CS-3103 : Operating Systems : Sec-A (NB) :

Virtual Memory Management…..

In Virtual Memory systems, the unit of transfer of data from a program lying on the hard disk to the main memory is: **Ans: a)**

a) A page b) An overlay c) A word d) All of the above

Hint: What is overlay? overlaying means "the process of transferring a block of program code or other data into internal memory, replacing what is already stored".

A page fault is generated when the page is not found in the

a) Page Table b) Dirty Page frame list

c) Free page frame list d) None of the above **Ans: a)**

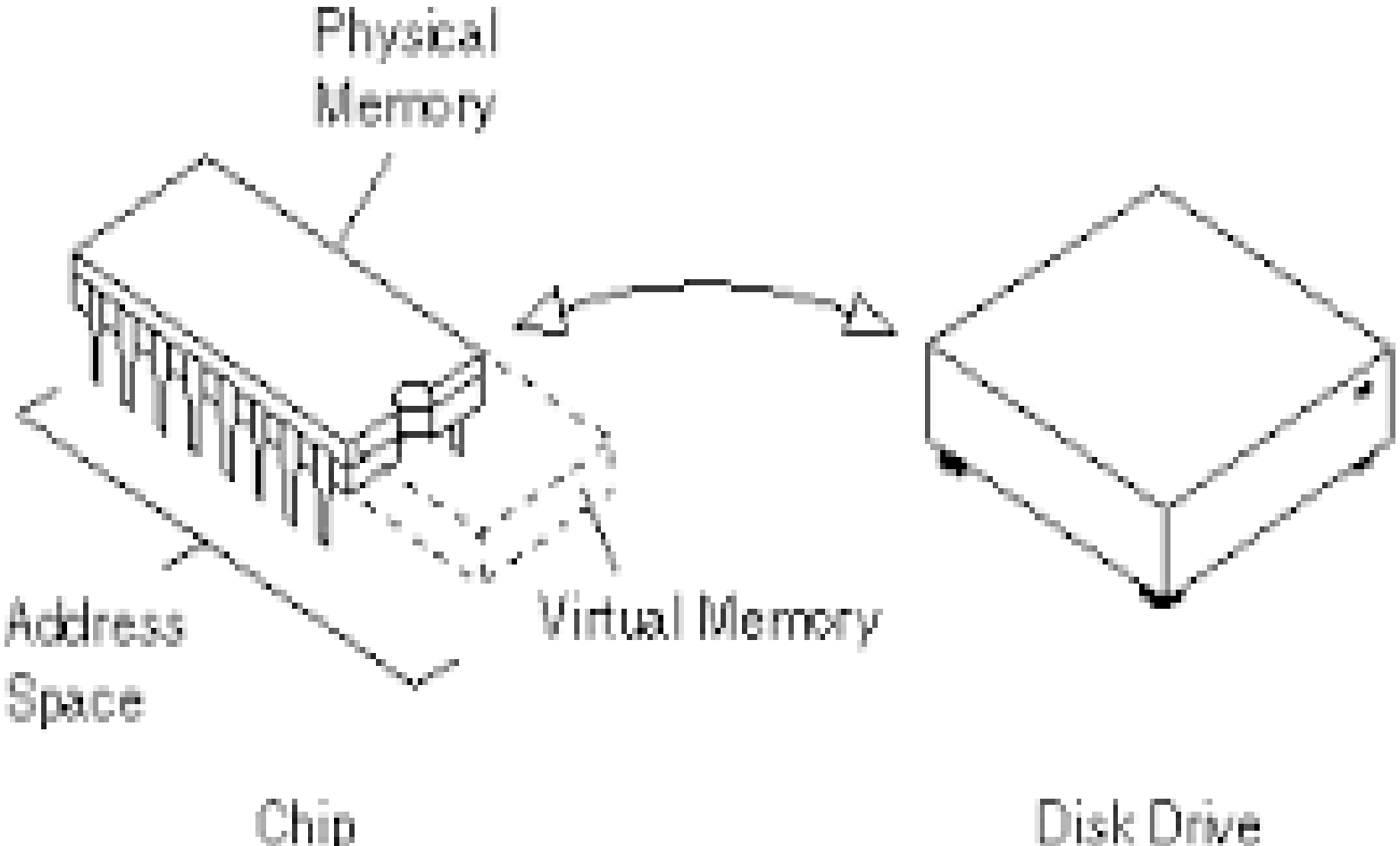
Hint: A **dirty bit** or **modified bit** is a bit that is associated with a block of computer memory and indicates whether or not the corresponding block of memory has been modified. The dirty bit is set when the processor writes to (modifies) this memory. The bit indicates that its associated block of memory has been modified and has not been saved to storage yet. When a block of memory is to be replaced, its corresponding dirty bit is checked to see if the block needs to be written back to secondary memory before being replaced or if it can simply be removed. Dirty bits are used by the CPU cache and in the **page replacement algorithms** of an OS.

page replacement algorithms decide which memory pages to page out, sometimes called swap out, or write to disk, when a page of memory needs to be allocated. Page replacement happens when a requested page is not in memory (page fault).

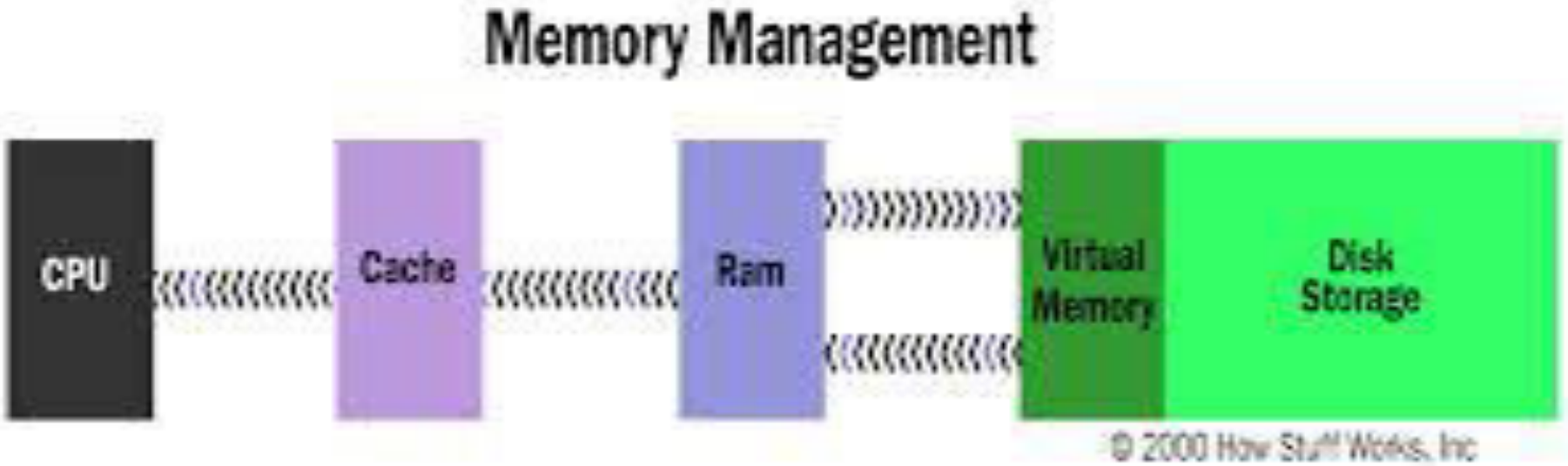
CSEN3103: Virtual Memory

- Background
- Demand Paging
- Performance of Demand Paging
- Page Replacement
- Page-Replacement Algorithms
- Allocation of Frames
- Thrashing
- Demand Segmentation

Virtual Memory



Virtual Memory



Virtual Memory

- The instruction being executed must be in physical memory.

```
loop:
        load        register, index
        add         42, register
        store       register, index
        inc         index
        skip_equal  index, final_address
        branch      loop
            ... continue ....
```

- The instruction to be executed should be both necessary and reasonable.
- Programs often have code to handle unusual error conditions. Since these errors seldom occur in practice, this code is almost never executed.
- Arrays, lists, and tables are often allocated more memory than they actually need. An array may be declared arr[100], but, may be used for the first 10 elements. Rest remains unutilized.

