CS 61C:
Great Ideas in Computer Architecture
Performance and Caches

Instructor:
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa13

New-School Machine Structures (It's a bit more complicated!)

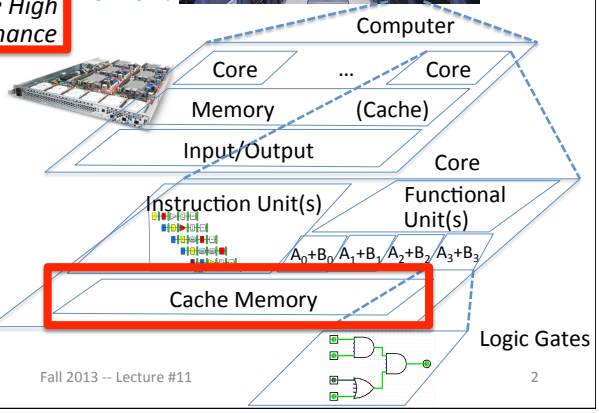
Software

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

Hardware

How do we know?



Agenda

- Defining Performance
- Administrivia
- Memory Hierarchy
- Technology Break
- Direct Mapped Caches
- And in Conclusion ...

Agenda

- Defining Performance
- Administrivia
- Memory Hierarchy
- Technology Break
- Direct Mapped Caches
- And in Conclusion …

What is Performance?

- *Latency* (or *response time* or *execution time*)
  - Time to complete one task
- *Bandwidth* (or *throughput*)
  - Tasks completed per unit time

Cloud Performance: Why Application Latency Matters

Server Delay (ms)	Increased time to next click (ms)	Queries/ user	Any clicks/ user	User satisfac- tion	Revenue/ User
50	--	--	--	--	--
200	500	--	-0.3%	-0.4%	--
500	1200	--	-1.0%	-0.9%	-1.2%
1000	1900	-0.7%	-1.9%	-1.6%	-2.8%
2000	3100	-1.8%	-4.4%	-3.8%	-4.3%

Figure 6.10 Negative impact of delays at Bing search server on user behavior [Brutlag and Schurman 2009].

- Key figure of merit: application responsiveness
  - Longer the delay, the fewer the user clicks, the less the user happiness, and the lower the revenue per user

Defining CPU Performance

- What does it mean to say X is faster than Y?
- Ferrari vs. School Bus?
- 2013 Ferrari 599 GTB
  - 2 passengers, 11.1 secs in quarter mile
- 2013 Type D school bus
  - 54 passengers, quarter mile time?

  http://www.youtube.com/watch?v=KwyCoQuhUNA
- *Response Time/Latency*: e.g., time to travel ¼ mile
- *Throughput/Bandwidth*: e.g., passenger-mi in 1 hour

Defining Relative CPU Performance

- $\text{Performance}_X = 1/\text{Program Execution Time}_X$
- $\text{Performance}_X > \text{Performance}_Y \Rightarrow$
  $1/\text{Execution Time}_X > 1/\text{Execution Time}_y \Rightarrow$
  $\text{Execution Time}_Y > \text{Execution Time}_X$
- Computer X is N times faster than Computer Y
  $\text{Performance}_X / \text{Performance}_Y = N$ or
  $\text{Execution Time}_Y / \text{Execution Time}_X = N$
- Bus is to Ferrari as 12 is to 11.1:
  Ferrari is 1.08 times faster than the bus!

