- Your professor du jour:

Steve Gribble
gribble@cs.washington.edu
323B Sieg Hall

  - all material in this lecture in Henessey and Patterson, Chapter 8
  - 635-640
  - 645, 646
  - 654-665

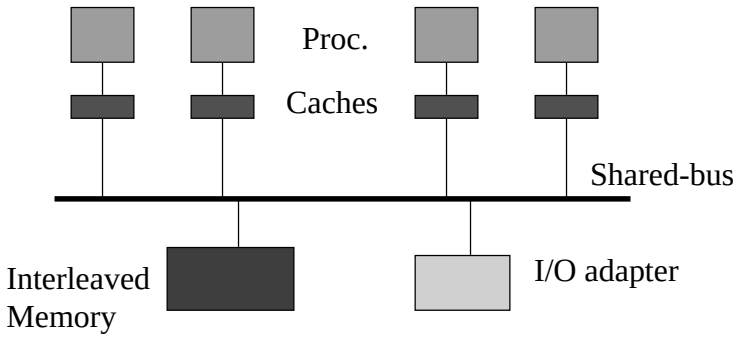
Limitations of parallel processing

- Multiprocessors are used to
  - speed up computation
  - solve larger problems
- But most workloads include sequential **and** parallel phases
  - synchronization increases sequentiality
  - Amdahl's law will limit speedup
  - e.g., with 100 processors, and 10% sequential, what is speedup?
- Communication is expensive (50-10000 clock cycles)
  - communication:computation ratio limits speedup
  - "embarrassingly parallel" applications

What is maximum speedup for N processors?

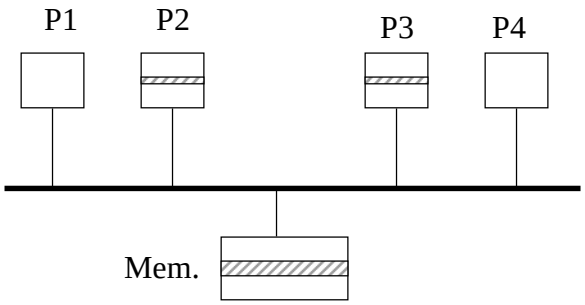
- If a program takes X seconds on a uniprocessor, what is the fastest it can run on an N-way multiprocessor?
  - N?

SMP (Symmetric MultiProcessors) single shared-bus systems



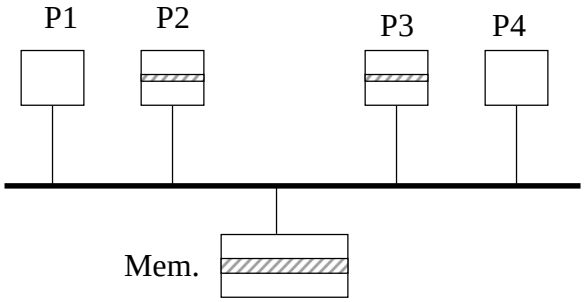
Replication of shared data

P2 reads A; P3 reads A



The coherence problem…

Now what happens if P4 writes A??



Cache coherence

- Determines what are valid values returned by a read
- If P1 writes to X, followed by P1 reads from X
  - then **P1 must read its own write**
- If P1 writes to X, then P2 reads from X a short while later
  - must have **P2 reads P1's write**
- If P1 writes value "A" to X, and P2 writes value "B" to X at about the same time
  - require that **if any processor reads "A" then "B", all processors must read "A" then "B".**

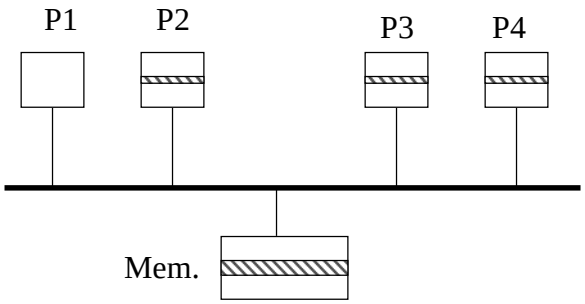
"Snooping" vs. "directory based"

- Directory based coherence: (distributed memory machines)
  - a special location called a "directory" maintains sharing status of cache blocks
  - cache controllers must explicitly communicate with directory
- Snooping cache coherence: (mostly SMPs)
  - all caches maintain the sharing status of cache blocks
  - no centralized state
  - all cache controllers listen on the shared-memory bus to update status of cache blocks
  - automatic serialization of writes (needed for coherence)

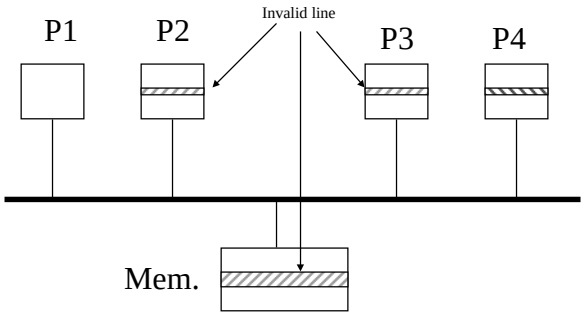
“Snooping” cache coherence techniques

- Two main flavors:
- Write-invalidate protocols
  - P4 sends an invalidate message out on memory bus
  - P2 and P3 invalidate the copy in their caches
- Write-update or write-broadcast protocols
  - P4’s write is write-through, and sent on memory bus
  - the new value of A is snooped by caches of P2 and P3

Write-update



Write-invalidate



False sharing and write-invalidate

- Occurs when two processors read/write to different words in the same cache block
  - from point of view of cache coherence protocol, looks like the entire block is shared data
  - block bounces back and forth between the processors
  - worst case: P1 is constantly reading word X, and P2 is constantly writing word Y, and both X and Y are in same cache block

Effects on false sharing…

- larger cache block size?
- larger cache?
- larger miss penalty?
- more processors?

