



香港中文大學

計算機科學與工程學系

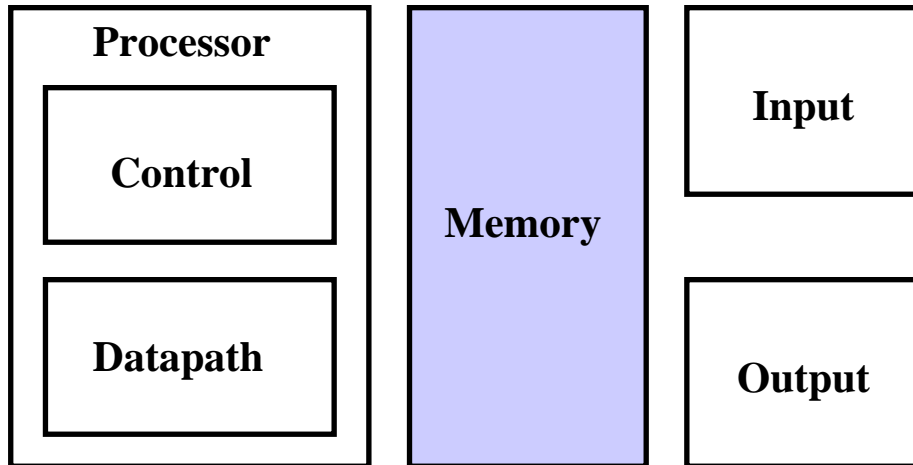
CSC 3420

Computer Systems Architecture

Chapter 7: Memory Hierarchy

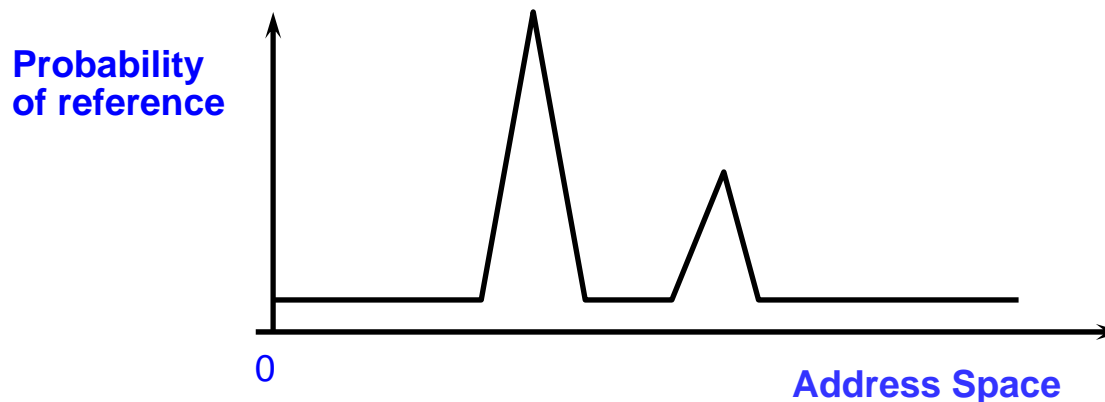
Memory System

- One of the Five Classic Components of a Computer

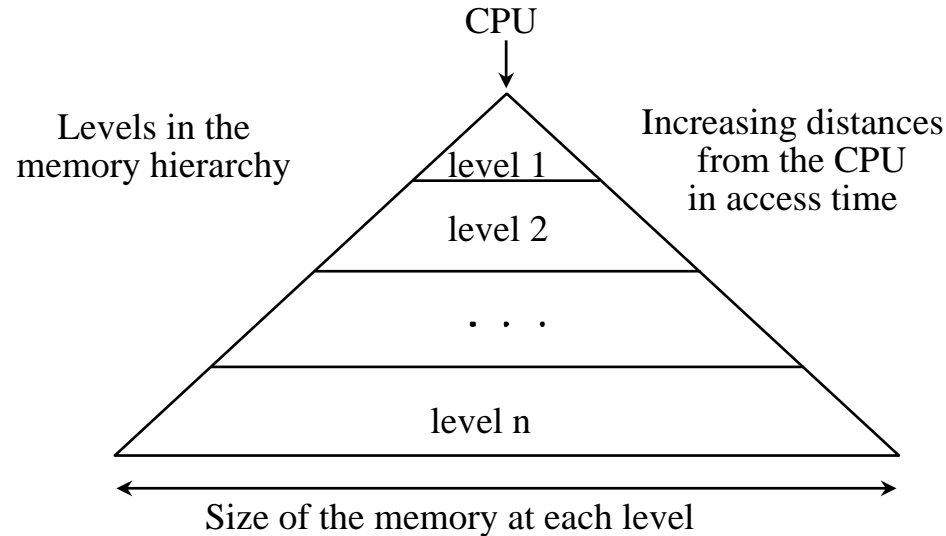


The Principle of Locality

- The **Principle of Locality (地點)**:
 - Program access a relatively small portion (部份) of the address space at any instant of time.
 - e.g. 90% of time in 10% of the code
- Two Different Types of Locality:
 - **Temporal Locality (Locality in Time)**: If an item is referenced, it will tend to be referenced again soon.
 - **Spatial Locality (Locality in Space)**: If an item is referenced, items whose addresses are close by tend to be referenced soon.



Memory Hierarchy : Why ?



- **Users want large and fast memories!**

Disk: 7200 rpm, 1.6 Gb/sec & access time 7-9 ms, HK\$0.72/Gbyte (640G - \$460)

DRAM: access times are 25-50 ns at cost of HK\$0.069/Mbyte (4G DDR2-RAM- \$275)

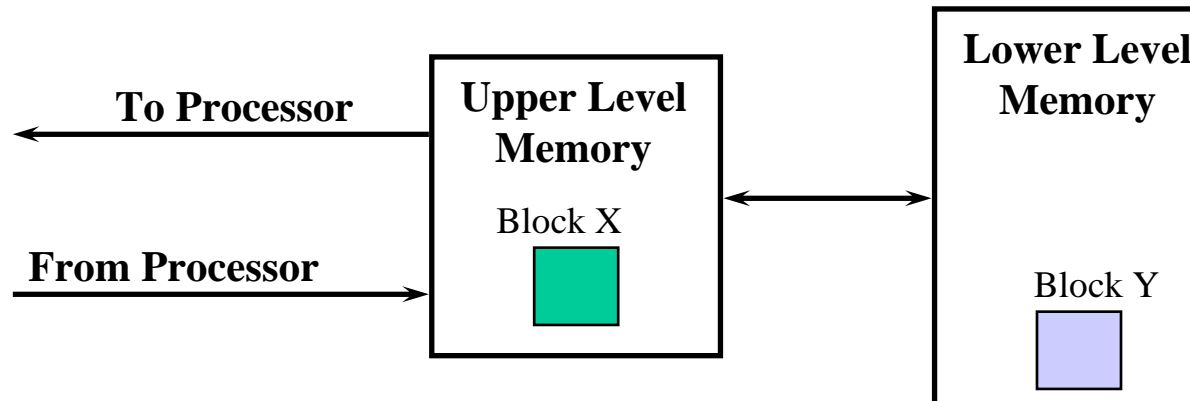
SRAM: access times are 1 – 2 ns at cost of ?? (cache embedded inside the CPU)

- **Try and give it to them anyway**

Build a memory hierarchy

Memory Hierarchy: Principles of Operation

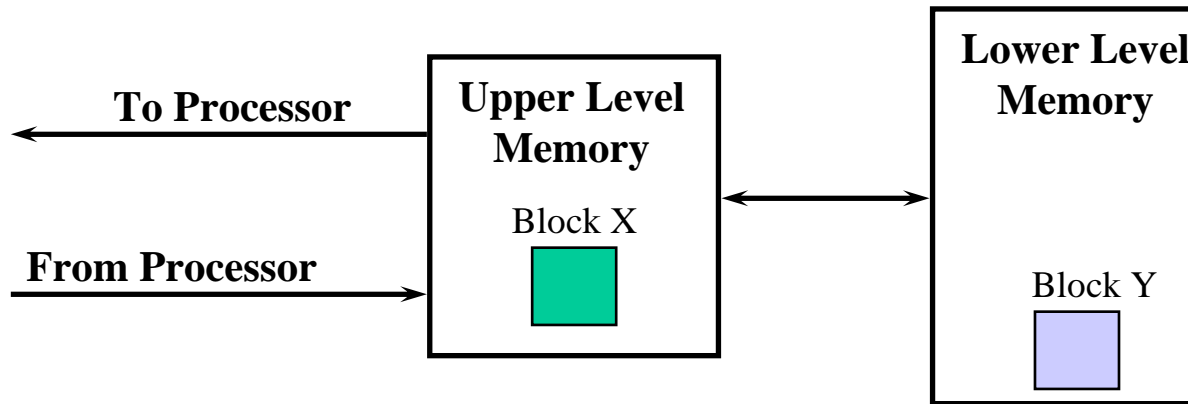
- At any given time, data is copied between only 2 adjacent levels:
 - Upper Level: the one closer to the processor
 - Smaller, faster, and uses more expensive technology
 - Lower Level: the one further away from the processor
 - Bigger, slower, and uses less expensive technology



- **Block:**

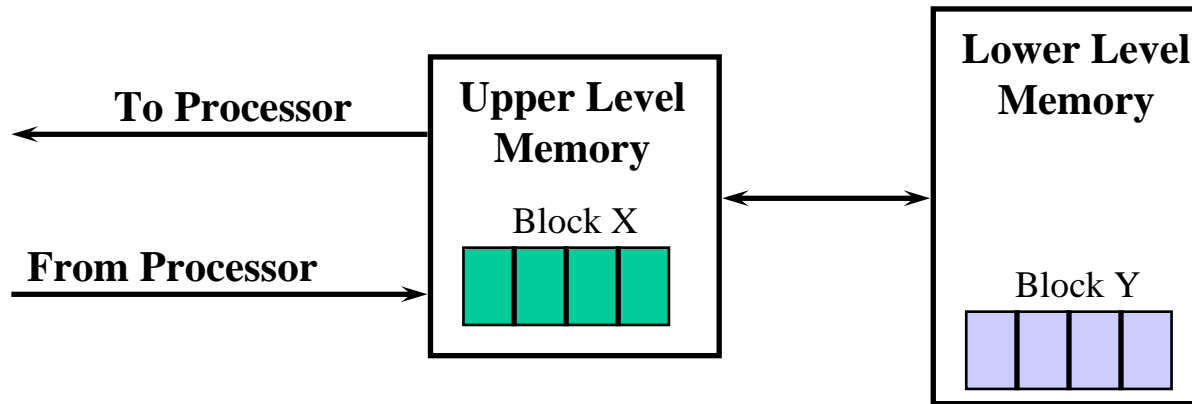
- The minimum unit of information that can either be present or not present in the two level hierarchy

Memory Hierarchy: Terminology



- **Hit**: data appears in a block in the upper level (example: Block X)
 - **Hit Rate**: the fraction of memory access found in the upper level
 - **Hit Time**: Time to access the upper level which consists of
RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieved from a block in the lower level (Block Y)
 - **Miss Rate** = $1 - (\text{Hit Rate})$
 - **Miss Penalty** : Time to replace a block in the upper level +
Time to deliver the block to the processor
- **Hit Time** \ll **Miss Penalty**

Memory Hierarchy: How Does it Work?



