UNIVERSITÀ DI PISA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Cache Memory

Phd: De Vitis Gabriele Antonio

gabrieleantonio.devitis@ing.unipi.it

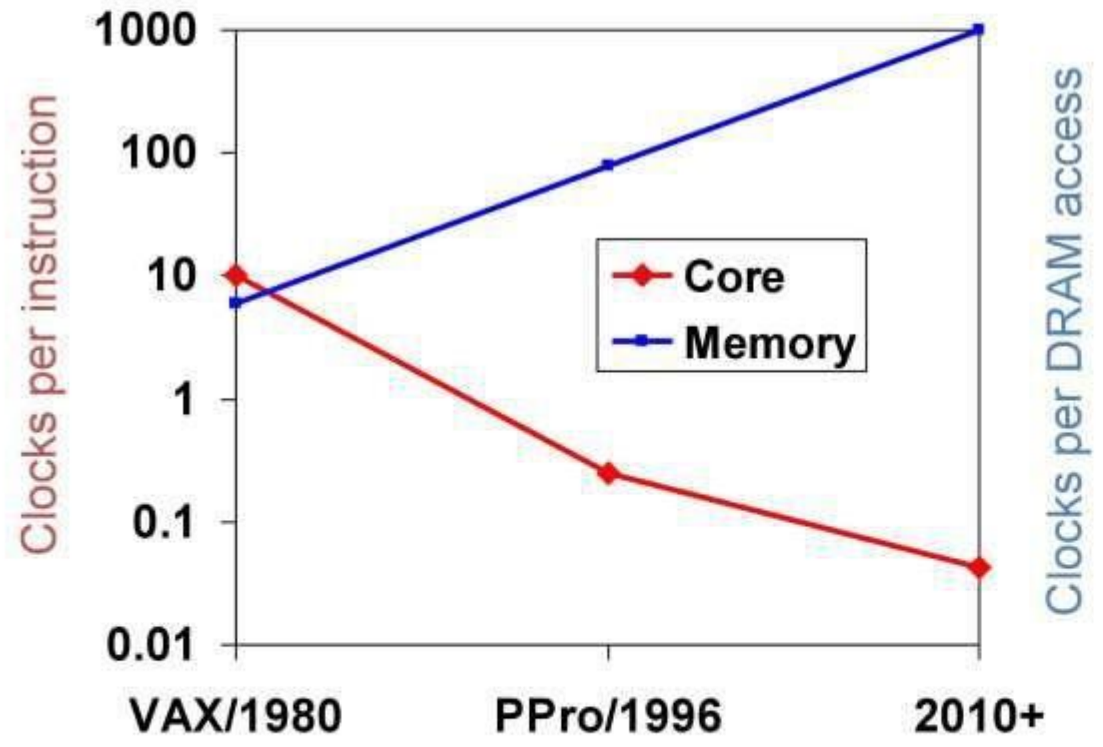
@CompArch2019

Why Cache - Inventor

M. V. Wilkes, "Slave Memories and Dynamic Storage Allocation," ***IEEE Transactions on Electronic Computers*****, vol. EC-14, no. 2, pp. 270-271, April 1965.**

Why Cache - Discussion

- Processor vs DRAM speed disparity continues to grow



- Good memory hierarchy (cache) design is increasingly important to overall performance

Why Cache - Technologies

- Static RAM (SRAM)

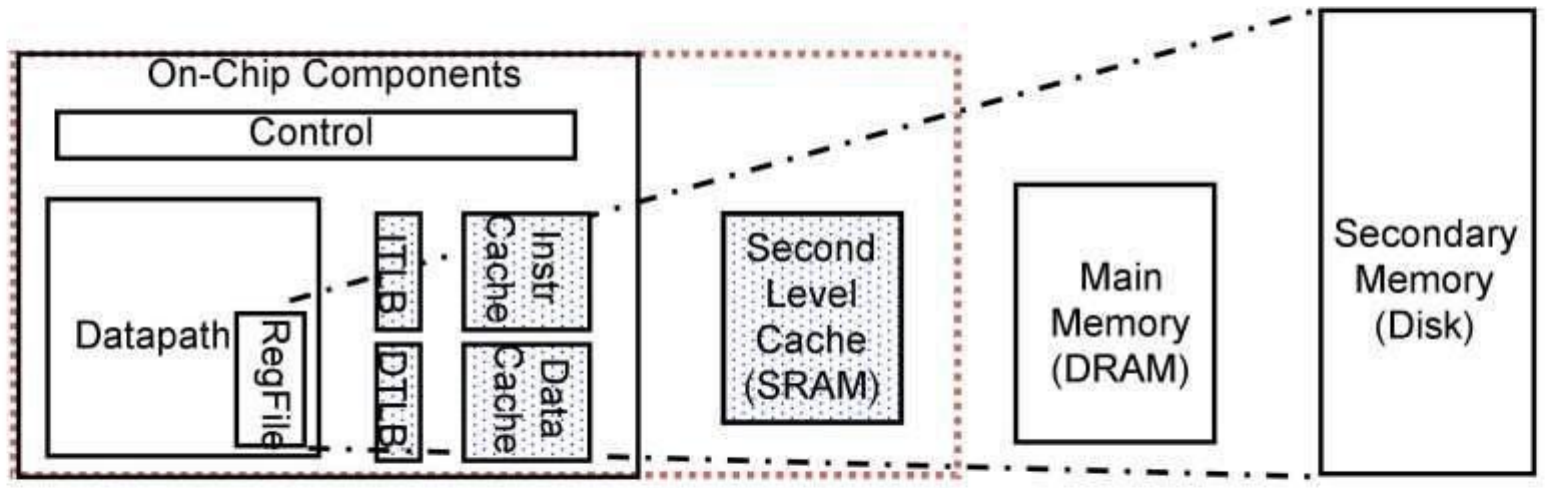
  0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM)

