CSCI 4730/6730 OS
(Chap #4 Threads & Concurrency – Part II)

In Kee Kim

Department of Computer Science

University of Georgia

Recap: Process Overhead

- **Creating a new process is expensive**
  - All of the structures (e.g., page tables) that must be allocated
- **Context switching is expensive**
  - Context-switch time is pure **overhead**
  - The system does no useful work while switching
- **IPC is expensive**

Recap: Thread Idea

- What are the similarity in these processes which do the same operation (e.g., server)?
  - They all share the same code and data (in address space)
  - They all share the same privileges
  - They share almost everything in the process

- What don't they share? (processes in server e.g.)
  - Each has its own PC, registers, and stack pointer

Recap: Thread Idea

- **Idea:** why don't we separate the idea of process (address space, accounting, etc.) from that of the minimal "thread of control" (PC, registers)?
- Each process has one or more threads within it
  - **Process**
    - The address space and general process attributes
  - **Thread**
    - A sequential execution stream within a process
    - Each thread has its own stack, CPU registers, etc.
    - All threads in a process share the same address space and OS resources

Thread is now the unit of CPU scheduling

Recap: Memory Map with Threads

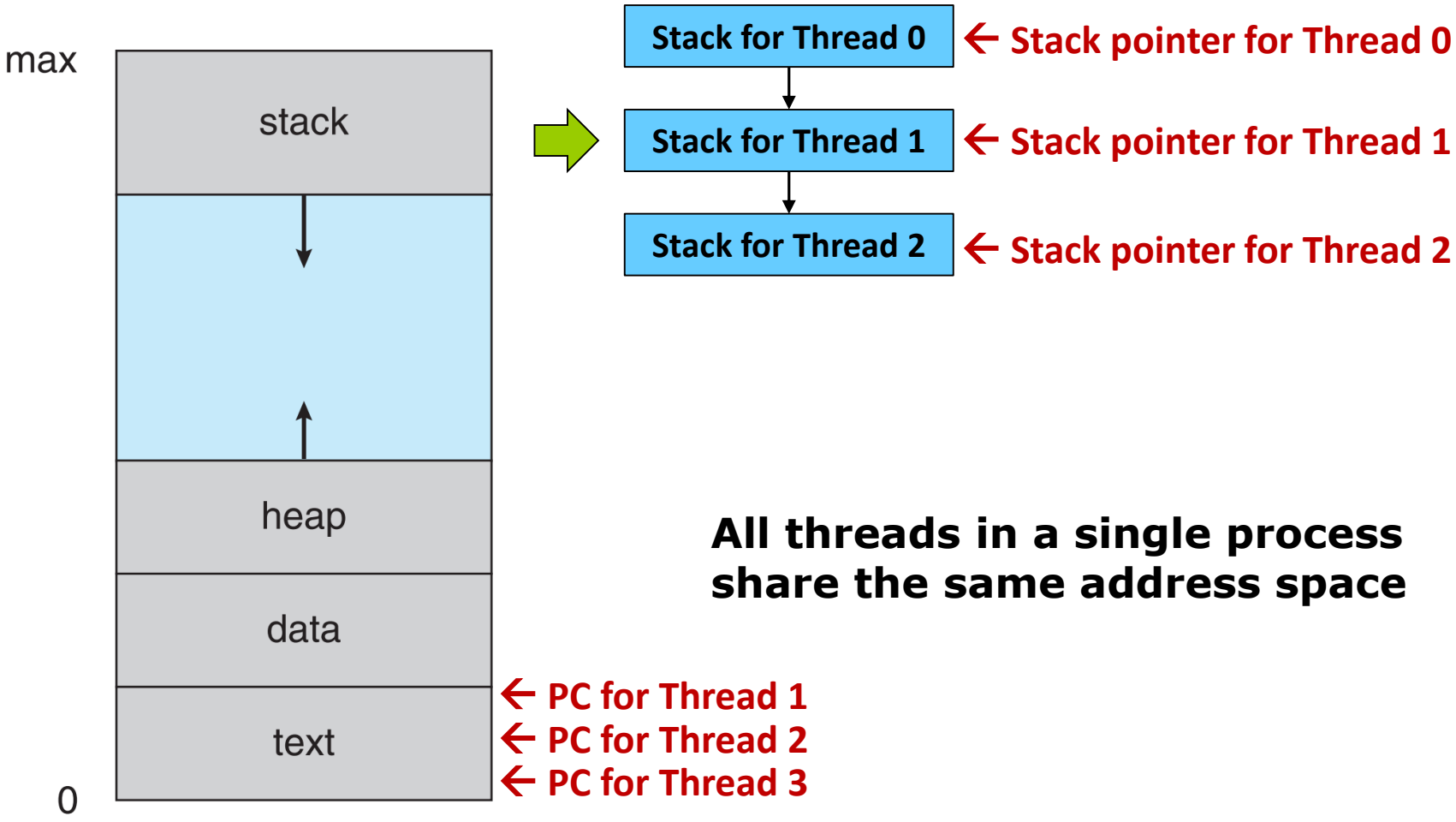
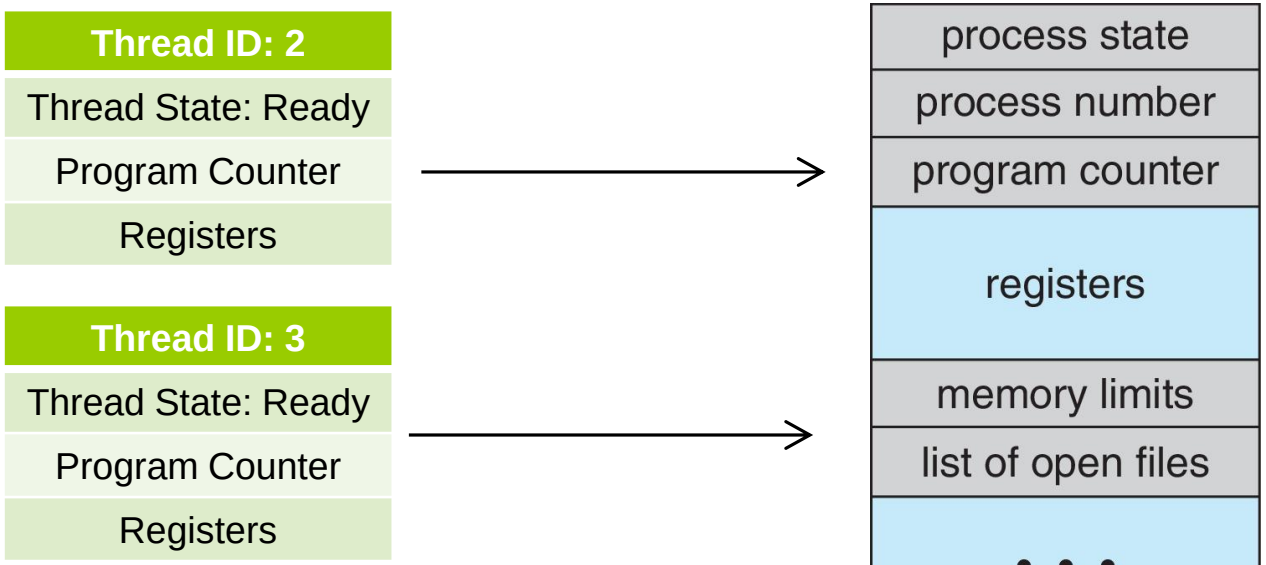


