Introduction

Yajin Zhou (http://yajin.org)

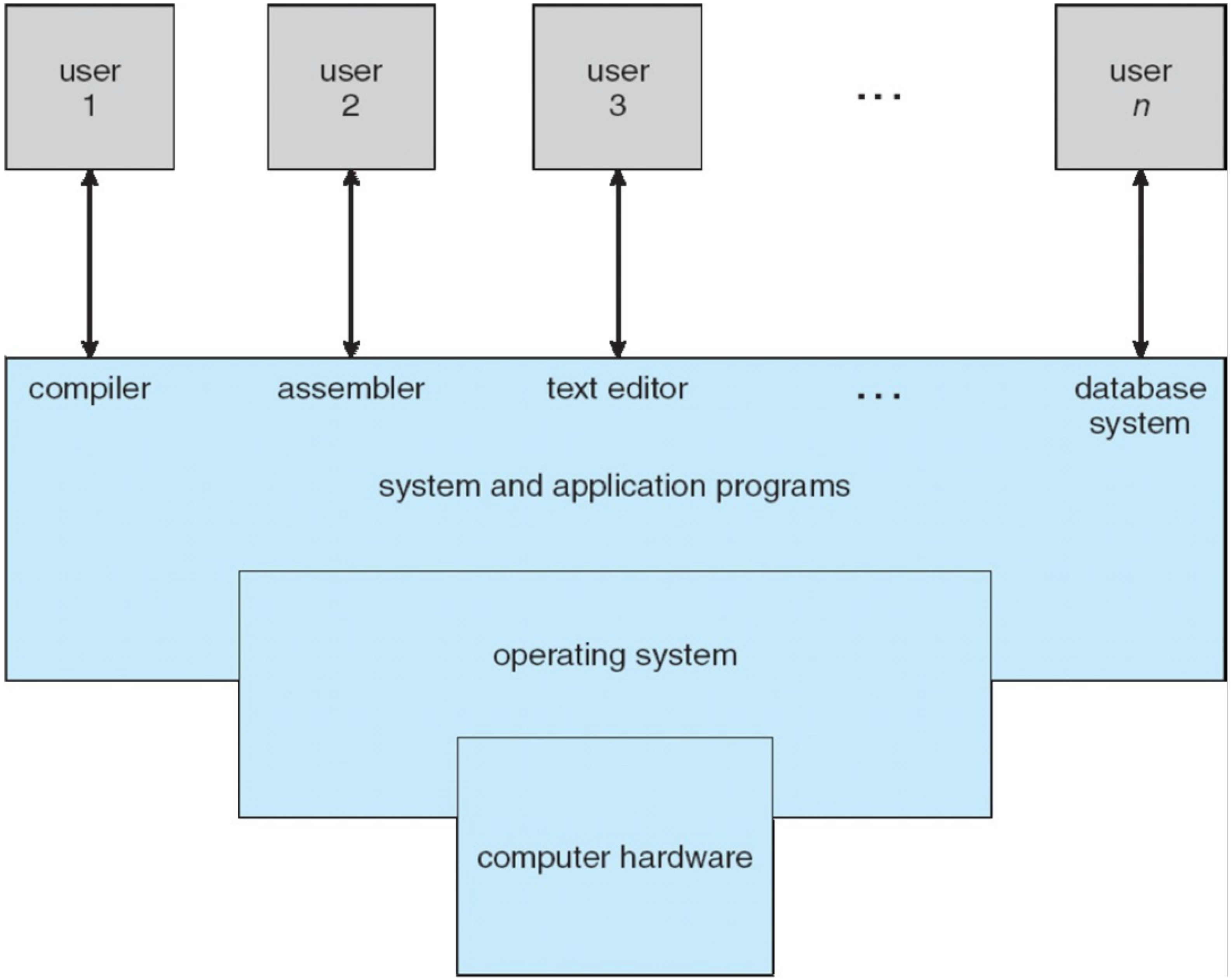
Zhejiang University

Acknowledgement: Some slides are based on Zhi Wang (FSU)'s slides.

Four Components of a Computer System

- Computer system has four components:
  - **hardware** provides basic computing resources
    - e.g., CPU, memory, I/O devices
  - **operating system** controls and coordinates use of hardware among users
  - **application** programs use system resources to solve computing problems
    - e.g., word processors, compilers, web browsers….
  - **users**
    - e.g., people, machines, other computers

Four Components of a Computer System



What Operating Systems Do

- User View vs System View
- Users want convenience, ease of use
  - **don't** care much about resource utilization
- Shared computers (e.g., mainframe) must keep all users happy
  - users of dedicate systems frequently use shared resources from servers
  - e.g., gmail, google doc…
- Handhold devices are resource constrained, optimized for usability and battery life
  - e.g., smartphones, tablets
- Some computers have little or no user interface
  - e.g., embedded computers in devices and automobiles

What Operating Systems Do

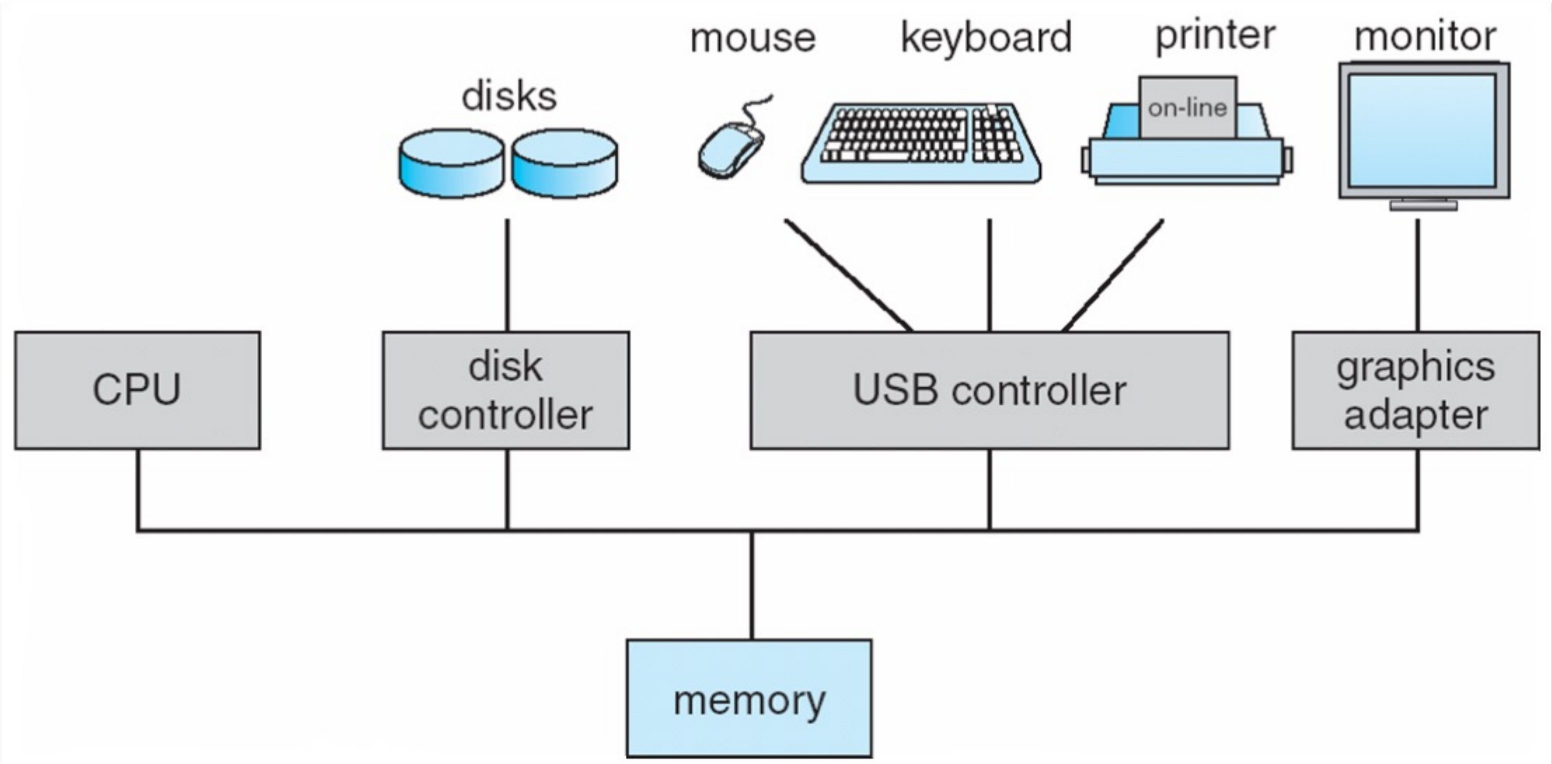
- OS is a **resource allocator**
  - it manages all resources
  - it decides between conflicting requests for efficient and fair resource sharing
- OS is a **control program**
  - it controls program execution to prevent **errors** and **improper use** of system

Operating System Definition

- A good approximation is "**everything a vendor ships when you order an operating system**"
  - no universally accepted definition
  - what the vendor ships can vary wildly
- Kernel is "**the one program running at all times on the computer**"
  - what about demon programs that starts with the kernel such as init?
- Everything else is either a **system program** or an **application program**
- Operating system may have different meanings in different contexts
  - Is WeChat an operating system?

Hardware Components

- CPUs & device controllers connect through buses to share memory
- Concurrent execution of CPUs & devices compete for memory cycles

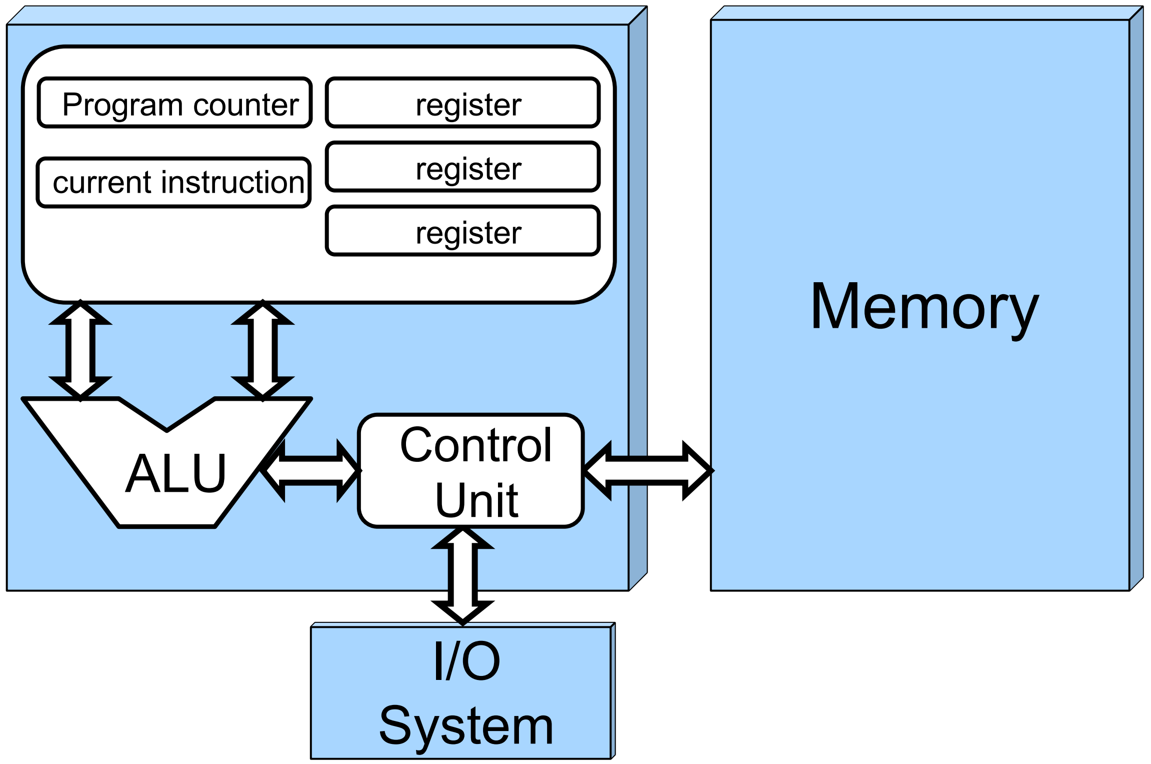


Devices

- Each **device controller** is in charge of a particular device type
  - disk controller, USB controller…
- Each device controller has a local buffer
  - I/O: between the device and local buffer of the controller
  - CPU moves data between main memory and controller buffers
- I/O devices and the CPU can execute concurrently
  - DMA (direct memory access)
  - device controller informs CPU that it has finished its operation by causing an interrupt

Review

- Course information and projects
- Von-Neumann mode: cpu, memory, I/O
- CPU: fetch-decode-execute
- Optimizations: cache, DMA



