

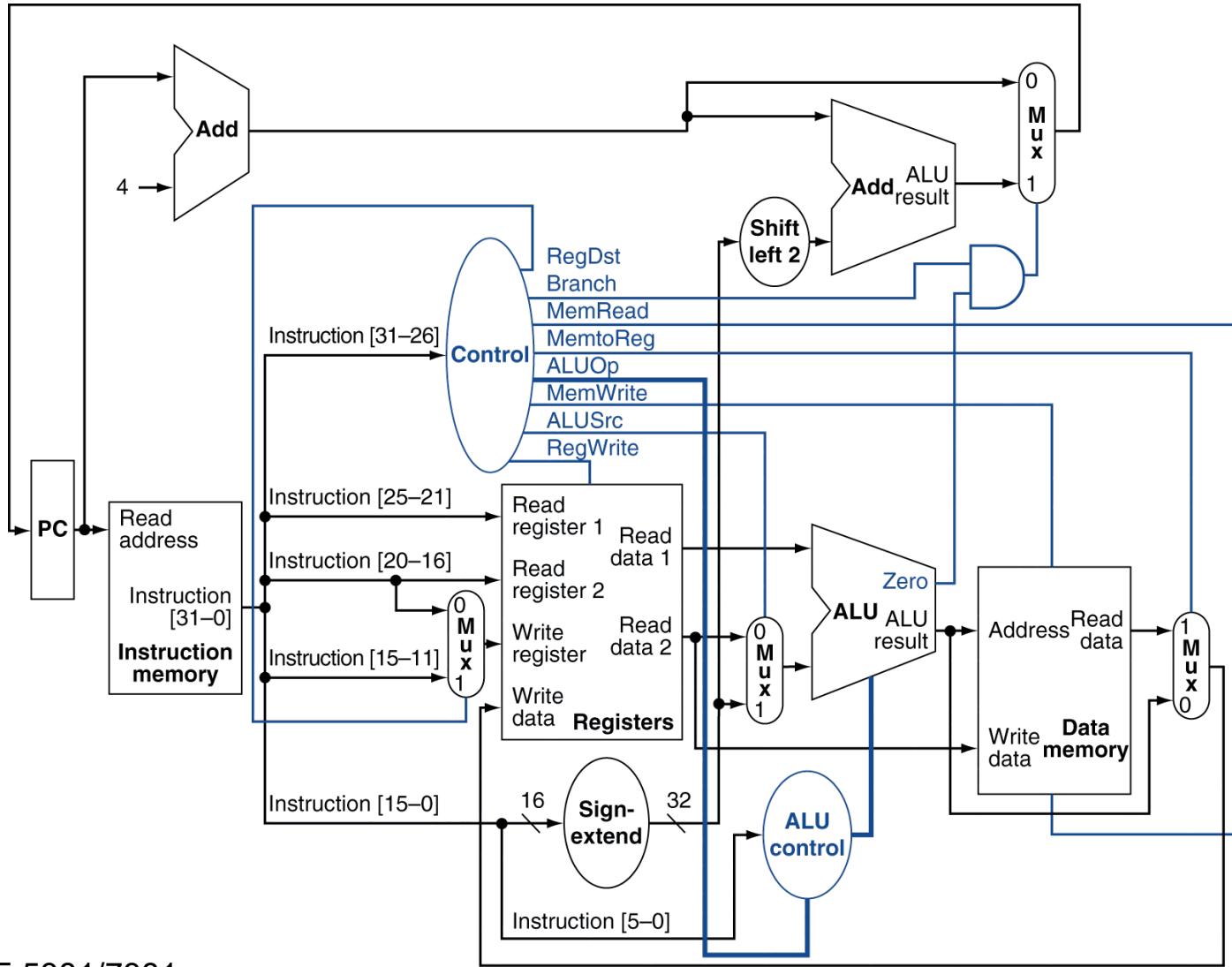
CS/ECE 5381/7381
Computer Architecture
Spring 2023

Dr. Manikas
Computer Science
Lecture 21: Apr. 18, 2023

Project 7

- Due Sat, May 6 (11:59 pm)
 - (Final Exam Period)
- Xcelium tool – more Verilog
- Assignment:
 - Develop Verilog code to implement the MIPS Data Path Control Unit
 - (common control signals – recall from App. A)

Datapath With Control



Project 7

- Control unit will need to handle the following MIPS instructions: **sub**, **addi**, and **lw**
- Control unit will need to output values for the following control signals:
 - **RegDst**
 - **ALUSrc**
 - **MemRead**
 - **MemWrite**
 - **MemtoReg**

Thread-Level Parallelism

(Chapter 5, Hennessy and Patterson)

Note: some course slides adopted
from publisher-provided material

Outline

- 5.1 Introduction
- 5.2 Centralized Shared-Memory Architectures

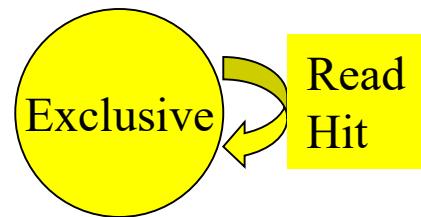
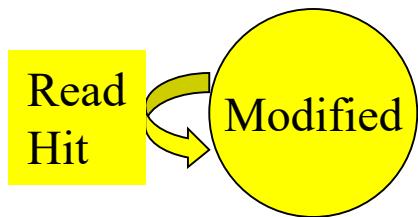
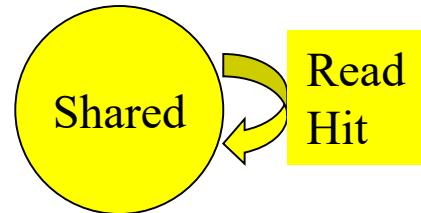
MESI Protocol – Cache Line States

- **(M) Modified** - cache line has been modified, is different from main memory - is the only cached copy. (multiprocessor ‘dirty’)
- **(E) Exclusive** - cache line is the same as main memory and is the only cached copy
- **(S) Shared** - Same as main memory but copies may exist in other caches.
- **(I) Invalid** - Line data is not valid (as in simple cache)

MESI Protocol Operation

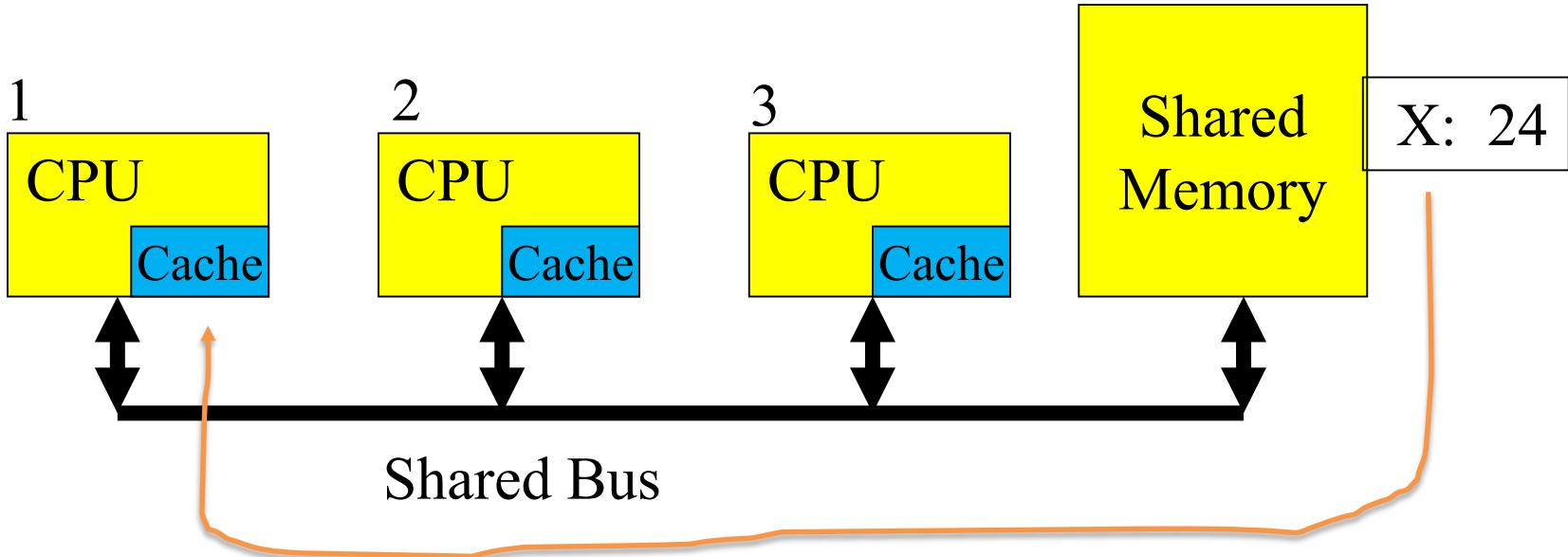
- Operation can be described informally by looking at action in local processor
 - A. Read Hit
 - B. Read Miss
 - C. Write Hit
 - D. Write Miss
- More formally by state transition diagram
- We will use a working example to illustrate