Figure 3.1 Layout of a process in memory.

Recap: Thread Implementation

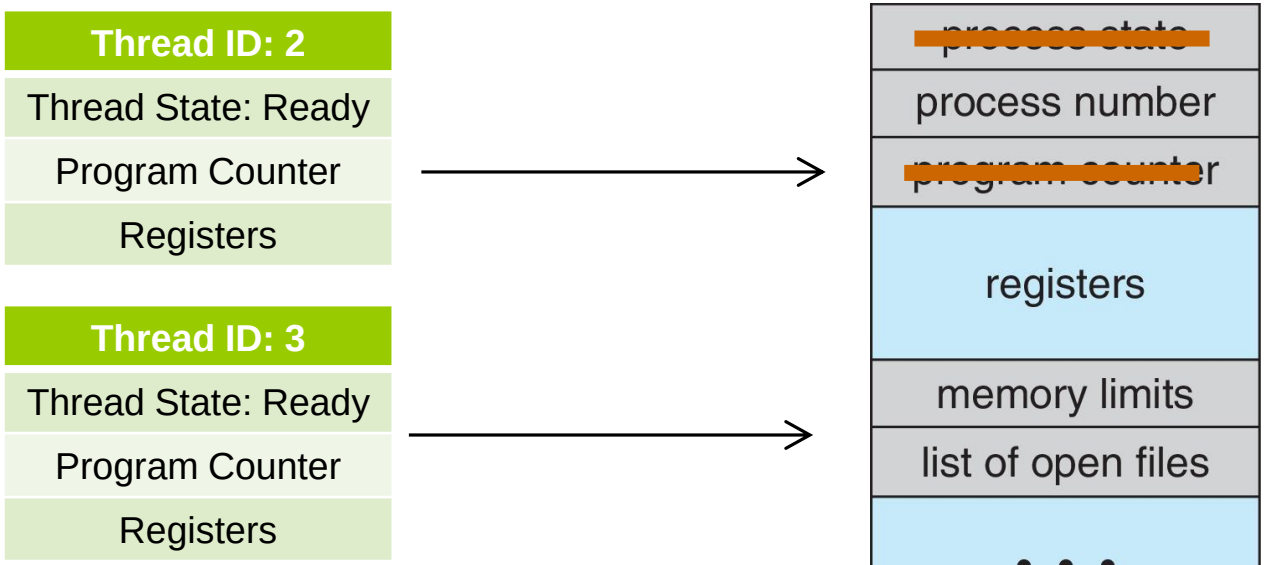
- Idea: Break PCB into two data structures
  - Process specific: address space (memory map) and OS resources
  - Thread specific: program counter, register, etc



