ECE/CS 250
Computer Architecture

Summer 2023

Multicore

Dan Sorin and Tyler Bletsch
Duke University

Outline

- **Why multicore?**
- **Thread-level parallelism**
- Multithreaded cores
- Multiprocessors
- Design issues
- Examples

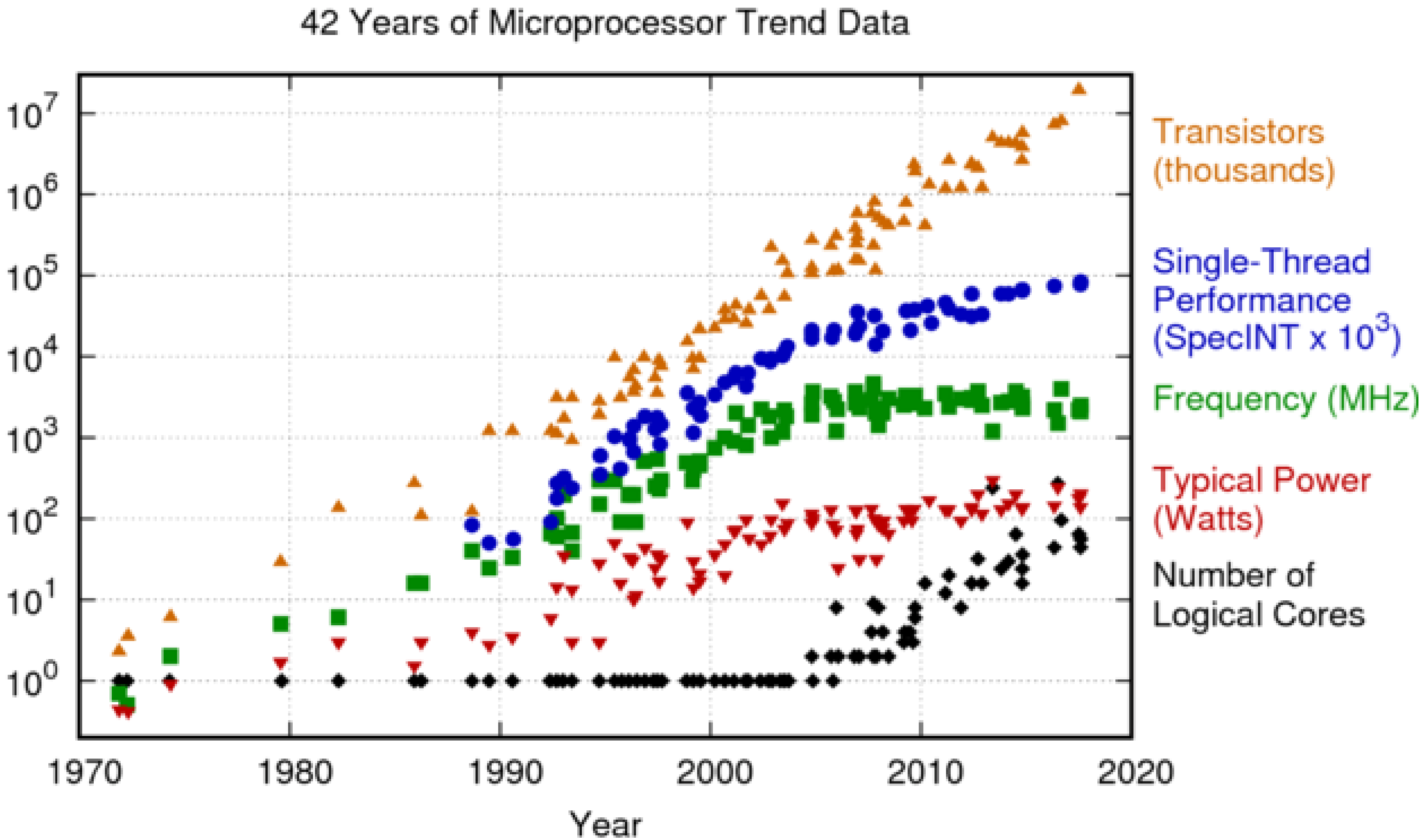
Readings

- Patterson and Hennessy
  - Chapter 6

Why Multicore?

- Why is everything now multicore?
  - This is a fairly new trend
- Reason #1: Running out of “ILP” that we can exploit
  - Can’t get much better performance out of a single core that’s running a single program at a time
- Reason #2: Power/thermal constraints
  - Even if we wanted to just build fancier single cores at higher clock speeds, we’d run into power and thermal obstacles
- Reason #3: Moore’s Law
  - Lots of transistors → what else are we going to do with them?
  - Historically: use transistors to make more complicated cores with bigger and bigger caches
  - But this strategy has run into problems

Microprocessor Trends (for Intel CPUs)



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

How do we keep multicores busy?

- Single core processors exploit ILP
- Multicore processors exploit TLP: thread-level parallelism
- What's a thread?
  - A program can have 1 or more threads of control
  - Each thread has own PC
  - All threads in a given program share resources (e.g., memory)
- OK, so where do we find more than one thread?
- Option #1: Multiprogrammed workloads
  - Run multiple single-threaded programs at same time
- Option #2: Explicitly multithreaded programs
  - Create a single program that has multiple threads that work together to solve a problem

Parallel Programming

- How do we break up a problem into sub-problems that can be worked on by separate threads?
- ICQ: How would you create a multithreaded program that searches for an item in an array?
- ICQ: How would you create a multithreaded program that sorts a list?

- Fundamental challenges
  - Breaking up the problem into many reasonably sized tasks
    - What if tasks are too small? Too big? Too few?
  - Minimizing the communication between threads
    - Why?

Writing a Parallel Program

- Would be nice if compiler could turn sequential code into parallel code...
  - Been an active research goal for years, no luck yet...
- Can use an explicitly parallel language or extensions to an existing language
  - Map/reduce (Google), Hadoop
  - Pthreads
  - Java threads
  - Message passing interface (MPI)
  - CUDA
  - OpenCL
  - High performance Fortran (HPF)
  - Etc.

Parallel Program Challenges

- Parallel programming is HARD!
  - Why?
- Problem: #cores is increasing, but parallel programming isn't getting easier → how are we going to use all these cores???

HPF Example

```
forall(i=1:100, j=1:200){
   MyArray[i,j] = X[i-1, j] + X[i+1, j];
}
```

// “forall” means we can do all i,j combinations in parallel

// I.e., no dependences between these operations

Some Problems Are “Easy” to Parallelize

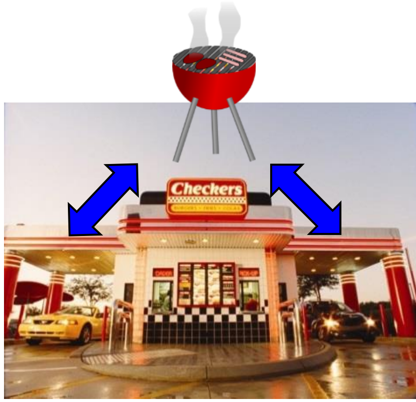
- Database management system (DBMS)
- Web search (Google)
- Graphics
- Some scientific workloads (why?)
- Others??

