Shared Memory Parallel Programming Basics

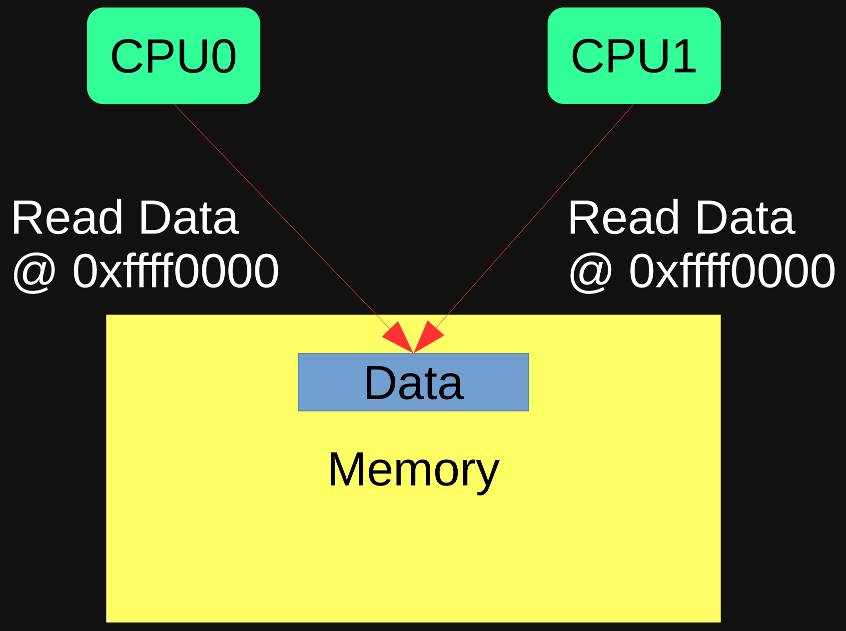
Wei Wang

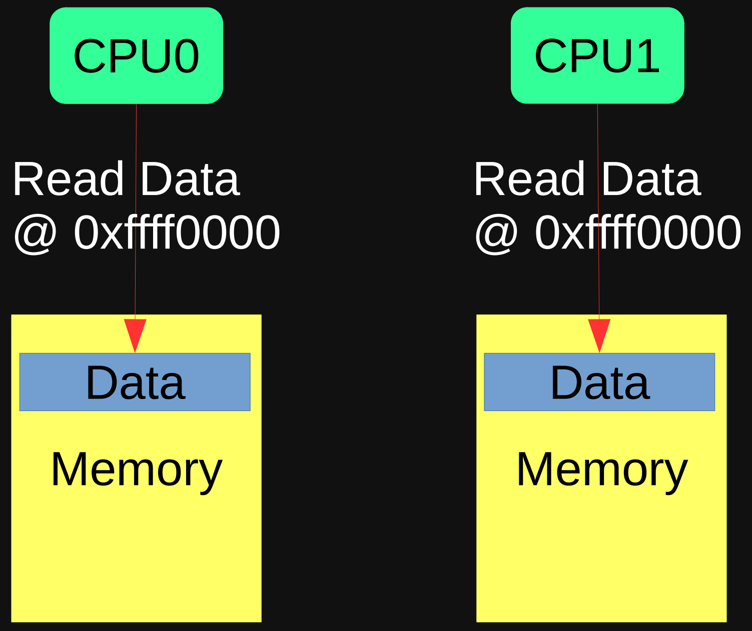
Shared Memory Parallel Programming

- Concurrent tasks share memory space
  - i.e., one task can directly read and write another task’s data
- Usually required shared-memory architectures
- Operating Systems (OS) support parallel programming with
  - Threads, Processes and mapping
  - Synchronizations for coordinately data accesses

Shared-Memory VS Distributed-Memory

- Shared-Memory: all processors access the same chunk of memory with the same address
- Distributed-memory: processors access different chunks of memory with the same address



