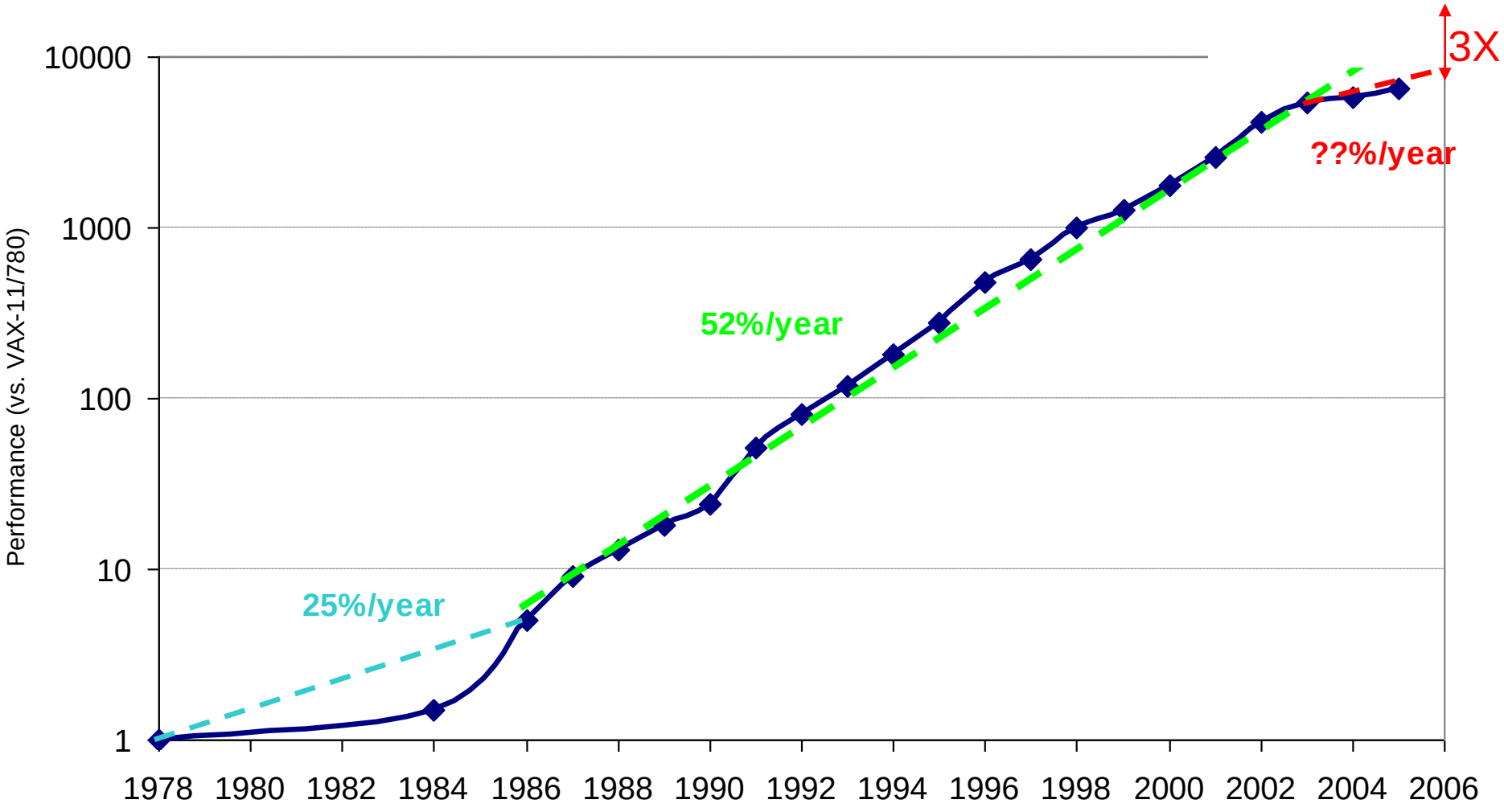
Multiprocessors and Thread-Level Parallelism

Uniprocessor Performance (SPECint)



Shift to Multiprocessors

"… today's processors … are nearing an impasse as technologies approach the speed of light.."

