CS654 Advanced Computer Architecture

Lec 12 – Vector Wrap-up and Multiprocessor Introduction

Peter Kemper

Adapted from the slides of EECS 252 by Prof. David Patterson
Electrical Engineering and Computer Sciences
University of California, Berkeley

Outline

- Review
- Vector Metrics, Terms
- Cray 1 paper discussion
- MP Motivation
- SISD v. SIMD v. MIMD
- Centralized vs. Distributed Memory
- Challenges to Parallel Programming
- Consistency, Coherency, Write Serialization
- Write Invalidate Protocol
- Example
- Conclusion

Properties of Vector Processors

- Each result independent of previous result
  => long pipeline, compiler ensures no dependencies
  => high clock rate
- Vector instructions access memory with known pattern
  => highly interleaved memory
  => amortize memory latency of over - 64 elements
  => no (data) caches required! (Do use instruction cache)
- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work (- loop)
  => fewer instruction fetches

Operation & Instruction Count: RISC v. Vector Processor

(from F. Quintana, U. Barcelona.)

Spec92fp	Operations (Millions)			Instructions (M)		
Program	RISC	Vector	R / V	RISC	Vector	R / V
swim256	115	95	1.1x	115	0.8	142x
hydro2d	58	40	1.4x	58	0.8	71x
nasa7	69	41	1.7x	69	2.2	31x
su2cor	51	35	1.4x	51	1.8	29x
tomcatv	15	10	1.4x	15	1.3	11x
wave5	27	25	1.1x	27	7.2	4x
mdljdp2	32	52	0.6x	32	15.8	2x

Vector reduces ops by 1.2X, instructions by 20X

Common Vector Metrics

- R_∞: MFLOPS rate on an infinite-length vector
  - vector "speed of light"
  - Real problems do not have unlimited vector lengths, and the start-up penalties encountered in real problems will be larger
  - (R_n is the MFLOPS rate for a vector of length n)
- $N_{1/2}$: The vector length needed to reach one-half of R_∞
  - a good measure of the impact of start-up
- N_V: The vector length needed to make vector mode faster than scalar mode
  - measures both start-up and speed of scalars relative to vectors, quality of connection of scalar unit to vector unit

Vector Execution Time

- Time = f(vector length, data dependencies, struct. hazards)
- *Initiation rate*: rate that FU consumes vector elements (= number of lanes; usually 1 or 2 on Cray T-90)
- *Convoy*: set of vector instructions that can begin execution in same clock (no struct. or data hazards)
- *Chime*: approx. time for a vector operation
- *m* convoys take *m* chimes; if each vector length is n, then they take approx. *m* x *n* clock cycles (ignores overhead; good approximation for long vectors)

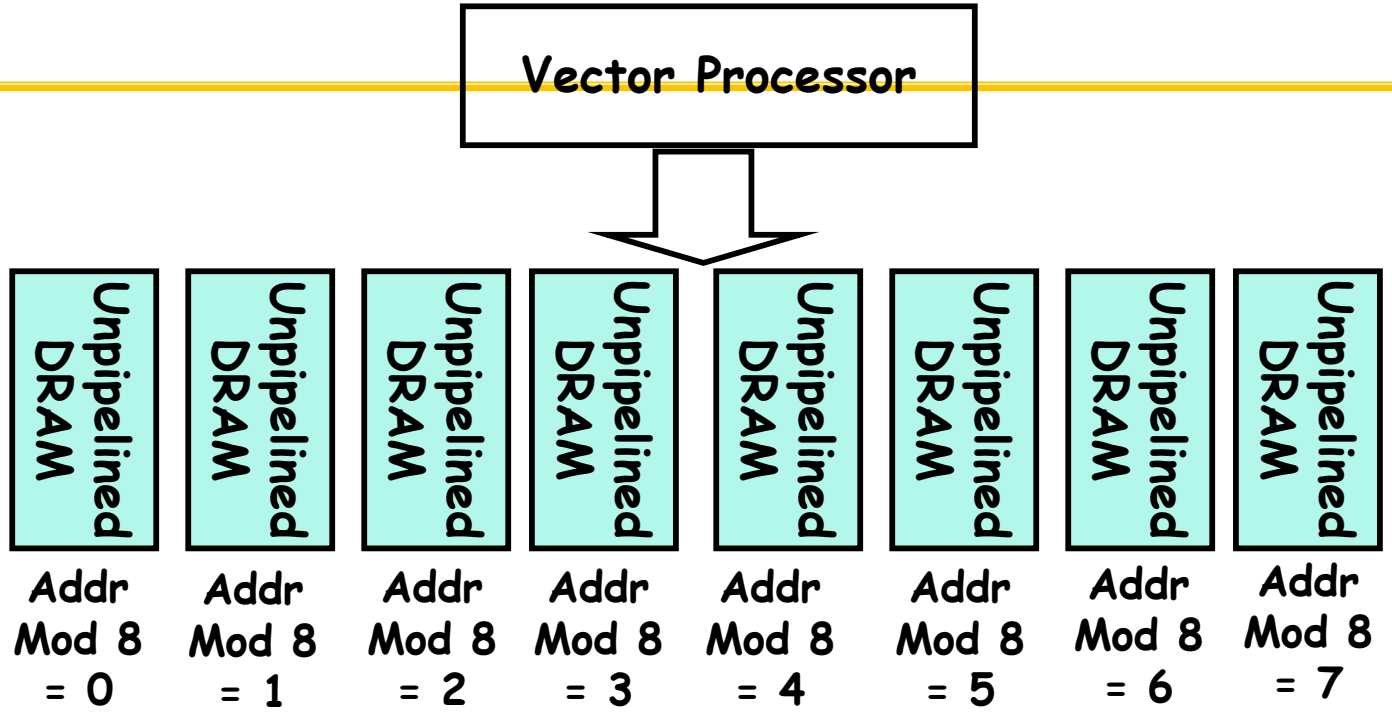
```
1:  LV    V1,Rx      ;load vector X
2:  MULV  V2,F0,V1   ;vector-scalar mult.
    LV    V3,Ry      ;load vector Y
3:  ADDV  V4,V2,V3   ;add
4:  SV    Ry,V4      ;store the result
```

4 convoys, 1 lane, VL=64
=> 4 x 64 = 256 clocks
(or 4 clocks per result)

Memory operations

- Load/store operations move groups of data between registers and memory
- Three types of addressing
  - Unit stride
    - » Contiguous block of information in memory
    - » Fastest: always possible to optimize this
  - Non-unit (constant) stride
    - » Harder to optimize memory system for all possible strides
    - » Prime number of data banks makes it easier to support different strides at full bandwidth
  - Indexed (gather-scatter)
    - » Vector equivalent of register indirect
    - » Good for sparse arrays of data
    - » Increases number of programs that vectorize

Interleaved Memory Layout



- **Great for unit stride:**
  - Contiguous elements in different DRAMs
  - Startup time for vector operation is latency of single read
- **What about non-unit stride?**
  - Above good for strides that are relatively prime to 8
  - Bad for: 2, 4
  - Better: prime number of banks…!

How to get full bandwidth for Unit Stride?

- Memory system must sustain (# lanes x word) /clock
- No. memory banks > memory latency to avoid stalls
  - *m* banks ⇒ *m* words per memory latency *l* clocks
  - if *m* < *l*, then gap in memory pipeline:

clock:	0	…	*l*	*l*+1	*l*+2	…	*l*+*m*- 1	l+m	…	2 *l*
word:	--	…	0	1	2	…	*m*-1	--	…	*m*

  - may have 1024 banks in SRAM
- If desired throughput greater than one word per cycle
  - Either more banks (start multiple requests simultaneously)
  - Or wider DRAMS. Only good for unit stride or large data types
- More banks/weird numbers of banks good to support more strides at full bandwidth
  - can read paper on how to do prime number of banks efficiently

Vectors Are Inexpensive

Scalar

- N ops per cycle
  $\Rightarrow O(N^2)$ circuitry
- HP PA-8000
  - 4-way issue
  - reorder buffer:
    850K transistors
    - incl. 6,720 5-bit register number comparators

Vector

- N ops per cycle
  $\Rightarrow O(N + \varepsilon N^2)$ circuitry
- T0 vector micro

  (Torrent-0 vector microprocessor, 1995)
  - 24 ops per cycle
  - 730K transistors total
    - only 23 5-bit register number comparators

Vectors Lower Power

Single-issue Scalar

- One instruction fetch, decode, dispatch per operation
- Arbitrary register accesses, adds area and power
- Loop unrolling and software pipelining for high performance increases instruction cache footprint
- All data passes through cache; waste power if no temporal locality
- One TLB lookup per load or store
- Off-chip access in whole cache lines

Vector

- One inst fetch, decode, dispatch per vector
- Structured register accesses
- Smaller code for high performance, less power in instruction cache misses
- Bypass cache
- One TLB lookup per group of loads or stores
- Move only necessary data across chip boundary

Superscalar Energy Efficiency Even Worse

Superscalar

- Control logic grows quadratically with issue width
- Control logic consumes energy regardless of available parallelism
- Speculation to increase visible parallelism wastes energy

Vector

- Control logic grows linearly with issue width
- Vector unit switches off when not in use
- Vector instructions expose parallelism without speculation
- Software control of speculation when desired:
  - Whether to use vector mask or compress/expand for conditionals

Vector Applications

Limited to scientific computing?

