CSC 256/456: Operating Systems

Multiprocessor Support

John Criswell
University of Rochester

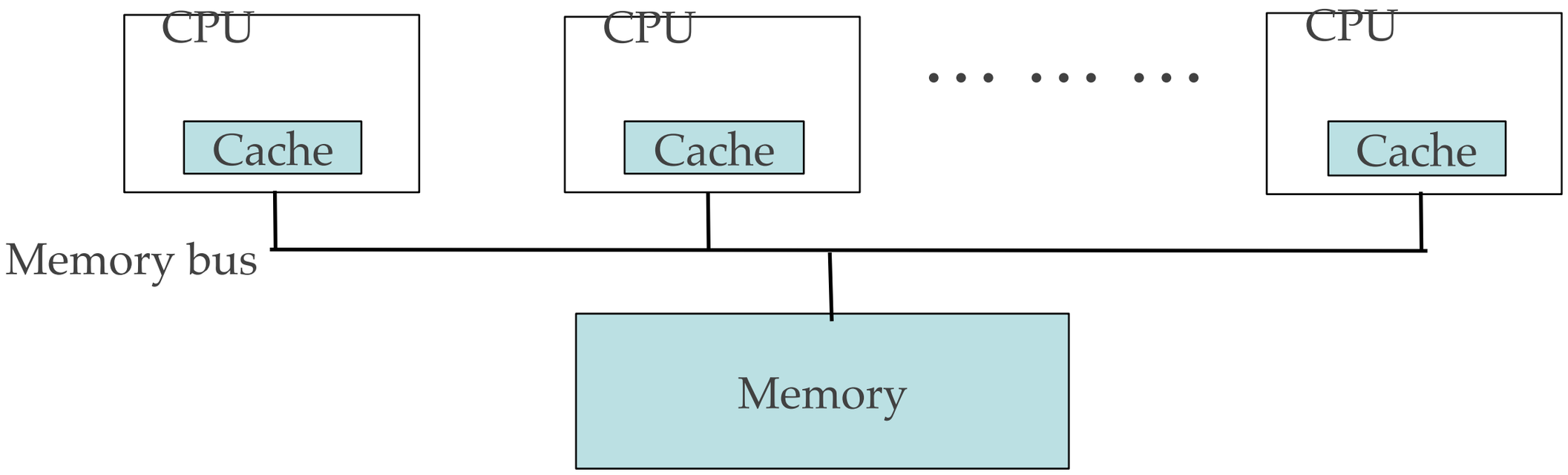
Outline

- Multiprocessor hardware
- Types of multi-processor workloads
- Operating system issues
  - Where to run the kernel
  - Synchronization
  - Where to run processes

