

inst.eecs.berkeley.edu/~cs61c  
**CS61C : Machine Structures**

**Lecture #22**  
**Caches II**



**CPS**  
**today!**

**2005-11-16**

There is **one** handout  
today at the front and  
back of the room!

**Lecturer PSOE, new dad Dan Garcia**

**[www.cs.berkeley.edu/~ddgarcia](http://www.cs.berkeley.edu/~ddgarcia)**

**IBM announces a 3D TV video system! ⇒**

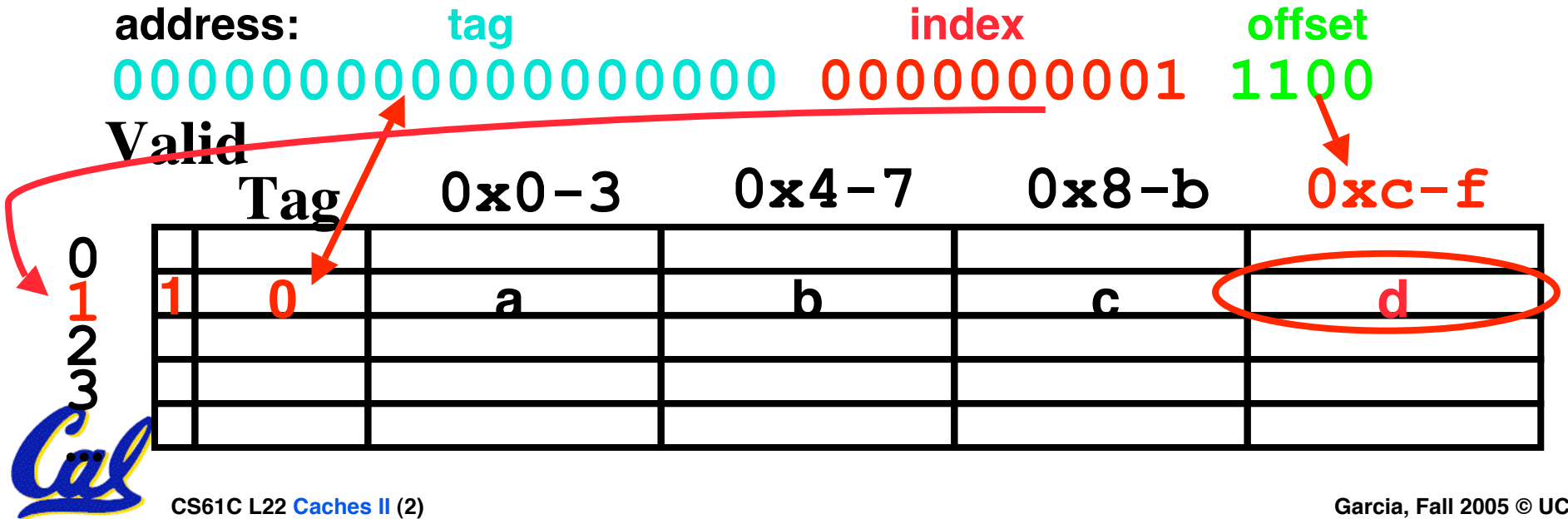
**A new video system to  
work with DLP televisions will be  
available soon for < \$1K. The system  
processes at 144 frames per second; imagine 3D  
video games or watching 3D NFL games! Cool.**



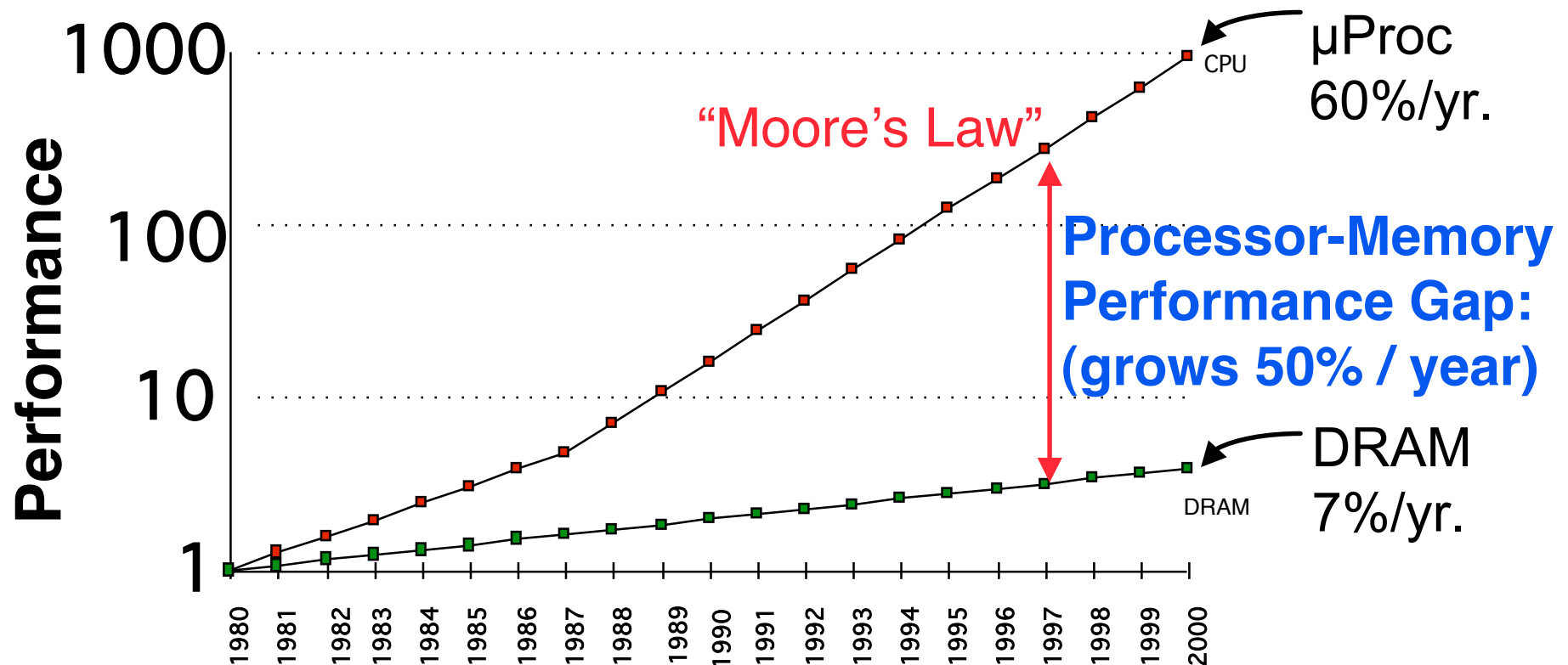
**[www.physorg.com/news8113.html](http://www.physorg.com/news8113.html)**

