OPERATING SYSTEM (CSL3030)

Palash Das, CSE, IIT Jodhpur

TENTATIVE EVALUATION POLICY (I MAY CHANGE IN REAL-TIME)

- Continuous Evaluation – 50%
  - Lab – 25% (Assignments and In-lab Evaluations)
  - Quiz (one or two) – 10%
  - Projects / Emerging Research involvements – 15%

- Primary Evaluation - 50%
  - Minor 1 – 15%
  - Minor 2 – 15%
  - Major – 20%

SYLLABUS / BOOKS

Course Title	**Operating Systems**	Course No.	**CS 3XXX**
Department	Computer Science and Engineering	Structure (L-T-P [C])	3-0-2 [4]
Offered for	B.Tech	Type	Compulsory
Prerequisite	Data Structures and Algorithms	Antirequisite	None

Objectives

1. To learn about design principles of operating systems
2. To learn various functionalities and services of an operating system in general
3. To do a case study of Operating System

Learning Outcomes

1. Ability to understand the system performance that is driven by the Operating System services
2. Ability to modify and compile OS
3. Ability to diagnose and solve various problems in Operating Systems

Contents

Overview of Operating Systems: Types of Operating Systems, System calls and OS structure. (4 Lectures)
Processes Management: Process, Threads, CPU scheduling. (6 Lectures)
Process Coordination: Mutual exclusion, Mutex implementation, Semaphores, Monitors and condition variables, Deadlocks. (10 Lectures)
Memory Management: Swapping, Paging, Segmentation, Virtual memory, Demand paging, Page Replacement algorithms. (8 Lectures)
Storage Management: I/O devices and drivers, Disks and file Systems, File layout and directories, File system performance, File system reliability. (8 Lectures)
Protection and Security: System protection, System security. (6 Lectures)

Laboratory

Designing a shell in Linux
Multithreaded programming using pthread Solving the Sleeping-Barber problem Modification of scheduling algorithm in Linux
Solving the Producer-Consumer problem over a network Finding text, data, and stack segments of a process in Linux Implementation of page replacement algorithms
Changing file attributes in Linux
Implementing an encrypted file system in Linux
Implementing symbolic links in Linux

Text Book

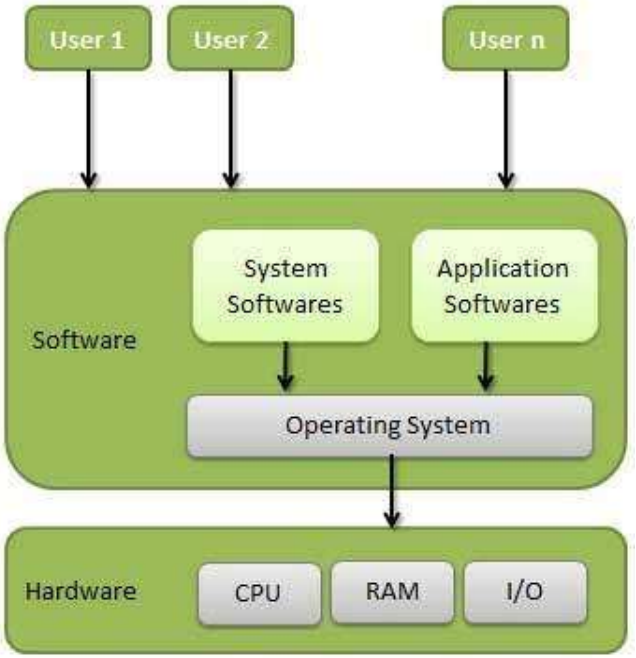
A. SILBERSCHATZ, P.B. GALVIN, G. GAGNE (2018), Operating System Concepts, John Wiley & Sons Inc., 10th Edition.

Reference Book

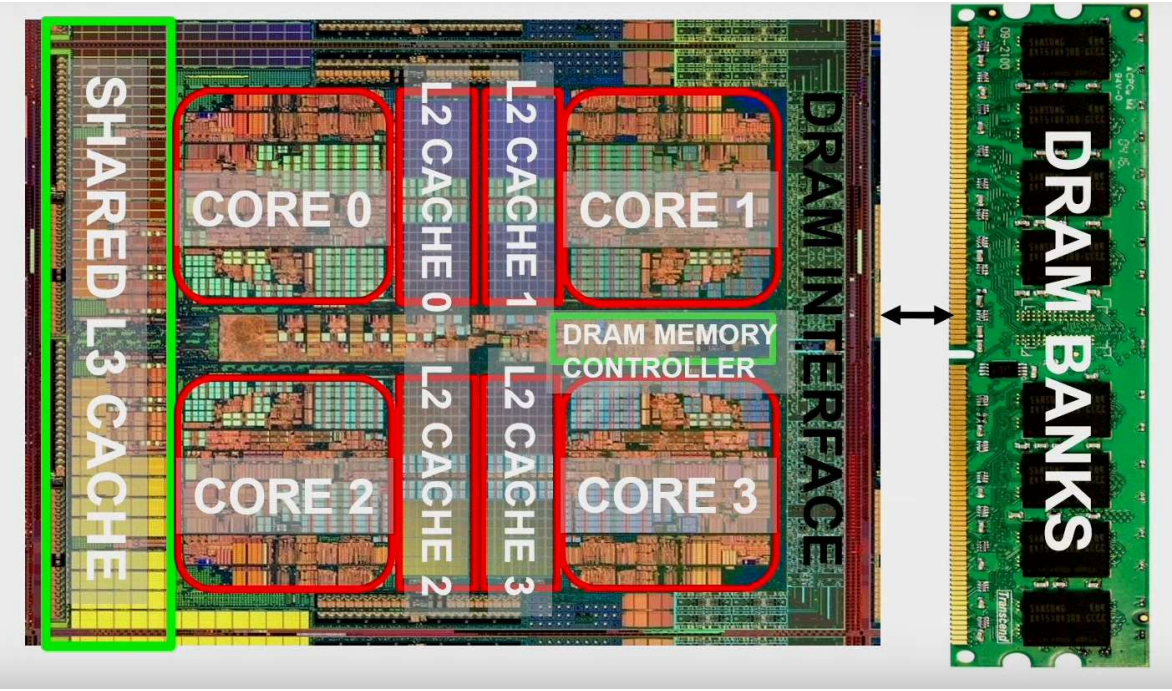
A.S. TANENBAUM, A.S. WOODHULL (2006), Operating Systems Design and Implementation, Pearson, 3rd Edition.
W. STALLINGS (2017), Operating Systems Internals and Design Principles, Pearson, 9th Edition.

COMPUTER SYSTEM STRUCTURE

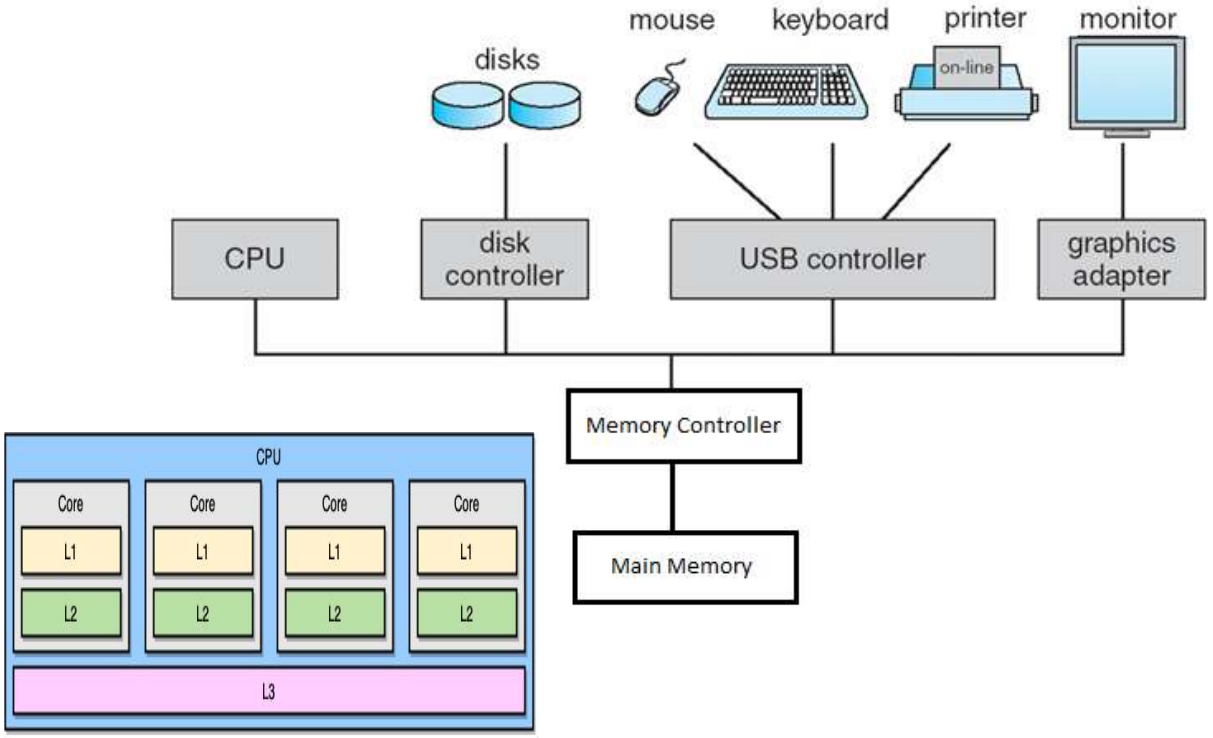
- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

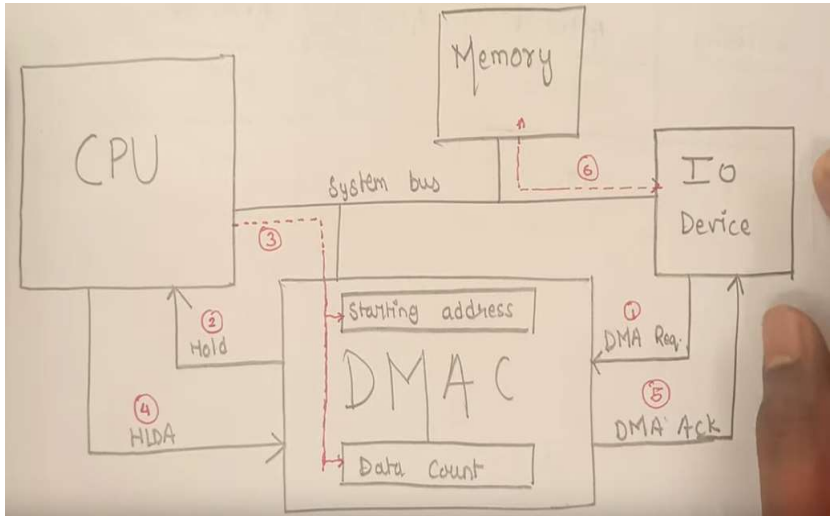


MODERN SYSTEM

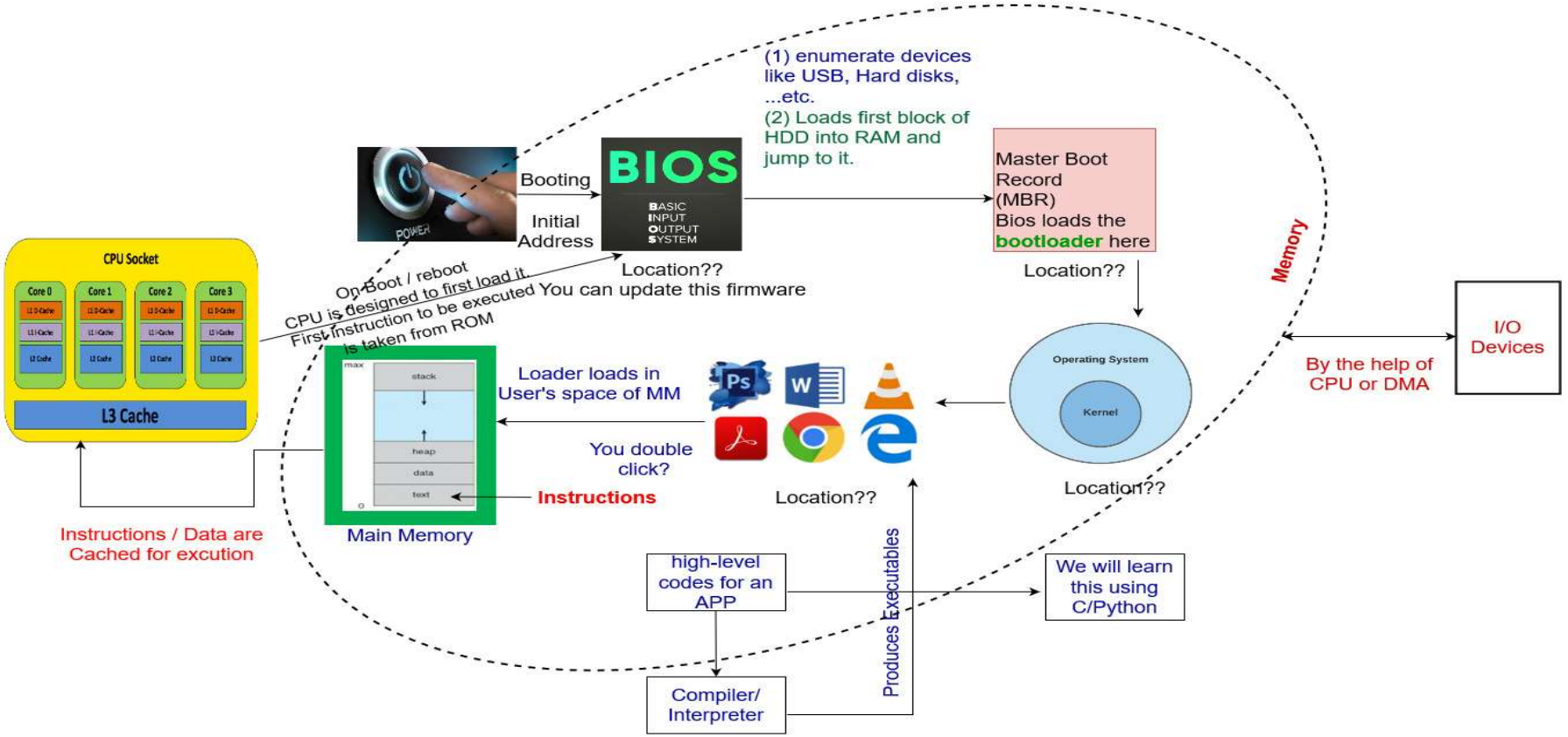


HIGH-LEVEL VIEW OF A SYSTEM





WHEN PRESS THE POWER BUTTON....



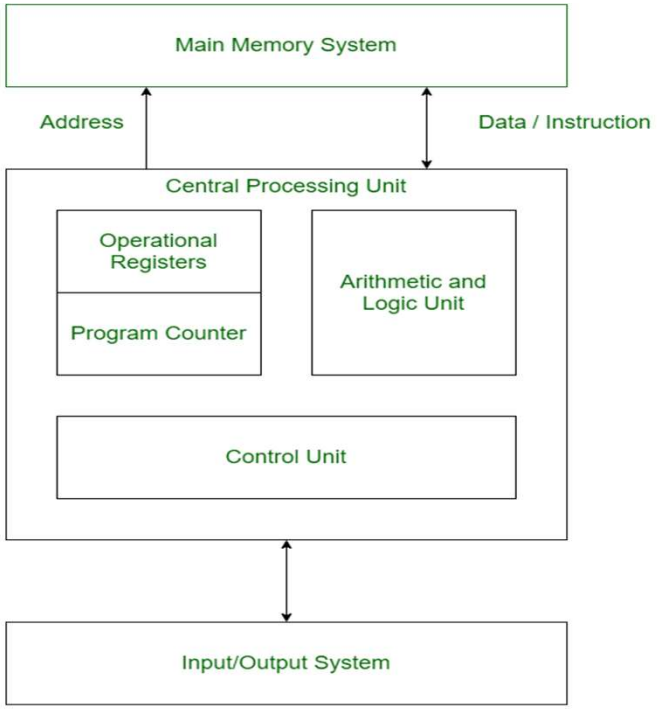
HARDWARE ASPECTS

- Computer Organization: Design of each units/components/functional block using which a system is built (low level design).
- Analogy: One who build the building with cement, bricks, etc..

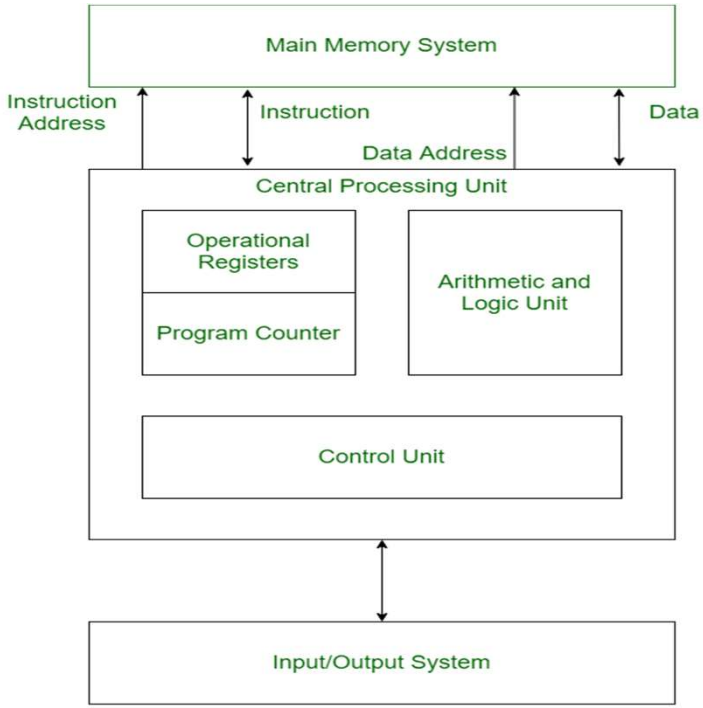
- Computer Architecture: How to integrate various components to build a computer system to achieve the best performance (high level design).

- Analogy: One who makes the plan of building.

HIGH-LEVEL VIEW OF A SYSTEM (ARCHITECTURE)



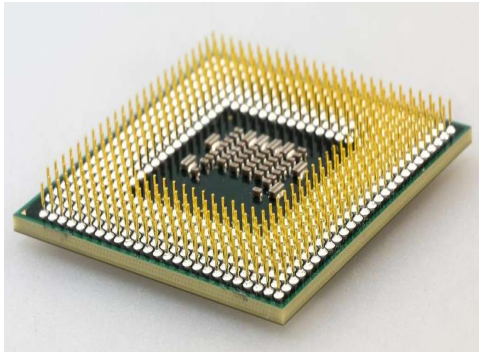
Von Neumann Architecture



Harvard Architecture

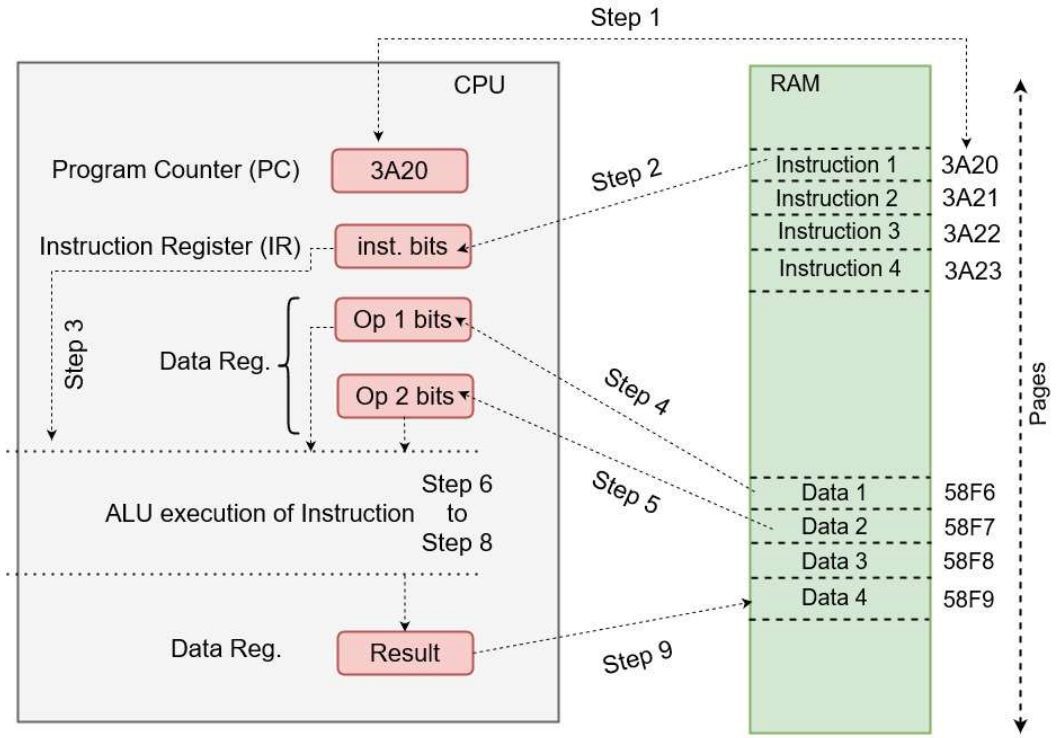
CPU/PROCESSOR

- All calculations are done by ALU (Adders, Multipliers, FPUs…).
- The Control Unit generates the sequence of control signals to carry out one operation.
- Processor fetches instructions/data from memory for execution.
- What is an instruction?
- A program refers to set of instructions.



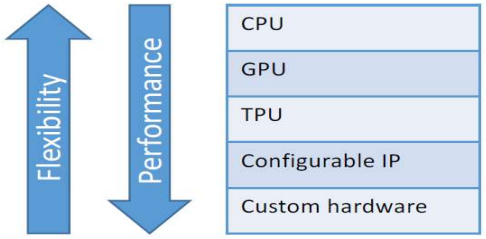
- Any other types of processing units?

A HIGH-LEVEL VIEW OF PROGRAM EXECUTION



PROCESSING OPTIONS IN THE PRESENT ERA

NMP Options	Area efficiency	Power Efficiency	Flexibility
Programmable Core	✗	✗	✓
GPUs	✗	✗	✓
FPGA	✗	✓	✓
ASICs	✓	✓	✗



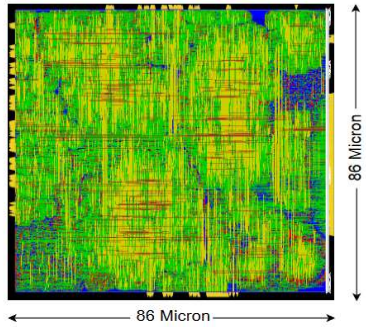


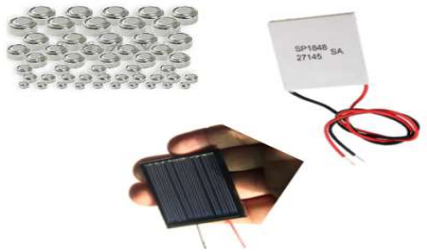
Fig. 11: ALAMNI-Chip Layout from Innovus.

If your power cable looks like one of these:

Use CPUs and GPUs

(a)

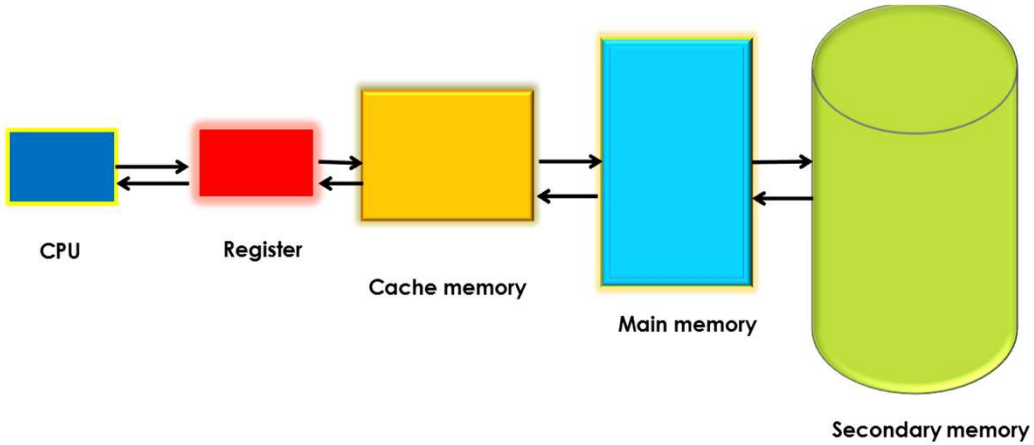
If your power supply looks like these:



Need to look for efficient inferencing alternatives

(b)

MEMORY HIERARCHY (CONT....)



Level	Typical Access Time	Typical Capacity	Other Features
Register	300-500 ps	500-1000 B	On-chip
Level-1 cache	1-2 ns	16-64 KB	On-chip
Level-2 cache	5-20 ns	256 KB – 2 MB	On-chip
Level-3 cache	20-50 ns	1-32 MB	On or off chip
Main memory	50-100 ns	1-16 GB	
Magnetic disk	5-50 ms	100 GB – 16 TB	

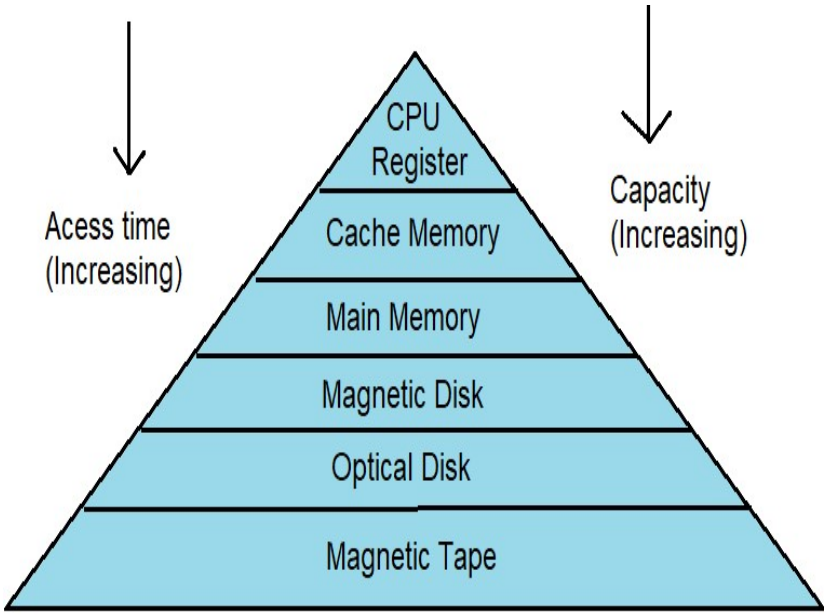


Fig:- Memory Hierarchy

CACHE

- A cache is an optimization technique.
- Cache reduces off-chip MM accesses.
- Multiple levels like l1, l2, LLC.
- Unified vs Split cache.
- It works on the principle of **locality of reference**.

