NC STATE UNIVERSITY

Invalidate and Update Protocols

Lecture 14
(Chapter 7, cont.)

E. F. Gehringer,
based on slides by Yan Solihin

1

Lecture 15 Outline

- **MSI protocol**
  - **State diagram**
  - **Animations**
- **MESI protocol**
- **Dragon protocol**
- **Firefly protocol**

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

2

Basic MSI Writeback Invalidation Protocol

- States
  - Invalid (I)
  - Shared (S): one or more copies, and memory copy is up-to-date
  - Dirty or Modified (M): only one copy
- Processor Events:
  - PrRd (read), PrWr (write)
- Bus Transactions
  - BusRd: asks for copy with no intent to modify
  - BusRdX: asks for copy with intent to modify (instead of BusWr)
  - Flush: updates memory
- Actions
  - Update state, perform bus transaction, flush value onto bus

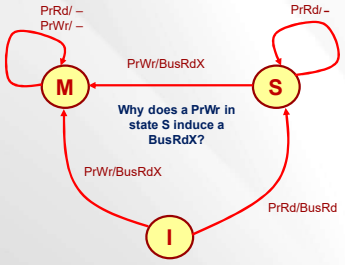
3

State-Transition Diagrams

- On the following slides, we will display the state-transition diagrams
  - for processor-initiated transactions
  - for bus-initiated transactions
- We will see transitions of the following form:
  - Invalidation: ⟨Any⟩ → I
  - Intervention: {Exclusive, Modified} → Shared

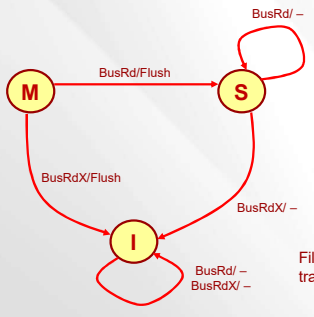
4

MSI: Processor-Initiated Transactions



5

MSI: Bus-Initiated Transactions

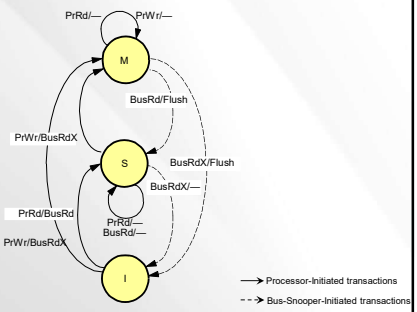


Thus, valid data must be supplied by memory

Fill in the last two transitions here.

6

MSI State Transition Diagram



7

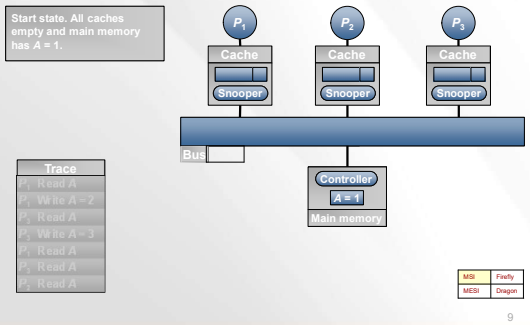
Lecture 15 Outline

- **MSI protocol**
  - **State diagram**
  - **Animations**
- **MESI protocol**
- **Dragon protocol**
- **Firefly protocol**