◦ Temporal Locality (Locality in Time):

- If an item is referenced, it will tend to be referenced again soon.
- Keep more recently (近來) accessed data items closer to the processor.

◦ Spatial Locality (Locality in Space):

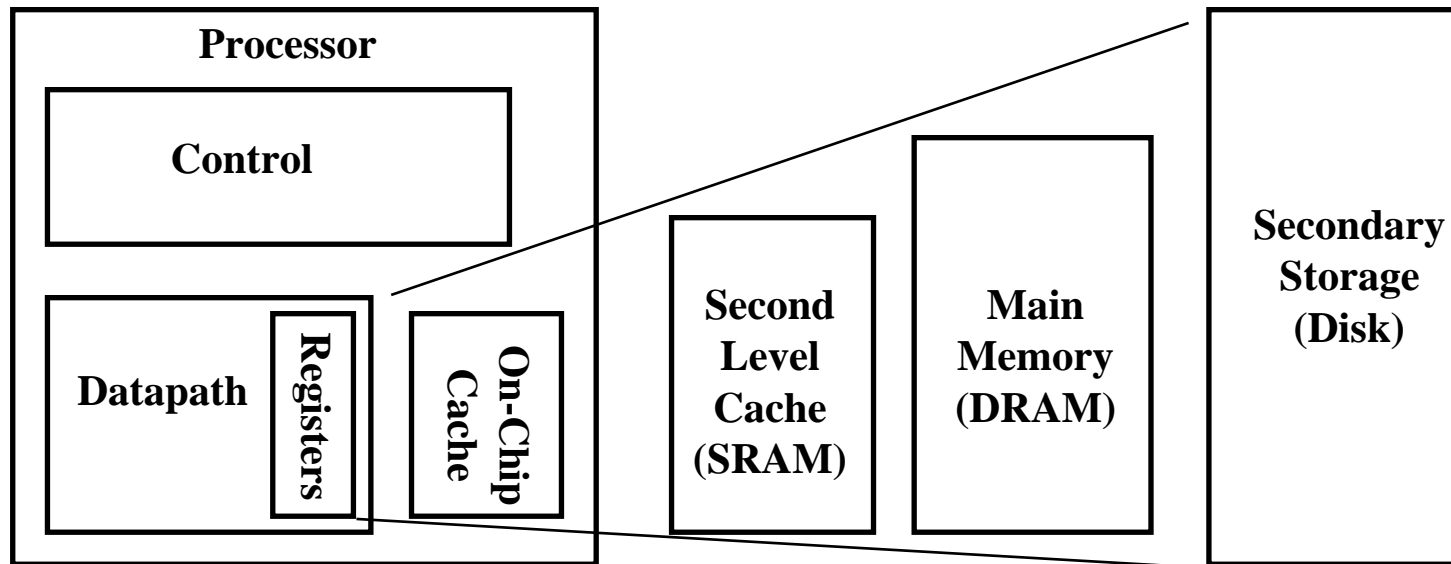
- If an item is referenced, items whose addresses are close by tend to be referenced soon.
- Move blocks consisting of contiguous (緊接) words to the upper levels

◦ Average Access time :

$$= \text{hit rate} * \text{hit time} + \text{miss rate} * \text{miss penalty}$$

Memory Hierarchy of a Modern Computer System

- By taking advantage of the principle of locality:
 - Present the user with as much memory as is available in the cheapest technology.
 - Provide access at the speed offered by the fastest technology.



Speed (ns):	1X (fast)	10X	100X	10,000,000X (10s ms) (slow)
Size (bytes):	100s (small)	Ks	Gs	Ts (large)
Cost :	highest			lowest

Levels of the Memory Hierarchy

Capacity
Access Time

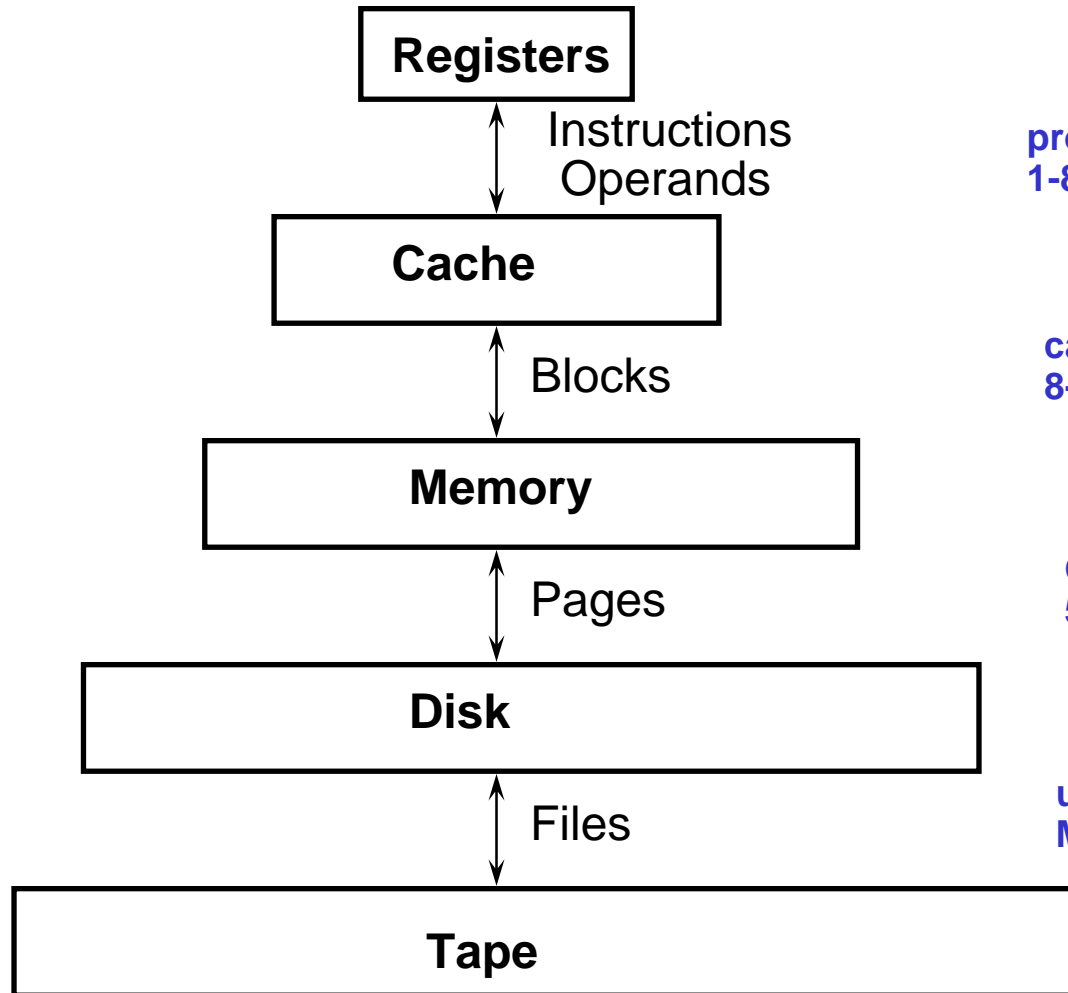
CPU Registers
100s Bytes
<1 ns

Cache
K-M Bytes
1-10 ns

Main Memory
G Bytes
10-100ns

Disk
G-T Bytes
ms

Tape
infinite
sec-min



Staging: Cross Reference Unit

program/compiler
1-8 bytes

cache control
8-128 bytes

OS
512-4K bytes

user/operator
Mbytes

Upper Level

faster

Larger

Lower Level

Memory Hierarchy : Technology

◦ **Random Access:**

- “Random” is good : access time is the same for all locations
- **DRAM** : Dynamic Random Access Memory
 - High density, low power, cheap, slow
 - Dynamic: need to be “refreshed” (使恢復活力) regularly
- **SRAM** : Static Random Access Memory
 - Low density, high power, expensive, fast
 - Static: content will last “forever”

◦ **“Not-so-random” Access Technology:**

- Access time varies from location to location and from time to time
- Examples: Disk, tape drive, CDROM

◦ A very brief introduction of random access technology will be presented

- The Main Memory: DRAMs
- Caches (隱藏所; 寶庫): SRAMs

Random Access Memory (RAM) Technology

- **Why do we need to know about RAM technology?**
 - **Processor performance is usually limited by memory bandwidth**
 - **As IC densities increase, lots of memory will fit on processor chip**
 - **Tailor on-chip memory to specific needs**
 - **Instruction cache**
 - **Data cache**
 - **Write buffer**
- **What makes RAM different from a bunch of flip-flops?**
 - **Density: RAM is much more denser**

Technology Trends

Capacity Speed

Logic: 2x in 3 years 2x in 3 years

DRAM: 4x in 3 years 1.4x in 10 years

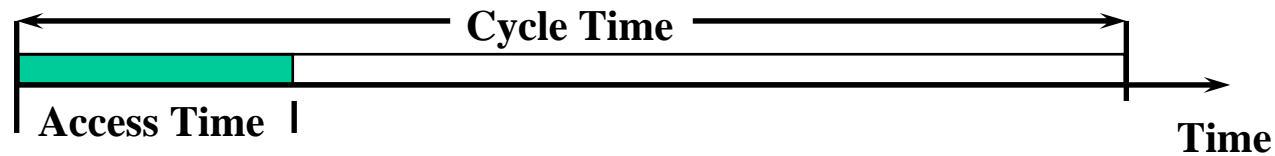
Disk: 2x in 3 years 1.4x in 10 years

DRAM

<u>Year</u>	<u>Size</u>	<u>Cycle Time</u>
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns
1998	128 Mb	100 ns

