BITS Pilani
K K Birla Goa Campus

# Operating Systems

**Dr. Lucy J. Gudino**
**Dept. of CS and IS**

# Last Class

- Segmentation
- Segmentation hardware
- Protection
- Virtual Memory

# Virtual Memory

**BITS** Pilani
K K Birla Goa Campus

# Introduction

- How to increase the degree of multiprogramming ?
- Logical vs physical memory
- Virtual memory is a technique that allows the execution of processes that are not completely in memory.
- Motivation:
  - All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time
  - A process may be broken up into a number of pieces (pages or segments) and these pieces need not be contiguously located in main memory during execution.
  - Programs often have code to handle unusual error conditions which is almost never executed.

# Contd…

- Arrays, lists, and tables are often allocated more memory than they actually need.
- Certain options and features of a program may be used rarely
- Principle of locality
- trashing is a condition where system spends more time in swapping than executing instructions

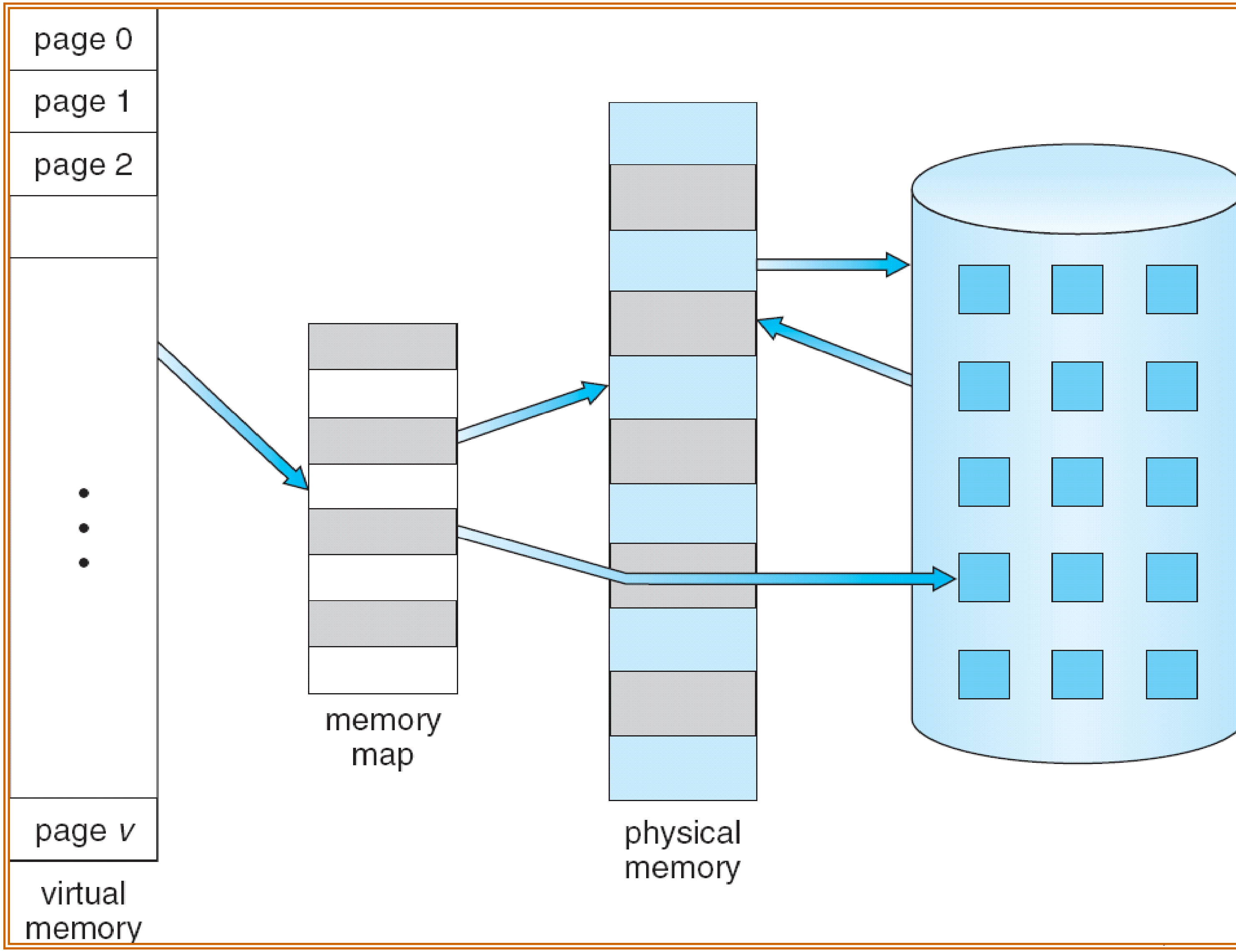
# Contd…

- Advantages:
  - A program would no longer be constrained by the amount of physical memory that is available
  - Because each user program could take less physical memory, more programs could be run at the same time
  - increases the CPU utilization and throughput
  - Less I/O would be needed to load or swap each user program into memory, so each user program would run faster

