CS343: Operating System

Virtual Memory, Demand Paging, Page Replacement

Lect33 : 31st Oct 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Memory Management
  - Paging
  - **Virtual memory**
  - **Demand Paging**
  - **Page Replacement**
  - **Frame Allocation**

Memory Allocation: Top Down

Assume 4GB of RAM,
Avg Process Size 200MB
4 Segments/Process

OS
P5
P9
P8

P2

Finding Hole
First Fit
Worst fit
Best Fit

Very low complexity

Very High Mem Wastage

Process Continuous Allocation
10 process and 20 holes

Segment continuous Allocation
40 segment and 80 holes

low complexity

High Memory Wastage

Paging: 200MB process, 4KB page
Two Issue: pages and frames

Very Low Mem Wastage

High Complexity: 51200 pages/Process

Loading 51K pages in 10^6 frames

High Complexity: $4GB/4KB=2^{20}=10^6$ frames

4GB paged

SOL

Demand Paging Load the required 50-100 pages and load as an when necessary

SOL

Allocate 100-200 frames per process: reduce search Space

Virtual Memory

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

Virtual Memory-Background

- 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

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

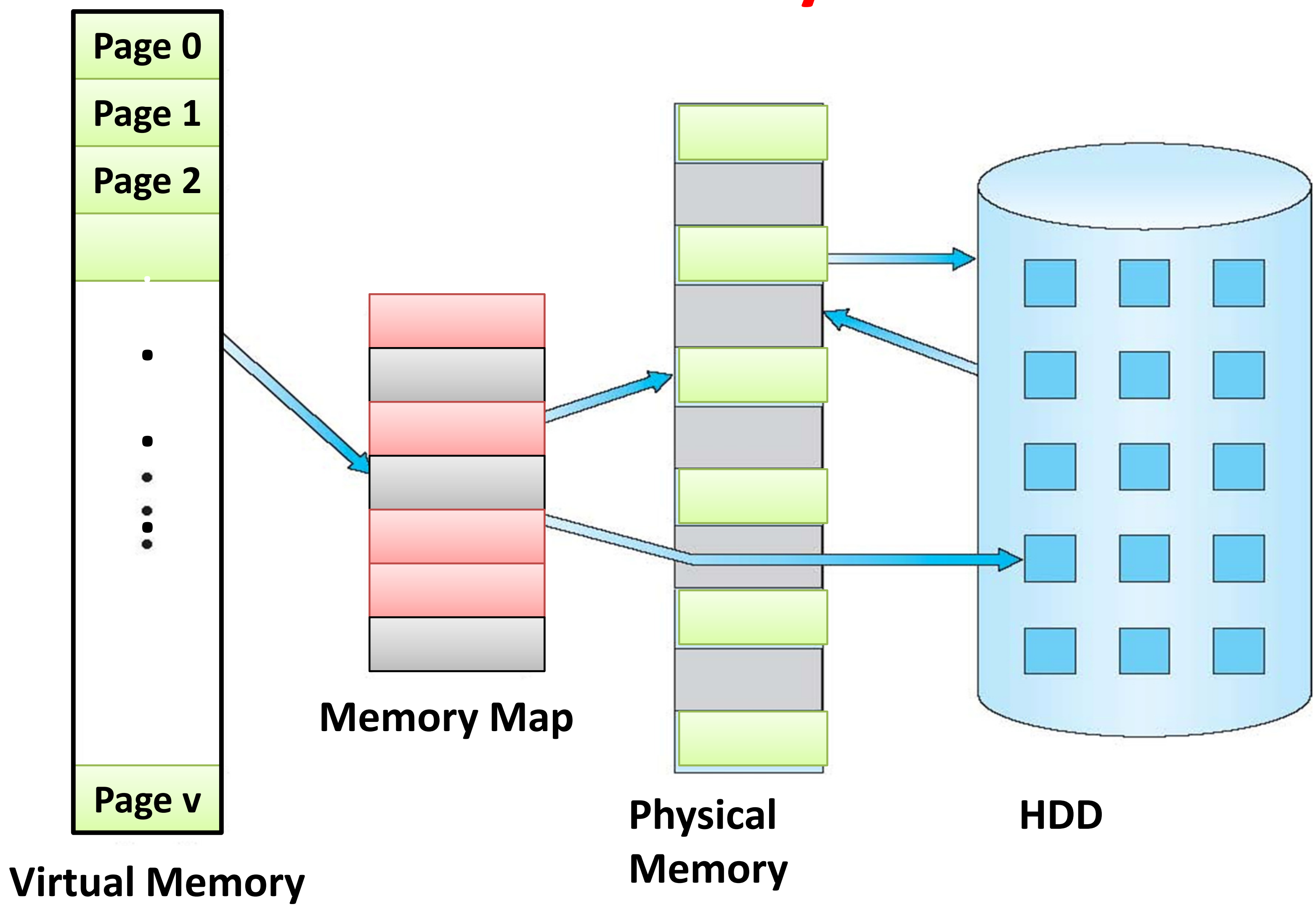
Virtual Memory

- 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

Virtual Memory

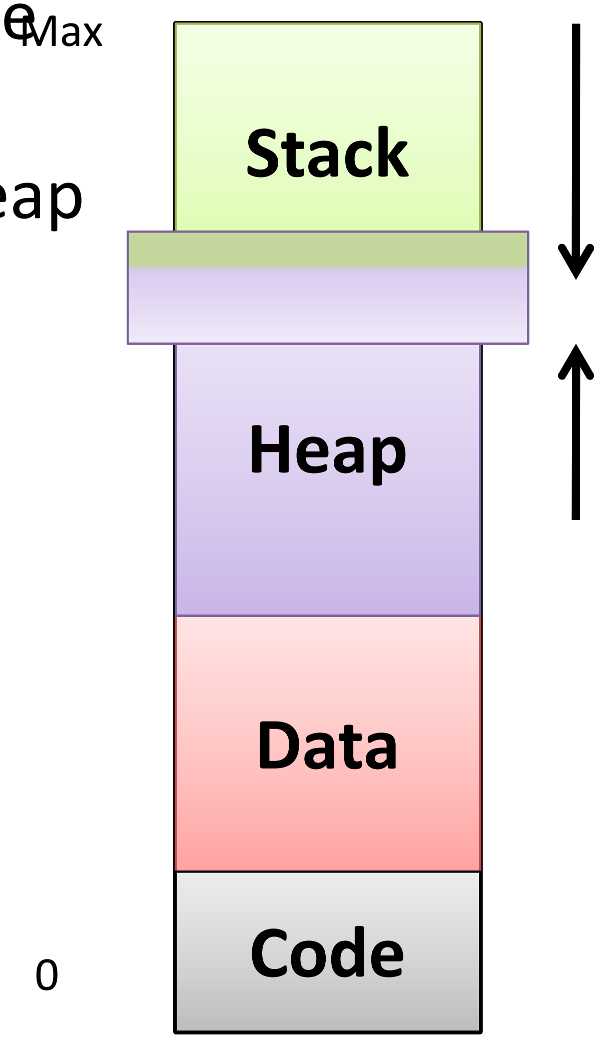
- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - **MMU must map logical to physical**
- Virtual memory can be implemented via:
  - Demand paging

