The Memory hierarchy → How much, fast, expensive.

There is a trade off b/w Capacity, access time and Cost.

- fast access → more cost per bit.
- More Capacity → less cost per bit.
- More Capacity → more access time.

{Match Processor} ⇓ Very Costly solution

We require Large Capacity memory (1) with fast access time (3) and low Cost per bit (2)

Not directly available

Solution is to use relatively low Capacity memory with fast access time.

[Not rely on single memory — Employ → Memory Hierarchy]

(a) Inboard Memory : Register, Cache, Main mem.

(b) Outboard Storage : Mag. disk, CD, DVD

(c) Offline Storage : Mag. tapes

Going down in M.H

1. decreasing Cost per bit
2. Increasing Capacity
3. Increasing access time
4. Decreasing frequency of access to the memory by processor.

This is Key to Success

[Locality of reference]

Example: → Byte required by CPU → a) in level (a)
*) access directly
→ b) in level (b)
*) transfer to level (a) then access.

→ **Hit ratio (H):** fraction of all memory access that are found in faster mem.

*) Our 4th point ensure high (H) that make sure the average total access time as close as to highest/higer level.

→ 4th Condition is ensured using

Locality of reference

- Memory addressing/referencing tends to cluster.
  - → looping/subroutine (instructions)
  - → table/array accessing (data)

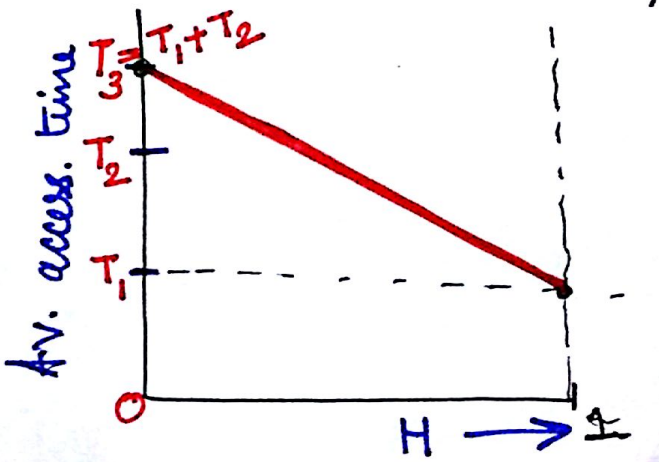
Over short period – processor works with fixed clusters.

Place current cluster in level 1

time to time swap cluster back to level 2 to accomodate more cluster

level 1 – 1000 by – 0.1 μs
level 2 – 100,000 by – 1 μs
H = 0.95 ⇒ 95%

av. access time = (0.95)(0.1 μs) + (0.05)(0.1 μs + 1 μs)
= 0.095 + 0.055 = 0.15 μs



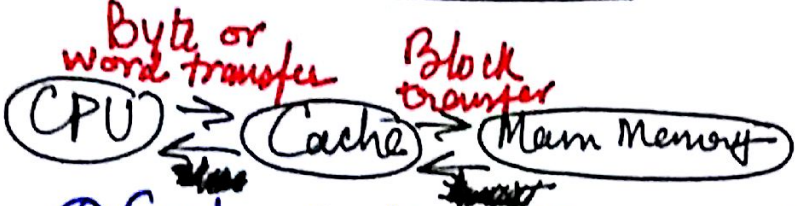
*) Each location in main memory has a unique address and is extended with a higher speed smaller cache ← Not visible to programmer, Staging data movement b/w Main mem. & Proc. Reg.

*) Register, Cache, main memory ⇒ Volatile

*) Secondary memory is non volatile.

Cache Memory

: The instruction cycle time is limited by memory cycle time ⇒ Mismatch b/w processor & main memory speed

