10
Multi-Level Strategies

18-548/15-548 Memory System Architecture

Philip Koopman

October 5, 1998

Required Reading: **Cragon 2.6-2.7, 2.8-2.8.2, 2.8.4**
Jouppi paper, 1990 ISCA, pp. 364-373
Supplemental Reading: Hennessy & Patterson 5.5

Carnegie Mellon

Assignments

- **By next class read:**
  - Cragon: 3.6-3.6.1
  - Supplemental:
    - Hennessy & Patterson: 5.9
    - Mogul paper, 1991 Asplos, pp. 75-84
- **Homework 6 due October 14**
- **Lab #4 due October 21**

Where Are We Now?

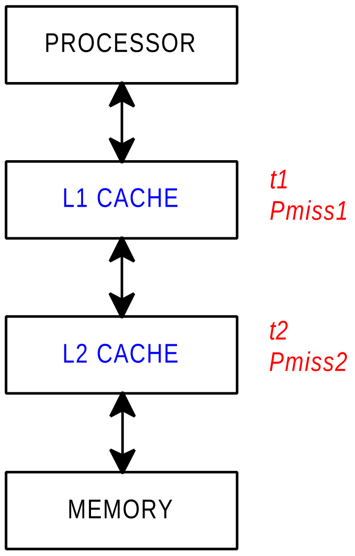
- **Where we've been:**
  - Data organization, Associativity, Cache size
  - Policies -- how to manage the data once it's been arranged
- **Where we're going today:**
  - Multi-level caches to improve performance
    - Another layer to the memory hierarchy
    - Permits employing diverse data organizations
    - Permits exploiting diverse policies
- **Where we're going next:**
  - System-level effects
  - Test
  - Tuning for speed & deeper levels of memory hierarchy

Preview

- **Understanding Multi-Level Caches**
  - Why they are used
  - Organization tradeoffs
  - Policy tradeoffs
- **Optimizing multi-level cache performance -- L1 vs. L2 diversity**
  - Organization
  - Policy
- **Make bandwidth vs. latency tradeoffs**
  - Cache pipelining techniques
  - Block/sector size vs. bus width

Multilevel Caches

- **Small, fast Level 1 (L1) cache**
  - Often on-chip for speed and bandwidth
- **Larger, slower Level 2 (L2) cache**
  - Closely coupled to CPU; may be on-chip, or "nearby" on module



MULTI-LEVEL SIZE & SPEED

Multilevel Cache Sizes

Intel:	L1	L2
80386	sometimes off-chip	none
80486:	8K	none; or 64K+ off-chip
Pentium:	16K (split)	256K - 512K off-chip
Pentium Pro:	16K (split)	256K - 512K on-module
Pentium II:	32K (split)	512K on-module

MIPS:	L1	L2
R2000	128K (split) off-chip	none
R3000	128K (split) off-chip	~1 MB off-chip
R4400	32K (split)	128K-4MB off-chip
R5000	64K (split)	512K-2MB off-chip
R10000	32K (split)	512K-16MB off-chip

- **It's nice to have total cache size bigger than L1 that fits on chip**
  - But, putting even a small L1 on-chip is a Good Thing

Why L2 Cache is Necessarily Slower

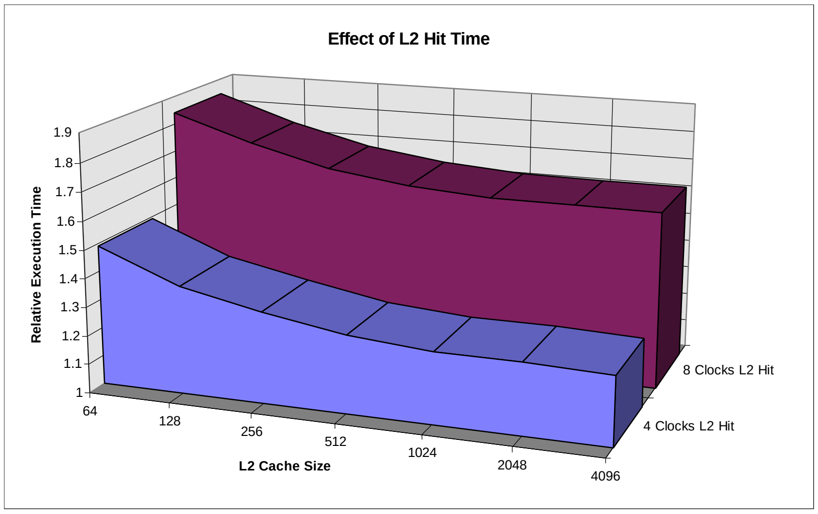
- **Longer critical path**
  - Line length (capacitive delay) grows as square root of memory array size
  - Addressing & data multiplexing grow as *n* log *n* with array size
- **Off-chip access is slower than on-chip access**
  - Off-chip driving delays
    - Pad drivers
    - Traces
    - EMI/analog limitations to circuit board speed & planar RF transmission
    - Length vs. speed of light
  - Allowance for clock skew
  - Limits on power dissipation (SRAM array; pad drivers)
- **Off-chip access is narrower than on-chip access (less bandwidth)**
  - Pins cost money -- packaging, board density
    - May need multi-cycle transfers for larger blocks
  - On-chip routing is cheaper
    - Block size limited by memory array dimensions, not by pin count

