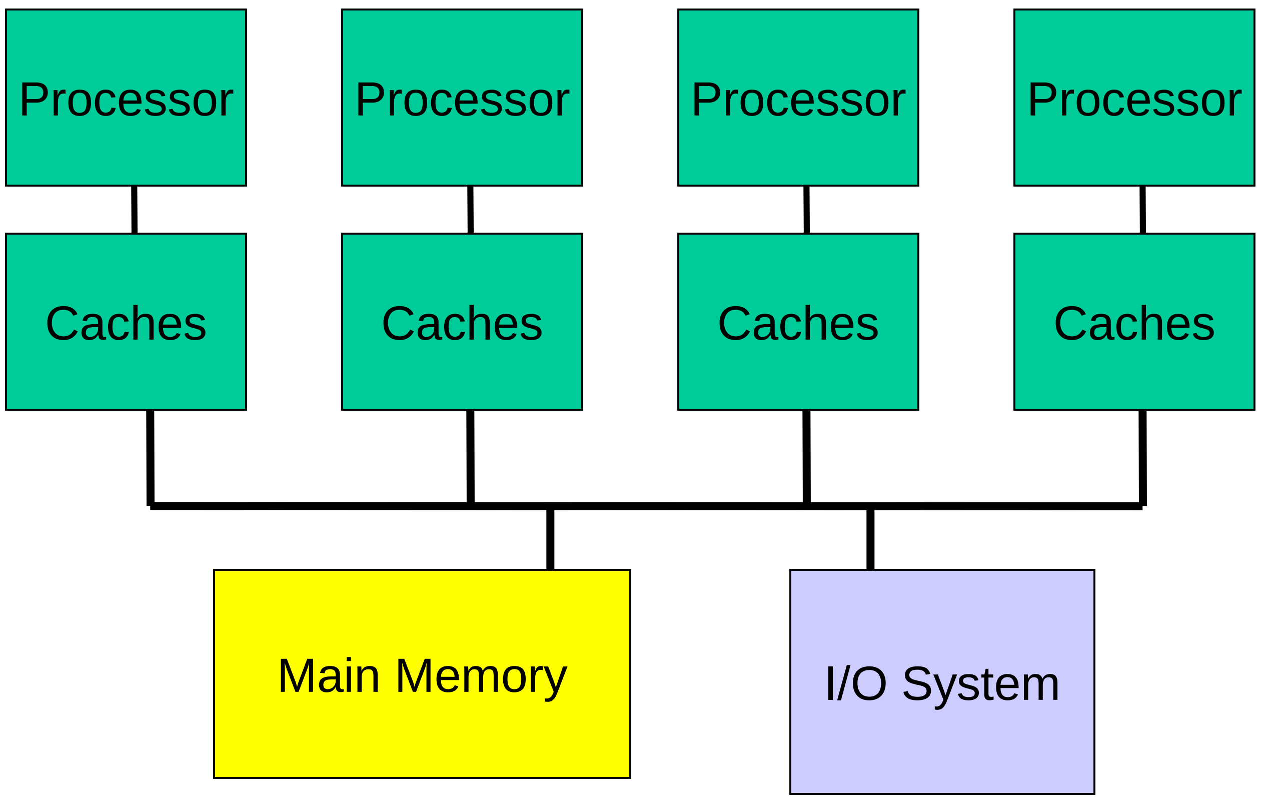
Lecture 24: Multiprocessors

- Today's topics:
  - Directory-based cache coherence protocol
  - Synchronization
  - Consistency