TCB contains information on a single thread (pc, registers, ptr to PCB)

Recap: Thread Implementation

- Idea: Break PCB into two data structures
  - Process specific: address space (memory map) and OS resources
  - Thread specific: program counter, register, etc



TCB contains information on a single thread (pc, registers, ptr to PCB)

Recap: Threads vs. Processes

- A thread has no data segment or heap
- A thread cannot live on its own, it must live within a proc.
- Inexpensive creation
- Inexpensive context switching
- If a thread dies, thread specific data and resources are reclaimed and the process is safe

- A process has code/data/heap & other segments
- There must be at least one thread in a process
- Expensive creation
- Expensive context switching
- If a process dies, its resources are reclaimed & all threads die

Multithreaded Applications

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

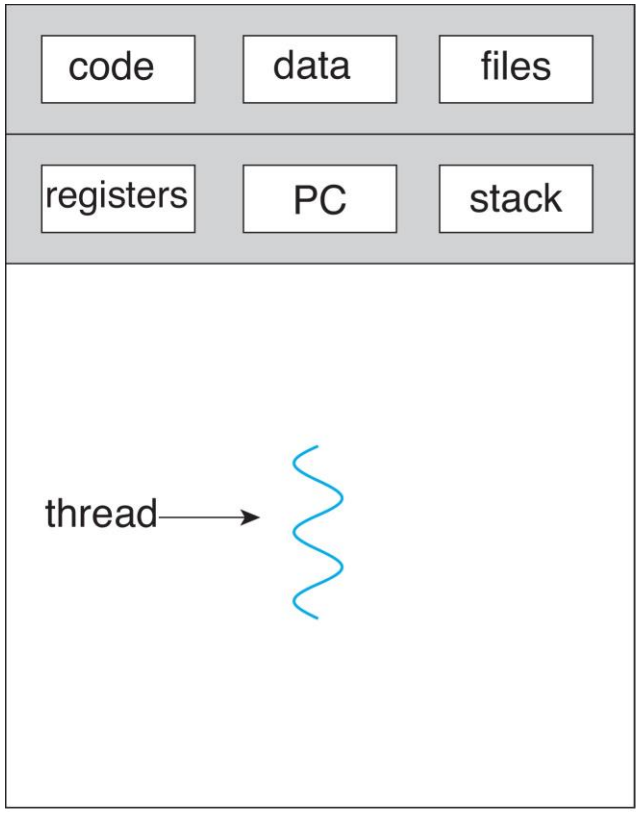
Multithreading Benefits

- Responsiveness
- Resource Sharing
- Economy
- Scalability

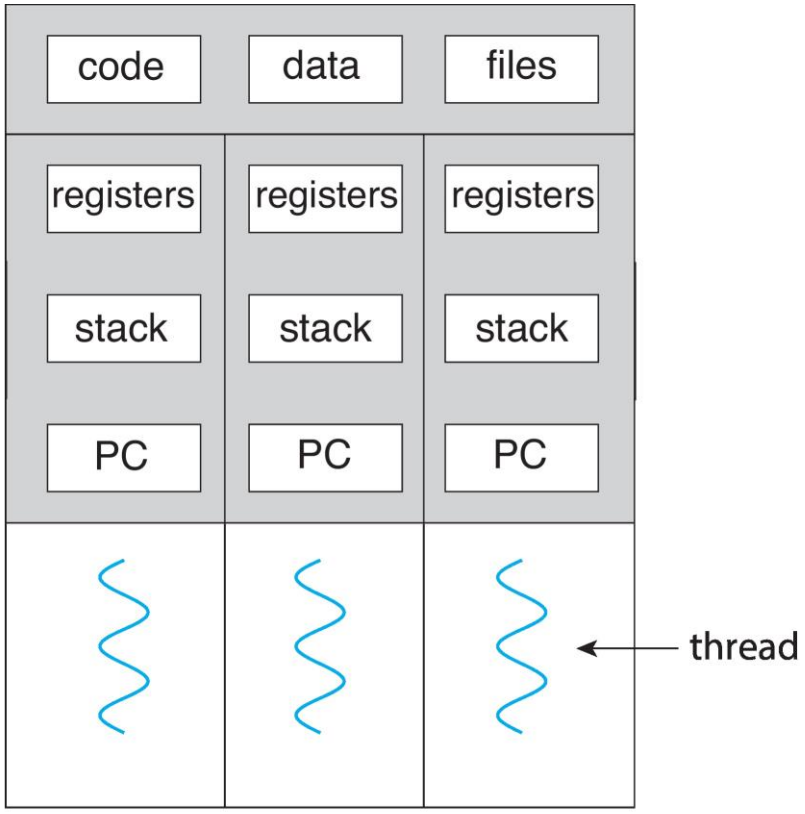
Multithreading Benefits

- **Responsiveness**
  - May allow continued execution if part of process is blocked, especially important for user interfaces
