CS465: Computer Systems Architecture

Lecture 10: Multiprocessor and Parallel Computing

*Slides adapted from Computer Organization and Design by Patterson and Henessey

Outline

- Introduction to multiprocessors
  - Models, classifications, issues
- Cache coherence problem
- Introduction and challenges of parallel processing (if time permits)

Multiprocessor Systems

- Multiprocessors used to build servers and supercomputers
    - Goal: connecting multiple processors to get higher performance
- New trend in microprocessor industry: all companies switch to multicore
    - Multiple simple processors w/ lower frequency
- New trend: warehouse-scale computing
    - Delivery of computing services over the Internet
    - Collections of nodes that are connected by LAN to provide the service

Multiprocessor Architecture Study

- Multiprocessor architecture is a large and diverse field
  - Culler, Singh, Gupta: Parallel Computer Architecture: A Hardware/Software Approach, MKP
  - An active field of research
- Our focus:
  - Basic models and classifications
    - Ch6.1, 6.3-6.5, 6.7
  - Cache coherence
    - Ch5.10
  - Introduction to parallel processing/programming
    - Ch6.2

Flynn's Taxonomy

- Computer models classified by data and instruction streams [Flynn' 1966]

Single Instruction Single Data (SISD) (Uniprocessor)	Single Instruction Multiple Data (SIMD) (Vector, MMX)
Multiple Instruction Single Data (MISD) (????)	Multiple Instruction Multiple Data (MIMD) (Clusters, SMP servers)

- SIMD ⇒ Data Level Parallelism
- MIMD ⇒ Thread/process Level Parallelism
- SPMD: Single Program Multiple Data
  - A parallel program on a MIMD computer
  - Conditional code for different processors

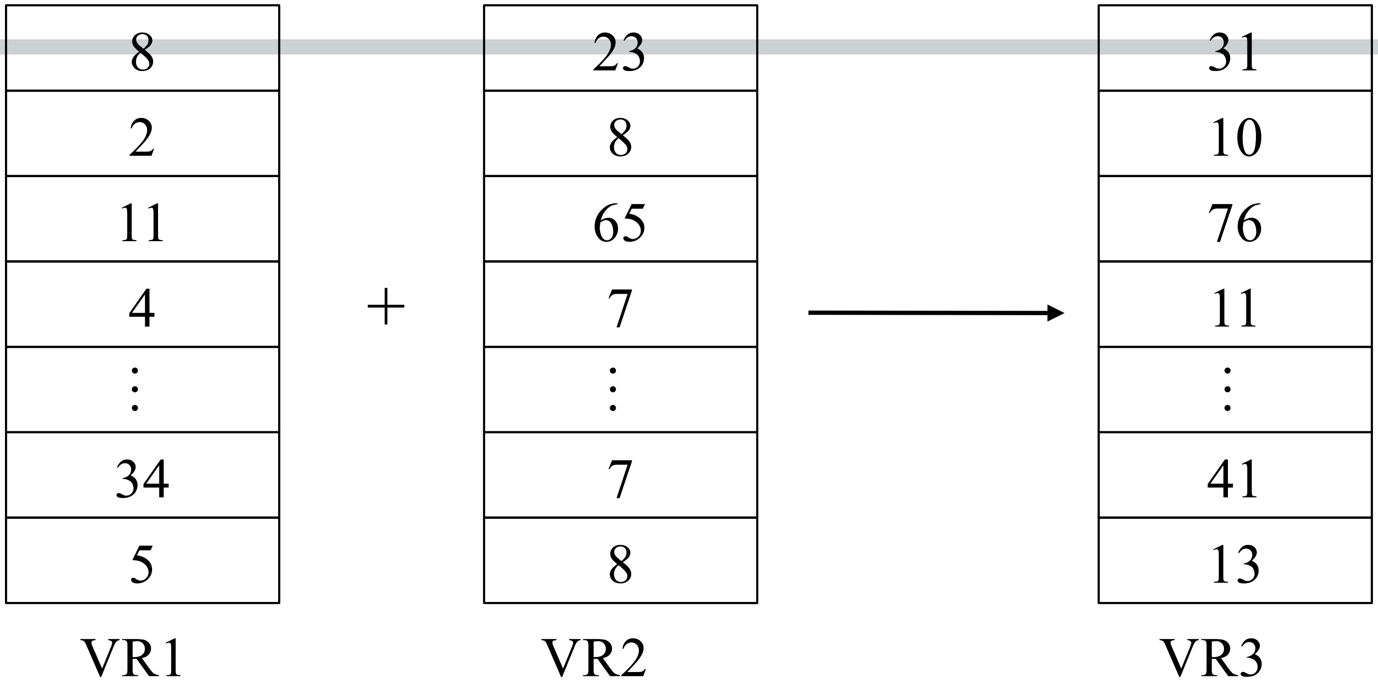
SIMD

- All units execute the same instruction at the same time
  - Each with a different data address
  - Exploit significant data-level parallelism
- Benefits:
  - SIMD allows programmer to continue to think sequentially
  - Simplified synchronization / instruction control hardware, more energy efficient
- Examples: Vector, GPU, SIMD extensions

