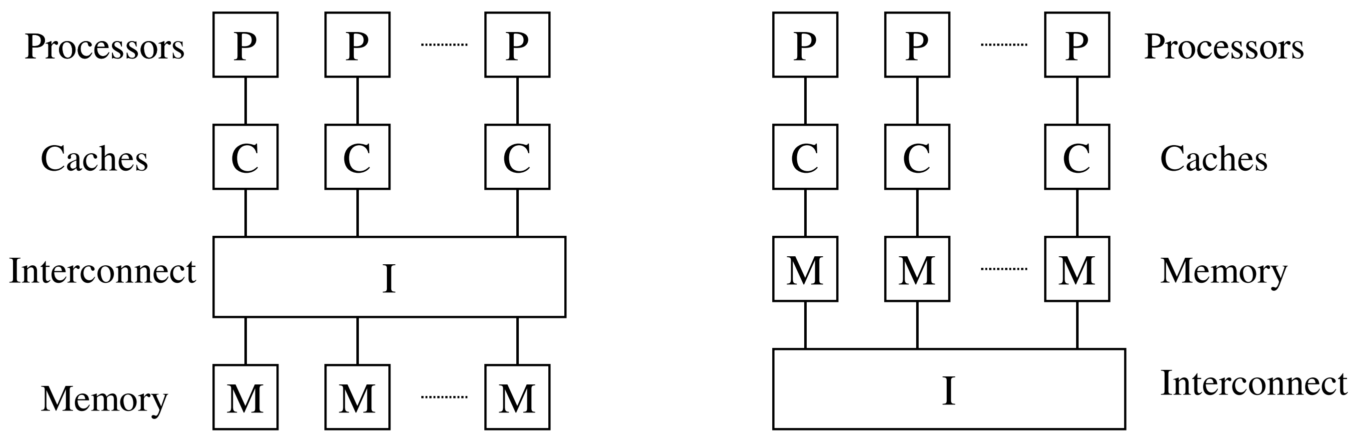
Shared Memory Architecture and Cache Coherency

Outline

- Overview of shared memory architectures
- Interconnects
- Memory hierarchy
- Cache coherence
  - Update vs. invalidate
  - Snoopy vs. directory
- Reading:
  - Chpt2 of Kumar's book

Shared Address Space Machines

- All processors share a single global address space
- Single address space facilitates a simple programming model



(a) UMA (b) NUMA

Example: Shared Address Space Machines

name	Maximum # of procs	Processor name	Processor clock rate
Compaq ProLiant 5000	4	Pentiium Pro	200MHz
Digital AlphaServer 8400	12	Alpha 21164	440MHz
HP 9000 K460	4	PA-8000	180MHz
IBM RS/6000 R40	8	PowerPC 640	112MHz
SGI Power Challenge	36	MIPS R10000	195MHz
Sun Enterprise 6000	30	UltraSPARC 1	167MHz

(a) Characteristics of single-bus multiprocessor for sale in 1997

name	Maximum # of procs	Processor name	Processor clock rate
Cray T3E	2048	Alpha 21164	450MHz
HP/Convex Exemplar	64	PA-8000	180MHz
Sequent NUMA-Q	32	Pentium Pro	200MHz
SGI Origin2000	128	MIPS R10000	195MHz
Sun Enterprise 10000	64	UltraSPARC 1	250MHz

(b) Characteristics of network-connected Multiprocessor for sale in 1997

Shared Address Space Machines

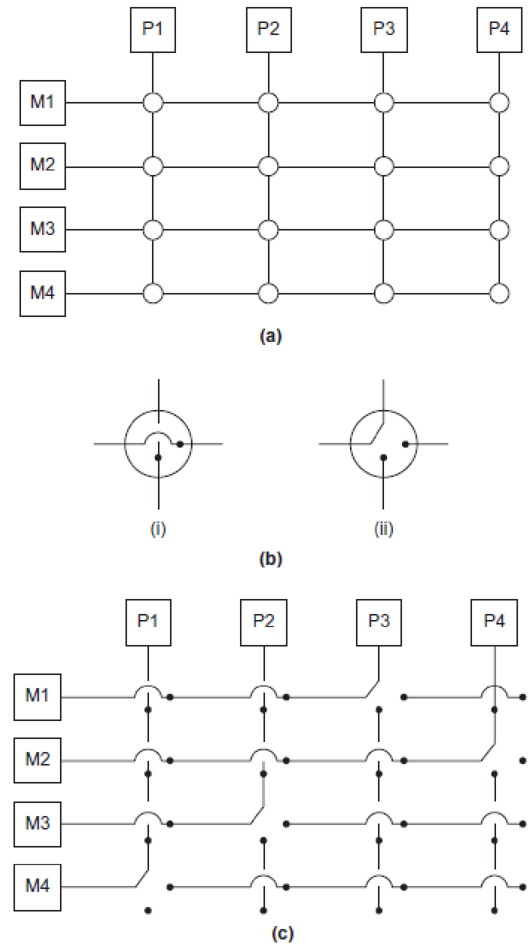
- Important aspects are:
  - Memory organization
  - Interconnect
  - Cache coherence mechanism
- These aspects determine:
  - Performance: programming techniques
  - Scalability
  - Cost