Measuring CPU Performance

- Computers use a clock to determine when events takes place within hardware
- *Clock cycles:* discrete time intervals
  - aka clocks, cycles, clock periods, clock ticks
- *Clock rate* or *clock frequency:* clock cycles per second (inverse of clock cycle time)
- 3 GigaHertz clock rate
  => clock cycle time = $1/(3 \times 10^9)$ seconds
  clock cycle time = 333 picoseconds (ps)

CPU Performance Factors

- To distinguish between processor time and I/O, *CPU time* is time spent in processor
- `CPU Time/Program`
  `= Clock Cycles/Program`
  `x Clock Cycle Time`
- Or
  `CPU Time/Program`
  `= Clock Cycles/Program ÷ Clock Rate`

CPU Performance Factors

- But a program executes instructions
- `CPU Time/Program`
  `= Clock Cycles/Program x Clock Cycle Time`
  `= Instructions/Program`
  `x Average Clock Cycles/Instruction`
  `x Clock Cycle Time`
- 1st term called *Instruction Count*
- 2nd term abbreviated *CPI* for average ***C****lock Cycles* ***P****er* ***I****nstruction*
- 3rd term is 1 / Clock rate

Restating Performance Equation

- $\text{Time} = \frac{\text{Seconds}}{\text{Program}}$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

What Affects Each Component? Instruction Count, CPI, Clock Rate

Hardware or software component?	Affects What?
Algorithm	
Programming Language	
Compiler	
Instruction Set Architecture	

Computer A clock cycle time 250 ps, $CPI_A = 2$
Computer B clock cycle time 500 ps, $CPI_B = 1.2$
Assume A and B have same instruction set
Which statement is true?

- ☐ Computer A is ≈1.2 times faster than B
- ☐ Computer A is ≈4.0 times faster than B
- ☐ Computer B is ≈1.7 times faster than A
- ☐ Computer B is ≈3.4 times faster than A

Workload and Benchmark

- *Workload:* Set of programs run on a computer
  - Actual collection of applications run or made from real programs to approximate such a mix
  - Specifies both programs and relative frequencies
- *Benchmark:* Program selected for use in comparing computer performance
  - Benchmarks form a workload
  - Usually standardized so that many use them

SPEC
(System Performance Evaluation Cooperative)

- Computer Vendor cooperative for benchmarks, started in 1989
- SPECCPU2006
  - 12 Integer Programs
  - 17 Floating-Point Programs
- Often turn into number where bigger is faster
- *SPECratio*: reference execution time on old reference computer divide by execution time on new computer to get an effective speed-up

SPECINT2006 on AMD Barcelona

Description	Instruc-tion Count (B)	CPI	Clock cycle time (ps)	Execu-tion Time (s)	Refer-ence Time (s)	SPEC-ratio
Interpreted string processing	2,118	0.75	400	637	9,770	15.3
Block-sorting compression	2,389	0.85	400	817	9,650	11.8
GNU C compiler	1,050	1.72	400	724	8,050	11.1
Combinatorial optimization	336	10.0	400	1,345	9,120	6.8
Go game	1,658	1.09	400	721	10,490	14.6
Search gene sequence	2,783	0.80	400	890	9,330	10.5
Chess game	2,176	0.96	400	837	12,100	14.5
Quantum computer simulation	1,623	1.61	400	1,047	20,720	19.8
Video compression	3,102	0.80	400	993	22,130	22.3
Discrete event simulation library	587	2.94	400	690	6,250	9.1
Games/path finding	1,082	1.79	400	773	7,020	9.1
XML parsing	1,058	2.70	400	1,143	6,900	6.0

Summarizing Performance …

System	Rate (Task 1)	Rate (Task 2)
A	**10**	**20**
B	**20**	**10**

Flashcard Quiz: Which system is faster?

System A
System B
Same performance
Unanswerable question!

