Final Exam

- **December 14 (Tuesday), 2010.**
- **15:00 – 16:30, #400102**
- **Closed-book exam**
- **Scope**
  - Chap. 1, Chap. 2, Chap. 3.1-3.12
  - Chap. 4.1, 4.4
  - Chap. 6, Chap. 7.1-7.9
  - Chap. 8.1 – 8.3
  - Chap. 9.1 – 9.6
  - Lecture slides

Operating Systems

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

What is an OS? (1)

- **Application view**
  - Provides an execution environment for running programs
  - Provides an abstract view of the underlying computer system
    - Processors → Processes, Threads
    - Memory → Address spaces (virtual memory)
    - Storage → Volumes, Directories, Files
    - I/O Devices → Files (ioctls)
    - Networks → Files (sockets, pipes, ...)
    - ...

What is an OS? (2)

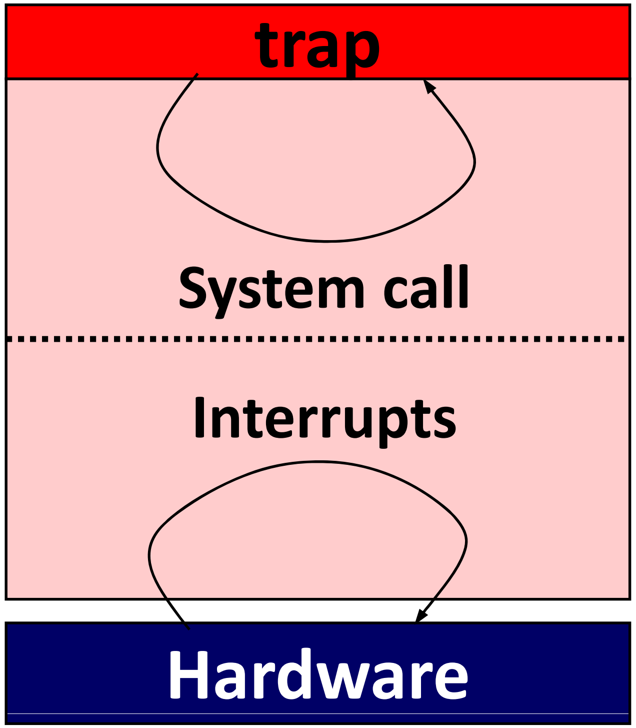
- **System view**
  - Manages various resources of a computer system
  - Sharing
  - Protection
  - Fairness
  - Efficiency
  - …

Resources

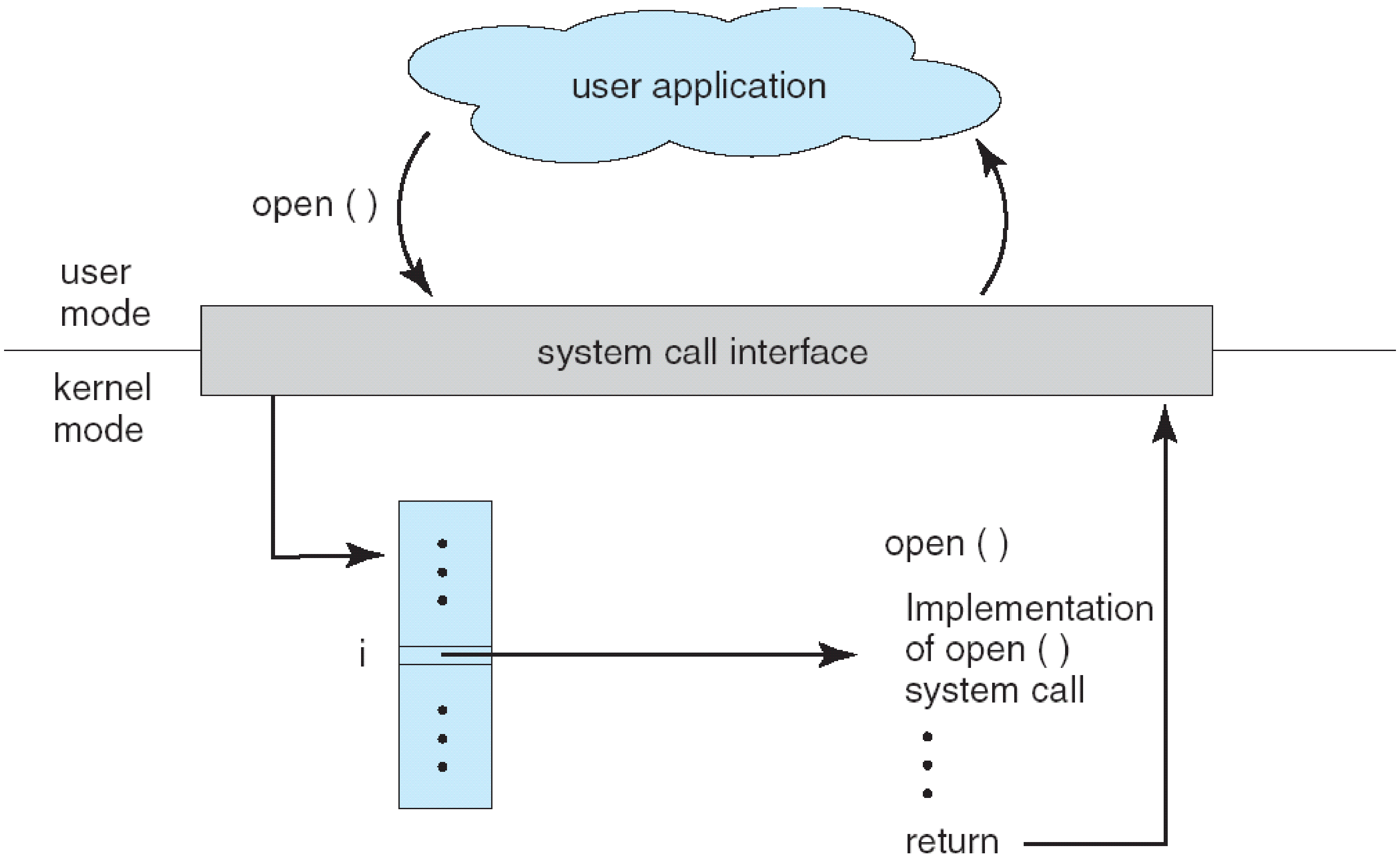
- **CPU**
- **Memory**
- **I/O devices**
- **Queues**
- **Energy**
- **…**

What is an OS? (3)

