16.482 / 16.561 Computer Architecture and Design

Instructor: Dr. Michael Geiger

Spring 2014

Lecture 10:

Storage

Multiprocessors

Lecture outline

- Announcements/reminders
  - HW 7 due today
  - HW 8 due Monday 4/28 by 5:00 PM
    - Extra credit assignment—replaces lowest grade for HW 1-7
  - Final exam: Thursday 5/1
    - Must complete course eval and bring to exam (will post online)
- Review
  - Virtual memory
  - Cache optimizations
- Today's lecture
  - Storage
  - Multiprocessors
  - Final Exam Preview

Reviewing optimizations

- Way prediction
  - Want benefits of
    - Direct-mapped: fast hits
    - Set-associative: fewer conflicts
  - Predict which way within set holds data
- Trace cache
  - Intel-specific
  - Track "traces" of decoded micro-ops
- Multi-banked caches
  - Cache physically split into pieces ("banks")
  - Data sequentially interleaved
    - Spread accesses across banks
  - Allows for cache pipelining, non-blocking caches

Reviewing optimizations

- Critical word first / early restart
  - Fetch desired word in cache block first on miss
  - Restart processor as soon as desired word received
- Merging write buffers
  - Eliminate multiple copies of block in write buffer

Case for Storage

- Shift in focus from computation to communication and storage of information
  - E.g., Cray Research/Thinking Machines vs. Google/Yahoo
  - "The Computing Revolution" (1960s to 1980s)
    $\Rightarrow$ "The Information Age" (1990 to today)
- Storage emphasizes reliability and scalability as well as cost-performance

Disk Size and Performance

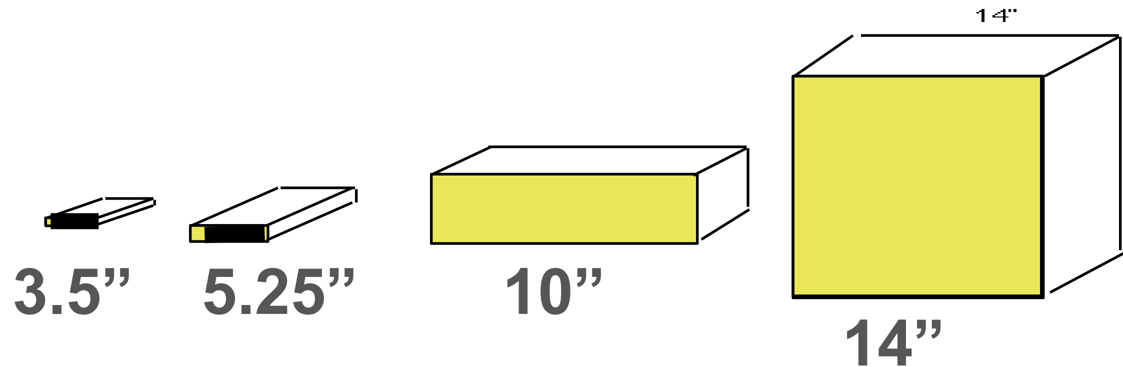
- Continued advance in capacity (60%/yr) and bandwidth (40%/yr)
- Slow improvement in seek, rotation (8%/yr)
- Time to read whole disk

Year	Sequentially	Randomly (1 sector/seek)
1990	4 minutes	6 hours
2000	12 minutes	1 week(!)
2006	56 minutes	3 weeks (SCSI)
2006	171 minutes	7 weeks (SATA)

Use Arrays of Small Disks?

- **Katz and Patterson asked in 1987:**
  - **Can smaller disks be used to close gap in performance between disks and CPUs?**

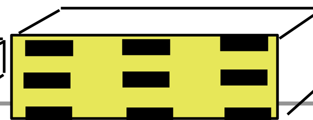
Conventional: 4 disk designs

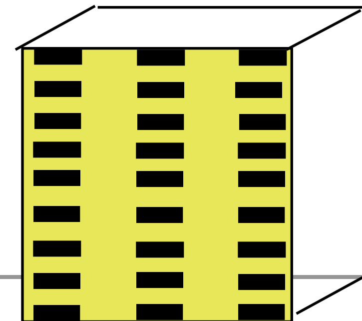


Low End **High End**

Disk Array: 1 disk design

3.5"





Array Reliability

- **Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, but what about reliability?**

- **Reliability of N disks = Reliability of 1 Disk ÷ N**

  50,000 Hours ÷ 70 disks = 700 hours

  Disk system MTTF: Drops from 6 years to 1 month!

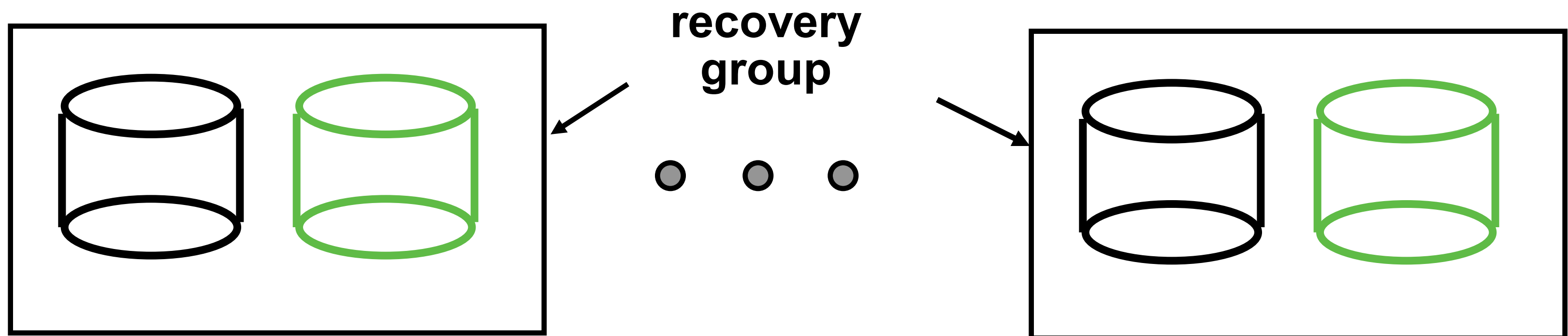
- **Arrays (without redundancy) too unreliable to be useful!**

Hot spares support reconstruction in parallel with access: very high media availability can be achieved

Redundant Arrays of (Inexpensive) Disks

- Files are "striped" across multiple disks
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - Capacity penalty to store redundant info
  - Bandwidth penalty to update redundant info