Outline

- Why multicore?
- Thread-level parallelism
- **Multithreaded cores**
- **Multiprocessors**
- Design issues
- Examples

Multithreaded Cores

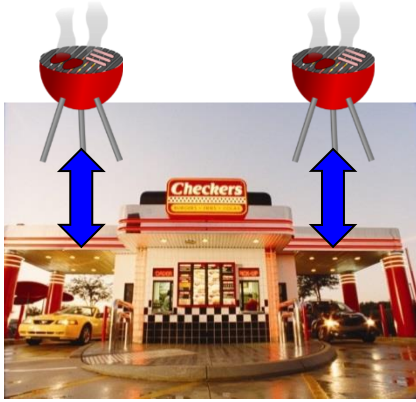
- So far, our core executes one thread at a time
- Multithreaded core: execute multiple threads at a time
- Old idea … but made a big comeback fairly recently
- How do we execute multiple threads on same core?
  - Coarse-grain switching (what the OS does every millisecond or so)
  - Fine-grain switching (what multithreading CPUs can do – cheaper/faster)
  - Simultaneous multithreading (SMT) → "hyperthreading" (Intel)
- Benefits?
  - Better instruction throughput
    - Greater resource utilization
    - Tolerates long latency events (e.g., cache misses)
  - Cheaper than multiple complete cores



Multithreaded: Two drive-throughs being served by one kitchen

Multiprocessors

- Multiprocessors have been around a long time … just not on a single chip
  - Mainframes and servers with 2-64 processors
  - Supercomputers with 100s or 1000s of processors
- Now, multiprocessor on a single chip
  - “multicore processor” (sometimes “chip multiprocessor”)
- Why does “single chip” matter so much?
  - ICQ: What’s fundamentally different about having a multiprocessor that fits on one chip vs. on multiple chips?



Multiprocessor: Two drive-throughs, each with its own kitchen

Outline

- Why multicore?
- Thread-level parallelism
- Multithreaded cores
- Multiprocessors
- **Design issues**
- Examples

Multiprocessor Microarchitecture

- Many design issues unique to multiprocessors
  - Interconnection network
  - Communication between cores
  - Memory system design
  - Others?

Interconnection Networks

- Networks have many design aspects
  - We focus on one design aspect here (topology) → see ECE 552 (CS 550) and ECE 652 (CS 650) for more on this

- Topology is the structure of the interconnect
  - Geometric property → topology has nice mathematical properties

- Direct vs Indirect Networks
  - Direct: All switches attached to host nodes (e.g., mesh)
  - Indirect: Many switches not attached to host nodes (e.g., tree)

Direct Topologies: k-ary d-cubes

- Often called k-ary n-cubes

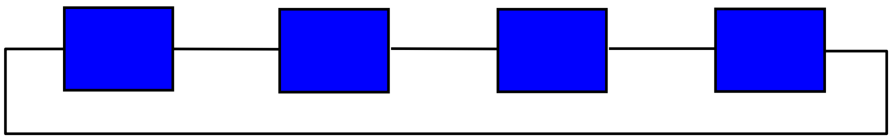
- General class of regular, direct topologies
  - Subsumes rings, tori, cubes, etc.

- d dimensions
  - 1 for ring
  - 2 for mesh or torus
  - 3 for cube
  - Can choose arbitrarily large d, except for cost of switches

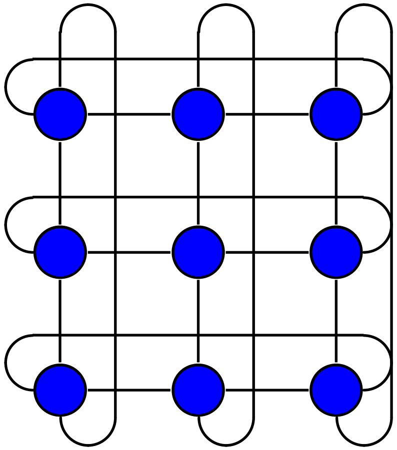
- k switches in each dimension
  - Note: k can be different in each dimension (e.g., 2,3,4-ary 3-cube)

Examples of k-ary d-cubes (for N cores)

- 1D Ring = k-ary 1-cube
  - d = 1 [always]
  - k = N [always] = 4 [here]
  - Ave dist = ?



- 2D Torus = k-ary 2-cube
  - d = 2 [always]
  - k = $\log_d N$ (always) = 3 [here]
  - Ave dist = ?



k-ary d-cubes in Real World

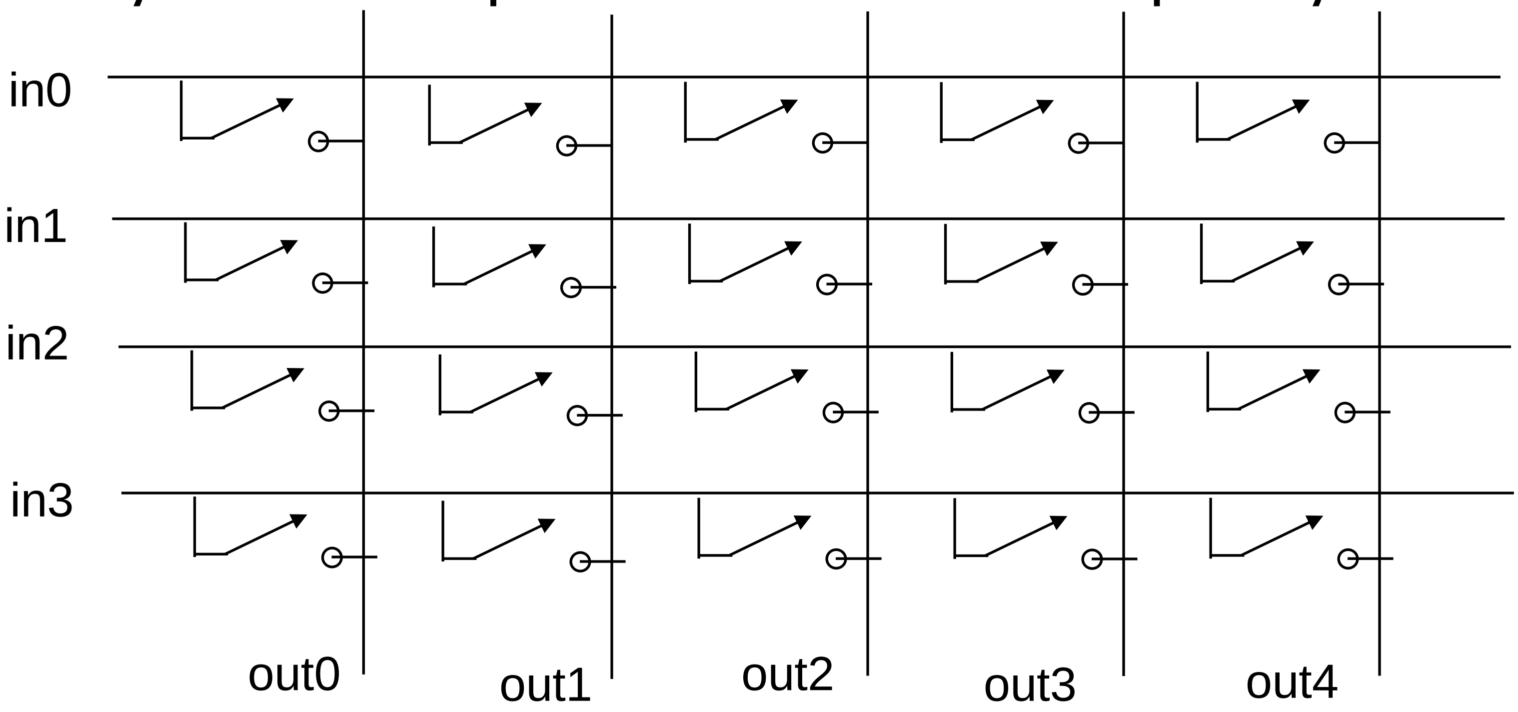
- Compaq Alpha 21364 (and 21464, R.I.P.)
  - 2D torus (k-ary 2-cube)
- Cray T3D and T3E
  - 3D torus (k-ary, 3-cube)
- Intel's MIC (formerly known as Larrabee)
  - 1D ring
- Intel's SandyBridge (one flavor of core i7)
  - 2D mesh

