Operating Systems

Virtual Memory

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Fall 2023

Copyright Notice

Slides are based on the slides of the main textbook.

Silberschatz

https://www.os-book.com/OS10/slide-dir/index.html

Chapter 10: Virtual Memory

- Background

- Demand Paging

- Copy-on-Write

- Page Replacement

Amirkabir University of Technology
(Tehran Polytechnic)

Objectives

- Define virtual memory and describe its benefits.

- Illustrate how pages are loaded into memory using demand paging.

- Apply the FIFO, optimal, and LRU page-replacement algorithms.

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

```
try {

  //code may cause exception
}


catch (Exception e) {

  // code that handles an exception

}


finally {

  // default piece of code

}
```

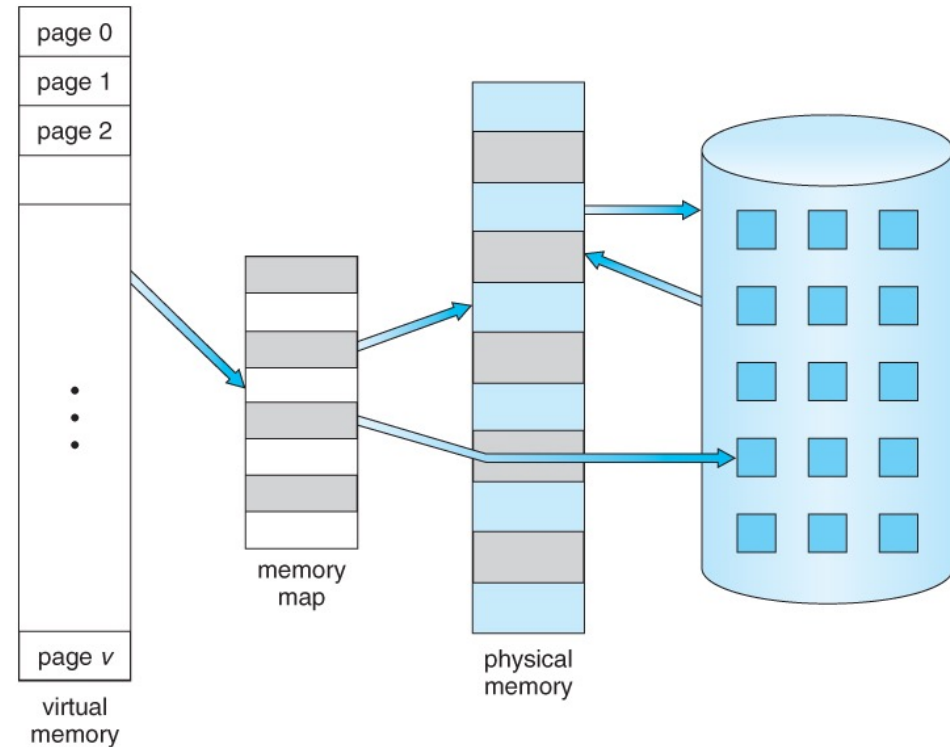# Benefits of Executing Partially-load Programs

- Program no longer constrained by limits of physical memory
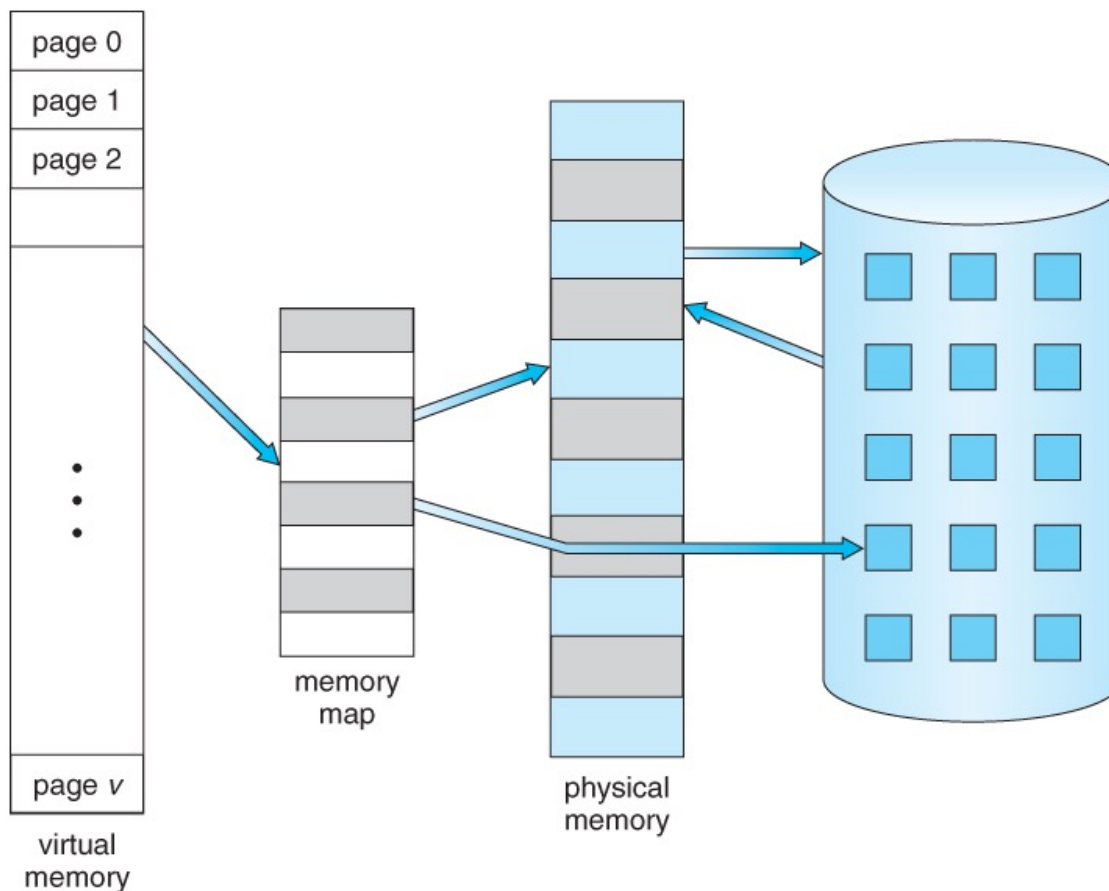
Benefits of Executing Partially-load Programs

