

# Lecture 18: Coherence and Synchronization

---

- Topics: directory-based coherence protocols, synchronization primitives (Sections 5.1-5.5)

# Cache Coherence Protocols

---

- Directory-based: A single location (directory) keeps track of the sharing status of a block of memory
- Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary
  - Write-invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
  - Write-update: when a processor writes, it updates other shared copies of that block

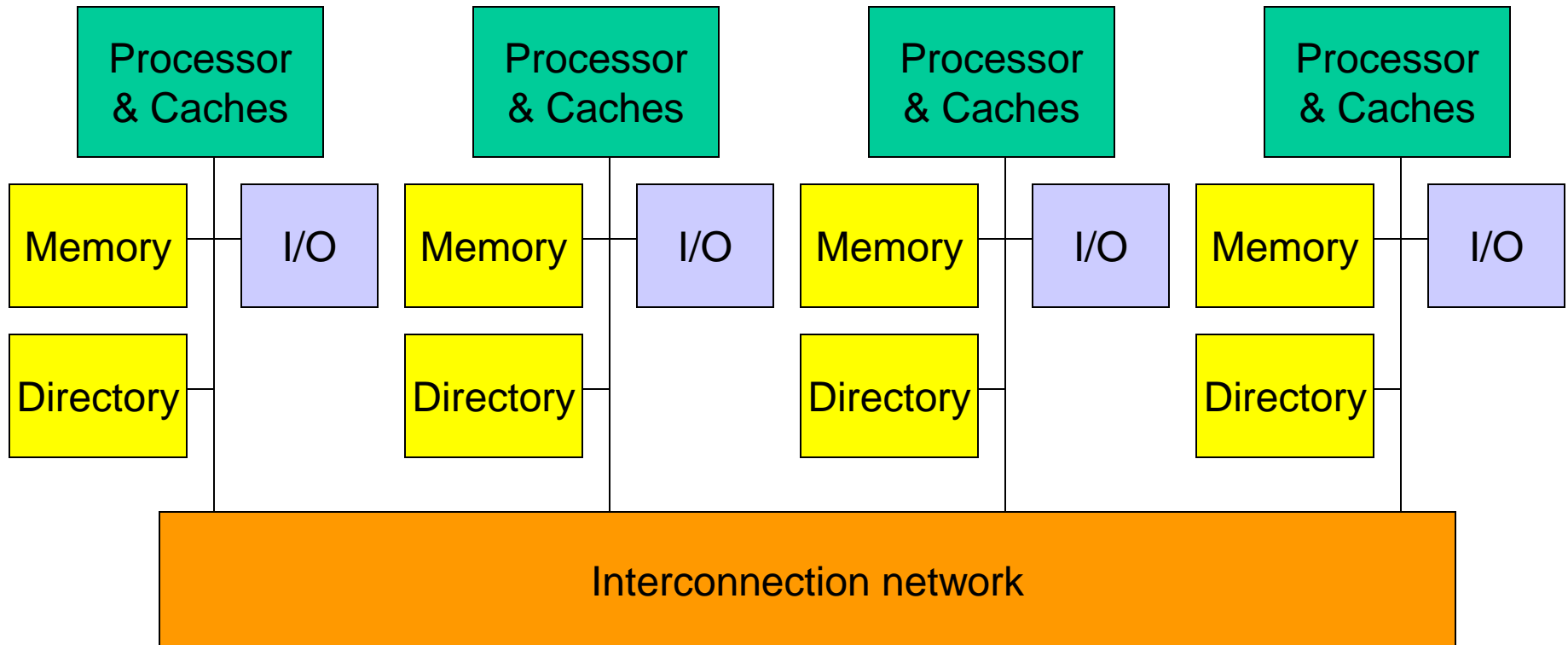
# Directory-Based Cache Coherence

---

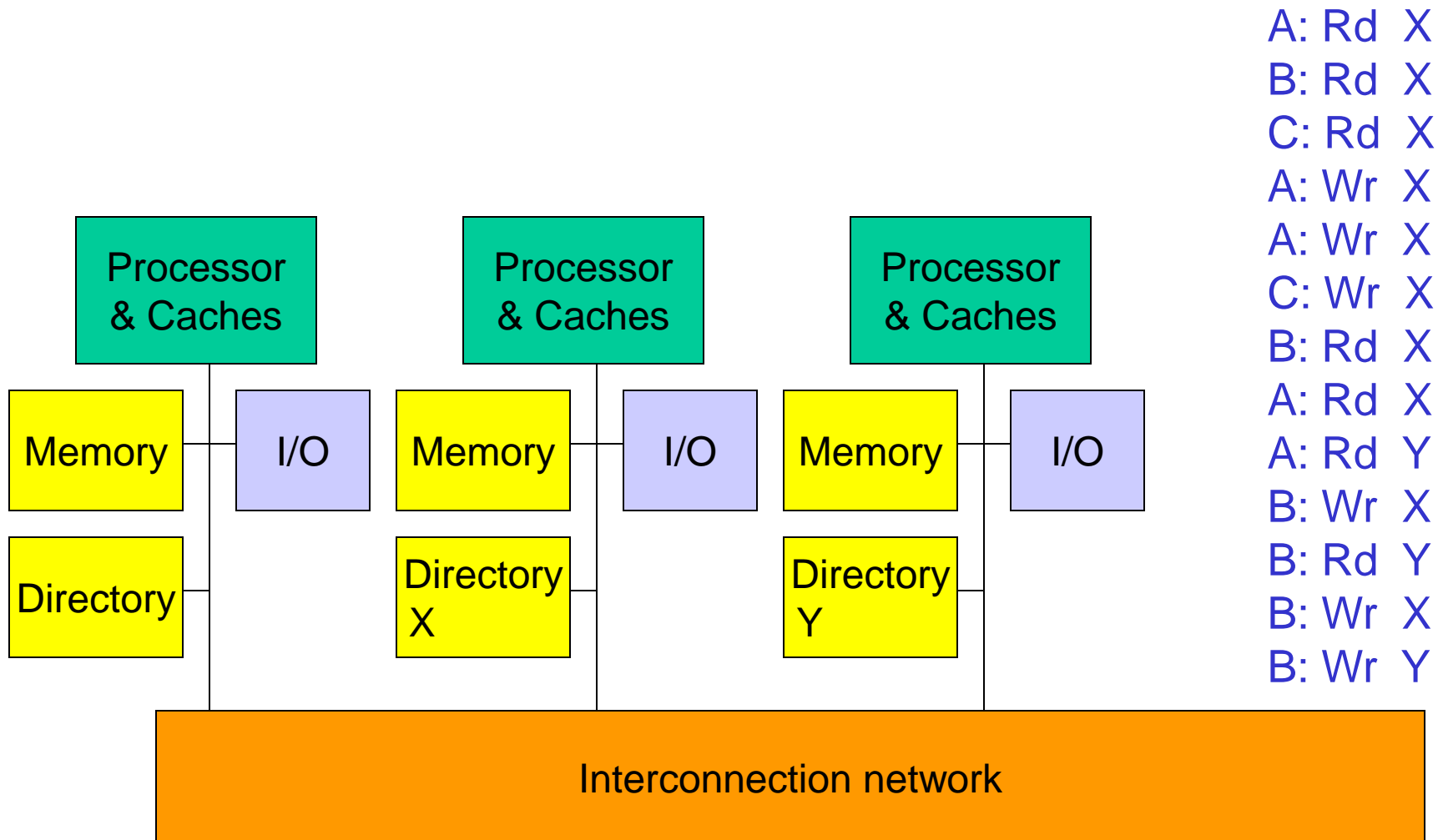
- The physical memory is distributed among all processors
- The directory is also distributed along with the corresponding memory
- The physical address is enough to determine the location of memory
- The (many) processing nodes are connected with a scalable interconnect (not a bus) – hence, messages are no longer broadcast, but routed from sender to receiver – since the processing nodes can no longer snoop, the directory keeps track of sharing state

# Distributed Memory Multiprocessors

---



# Directory-Based Example



# Cache Block States

---

- What are the different states a block of memory can have within the directory?
- Note that we need information for each cache so that invalidate messages can be sent
- The block state is also stored in the cache for efficiency
- The directory now serves as the arbitrator: if multiple write attempts happen simultaneously, the directory determines the ordering

# Directory Actions

---

- If block is in uncached state:
  - Read miss: send data, make block shared
  - Write miss: send data, make block exclusive
- If block is in shared state:
  - Read miss: send data, add node to sharers list
  - Write miss: send data, invalidate sharers, make excl
- If block is in exclusive state:
  - Read miss: ask owner for data, write to memory, send data, make shared, add node to sharers list
  - Data write back: write to memory, make uncached
  - Write miss: ask owner for data, write to memory, send data, update identity of new owner, remain exclusive

# Performance Improvements

---

- What determines performance on a multiprocessor:
  - What fraction of the program is parallelizable?
  - How does memory hierarchy performance change?
- New form of cache miss: coherence miss – such a miss would not have happened if another processor did not write to the same cache line