Two-Level Miss Rates

- **Local miss rate: misses in cache / accesses to cache**
  - L1 cache => P_{miss1}
  - L2 cache => P_{miss2}
  - Useful for optimizing a particular level of cache given a fixed design otherwise
- **Global miss rate: misses in cache / accesses from CPU**
  - L1 cache => P_{miss1}
  - L2 cache => $P_{miss1} * P_{miss2}$
    (only L1 misses are seen by L2, which has a local miss ratio of P_{miss2})
  - Good for measuring traffic that will be seen by next level down in memory hierarchy
- **Global L2 miss rate equals miss rate of composite cache**

Example Effect of Relative L2 Speed

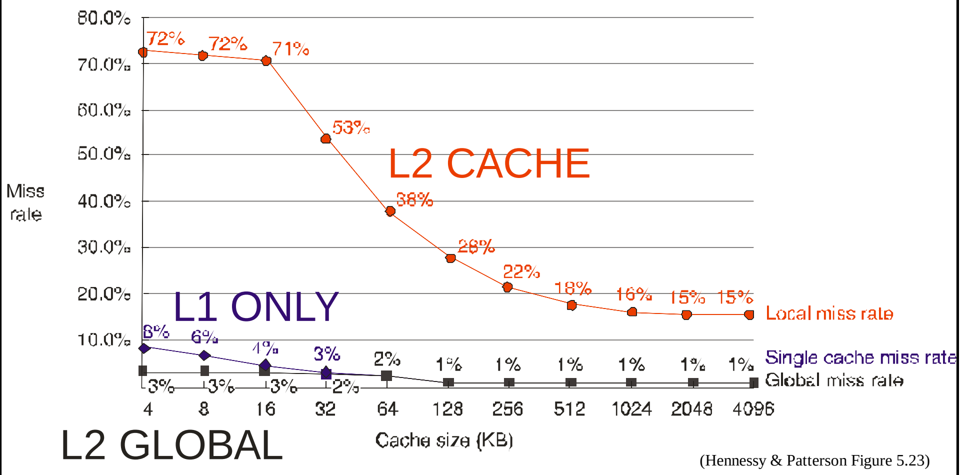
- **L1 cache fixed at 32 KB**



(Data from Hennessy & Patterson Figure 5.24)

