L14-CS8421-10-8-08

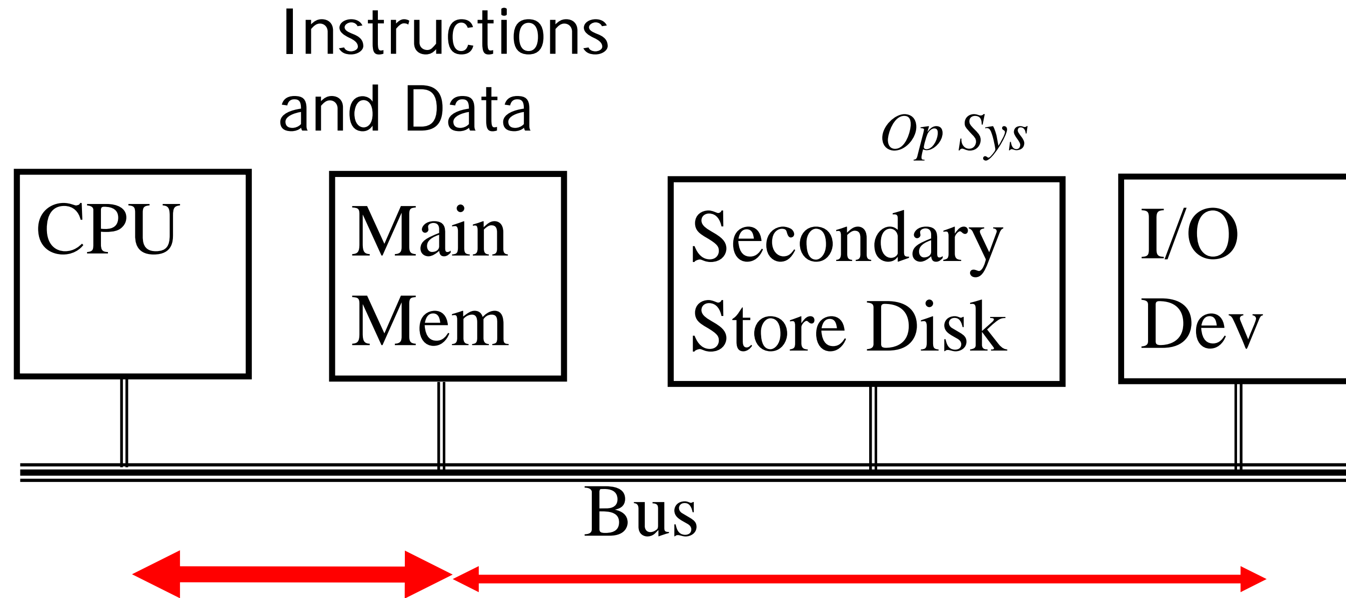
Cache Part 1

CS8421
Computing Systems
Dr. Ken Hoganson

Class
Will
Start
Momentarily…

Copyright © 2004,2006, Dr. Ken Hoganson

Von Neuman Architecture

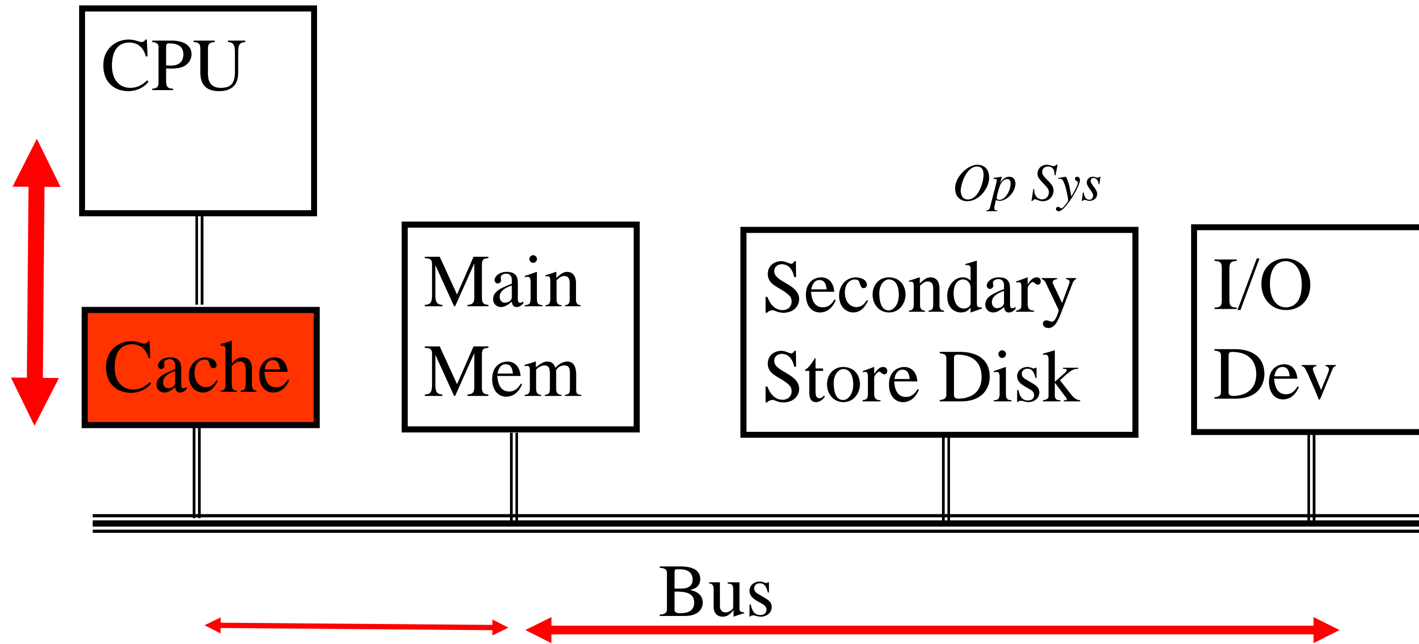


Copyright © 2004,2006, Dr. Ken Hoganson

The Need for Cache

- Memory performance has not kept pace with processor performance
- The Von-Neuman architecture requires multiple memory accesses for many instructions
- The use of pipelines (covered later) to increase the number of instructions processed per unit time, further increases the memory/bus bandwidth
- bandwidth is often used to talk about the communication requirements in terms of bits per second, in addition to its more traditional sense of a range of frequencies