Vector Architectures

- Read sets of data elements into “vector registers”
  - Each vector register holds multiple elements
- Operate on those registers with pipelined processing units and/or replicated units
  - Vector functional units / load-store unit
  - Data and control hazards are detected
- Disperse the results back into memory

Vectorized Program Example



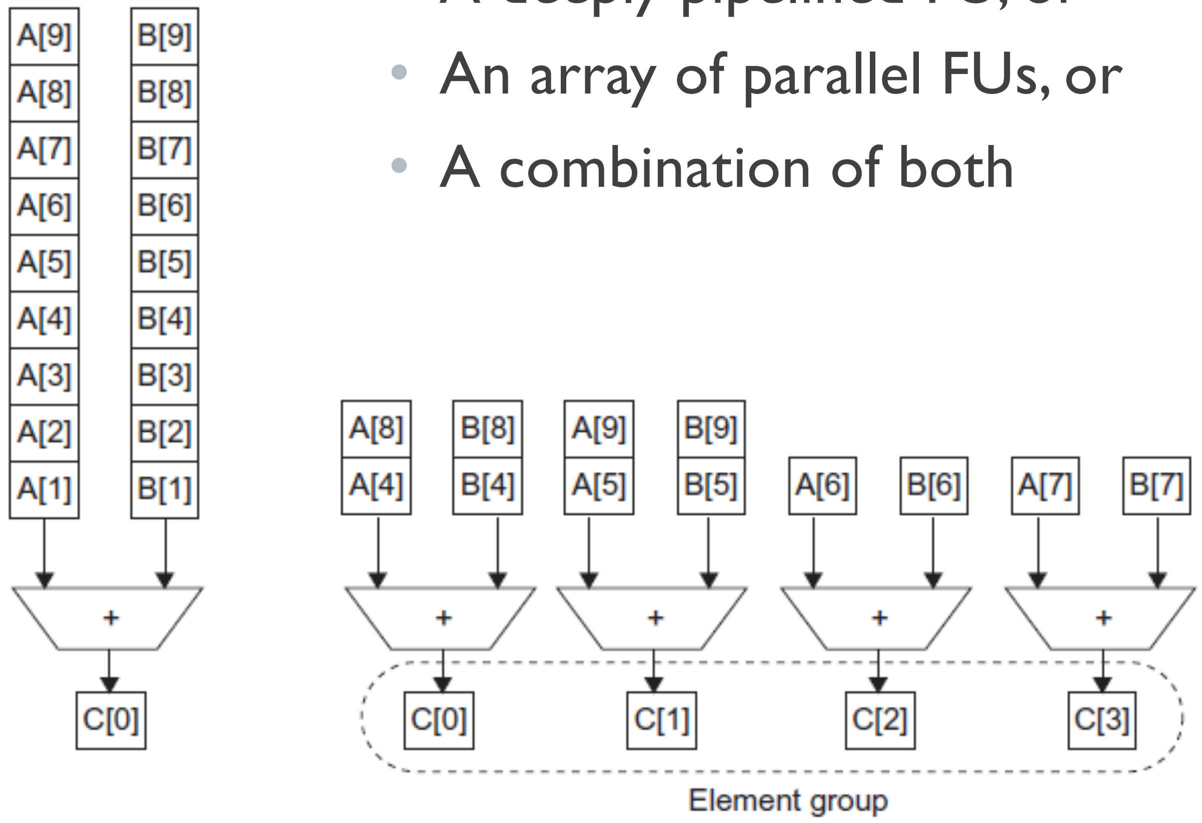
```
      DO I = 1, N
S       C(I) = A(I) + B(I)
      ENDDO
```

- Vector instructions

```
VLOAD VR1,A
VLOAD VR2,B
VADD VR3,VR1,VR2
VSTORE C,VR3
```

Vector Implementation

- A vector addition can be implemented using
  - A deeply pipelined FU, or
  - An array of parallel FUs, or
  - A combination of both