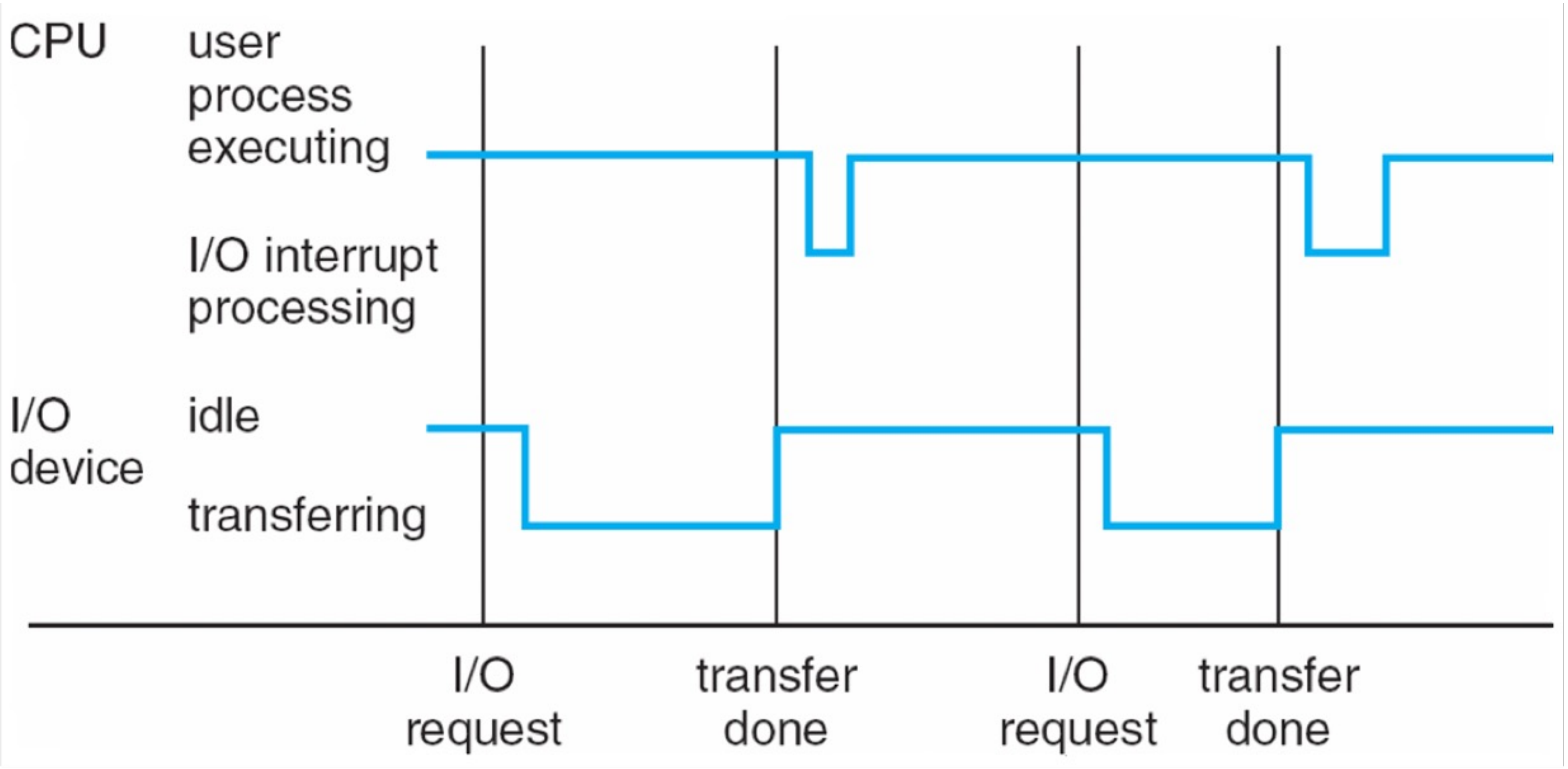
Interrupts and Traps

- Interrupt transfers control to the interrupt service routine
  - **interrupt vector:** a table containing addresses of all the service routines
  - incoming interrupts are disabled while serving another interrupt to prevent a lost interrupt
  - **interrupt handler** must save the (interrupted) execution states
- A **trap** is a software-generated interrupt, caused either by an error or a user request
  - an **interrupt** is asynchronous; a **trap** is synchronous
  - e.g., **system call**, divided-by-zero exception, general protection exception…
- Operating systems are usually **interrupt-driven**

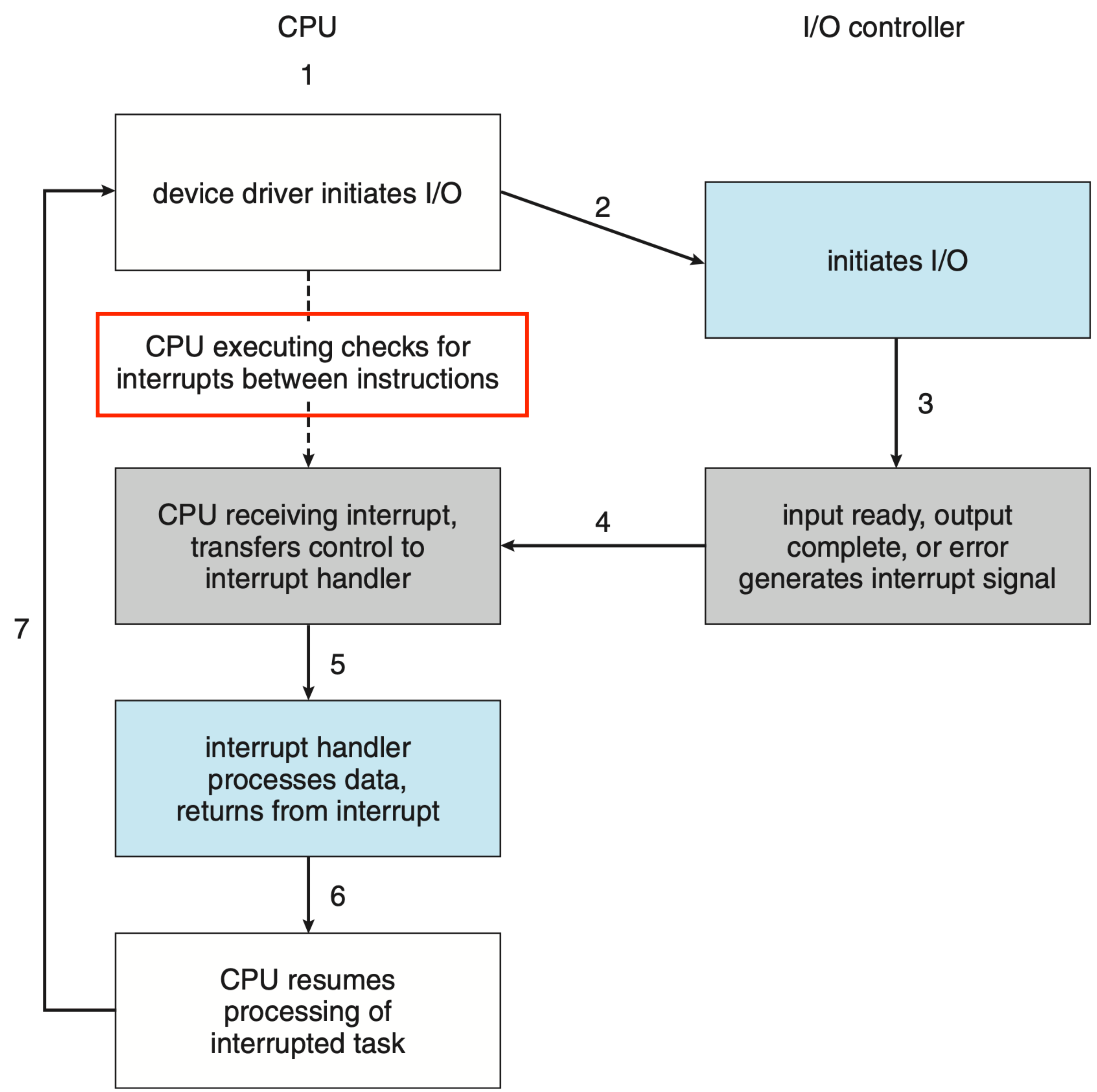
Interrupt Handling

- Operating system preserves the execution state of the CPU
  - save registers and the program counter (PC)
- OS determines which device caused the interrupt
  - **polling**
  - **vectored** interrupt system
- OS handles the interrupt by calling the device's driver
- OS restores the CPU execution to the saved state

Interrupt Timeline



Interrupt-drive I/O Cycle



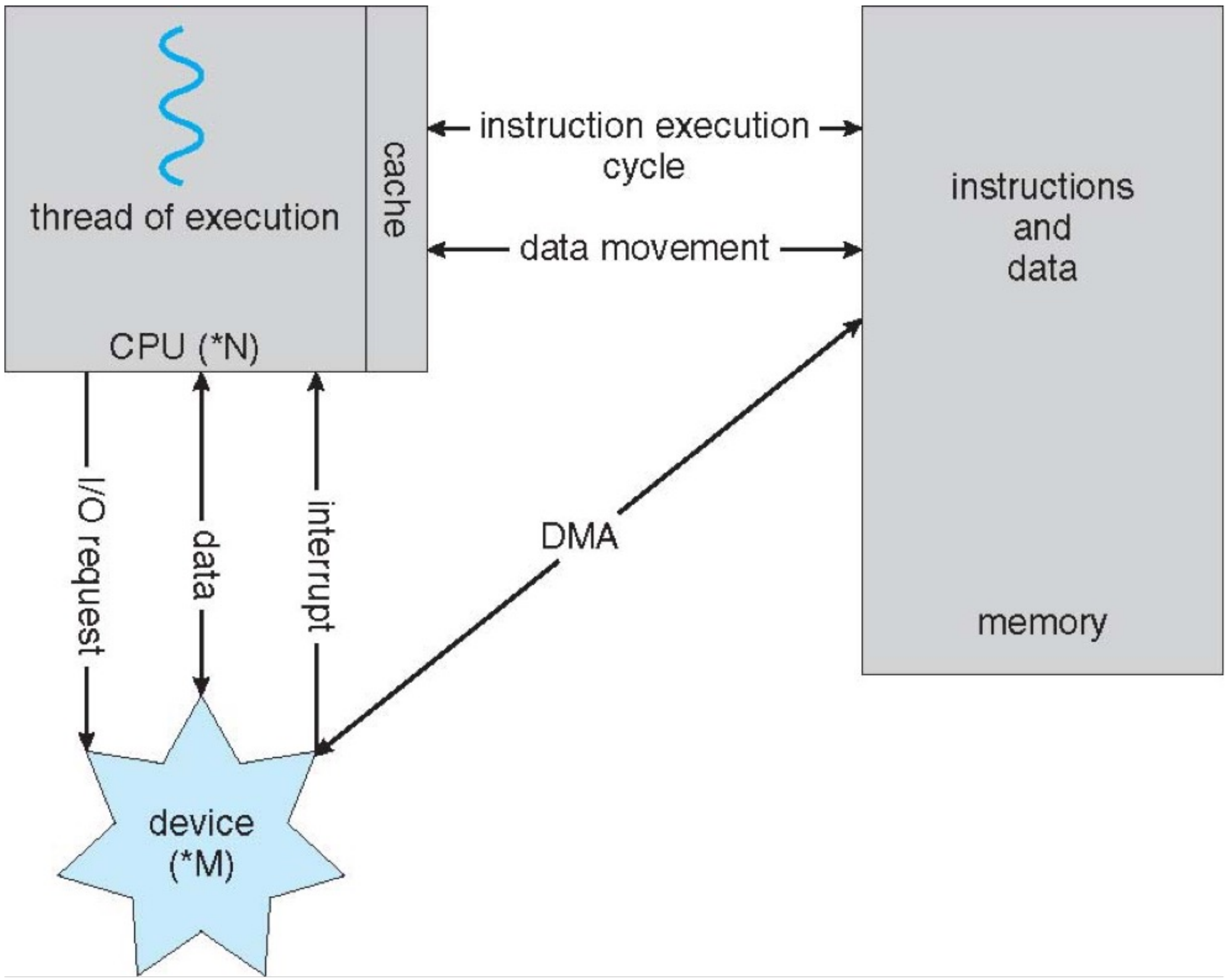
I/O: from System Call to Devices, and Back

- A program uses a **system call** to access system resources
  - e.g., files, network
- Operating system converts it to device access and issues I/O requests
  - I/O requests are sent to the device driver, then to the controller
  - e.g., read disk blocks, send/receive packets…
- OS puts the program to wait (**synchronous I/O**) or returns to it without waiting (**asynchronous I/O**)
  - OS may switches to another program when the requester is waiting
- I/O completes and the controller interrupts the OS
- OS processes the I/O, and then wakes up the program (synchronous I/O) or send its a signal (asynchronous I/O)

Direct Memory Access

- DMA is used for high-speed I/O devices able to transmit information at close to memory speeds
    - e.g., Ethernet, hard disk, cd rom…
- Device driver sends an I/O descriptor the controller
    - **I/O descriptor**: operation type (e.g., send/receive), memory address…
- The controller transfers blocks of data between its local buffer and main memory **without CPU intervention**
    - only one interrupt is generated when whole I/O request completes