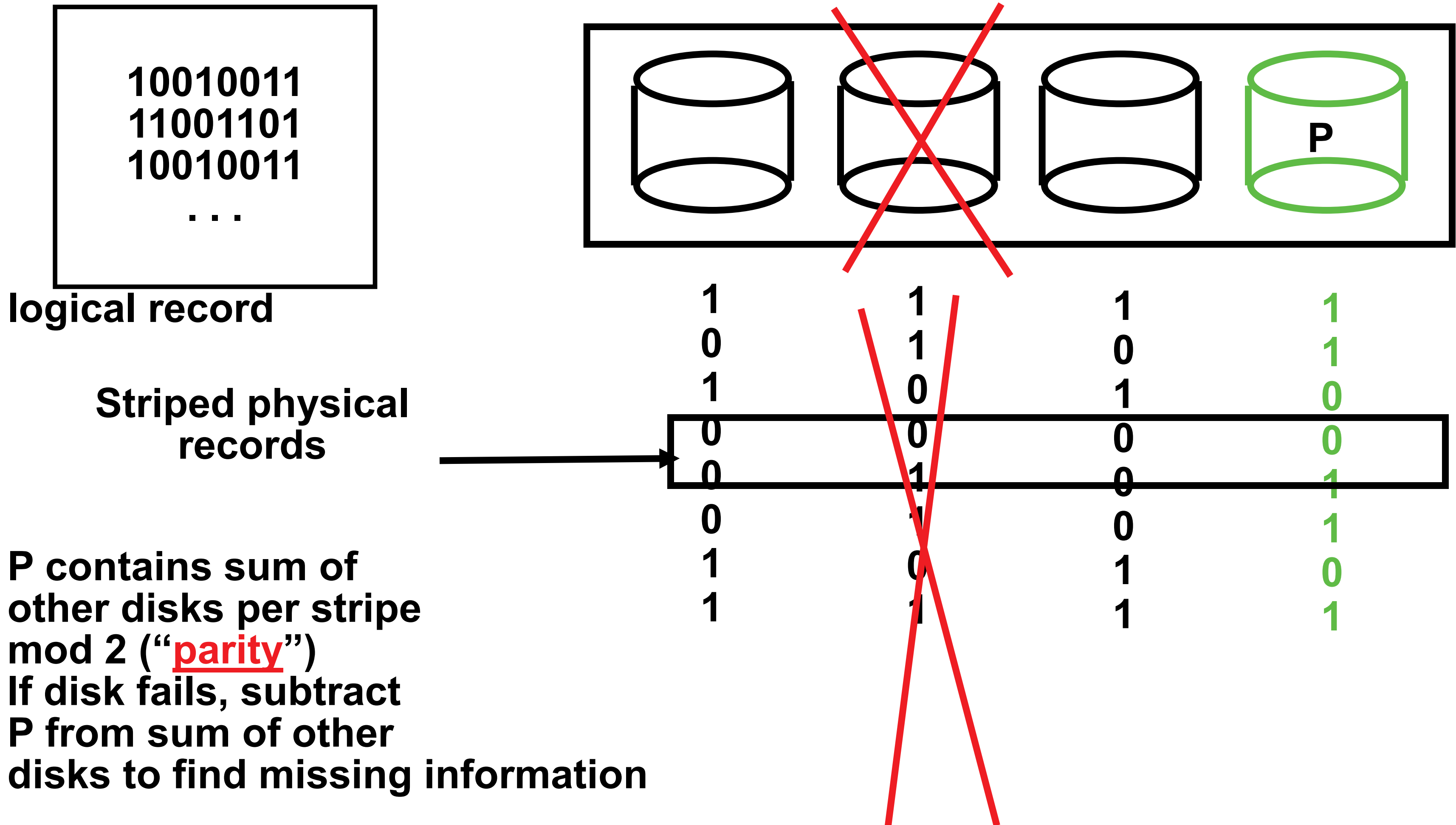
RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its "mirror"
  Very high availability can be achieved
- Bandwidth sacrifice on write:
  Logical write = two physical writes
  - Reads may be optimized
- Most expensive solution: 100% capacity overhead

- (RAID 2 not interesting, so skip)

RAID 3: Parity Disk



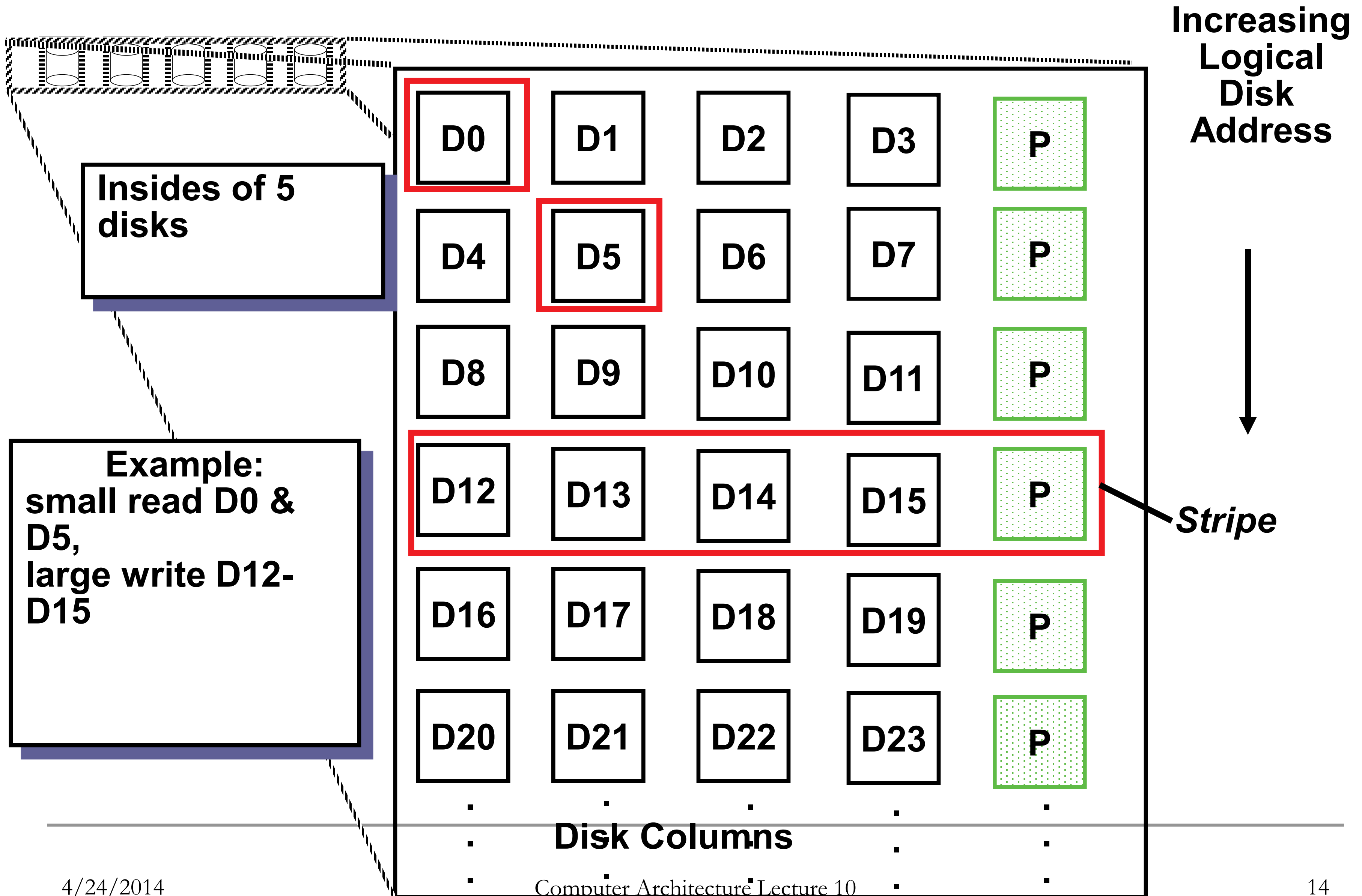
RAID 3

- Sum computed across recovery group to protect against hard disk failures, stored in P disk
- Logically, a single high capacity, high transfer rate disk: good for large transfers
- Wider arrays reduce capacity costs, but decreases availability
- 33% capacity cost for parity if 3 data disks and 1 parity disk

