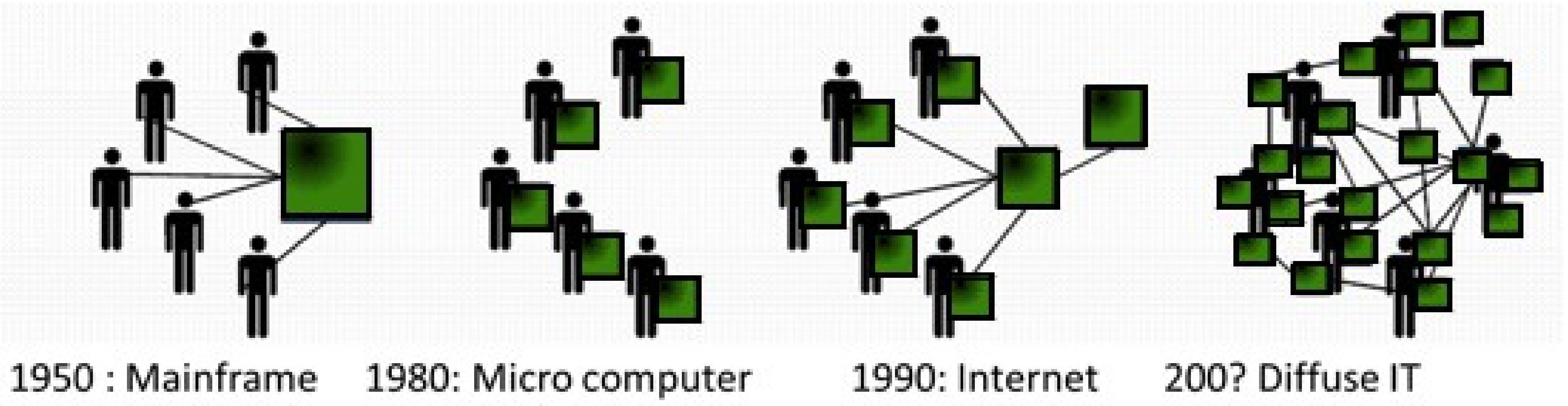
Disciplina
Sistemas de Computação

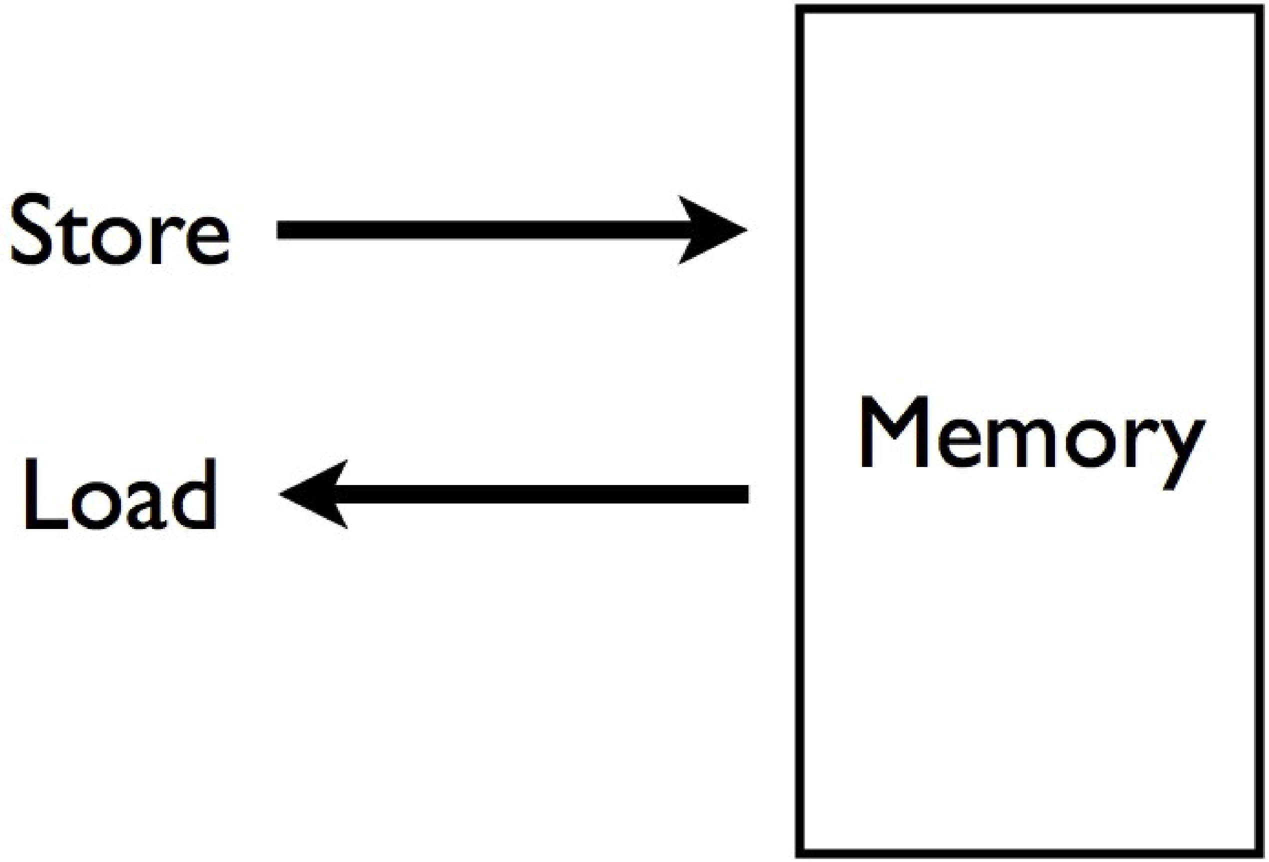
Aula 07



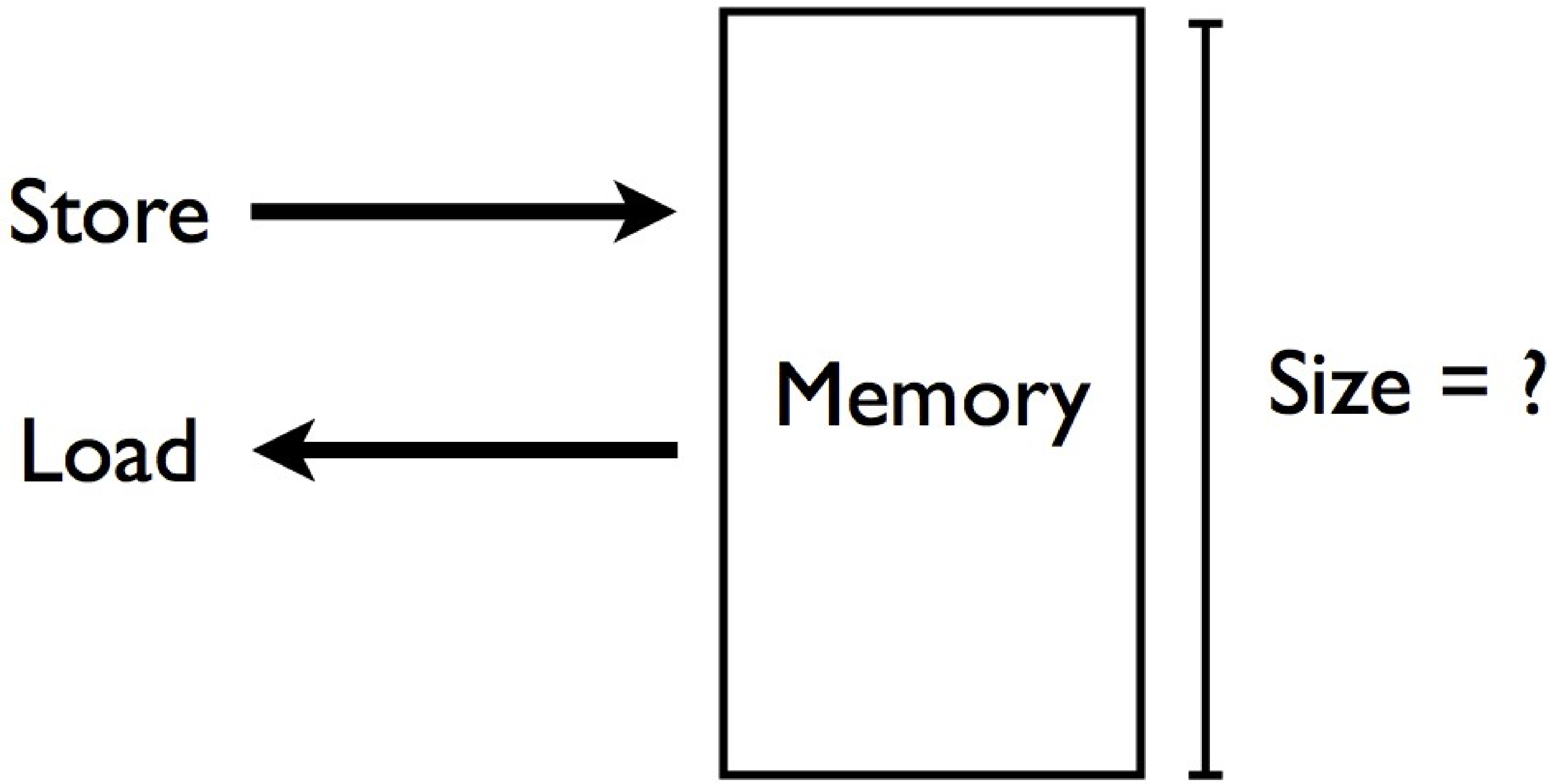
AVISO

- Podemos ter aula na próxima 3a-feira, segundo o calendário acadêmico não é feriado!
  - O que acham?