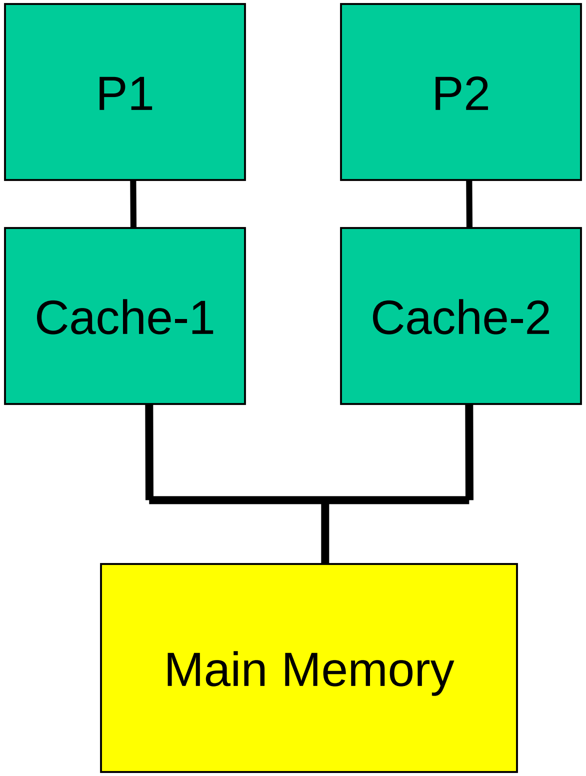
Snooping-Based Protocols

- Three states for a block: invalid, shared, modified
- A write is placed on the bus and sharers invalidate themselves
- The protocols are referred to as MSI, MESI, etc.



Example

- P1 reads X: not found in cache-1, request sent on bus, memory responds, X is placed in cache-1 in shared state
- P2 reads X: not found in cache-2, request sent on bus, everyone snoops this request, cache-1does nothing because this is just a read request, memory responds, X is placed in cache-2 in shared state



- P1 writes X: cache-1 has data in shared state (shared only provides read perms), request sent on bus, cache-2 snoops and then invalidates its copy of X, cache-1 moves its state to modified
- P2 reads X: cache-2 has data in invalid state, request sent on bus, cache-1 snoops and realizes it has the only valid copy, so it downgrades itself to shared state and responds with data, X is placed in cache-2 in shared state, memory is also updated

Example

Request	Cache Hit/Miss	Request on the bus	Who responds	State in Cache 1	State in Cache 2	State in Cache 3	State in Cache 4
				Inv	Inv	Inv	Inv
P1: Rd X	Miss	Rd X	Memory	S	Inv	Inv	Inv
P2: Rd X	Miss	Rd X	Memory	S	S	Inv	Inv
P2: Wr X	Perms Miss	Upgrade X	No response. Other caches invalidate.	Inv	M	Inv	Inv
P3: Wr X	Write Miss	Wr X	P2 responds	Inv	Inv	M	Inv
P3: Rd X	Read Hit	-	-	Inv	Inv	M	Inv
P4: Rd X	Read Miss	Rd X	P3 responds. Mem wrtbk	Inv	Inv	S	S

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

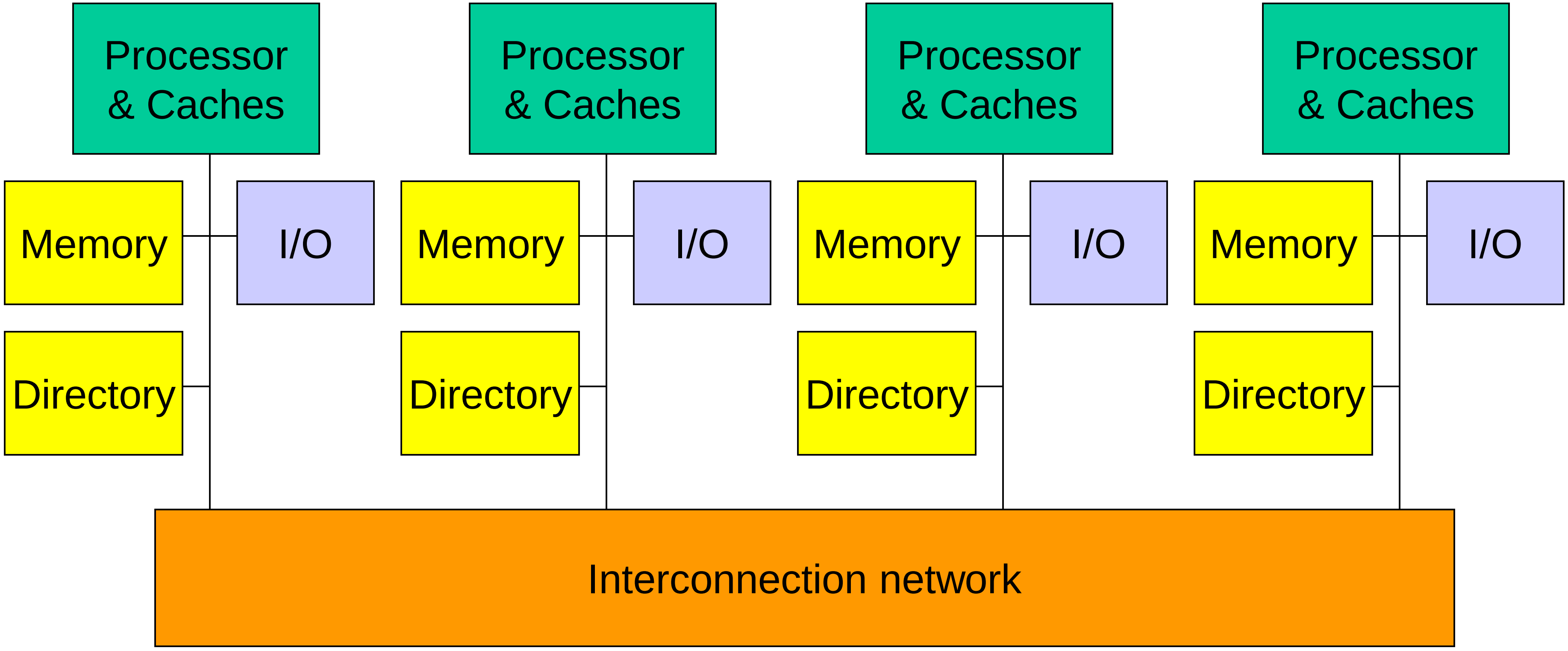
Coherence in Distributed Memory Multiprocs