David Mitchell, *The Transputer: The Time Is Now* (1989)

- "We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

Paul Otellini, President, Intel (2005)

- All microprocessor companies switch to MP (2X CPUs / 2 yrs)

Manufacturer/Year	AMD/'05	Intel/'06	IBM/'04	Sun/'05
Processors/chip	2	2	2	8
Threads/Processor	1	2	2	4
Threads/chip	2	4	4	32

Other Factors $\Rightarrow$ Multiprocessors

- Growth in data-intensive applications
  - Data bases, file servers, …
- Growing interest in servers, server perf.
- Increasing desktop perf. less important
  - Outside of graphics
- Improved understanding in how to use multiprocessors effectively
  - Especially server where significant natural TLP
- Advantage of leveraging design investment by replication
  - Rather than unique design

M.J. Flynn, "Very High-Speed Computers", *Proc. of the IEEE*, V 54, 1900-1909, Dec. 1966.

Flynn's Taxonomy

- Flynn classified by data and control streams in 1966

Single Instruction Single Data (SISD) **(Uniprocessor)**	**Single Instruction Multiple Data SIMD** **(single PC: Vector, CM-2)**
Multiple Instruction Single Data (MISD) **(????)**	**Multiple Instruction Multiple Data MIMD** **(Clusters, SMP servers)**

- SIMD ⇒ Data Level Parallelism
- MIMD ⇒ Thread Level Parallelism

Back to Basics

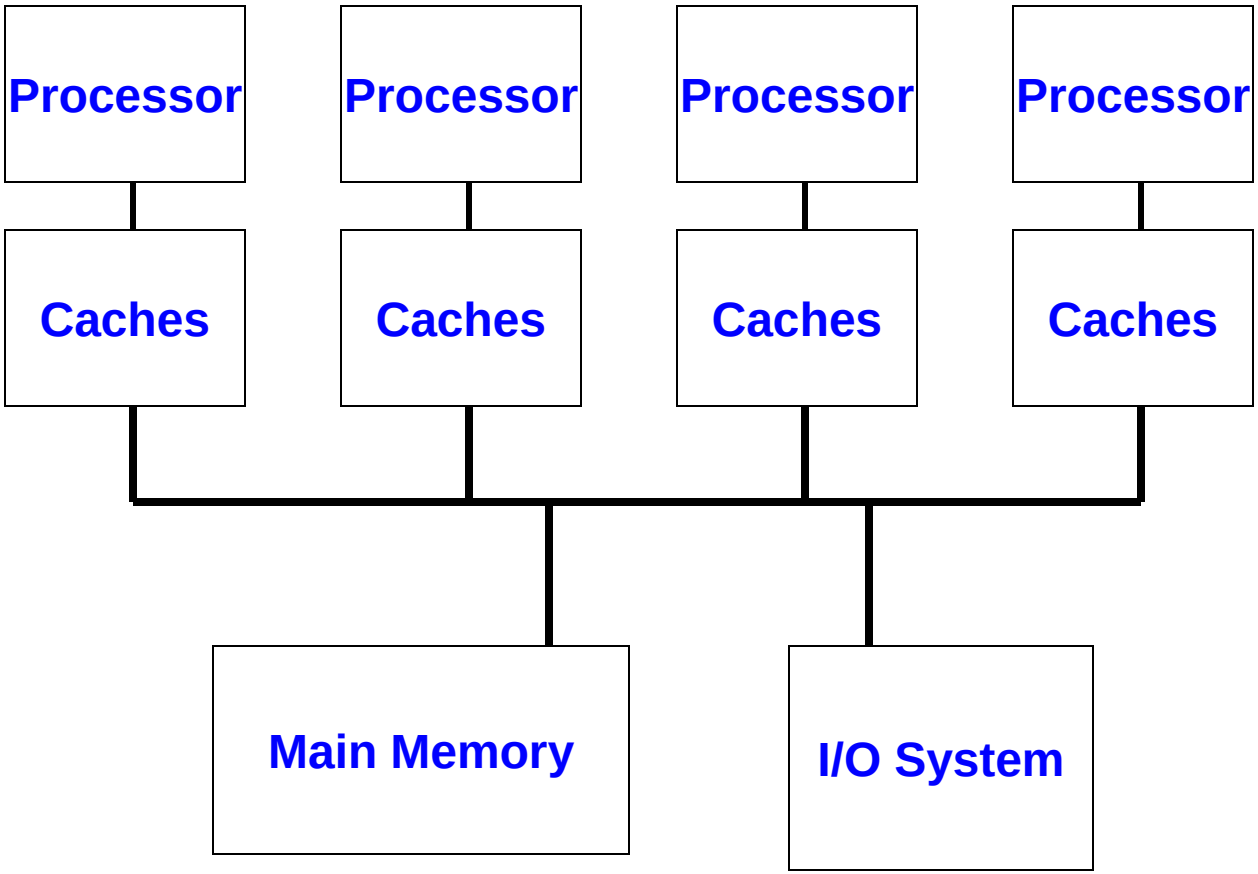
- "A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast."
- Parallel Architecture = Computer Architecture + Communication Architecture
- 2 classes of multiprocessors WRT memory:

