CS & IT ENGINEERING

Operating System

Virtual Memory (Part - 01)

Revision

Lecture No.- 11

By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

Topic Paging, Segmentation

Topics to be Covered

Topic	Virtual memory
Topic	Demand Paging
Topic	Basic concept
Topic	Instruction Restart
Topic	Valid-Invalid Bit

Topics to be Covered

Topic Aspects of Demand Paging

Topic Allocation of Frames

Topic Thrashing

Topic : Hashed Page Table

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashing to the same location
- Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted
- Variation for 64-bit addresses is **clustered page tables**
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for **sparse** address spaces (where memory references are non-contiguous and scattered)

Topic : Hashed Page Table

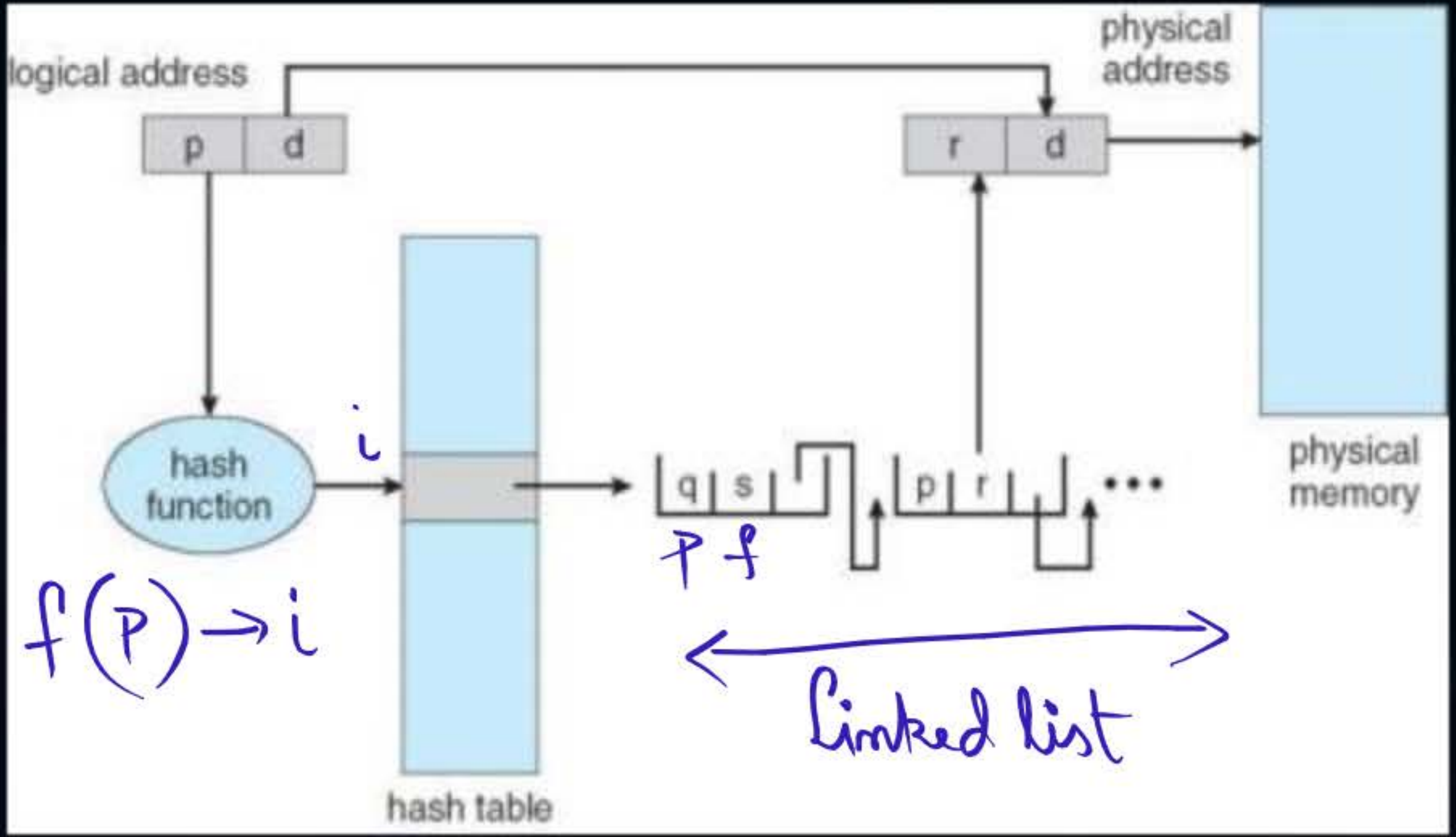
$[x \% 10] \rightarrow \langle 0 \ldots 9 \rangle \rightarrow$ Collission

Probing   chaining

(i) Traditional Paging : $O(1)$

(ii) Hashed Paging : $O(\ell)$

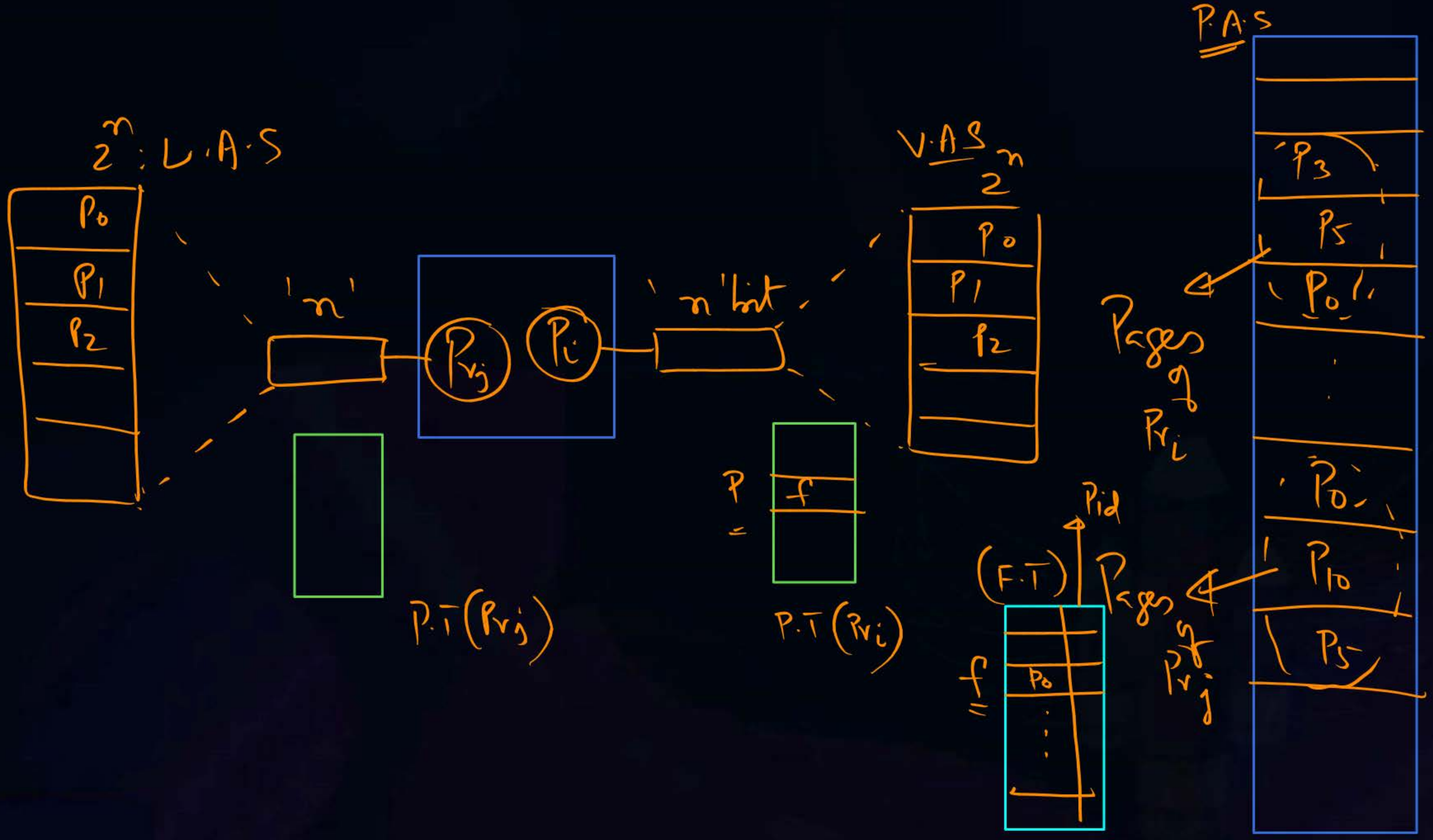
ℓ : Avg. Size of L.L



⟨ If the referred Page entry is NOT found in the chain – Page fault ⟩

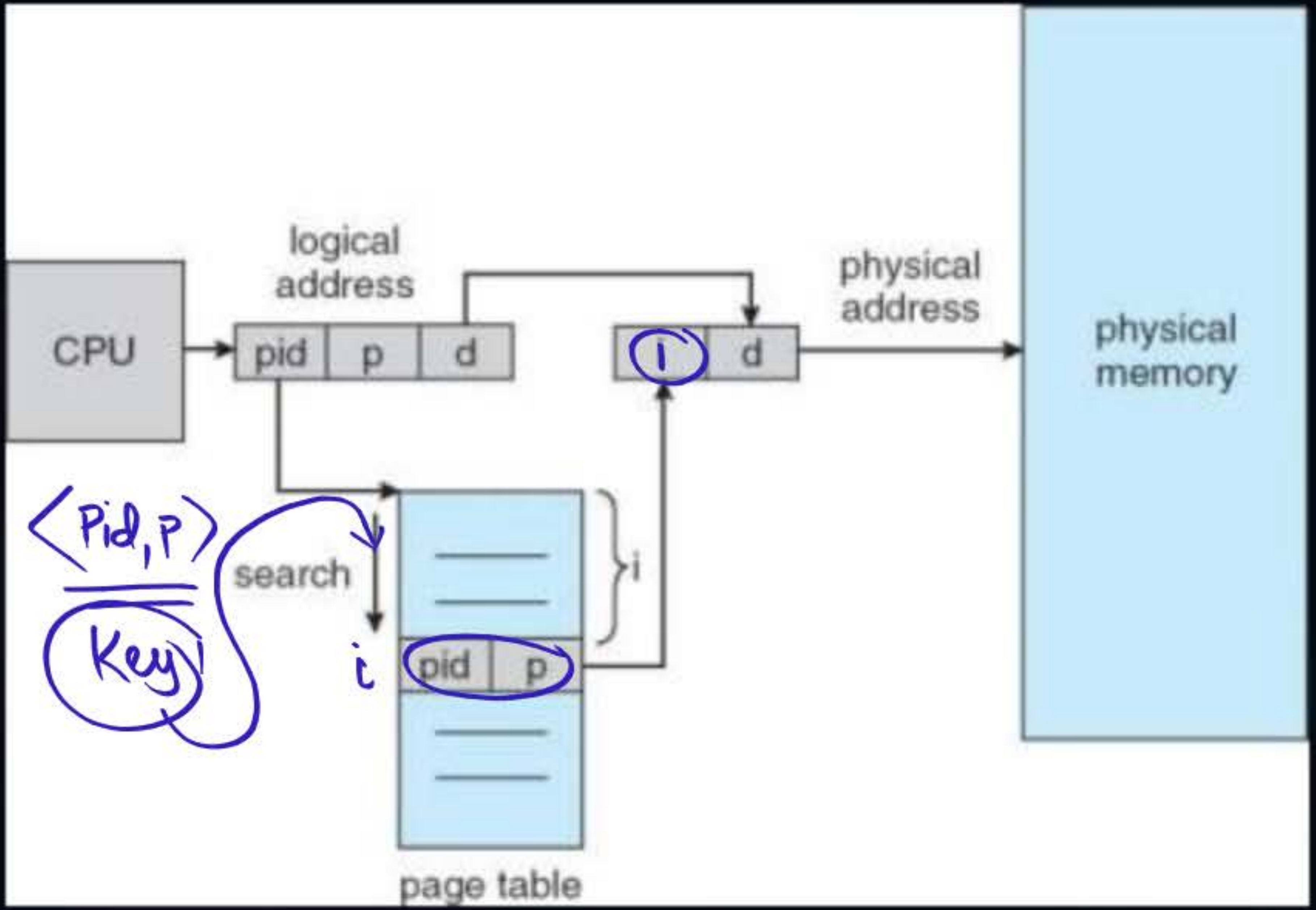
Topic : Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address



