CSE 520
Computer Architecture II

Caching Principles

Prof. Michel A. Kinsy

Caching Principles

- Cache contains copies of some of Main Memory
  - Those storage locations recently used
    - When Main Memory address A is referenced in CPU
    - Cache checked for a copy of contents of A
  - If found, cache hit
    - Copy used
    - No need to access Main Memory
  - If not found, cache miss
    - Main Memory accessed to get contents of A
    - Copy of contents also loaded into cache

Caching principles

- Cache size (in bytes or words)
  - Total cache capacity
  - A larger cache can hold more of the program's useful data but is more costly and likely to be slower
- Block or cache-line size
  - Unit of data transfer between cache and main
  - With a larger cache line, more data is brought in cache with each miss. This can improve the hit rate but also may bring low-utility data in cache

Caching principles

- Placement policy
  - Determining where an incoming cache line is stored
  - More flexible policies imply higher hardware cost and may or may not have performance benefits (due to more complex data location)
- Replacement policy
  - Determining which of several existing cache blocks (into which a new cache line can be mapped) should be overwritten
  - Typical policies: choosing a random or the least recently used block

Caching Principles

- Compulsory misses
  - With on-demand fetching, first access to any item is a miss
- Capacity misses
  - We have to evict some items to make room for others
  - This leads to misses that are not incurred with an infinitely large cache
- Conflict misses
  - The placement scheme may force us to displace useful items to bring in other items
  - This may lead to misses in future

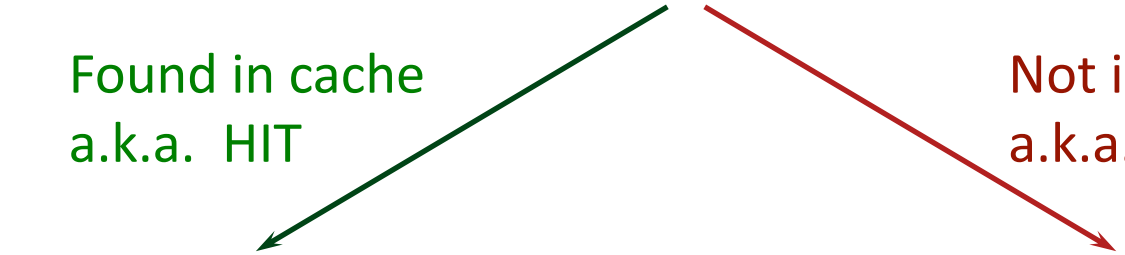
Caching principles

- Line width (2^W)
  - Too small a value for W causes a lot of main memory accesses
  - Too large a value increases the miss penalty and may tie up cache space with low-utility items that are replaced before being used

- Set size or associativity (2^S)
  - Direct mapping (S = 0) is simple and fast
  - Greater associativity leads to more complexity, and thus slower access, but tends to reduce conflict misses

Cache Algorithm (Read)

- Look at Processor Address, search cache tags to find match. Then either



Found in cache
a.k.a. HIT

Return copy of data from cache

Not in cache
a.k.a. MISS

Read block of data from Main Memory

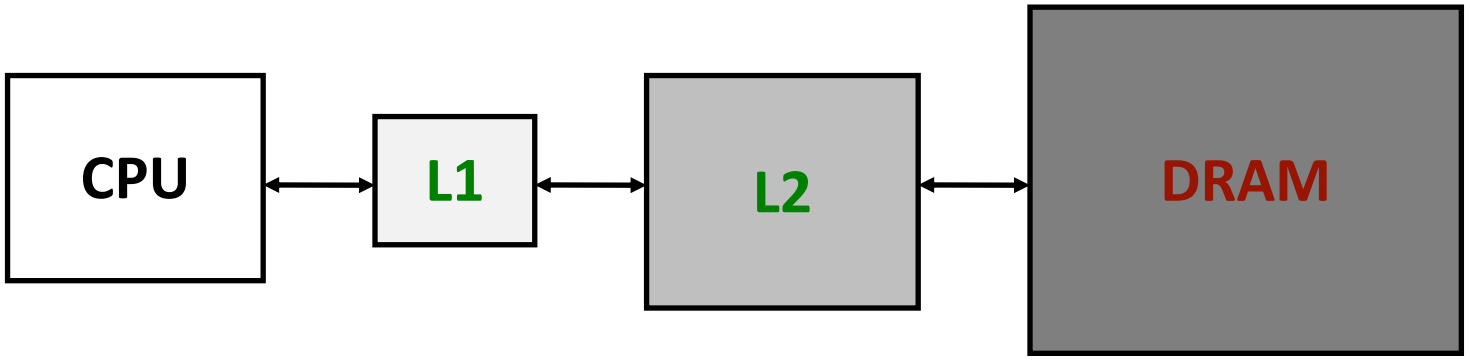
Wait …

Return data to processor and update cache

Q: Which line do we replace?