- **Multimedia Processing** (compress., graphics, audio synth, image proc.)
- **Standard benchmark kernels** (Matrix Multiply, FFT, Convolution, Sort)
- **Lossy Compression** (JPEG, MPEG video and audio)
- **Lossless Compression** (Zero removal, RLE, Differencing, LZW)
- **Cryptography** (RSA, DES/IDEA, SHA/MD5)
- **Speech and handwriting recognition**
- **Operating systems/Networking** (`memcpy`, `memset`, parity, checksum)
- **Databases** (hash/join, data mining, image/video serving)
- **Language run-time support** (stdlib, garbage collection)
- **even SPECint95**

Older Vector Machines

Machine	Year	Clock	Regs	Elements	FUs	LSUs
Cray 1	1976	80 MHz	8	64	6	1
Cray XMP	1983	120 MHz	8	64	8	2 L, 1 S
Cray YMP	1988	166 MHz	8	64	8	2 L, 1 S
Cray C-90	1991	240 MHz	8	128	8	4
Cray T-90	1996	455 MHz	8	128	8	4
Conv. C-1	1984	10 MHz	8	128	4	1
Conv. C-4	1994	133 MHz	16	128	3	1
Fuj. VP200	1982	133 MHz	8-256	32-1024	3	2
Fuj. VP300	1996	100 MHz	8-256	32-1024	3	2
NEC SX/2	1984	160 MHz	8+8K	256+var	16	8
NEC SX/3	1995	400 MHz	8+8K	256+var	16	8

Newer Vector Computers

- Cray X1
  - MIPS like ISA + Vector in CMOS
- NEC Earth Simulator
  - Fastest computer in world for 3 years; 40 TFLOPS
  - 640 CMOS vector nodes

Recent Supercomputers:

- IBM Blue Gene
- IBM Roadrunner
  - Cell / AMD Opteron based

Key Architectural Features of X1

New vector instruction set architecture (ISA)

- Much larger register set (32x64 vector, 64+64 scalar)
- 64- and 32-bit memory and IEEE arithmetic
- Based on 25 years of experience compiling with Cray1 ISA

Decoupled Execution

- Scalar unit runs ahead of vector unit, doing addressing and control
- Hardware dynamically unrolls loops, and issues multiple loops concurrently
- Special sync operations keep pipeline full, even across barriers

⇒ Allows the processor to perform well on short nested loops

Scalable, distributed shared memory (DSM) architecture

- Memory hierarchy: caches, local memory, remote memory
- Low latency, load/store access to entire machine (tens of TBs)
- Processors support 1000's of outstanding refs with flexible addressing
- Very high bandwidth network
- Coherence protocol, addressing and synchronization optimized for DM

Cray X1E Mid-life Enhancement

- Technology refresh of the X1 (0.13μm)
  - ~50% faster processors
  - Scalar performance enhancements
  - Doubling processor density
  - Modest increase in memory system bandwidth
  - Same interconnect and I/O
- Machine upgradeable
  - Can replace Cray X1 nodes with X1E nodes
- released 2005

ESS – configuration of a general purpose supercomputer

1. **Processor Nodes (PN) Total number of processor nodes is 640. Each processor node consists of eight vector processors of 8 GFLOPS and 16GB shared memories. Therefore, total numbers of processors is 5,120 and total peak performance and main memory of the system are 40 TFLOPS and 10 TB, respectively. Two nodes are installed into one cabinet, which size is 40"x56"x80". 16 nodes are in a cluster. Power consumption per cabinet is approximately 20 KVA.**

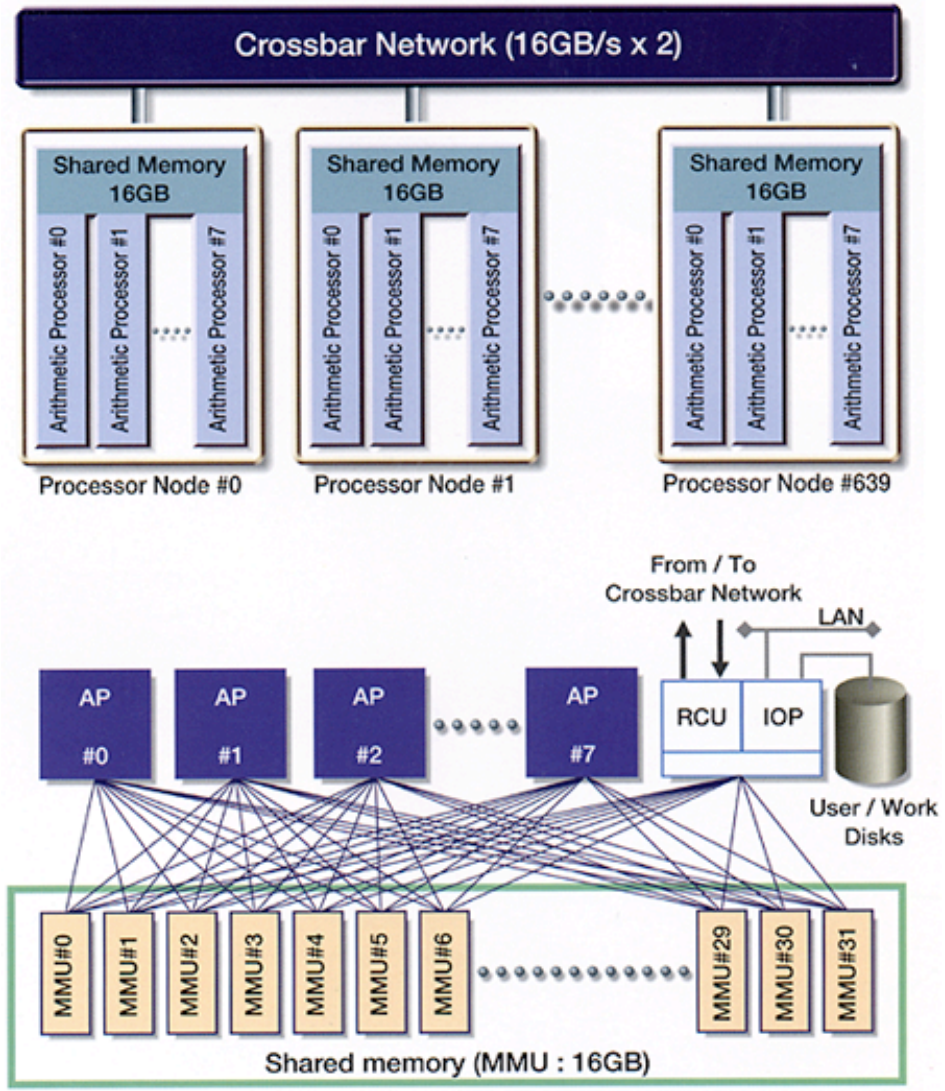
2) Interconnection Network (IN): Each node is coupled together with more than 83,000 copper cables via single-stage crossbar switches of 16GB/s x2 (Load + Store). The total length of the cables is approximately 1,800 miles.

3) Hard Disk. Raid disks are used for the system. The capacities are 450 TB for the systems operations and 250 TB for users.

