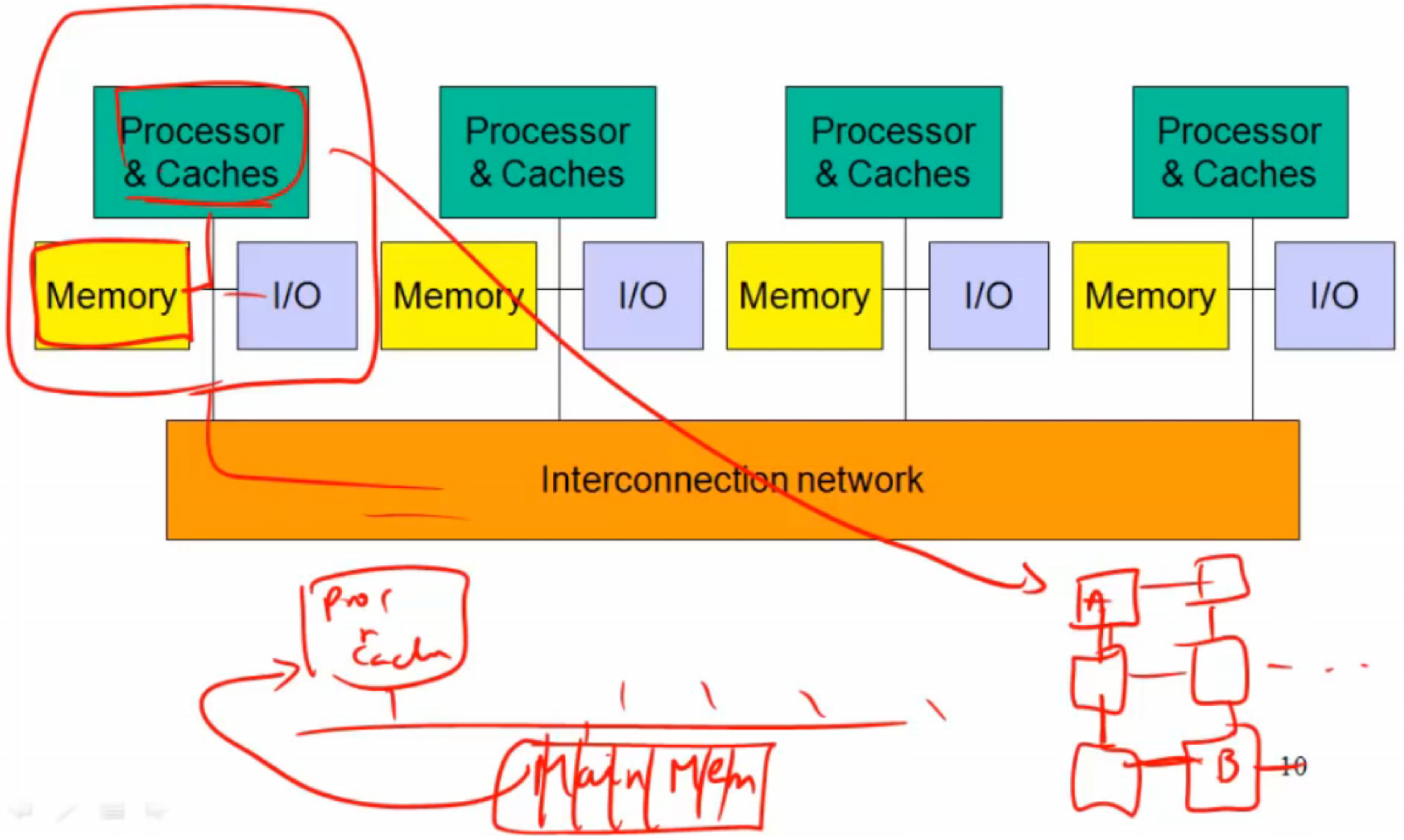
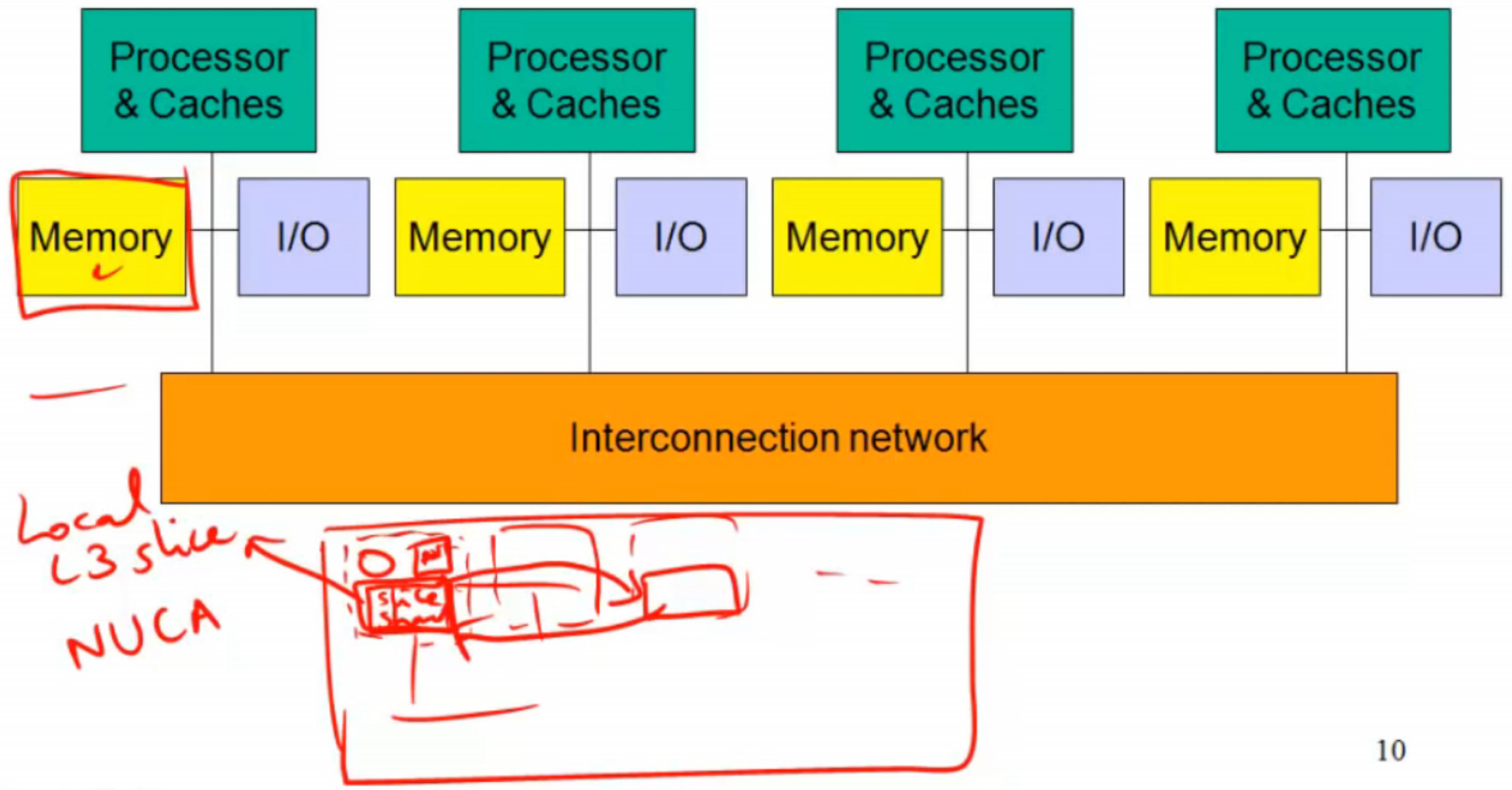
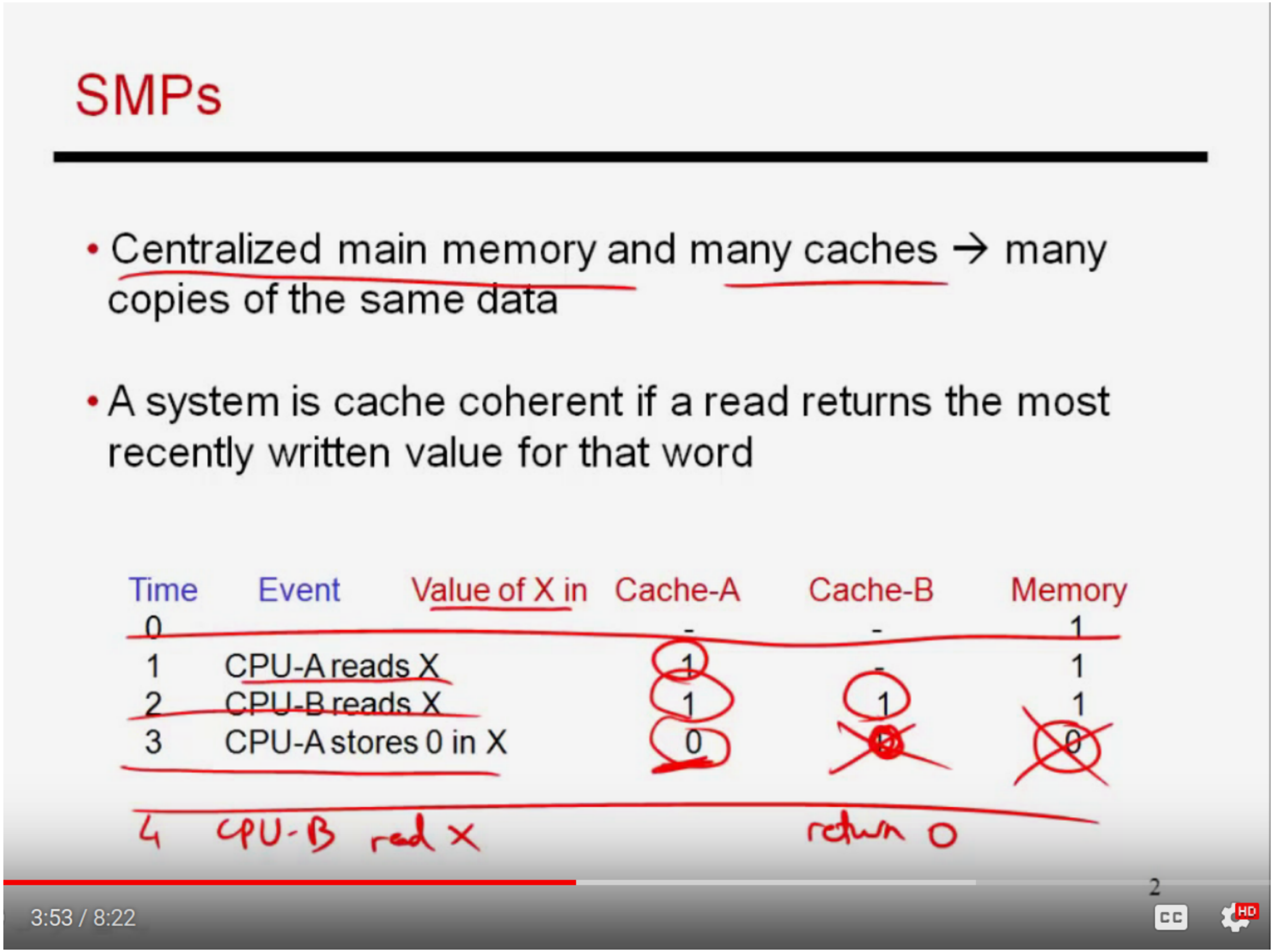
Distributed Memory Multiprocessors



Distributed Memory Multiprocessors





SMPs

- Centralized main memory and many caches → many copies of the same data
- A system is cache coherent if a read returns the most recently written value for that word