Inspiration for RAID 4

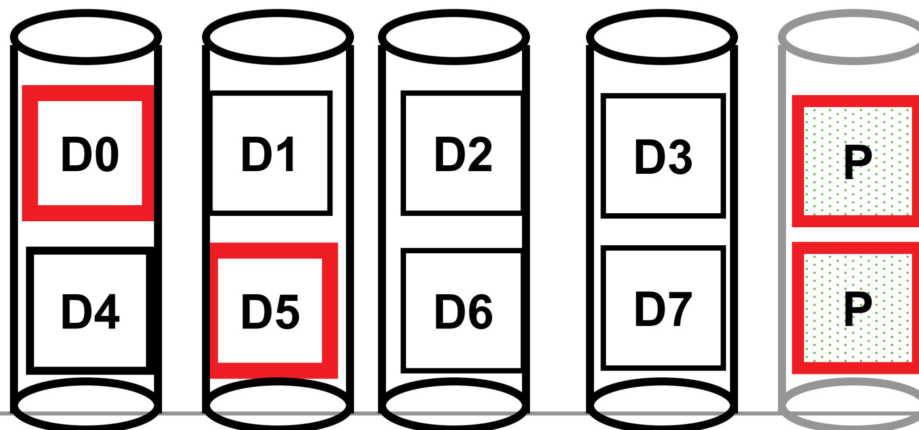
- RAID 3 relies on parity disk to discover errors on Read
- But every sector has an error detection field
- To catch errors on read, rely on error detection field vs. the parity disk
- Allows independent reads to different disks simultaneously

RAID 4: High I/O Rate Parity

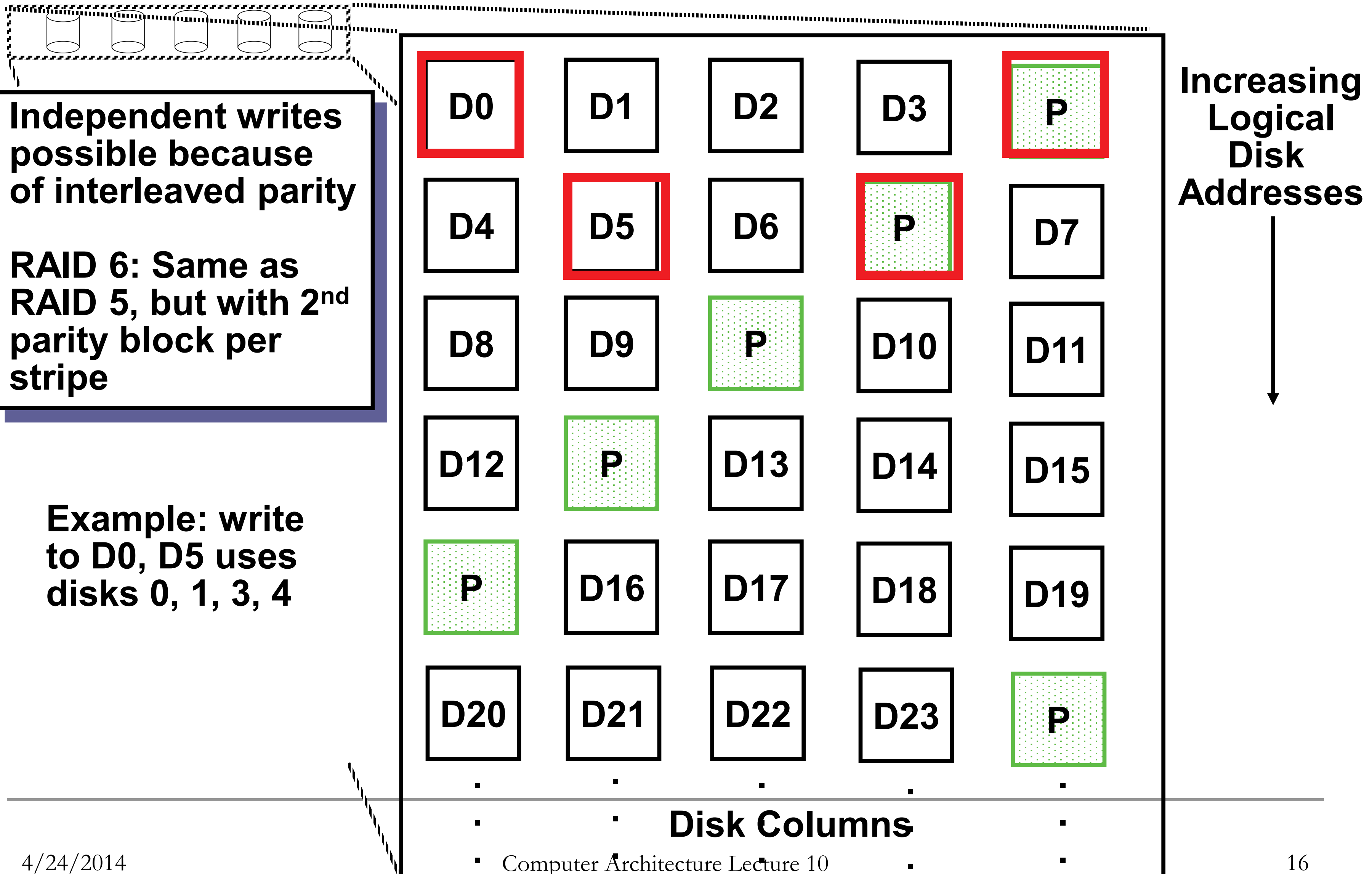


Inspiration for RAID 5

- RAID 4 works well for small reads
- Small writes (write to one disk):
  - Option 1: read other data disks, create new sum and write to Parity Disk
  - Option 2: since P has old sum, compare old data to new data, add the difference to P
- Small writes are limited by Parity Disk: Write to D0, D5 both also write to P disk