Virtual Memory …..

The ability to execute a program that is only partially in memory has the following advantages:

- A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large ***virtual*** address space, simplifying the programming task.
- Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding *increase* in *CPU utilization* and *throughput* but with no increase in response time or turnaround time.

[Turnaround time is the interval between the submission of a job and its completion.
Response time is the interval between submission of a request, and the first response to that request.]

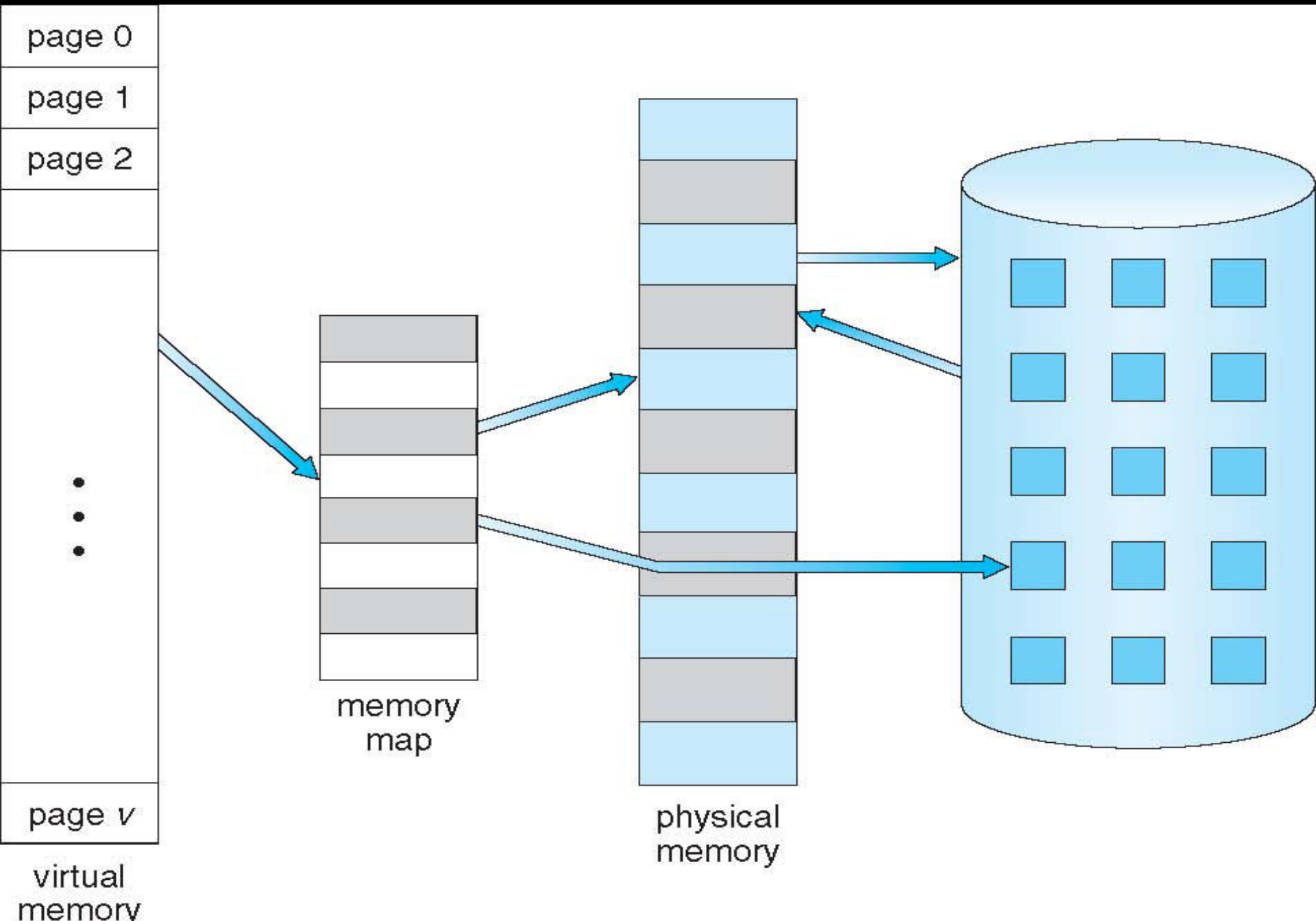
Virtual Memory …..

- Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.

Thus, running a program that is not entirely in memory would benefit both the system & the user:

- ***Virtual Memory (VM)*** involves the separation of logical memory as perceived by users from physical memory.
- This separation allows an ***extremely large VM*** to be provided for programmers when only a smaller physical memory is available.