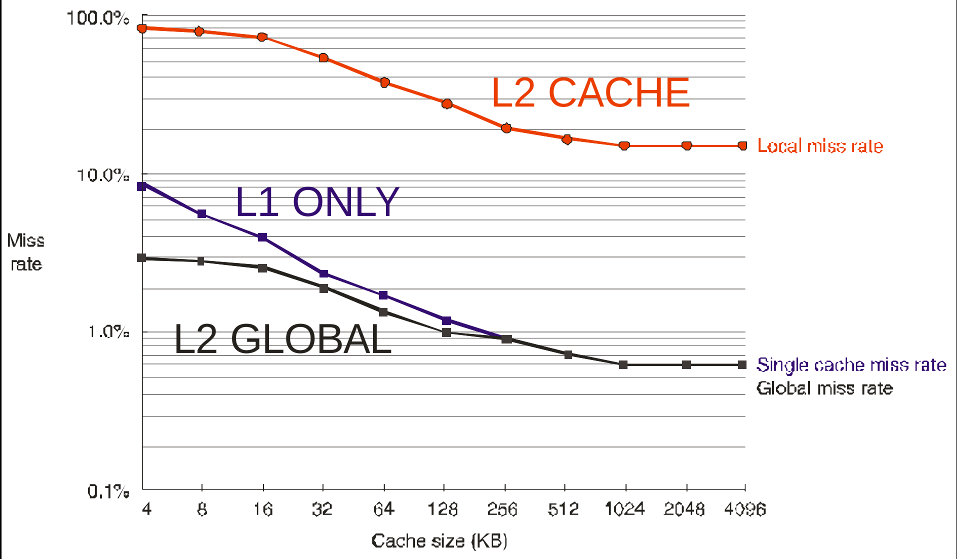
Example Performance

- Local/global miss rate is for L2 cache given 32 KB L1 cache
- Single cache miss rate assumes only L1 cache of varying size



(Hennessy & Patterson Figure 5.23)

Same Chart, Log-Log Scale



Evaluating Multi-Level Miss Rates

- **Use Global Miss rates when evaluating traffic filtering of 2-level caches**
  - Effectiveness of L2 strategy depends on which L1 strategy is used
    - Changing L1 strategy may require changing L2 strategy as well
  - Global L2 miss rate equals effective composite cache miss rate
- **Sequential forward model (local miss rates):**

$$t_{ea} = t_{hitL1} + (P_{miss1} * t_{hitL2}) + (P_{miss1} * P_{miss2} * t_{transport})$$

[Note: Cragon uses global miss rates for this equation, which might be confusing]

MULTILEVEL FOR POLICY & ORGANIZATION DIVERSITY

Diversity Motivation

- **L1 and L2 should have differences to improve overall performance**
  - Small, fast, relatively inflexible L1
  - Larger, slower, relatively flexible L2
- **Issues:**
  - Cache size & virtual memory address translation
  - Split vs. Unified & bandwidth vs. flexibility
  - Write through vs. write back & write allocation
  - Block size & latency vs. bandwidth
  - Associativity vs. cycle time
- **Following slides are *representative* tradeoffs**
  - The cache system in its entirety is what matters, not just any single parameter

Cache Size & Address Translation

- **Late select cache -- cache access performed in parallel with address mapping**
- **Virtual memory page size determines unmapped address bits**
  - 4 KB page -- 12 bits -- maximum direct map cache size 4 KB
  - 8 KB page -- 13 bits -- maximum direct map cache size 8 KB
- **Example: Pentium Pro**
  - Virtual memory uses 4 KB pages
    - 12 unmapped bits available for cache access
  - But 16K total L1 cache size!
  - Obvious solutions: use only 4 KB to address the cache sets
    - Split caches -- only need to address an 8K cache
    - Then make each cache 2-way+ set associative -- only need to address 4K
      - » (D-cache is 2-way; I-cache is 4-way)
    - 4K sets takes 12 address bits; and 12 unmapped address bits available

Maximum L1 Cache Size *vs.* L2 Cache Size

- **In the absence of "slight-of-hand," L1 cache size is limited by combination of virtual memory page size and organization**
  - P = VM page size (often 4KB or 8 KB)
  - A = Associativity (sectors per set)
  - N = number of caches (1=unified 2=split -- assume equal sizes)
- **Max total L1 cache = P * A * N**
  - Can exceed using mapping restrictions for virtual memory
- **But, L2 cache is accessed after translation -- no size restriction!**
  - Size driven by cost & physical limits:
    - Want single bank of cache chips to avoid chip select delays
    - Want few cache chips for address line loading & space (ideally, single chip)
    - Want flexibility for cache size for cost/speed tradeoffs depending on customer budget

Split *vs.* Unified

- **Split caches give bandwidth; unified caches give flexibility**
  - Use split L1 combined with unified L2 for good aggregate performance
- **Split L1 cache advantages**
  - Can provide simultaneous data & instruction access -- high bandwidth
  - Gives factor of 2 improvement with address translation size limit
  - Reduces hit rate, but not catastrophic if L2 cache is available to keep miss penalties low
- **Unified L2 cache advantages**
  - Reduces pin & package count -- only one path needed to off-chip L2
  - Can be used for I-cache/D-cache coherence (invalidate I-cache line on modification)
  - Reduces brittleness of assuming half of memory used is instructions
    - Some working sets are mostly data, some are mostly instructions

