

# Parallel Programming in C with MPI and OpenMP

---

Michael J. Quinn



# Chapter 2

## Parallel Architectures

---



# Outline

---

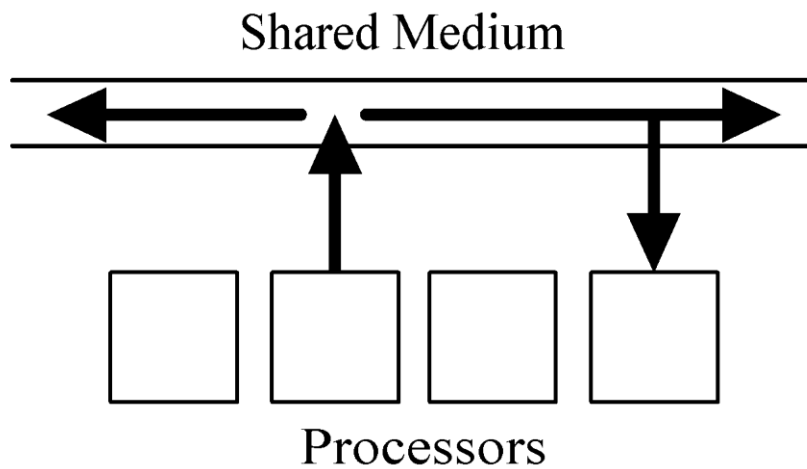
- Interconnection networks
- Processor arrays
- Multiprocessors
- Multicomputers
- Flynn's taxonomy

# Interconnection Networks

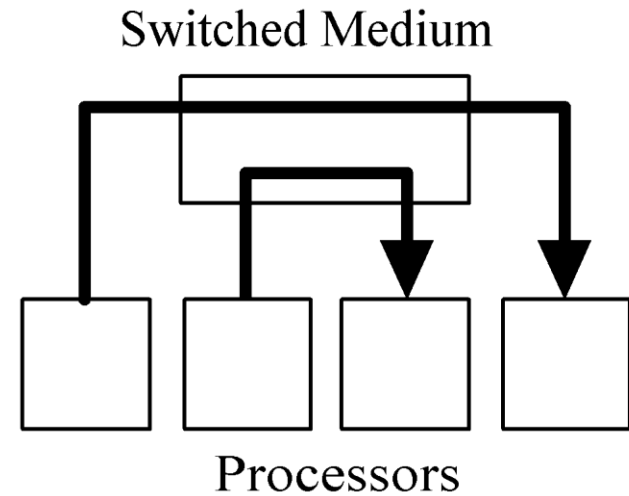
---

- Uses of interconnection networks
  - Connect processors to **shared memory**
  - Connect processors to each other
- Interconnection media types
  - **Shared** medium
  - **Switched** medium

# Shared versus Switched Media



(a)



(b)

# Shared Medium

---

- Allows only message at a time
- Messages are broadcast
- Each processor “listens” to every message
- Arbitration is decentralized
- Collisions require resending of messages
- Ethernet is an example

# Switched Medium

---

- Supports point-to-point messages between pairs of processors
- Each processor has its own path to switch
- Advantages over shared media
  - Allows **multiple messages** to be sent simultaneously
  - Allows **scaling** of network to accommodate increase in processors

# Switch Network Topologies

---

- View switched network as a graph
  - Vertices = processors or switches
  - Edges = communication paths
- Two kinds of topologies
  - Direct
  - Indirect



# Direct Topology

---

- Ratio of switch nodes to processor nodes is 1:1
- Every switch node is connected to
  - 1 processor node
  - At least 1 other switch node

# Indirect Topology

---

- Ratio of switch nodes to processor nodes is greater than 1:1
- Some switches simply connect other switches

# Evaluating Switch Topologies

---

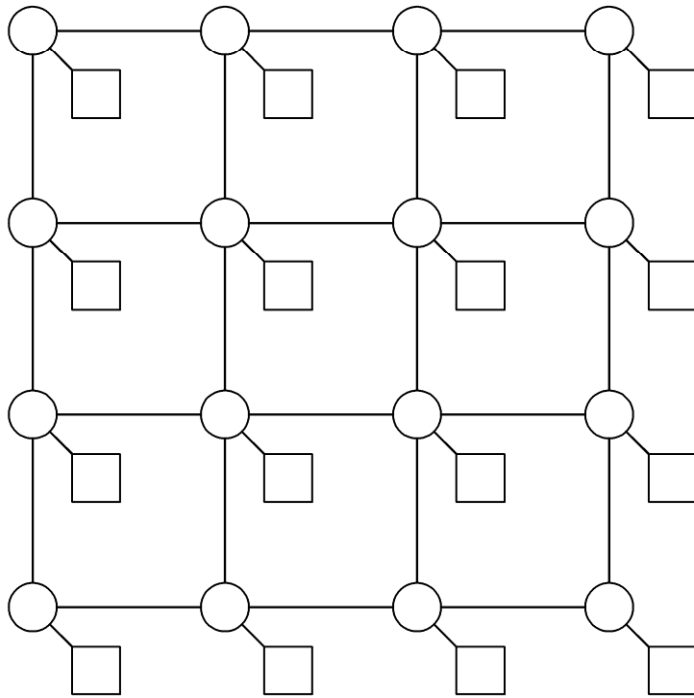
- Diameter
  - Low is better
- Bisection width
  - High is better
- Number of edges / node
  - Best for constant
- Constant edge length? (yes/no question)
  - Best for independence of network size

## 2-D Mesh Network

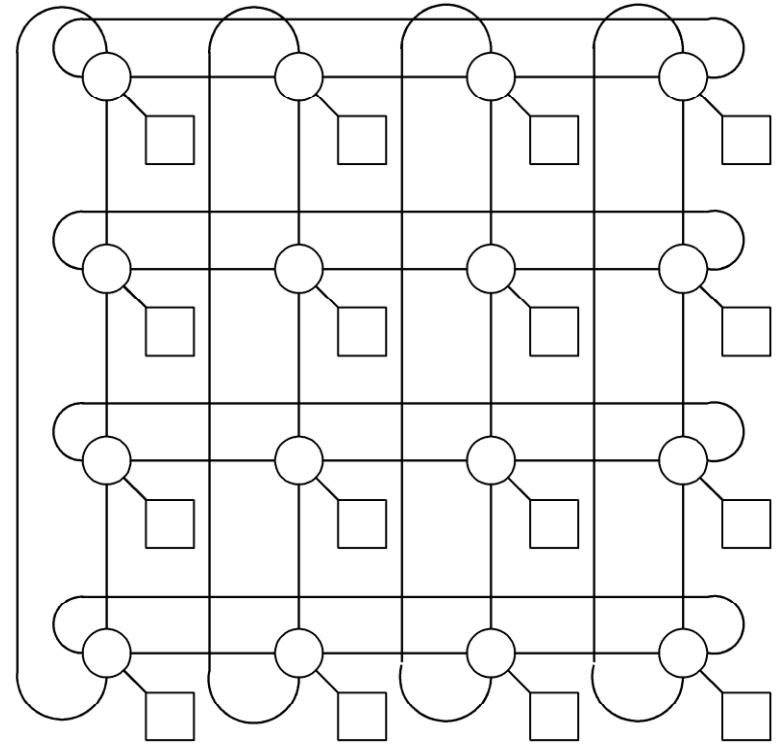
---

- Direct topology
- Switches arranged into a 2-D lattice
- Communication allowed only between neighboring switches
- Variants allow wraparound connections between switches on edge of mesh

# 2-D Meshes



(a)



(b)

With wraparound connection

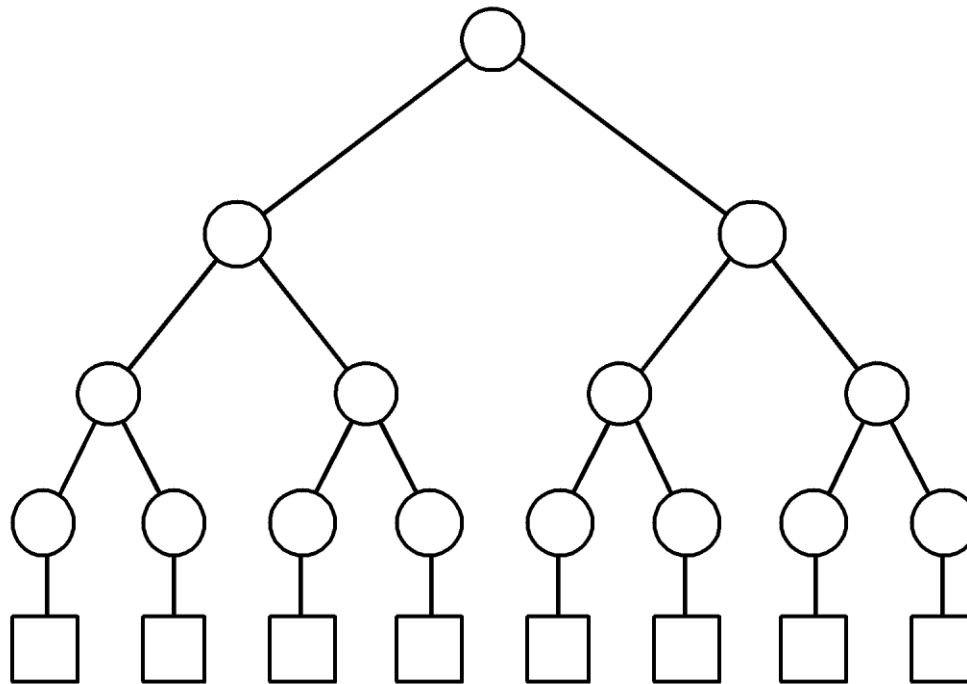
# Evaluating 2-D Meshes

---

- Diameter:  $\Theta(n^{1/2})$
- Bisection width:  $\Theta(n^{1/2})$
- Number of edges per switch: 4
- Constant edge length? Yes

# Binary Tree Network

- Indirect topology
- $n = 2^d$  processor nodes,  $2n-1$  switches



# Evaluating Binary Tree Network

---

- Diameter:  $2 \log n$
- Bisection width: 1
- Edges / node: 3
- Constant edge length? No

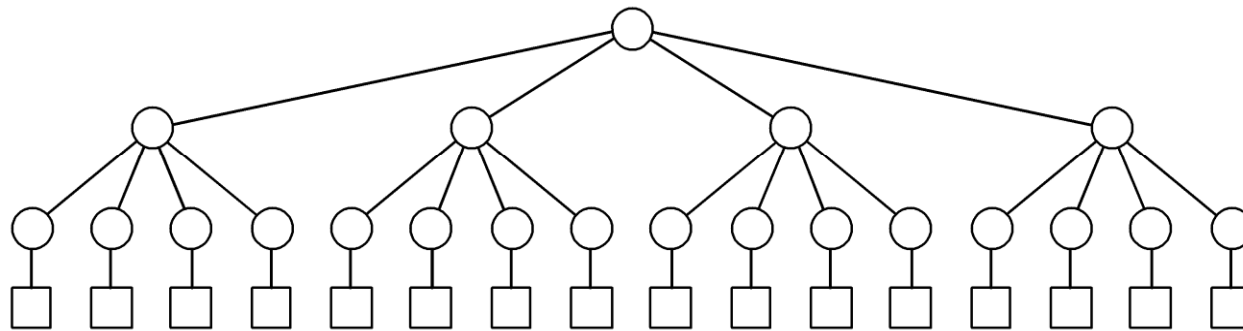


# Hypertree Network

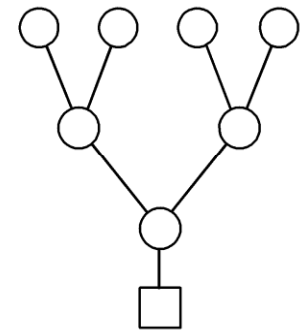
---

- Indirect topology
- Shares low diameter of binary tree
- Greatly improves bisection width
- From “front” looks like  $k$ -ary tree of height  $d$
- From “side” looks like upside down binary tree of height  $d$

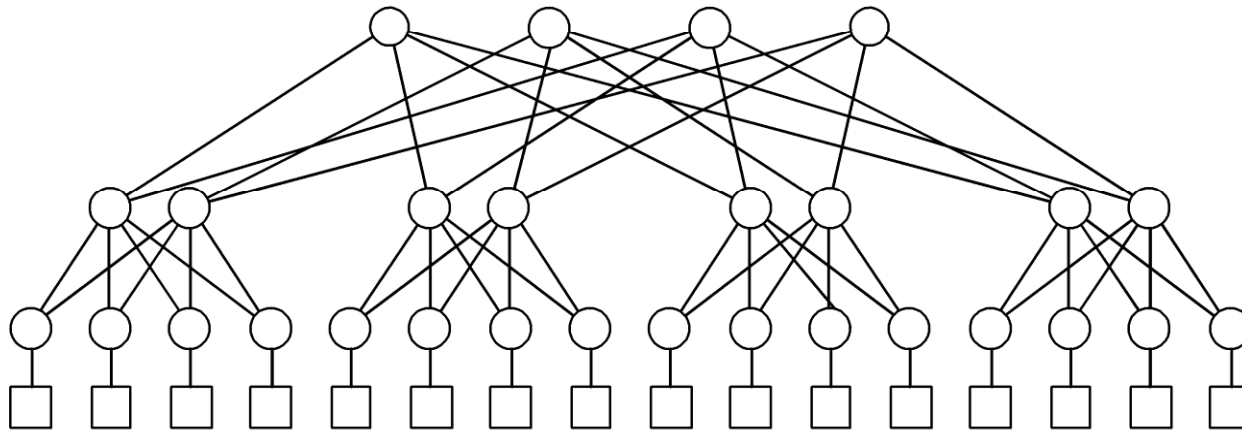
# Hypertree Network



(a) Front view



(b) Side view



(c) Complete network

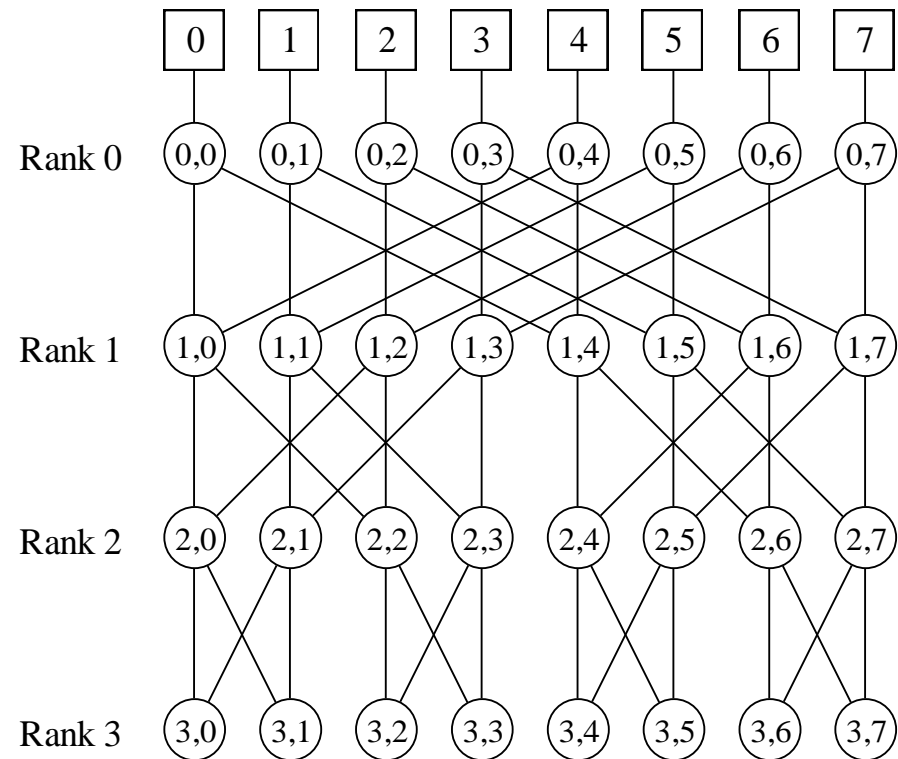
# Evaluating 4-ary Hypertree

---

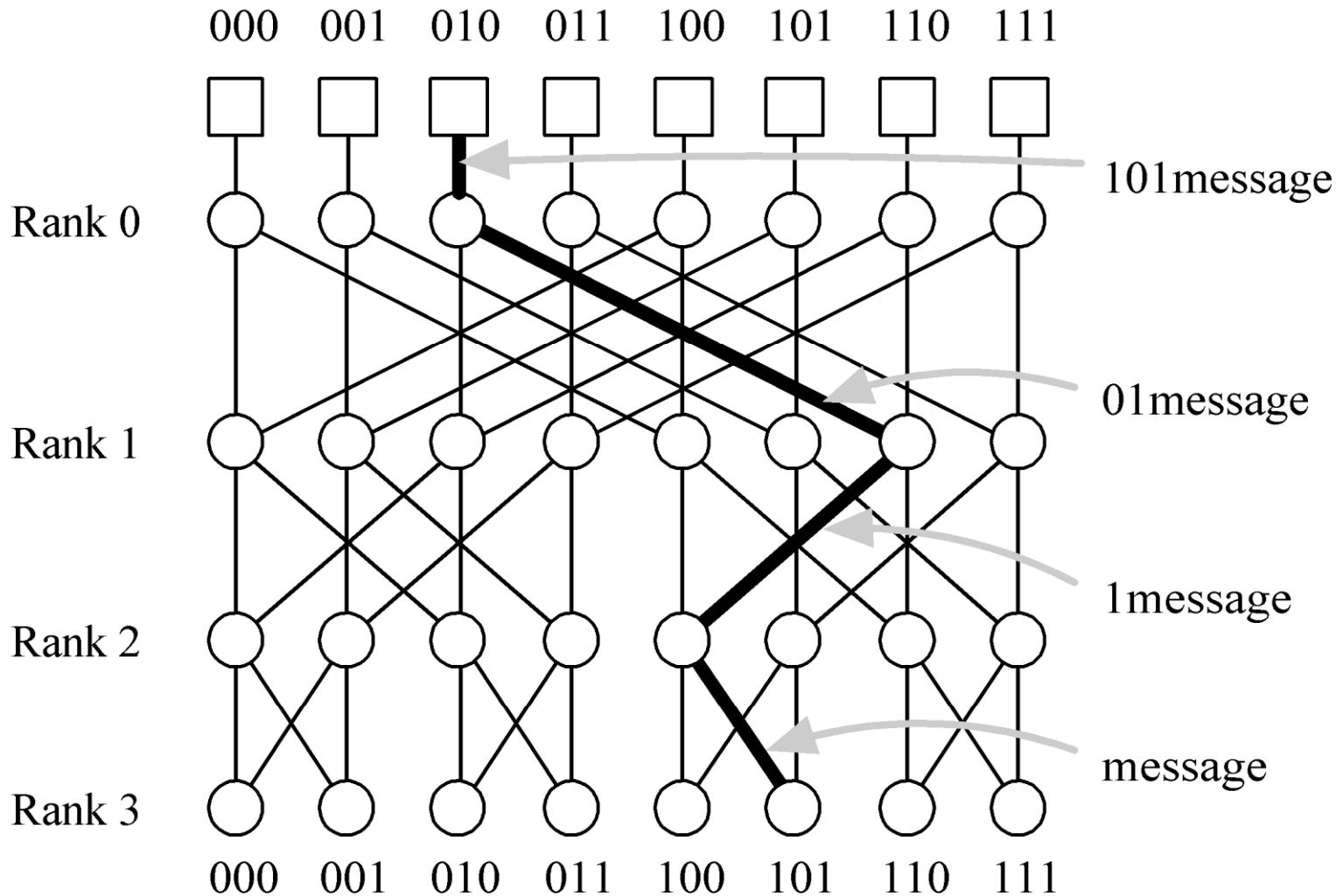
- Diameter:  $2d=2\log n$
- Bisection width:  $2^{d+1}=2n$
- Edges / node: 6
- Constant edge length? No

# Butterfly Network

- Indirect topology
- $n = 2^d$  processor nodes connected by  $n(\log n + 1)$  switching nodes



# Butterfly Network Routing



# Evaluating Butterfly Network

---

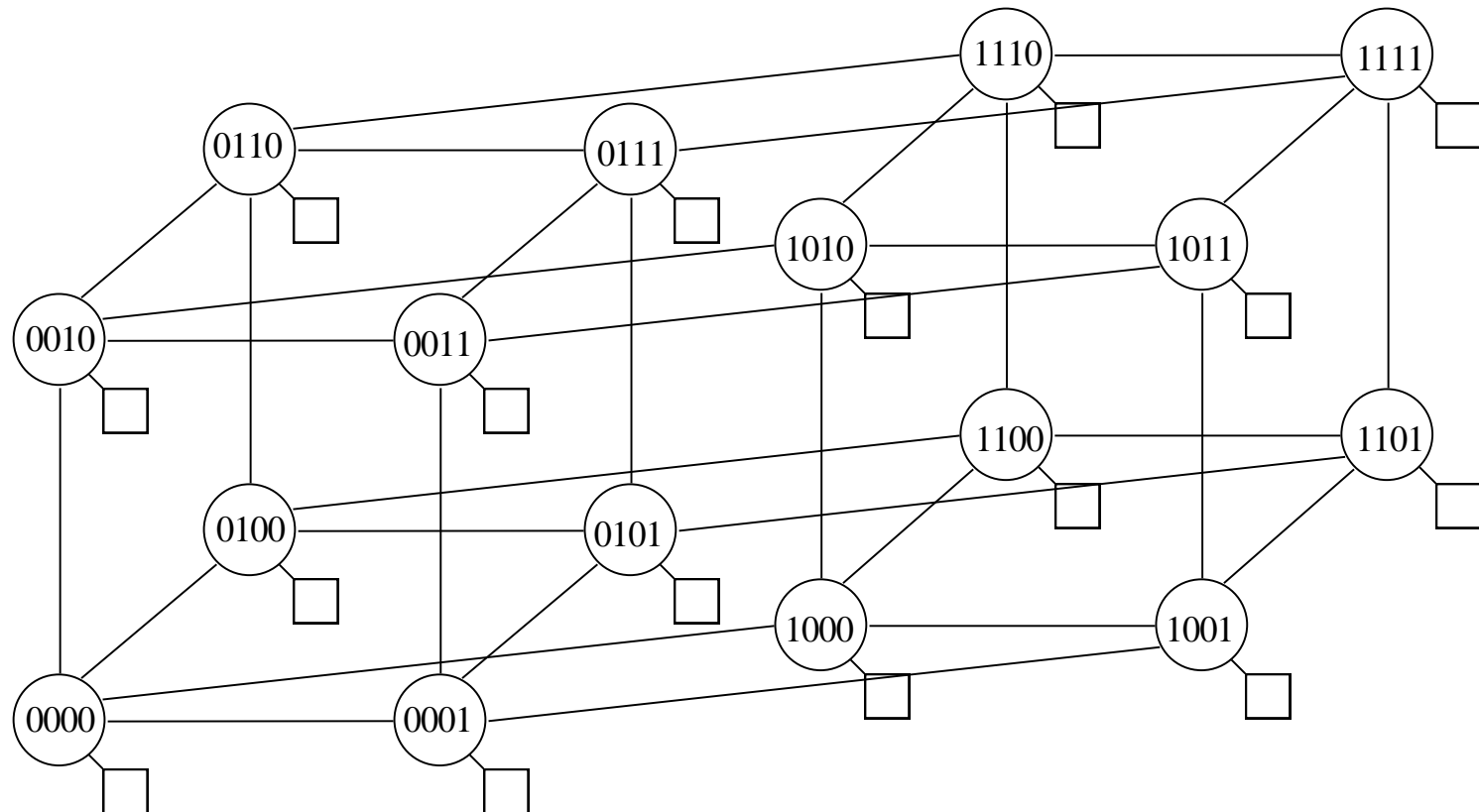
- Diameter:  $\log n$
- Bisection width:  $n$  (ref. errata)
- Edges per node: 4
- Constant edge length? No

# Hypercube

---

- Direct topology
- $2 \times 2 \times \dots \times 2$  mesh
- Number of nodes a power of 2
- Node addresses  $0, 1, \dots, 2^k-1$
- Node  $i$  connected to  $k$  nodes whose addresses differ from  $i$  in exactly one bit position

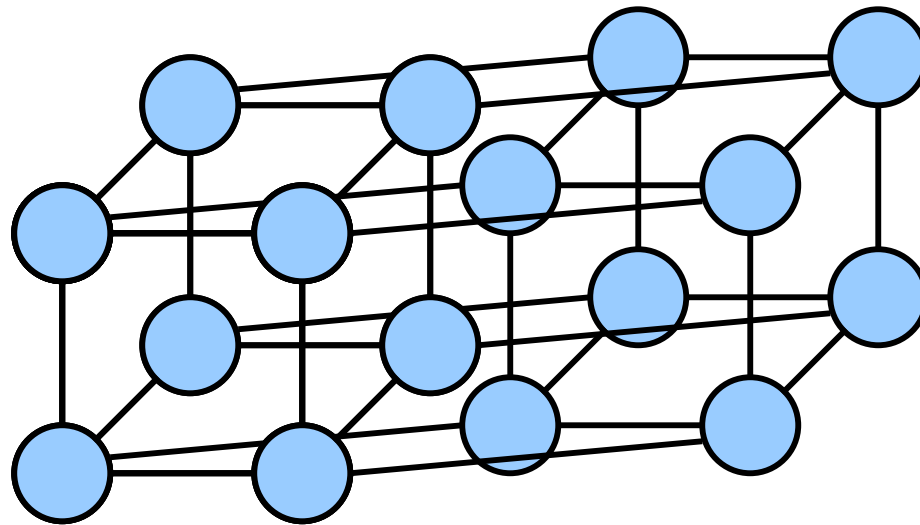
# Hypercube Addressing





# Hypercubes Illustrated

---



# Evaluating Hypercube Network

---

- Diameter:  $\log n$
- Bisection width:  $n / 2$
- Edges per node:  $\log n$
- Constant edge length? No