1. Centralized Memory Multiprocessor
   - < few dozen processor chips (and < 100 cores) in 2006
   - Small enough to share single, centralized memory
2. Physically Distributed-Memory multiprocessor
   - Larger number chips and cores than 1.
   - BW demands ⇒ Memory distributed among processors

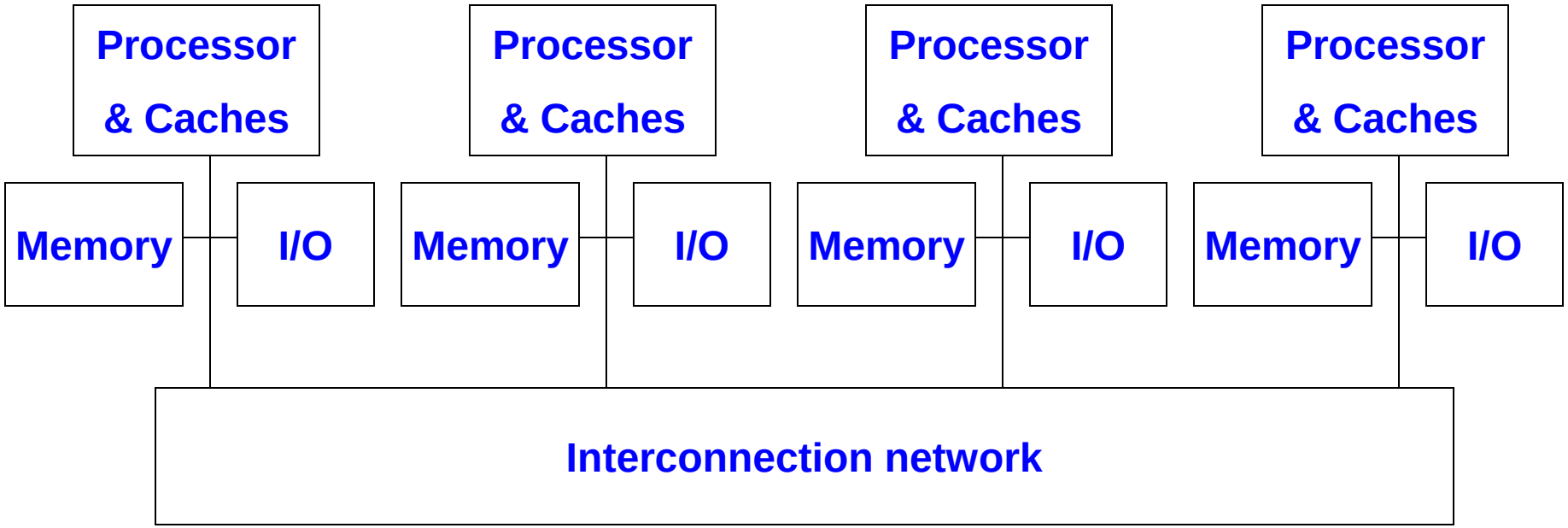
Centralized Memory Multiprocessor

- Also called symmetric multiprocessors (SMPs) because single main memory has a symmetric relationship to all processors
- Large caches $\Rightarrow$ single memory can satisfy memory demands of small number of processors
- Can scale to a few dozen processors by using a switch and by using many memory banks
- Although scaling beyond that is technically conceivable, it becomes less attractive as the number of processors sharing centralized memory increases

Centralized Shared-Memory



Distributed Memory Multiprocessors



Distributed Memory Multiprocessor

- Pro: Cost-effective way to scale memory bandwidth
  - If most accesses are to local memory
- Pro: Reduces latency of local memory accesses
- Con: Communicating data between processors more complex
- Con: Must change software to take advantage of increased memory BW