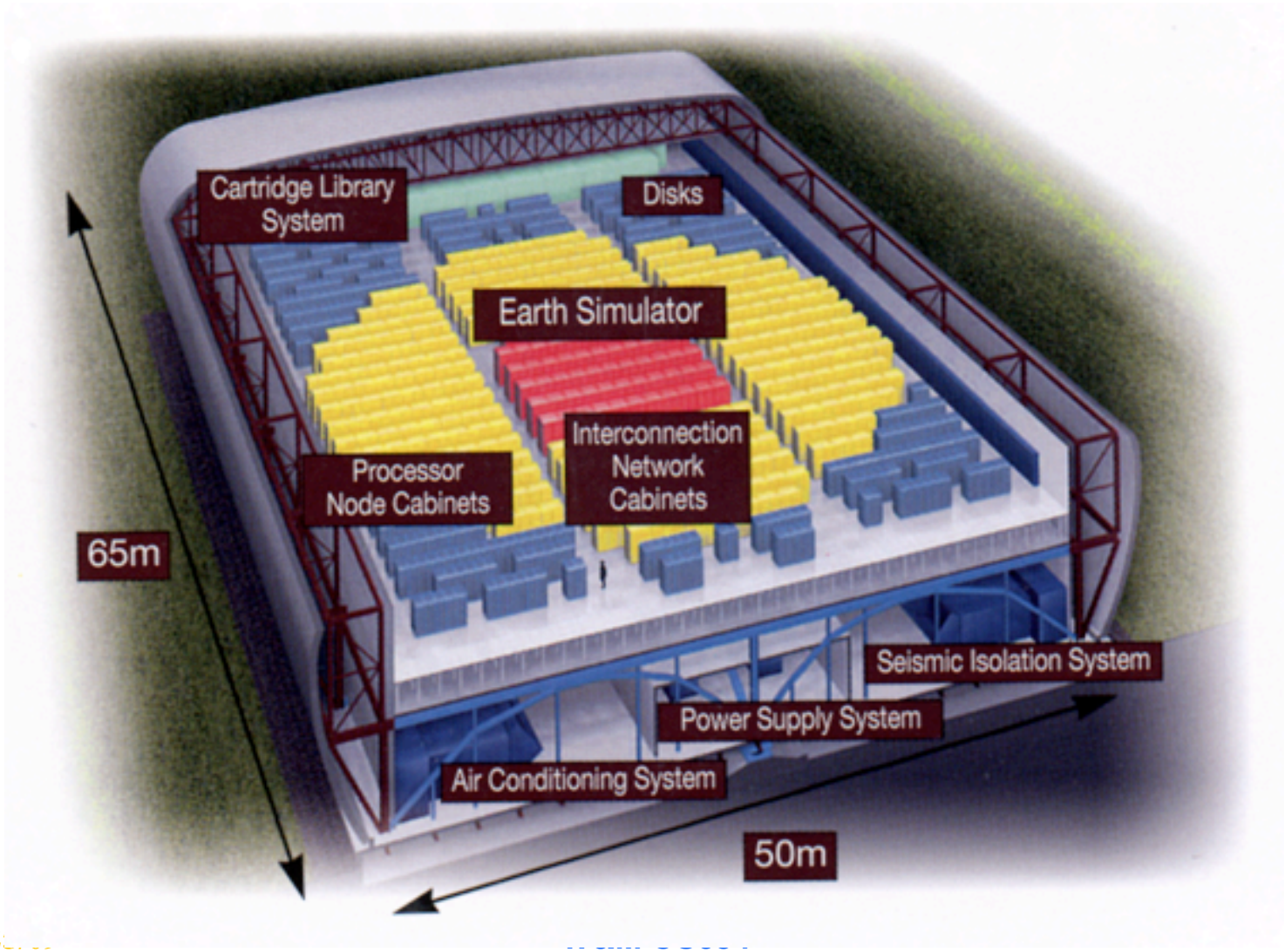
4) Mass Storage system: 12 Automatic Cartridge Systems (STK PowderHorn9310); total storage capacity is approximately 1.6 PB.

From Horst D. Simon, NERSC/LBNL, May 15, 2002, "ESS Rapid Response Meeting"

Earth Simulator



Earth Simulator Building



ESS – complete system installed 4/1/2002

Vector Summary

- Vector is alternative model for exploiting ILP
- If code is vectorizable, then simpler hardware, more energy efficient, and better real-time model than Out-of-order machines
- Design issues include number of lanes, number of functional units, number of vector registers, length of vector registers, exception handling, conditional operations
- Fundamental design issue is memory bandwidth
  - With virtual address translation and caching
- Will multimedia popularity revive vector architectures?

"The CRAY-1 computer system"

- by R.M. Russell, *Comm. of the ACM*, January 1978
- Number of functional units?
  - Compared to today?
- Clock rate?
  - Why so fast?
  - How balance clock cycle?
- Size of register state?
- Memory size?
- Memory latency?
  - Compared to today?
- "4 most striking features?"
- Instruction set architecture?
- Virtual Memory? Relocation? Protection?

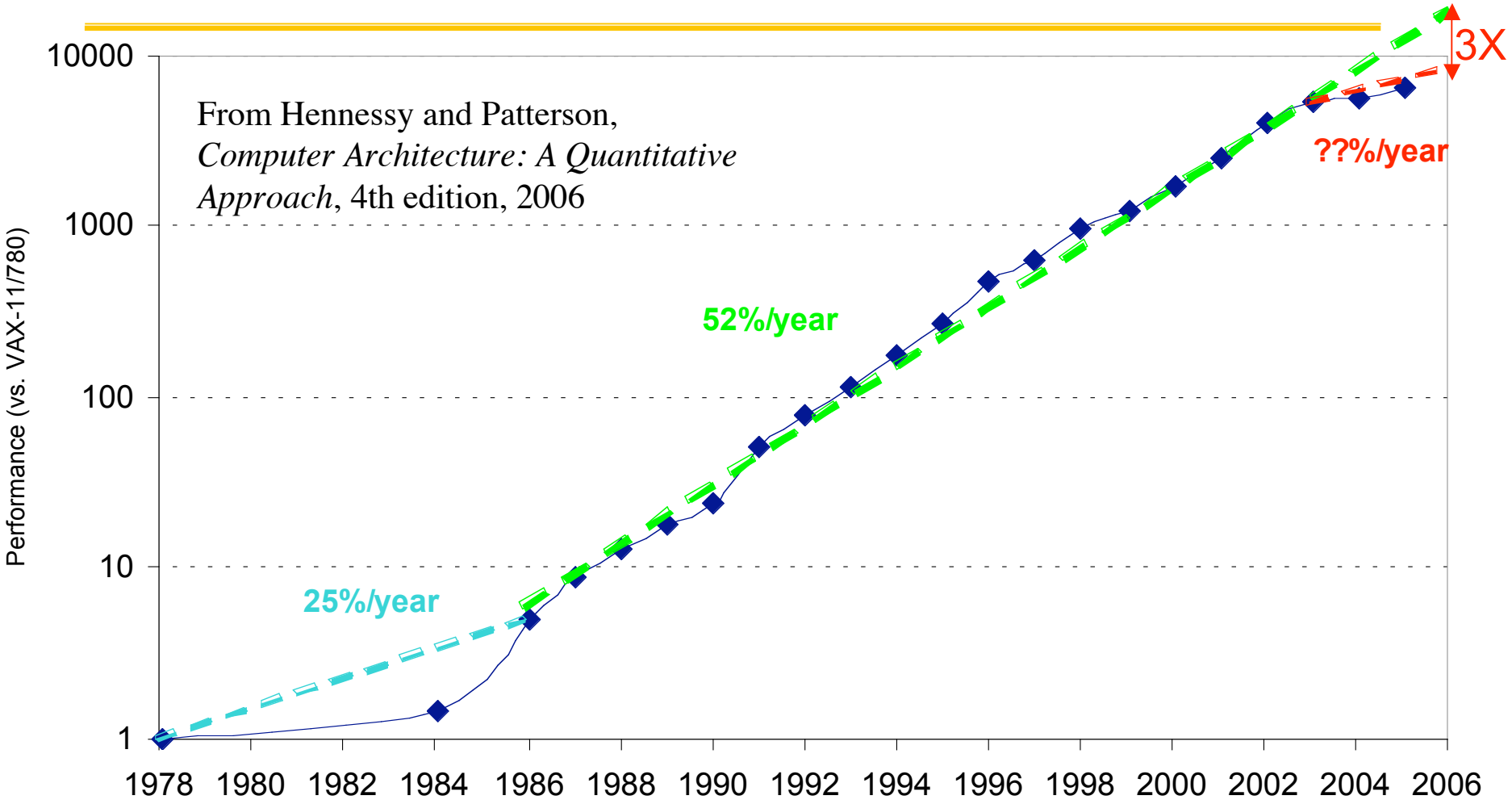
"The CRAY-1 computer system"

- Floating Point Format?
  - How differs from IEEE 754 FP?
- Vector vs. scalar speed?
- Min. size vector faster than scalar loop?
- What meant by "long vector vs. short vector" computer?
- Relative speed to other computers?
  - Of its era?
  - Pentium-4 or AMD 64?
- General impressions compared to today's CPUs

Outline

- Review
- Vector Metrics, Terms
- Cray 1 paper discussion
- **MP Motivation**
- **SISD v. SIMD v. MIMD**
- **Centralized vs. Distributed Memory**
- **Challenges to Parallel Programming**
- **Consistency, Coherency, Write Serialization**
- **Write Invalidate Protocol**
- **Example**
- **Conclusion**

Uniprocessor Performance (SPECint)



- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: ??%/year 2002 to present**

Déjà vu all over again?