# Caches Review...

- **Mechanism for transparent movement of data among levels of a storage hierarchy**
  - set of address/value bindings
  - address  $\Rightarrow$  index to set of candidates
  - compare desired address with tag
  - service hit or miss
    - load new block and binding on miss



# Review: Why We Use Caches



- 1989 first Intel CPU with cache on chip
- 1998 Pentium III has two levels of cache on chip



# Block Size Tradeoff (1/3)

---

- **Benefits of Larger Block Size**
  - **Spatial Locality**: if we access a given word, we're likely to access other nearby words soon
  - **Very applicable with Stored-Program Concept**: if we execute a given instruction, it's likely that we'll execute the next few as well
  - **Works nicely in sequential array accesses too**



## Block Size Tradeoff (2/3)

---

- Drawbacks of Larger Block Size
  - Larger block size means **larger miss penalty**
    - on a miss, takes longer time to load a new block from next level
  - If block size is too big relative to cache size, then there are too few blocks
    - Result: miss rate goes up
- In general, minimize **Average Memory Access Time (AMAT)**
  - = Hit Time
  - + Miss Penalty x Miss Rate



## Block Size Tradeoff (3/3)

---

- **Hit Time** = time to find and retrieve data from current level cache
- **Miss Penalty** = average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)
- **Hit Rate** = % of requests that are found in current level cache
- **Miss Rate** =  $1 - \text{Hit Rate}$



## Extreme Example: One Big Block

---

Valid Bit



Tag



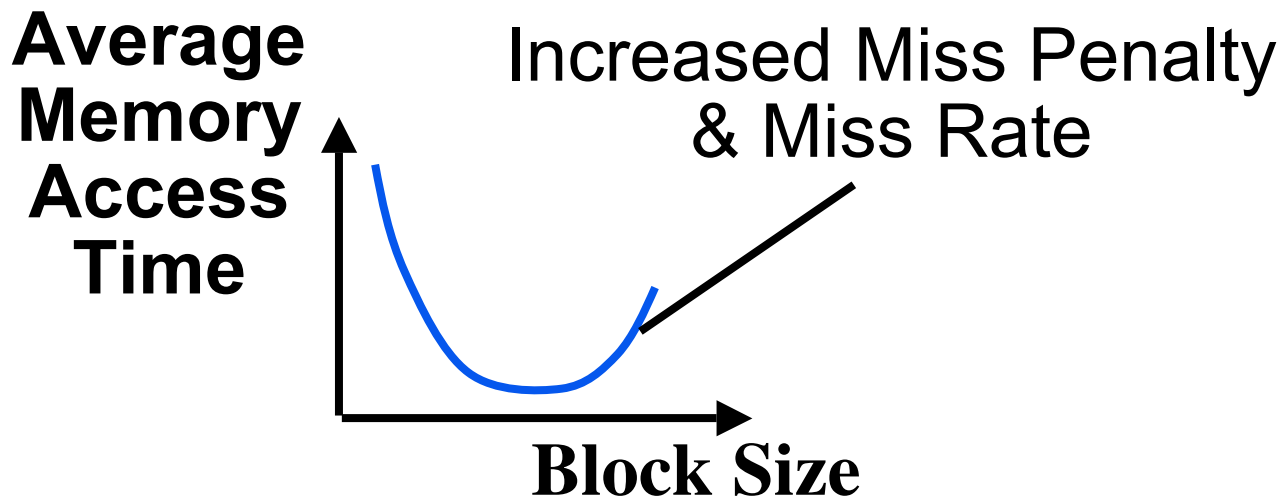
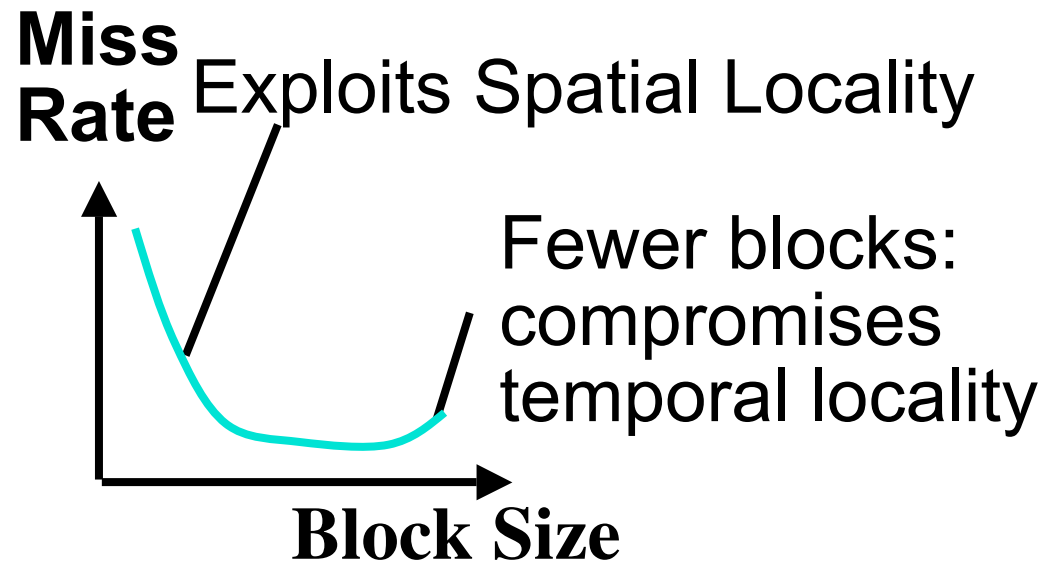
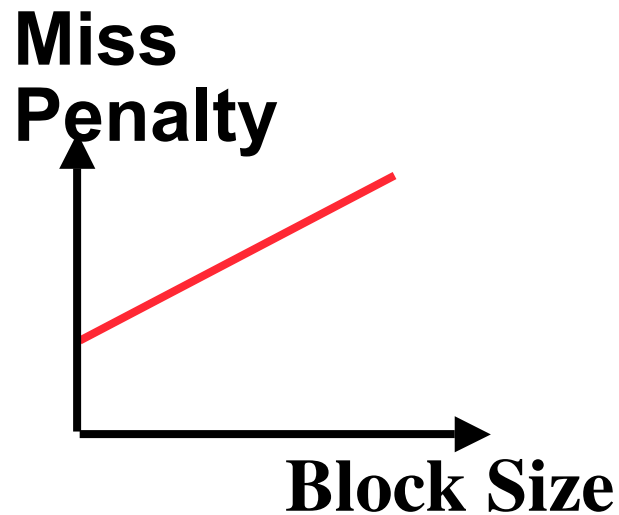
Cache Data