2 Models for Communication and Memory Architecture

1. Communication occurs by explicitly passing messages among the processors: message-passing multiprocessors
2. Communication occurs through a shared address space (via loads and stores): shared memory multiprocessors either
   - UMA (Uniform Memory Access time) for shared address, centralized memory MP
   - NUMA (Non Uniform Memory Access time multiprocessor) for shared address, distributed memory MP

Challenges of Parallel Processing

- First challenge is % of program inherently sequential
- Suppose 80X speedup from 100 processors. What fraction of original program can be sequential?

  a. 10%

  b. 5%

  c. 1%

  d. <1%

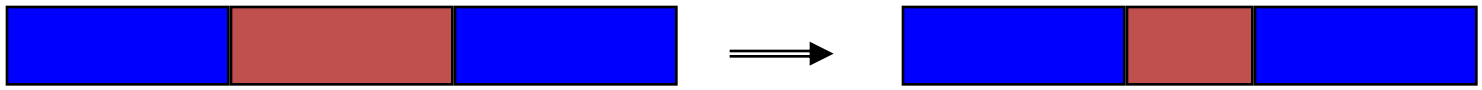
Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$



Challenges of Parallel Processing

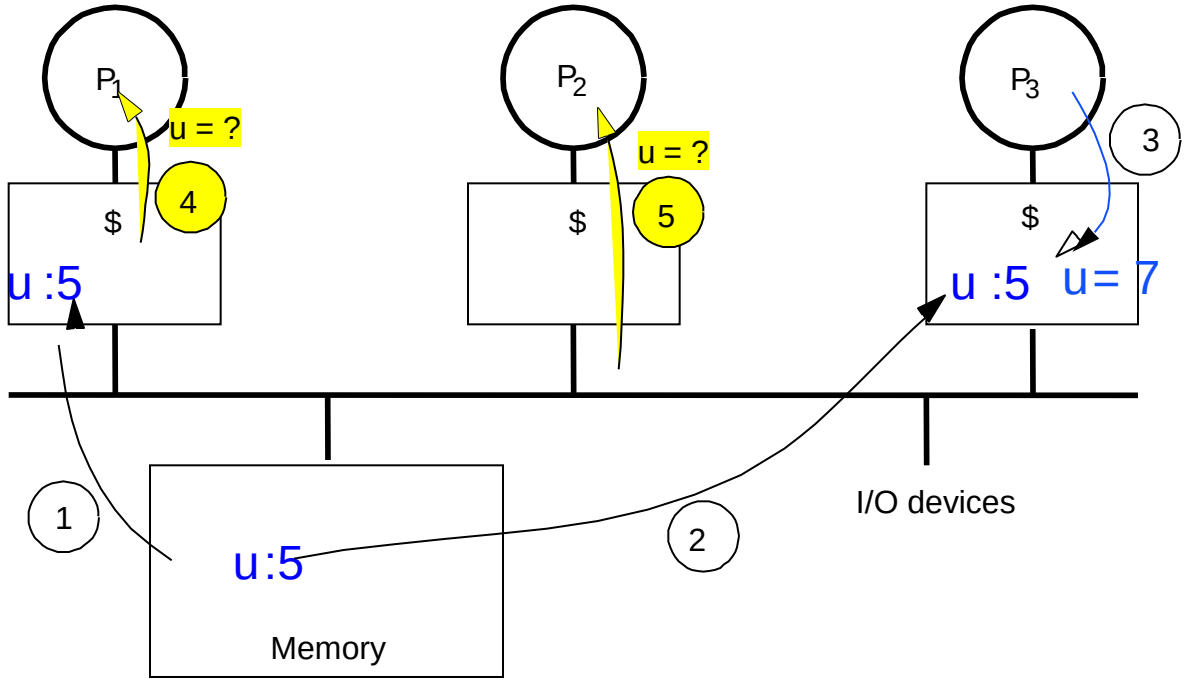
1. Application parallelism $\Rightarrow$ primarily via new algorithms that have better parallel performance
2. Long remote latency impact $\Rightarrow$ both by architect and by the programmer

- For example, reduce frequency of remote accesses either by
  - Caching shared data (HW)
  - Restructuring the data layout to make more accesses local (SW)

Symmetric Shared-Memory Architectures

- From multiple boards on a shared bus to multiple processors inside a single chip
- Caches both
  - 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 interconnect bandwidth
  ⇒ cache coherence problem

Example Cache Coherence Problem



- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on which cache flushes or writes back value when
- Unacceptable for programming, and its frequent!