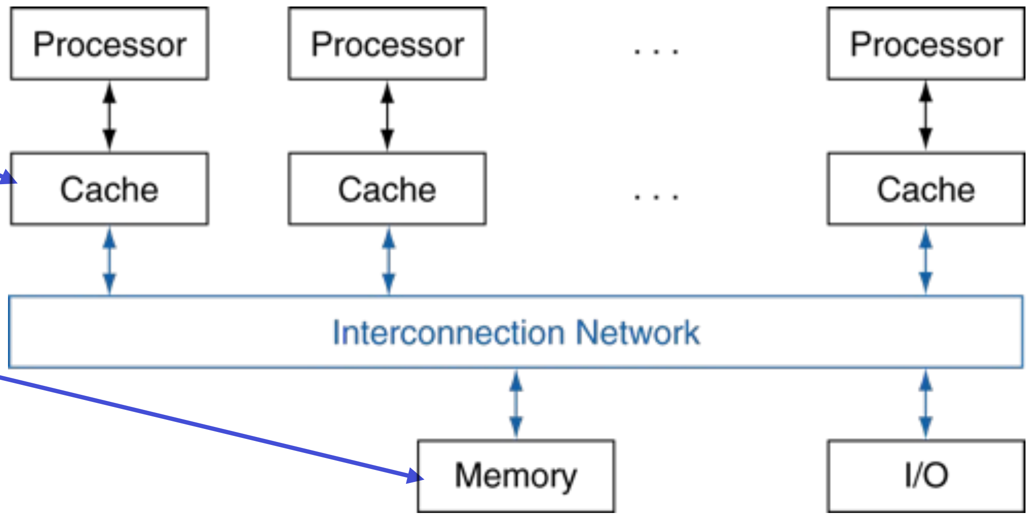
MIMD: Multiprocessors

- Multiple processors running multiple (different) instruction sequences
  - Instructions may come from multiple processes or threads
- Two classes of multiprocessors WRT memory:
  - Shared-memory multiprocessor
    - Share a single, centralized memory
    - Small processor counts
  - Physically distributed-memory multiprocessor
    - Each processor has its own private address
    - Larger number chips and cores

Shared Memory Multiprocessor

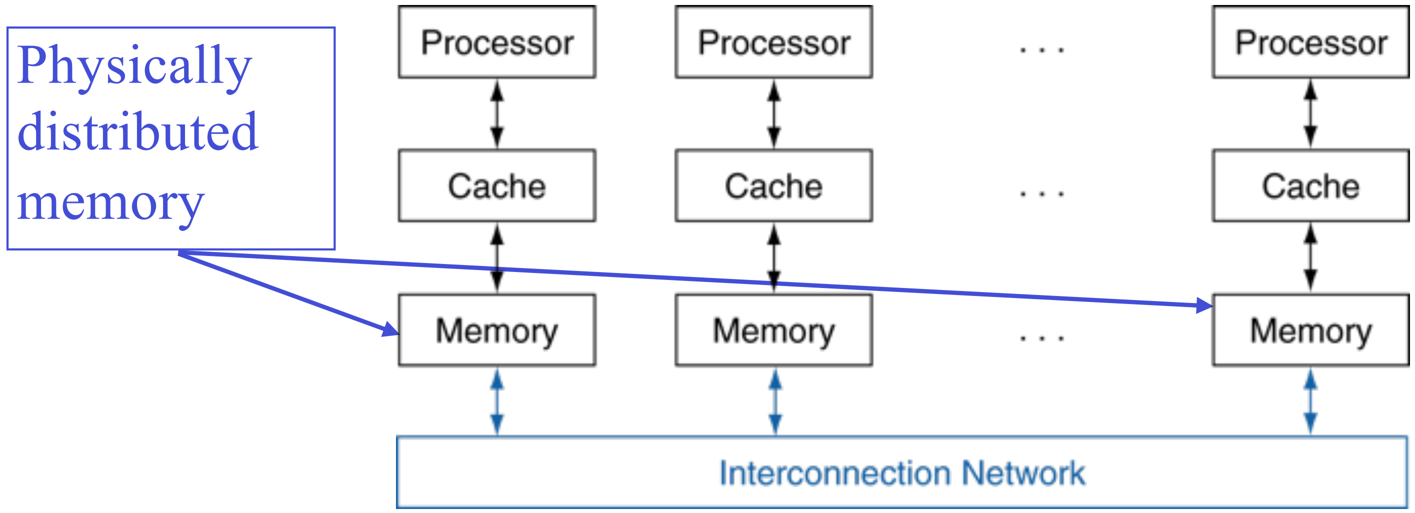
Local cache for fast data accesses

One memory shared by all processors



- SMP: A single shared memory architecture
  - Hardware provides single physical memory for all processors
  - Communicating w/ shared variables and locks
  - Symmetric relationship to all processors
  - Typically large caches: local copy of data
  - Limiting the number of processors