RAID 5: High I/O Rate Interleaved Parity



Flynn's Taxonomy

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

- Flynn classified by data and control streams in 1966
- SIMD $\Rightarrow$ Data Level Parallelism
- MIMD $\Rightarrow$ Thread Level Parallelism
- MIMD popular because
  - Flexible: N pgms and 1 multithreaded pgm
  - Cost-effective: same MPU in desktop & MIMD

Classifying Multiprocessors

- Parallel Architecture = Computer Architecture + Communication Architecture
- Classifying by memory:
  - Centralized Memory (Symmetric) Multiprocessor
    - Typically for smaller systems → bandwidth demands on both network and memory system
  - Physically Distributed-Memory multiprocessor
    - Scales better → bandwidth demands (mostly) for network
- Classifying by communication:
  - Message-Passing Multiprocessor: processors explicitly pass messages
  - Shared Memory Multiprocessor: processors communicate through shared address space
    - If using centralized memory, UMA (Uniform Memory Access time)
    - If distributed memory, NUMA (Non Uniform Memory Access time)

Coherence and consistency

- Issues with caching shared data
  - Coherence: How are accesses from multiple processors to the same location viewed?
    - P1 and P2 read A, then P1 writes A
    - Must ensure P2 either gets up-to-date value or has copy invalidated
  - Consistency: How are accesses from multiple processors to different locations viewed?
    - Enforcing ordering—if P1 sees A written before B, all processors should see A written before B
    - Will address later

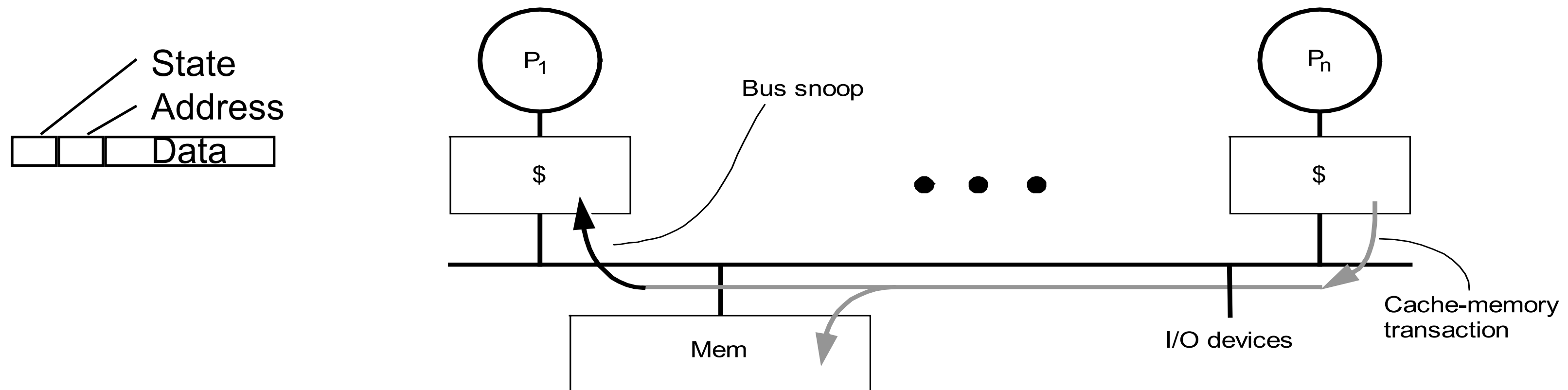
Enforcing coherence

- Preserve program order
  - Processor should see value it writes to variable if no other processors write same variable
- Coherent view of memory
  - If P1 writes variable, and there is enough time between P1 write and P2 read, P2 should read the value written by P1
- Write serialization
  - Multiple writes to same location are seen in same order by all processors

Cache Coherence Protocols

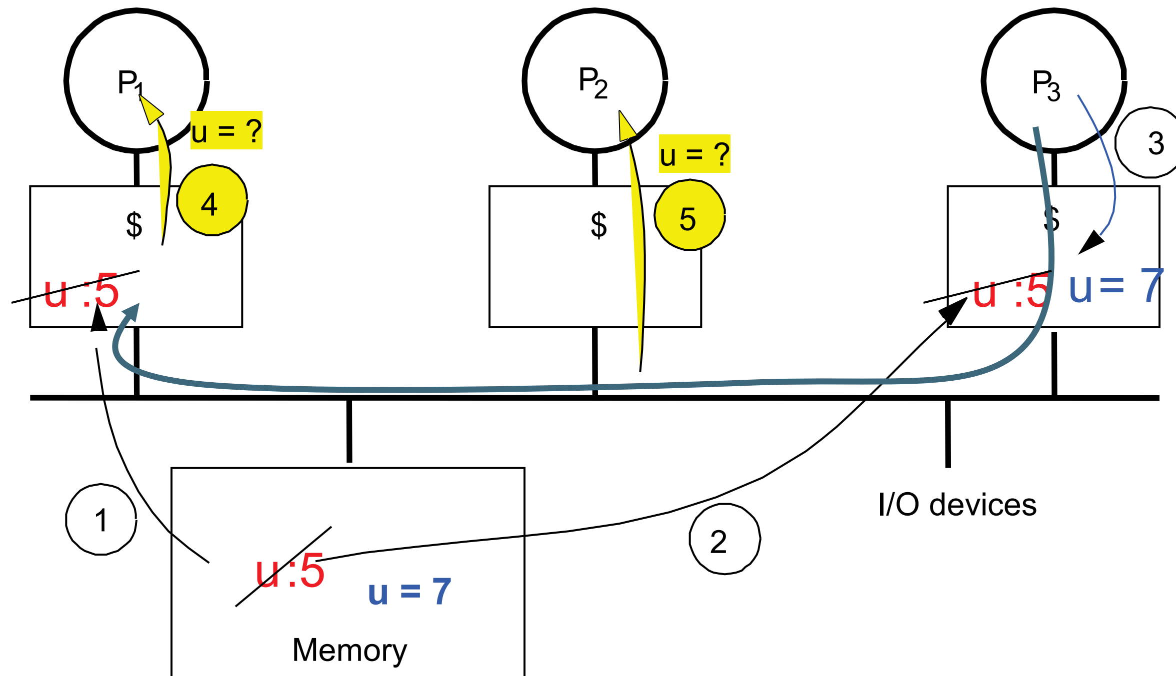
- Two types of protocol to track sharing status
  - Directory based — One centralized structure holding all sharing information
  - Snooping — Every cache that has a copy of data has a copy of its sharing state as well
- Two different ways to handle writes in snooping protocols
  - Write update—Broadcast value written for every write
  - Write invalidate—Invalidate all other shared copies on a write

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 (write update)

