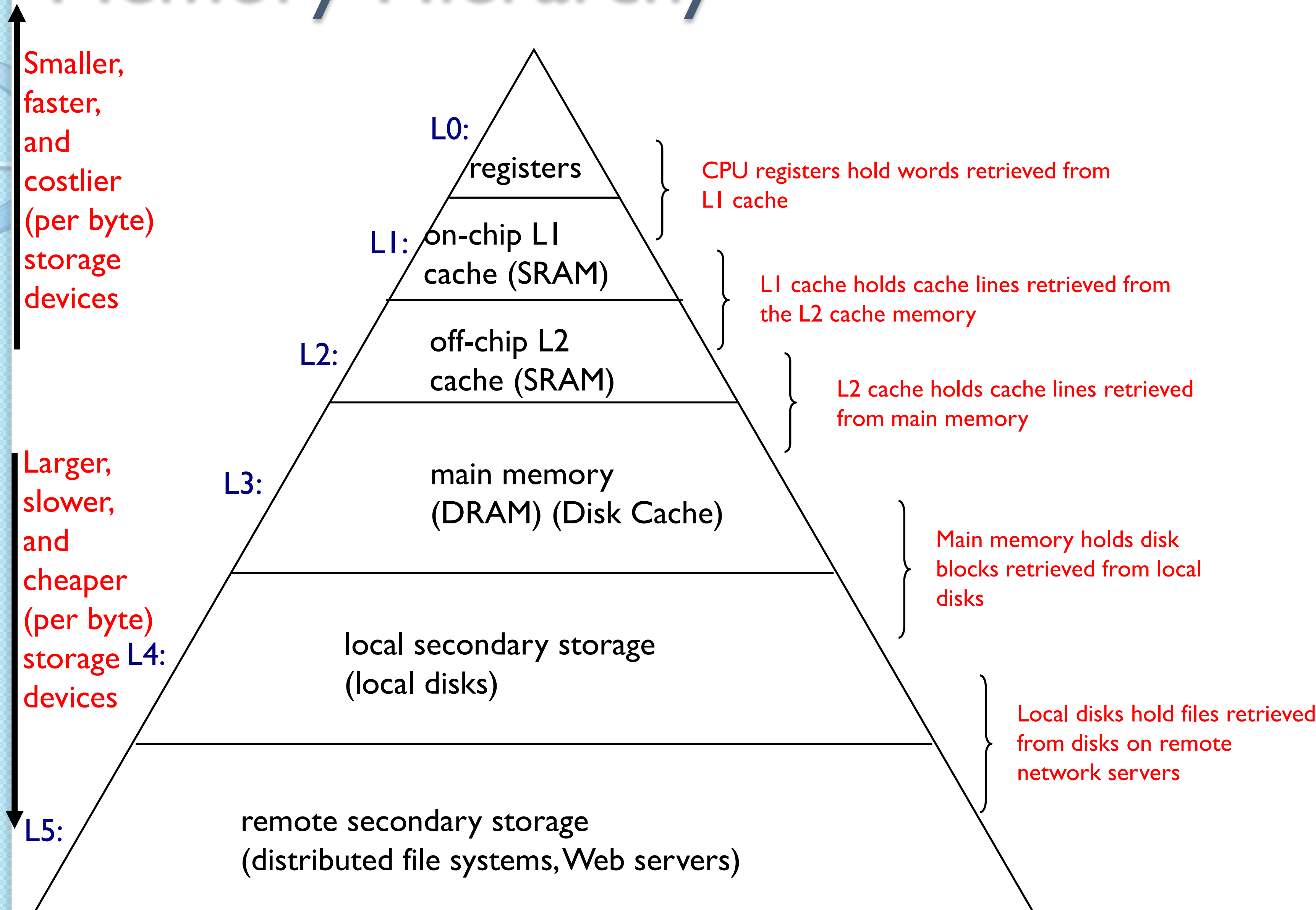
PreCAT-Operating System Day 2

PPT's Compiled by :
Mrs. Akshita. S. Chanchlani
SunBeam Infotech
akshita.chanchlani@sunbeaminfo.com

Memory Hierarchy



Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

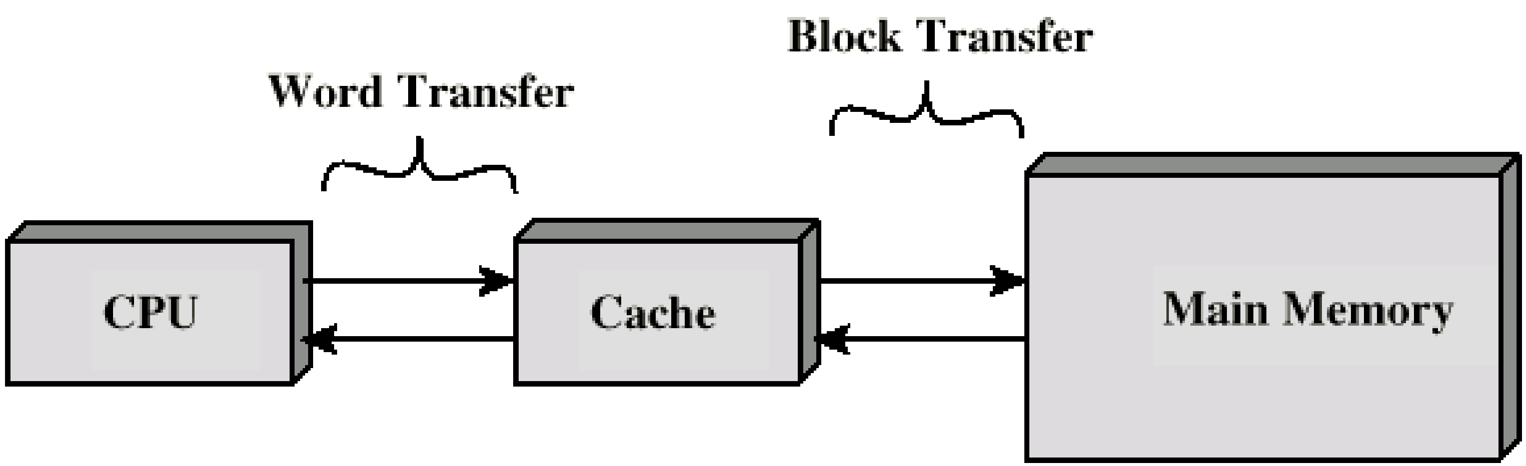
- As one goes down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor – locality of reference

Memory Access Method

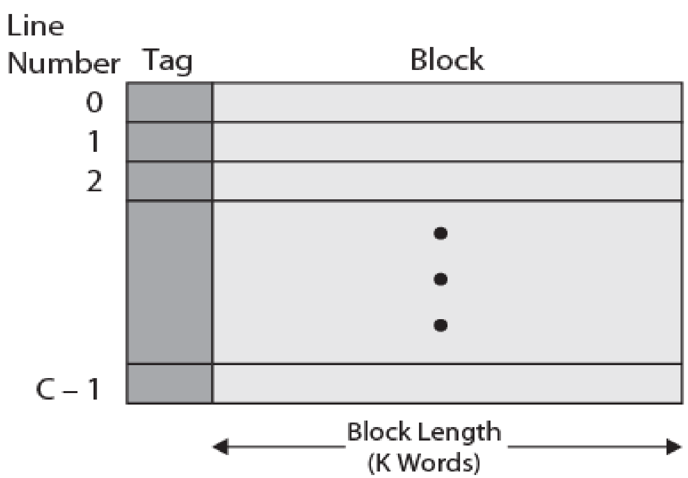
Sequential	Direct	Random	Associative
• Start at the beginning and read through in order • Access time depends on location of data and previous location • e.g. tape	• Individual blocks have unique address • Access is by jumping to vicinity plus sequential search • Access time depends on location and previous location • e.g. disk	• Individual addresses identify locations exactly • Access time is independent of location or previous access • e.g. RAM	• Data is located by a comparison with contents of a portion of the store • Access time is independent of location or previous access • e.g. cache

