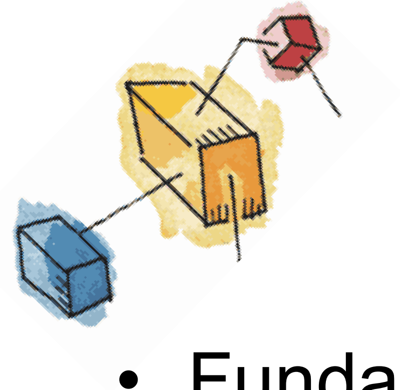
Abstraction: Process

Objectives

- How are processes represented and controlled by the OS.
- ***Process states*** which characterize the behaviour of processes.
- ***Data structures*** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.

Time Sharing Systems

- Multiple interactive jobs running simultaneously
- OS needs to manage and control the execution of multiple programs to time-share the use of processor(s)
  - Process is an abstraction to virtualize CPU resource



Process

- Fundamental to the structure of operating systems

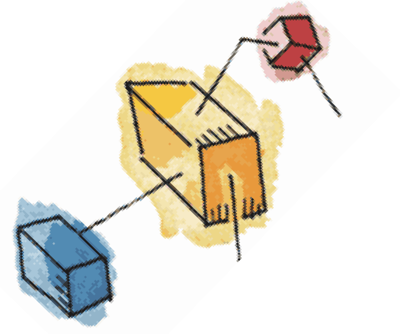
A *process* can be defined as:

a program in execution

an instance of a running program

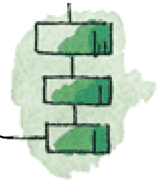
the entity that can be assigned to, and executed on, a processor

a unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources



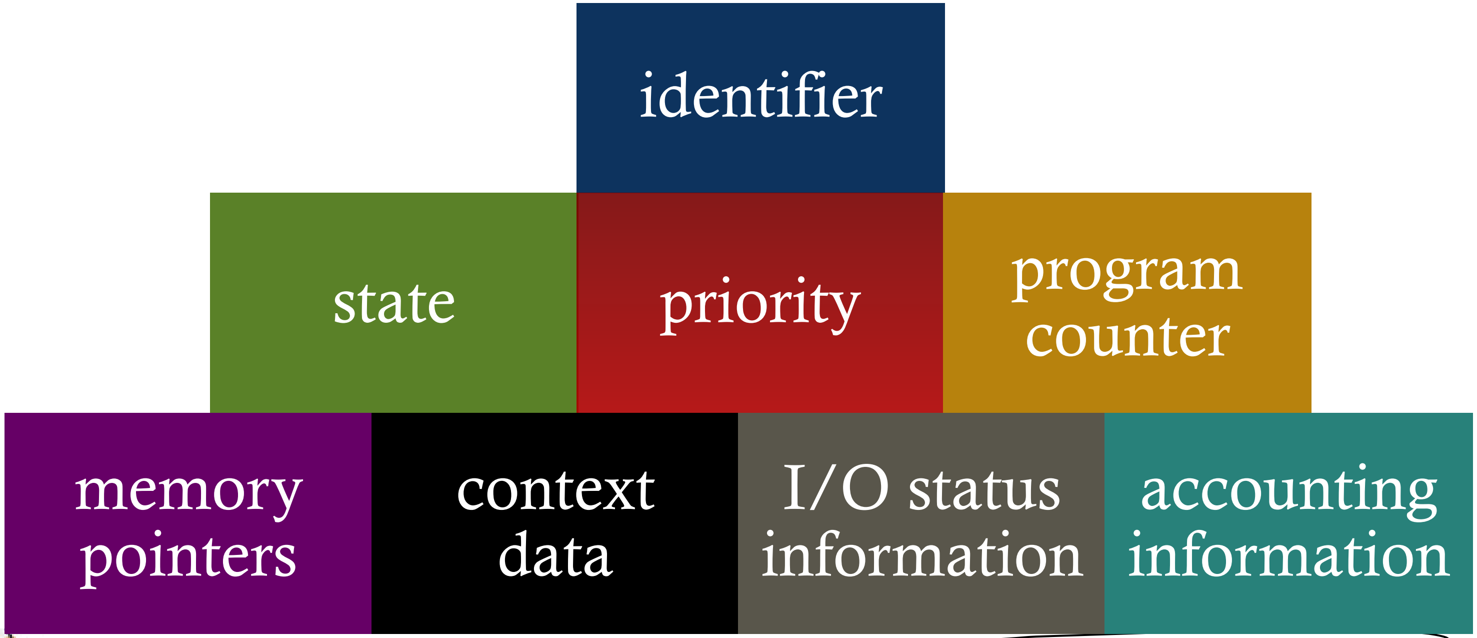
Process Management

- Is the *Fundamental Task*
- The Operating System must
  - Allocate resources to processes, and protect the resources of each process from other processes
  - Interleave the execution of multiple processes
  - Enable processes to share and exchange information
  - Enable synchronization among processes



Process Elements

- While the program is executing, this process can be uniquely characterized by a number of *attributes*, including:



identifier

state

priority

program counter

memory pointers

context data

I/O status information

accounting information

Process Control Block

- The most important data structure in an OS
- Contains the process attributes
- Created and managed by the operating system
- Key tool that allows support for multiple processes
- Attributes in general categories:
  - Process identification
  - Processor state information
  - Process control information