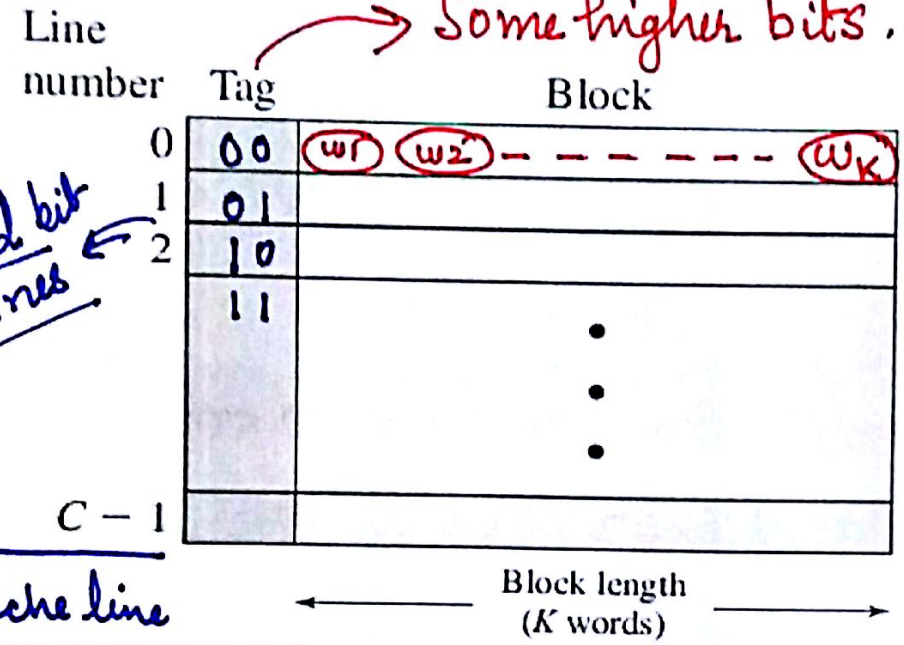


Using same technology to build both is too expensive

hence again use help of locality + Small fast memory in b/w ≡ Cache

* Cache Contain Copy of portion of main memory. Processor read a word and check wheather it is in Cache, if not take it from main mem. [Block of data is fetched into the cache to satisfy a single memory reference]

and then there is Three level Cache organisation



→ Some higher bits.