Shared Memory Interconnects

- Bus interconnect
  - A collection of parallel communication wires together with some hardware that controls acess to the bus
  - Communication wires are shared by the devices that are connected to it
  - As the number of devices connected to the bus increases, contention for use of the bus increases, and performance decreases

Shared Memory Interconnects

- Switched interconnect
  - Uses switches to control the routing of data among the connected devices
  - Crossbar
    - Allows simultaneous communication among different devices
    - Faster than buses
    - But the cost of the switches and links is relatively high
    - Figure (a) a 4*4 crossbar
    - Figure (b) configuration of internal switches
    - Figure (c) simultaneous memory accesses by the processors

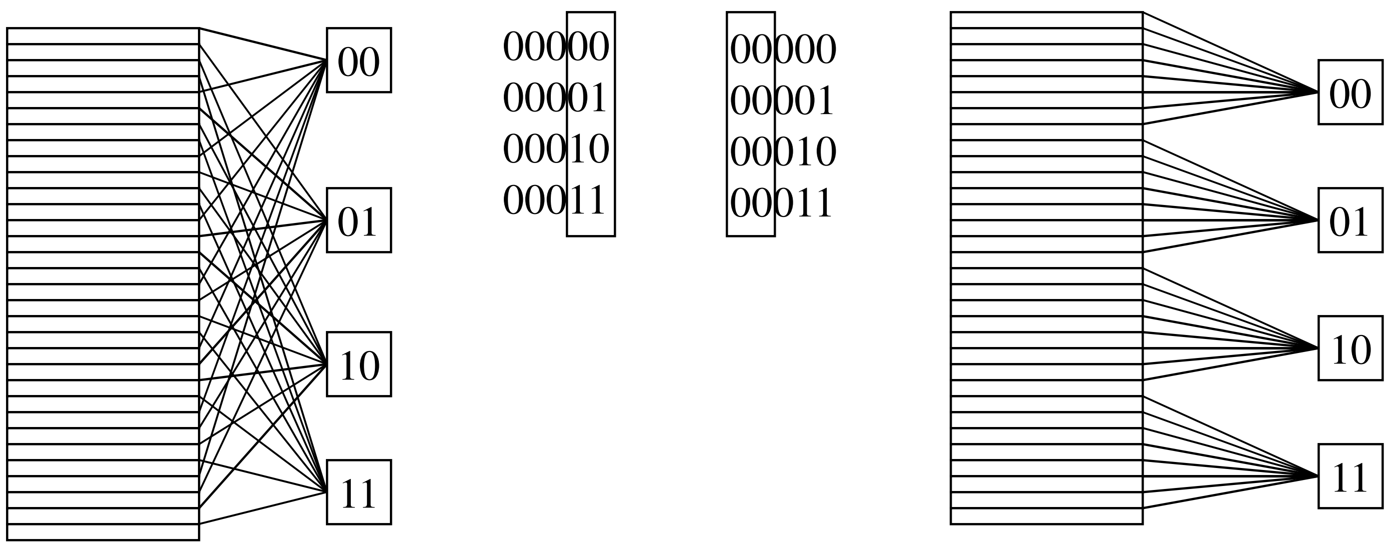


Memory Organization

- Single memory shared among processors causes sequentialization of accesses

- Memory interleaving
  - Splits memory across multiple models (banks)
  - Non-overlapping regions of address space mapped to banks
  - Banks service read/write requests independently