Read Hit State Diagram



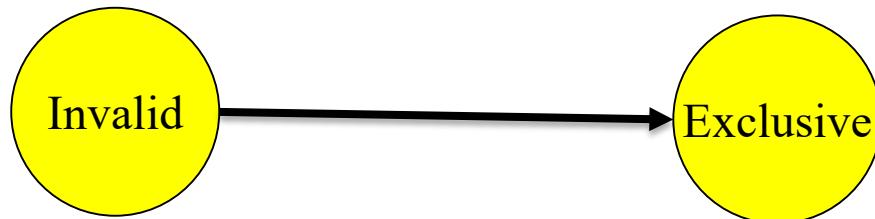
Local Read Miss (1)

- **If no copies in other caches**
 - Processor must make a bus request to main memory
 - Value read to local cache
 - Cache line state is now E (Exclusive)
 - Cache line same as main memory and is only cached copy



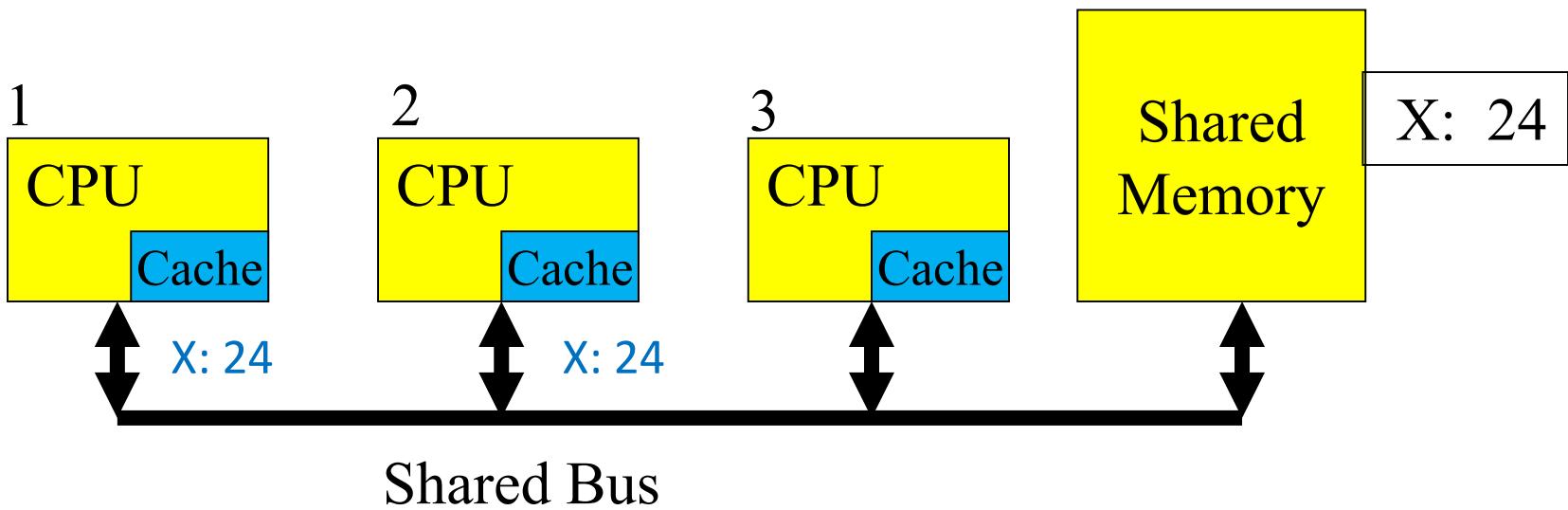
Read miss for Cache 1: get X from shared (main) memory and read into appropriate cache line for cache 1 (assume X is in block 257)

Cache 1 MESI state diagram

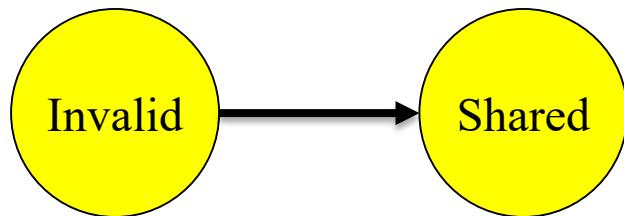


Local Read Miss (2)

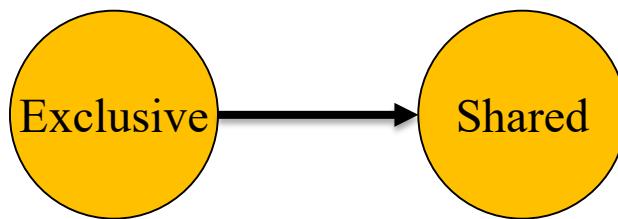
- **Else, if another cache has an exclusive copy of item (state E)**
 - Processor makes a bus request to main memory
 - Snooping cache puts copy of item on bus
 - Value read to local cache
 - Both cache line states are now S (Shared)
 - Both cache lines now have same value as main memory



Cache 1 MESI state diagram

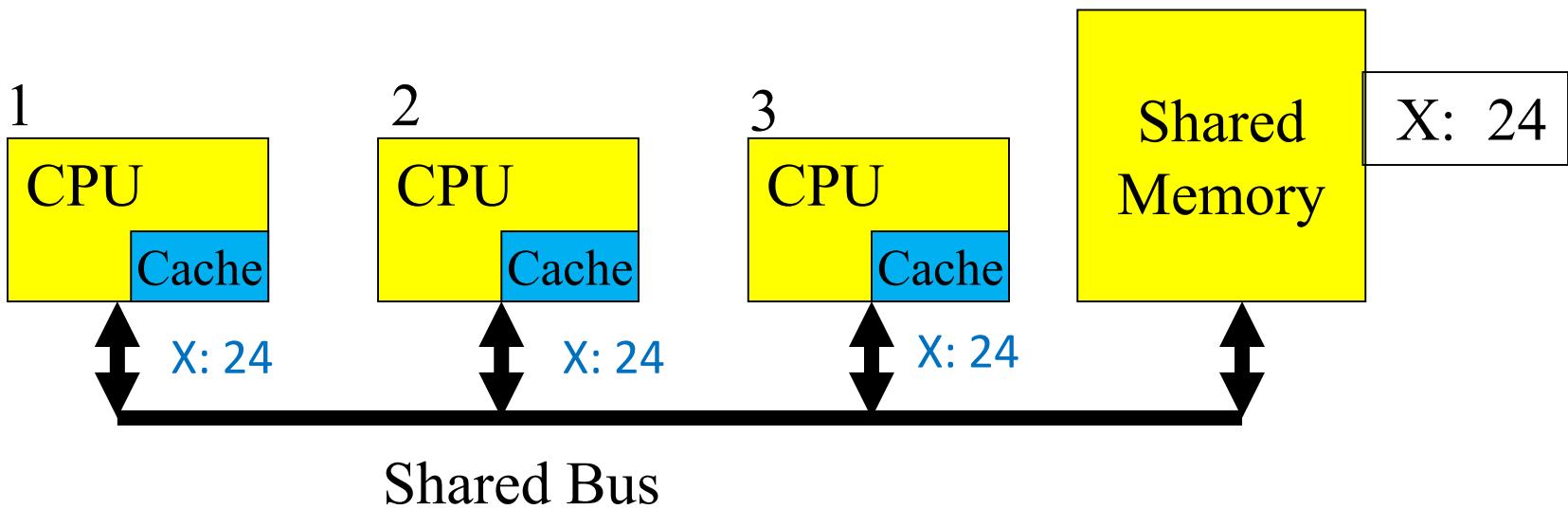


Cache 2 MESI state diagram

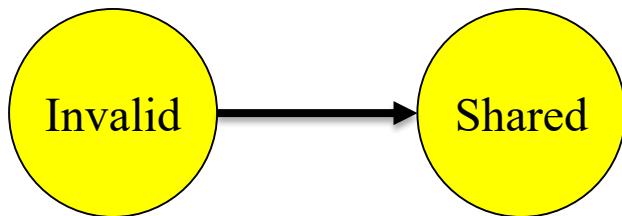


Local Read Miss (3)

