Lecture 6

Chip multiprocessors

- **Multithreading techniques (Ch. 5.2.1, 8.3)**
  - ✓ Interleaved (fine-grain) multithreading
  - ✓ Block (coarse-grain) multithreading
  - ✓ Simultaneous multithreading
- **Cache coherence solutions (Ch. 5.4.1-5.4.3)**

Multi-core and Thread-level Parallelism

Thread-Level Parallelism

Process: A program that can run independently of other programs on a single or multiprocessor system.

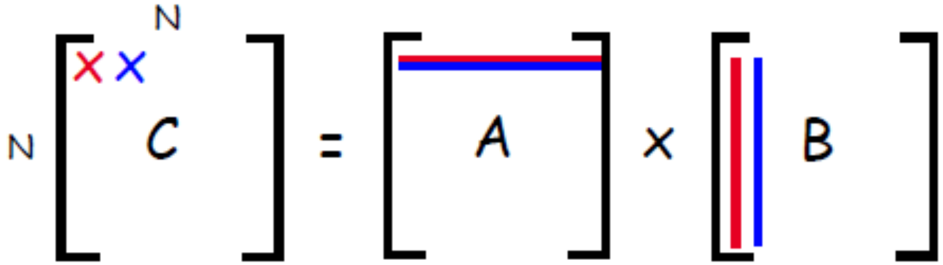
Thread: A piece of a program within a process that runs on a processor.

Example:

A program that does matrix multiplication can be partioned into threads that do matrix multiplication on a part of the matrix.

Example Sequential Algorithm

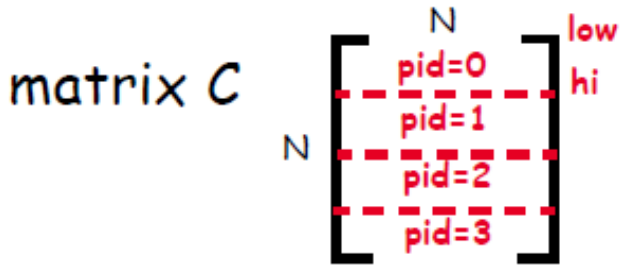
- Multiply matrices A[N,N] by B[N,N] and store result in C[N,N]
- Add all elements of C in sum




```
sum = 0;
for (i=0,i<N, i++)
        for (j=0,j<N, j++){
                C[i,j] = 0;
                for (k=0,k<N, k++)
                        C[i,j] = C[i,j] + A[i,k]*B[k,j];
                sum += C[i,j];
        }
```

Shared-memory Parallel Program

```
/* A, B, C, BAR, LV and sum are shared. All other variables are private.
        low = pid*N/nproc;            /* pid=0...nproc-1
        hi = low + N/nproc;           /* rows of A
        mysum = 0; sum = 0;           /* A and B are in
        for (i=low,i<hi, i++)         /* shared memory
            for (j=0,j<N, j++){
                C[i,j] = 0;
                for (k=0,k<N, k++)
                    C[i,j] = C[i,j] +  A[i,k]*B[k,j]; /* at the end matrix C is
                mysum +=C[i,j];                   /* C is in shared memory
            }
        BARRIER(BAR);
        LOCK(LV);
            sum += mysum;
        UNLOCK(LV);
```



Guarantees that all threads have arrived before anyone is allowed to continue.

Critical section guarantees that sum is updated atomically

Multithreading
(Ch. 5.2.1, 8.3)

Multithreading

Idea: Increase resource utilization by multiplexing the execution of multiple threads on the same pipeline

Two approaches:

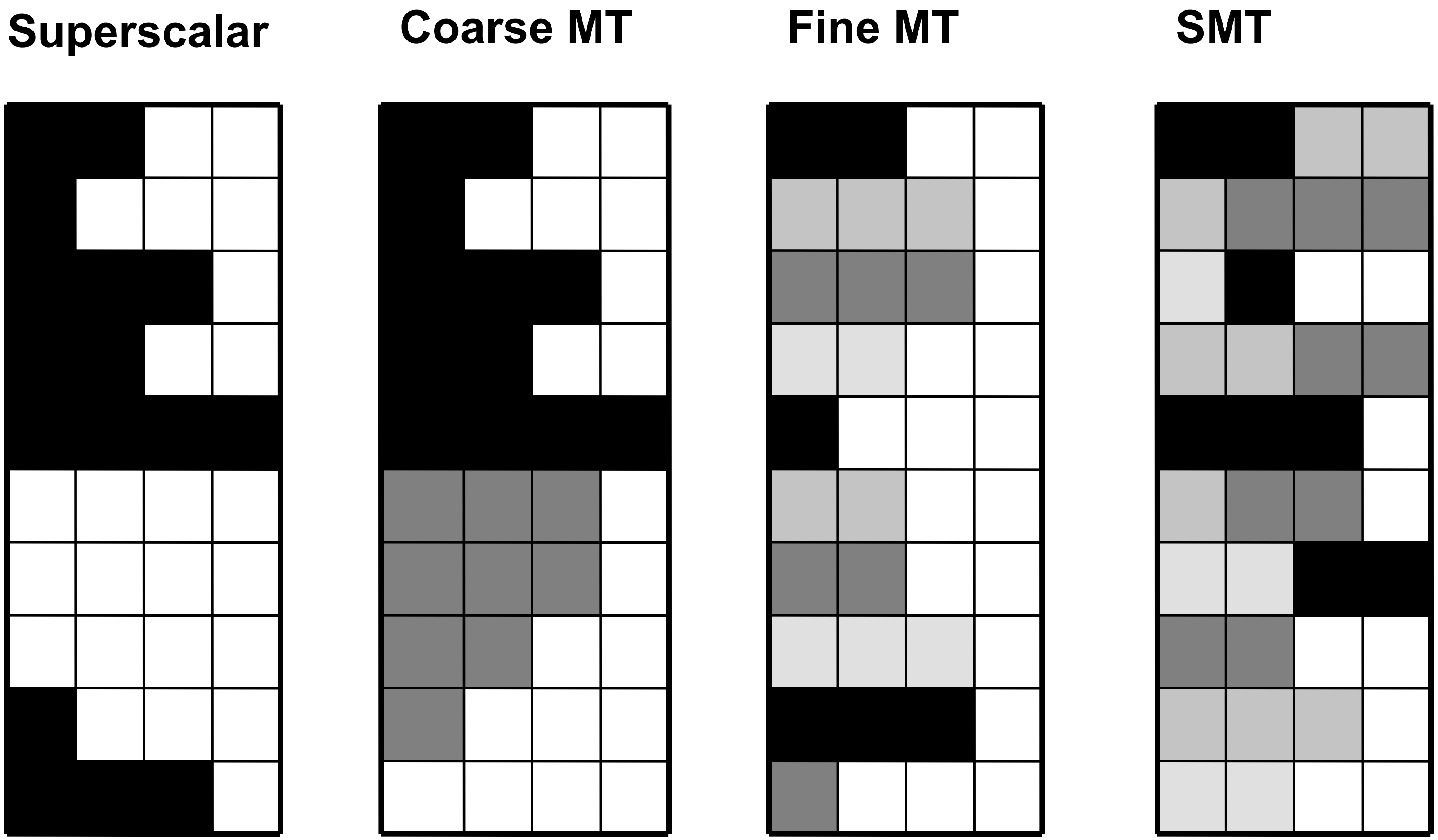
Fine-grain : Switch to another context each cycle