Memory Organization



Low Order Interleaving

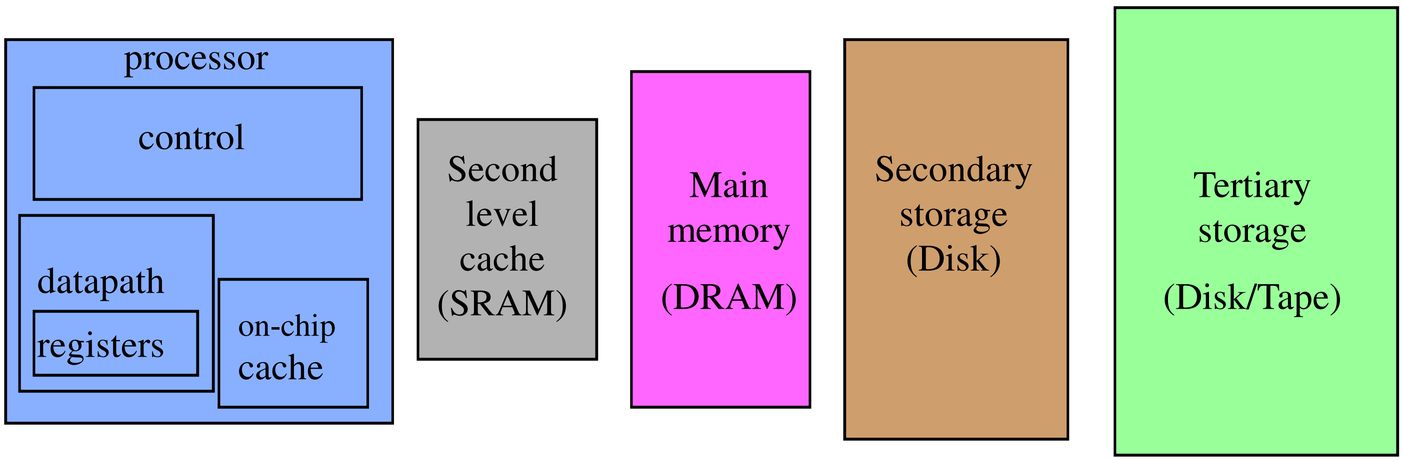
High Order Interleaving

Memory Organization

- Typically, which interleaving is used? Why?
- Programming issues:
  - Must spread accesses across banks to avoid bank conflicts
  - Involves data placement as well as code restructuring

Memory Hierarchy

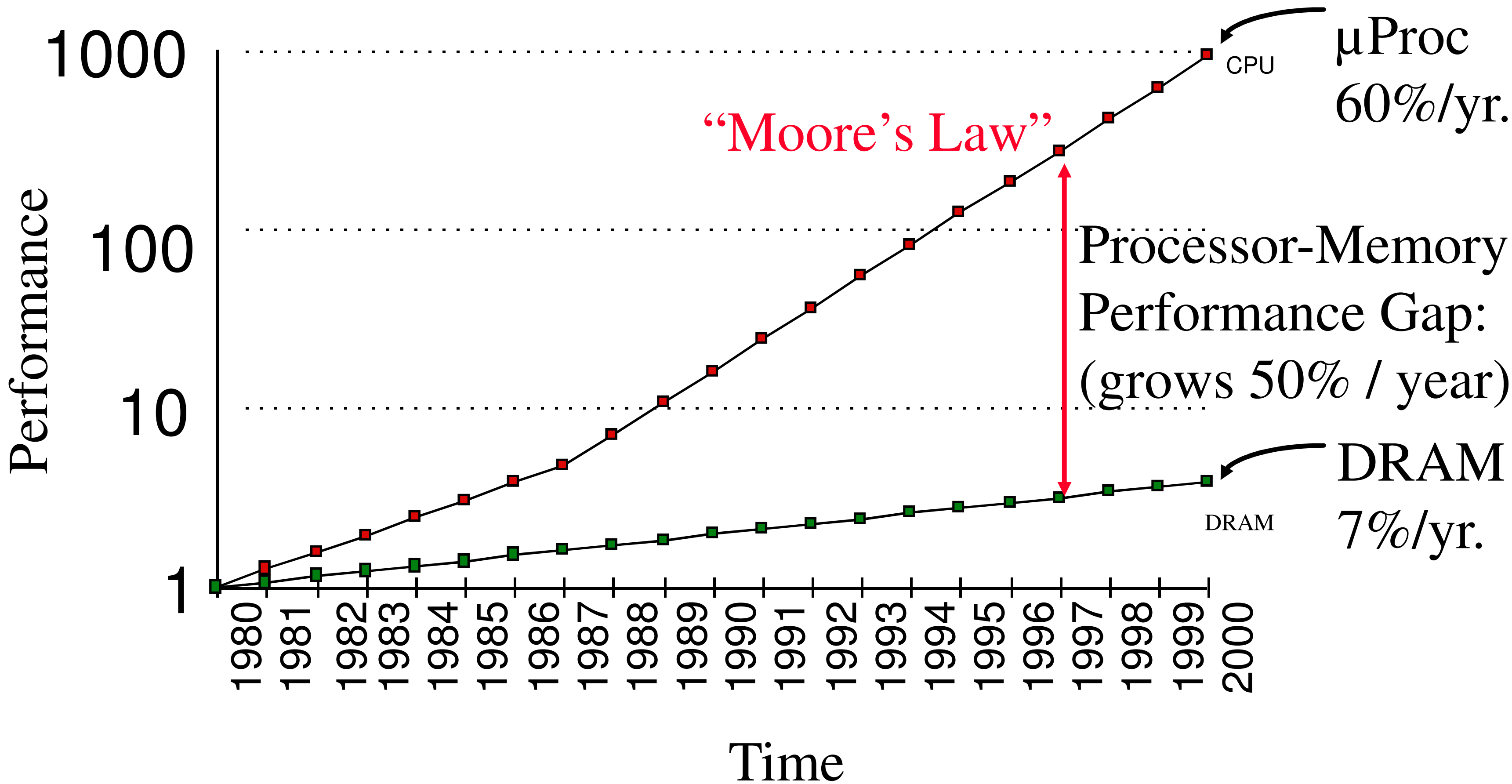
- Most programs have a high degree of locality in their accesses
  - **spatial locality**
  - **temporal locality**
- Memory hierarchy tries to exploit locality



