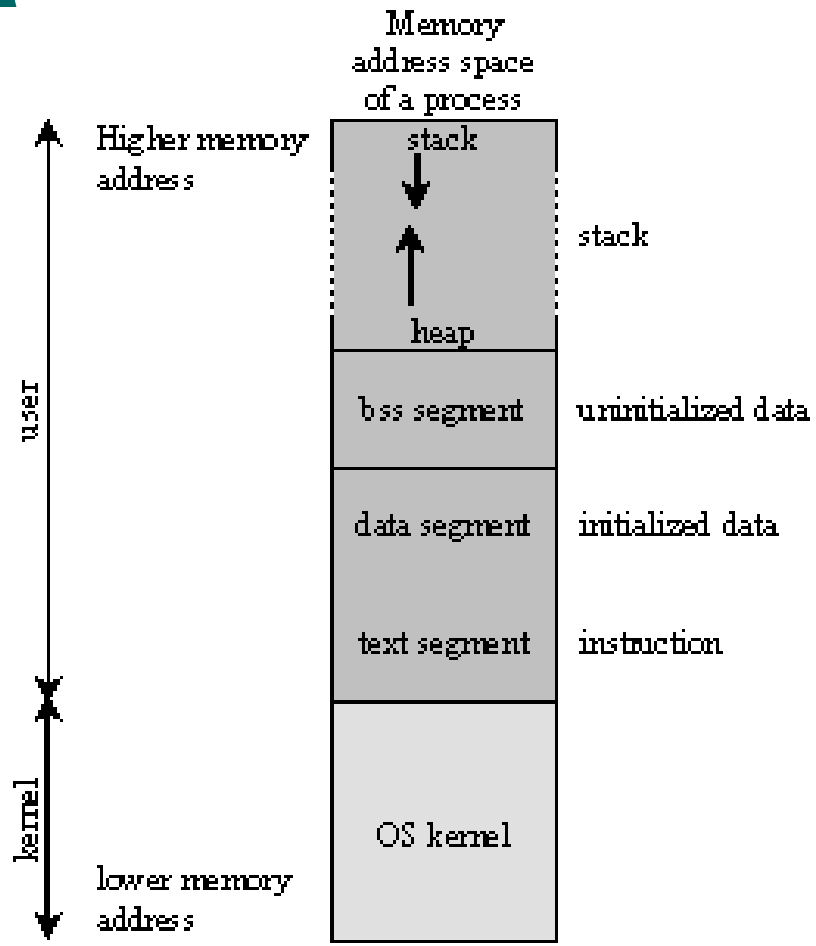
תרגיל 7

Virtual memory

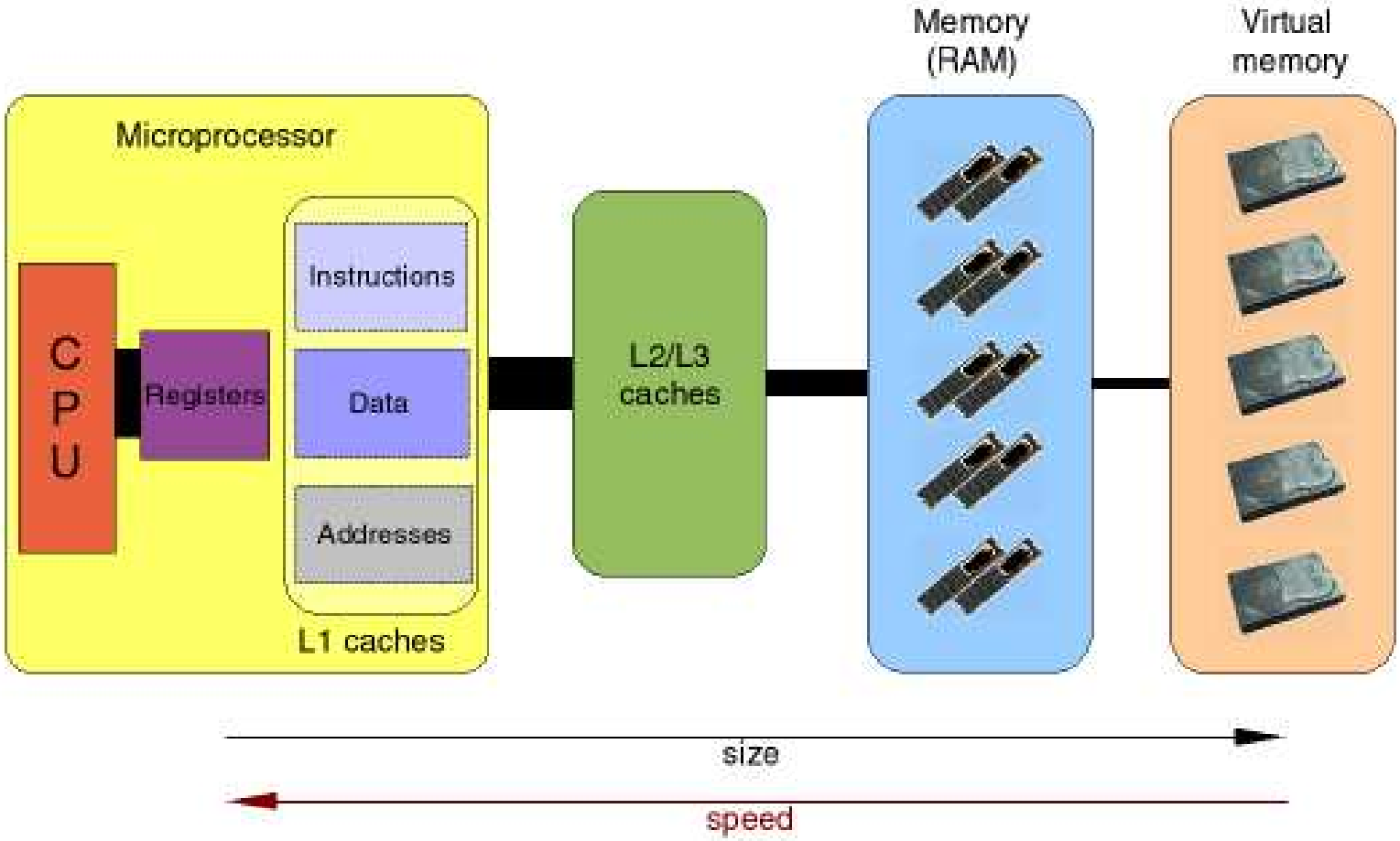
התהליך

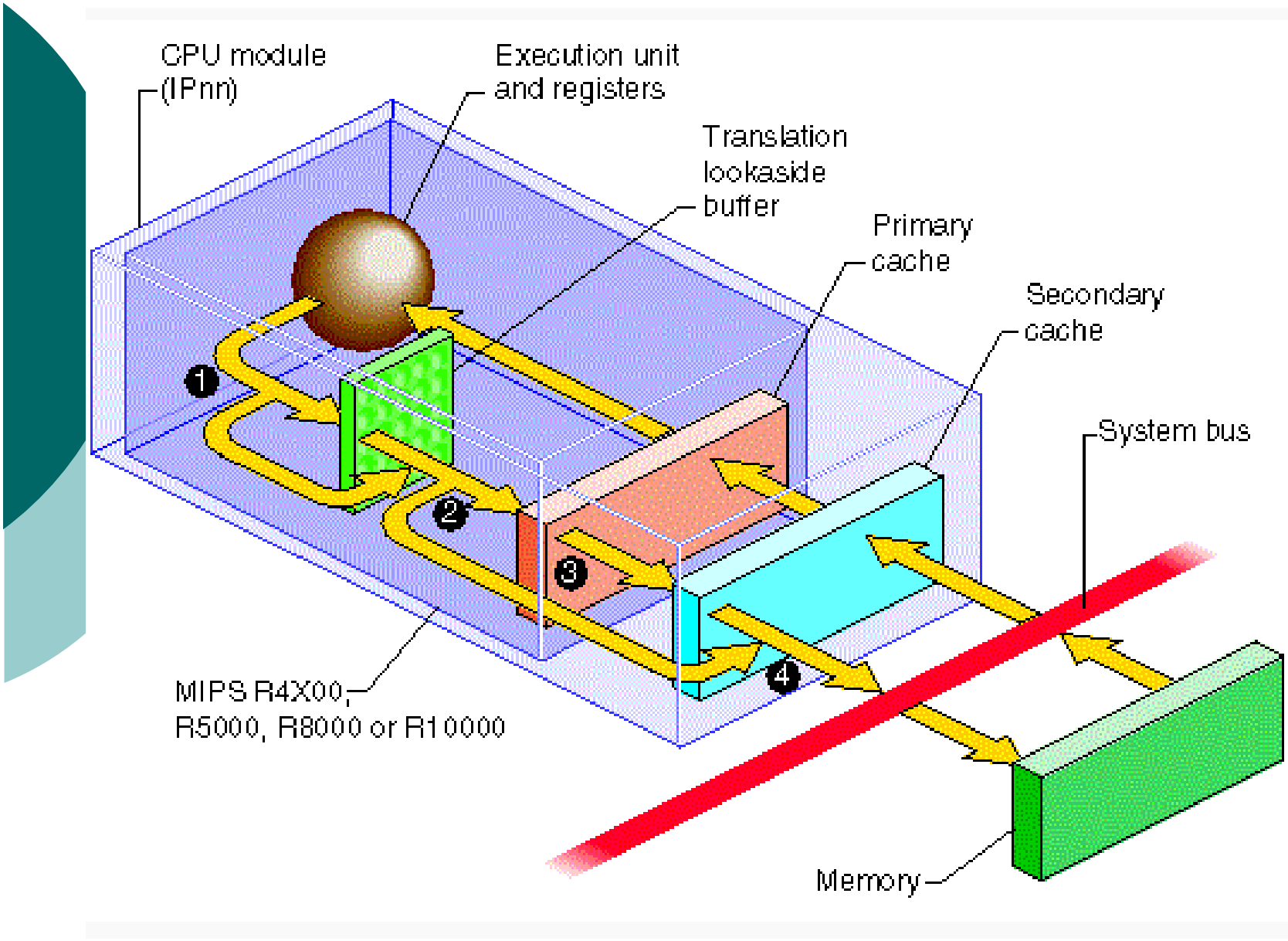