Defining Coherent Memory System

1. Preserve Program Order: A read by processor P to location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P
2. Coherent view of memory: Read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses
3. Write serialization: 2 writes to same location by any 2 processors are seen in the same order by all processors

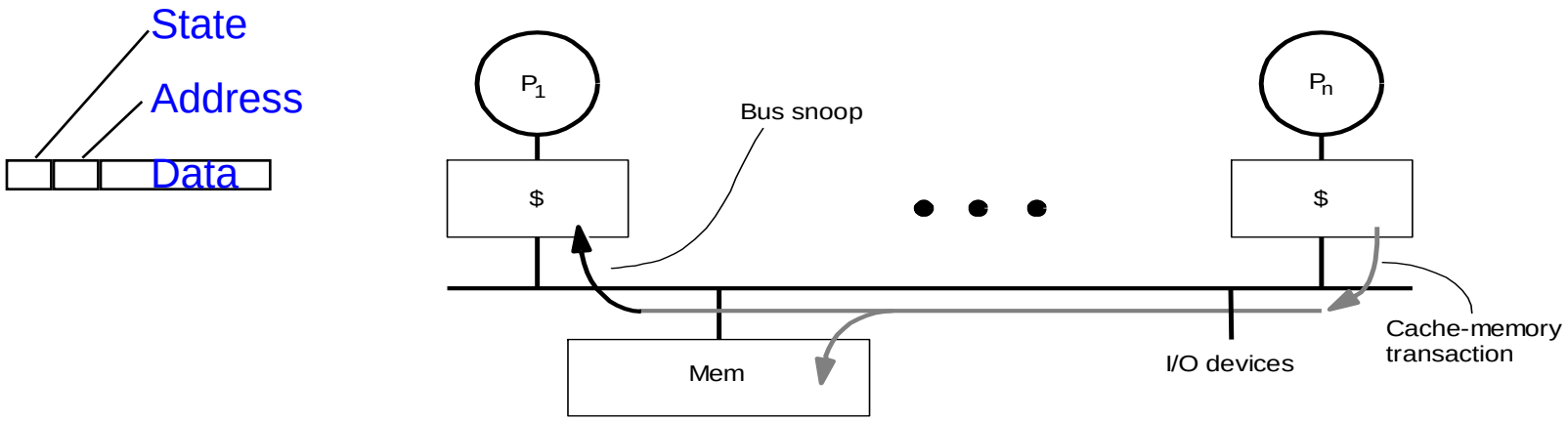
Basic Schemes for Enforcing Coherence

- Program on multiple processors will normally have copies of the same data in several caches
- Rather than trying to avoid sharing in SW, SMPs use a HW protocol to maintain coherent caches
  - Migration and Replication key to performance of shared data
- Migration - data can be moved to a local cache and used there in a transparent fashion
  - Reduces both latency to access shared data that is allocated remotely and bandwidth demand on the shared memory
- Replication – for shared data being simultaneously read, since caches make a copy of data in local cache
  - Reduces both latency of access and contention for read shared data

2 Classes of Cache Coherence Protocols

1. Directory based — Sharing status of a block of physical memory is kept in just one location, the directory
2. Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
   - All caches are accessible via some broadcast medium (a bus or switch)
   - All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus

Snoopy Cache-Coherence Protocols

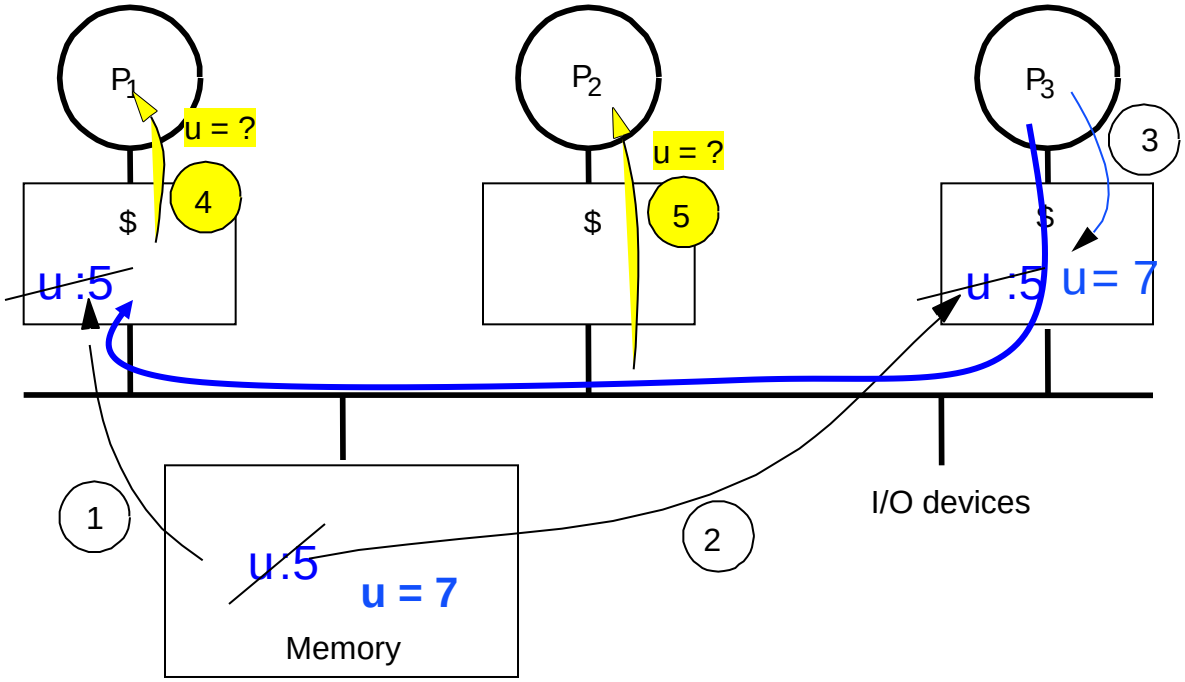


- Cache Controller "snoops" all transactions on the shared medium (bus or switch)
  - relevant transaction if for a block it contains
  - take action to ensure coherence
    - invalidate, update, or supply value
- Either get exclusive access before write via write invalidate or update all copies on write

Locate up-to-date copy of data

- Write-through: get up-to-date copy from memory
  - Write through simpler if enough memory BW
- Write-back harder
  - Most recent copy can be in a cache
- Can use same snooping mechanism
  1. Snoop every address placed on the bus
  2. If a processor has dirty copy of requested cache block, it provides it in response to a read request and aborts the memory access
  - Complexity from retrieving cache block from a processor cache, which can take longer than retrieving it from memory
- Write-back needs lower memory bandwidth
  $\Rightarrow$ Support larger numbers of faster processors
  $\Rightarrow$ Most multiprocessors use write-back

Example: Write-thru Invalidate



- Must invalidate before step 3
- Write update uses more broadcast medium BW
  $\Rightarrow$ all recent MPUs use write invalidate

Example Write Back Snoopy Protocol

- Invalidation protocol, write-back cache
  - Snoops every address on bus
  - If it has a dirty copy of requested block, provides that block in response to the read request and aborts the memory access
- Each memory block is in one state:
  - Clean in all caches and up-to-date in memory (Shared)
  - OR Dirty in exactly one cache (Exclusive)
  - OR Not in any caches
- Each cache block is in one state (track these):
  - Shared : block can be read
  - OR Exclusive : cache has only copy, its writeable, and dirty
  - OR Invalid : block contains no data (in uniprocessor cache too)
- Read misses: cause all caches to snoop bus
- Writes to clean blocks are treated as misses

Coherency Misses

1. True sharing misses arise from the communication of data through the cache coherence mechanism
   - Invalidates due to 1st write to shared block
   - Reads by another CPU of modified block in different cache
   - Miss would still occur if block size were 1 word
2. False sharing misses when a block is invalidated because some word in the block, other than the one being read, is written into
   - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
   - Block is shared, but no word in block is actually shared
     ⇒ miss would not occur if block size were 1 word

Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary

Synchronization

- Why Synchronize? Need to know when it is safe for different processes to use shared data
- Issues for Synchronization:
  - Uninterruptable instruction to fetch and update memory (atomic operation);
  - User level synchronization operation using this primitive;
  - For large scale MPs, synchronization can be a bottleneck; techniques to reduce contention and latency of synchronization

Multithreading on Multicore

- Basic idea: Processor resources are expensive and should not be left idle
- Long memory latency to memory on cache miss?
- Hardware switches threads to bring in other useful work while waiting for cache miss
- Cost of thread context switch must be much less than cache miss latency
- Put in redundant hardware so don't have to save context on every thread switch:
  - PC, Registers?
- Attractive for apps with abundant TLP