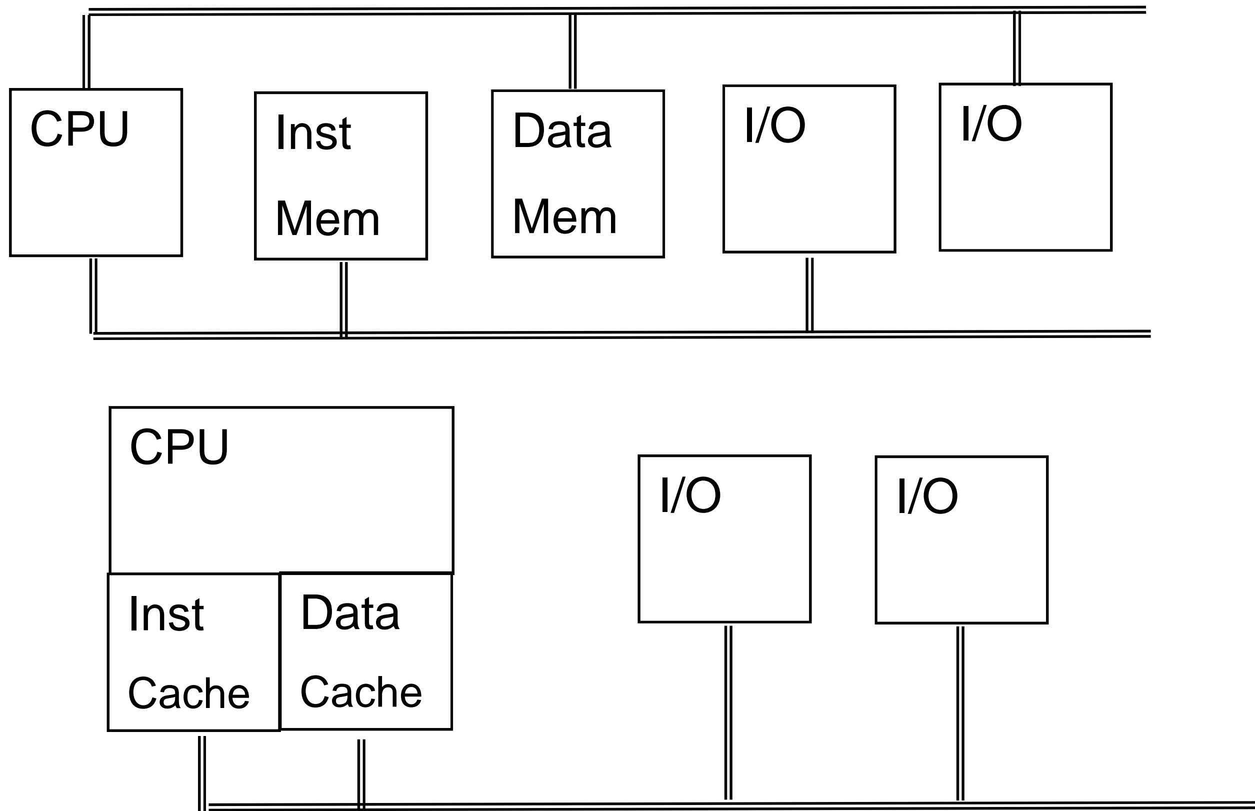
Simple Cache



Copyright © 2004,2006, Dr. Ken Hoganson

Harvard Architecture

- Allows fetch of instructions and data simultaneously
- Separate caches, one for data, one for instructions



Copyright © 2004,2006, Dr. Ken Hoganson

Principles behind Caching

- **Temporal Locality**
  - the most recently accessed memory locations are more likely to be accessed again in the future than are less recently accessed memory locations
- **Spatial Locality**
  - the most recently accessed blocks of a program are more likely to be accessed again than are less recently accessed blocks of a program

- What characteristics of programs yield these observations?
  - Programs are sequential - next instruction is the most likely to be needed instruction (spacial)
  - Programs contain loops - repeat same instructions in same areas (both principles)

Cache Principles

- These two principles means that it makes sense to store the most likely future accessed memory locations in expensive high speed memory.
- Memory is cached in blocks - 512 bytes or more per block is common
- The two principles say that the most recently accessed memory block is very likely to be accessed again, so store it in the high speed cache
- less likely to be accessed blocks will be stored in memory (and perhaps even on secondary storage in a virtual memory system - more later)

Copyright © 2004,2006, Dr. Ken Hoganson

Cache

- Cache –
  - increased overall speed of bus/memory system
  - some percentage of needed memory locations will be found in the cache,
  - so the average time to access (also called latency) will drop.
- The cache is located on the processor side of the bus,
  - so accesses that are satisfied by the cache do not need the bus
  - reducing BW requirements.

Copyright © 2004,2006, Dr. Ken Hoganson

Storage Latency

- Cache can be used to “hide the latency” of the bus/memory system, as we have been discussing.
- Cache can also be used to “hide the latency” of disk accesses (which are much slower than memory).
- Cache can also be used to “hide the latency” of distributed processing over a network (caching web pages).