More Information: http://en.wikipedia.org/wiki/Dynamic_random_access_memory

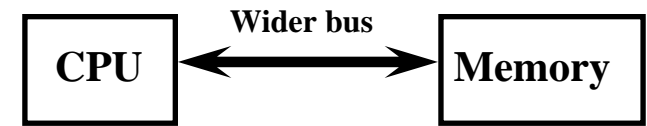
Cycle Time versus Access Time



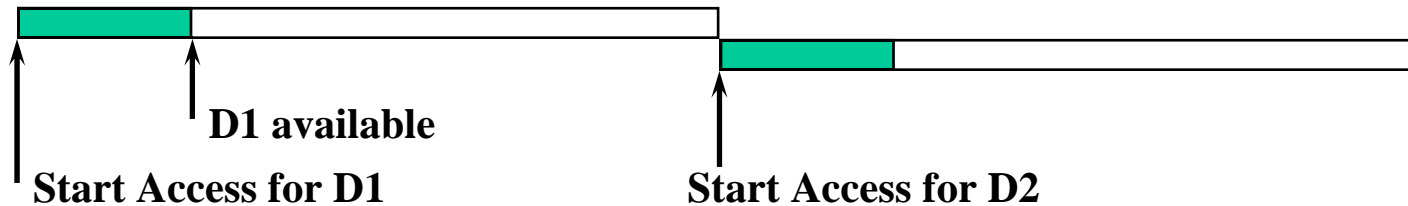
- DRAM (Read/Write) Cycle Time \gg DRAM (Read/Write) Access Time
- **DRAM (Read/Write) Cycle Time :**
 - How frequent can you initiate an access?
 - Analogy: How long does it take for a dentist to treat a patient?
- **DRAM (Read/Write) Access Time:**
 - How quickly will you get what you want once you initiate an access?
 - Analogy: If the dentist is free, you can see the dentist with little delay
- **DRAM Bandwidth Limitation analogy:**
 - What happens if it takes a longer time to clean the equipment after each treatment? It takes a longer time before the dentist can see the next patient!

Increasing Bandwidth - Interleaving

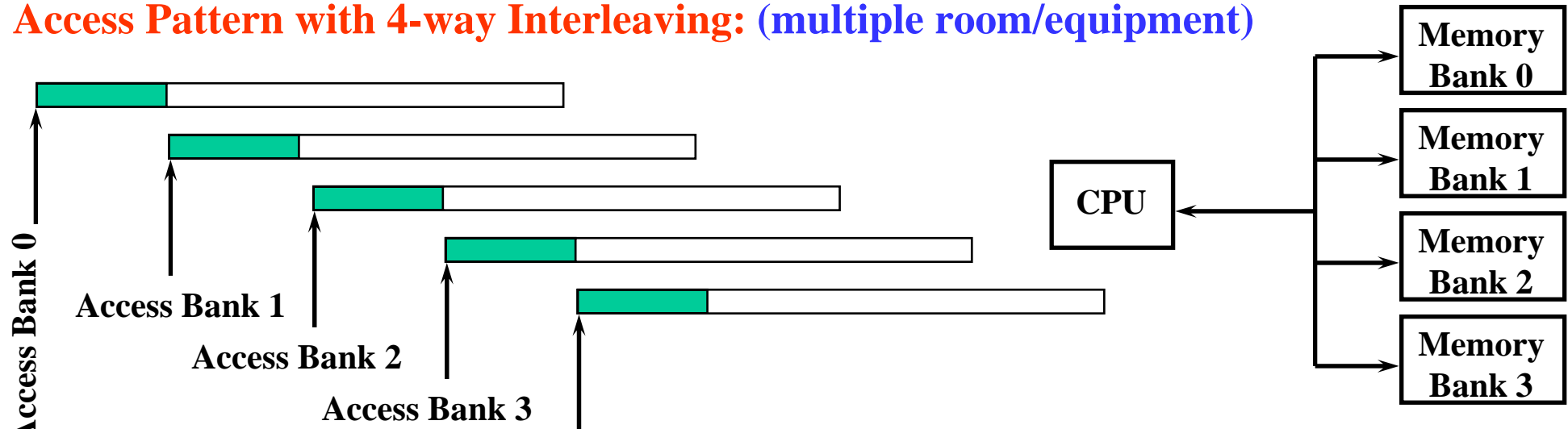
Wider data bus and interleaving technology can be used to increase bandwidth (**more dentists**)



Access Pattern without Interleaving:

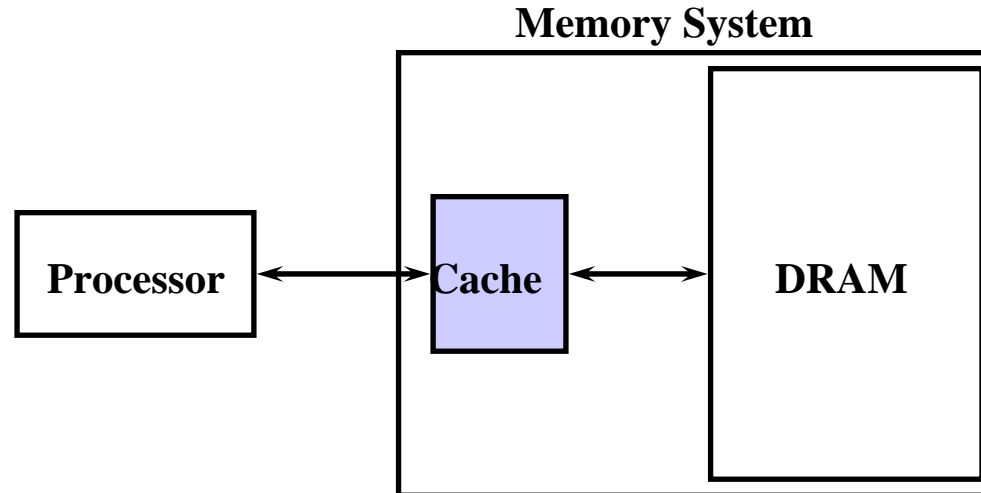


Access Pattern with 4-way Interleaving: (multiple room/equipment)



We can Access Bank 0 again

Memory Hierarchy : Caches

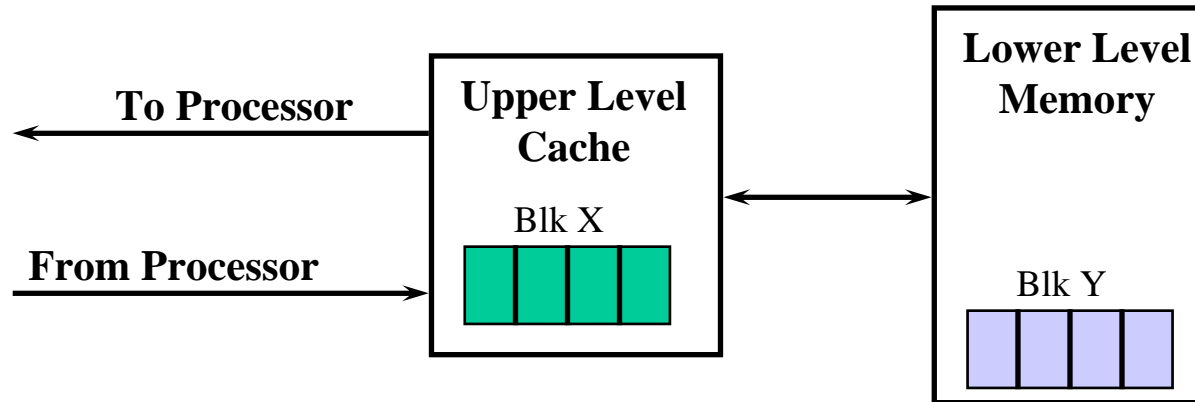


- **Motivation:**
 - **DRAM** is slow but cheap and dense; good choice for providing the user with a **BIG** memory system.
 - **SRAM** is fast but expensive and not very dense; good choice for applications require **FAST** access time.
- Make the *average access time* small by:
 - Servicing most accesses from a small, fast memory.
- Reduce the *bandwidth* required of the large memory.
- Rely on the **Locality principle**.

Typical Values

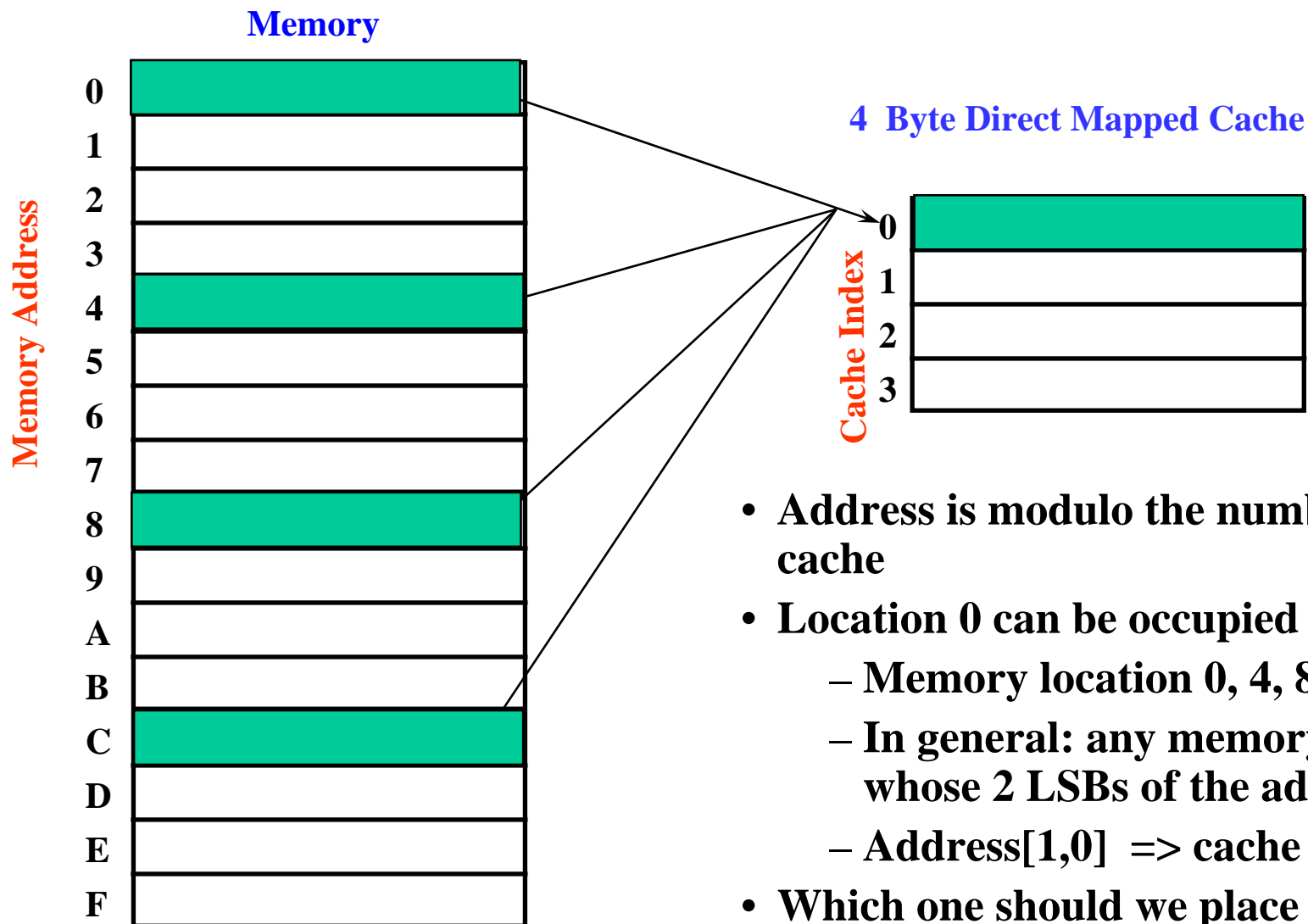
	Typical Values
Block (line) size	4 - 128 bytes
Hit time	1 - 4 cycles
Miss penalty	8 - 32 cycles (and increasing)
(access time)	(6-10 cycles)
(transfer time)	(2 - 22 cycles)
Miss rate	1% - 20%
Cache Size	1 KB - 512 KB

How Does Cache Work?