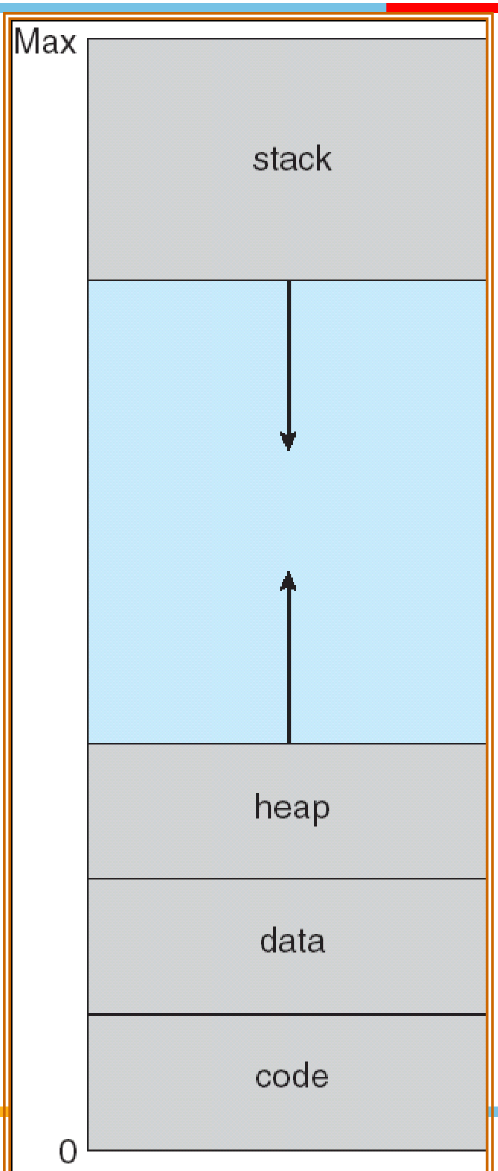
# Background

- **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

# Virtual Memory That is Larger Than Physical Memory



# Virtual-address Space

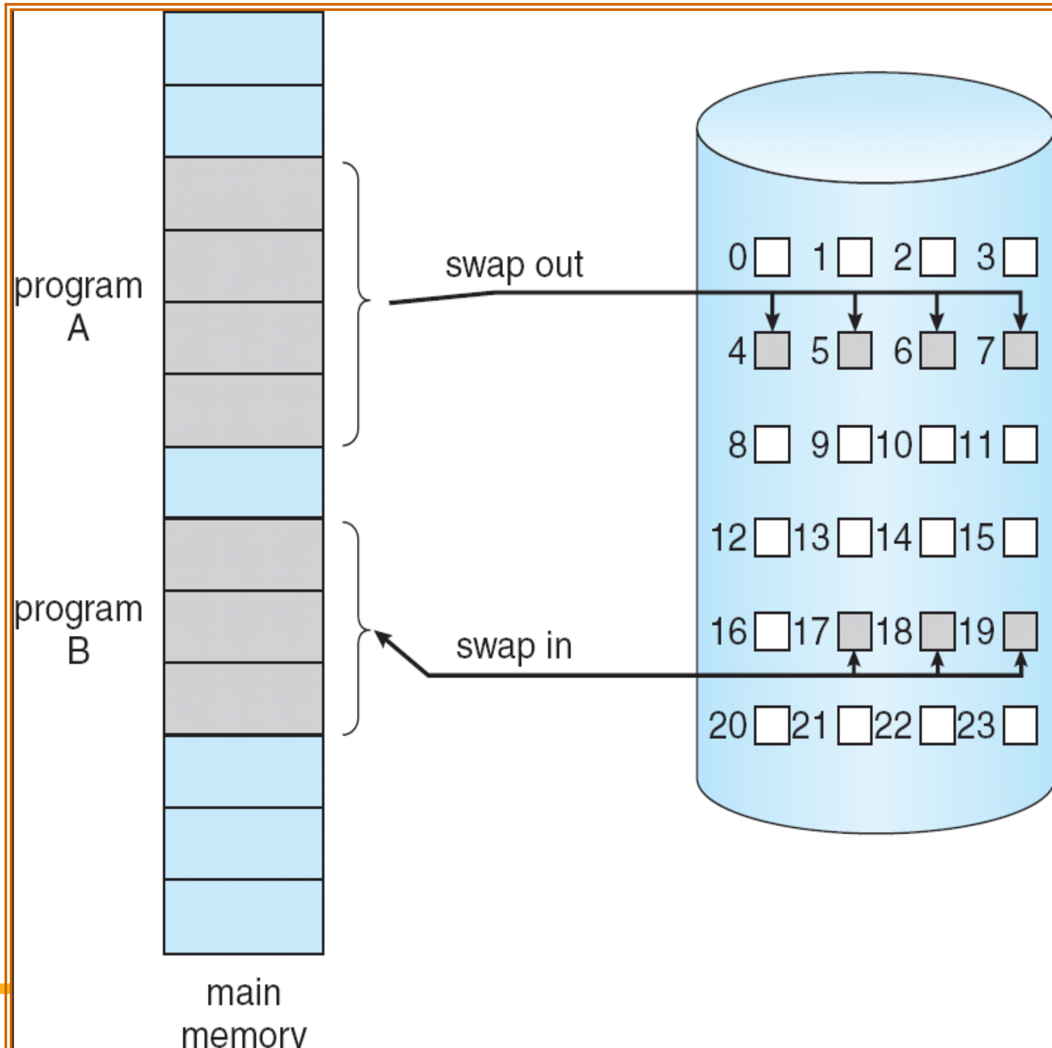


# Virtual Memory Implementation

- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

# Demand Paging

- Bring a page into memory only when it is needed
  - – similar to swapping

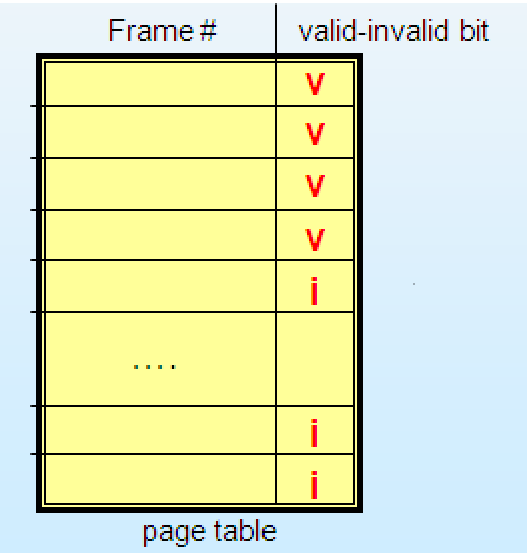


# Demand Paging

- Bring a page into memory only when it is needed
  - similar to swapping
  - uses **Lazy swapper** – never swaps a page into memory unless page is needed
    - Swapper that deals with pages is a **pager**
- Advantages:
  - 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

