Introduction to Parallel & Distributed Programming

Lec 02— Unbundling OS and Architecture

Subodh Sharma (a.k.a SVS) | Jan 05, 2026

Course Logistics

- COL 331 is a **co-requisite**
    - Those who drop COL331 will be de-registered from COL380 at the end of the add-drop period
- Course Discussion: **Piazza**
- Course Webpage: **https://subodhvsharma.github.io/course/col380**
- Audits: **Not allowed**
- Attendace: **RollCall (**htpps://rollcall.iitd.ac.in**)**
    - 75% pre-minor required for reminor, 75% in semester for remajor

Transcribing — Begins from this class!

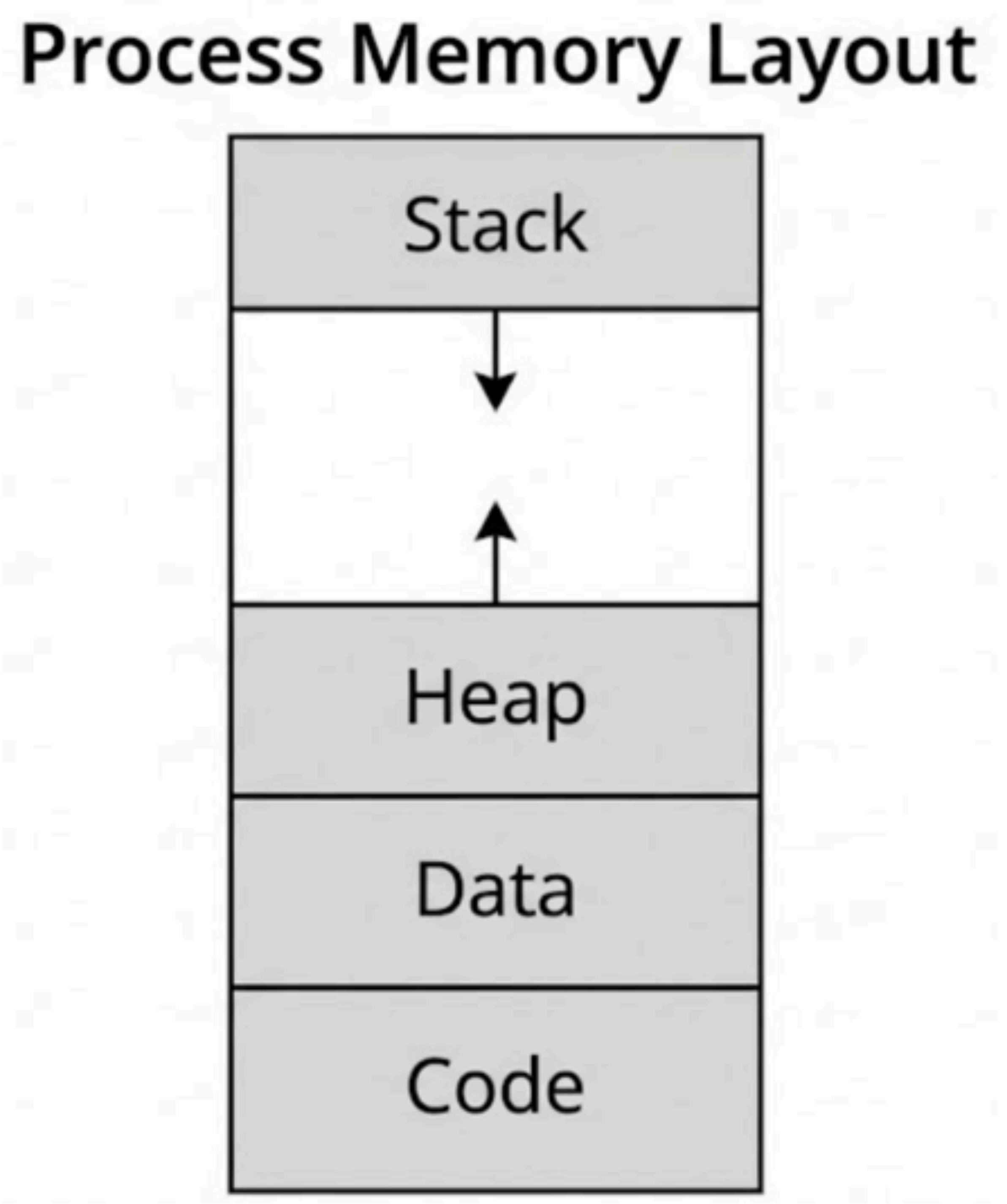
- Drafting of the entire lecture content **in Latex**
  - **Key learning outcomes** for that content
  - Key takeaways (5-10 bullet items)
  - In **2 to 3 examples to motivate** the topical discussion
  - Comprehensive references
  - **4-8 problems** and their solutions

