Computer Architecture
计 算 机 体 系 结 构

Lecture 8. Multiprocessor and TLP
第八讲、多处理器和线程级并行

Chao Li, PhD.
李超 博士

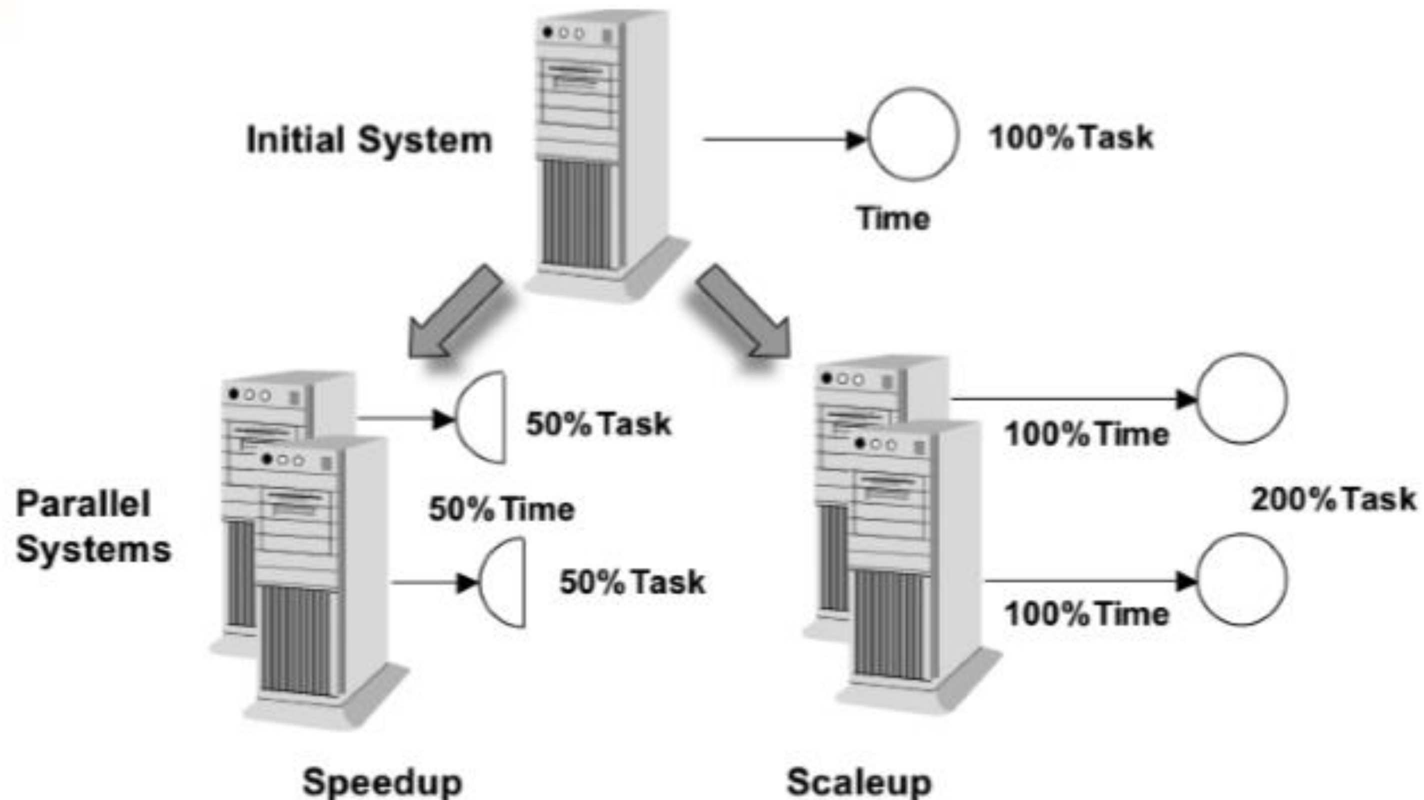
SJTU-SE346, Spring 2019

Review

- Amdahl's Law， Little's Law
- CPI, memory latency
- Processor/Server power estimation
- Trace-/Execution- driven simulation
- Simulation acceleration, sampling and checkpointing
- Workload characterization

Towards More Powerful Computer Systems

- A linear speedup if n times more resources make it possible to treat a given task in n times less time than the reference system
- A linear scaleup if n times more resources make it possible to deal with an n times larger problem in the same time as the reference system



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Flynn's Classification of Parallel Architectures

- **SISD**: Single Instruction Single Data
  - One stream of instructions on a single stream of data
  - e.g. Uniprocessors
- **SIMD**: Single Instruction Multiple Data
  - The same stream of instruction is applied to disjoint sets of data
  - vector supercomputers
- **MISD**: Multiple Instruction Single Data
  - No commercial multiprocessor of this type has been built to date
- **MIMD**: Multiple Instruction Multiple Data
  - Each node executes its own instruction stream on its own data
  - The architecture of choice for general-purpose multiprocessors

The Increased Importance of Multiprocessing

- More ILP can be inefficient
  - Especially for server applications
- Cloud-oriented processing
  - Massive data and internet requests
- Less motivation for scaling up
  - Increasing desktop performance is less important
- Better return on investment:
  - replication rather than unique design

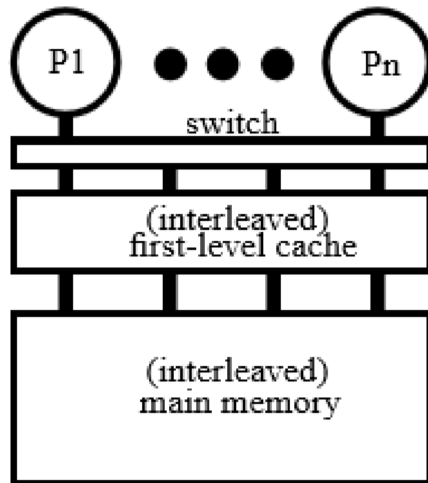
Topics to be Discussed in the Following Weeks

- **Thread-Level Parallelism**
  - Multiprocessor system consists of multiple chips
  - Single-chip systems with multiple cores (multicore)
- **Data-Level Parallelism**
  - Many-core accelerators
  - Interconnection network
- **Request-Level Parallelism**
  - Multicomputer system that is a cluster of servers
  - Data center and warehouse-scale computers

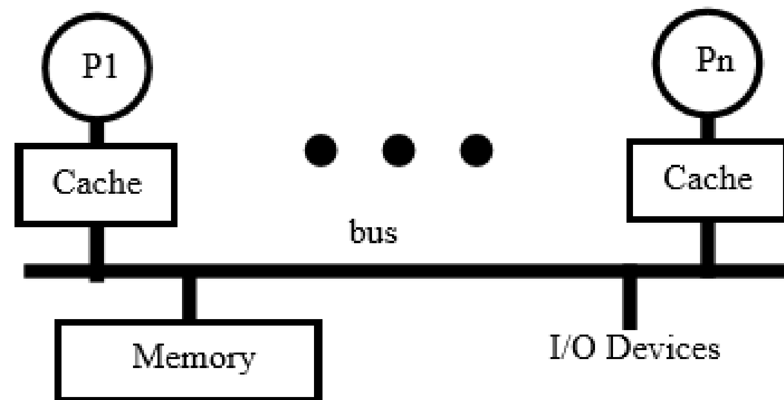
Outlines Today

- Multiprocessor Architecture
- Cache Coherence Problem
- Snooping Protocol

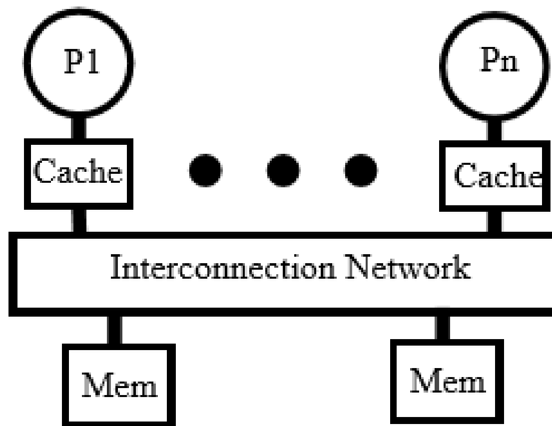
Common Memory Hierarchies in Multiprocessors



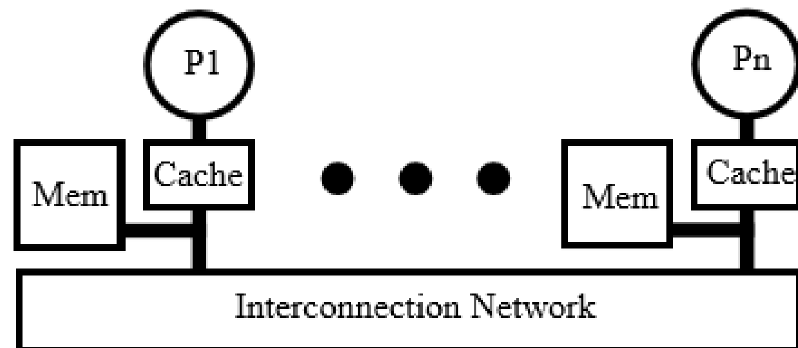
(a) Shared Cache



(b) Bus-based Shared Memory



(c) Dance-hall



(d) Distributed Memory

Two Classes of Multiprocessors

Centralized Shared-Memory

- Multiple processors share the same physical memory
- Uniform memory access (UMA)

