1

Computer Organization & Architecture

- Multiprocessor Architecture
  - Basics of Multiprocessor Architecture
    - Flynn's Taxonomy –Recall
    - Challenges in Multiprocessor Architecture
      - Interconnect Network
      - Memory Architecture
  - Data Coherency
    - Issues
    - Protocols

2

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Flynn's Taxonomy
  - Micheal Flynn in 1966 proposed a classification for exploitation techniques of Parallelism by the hardware.
  - He classified all computers into FOUR classes based on the instruction and data streams being used by them

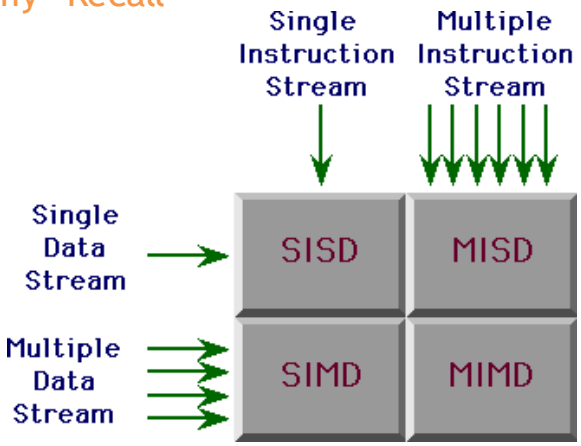
① Single instruction stream, single data stream (SISD)
② Single instruction stream, multiple data streams (SIMD)
③ Multiple instruction streams, single data stream (MISD)
④ Multiple instruction streams, multiple data streams (MIMD)

3

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Flynn's Taxonomy – Recall

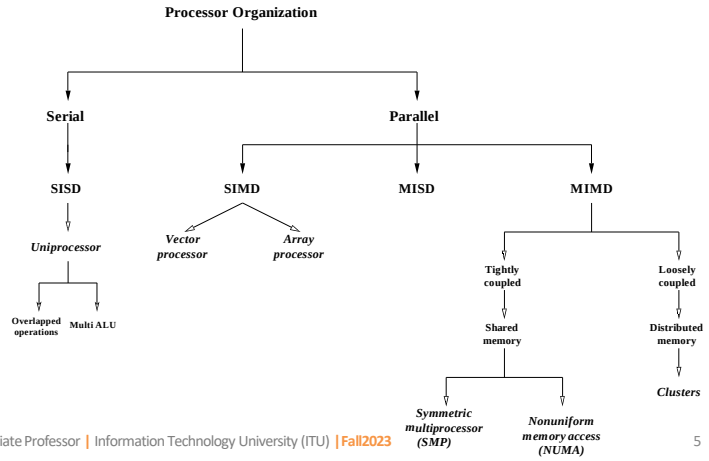


4

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Flynn's Taxonomy – Recall



5

ACA: Memory Hierarchy

Multiprocessor Architecture

- By Definition?
- Computers consisting of tightly coupled processors whose coordination and usage are typically controlled by a single operating system and that share memory through a shared address space.
- Such systems exploit thread-level parallelism through two different software models
  - Execution of tightly coupled set of threads collaborating on a single task, which is typically called parallel processing.
  - Execution of multiple, relatively independent processes that may originate from one or more users, which is a form of request-level parallelism

6

ACA: Memory Hierarchy

Multiprocessor Architecture

- Motivation
  - Difficult to make single-core clock frequencies even higher
  - Deeply pipelined circuits:
    - Leakage Power, Heat Problems
    - Difficult design and verification
    - Large design teams necessary
    - Server farms need expensive air-conditioning
  - Many new applications are multi-threaded
  - General trend in computer architecture (shift towards more parallelism)

7

ACA: Memory Hierarchy

Multiprocessor Architecture

- Motivation –Instruction Level Parallelism (ILP)
  - Parallelism at the machine-instruction level
  - The processor can re-order, pipeline instructions, split them into microinstructions, do aggressive branch prediction, etc.
  - Instruction-level parallelism enabled rapid increases in processor speeds over the last 15 years

8

ACA: Memory Hierarchy

Multiprocessor Architecture

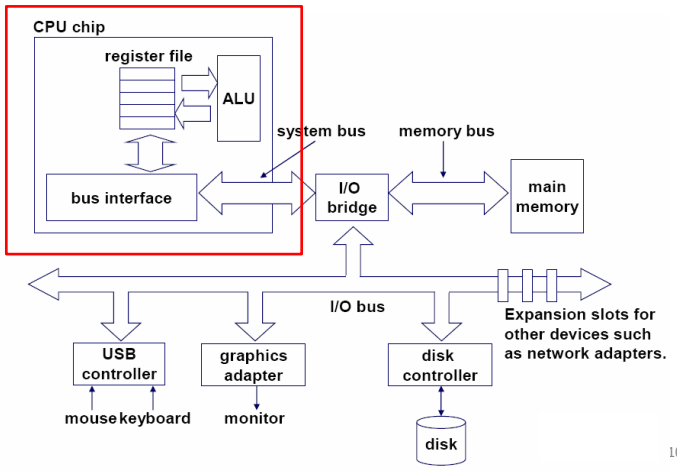
- Motivation –Thread Level Parallelism (TLP)
  - Parallelism on a more coarser scale
  - Server can serve each client in a separate thread (Web server, database server)
    - A computer game can do AI, graphics, and physics in three separate threads
  - Single-core superscalar processors cannot fully exploit TLP
  - Multi-core architectures are the next step in processor evolution: explicitly exploiting TLP

