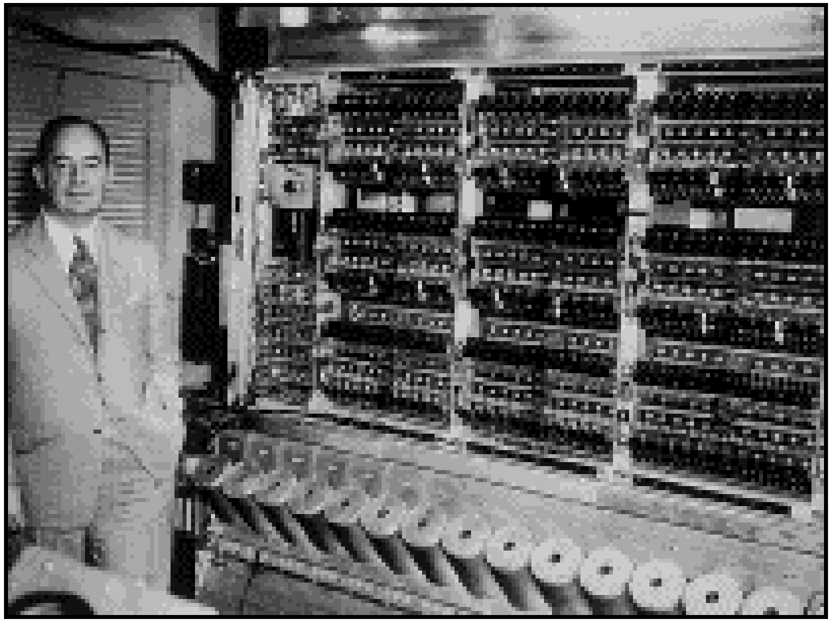
CSE 3421
Introduction to Computer Architecture

Chapter 5:
The Memory Hierarchy

Xiaodong Zhang

Von Neumann Model: Computer Architecture Design



Von Neumann model

- **a memory** containing both data and instructions
- **a computing unit** for both arithmetic and logical operations
- **a control unit** to interpret instructions and make actions

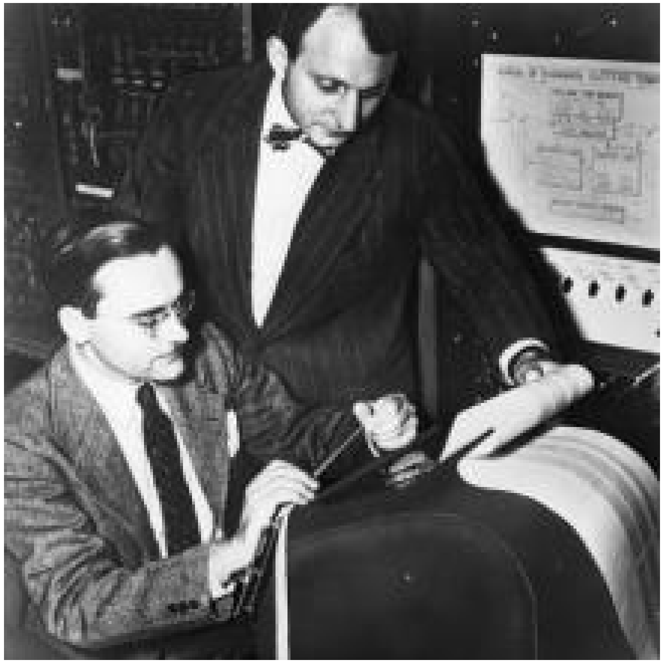
- Before Von Neumann's computer project (based on a stored-program concent), three computers were built:
  - 1936, Zuse's Z1 (1st binary digital computer) in Germany
  - 1937, Atanasoff and Berry's ABC (1st electronic digital computer) in Iowa State University
  - 1943, ENIAC based on ABC in UPenn
- A milestone Von Neumann made in computing history: "First Draft of a Report to the EDVAC (Electronic Discrete Variable Automatic Computer", 1945. (a proposal to US Army) before his IAS Computer Project

Principle of Von Neumann Model

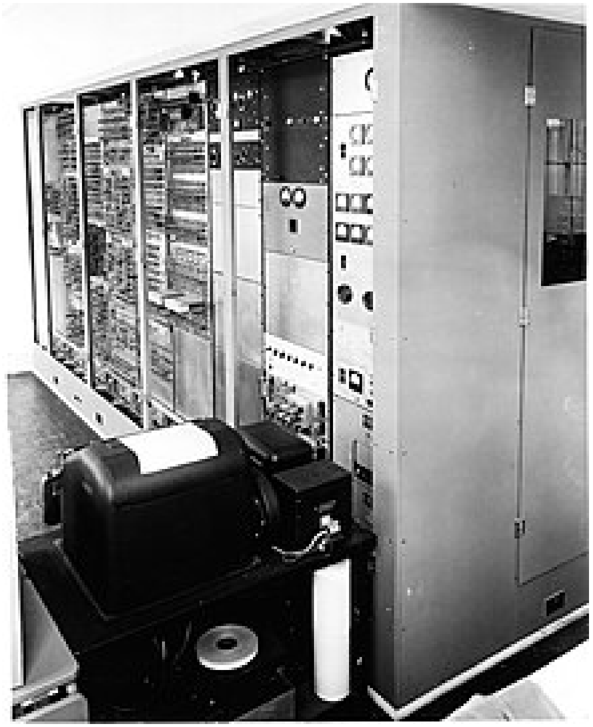
- **Computing unit** is advanced to high-end CPU, multicores, GPU, FPGA, and many other types
- **Memory unit** is advanced to DRAM, SSD, HDD, NVM, …
- **Communication channel** is advanced to Ethernet, fast interconnection networks, Internet, wireless, ….
- But the principle of Von Neumann model remains the same

1950: First Stored-Program Machine: SEAC

- SEAC (Standards Eastern Automatic Computer) was developed by National Bureau of Standards in 1950 (NBS is NIST today)
  - I/O mechanisms for data processing
  - time-sharing (running multiple programs in the computer concurrently)
  - An interconnection of two computers in 1954 for data communication
  - A graphical display.



Chief architects **Ralph Slutz** and **Samuel Alexander**



The SEAC Machine in 1950

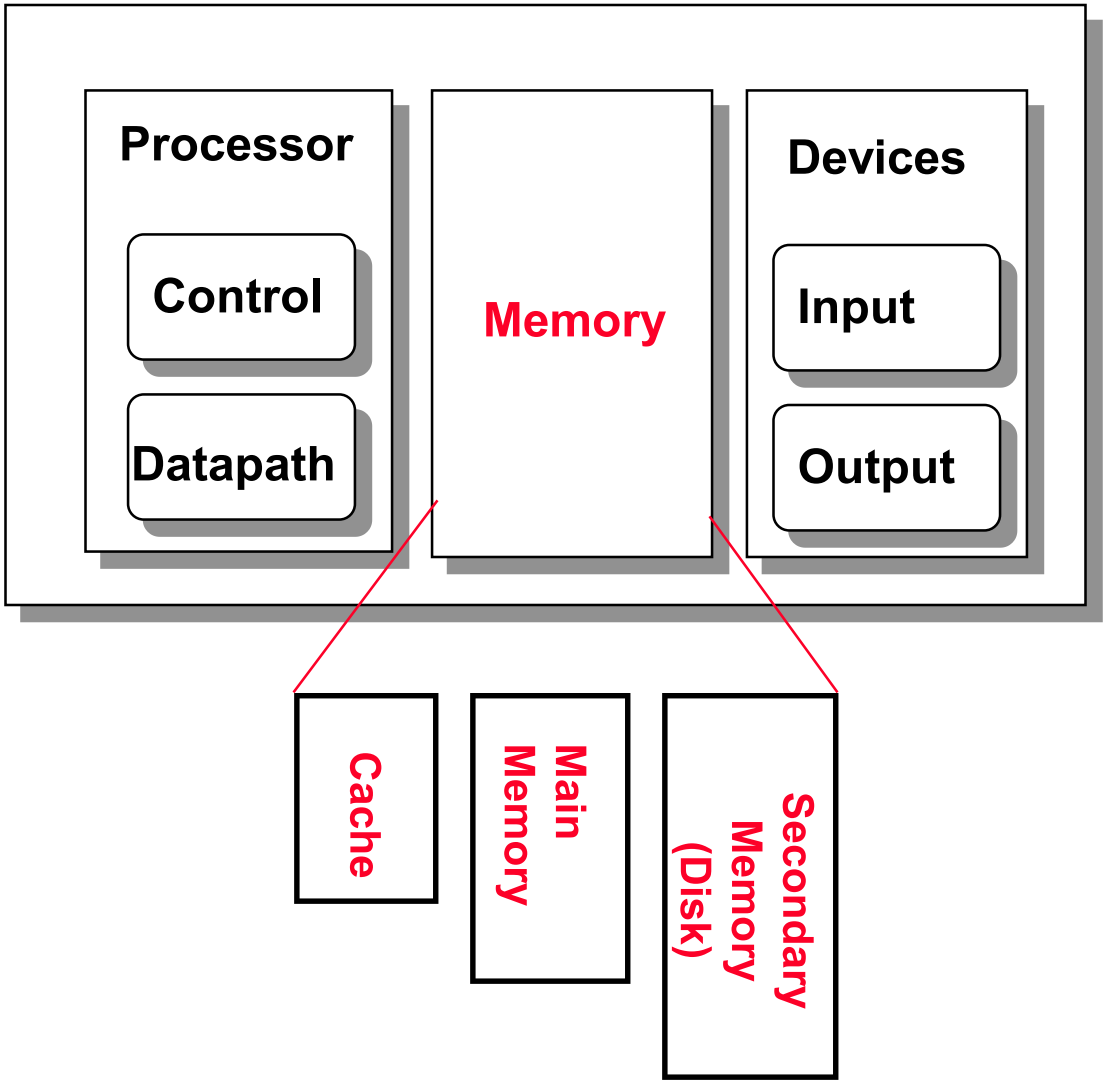
Dr. Ralph Slutz (1917-2005)

Slutz's major milestones in Computer Architecture:

- **Von Neumann's Electronic Computer Project,** Implementation of Von Neumann model, Chief Engineer, 1946-1948
- **SEAC**, the first operational architecture of Von Neumann Model in the world, Chief Architect, 1948-1952
- **COADS**, Comprehensive Ocean-Atmosphere Data Set, Chief Architect, University of Colorado at Boulder, 1980-1990.
- Physicist, National Institute of Standard and Technology, 1952-1980
- Silver Medal, for contribution to SEAC, US Department of Commerce, 1953

Ralph Slutz and Xiaodong Zhang in the Commencement of University of Colorado at Boulder, USA, 1989.

Review: Major Components of a Computer



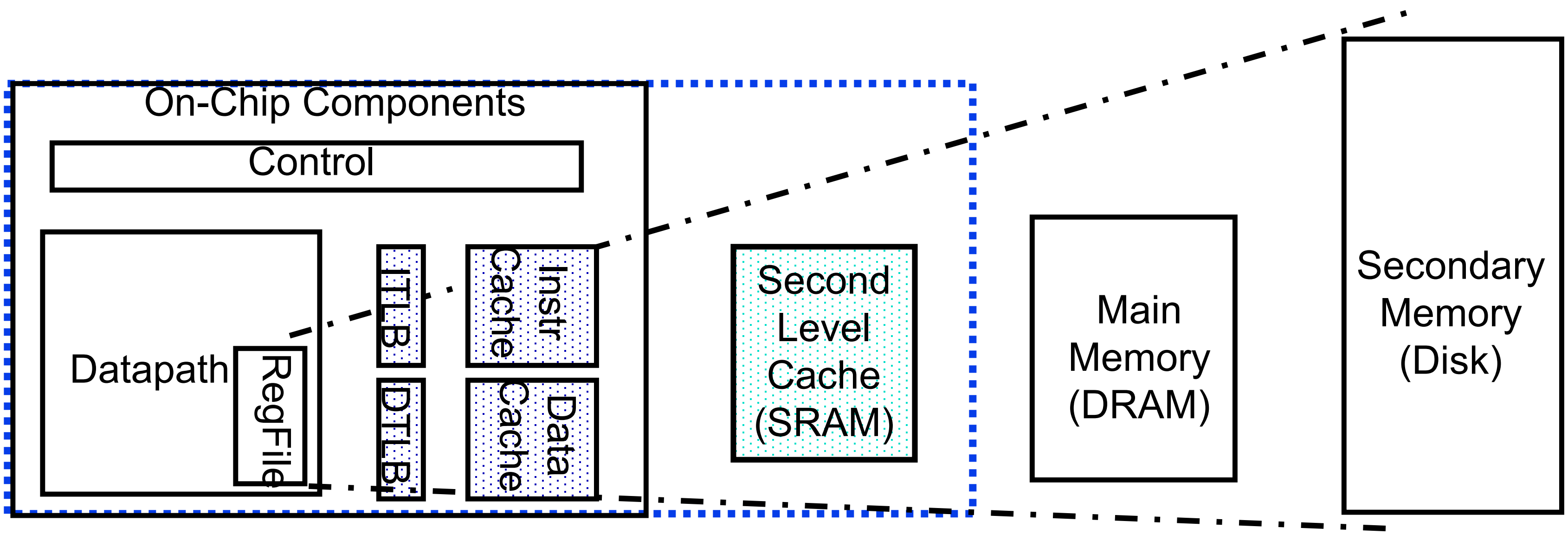
The Memory Hierarchy Goal

- Fact: Large memories are slow and fast memories are small

- How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?
  - With hierarchy
  - With parallelism

