CS4410

Operating Systems

Lecture 2: Four fundamental OS concepts

Rachit Agarwal

Context for today's lecture

- One of the harder lectures
  - A lot of new "terminology"
  - Little connection on what you have seen in the past: applications
  - You may feel lost
  - It is okay ….

- We will discuss some of the building blocks of an OS
  - We will cover them in much more detail in upcoming lectures
  - **Today: understand "why" we need these building blocks**
  - And what are the **conceptual challenges in designing them**

Goal of Today's Lecture

- Wrap up discussion from the last lecture
  - Now I know we finish at 4 PM :-)
  - My teaching style, and caveats
- **Four fundamental OS concepts**
- **Some announcements**

Last lecture: 8 basic questions

1. **What is an "operating system", and what does it do?**
2. Why study operating systems?
3. **What is CS4410 about?**
4. What is the course workload, grading policies, etc.?
5. How will this course be organized?
6. **Who am I?**
7. **How do I teach?**
8. Is CS4410 the right class for you?

Recall: What is an operating system, and what does it do?

A **software layer** designed with three goals:

- Enable applications to **conveniently access hardware**
- **Manage** all hardware resources
- **Implement common services** for applications

Recall: What does an OS do?

- Enables **convenient "abstractions"** for applications to access hardware
  - **CPU:** threads
  - **Memory:** virtual memory
  - **Storage devices:** files
  - **Network:** sockets
  - **Server:** collection of resources needed by an application (processes, VM,..)
- **Manages** hardware resources
  - Resource **allocation, sharing and isolation**
- Implements **common services** for applications
  - Security, protection and authentication
  - Reliability
  - Communication
  - Input/output operations
  - Program execution
  - ….

Recall: What is this course about?

Architectural principles, design goals and performance objectives in OS

- **<u>How to think</u> about abstractions offered by OSes?**
  - What abstractions should an OS offer, and why?
  - What should be the semantics (correctness conditions)?

- **<u>How to think</u> about performing resource management in OSes?**
  - What should applications know about other applications?
  - How to share resources? How to ensure isolation?
  - Why statistical multiplexing?

- **<u>How to think</u> about the common services in OSes?**
  - What constitutes a "common service"?
  - How to achieve commonality?

#6: Who am I?

Instructor — Rachit Agarwal

- **Assistant Professor, starting Fall 2016**
- **Previously**: UC Berkeley, UIUC
- **Office:** 411c, Gates Hall
- **Proud of: my students**
  - PhD students (Saksham, Qizhe, **Midhul, Abhishek, Shubham**)
  - Postdocs (Jaehyun, Mina)
  - Undergrad researchers (Grace Jia, Melissa Genaldi)
  - Graduated 5 students so far
    - 4x undergrads
      - 3x now PhD students at MIT (Alana, Akshay, Yannan)
      - 1x now PhD student at UC Berkeley (Lloyd Brown)
    - 1x MS—now PhD student at CMU

Instructor — Rachit Agarwal

- **Research interests**: **problems that excite me**
    - Publish in top conferences of several areas:
        - **Operating systems (OSDI)**
        - **Networking (NSDI, SIGCOMM)**
        - **Databases (SIGMOD)**
        - **Theory (SODA)**
        - **Information Theory (ISIT)**
    - **Diversity reflects my learning and teaching style!**
    - Competitive advantage: ignorance (and curiosity)!

- **Non-research interests:**
    - Food: Chocolate
    - Activity: Flying planes (still training; rarely get time)
    - Skill: Mixing cocktails
    - Sleep: 2-3 hours (so, expect Ed Discussions answers at random hours)

#8: Is 4410 the right course for you?