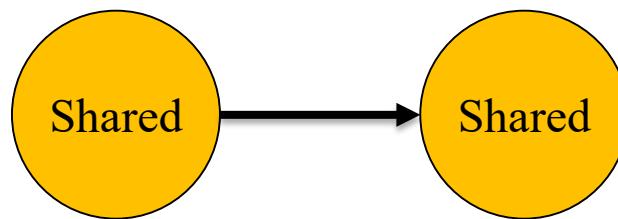
- **Else, if other caches have a shared copy of item (state S)**
 - Processor makes a bus request to main memory
 - **One cache** puts copy of item on bus
 - Value read to local cache
 - Local cache state is now S
 - Other caches with copies remain at state S



Cache 1 MESI state diagram

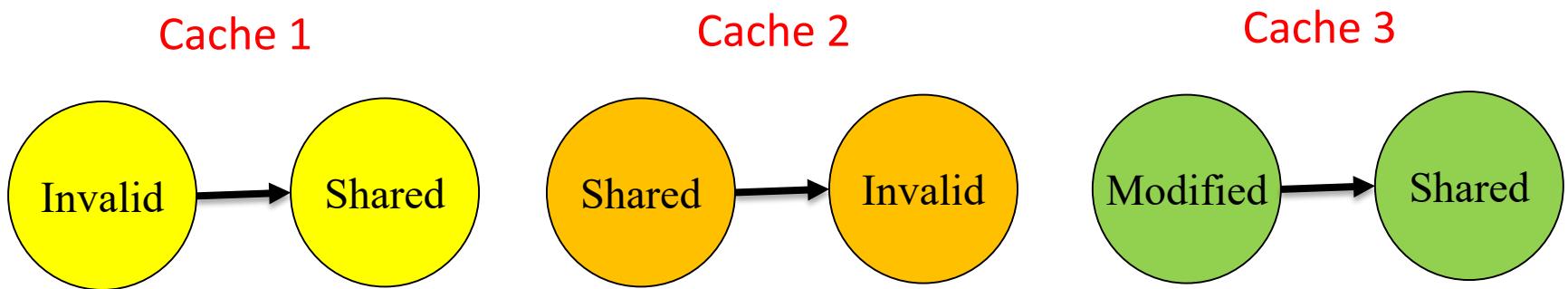
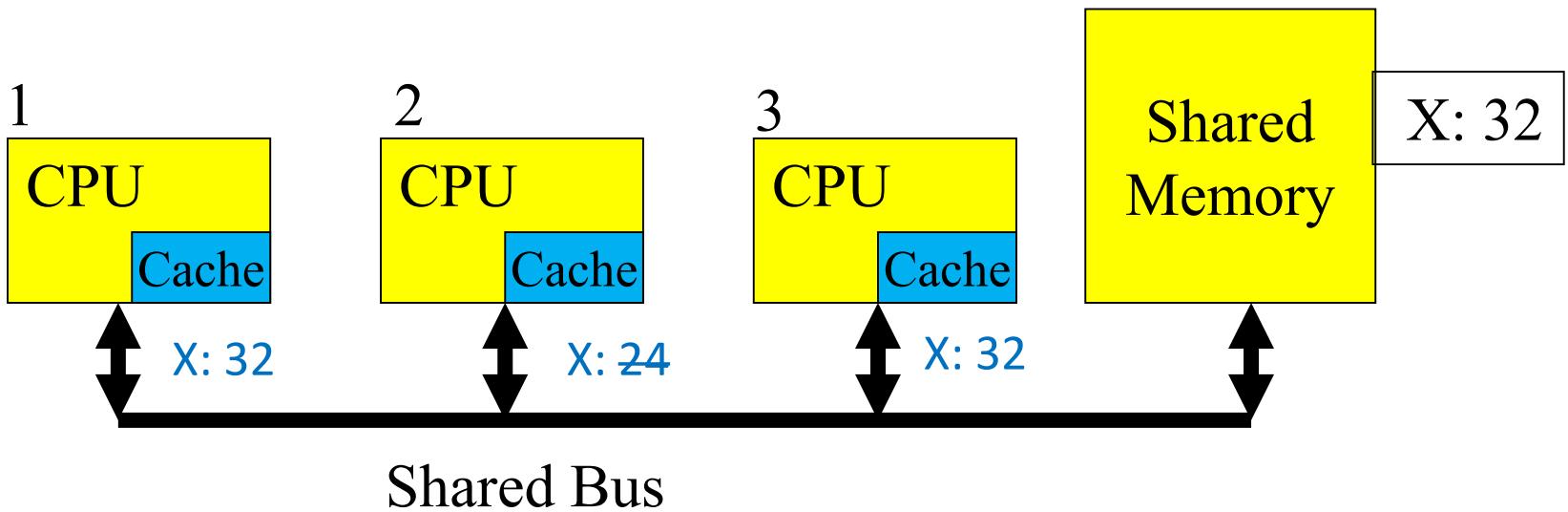


Cache 2 and 3 MESI state diagram



Local Read Miss (4)

- Else, if one cache has a *modified* copy of item (state M)
 - Processor makes a bus request to main memory
 - Cache puts copy of item on bus
 - Value read to local cache
 - Local cache state is now S
 - M cache copies value to main memory
 - State changes from M to S

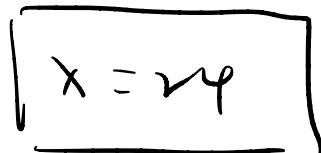


CACHE 1

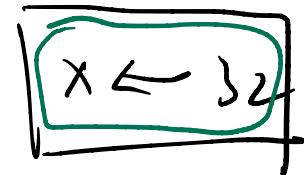
MISS



CACHE 2



CACHE 3



READ MISS

I

S

M

$X = 32$

$X = 24$ "DIRTY"
(OLD)

$X = 32$

From CACHE 3

S

I

S

$X : \text{READ HIT}$

S

$X = 32$ (COPY)

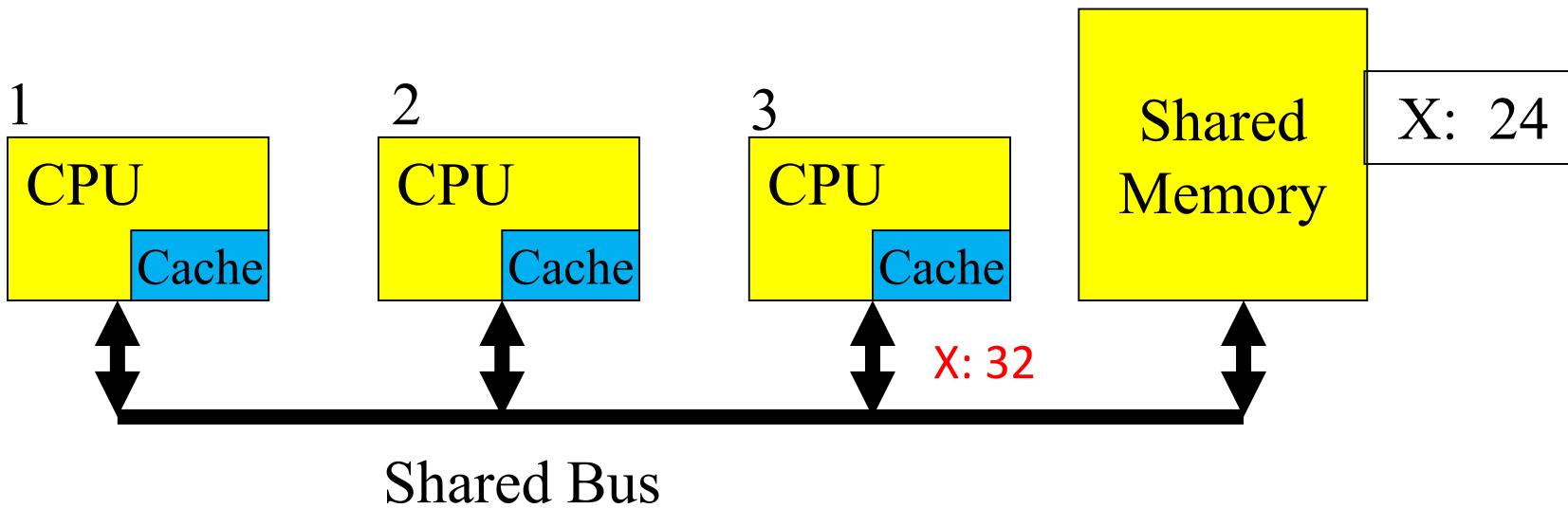
S

$X : \text{READ HIT}$

S

Local Write Hit (M)

