7
Associativity

18-548/15-548 Memory System Architecture
Philip Koopman
September 16, 1998

Required Reading: **Cragon pg. 166-174**

Assignments

- **By next class read about data management policies:**
  - Cragon 2.2.4-2.2.6, 3.5.2
  - Supplemental Reading:
    - VanderWiel paper, July 1997 Computer, pp. 23-30
    - Przybylski paper, 1990 ISCA, pp. 160-169 (class reserve in library)
- **Homework 4 due Wednesday September 23**
- **Lab 2 due Friday September 25**
- **Test #1 Monday September 28**
  - In-class review Wednesday September 23; look at sample tests before then

Where Are We Now?

- **Where we've been:**
  - Split I-/D- caching
  - Block size tradeoffs from miss rate & traffic ratio point of view
- **Where we're going today:**
  - Associativity
    - Having more than one victim available for cache sector replacement
    - In general, associative searching (how to find something based on its value instead of its address)
- **Where we're going next week:**
  - Policies for managing cached data
  - Multi-level caching & buffering

Preview

- **Degrees of associativity**
  - Direct mapped
  - Fully associative
  - Set Associative
- **Implementing associativity**
  - How data is looked up from the cache (in detail)
  - Performance costs & benefits of increased associativity
  - Hacks & tricks

Associativity

- **In some cases, two or more frequently used data words might end up mapped to same cache set**
- **Associativity - reserves multiple cache sectors for each potential address set**
  - All cache sectors that are candidates for holding any particular address form a set
- **Level of associativity varies depending on sectors/set**
  - Number of sectors in a set used to describe associativity
    - 1 sector/set is Direct Mapped = “1-way set associative”
    - *k* sectors/set is k-way set associative
    - All sectors in one set is fully associative
  - Higher associativity can improve hit rate
    - Reduces conflict misses
    - Costs more
    - Slower cycle time because of comparator

BASIC ASSOCIATIVITY:
DIRECT-MAPPED, SET, FULL

Associativity Options

- **[Sets, Sectors, Blocks, Words]**

- **Direct Mapped cache [S, 1, B, W]**
  - Each memory location maps into one and only one cache sector
  - Fast, simple, inefficient? (this is controversial)
  - Maximum conflict misses

- **Fully Associative cache [1, Se, B, W]**
  - Any sector can map to anywhere in memory
  - Slow, complex, efficient
  - No conflict misses given perfect replacement policy

- **Set Associative cache [S, Se, B, W]**
  - Groups of sectors ("sets") form associative pools
  - A compromise
  - Can greatly reduce conflict misses except in degenerate cases

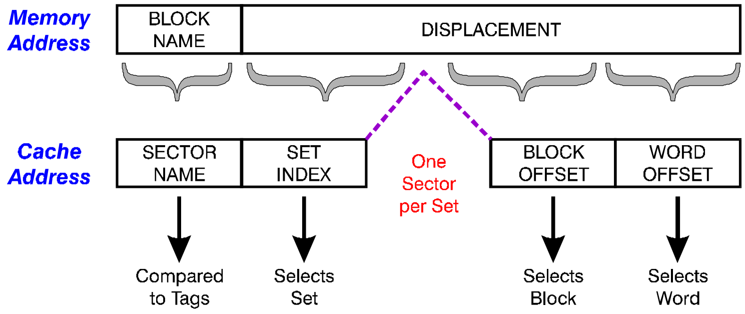
Direct Mapped Structure

- **Example: [8, 1, 4, 2]**
  - 8 sets, 1 sector/set, 4 blocks/sector, 2 words/block

		BLOCK 0				BLOCK 1				BLOCK 2				BLOCK 3			
SET 0	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
SET 1	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
SET 2	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
SET 3	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
SET 4	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
SET 5	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
SET 6	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
SET 7	TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD

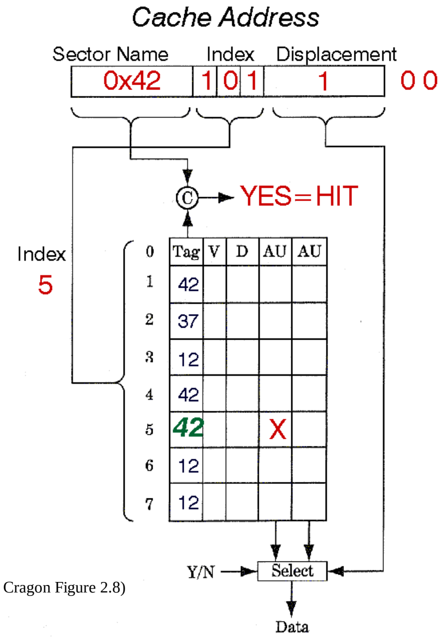
Direct Mapped Addressing

- **Each set has exactly 1 sector**
  - Exactly one possible location for any memory location to map to in cache
- **One sector per set -- [S, 1, B, W]**



Direct Mapped Operation

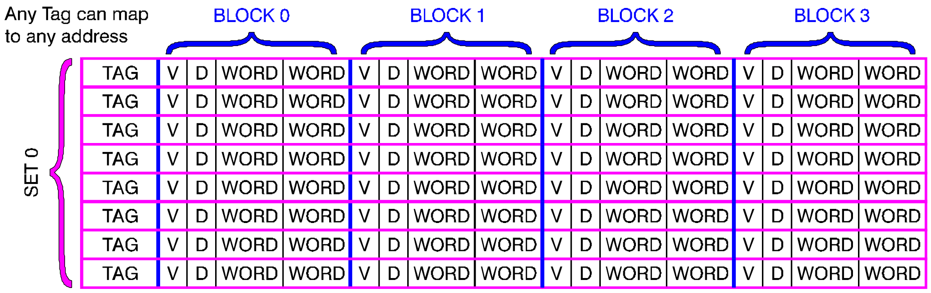
- **[8, 1, 1, 2]**
  - 8 Sets
  - 1 sector/set
  - 1 block/sector
  - 2 words/block
- **Same tag value can occur in multiple locations**
  - Only tag at specified index is checked



(From Cragon Figure 2.8)

Fully Associative Structure

- **All sectors are together in a single set**
  - Any memory location can map to any sector
- **Example: [1, 8, 4, 2]**
  - 1 set, 8 sectors/set, 4 blocks/sector, 2 words/block



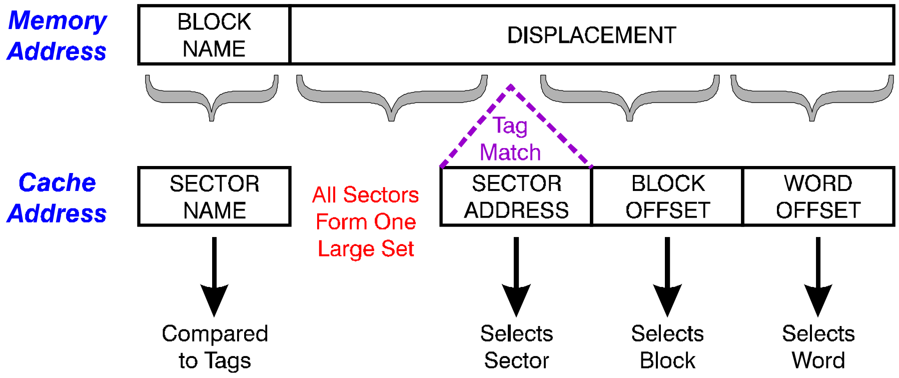
Any Tag can map to any address

SET 0

	BLOCK 0				BLOCK 1				BLOCK 2				BLOCK 3			
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD
TAG	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD	V	D	WORD	WORD