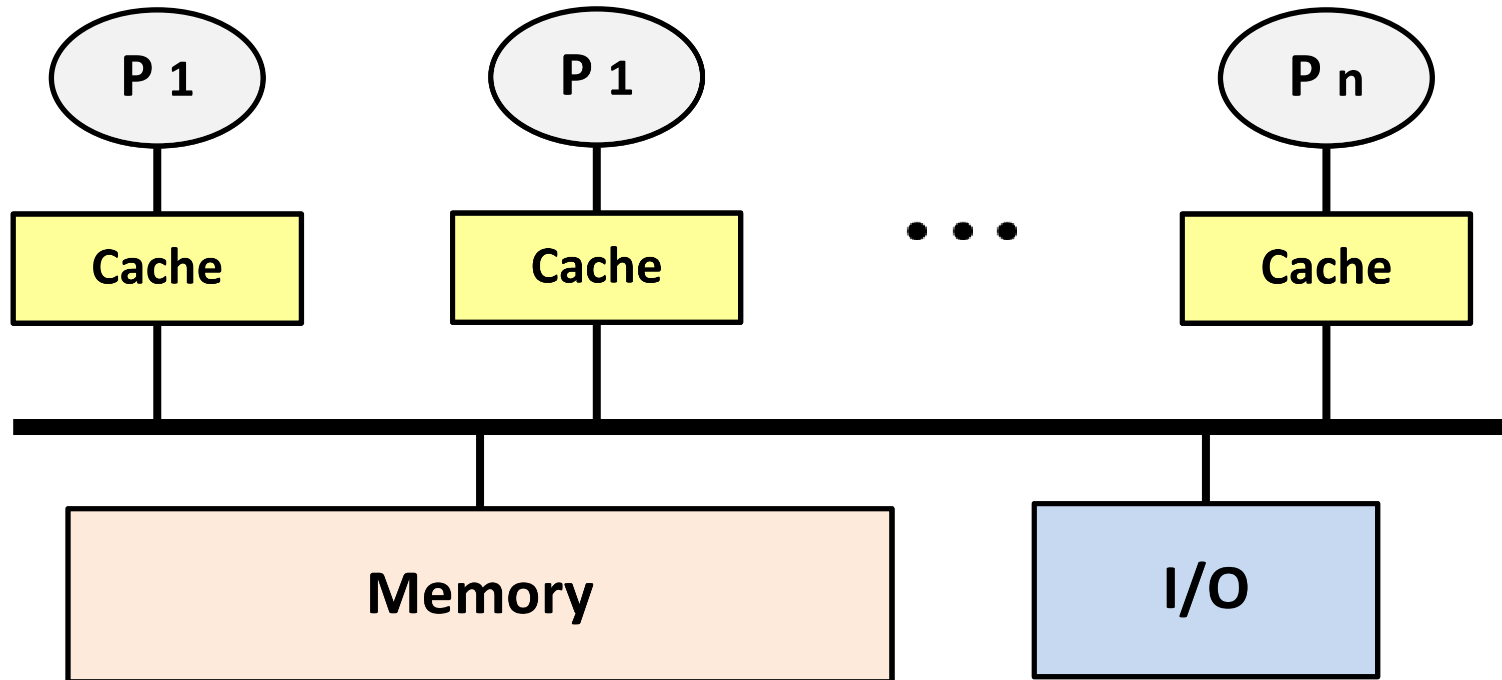
Distributed Shared-Memory

- Physically separate memories for each processor
- Non-uniform memory access (NUMA)

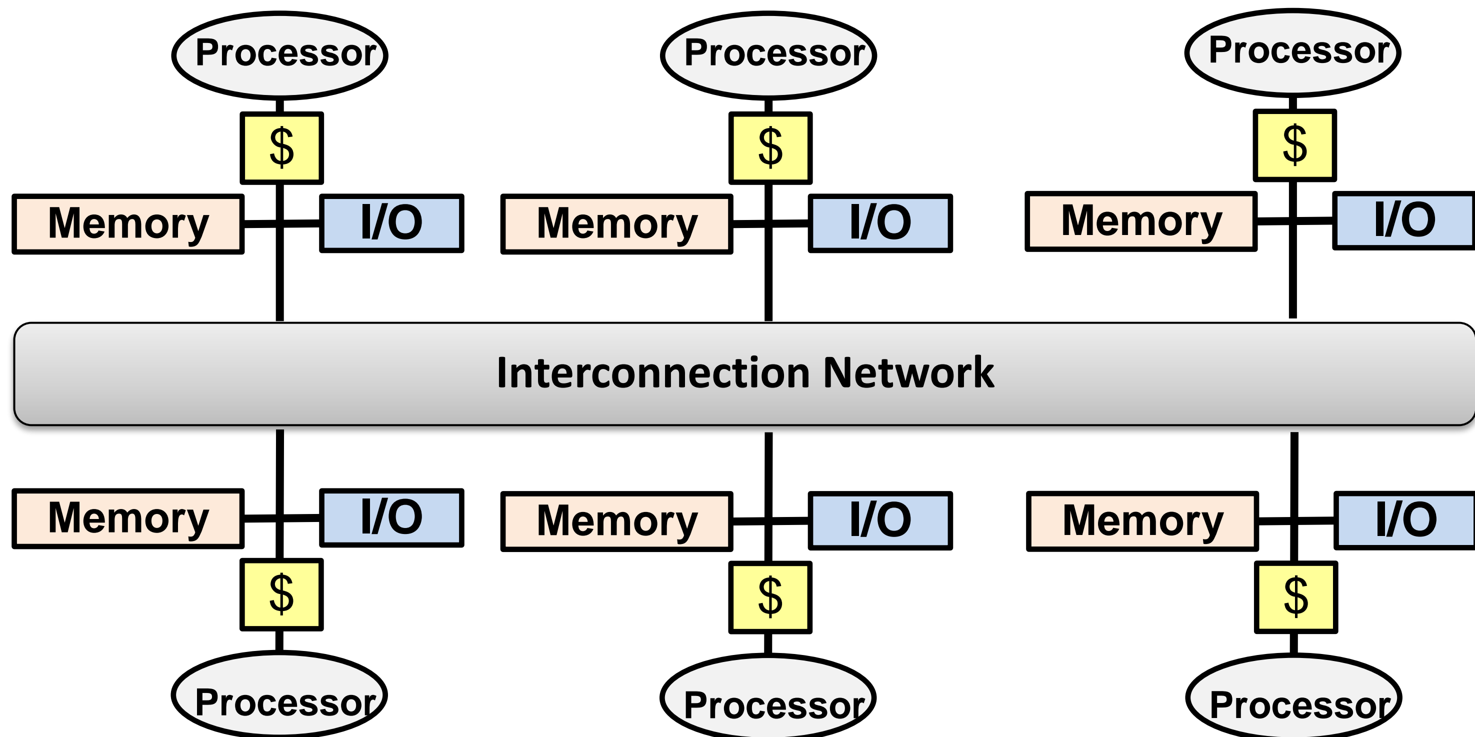
- Classification depending on the memory organization
- Shared memory means the address space is shared

Centralized Shared Memory

- Often referred to as symmetric multiprocessors (SMPs)
  - Processors have equal access time to any memory locations



Distributed Shared Memory (DSM)



- Two major benefits
  - Scalable memory bandwidth
  - Low-latency local access
- Key disadvantages
  - Complex communication
  - Higher node-node latency

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

Comparison of Different MIMD Systems

- **Multiprocessors (both SMP and DSM)**
  - Tightly coupled architecture
  - Processors connected via bus or interconnect network
  - Consists of a few processors (2 ~ dozens)
  - A single shared address space
  - Communicate data implicitly via load and store
  - Thread-level parallelism
- **Multicomputers, Clusters (WSCs)**
  - Loosely coupled architecture
  - Individual computers connected on a local area network
  - Consists of large number of nodes
  - Multiple private address spaces
  - Explicitly passing messages among the processors
  - Request/Task level parallelism

