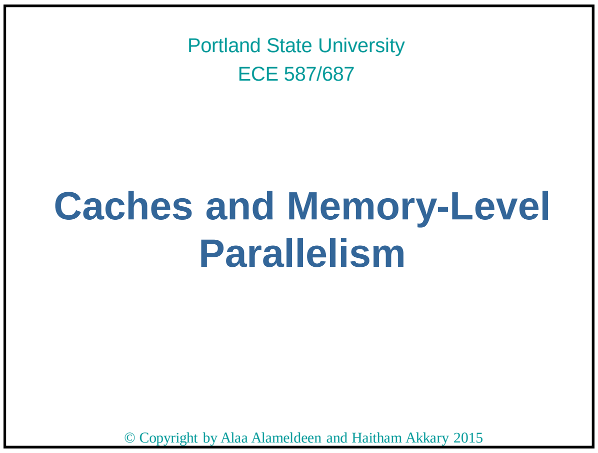
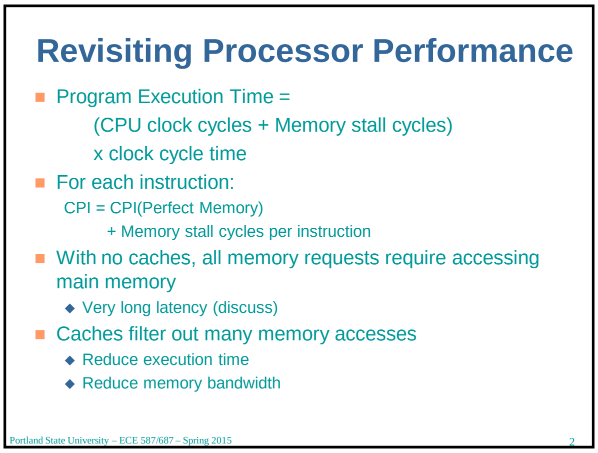


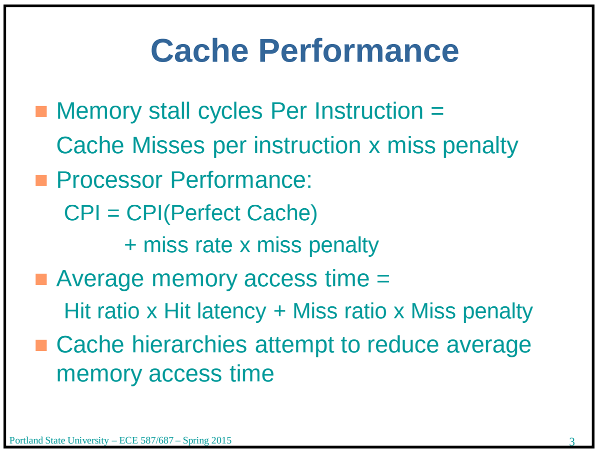
Portland State University
ECE 587/687

Caches and Memory-Level Parallelism



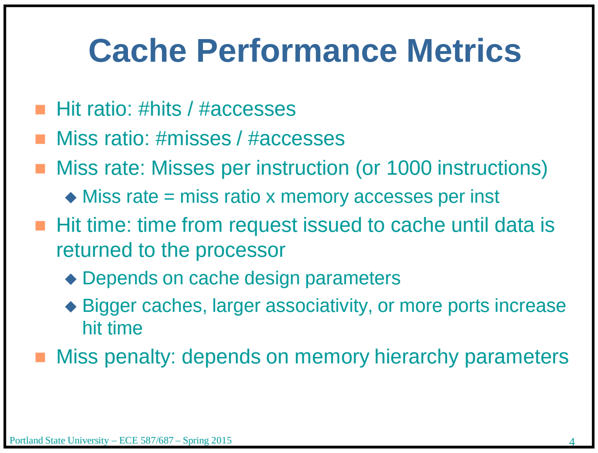
Revisiting Processor Performance

- Program Execution Time =
  (CPU clock cycles + Memory stall cycles)
  x clock cycle time
- For each instruction:
  CPI = CPI(Perfect Memory)
  + Memory stall cycles per instruction
- With no caches, all memory requests require accessing main memory
  - Very long latency (discuss)
- Caches filter out many memory accesses
  - Reduce execution time
  - Reduce memory bandwidth