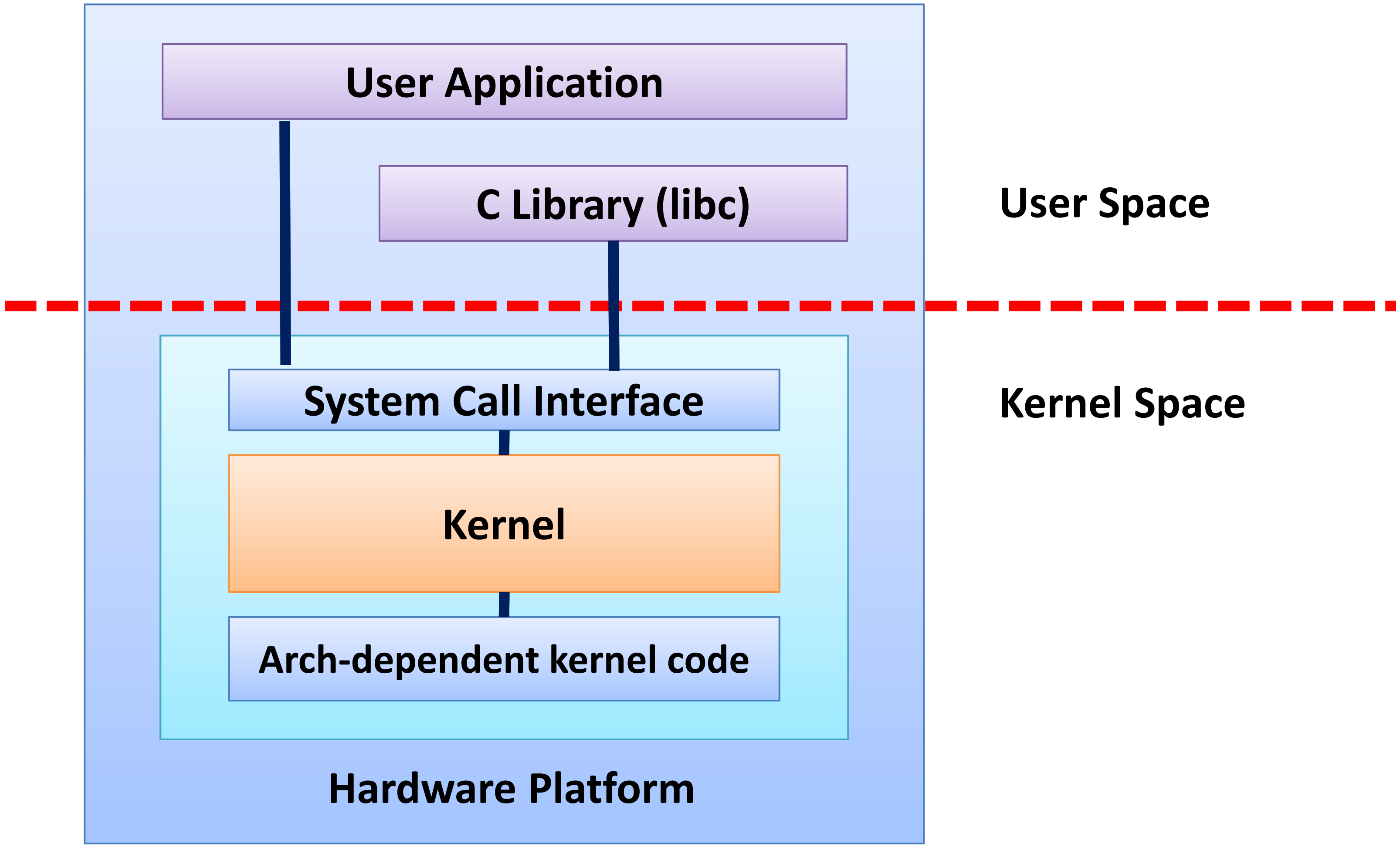
- **Implementation view**
  - Highly-concurrent, event-driven software



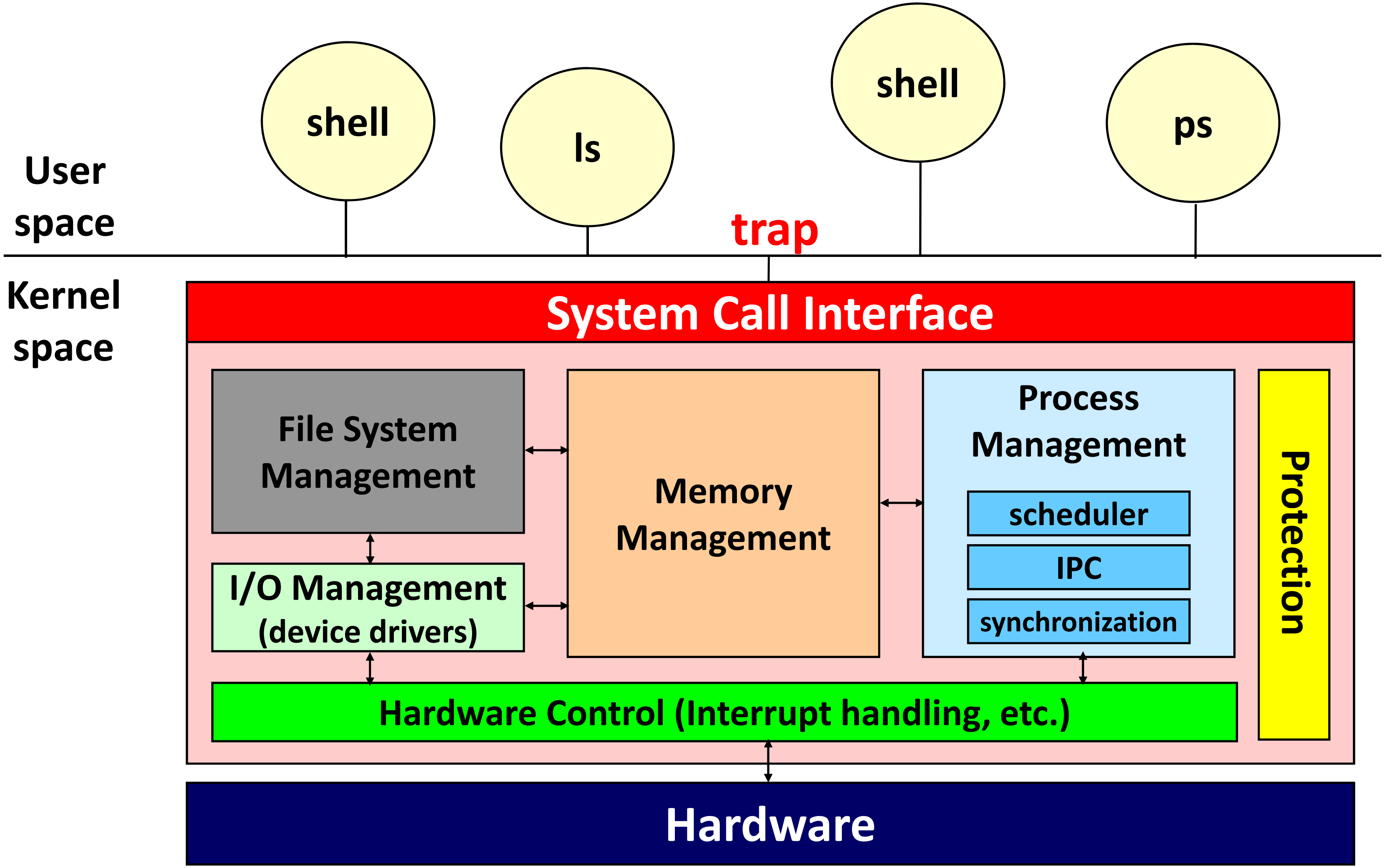
System Calls



OS Structure (1)



OS Structure (2)