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

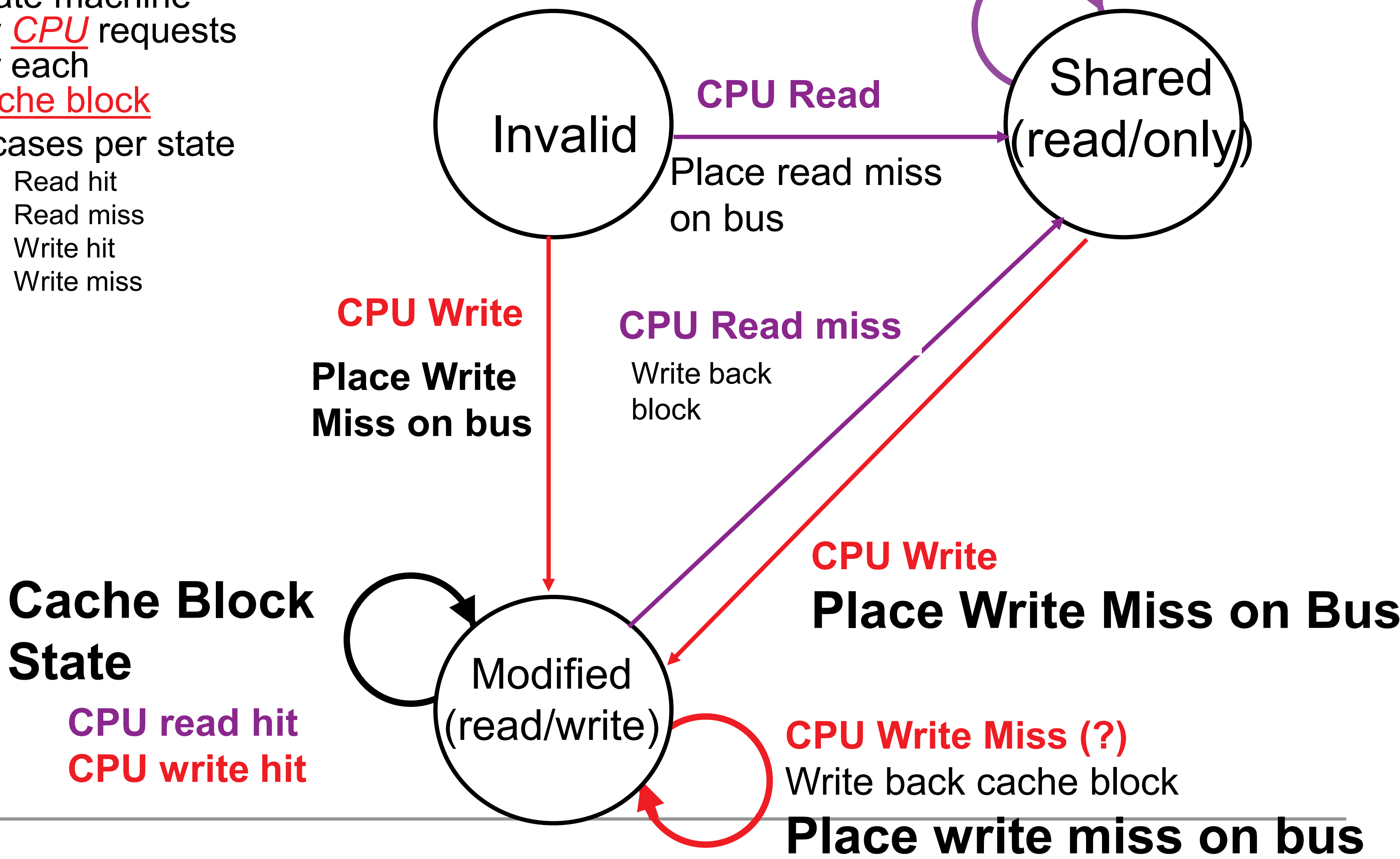
- What do we need to implement a cache coherence protocol?
  - Cache block state transition diagram
    - "States" encoded in valid/dirty bits
  - Broadcast Medium Transactions (e.g., bus)
  - Broadcast medium enforces serialization of read or write accesses $\Rightarrow$ Write serialization
  - Ability to find up-to-date copy of cache block
    - Easy for write-through, harder for write-back
    - Write-back can use snooping mechanism
      - If processor has dirty copy of block, can respond
      - May actually take longer to read data from cache than from memory—you have to work your way "up" the memory hierarchy!

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 Modified : 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
- 4 cases per state
  - Read hit
  - Read miss
  - Write hit
  - Write miss



Write-Back State Machine- Bus request

- State machine for *bus* requests for each cache block
  - Write miss/read miss

Invalid

Shared (read/only)

Modified (read/write)

Write miss for this block (Shared → Invalid)

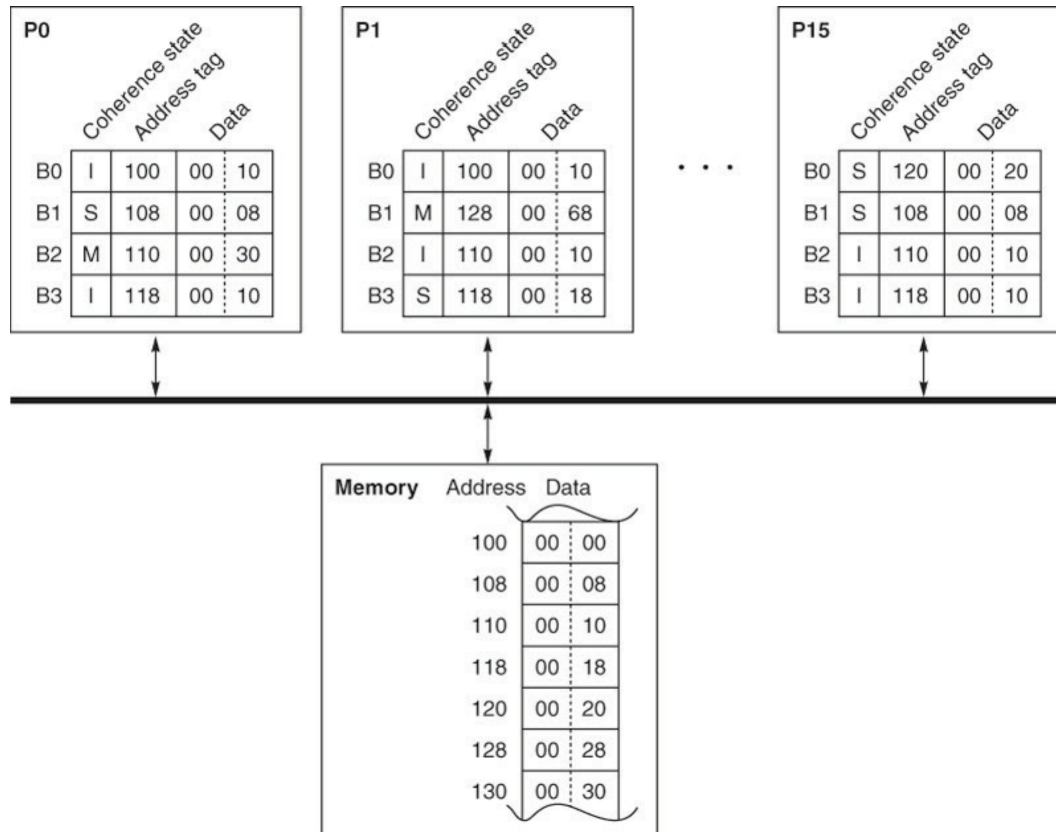
Write miss for this block (Modified → Invalid)
Write Back Block; (abort memory access)

Read miss for this block (Modified → Shared)
Write Back Block; (abort memory access)

Snooping protocol example

- Assume we have the system shown on the next slide
  - Each cache is direct-mapped, and each block contains two 32-bit words of memory—only the least significant byte of each word is shown.
  - Addresses are written in hexadecimal. In this figure, the address tag shows the full address for simplicity.
  - The coherence states are M (Modified—both exclusive and dirty), S (Shared), and I (Invalid). The system uses write-back caches and a write-invalidate protocol.