  50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk

  5ms – 20ms, \$0.20 – \$2 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

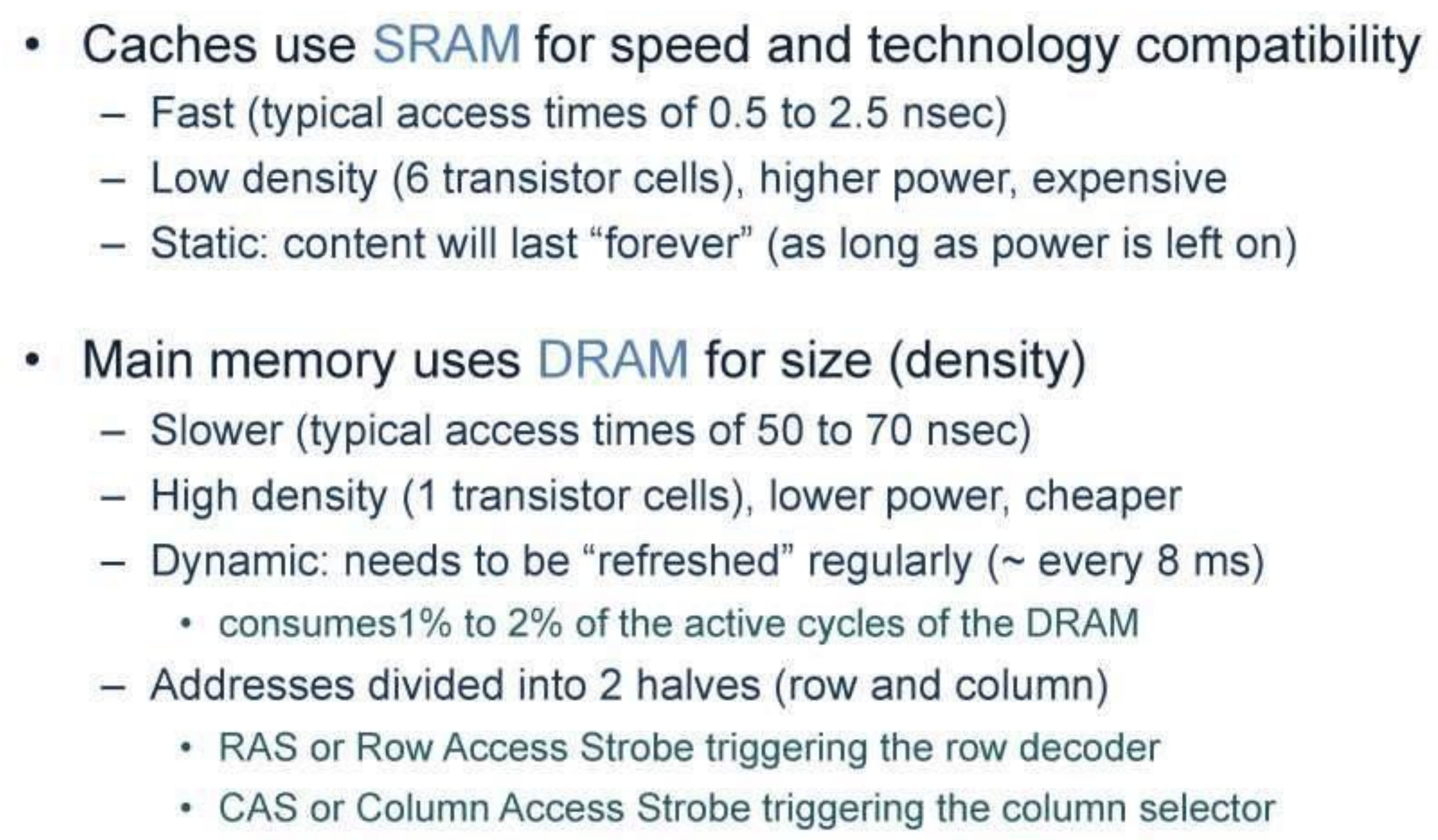
Why Cache - Hierarchy

- 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



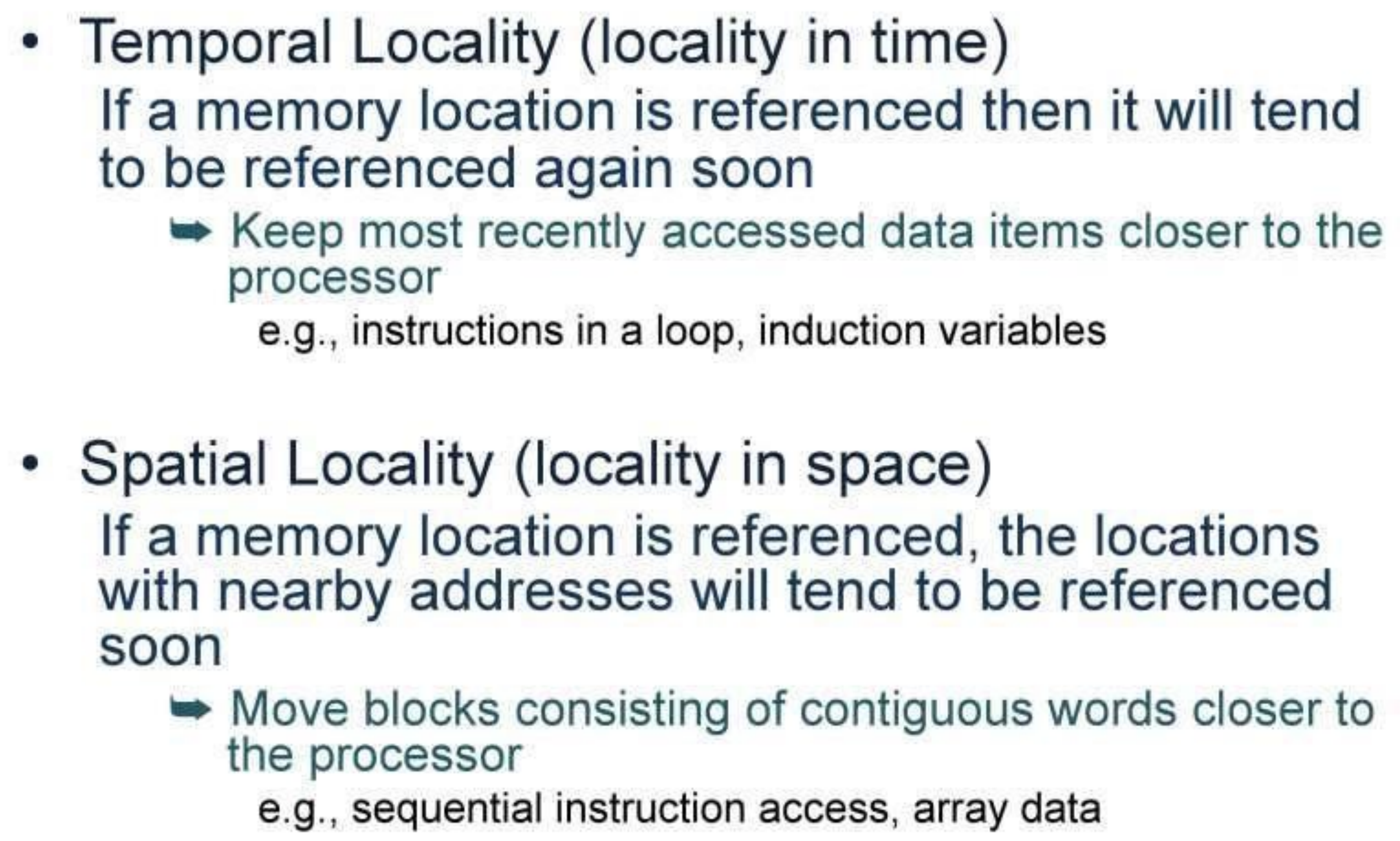
Speed (%cycles):	½'s	1's	10's	100's	10,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost:	highest				lowest

Why Cache - Technologies



- Caches use SRAM for speed and technology compatibility
  - Fast (typical access times of 0.5 to 2.5 nsec)
  - Low density (6 transistor cells), higher power, expensive
  - Static: content will last “forever” (as long as power is left on)
- Main memory uses DRAM for size (density)
  - Slower (typical access times of 50 to 70 nsec)
  - High density (1 transistor cells), lower power, cheaper
  - Dynamic: needs to be “refreshed” regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM
  - Addresses divided into 2 halves (row and column)
    - RAS or Row Access Strobe triggering the row decoder
    - CAS or Column Access Strobe triggering the column selector

Why Cache – Why Does it Works?



- Temporal Locality (locality in time)

  If a memory location is referenced then it will tend to be referenced again soon

  ➡ Keep most recently accessed data items closer to the processor

  e.g., instructions in a loop, induction variables

- Spatial Locality (locality in space)

  If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

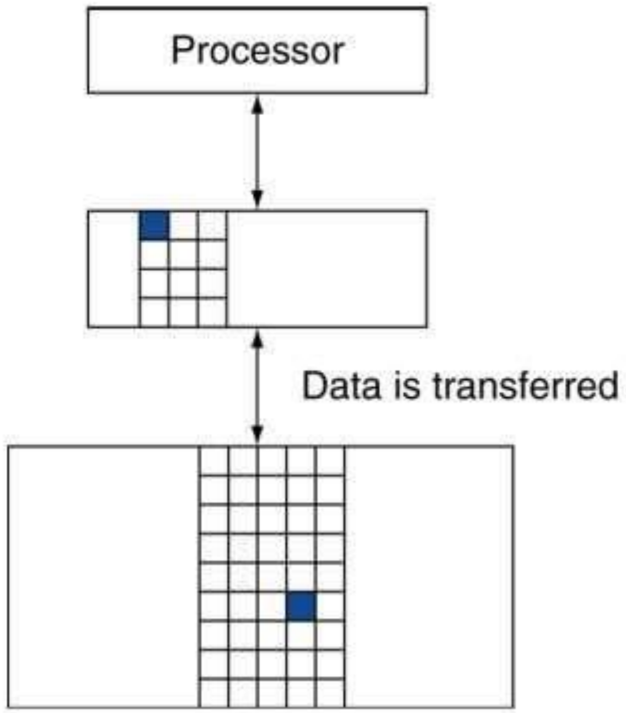
  ➡ Move blocks consisting of contiguous words closer to the processor