- Distributed memory systems are typically larger → bus-based snooping may not work well

- Option 1: software-based mechanisms – message-passing systems or software-controlled cache coherence

- Option 2: hardware-based mechanisms – directory-based cache coherence

Distributed Memory Multiprocessors



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

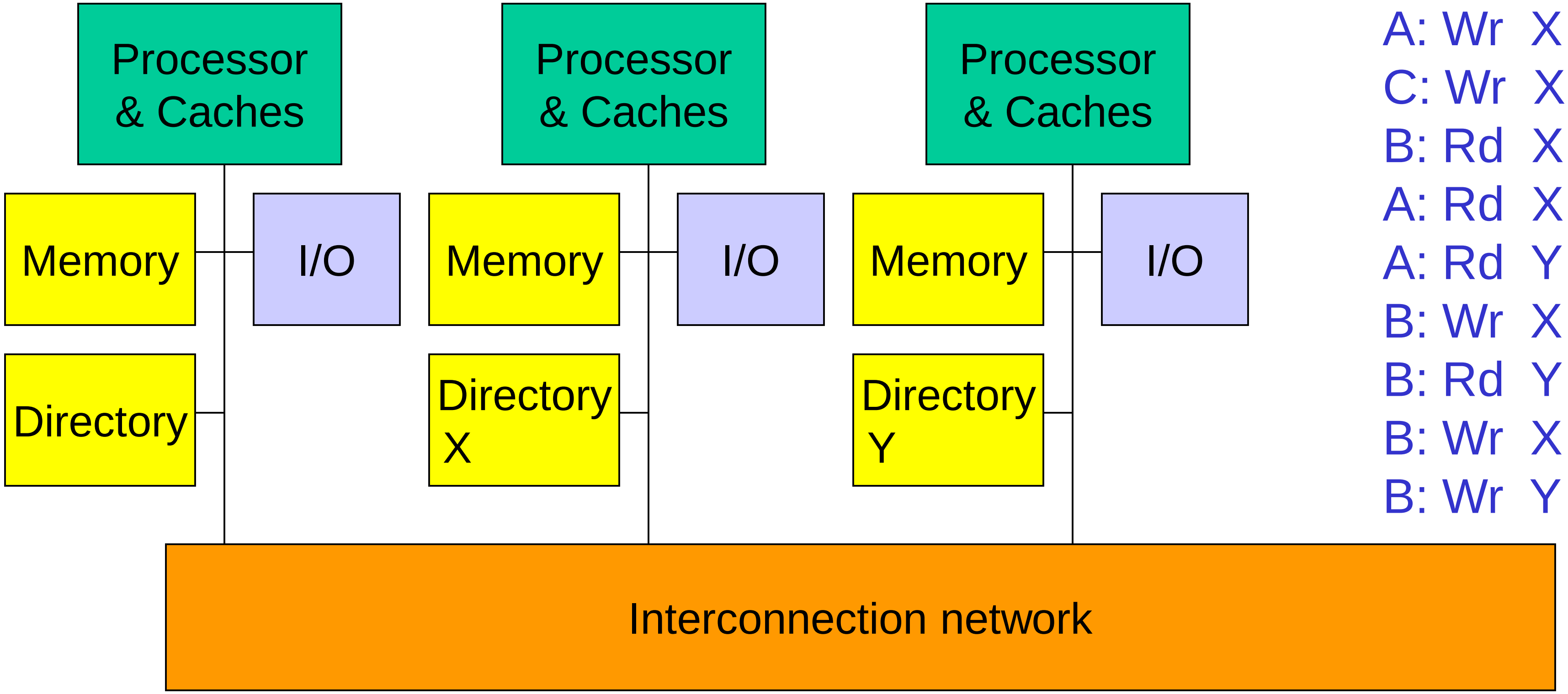
Cache Block States

- 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 directory now serves as the arbitrator: if multiple write attempts happen simultaneously, the directory determines the ordering

Directory-Based Example



A: Rd X
B: Rd X
C: Rd X
A: Wr X
A: Wr X
C: Wr X
B: Rd X
A: Rd X
A: Rd Y
B: Wr X
B: Rd Y
B: Wr X
B: Wr Y

Example

Request	Cache Hit/Miss	Messages	Dir State	State in C1	State in C2	State in C3	State in C4
				Inv	Inv	Inv	Inv
P1: Rd X	Miss	Rd-req to Dir. Dir responds.	X: S: 1	S	Inv	Inv	Inv
P2: Rd X	Miss	Rd-req to Dir. Dir responds.	X: S: 1, 2	S	S	Inv	Inv
P2: Wr X	Perms Miss	Upgr-req to Dir. Dir sends INV to P1. P1 sends ACK to Dir. Dir grants perms to P2.	X: M: 2	Inv	M	Inv	Inv
P3: Wr X	Write Miss	Wr-req to Dir. Dir fwds request to P2. P2 sends data to Dir. Dir sends data to P3.	X: M: 3	Inv	Inv	M	Inv
P3: Rd X	Read Hit	-	-	Inv	Inv	M	Inv
P4: Rd X	Read Miss	Rd-req to Dir. Dir fwds request to P3. P3 sends data to Dir. Memory wrtbk. Dir sends data to P4.	X: S: 3, 4	Inv	Inv	S	S

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

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

Bank balance
$1000

Parallel (unlocked) banking transactions

Rd $1000
Add $100
Wr $1100

Rd $1000
Add $200
Wr $1200

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 (if memory has 0, lock is free)

- lock:

```
lock:   t&s    register, location
        bnz    register, lock
        CS
        st     location, #0
