Interconnect Routing

- *store-and-forward* routing
  - switch buffers entire message before passing it on
  - latency = [(message length / bandwidth) + fixed overhead] * # hops
- *wormhole* routing
  - pipeline message through interconnect
  - switch passes message on before completely arrives
  - latency = (message length / bandwidth) + (fixed overhead * # hops)

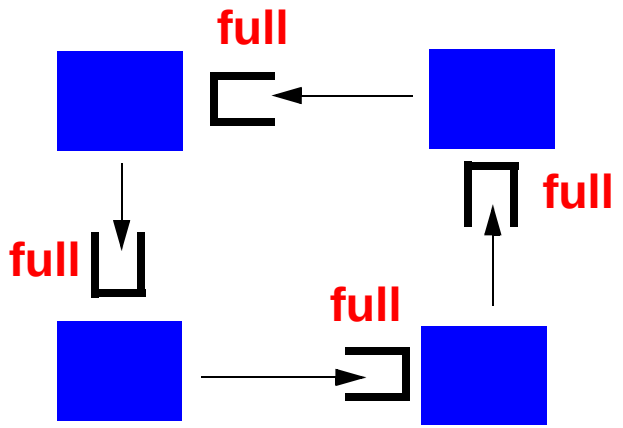
  + no buffering needed at switch

  + latency (relatively) independent of number of intermediate hops

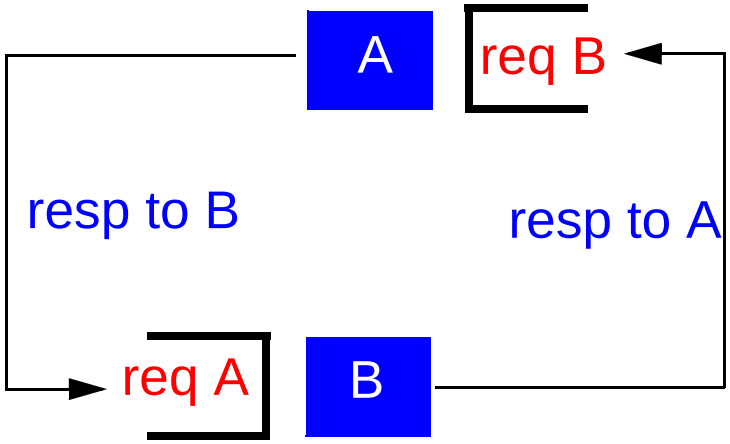
Avoiding Deadlock in Interconnect

two types of deadlock

- routing deadlock (~gridlock)
  - circular dependence on buffers
- solutions (more in ECE 259!)
  - routing restrictions (“turn model”)
  - virtual channels



- request/response deadlock
  - circular dependence on messages
- solutions
  - separate networks
  - virtual networks



Shared Memory vs. Message Passing

MIMD dimension II: appearance of address space to software

- *message passing* (multicomputers, clusters)
  - each processor has its own address space (and unique processor #)
  - processors send (receive) messages to (from) each other
  - communication pattern explicit and precise (only way)
  - used for scientific codes (explicit communication patterns)
  - message passing systems: PVM, MPI

  \+ simple hardware

  – difficult programming model (in general)

Shared Memory vs. Message Passing

- *shared memory* (multiprocessors, multicores)
  - one shared address space
  - processors use conventional loads/stores to access shared data
  - communication can be complex/dynamic

  + simpler programming model (compatible with uniprocessors)

  – but with its own nasties (e.g., synchronization)

  – more complex hardware... (we'll see soon)

  + but more room for hardware optimization
- aside: software shared virtual memory (SVM) exists

Multiprocessor Industry Trends

- shared memory
  - easier, more dynamic program model (it IS the software, stupid!)
  - can do more to optimize the hardware
- multicore chips
  - achieve greater scalability by using multiple chips
  - *glueless MP*: slap these together and it just works! e.g., Opteron
- larger NUMA systems built from smaller (N)UMA systems
  - exploit commodity nature of small systems
  - use commodity interconnect (e.g., gigabit Ethernet, Myrinet)
  - called NUMA clusters

Caching Shared Memory

- three issues
  - cache coherence
  - synchronization
  - memory consistency model
- not completely unrelated to each other
- not issues for message passing machines
  - why not?