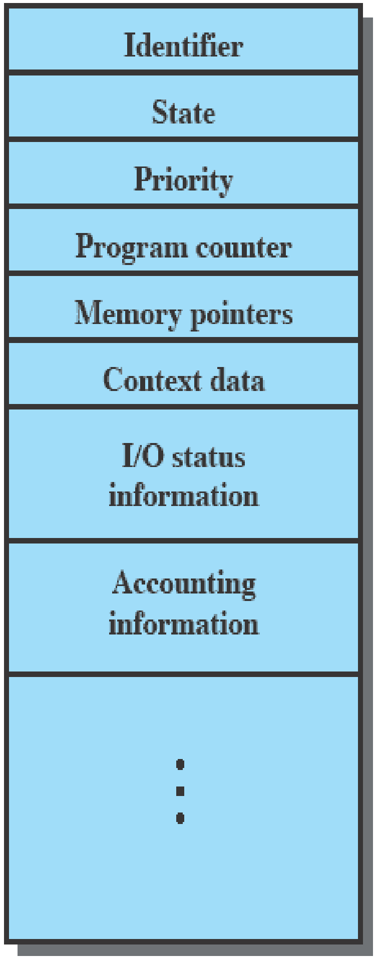
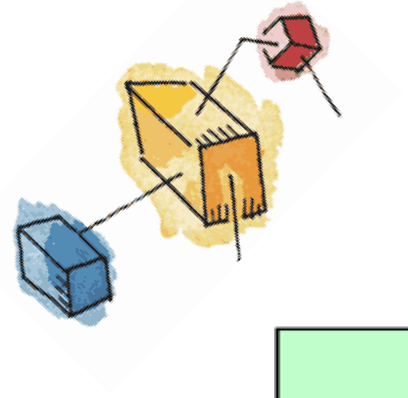
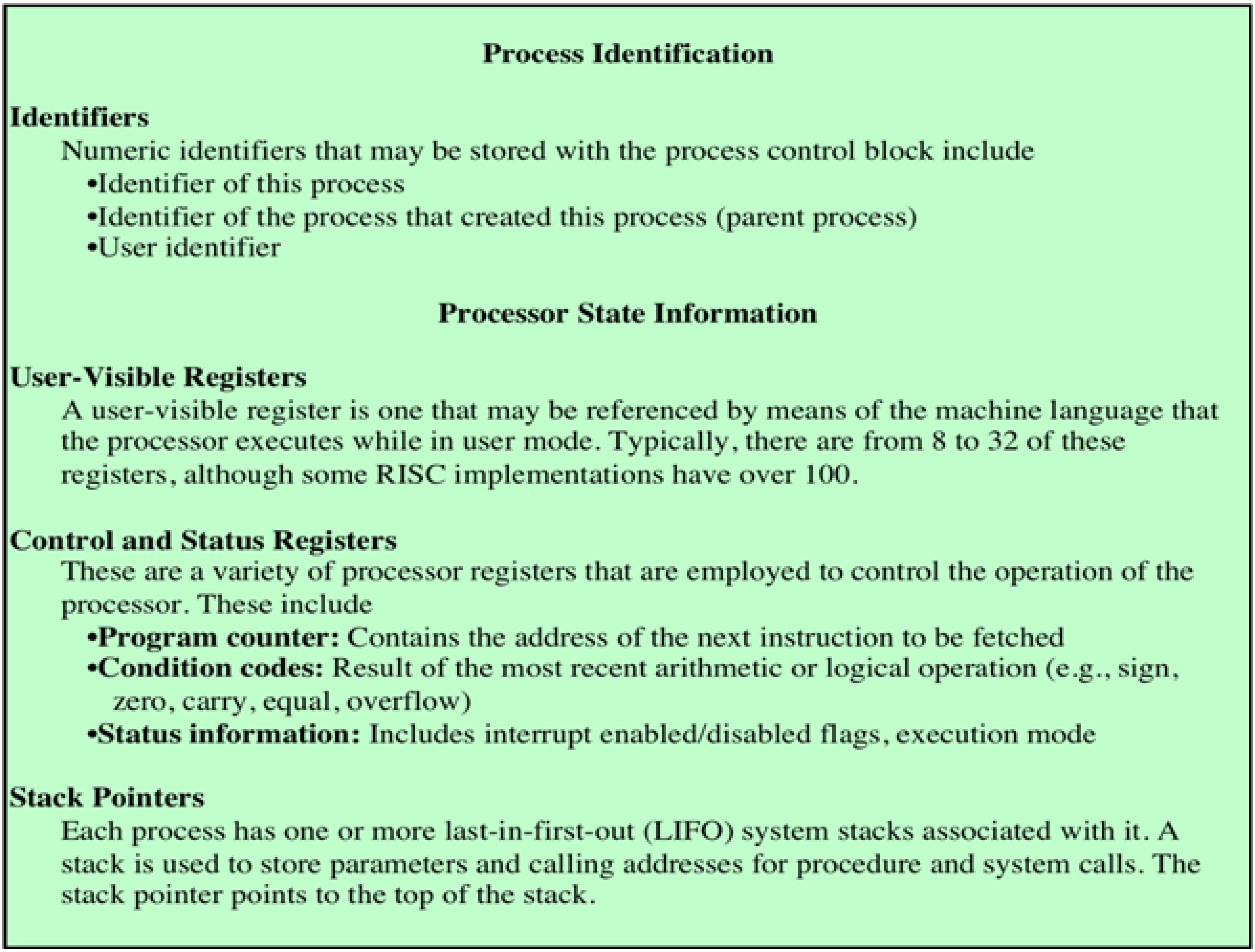


Figure 3.1 Simplified Process Control Block



Process Attributes



Process Identification

Identifiers

Numeric identifiers that may be stored with the process control block include

- Identifier of this process
- Identifier of the process that created this process (parent process)
- User identifier

Processor State Information

User-Visible Registers

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

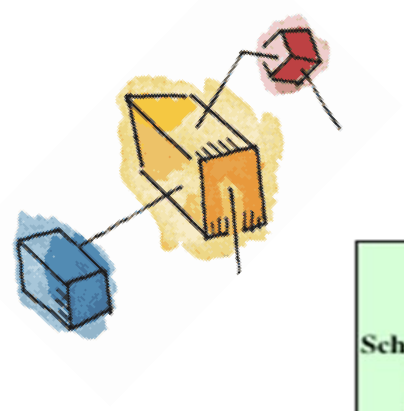
Control and Status Registers

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

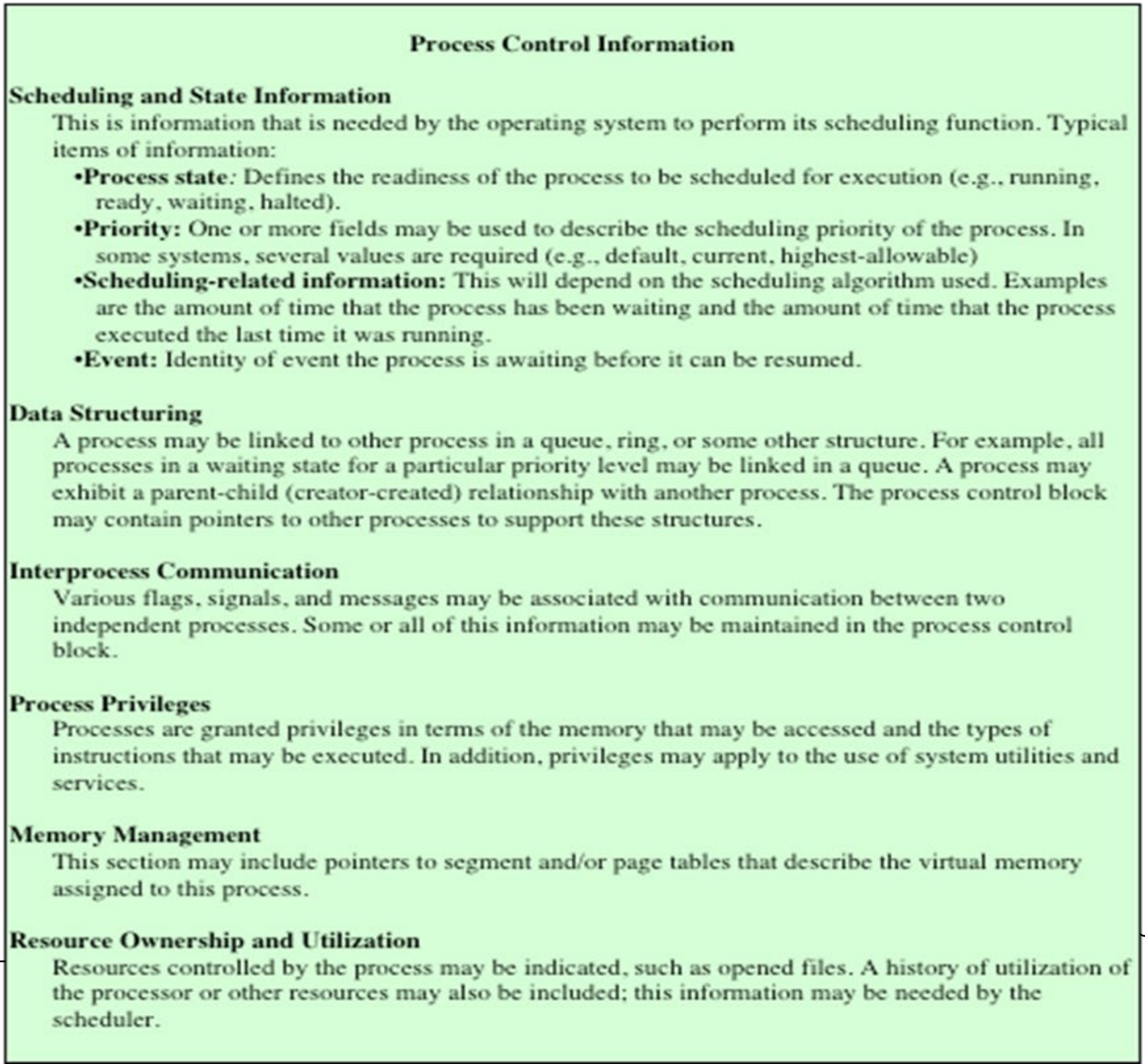
- **Program counter:** Contains the address of the next instruction to be fetched
- **Condition codes:** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- **Status information:** Includes interrupt enabled/disabled flags, execution mode

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.