2^n : L.A.S
P_0
P_1
P_2
'n'
P_{r_j}
P_i
'n' bit
V.A.S
2^n
P_0
P_1
P_2
P.A.S
P_3
P_5
P_0
Pages of P_{r_i}
P_0
P_{10}
P_5
Pages of P_{r_j}
p
f
P.T (P_{r_j})
P.T (P_{r_i})
Pid
(F.T)
f
P_0

Topic : Inverted Page Table Architecture



→ All processes will use a Single P.T

→ Search Time

$O(n)$

n = Size of Inv. P.T

→ System supports $\overset{N}{\underline{2K}}$ Pages with a 9 bit frame No.
Pid is 2B; If the Conv. P.T entry Contains additional $\underline{7 \text{ bits}}$
Compute the approx. Sizes of Conv. P.T & Inv. P.T in Bytes

$e = 9 + 7 = \underline{16}$

Conv. P.T

$P.T.S = \underline{N * e}$

$= 2K * 2B$

$= \boxed{4KB}$ ✓

Inv. P.T

$e' \sim \underline{P + Pid}$

$11 + 16 = 27 \text{ bits}$

$\hookrightarrow \underline{32 \text{ bits}}$

4B

$P.T.S = M * e'$

$= 512 * 4B$

$= \boxed{2KB}$ ✓

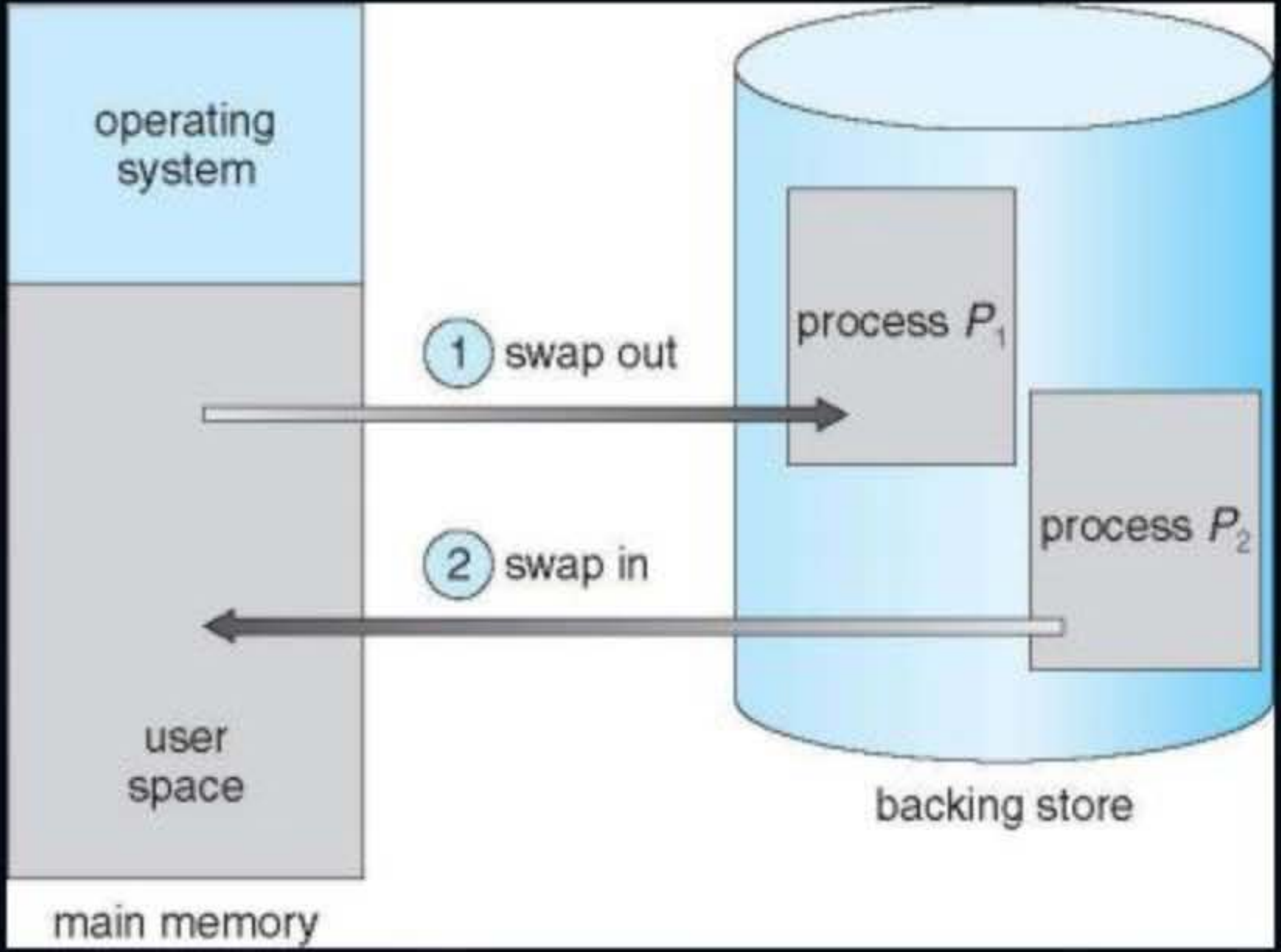
Topic : Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought **back** into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Topic : Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