Cache (In)Coherence

- most common cause: sharing of writeable data
  - example

processor 0	processor 1	correct value of A is in..
		memory
read A		memory, p0 cache
	read A	memory, p0 cache, p1 cache
write A		p0 cache, memory (if wthru)
	read A	p1 gets stale value on hit

- other causes
  - process migration (even if jobs are independent)
  - I/O (can be fixed by OS cache flushes)

Solutions to Coherence Problem

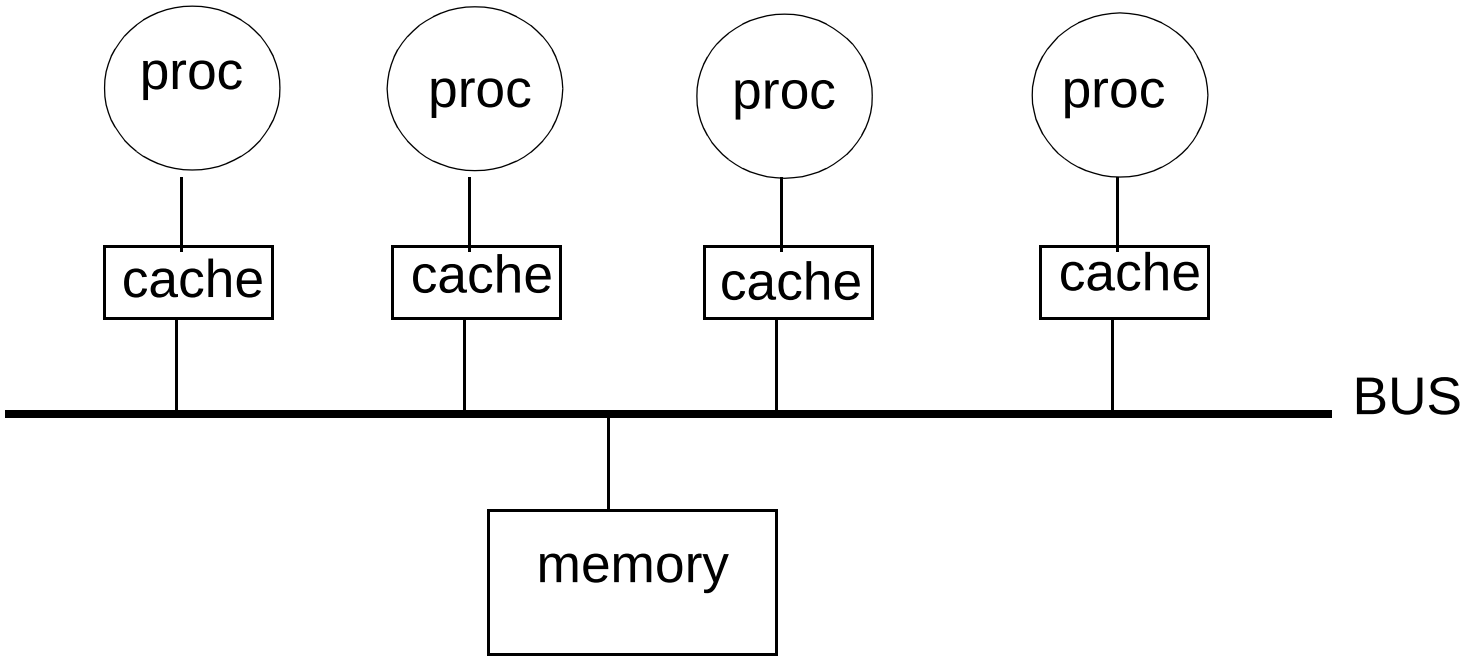
- no caches
  - not a good solution - caches are important!
- make shared data non-cacheable
  + simplest software solution
  – low performance if a lot of data is shared
- software flush at strategic times: e.g., after critical sections
  + relatively simple
  – low performance if synchronization is frequent
- hardware *cache coherence*
  - make memory and caches coherent (consistent) with each other
  - in other words: let memory and other processors see writes
  - invisible to software

Cache Coherence Protocols

- absolute coherence
  - all copies of each block have same data at all times
  - not necessary
- what is required is *appearance of absolute coherence*
  - temporary incoherence is OK (e.g., write-back cache)
  - as long as all loads get “correct” values
- *cache coherence protocol*: FSM that runs at every cache
  - and usually a FSM at every memory, too
- two ways of handling writes
  - *invalidate protocol*: invalidate copies in other caches
  - *update protocol*: update copies in other caches

Bus-Based Protocols (Snooping)

- bus-based cache coherence protocol (snooping)
  - ALL caches/memories see and react to ALL bus events
  - protocol relies on global visibility of requests (ordered broadcast)
  - owner (either proc or mem) responds to request with data