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

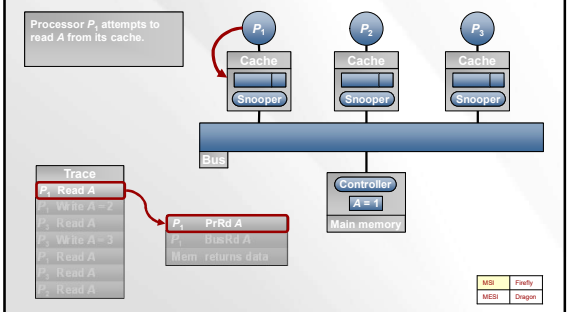
8

MSI Visualization – Start State



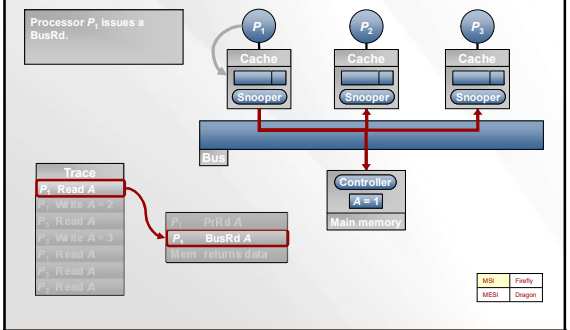
9

MSI: Processor P_1 Reads A



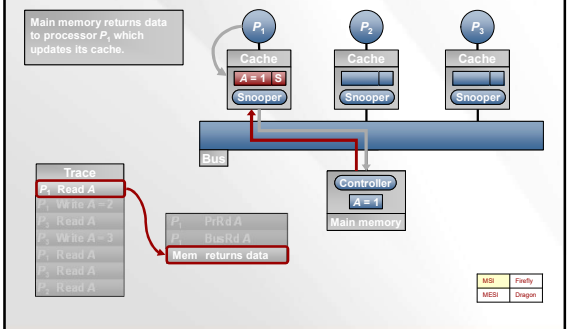
10

MSI: Processor P_1 Reads A

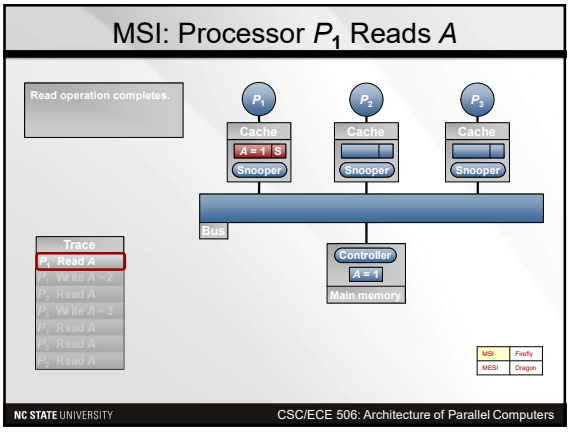


11

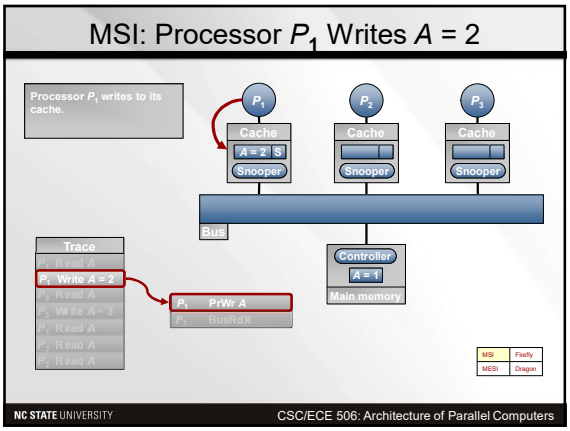
MSI: Processor P_1 Reads A



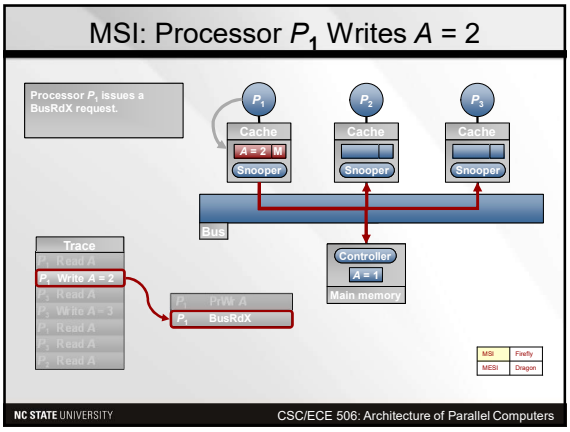
12



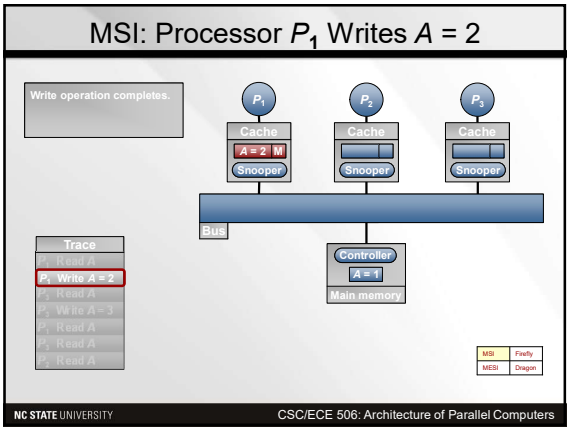
13



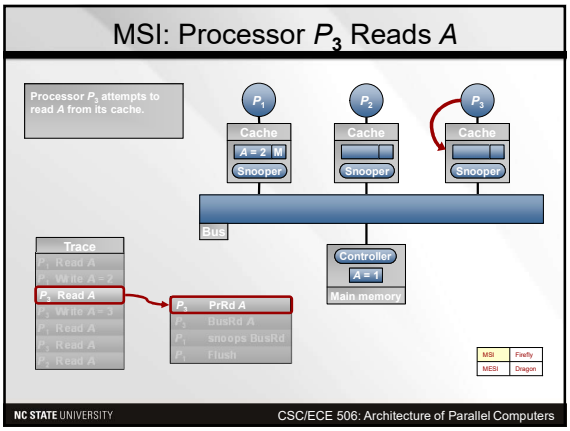
14



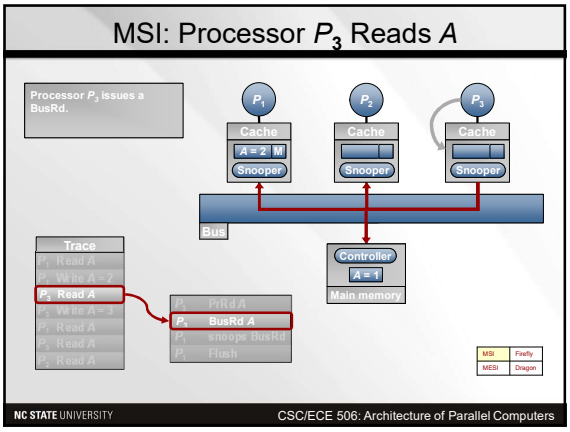
15



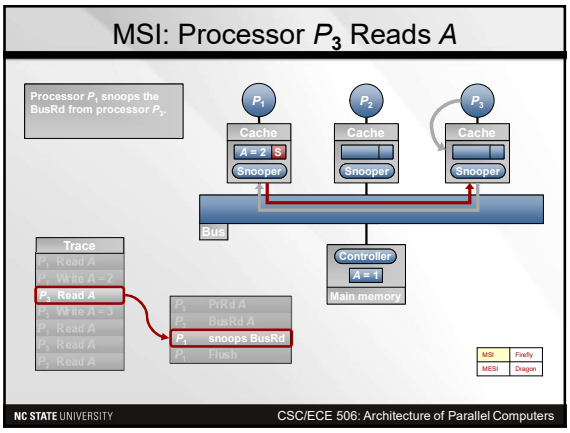
16



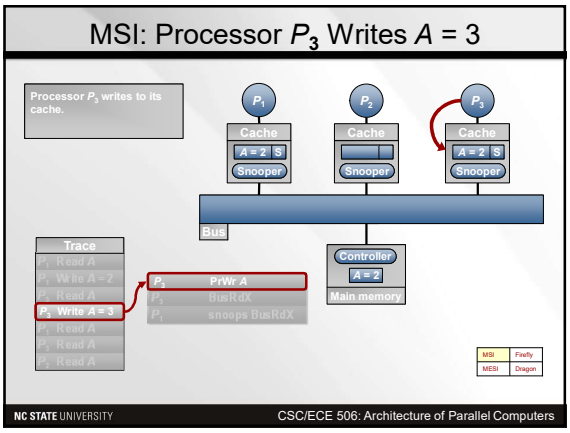
17



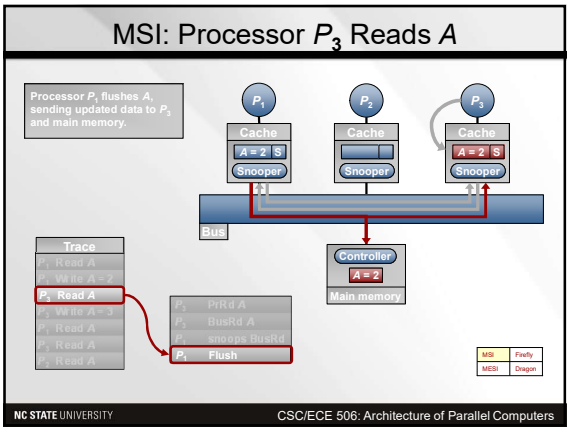
18



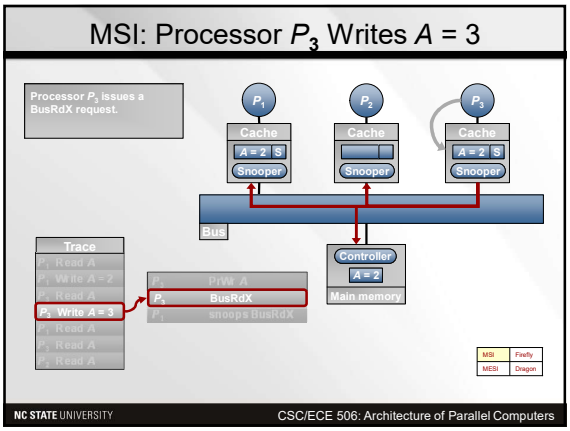
19



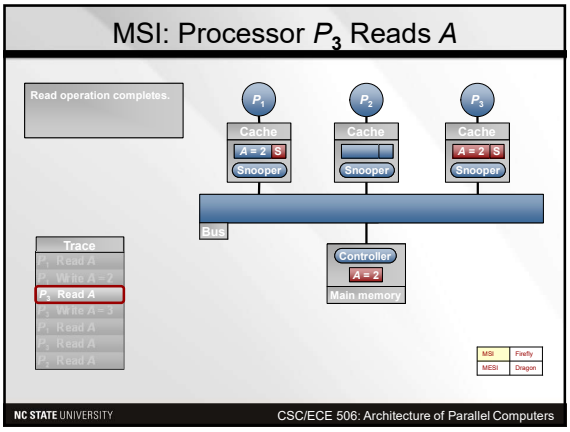
22



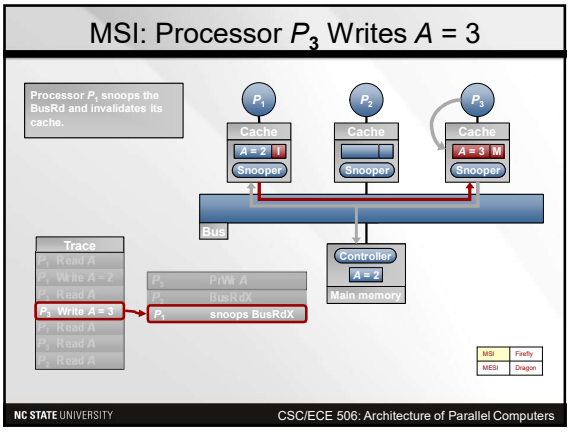
20



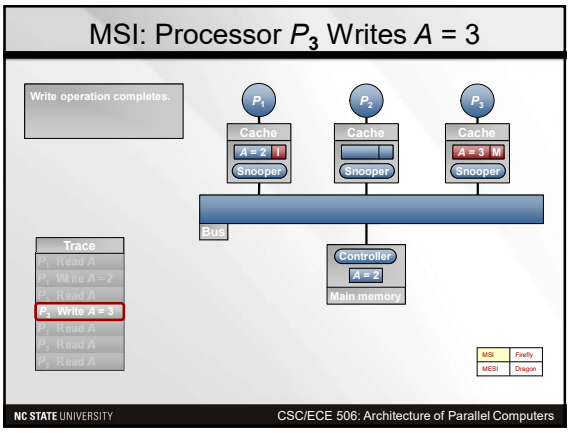
23



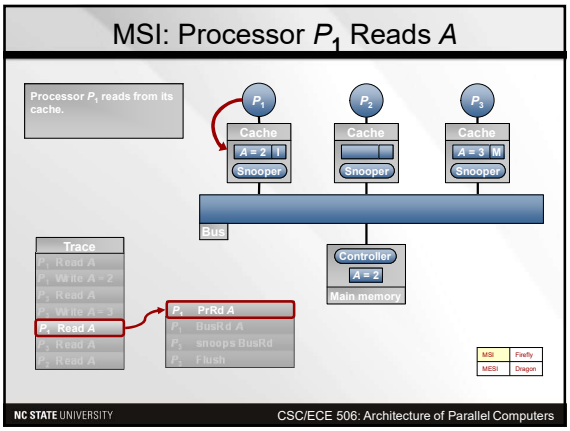
21



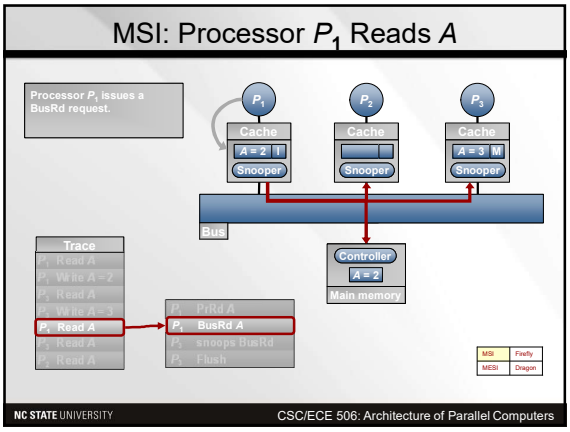
