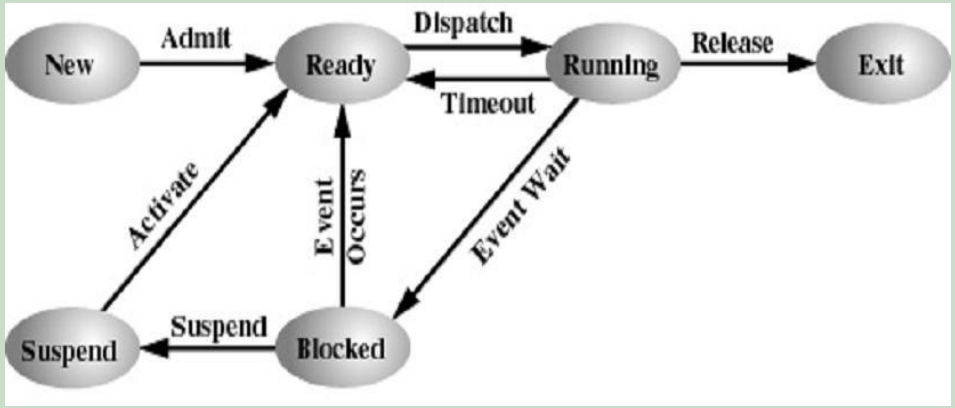
Multiple Blocked Queues



Suspended Processes

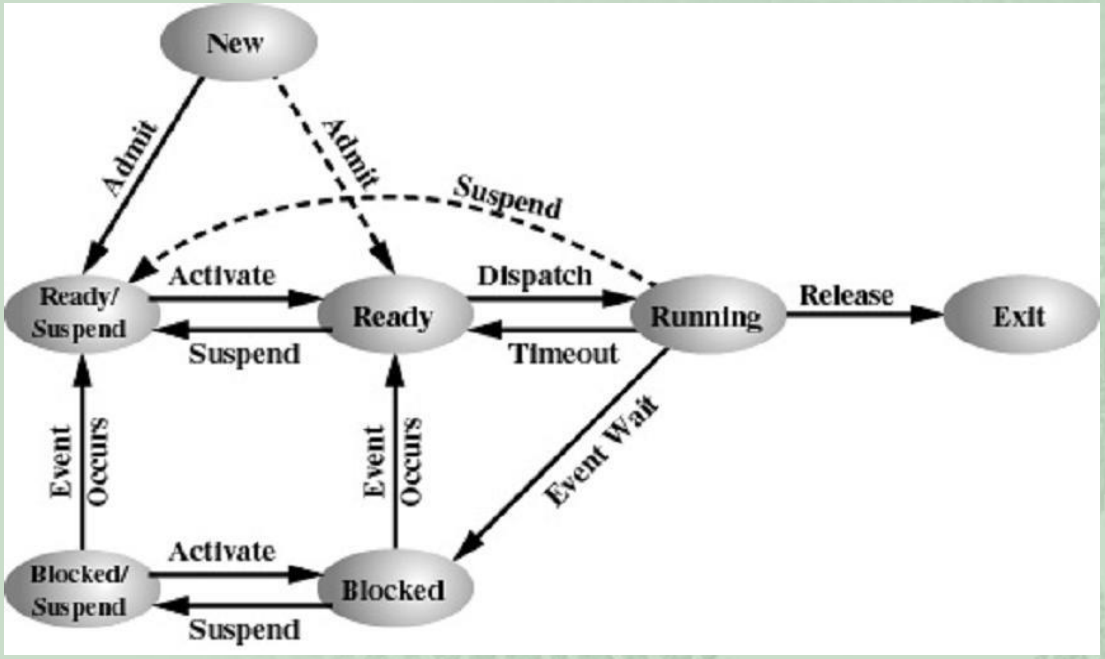
- Processor is faster than I/O so all processes could be waiting for I/O
- Swap these processes to disk to free up more memory
- Blocked state becomes suspended state when swapped to disk
- Two new states
  - Blocked, suspend
  - Ready, suspend

