Shared Memory Multiprocessors

Mark Greenstreet

CpSc 418 – Jan. 27, 2016

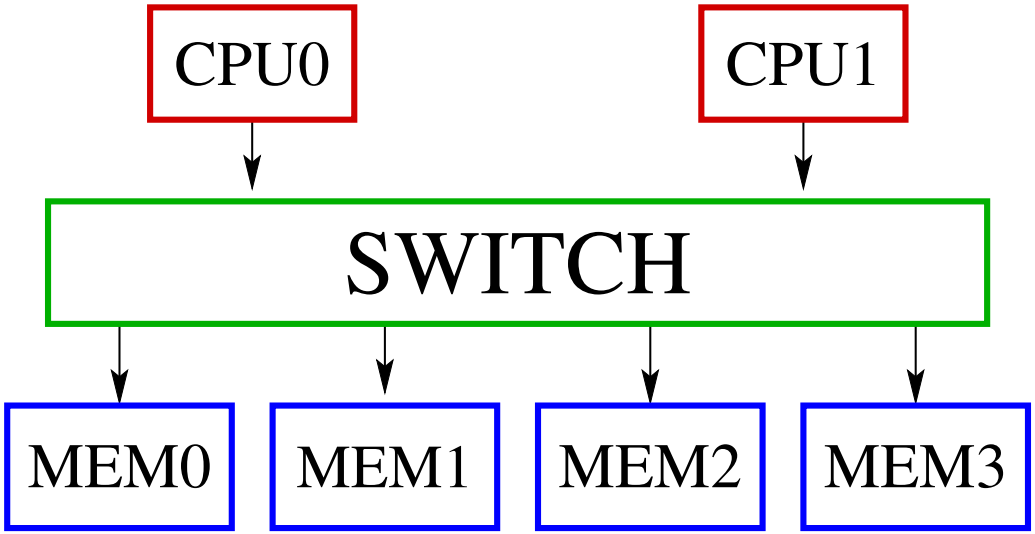
Outline:

- Shared-Memory Architectures
- Memory Consistency
- Examples

Objectives

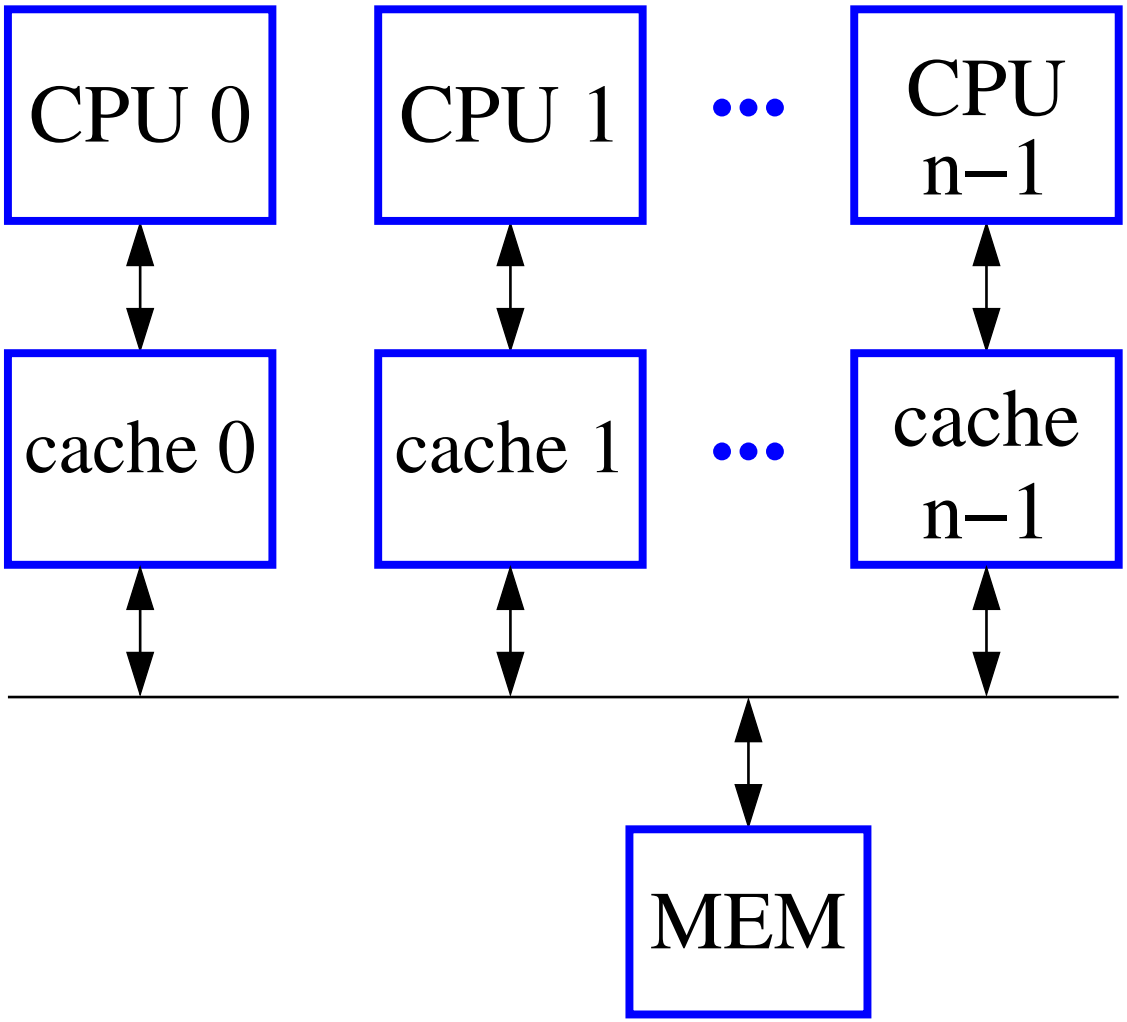
- Understand how processors can communicate by sharing memory.
- Able to explain the term “sequential consistency”
  - Describe a simple cache-coherence protocol, MESI
  - Describe how the protocol can be implemented by snooping.
  - Be aware that real machines make guarantees that are weaker than sequential consistency.