  e.g., sequential instruction access, array data

Why Cache - Idea

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Why Cache - Levels

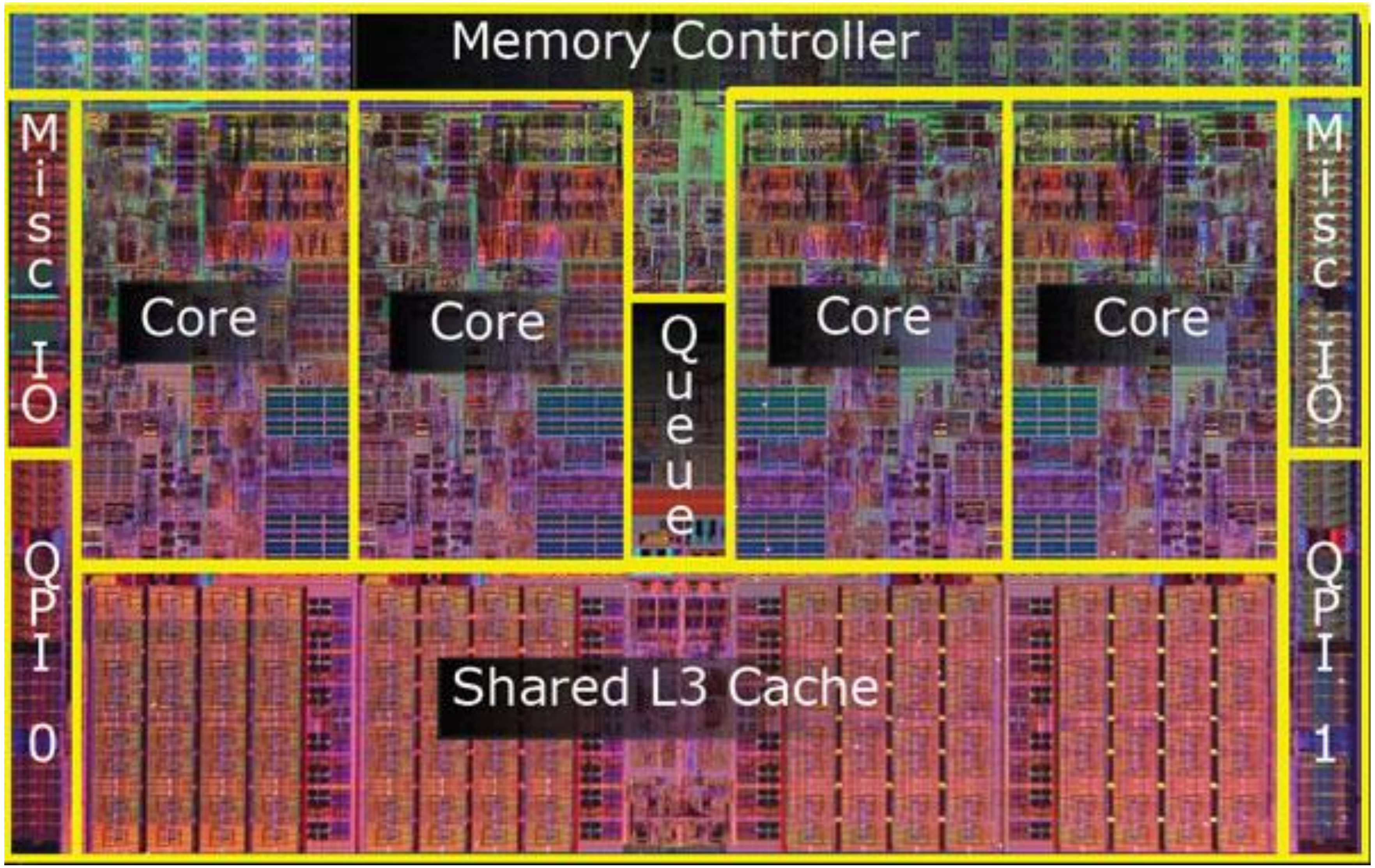


- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses = 1 – hit ratio
  - Then accessed data supplied from upper level

Cache - Terminology

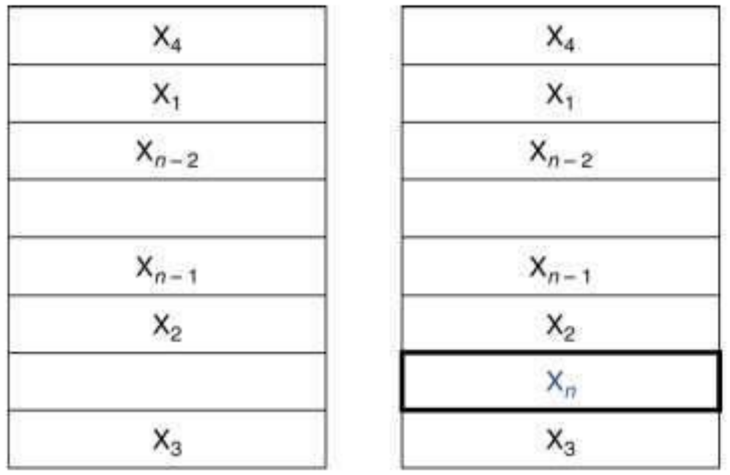
- Block (or line): the minimum unit of information that is present (or not) in a cache
- Hit Rate: the fraction of memory accesses found in a level of the memory hierarchy
- Miss Rate: the fraction of memory accesses *not* found in a level of the memory hierarchy $\Rightarrow$ 1 - (Hit Rate)
- Miss Penalty: Time to replace a block in that level with the corresponding block from a lower

