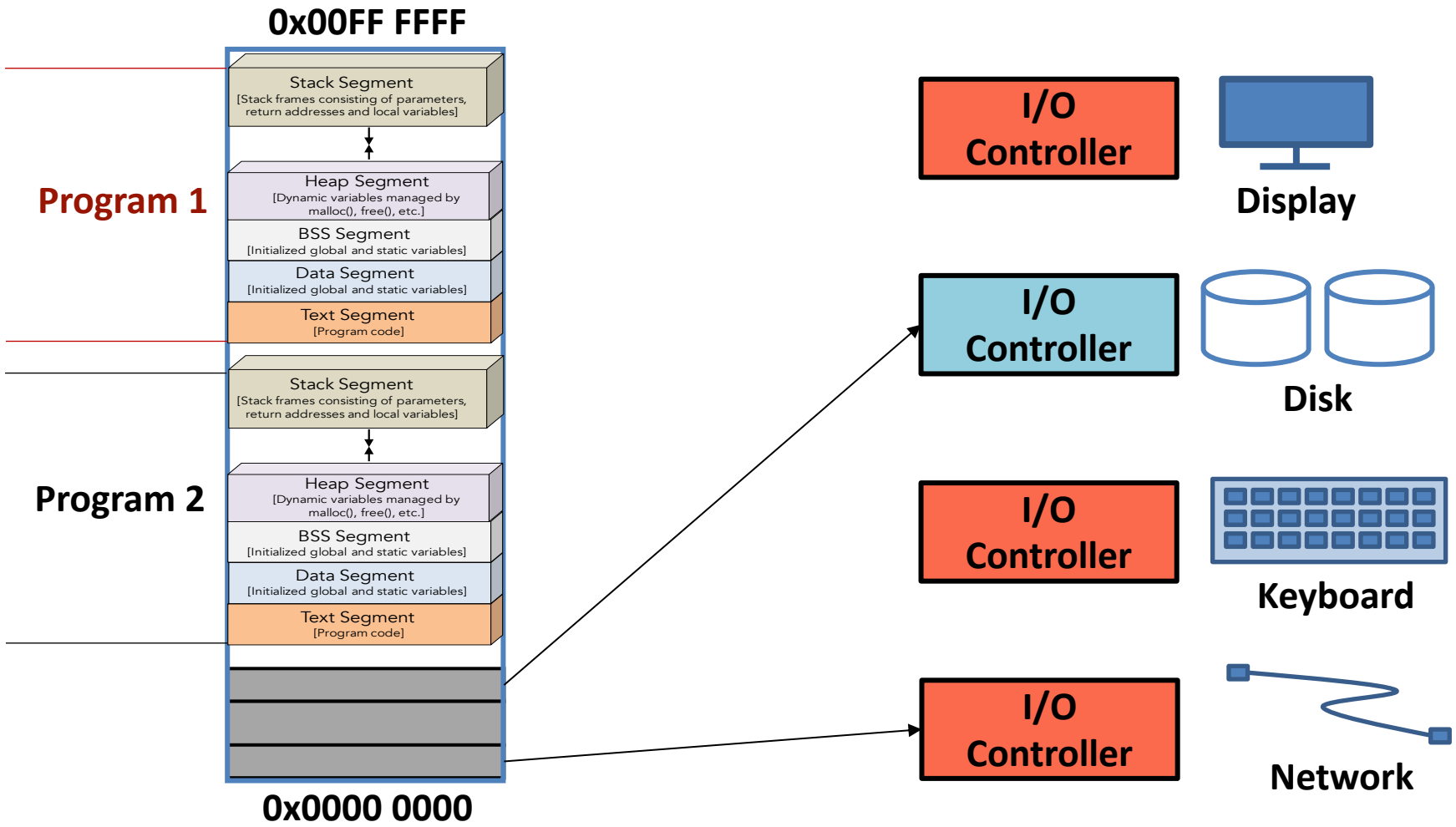
# CSE 520
# Computer Architecture II

## Advanced Memory Operations

Prof. Michel A. Kinsy

# Memory – Programs - I/O



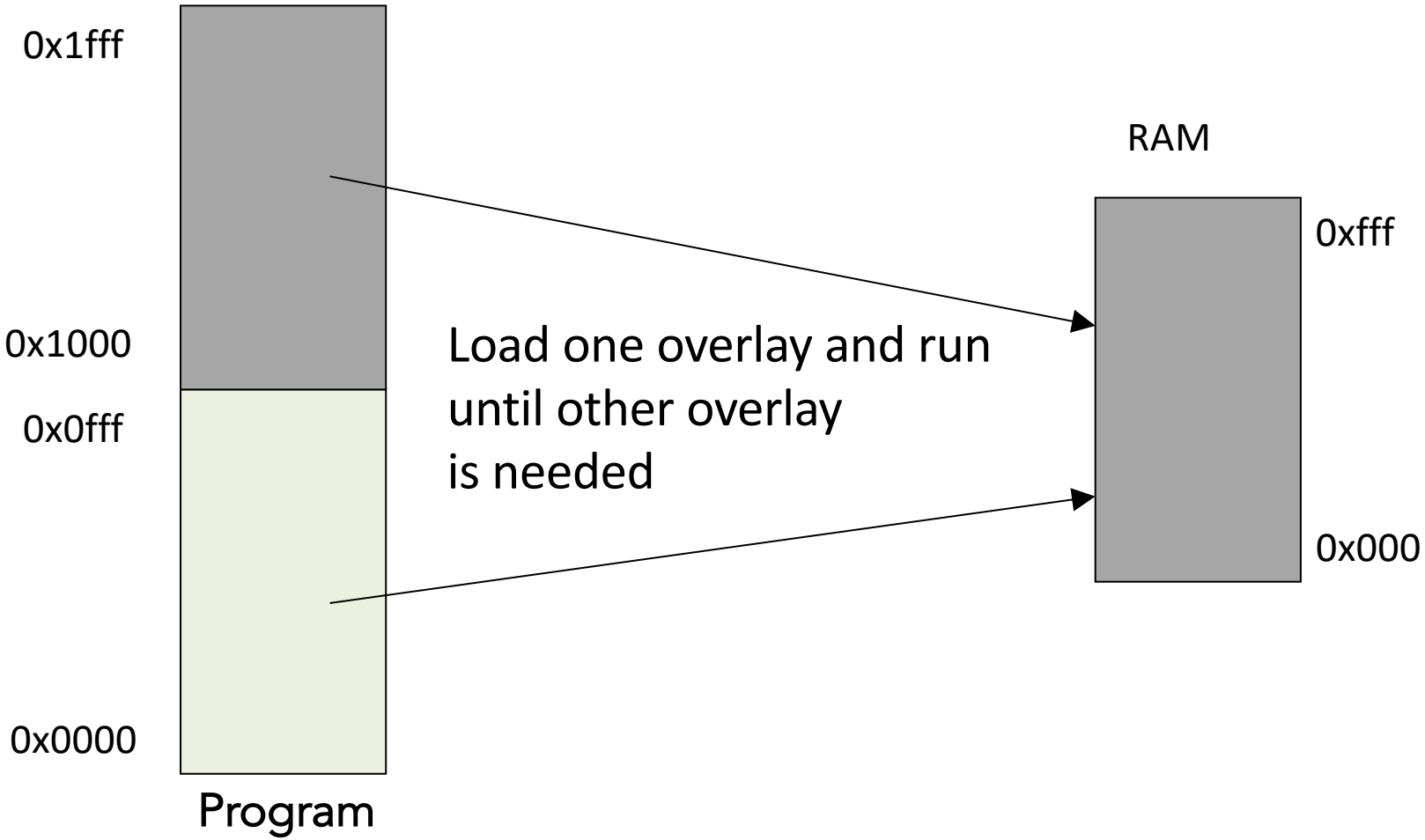
# Running a Program

- Operating System (loader) copies a program from permanent storage into RAM
  - Note: The OS is just another program
- For PCs and workstations, the OS copies the program (bits) from disk
- The CPU's Program Counter is then set to the starting address of the program and the program begins execution