Recap

- Concurrency vs Parallelism vs Distributed
- Concurrency in Hardware
  - Single Cores — **Instruction pipelining**
    - Superpipelined processors — **Superscalar pipelines**
      - **I1:** `load R1, @1000;` **I2:** `Add R1, @1004.` **Can I1 and I2 be simultaneously issued?**
      - **Q: When can an instruction scheduler truly exploit pipelined parallelism?**
  - Multicores - **Multiple instruction execution streams**

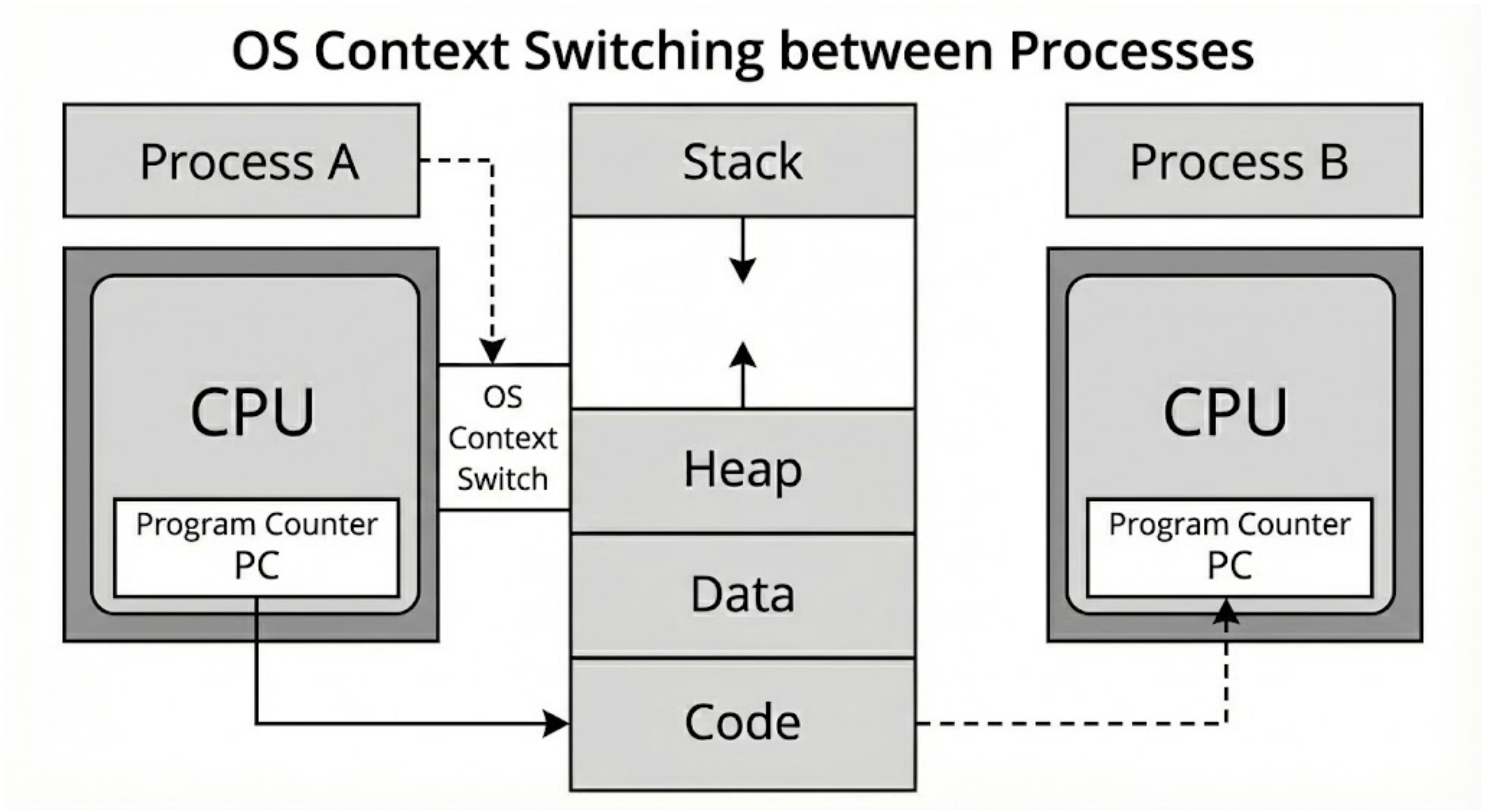
Process

- Process is a **program under execution**
  - `code.cpp —> gcc code.cpp —o exe —> ./exe`
- **Code segment:** Read-only storage for code instr
- **Data segment:** Global and static vars
- **Heap:** Dynamic memory allocation area
- **Stack:** Function parameters, return address, local vars