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

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

- Transputer had bad timing (Uniprocessor performance↑)
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- "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)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 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 ⇒ 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

Flynn's Taxonomy

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

- 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)

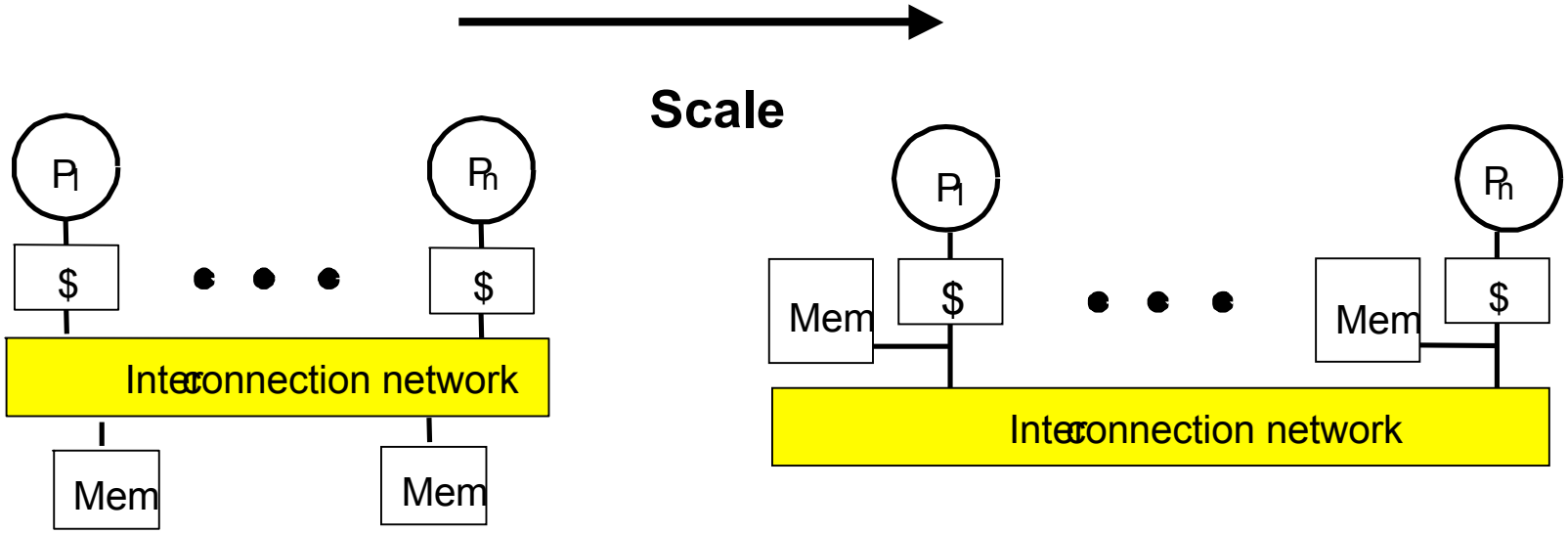
- SIMD ⇒ Data Level Parallelism
- MIMD ⇒ Thread Level Parallelism
- MIMD popular because
  - Flexible: N pgms and 1 multithreaded pgm
  - Cost-effective: same MPU in desktop & MIMD

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 vs. Distributed Memory



Centralized Memory **Distributed Memory**

Centralized Memory Multiprocessor

- Also called symmetric multiprocessors (SMPs) because single main memory has a symmetric relationship to all processors
- Large caches ⇒ 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

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

- In past, confusion whether "sharing" means sharing physical memory (Symmetric MP) or sharing address space

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 Answers

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{parallel}}}{\text{Speedup}_{\text{parallel}}}}$$

$$80 = \frac{1}{(1 - \text{Fraction}_{\text{parallel}}) + \frac{\text{Fraction}_{\text{parallel}}}{100}}$$

$$80 \times ((1 - \text{Fraction}_{\text{parallel}}) + \frac{\text{Fraction}_{\text{parallel}}}{100}) = 1$$

$$79 = 80 \times \text{Fraction}_{\text{parallel}} - 0.8 \times \text{Fraction}_{\text{parallel}}$$

$$\text{Fraction}_{\text{parallel}} = 79 / 79.2 = 99.75\%$$

Challenges of Parallel Processing

- Second challenge is long latency to remote memory
- Suppose 32 CPU MP, 2GHz, 200 ns remote memory, all local accesses hit memory hierarchy and base CPI is 0.5. (Remote access = 200/0.5 = 400 clock cycles.)
- What is performance impact if 0.2% instructions involve remote access?

  a. 1.5X

  b. 2.0X

  c. 2.5X

CPI Equation

- CPI = Base CPI +
  Remote request rate
  x Remote request cost
- CPI = 0.5 + 0.2% x 400 = 0.5 + 0.8 = 1.3
- No communication is 1.3/0.5 or 2.6 faster than 0.2% instructions involve local access

Challenges of Parallel Processing

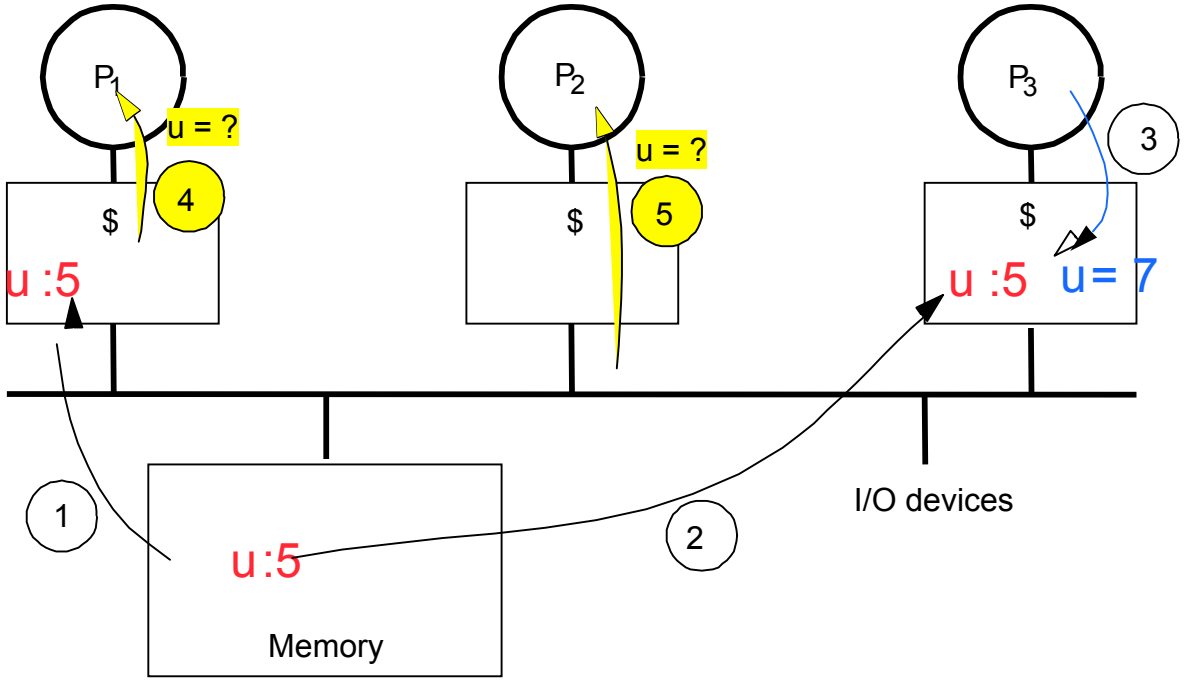
1. Application parallelism ⇒ primarily via new algorithms that have better parallel performance
2. Long remote latency impact ⇒ both by architect and by the programmer

- For example, reduce frequency of remote accesses either by
  - Caching shared data (HW)
  - Restructuring the data layout to make more accesses local (SW)
- Today's lecture on HW to help latency via caches

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

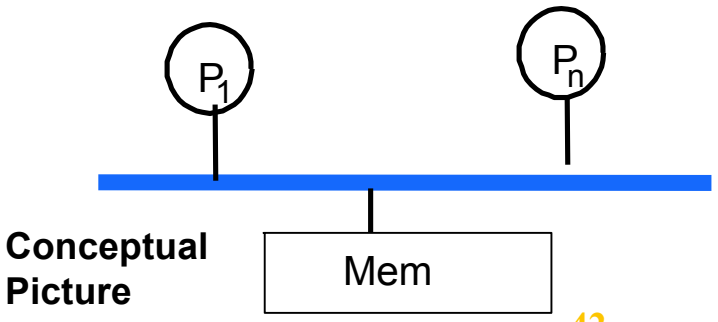


- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value when
  - » Processes accessing main memory may see very stale value
- Unacceptable for programming, and its frequent!