Suspended State Scheduling



One Suspend State

Two Suspend State



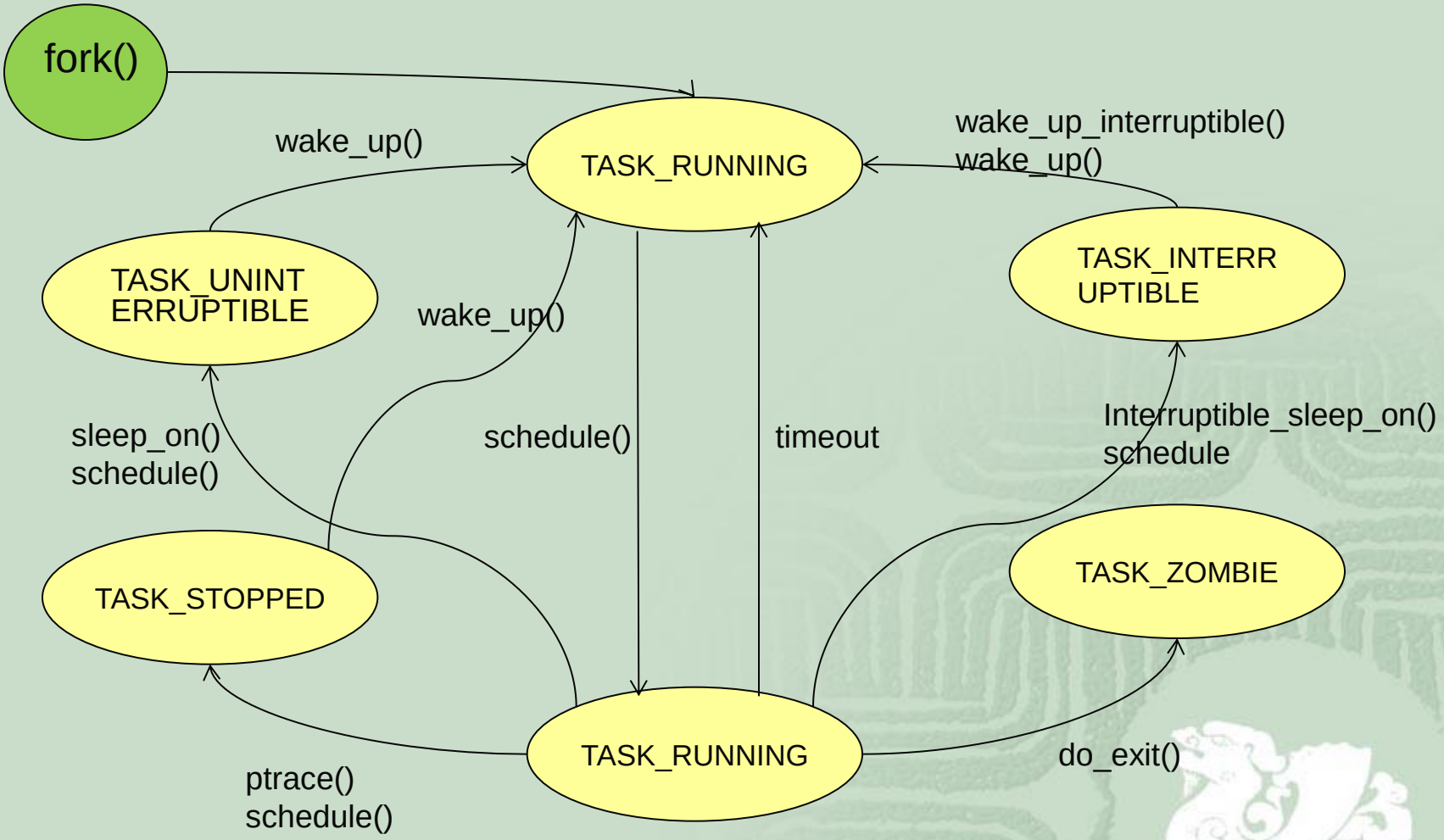
Reasons for Process Suspension

- Swapping
  - The OS needs to release sufficient main memory to bring in a process that is ready to execute
- Other OS reason
  - The OS 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 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