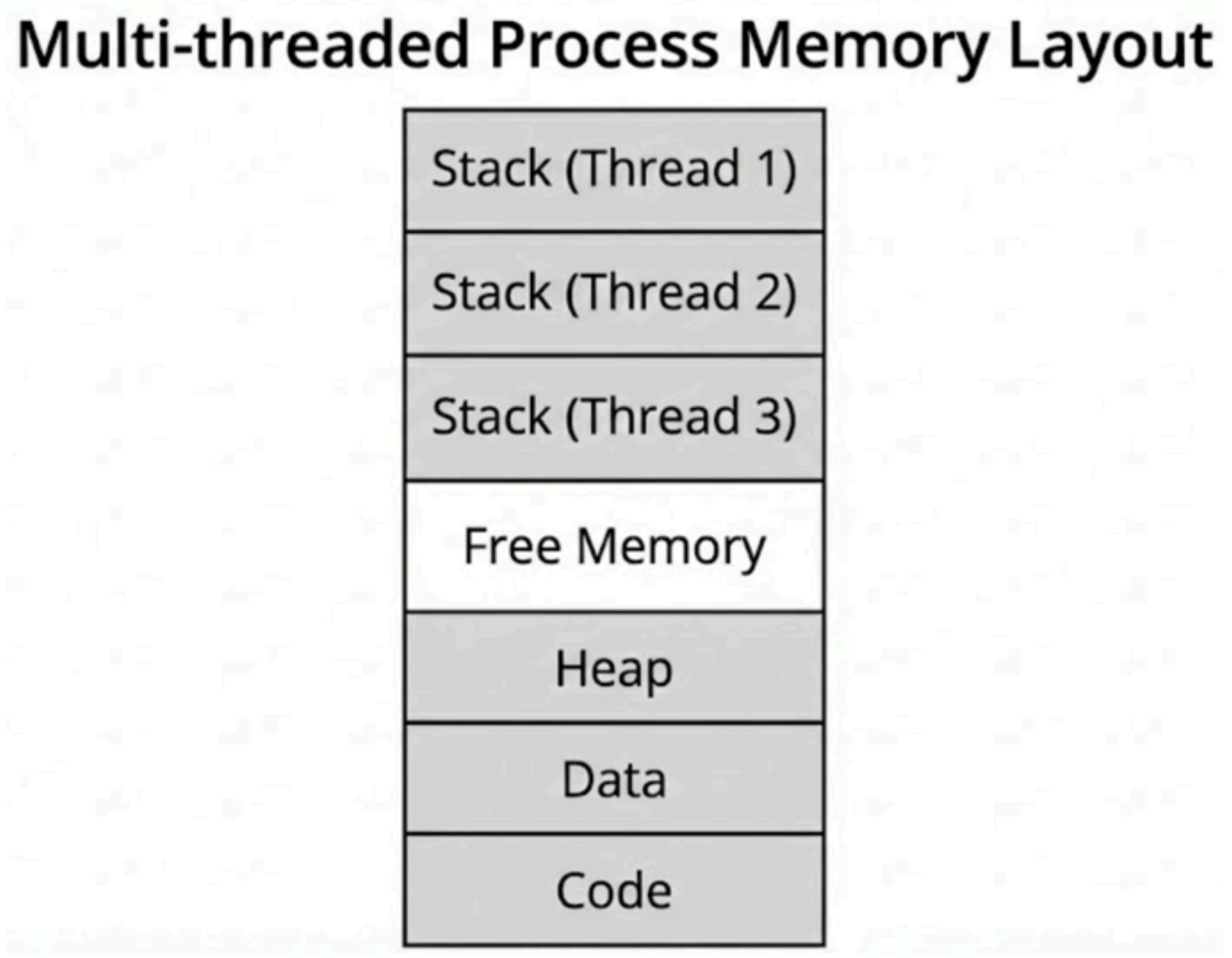
OS Managing Processes

- **PC (program counter)** holds the address of the next instruction to be executed
- **Multiple processes:** OS manages them via **context-switching**
  - **Context:** PC, Process state, Register values, Ptrs to Page Tables, I/O devices allocated …
    - **PCB:** OS saves the PCB of one process and makes a context switch to another