9

ACA: Memory Hierarchy

Multiprocessor Architecture

- Multi-Core Computers

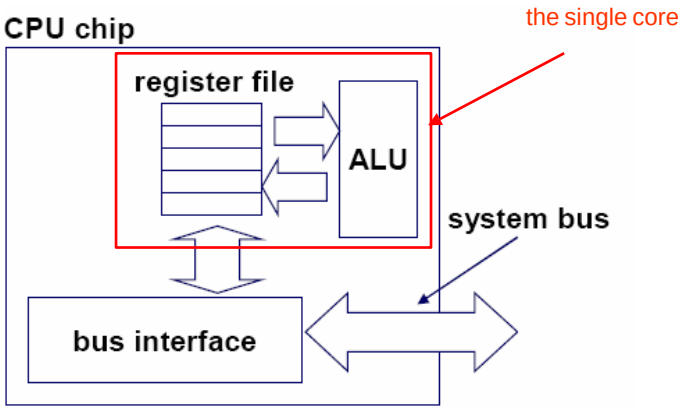


10

ACA: Memory Hierarchy

Multiprocessor Architecture

- Multi-Core Computers

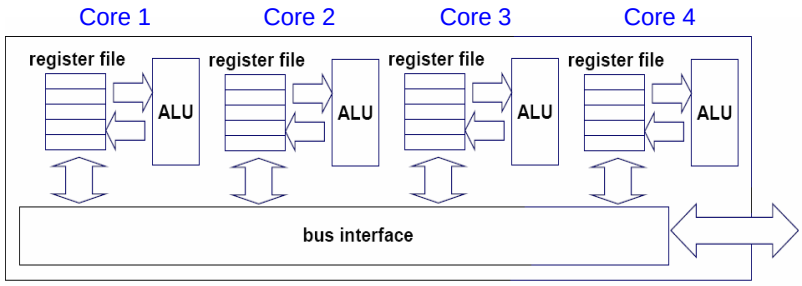


11

ACA: Memory Hierarchy

Multiprocessor Architecture

- Multi-Core Computers



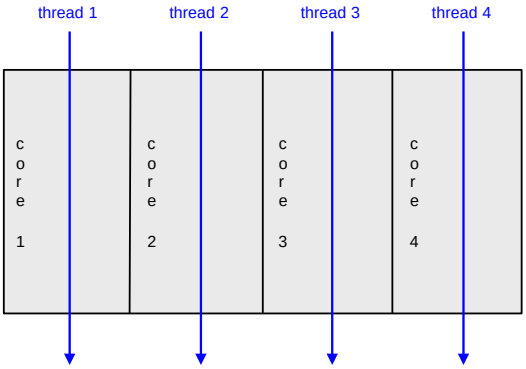
Multi-core CPU chip

12

ACA: Memory Hierarchy

Multiprocessor Architecture

- Multi-Core Computers



13

ACA: Memory Hierarchy

Multiprocessor Memory Architecture

- Shared Memory Architecture
- Distributed Memory Architecture

14

ACA: Memory Hierarchy

Multiprocessor Memory Architecture

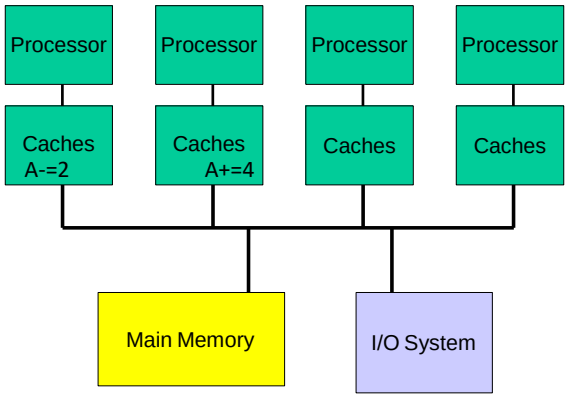
- Shared Memory Architecture:
  - In this model, there is one (large) common shared memory for all processors
    - All processors can access the entire memory address space
  - Called Centralized Shared-memory Multiprocessor or *Symmetric shared-memory multiprocessor (SMP)*
  - Since all processors see the same memory organization, arrangement is called Uniform Memory Access (UMA)

15

ACA: Memory Hierarchy

Multiprocessor Memory Architecture

- Shared Memory Architecture:



16

ACA: Memory Hierarchy

Multiprocessor Memory Architecture

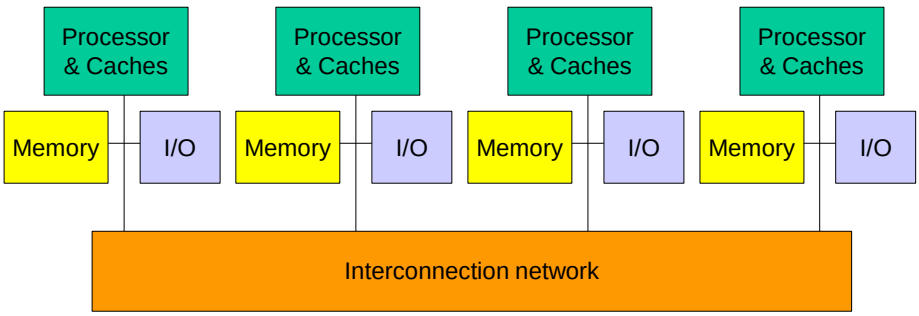
- Distributed Memory:
  - In this model, each processor has its own (small) local memory, and its content is not replicated anywhere else
    - For higher scalability, memory is distributed among processors
    - If memories are strictly local, message-passing is used to communicate data
  - Non-Uniform Memory Architecture (NUMA), Since local memory has lower latency than remote memory

17

ACA: Memory Hierarchy

Multiprocessor Memory Architecture

- Distributed Memory:



18

ACA: Multiprocessor Architecture

Design Challenges

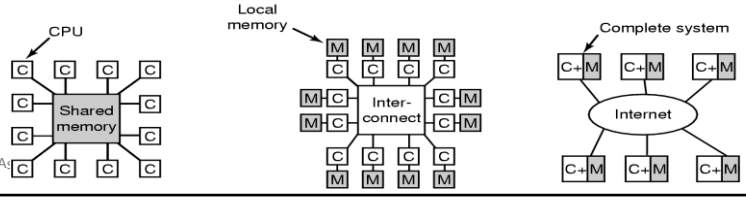
- Distributed Vs Shared Memory
  - Shared Memory
    - Well-understood programming model
    - Communication is implicit and hardware handles protection
    - Hardware-controlled caching
  - Distributed Memory
    - Message passing
    - No cache coherence -Simpler hardware
    - Explicit communication -easier for the programmer to restructure code
    - Sender can initiate data transfer

19

ACA: Multiprocessor Architecture

Design Challenges

- Multiprocessors
  - Multiple CPUs with shared memory
  - Memory access delays about 10 – 50 nsec
- Multicomputers
  - Multiple computers, each with own CPU and memory, connected by a high-speed interconnect
  - Tightly coupled with delays in micro-seconds
- Distributed Systems
  - Loosely coupled systems connected over Local Area Network (LAN), or even long-haul networks such as Internet
  - Delays can be seconds, and unpredictable



20

ACA: Multiprocessor Architecture

Design Challenges

- Transparency: How to achieve a single-system image
  - How to hide distribution of memory from applications?
  - How to maintain consistency of data?
- Performance: How to improve/maintain
  - How to exploit parallelism?
  - How to reduce communication delays?
- Scalability: As more components (e.g., processors) are added, performance should not degrade
  - Centralized schemes (e.g. broadcast messages) don't work
- Security: Against Side Channel Attacks & other malicious applications