24



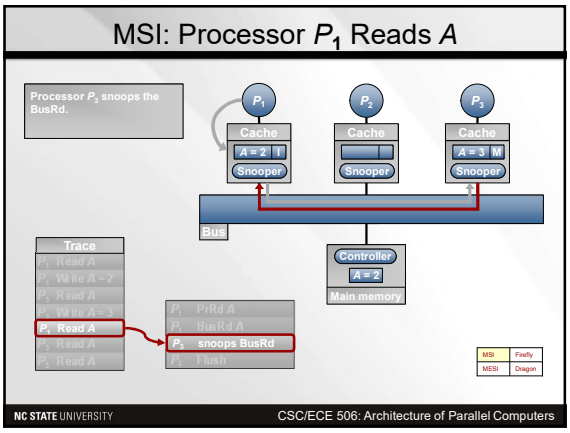
25



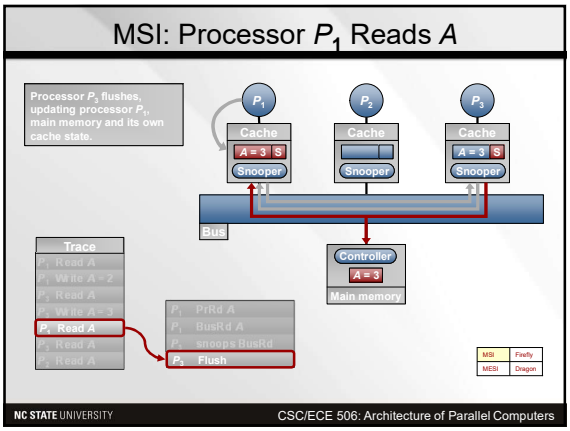
26



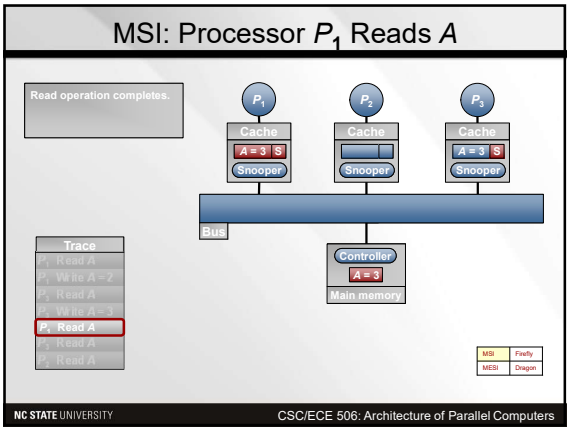
27



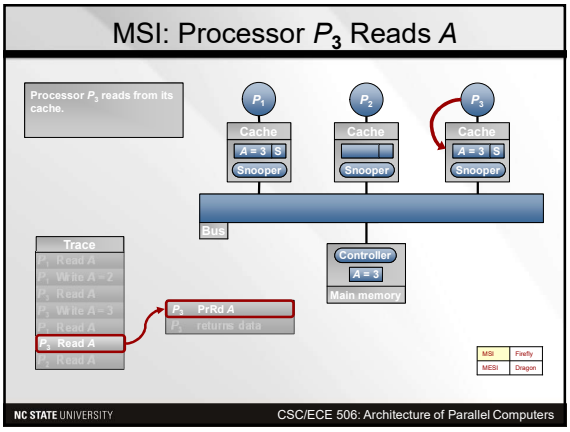
28



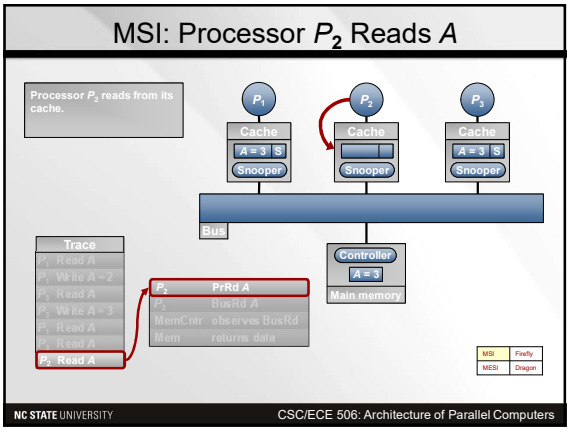
29



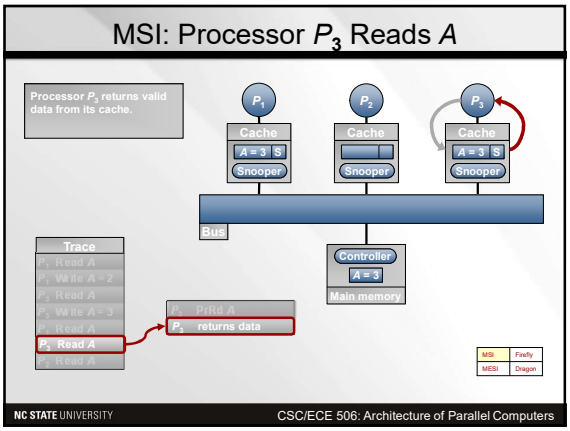
30



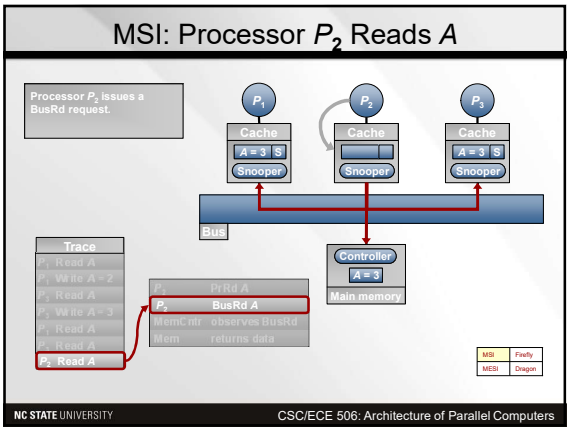
31



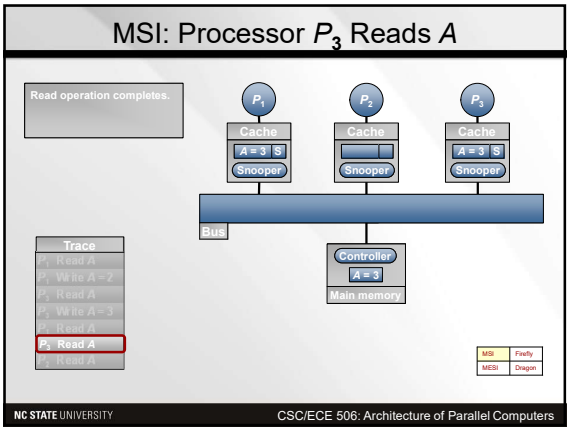
34



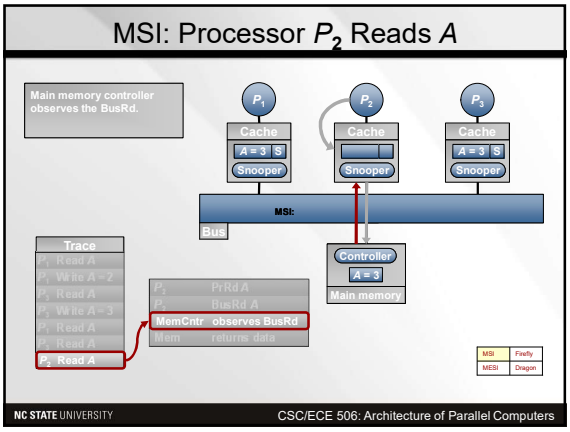
32



35

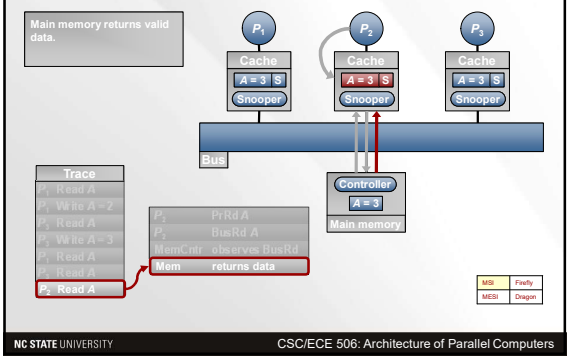


33



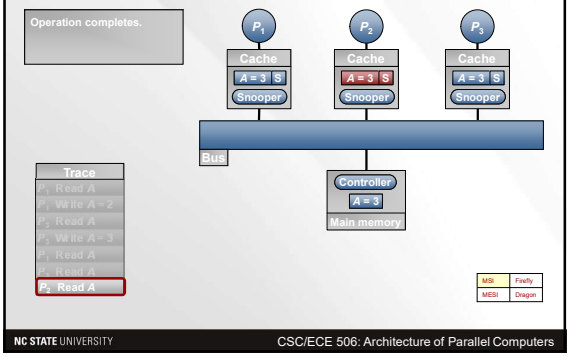
36

MSI: Processor P_2 Reads A



37

MSI: Processor P_2 Reads A



38

MSI Example: Rd/Wr to a single line

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	S	–	–	BusRd	Mem
W1	M	–	–	BusRdX*	Mem
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX*	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own Cache
R2	S	S	S	BusRd	Mem

*or, BusUpgr (data from own cache)

39

Notes on MSI Protocol

- For M → I, BusRdX/Flush: why flush?