# Early Problems -Sixties

- There were many applications whose data could not fit in the main memory,
  - e.g., Payroll
- Paged memory system reduced fragmentation but still required the whole program to be resident in the main memory
- Programmers moved the data back and forth from the secondary store by overlaying it repeatedly on the primary store

# Use of Overlays



# Use of Overlays

- Programmer divides the code into pieces that fit into RAM
- Pieces, called overlays, are loaded and unloaded by the program
- Does not require OS help
- Problems with overlays
  - Difficult for programmer to manage
  - Manual Overlays
    - Assuming an instruction can address all the storage on the drum
      - Approach 1 - programmer keeps track of addresses in the main memory and initiates an I/O transfer when required
      - Approach 2 - automatic initiation of I/O transfers by software address translation Brooker's interpretive coding, 1960

# Memory Management

- The Fifties:
  - Absolute Addresses
  - Dynamic address translation
- The Sixties:
  - Paged memory systems and TLBs
  - Atlas' Demand paging

- Modern Virtual Memory Systems

# General Virtual Memory

- Virtual memory
  - Technique that allows execution of a program that may not completely reside in memory (RAM)
- Importance of virtual memory
  - Allows available (fast and expensive) physical memory to be very well utilized
  - Simplifies memory management (main reason today)
  - Removes burden of memory resource management from the programmer

# Virtual Memory

- Two memory "spaces"
  - Virtual memory space  what the program "sees"
  - Physical memory space  what the program runs in (size of RAM)
- On program startup
  - OS copies program into RAM
  - If there is not enough RAM, OS stops copying program and starts it running with only a portion of the program loaded in RAM

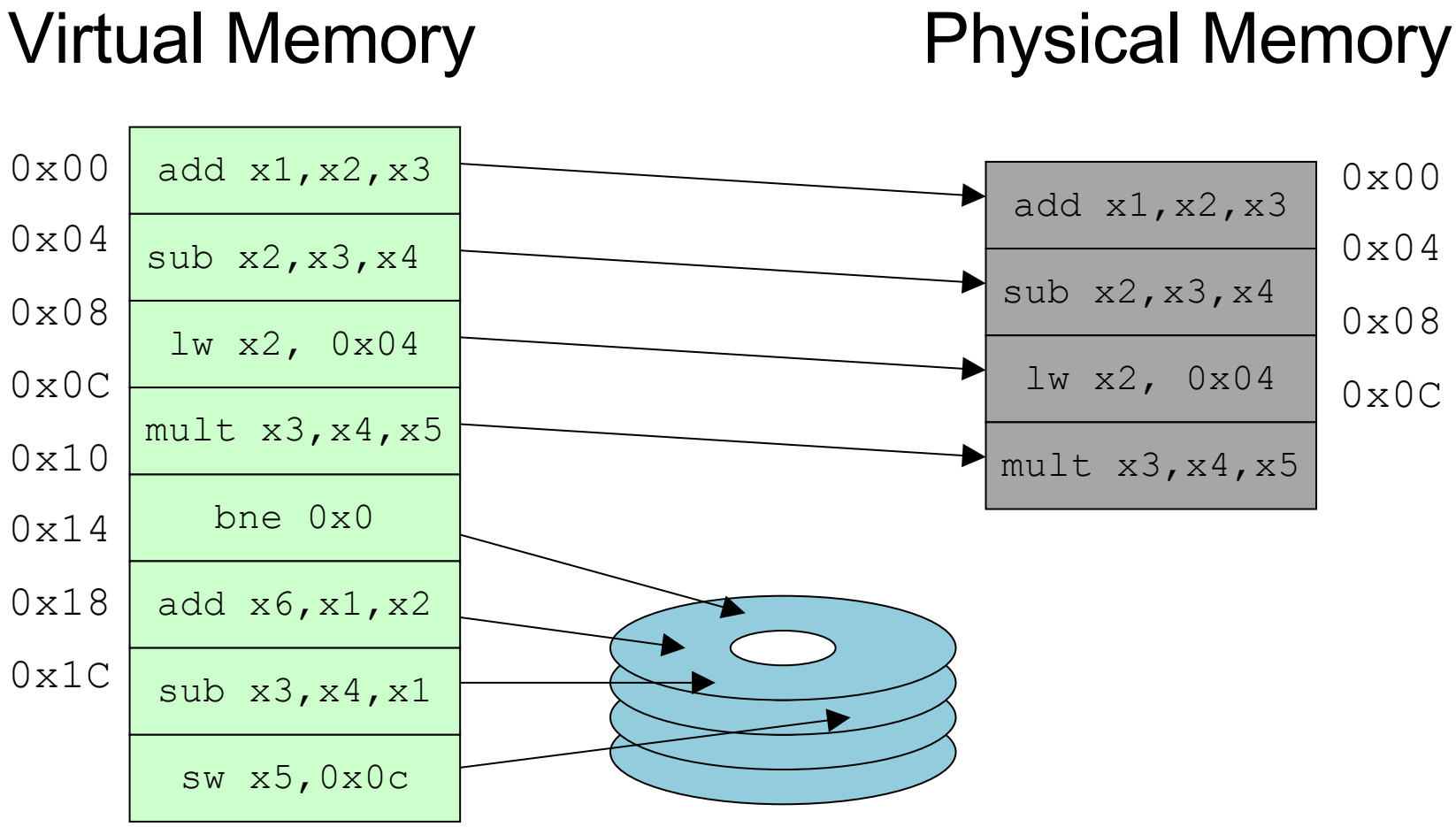
# Virtual Memory

- On program startup
  - OS copies program into RAM
  - If there is not enough RAM, OS stops copying program and starts it running with only a portion of the program loaded in RAM
  - When the program touches a part of the program not in physical memory (RAM), OS catches the memory abort (called a page fault) and copies that part of the program from disk into RAM

# Virtual Memory

- On program startup
  - OS copies program into RAM
  - If there is not enough RAM, OS stops copying program and starts it running with only a portion of the program loaded in RAM
  - When the program touches a part of the program not in physical memory (RAM), OS catches the memory abort (called a page fault) and copies that part of the program from disk into RAM
  - In order to copy some of the program from disk to RAM, OS must evict parts of the program already in RAM
    - OS copies the evicted parts of the program back to disk