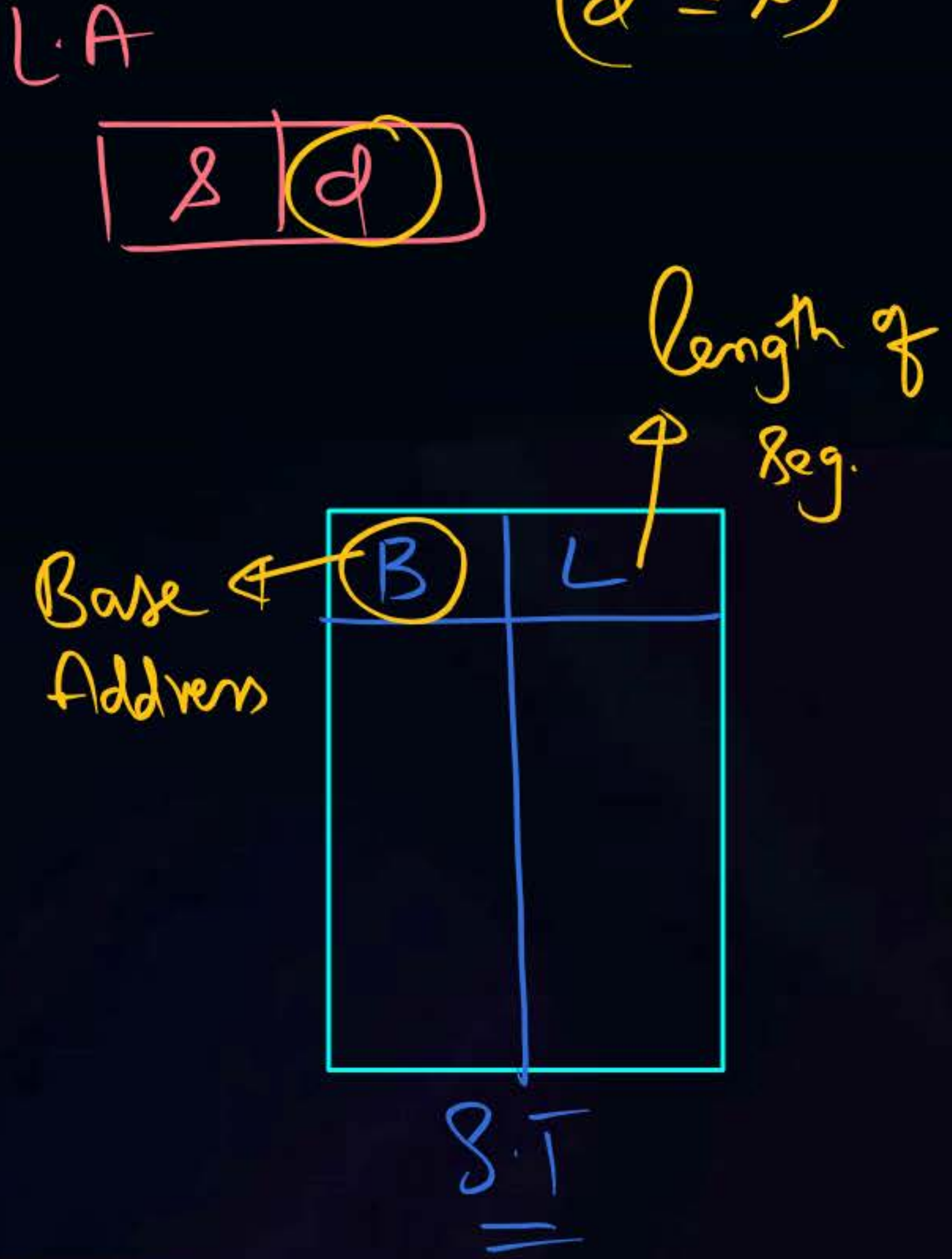
Topic : Schematic View of Swapping

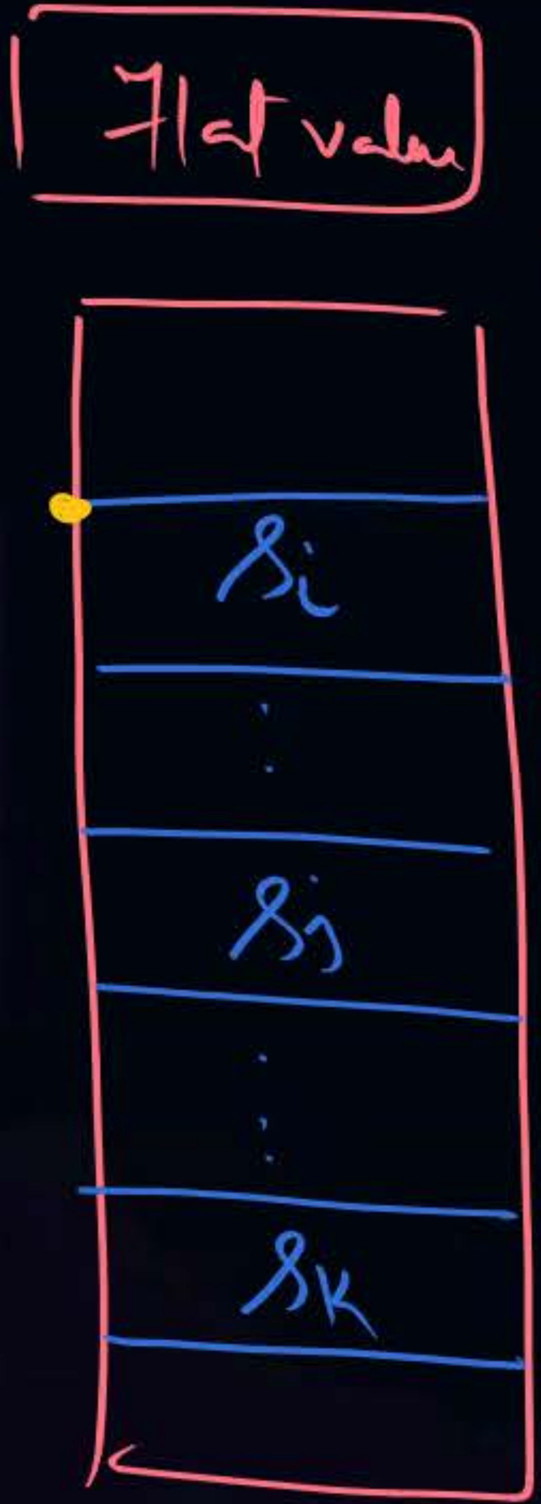


Swap-time :

Segmentation :

$(d \leq l)$





P.A.S (variable partitions)

Program
$\hookrightarrow$ Segments

(may be of different sizes)

< Logical Entity >

(Function, Procedure, class (object), data structure, Heap, Stack)

Paging vs Segmentation

- → Internal Frag. (last Page)
- → Never E. Frag

- → Never Int. Frag.
- → May've Ext. Frag (P.A.S) → V. Part.
  - ↓ Paging ⟨Seg-Paging⟩

Consider a simple segmentation system that has the following segment table:

	Starting Address	Length (bytes)
0	660	248
1	1,752	422
2	222	198
3	996	604

For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs:

a. 0, 198 $\longrightarrow$ 660 + 198
b. 2, 156 $\longrightarrow$ 222 + 156
c. 1, 530 $\longrightarrow$ Trap
d. 3, 444 $\longrightarrow$ 996 + 444
e. 0, 222 $\longrightarrow$ 660 + 222

Topic : Virtual Memory Background

- Code needs to be in memory to execute, but entire program rarely used
  - Error code, unusual routines, large data structures
- Entire program code not needed at same time
- Consider ability to execute partially-loaded program
  - Program no longer constrained by limits of physical memory
  - Each program takes less memory while running -> more programs run at the same time
    - Increased CPU utilization and throughput with no increase in response time or turnaround time
  - Less I/O needed to load or swap programs into memory -> each user program runs faster

Topic : Virtual memory

- 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
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes

V.M $\longrightarrow$ Managing Larger (Bigger) Programs in Small Mem (Less Mem)

$\downarrow$

⟨Demand Paging⟩ (DP) : Loading the Pages on Demand @ R.Time, from Sec Storage (Disk) to Main Memory (P.A.S) (DMA)

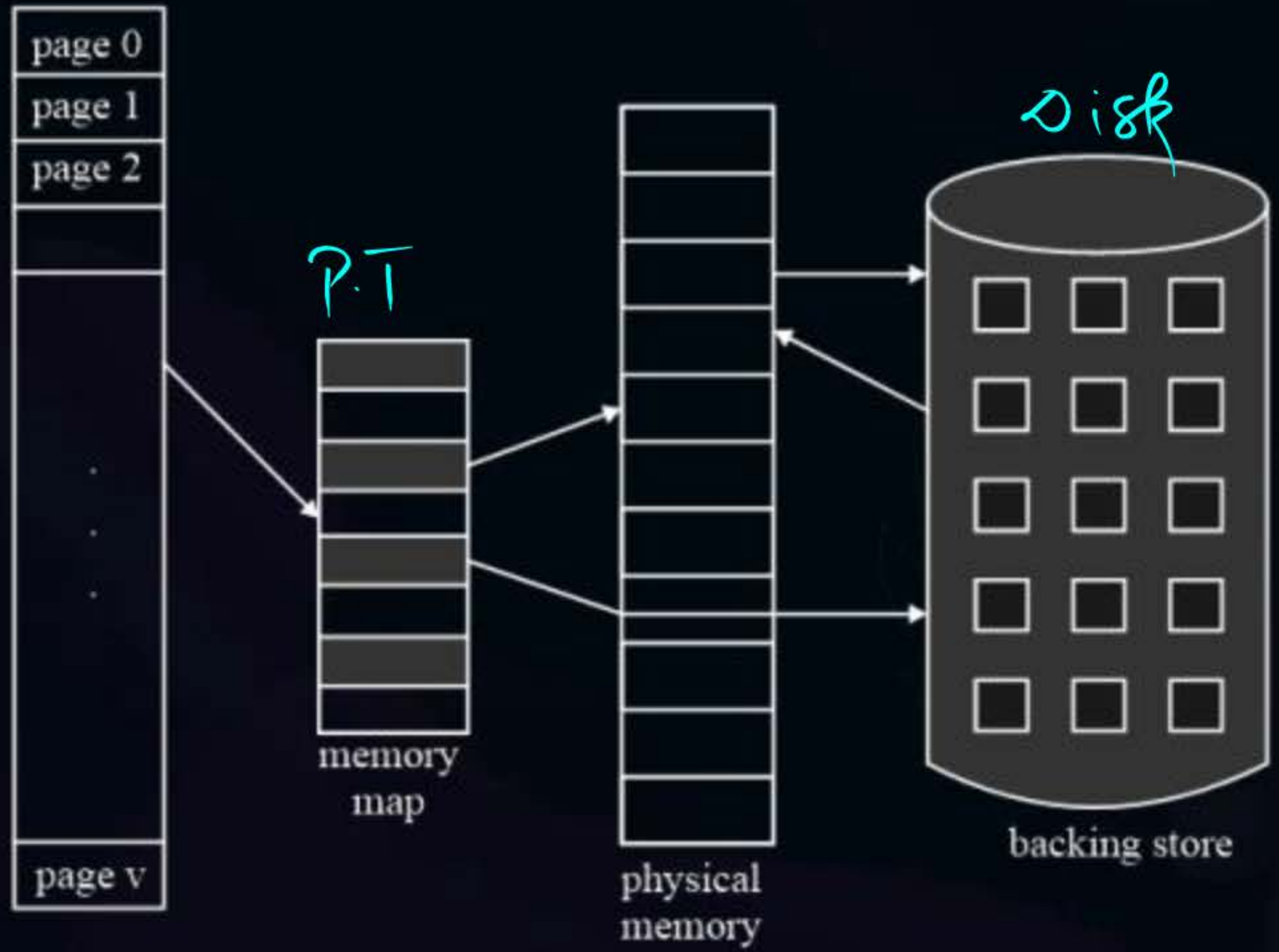
✓ Pure D.P ⟨Empty frames⟩

Prefetch D.P ⟨Frames Loaded with Pages⟩

Topic : Virtual memory (Cont.)

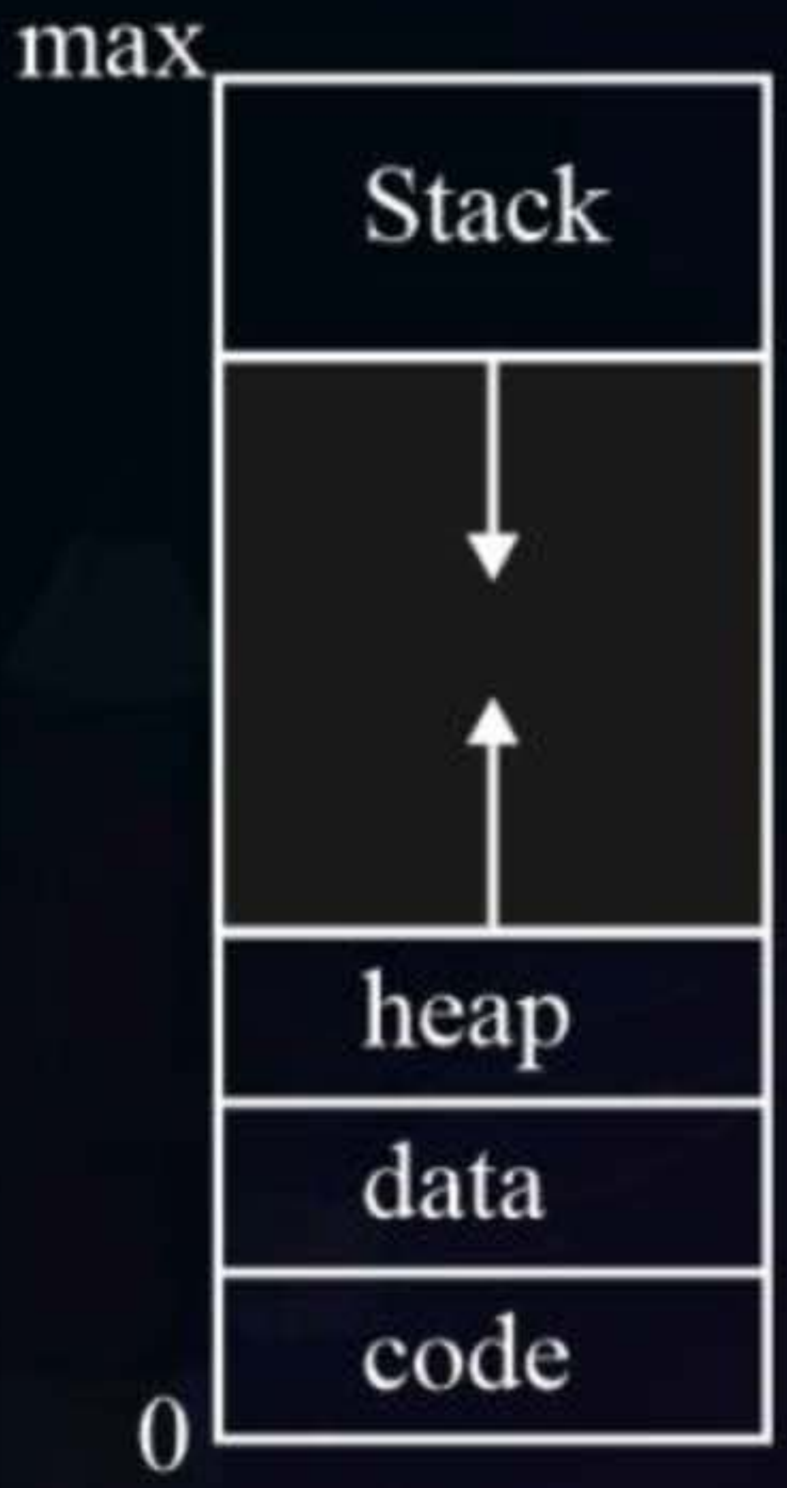
- Virtual address space – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
- Virtual memory can be implemented via:
  - ✓ Demand paging : In practice
  - ✓ Demand segmentation : Theory

