Cache Memory

S.M. Shovan

Contents

Caches

Cache: A small, interim storage component that transparently retains (caches) data from recently accessed locations

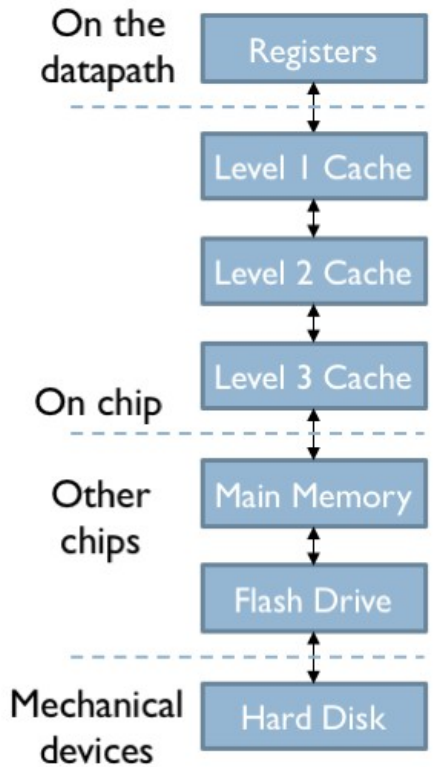
- Very fast access if data is cached, otherwise accesses slower, larger cache or memory
- Exploits the locality principle

Computer systems often use multiple levels of caches

Caching widely applied beyond hardware (e.g., web caches)

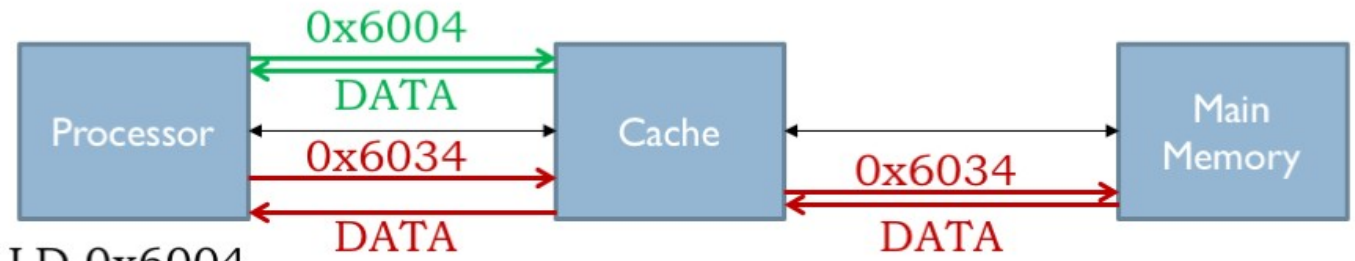
A Typical Memory Hierarchy

- Everything is a cache for something else…



Access time	Capacity	Managed By
1 cycle	1 KB	Software/Compiler
2-4 cycles	32 KB	Hardware
10 cycles	256 KB	Hardware
40 cycles	10 MB	Hardware
200 cycles	10 GB	Software/OS
10-100us	100 GB	Software/OS
10ms	1 TB	Software/OS

Cache Access



- Processor sends address to cache
- Two options:
  - Cache hit: Data for this address in cache, returned quickly
  - Cache miss: Data not in cache
    - Fetch data from memory, send it back to processor
    - Retain this data in the cache (replacing some other data)
  - Processor must deal with variable memory access time

Cache Metrics

Hit Ratio: $$HR = \frac{hits}{hits + misses} = 1 - MR$$

Miss Ratio: $$MR = \frac{misses}{hits + misses} = 1 - HR$$

Average Memory Access Time (AMAT):