40

Notes on MSI Protocol

- For M → I, BusRdX/Flush: why flush? Because it is a read with intention to write, as opposed to write.
  - Thus, there is a possibility for a read before the write is performed.
  - In addition, the write could be to a different word in the line (so the whole line needs to be flushed).

41

Notes on MSI Protocol

- For M → I, BusRdX/Flush: why flush? Because it is a read with intention to write, as opposed to write.
  - Thus, there is a possibility for a read before the write is performed.
  - In addition, the write could be to a different word in the line (so the whole line needs to be flushed).
- In case of a write to a shared block:
  - Cache already has latest data; can use upgrade (BusUpgr) instead of BusRdX
- Replacement changes state of two blocks: outgoing and incoming
- Flush has to modify both caches and main memory

Note: Coherence granularity is *u* (a single line). What happens when all the reads go to word 0 on line *u*, but write by P3 goes to word 1 on line *u*? False-sharing miss on the 2nd R1

42

MSI: Coherence and SC

- Coherence:
  - Write propagation:
    - through invalidation, and flush on subsequent BusRds
  - Write serialization?
    - Writes (BusRdX) that go to the bus appear in bus order (and handled by snoopers in bus order!)
    - Writes that do not go to the bus?
      - Only happen when the line state is M, i.e. when I am the only processor holding the line. Local writes are only visible to me, so they are serialized.
- To enforce SC:
  - Program order: enforced by following the bus transaction order
    - All writes appear on the bus
    - All local writes (within 1 processor) can follow program order
  - Write completion: Occurs when write appears on bus
  - Write atomicity: A read returns the latest value of a write. At that time, the value is visible to all others (on a bus transaction, or on a local write).

Lecture 15 Outline

- MSI protocol
- MESI protocol
- Dragon protocol
- Firefly protocol

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

Lower-Level Protocol Choice

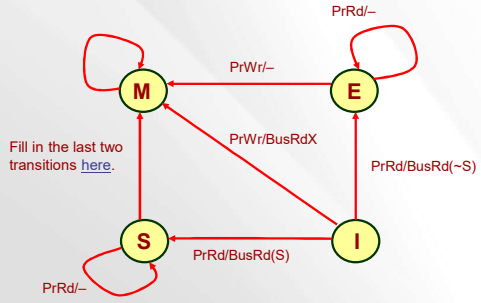
- What transition should occur when a BusRd is observed in state M?
  - Should the state change to S or to I?

MESI (4-state) Invalidation Protocol

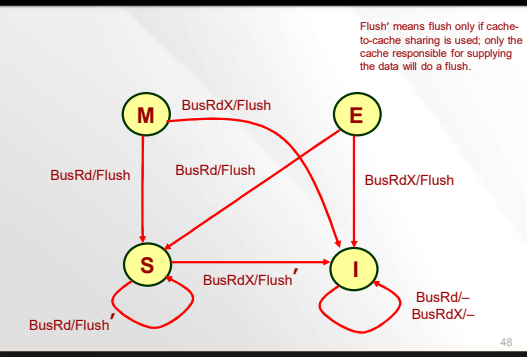
- Here's a problem with the MSI protocol:
  - A {Rd, Wr} sequence causes two bus transactions
    - BusRd (I → S) followed by BusRdX or BusUpgr (S → M)
    - even when no one is sharing (e.g., serial program!)
    - *In general, coherence traffic from serial programs is unacceptable*
- To avoid this, add a fourth state, Exclusive:
  - Invalid
  - Modified (dirty)
  - Shared (two or more caches may have copies)
  - Exclusive (only this cache has clean copy, same value as in memory)

- How does the protocol decide whether I → E or I → S?
  - Need to check whether someone else has a copy
  - "Shared" signal on bus: wired-or line asserted in response to BusRd

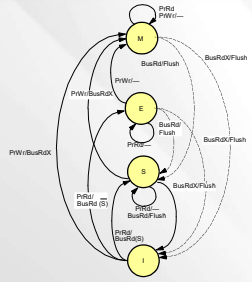
MESI: Processor-Initiated Transactions



MESI: Bus-Initiated Transactions



MESI State Transition Diagram

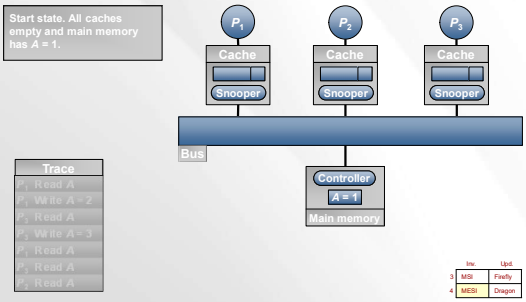


- BusRd(S) means shared line asserted on BusRd transaction

49

MESI Visualization

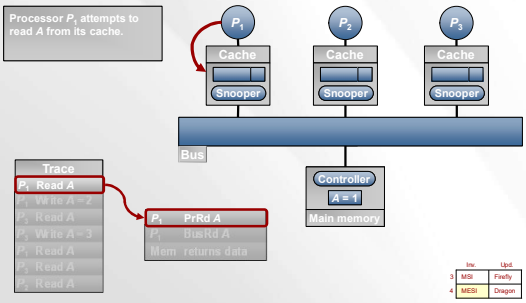
Start state. All caches empty and main memory has $A = 1$.



50

Processor P_1 Reads A

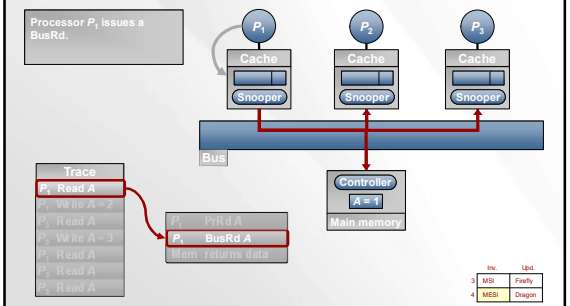
Processor P_1 attempts to read A from its cache.



51

Processor P_1 Reads A

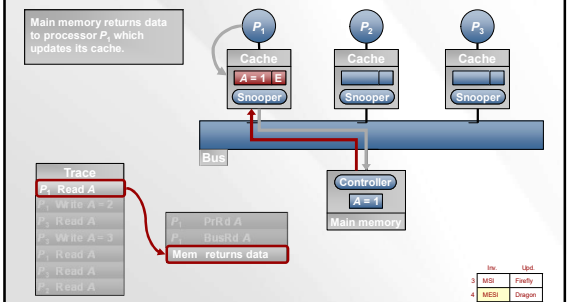
Processor P_1 issues a BusRd.



52

Processor P_1 Reads A

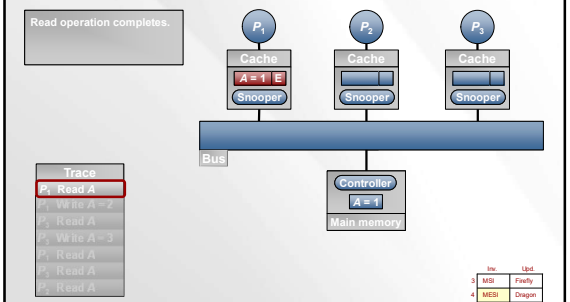
Main memory returns data to processor P_1 which updates its cache.



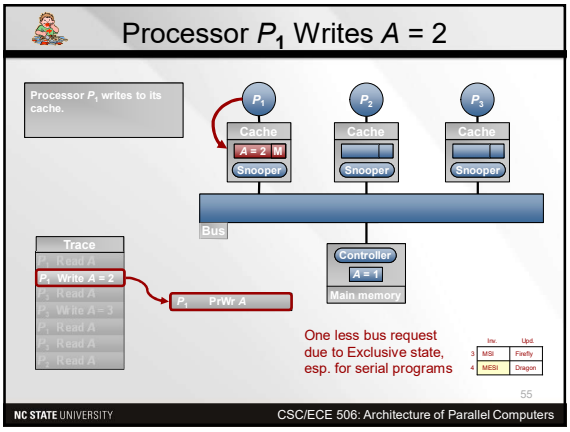
53

Processor P_1 Reads A

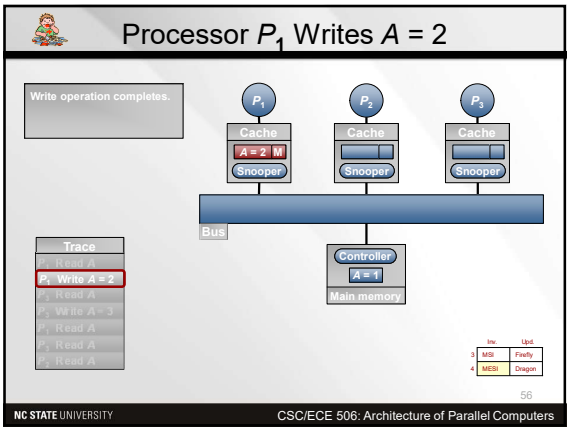
Read operation completes.