P1	P2	P3	....	PN	P1	P2

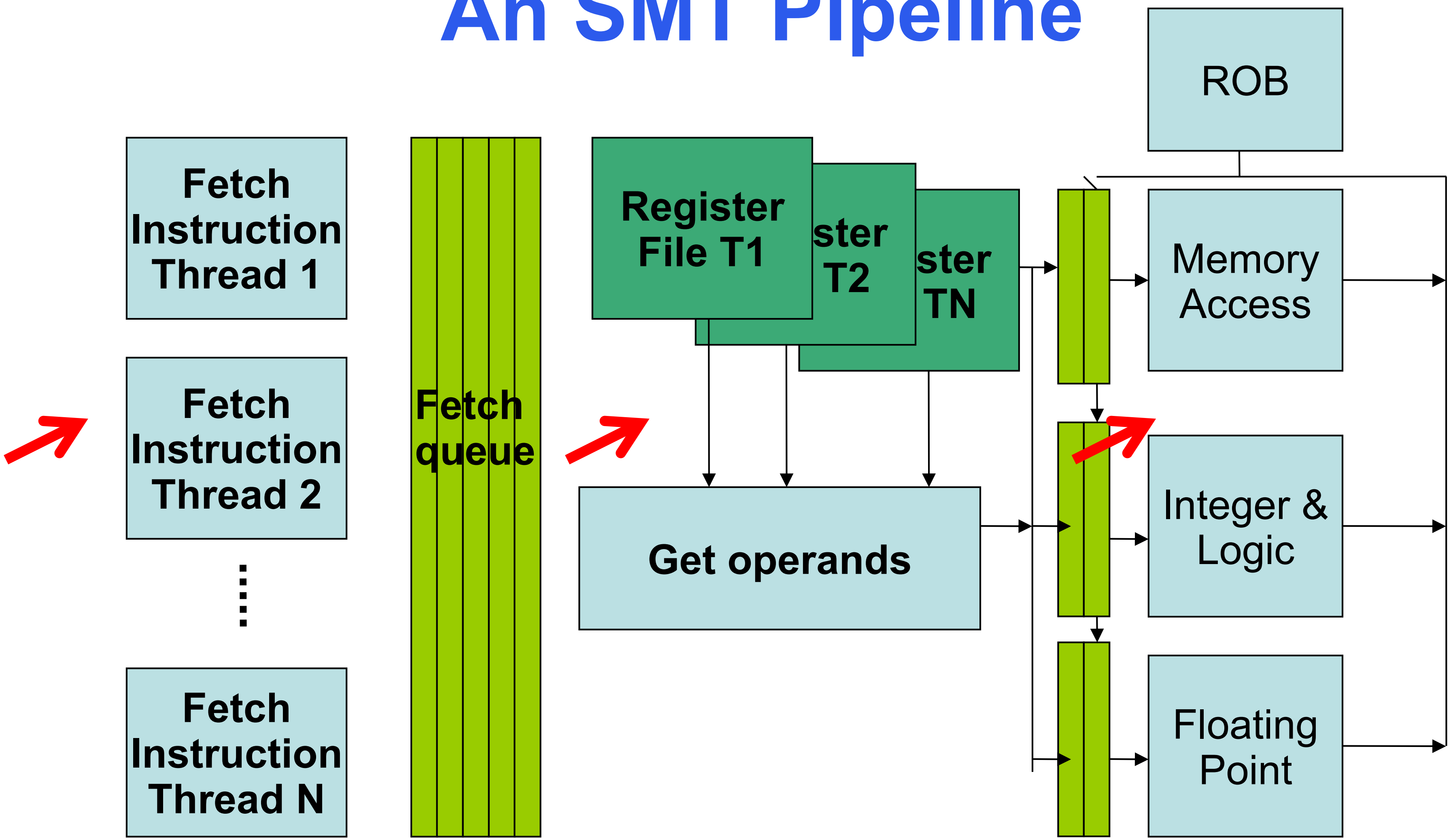
Coarse-grain: Switch to another context on costly stalls

P1	P1	P2	....	P2	P3	P3
	miss			miss		

Simultaneous Multithreading



An SMT Pipeline



Front-end and register file must be replicated

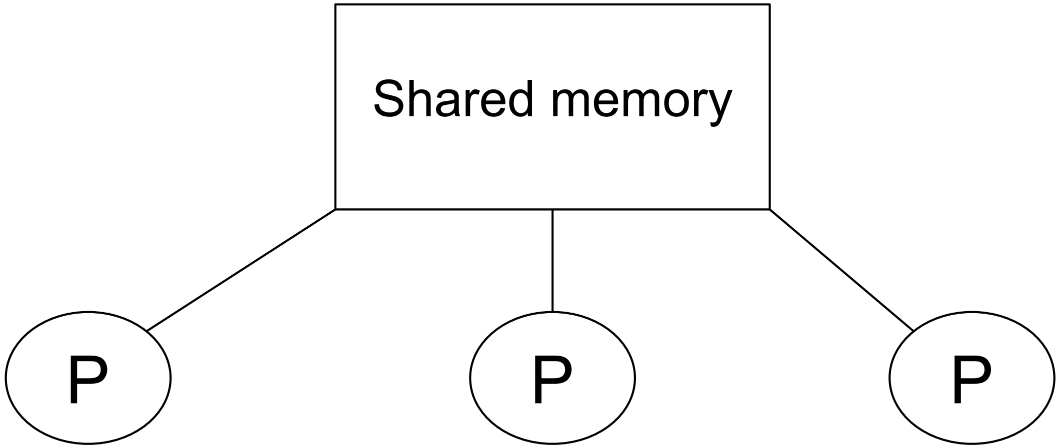
Quiz 6.1

Which of the following statements are correct

a) Coarse-grain multithreading switches between threads every cycle
b) Fine-grain multithreading switches between threads every cycle
c) Simultaneous multithreading allows free pipeline cycles to be used by other threads
d) Coarse-grain multithreading switches to another thread on long-latency operations