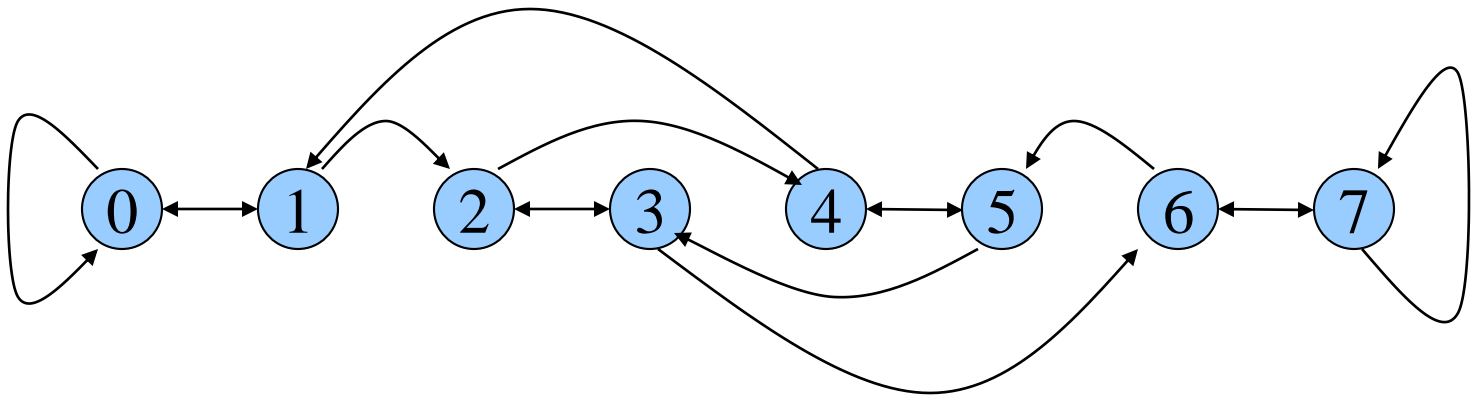
# Shuffle-exchange

---

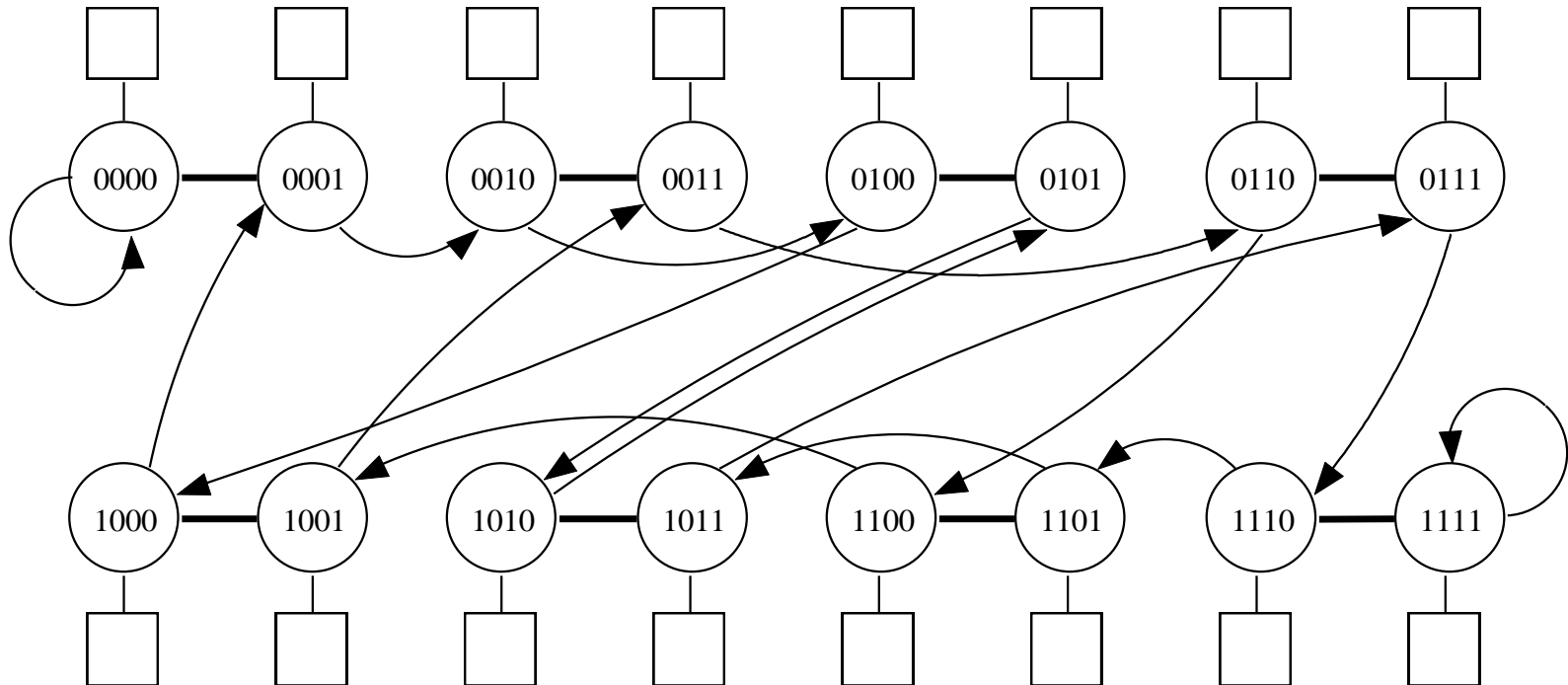
- Direct topology
- Number of nodes a power of 2
- Nodes have addresses  $0, 1, \dots, 2^k-1$
- Two outgoing links from node  $i$ 
  - Shuffle link to node  $LeftCycle(i)$
  - Exchange link to node  $[xor(i, 1)]$

# Shuffle-exchange Illustrated

---



# Shuffle-exchange Addressing



# Evaluating Shuffle-exchange

---

- Diameter:  $2\log n - 1$
- Bisection width:  $\approx n / \log n$
- Edges per node: 2
- Constant edge length? No

# Comparing Networks

---

- All have logarithmic diameter except 2-D mesh
- Hypertree, butterfly, and hypercube have bisection width  $n / 2$
- All have constant edges per node except hypercube
- Only 2-D mesh keeps edge lengths constant as network size increases

# Summary

	Processor nodes	Switch nodes	Diameter	Bisection width	<u>Edges</u> nodes	Constant edge length
2 - D mesh	$n = d^2$	$n$	$2(\sqrt{n} - 1)$	$\sqrt{n}$	4	<i>Yes</i>
Binary tree	$n = 2^d$	$2n - 1$	$2 \log n$	1	3	<i>No</i>
4 - ary hypertree	$n = 4^d$	$2n - \sqrt{n}$	$\log n$	$n/2$	6	<i>No</i>
Butterfly	$n = 4^d$	$n(\log n + 1)$	$\log n$	$n$	4	<i>No</i>
Hypercube	$n = 2^d$	$n$	$\log n$	$n/2$	$\log n$	<i>No</i>
Shuffle - exchange	$n = 2^d$	$n$	$2 \log n - 1$	$\approx n/\log n$	2	<i>No</i>



# Vector Computers

---

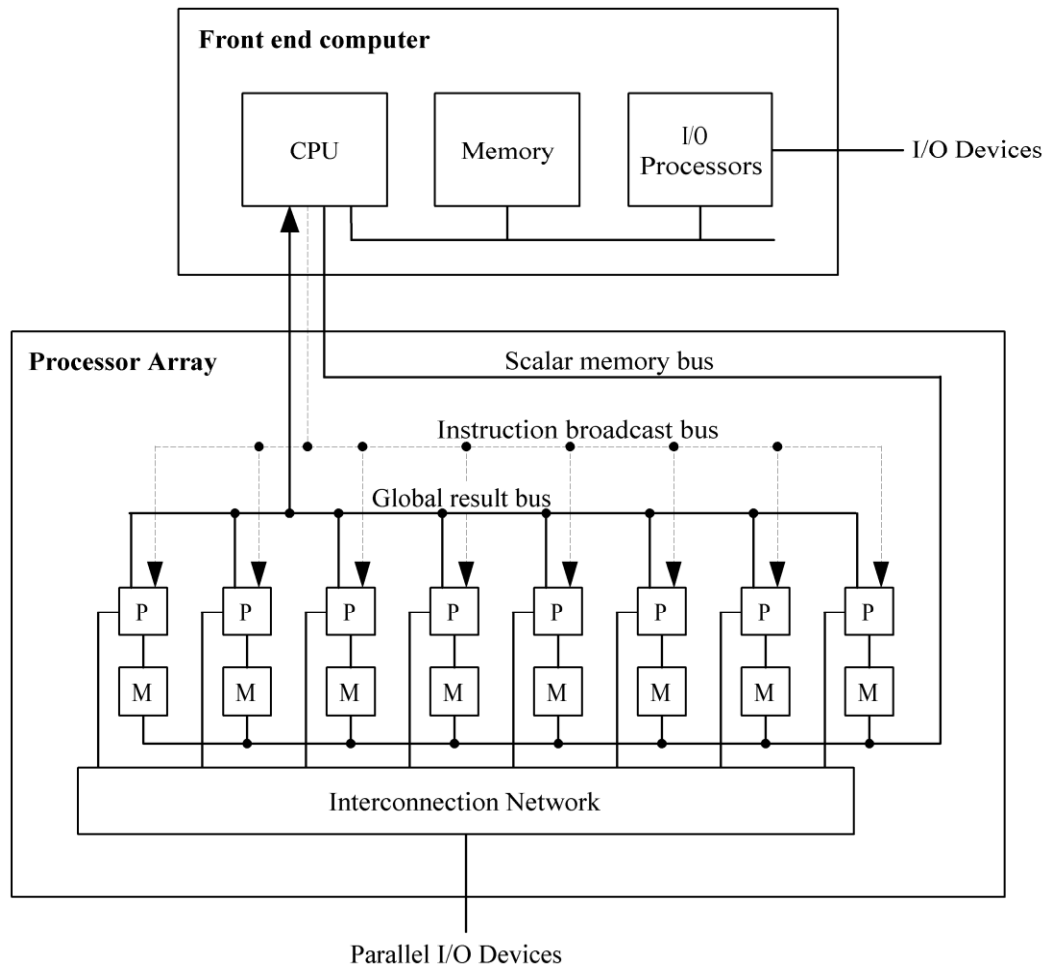
- Vector computer: instruction set includes operations on **vectors** as well as scalars
- Two ways to implement vector computers
  - **Pipelined** vector processor: streams data through pipelined arithmetic units
  - Processor **array**: many identical, synchronized arithmetic processing elements

# Why Processor Arrays?

---

- Historically, high cost of a control unit
- Scientific applications have data parallelism

# Processor Array



# Data/Instruction Storage

---

- Front end computer
  - Program
  - Data manipulated sequentially
- Processor array
  - Data manipulated in parallel

# Processor Array Performance

---

- Performance: work done per time unit
- Performance of processor array
  - Speed of processing elements
  - Utilization of processing elements

# Performance Example 1

---

- 1024 processors
- Each adds a pair of integers in 1  $\mu$ sec
- What is performance when adding two 1024-element vectors (one per processor)?

$$\text{Performance} = \frac{1024 \text{ operations}}{1 \mu\text{sec}} = 1.024 \times 10^9 \text{ ops/sec}$$

## Performance Example 2

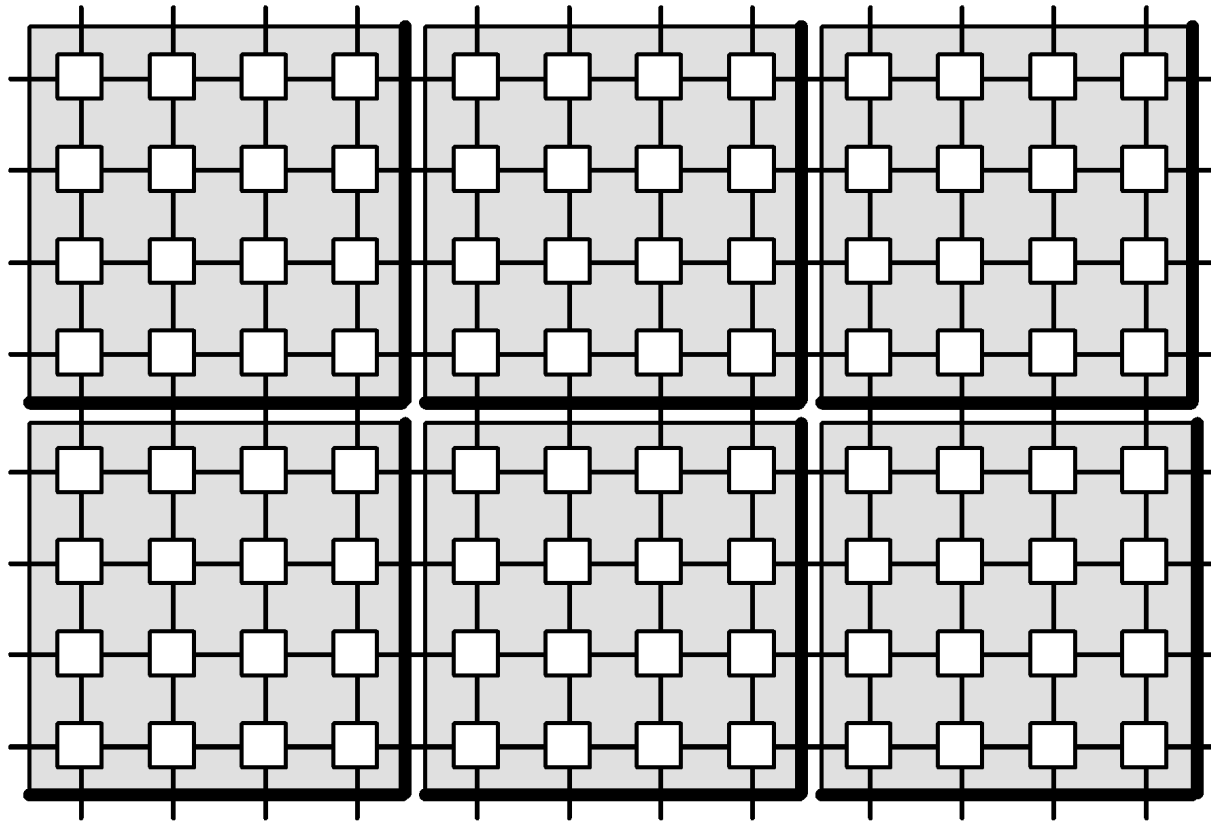
---

- 512 processors
- Each adds two integers in 1  $\mu\text{sec}$
- Performance adding two vectors of length 600?

$$\text{Performance} = \frac{600 \text{ operations}}{2 \mu\text{sec}} = 3 \times 10^6 \text{ ops/sec}$$

# 2-D Processor Interconnection Network

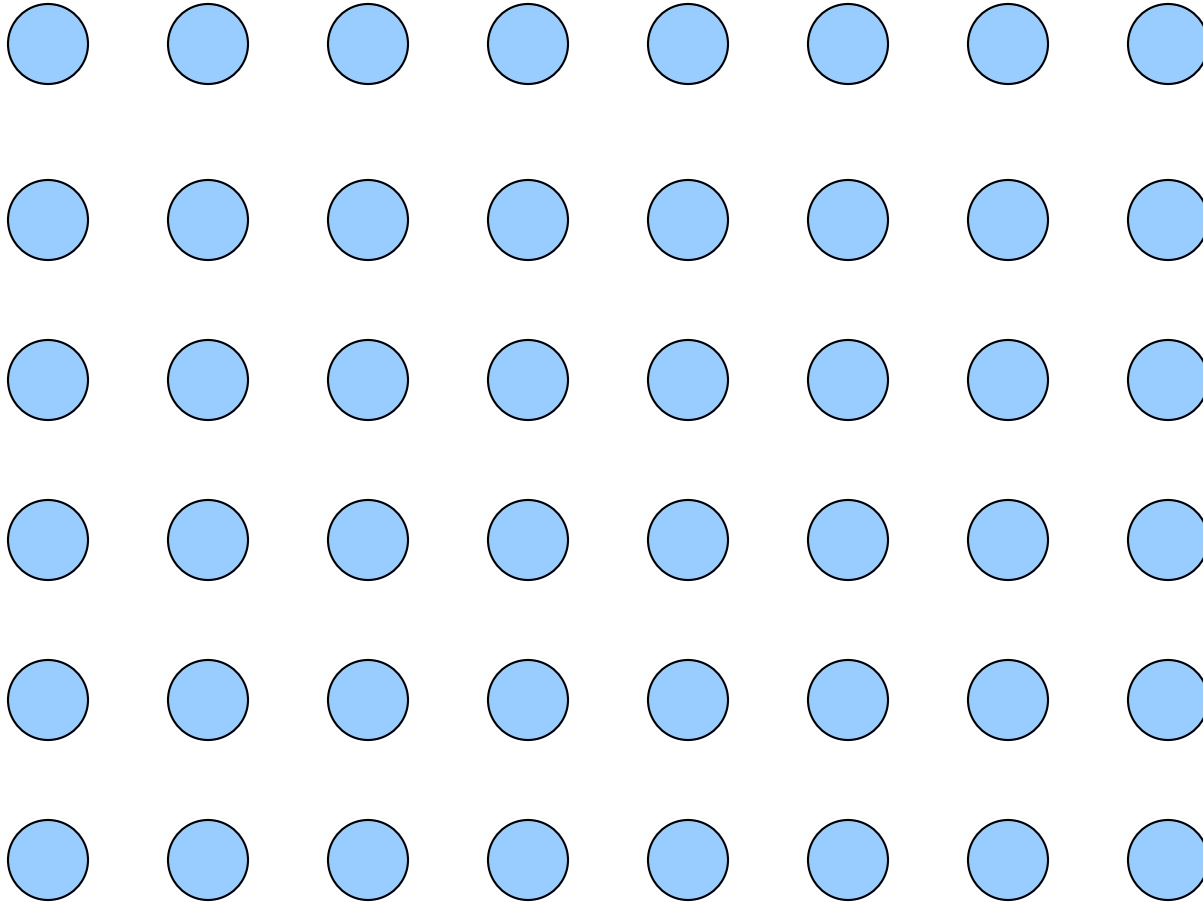
- Each VLSI chip has 16 processing elements





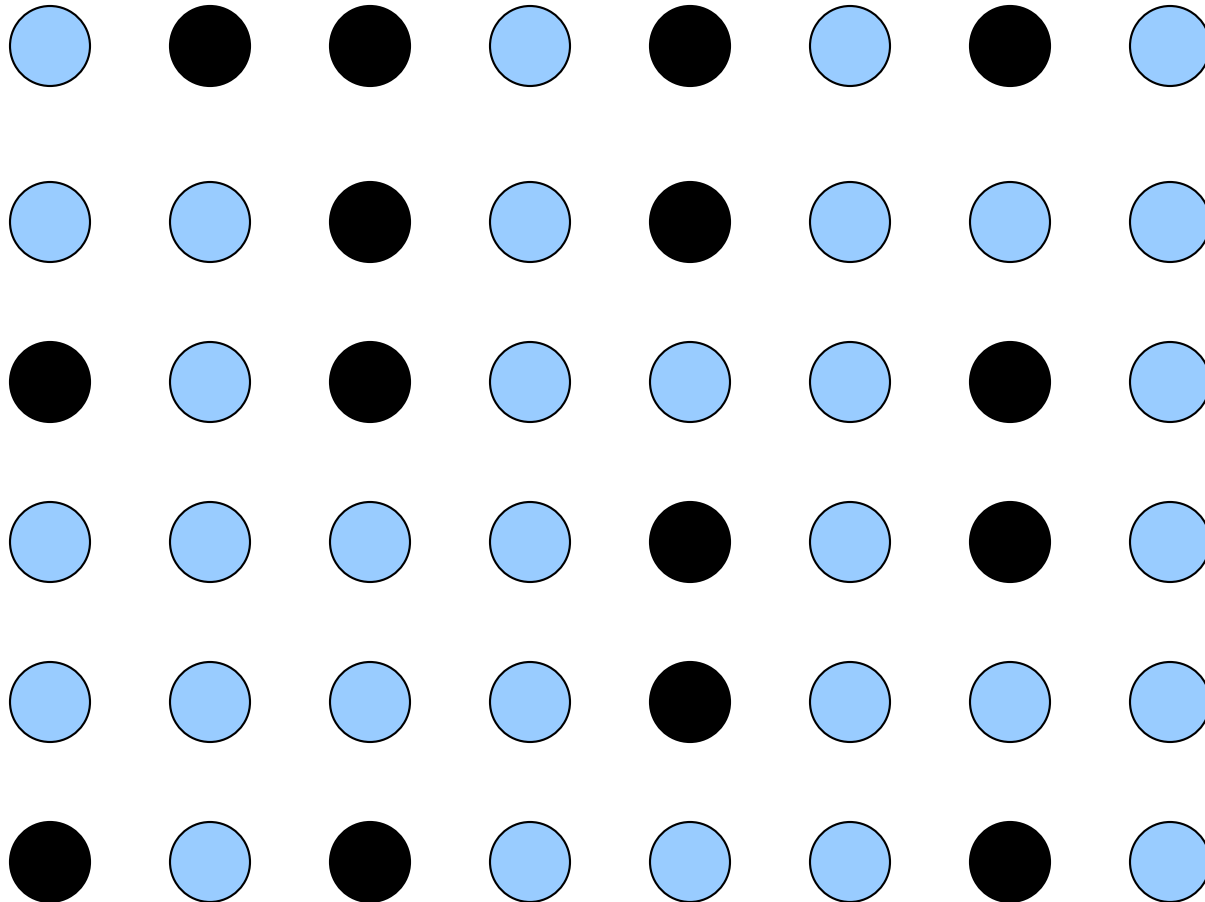
# if (**COND**) then A else B

---



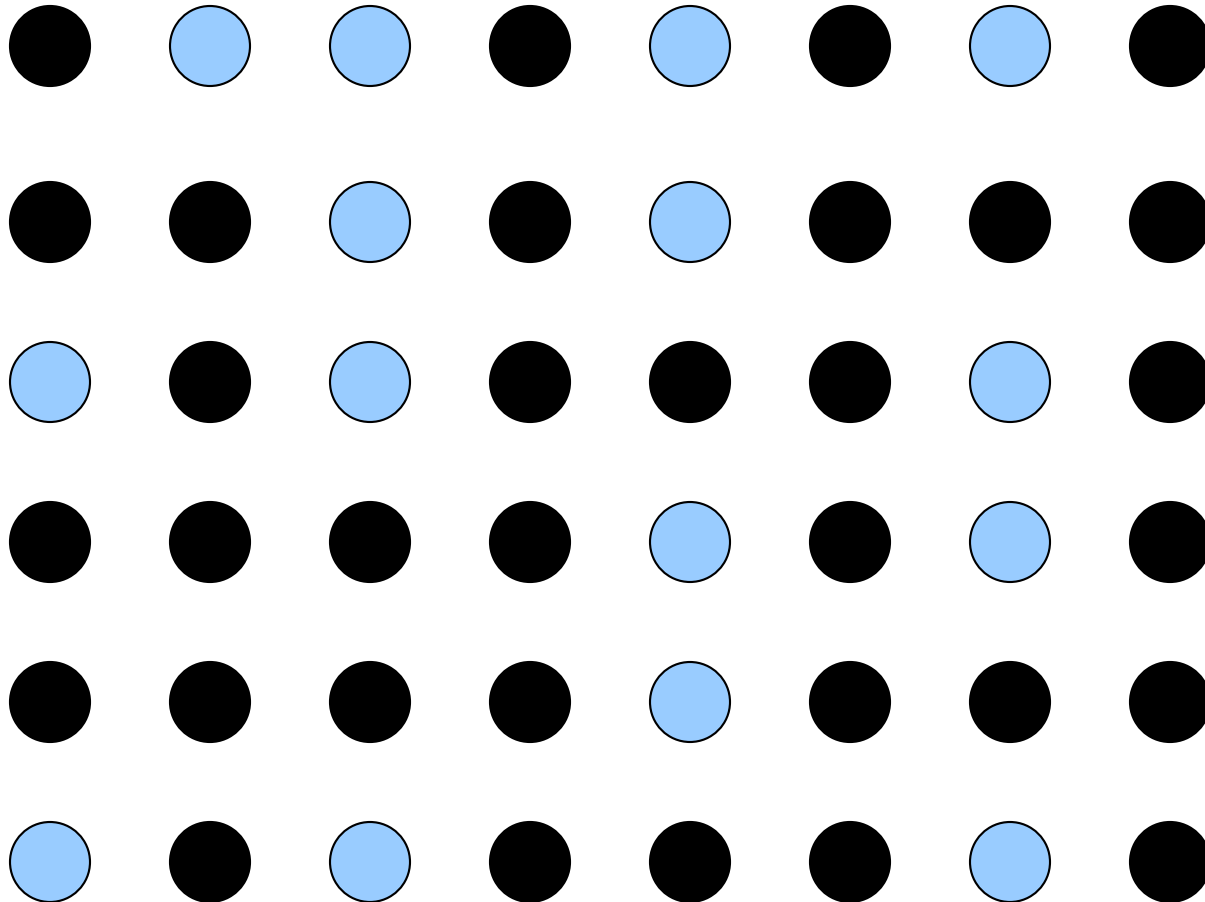
# if (COND) then **A** else B

---



# if (COND) then A else B

---



# Processor Array Shortcomings

---

- Not all problems are data-parallel
- Speed drops for conditionally executed code
- Don't adapt to multiple users well
- Do not scale down well to “starter” systems
- Rely on custom VLSI for processors
- Expense of control units has dropped