- Latency hierarchy:
  - cache
  - memory
  - hard disk
  - networked storage

Copyright © 2004,2006, Dr. Ken Hoganson

Cache Performance

- Cache improves the average performance of a system
- The accesses or requests that are satisfied by the cache are termed "hits" in the cache
- The accesses or requests that are not satisfied by the cache (have to go out to memory or other storage) are termed "cache misses"
- Accesses or requests are satisfied by one or the other method

Prop(hit) + Prob(miss) = 1

Memory with Cache Performance =
Prob(hit)*Time(cache) + (1-Prob(hit))*Time(miss)

Copyright © 2004,2006, Dr. Ken Hoganson

Cache Performance Example

Example:
- Cache access time = 5 nanoseconds
- Memory access time = 50 nanosecond
- Cache hit rate = 90% (0.9)

Ave Latency = 0.9(5) + 0.1(50) = 4.5 + 5 = 9.5ns
- Much better than memory, almost as good as the cache!

Are high hit rates reasonable?

Yes it turns out, often in the high 90s!
- 0.95(5) + 0.05(50) = 4.75 + 2.5 = 7.25ns
- 0.99(5) + 0.01(50) = 4.95 + 0.5 = 5.45ns
- 0.50(5) + 0.50(50) = 2.50 + 25 = 27.50

Copyright © 2004,2006, Dr. Ken Hoganson

Cache Effectiveness and Speed Disparity

The effectiveness of caching depends on the speed differential between the cache and the memory - large differences, large payoffs

Cache access time = 5 ns, Memory access time = 50 ns

- Cache hit rate = 95% (0.95)
- Ave Mem Latency 0.95(5) + 0.05(50) = 4.75 + 2.5 = 7.25ns

Large disparity: 50ns to 7.25ns == 85.5% improvement

Cache access time = 5 ns, Memory access time = 10 ns

- Cache hit rate = 95% (0.95)
- Ave Mem Latency 0.95(5) + 0.05(10) = 4.75 + 0.5 = 5.25ns

Smaller disparity: 10ns to 5.25ns == 47.5% improvement

Other things to look at later:

- what about cache writes
- how to manage blocks in cache (block replacement)

Copyright © 2004,2006, Dr. Ken Hoganson

Simple Model So Far

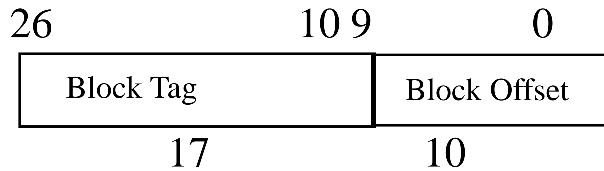
Caveats:

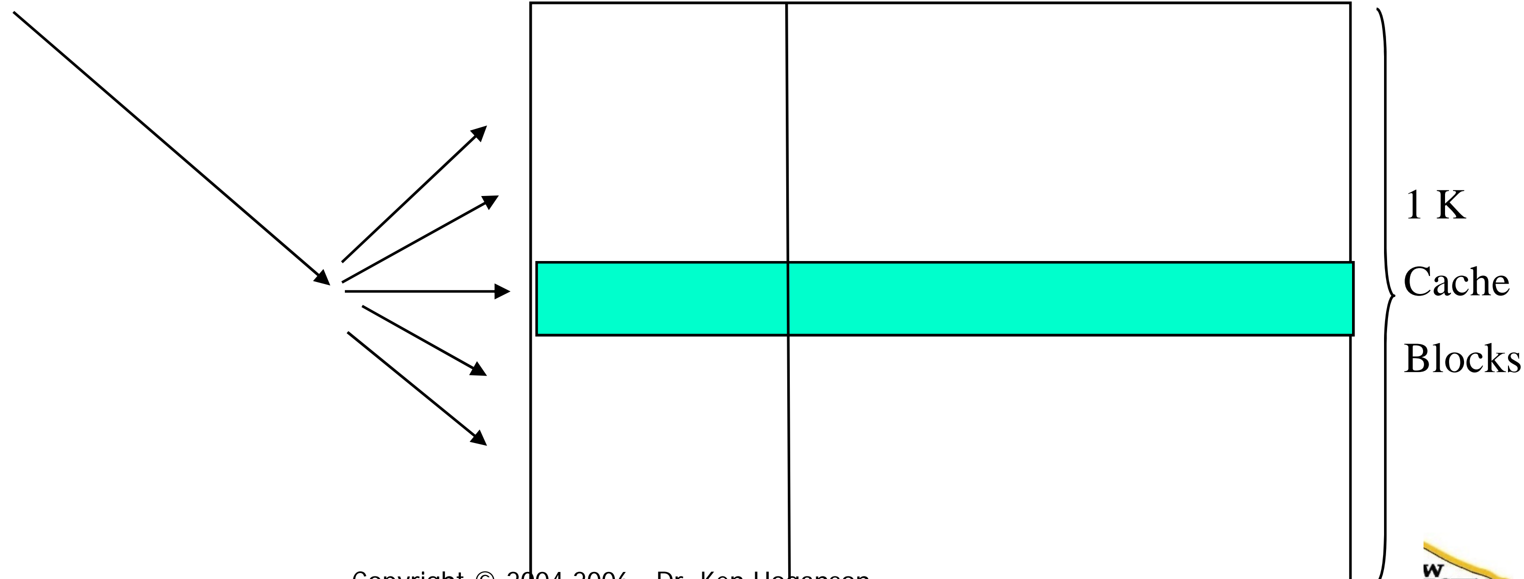
- The calculations we have been doing are somewhat simplified:
  - Memory speed is time to read a single value
  - What about blocks of memory? - longer load times
  - Cache Writes? - write to cache
  - Finding a cache block, or freeing a spot for the new block
- The "miss penalty" could be much worse than our simplified analysis.