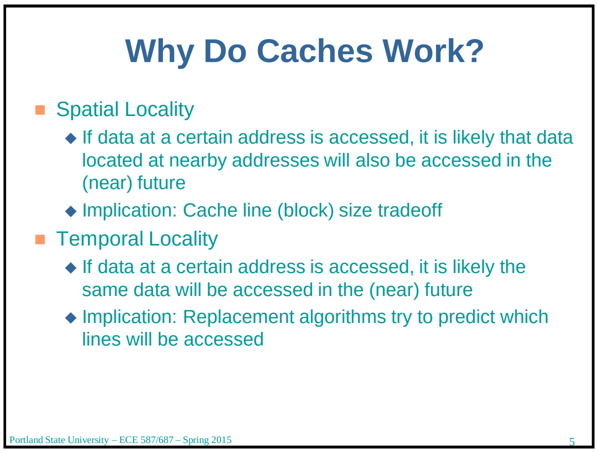
Cache Performance

- Memory stall cycles Per Instruction =
  Cache Misses per instruction x miss penalty
- Processor Performance:
  CPI = CPI(Perfect Cache)
  + miss rate x miss penalty
- Average memory access time =
  Hit ratio x Hit latency + Miss ratio x Miss penalty
- Cache hierarchies attempt to reduce average memory access time



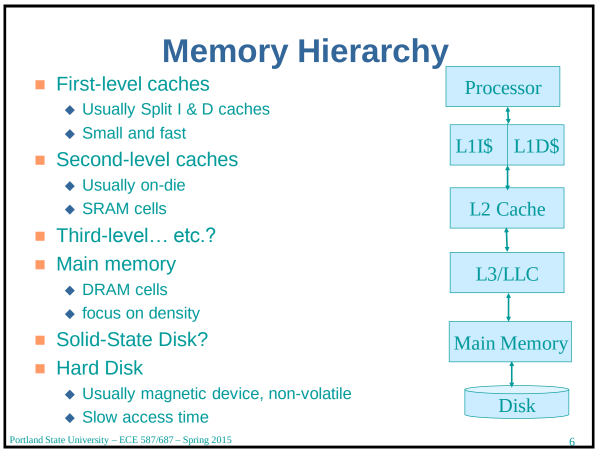
Cache Performance Metrics

- Hit ratio: #hits / #accesses
- Miss ratio: #misses / #accesses
- Miss rate: Misses per instruction (or 1000 instructions)
  - Miss rate = miss ratio x memory accesses per inst
- Hit time: time from request issued to cache until data is returned to the processor
  - Depends on cache design parameters
  - Bigger caches, larger associativity, or more ports increase hit time
- Miss penalty: depends on memory hierarchy parameters



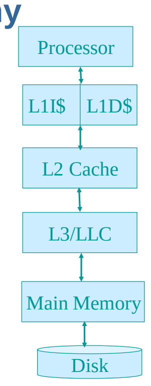
Why Do Caches Work?

- Spatial Locality
  - If data at a certain address is accessed, it is likely that data located at nearby addresses will also be accessed in the (near) future
  - Implication: Cache line (block) size tradeoff
- Temporal Locality
  - If data at a certain address is accessed, it is likely the same data will be accessed in the (near) future
  - Implication: Replacement algorithms try to predict which lines will be accessed



Memory Hierarchy

- First-level caches
  - Usually Split I & D caches
  - Small and fast
- Second-level caches
  - Usually on-die
  - SRAM cells
- Third-level… etc.?
- Main memory
  - DRAM cells
  - focus on density
- Solid-State Disk?
- Hard Disk
  - Usually magnetic device, non-volatile
  - Slow access time



Basic Cache Structure

- Array of blocks (lines)
  - Each block is usually 32-128 bytes
- Finding a block in cache:

Data Address	Tag	Index	Offset

- Offset: byte offset in block
- Index: Which set in the cache is the block located
- Tag: Needs to match address tag in cache

Associativity

- Set associativity
  - Set: Group of blocks corresponding to same index
  - Each block in the set is called a *Way*
  - 2-way set associative cache: each set contains two blocks
  - Direct-mapped cache: each set contains one block
  - Fully-associative cache: the whole cache is one set
- Need to check all tags in a set to determine hit/miss status

Example: Cache Block Placement

- Consider a 4-way, 32KB cache with 64-byte lines
- Where is 48-bit address 0x0000FFFFAB64?
  - Number of lines = cache size / line size = 32K / 64 = 512
  - Each set contains 4 lines $\Rightarrow$ Number of sets = 512/4 = 128 sets
  - Offset bits = $\log_2$ (64) = 6: 0x24
  - Index bits = $\log_2$ (128) = 7: 0x2D
  - Tag bits = 48-(6+7) = 35: 0x00007FFFD