Locality of Reference

- Temporal Locality (loops)
- Spatial Locality (Array)

```
mul = 1;
sum = 0;
for (i=0; i<1000; i++) {
      mul = mul * a[i];
      sum = sum + b[i];
}
return mul;
```

Rule of thumb:
90% of the total Execution time of a Program is spent in only 10% of the code

- If an item is referenced in memory, it will tend to be referenced again soon (Temporal Locality).
- If an item is referenced in memory, nearby items will tend to be referenced soon (Spatial Locality).

CACHE (CONT....)

- Cache memory is logically divided into blocks or lines. Block size may vary from 8 to 256 bytes.
- When CPU wants to access a word in the memory, cache controller first checks the L1 cache (the fastest and smallest one.)
  - If it finds a block, we call it cache hit or else it is a cache miss.
  - In case of miss, the block is searched in the immediate next level.
  - For writes, thinks are little different.

WHAT DO WE WANT FROM A MEMORY?

- Old requirements
  - Zero Cost
  - Zero Latency
  - Infinite Bandwidth
  - Infinite capacity

DRAM and Memory Controller (as we know them today) are unlikely to satisfy these needs.

- And now some new requirements
  - Nonvolatility
  - Low leakage or Zero standby power consumptions

NOT ONLY PROCESSORS, MEMORY IS HIGHLY CRITICAL FOR A SYSTEM

- Of course Processors will sit idle if data is not timely supplied.
- Second is energy concerns which may be at this point we ignore.

Main memory energy/power is a key system design concern

- ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer'03] >40% power in DRAM [Ware, HPCA'10][Paul,ISCA'15
- DRAM consumes power even when not used (periodic refresh)

every 64 ms at normal temperature (<85 °C) and 32 ms at high temperature (>85 °C)

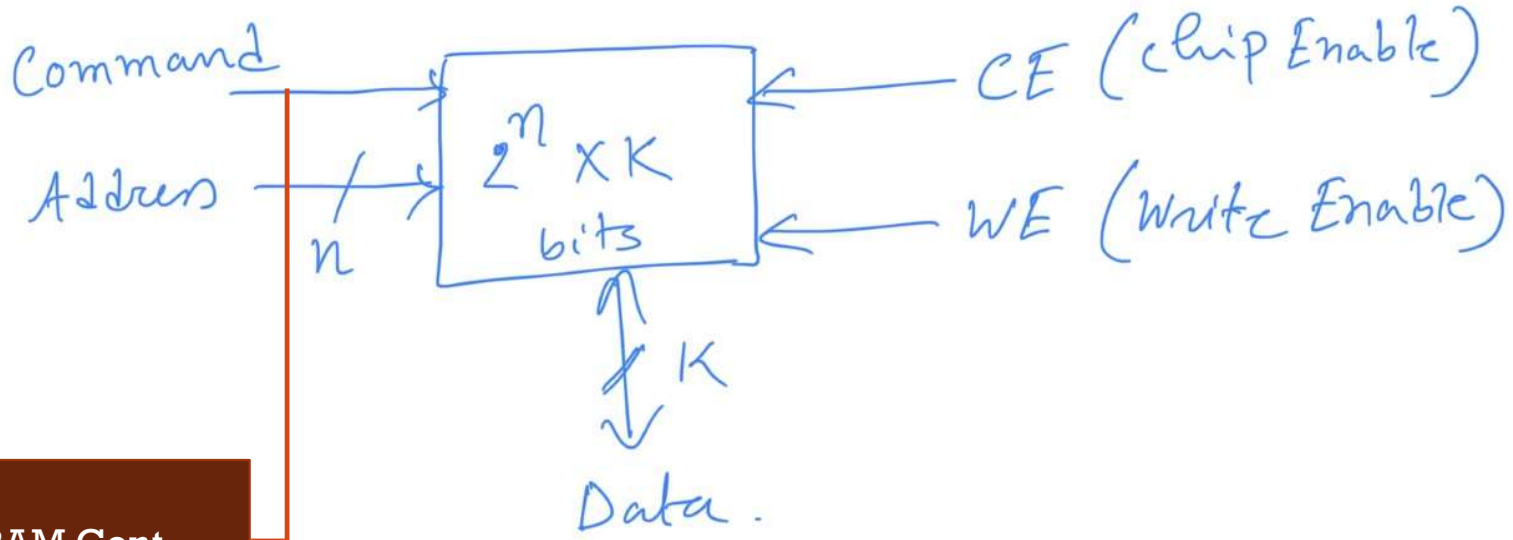
Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand[1] Saugata Ghose[1] Youngsok Kim[2]
Rachata Ausavarungnirun[1] Eric Shiu[3] Rahul Thakur[3] Daehyun Kim[4,3]
Aki Kuusela[3] Allan Knies[3] Parthasarathy Ranganathan[3] Onur Mutlu[5,1]

[1]Carnegie Mellon University [2]Dept. of ECE, Seoul National University [3]Google [4]Samsung Research [5]ETH Zürich

Just imagine ~62.7% of total energy is spent only on data movements

MEMORY (DRAM)



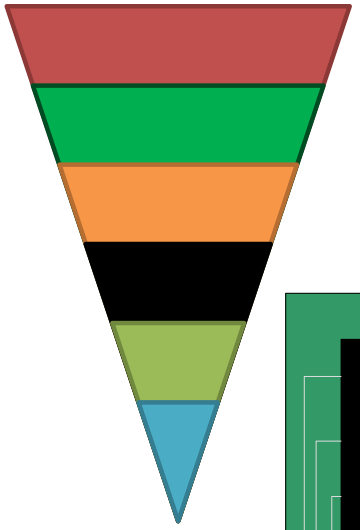
Memory Access from processor

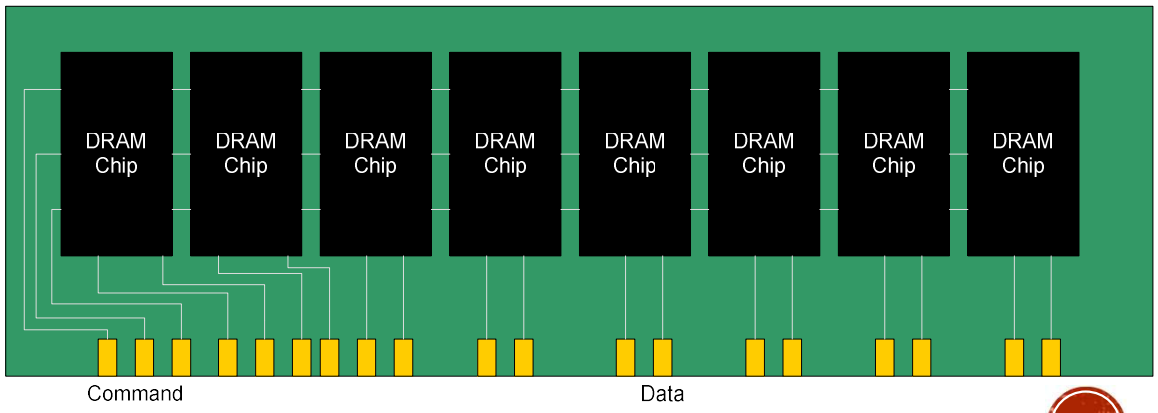
DRAM Cont.

DRAM SUBSYSTEM ORGANIZATION

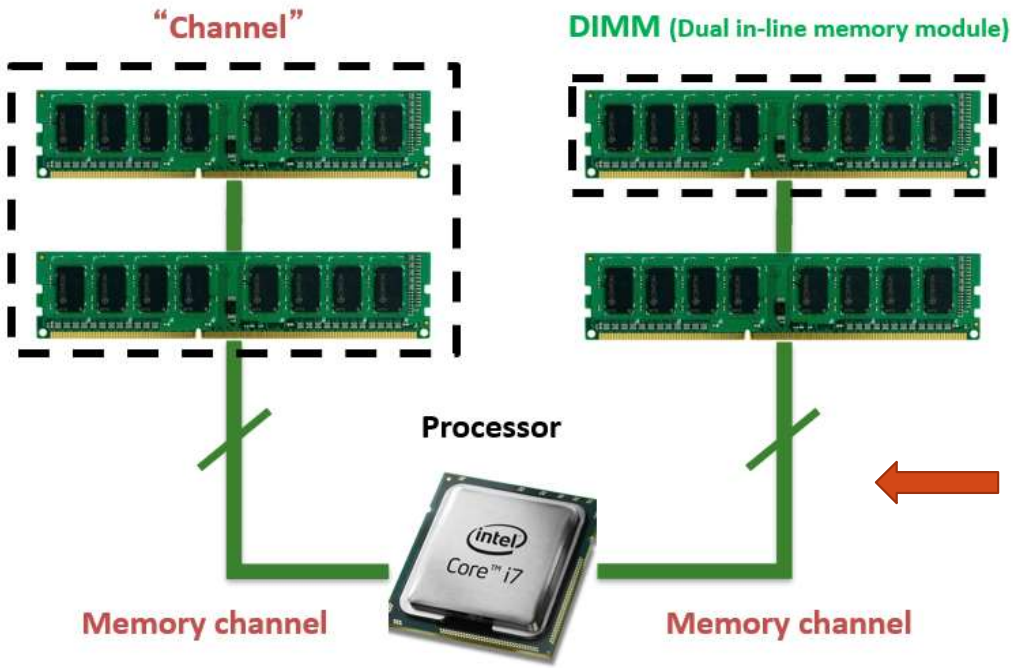
Sub-hierarchy:

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
- Cell





THE DRAM SUB-SYSTEM

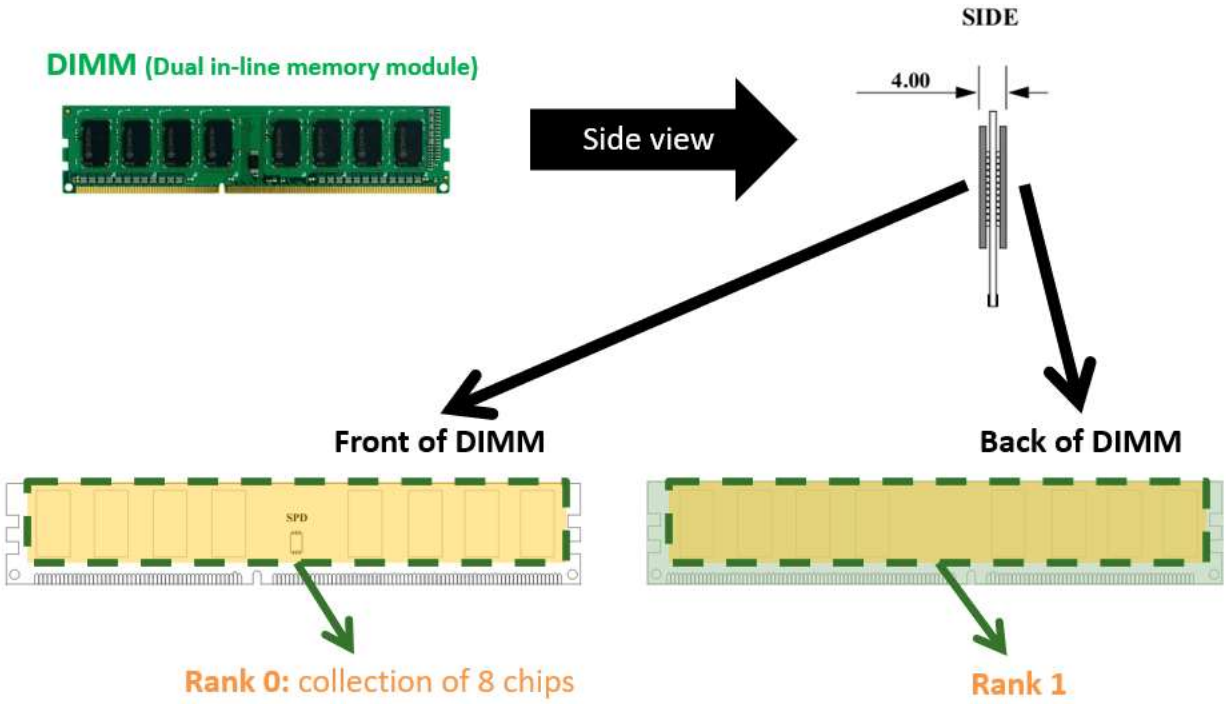


Two programs can access two channels independently.

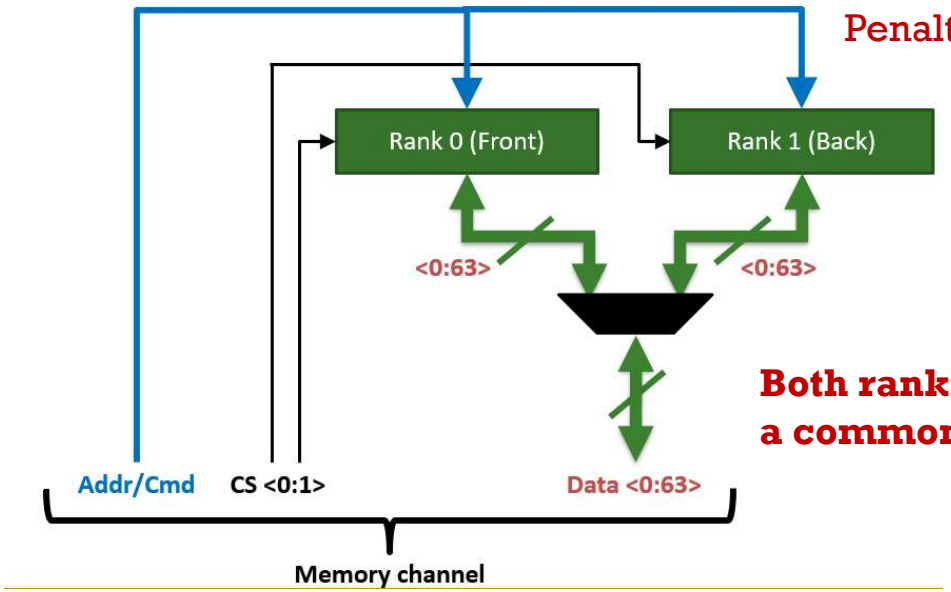
Independent Bus

More channel-> More Pins-> More controller or a complex controller-> More Power-> More Parallelism

BREAKING DOWN A DIMM



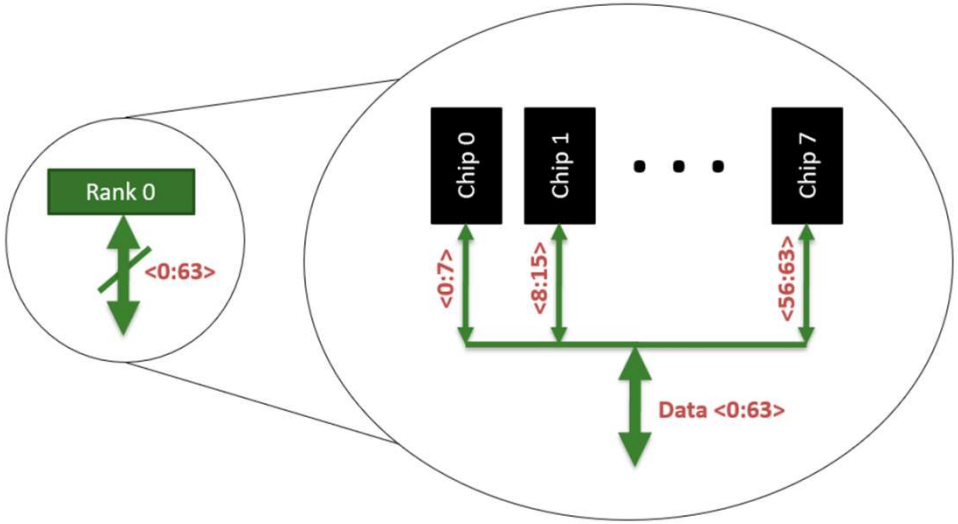
A RANK



Penalty switching the ranks

Both ranks have a common add./comm./data bus

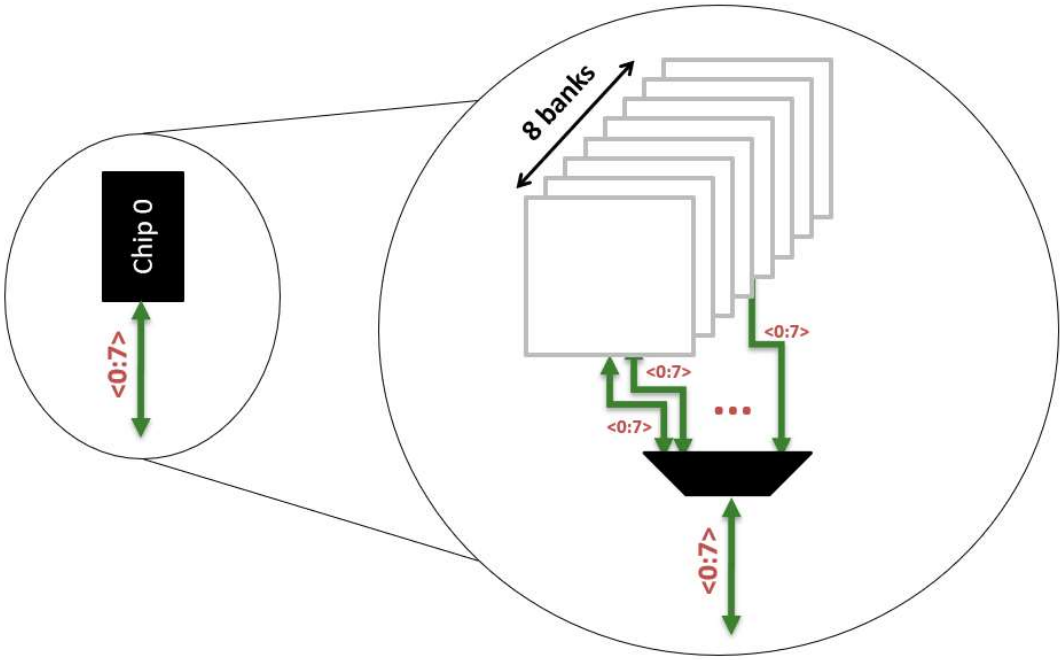
BREAKING DOWN A RANK



- Rank: Multiple chips operated together to form a wide interface
- All chips comprising a rank are controlled at the same time
  - Respond to a single command
  - Share address and command buses, but provide different data

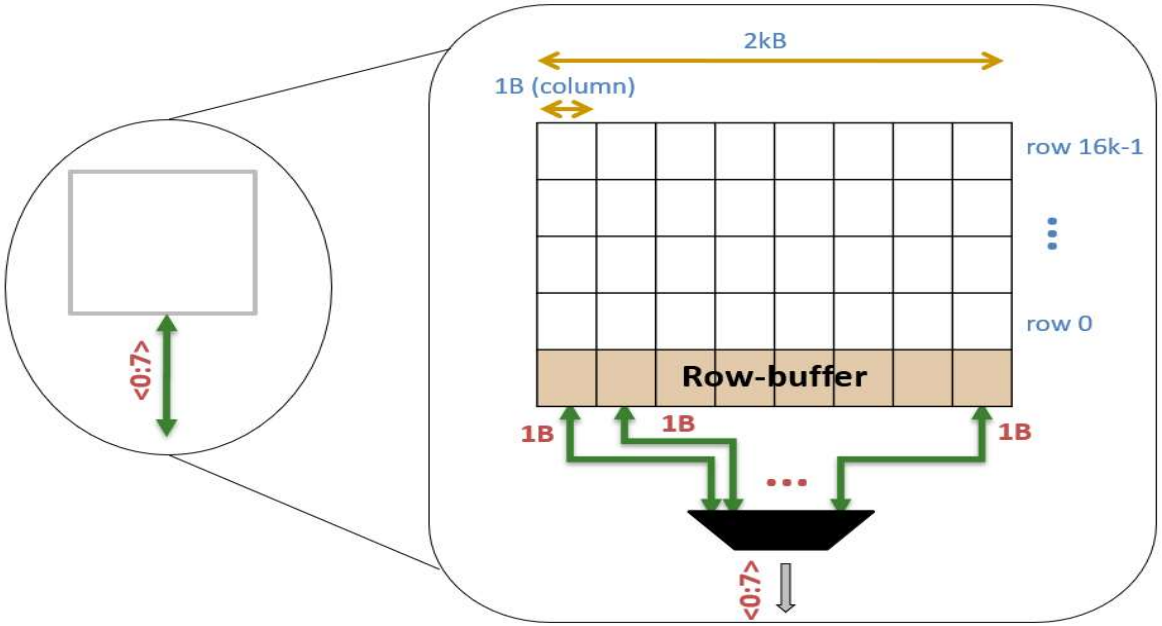
Chips controlled by same address/command buses but provide different data. Also Share the wide data bus.

BREAKING DOWN A CHIP

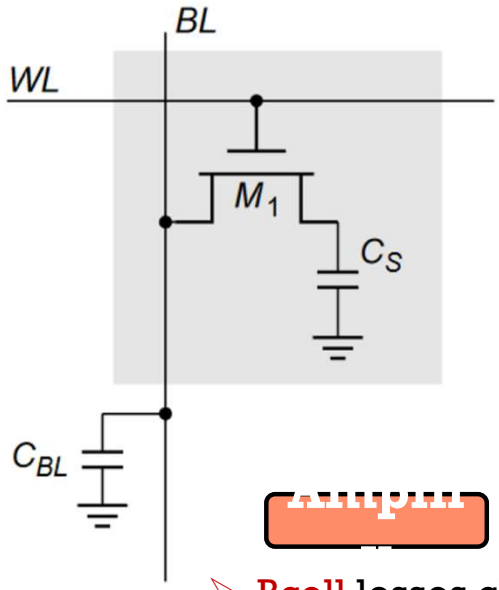


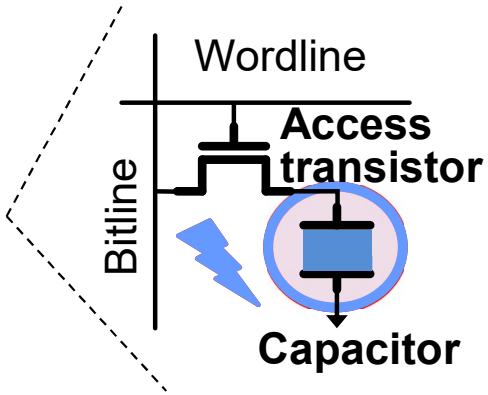
- 8-16 banks/chip is very common today.
- Banks share com./add./data buses.
- The chips have a narrow interface (4-16 bits per read)
- **Changing the number of banks, size of the interface (pins), whether or not com./add./data buses** are shared has a significant impact on the DRAM system cost.

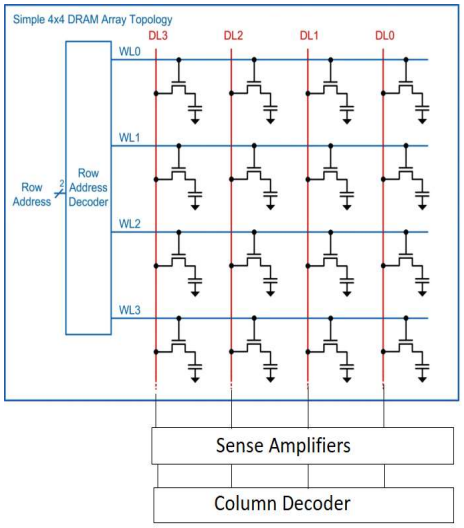
BREAKING DOWN A BANK



1T1C DRAM CELL AND SENSE AMPLIFIER







- Bcell losses charge when read.
- Bcell losses charge over the time.

DRAM BANK OPERATION

A few important timing Parameters:

t_{RAS}: row access strobe

t_{CAS}: column access strobe

t_{RP}: Precharge time

Access Address:

(Row 0, Column 0)

(Row 0, Column 1)

(Row 0, Column 85)

(Row 1, Column 0)

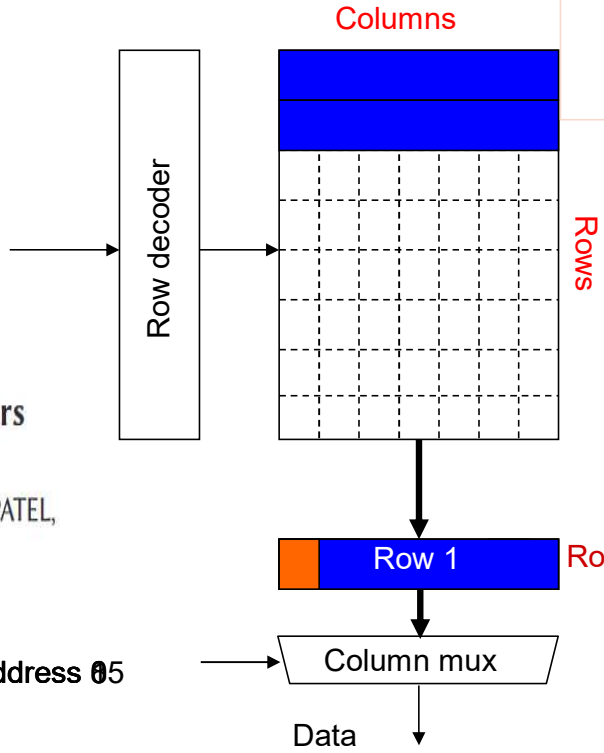


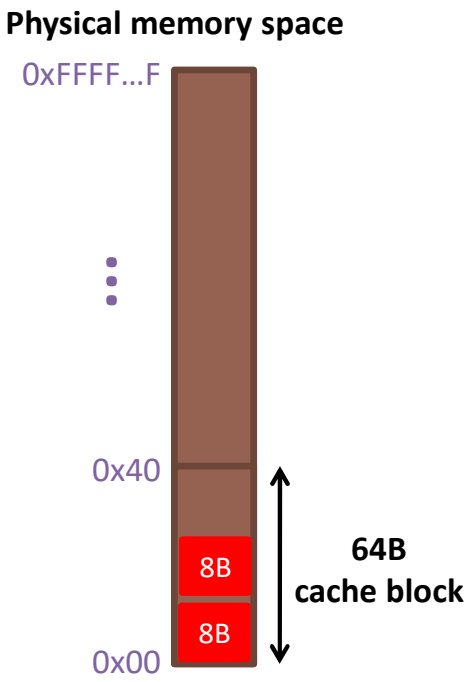
Table 1. JEDEC Timing Constraints [12]

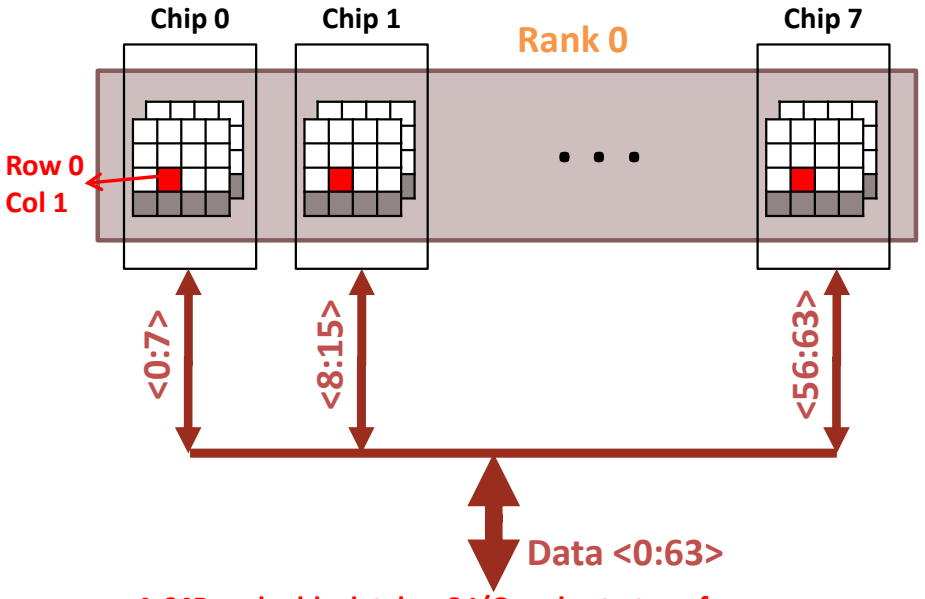
JEDEC Specifications (cycles)			
Parameters	Description	DDR3-1600H	DDR4-1600K
t_{RCD}	ACT to RD/WR delay	9	11
t_{RL}	RD to Data Start	9	11
t_{RP}	PRE to ACT Delay	9	11
t_{WL}	WR to Data Start	8	9
t_{RTW}	RD to WR Delay	7	8
t_{WTR}	WR to RD Delay	6	6
t_{RTP}	Read to PRE Delay	6	6
t_{WR}	Data End of WR to PRE	12	12
t_{RAS}	ACT to PRE Delay	28	28
t_{RC}	ACT-ACT (same bank)	37	39
t_{RRD}	ACT-ACT (diff bank)	5	4
t_{FAW}	Four ACT Window	24	20
t_{BUS}	Data bus transfer	4	4
t_{RTR}	Rank to Rank Switch	2	2

A Comparative Study of Predictable DRAM Controllers

DANLU GUO, MOHAMED HASSAN, RODOLFO PELLIZZONI, and HIREN PATEL,
University of Waterloo

EXAMPLE: TRANSFERRING A CACHE BLOCK



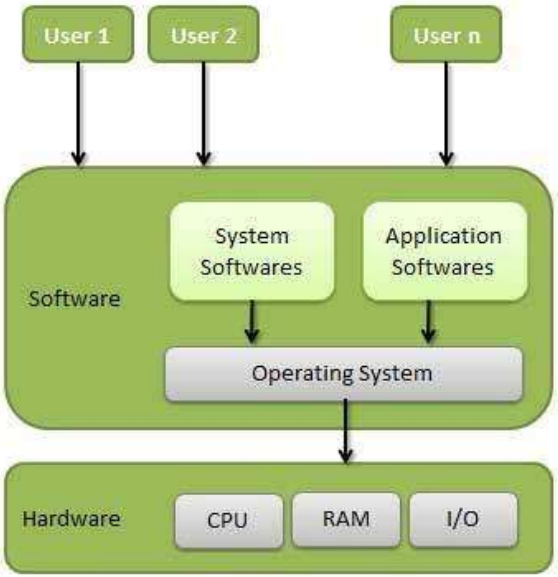


A 64B cache block takes 8 I/O cycles to transfer.
During the process, 8 columns are read sequentially.