Put it Together



Storage Structure

- Main memory: the only large storage that CPU can directly access
  - **random access**, and typically **volatile**
- Secondary storage: large **nonvolatile** storage capacity
  - Magnetic disks are most common second-storage devices (HDD)
    - rigid metal or glass platters covered with magnetic recording material
    - disk surface is logically divided into **tracks** and **sectors**
    - disk controller determines the interaction between OS and the device

Storage Hierarchy

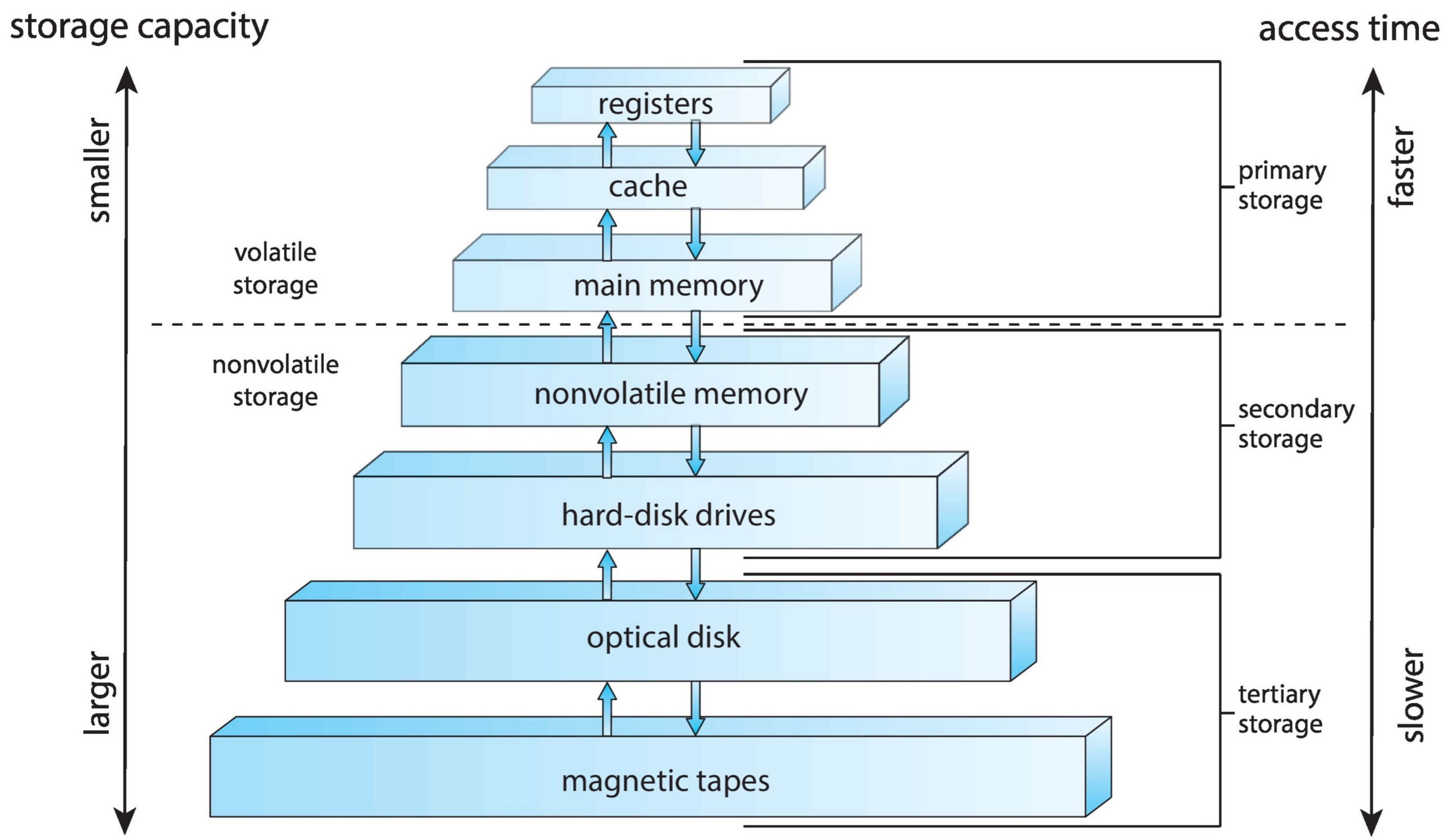
- Storage systems can be organized in hierarchy
  - speed
  - cost
  - volatility

Review

The basic unit of computer storage is the **bit** . A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes. A **kilobyte** , or KB , is 1,024 bytes; a **megabyte** , or **MB** , is $1{,}024^2$ bytes; a **gigabyte** , or GB , is $1{,}024^3$ bytes; a **terabyte** , or **TB** , is $1{,}024^4$ bytes; and a **petabyte** , or **PB** , is $1{,}024^5$ bytes. Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

Storage Hierarchy

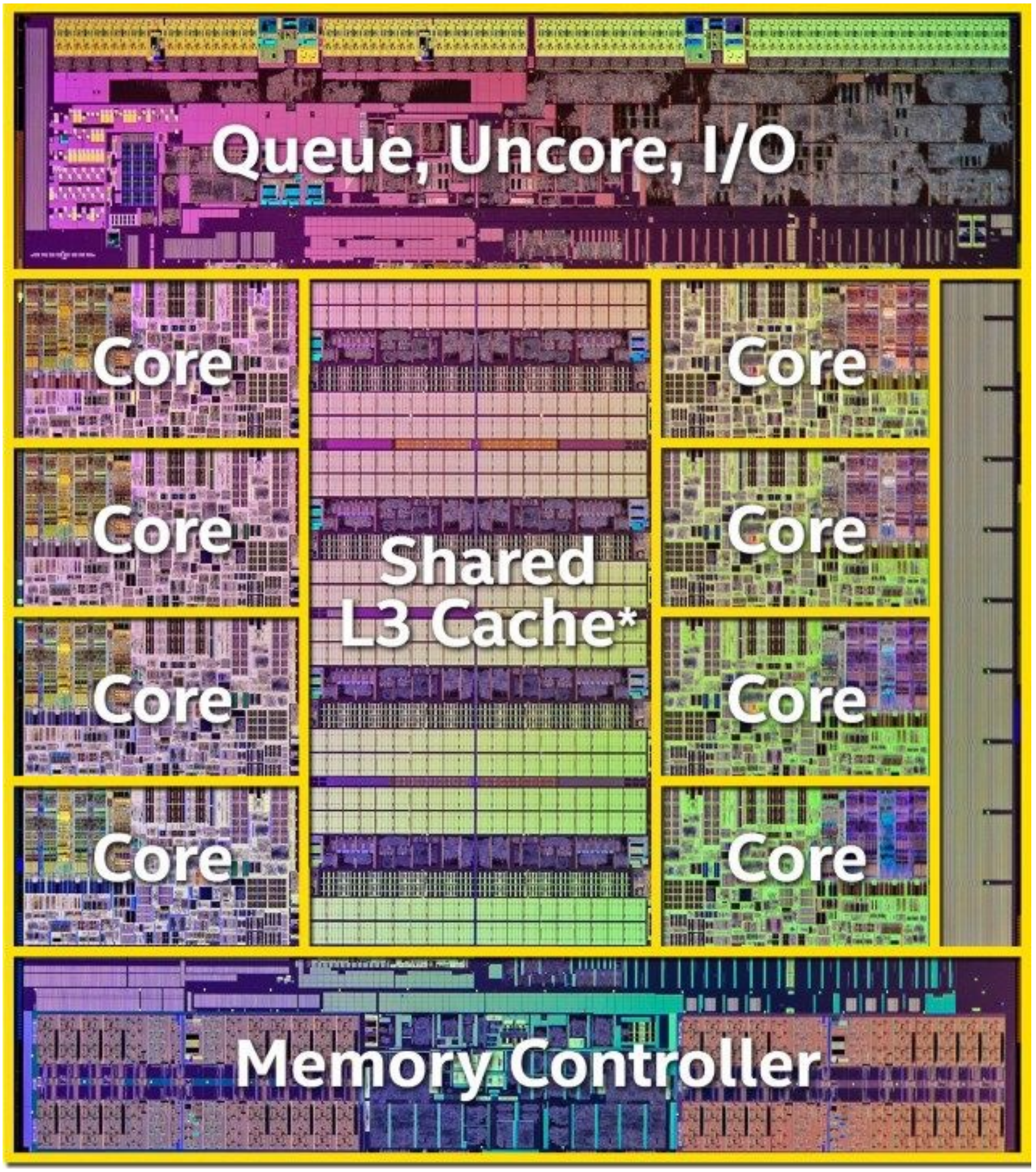


- **Caching**: copying information into faster storage system
  - main memory can be viewed as a cache for secondary storage
  - CPU has a cache for main memory

Performance of Storages

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

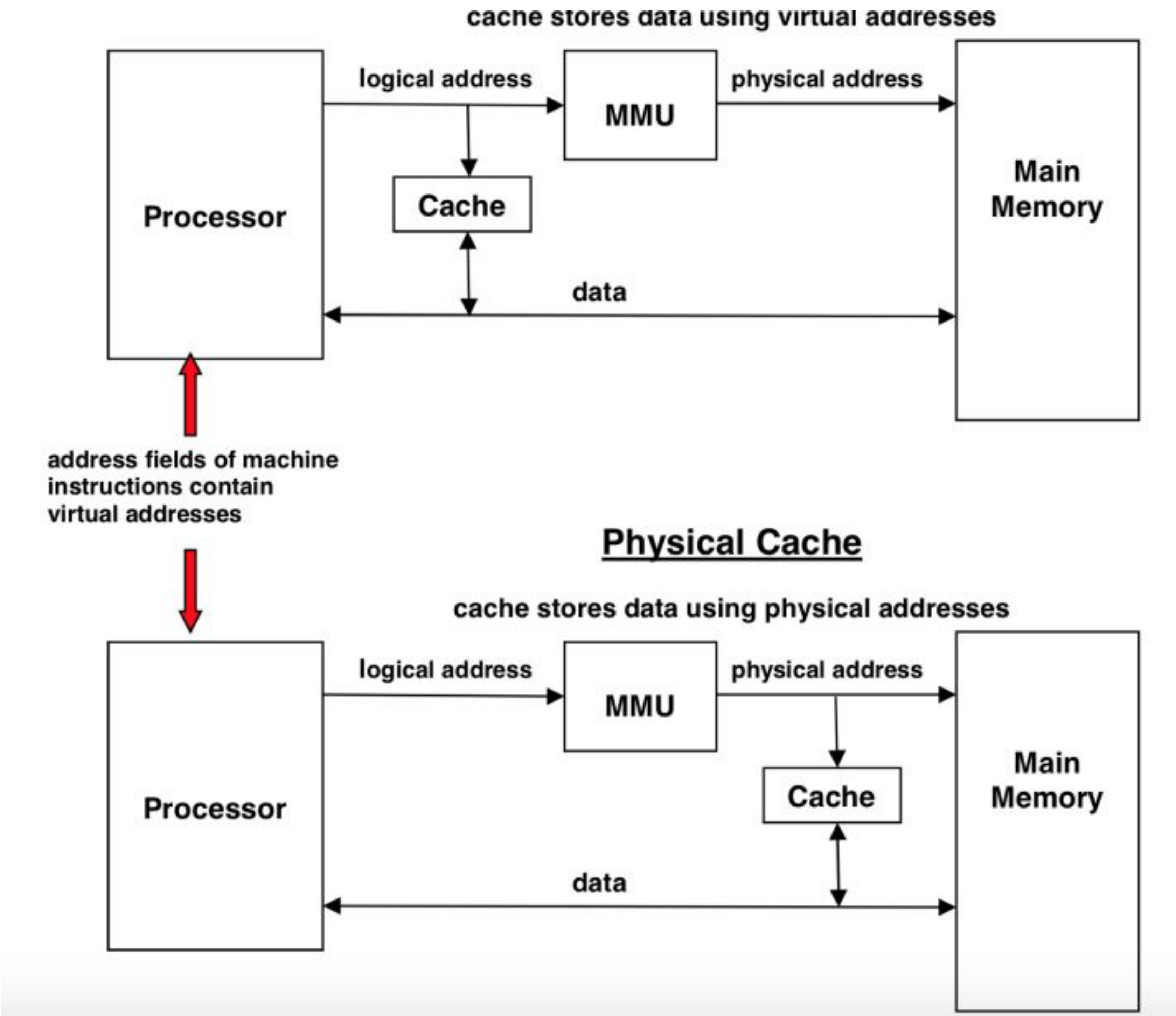
Shared L3 Cache



Caching

- Caching is an important principle, performed at many levels
  - e.g., in hardware, operating system, user program…
- **Caching**: data in use copied from slower to faster storage temporarily
  - faster storage (cache) is checked first to determine if data is there
    - if it is, data is used directly from the cache (fast)
    - if not, data is first copied to cache and used there
- Cache is usually smaller than storage being cached
- Cache management is an important design problem
  - e.g., cache size and replacement policy