Ask yourself…

- **Agree with the contract?**
    - No violation to the agreement

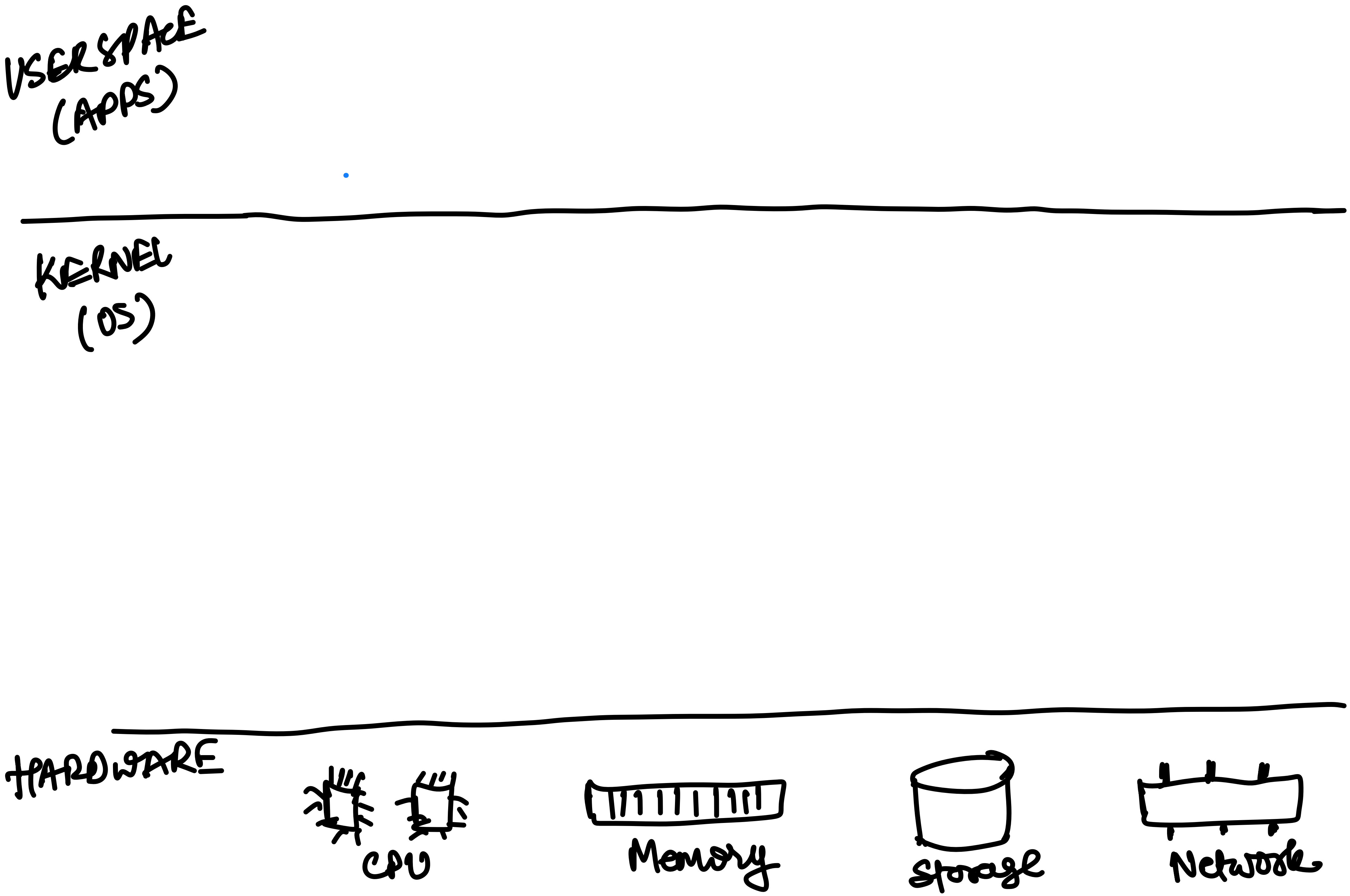
- **Want to understand the “concepts” and the “why” of OS**
    - **4411:** Implementation details
    - **4414:** optimizations and building high-performance application

Questions?

Diving one level deeper

Today: Four Fundamental OS Concepts

- **Thread: Execution Context**
  - A single, sequential execution context
- **Address space** (with **translation**)
  - Program's view of memory is distinct from physical memory
- **Process: an instance of a running program**
  - Address Space + One or more Threads + …
- **Protection/Isolation**
  - Only the “system” can access certain resources
  - Combined with translation, isolates programs from each other