Multiprocessor Hardware

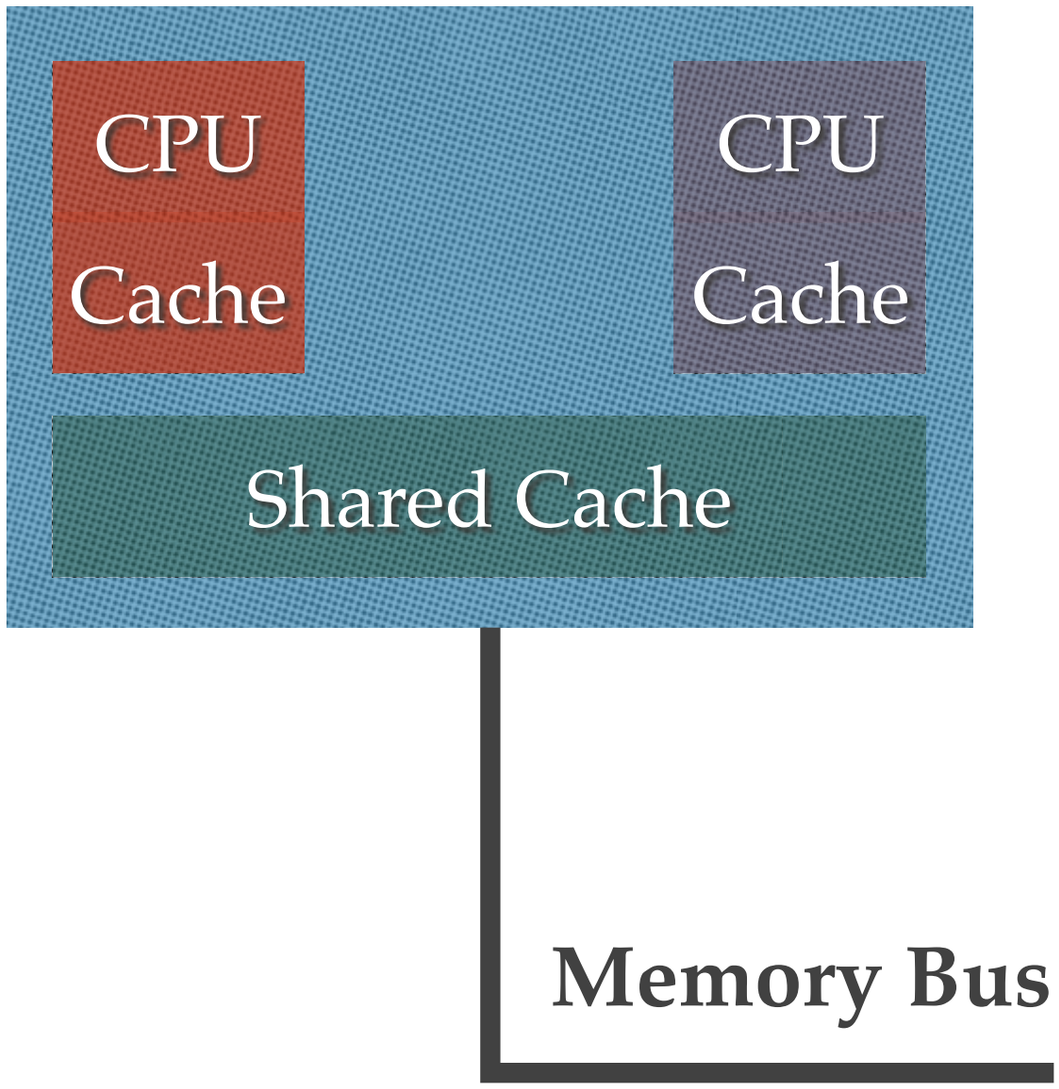
Multiprocessor Hardware

- System in which two or more CPUs share full access to the main memory
- Each CPU might have its own cache and the coherence among multiple caches is maintained