Time	Event	Value of X in		
		Cache-A	Cache-B	Memory
0		-	-	1
1	CPU-A reads X	1	-	1
2	CPU-B reads X	1	1	1
3	CPU-A stores 0 in X	0	~~1~~	~~0~~
4	CPU-B read X		return 0	

Cache Coherence

P1 $x \leftarrow 5$

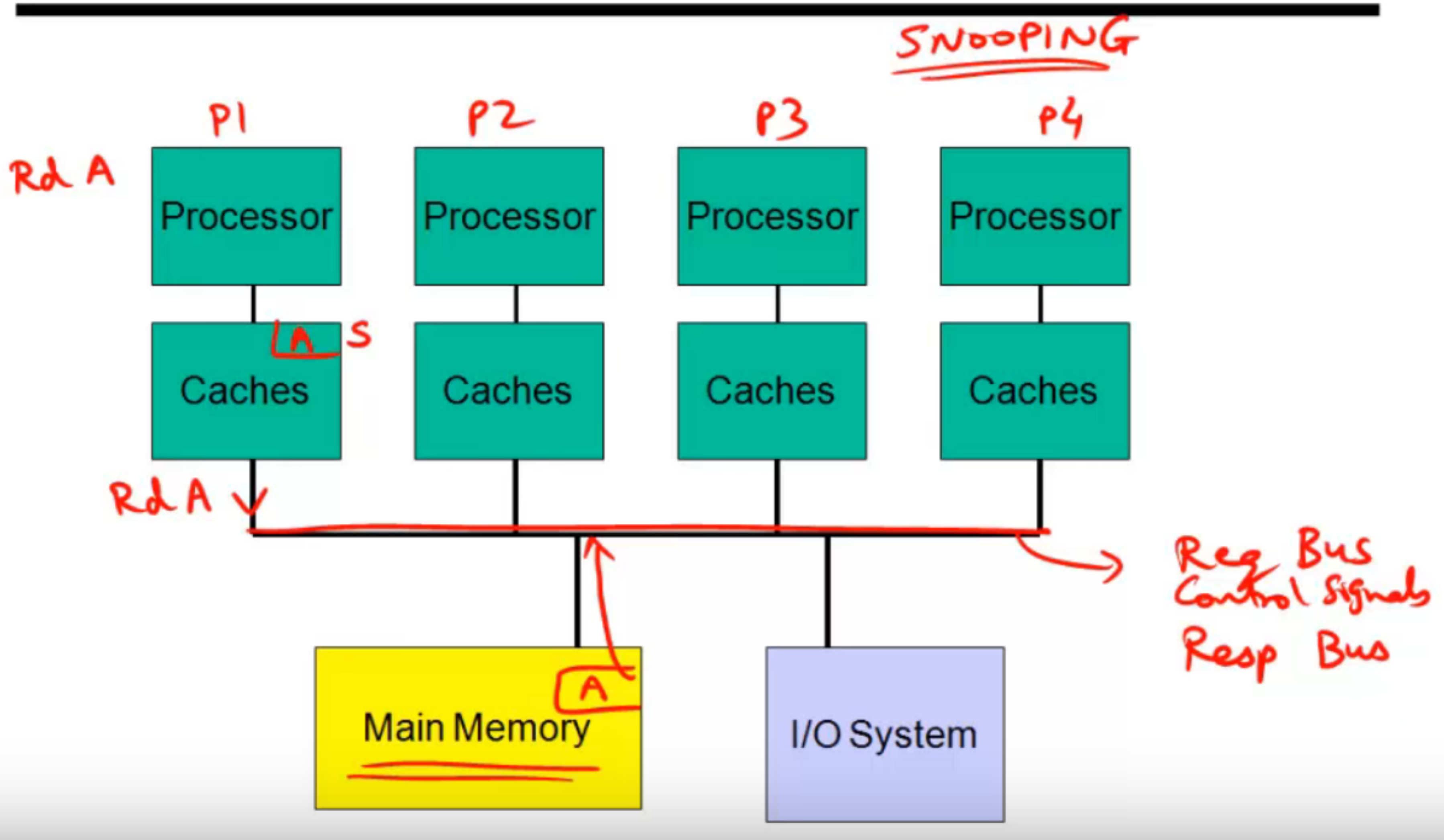
P2 $x \leftarrow 7$

P3 $x \leftarrow 7$ $x \leftarrow 5$

A memory system is coherent if:

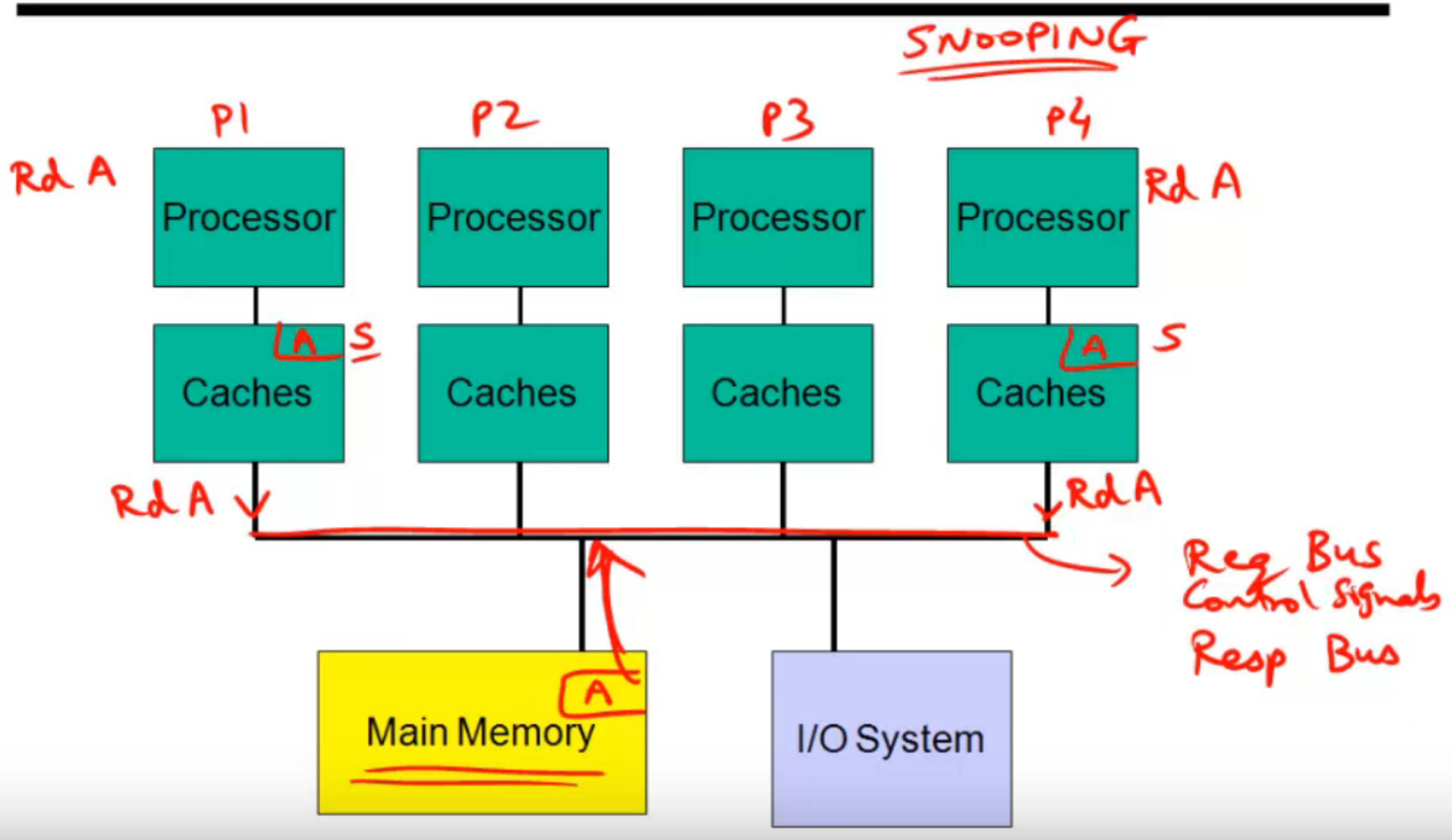
- Write propagation: P1 writes to X, sufficient time elapses, P2 reads X and gets the value written by P1
- Write serialization: Two writes to the same location by two processors are seen in the same order by all processors
- The memory *consistency* model defines "time elapsed" before the effect of a processor is seen by others and the ordering with R/W to other locations (loosely speaking – more later)