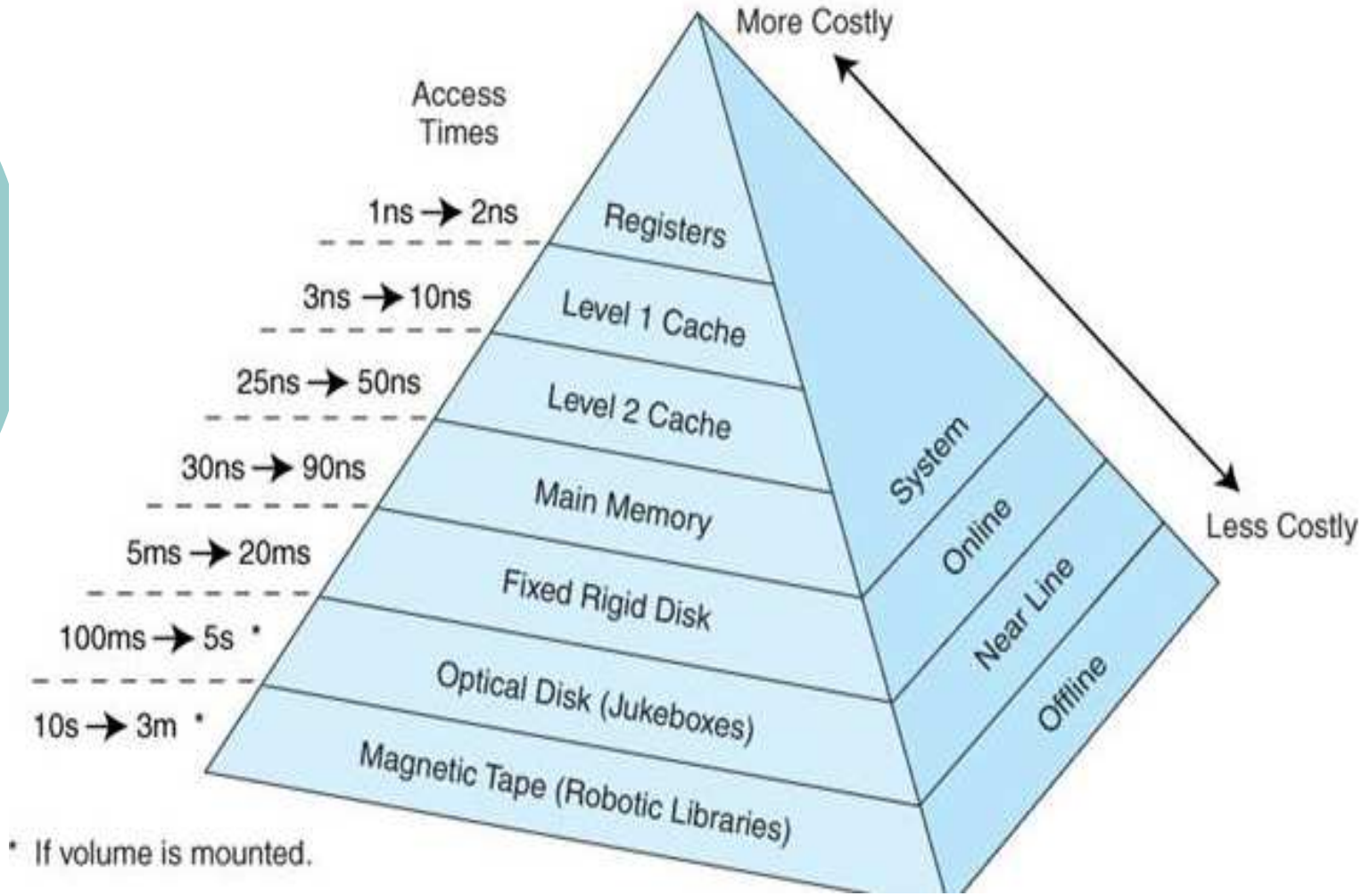
- לכל תהליך מרחב כתובות (address space) שלו – אליהן, ורק אליהן, הוא רשאי לפנות