- 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.

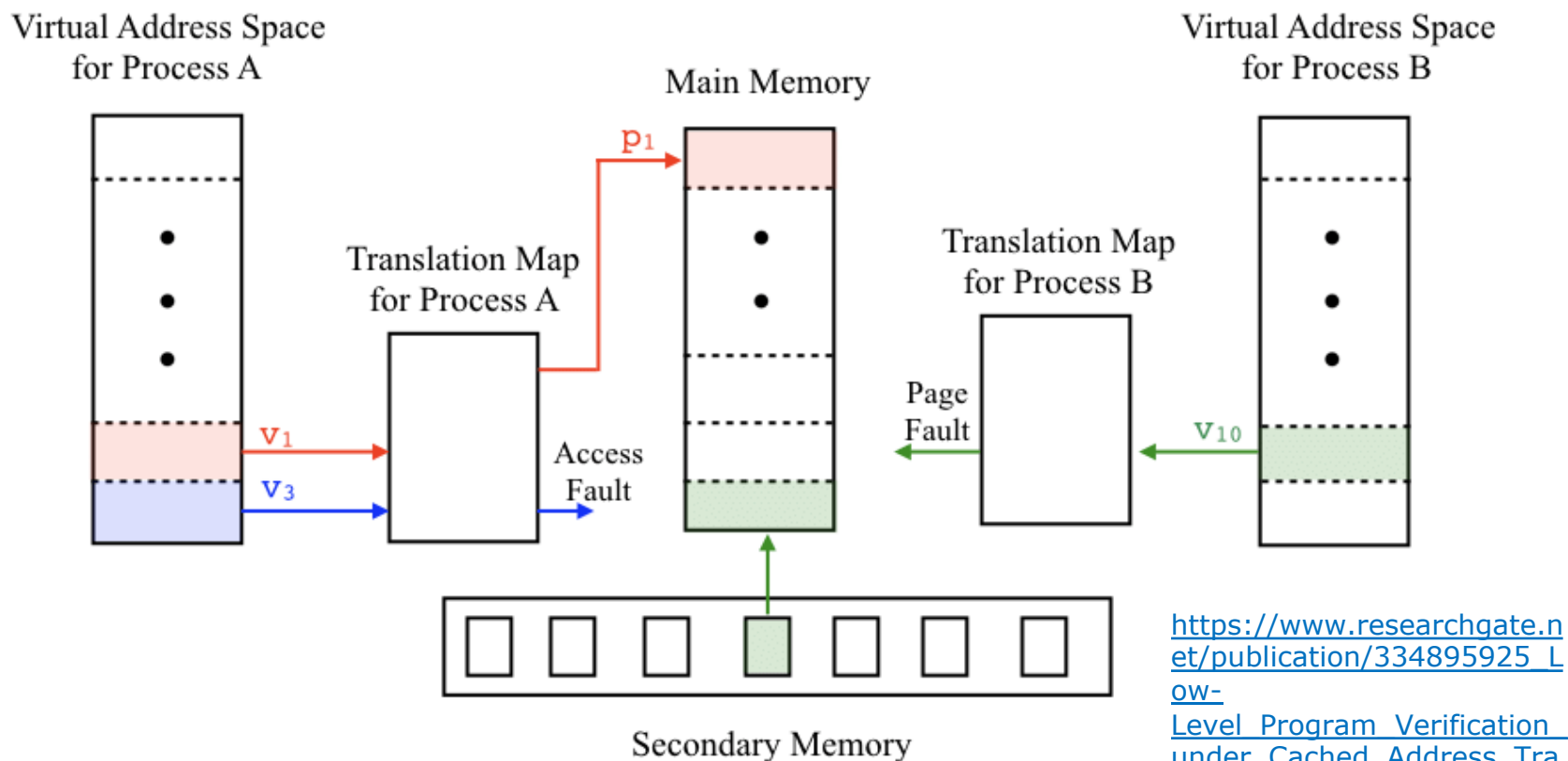# Benefits of Executing Partially-load Programs