SMPs or Centralized Shared-Memory



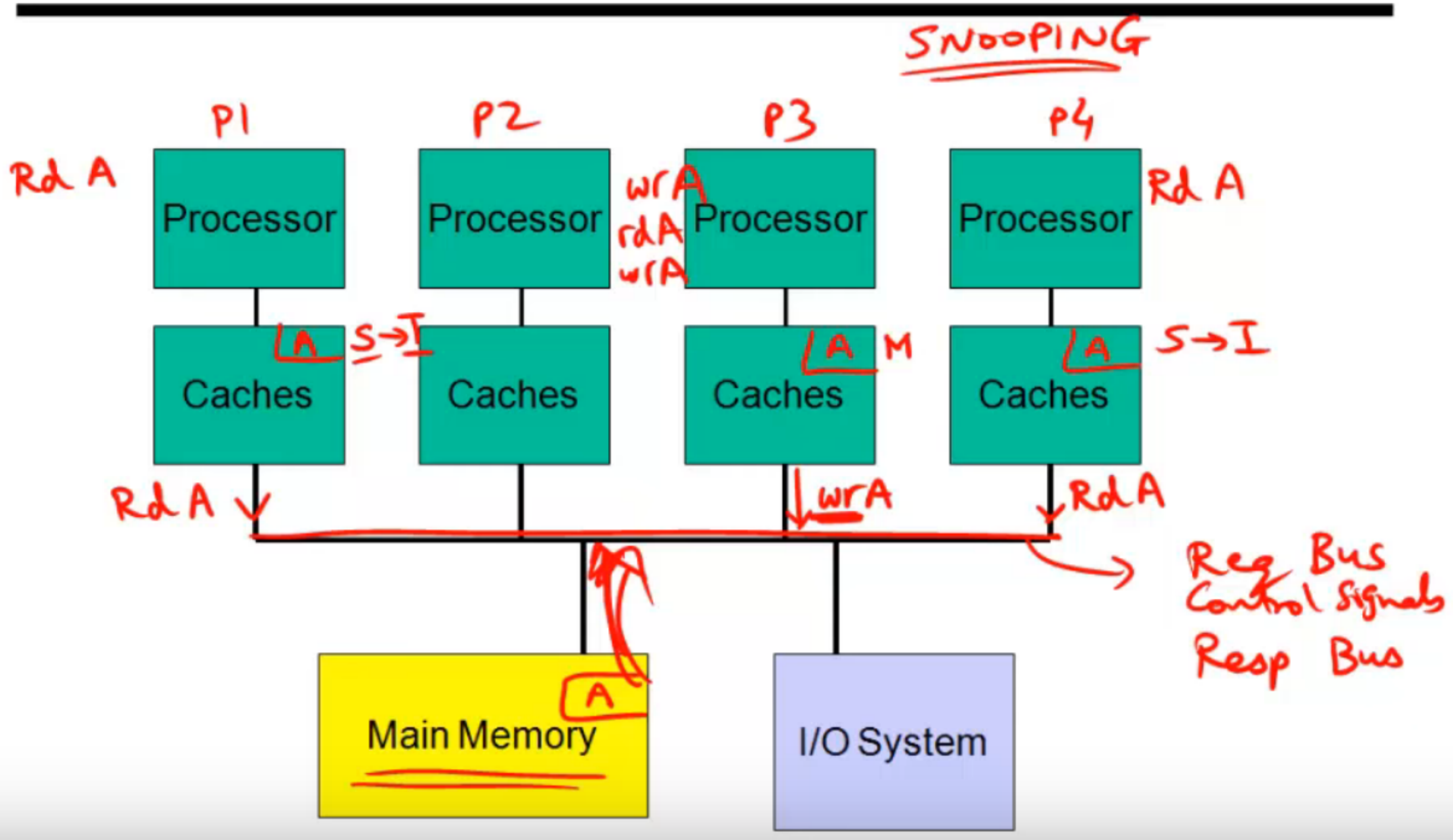
3:22 / 10:08

SMPs or Centralized Shared-Memory

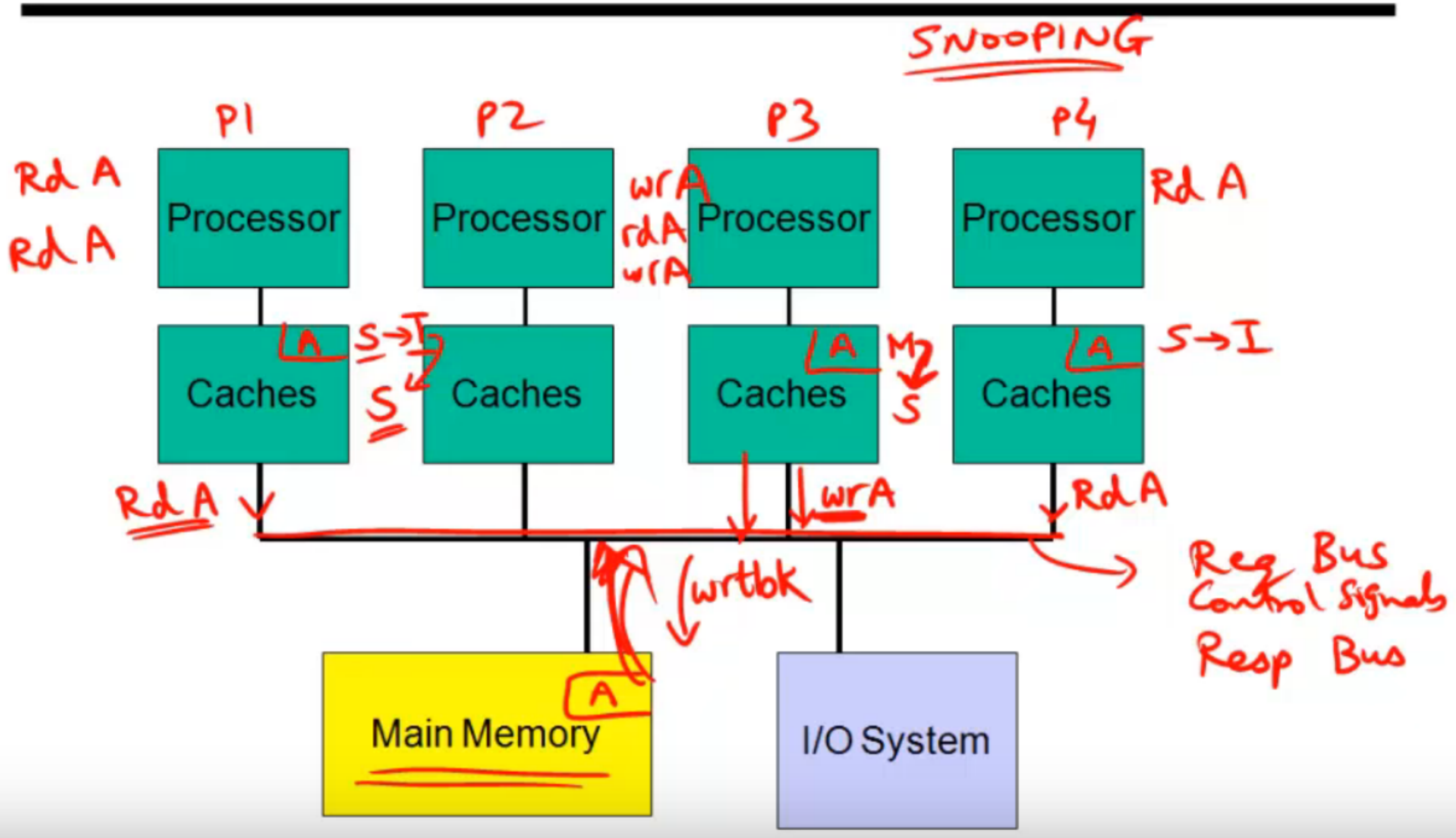


5:16 / 10:08

SMPs or Centralized Shared-Memory

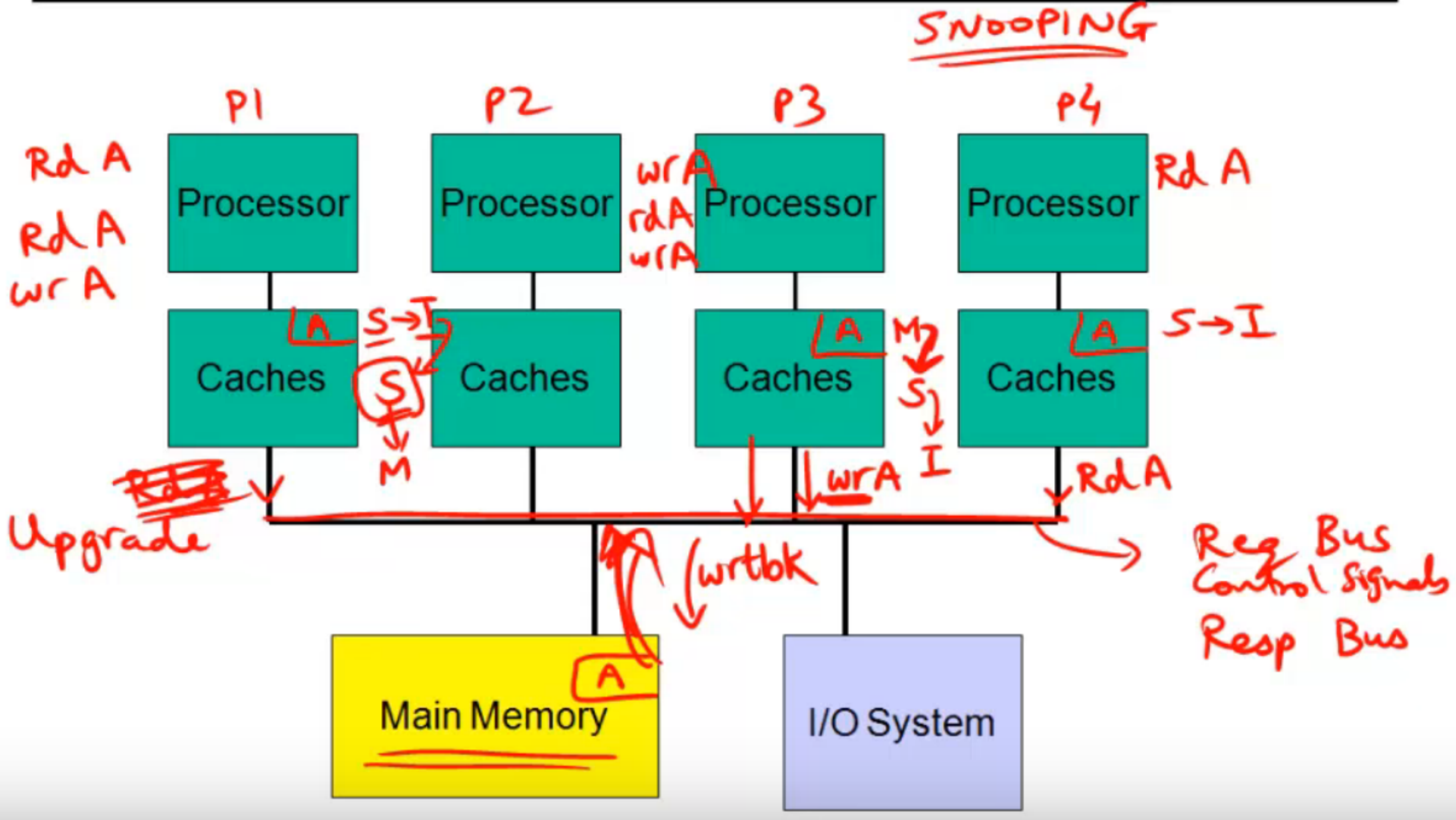


SMPs or Centralized Shared-Memory



10:04 / 10:08

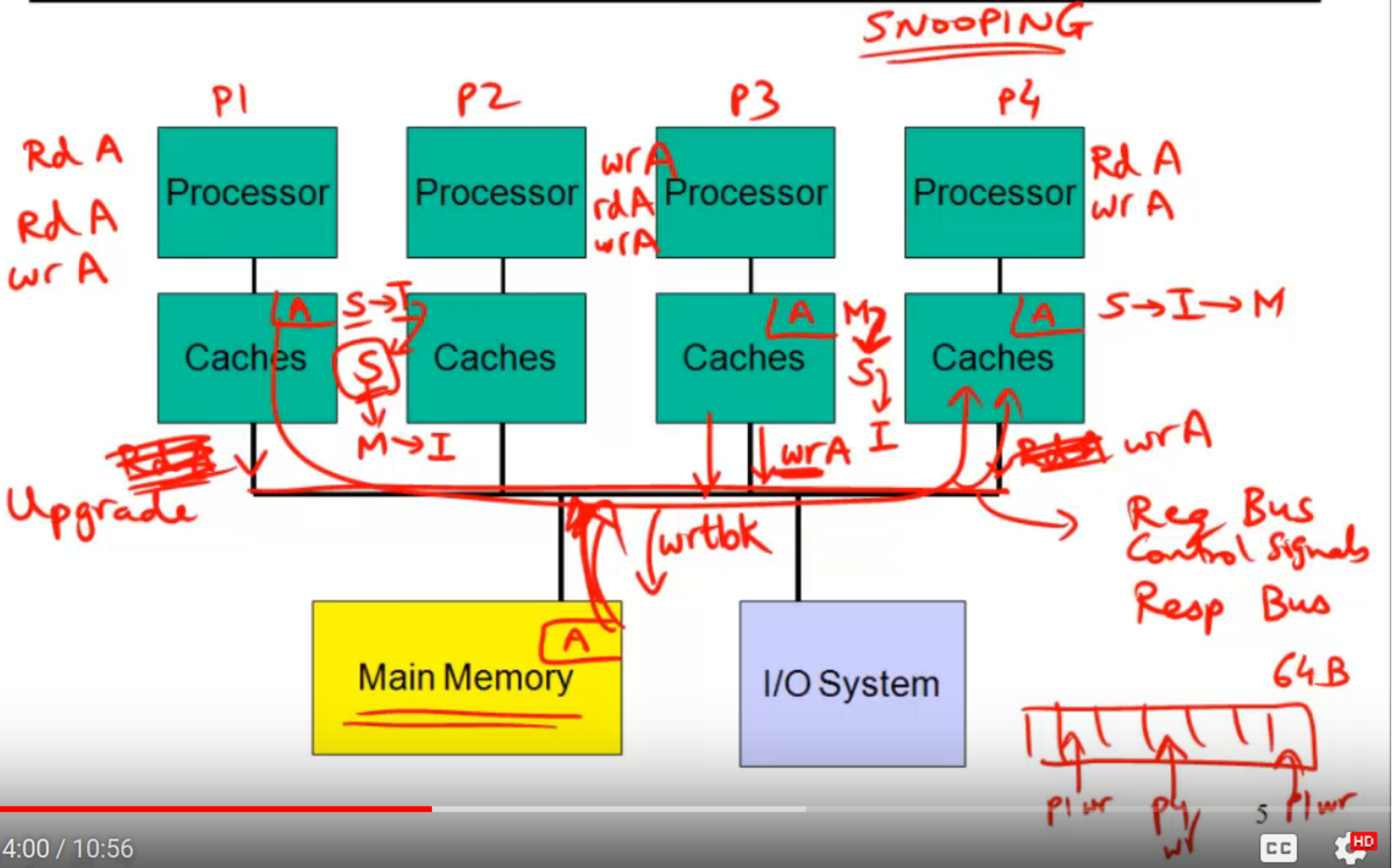
SMPs or Centralized Shared-Memory