Topic : Virtual Memory That is Larger Than Physical Memory

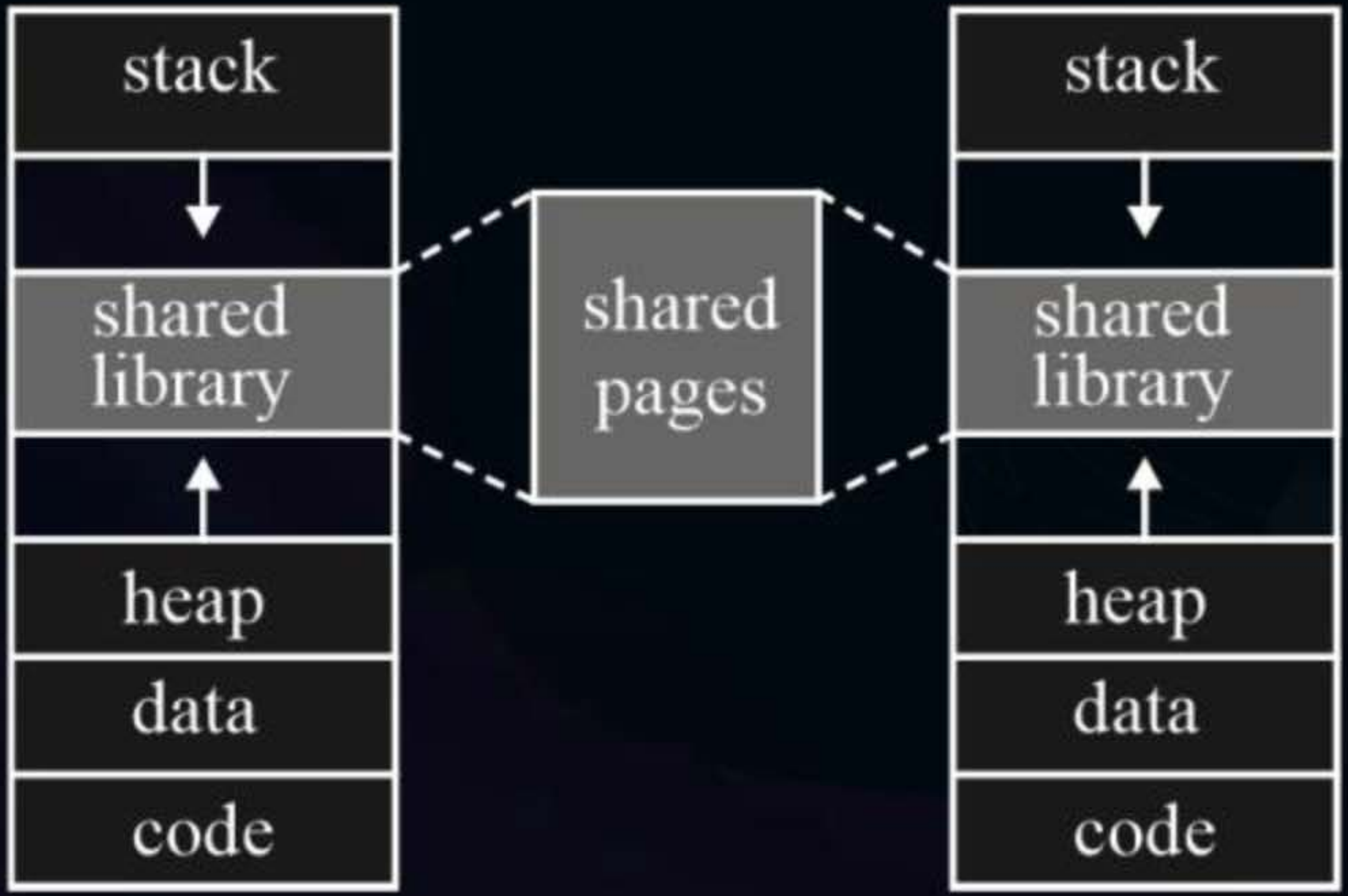


Topic : Virtual-address Space

- Usually design logical address space for stack to start at Max logical address and grow “down” while heap grows “up”
  - Maximizes address space use
  - Unused address space between the two is hole
    - No physical memory needed until heap or stack grows to a given new page
- Enables sparse address spaces with holes left for growth, dynamically linked libraries, etc.
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during fork(), speeding process creation



Topic : Shared Library Using Virtual Memory

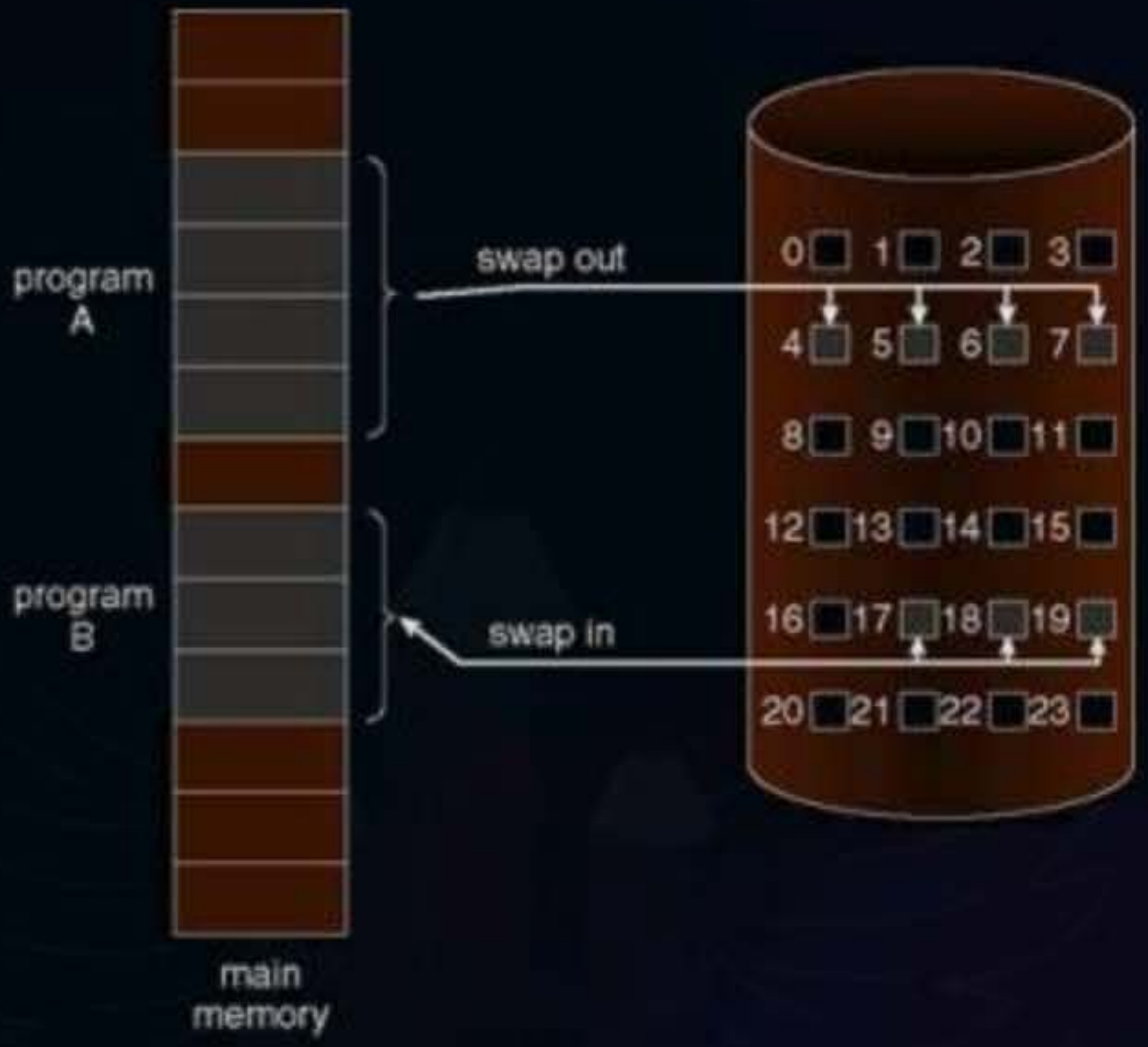


Topic : Demand Paging

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Similar to paging system with swapping (diagram on right)
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- Lazy swapper – never swaps a page into memory unless page will be needed
- Swapper that deals with pages is a pager

Topic : Demand Paging

- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Similar to paging system with swapping (diagram on right)