Copyright © 2004,2006, Dr. Ken Hoganson

Fully-Associative Cache

- Blocks can go anywhere in the cache
- Example: 128Mbytes of Mem (2^{27}), 1MB of cache (2^{20}), 1KB block size (2^{10}),
- Number of blocks in cache = $2^{20} / 2^{10} = 2^{10} = 1K$
- Fraction of memory in cache = 1/128 = 0.78%
- Disadvantage: need to compare ALL tags with the address, expensive hardware to locate a block - parallel compare all tags is too expensive for large cache

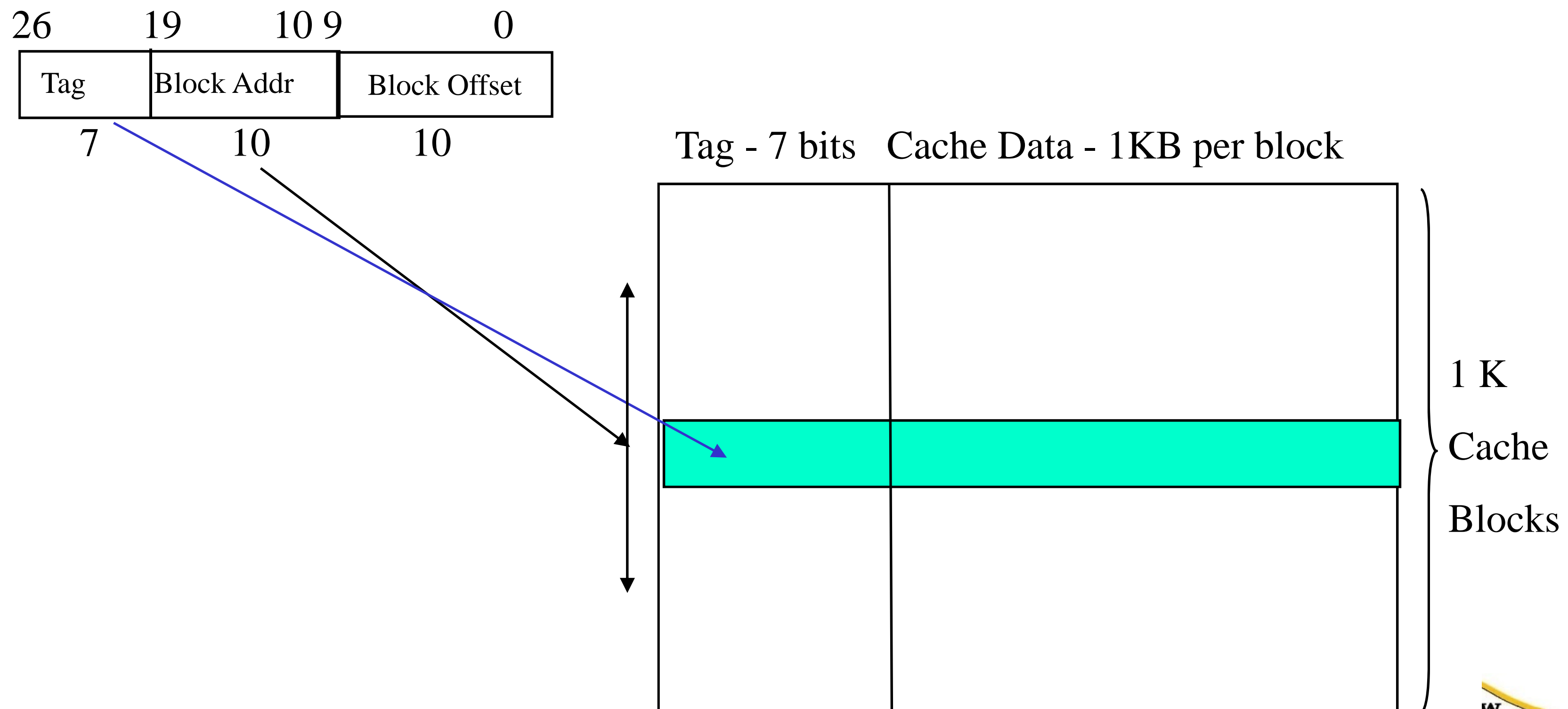




Copyright © 2004,2006, Dr. Ken Hoganson