Process Attributes



Process Control Information

Scheduling and State Information

This is information that is needed by the operating system to perform its scheduling function. Typical items of information:

- **Process state**: Defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
- **Priority:** One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable)
- **Scheduling-related information:** This will depend on the scheduling algorithm used. Examples are 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.

Data Structuring

A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting 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. The process control block may contain pointers to other processes to support these structures.

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

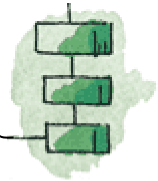
Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

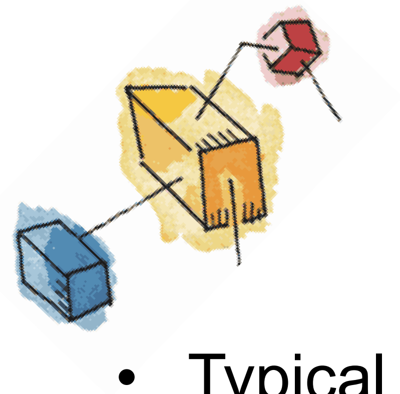
Memory Management

This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

Resource Ownership and Utilization

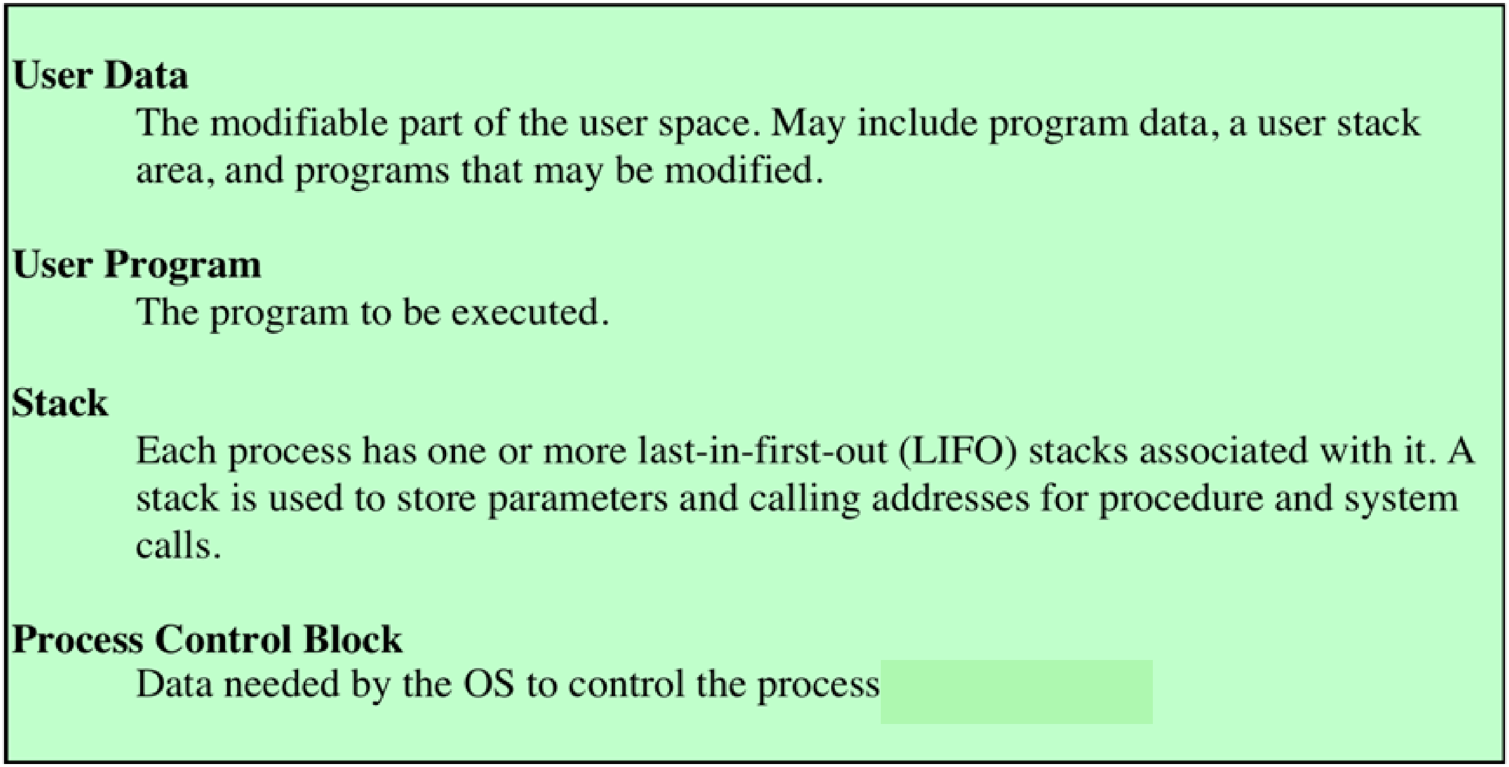
Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.





Process Image

- Typical elements:



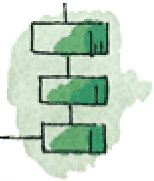
User Data
The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.

User Program
The program to be executed.

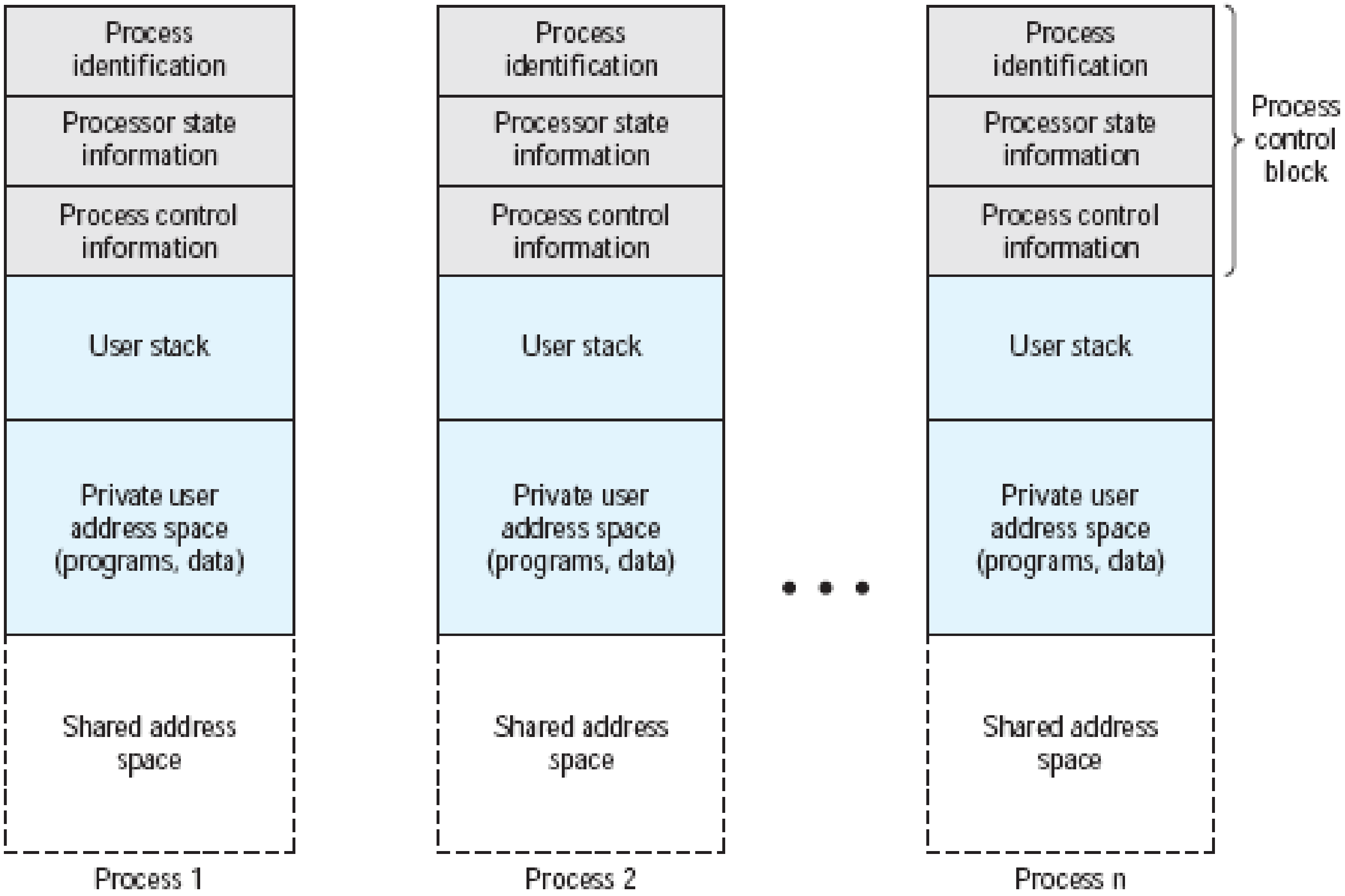
Stack
Each process has one or more last-in-first-out (LIFO) stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.