USERSPACE
(APPS)
KERNEL
(OS)
HARDWARE
CPU
Memory
Storage
Network

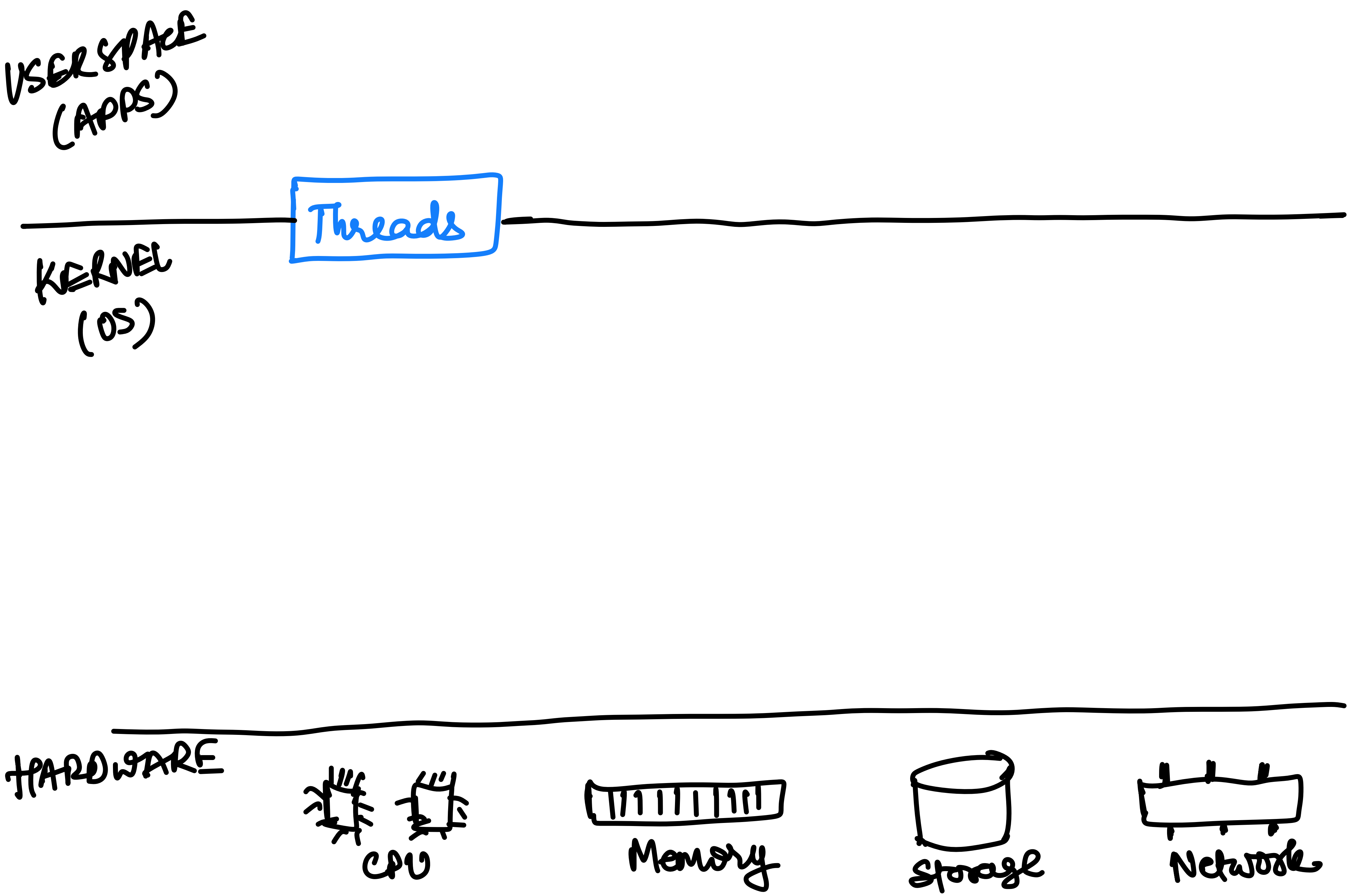
Today: Four Fundamental OS Concepts

- **Thread: Execution Context**
    - A single, sequential execution context

- **Address space** (with **translation**)
    - Program's view of memory is distinct from physical memory

- **Process: an instance of a running program**
    - Address Space + One or more Threads + …

- **Protection/Isolation**
    - Only the “system” can access certain resources
    - Combined with translation, isolates programs from each other


USERSPACE
(APPS)
Threads
KERNEL
(OS)
HARDWARE
CPU
Memory
Storage
Network

Thread

- **Definition: A single, sequential *execution context***
  - Executes a series of instructions in order
  - Only one thing happens at a time
- **Executes** on a processor (core) when resident in that processor's registers
- **Each thread has some "state"**
  - *Program counter (PC):* progress of thread's instruction sequence execution
  - *Thread stack:* reserved region of memory
  - *Stack pointer (SP):* location of last item put onto the stack
  - …. (details in next lecture)
- **Where is thread state stored?**
  - Registers of the processor where thread is running (PC, SP, ..)
  - The rest is "in memory" (*Thread Control Block*)
  - What if there is not enough memory?

Thread

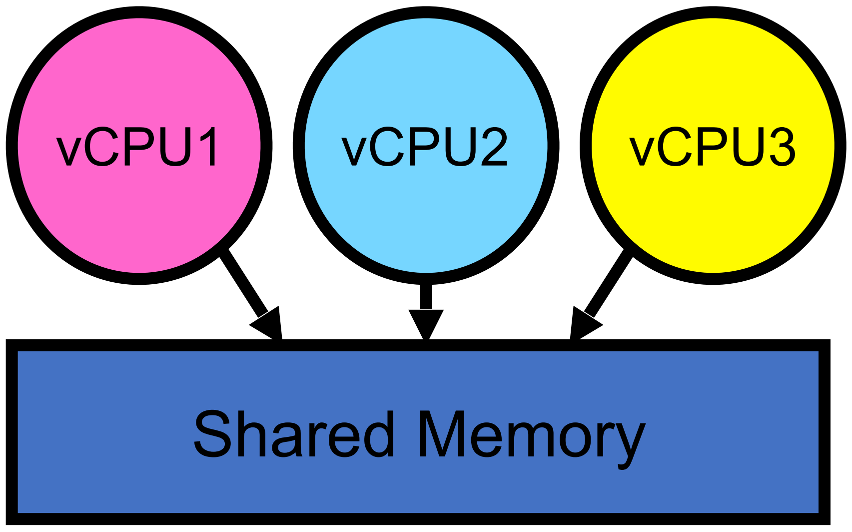
- **What is the difference between a thread and a core?**
  - Thread: “virtual” core
- **Why do we need virtual cores?**
  - In early years: single application, single thread
    - Multiplex over long timescales
    - Problem?
    - Resource underutilization (why? when?)
  - *Statistical multiplexing:* multiple applications, multiple threads
    - When one thread is idle, run another thread
- As an aside, many modern processors support **hyperthreading:**
  - Each physical core behaves as if it is actually two cores
  - Can run two threads simultaneously
  - E.g., execute one thread while the other is waiting on a cache miss

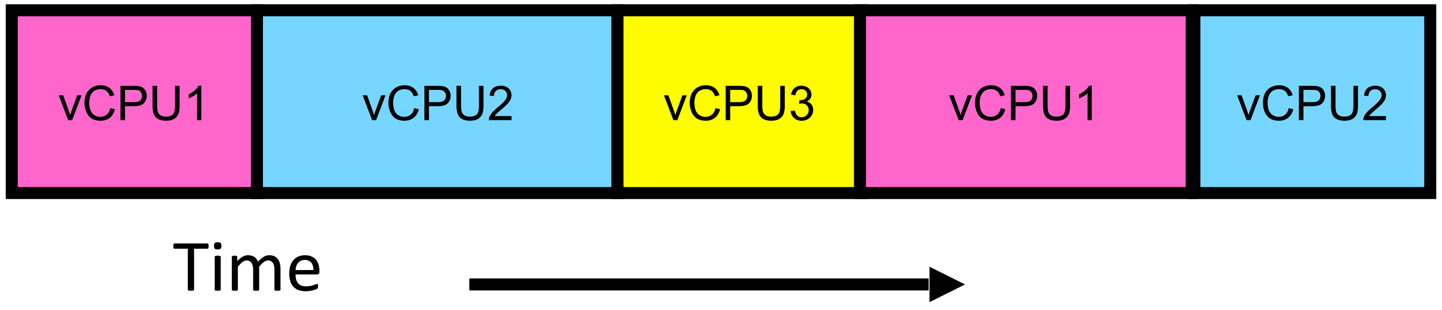
Challenges in designing virtual cores?