2:03 / 10:56

5

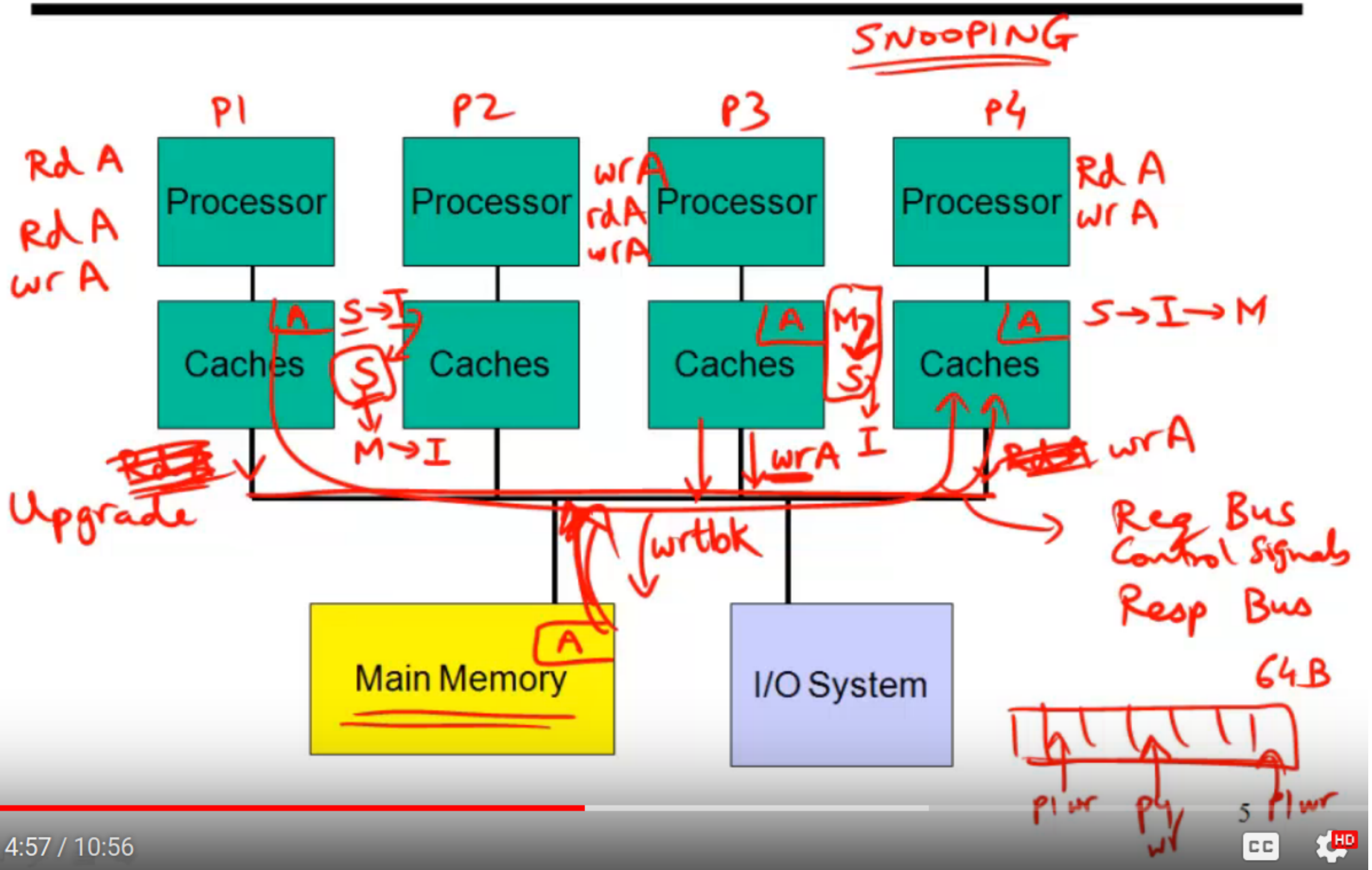
SMPs or Centralized Shared-Memory



5

4:00 / 10:56

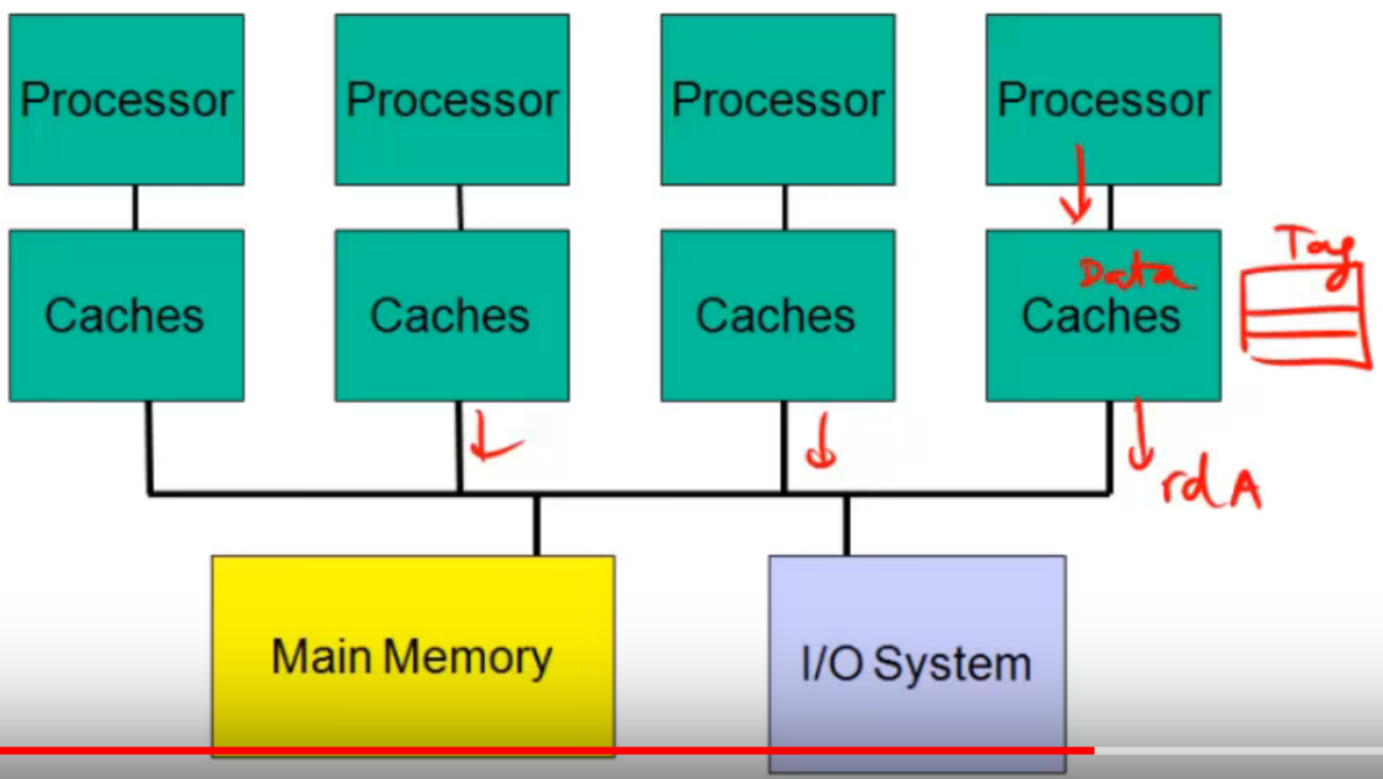
SMPs or Centralized Shared-Memory



Design Issues

Write prop^n
Write ser^n

- Invalidate
- Find data
- Writeback / writethrough
- Cache block states — MSI
- Contention for tags
- Enforcing write serialization