An Ancient Shared-Memory Machine



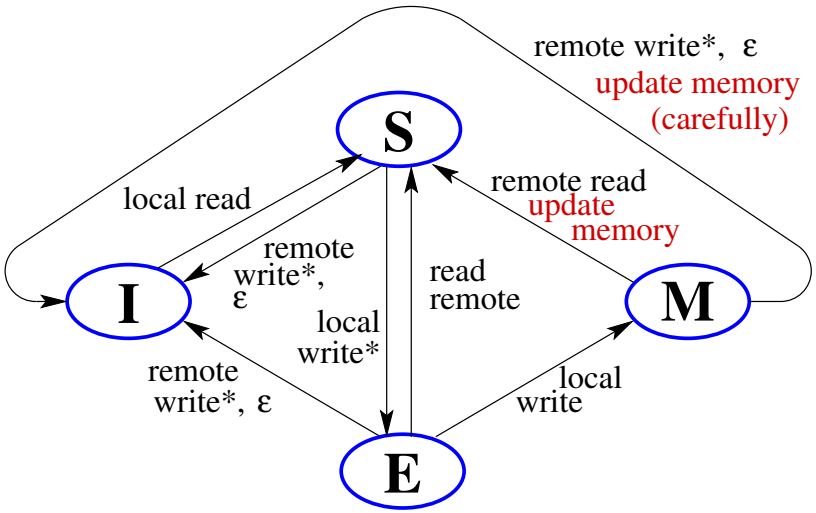
- Multiple CPU's (typically two) shared a memory
- If both attempted a memory read or write at the same time
  - One is chosen to go first.
  - Then the other does its operation.
  - That's the role of the switch in the figure.
- By using multiple memory units (partitioned by address), and a switching network, the memory could keep up with the processors.
- But, now that processors are 100's of times faster than memory, this isn't practical.

A Shared-Memory Machine with Caches



- Caches reduce the number of main memory reads and writes.
- But, what happens when a processor does a write?