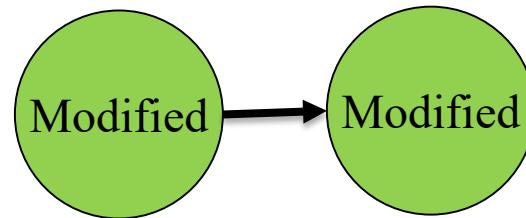
- **Cache line state M: cache line was modified but main memory not yet updated**
 - Cache line is exclusive and already “dirty” (old)
 - Update local cache value
 - No state change



Cache 3 has a *modified* copy of X.

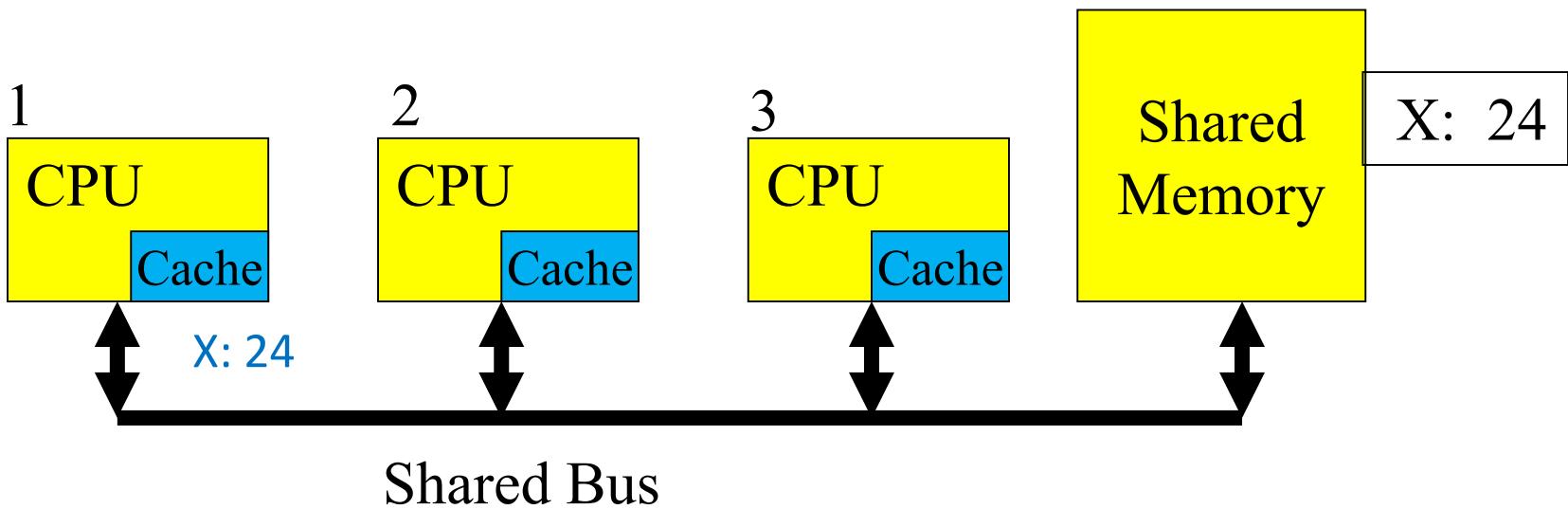
Update Cache 3 line with new value of X.

Cache 3



Local Write Hit (E)

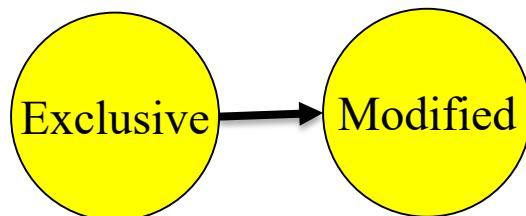
- **Cache line state E: cache line same as main memory and is the only cached copy**
 - Update local cache value
 - Cache line state changes to M since we are modifying value



Cache 1 line is in **Exclusive (E)** state: same value as main (shared) memory for address X

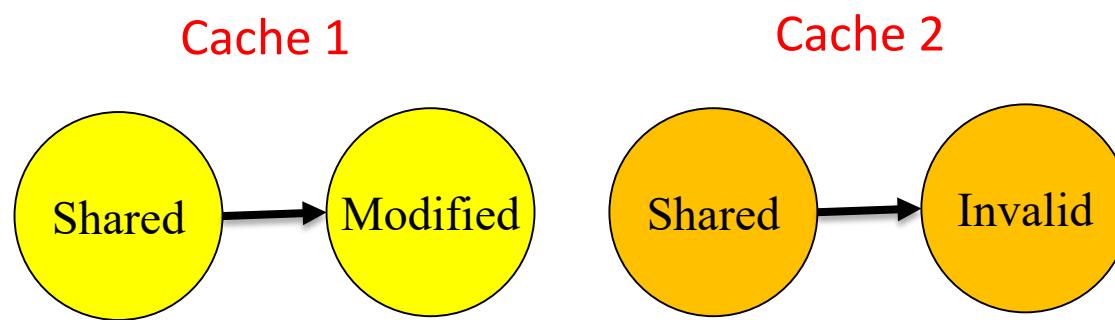
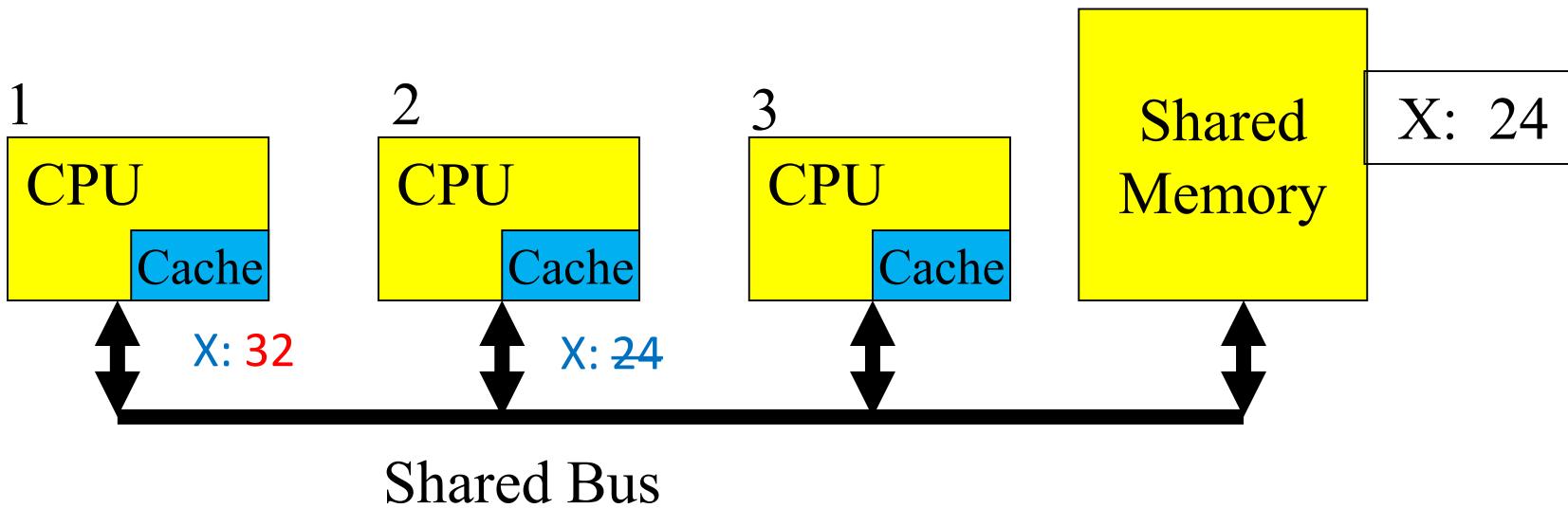
Update Cache 1 line with new value of X: **Modified (M)** state

