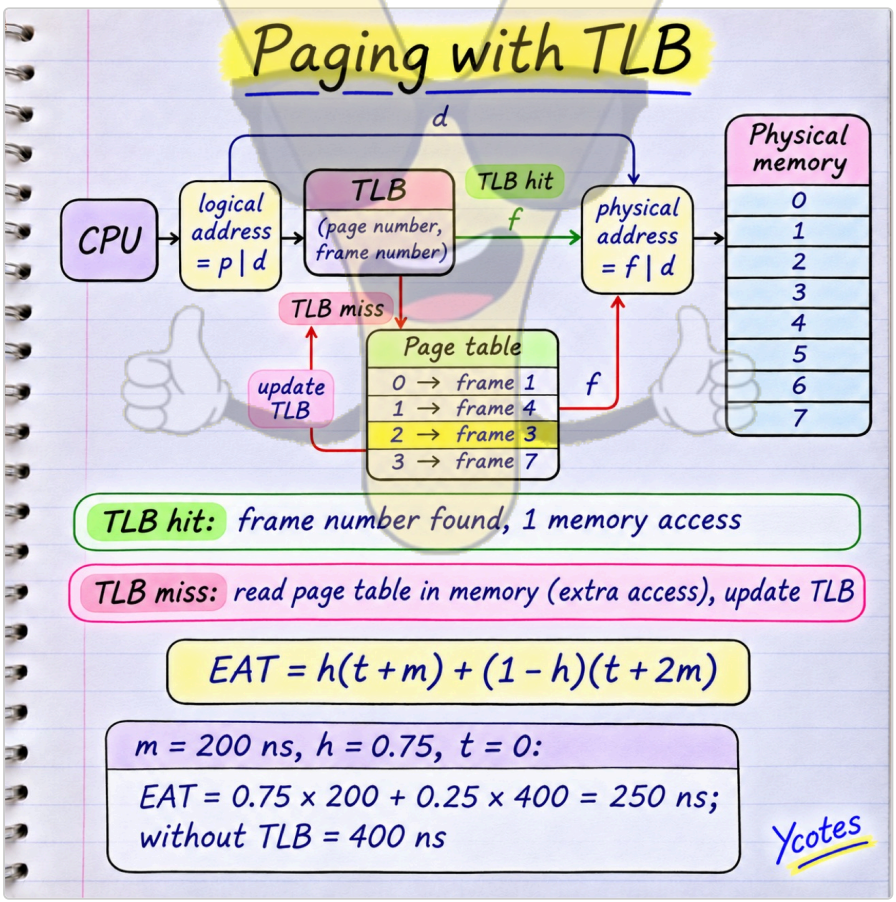
UNIT–2 : Memory Management (PYQ Solution)

PART–A Answers (≈70 Words Each)

Q1. What is TLB? Explain its function. (2019)

The Translation Lookaside Buffer (TLB) is a small, very fast associative cache inside the MMU that holds recently used page-table entries (page number to frame number).

Function: on every address translation the TLB is searched in parallel. On a hit the frame number is obtained at once and only one memory access is needed. On a miss the page table in memory is read (one extra access) and the entry is loaded into the TLB. It greatly reduces the cost of paging.



Q2. Why page size is always power of 2? (2019, 2023, Feb 2024)

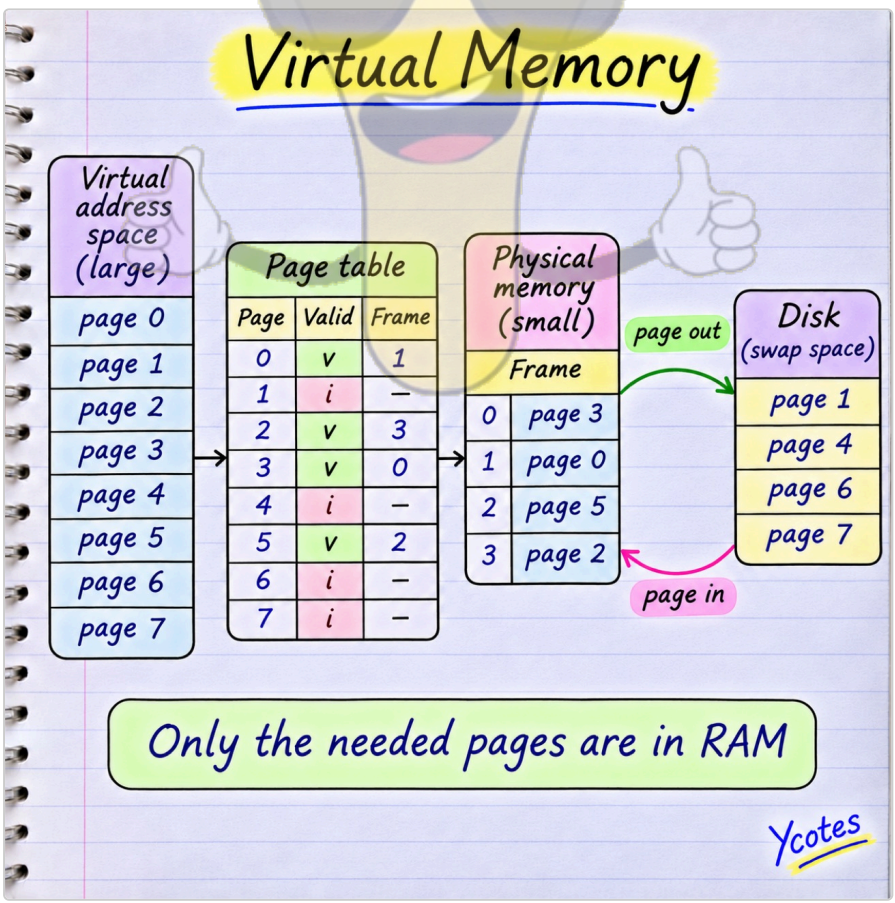
The logical address is divided into a page number and a page offset. If the page size is 2^n, the lower n bits of the address are the offset and the remaining high bits are the page number.

This split is a simple bit separation, with no division or modulo, so the hardware translation is fast. The offset bits are copied unchanged to the physical address, and the physical memory divides exactly into frames.

Q3. Define the term virtual memory. (2021, Feb 2024)

Virtual memory is a technique that allows a process to execute even if it is only partly in main memory. The logical address space can be much larger than the physical memory.

The OS keeps the unused parts of a program on the disk (swap space) and brings them into memory when they are needed, using demand paging. It increases multiprogramming and lets programs larger than RAM run.

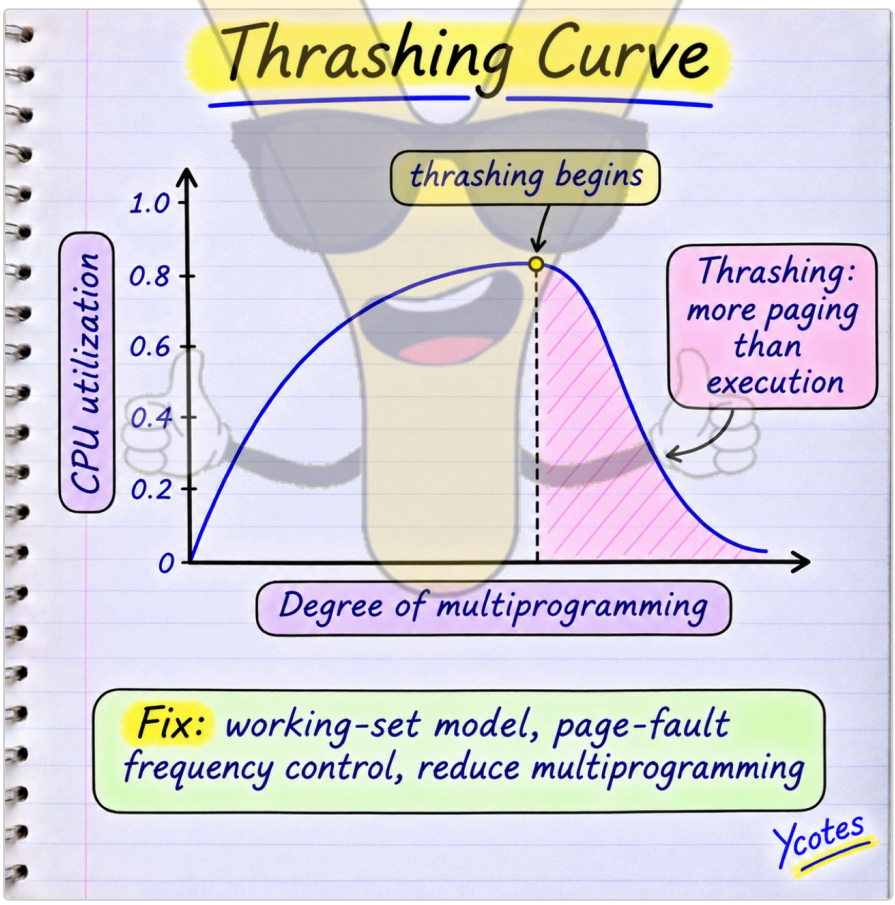


Q4. What is thrashing? / Define Thrashing. What are the reasons for thrashing? / What is thrashing in operating systems? (2021, 2022, 2023, Feb 2024, 2025)

Thrashing is a condition in which the system spends more time paging (swapping pages in and out) than executing processes. CPU utilisation falls sharply.

Reasons:

- **Too many processes are in memory (high degree of multiprogramming), so each gets too few frames.**
- **Frequent page faults make the CPU idle, and the OS then admits even more processes.**
- **Poor page-replacement choice or lack of locality.**



Q5. Define logical and physical address. (2022, Feb 2024)

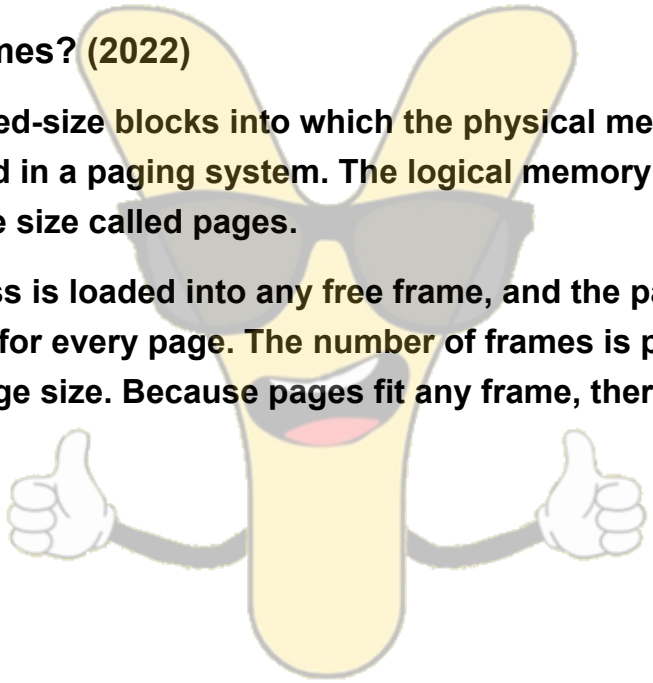
Logical address	Physical address
Generated by the CPU during execution	Actual location in main memory
Also called virtual address	Loaded into the memory address register
The set of all of them is the logical address space	The set of all of them is the physical address space
Seen by the user program	Never seen directly by the user

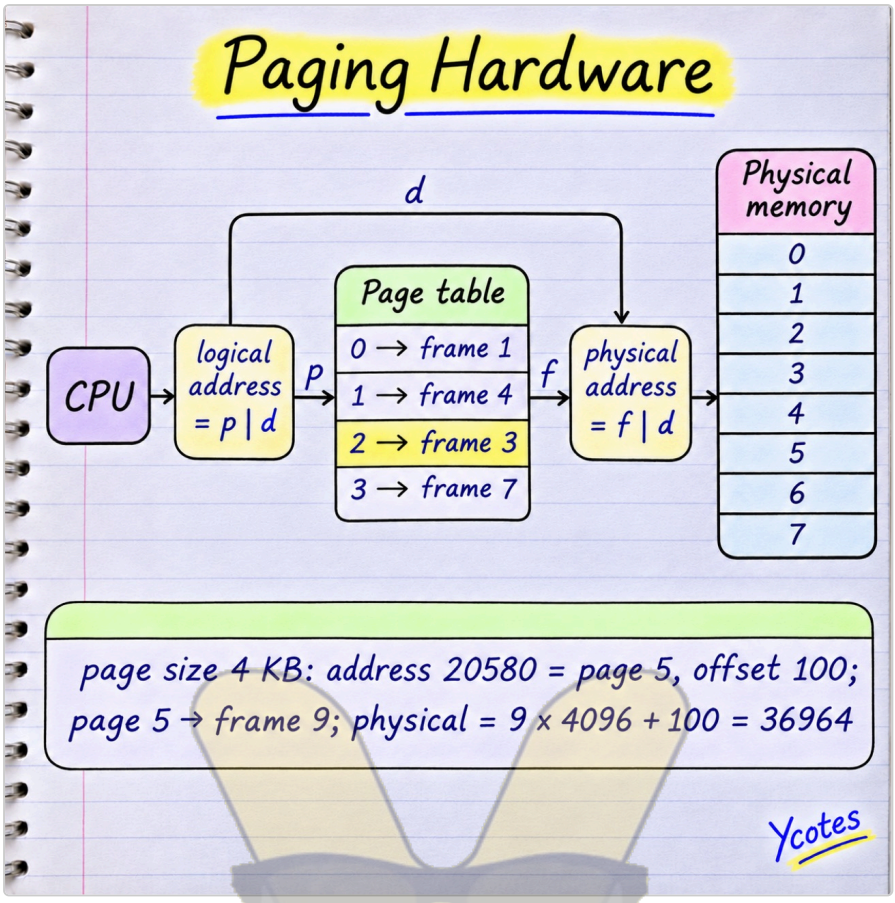
The MMU maps a logical address to a physical address at run time.

Q6. What are frames? (2022)

Frames are the fixed-size blocks into which the physical memory (main memory) is divided in a paging system. The logical memory is divided into blocks of the same size called pages.

A page of a process is loaded into any free frame, and the page table stores the frame number for every page. The number of frames is physical memory size divided by page size. Because pages fit any frame, there is no external fragmentation.





Q7. Differentiate between pager and swapper. (2022, Feb 2024)

Pager	Swapper
Moves individual **pages** between memory and disk	Moves a **whole process** between memory and disk
Used in demand paging	Used in swapping
Brings only the pages that are needed, so less overhead	Swapping the complete process has high overhead
Called a lazy swapper	Part of the medium-term scheduler

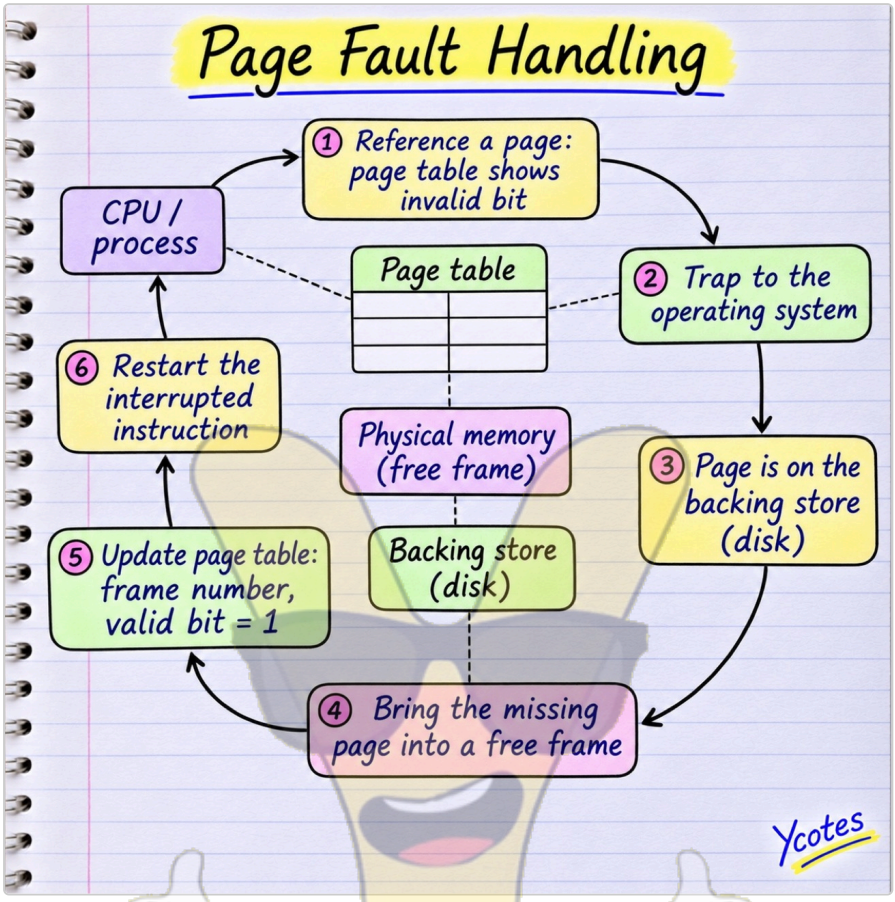
Q8. Explain 'valid' and 'invalid' bit in page table. (2022)

The valid-invalid bit is an extra bit attached to every page-table entry.