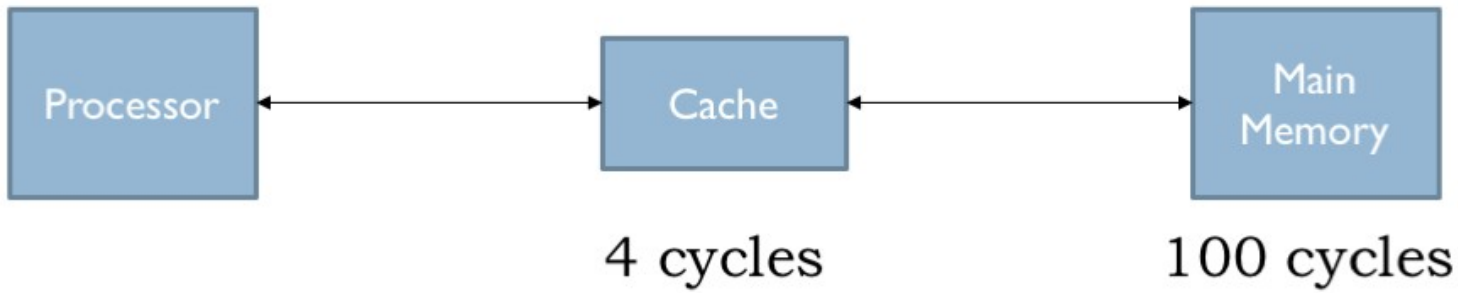
$$AMAT = HitTime + MissRatio \times MissPenalty$$

- Goal of caching is to improve AMAT
- Formula can be applied recursively in multi-level hierarchies:

$$AMAT = HitTime_{L1} + MissRatio_{L1} \times AMAT_{L2} =$$

$$AMAT = HitTime_{L1} + MissRatio_{L1} \times (HitTime_{L2} + MissRatio_{L2} \times AMAT_{L3}) = \ldots$$

Example: How High of a Hit Ratio?



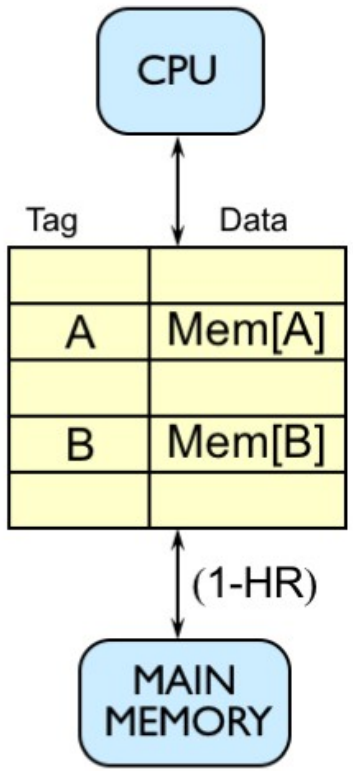
What hit ratio do we need to break even?
(Main memory only: AMAT = 100)

$$100 = 4 + (1 - HR) \times 100 \Rightarrow HR = 4\%$$

What hit ratio do we need to achieve AMAT = 5 cycles?

$$5 = 4 + (1 - HR) \times 100 \Rightarrow HR = 99\%$$

Basic Cache Algorithm



ON REFERENCE TO Mem[X]:
Look for X among cache tags...

HIT: *X = TAG(i) , for some cache line i*

- READ: return DATA(i)
- WRITE: change DATA(i); Start Write to Mem(X)

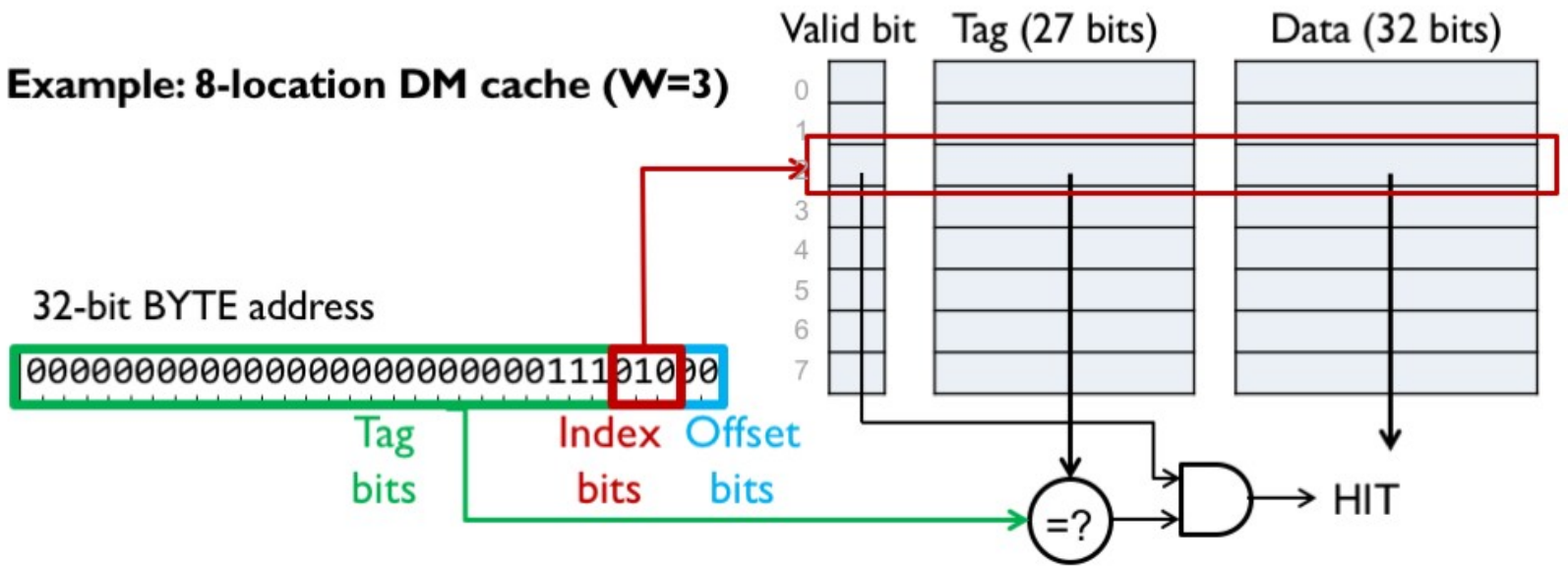
MISS: *X not found in TAG of any cache line*