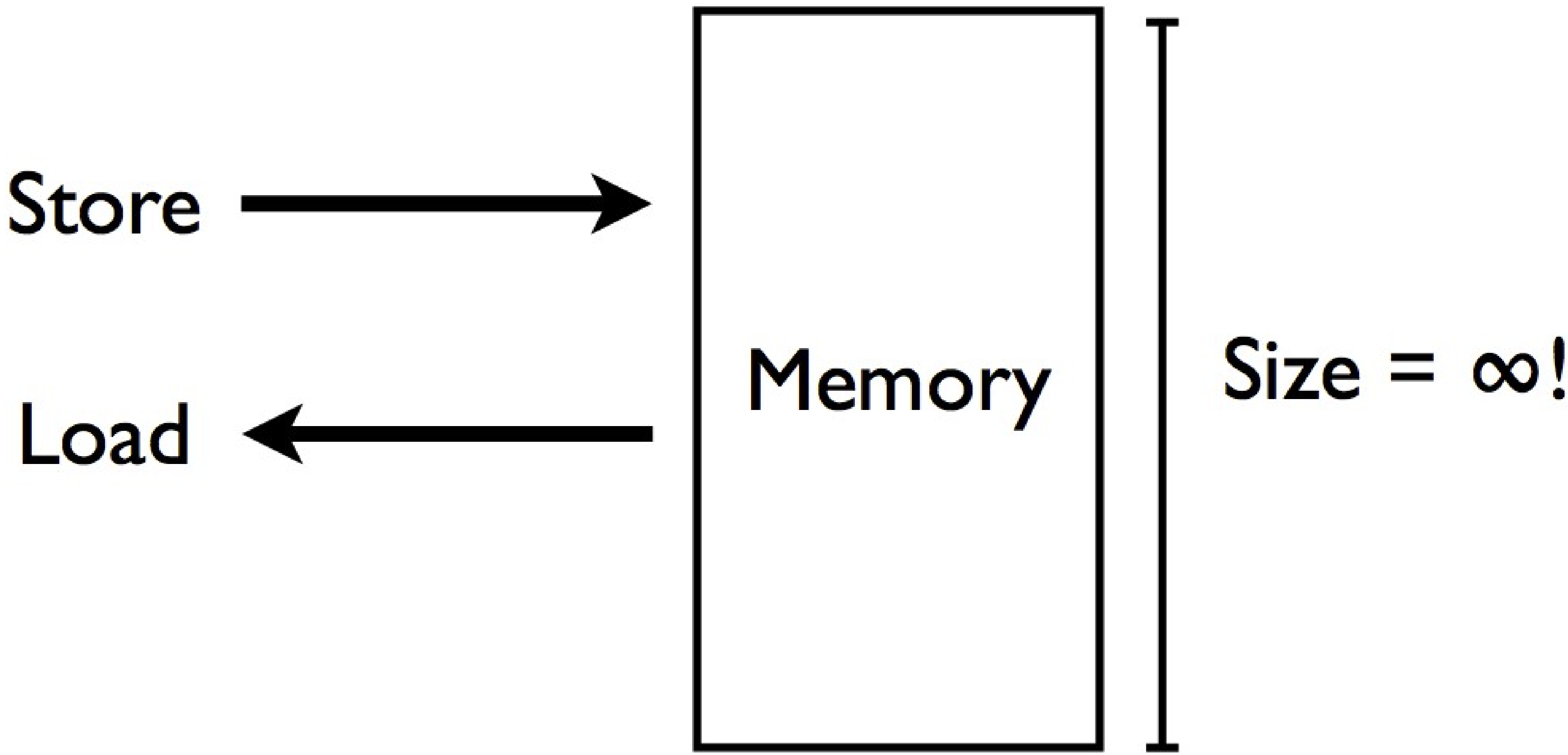
Memory: Programmer's View



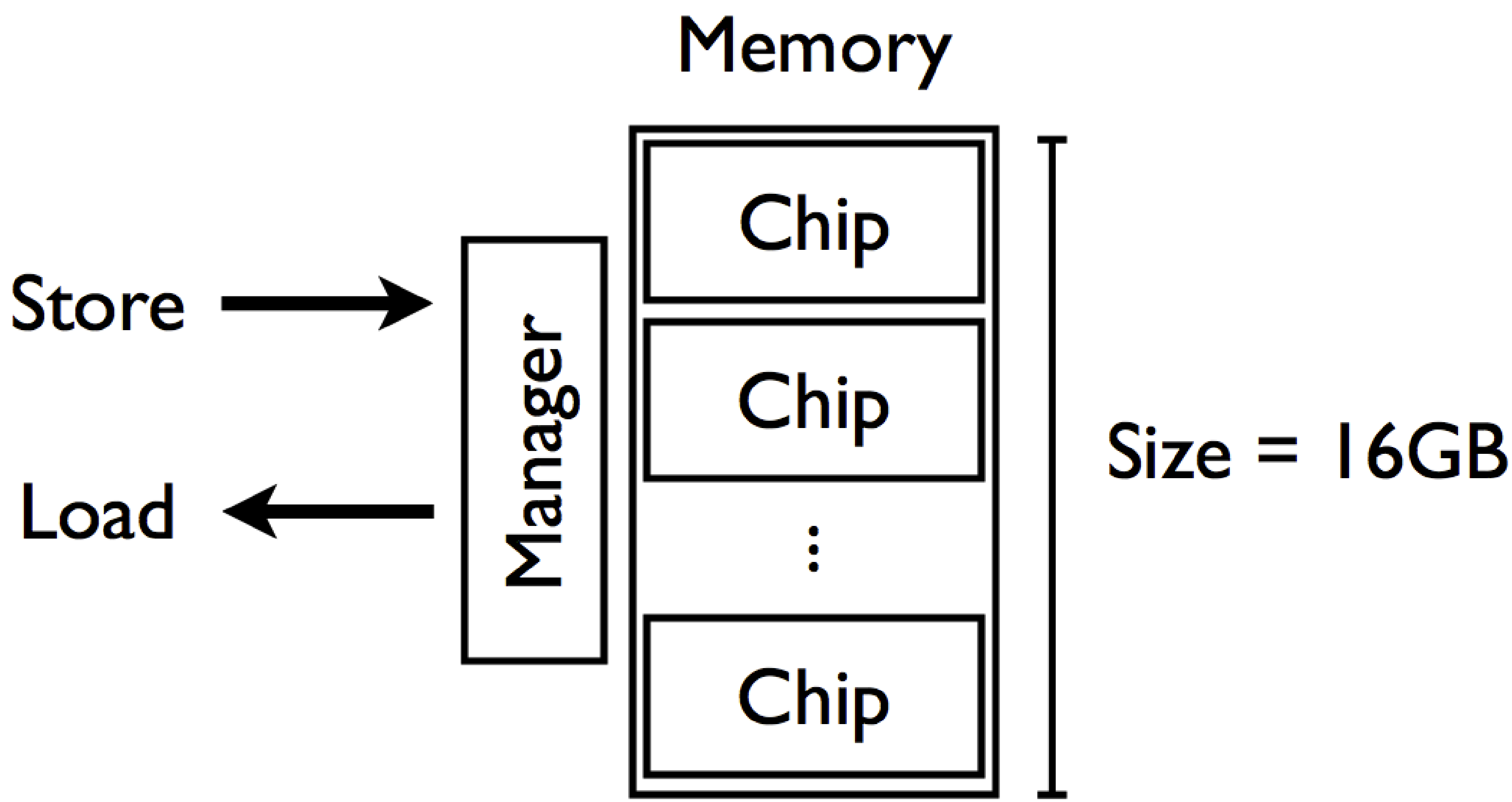
Memory: Programmer's View



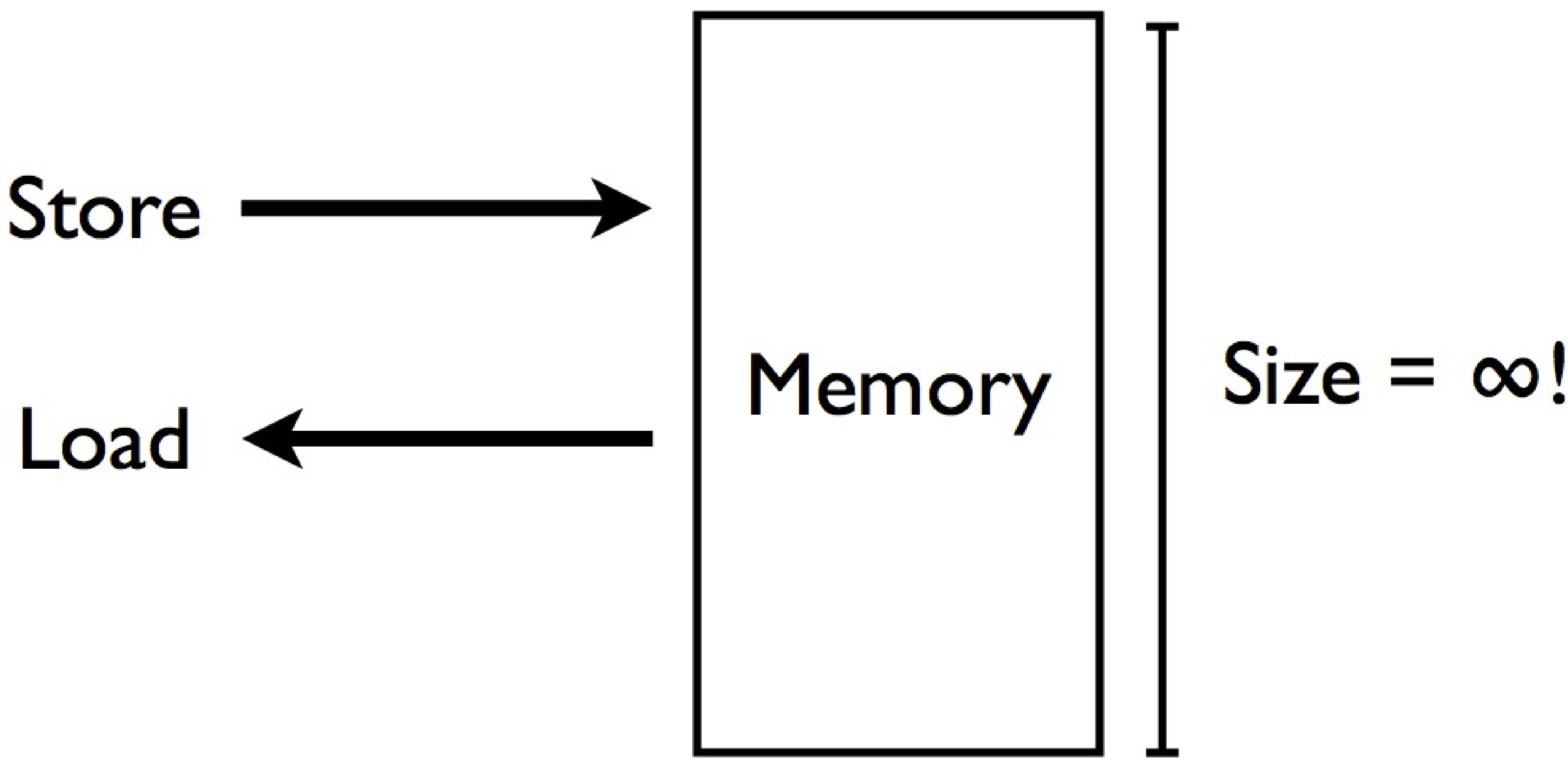
Memory: Programmer's View



How Memory Really Works

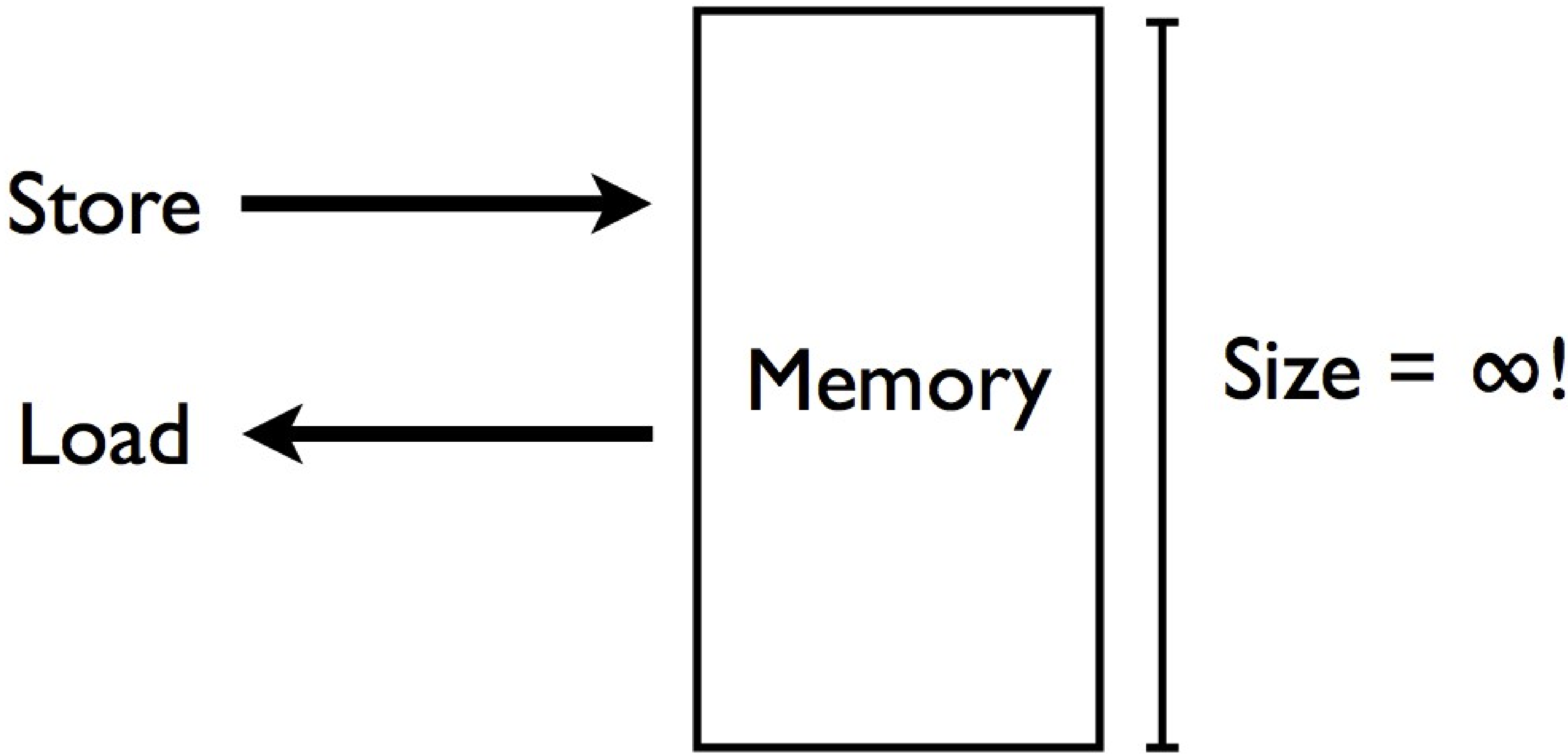


Memory: Programmer's View



How is it possible?

Memory: Programmer's View



How is it possible? **Virtual Memory!**

Virtual Memory

- Why “virtual”?
  - If you think it’s there, and it’s there... it’s real
  - If you think it’s not there, and it’s not there... it’s non-existent
  - If you think it’s not there, and it’s there... it’s transparent
  - If you think it’s there, and it’s not there... it’s imaginary
- Virtual Memory is imaginary memory
  - It gives you the illusion of memory that’s not physically there

Why Virtual Memory?

- Using physical memory efficiently
- Using physical memory simply
- Using physical memory safely

Using Physical Memory Efficiently

- Virtual memory uses gets the most out of physical memory
- Demand paging
  - Main memory is a cache for portions of virtual address space
  - The rest of the virtual address space is stored on disk
- Keep only active areas of virtual address space in fast memory
- Transfer data back and forth as needed

Using Physical Memory Simply

- Virtual memory simplifies memory management
- Programmer can think in terms of a large, linear address space
- Processes access same large, linear address space

Using Physical Memory Simply

- Virtual memory simplifies memory management
- Programmer can think in terms of a large, linear address space
- Processes access same large, linear address space

Using Physical Memory Safely

- Virtual memory protests process’ address spaces
- Processes cannot interfere with each other
  - Because they operate in different address space
- User processes cannot access priviledged information
  - Different sections of address space have different permissions
  - Think: read-only, read/write, execute, ...