Direct-Mapped Cache

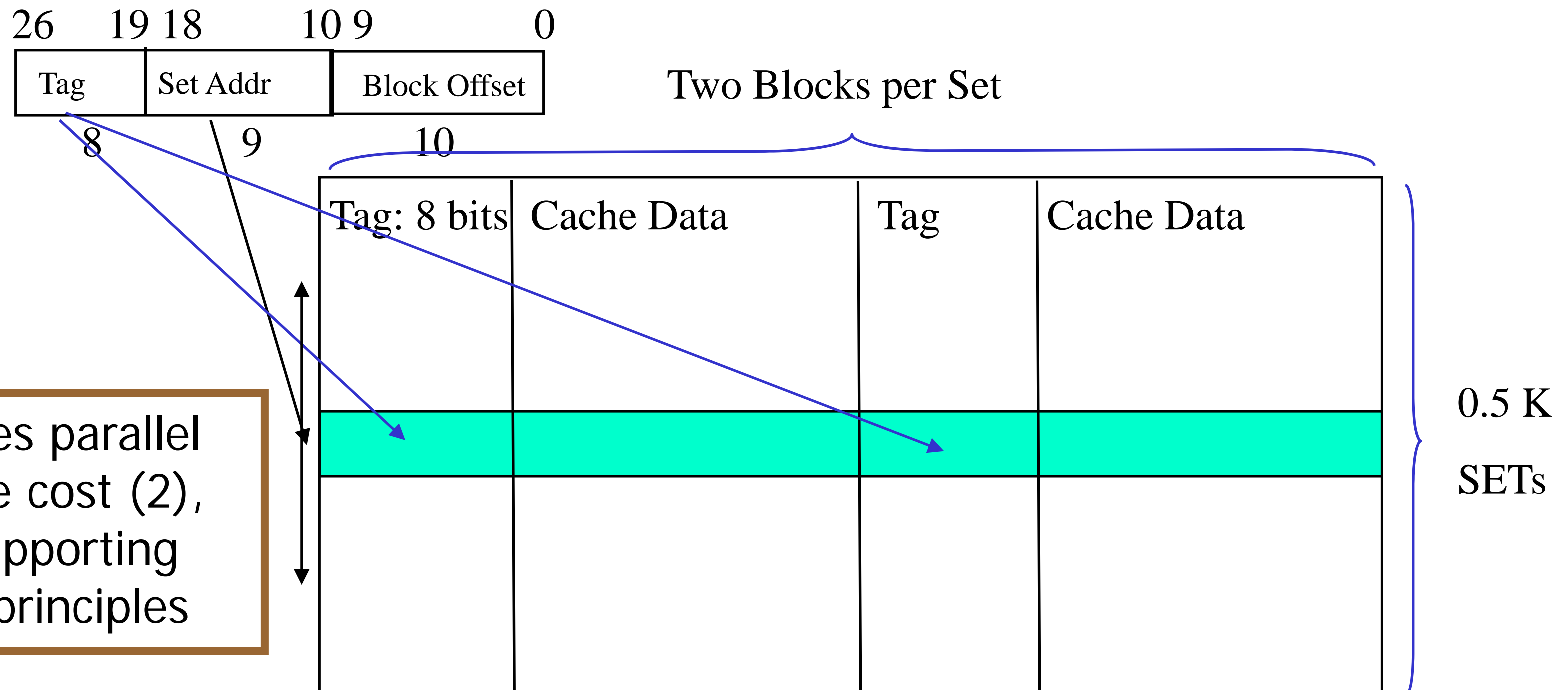
- Blocks can be cached in **one location only**
- Example: 128Mbytes of Mem (2^{27}), 1MB of cache (2^{20}), 1KB block size (2^{10}),
- Number of blocks in cache = $2^{20} / 2^{10} = 2^{10} = 1K$
- Fraction of memory in cache = 1/128 = 0.78%
- Must compare Tag to see which block is cached (many go to same location)
- 128 blocks map to the same cache location in this example
- Restrictive block location undermines locality principles