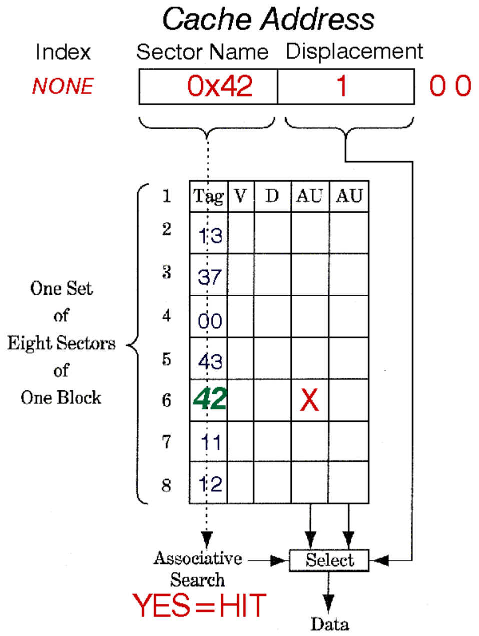
Fully Associative Addressing

- **One sector per set -- [1, Se, B, W]**



Associative Cache ("fully associative")

- **[1, 8, 1, 2]**
  - 1 "set" of all sectors
  - 8 sectors/"set"
  - 1 block/set
  - 2 words/block
- **Tag value must be unique within cache**
  - All tags are checked in parallel for a match = "associative search"



1	Tag	V	D	AU	AU
2	13				
3	37				
4	00				
5	43				
6	42			X	
7	11				
8	12				

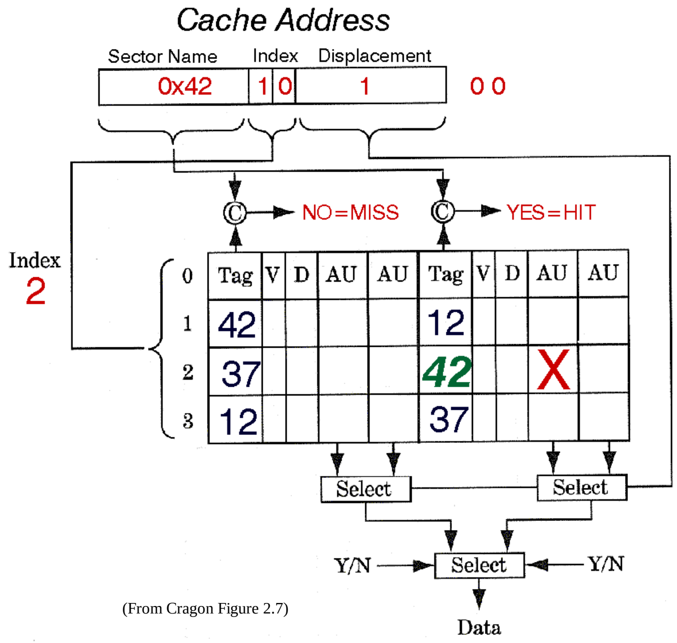
(From Cragon Figure 2.6)

Associative tradeoffs

- **Can be quite slow because of large number of comparisons**
  - All tags must be checked before "hit" or "miss" can be declared
  - Uses a content-addressable memory cell > 3x bigger than SRAM bit
- **Complete associativity gives diminishing returns for large cache**
  - Conflict misses decrease as there are a large number of sets available
  - BUT, commonly used for Virtual Memory TLB

Set Associative Cache [S, Se, B, W]

- **Example: [4, 2, 1, 2]**
  - 4 sets
  - 2 sectors/set
  - 1 block/sector
  - 2 words/block
  - "2-way set associative cache"