מרחב הכתובות (address space) של התהליך

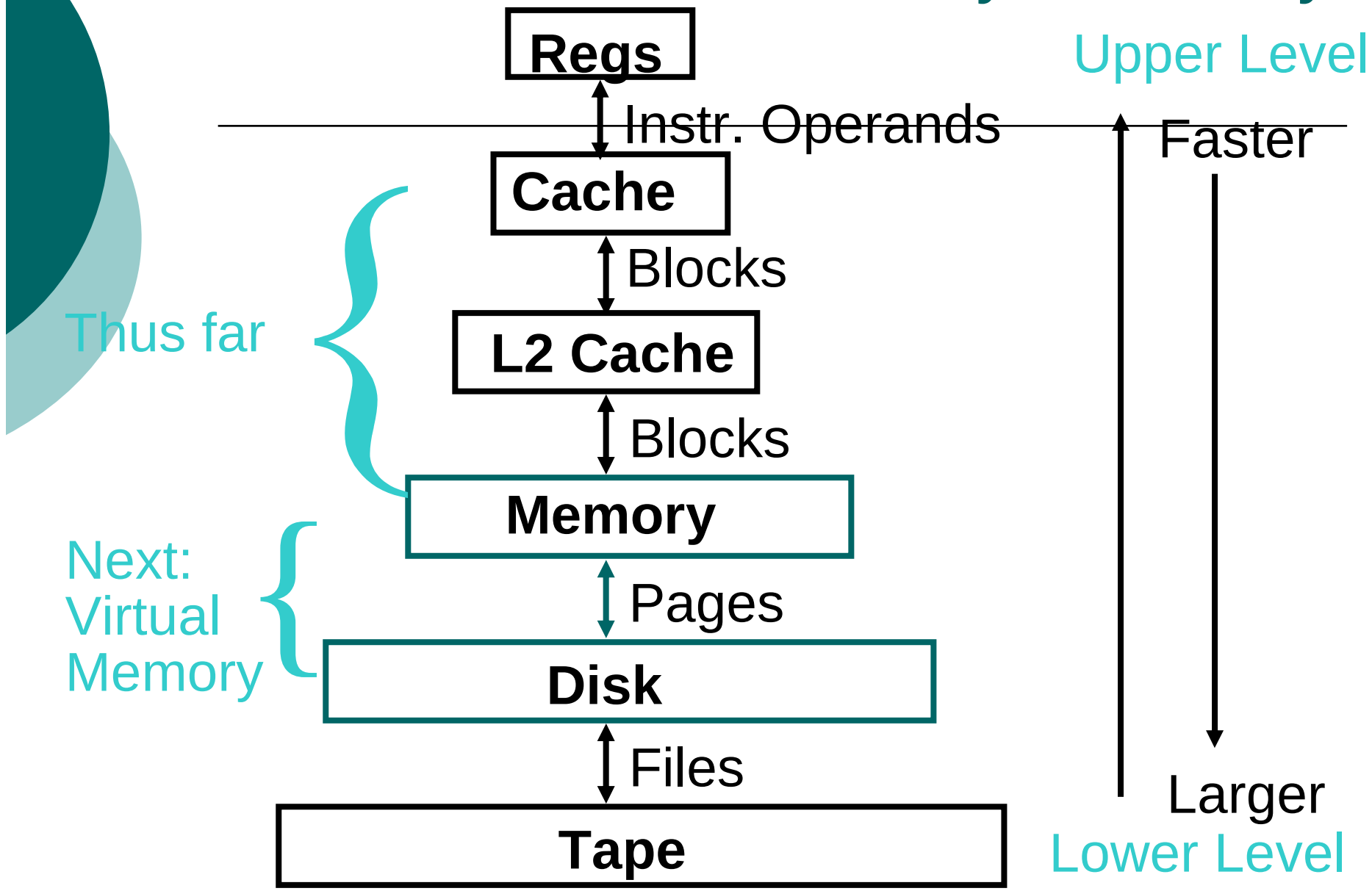
1. text segment - פקודות התכנית (בשפת מכונה) = [בד"כ מוגן מפני כתיבה]
2. Stack - מחסנית (פרמטרי התוכנית, קריאות לפונקציות ומשתנים לוקליים)
3. heap ערמה (להקצאה דינאמית)
4. data section – משתנים סטטיים וגלובליים
5. bss (block started by symbol) משתנים גלובליים וסטטיים שאינם מאותחלים (וע"כ אינם נשמרים בקובץ המכיל את התכנית)
6. מידע נוסף על התהליך (רשימת קבצים שפתח, עדיפות, זמן ריצה, מצב האוגרים ב-context switch האחרון, ..) - [מתוחזק על ידי מערכת ההפעלה]


Memory
(RAM)
Virtual
memory
Microprocessor
CPU
Registers
Instructions
Data
Addresses
L1 caches
L2/L3
caches
size
speed


CPU module
(IPnn)
Execution unit
and registers
Translation
lookaside
buffer
Primary
cache
Secondary
cache
System bus
1
2
3
4
MIPS R4X00,
R5000, R8000 or R10000
Memory


More Costly
Access Times
1ns → 2ns
3ns → 10ns
25ns → 50ns
30ns → 90ns
5ms → 20ms
100ms → 5s *
10s → 3m *
Registers
Level 1 Cache
Level 2 Cache
Main Memory
Fixed Rigid Disk
Optical Disk (Jukeboxes)
Magnetic Tape (Robotic Libraries)
System
Online
Near Line
Offline
Less Costly
* If volume is mounted.