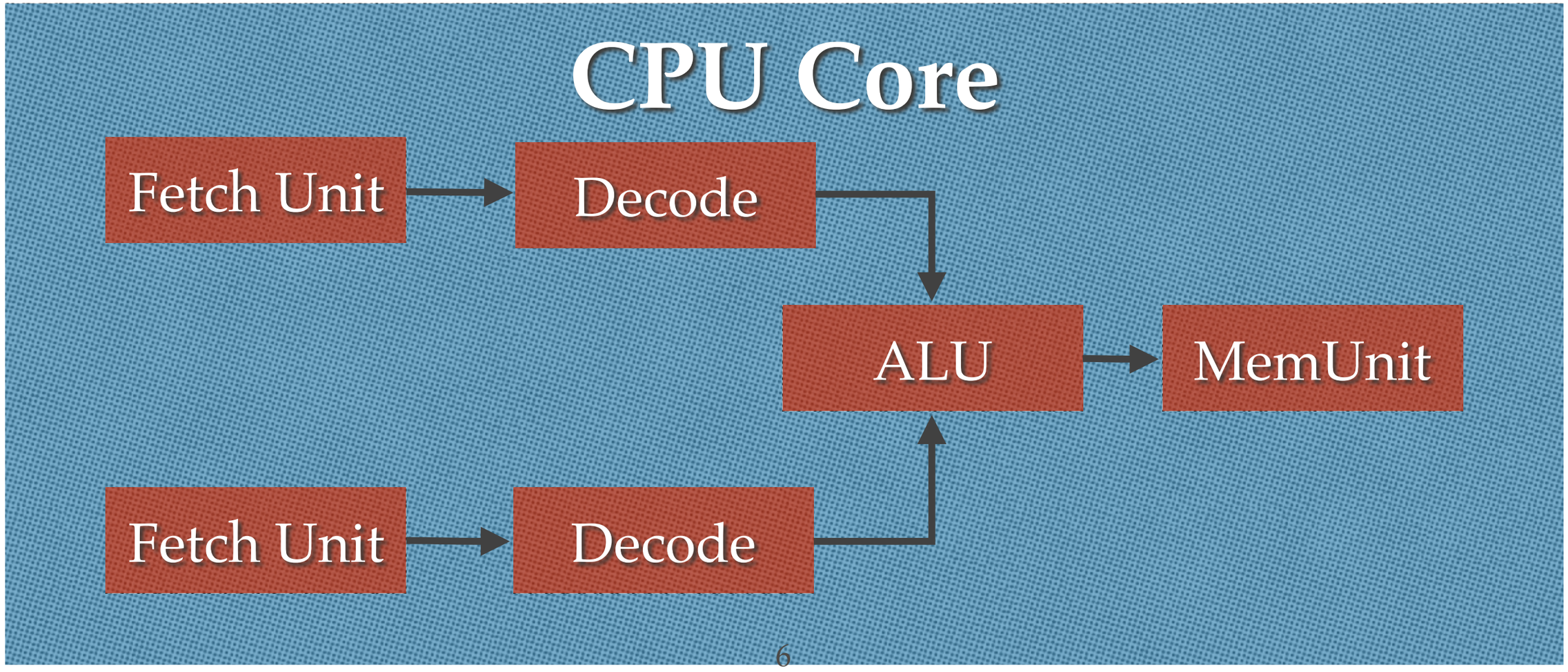
Multi-core Processor

- Multiple processors on “chip”
- Some caches shared
- Some not shared



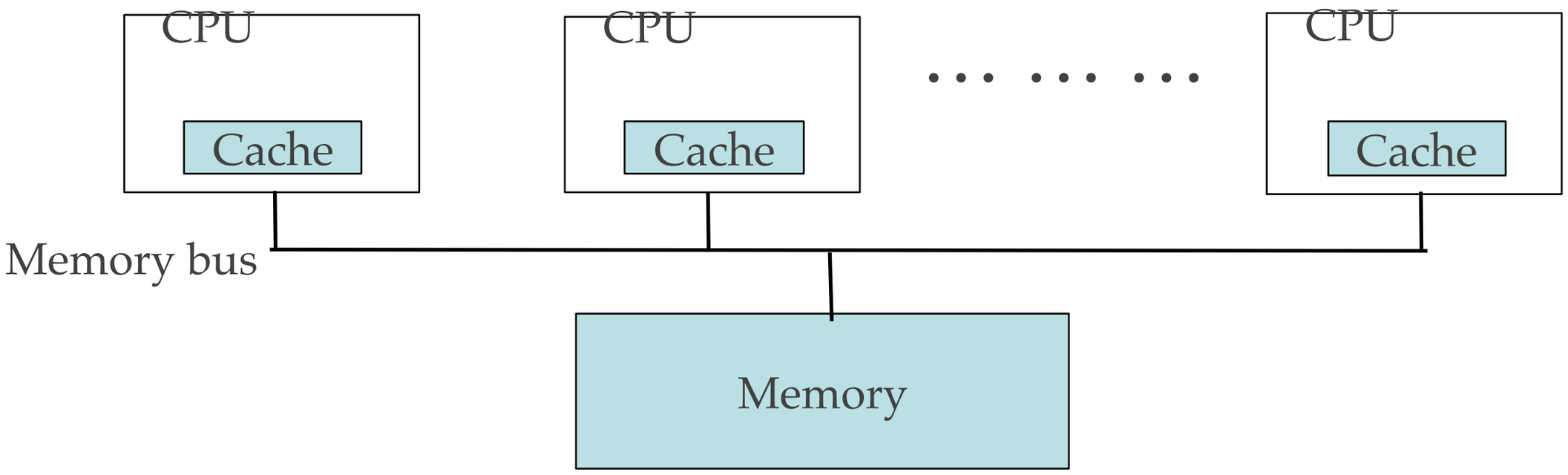
Hyper-Threading

- Replicate parts of processor; share other parts
- Create illusion that one core is two cores



Cache Coherency

- Ensure processors not operating with stale memory data
- Writes send out cache invalidation messages



Non Uniform Memory Access (NUMA)

- Memory clustered around CPUs
- For a given CPU
- Some memory is nearby (and fast)
- Other memory is far away (and slow)

Multiprocessor Workloads

Multiprogramming

- Non-cooperating processes with no communication
- Examples
  - Time-sharing systems
  - Multi-tasking single-user operating systems
  - make -j<very large number here>

Concurrent Servers

- Minimal communication between processes and threads
- Throughput usually the goal
- Examples
  - Web servers
  - Database servers