Cache Address

Sector Name Index Displacement

0x42 | 1 | 0 | 1 | 0 0

Index 2

NO=MISS

YES=HIT

0	Tag	V	D	AU	AU	Tag	V	D	AU	AU
1	42					12				
2	37					42			X	
3	12					37				

Select

Select

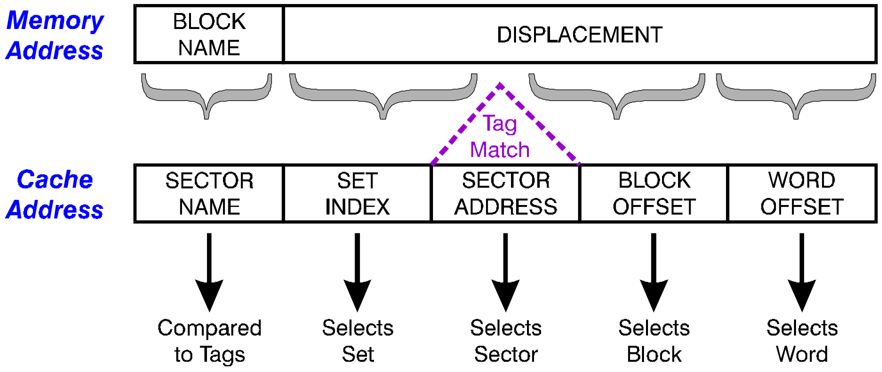
Y/N → Select ← Y/N

Data

(From Cragon Figure 2.7)

Set Associative Address Formation

- **Same as the "general" case from earlier lecture**



Memory Address

BLOCK NAME | DISPLACEMENT

Tag Match

Cache Address

SECTOR NAME | SET INDEX | SECTOR ADDRESS | BLOCK OFFSET | WORD OFFSET

Compared to Tags

Selects Set

Selects Sector

Selects Block

Selects Word

Set Associative Tradeoffs

- **Robust to accidental mapping of heavily used addresses to the same sector**
  - Cache can provide up to *k* hit locations within same set for *k*-way set associativity
  - As number of sets gets large (large cache size), chance of getting unlucky with *k*+1 distinct accesses to a particular set within a loop reduces
    - *k+1* distinct accesses is the pathological worst case for LRU -- 0% hit rate
- **Compromises complexity/latency compared to fully associative and direct-mapped**
  - Can simply read all tags in parallel and use *k* comparators for *k*-way set associativity (want entire set in same memory array row; discussed later)
  - Doesn’t require full content-addressable memory arrangement
  - Selecting which comparator found the match and gating data increases critical path

ASSOCIATIVITY TECHNIQUES

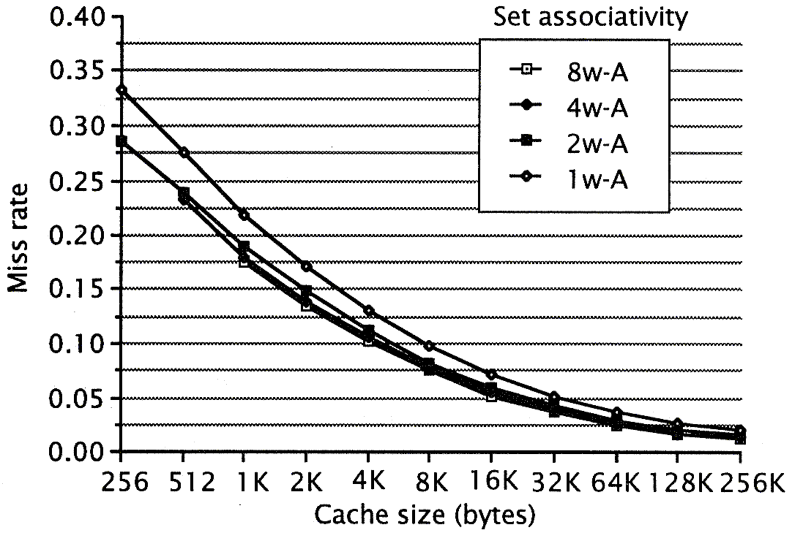
Set Associativity for Larger Caches

- **Higher associativity may be only reasonable way to increase physically addressed cache size**
  - Page offset bits unaffected by translation -- are only bits available for cache addressing before address translation (for concurrent address translation & cache access)
  - Cache limited to $2^{\#\text{offset bits}}$ sets; increasing associativity permits use of larger cache size
  - (Can also use virtually addressed cache, but this causes problems with aliasing for data/unified caches)
- **Example: IBM 3033 had 16-way set-associative cache of 64KB**
- **BUT:**
  - Problem only applies to L1 cache, which is generally small for speed reasons anyway

