

Introduction to Cache's

This week's Focus is on Fundamentals

David Andrews
Computer Engineering Group
University of Paderborn

`dandrews@ittc.ku.edu`



UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Agenda

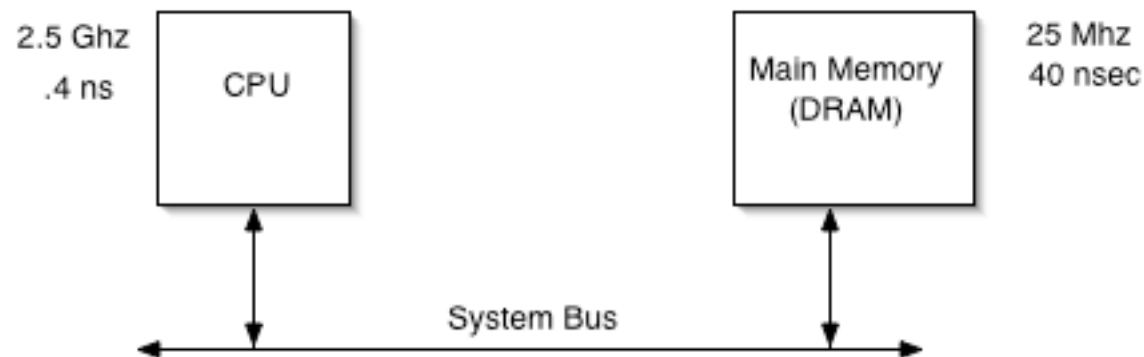
- The Domain of Cache's
 - ◆ Fundamental Level in Memory Hierarchy
 - ◆ Prevent Slowdowns of CPU
 - Instruction Fetching
 - Data Fetching
- Why The Work
 - ◆ Locality of Reference
 - Temporal
 - Spatial
- Baseline Cache Operation
 - ◆ Address Comparisons and Data Blocks (Lines)
 - ◆ Address Comparisons based on Tags
- Common Organizations
 - ◆ Direct Mapped
 - Simple but slowest
 - ◆ Fully Associative
 - Most Complex and Fastest
 - ◆ Set Associative
 - Close to Fully Associative Performance + Simplicity of Direct

Agenda (Continued)

- Cache Control
 - ◆ Update Policies
 - Write Back, Write Through
- Replacement Policies
 - ◆ Selecting Which “Block” to Replace

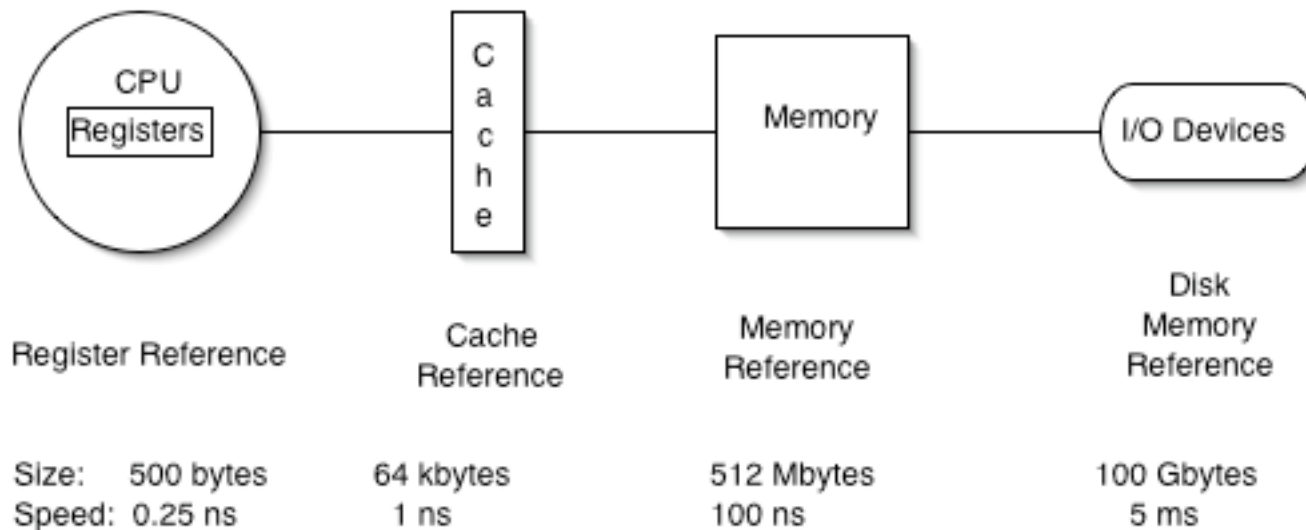
The Domain of Caches

- Why Were Caches Created ?
 - ◆ Performance, Performance, Performance.....Any Questions ?
- Consider This.....
 - ◆ We want RISC Scalar CPU to Input 1 Instruction per Clock
 - CPU Fetches Each Instruction From DRAM (Cheap but Dense Memory)
 - but has to wait $40/.4 = 100$ Clocks between Accesses Due to Cycle Time
 - DRAM Slow
 - Bus Slow (Cycle Times + Arbitration not Even Considered Here)
 - CPU Pin Limited (Prevents Simultaneous Instruction Fetch)



Achieving A Memory Hierarchy

- Objective: Make System that:
 - ◆ 1: Provides Bulk and Cost Close to Disk
 - ◆ 2: Provides Performance of Registers/CPU