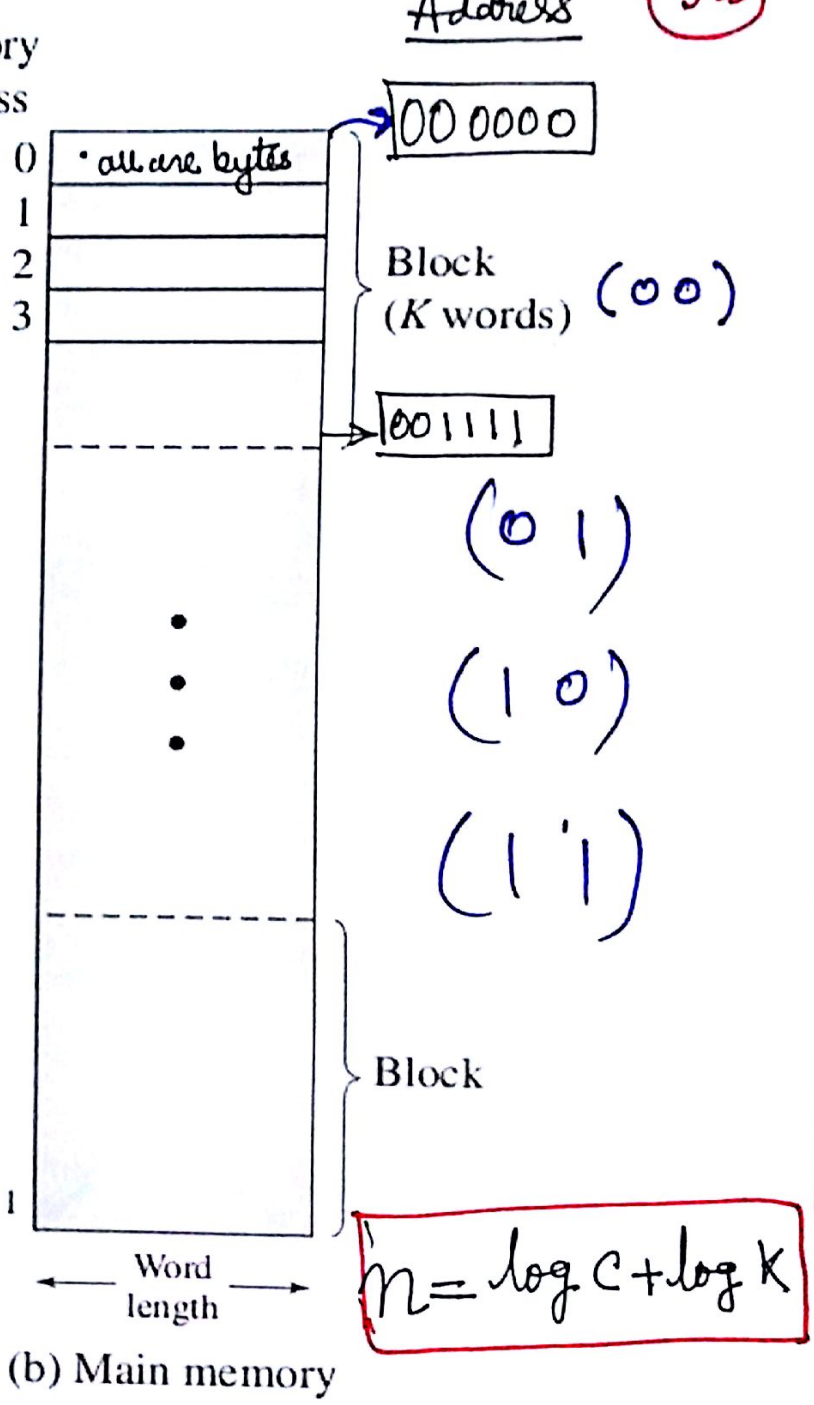
let 6 bit address with 2 bit tag

If 2 bit ← 4 Lines

(a) Cache

C = Cache line

$\log_2 C$ bit tag



(b) Main memory

- Main memory $\equiv 2^n$ addressable words (n-bit address)

M fixed length block of (K) words. $\Rightarrow M = \left\lfloor \frac{2^n}{K} \right\rfloor$

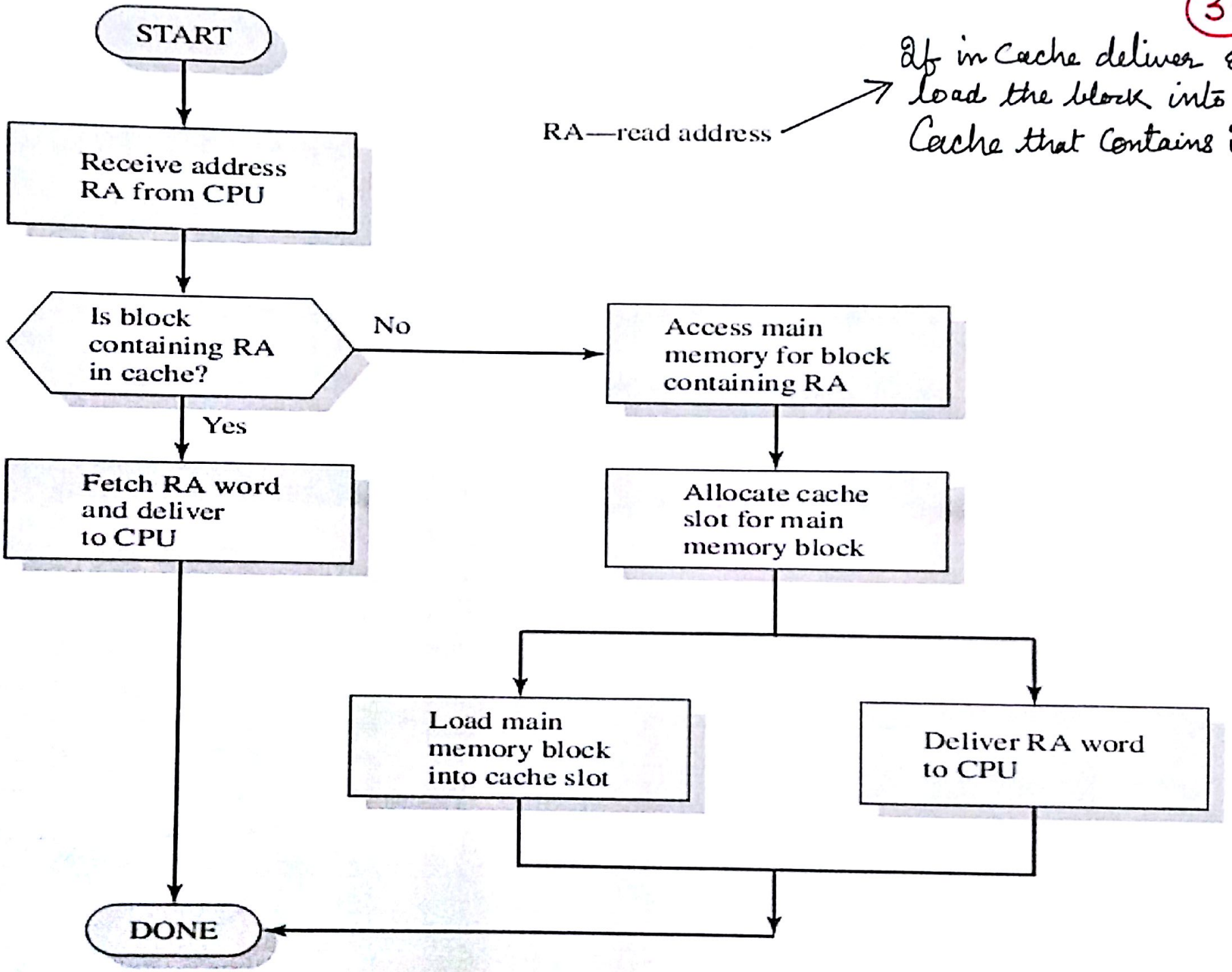
- typically $C << M$.