Virtual Memory That is Larger Than Physical Memory



Virtual-address Space

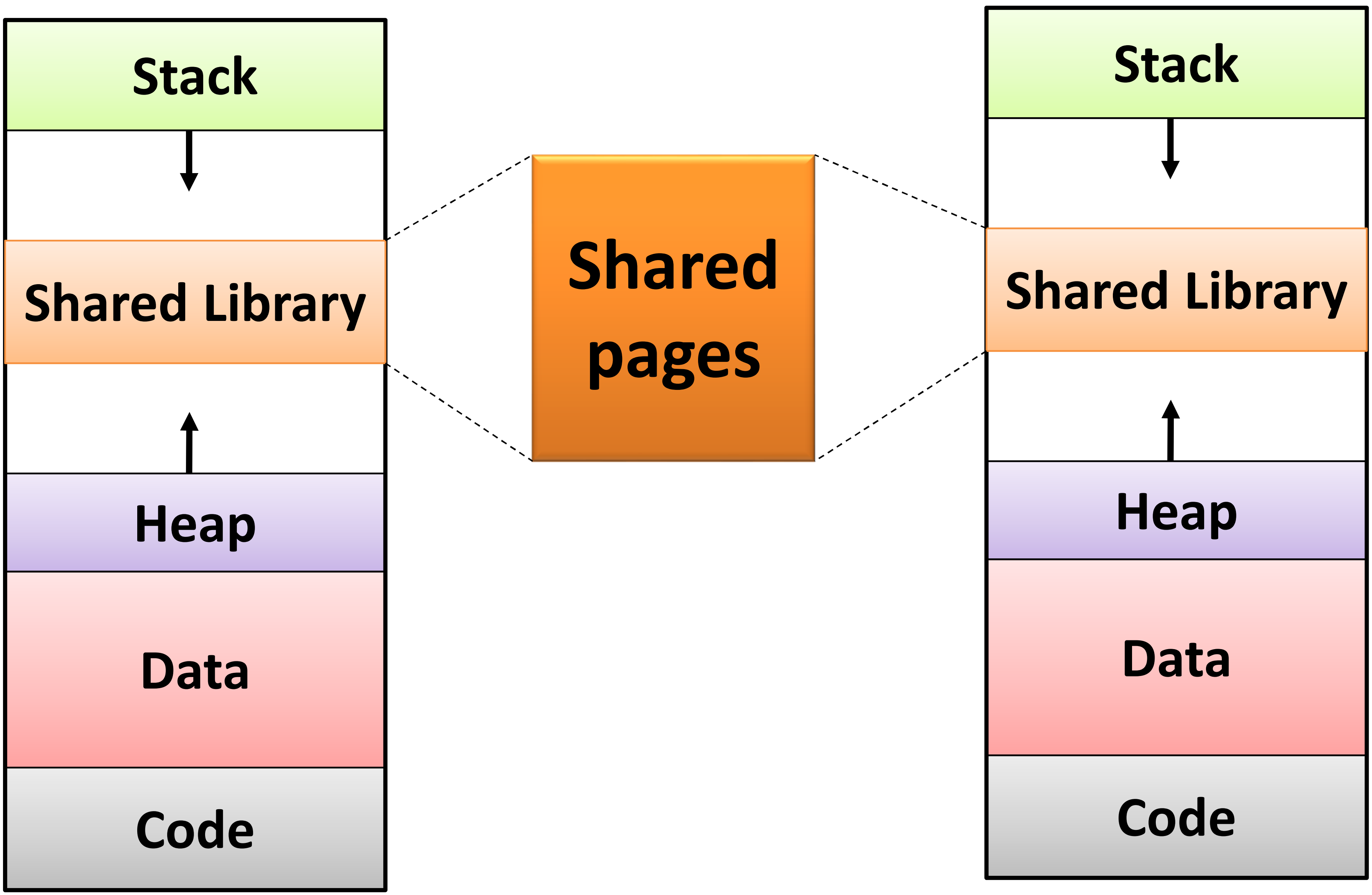
- Usually design logical address space for stack to start at Max logical address and grow "down" while heap grows "up"
  - Maximizes address space use
  - Unused address space between the two is hole
    - **Key: No physical memory needed until heap or stack grows to a given new page**