Cache 1

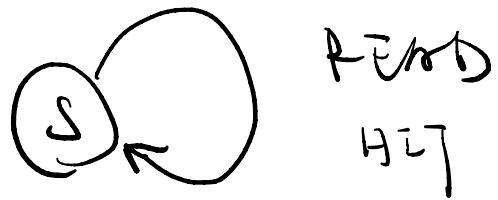


Local Write Hit (S)

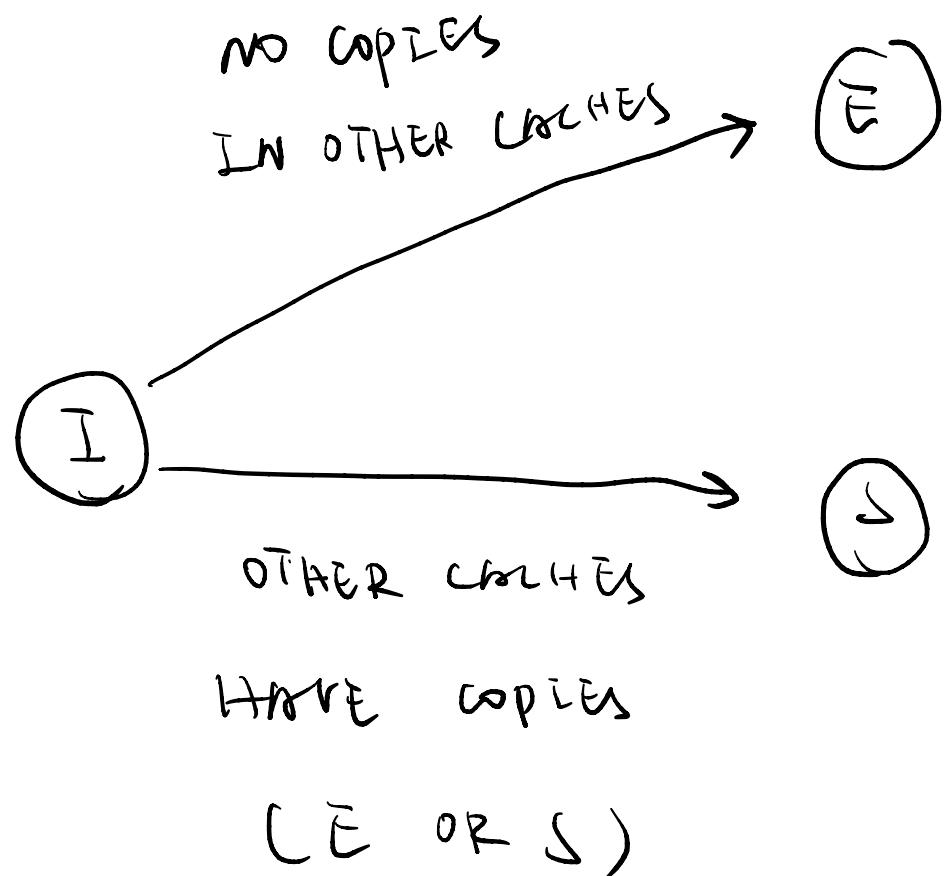
- **Cache line state S: cache line is same as main memory and there may be copies in other caches**
 - Processor broadcasts “invalidate” on bus
 - Snooping processors with S caches change cache line state to I (Invalid)
 - Local cache value is updated
 - Local cache line state changes to M (Modified)