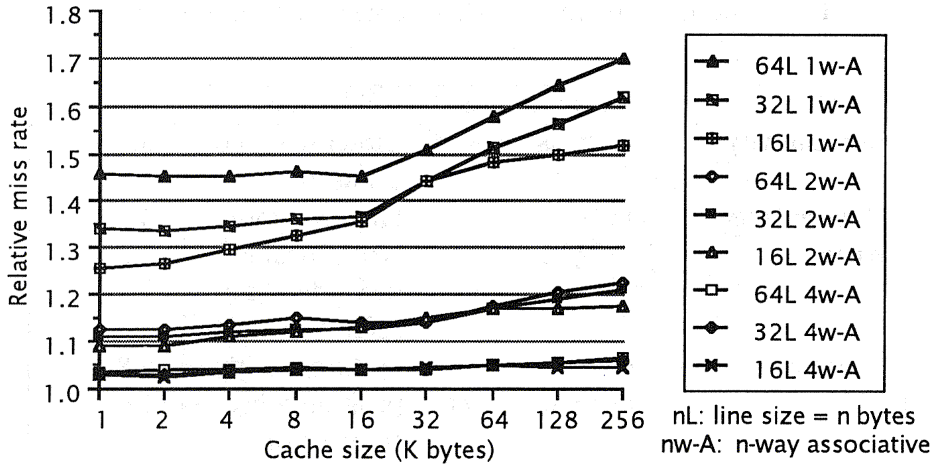
DTMR Associativity Data

- **16-byte lines**

(Flynn Figure 5.13)



Degradation Compared to Fully Associative



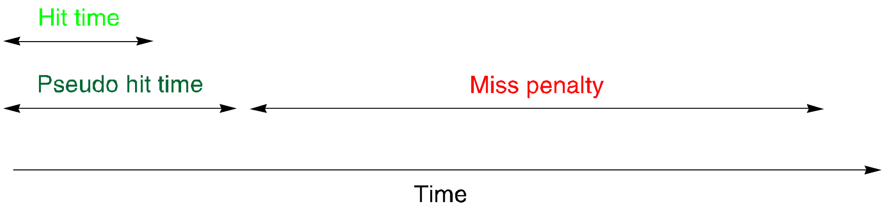
(Flynn Figure 5.14)

Concept In Everyday Life:

- **What everyday situations/systems display associative look-up behavior?**
  - Fully associative
  - Set associative
  - Direct-mapped lookup

A Hack: Pseudo-Associative Caches

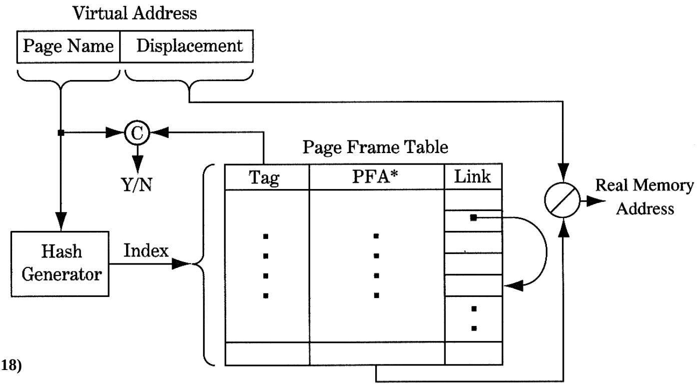
- **Direct-mapped cache with 2 access attempts**
  - (*e.g.*, if cache address 0x300 is a miss, flip top address bit and try 0x100)



- **Variable access time -- better for L2 cache+**
  - Hit on 1st attempt same as normal hit
  - If miss on 1st attempt, modify address and try a 2nd time
  - It's a win if:
    - Direct-mapped cache faster than 2-way set associative cache
    - Miss time is large
    - Higher level can tolerate non-uniform hit time

Associativity For Big Caches

- **Set associativity might not work for really big cache structures, such as inverted page tables**
- **Use general hashing techniques from your favorite algorithms/data structures course**
  - Inverted page table example (similar to HP PA):