- Each slot/line includes a tag that identifies which particular block is currently stored.

$n = \log C + \log K$

Cache/Main-Memory Structure

If in cache deliver else load the block into Cache that contains it.



Cache Read Operation

Cache Designing:

(a) **Cache size**: Reasonably small may improve performance. But **block size** is an issue.

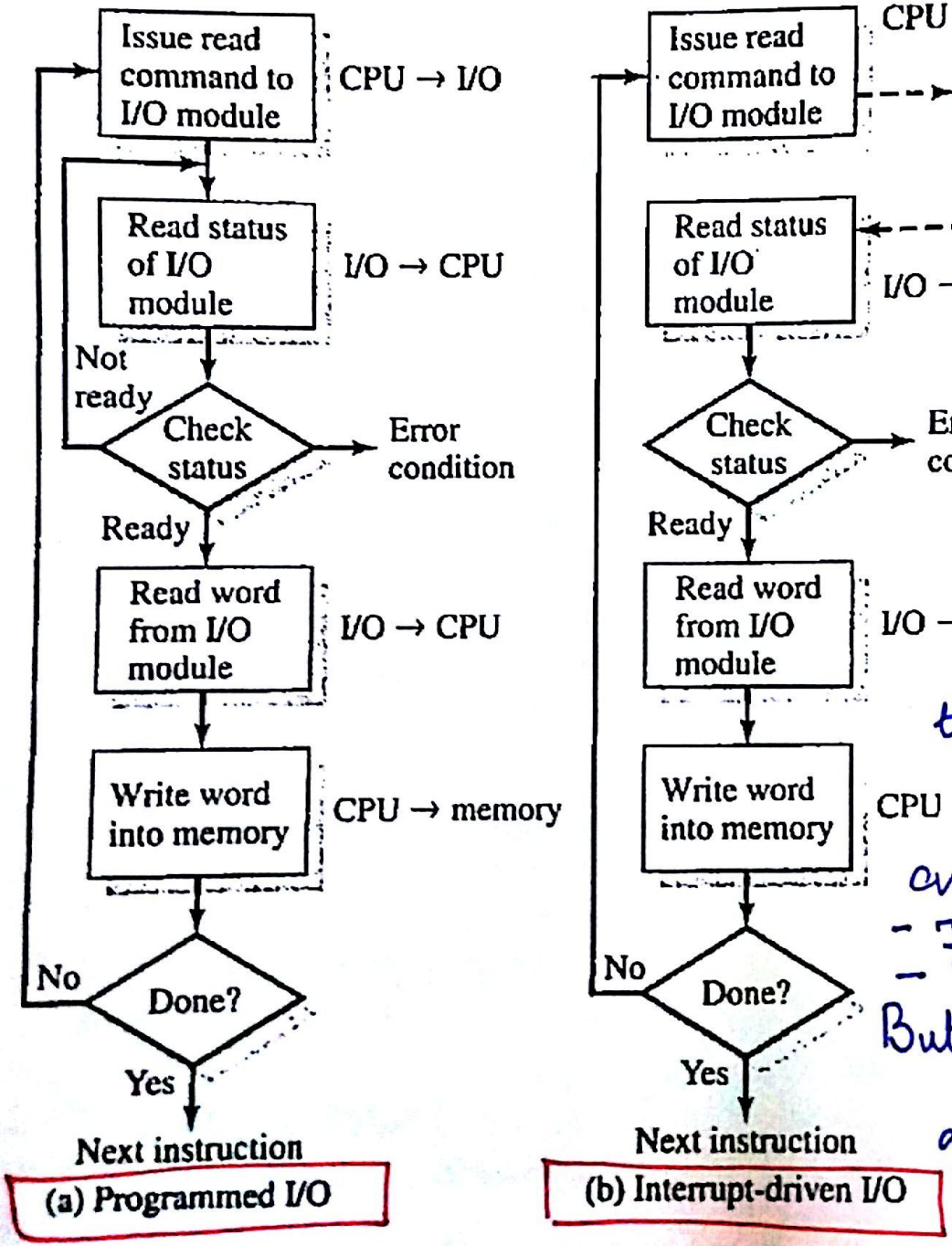
(b) **Block size**: is the unit of data exchanged b/w Cache & main memory.