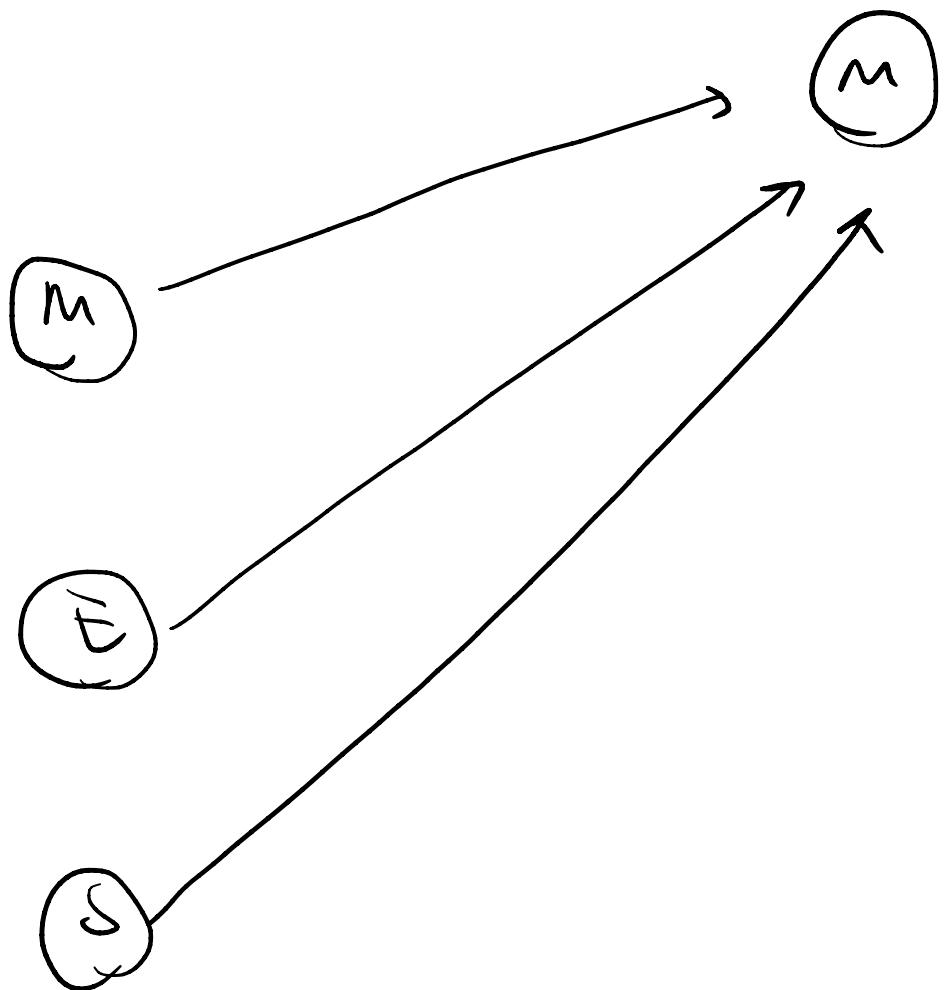
REVIEW : Rend Hit



REVIEW : READ MISS



REVIEW : WRITE HIT



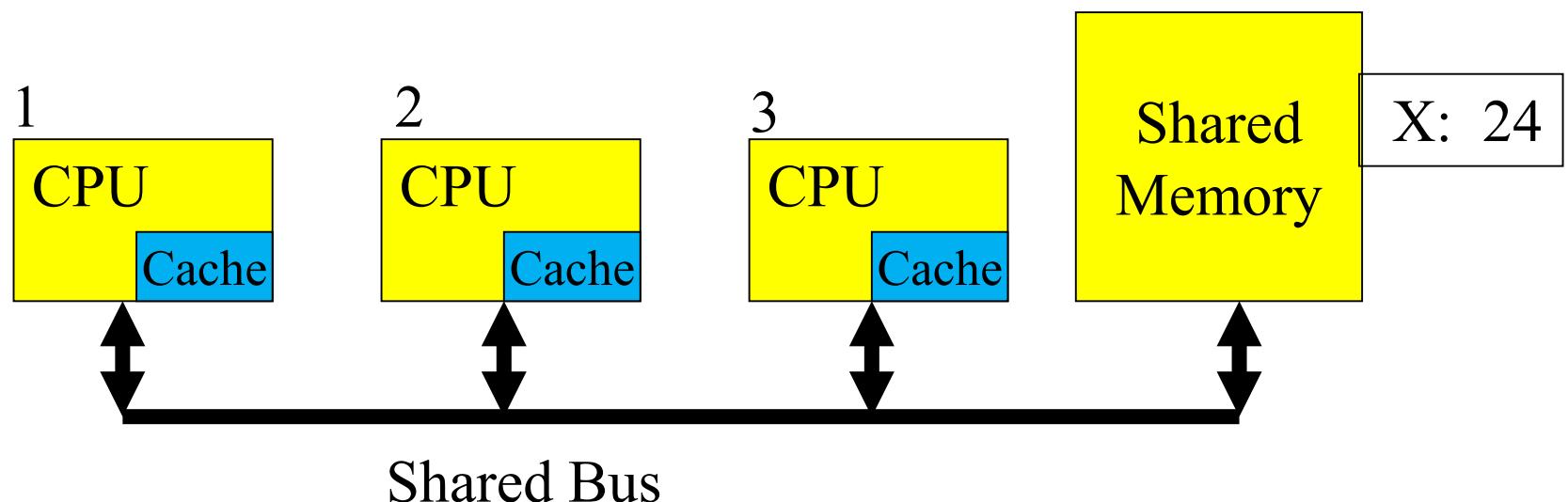
WRITE
MODIFIES
CACHE
VALUE

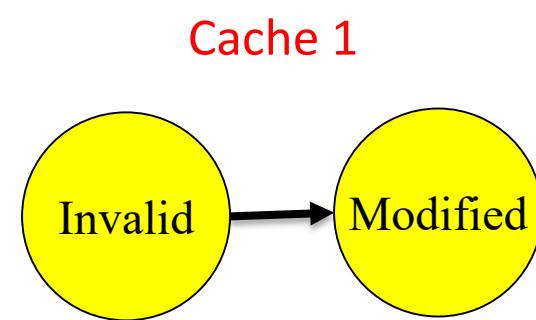
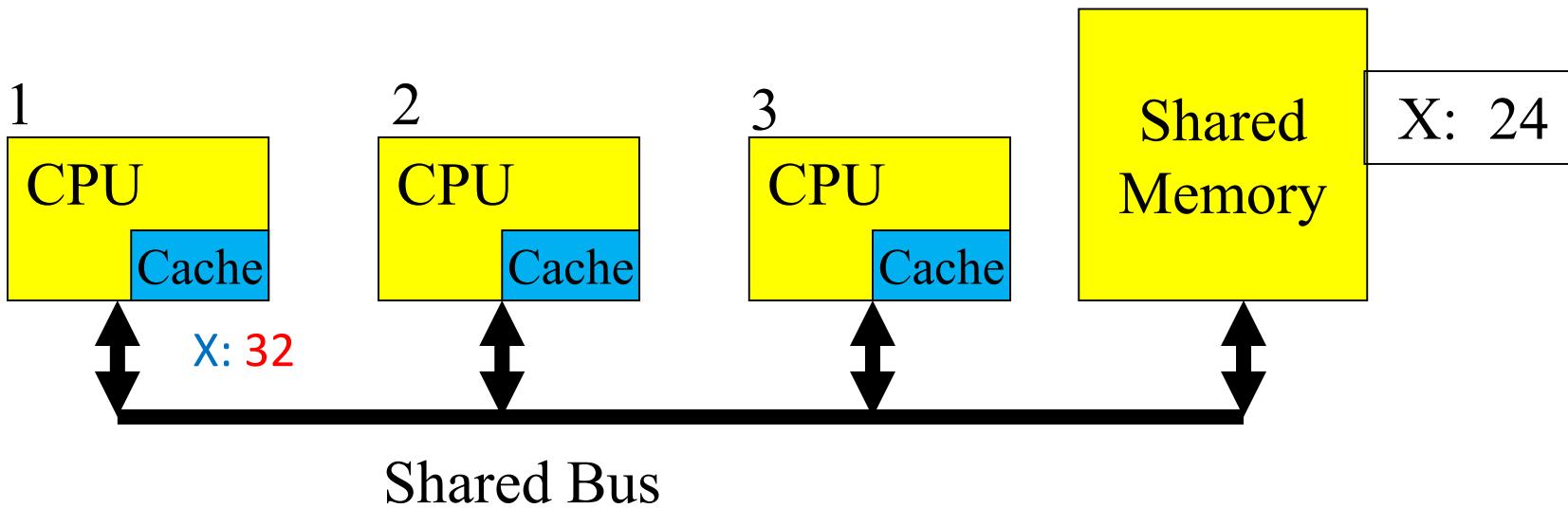
D. Local Write Miss

- Write Miss: address not found in cache
 - Cache line state is I (Invalid)
- Action depends on copies of X in other caches, if they exist

Local Write Miss (1)

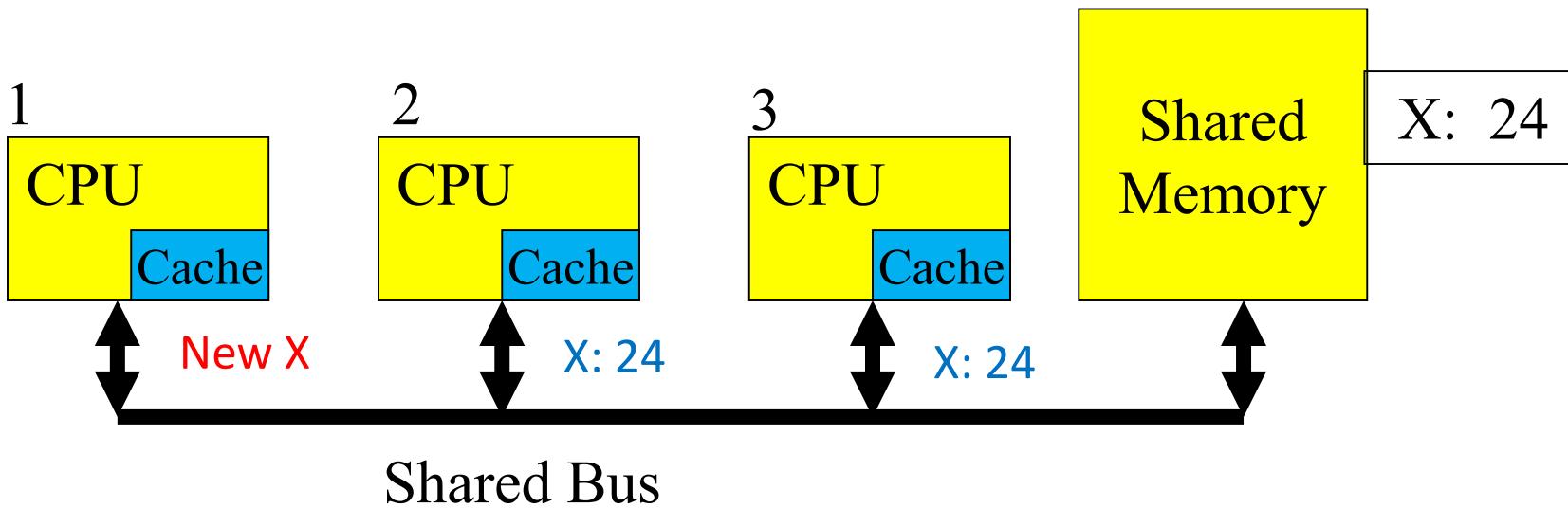
- If no copies of X exist in other caches
 - Update value of X in local cache line
 - Set local cache state to M (modified)





Local Write Miss (2)