Speed	1ns	10ns	100ns	10ms	10sec
Size	KB	MB	GB	TB	PB

Processor-DRAM Gap (latency)

- Memory hierarchies are getting deeper
  - Processors get faster more quickly than memory



Handling Memory Latency

- Bandwidth has improved more than latency
  - 23% per year vs 7% per year
- Approach to address the memory latency problem
  - Eliminate memory operations by saving values in small, fast memory (cache) and reusing them
    - Which locality is needed?
  - Take advantage of better bandwidth by getting a chunk of memory and saving it in small fast memory (cache) and using whole chunk
    - Which locality is needed?
  - Take advantage of better bandwidth by allowing processor to issue multiple reads to the memory system at once
    - concurrency in the instruction stream, e.g. load whole array, as in vector processors; or prefetching
  - Overlap computation & memory operations

Little’s Law

- Latency vs. Bandwidth
  - Latency is physics (wire length)
    - e.g., the network latency on the Earth Simulation is only about 2x times the speed of light across the machine room
  - Bandwidth is cost:
    - add more wires to increase bandwidth (over-simplification)
- Principle (Little's Law): the relationship of a production system in steady state is:

  Inventory = Throughput $\times$ Flow Time

- For parallel computing, Little’s Law is about the required concurrency to be limited by bandwidth rather than latency
  - Required concurrency = Bandwidth * Latency
- For parallel computing, this means:

  Concurrency = bandwidth x latency

Little's Law

- Example 1: a single processor:
  - If the latency is to memory is 50ns, and the bandwidth is 5 GB/s (.2ns / Bytes = 12.8 ns / 64-byte cache line)
  - What does Little's Law tell us?

- Example 2: 1000 processor system
  - 1 GHz clock, 100 ns memory latency, 100 words of memory in data paths between CPU and memory.
  - What does Little's Law tell us?

Cache Basics

- Cache is fast (expensive) memory which keeps copy of data in main memory; it is hidden from software
- Cache hit
  - In-cache memory access, cheap!
- Cache miss
  - Non-cached memory access, expensive!
- Cache line length
- Associativity
  - direct-mapped
  - *n*-way

Why Multiple Levels of Cache?

- On-chip vs. off-chip
  - On-chip caches are faster, but limited in size
- A large cache has delays
  - Hardware to check longer addresses in cache takes more time
  - Associativity, which gives a more general set of data in cache, also takes more time
- Some examples:
  - Cray T3E eliminated one cache to speed up misses
  - IBM uses a level of cache as a “victim cache” which is cheaper
- There are other levels of the memory hierarchy
  - Register, pages (TLB, virtual memory), …
  - And it isn’t always a hierarchy

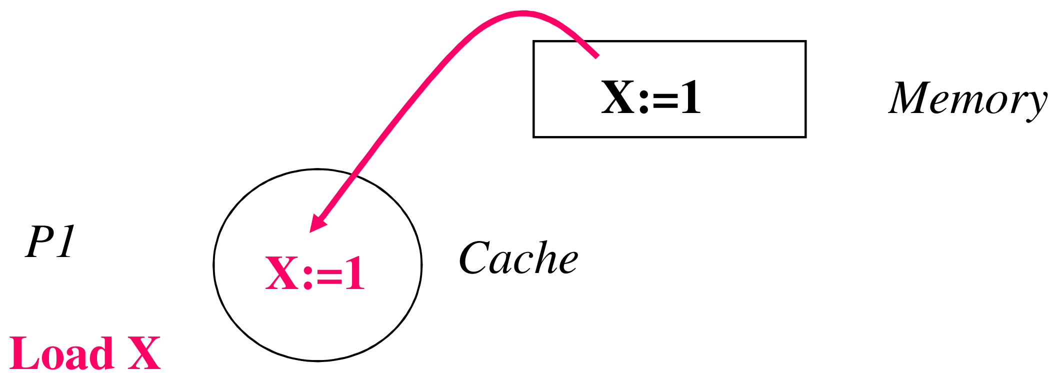
Cache Coherence

- A significant issue with shared memory
- Processors may cache the same location
- If one processor writes to the location, all others must *eventually* see the write