Indirect Topologies

- Indirect topology – most switches not attached to nodes
- Some common indirect topologies
  - Crossbar
  - Tree
  - Butterfly
- Each of the above topologies comes in many flavors

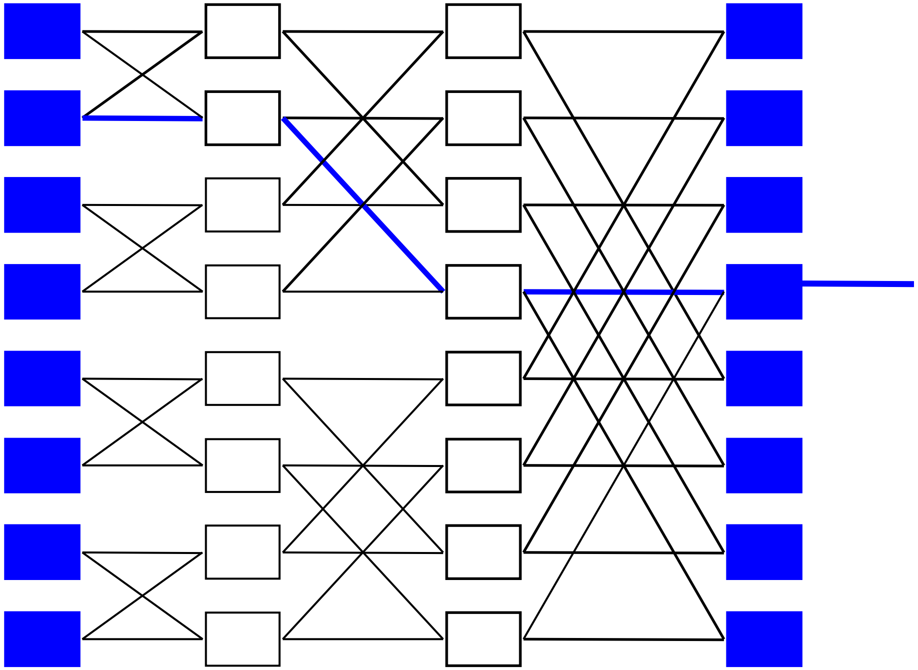
Indirect Topologies: Crossbar

- Crossbar = single switch that directly connects n inputs to m outputs
  - Logically equivalent to m n:1 muxes
- Very useful component that is used frequently



Indirect Topologies: Butterflies

- **Multistage: nodes at ends, switches in middle**
- **Exactly one path between each pair of nodes**
- **Each node sees a tree rooted at itself**



Indirect Networks in Real World (ancient)

- Thinking Machines CM-5 (really old machine)
  - Fat tree
- Sun UltraEnterprise E10000 (old machine)
  - 4 trees (interleaved by address)
- And lots and lots of buses!

Multiprocessor Microarchitecture

- Many design issues unique to multiprocessors
  - Interconnection network
  - Communication between cores
  - Memory system design
  - Others?

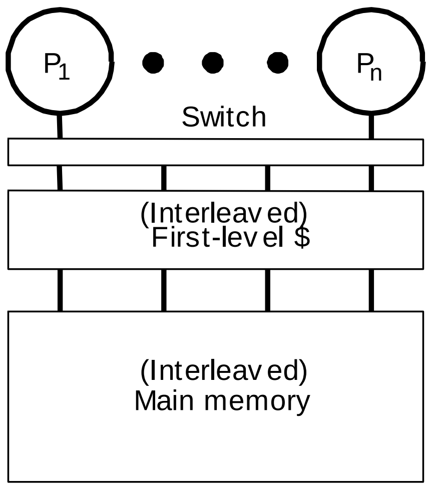
Communication Between Cores (Threads)

- How should threads communicate with each other?
- Two popular options
- Shared memory
  - Perform loads and stores to shared addresses
  - Requires synchronization (can't read before write)
- Message passing
  - Send messages between threads (cores)
  - No shared address space

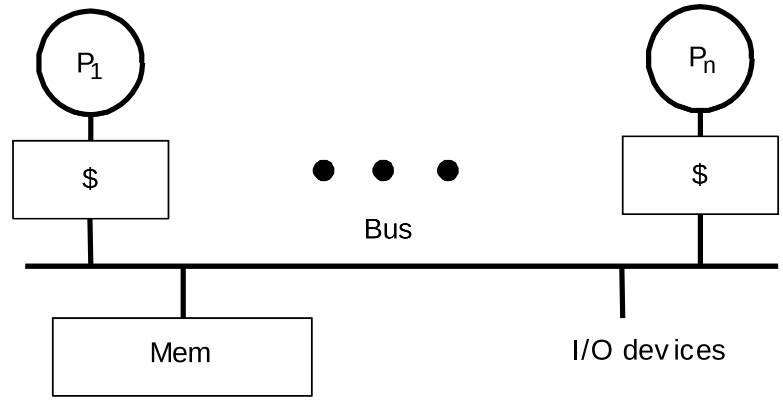
What is (Hardware) Shared Memory?

- Take multiple microprocessors
- Implement a memory system with a single global physical address space (usually)
  - Special HW does the “magic” of cache coherence