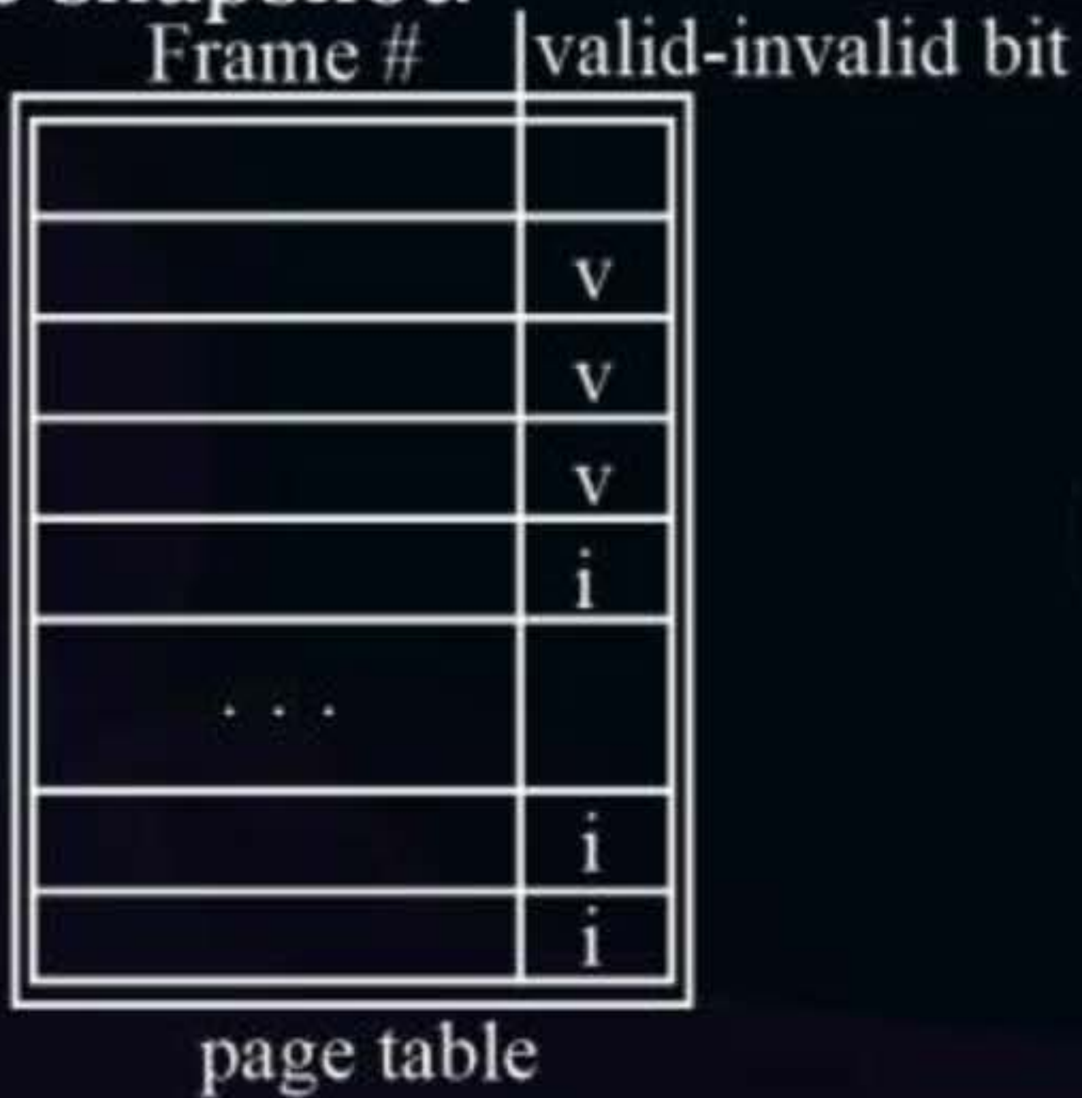


Topic : Basic Concepts

- With swapping, pager guesses which pages will be used before swapping out again
- Instead, pager brings in only those pages into memory
- How to determine that set of pages?
  - Need new MMU functionality to implement demand paging
- If pages needed are already memory resident
- No difference from non demand-paging
- If page needed and not memory resident
  - Need to detect and load the page into memory from storage
    1. Without changing program behavior
    2. Without programmer needing to change code

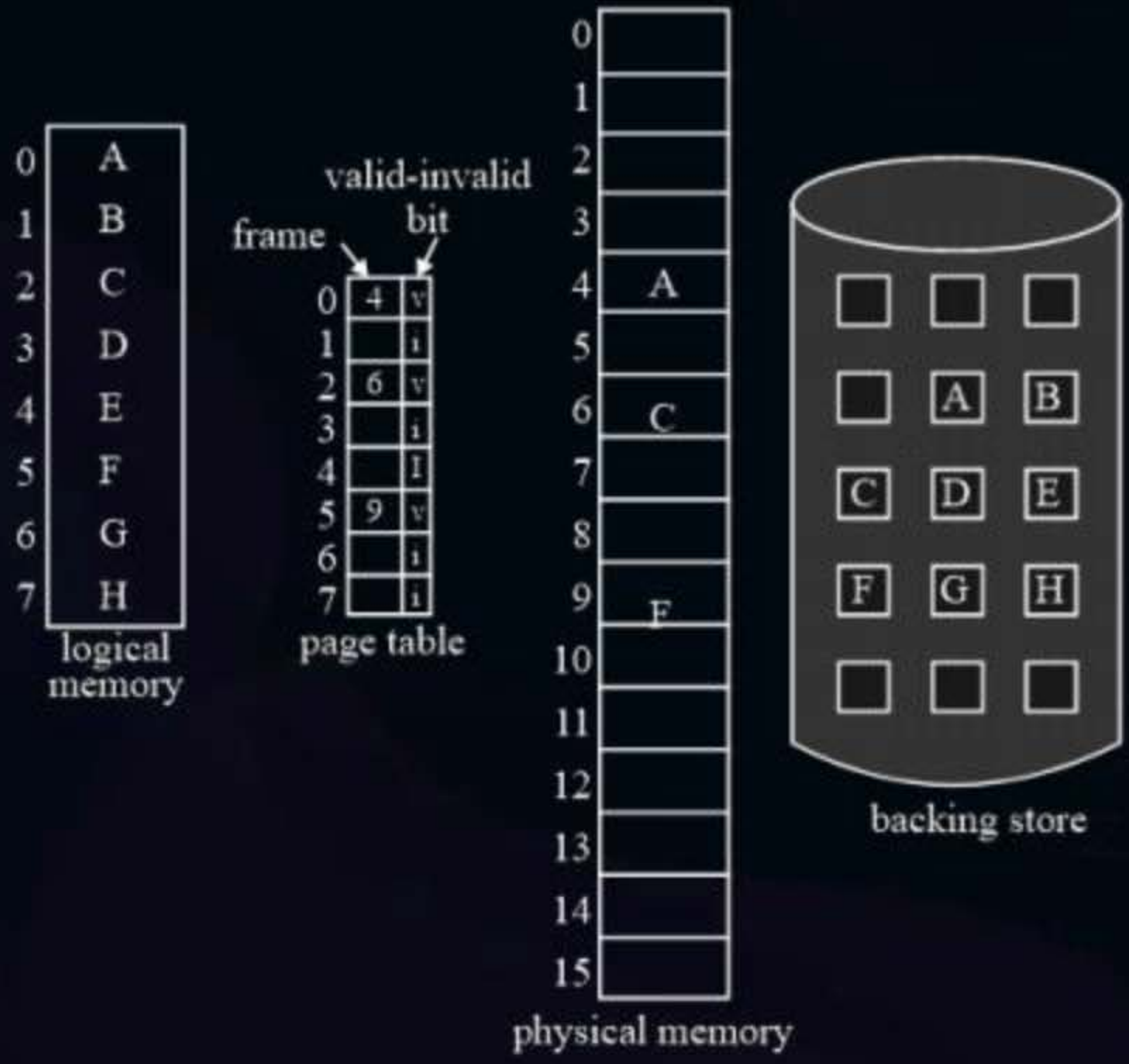
Topic : Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (v ⇒ in-memory – memory resident, i ⇒ not-in-memory)
- Initially valid–invalid bit is set to i on all entries
- Example of a page table snapshot:



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

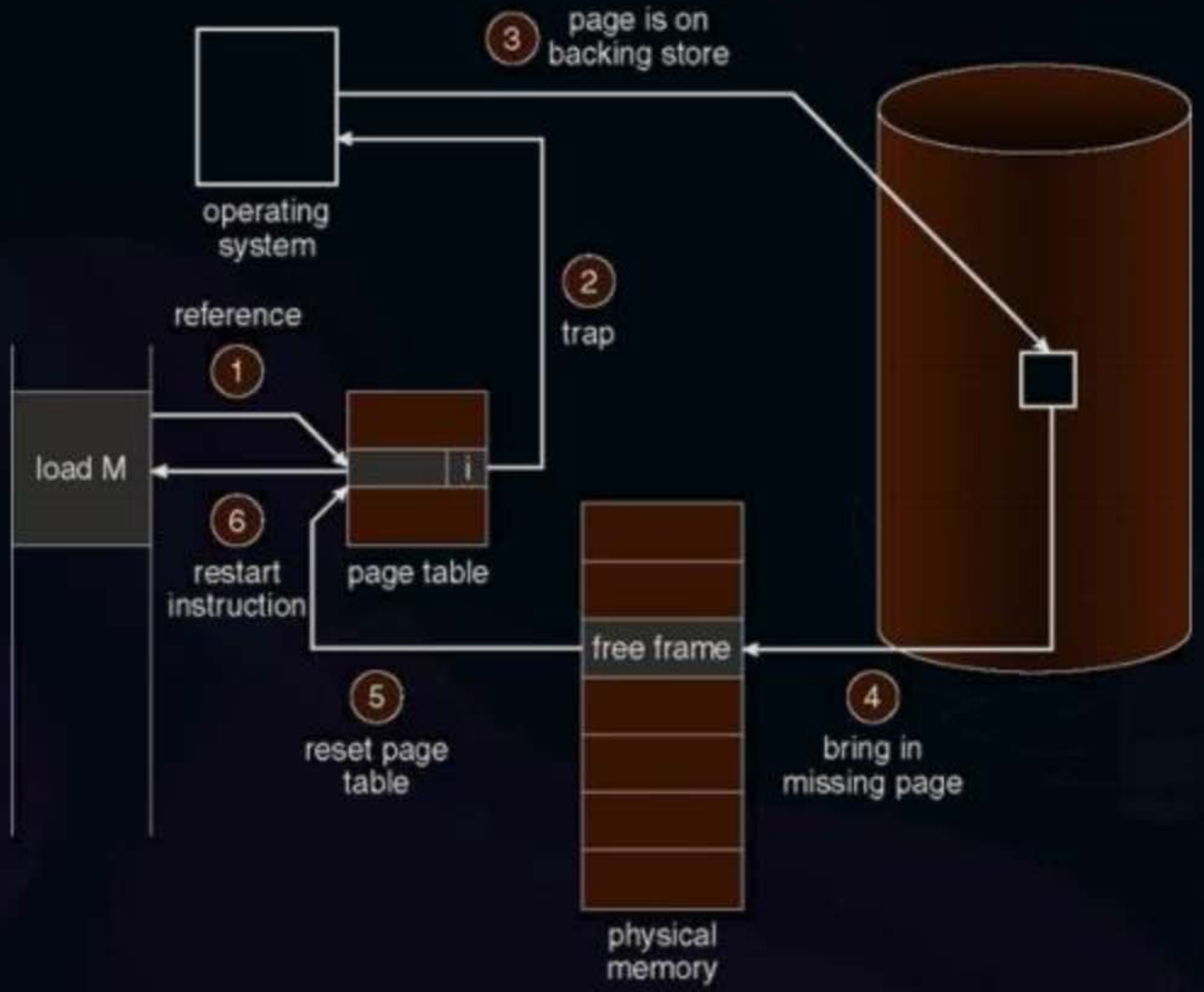
Topic : Page Table When Some Pages Are Not in Main Memory



Topic : Steps in Handling Page Fault

1. If there is a reference to a page, first reference to that page will trap to operating system
    - Page fault
2. Operating system looks at another table to decide:
    - Invalid reference $\Rightarrow$ abort
    - Just not in memory
3. Find free frame
4. Swap page into frame via scheduled disk operation
5. Reset tables to indicate page now in memory Set validation bit = v
6. ✓ Restart the instruction that caused the page fault

Topic : Steps in Handling a Page Fault (Cont.)