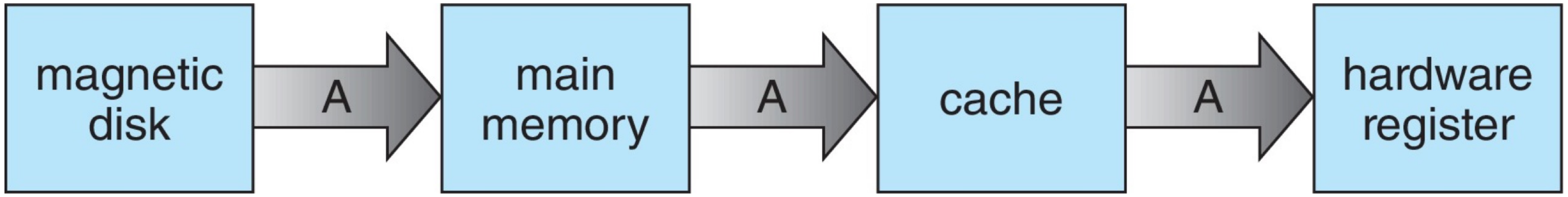
Virtual Cache vs Physical Cache



Caching

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy

- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache



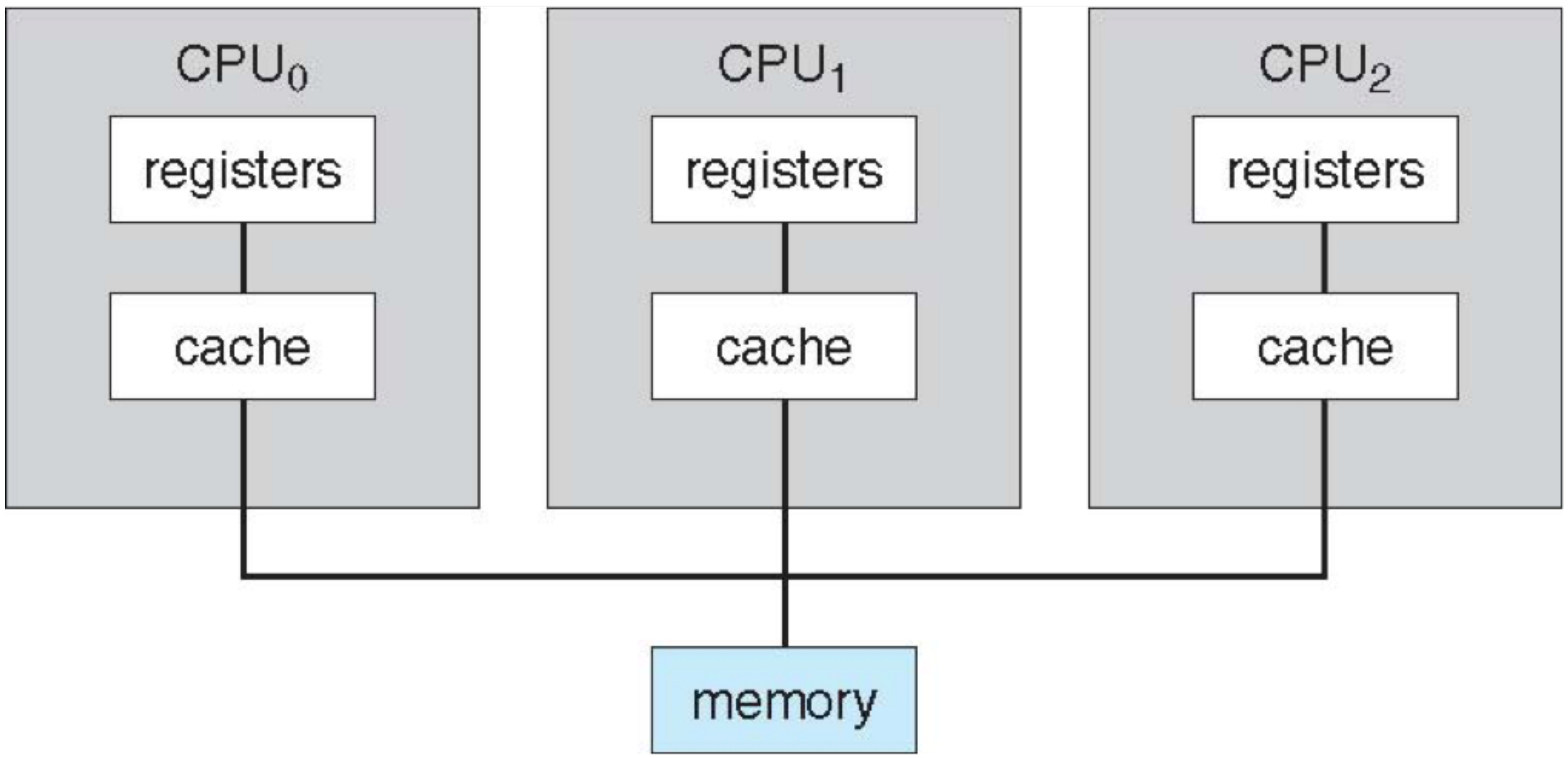
Computer-System Architecture

- Categorized according to the number of general-purpose processors
    - Single-Processor
    - Multiple-Processor

Multiprocessor Systems

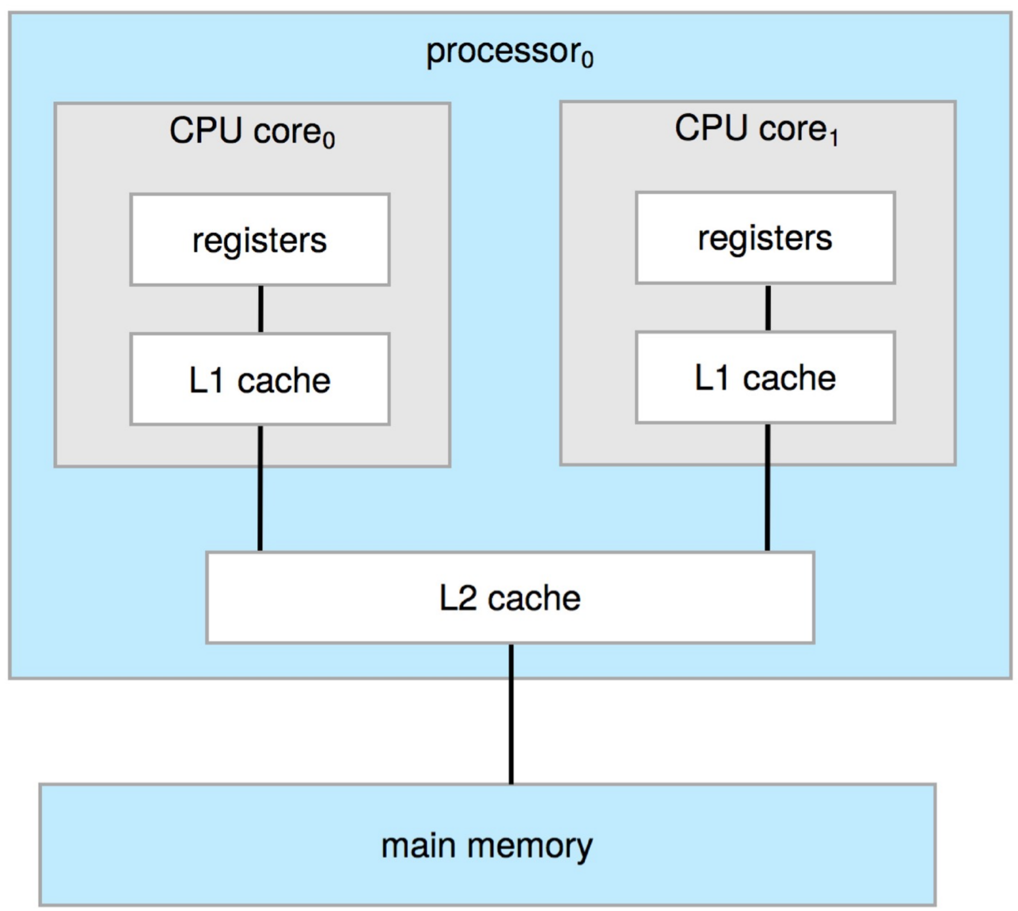
- Most **old systems** have one single general-purpose processor
  - e.g., smartphone, PC, server, mainframe
  - most systems also have special-purpose processors as well
- Multiprocessor systems have grown in use and importance
  - also known as parallel systems, tightly-coupled systems
  - advantages: increased throughput, economy of scale, increased reliability -- graceful degradation or fault tolerance
  - two types: **asymmetric multiprocessing** and **symmetric multiprocessing (SMP)**

Symmetric Multiprocessing Architecture



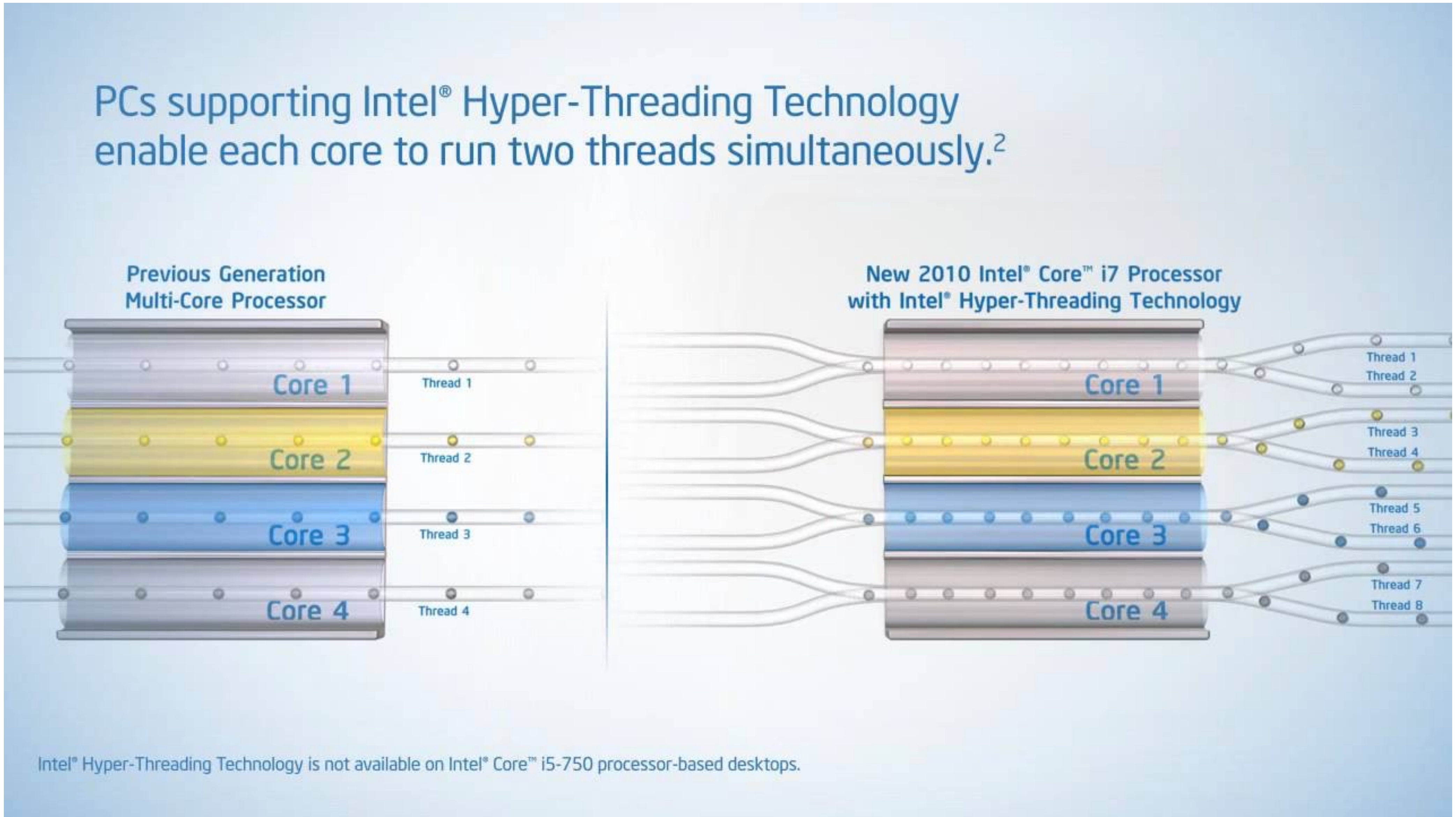
A Dual-Core Design

- Multiple CPU Cores in a single chip
    - Shared local cache/memory -> what happens when the number of cores increases?