Distributed Memory Multiprocessor



- Each processor has private physical address space
  - Hardware sends/receives messages between processors
- Pros
  - Distribution is cost-effective in scaling memory bandwidth if most accesses are to local memory
  - Access latency is reduced
- Cons
  - Communicating data becomes complex
  - More software effort to take advantage of the increased BW

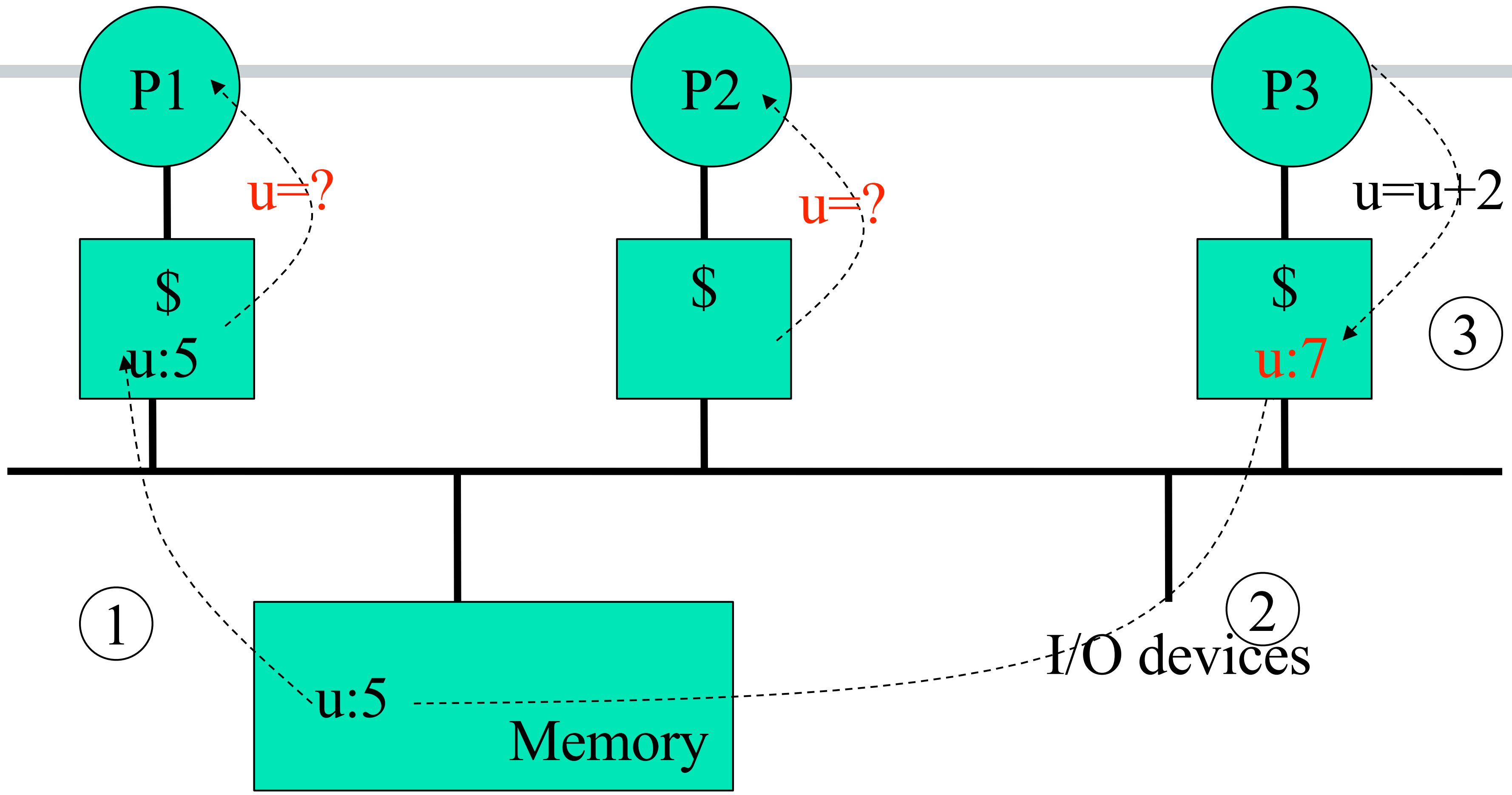
Outline

- Introduction to multiprocessors
    - SIMD, MIMD
    - SMP, message-passing multiprocessors
- Cache coherence problem
- Introduction and challenges of parallel processing (if time permits)