Example

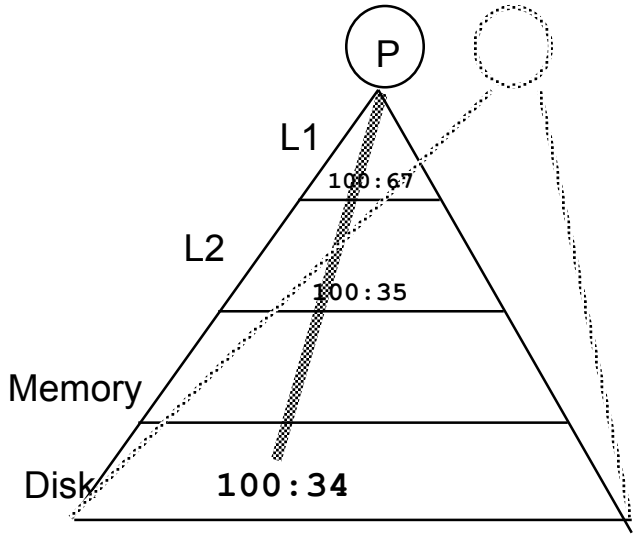
P_1	P_2
/*Assume initial value of A and flag is 0*/	
`A = 1;`	`while (flag == 0);` /*spin idly*/
`flag = 1;`	`print A;`

- **Intuition not guaranteed by coherence**
- **expect memory to respect order between accesses to *different* locations issued by a given process**
  - **to preserve orders among accesses to same location by different processes**
- **Coherence is not enough!**
  - **pertains only to single location**



Conceptual Picture

Intuitive Memory Model



- **Reading an address should return the last value written to that address**
  - **Easy in uniprocessors, except for I/O**

- **Too vague and simplistic; 2 issues**

1. **Coherence defines values returned by a read**
2. **Consistency determines when a written value will be returned by a read**

- **Coherence defines behavior to same location, Consistency defines behavior to other locations**

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
   - If not, a processor could keep value 1 since saw as last write
   - For example, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1

Write Consistency

- For now assume

1. A write does not complete (and allow the next write to occur) until all processors have seen the effect of that write
2. The processor does not change the order of any write with respect to any other memory access

⇒ if a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

- These restrictions allow the processor to reorder reads, but forces the processor to finish writes in program order

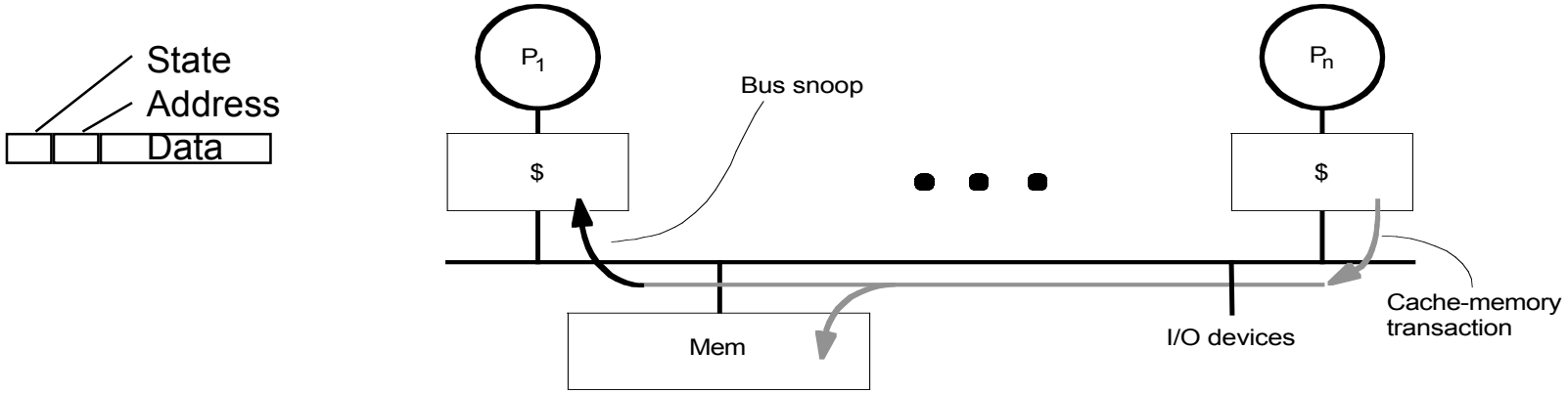
Basic Schemes for Enforcing Coherence

- Program on multiple processors will normally have copies of the same data in several caches
  - Unlike I/O, where its rare
- 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

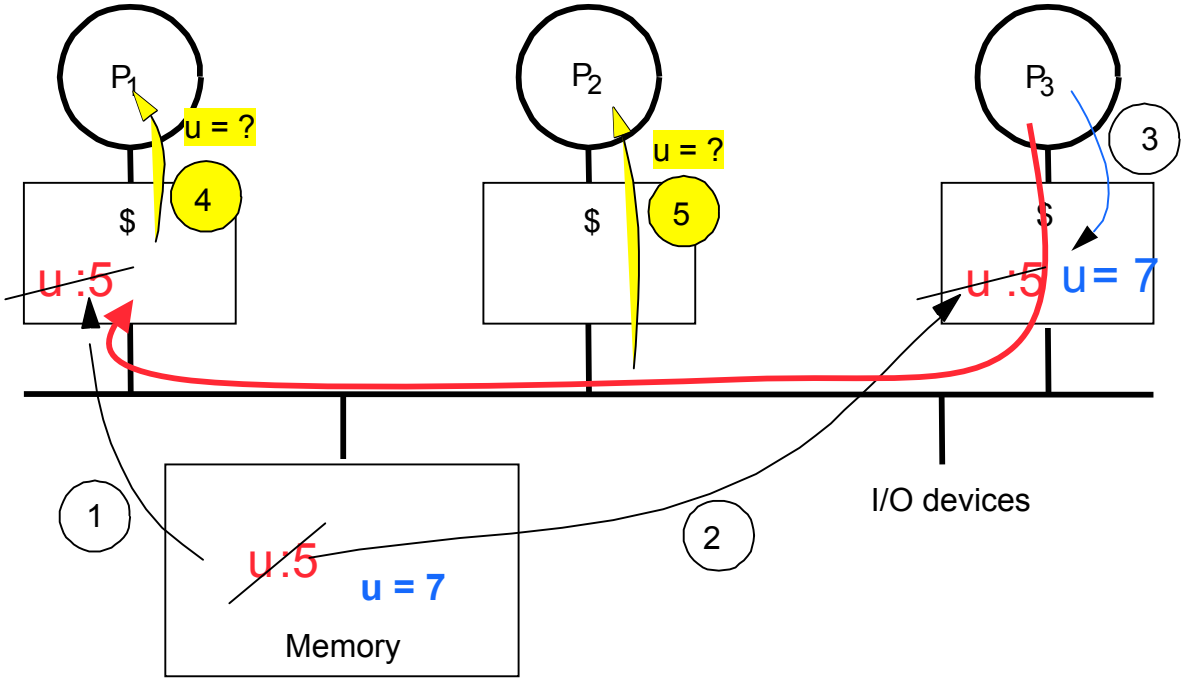
1. Directory based — Sharing status of a block of physical memory is kept in just one location, the directory
2. Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
   - All caches are accessible via some broadcast medium (a bus or switch)
   - 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 or switch access