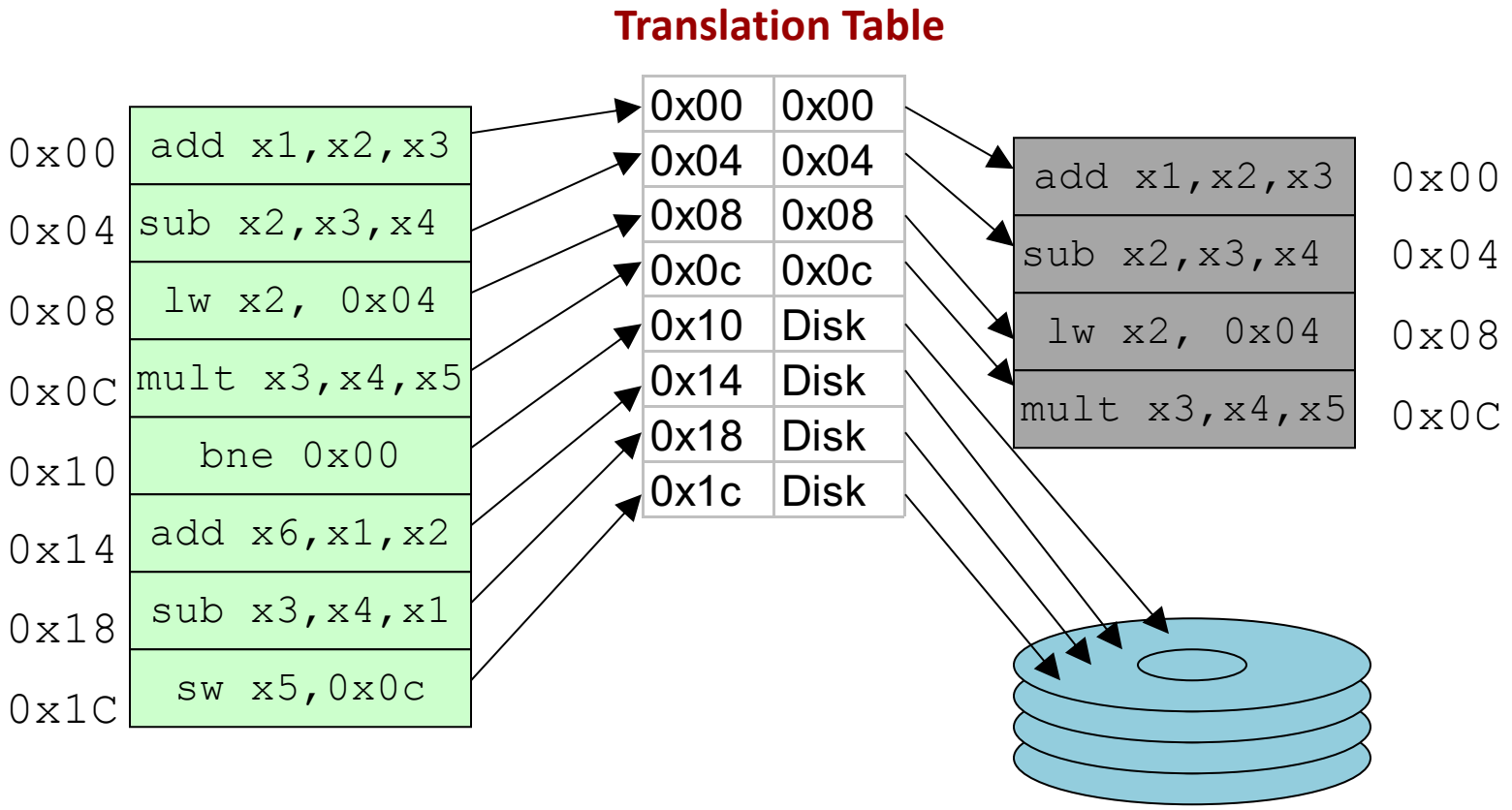
# Virtual Memory



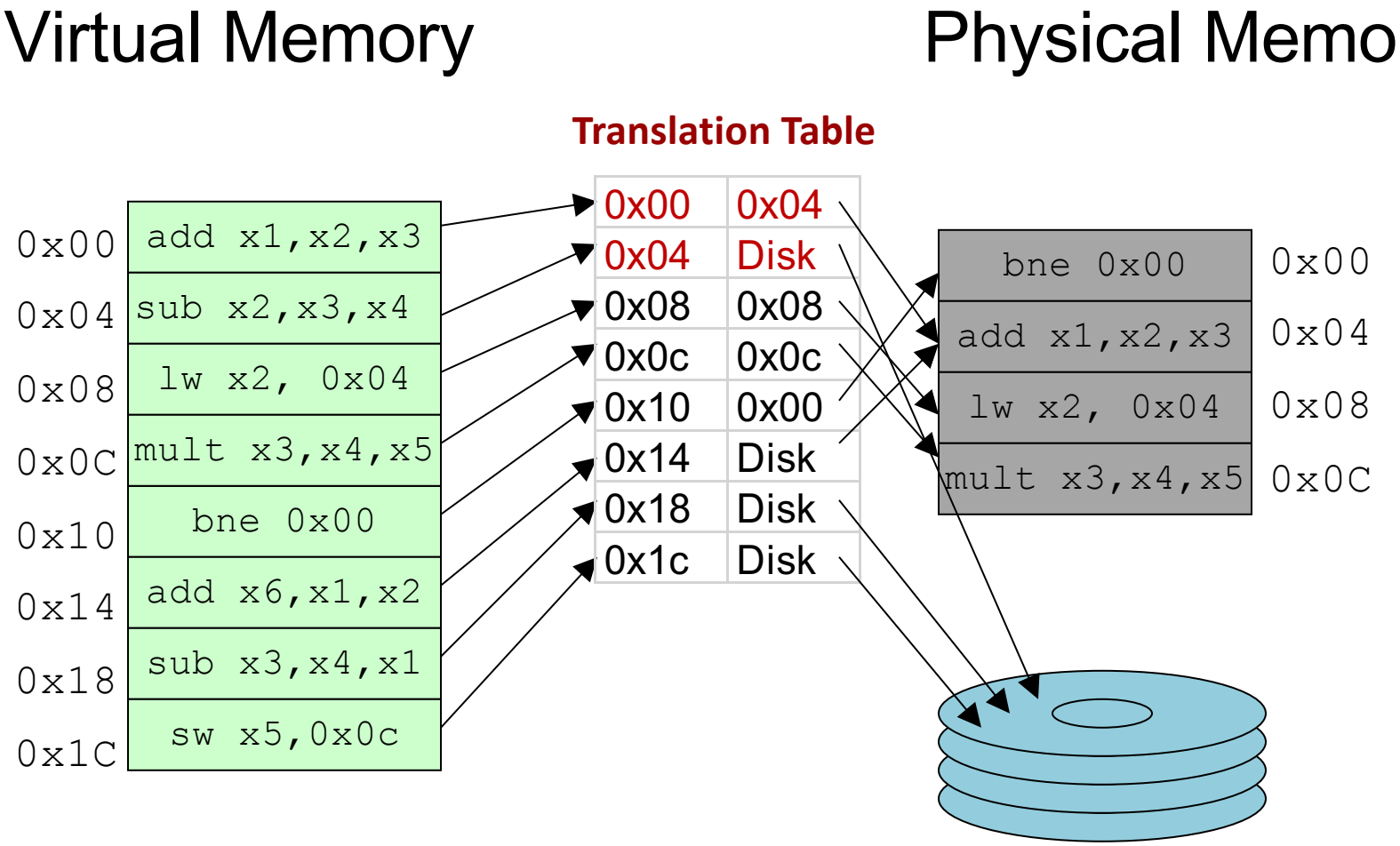
# Virtual Memory

## Virtual Memory

## Physical Memory



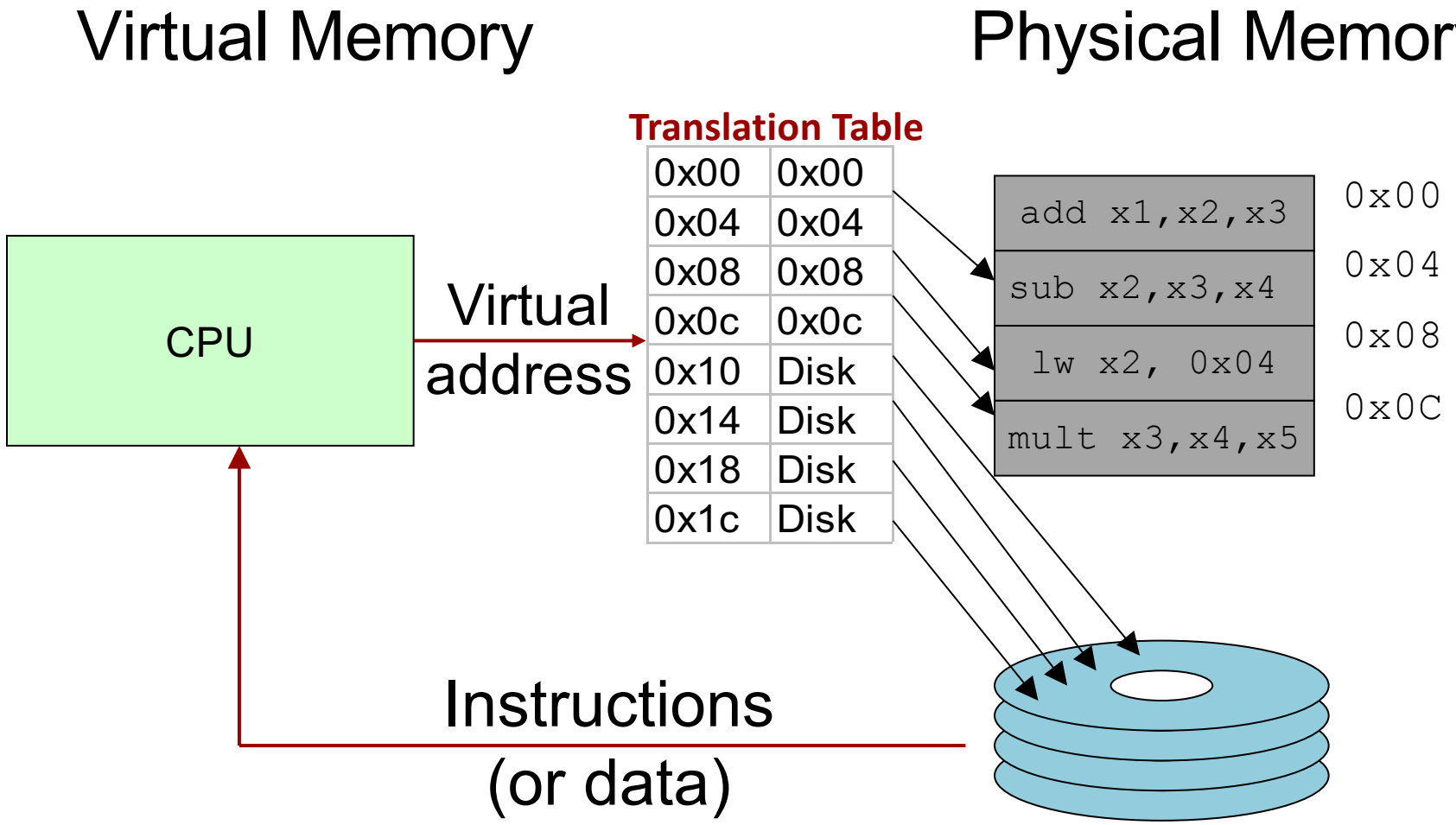
# Page Fault


Virtual Memory
Physical Memory
Translation Table
0x00 add x1,x2,x3
0x04 sub x2,x3,x4
0x08 lw x2, 0x04
0x0C mult x3,x4,x5
0x10 bne 0x00
0x14 add x6,x1,x2
0x18 sub x3,x4,x1