Outlines

- Multiprocessor Architecture
- Cache Coherence Problem
- Snooping Protocol

Expectation of Shared Memory System

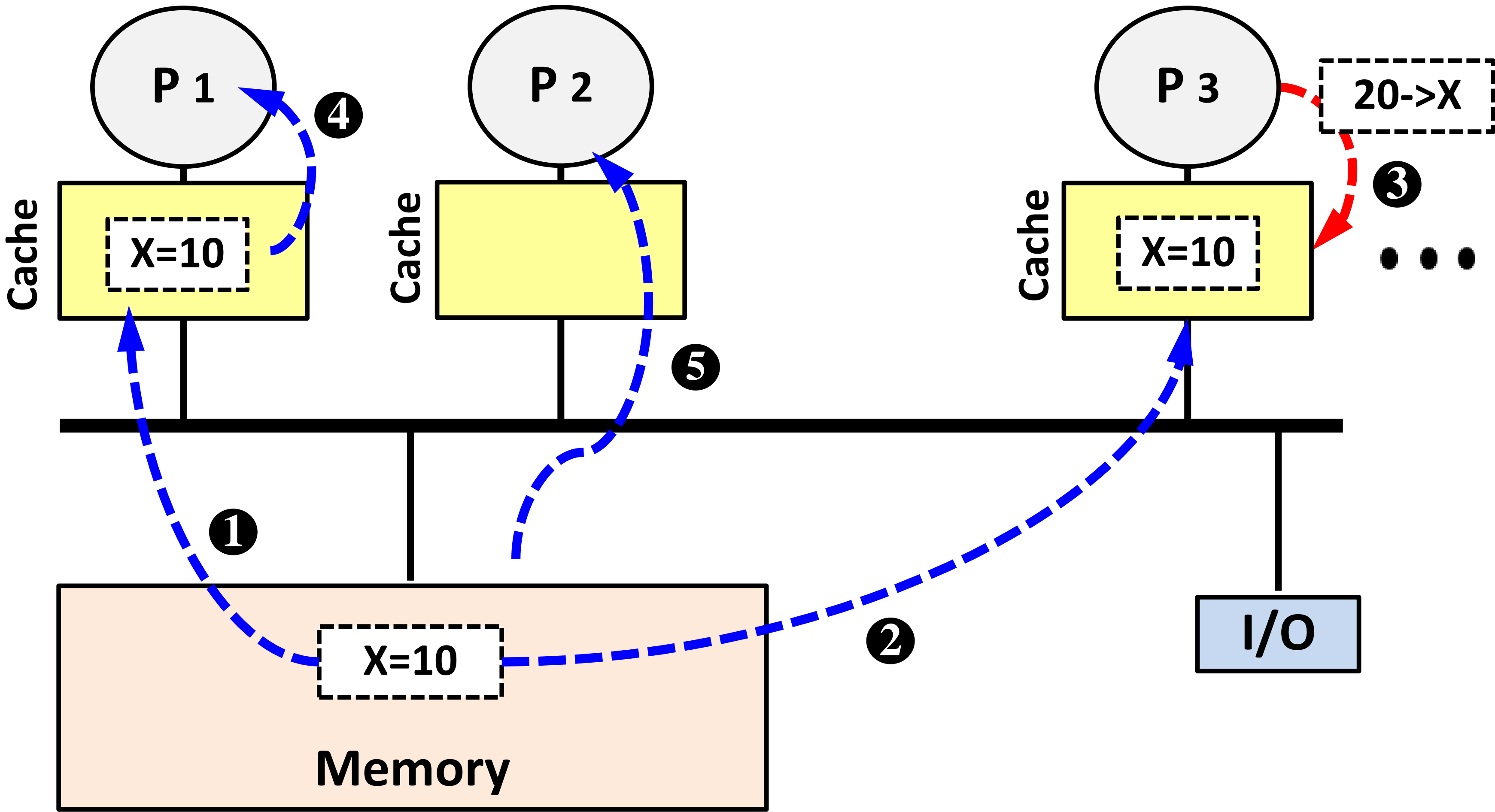
- Leveraging parallelism to improve performance
- But more importantly, the results of a parallel program that uses multiple processes to be no different when the processes run on different physical processors than when they run on the same physical processor

The Cache Coherence Problem

- Both **private** and **shared** data exist
  - Private data (local) is used by a single processor
  - Shared data (global) is used by multiple processors
- Shared data have multiple copies, spread throughout the caches, and are manipulated by different processors
- A cache coherence problem arises when different CPUs see different values for the same memory location
  - e.g. have an incoherent view of the memory

cache coherence : 缓存一致性

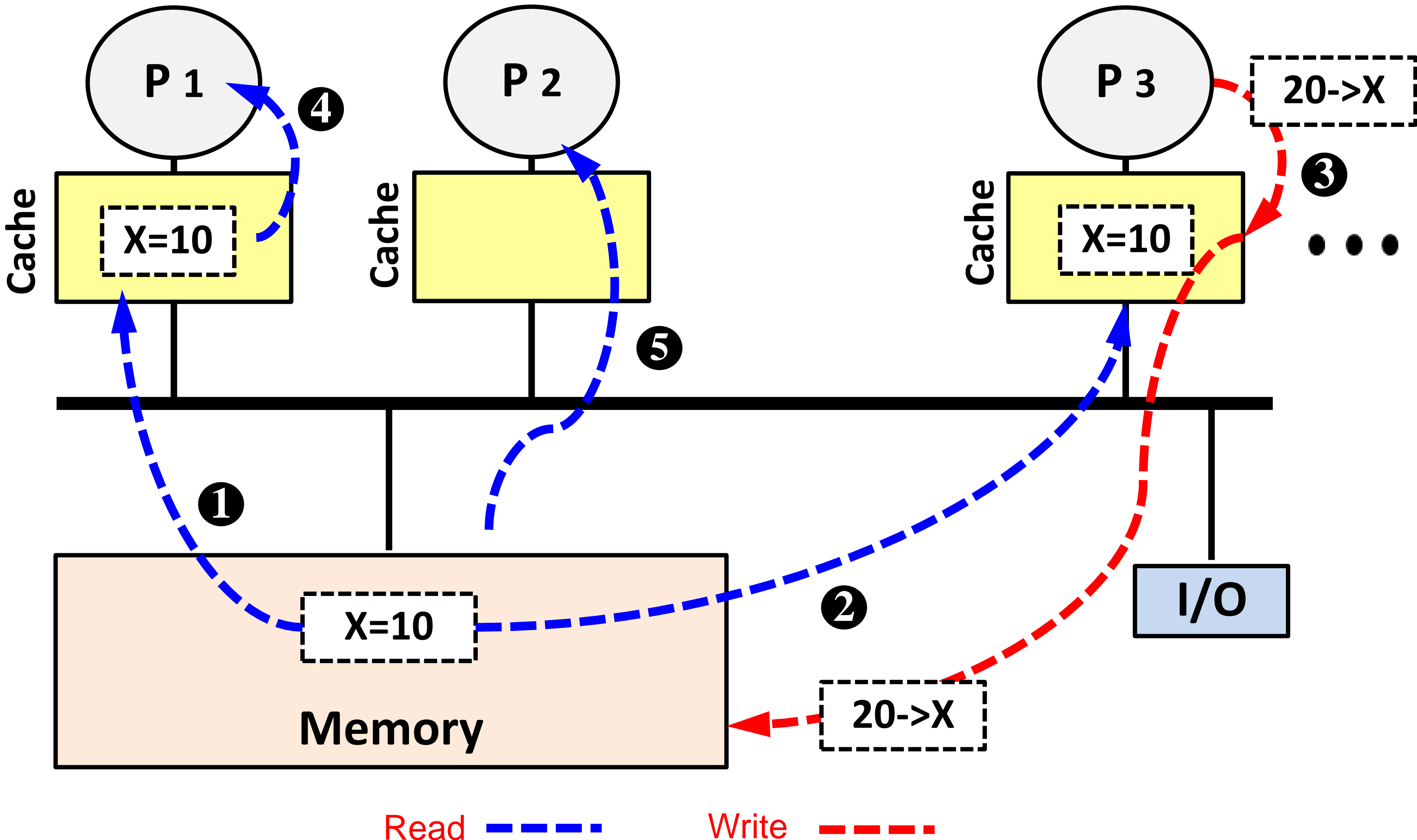
Write-back Cache w/o Coherence



Results: P1: X=10, P2: X=10, P3: X=20

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Write-through Cache w/o Coherence