- **Scheduling**
    - Sharing physical resources across virtual cores

- **Synchronization**
    - Correctness despite multiple virtual cores

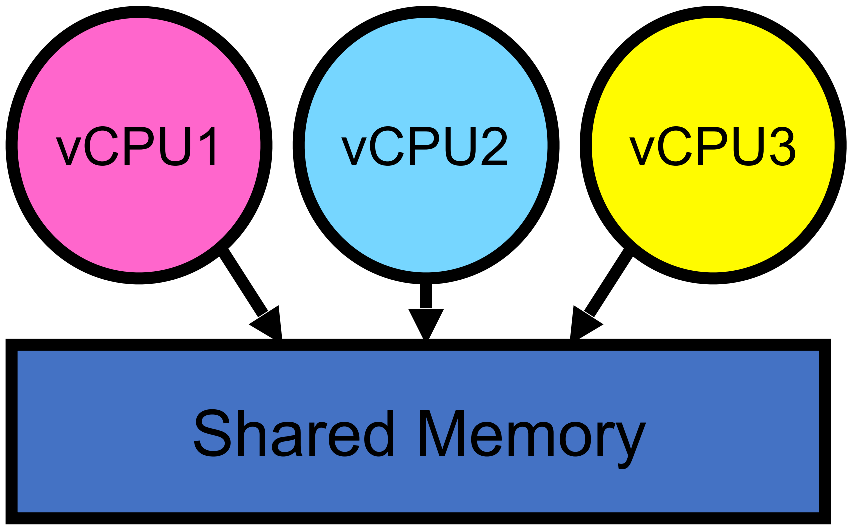
Threads give an illusion of multiple processors

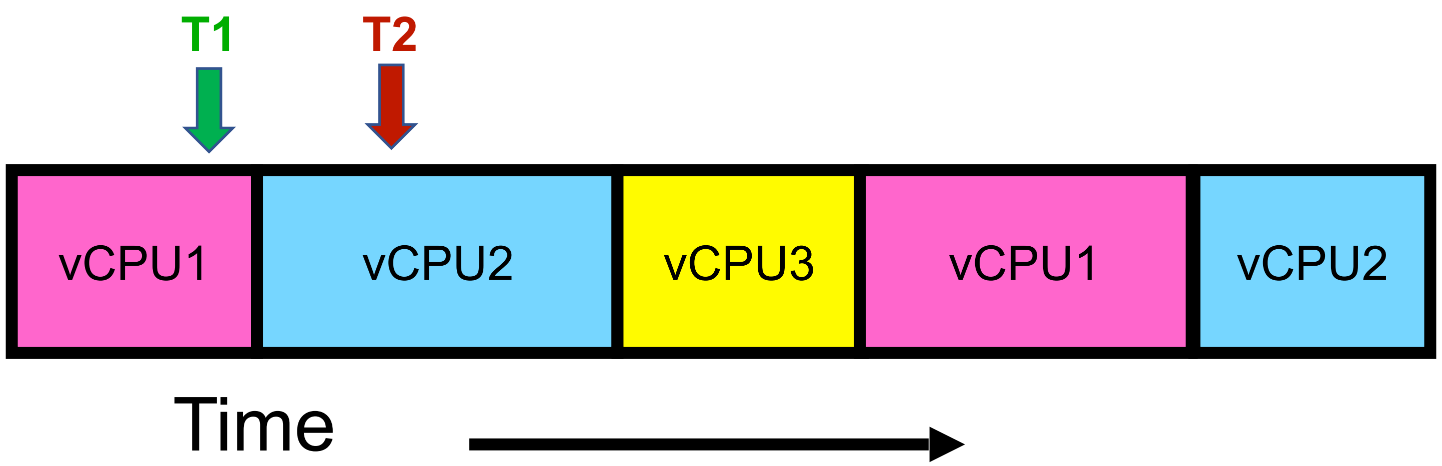




- Each thread has its own state
    - Program counter (PC)
    - Stack pointer (SP)
    - Thread control block (TCB)
        - Everything not in registers