A Typical Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology

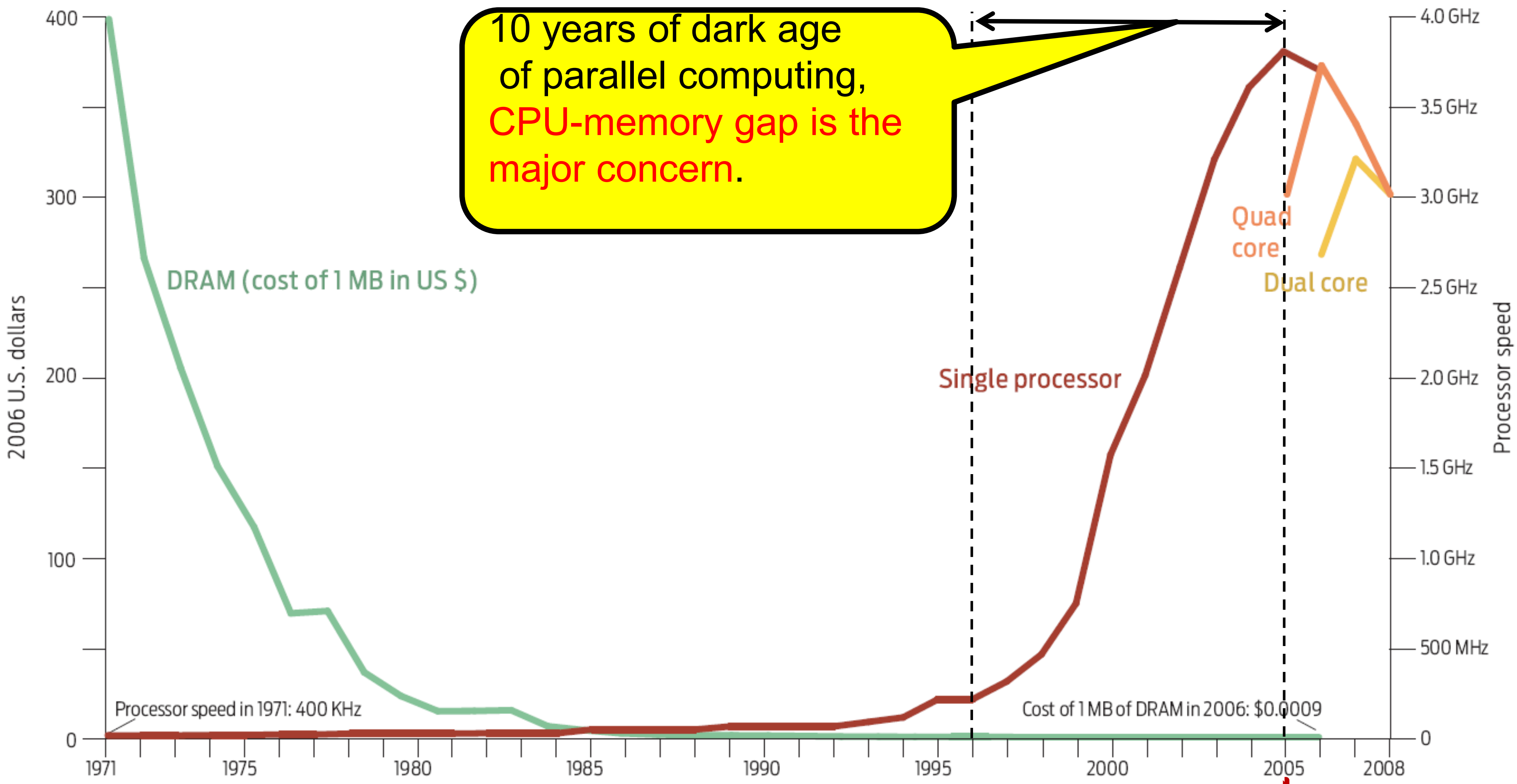


Speed (%cycles): ½'s 1's 10's 100's 10,000's

Size (bytes): 100's 1MB's(L1),10MBs(L2),100MB's(L3) GB's TB's

Cost: highest lowest

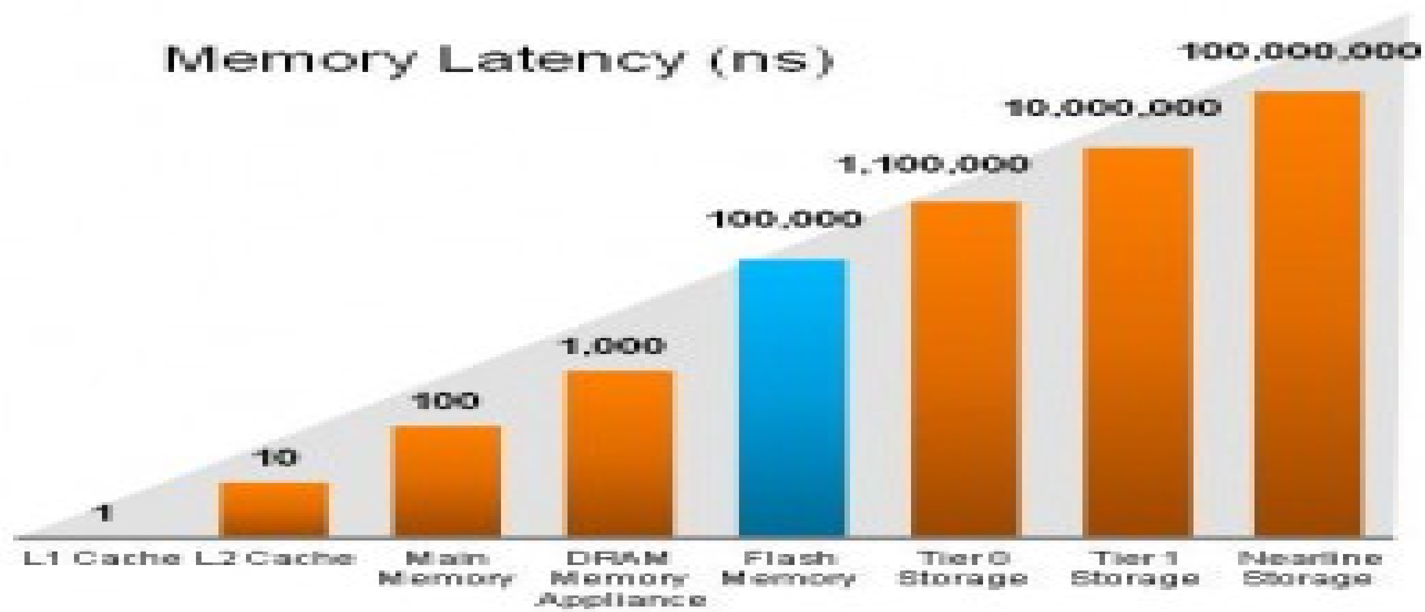
CPU and memory updates under Moore's Law (IEEE Spectrum, 2008)



25 year of golden age of parallel computing

New era of multicore computing
Memory problem continues

1 to 100 Millions Times Delay Today for Disk Accesses



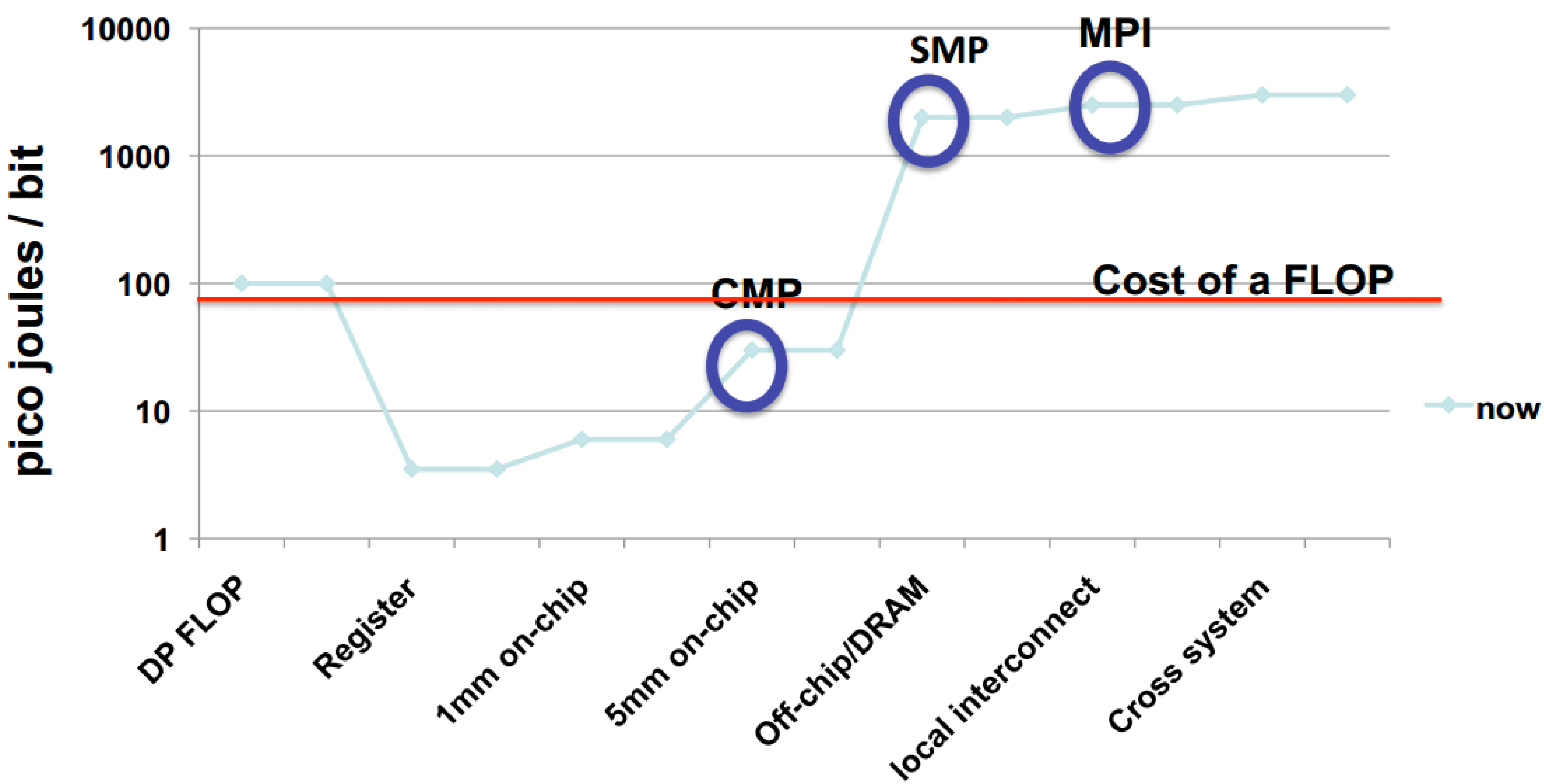
Tier 0 storage: high-end disks connected by fast switches for transactional data

Tier 1 storage: SATA disk arrays for mission critical data

Tier 2 storage: Disk arrays for seldom used archived data

Jeff Richardson, “Bridging the I/O Gap”, The Data Center Journal, 2012

The Cost of Data Movement



DP Flop: double precision flop; **1 mm**= 1 $\times 10^{-3}$ m, **1nm =** 1 $\times 10^{-9}$ m, **CMP:** chip multiprocessor
SMP: shared-memory processor**, MPI:** message passing interface for cluster computing

The Memory Hierarchy: Why Does it Work?

- Temporal Locality (locality in time)
  - If a memory location is referenced, then it will tend to be referenced again soon

  $\Rightarrow$ Keep most recently accessed data items closer to the processor

- Spatial Locality (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

  $\Rightarrow$ Move blocks consisting of contiguous words closer to the processor

The Memory Hierarchy: Terminology

- Block (or line): the basic data unit that is present (or not) in a cache
- Hit Rate: the fraction of memory accesses found in a level of the memory hierarchy
  - Hit Time: Time to access that level, which consists of

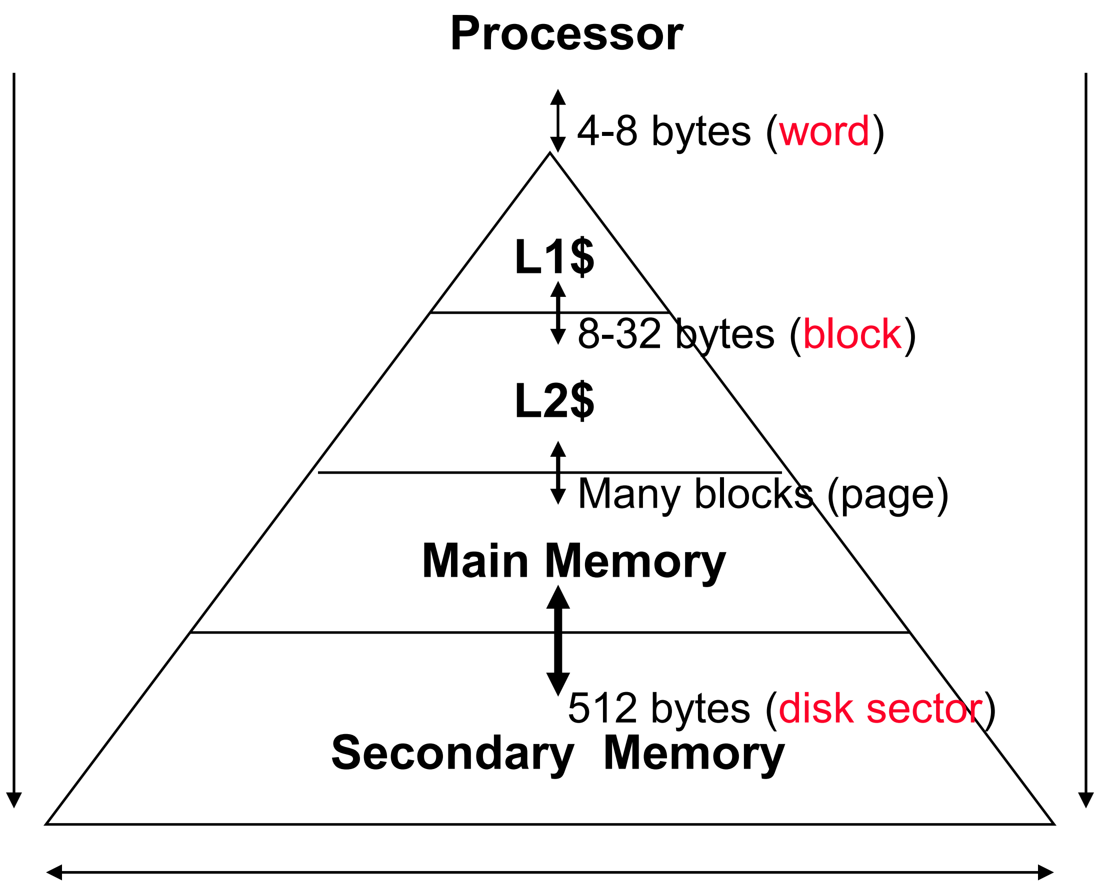
    Time to access the block + Time to determine hit/miss
- Miss Rate: the fraction of memory accesses *not* found in a level of the memory hierarchy $\Rightarrow$ 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in that level with the corresponding block from a lower level which consists of

    Time to access the block in the lower level + Time to transmit that block to the level that experienced the miss + Time to insert the block in that level