21

ACA: Multiprocessor Architecture

Performance Issues

- Limited Parallelism
  - Makes it difficult to achieve good speedups
  - Must be solved primarily at software level
  - Algorithms that provide more extractable parallelism
- Large Latency
  - Limits scalability, sharing of memory
  - Both software and hardware mechanisms can help
    - Caching of shared data (hardware mechanism)
    - Restructuring of data to make more accesses local (Software mechanism)

22

ACA: Memory Hierarchy

Memory Coherence & Consistency

- Many processors can have locally cached copies of the same object
  - Level of granularity can be an object or a block
- Wish List:
  - We want to maximize concurrency
    - If many processors just want to read, then each one can have a local copy, and reads won't generate any bus traffic
  - We want to ensure coherence
    - Processors should always work on the latest copy of data
- Coherence refers to a logically consistent global ordering of reads and writes of multiple processors

23

ACA: Multiprocessor Architecture

Memory Coherence & Consistency

- Problem: View of memory held by two different processors is through their individual caches
  - Without any additional precautions, processors could end up seeing two different values
- *Coherence problem* exists because we have both a global state, defined primarily by the main memory, and a local state, defined by the individual caches, which are private to each processor core.

24

ACA: Multiprocessor Architecture

Memory Coherence & Consistency

- Example: View of memory held by two different processors
  - Consider Cache Write-Through Policy

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X			
2	Processor B reads X			
3	Processor A stores 0 into X			

25

ACA: Multiprocessor Architecture

Memory Coherence & Consistency

- Definition of Coherency: A memory system is coherent if any read of a data item returns the most recently written value of that data item
- Coherence, defines what values can be returned by a read
- A coherent memory system must have certain properties

26

ACA: Multiprocessor Architecture

Memory Coherence & Consistency

- A coherent memory system must have following properties:
  1. A read by processor P to location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P
  2. A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.
  1. Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors. For istance, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1.

27

ACA: Multiprocessor Architecture

Memory Coherence & Consistency

- Definition of Consistency: Memory consistency determines *when* a written value will be returned by a read.
- Consistency tells exactly when a written value must be seen by any reader
  - Example: A write of X on one processor precedes a read of X on another processor by a very small time
  - It may be impossible to ensure that the read returns the value of the data written, since the written data may not even have left the processor at that point

28

ACA: Multiprocessor Architecture

Memory Coherence & Consistency

- Consistency: Following properties must hold to ensure consistency:
  - A write does not complete (and allow the next write to occur) until all processors have seen the effect of that write
  - The processor does not change the order of any write with respect to any other memory access

29

ACA: Multiprocessor Architecture

Memory Coherence & Consistency

- Coherence & Consistency are complimentary

Coherence defines the behavior of reads and writes to the same memory location

Consistency defines the behavior of reads and writes with respect to accesses to other memory locations

30

ACA: Multiprocessor Architecture

Enforcing Coherency

- In a coherent multiprocessor, the caches provide both *migration* and *replication* of shared data items
  - Data Migration:
    - Data item can be moved to a local cache and used there in a transparent fashion
    - Reduces both the latency to access a shared data item that is allocated remotely and the bandwidth demand on the shared memory
  - Data Replication:
    - Caches make a copy of the data item in the local cache
    - Reduces latency of access and contention for a read shared data item

31

ACA: Multiprocessor Architecture

Enforcing Coherency

- The Key to implementing a cache coherence protocol is tracking the state of any sharing of a data block
- *Cache Coherence Protocols (CCPs)*
- Two Classes
  - Directory-Based CCPs
  - Snooping CCPs

32

ACA: Multiprocessor Architecture

Enforcing Coherency

- *Cache Coherence Protocols (CCPs)*
- Directory-Based CCPs
  - The sharing status of a particular block of physical memory is kept in one location referred as the *directory*
  - In an SMP, one centralized directory is used, associated with the memory or some other single serialization point
  - In a DSM, it makes no sense to have a single directory, since that would create a single point of contention
    - Makes it difficult to scale to many multicore chips given the memory demands of multicores with eight or more cores.
    - Distributed directories are more complex than a single directory

33

ACA: Multiprocessor Architecture

Enforcing Coherency

- *Cache Coherence Protocols (CCPs)*
- Snoopy CCPs
  - Rather than keeping the state of sharing in a single directory, every cache that has a copy of the data from a block of physical memory could track the sharing status of the block.
  - In an SMP, the caches are typically all accessible via some broadcast medium (Buses, switches)
  - All cache controllers monitor on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access

34

ACA: Multiprocessor Architecture

Enforcing Coherency

- *Cache Coherence Protocols (CCPs)*
- Snoopy CC Protocols
  - Two techniques of Snoopy CC protocols exist
    - Write Invalidate Protocol:
      - A processor has an EXCLUSIVE access to data item before it writes that item.
      - It invalidates all other copies of data item on a write
      - Exclusive access ensure no other readable or writable copies exist.

35

ACA: Multiprocessor Architecture

Enforcing Coherency

- *Cache Coherence Protocols (CCPs)*
- Snoopy CC Protocols
  - Two techniques of Snoopy CC protocols exist
    - Write Invalidate Protocol: Example

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

36

ACA: Multiprocessor Architecture

Enforcing Coherency

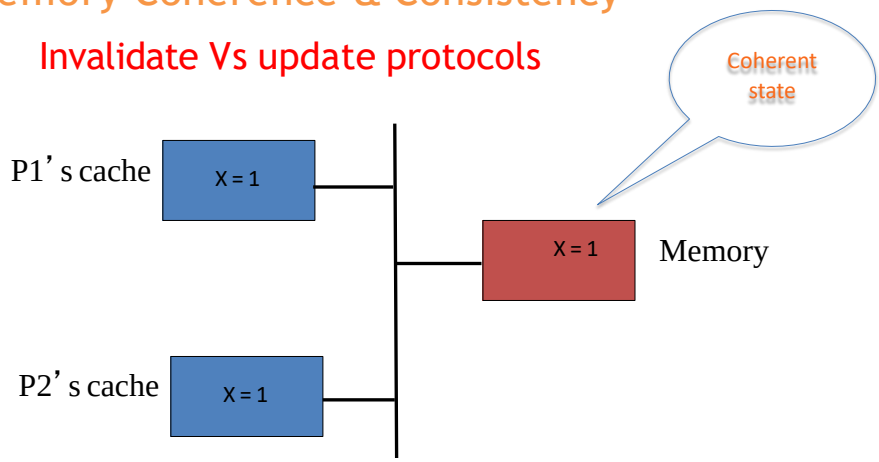
- *Cache Coherence Protocols (CCPs)*
- Snoopy CC Protocols
  - Two techniques of Snoopy CC protocols exist
    - Write Update or Write Broadcast protocol:
      - Alternative to an invalidate protocol
      - Updates main memory and invalidates all the cached copies of a data item when that item is written
      - Write update protocol must broadcast all writes to shared cache lines
      - Consumes considerably more bandwidth