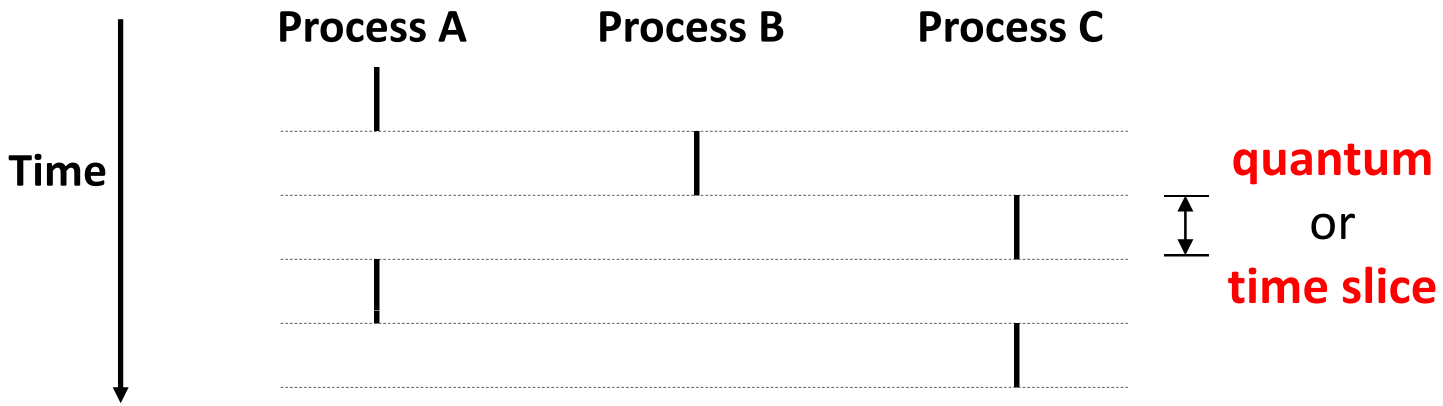


Process

- **An instance of a program in execution**
  - (cf.) program? processor?
- **Two key abstractions**
  - Logical control flow
    - Each program seems to have exclusive use of the CPU
  - Private address space
    - Each program seems to have exclusive use of main memory
- **How are these illusions maintained?**
  - Process executions interleaved (multitasking)
  - Address space managed by virtual memory

