Topic V37

Memory Hierarchy Concepts

Readings: (Section 5.1, 5.3, 5.4, 5.6)

The Memory Hierarchy

Common principles apply at all levels of the memory hierarchy

- Based on notions of caching

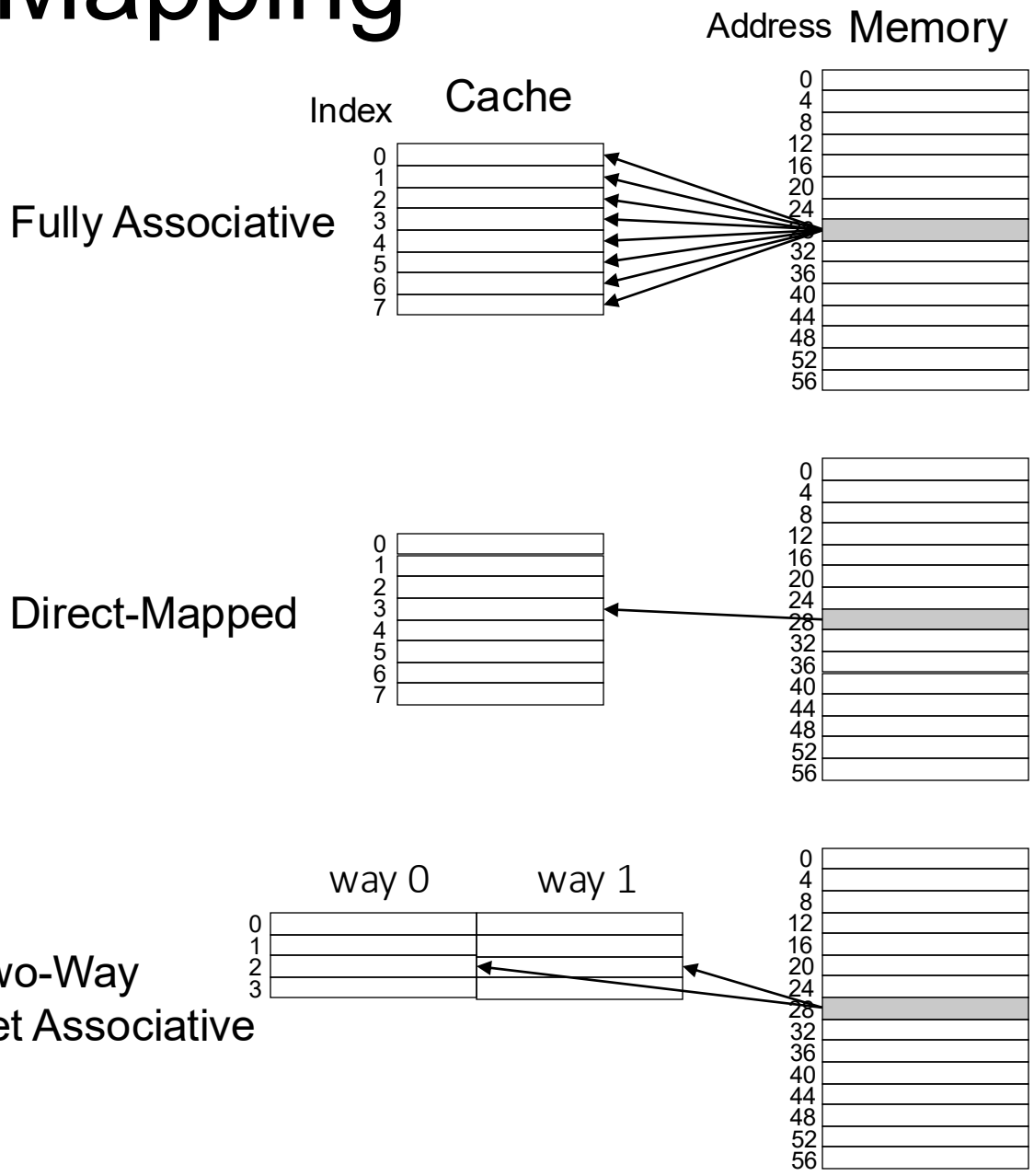
At each level in the hierarchy

- Block placement
- Finding a block
- Replacement on a miss
- Write policy

Block Placement

Determined by associativity

Mapping


Address
Memory
Index
Cache
Fully Associative
Direct-Mapped
way 0
way 1
Two-Way
Set Associative