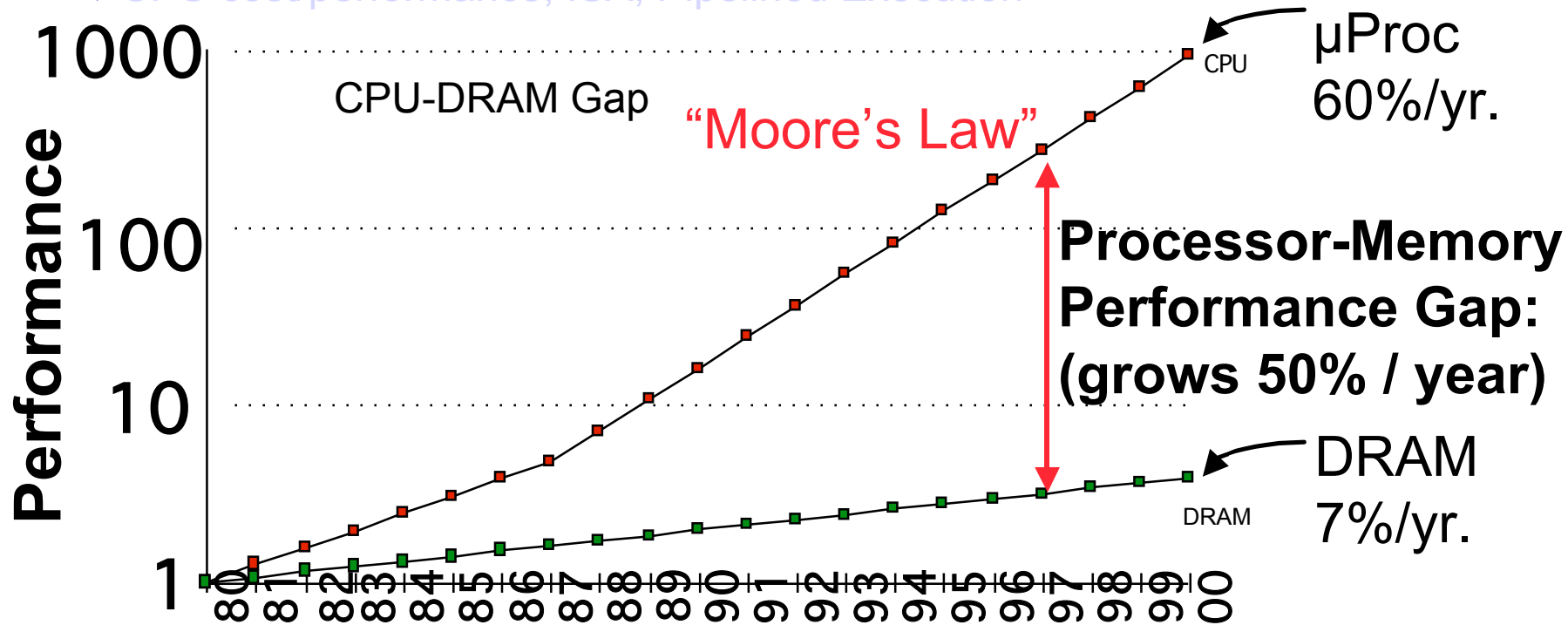


Taken from Hennessey & Patterson Circa 2000

Review: Who Cares About the Memory Hierarchy? (Slide from David Patterson)

- Processor Only Thus Far in Course:

- ◆ CPU cost/performance, ISA, Pipelined Execution



- 1980: no cache in μ proc; 1995 2-level-cache on chip (1989 first Intel μ proc with a cache on chip)

Processor-Memory Performance Gap “Tax” (Slide from David Patterson)

Processor	% Area (<i>_cost</i>)	%Transistors (<i>_power</i>)
■ Alpha 21164	37%	77%
■ StrongArm SA110	61%	94%
■ Pentium Pro	64%	88%
◆ 2 dies per package: Proc/I\$/D\$ + L2\$		
■ Caches have no inherent value, only try to close performance gap		

Why Caches Work

■ Principle of Locality:

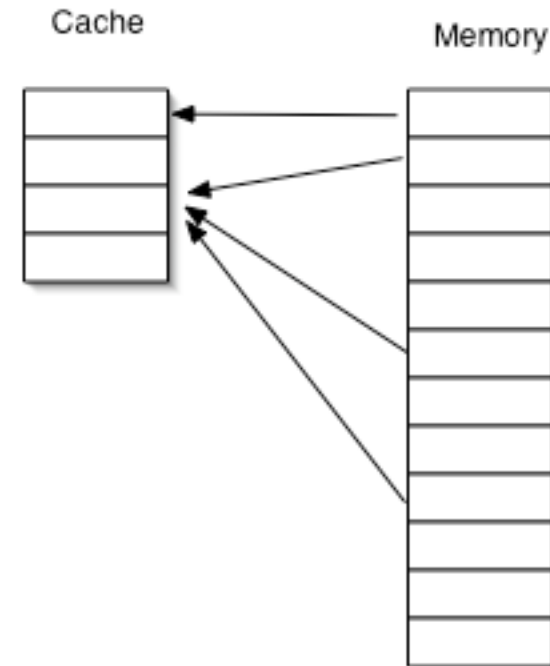
- ◆ Temporal: If you use an instruction/data item in the near past, then you will probably use it again in the near future.
 - Loops
 - Variable re-use
- ◆ Spatial: If you use an instruction/data item, then you will probably use others close in address space
 - Sequential Instruction Execution
 - Data Arrays

■ Basic Operation:

- ◆ CPU Issues Address
- ◆ Cache Compares To Existing Addresses (Tag Compare)
 - If hit, continue
 - If miss, stop execution and pull in complete line
 - Cache refill times can be considerable. Worsens with multi-level caches.
We won't consider refill times in our basic coverage today

Big Picture Operation

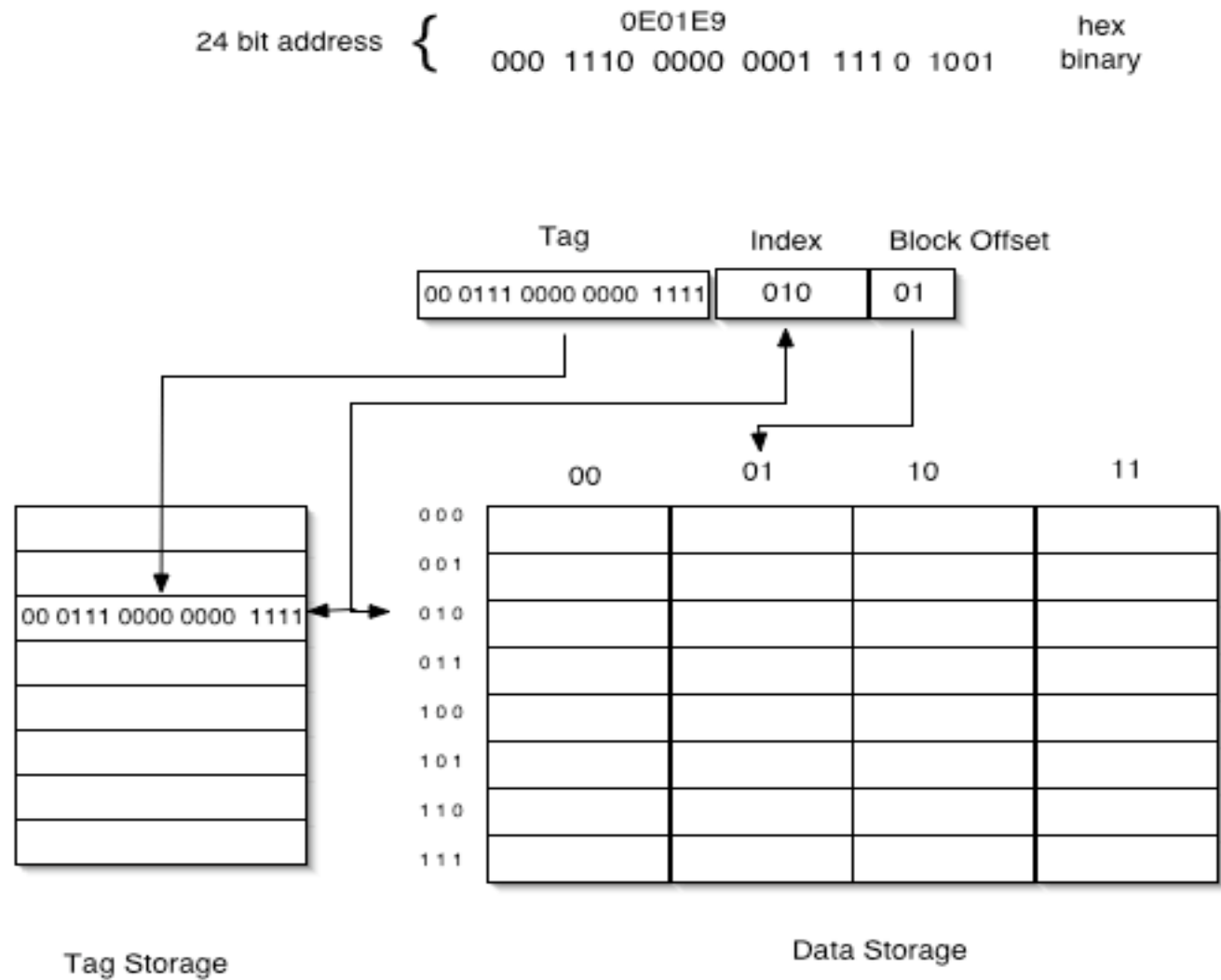
- Cache is Based on SRAM (D-Flip Flops).
 - ◆ Much Faster than DRAM
- Cache Memory is Limited
 - ◆ Obviously, Map Multiple Locations From Main Memory into Cache
 - ◆ Question: How Do We Decide The Mapping ?
 - Direct Mapped
 - Fully Associative
 - Set Associative
 - ◆ Lets First Look At Cache Organization