Bringing from one rank-> Saves switching penalty between ranks

WHAT IS OPERATING SYSTEM?

- An interface between H/W and S/W.



Operating system goals:

- Execute user programs.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

IMPORTANT TASKS OF AN OPERATING SYSTEM.

- Memory Management (tracking the MM's availability, MM allocation/deallocation for process).
- Processor Management (status of process, scheduling, context switch, etc.).
- Device Management (device communication via their respective drivers, Keeps tracks of all devices, Decides which process gets the device when and for how much time, allocation/deallocation for higher resource utilization).
- File Management (Keeps track of information, location, uses, status etc.).
- Protection and Security (Process address space, File access rights, etc.).

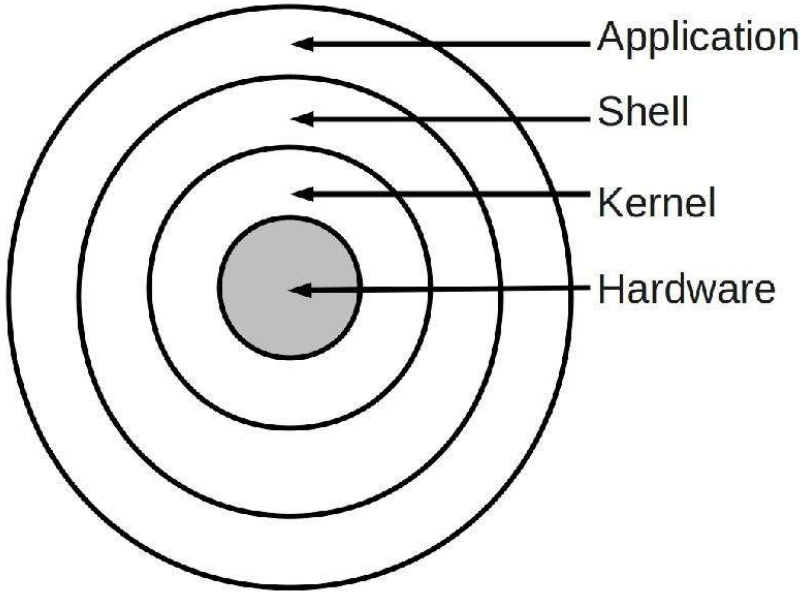
TYPE OF OS

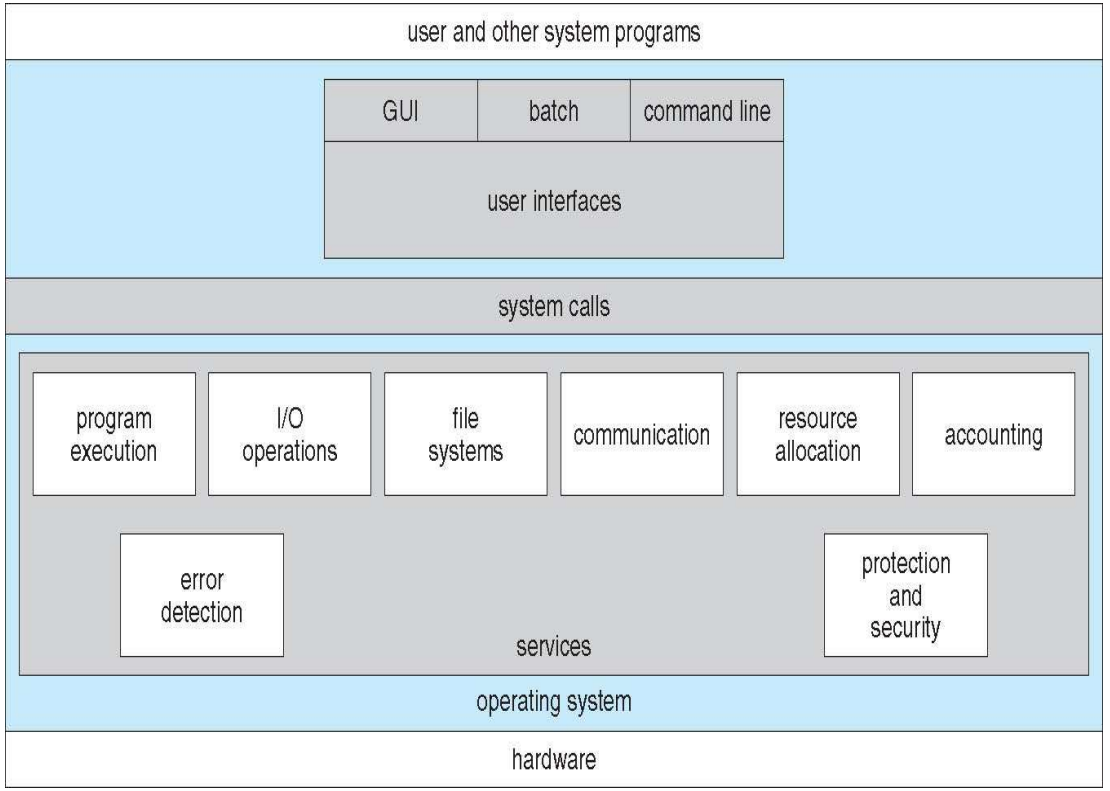
- Batch OS
  - The users who are using a batch operating system do not interact with the computer directly. Each user prepares its job on an off-line device and submits it to the computing system.
  - There is a lack of interaction between the user and the job.
  - The other jobs will have to wait for an unknown time if any job fails.
  - Throughput is less.

  Example: IBM's z/OS , 2000.
- Multiprogramming
  - **Multiprogramming OS** is an ability of an operating system that executes more than one program using a single processor machine.
  - More than one task or program or jobs are present inside the main memory at one point of time.
  - If a process goes to I/O, CPU can be scheduled to other process.
- Multitasking
  - Multi-tasking is a logical extension of multiprogramming.
  - Multitasking is the ability of an OS to execute more than one **task** simultaneously on *a CPU machine.*
  - The CPU executes multiple jobs by switching among them using a small-time quantum, and the switches occur so quickly that the users feel like interacting with each tasks at the same time.
- Multiprocessing
  - There are more than one processor which work in parallel to the others.
  - More processes can be executed in parallel.
  - Multiprocessor systems are more reliable.
- Realtime
  - A real-time operating system (RTOS) is an operating system (OS) for real-time applications that processes data and events that have critically defined time constraints.

  Example: Airline traffic control systems.

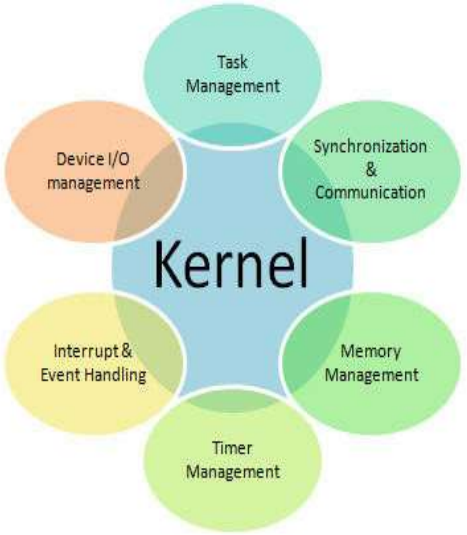
ARCHITECTURE AND SERVICES OF OS



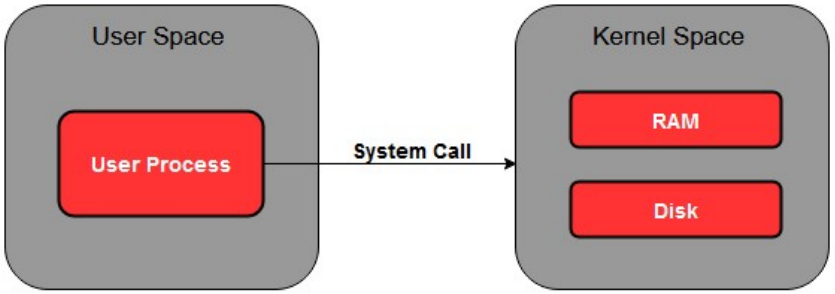


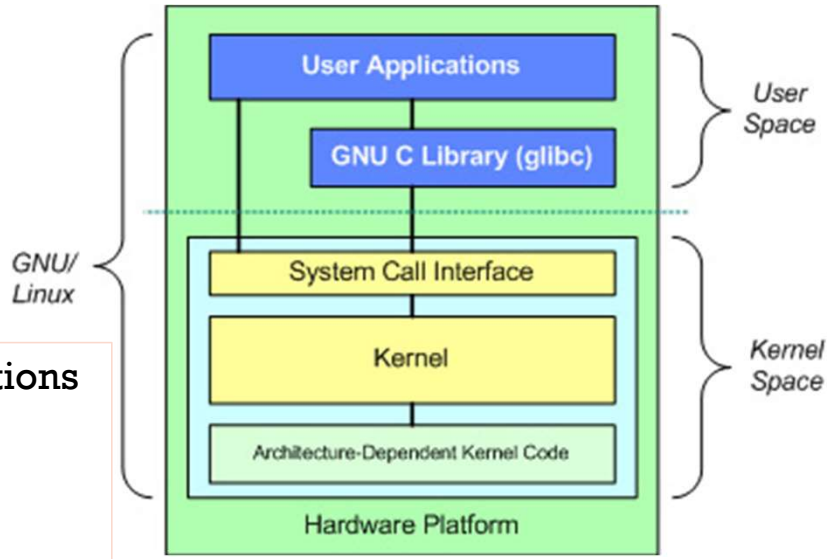
KERNEL

- Kernel is the core part of an OS.
- It has full control over everything in the system.
- It is an interface between applications and data processing done at the hardware level.
- It is the computer program that first loaded on start-up the system (After the bootloader).
- Once it is loaded, it manages the remaining start-ups.
- It also manages memory, peripheral, and I/O requests from software.
- Device Management, Memory Management, Resource Management.



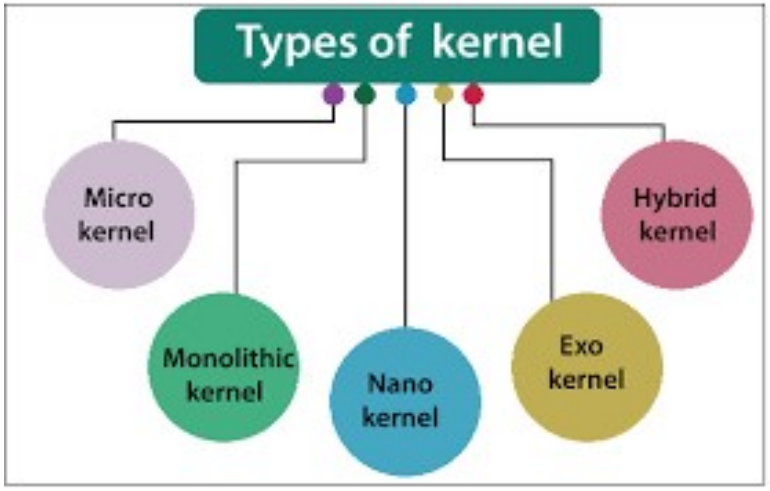
LOGICAL SEGREGATION OF MEMORY

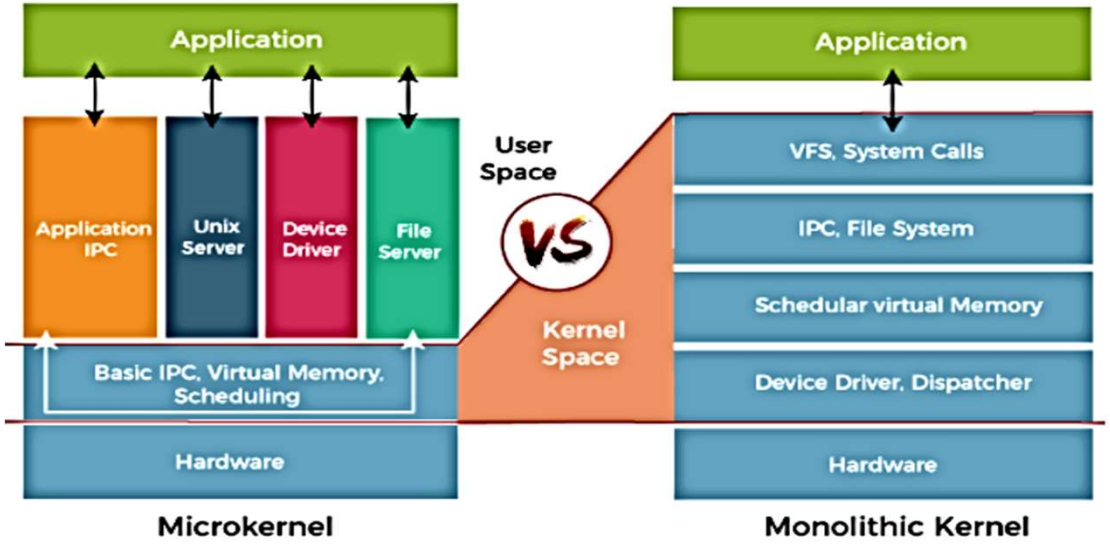




- The user space is the memory space where all user applications or application software executes.
- Kernels and OS core execute in the kernel space.
- System Calls are the interface between them.
- Full access to the memory and h/w resources vs. limited access.
- Page tables of processes, kernel data structures, and threads are created in kernel space while program codes, stacks, heaps of the processes are created in user's space.

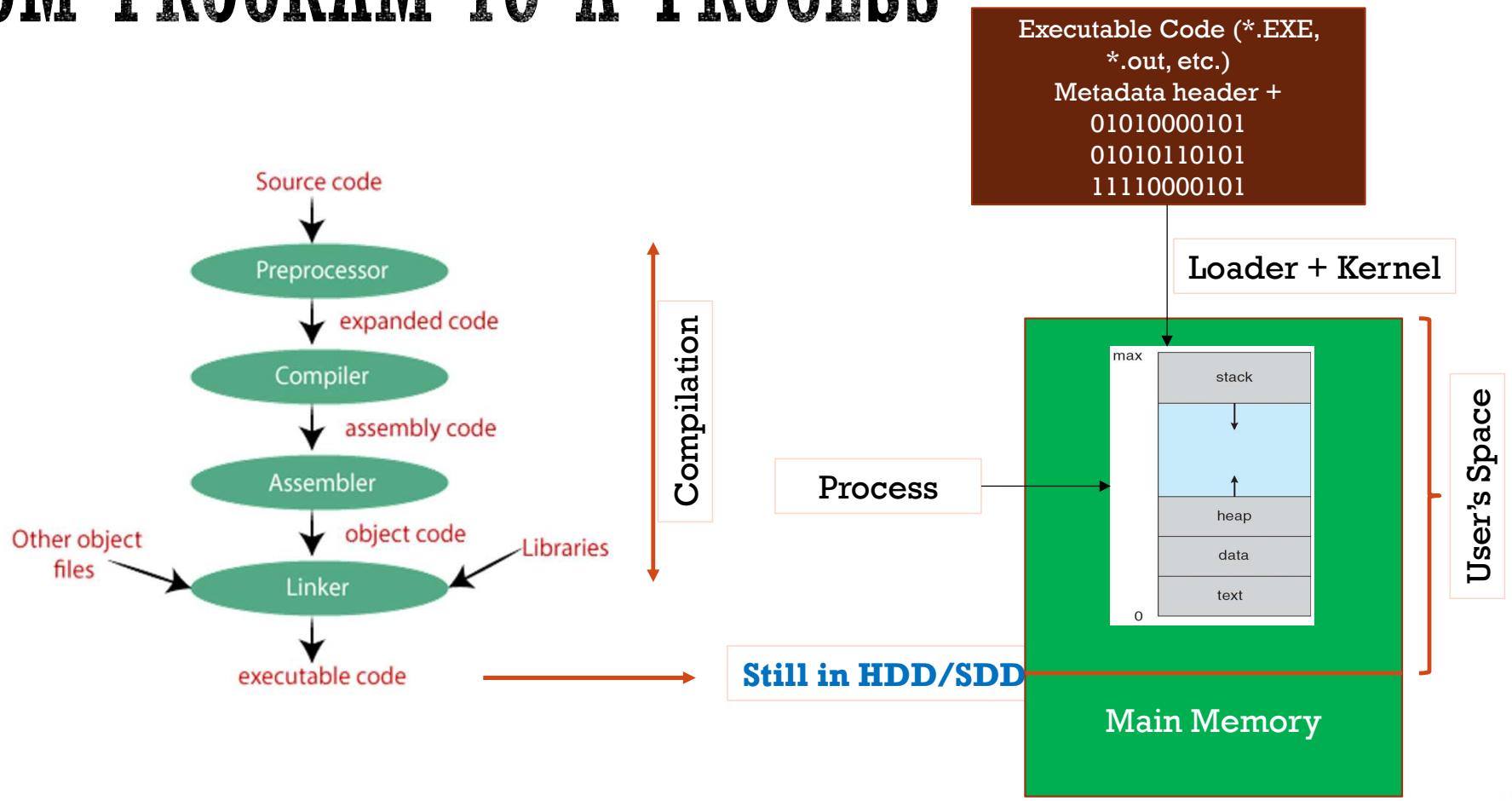
VARIOUS TYPES OF KERNELS



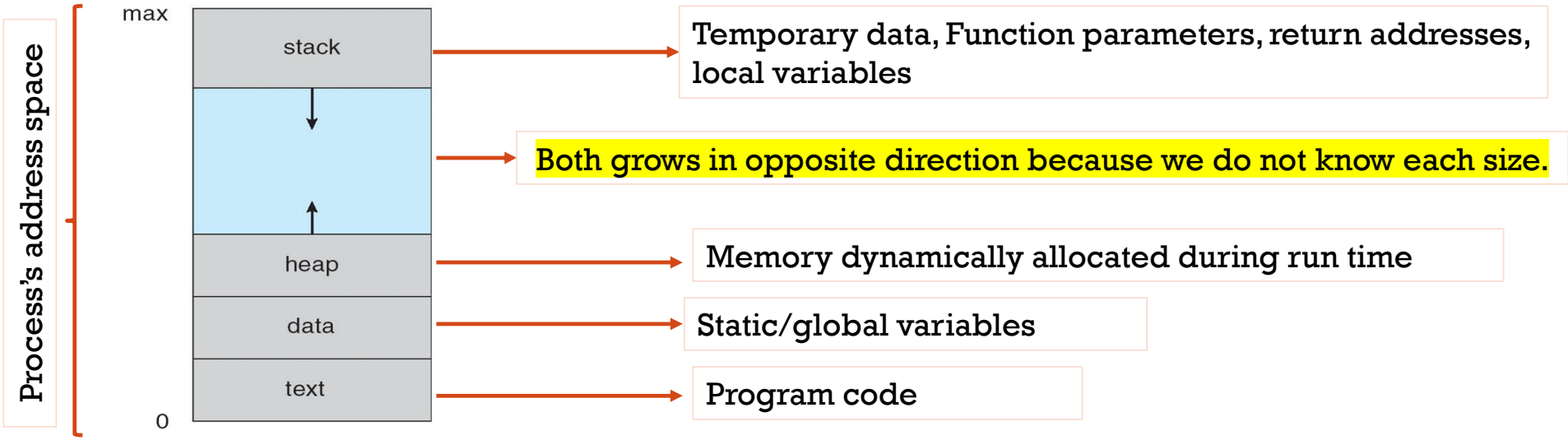


- Both user services and kernel services are kept in the same address space vs. user services and kernels services are kept in separate address spaces.
- The monolithic kernel is larger than the microkernel.
- Faster Vs. slower execution.
- Linux/DOS (monolithic kernel) vs. MAC Os (micro kernel).
- If any service generates any error in monolithic kernel, it may crash down the whole system. Portability is also less.
- Hybrid kernel is the combination of both monolithic kernel and microkernel. It has speed and design of monolithic kernel and modularity and stability of microkernel (Ex. Windows NT).

FROM PROGRAM TO A PROCESS



PROCESS (A PROGRAM IN EXECUTION) IN MEMORY



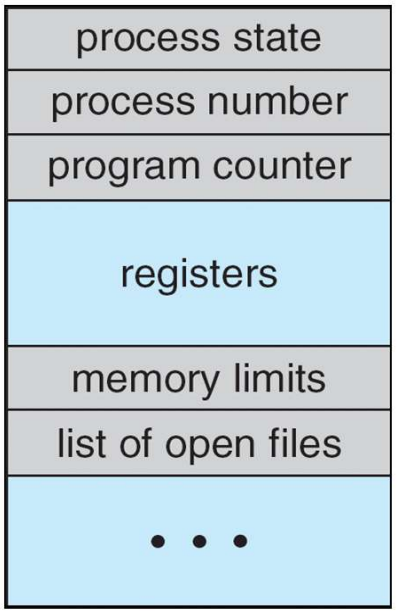
***Trying to access beyond process boundaries may lead to segmentation fault.

PROCESS'S ATTRIBUTES: HELPS TRACKING EACH PROCESS

Attributes: Information associated with each process.

Attributes (context of a process) are stored in Process Control Block (PCB), also called as **task control block.**

- Process state – running, waiting, etc.
- Process number – Id for a process.
- Program counter – location of instruction to next execute.
- CPU registers – contents of all process-centric registers.
- CPU scheduling information- priorities, scheduling queue pointers.
- Memory-management information – memory allocated to the process.
- Accounting information – CPU used, clock time elapsed since start, time limits.
- I/O status information – I/O devices allocated to process, list of open files.