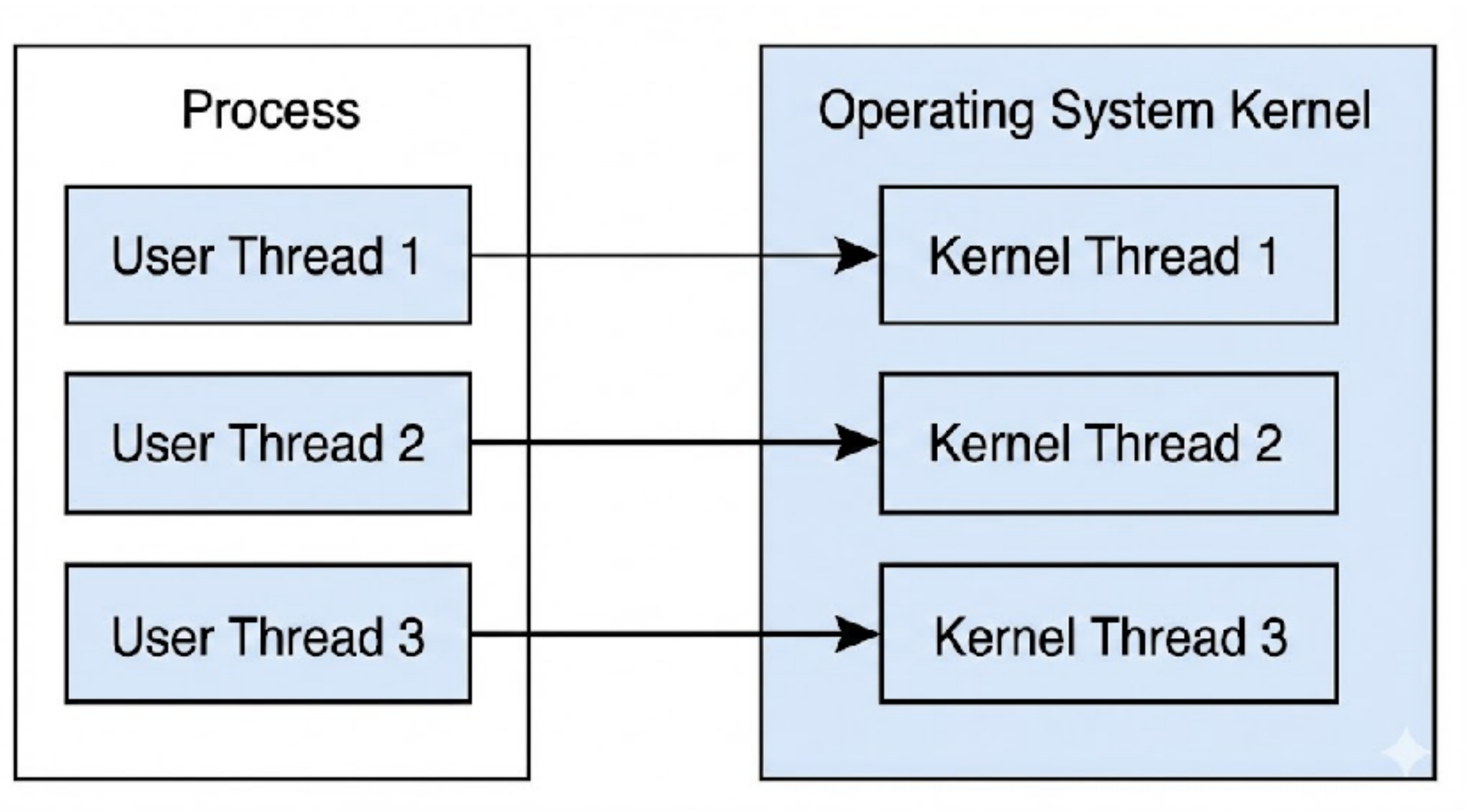


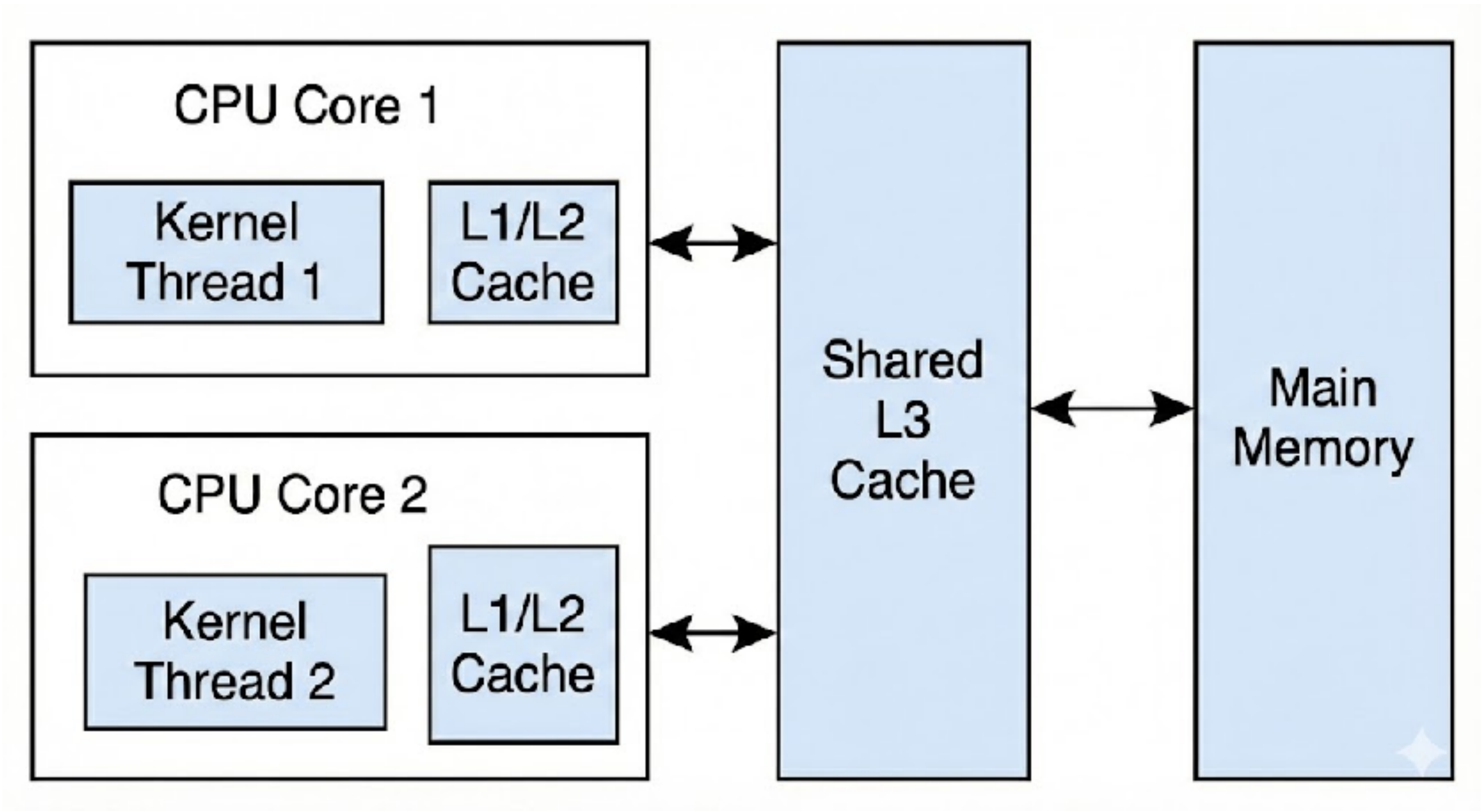
Thread

- Threads **have their own stack** but share the code, data and heap segments
- **Thread Control Block (TCB)**: Has much smaller memory footprint (don’t have to same entire memory maps)
    - TCB is used for context switching