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**
  - **depends on state of the block and the protocol**
- **Either get exclusive access before write via write invalidate or update all copies on write**

Example: Write-thru Invalidate



- **Must invalidate before step 3**
- **Write update uses more broadcast medium BW ⇒ all recent MPUs use write invalidate**

Architectural Building Blocks

- Cache block state transition diagram
  - FSM specifying how disposition of block changes
    - » invalid, valid, dirty
- Broadcast Medium Transactions (e.g., bus)
  - Fundamental system design abstraction
  - Logically single set of wires connect several devices
  - Protocol: arbitration, command/addr, data

  ⇒ Every device observes every transaction
- Broadcast medium enforces serialization of read or write accesses ⇒ Write serialization
  - 1st processor to get medium invalidates others copies
  - Implies cannot complete write until it obtains bus
  - All coherence schemes require serializing accesses to same cache block
- Also need to find up-to-date copy of cache block

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**
  ⇒ Support larger numbers of faster processors
  ⇒ Most multiprocessors use write-back

Cache Resources for WB Snooping

- Normal cache tags can be used for snooping
- Valid bit per block makes invalidation easy
- Read misses easy since rely on snooping
- Writes ⇒ Need to know if know whether any other copies of the block are cached
  - No other copies ⇒ No need to place write on bus for WB
  - Other copies ⇒ Need to place invalidate on bus

Cache Resources for WB Snooping

- To track whether a cache block is shared, add extra state bit associated with each cache block, like valid bit and dirty bit
  - Write to Shared block ⇒ Need to place invalidate on bus and mark cache block as private (if an option)
  - No further invalidations will be sent for that block
  - This processor called owner of cache block
  - Owner then changes state from shared to unshared (or exclusive)

Cache behavior in response to bus

- Every bus transaction must check the cache-address tags
  - could potentially interfere with processor cache accesses
- A way to reduce interference is to duplicate tags
  - One set for caches access, one set for bus accesses
- Another way to reduce interference is to use L2 tags
  - Since L2 less heavily used than L1

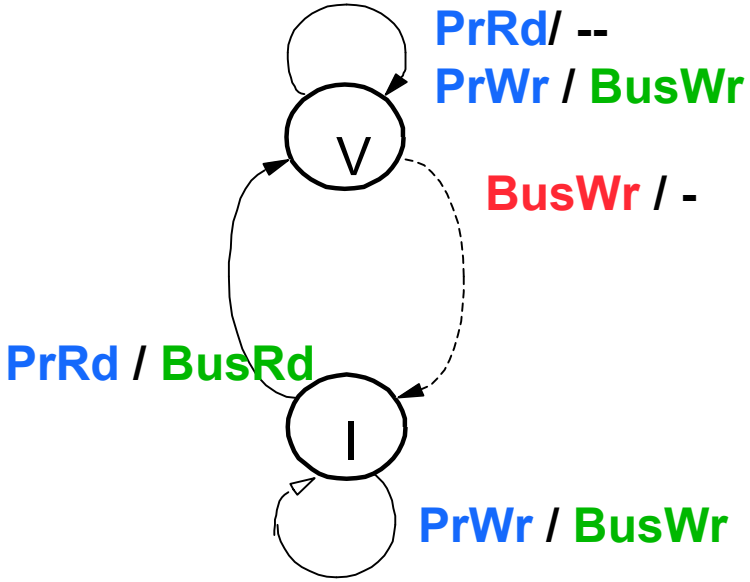
  ⇒ Every entry in L1 cache must be present in the L2 cache, called the inclusion property
  - If Snoop gets a hit in L2 cache, then it must arbitrate for the L1 cache to update the state and possibly retrieve the data, which usually requires a stall of the processor

Example Protocol

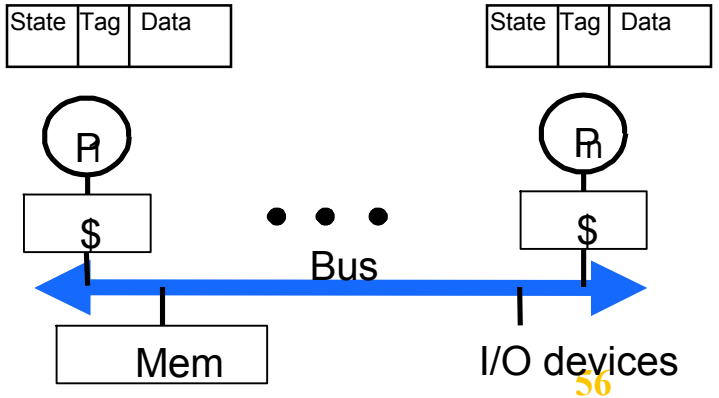
- Snooping coherence protocol is usually implemented by incorporating a finite-state controller in each node
- Logically, think of a separate controller associated with each cache block
  - That is, snooping operations or cache requests for different blocks can proceed independently
- In implementations, a single controller allows multiple operations to distinct blocks to proceed in interleaved fashion
  - that is, one operation may be initiated before another is completed, even through only one cache access or one bus access is allowed at time

Write-through Invalidate Protocol

- **2 states per block in each cache**
  - as in uniprocessor
  - state of a block is a *p*-vector of states
  - Hardware state bits associated with blocks that are in the cache
  - other blocks can be seen as being in invalid (not-present) state in that cache
- **Writes invalidate all other cache copies**
  - can have multiple simultaneous readers of block,but write invalidates them



PrRd: Processor Read
PrWr: Processor Write
BusRd: Bus Read
BusWr: Bus Write

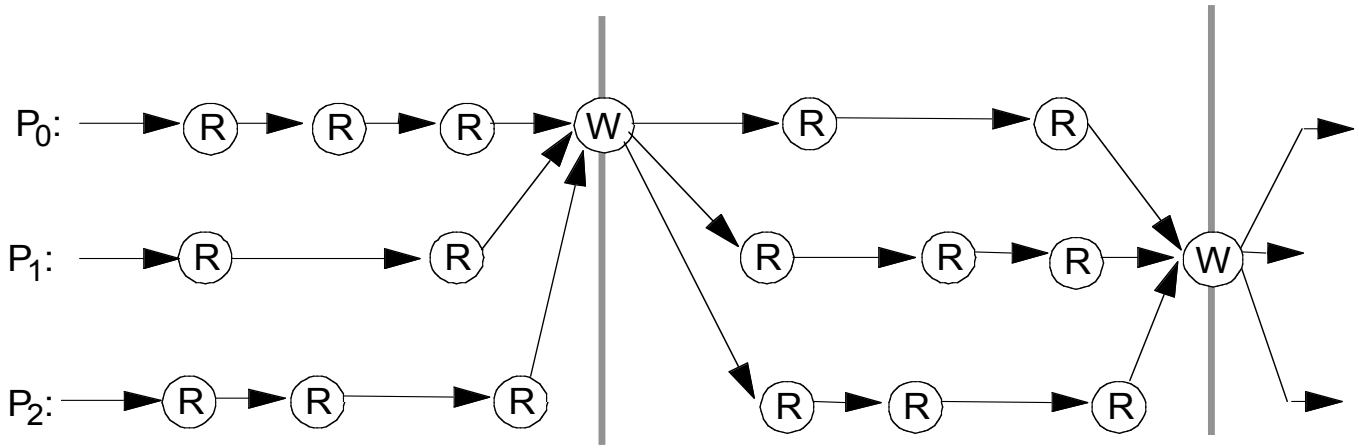


Is 2-state Protocol Coherent?