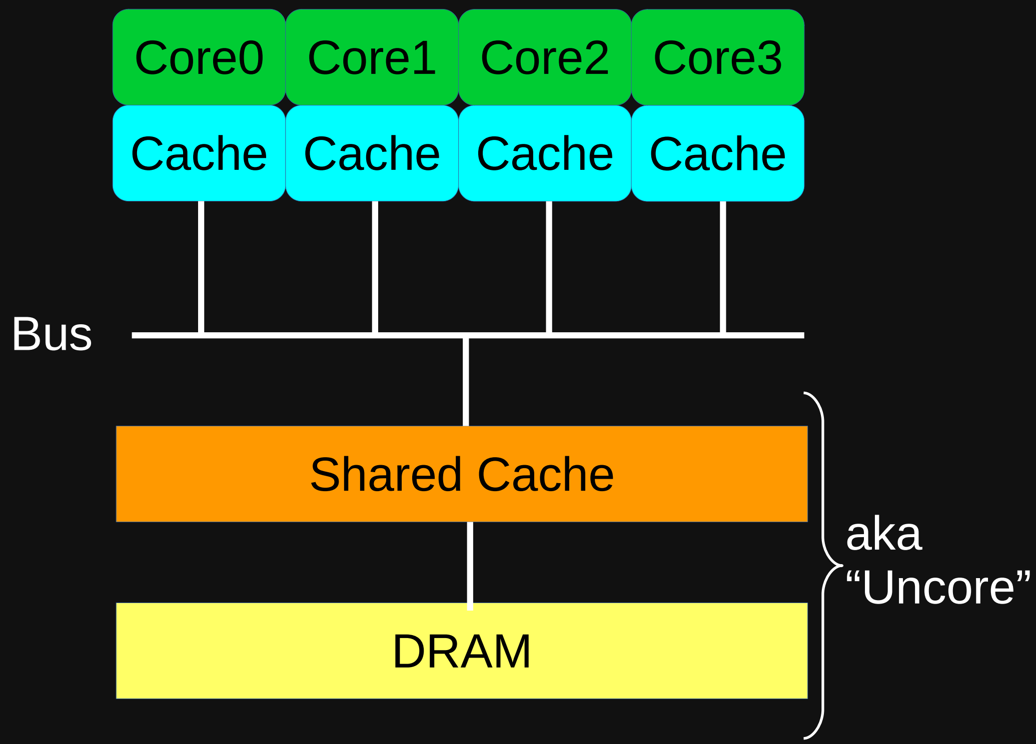


Shared-Memory Architectures

- Processors that physically or conceptually share memory chips (e.g., cache and DRAM)
- The common type of shared-memory architecture is called Symmetric multiprocessing (SMP),
  - All processors are directly connected to one memory
  - Typical example: Multi-core CPUs (UMA)
- Alternatively, for better scalability, Non-uniform Memory Architecture (NUMA) is used
  - Memory is partitioned and distributed among processors
  - Hardware provides an illusion that all processors are directly connected to all memory

Shared Memory Architecture: Multi-core CPUs

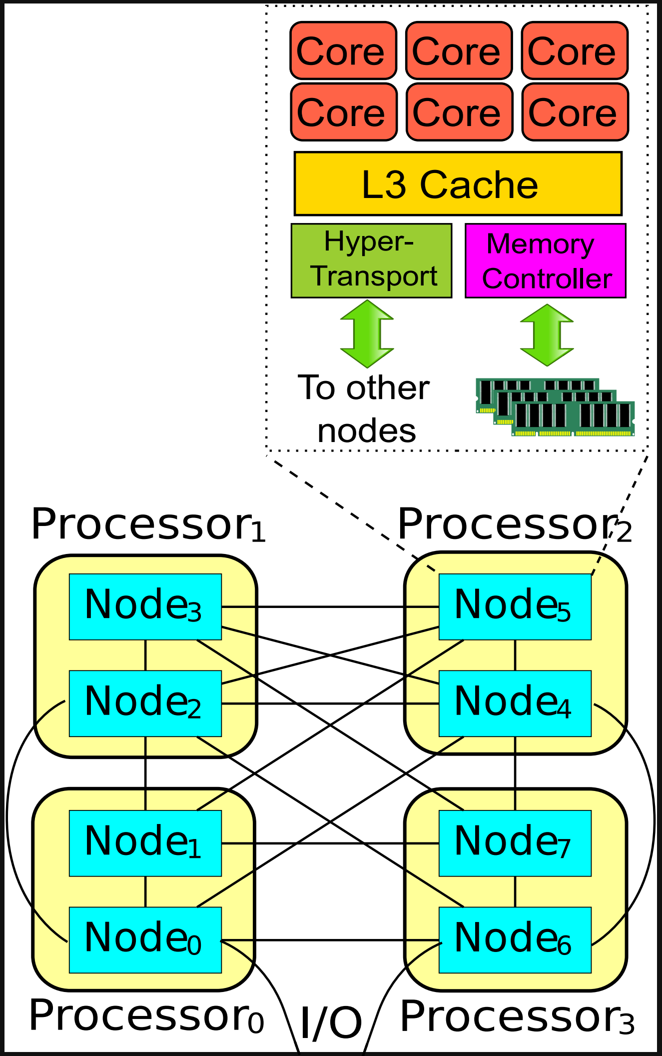
- Multiple cores on the same chip
- All cores share one last-level cache (LLC)
- Cores are independent to each other
- Each core has private caches
- All cores share all DRAM chips
  - Data placement is not important
- Cores send messages and data to each other through Bus to coordinate their computations
  - HW managed, no-user-involvement