- REPLACEMENT SELECTION:
  Select some line k to hold Mem[X] (Allocation)
- READ: Read Mem[X]
  Set TAG(k)=X, DATA(k)=Mem[X]
- WRITE: Start Write to Mem(X)
  Set TAG(k)=X, DATA(k)= new Mem[X]

Q: How do we "search" the cache?

Direct-Mapped Caches

- Each word in memory maps into a single cache line
- Access (for cache with 2^W lines):
  - Index into cache with W address bits (the index bits)
  - Read out valid bit, tag, and data
  - If valid bit == 1 and tag matches upper address bits, HIT



Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indexes → 6 index bits

Read Mem[0x400C]

0100 0000 0000 1100

TAG: 0x40

INDEX: 0x3

OFFSET: 0x0

HIT, DATA 0x42424242

	Valid bit	Tag (24 bits)	Data (32 bits)
0	1	0x000058	0xDEADBEEF
1	1	0x000058	0x00000000
2	0	0x000058	0x00000007
3	1	0x000040	0x42424242
4	1	0x000007	0x6FBA2381
⋮	⋮	⋮	⋮
63	1	0x000058	0xF7324A32

Would 0x4008 hit?

INDEX: 0x2 → tag mismatch → miss

What are the addresses of data in indexes 0, 1, and 2?

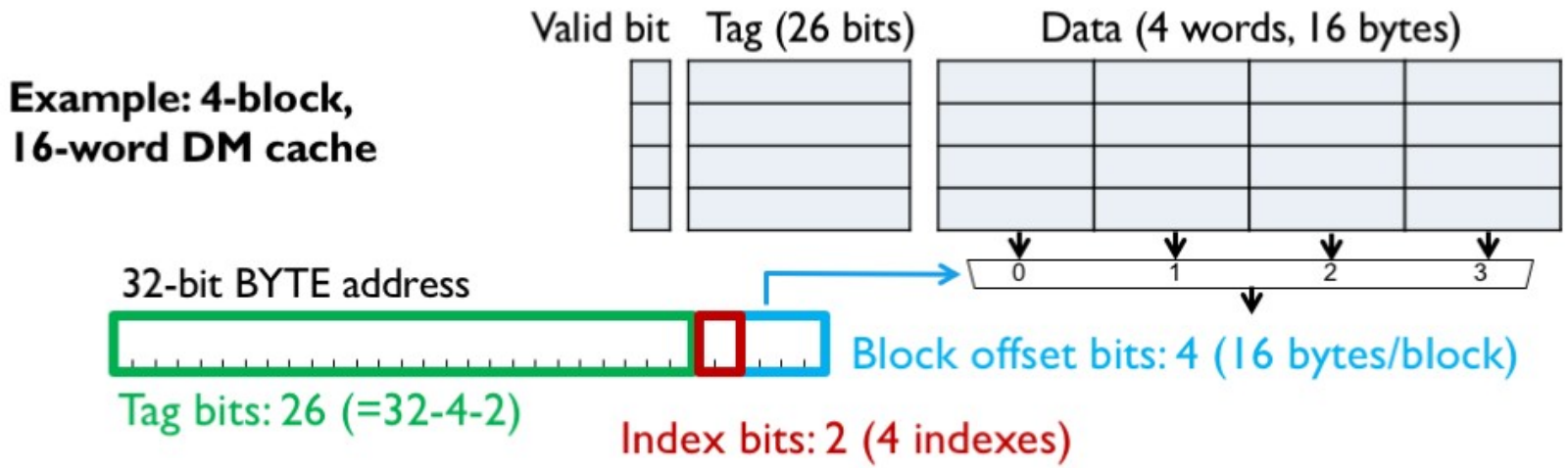
TAG: 0x58 → 0101 1000 iiii ii00 (substitute line # for iiiiii) → 0x5800, 0x5804, 0x5808