Hit Time << Miss Penalty

Characteristics of the Memory Hierarchy

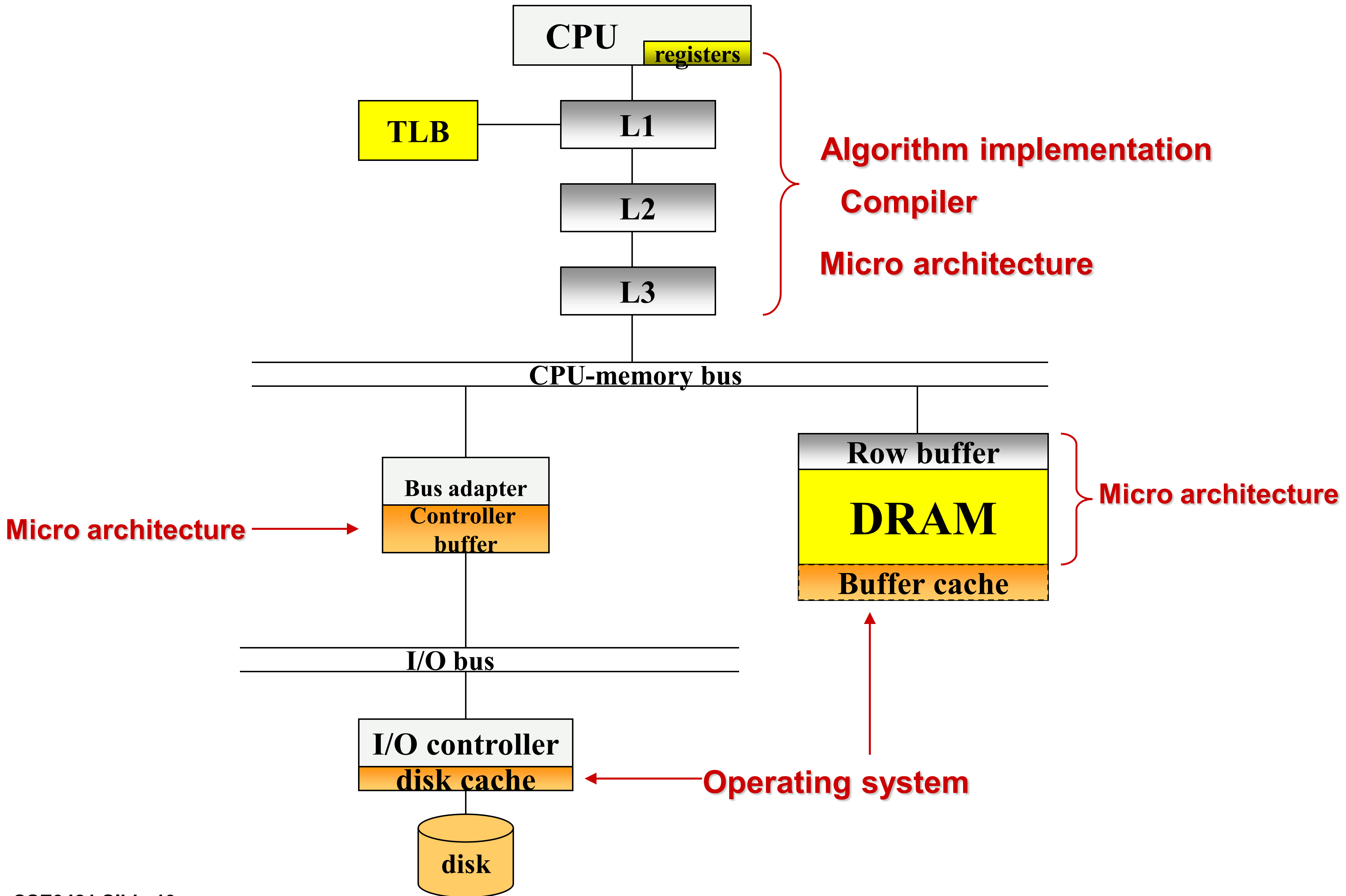
Increasing distance from the processor in access time



(Relative) size of the memory at each level

Inclusive– what is in L1$ is a subset of what is in L2$ is a subset of L3 cache is what is in MM that is a subset of is in SM

Where are Buffers in Deep Memory Hierarchy



Data Size: from small to big

- KB (kilobyte, 2^{10} Bytes approx. = 10^3 Bytes, thousand)
- MB (megabyte, 10^6 Bytes, million)
- GB (gigabyte, 10^9 Bytes, billion)

- TB (terabyte, 10^{12} Bytes, trillion)
- PB (petabyte, 10^{15} Bytes, quadrillion)
- EB (exabyte, 10^{18} Bytes, quintillion)
- ZB (zettabyte, 10^{21} Bytes, sextillion)

Famous Historical Quotes on Computers

- "I think there is a world market for maybe five computers".
  - T. J. Watson, Board Chair and CEO, IBM, **1943**
- "There is no reason for any individual to have a computer in their home."
  - Ken Olsen, Founder and CEO of DEC, **1977**
- "640K (memory) ought to be enough for anyone"
  - Bill Gates, **1981** (IBM PC-1 with 64K memory using DOS)
- "Apple was making a big mistake on a device called iPhone".
  - Steve Ballmer, Microsoft CEO, **2007** (Apple iPhone was released this year)

How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (assembly code programmer)
- cache ↔ main memory (same memory address)
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB in L1 cache)
  - by users (directly saving files in disks)

Basics of Hardware Caches

- A data item is referenced by its memory address.
- It is first searched in the cache.
- Three questions cover all the cache operations:
  - How do we know it is in the cache?
  - If it is (a hit), how do we find it?
  - If it is not (a miss), how to replace the data if the location is already occupied?

Direct-Mapped Cache

- The simplest, but efficient and commonly used. (Maurice Wilkes, IEEE TC 1965, a two-page paper)

- Each access is mapped to exactly one location.

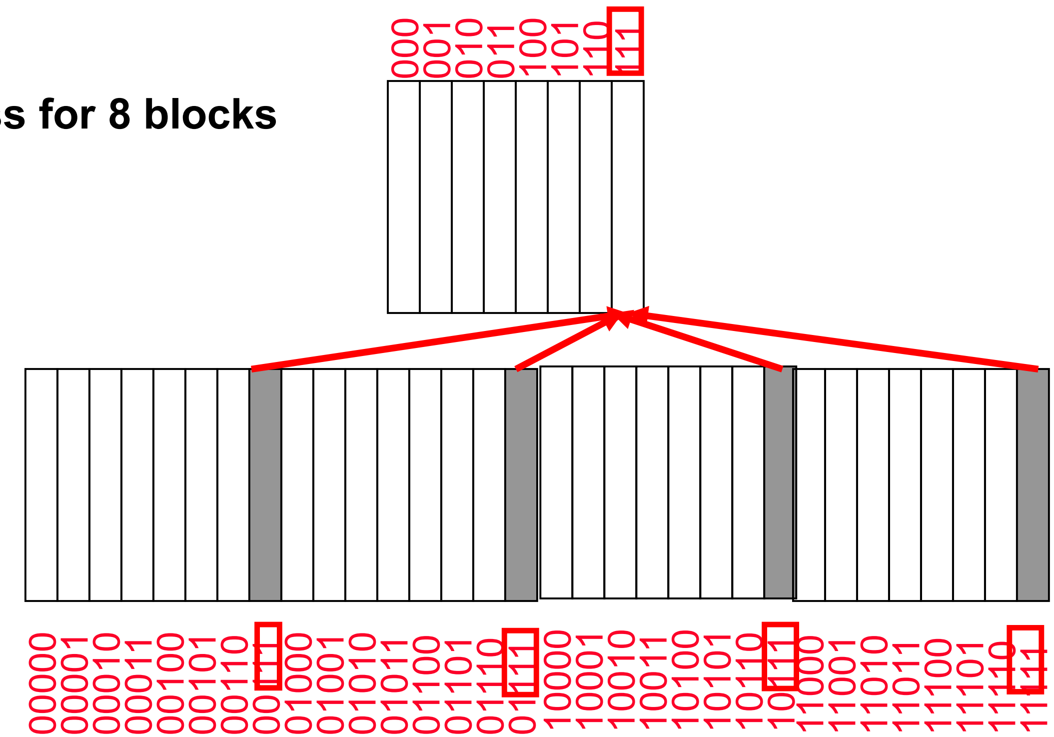
- The mapping follows:
  - (memory address) mod (number of cache blocks)

- The original block (cache line) has 4 bytes (a word), but it is increasingly longer for spatial locality.

An Example of a Direct Mapped-cache

Cache: **3-bit address for 8 blocks**

Memory: 5 bit add for 32 locations



7_{10} $(00111)_2$ Mod $8_{10} =$ 111_2

15_{10} $(01111)_2$ Mod $8_{10} =$ 111_2

23_{10} $(10111)_2$ Mod $8_{10} =$ 111_2

31_{10} $(11111)_2$ Mod $8_{10} =$ 111_2

The Fact of Direct-mapped Caches

- If the number of cache blocks is a power of 2, the mapping is exactly the low-order $\log_2$ (cache size in blocks) bits of the address.

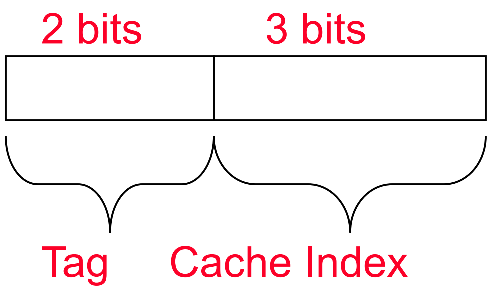
- If cache = 2^3 = 8 blocks, the 3 low-order bits of the memory address are directly mapped addresses.

- The lower-order bits are also called cache index.

Tags in Direct-Mapped Caches

- A cache index for a cache block is not enough because multiple addresses will be mapped to the same block.

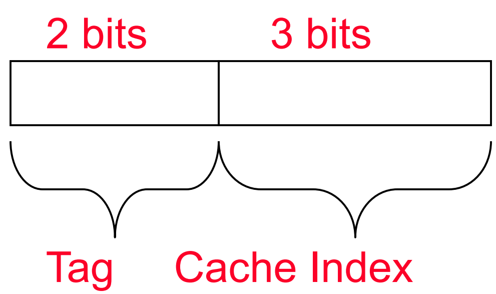
- A ``tag" is used to make this distinction. The upper portion of the address forms the tag:



- If both the tag and index are matched between memory and cache, a ``hit" happens.
- A valid bit is also attached for each cache block.

How are Tag bits set?

- Specifically, a tag is used to distinguish the blocks sharing the same direct mapping address.
- For a three 8-block cache, the last three bit address have the 4 opportunities to directly map to the same block address, thus tag is set 2-bits for 4 tags.



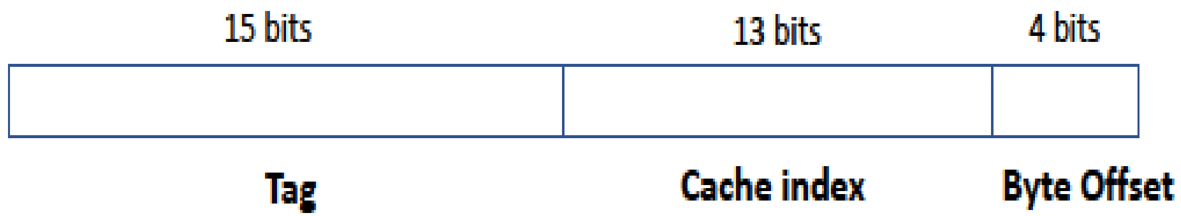
- In general, the difference between the number of the memory address bits and number of cache memory address bits is the number of tag bits.
  - Tag (bits) = Mem (bits) – Cache (bits)

Allocation of Tag, Cache Index, and Offset Bits

- Cache size: 128 Kbytes
- Block size: 16 Bytes (4 bits for offset)
- 128 Kbytes = 8 K blocks = 2^{13} (13 bits for index)
- For a 32-bit memory address: 15 bits left for tag.

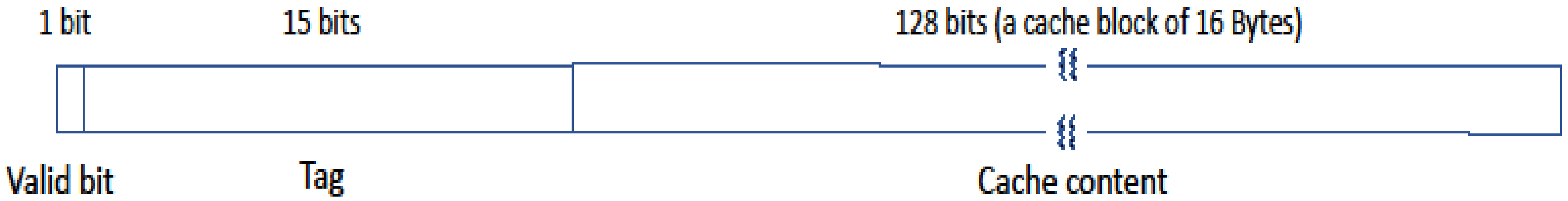
- Each cache line (block) contains a total of 144 bits:
  - 128 bits (data of 16 Bytes)
  - 15 bits (tag)
  - 1 bit for valid bit.

Memory Address format vs one cache block



(a) Memory Address Format for Direct-Mapped Cache

(1) Cache storage size = 128 KB, and (2) Cache block size = 16 Bytes



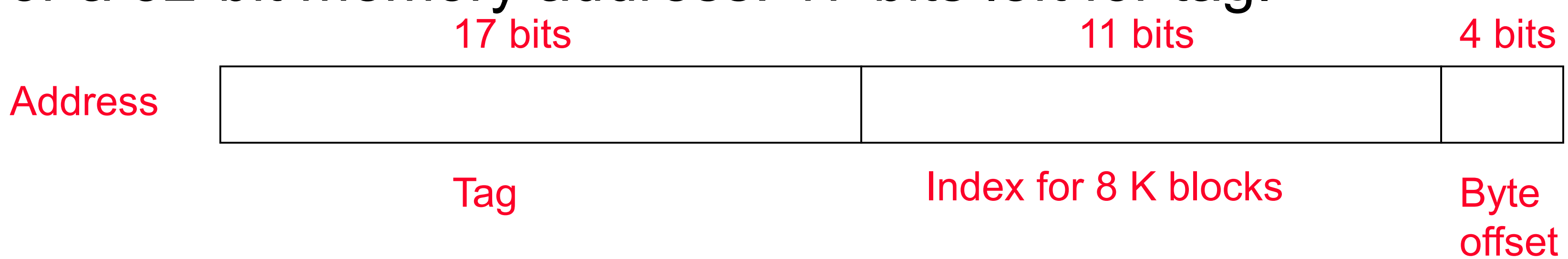
(b) One cache block only

Set-associative and Fully-associative Caches

- Direct-mapping one location causes high miss rate.
- How about increasing the number of possible locations for each mapping?
- The number of locations is called ``associativity". A set contains more than one location.
- associativity = 1 for direct-mapped cache
- Set-associative cache mapping follows:
  - (memory address) mod (number of sets)
- Associativity = total number of blocks for fully associative cache.

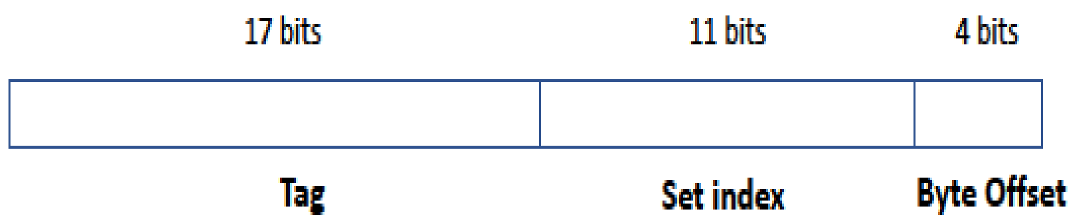
Allocation of Tag, Set Index, and Offset Bits

- Cache size: 128 Kbytes
- Block size: 16 Bytes (4 bits for offset)
- 4-way set associative cache, each set has 4 blocks
- 128 Kbytes = 2K sets = 2^{11} (11 bits for index)
- For a 32-bit memory address: 17 bits left for tag.



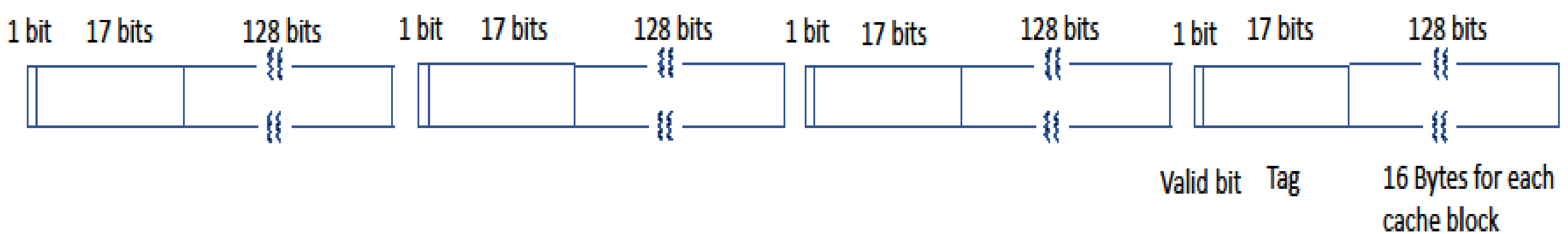
- Each cache set contains 4 following cache blocks
  - 128 bits (data of 16 Bytes)
  - 17 bits (tag)
  - 1 bit for valid bit.