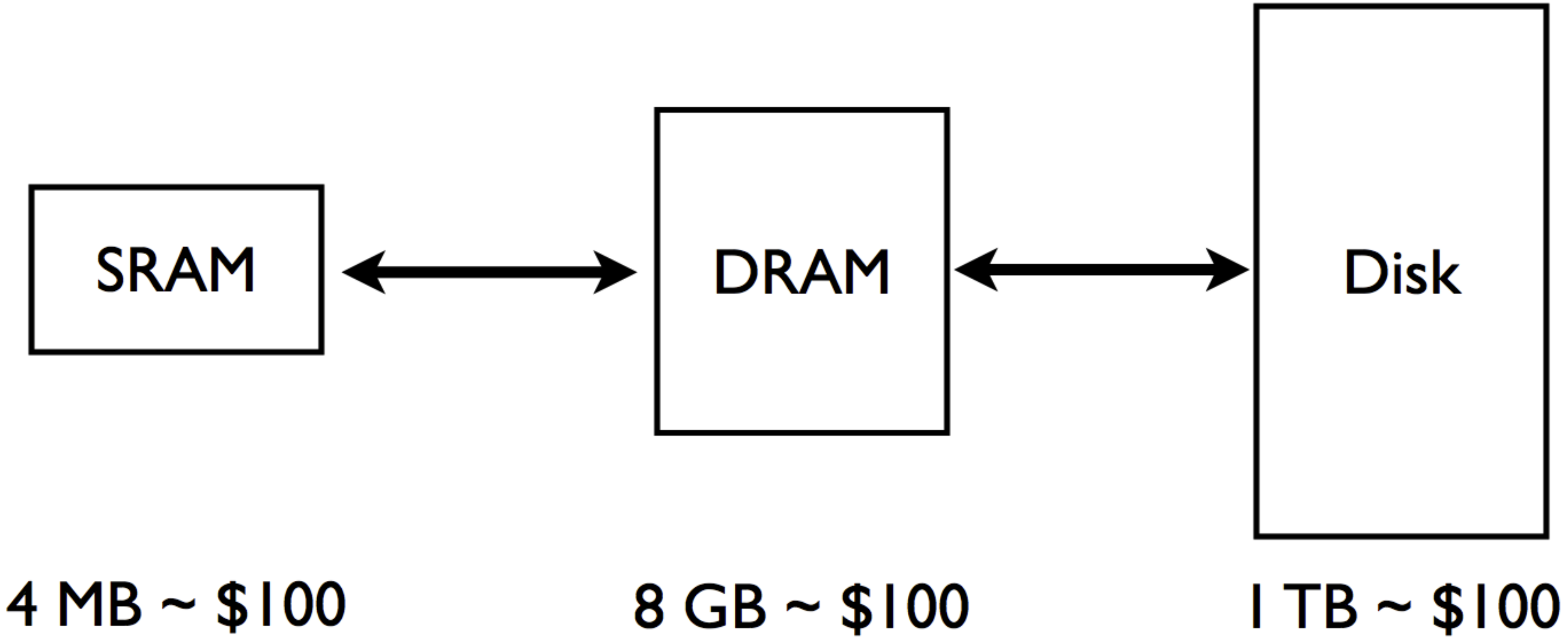
Virtual Memory Benefits

- **Demand paging:** Using physical memory efficiently
- **Memory management:** Using physical memory simply
- **Protection:** Using physical memory safely

The Cost of Maintaining the Illusion of ∞ Memory

- Address space is large:
  - 32-bits: ~4,000,000,000 (four billion) bytes
  - 64-bits: ~16,000,000,000,000,000,000 (sixteen quintillion) bytes
- Memory (DRAM) is expensive
  (1 TB of DRAM ~$10,000)
- But disk storage is relatively cheap
  (1 TB of disk < $100)
- Store most data on disk to maintain the illusion of ∞ memory in a cost-effective way

The Cost of Maintaining the Illusion of ∞ Memory



The Cost of Maintaining the Illusion of ∞ Memory

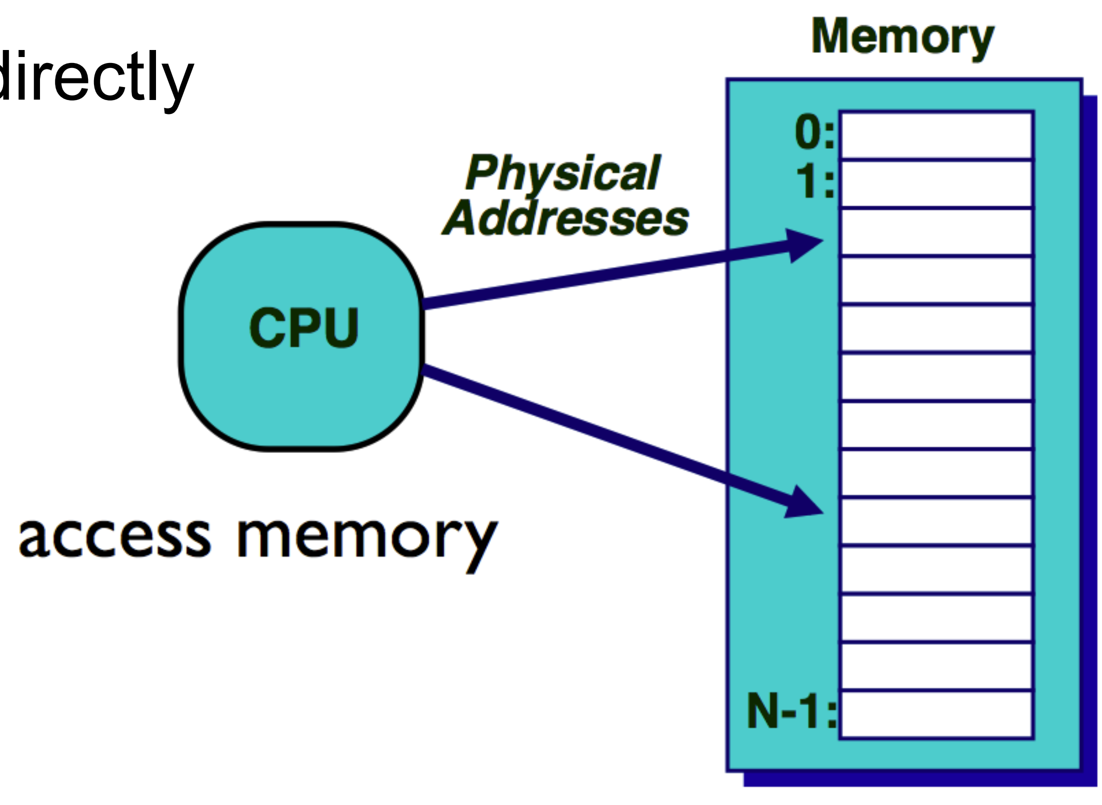
- So, DRAM caches disk data and SRAM caches DRAM data
- Should these caches be built in the same way?

The Cost of Maintaining the Illusion of ∞ Memory

- So, DRAM caches disk data and SRAM caches DRAM data
- Should these caches be built in the same way?
  - Big difference: DRAM ~10X slower than SRAM but disk ~100,000X slower than DRAM