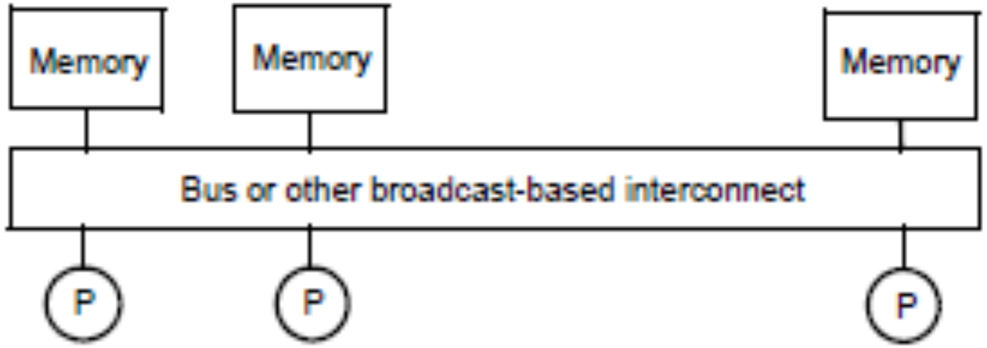
Organization of Multi-core/Multiprocesor Systems

Shared-memory Multiprocessors



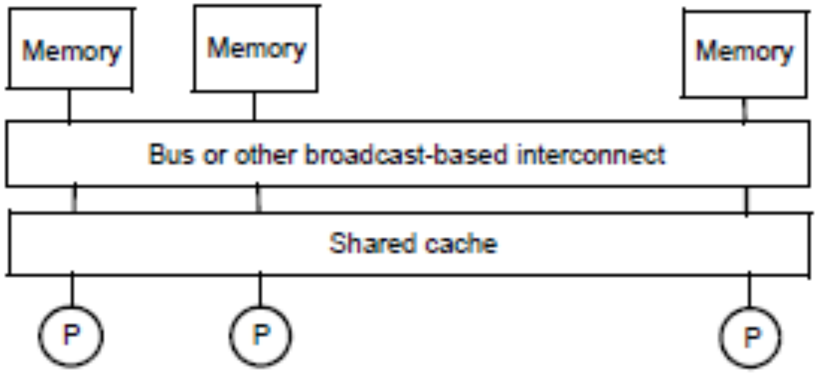
- All processors conceptually share memory
- **Challenge:** How to provide low latency and high bandwidth for high processor counts

Cache Organizations 1(4)



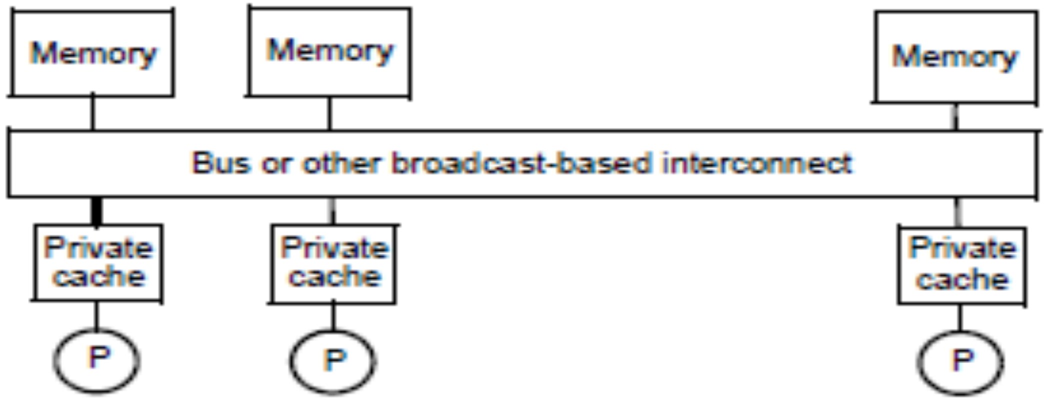
- Conceptual "dance-hall" model of shared memory multiprocessor
- **No cache – not practical**

Cache Organizations 2(4)



- Shared first-level cache organization
- **Advantage:** constructive sharing
- **Disadvantage:** interconnect between processors and cache on critical path