OS Managing Threads

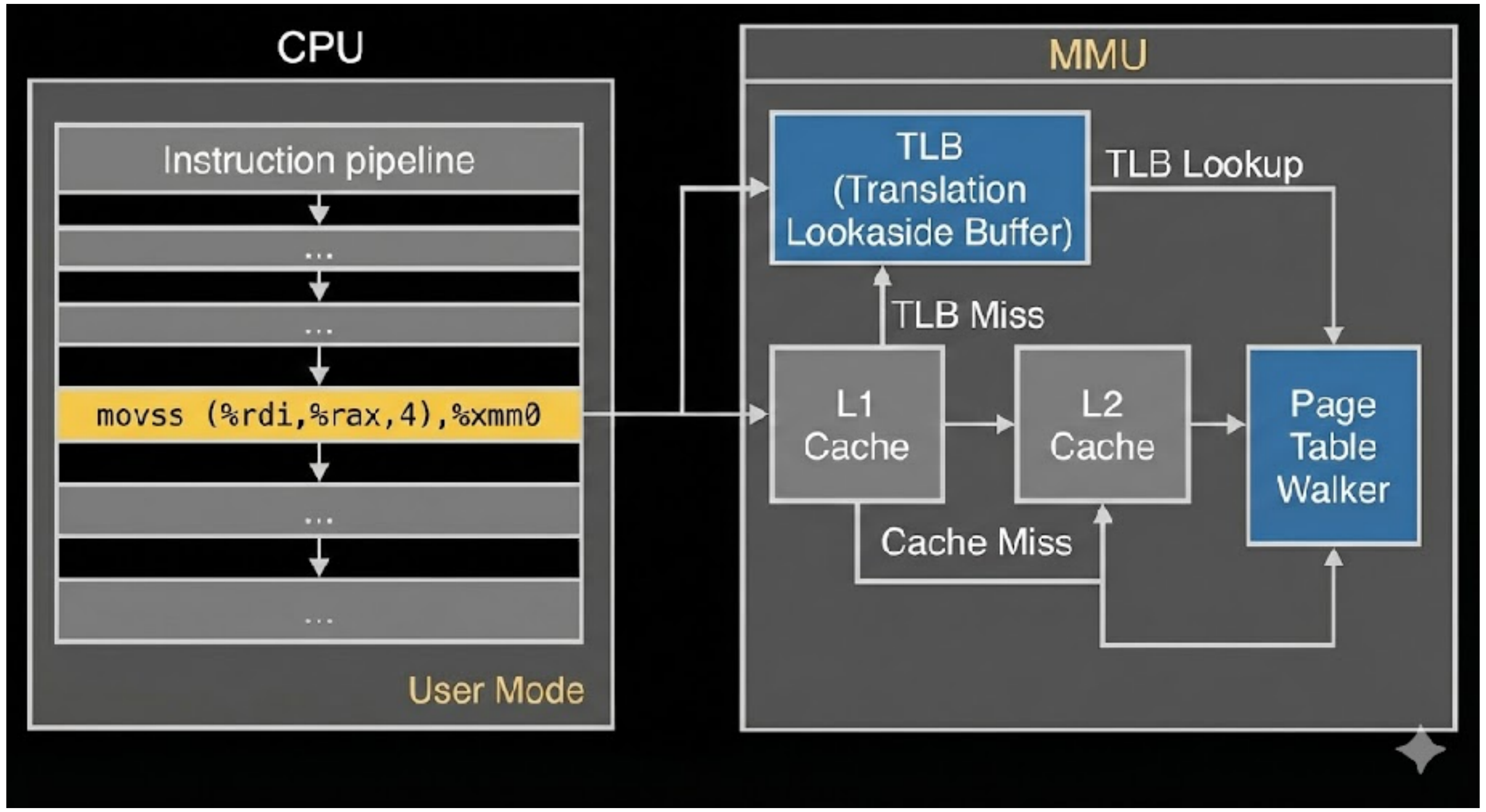


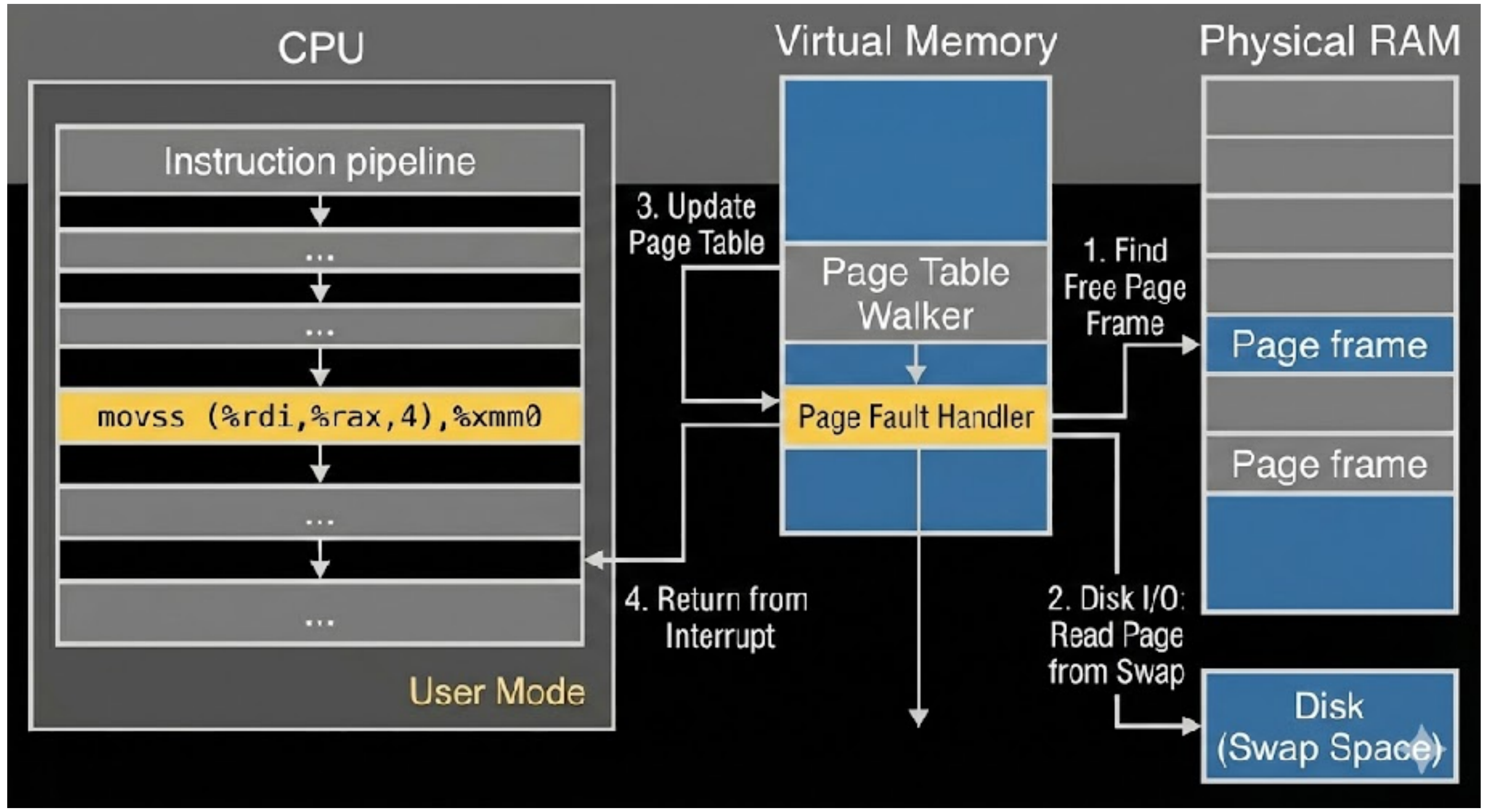


- Each User thread is mapped to a Kernel thread (mapping can be one-2-one or many-2-many)
- Kernel threads are mapped to physical cores;