0x1C sw x5,0x0c
0x00 0x04
0x04 Disk
0x08 0x08
0x0c 0x0c
0x10 0x00
0x14 Disk
0x18 Disk
0x1c Disk
bne 0x00 0x00
add x1,x2,x3 0x04
lw x2, 0x04 0x08
mult x3,x4,x5 0x0C

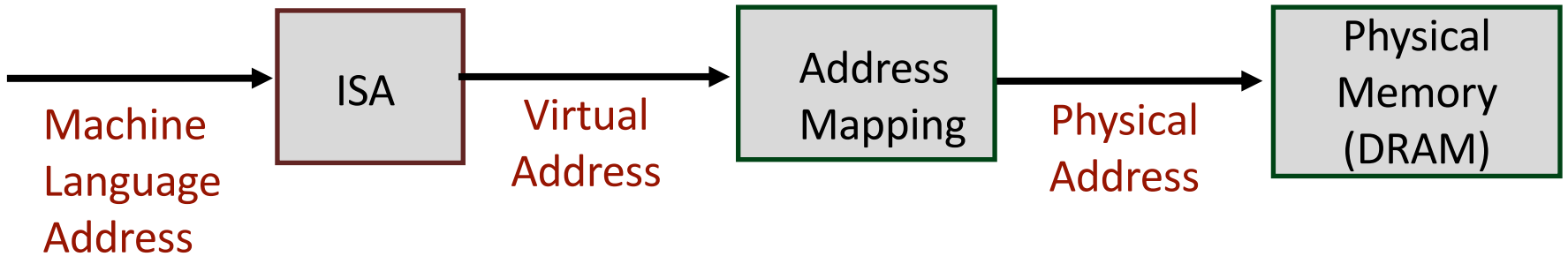
# Program View

- Program asks for virtual address
- Computer translates virtual address (VA) to physical address (PA)
- Computer reads PA from RAM and return the content to the program