Snooping protocol example (cont.)



© 2007 Elsevier, Inc. All rights reserved.

Snooping protocol example (cont.)

- Each operation below is independently performed on the system using the initial state above. For each part, answer the following questions:
  - What is the resulting state—coherence state, tags, and data—for the caches and memory after each given action?
    - Show only the blocks that change, using the following format:
      `P#.B#: (<state>, <tag>, <data>)`
    - If an access changes memory, list the state of the changed memory block using the format:
      `M[<address>]: <data>`
    - Remember, a single access may affect multiple blocks, depending on their state.
  - If the operation is a read, what value is returned?

a. P1: read 108
b. P0: write 108 ← 83
c. P1: write 108 ← 83
d. P15: read 118
e. P0: write 110 ← 52
f. P1: write 118 ← 99
g. P0: write 118 ← 99

Snooping protocol example solution

- P1: read 108
  - This block is in the shared state in two other caches, neither of which is affected. However, 108 maps to the same cache line as 128, a block which is in the modified state in P1's cache, so that block must be written back to memory before 108 can be retrieved:
  - P1.B1: (S, 108, 00 08)
  - M[128]: 00 68
  - The read returns 00.
- P0: write 108 ← 83
  - P0 and P15 hold this block in the shared state, so this write will invalidate P15's copy and switch P0's copy to modified:
  - P0.B1: (M, 108, 83 08)
  - P15.B1: (I, 108, 00 08)
- P1: write 108 ← 83
  - P1 is writing a block that both P0 and P15 have in the shared state—both of those copies must be invalidated. As in part (a), this access will evict 128 from P1's cache, causing a write back and update of memory.
  - P0.B1: (I, 108, 00 08)
  - P1.B1: (M, 108, 83 08)
  - P15.B1: (I, 108, 00 08)
  - M[128]: 00 68
- P15: read 118
  - P15 reads a block not present in its cache; the block is in the shared state elsewhere.
  - P15.B3: (S, 118, 00 18)
  - The read returns 00.

Snooping protocol example soln (cont.)

- P0: write 110 ← 52
  - P0 writes a block it already holds in the modified state—no other caches are affected.
  - P0.B2: (M, 110, 52 30)
- P1: write 118 ← 99
  - P1 holds this block in the shared state; no other caches have a copy, so the write only affects P1.
  - P1.B3: (M, 118, 99 18)
- P0: write 118 ← 99
  - The only copy of this block is in P1. It must be invalidated, while P0 now has a modified copy.
  - P0.B3: (M, 118, 99 18)
  - P1.B3: (I, 118, 00 18)

Sharing misses

- Cache performance is combination of
  - Uniprocessor cache miss traffic
  - Traffic caused by communication
    - Results in invalidations and subsequent cache misses
- 4th C: *coherence miss*
  - Joins Compulsory, Capacity, Conflict
  - Two types of coherence misses, both related to sharing
    - True sharing miss: word is written by one processor, thus invalidating block, and then accessed by a different processor
    - False sharing miss: word is written by one processor, thus invalidating block, and then different word in that same block accessed by a different processor
  - Increased sharing → increase in coherence misses

A Cache Coherent System Must:

- Provide set of states, state transition diagram, and actions
- Manage coherence protocol
  - (0) Determine when to invoke coherence protocol
  - (a) Find info about state of block in other caches to determine action
    - whether need to communicate with other cached copies
  - (b) Locate the other copies
  - (c) Communicate with those copies (invalidate/update)
- (0) is done the same way on all systems
  - state of the line is maintained in the cache
  - protocol is invoked if an "access fault" occurs on the line
- Different approaches distinguished by (a) to (c)

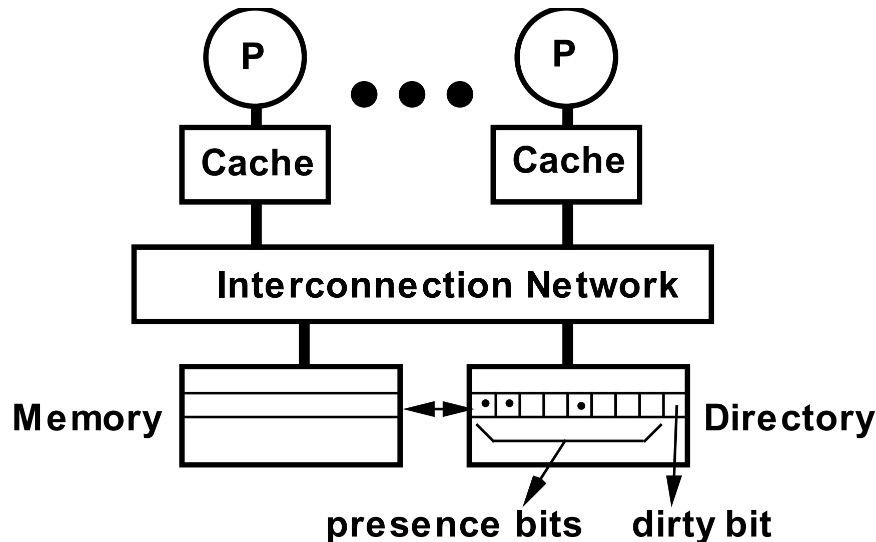
Bus-based Coherence

- All of (a), (b), (c) done through broadcast on bus
  - faulting processor sends out a “search”
  - others respond to the search probe and take necessary action
- Could do it in scalable network too
  - broadcast to all processors, and let them respond
- Conceptually simple, but broadcast doesn’t scale with p
  - on bus, bus bandwidth doesn’t scale
  - on scalable network, every fault leads to at least p network transactions
- Scalable coherence:
  - can have same cache states and state transition diagram
  - different mechanisms to manage protocol

Scalable Approach: 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
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

Basic Operation of Directory



- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

- Read from main memory by processor i:
  - If dirty-bit OFF then { read from main memory; turn p[i] ON; }
  - if dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}
- Write to main memory by processor i:
  - If dirty-bit OFF then { supply data to i; send invalidations to all caches that have the block; turn dirty-bit ON; turn p[i] ON; ... }
  - ...