* With increasing block size it first increase then decrease. (related to (H))

(c) **Mapping function** – which Cache location the block will occupy.
Direct, Associative and Set associative (read it)
liberal to Complex →

(d) **Replacement Algorithm**: With in constrains of Mapping fn. which block to replace when new block is loaded.
{ replace block which is in the cache for longest time w/o reference. ⇐ (LRU)

(e) **Write policy**: If modified must be written back. { Write through / Write back / write once }
→ Block updated → write
↓
Updated Block replaced – write → minimize memory write but leaves main memory in obsolete state.



Three Techniques for Input of a Block of Data

4a

① Processor encounter I/O then issue command appropriate I/O module.

② In P. I/O : I/O module perform action But donot alert the processor. Processor periodically polls.

③ In Interrupt. I/O :
- Processor issue I/O command and do some other work.
- I/O module interrupt when it is ready
- Processor transfer data and resumes.

But there are multiple I/O modules.

④ In DMA : when large volume of data is to be moved. CPU send DMA req
- Read/write — I/O address - Starting loc
- No. of words. After Completion DMA interrupts CPU.

OS - evolution

[a] Serial Processing : No - OS, programmer interacts directly with Comp. h/w.

Serial Processing mode

2-Step process – (i) Scheduling – Reserve a block of CPU time.
(ii) Setup time – loading and linking of the job.

→ later several System software are written to make it efficient.

[b] Simple batch System : Huge amount of CPU time got wasted in setup.

- It has monitor and user donot access processor directly instead via monitor
- User Submit job to operator, it batches on an input device used by monitor
- Monitor read job one by one place into user program area (Call it automatically loads) and pass control to it.
- After Completion Control Comes back to monitor that loads in next job.

[C] Multiprogramed batch System : Still processor is idle due to Slow I/O.

• Read one record from file	15 MS
• Execute 100 inst	1 MS
• Write one record	15 MS
% CPU-Utilization	3.2%

*) Assuming there is enough memory to hold multiple User programs.

*) when a program wait for I/O switch to other.

Multiprograming

What are its ISSUES ?