Block Placement

- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1

Hardware caches

Reduce comparisons to reduce cost

Virtual memory

Full table lookup makes full associativity feasible

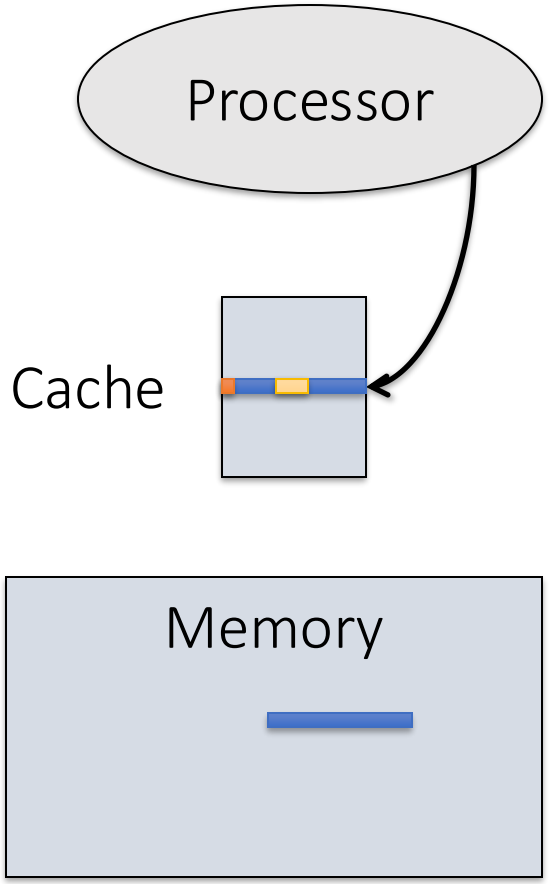
Benefit in reduced miss rate

Replacement

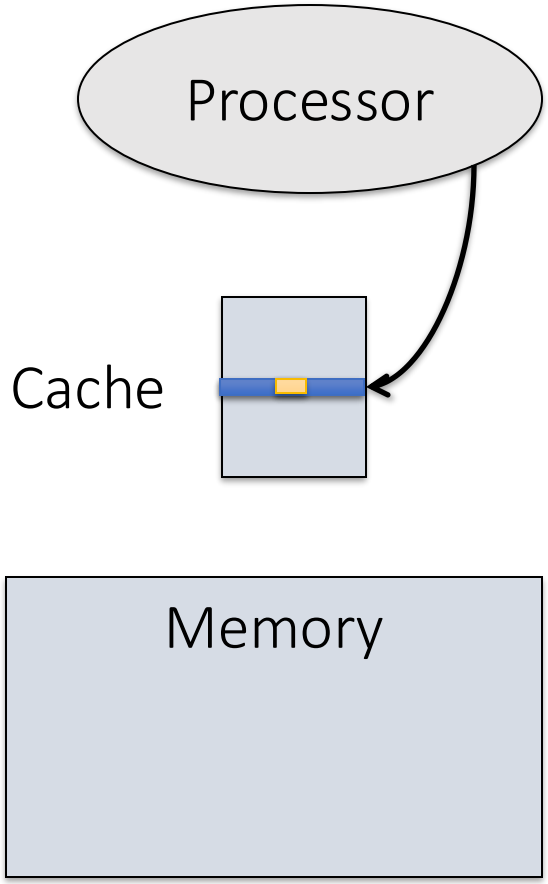
- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Not most-recently used with Random selection
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

Write Policy

Write Back



Write Through



Virtual Memory Write Policy

Only write-back policy is feasible, given disk write latency

Source of Misses

Compulsory misses (aka cold start misses)

- First access to a block

Capacity misses

- Due to finite cache size
- A replaced block is later accessed again

Conflict misses (aka collision misses)

- In a non-fully associative cache
- Due to competition for entries in a set
- Would not occur in a fully associative cache of the same total size

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect

Virtual Machines

Host computer emulates guest operating system and machine resources

- Improved isolation of multiple guests
- Avoids security and reliability problems
- Aids sharing of resources