$\boxed{\mathbf{X:=1}}$ *Memory*

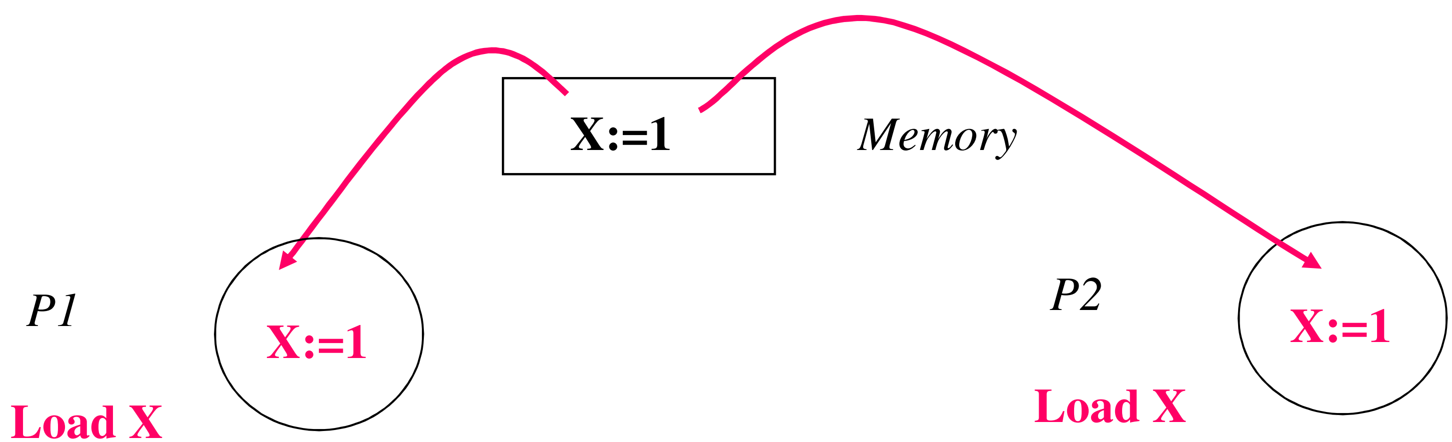
Cache Coherence

- P1 loads X from main memory into its cache



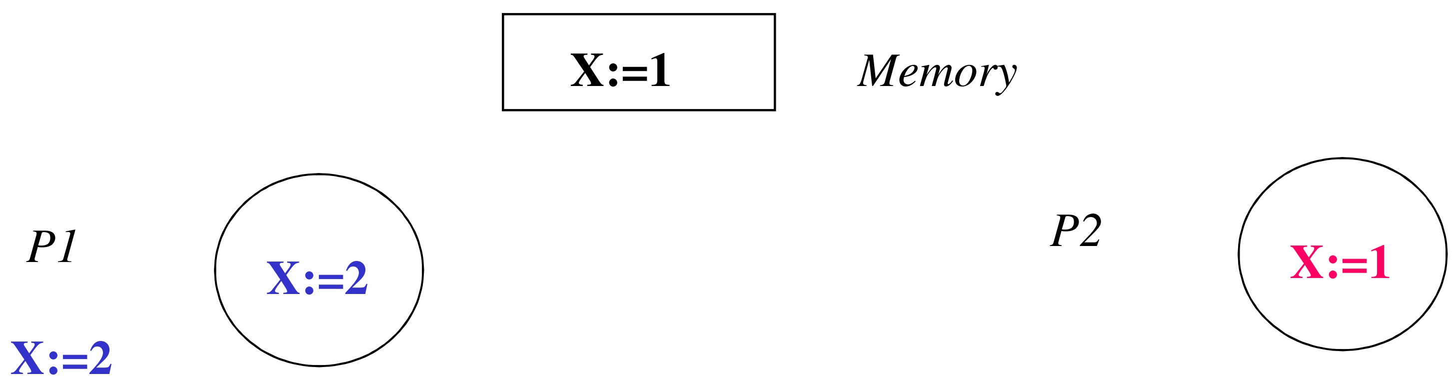
Cache Coherence

- P1 loads X from main memory into its cache
- P2 loads X from main memory into its cache