Virtual Memory That is Larger Than Physical Memory



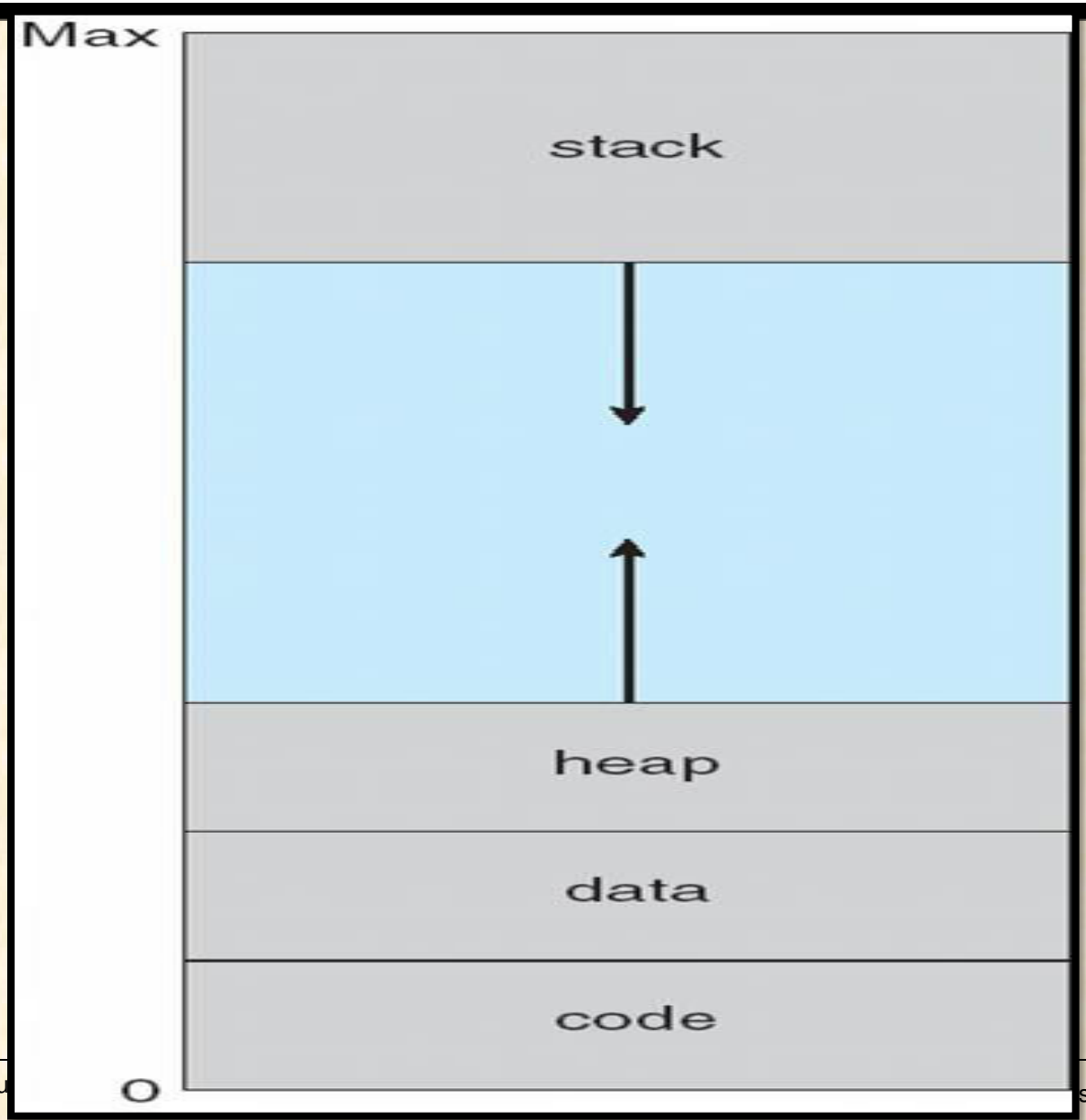
Virtual Memory – key points to remember

- **Virtual memory** – separation of user logical memory from physical memory.
  - ***Only part of the program needs to be in memory for execution.***
  - ***Logical address space can therefore be much larger than physical address space.***
  - ***Need to allow pages to be swapped in and out***.
- **Virtual memory** can be implemented via:
  - ***Demand paging***
  - ***Demand segmentation***

Virtual Address Space

- The ***virtual address space*** of a process refers to the *logical (or virtual) view of how a process is stored in memory. In this view, a process begins at a certain logical address – say, address 0 – and exists in contiguous memory space.*
- The physical memory may be organized in page frames and that the physical page frames assigned to a process may not be contiguous.
- It is up to the memory management unit (MMU) to map logical pages to physical page frames in memory.
- We allow the heap to grow upward in memory as it is used for dynamic memory allocation.
- Similarly, we allow for the stack to grow downward in memory through successive function calls.

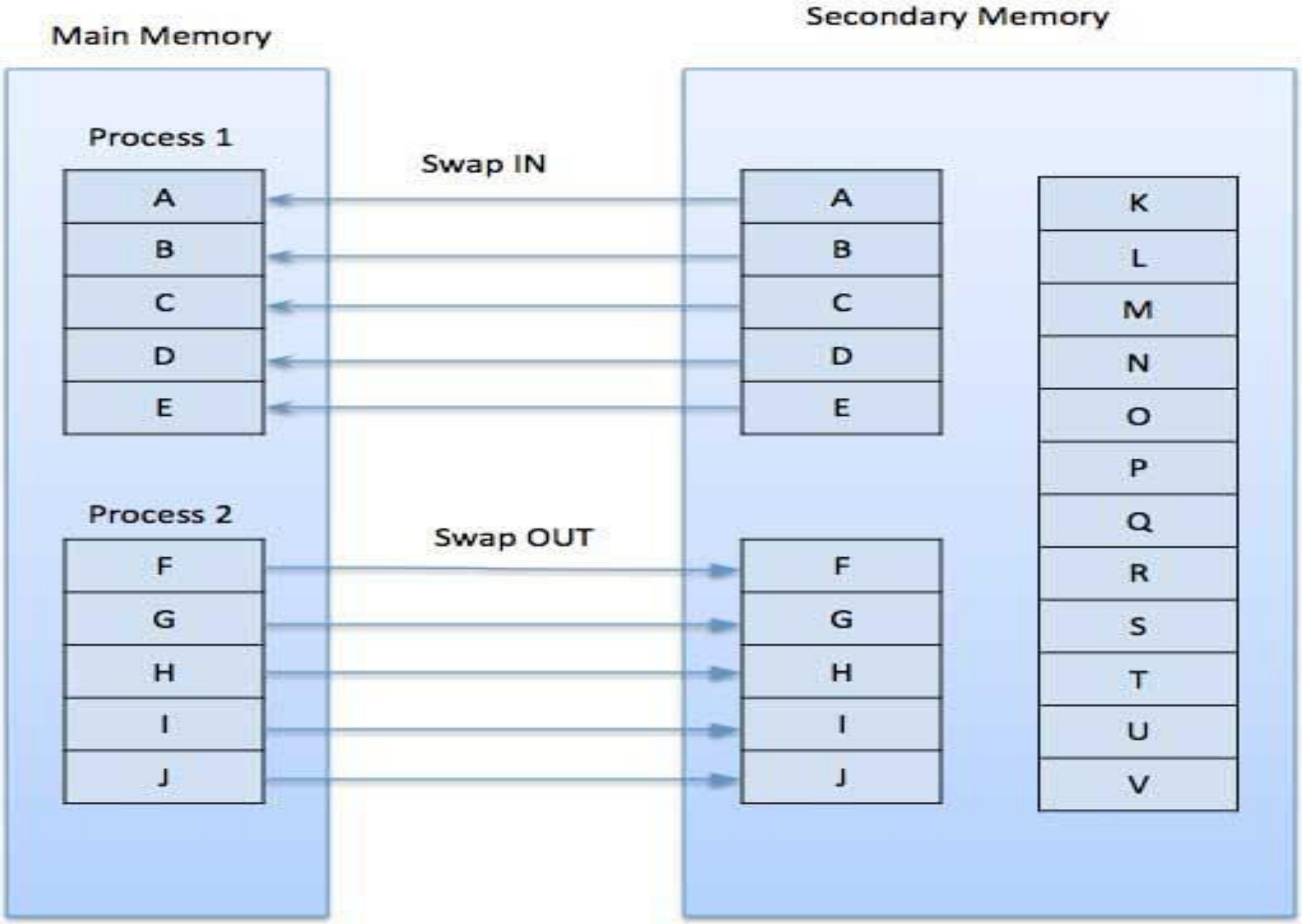
Virtual Address Space



Demand Paging

- ***Bring a page into memory only when it is needed***.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- ***Page is needed*** $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory

Demand Paging



Valid-Invalid Bit

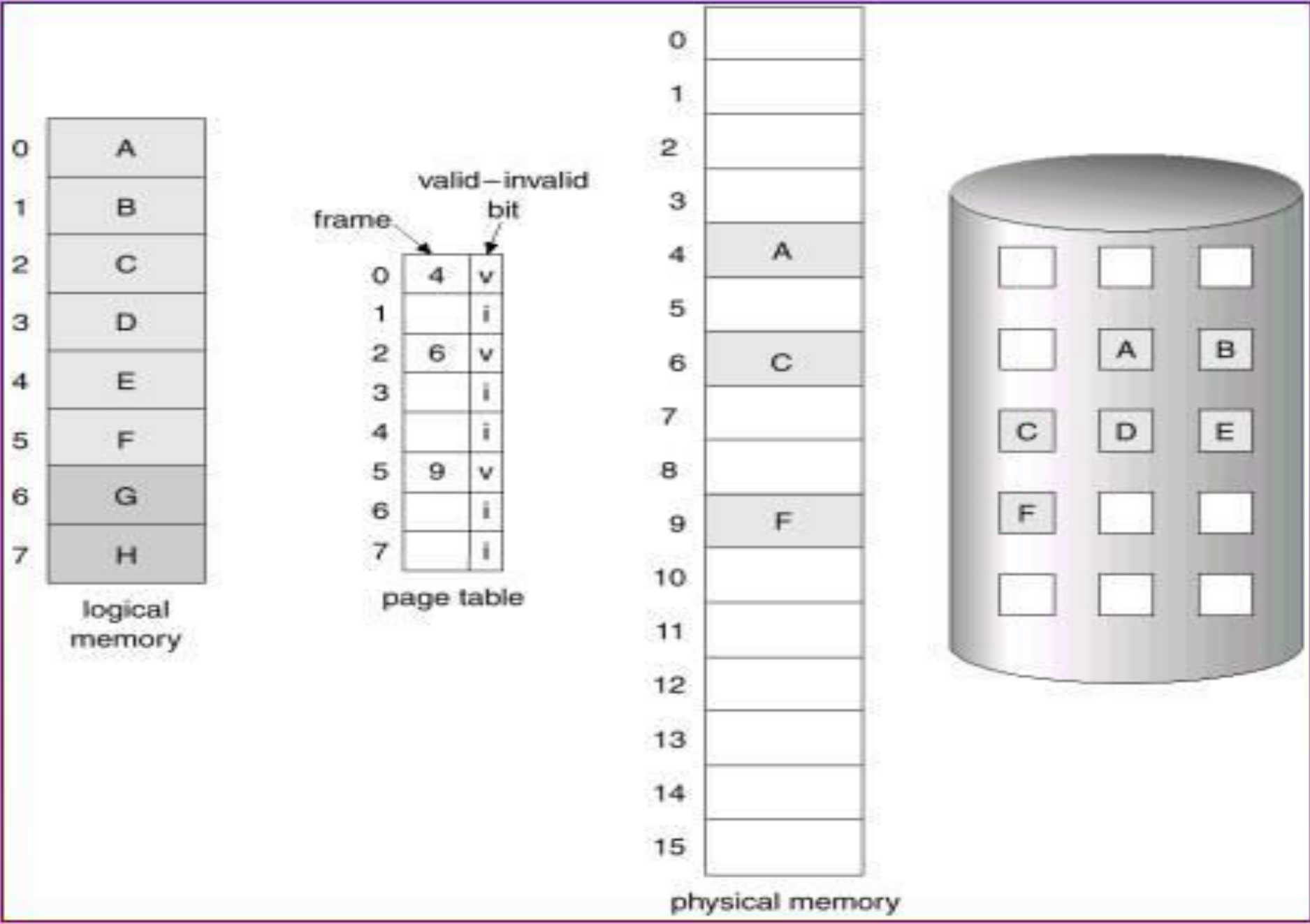
- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot.