Types of Cache Misses

- Compulsory (cold) misses: First access to a block
  - Prefetching can reduce these misses
- Capacity misses: A cache cannot contain all blocks needed in a program - some blocks are discarded then accessed
  - Replacement policies should target blocks that won't be used later
- Conflict misses: Blocks mapping to the same set may be discarded (in direct-mapped and set-associative caches)
  - Increasing associativity can reduce these misses
- For multiprocessors, coherence misses can also happen
  - Details in ECE 588/688

Cache Replacement and Insertion Policies

- Cache replacement policy:
  - On a cache line fill, which victim line to replace
  - Only applicable to set-associative caches
  - Examples: LRU, more advanced policies
  - Discuss stack algorithms
- Cache insertion policy:
  - When a cache line is filled, what would be its priority in the replacement stack
  - LRU: fill line is inserted in "Most Recently Used" position
  - Other policies: LIP, BIP, DIP
  - Dead block prediction helps determine lines that won't be reused

Non-Blocking Cache Hierarchy

- Superscalar processors require parallel execution units
  - Multiple pipelined functional units
  - Cache hierarchies capable of simultaneously servicing multiple memory requests
    - Do not block cache references that do not need the miss data
    - Service multiple miss requests to memory concurrently
  - Revisit miss penalty with memory-level parallelism

Miss Status Holding (Handling) Registers

- MSHRs facilitate non-blocking memory level parallelism
- Used to track address, data, and status for multiple outstanding cache misses
- Need to provide correct memory ordering, respond to CPU requests, and maintain cache coherence
- Design details vary widely between different processors
  - But basic functions are similar

Cache & MSHR Organization

- Paper Fig 1: block diagram of cache organization
- Main Components:
  - MSHR: One register for each miss to be handled concurrently
  - N-way comparator: Compares an address to all block addresses in MSHRs (N = #MSHRs)
  - Input Stack: Buffer space for all misses corresponding to MSHR entries
    - Size = #MSHRs x block size
  - Status update and collecting networks
- Current implementations combine the MSHR and input stack

MSHR Structure

- Each MSHR contains the following information
  - Data address
  - PC of requesting instruction
  - Input identification tags & Send-to-CPU flags: Allows forwarding requested word(s) to the CPU
  - Input stack indicators (flags): Allows reading data directly from input stack
  - Partial write codes: Indicates which bytes in a word has been written to the cache
  - Valid flag
  - Obsolete flag: either info is not valid or MSHR hit on data in transit

MSHR Operation

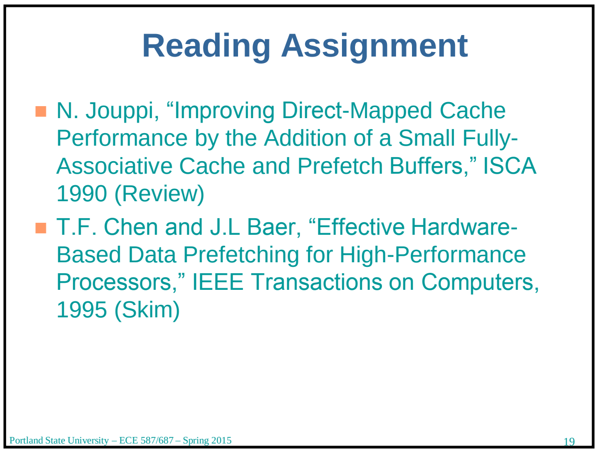
- On a cache miss, one MSHR is assigned
  - Valid flag set
  - Obsolete flag cleared
  - Data address saved
  - PC of requesting instruction saved
  - Appropriate send-to-CPU flags set and others cleared
  - Input identification tag saved in correct position
  - Partial write codes cleared
  - All MSHRs pointing to the same data address purged
- Optimal number of MSHRs: paper figure 2

Virtual vs. Physical Addressing

- Using virtual addresses to access the L1 cache reduces latency
  - Physical addresses need address translation
- However, virtually-addresses caches introduce problems
  - Handing synonyms: multiple VAs mapping to same PA
  - Address translation needed on L1 misses
  - Reverse translation needed for coherence in a multiprocessor system
  - Need to invalidate whole cache on a context switch
- Two-level hierarchies can help achieve the best of both worlds (Wang paper Figure 1)

Reducing Cache Misses

- Cache misses are very costly
- Need multiple cache levels with
  - High associativity or/and victim caches to reduce conflict misses
  - Effective replacement algorithms
  - Data and instruction prefetch
    - Preferably with multiple stream buffers
- More details in Wednesday's class



Reading Assignment

- N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," ISCA 1990 (Review)
- T.F. Chen and J.L Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors," IEEE Transactions on Computers, 1995 (Skim)