- **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
 - Examples: program loops, variables
 - Keep more recently accessed data items closer to the processor
- **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
 - Examples: program codes, array
 - Move blocks consists of contiguous words to the cache
- Many ways to map upper and lower level memory

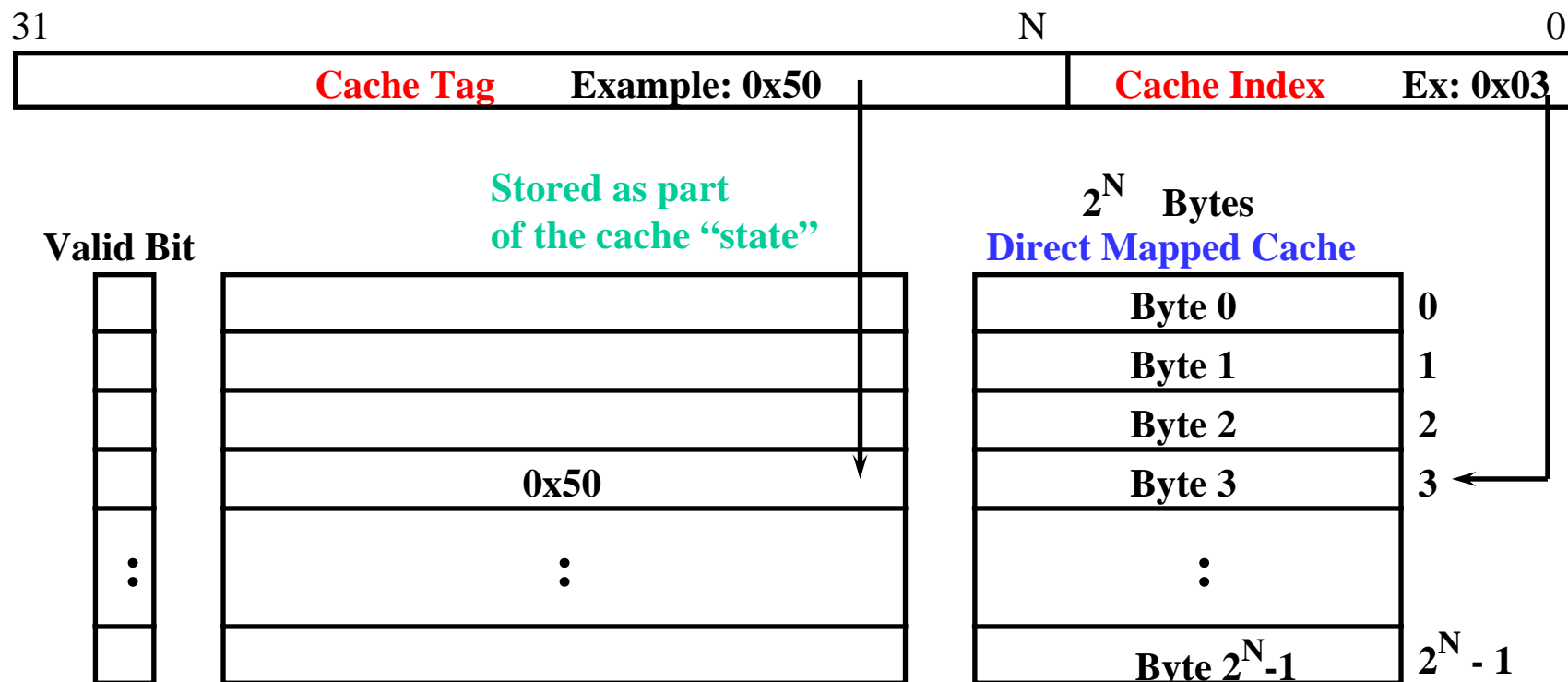
Direct Mapped Cache



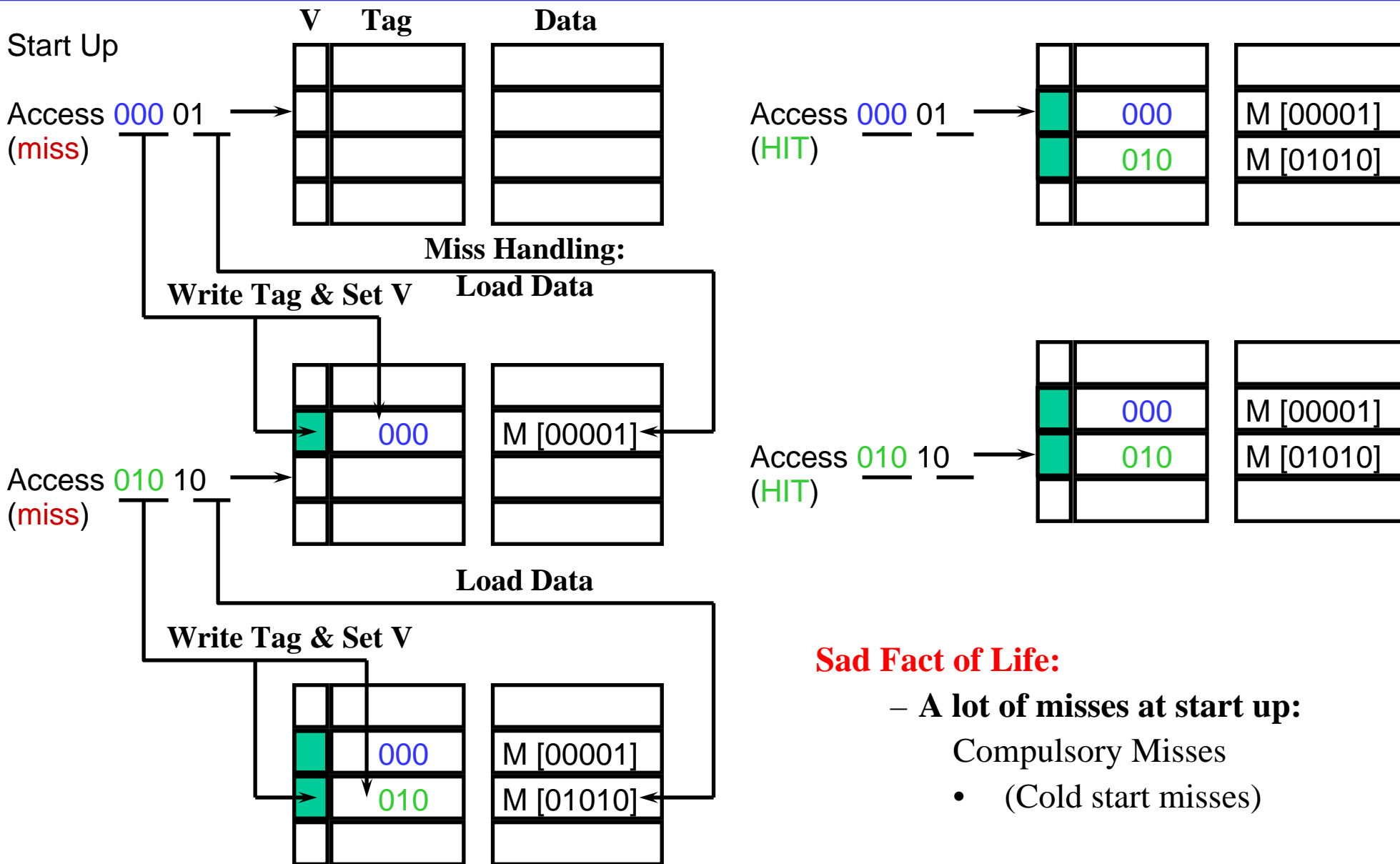
- Address is modulo the number of blocks in the cache
- Location 0 can be occupied by data from:
 - Memory location 0, 4, 8, ... etc.
 - In general: any memory location whose 2 LSBs of the address are 0s
 - $\text{Address}[1,0] \Rightarrow \text{cache index}$
- Which one should we place in the cache?
- How can we tell which one is in the cache?