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault.

Page table when some pages are not in main memory



Demand Paging

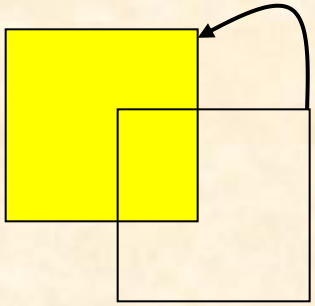
- Processes reside on secondary memory (high-speed disk)
- When process is to be executed, only the needed pages are brought into memory (**lazy swapping**)
- Page table should specify location of pages (memory vs. on-disk)
  - valid/invalid bit may be used
  - for page that is not currently in memory, page table entry may contain address of page on disk
- While process accesses pages resident in memory, execution proceeds normally
- When process accesses page not in memory, paging hardware traps to OS (**page fault**)

Page Fault

1. Check internal table to determine whether reference was to valid / invalid page.
2. Invalid access $\Rightarrow$ terminate process.
3. Find a free frame from the free-frame list.
4. Read the desired page from swap device into the free frame.
5. When I/O is complete, update internal table and page table.
6. Restart the instruction that was interrupted by the illegal address trap.
   (state/context of the process is saved so that process can be restarted in exactly the same state)

Page Fault

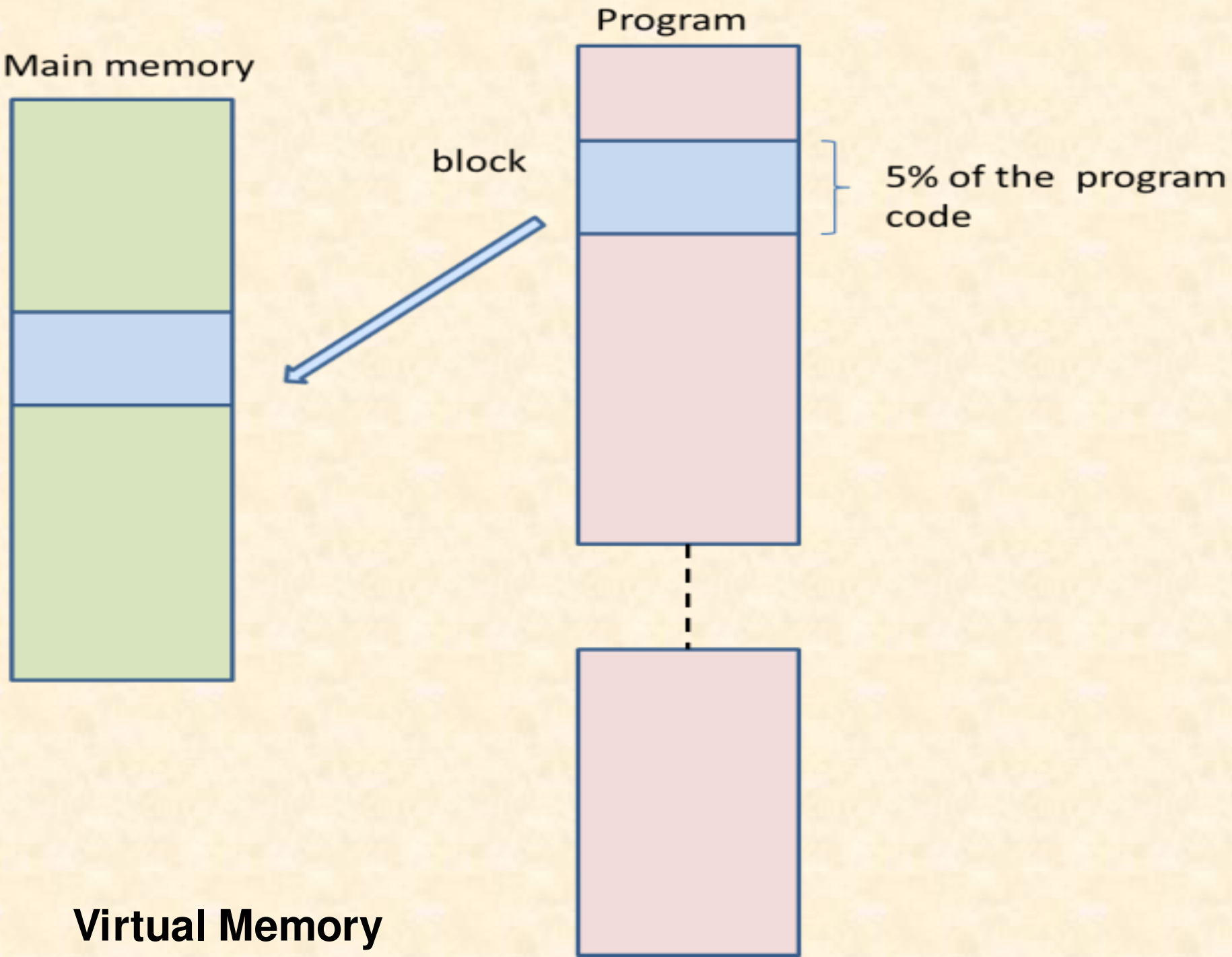
- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction: Least Recently Used
  - block move