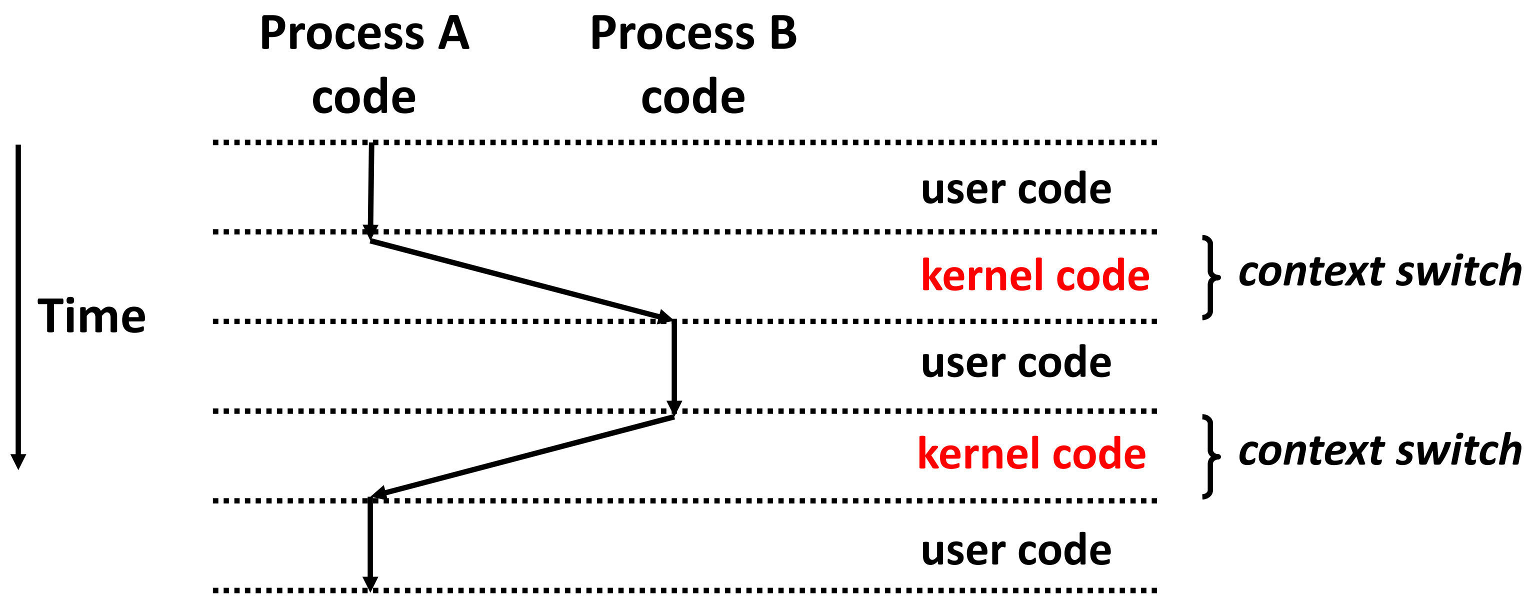
Logical Control Flows

- **Each process has its own logical control flow.**

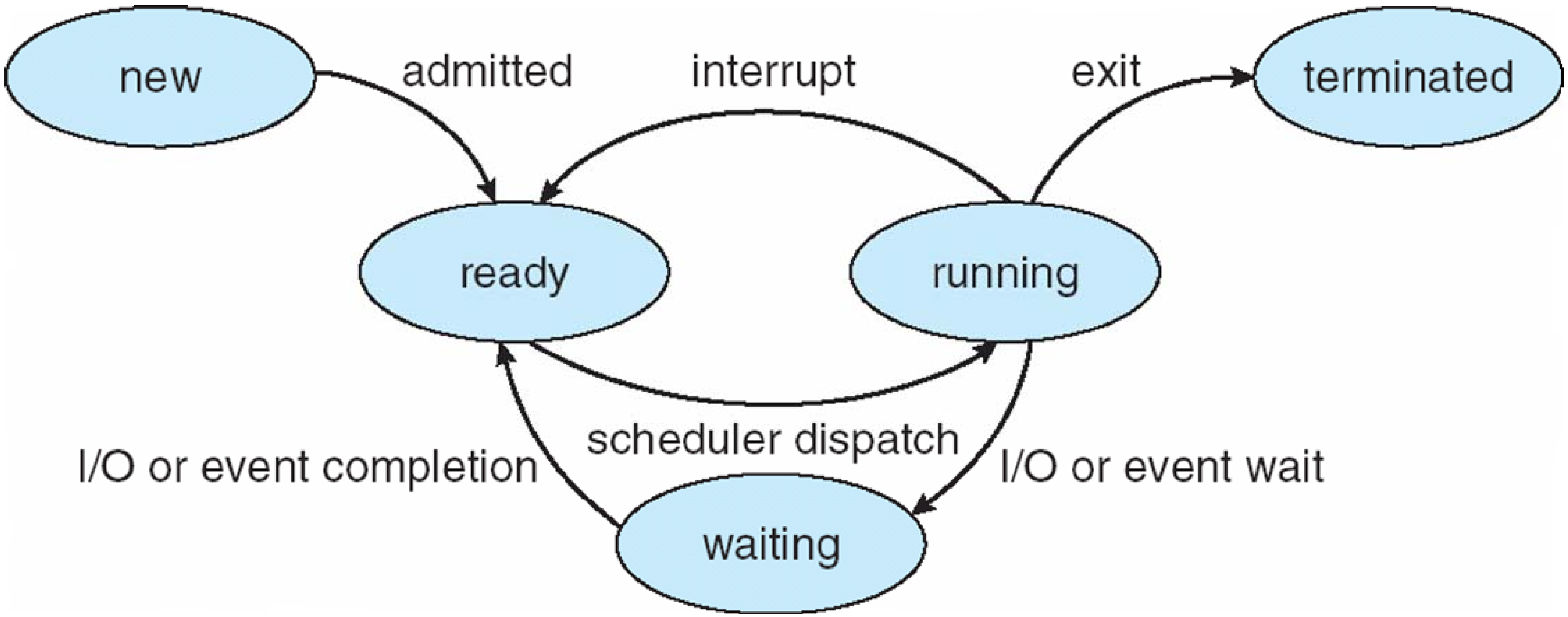


Context Switching

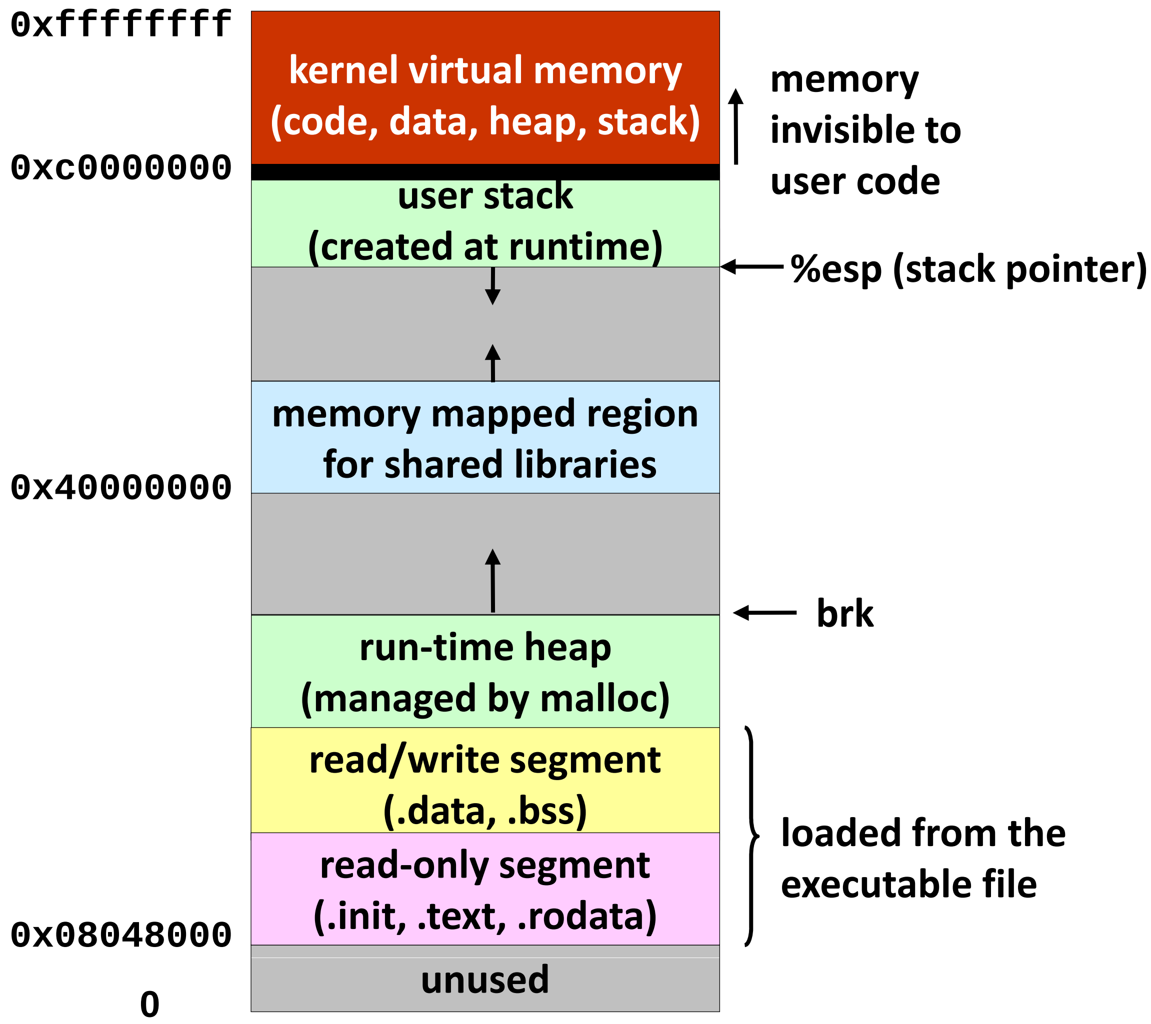
- **Context switching**
  - Control flow passes from one process to another via a context switch.



Process State Transition



Private Address Spaces (1)



Private Address Spaces (2)

- **Example**

```
#include <stdio.h>

int n = 0;

int main ()
{
     n++;
     printf ("n = %d, &n = 0x%08x\n", n, &n);
}

% ./a.out
n = 1, &n = 0x08049508
% ./a.out
n = 1, &n = 0x08049508
```

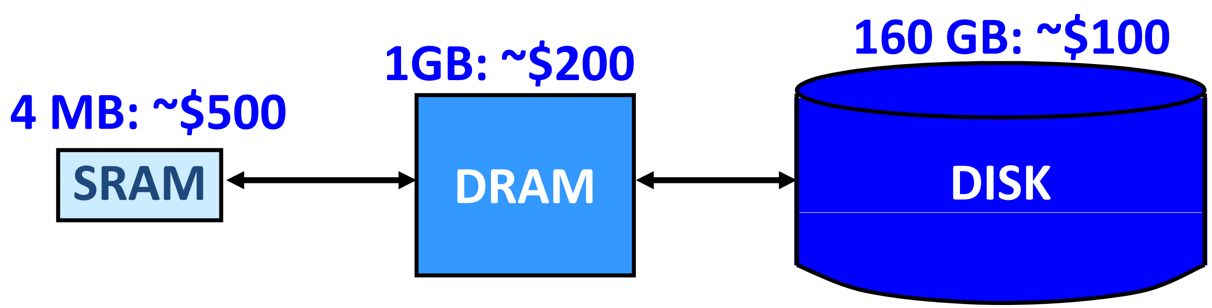
- What happens if two users simultaneously run this application?

VM Motivations

- **Use physical DRAM as a cache for the disk**
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- **Simplify memory management**
  - Multiple processes resident in main memory
    - Each process with its own address space
  - Only "active" code and data is actually in memory
    - Allocate more memory to process as needed
  - Provide virtually contiguous memory space
- **Provide protection**
  - One process can't interfere with another
    - Because they operate in different address spaces
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions.