# Multiprocessors

---

- Multiprocessor: multiple-CPU computer with a shared memory
- Same address on two different CPUs refers to the same memory location
- Avoid three problems of processor arrays
  - Can be built from commodity CPUs
  - Naturally support multiple users
  - Maintain efficiency in conditional code

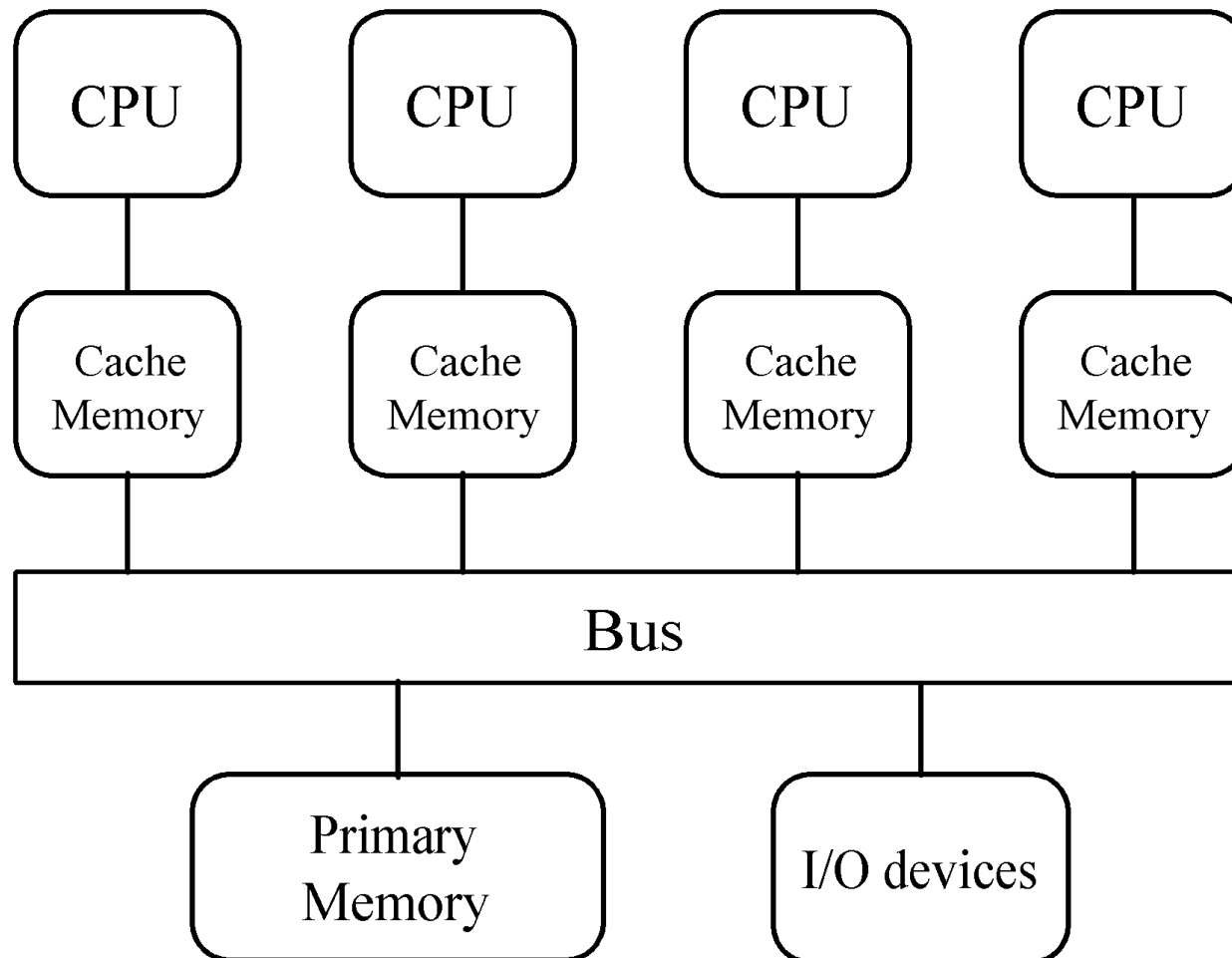
# Centralized Multiprocessor

---

- Straightforward extension of uniprocessor
- Add CPUs to bus
- All processors share same primary memory
- Memory access time same for all CPUs
  - Uniform memory access (UMA) multiprocessor
  - Symmetrical multiprocessor (SMP)

# Centralized Multiprocessor

---



# Private and Shared Data

---

- Private data: items used only by a single processor
- Shared data: values used by multiple processors
- In a multiprocessor, processors communicate via shared data values



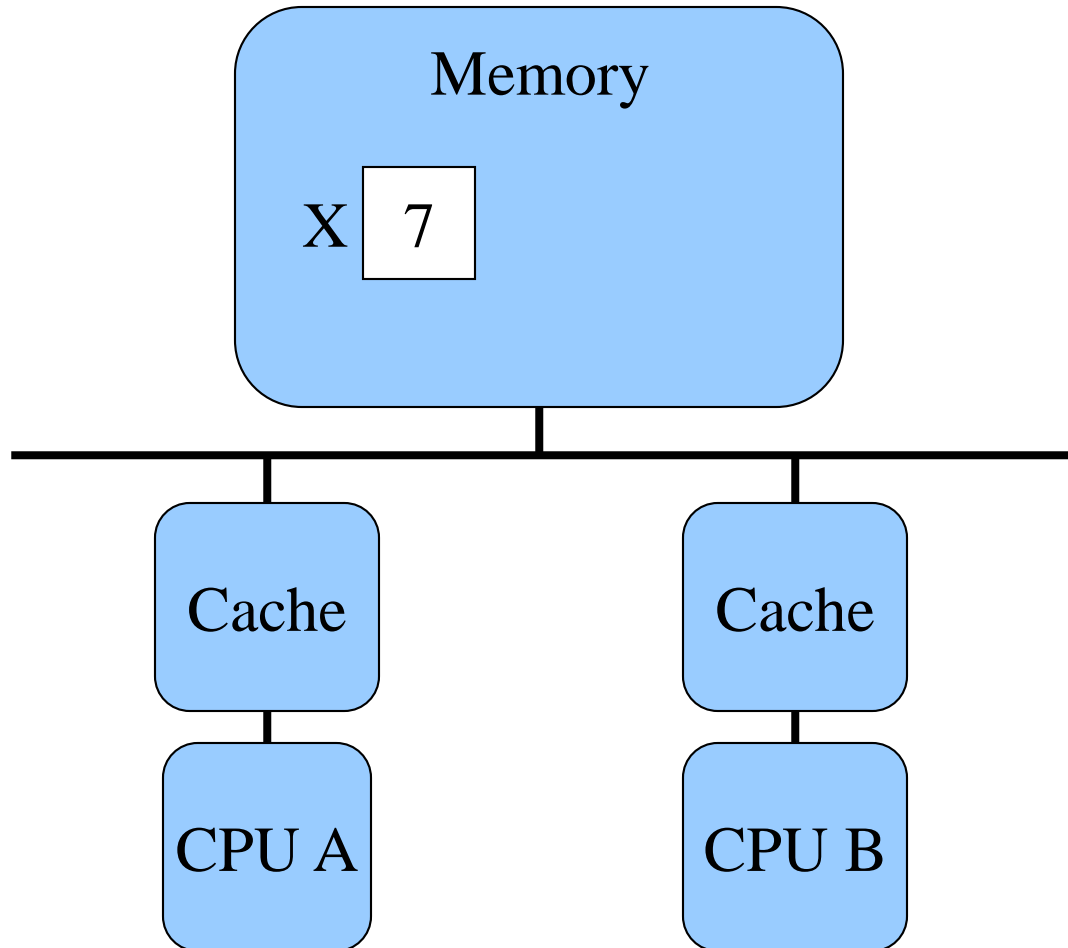
# Problems Associated with Shared Data

---

- Cache coherence
  - Replicating data across multiple caches reduces contention
  - How to ensure different processors have same value for same address?
- Synchronization
  - Mutual exclusion
  - Barrier

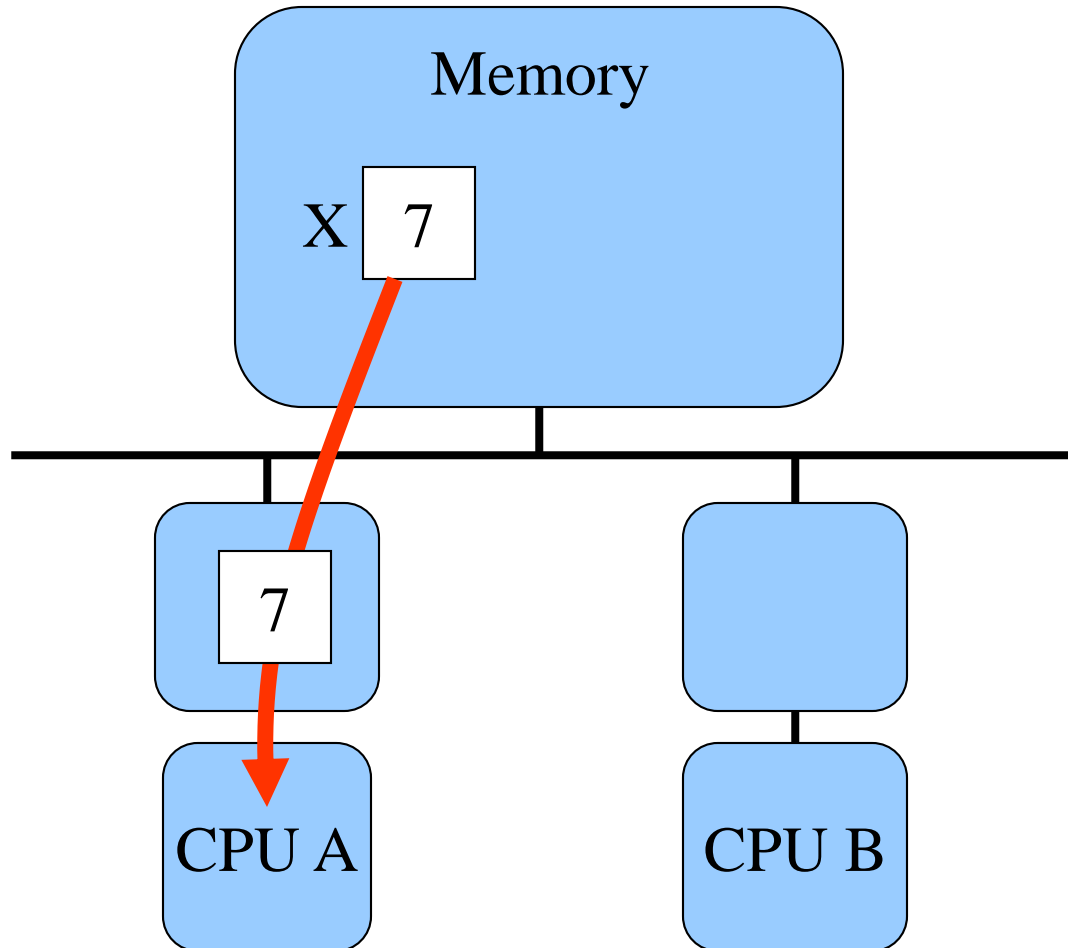
# Cache-coherence Problem (1)

---



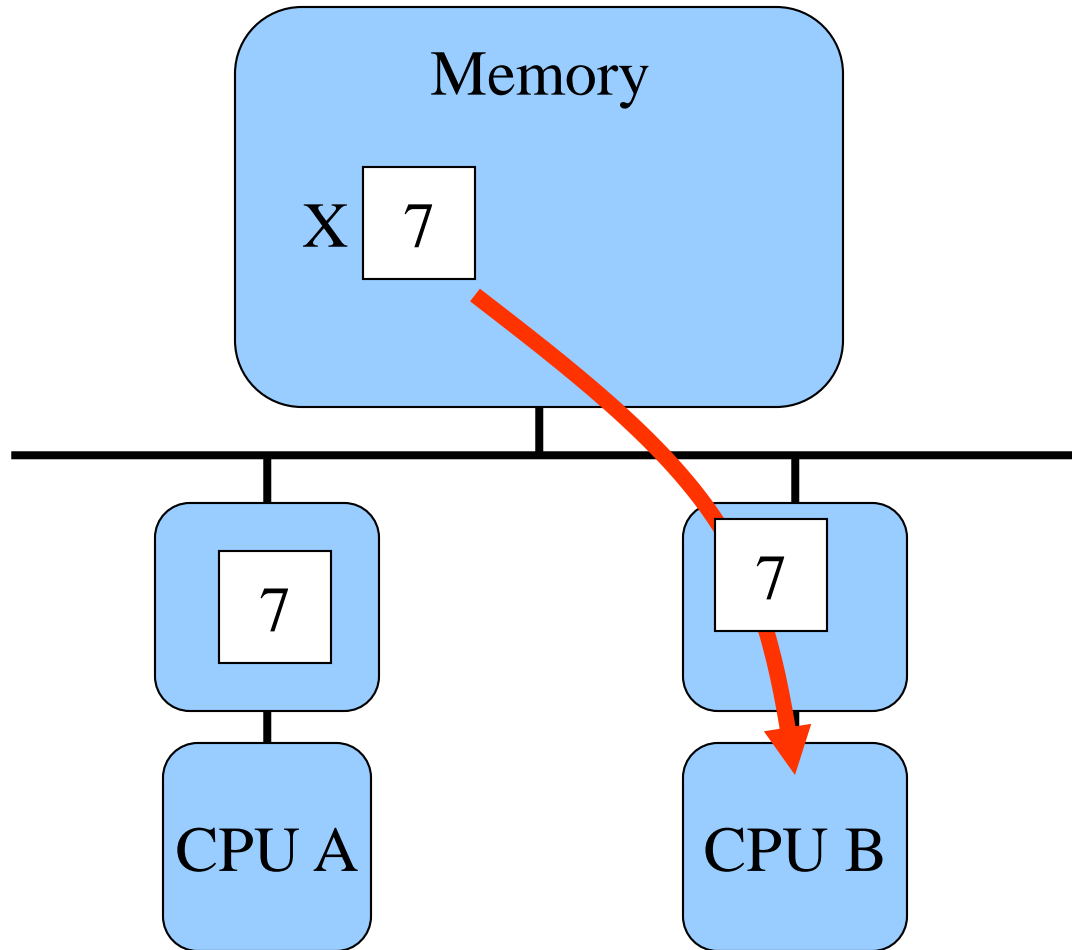
# Cache-coherence Problem (2)

---

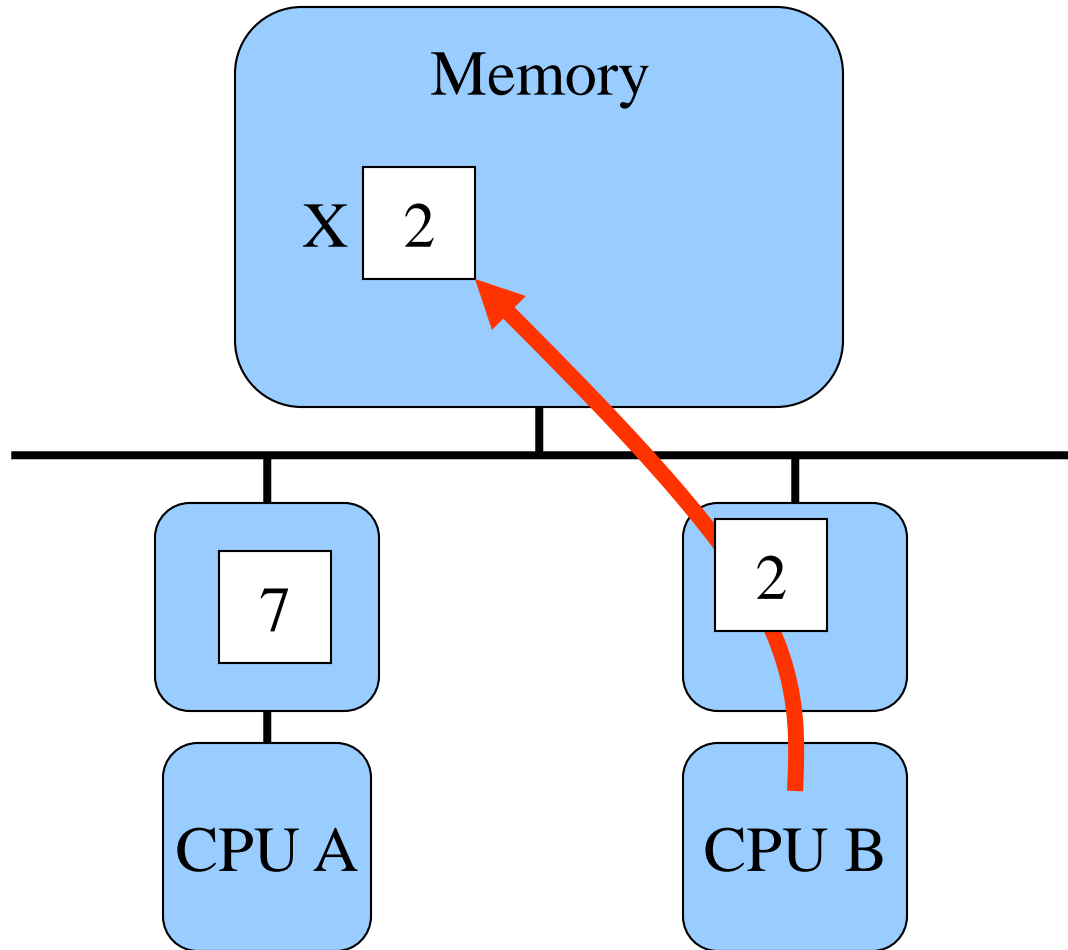


# Cache-coherence Problem (3)

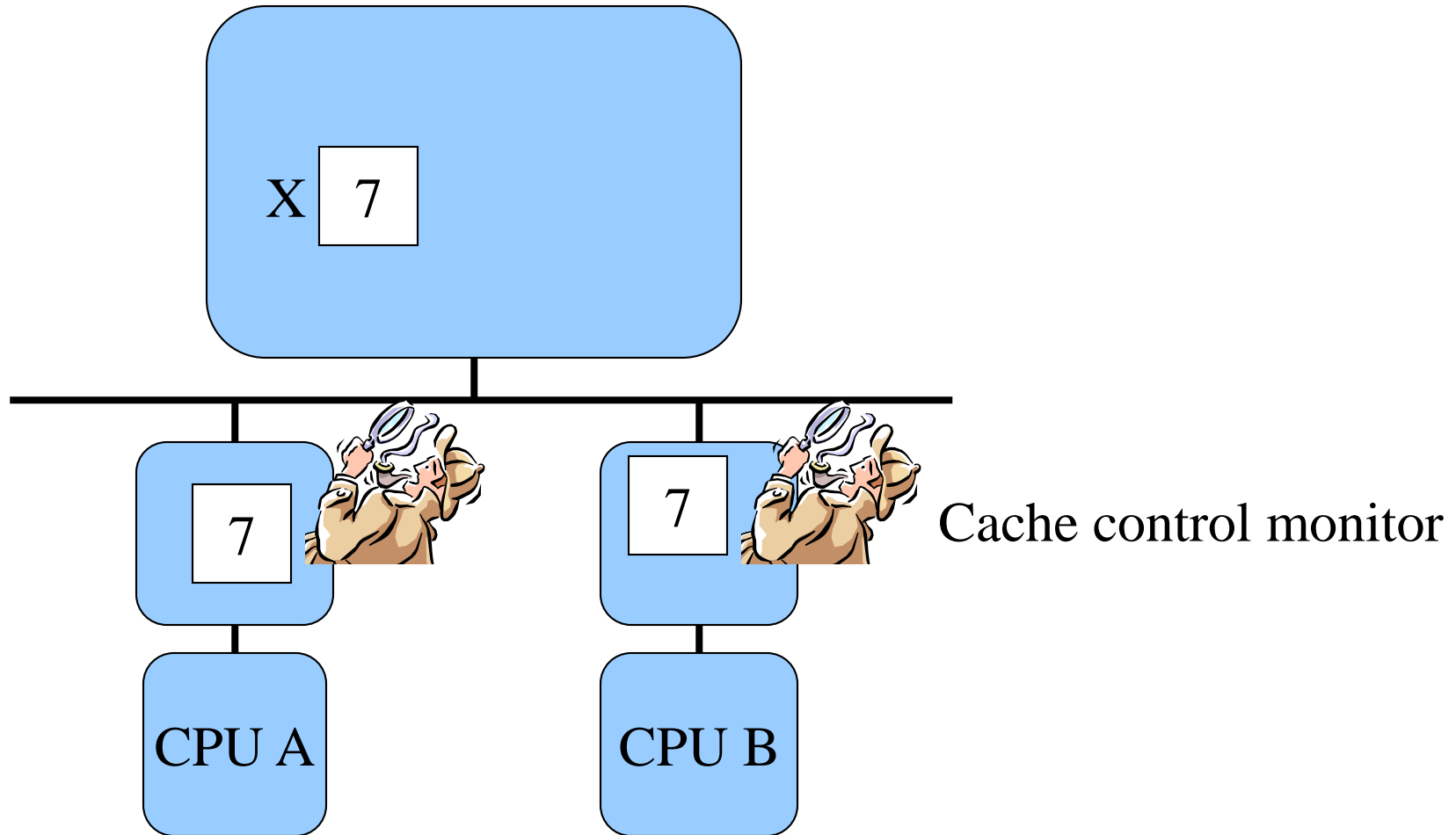
---



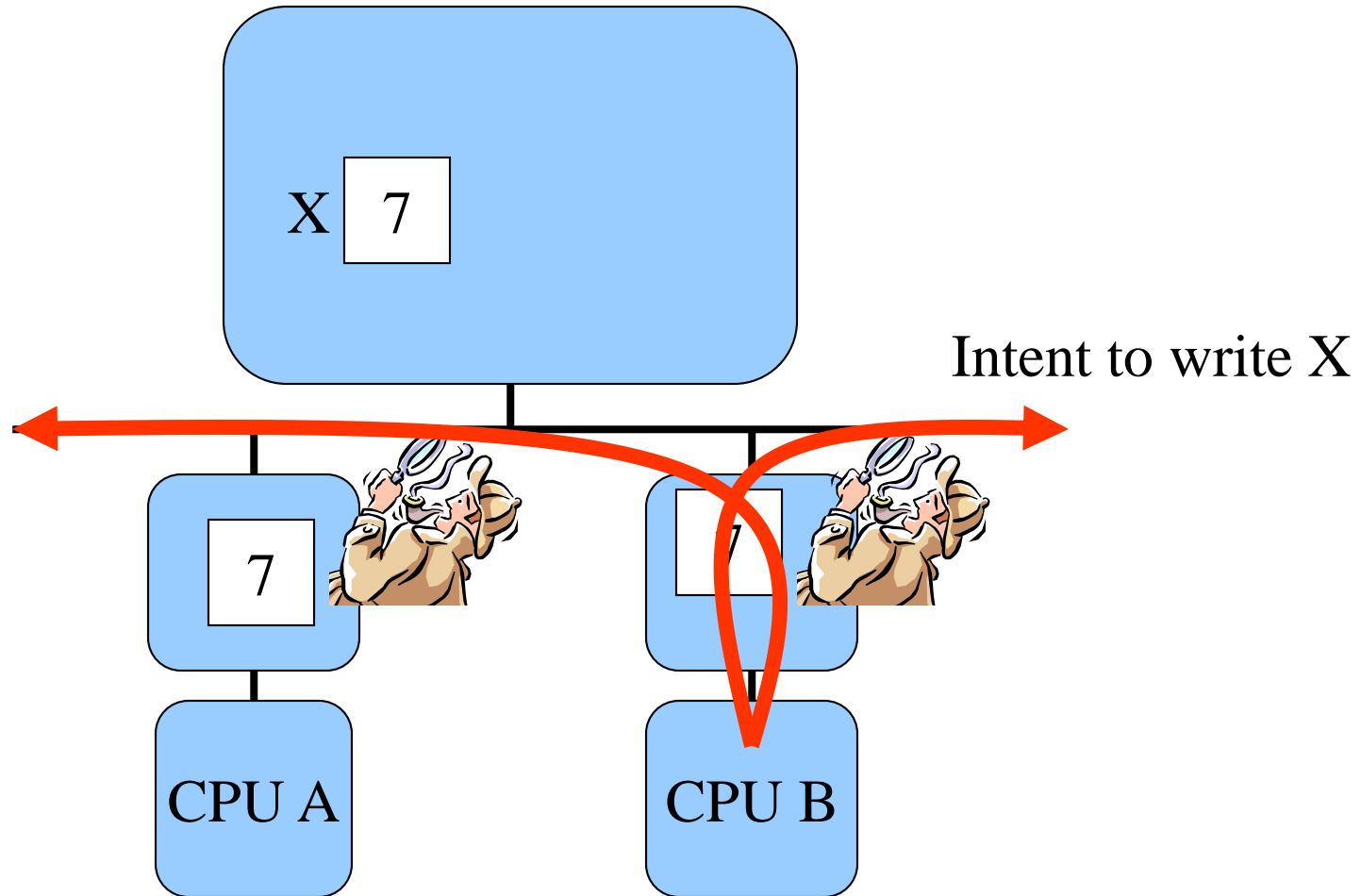
# Cache-coherence Problem (3)