37

ACA: Memory Hierarchy

Memory Coherence & Consistency

- Invalidate Vs update protocols

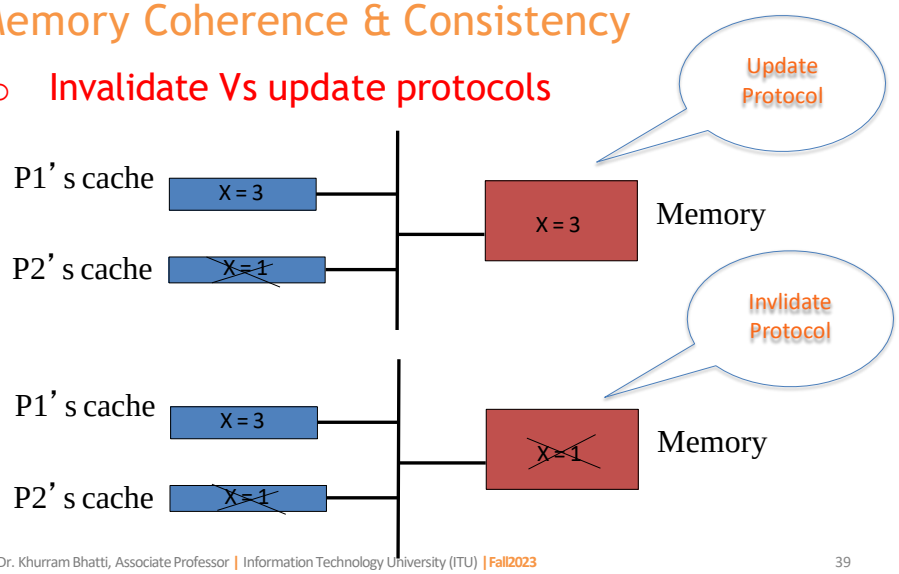


38

ACA: Memory Hierarchy

Memory Coherence & Consistency

- Invalidate Vs update protocols



39

ACA: Memory Hierarchy

Memory Coherence & Consistency

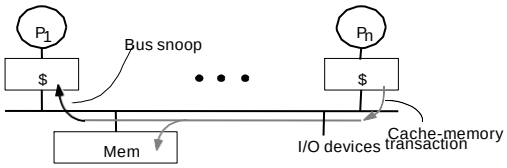
- Invalidate Vs update protocols
  - Update looks the simplest, most obvious and fastest, but:
    - Multiple writes to same word (no intervening read) need only one invalidate message but would require an update for each
  - Due to both spatial and temporal locality, previous cases occur often
    - Bus bandwidth is a precious commodity in shared memory multi-processors
    - Invalidate protocols use significantly less bandwidth

40

41

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP



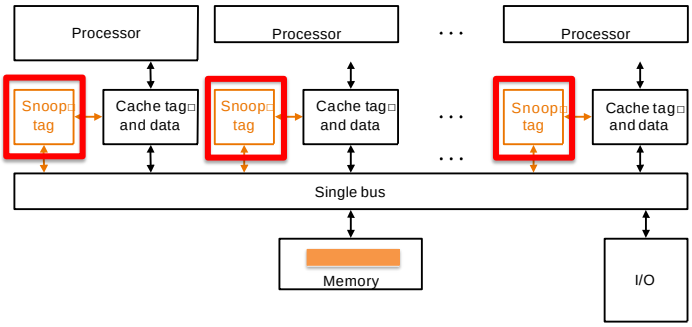
- Bus is a broadcast medium & caches know what they have!
- Cache Controller snoops all transactions on the shared bus
  - A transaction is a relevant transaction if it involves a cache block currently contained in this cache
  - Take action to ensure coherence: invalidate, update, or supply value
  - Action depends on state of the block in cache and the protocol applied

42

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- Structure for Snoopy CCPs



43

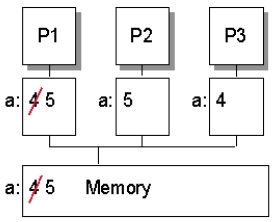
ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- Write Invalidate CC Protocol
  - Analysis of Write-Through (WT) cache coherency using Snoopy Write Invalidate Protocol
    - Independent Processors modifying shared locations can change values without the other processors being aware of it!

P1 reads a into cache
P3 reads a into cache
P1 writes a to 5, and writes through to main memory, correcting it
-- P3 has stale data --
P2 reads a into cache

Memory is arbiter of present value ... move it closer to processors



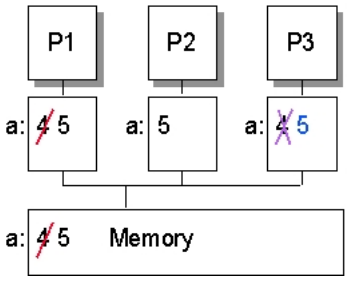
44

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- Write Invalidate CC Protocol
  - Analysis of Write-Through (WT) cache coherency using Snoopy Write Invalidate Protocol

P1 reads a into cache
P3 reads a into cache
P1 writes a to 5, and writes through to main memory, correcting it
P3 sees WT, invalidates or updates
P2 reads a into cache



45

ACA: Multiprocessor Architecture

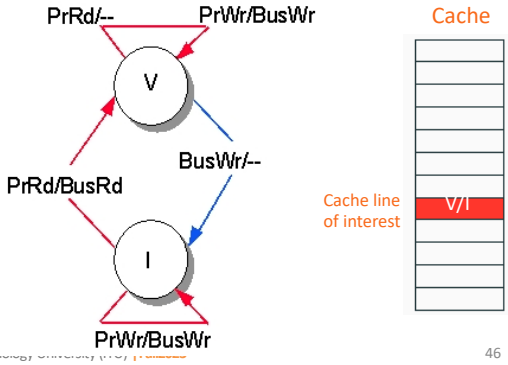
Enforcing Coherency –Snoopy CCP

- Write Invalidate CC Protocol
  - Analysis of Write-Through (WT) cache coherency using Snoopy Write Invalidate Protocol