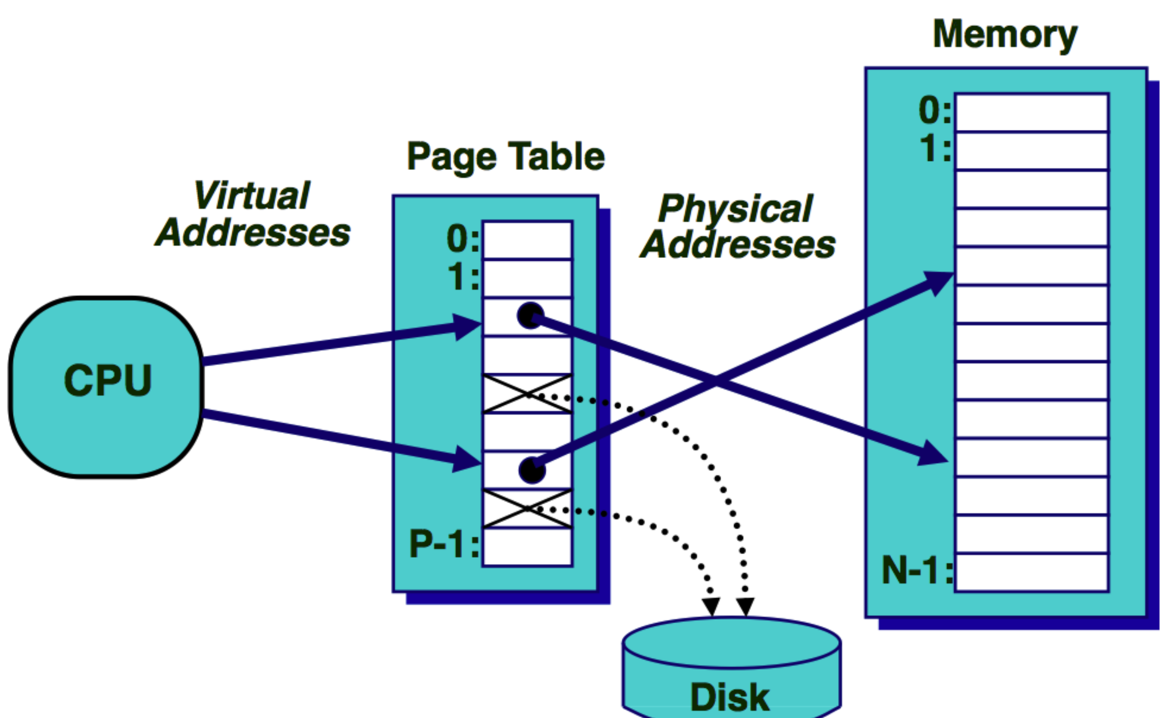
A System with Only Physical Memory

- Example:
  - Most cray machines
  - Early Pcs
  - Most embedded systems
- Loads and stores uses directly to access memory



A System with Virtual Memory

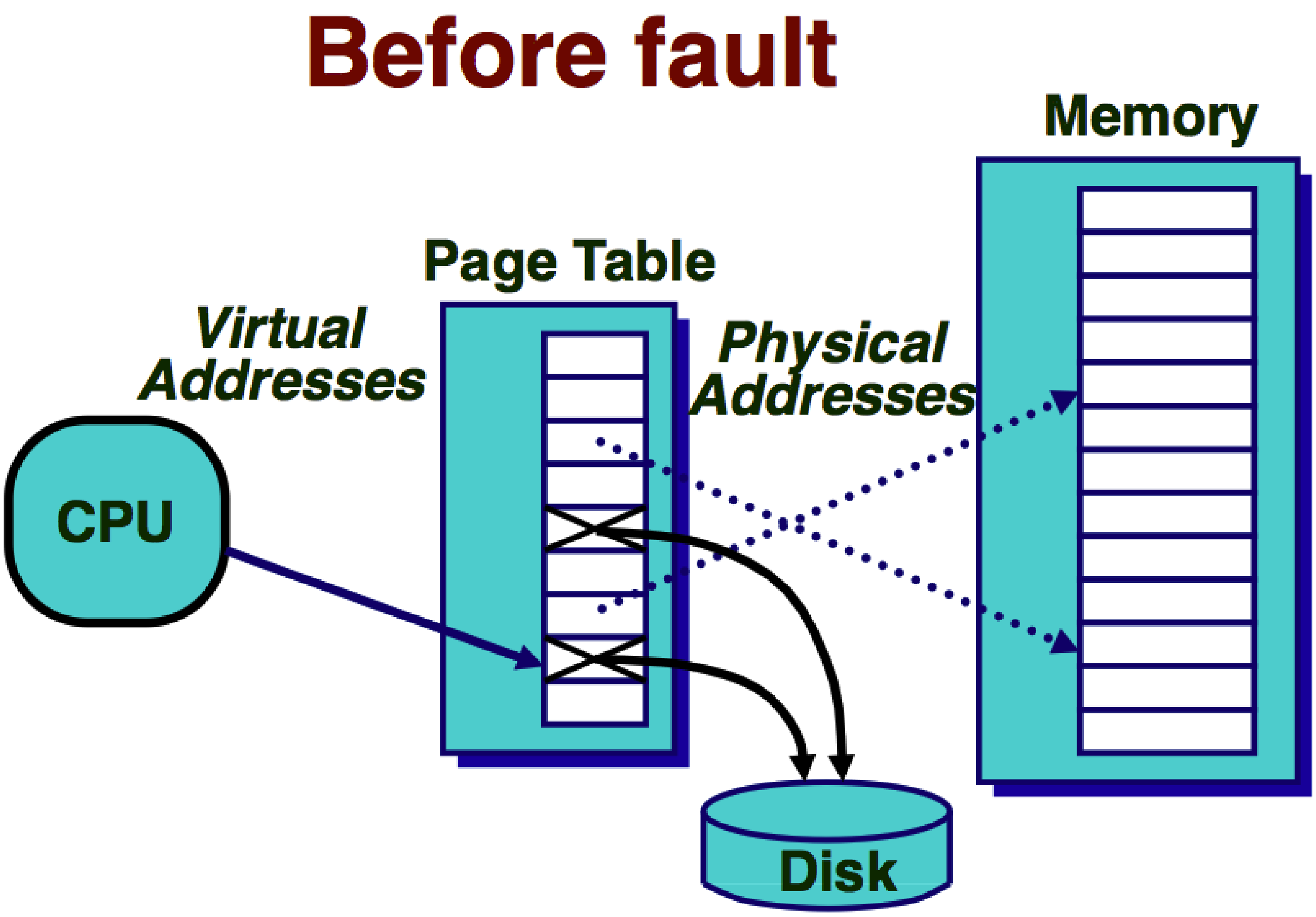
- Example:
  - Most laptops, servers and modern PCs
- Page (i.e., a block)
- Address translation: Hardware converts virtual addresses into physical addresses using an OS-managed lookup table (the page table)



Page Faults

- Problem: A page is on disk and not in memory
  - Page table entry indicates virtual address is not in memory
- Solution: An OS routine is called to load data from disk to memory
  - Current process suspends execution, others may resume
  - OS has full control over placement