Cache & Main Memory

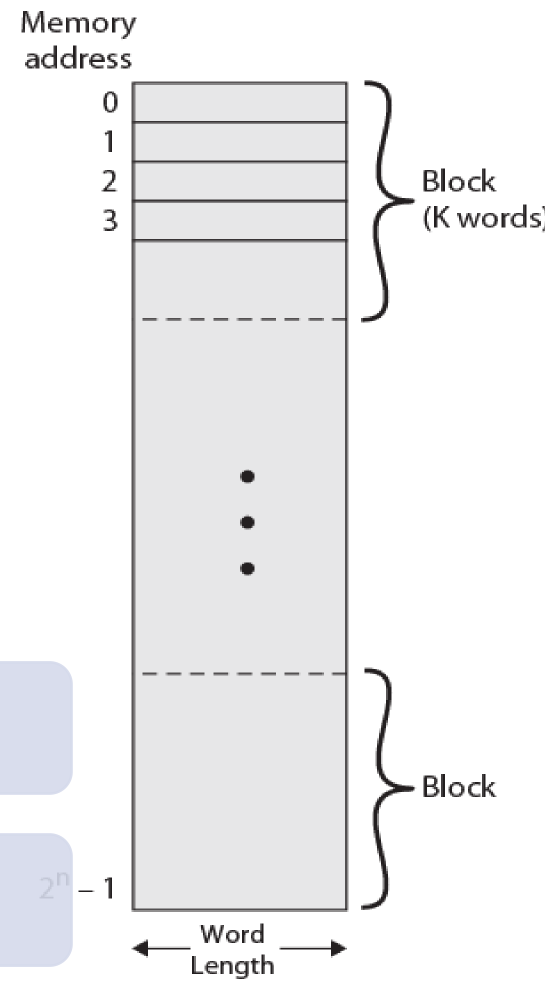
- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module
  - An entire blocks of data is copied from memory to the cache because the principle of locality tells us that once a byte is accessed, it is likely that a nearby data element will be needed soon.



Cache/Main Memory Structure



(a) Cache



(b) Main memory

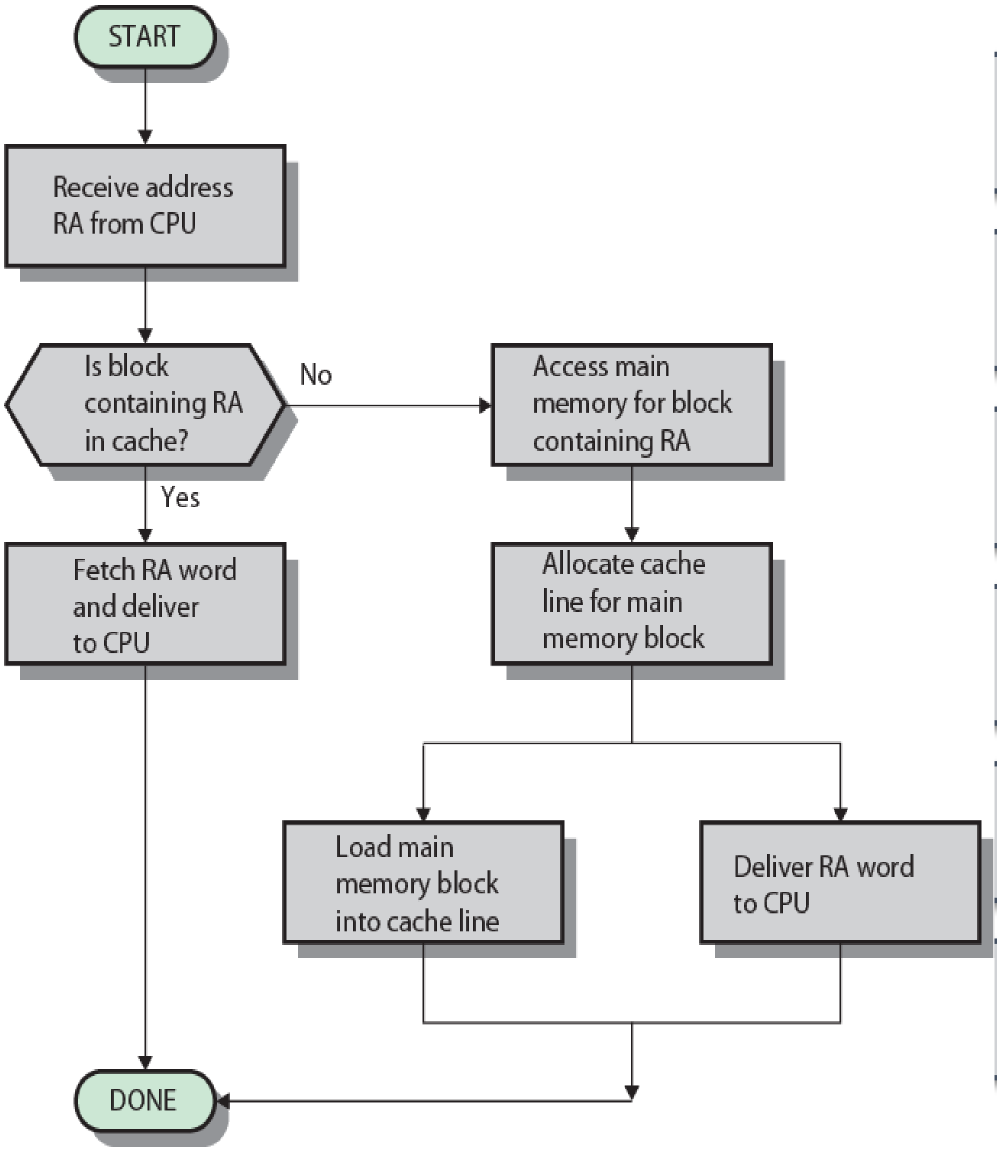
A Hit

- is when data is found at a given memory level.

A *miss*

- is when it is not found.