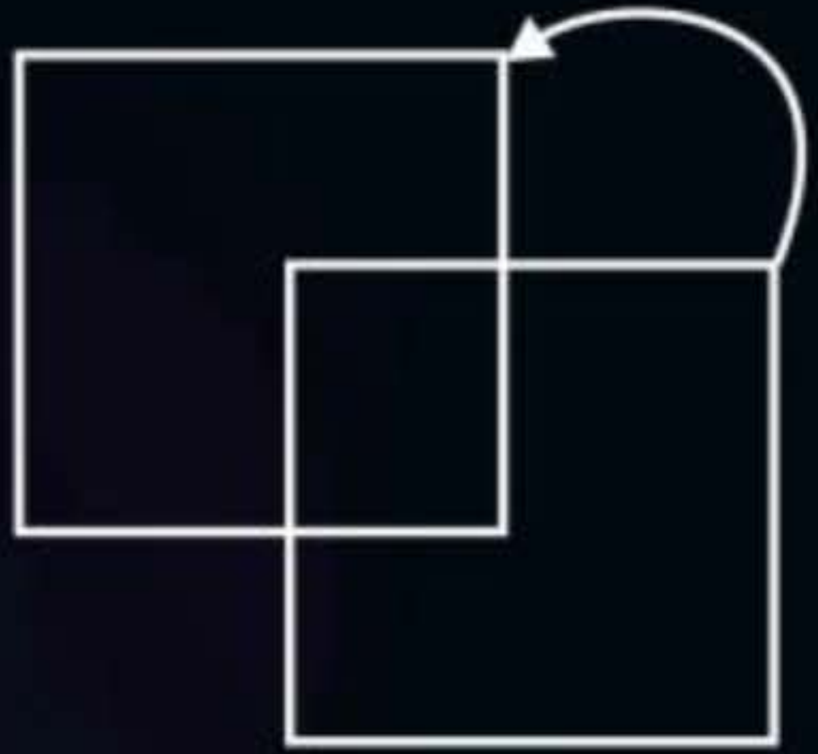


Topic : Aspects of Demand Paging

- Extreme case – start process with no pages in memory
  - OS sets instruction pointer to first instruction of process, non-memory-resident -> page fault
  - And for every other process pages on first access
  - Pure demand paging
- Actually, a given instruction could access multiple pages -> multiple page faults
  - Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory
  - Pain decreased because of locality of reference
- Hardware support needed for demand paging
  - Page table with valid / invalid bit
  - Secondary memory (swap device with swap space)
  - Instruction restart

Topic : Instruction Restart

- Consider an instruction that could access several different locations
  - Block move



  - Auto increment/decrement location
  - Restart the whole operation?
    - What if source and destination overlap?

Topic : Free-Frame List

- When a page fault occurs, the operating system must bring the desired page from secondary storage into main memory.
- Most operating systems maintain a free-frame list -- a pool of free frames for satisfying such requests.

$$\begin{tikzcd}
\text{head} \arrow[r] & \boxed{7} \arrow[r] & \boxed{97} \arrow[r] & \boxed{15} \arrow[r] & \boxed{126} & \cdots \arrow[r] & \boxed{75}
\end{tikzcd}$$

- Operating system typically allocate free frames using a technique known as zero-fill-on-demand -- the content of the frames zeroed-out before being allocated.
- When a system starts up, all available memory is placed on the free-frame list.

Topic : Stages in Demand Paging – Worse Case

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
   a) Wait in a queue for this device until the read request is serviced
   b) Wait for the device seek and/or latency time
   c) Begin the transfer of the page to a free frame

Topic : Stages in Demand Paging (Cont.)

6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

Topic : Performance of Demand Paging

- Three major activities
  - Service the interrupt – careful coding means just several hundred instructions needed
  - Read the page – lots of time
  - Restart the process – again just a small amount of time
- Page Fault Rate $0 \le p \le 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

EAT = (1 – p) x memory access
+ p (page fault overhead
+ swap page out
+ swap page in)

Galvin Text

$E.A.T = (1-p)\underline{\underline{m}} + P * S$

D.P

$\underline{\underline{TLB}}$

hit prob

Topic : Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p x 200 + p x 8,000,000

  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then

  EAT = 8.2 microseconds.

  This is a slowdown by a factor of 40!!
- If want performance degradation < 10 percent
  - 220 > 200 + 7,999,800 x p
  - 20 > 7,999,800 x p
  - p < .0000025
  - < one page fault in every 400,000 memory accesses

Topic : Demand Paging Optimizations

- Swap space I/O faster than file system I/O even if on the same device
  - Swap allocated in larger chunks, less management needed than file system
- Copy entire process image to swap space at process load time
  - Then page in and out of swap space
  - Used in older BSD Unix
- Demand page in from program binary on disk, but discard rather than paging out when freeing frame
  - Used in Solaris and current BSD
  - Still need to write to swap space
    1. Pages not associated with a file (like stack and heap) – anonymous memory
    2. Pages modified in memory but not yet written back to the file system
- Mobile systems
  - Typically don't support swapping
  - Instead, demand page from file system and reclaim read-only pages (such as code)

V.A.S

P.A.S

Disk A.S

In practice

1. $\boxed{\text{Size of V.A.S} \leq \text{Disk A.S}}$

2. $\boxed{\text{P.A.S} \leq \text{V.A.S} \leq \text{Disk A.S}}$

$\text{P.A.S} = O(\text{V.A.S})$

$\text{V.A.S} = O(\text{Disk A.S})$

Topic : Demand Paging

P.S = 1024 B ;

Suppose the page table for the process currently executing on the processor looks like the following. All numbers are decimal, everything is numbered starting from zero, and all addresses are memory byte addresses. The page size is 1,024 bytes.

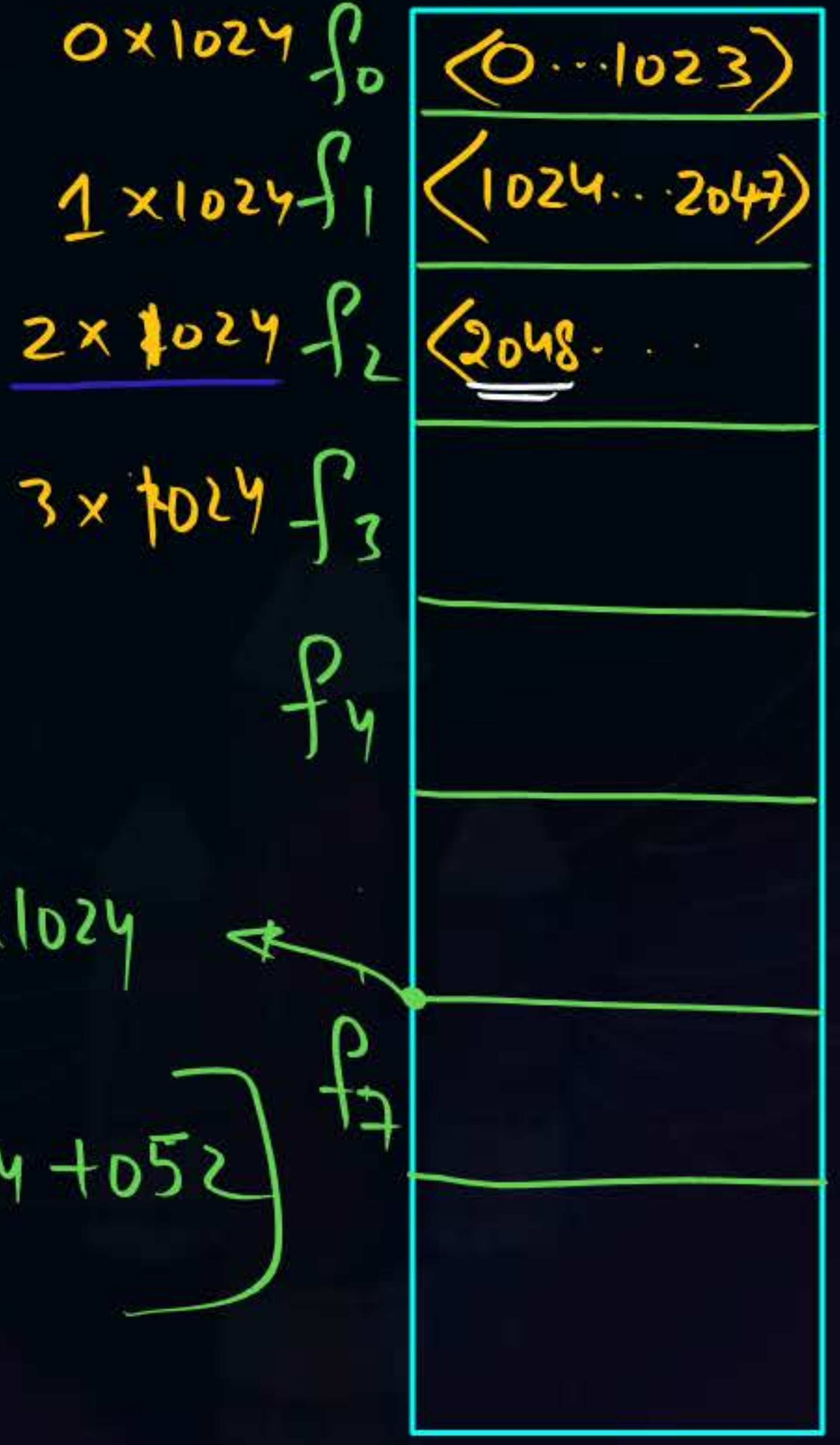
Virtual page number	Valid bit	Reference bit	Modify bit	Page frame number
0	1	1	0	4
1	1	1	1	7
2	0	0	0	—
3	1	0	0	2
4	0	0	0	—
5	1	0	1	0

a. Describe exactly how, in general, a virtual address generated by the CPU is translated into a physical main memory address.

b. What physical address, if any, would each of the following virtual addresses correspond to? (Do not try to handle any page faults, if any.)
   (i) 1,052 — P = 1 ; d = 052
   (ii) 2,221 → ×
   (iii) 5,499 → $\boxed{0+499}$ ✓

(i)

P.A



P.A = [7×1024 + 052]

Topic : Demand Paging

A process references five pages, A, B, C, D, and E, in the following order:

A; B; C; D; A; B; E; A; B; C; D; E

Assume that the replacement algorithm is first-in-first-out and find the number of page transfers during this sequence of references starting with an empty main memory with three page frames. Repeat for four page frames.