# Program View



# Names for Memory Locations



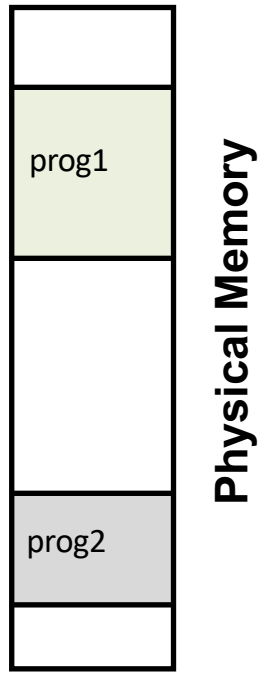

- Machine language address
  - As specified in machine code
- Virtual address
  - ISA specifies translation of machine code address into virtual address of program variable
- Physical address
  - Operating system specifies mapping of virtual address into name for a physical memory location

# Absolute Addresses

- EDSAC, early 50's
  - Effective address = physical memory address
- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
- Addresses in a program depended upon where the program was to be loaded in memory
- But it was more convenient for programmers to write location-independent subroutines
  - How could location independence be achieved?

# Dynamic Address Translation

- Motivation
  - In the early machines, I/O operations were slow and each word transferred involved the CPU
  - Higher throughput if CPU and I/O of 2 or more programs were overlapped. Why?
    - Multiprogramming
- Location independent programs
  - Programming and storage management ease
  - Need for a base register