Multi Core vs Hyper Threading

- Two programs can use one execution unit (inside one core) at the same time
- The performance depends on OS, compiler, application



Multi Core vs Hyper Threading

Hyper-Threading (HT) Technology

- Provides more satisfactory solution
- Single physical processor is shared as two logical processors
- Each logical processor has its own architecture state
- Single set of execution units are shared between logical processors
- N-logical PUs are supported
- Have the same gain % with only 5% die-size penalty.
- **HT allows single processor to fetch and execute two separate code streams simultaneously.**

Figure 2: Processors without Hyper-Threading Tech

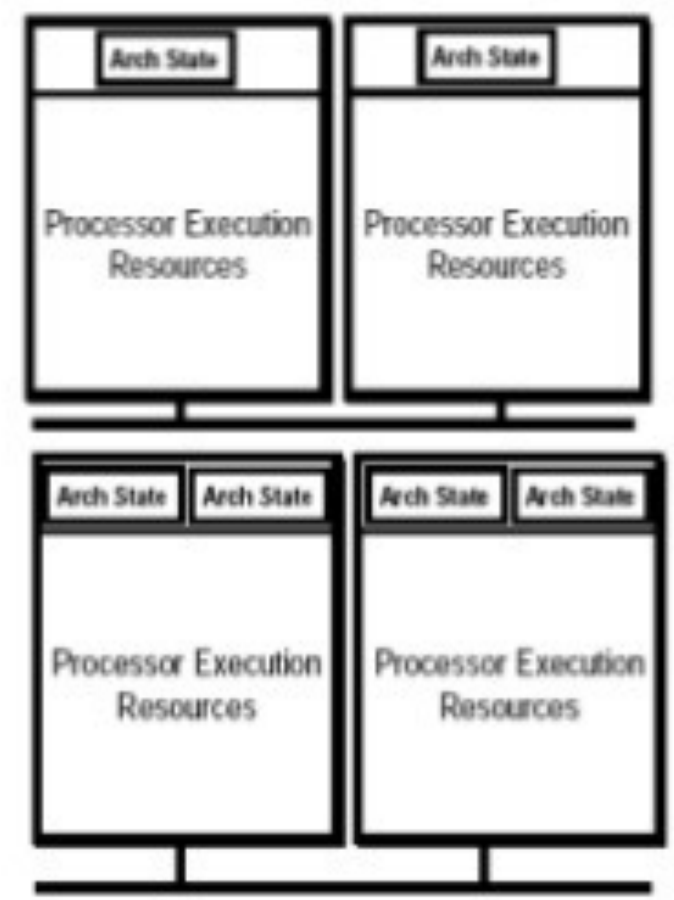
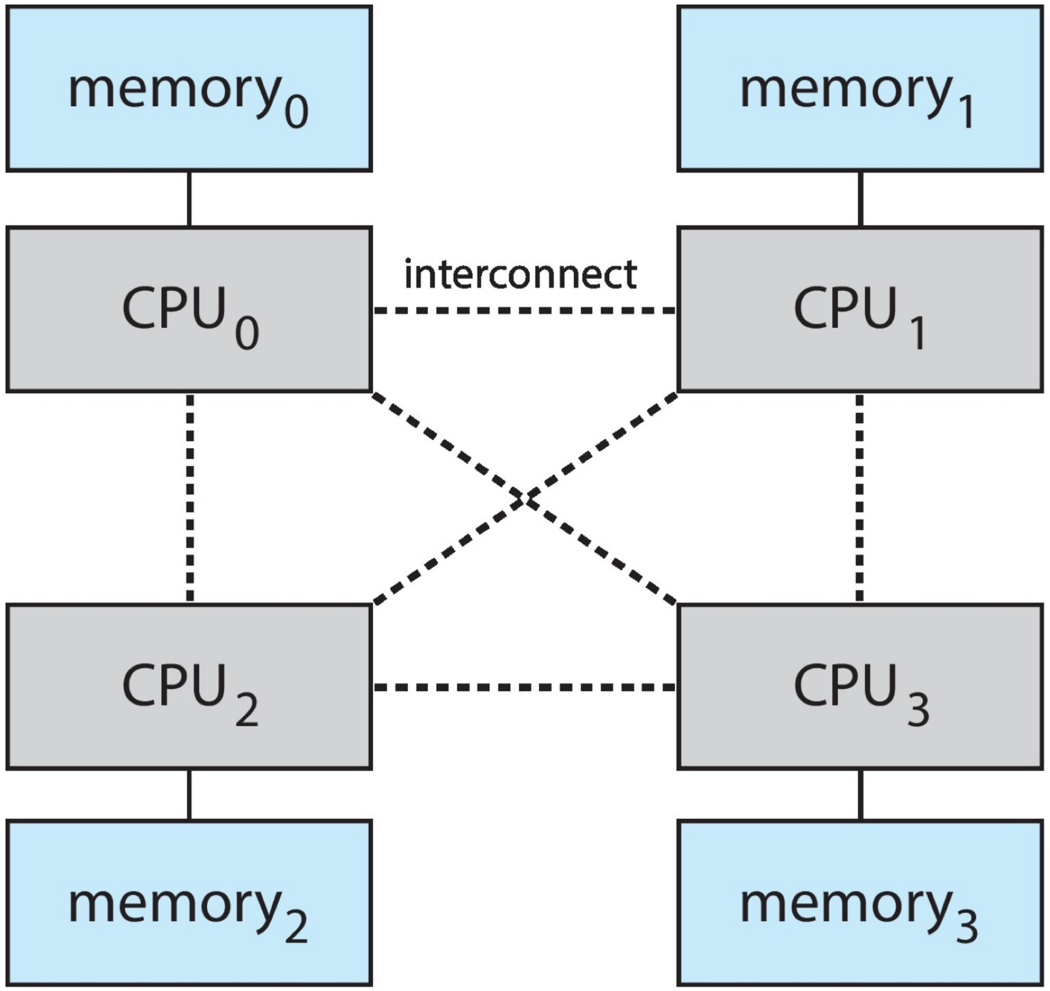


Figure 3: Processors with Hyper-Threading Technology

NUMA

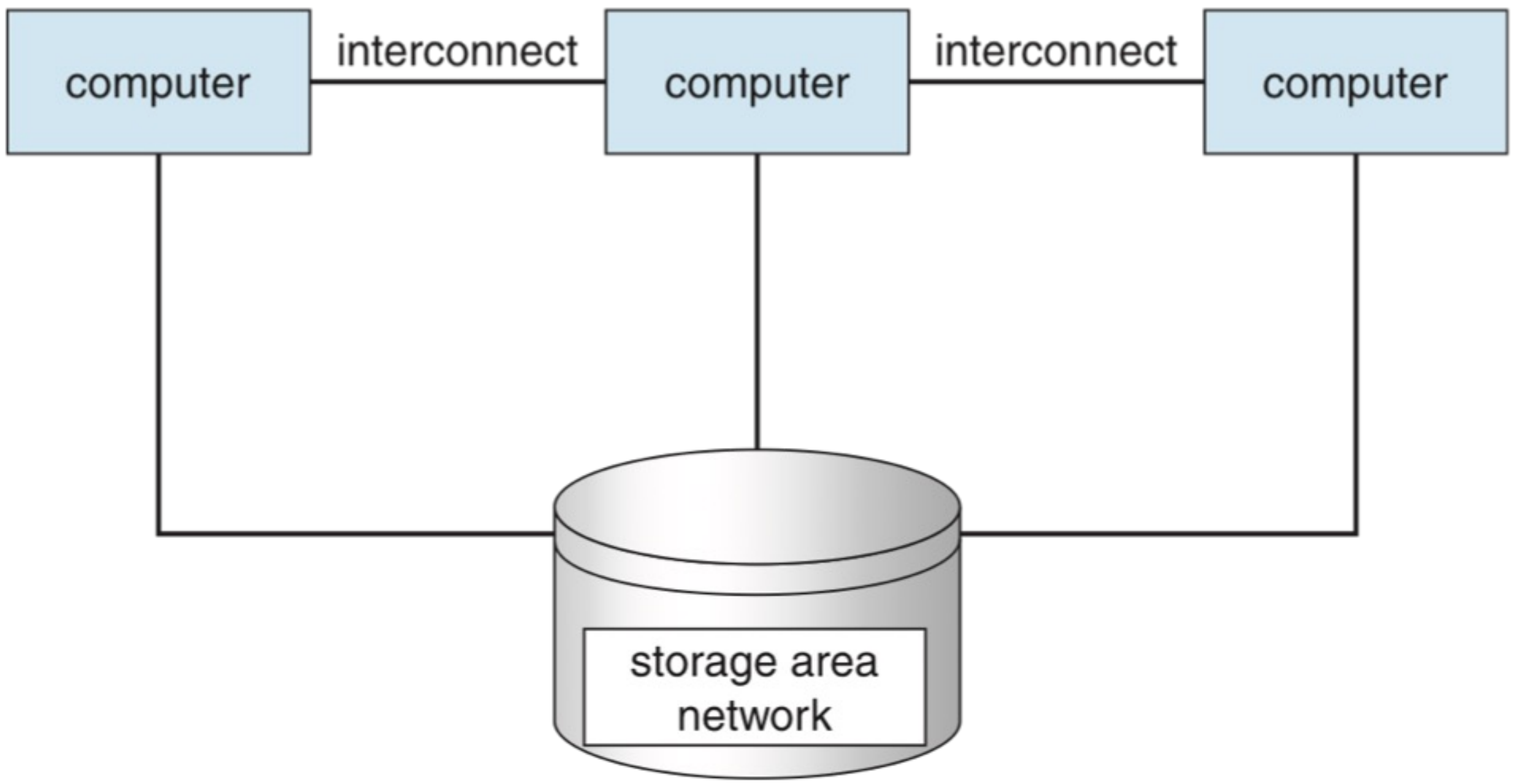
- Non-Uniform Memory Access System
    - Access local memory is fast, scale well



Clustered Systems

- Multiple systems work together ***through high-speed network***
  - usually sharing storage via a storage-area network (SAN)
- Clusters provide a high-availability service that can survive failures
  - **asymmetric** clustering has one machine in hot-standby mode
  - **symmetric** clustering has multiple nodes running applications, monitoring each other
- Some clusters are designed for high-performance computing (HPC)
  - applications must be written to use parallelization

Clustered Systems



Distributed Systems

- A collection of separate, possibly heterogeneous, systems inter-connected through networks

- **Network OS** allows systems to exchange messages

- A **distributed system** creates the illusion of a single system

Special-Purpose Systems

- Real-time embedded systems most prevalent form of computers
  - vary considerably
  - use special purpose (limited purpose) real-time OS
- Multimedia systems
  - streams of data must be delivered according to time restrictions
- Handheld systems
  - e.g., PDAs, smart phones
  - limited CPU (?), memory(?), and power
  - used to use reduced feature OS (?)

Peer-to-Peer Computing

- Another model of distributed system
- P2P does not distinguish clients and servers
  - instead all nodes are considered peers
  - may each act as client, server or both
- A node must join P2P network
  - registers its service with central lookup service, or
  - broadcast request for and respond to service via a discovery protocol
- Examples include BitTorrent, Napster and Gnutella and **Blockchain** platforms

Operating System Operations: Multiprogramming

- **Multiprogramming** is necessary for efficiency
  - single user cannot keep CPU and I/O devices busy at all times
  - user's computing tasks are organized as jobs (code and data)
  - kernel schedules jobs (job scheduling) so CPU always has things to do
    - a subset of total jobs in system is kept in memory
    - when a job **has to wait** (e.g., for I/O), kernel switches to another job
    - What are the problems here?

Operating System Operations: multitasking

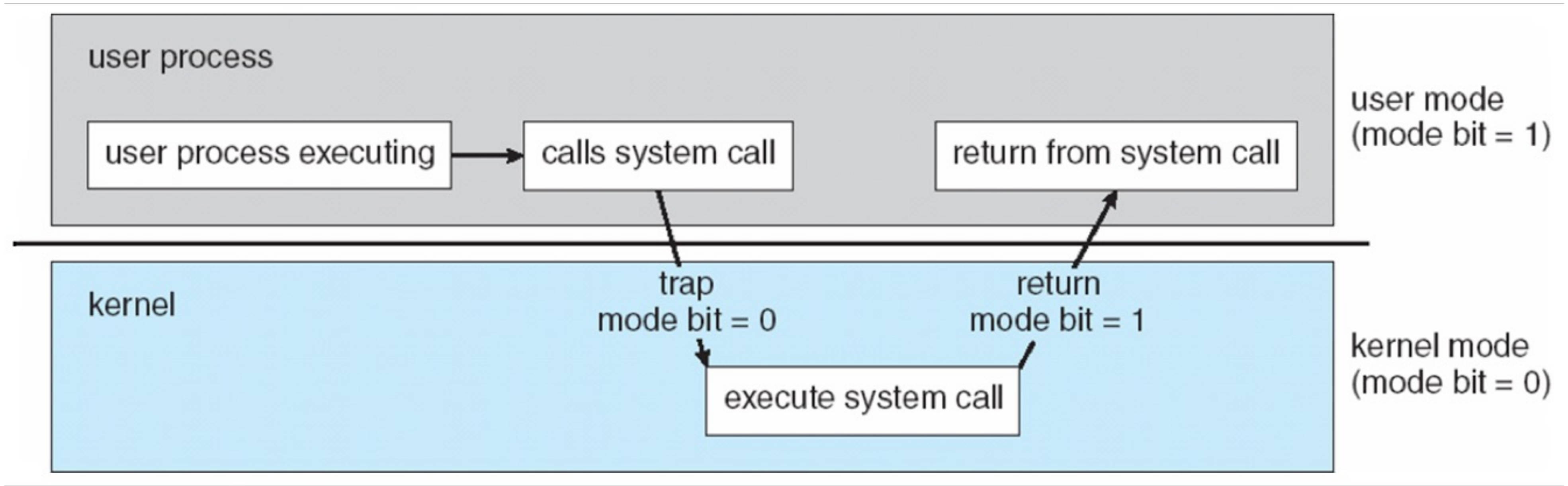
- **Timesharing** (multitasking) extends the multiprogramming
  - OS switches jobs **so frequently** that users can interact with each running job
  - response time should be < 1s
  - each user has at least one program executing in memory (**process**)
  - if several jobs ready to run at the same time (**CPU scheduling**)
  - It makes the programmer easier: virtual/physical memory
  - What are the problems/challenges here?