Parallel Programs

- Use parallelism to speed up computation
- Significant data sharing between processes and threads
- Examples
  - Gaussian Elimination
  - Matrix multiply

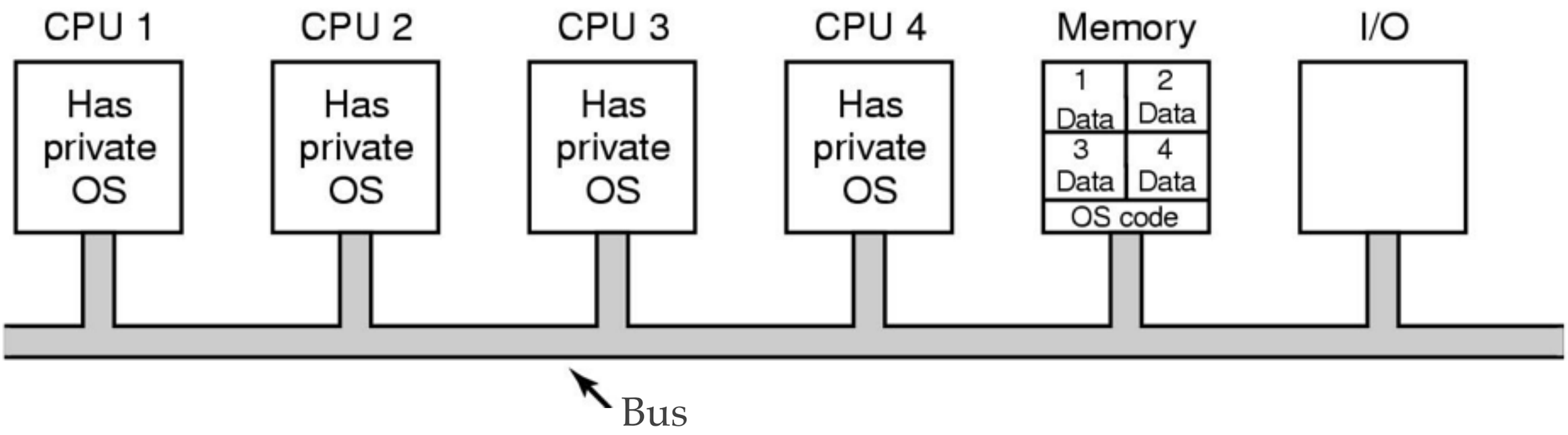
Operating System Issues

Three Challenges

- Where to run the OS
- How to do synchronization
- Where to schedule processes

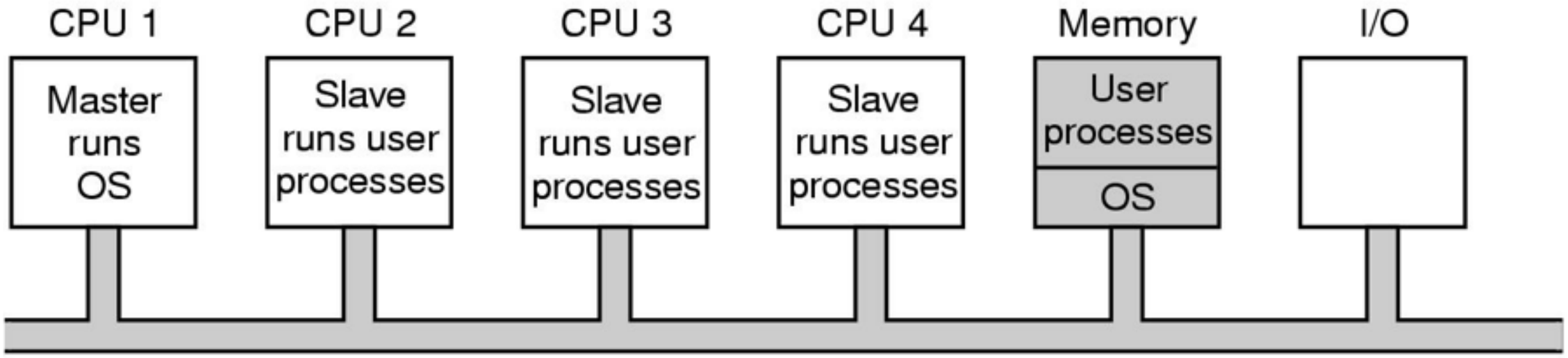
Where to Run the OS?