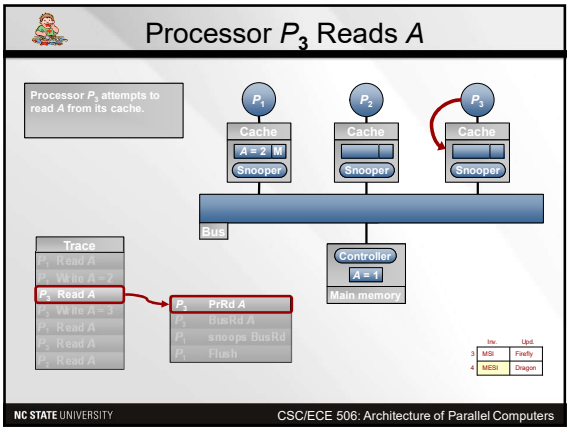
54



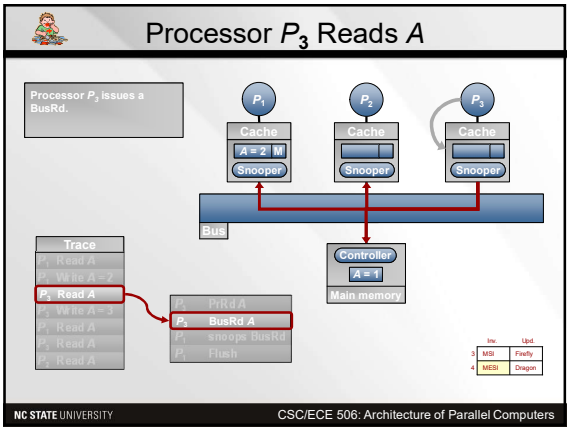
55



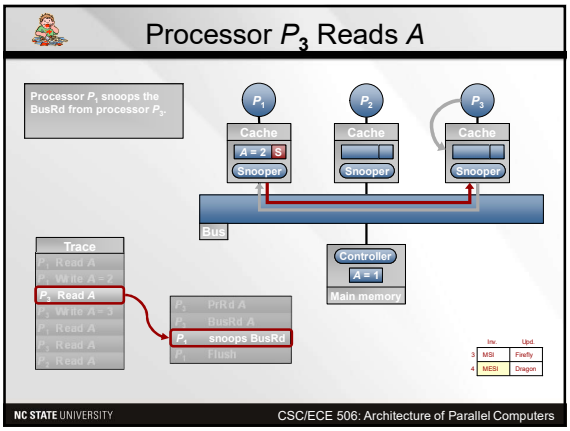
56



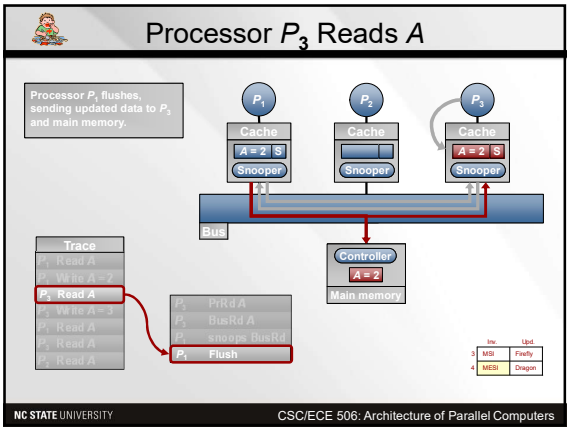
57



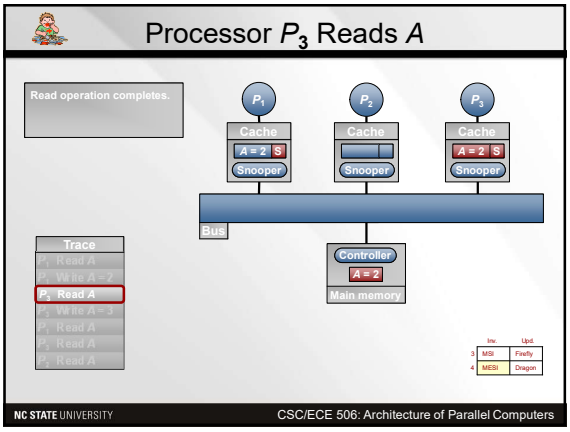
58



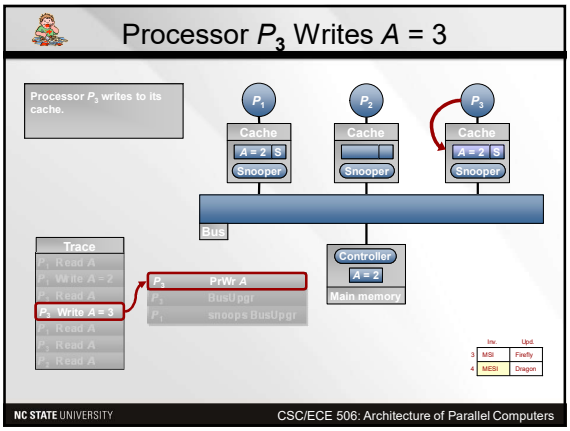
59



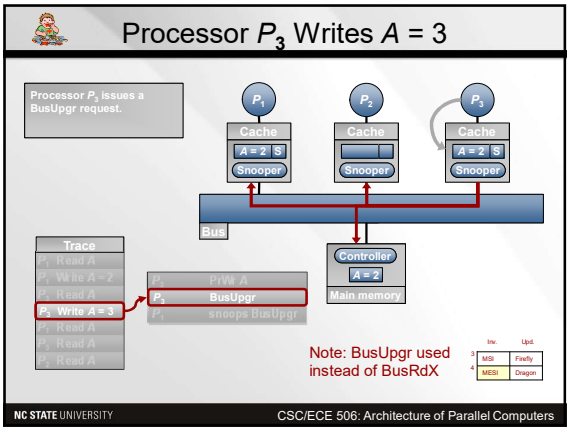
60



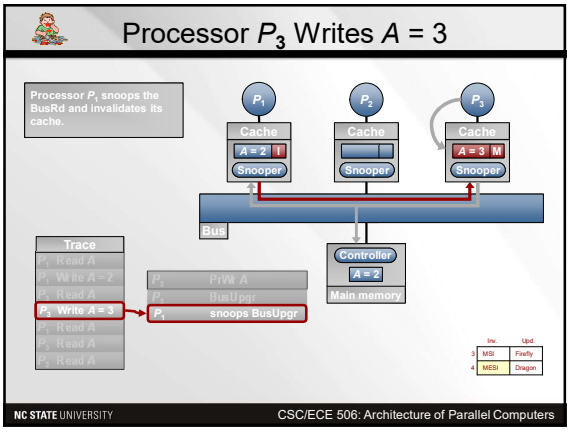
61



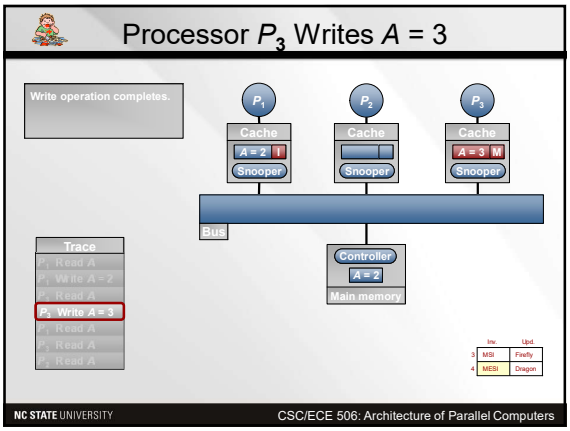
62



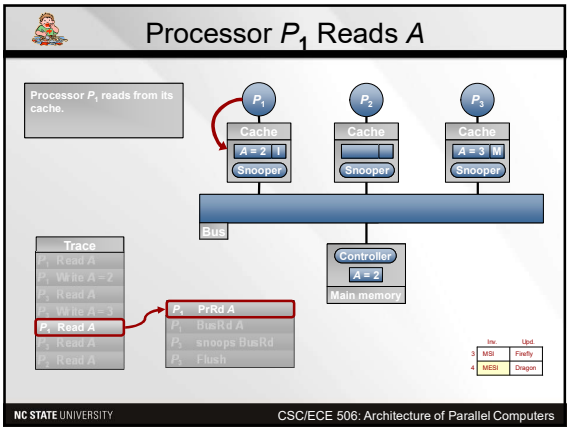
63



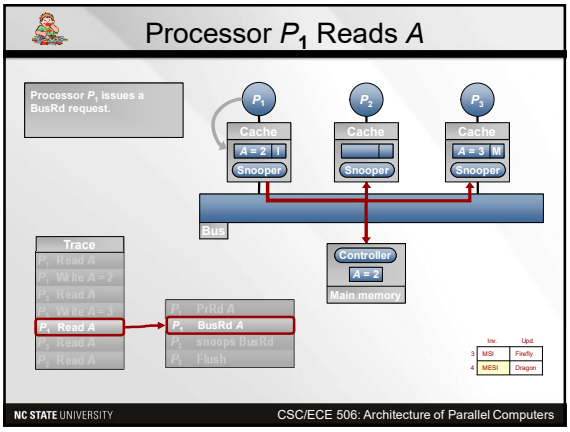
64



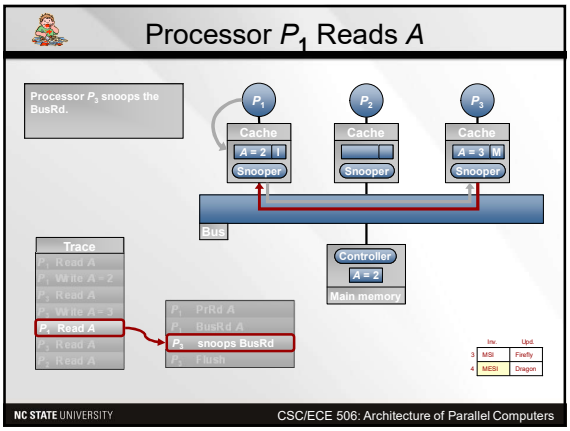
65



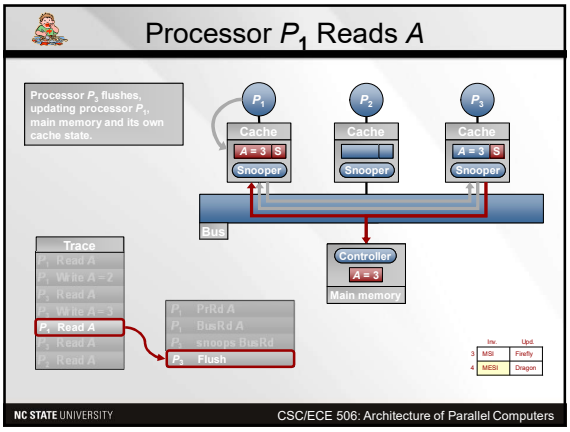
66



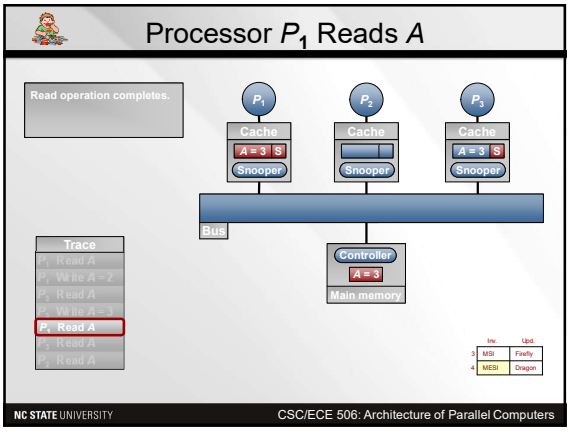
67



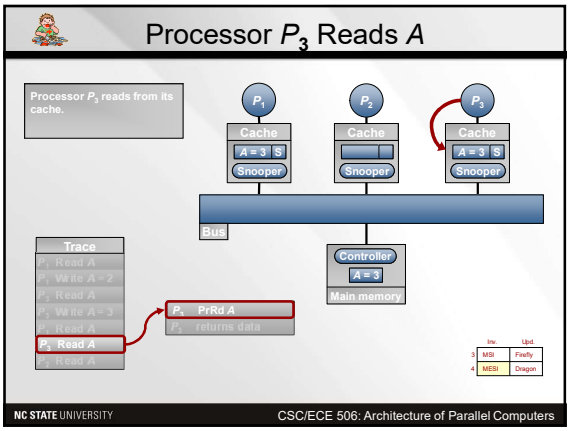
68



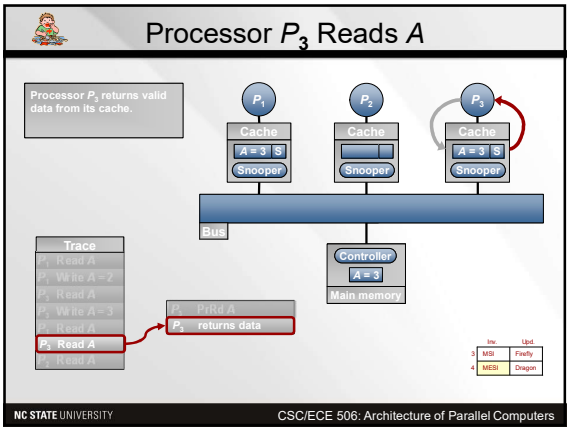
69



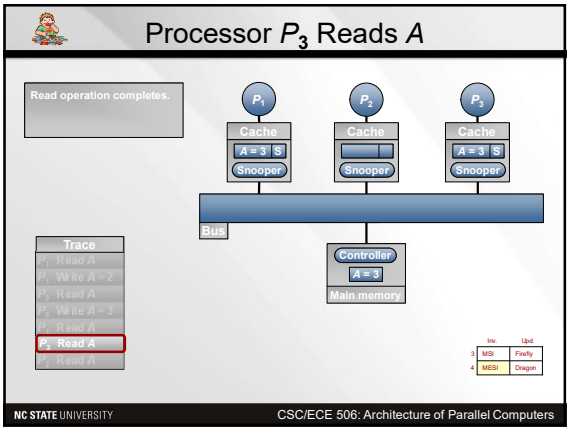
70



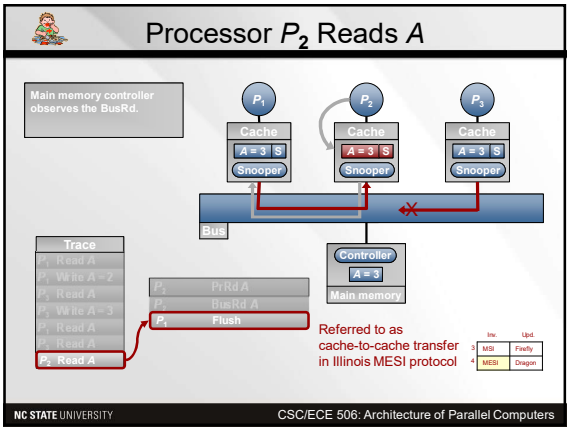
71



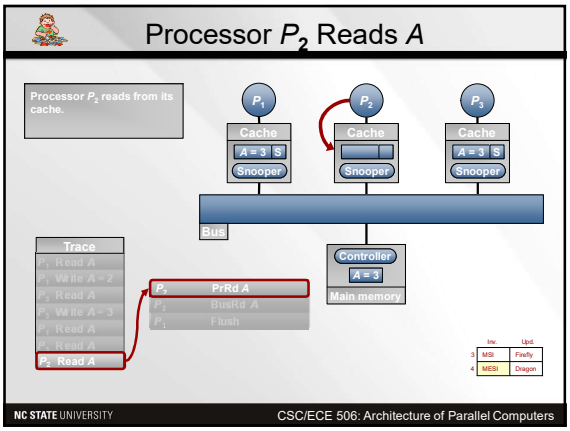
72



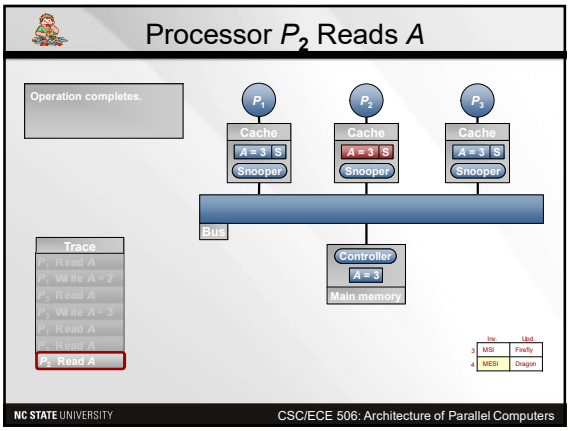
73



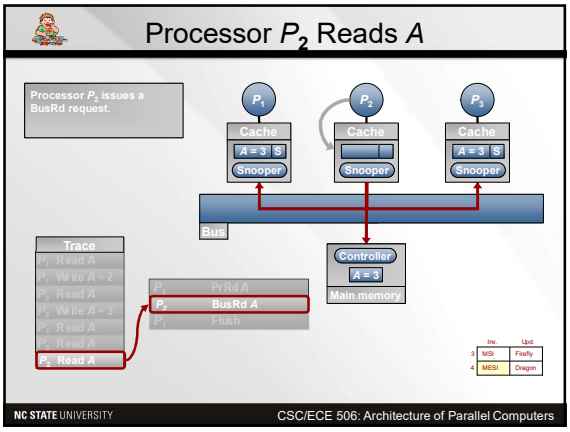
76



74



77



75

MESI Example (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

78

NC STATE UNIVERSITY CSC/ECE 506: Architecture of Parallel Computers

78

Change from MSI (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

79

79

Change from MSI (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

80

80

Change from MSI (Cache-to-Cache Transfer)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	S	–	S	BusRd/Flush	P1 cache
W3	I	–	M	BusRdX	Mem
R1	S	–	S	BusRd/Flush	P3 cache
R3	S	–	S	–	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

81

81

MESI Example (Cache-to-Cache Transfer+BusUpgr)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	-	-	BusRd	Mem
W1	M	-	-	-	Own cache
R3	S	-	S	BusRd/Flush	P1 cache
W3	I	-	M	BusUpgr	Own cache
R1	S	-	S	BusRd/Flush	P3 cache
R3	S	-	S	-	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

82

82

MESI Example (Cache-to-Cache Transfer+BusUpgr)

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	E	-	-	BusRd	Mem
W1	M	-	-	-	Own cache
R3	S	-	S	BusRd/Flush	P1 cache
W3	I	-	M	BusUpgr	Own cache
R1	S	-	S	BusRd/Flush	P3 cache
R3	S	-	S	-	Own cache
R2	S	S	S	BusRd/Flush′	P1/P3 Cache*

* Data from memory if no cache-to-cache transfer, BusRd/ –

83

83

Lower-Level Protocol Choices

- Who supplies data on miss when not in M state: memory or cache?
- Original, *Illinois* MESI: cache
  - assumes cache is faster than memory (*cache-to-cache transfer*)
  - Not necessarily true
- Adds complexity
  - How does memory know it should supply data? (must wait for caches)
  - A selection algorithm is needed if multiple caches have valid data.
- Useful in a distributed-memory system
  - May be cheaper to obtain from nearby cache than distant memory
  - Especially when constructed out of SMP nodes (Stanford DASH)

84

84

Lecture 15 Outline

- MSI protocol
- MESI protocol
- Dragon protocol
- Firefly protocol

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

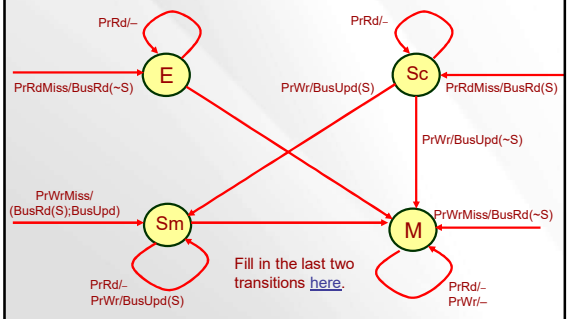
85

Dragon Writeback Update Protocol

- Four states
  - **Exclusive-clean** (E): Memory and I have it
  - **Shared clean** (Sc): I, others, and maybe memory, but I'm not owner
  - **Shared modified** (Sm): I and others but not memory, and I'm the **owner**
    - Sm and Sc can coexist in different caches, with at most one Sm
  - **Modified** or dirty (M): I and, no one else
  - On replacement: Sc can silently drop, Sm has to flush
- No invalid state
  - If in cache, cannot be invalid
  - If not present in cache, can view as being in not-present or invalid state
- New processor events: PrRdMiss, PrWrMiss
  - Introduced to specify actions when block not present in cache
- New bus transaction: BusUpd
  - Broadcasts single word written on bus; updates other relevant caches