Direct Mapped Cache Organization

■ Break Address Into 3 Parts

- ◆ Block Offset
- ◆ Index
- ◆ Tag:



Sizing Analysis

■ Direct Mapped Cache Sizing

◆ Given by Index x Block Size (in Bytes)

- Total Size = $2^{\text{\#index_bits}} \times 2^{\text{\#block_offset_bits}}$
- This Example = $2^3 \times 2^2 = 2^5 = 32$ bytes

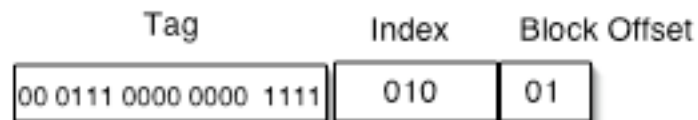
◆ Note* Independent of Tag Size

- Does Cache Size Change for this Example for 32 bit Address ?

◆ How Many Blocks Mapped into Common Line ?

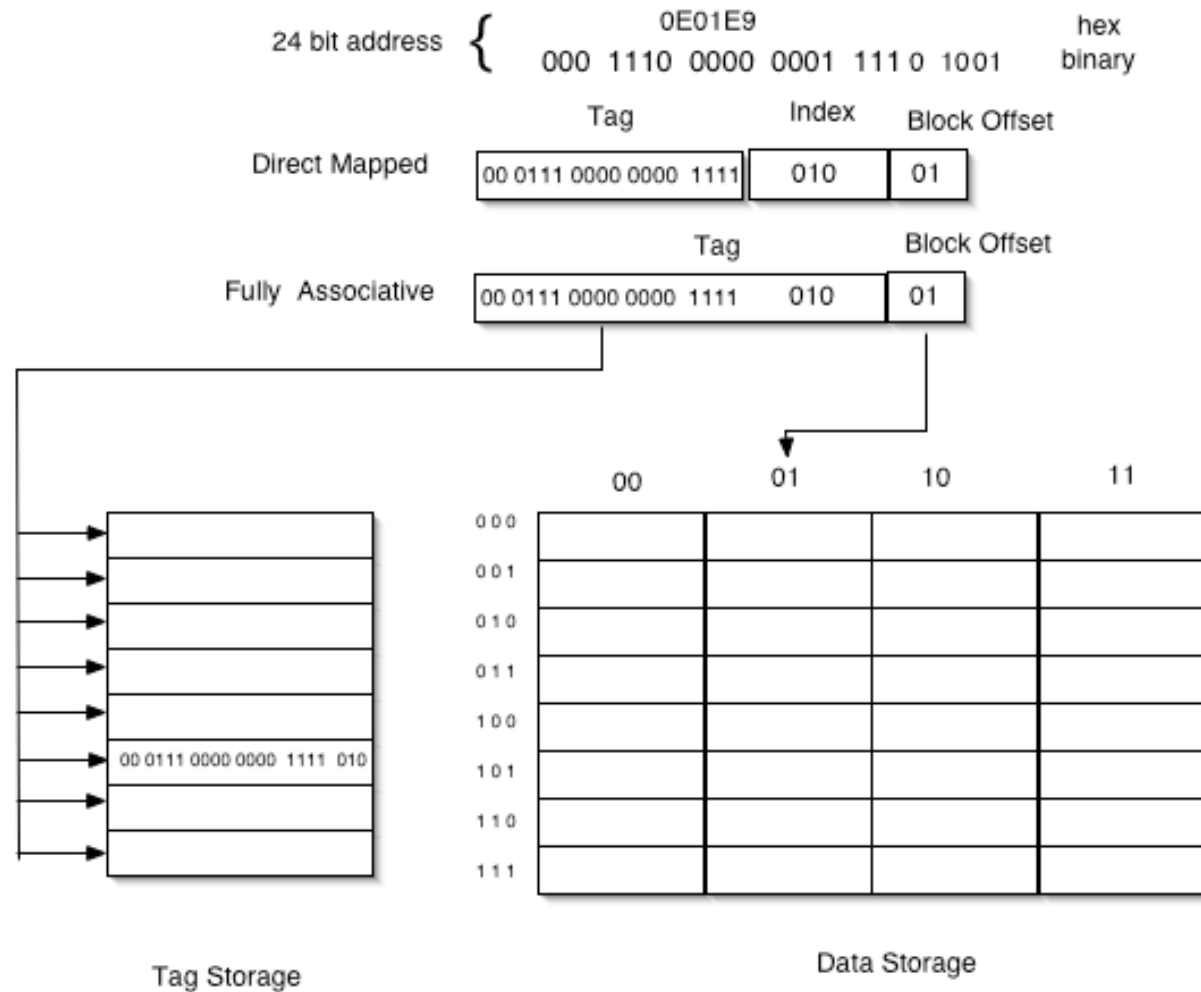
- Does this change for 32 bit Address ?