  - Why?
    - Inexpensive context switching and multicores…
- **Resource Sharing**
  - Threads share resources of process, easier than shared memory or message passing
  - Threads are bound to a single process

Single and Multithreaded Processes



single-threaded process



multithreaded process

Multithreading Benefits

- **Economy**
  - Cheaper than process creation, thread switching lower overhead than context switching
  - `fork()` is an expensive system call
- **Scalability**
  - Multicores
  - Other threads in a single process can be executed on different cores

Let’s talk about Multicore Programming

Multicore Programming

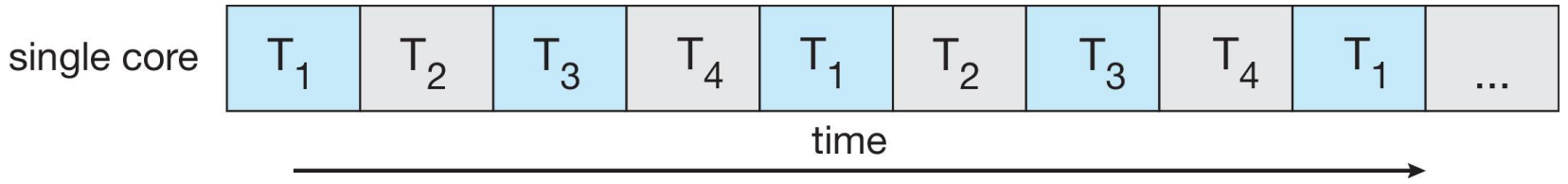
- Multicore programming is not easy
- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Identifying Tasks**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**

Multicore Programming

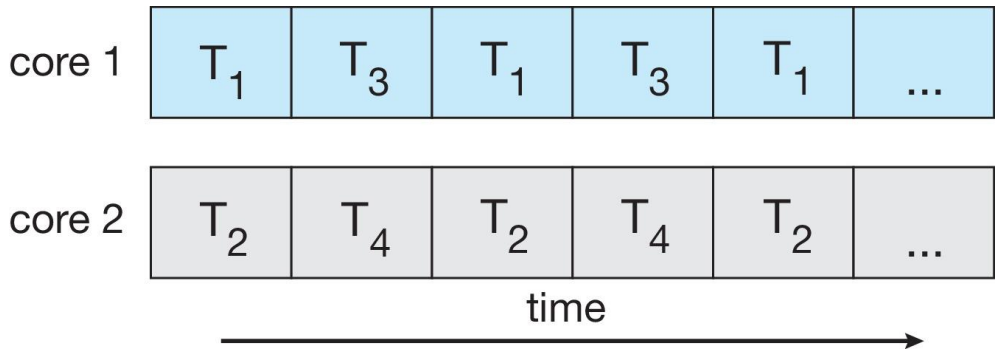
- Concurrency vs. Parallelism
- ***Parallelism***
  - Implies a system can perform more than one task simultaneously
  - More than one processor
- ***Concurrency***
  - Supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