  - auto increment/decrement location

1. Fetch and decode the Instruction (ADD)
2. Fetch A
3. Fetch B
4. Add A and B
5. Store the sum in C

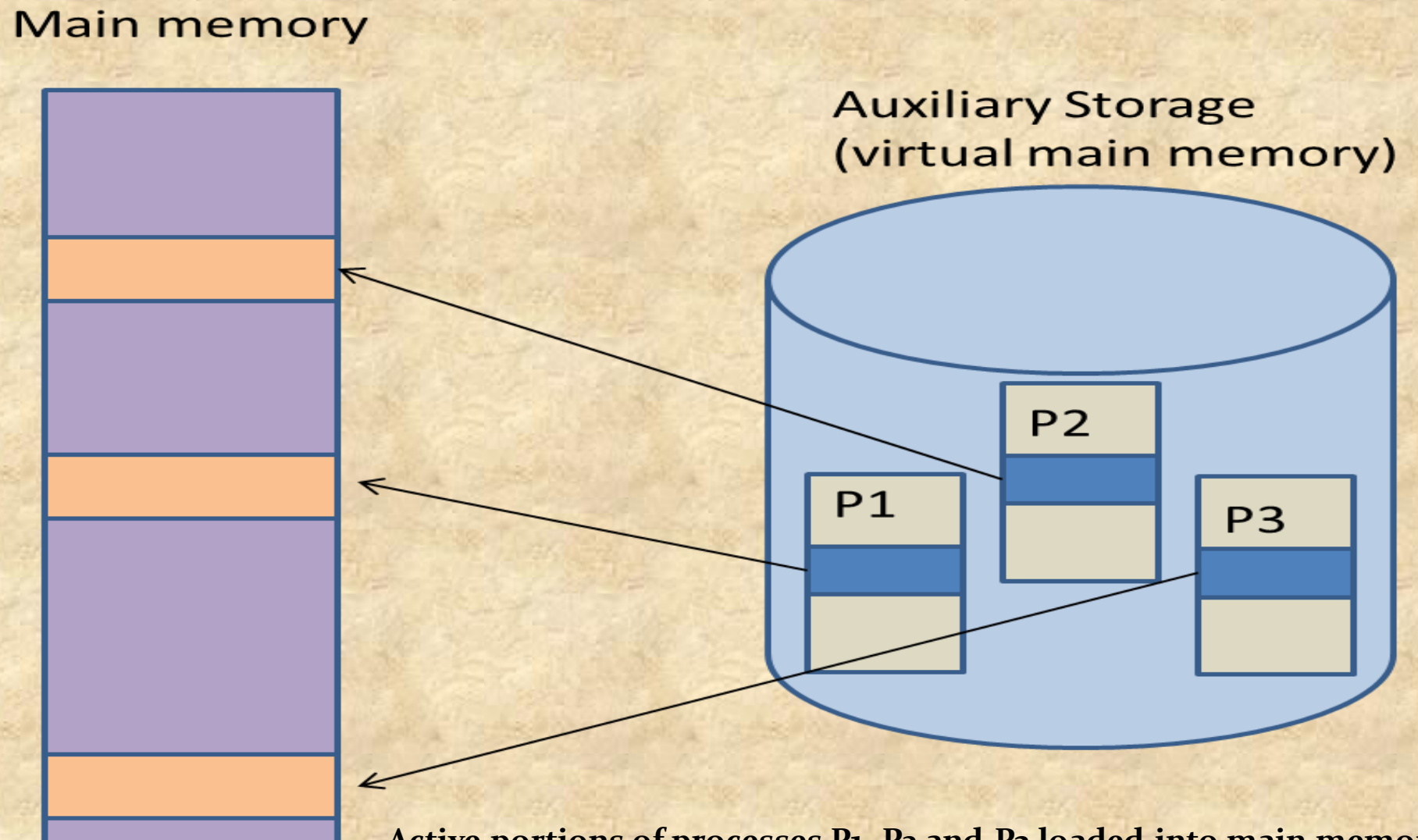
If we fault when we try to store in C (because C is in a page not currently in memory), we will have to get the desired page, bring it in, correct the page table, and restart the instruction.



Virtual Memory

Virtual Memory

- Main memory and auxiliary storage have been logically combined by the OS to give the illusion that a main memory of extremely large size is available with the system.



Active portions of processes P1, P2 and P3 loaded into main memory


START

What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.

Page Replacement

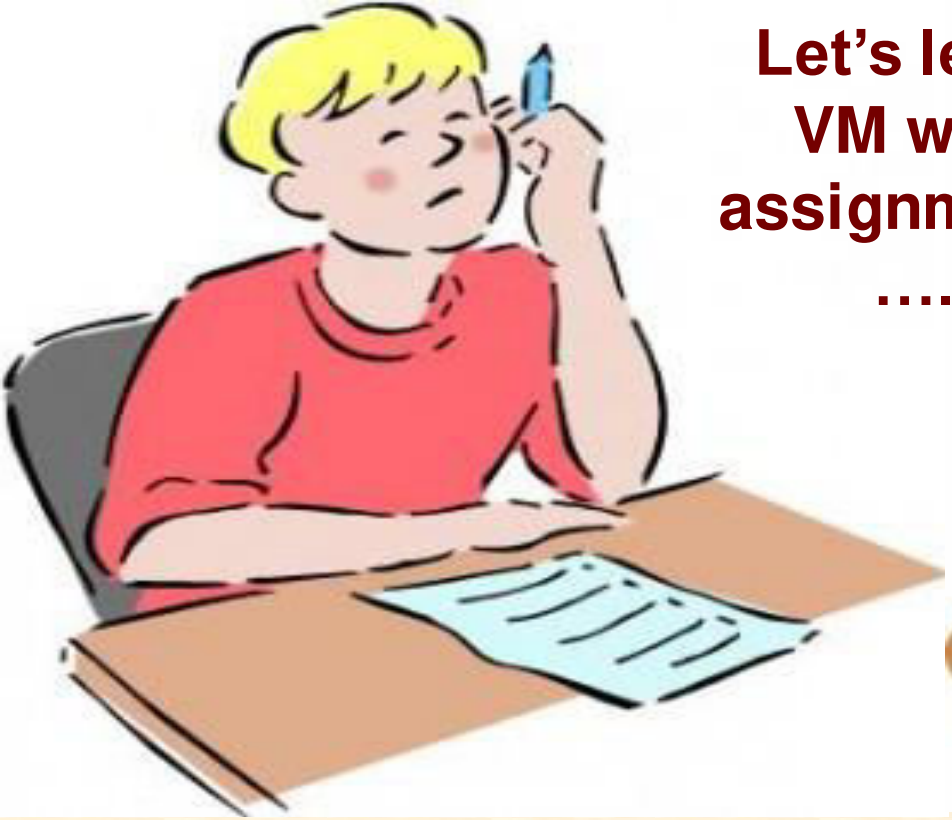
- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
- Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Active Page Replacement Policies

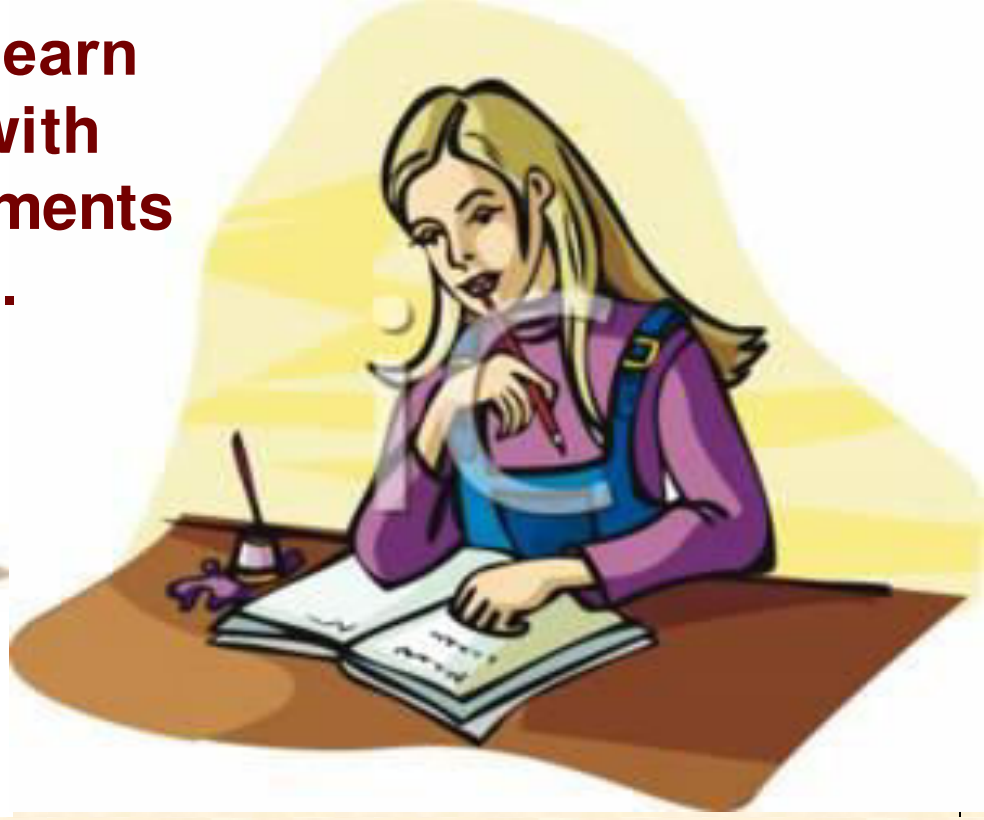
- **FIFO**: Utilizing queue data structure, replaces the first VM page that got accepted/included in MM for program execution.
- **Optimal Page Replacement Policy**: According to Belady (1966), at a page fault, replace the page whose next reference is farthest in the page reference string. This policy's drawback in practical scenario :- Infeasible because VM handler does not have knowledge of the future behavior of a process.