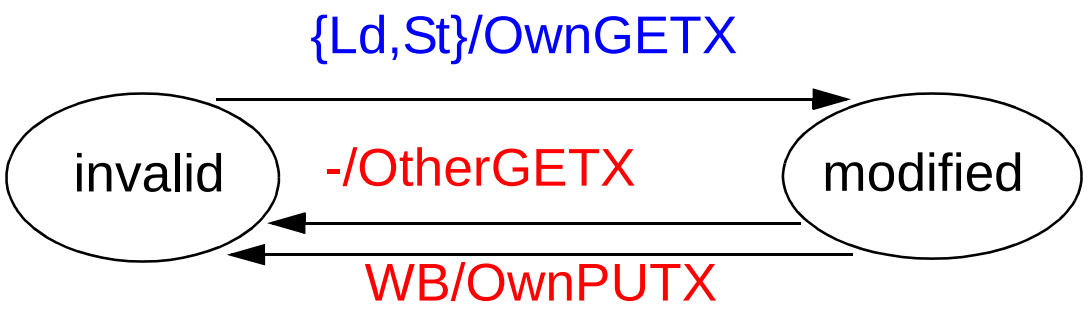


Snooping Protocol Events

- requests from proc/cache to cache coherence controller
  - load (Ld)
  - store (St)
  - writeback (WB)
- bus events (= cache coherence transactions)
  - GetShared (GETS) - broadcast request for read-only data
  - GetExclusive (GETX) - broadcast request for read-write data
  - PutExclusive (PUTX) - broadcast request to write data back to mem
- coherence transactions on bus can be from self or others
- we'll assume atomic bus transactions
  - thus, we have atomic cache coherence transactions

Two-State (MI) Invalidate Protocol

- two states
  - *invalid (I)*: either don't have block or have it but not allowed to use it
  - *modified (M)*: have block with read-write access
- problem
  - – block can be in only one cache at a time
  - – not efficient, especially if data is only being read



notation "a/b":

a = proc request

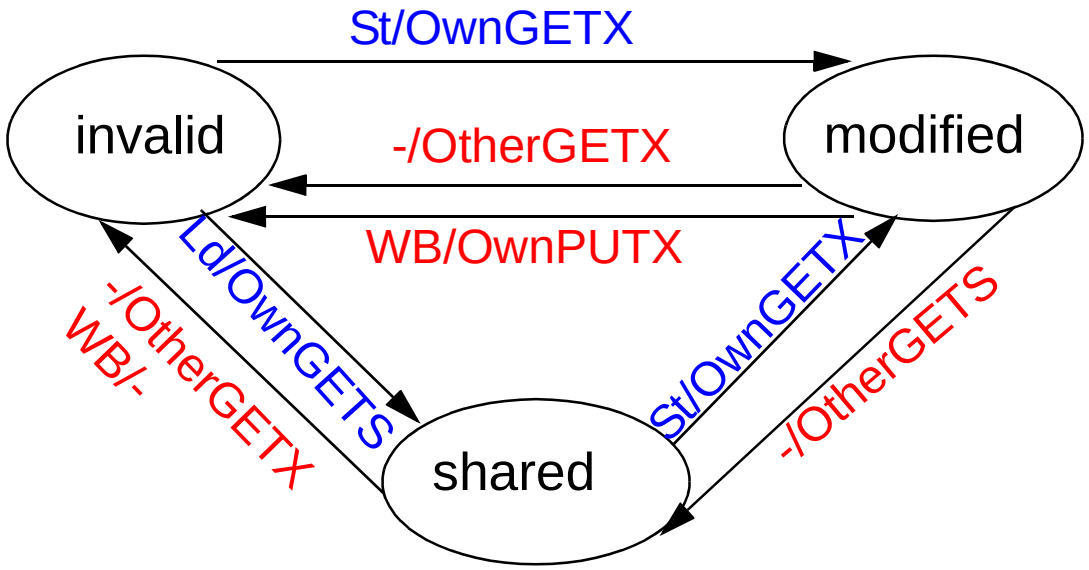
b = coherence transaction

blue = upgrade

red = downgrade

Three-State (MSI) Invalidate Protocol

- three states
  - *idea: add new "read-only" state (shared) - allows multiple readers!*
  - *invalid*
  - *modified*: have block with read-write access
  - *shared (S)*: have block with read-only access



Scalable Coherence Protocols: Directories

– bus-based protocols (i.e., broadcast) are not scalable!
  - not enough bus b/w for everyone's coherence traffic
  - not enough processor snooping b/w to handle everyone's traffic