Multicore Programming

- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



Each core performs concurrent executions

Multicore Programming

- Types of parallelism
- **Data parallelism**
  - Distributes subsets of the same data across multiple cores, same operation on each
- **Task parallelism**
  - Distributing threads across cores, each thread performing unique operation

Data and Task Parallelism

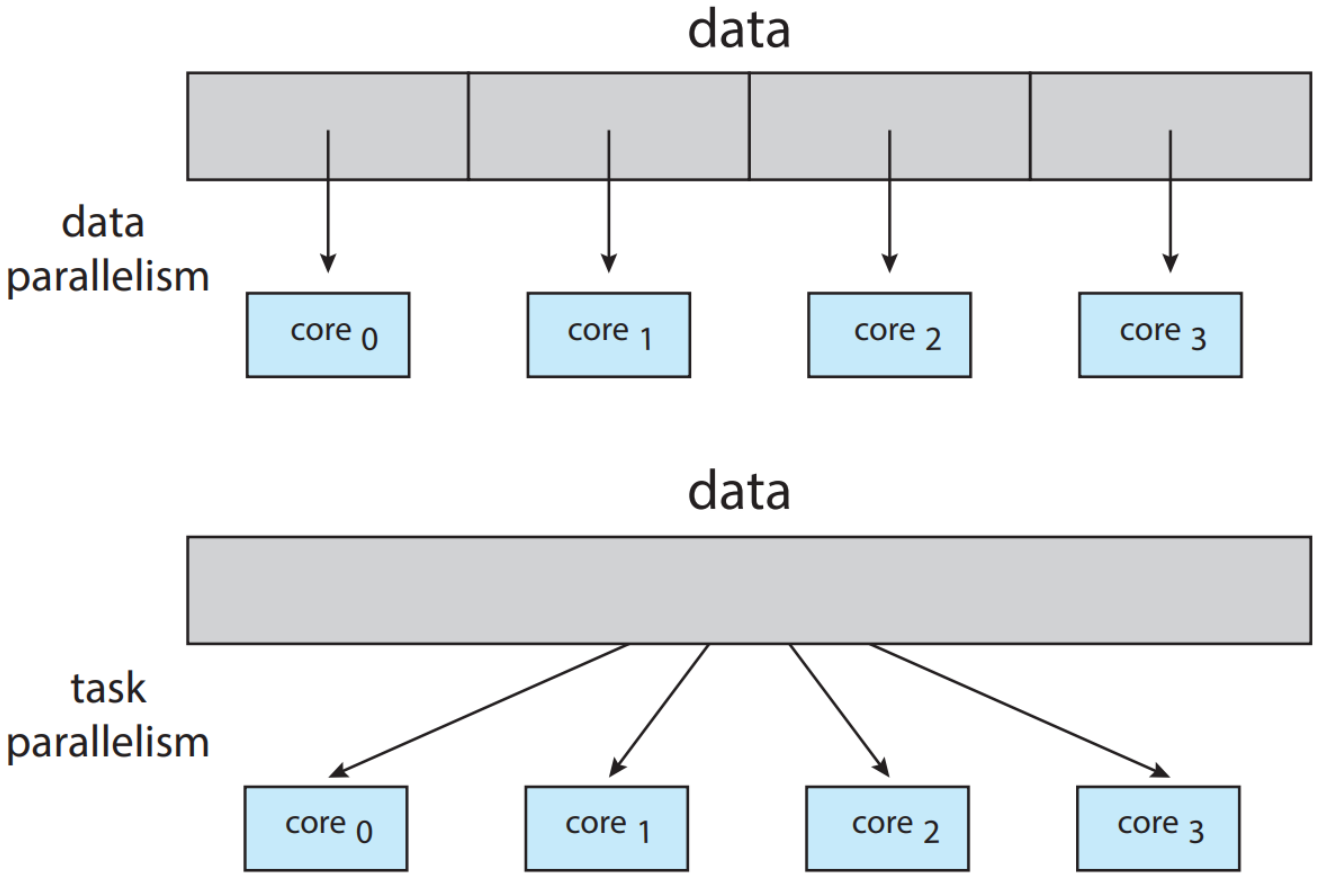


Figure 4.5 Data and task parallelism.