Cache – Inside CPU i7 Intel



Cache Memory - Idea

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

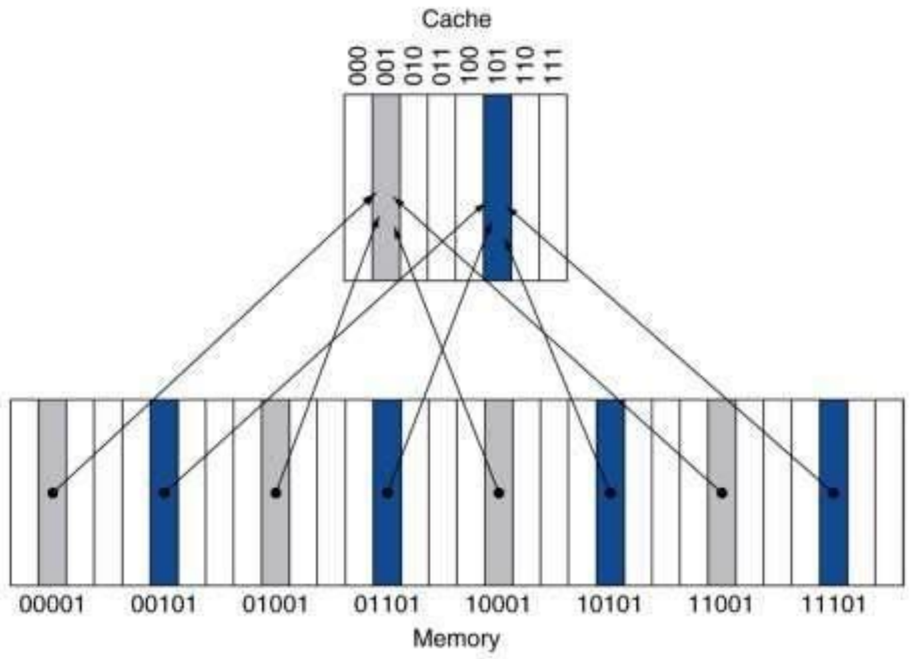


a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

Cache Memory – Local Mapping

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

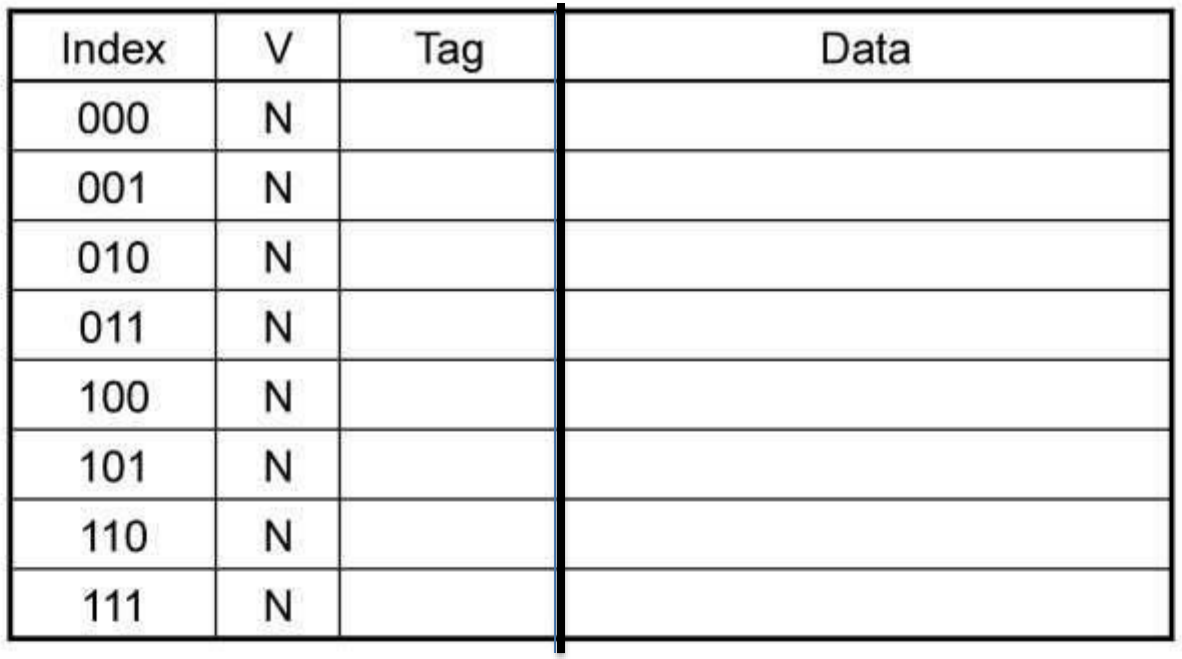
Cache Memory – Tags and Valid bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
    - Actually, only need the high-order bits
    - Called the tag

- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

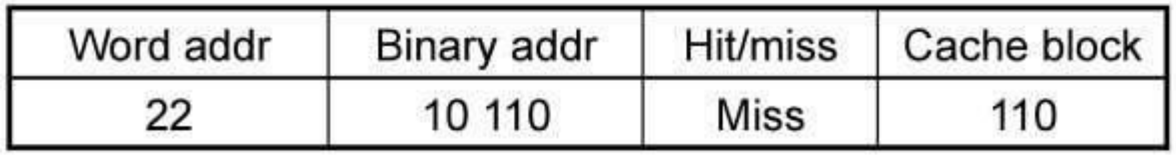
Cache Memory – Direct Mapping example (1)

- 8-blocks, 1 word/block, direct mapped
- Initial state

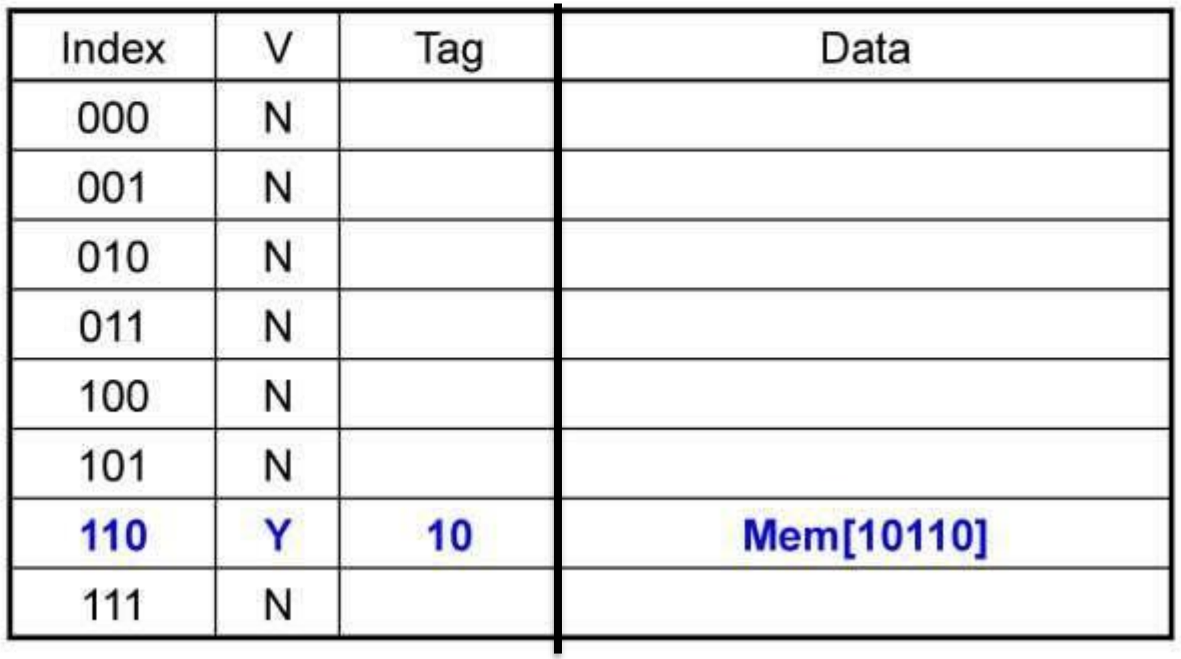


Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Memory – Direct Mapping example (2)



Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110



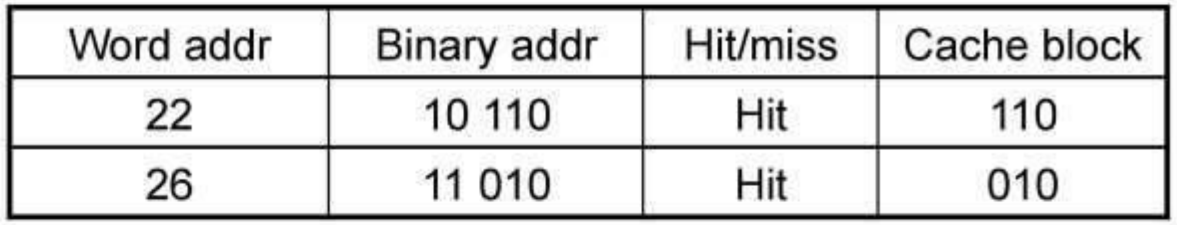
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Memory – Direct Mapping example (3)