The MESI protocol



I = invalid
S = shared
E = exclusive
M = modified

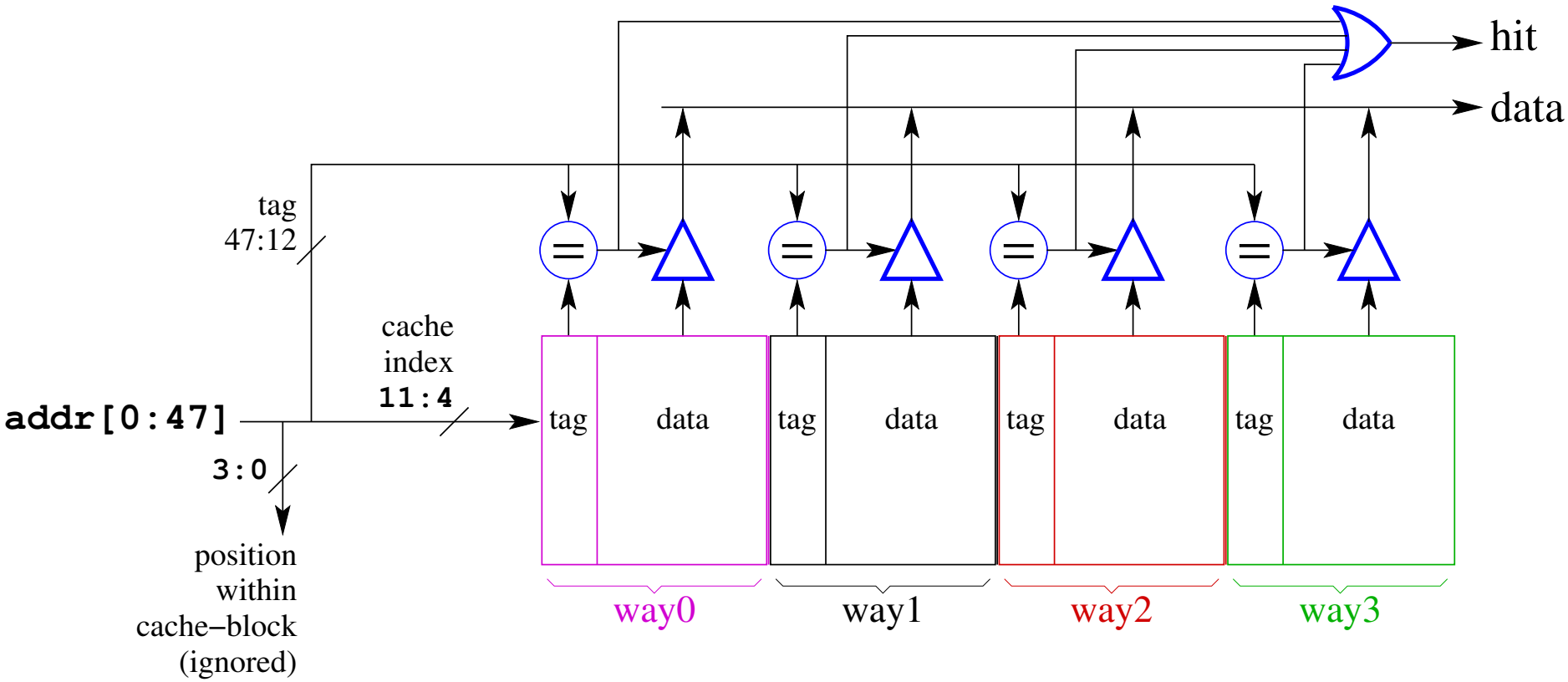
write* = write–through (to memory)

write = write–back (local–cache only)

ε = "spontaneous" transition

- Caches can share read-only copies of a cache block.
- When a processor writes a cache block, the first write goes to main memory.
  - The other caches are notified and invalidate their copies.
  - This ensures that writeable blocks are exclusive.

A typical cache

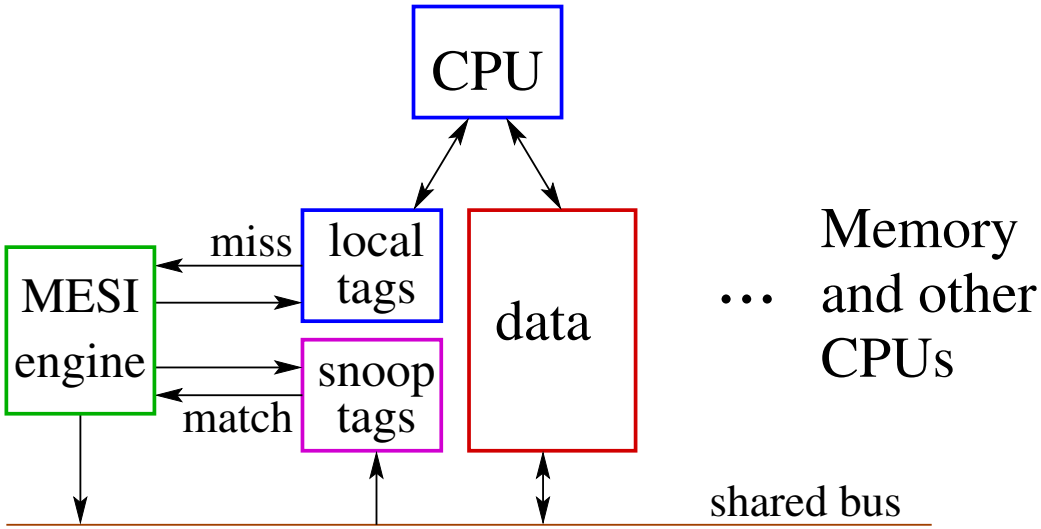


- Only the read-path is shown. Writing is similar.
- This is a 16K-byte, 4-way set-associative cache, with 16 byte cache blocks.