- Less I/O needed to load or swap programs into memory -> **each user program runs faster.**
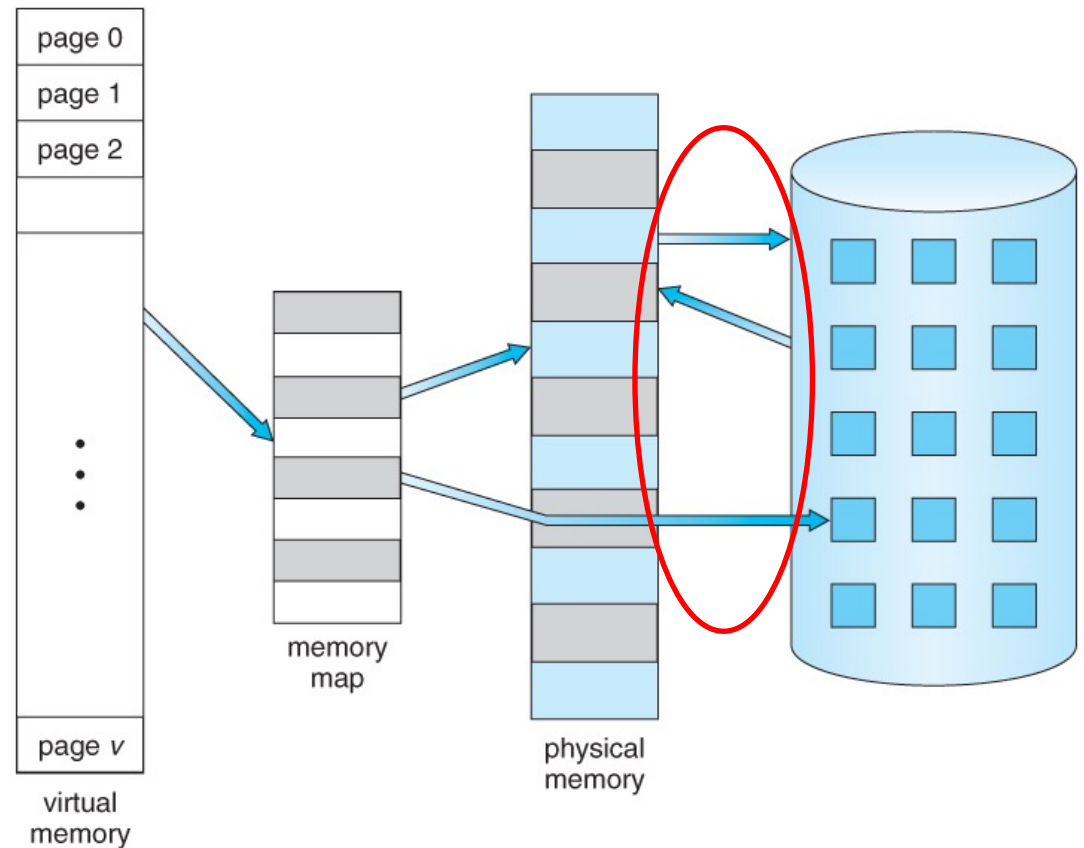
Virtual memory