- **Valid (1): the page belongs to the process address space and is currently in main memory.**

- **Invalid (0): the page is either outside the address space of the process (illegal reference) or is legal but not currently in memory.**

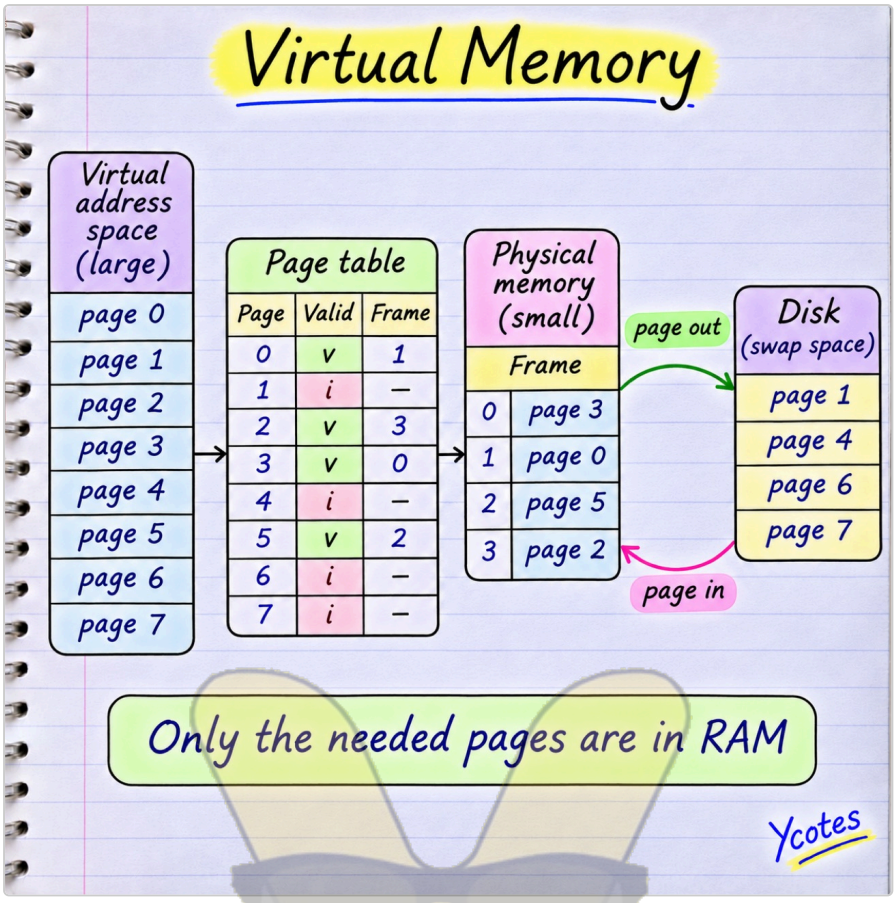
If the process refers to a page with the invalid bit, the hardware raises a trap called a page fault and the OS handles it.



Q9. What is Demand Paging? (2023)

Demand paging is a technique in which a page is brought into main memory only when it is needed, that is when it is referenced for the first time (lazy loading). It combines paging with swapping.

The page table uses a valid-invalid bit. If a referenced page is not in memory, a page fault occurs and the OS loads the page from the disk. In pure demand paging a process starts with no page in memory. It saves memory and I/O and supports virtual memory.



Q10. Define effective access time? (2024)

Effective access time (EAT) is the average time taken to access a memory location, taking into account the page-fault rate or the TLB hit ratio.

With demand paging, for page-fault rate p, memory access time m and page-fault service time T_f:

$$EAT = (1 - p)\, m + p\, T_f$$

With a TLB, for hit ratio h and TLB time t:

$$EAT = h(t + m) + (1 - h)(t + 2m)$$

Q11. Name any five page replacement algorithm used for page replacement. (2024)

Five page-replacement algorithms are:

1. **FIFO (First In First Out) replaces the oldest page.**

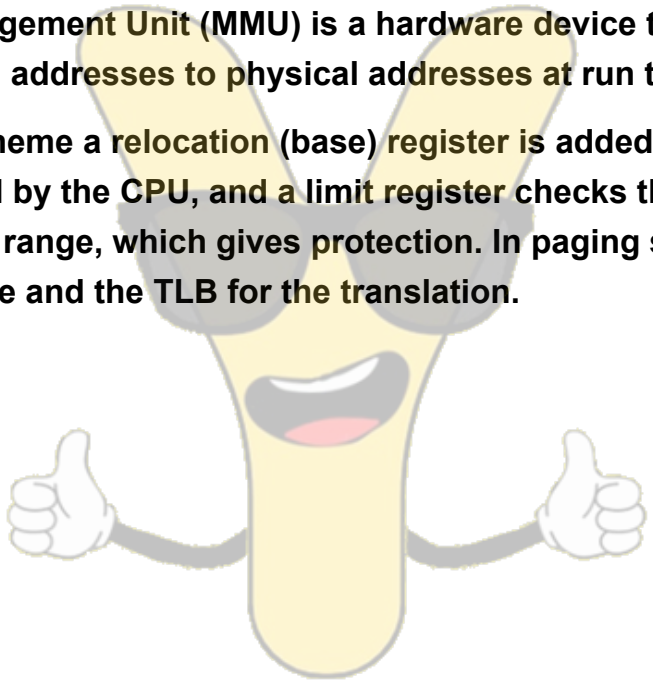
2. **Optimal (OPT) replaces the page that will not be used for the longest time in the future.**
3. **LRU (Least Recently Used) replaces the page unused for the longest time in the past.**
4. **LFU (Least Frequently Used) replaces the page with the lowest reference count.**
5. **Second chance (clock), FIFO with a reference bit.**

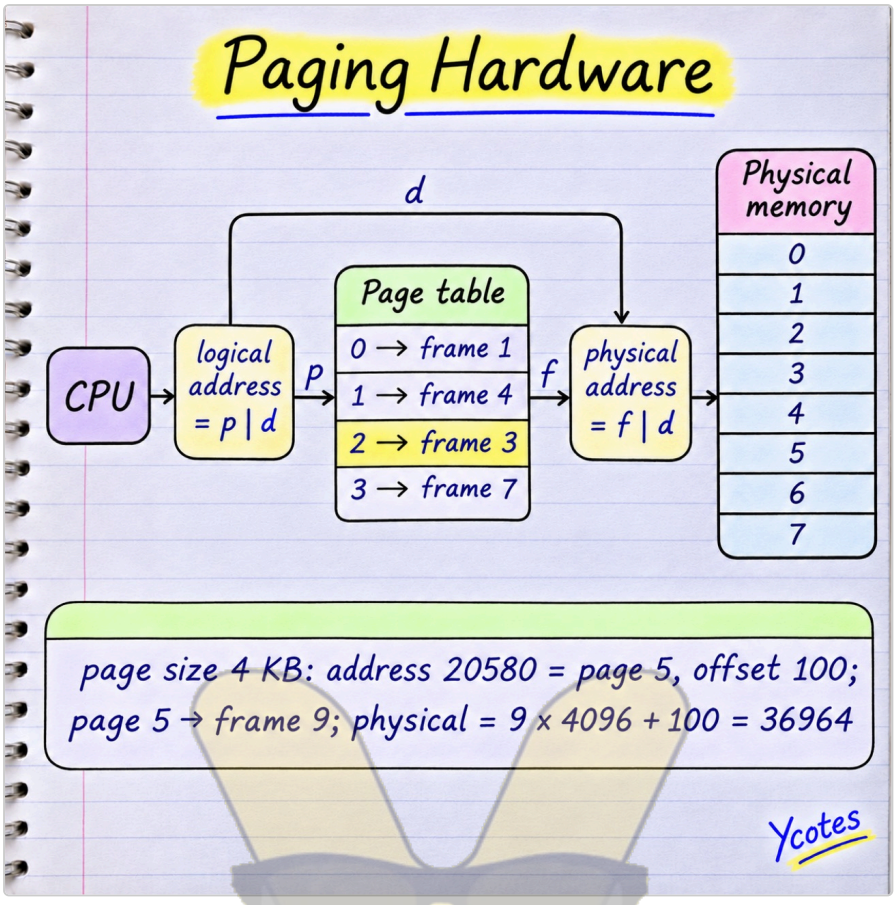
MFU and NRU are also used.

Q12. What is the main function of Memory Management Unit (MMU)? (2024)

The Memory Management Unit (MMU) is a hardware device that maps (translates) logical addresses to physical addresses at run time.

In the simplest scheme a relocation (base) register is added to every logical address generated by the CPU, and a limit register checks that the address is within the allowed range, which gives protection. In paging systems the MMU uses the page table and the TLB for the translation.





Q13. What is multithreading? What is its role in memory management? (2025)

Multithreading means executing several threads within one process; they share the same code, data, files and address space, but each has its own program counter, registers and stack.

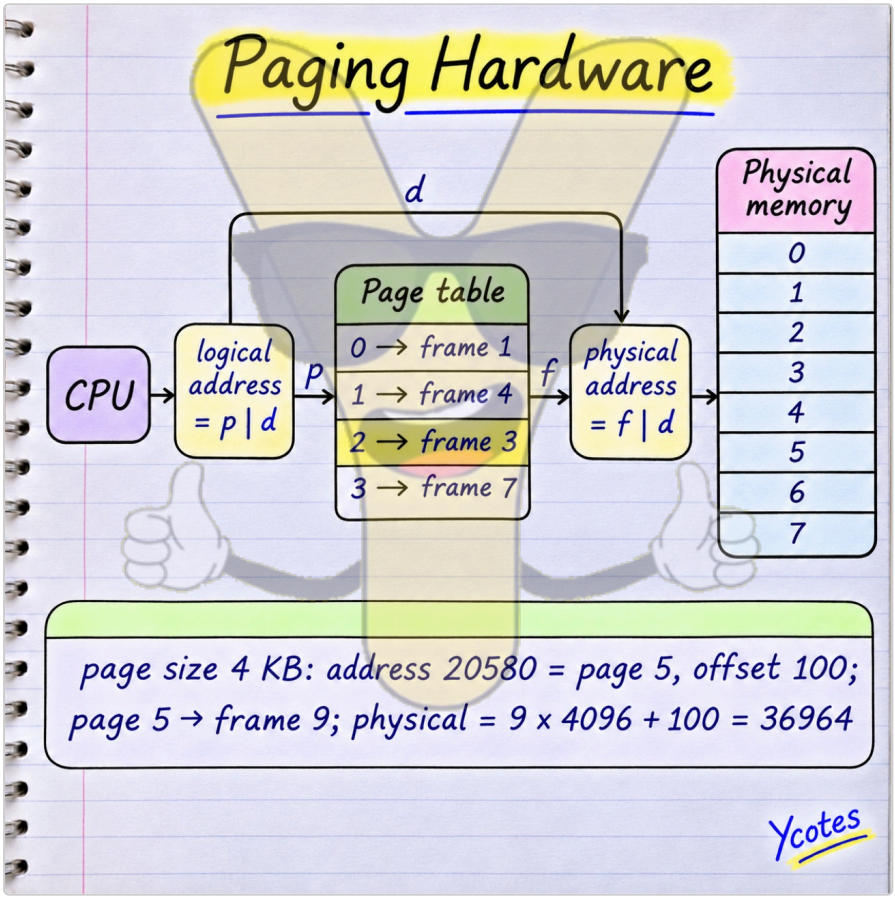
Role in memory management: because all threads of a process share one address space and one page table, they need less memory than separate processes, and a thread switch does not need a change of address space or a TLB flush. This makes thread creation and switching cheaper and allows data sharing without IPC.

PART–B Answers (≈250 Words Each)

Q1. What is paging? Explain with the help of a diagram. (2019)

Paging is a memory-management scheme that allows the physical address space of a process to be **non-contiguous**. Logical memory is divided into fixed-size blocks called **pages** and physical memory into blocks of the same size called **frames**. A **page table** stores the frame number for each page.

Address translation: the logical address generated by the CPU has two parts, the **page number (p)** and the **page offset (d)**. The page number indexes the page table to get the frame number **f**, and the physical address is (f, d).



For a logical space of 2^m and page size 2^n: the high $m - n$ bits give p and the low n bits give d.

Example: page size = 4 KB, page 5 is in frame 9, logical address = 20580. Then $20580 = 5 \times 4096 + 100$, so p = 5 and d = 100. The physical address = $9 \times 4096 + 100 = 36964$.

Advantages: no external fragmentation, efficient use of memory, easy sharing of pages and simple allocation.

Disadvantages: internal fragmentation in the last page of a process, extra memory for page tables, and an additional memory access for each reference (reduced by the **TLB**).

The valid-invalid bit in each page-table entry gives protection.

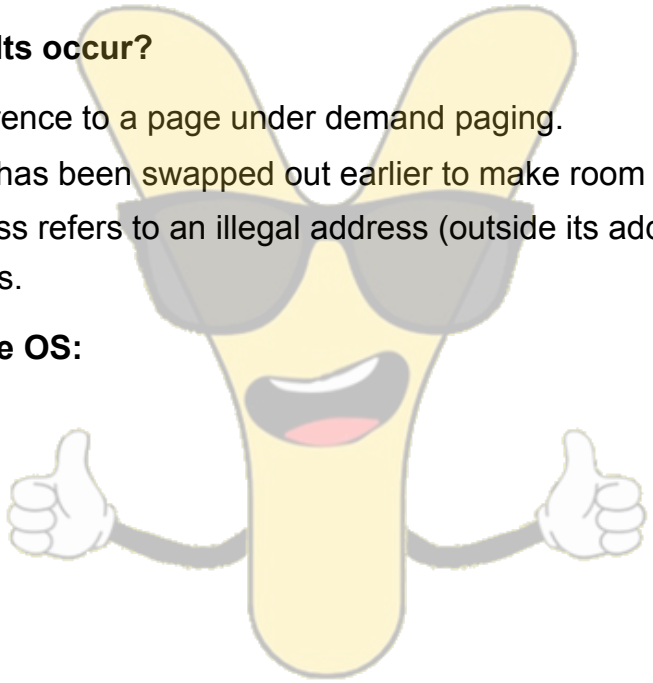
Q2. What do you mean by page-faults? When do page-faults occur? Describe the action taken by the O.S. when page-fault occurs? (2019)

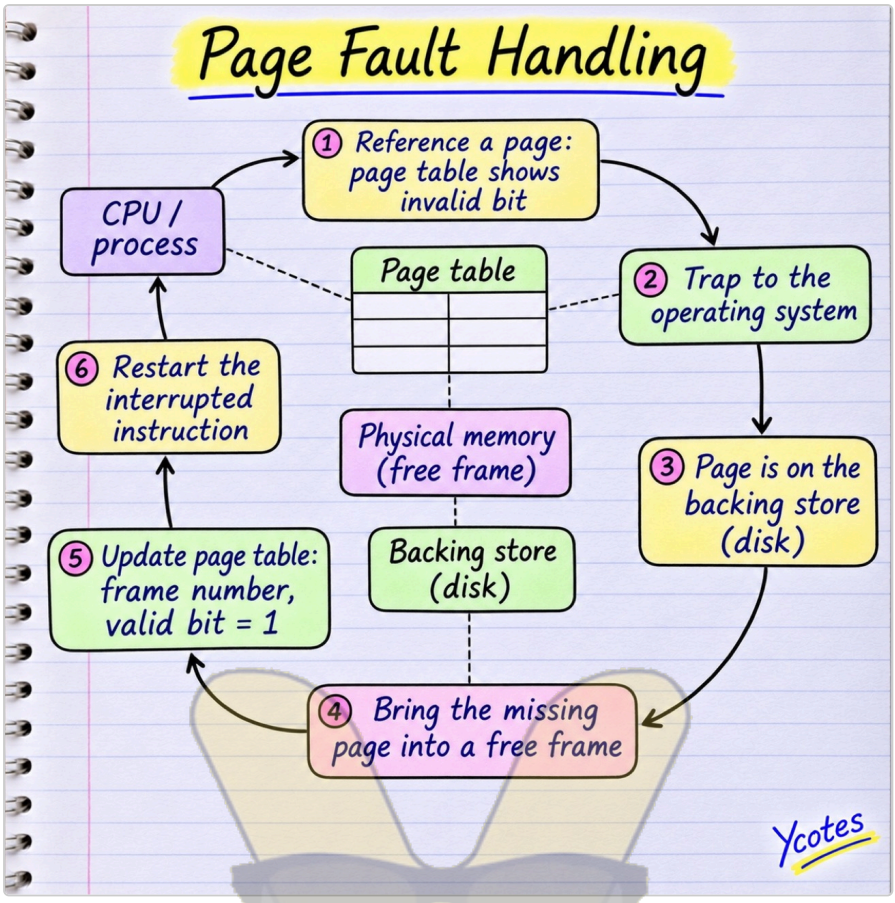
A **page fault** is a trap raised by the hardware when a process refers to a page that is **not in main memory** (valid-invalid bit = 0 in its page-table entry).