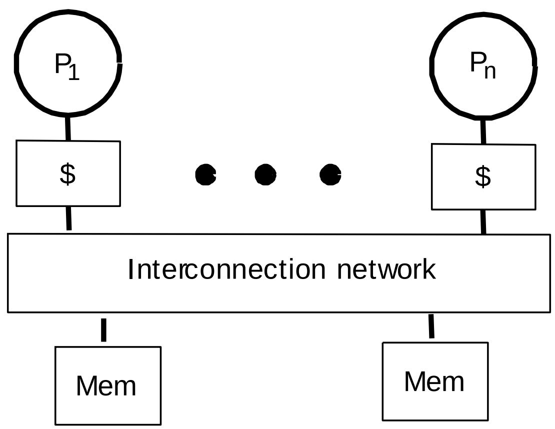
Some (Old) Memory System Options



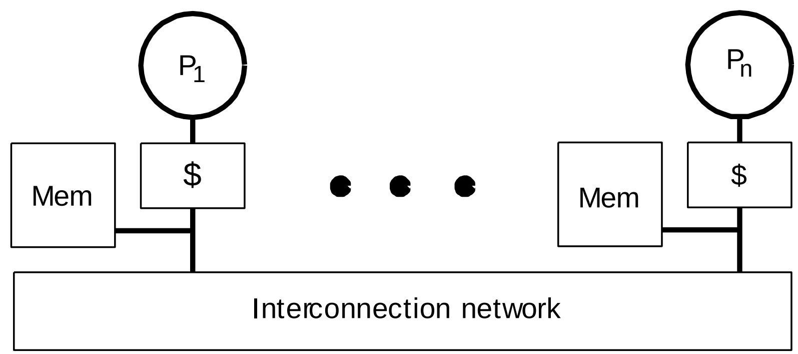
(a) Shared cache



(b) Bus-based shared memory

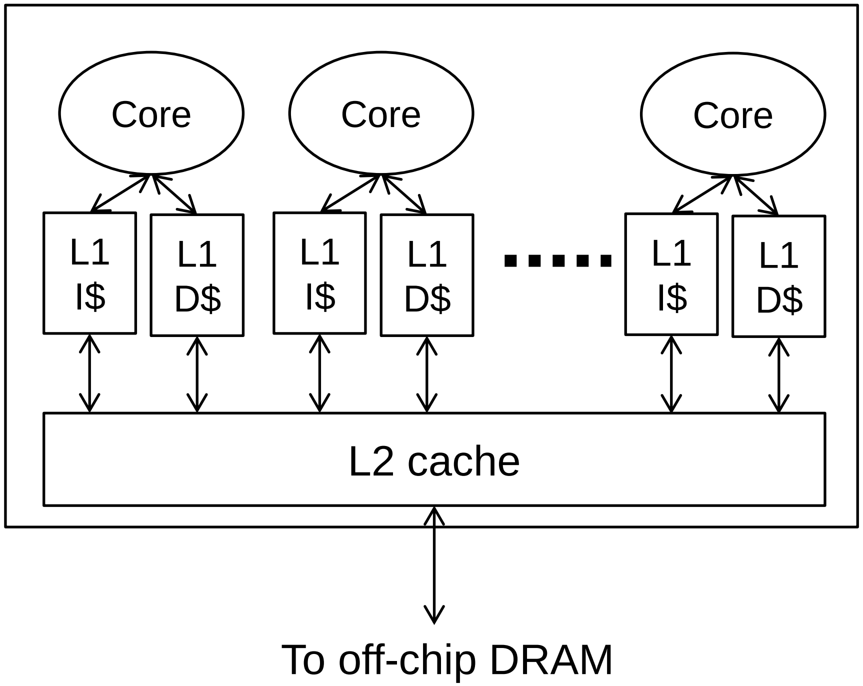


(c) Dancehall



(d) Distributed-memory

A (Newer) Memory System Option



Cache Coherence

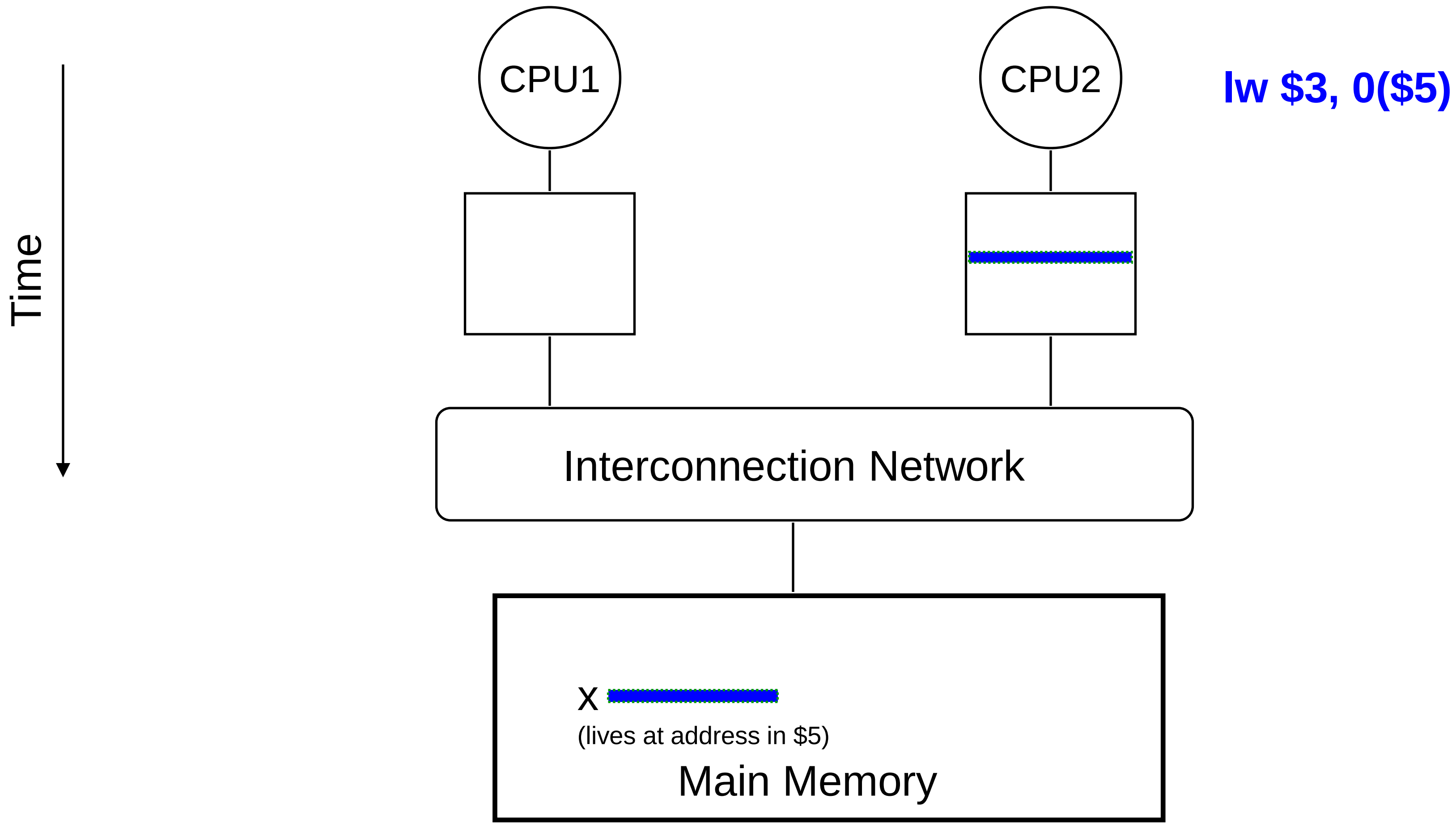
- According to Webster's dictionary …
  - Cache: a secure place of storage
  - Coherent: logically consistent

- Cache Coherence: keep storage logically consistent
  - Coherence requires enforcement of 2 properties per block

1) At any time, only one writer or >=0 readers of block
   - Can't have writer at same time as other reader or writer
2) Data propagates correctly
   - A request for a block gets the most recent value

Cache Coherence Problem (Step 1)

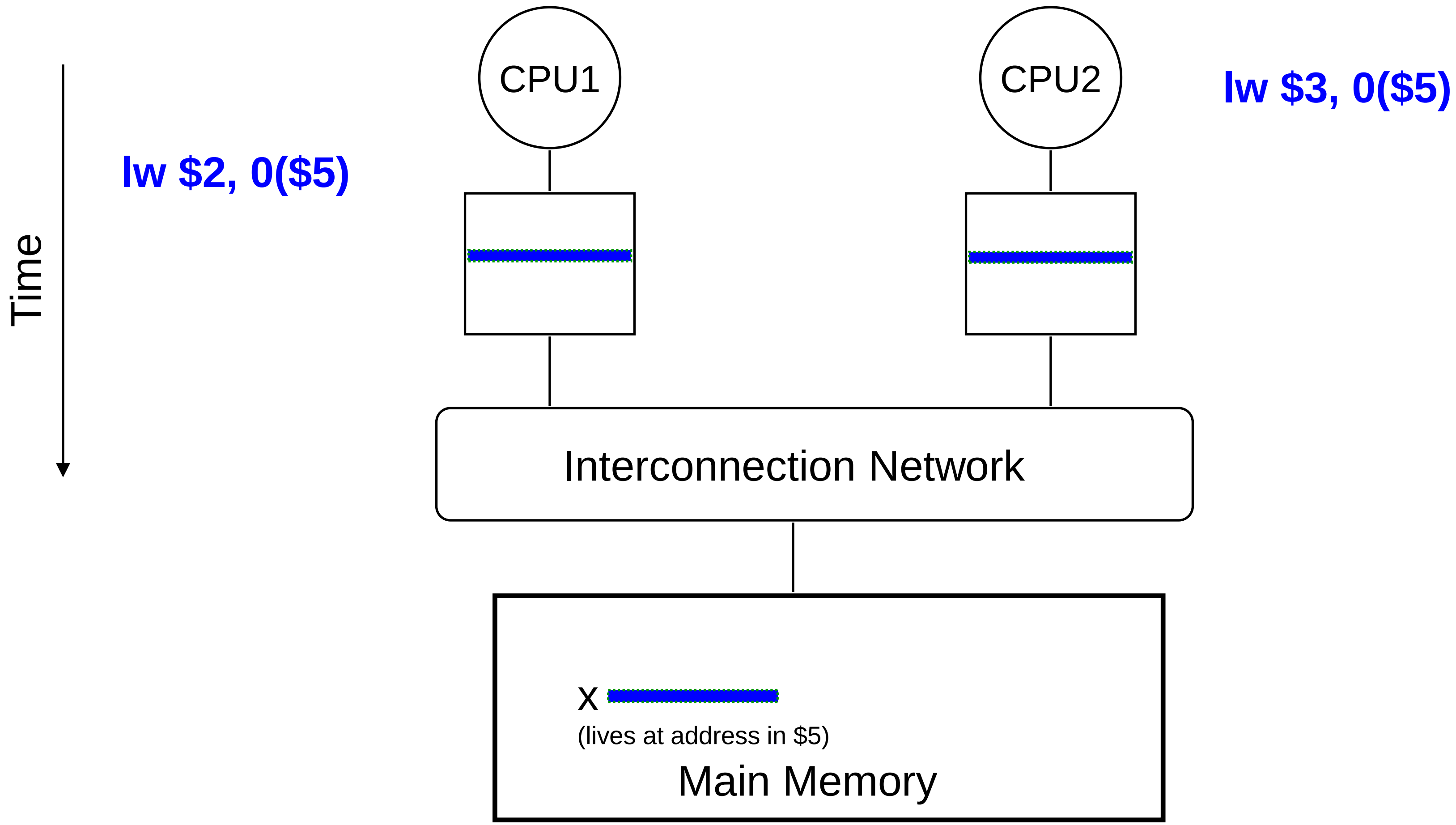
CPU2 loads from address $5, it's a cache miss, so we load that block into CPU2's cache.