Shared Memory Architecture: Non-Uniform Mem Arch (NUMA)

- Multiple independent CPUs
  - Nowadays, each CPU is usually a multi-core chip
- Each CPU has its own last-level cache and DRAM chips
  - Data placement is important
- CPUs are connected using inter-connections, e.g.,
  - Intel QuickPath Inter-connect (QPI)
  - AMD HyperTransport
- CPUs send messages and data to each other through inter-connections to coordinate
  - HW managed, no-user-involvement
  - Provides an illusion that all CPUs are directly connected to all DRAM chips

Shared Memory Architecture: NUMA cont'd

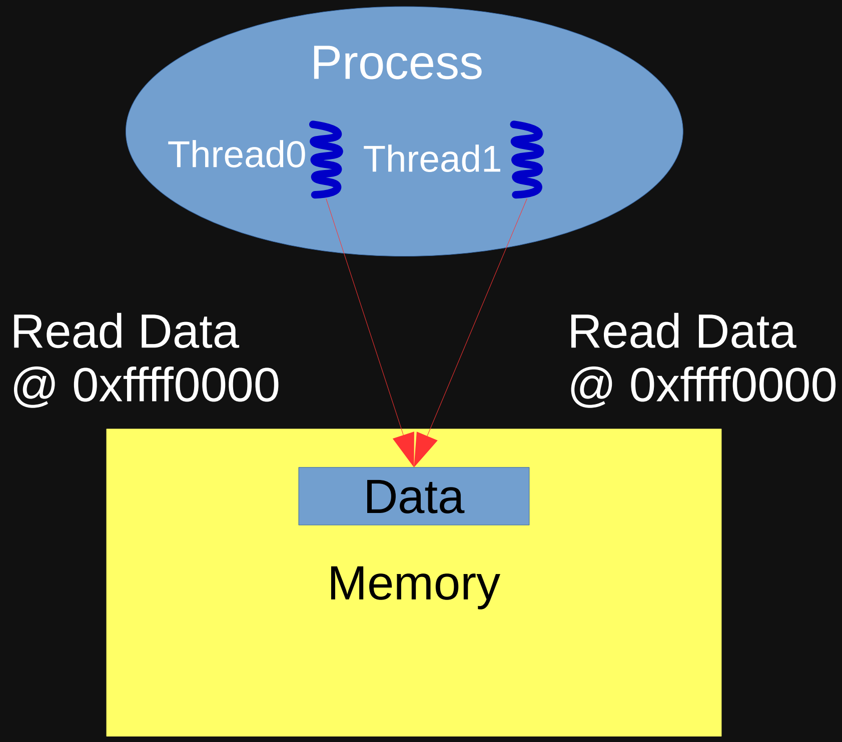


Operating System Support: Processes and Threads

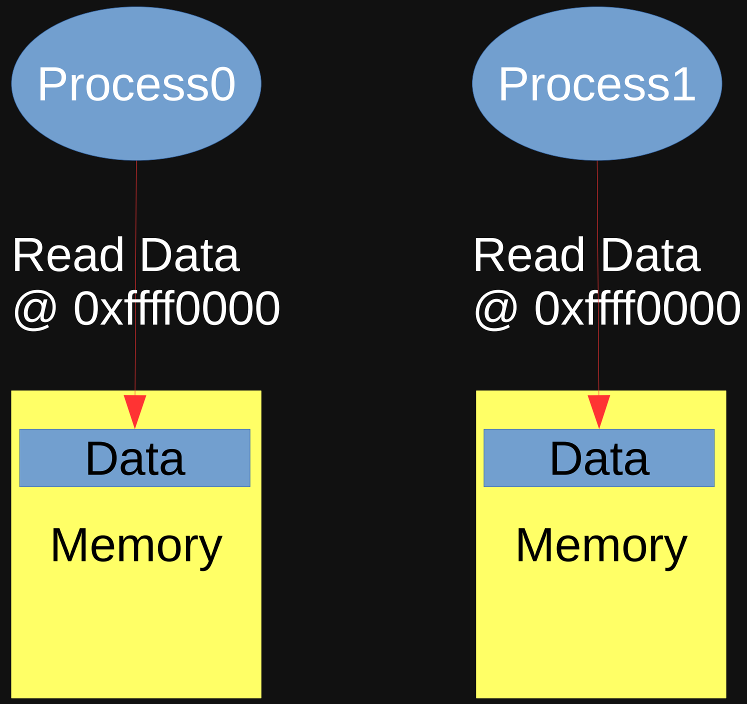
- Operating Systems (OS) provide basic supports for writing shared-memory parallel programs
- A Process is an instance of a computer program that is being executed.
- A Thread is an instance of a **sequential** computer program that is being executed.
  - Threads are the basic unit for scheduling in modern OS
  - A process contains at least one thread
  - A process may contain multiple threads for parallel execution
- Threads of the same processes share memory space; i.e., they accesses the same chunk of memory with the same address
  - Threading represents the OS support for shared-memory programming