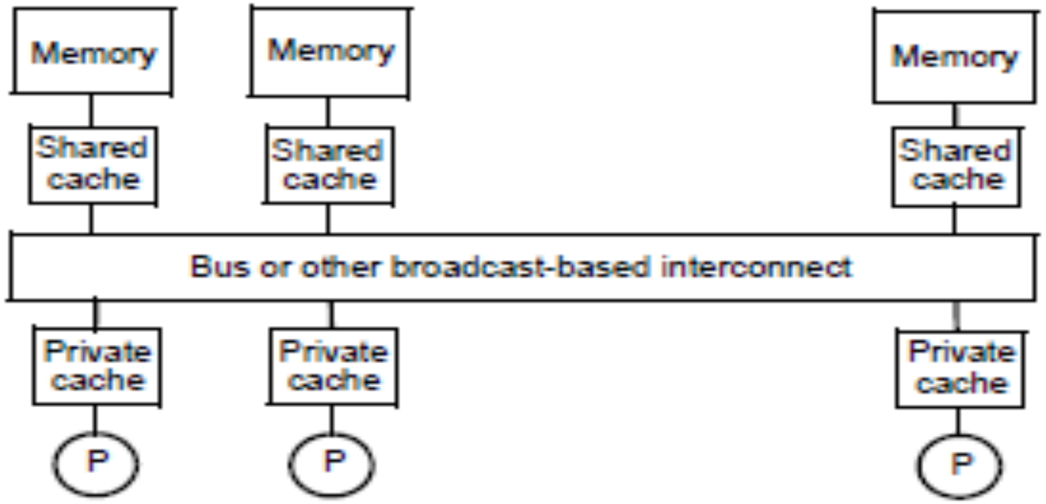
Cache Organizations 3(4)



- Private first-level cache organization
- **Advantage:** Fast access
- **Disadvantage:** Multiple copies of the same data can exist

Private caches give rise to the **cache coherence problem**

Cache Organizations



(d) SMP with private caches and shared Level 2 cache

- Hybrid first-level private and second-level shared cache organization
- Common in today's multicore chips

Quiz 6.2

Which of the following statements are correct

a) A shared cache allows multiple processors to share cache space
b) A shared cache is faster than a private cache
c) Copies of the same block pose a problem for private caches
d) Private caches are faster than shared caches

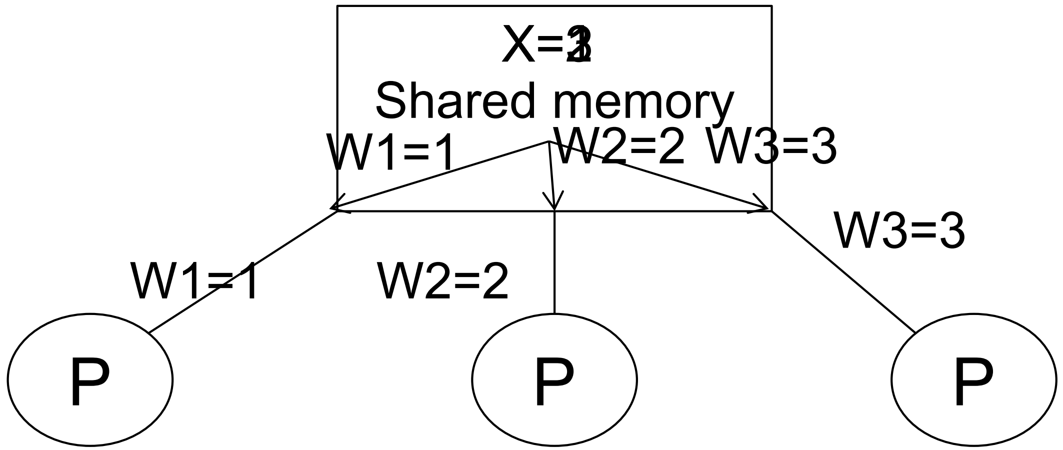
Cache Coherence Problem (Ch. 5.4.1-5.4.3)

The Cache Coherence Problem

Definition 1 [Performed]: A write is **performed** when the old value cannot be returned any more

Definition 2 [Last globally written value]: Assume that N processors in turn issue a write to a location: W_1, W_2, …, W_N. If W_1 is the last performed write, it leads to the **last globally written value.**