Assume $5 is the same in both CPUs and refers to a shared memory address

Cache Coherence Problem (Step 2)

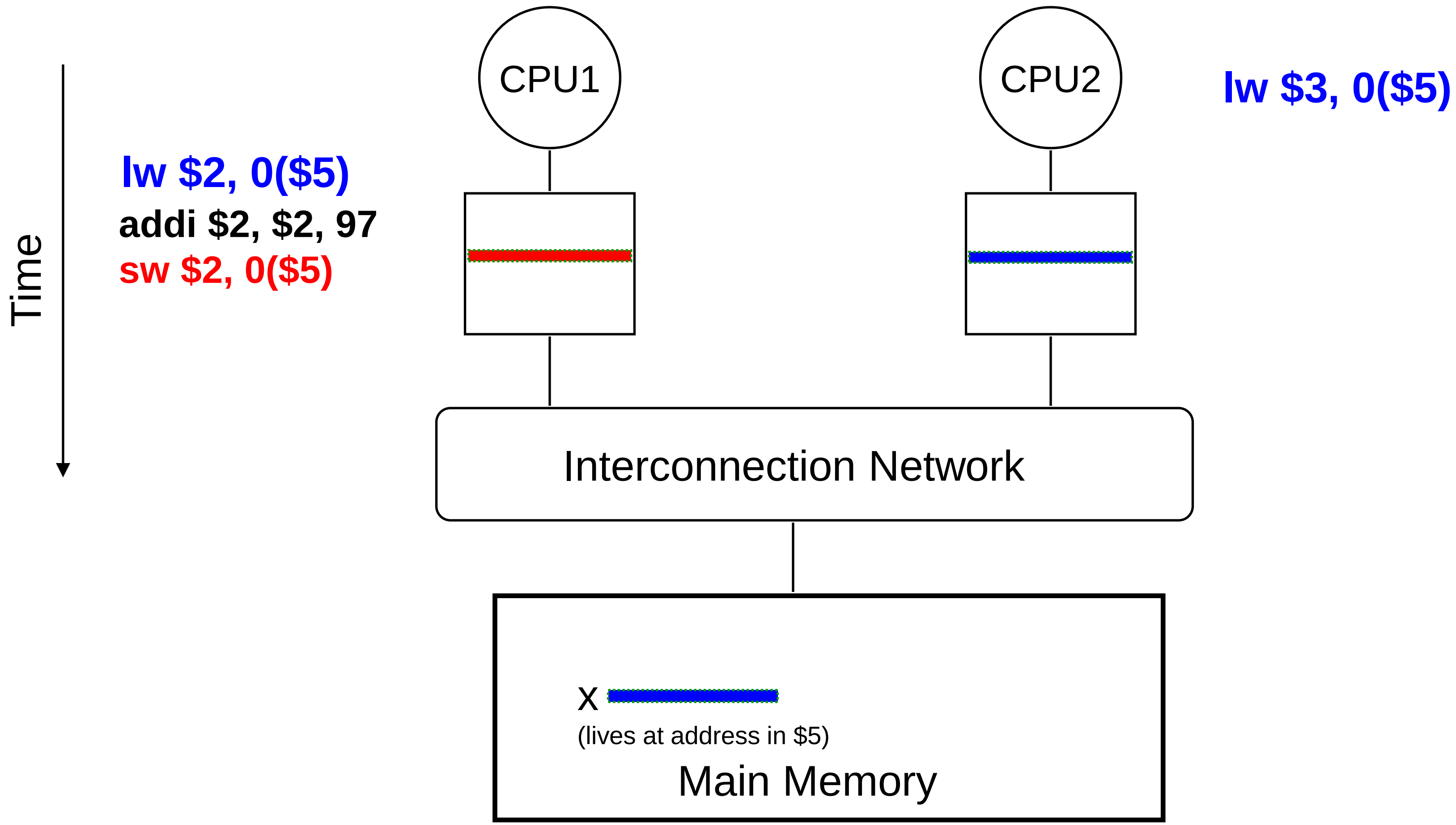
CPU1 also loads from address $5, it's a cache miss, so we load that block into CPU1's cache.



Assume $5 is the same in both CPUs and refers to a shared memory address

Cache Coherence Problem (Step 3a)