Virtual-address Space

- 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

Shared Library Using Virtual Memory



Demand Paging

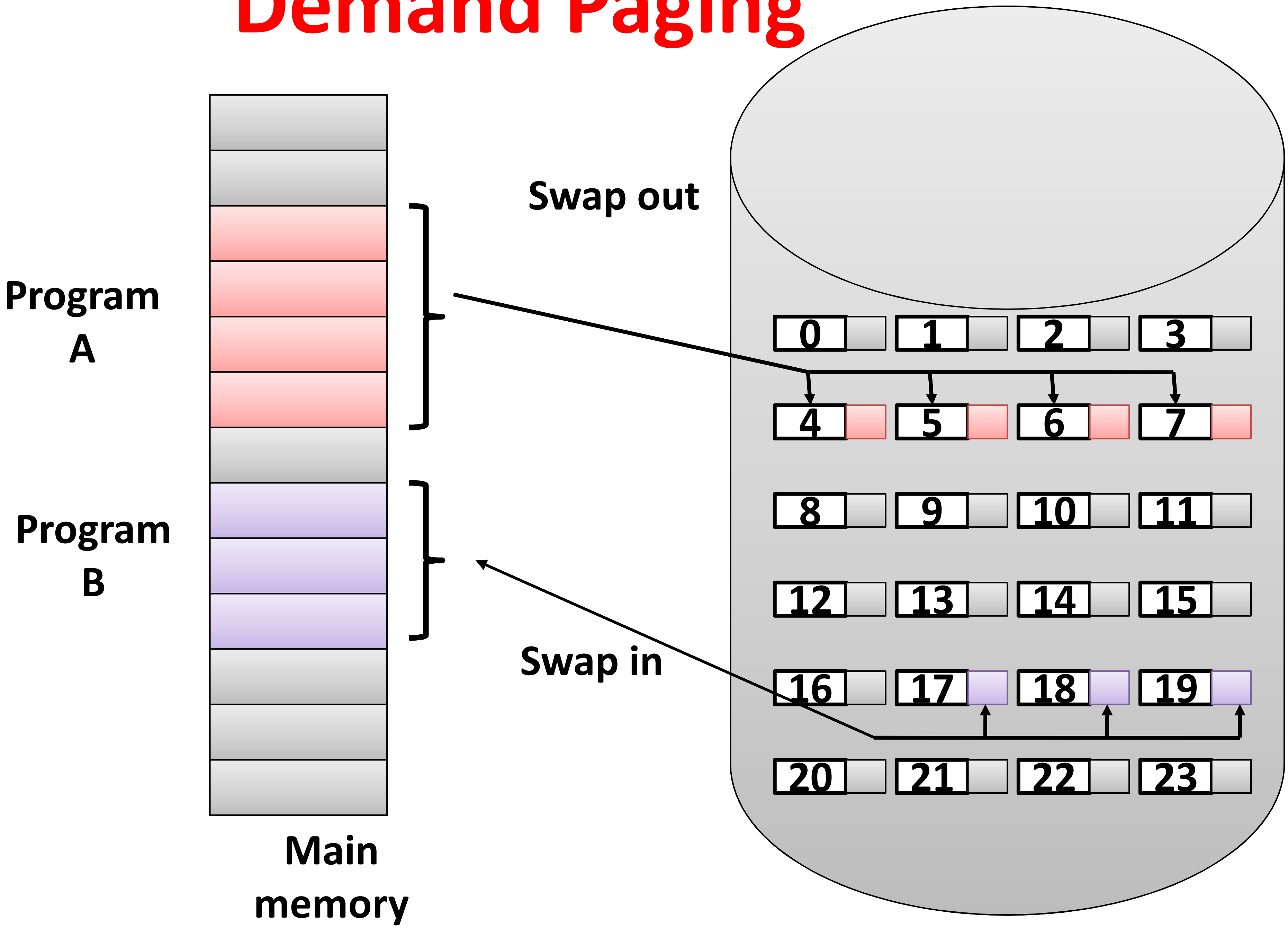
Demand Paging

- 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

Demand Paging

- Similar to paging system with swapping
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

Demand Paging



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**

Basic Concepts

- 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
    - **IMPortant : Without changing program behavior**
    - **IMPortant : Without programmer needing to change code**

Valid-Invalid Bit

- With each page table entry a ***valid–invalid*** bit is associated
  - **v** $\Rightarrow$ in-memory – **memory resident**
  - **i** $\Rightarrow$ not-in-memory
- Initially ***valid–invalid*** bit is set to **i** on all entries

Valid-Invalid Bit

- Example of a page table snapshot
- During MMU address translation,
  - if ***valid–invalid*** bit in page table entry is **i** ⇒ page fault