Memory Management

How is instruction executed?



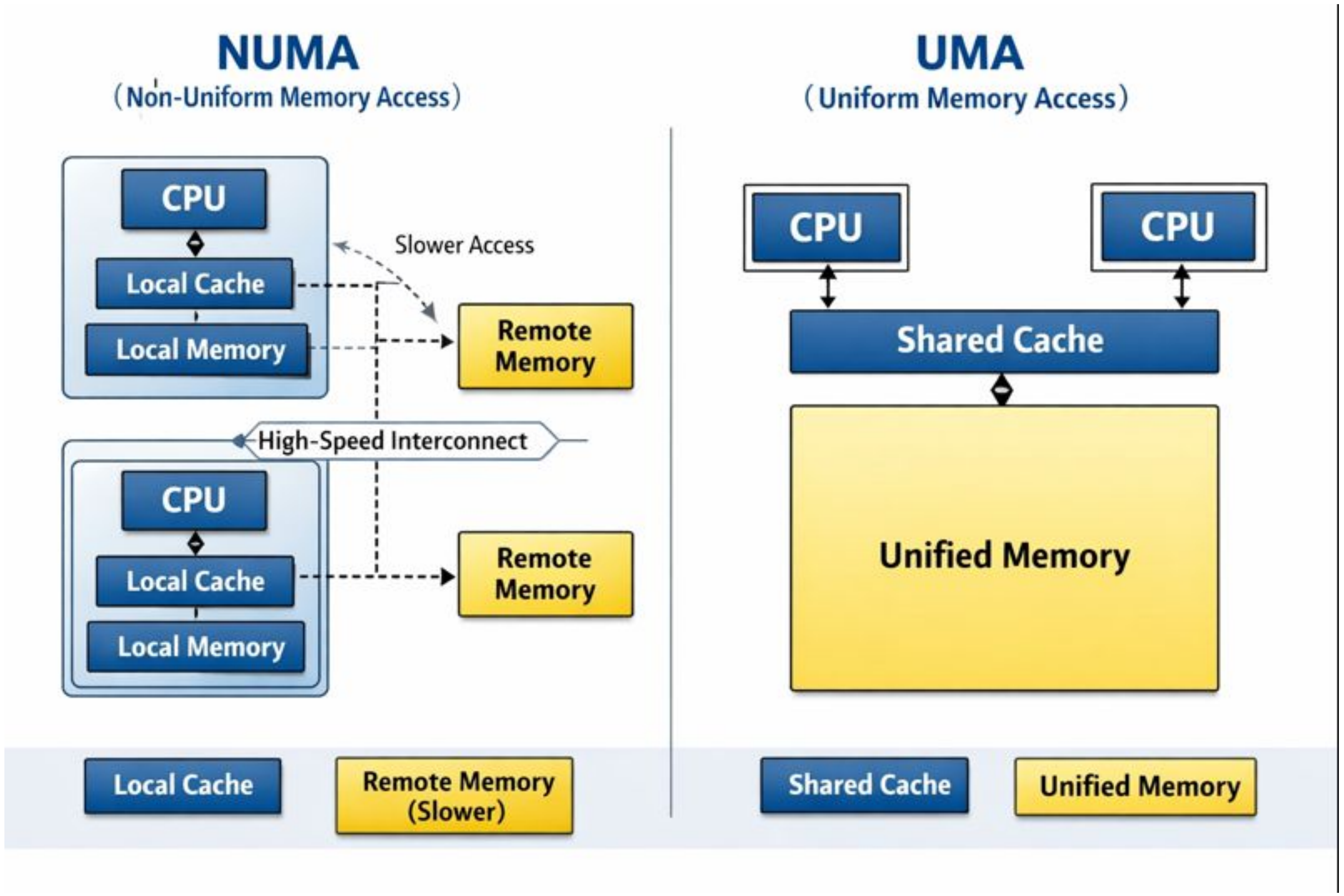


- Instr: **`a[i] -> tmp`**; Memory access is much slower; Caches are small

Caches

UMA vs NUMA

- Caches help exploit temporal and spatial locality of references
- Two types:
  - UMA: Uniform memory access
  - NUMA: Non-uniform memory access
    - Accessing memory stores on different NUMA nodes may result in variable access times