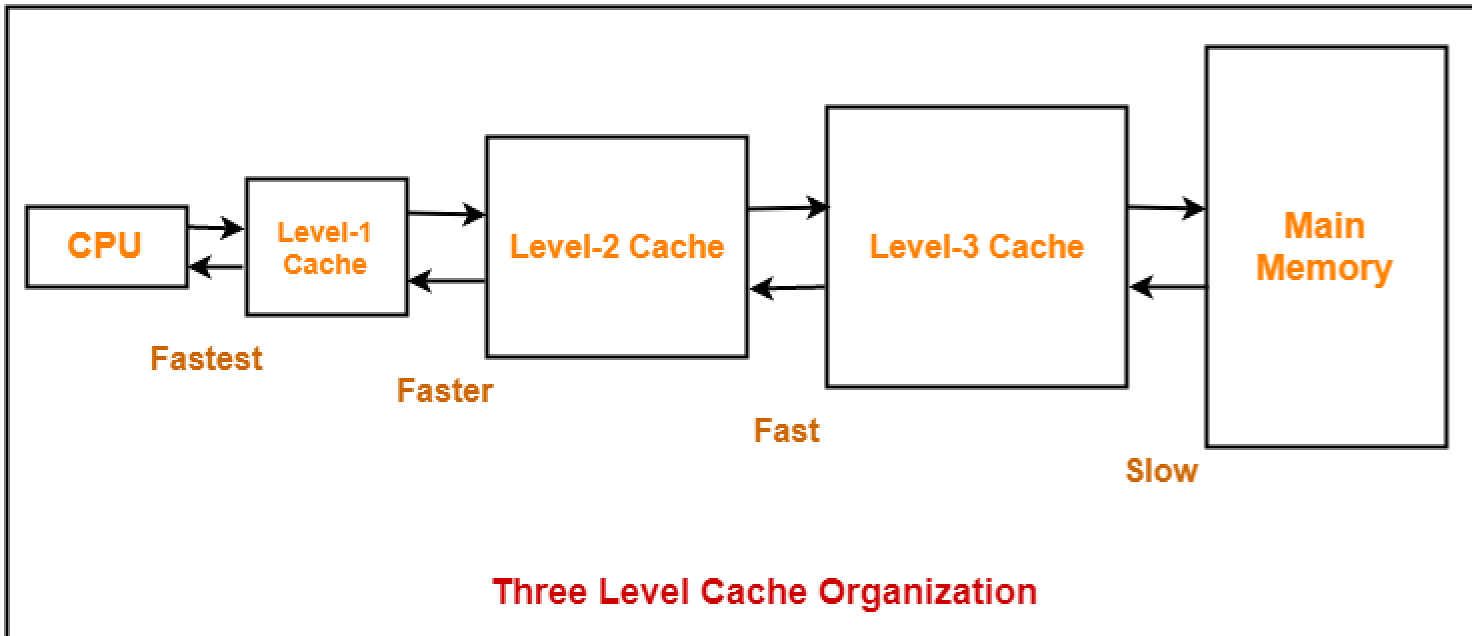
Cache Operations



- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Multilevel Cache Organization

- A multilevel cache organization is an organization where cache memories of different sizes are organized at multiple levels to increase the processing speed to a greater extent.
- The smaller the size of cache, the faster its speed.
- The smallest size cache memory is placed closest to the CPU.
- This helps to achieve better performance in terms of speed.



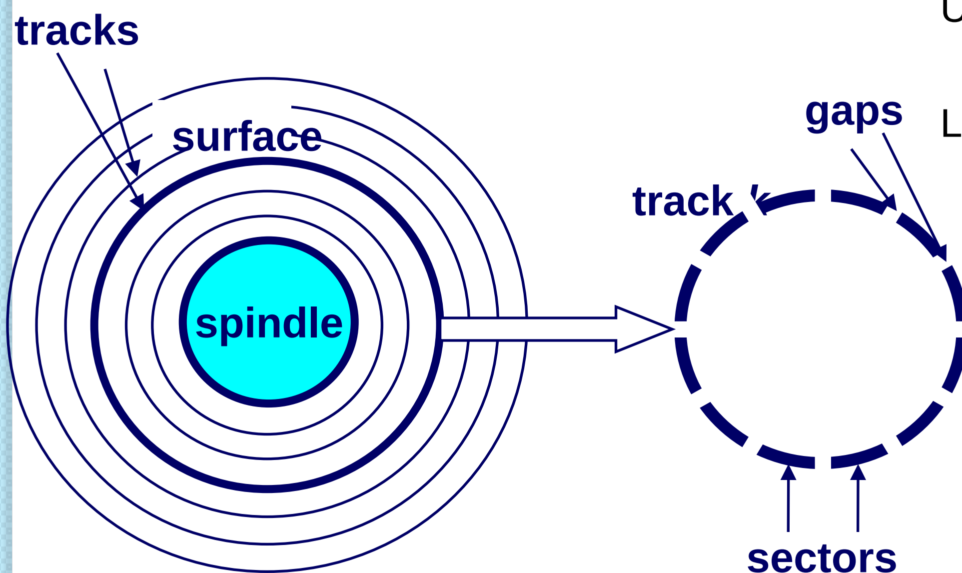
Random-Access Memory (RAM)

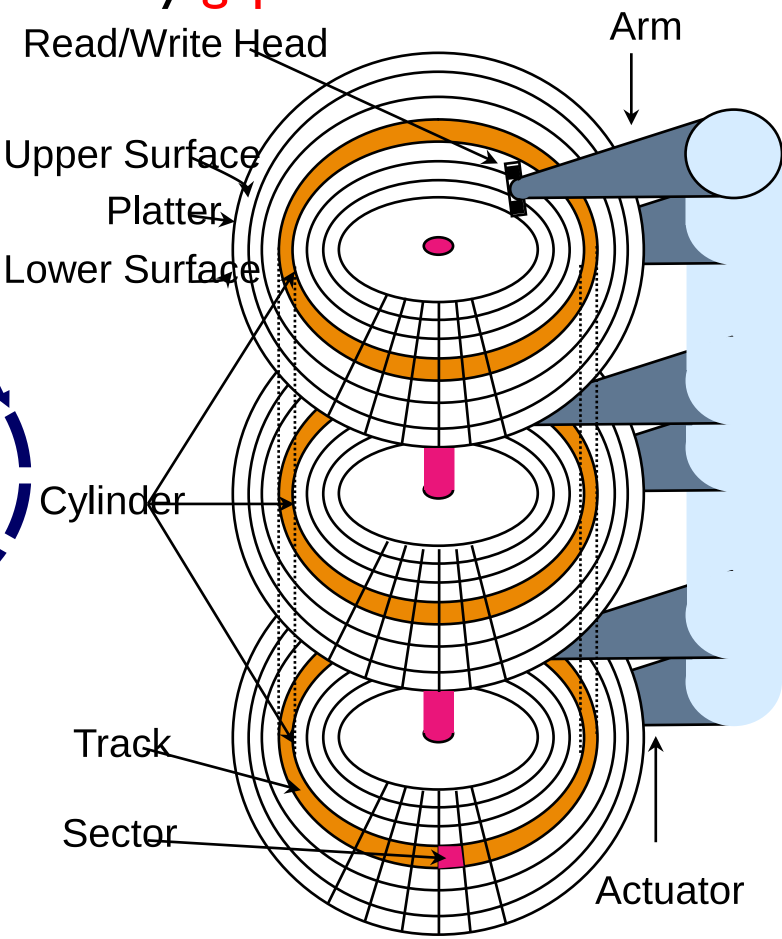
- Key features
  - RAM is packaged as a chip, Basic storage unit is a cell (one bit per cell)
  - Its internal memory of the CPU for storing data, program, and program result
  - Used for Read/ Write
  - Volatile (Temporary Storage)

Static RAM (SRAM)	Dynamic RAM (DRAM)
• memory retains its contents as long as power is being supplied. • Made up of transistor • Static because it doesn't need to be refreshed • SRAM is more often used for system cache. • SRAM is faster than DRAM	• memory must be constantly refreshed or it will lose its contents. • This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second • Made up of memory cells composed of capacitors and one transistor. • DRAM is typically used for the main memory in computing devices