Cache Tag and Cache Index

- Assume a 32-bit memory (byte) address:
 - A 2^N bytes direct mapped cache:
 - Cache Index: The lower N bits of the memory address ($A[n-1, n-2, \dots, 1, 0]$)
 - Cache Tag: The upper $(32 - N)$ bits of the memory address ($A[31, 30, \dots, n]$)



Cache Access Example

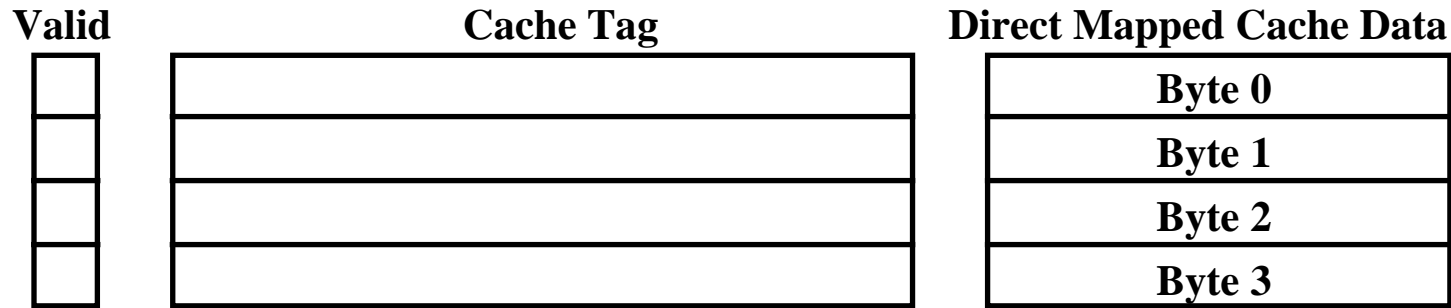


Sad Fact of Life:

- A lot of misses at start up:
 - Compulsory Misses
 - (Cold start misses)

Definition of a Cache Block

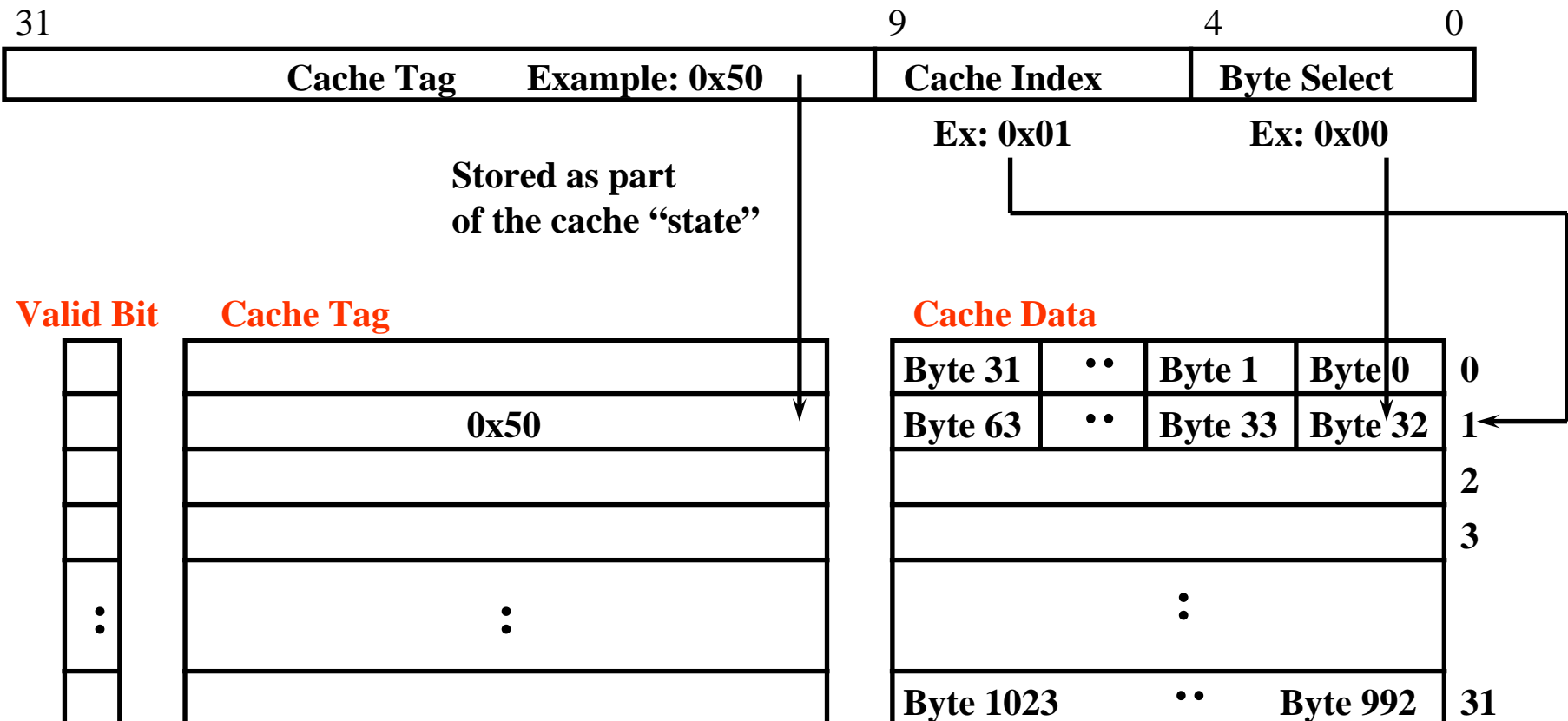
- **Cache Block**: the cache data that has in its own cache tag
- Our previous “extreme” example:
 - 4-byte Direct Mapped cache: Block Size = 1 Byte
 - Take advantage of Temporal Locality: If a byte is referenced, it will tend to be referenced soon.
 - Did not take advantage of Spatial Locality: If a byte is referenced, its adjacent bytes will be referenced soon.



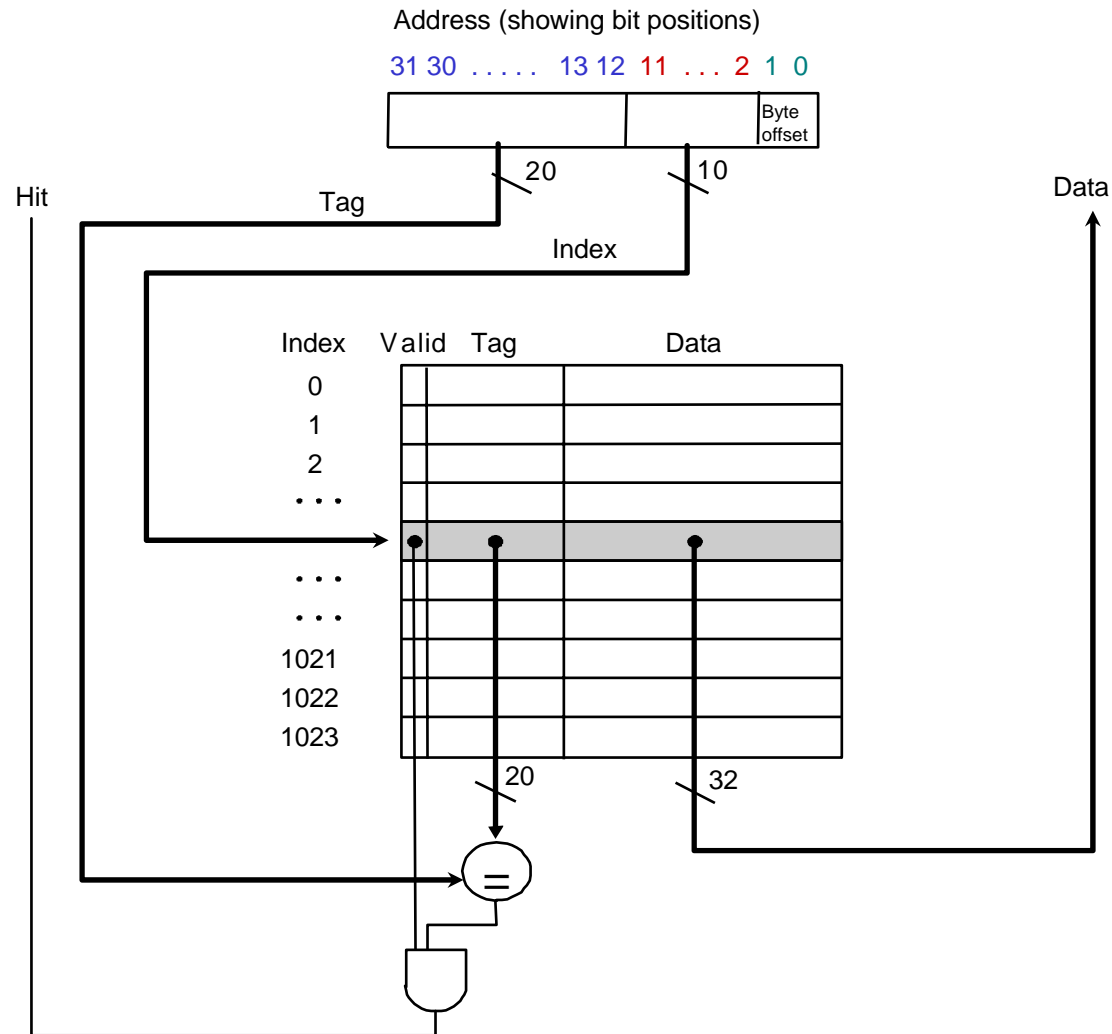
- In order to take advantage of Spatial Locality: **increase the block size**

Bigger Cache Blocks: (e.g. 1 KB Direct Mapped Cache with 32 B Blocks)

- For a 2^N byte cache with 2^M byte block
 - 2^{N-M} blocks of 2^M bytes each
 - The uppermost (32 - N) bits, $A[31 : N]$, are always the Cache Tag
 - The lowest M bits, $A[M-1 : 0]$, are the Byte Select
 - The rest, $A[N-1 : M]$, are the Cache index



Cache in MIPS



What kind of locality are we taking advantage of?

Performance Estimation

- **Simplified model:**

execution time = (execution cycles + stall cycles) × cycle time

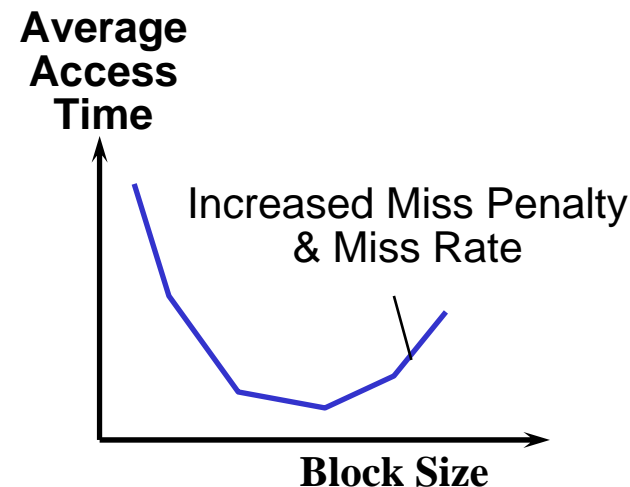
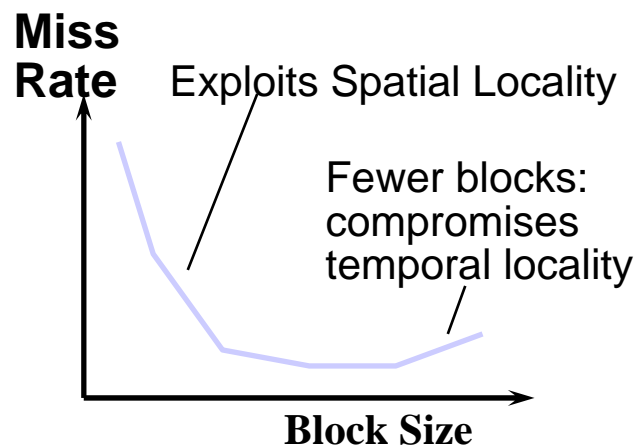
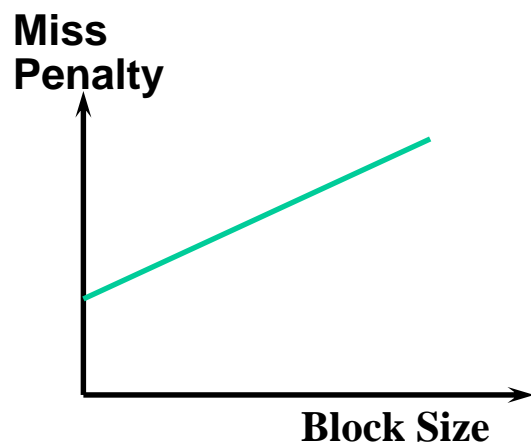
stall cycles = # of instructions × miss ratio × miss penalty

- **Two ways of improving performance:**
 - decreasing the miss ratio
 - decreasing the miss penalty

What happens if we increase block size?