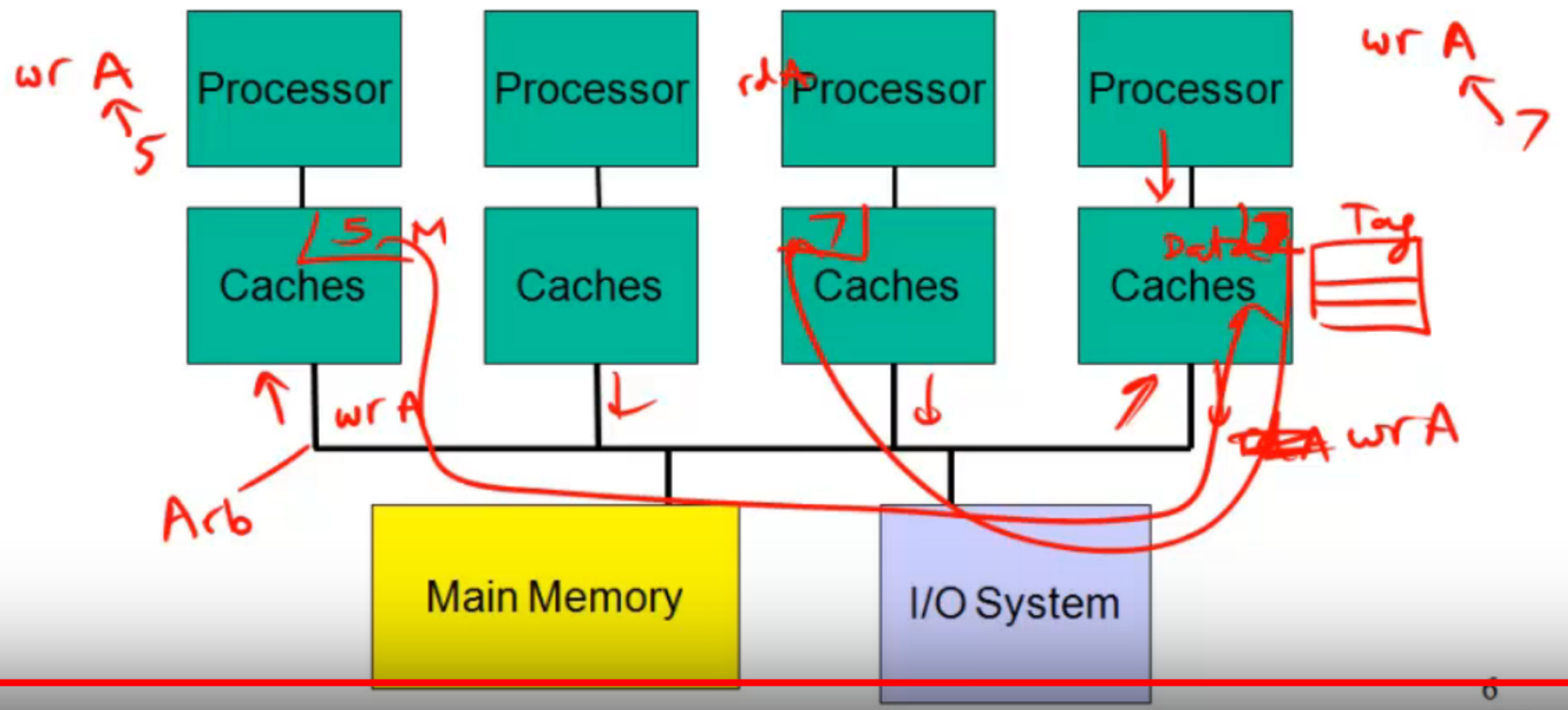
6

7:22 / 10:56

Design Issues

Write Prop^
Write Ser^

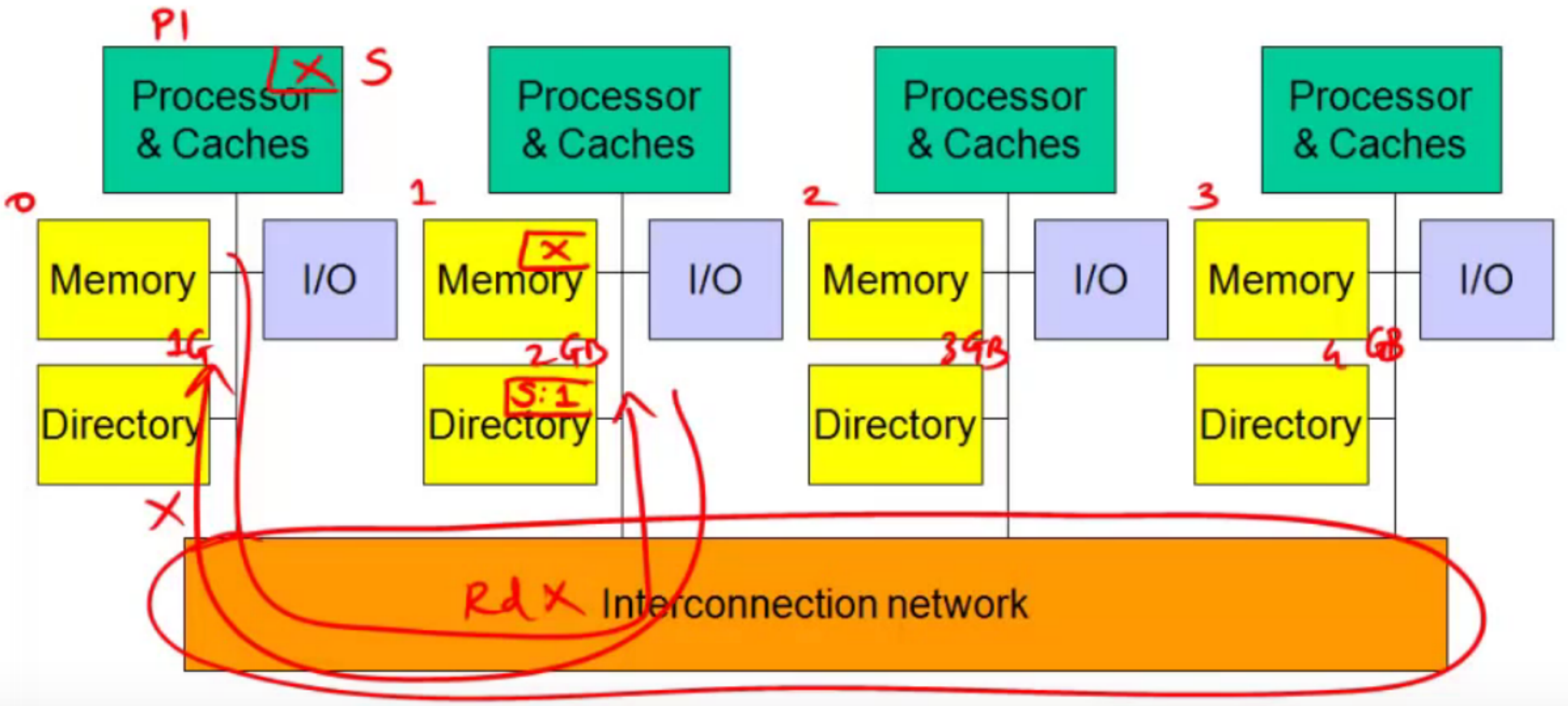
- Invalidate
- Find data
- Writeback / writethrough
- Cache block states — MSI
- Contention for tags
- Enforcing write serialization