Shared-Memory Architectures

- Multiple processors with a centralized shared-memory
  - From multiple boards on a shared bus to multiple processors inside a single chip
- Cache
  - Private data are used by a single processor
  - Shared data are used by multiple processors
- Caching shared data
  - Reduces latency to shared data, memory bandwidth for shared data, and contention
  - Shared value may be replicated in multiple caches: cache coherence problem

Cache Coherence Problem Example



© Parallel Computer Architecture, Culler, Singh & Gupta, MKP

- Assume write back caches, similar problem for write through caches

Cache Coherence Problem

- Processors see different values for the same memory location
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value
  - Processors accessing main memory may see very stale value
- Even with write through caches, different views of a memory location may be held by two different processors
- Unacceptable for programming!

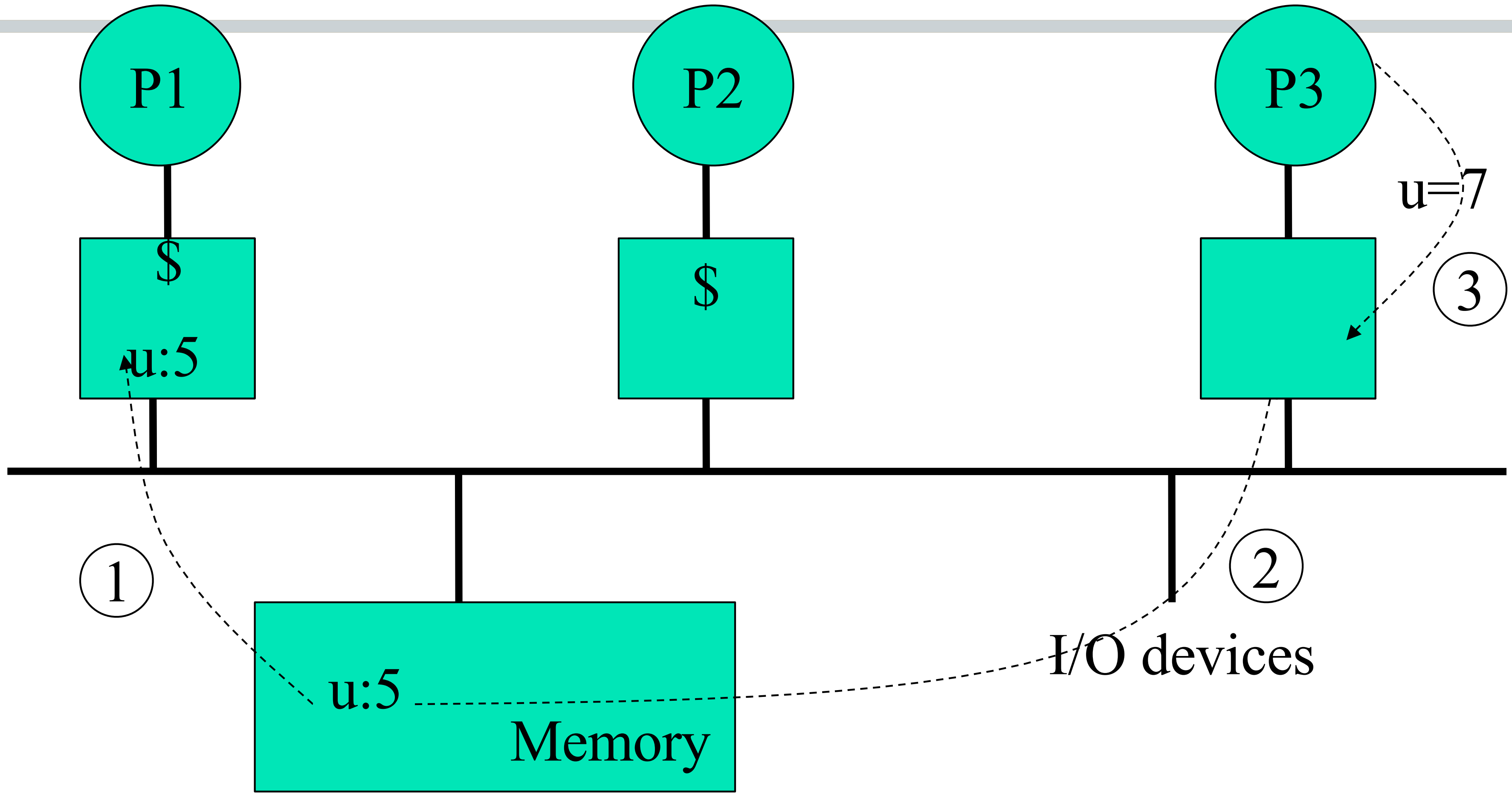
Cache Coherent Protocols

- Rather than trying to avoid sharing in SW, SMPs use a HW protocol to maintain coherent caches
- Key: eliminating incoherent copies
  - Option 1: update all copies on write
    - Write update protocol
    - Expensive communication (longer messages for updating data, more bus traffic)
  - Option 2: get exclusive access before write
    - All other cached copies are invalidated
    - Write invalidate protocol ←
    - Longer access latency

