Multiprocessor Microarchitecture

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

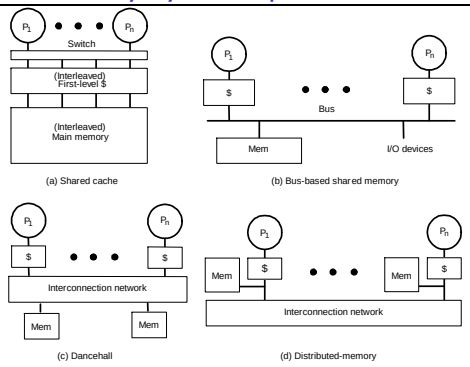
Communication Between Cores (Threads)

- 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)
  - Communication assist HW does the "magic" of cache coherence
- Goal 1: Minimize memory latency
  - Use co-location & caches
- Goal 2: Maximize memory bandwidth
  - Use parallelism & caches

Some Memory System Options



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

1) Write propagation
   - All writes eventually become visible to other processors
2) Write serialization
   - All processors see writes to same block in same order

Why Cache Coherent Shared Memory?

- Pluses
  - For applications - looks like multitasking uniprocessor
  - For OS - only evolutionary extensions required
  - Easy to do communication without OS
  - Software can worry about correctness first and then performance
- Minuses
  - Proper synchronization is complex
  - Communication is implicit so may be harder to optimize
  - More work for hardware designers (i.e., us!)
- Result
  - Symmetric Multiprocessors (SMPs) are the most successful parallel machines ever
  - And the first with multi-billion-dollar markets!

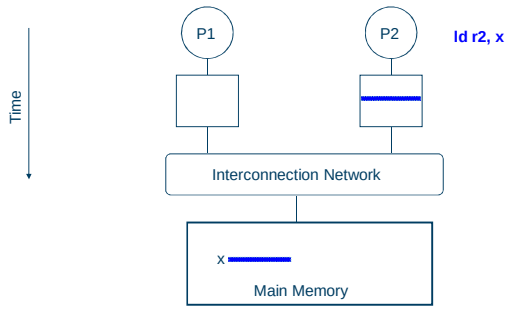
In More Detail

- Efficient naming
  - Virtual to physical mapping with TLBs
- Easy and efficient caching
  - Caching is natural and well-understood
  - Can be done in HW automatically

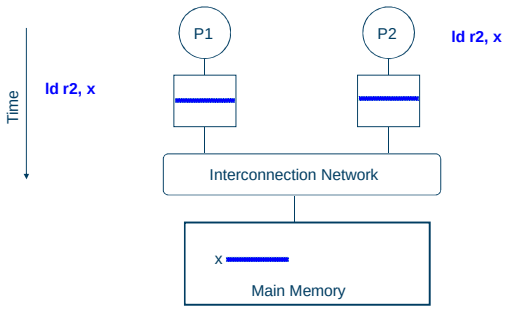
Symmetric Multiprocessors (SMPs)

- Multiple cores
- Each has a cache (or multiple caches in a hierarchy)
- Connect with logical bus (totally-ordered broadcast)
  - Physical bus = set of shared wires
  - Logical bus = functional equivalent of physical bus
- Implement Snooping Cache Coherence Protocol
  - Broadcast all cache misses on bus
  - All caches "snoop" bus and may act (e.g., respond with data)
  - Memory responds otherwise

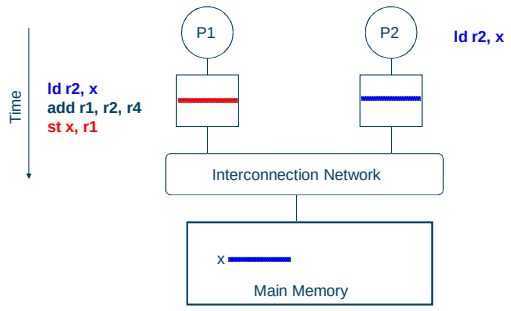
Cache Coherence Problem (Step 1)



Cache Coherence Problem (Step 2)



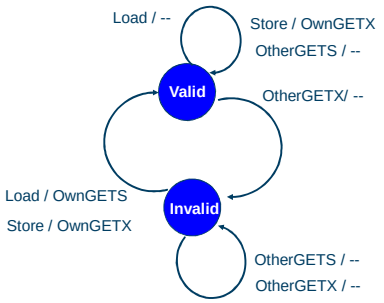
Cache Coherence Problem (Step 3)



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

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*

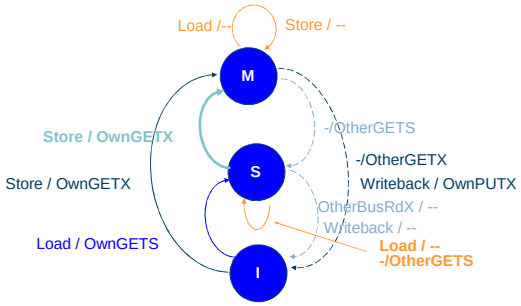
A 3-State Write-Back Invalidation Protocol

- 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 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!

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**

Multicore and Multithreaded Processors

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

Some Real-World Multicores

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