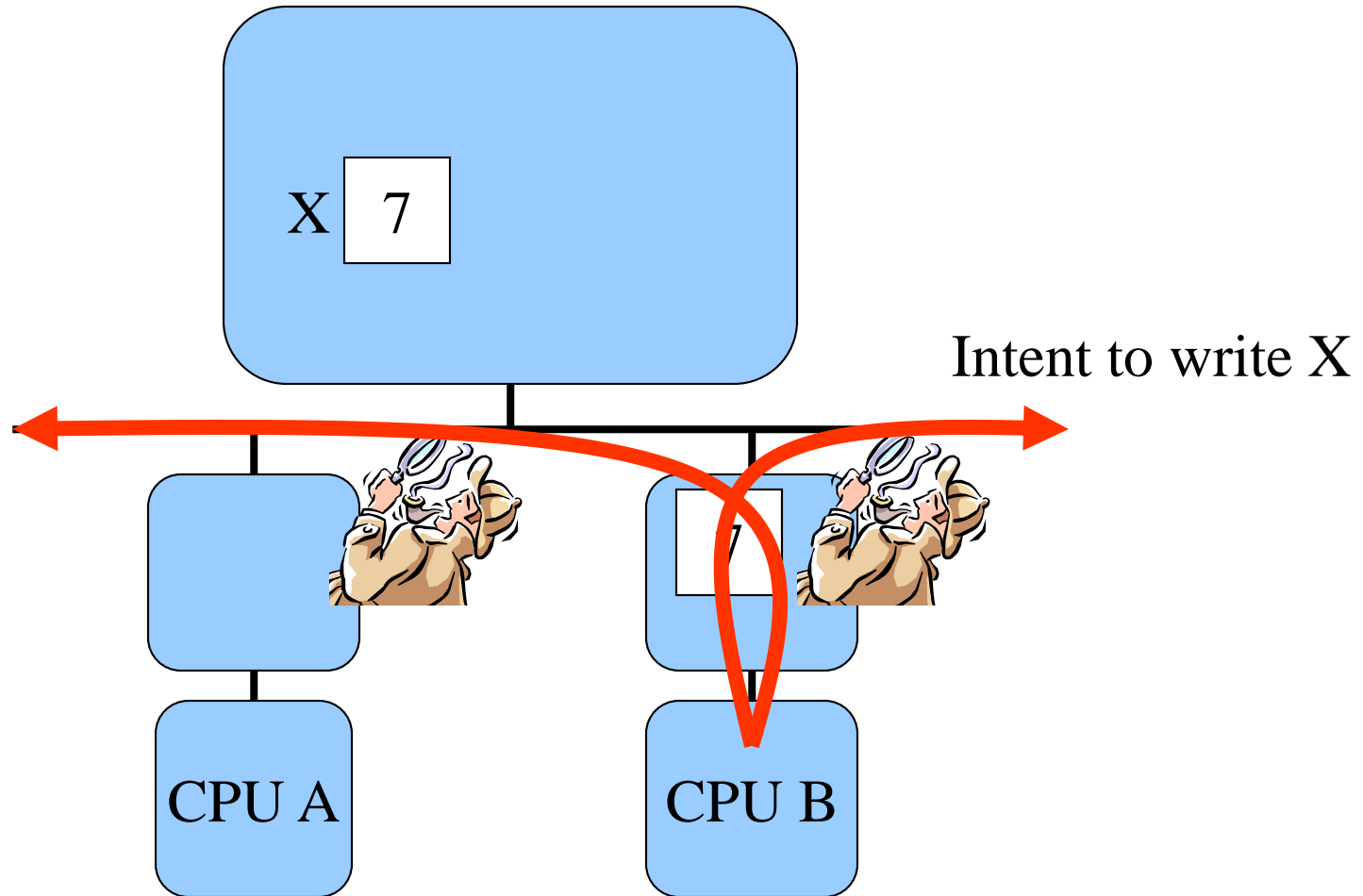
# Write Invalidate Protocol (1)



# Write Invalidate Protocol (2)

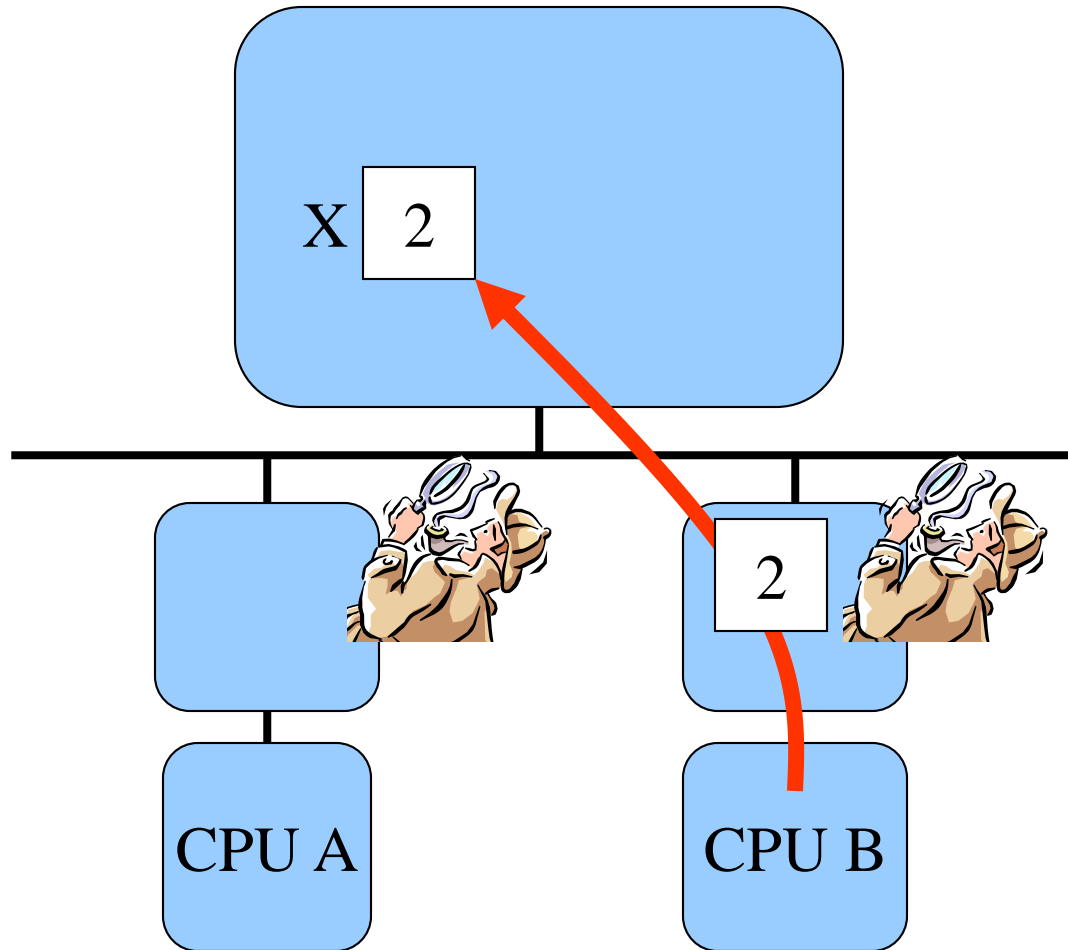


# Write Invalidate Protocol (3)





# Write Invalidate Protocol (4)

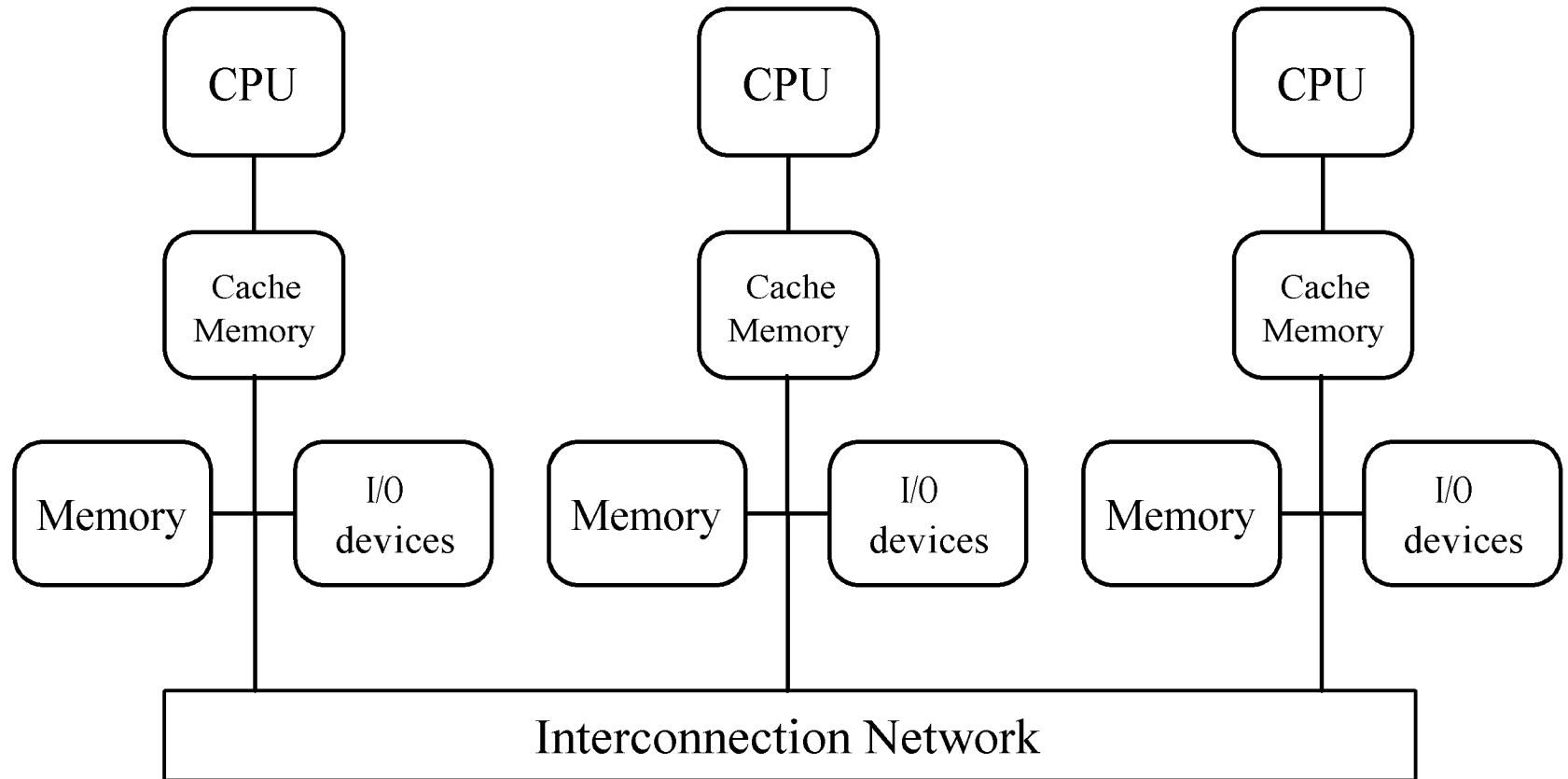


# Distributed Multiprocessor

---

- Distribute primary memory among processors
- Increase aggregate memory bandwidth and lower average memory access time
- Allow greater number of processors
- Also called non-uniform memory access (NUMA) multiprocessor

# Distributed Multiprocessor



# Cache Coherence

---

- Some NUMA multiprocessors do not support it in hardware
  - Only instructions, private data in cache
  - Large memory access time variance
- Implementation more difficult
  - No shared memory bus to “snoop”
  - Directory-based protocol needed

# Directory-based Protocol

---

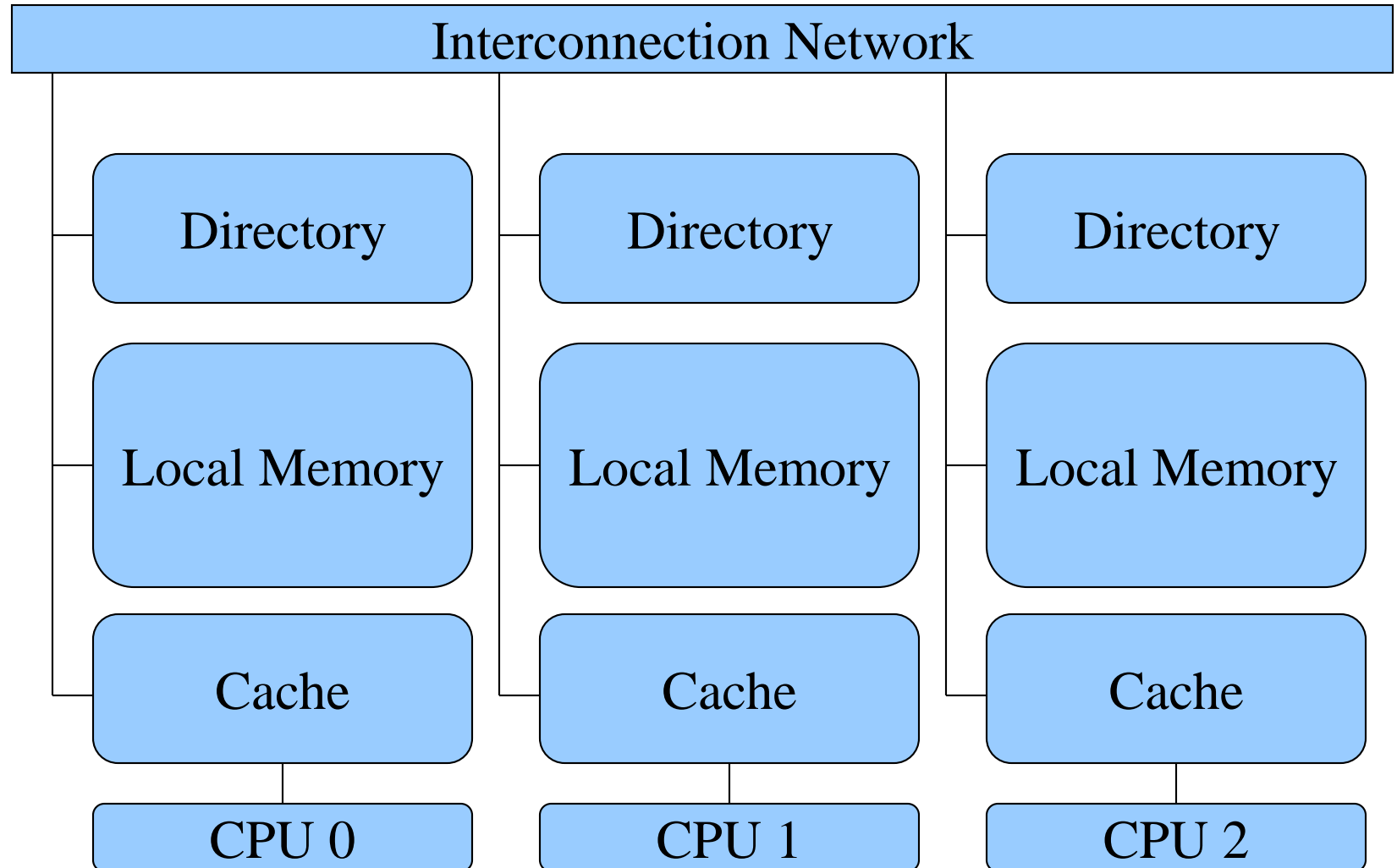
- Distributed directory contains information about cacheable memory blocks
- One directory entry for each cache block
- Each entry has
  - Sharing status
  - Which processors have copies

# Sharing Status

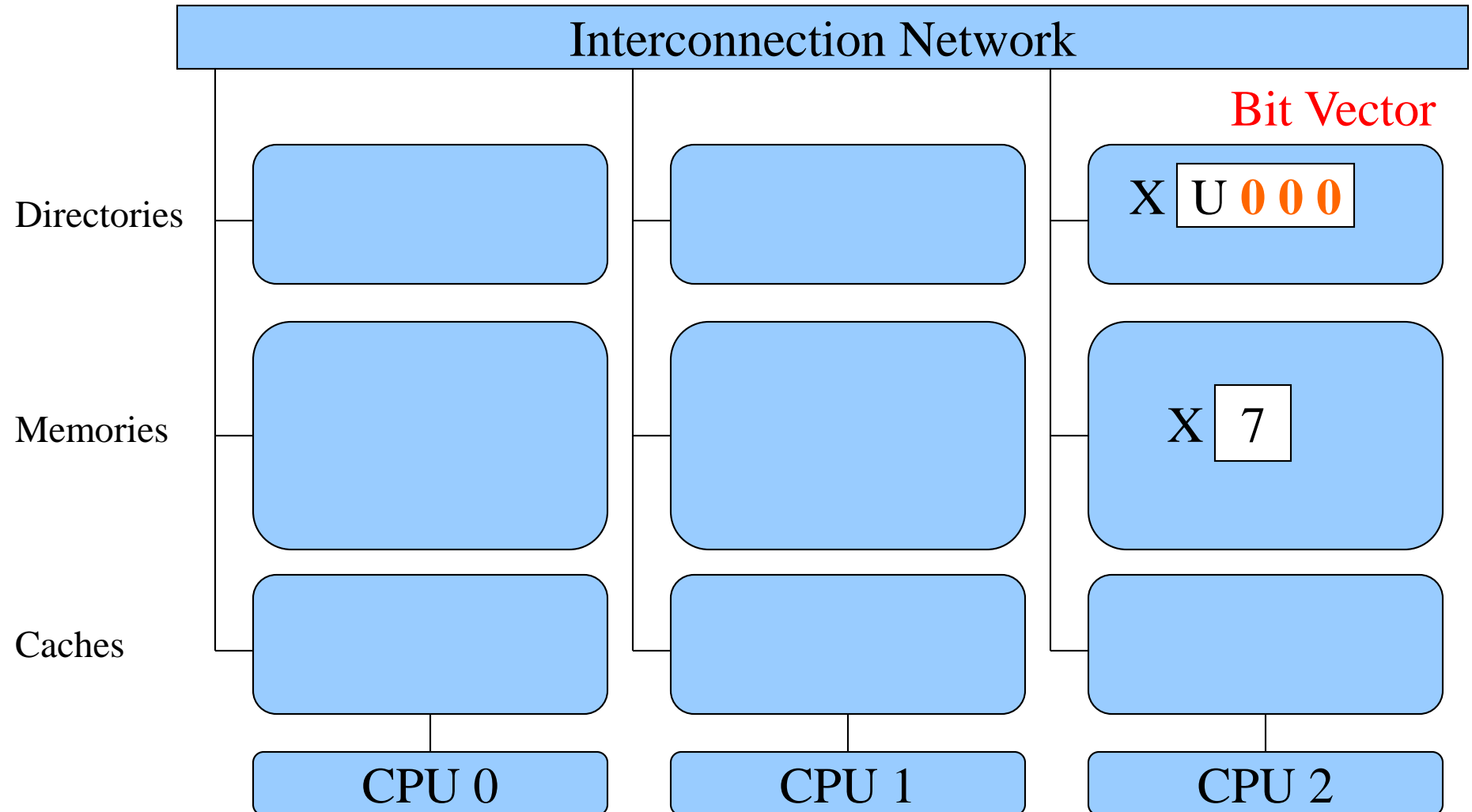
---

- Uncached
  - Block not in any processor's cache
- Shared
  - Cached by one or more processors
  - Read only
- Exclusive
  - Cached by exactly one processor
  - Processor has written block
  - Copy in memory is obsolete