Frame #	Valid-invalid bit
20	V
21	V
22	I
….	…
210	I
211	v

Page Table

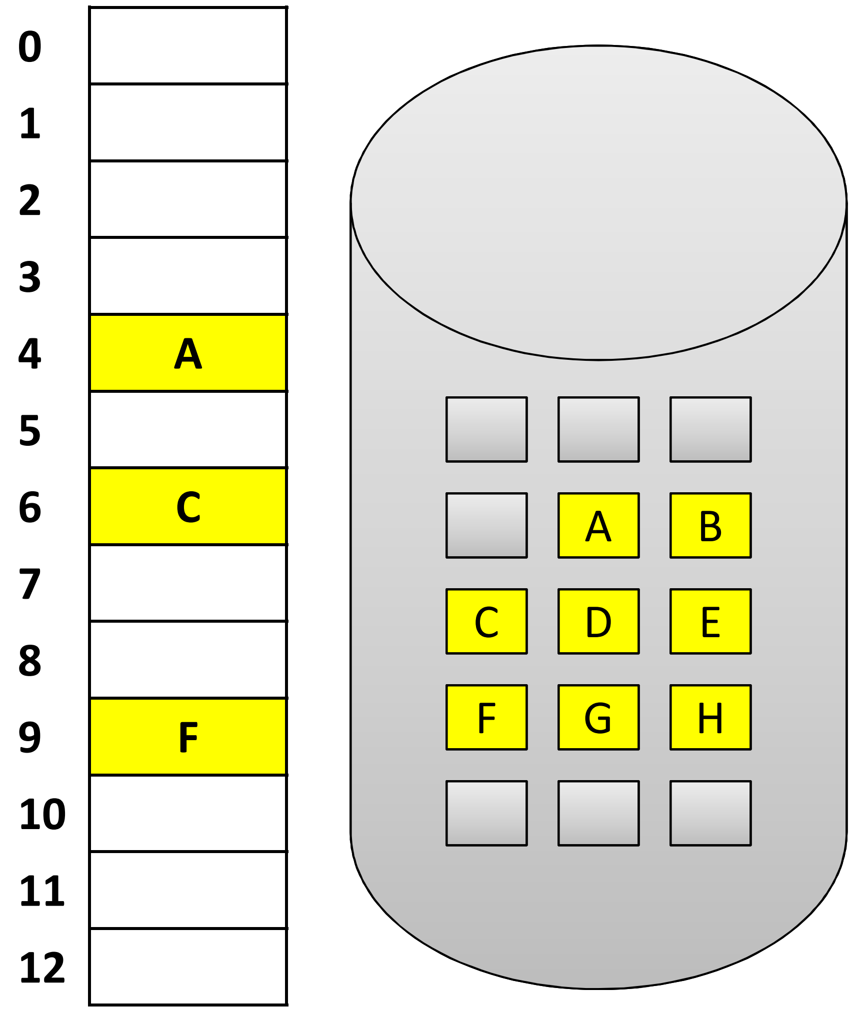
Page Table : Some Pages Are Not in Memory