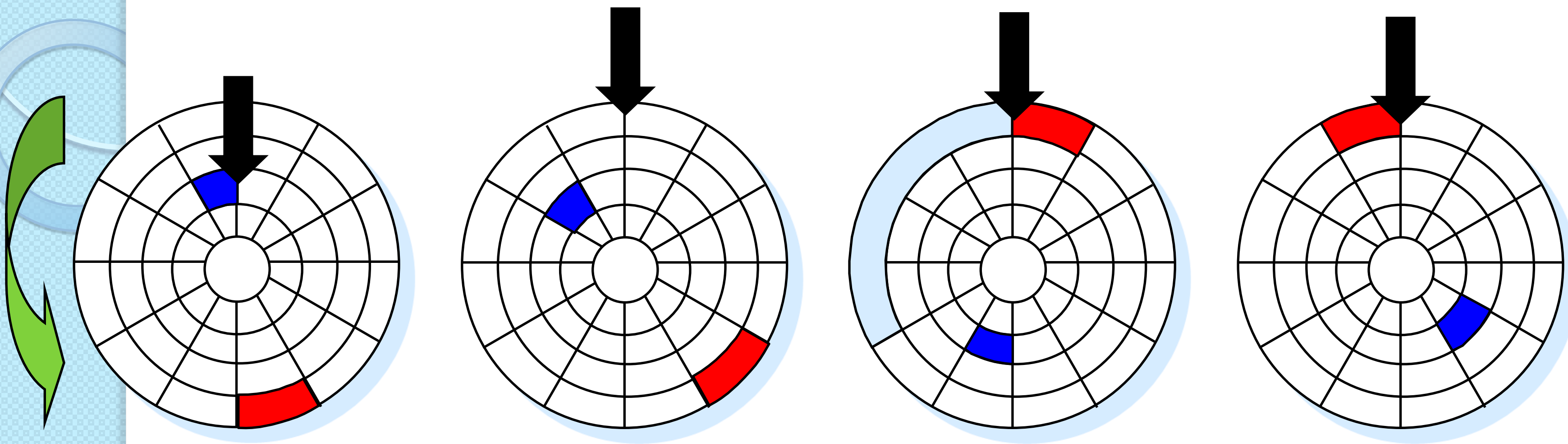
Disk Structure

- Disks contain platters, each with two surfaces
- Each surface organized in concentric rings called tracks
- Each track consists of sectors separated by gaps





Disk Access – Service Time Components



After BLUE read

Seek for RED

Rotational latency

After RED read

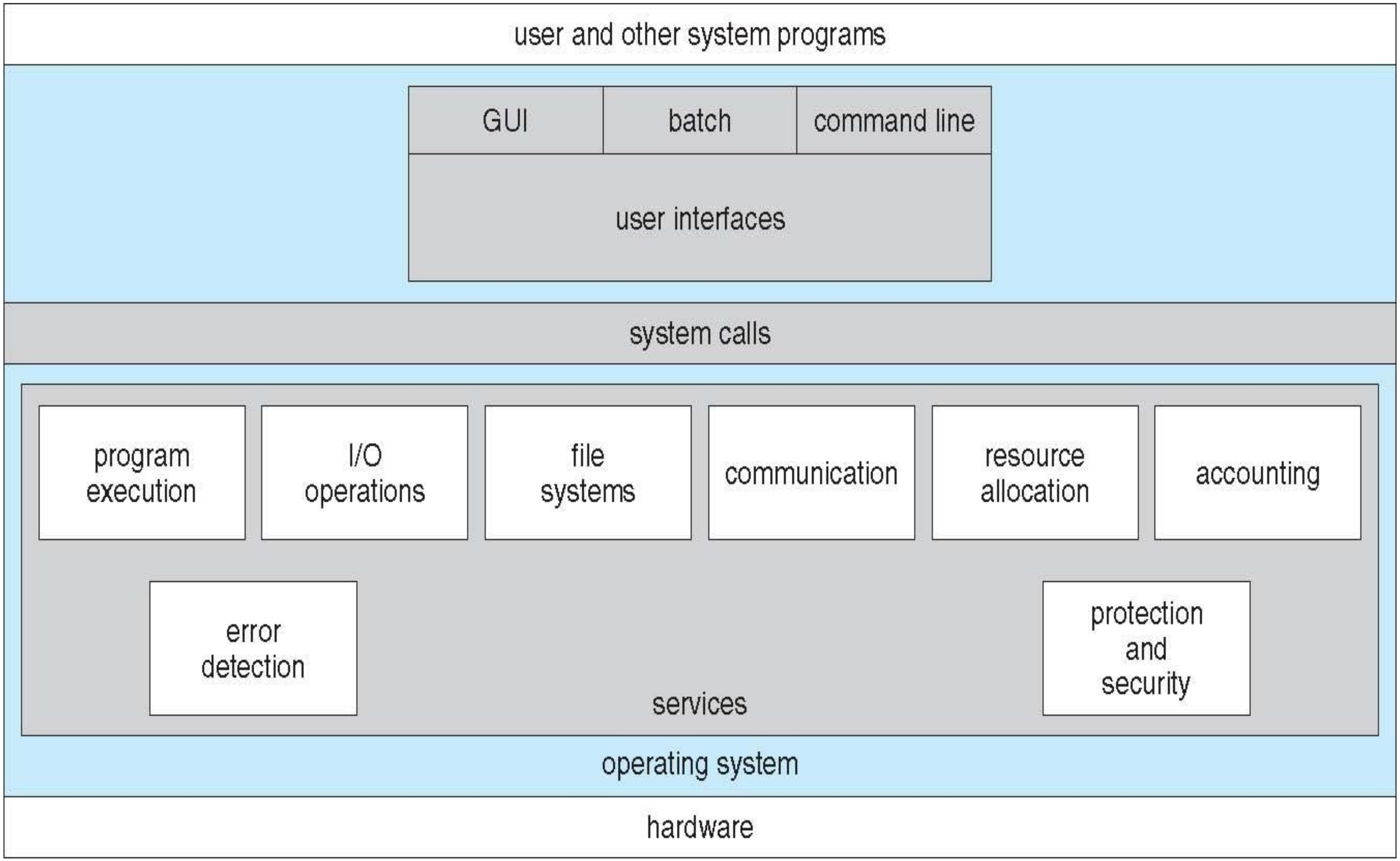
Seek
Rotational Latency
Data Transfer

Seek time	• Time to position heads over cylinder containing target sector.
Rotational latency	• Time waiting for first bit of target sector to pass under r/w head
Transfer time	• Time to read the bits in the target sector