Another View of the Memory Hierarchy

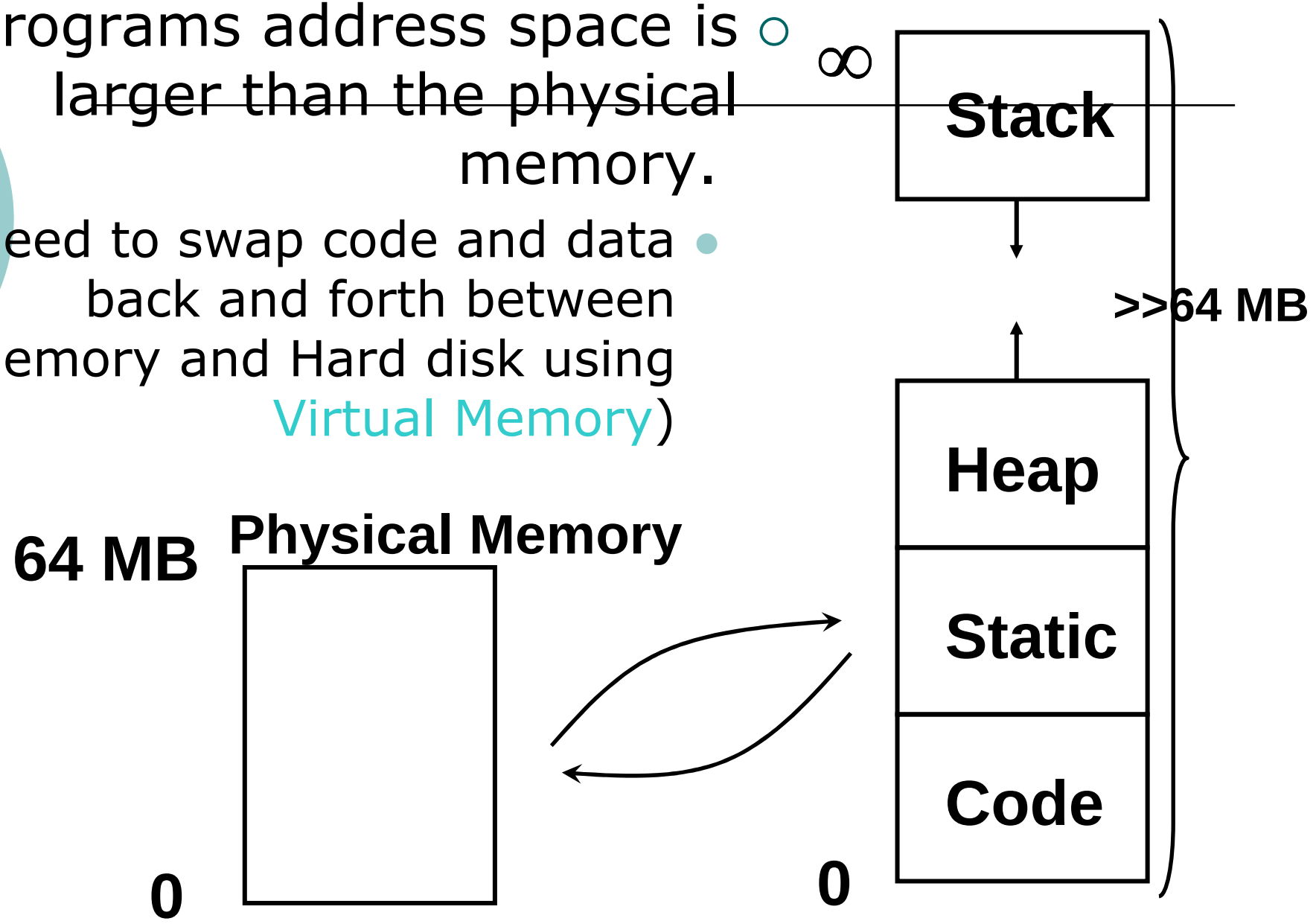


Virtual Memory

- If Principle of Locality allows caches to offer (usually) speed of cache memory with size of DRAM memory,
  then why not, recursively, use at next level to give speed of DRAM memory, size of Disk memory?
- Called "Virtual Memory"
  - Also allows OS to share memory, protect programs from each other
  - Today, more important for protection vs. just another level of memory hierarchy
  - Historically, it predates caches

(#1/2)

- Programs address space is larger than the physical memory.
  - Need to swap code and data back and forth between memory and Hard disk using Virtual Memory)



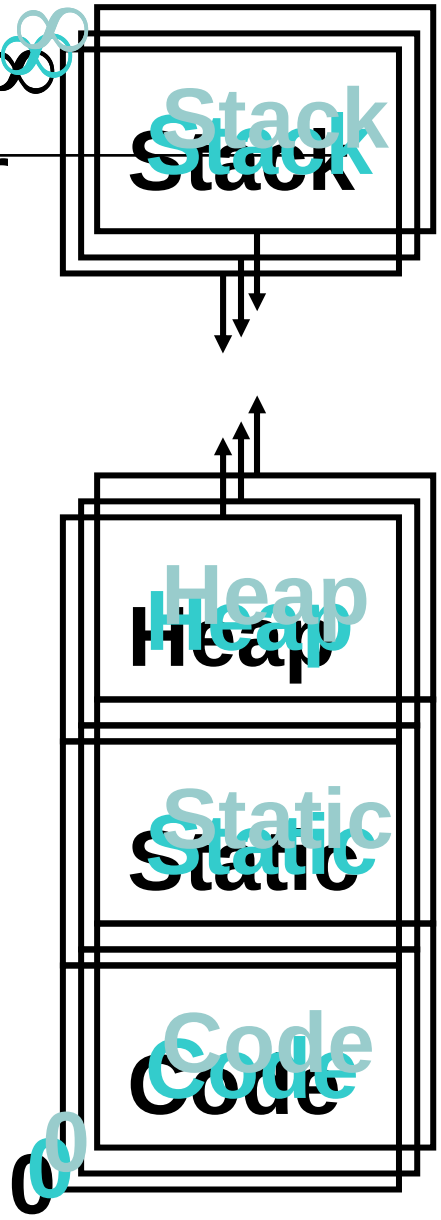
(#2/2)

Many Processes (programs) active at the same time. (Single Processor - many Processes)

- Processor appears to run multiple programs all at once by rapidly switching between active programs.
- The rapid switching is managed by Memory Management Unit (MMU) by using Virtual Memory concept.

•Each program sees the entire address space as its own.