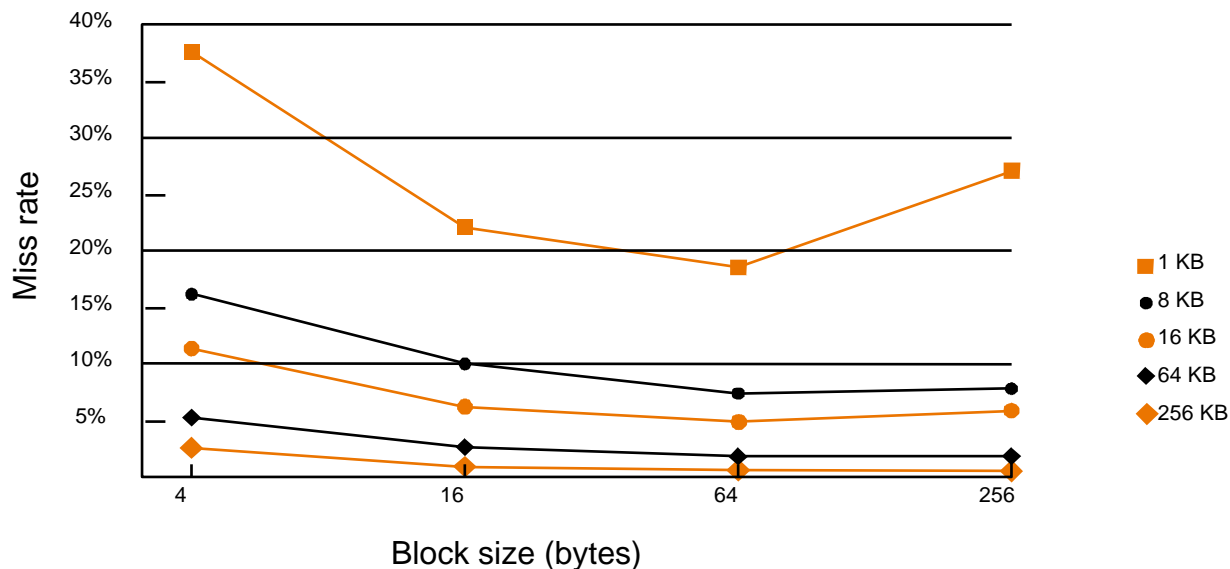
Block Size Tradeoffs

- In general, larger block size takes advantage of spatial locality BUT:
 - Larger block size means larger miss penalty:
 - Takes longer time to fill up the block
 - If block size is too big relative to cache size, miss rate will go up
- Average Access Time:
= Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate



Performance Statistics

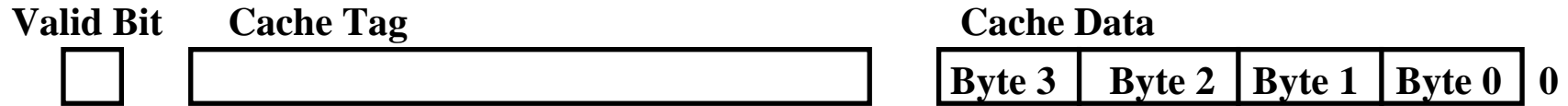
- Increasing the block size tends to decrease miss rate:



- Use split caches because there is more spatial locality in code:

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Another Extreme Example



- **Cache Size = 4 bytes** **Block Size = 4 bytes**
 - **Only ONE entry in the cache**
- **True: If an item is accessed, likely that it will be accessed again soon**
 - **But it is unlikely that it will be accessed again immediately!!!**
 - **The next access will likely to be a miss again**
 - Continually loading data into the cache but discard (force out) them before they are used again
 - Worst nightmare of a cache designer: Ping Pong Effect; **thrashing**
- **Conflict Misses are misses caused by:**
 - **Different memory locations mapped to the same cache index**
 - Solution 1: make the cache size bigger (have just discussed)
 - Solution 2: Multiple entries for the same Cache Index

Decreasing miss ratio with associativity

One-way set Associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set Associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set Associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set Associative **(fully associative)**

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

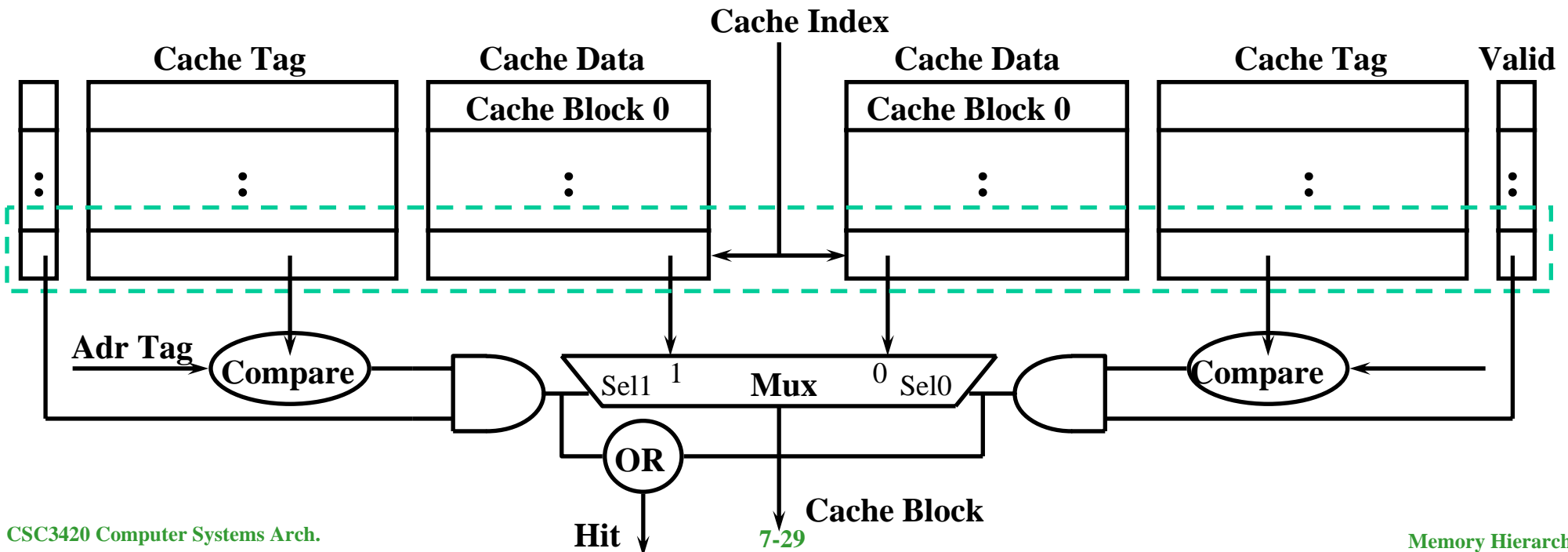
Compared to direct mapped, give a series of references that:

- results in a lower miss ratio using a 2-way set associative cache*

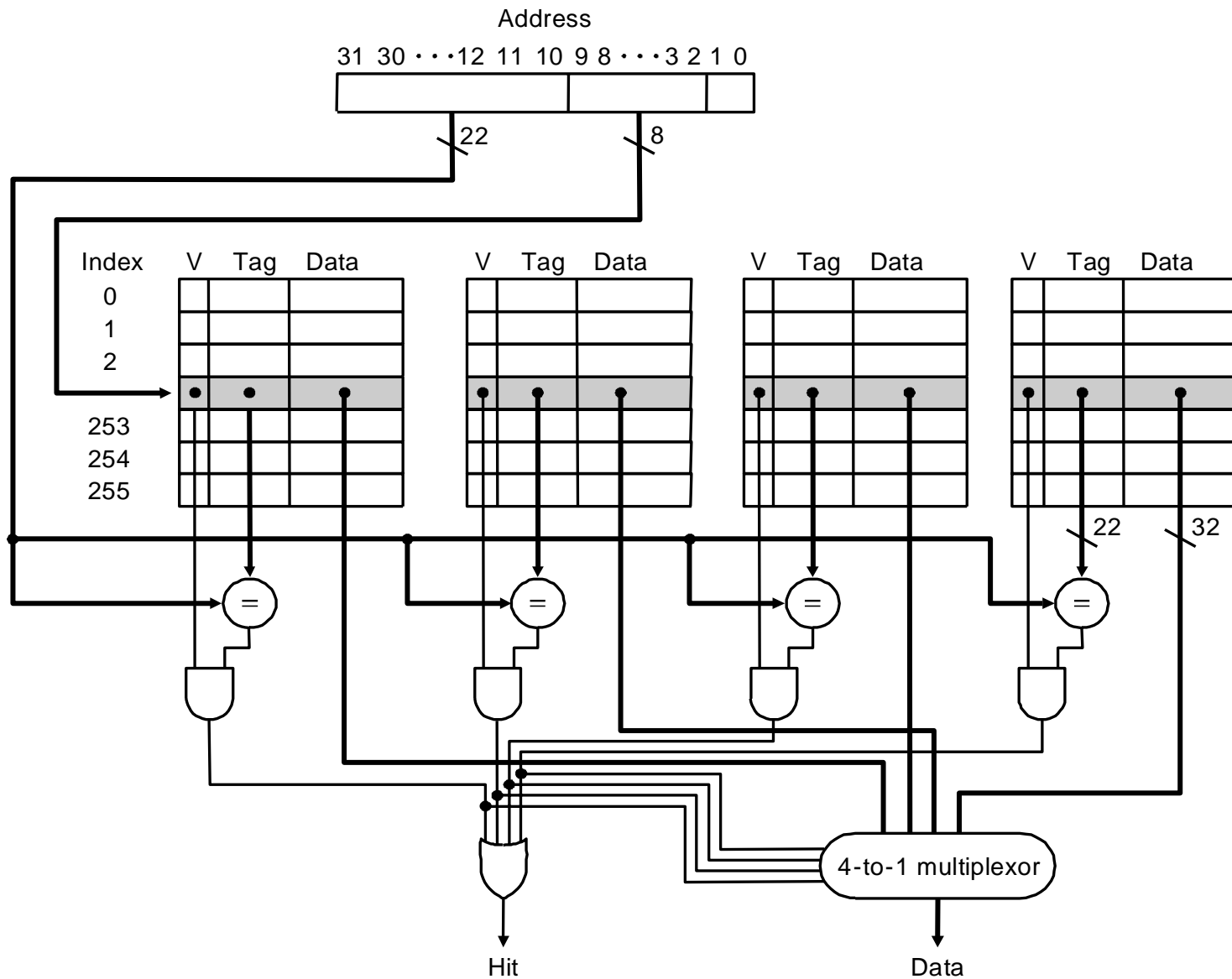
Assuming we use the “least recently used” replacement strategy