Caches

- Local miss rate = misses in cache / accesses to cache
- Global miss rate = misses in cache / CPU memory accesses
- Misses per instruction = misses in cache / number of instructions



Cache Performance Metrics

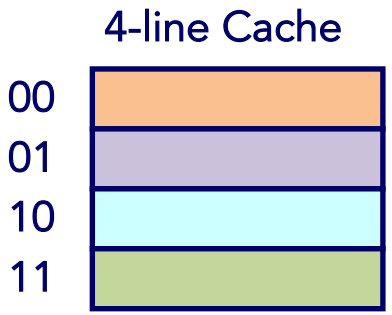
- Cache miss rate
  - Number of cache misses divided by number of accesses
- Cache hit time
  - Time between sending address and data returning from cache
- Cache miss latency
  - Time between sending address and data returning from next-level cache/memory
- Cache miss penalty
  - Extra processor stall caused by next-level cache/memory access

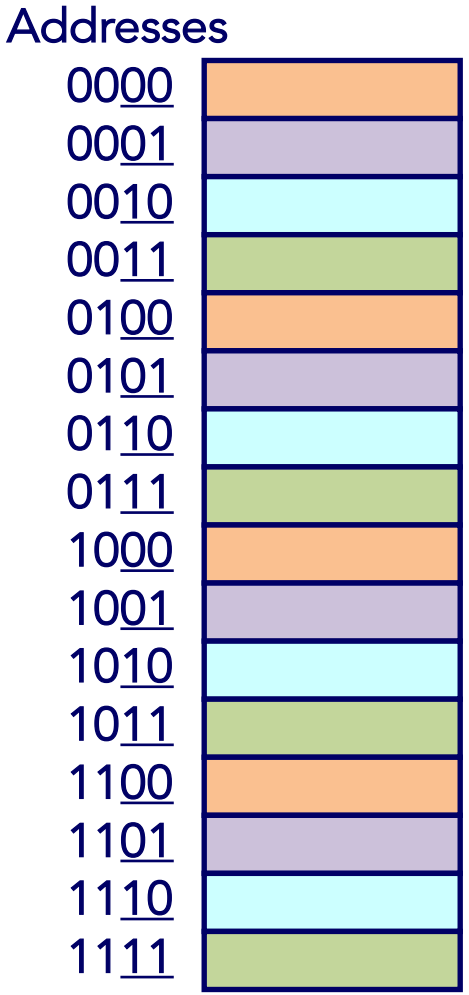
Average Memory Access Time

- Average Memory Access Time (AMAT)
  - AMAT = Hit time + (Miss rate x Miss penalty)
  - Memory stall cycles = Memory accesses x miss rate x miss penalty
  - CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time
  - *CPI = ideal CPI + average stalls per instruction*
- Having L1 and L2 Caches
- AMAT = $\text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$
  - $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$
- AMAT = $\text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$

Placement Policy

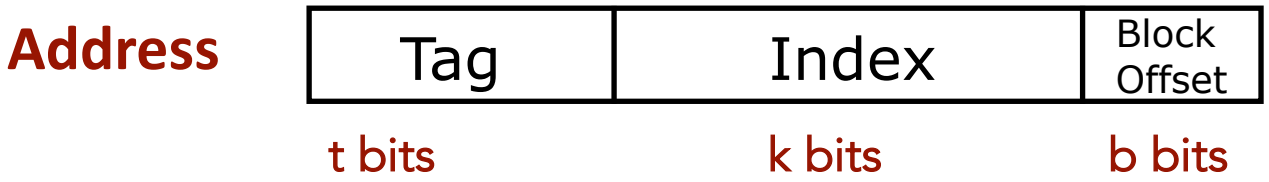
- There are multiple approaches
  - Modulo operation creates a set that we can use for placing data blocks into the cache
  - The result of the modulo operation with modulus n is always an integer between 0 and n-1