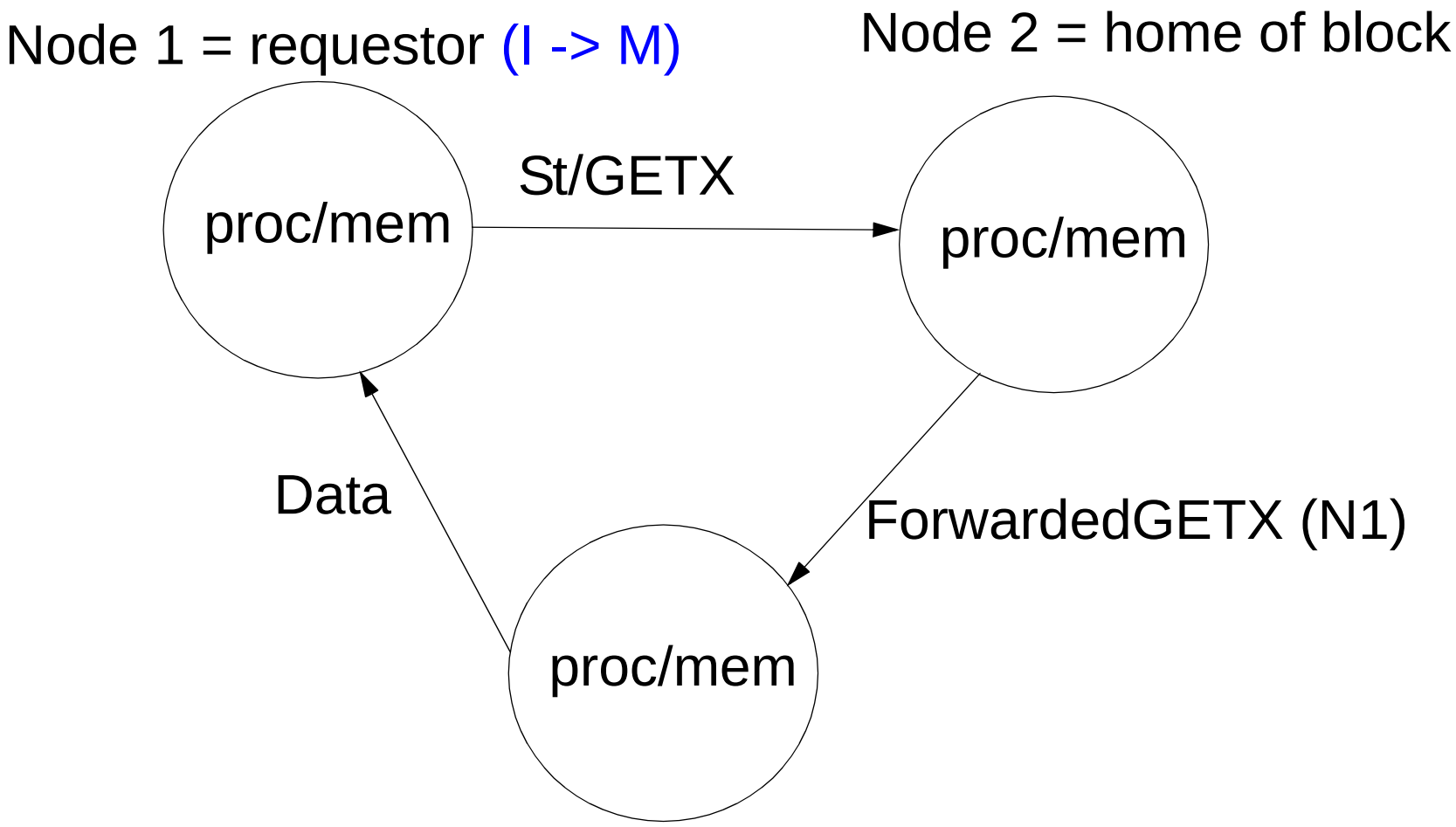
- *directories*: scalable cache coherence for large MPs
  - each memory entry (cache line) has a bit vector (1 bit per processor)
  - bit vector tracks which processors have cached copies of line
  - send all requests to directory at home memory
  - if no other cached copies, memory is owner and returns data
  - otherwise, memory forwards request to current owner processor

  + low b/w consumption (communicate only with processors that care)

  + works with general interconnect (bus not needed)

  – longer latency (3-hop transactions: p0 ⇒ directory ⇒ p1 ⇒ p0)

Directory Protocol in Action (MI)



Coherence Protocols: Performance

- 3C miss model $\Rightarrow$ 4C miss model
  - capacity, compulsory, conflict
  - *coherence*: additional misses due to coherence protocol
  - complicates uniprocessor cache analysis
- as processors are added
  - – coherence misses increase (more communication)
- as cache size is increased
  - \+ capacity misses decrease
  - – coherence misses increase (more shared data is cached)
- as block size is increased
  - – coherence misses increase (false sharing)
  - *false sharing*: sharing of different data in same cache line