Threads VS Processes

- Threads: shared memory space

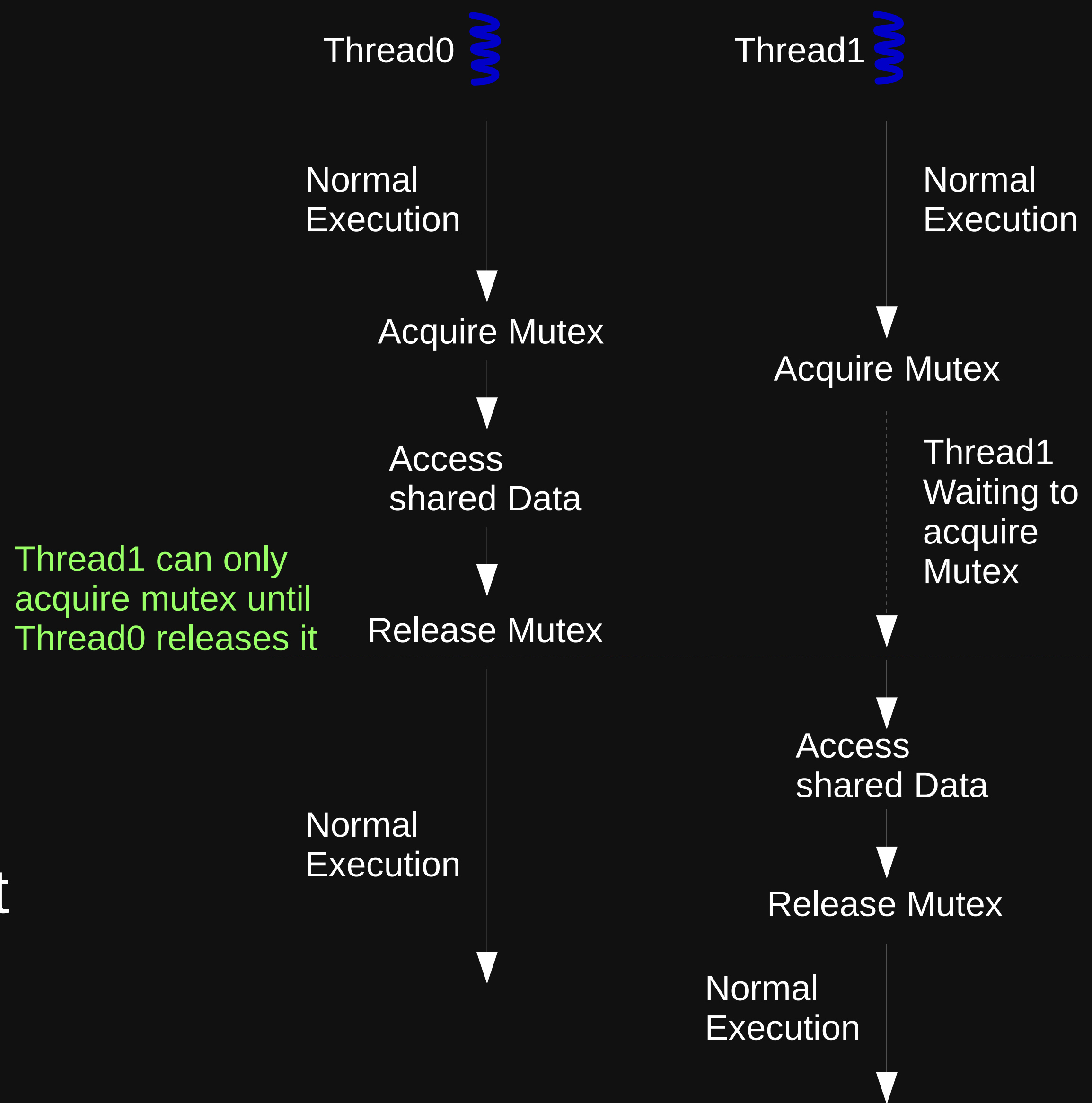


- Processes: do not share memory space



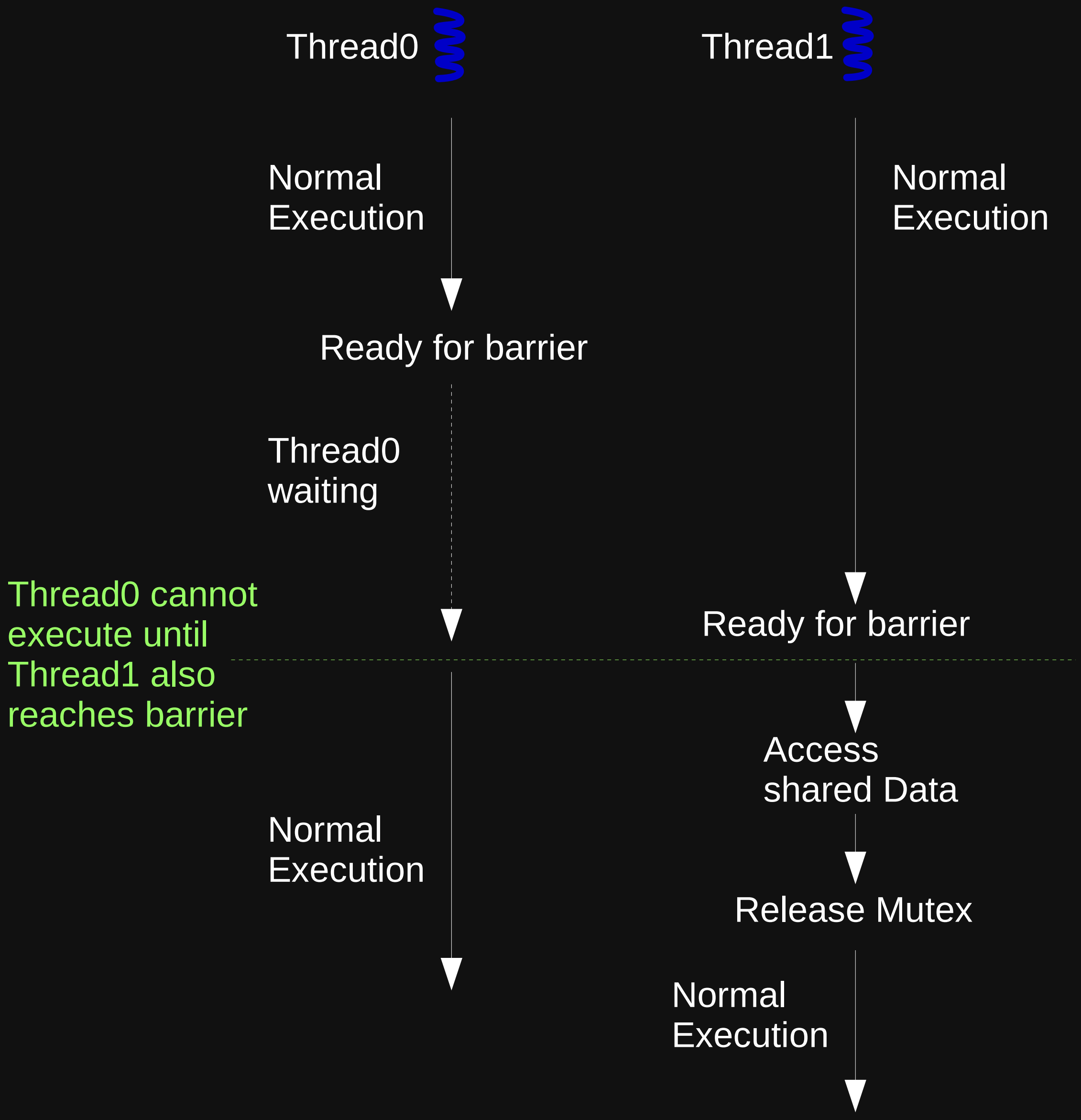
OS and HW Support: Synchronization Primitives

- Synchronization primitives help threads coordination accesses to shared data
- Mutex:
  - Ensures only one thread may read or write to a shared memory at a time



OS and HW Support: Synchronization Primitives cont’d

- Barriers:
  - any thread must stop at a barrier and cannot proceed until all other threads reach this barrier.



OS and HW Support: Synchronization Primitives cont’d

- There are more synchronization primitives:
  - Atomic operations (HW)
  - Semaphores / Locks (OS)
  - Monitor / Condition variables (OS)
  - We will learn them later