The ‘states’ of a Linux process

- A process can be in one of several states:
  - 0: TASK_RUNNING (R)
  - 1: TASK_INTERRUPTIBLE (S)
  - 2: TASK_UNINTERRUPTIBLE (D)
  - 4: TASK_ZOMBIE (Z)
  - 8: TASK_STOPPED (T)

State Transitions of a Linux Process



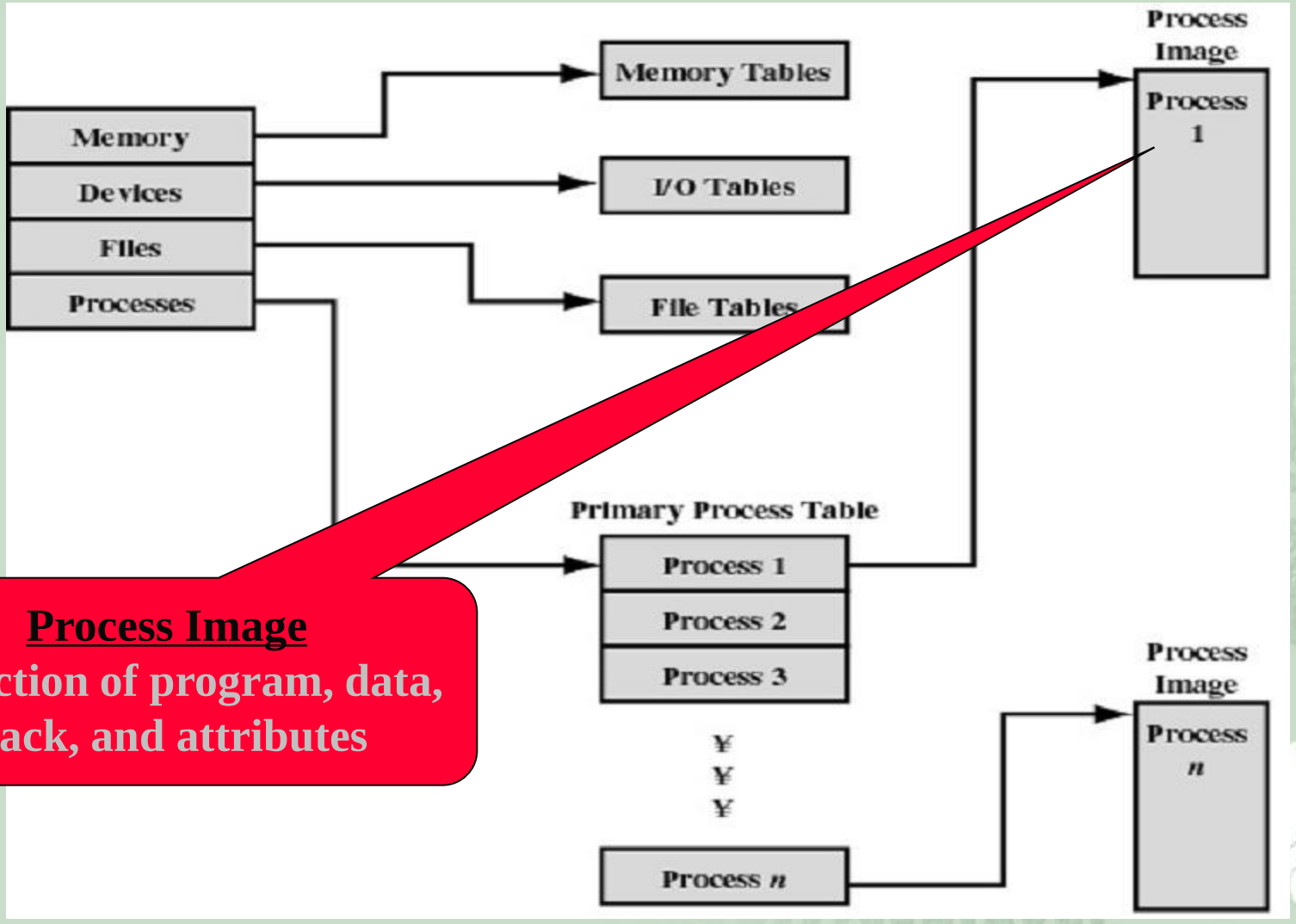
Process Concepts

- Process Definition
- Process Creation
- Process States and State Transitions
- Important Information Associated with Processes

Operating System Control Structures

- Information the OS needs to keep around:
  - Memory tables
  - I/O tables
  - File tables
  - Process tables