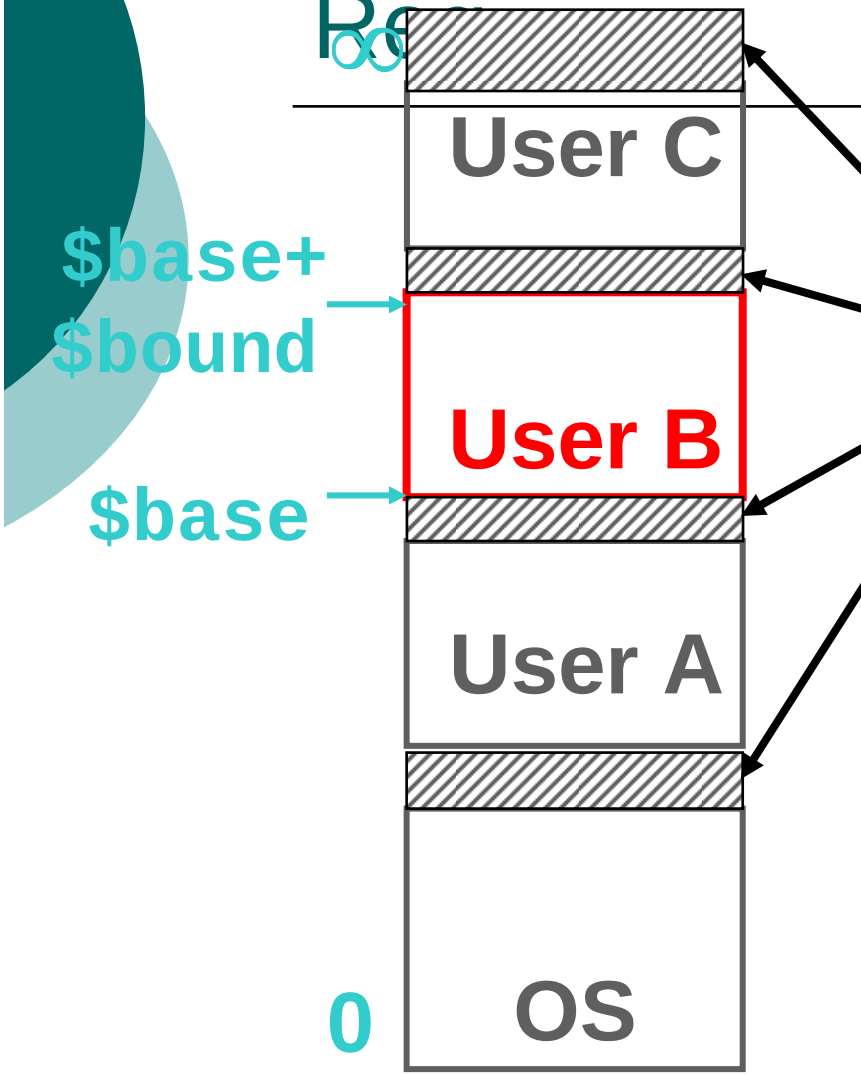
•How to avoid multiple programs overwriting each other.



Segmentation Solution

- Segmentation provides simple MMU
  - Program views its memory as set of segments. Code segment, Data Segment, Stack segment, etc.
  - Each program has its own set of private segments.
  - Each access to memory is via a segment selector and offset within the segment.
  - It allows a program to have its own private view of memory and to coexist transparently with other programs in the same memory space.

Simple Example: Base and Bound Re...

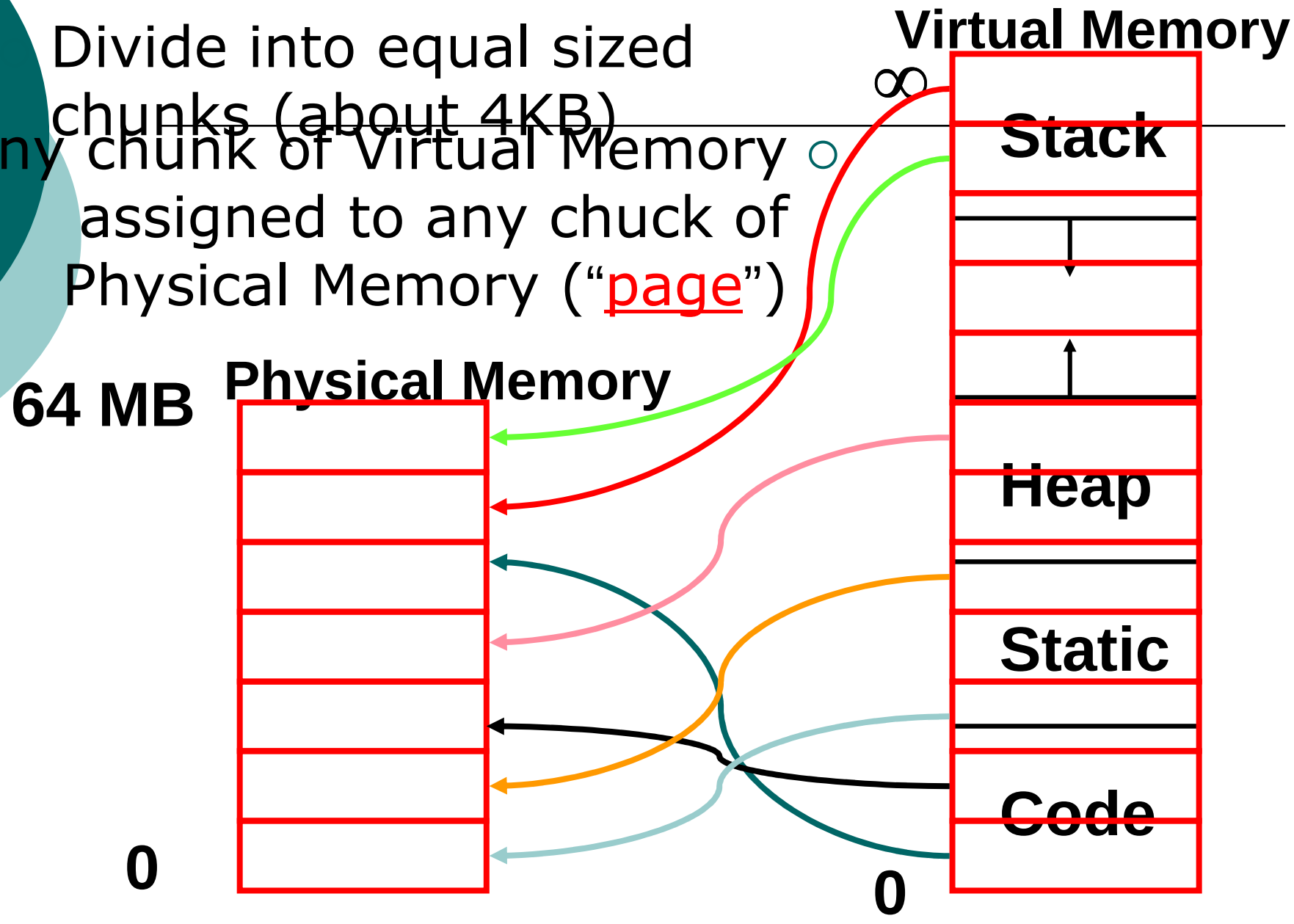


Enough space for User D, but discontinuous ("fragmentation problem")

- Want discontinuous mapping
- Process size >> mem
- Addition not enough!