Threads give an illusion of multiple processors



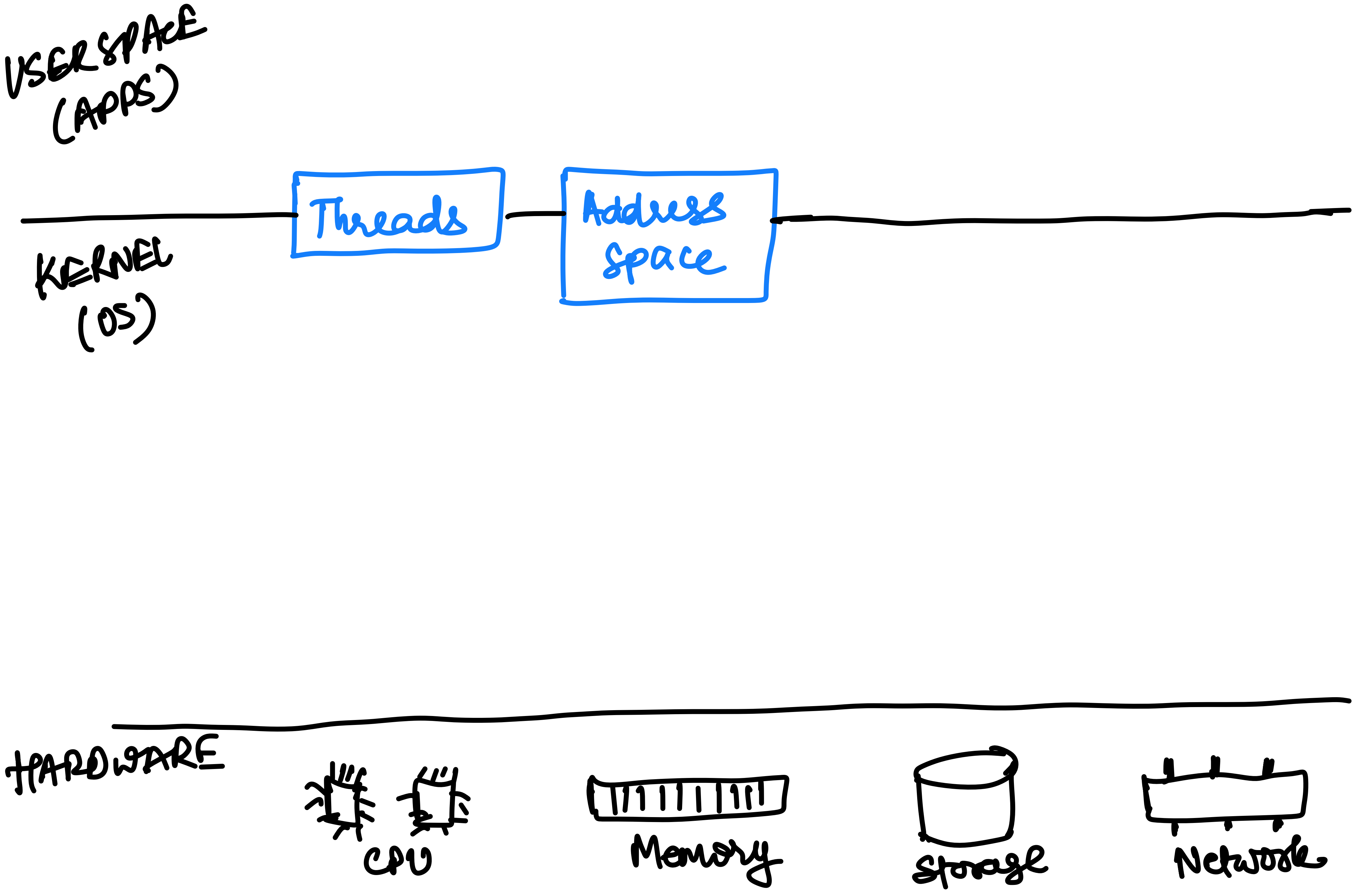


- **At T1:** vCPU1 on real core, vCPU2 in memory
- **At T2:** vCPU2 on real core, vCPU1 in memory
- What does the OS do at the end of T1?
  - Saved PC, SP, … in vCPU1's thread control block (memory)
  - Loaded PC, SP, … from vCPU2's thread control block
    - Jumped to PC

Questions?

Today: Four Fundamental OS Concepts

- **Thread: Execution Context**
  - A single, sequential execution context
- **Address space** (with **translation**)
  - Program's view of memory is distinct from physical memory
- **Process: an instance of a running program**
  - Address Space + One or more Threads + …
- **Protection/Isolation**
  - Only the "system" can access certain resources
  - Combined with translation, isolates programs from each other


USERSPACE
(APPS)
Threads
Address
Space
KERNEL
(OS)
HARDWARE
CPU
Memory
Storage
Network

Key OS Concept: Address Space

- **Physical** address space: where the data *actually* resides
- "**Virtual**" address space: where the program *thinks* the data resides
- Definition: **Set of accessible addresses and the state associated with them**
    - 2^{32} = ~4 Gigabytes on a 32-bit machine
    - 2^{64} = ~18 Exabytes on a 64-bit machine

Virtual address space

- **Why do we need a virtual address space?**
  - In early years: single application over long timescales
  - Now: multiple applications at the same time
    - How do we share memory across applications?
- One possible approach: **static partitioning of the physical address space**
  - Any physical address can be used only by one application
    - Problem?
    - Memory underutilization (why? when?)
- *Statistical multiplexing:* fine-grained sharing of physical address space
  - Give each application an illusion of infinitely large memory

Challenges in designing Virtual address space

- **Granularity**
    - Individual addresses?
    - Memory regions?
    - ….?

- ***Efficient* translation from virtual to physical?**
    - Why *efficient*?