Unix

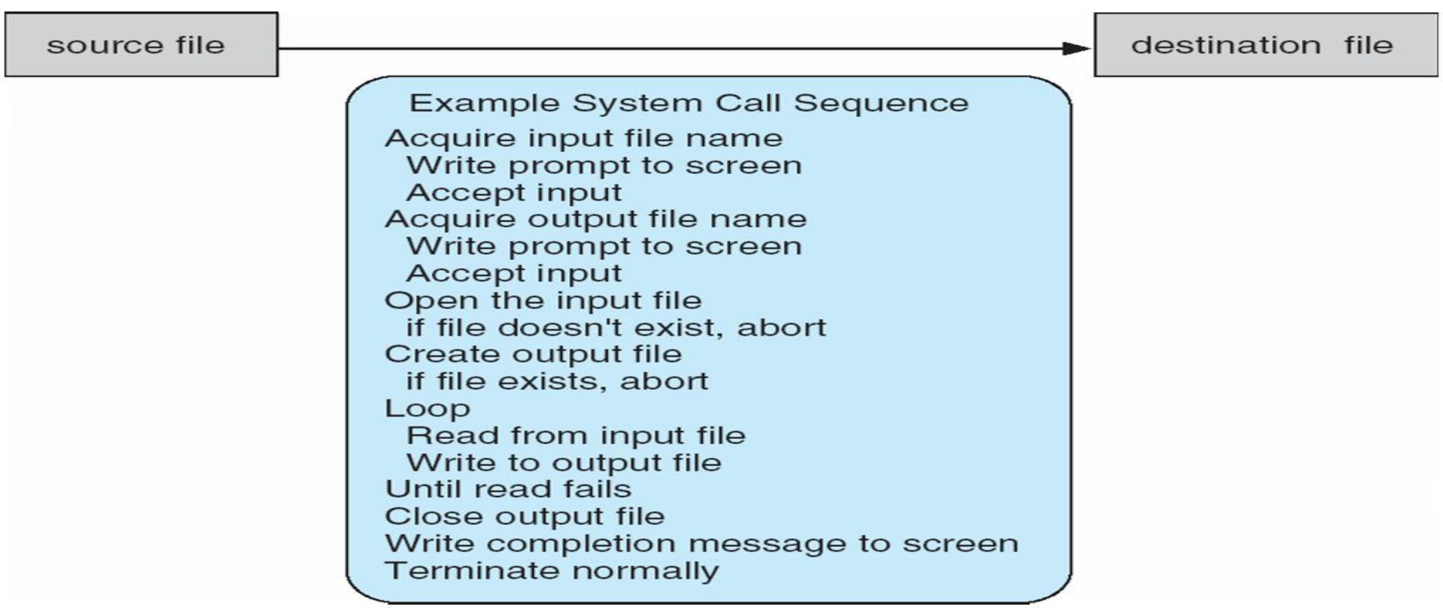
- **UNIX** was originally spelled “Unics”.
- UNICS stands for UNiplexed Information and Computing System, is a popular operating system developed at Bell Labs in the early 1970s.
- Invented by AT&T Bell Labs in late 60’s
- Currently there are different versions and variants of UNIX such as SunOS, Linux, Solaris, BSD…
- An operating system run on many servers/workstations

View of Operating System Services

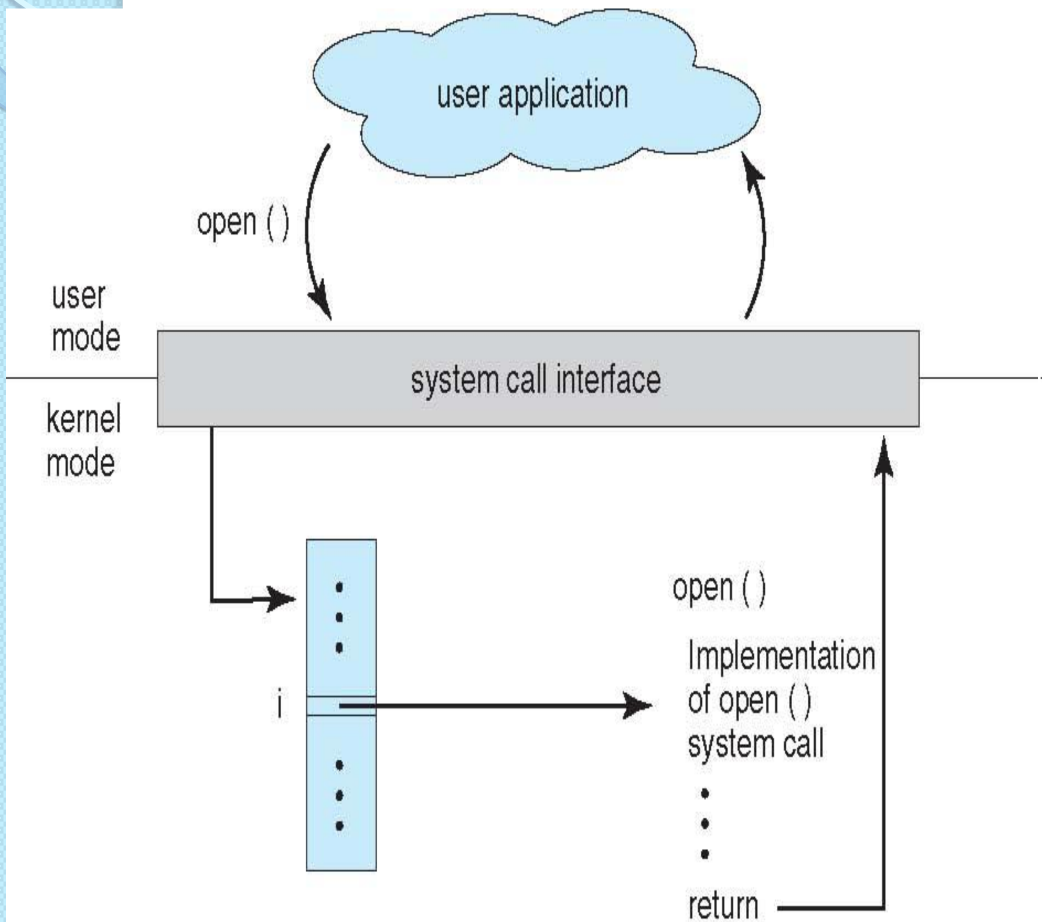


System Calls

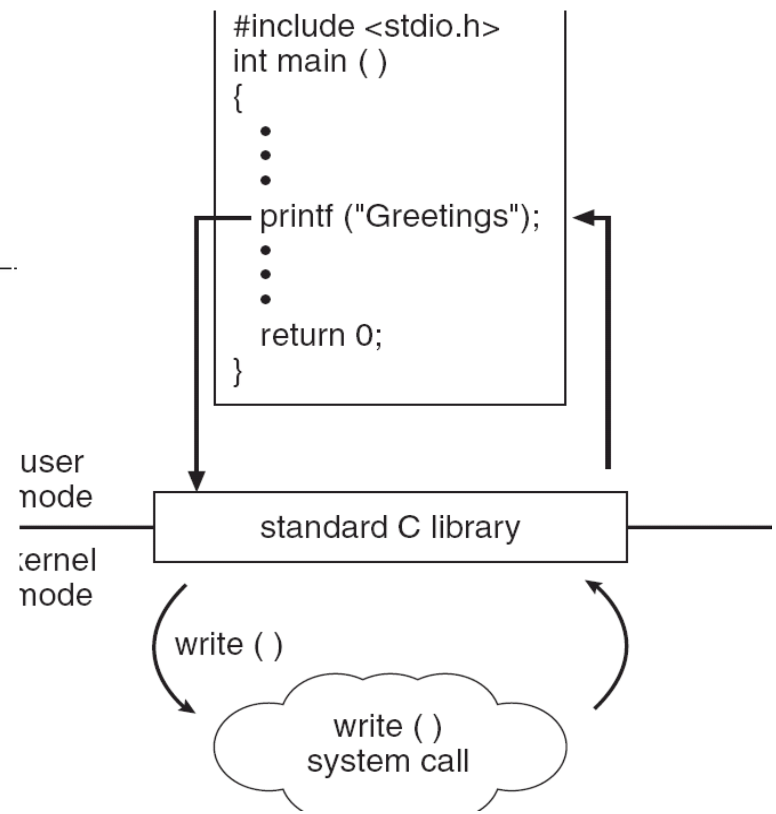
- Programming interface to the services provided by the OS
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use.
- The system call interface invokes intended system call in OS kernel and returns status of the system call
- E.g. Copying a file from source to destination



API – System Call – OS Relationship



C Example



System Calls Categories

Process control
(fork(),exit(),wait()