```

When multiple parallel threads execute this code, only one will be able to enter CS

Coherence Vs. Consistency

- Recall that coherence guarantees (i) write propagation (a write will eventually be seen by other processors), and (ii) write serialization (all processors see writes to the same location in the same order)

- The consistency model defines the ordering of writes and reads to different memory locations – the hardware guarantees a certain consistency model and the programmer attempts to write correct programs with those assumptions

Consistency Example

- Consider a multiprocessor with bus-based snooping cache coherence

```
Initially A = B = 0
  P1                P2
A ← 1             B ← 1
 …                 …
if (B == 0)       if (A == 0)
  Crit.Section      Crit.Section
```

Consistency Example

- Consider a multiprocessor with bus-based snooping cache coherence

```
Initially A = B = 0
  P1                 P2
A ← 1              B ← 1
 …                   …
if (B == 0)        if (A == 0)
  Crit.Section       Crit.Section
```

The programmer expected the above code to implement a lock – because of ooo, both processors can enter the critical section

The consistency model lets the programmer know what assumptions they can make about the hardware's reordering capabilities

Sequential Consistency

- A multiprocessor is sequentially consistent if the result of the execution is achieveable by maintaining program order within a processor and interleaving accesses by different processors in an arbitrary fashion

- The multiprocessor in the previous example is not sequentially consistent

- Can implement sequential consistency by requiring the following: program order, write serialization, everyone has seen an update before a value is read – very intuitive for the programmer, but extremely slow

Title

- Bullet