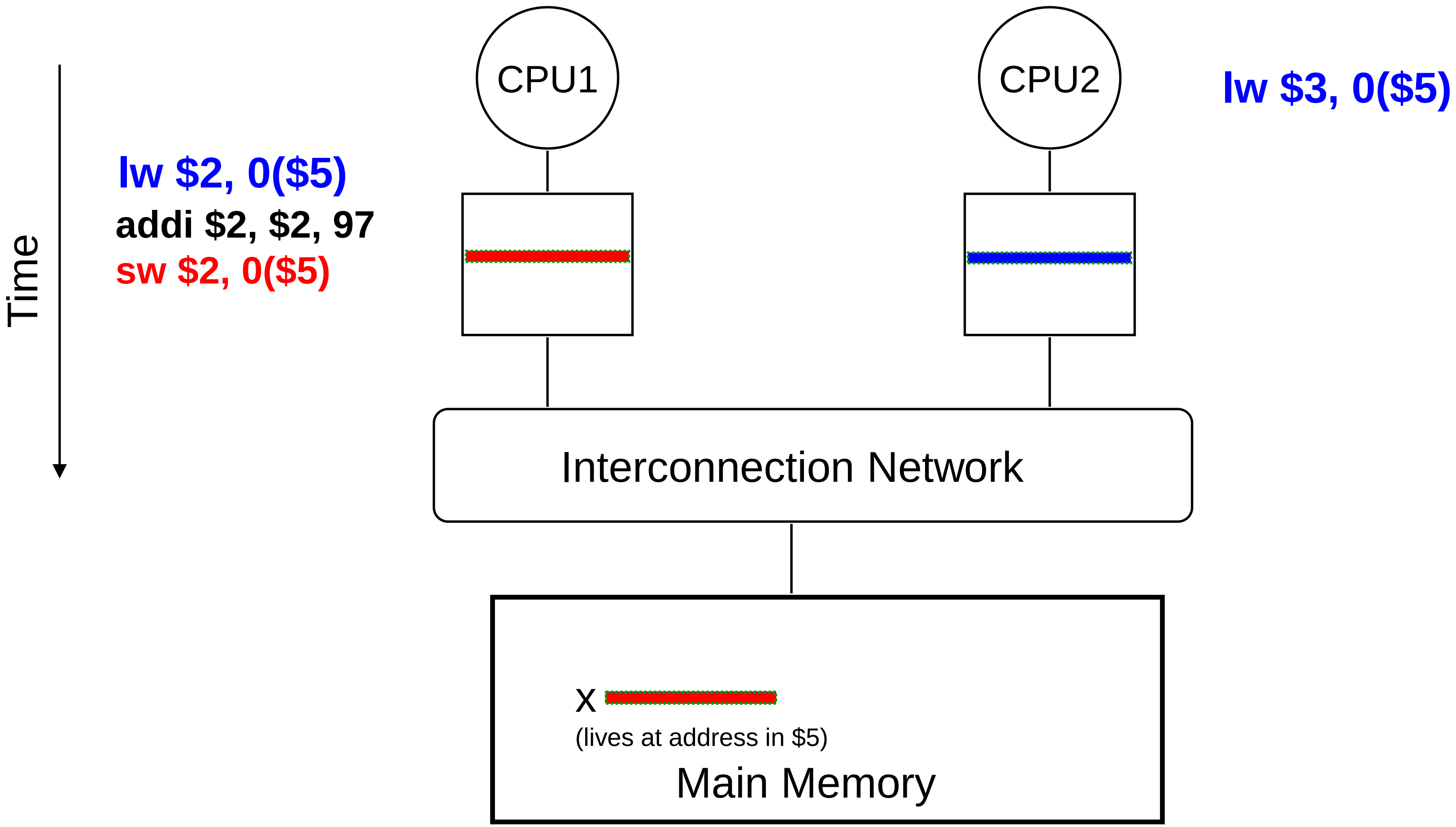
CPU1 also stores a different value into that same memory location.
If it's a write-back cache, then only the cache changes.



Assume $5 is the same in both CPUs and refers to a shared memory address

Cache Coherence Problem (Step 3b)

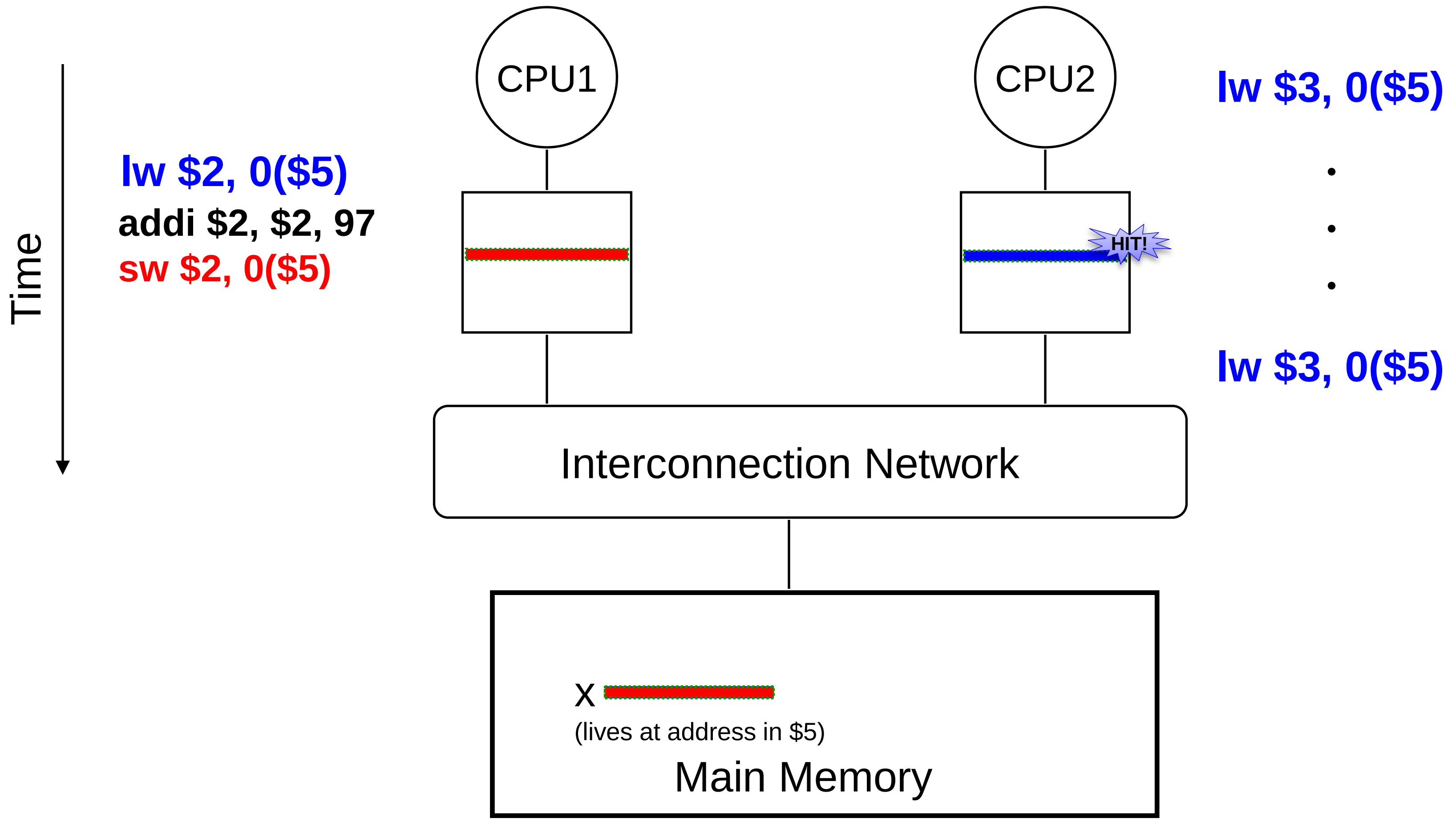
CPU1 also stores a different value into that same memory location.
If it's a <u>write-through</u> cache, then memory also changes.
The cache coherence problem will occur either way!



Assume $5 is the same in both CPUs and refers to a shared memory address

Cache Coherence Problem (Step 4)

CPU2 loads the thing at address $5 again, and it's a cache hit, so we get the OLD value!
PROBLEM!! CPU2's cache is stale!!
The correct value is in CPU1's cache (if write-back) or main memory (if write-through, as shown).