24 bit address { 000 1110 0000 0001 1110 1001 hex
binary



Fully Associative Cache

- Tag Can Go Anywhere: Better Utilization

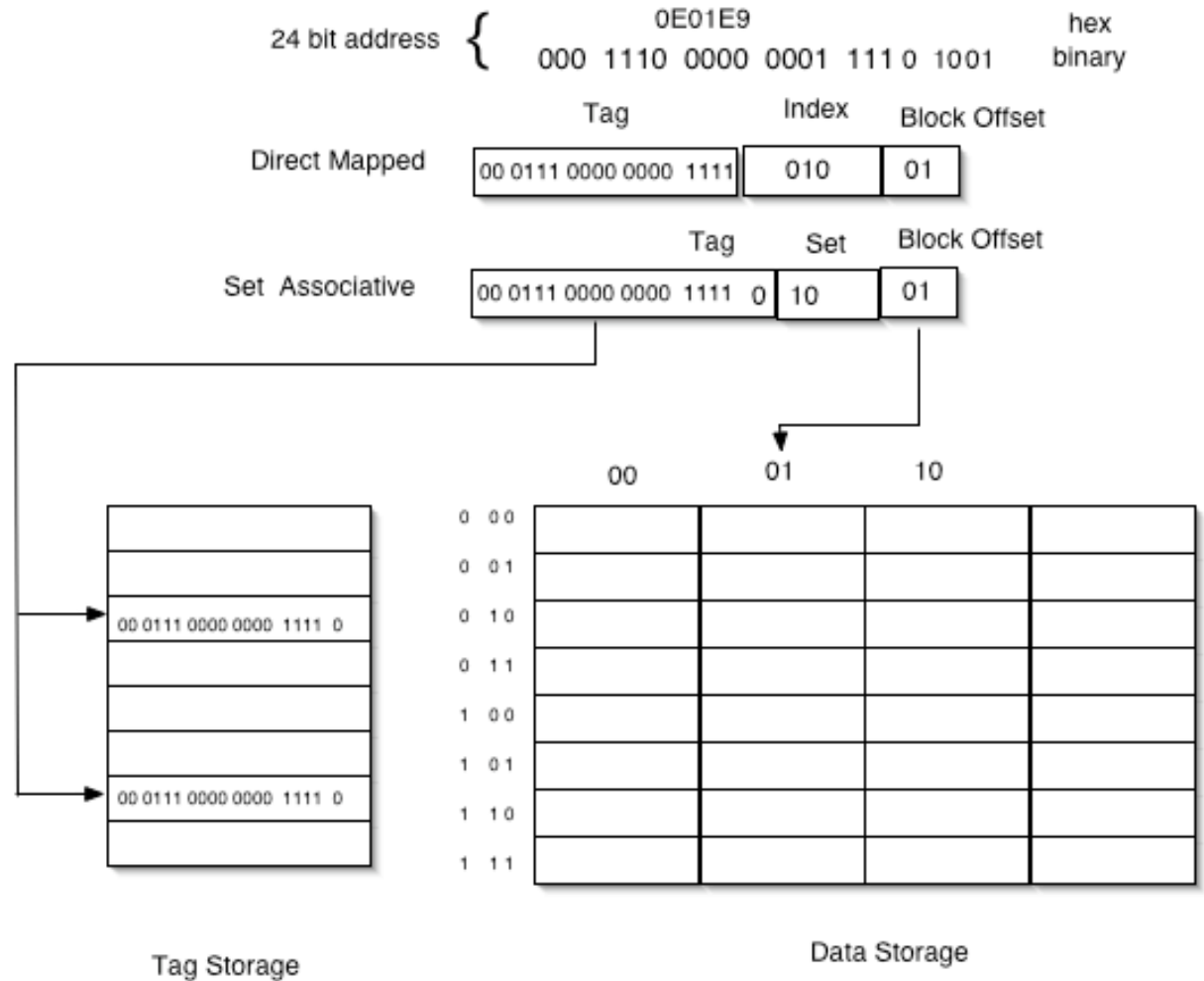


Set Associative

2-Way Set Associative

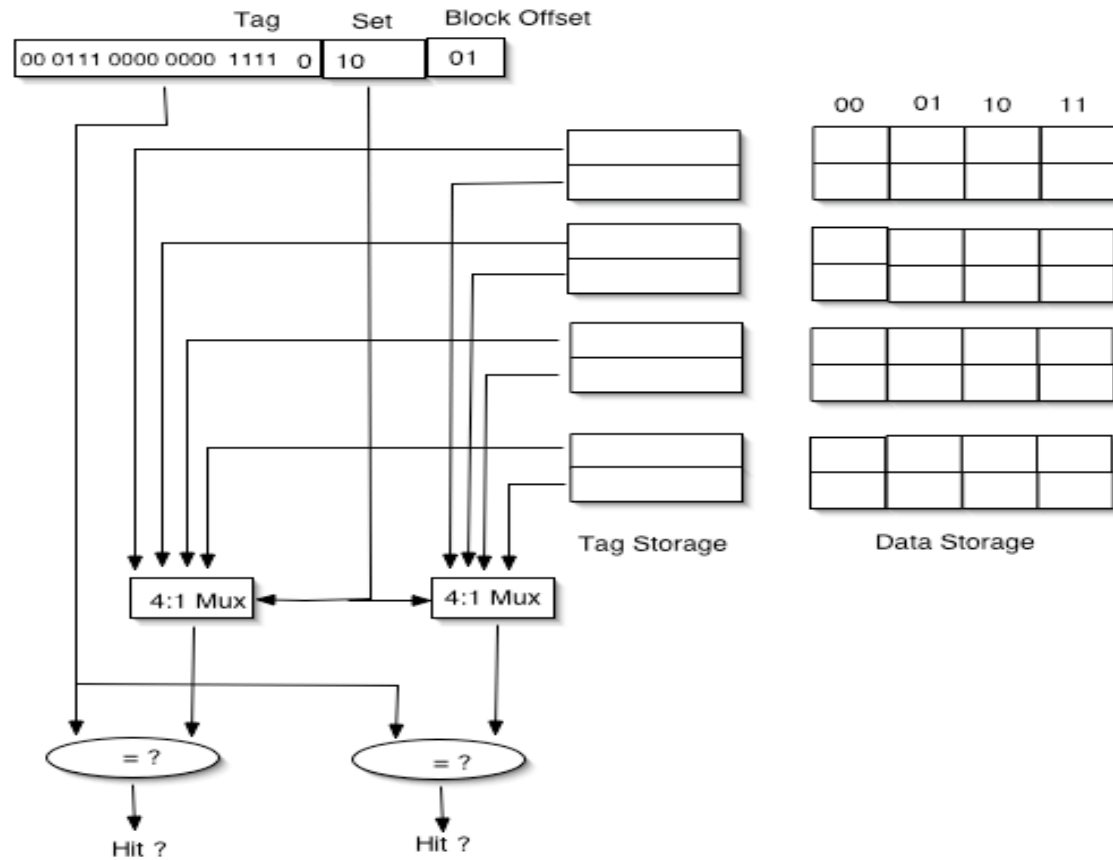
“Way-ness” :