- load, execute, end, abort
- create process, terminate process
- get and set process
- allocate and free memory

File management
Read(),write()

- create file, delete file, open, close file
- read, write

Device management
Read(),write()

- request device, release device
- read, write
- get device attributes, set device attributes

Communications
Pipe(),shmget()

- send, receive messages
- transfer status information

Protection and Security
Chmod(),chown()

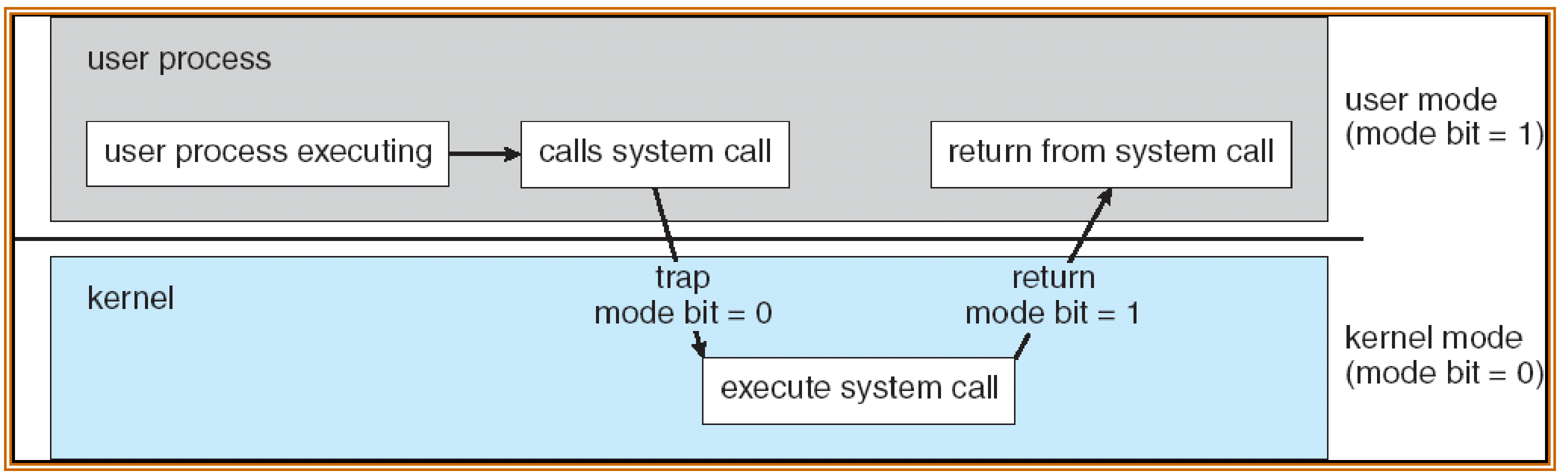
- Grant permissions
- Change ownership

Information maintenance
Getpid(),sleep()

- get time or date, set time or date
- get system data, set system data

Dual Mode Operation

- Allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
- To perform privileged operations, must transit into OS through well defined interfaces
  - System calls
  - Interrupt handlers



CPU Modes

System Mode/ Kernel Mode	User Mode

System Mode/ Kernel Mode

- privileged mode/master mode/supervisor mode
  - Can execute any instruction
  - Can access any memory locations, e.g., accessing hardware devices,
  - Can enable and disable interrupts
  - Can change privileged processor state
  - Can access memory management units
  - Can modify registers for various descriptor tables

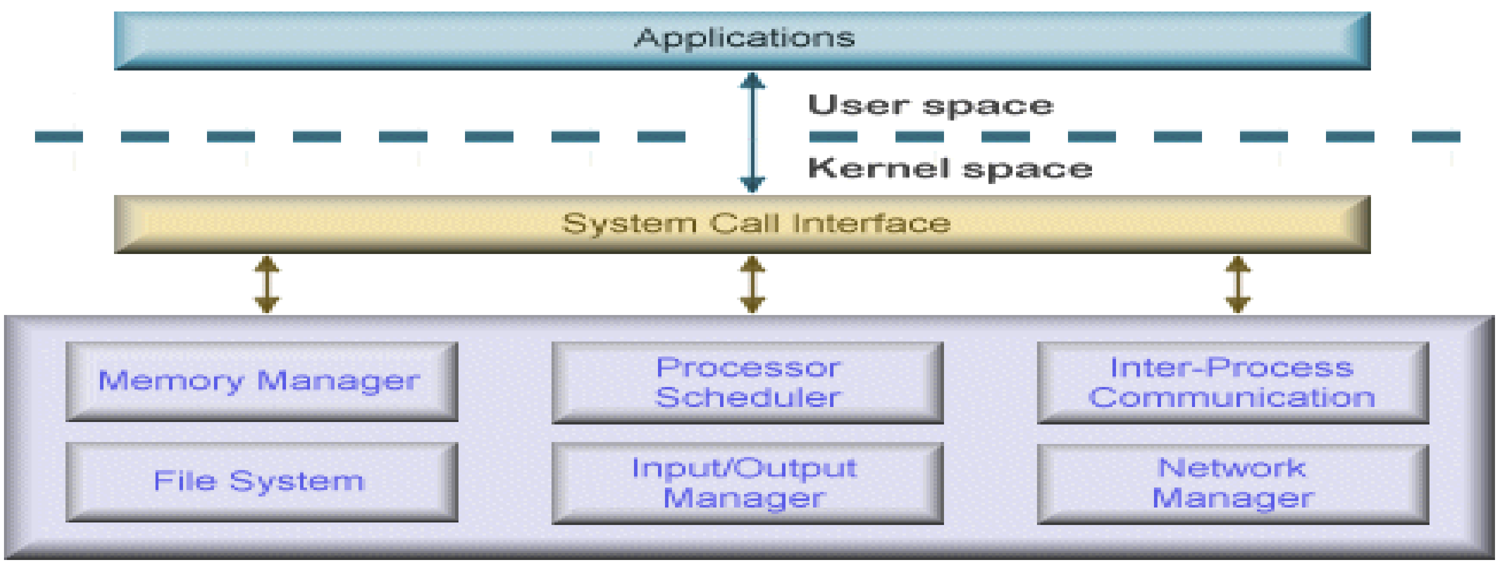
User Mode

- Unprivileged Mode
  - Access to memory is limited,
  - Cannot execute some instructions
  - Cannot disable interrupts,
  - Cannot change arbitrary processor state,
  - Cannot access memory management units

Transition from user mode to system mode must be done through well defined call gates (system calls)

Kernel space vs User space

- Part of the OS runs in the kernel model
  - known as the OS kernel
- Other parts of the OS run in the user mode, including service programs , user applications, etc.
  - they run as processes
  - they form the user space (or the user land)



User Mode VS Kernel Mode

User Mode vs Kernel Mode	
User Mode is a restricted mode, which the application programs are executing .	Kernel Mode is the privileged mode, which the computer enters when accessing hardware resources.
Modes	
User Mode is considered as the slave mode or the restricted mode.	Kernel mode is the system mode, master mode or the privileged mode.
Address Space	
In User mode, a process gets their own address space.	In Kernel Mode, processes get single address space.
Interruptions	
In User Mode, if an interrupt occurs, only one process fails.	In Kernel Mode, if an interrupt occurs, the whole operating system might fail.
Restrictions	
In user mode, there are restrictions to access kernel programs. Cannot access them directly.	In kernel mode, both user programs and kernel programs can be accessed.