- **Else, if 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

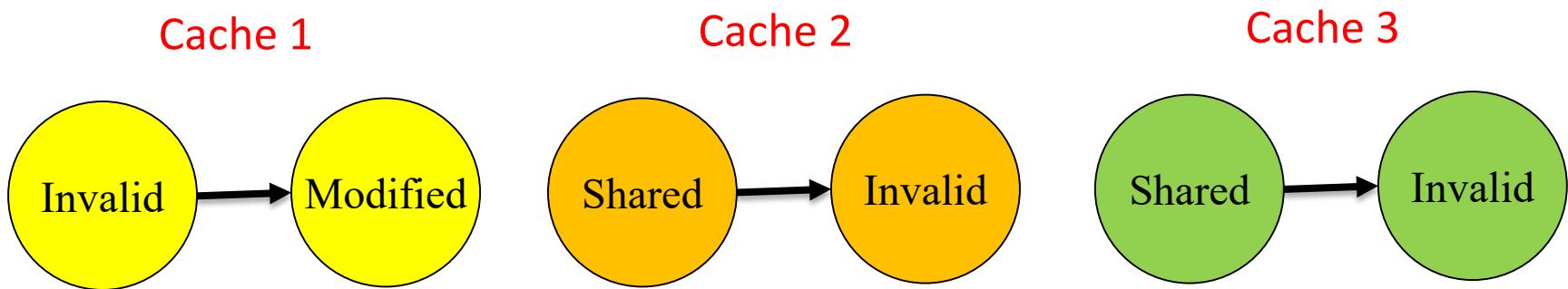
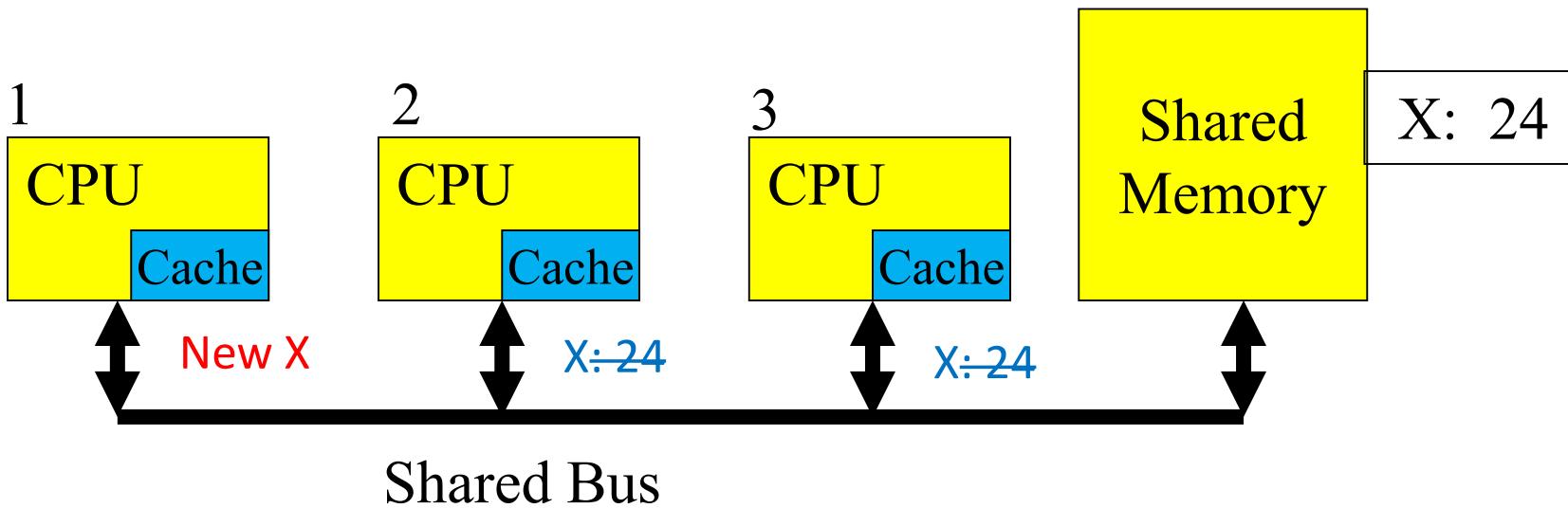


Caches 2 and 3 both have copy of current value of X

Cache 1 does NOT have copy of X, but wants to modify X. CPU 1 sends RWITM signal to bus

Caches 2 and 3 set their copies of X to “invalid”

Cache 1 modifies X



Warehouse-Scale Computers

(Chapter 6, Hennessy and Patterson)

Note: some course slides adopted
from publisher-provided material

Outline

- 6.1 Introduction
- 6.2 Programming Models and Workloads
- 6.3 Architecture
- 6.4 Physical Infrastructure
- 6.5 Cloud Computing

REVIEW OF COURSE

(CH. 1: OVERVIEW)

PROCESSORS: CH. 3

MEMORY: CH. 2

VECTOR PROCESSING: CH. 4

MULTIPLE PROCESSORS: CH. 5

CH. 6: MODERN APPLICATIONS

⇒ "WARPHOUSE-SCALE COMPUTERS"

Internet Services Becoming Prevalent

تَنَاهُ

- Internet services:
 - Webmail, search, social network, maps, video/picture sharing, storage, video streaming, language translation, ...
- The trend toward server-side or “cloud” computing
 - Moving away from PC-like clients to smaller, often mobile, devices combined with large-scaled internet services
 - Many Internet services were previously resided in the client, but now are moved to the cloud

الإنترنت

مُهَاجِرٌ

"CLIENT"

vs

"CLOUD"

"GENUINE SERVERS"

YOUR

COMPUTER

(PC/Laptop)

(Tablet/Phone)

PROGRAMS

ON YOUR MACHINE

S/7381: MARS TOOL

(MIPS: PROJECTS 1-3)

REMOTE

COMPUTER (SERVERS)

PROGRAMS/DATA

RUN ON SERVER

- YOU HAVE REMOTE
ACCESS.

S/7381: XCELLON (GENIUS)

(VERILOG: PROJECTS 4-7)

Internet Services Becoming Prevalent

- Characteristics of internet services
 - Multiple individual programs that interact to implement complex services to serve a massive amount of end users → one service for many

What Computers for Internet Services?

- WSC: *warehouse-scale computer*
 - Large-scaled internet services require a computing power that can only be provided by clusters of 100s or 1000s of machines, i.e. a warehouse full of computers
 - Hardware and software resources in such a facility must work in concert as would a single computer
- **datacenter is the computer**

Warehouse-Scale Computer (WSC)

- Large clusters – 50 to 100 thousand servers
- Housed in dedicated buildings, with special electrical and cooling requirements
- Provides Internet services
 - Google
 - Amazon



Warehouse-Scale Computer (WSC)

- Program is an Internet service
 - May consist of tens or more individual programs that interact to implement complex end-user services such as email, search, or maps
 - Software runs on hardware consisting of thousands of individual computing nodes with:
 - Networking and storage subsystems
 - Power distribution and conditioning equipment
 - Extensive cooling systems
- 大规模

Warehouse-Scale Computer (WSC)

- The enclosure for these systems is in fact a building structure and often indistinguishable from a large warehouse
 - Hence the name “Warehouse”-Scale Computer

WSC Design Factors

- Important design factors for WSC:
 - Cost-performance
 - Small savings add up
 - Energy efficiency
 - Affects power distribution and cooling
 - Work per joule
 - Dependability via redundancy
 - Network I/O
 - Interactive and batch processing workloads

WSC DESIGN FACTORS

- WE HAVE 100,000+ COMPUTERS (SERVERS)
- POWER - CONSUMPTION (EXTRA POWER)
 - DISSIPATION (HEAT - EXTRA COOLING)
- NETWORK CONNECTIVITY HIGH BANDWIDTH
- HIGH AVAILABILITY $\frac{MTTF}{MTTF + MTTR}$
- REDUNDANCY
 - BACK UP DATA
 - DUPLICATE STORAGE
- COST / PERFORMANCE

- HIGHER COST SINCE HIGHER PERFORMANCE
- BUT: CAN SPREAD COST AMONG MILLIONS OF USERS
- SECURITY: PHYSICAL (WAREHOUSE)
→ NETWORK

Outline

- 6.1 Introduction
- 6.2 Programming Models and Workloads
- 6.3 Architecture
- 6.4 Physical Infrastructure
- 6.5 Cloud Computing

Programming Models and Workloads

- Batch processing framework: MapReduce
 - **Map:** applies a programmer-supplied function to each logical input record
 - Runs on thousands of computers
 - Provides new set of key-value pairs as intermediate values
 - **Reduce:** collapses values using another programmer-supplied function