- Cache Size = 4 bytes    Block Size = 4 bytes
  - Only **ONE** entry in the cache!
- If item accessed, likely accessed again soon
  - But unlikely will be accessed again immediately!
- The next access will likely to be a miss again
  - Continually loading data into the cache but discard data (force out) before use it again
  - Nightmare for cache designer: **Ping Pong Effect**



# Block Size Tradeoff Conclusions





# Types of Cache Misses (1/2)

---

- “Three Cs” Model of Misses
  - 1st C: Compulsory Misses
    - occur when a program is first started
    - cache does not contain any of that program’s data yet, so misses are bound to occur
    - can’t be avoided easily, so won’t focus on these in this course



# Types of Cache Misses (2/2)

---

- **2nd C: Conflict Misses**

- miss that occurs because two distinct memory addresses map to the same cache location
- two blocks (which happen to map to the same location) can keep overwriting each other
- big problem in direct-mapped caches
- how do we lessen the effect of these?

- **Dealing with Conflict Misses**

- **Solution 1: Make the cache size bigger**
  - Fails at some point
- **Solution 2: Multiple distinct blocks can fit in the same cache Index?**



# Fully Associative Cache (1/3)

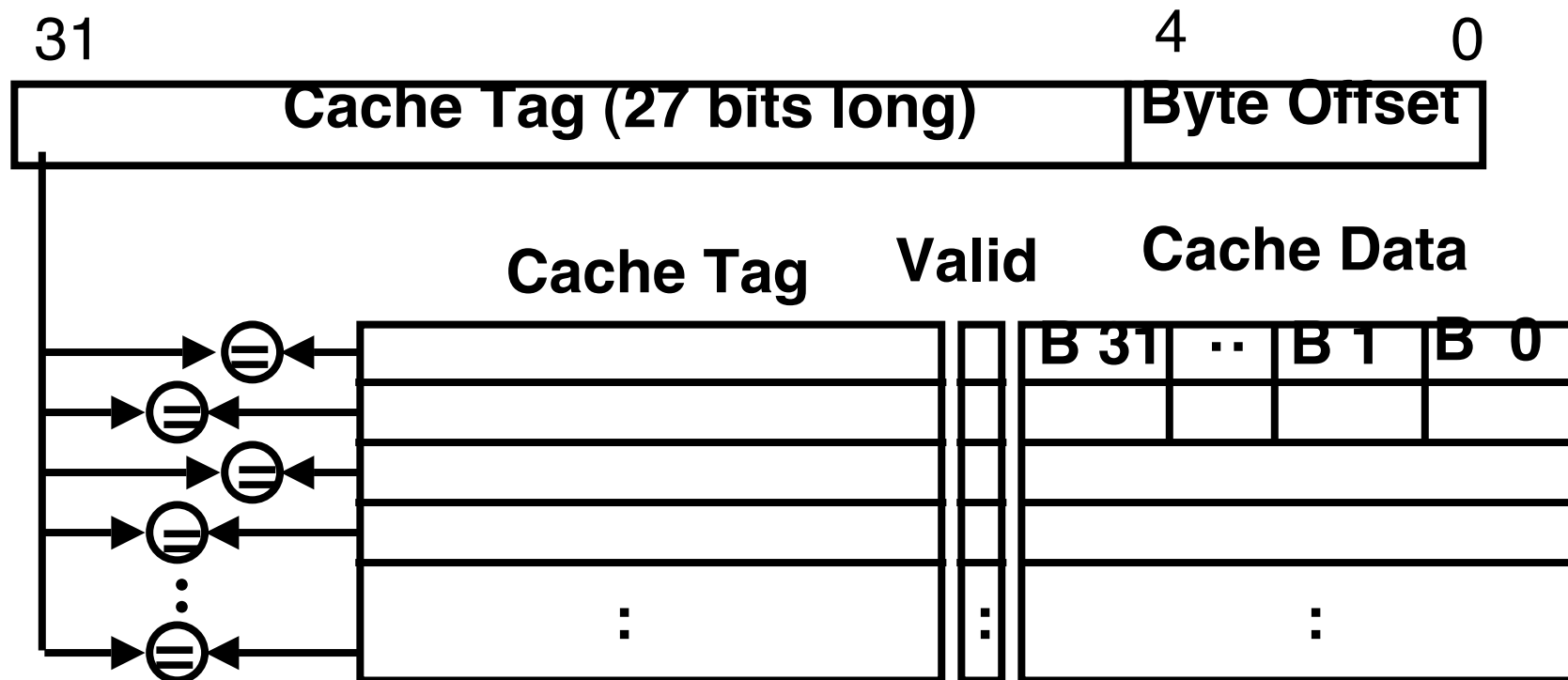
---

- **Memory address fields:**
  - **Tag: same as before**
  - **Offset: same as before**
  - **Index: non-existent**
- **What does this mean?**
  - **no “rows”: any block can go anywhere in the cache**
  - **must compare with all tags in entire cache to see if data is there**