Cache Coherent Protocols

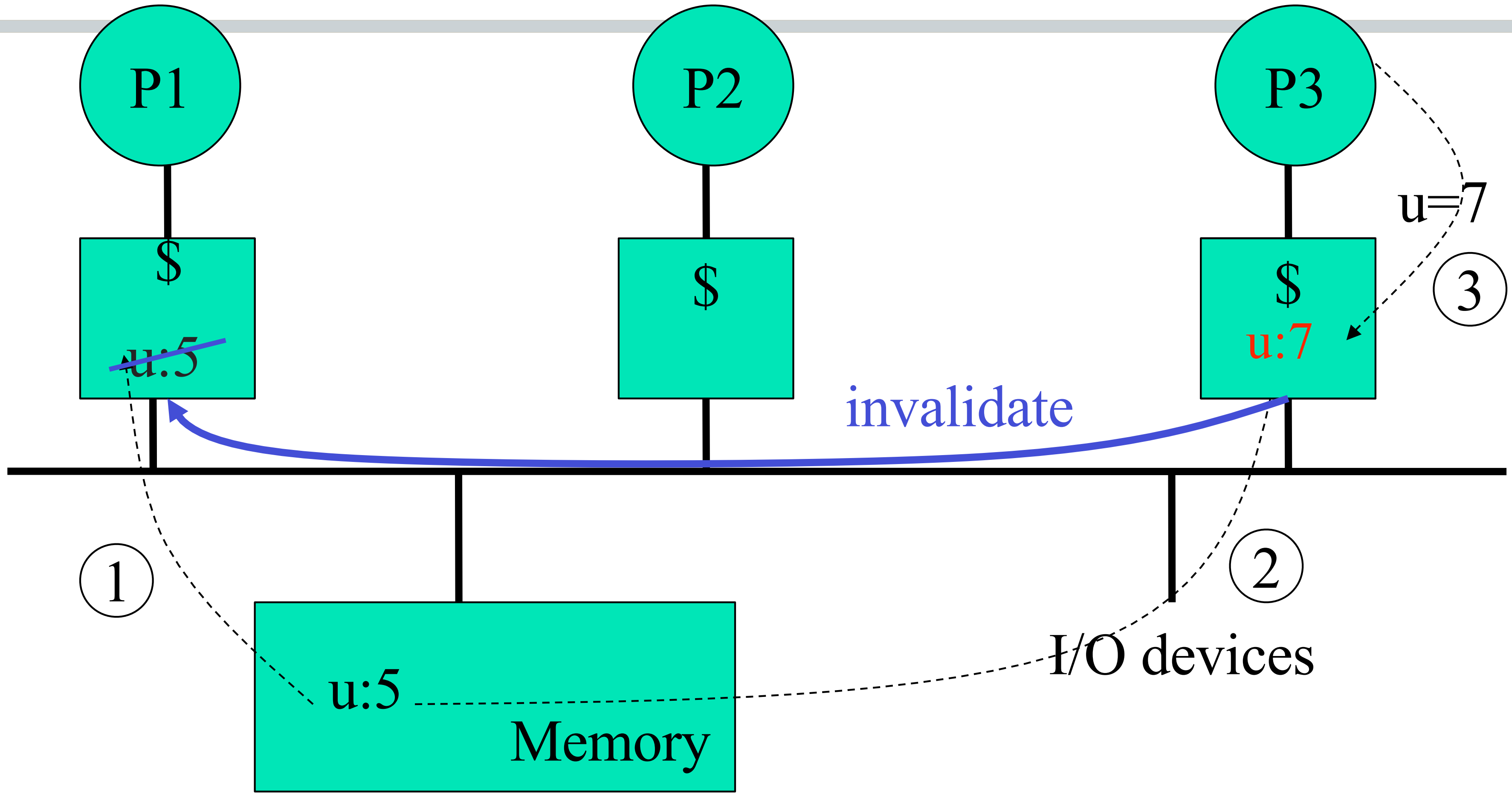
- Key: bookkeep sharing status of shared blocks:
  - Shared, exclusive, invalid, etc.
- Snooping protocols ←
  - Every cache keeps sharing status of its own blocks
  - All caches are accessible via some broadcast medium (e.g. a bus)
  - Sharing status are maintained by a cache controller
    - Status change in response to broadcasted transactions and local processor requests
- Directory based protocols
  - Sharing status of a block of kept in just one location, the directory
  - Scales better than snoop