Dual-mode operation

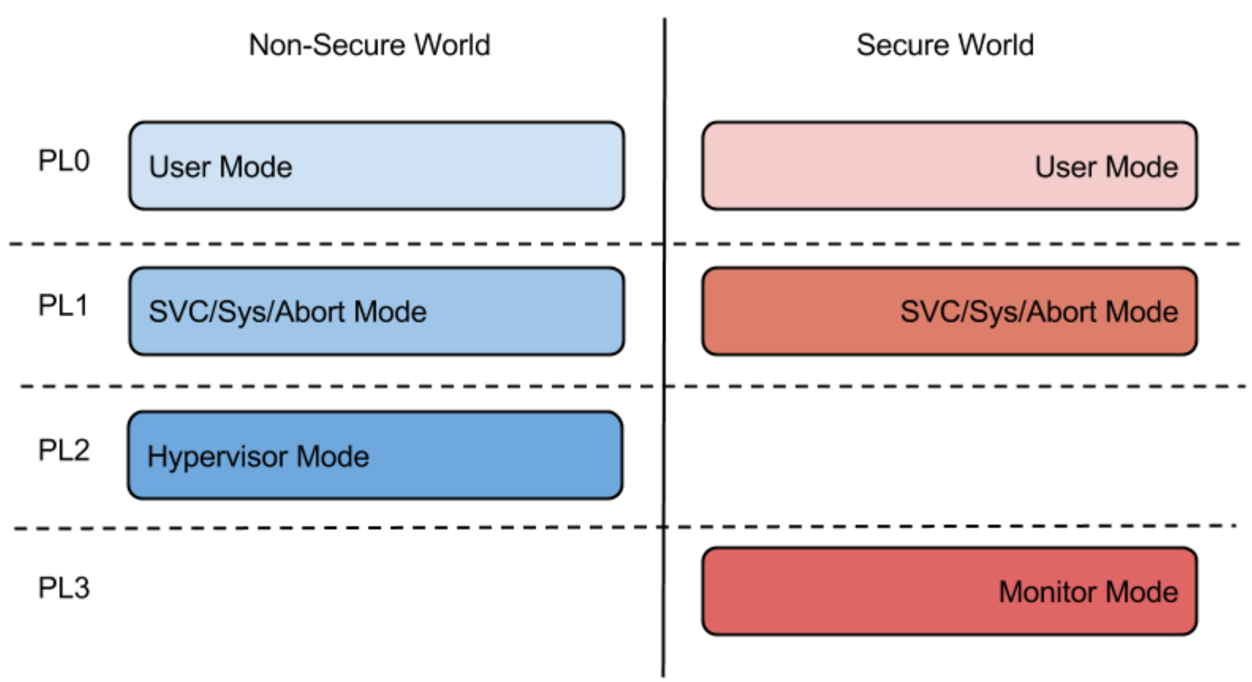
- Operating system is usually interrupt-driven
  - Efficiency, regain control (timer interrupt)
- **Dual-mode operation** allows OS to protect itself and other system components
  - **user mode** and **kernel mode (or other names)**
  - a **mode** bit distinguishes when CPU is running user code or kernel code
  - some instructions designated as **privileged**, only executable in kernel
  - **system call** changes mode to kernel, return from call resets it to user

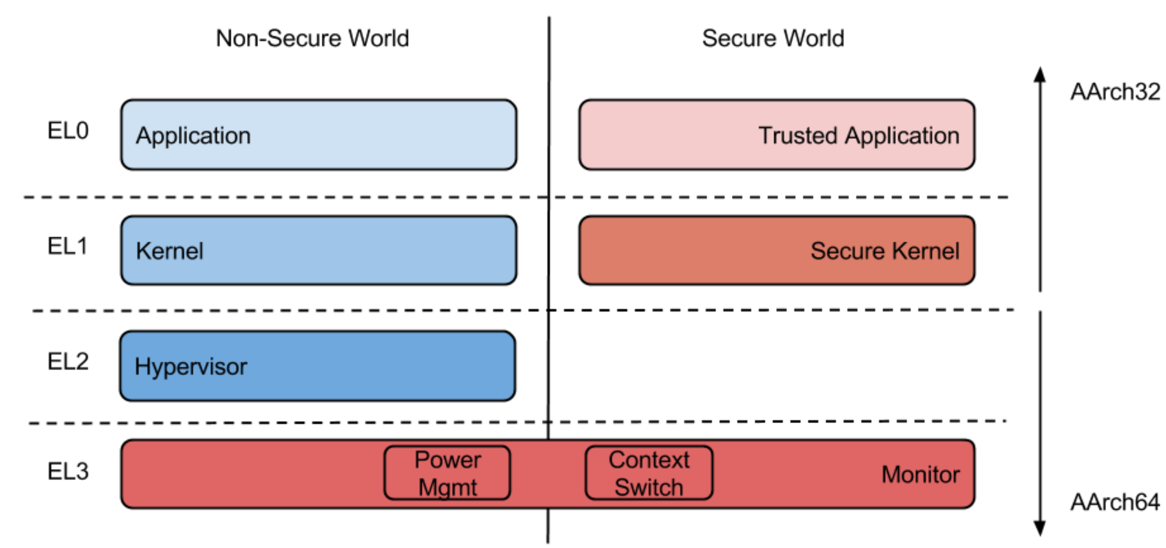
Transition between Modes

- **System calls, exceptions, interrupts** cause transitions between kernel/user modes



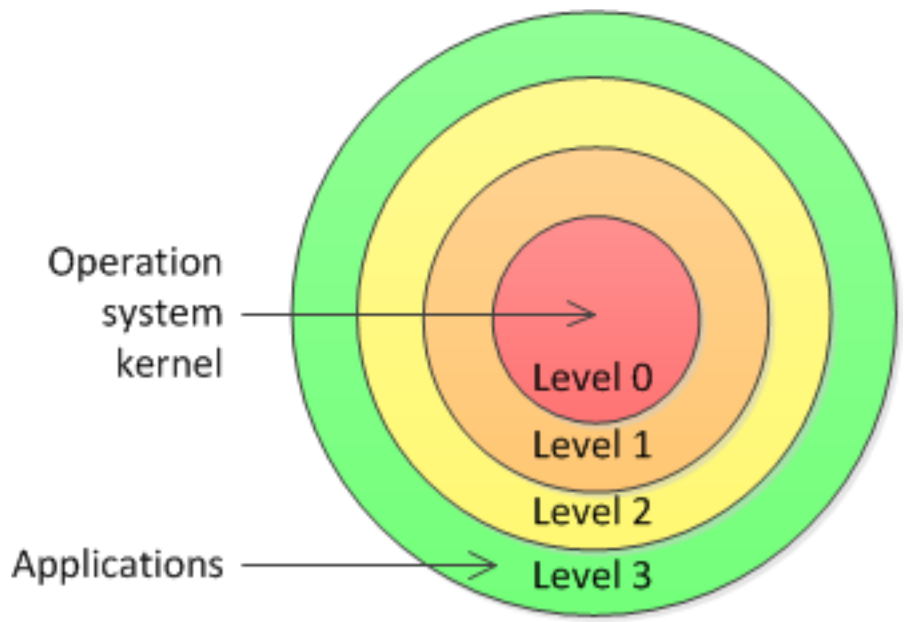
Examples: ARM





Figures: http://dafulv.blogspot.com/2014/06/arm-trustzone-powered-mobile-security.html

Examples: X86



Figures: https://zhihuicao.wordpress.com/2015/06/15/privilege-level-in-x86-architecture-instructions/

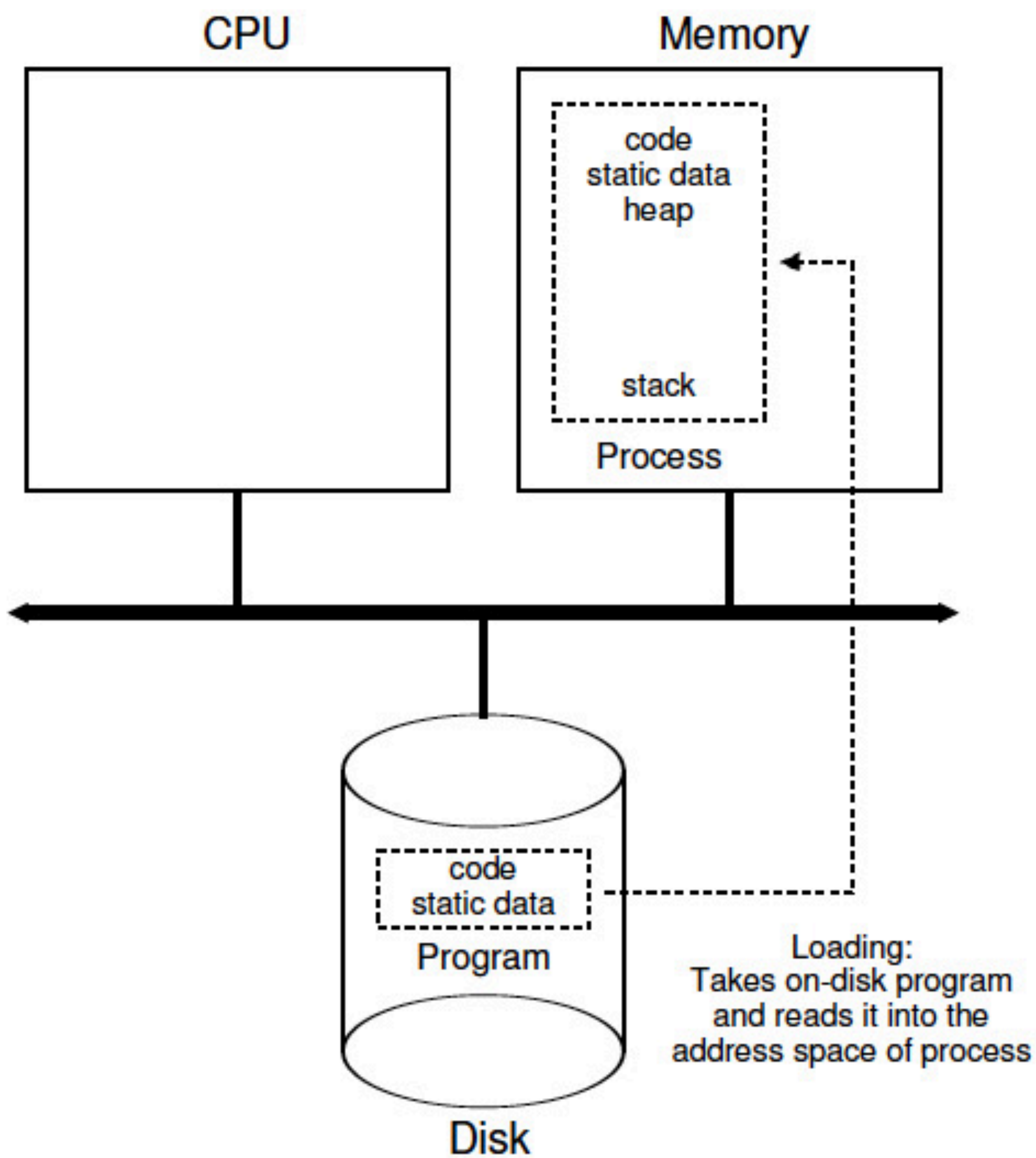
Timer

- Timer used to prevent infinite loop or process hogging resources
  - to enable a timer, set the hardware to interrupt after some period
  - OS sets up a timer before scheduling process to regain control
    - the timer for scheduling is usually **periodical** (e.g., 250HZ)
    - **tickless kernel**: on-demand timer interrupts(Linux)