separation of user logical memory from physical memory

Only part of the program needs to be in memory for execution

Virtual Address Space for Process A

Main Memory

Virtual Address Space for Process B

Translation Map for Process A

Translation Map for Process B

p_1

v_1

v_3

Access Fault

Page Fault

v_{10}

Secondary Memory

https://www.researchgate.net/publication/334895925_Low-Level_Program_Verification_under_Cached_Address_Translation/figures?lo=1
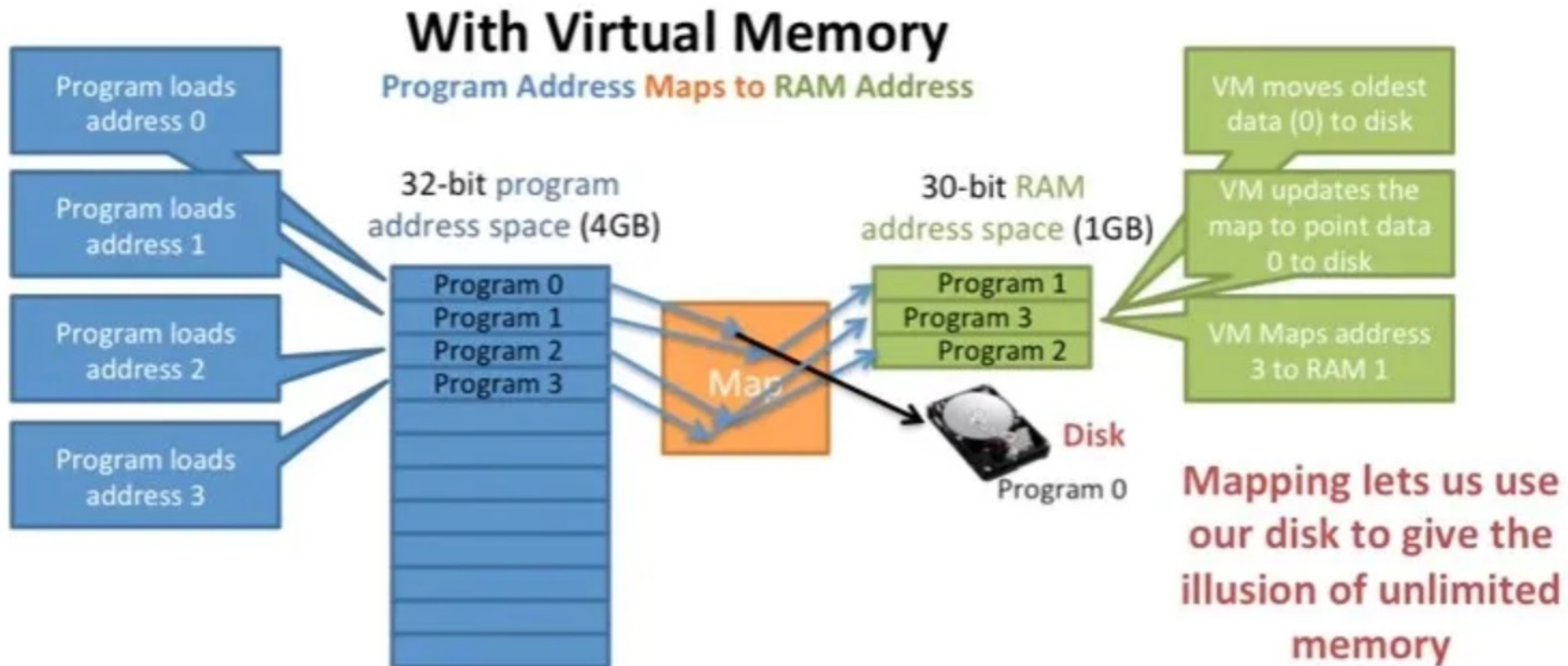
Benefits of Virtual memory

- Logical address space can be much larger than physical address space
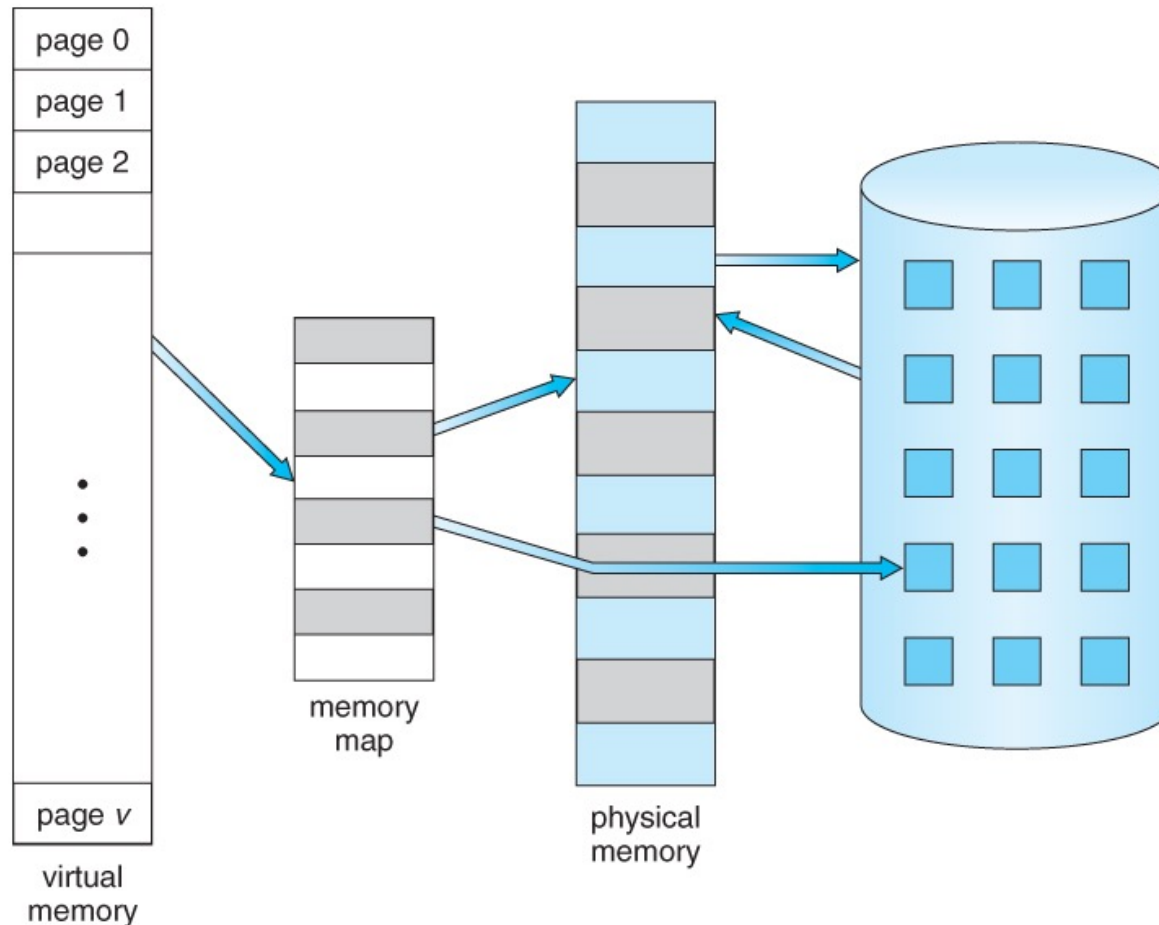
Benefits of Virtual memory