Topic : Demand Paging

Assuming a page size of 4 Kbytes and that a page table entry takes 4 bytes, how many levels of page tables would be required to map a $\underline{\text{64-bit address space}}$, if the top level page table fits into a single page?

H/W

Topic : Demand Paging

Segmented -Paging

Assume that a task is divided into four equal-sized segments and that the system builds an eight-entry page descriptor table for each segment. Thus, the system has a combination of segmentation and paging. Assume also that the page size is 2 Kbytes.

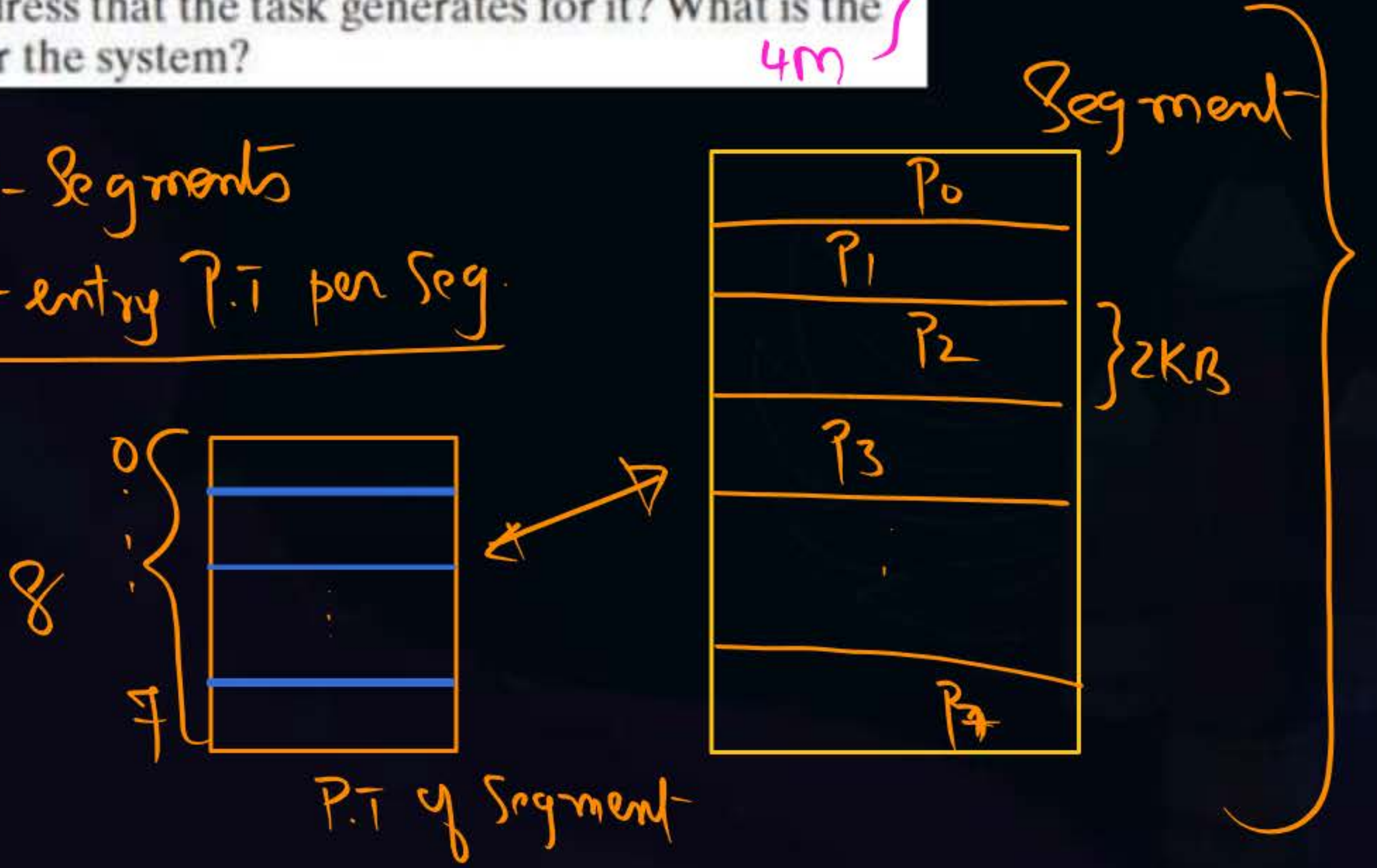
a. What is the maximum size of each segment? 2m
b. What is the maximum logical address space for the task? → P.A Size
c. Assume that an element in physical location $\boxed{00021ABC}$ is accessed by this task. What is the format of the logical address that the task generates for it? What is the maximum physical address space for the system? 4m

(HEX)