9:22 / 10:56

Distributed Memory Multiprocessors

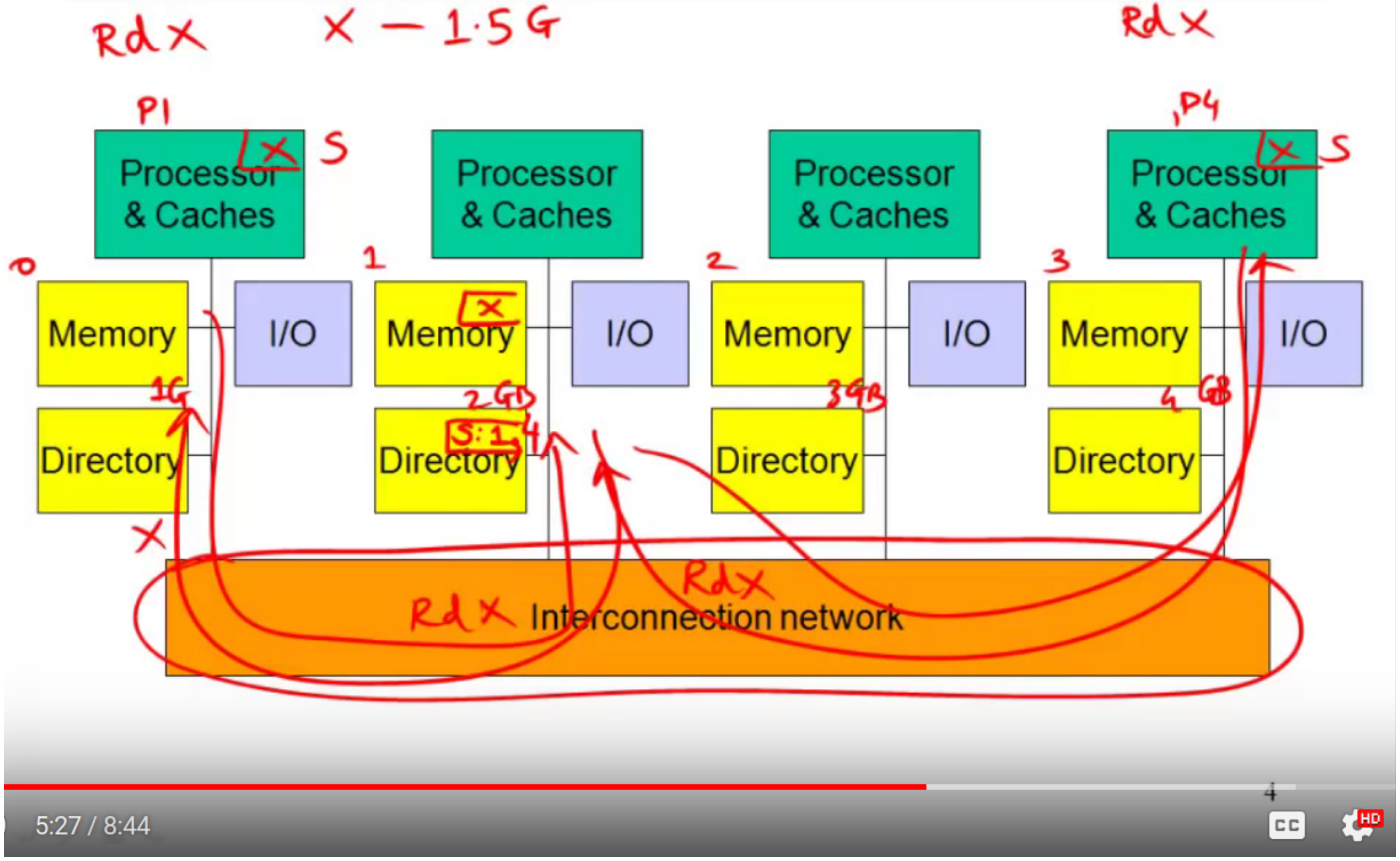
Rd X $\quad X - 1.5G$



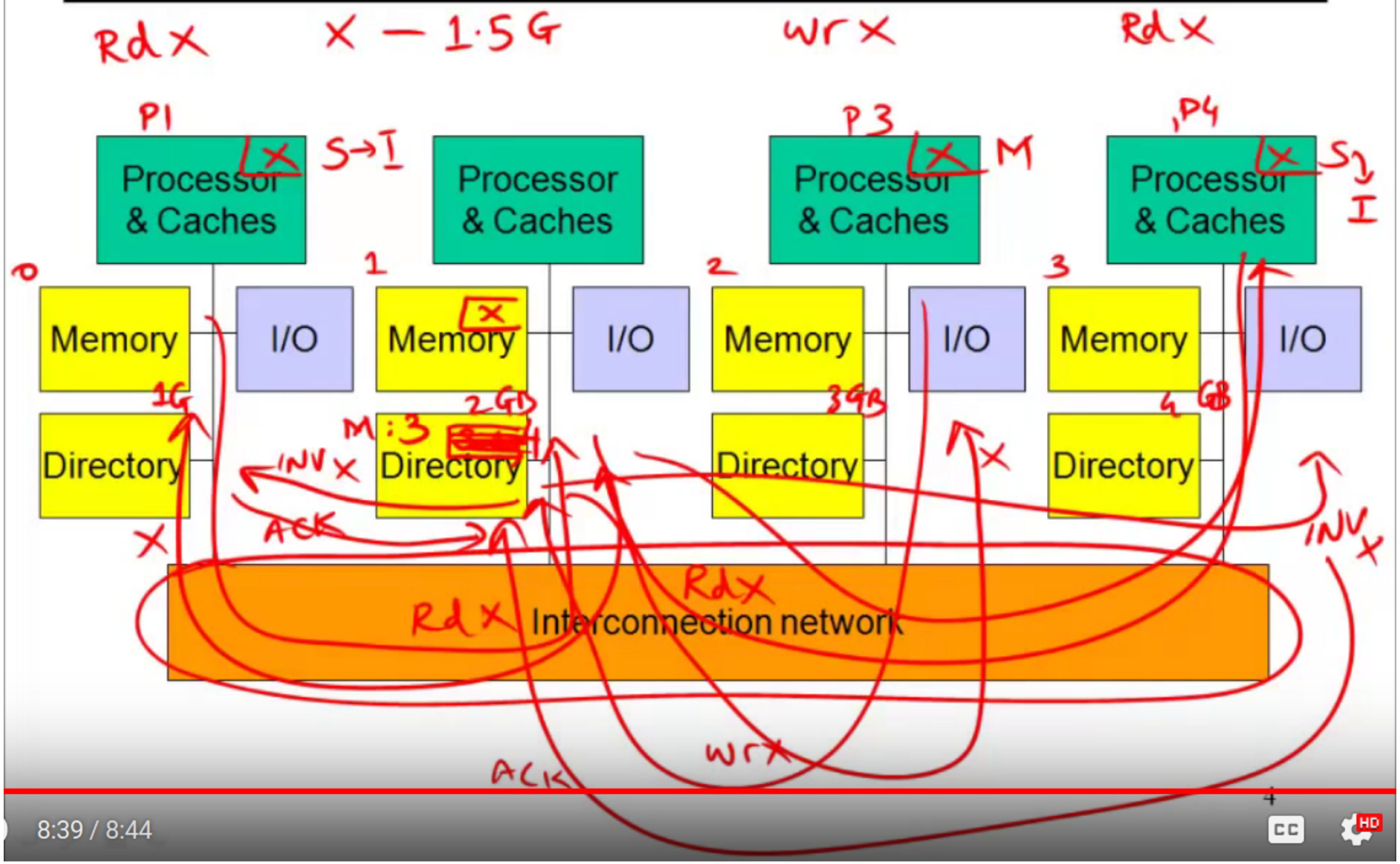
4:58 / 8:44

4

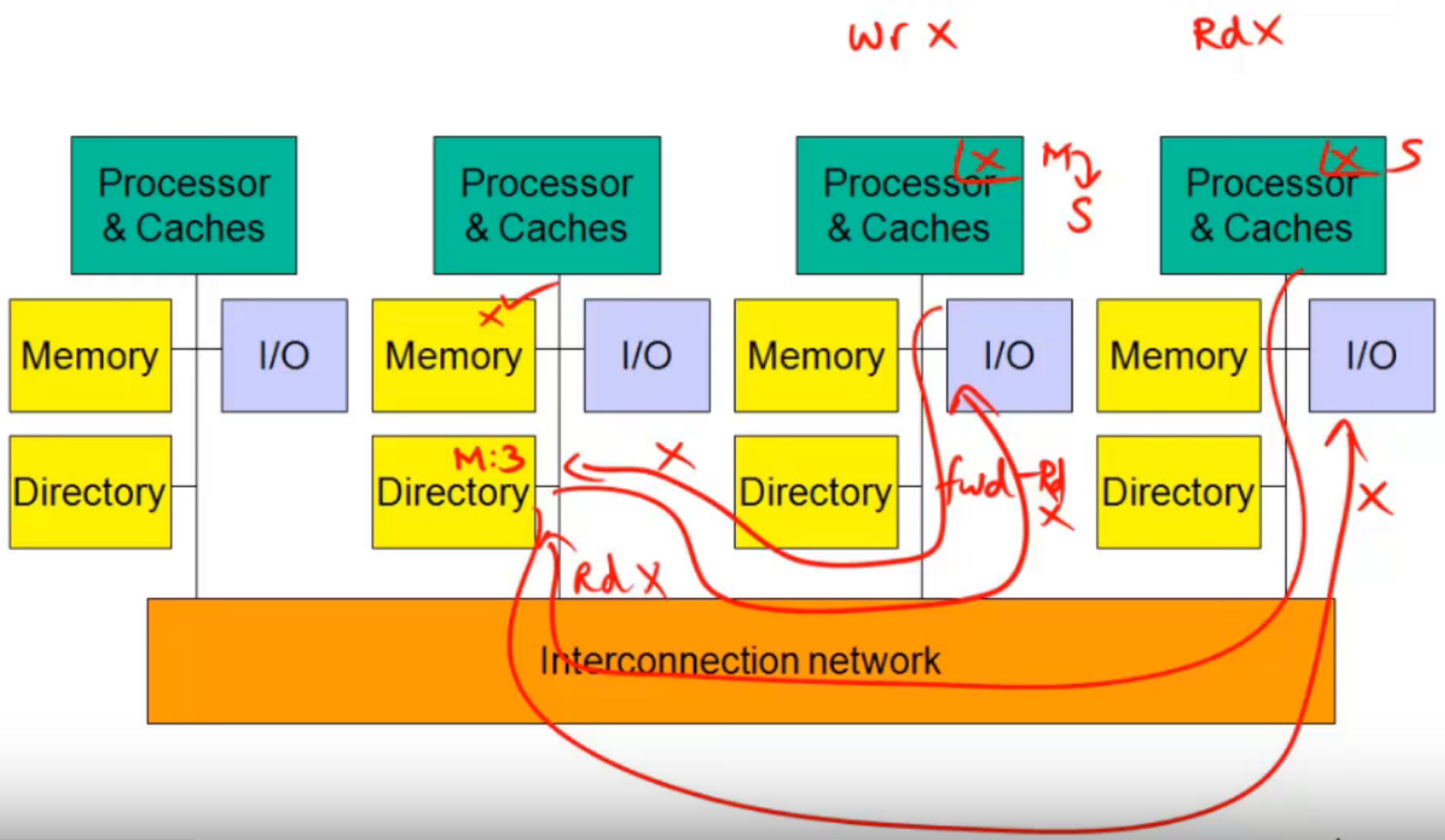
Distributed Memory Multiprocessors



Distributed Memory Multiprocessors

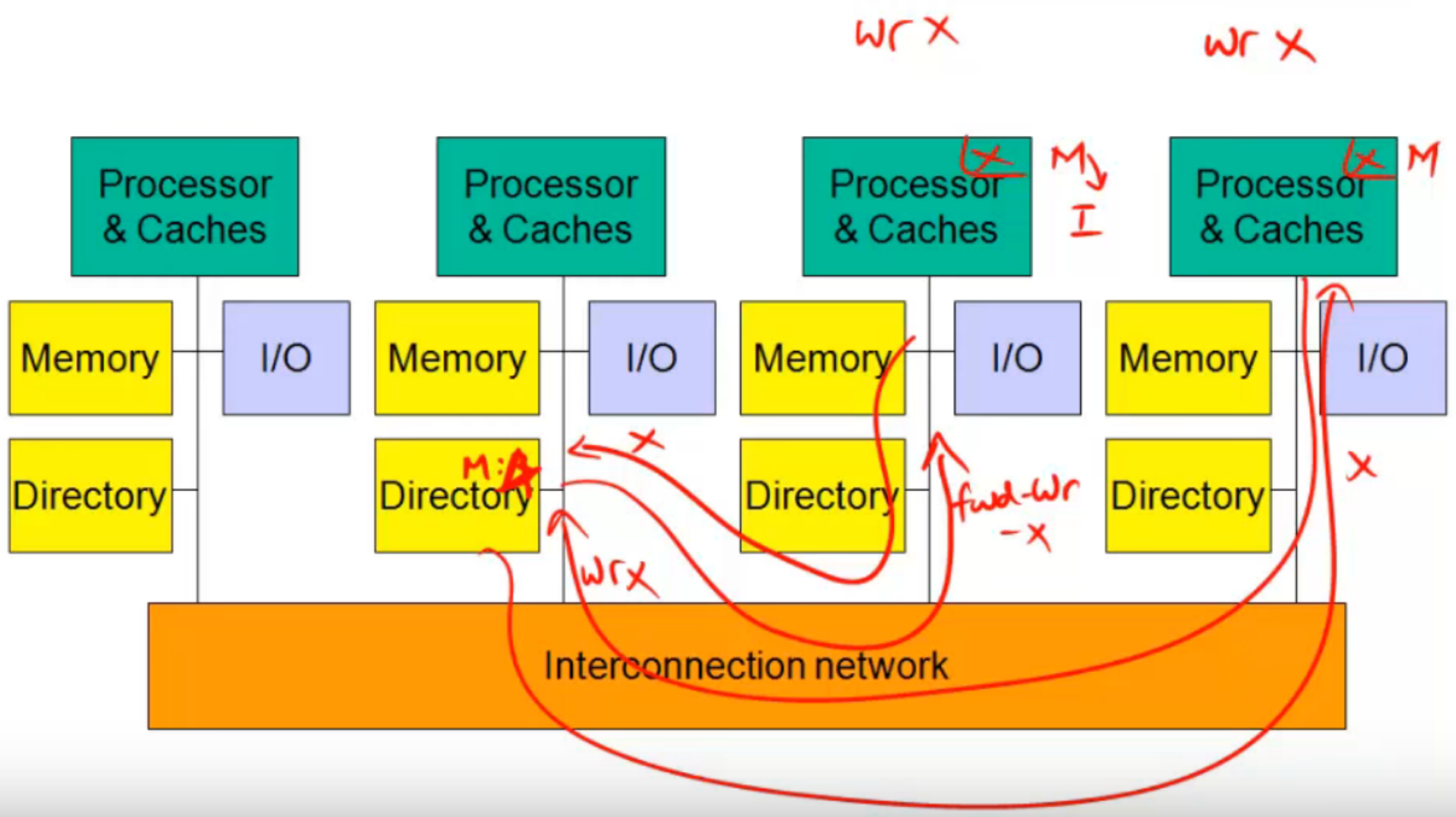


Distributed Memory Multiprocessors



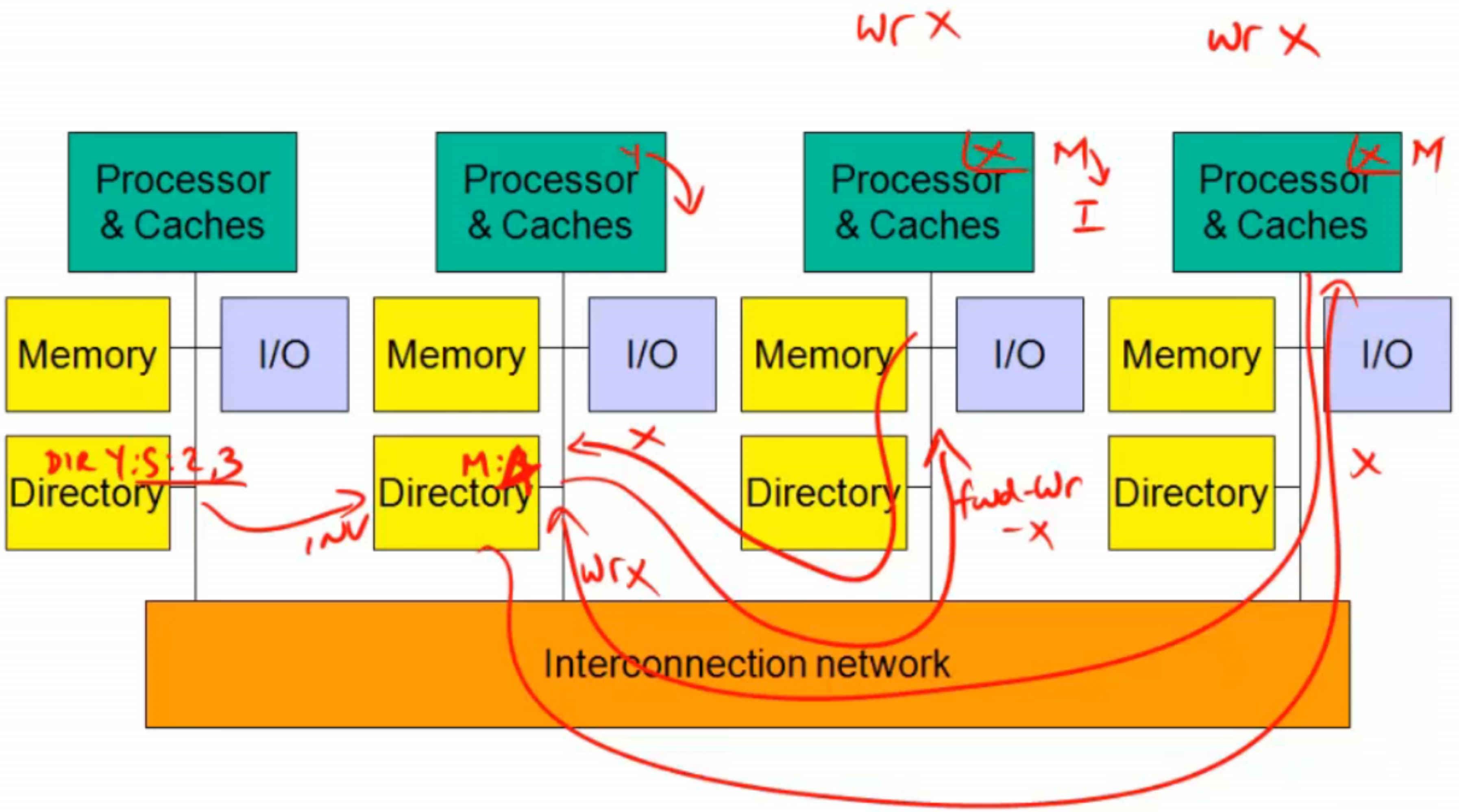