Part of the address (index bits) is **encoded in the location**!
Tag + Index bits unambiguously identify the data's address

Block Size

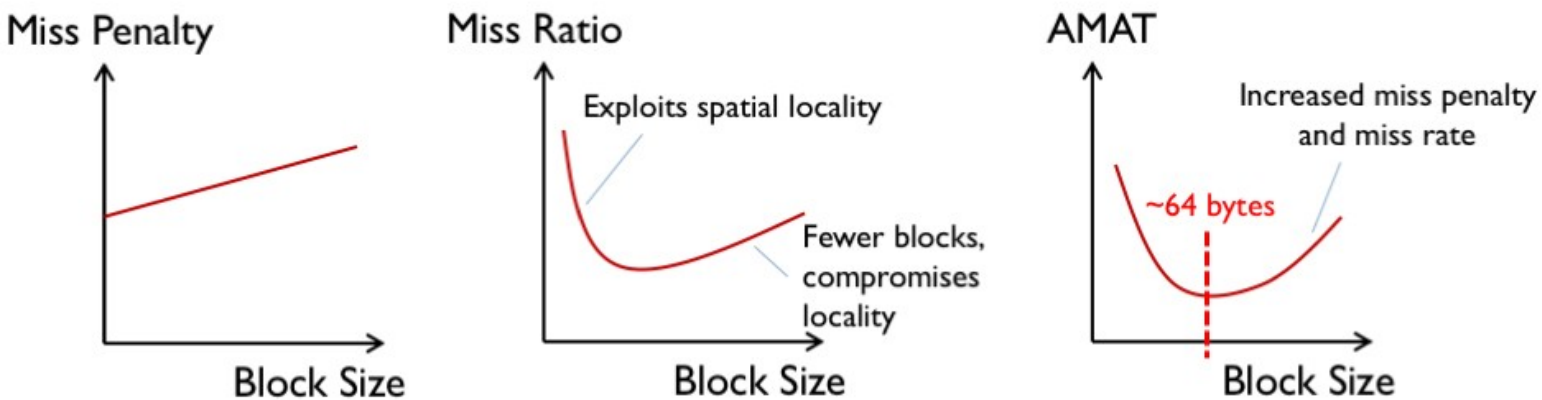
Take advantage of locality: increase block size

- Another advantage: Reduces size of tag memory!
- Potential disadvantage: Fewer blocks in the cache



Block Size Tradeoffs

- Larger block sizes…
  - Take advantage of spatial locality
  - Incur larger miss penalty since it takes longer to transfer the block into the cache
  - Can increase the average hit time and miss rate
- Average Access Time (AMAT) = HitTime + MissPenalty*MR



Direct-Mapped Cache Problem: Conflict Misses

Loop A:
Pgm at 1024,
data at 37:

Word Address	Cache Line index	Hit/ Miss
1024	0	HIT
37	37	HIT
1025	1	HIT
38	38	HIT
1026	2	HIT
39	39	HIT
1024	0	HIT
37	37	HIT
…		

Assume:
- 1024-line DM cache
- Block size = 1 word

Consider looping code, in steady state

Assume WORD, not BYTE, addressing

Loop B:
Pgm at 1024,
data at 2048:

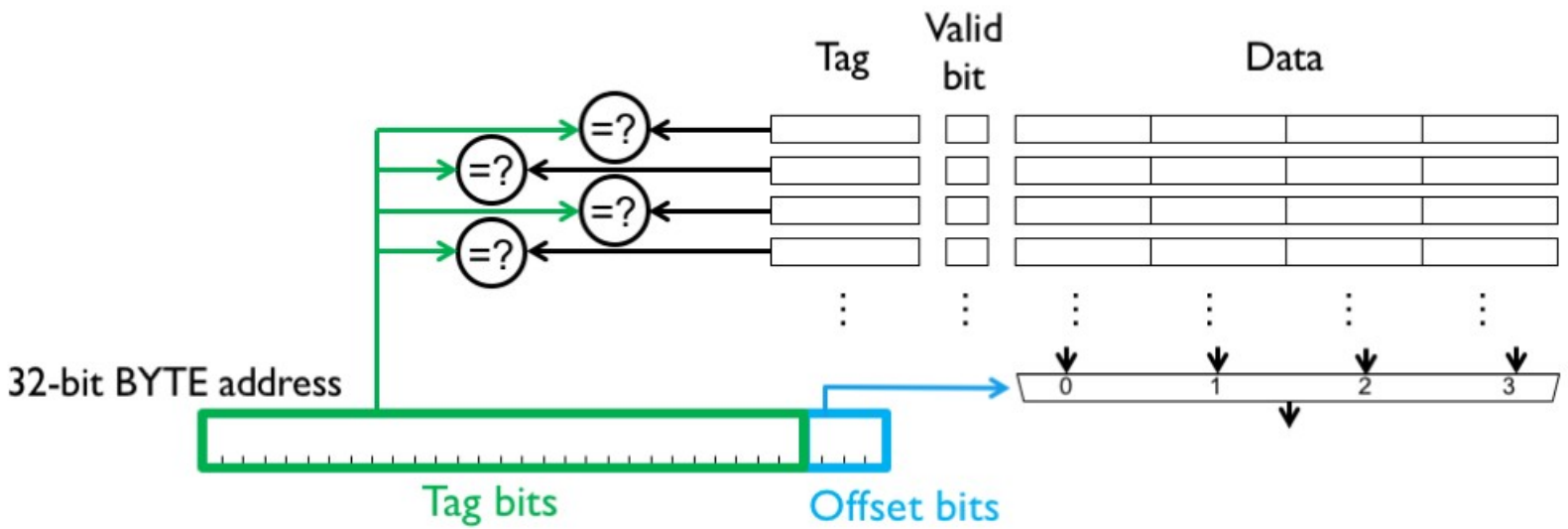
1024	0	MISS
2048	0	MISS
1025	1	MISS
2049	1	MISS
1026	2	MISS
2050	2	MISS
1024	0	MISS
2048	0	MISS
…		

Inflexible mapping (each address can only be in one cache location) → Conflict misses!

Fully-Associative Cache

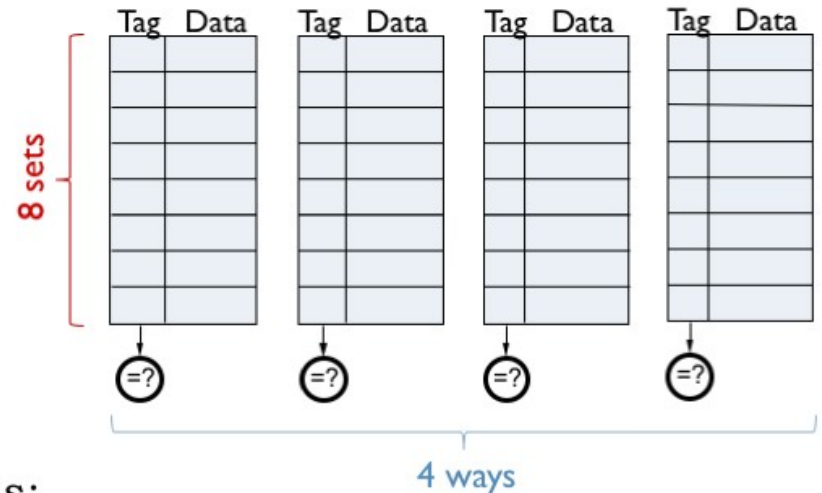
Opposite extreme: Any address can be in any location

- No cache index!
- Flexible (no conflict misses)
- Expensive: Must compare tags of all entries in parallel to find matching one (can do this in hardware, this is called a CAM)