Resource Management: Process Management

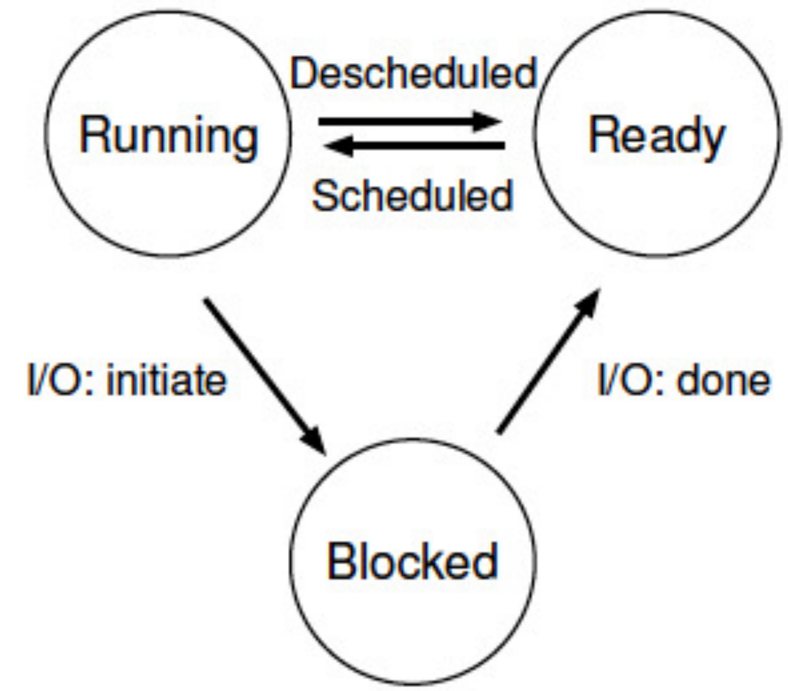
- A process is **a program in execution**
    - program is a ***passive*** entity, process is an ***active*** entity
    - a system has many processes running concurrently
- Process needs resources to accomplish its task
    - OS reclaims all reusable resources upon process termination
    - e.g., CPU, memory, I/O, files, initialization data

From Program to Process



Process Management Activities

- Process creation and termination
- Processes suspension and resumption
- Process synchronization primitives
- Process communication primitives
- Deadlock handling



From Process to Thread

- Single-threaded process has one program counter
  - **program counter** specifies *location of next instruction to execute*
  - processor executes instructions sequentially, one at a time, until completion
- Multi-threaded process has **one program counter per thread**
- Quiz: What are the benefits of using thread instead of process?

轻量级：线程通常比进程更轻量级，因为它们共享父进程的内存空间和资源。创建线程所需的系统资源较少，与创建新进程相比更为高效。

高效通信：同一进程中的线程可以通过共享内存轻松地进行通信。它们可以直接访问和修改共享数据，无需使用复杂的进程间通信机制，如管道或消息队列。

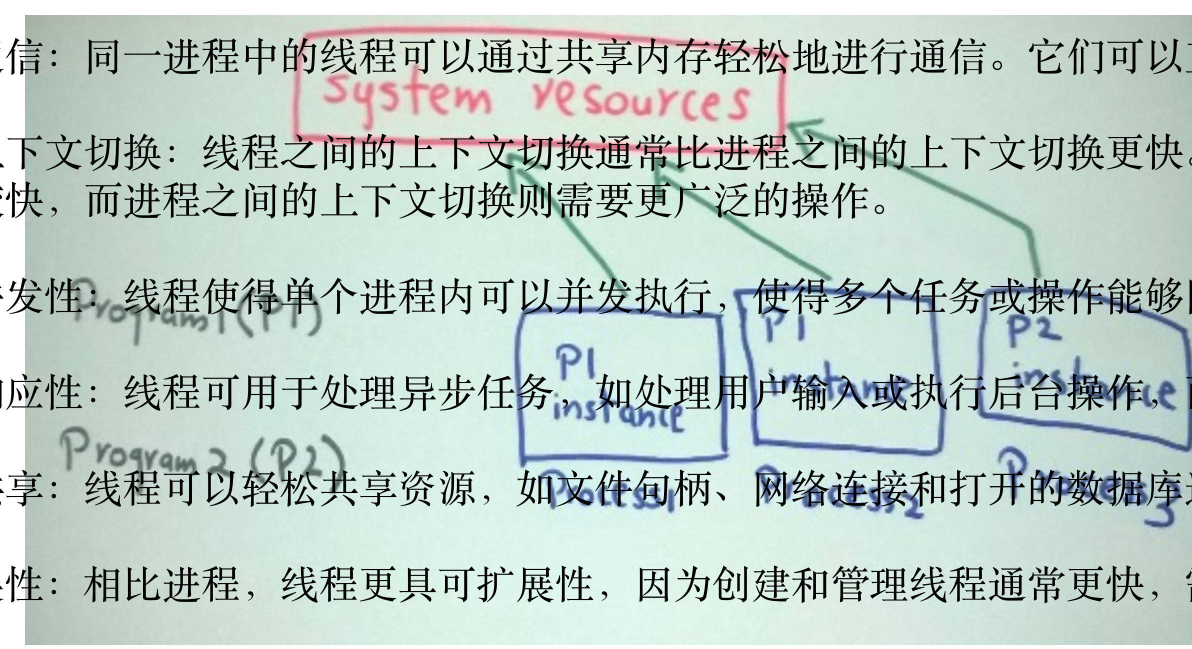
快速上下文切换：线程之间的上下文切换通常比进程之间的上下文切换更快。由于线程共享同一内存空间，切换线程只需改变程序计数器和栈指针，这些操作相对较快，而进程之间的上下文切换则需要更广泛的操作。

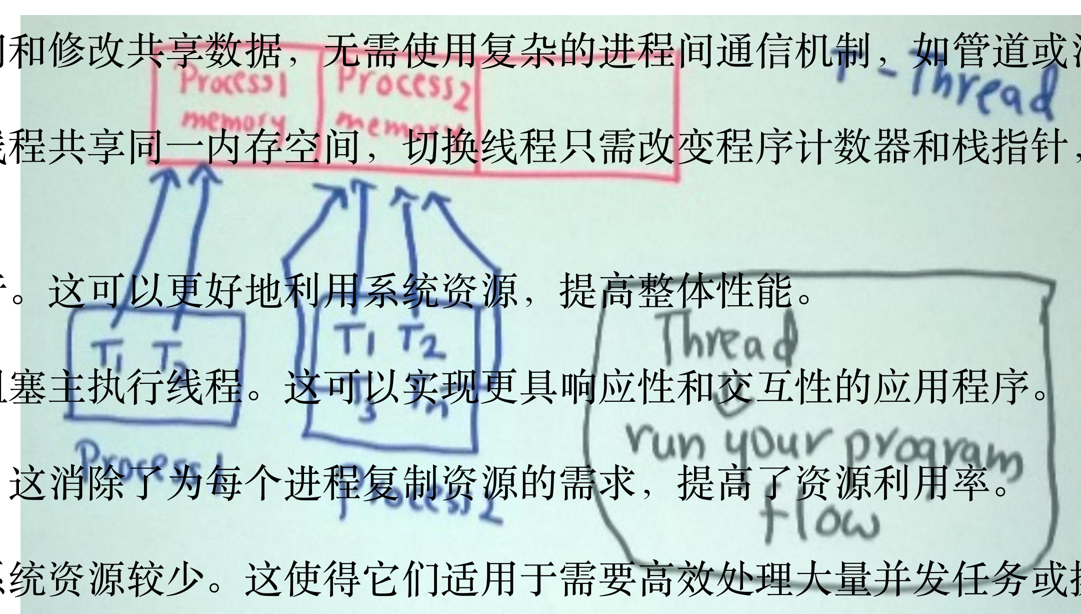
改善并发性：线程使得单个进程内可以并发执行，使得多个任务或操作能够同时进行。这可以更好地利用系统资源，提高整体性能。

提高响应性：线程可用于处理异步任务，如处理用户输入或执行后台操作，而不会阻塞主执行线程。这可以实现更具响应性和交互性的应用程序。

资源共享：线程可以轻松共享资源，如文件句柄、网络连接和打开的数据库连接等。这消除了为每个进程复制资源的需求，提高了资源利用率。

可扩展性：相比进程，线程更具可扩展性，因为创建和管理线程通常更快，需要的系统资源较少。这使得它们适用于需要高效处理大量并发任务或操作的场景。





Figures: https://medium.com/coderscorner/threads-vs-processes-a0b199f56c2d

Resource Management: Memory Management

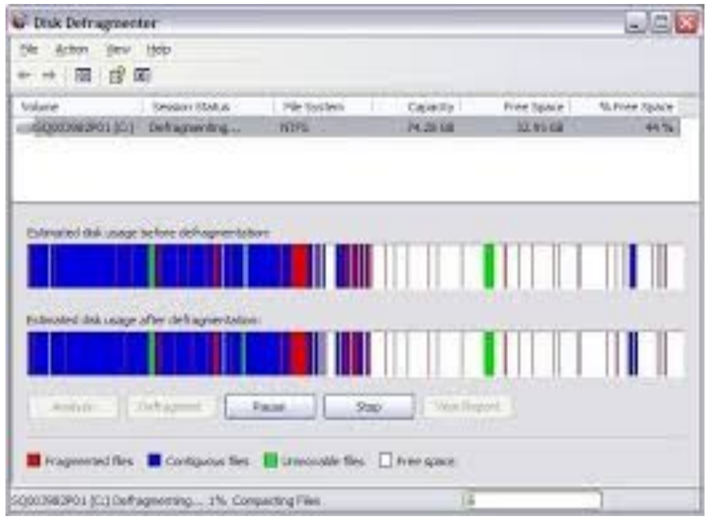
- Memory is the main storage directly accessible to CPU
  - data needs to be kept in memory before and after processing
  - all instructions should be in memory in order to execute
- Memory management determines what is in memory to **optimize CPU utilization** and **response time, provides a virtual view of memory for programmer**
- Memory management activities:
  - keeping track of which parts of memory are being used and by whom
  - deciding which processes and data to move into and out of memory
  - allocating and deallocating memory space as needed

Resource Management: File Systems

- OS provides a uniform, logical view of data storage
  - **file** is a logical storage unit that abstracts physical properties
    - files are usually organized into **directories**
    - **access control** determines who can access the file
- File system management activities:
  - creating and deleting files and directories
  - primitives to manipulate files and directories
  - mapping files onto secondary storage
  - backup files onto stable (non-volatile) storage media

Resource Management: Mass-Storage Management

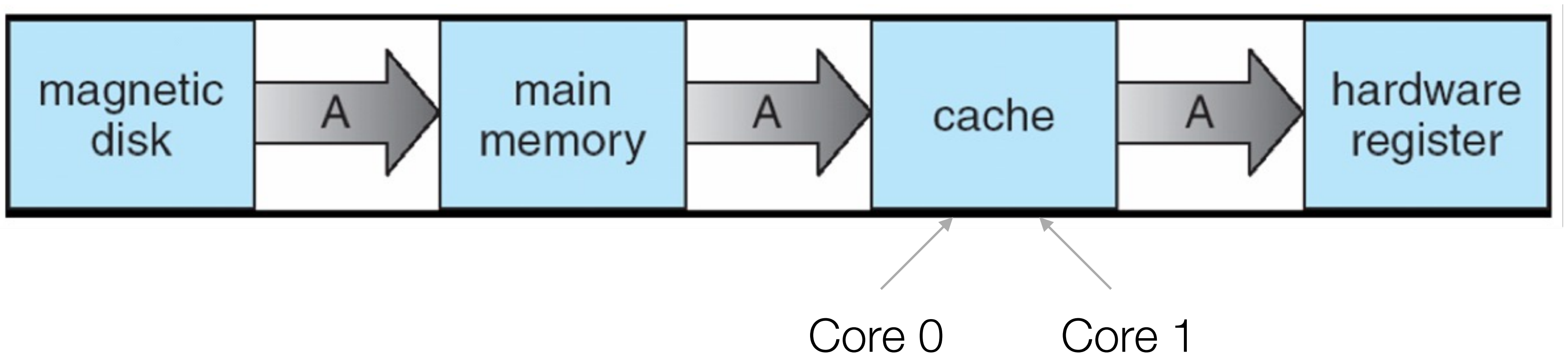
- Disk subsystem manages mass storages
  - disks are used to store:
    - data that does not fit in main memory
    - data that must be kept for a “long” period of time
  - entire speed of the system hinges on disk subsystem and its algorithms
    - some storage needs not be fast (e.g., optical storage or magnetic tape)
- Mass-storage management activities:
  - free-space management
  - storage allocation
  - disk scheduling



Figures: https://www.newyorkcomputerhelp.com/blog/2017/05/03/should-you-defrag-your-computer/

Migration of Data Through Storage Layers

- System must use most recent value, no matter where it is stored
- Many levels of **data coherency**
  - cache coherency for multiprocessors (cache snooping): by hardware
    - all CPUs have the most recent value in their cache
  - synchronization for multi-processes or multiple threads
  - distributed environment situation even more complex
    - several copies of a datum can exist: how to sync the changes?