2:14 / 9:47

Distributed Memory Multiprocessors

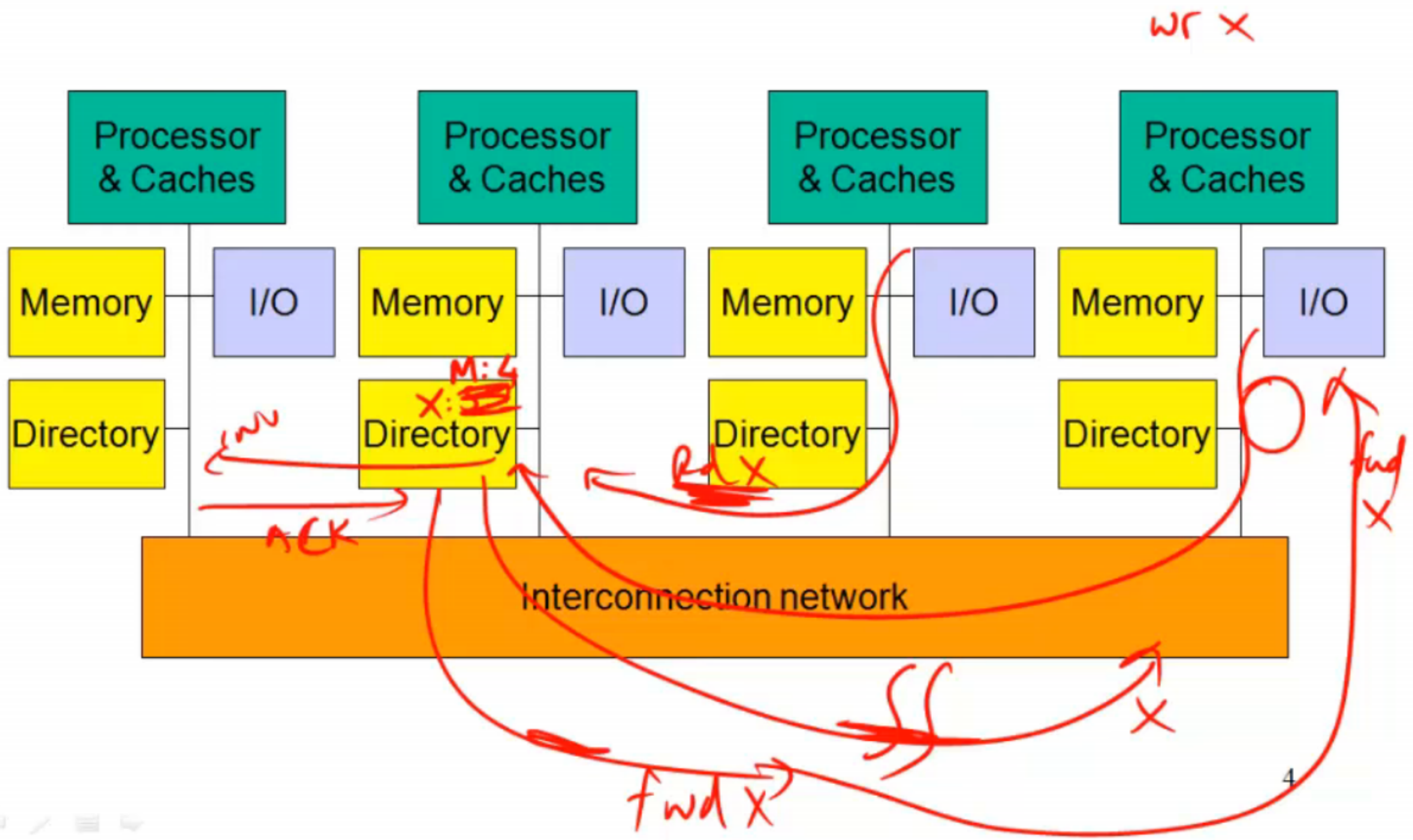


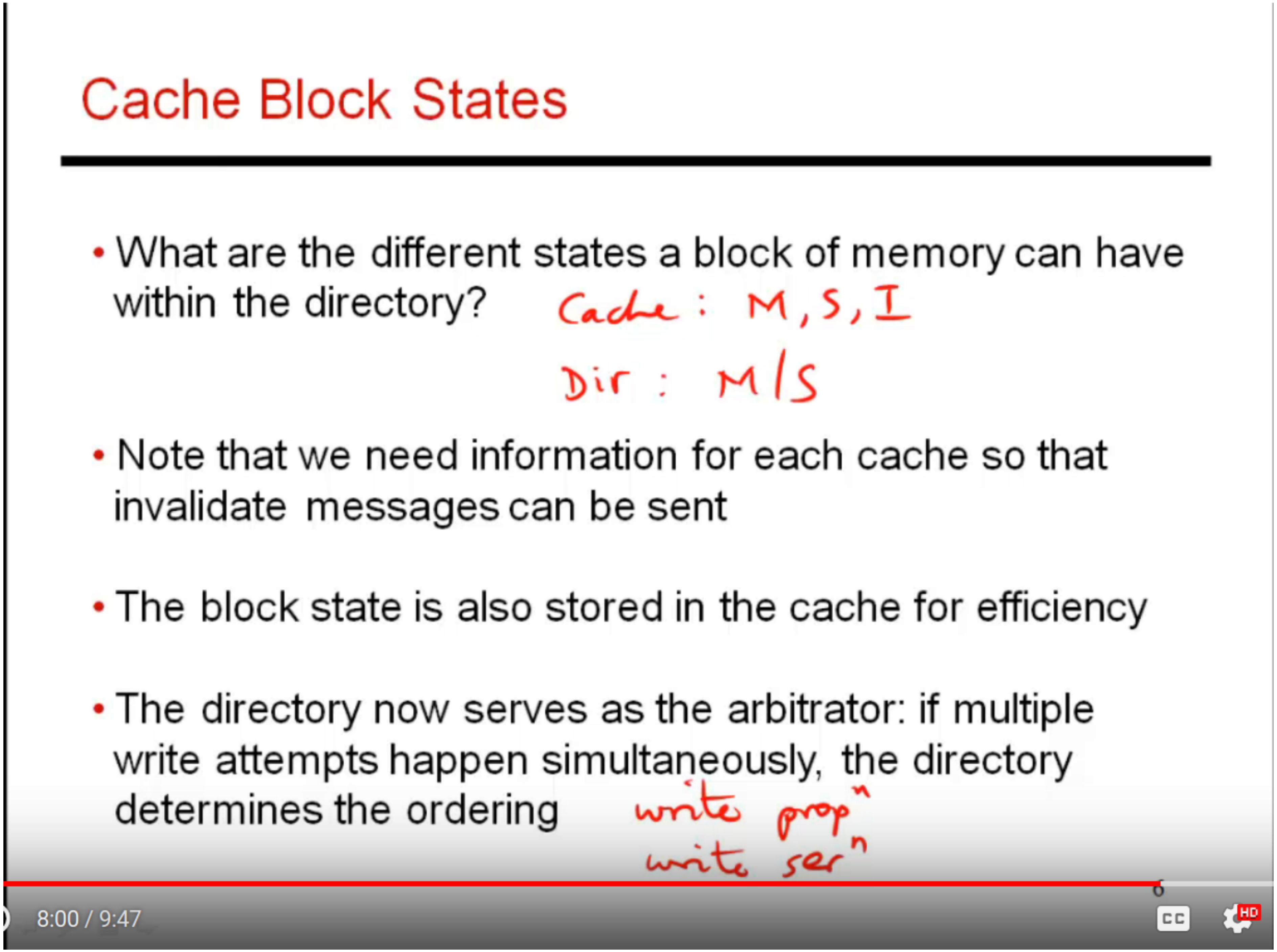
3:54 / 9:47

Distributed Memory Multiprocessors



Distributed Memory Multiprocessors





Cache Block States

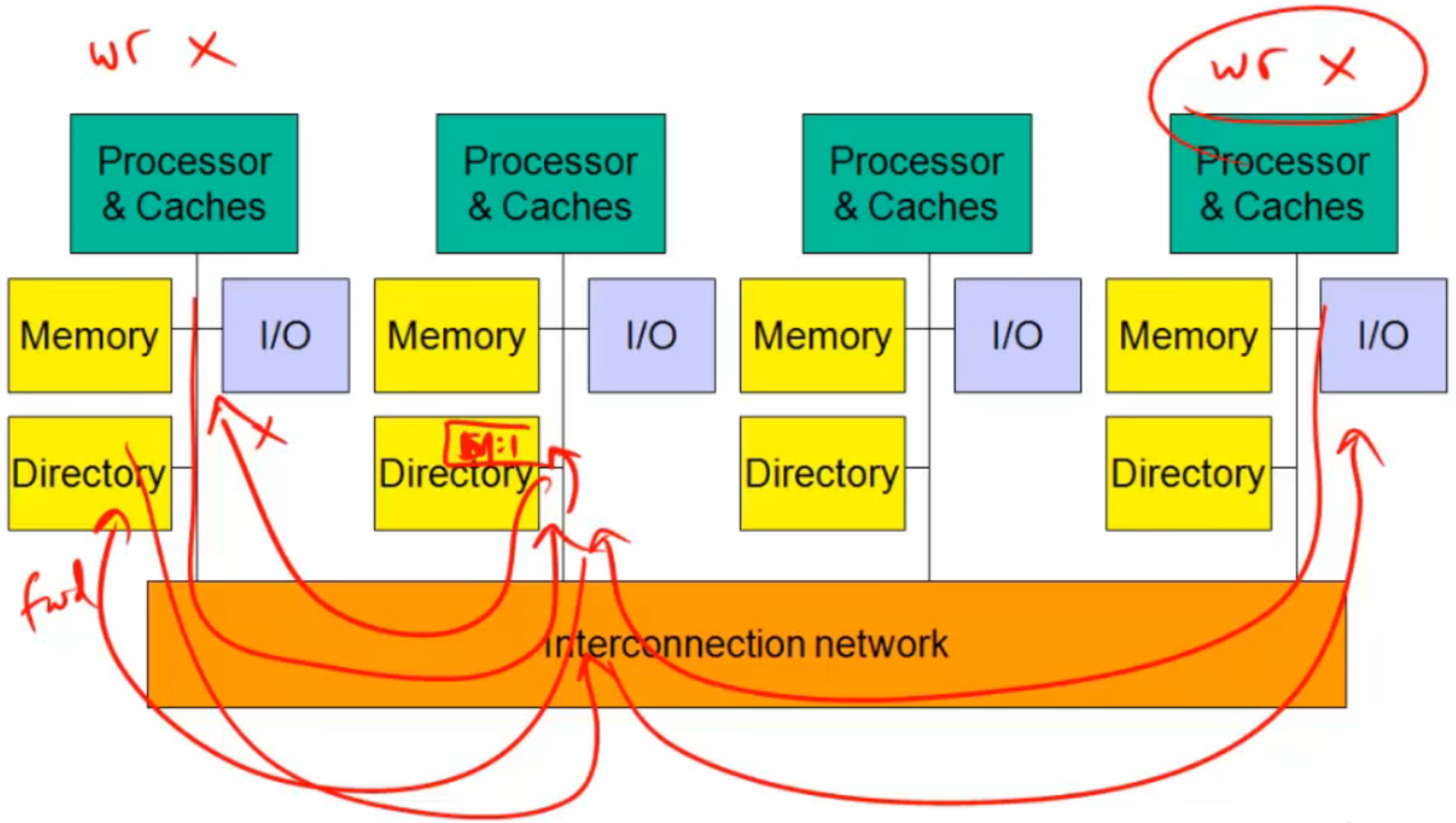
- What are the different states a block of memory can have within the directory? Cache : M, S, I