- ways to reduce false sharing:
  - compiler optimization (layout of data in memory, block padding)
  - cache-conscious programming
    - can massively inflate/deflate constants in big-O notation

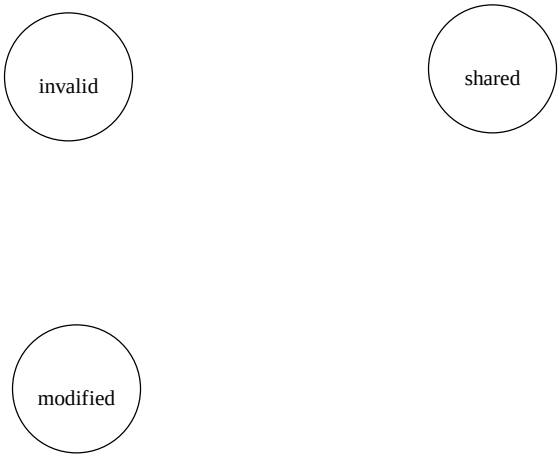
State machine implementation of snooping protocols

- Associate states with each cache block (minimal 3)
  - **I**nvalid (can use existing valid bit in cache block)
  - **S**hared (possibly many copies, all up to date - need another bit)
  - **M**odified (dirty data; exists in only one cache - use dirty bit)
- Fourth state (and sometimes more) for performance purposes
  - **E**xclusive (clean, only copy)

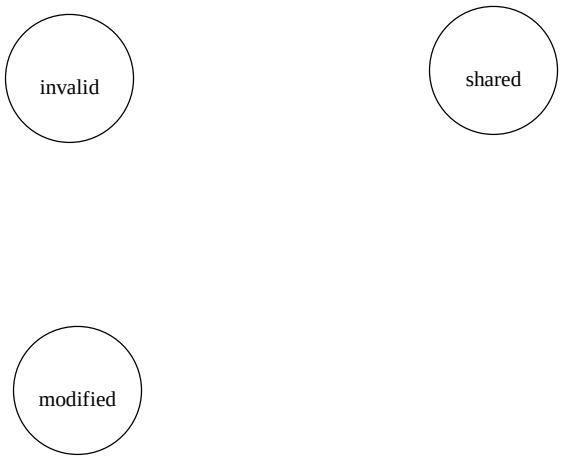
State transitions for a given cache block

- Events incurred by processor associated with the cache
  - Read miss, write miss, write on clean block
- Events incurred by snooping on the bus as result of other processor actions, e.g.,
  - Read miss by Q might make P's block transit from dirty to clean
  - Write miss by Q might make P's block transit from dirty/clean to invalid (write invalidate protocol)

Simple 3-state protocol (CPU activity)



Simple 3-state protocol (snooped bus activity)



Illinois protocol: more advanced

- Add 4th exclusive state to enhance performance
  - On a write to a block in E state, no need to send an invalidation message (occurs often for private variables).
- On a read miss with no cache having the block in dirty state
  - Who sends the data: memory or cache (if any)?
    - Answer: cache for that particular protocol; other protocols might use the memory
  - If more than one cache, which one?
    - Answer: the first to grab the bus (tri-state devices)

Performance of snoopy protocols

- Performance depends on the length of a *write run*
- Write run: sequence of write references by 1 processor to a shared address (or shared block) uninterrupted by either access by another processor or replacement
  - long write runs better to have write invalidate
  - short write runs better to have write update
- There have been proposals to make the choice between protocols at run time

Wrinkle #1: snooping interfering with CPU

- each bus transaction causes a check on cache tags
  - could interfere with CPU's cache access, stalling processor
- fix #1: extra set of tags for snooping activity
  - on cache miss, processor must arbitrate for and update both sets
  - if snoop finds match, it must also update both tags for invalidates or to update shared bit
- fix #2:
  - exploit multilevel cache, and cache inclusion property
    - every entry in L1 cache also in L2 cache
    - snoop uses L2 cache, CPU uses L1 cache
    - snoop hit: snoop may need to update L1 cache as well
  - can do fix #1 and fix #2 combined…

Wrinkle #2: atomicity of operations

- protocols assume that operations are atomic
  - e.g., assumes write miss can be detected, acquire bus, and receive response as a single atomic action
- introduces possibility that protocol can deadlock
  - to fix, need to augment protocol to deal with non-atomic writes without adding deadlock
  - a topic for a different course…