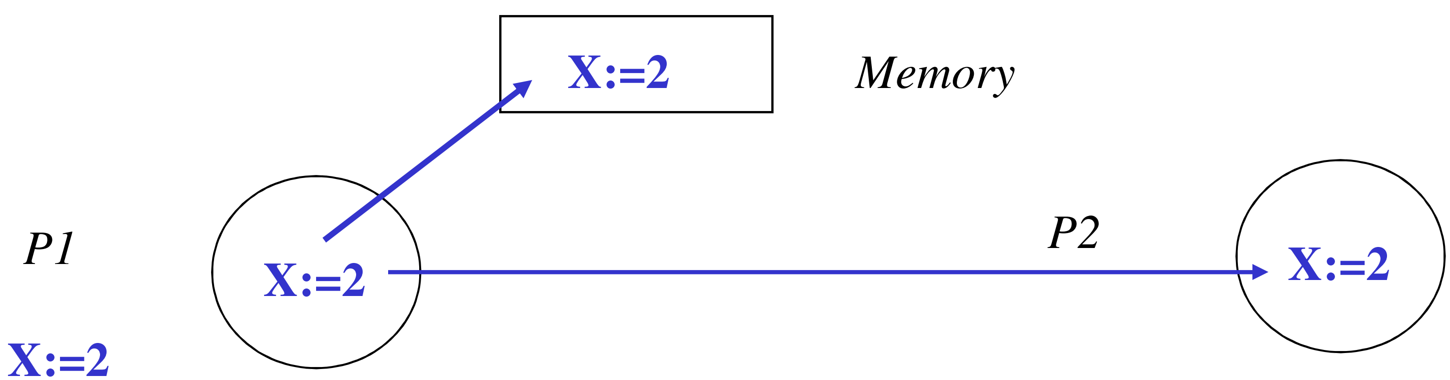
Cache Coherence

- P1 stores 2 into X
- We don’t have consistent values for X across the memory hierarchy



Cache Coherence

- Ensure that all processors *eventually* see the same value for x
- Cache coherence mechanisms:
  - Update vs. invalidate
  - Snoopy based vs. directory based

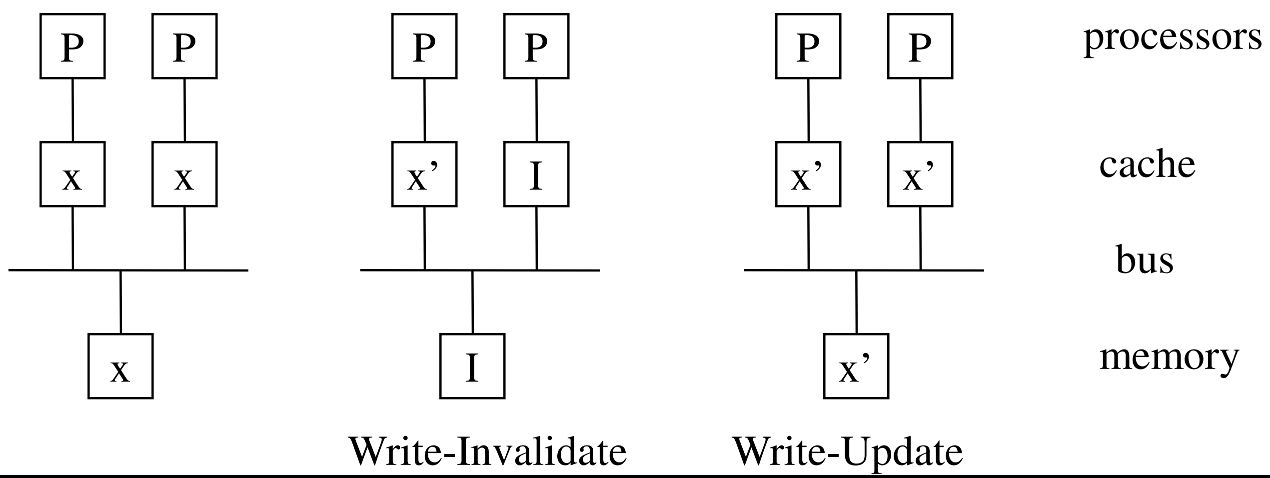


Approaches to Cache Coherence

- Hardware
  - Caches implement coherence protocols to ensure that data appears globally consistent
  - Common in hardware today
- Software
  - Relies on compiler and/or runtime support
    - May or may not have help from the hardware
  - Must be conservative to be safe
    - Assume the worst about potential memory aliases
  - Of increasing interest
    - Concerns about cost of coherence in joules
    - Scales well for microprocessor based on “tiled” designs