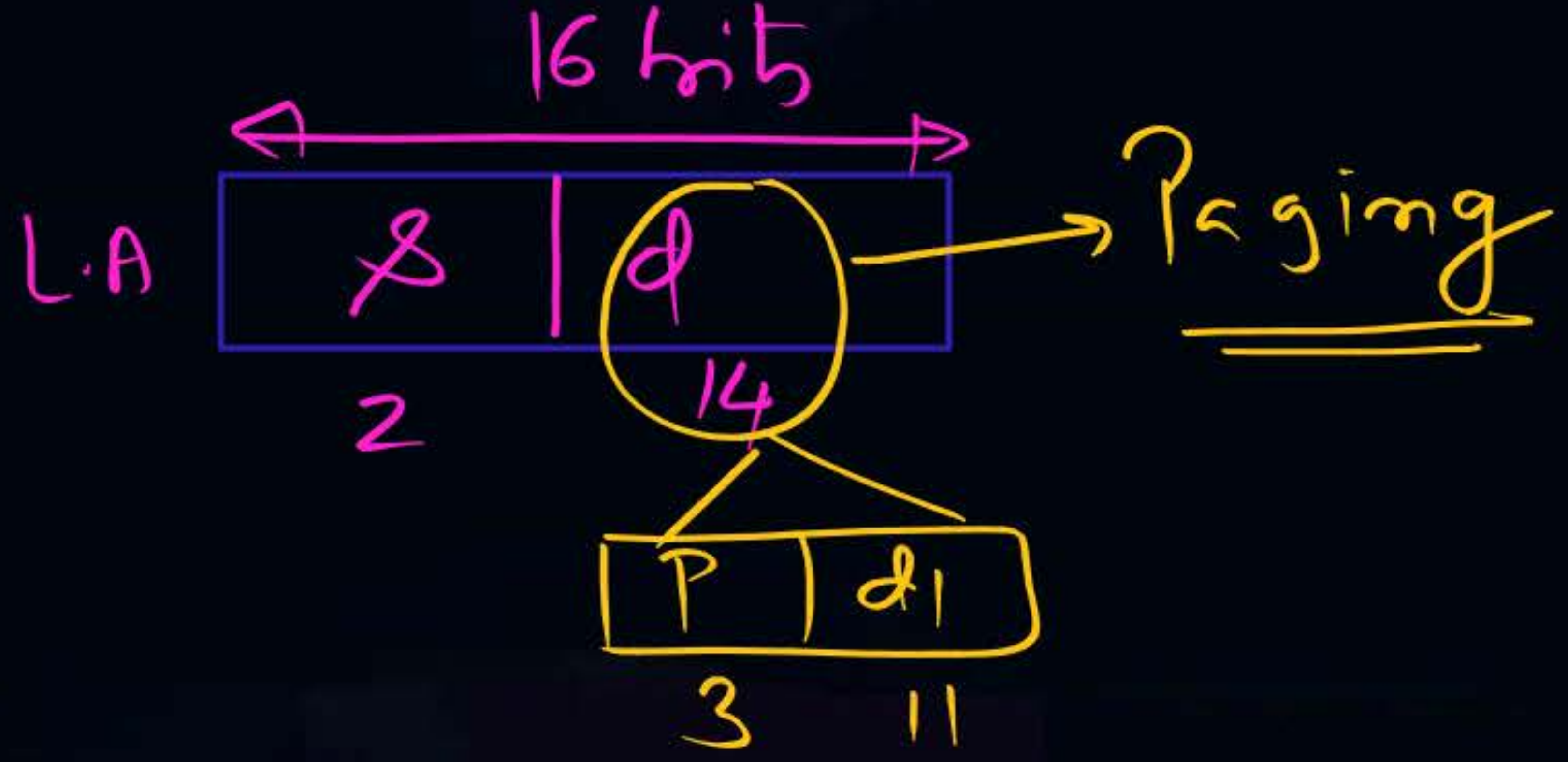
P.S = 2KB → 11

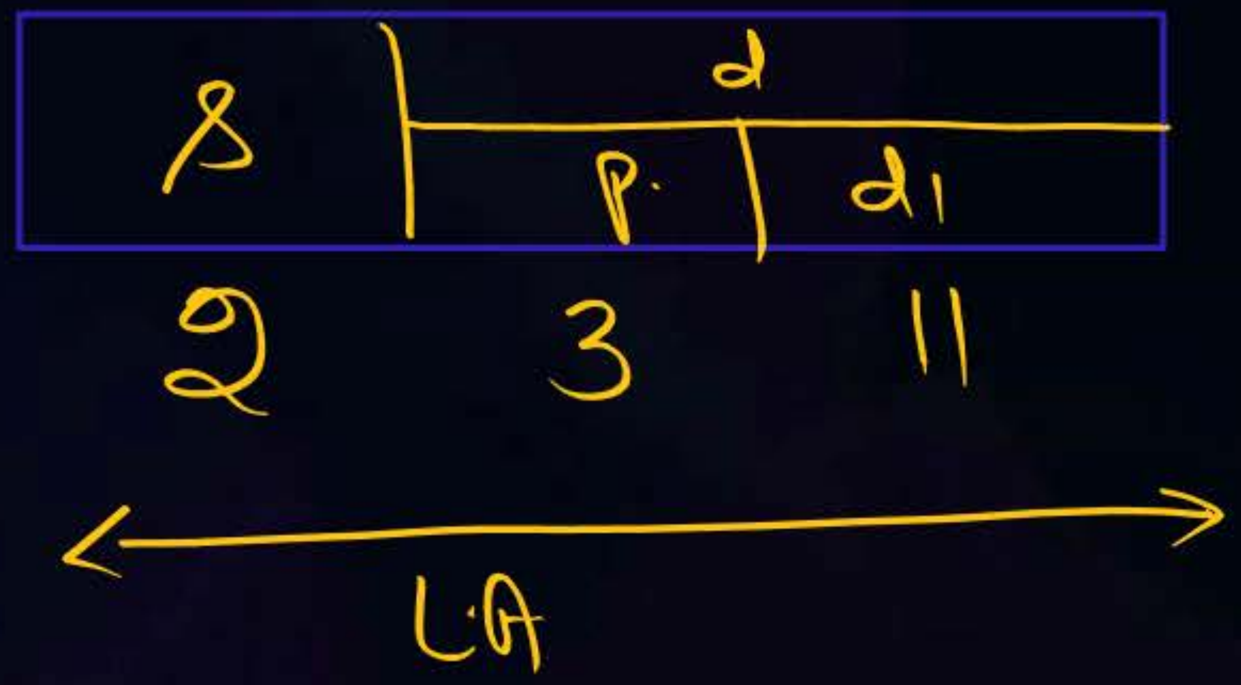
Process/Task → 4 - Segments
8 - entry P.T per Seg.

$\frac{V.A.S}{L.A.S} = 4 \times 16KB = 64KB$



8 × 2KB = 16KB ✓





$8 \text{ Hex digit} \rightarrow 4 \text{ bits}$

$\Rightarrow P.A = 8 \times 4 \text{ bits} = 32 \text{ bits}$

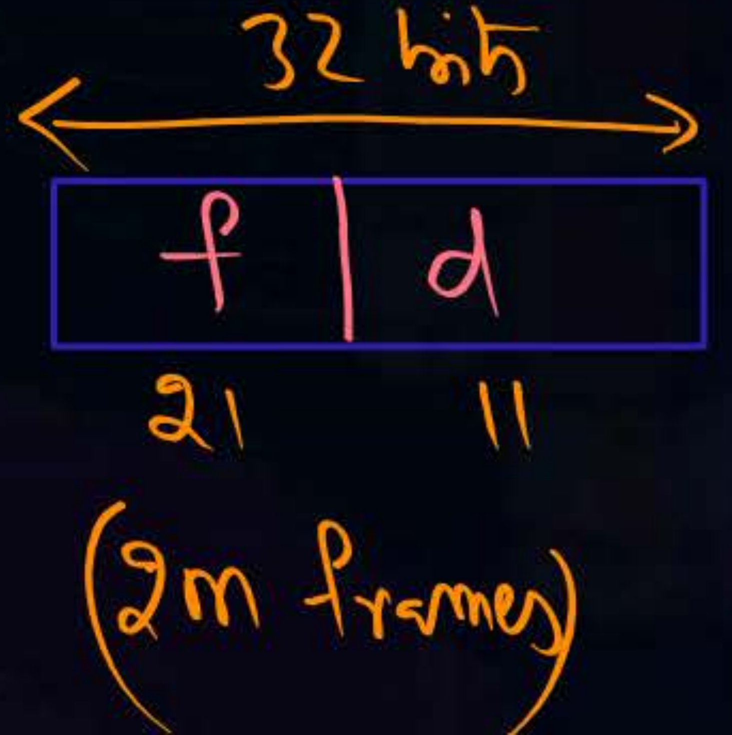
$\therefore PAS = 2^{32} = \underline{\underline{4GB}}$ ✓

P.A format:

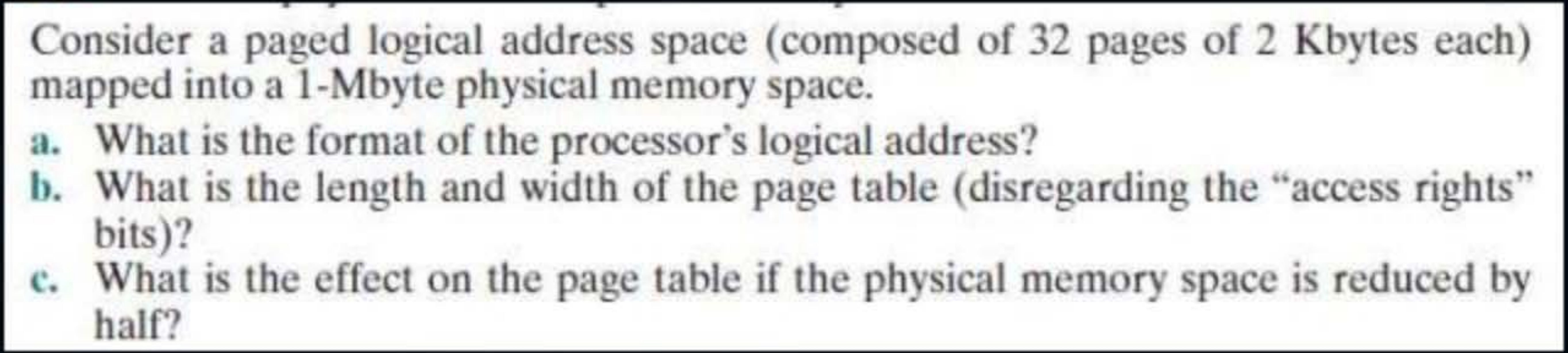
P.S = 2KB

F.S = 2KB

d = 11



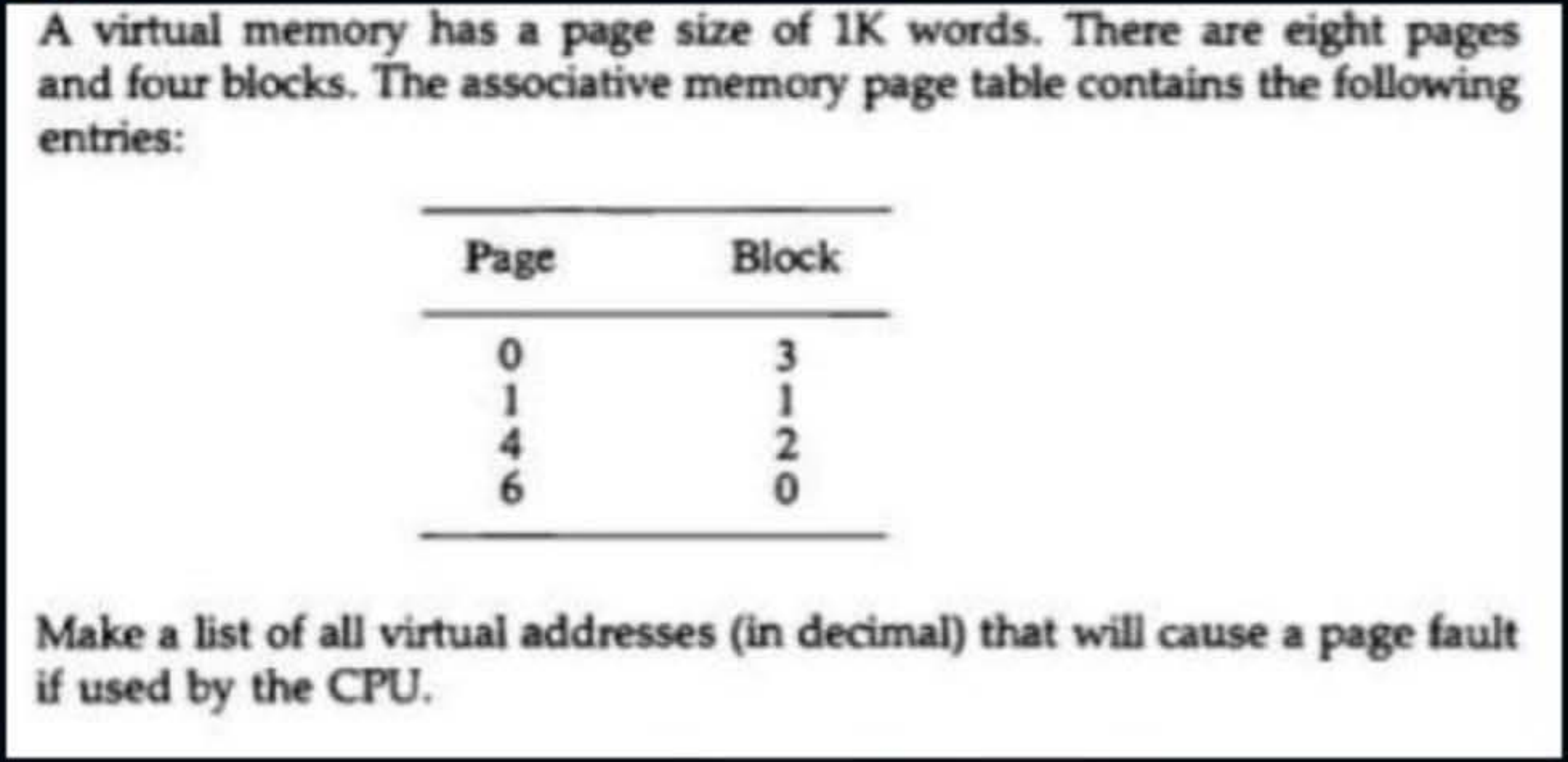
Topic : Demand Paging



Consider a paged logical address space (composed of 32 pages of 2 Kbytes each) mapped into a 1-Mbyte physical memory space.

a. What is the format of the processor's logical address?
b. What is the length and width of the page table (disregarding the "access rights" bits)?
c. What is the effect on the page table if the physical memory space is reduced by half?

Topic : Demand Paging



A virtual memory has a page size of 1K words. There are eight pages and four blocks. The associative memory page table contains the following entries:

Page	Block
0	3
1	1
4	2
6	0

Make a list of all virtual addresses (in decimal) that will cause a page fault if used by the CPU.

$P.S = 1024\ W$

$N_{pages} = 8\ ;\ M_{\frac{Frames}{Blocks}} = 4$

Pages in Memory : $\langle 0, 1, 4, 6 \rangle$

Pages that Cause P.F : $\langle 2, 3, 5, 7 \rangle$

List of V.A's in P_2 : $\langle 2048 \cdots 3071 \rangle$

2 mins Summary

Topic	One	: V.M Concept
Topic	Two	: Benefits
Topic	Three	: V.M Implementation
Topic	Four	: Perf. of V.M
Topic	Five	: Problem Solving

THANK - YOU