## Fully Associative Cache (2/3)

- Fully Associative Cache (e.g., 32 B block)
  - compare tags in parallel



# Fully Associative Cache (3/3)

---

- **Benefit of Fully Assoc Cache**
  - No Conflict Misses (since data can go anywhere)
- **Drawbacks of Fully Assoc Cache**
  - Need hardware comparator for every single entry: if we have a 64KB of data in cache with 4B entries, we need 16K comparators: infeasible



# Third Type of Cache Miss

---

- Capacity Misses

- miss that occurs because the cache has a limited size
  - miss that would not occur if we increase the size of the cache
  - sketchy definition, so just get the general idea
- This is the primary type of miss for Fully Associative caches.



# N-Way Set Associative Cache (1/4)

---

- **Memory address fields:**
  - Tag: same as before
  - Offset: same as before
  - Index: points us to the correct “row” (called a **set** in this case)
- **So what’s the difference?**
  - each set contains multiple blocks
  - once we’ve found correct set, must compare with all tags in that set to find our data



# N-Way Set Associative Cache (2/4)

---

- **Summary:**

- **cache is direct-mapped w/respect to sets**
- **each set is fully associative**
- **basically N direct-mapped caches working in parallel: each has its own valid bit and data**





# N-Way Set Associative Cache (3/4)

---

- **Given memory address:**
  - Find correct set using Index value.
  - Compare Tag with all Tag values in the determined set.
  - If a match occurs, hit!, otherwise a miss.
  - Finally, use the offset field as usual to find the desired data within the block.



# **N-Way Set Associative Cache (4/4)**

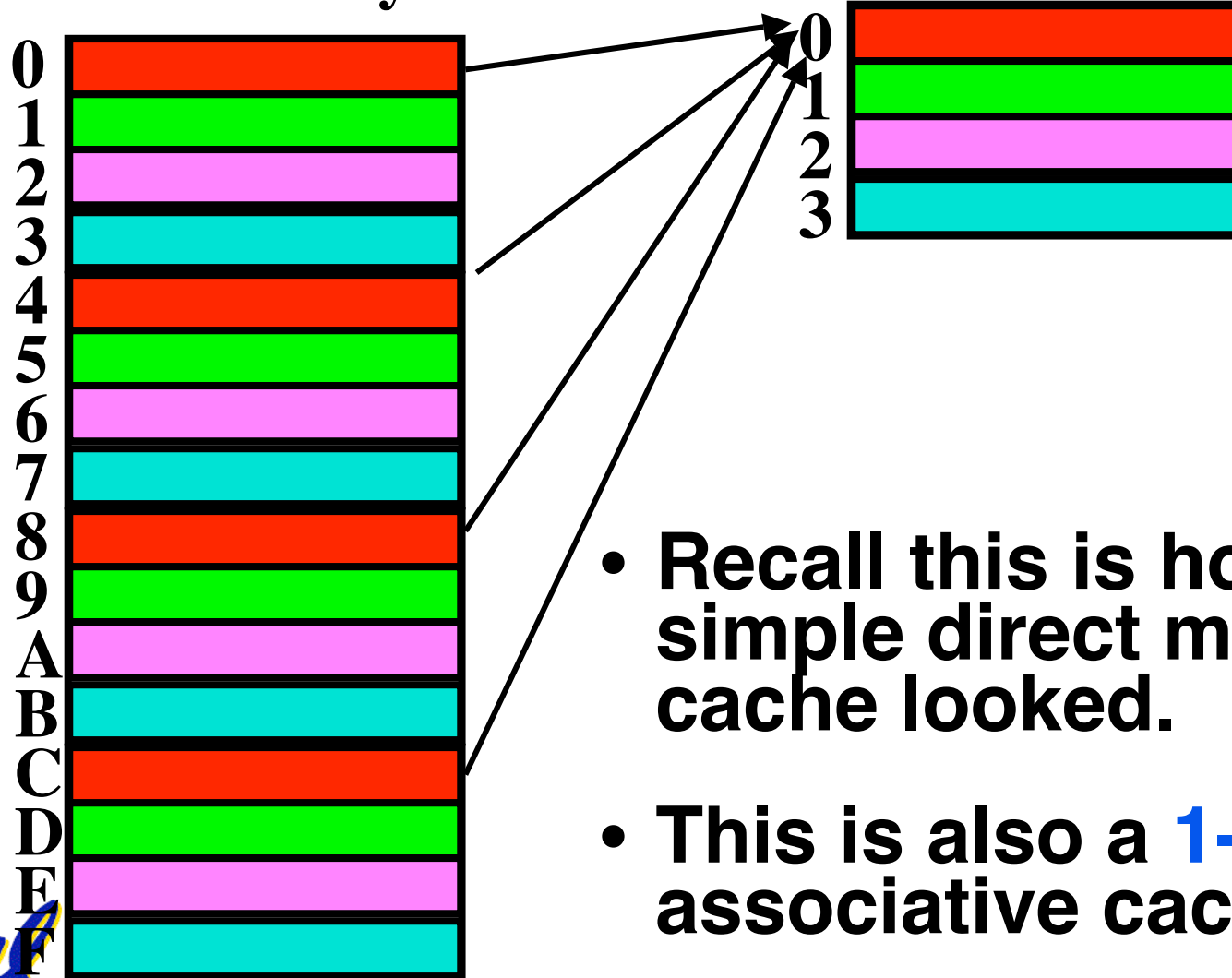
---

- **What's so great about this?**
  - **even a 2-way set assoc cache avoids a lot of conflict misses**
  - **hardware cost isn't that bad: only need N comparators**
- **In fact, for a cache with M blocks,**
  - **it's Direct-Mapped if it's 1-way set assoc**
  - **it's Fully Assoc if it's M-way set assoc**
  - **so these two are just special cases of the more general set associative design**