When do page faults occur?

- On the first reference to a page under demand paging.
- When the page has been swapped out earlier to make room for another page.
- When the process refers to an illegal address (outside its address space), which ends the process.

Action taken by the OS:





1. The CPU references the page and finds the invalid bit, so a **trap** goes to the OS.
2. The OS checks an internal table (in the PCB). If the reference is illegal, the process is **terminated**. If it is legal but the page is not in memory, continue.
3. The OS finds a **free frame**. If none is free, a **page-replacement algorithm** selects a victim; a modified victim is written to the disk.
4. A **disk read** is scheduled to bring the required page into the frame, and the process waits meanwhile.
5. After the read the page table and internal tables are updated: the frame number is stored and the valid bit is set to 1.
6. The interrupted instruction is **restarted**, and the process continues as if the page had always been in memory.

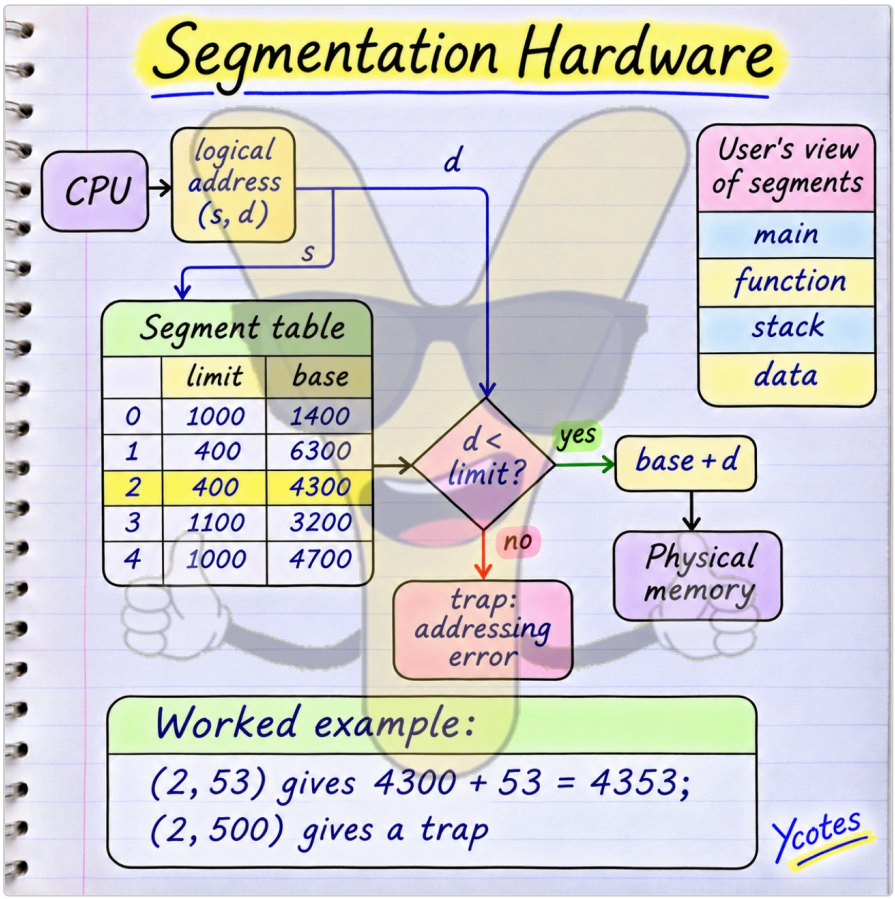
Q3. What is the importance of paging and segmentation in memory management? Explain with diagram. (2021, Feb 2024)

Importance of paging and segmentation: both are memory-management schemes that allow the physical memory of a process to be **non-contiguous**. This removes

the need for a single large free hole, supports multiprogramming and improves memory utilisation.

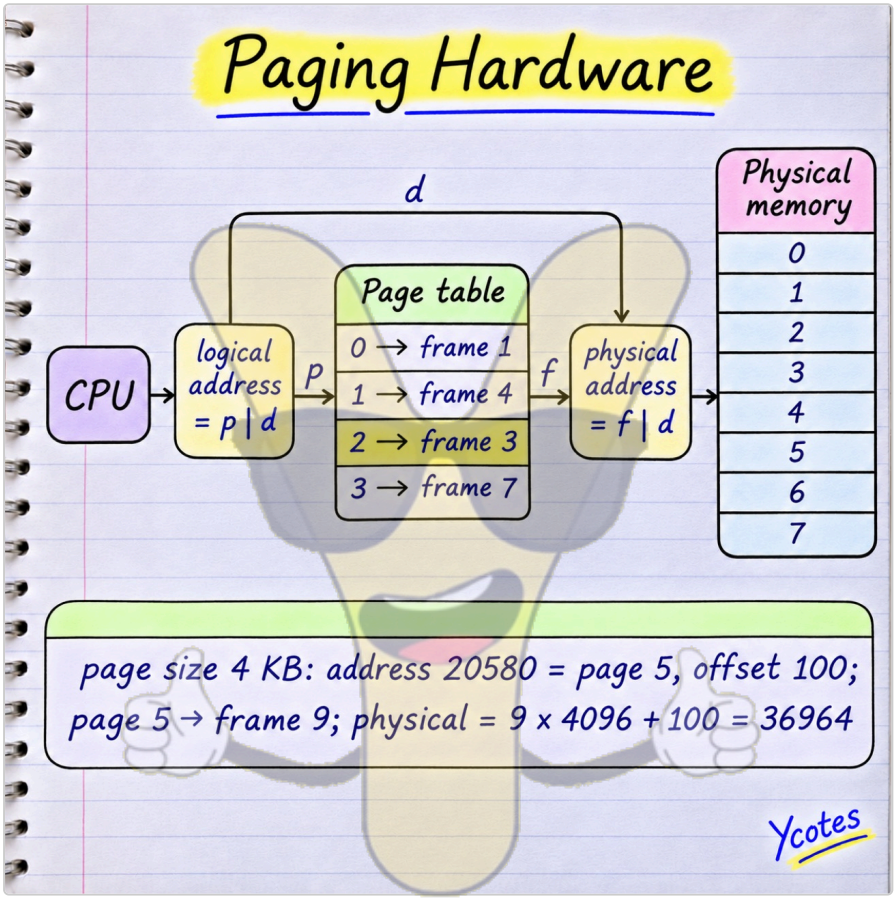
Paging: logical memory is divided into fixed-size pages and physical memory into frames. The address (p, d) is translated through the page table to (f, d). It removes external fragmentation, but the last page may waste space (internal fragmentation). It is invisible to the user.

Segmentation: a user-view scheme. The program is divided into logical segments of different sizes (main program, functions, stack, data). The address is (s, d). The **segment table** has a **base** and a **limit** for each segment: if $d <$ limit then physical address = base + d, otherwise a trap occurs.



Example: segment 2 has base 4300 and limit 400. The address (2, 53) gives 4300 + 53 = 4353. The address (2, 500) causes a trap, because 500 > 400.

Paging	Segmentation
Fixed-size pages	Variable-size segments
Invisible to the user	Matches the user's view of the program
Page table holds frame numbers	Segment table holds base and limit
Internal fragmentation	**External fragmentation**
Sharing by pages	Easy sharing and protection by logical unit

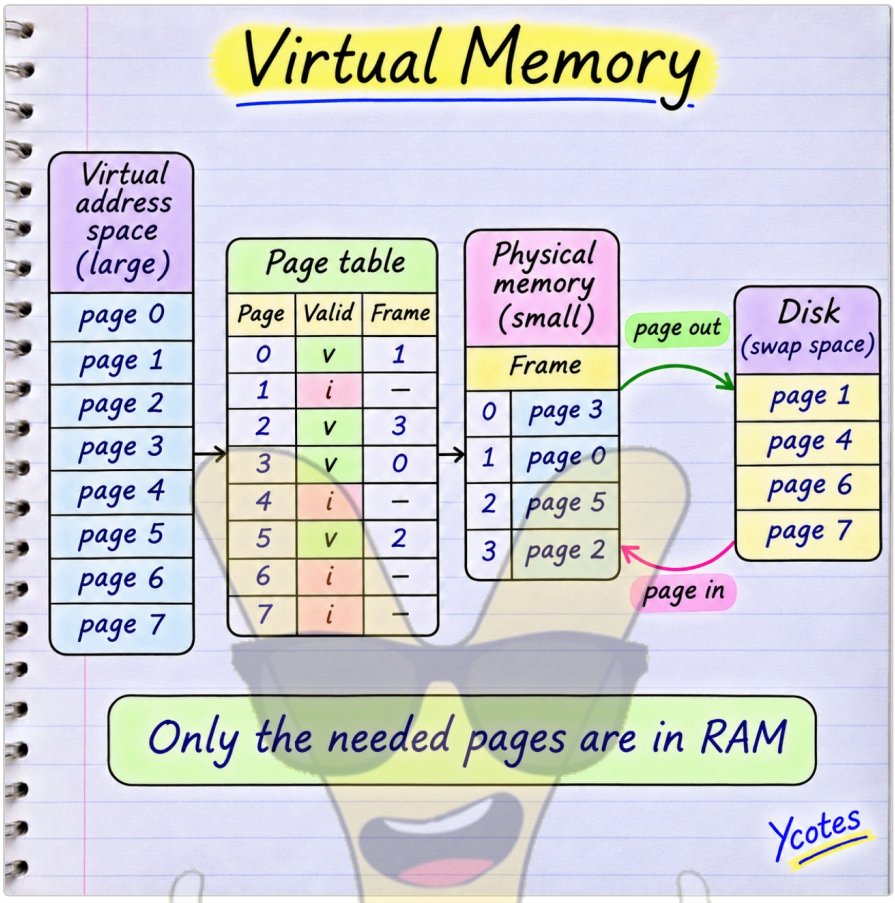


Both are combined in segmentation with paging, where each segment is itself paged.

Q4. What do you mean by demand paging? Explain virtual memory and page fault concept in detail. (2022)

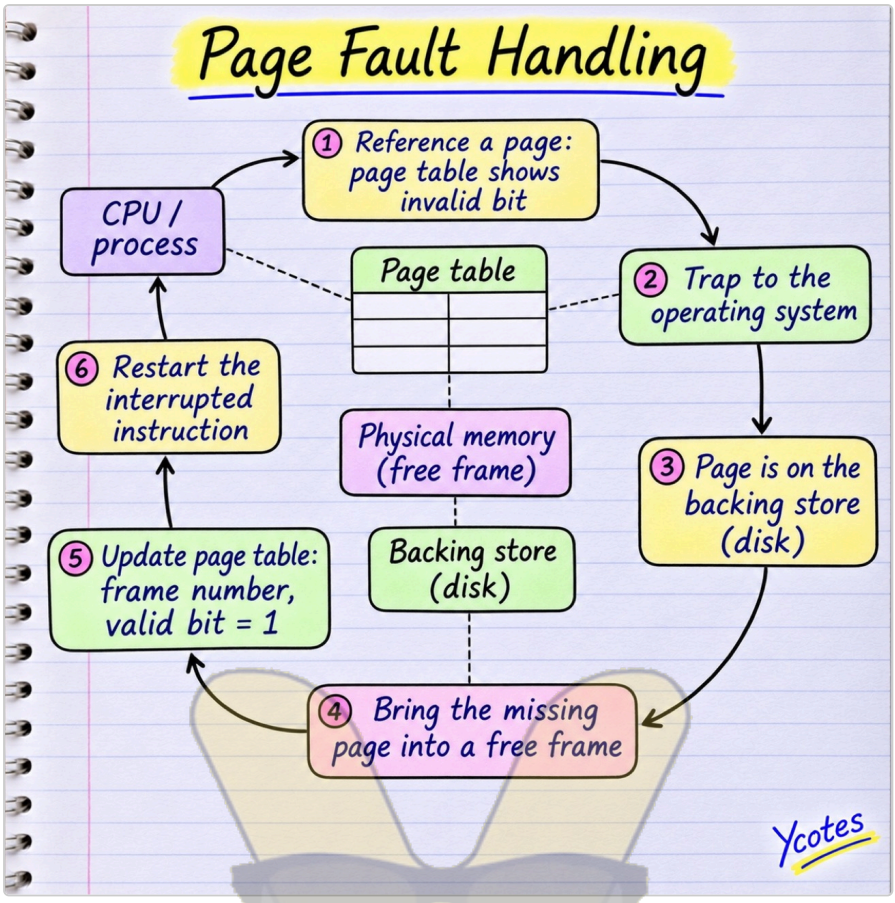
Demand paging: a process is divided into pages and a page is loaded into memory **only when it is referenced**, unlike loading the whole process at the start. A valid-invalid bit in the page table shows whether a page is in memory. A reference to an invalid page causes a **page fault**.

Virtual memory: it allows a process to run even though it is only partly in memory, so the logical address space can be much larger than the physical memory. Unused pages stay in the swap space on the disk. Benefits: programs larger than RAM, more processes in memory, less I/O than swapping whole processes.



Page fault concept: a page fault is a trap raised when the referenced page is not in memory. The OS handles it as follows:

1. Trap to the OS and check whether the reference is legal; if illegal, terminate the process.
2. Find a free frame (or choose a victim with a replacement algorithm).
3. Read the page from the disk into the frame.
4. Update the page table (frame number, valid bit = 1).
5. Restart the instruction.



Cost of page faults: servicing a fault needs a disk read, which is millions of times slower than a memory access, so the fault rate must be very small. This works because of the **locality of reference**: programs use a small set of pages repeatedly. If the faults become too many, **thrashing** results.