- False coherence miss: the second processor writes to a different word in the same cache line – this miss would not have happened if the line size equaled one word



# Constructing Locks

---

- Applications have phases (consisting of many instructions) that must be executed atomically, without other parallel processes modifying the data
- A lock surrounding the data/code ensures that only one program can be in a critical section at a time
- The hardware must provide some basic primitives that allow us to construct locks with different properties
- Lock algorithms assume an underlying cache coherence mechanism – when a process updates a lock, other processes will eventually see the update

# Synchronization

---

- The simplest hardware primitive that greatly facilitates synchronization implementations (locks, barriers, etc.) is an atomic read-modify-write
- Atomic exchange: swap contents of register and memory
- Special case of atomic exchange: test & set: transfer memory location into register and write 1 into memory
- lock:   t&s   register, location  
          bnz   register, lock  
          CS  
          st    location, #0

# Caching Locks

---

- Spin lock: to acquire a lock, a process may enter an infinite loop that keeps attempting a read-modify till it succeeds
- If the lock is in memory, there is heavy bus traffic → other processes make little forward progress
- Locks can be cached:
  - cache coherence ensures that a lock update is seen by other processors
  - the process that acquires the lock in exclusive state gets to update the lock first
  - spin on a local copy – the external bus sees little traffic

# Coherence Traffic for a Lock

---

- If every process spins on an exchange, every exchange instruction will attempt a write → many invalidates and the locked value keeps changing ownership
- Hence, each process keeps reading the lock value – a read does not generate coherence traffic and every process spins on its locally cached copy
- When the lock owner releases the lock by writing a 0, other copies are invalidated, each spinning process generates a read miss, acquires a new copy, sees the 0, attempts an exchange (requires acquiring the block in exclusive state so the write can happen), first process to acquire the block in exclusive state acquires the lock, others keep spinning

# Title

---

- Bullet