# Directory-based Protocol

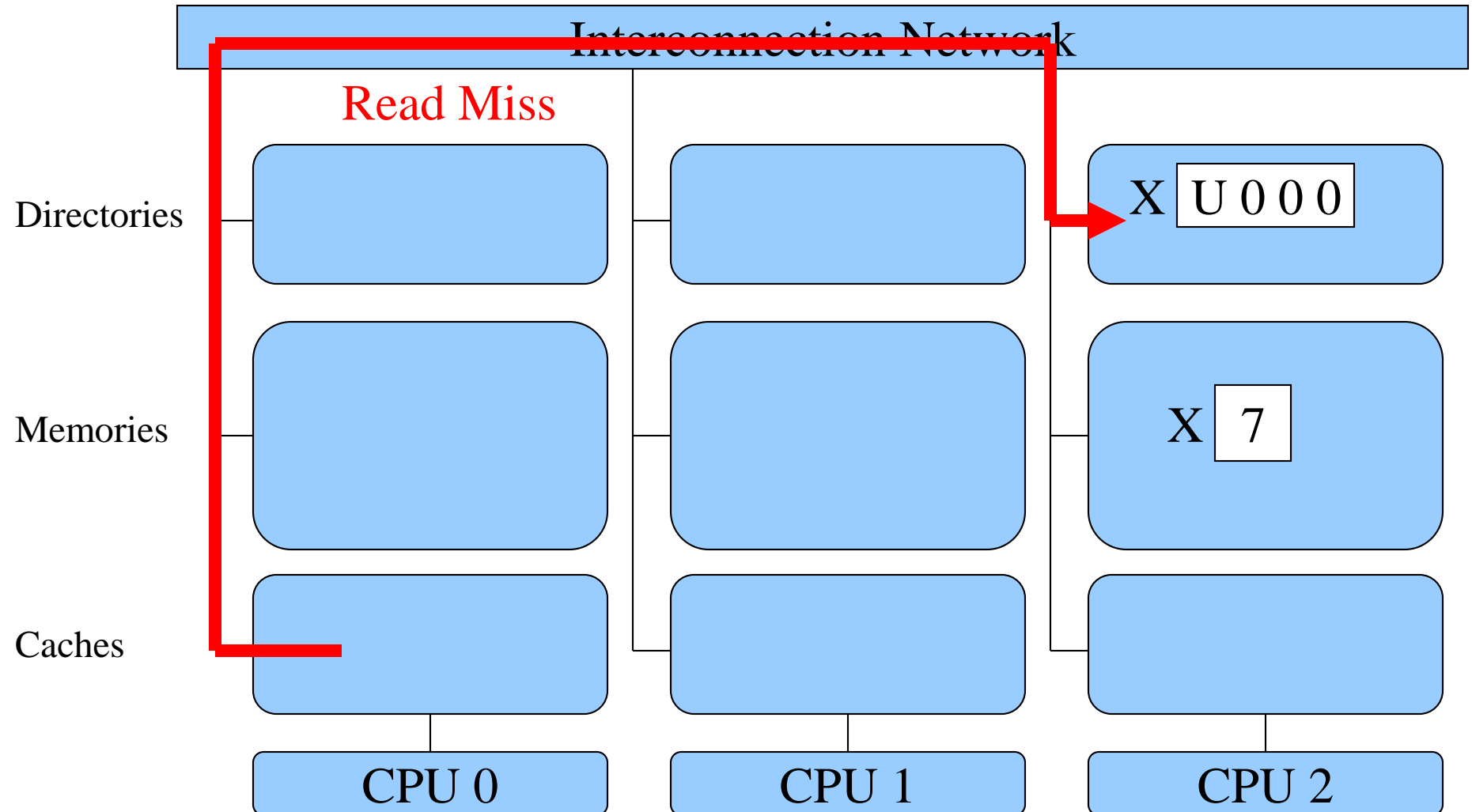


# Directory-based Protocol

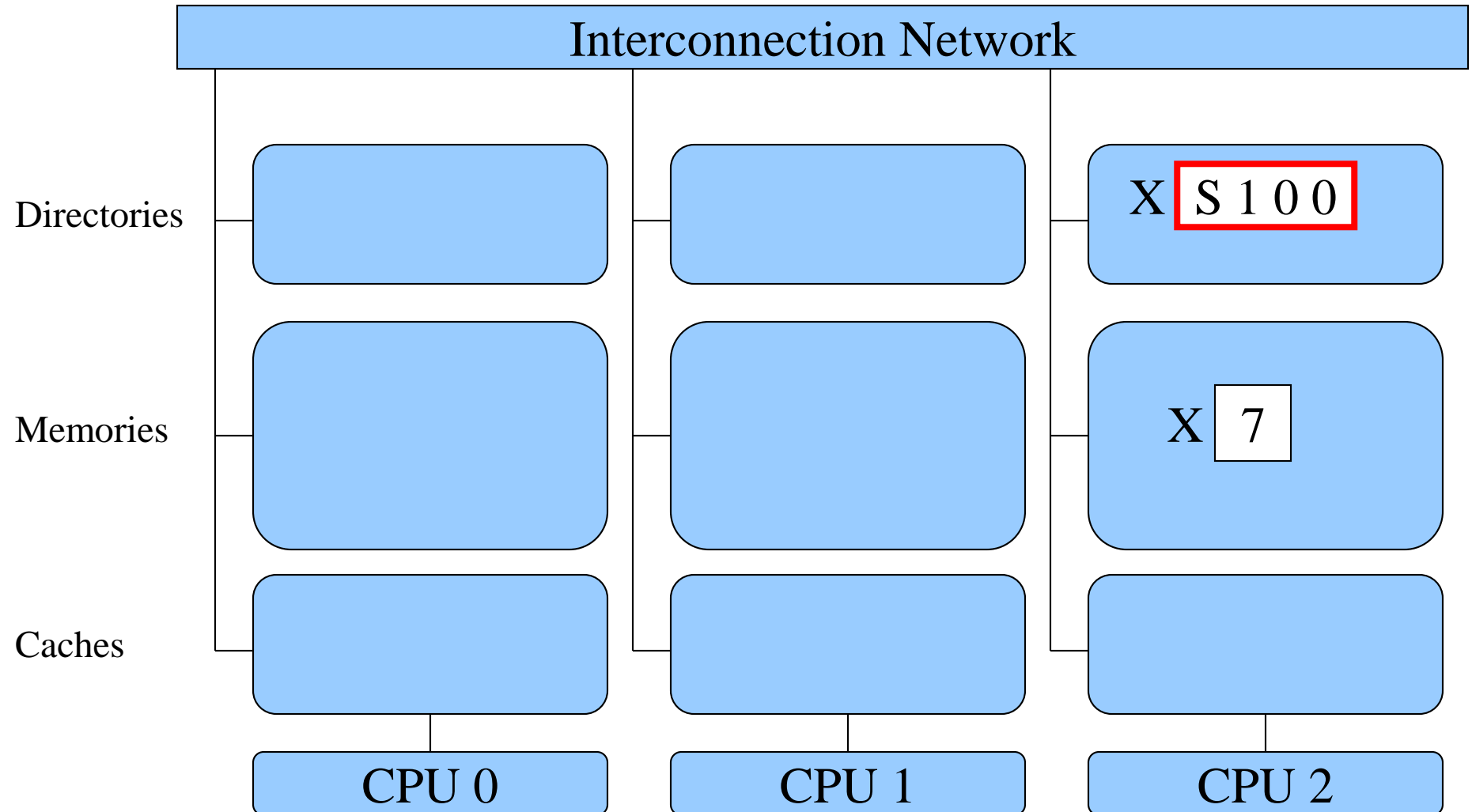




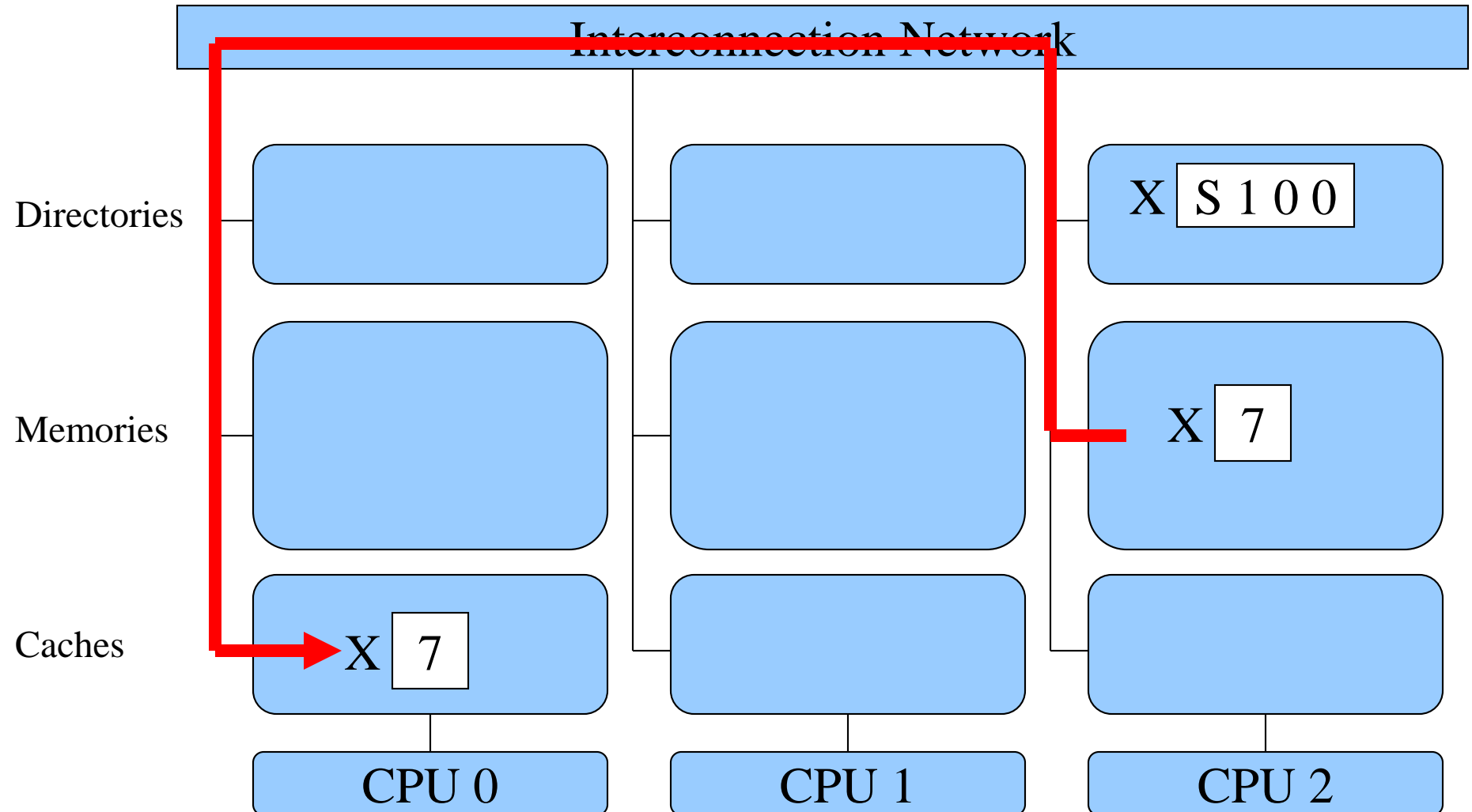
# CPU 0 Reads X



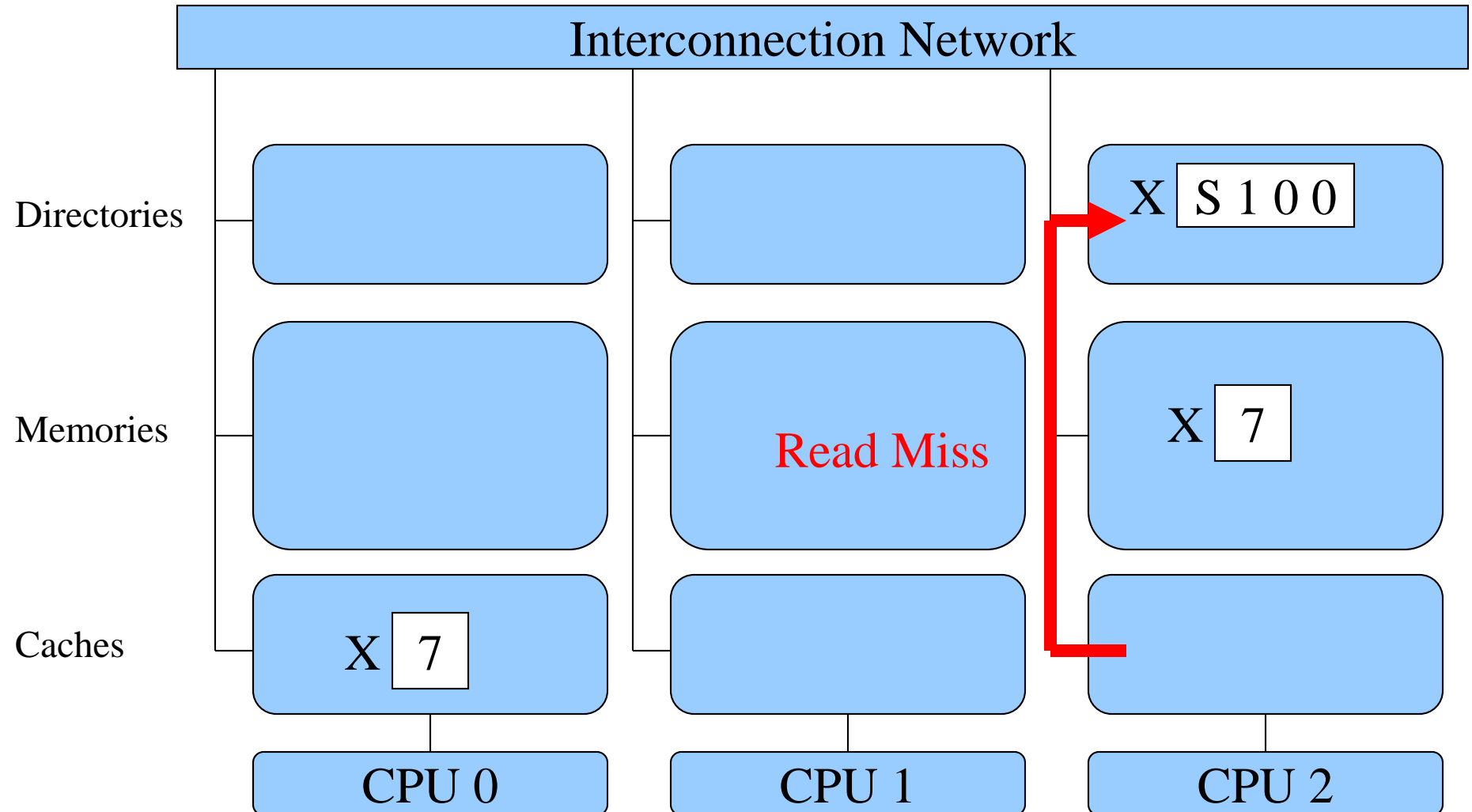
# CPU 0 Reads X



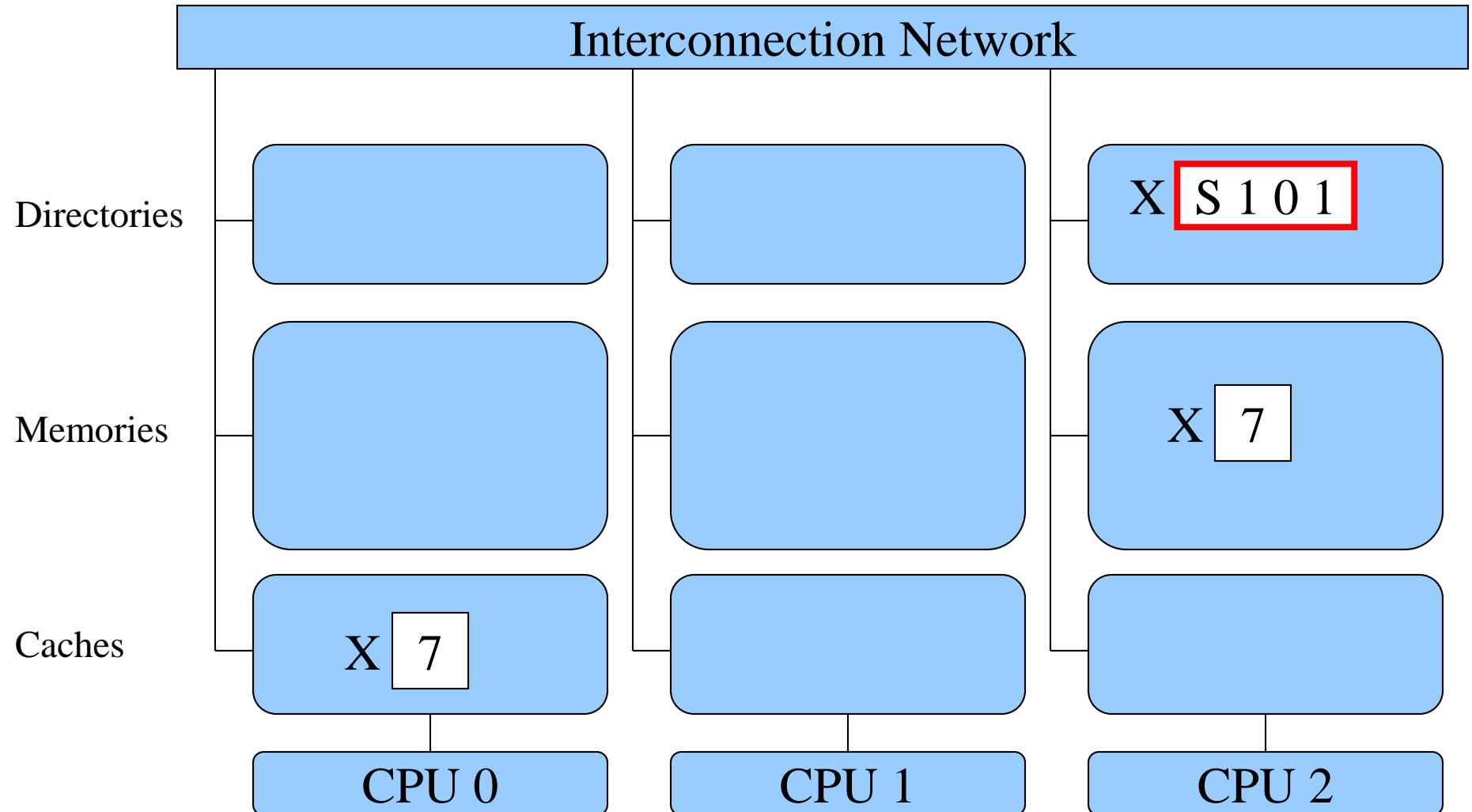
# CPU 0 Reads X



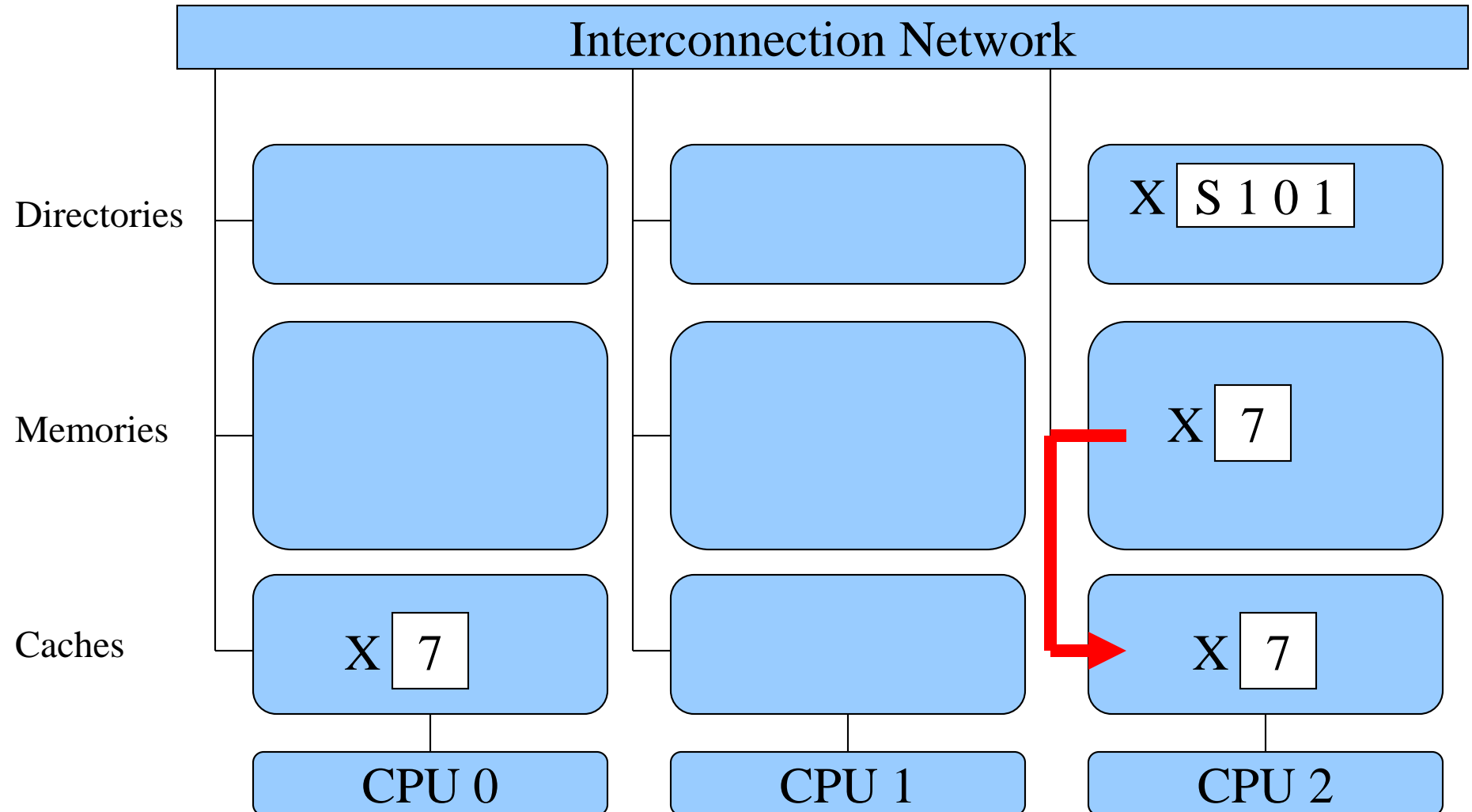
# CPU 2 Reads X



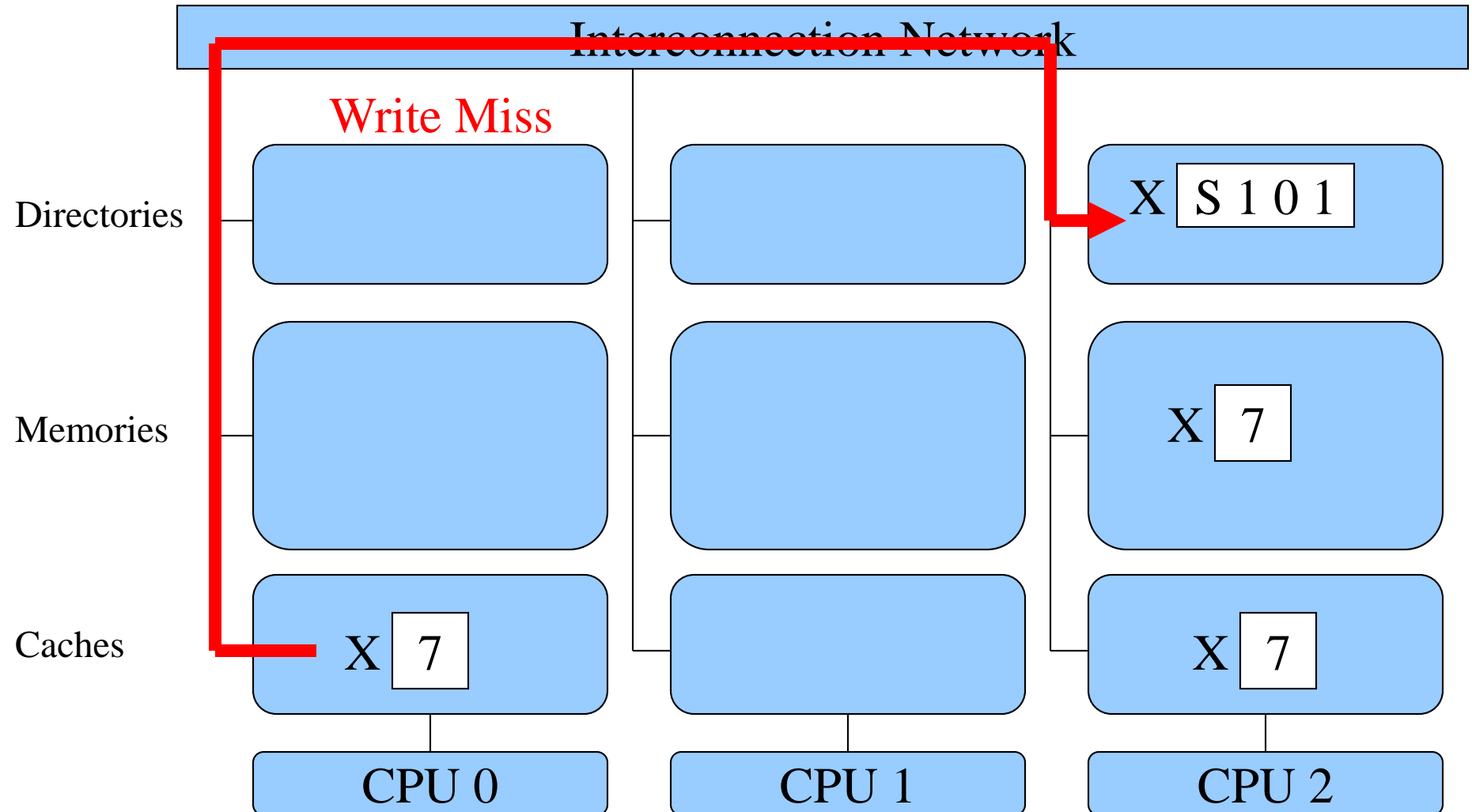
# CPU 2 Reads X



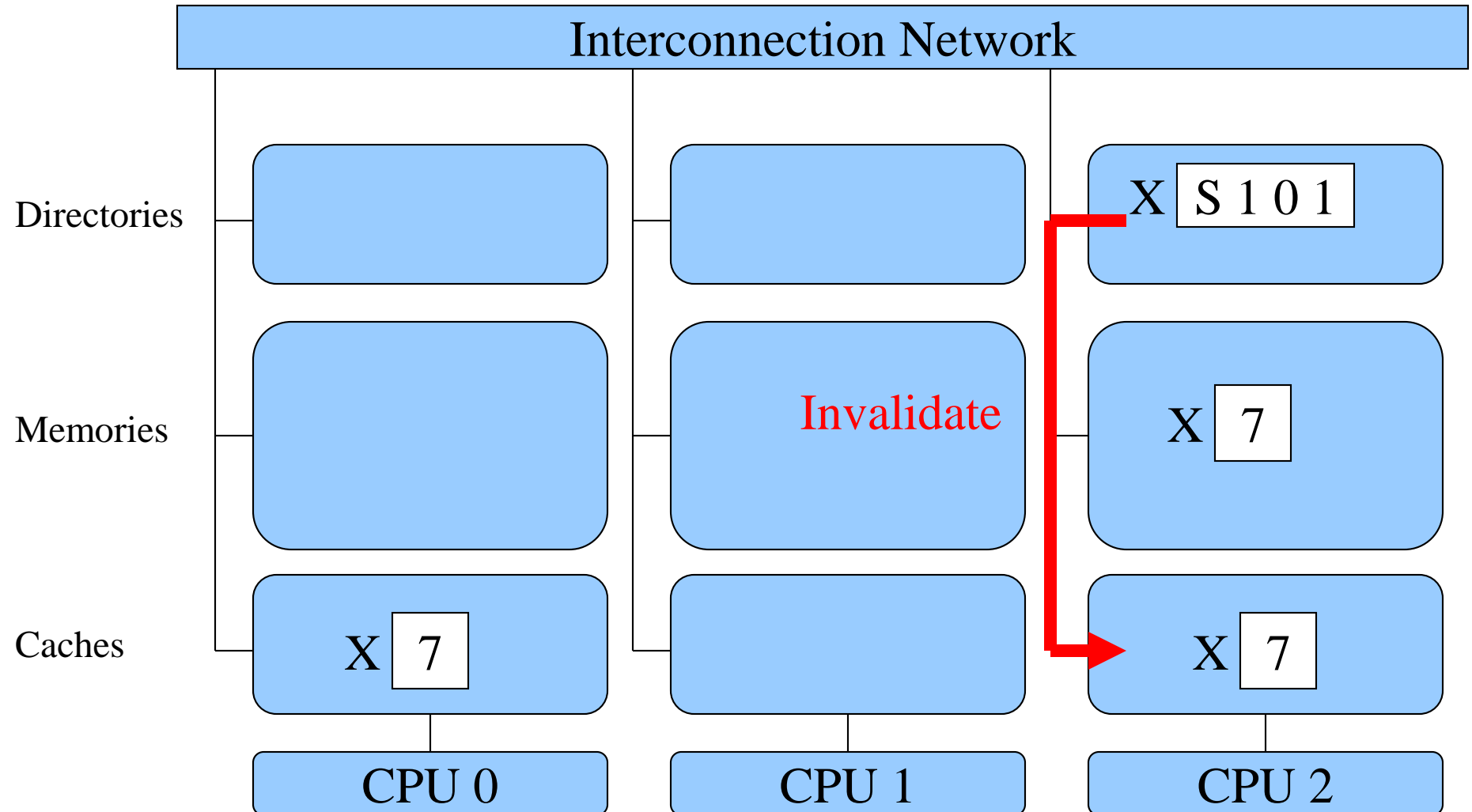
# CPU 2 Reads X



# CPU 0 Writes 6 to X

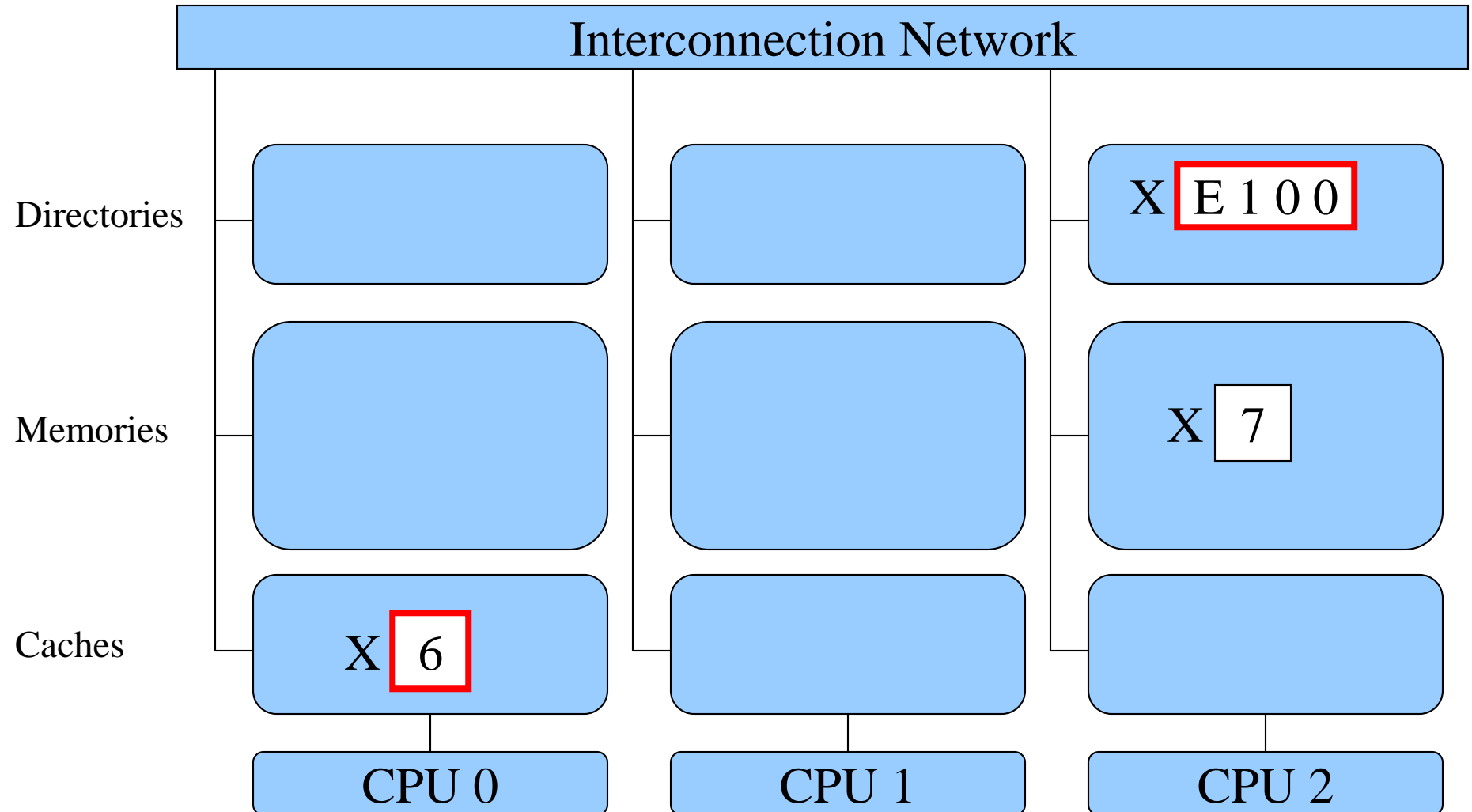


# CPU 0 Writes 6 to X

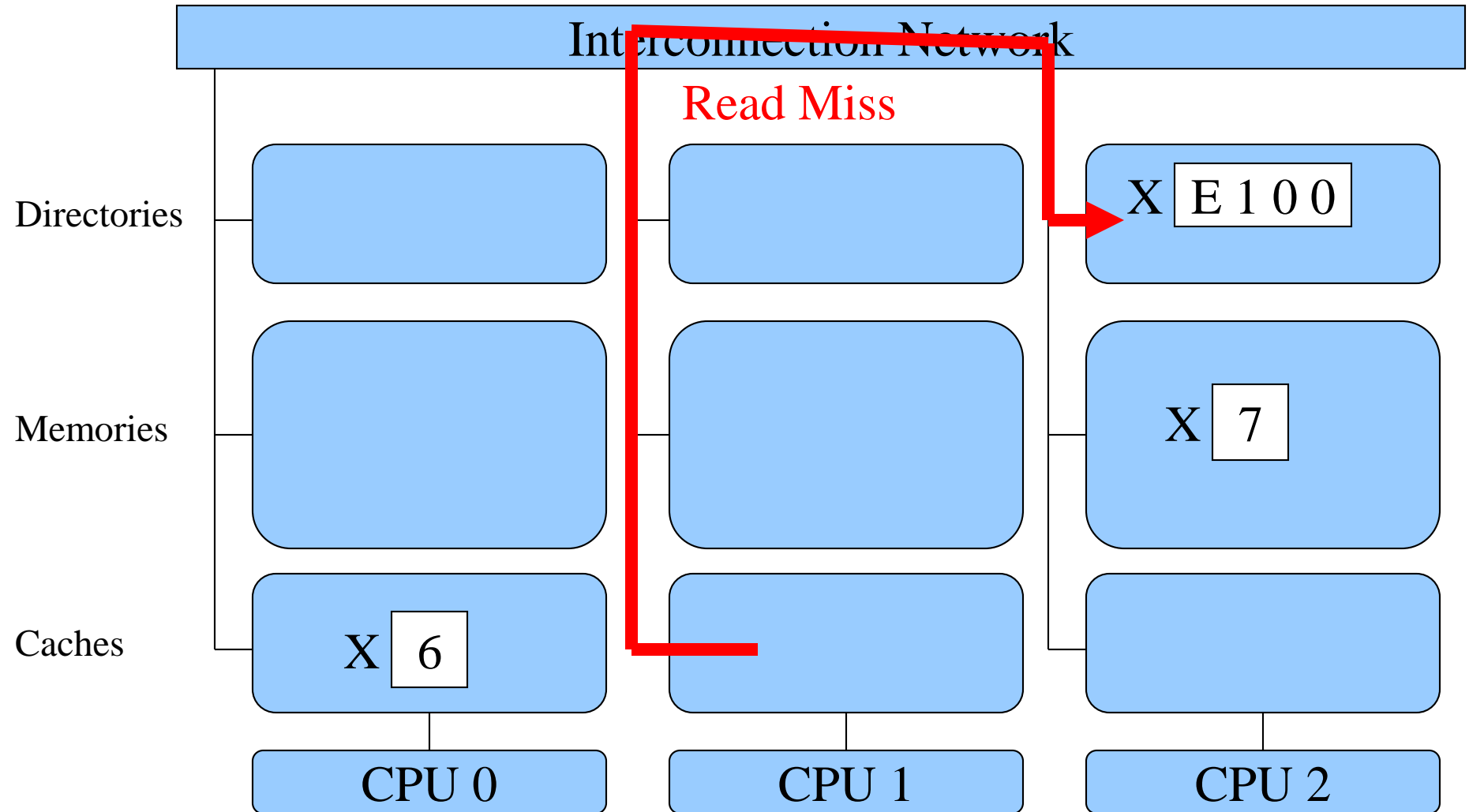




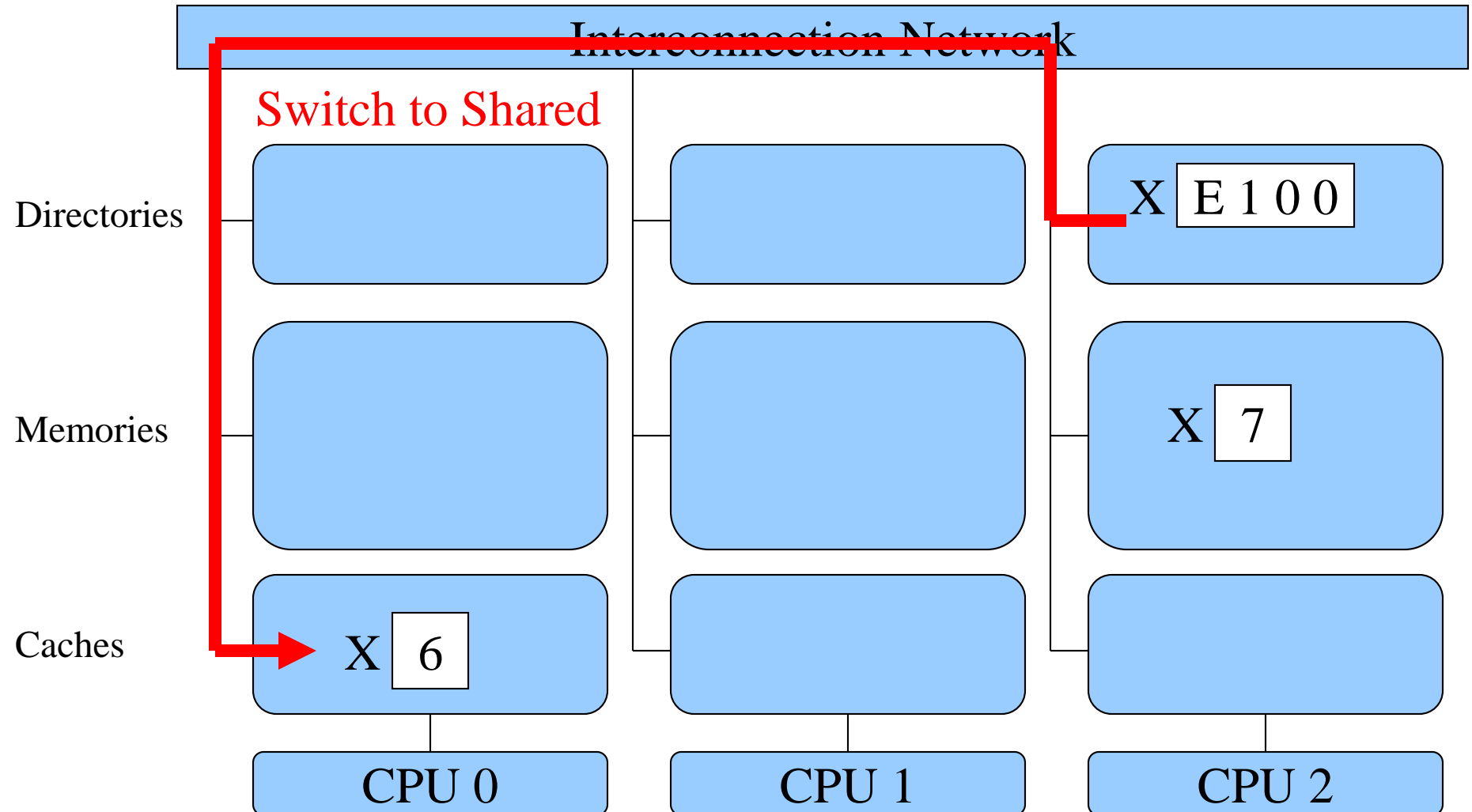
# CPU 0 Writes 6 to X



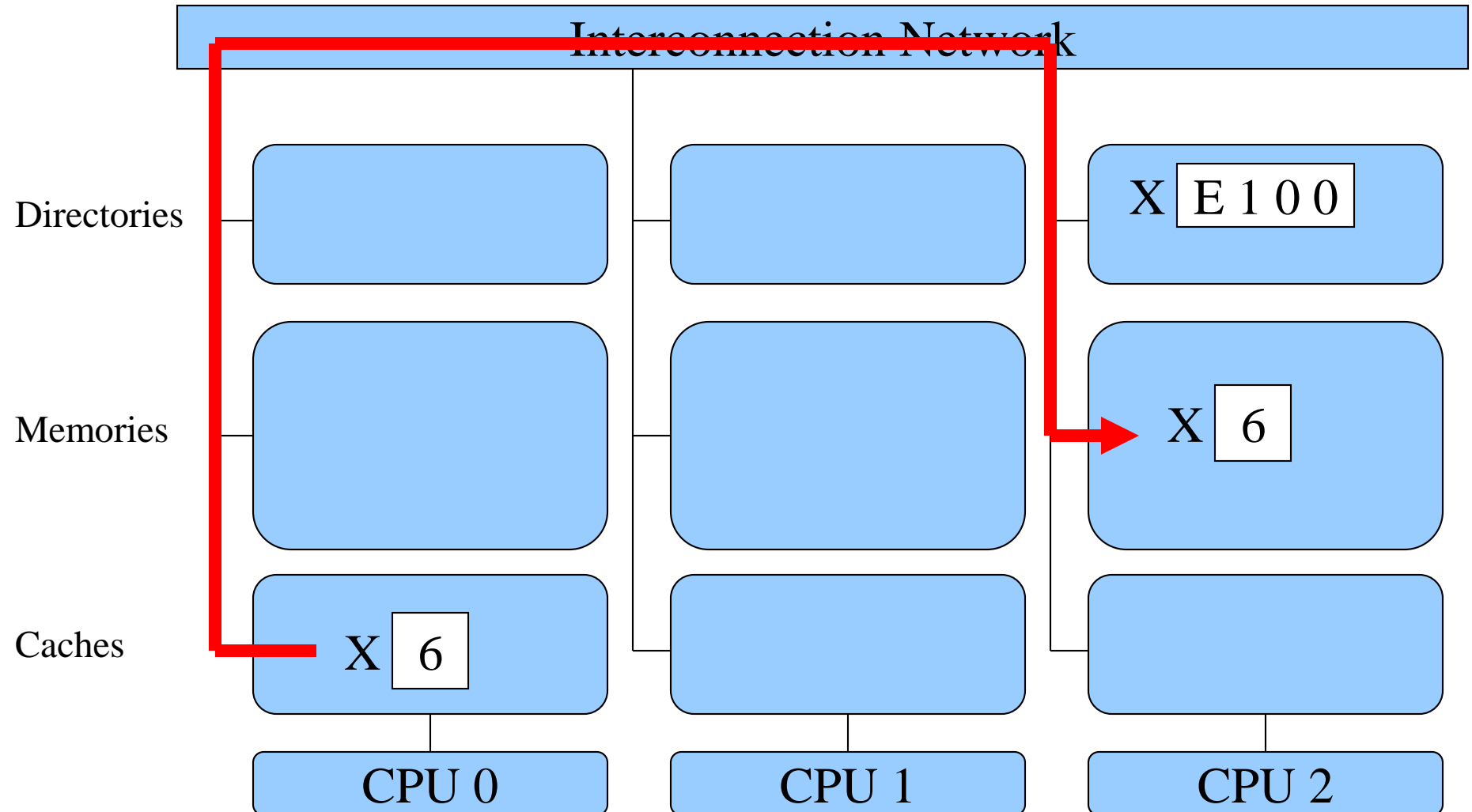
# CPU 1 Reads X



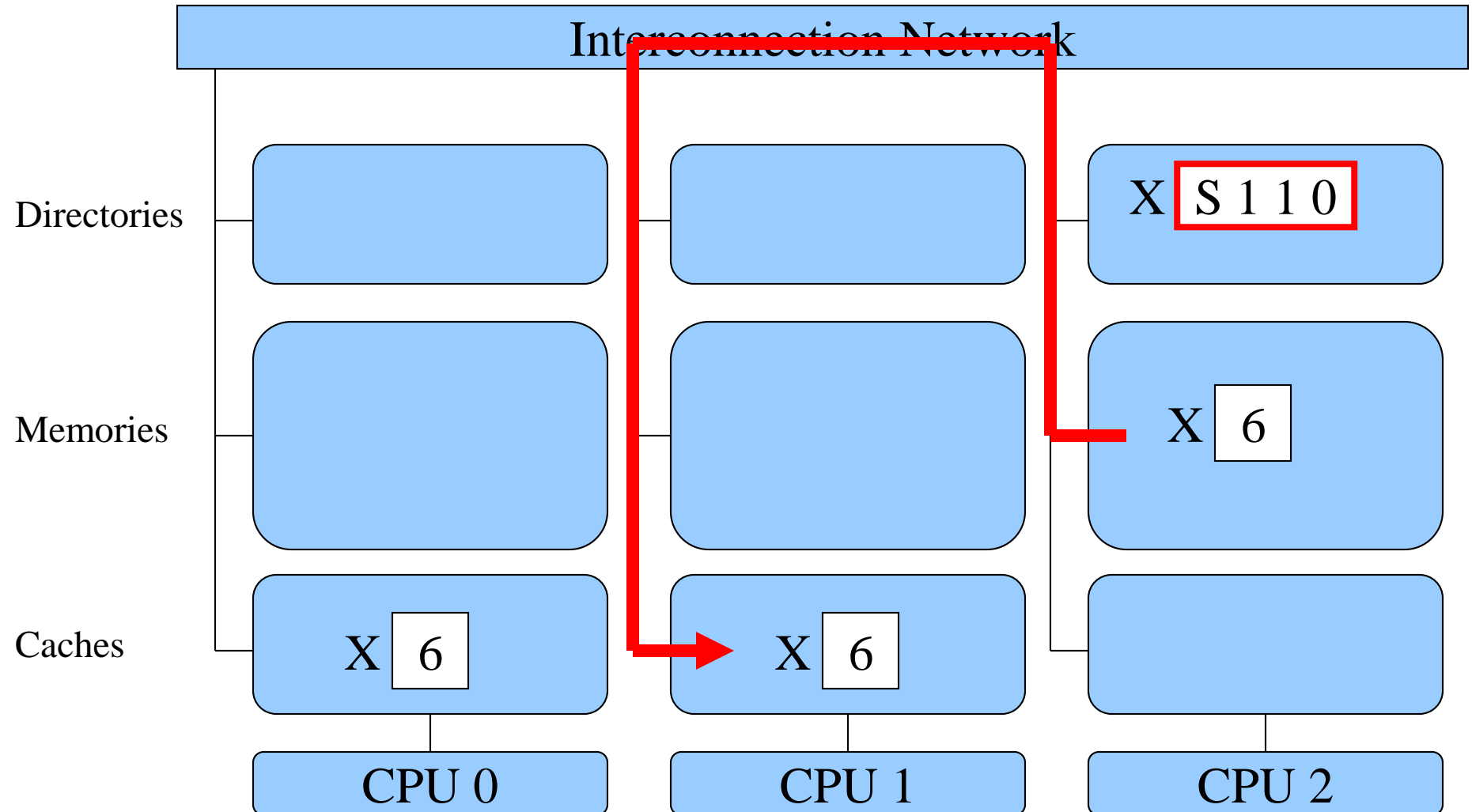
# CPU 1 Reads X



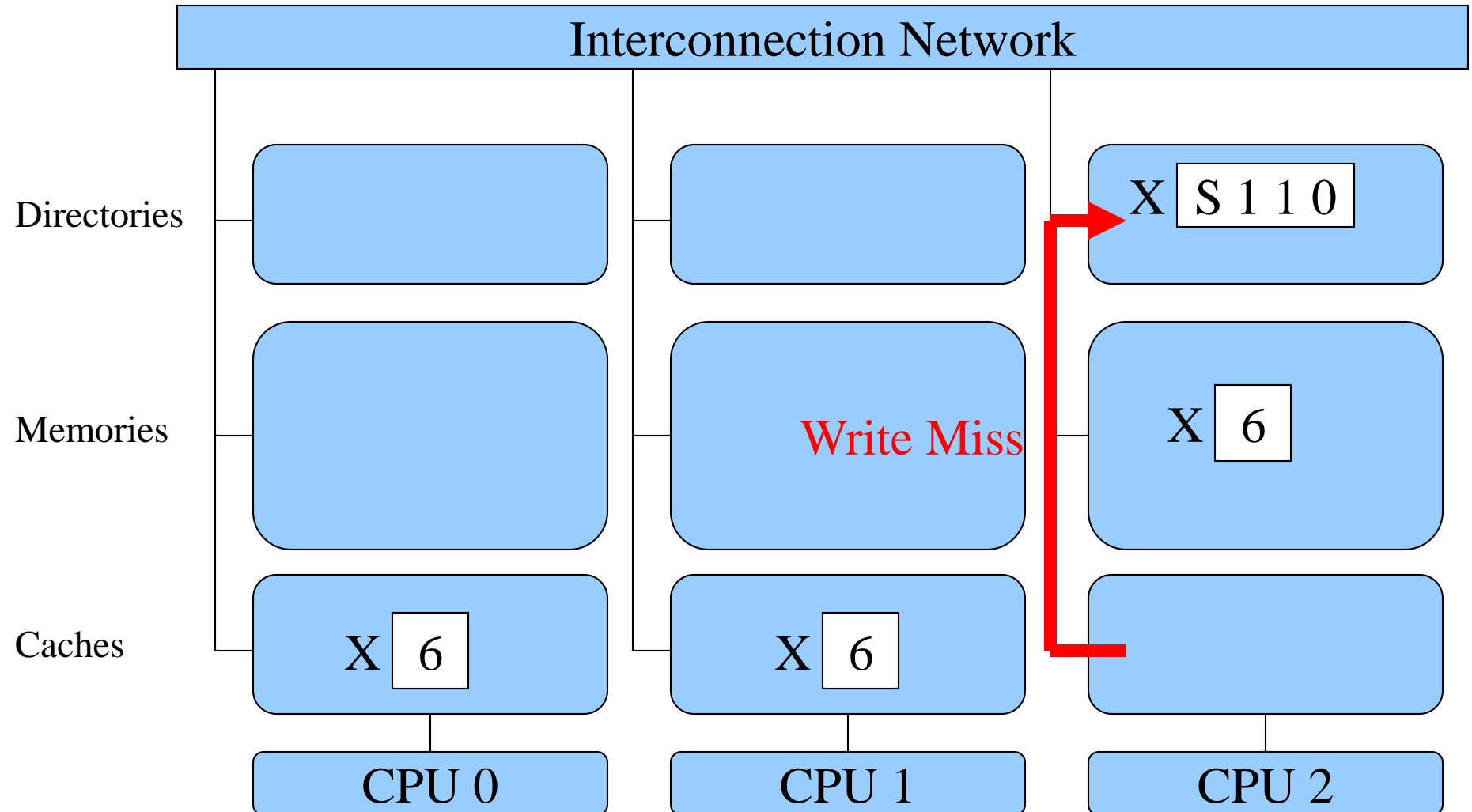
# CPU 1 Reads X



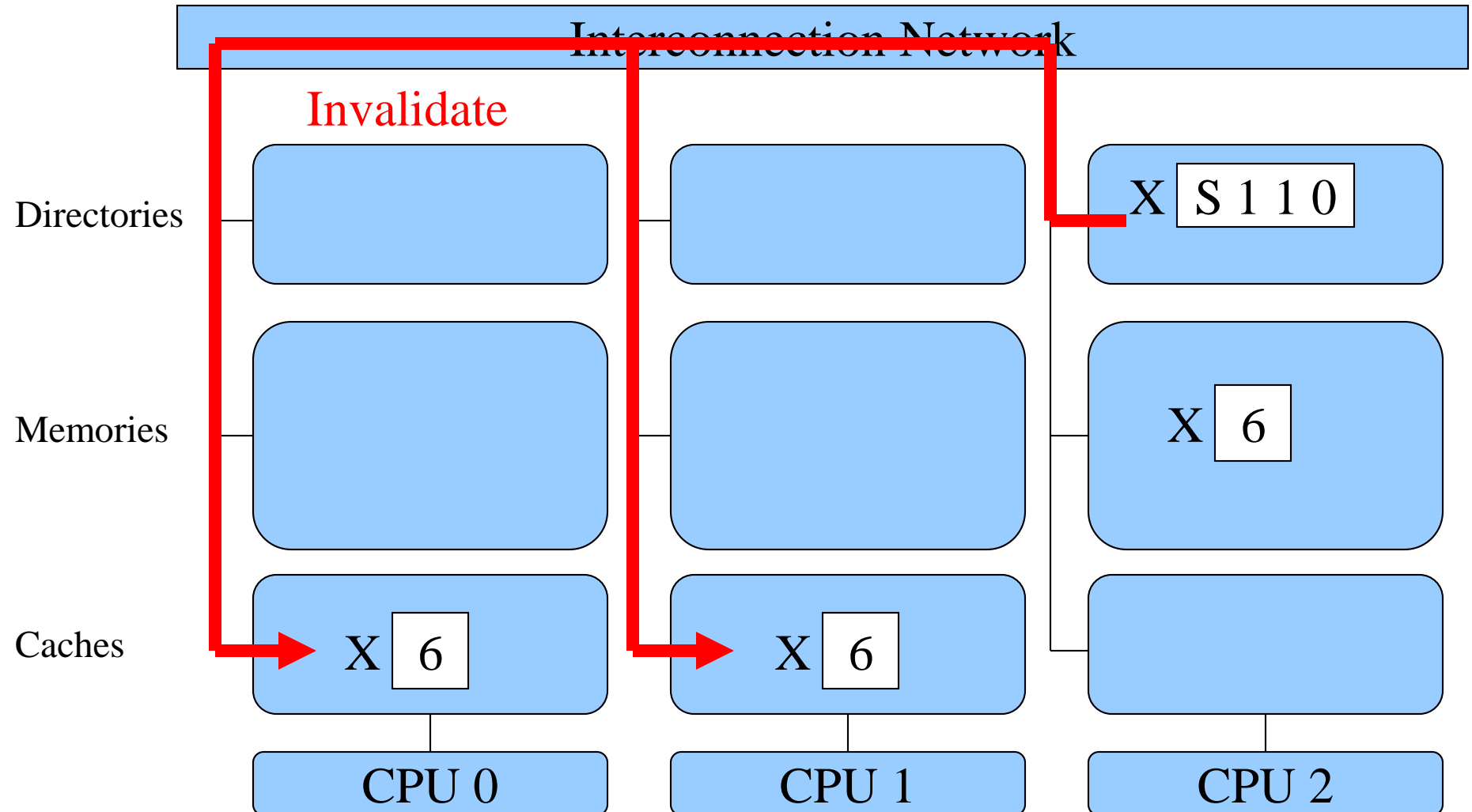
# CPU 1 Reads X



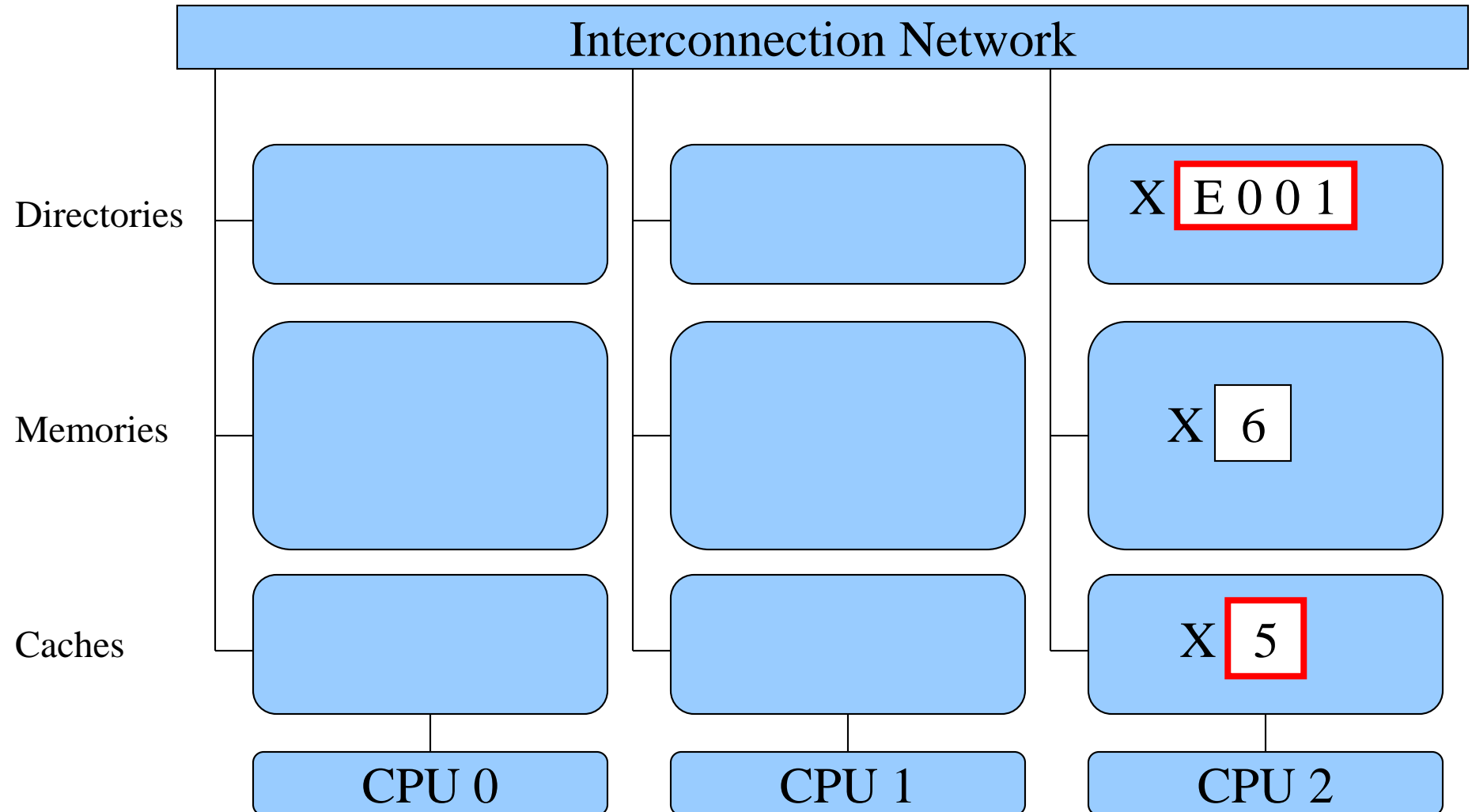
# CPU 2 Writes 5 to X



# CPU 2 Writes 5 to X

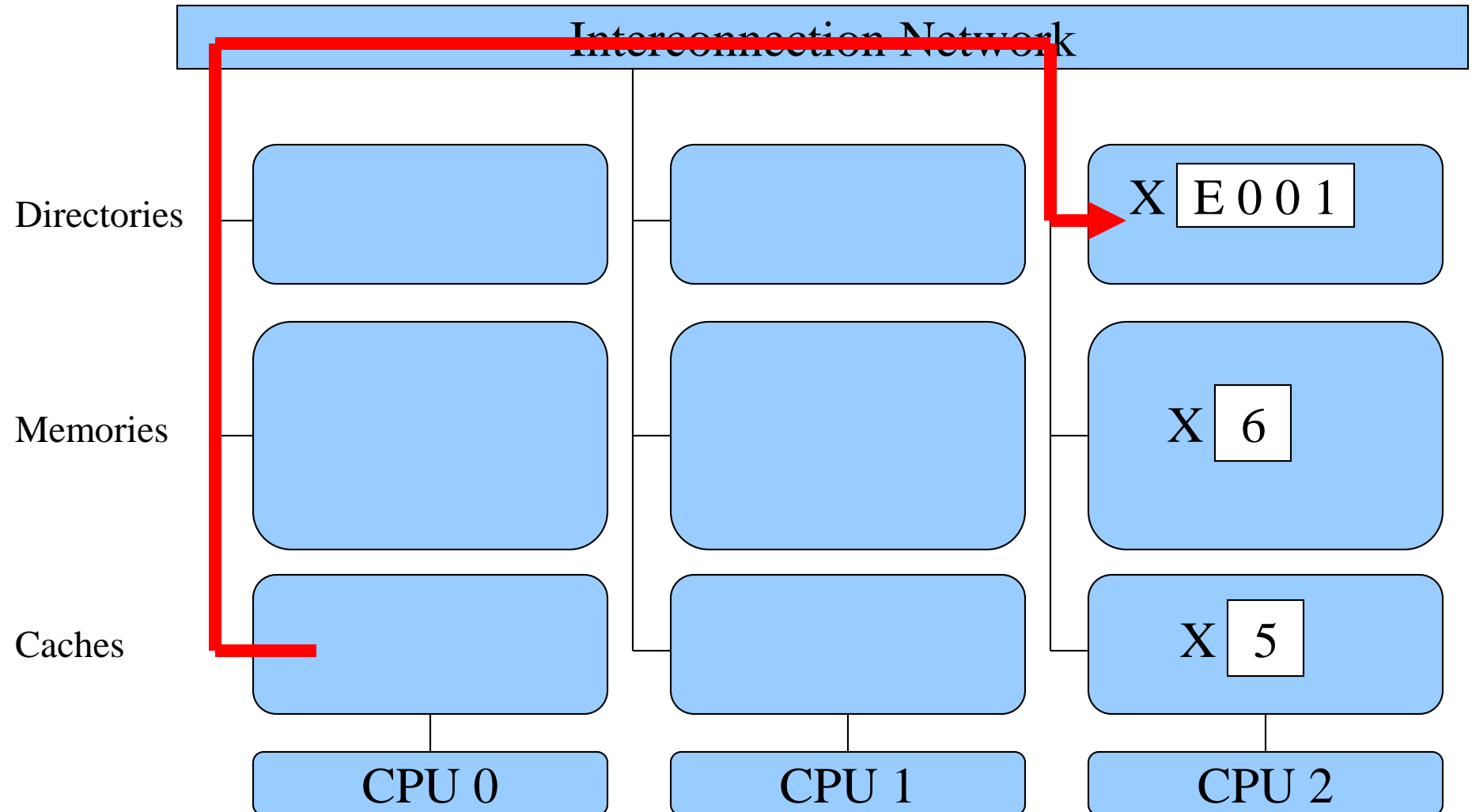


# CPU 2 Writes 5 to X

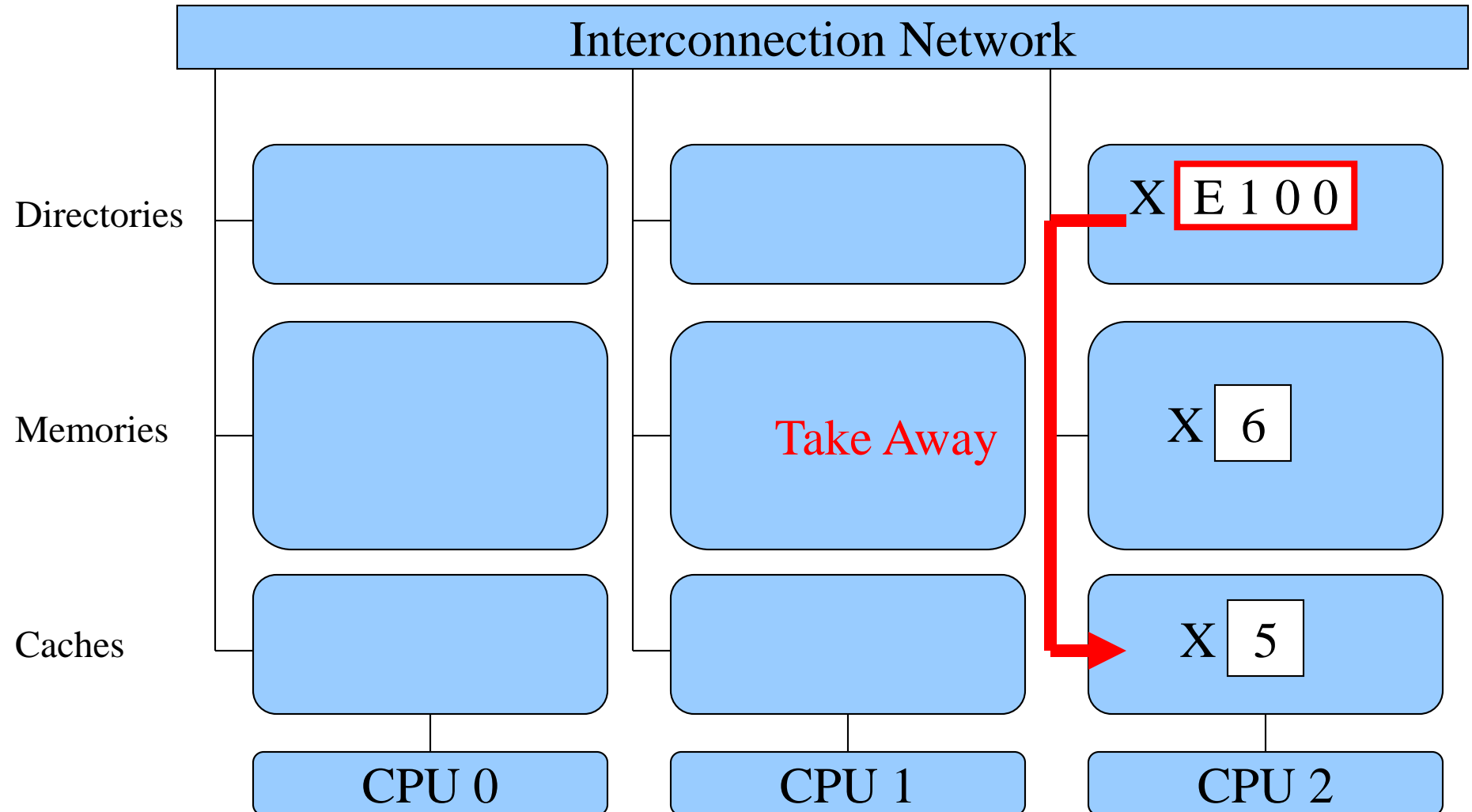




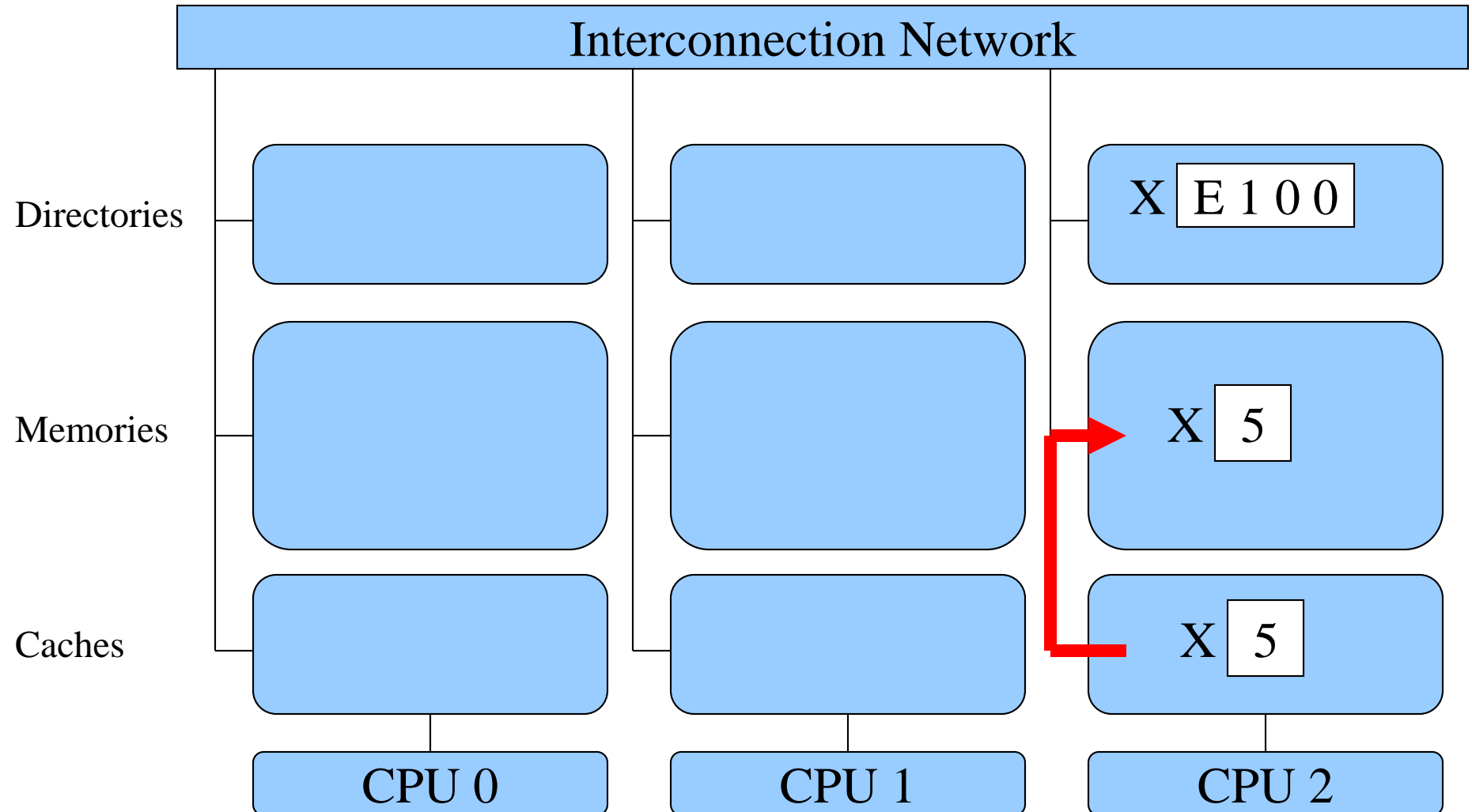