Copyright © 2004,2006, Dr. Ken Hoganson

Set-Associative Cache

- Blocks can go anywhere within a particular **SET**
- A set is a grouping of blocks - **2 way** set associative means **2 blocks make a set**
- Example: 128Mbytes of Mem (2^{27}), 1MB of cache (2^{20}), 1KB block size (2^{10}),
- Number of blocks in cache = $2^{20} / 2^{10} = 2^{10} = 1K$, number of sets is $2^9 = 0.5$ K
- Fraction of memory in cache = 1/128 = 0.78%
- Must compare both **tags** to see which block is cached (many go to same location)
- 128 blocks map to the same set, but two of those in set at a time



Minimizes parallel compare cost (2), while supporting locality principles

Copyright © 2004,2006, Dr. Ken Hoganson

Cache Writes

Writes of memory in a cached system pose an interesting problem

- Are the writes themselves cached, and written back later?
  - **WRITE-BACK**
  - A crash would lose data (caching disk in memory)
  - Best performance - do not wait for slow storage, write to cache is fast.
  - Write back to slower storage when system bandwidth is available

OR

- Write information through to the memory (or disk if caching disk)
  - **WRITE -THROUGH**
  - Slower, must wait for slower memory (or wait for disk)
  - Safer, changed data goes to disk - non-volatile

Copyright © 2004,2006, Dr. Ken Hoganson

Cache Block Replacement

- Direct-Mapped
  - Not an issue, since each block can go only one place in the cache, and overwrites current contents (must force a write if using Write-Back)
- Fully-Associate
  - Which block to replace?
  - Perhaps Least Recently Used (LRU)
    - How to know which was LRU?
    - Track accesses to the block: reference bits, exponential averaging?
    - Random? - Actually works reasonably well!
- Set-Associative
  - Replace the LRU block within the set

Copyright © 2004,2006, Dr. Ken Hoganson

End of Lecture

End

Of

Today's

Lecture.

Copyright © 2004,2006, Dr. Ken Hoganson

Blank Slide

Copyright © 2004,2006, Dr. Ken Hoganson