Example Revisited: Invalidate/Snoopy



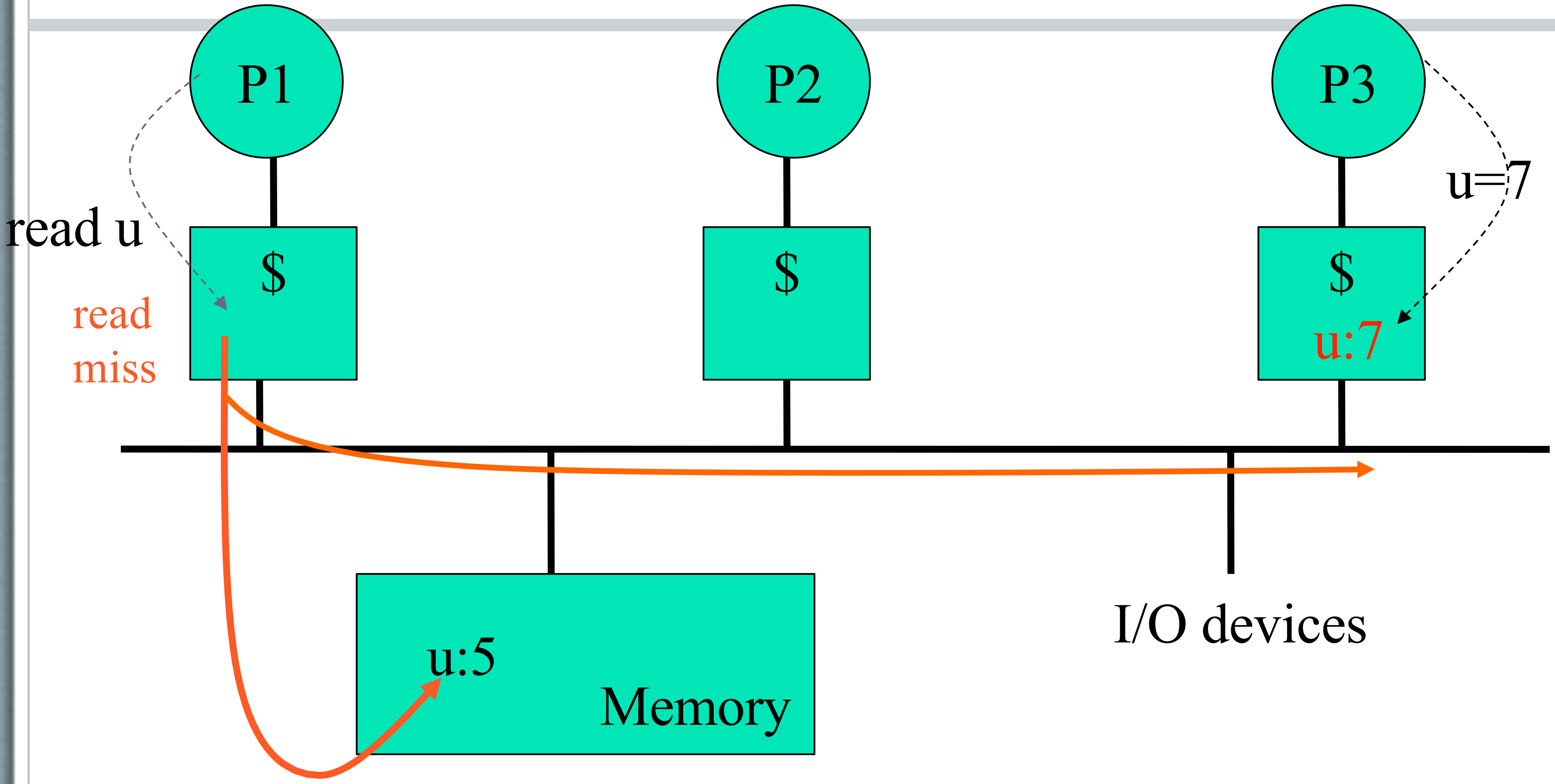
- Write back cache

Example Revisited: Invalidate/Snoopy



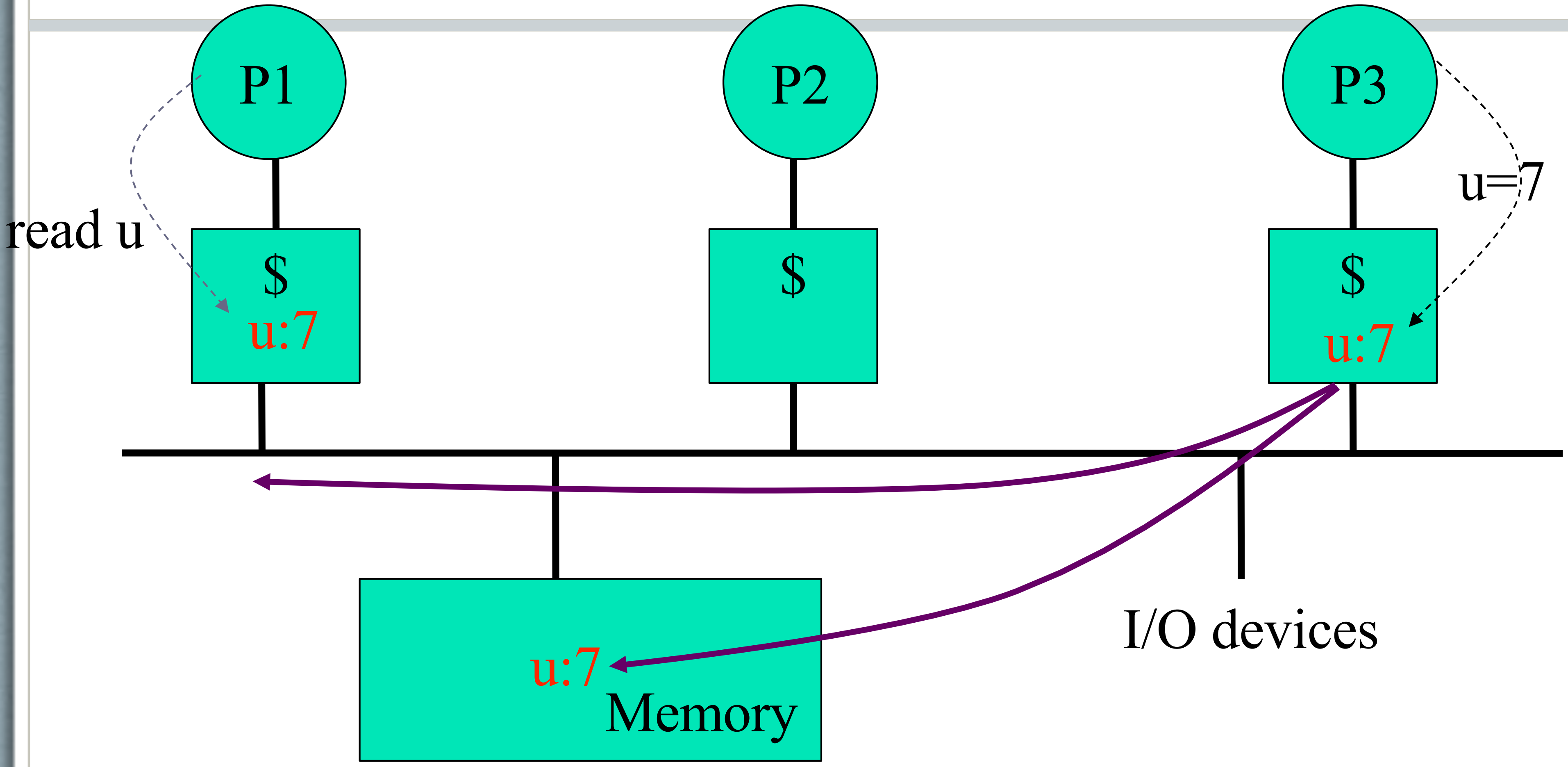
- Invalidate before P3 updating to get an exclusive access to the shared data

Example Revisited: Invalidate/Snoopy



- Invalidated copy triggers read miss: broadcast to all processors

Example Revisited: Invalidate/Snoopy



- P3 responds with updated shared data

Implementing Snooping-based Write-invalidate Protocol

- Key elements
  - Communicating with bus (or other broadcast medium)
    - Broadcast requests, monitor messages, respond if needed
  - Exclusive access before updating a shared block
    - Broadcasting invalidations before writing
  - Up-to-date copy might be in a cache
    - Snoop every address placed on the bus
    - Cache with a dirty copy of the requested block must respond

- Cache controller maintains the sharing status of all blocks contained in the cache
  - Valid/invalid, shared/exclusive, dirty/clean
  - Coherence protocols define block state transition as a finite-state machine