- = # Storage Loc
- = # Comparisor



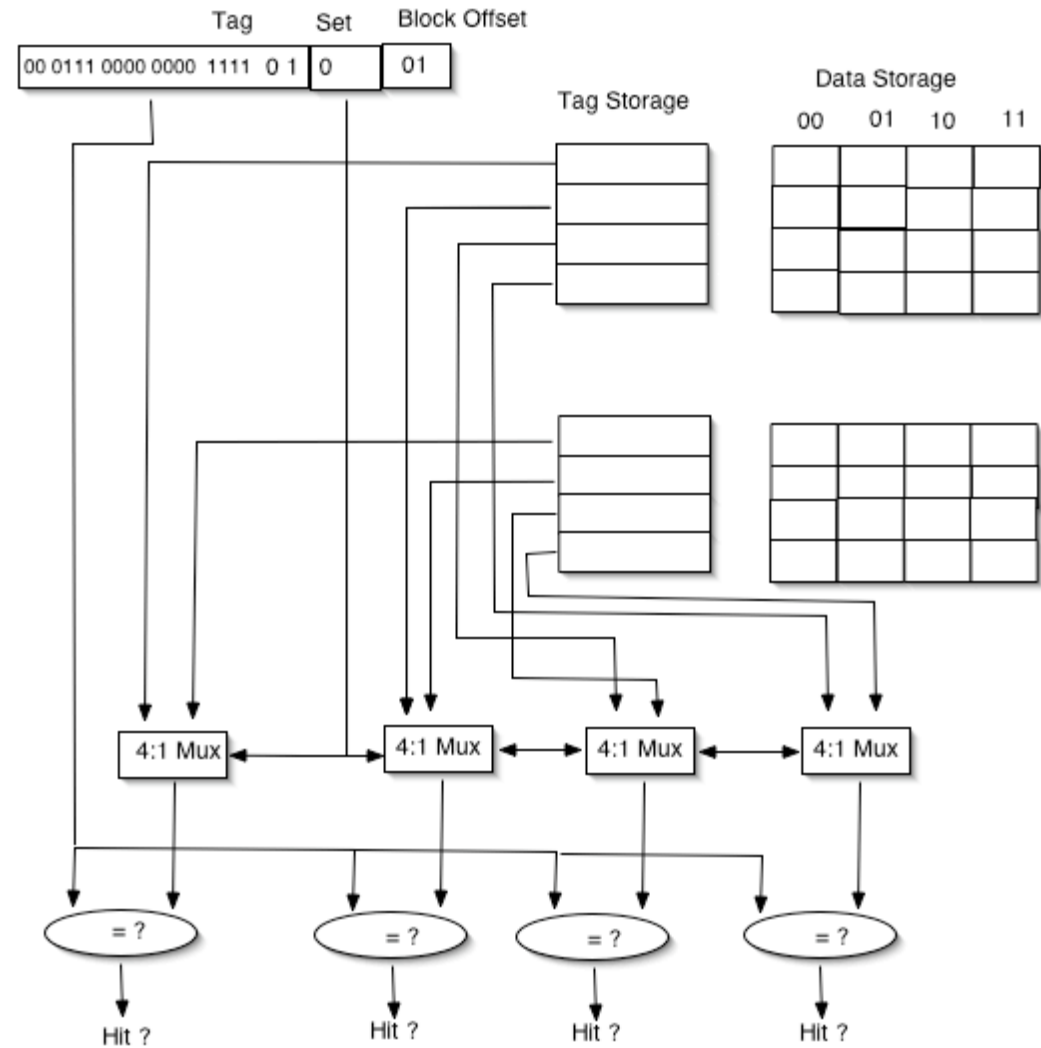
2-Way Set Associative Cache (Better Representation)

- Sets Formation as Grouped Blocks
- N sets := N:1 Multiplexers
- Wayness = # Multiplexers
- Wayness = # Comparitors



4-Way Set Associative

- 4-Way Uses 4 Comparitors
- 2 Sets (In this Example)
- 4 Places to put a Block



A Little Comparison Between Organizations

- Direct, Full, and Set Associative are all really the same

Associativity N Lines	Way- ness	#sets	# Muxes	Size of Muxes	# Comp	Comments
Direct	1	N	1	N:1	1	
Set Ass. M way	M	N/M	M	N/M:1	M	
Full Ass.	N	1	1	N:1	N	