Process and Program

- A **process** is an instance of a program in execution.
- Running program is also knows as **Process.**
- When a program gets loaded in to memory is also known as **Process.**
- A **Program** is a set of instructions given to the machine to do specific task.
  - Three types of Programs:
    - User Programs (c/java program)
    - Application Programs (ms office)
    - System Programs (device drivers, interrupt handlers etc)

Process Management

- Operating systems provide fundamental services to processes including:
  - Creating processes
  - Destroying processes
  - Suspending processes
  - Resuming processes
  - Changing a process's priority
  - Blocking processes
  - Dispatching processes
  - Inter Process communication (IPC)

Memory Layout of Program and Process

Program Consist of
exe header/primary header
Block started by symbol (bss) section (un initialized static / global variables)
Data section (initialized static / global variables)
Rodata Section(Constant/literals)
code/text section (contains executable instructions)
Symbol Table

Process Consist of
Skipped
Block started by symbol (bss) section (un initialized static / global variables)
Data section (initialized static / global variables)
Rodata Section(Constant/literals)
code/text section (contains executable instructions)
Skipped
Stack Section
Heap Section

Process Control Block/ Process Descriptors

- When an execution of any program is started one structure gets created for that program/process to store info about it, for controlling its execution, such a structure is known as PCB: Process Control Block.
- It has information about process like:
  - process id – pid
  - process state - current state of the process
  - memory management info
  - CPU scheduling info
  - Program Counter -- address of next instruction to be executed
  - Exit status
  - Execution Context
  - I/O devices info etc...

A Two State Process Model

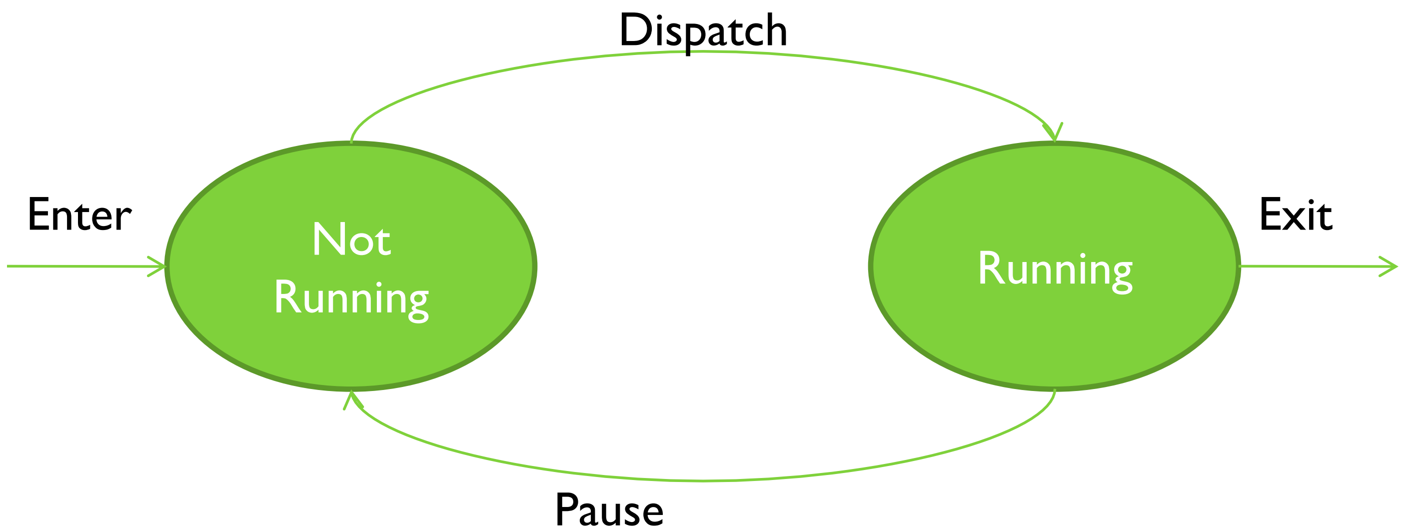


Fig. Two State Process Diagram

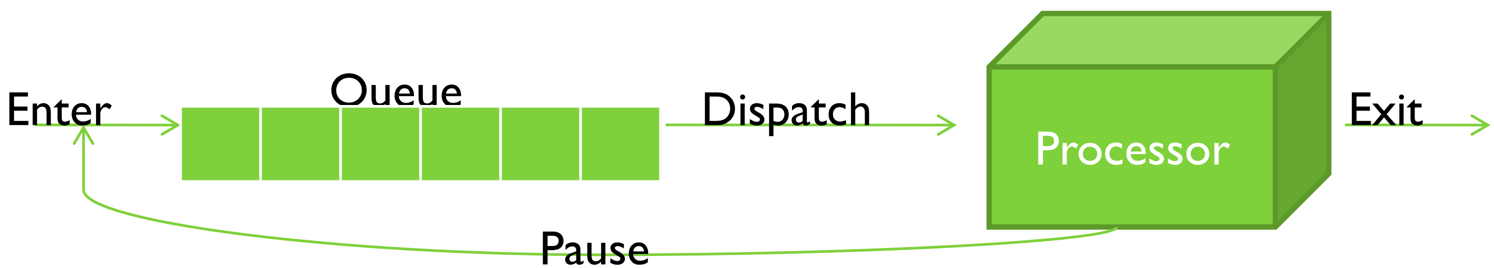
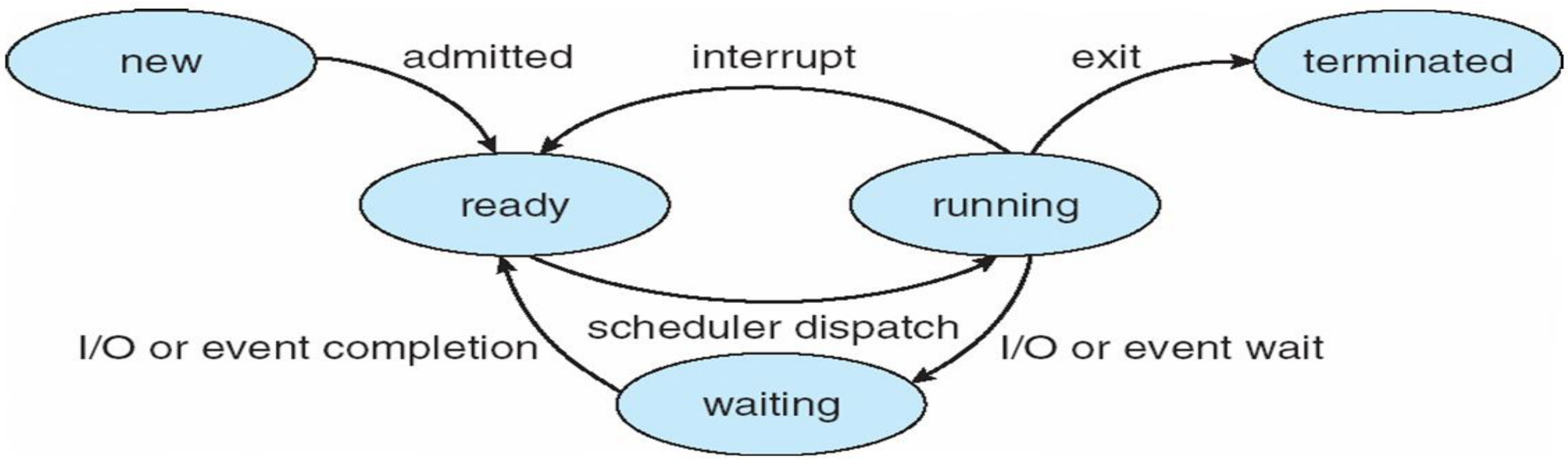


Fig. Queuing Diagram

Five State Process Diagram



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

Data Structures Maintained by Kernel at the time of process Execution

- Job Queue
- Ready Queue
- Waiting Queue

Process May be in one of the state at a time

New
- when program execution is started or upon process submission process
- when a PCB of any process is in a job queue then state of the process is referred as a new state.