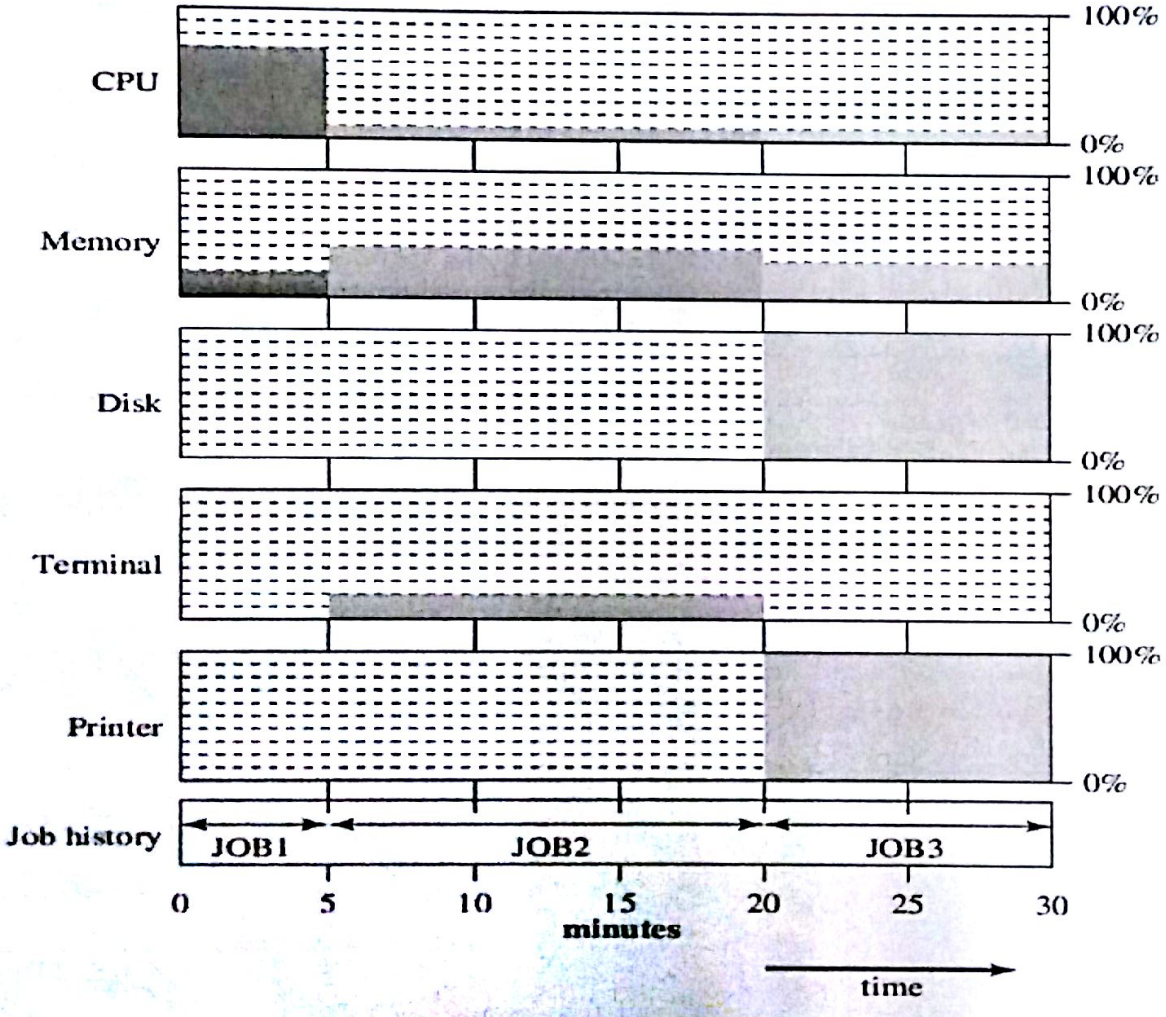
- System with 250 Mb memory (available)
- 3 Jobs are submitted
- Since little resource contention, they can run 11'y.