# Associative Cache Example

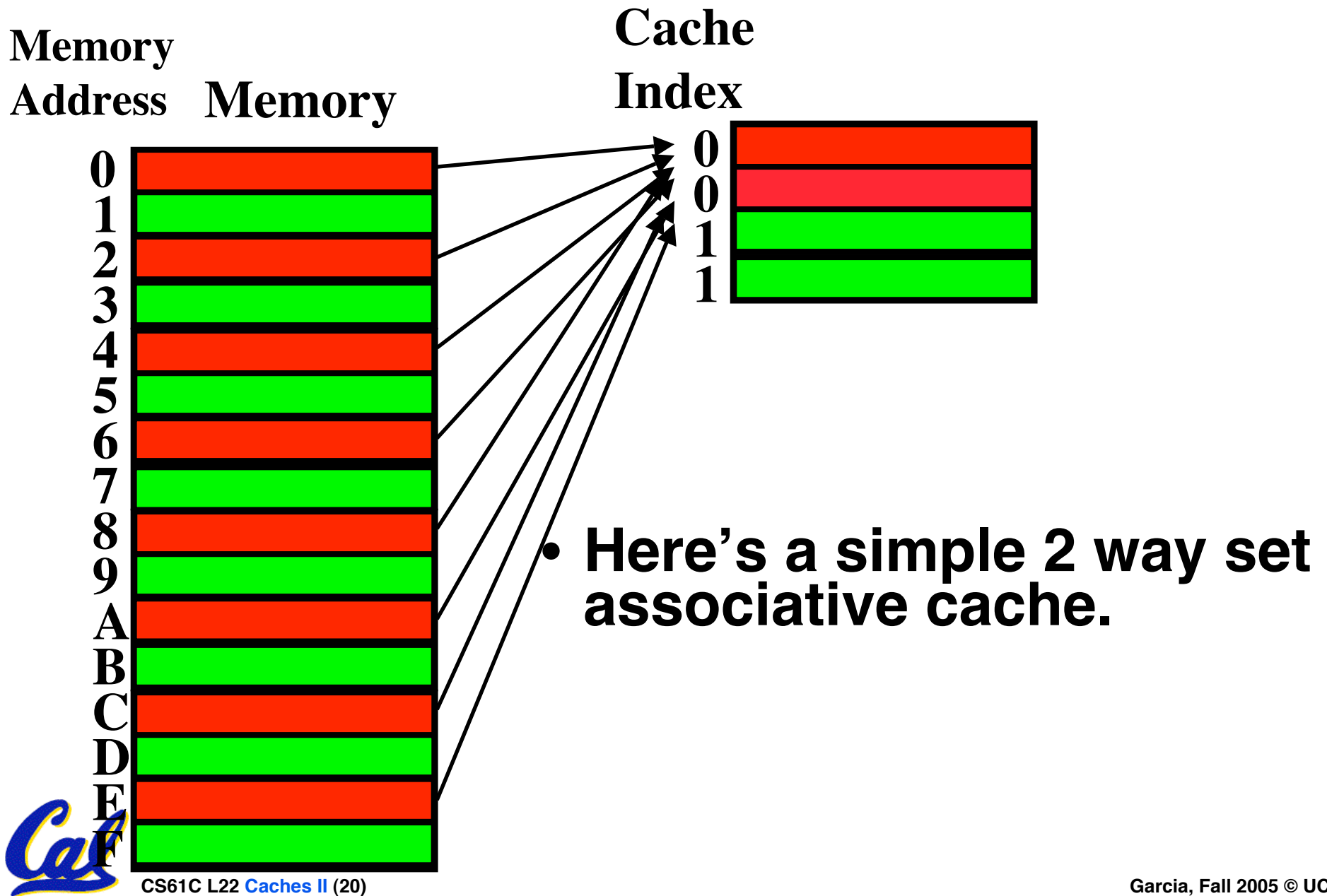
Memory Address      Memory      Cache Index      4 Byte Direct Mapped Cache



- Recall this is how a simple direct mapped cache looked.
- This is also a **1-way** set-associative cache!



# Associative Cache Example



## Block Replacement Policy (1/2)

---

- **Direct-Mapped Cache**: index completely specifies position which position a block can go in on a miss
- **N-Way Set Assoc**: index specifies a set, but block can occupy any position within the set on a miss
- **Fully Associative**: block can be written into any position
- **Question**: if we have the choice, where should we write an incoming block?



## Block Replacement Policy (2/2)

---

- If there are any locations with valid bit off (empty), then usually write the new block into the first one.
- If all possible locations already have a valid block, we must pick a **replacement policy**: rule by which we determine which block gets “cached out” on a miss.



# Block Replacement Policy: LRU

---

- **LRU (Least Recently Used)**

- Idea: cache out block which has been accessed (read or write) least recently
- Pro: temporal locality  $\Rightarrow$  recent past use implies likely future use: in fact, this is a very effective policy
- Con: with 2-way set assoc, easy to keep track (one LRU bit); with 4-way or greater, requires complicated hardware and much time to keep track of this



## Block Replacement Example

---

- We have a 2-way set associative cache with a four word *total* capacity and one word blocks. We perform the following word accesses (ignore bytes for this problem):

0, 2, 0, 1, 4, 0, 2, 3, 5, 4

How many hits and how many misses will there be for the LRU block replacement policy?