- Tables are constructed for each entity the operating system manages
- The OS has to manage and protect these tables

Process Control Structures



Control Tables

- Memory Tables
  - Allocation of main memory to processes
  - Allocation of secondary memory to processes
  - Protection attributes for access to shared memory regions
  - Information needed to manage virtual memory
- I/O device is available or assigned
  - Status of I/O operation
  - Location in main memory being used as the source or destination of the I/O transfer

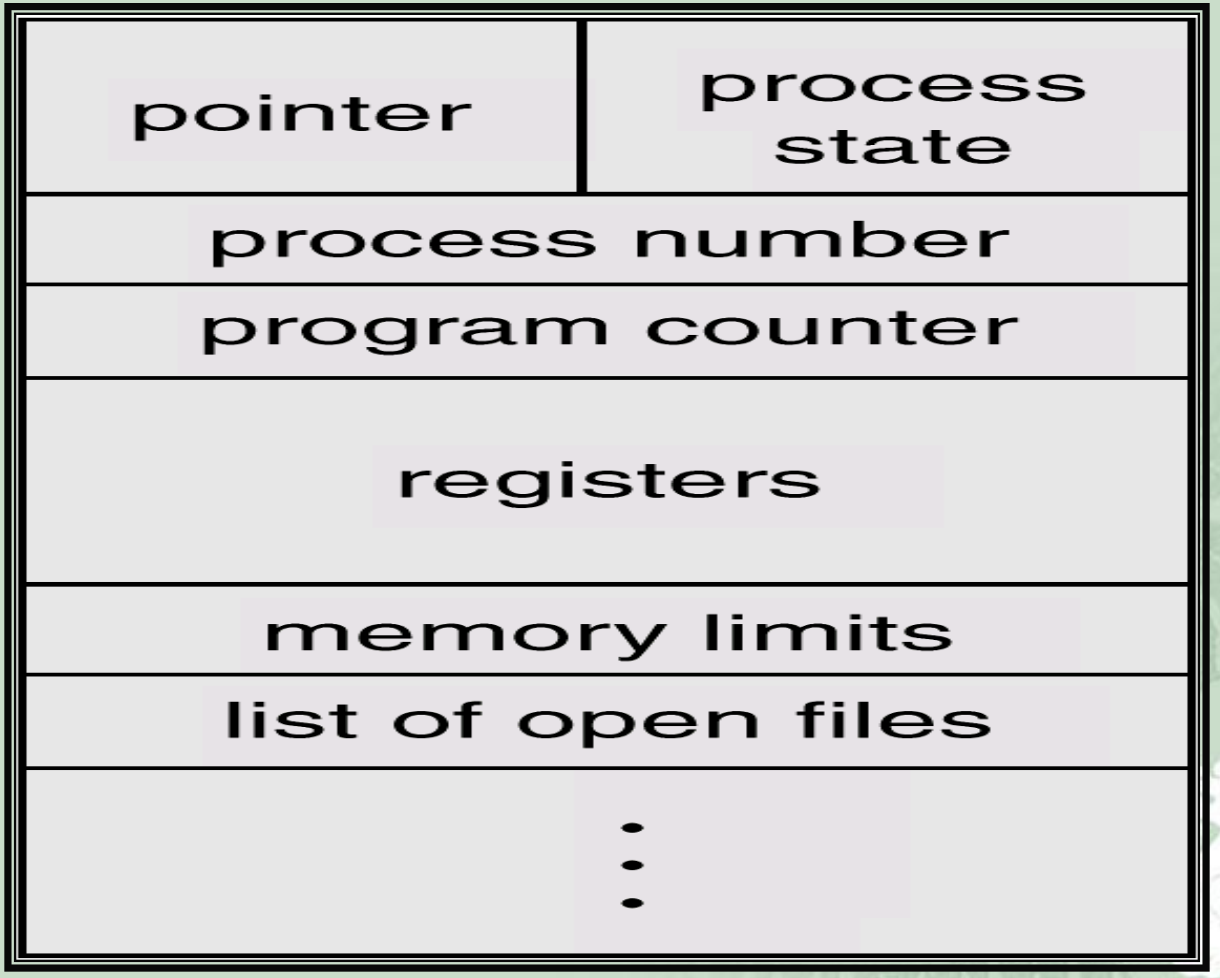
Control Tables (continued…)