Process Control Block
Data needed by the OS to control the process

- Process image location will depend on the memory management scheme being used



Structure of Process Images



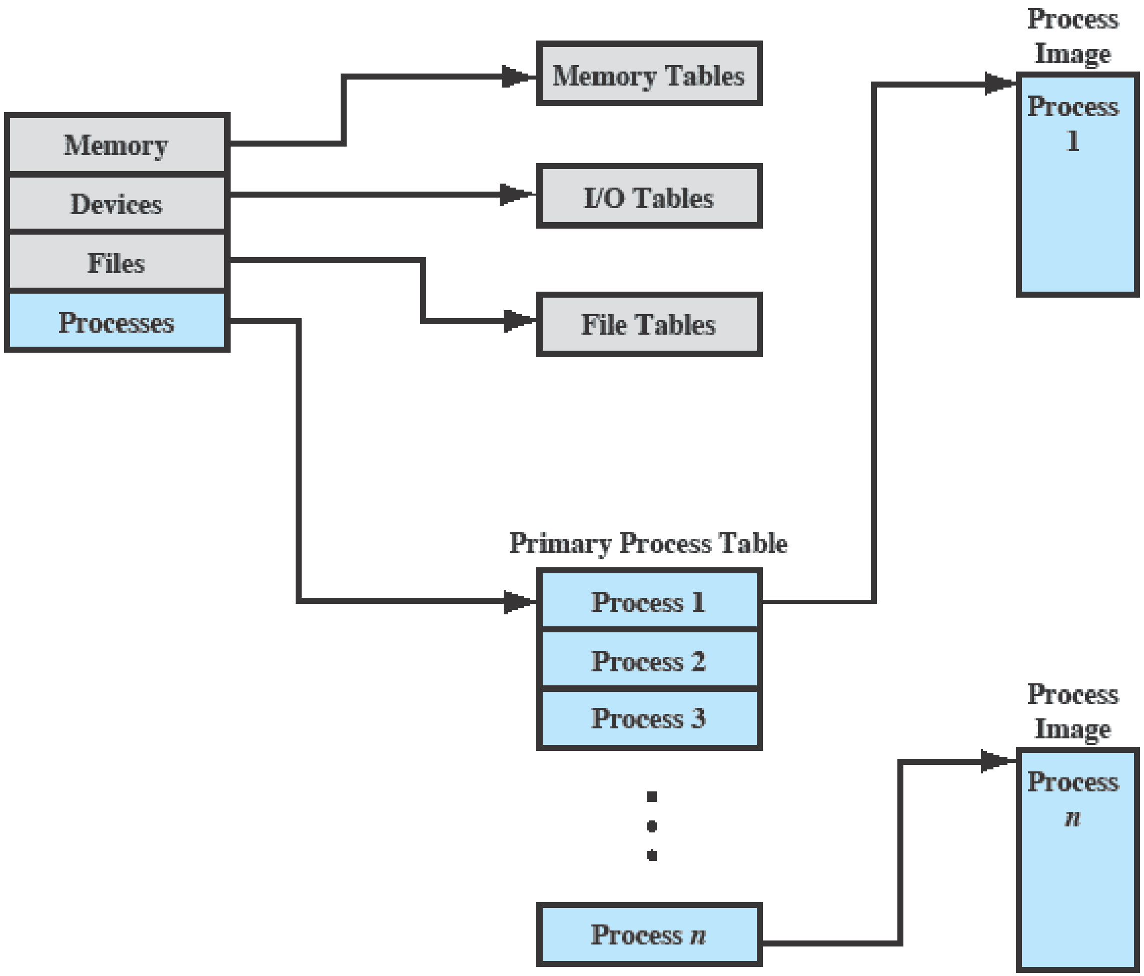
Process identification	Process identification		Process identification	Process control block
Processor state information	Processor state information		Processor state information	
Process control information	Process control information		Process control information	
User stack	User stack		User stack	
Private user address space (programs, data)	Private user address space (programs, data)	• • •	Private user address space (programs, data)	
Shared address space	Shared address space		Shared address space	
Process 1	Process 2		Process n	

User Processes in Virtual Memory

OS Control Structures

- For the OS to manage processes and resources, it must have information about the current status of each process and resource.
- Tables are constructed for each entity (memory, I/O and files) the operating system manages

OS Control Tables



Memory

Devices

Files

Processes

Memory Tables

I/O Tables

File Tables

Primary Process Table

Process 1

Process 2

Process 3

Process *n*

Process Image

Process 1

Process Image

Process *n*

Figure 3.11 General Structure of Operating System Control Tables

Process Tables

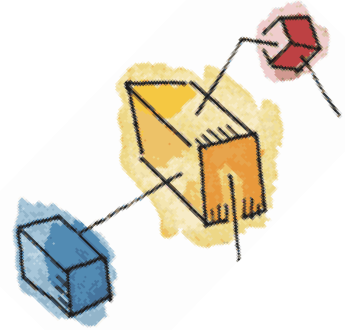
- Must be maintained to manage processes
- The OS tables must be linked or cross-referenced
  - Memory, I/O and files are managed on behalf of processes, there are some references to these resources, directly or indirectly , in the process tables.

Process Dispatching

- How does the operating system decide which process to run next?
- It's a two-part operation, separating the policy from the mechanism
  - Scheduler
    - Make decisions based on process and user priorities, and past process behavior, as to which processes to run next
    - We will study this more in later lectures
  - Dispatcher
    - Implements the scheduler's decisions
    - This means the mechanisms to collect the necessary performance data for the schedule, quickly finding the next process to run, and switching to the next process

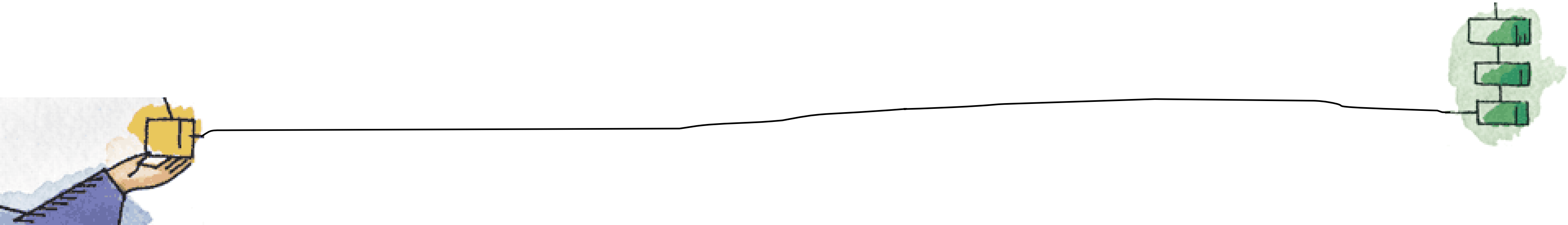
Traps and Interrupts

- CPU can only be doing one thing at a time
  - If user process is executing, dispatcher is not - OS has lost control
- How does OS regain control of processor?
- Internal events (things occurring within user process)
  - System call
  - Error (illegal instruction, addressing violation, etc.)
  - Page fault
- These are also called traps
- External events (things occurring outside the control of the user process)
  - Character typed at terminal
  - Completion of disk operation (controller is ready for more work)
  - Timer: to make sure OS eventually gets control
- External events are usually called interrupts
- Traps and interrupts all cause a state switch into the OS