Summarizing SPEC Performance

- Varies from 6x to 22x faster than reference computer
- *Geometric mean* of ratios: N-th root of product of N ratios

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

  - Geometric Mean gives same relative answer no matter what computer is used as reference
- Geometric Mean for Barcelona is 11.7

Agenda

- Defining Performance
- Administrivia
- Memory Hierarchy
- Technology Break
- Direct Mapped Caches
- And in Conclusion …

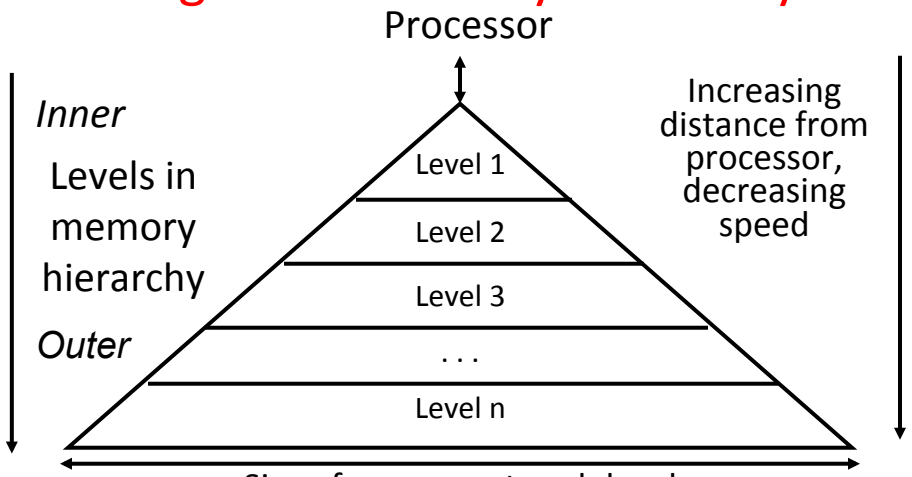
Administrivia

- Lab #5, Homework #4, Project #2-1
- Midterm, 17 October, 6-9 PM

Agenda

- Defining Performance
- Administrivia
- Memory Hierarchy
- Technology Break
- Direct Mapped Caches
- And in Conclusion ...

Big Idea: Memory Hierarchy



As we move to outer levels the latency goes up and price per bit goes down. Why?

Library Analogy

- Writing a report based on books on reserve
  - E.g., works of J.D. Salinger
- Go to library to get reserved book and place on desk in library
- If need more, check them out and keep on desk
  - But don't return earlier books since might need them
- You hope this collection of ~10 books on desk enough to write report, despite 10 being only 0.00001% of books in UC Berkeley libraries

Principle of Locality

- *Principle of Locality*: Programs access small portion of address space at any instant of time
- What program structures lead to locality in instruction accesses?

Cache Philosophy

- Programmer-invisible hardware mechanism to give illusion of speed of fastest memory with size of largest memory
  - Works fine even if programmer has no idea what a cache is
  - However, performance-oriented programmers today sometimes "reverse engineer" cache design to design data structures to match cache
  - We'll do that in Project 3

Memory Access without Cache

- Load word instruction: `lw $t0, 0($t1)`
- $t1 contains $1022_{ten,}$ Memory[1022] = 99

1. Processor issues address 1022_{ten} to Memory
2. Memory reads word at address 1022_{ten} (99)
3. Memory sends 99 to Processor
4. Processor loads 99 into register $t1

Memory Access with Cache

- Load word instruction: `lw $t0, 0($t1)`
- $t1 contains $1022_{ten,}$ Memory[1022] = 99
- With cache (similar to a hash)
  1. Processor issues address 1022_{ten} to Cache
  2. Cache checks to see if has copy of data at address 1022_{ten}
     - 2a. If finds a match (Hit): cache reads 99, sends to processor
     - 2b. No match (Miss): cache sends address 1022 to Memory
       - I. Memory reads 99 at address 1022_{ten}
       - II. Memory sends 99 to Cache
       - III. Cache replaces word with new 99
       - IV. Cache sends 99 to processor
  3. Processor loads 99 into register $t1

Cache "Tags"