States on cache line ...
V is valid
I is invalid

Transitions ...
Red is processor initiated
Blue is bus initiated

Labeling A/B ...
If A is observed
Then transaction B is generated



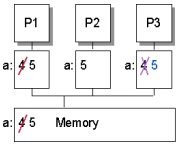
46

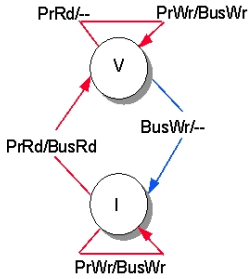
ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- Write Invalidate CC Protocol
  - Analysis of Write-Through (WT) cache coherency using Snoopy Write Invalidate Protocol

P1 reads a into cache
P3 reads a into cache
P1 writes a to 5, and writes through to main memory, correcting it
P3 sees WT, invalidates or updates
P2 reads a into cache





States on cache line ...
V is valid
I is invalid

Transitions ...
Red is processor initiated
Blue is bus initiated

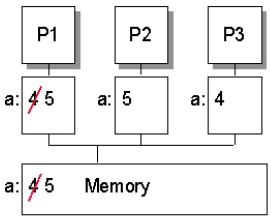
Labeling A/B ...
If A is observed
Then transaction B is generated

47

ACA: Multiprocessor Architecture

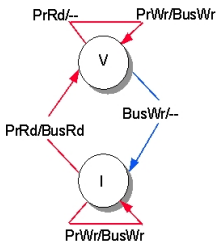
Enforcing Coherency –Snoopy CCP

- Write Invalidate CC Protocol
  - Analysis of Write-Through (WT) cache coherency using Snoopy Write Invalidate Protocol



P1 reads a into cache
P3 reads a into cache
P1 writes a to 5, and writes through to main memory, correcting it
P3 sees WT, invalidates
P2 reads a into cache

P1: I ---> V
PrRd/BusRd
P3: I ---> V
PrRd/BusRd
P1: V ---> V
PrWr/BusWr
P3: V ---> I
BusWr/--
P2: I ---> V
PrRd/BusRd



- Initial state of a Cache line (Variable) = INVALID

48

49

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- Problem with Write-Through caches
  - High bandwidth requirements
    - Every write from every processor goes to shared bus and memory
  - Write-Back absorbs cache writes as cache hits
    - Write hits won't go to the bus!
  - How to ensure write serialization?
    - More sophisticated coherence protocols are required

50

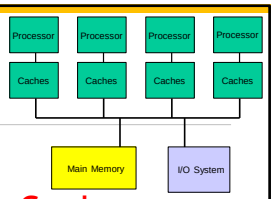
ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- Write Invalidate CCP for Write-Back Caches
  - In a write back scheme, main memory is not updated until 'dirty' cache line is replaced!
  - Relatively more complex to maintain coherence in write-back (as the dirty cache lines can still be local to caches)
  - A cache line can have more states possible than just VALID or INVALID!

51

ACA: Multiprocessor Architecture

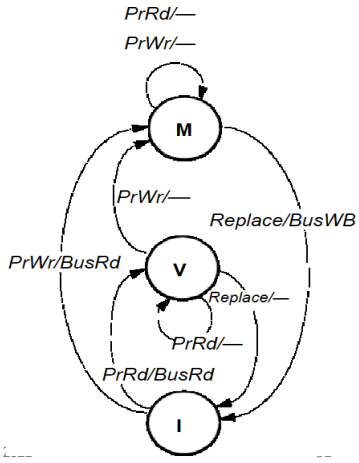


Enforcing Coherency –Snoopy CCP

- MSI Write-Invalidate for Write-Back Caches

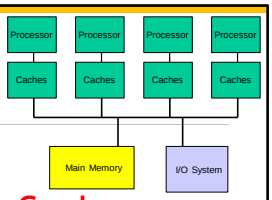
- States
  - Invalid (I), Valid/Shared (S), Modified (M)
- Processor / Cache Operations
  - PrRd, PrWr, Block Replace
- Bus Transactions
  - Bus Read (BusRd), Write-Back (BusWB)

- Treat Valid=Shared => clean
- Treat Modified=Exclusive => dirty



52

ACA: Multiprocessor Architecture



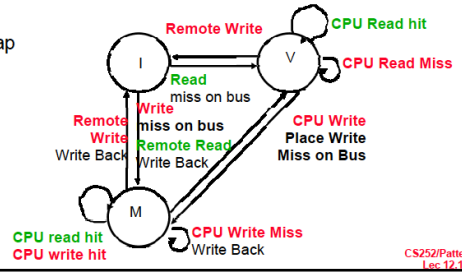
Enforcing Coherency –Snoopy CCP

- MSI Write-Invalidate for Write-Back Caches

Example

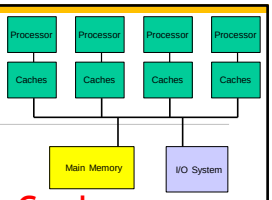
	Processor 1			Processor 2			Bus			Memory	
step	P1 State	Addr	Value	P2 State	Addr	Value	Action	Proc.	Addr	Value	Memory Addr Value
P1: Write 10 to A1											
P1: Read A1											
P2: Read A1											
P2: Write 20 to A1											
P2: Write 40 to A2											

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



53

ACA: Multiprocessor Architecture



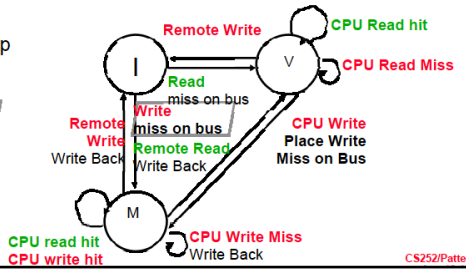
Enforcing Coherency –Snoopy CCP

- MSI Write-Invalidate for Write-Back Caches

Example

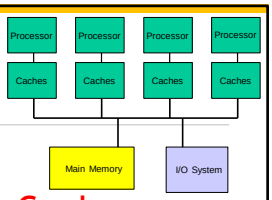
	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.
Active arrow =



54

ACA: Multiprocessor Architecture



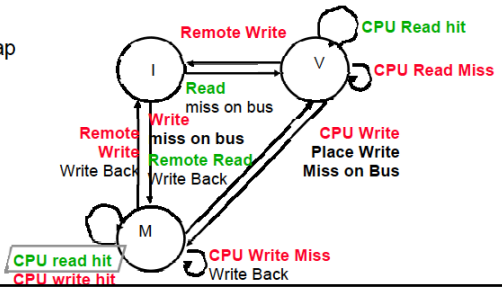
Enforcing Coherency –Snoopy CCP

- MSI Write-Invalidate for Write-Back Caches

Example

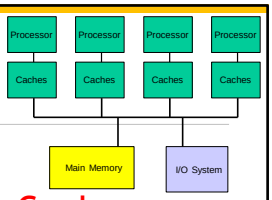
	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2