# Block Replacement Example: LRU

- Addresses 0, 2, 0, 1, 4, 0, ...

0: miss, bring into set 0 (loc 0)

2: miss, bring into set 0 (loc 1)

0: hit

1: miss, bring into set 1 (loc 0)

4: miss, bring into set 0 (loc 1, replace 2)

0: hit

	loc 0	loc 1
set 0	0	iru
set 1		
set 0	iru 0	2
set 1		
set 0	0	iru 2
set 1		
set 0	0	iru 2
set 1	1	iru
set 0	iru 0	4
set 1	1	iru
set 0	0	iru 4
set 1	1	iru



# Administrivia

---

- **Do your reading! VM is coming up, and it's shown to be hard for students!**
- **Any other announcements?**



# Big Idea

---

- How to choose between associativity, block size, replacement policy?
- Design against a performance model
  - Minimize: *Average Memory Access Time*  
$$= \text{Hit Time} + \text{Miss Penalty} \times \text{Miss Rate}$$
  - influenced by technology & program behavior
  - Note: Hit Time encompasses Hit Rate!!!
- Create the illusion of a memory that is large, cheap, and fast - on average



# Example

---

- **Assume**
  - Hit Time = 1 cycle
  - Miss rate = 5%
  - Miss penalty = 20 cycles
  - Calculate AMAT...
- **Avg mem access time**
  - =  $1 + 0.05 \times 20$
  - = 1 + 1 cycles
  - = 2 cycles



# Ways to reduce miss rate

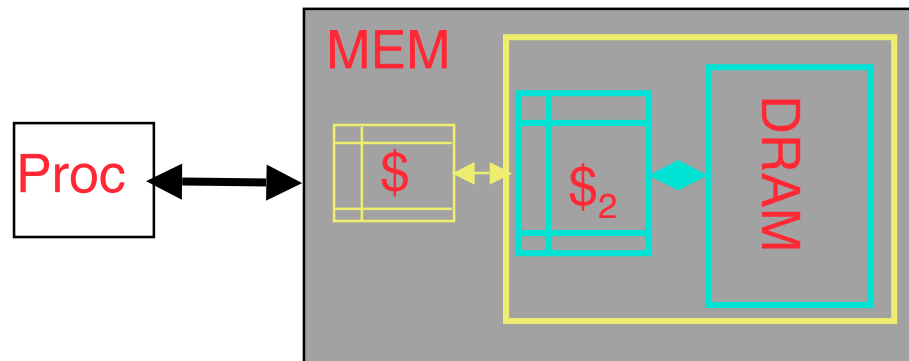
---

- **Larger cache**
  - limited by cost and technology
  - hit time of first level cache  $<$  cycle time
- **More places in the cache to put each block of memory – associativity**
  - **fully-associative**
    - any block any line
  - **N-way set associated**
    - N places for each block
    - direct map:  $N=1$



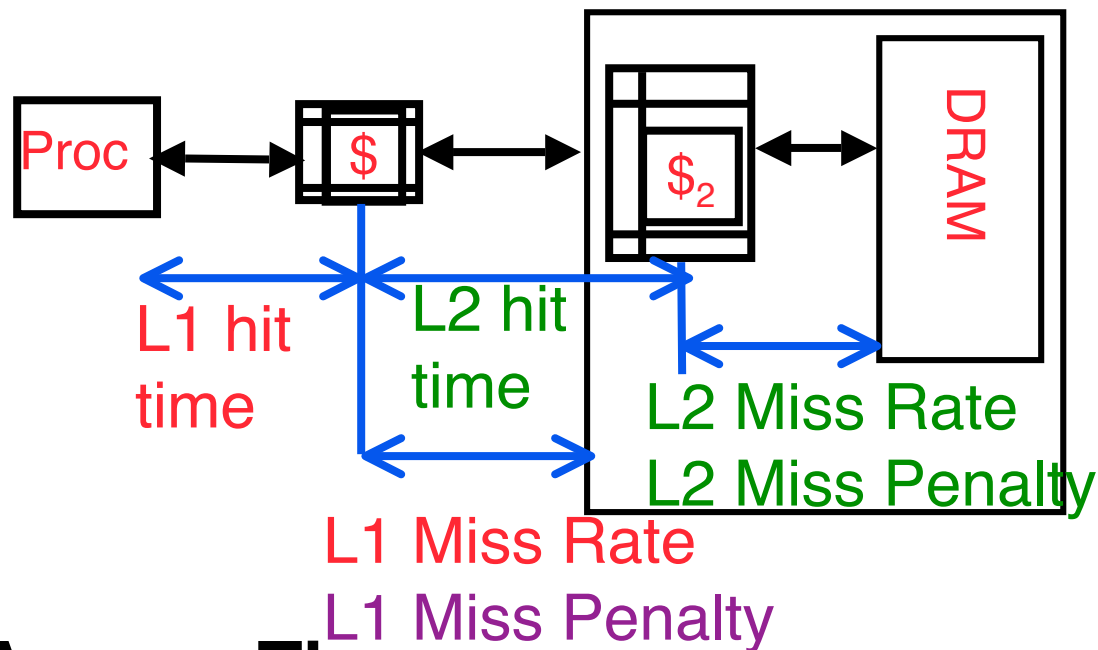
# Improving Miss Penalty

- When caches first became popular, Miss Penalty  $\sim 10$  processor clock cycles
- Today 2400 MHz Processor (0.4 ns per clock cycle) and 80 ns to go to DRAM  
 $\Rightarrow$  **200 processor clock cycles!**