Multiprocessor OS



- Each CPU has its own operating system
  - quick to port from a single-processor OS
- Disadvantages
  - difficult to share things (processing cycles, memory, buffer cache)

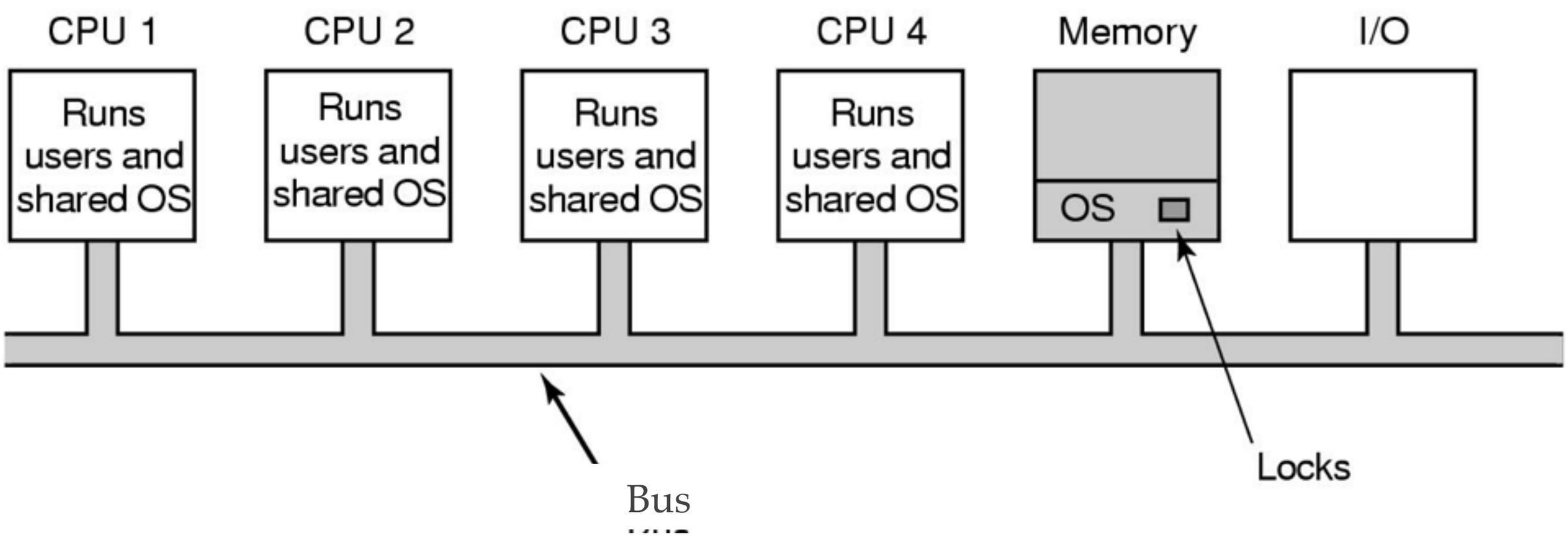
Multiprocessor OS – Master/Slave



- All operating system functionality goes to one CPU
  - no multiprocessor concurrency in the kernel
- Disadvantage
  - OS CPU consumption may be large so the OS CPU becomes the bottleneck (especially in a machine with many CPUs)

Multiprocessor OS – Shared OS

- A single OS instance may run on all CPUs
- The OS itself must handle multiprocessor synchronization
  - multiple OS instances from multiple CPUs may access shared data structure



Synchronization Issues

Synchronization

- Traditional atomic operations write memory
  - Atomic compare and swap
  - Atomic fetch and add
- This is very bad for multi-processors

Load-Linked and Store-Conditional

- Load-linked sets a bit in the cache line
- Cache invalidation clears bit
- Store-conditional checks that bit is still set in cache line
- Cleared bit means a cache invalidation occurred
- Which implies that another core wrote the memory
- Which implies that atomicity was violated

Performance Measures for Synchronization

- Latency
  - Cost of thread management under the best case assumption of no contention for locks
- Throughput
  - Rate at which threads can be created, started, and finished when there is contention

Synchronization (Fine/Coarse-Grain Locking)

- Fine-grain locking – lock only what is necessary for critical section
- Coarse-grain locking – locking large piece of code, much of which is unnecessary
- simplicity, robustness
- prevent simultaneous execution
  - simultaneous execution is not possible on uniprocessor anyway