Memory Address format vs one cache block



(a) Memory Address Format for 4-Way Set Associates Cache

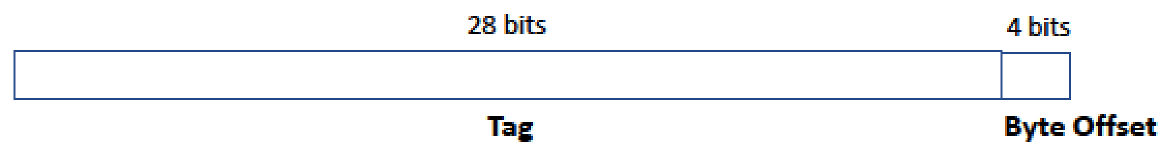
(1) Cache storage size = 128 KB, and (2) Cache block size = 16 Bytes



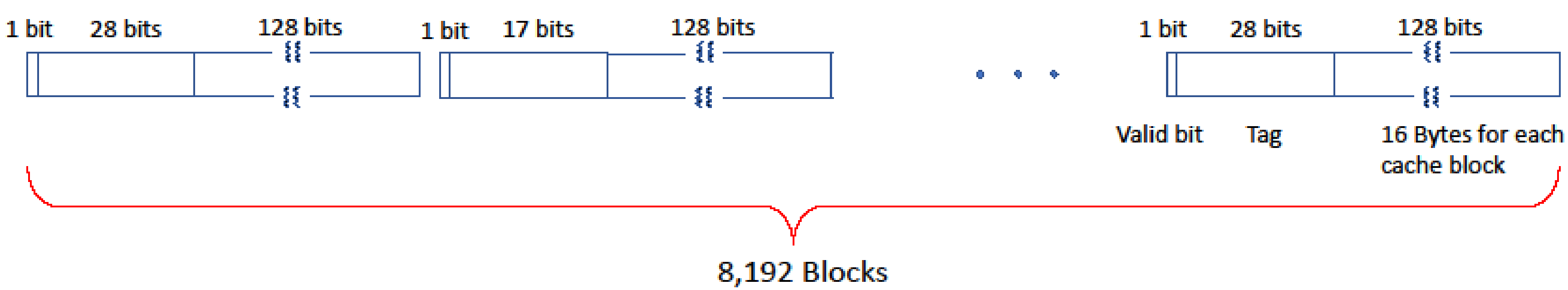
(b) 4 cache blocks in each set

Allocation of Tag, No Index, and Offset Bits

- Cache size: 128 Kbytes
- Block size: 16 Bytes (4 bits for offset)
- Fully-associative cache, only 1 set has 8K blocks
- No index is needed, 28 bits for tag



(a) Memory Address Format for Fully-Associative Cache
(1) Cache storage size = 128 KB, and (2) Cache block size = 16 Bytes



(b) All cache blocks in the single set

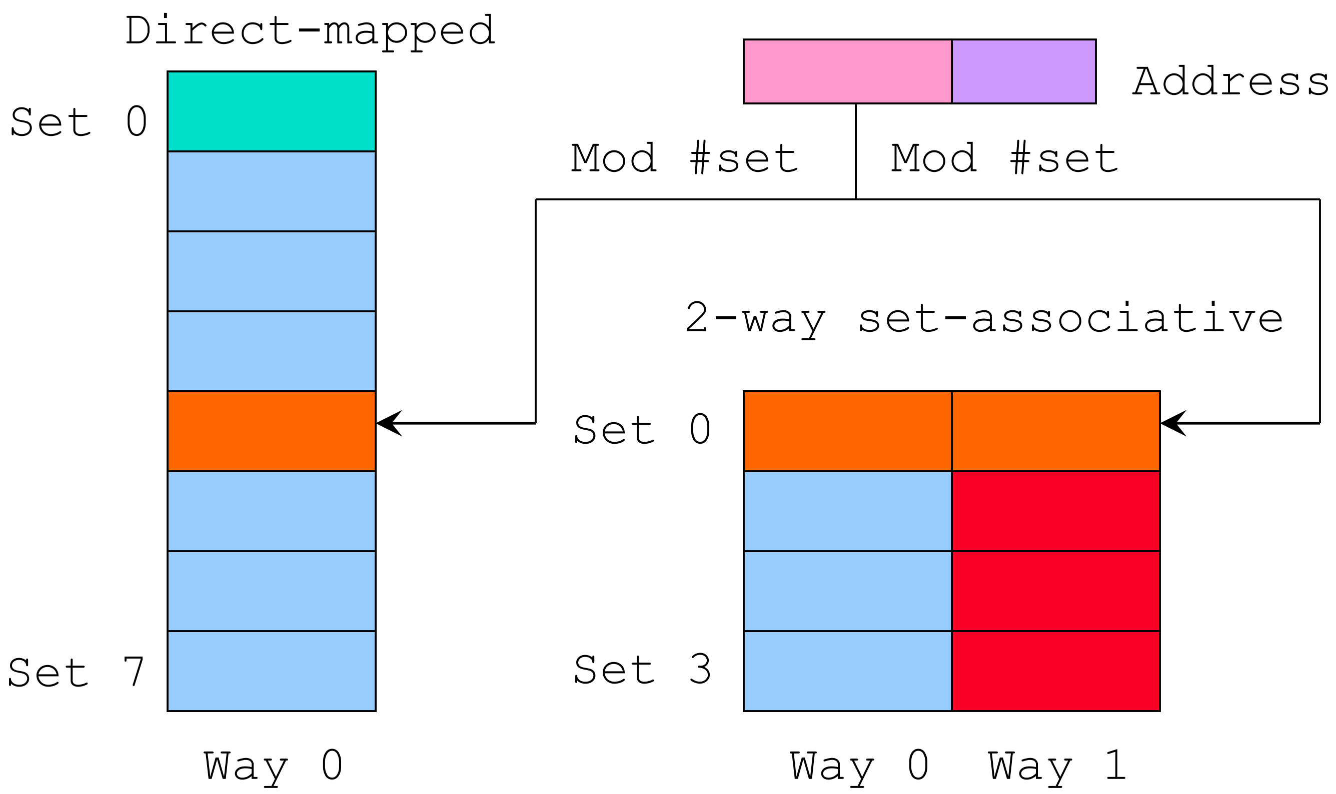
Sir Maurice Wilkes (1913-2010)

Wilkes' milestones in computer architecture:

- **EDSAC** (Electronic Delay Storage Automatic Calculator), the second stored-program computer, 1951
- **Microprogramming**, having become a standard mechanisms for firmware, and control units, 1953
- **CPU cache**, direct mapped cache and set associative cache, 1965
- Professor in Cambridge University, 1936-2010
- Turing Award, 1967

Maurice Wilkes and Xiaodong Zhang in ISCA 2002, Alaska, USA

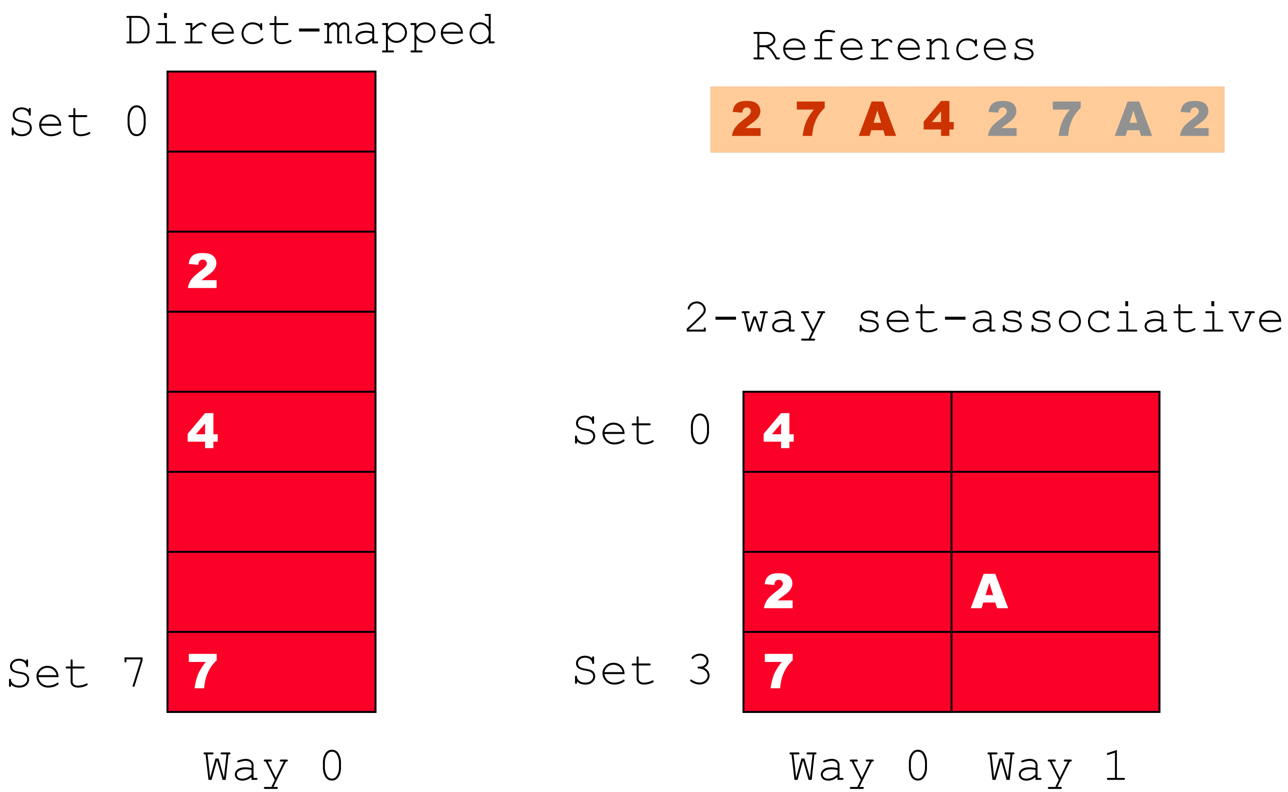
Direct-mapped vs. Set-associative



Cache Accesses

References

2 7 A 4 2 7 A 2



7 misses

4 misses

Counting misses and hits

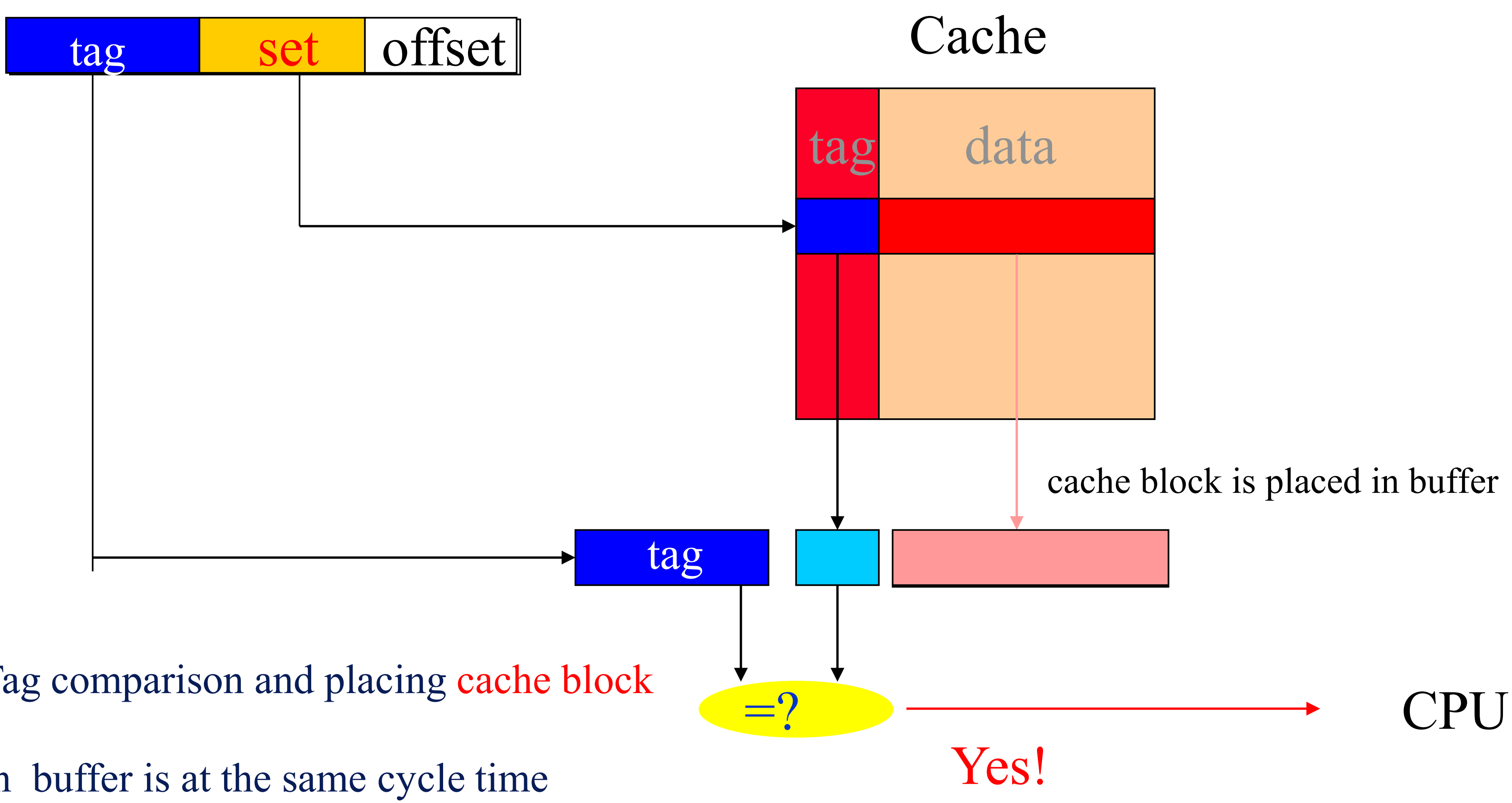
- Direct Map Cache:
  - 2: cold miss
  - 7: cold miss
  - A: conflict miss
  - 4: cold miss
  - 2: conflict miss
  - **7: hit**
  - A: conflict miss
  - 2: conflict miss

  1 hit and 7 misses

- 2-way Set-Associative Cache:
  - 2: cold miss
  - 7: cold miss
  - A: cold miss
  - 4: cold miss
  - **2: hit**
  - **7: hit**
  - **A: hit**
  - **2: hit**