**Solution: another cache between memory and the processor cache: Second Level (L2) Cache**

# Analyzing Multi-level cache hierarchy



**Avg Mem Access Time =**

$$\frac{\text{L1 Hit Time} + \text{L1 Miss Rate} * \text{L1 Miss Penalty}}{\text{L1 Miss Penalty} =}$$

**L1 Miss Penalty =**

$$\frac{\text{L2 Hit Time} + \text{L2 Miss Rate} * \text{L2 Miss Penalty}}{\text{Avg Mem Access Time} =}$$

**Avg Mem Access Time =**

$$\text{L1 Hit Time} + \text{L1 Miss Rate} *$$

$$(\text{L2 Hit Time} + \text{L2 Miss Rate} * \text{L2 Miss Penalty})$$

# Typical Scale

---

- **L1**
  - **size: tens of KB**
  - **hit time: complete in one clock cycle**
  - **miss rates: 1-5%**
- **L2:**
  - **size: hundreds of KB**
  - **hit time: few clock cycles**
  - **miss rates: 10-20%**
- **L2 miss rate is fraction of L1 misses that also miss in L2**
  - **why so high?**





## Example: with L2 cache

---

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L2 Hit Time = 5 cycles
  - L2 Miss rate = 15% (% L1 misses that miss)
  - L2 Miss Penalty = 200 cycles
- L1 miss penalty =  $5 + 0.15 * 200 = 35$
- Avg mem access time =  $1 + 0.05 \times 35$   
= 2.75 cycles



## Example: without L2 cache

---

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L1 Miss Penalty = 200 cycles
- Avg mem access time =  $1 + 0.05 \times 200$   
= 11 cycles
- 4x faster with L2 cache! (2.75 vs. 11)



# What to do on a write hit?

---

- Write-through

- update the word in cache block and corresponding word in memory

- Write-back

- update word in cache block
- allow memory word to be “stale”

⇒ add ‘dirty’ bit to each block indicating that memory needs to be updated when block is replaced

⇒ OS flushes cache before I/O...

- Performance trade-offs?



# Peer Instructions

---

1. In the last 10 years, the gap between the access time of DRAMs & the cycle time of processors has decreased. (I.e., is closing)
2. A 2-way set-associative cache can be outperformed by a direct-mapped cache.
3. Larger block size  $\Rightarrow$  lower miss rate

	ABC
1 :	<b>FFF</b>
2 :	<b>FFT</b>
3 :	<b>FTF</b>
4 :	<b>FTT</b>
5 :	<b>TFF</b>
6 :	<b>TFT</b>
7 :	<b>TTF</b>
8 :	<b>TTT</b>



# Peer Instructions Answer

1. That was one of the motivation for caches in the first place -- that the memory gap is big and widening.
  2. Sure, consider the caches from the previous slides with the following workload: 0, 2, 0, 4, 2  
2-way: 0m, 2m, 0h, 4m, 2m; DM: 0m, 2m, 0h, 4m, 2h
  3. Larger block size  $\Rightarrow$  lower miss rate, true until a certain point, and then the ping-pong effect takes over
1. In the last 10 years, the gap between the access time of DRAMs & the cycle time of processors has decreased. (I.e., is closing)
  2. A 2-way set-associative cache can be outperformed by a direct-mapped cache.
  3. Larger block size  $\Rightarrow$  lower miss rate

	ABC
1 :	FFF
2 :	FFT
3 :	FTF
4 :	FTT
5 :	TFF
6 :	TFT
7 :	TTF
8 :	TTT



# Generalized Caching

---

- **We've discussed memory caching in detail. Caching in general shows up over and over in computer systems**
  - Filesystem cache
  - Web page cache
  - Game Theory databases / tablebases
  - Software memoization
  - Others?
- **Big idea: if something is expensive but we want to do it repeatedly, do it once and cache the result.**