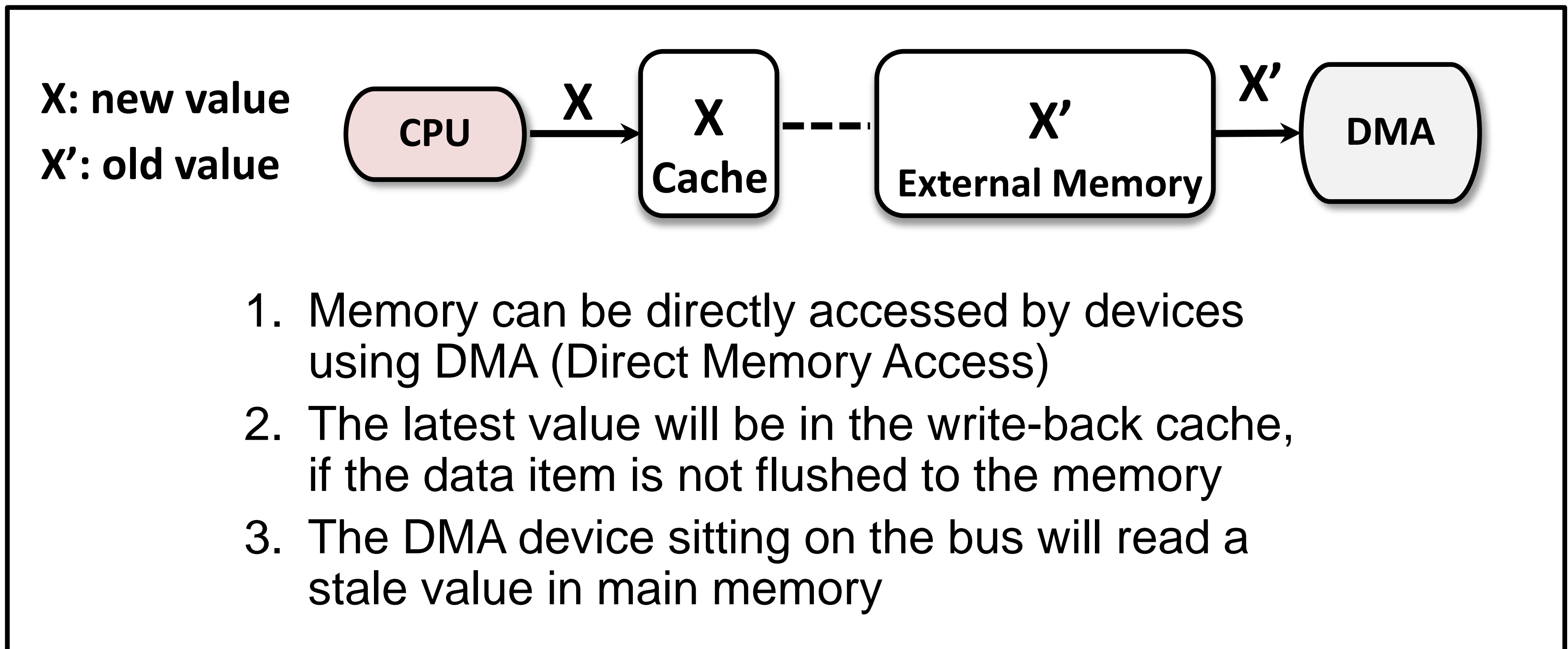


Results: P1: X=10, P2: X=20, P3: X=20

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Coherence Issue in Uniprocessors

- Coherence problems arise even in uniprocessors when certain I/O operations occur



Intuitive Thinking of Coherence

- What a memory should do?

  It provides a set of locations holding values, and when a location is read it should return the last value written to that location

- Similarly, a coherent shared-memory system should be:

  Reading memory address X should return the last value written at address X by any processor

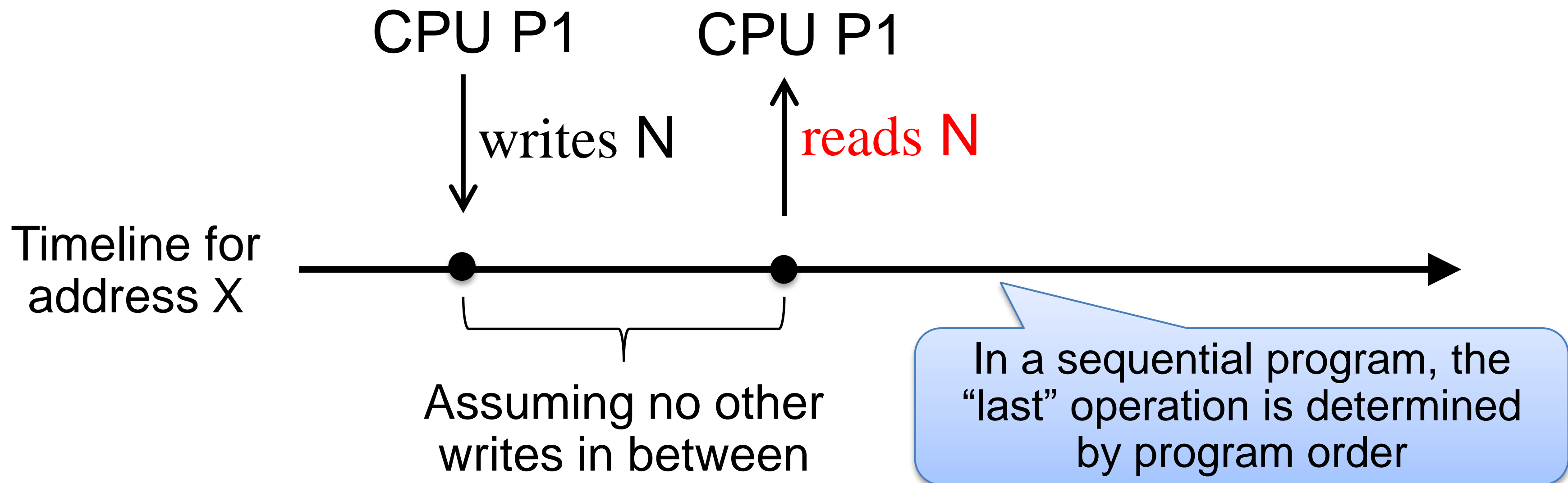
The word "last" may not be well defined for a parallel system

- What if two CPUs write to the same location simultaneously?
- What if the time between read and write is so close?

Cache Coherency: Precise Definition

Condition 1: program order

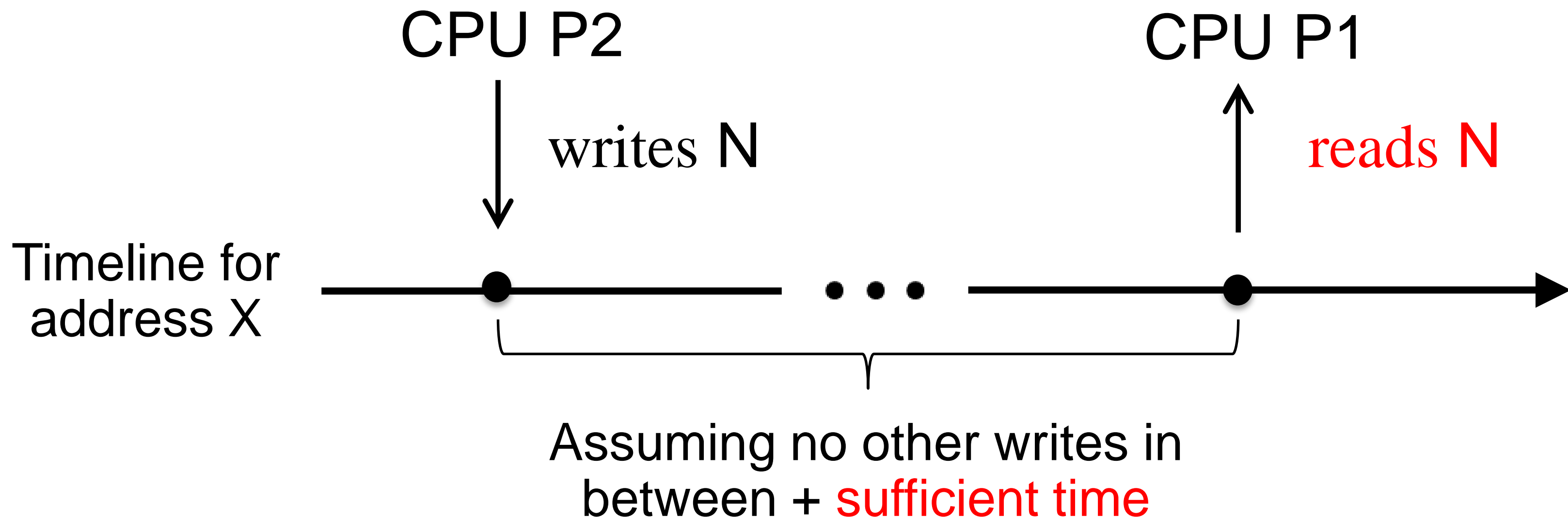
- **"Read after write" works for a single processor**
  - If processor P1 writes N to location X, the following read of X by P1 should return N (if no other writes of X occur in between)