Active Page Replacement Policies

- **LRU Page Replacement**: Uses the principle of locality of reference (once a page is referred, it may be referred in near future). At every page fault, the least recently used page is replaced by a new page.
- The page table entry (**PT**) of a page records the time when the page was last referenced. This piece of information is initialized when the VM page is loaded in VM system and is modified every time the same page is referenced.



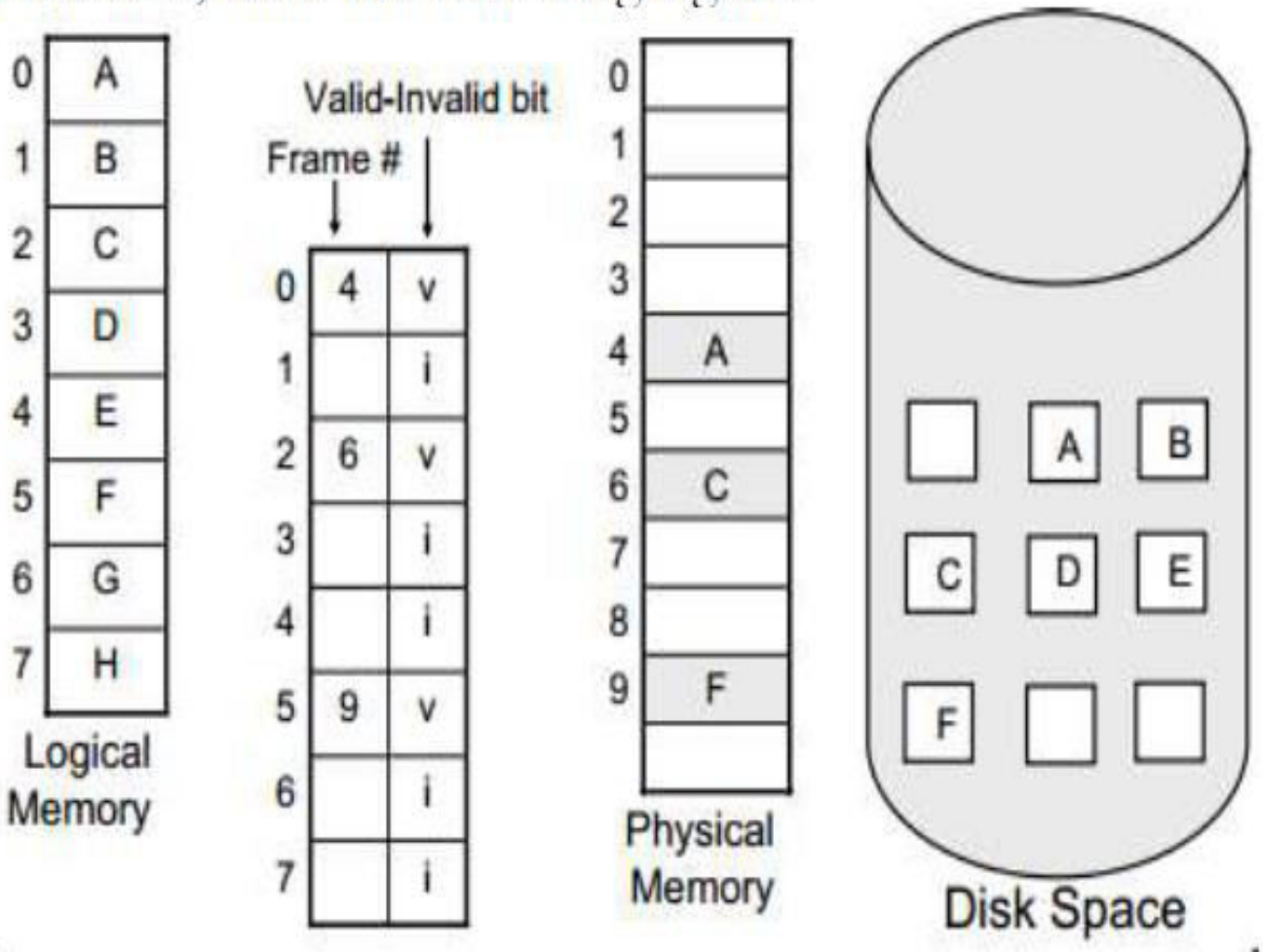
Let's learn VM with assignments ….



Problems on Demand Paging, Page replacement Algorithms (FCFS, LRU), Allocation of frames, thrashing……

Q How can the system distinguish between the pages that are in main memory from the pages that are on the disk?

The system uses valid-invalid bit is used. This bit is set to "valid" when the page in memory, while it set to "invalid" when the page either not valid or is the page is valid but is on the disk, as in the following figure.



Q: When will the page faults occur? What is the procedure for handling the page fault?

The page fault will occur when the process tries to access a page which was not into main memory.

The procedure for handling the page fault is:

1) Check an internal table for this process to determine whether the reference was a valid or an invalid memory access.
2) If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
3) Find a free frame.
4) Schedule a disk operation to read the desired page into the newly allocated frame.
5) When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6) Restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

Q : How can measure the performance of demand paging?

To measure the demand paging, the **effective access time** for a demand –paged memory is calculated by:

Effective access time = (1 – p) x ma + p x page fault time

Where **p**: The probability of page fault, $0 < p < 1$;

ma: Memory access time , ranges from 10 to 200 nanosecond.

Q: What are the operations of page replacement algorithm?

Every page replacement algorithm is operated by the following three operations:

1) Search: To search required pages from main memory.

2) Delete: To delete the evict page from main memory.

3) Insert: To insert the page into main memory.

Delete: Swap – out equivalent to evict.
Insert : Swap - in

Q: What are the principles of the following replacement algorithms?

a) FIFO.

b) LRU.

---

a) A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

b) In LRU, we can replace the page that *has not been used* for the longest period of time.

Q : Consider the following page reference using <u>three</u> frames that are initially empty. Find the page faults using FIFO algorithm, where the page reference sequence: 7,0,1, 2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1?

3 frames vs 20 pages

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
	0	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
		1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

The page fault = 15. Page fault

Belady's Anomaly

Bélády's anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.

Belady's Anomaly in FIFO –

Assuming a system that has no pages loaded in the memory and uses the FIFO Page replacement algorithm. Consider the following reference string:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Case-1: If the system has 3 frames, the given reference string on using FIFO page replacement algorithm yields a total of 9 page faults. The diagram below illustrates the pattern of the page faults occurring in the example.