Resource Management: I/O System Management

- I/O subsystem hides peculiarities of hardware devices from the user
- I/O subsystem is responsible for:
  - manage I/O memory
    - **buffering**: to store data temporarily while it is being transferred
    - **caching**: to store parts of data in faster storage for performance
    - **spooling**: the overlapping of output of one job with input of other jobs

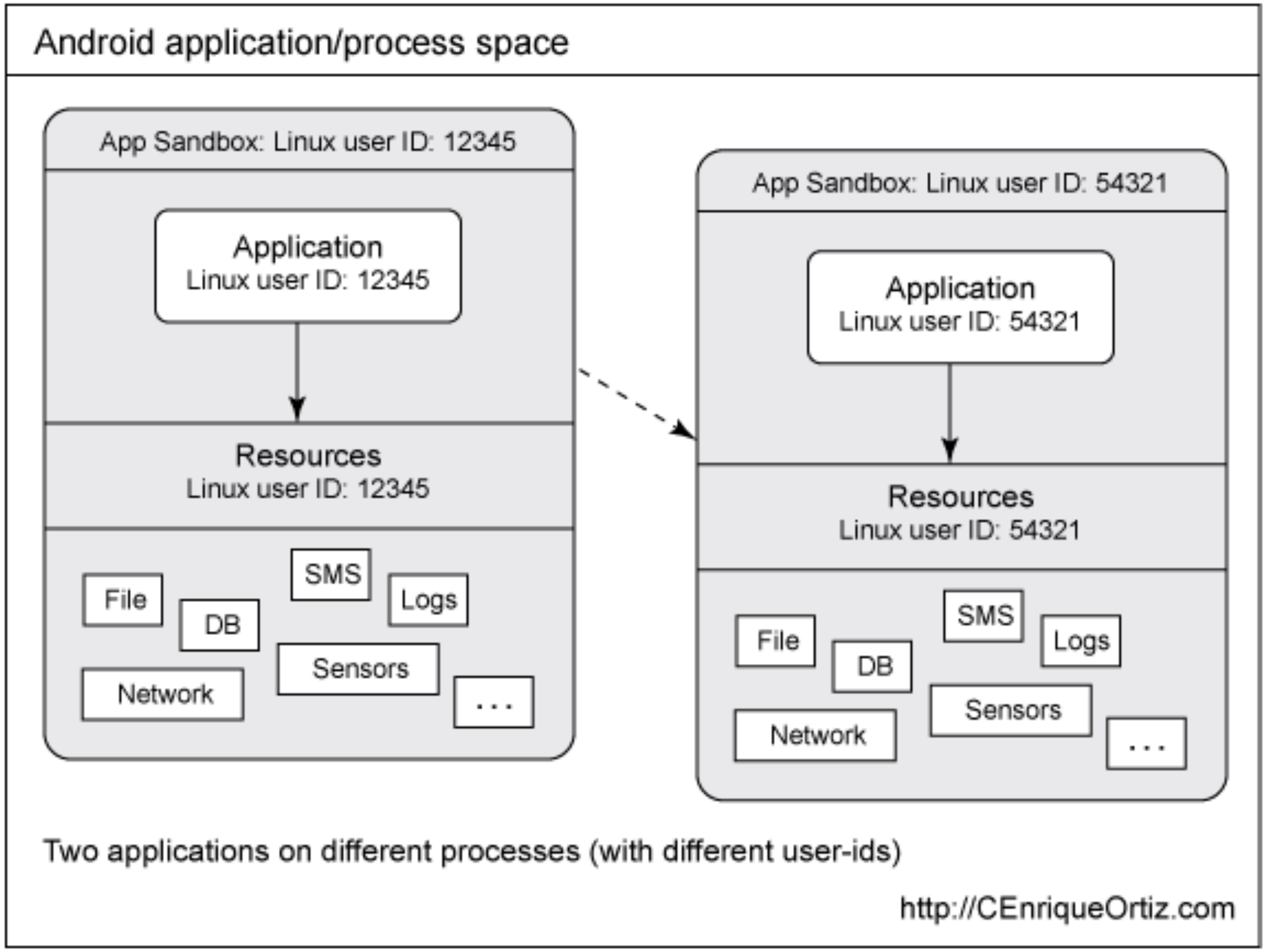
Resource Management: I/O System Management

- OS May provide general **device-driver** interfaces
  - good for programmers: object-oriented design pattern
  - bad from the security perspective: function pointers are heavily used

```
#include<stdio.h>
int(*fpointer)(int, int);  /*Define a pointer to a function */
int add(int, int);        /* Define a few  functions. */
int sub(int, int);
int main()
{
     fpointer= add;       /*     Put the address   of ‘add’ in ‘fpointer’*/
     printf("%d \n", fpointer(4, 5));  /*Execute ‘add’ and print results */
     fpointer=sub;                     /* Repeat for ‘sub’ */
     printf("%d \n", fpointer(6, 2));
     return   0;
}
 int add(int a,   int b)
 {
   return(a + b);
 }
 int sub(int a,   int b)
 {
   return(a - b);
 }
```

What if we can overflow fpointer and points to the shell code?

An Example



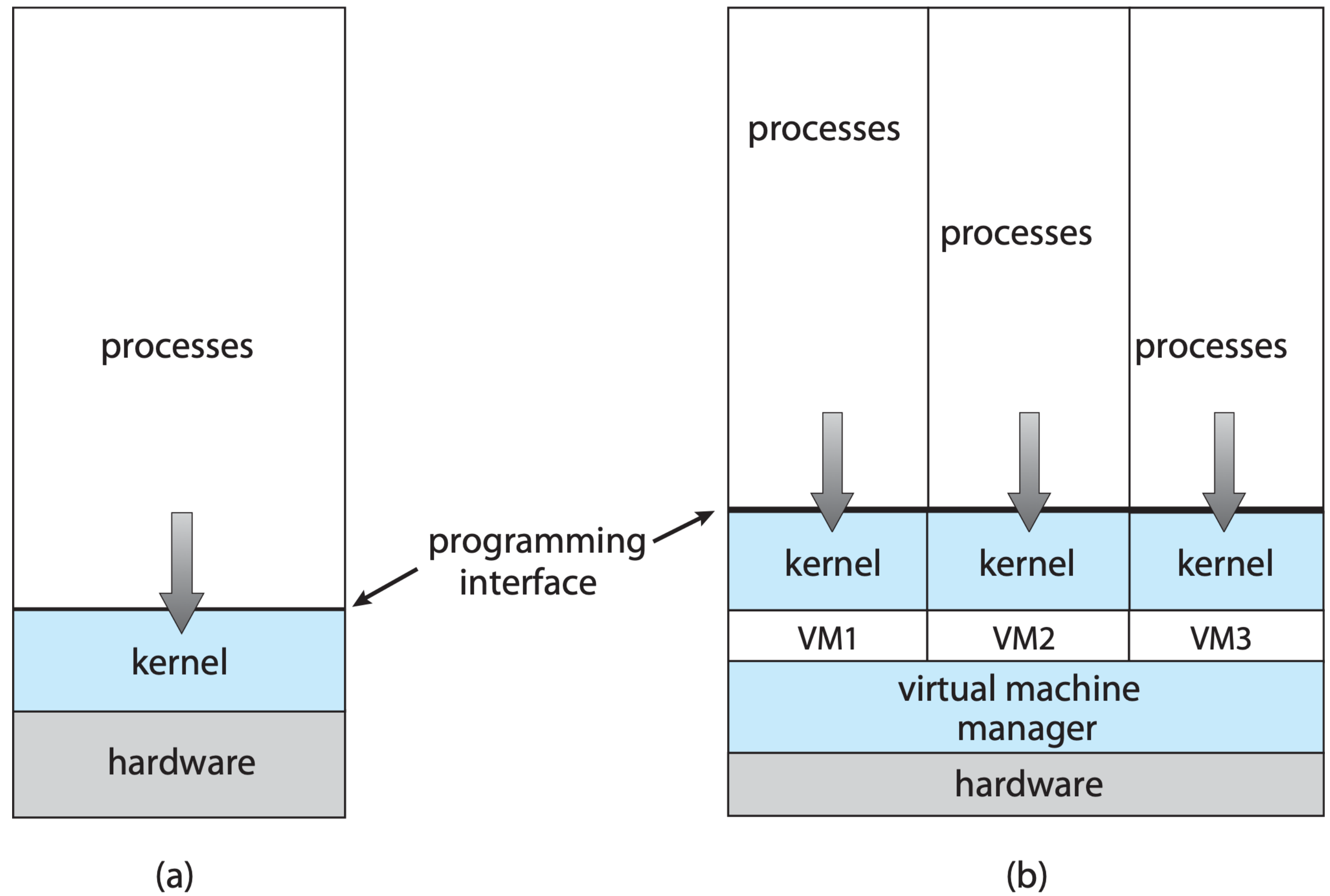
Figures:http://www.ryantzj.com/the-mystical-android-sandbox-and-ipc-inter-process-communication.html

Separate Policy and Mechanism

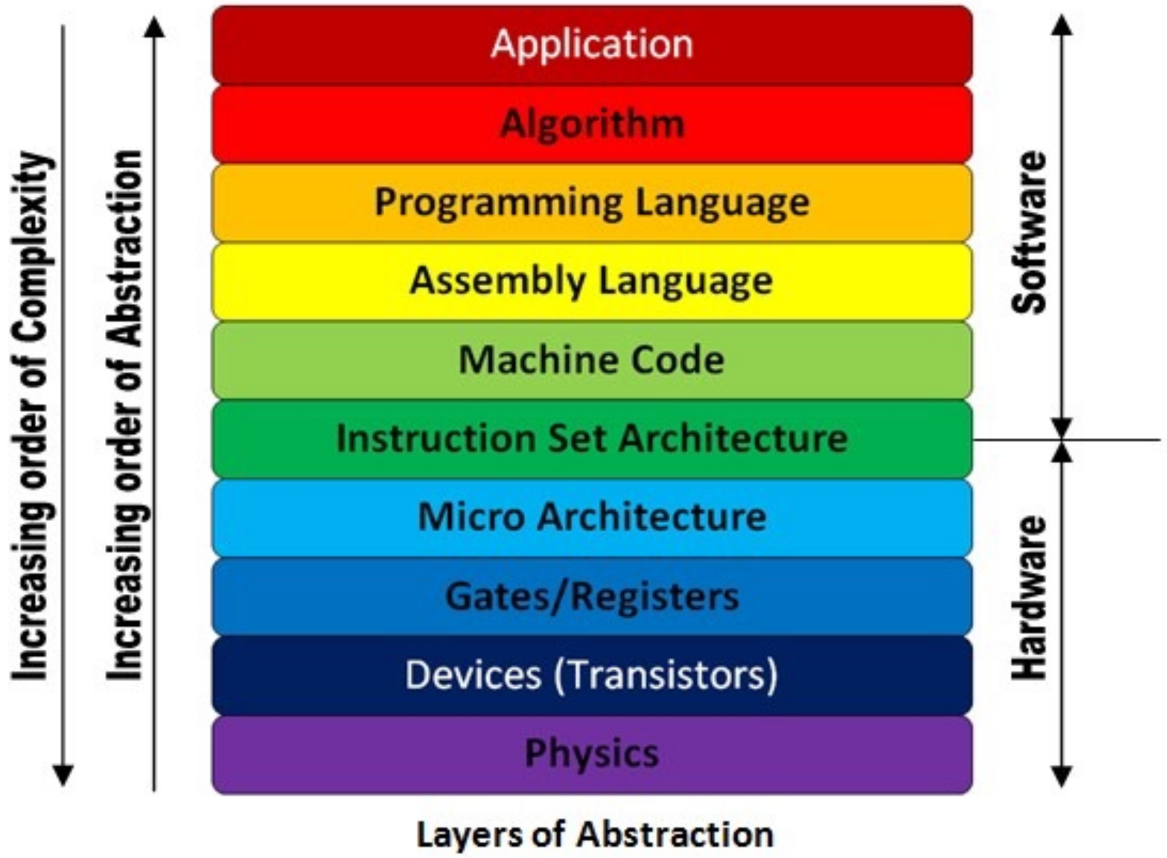
- Mechanism: **how** question about a system
  - How does an operating system performs a context switch
- Policy: **which** question
  - Which process should the process to be switched
- Any other examples about mechanism & policy?
- Advantages & Disadvantages

Virtualization

- **Abstract** the hardware of a single computer (CPU/Memory/IO …) into different environments



Abstraction



Abstractions are fundamental to everything we do in computer science. Abstraction makes it possible to write a large program by dividing it into small and understandable pieces, to write such a program in a high-level language like C without thinking about assembly, to write code in assembly without thinking about logic gates, and to build a processor out of gates without thinking too much about transistors.

Figures: https://electronics.stackexchange.com/questions/353915/what-is-the-role-of-isa-instruction-set-architecture-in-the-comp-arch-abstract?rq=1/