A Two-way Set Associative

- **N-way set associative: N entries for each Cache Index**
 - N direct mapped caches operates in parallel
- **Example: Two-way set associative cache**
 - Cache Index selects a “set” from the cache
 - The two tags in the set are compared in parallel
 - Data is selected based on the tag result

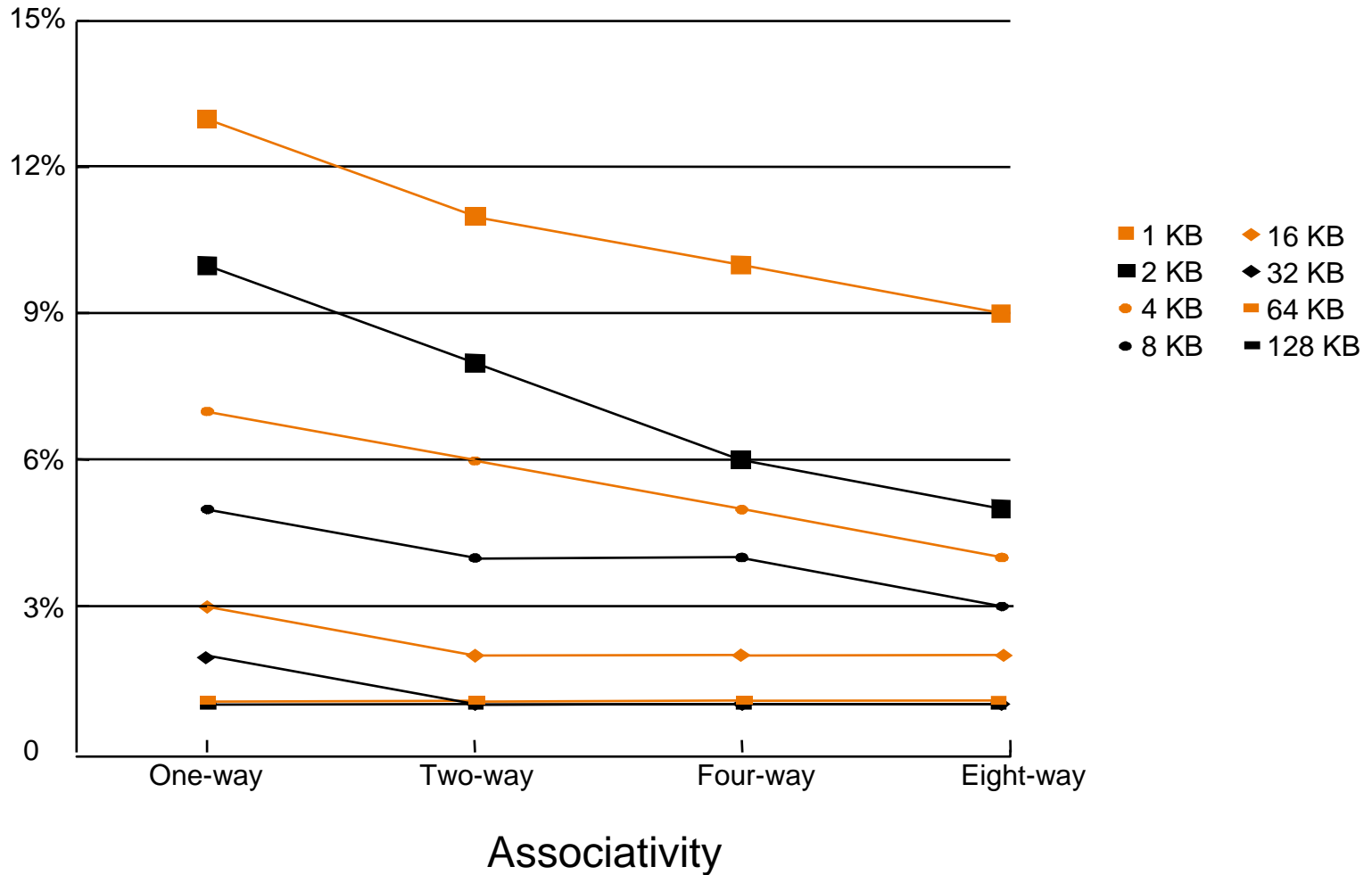


A 4-way Set Associative Implementation



Performance Statistics

Miss ratio



Fully Associative:



- **Fully Associative Cache** -- push the set associative idea to its limit!
 - Forget about the Cache Index
 - Compare the Cache Tags of all cache entries in parallel
 - Example: Block Size = 32 bytes blocks, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache

Sources of Cache Misses

- **Compulsory (cold start, first reference):** first access to a block
 - “Cold” fact of life: not a whole lot you can do about it
- **Conflict (collision):**
 - Multiple memory locations mapped to the same cache location
 - Solution 1: increase cache size
 - Solution 2: increase associativity
- **Capacity:**
 - Cache cannot contain all blocks accessed by the program
 - Solution: increase cache size
- **Invalidation:** other process (e.g., I/O) updates memory

Hits vs. Misses

- **Read hits**
 - this is what we want!
- **Read misses**
 - stall the CPU, fetch block from memory, deliver to cache, restart
- **Write hits**
 - can replace data in cache and memory (write-through)
 - write the data only into the cache (write-back the cache later)
- **Write misses**
 - read the entire block into the cache, then write the word

Comparisons Between Different Mapping Methods

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss See Note	High (but who cares!)	Medium	Low
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Invalidation Miss	Same	Same	Same

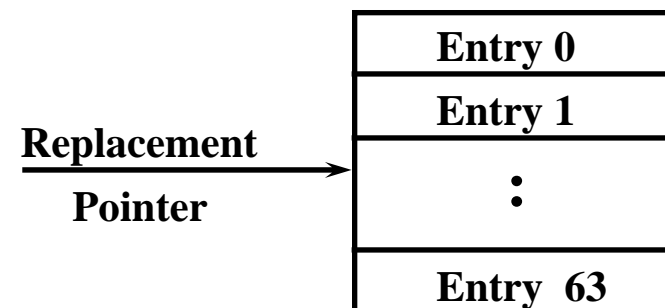
Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant.

Block Replacement Decision

- **Direct Mapped Cache:**
 - Each memory location can only mapped to 1 cache location
 - No need to make any decision :
 - Current item replaced the previous item in that cache location
- **N-way Set Associative Cache:**
 - Each memory location have a choice of N cache locations
- **Fully Associative Cache:**
 - Each memory location can be placed in ANY cache location
- **Cache miss in a N-way Set Associative or Fully Associative Cache:**
 - Bring in new block from memory
 - Throw out a cache block to make room for the new block
 - How do make a decision on which block to throw out?

Cache Block Replacement Policy

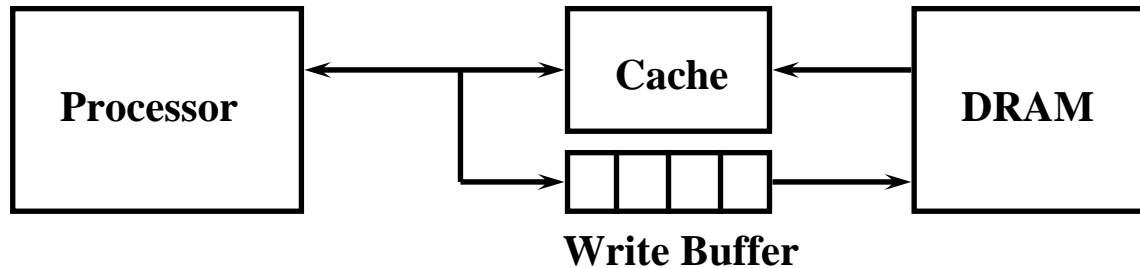
- **Random Replacement:**
 - Hardware randomly selects a cache item and throw it out
- **Least Recently Used:**
 - Hardware keeps track of the access history
 - Replace the entry that has not been used for the longest time
- **Example of a Simple “Pseudo” Least Recently Used Implementation:**
 - Assume 64 Fully Associative Entries
 - Hardware replacement pointer points to one cache entry
 - Whenever an access is made to the entry the pointer points to:
 - Move the pointer to the next entry
 - **Otherwise: do not move the pointer**
- **Another “Pseudo” Least Recently Used Implementation**
 - Set a “referenced bit” if entry is accessed
 - reset all “referenced bits” after a miss
 - look for entry with reset “referenced bit” to be replaced.



Cache Write Policy: Write Through versus Write Back

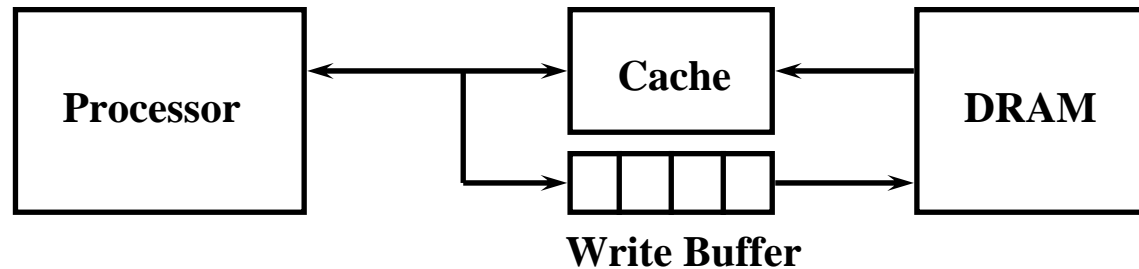
- Cache read is much easier to handle than cache write:
 - Instruction cache is much easier to design than data cache
- Cache write:
 - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
 - **Write Back** : write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
 - Need a “dirty” bit for each cache block
 - Greatly reduce the memory bandwidth requirement
 - Control can be complex
 - **Write Through** : write to cache and memory at the same time.
 - What!!! How can this be? Isn't memory too slow for this?