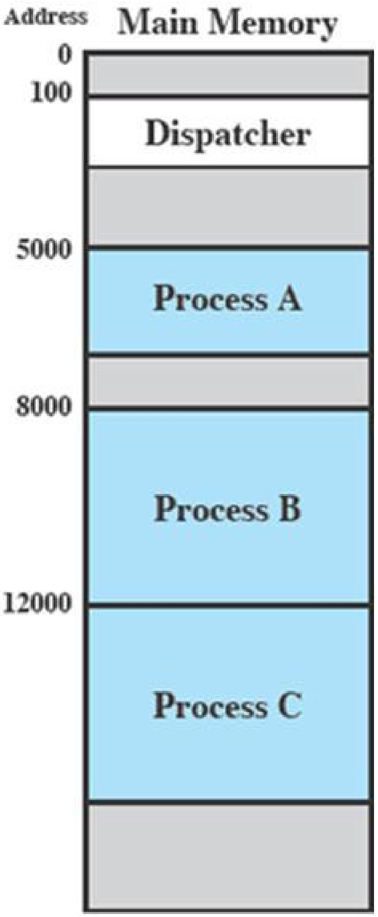


Timer-based Interrupts

- Guarantee OS can obtain control periodically
- Enter OS by enabling periodic alarm clock
  - Hardware generates timer interrupt (CPU or separate chip) Example: Every 10ms
  - User must not be able to mask timer interrupt

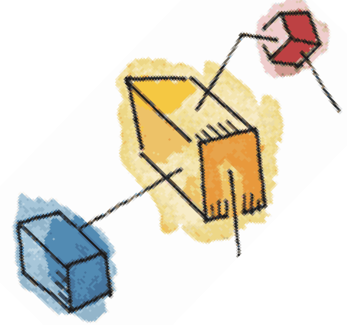


Process Execution

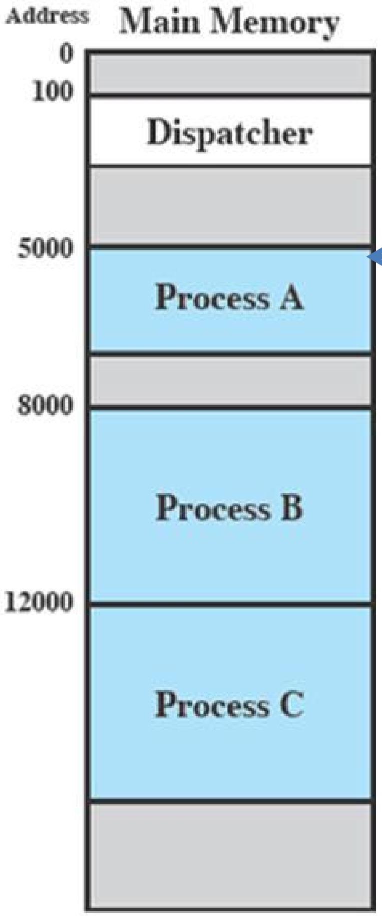


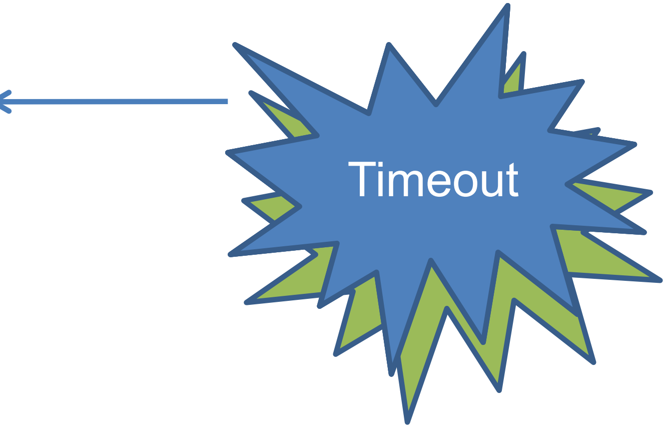
- ***Dispatcher*** is a small program which switches the processor from one process to another

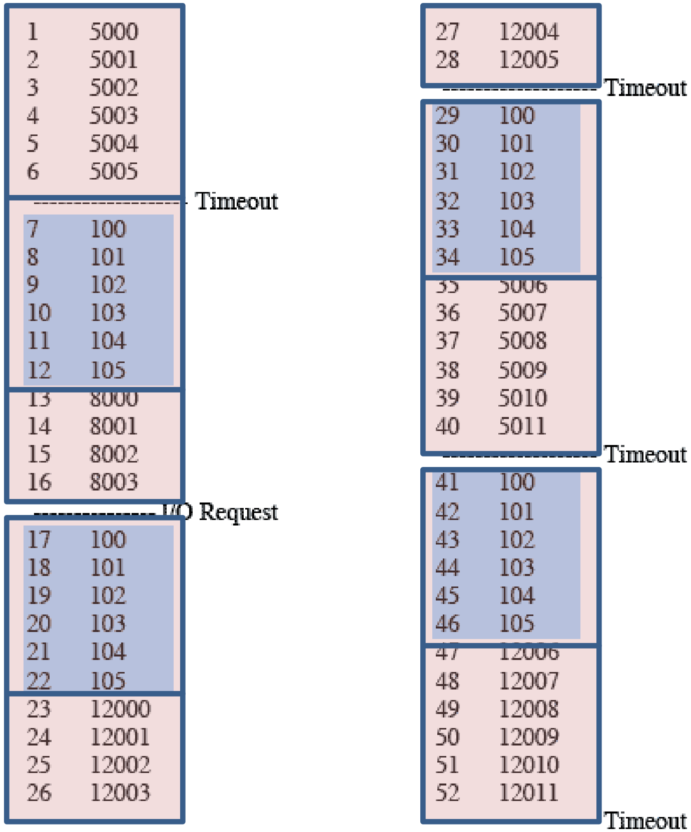
- Consider three processes being executed
- All are in memory (plus the dispatcher)



Trace from Processors point of view







1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
Timeout	
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

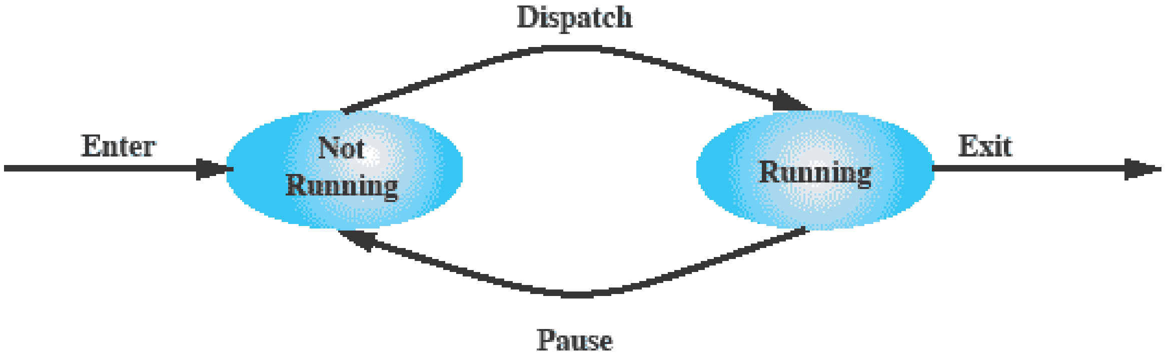
27	12004
28	12005
Timeout	
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
Timeout	
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
Timeout	