  4 hits and 4 misses

Direct-mapped Cache Operations: minimum hit time

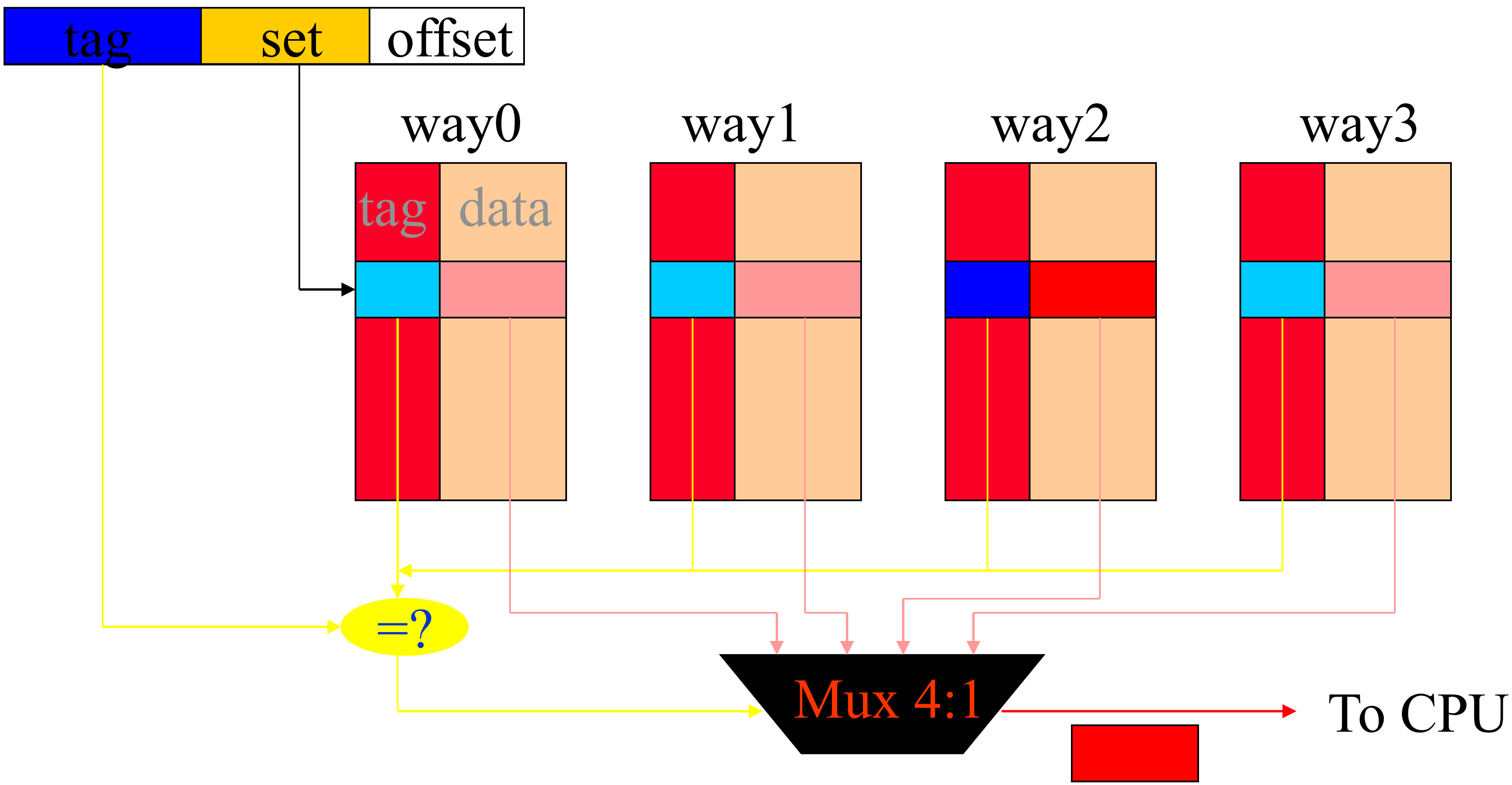


Tag comparison and placing cache block

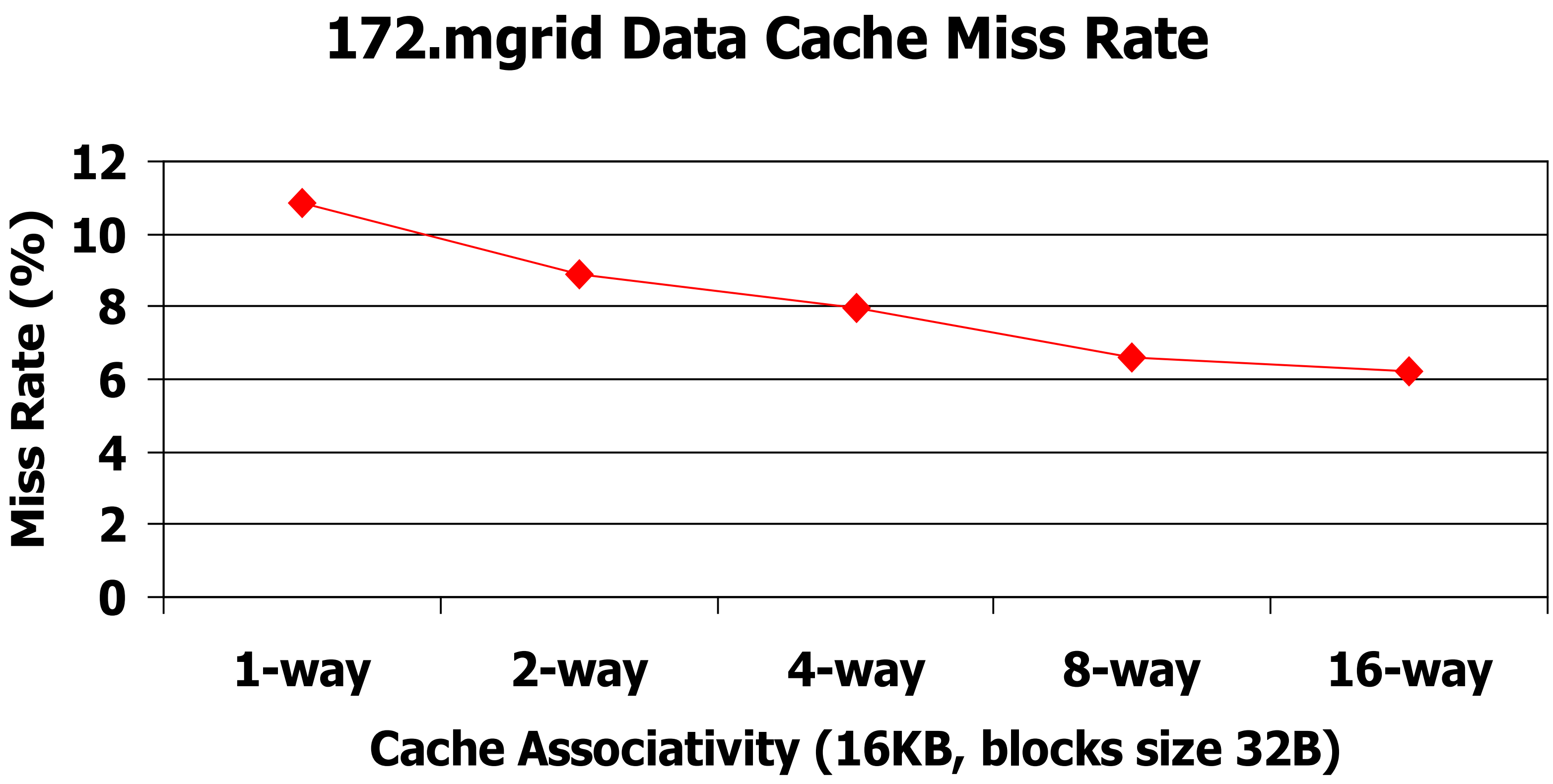
in buffer is at the same cycle time

Set-associative Cache: delayed hit time

Latency source: **tag comparisons** (in parallel) have to be done before selecting the right block by **multiplexor**



Set-associative Caches Reduce Miss Ratios



172.mgrid: SPEC 2000, multi-grid solver

Trade-offs between High Hit Ratios and Low Latency

- Set- associative cache achieves high hit-ratios: 30% higher than that of direct-mapped cache.
- But it suffers high access times due to
  - Multiplexing logic delay during the selection.
  - Tag checking, selection, and data dispatching are sequential.
- Direct-mapped cache loads data and checks tag in parallel: minimizing the access time.
- Can we get both high hit ratios and low access times?
- The Key is the Way Prediction: speculatively determine which way is the hit so that only that way is accessed.

Cache Summary

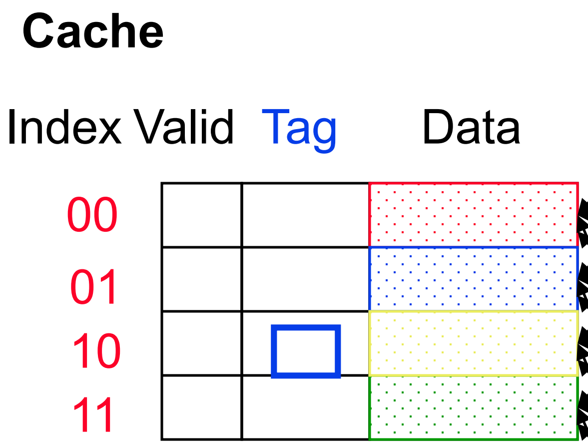
- Two questions to answer (in hardware):
  - Q1: How do we map to its cache location?
  - Q2: How do we know if the cache block is the right one?

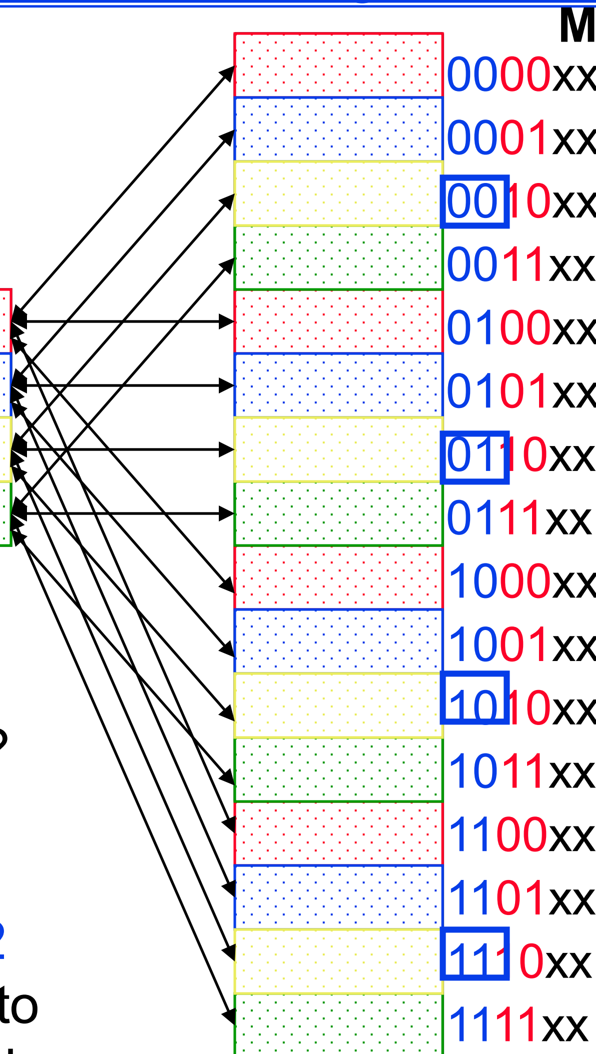
- Direct mapped
  - Each memory block is mapped to exactly one block in the cache
    - Multiple memory addresses map to the same blocks (conflicts)
  - Address mapping (to answer Q1):

    (block address) modulo (# of blocks in the cache)

  - Using a tag associated with each cache block that contains the address information (the upper portion of the address) required to identify the block (to answer Q2)

Mapping between 16 memory add to 4-cache blocks





4-byte for each block
Two low order bits are used for the offset

Q1: How to map to cache?

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)

Q2: Is it the right one?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

(block address) modulo (# of blocks in the cache)