How the cache works

- Read:
  - The address is divided into three pieces: block-offset, cache-index, and tag.
  - The index is used to look up one entry in each “way”.
  - The tags from each way are compared with the tag of the address:
    - If any tag matches, that way provides the data.
    - If no tags match, then a cache miss occurs.
    - Some current block is evicted from the cache to make room for the incoming block.
  - The block-offset determines which bytes from the cache block are returned to the CPU.
- Writes are similar to reads.

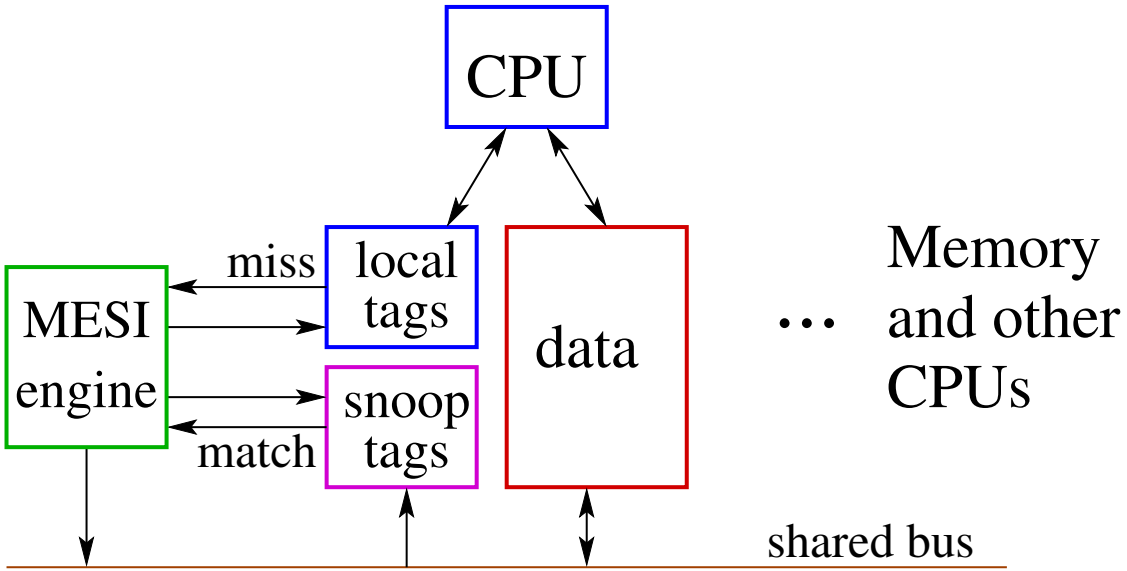
Snooping caches (part 1 of 2)



Each cache has two copies of the tags.

- One copy is used for operations by the local processor.
- The other copy is used to monitor operations on the main memory bus.
  - if another processor attempts to read a block which we have in the `exclusive` or `modified` state, we provide the data (and update main memory).
  - if another processor attempts to write a block that we have, we invalidate our block (updating main memory first if our copy was in the `modified` state.

Snooping caches (part 2 of 2)



Pros and cons:

- Fairly easy to implement.
- Doesn't scale to large numbers of processors.
  - All cache misses processed on the same bus.
  - Engineering marvels push this with multi-level caches and multiple buses, but it get very expensive, and still doesn't scale to 1000s of processors.

Directory schemes

- Main memory keeps a copy of the data and
  - a bit-vector that records which processors have copies, and
  - a bit to indicate that one processor has a copy and it may be modified.
- A processor accesses main memory as required by the MESI protocol.
  - The memory unit sends messages to the other CPUs to direct them to take actions as needed by the protocol.
  - The ordering of these messages ensures that memory stays consistent.

Sequential Consistency

Memory is said to be sequentially consistent if

- All memory reads and writes from all processors can be arranged into a single, sequential order, such that:
  - The operations for each processor occur in the global ordering in the same order as they did on the processor.
  - Every read gets the value of the preceding write to the same address.
- Sequential consistency corresponds to what programmers think "ought" to happen.
  - Very similar to "serialiazability" for database transactions.