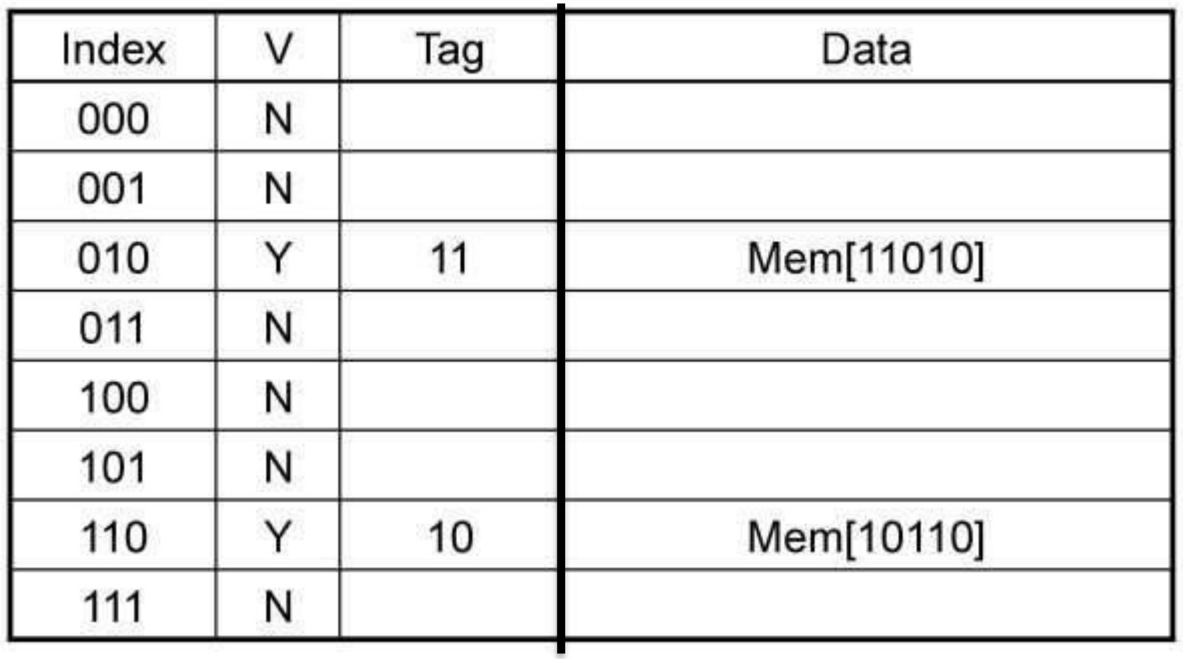
Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Memory – Direct Mapping example (4)



Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010



Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Memory – Direct Mapping example (5)

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

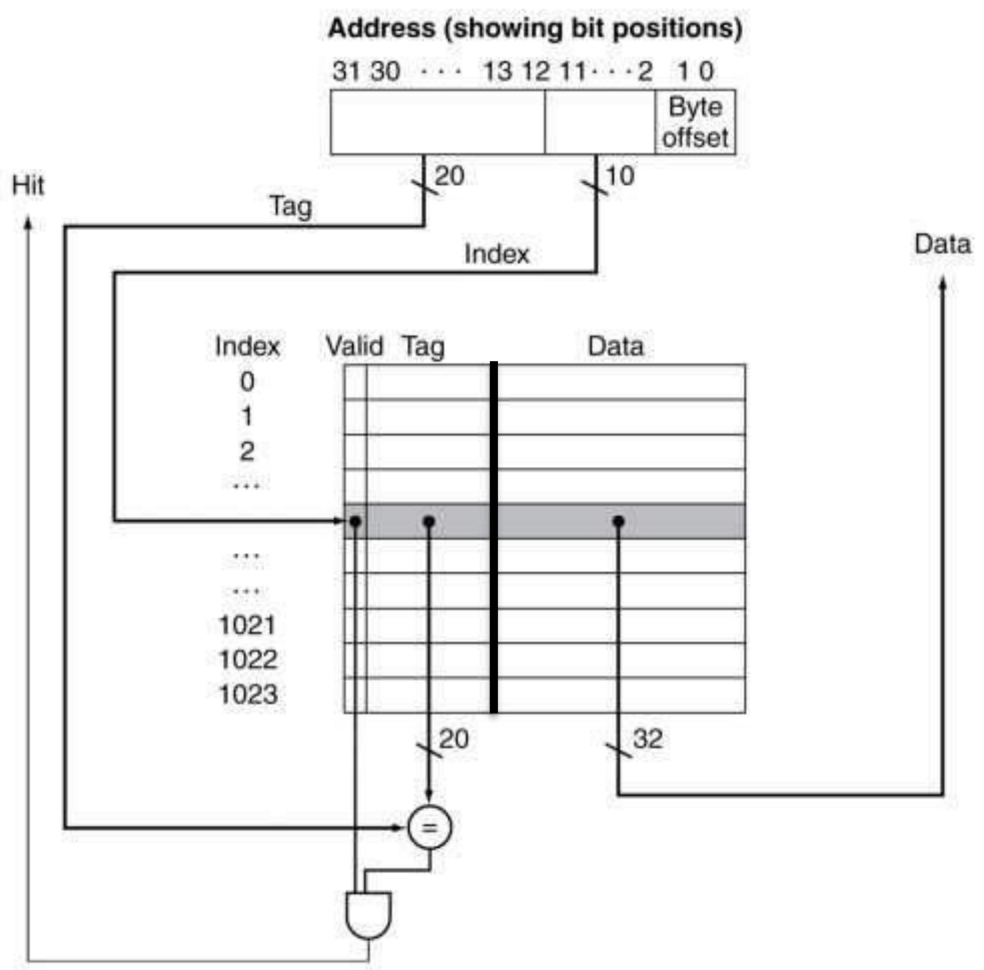
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Memory – Direct Mapping example (6)

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Memory – Direct Mapping Schema



Cache Memory – Addressing Example

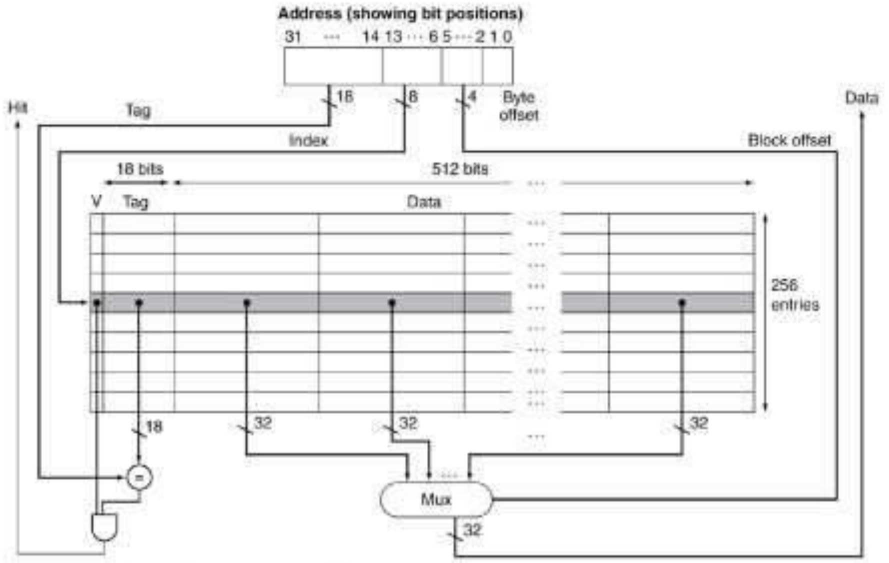
- 256 blocks, 16 words/block
  - To what block number does address 19200 map?