1	1	1	2	3	4	1	1	1	2	5	5
	2	2	3	4	1	2	2	2	5	3	3
		3	4	1	2	5	5	5	3	4	4
PF	PF	PF	PF	PF	PF	PF	X	X	PF	PF	X

Belady's Anomaly

Case-2: If the system has 4 frames, the given reference string on using FIFO page replacement algorithm yields a total of 10 page faults. The diagram below illustrates the pattern of the page faults occurring in the example.

1	1	1	1	1	1	2	3	4	5	1	2
	2	2	2	2	2	3	4	5	1	2	3
		3	3	3	3	4	5	1	2	3	4
			4	4	4	5	1	2	3	4	5
PF	PF	PF	PF	X	X	PF	PF	PF	PF	PF	PF

It can be seen from the above example that on increasing the number of frames while using the FIFO page replacement algorithm, the number of **page faults increased** from 9 to 10.

Q: Consider the following page reference using three frames that are initially empty. Find the page faults using LRU algorithm, where the page reference sequence: 7,0,1, 2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1?

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

Page fault= 12

Q: Consider the following page reference using **four** frames that are initially empty. Find the page faults using **LRU algorithm**, where the page reference sequence: 5,2,5,1,4,5,2,0,4,2,3,1,2,1,0,0,2,4,5,1?

5	2	5	1	4	5	2	0	4	2	3	1	2	1	0	0	2	4	5	1
5	5		5	5			5			3	3			3			4	4	4
	2		2	2			2			2	2			2			2	2	2
			1	1			0			0	1			1			1	5	5
				4			4			4	4			0			0	0	1
*	*		*	*			*			*	*			*			*	*	*

Page fault= 11

Q: Consider the following page reference using **four** frames that are initially empty. Find the page faults using **FIFO** algorithm, where the page reference sequence: 5,2,1,5,1,0,3,1,2,1,4,0,5,4, 2,3,3, 4,2,1?

5	2	1	5	1	0	3	1	2	1	4	0	5	4	2	3	3	4	2	1
5	5	5			5	2				1		0		3					4
	2	2			2	1				0		3		4					5
		1			1	0				3		4		5					2
					0	3				4		5		2					1
*	*	*			*	*				*		*		*					*

Page fault = 9

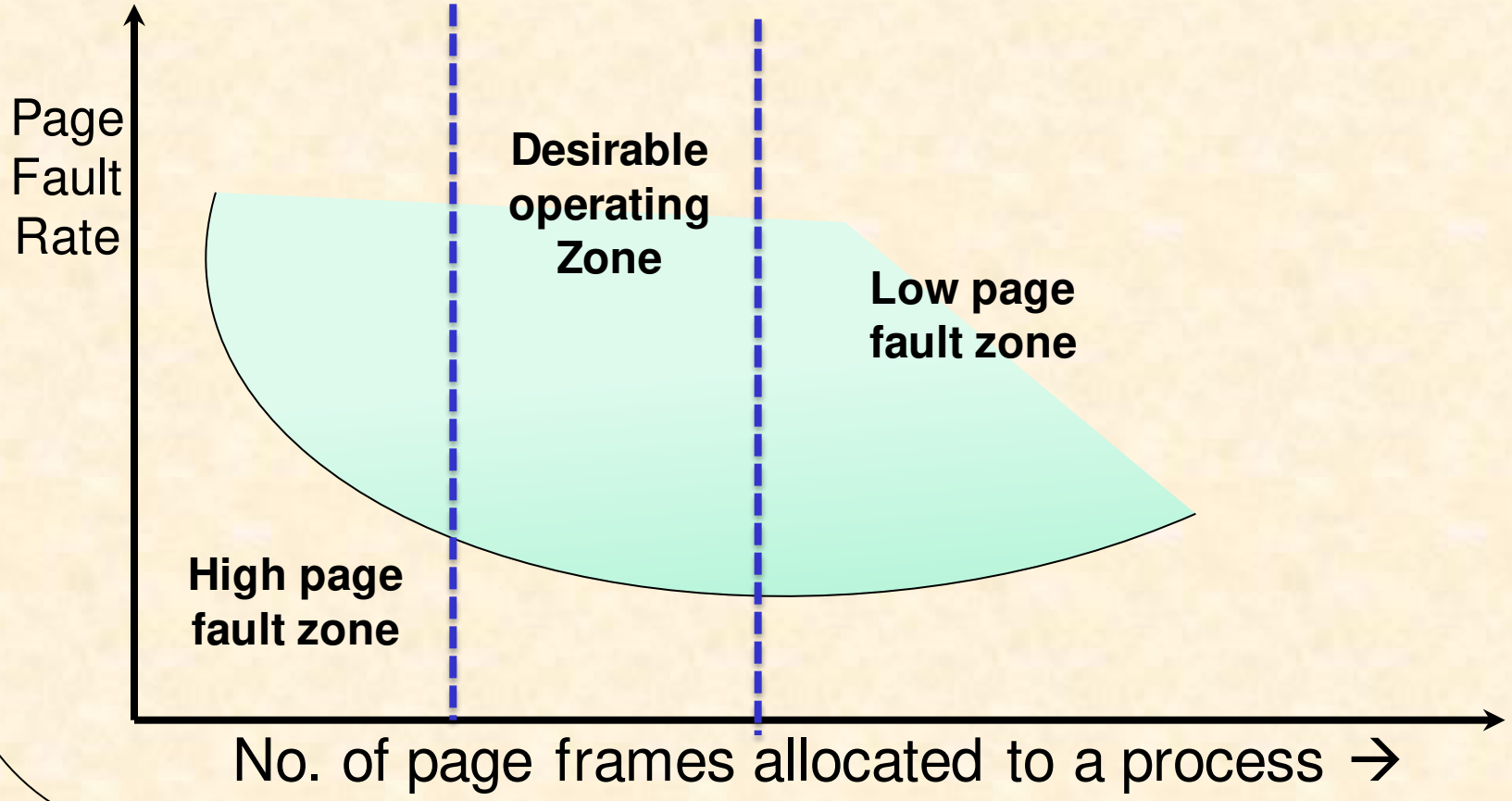
Q: Consider we have the following reference string: 5, 0, 4, 4, 0, 3, 0, 4, 1, 0, 2, 0, 5, 3, 0, 1.

Find the page fault of virtual memory using LRU algorithm, where we used 4 frames?

5	0	4	4	0	3	0	4	1	0	2	0	5	3	0	1
5	5	5	5	5	5	5	5	1	1	1	1	1	3	3	3
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		4	4	4	4	4	4	4	4	4	4	5	5	5	5
					3	3	3	3	3	2	2	2	2	2	2
*	*	*			*			*		*		*	*		*

Page fault= 8

Desirable Page Fault Characteristics



Page Fault & Thrashing

- Page fault rate decreases monotonically with the increase of number of page frames, page fault $\propto \frac{1}{No.of\ page\ frames}$

- **Let's consider a process (Pi) operating to the left of the desirable operating zone, i.e., in the region of high page fault rates. If all processes in the system operate in high page fault rate zones, the CPU would be mostly busy performing page traffic & process switching, resulting in poor average response time or throughput → resulting in thrashing.**

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization.
  - operating system thinks that it needs to increase the degree of multiprogramming.
  - another process added to the system.
- Thrashing ≡ a process is busy swapping pages in and out.
- Definition: situation in which a process is spending more time paging than executing

What is 'thrashing'? What is the effect of it on page fault frequency? What is the problem of fragmentation and how can it be solved?

A process that is spending more time paging than executing is said to be thrashing. In other words it means, that the process doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.

- ***Page Fault Frequency***: another approach to preventing thrashing.
- Per-process replacement; at any given time, each process is allocated a fixed number of physical page frames.
- Monitor the rate at which page faults are occurring for each process.
- If the rate gets too high for a process, assume that its memory is overcommitted; increase the size of its ***memory pool***.
- If the rate gets too low for a process, assume that its ***memory pool*** can be reduced in size.
- If the sum of all ***memory pools*** don't fit in memory, deactivate some processes.