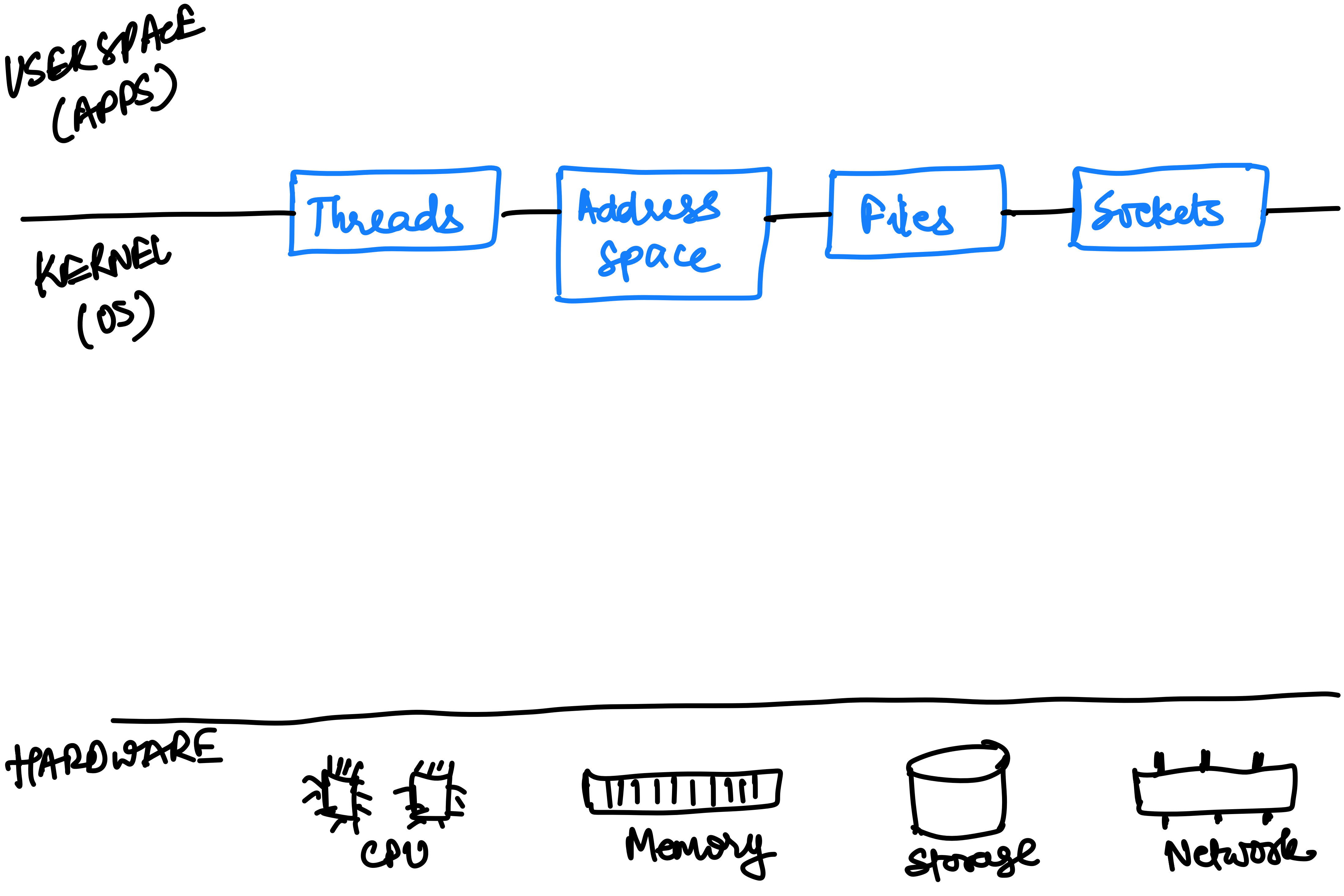
Virtual Address Space at the "page" granularity

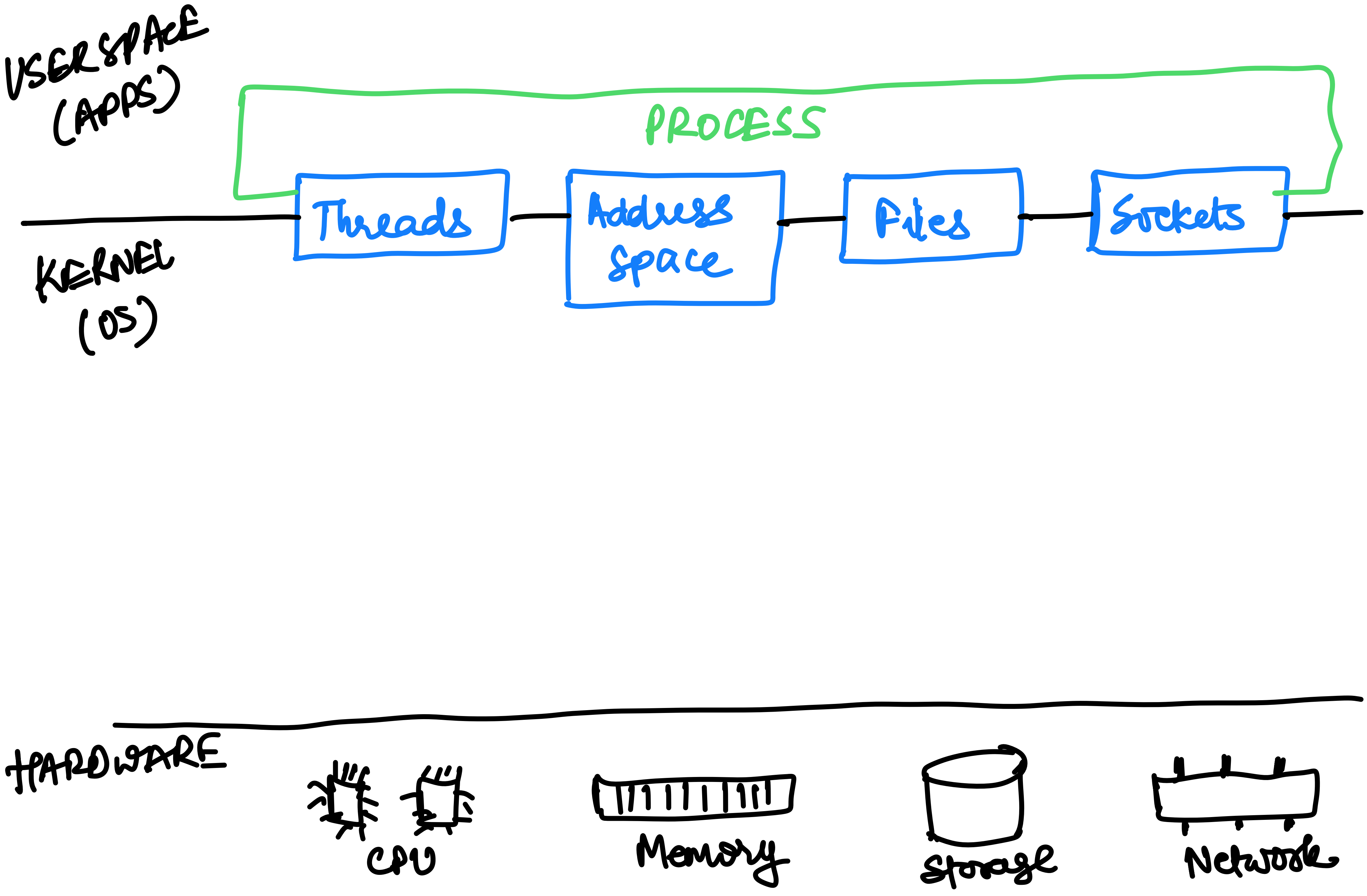
- Sharing at the granularity of "pages"
- Treat memory as page size frames and put any page into any frame
- Map each page in virtual address space to any (page-sized) memory frame
  - What if virtual address space is larger than physical memory?
  - Interesting design questions; return later
- Whenever one needs to access a virtual address
  - Find the page (and offset) that contains that virtual address
  - Translate to page's physical address
  - **Done by the hardware: using a look up table (page table)**
- **Where is the "efficient" part?**

Questions?