Direct Mapped Cache

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

0 miss

Tag	
00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss, replacement

01 ~~00~~	~~Mem(0)~~ 4
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

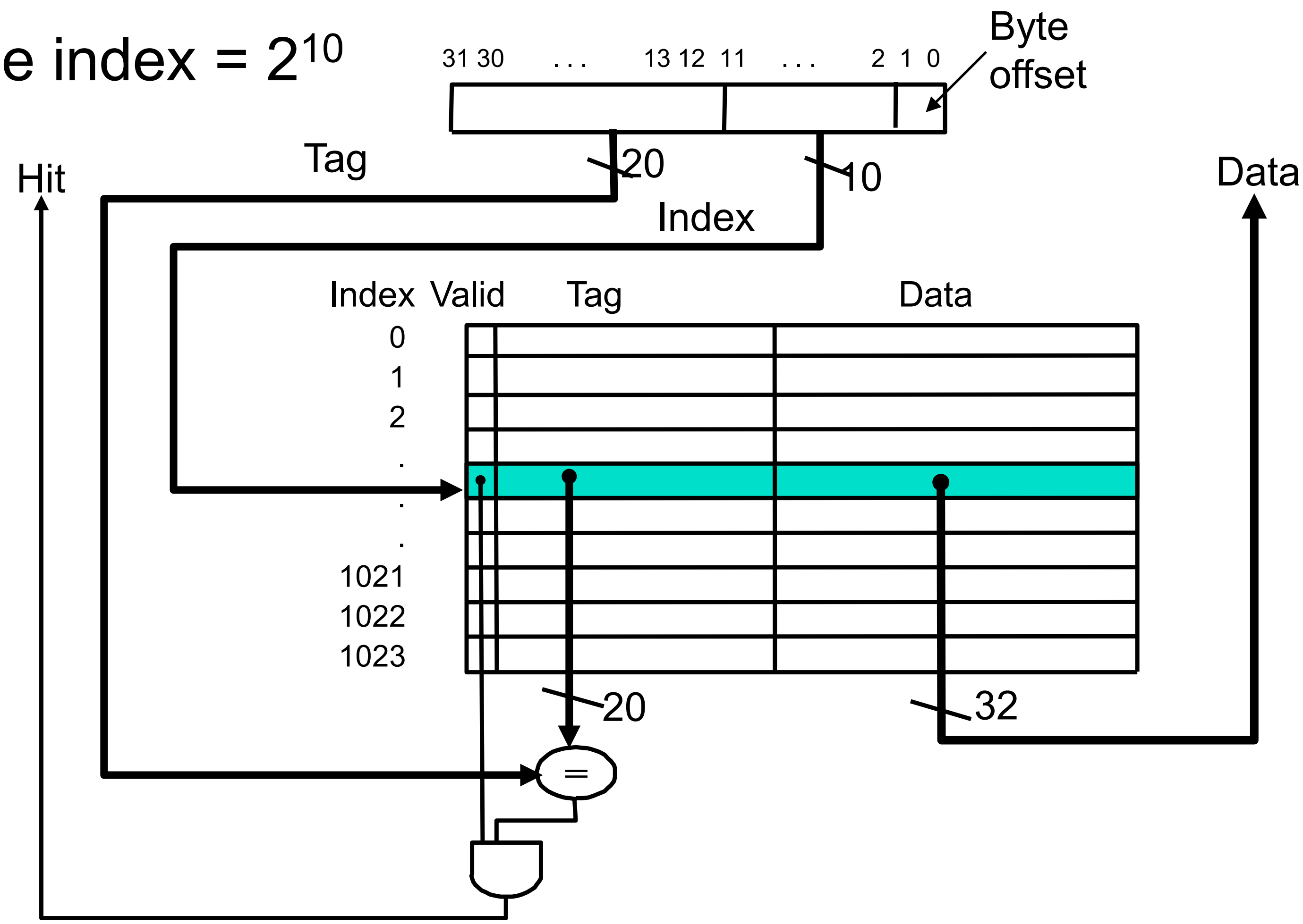
15 Miss, replacement

01	Mem(4)
00	Mem(1)
00	Mem(2)
11 ~~00~~	~~Mem(3)~~ 15

- 8 requests, 6 misses

MIPS Direct Mapped Cache Example

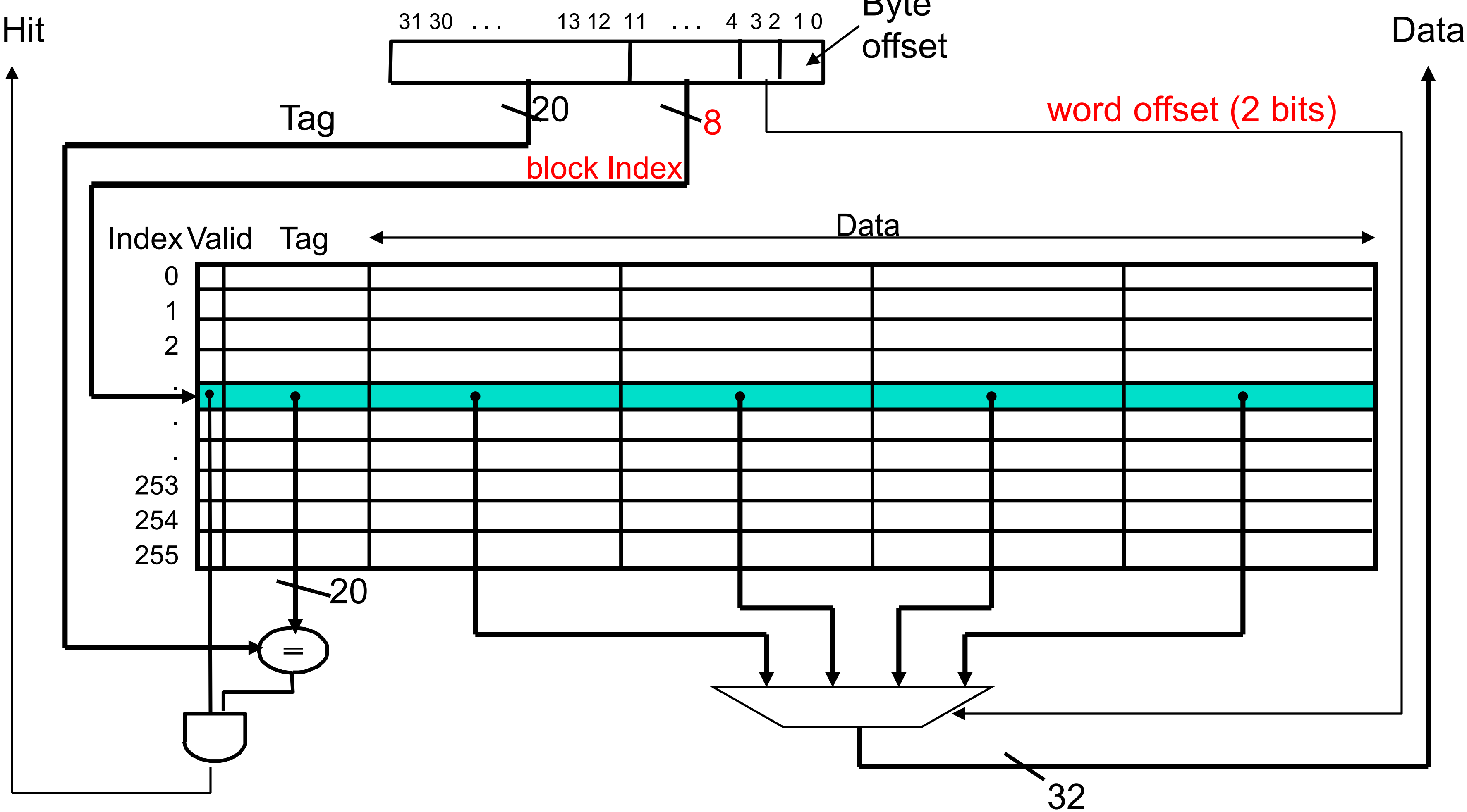
- One-word blocks, cache size = 1K words (or 4KB)
- Cache index = 2^{10}



Minimum latency due to overlapping tag checking and preparing data

Multi-Block Direct Mapped Cache

- Four words/block, cache size = 1K words
- Cache index = 2^8 =256, each for 4 words



If bits 0-3 are used for byte offset only, what is the difference?

Long cache blocks

- The byte offset bits determine the length of the cache block
  - 2 bits (bits 0-1) for 4 bytes, 3 bits (bits 0-2) for 8 bytes, …
- 4 bits (Bits 0-3) for 16 bytes, 4 words long
  - 256 cache blocks each has 4 words
- If Bits 0-1 for byte offset, and bits 2-3 for word selection
  - Each of 256 cache indices (2^8. 8 bits) maps to a set of 4 words. Bits 2-3 determines which word to be selected by a multiplexor

Taking Advantage of Spatial Locality

- Let cache block hold more than one word (prefetching)

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

01 ~~00~~	~~Mem(1)~~ 5	~~Mem(0)~~ 4
00	Mem(3)	Mem(2)

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

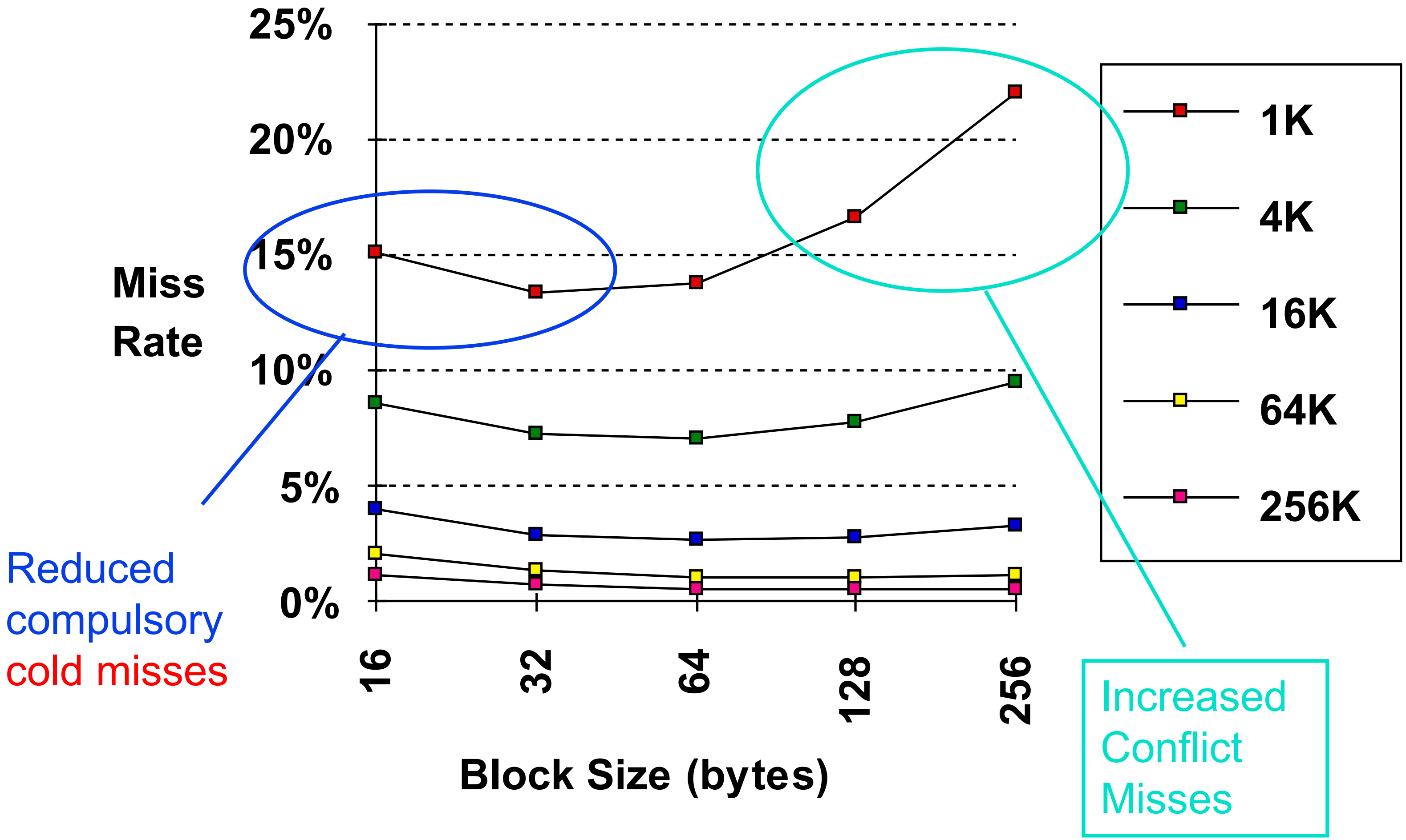
01	Mem(5)	Mem(4)
11 ~~00~~	~~Mem(3)~~ 15	~~Mem(2)~~ 14

- 8 requests, 4 misses

Sources of Cache Misses

- Compulsory (cold start or process migration, first reference, cold misses):
  - First access to a block, “cold” fact of life, not a whole lot you can do about it. If you are going to run “millions” of instruction, compulsory misses are insignificant
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity, but also increase access latency

Miss Rate vs Block Size vs Cache Size



- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing conflict misses). Replacement will have to evict useful words in the block.

Cache Field Sizes

- The number of bits in a cache includes both the storage for data, for tags, and valid bit
  - 32-bit memory address (byte addressable)
  - For a direct mapped cache with 2^n blocks, *n* bits are used for index
  - In general, for a cache size of 2^n blocks (2^{n+m} bytes), n is cache index, and m is byte offset
  - For example, for a cache size of 2^n words (2^{n+2} bytes), *n* bits are used to address the word within the block and 2 bits are used to address the 4 bytes within the word
- The total number of bits in a direct-mapped cache is then

$$2^n \times (\text{block size} + \text{tag field size} + \text{valid field size})$$

where 2^n is the total number of blocks, the rest variables are in bits

Cache Field Sizes (continued)

- How many total bits are required for a direct mapped cache with 16KB of data and 16-byte block size assuming a 32-bit address?
  - Memory address division:
    - 4 bits for byte offset (2^4 = 16 bytes)
    - 10 bits for cache index (2^{10} = 1024 cache blocks, each has 16 bytes)
    - The rest of the 18 bits are used for tags
  - The total number of bits is
    - 2^{10} (16 * 8 bits (16 bytes) + 18 bits (tag) + 1 bit (valid bit))
    - = 1024 * (128 + 18 +1)
    - = 1024 & 147 = 150,528 bits (147K bits)

Handling Cache Hits

- Read hits (I$ and D$)
  - this is what we want!

- Write hits (D$ only)
  - require the cache and memory to be consistent
    - always write the data into both the cache block and the next level in the memory hierarchy (write-through)
    - writes run at the speed of the next level in the memory hierarchy – so slow! – or can use a write buffer and stall only if the write buffer is full
  - allow cache and memory to be inconsistent
    - write the data only into the cache block (write-back the cache block to the next level in the memory hierarchy when that cache block is "evicted")
    - need a dirty bit for each data cache block to tell if it needs to be written back to memory when it is evicted – can use a write buffer to help "buffer" write-backs of dirty blocks

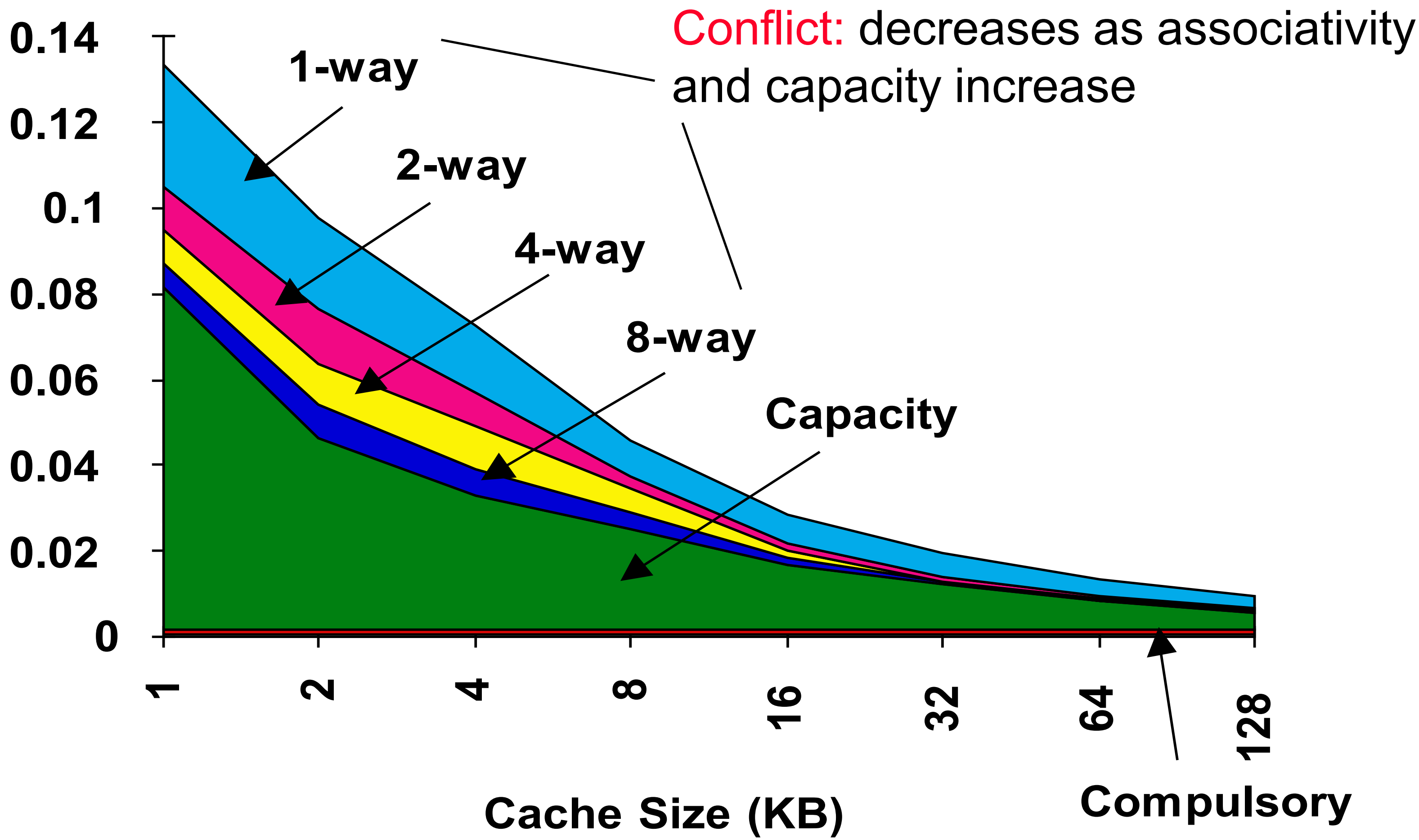
Write Allocate or No-write allocate

- Write miss: to be updated data is not in the cache
- On a write miss, we don't need the data, we can just forward the write request to update the memory
- If the cache allocates cache lines (bring them from the memory) on a write miss
  - It is write-allocate cache, otherwise, it is no-write allocate cache
- Advantages of Write Allocate
  - Data written will likely be read soon in the cache (temporal locality)
- Advantages of No-write allocate
  - The temporal locality may not be correct