Cache Coherency: Precise Definition

Condition 2: write propagation

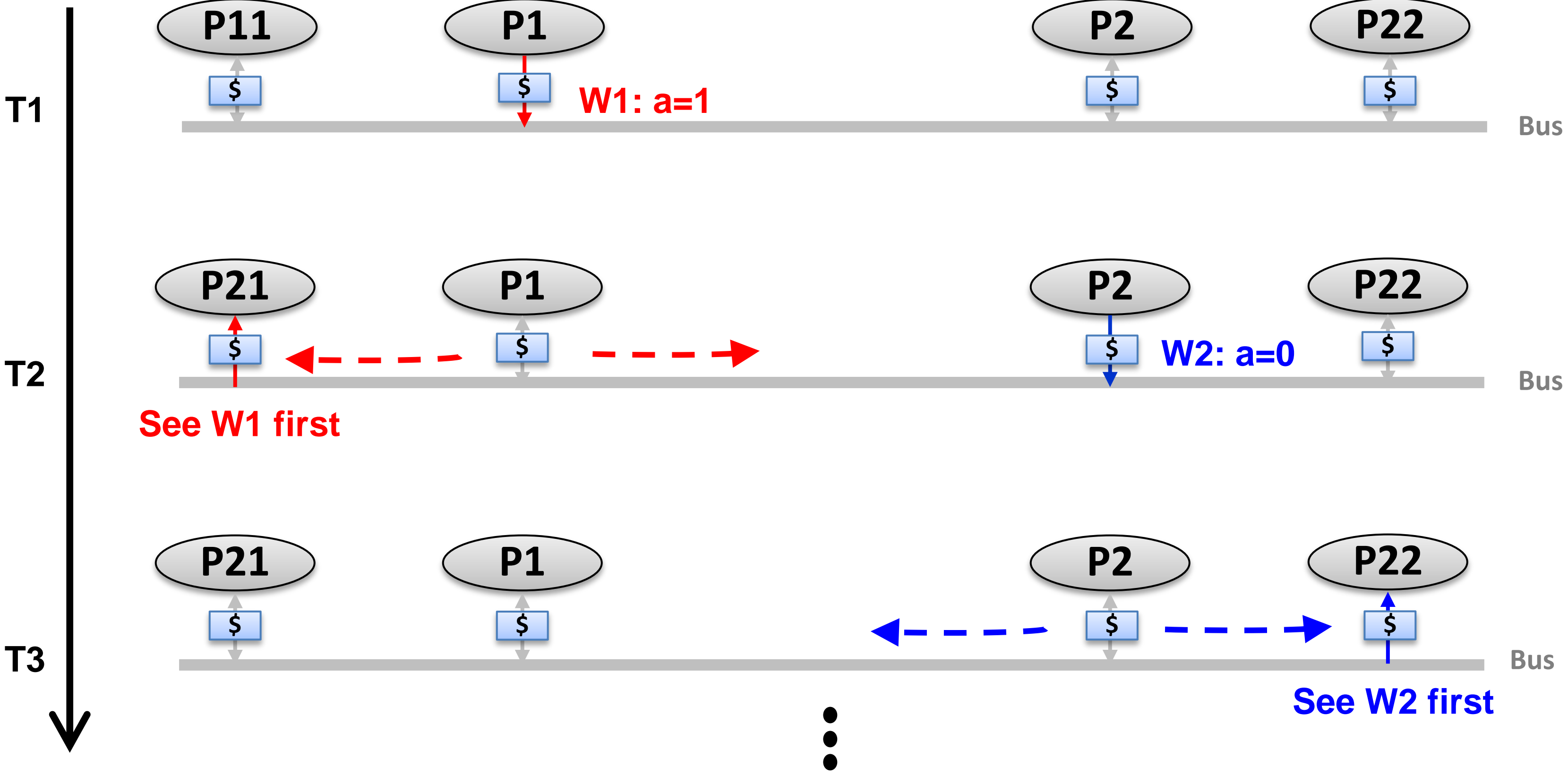
- **Writes can eventually get to the other processors**
  - If another processor P2 writes N to location X, processor P1 will eventually be able to read the updated value from location X.



上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Write Serialization

- Conditions 1 & 2 are necessary, but **not sufficient**

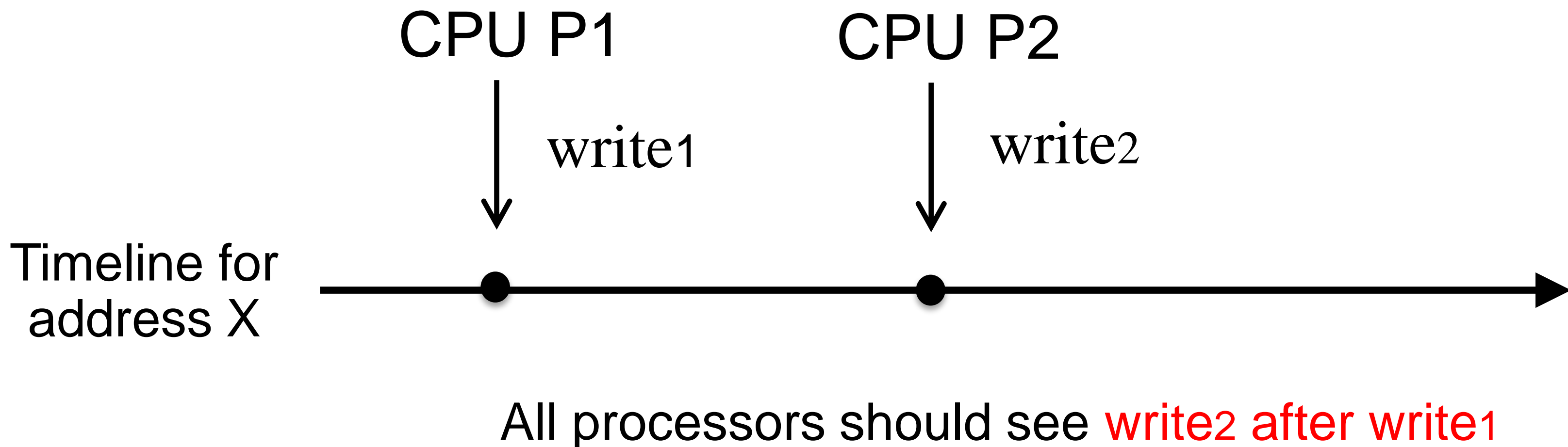


上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Cache Coherency: Precise Definition

Condition 3: write serialization

- **Writes to the same location are serialized**
  - If processors P1 and P2 both write to location X, all processors see the same order of the two writes.



Cache Coherency: A Formal Definition

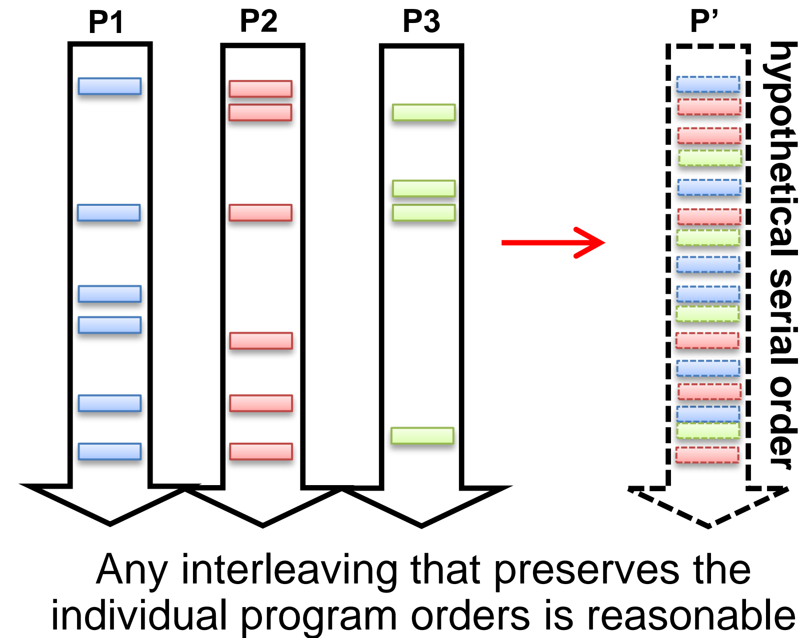
A multiprocessor memory system is coherent if:

- The results of any execution of a parallel program are such that, for each location, it is possible to construct a hypothetical serial order of all operations to the location which is consistent with the results of execution, and:

  1. Memory operations issued by any particular processor occur in the above sequence in the order issued by that processor
  2. The value returned by each read operation is the value written by the last write to that location in the serial order

Cache Coherency: A Formal Definition (Cont'd)

- Think about a shared memory system without caches
  - The memory would impose a serial order on all the read and write operations to the location
  - the reads/writes to the location from any individual processor should be in program order



Any interleaving that preserves the individual program orders is reasonable

Since the serial order must be consistent, it is important that all processors see the writes to a location in the same order

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Coherence vs. Consistency

- Coherence: (一致性，强调读出值的异同)
  - Defines what values can be returned by a read
  - Looks at the same memory location
- Consistency: (一贯性，强调读的时间概念)
  - Defines when a written value will returned by a read
  - Includes operations to other locations

```
/* Assume the initial value of A and the flag is 0 */
P1                P2
A = 1;            while (flag == 0); /*spin idly */
flag = 1;         print A;
```

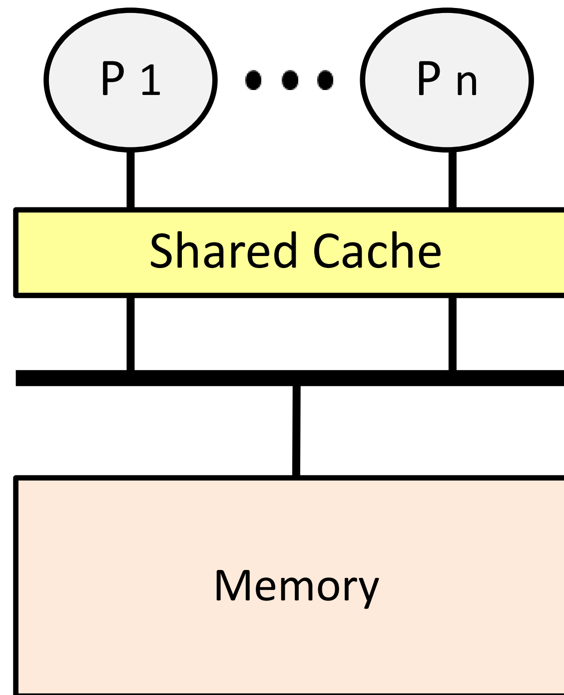
The above program orders (if flag==1, then A=1) within P1 and P2 ‘s accesses are not implied by coherence

Outlines

- Multiprocessor Architecture
- Cache Coherence Problem
- Snooping Protocol

Discussion

What are the benefits and problems of shared-cache system?