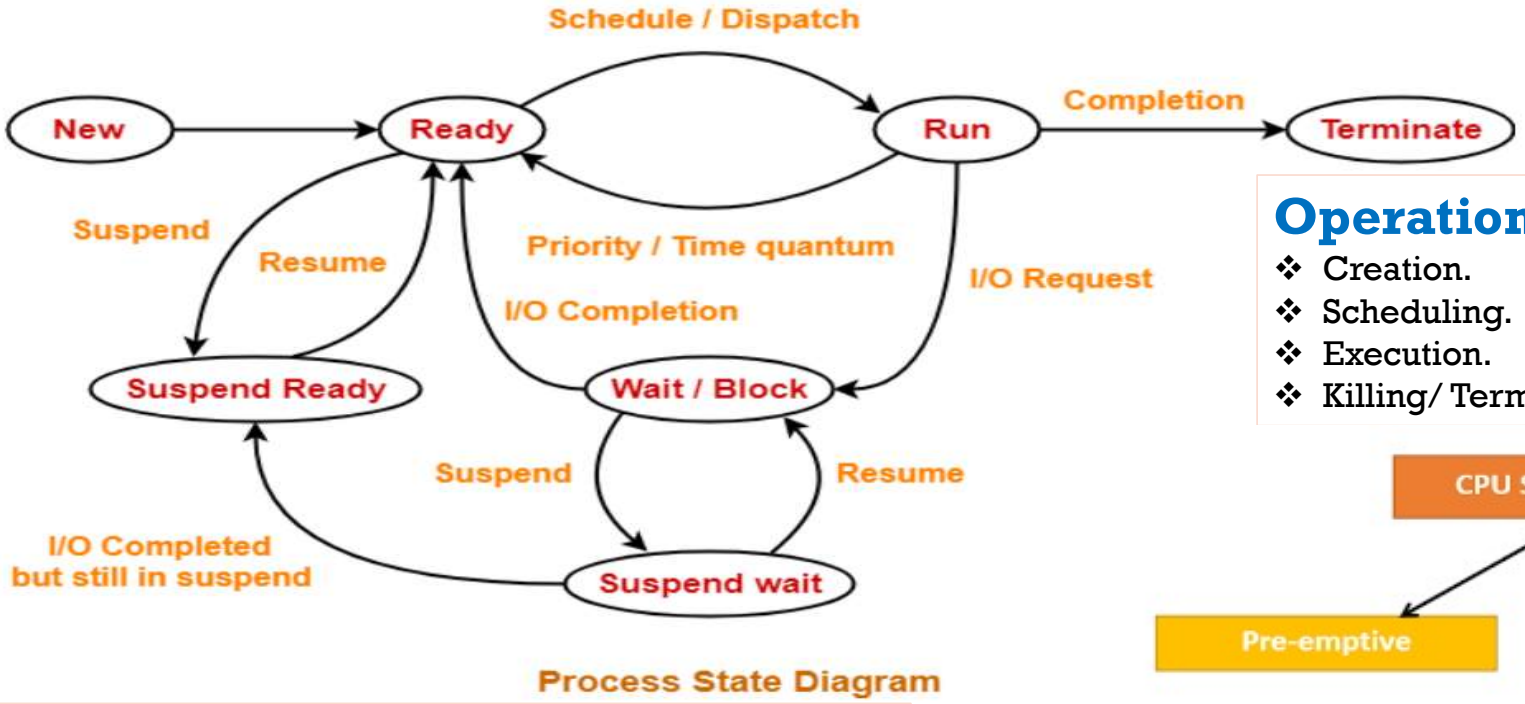


PCB

**********PCBs (linked list of PCBs) are stored in kernel space.**********

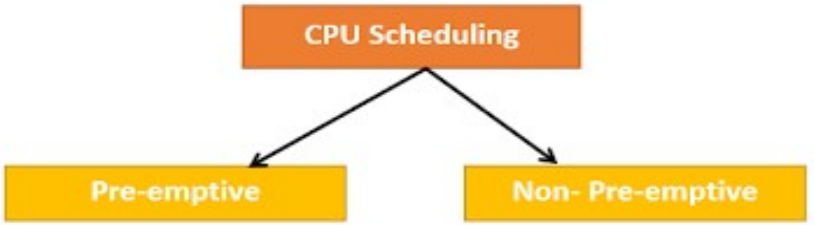
STATE TRANSITION DIAGRAM OF PROCESS

Process States:
- New
- Ready
- Run
- Terminate
- Wait/Block
- Suspend wait
- Suspend Ready



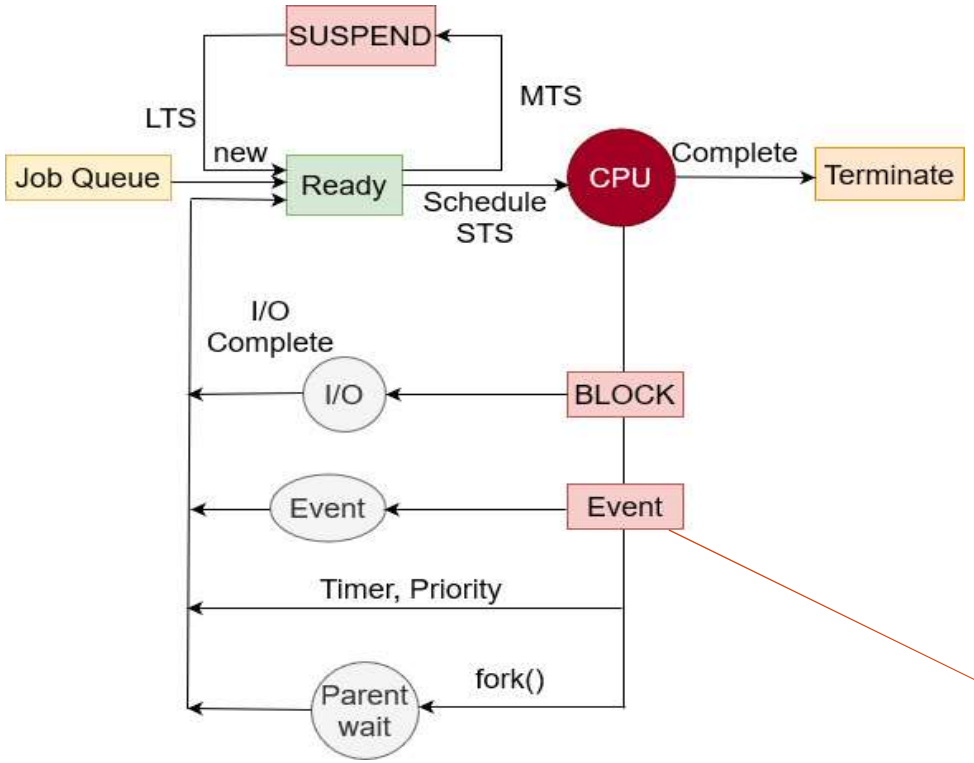
Process State Diagram

Operations on processes:
- Creation.
- Scheduling.
- Execution.
- Killing/ Termination.



Long term, Short term, Medium term schedulers.
CPU Bound vs. I/O Bound.

SCHEDULING QUEUE



Consider a system with N CPU processors and M Processes, then

	Min	Max
Ready	?	?
Running	?	?
Block	?	?

Synchronization techniques

SOME IMPORTANT TIMING PARAMETERS OF PROCESSES

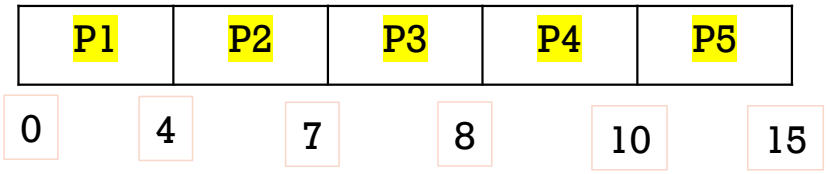
- Arrival time. ---> P1 arrives to ready queue (time point).
- Burst time. ---> Total CPU cycles consume by P1 (duration).
- Completion time. ---> P1 terminates (time point).
- Turn around time. ---> Completion time – arrival time (duration).
- Waiting time. ---> Turn around time – Burst time (duration).
- Response time. ---> P1 gets the first CPU access after it arrives in ready queue (duration).

CPU SCHEDULING (WHO, WHERE, AND WHEN)

- Who ---> Short Term Scheduler
- Where ---> Ready State to running state
- When ----> when a process moves from
  - Run -------> Termination **(Sure shot) -------> Execution over.**
  - Run --------> Wait **(Sure shot) -------> I/O Occurred.**
  - Run --------> Ready **(Sure shot) -------> Time quantum expired, high priority process arrives.**
  - New -------> Ready **(may be) -------> New process with high priority is created.**
  - Wait -------> ready **(may be) -------> High priority process completes its I/O.**

FCFS ALGORITHM

- Criteria: Arrival time
- Mode : Non- preemptive
- Will consider 1 CPU
- 4 basic states
- Response time = Waiting time (Non-preemptive)



Gantt chart

Pid	Arrival time	Burst time	Completion time	Turn around time	Waiting time
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6

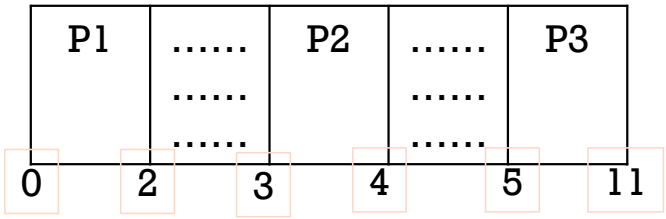
Avg. waiting time?
Avg. Turn around time?

Convoy effect or starvation

A Simple queue can be used to implement FCFS.

FCFS (CONT…)

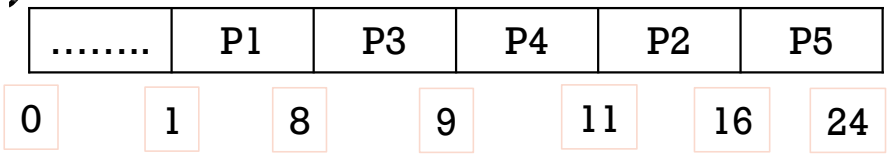
PNO	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	3	1	4	1	0
3	5	6	11	6	0



Here, the processor is actually waiting for the processes.

SHORTEST JOB FIRST (SJF)

- Criteria: Burst time
- Mode : Non- preemptive



Gantt chart

Pid	Arrival time	Burst time	Completion time	Turn around time	Waiting time
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11

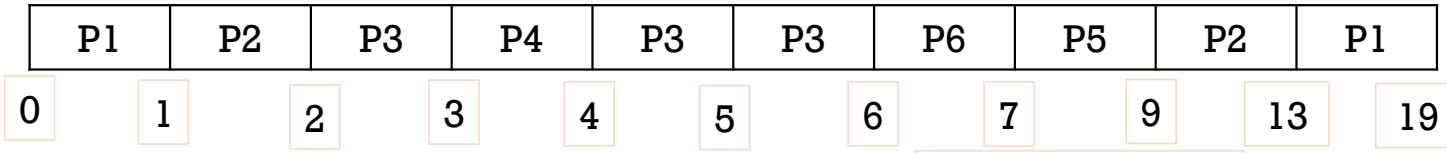
Avg. waiting time?
Avg. Turn around time?

- Best, but is it practical?
- Min heap DS can be used.
- If burst time is same, arrival time can be used.
- Maximum throughput, minimum WT and TAT.
- Burst time prediction based on size, type, average, etc.

Can there be any convoy effect?

SHORTEST REMAINING TIME FIRST (SRTF)

- Criteria: Burst time
- Mode : Preemptive



Pid	Arrival time	Burst time	Completion time	Turn around time	Waiting time
1	0	7	?	?	?
2	1	5	?	?	?
3	2	3	?	?	?
4	3	1	?	?	?
5	4	2	?	?	?
6	5	1	?	?	?

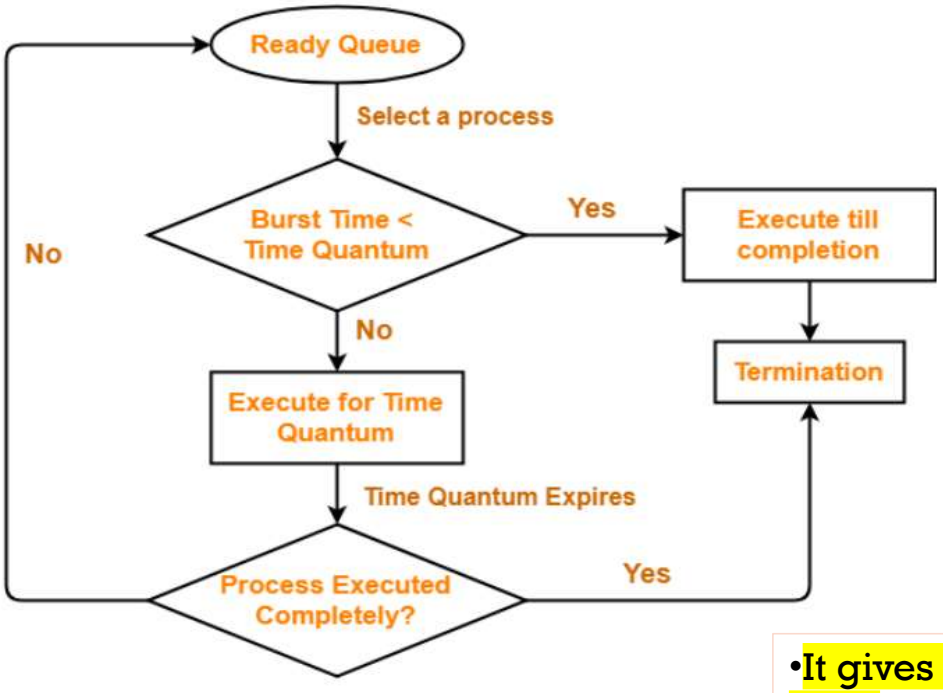
Gantt chart

Avg. waiting time?
Avg. Turn around time?

Practical implementation?
Response Time?

Can there be any convoy effect?

ROUND ROBIN (RR) ALGORITHM



Round Robin Scheduling

- It gives the best performance in terms of average response time.
- It is best suited for time sharing system.

ROUND ROBIN (RR) ALGORITHM

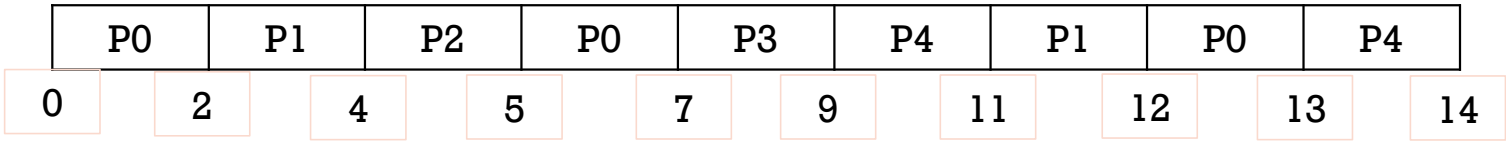
Time quantum = 2 units

Avg. waiting time?
Avg. Turn around time?

Process	Arrival Time	Brust Time
P0	0	5
P1	1	3
P2	2	1
P3	3	2
P4	4	3

- Provides best response time
- Best for time sharing system
- Kind of brings the flavour of SJF
- Longer process may starve
- Highly dependant on time quantum (Overhead of context switching)
- No idea of priority

Recommended Reading- RR with priority



Gantt chart

PRIORITY SCHEDULING

PNO	Priority	AT	BT
1	2 (L)	0	4
2	4	1	2
3	6	2	3
4	10	3	5
5	8	4	1
6	12 (H)	5	4
7	9	6	6

- Priority
  - Static
  - Dynamic
- Priority Scheduling
  - Preemptive
  - Non-preemptive

Lower value can also be considered as higher priority (can very from system to system).

Solve using preemptive and non-preemptive approach

IPC: INTER-PROCESS COMMUNICATION

Processes running concurrently in the OS may be either

- **Independent process (Can not affect or be affected by other processes).**
- **Cooperating process (Can affect or be affected by other processes), ex. Data sharing between processes.**

IPC is the way of interacting between processes→ requires for cooperating processes.

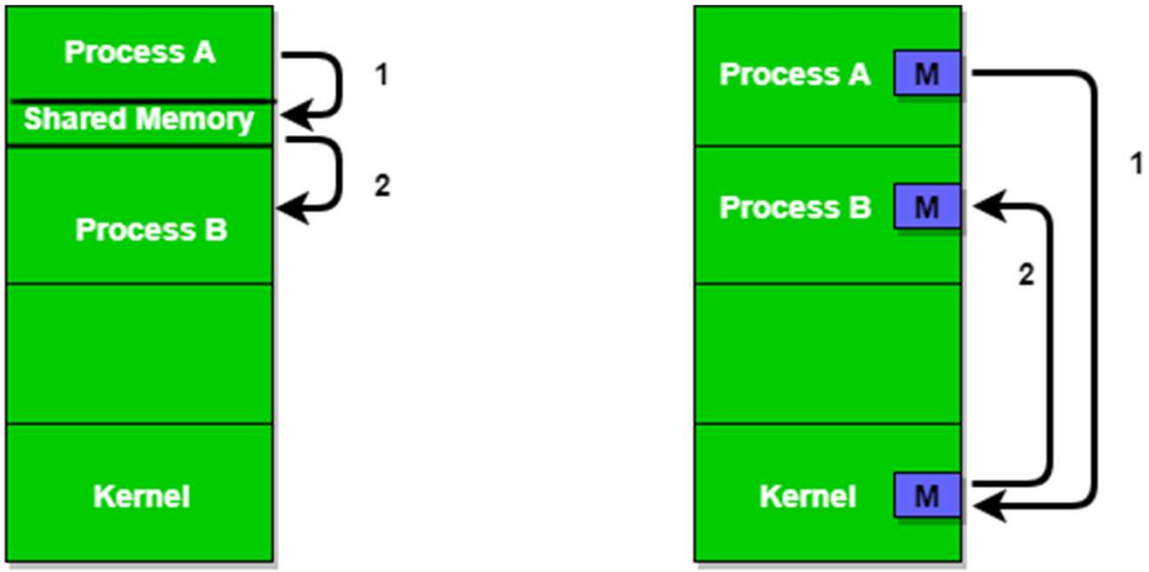
Cooperating processes.

A few reasons for IPC:

1. Information Sharing.
2. Computational Speedup.
3. Modularity.

Cooperation or competition is a problem.

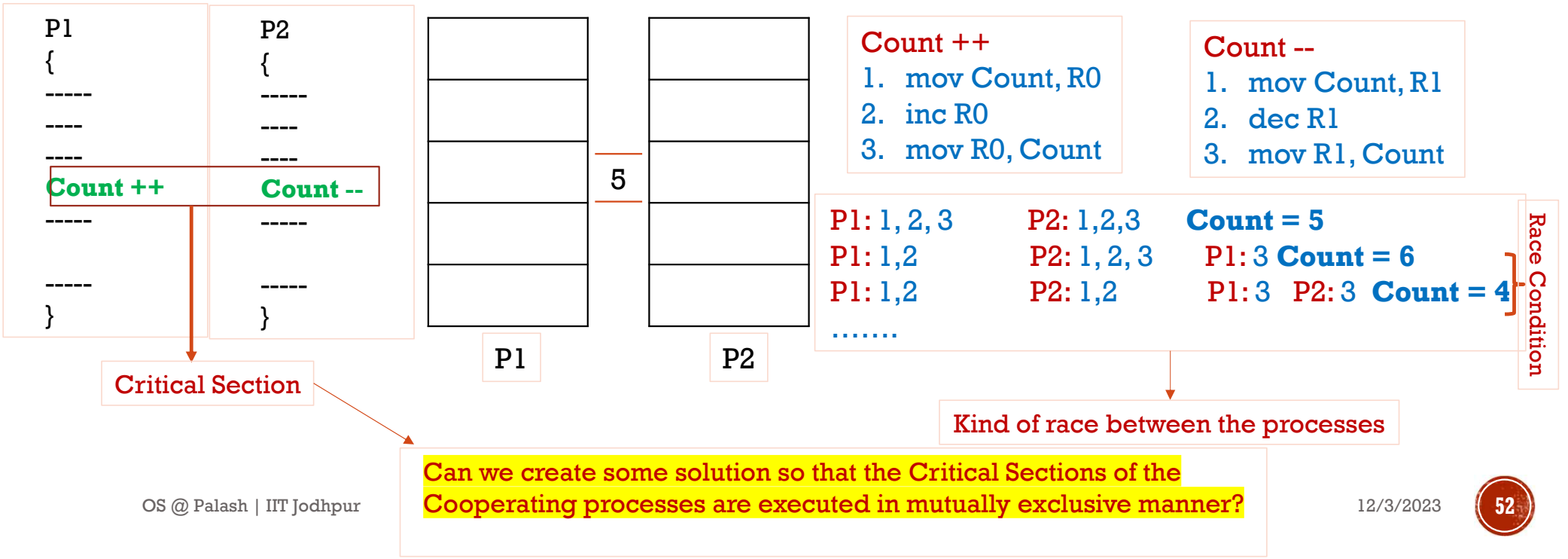
TWO WAYS OF IPC



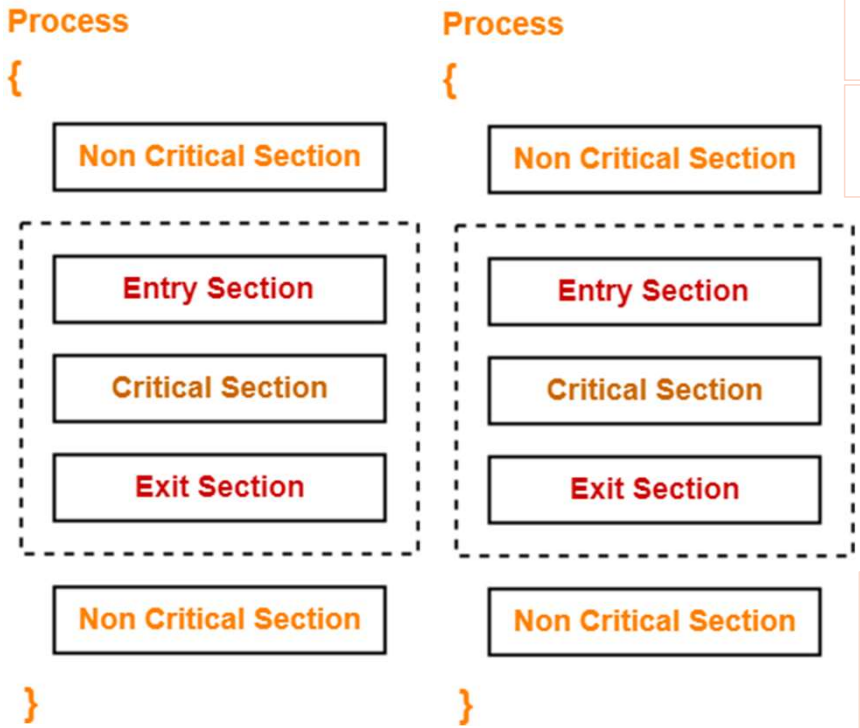
Shared memory and message passing

PROCESS SYNCHRONIZATION

Cooperating processes improves performance but it also leads to problem.



PROCESS SYNCHRONIZATION (CONT....)



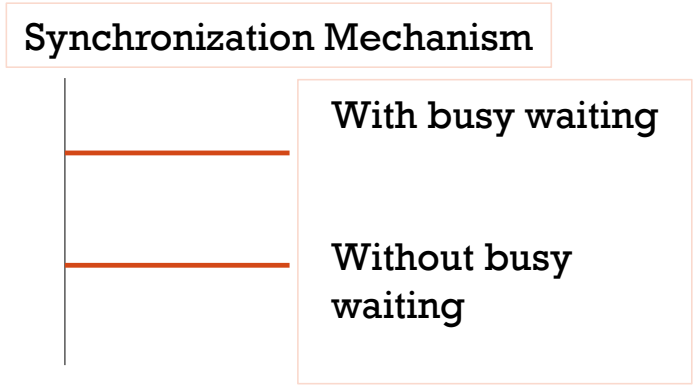
Critical Section: It is the part of the program where shared resources are accessed by the concurrent processes.

Race Condition: The order of execution of instructions defines the result produced (inconsistent).

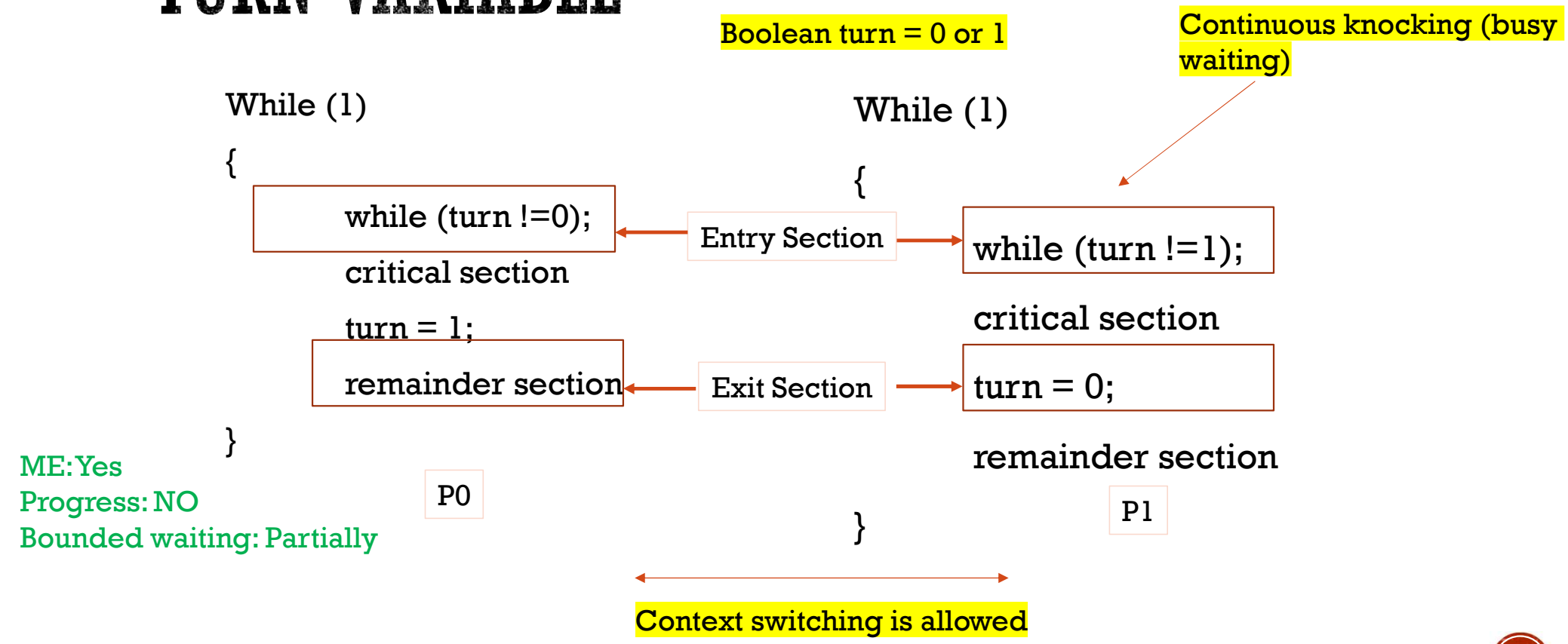
Entry/Exit sections are the part of synchronization process. These sections need to be designed carefully so that certain requirements are fulfilled.

A FEW REQUIREMENTS OF SYNCHRONIZATION MECHANISM

- Primary:
  - Mutual exclusion
  - Progress
- Secondary:
  - Bounded waiting
  - Portability or architecture neutrality



TURN VARIABLE



ME: Yes
Progress: NO
Bounded waiting: Partially

FLAG VARIABLE

```
While (1)
{
        flag[0] = T;
        while (flag[1]);
        critical section
        flag [0] = F
}
```

P0

```
While (1)
{
        flag[1] = T;
        while (flag[0]);
        critical section
        flag [1] = F
}
```

P1

flag	F	F
	0	1

ME: Yes
Progress: Yes (Partial)
Bounded waiting: Partially because of deadlock
Deadlock??

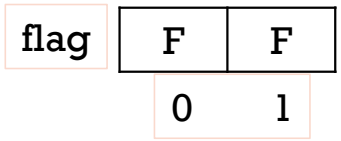
PETERSON'S SOLUTION

```
While (1)
{
        flag[0] = T;
        turn = 1;
        while (turn == 1 && flag[1] == T);
        critical section
        flag [0] = F
}
```

P0

```
While (1)
{
        flag[1] = T;
        turn = 0;
        while (turn == 0 && flag[0] == T);
        critical section
        flag [1] = F
}
```

P1



ANSWER ME THE FOLLOWING QUESTION

- Consider the following two processes, P1 and P2 for accessing critical section. The initial value of shared boolean variable S1, S2 is randomly assigned.

P0	P1
While (s1 == s2);	While (s1 != s2);
Critical section	Critical section
S1 = S2;	S1 = not (S2);

Check the synchronization requirements….?