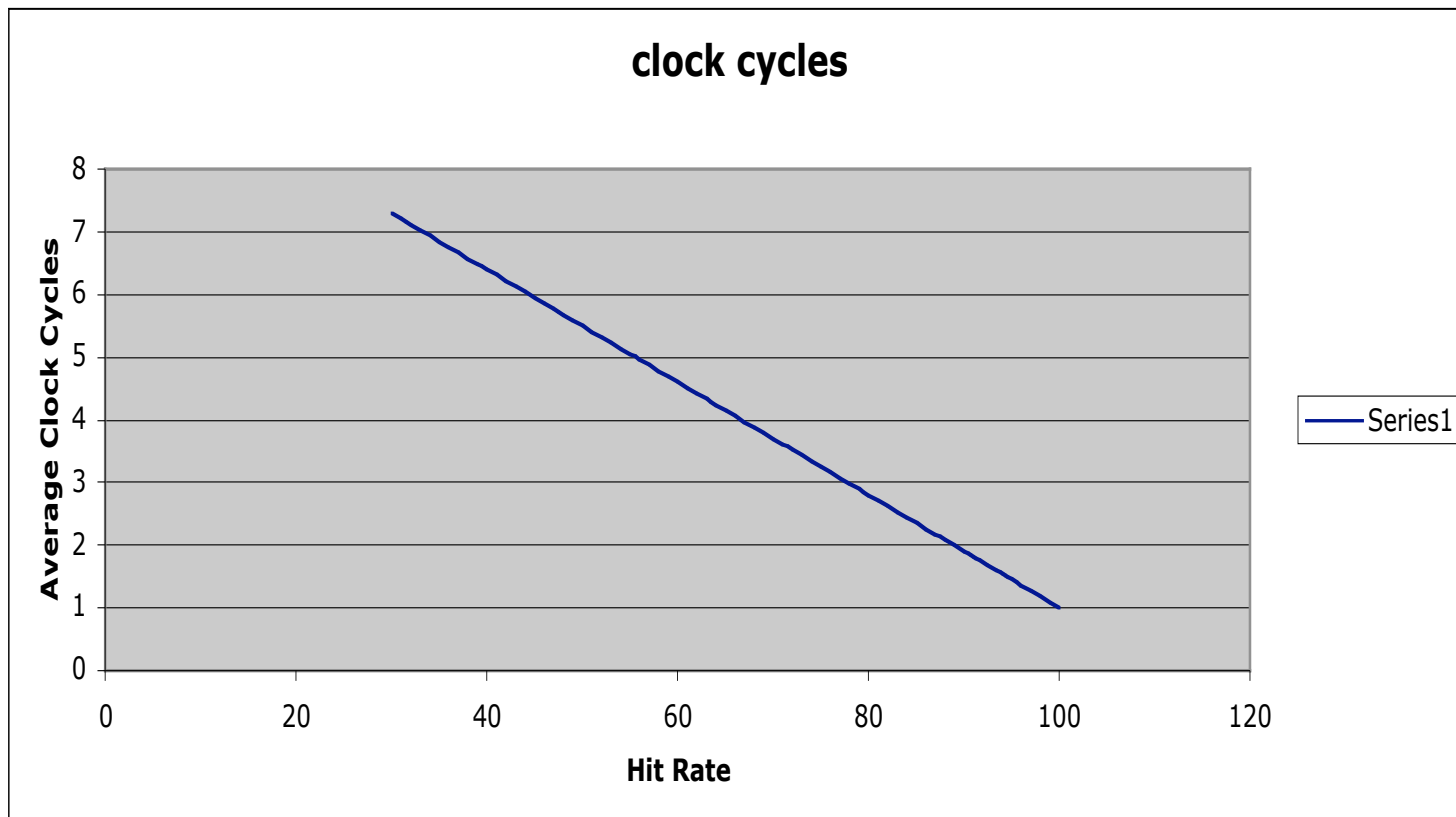
Measuring Performance

How We Measure Cache Performance:

- Hit rate: Percentage of Accesses Issued by CPU Found in Cache
 - ◆ H usually pretty high; say 96 - 99%
- Average Access Time: The Average, or Effective Access Time Using a Cache
 - ◆ $T_{acc} = t_{cache} \times h + t_{mm}(1-h)$
- Performance is Very Sensitive to Miss Rate (1- Hit Rate)
 - ◆ Consider ratio of 100:1 cycle time difference

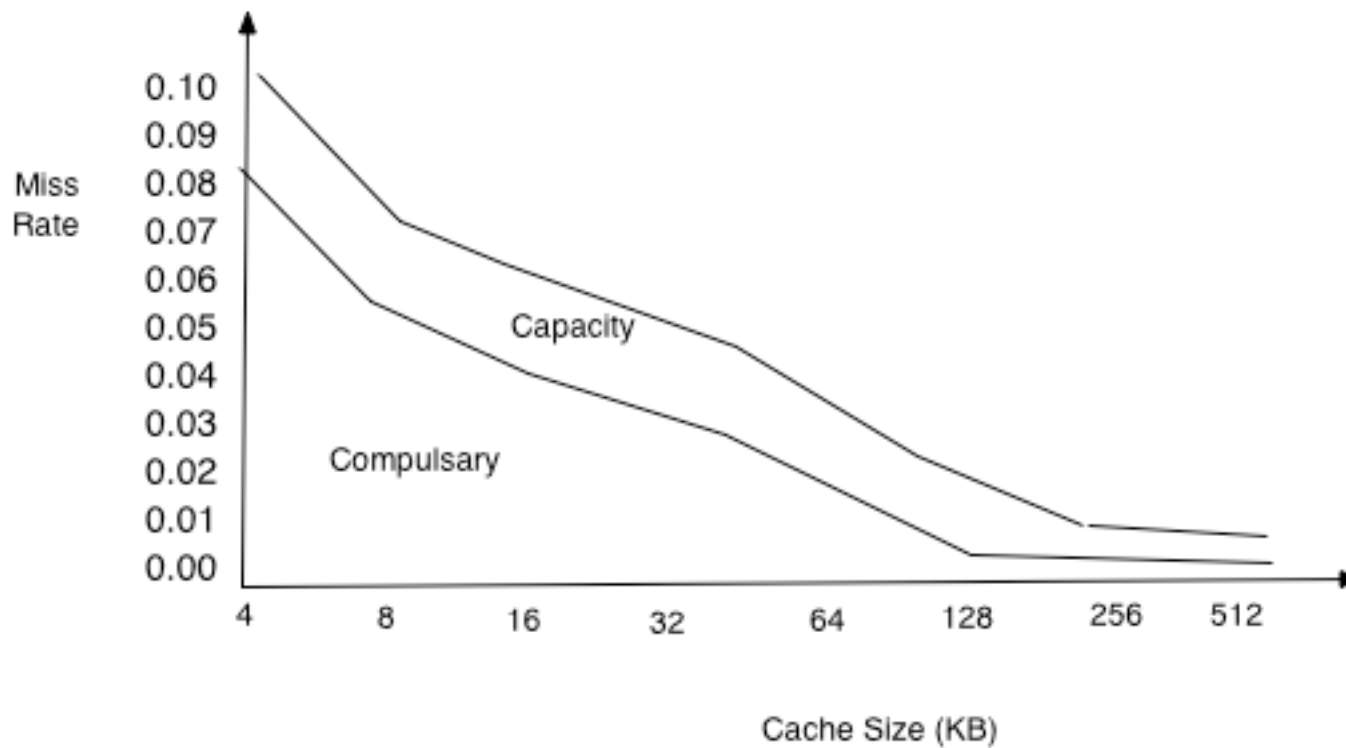
Performance Effects of Cache

- Assume Cache is 100x Faster Than Main Memory



Cache Misses and Size

- **Compulsary Misses:** Assumes an infinite size cache. Compulsary misses occur when a block is first accessed. Also called “Cold Start” misses
- **Capacity Misses:** If cache cannot contain all blocks (program/data) needed, then misses occur because blocks are discarded and then later retrieved.
- **Conflict:** Misses due to associativity constraints. No Conflict misses for Fully Associative. Some for set associative and the most for direct mapped.
- Consider Graph for Fully Associative Cache (No Conflict Info in this Graph)