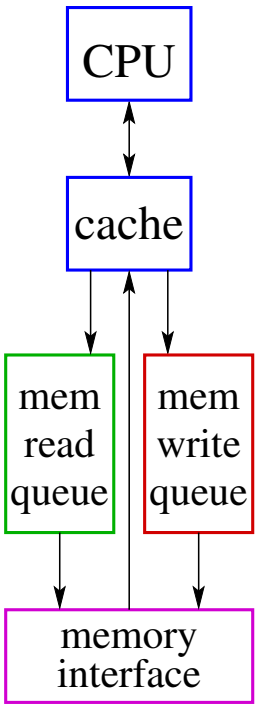
MESI Guarantees Sequential Consistency

- First we prove that at most one processor can have the cache block for any particular address in the `E` or `M` state.
- Define:

$$
\begin{aligned}
& \mathit{value}(\mathit{addr}) \\
& \quad = \quad \mathit{cache}_i(\mathit{addr}).\mathit{data}, \quad \text{if } \exists i.\ \mathit{cache}_i(\mathit{addr}).\mathit{state} \in \{\mathtt{E}, \mathtt{M}\} \\
& \quad = \quad \mathit{MEM}(\mathit{addr}), \quad \text{otherwise}
\end{aligned}
$$

- We can show that every *read*(*addr*) gets the value *value*(*addr*), and that
- We can show *value*(*addr*) gives the value from the most recent write to *addr*.
- Proof: consider each transition in the protocol. Show that if a cache line is in state `S`, `E`, or `M`, the value of that line is the correct one.

Weak Consistency



- A CPU may have multiple cache-misses and protocol operations in-flight at the same time.
- Typically, reads can move ahead of writes to maximize program performance.
- Why?
  - Because there may be instructions waiting for the data from a load.
  - A transition from “shared” to “modified” requires notifying all processors – this can take a long time.
  - Memory writes don’t happen until the instruction commits.
- This means that real computers don’t guarantee sequential consistency.
- But they still make some promises.
  - Look up “Total Store Order” if you want to learn more.

Programming Shared Memory Machines

- We'll do a bit with Java threads at the end of the term.
- Shared memory make parallel programming "easier" because:
  - One thread can pass an entire data structure to another thread just by giving a pointer.
  - No need to pack-up trees, graphs, or other data structures as messages and unpack them at the receiving end.
- Shared memory make parallel programming harder because:
  - It's easy to overlook synchronization (control to shared data structures). Then, we get data races, corrupted data structures, and other hard-to-track-down bugs.
  - A defensive reaction is to wrap *every* shared reference with a lock. But locks are slow (that λ factor for communication), and this often results is slow code, or even deadlock.
- In practice, shared memory code that works often has a message-passing structure.
- Finally, beware of weak consistency
  - Use a thread library.
  - There are elegant algorithms that avoid locking overhead, even with weak consistency, but they are beyond the scope of this class.

Summary

- Shared-Memory Architectures
  - Use cache-coherence protocols to allow each processor to have its own cache while maintaining (almost) the appearance of having one shared memory for all processors.
    - ★ A typical protocol: MESI
    - ★ The protocol can be implemented by snooping or directories.
  - Using cache-memory interconnect for interprocessor communication provides:
    - ★ High-bandwidth
    - ★ Low-latency, but watch out for fences, etc.
    - ★ High cost for large scale machines.
- Shared-Memory Programming
  - Need to avoid interference between threads.
    - ★ Assertional reasoning (e.g. invariants) are crucial, much more so than in sequential programming.
    - ★ There are too many possible interleavings to handle intuitively.
    - ★ In practice, we don't formally prove complete programs, but we use the ideas of formal reasoning.
  - Real computers don't provide sequential consistency.
    - ★ Use a thread library.

Preview

January 25: Superscalars and Simultaneous Multi-Threading	
Reading:	Pacheco, Chapter 2, Sections 2.1 and 2.2.
January 27: Shared-Memory Machines	
Reading:	Pacheco, Chapter 2, Section 2.3
January 29: Distributed-Memory Machines	
Reading:	Pacheco, Chapter 2, Sections 2.4 and 2.5.
February 1: Parallel Performance: Speed-up	
Reading:	Pacheco, Chapter 2, Section 2.6.
Mini-assignment:	Mini 3 goes out (I hope)
Homework:	Homework 2 – early bird deadline
February 3: Parallel Performance: Modeling	
Homework:	Homework 2 – hard deadline
February 5: Matrix Multiplication	
Reading:	Lin & Snyder, Chapter 5, pp. 125–133.
Homework:	Mini 3 due
February 10: Midterm	
February 10: Something fun	

Review

- What is sequential consistency?
- Using the MESI protocol, can multiple processors simultaneously have entries in their caches for the same memory address?
- Using the MESI protocol, can multiple processors simultaneously modify entries in their caches for the same memory address?
- How can a cache-coherence protocol be implemented by snooping?
- How do these issues influence good software design practice?