SEMAPHORES

- A semaphore is an integer variable, shared among multiple processes.
- Variables on which read, modify, and update happens atomically in kernel mode (no preemption)
- Solution for N processes
- Two types of semaphores:
  - Binary Semaphores (Critical Section problems)
  - Counting Semaphores (Resource management)

  - We can also do ordering of processes.

SEMAPHORES (CONT....)

```
do {
wait (S);
Critical section
Signal (S)
Remainder section
}
```

```
wait(S)

wait (S){
while(s<=0);
S = S-1;
}
```

```
Signal (S)

Signal (S)

{
    S = S + 1
}
```

TWO MORE WORKS OF SEMAPHORES

We want: P2-----> P1 -------> P3

Wait (S1) P1 Signal (S2)	P2 Signal (S1)	Wait (S2) P3

S1 = 0 | S2 = 0

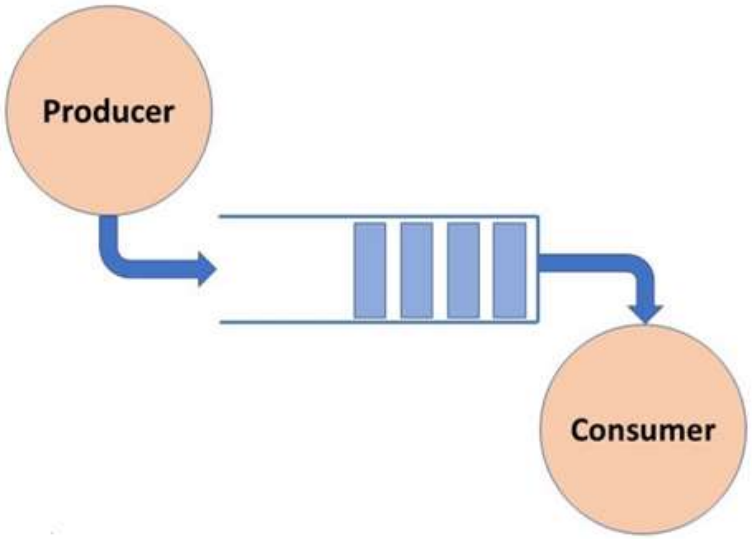
Scheduler can schedule any processes

Deciding the order of execution

```
P1
{
        wait (S)
        CS
        signal (S)
                RS
}
S = 5 (may be 5 printers)
```

Resource Management

PRODUCER-CONSUMER PROBLEM



- A classical synchronization problem
- Overflow
- Underflow
- CS problem

Producer/consumer can run many times back to back since buffer has more than one place

SOLUTION

```
S = 1
E = N
F = 0
```

```
Producer ()
{
      while (T) {
            wait(E)
            wait(S)
            append()
            Signal (S)
            Signal (F)

      }
}
```

```
Consumer ()
{
      while (T) {
            wait(F)
            wait(S)
            take()
            Signal (S)
            Signal (E)

      }
}
```

READER-WRITER PROBLEM

```
Wait (wrt)
    write operation
Signal (wrt)
```

Writer Process

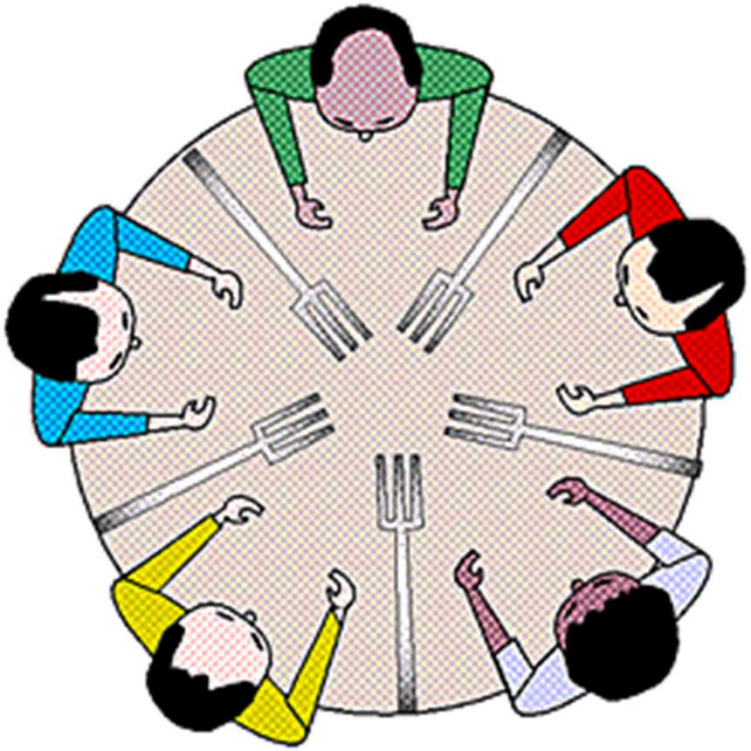
```
Wait (mutex)
    readcount ++
        if (readcount ==1)
            wait(wrt)
Signal (mutex)


Read operation

Wait (mutex)
    readcount --
        if (readcount ==0)
            Signal (wrt)
Signal (mutex)
```

Reader Process

DINING PHILOSOPHERS PROBLEM



- A version of the classical synchronization problem
- 5 philosophers sit around a circular table and alternate between thinking and eating.
- A bowl of noodles and five forks (chopstick) for each philosopher are placed at the center of the table.

Conditions:

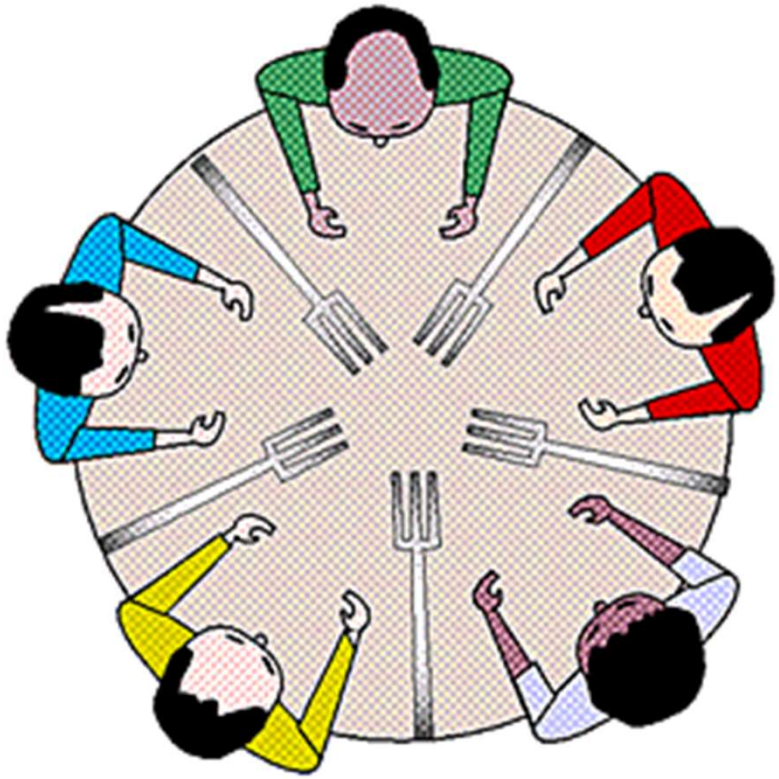
- A philosopher must use both their right and left forks to eat.
- A philosopher can only eat if both of his or her immediate left and right forks are available.
- Only one fork may be picked at a time.
- Philosopher must release the forks once eating is done. Then he/she may again starts thinking.
- If one fork is available, he/she will pick one and wait for the other (Hold and wait).

DINING PHILOSOPHERS PROBLEM

```
Void Philosopher (void)
{
      while (T) {
            Thinking ();
            wait(chopstick [i]);
            wait(chopstick [(i+1)%5]);
            Eat();
            signal(chopstick [i]);
            signal(chopstick [(i+1)%5]);


      }
}
```

chopstick	1	1	1	1	1
	0	1	2	3	4

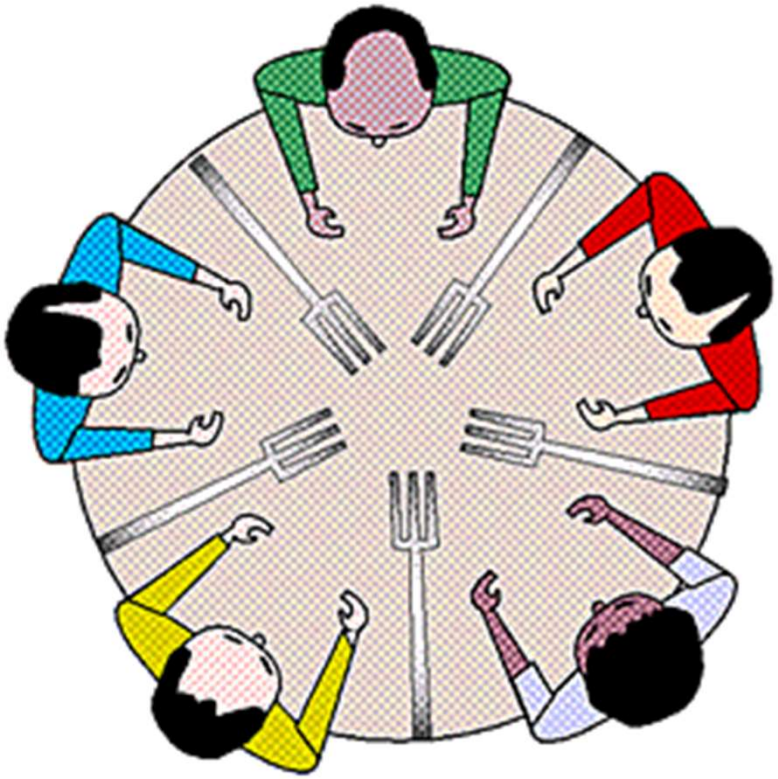


A FEW PROBABLE SOLUTIONS

- Allow at most four philosophers to sit
- Allow six chopstick to be used simultaneously on table
- Allow a philosopher to pick up chopsticks only if both chopsticks are available

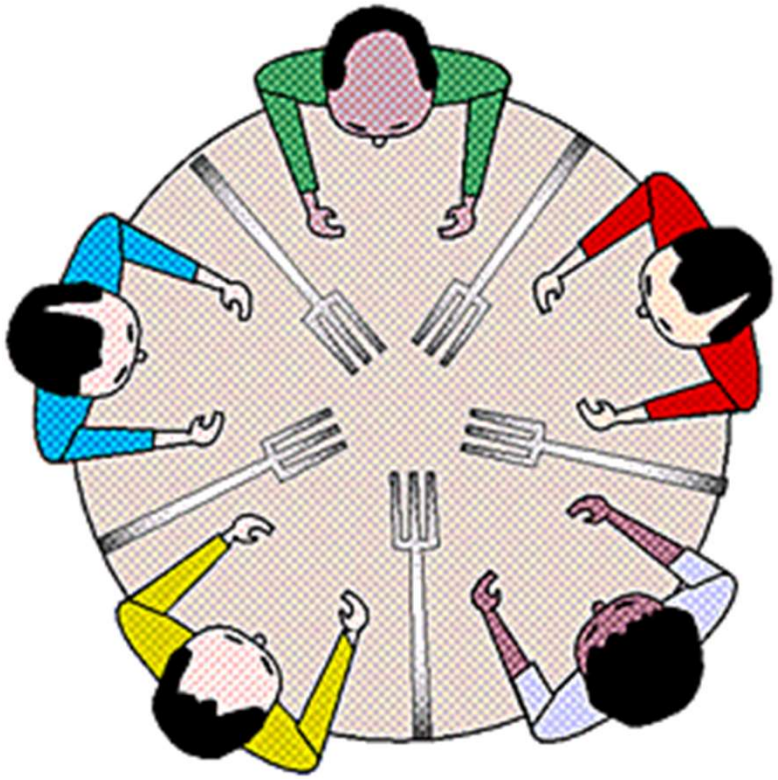
```
Void Philosopher (void)
{
        while (T) {
                Thinking ();
                p(s)
                wait(chopstick [i]);
                wait(chopstick [(i+1)%5]);
                v(s)
                Eat();
                signal(chopstick [i]);
                signal(chopstick [(i+1)%5]);


        }
}
```



A FEW PROBABLE SOLUTIONS (CONT....)

- Reverse the chopstick picking algo for any philosopher
- Any more solution ?



MONITORS

Abstract datatype

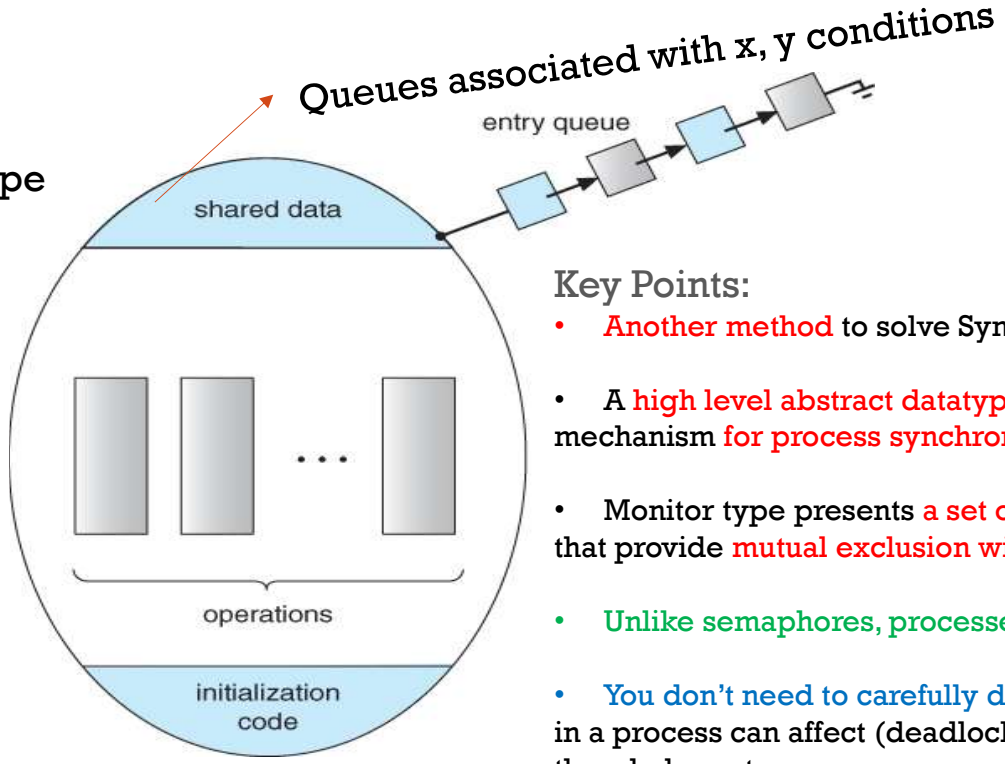
```
Monitor Demo //Name of Monitor
{
variables;
condition variables;

procedure p1 {....}
prodecure p2 {....}

Initialization Code {....}

}
```

Syntax of Monitor



High Level Schematic of a Monitor

***** Condition variable must be defined. Wait () and Signal () operations are performed on these variables.**

Condition x, y; // declaring a condition variable

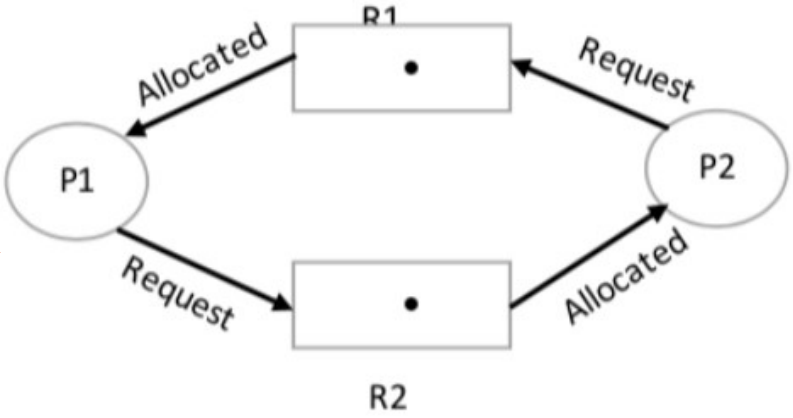
x.wait() means that the process invoking this operation is suspended until another process invokes x.signal()

Key Points:

- Another method to solve Synchronization problems
- A high level abstract datatype that provides a convenient mechanism for process synchronization
- Monitor type presents a set of programmer-defined operations that provide mutual exclusion within the monitor
- Unlike semaphores, processes are free here (light weight)
- You don't need to carefully design each process---> one mistake in a process can affect (deadlock or other timing issues) the whole system.
- Here, monitor will provide the mutual exclusion and others
- The processes will use the shared data through monitors
- Monitor construct ensures that only one process at a time can be active within the monitor

DEADLOCK

- Multiprogram → Resources limited → Wait state
- A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process.
- No process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.



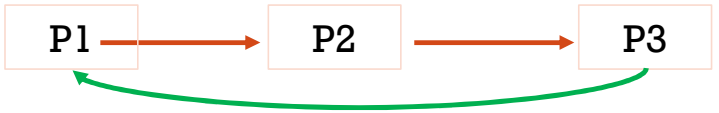
Resource Allocation Graph

System model (3 step process): Each process must follow this.

- The process requests some resources.
- OS grants the resource if available (Else process waits in block state).
- The process must release the resources on termination.

DEADLOCK (CONT...)

- ***Mutual Exclusion:*** Two or more resources are non-shareable (Only one process can use at a time)
- ***Hold and Wait:*** A process is holding at least one resource and waiting for resources.
- ***No Preemption:*** A resource cannot be taken from a process unless the process releases the resource.
- ***Circular Wait:*** A set of processes are waiting for each other in circular form.



Deadlock can arise if the **following four conditions hold simultaneously.**

DEADLOCK HANDLING METHODS

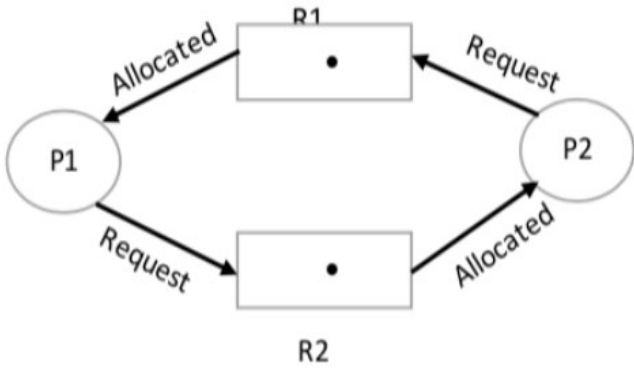
- Prevention: Design a system that violates at least one of the four necessary conditions of deadlock and ensures deadlock free system.
- Avoidance: The system maintains a set of data using which it makes a decision whether to entertain a new request or to be in a safe state.
- Detection and recovery: We will wait until deadlock, and once detected, we will recover the system from it.
- Ignorance: We will ignore the problem as if it does not exist.

PREVENTION APPROACH (AVOID AT LEAST ONE OF THEM)

- No mutual exclusion (depends on h/w property)

- No Hold and Wait
  - Conservative approach (less efficient, not implementable, easy, deadlock free)
  - Do not hold (acquires only desired resources, but before making new request it must release the resources it currently holds ---> efficient and implementable)
  - Wait timeouts (after a certain time quantum release the present holding resources)

- Forceful Preemption We can allow forceful preemption but
  - The method must be used by high priority processes or system processes.
  - Victimize the processes from waiting state not from the running state.



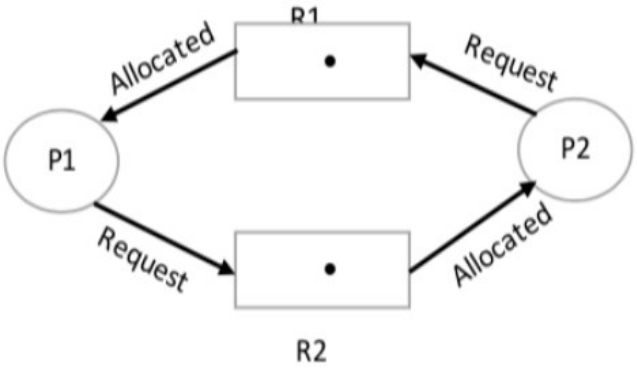
PREVENTION APPROACH (AVOID AT LEAST ONE OF THEM)

- Avoid Circular wait
  - Assign a natural number to every process (f: N ⟶ R).
  - Allow every process's request either only in the increasing or decreasing order of resource number.
  - If a process require a lesser number (in case of increasing order), then it must release all the resources larger than the required number.

P1	P2
R1	R2
R2	R1

⟶

P1	P2
R1	R1
R2	R2



AVOIDANCE (BANKERS ALGORITHM)

	Max Need		
	E	F	G
P_0	4	3	1
P_1	2	1	4
P_2	1	3	3
P_3	5	4	1

	Allocation		
	E	F	G
P_0	1	0	1
P_1	1	1	2
P_2	1	0	3
P_3	2	0	0

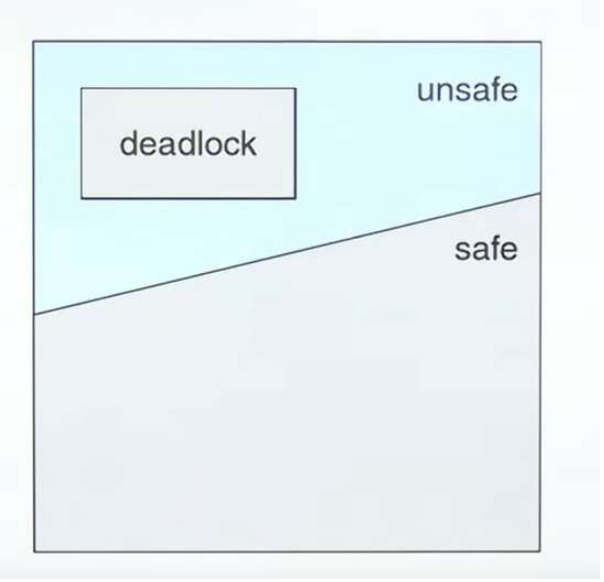
	Current Need		
	E	F	G
P_0	3	3	0
P_1	1	0	2
P_2	0	3	0
P_3	3	4	1

System Max		
E	F	G
8	4	6

Available		
E	F	G
3	3	0

Resource allocation state

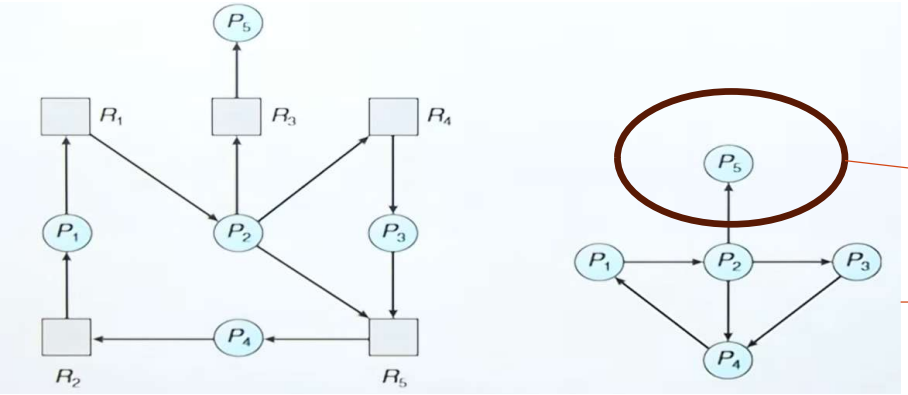
Find the safe sequence?

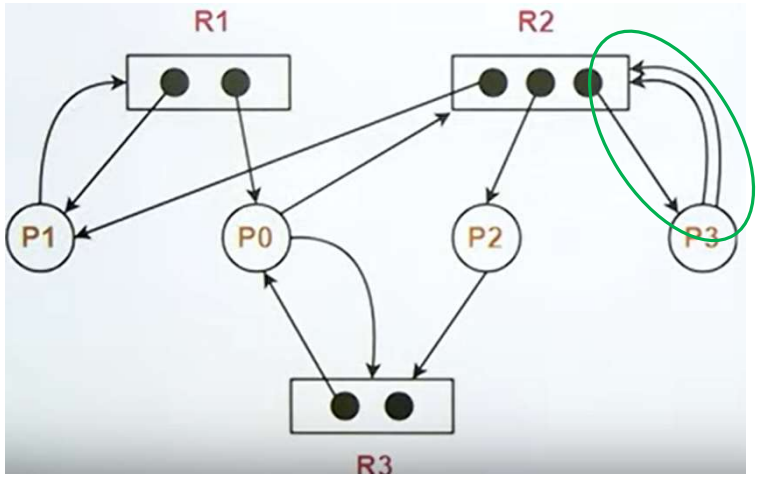


RESOURCE ALLOCATION GRAPH

- Deadlock can be represented by a directed graph (set of vertices V and Edges E)
- Vertices are partitioned into two different types of nodes:
  - P = {P1, P2, …., Pn}
  - R = {R1, R2,….., Rn}
- Avoidance technique for simple cases.

If there is only one instance of resources



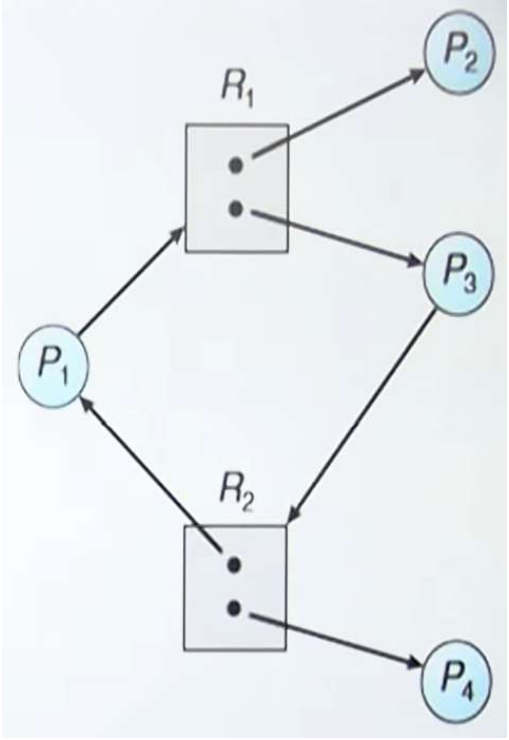


Current allocation, Max Need, Current Need??

This can run easily

Cycle, here, is necessary and sufficient condition for deadlock. ---> as 1 instance of each resources

BUT FOR MULTIPLE INSTANCES OF RESOURCES??