Cache Coherence Protocols

- When changing variable's value: invalidate or update all copies



Update and Invalidate Protocols

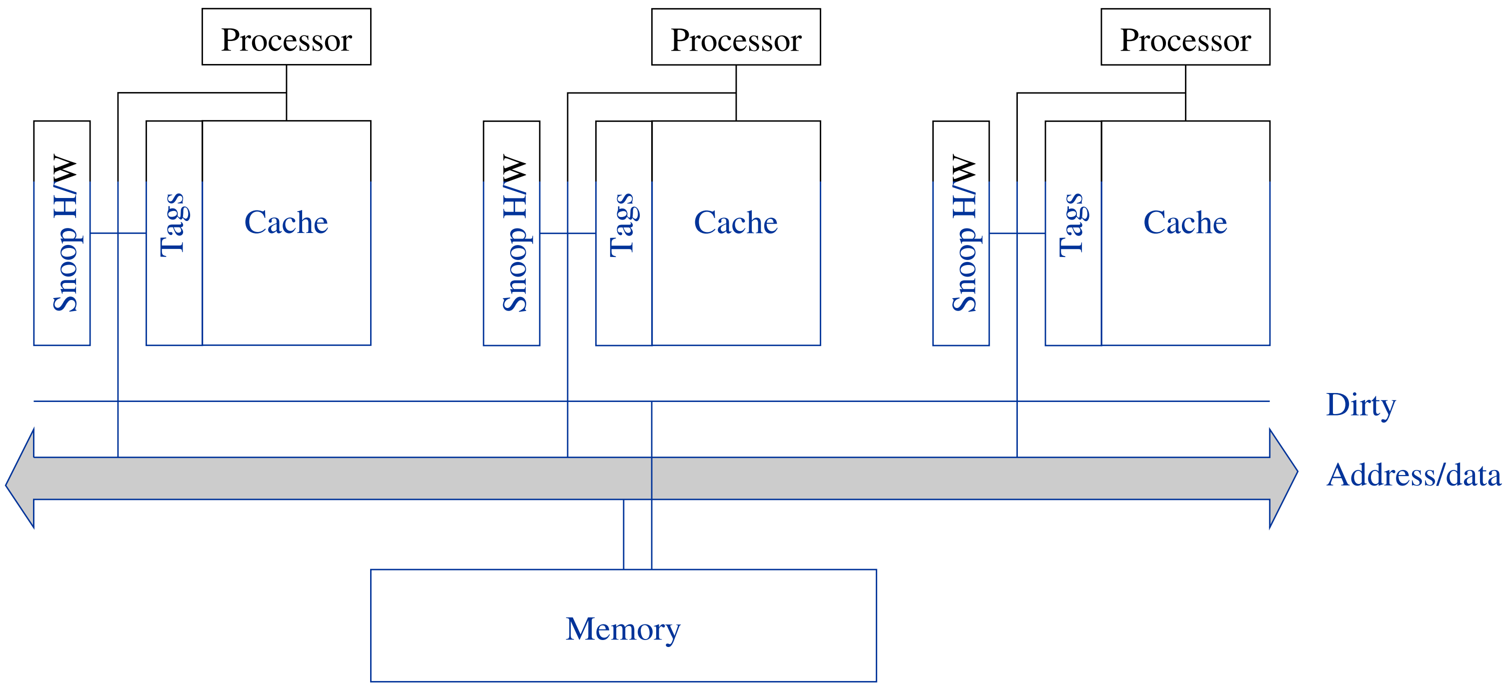
- Cost-benefit tradeoff depends upon traffic pattern
  - Invalidate is worse when
    - ??
  - Update is worse when
    - ?
- Both protocols suffer from false sharing overheads
  - What is false sharing?
- Modern machines use invalidate protocols as the default

Using Invalidate Protocols

- Each copy of a data item is associated with a state
- Example set of states: shared, invalid, or dirty
  - Shared: multiple valid copies of the data item
    - A write needs to generate an invalidate
  - Dirty: only one copy exists
    - A write need not generate any invalidates
  - Invalidate: data copy is invalid
    - A read generates a data request and updates the state

Snoopy Cache Coherence

- How are invalidates sent to the right processors?
  - Broadcast all invalidates and read requests
  - Snoopy cache listens and performs appropriate coherence operations locally