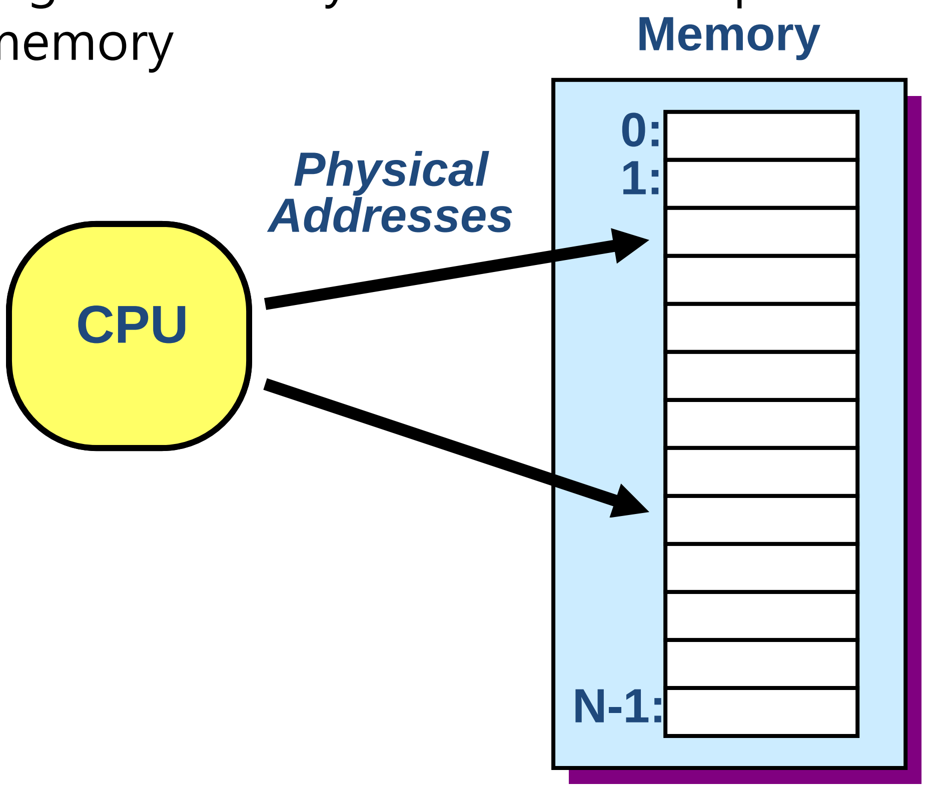
Motivation #1: Caching

- **Use DRAM as a cache for the disk**
  - Full address space is quite large:
    - 32-bit addresses: 4 billion (~4 x 10^9) bytes
    - 64-bit addresses: 16 quintillion (~16 x 10^{18}) bytes
  - Disk storage is ~300X cheaper than DRAM storage
    - 160GB of DRAM: ~$32,000
    - 160GB of disk: ~$100
  - To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk



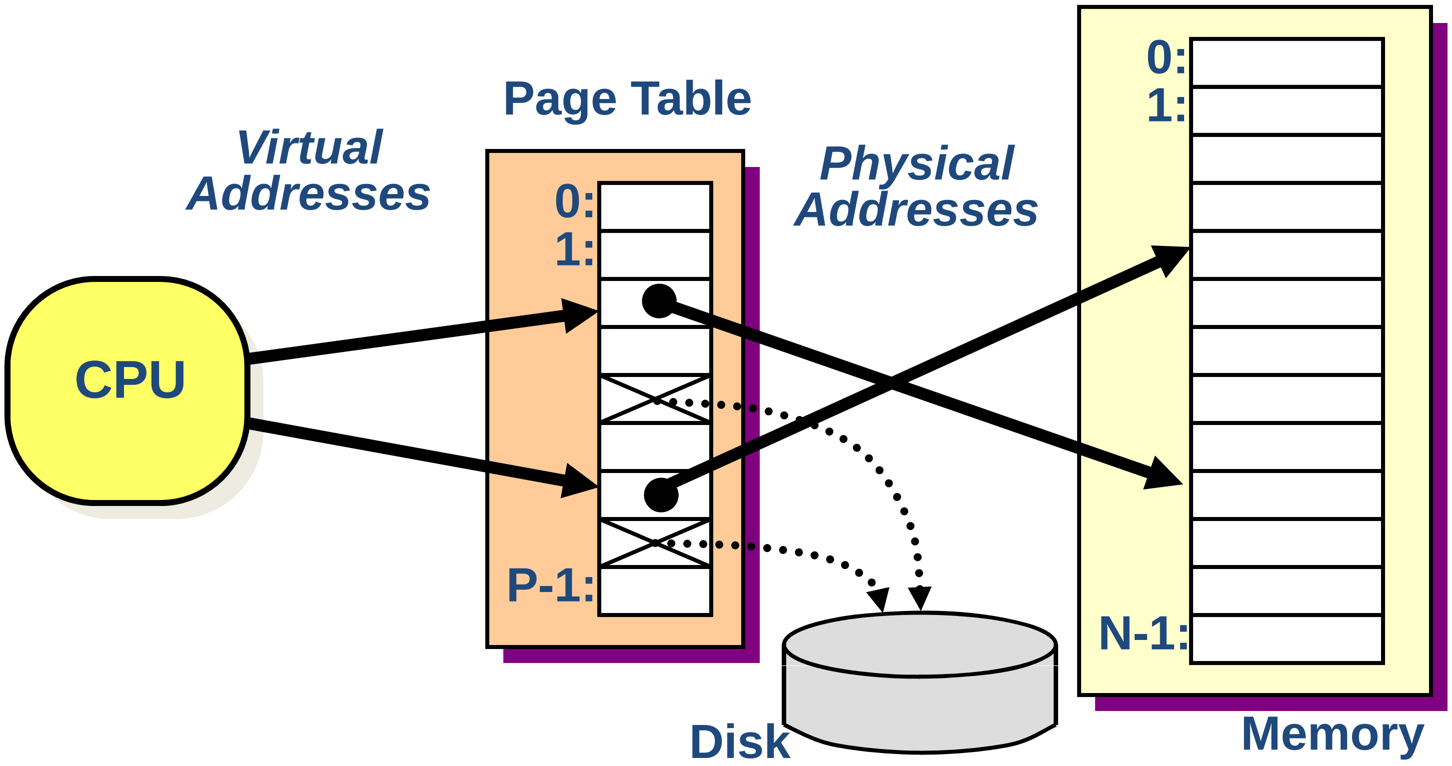
Physical Addressing

- **Examples**
  - Most Cray machines, early PCs, nearly all embedded systems, etc.
  - Addresses generated by the CPU correspond directly to bytes in physical memory