Virtualization has some performance impact

- Feasible with modern high-performance computers

Examples

- IBM VM/370 (1970s technology!)
- VMWare
- Microsoft Virtual PC

Virtual Machine Monitor (VMM)

Maps virtual resources to physical resources

- Memory, I/O devices, CPUs

Guest code runs on native machine in user mode

- Traps to VMM on privileged instructions and access to protected resources

Guest OS may be different from host OS

VMM handles real I/O devices

- Emulates generic virtual I/O devices for guest

Timer Virtualization (example)

In native machine, on timer interrupt

- OS suspends current process, handles interrupt, selects and resumes next process

With Virtual Machine Monitor

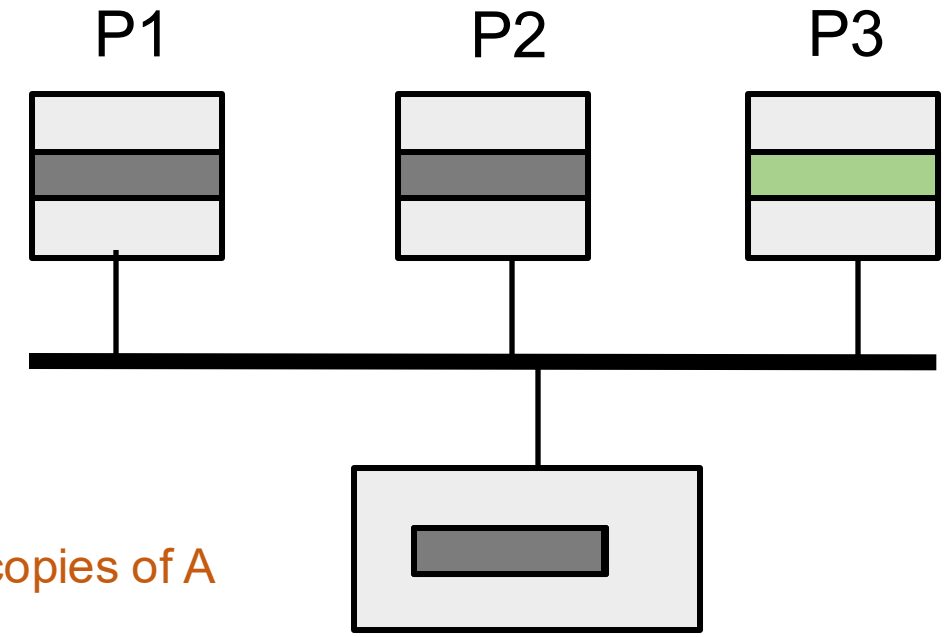
- VMM suspends current VM, handles interrupt, selects and resumes next VM

If a VM requires timer interrupts

- VMM emulates a virtual timer
- Emulates interrupt for VM when physical timer interrupt occurs

Cache Coherence

Example



1. P1 reads some data A
2. P2 reads same data A

   // P1, P2 and memory have valid copies of A
3. P3 has a write miss on A

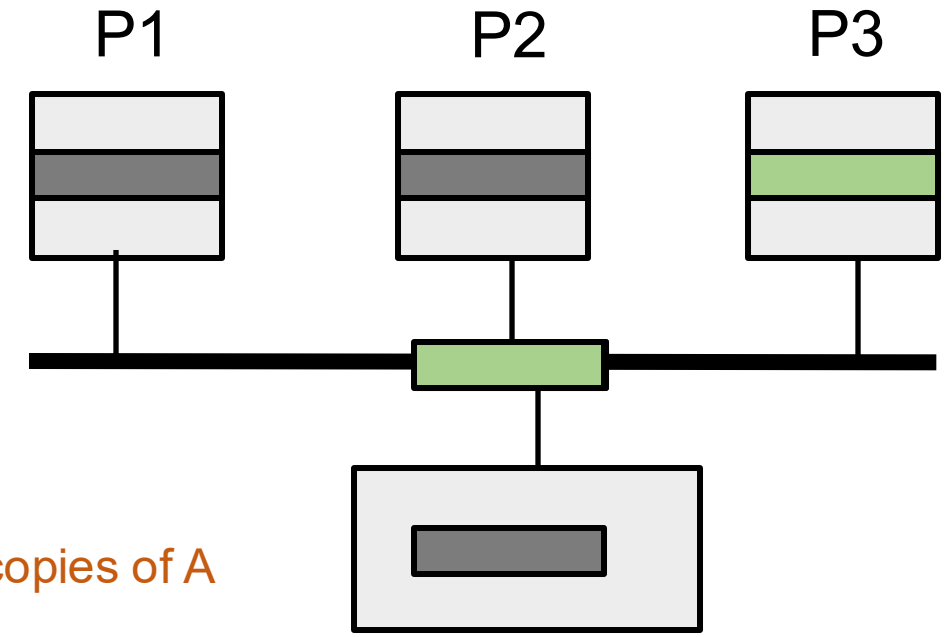
   3.1. A copy of A is brought into P3 Cache