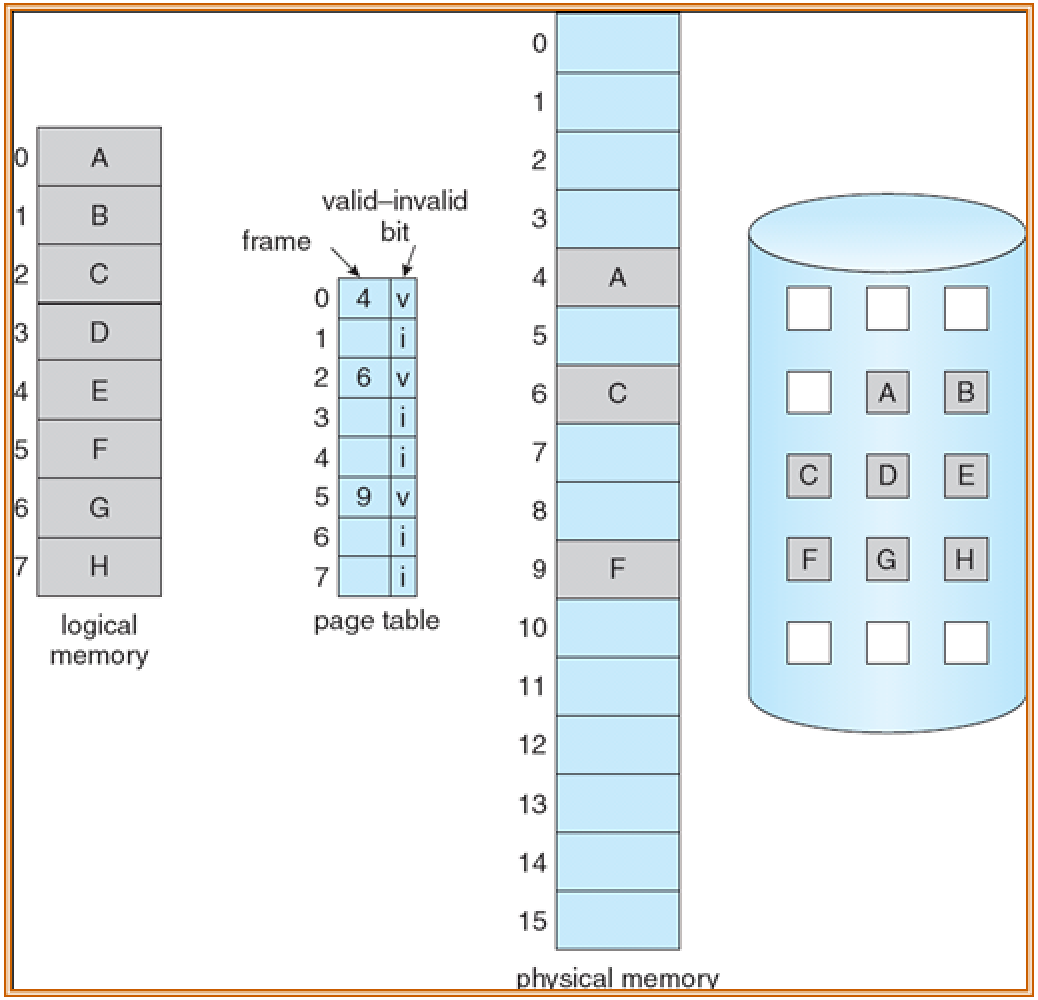
# Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated ($\mathbf{v} \Rightarrow$ in-memory, $\mathbf{i} \Rightarrow$ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