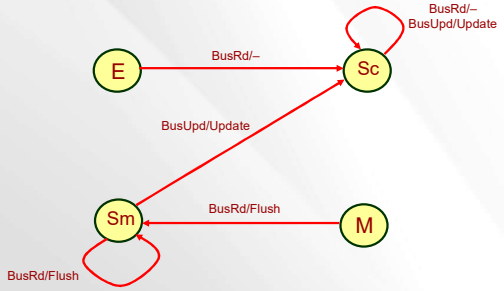
86

Dragon: Processor-Initiated Transactions



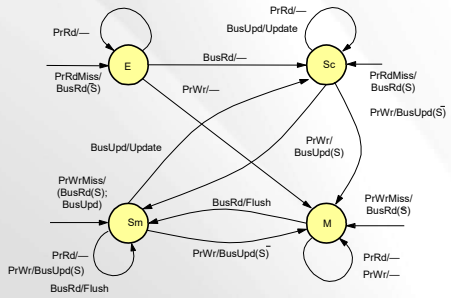
87

Dragon: Bus-Initiated Transactions



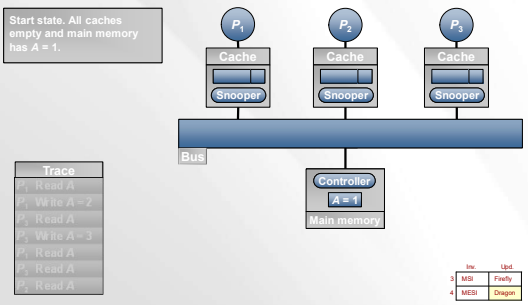
88

Dragon State Transition Diagram

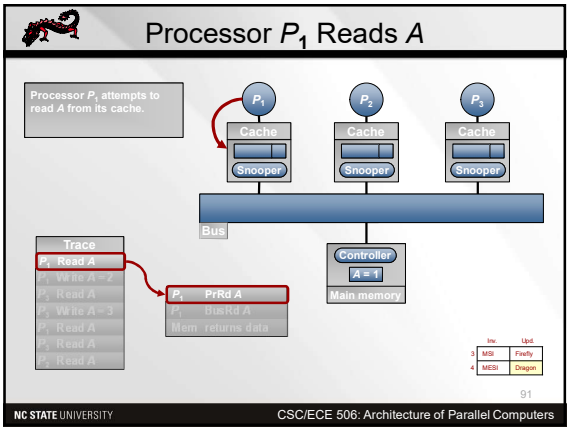


89

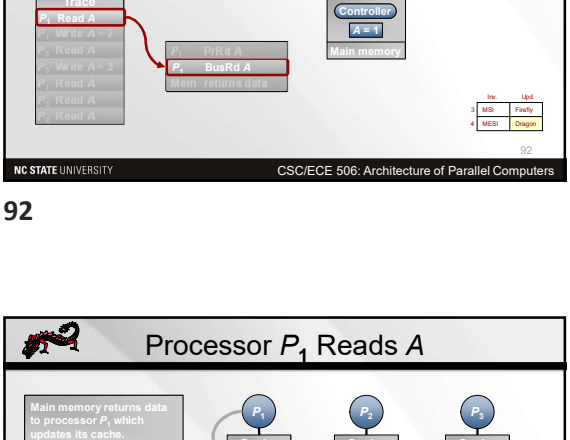
Dragon Visualization



90

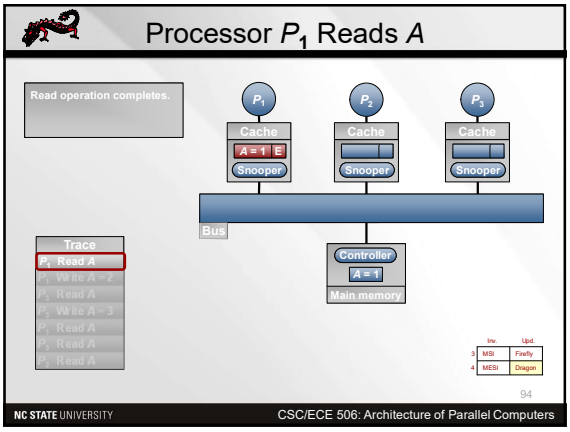


91

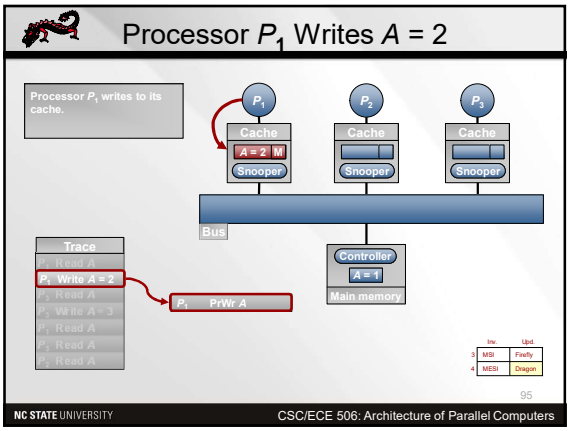


92

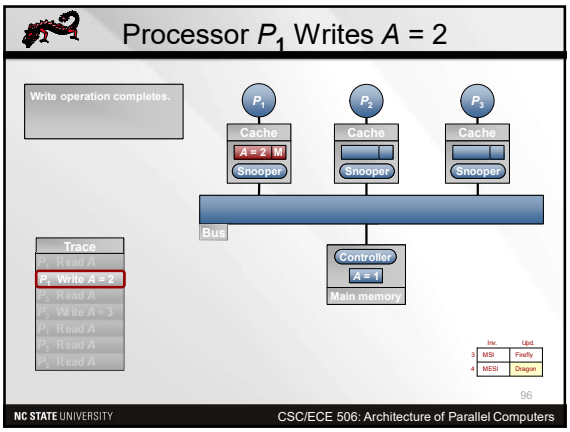
93



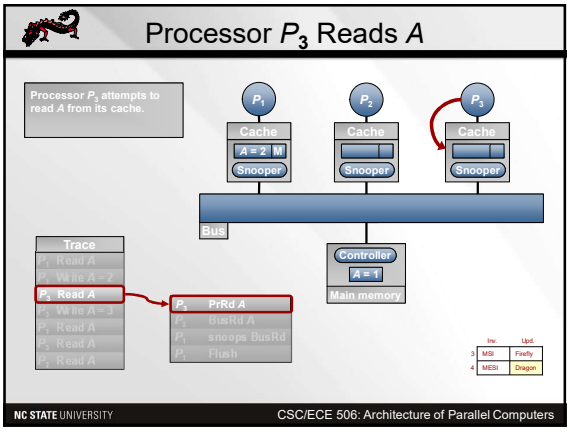
94



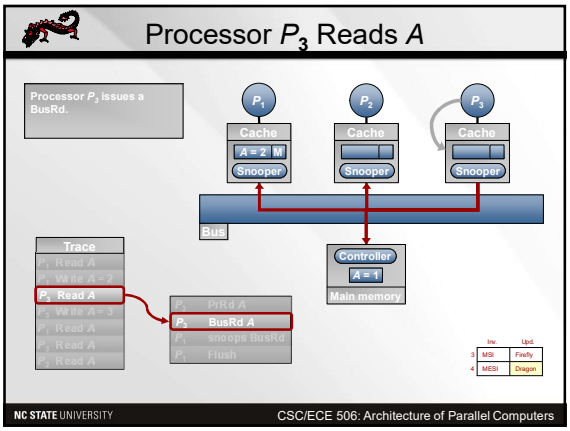
95



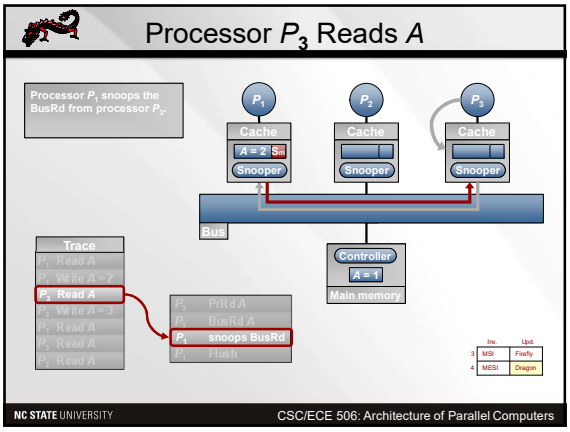
96



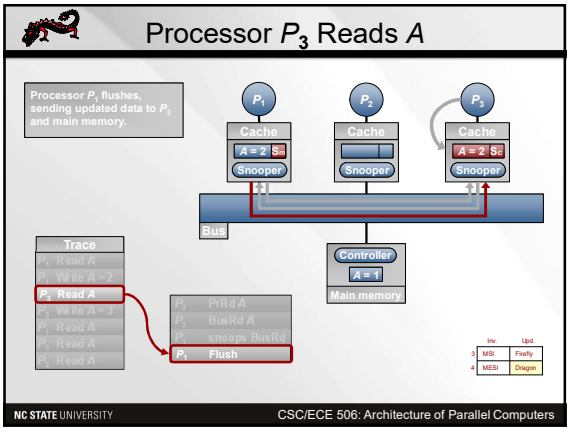
97



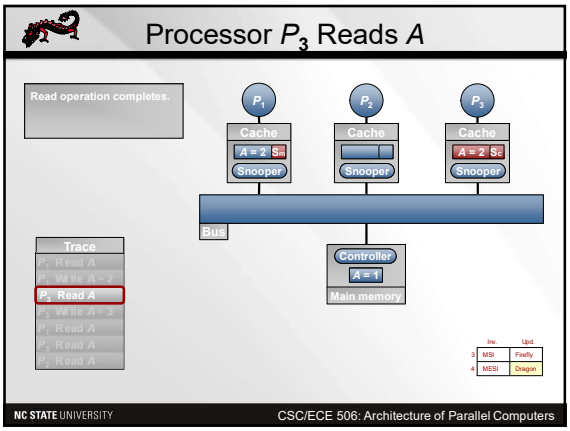
98



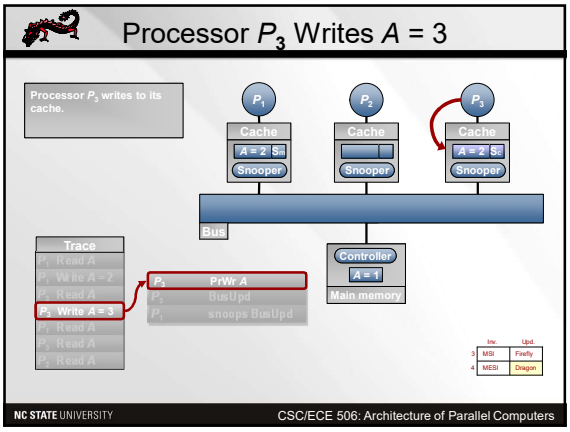
99



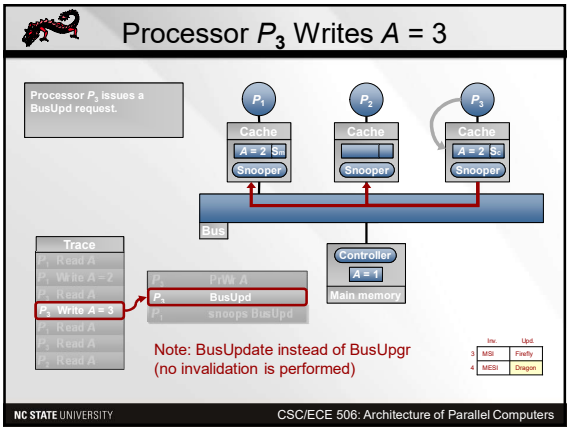
100



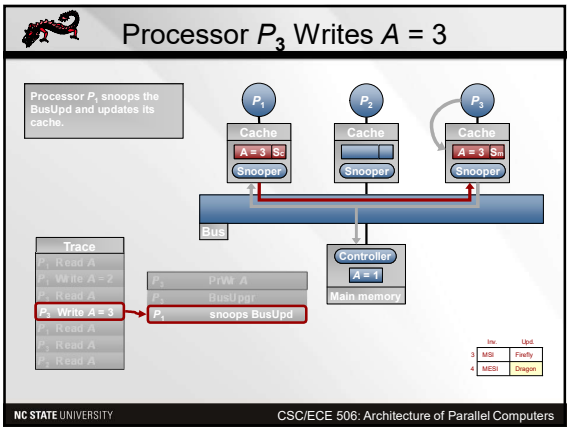
101



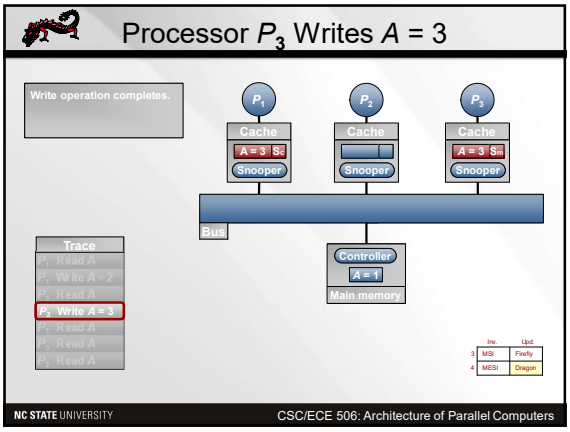
102



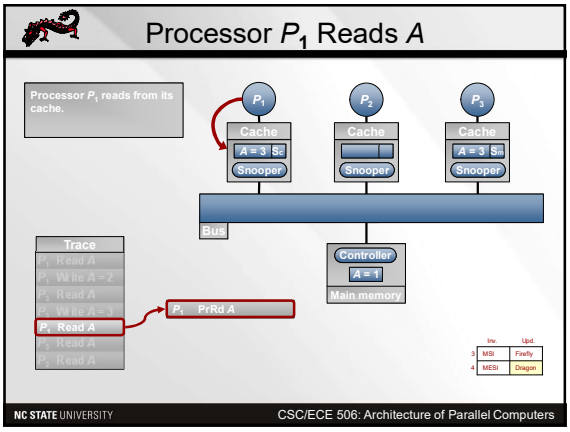
103



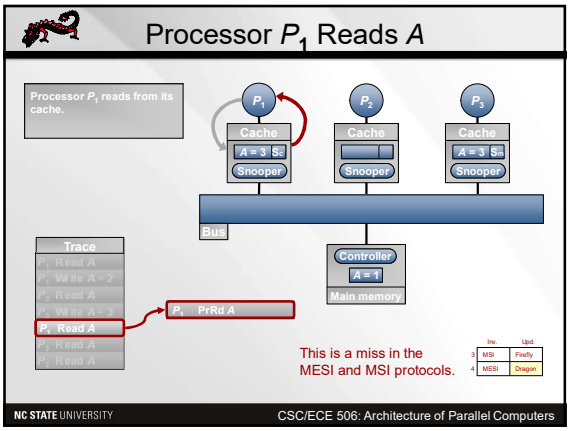
104



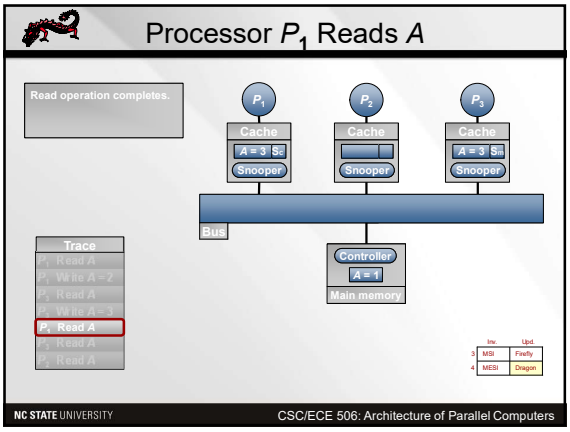
105



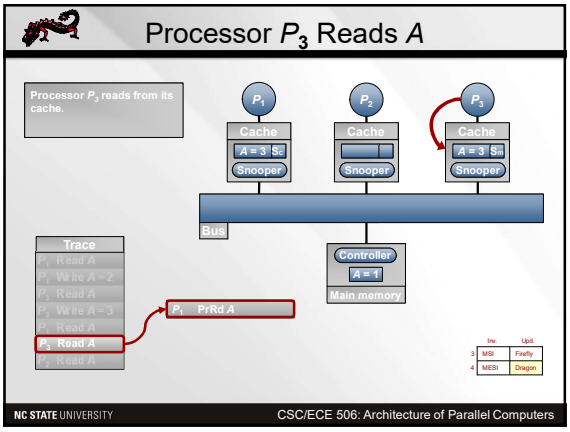
106



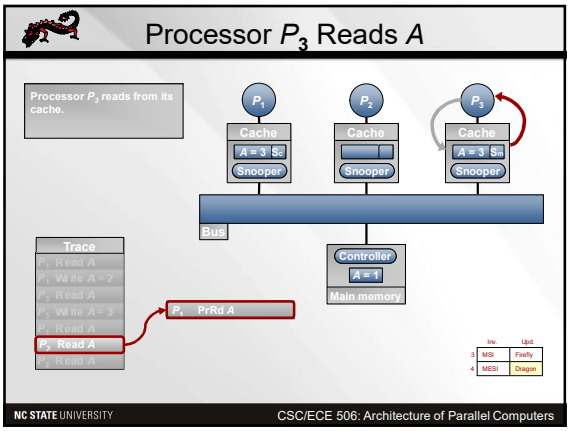
107



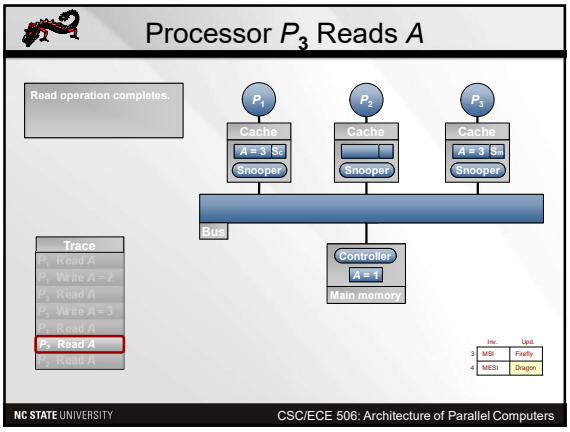
108



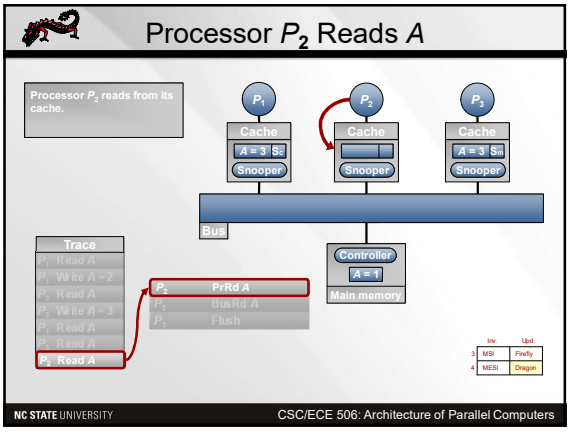
109



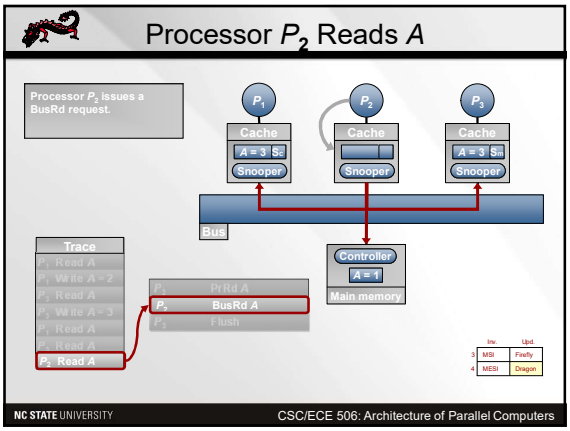
110



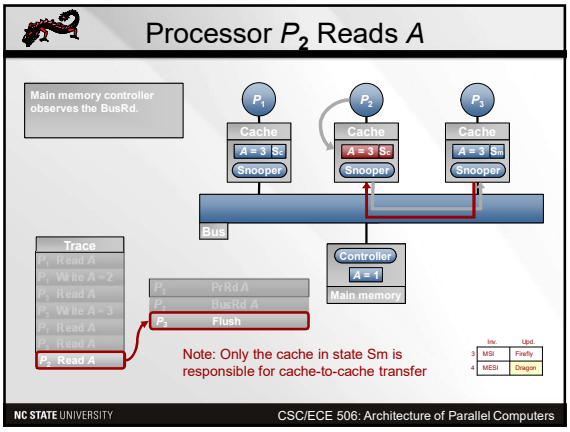
111



112

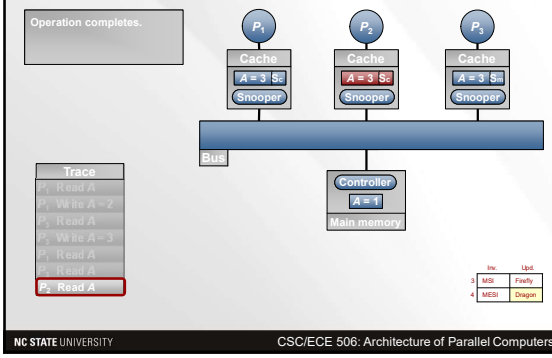


113



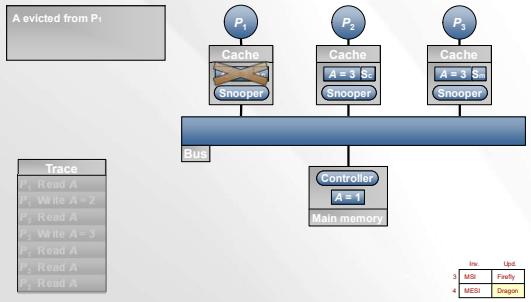
114

Processor P_2 Reads A



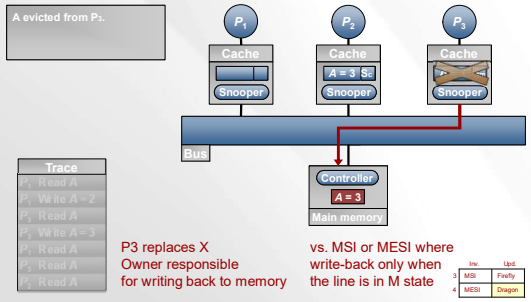
115

P_1 Replaces A



116

P_1 Replaces A



117

Dragon Example

Proc Action	State P1	State P2	State P3	Bus Action	Data from
R1	E	–	–	BusRd	Mem
W1	M	–	–	–	Own cache
R3	Sm	–	Sc	BusRd/Flush	P1 cache
W3	Sc	–	Sm	BusUpd/Upd	Own cache
R1	Sc	–	Sm	–	Own cache
R3	Sc	–	Sm	–	Own cache
R2	Sc	Sc	Sm	BusRd/Flush	P3 cache

118

Lower-Level Protocol Choices

- Can shared-modified state be eliminated?
  - If memory is updated too on BusUpd transactions (DEC Firefly)