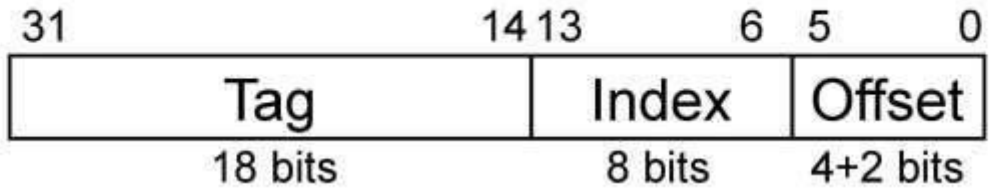
- Block address

  $= \lfloor 19200/(16*4) \rfloor = 300$

- Block number

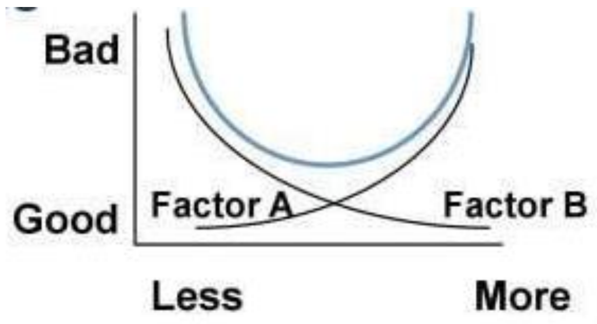
  $= 300 \bmod 256 = 44$



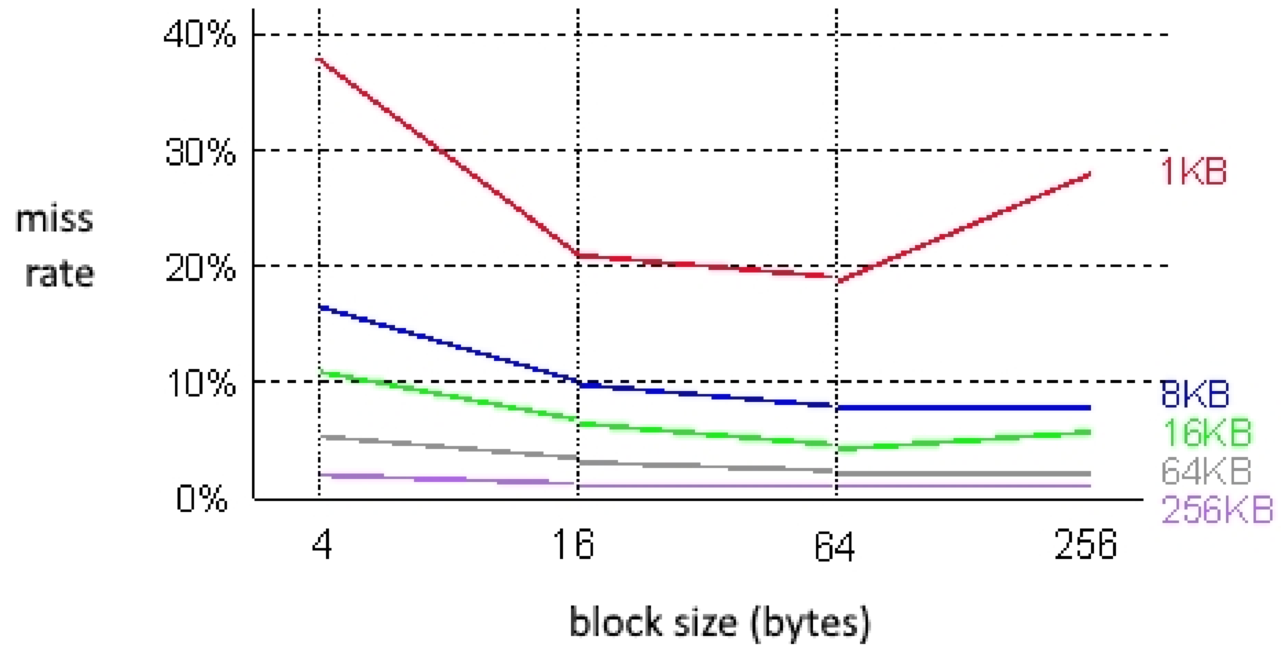


Cache Memory – How many block in cache

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer blocks
    - More competition ⇒ increased miss rate
  - Larger blocks ⇒ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help



Cache Memory – Miss Rate: Block Size vs Cache Size



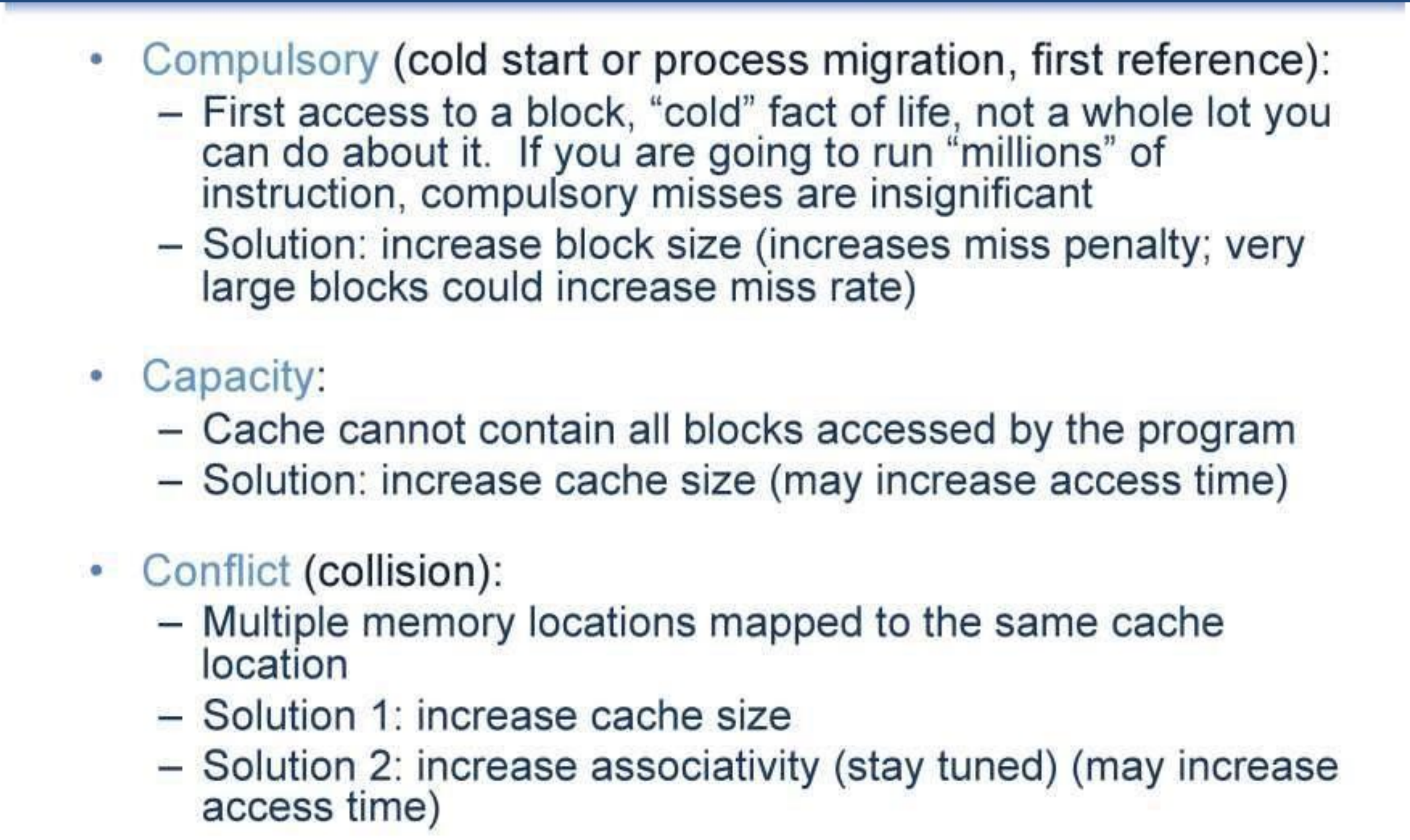
- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Cache Memory – Cache Miss (1)

- On cache hit, CPU proceeds normally

- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Cache Memory – Cache Miss (2)



- Compulsory (cold start or process migration, first reference):
  - 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 (stay tuned) (may increase access time)