In demand paging, pages are brought into the physical memory only when the pages are needed. All the pages of a process need not be kept in the main memory at the same time. Only the pages that are used currently need to be kept in the main memory. Suppose a page is brought into the main memory and if there is no free frame in the main memory, a victim page is chosen from the main memory, the victim page is moved out to the backing store and the required page is brought into the main memory. Choosing a victim page for replacement is done by the page replacement algorithms.
The other option is to maintain a ***pool of free frames always*** and ***move in the required page into a frame***. The victim page can be moved out later and that frame can be added to the pool of free frames.

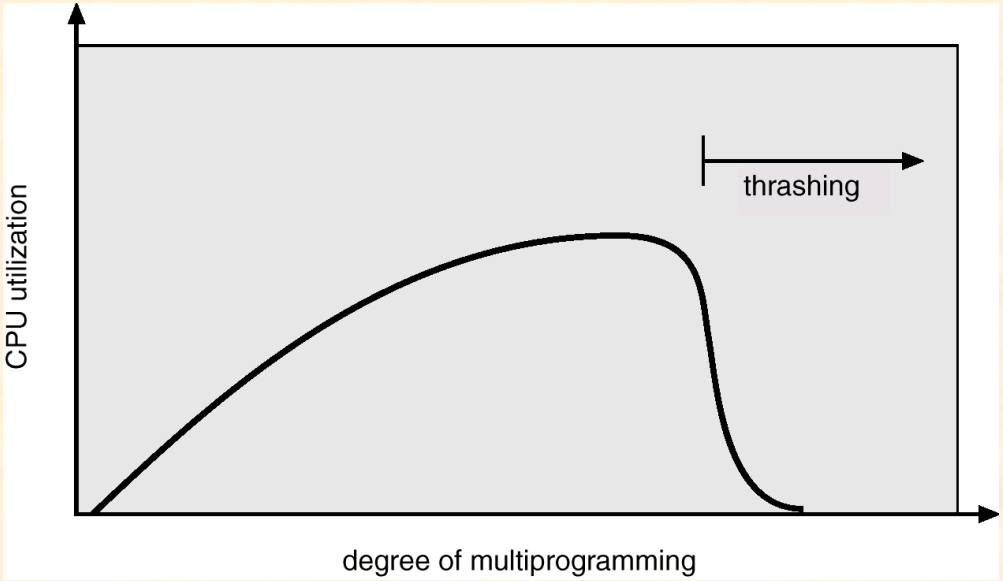
Fragmentation: The most **severe** problem caused by fragmentation is causing a process or system to fail, due to premature resource exhaustion: ***if a contiguous block must be stored and cannot be stored, failure occurs***. Fragmentation causes this to occur even if there is enough of the resource, but not a contiguous amount.

Solution to **external fragmentation** :

1) **Compaction** : shuffling the **fragmented** memory into one contiguous location.

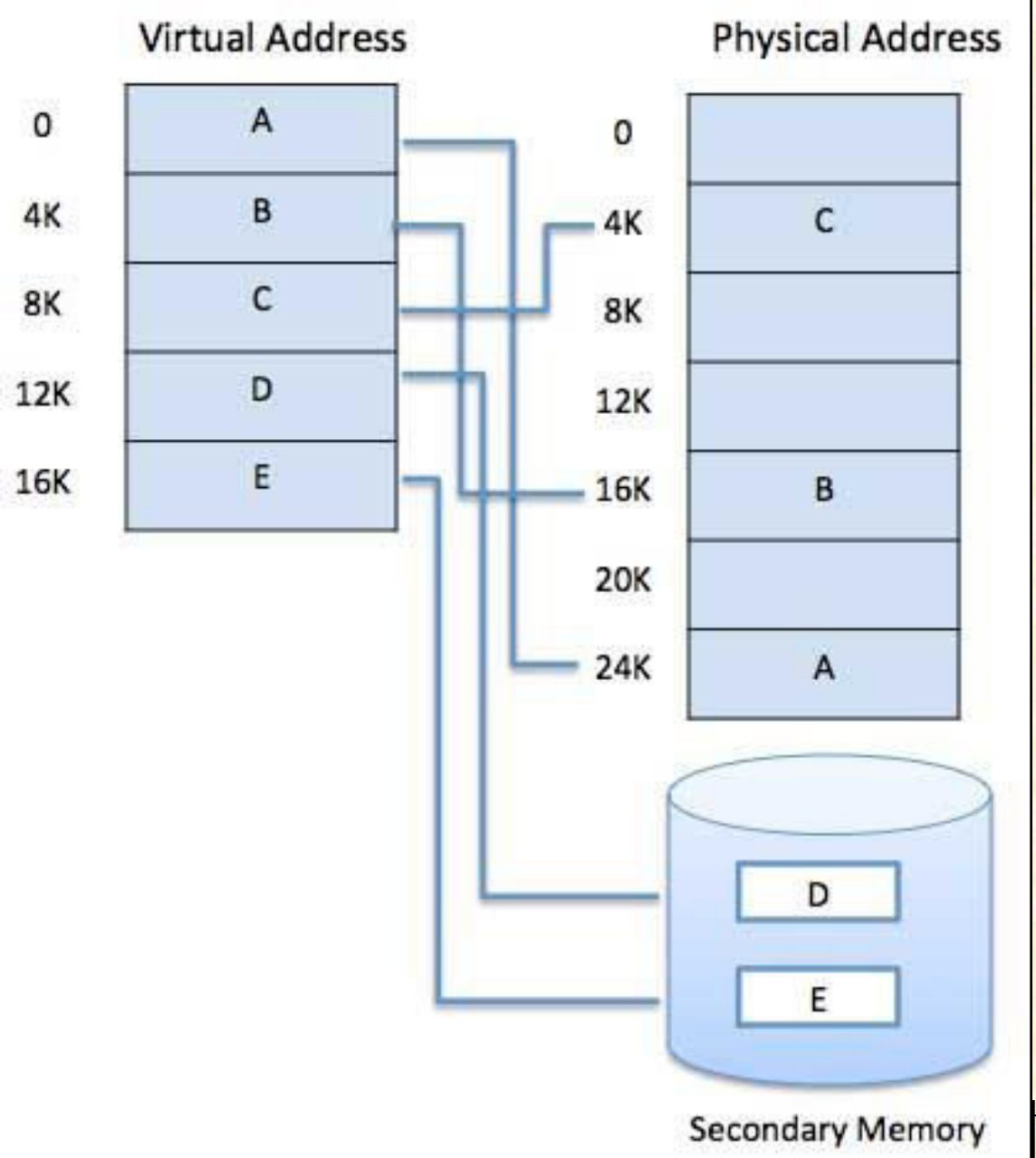
2) Virtual memory addressing by using paging and segmentation. **External fragmentation can** be reduced by compaction or shuffle memory contents to place all free memory together in one large block.

Thrashing Diagram



- Why does paging work?
  Locality model
  - Process migrates from one locality to another.
  - Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size

Demand Segmentation



Operating system also uses **demand segmentation**, which is similar to **demand** paging.

Operating system to uses **demand segmentation** where there is ***insufficient hardware*** available to implement ***'Demand Paging***'. The segment table has a valid bit to specify if the segment is already in physical **memory** or not.

Demand Segmentation

- Operating system also uses **demand segmentation**, which is similar to demand paging. Operating system to uses demand segmentation where there is insufficient hardware available to implement 'Demand Paging'.
- The segment table has a valid bit to specify if the segment is already in physical memory or not. If a segment is not in physical memory then segment fault results, which traps to the operating system and brings the needed segment into physical memory, much like a page fault.
- Demand segmentation allows for pages that are often referenced with each other to be brought into memory together, this decreases the number of page faults.
- Another space server would be to keep some of a segment's page tables on disk and swap them into memory when needed.