- Allows address spaces to be shared by several processes

Benefits of Virtual memory

- Allows for more efficient process creation

- More programs running concurrently

- Less I/O needed to load or swap processes

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
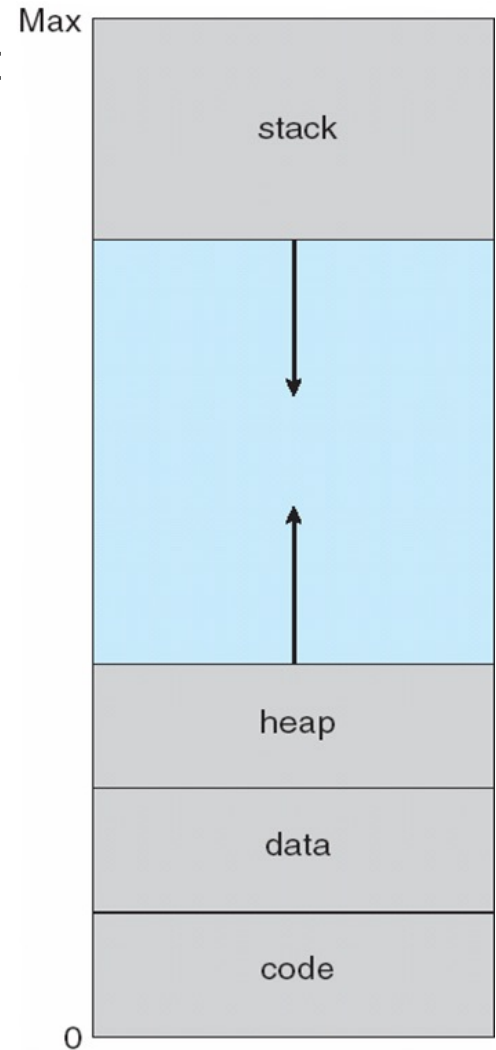
- Virtual memory can be implemented via:

  - Demand **paging**

  - Demand **segmentation**

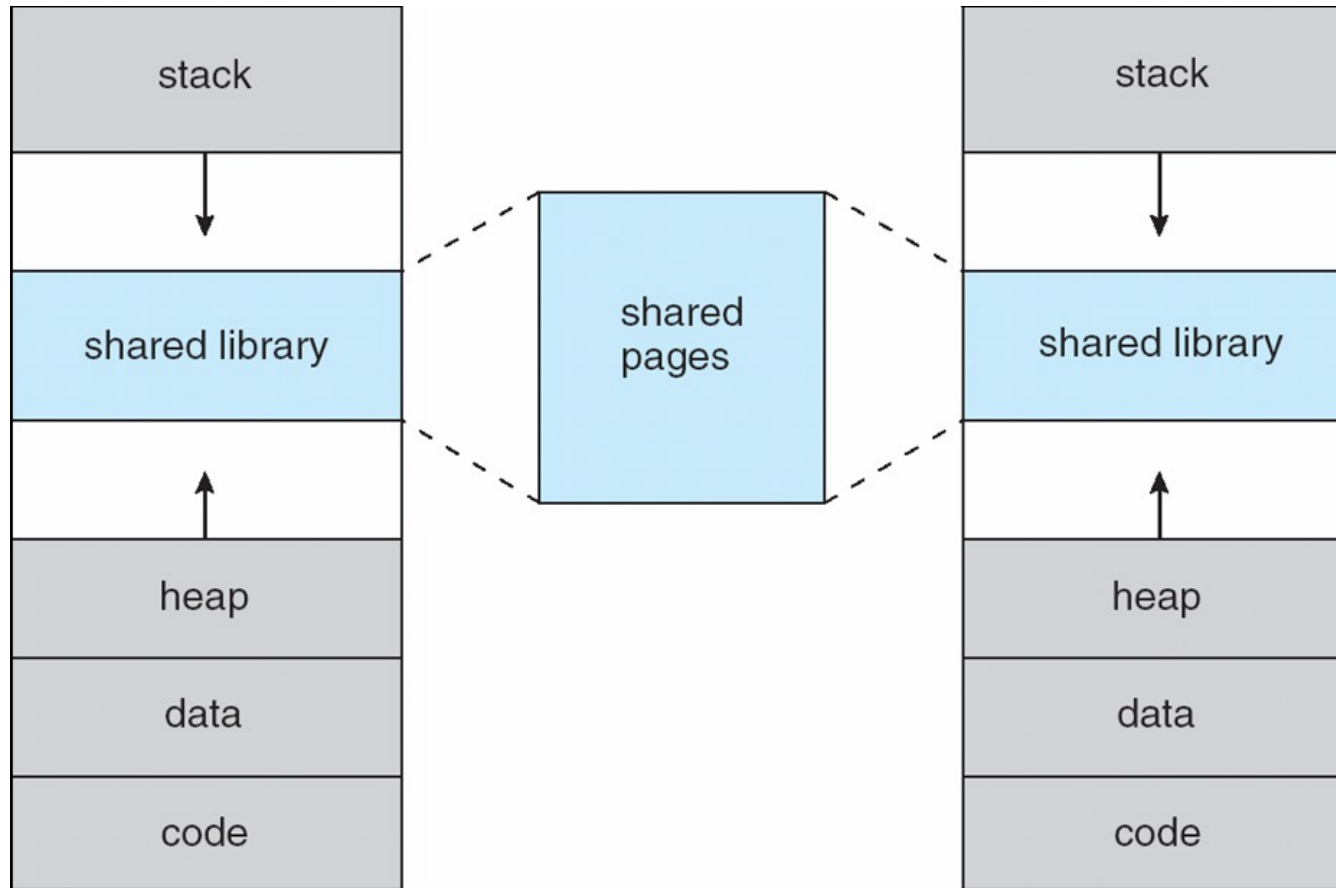# Virtual-address Space

- 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
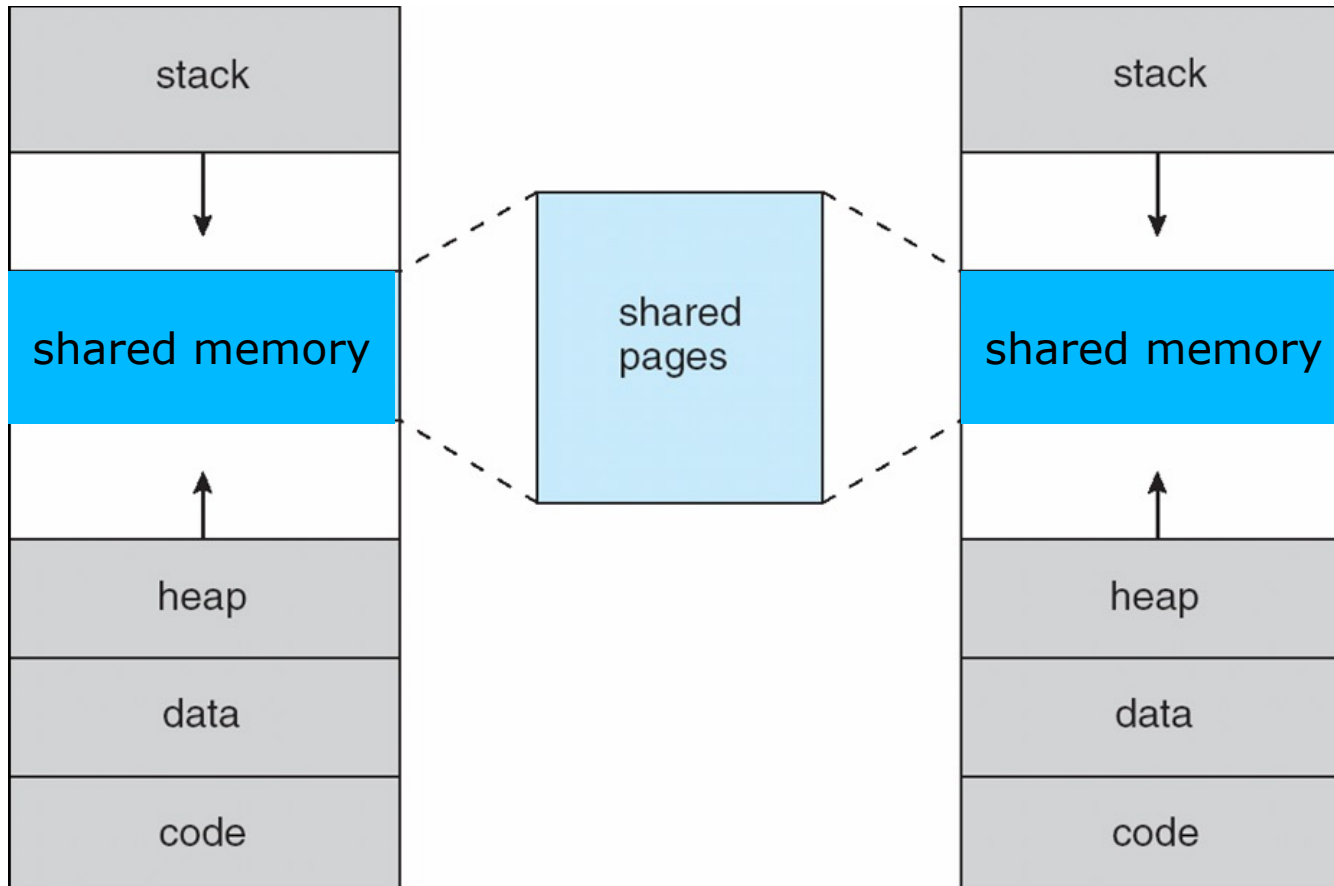
Virtual-address Space (cont.)