Ready
- When a program is in a main memory and waiting for the cpu
- when a PCB of any process is in a ready queue then state of the process is referred as a ready state.

Running
- When a CPU is executing a process

Exit
- when a process is terminated

Waiting
- when a process is requesting for any i/o device then process change its change from running to waiting state
- When a PCB of any process is in a waiting queue of any device

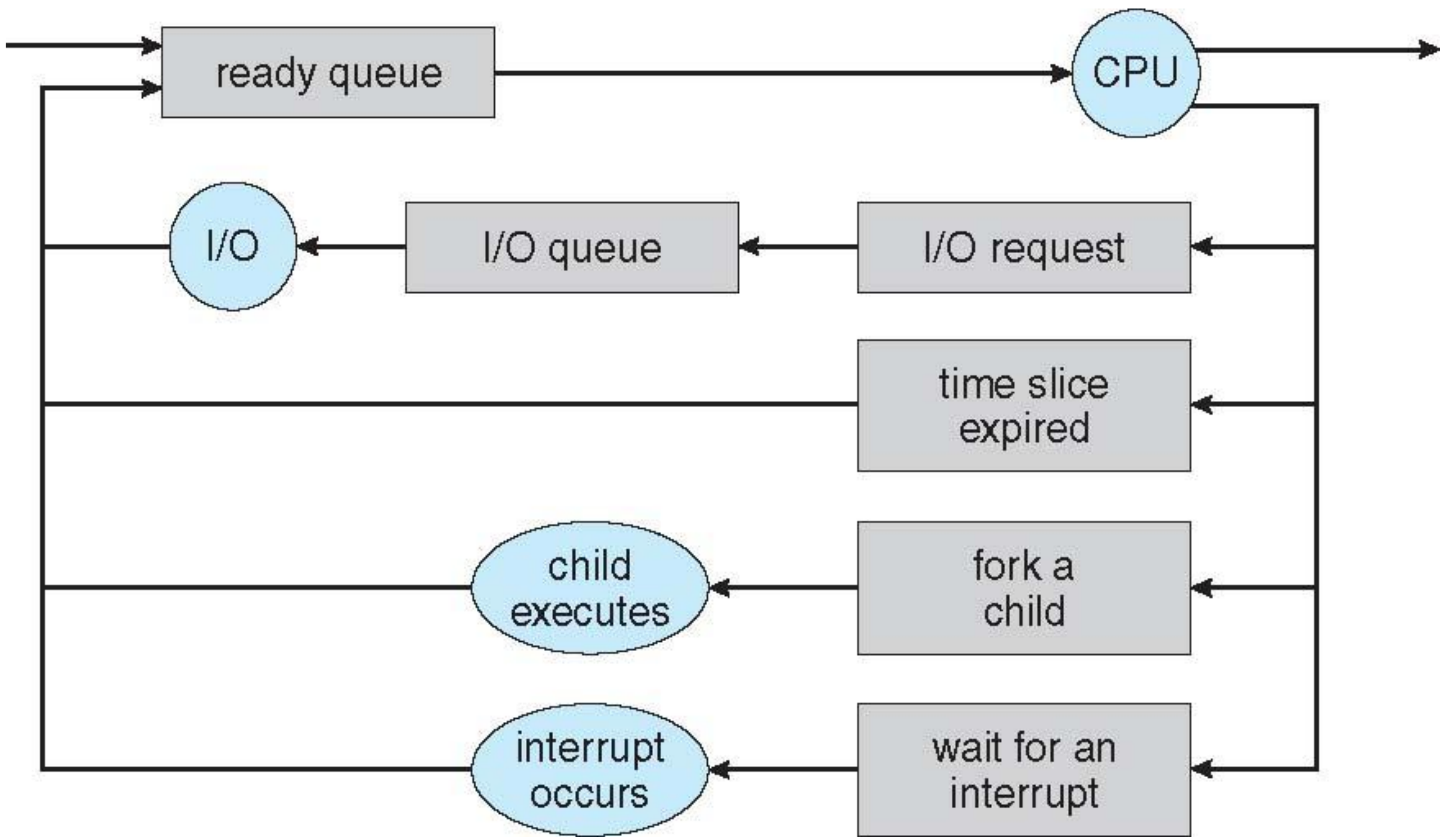
Schedulers

- Job Scheduler/long term schedulers :
  - Selects which processes should be brought into the ready queue
- CPU Scheduler/Short term schedulers
  - Process from ready queue to load into CPU
  - selects which process should be executed next and allocates CPU

Dispatcher

- Gives control of the CPU to the process which is scheduled by the CPU scheduler
- time taken by the dispatcher to stops execution and one process and starts execution of another process is called as "dispatcher latency".

Representation of Process Scheduling



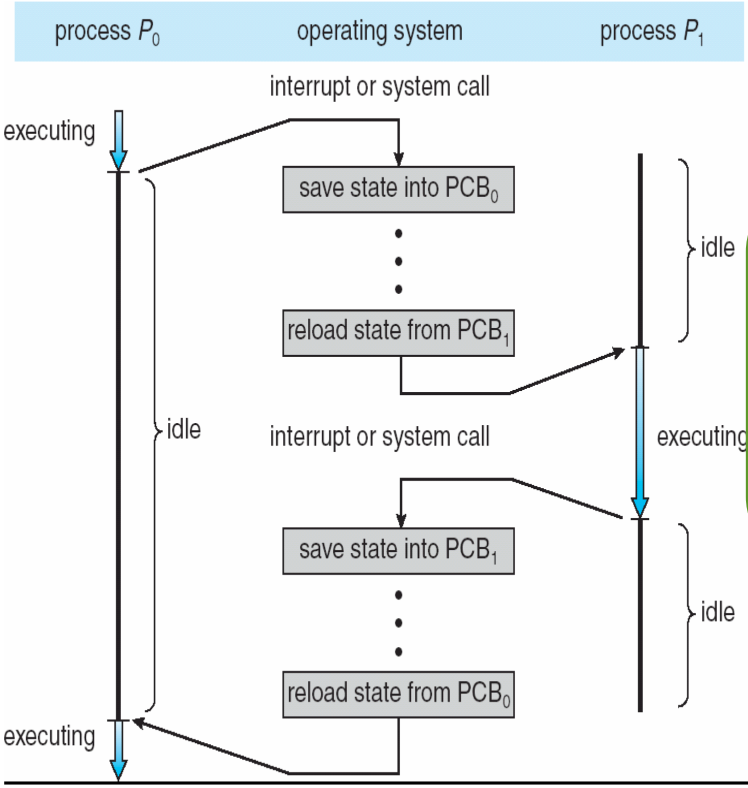
Context Switch

- 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 time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch.

Context Switch = state-save + state-restore

"state-save" -- to save execution context of suspended process into its PCB
"state-restore" -- to restore execution context of the process which is scheduled by the cpu scheduler onto the cpu registers.

Context Switching



CPU scheduler should get called in following four cases:

1.running --> terminated

2.running --> waiting

3.. running --> ready

4. waiting --> ready

Thank you