# CPU 0 Writes 4 to X



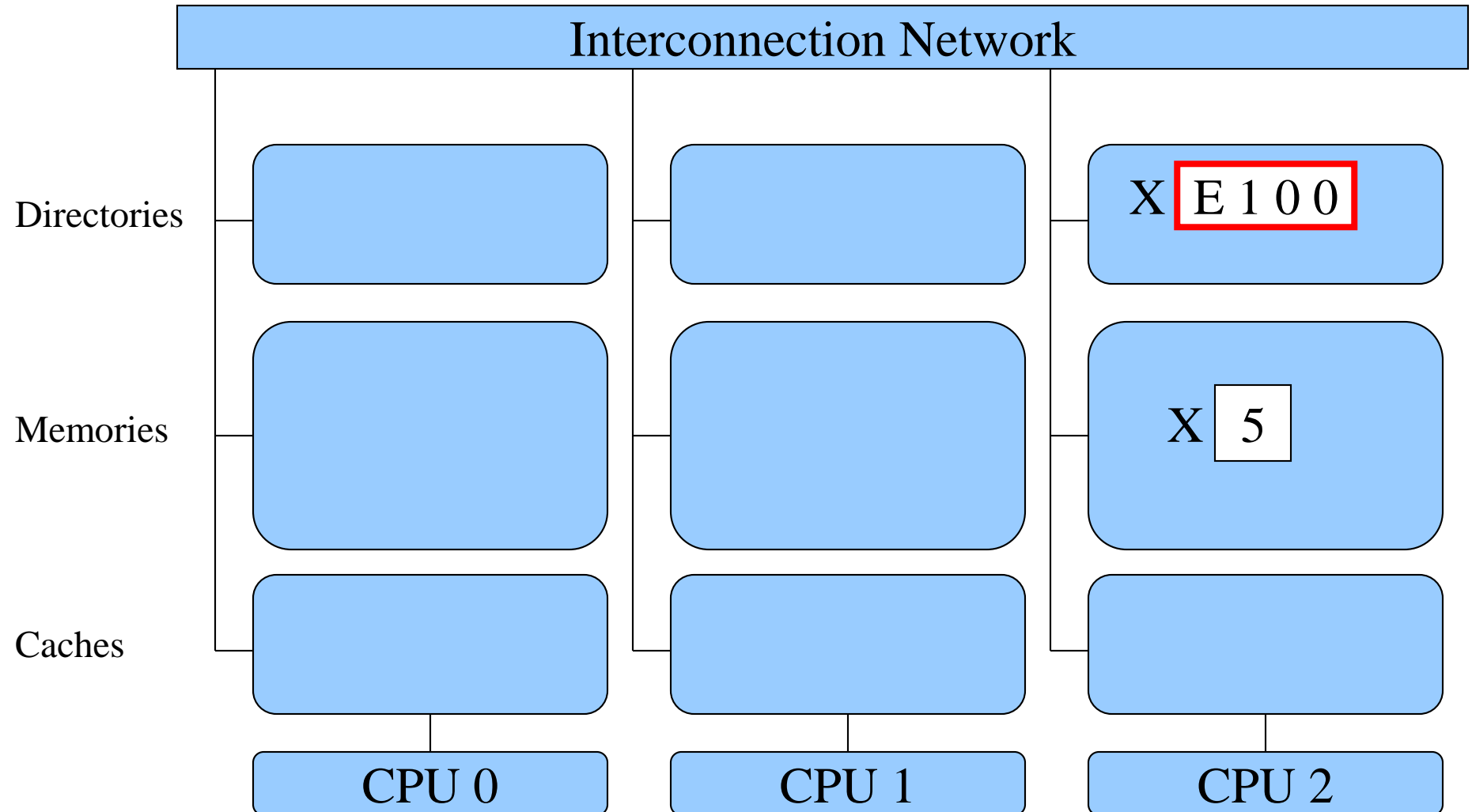
# CPU 0 Writes 4 to X



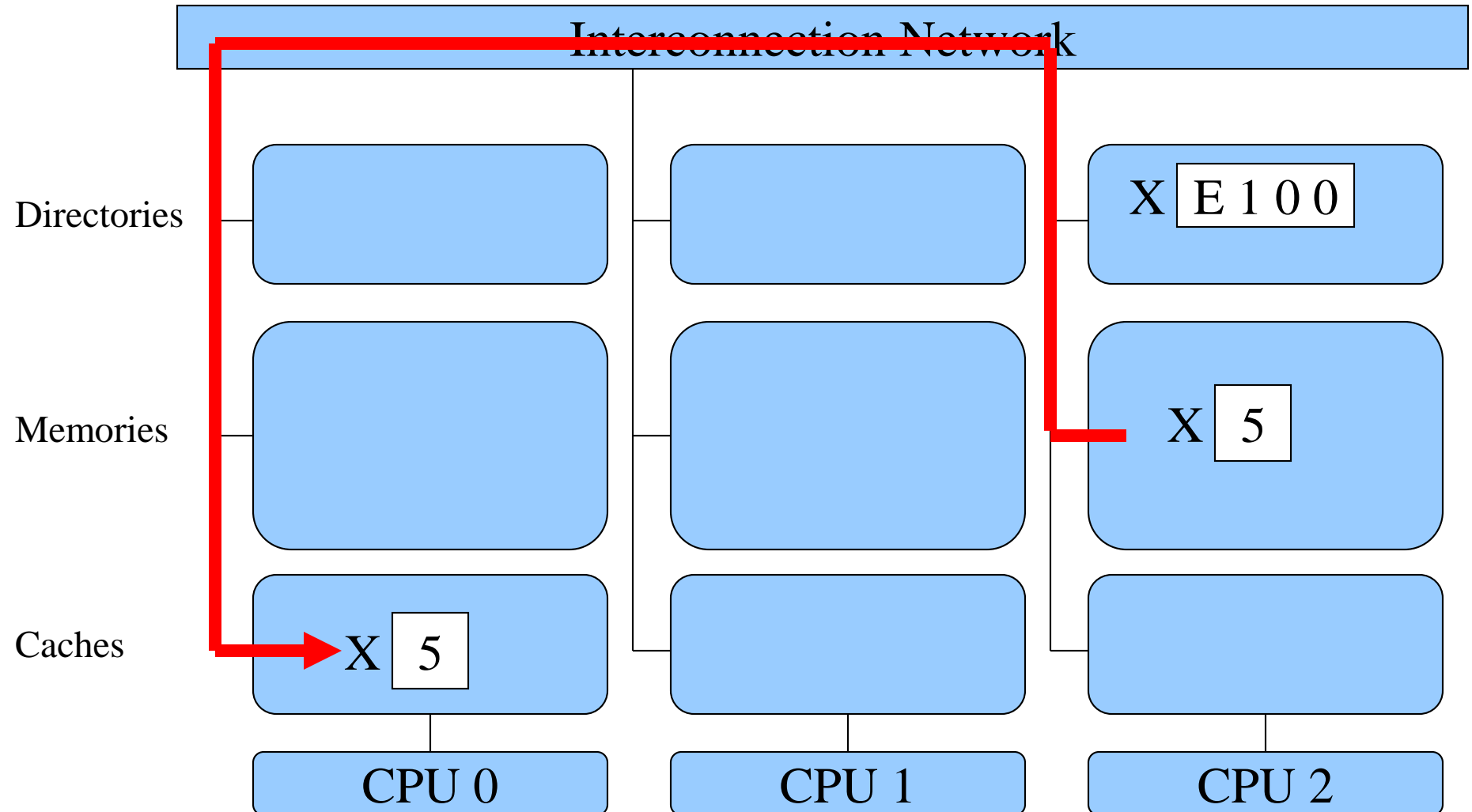
# CPU 0 Writes 4 to X



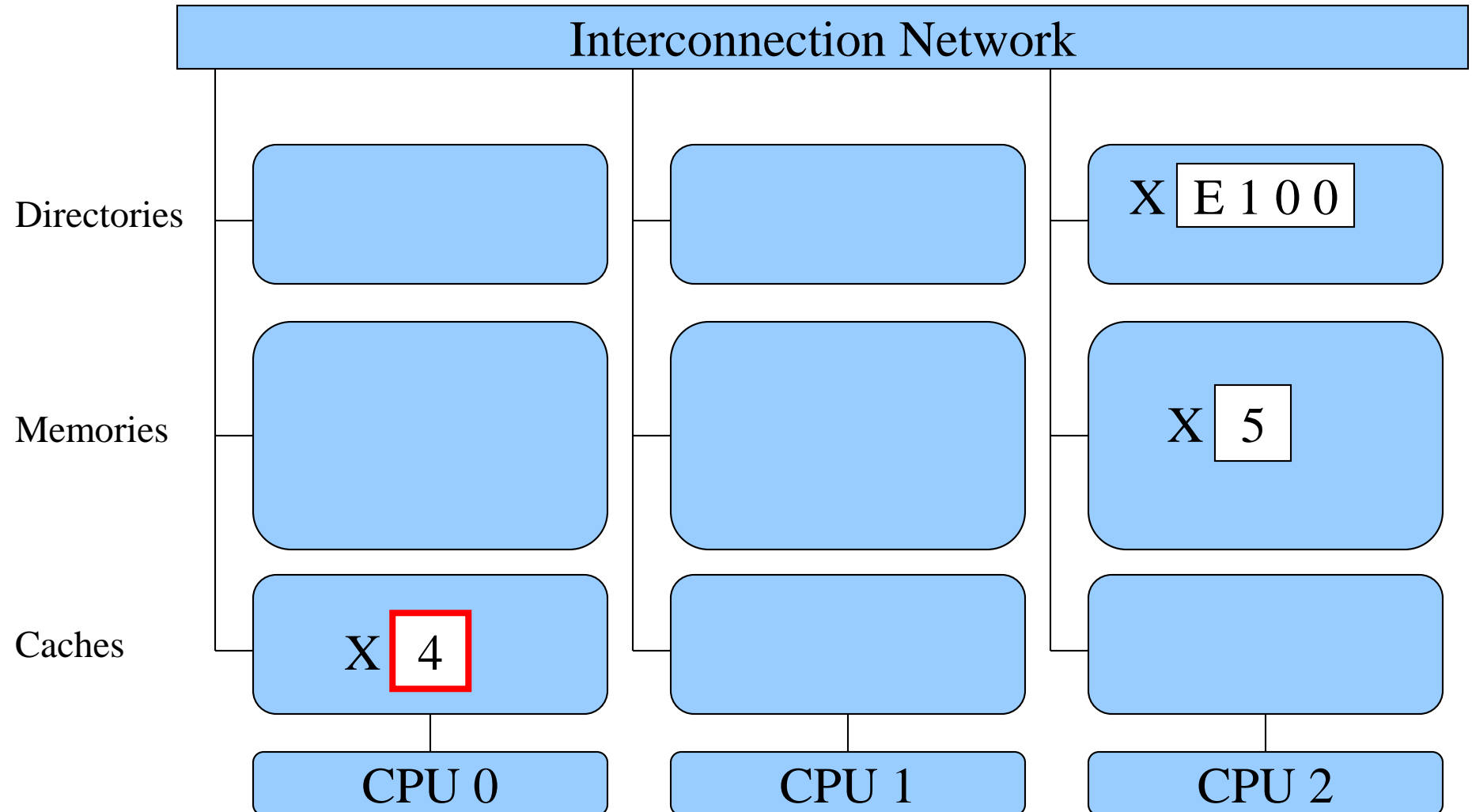
# CPU 0 Writes 4 to X



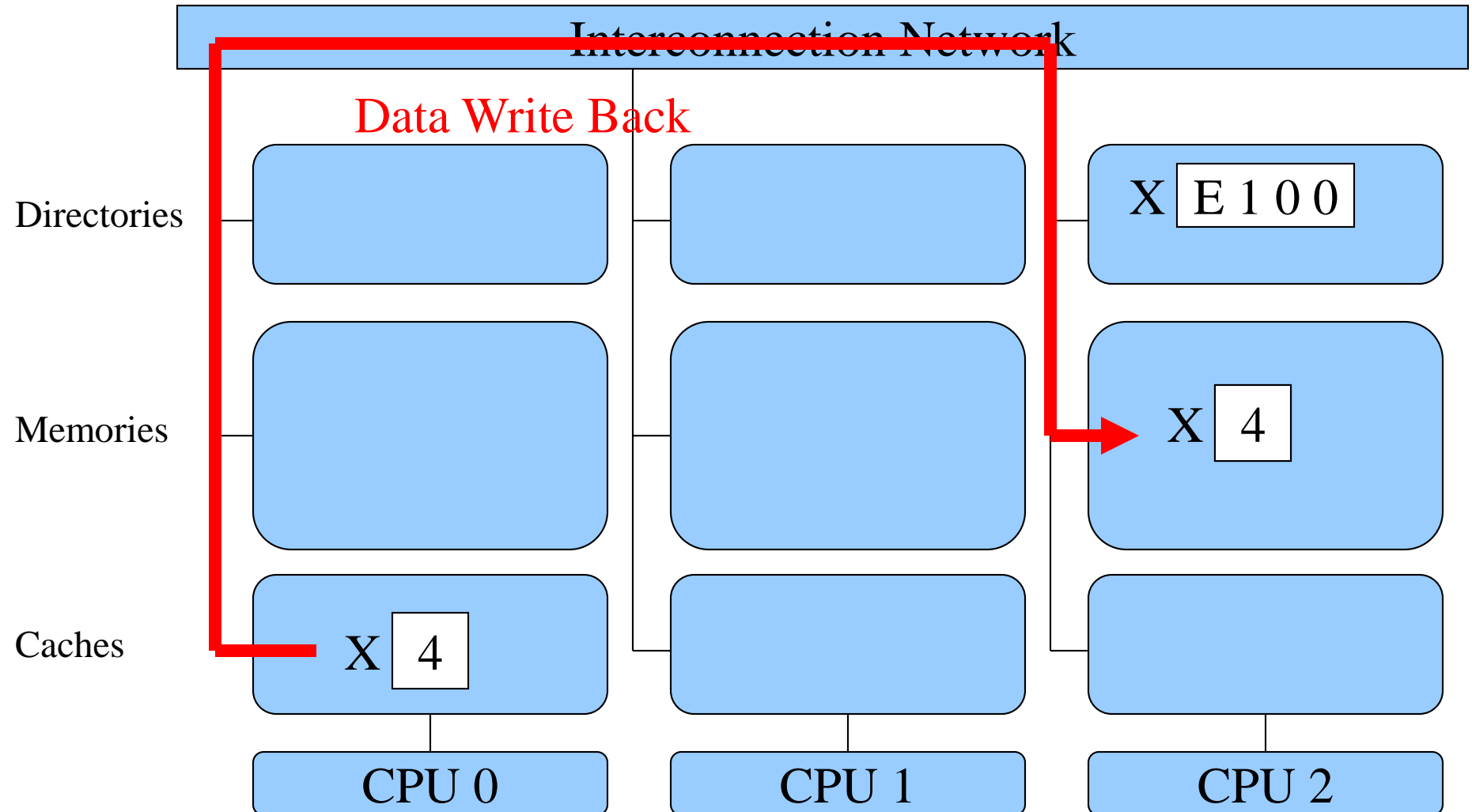
# CPU 0 Writes 4 to X



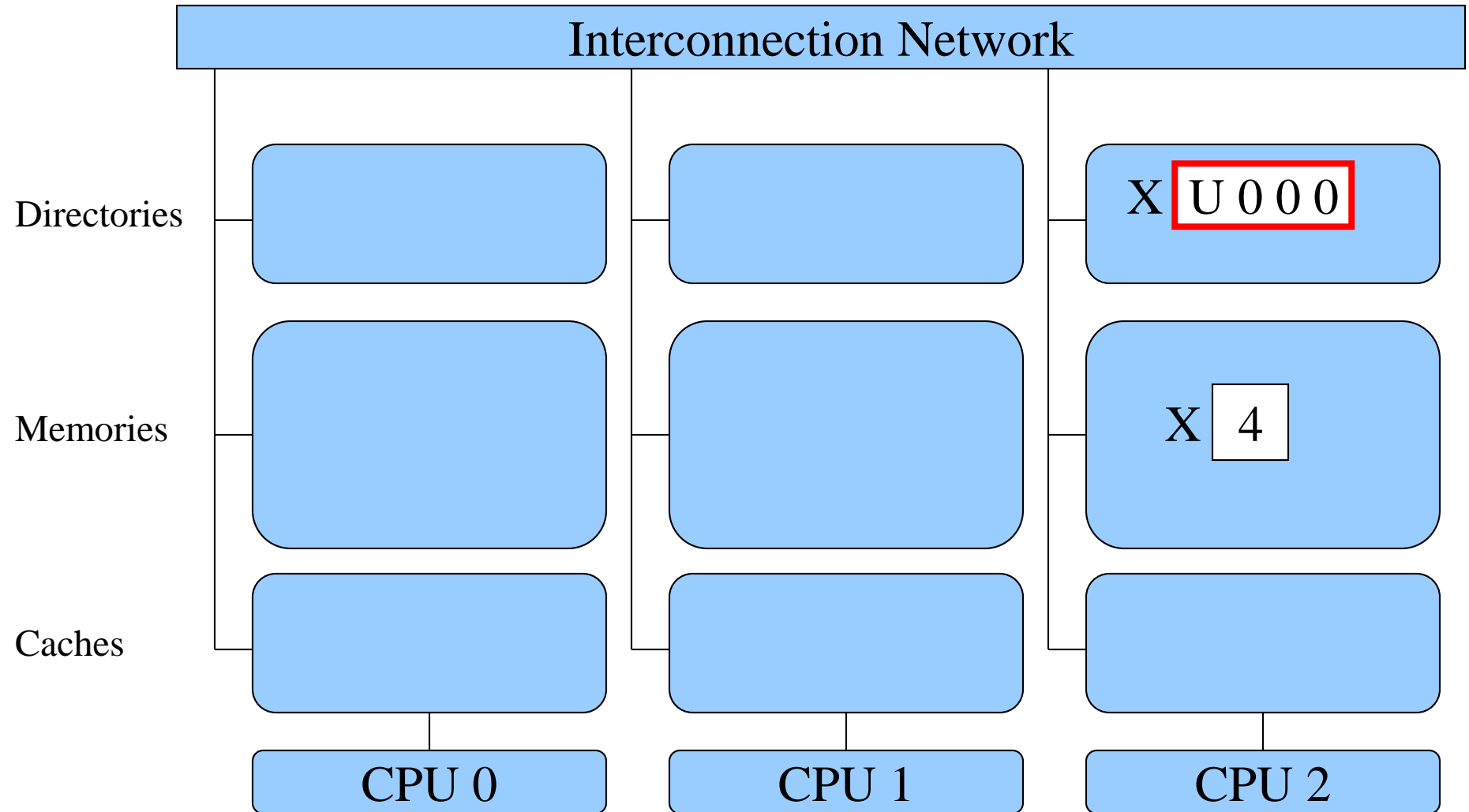
# CPU 0 Writes 4 to X



# CPU 0 Writes Back X Block



# CPU 0 Writes Back X Block



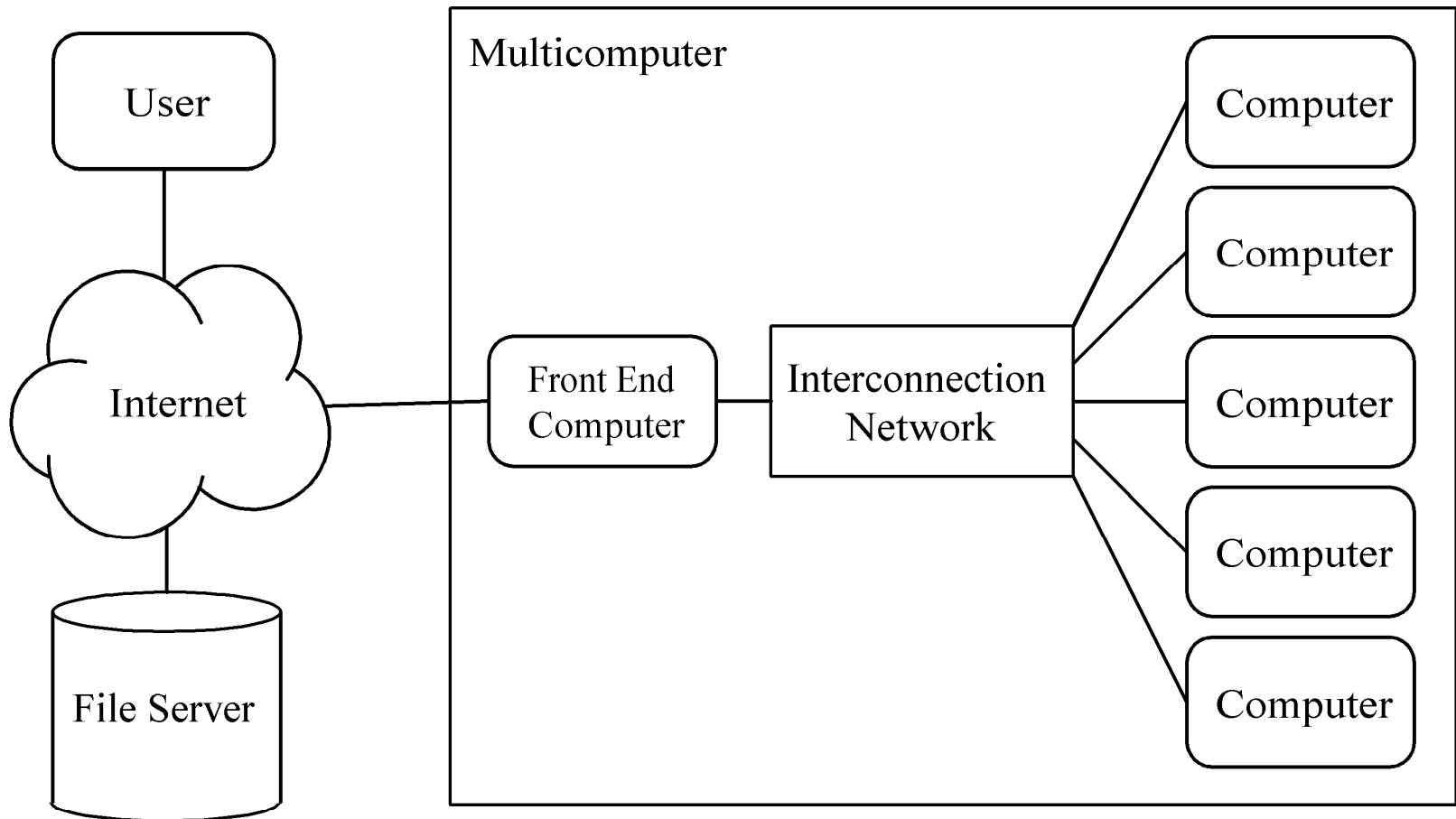


# Multicomputer

---

- Distributed memory multiple-CPU computer
- Same address on different processors refers to different physical memory locations
- Processors interact through message passing
- Commercial multicomputers
- Commodity clusters

# Asymmetrical Multicomputer



# Asymmetrical MC Advantages

---

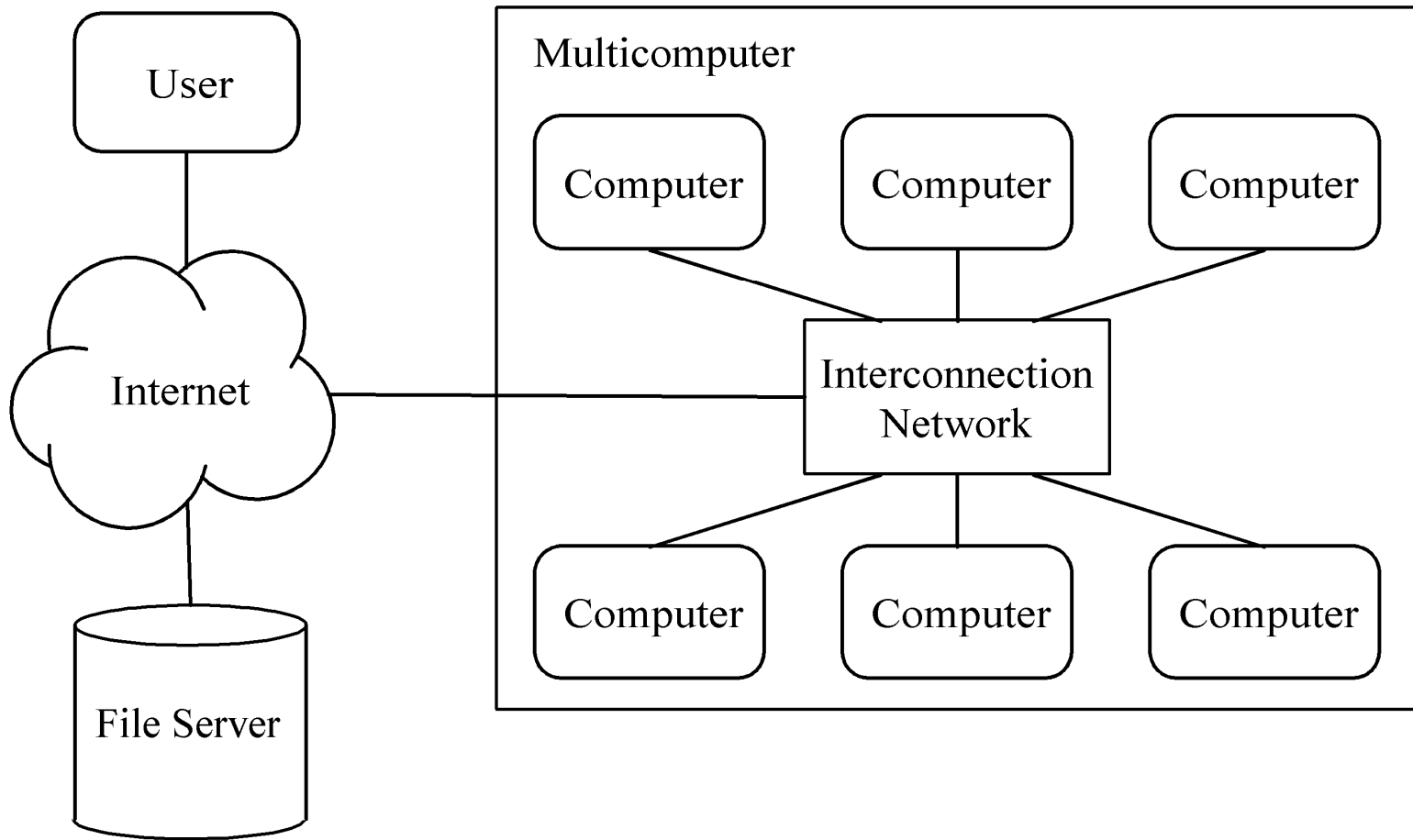
- Back-end processors dedicated to parallel computations  $\Rightarrow$  Easier to understand, model, tune performance
- Only a simple back-end operating system needed  $\Rightarrow$  Easy for a vendor to create

# Asymmetrical MC Disadvantages

---

- Front-end computer is a single point of failure
- Single front-end computer limits scalability of system