   3.2. P3 modifies its copy of A

   // Two possibilities: *write-update protocol* × *write-invalidate protocol*

Baer p. 269

Example



1. P1 reads some data A
2. P2 reads same data A

   // P1, P2 and memory have valid copies of A
3. P3 has a write miss on A

   3.1. A copy of A is brought into P3 Cache

   3.2. P3 modifies its copy of A

   // Two possibilities: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

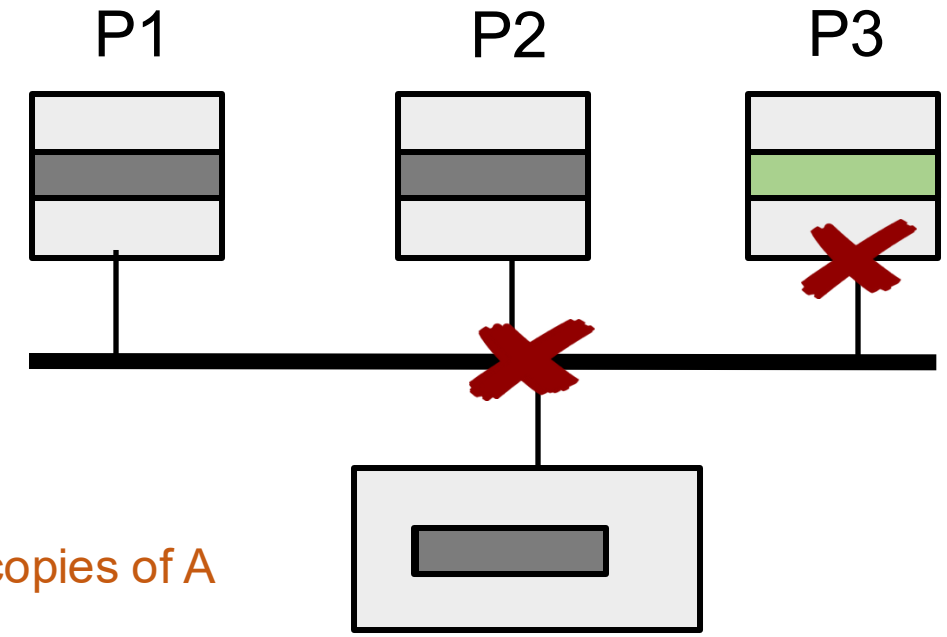
// (similar to write-through policy)

3.3. P3 sends new A to all caches and to main memory

// A is valid in all caches and memory

Baer p. 269

Example



1. P1 reads some data A
2. P2 reads same data A

   // P1, P2 and memory have valid copies of A

3. P3 has a write miss on A

   3.1. A copy of A is brought into P3 Cache

   3.2. P3 modifies its copy of A

   // Two possibilities: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

// (similar to write-through policy)

3.3. P3 sends new A to all caches and to main memory

// A is valid in all caches and memory

// write-invalidate protocol

// (similar to write-back policy)