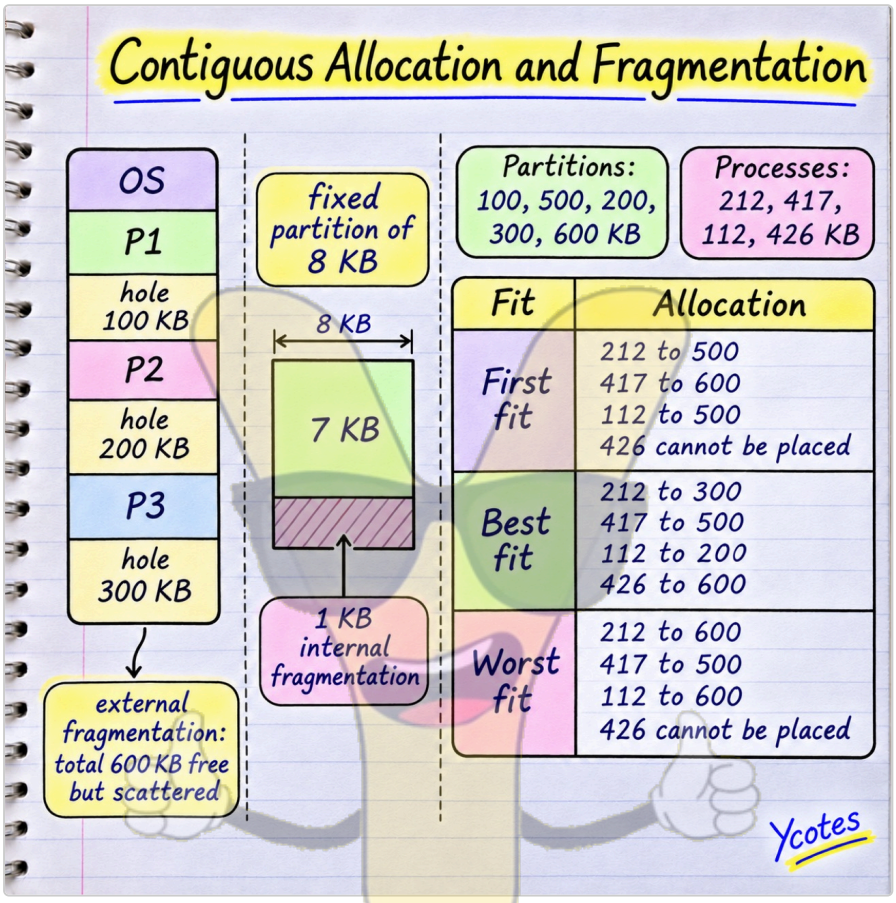
Q5. What is Memory Management Unit (MMU)? Explain Best Fit, Worst Fit and Quick Fit Algorithms in detail. (2022, Feb 2024)

Memory Management Unit (MMU): the hardware that translates a logical address generated by the CPU into a physical address at run time. A relocation (base) register is added to the logical address, and a limit register checks that it is within range. In paging, the MMU also uses the page table and TLB.

Memory allocation algorithms (contiguous variable-partition allocation) decide which free hole is given to a process.

- **Best fit:** allocates the **smallest** hole that is large enough. It leaves the tiniest leftover, but needs a search of the whole list and creates many small useless holes.

- **Worst fit:** allocates the **largest** hole, so the leftover is large enough to be useful, but it wastes the large blocks quickly and performs poorly.
- **Quick fit:** keeps **separate lists of holes of commonly requested sizes** (for example 4 KB, 8 KB, 16 KB). A request is served directly from the matching list, so finding a hole is very fast. The disadvantage is that merging neighbouring holes is difficult.
- **First fit** (for comparison) takes the first hole that is large enough; it is fast.



Example (KB): partitions 100, 500, 200, 300, 600 and processes 212, 417, 112, 426.

Process	First fit	Best fit	Worst fit
212	500 (leaves 288)	300 (leaves 88)	600 (leaves 388)
417	600 (leaves 183)	500 (leaves 83)	500 (leaves 83)
112	288 hole of the 500 (leaves 176)	200 (leaves 88)	388 hole of the 600 (leaves 276)
426	**cannot be placed**	600 (leaves 174)	**cannot be placed**

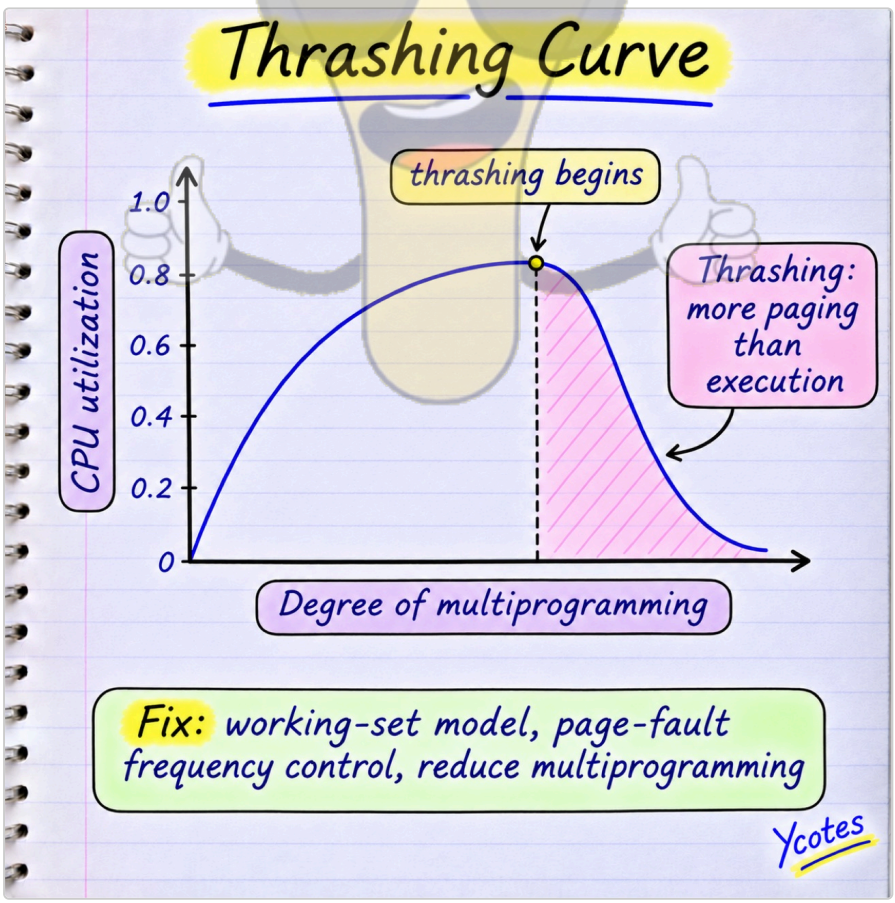
Only best fit places all four processes here.

Q6. Explain why the "Principle of Locality" is crucial to the use of virtual memory. What is accomplished by page buffering? (2023)

Principle of locality: programs do not reference all their pages uniformly. During any short time they reference a **small set of pages repeatedly**.

- **Temporal locality:** a recently used location is likely to be used again soon (loops, variables).
- **Spatial locality:** locations near a recently used one are likely to be used next (arrays, sequential code).

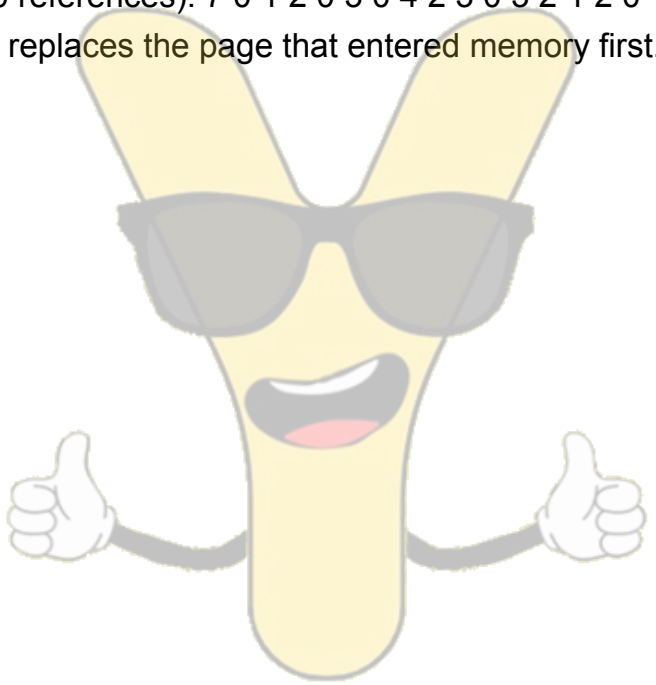
Why it is crucial for virtual memory: only part of a process is in memory. If references were random, almost every reference would be a page fault, and a fault costs a disk access, millions of times slower than memory. Thanks to locality, the working set of a process stays in RAM, so page faults are rare and the effective access time stays close to memory speed. Without locality virtual memory would cause **thrashing**.



Page buffering: the system keeps a **pool of free frames**. When a page fault occurs, the required page is read **into a free frame from the pool immediately**, without first writing out the victim page, so the faulting process restarts quickly. The victim frame is added to the pool later, after its contents are written to the disk if it is dirty. A **list of modified pages** can be written to the disk in batches when the paging device is idle, and the victim pages stay in the pool so that they can be reclaimed without a disk read if they are referenced again soon.

Q7. Consider 3 page frames and following reference string, use FIFO page replacement algo to calculate the number of page faults: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1. (Feb 2024)

Reference string (20 references): 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1, **3 frames**, all initially empty. FIFO replaces the page that entered memory first.



Ref	Frame 1	Frame 2	Frame 3	Result
7	7	-	-	**Fault**
0	7	0	-	**Fault**
1	7	0	1	**Fault**
2	2	0	1	**Fault**
0	2	0	1	Hit
3	2	3	1	**Fault**
0	2	3	0	**Fault**
4	4	3	0	**Fault**
2	4	2	0	**Fault**
3	4	2	3	**Fault**
0	0	2	3	**Fault**
3	0	2	3	Hit
2	0	2	3	Hit
1	0	1	3	**Fault**
2	0	1	2	**Fault**
0	0	1	2	Hit
1	0	1	2	Hit
7	7	1	2	**Fault**
0	7	0	2	**Fault**
1	7	0	1	**Fault**

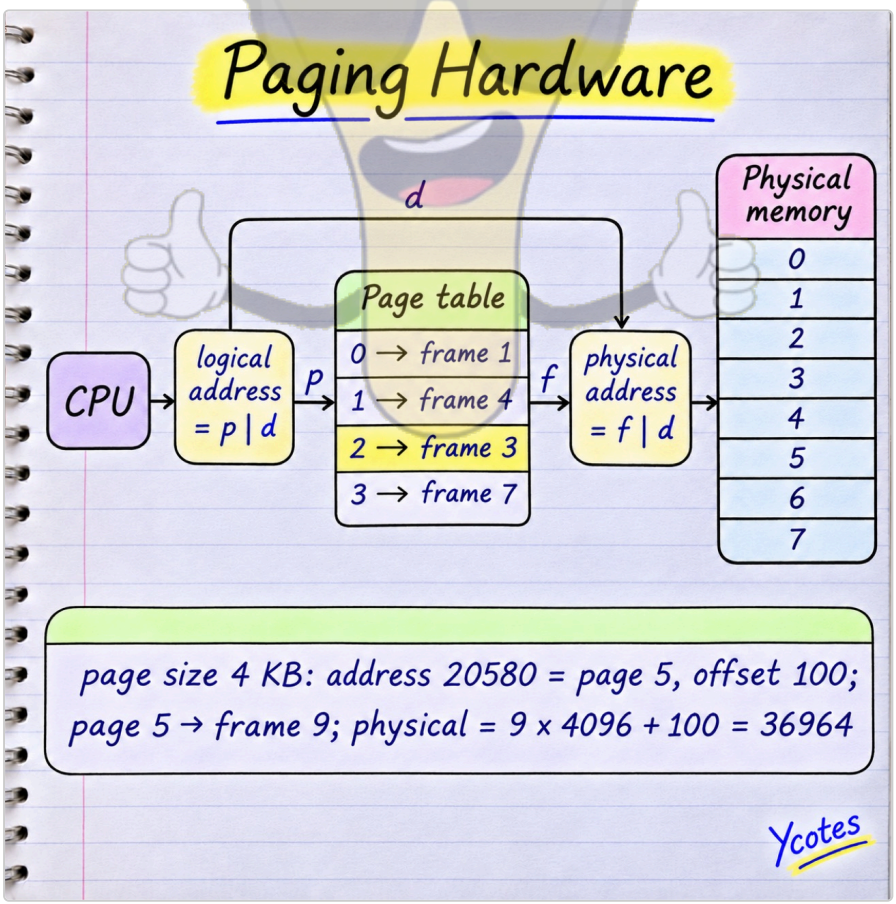
Total page faults = 15 (hits = 20 - 15 = 5). Page fault rate = 15/20 = 75%.

Q8. State the differences between logical and physical address space.

(2024)

Logical address space	Physical address space
Set of all logical (virtual) addresses generated by the CPU	Set of all physical addresses present in main memory
Size depends on the program and the address bits of the CPU	Size depends on the installed RAM
Generated by the CPU	Computed by the MMU
Seen and used by the user program	Never seen directly by the user
Can be larger than the physical space (virtual memory)	Limited by the hardware
Starts at 0 for every process	Is shared among all processes
Converted to a physical address by the MMU	Actual memory location accessed

In compile-time and load-time binding the logical and physical addresses are identical; in execution-time binding they differ and the MMU maps them.



Q9. Consider three page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, calculate the number of page faults using the optimal page replacement policy. (2024)

Reference string: 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6 with **3 frames**. The Optimal algorithm replaces the page whose next use is farthest in the future, or that is never used again.

Ref	Frame 1	Frame 2	Frame 3	Result
1	1	-	-	**Fault**
2	1	2	-	**Fault**
3	1	2	3	**Fault**
4	1	2	4	**Fault**
2	1	2	4	Hit
1	1	2	4	Hit
5	5	2	4	**Fault**
3	3	2	4	**Fault**
2	3	2	4	Hit
4	3	2	4	Hit
6	6	2	4	**Fault**

Total page faults = 7 (hits = 11 - 7 = 4).

At the first replacement (reference 4) pages 1, 2, 3 are in memory; the next uses are 1 at position 6, 2 at position 5 and 3 at position 8, so page 3 is replaced. At reference 5 the frames hold 1, 2, 4 and the next uses are 1 never, 2 at position 9 and 4 at position 10, so page 1 is replaced.

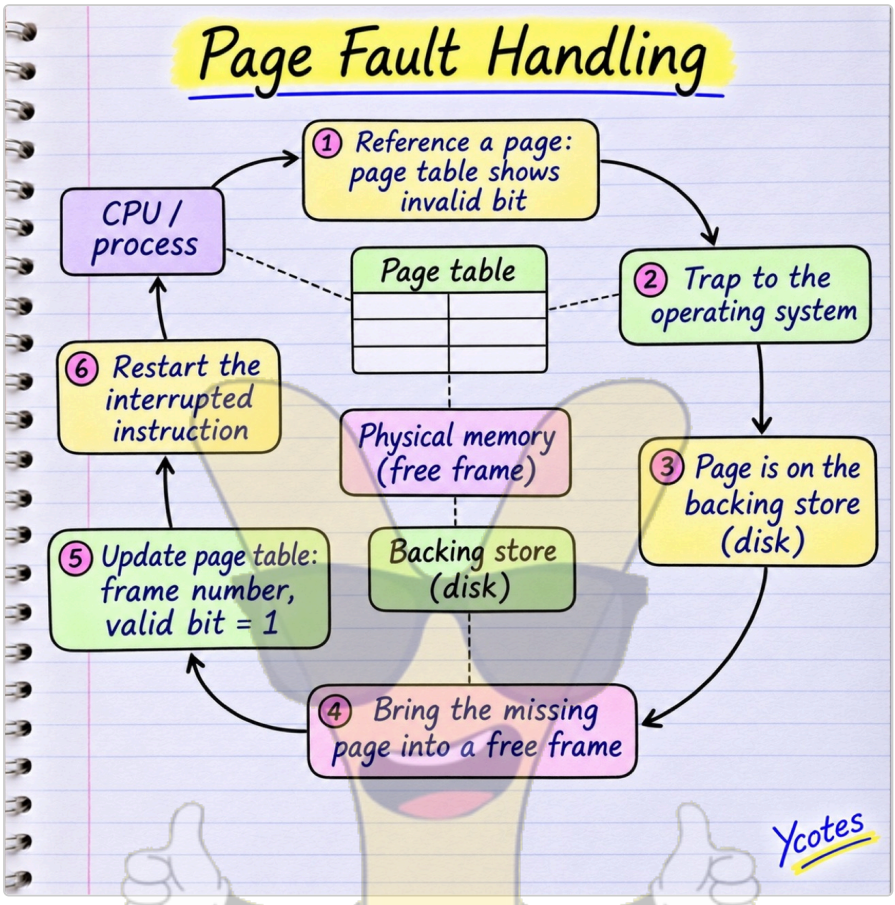
Q10. Explain the performance of demand paging with necessary examples. (2024)

Demand paging loads a page only when it is referenced. Its performance is measured by the **effective access time (EAT)**. Let p be the page-fault rate ($0 \leq$

$p \leq 1$), m the memory access time and T_f the page-fault service time (trap, disk read, restart).

$$EAT = (1 - p)\, m + p \times T_f$$

If $p = 0$ there are no faults and EAT = m; if $p = 1$ every reference faults.



Page-fault service steps (time T_f): trap and save state, find the page on the disk, read it into a free frame (the slowest step, several milliseconds), update the tables and restart the process.

