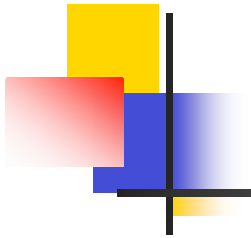
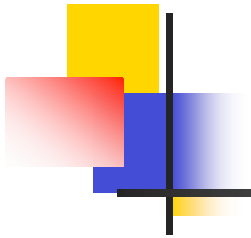
Operating Systems

Processes



Process Concept

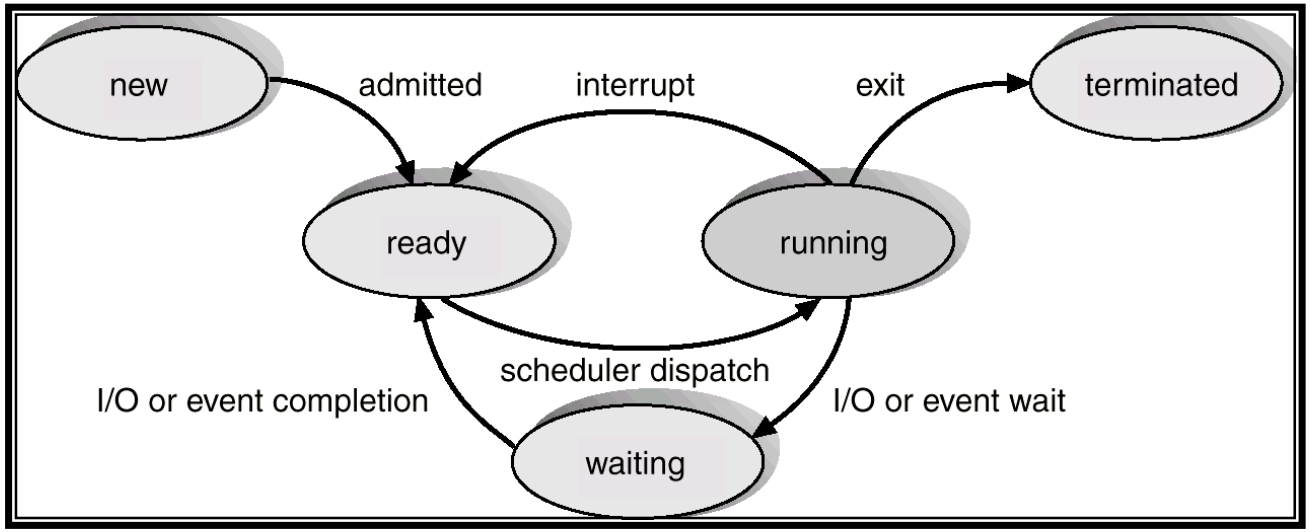
- An operating system executes a variety of programs.
- Process
  - Sometimes referred to as *job* or *task*
  - a program in execution
  - process execution must progress in sequential fashion.
- A process includes:
  - program counter
  - instructions
  - stack
  - data section

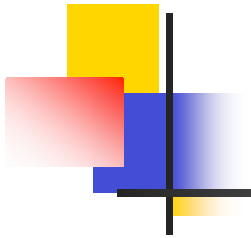


Process State

- As a process executes, it changes *state*
  - **new**: The process is being created.
  - **running**: Instructions are being executed.
  - **waiting**: The process is waiting for some event to occur.
  - **ready**: The process is waiting to be assigned to a processor.
  - **terminated**: The process has finished execution.

Diagram of Process State



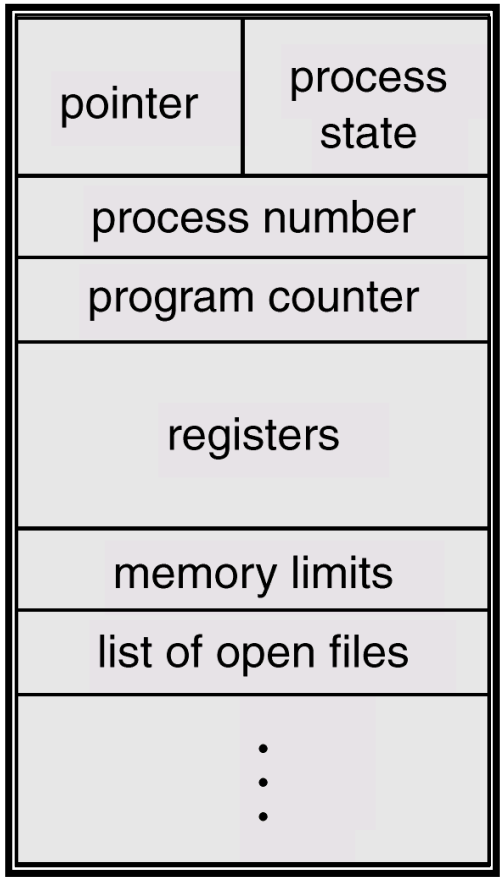


Process Control Block (PCB)