Enforcing Coherence

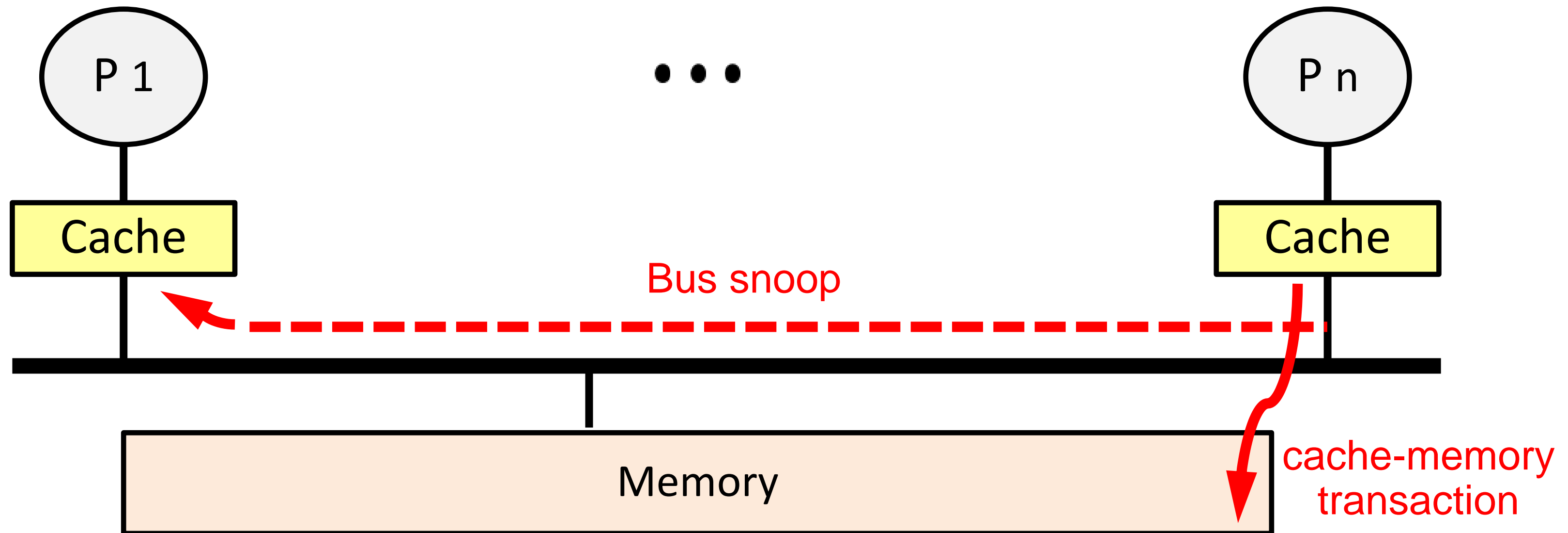
The key to implementing coherence is tracking the state of any sharing of a data block

Two Classes of Protocols

- Snooping Protocol
  - Monitoring all transactions on the interconnect (snooping)
  - Every cache has a copy of the sharing status
  - Normally faster if there is enough bandwidth
- Directory-based Protocol
  - Does not broadcast on the interconnect
  - The sharing status is kept in a single directory
  - Easier to support a large number of processors

Snoop *[snup]* n. 探听，窥探

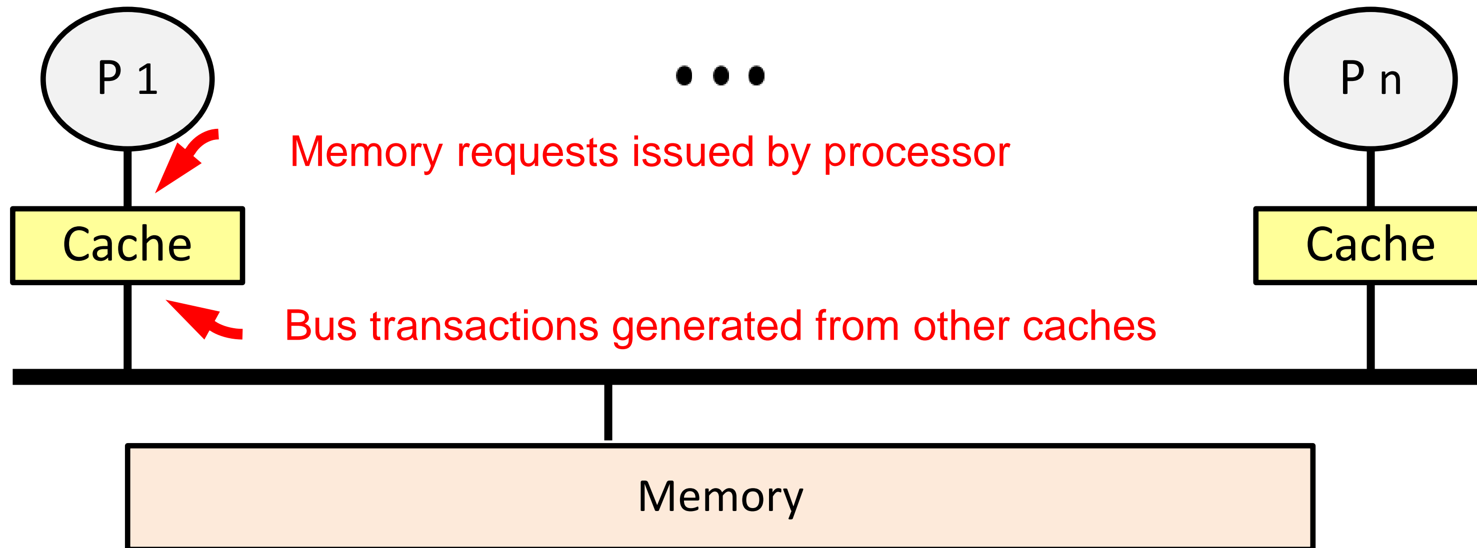
Cache Coherence Through Bus Snooping



Basic Facts

- Multiple processors with local caches are placed on a shared bus
- All the writes will be shown as a transaction on the bus to memory
- All transactions are visible to all processors in the same order
- Each processor continuously “snoops” on the bus

Cache Coherence Through Bus Snooping



Basic Facts

- Enhanced cache: now receives requests from two sides
- The cache can be viewed as having two controllers:
  - A processor-side controller
  - A bus-side controller (snooper)

Snooping Protocol

- The protocol is a distributed algorithm, specified by:
  - A set of states associated with local cache blocks
  - A state transition diagram (a finite state machine)
  - The actual actions associated with each state transition

- Two way to maintain cache coherence
  - Update-based : updating other cached copies on a write
  - Invalidation-based : invalidating other cached copies on a write

Write-updated protocol for write-through caches	Write-update protocol for write-back caches
Write-invalidate protocol for write-through caches	Write-invalidate protocol for write-back caches

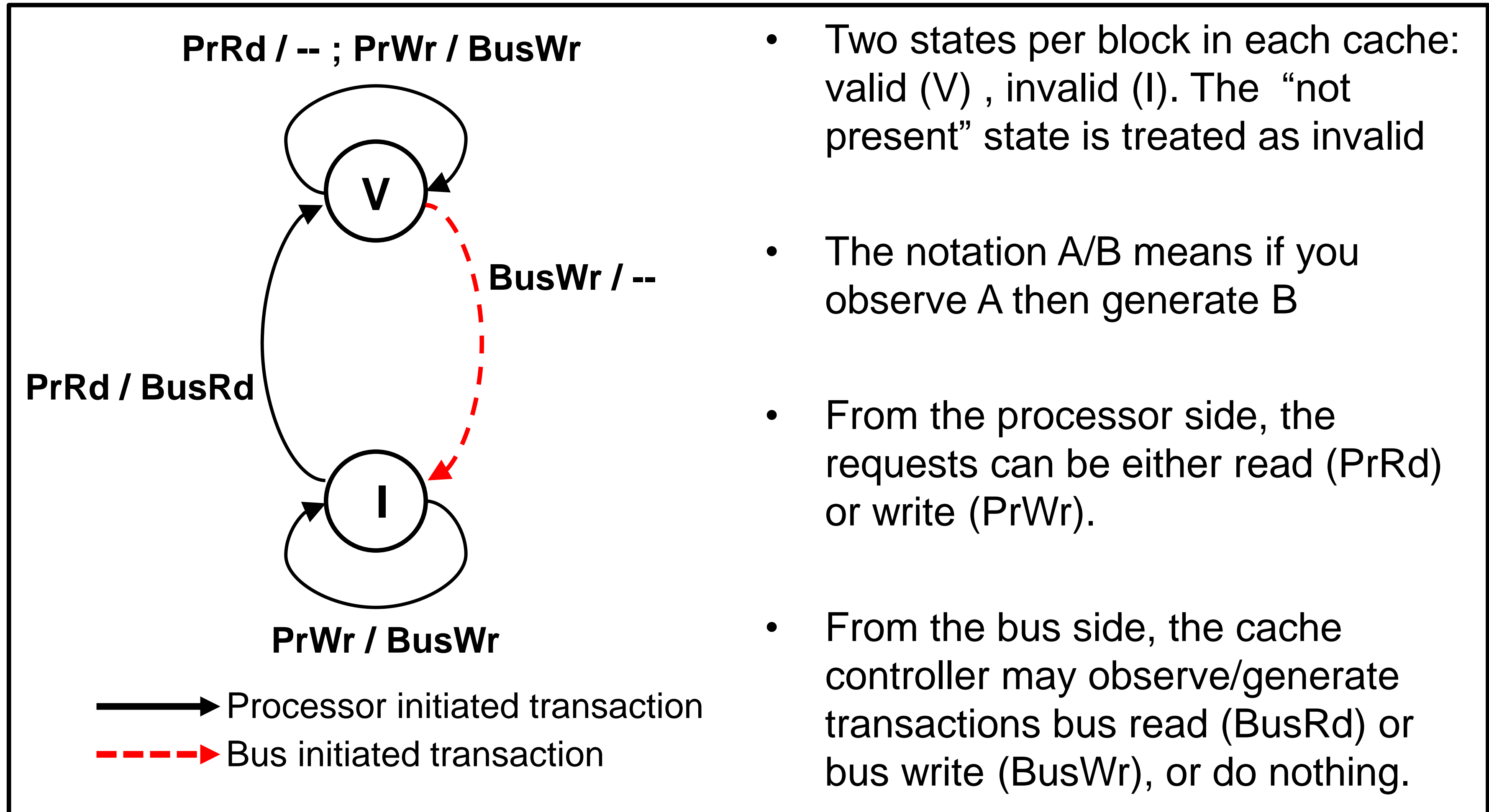
Four combinations of cache coherence designs