Assume $5 is the same in both CPUs and refers to a shared memory address

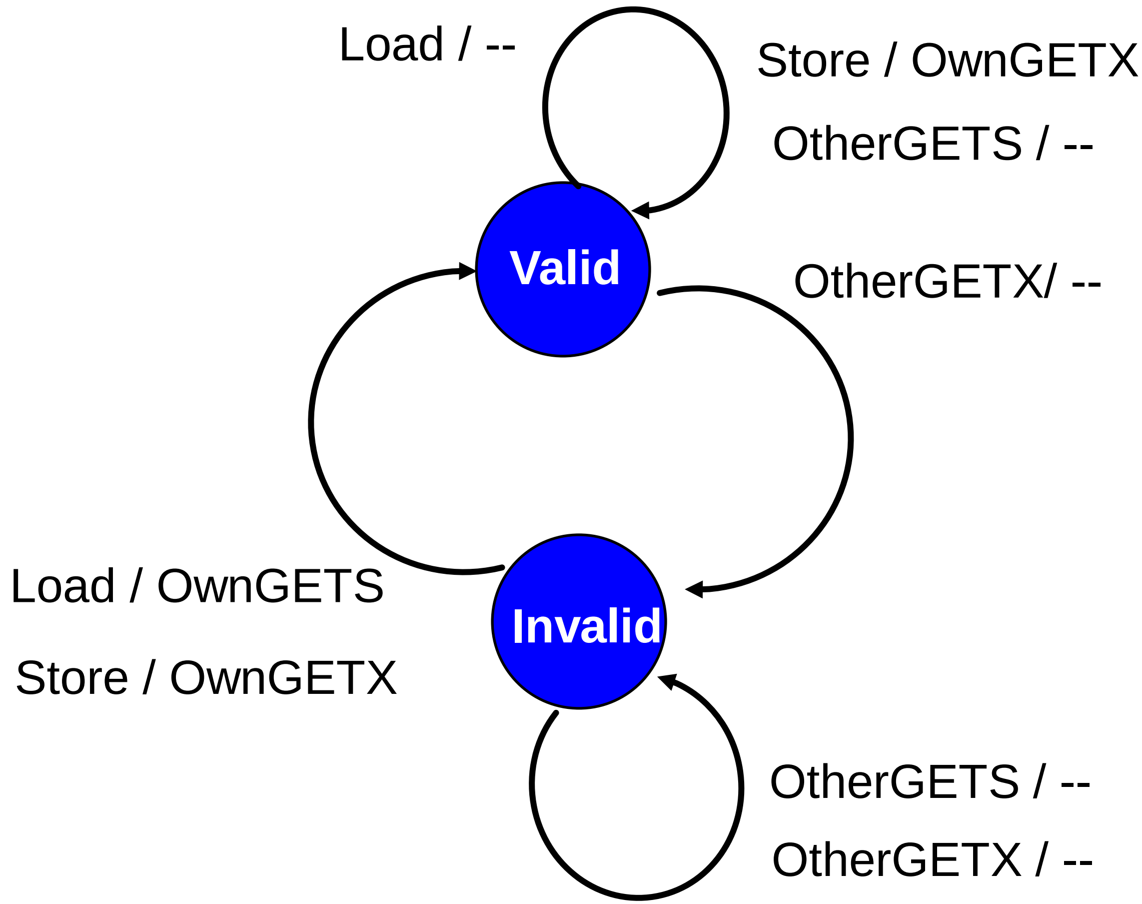
Snooping Cache-Coherence Protocols

- Each cache controller "snoops" all bus transactions
  - Transaction is relevant if it is for a block this cache contains
  - Take action to ensure coherence
    - Invalidate
    - Update
    - Supply value to requestor if Owner
  - Actions depend on the state of the block and the protocol
- Main memory controller also snoops on bus
  - If no cache is owner, then memory is owner
- Simultaneous operation of independent controllers

Processor and Bus Actions

- Processor:
  - Load
  - Store
  - Writeback on replacement of modified block
- Bus
  - GetShared (GETS): Get without intent to modify, data could come from memory or another cache
  - GetExclusive (GETX): Get with intent to modify, must invalidate all other caches' copies
  - PutExclusive (PUTX): cache controller puts contents on bus and memory is updated
  - Definition: cache-to-cache transfer occurs when another cache satisfies GETS or GETX request
- Let's draw it!

Simple 2-State Invalidate Snooping Protocol



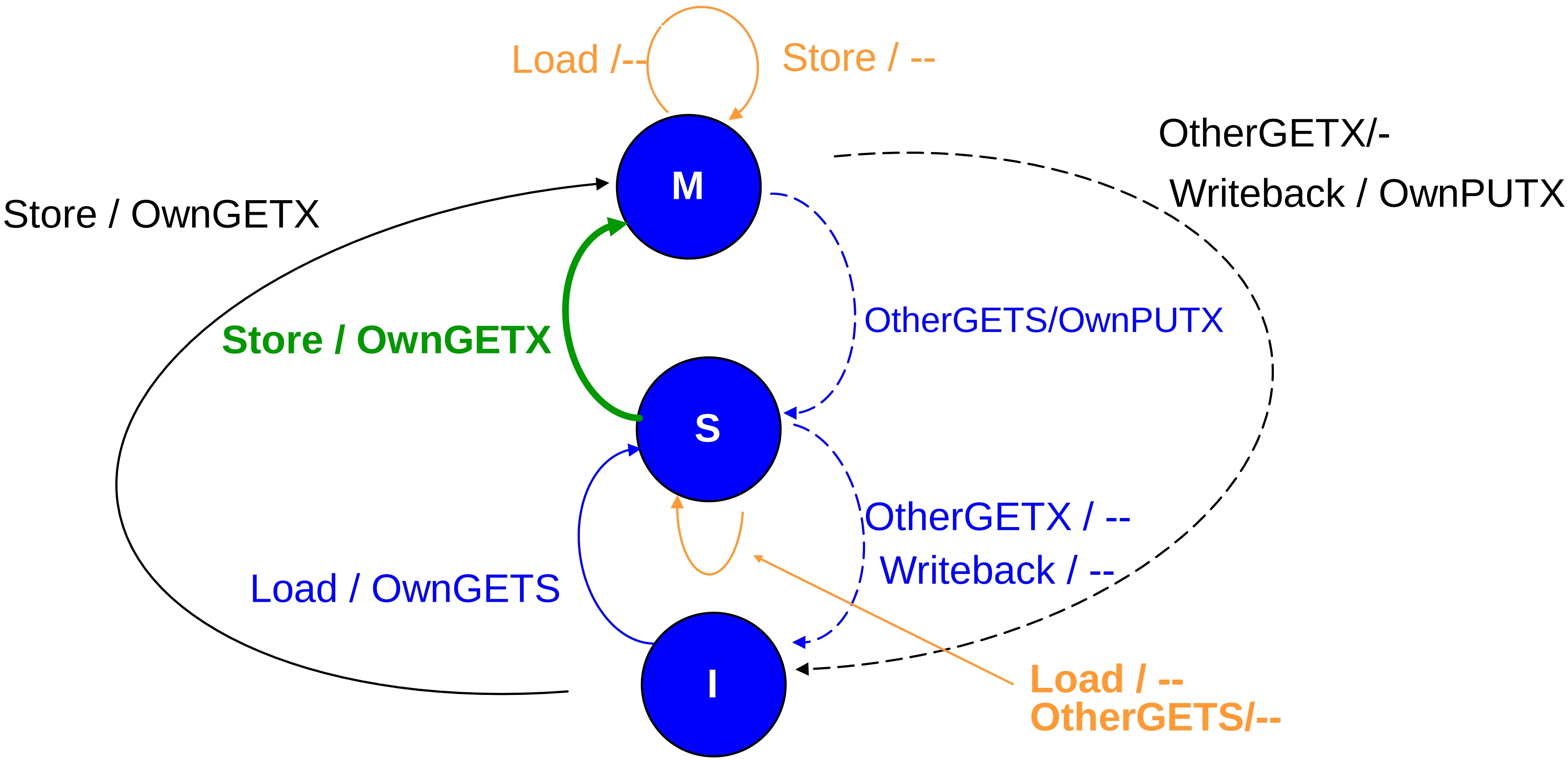
- Write-through, no-write-allocate cache
- Proc actions: Load, Store
- Bus actions: GETS, GETX