Today: Four Fundamental OS Concepts

- **Thread: Execution Context**
    - A single, sequential execution context
- **Address space** (with **translation**)
    - Program's view of memory is distinct from physical memory
- **Process: an instance of a running program**
    - Address Space + One or more Threads + …
- **Protection/Isolation**
    - Only the “system” can access certain resources
    - Combined with translation, isolates programs from each other


USERSPACE
(APPS)
KERNEL
(OS)
Threads
Address space
Files
Sockets
HARDWARE
CPU
Memory
Storage
Network


USERSPACE (APPS)
PROCESS
Threads
Address Space
Files
Sockets
KERNEL (OS)
HARDWARE
CPU
Memory
Storage
Network

Process

- **Definition: execution environment with restricted rights**
  - One or more threads
  - Execution state: everything that can affect, or be affected by, a thread
    - Code, data, registers, call stack, files, sockets, etc.
  - Part of the process state is “owned” by individual threads
  - Part is shared among all threads in the process

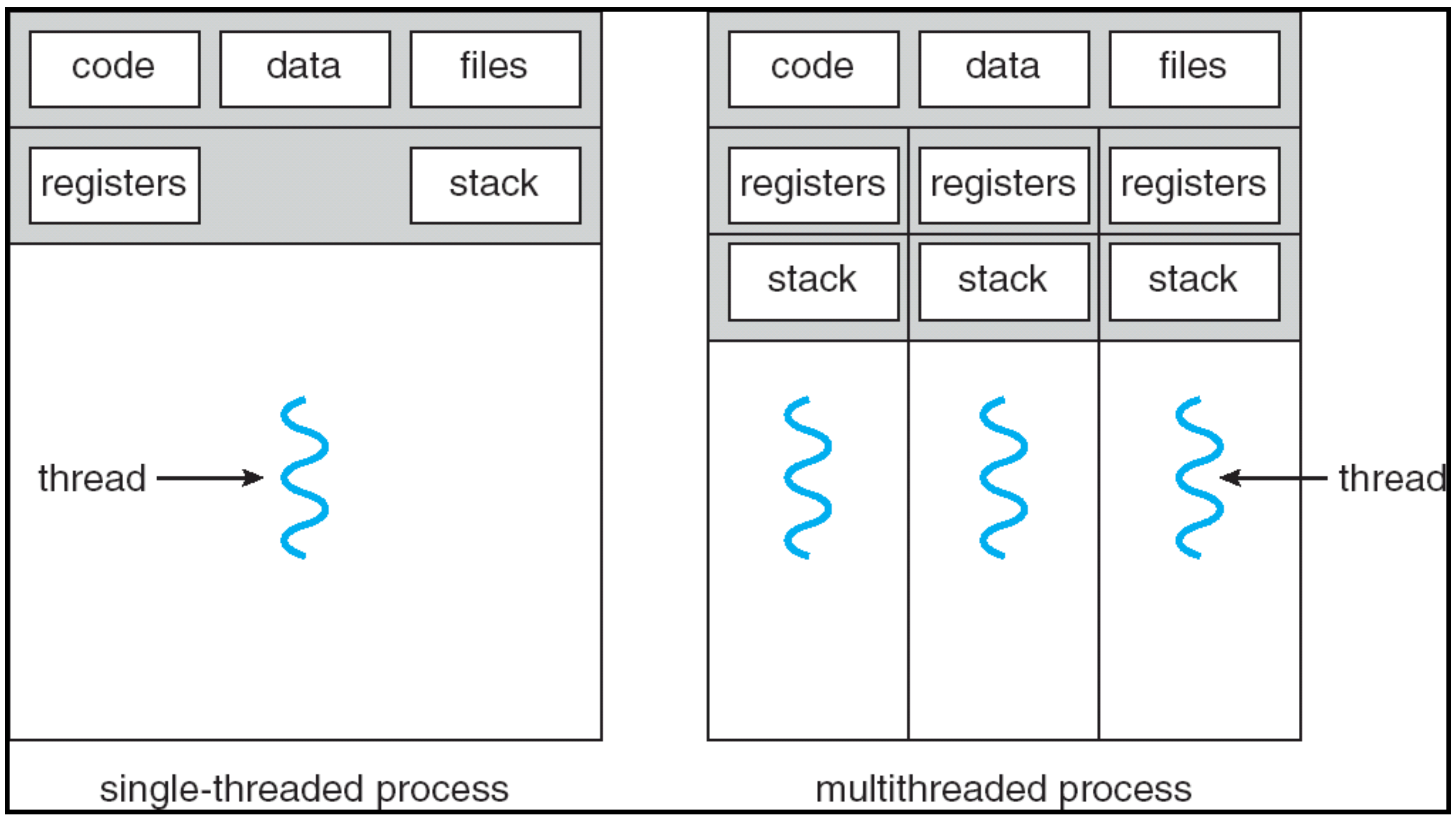
- **Each process has a “state”—Process control block (PCB)**
  - Execution state for each thread
  - Scheduling information
  - Information about memory used by the process
  - Information about files, sockets, etc.
  - ….