A Simple Example: Write-Through Invalidation

- Interconnect and memory transactions are atomic: only one bus transaction is in progress at a time
- All writes to a location are serialized by the order in which they appear on the shared bus (bus order)
- Upon processor write, broadcast invalidation; next read from other processors will trigger cache miss

A Simple Example: Write-Through Invalidation

Invalidation-based coherence protocol for a write-through no-allocate cache



- Two states per block in each cache: valid (V) , invalid (I). The “not present” state is treated as invalid
- The notation A/B means if you observe A then generate B
- From the processor side, the requests can be either read (PrRd) or write (PrWr).
- From the bus side, the cache controller may observe/generate transactions bus read (BusRd) or bus write (BusWr), or do nothing.

Problems with Write-Through Cache

- High bandwidth requirements with write-through
  - All write operation goes to shared bus and memory

Q. Consider a processor running at 1GHz. Suppose the average CPI is 1.5, and 15% of all instructions are stores, and each store write writes 8 bytes of data. How many processors will a 1 GB/s bus be able to support without becoming saturated?

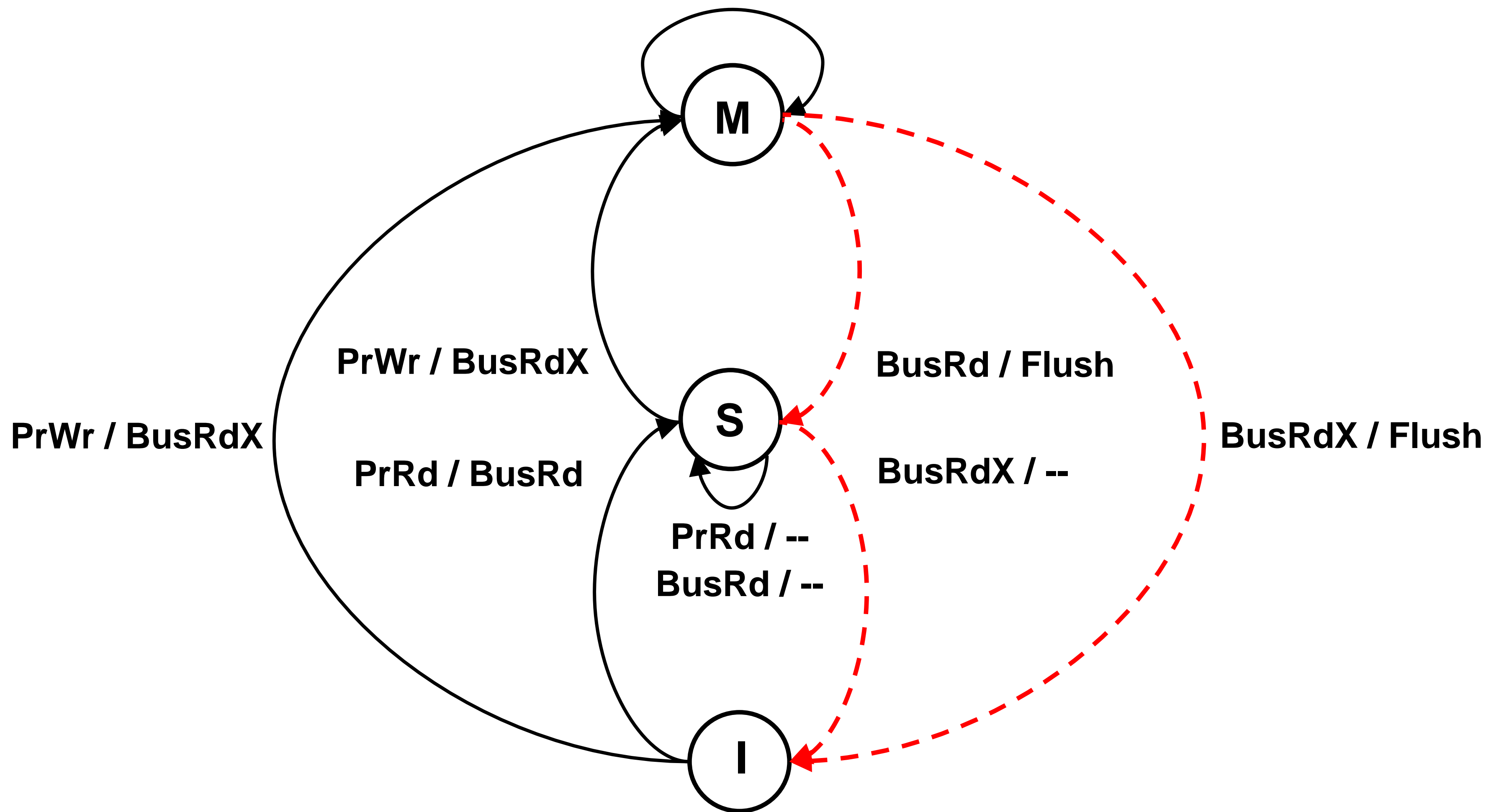
A. 0.15 stores/instruction * (1/1.5) instruction/cycle * 1G cycles per second = 0.1G stores per second. Total write-through bandwidth is 0.8GB of data per second per processor (ignoring read misses and other information). A 1GB/s bus will therefore support only 1 processor.

- Write-back caches saves bandwidth for SMPs
  - Require more sophisticated coherence protocols

A 3-state (MSI) Write-Back Invalidation Protocol

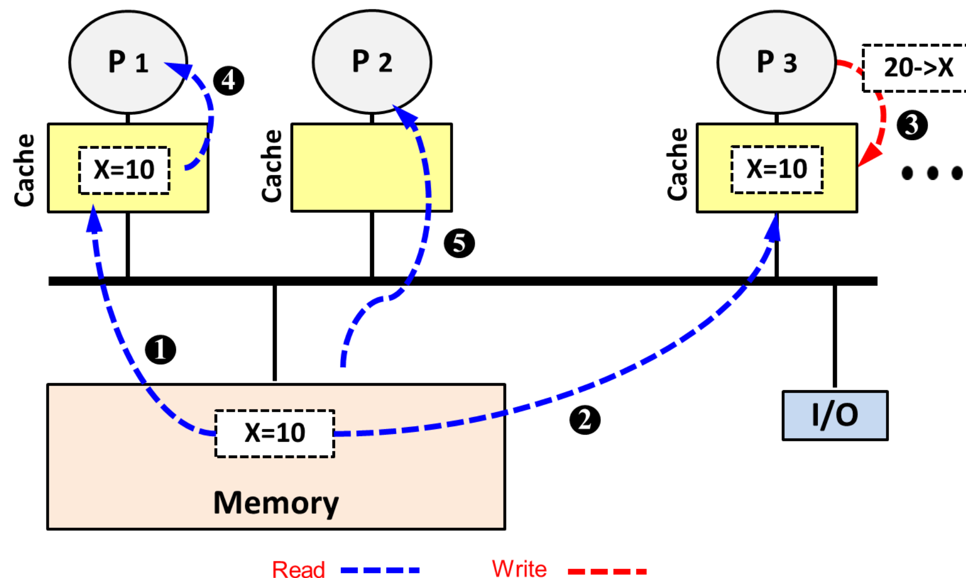
- Saves the bandwidth of the shared bus
- Features 3 states: modified (M), Shared (S), Invalid (I)
- 2 possible processor requests: PrRd and PrWr
- 3 possible bus-side requests:
  - Bus Read (BusRd)
  - Bus Read Exclusive (BusRdX): ensures write propagation
  - Bus Write Back (BusWB)