- Processor only observes state of memory system by issuing memory operations
- Assume bus transactions and memory operations are atomic and a one-level cache
  - all phases of one bus transaction complete before next one starts
  - processor waits for memory operation to complete before issuing next
  - with one-level cache, assume invalidations applied during bus transaction
- All writes go to bus + atomicity
  - Writes serialized by order in which they appear on bus (bus order)

  => invalidations applied to caches in bus order
- How to insert reads in this order?
  - Important since processors see writes through reads, so determines whether write serialization is satisfied
  - But read hits may happen independently and do not appear on bus or enter directly in bus order
- Let's understand other ordering issues

Ordering



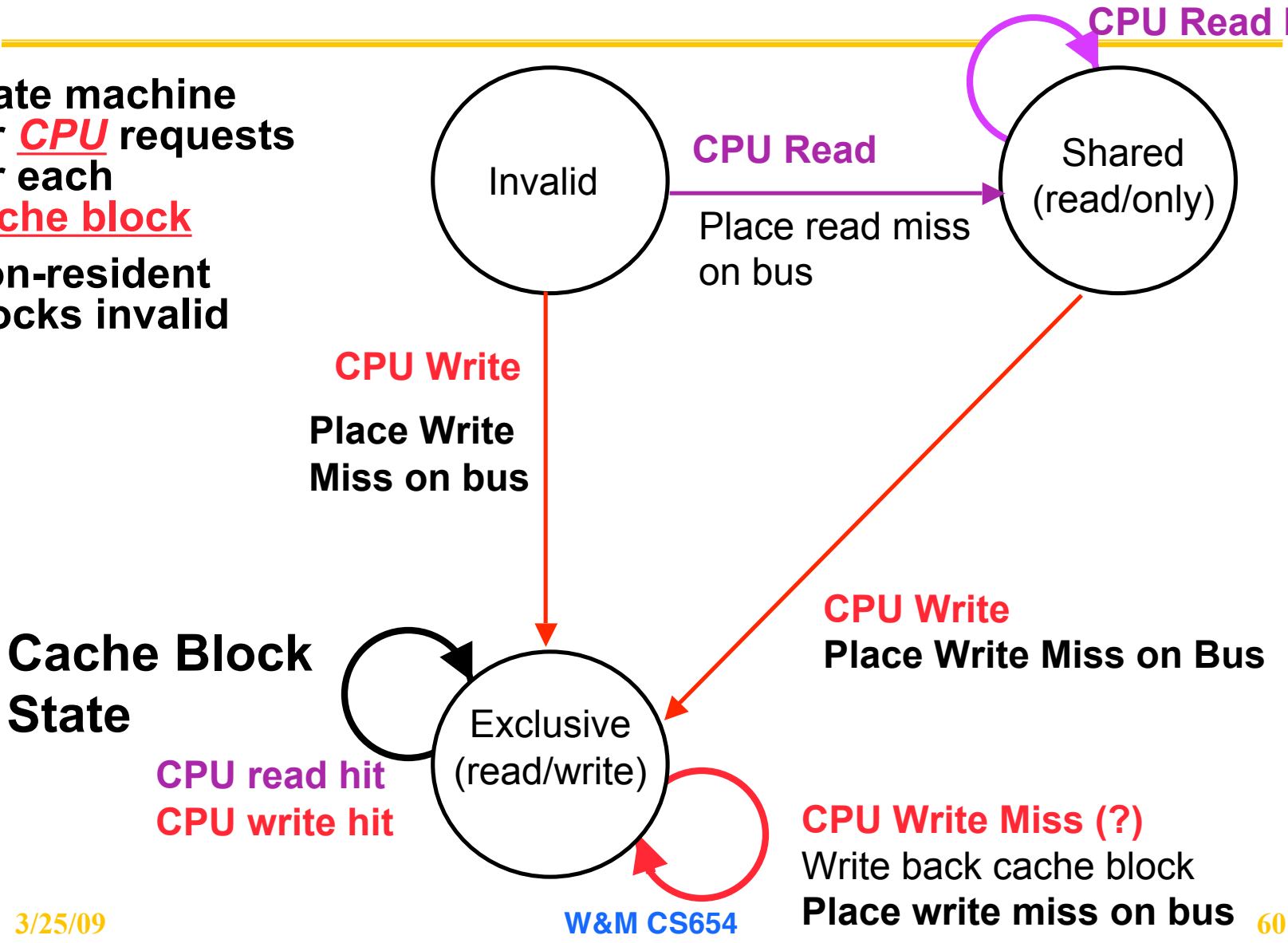
- **Writes establish a partial order**
- **Doesn't constrain ordering of reads, though shared-medium (bus) will order read misses too**
  - **any order among reads between writes is fine, as long as in program order**

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

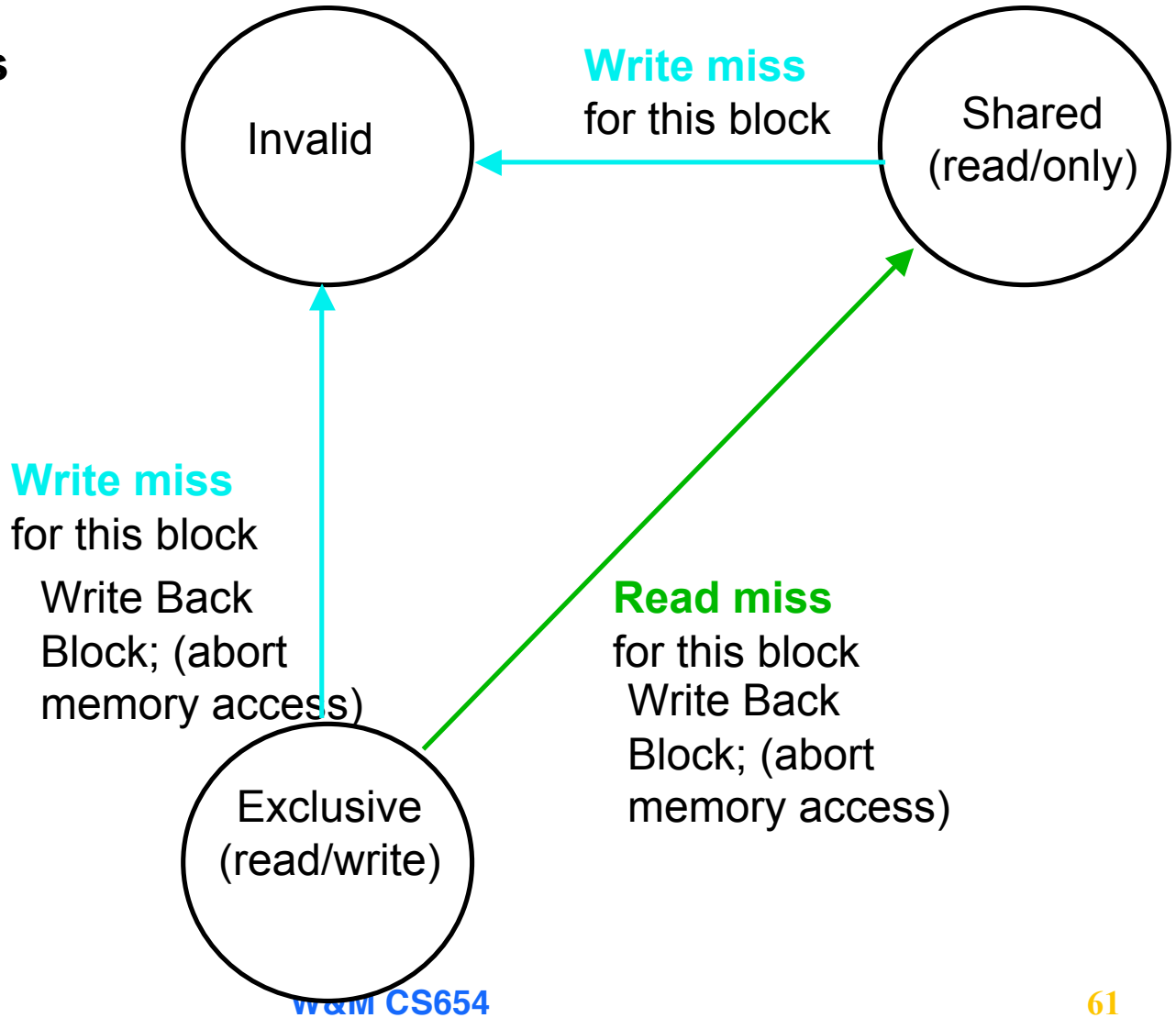
Write-Back State Machine - CPU

- State machine for *CPU* requests for each cache block
- Non-resident blocks invalid