Write Buffer for Write Through

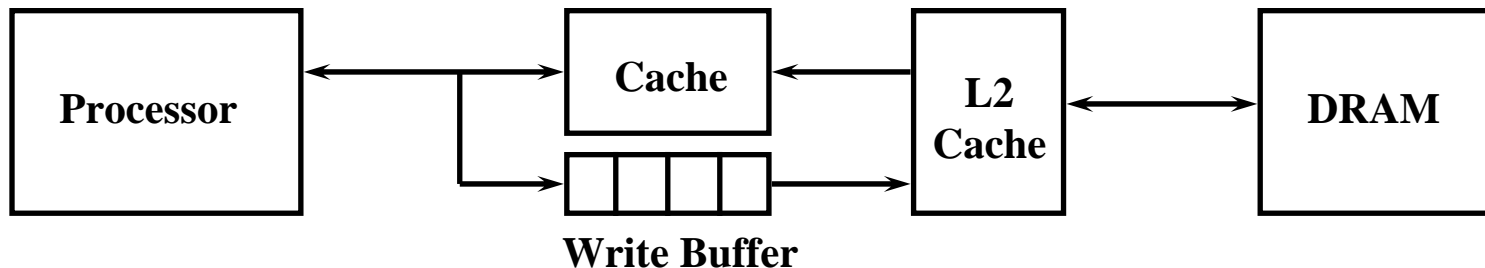


- A **Write Buffer** is needed between the Cache and Memory
 - Processor: writes data into the cache and the write buffer
 - Memory controller: write contents of the buffer to memory
- Write buffer is just a **FIFO**:
 - Typical number of entries: 4
 - Works fine if: Store frequency (w.r.t. time) $\ll 1 / \text{DRAM write cycle}$
- Memory system designer's nightmare:
 - Store frequency (w.r.t. time) $\geq 1 / \text{DRAM write cycle}$
 - Write buffer saturation

Write Buffer Saturation



- **Store frequency (w.r.t. time) $\geq 1 / \text{DRAM write cycle}$**
 - **If this condition exists for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):**
 - Store buffer will overflow no matter how big you make it
 - The CPU Cycle Time \ll DRAM Write Cycle Time
- **Solution for write buffer saturation:**
 - Use a write back cache
 - Install a second level (L2) cache:

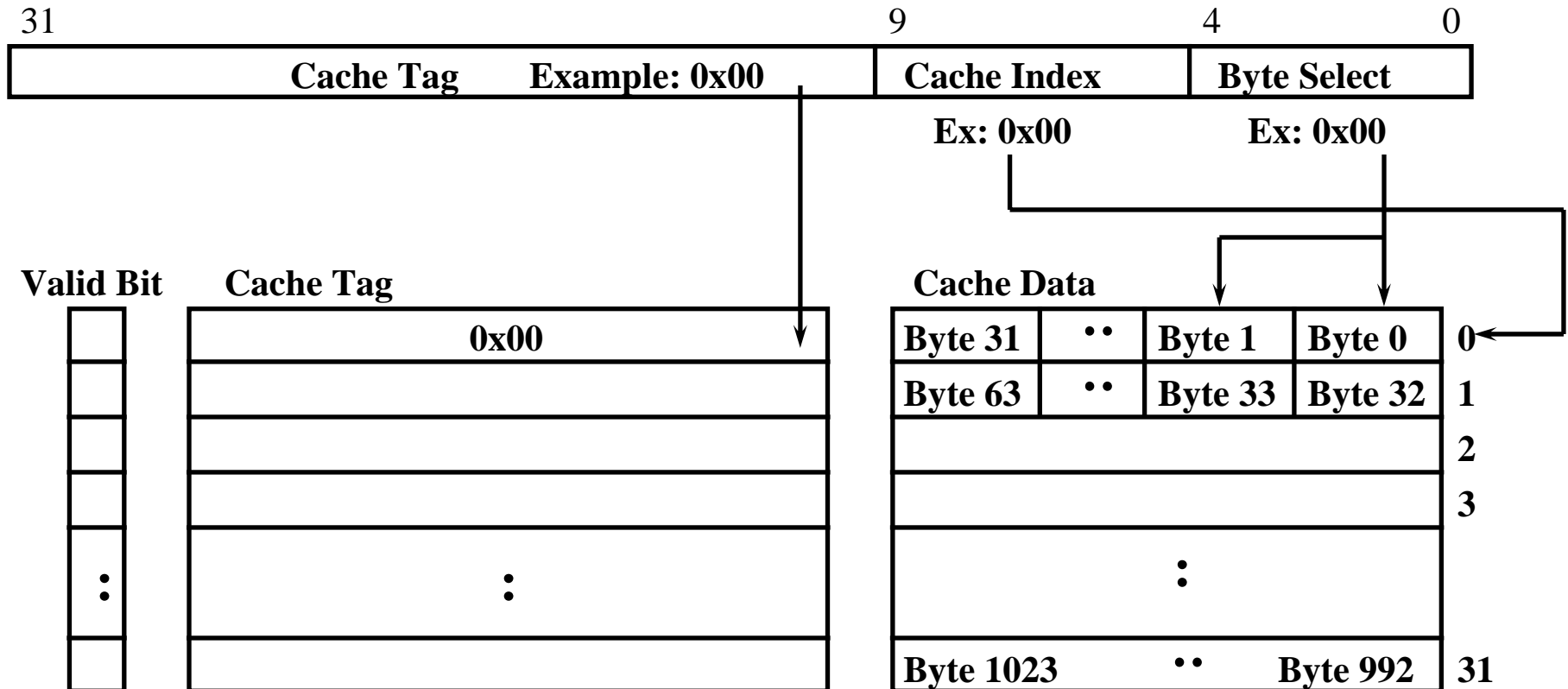


Write Allocate versus Not Allocate

- Assume: a 16-bit write to memory location 0x0 and causes a miss
 - Do we read in the rest of the block (Byte 2, 3, ... 31)?

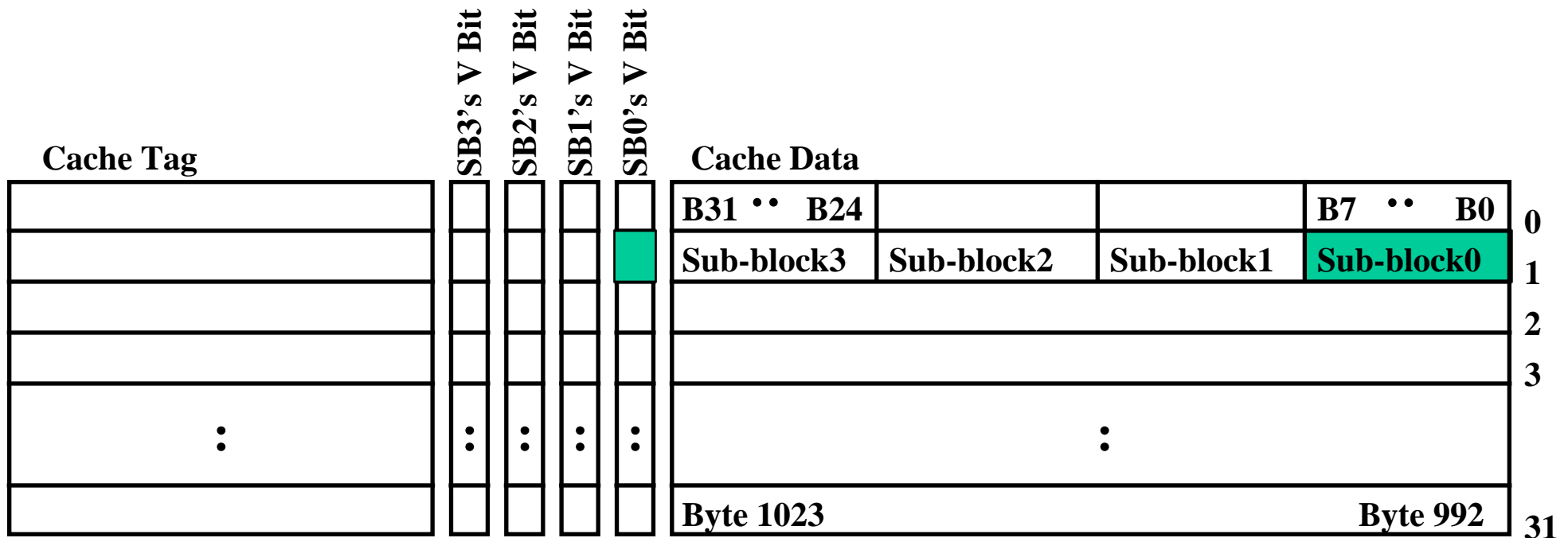
Yes: Write Allocate

No: Write Not Allocate



Sub-block Write

- **Sub-block:**
 - A unit within a block that has its own valid bit
 - **Example: 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block**
 - Each cache entry will have: $32/8 = 4$ valid bits
- **Write miss: only the bytes in that sub-block is brought in.**



Decreasing miss penalty with multilevel caches

- **Add a second level cache:**
 - often primary cache is on the same chip as the processor
 - use SRAMs to add another cache above primary memory (DRAM)
 - miss penalty goes down if data is in 2nd level cache
- **Example:**
 - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
 - Adding 2nd level cache with 20ns access time decreases miss rate to 2%
- **Using multilevel caches:**
 - try and optimise the hit time on the 1st level cache
 - try and optimise the miss rate on the 2nd level cache

Summary (Cache) :

- **The Principle of Locality:**

- **Program access a relatively small portion of the address space at any instant of time.**

- Temporal Locality: Locality in Time
- Spatial Locality: Locality in Space

- **Three Major Categories of Cache Misses:**

- **Compulsory Misses: sad facts of life. Example: cold start misses.**

- **Conflict Misses: increase cache size and/or associativity.**

Nightmare Scenario: ping pong effect!

- **Capacity Misses: increase cache size**

- **Write Policy:**

- **Write Through: need a write buffer. Nightmare: WB saturation**

- **Write Back: control can be complex**

Where to get more information?

- **Your textbook, Ch 7**
- **General reference, Chapter 5 of:**
 - **John Hennessy & David Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers Inc., 1996**
- **A landmark paper on caches:**
 - **Alan Smith, Cache Memories, Computing Surveys, September 1982**
- **A book on everything you need to know about caches:**
 - **Steve Przybylski, Cache and Memory Hierarchy Design: A Performance-Directed Approach, Morgan Kaufmann Publishers Inc., 1990.**