100 = Starting address of dispatcher program

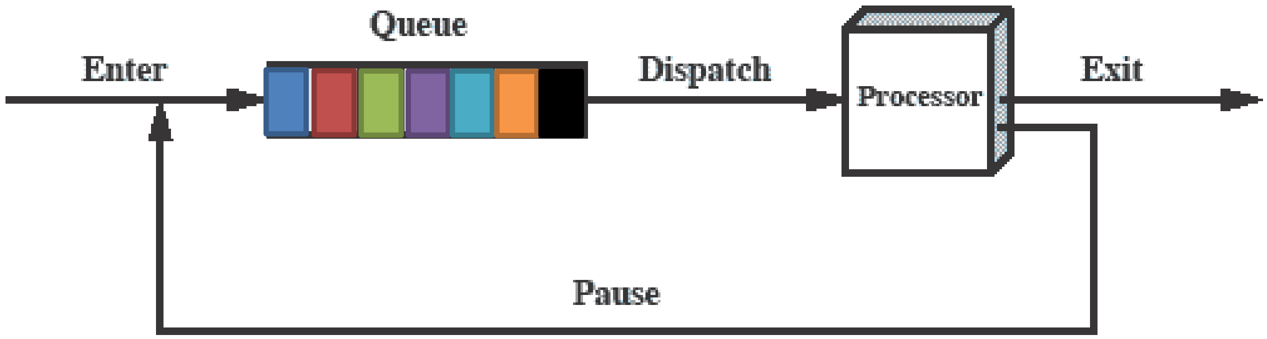
Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Two-State Process Model

- The state of a process may be defined by the current activity of that process
  - Used to describe the behaviour that we would like each process to exhibit
- E.g., assume process has only two states
  - Running
  - Not-running



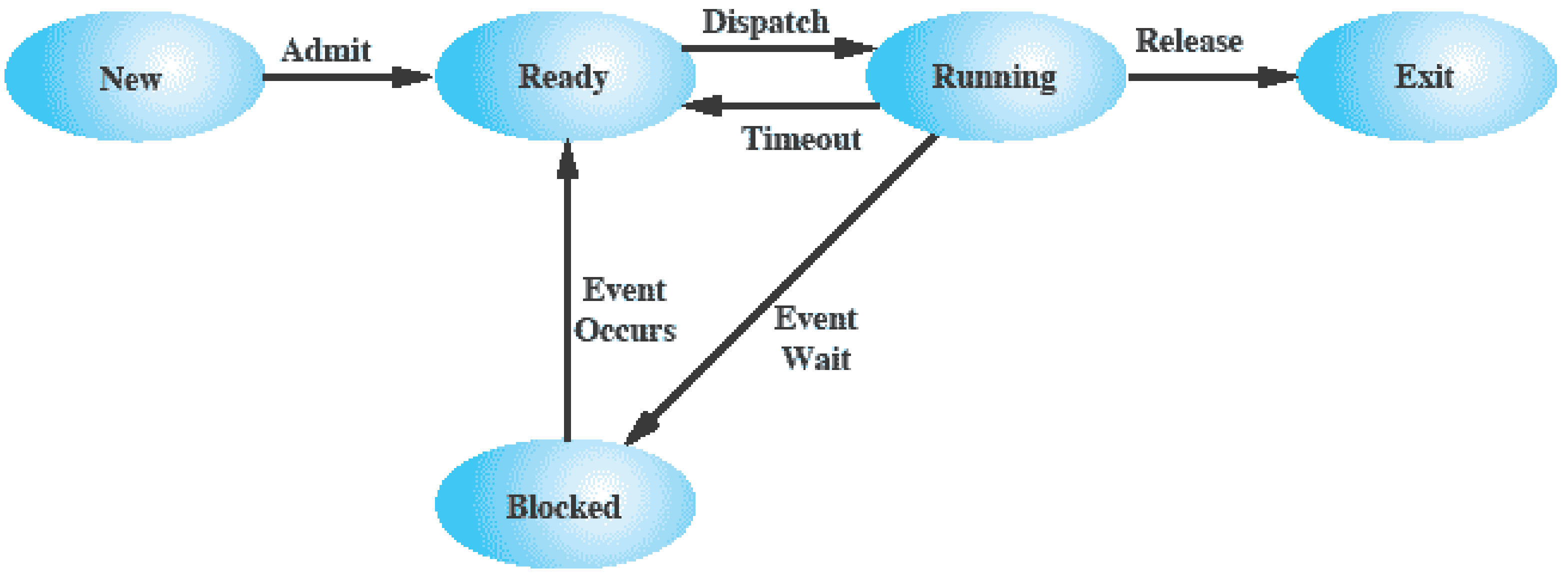
Queuing Diagram



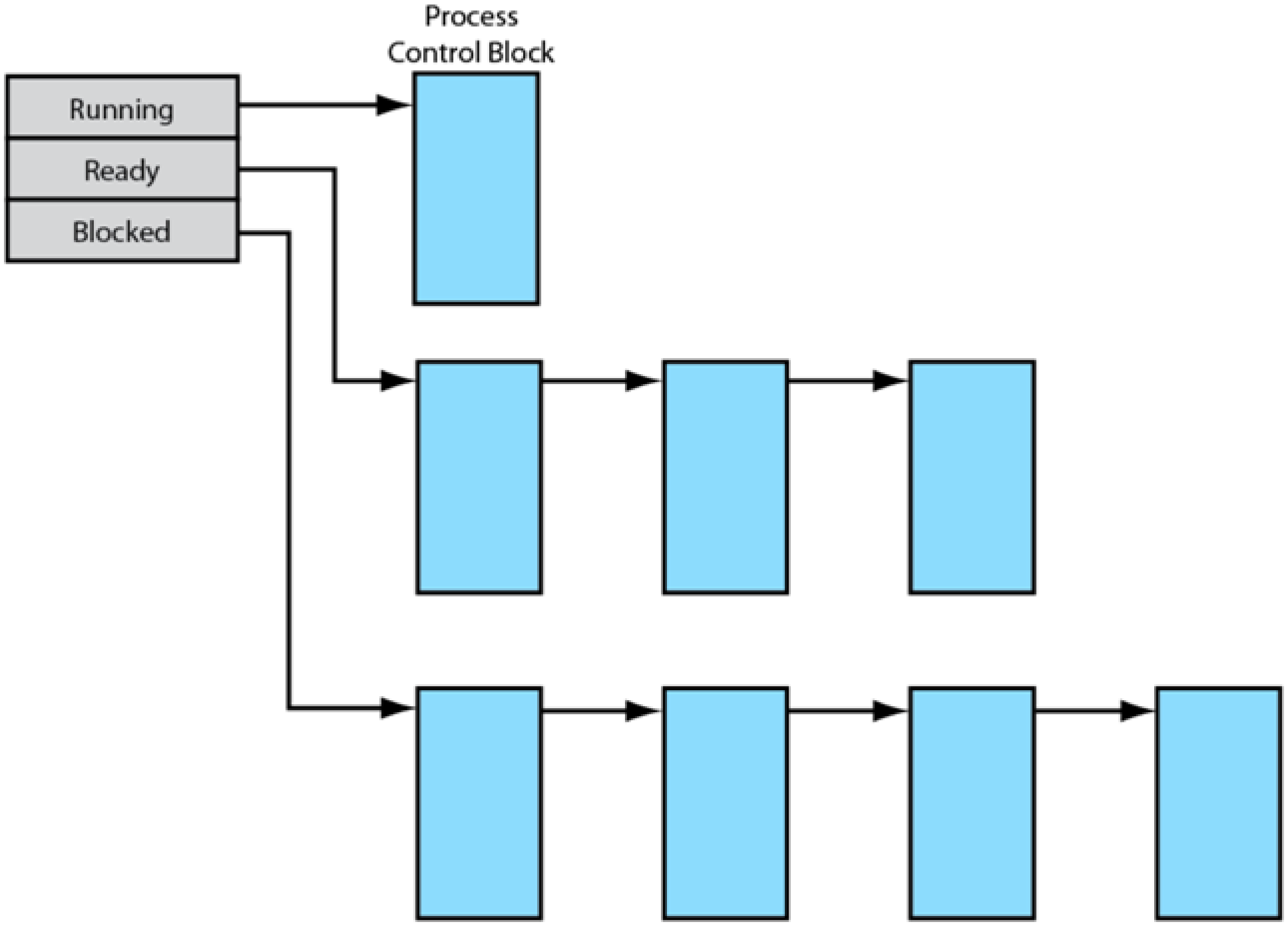
(b) Queuing diagram

Etc … processes moved by the dispatcher of the OS to the CPU then back to the queue until the task is completed