55

ACA: Multiprocessor Architecture



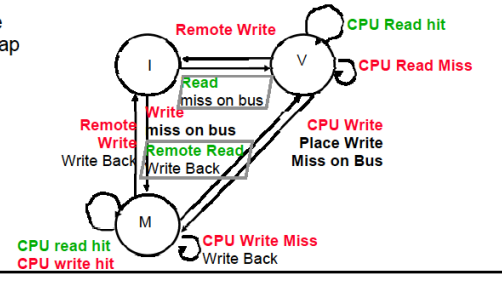
Enforcing Coherency –Snoopy CCP

- MSI Write-Invalidate for Write-Back Caches

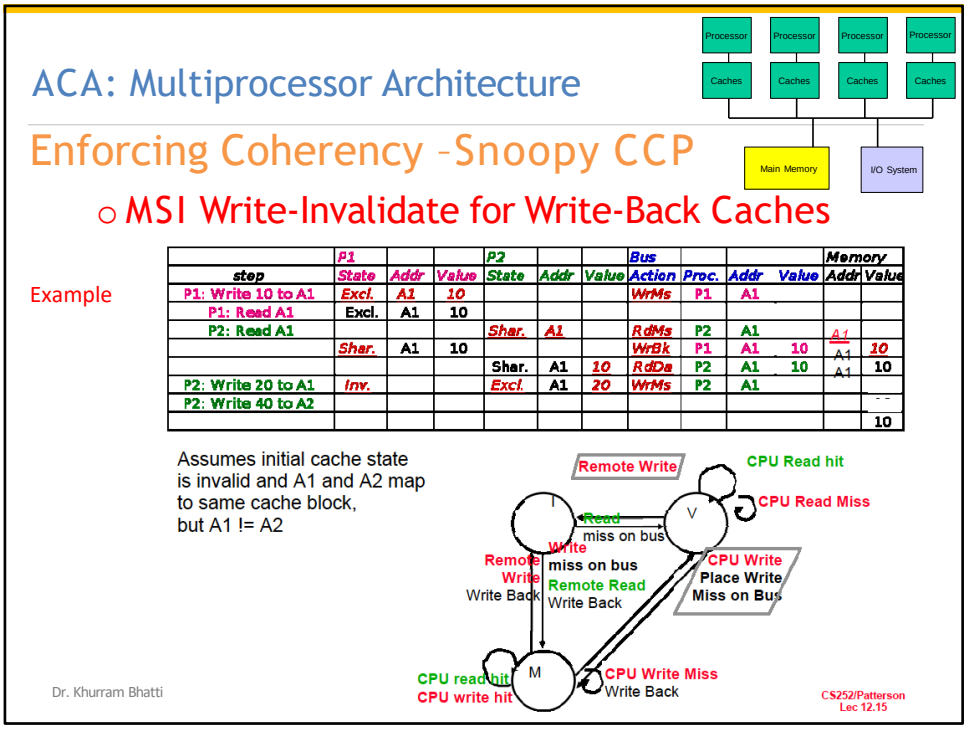
Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes initial cache state is invalid and A1 and A2 map to same cache block, but A1 != A2.

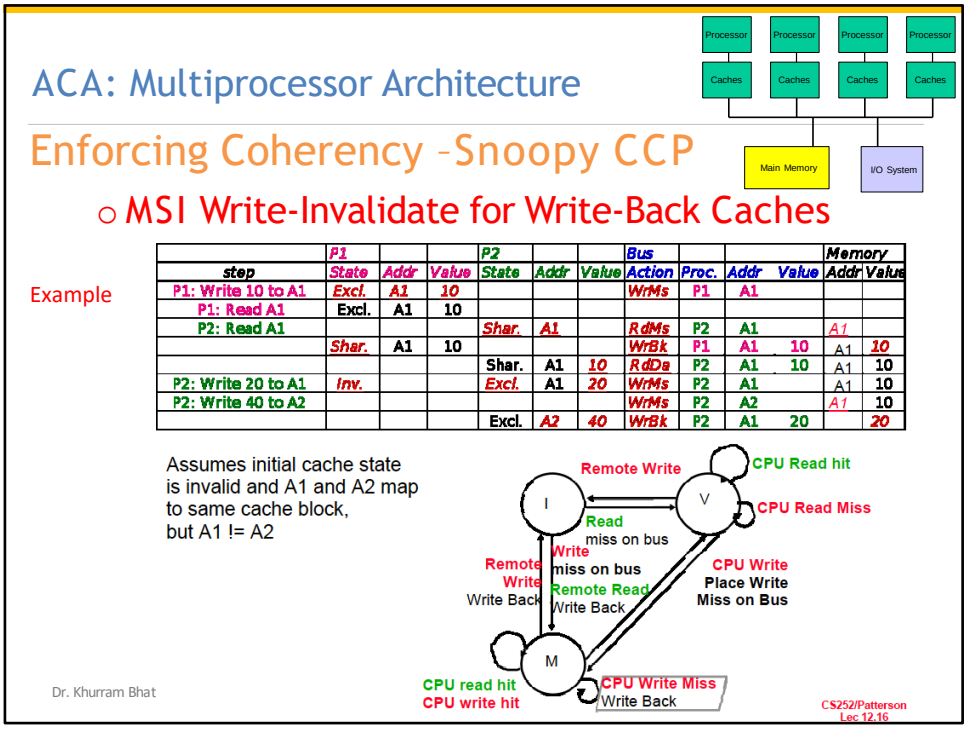


56



	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1		A1	
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1			
P2: Write 40 to A2												
												10

57

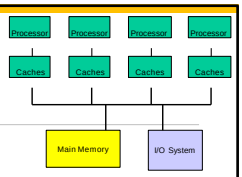


	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1		A1	
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2							WrMs	P2	A2		A1	10
				Excl.	A2	40	WrBk	P2	A1	20		20

58

59

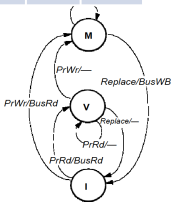
ACA: Multiprocessor Architecture



- MSI Exercise

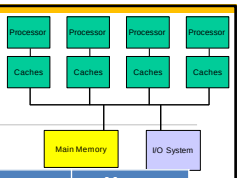
STEPS	P1			P2			Bus				Memory	
	State	Addr	Value	State	Addr	Value	Action	Proc	Addr	Value	Addr	Value
P1 reads X												
P2 reads Y												
P1 writes X												
P1 reads Z												
P1 writes Y												
P2 writes Z												