Cache Line Granularity & Coherence

- **Cache line**: Typically 64 Bytes
    - **Granularity Rule**: When you ask for an integer, **CPU fetches the entire line** that contains that integer. (**Spatial Locality)**
- **Coherence:** A h/w protocol that ensures all cores agree on the value of a piece of data
    -

False Sharing

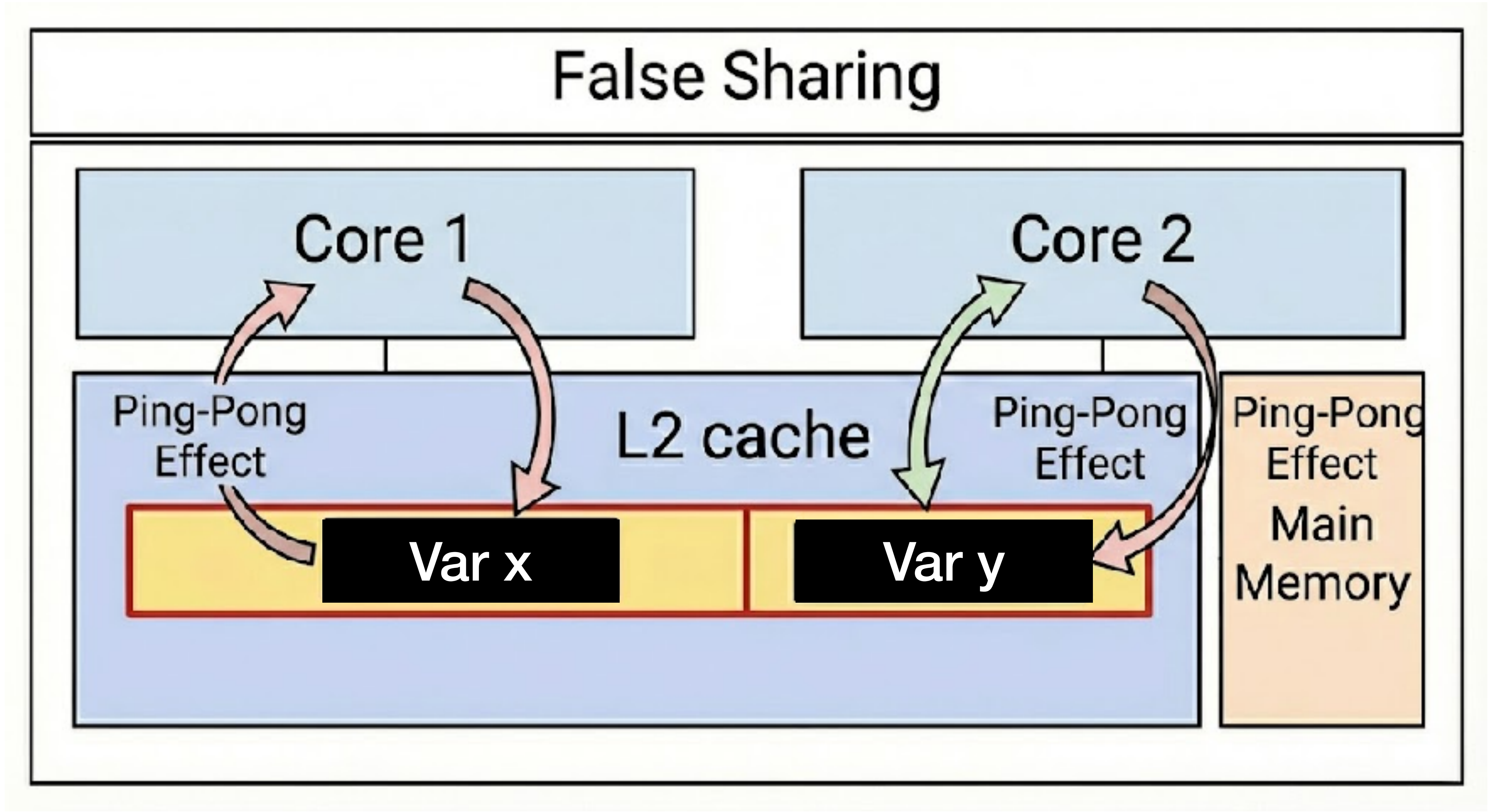
The Ping-Pong Effect

```
typedef struct {
    long x;   // Thread A increments x
    long y;   // Thread B increments y
} Counters;

Counters c;  // x and y likely on same cache line

// Thread A:
for (long i = 0; i < N; i++) c.x++;

// Thread B:
for (long i = 0; i < N; i++) c.y++;
```



- Repeated cache line invalidation
  - Main memory access is slow
- Severe performance degradation
- **FIX: Pad the cache lines**