Mapping Virtual Memory to Physical Memory

- Divide into equal sized chunks (about 4KB)
- Any chunk of Virtual Memory assigned to any chuck of Physical Memory ("page")



Virtual Memory Mapping Function

- Cannot have simple function to predict arbitrary mapping
- Use table lookup of mappings

Page Number	Offset

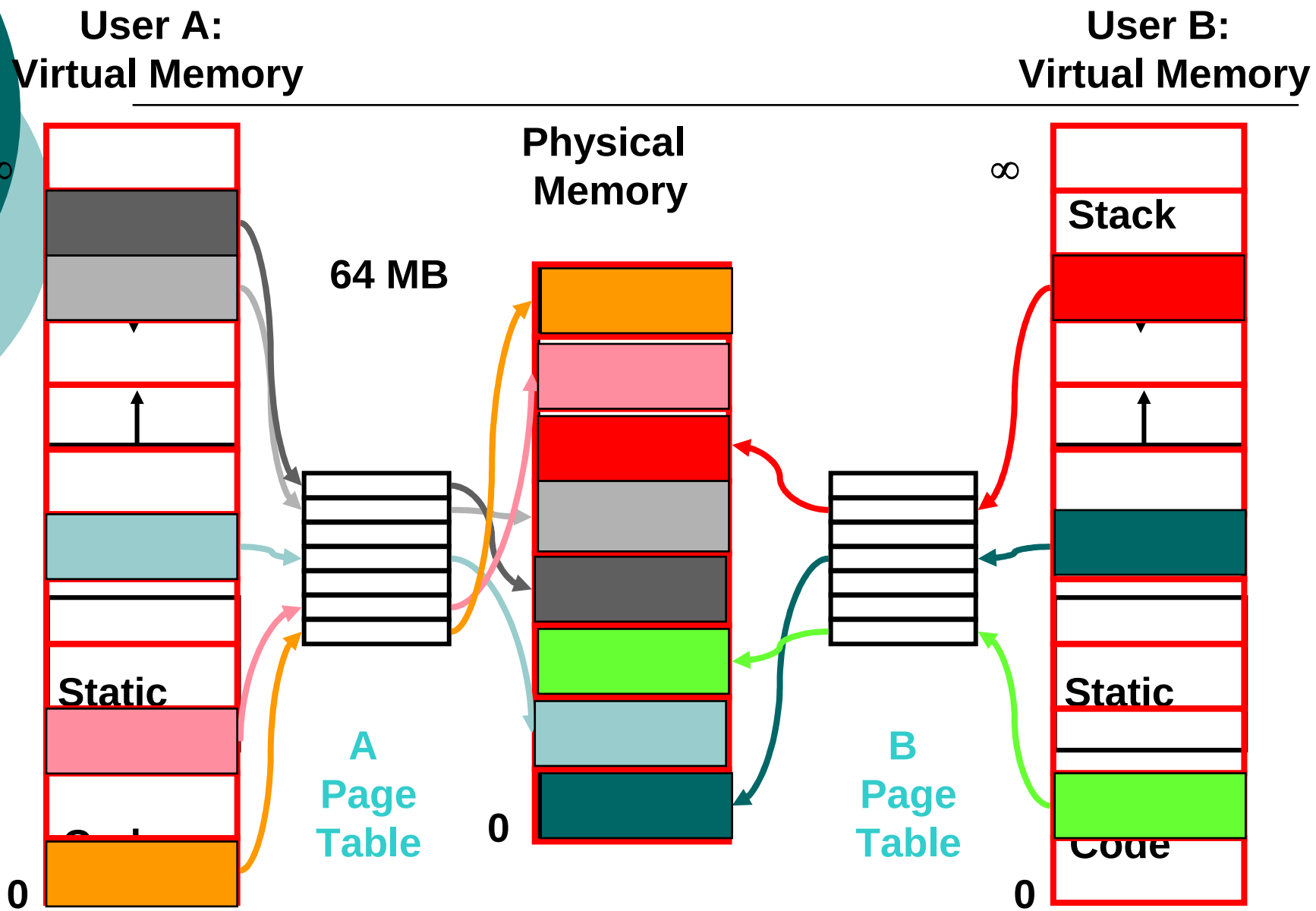
- Use table lookup ("Page Table") for mappings: Page number is index
- Virtual Memory Mapping Function
  - Physical Offset = Virtual Offset
  - Physical Page Number = PageTable[Virtual Page Number]

(P.P.N. also called "Page Frame")

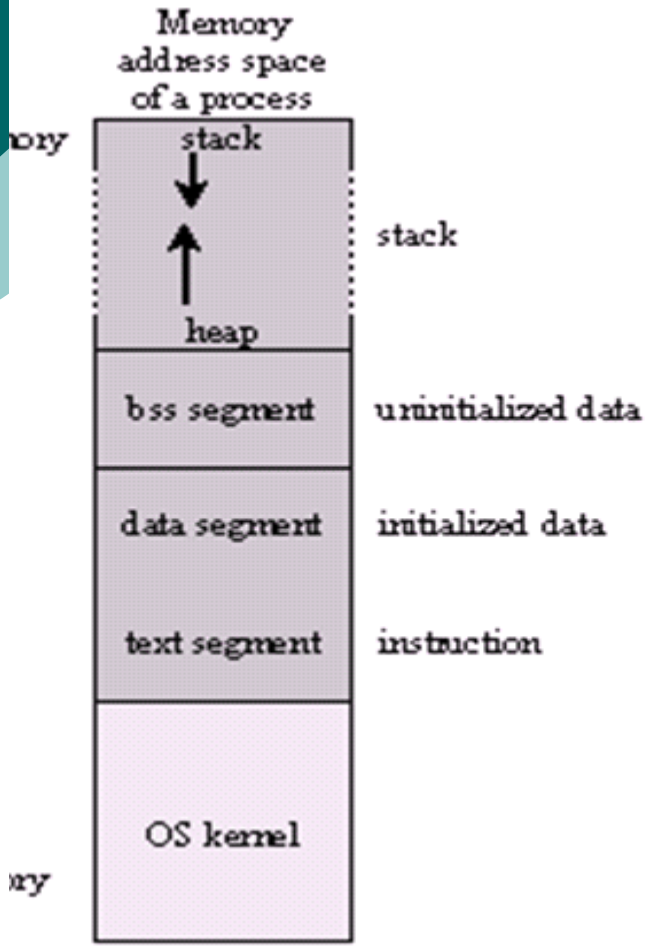
Page Table

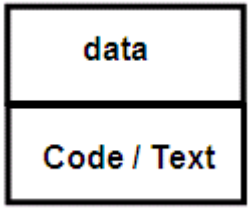
- A page table is an operating system structure which contains the mapping of virtual addresses to physical locations
  - There are several different ways, all up to the operating system, to keep this data around
- Each process running in the operating system has its own page table
  - "State" of process is PC, all registers, plus page table
  - OS changes page tables by changing contents of Page Table Base Register

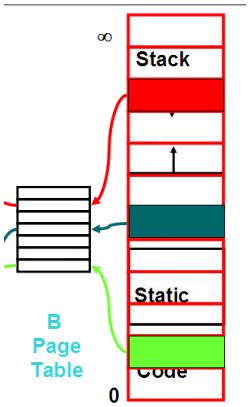
Paging/Virtual Memory for Multiple Pocesses

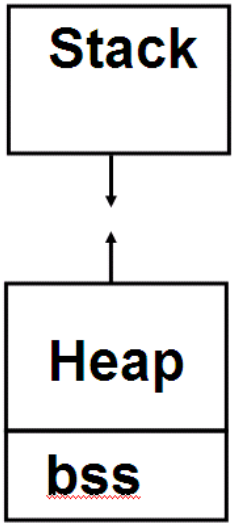


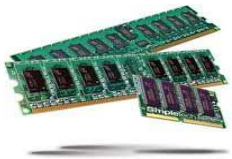
Pages, Who? Where ?