Variables X and Y map to the same cache block
But X !=Y
Variable Z maps to a different cache block
Initial State for all variables = INVALID



60

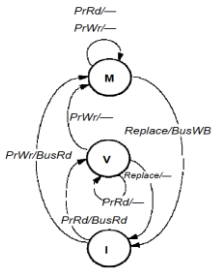
ACA: Multiprocessor Architecture



○ MSI Exercise

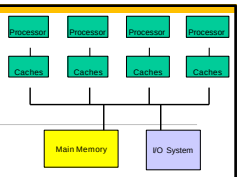
STEPS	P1			P2			Bus				Memory	
	State	Addr	Value	State	Addr	Value	Action	Proc	Addr	Value	Addr	Value
P1 reads X	S	X					RdMs	P1	X		X	
P2 reads Y				S	Y		RdMs	P2	Y		Y	
P1 writes X	M	X										
P1 reads Z	S	Z					RdMs	P1	Z			
P1 writes Y	S	Y					WrMs	P1	Y			
							WrBk	P1	X		X	
	M	Y		Inv	Y		Inv	P1	Y			
P2 writes Z	I	Z		M	Z		WrMs	P2	Z			

Variables X and Y map to the same cache block
X !=Y
Variable Z maps to a different cache block
Initial State = INVALID



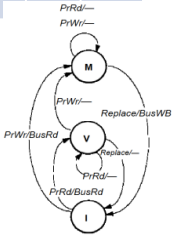
61

ACA: Multiprocessor Architecture



○ MSI Exercise

STEPS	P1			P2			Bus				Memory	
	State	Addr	Value	State	Addr	Value	Action	Proc	Addr	Value	Addr	Value
P2 writes 5 to A				M	A	5	BsRd	P2	A	-	A	-
P2 reads C				V	C	C	BsRd	P2	C	-	C/A	-/5
P1 reads B	V	B	-				BsRd	P1	B	-	B	-
P1 writes 10 to B	M	B	10				Inv	P1	B	-	B	-
P2 reads B	V	B	10	V	B	10	BsRd	P2	B	10	B	10
P1 reads C	V	C	-				BsRd	P1	C	-	C	-
P1 reads A	V	A	5				BsRd	P1	A	5	A	5



62

63

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
  - MESI stands for: Modified/Exclusive/Shared/Invalid
  - Each Cache line can be in any of the FOUR states
  - Cache line changes state as a function of memory access events
  - Memory access event may be either
    - Due to local processor activity (i.e. cache access)
    - Due to bus activity - as a result of snooping

64

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- STATES
  - Modified:
    - The cache line is *present only* in the current cache, and is *dirty*, i.e., It has been modified from the value in main memory
    - The cache is required to write the data back to main memory at some time in the future, before permitting any other read of the (no longer valid) main memory state
    - The write-back changes the line to the Shared state

65

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- STATES
  - Exclusive:
    - The cache line is present only in the current cache, but it is clean
    - It matches main memory
    - It may be changed to the Shared state at any time, in response to a read request
    - Alternatively, it may be changed to the Modified state in response to write

66

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- STATES
  - Shared:
    - Indicates that this cache line may be stored in other caches as well and it is clean
    - It matches the main memory
    - The line may be discarded (changed to the Invalid state) at any time.

67

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- STATES
  - Invalid:
    - Indicates that this cache line is invalid.

68

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
  - ALLOWED STATES for a cache line in any TWO caches

Modified
Exclusive
Shared
Invalid

	M	E	S	I
M	✗	✗	✗	✓
E	✗	✗	✗	✓
S	✗	✗	✓	✓
I	✓	✓	✓	✓

How can a cache line be Shared & Invalid at the same time in 2 caches?

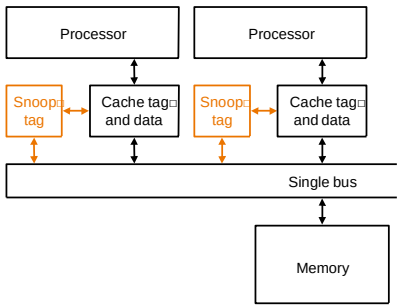
69

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
  - Local Processor Activity on the cache lines can be defined by following operations

- Read Hit
- Read Miss
- Write Hit
- Write Miss



70

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local READ HIT
  - Cache line must be in one of M E S states
  - This must be correct local value (if M, it must have been modified locally)
  - Simply return value to processor from cache
  - No state change takes place for cache line

71

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local READ MISS (4-cases)
  - Case1: No other copy in snooping caches
    - Processor makes bus request to memory
    - Value is read to local cache & marked as E
  - Case 2: At least ONE cache has EXCLUSIVE copy
    - Processor makes bus request to memory
    - Snooping cache puts copy value on the bus
    - Memory access is abandoned
    - Local processor caches value in its local cache
    - Both lines (provider & recipient) are set to SHARED

72

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local READ MISS (4-cases)
  - Case 3: Several caches have S copy
    - Processor makes bus request to memory
    - One cache puts copy value on the bus (arbitrated)
    - Memory access is abandoned
    - Local processor caches value in its local cache
    - Local copy set to SHARED
    - Other copies remain SHARED

73

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local READ MISS (4-cases)
  - Case 4: One cache has M copy
    - Processor makes bus request to memory
    - Snooping cache puts copy value on the bus
    - Memory access is abandoned
    - Local processor caches value in local cache
    - Local copy is tagged as S
    - Source copy (which was M) is written back to memory
    - Source copy change state: M -> S

74

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local WRITE HIT (1)
  - Cache Line must be in one of M E S states
  - If in M state
    - Line is the only copy and already 'dirty'
    - Update local cache value
    - No state change
  - If in E state
    - Update local cache value
    - State changes: E->M

75

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local WRITE HIT (2)
  - Line must be in one of M E S states
  - If in S state
    - Processor broadcasts an invalidate on bus
    - Snooping processors with S copy change S->I
    - Local cache value is updated
    - Local state change S->M

76

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local WRITE MISS (1)
  - Action depends on copies in other processors
  - If no other copies available
    - Value read from memory to local cache
    - Value updated
    - Local copy state set to M