Write Policies

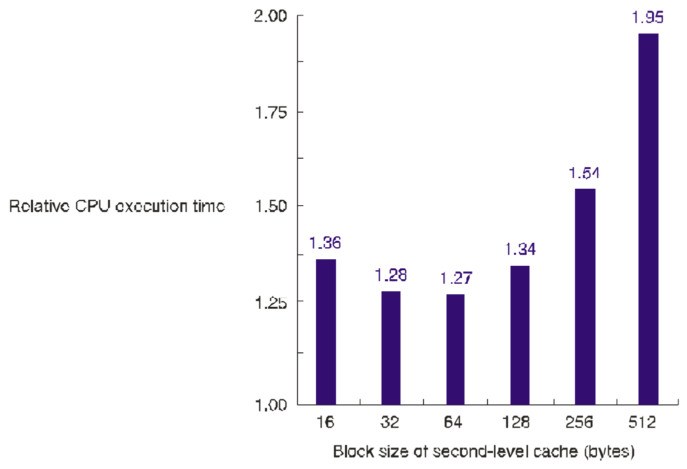
- **Write through? Write allocation?**
  - L1: write through + no-write allocate; L2 write back + write-allocate
- **L1 cache: advantages of write through + no-write-allocate**
  - Simpler control
  - No stalls for evicting dirty data on L1 miss with L2 hit
  - Avoids L1 cache pollution with results that aren't read for a long time
  - Avoids problems with coherence (L2 always has modified L1 contents)
- **L2 cache: advantages of write back + write-allocate**
  - Typically reduces overall bus traffic by "catching" all the L1 write-through traffic
  - Better able to capture temporal locality of infrequently written memory locations
  - Provides a safety net for programs where write-allocate helps a lot
    - Garbage-collected heaps
    - Write-followed-by-read situations
    - Linking loaders (if unified cache, need not be flushed before execution)

Block/Sector Size

- **Balancing miss rate vs. traffic ratio; latency vs. bandwidth**
- **Smaller L1 cache sectors & blocks**
  - Smaller sectors reduces conflict/capacity misses
  - Smaller blocks reduces time to refill cache block (which may reduce CPU stalls due to cache being busy for refill)
  - But, still want blocks > 32 bits
    - Direct access to long floats
    - Exploit block transfers from L2 cache
    - Limit tag storage overhead space for sectors
- **Larger L2 cache sectors & blocks**
  - Larger sectors create less of a conflict problem with large cache size
  - Main memory has large latency on L2 miss, so proportionally lower cost to refill larger cache block once memory transfer started
  - Once L1 cache block is refilled, larger L2 block refill can continue with lower probability of stall (refill overlapped with/hidden by subsequent L1 cache hits)

Larger Block Sizes for L2

- **Conflict misses relatively less important with larger cache**
  - If L2 cache is 16x bigger, might be OK to have 2x or 4x larger block size



(Hennessy & Patterson Figure 5.25)

Associativity

- **Balance complexity, speed, efficiency**
- **L1 -- no clear winner**
  - Direct mapped L1 gives faster cycle time
    - But, lower hit rate on an already small cache
  - Set associative L1 gives slower cycle time, better hit rate
    - Set associativity may be encouraged by address translation issue
    - May be less of a problem with on-chip L1 cache
- **L2 -- no clear winner**
  - Direct mapped L2 minimizes pin & package count for cache
    - Only 1 tag need be fetched
    - No problem with multiplexing multiple data words based on tag match
    - Set associativity less advantageous for really large caches
  - Set associative L2 gives flexibility
    - Less brittle to degenerate cases with data structures mapped to same location
    - Associative time penalty less of an issue for L2 cache than L1 cache (smaller percentage of total miss delay)

Multi-Level Inclusion

- **Complete inclusion means all elements in highest level of memory hierarchy are present in lower levels (also called “subset property”)**
  - For example, everything in L1 is also in L2 cache
  - Useful for multiprocessor coherence; only have to check lowest cache level
- **Inclusion requires**
  - Number of L2 sets >= number of L1 sets
  - L2 associativity >= L1 associativity
  - L1 shares LRU data with L2 to coordinate replacements
- **Whenever non-inclusion is encountered, special effort is required to maintain coherence for:**
  - Write back L1 cache (L2 might not know L1 has been modified)
  - Temporary non-inclusion for pending writes in write buffer
  - L2 block size > L1 block size
    - Flush/evict any L1 block mapping to invalidated L2 block