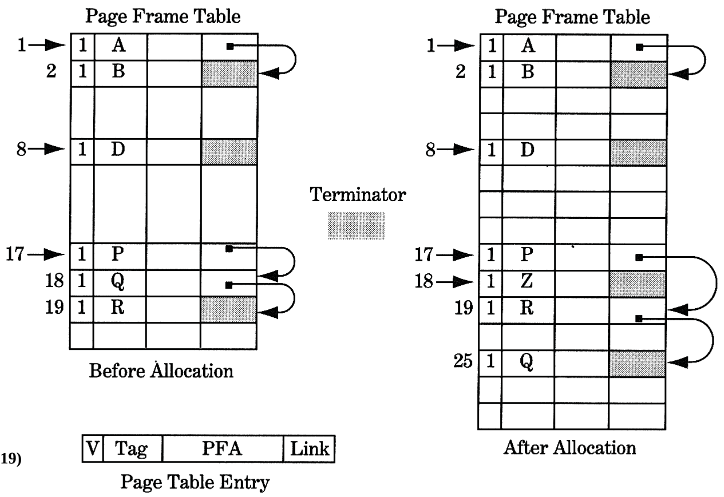


(Cragon Figure 3.18)

*Page Frame Address

Hashing Name Collisions

- **Typically uses a linked list of entries from each hashed entry point**
  - In reality, don’t want hash table more than about half-full or so; otherwise it gets clogged with long lists to search.



(Cragon Figure 3.19)

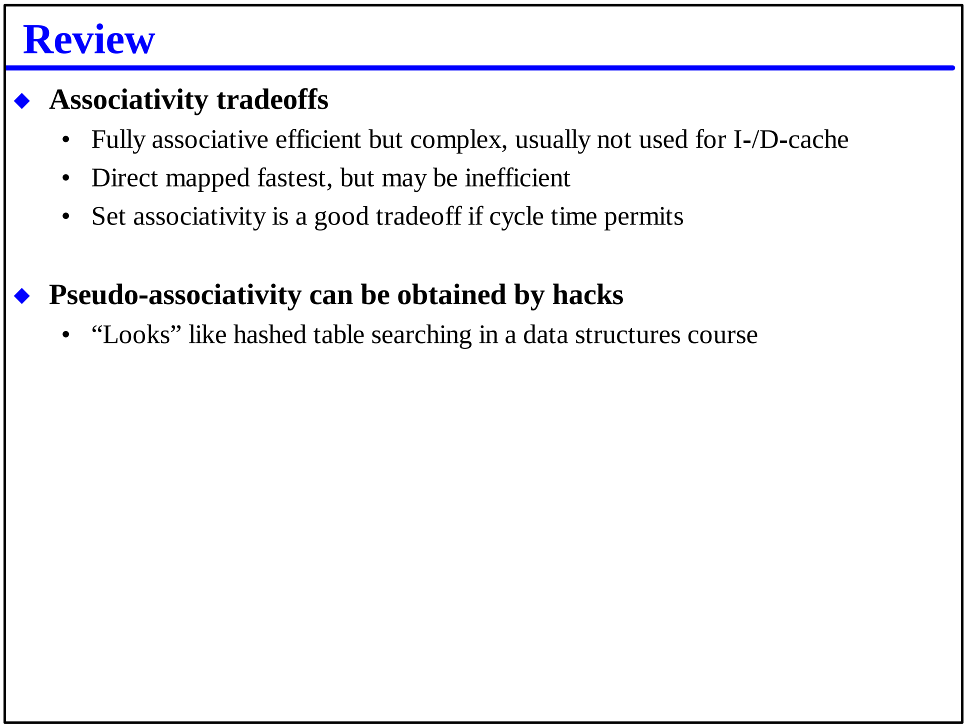
Associativity Rules of Thumb

- **“Ideally, associativity should be in range of 4-16”** (Cragon pg. 27)
- **“The miss rate of a direct-mapped cache of size X is about the same as a 2- to 4-way set associative cache of size X/2.”** (Hennessy & Patterson, pg. 391)
- **Single-level caches are made too slow by set-associativity; direct mapped is better for L1 caches. L2 caches should be, say, 8-way set associative.** (Przybylski section 5.3.3)
- **Conclusion -- mild set associativity is a win if:**
  - You can spare the cycle time (*e.g.*, L2 cache and beyond)
  - You can spend the power/area to make the tag fetch and compare faster than data access
  - Signal delays are probably an important factor in deciding associativity (*e.g.*, if pressed for space, might put tags on-chip and data off-chip)