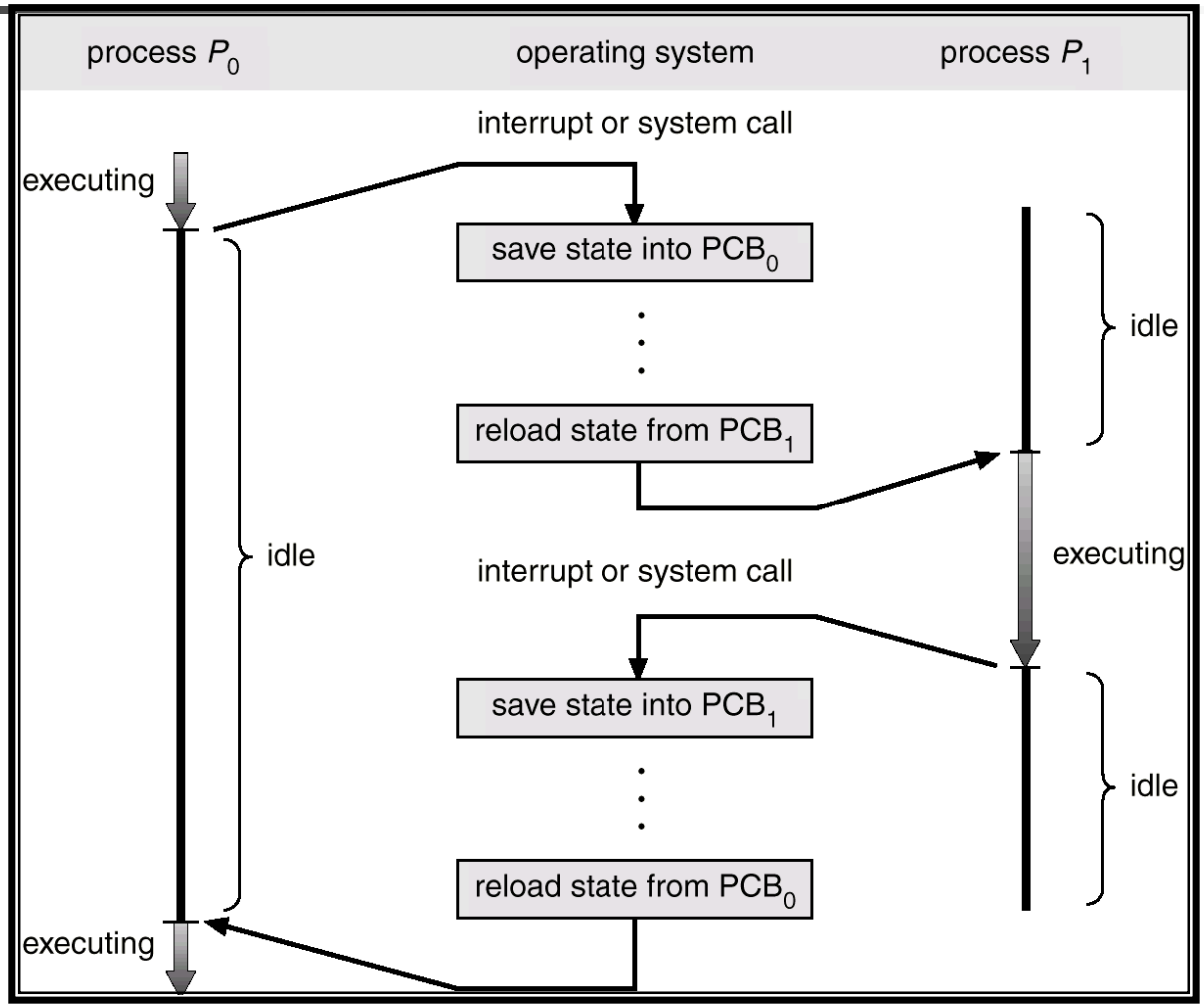
Information associated with each process.