Directory Protocol

- Similar to Snoopy Protocol: Three states
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached: no processor has it; not valid in any cache
  - Exclusive: 1 processor (owner) has data; memory out-of-date
- In addition to cache state, must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)
- Keep it simple(r):
  - Writes to non-exclusive data => write miss
  - Processor blocks until access completes
  - Assume messages received and acted upon in order sent

Directory Protocol

- No bus and don’t want to broadcast:
  - Interconnect no longer single arbitration point
  - All messages have explicit responses
- Terms: typically 3 processors involved
  - Local node where a request originates
  - Home node where the memory location of an address resides
  - Remote node has a copy of a cache block, whether exclusive or shared
- Example messages on next slide: P = processor number, A = address

Directory Protocol Messages (Fig 4.22)

Message type	Source	Destination	Msg Content
Read miss	Local cache	Home directory	P, A

- *Processor P reads data at address A; make P a read sharer and request data*

Message type	Source	Destination	Msg Content
Write miss	Local cache	Home directory	P, A

- *Processor P has a write miss at address A; make P the exclusive owner and request data*

Message type	Source	Destination	Msg Content
Invalidate	Home directory	Remote caches	A

- *Invalidate a shared copy at address A*

Message type	Source	Destination	Msg Content
Fetch	Home directory	Remote cache	A

- *Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared*

Message type	Source	Destination	Msg Content
Fetch/Invalidate	Home directory	Remote cache	A

- *Fetch the block at address A and send it to its home directory; invalidate the block in the cache*

Message type	Source	Destination	Msg Content
Data value reply	Home directory	Local cache	Data

- *Return a data value from the home memory (read miss response)*

Message type	Source	Destination	Msg Content
Data write back	Remote cache	Home directory	A, Data

- *Write back a data value for address A (invalidate response)*

State Transition Diagram

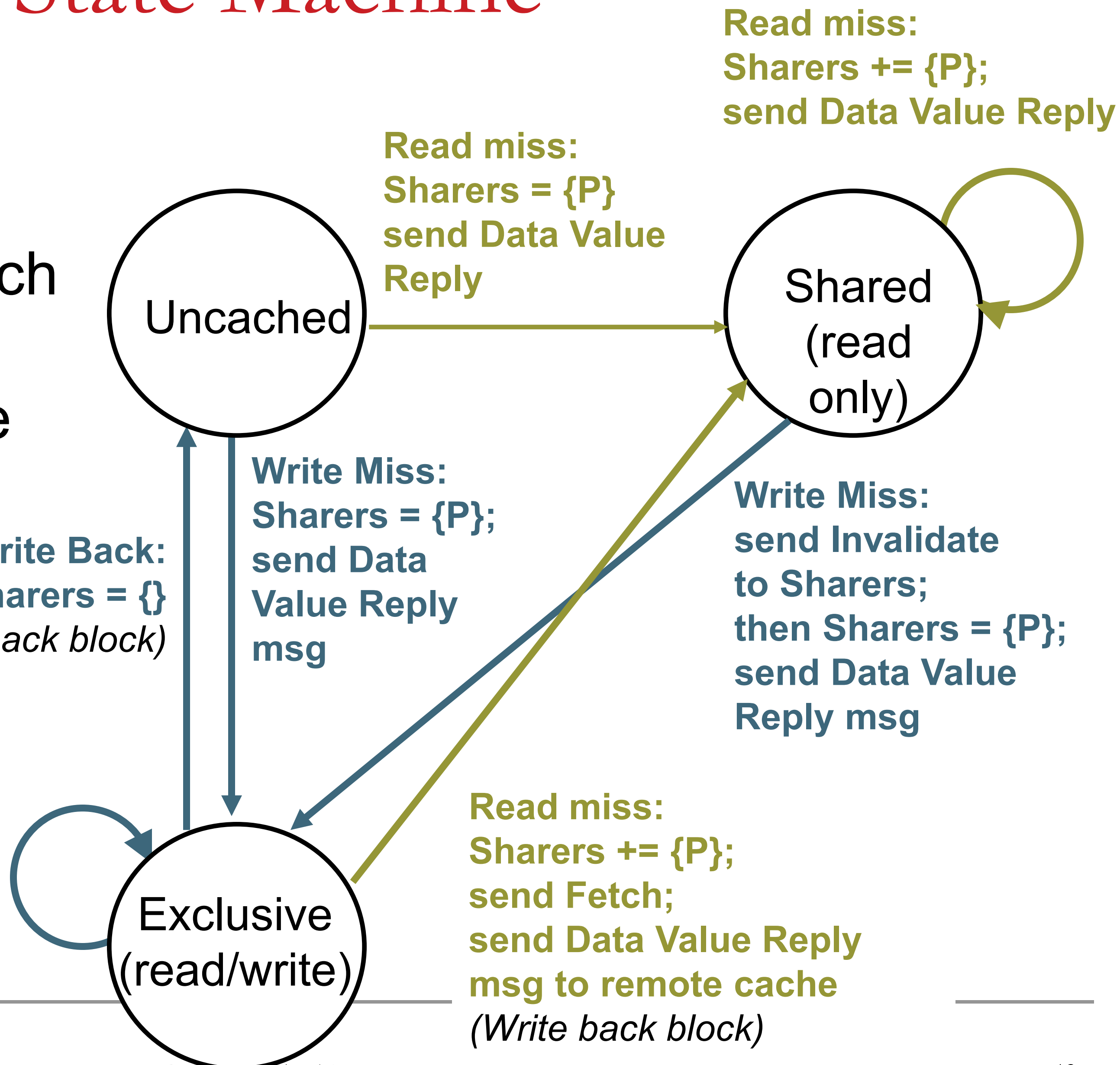
- For single cache block in directory-based system:
  - States identical to snoopy case; transactions very similar.
  - Transitions caused by read misses, write misses, invalidates, data fetch requests
  - Generates read miss & write miss msg to home directory.
  - Write misses that were broadcast on the bus for snooping => explicit invalidate & data fetch requests.
  - Note: on a write, a cache block is bigger, so need to read the full cache block

State Transition Diagram for Directory

- Similar states & structure as the transition diagram for an individual cache
- 2 actions: update of directory state & send messages to satisfy requests
- Tracks all copies of memory block
- Also indicates an action that updates the sharing set, Sharers, as well as sending a message

Directory State Machine

- State machine for Directory requests for each memory block
- Uncached state if in memory



Directory Protocol

- Message sent to directory causes two actions:
  - Update the directory
  - More messages to satisfy request
- Block is in Uncached state: the copy in memory is the current value; only possible requests for that block are:
  - Read miss: requesting processor sent data from memory & requestor made only sharing node; state of block made Shared.
  - Write miss: requesting processor is sent the value & becomes the Sharing node. The block is made Exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.
- Block is Shared => the memory value is up-to-date:
  - Read miss: requesting processor is sent back the data from memory & requesting processor is added to the sharing set.
  - Write miss: requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, & Sharers is set to identity of requesting processor. The state of the block is made Exclusive.