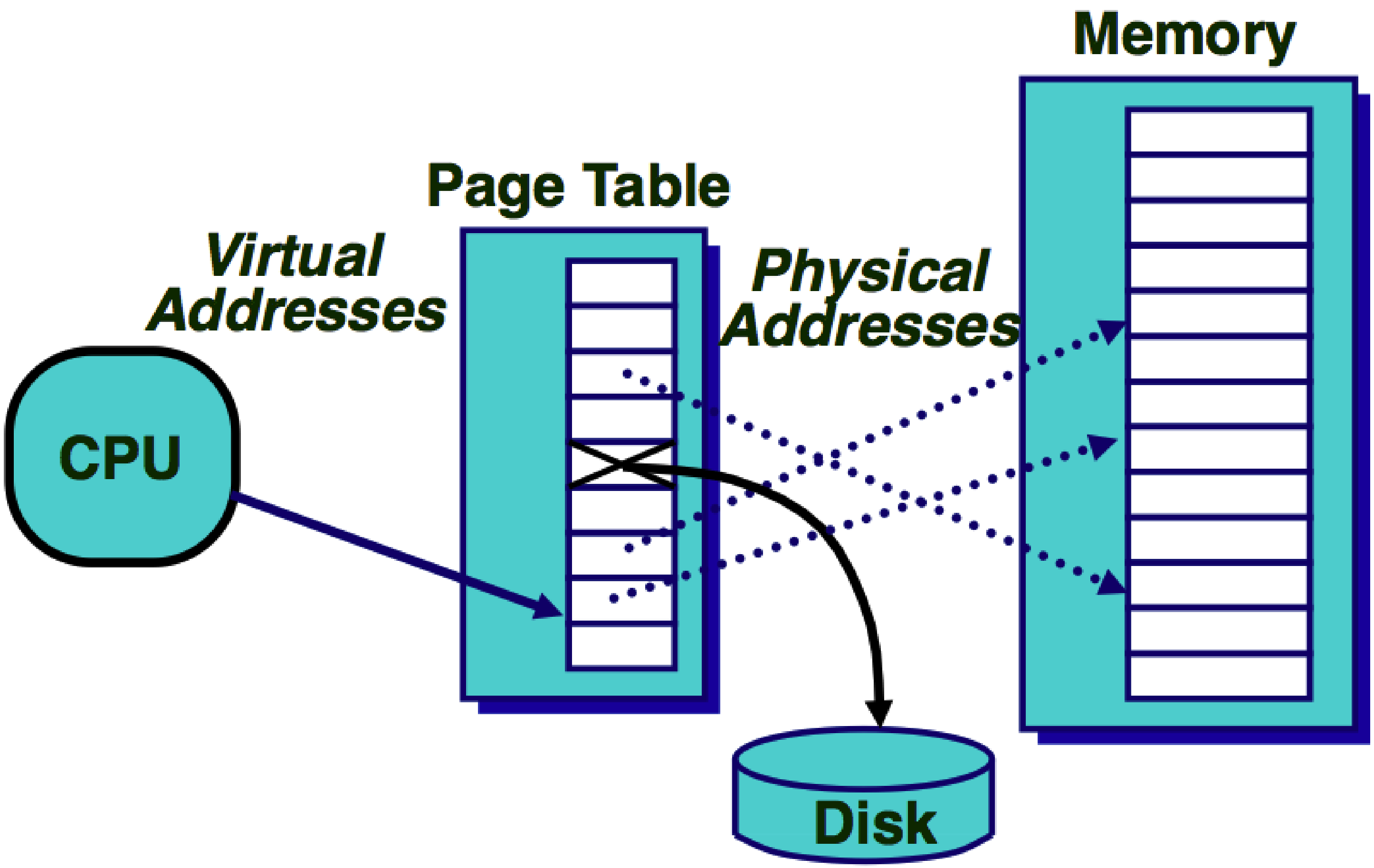
Page Faults



Before fault
Memory
Page Table
Virtual
Addresses
Physical
Addresses
CPU
Disk

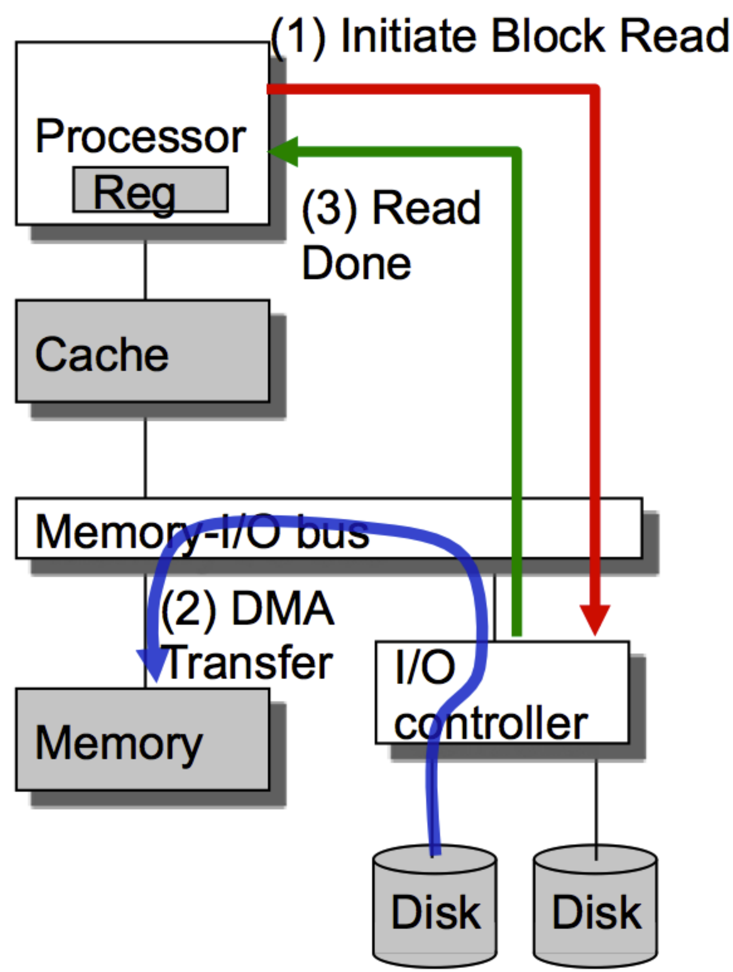
Page Faults

After fault



Servicing a Page Fault

- Processor communicates with controller
  - Read block of length *P* starting at disk address *X* and store starting at memory address *Y*
- Read occurs
  - Direct Memory Access (DMA)
  - Done by I/O controller
- Controller signals completion
  - Interrupt processor invokes OS
  - OS resumes suspended process



Why Does Virtual Memory Work?

- Locality
  - Temporal and Spatial

- Working set: The set of active virtual pages
  - Programs with higher temporal locality have smaller working sets
  - If working set < memory size: good performance after initial misses
  - If working set > memory size: thrashing, pages are copied in and out