3Cs Absolute Miss Rate (SPEC92)

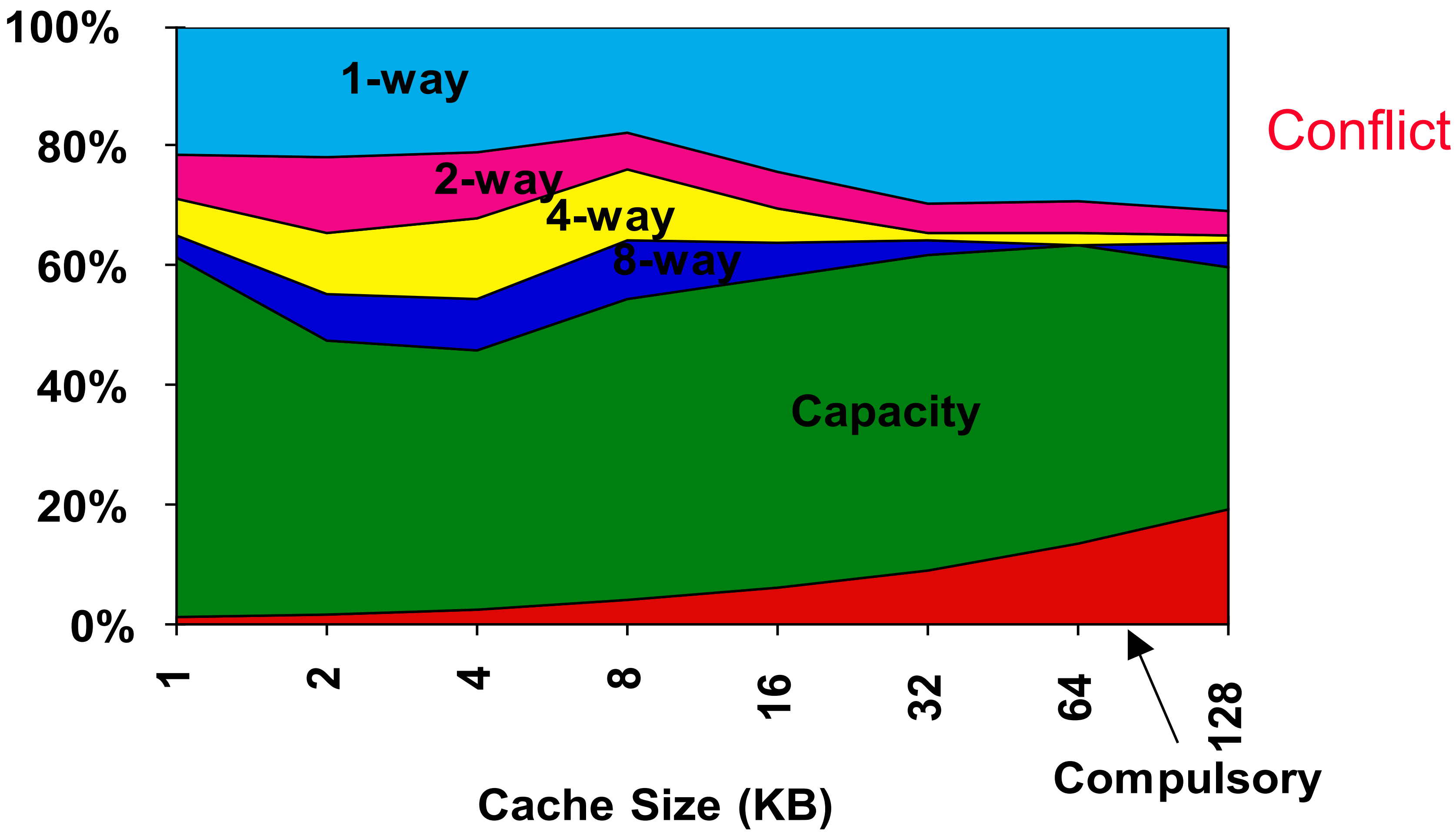
Compulsory (cold) misses are independent of capacity and associativity

Capacity misses decrease as the cache size increases.



3Cs Relative Miss Rate (SPEC92)

Conflict misses decrease as the associativity increases to a certain degree
Capacity misses decrease as the cache size increases
Cold misses increases as the capacity increases relatively



Measuring Cache Performance

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

$$\text{CPU time} = \text{IC}\ {\scriptstyle\text{(instruction count)}} \times \text{CPI} \times \text{CC}\ {\scriptstyle\text{(cycle time)}}$$

$$= \text{IC} \times \underbrace{(\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles})}_{\text{CPI}_{\text{stall}}} \times \text{CC}$$

$\text{CPI}_{\text{ideal}}$ is the CPI without memory accesses

- Within the memory hierarchy, CPI varies, and is workload dependent and memory system design dependent

Impacts of Cache Performance

- Relative cache penalty increases as processor performance improves (faster clock rate and/or lower CPI)
  - The lower the CPI_{ideal}, the higher the impact to memory stalls
- A processor with a CPI_{ideal} of 2, a 100 cycle miss penalty, 36% load/store instr's, and 2% I$ and 4% D$ miss rates

Memory-stall cycles = 2% × miss penalty for I$

+ load-store percentage ×(4% × miss penalty for D$)

Memory-stall cycles = 2% × 100 + 36% × 4% × 100 = 3.44

So CPI_{stalls} = 2 + 3.44 = **5.44**

more than twice the CPI_{ideal} ! 3.44/5.44 = 63% time in memory stall.

Note: only instruction fetching and load/store need memory accesses

More Examples of Cache Performance

- What if the CPI_{ideal} is reduced to 1? 0.5? 0.25?
  - CPI_{ideal} = 1, CPI_{stalls} = 4.44, 3.44/4.44 = 77% time in memory stall
  - CPI_{ideal} = 0.5, CPI_{stalls} = 3.94, 3.44/3.94 = 87% time in memory stall
  - CPI_{ideal} = 0.25, CPI_{stalls} = 3.69, 3.44/3.69 = 93% time in memory stall

- What if the D$ miss rate went up 1%? 2%?
  - For 3% D$ miss rate, Memory-stall cycles = **3%** × 100 + 36% × 4% × 100 = 4.44, , CPI_{stalls} = 2 + 4.44 = 6.44
  - For 4% D$ miss rate, Memory-stall cycles = **4%** × 100 + 36% × 4% × 100 = 5.44, , CPI_{stalls} = 2 + 5.44 = 7.44

- What if the processor clock rate is doubled (doubling the miss penalty because memory access latency remain the same)?

  Memory-stall cycles = 2% × **200** + 36% × 4% × **200** = 6.88

  So CPI_{stalls} = 2 + 6.88 = **8.88**

Average Memory Access Time (AMAT)

- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

AMAT = Time for a hit + Miss rate x Miss penalty

- What is the AMAT for a processor with a miss penalty of 50 clock cycles in a miss rate of 2%, and a cache access time of 1 clock cycle?

  - AMAT = 1 + 0.02 x 50 = 2 cycles

Reducing Cache Miss Rates #1

1. Allow more flexible block placement

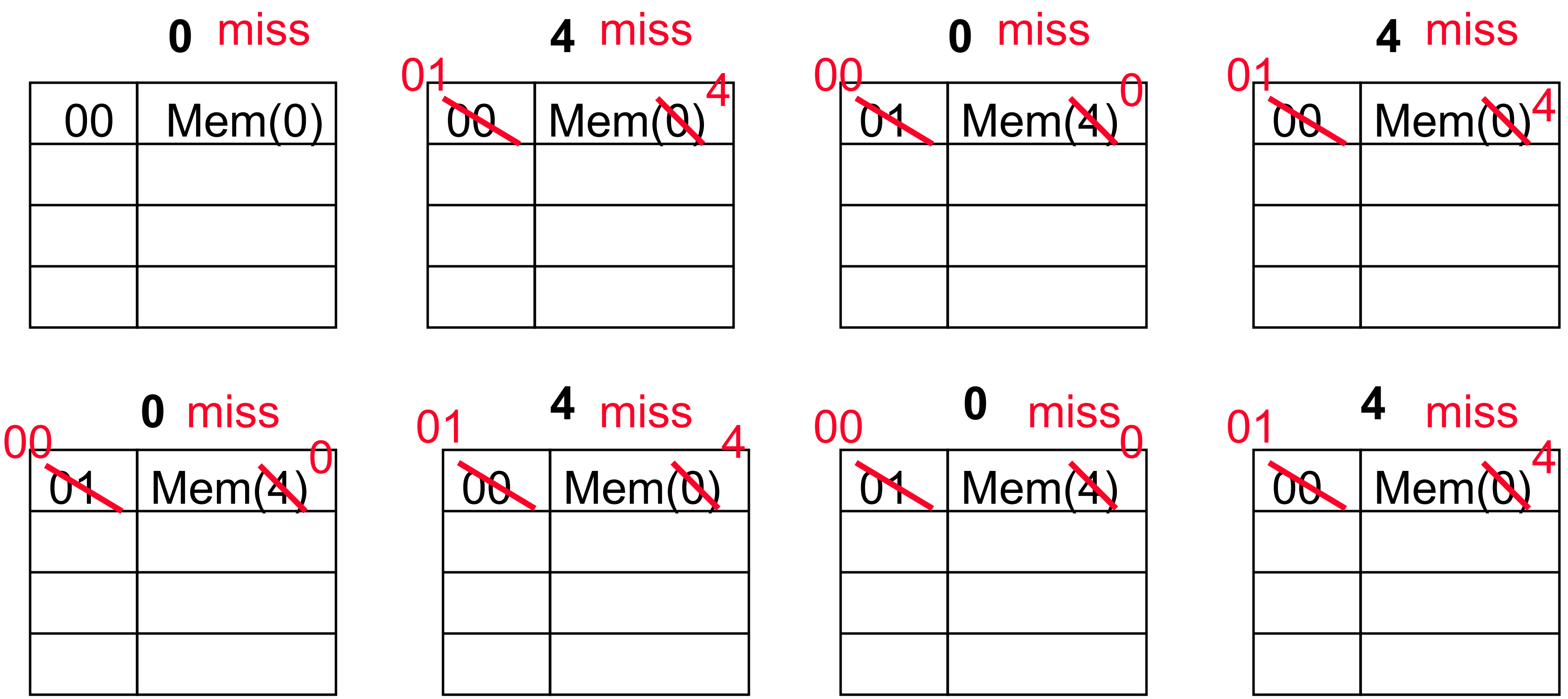
- In a direct-mapped cache a memory block maps to exactly one cache block
- At the other extreme, could allow a memory block to be mapped to *any* cache block – fully associative cache
- A compromise is to divide the cache into sets each of which consists of n "ways" (n-way set associative). A memory block maps to a unique set (specified by the index field) and can be placed in any way of that set (so there are n choices)

(block address) modulo (# sets in the cache)

Another Reference String Mapping

- Consider the main memory word reference string

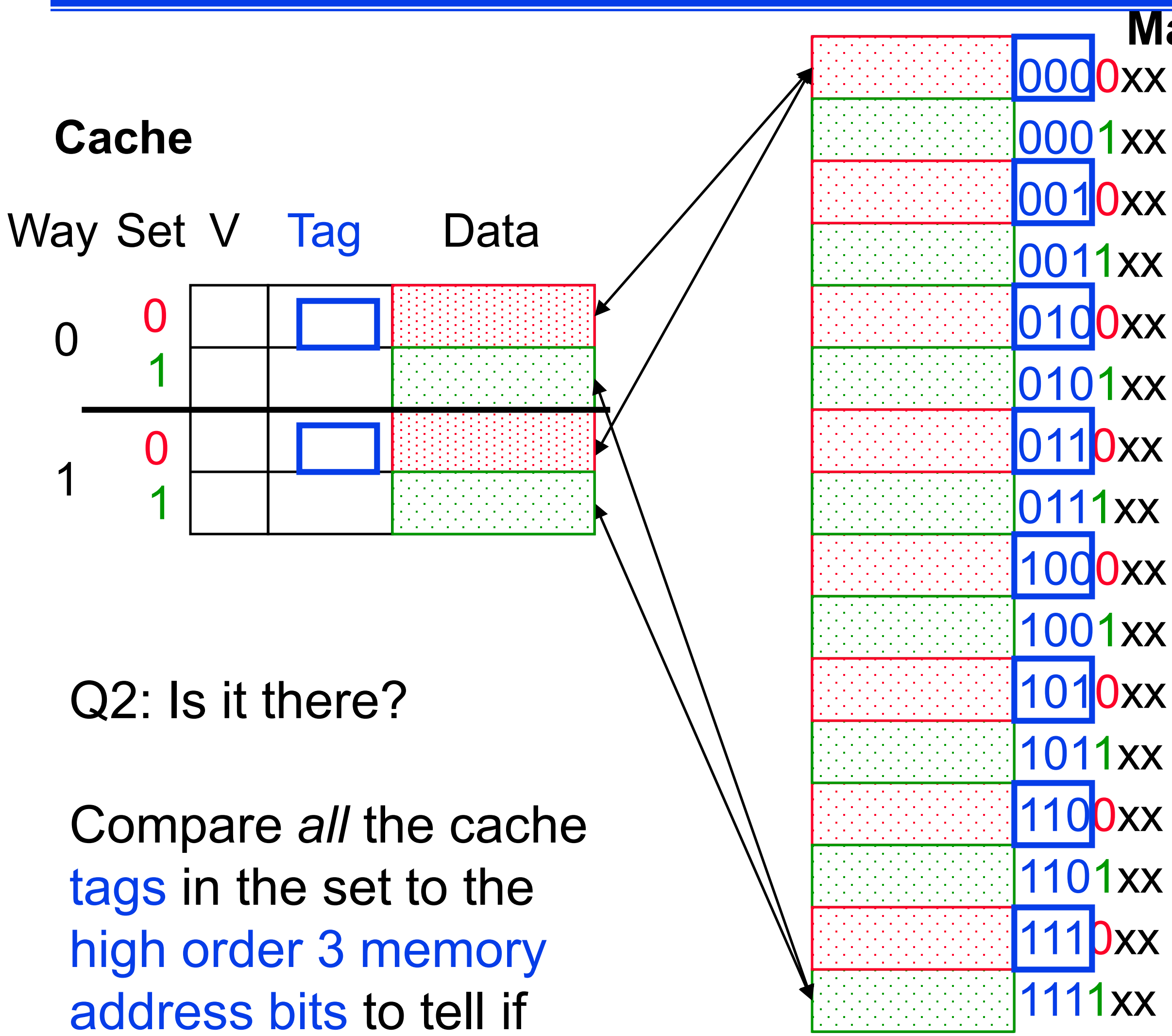
Start with an empty cache   0 4 0 4 0 4 0 4



  - 8 requests, 8 misses

- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

Set Associative Cache Example



4-Byte block size, Two low order bits are used as the byte offset

Q1: How do we find it?

Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

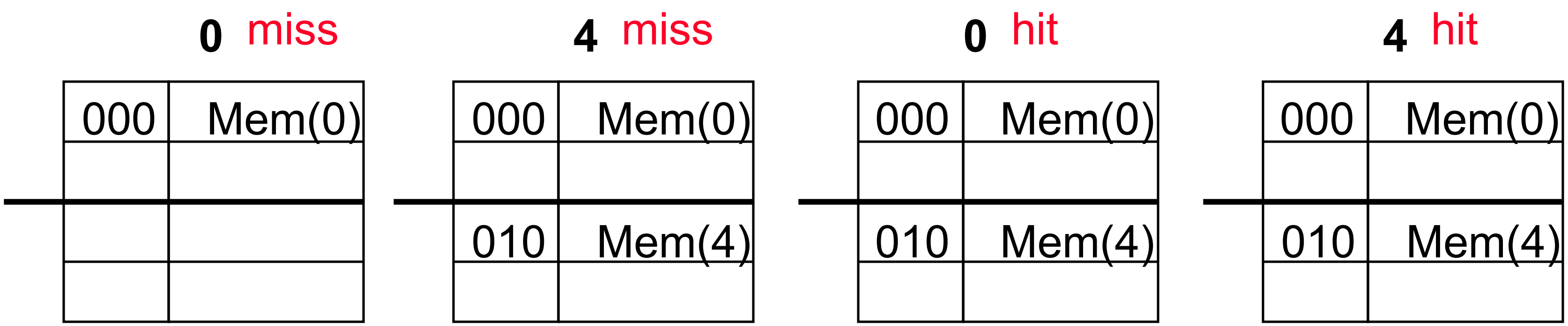
Q2: Is it there?

Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache $\quad$ 0 4 0 4 0 4 0 4

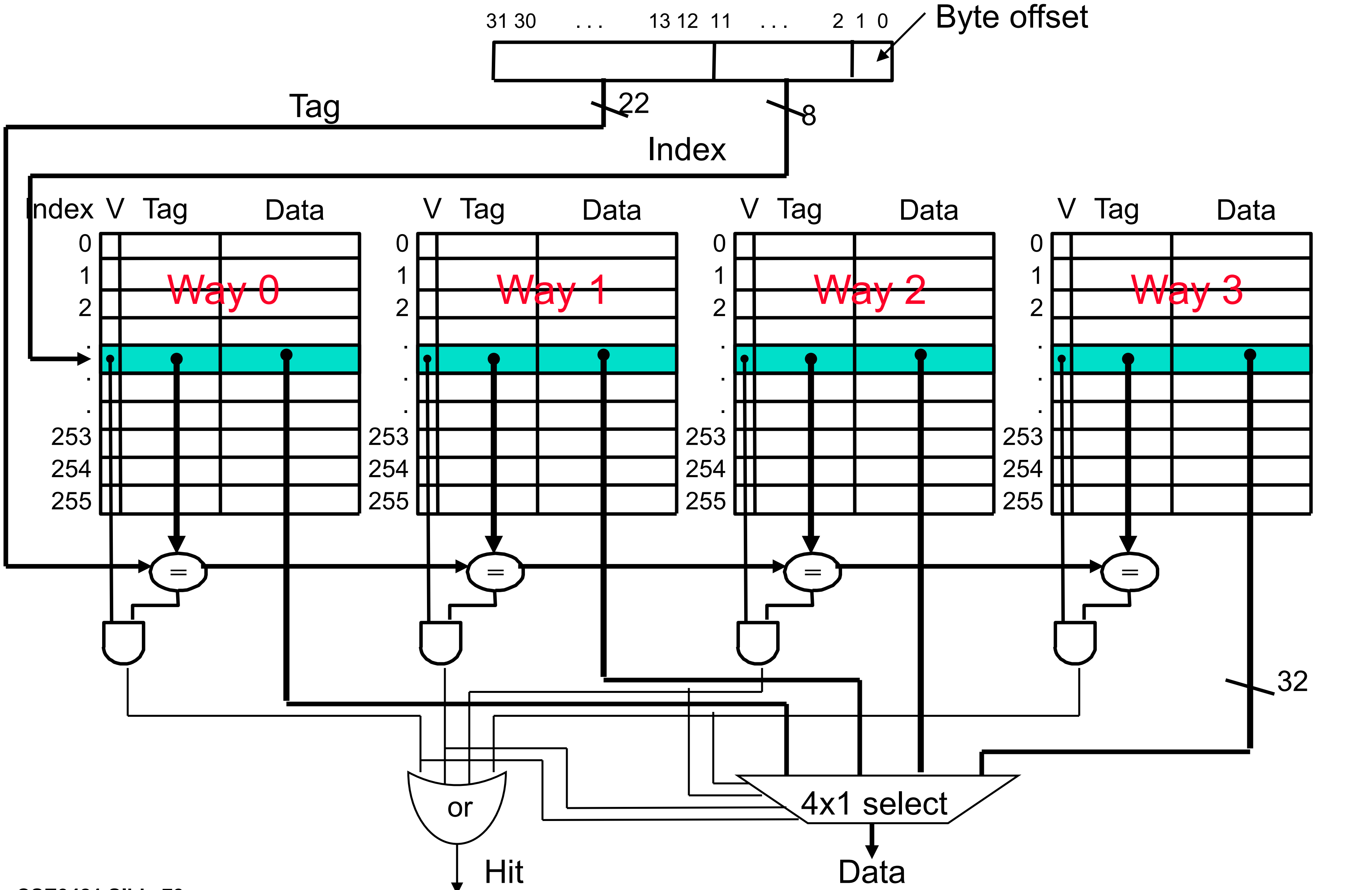


  - 8 requests, 2 misses

- Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

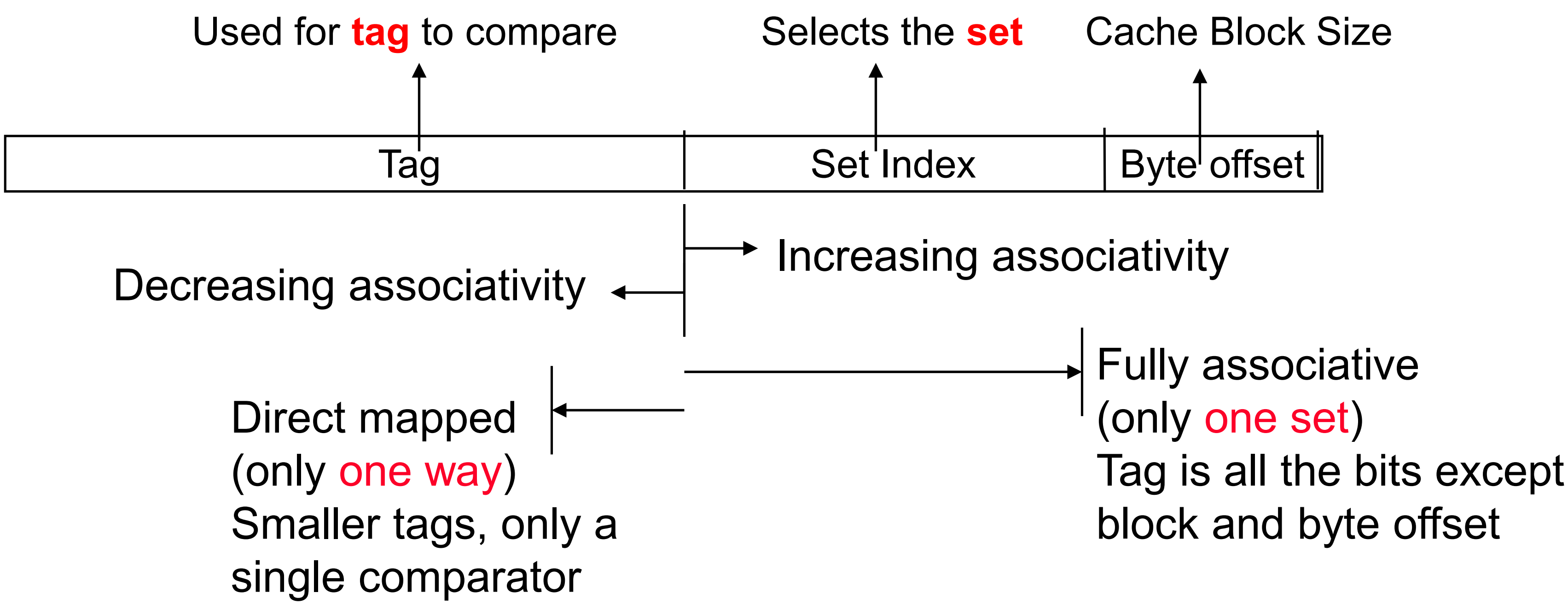
Four-Way Set Associative Cache of a 4KB cache)

- 2^8 = 256 sets each with four ways (each with 1 block of 4 Bytes)



Range of Set Associative Caches

- For a fixed size cache, each time the associativity increases it doubles the number of blocks per set (or the number of ways)
  - halves the number of sets – decreases the set index by 1 bit
  - increases the size of the tag by 1 bit
- Note: word offset is for multiple words in a block. word offset + byte offset = long cache block

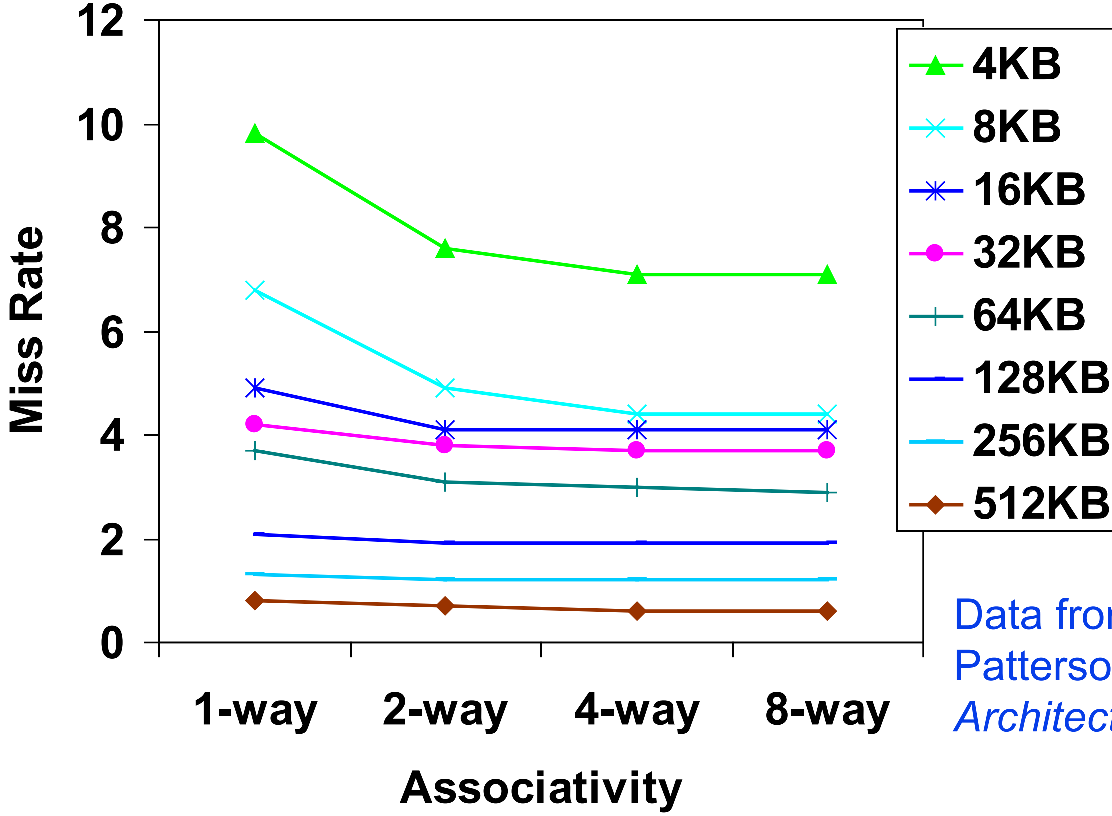


Costs of Set Associative Caches

- When a miss occurs, which way's block do we pick for replacement?
  - Least Recently Used (LRU): the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used relative to the other blocks in the set
    - A reference bit is added to each block to indicate the reference status (0: not-accessed, 1: accessed)
- N-way set associative cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection (and Hit/Miss decision). In a direct mapped cache, the cache block is available before the Hit/Miss decision
    - So, it is not possible to just assume a hit and continue. If it was a miss, nothing to lose

Benefits of Set Associative Caches

- The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation



Data from Hennessy & Patterson, *Computer Architecture*, 2003

- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

Reducing Miss Rates #2 by multi-level caches

- With advancing technology have more than enough room on the die for bigger L1 caches *or* for a second level of caches – normally a unified L2 cache (i.e., it holds both instructions and data) and in some cases even a unified L3 cache
- For our example, CPI_{ideal} of 2, 100 cycle miss penalty (to main memory) and a 25 cycle miss penalty (to UL2$), 36% load/stores, a 2% (4%) L1 I$ (D$) miss rate, add a 0.5% UL2$ miss rate

CPI_{stalls} = CPI_{ideal} + L1 miss rate × miss penalty to access L2 + L2 miss rate × miss penalty access memory

CPI_{stalls} = 2 + .02×25 (L1I$) + .36×.04×25 (L1D$)
+ .005×100 (both D and I miss from L2) = 3.36
(as compared to 5.44 with no L2$)

Two Machines' Cache Parameters

	Intel Nehalem	AMD Barcelona
L1 cache organization & size	Split I$ and D$; 32KB for each per core; 64B blocks	Split I$ and D$; 64KB for each per core; 64B blocks
L1 associativity	4-way (I), 8-way (D) set assoc.; ~LRU replacement	2-way set assoc.; LRU replacement
L1 write policy	write-back, write-allocate	write-back, write-allocate
L2 cache organization & size	Unified; 256MB (0.25MB) per core; 64B blocks	Unified; 512KB (0.5MB) per core; 64B blocks
L2 associativity	8-way set assoc.; ~LRU	16-way set assoc.; ~LRU
L2 write policy	write-back	write-back
L2 write policy	write-back, write-allocate	write-back, write-allocate
L3 cache organization & size	Unified; 8192KB (8MB) shared by cores; 64B blocks	Unified; 2048KB (2MB) shared by cores; 64B blocks
L3 associativity	16-way set assoc.	32-way set assoc.; evict block shared by fewest cores
L3 write policy	write-back, write-allocate	write-back; write-allocate

Basic concepts of hardware cache

- Cache design is memory-address centric
  - The default length is 32 bits, each memory address is unique and points to a byte (byte addressable)
  - Each memory address connects to the first byte of the content, and its default length is a word (4 Bytes or 32 bits)

- Cache is a small storage inclusively built in the CPU chip
  - It contains copies of the data from memory (and disks)
  - A cache-block storage can be multiple-words long
  - Besides data storage, each block needs tags, V bit, and others

- For a given memory address, CPU knows where it is
  - By direct, set-associative or fully associative mapping
  - How do handle write hit, write miss, read hit, and read miss?
  - How to address the conflicts between high hit rates and low latency?