

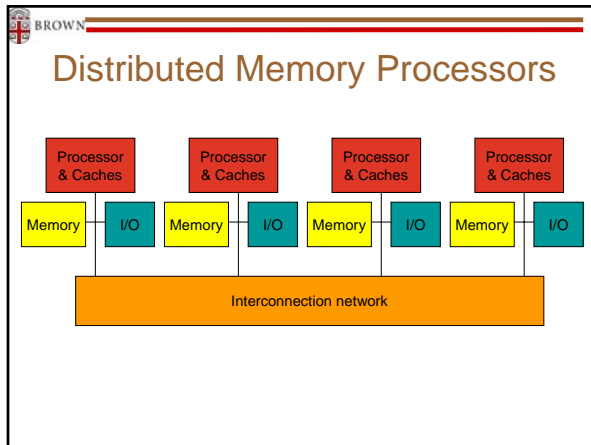
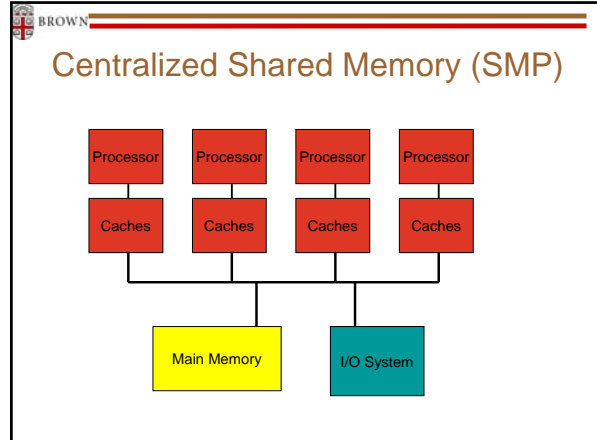


# Computer System Design

## Lecture 31: Coherence Protocols

Prof. R. Iris Bahar  
EN164  
April 18, 2007

Reading: Chapter 4, section 4.2, 4.3



### 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

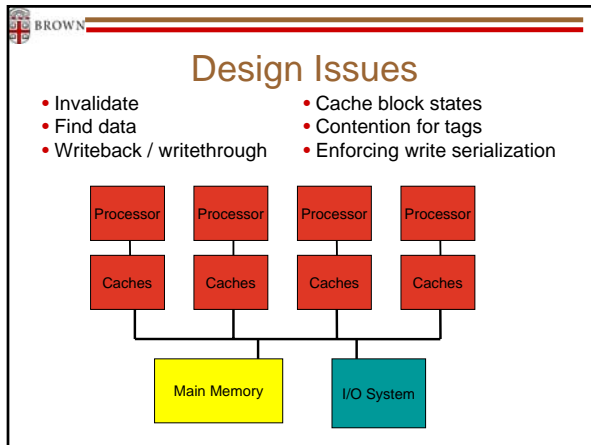
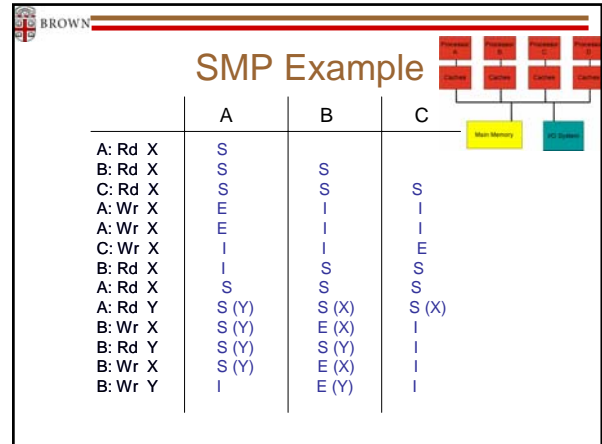
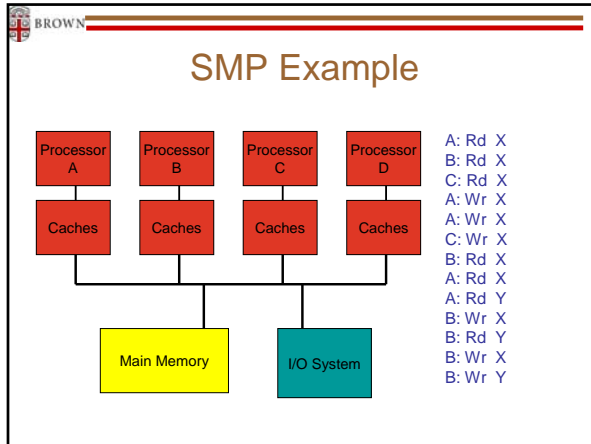
### Coherence Protocols