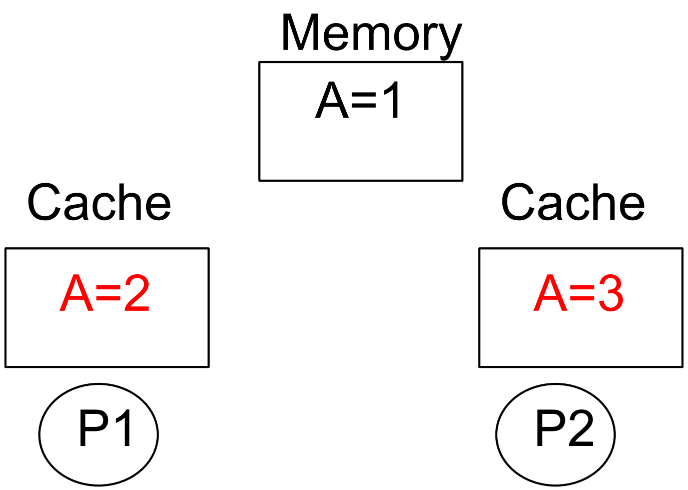
Example



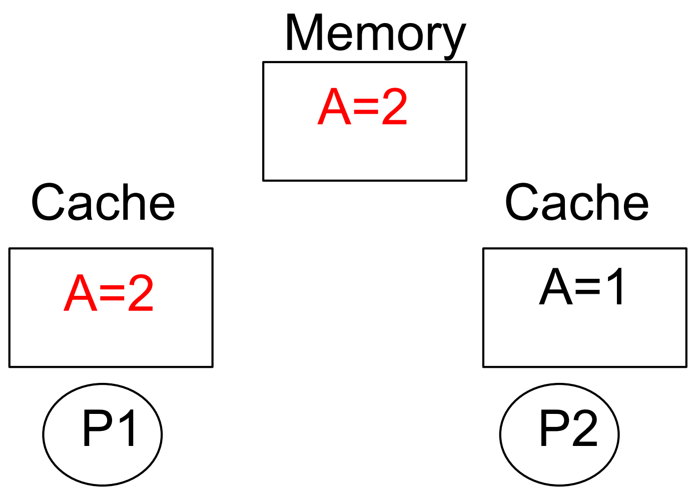
- Writes are performed when they change memory value
- W3 corresponds to the last globally written value
- **Definition 3 [Memory coherence]:** At any point in time, all processors have a consistent view of the last globally written value to each memory location.

Conventional Caches

Write-back caches



Write-through caches



- **Both cases:** No consistent view of last globally written value
- Need to devise a protocol that maintains cache coherence

Quiz 6.3

Consider the following access sequence, where $W_i(X)$ and R_i refer to a write and a read operation by processor P_i, respectively to the same variable and X refers to the value written.

$W_1(1)$ $W_2(2)$ R_3 $W_3(3)$ R_4

Which statements are correct?

a) Processor 3 reads value 1

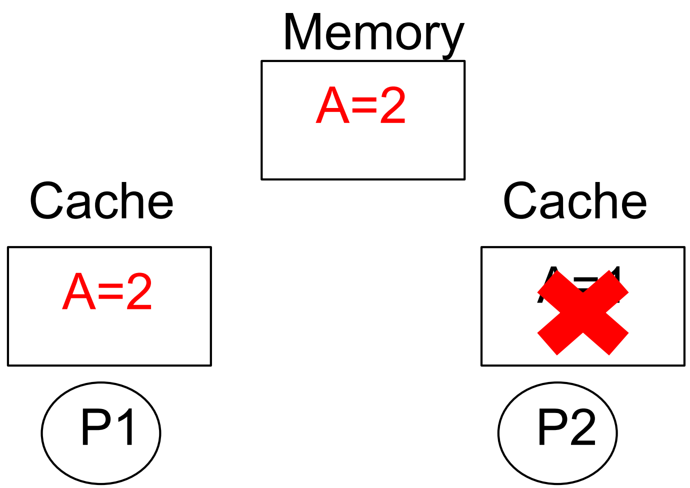
b) Processor 3 reads value 2

c) Processor 4 reads value 3

d) Last globally written value is 3

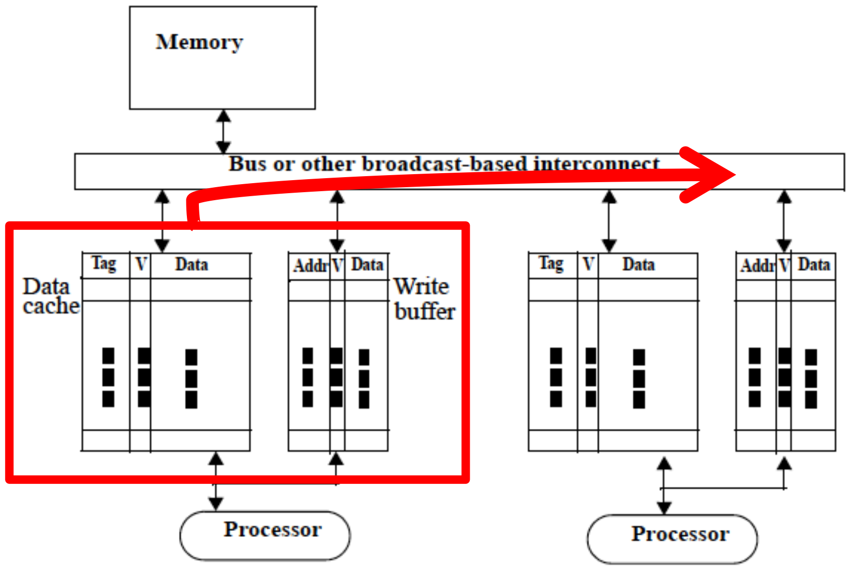
Cache Coherence Solutions

A Simple Snoopy Cache Protocol



- Same actions as a write-through cache for reads
- For writes, send **invalidation request** on the bus to all caches
- Snoopy cache protocol