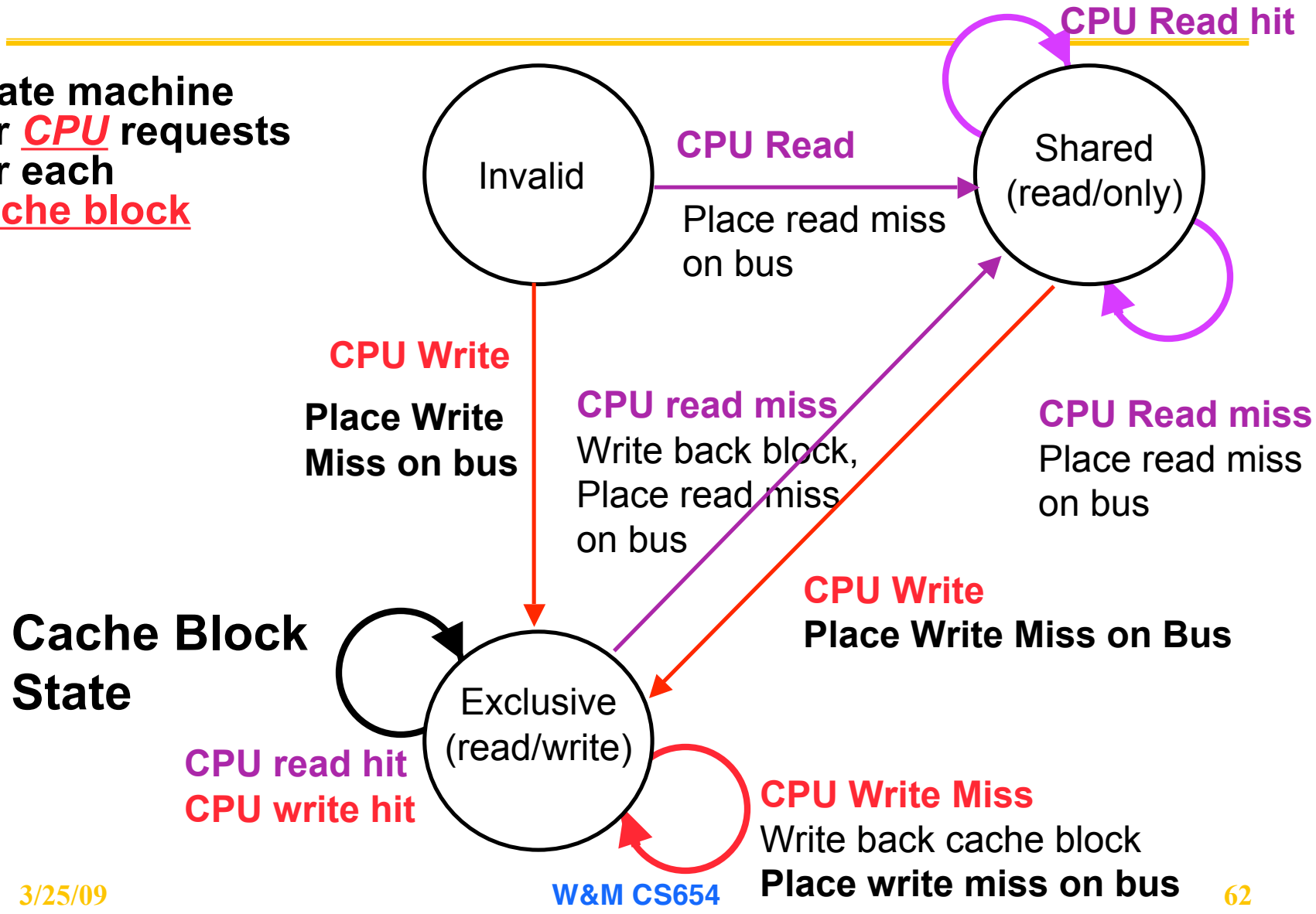
Write-Back State Machine- Bus request

- **State machine for *bus* requests for each cache block**



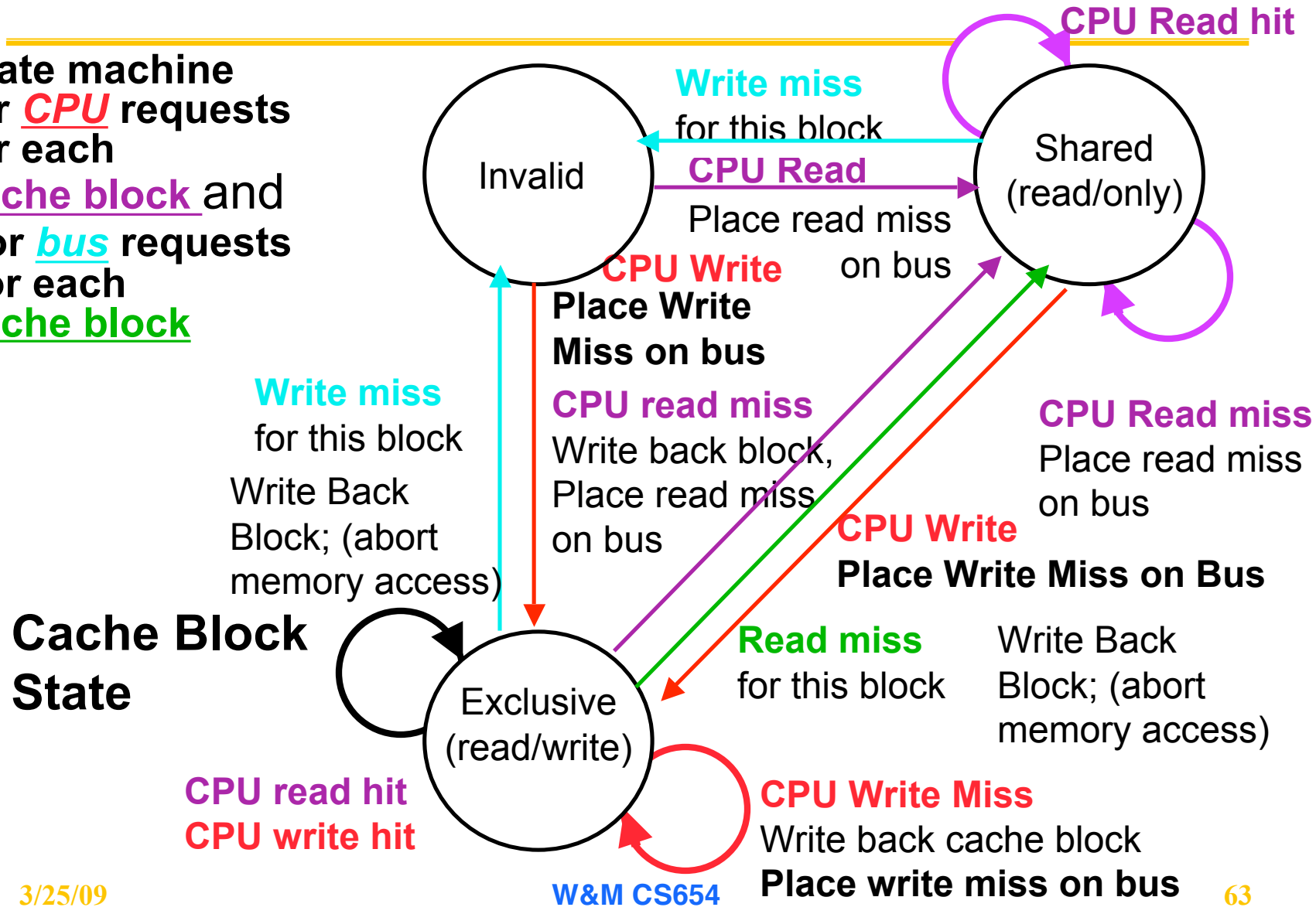
Block-replacement

- State machine for *CPU* requests for each cache block



Write-back State Machine-III

- State machine for *CPU* requests for each cache block and for *bus* requests for each cache block



Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1												
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block, initial cache state is invalid

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1												
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2												

Assumes A1 and A2 map to same cache block

Example

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1 Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1			
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2							WrMs	P2	A2		A1	10
				Excl.	A2	40	WrBk	P2	A1	20	A1	20

Assumes A1 and A2 map to same cache block,
but A1 != A2

And in Conclusion [1/2] …

- 1 instruction operates on vectors of data
- Vector loads get data from memory into big register files, operate, and then vector store
- E.g., Indexed load, store for sparse matrix
- Easy to add vector to commodity instruction set
  - E.g., Morph SIMD into vector
- Vector is very efficient architecture for vectorizable codes, including multimedia and many scientific codes

And in Conclusion [2/2] …

- "End" of uniprocessors speedup => Multiprocessors
- Parallelism challenges: % parallelizable, long latency to remote memory
- Centralized vs. distributed memory
  - Small MP vs. lower latency, larger BW for Larger MP
- Message Passing vs. Shared Address
  - Uniform access time vs. Non-uniform access time
- Snooping cache over shared medium for smaller MP by invalidating other cached copies on write
- Sharing cached data ⇒ Coherence (values returned by a read), Consistency (when a written value will be returned by a read)
- Shared medium serializes writes
  ⇒ Write consistency