- File Tables
  - Existence of files
  - Location on secondary memory
  - Current Status
  - Attributes
  - Usually maintained by a file management system
- Process Table
  - Process ID
  - Process state
  - Location in memory

Process Location

- A Process includes:
  - Data locations for local and global variables
  - Any defined constants
  - Stack
  - Heap
  - Set of programs to be executed
- Process Control Block (PCB)
  - Collection of attributes

Process Control Block (PCB)



Process Control Block

- Process identification
- Processor State Information
  - User-Visible Registers
  - Control and Status Registers
  - Stack Pointers
- Process Control Information
  - Scheduling and State Information
    - *Process state*
    - *Priority*
    - *Scheduling-related information*
    - *Event*
  - Data Structuring
  - Interprocess Communication
  - Process Privileges
  - Memory Management
  - Resource Ownership and Utilization

Process Control Block (continued…)

- Process identification
  - Numeric identifiers that may be stored with the process control block and include
    - Identifier of this process
    - Identifier of creator process (parent process)
    - User identifier
- Processor State Information
  - User-Visible Registers
  - Control and Status Registers
  - Stack Pointers

Process Control Block (continued…)

- Processor State Information
  - User-Visible Registers
    - A register is one that may be referenced by means of the machine language that the processor executes. (Typically, 8 to 32)
  - Control and Status Registers
    - •*Program counter:* Address of the next instruction to be fetched
    - •*Condition codes:* Results of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
    - •*Status information:* Interrupt enabled/disabled flags, execution mode
  - 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.

Several dozen fields

- Dozens of separate items of information are kept in a Linux ‘task_struct’; e.g.:
  - pid; // process ID-number
  - state; // current task-state
  - priority; // current task-priority
  - start_time; // time when task was created
  - sleep_time; // time when task began sleeping
  - . . . // … many more items …
- This info is used by the Linux ‘scheduler’

Execution Stack in C

- The C runtime system keeps track of the chain of active functions with a stack
- When a function is called, the runtime system pushes on the stack a frame containing
  - Local variables and return value
  - Program counter, keeping track of the statement being executed
- When a function ends, its frame is popped from the stack and control is passed to the function on top of the stack

```
main() {
   int i = 5;
   foo(i);
   }

foo(int j) {
   int k;
   k = j+1;
   bar(k);
   }

bar(int m) {
   …
   }
```