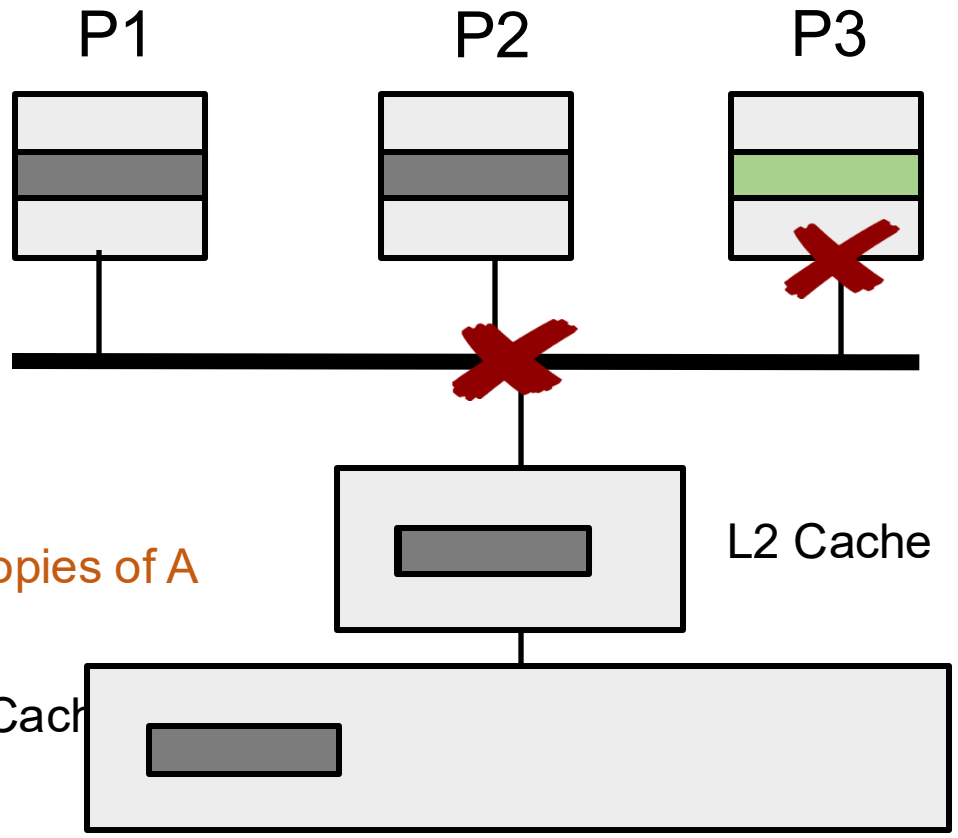
3.3. P3 sends message invalidating A to bus

// P3 has the only valid copy of A

// memory has a stale copy of A

Baer p. 269

Example



1. P1 reads some data A
2. P2 reads same data A

   // P1, P2 and memory have valid copies of A
3. P3 has a write miss on A

   3.1. A copy of A is brought into P3 Cach

   3.2. P3 modifies its copy of A

   // Two possibilities: *write-update protocol* × *write-invalidate protocol*

// write-update protocol

// (similar to write-through policy)

3.3. P3 sends new A to all caches and to main memory

// A is valid in all caches and memory

// write-invalidate protocol

// (similar to write-back policy)

3.3. P3 sends message invalidating A to bus

// P3 has the only valid copy of A

// memory still has a stale copy of A

Baer p. 269

Cache Coherence Protocols

Operations performed by caches in multiprocessors to ensure coherence

- Migration of data to local caches
  - Reduces bandwidth for shared memory
- Replication of read-shared data
  - Reduces contention for access

Snooping protocols

- Each cache monitors bus reads/writes

Directory-based protocols

- Caches and memory record sharing status of blocks in a directory

Memory Consistency

When are writes seen by other processors?