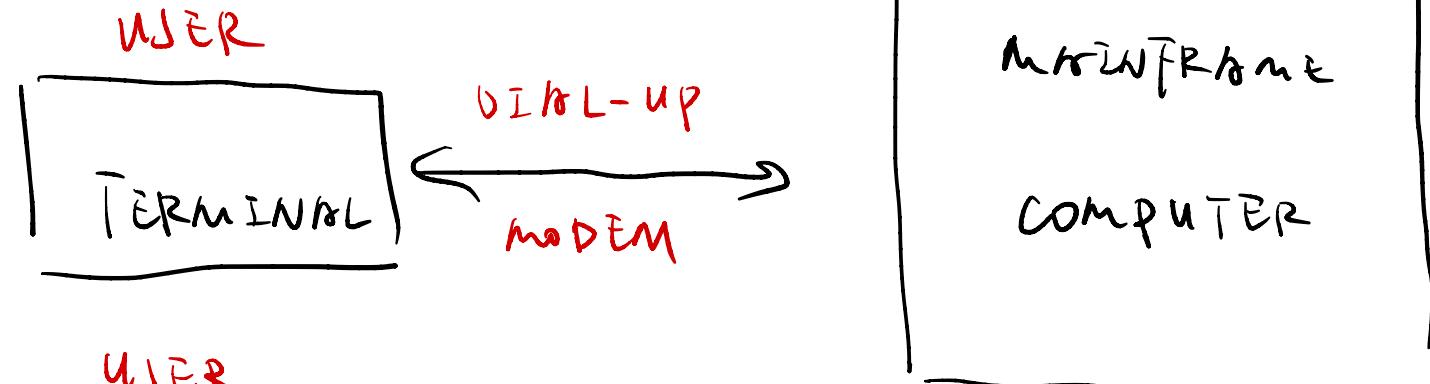
"BATCH PROCESSING"

- HANDLING DIFFERENT USERS
(PROGRAMS AND DATA) AT THE
SAME TIME \Rightarrow "PARALLEL"

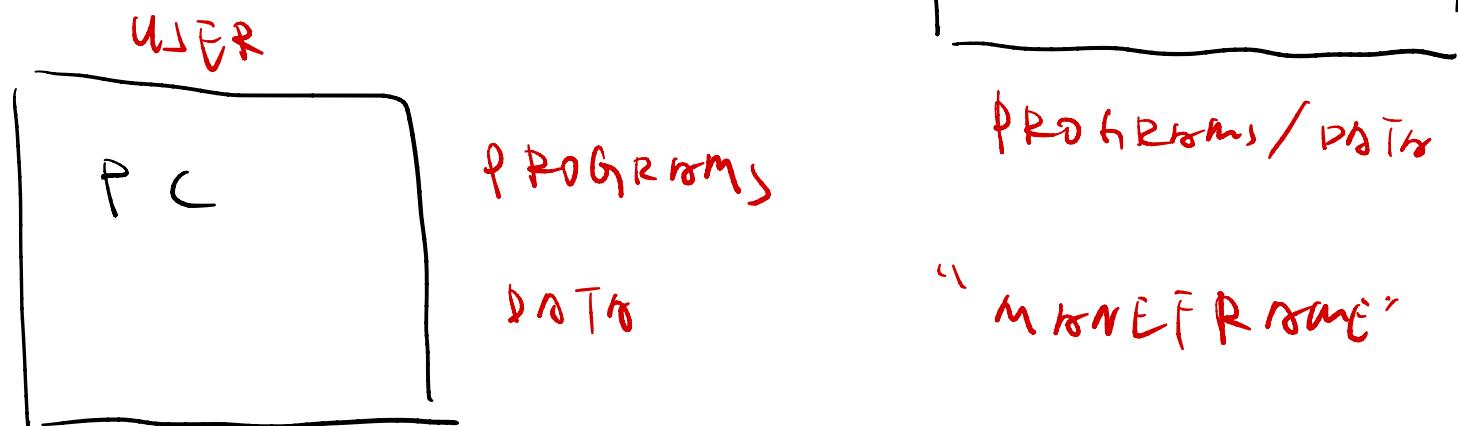
"BATCH" OF PROGRAMS AND DATA

"CLOUD COMPUTING": IS THIS A NEW CONCEPT?

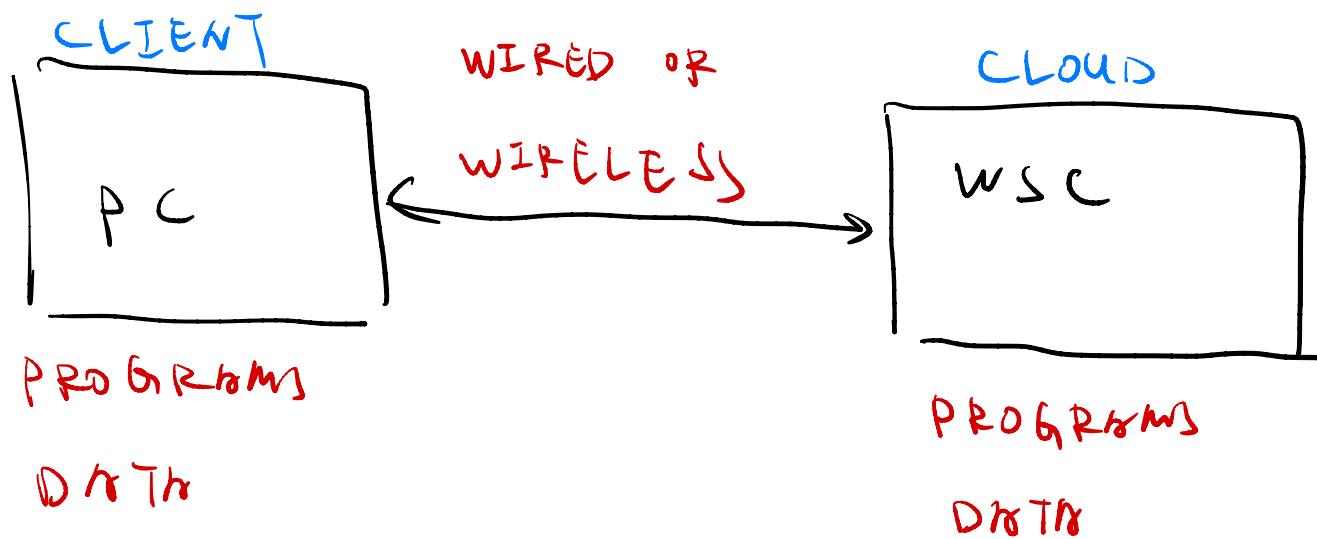
1950's - 1970's



1980's - 90's



NOW



Programming Models and Workloads

- Example:
 - **map (String key, String value):**
 - // key: document name
 - // value: document contents
 - for each word w in value
 - EmitIntermediate(w,"1"); // Produce list of all words
 - **reduce (String key, Iterator values):**
 - // key: a word
 - // value: a list of counts
 - int result = 0;
 - for each v in values:
 - result += ParseInt(v); // get integer from key-value pair
 - Emit(AsString(result));

Programming Models and Workloads

- **MapReduce runtime environment schedules map and reduce task to WSC nodes**
- **Availability:**
 - Use replicas of data across different servers
 - Use relaxed consistency:
 - No need for all replicas to always agree
- **Workload demands**
 - Often vary considerably

Outline

- 6.1 Introduction
- 6.2 Programming Models and Workloads
- 6.3 Architecture
- 6.4 Physical Infrastructure
- 6.5 Cloud Computing

Computer Architecture of WSC

- WSC often use a hierarchy of networks for interconnection
- Each 19" rack holds 48 servers connected to a rack switch
- Rack switches are uplinked to switch higher in hierarchy
 - Uplink has 6-24X times lower bandwidth
 - Goal is to maximize locality of communication relative to the rack



Storage

- **Storage options:**
 - Use disks inside the servers, or
 - Network attached storage
- WSCs generally rely on local disks
- Google File System (GFS) uses local disks and maintains at least three replicas

WSC Memory Hierarchy

- Servers can access DRAM and disks on other servers

	Local	Rack	Array
DRAM latency (μs)	0.1	300	500
Flash latency (μs)	100	400	600
Disk latency (μs)	10,000	11,000	12,000
DRAM bandwidth (MB/s)	20,000	100	10
Flash bandwidth (MB/s)	1000	100	10
Disk bandwidth (MB/s)	200	100	10
DRAM capacity (GB)	16	1024	31,200
Flash capacity (GB)	128	20,000	600,000
Disk capacity (GB)	2000	160,000	4,800,000

Outline

- 6.1 Introduction
- 6.2 Programming Models and Workloads
- 6.3 Architecture
- 6.4 Physical Infrastructure
- 6.5 Cloud Computing