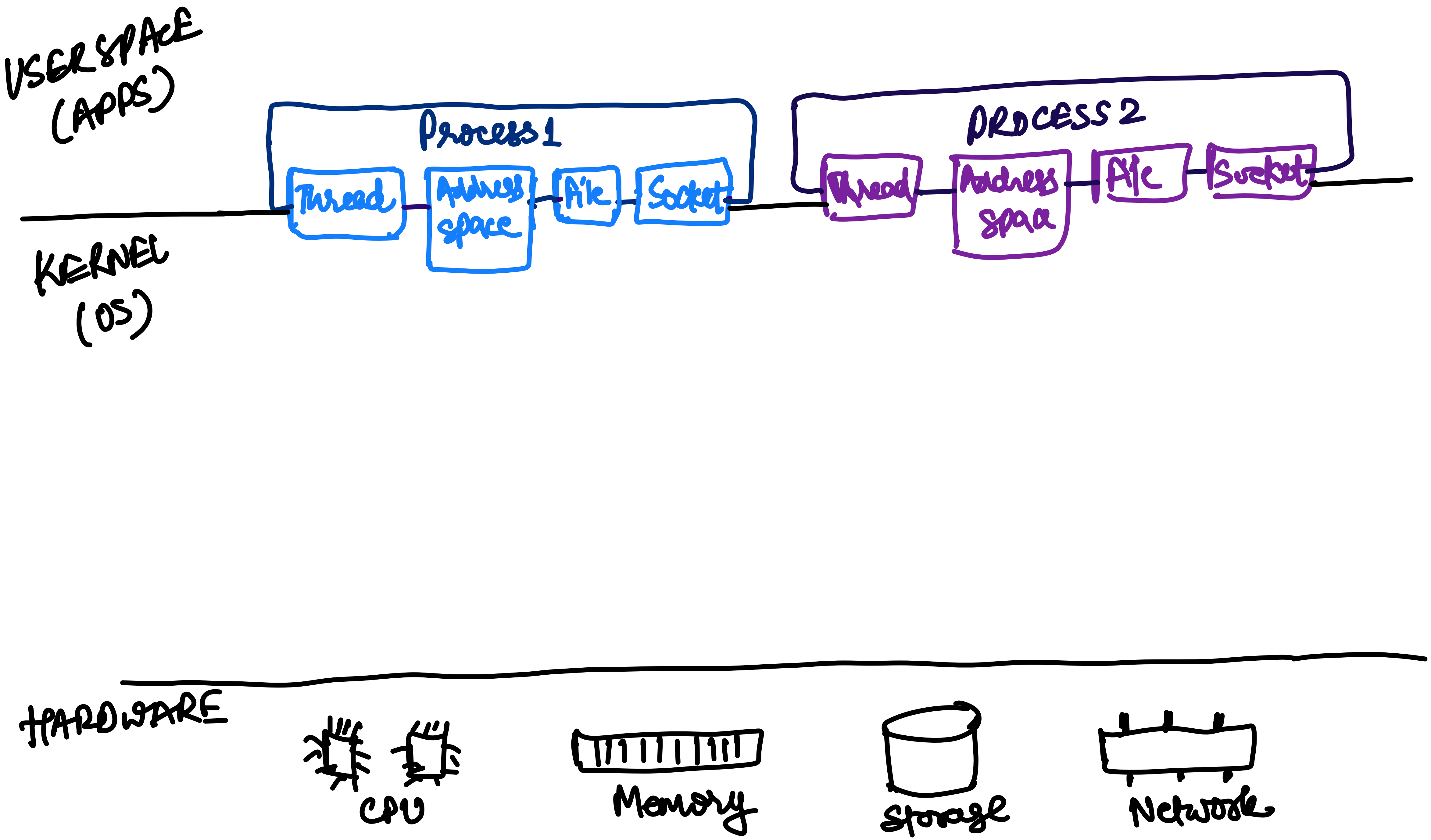
Evolution of OS process model

- **Early operating systems: single tasking**
    - Single process, single thread
    - “switch” applications over long timescales
    - Problem?
- **Late 1970s: multitasking**
    - Multiple processes, single thread per process
    - Share resources across processes
    - Problem?
- **1990s: multitasking, multithreading**
    - Multiple processes, multiple threads
    - Challenges?

Single and Multithreaded Processes

- Why have multiple threads within the same process?
- Threads encapsulate concurrency




USERSPACE (APPS)
KERNEL (OS)
Process 1
Thread
Address space
File
Socket
PROCESS 2
Thread
Address space
File
Socket
HARDWARE
CPU
Memory
Storage
Network

The core challenge with multiple processes?

- Protection/Isolation/Sharing
    - **Reliability:** buggy processes can only hurt themselves
    - **Security:** a process does not have to trust other processes
    - **Fairness:** a good granularity to enforce fair utilization of resources

Questions?

Today: Four Fundamental OS Concepts

- **Thread: Execution Context**
  - A single, sequential execution context
- **Address space** (with **translation**)
  - Program's view of memory is distinct from physical memory
- **Process: an instance of a running program**
  - Address Space + One or more Threads + …
- **Protection/Isolation**
  - Only the "system" can access certain resources
  - Combined with translation, isolates programs from each other

The core challenge with multiple processes?

- Protection/Isolation/Sharing
  - Reliability: buggy processes can only hurt themselves
  - Security: a process does not have to trust other processes
  - Fairness: a good granularity to enforce fair utilization of resources

- **Mechanisms to enable isolation:**
  - **Virtualization**
    - Virtual cores, virtual address space (in particular)
  - **Dual mode operations**
    - Only the OS can access certain resources