Cycle, here, is necessary but not sufficient condition for deadlock

DEADLOCK DETECTION AND RECOVERY

- Here we do not check safety. We can allocate the process immediately, if available.
- The possibility of deadlock must detected using two different approaches.
  - Active Approach
  - Lazy Approach

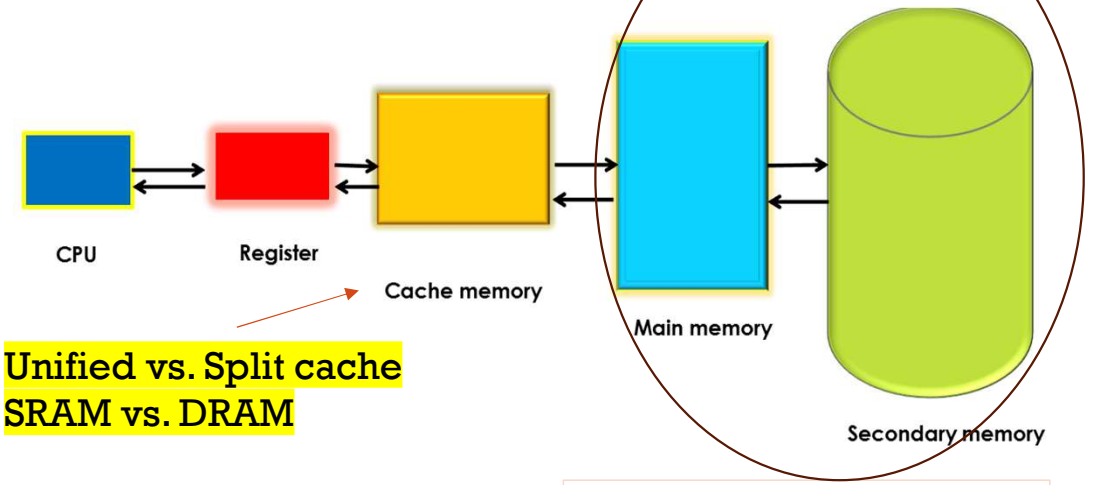
MEMORY MANAGEMENT

OS plays a huge role here

Zero Cost
Zero Latency
Infinite Bandwidth
Infinite capacity

Parameter	SRAM	DRAM
Full Form	SRAM stands for Static Random Access Memory.	DRAM stands for Dynamic Random Access Memory.
Component	SRAM stores information with the help of transistors.	DRAM stores data using capacitors.
Need to Refresh	In SRAM, capacitors are not used which means refresh is not needed.	In DRAM, contents of a capacitor need to be refreshed periodically.
Speed	SRAM provides faster speed of data read/write.	DRAM provides slower speed of data read/write.
Power Consumption	SRAM consumes more power.	DRAM consumes less power.
Data Life	SRAM has long data life.	DRAM has short data life.
Cost	SRAM are expensive.	DRAM are less expensive.
Density	SRAM is a low density device.	DRAM is a high density device.
Usage	SRAMs are used as cache memory in computer and other computing devices.	DRAMs are used as main memory in computer systems.

This layered architecture is made based on the locality of references.



Unified vs. Split cache
SRAM vs. DRAM

What is AMAT?

OS deals with important tasks regarding memory management:

- Space allocation (Contiguous, Non Contiguous)
- Address translation (Logical address to Physical Address)

MEMORY ALLOCATION

- Contiguous memory allocation (contiguous segment of memory is allocated)

External Fragmentation
(Major Problem)

- Non-contiguous memory allocation (different parts of a process are allocated to different places in Main Memory).

- Array vs. linked list (example analogy)
- Address translation easy in CA (only base + index offset)
- External fragmentation is a serious issue in CA

CONTIGUOUS MM ALLOCATION

- Fixed size partition scheme
  - Internal fragmentation
  - External fragmentation
- Variable size partition scheme
  - No Internal fragmentation
  - External fragmentation

FIRST FIT, BEST FIT, AND WORST FIT ALLOCATION POLICY FOR VARIABLE PARTITIONING

P1 = 300
P2 = 25
P3 = 125
P4 = 50

Calculate the external fragmentations?

Worst fit can perform better than best fit for variable size partitioning… think why?

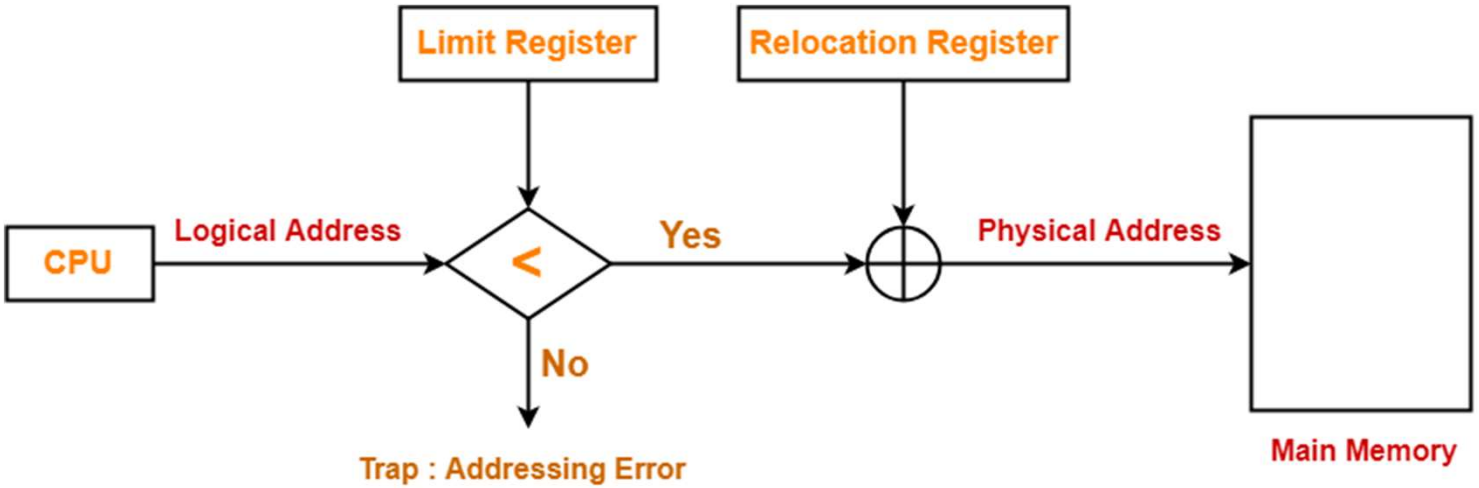
FIRST FIT, BEST FIT, AND WORST FIT ALLOCATION POLICY FOR FIXED PARTITIONING

P1 = 357
P2 = 210
P3 = 468
P4 = 491

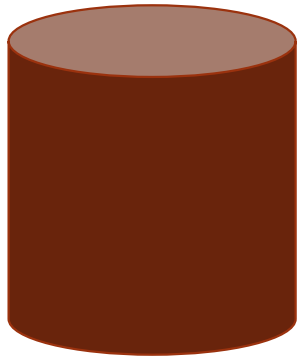
200	400	600	500	300	250

Calculate the internal and external fragmentations?

ADDRESS TRANSLATION IN CONTIGUOUS MEMORY ALLOCATION

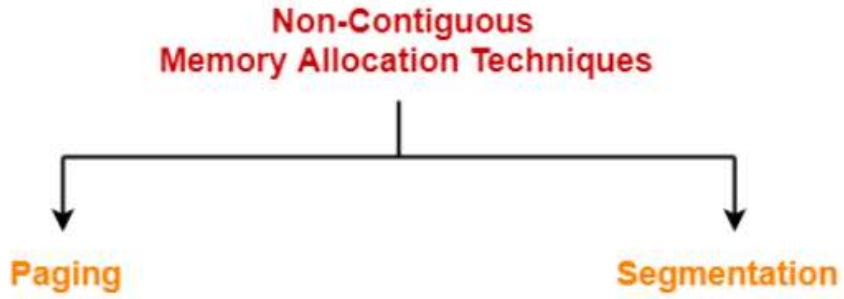


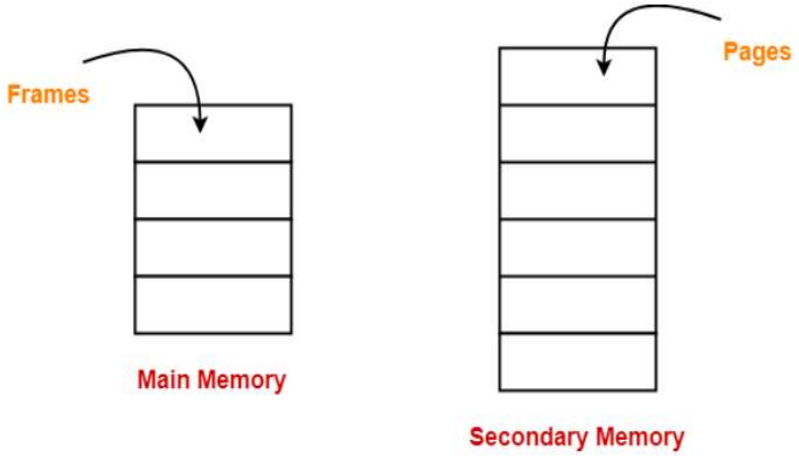
Translating Logical Address into Physical Address



Secondary Memory

NON-CONTIGUOUS MEMORY ALLOCATION



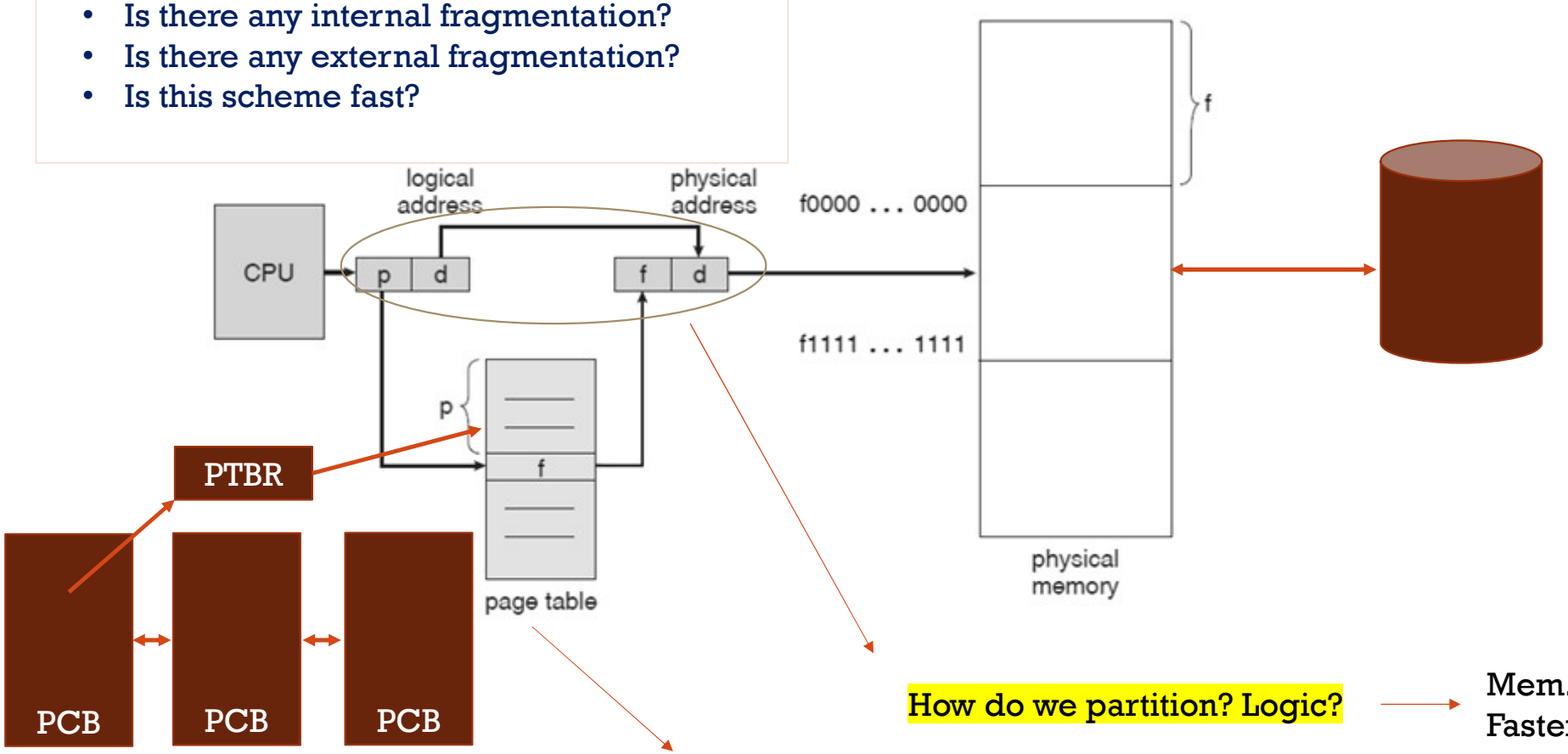


Paging-

- Paging is a fixed size partitioning scheme (Internal Fragmentation but no External Fragmentation).
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as **pages**.
- The partitions of main memory are called as **frames**.

PAGING

- Can we Follow pointer based approach?
- Is there any internal fragmentation?
- Is there any external fragmentation?
- Is this scheme fast?

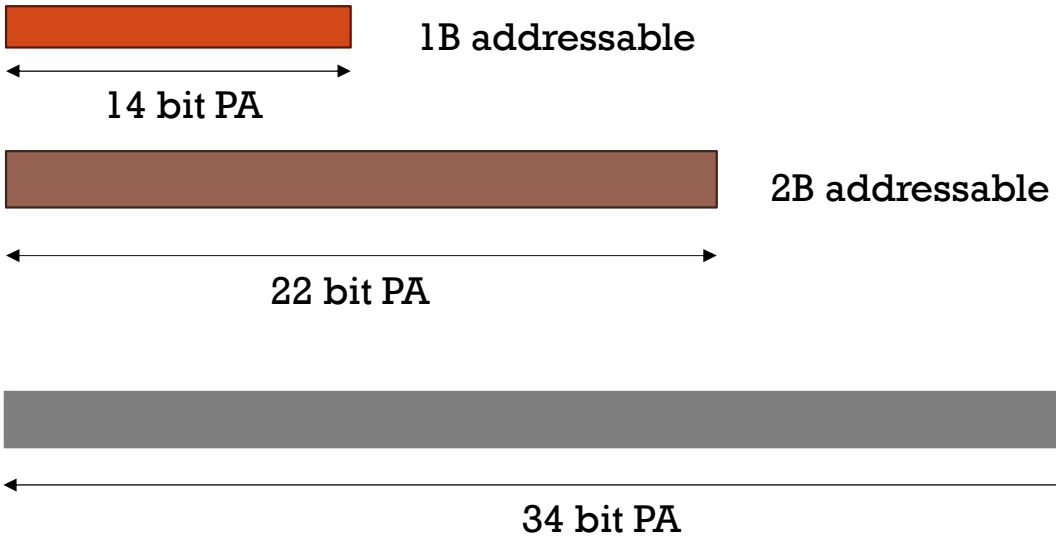


How do we partition? Logic? → Mem. Access is Faster.

Every process has a separate page table

PAGING (CONT...)

- Calculate the size of the following memory.



Trick for quick calculation

2^{10} ----------> K

2^{20}----------> M

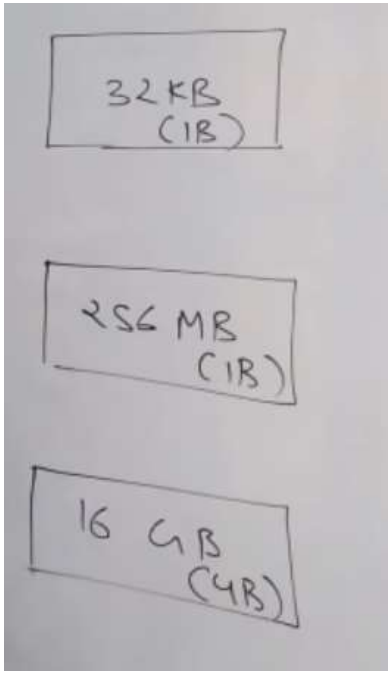
2^{30}----------> G

2^{40}----------> T

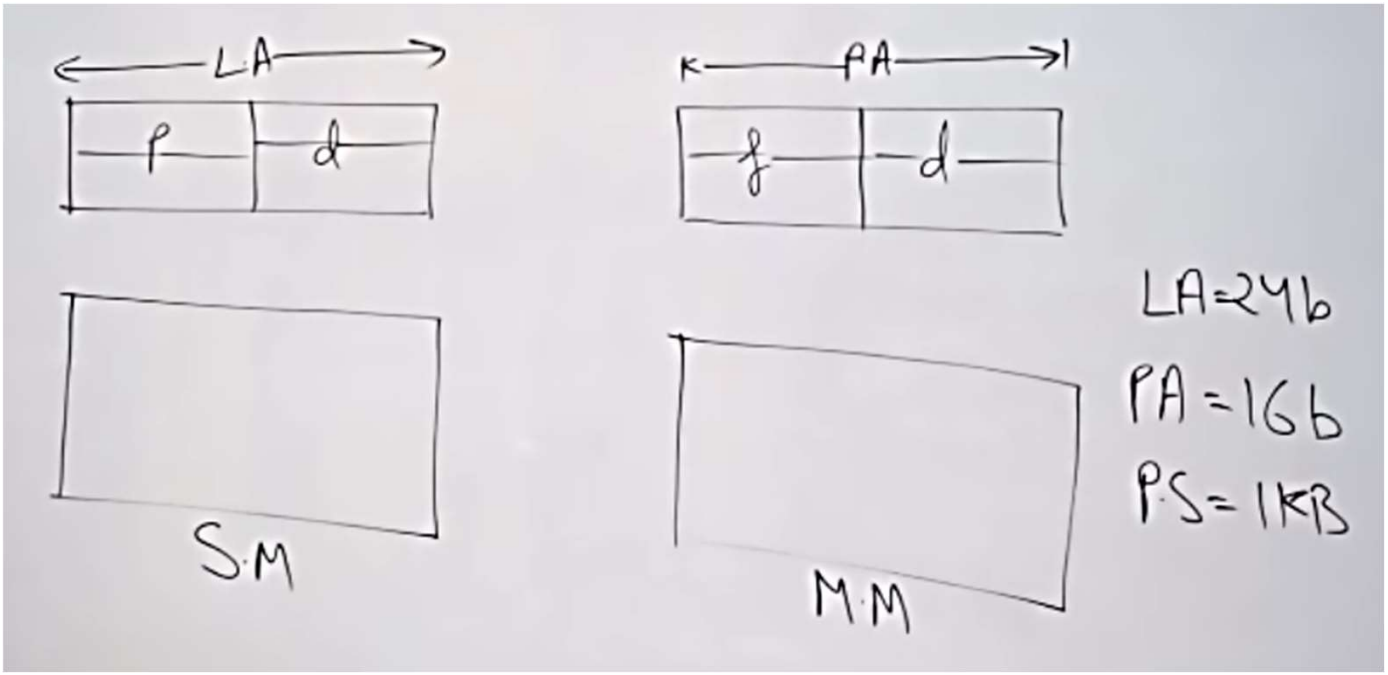
2^{50}----------> P

PAGING (CONT...)

Calculate the number of bits required for the PA.



PAGING (CONT...)



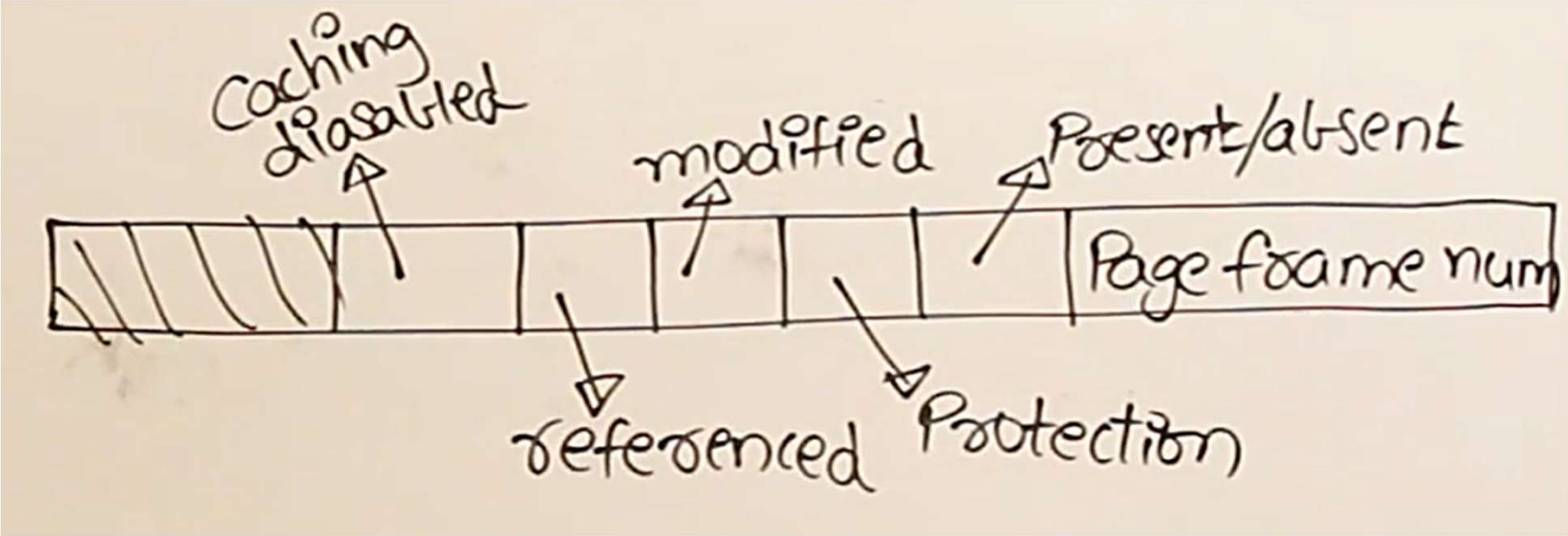
# of pages?
# of frames?

SOLVE THIS

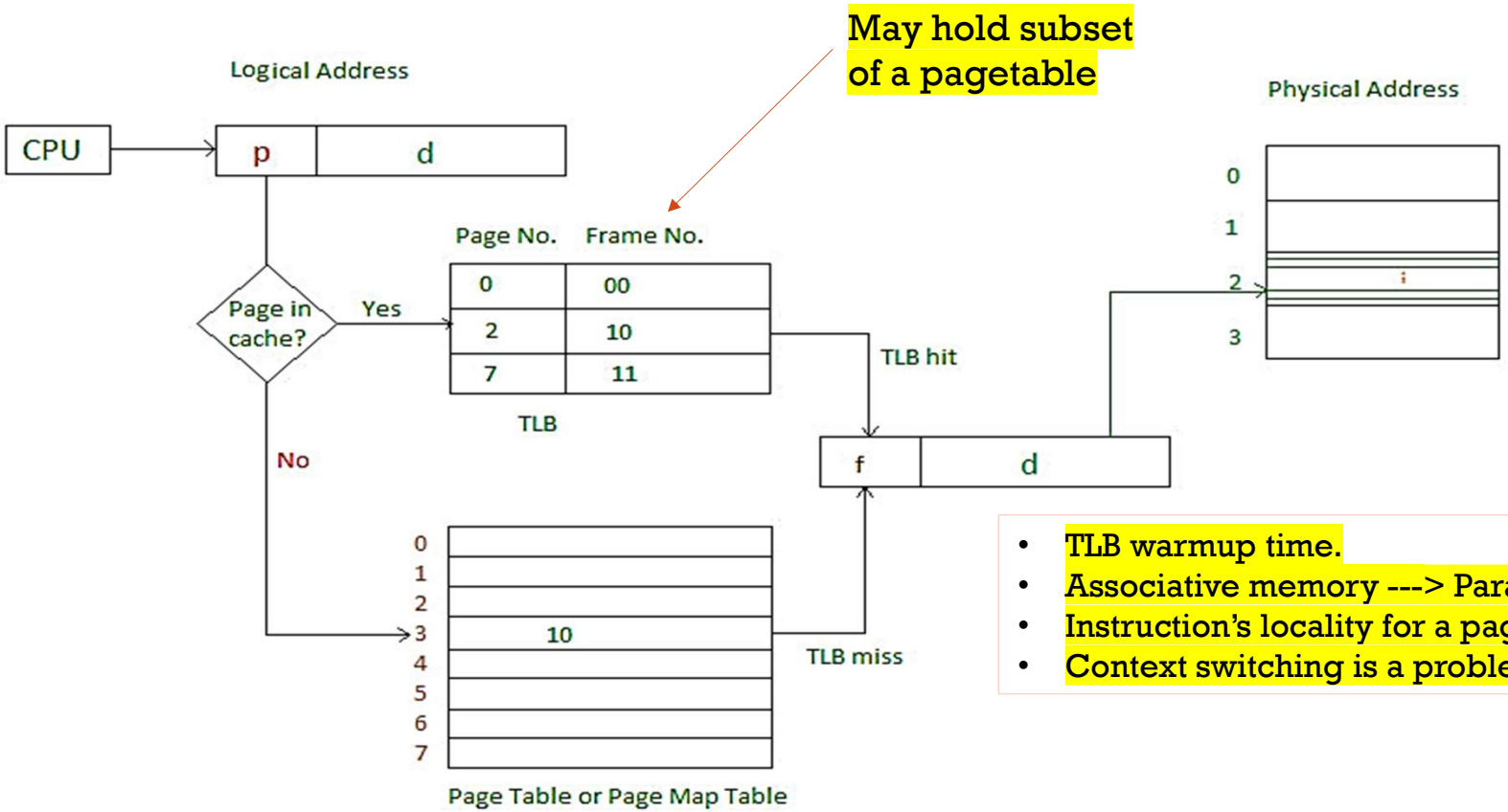
LAS	PAS	LA	PA	Page size	Page offset	Pages	frames	PTE	PTS
128 KB	128 KB			4 KB					
256 KB	1 MB			4 KB				2 B	

Each word is 1B and memory is byte addressable.
LAS = # of pages * page size || LA – pageoffset then find # of pages (both is fine)

SOME ADDITIONAL INFORMATION FOR PTE



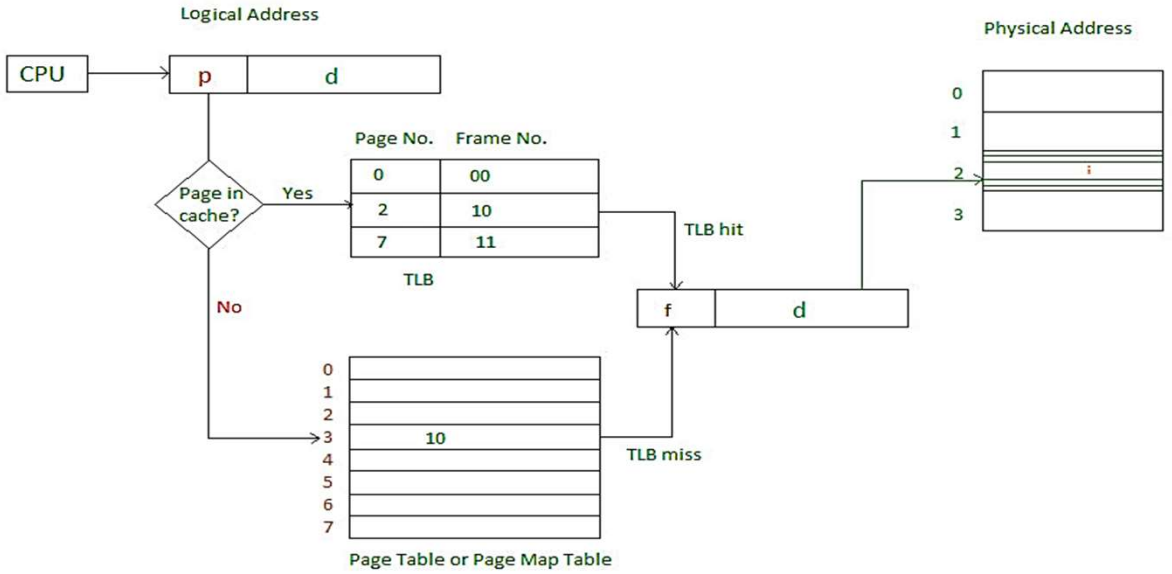
TRANSLATION LOOKASIDE BUFFER (TLB)



- TLB warmup time.
- Associative memory ---> Parallel Searching.
- Instruction's locality for a page is used.
- Context switching is a problem.