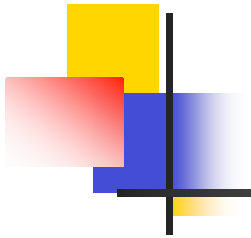
- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



CPU Switch From Process to Process

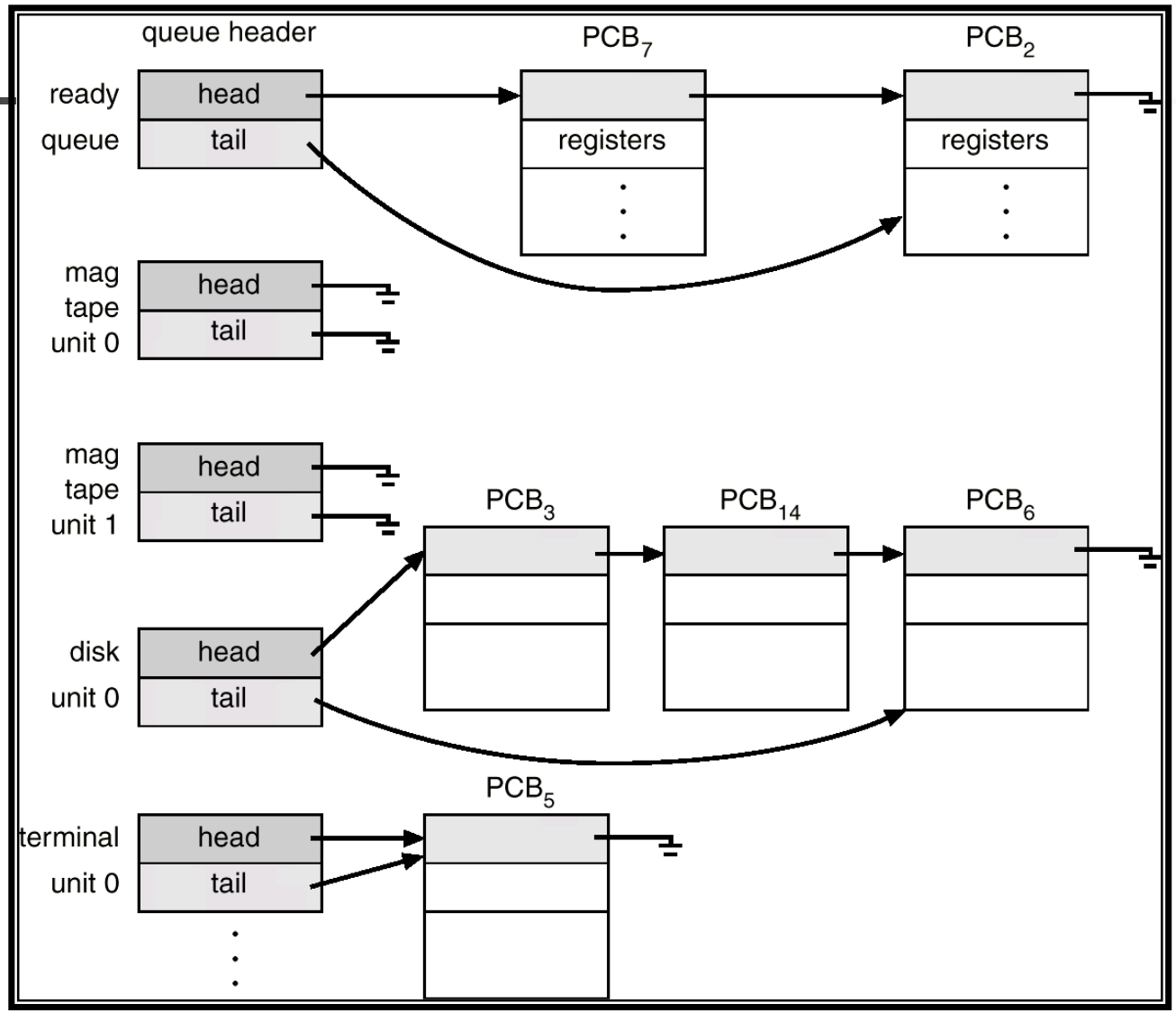


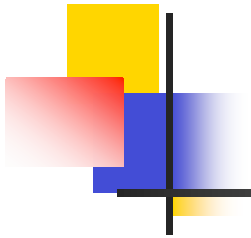


Process Scheduling Queues

- 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 between the various queues.