Example: m = 200 ns, T_f = 8 ms = 8,000,000 ns and p = 1/1000.

$$EAT = 0.999 \times 200 + 0.001 \times 8{,}000{,}000 = 199.8 + 8000 = 8199.8 \text{ ns}$$

The EAT is about 8.2 microseconds, a slowdown by a factor of about 41, although only 1 reference in 1000 faults.

For a slowdown of less than 10%: $220 > 200 + p \times 7{,}999{,}800$ gives $p < 2.5 \times 10^{-6}$, i.e. fewer than 1 fault in about 400,000 accesses.

Conclusion: because disk access is so slow, the page-fault rate must be kept extremely low. This is achieved by locality of reference, a good replacement

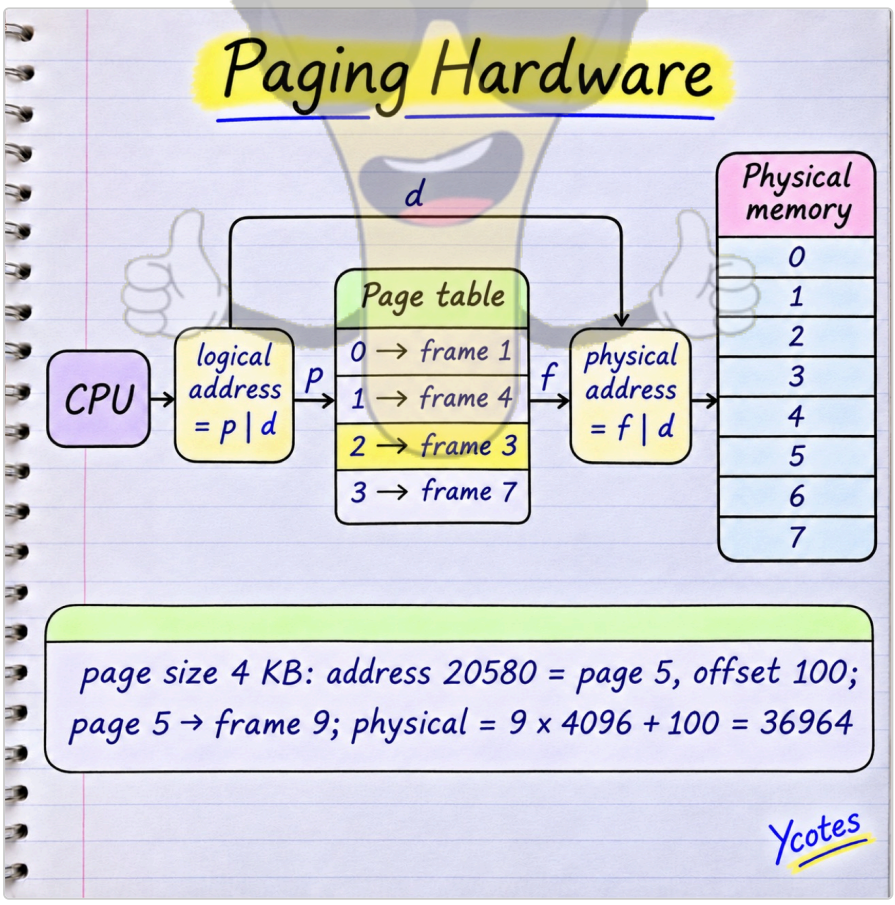
algorithm and enough frames. Demand paging has no performance problem only when the fault rate is very small.

Q11. A program requires 5 MB of memory, but only 3 MB is available in contiguous blocks. Can it run under paging? Why? (2025)

Assumption: the question does not say whether 5 MB is the process size or whether more memory is free elsewhere. It is read as: the free memory is scattered (a 3 MB contiguous free block and other free pieces), and the process needs 5 MB.

Yes, it can run under paging, provided the **total free memory** (the number of free frames) is at least 5 MB.

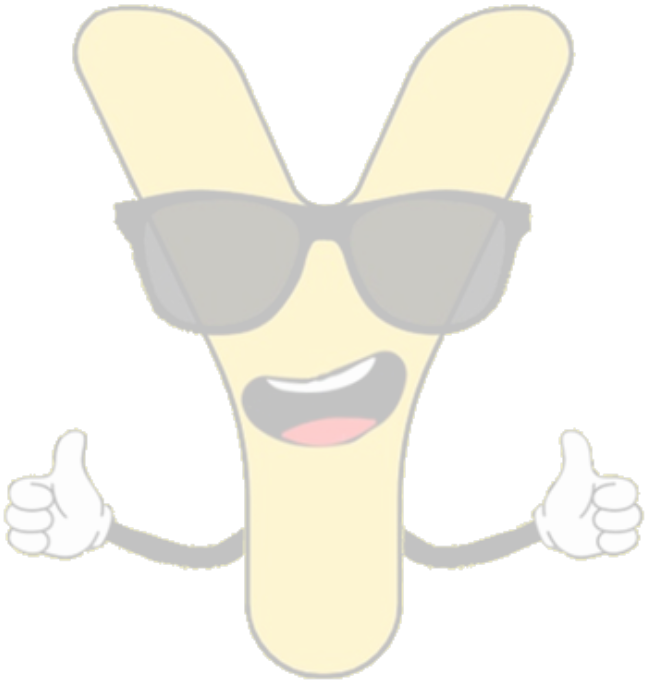
Why: paging allows **non-contiguous allocation**. The 5 MB process is divided into fixed-size pages, and each page can be placed in **any free frame**, wherever it lies in memory. The page table maps each page to its frame, and the MMU does the translation. The program needs no single 5 MB contiguous block, so external fragmentation does not stop it.



If only 3 MB in total is free: the whole process cannot be loaded at once. It can still run under **demand paging (virtual memory)**, because only the pages actually

needed are brought in and the remaining pages stay on the disk. Pages are replaced when the 3 MB of frames are full.

Contrast: under contiguous allocation (first fit, best fit) the process would fail because of external fragmentation.

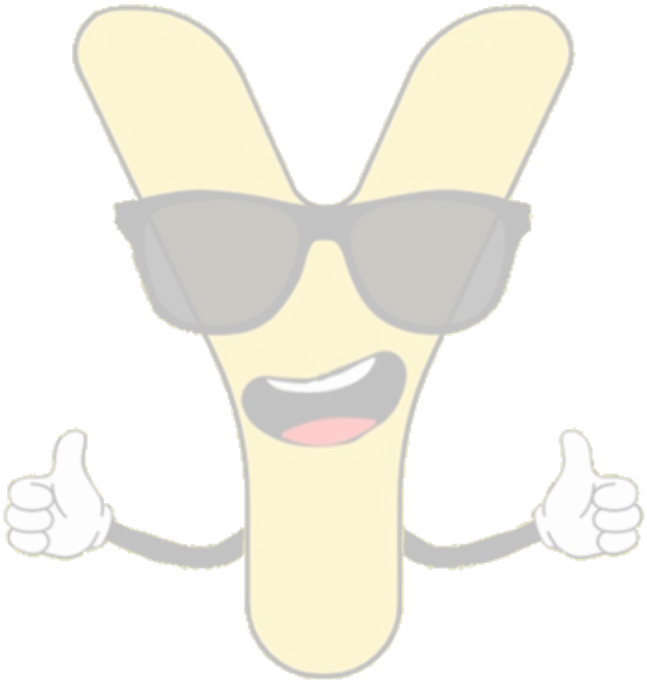


PART–C Answers (≈500 Words Each)

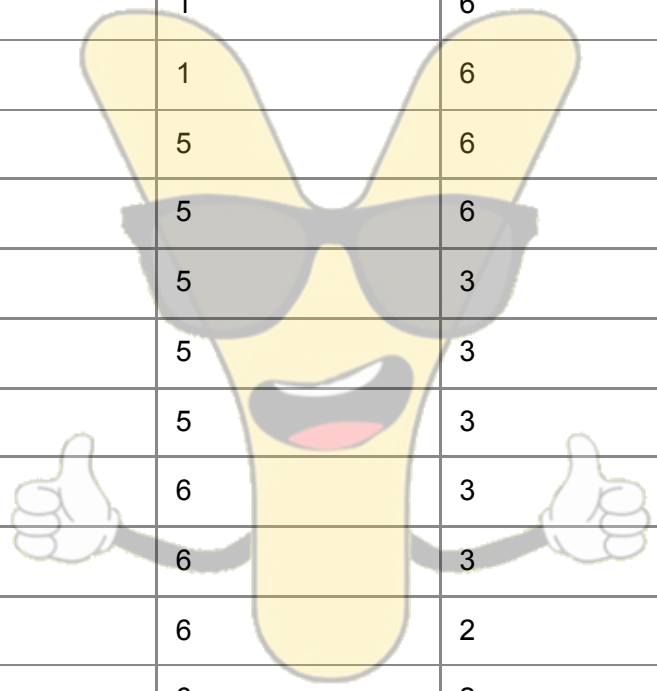
Q1. Find the number of page faults for the following input string using FIFO and LRU page replacement algorithms: 1,2,3,2,1,5,2,1,6,2,5,6,3,1,3,6,1,2,4,3. (Consider frame size = 3). (2019)

Reference string (20 references): 1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3 with **3 frames**, all initially empty.

FIFO replaces the oldest page in memory.



Ref	Frame 1	Frame 2	Frame 3	Result
1	1	-	-	**Fault**
2	1	2	-	**Fault**
3	1	2	3	**Fault**
2	1	2	3	Hit
1	1	2	3	Hit
5	5	2	3	**Fault**
2	5	2	3	Hit
1	5	1	3	**Fault**
6	5	1	6	**Fault**
2	2	1	6	**Fault**
5	2	5	6	**Fault**
6	2	5	6	Hit
3	2	5	3	**Fault**
1	1	5	3	**Fault**
3	1	5	3	Hit
6	1	6	3	**Fault**
1	1	6	3	Hit
2	1	6	2	**Fault**
4	4	6	2	**Fault**
3	4	3	2	**Fault**



FIFO page faults = 14 (hits = 6).

LRU replaces the page that has not been used for the longest time in the past.

Ref	Frame 1	Frame 2	Frame 3	Result
1	1	-	-	**Fault**
2	1	2	-	**Fault**
3	1	2	3	**Fault**
2	1	2	3	Hit
1	1	2	3	Hit
5	1	2	5	**Fault**
2	1	2	5	Hit
1	1	2	5	Hit
6	1	2	6	**Fault**
2	1	2	6	Hit
5	5	2	6	**Fault**
6	5	2	6	Hit
3	5	3	6	**Fault**
1	1	3	6	**Fault**
3	1	3	6	Hit
6	1	3	6	Hit
1	1	3	6	Hit
2	1	2	6	**Fault**
4	1	2	4	**Fault**
3	3	2	4	**Fault**

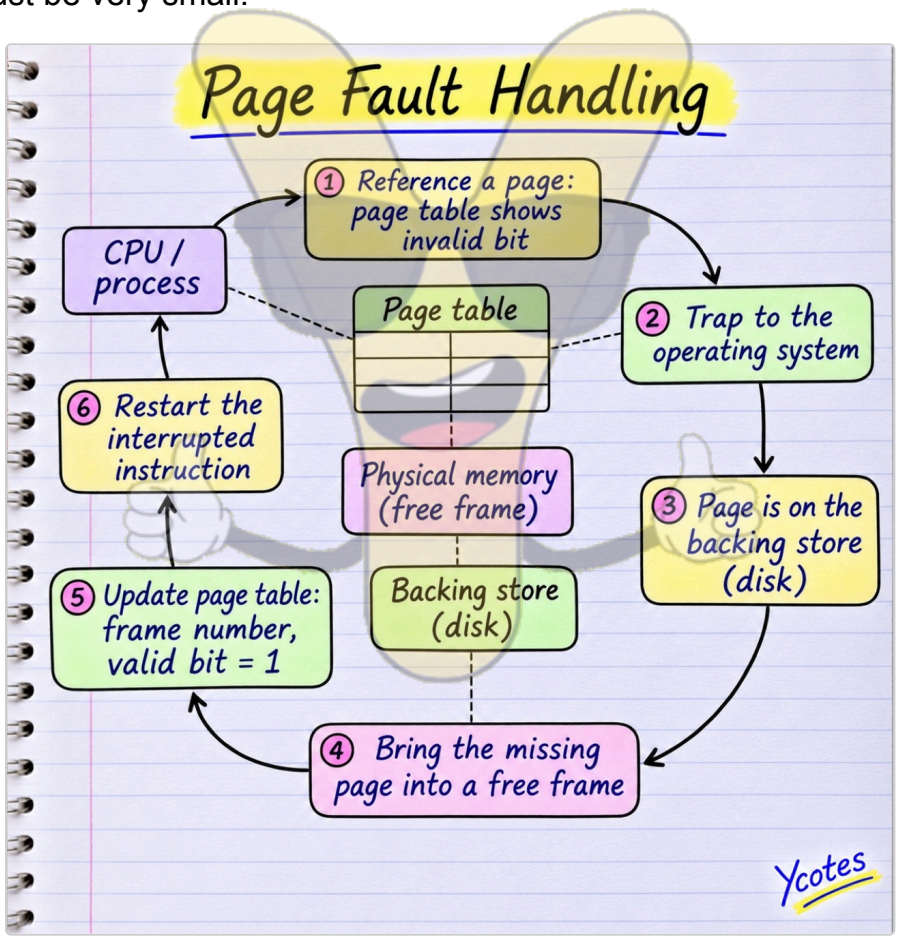
LRU page faults = 11 (hits = 9).

Algorithm	Page faults	Hits	Fault rate
FIFO	14	6	70%
LRU	11	9	55%
Optimal (for comparison)	9	11	45%

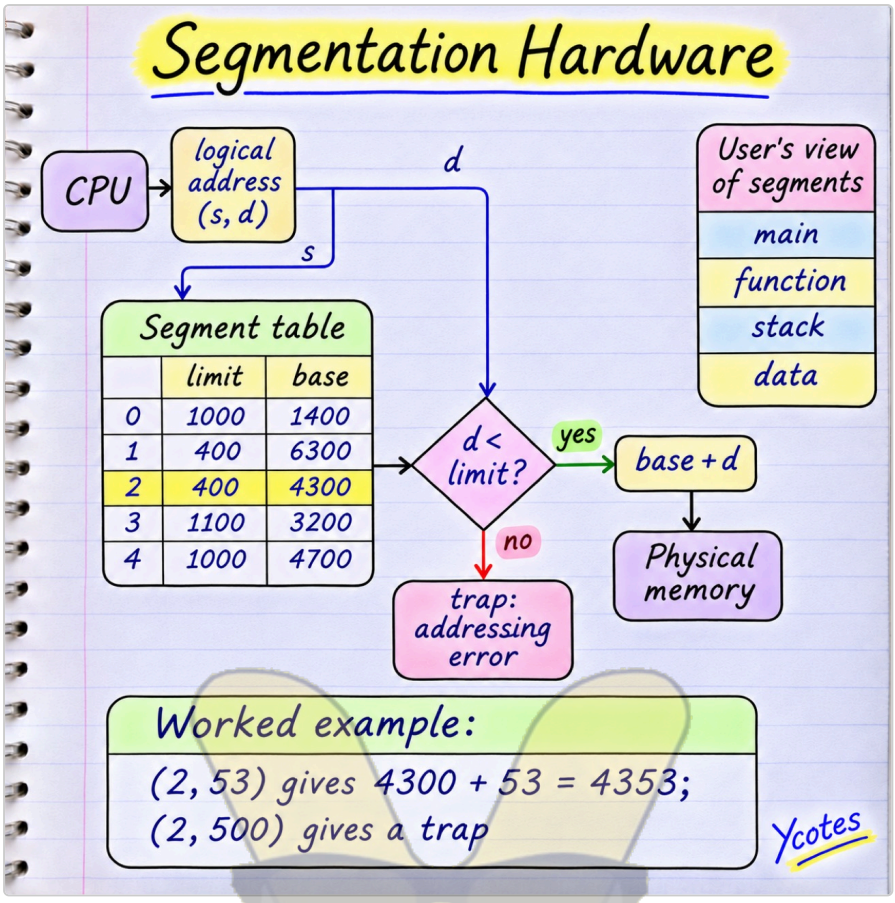
LRU (11) performs better than FIFO (14) and is close to the optimal result (9).