  - Dragon protocol doesn't (assumes DRAM memory slow to update)
- Should replacement of an Sc block be broadcast?
  - Would allow last copy to go to Exclusive state and not generate updates
  - Replacement bus transaction isn't in critical path, but later update may be
- Shouldn't update local copy on write hit before controller gets bus
  - Can mess up serialization
- Coherence, consistency considerations much like write-through case
- In general, there are many subtle race conditions in protocols.

119

Lecture 15 Outline

- MSI protocol
- MESI protocol
- Dragon protocol
- Firefly protocol

	Inval-idate	Update
3-state	MSI	Firefly
4-state	MESI	Dragon

120

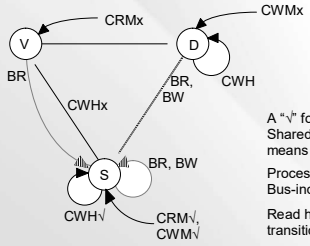
A Three-State Update Protocol

- Whenever a bus update is generated, suppose that main memory—as well as the caches—updates its contents.
- Then which state don't we need?
- What's the advantage, then, of having the fourth state?

- The Firefly protocol, named after a multiprocessor workstation developed by DEC, is an example of such a protocol.

121

Firefly State-Transition Diagram



Key:
CRM — CPU read miss
CWM — CPU write miss
CWH — CPU write hit
BR — bus read
BW — bus write

A "√" following a transition means SharedLine was asserted. An "x" means it was not.

Processor-induced transitions ——
Bus-induced transitions ·····

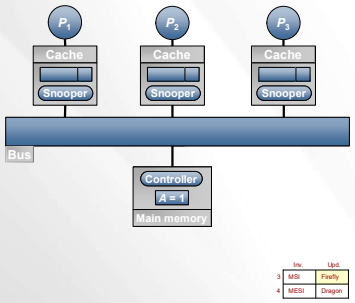
Read hits do not cause state transitions and are not shown.

- Answer some questions about this diagram.

122

Firefly Visualization

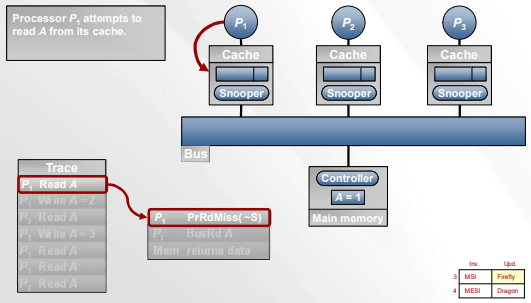
Start state. All caches empty and main memory has $A = 1$.



123

Processor P_1 Reads A

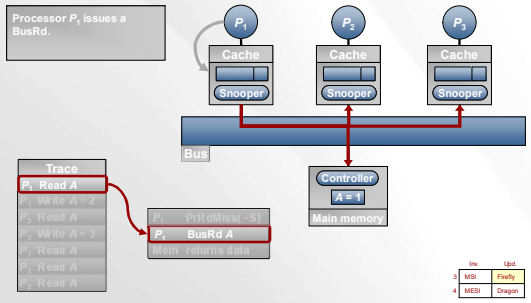
Processor P_1 attempts to read A from its cache.



124

Processor P_1 Reads A

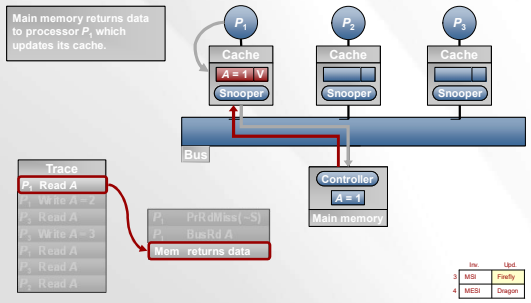
Processor P_1 issues a BusRd.



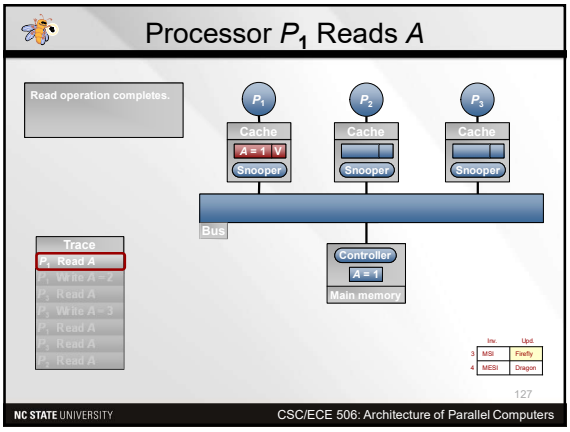
125

Processor P_1 Reads A

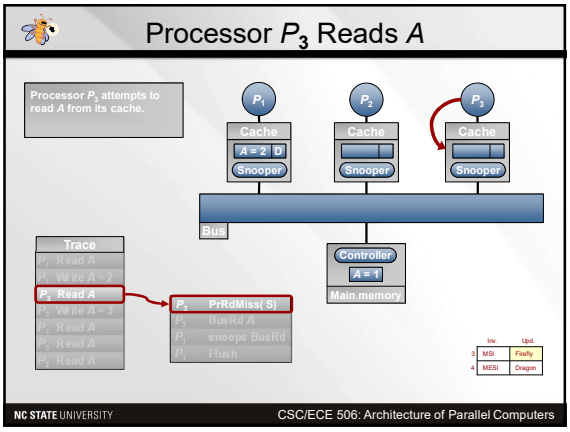
Main memory returns data to processor P_1 which updates its cache.