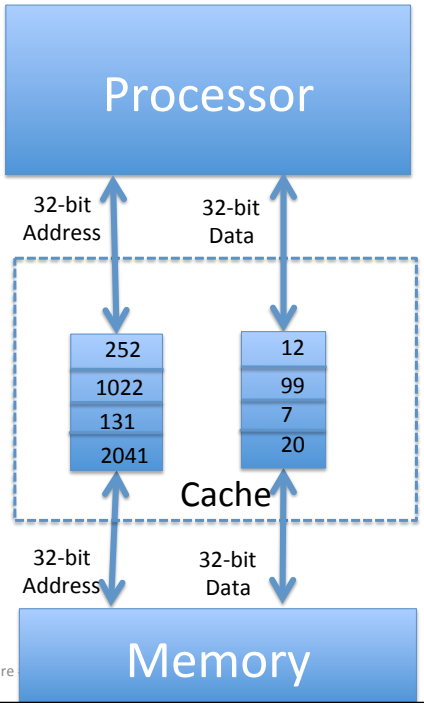
- Need way to tell if have copy of location in memory so that can decide on hit or miss
- On cache miss, put memory address of block in "tag address" of cache block
  - 1022 placed in tag next to data from memory (99)

Tag	Data
252	12
1022	99
131	7
2041	20

From earlier instructions

Anatomy of a 16 Byte Cache, 4 Byte Block

- Operations:
  1. Cache Hit
  2. Cache Miss
  3. Refill cache from memory
- Cache needs Address Tags to decide if Processor Address is a Cache Hit or Cache Miss
  - Compares all 4 tags



Cache Requirements

- Suppose processor now requests location 511, which contains 11?
- Doesn't match any cache block, so must "evict" one resident block to make room
  - Which block to evict?
- Replace "victim" with new memory block at address 511

Tag	Data
252	12
1022	99
511	11
2041	20

Block Must be Aligned in Memory

- Word blocks are aligned, so binary address of all words in cache always ends in 00_{two}
- How to take advantage of this to save hardware and energy?
- Don't need to compare last 2 bits of 32-bit byte address (comparator can be narrower)

=> Don't need to store last 2 bits of 32-bit byte address in Cache Tag (Tag can be narrower)

Anatomy of a 32B Cache, 8B Block

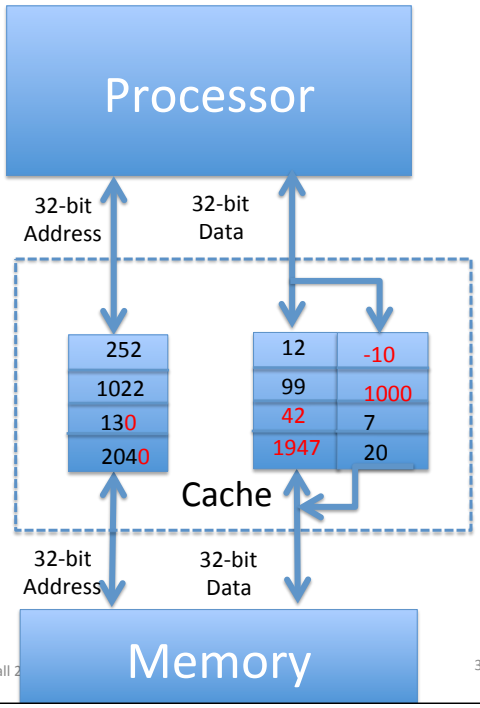
- Blocks must be aligned in pairs, otherwise could get same word twice in cache

⇒Tags only have even-numbered words

⇒ Last 3 bits of address always 000_{two}

⇒Tags, comparators can be narrower

- Can get hit for either word in block



Big Idea: Locality

- *Temporal Locality* (locality in time)
  - Go back to same book on desktop multiple times
  - If a memory location is referenced, then it will tend to be referenced again soon
- *Spatial Locality* (locality in space)
  - When go to book shelf, pick up multiple books on J.D. Salinger since library stores related books together
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