Five-State Process Model



Process List Structures



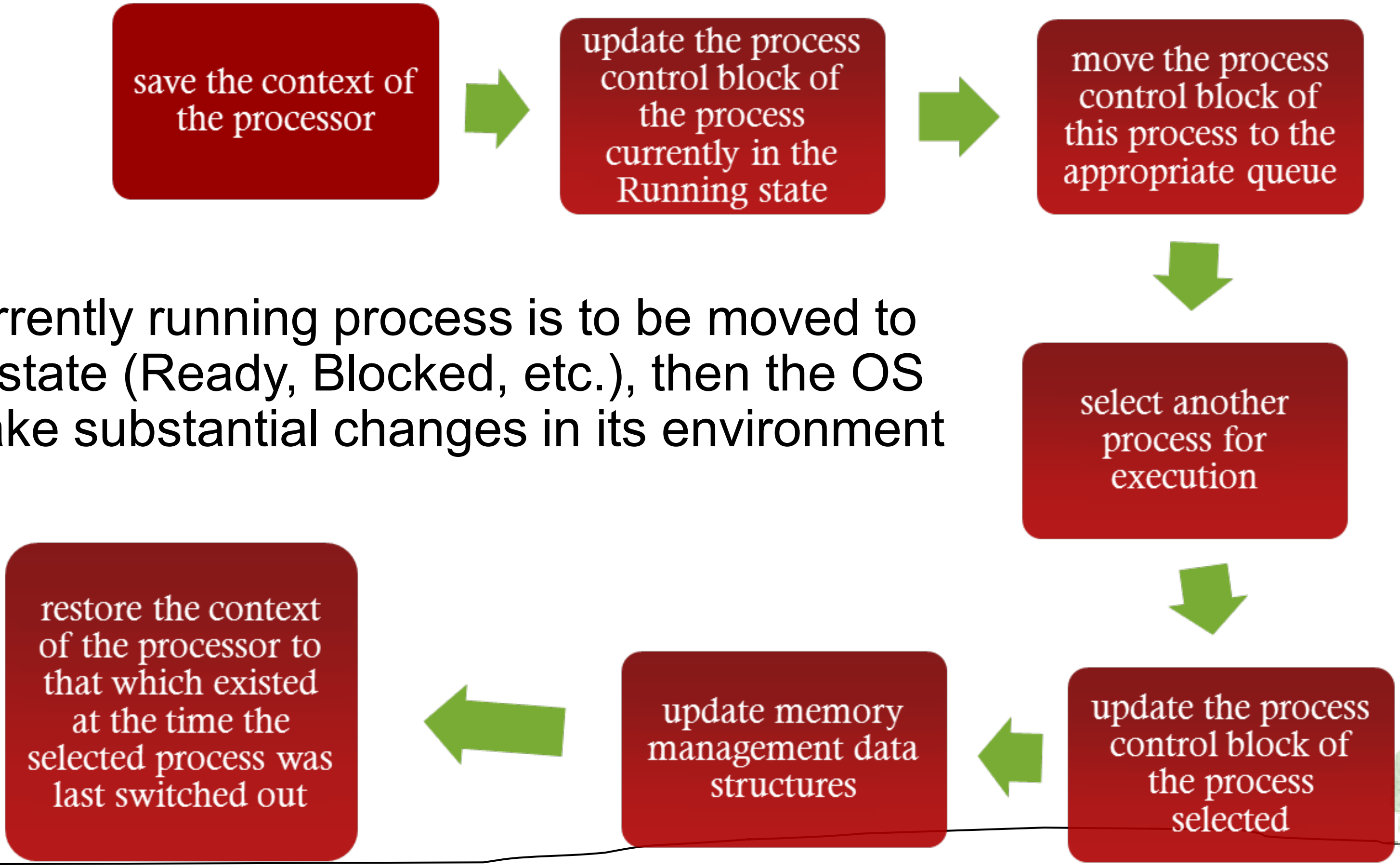
Switching Processes

- When process is not running, its state must be saved in its process control block
- What gets saved?
  - Everything that next process could trash
    - Program counter
    - Processor status word (condition codes, etc.)
    - General purpose registers

Change of Process State

- The steps in a process switch are:

If the currently running process is to be moved to another state (Ready, Blocked, etc.), then the OS must make substantial changes in its environment



save the context of the processor

update the process control block of the process currently in the Running state

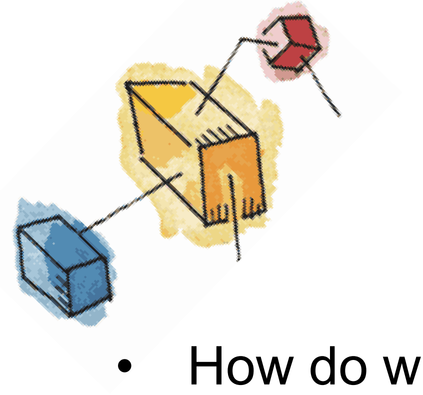
move the process control block of this process to the appropriate queue

select another process for execution

update the process control block of the process selected

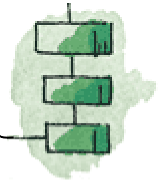
update memory management data structures

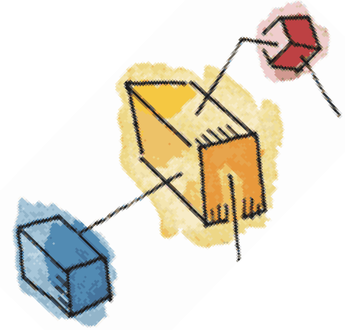
restore the context of the processor to that which existed at the time the selected process was last switched out



Switching Processes

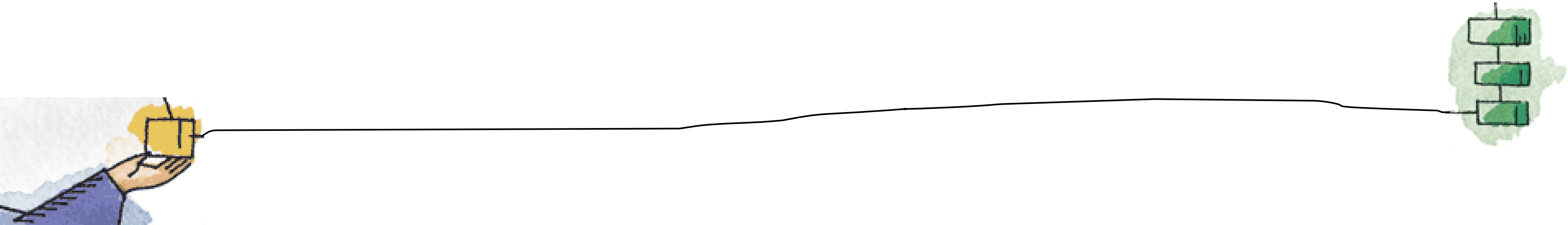
- How do we switch contexts between the user and OS?
  - Must be careful not to mess up process state while saving and restoring it
- All machines provide some special hardware support for saving and restoring state
  - Most modern processors: hardware does not know much about processes
    - It just moves PC and PS to/from the stack
    - OS then transfers to/from PCB, and handles rest of state itself
- Full save/restore everything is expensive
  - Sometimes different amounts are saved at different times, e.g., to handle interrupts, might save only a few registers
  - Sometimes state can be saved and restored incrementally