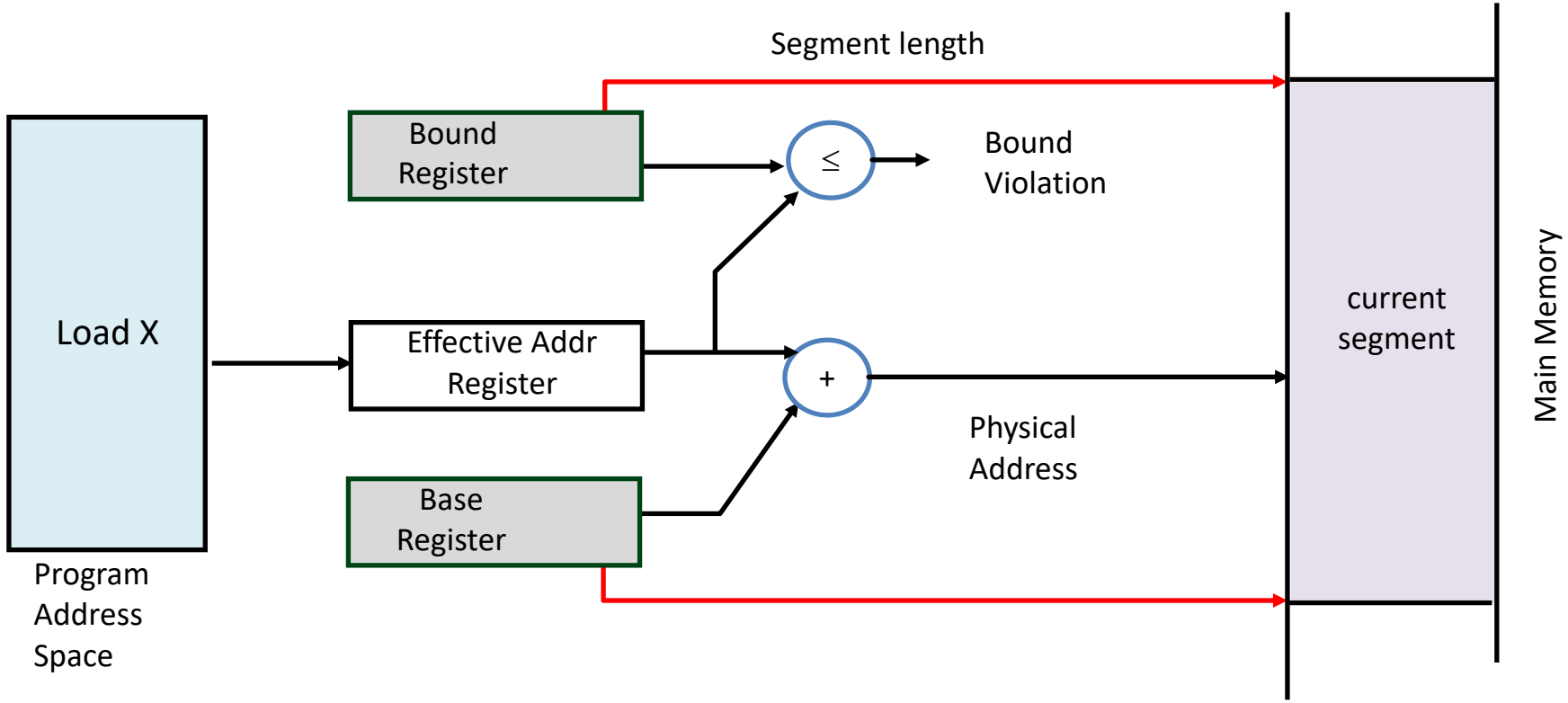


# Dynamic Address Translation

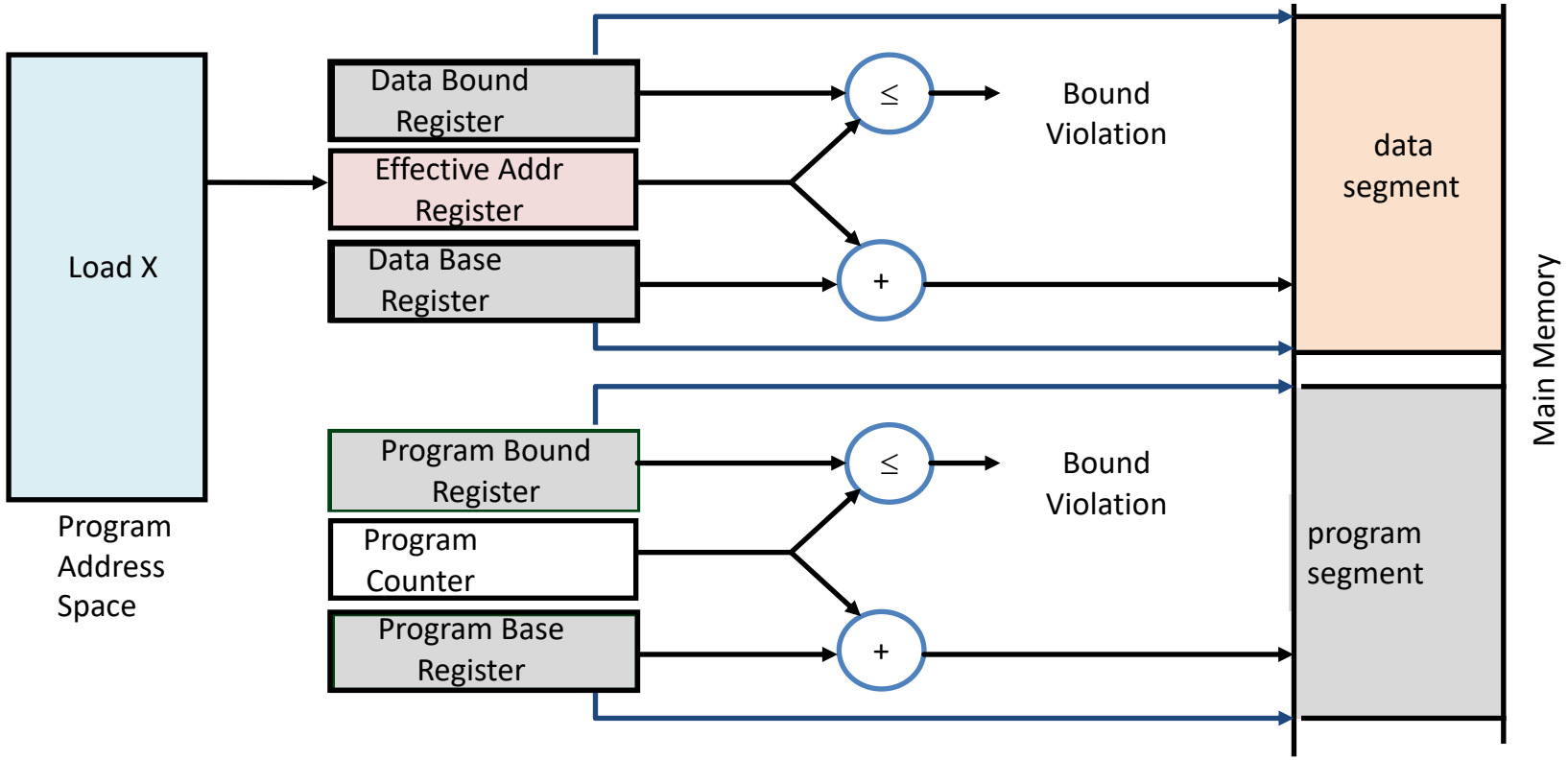
- Protection:
  - Independent programs should not affect each other inadvertently
    - Need for a bound register

# Simple Base and Bound Translation

- Base and bounds registers only visible/accessible when