A simple snoopy bus based cache coherence system

Operation of Snoopy Caches

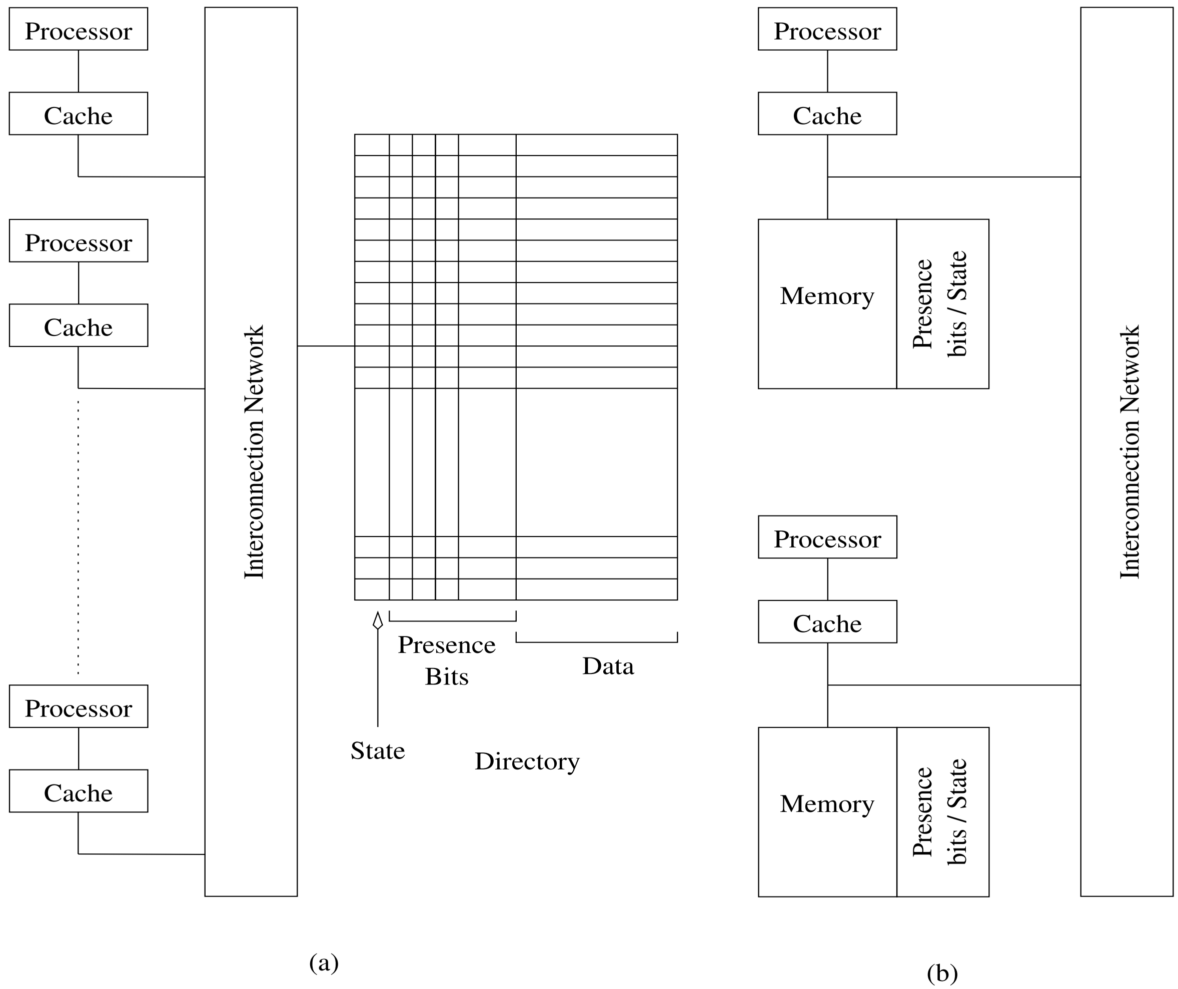
- Once a datum is tagged dirty
  - All subsequent operations can be performed locally in cache
  - No external traffic needed
- If a data item is read by a number of processors
  - Transitions to the shared state in all caches
  - All subsequent read operations become local
- If multiple processors read and update data
  - Generate coherence requests on the bus
  - Bus is BW limited: imposes a limit on updates per second

The Cost of Coherence

- Snoopy caches
  - Each coherence op is sent to all processors
  - What is the problem?
- Why not send coherence requests to only those processors that need to be notified?

Directory-based Cache Coherence

- Directory-based: the sharing status is kept in directory



Directory Implementations

- Bit vector
  - Presence bit for each cache line along with its global state
- Pointer set
  - Limited set of pointers (node IDs)
    - Less overhead than full map
  - Issue?

Performance of Directory-based Schemes

- Bits to store the directory may add significant overhead
  - Think about scaling to many processors
    - Data bits per cache block vs. presence bits per cache block
- Underlying network must carry all coherence requests
- Directory becomes a point of contention
  - Distributed directory schemes are necessary

Shared Memory Programming

- In shared memory programs
  - Start a single process and fork threads
  - Threads carry out tasks
- Dynamic threads
  - Master thread waits for work, forks new threads, and when threads are done, they terminate
  - Pro vs con?
- Static threads
  - Pool of threads created and are allocated work, but do not terminate until clean up
  - Pro vs con?

Summary

- Overview of shared memory architectures
- Interconnects
- Memory hierarchy
- Cache coherence
  - Update vs. invalidate
  - Snoopy vs. directory

- Reading:
  - Chpt2 of Kumar's book