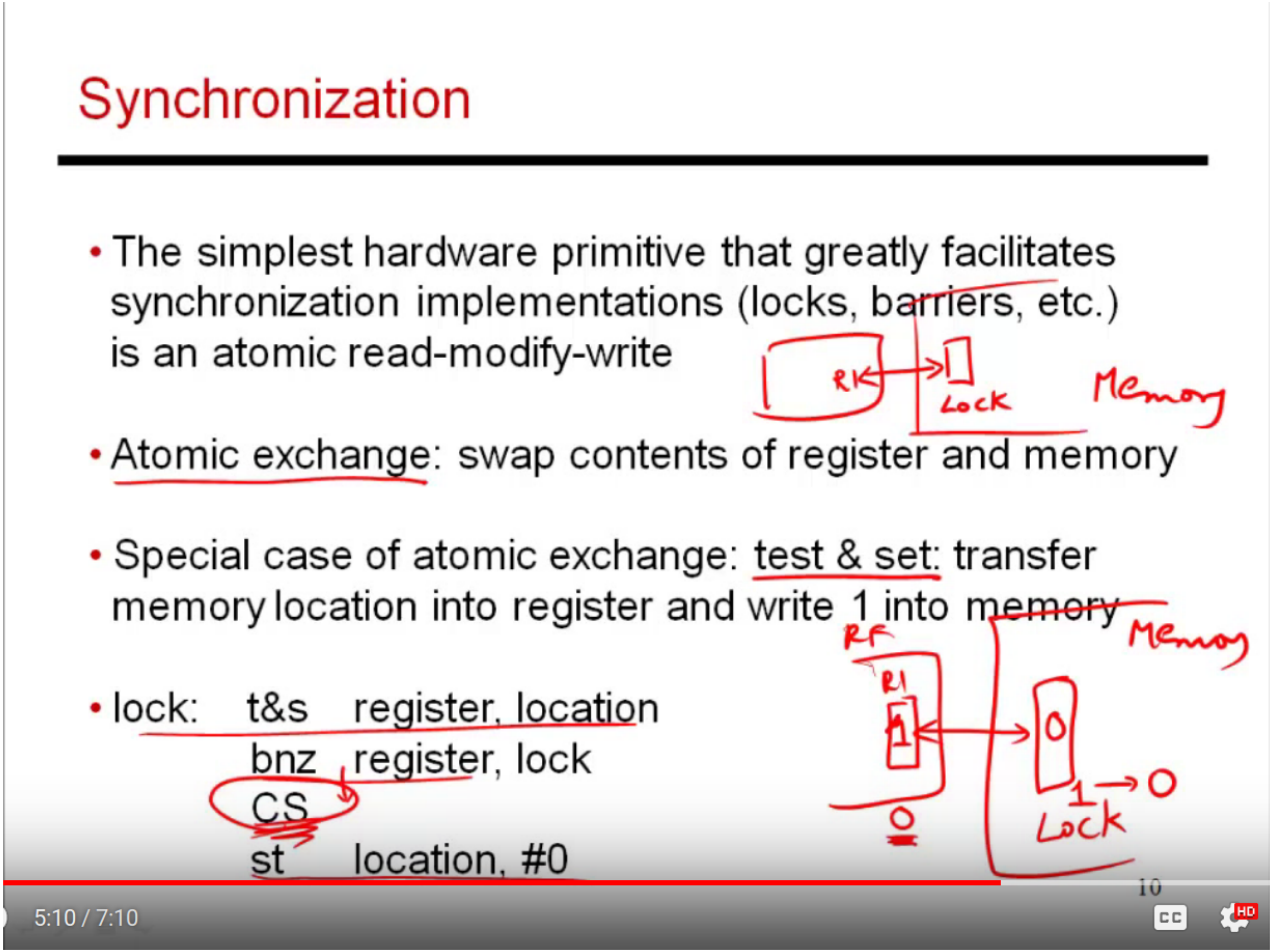
  Dir : M/S

- 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 write propⁿ

  write serⁿ

Distributed Memory Multiprocessors





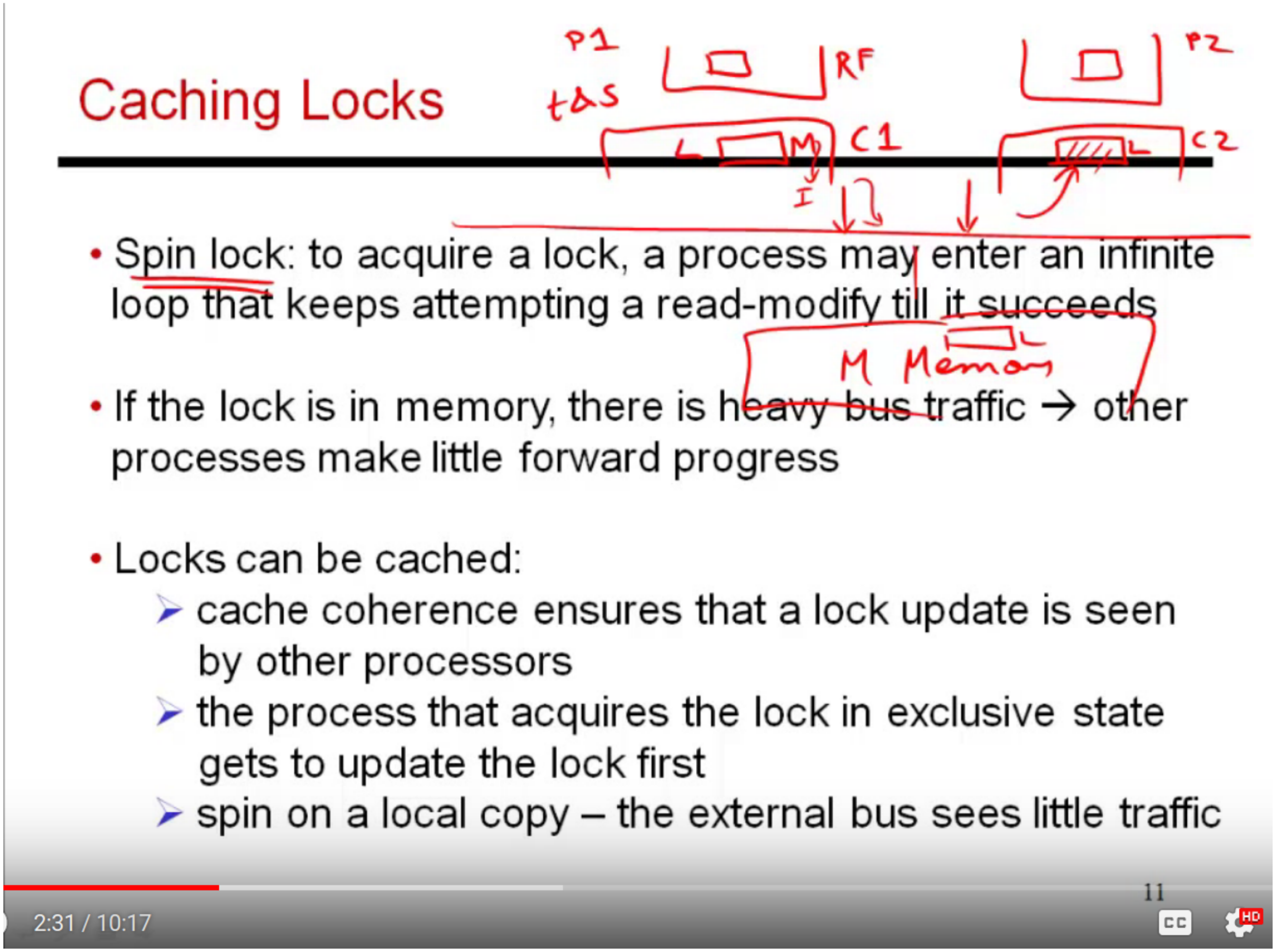
Synchronization

- 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:

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

10

5:10 / 7:10



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

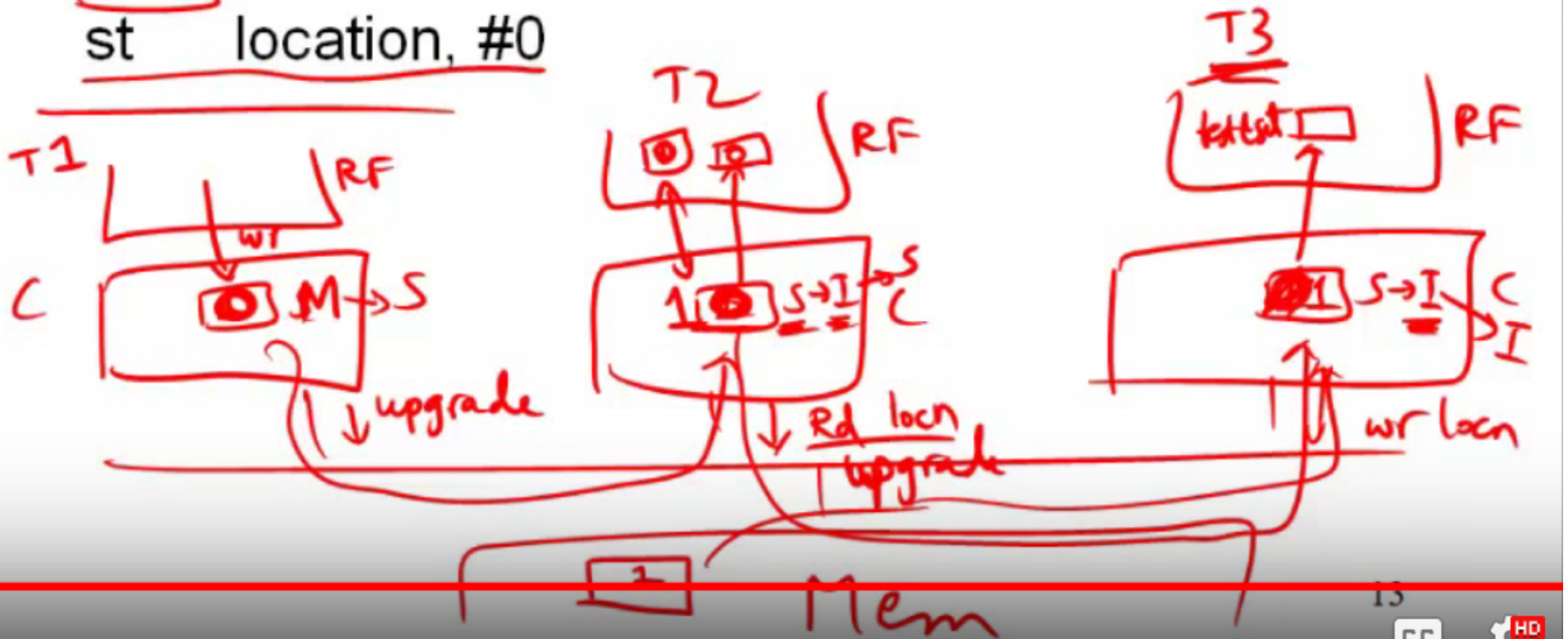
11

2:31 / 10:17

Test-and-Test-and-Set

T1

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



10:15 / 10:17

Spin Lock with Low Coherence Traffic

```
lockit: LL       R2, 0(R1)   ; load linked, generates no coherence traffic
        BNEZ     R2, lockit  ; not available, keep spinning
        DADDUI R2, R0, #1    ; put value 1 in R2
        SC       R2, 0(R1)   ; store-conditional succeeds if no one
                             ; updated the lock since the last LL
        BEQZ     R2, lockit  ; confirm that SC succeeded, else keep trying
```

- If there are i processes waiting for the lock, how many bus transactions happen?

  1 write by the releaser + i read-miss requests + $\overset{1}{\text{i}}$ responses + 1 write by acquirer + 0 (i-1 failed SCs) + i-1 read-miss requests + $\underbrace{\text{i-1}}_{1}$ responses