77

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local WRITE MISS (2)
  - Action depends on copies in other processors
  - Other copies, either one in state E or more in state S
    - Value read from memory to local cache
    - Bus transaction marked RWITM (read with intent to modify)
    - Snooping processors see this and set their copy state to I
    - Local copy updated & state set to M

78

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- Local WRITE MISS (3)
  - Action depends on copies in other processors
  - Another copy in state M
    - Processor issues bus transaction marked RWITM
    - Snooping processor sees this transaction and blocks RWITM request on bus
    - Takes control of bus
    - Writes back its copy to memory
    - Sets its copy state to I
    - Original local processor re-issues RWITM request
    - Value is read from memory to local cache
    - Local copy value updated
    - Local copy state set to M

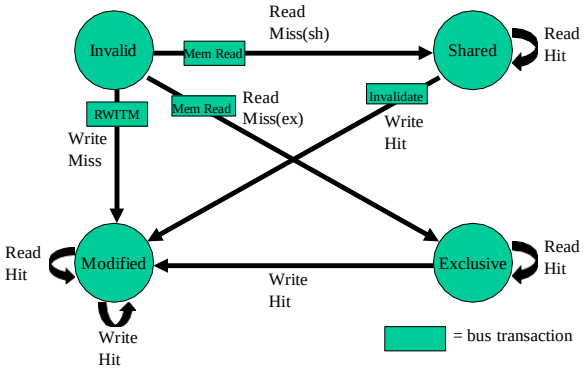
79

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- OR simply try to understand State Machine!

Locally (Processor) initiated accesses



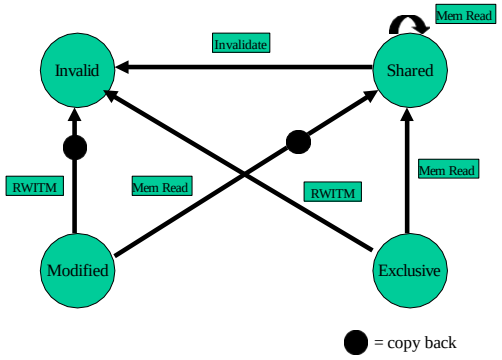
80

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
- OR, try to understand State Machine!

Remotely (Bus) initiated accesses



81

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol
  - ALLOWED STATES for a cache line in any TWO caches

Modified
Exclusive
Shared
Invalid

	M	E	S	I
M	✗	✗	✗	✓
E	✗	✗	✗	✓
S	✗	✗	✓	✓
I	✓	✓	✓	✓

- Shared state may be imprecise
- If another cache discards a Shared line, this cache may become the sole owner of that cache line, but it will not be promoted to Exclusive state
- Other caches do not broadcast notices when they discard shared (clean) cache lines
- A cache can not promote a line from S to E without maintaining a count of the number of shared copies

82

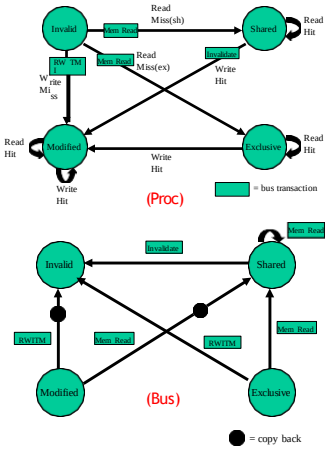
ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

○ MESI Cache Coherence Protocol: Example

B1!=B2, both map to L of local caches

#	Event	P1's Cache content	P2's Cache content
		L = unknown (Invalid)	L = unknown (Invalid)
1	P1 writes 10 to B1		
2	P1 reads B1		
3	P2 reads B1		
4	P2 writes 20 to B1		
5	P2 writes 40 to B2		
6	P1 reads B1		
7	P1 writes 30 to B1		
8	P2 writes 50 to B1		
9	P1 reads B1		
10	P2 reads B2		
11	P1 writes 60 to B2		



1

83

83

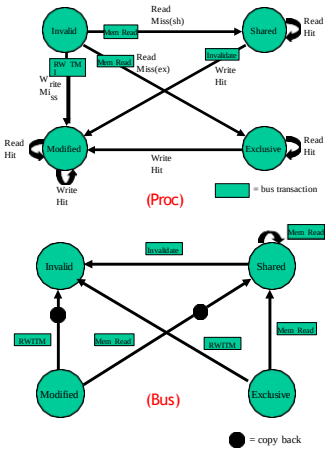
ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

○ MESI Cache Coherence Protocol: Example

B1!=B2, both map to L of local caches

#	Event	P1's Cache content	P2's Cache content
		L = unknown (Invalid)	L = unknown (Invalid)
1	P1 writes 10 to B1 (write miss)	B1=10(M)	
2	P1 reads B1 (read hit)	B1=10(M)	
3	P2 reads B1 (read miss)	B1=10 (S)	B1=10(S)
4	P2 writes 20 to B1 (write hit)	B1=10(I)	B1=20(M)
5	P2 writes 40 to B2 (write miss)		B2=40(M)
6	P1 reads B1 (read miss)	B1=20(E)	B2=40(M)
7	P1 writes 30 to B1 (write hit)	B1=30(M)	B2=40(M)
8	P2 writes 50 to B1 (write miss)		B1=50(M)
9	P1 reads B1 (read miss)		
10	P2 reads B2 (read miss)		
11	P1 writes 60 to B2 (write miss)		



1

84

84

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- MESI Cache Coherence Protocol: Example
- TO DO: State Machine representation of all transitions

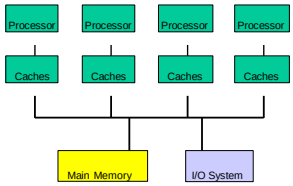
85

ACA: Multiprocessor Architecture

- MESI Exercise

STEPS	P1			P2			Bus				Memory	
	State	Addr	Value	State	Addr	Value	Action	Proc	Addr	Value	Addr	Value
P2 writes 5 to A												
P2 reads C												
P1 reads B												
P1 writes 10 to B												
P2 reads B												
P1 reads C												
P1 writes 15 to A												
P2 writes 2 to A												

Variables A, B and C map to the same cache block
But A!=B!=C
Initial State for all variables = INVALID



86

ACA: Multiprocessor Architecture

Enforcing Coherency –Snoopy CCP

- Summary of Snoopy Cache Coherence
  - The number of states of CCPs represents the capability to classify cache lines by their degree of sharing
  - In general, the more states a CCP employs, the more categories it can separate the cache events with more possible transitions –better resolution
  - The more sharing that can be supported, the less bus activity will be required to maintain coherency –better bandwidth
  - At the cost of snoopy hardware complexity

87

88