Principle of Locality

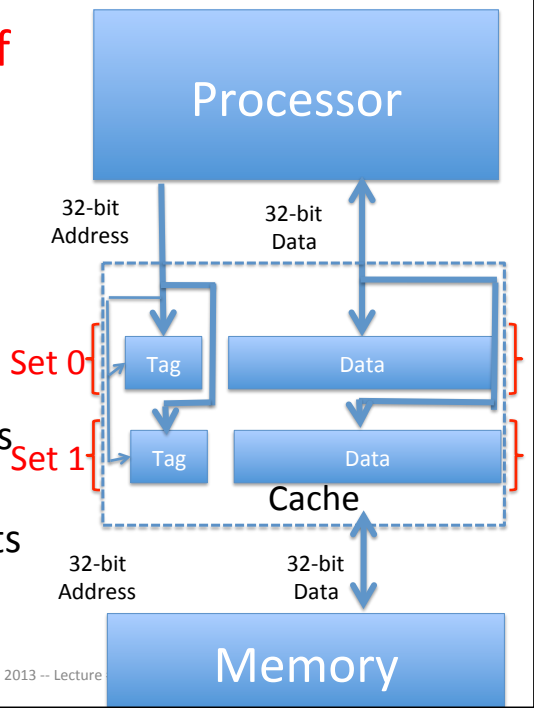
- *Principle of Locality*: Programs access small portion of address space at any instant of time
- What program structures lead to temporal and spatial locality in instruction accesses?
- In data accesses?

Common Cache Optimizations

- Reduce tag overhead by having larger blocks
  - E.g., 2 words, 4 words, 8 words
- Separate caches for instructions and data
  - Double bandwidth, don't interfere with each other
- Bigger caches (but access time could get bigger than one clock cycle if too big)
- Divide cache into multiple sets, only search inside one set => saves comparators, energy
  - If as many sets as blocks, then only 1 comparator (aka Direct-Mapped Cache)
  - But may increase Miss Rate

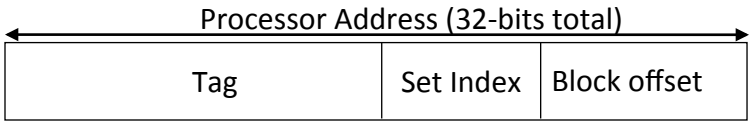
Hardware Cost of Cache

- Need to compare every tag to the Processor address
- Comparators are expensive
- Optimization: 2 sets => ½ comparators
- 1 Address bit selects which set



Processor Address Fields used by Cache Controller

- Block Offset: Byte address within block
- Set Index: Selects which set
- Tag: Remaining portion of processor address



- Size of Index = log2 (number of sets)
- Size of Tag = Address size – Size of Index – log2 (number of bytes/block)

What is limit to number of sets?

- Can save more comparators if have more than 2 sets
- Limit: As Many Sets as Cache Blocks – only needs one comparator!
- Called "Direct-Mapped" Design

One More Detail: Valid Bit

- When start a new program, cache does not have valid information for this program
- Need an indicator whether this tag entry is valid for this program
- Add a "valid bit" to the cache tag entry
  - 0 => cache miss, even if by chance, address = tag
  - 1 => cache hit, if processor address = tag