Process Creation

- Once the OS decides to create a new process it
  - Assign a unique process identifier to the new process
  - Create and initialize the process control block
  - Allocate memory space for the process
  - Set the appropriate linkages
  - Create or expand other data structures

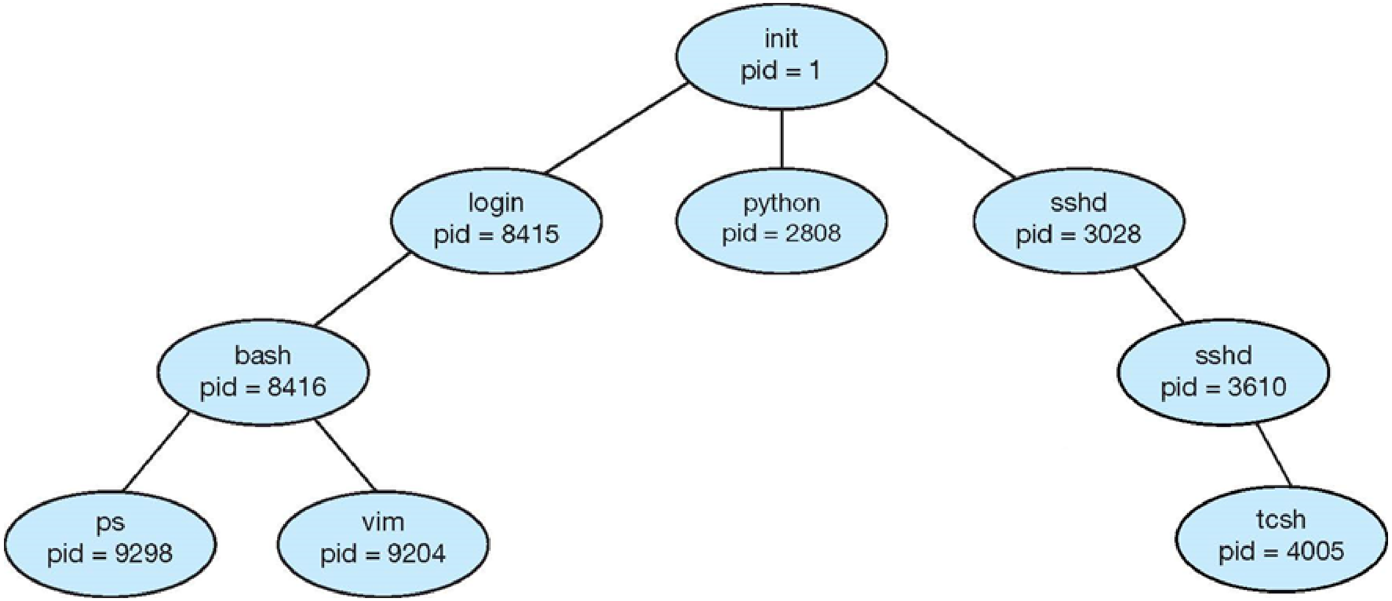


Process Creation (cont.)

- Traditionally, the OS created all processes
  - But it can be useful to let a running process create another
- This action is called ***process spawning***
  - ***Parent Process*** is the original, creating, process
  - ***Child Process*** is the new process
- Parent process create children processes, which, in turn create other processes, forming a tree of processes

Process Creation (cont.)

- A Tree of Processes in UNIX/Linux:



UNIX Process Creation

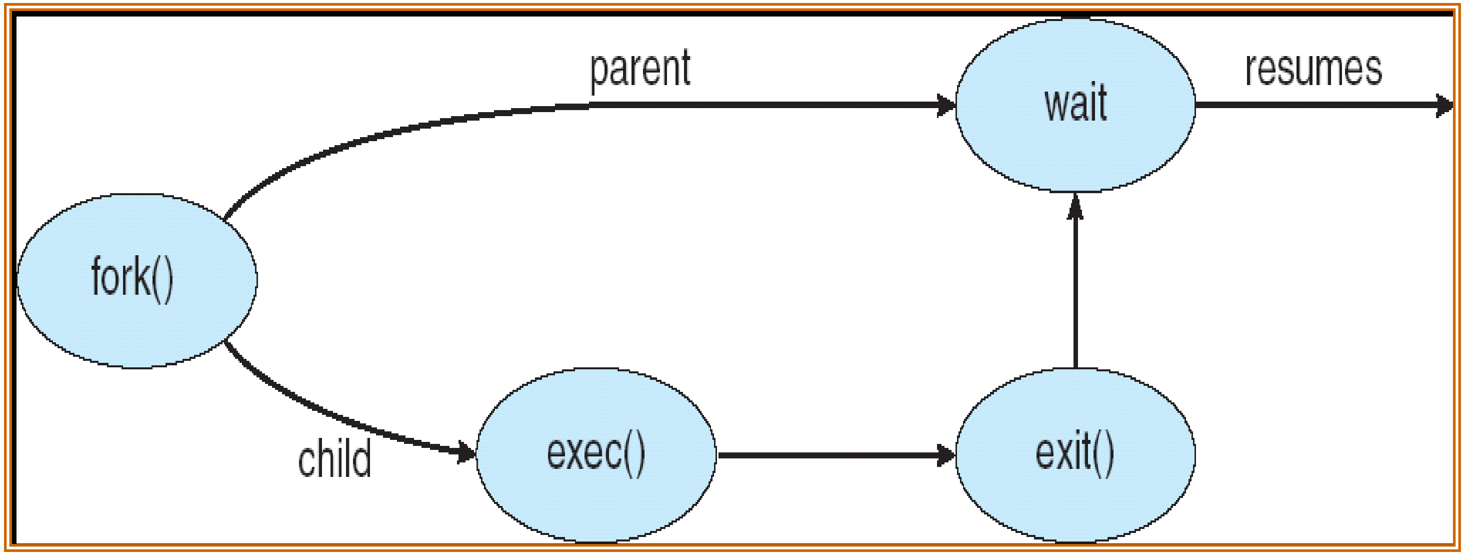
- Process creation is by means of the kernel system call, fork( ).
- This causes the OS, in Kernel Mode, to:

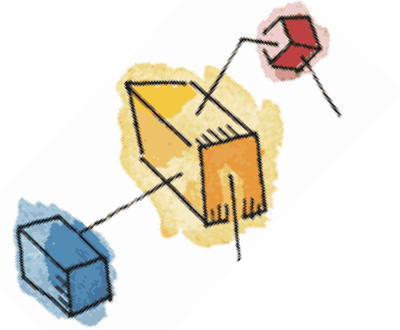
1. Allocate a slot in the process table for the new process.
2. Assign a unique process ID to the child process.
3. Copy of process image of the parent, with the exception of any shared memory.
4. Increment the counters for any files owned by the parent, to reflect that an additional process now also owns those files.
5. Assign the child process to the Ready state.
6. Returns the ID number of the child to the parent process, and a 0 value to the child process.

UNIX Process Creation (cont.)

- After creating the process, the Kernel can do one of the following, as part of the dispatcher routine:
  - Stay in the parent process
  - Transfer control to the child process
  - Transfer control to another process

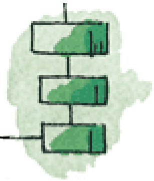
UNIX Process Creation (cont.)





Process Creation

```
int main()
{
pid_t  pid;
    /* fork another process */
    pid = fork();
    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);
    }
}
```



Process Termination

- There must be some way that a process can indicate completion.
- This indication may be:
  - A HALT instruction generating an interrupt alert to the OS.
  - A user action (e.g. log off, quitting an application)
  - A fault or error
  - Parent process terminating

Summary

- The principal function of the OS is to create, manage, and terminate processes
- The most fundamental concept in a modern OS is the process
- Process control block contains all of the information that is required for the OS to manage the process, including its current state, resources allocated to it, priority, and other relevant data
- The most important states are Ready, Running and Blocked
  - The running process is the one that is currently being executed by the processor
  - A blocked process is waiting for the completion of some event
  - A running process is interrupted either by an interrupt or by executing a system call to the OS