- Processor running in kernel (a.k.a supervisor) mode



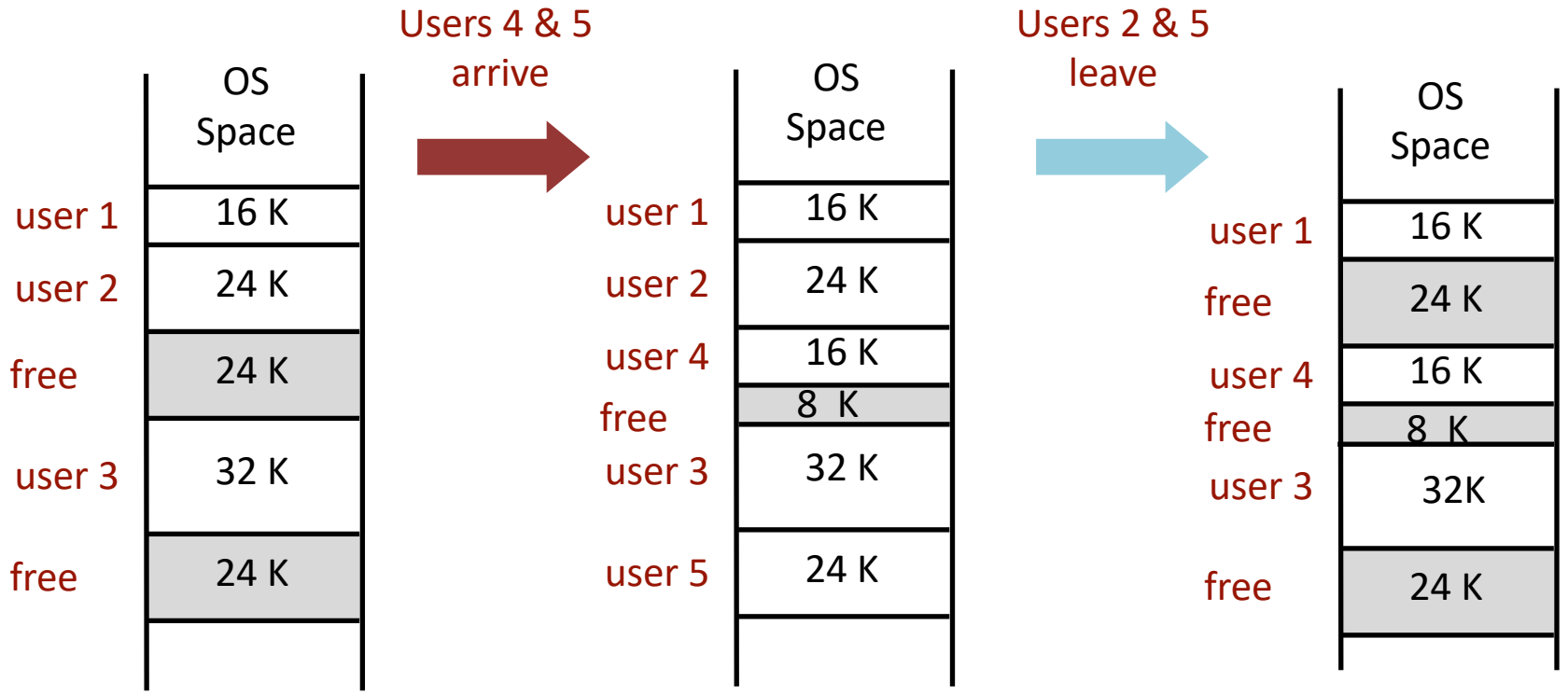
# Separate Areas for Program and Data




- What is an advantage of this separation?
  - Used today on Cray vector supercomputers

# Memory Fragmentation

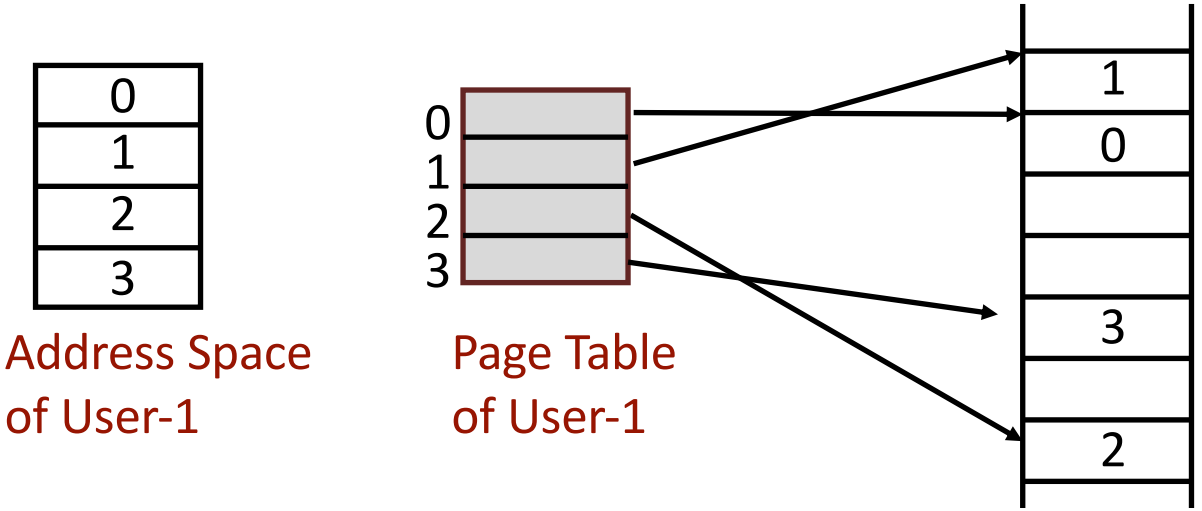
- As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.