Agenda

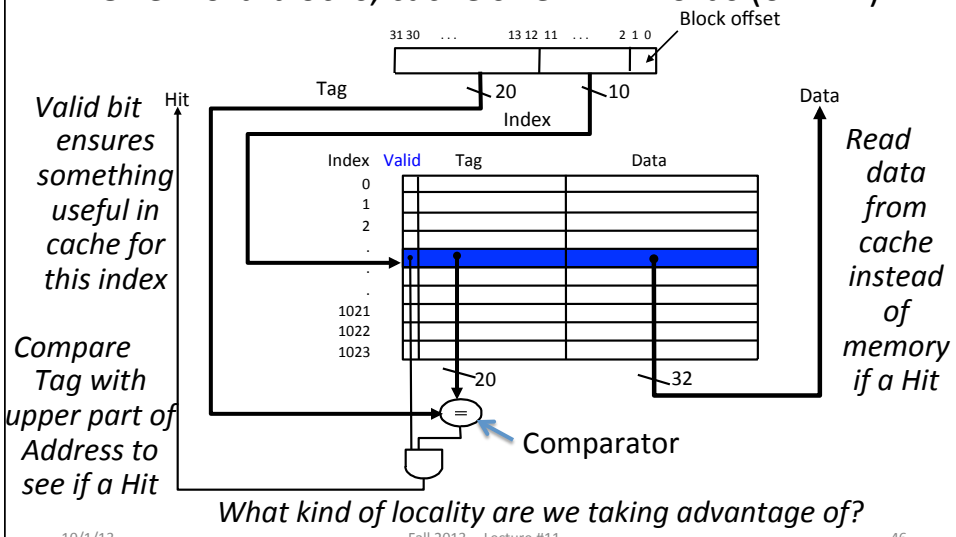
- Defining Performance
- Administrivia
- Memory Hierarchy
- Technology Break
- Direct Mapped Caches
- And in Conclusion …

Agenda

- Defining Performance
- Administrivia
- Memory Hierarchy
- Technology Break
- Direct Mapped Caches
- And in Conclusion …

Direct-Mapped Cache Example

- One word blocks, cache size = 1K words (or 4KB)

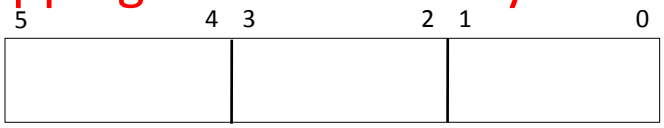


Cache Terms

- Hit rate: fraction of access that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)

- Abbreviation: "$" = cache (A Berkeley innovation!)

Mapping a 6-bit Memory Address



Mem Block Within $ Block
Tag

Block Within $
Index

Byte Offset Within Block
(e.g., Word)

- In example, block size is 4 bytes/1 word (it could be multi-word)
- Memory and cache blocks are the same size, unit of transfer between memory and cache
- # Memory blocks >> # Cache blocks
  - 16 Memory blocks/16 words/64 bytes/6 bits to address all bytes
  - 4 Cache blocks, 4 bytes (1 word) per block
  - 4 Memory blocks map to each cache block
- Byte within block: low order two bits, ignore! (nothing smaller than a block)
- Memory block to cache block, aka *index*: middle two bits
- Which memory block is in a given cache block, aka *tag*: top two bits

Caching: A Simple First Example

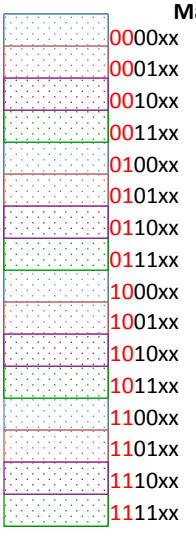
Cache

Index	Valid	Tag	Data
00			
01			
10			
11			

Q: Is the mem block in cache?

Compare the cache tag to the high-order 2 memory address bits to tell if the memory block is in the cache
(provided valid bit is set)

Main Memory



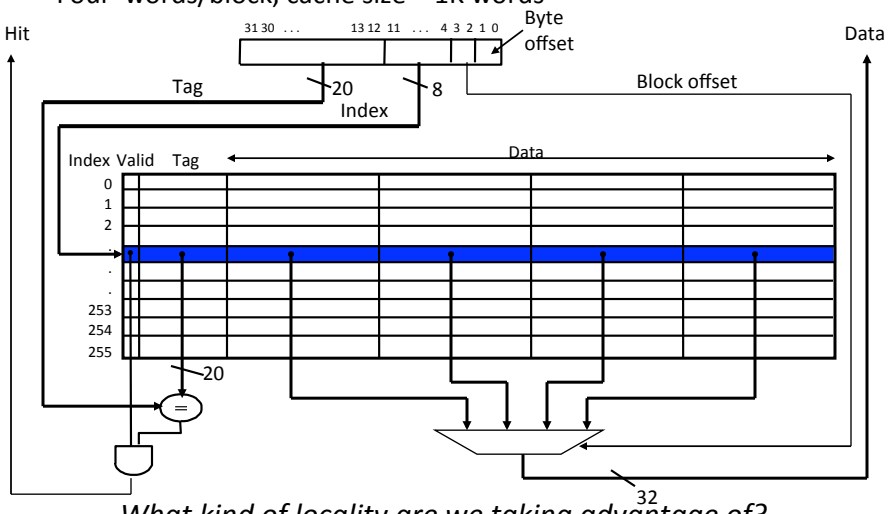
One word blocks. Two low-order bits define the byte in the block (32b words).