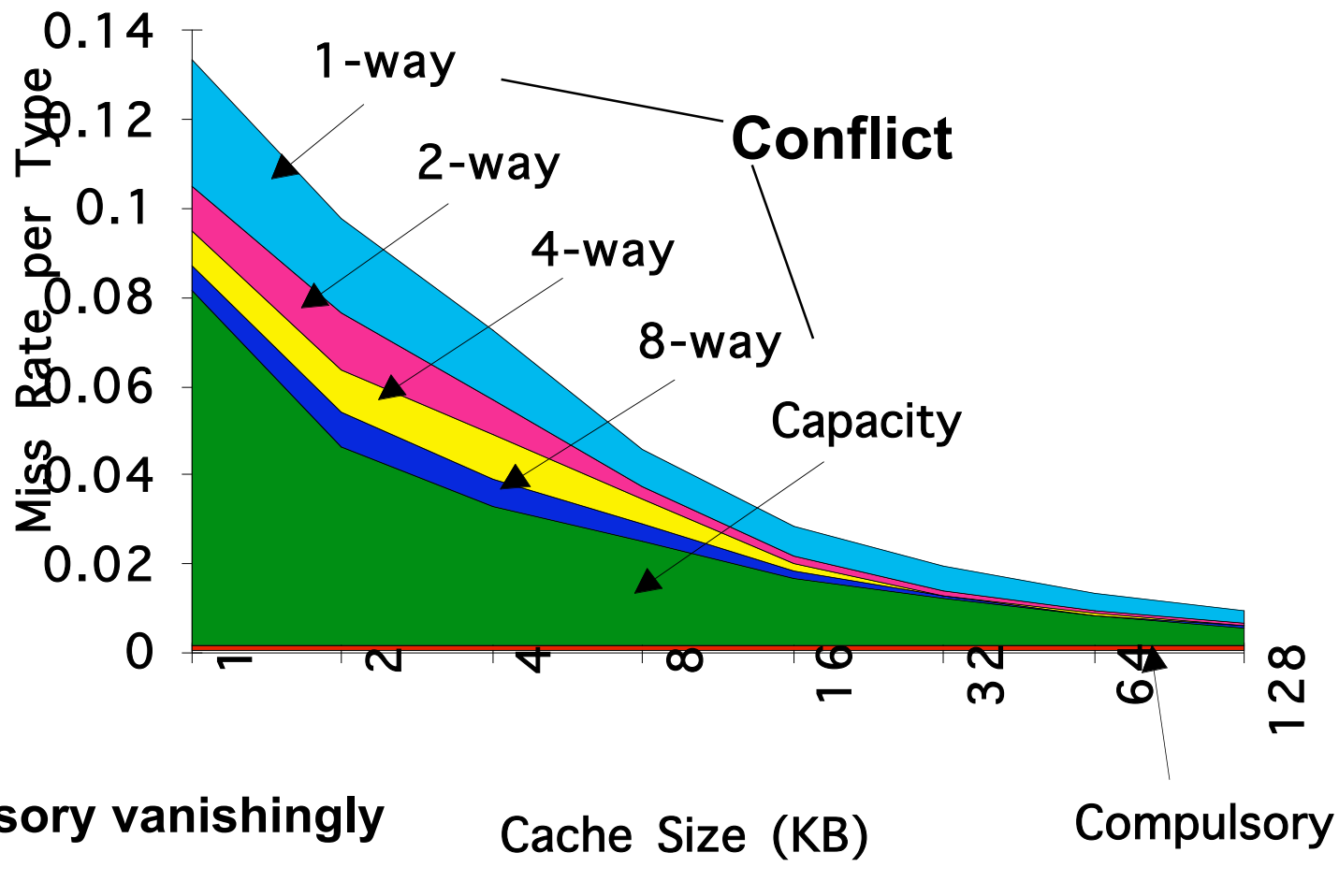


Effects of Associativity

- Does Associativity Effect Hit Rate ?
 - ◆ You bet.....
- Simple Thought Game...
 - ◆ An Increase in Associativity Enables More Options on Where an Instruction/Datum Can be Stored in Cache
 - Will a Set Associative Cache Ever Perform Worse than A Direct Mapped Cache ?
 - Will a Fully Associative Cache Ever Perform Worse than a Set Associative Cache ?
 - ◆ Conflict Misses: Hit Rate Differences Between Levels of Associativity.

3Cs Absolute Miss Rate (SPEC92)

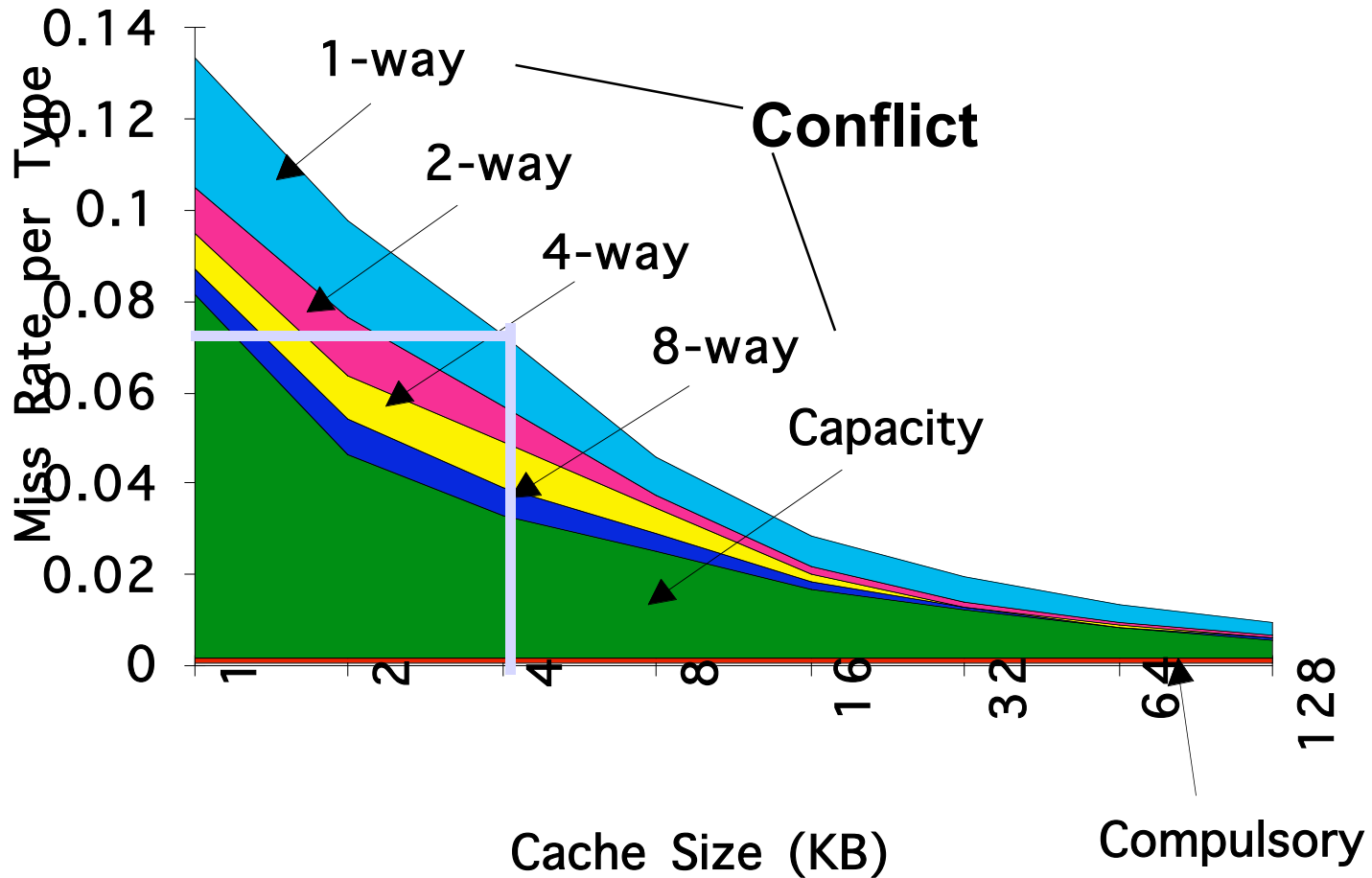
(Slide from David Patterson)