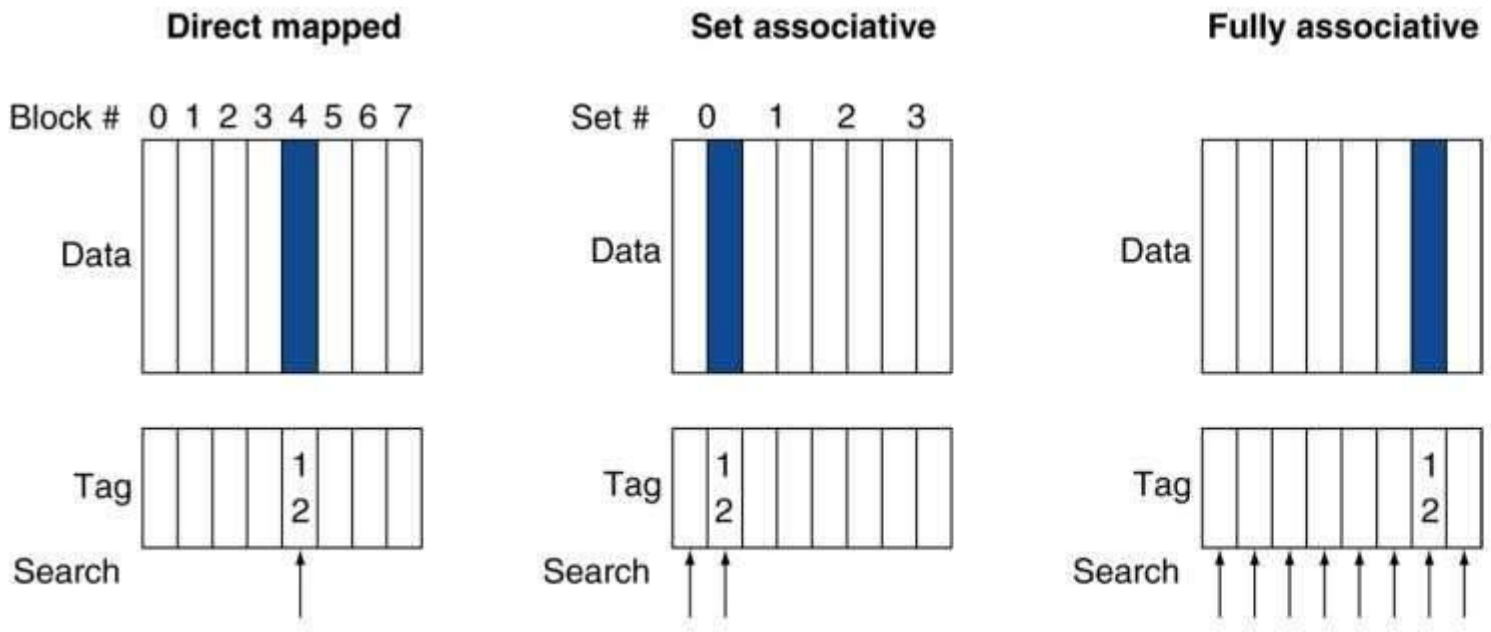
Cache Memory – Associative Cache (1)

- Allow for more flexible block placement:
  - In a direct mapped cache a memory block maps to exactly one cache block
  - At the other extreme, we can allow a memory block 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)

Cache Memory – Associative Cache (2)

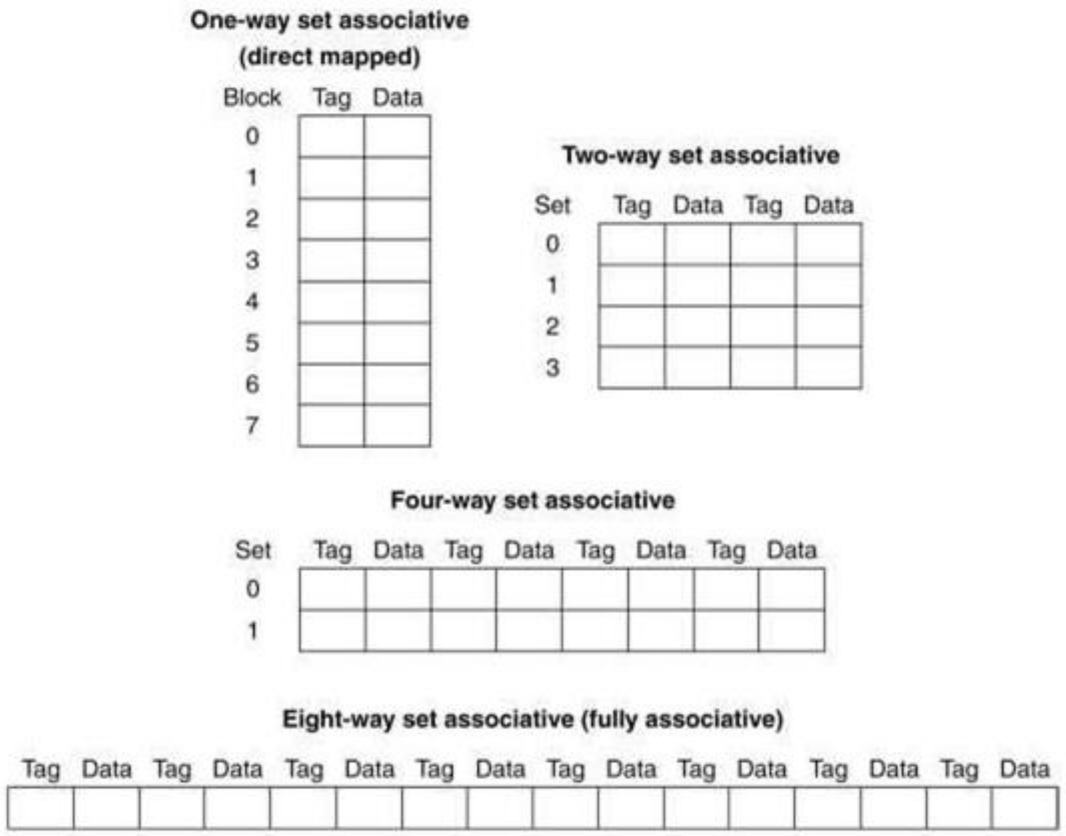
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) *modulo* (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Cache Memory – Associative Cache (3)