- "Seen" means a read returns the written value
- Can't be instantaneously

Assumptions

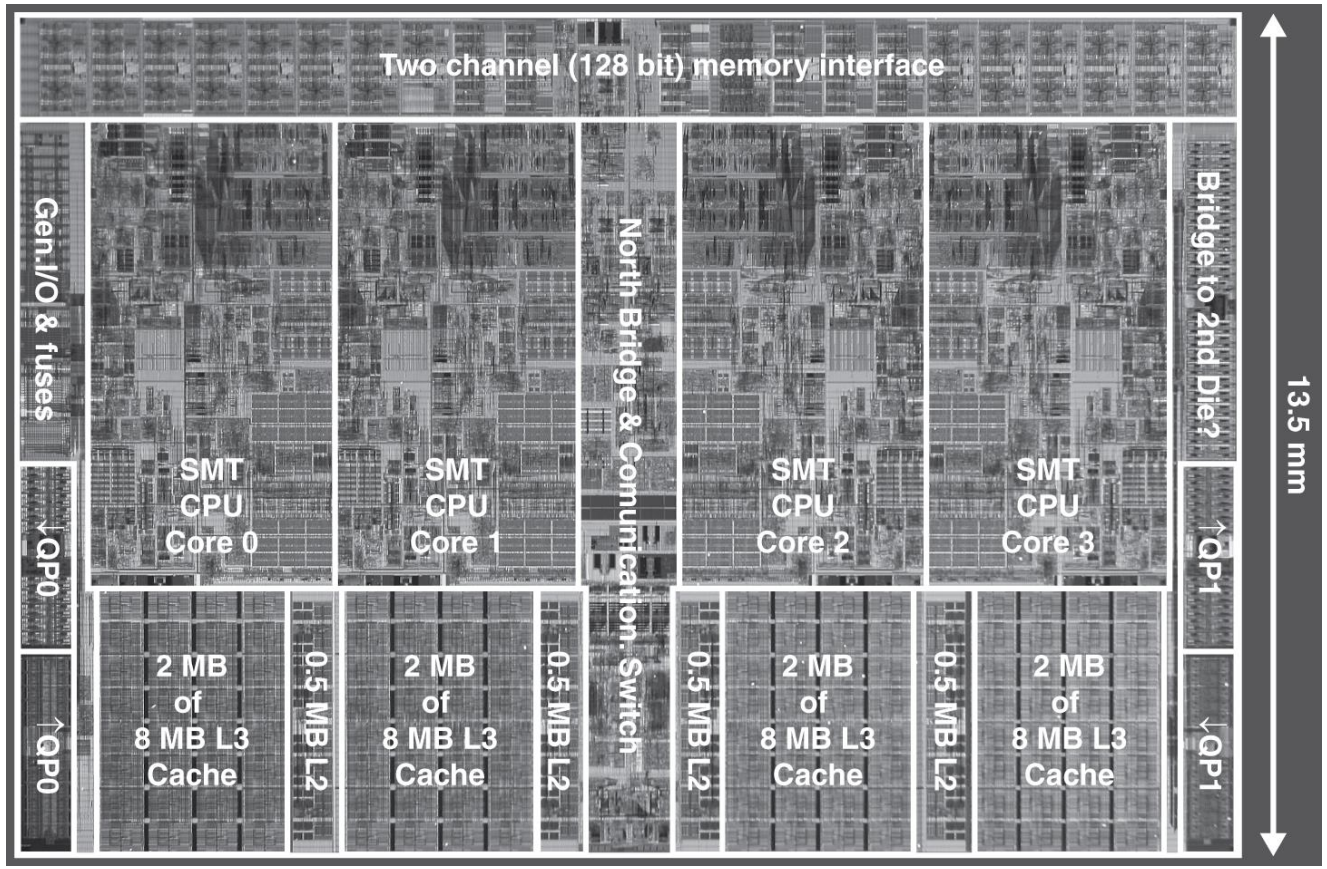
- A write completes only when all processors have seen it
- A processor does not reorder writes with other accesses

Consequence

- P writes X then writes Y ⇒ all processors that see new Y also see new X
- Processors can reorder reads, but not writes

Multilevel On-chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

2-level TLB Organization

	Intel Nehalem	AMD Opteron X4

3-level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles

Reducing Miss Penalty

Return requested word first

- Then back-fill rest of block

Non-blocking miss processing

- Hit under miss: allow hits to proceed
- Miss under miss: allow multiple outstanding misses

Hardware prefetch: instructions and data

Opteron X4: bank-interleaved L1 D-cache

- Two concurrent accesses per cycle

Pitfalls

- Byte vs. word addressing
  - Example: 32-byte direct-mapped cache, 4-byte blocks
    - Byte 36 maps to block 1
    - Word 36 maps to block 4
- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality

Pitfalls

In multiprocessor with shared L2 or L3 cache

- Less associativity than cores results in conflict misses
- More cores ⇒ need to increase associativity

Using AMAT to evaluate performance of out-of-order processors

- Ignores effect of non-blocked accesses
- Instead, evaluate performance by simulation

Concluding Remarks

Fast memories are small, large memories are slow

- We really want fast, large memories ☹
- Caching gives this illusion ☺

Principle of locality

- Programs use a small part of their memory space frequently

Memory hierarchy

- L1 cache < L2 cache < … < DRAM memory < disk

Memory system design is critical for multiprocessors