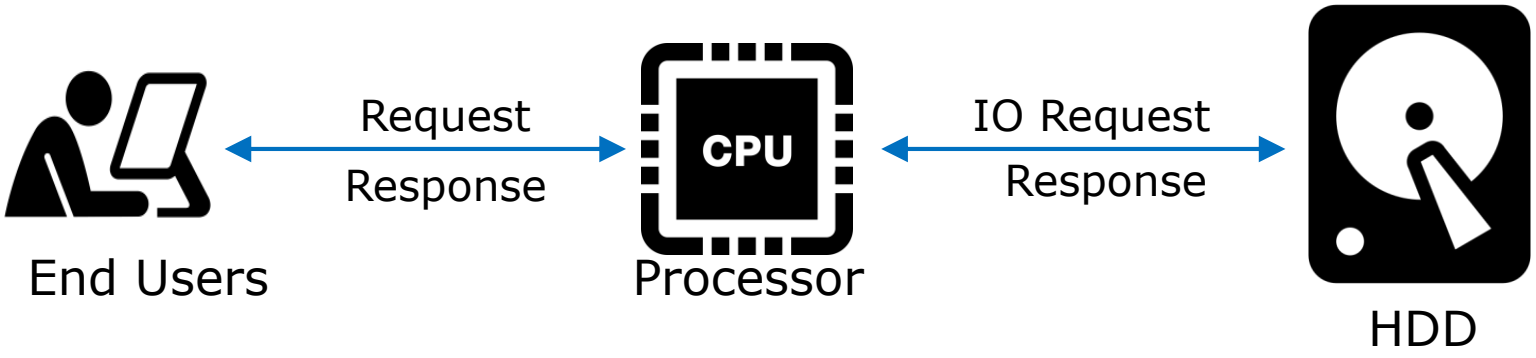
Amdahl's Law

- This is interesting!
- Theoretical performance gain from parallelism (adding more processors) to an application
- Simple question
  - Can I get 4x speed up if I use four threads for my app?
    - Assumption: You have a four-core machine.
    - Compared to single thread architecture?

Amdahl's Law

- Simple question
  - Can I get 4x speed up if I use four threads for my app?
    - Assumption: You have a four-core machine.
    - Compared to single thread architecture?
  - PA #1 -- How much performance improvement?
    - `time ./wc_mul 1 large.txt 0`
    - `time ./wc_mul 10 large.txt 0`

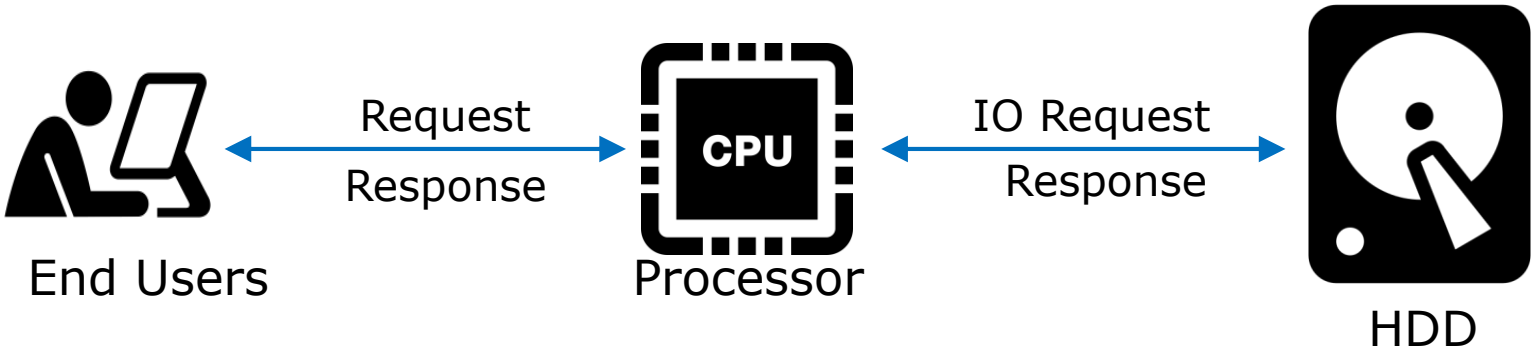
Speedup



Assuming Processor (CPU) takes 0.3 sec, HDD takes 0.7 sec.

What is throughput of this system (Processor + HDD)?

Speedup



Assuming Processor (CPU) takes 0.3 sec, HDD takes 0.7 sec.

What is throughput of this system (Processor + HDD)?

- e.g., 2 reqs / 2 sec = 1 req/sec