Notation: *observed event / action taken*

A 3-State Write-Back Invalidation Protocol

- 2-State Protocol
  - + Simple hardware and protocol
  - Uses lots of bandwidth (every write goes on bus!)
- 3-State Protocol (MSI)
  - Modified
    - One cache exclusively has valid (modified) copy ➔ Owner
    - Memory is stale
  - Shared
    - >= 1 cache and memory have valid copy (memory = owner)
  - Invalid (only memory has valid copy and memory is owner)
- Must invalidate all other copies before entering Modified state
- Requires bus transaction (order and invalidate)

MSI State Diagram



Note: we never take any action on an OtherPUTX

An MSI Protocol Example

Proc Action	P1 State	P2 state	P3 state	Bus Act	Data from
	initially	I	I		I
1. P1 load u	I→S	I	I	GETS	Memory
2. P3 load u	S	I	I→S	GETS	Memory
3. P3 store u	S→I	I	S→M	GETX	Memory or P1 (?)
4. P1 load u	I→S	I	M→S	GETS	P3's cache
5. P2 load u	S	I→S	S	GETS	Memory

- Single writer, multiple reader protocol
- Why Modified to Shared in line 4?
- What if not in any cache? Memory responds
- Read then Write produces 2 bus transactions
  - Slow and wasteful of bandwidth for a common sequence of actions

Outline

- Why multicore?
- Thread-level parallelism
- Multithreaded cores
- Multiprocessors
- Design issues
- **Examples**

Some Real-World Multicores

- Intel/AMD 2/4/8/12/16-core chips
  - Pretty standard
- Sun's Niagara (UltraSPARC T1-T3)
  - 4-16 simple, in-order, multithreaded cores
- Sun's Rock processor: 16 cores
- Cell Broadband Engine: in PlayStation 3
- Intel's MIC/Larrabee chip: 80 simple x86 cores in a ring
- Cisco CRS-1 Processor: 188 in-order cores
- Graphics processing units (GPUs): hundreds of "cores"

Understanding parallel performance

- Common metric: **CPU utilization (aka CPU%)**

  CPU% = Time spent actively computing / Real time

- Example: The kernel sees that a process ran 35ms out of the last 100ms

  → Utilization = 35%

- If you have multiple cores, a process could use time on more than one core at once – how to account?

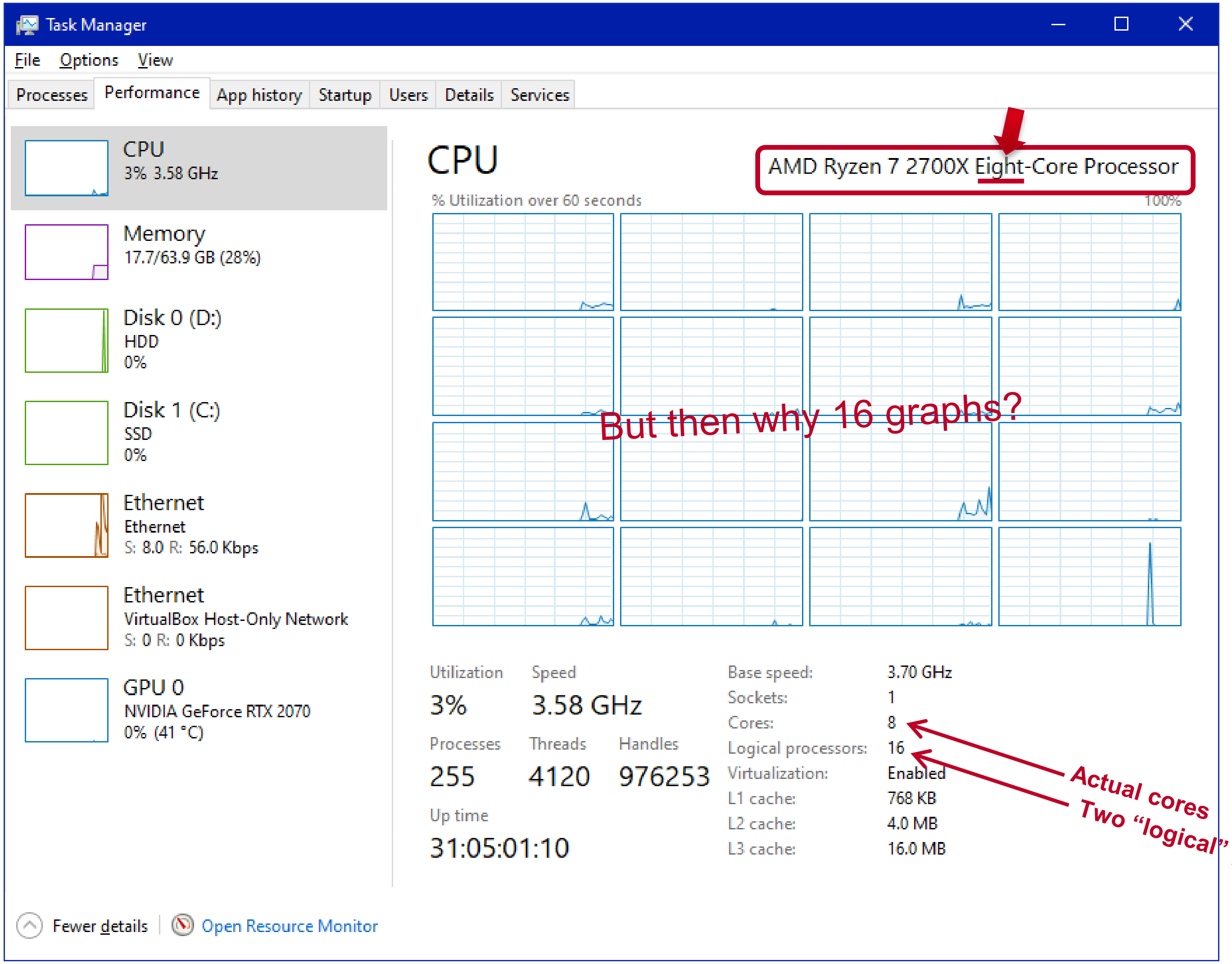
  - **Linux**: CPU% = Sum of time on all cores / Real time
    - Can go over 100% for multithreaded processes
    - Process at 100%? Probably single-threaded and going flat out

  - **Windows**: CPU% = Sum of time on all cores / (Real time * #cores)
    - Will never go over 100%
    - Single-threaded process going flat out shows as (100%/#cores)!
      E.g., 12.5% for an 8-core machine

Understanding CPU time

	Linux: 100% = one core		Windows: 100% = all cores	
Program behavior	**Utilization if you have 1 core**	**Utilization if you have 8 cores**	**Utilization if you have 1 core**	**Utilization if you have 8 cores**
One thread: while (1) {}	100%	100%	100%	12.5%
Two threads: while(1) {}	100% (performance is ~50%)	200%	100% (performance is ~50%)	25%
Eight threads: while(1) {}	100% (performance is ~12.5%)	800%	100% (performance is ~12.5%)	100%
One thread: Busy for 1ms, then idle 1ms	50%	50%	50%	6.25%
Eight threads: Busy for 1ms, then idle 1ms	100% (performance is ~25%)	400%	100% (performance is ~25%)	50%

But what about Symmetric Multi-Threading (SMT)?



- Impact of going from 7 to 8 busy threads?
  - Little impact
  - New thread gets its own core to go crazy on
- Impact of going from 8 to 9 busy threads?
  - Big impact!
  - New thread now has to time-share with competing threads

Reminder: SMT threads offer modest performance benefit: they aren't whole cores, just additional fetchers!

Parallel performance is complicated

- So just looking at CPU performance is complicated
- Just the start… Writing efficient parallel programs is hard!

- That's why we have a whole course on it:
  ECE 565: Performance Optimization & Parallelism