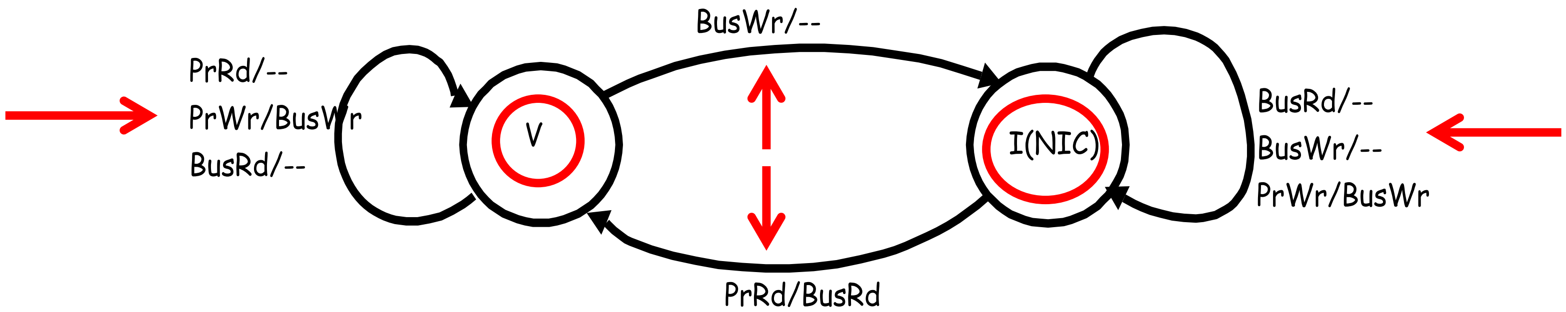
Hardware Structures



- Add new bus transaction: Invalidate
- Invalidate is broadcast on writes and “snooped” by all caches
- All copies of block are invalidated

State-Transition Diagrams

- Each cache is represented by a finite state machine (FSM)
- Imagine P identical FSMs working together, one per cache
- FSM represents the behavior of a cache w.r.t. a memory block
  - Not a cache controller!