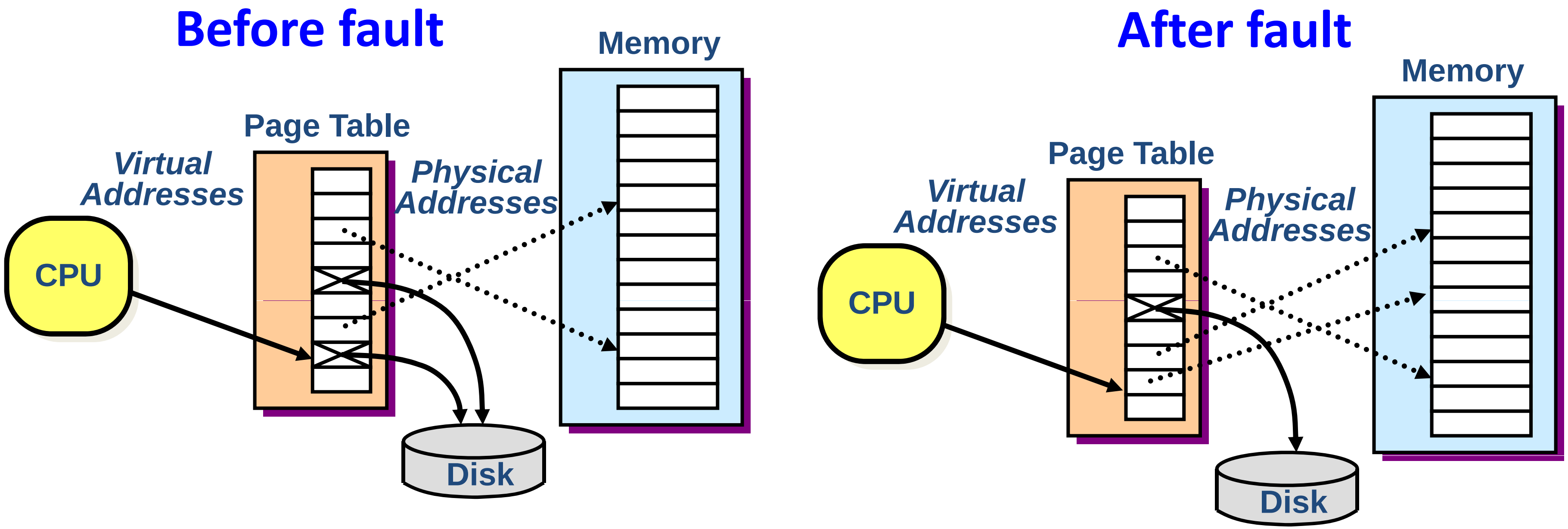
Virtual Addressing

- **Examples**
  - Workstations, servers, modern PCs, etc.
  - Address translation: Hardware converts virtual addresses to physical addresses via OS-managed lookup table (page table)



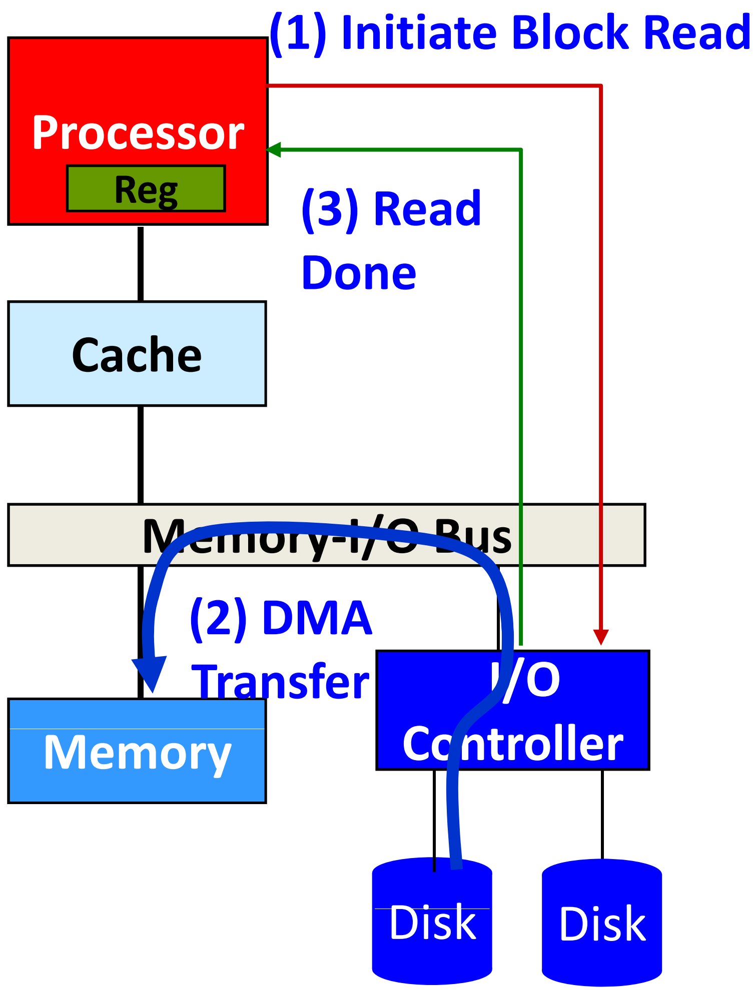
Page Faults ("Cache Misses")

- **What if an object is on disk rather than in memory?**
  - Page table entry indicates virtual addresses not in memory
  - OS exception handler invoked to move data from disk into memory
    - Current process suspends, others can resume
    - OS has full control over placement, etc.



Servicing a Page Fault

- **Processor signals controller**
  - Read block of length P starting at disk address X and store starting at memory address Y
- **Read occurs**
  - Direct Memory Address (DMA)
  - Under control of I/O controller
- **I/O controller signals completion**
  - Interrupt processor
  - OS resumes suspended process



Motivation #2: Management

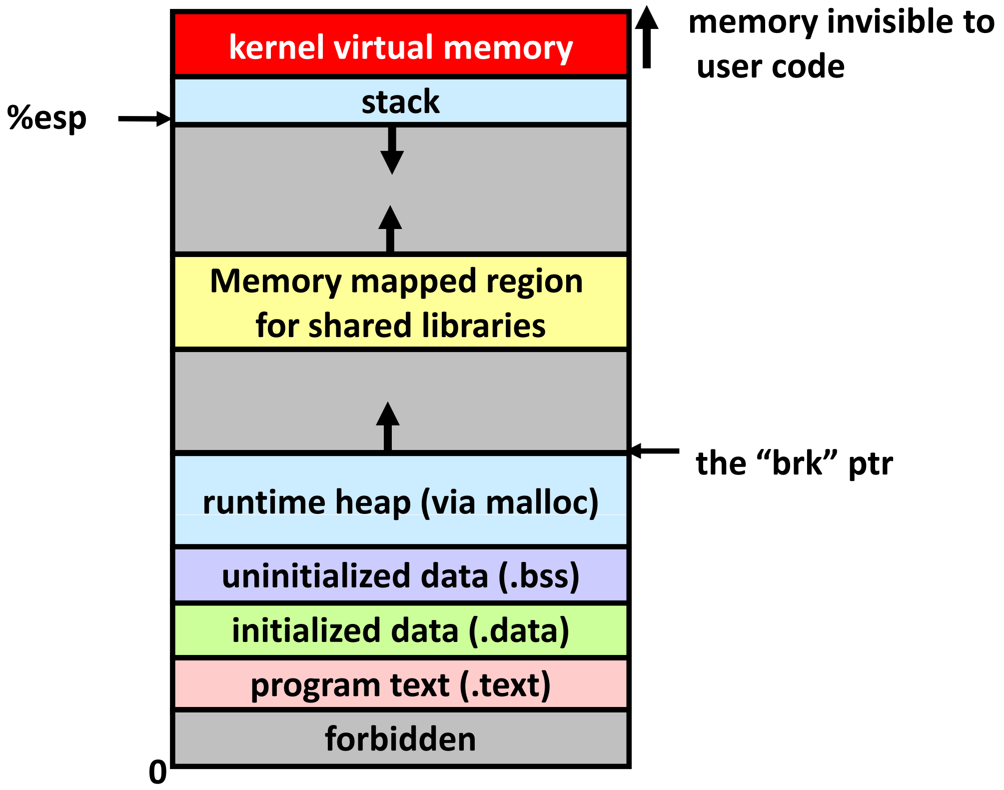
- **Multiple processes can reside in physical memory**
- **How do we resolve address conflicts?**
  - What if two processes access something at the same address?