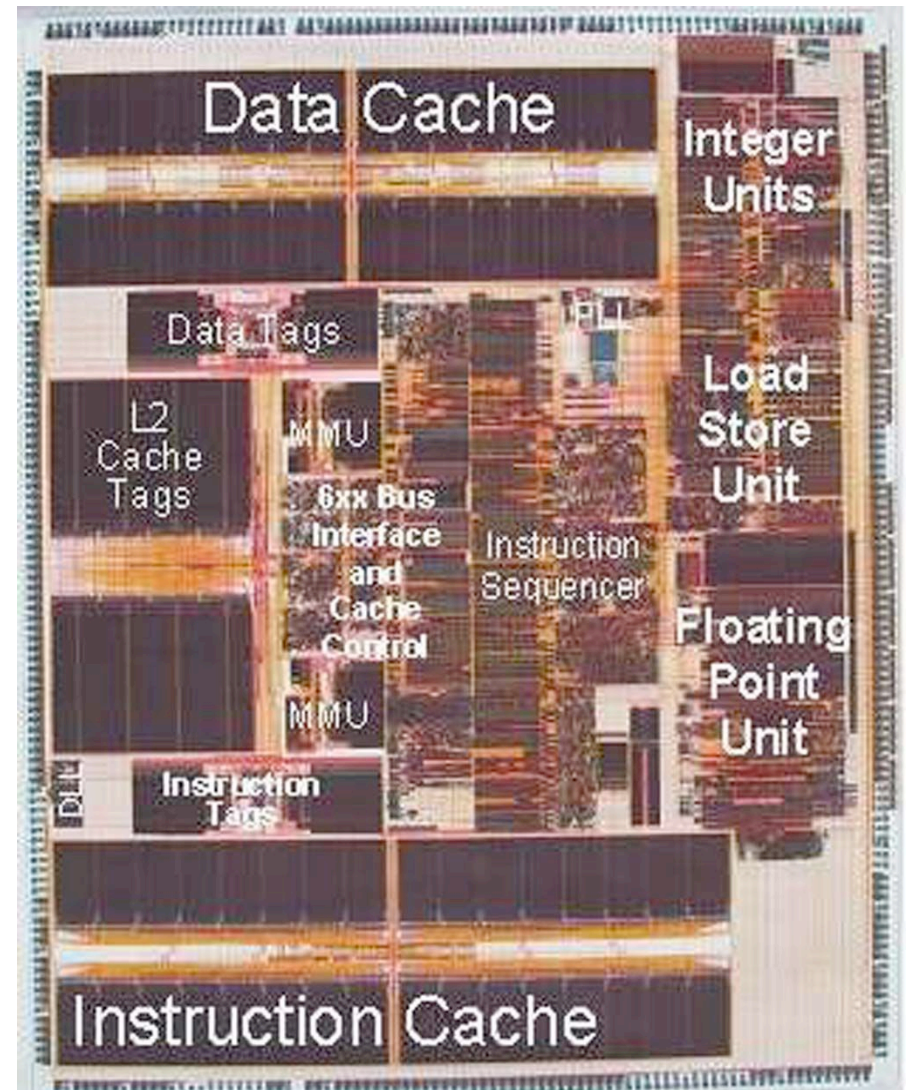
# An actual CPU -- Early PowerPC

- **Cache**

- 32 KiByte Instructions and 32 KiByte Data L1 caches
- External L2 Cache interface with integrated controller and cache tags, supports up to 1 MiByte external L2 cache
- Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)

- **Pipelining**

- Superscalar (3 inst/cycle)
- 6 execution units (2 integer and 1 double precision IEEE floating point)



# Cache Things to Remember

---

- Caches are NOT mandatory:
  - Processor performs arithmetic, memory stores data
  - Caches simply make data transfers go *faster*
- Each Memory Hierarchy level subset of next higher level
- Caches speed up due to **temporal locality**: store data used recently
- Block size  $> 1$  wd **spatial locality** speedup: Store words next to the ones used recently
- Cache design choices:
  - size of cache: speed v. capacity
  - direct-mapped v. associative
  - choice of N for N-way set assoc
  - block replacement policy
  - 2<sup>nd</sup> level cache? 3<sup>rd</sup> level cache?
  - Write through v. write back?
- Use performance model to pick between choices, depending on programs, technology, budget, ...





# An Actual CPU – Pentium M

Intel® Pentium®  
M Processor



## New Micro Architecture

77 Million Transistors



### Micro-Ops Fusion

– fuses operations together to enable faster execution of instructions at lower power

### Advanced Branch Prediction

– fewer re-dos for increased performance

### 1MB Power Optimized L2 Cache

– enables higher CPU performance

### Streaming SIMD Extensions II

compatible with Pentium® 4 Processor optimized software

### Dedicated Stack Management

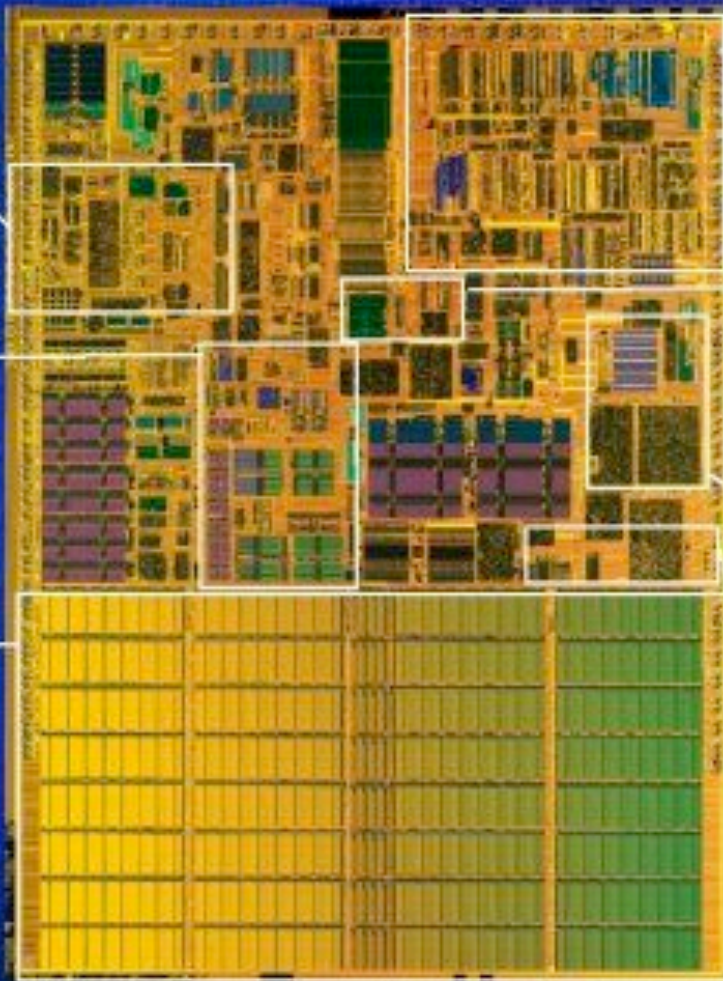
– faster instruction at lower power levels

### Enhanced Intel® SpeedStep® Technology

– Multiple voltages & frequency operating points

### 400 MHz Power Optimized System Bus

– faster system bus to enhance performance at lower power levels



intel.

Ca

# Peer Instruction

---

1. Increased associativity (1->2->4->8-way)  $\Rightarrow$  decreased or steady miss rate.
2. Increased associativity  $\Rightarrow$  increased cost & slower access time.
3. The ratio of costs of a “miss” vs. a “hit” are within an order of magnitude between VM & cache

	ABC
1 :	<b>FFF</b>
2 :	<b>FFT</b>
3 :	<b>FTF</b>
4 :	<b>FTT</b>
5 :	<b>TFF</b>
6 :	<b>TFT</b>
7 :	<b>TF</b> <b>F</b>
8 :	<b>TTT</b>