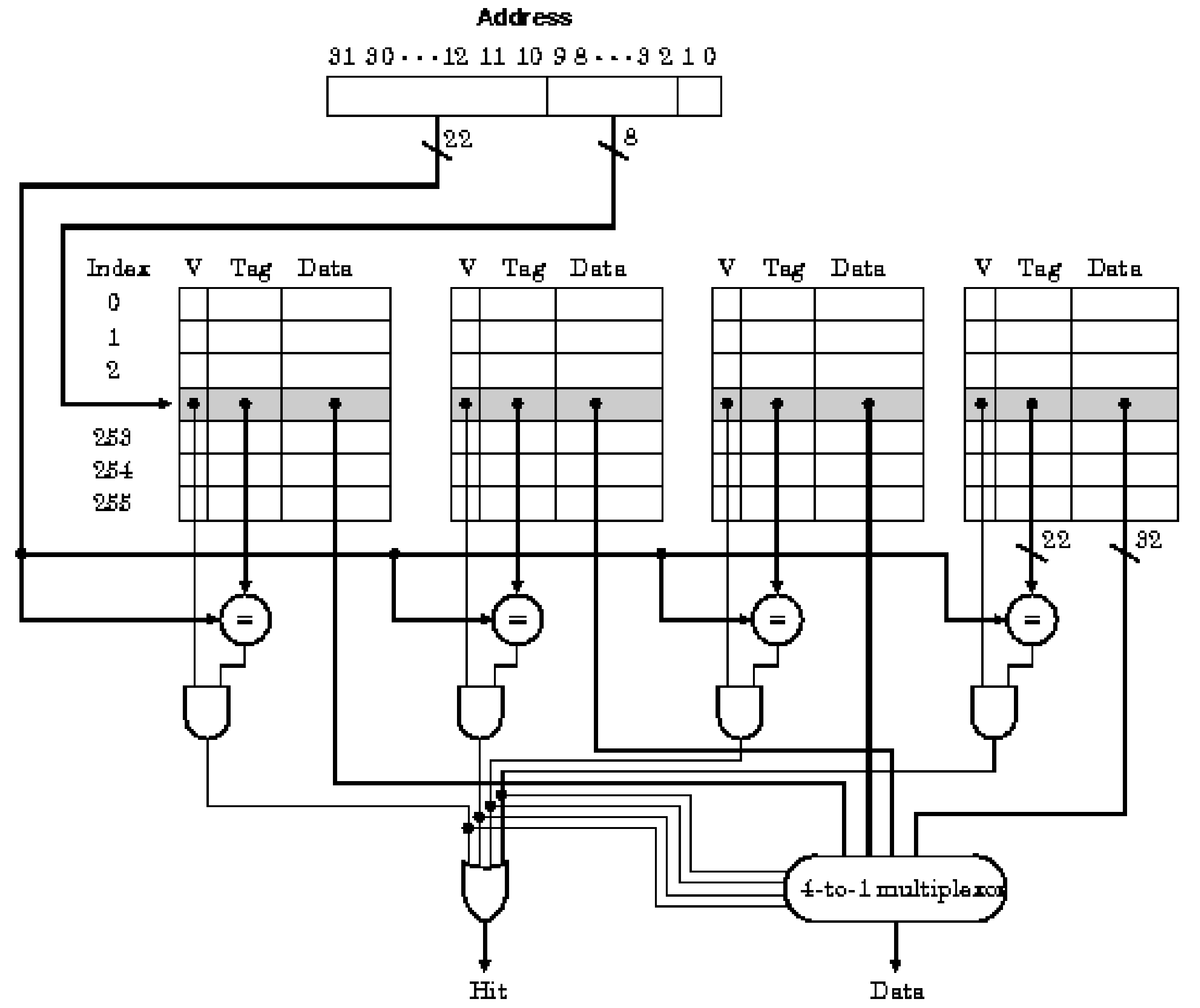


Cache Memory – Associative Cache (4)

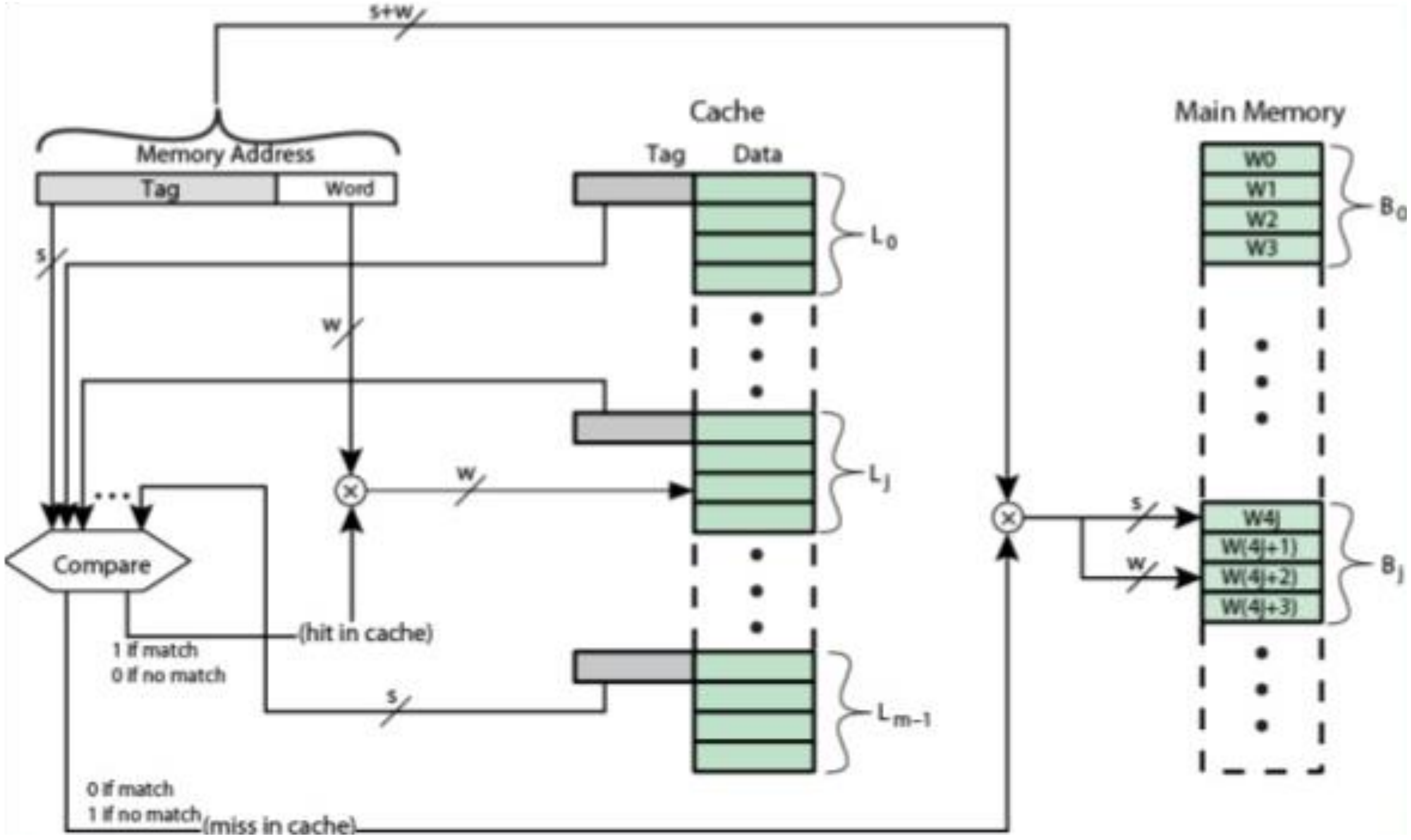
For a cache with 8 entries:


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

Cache Memory – Set Associative Cache

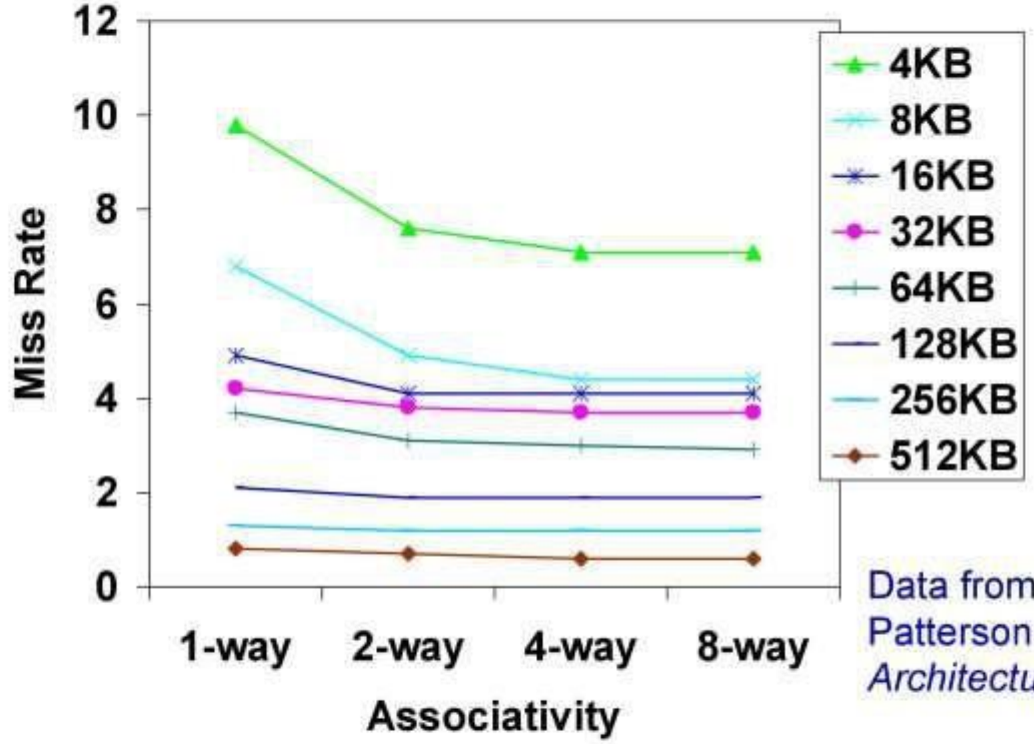


Cache Memory – Fully Associative Cache



Cache Memory – Miss in Associative

- 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)