2:1 Cache Rule

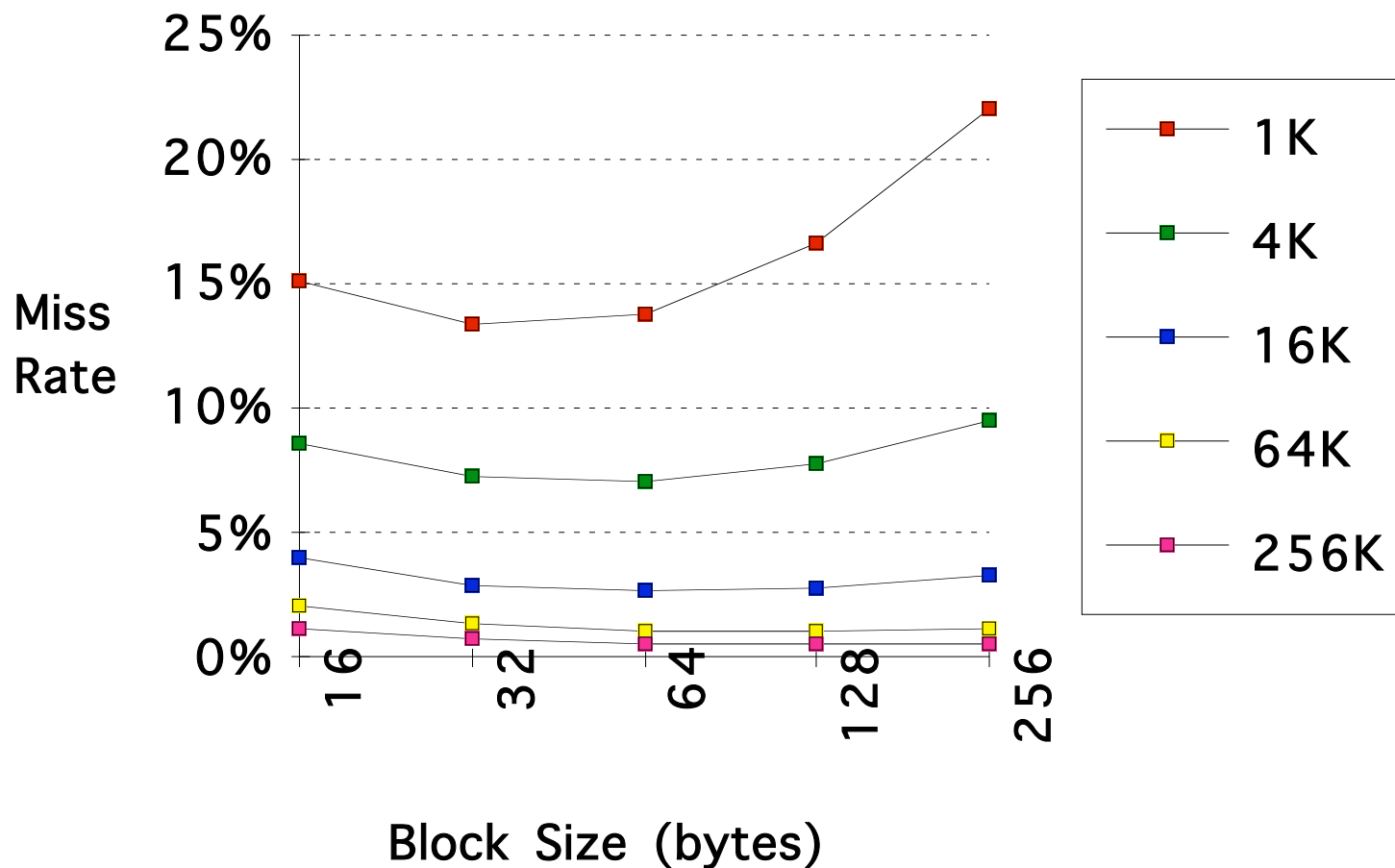
(Slide from David Patterson)

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



1. Reduce Misses via Larger Block Size

Slide from David Patterson



Block Replacement

- When a miss occurs and all blocks (direct, set, full ?) are occupied, which one do you replace ?
 - ◆ Thought Experiment: What would ideal replacement policy be ?
 - Requires us to predict future

- Realistic Policies
 - ◆ Random: Simply pick one
 - ◆ Least Recently Used (LRU): Relies on the past to predict the future. Don't replace a block that has recently been used, replace block that has not been used for the longest time.
 - ◆ First In, First Out (FIFO): Simpler version of LRU.

 - ◆ LRU is probably the most common. Best approximation we have to ideal. Check out data sheets for your Pentiums....

What Happens on a Write ?

- Write Back: Only Update the Cache, not Main Memory
 - ◆ Pro's
 - Best Performer: All Accesses Occur At Cache Cycle Times
 - Minimizes Updates to a Single Variable (summation etc.)
 - ◆ Con's
 - Modest Increase in Complexity (A Dirty Bit)
 - Must First “Flush” Back Dirty Line Before Replacement
 - Inconsistent Memory State (Multiple Values in Cache and Main Memory Possible)

- Write Through: Update Through Cache and Into Main Memory
 - ◆ Pro's
 - Keeps Cache and Main Memory Consistent
 - Important for Multiprocessors ?
 - Line Refills Simple and Fast, No Need to Flush Stale Data
 - ◆ Con's
 - ◆ Writes Occur at Main Memory Speed, not Cache
 - How Often Do We Write ?