TLB (CONT…)

- Avg mem access time : h [TLB + MM] + (1-h) [TLB + MM +MM]
- Find avg. mem access time for the following config.
  - MM = 400 microsec
  - TLB = 50 microsec
  - Hit percentage = 90%
  - 400---------->800---------->???????



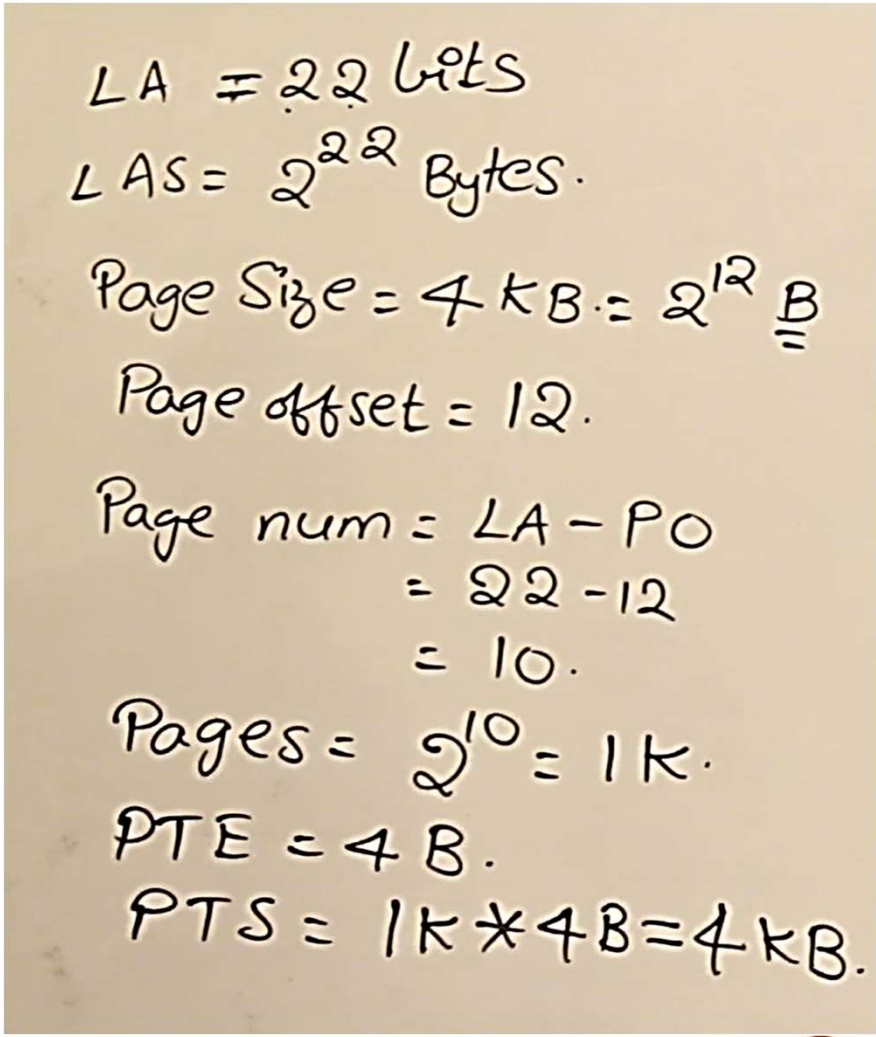
TLB (CONT…) DISADVANTAGES

- Too much Context Switches ???
  - More TLBs??
  - Wired-down entries (reserving some area in TLBs for the below data)
    - Critical Kernel Data
    - Real-Time or High-Performance Systems (lock specific TLB entries to ensure that certain memory translations are always available)
    - Security Considerations
    - Interrupt handling routines.

Not exposed to user-level process

MULTILEVEL PAGING

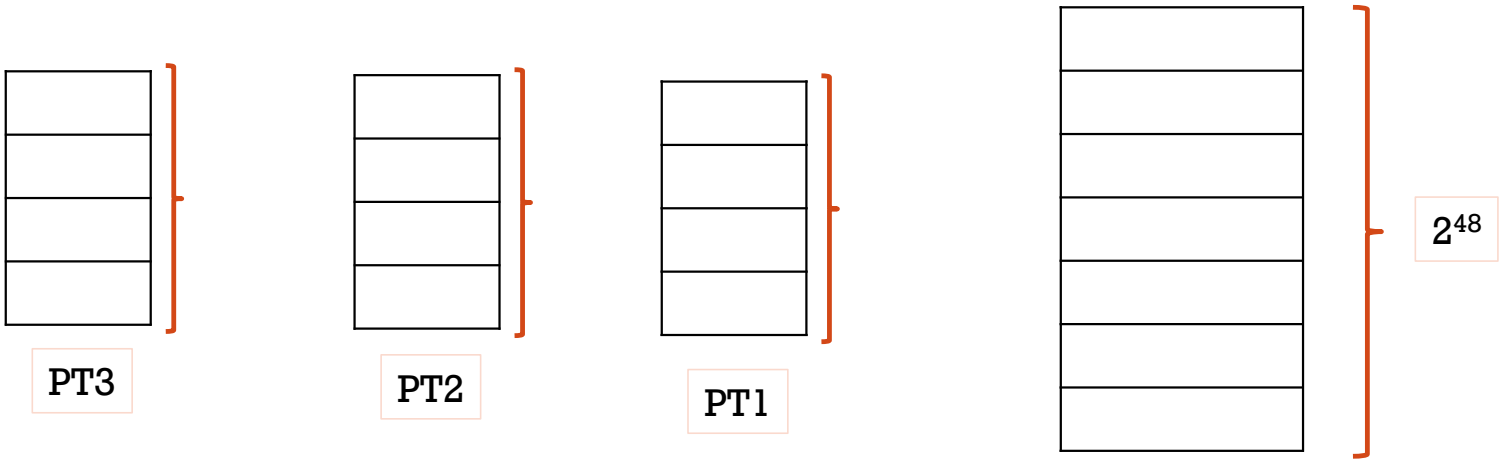
- Page table must be there in the main memory.
- Frame size is 4KB.
- If PTS is more than 4KB? May be because of Larger PTE. Then??



MULTI-LEVEL PAGING

Page Table Size > Frame Size

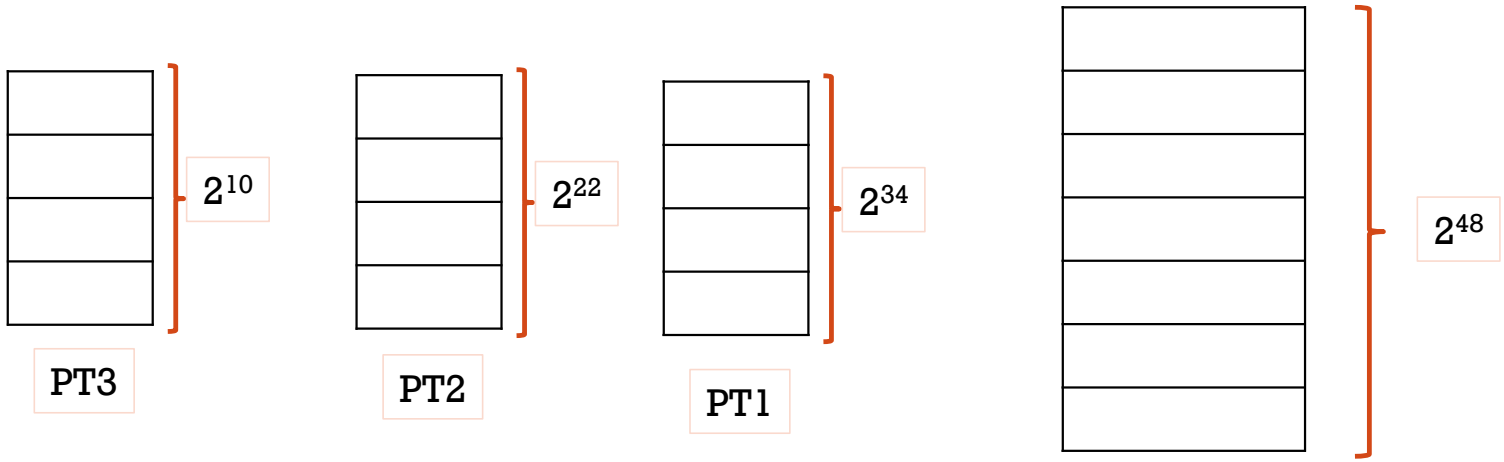
VA	Page Size	PTE	PT1	PT2	PT3	Address bit
48 bits	16 KB	4B				



MULTI-LEVEL PAGING

Page Table Size > Frame Size

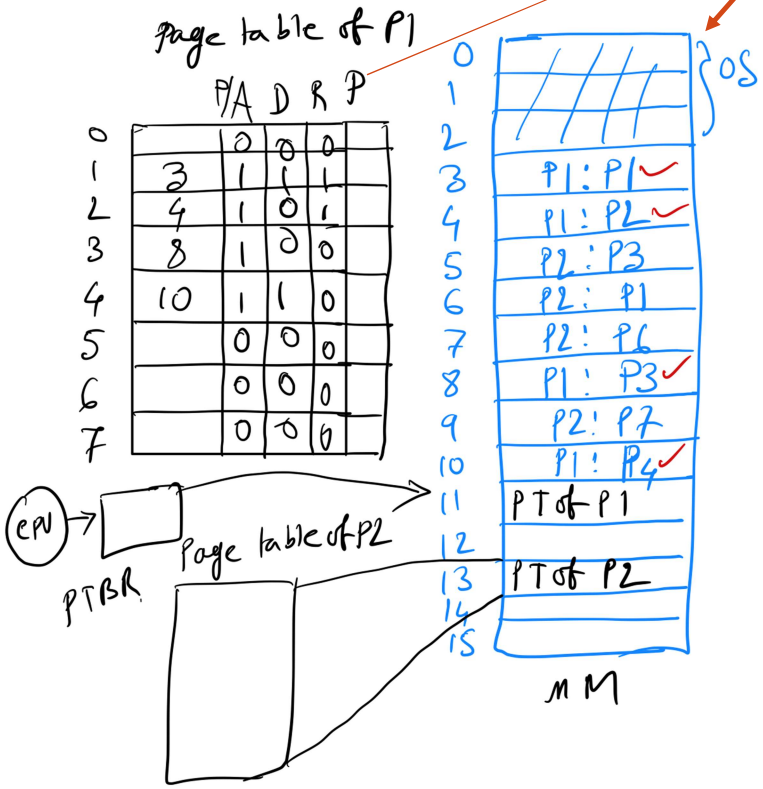
VA	Page Size	PTE	PT1	PT2	PT3	Address bit
48 bits	16 KB	4B	2^{34} X 2^{2} = 2^{36} B	2^{22} X 2^{2} = 2^{24} B	2^{10} X 2^{2} = 2^{12} B	10, 12, 12, 14



VIRTUAL MEMORY

Loader is also there

Think about code/data section of a process



- Increases degree of multiprogramming
- Gives an illusion of having larger MM than its physical Limit
- Demand paging (Do not load any page until required)
- Page fault---> page fault service
- Time -->starts from the same Instruction that caused the PF.
- Too many pagefaults causes Thrashing.
- EMAT = P * (pagefault service time +ma) + (1-p) ma

Demand paging: Do not load any page until it is required. It creates some mandatory page Faults.

Try to think TLB hit and page fault together

BEHIND A PAGE FAULT

1. **Page Fault Triggering**

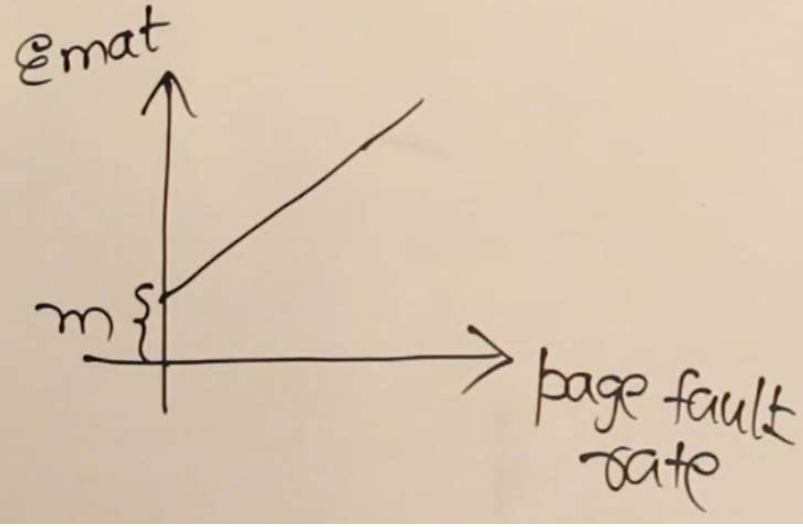
2. **Exception Handling:** CPU raises a page fault exception, causing the operating system's kernel to take control.

3. **Handling by the Operating System:** Initiates the page fault handler routine.

4. **Lookup:** Location of the required page in secondary storage.

5. **Page Retrieval:** The OS schedules a page-in operation to bring the required page into physical memory (RAM). This involves loading the page from disk into an available page frame in RAM.
6. **Page Replacement (if necessary)**
7. **Updating Page Table**
8. **Resuming Process:** The instruction that caused the page fault is restarted.
9. **Return from Exception:** The operating system hands back control to the application, which can continue its execution with the requested page now in memory.

$$Emat = p(PS+m)+(1-p)(m)$$
$$= p(PS)+\cancel{pm}+m-\cancel{pm}$$
$$Emat = (PS)p + m$$
$$y = mx + c$$



Let the page fault service time be 10ms in a computer with average memory access time being 20ns. If one page fault is generated for every 10^6 memory accesses, what is the effective access time for the memory?

A) 21 ns B) 30 ns C) 23 ns

D) 35 ns

Average time taken to access VA

$= (VA \rightarrow PA) + (\text{fetch the word from Process})$

$= t + (1 - h_t)(k * m) +$

$c + (1 - h_c)(m)$

) A processor uses 2 level paging. VA = PA = 32 bits. Byte addressable

VA

10	10	12

. PTE = 4 Bytes.

TLB has hit rate of 96%.

Cache has hit rate of 90%.

main memory access time is 10ns,

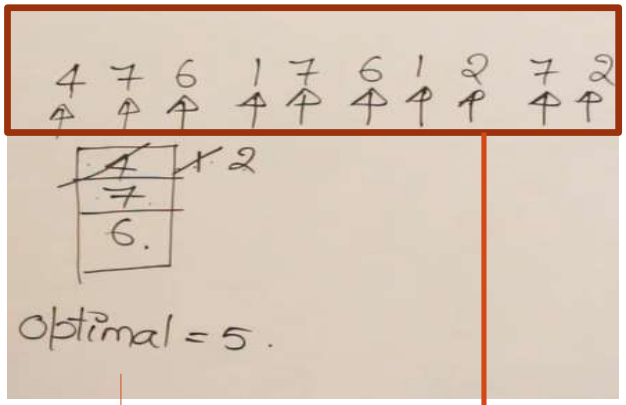
Cache access time is 1ns and

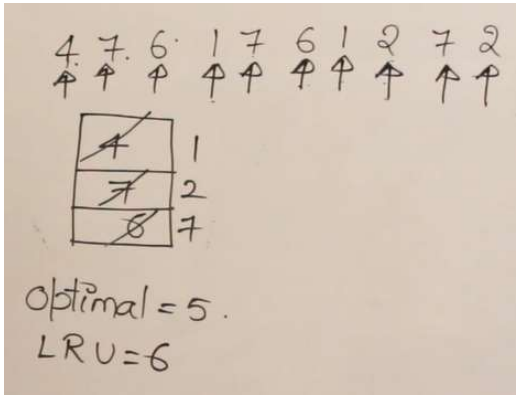
TLB access time is 1ns.

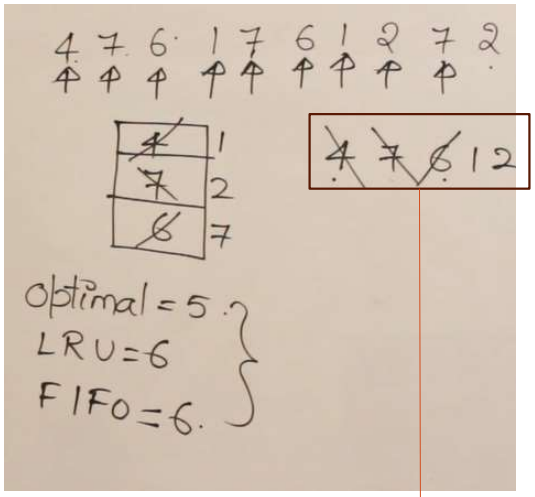
a) Assuming no page faults, the average time taken to access a VA is approximately (to the nearest 0.5ns)

a) 1.5 ns b) 2 ns c) 3 ns d) 4 ns

PAGE REPLACEMENT (OPTIMAL, LRU, AND FIFO)







Reference String: page numbers

Maintain a queue

Not Practical to implement, Used for benchmarking

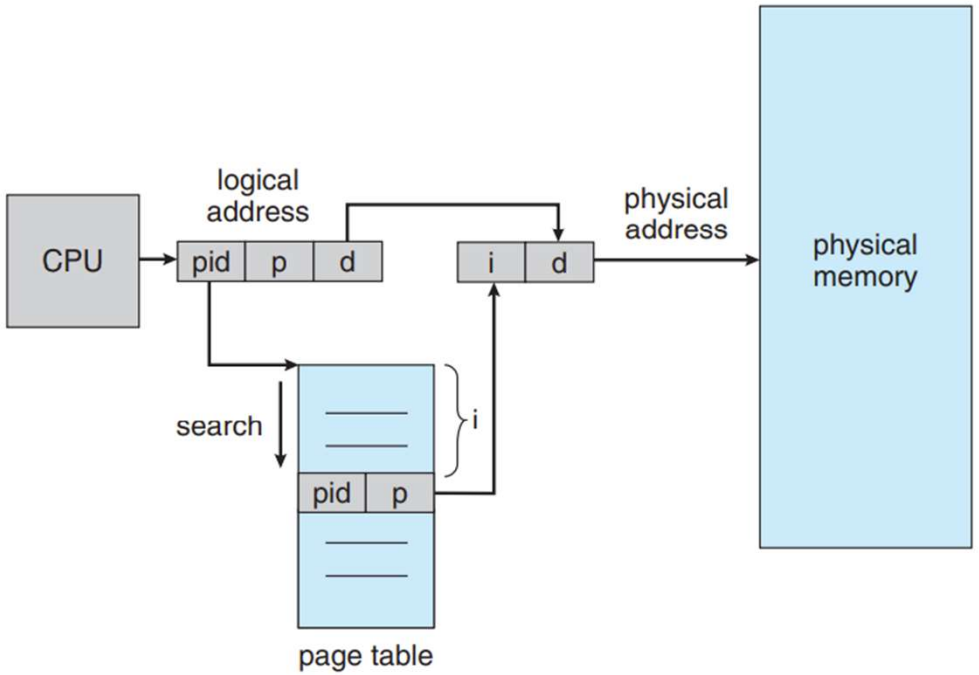
FIND THE PAGE FAULTS FOR OPTIMAL, LRU AND FIFO

Reference Sting: 0 1 2 3 0 1 4 0 1 2 3 4

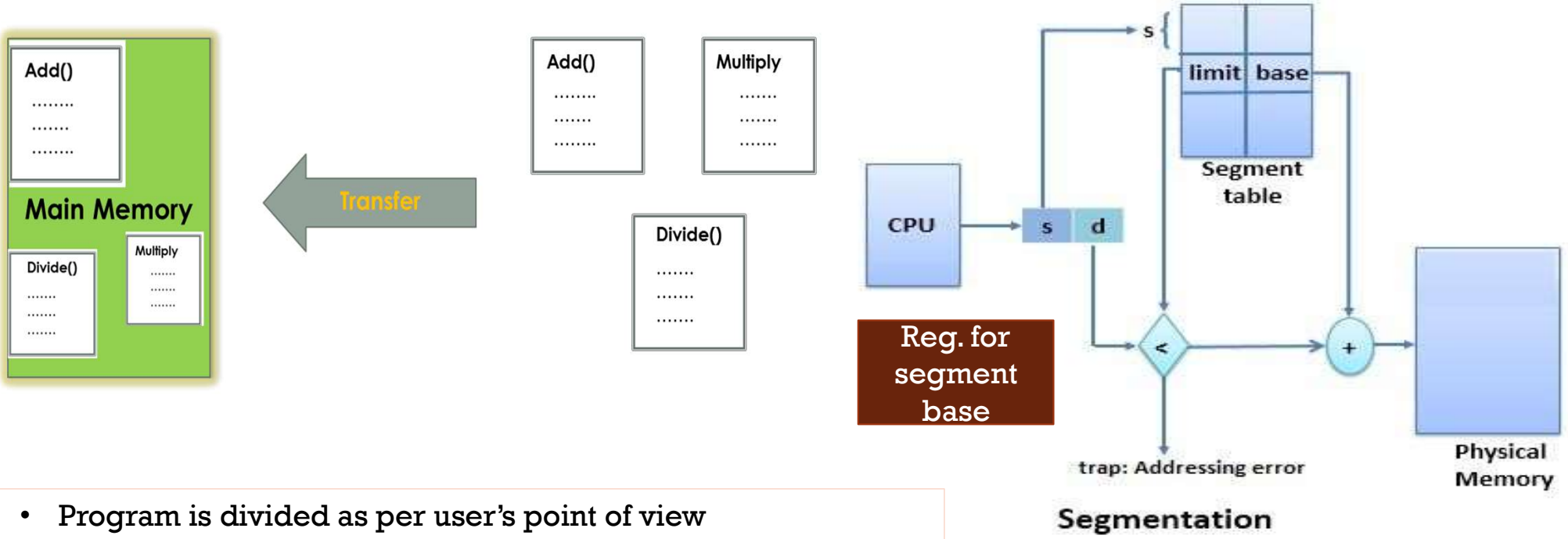
With # of Frames 3 and 4

Belady's anamoly.

INVERTED PAGE TABLE



SEGMENTATION



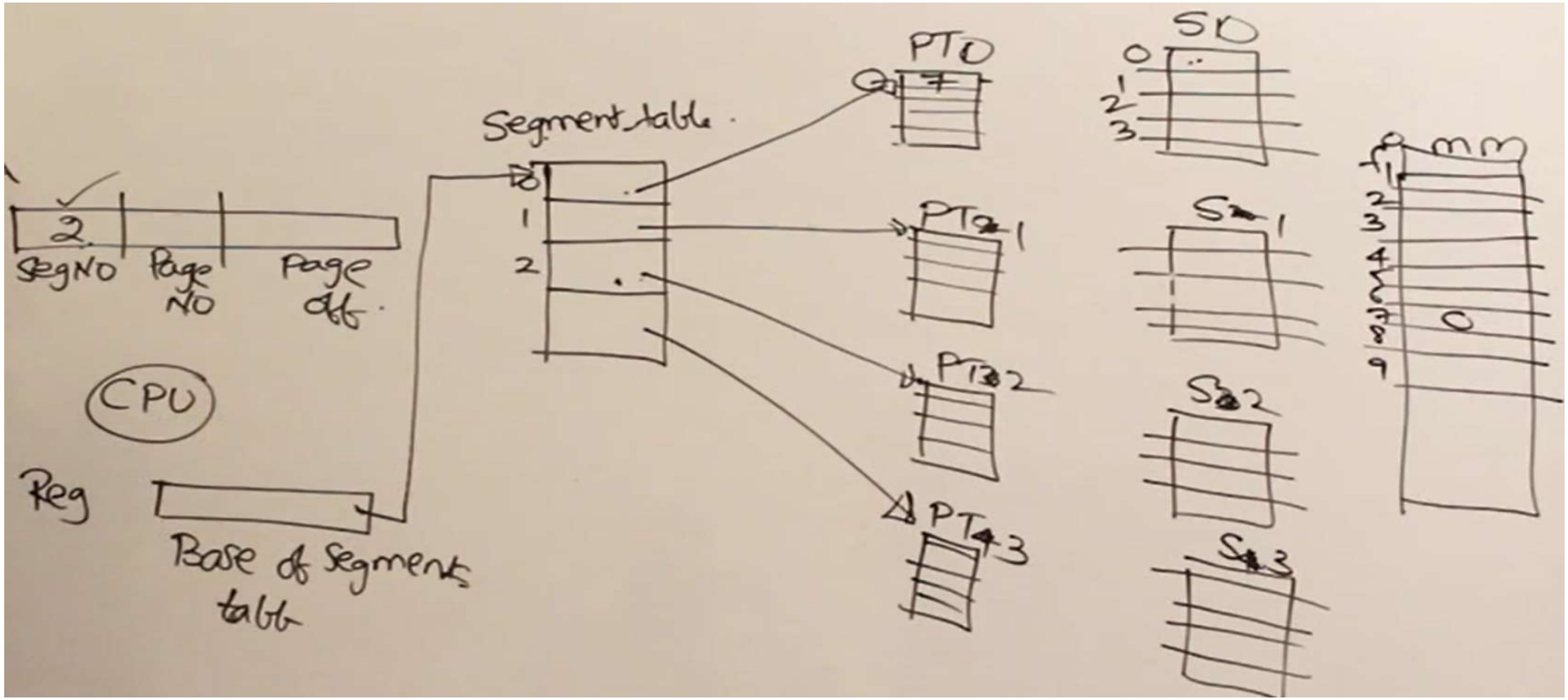
- Program is divided as per user's point of view
- Like page table, it has segment table which hold base + limit

A FEW ADVANTAGES/DISADVANTAGES OF SEGMENTATION

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.
- As a complete module is loaded all at once, segmentation improves CPU utilization.
- The user specifies the segment size, whereas, in paging, the hardware determines the page size (more flexibility).

- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.
- Due to the need for two memory accesses, one for the segment table and the other for main memory, access time to retrieve the instruction increases.

SEGMENTATION WITH PAGING



PAS = LAS = 2^{16} B

Byte addressable.