Not much CPU req

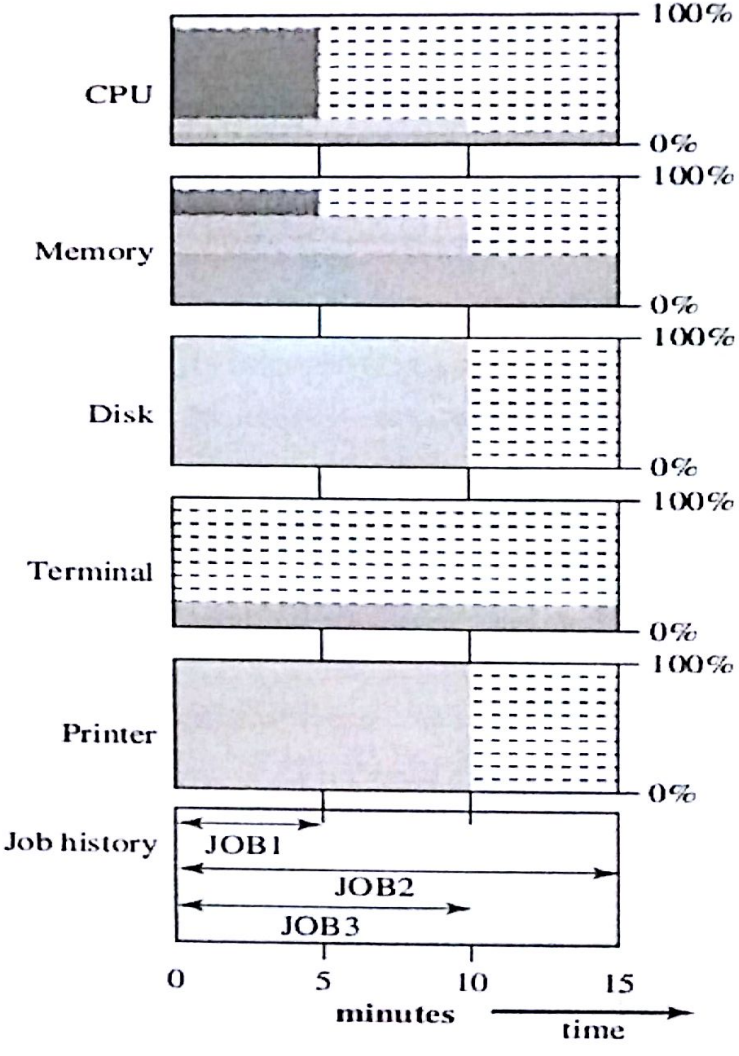
5a

Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes



(a) Uniprogramming



(b) Multiprogramming

Utilization Histograms

Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

- Multiprograming rely on I/O interrupts and DMA
- Processor issue I/O command for one job and proceed execution another job. I/O Carried by device controller.
- After I/O Completes processor got interrupted by device Contr..

Moving towards process model :-

Multiprograming was designed to keep processor and I/O devices busy to achieve maximum efficiff

↳ ⓐ **Key mechanism** : When processor got I/O Completion signal it switches to some other resident program.

ⓑ **Principal tool** : Early system programer rely on **Interrupt**.
(we have seen Interrupt handling)
⇒ load content
⇒ Process interrupt
— —
— — . .

* But in multiprograming environment, Co-ordinating various such activities with multiple jobs in progress (require different steps & instruction to be perfomed)* turns out to be very difficult. (Messy)

* Such system vulnerable to errors, that are hard to diagnose, detect and debug.

Four Main Causes for such error :

(i) **Improper Synchronization** : Read-Write I/O. (lost/duplicate signal problem may cause error.)

(ii) **Failed mutual exclusion** : More than one program attempt to use same resource.

(iii) **Non determine program operation** : Programs running in shared memory with interleaved execution they started to interfere in unpredicted way.

(iv) **Deadlocks** : Two program hung up waiting for each other.

A Systematic procedure is required to handle such scenarios.
How to Control various programs executing on the processor.
(Augumentation)