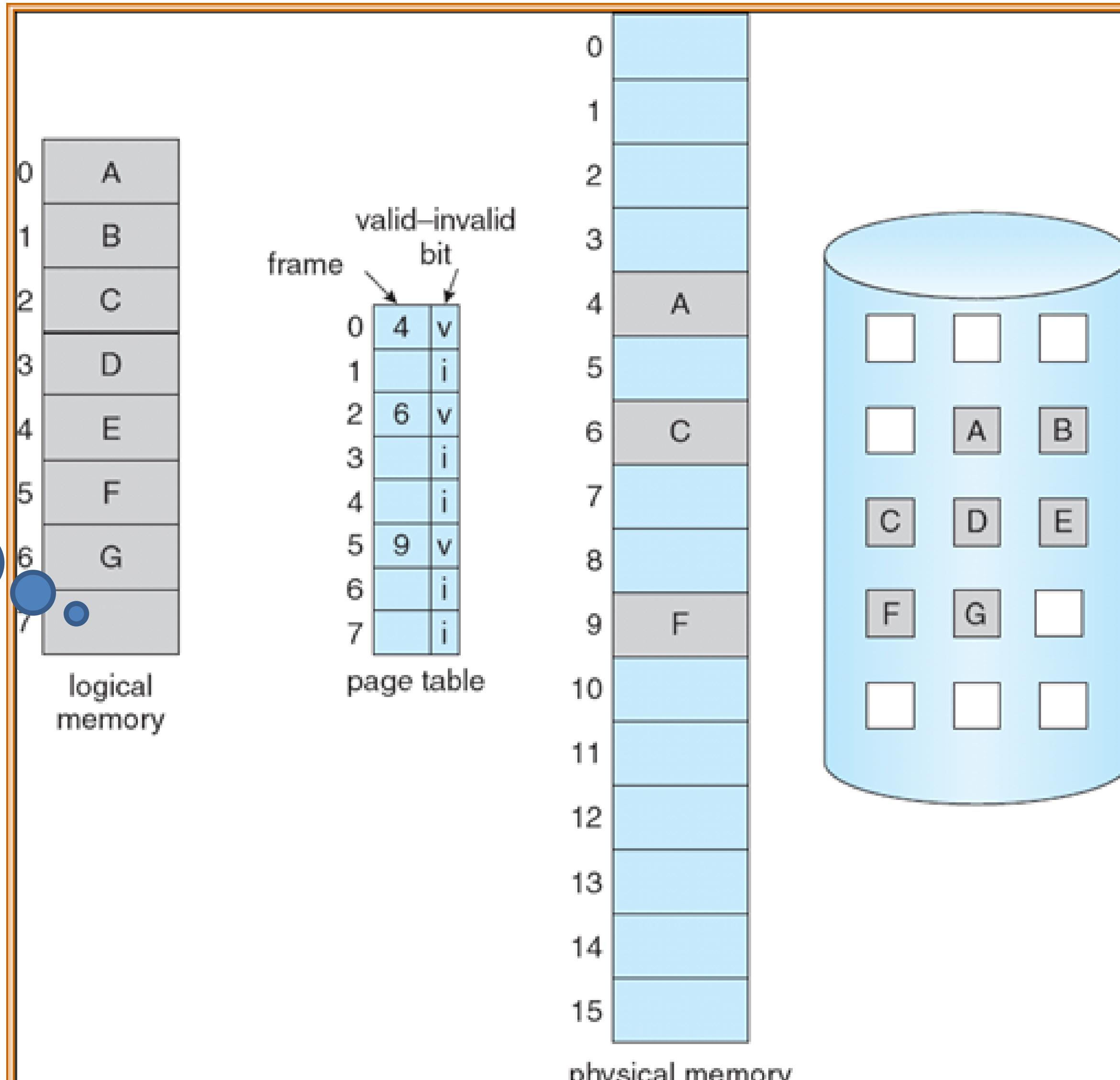

- During address translation, if valid–invalid bit in page table entry is **I** $\Rightarrow$ page fault

# Page Table When Some Pages Are Not in Main Memory



# Contd…

If logical address corresponds to page 7?