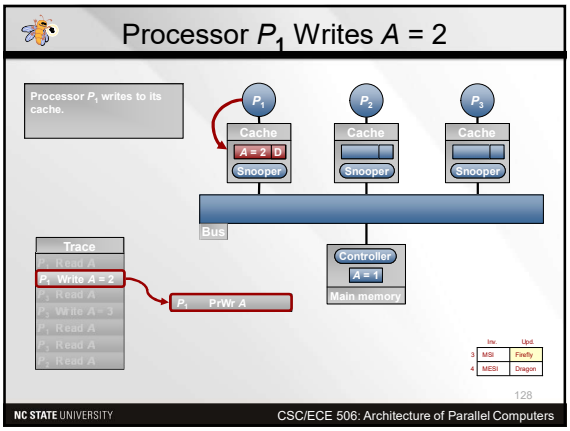
126



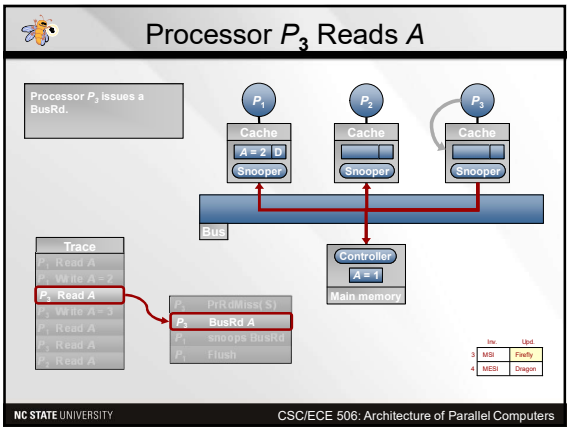
127



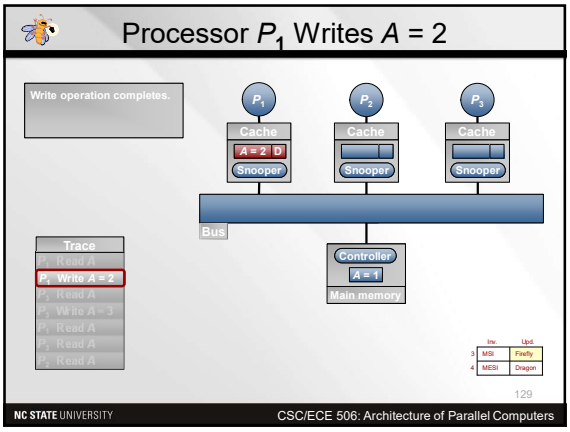
130



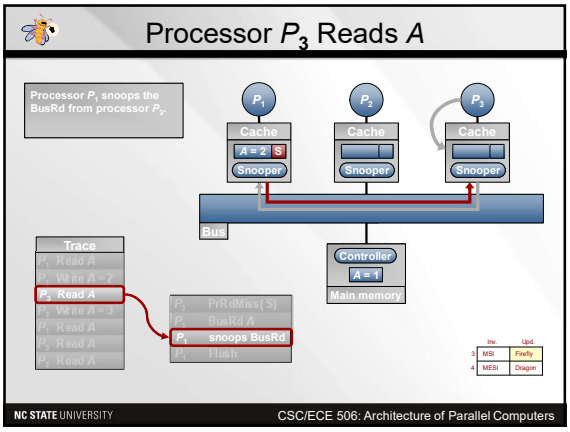
128



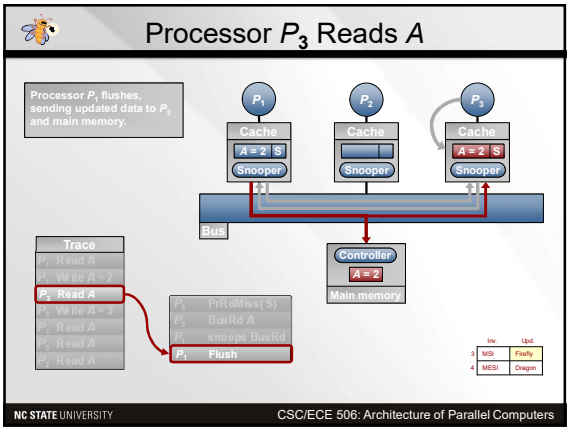
131



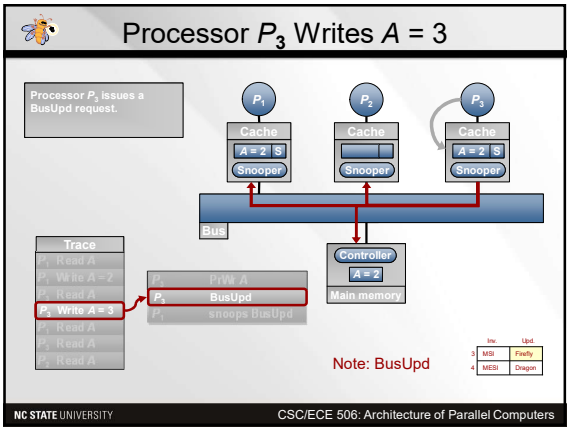
129



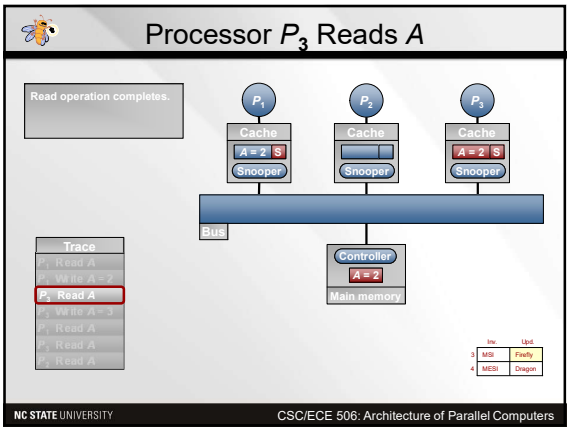
132



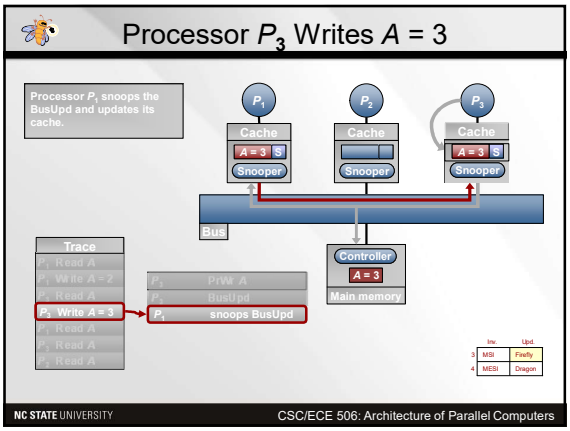
133



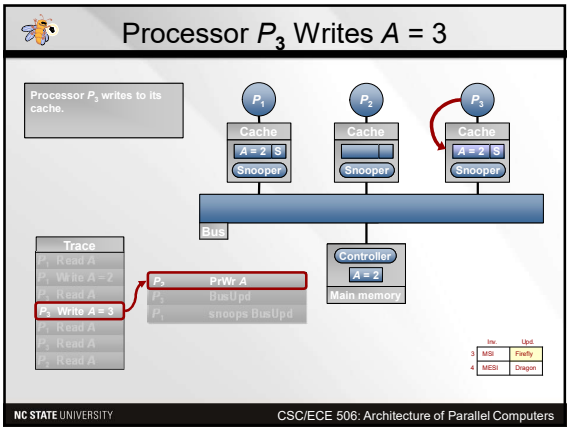
136



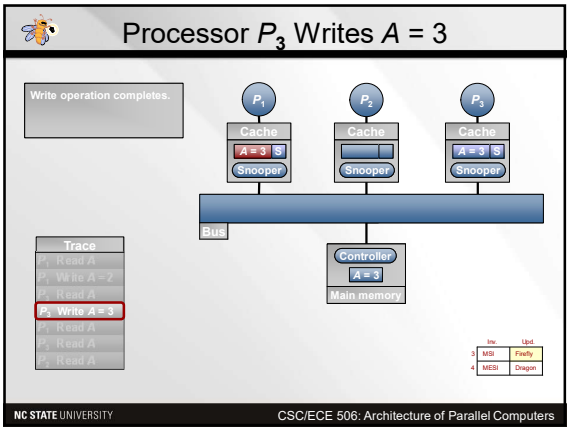
134



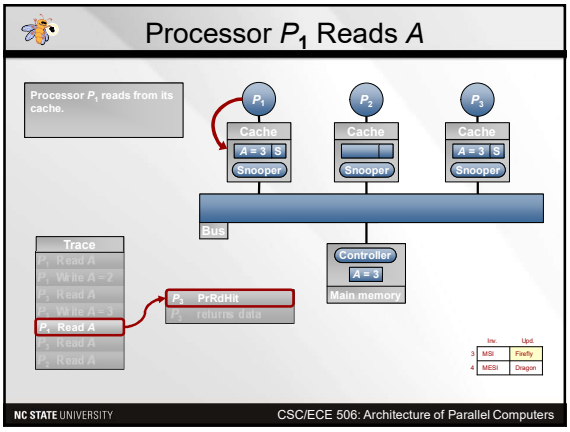
137



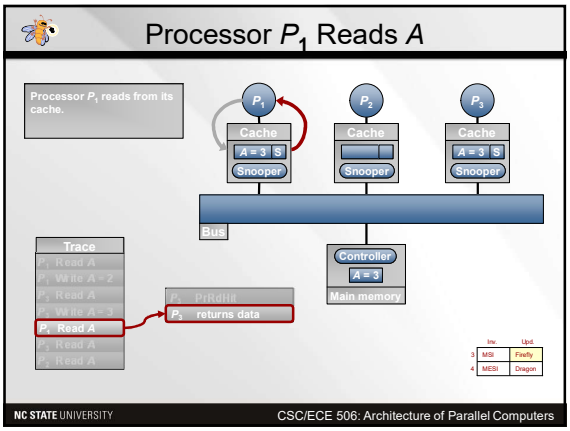
135



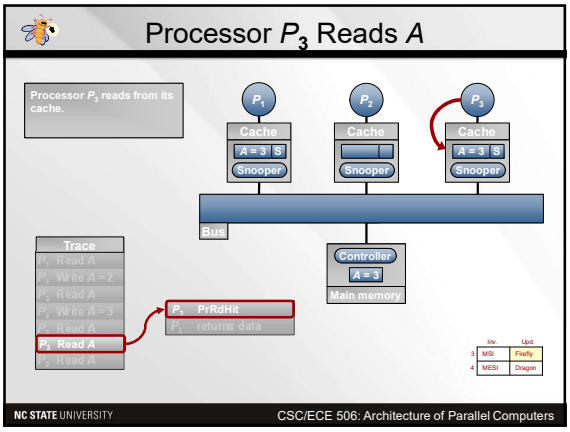
138



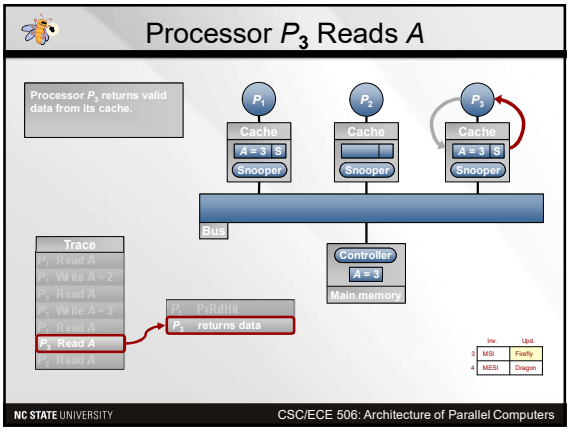
139



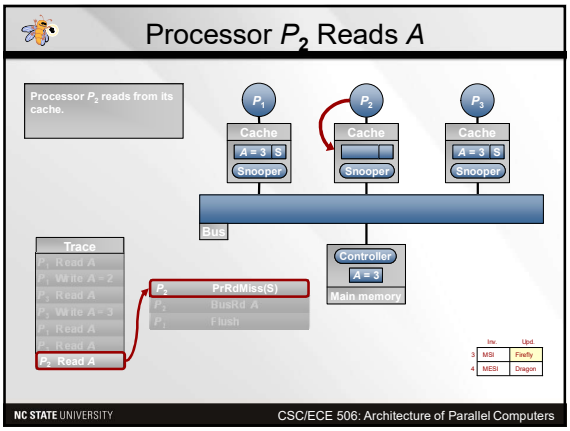
140



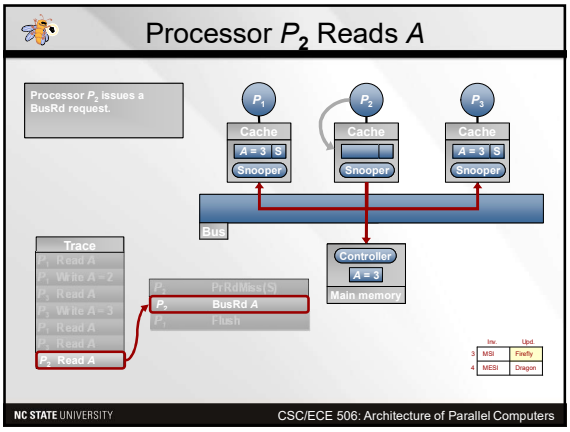
141



142

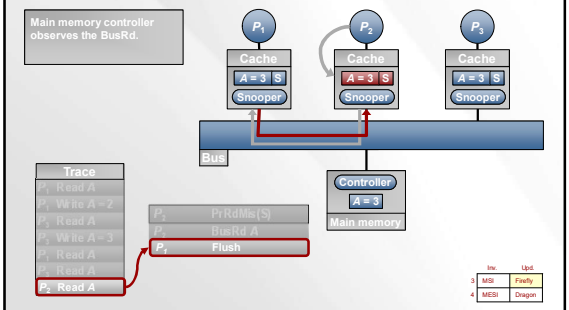


143



144

Processor P_2 Reads A



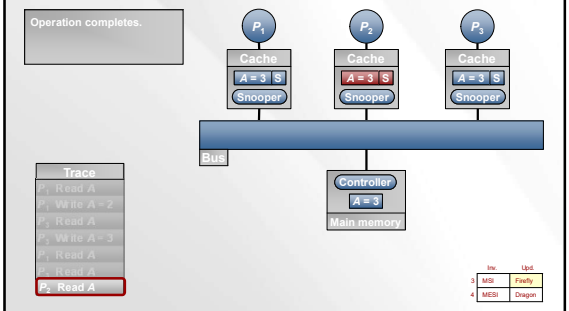
145

Firefly Example

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	*V*	–	–	BusRd	Mem
W1	*D*	–	–	–	Own cache
R3	S	–	*S*	BusRd/Flush	P1 cache
W3	*S*	–	S	BusUpd	Own cache
R1	*S*	–	S	-	Own cache
R3	*S*	–	S	–	Own cache
R2	*S*	*S*	S	BusRd/Flush	P1 Cache

148

Processor P_2 Reads A



146

Assessing Protocol Tradeoffs

- In the next lecture, we will look at results of comparing protocols by simulation.
- Methodology:
  - Use simulator; default 1MB, 4-way cache, 64-byte block, 16 processors. Some runs use 64K cache.
  - Focus on frequencies, not end performance for now
    - transcends architectural details, but not what we're really after
  - Use idealized memory performance model to avoid changes of reference interleaving across processors with machine parameters
    - Cheap simulation: no need to model contention

149

Firefly Example

Proc Action	State P1	State P2	State P3	Bus Action	Data From
R1	*V*	–	–	BusRd	Mem
W1	*D*	–	–	–	Own cache
R3	S	–	*S*	BusRd/Flush	P1 cache
W3	*S*	–	S	BusUpd	Own cache
R1	*S*	–	S	-	Own cache
R3	*S*	–	S	–	Own cache
R2	*S*	*S*	S	BusRd/Flush	P1 Cache

147