A 3-state (MSI) Write-Back Invalidation Protocol



上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

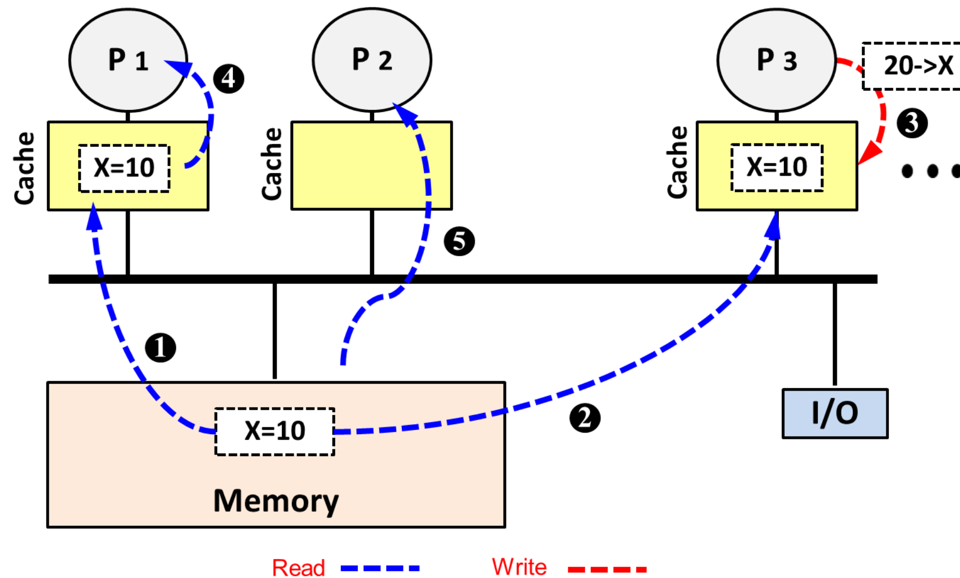
A 3-state (MSI) Write-Back Invalidation Protocol



Processor Action	State in P1	State in P2	State in P3	Bus Action	Data Supplied By
1. P1 reads x					
2. P3 reads x					
3. P3 writes x					
4. P1 reads x					
5. P2 reads x					

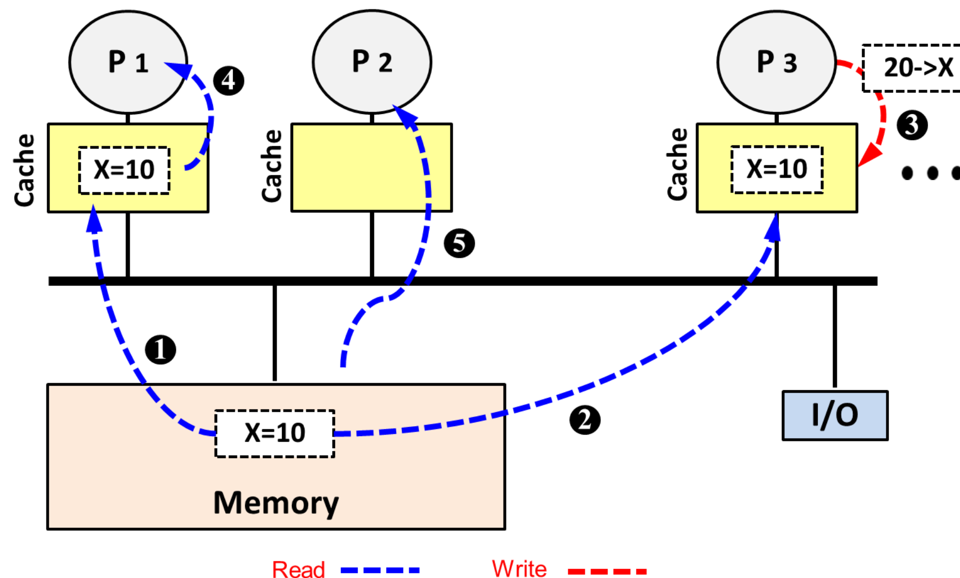
上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

A 3-state (MSI) Write-Back Invalidation Protocol



Processor Action	State in P1	State in P2	State in P3	Bus Action	Data Supplied By
1. P1 reads x					
2. P3 reads x					
3. P3 writes x					
4. P1 reads x					
5. P2 reads x					

A 3-state (MSI) Write-Back Invalidation Protocol

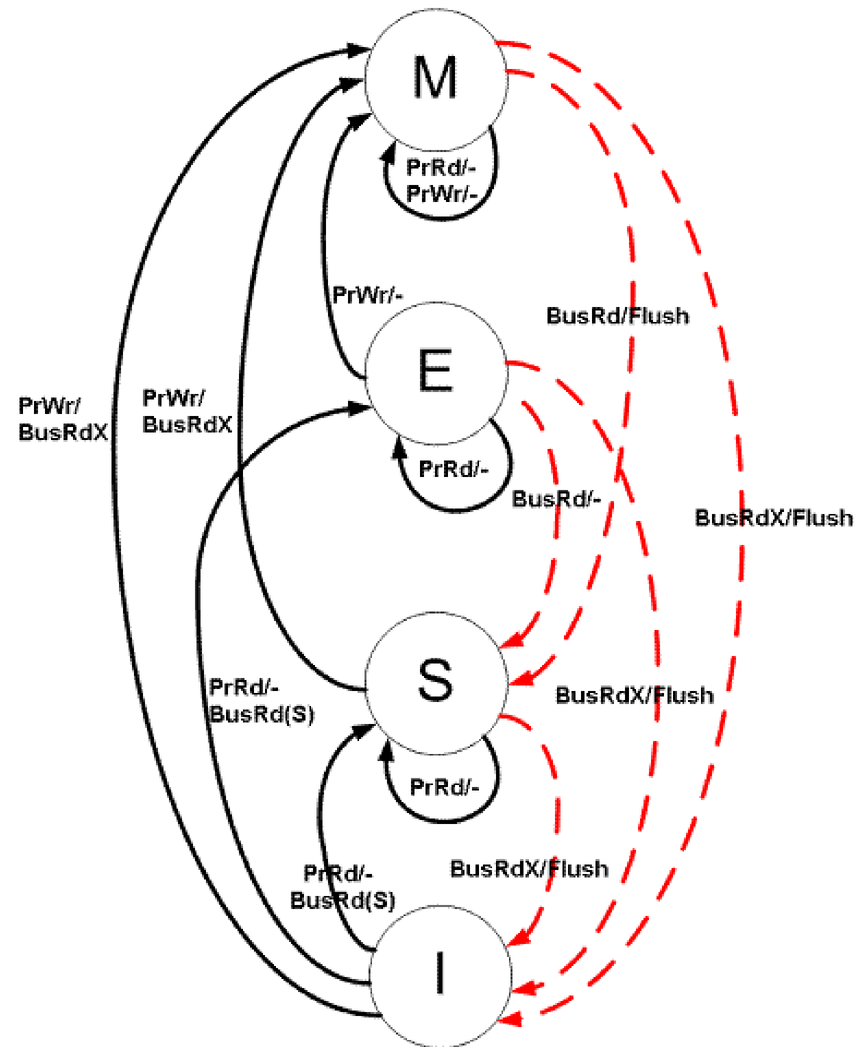


Processor Action	State in P1	State in P2	State in P3	Bus Action	Data Supplied By
1. P1 reads x	S	-	-	BusRd	Memory
2. P3 reads x	S	-	S	BusRd	Memory
3. P3 writes x	I	-	M	BusRdx	Memory
4. P1 reads x	S	-	S	BusRd	P3 Cache
5. P2 reads x	S	S	S	BusRd	Memory

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

A 4-state (MESI) Write-Back Invalidation protocol

- An extension to MSI
- Its variants are used in many modern processors
- Four States:
  - M: Modified
  - E: Exclusive-Clean
  - S: Shared
  - I: Invalid
- The E State means only this cache has a copy and it has not been modified



Summary

- SIMD, MIMD, TLP
- Multiprocessors, UMA and NUMA
- Definition of cache coherency
- Cache coherency and memory consistency
- Basic facts of the snooping protocol
- A simple write-through invalidation protocol
- 3-state MSI protocol

References

- 课本内容：J. Hennessy, D. Patterson. Computer Architecture, Fifth Edition: A Quantitative Approach.
  - Chapters: 5.1, 5.2
- 其它参考：D. Culler et al., 《Parallel Computer Architecture: A Hardware/Software Approach》, Second Edition.
  - Chapters: 5.1, 5.2, 5.3.1

Exercises

- What would happen if a cache has the block in modified state and it observes a BusRd transaction on the bus?

- How would register allocation affect semantics in a parallel program (showing below) running on a multiprocessor?

```
/* Assume the initial value of A and flag is 0 */
P1                P2
A = 1;            while (flag == 0); /*spin idly */
flag = 1;         print A;
```