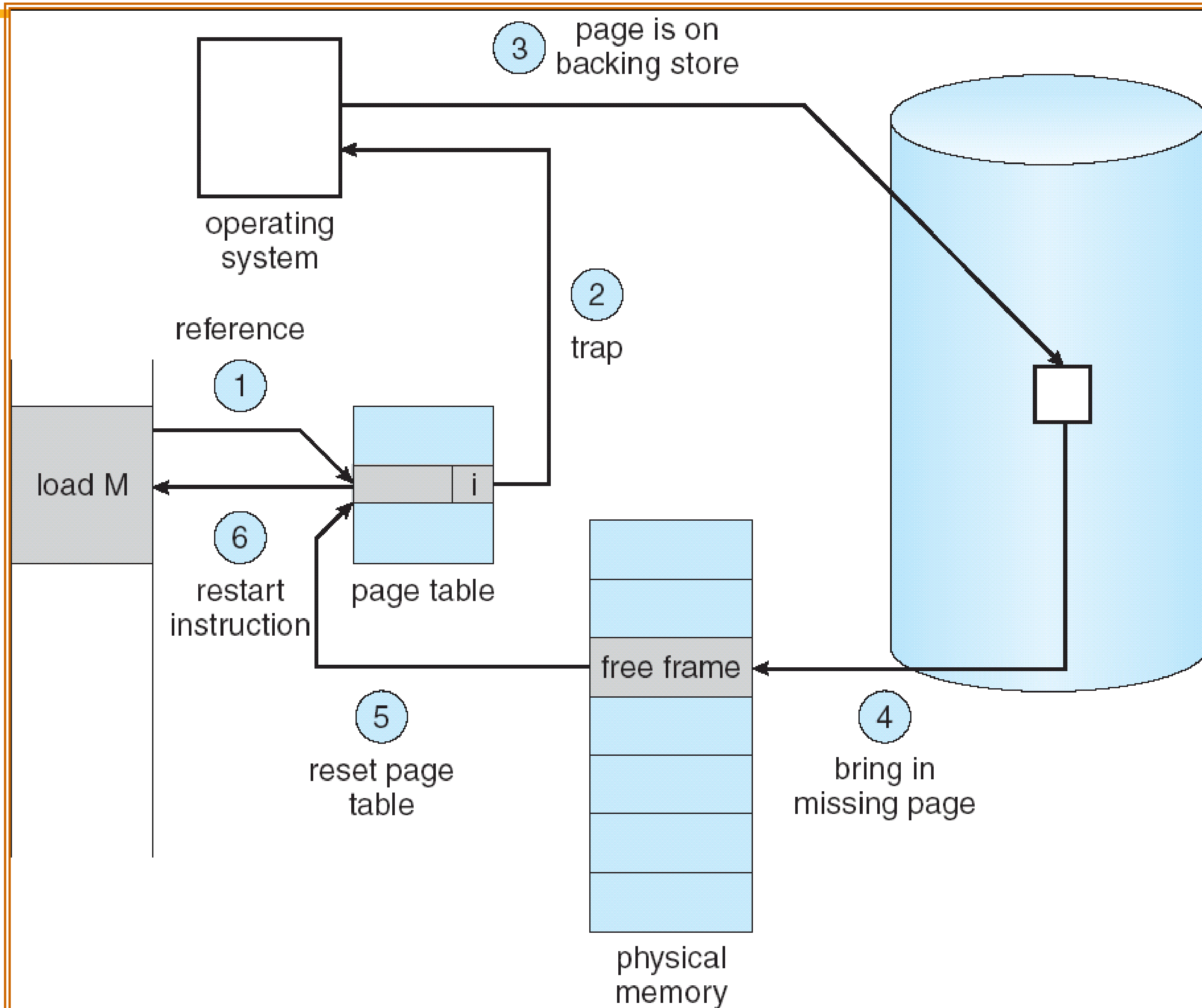
# Page Fault

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

**page fault**

1. Operating system looks at another table to decide:
   - Invalid reference $\Rightarrow$ abort
   - Just not in memory
2. Find empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

# Steps in Handling a Page Fault



# Hardware support

- Same as the hardware for paging and swapping
  - Page table
  - Secondary memory  also known as swap device
- How to restart after a page fault?
  - Page fault during instruction fetch
  - Page fault during data fetch

# Performance of Demand Paging

- Effective Access Time (EAT)
- EAT = (1 – $p$) x ma + $p$ x page fault time
- where
- Page Fault Rate $p$: $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault
- ma : memory access

# Sequence of events during page fault

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

# Contd…

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

# Demand Paging Example

Memory access time = 200 nanoseconds

Average page-fault service time = 8 milliseconds

One access out of 1,000 causes a page fault

EAT = (1 – *p*) x ma + *p* x page fault time

EAT = (1 – p) x 200ns + p x(8 ms)

then

EAT = 8.2 microseconds.

This is a slowdown by a factor of 40!!