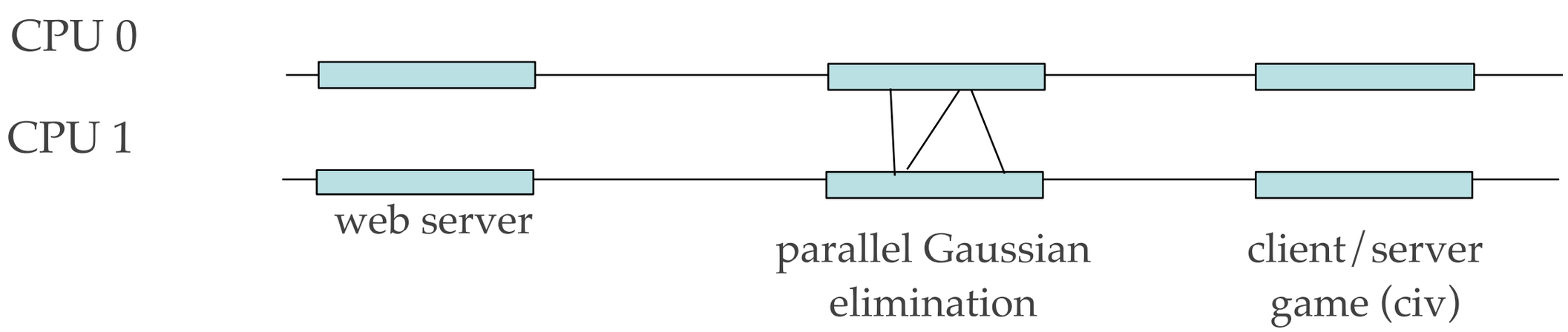
Synchronization Optimizations

- Avoid synchronization
  - Per-processor data structures
  - Lock-free data structures
- Use fine-grained locking to increase throughput
- Reuse (cache) data structures
  - Allocation/deallocation just a few pointer operations
  - Locks not held for very long

Where to Run Processes?

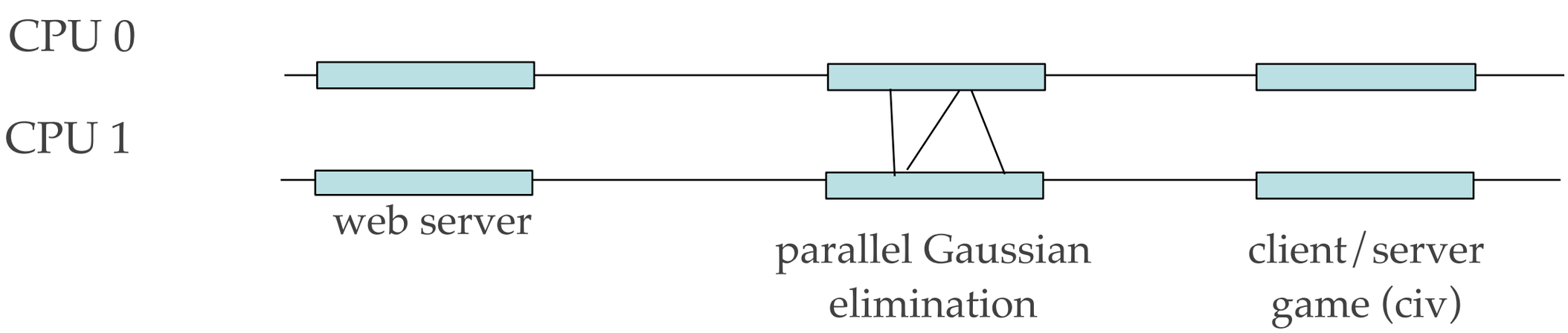
Multiprocessor Scheduling

- Affinity-based scheduling
  - Try to run each process on the processor that it last ran on
  - Takes advantage of cache locality



Multiprocessor Scheduling

- Gang/Cohort scheduling
  - Utilize all CPUs for one parallel/concurrent application at a time
  - Cache sharing and synchronization of parallel/concurrent applications



Resource Management To Date

- Capitalistic - generation of more requests results in more resource usage
  - Performance: resource contention can result in significantly reduced overall performance
  - Fairness: equal time slice does not necessarily guarantee equal progress

Fairness and Security Concerns

- Priority inversion
- Poor fairness among competing applications
- Information leakage at chip level
- Denial of service attack at chip level

Disclaimer

- Parts of the lecture slides contain original work by Andrew S. Tanenbaum. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).