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

Logical Memory

#P	#F	V
0	4	v
1		i
2	6	v
3		i
4		i
5	9	v
6		i
7		i

Page Table



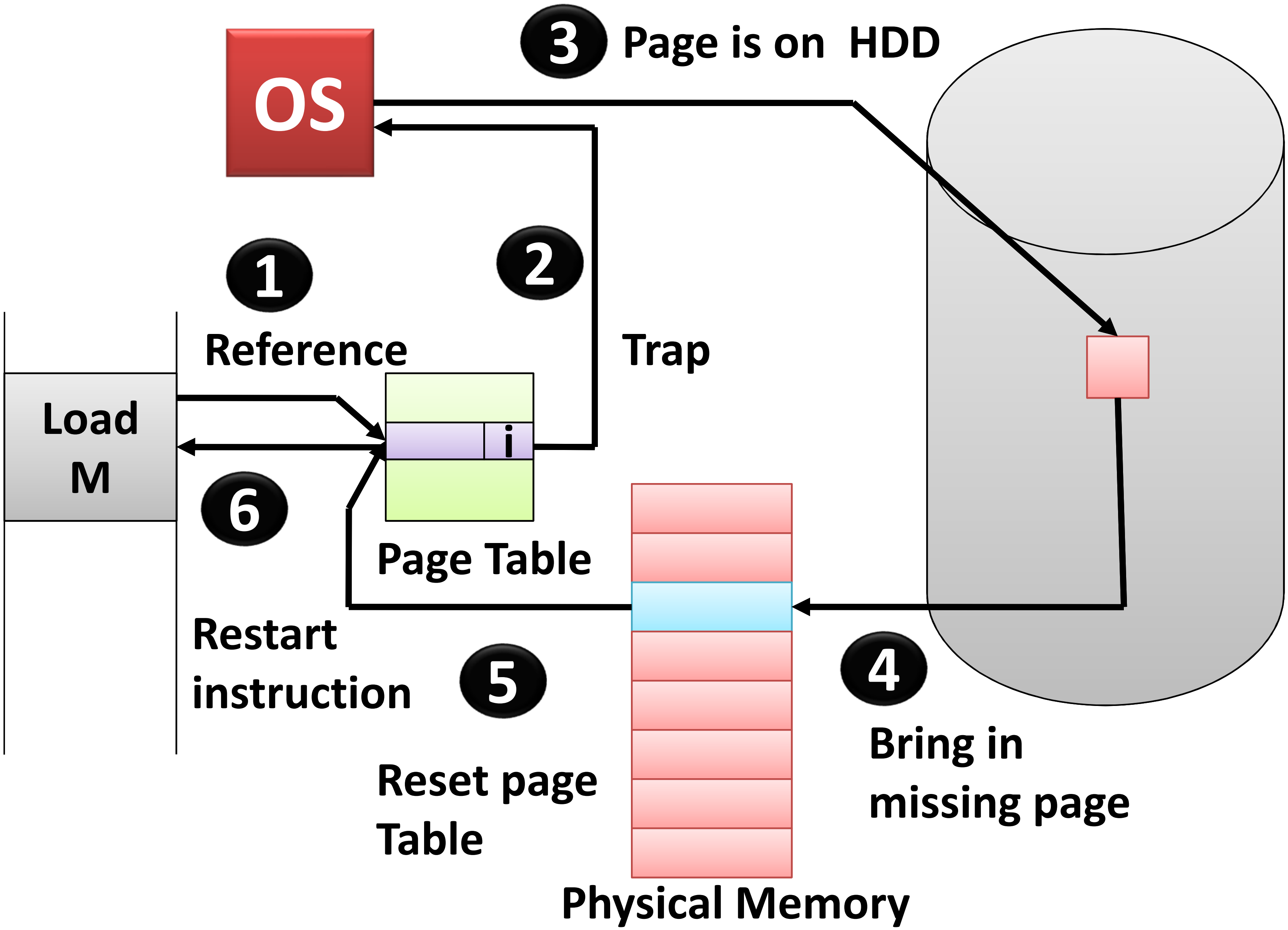
Main memory

Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system: **page fault**

1. Page reference
2. OS looks at page table to decide:
   - **Invalid reference $\Rightarrow$ abort**
   - Just not in memory
3. Find free frame
4. Swap page into frame via disk operation
5. Indicate page now in memory
   Set validation bit = **v**
6. Restart the instruction that caused page fault

Steps in Handling a Page Fault



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**

Aspects of 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

Page Fault along with the Short Term Scheduling

- When a page fault occurs it may switches to other process : Before switching
  - Page Replacement Algorithm
  - Initiate the I/O transfer
  - Save Context
- When pages comes
  - It generate an Interrupt, this process can be restarted
    - Save other process context before switching to the same
  - Page table need to be modified
  - Same process need to be restarted

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 $0 \leq p \leq 1$
- Effective Access Time (EAT)

$$
\begin{aligned}
EAT = (1 - p) &\times \text{memory access} \\
&+ p\ (\text{page fault overhead} \\
&\quad + \text{swap page out} + \text{swap page in})
\end{aligned}
$$

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 out of 1,000 causes a page fault, then

  EAT = 8.2 microseconds.

  This is a slowdown by a factor of **40!!**

Demand Paging Example

- **Virtual memory similar to caching in processor level**
  - **Except higher page fault rate is not acceptable**
  - **Page fault rate < < Cache miss rate**
- If want performance degradation < 10 percent
  - $220 > 200 + 7{,}999{,}800 \times p$
    $20 > 7{,}999{,}800 \times p$
  - $p < .0000025$
  - < one page fault in every 400,000 memory accesses