L1 vs. L2 Tradeoff Examples

Pentium Pro	**L1**	**L2**	
Size	16KB	none - 256KB - 512KB	
Organization	Split (8KB + 8KB)	Unified	
Write Policies	programmable; same for both		
Block size	32 bytes	32 bytes	(1 block/sector)
Associativity	D: 2-way; I: 4-way	4-way	

MIPS R10000	**L1**	**L2**
Size	64KB	512KB - 16 MB
Organization	Split (32KB + 32KB)	Unified
Write Policies	write back	write back
Block size	D: 32 bytes I: 64 bytes	64 or 128 bytes
Associativity	2-way	2-way

- **Isn’t L1 write back a problem for coherence?**

Concept In Everyday Life:

- **What's an everyday example of 2-level caching with differing management policies**
  - What is the motivation for the caching?
  - How are the policies different between the levels?

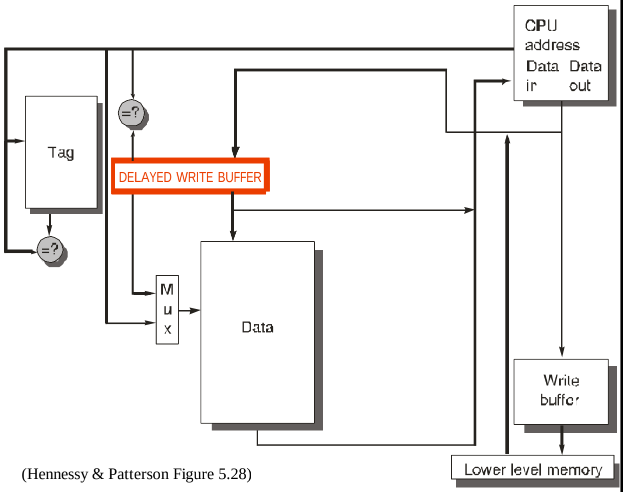
BANDWIDTH VS. LATENCY

Cache Bandwidth *vs.* Latency Tradeoffs

- **Pipelined caches**
  - Multiple concurrent access operations gives increased throughput
  - But:
    - Increases latency for read accesses
    - Risk of stall for deferred write access
- **Block size vs. transfer size**
  - Large blocks increase memory bandwidth & refill latency
  - But, large blocks decrease miss ratio

2-Stage Pipelined Writes

- **Decouple write access to tags and data for faster (average) write hits**
  - Stage 1: check tag for write hit
  - Stage 2: actually write data

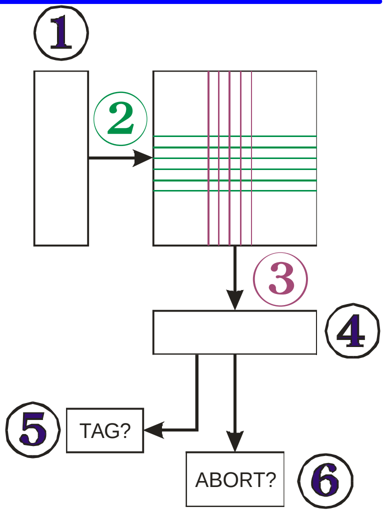


(Hennessy & Patterson Figure 5.28)

- **But, need 2nd port for reads, or stall for write followed by read**

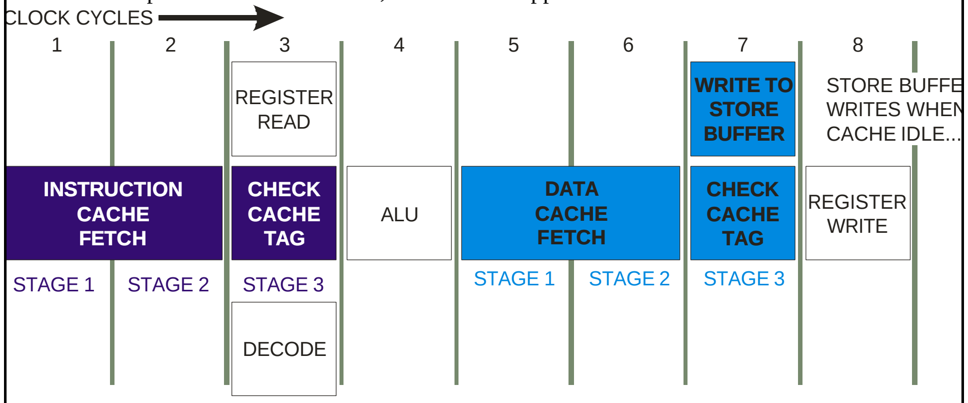
3-Stage Generalized Cache Access Pipelining