VAS is divided into 8 non-overlapping equal size segments.

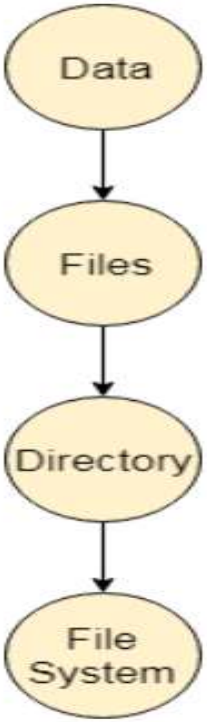
The MMU has a segment table, each entry of which contains PA of the PT for the segment.

PT are stored in the main memory and consist of 2 byte PTE.

a) what is the minimum page size in bytes so that the page table for a segment requires at most one page to store it. Assume that page size is a power of '2'.

FILE

- ✓ A file can be defined as a data structure which stores the sequence of records.
- ✓ Files are managed by the operating system to organize and store data in a structured manner.
- ✓ A directory, also known as a folder, is a container or organizational unit used to store and organize files and other directories.
  - ✓ *More specifically, A directory is a unique type of file that contains only the information needed to access files or other directories.*
- ✓ FS provides a way to store, retrieve, and organize data in a hierarchical and efficient manner.



ATTRIBUTES OF THE FILE (META-DATA)

- Name: Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.
- Identifier: Along with the name, Each File has its own extension which identifies the type of the file.
- Type: In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.
- Location: Each file carries its location as its attribute.
- Size: By size of the file, we mean the number of bytes acquired by the file in the memory.
- Protection: Each file carries its own set of permissions to the different group of Users.
- Time and Date: Every file carries a time stamp which contains the time and date on which the file is last modified.

OPERATIONS ON FILE

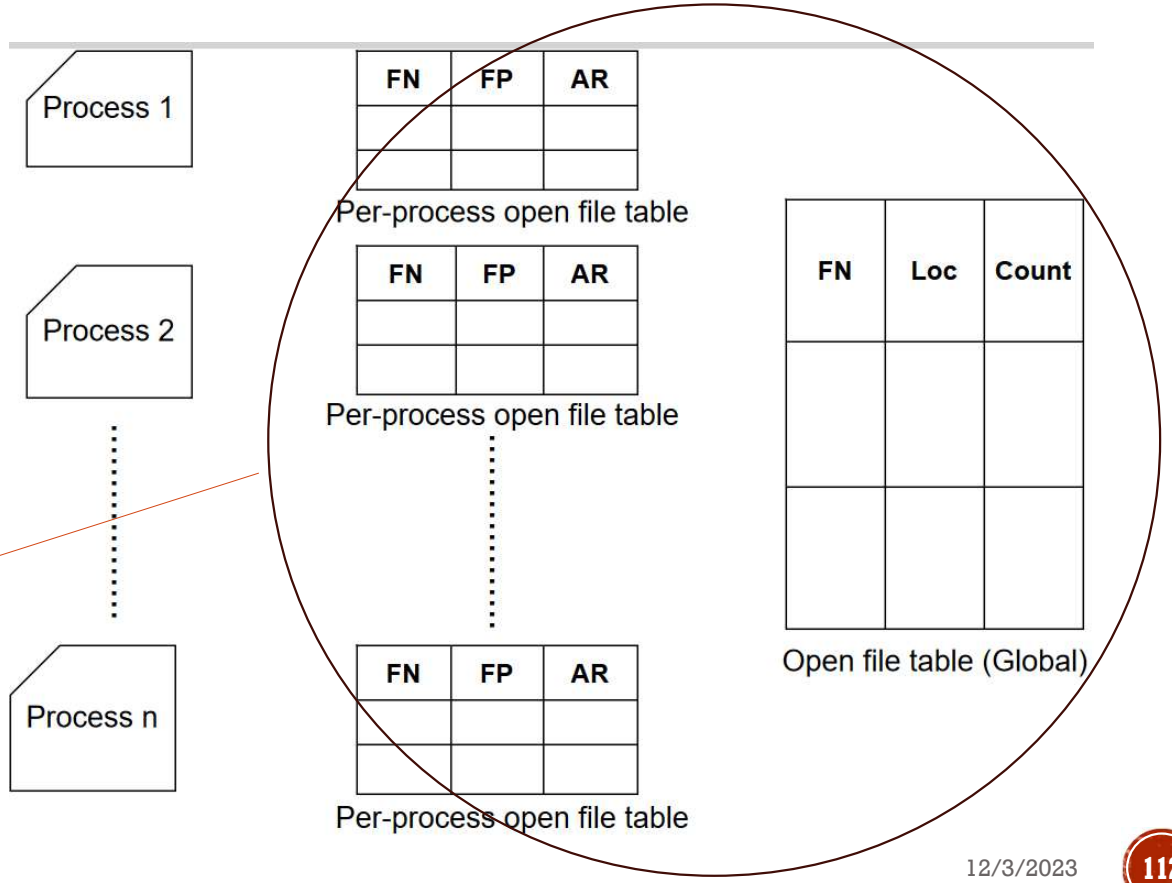
- Creating: Creating a file involves two steps. First, check if there is free space. You must create a new directory file entry if you have enough space.
- Open: After you create a file, you need to open it before you can perform any file processing operations.
- Reading: Read Pointer
- Writing: Write Pointer
- Deleting: Deleting all
- Truncating: Delete data but not attributes
- Close: Once the program has finished using the file, it needs to close the file (Free up the resources)

OPEN FILE (HOW TO ACCESS A FILE)

Information required to access a file:

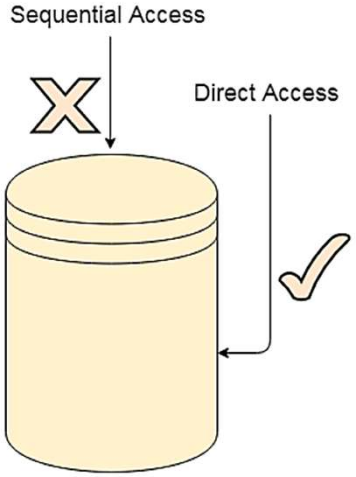
1. File pointer
2. File open count
3. Disk Location of a file
4. Access Rights

Where will they be stored?

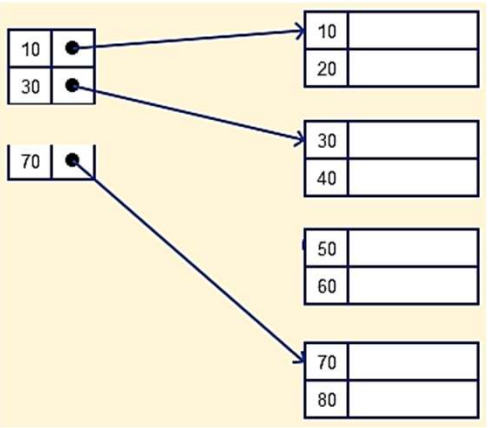


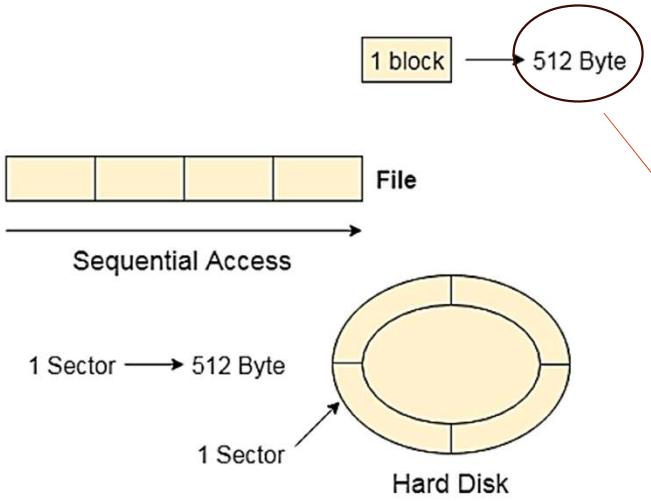
FILE ACCESS METHODS

- Sequential Access
- Direct Access
- Indexed Access



Logical Block Number is calculated directly





Direct Access: Fixed number of records are kept in blocks and the block number can be found directly.

Why 512 B?---> tradeoff between internal fragmentation and access speed

Indexed Access:
If a **file can be sorted** on any of the filed then an index can be assigned to a group of certain records.
(It could be multi-level)

OS DIRECTORY STRUCTURE

- Partition / volume / Minidisk

A directory is a unique type of file that contains only the information needed to access files or other directories (Directory Entries: Name, Loc pointer / Block Number, Size……etc.).

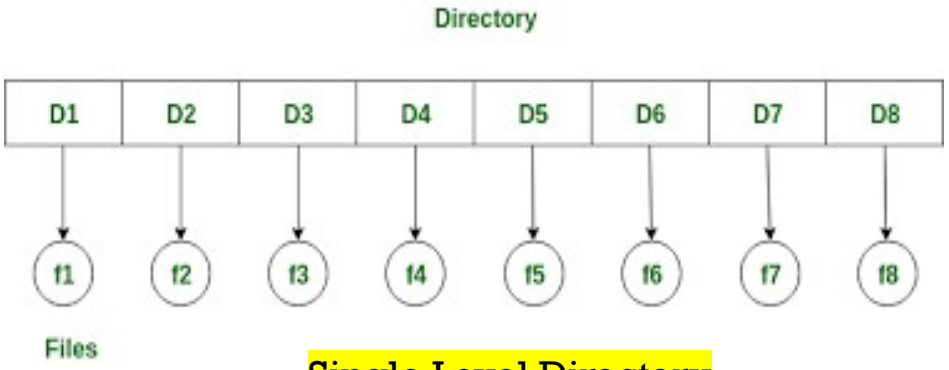
- Operation that a directory should support:
  - Search
  - Create
  - Delete
  - List
  - Traverse
  - Rename
- If the directory become too big??
  - Single Level
  - Two Level
  - Tree structure
  - Acyclic Graph
  - Graph

Directory Entries

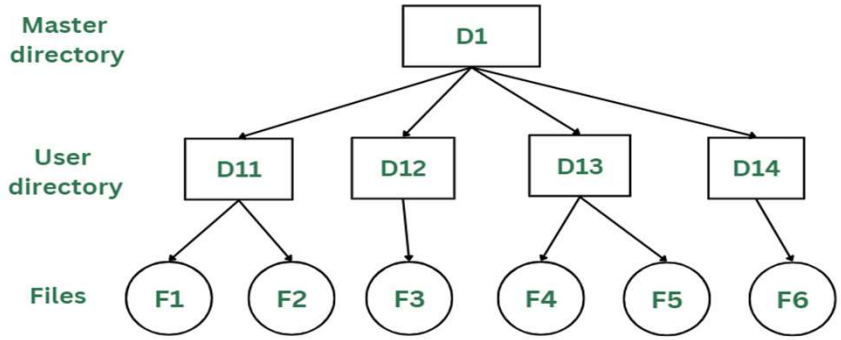
Name	Loc pointer / Block Number	Size

Directory can be assumed as a translation table to locate data

SINGLE VS TWO LEVEL DIRECTORY



Single Level Directory



Two Level Directory

All users will have one directory.

- Easy to implement
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.
- For smaller file sizes and directory size, searching can also be easy.

- Collision of Names.
- Problem for multiple users
- Searching can be problem for larger directory and files

……

Directory is created for every users.

- There can be more than two files with same name
- Prevents users to access other user's files
- Searching of the files becomes easy

- May become challenging to effectively manage a large number of files
- Placing the system files???

……..

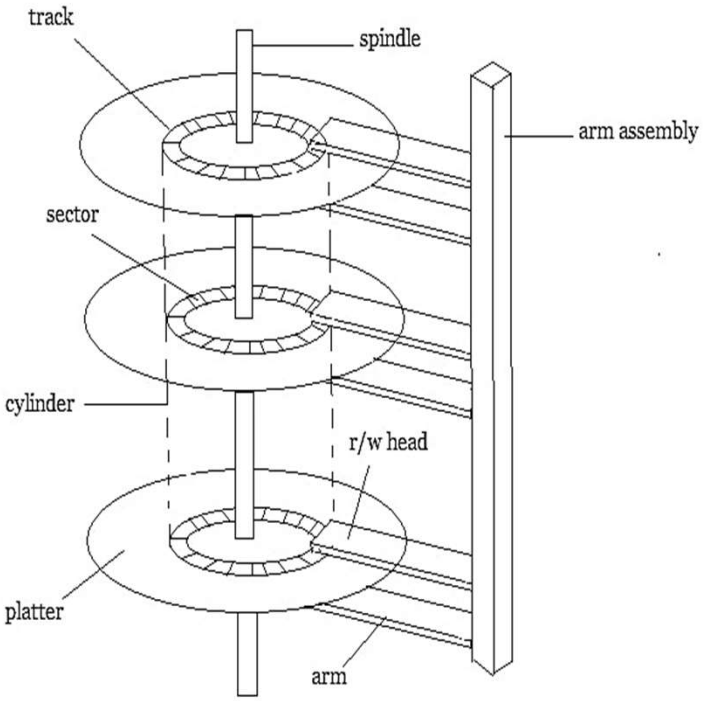
DISK: PART OF COMPUTER OR I/O?

- One way of implementing the secondary memory is disk.
- Each platter has a flat circular shape like a CD or DVD.
- Diameter could range from 1.8 to 5.25 inches
- Both surface have magnetic material.
- R/W head is attached for both surface.
- Surface of a disk is divided into circular tracks.
- Tracks are further divided into sectors.

Seek time: Time required to move r/w head on the desired track.

Rotational Latency: Time required to position a specific sector under the r/w head.

Disk Scheduling: For multiple I/O requests, disk scheduling algo must decide which request must be executed first.



Disk Pack Structure

FCFS

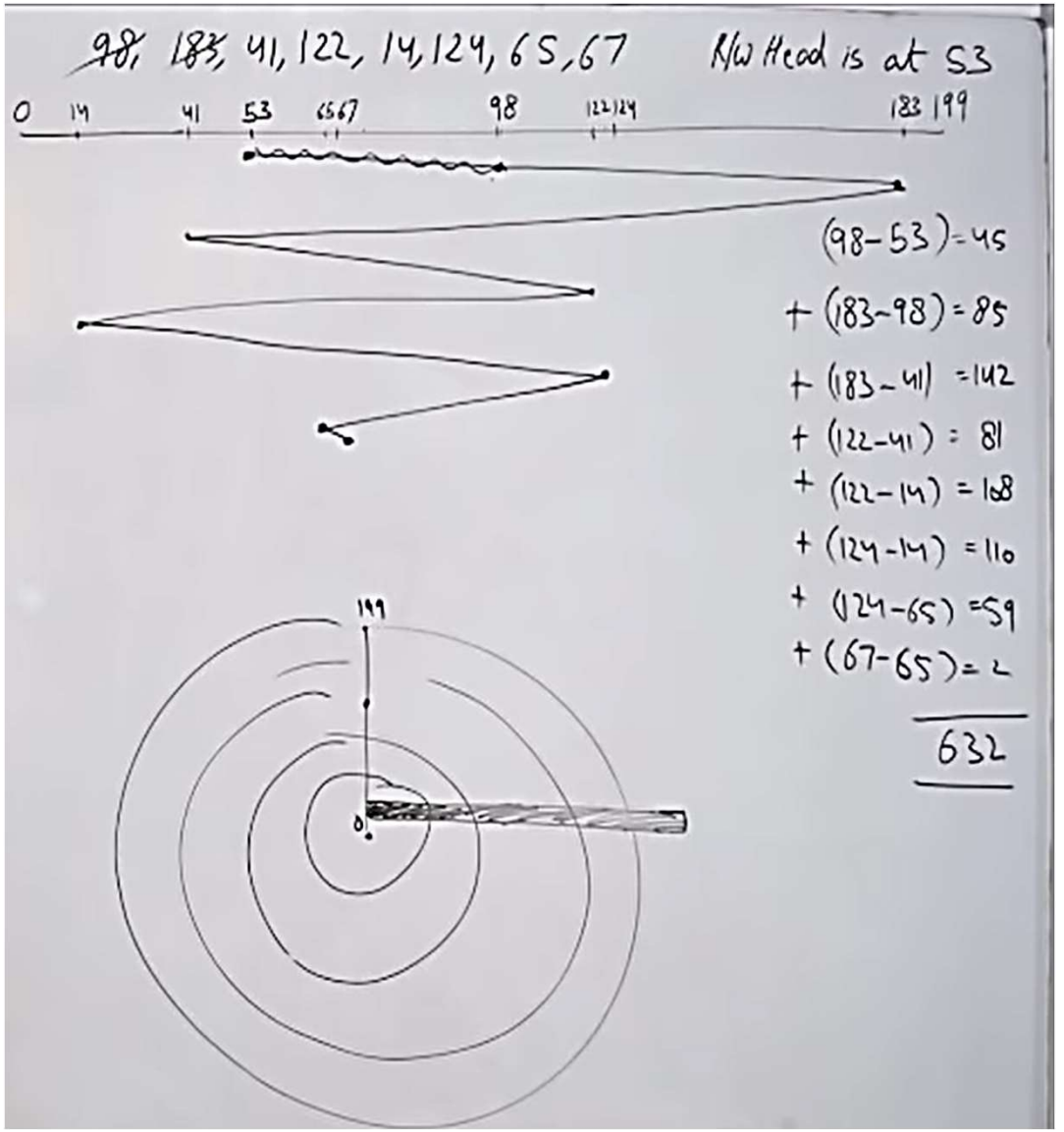
- Simplest disk Scheduling Algo
- Requests are entertained in the order they arrived in the disk queue

Advantages:

- Easy to understand and implement
- No starvation (may suffer from convoy effect)
- Can be used with less load

Disadvantages:

- Requires more seek time
- More waiting and response time
- Inefficient

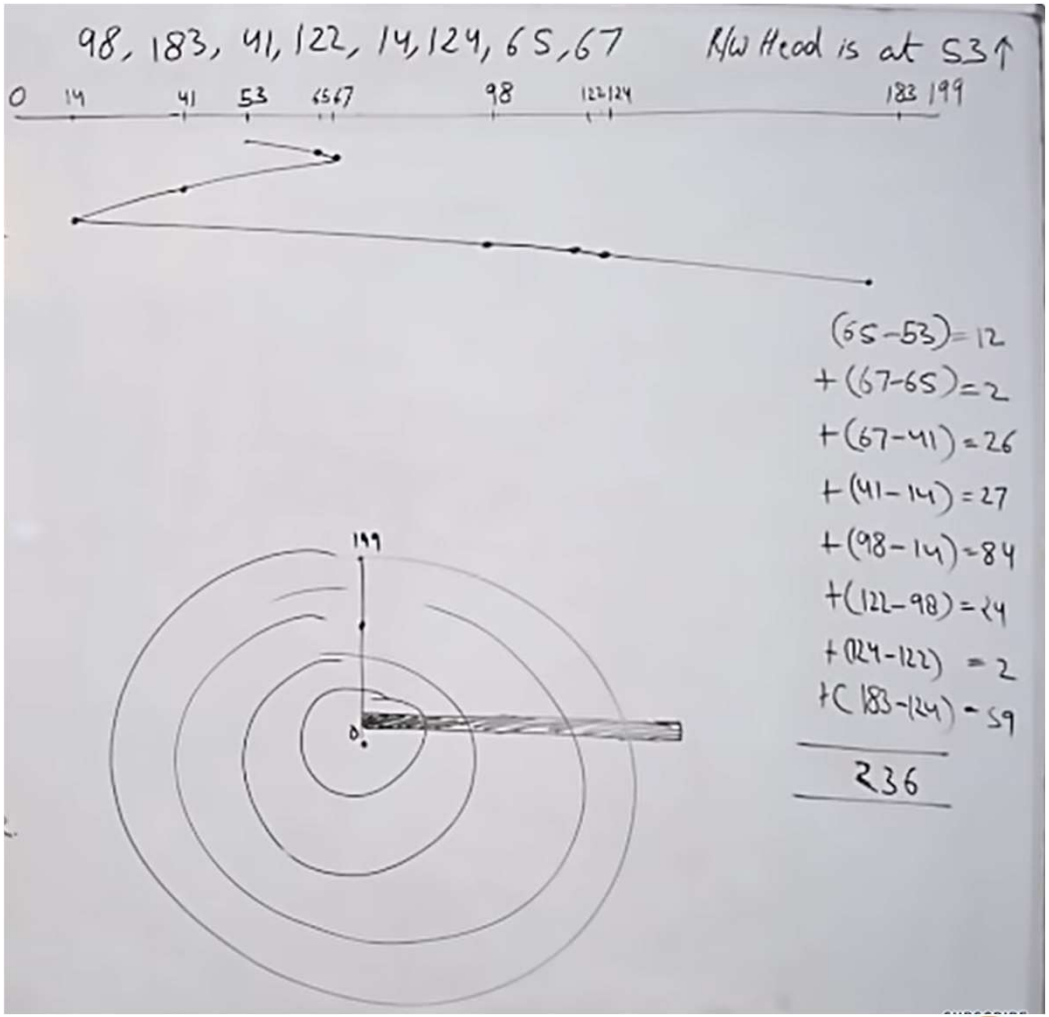


SSTF

- Serves the request which is closest to the current position of R/W head
- Tie is broken based on the direction of head movement.

Advantages:
- Very efficient in seek moves
- Less average response time and waiting time
- Increased throughput

Disadvantages:
- Overhead to find out the closest request
- Request which are far from head will starve
- High variance in response and waiting time

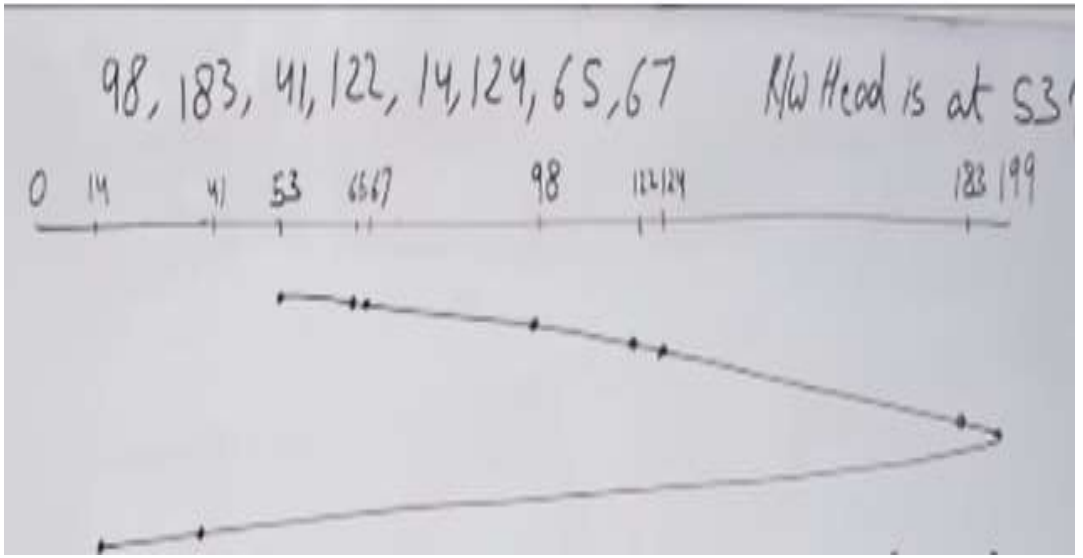


SCAN

- SCAN: Head start at one end and move towards the other end servicing requests in between.
- Sometimes called as elevator algo.

Advantages:
- Simple to implement
- No starvation (bounded waiting)
- Low variance and average waiting time

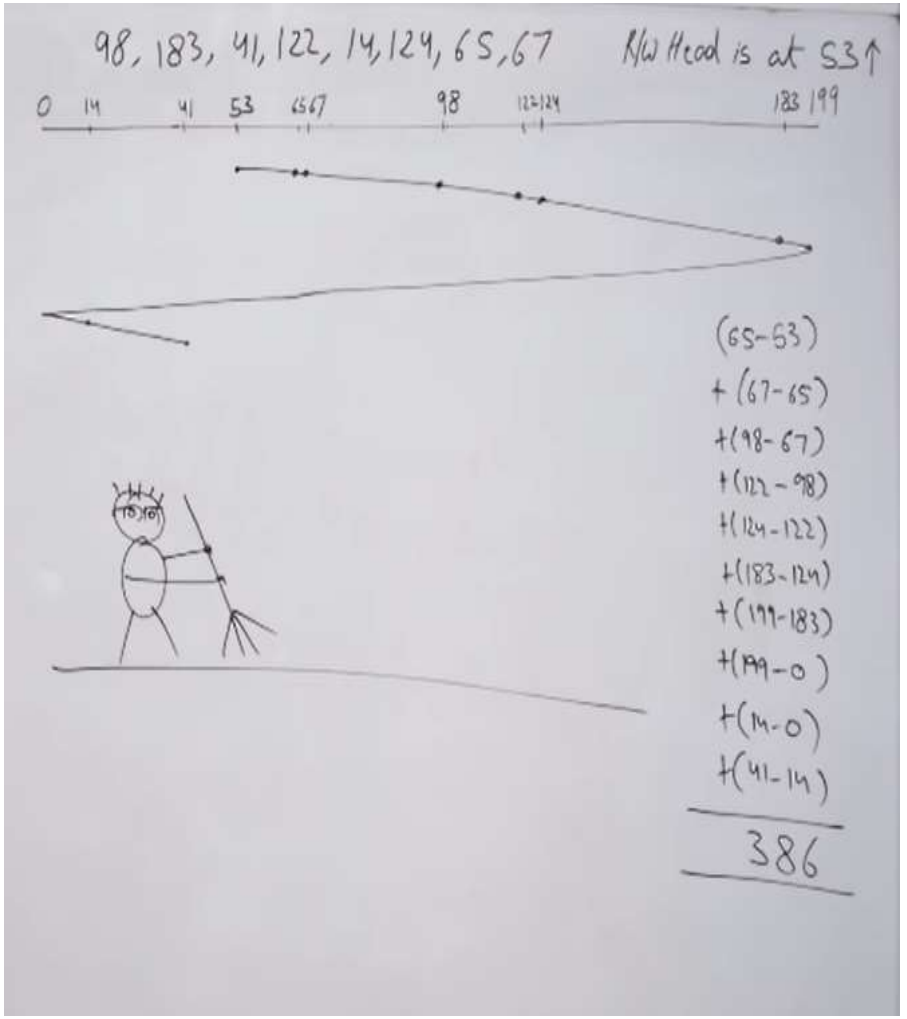
Disadvantages:
- Long waiting time for the location just visited by head
- Unnecessary move till the end of the disk, even if there is no request.



Number of move may increase compared to SSTF but Chances of starvation is reduced.

C-SCAN

Head starts at one end of the disk and moves towards the other end, servicing requests in between. Then the direction of the head is reversed and head reaches other end without satisfying any requests.



Advantages:

- Provides uniform waiting time
- Better response time

Disadvantages:

- More seek movement compared to SCAN