Directory Protocol (cont.)

- Block is Exclusive: current value of the block is held in the cache of the processor identified by the set Sharers (the owner) => three possible directory requests:
  - Read miss: owner processor sent data fetch message, causing state of block in owner's cache to transition to Shared and causes owner to send data to directory, where it is written to memory & sent back to requesting processor.
    Identity of requesting processor is added to set Sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy). State is shared.
  - Data write-back: owner processor is replacing the block and hence must write it back, making memory copy up-to-date
    (the home directory essentially becomes the owner), the block is now Uncached, and the Sharer set is empty.
  - Write miss: block has a new owner. A message is sent to old owner causing the cache to send the value of the block to the directory from which it is sent to the requesting processor, which becomes the new owner. Sharers is set to identity of new owner, and state of block is made Exclusive.

Directory protocol example

- Assume
  - Dual-processor system
  - Write-invalidate directory coherence protocol
  - Initial directory state:

Block #	P0	P1	Dirty
0	1	0	1
1	0	0	0
2	1	1	0
3	0	1	0

Directory protocol example (cont.)

- What messages are sent between nodes and directory for each transactions to ensure
  - Processor has the most up-to-date block copy
  - Appropriate invalidations are made
  - Directory holds the appropriate state
- What is the final sharing state?
- Assume answers are independent: B doesn't depend on A, for example

A. P0: read block 1
B. P1: write block 2
C. P1: read block 0
D. P0: read block 3

Directory protocol example: solution

- P0: read block 1
  - Read to uncached block: must request data
  - Messages
    - ReadMiss(P0,1)
    - DataValueReply(mem[1]) sent to P0
  - Final state: shared, with P0 as only sharer
- P1: write block 2
  - Write to shared block: write miss sent out to invalidate other copy
  - Messages
    - WriteMiss(P1,2)
    - Invalidate(2) sent to P0
    - DataValueReply(mem[2]) sent to P1
      - Indicates invalidation complete
  - Final state: exclusive, with P1 as only sharer

Directory protocol example: soln (cont.)

- P1: read block 0
  - Read to exclusive block: must get most up-to-date value
  - Messages
    - ReadMiss(P1,0)
    - Fetch(0) sent to P0
    - DataWriteBack(0, mem[0]) sent from P0 to directory
    - DataValueReply(mem[0] sent to P1
  - Final state: shared, with P0 and P1 both sharing
- P0: read block 3
  - Read to shared block: just need up-to-date value
  - Messages
    - ReadMiss(P0,3)
    - DataValueReply(mem[3]) sent to P0
  - Final state: shared, with P0 and P1 both sharing

Final exam notes

- Allowed to bring:
  - Two 8.5” x 11” double-sided sheets of notes
  - Calculator
- No other notes or electronic devices
- You can’t share anything—erasers, pencils, etc.
  - I’ll bring extra erasers/pencils for those who may need them
- Exam will last until 9:30
  - Will be written for ~90 minutes ... in theory
- Covers all lectures after Exam 1
  - Material starts with multiple issue/multithreading
- Question formats
  - Problem solving
    - Largely similar to homework problems, but shorter
  - Some short answer
    - Explain concepts in short paragraph

Review: TLP, multithreading

- ILP (which is implicit) limited on realistic hardware for single thread of execution
- Could look at explicit parallelism: Thread-Level Parallelism (TLP) or Data-Level Parallelism (DLP)
- Focus on TLP and multithreading
  - Coarse-grained: switch threads on a stall
  - Fine-grained: switch threads every cycle
  - Simultaneous: allow multiple threads to execute in same cycle

Review: Memory hierarchies

- Want large, fast, low-cost memory; can't have it all at once
- Use multiple levels: cache (may have >1), main memory, disk
- Discussed operation of hierarchy, focusing heavily on:
  - Caching: principle of locality, addressing cache, "4 questions" (block placement, identification, replacement, write strategy)
  - Virtual memory: advantages (translation, protection, sharing), page tables & address translation, page replacement, TLB

Review: cache optimizations

- Way prediction
  - Want benefits of
    - Direct-mapped: fast hits
    - Set-associative: fewer conflicts
  - Predict which way within set holds data
- Multi-banked caches
  - Cache physically split into pieces (“banks”)
  - Data sequentially interleaved
    - Spread accesses across banks
  - Allows for cache pipelining, non-blocking caches

Review: cache optimizations

- Critical word first / early restart
  - Fetch desired word in cache block first on miss
  - Restart processor as soon as desired word received

Review: Storage

- Disks → RAID
  - Build redundancy (parity) into disk array because MTTFarray = MTTFdisk / # disks
  - Files "striped" across disks along with parity
  - Different levels of RAID offer improvements
    - RAID 1: Mirroring
    - RAID 3: Introduced parity disk
    - RAID 4: Use per-sector error correction → allows small reads
    - RAID 5: Interleaved parity → allows small writes

Review: Multiprocessors

- Multiprocessor caches hold both private and shared data; must provide both:
  - Migration: Data can be moved to local cache and used transparently
  - Replication: Shared data can be used in multiple caches
- Cache coherence
  - Guarantee any read returns the most recently written value
  - Coherent system
    - Preserves program order
    - Maintains coherent view of memory
    - Serializes writes
      - Write consistency: write completes when all processors can read it, and all writes across all processors are serialized

Review: Cache coherence protocols

- Need to maintain coherence; two types of protocols for doing so
  - Directory based: One (logically) centralized structure holds sharing information
    - One entry per memory block
  - Snooping: Each cache holding a copy of data has copy of its sharing information
    - One entry per cache block
    - Caches "snoop" bus and respond to relevant transactions
  - You should have an understanding of both types of protocols
- Two ways to handle writes
  - Write invalidate: Ensure that cache has exclusive access to block before performing write
    - All other copies are invalidated
  - Write update (or write broadcast): Update all cached copies of a block whenever block is written

Review: More cache coherence

- Should be familiar with coherence protocol state transition diagrams
  - What happens on different CPU transactions? (Read/Write, Hit/Miss)
  - For snooping protocol, what happens for relevant transactions sent over bus? (Read or write miss)
    - In snooping protocol, all state transitions occur for cache block, since cache block tracks sharing info
  - For directory protocol, what happens for each message sent to given processor *or* directory? (Read/Write Miss, Invalidate and/or Fetch, Data Reply)
    - State transitions occur in cache blocks and directory entries (directory holds sharing info)
- Additional cache misses: coherence misses
  - True sharing misses: one processor writes to same location that's later accessed by another processor
  - False sharing misses: one processor writes to same block (but different location) later accessed by another processor

Final notes

- Next time: Final Exam
- Reminders
  - HW 7 due today
  - HW 8 due Monday 4/28 by 5:00 PM
    - Extra credit assignment—replaces lowest grade for HW 1-7
  - Final exam: Thursday 5/1
    - Must complete course eval and bring to exam (will post online)