- Two conditions for cache coherence:
  - write propagation
  - write serialization
- Cache coherence protocols:
  - snooping**: all cache controllers monitor the bus to update sharing status of blocks
  - directory-based**: directory keeps track of sharing status
    - write-update (broadcast)
    - write-invalidate

### Snooping Protocol

- Two protocols possible on a snooping bus:
  - Write invalidate
  - Broadcast

Event	Bus Activity	Value of X in		
		Cache A	Cache B	Memory
		-	-	0
CPU A reads X	Cache miss for X	0	-	0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1



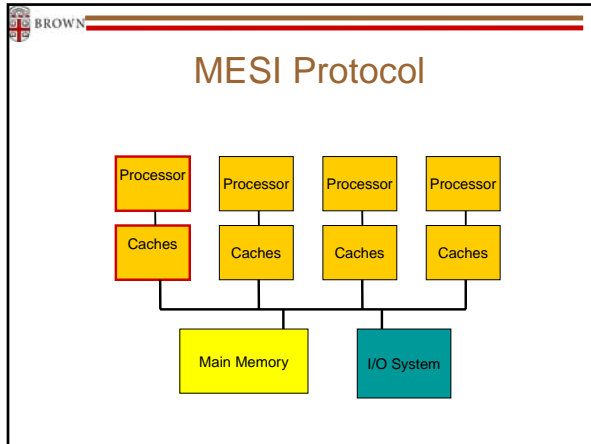
### Example Protocol

Request	Source	Block state	Action
Read hit	Proc	Shared/excl	Read data in cache
Read miss	Proc	Invalid	Place read miss on bus
Read miss	Proc	Shared	Conflict miss: place read miss on bus
Read miss	Proc	Exclusive	Conflict miss: write back block, place read miss on bus
Write hit	Proc	Exclusive	Write data in cache
Write hit	Proc	Shared	Place write miss on bus
Write miss	Proc	Invalid	Place write miss on bus
Write miss	Proc	Shared	Conflict miss: place write miss on bus
Write miss	Proc	Exclusive	Conflict miss: write back, place write miss on bus

### Example Protocol (cont.)

Request	Source	Block state	Action
Read miss	Bus	Shared	No action; allow memory to respond
Read miss	Bus	Exclusive	Place block on bus; change to shared
Write miss	Bus	Shared	Invalidate block
Write miss	Bus	Exclusive	Write back block; change to invalid

- ### MESI Protocol
- Most common model. Used in Pentium family.
  - Each cache line can be in one of the following states:
    - **Modified:** line resides exclusively in this cache only. Content is modified relative to memory
    - **Exclusive:** line resides exclusively in this cache only. Content is same as memory
    - **Shared:** line resides in this cache but may be shared with other. Content is same as memory
    - **Invalid:** Line contains no valid memory copy
  - The idea: modified/exclusive lines are "owned" by the cache
    - They can be changed w/o telling other caches
  - Attempt to access a non-owned line should be broadcasted. If owned by another: the owner should update the world and give up ownership



BROWN

## 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

BROWN

## How Do Cache Misses Scale?

	Compulsory	Capacity	Conflict	Coherence	
				True	False
Increasing cache capacity					
Increasing processor count					
Increasing block size					
Increasing associativity					

BROWN

## How Do Cache Misses Scale?

	Compulsory	Capacity	Conflict	Coherence	
				True	False
Increasing cache capacity	No affect	Decrease	Decrease	Slight increase	Slight increase
Increasing processor count	Slight increase	No affect	No affect	Increase	Increase (some)
Increasing block size	Decrease	Slight decrease	Slight decrease	Decrease	Increase
Increasing associativity	No affect	No affect	Decrease	Slight increase?	Slight increase?

Results may vary with the applications being run on the processors

BROWN

## Simplifying Assumptions

- All transactions on a read or write are atomic
  - on a write miss, the miss is sent on the bus, a block is fetched from memory/remote cache, and the block is marked exclusive
- Potential problem if the actions are non-atomic: P1 sends a write miss on the bus, P2 sends a write miss on the bus:
  - since the block is still invalid in P1, P2 does not realize that it should write after receiving the block from P1 – instead, it receives the block from memory
- Fix by keeping track of more state:
  - for example, don't acquire the bus unless all outstanding transactions for the block have completed