- **3-stage pipeline only works for direct mapped caches**
  - Possible stages (not necessarily how it is really done)...
  - Stage 1: (1) decode address
    (2) drive word lines
  - Stage 2: (3) read & latch memory row
    (4) select data word; precharge bit lines for next access
  - Stage 3: (5) check tag for match while speculatively using data
    (6) abort use of data if tag mismatch



MIPS R4000 Pipelined Cache Example

- **Three clock cache pipeline: address decode, fetch, tag compare**
  - New cache access can be started each clock cycle --> "superpipelining"
    - Store buffer uses idle cache cycles to deposit written data
    - Blocking cache -- tag mismatch causes stall for cache miss
    - Split cache for bandwidth; both direct mapped



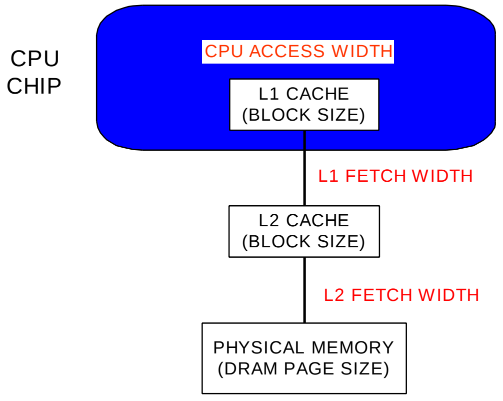
MIPS R4000 PIPELINE

Pipelined Cache Tradeoffs

- **Increases latency**
  - Takes *3 clocks* until L1 cache miss is declared!
  - 2 clock latency from Load instruction to data available at ALU
    - 1 clock for ALU to do address arithmetic counts as that instruction's execution
    - 2 clocks for D-cache read (assume result forwarded to ALU before register write)
- **Increases throughput**
  - *Up to* 3x improvement in clock speed if cache+tag check was critical path
  - Increasingly useful as larger, slower L1 caches are used
- **Requires direct mapped cache for 3rd stage**
  - Speculative execution needs correct data available before tag check
  - (2-way set associative would require 2 ALUs, 2 ports to cache write buffer)

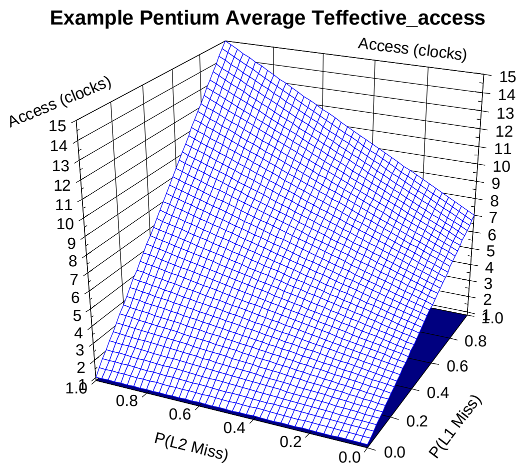
Multi-Level Block Sizes

- **Tradeoff for large block sizes vs. available access width**
- **Example: Pentium+430HX set**
  - CPU Access Width
    - 256 bits instruction / clock
    - 2 @ 32 bits data / clock
  - L1 Block Size = 256 bits
    - L1 Fetch Width = 64 bits
    - Example L2 Access: 3-1-1-1=6
  - L2 Block Size = 256 bits
    - L2 Fetch Width = 64 bits
    - Example L2 Miss: 8-2-2-2=14
  - DRAM Page size is proportional to sqrt(chip size)
    - (*e.g.*, 16K bits for 16Mx4 chip)