N-way Set-Associative Cache

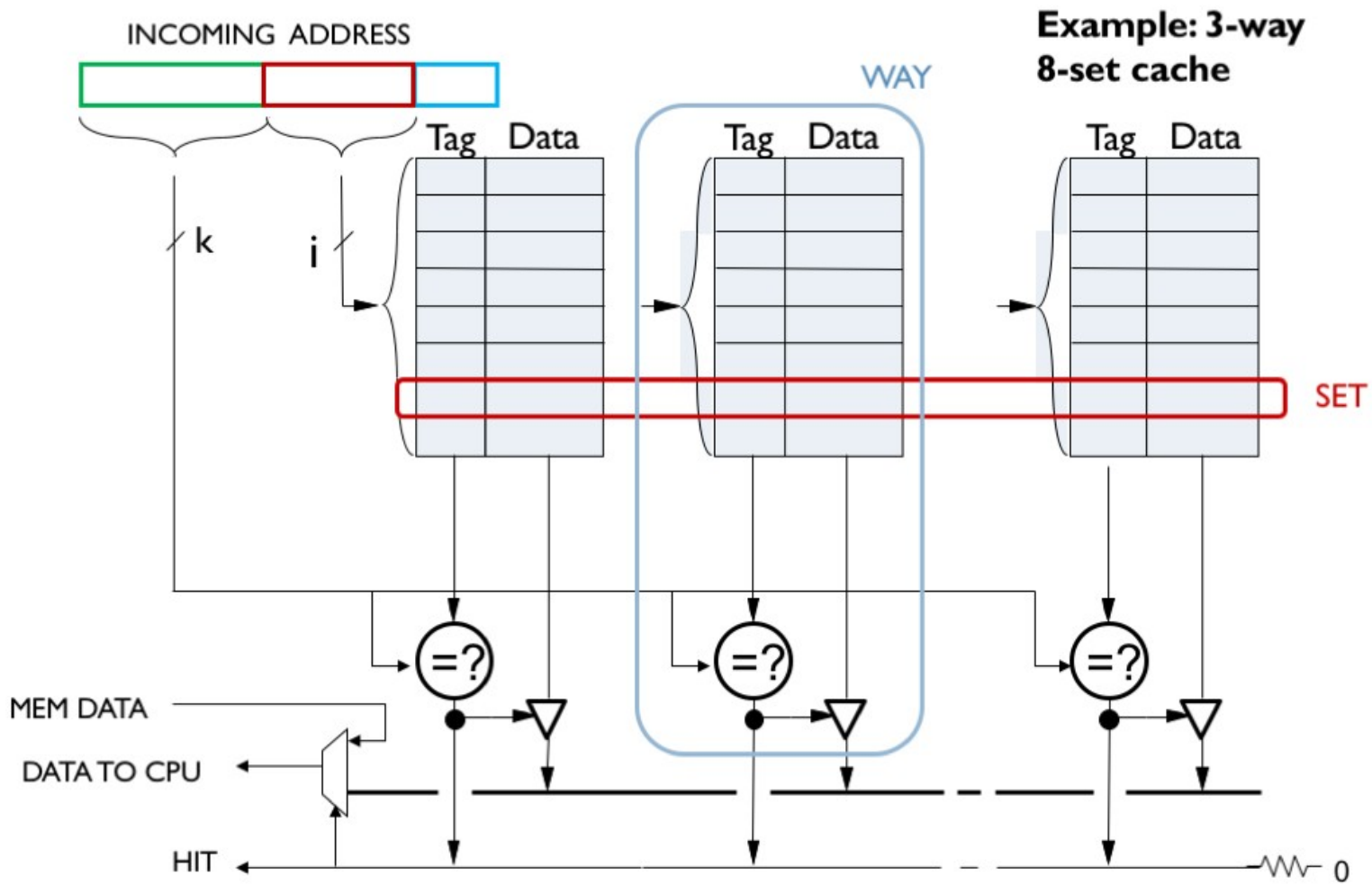
- Compromise between direct-mapped and fully associative
  - Nomenclature:
    - # Rows = # Sets
    - # Columns = # Ways
    - Set size = #ways
      = "set associativity"
      (e.g., 4-way → 4 entries/set)
  - compare all tags from all ways in parallel



- An N-way cache can be seen as:
  - N direct-mapped caches in parallel

- Direct-mapped and fully-associative are just special cases of N-way set-associative

N-way Set-Associative Cache



Source

- https://computationstructures.org/lectures/caches/caches.html#19