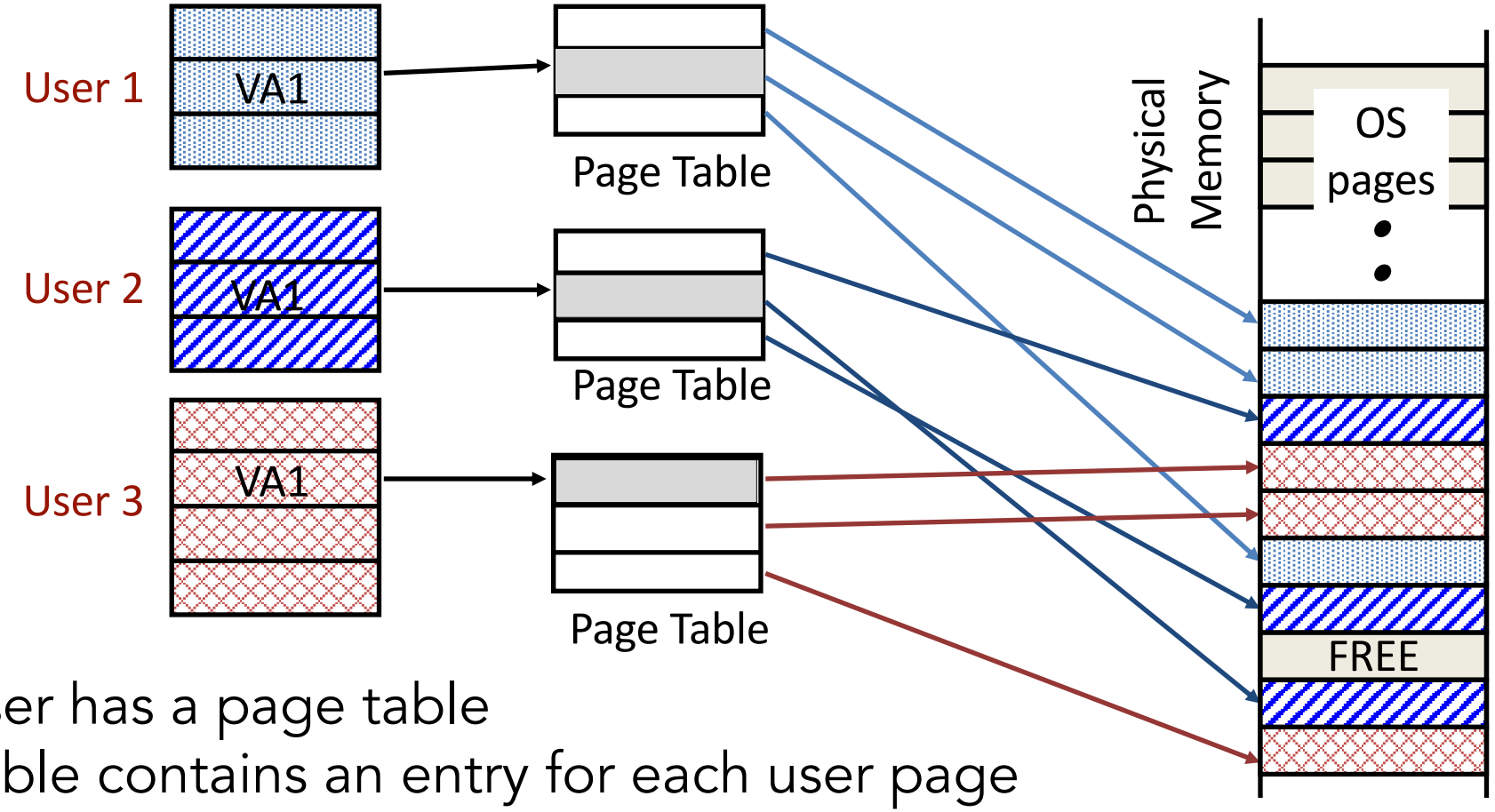
# Paged Memory Systems

- Processor generated address can be interpreted as a pair <page number,offset>


- A page table contains the physical address of the base of each page



# Private Address Space per User

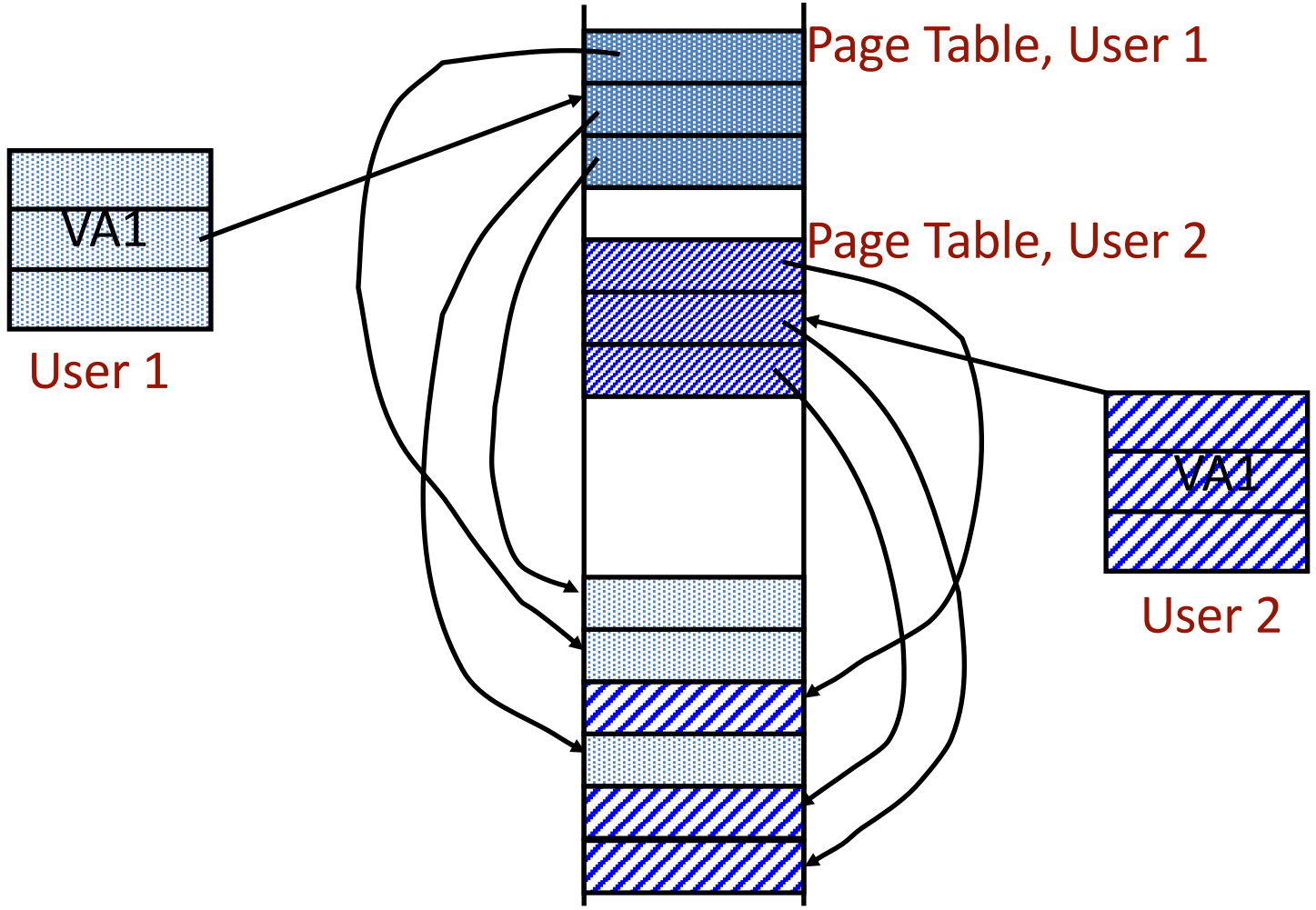



- Each user has a page table
- Page table contains an entry for each user page

# Where Should Page Tables Reside?

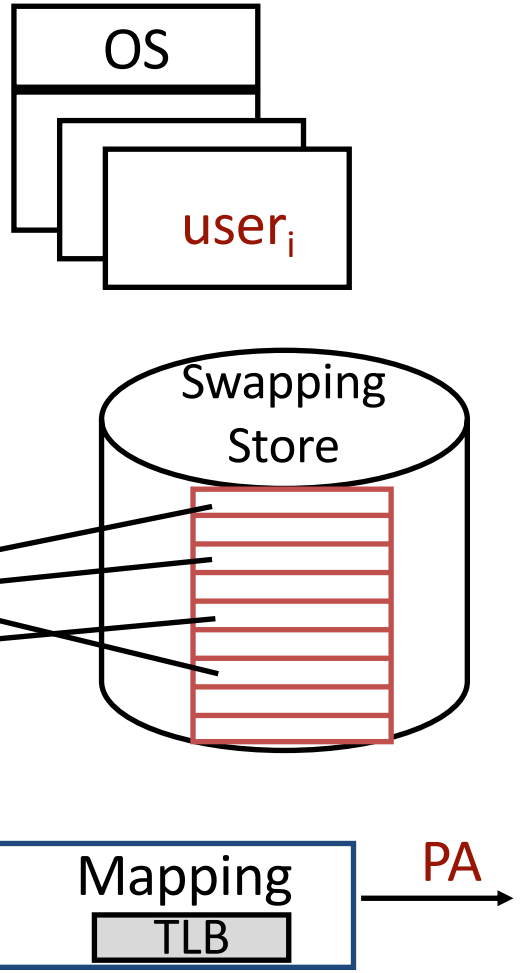
- Space required by the page tables is proportional to the address space, number of users, ...
  - Space requirement is large too expensive to keep in registers
- Special registers just for the current user:
  - What disadvantages does this have?
    - may not be feasible for large page tables
- Main memory:
  - Needs one reference to retrieve the page base address and another to access the data word
    - doubles number of memory references!

# Page Tables in Physical Memory



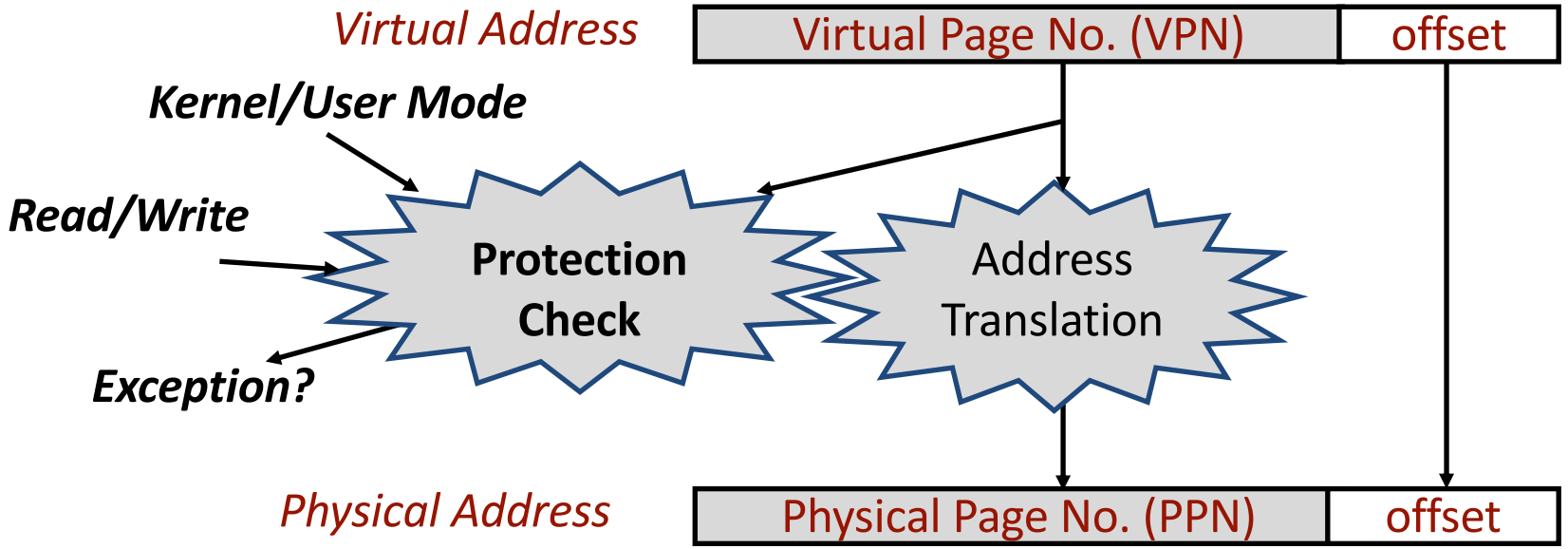
# Modern Virtual Memory Systems

- Protection & Privacy
  - Several users, each with their private address space and one or more shared address spaces
    - Page table ≡ name space
- Demand Paging
  - Ability to run a program larger than the primary memory
- What is another big benefit ?
  - The price is address translation on each memory reference



# Address Translation and Protection

- Every instruction and data access needs address translation and protection checks
- A good VM design needs to be fast (~ one cycle) and space efficient



# Next Learning Module

- Advanced Memory Technologies - PIM, NVM, etc.
- I/O and Interrupts Support System