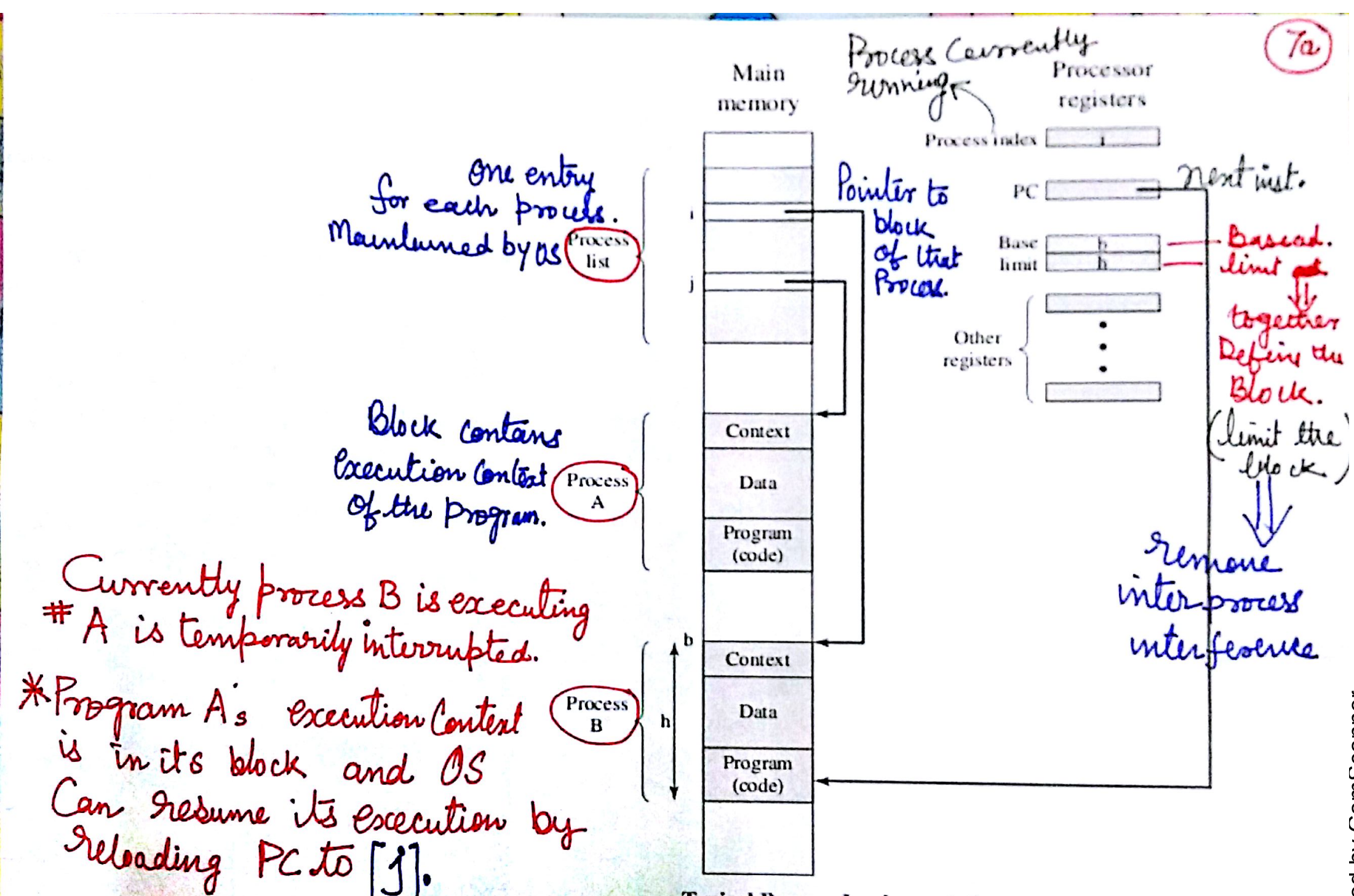
(*) Program under execution ≡> Process (how to model it)

- An executable program
- Associated data
- Execution Context of the program (Process state)

Required to Supervise the process.
⇒ Req. to manage process (By OS)
⇒ Req to execute (By CPU)

Content of various
— Processor reg., PC, data reg. --.
— Priority or waiting for I/O (which i/o) what event?

(*)
- Process can be realized as a data structure. (May be executing or waiting)
- It has various attributes — state
  — Priority (expand as req.)
- OS keep track of these states and manages the movement of process among these states. (Req. elaborated data structure)
- Process can be seen as an entity consist of number of elements.



Typical Process Implementation

Currently process B is executing
A is temporarily interrupted.

*Program A's execution Context is in its block and OS Can resume its execution by reloading PC to [j].

Process Control Block (PCB):-

Created & Managed by OS.

- Unique process modelling during it execution is req. (8)
- Hence it enables to interrupt and resume process.
- Key to multiprocessing (Process state can be switched to blocked and ready.)

(a) Identifier : Unique PID.

(b) State : Running, blocked, Suspended - - - .

(c) Priority : Its relative priority.

(d) PC : Address of next instruction to be executed.

(e) Memory pointer : Points to code & data (also shared block if any)

(f) Context data : Processor register data required for resumption.

(g) I/O Status : Outstanding I/O req, I/O devices holding, list of files using. - - -

(h) Accounting info : Amount of processor & clock time used - - - . (some bookkeeping).