Linux/x86

process

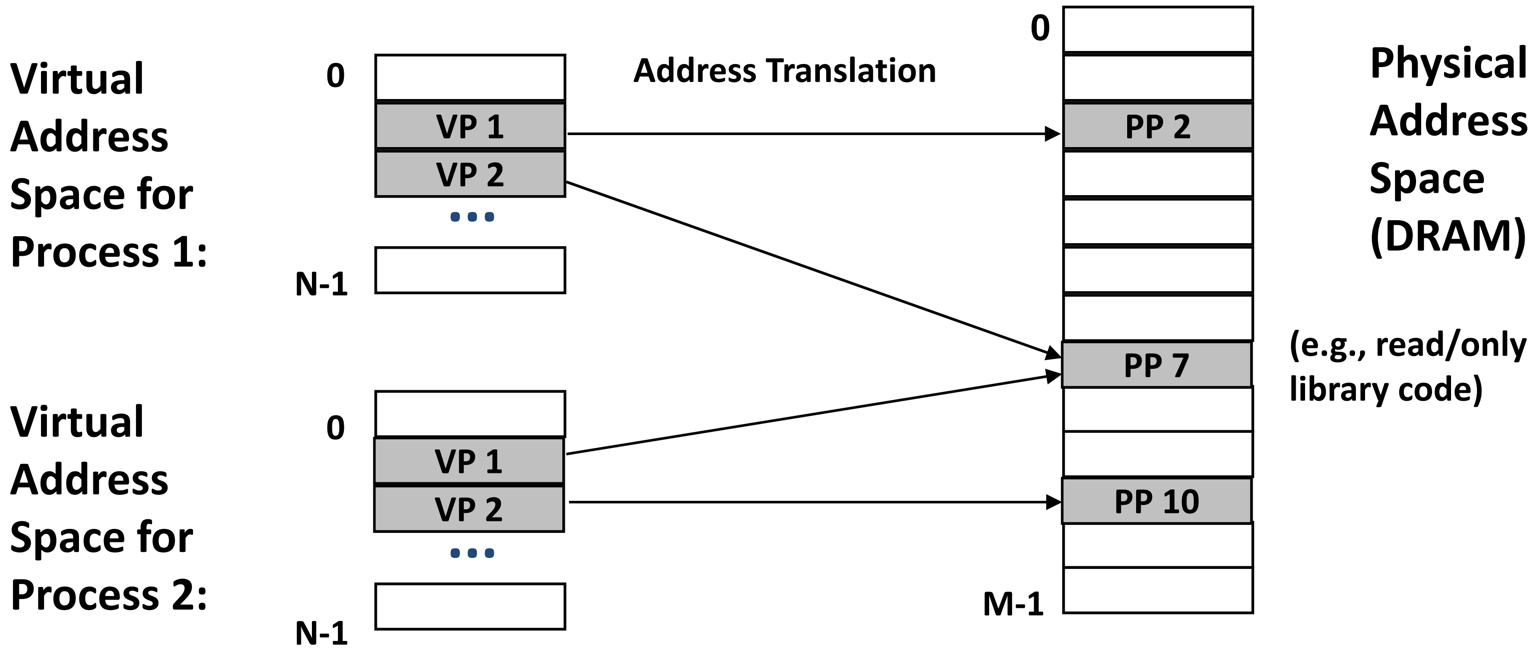
memory

image



Virtual Address Spaces

- **Solution: Separate Virtual Address Spaces**
  - Virtual and physical address spaces divided into equal-sized blocks
    - Blocks are called "pages" (both virtual and physical)
  - Each process has its own virtual address space
    - OS controls how virtual pages as assigned to physical memory