- Primitive operating system in back-end processors makes debugging difficult
- Every application requires development of both front-end and back-end program

# Symmetrical Multicomputer



# Symmetrical MC Advantages

---

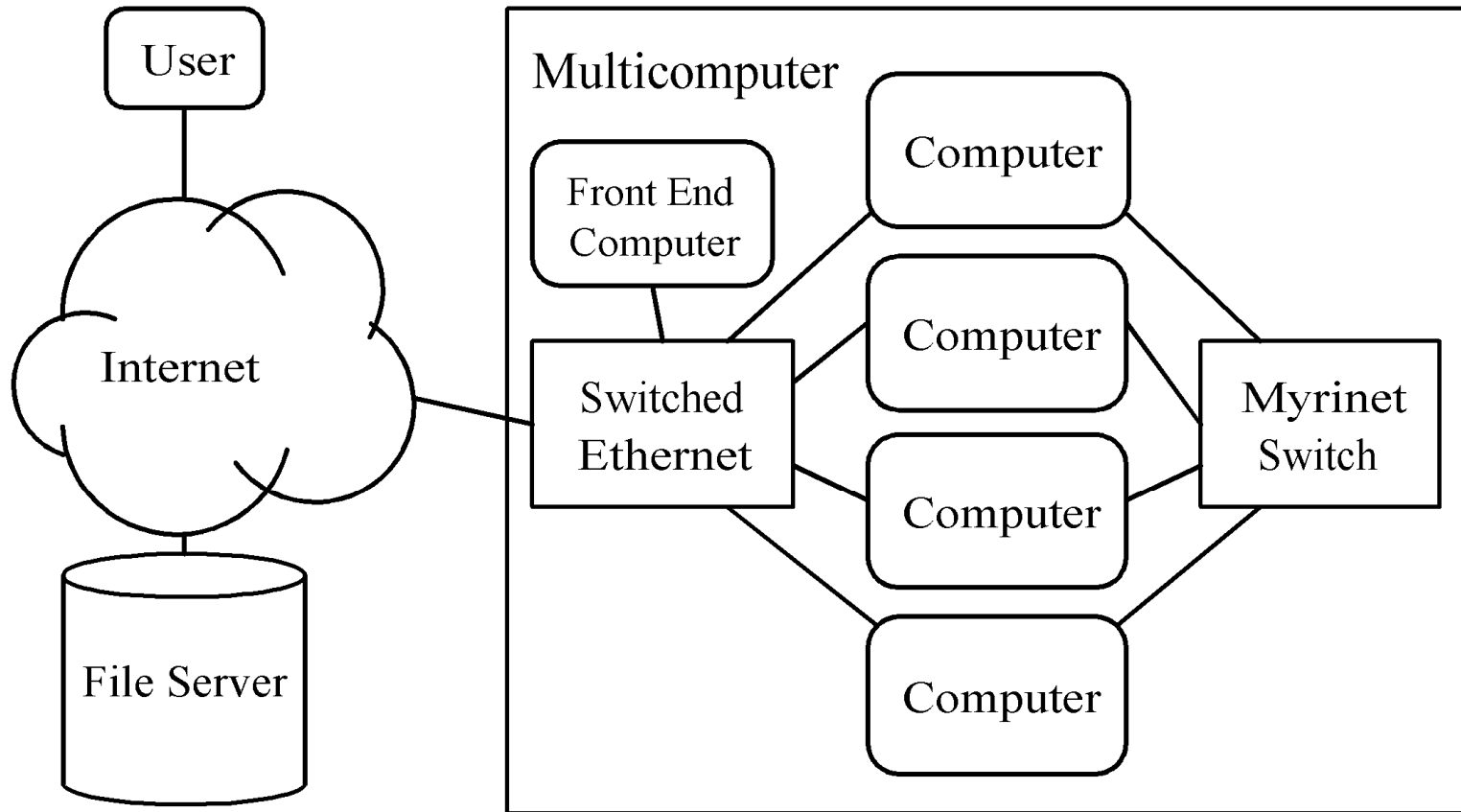
- Alleviate performance bottleneck caused by single front-end computer
- Better support for debugging
- Every processor executes same program

# Symmetrical MC Disadvantages

---

- More difficult to maintain illusion of single “parallel computer”
- No simple way to balance program development workload among processors
- More difficult to achieve high performance when multiple processes on each processor

# ParPar Cluster, A Mixed Model





# Commodity Cluster

---

- Co-located computers
- Dedicated to running parallel jobs
- No keyboards or displays
- Identical operating system
- Identical local disk images
- Administered as an entity

# Network of Workstations

---

- Dispersed computers
- First priority: person at keyboard
- Parallel jobs run in background
- Different operating systems
- Different local images
- Checkpointing and restarting important

# Flynn's Taxonomy

---