Example of Multi-Level Access Time Equation

- $t_{ea} = t_{L1hit} + P_{L1miss} * t_{L2hit} + P_{L1miss} * P_{L2miss} * t_{L2miss}$
  - Pentium example (using marginal L2 miss penalties, not absolute)
    - t_{L1hit} = 1 clock t_{L1miss} =6 clocks t_{L2miss} = 14 clocks



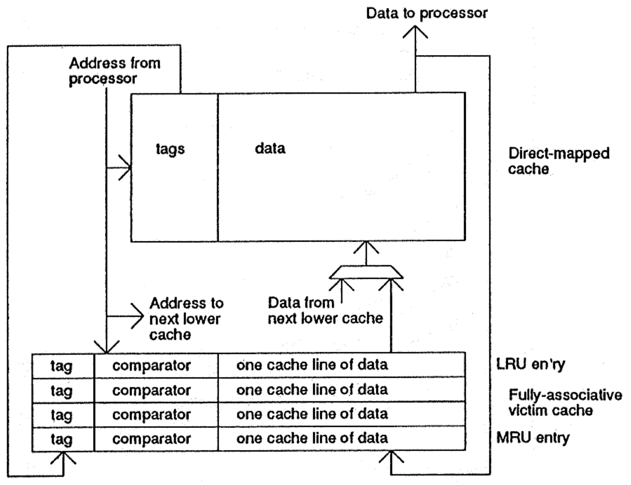
JOUPPI CACHES

Jouppi Caches

- **ISCA paper on small auxiliary caches**
  - Assume L1 cache is direct mapped
  - How can a small associative cache take the edges off conflict miss problems?
  - Attempt to get set associative-like behavior with direct mapped speed and simplicity

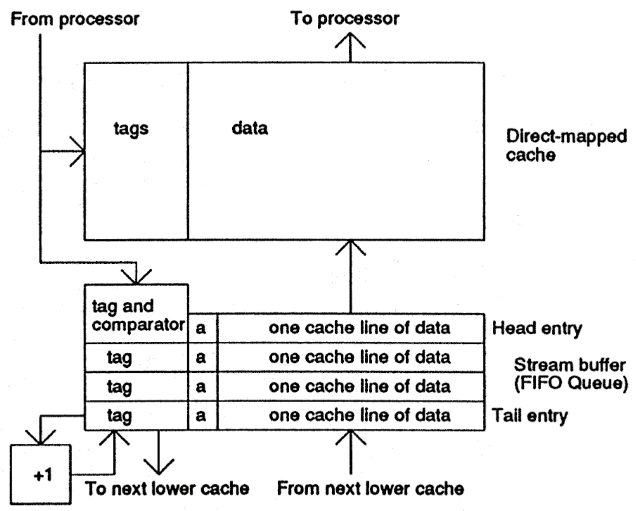
Victim Cache

- **Stores eviction victims**
  - Simulates set associativity for the last few sets touched
  - Some, or maybe even most, conflict misses removed; better than miss cache



Stream Buffer

- **Hardware prefetching "push" into cache, as opposed to CPU "pull"**
  - Need not be sequential; can be strided



REVIEW

Review

- **Multi-level caches are used to increase overall cache size & decouple CPU cache accesses from the memory bus**
  - Bigger is better; but L1 caches have size limits
  - L1 and L2 caches often have different tradeoffs
    - L1: split; write through/no-allocate; smaller blocks; low associativity
    - L2: unified; write back/allocate; larger blocks; moderate associativity
- **On-chip L1 cache design is highly constrained**
  - Size, aspect ratio, area usage
- **Both bandwidth and latency matter**
  - Cache pipelining techniques may help
  - Block/sector size vs. bus width is a key tradeoff
- **Jouppi caches demonstrate than sometimes a small auxiliary cache can big a good "win"**

Key Concepts

- **Latency**
  - Fast L1 cache can hide latency of slower L2 cache
  - Slow L2 cache can hide latency of even slower main memory
- **Bandwidth**
  - Pipelining cache accesses can improve bandwidth
- **Concurrency/Replication**
  - Heterogeneous replication provides diversity
    - L1 vs. L2 caches
    - Associative Jouppi caches vs. direct mapped L1 cache
- **Balance**
  - Balancing L1 and L2 parameters provides emergent behavior better than simply using a larger L1 cache