Q2. Explain the following - (a) Demand paging (b) Segmentation with paging scheme (c) Thrashing (d) Global versus local allocation. (2021)

(a) Demand paging: a page is brought into memory only when it is referenced (lazy loading). A valid-invalid bit in the page table shows whether the page is in memory. A reference to an invalid page causes a **page fault**: the OS finds a free frame (or a victim), reads the page from the disk, updates the page table and restarts the instruction. In **pure demand paging** a process starts with no page in memory. Its performance is given by $EAT = (1-p)m + pT_f$. Advantages: less I/O, less memory per process, more processes in memory and faster start. Disadvantage: the fault rate must be very small.

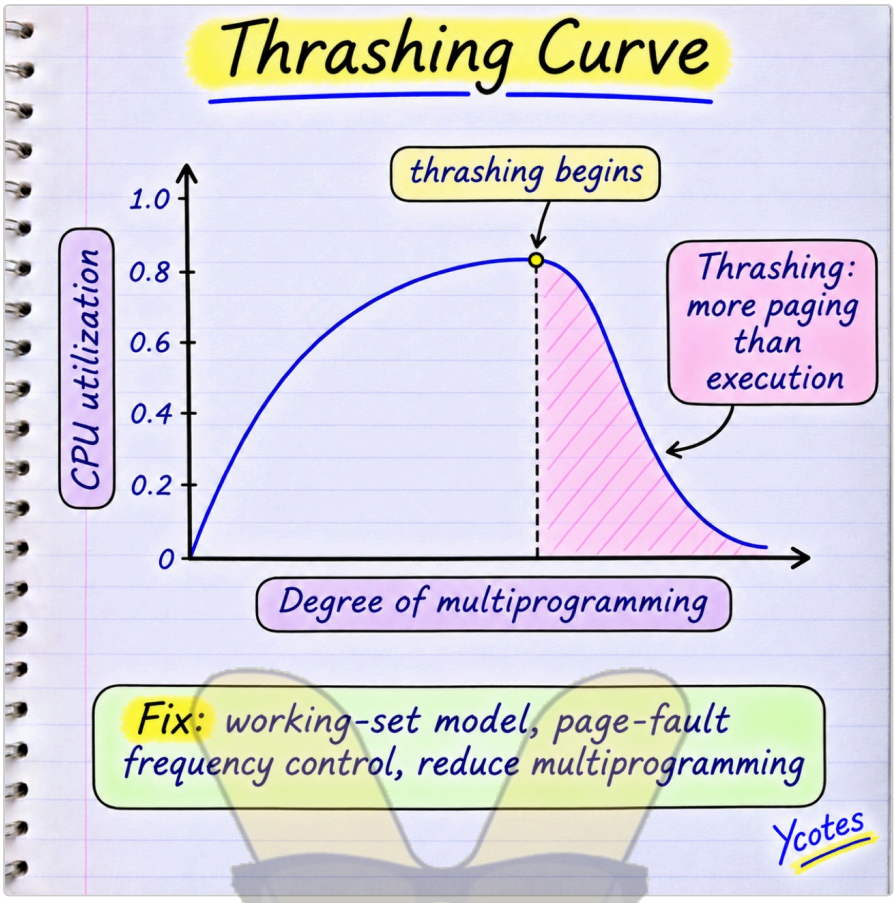


(b) Segmentation with paging: each segment is divided into fixed-size pages. The logical address (s, d) uses the **segment table** to find the page table of segment s; the offset d is split into (page number p, page offset d'). The segment's page table gives the frame number f, and the physical address is (f, d').



It keeps the user's view of segments, gives protection and sharing per segment, and removes **external fragmentation** because memory is allocated in frames. The disadvantages are the extra table lookups (two memory references more, reduced by the TLB) and some internal fragmentation in the last page of each segment. It is used in MULTICS and Intel x86.

(c) Thrashing: the system spends more time paging than executing. When there are too many processes, each gets too few frames, so the page-fault rate becomes very high and the CPU utilisation drops. The OS then admits more processes, which makes it worse.



Remedies: working-set model, page-fault frequency control, reducing the degree of multiprogramming, local replacement and more RAM.

(d) Global versus local allocation:

Global replacement	**Local replacement**
Victim chosen from **all** frames, including those of other processes	Victim chosen only from the process's **own** frames
Better system throughput	Performance of a process depends only on itself
One process can take frames from another, so its fault rate depends on others	Number of frames of a process stays fixed, so memory may be underused
Can cause thrashing to spread	Isolates thrashing to the faulting process

---

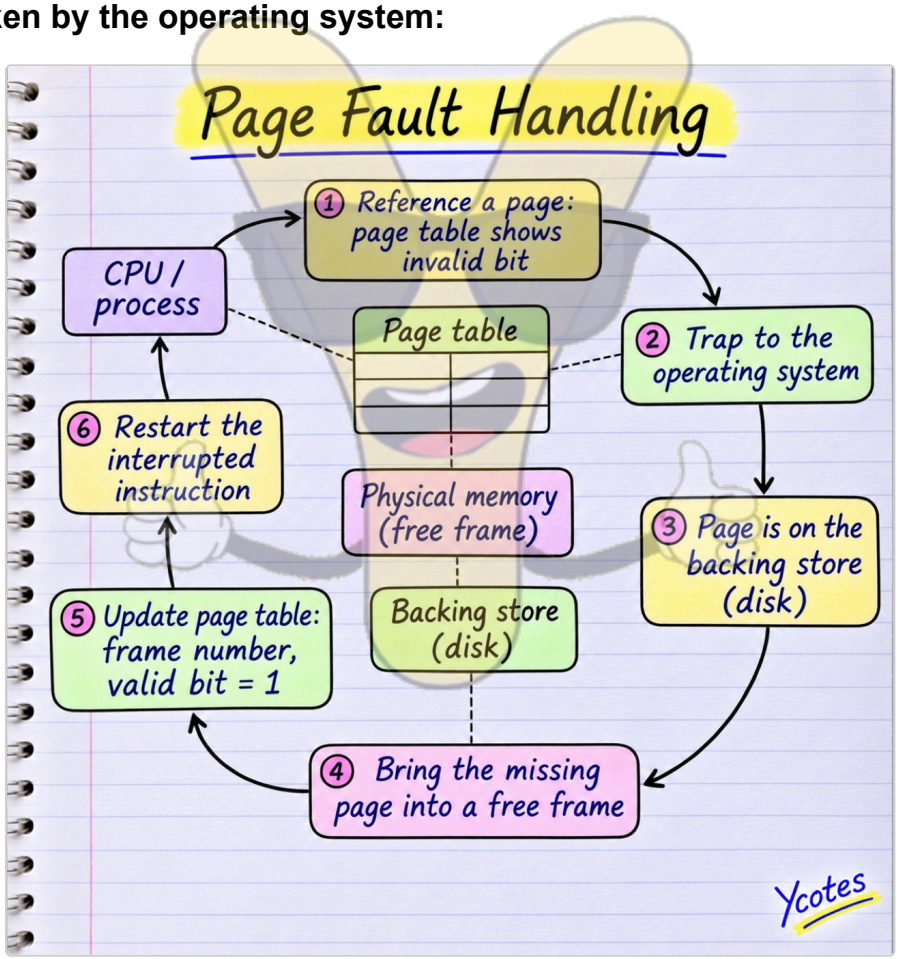
Q3. Under what circumstances do page fault occurs? Describe the actions taken by the operating system when a page fault occurs. (2021)

A **page fault** occurs when a process references a page that is **not in main memory**, that is, the valid-invalid bit in its page-table entry is invalid (0).

Circumstances in which a page fault occurs:

1. **First reference** to a page in demand paging, because pages are loaded only when needed.
2. The page was **swapped out** (replaced) earlier to make room for another page.
3. The process makes an **illegal reference** outside its address space (this ends the process).

Actions taken by the operating system:



1. The hardware traps to the OS on the invalid bit; the registers and process state are saved.
2. The OS checks an internal table (kept with the PCB). If the reference is **illegal**, the process is terminated. If it is **legal but the page is not in memory**, the OS goes on.

3. The OS finds a **free frame** in the free-frame list. If there is none, a **page-replacement algorithm** selects a victim; if the victim page is modified (dirty bit set) it is written back to the disk first.
4. A **disk read** operation is scheduled to bring the required page into the frame. Meanwhile the CPU is given to another process.
5. When the read is complete, the OS updates the **page table** (frame number, valid bit = 1) and the internal tables.
6. The process is put in the ready queue, and when it runs the **instruction that caused the fault is restarted**. It now finds the page in memory.

Time taken: servicing a fault takes several milliseconds, mostly the disk read, so the effective access time is $EAT = (1 - p)\, m + p\, T_f$. A very low fault rate is therefore needed. The fault rate is kept low by locality of reference and good replacement algorithms; a high rate leads to thrashing.

Q4. Consider the following page reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Compare the number of page faults with frame size 3, 4 with FIFO & LRU page replacement algorithms. Also explain Belady's anomaly in detail. (2022)

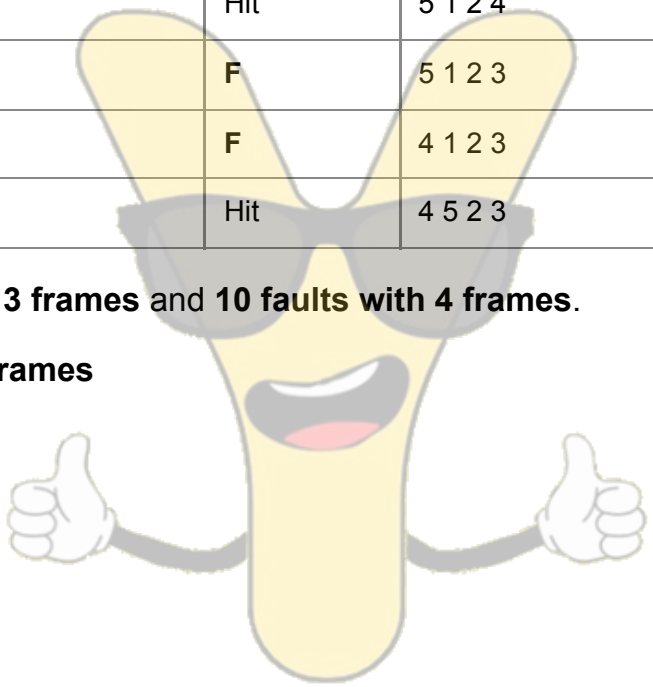
Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 (12 references), frames initially empty, **F** = page fault.

FIFO with 3 and 4 frames

Ref	FIFO, 3 frames	Result	FIFO, 4 frames	Result
1	1 - -	**F**	1 - - -	**F**
2	1 2 -	**F**	1 2 - -	**F**
3	1 2 3	**F**	1 2 3 -	**F**
4	4 2 3	**F**	1 2 3 4	**F**
1	4 1 3	**F**	1 2 3 4	Hit
2	4 1 2	**F**	1 2 3 4	Hit
5	5 1 2	**F**	5 2 3 4	**F**
1	5 1 2	Hit	5 1 3 4	**F**
2	5 1 2	Hit	5 1 2 4	**F**
3	5 3 2	**F**	5 1 2 3	**F**
4	5 3 4	**F**	4 1 2 3	**F**
5	5 3 4	Hit	4 5 2 3	**F**

FIFO: **9 faults with 3 frames** and **10 faults with 4 frames**.

LRU with 3 and 4 frames

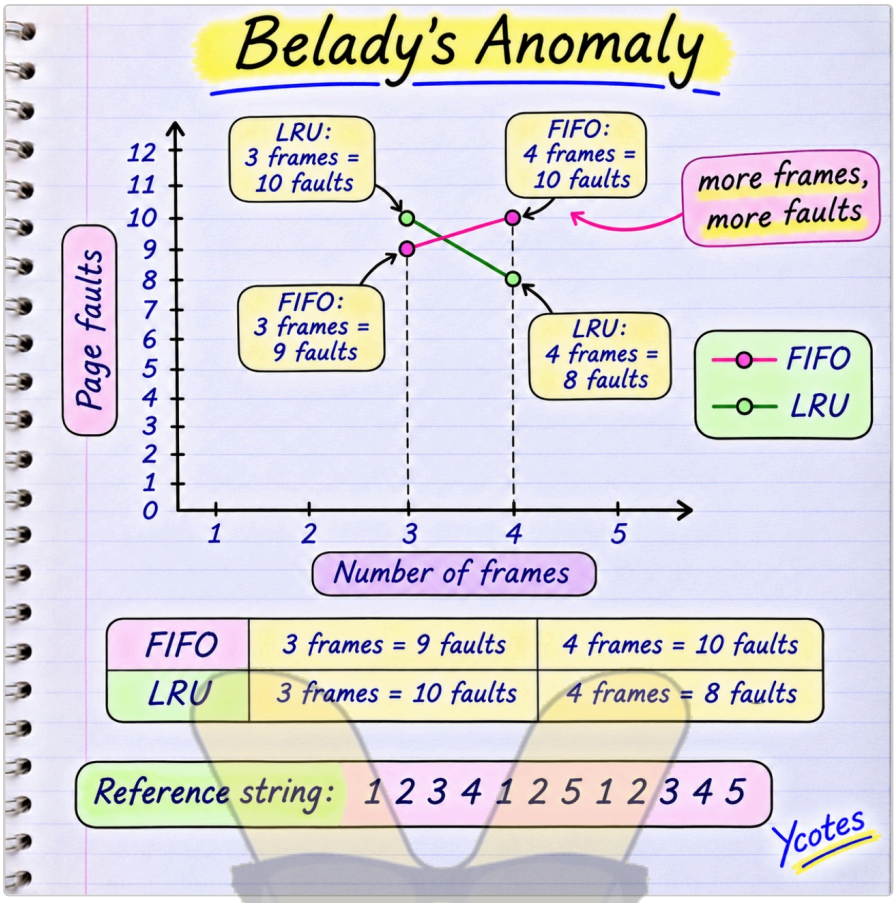


Ref	LRU, 3 frames	Result	LRU, 4 frames	Result
1	1 - -	**F**	1 - - -	**F**
2	1 2 -	**F**	1 2 - -	**F**
3	1 2 3	**F**	1 2 3 -	**F**
4	4 2 3	**F**	1 2 3 4	**F**
1	4 1 3	**F**	1 2 3 4	Hit
2	4 1 2	**F**	1 2 3 4	Hit
5	5 1 2	**F**	1 2 5 4	**F**
1	5 1 2	Hit	1 2 5 4	Hit
2	5 1 2	Hit	1 2 5 4	Hit
3	3 1 2	**F**	1 2 5 3	**F**
4	3 4 2	**F**	1 2 4 3	**F**
5	3 4 5	**F**	5 2 4 3	**F**

LRU: **10 faults with 3 frames** and **8 faults with 4 frames**.

Frames	FIFO	LRU
3	9	10
4	10	8

Belady's anomaly: normally more frames mean fewer page faults. For FIFO, increasing the frames from 3 to 4 **increased** the faults from 9 to 10. This is **Belady's anomaly**: the page-fault rate may increase when the number of allocated frames increases.



Reason: FIFO is not a **stack algorithm**. The set of pages in memory with n frames is not always a subset of the set with n + 1 frames, because FIFO removes the oldest page regardless of how often it is used. With 4 frames the pages leave in a different order, so useful pages (1 and 2) are evicted just before they are needed again.

LRU and Optimal never show the anomaly, because they are stack algorithms: the pages in n frames are always a subset of those in n + 1 frames. The LRU result above confirms it, since the faults fall from 10 to 8.