- Instruction stream
- Data stream
- Single vs. multiple
- Four combinations
  - SISD: uniprocessors
  - SIMD: proc arrays, pipelined vector procs
  - MISD: Systolic arrays
  - MIMD: multiprocessor, multicomputers

# SISD

---

- Single Instruction, Single Data
- Single-CPU systems
- Note: co-processors don't count
  - Functional
  - I/O
- Example: PCs

# SIMD

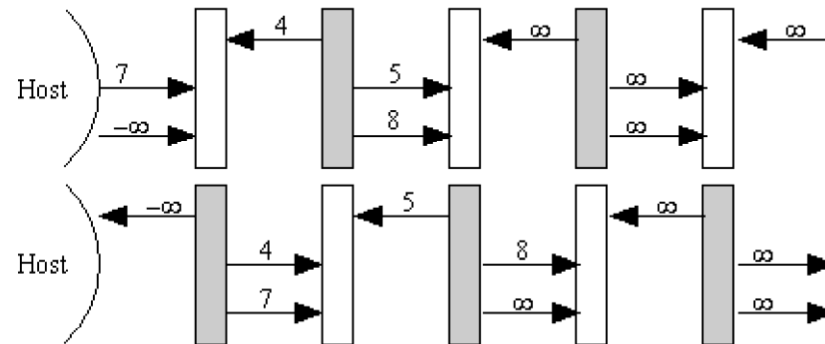
---

- Single Instruction, Multiple Data
- Two architectures fit this category
  - Pipelined vector processor  
(e.g., Cray-1)
  - Processor array  
(e.g., Connection Machine)

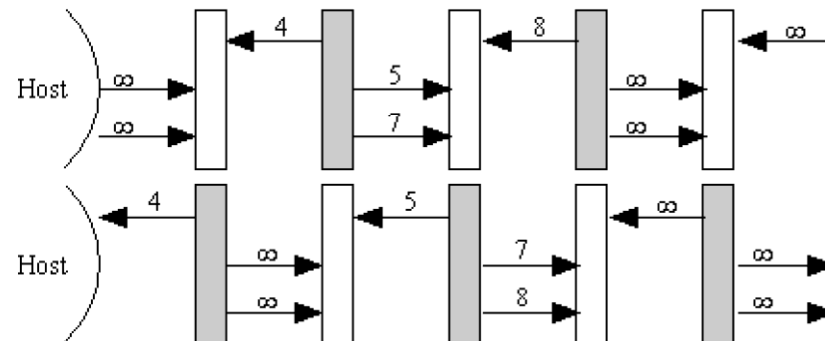
# MISD

- Multiple Instruction, Single Data
- Example: systolic array

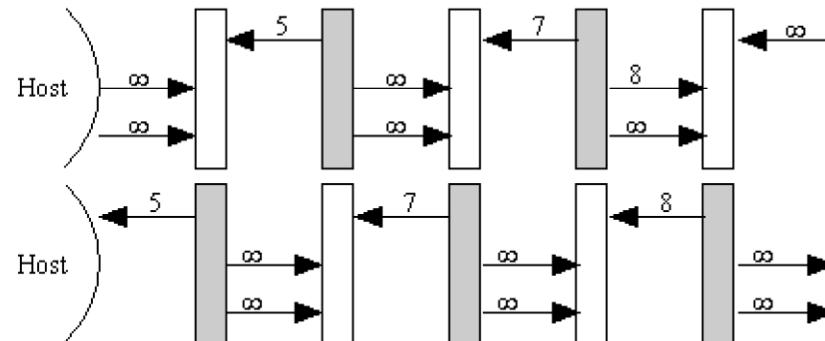
Insert 7



Extract Minimum



Extract Minimum



# MIMD

---

- Multiple Instruction, Multiple Data
- Multiple-CPU computers
  - Multiprocessors
  - Multicomputers

# Summary

---

- Commercial parallel computers appeared in 1980s
- Multiple-CPU computers now dominate
- Small-scale: Centralized multiprocessors
- Large-scale: Distributed memory architectures (multiprocessors or multicomputers)



## Exercise 2

---

- 2.14
- 2.19