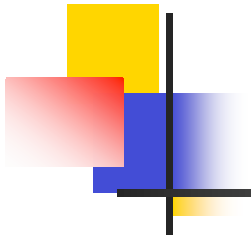
Device Queues





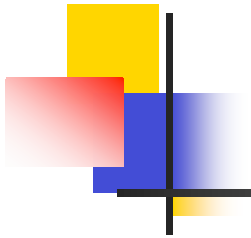
Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.



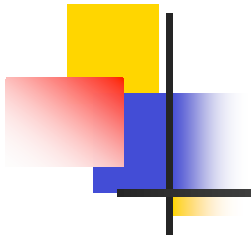
Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).
- The long-term scheduler controls the *degree of multiprogramming*.
- Processes can be described as either:
  - I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.
  - *CPU-bound process* – spends more time doing computations; few very long CPU bursts.



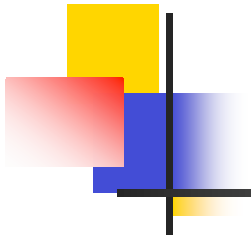
Context Switch

- 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.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support: available registers, page table setup…



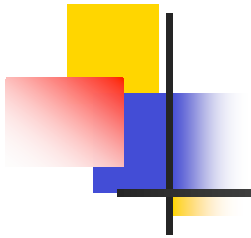
Process Creation

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



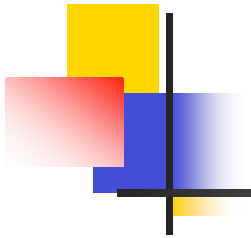
Child Address Space

- Copy of the parent's address space.
  - A different system call to load a different binary
- Independent address space, a different binary loaded in.
- First case Unix.
- Second case DEC VMS.
- Windows supports both paradigms.



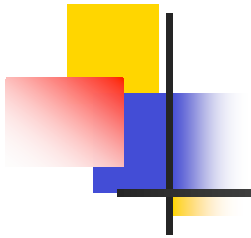
fork and exec

- Usually exec is invoked after a fork (since exec does not return)
- In this case fork followed by exec is not very efficient (why?)
- Use copy-on-write.



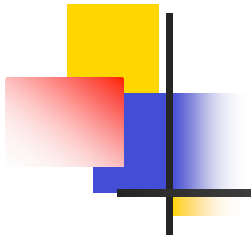
Process Creation (Cont.)

- 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.



Cooperating Processes

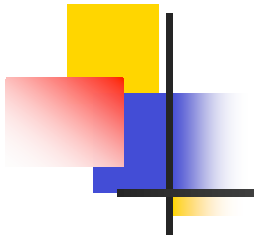
- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience



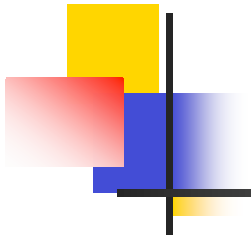
Shared Memory

- Two or more processes share a common memory area.
- The processes can exchange information using the shared memory.
- Synchronization issues: one process is writing and another is read from the same memory area.
- On solution is to gain exclusive access.
- Multiple reads, single write.

Interprocess Communication (IPC)



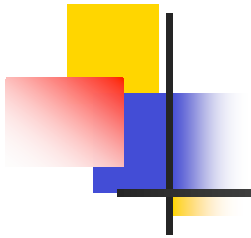
- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)



Procedure Call

- Part of a program are instructions stored in memory and executed sequentially.
- A procedure is a collection of related instruction grouped together. When a procedure is called, the CPU “jumps” to the address of the first instruction in the procedure.
- Thus to use procedure calls one needs to know the address of a procedure in memory

- Assigning memory addresses to instructions and procedures is a responsibility shared by the compiler, linker and loader.
- If the program is self contained the compiler can assign addresses to all instructions and procedures relative to the address of the first instruction of the program.
- This way when a program is loaded into memory all addresses are adjusted by a fixed offset.



Linking

- When a program is **not self contained**, it needs to call procedures from other programs or libraries.
- If the modules are compiled **separately** the addresses have to be readjusted. This step is done by the linker.
- The linking step can be done statically, before the program is executed, or dynamically, at run time.
- Linking at runtime is performed by a loader.