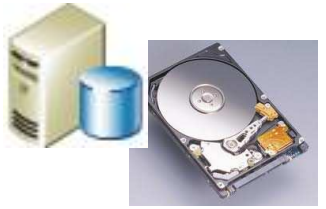








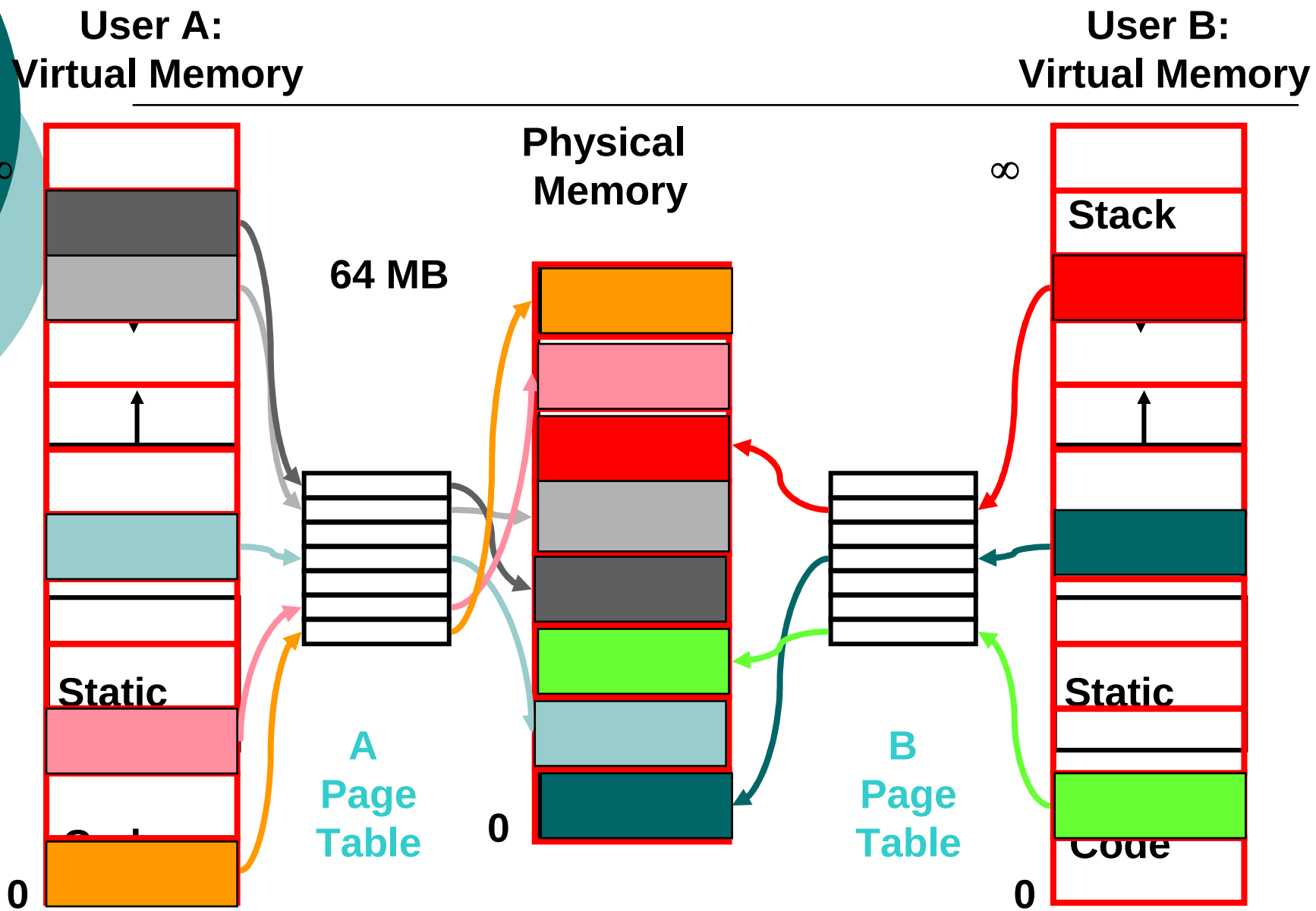




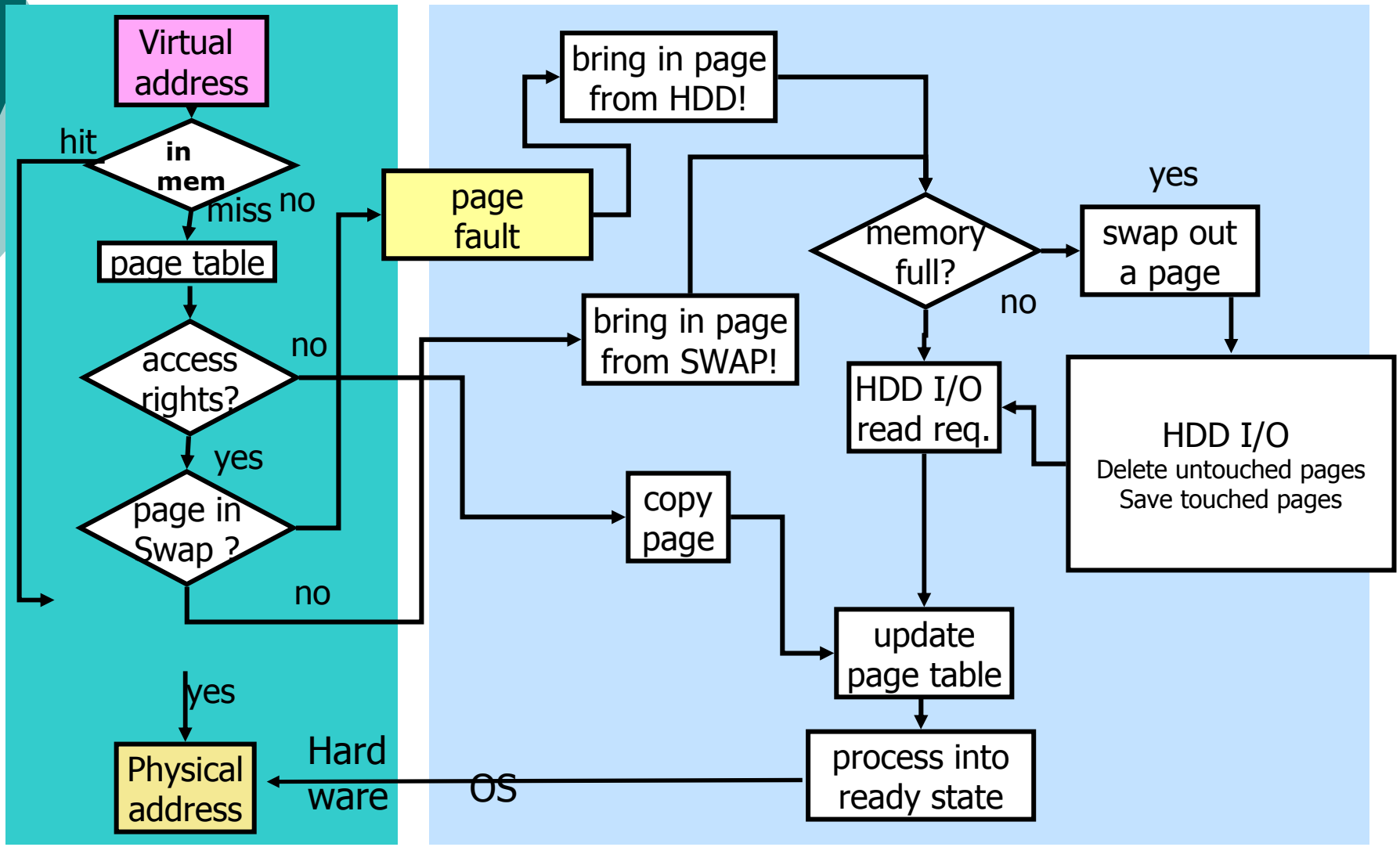
Main

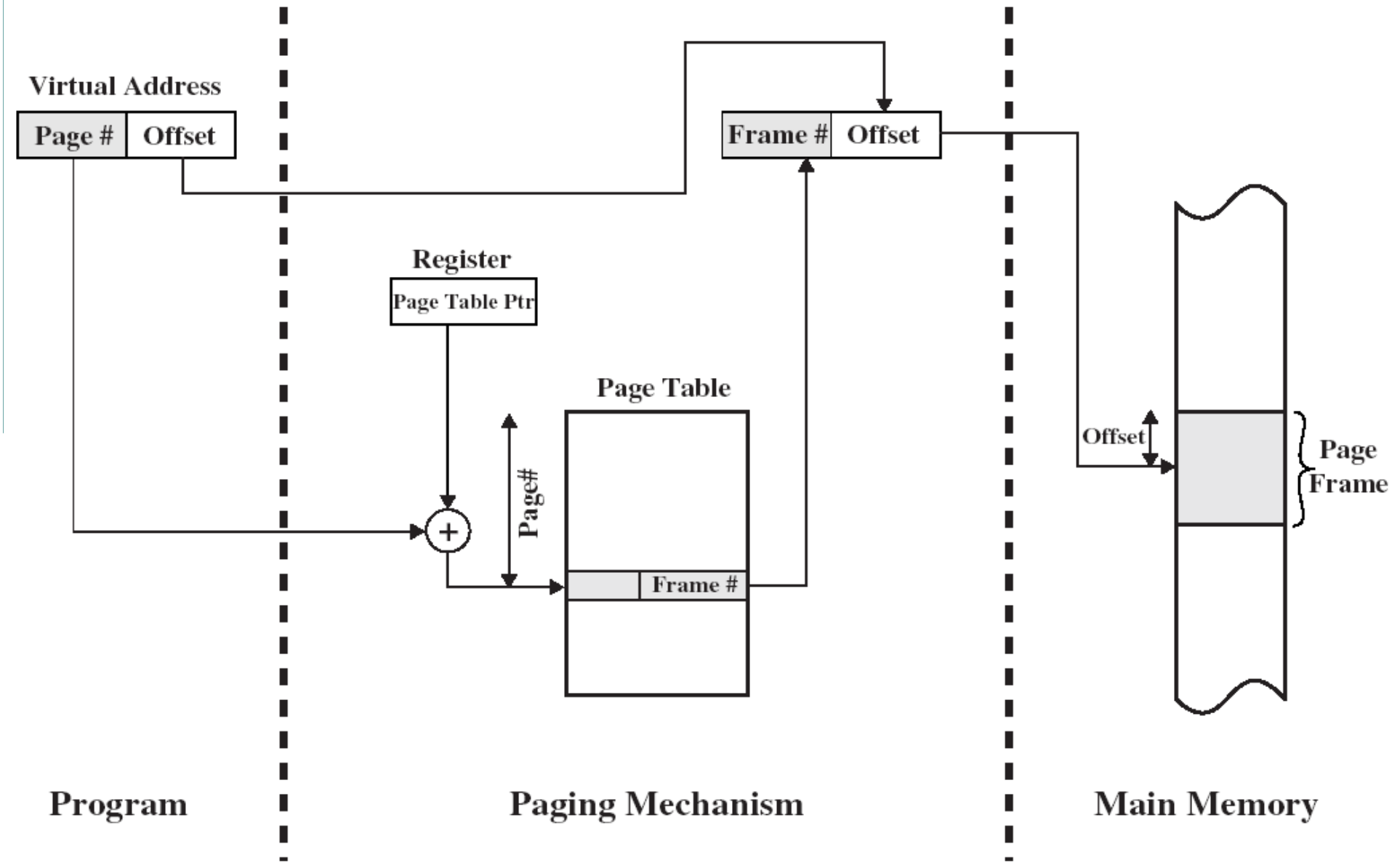
```
#define TEXT_SIZE 256        //size of text segment of executable
#define DATA_SIZE 256        //size of data segment of executable
#define BSS_SIZE 256         //size of data segment of executable
#define LETTER_START 65      //first capital letter in ascii table
#define DIFF 25              //range of capital letters in ascii table
#define EXEC_FILE "exec"
#define LOOPS 200

int main() {
    srand(time(NULL));
    sim_database* db = vm_constructor(EXEC_FILE, TEXT_SIZE, DATA_SIZE, BSS_SIZE);
    int i; unsigned short addr;      //virtual address

    for (i = 0; i < LOOPS; i++) {
        addr = rand() % EXEC_SIZE;
        vm_load(db, addr, val);
        char* val = rand() % DIFF + LETTER_START;        //capital letters
        vm_store(db, addr+1, val);
    }
    vm_print(db);
    vm_destructor(db);
}
```

Paging/Virtual Memory for Multiple Pocesses



תרשים זרימה תרגיל 7




Virtual Address
Page #
Offset
Register
Page Table Ptr
Page Table
Page#
Frame #
Frame #
Offset
Offset
Page
Frame
Program
Paging Mechanism
Main Memory