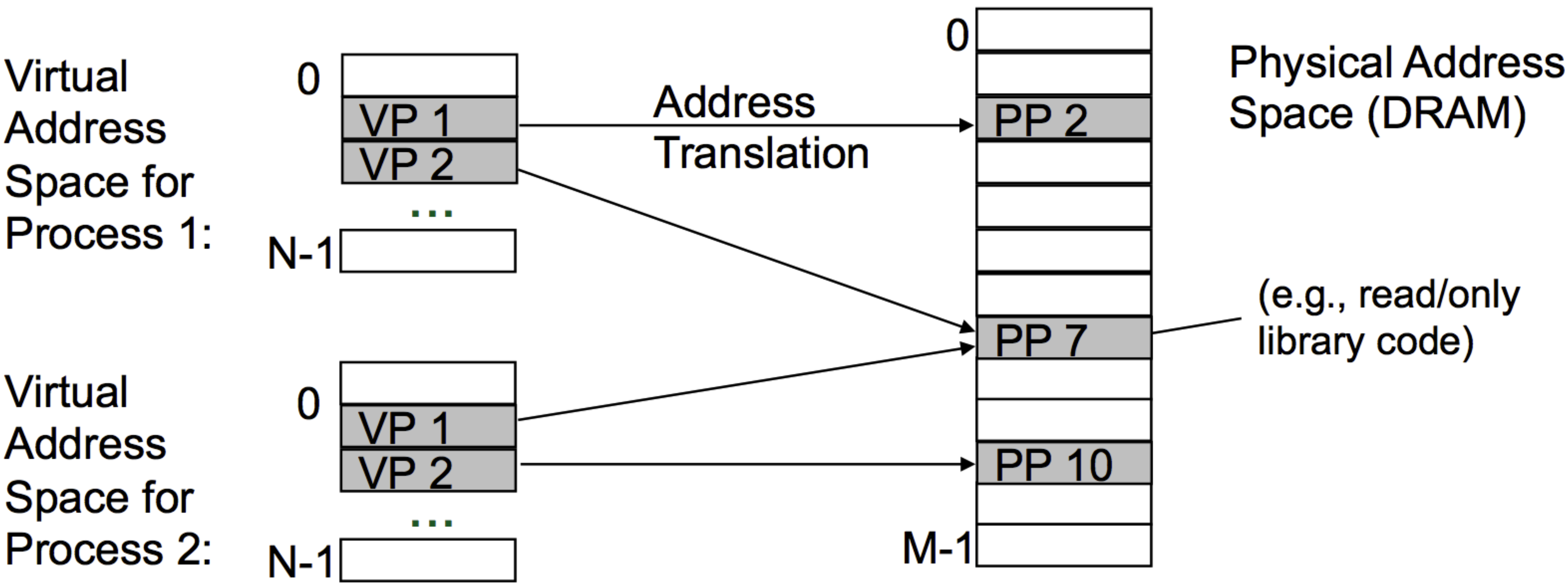
Virtual Memory Benefits

- **Demand paging:** Using physical memory efficiently
- **Memory management:** Using physical memory simply
- **Protection:** Using physical memory safely

Memory Management

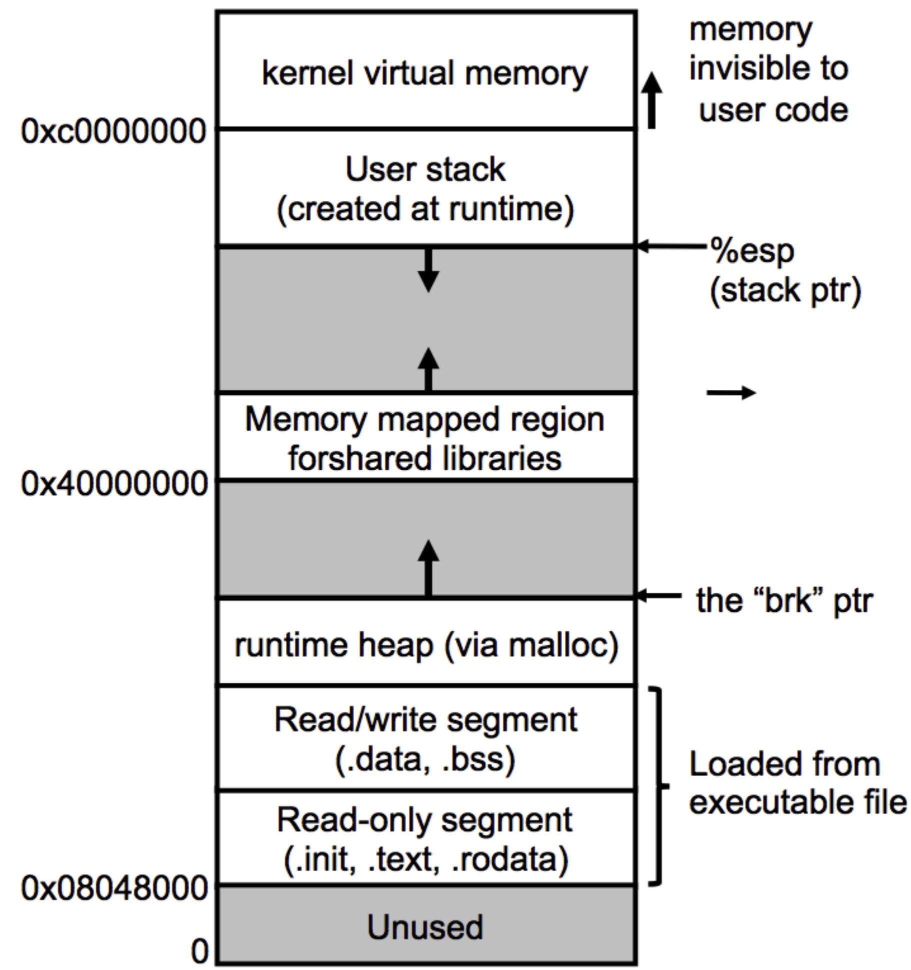
- Virtual / physical address spaces divided into equal-sized blocks
  - "Virtual pages" in virtual memory
  - "Physical pages" or "frames" in physical memory
- Key idea: Each process has its own virtual address space
  - Simplifies memory allocation
    - A virtual page can be mapped to any physical page
  - Simplifies sharing code and data among processes
    - The OS can map virtual pages to same shared physical page linking and loading

Memory Management



Memory Management

- Linking
  - Each program has similar virtual address space
  - Code, stack, and shared libraries always start at the same address
- Loading
  - Virtual pages can be loaded on demand (on first access)



Virtual Memory Benefits

- **Demand paging:** Using physical memory efficiently
- **Memory management:** Using physical memory simply
- **Protection:** Using physical memory safely

Protection

- A normal user process should not be able to:
  - Read/write another process' memory
  - Write into shared library data
- How does virtual memory help?
  - Address space isolation
  - Protection information in page table
  - Efficient clearing of data on newly allocated pages

Protection:Address Isolation

- Processes only access virtual addresses
- Cannot access physical addresses directly
- Go through per-process page table to perform translation
  - If physical page is not in page table, it is not accessible
- A normal user process should not be able to:
  - ~~Read/write another process' memory~~
  - Write into shared library data

Protection:Page Table Information

- Page table entry contains permission information
  - Hardware enforces this protection
  - OS is summoned if a violation occurs (send process SIGSEGV, segmentation fault)
  - The page table itself is in protected memory (only OS can update)

Protection:Leaked Information

- Programmer shouldn't have to worry about their data being leaked
- OS can ensure that pages are initialized to all zeros when allocated
- Let's use what we've learned to do this quickly in virtual memory
  - Remember shared pages? New pages can share an all-zero page
    - Saves a lot of initial stores of the value zero to memory
  - The OS can *copy-on-write* when the all-zero page is stored to
    - Allocates a new virtual page on demand (what is this also useful for? ➔ forking / threading)