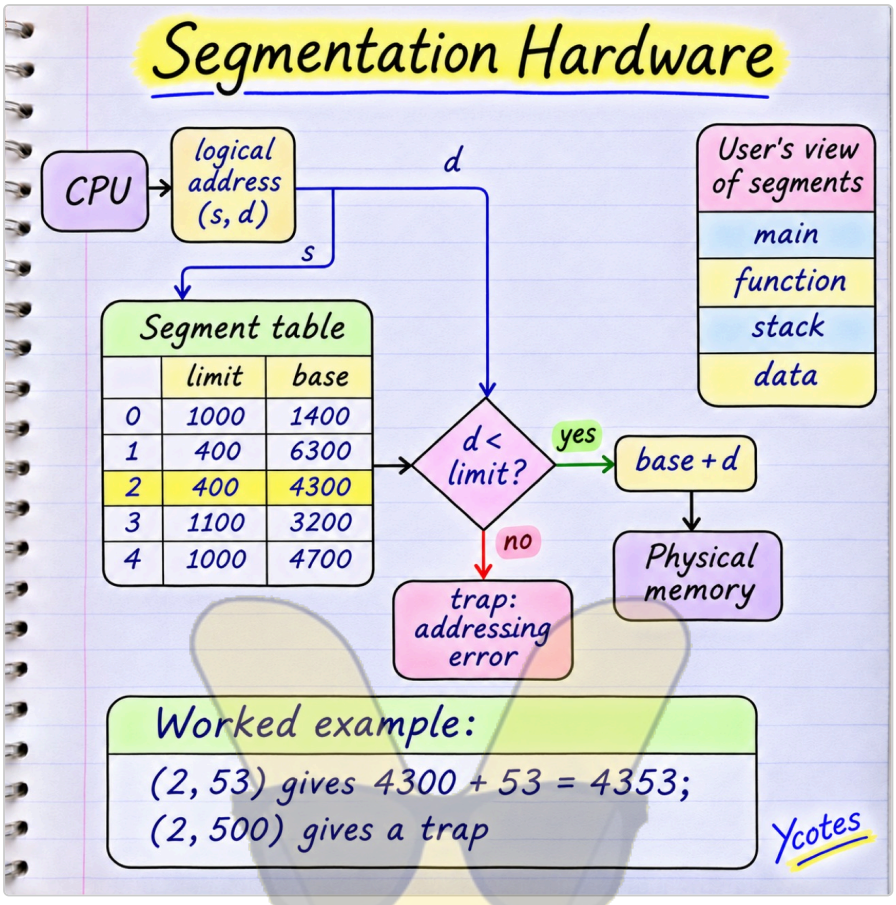
Q5. Explain the followings - (b) Segmentation. (2022)

Segmentation is a memory-management scheme that supports the **user's view** of memory. A program is a collection of logical units called **segments**, such as the main program, functions, global data, stack and symbol table. Each segment has a different length.

Logical address: a pair **(segment number s, offset d)**.

Segment table: one entry for each segment with a **base** (starting physical address) and a **limit** (length of the segment). A segment-table base register (STBR) and

length register (STLR) locate the table.



Address translation:

1. The CPU generates (s, d).
2. The segment number s is used as an index into the segment table.
3. The offset d is compared with limit[s]. If $d \geq$ limit[s], a trap (addressing error) occurs.
4. Otherwise physical address = base[s] + d.

Example: segment 2 has base 4300 and limit 400. The address (2, 53) gives 4300 + 53 = **4353**. The address (2, 500) causes a trap because 500 is greater than the limit 400.

Advantages:

- Matches the logical structure of the program.
- Protection and sharing can be applied to each segment (for example read-only code).
- No internal fragmentation.
- Segments can grow independently.

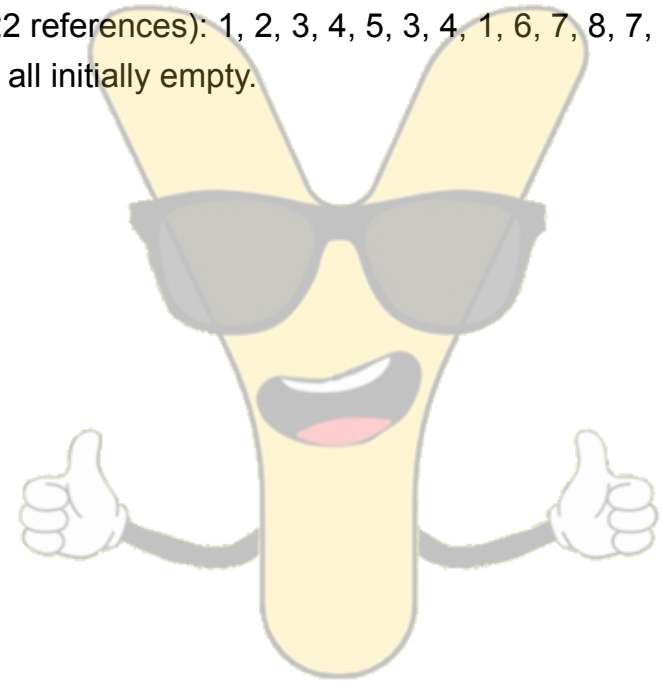
Disadvantages:

- **External fragmentation**, because segments have variable sizes (cured by compaction).
- Allocation of variable-sized blocks is complex.
- Each reference needs a segment-table access, so a cache of table entries is needed.

Q6. Consider the following page reference string 1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2 with four frames. How many page faults would occur for the FIFO, Optimal Page Replacement Algorithm. (Assume all frames are initially empty) (2023)

Reference string (22 references): 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2 with **4 frames**, all initially empty.

FIFO



Ref	Frame 1	Frame 2	Frame 3	Frame 4	Result
1	1	-	-	-	**Fault**
2	1	2	-	-	**Fault**
3	1	2	3	-	**Fault**
4	1	2	3	4	**Fault**
5	5	2	3	4	**Fault**
3	5	2	3	4	Hit
4	5	2	3	4	Hit
1	5	1	3	4	**Fault**
6	5	1	6	4	**Fault**
7	5	1	6	7	**Fault**
8	8	1	6	7	**Fault**
7	8	1	6	7	Hit
8	8	1	6	7	Hit
9	8	9	6	7	**Fault**
7	8	9	6	7	Hit
8	8	9	6	7	Hit
9	8	9	6	7	Hit
5	8	9	5	7	**Fault**
4	8	9	5	4	**Fault**
5	8	9	5	4	Hit
4	8	9	5	4	Hit
2	2	9	5	4	**Fault**

FIFO page faults = 13 (hits = 9).

Optimal (replace the page whose next use is farthest in the future)

Ref	Frame 1	Frame 2	Frame 3	Frame 4	Result
1	1	-	-	-	**Fault**
2	1	2	-	-	**Fault**
3	1	2	3	-	**Fault**
4	1	2	3	4	**Fault**
5	1	5	3	4	**Fault**
3	1	5	3	4	Hit
4	1	5	3	4	Hit
1	1	5	3	4	Hit
6	6	5	3	4	**Fault**
7	7	5	3	4	**Fault**
8	7	5	8	4	**Fault**
7	7	5	8	4	Hit
8	7	5	8	4	Hit
9	7	5	8	9	**Fault**
7	7	5	8	9	Hit
8	7	5	8	9	Hit
9	7	5	8	9	Hit
5	7	5	8	9	Hit
4	4	5	8	9	**Fault**
5	4	5	8	9	Hit
4	4	5	8	9	Hit
2	2	5	8	9	**Fault**

Optimal page faults = 11 (hits = 11).

Algorithm	Page faults	Hits
FIFO	13	9
Optimal	11	11

Brought to you By Ycotes : https://theycotes.com

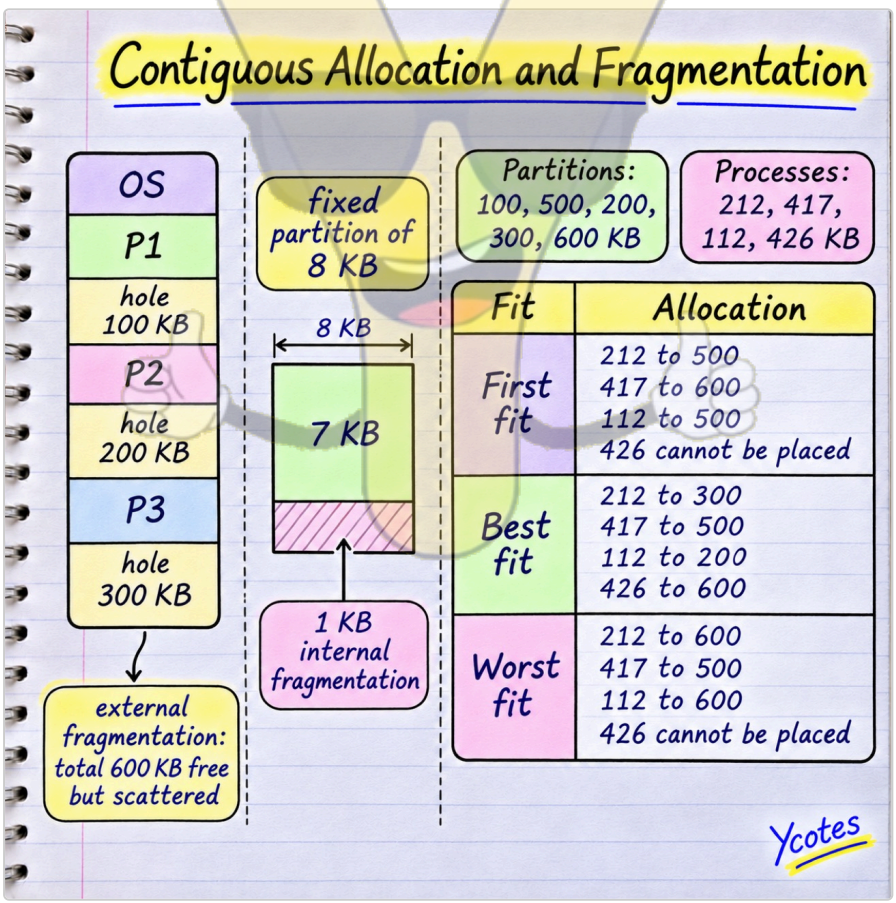
Optimal gives the fewest faults, but it needs knowledge of future references and is therefore not implementable; it is used as a benchmark for other algorithms.

Q7. Short notes - (i) Internal Vs External Fragmentation (ii) Paging and segmentation. (2023)

(i) Internal versus external fragmentation

Internal fragmentation is the wasted memory **inside** an allocated block. It occurs when a process gets a block (partition or page) larger than it needs, for example a 7 KB process in an 8 KB fixed partition wastes 1 KB. It occurs in fixed partitions and in the last page of a paging system.

External fragmentation is the wasted free memory **between** allocated blocks. Total free memory is enough for a request but it is divided into small non-contiguous holes. It occurs in variable partitions and segmentation.

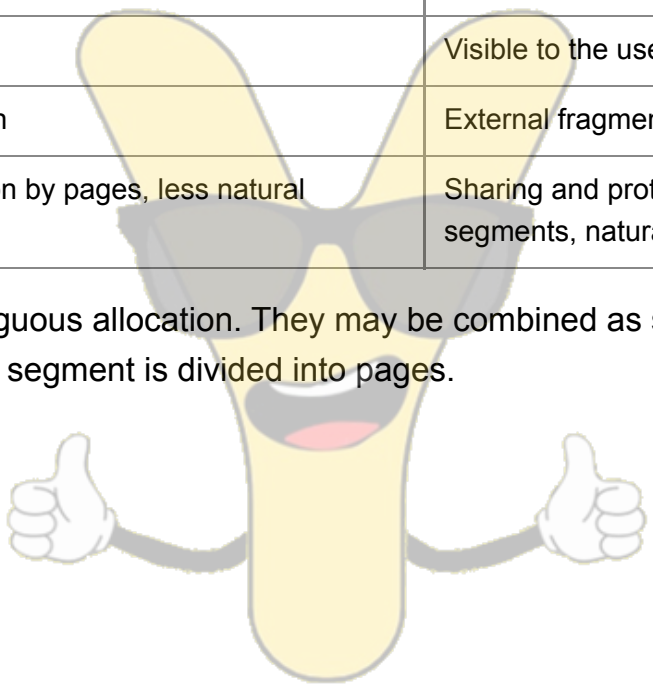


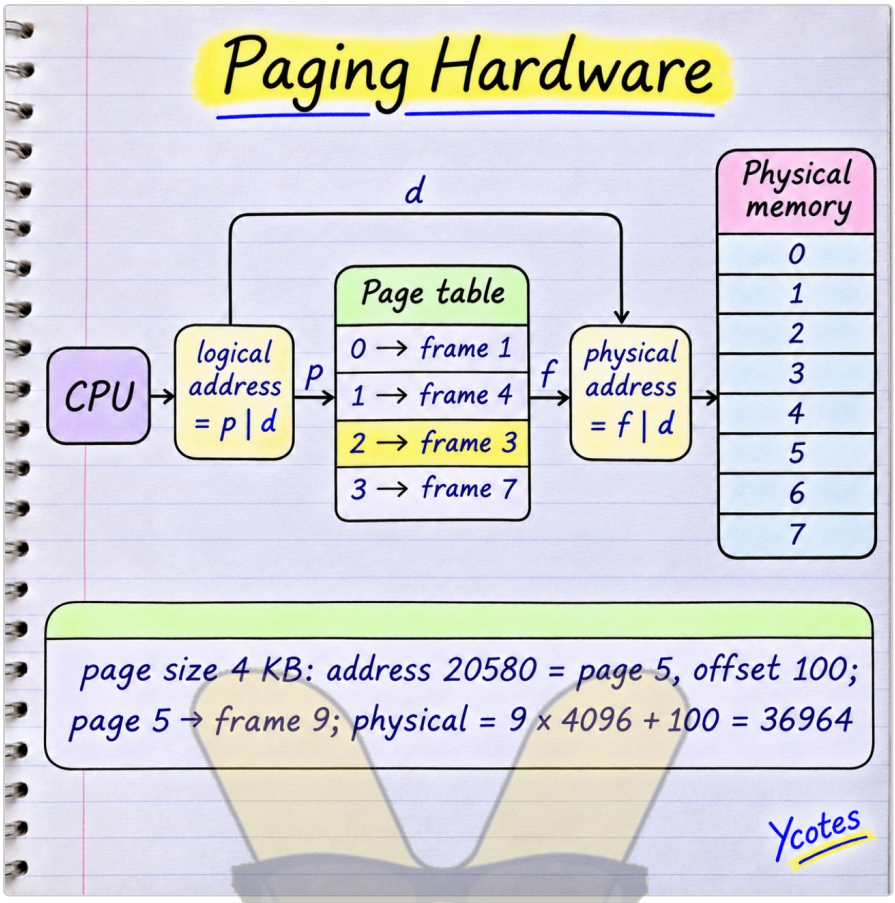
Internal fragmentation	External fragmentation
Waste inside an allocated block	Waste between allocated blocks
Fixed partitions, paging	Variable partitions, segmentation
Remedy: smaller or variable-sized blocks	Remedy: compaction, paging, segmentation

(ii) Paging and segmentation

Paging	Segmentation
Physical memory divided into fixed-size frames, logical memory into pages	Memory divided into variable-size logical segments
Address = (p, d), page table gives frame f	Address = (s, d), segment table gives base and limit
Invisible to the user	Visible to the user
Internal fragmentation	External fragmentation
Sharing and protection by pages, less natural	Sharing and protection by logical segments, natural

Both give non-contiguous allocation. They may be combined as segmentation with paging, where each segment is divided into pages.