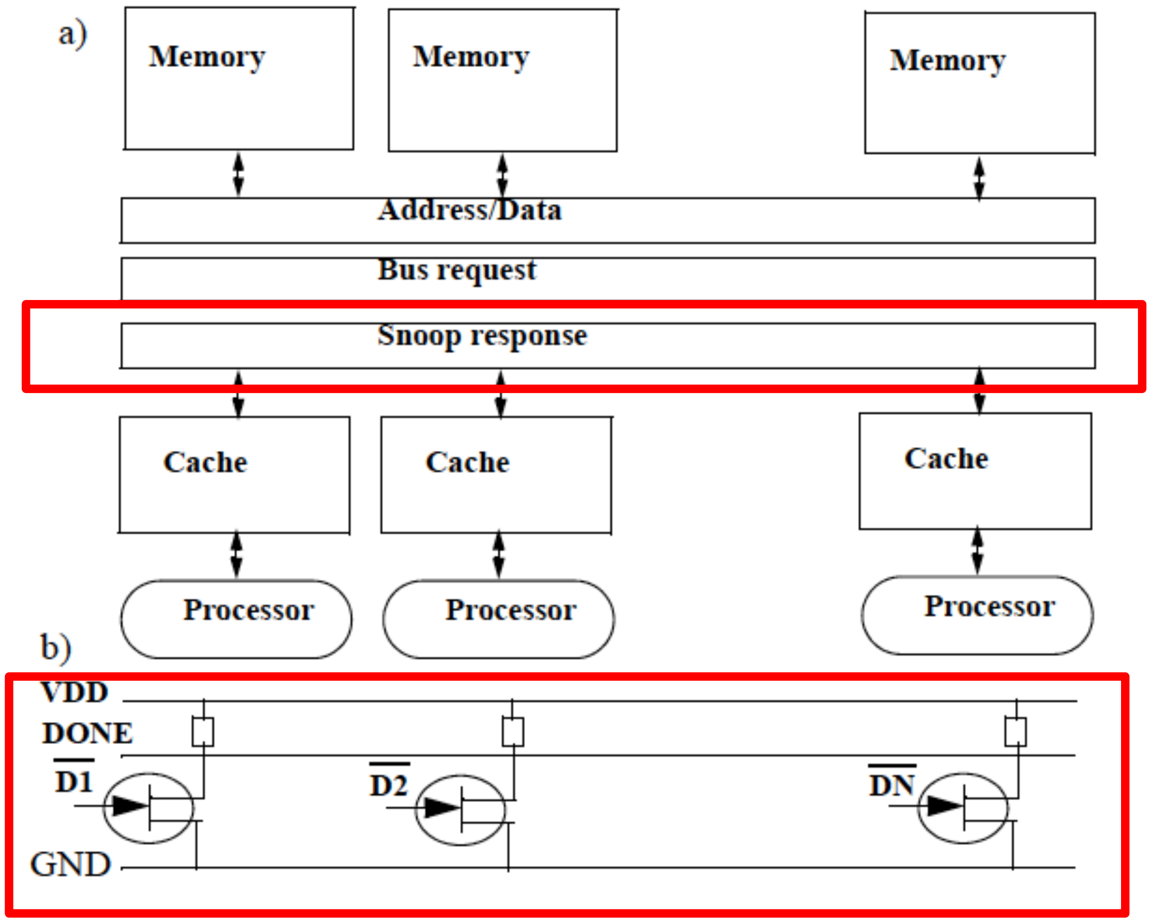
Problem with Simple Protocol

- Bus transaction on every write
- Consumes precious bus bandwidth
- Especially troublesome for sequential programs that do not exchange data

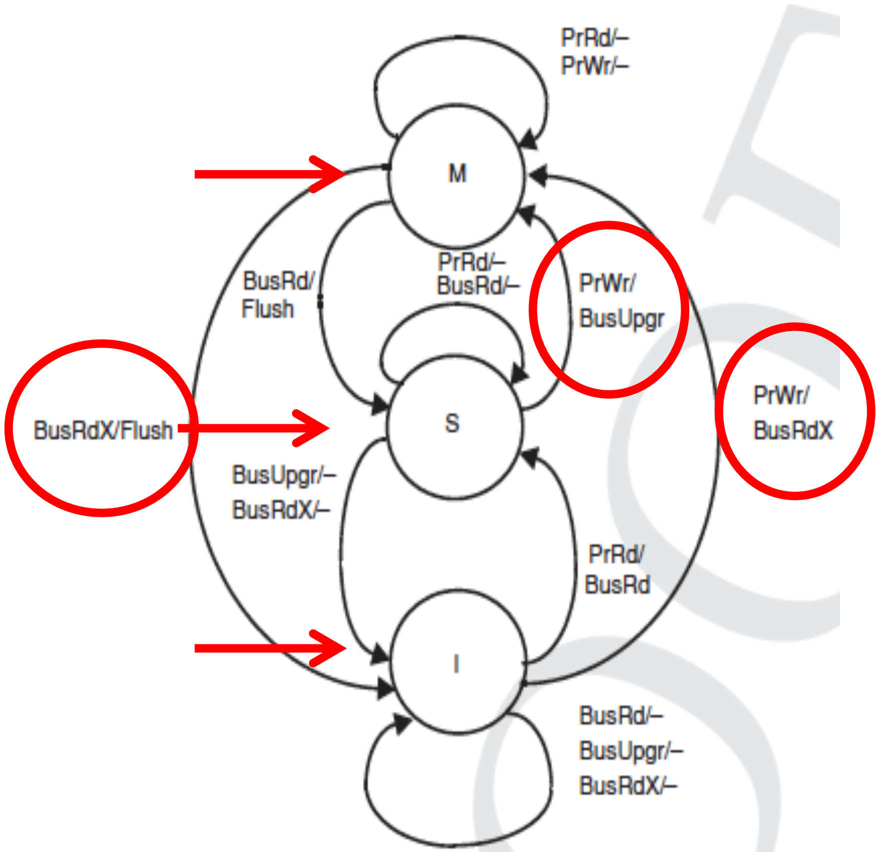
Write-invalidate protocol:

- Build cache coherence protocol around write-back caches
- Send invalidation only if there are other copies

Hardware Structures



Write-back Caches: the MSI Protocol



Quiz 6.4

Consider the following access sequence to the same variable:

$W_1(1)\ W_2(2)\ R_3\ W_3(3)\ R_4$

Which statements are correct?

a) One invalidation are sent out under the Simple protocol

b) One invalidation are sent out under the MSI protocol

c) Three invalidations are sent out under the Simple protocol

What you should know by now

- Multi-core systems
  - Thread-level parallelism
- Multithreading
  - Coarse and fine-grain multithreading
  - Simultaneous multithreading
- Cache coherence and schemes