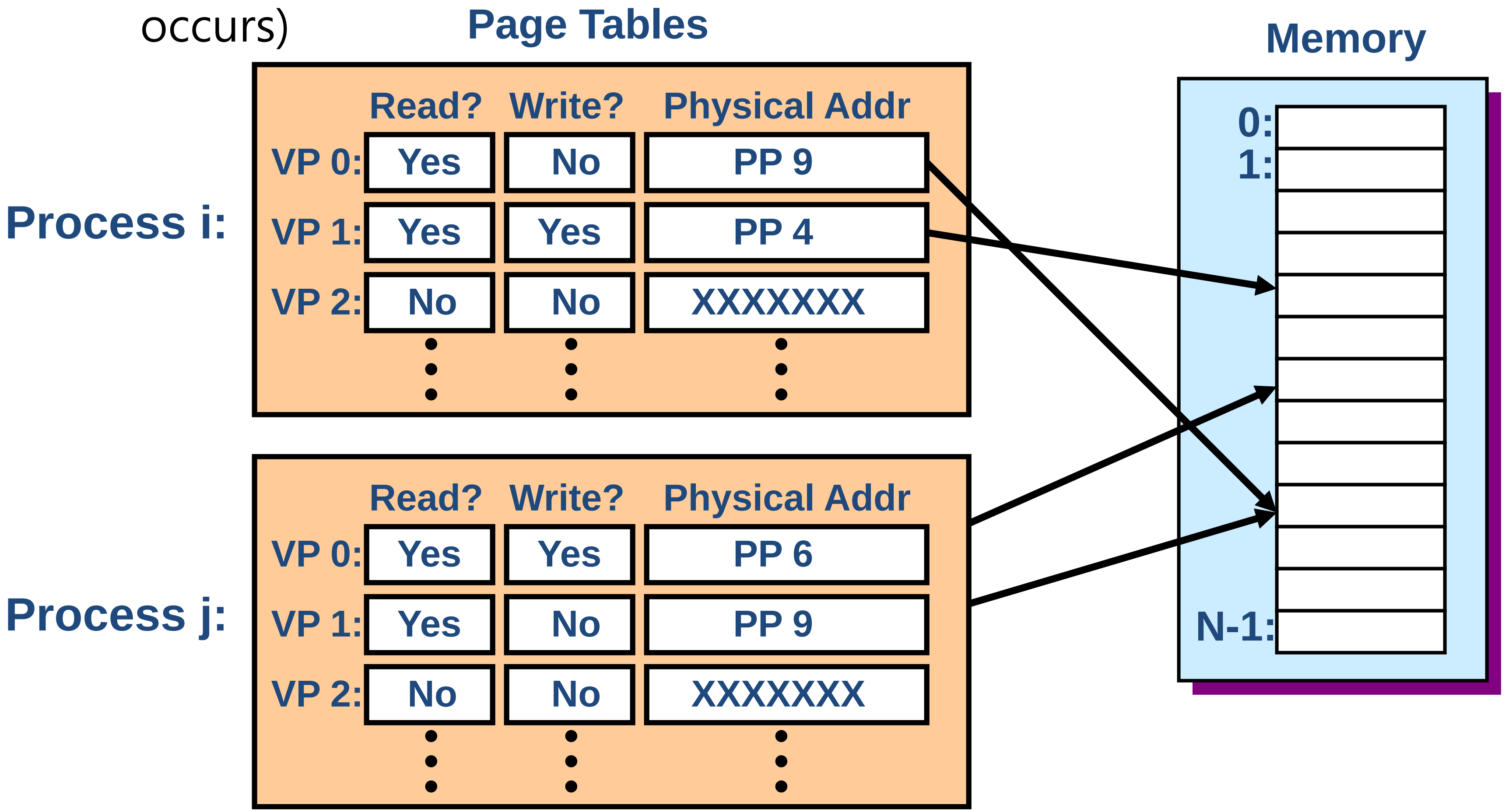


Memory Management

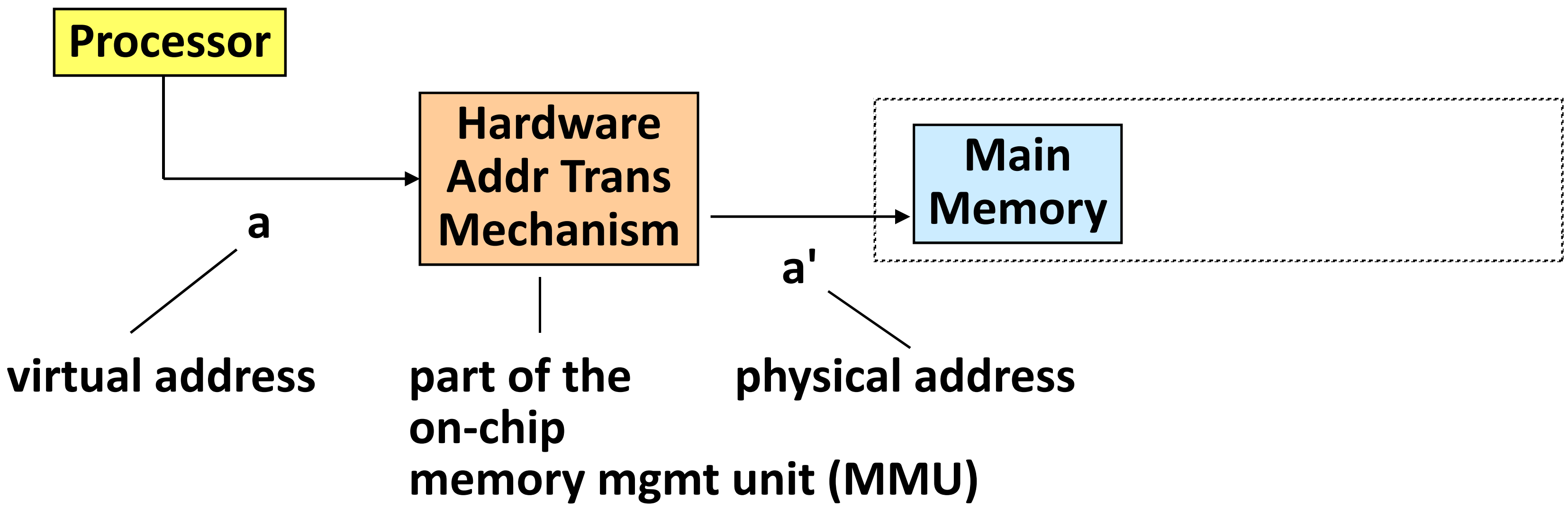
- **Allocating, deallocating, and moving memory**
  - Can be done by manipulating page tables
- **Allocating contiguous chunks of memory**
  - Can map contiguous rage of virtual addresses to disjoint rages of physical addresses
- **Loading executable binaries**
  - Just fix page tables for processes
  - Data in the binaries are paged in on demand
- **Protection**
  - Store protection information on page table entries
  - Usually checked by hardware

Motivation #3: Protection

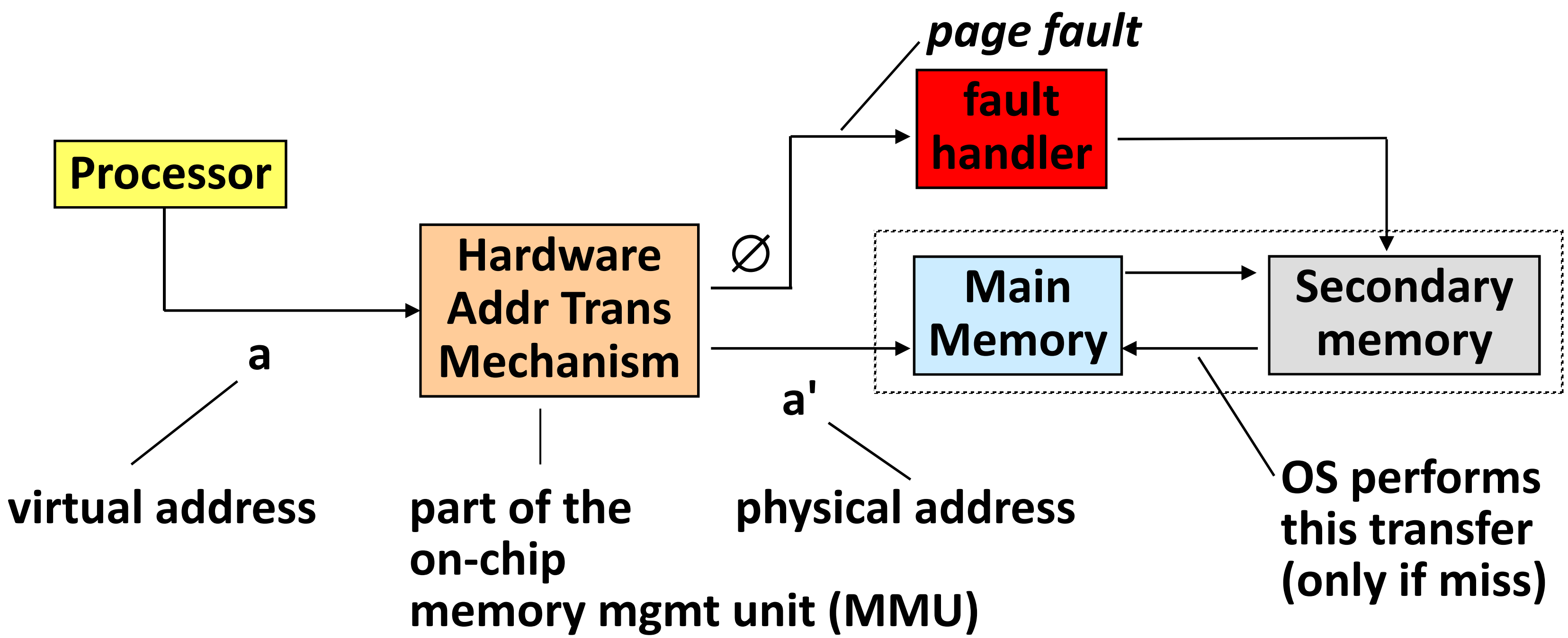
- **Page table entry contains access rights information**
  - Hardware enforces this protection (trap into OS if violation occurs)



VM Address Translation: Hit



VM Address Translation: Miss



Virtual Memory (1)

- **Programmer's View**
  - Large "flat" address space
    - Can allocate large blocks of contiguous addresses
  - Process "owns" machine
    - Has private address space
    - Unaffected by behavior of other processes

Virtual Memory (2)

- **System View**
  - Use virtual address space created by mapping to set of pages
    - Need not be contiguous
    - Allocated dynamically
    - Enforce protection during address translation
  - OS manages many processes simultaneously
    - Continually switching among processes
    - Especially when one must wait for resources
      - e.g. disk I/O to handle page faults