- System libraries shared via mapping into virtual address space

Virtual-address Space (cont.)

- Shared memory by mapping pages read-write into virtual address space

Virtual-address Space (cont.)

- Pages can be shared during `fork()`, speeding process creation

- How?

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

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

    ▸ Without changing program behavior

    ▸ 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

- Example of a page table snapshot:

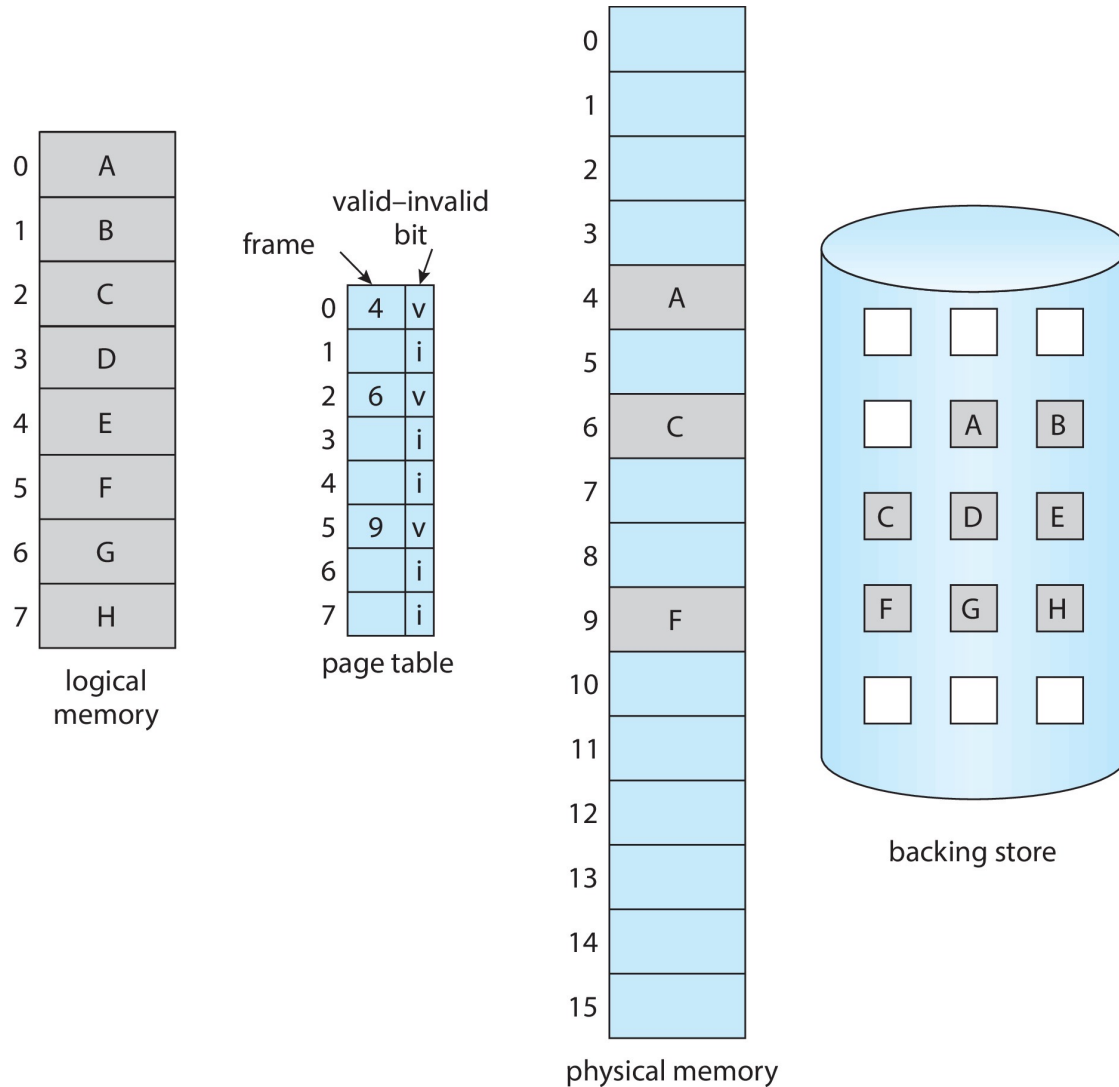| Frame #	valid-invalid bit
	v
	v
	v
	i
. . .	
	i
	i