Q: Where in the cache is the memory block?

Use next 2 low-order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

Multiword-Block Direct-Mapped Cache

- Four words/block, cache size = 1K words



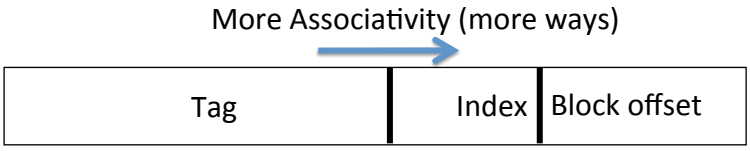
What kind of locality are we taking advantage of?

Cache Names for Each Organization

- "Fully Associative": Block can go anywhere
  - First design in lecture
  - Note: No Index field, but 1 comparator/block
- "Direct Mapped": Block goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for a block
  - Number of sets = number of blocks / N
  - Fully Associative: N = number of blocks
  - Direct Mapped: N = 1

Range of Set-Associative Caches

- For a fixed-size cache, each increase by a factor of 2 in associativity doubles the number of blocks per set (i.e., the number of "ways") and halves the number of sets –
  - decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



Note: IBM persists in calling sets "ways" and ways "sets". They're wrong.

For S sets, N ways, B blocks, which statements hold?

A) The cache has B tags

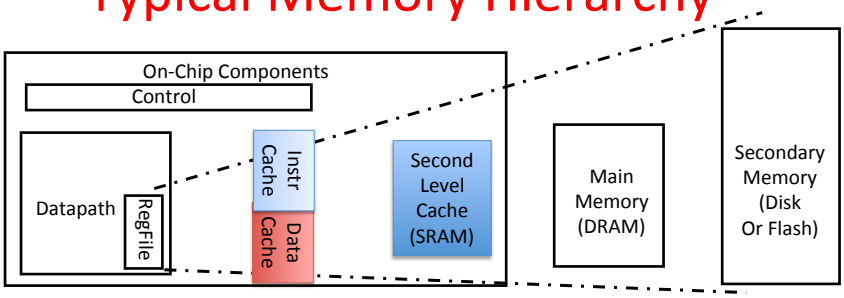
B) The cache needs N comparators

C) B = N x S

D) Size of Index = $\text{Log}_2(S)$

- ☐ A only
- ☐ A and B only
- ☐ A, B, and C only
- ☐ All four statements are true

Typical Memory Hierarchy



Speed (cycles):	½'s	1's	10's	100's	1,000,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost/bit:	highest	⟷			lowest

- Principle of locality + memory hierarchy presents programmer with ≈ as much memory as is available in the *cheapest* technology at the ≈ speed offered by the *fastest* technology

And In Conclusion, …

- Time (seconds/program) is measure of performance

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

- Principle of Locality for Libraries /Computer Memory
- Hierarchy of Memories (speed/size/cost per bit) to Exploit Locality
- Cache – copy of data lower level in memory hierarchy
- Direct Mapped to find block in cache using Tag field and Valid bit for Hit
- Larger caches reduce Miss rate via Temporal and Spatial Locality, but can increase Hit time
- Multilevel caches help Miss penalty
- AMAT helps balance Hit time, Miss rate, Miss penalty