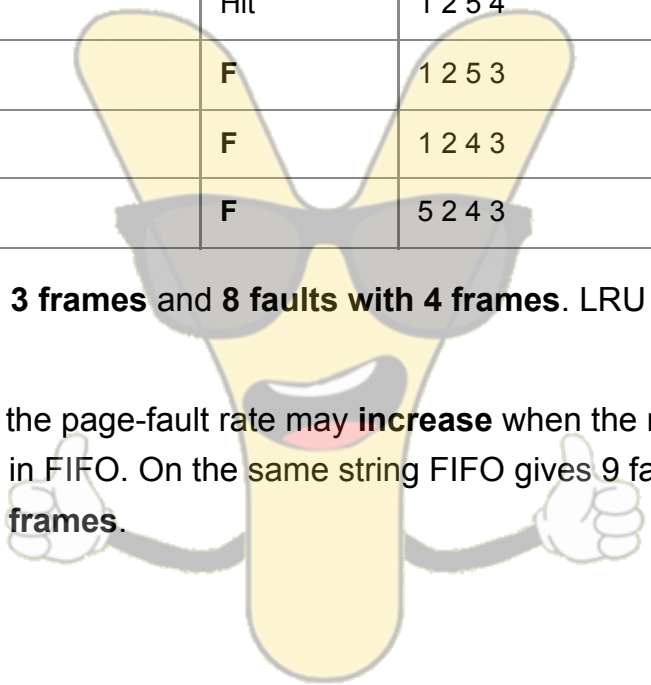
Q8. Consider the following page reference string 1,2,3,4,1,2,5,1,2,3,4,5 compare the number of page faults with frame size 3,4 with LRU page replacement algorithm. Also explain Belady's anomaly in detail. (Feb 2024)

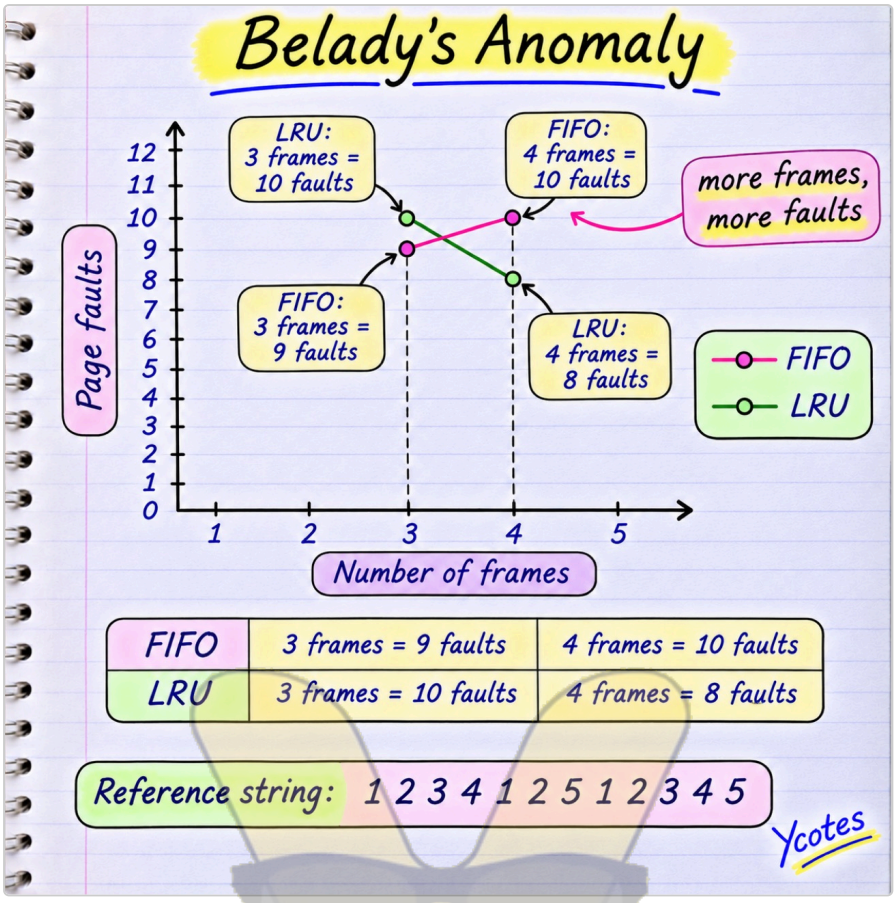
Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 with **LRU**, frames initially empty, **F** = page fault.

Ref	LRU, 3 frames	Result	LRU, 4 frames	Result
1	1 - -	**F**	1 - - -	**F**
2	1 2 -	**F**	1 2 - -	**F**
3	1 2 3	**F**	1 2 3 -	**F**
4	4 2 3	**F**	1 2 3 4	**F**
1	4 1 3	**F**	1 2 3 4	Hit
2	4 1 2	**F**	1 2 3 4	Hit
5	5 1 2	**F**	1 2 5 4	**F**
1	5 1 2	Hit	1 2 5 4	Hit
2	5 1 2	Hit	1 2 5 4	Hit
3	3 1 2	**F**	1 2 5 3	**F**
4	3 4 2	**F**	1 2 4 3	**F**
5	3 4 5	**F**	5 2 4 3	**F**

LRU: **10 faults with 3 frames** and **8 faults with 4 frames**. LRU improves with more frames.

Belady's anomaly: the page-fault rate may **increase** when the number of frames increases. It is seen in FIFO. On the same string FIFO gives 9 faults with 3 frames but **10 faults with 4 frames**.





Frames	FIFO	LRU
3	9	10
4	10	8

Reason: FIFO is not a stack algorithm. The pages kept in memory with n frames are not always a subset of those kept with n + 1 frames, because FIFO ignores how recently a page was used and may evict a page that is about to be used again. **LRU and Optimal are stack algorithms**, so they never show the anomaly; their faults never rise when frames are added.

Q9. Consider a paging system with the page table stored in memory. (a) If a memory reference takes 200 nanoseconds, how long does a paged memory reference take? (b) If we add TLBs and 75 percent of all page-table references are found in the TLBs, what is the effective memory

reference time? Assume that finding a page-table entry in the TLBs takes zero time, if the entry is there. (Feb 2024)

Assumption: each page-table reference needs one memory access; the TLB search time is zero as given.

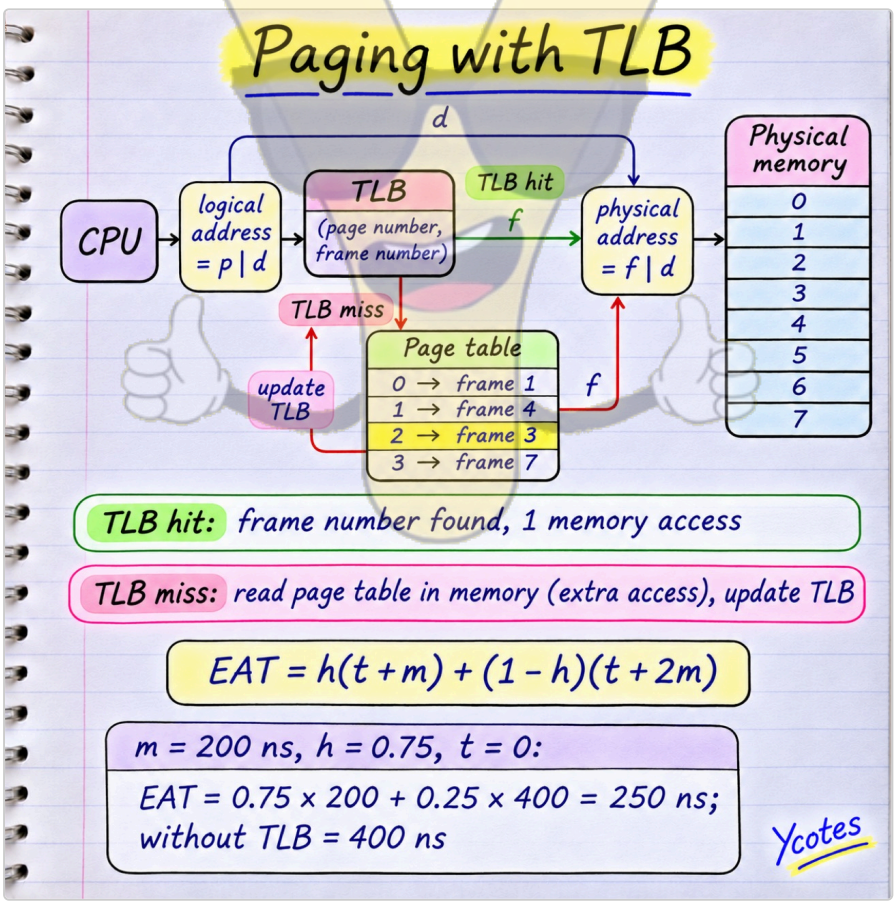
(a) Paged memory reference time (no TLB): one memory access reads the page-table entry and a second reads the data.

$$200 + 200 = 400 \text{ ns}$$

(b) With a TLB, hit ratio 75%:

- TLB hit (75%): the frame number is found at zero time, so only the data access is needed: 200 ns.
- TLB miss (25%): page table access plus data access: 200 + 200 = 400 ns.

$$EAT = 0.75 \times 200 + 0.25 \times 400 = 150 + 100 = 250 \text{ ns}$$

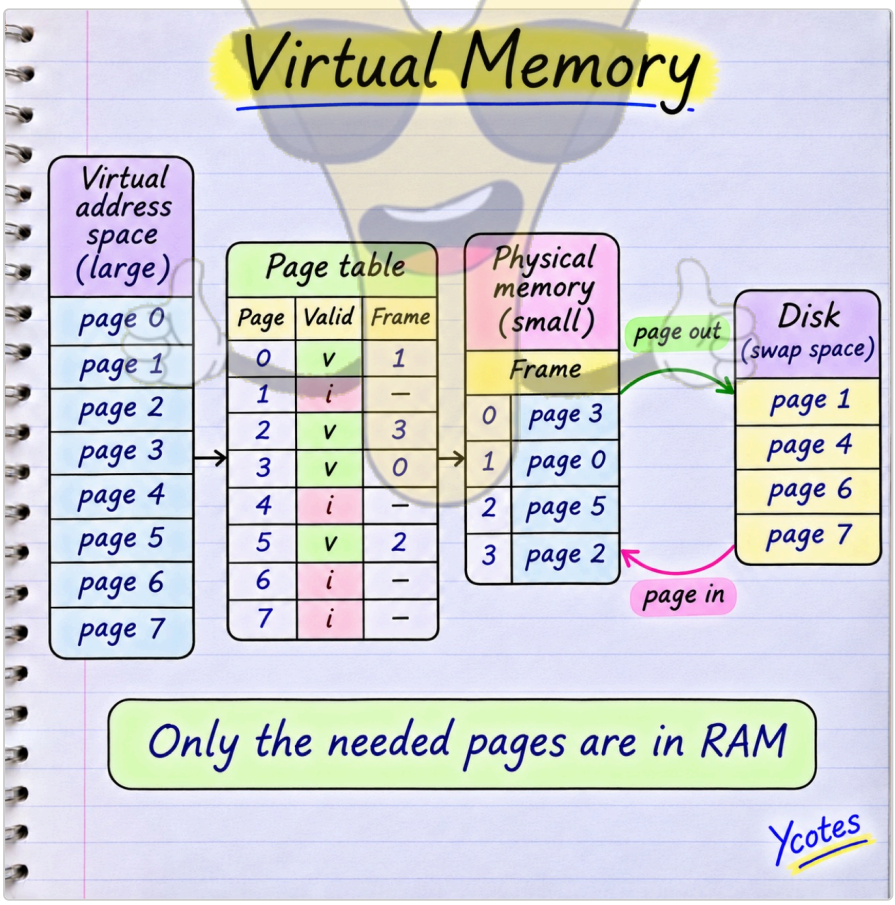


Case	Time
Paging without TLB	400 ns
Paging with TLB (75% hits)	**250 ns**

The TLB reduces the reference time from 400 ns to 250 ns, a saving of $(400 - 250)/400 = 37.5\%$.

Q10. What is Virtual Memory? How it is different from Cache memory and secondary memory? Also discuss the benefits of virtual memory techniques. [6] (2024)

Virtual memory is a technique that lets a process execute although it is only partly in main memory. The logical address space can be much larger than the physical memory. The OS keeps the unused parts of the program in the swap space of the disk and brings pages into memory when needed (demand paging).



Difference from cache memory and secondary memory:

Virtual memory	Cache memory	Secondary memory
Gives the illusion of a very large main memory	Small, fast memory between CPU and RAM	Permanent storage such as hard disk or SSD
Managed by the OS with the MMU	Managed by hardware	Managed by the file system
Part of the address space, backed by the disk	Keeps copies of recently used data and instructions	Keeps files, programs and the swap area
Unit is a page	Unit is a cache line	Unit is a block or sector
Purpose: run programs larger than RAM	Purpose: reduce the average access time	Purpose: long-term storage

Benefits of virtual memory:

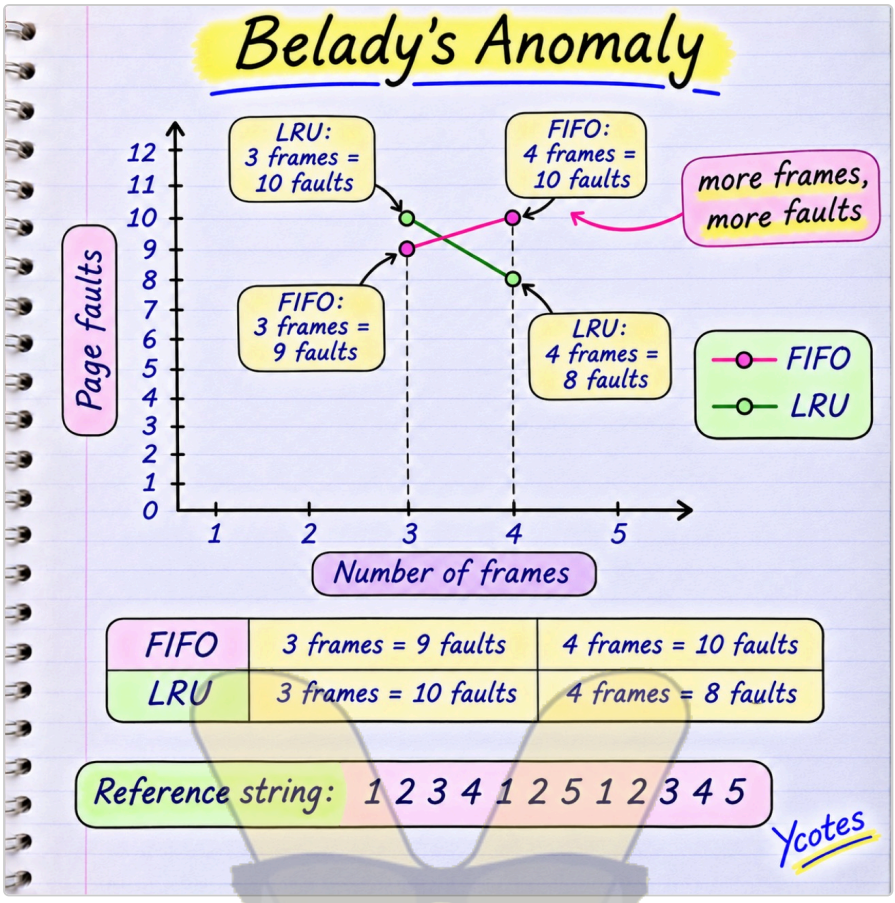
1. Programs **larger than physical memory** can run.
2. **More processes** can be kept in memory, so CPU utilisation and throughput increase.
3. Less I/O is needed than swapping whole processes.
4. Each process has its **own address space**, which gives protection.
5. Easy sharing of pages and libraries between processes.
6. Programming is simpler, because the programmer need not worry about the memory size.

Drawback: too many page faults cause **thrashing**; it is made to work by locality of reference.

Q11. Short notes - (c) Belady Anomaly (d) Principle of locality of reference. (2024)

(c) Belady's anomaly: in some page-replacement algorithms the number of page faults **increases when the number of allocated frames increases**, which is against the expectation that more memory means fewer faults. It occurs in **FIFO**.

Example: reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 under FIFO gives **9 faults with 3 frames** and **10 faults with 4 frames**.



Reason: FIFO is not a stack algorithm: the set of pages in memory with n frames is not always a subset of those with n + 1 frames. FIFO removes the oldest page even if it is used often. **LRU and Optimal never show it**, because they are stack algorithms. It is avoided by using LRU or an approximation of it.

(d) Principle of locality of reference: programs tend to reference a **small set of pages repeatedly** during any short interval.

- **Temporal locality:** a location referenced now will probably be referenced again soon (loops, counters).
- **Spatial locality:** locations near the one just referenced will probably be used next (arrays, sequential instructions).

Importance: locality is the reason why virtual memory and caches work. The working set of a process stays in main memory, so page faults are rare and the effective access time stays near the memory speed. It supports demand paging, the working-set model and page-replacement design. If a process is not given enough frames for its current locality, it thrashes.

Q12. Given the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 and 3 page frames, calculate the number of page faults using the LRU page replacement algorithm. (2025)

Assumption: the string is taken as printed, 13 references with 3 frames, all initially empty. LRU replaces the page not used for the longest time.

Ref	Frame 1	Frame 2	Frame 3	Result
7	7	-	-	**Fault**
0	7	0	-	**Fault**
1	7	0	1	**Fault**
2	2	0	1	**Fault**
0	2	0	1	Hit
3	2	0	3	**Fault**
0	2	0	3	Hit
4	4	0	3	**Fault**
2	4	0	2	**Fault**
3	4	3	2	**Fault**
0	0	3	2	**Fault**
3	0	3	2	Hit
2	0	3	2	Hit

Total page faults with LRU = 9 (hits = 13 - 9 = 4). Page fault rate = 9/13 = 69.2%.

Check: on the same string FIFO gives 10 faults and Optimal gives 7 faults, so LRU lies between them.