Associativity In Recent Processors

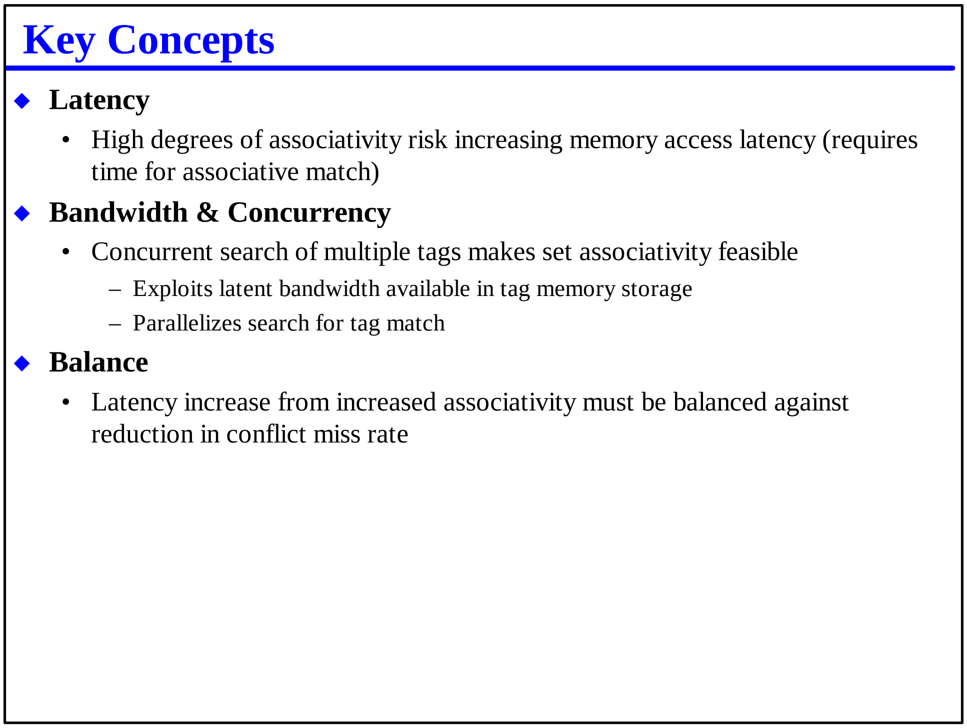
- **Alpha 21164**
  - Direct mapped L1
  - 3-way set associative L2
  - Direct mapped L3
  - Fully associative D-TLB (64 entries) & I-TLB (48 entries)
- **Pentium Pro**
  - 2-way set associative L1 D-cache; 4-way L1 I-cache
  - 4-way set associative L2 cache
  - 4-way set associative D-TLB & I-TLB (64 entries each)
- **MIPS R-8000**
  - Direct mapped L1 caches; 4-way set associative L2 cache
  - 384-entry(!) TLB; 3-way set associative
- **Power PC 604**
  - 4-way set associative L1 caches
  - 2-way set associative D-TLB & I-TLB (128 entries each)

REVIEW



Review

- **Associativity tradeoffs**
  - Fully associative efficient but complex, usually not used for I-/D-cache
  - Direct mapped fastest, but may be inefficient
  - Set associativity is a good tradeoff if cycle time permits
- **Pseudo-associativity can be obtained by hacks**
  - “Looks” like hashed table searching in a data structures course



Key Concepts

- **Latency**
  - High degrees of associativity risk increasing memory access latency (requires time for associative match)
- **Bandwidth & Concurrency**
  - Concurrent search of multiple tags makes set associativity feasible
    - Exploits latent bandwidth available in tag memory storage
    - Parallelizes search for tag match
- **Balance**
  - Latency increase from increased associativity must be balanced against reduction in conflict miss rate