page table

Valid-Invalid Bit

- During MMU 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

logical memory

valid–invalid bit

frame

page table

physical memory

backing store

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
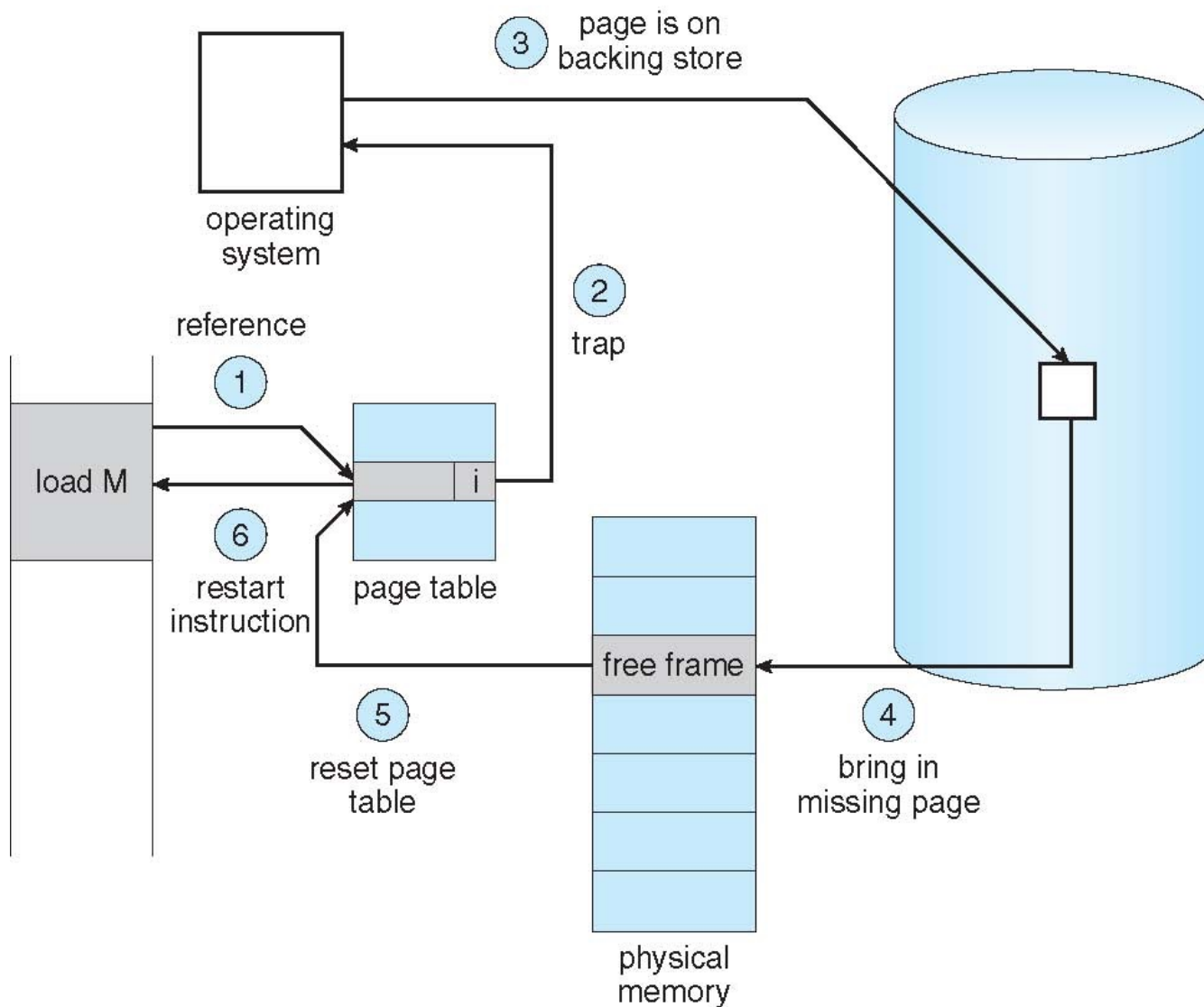
   - Just not in memory

...

Steps in Handling Page Fault

…

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

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 (cont.)

- …

- 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**

Locality of reference

- Page faults are expensive!
- **Thrashing**: Process spends most of the time paging in and out instead of executing code.
- Most programs display a pattern of behavior called the **principle of locality of reference**.

Locality of Reference: A program that references a location n at some point in time is likely to reference the same location n and locations in the immediate vicinity of n in the near future.

Source: https://people.engr.tamu.edu/bettati/Courses/410/2017A/Slides/virtmemory.pdf

Aspects of Demand Paging

- ...

- ...

- Hardware support needed for demand paging

  - Page table with valid / invalid bit

  - ...

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

Performance of Demand Paging (cont.)

- …

- 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)

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!!

Demand Paging Example (cont.)

- ....

- 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

Amirkabir University of Technology
(Tehran Polytechnic)