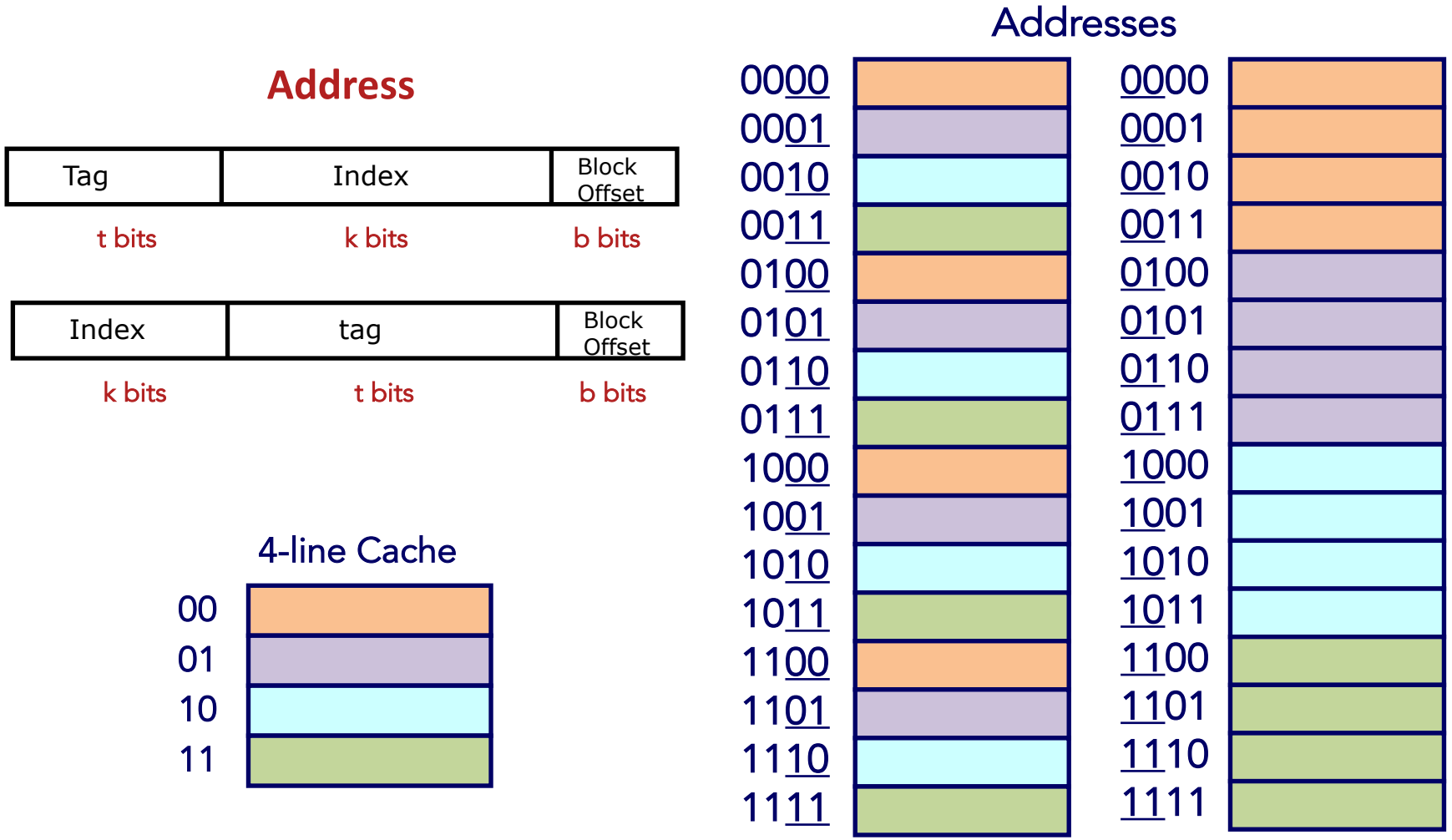


Address Bit-Field Partitioning

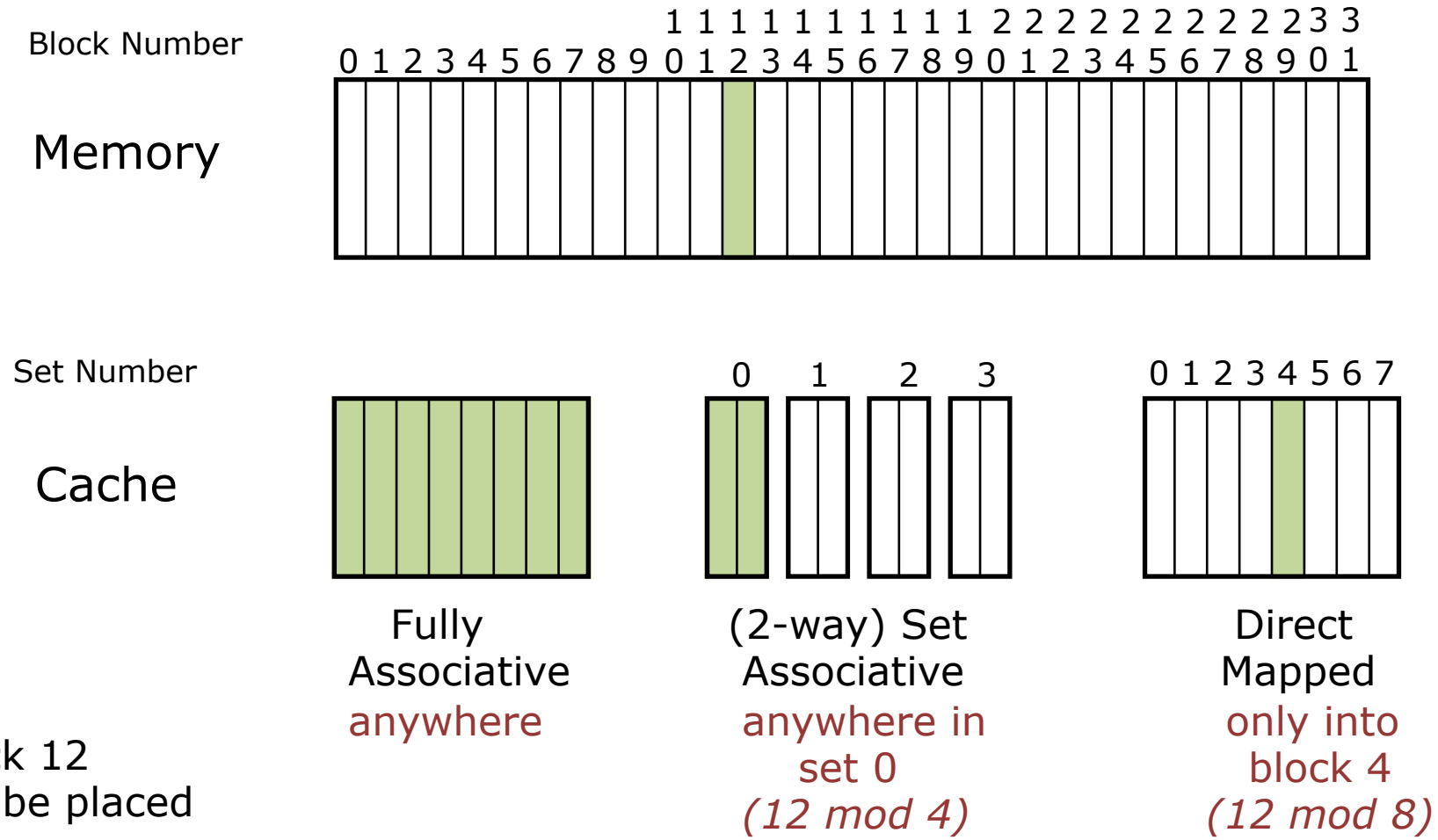
- The address (e.g., 32-bit) issued by the CPU is generally divided into 3 fields
  - Tag
    - Serves as the unique identifier for a group of data
    - Different regions of memory may be mapped to the same cache location/block
    - The tag is used to differentiate between them
  - Index
    - It is used to index into the cache structure
  - Block Offset
    - The least significant bits are used to determine the exact data word
    - If the block size is B then b = $\log_2 B$ bits will be needed in the address to specify data word



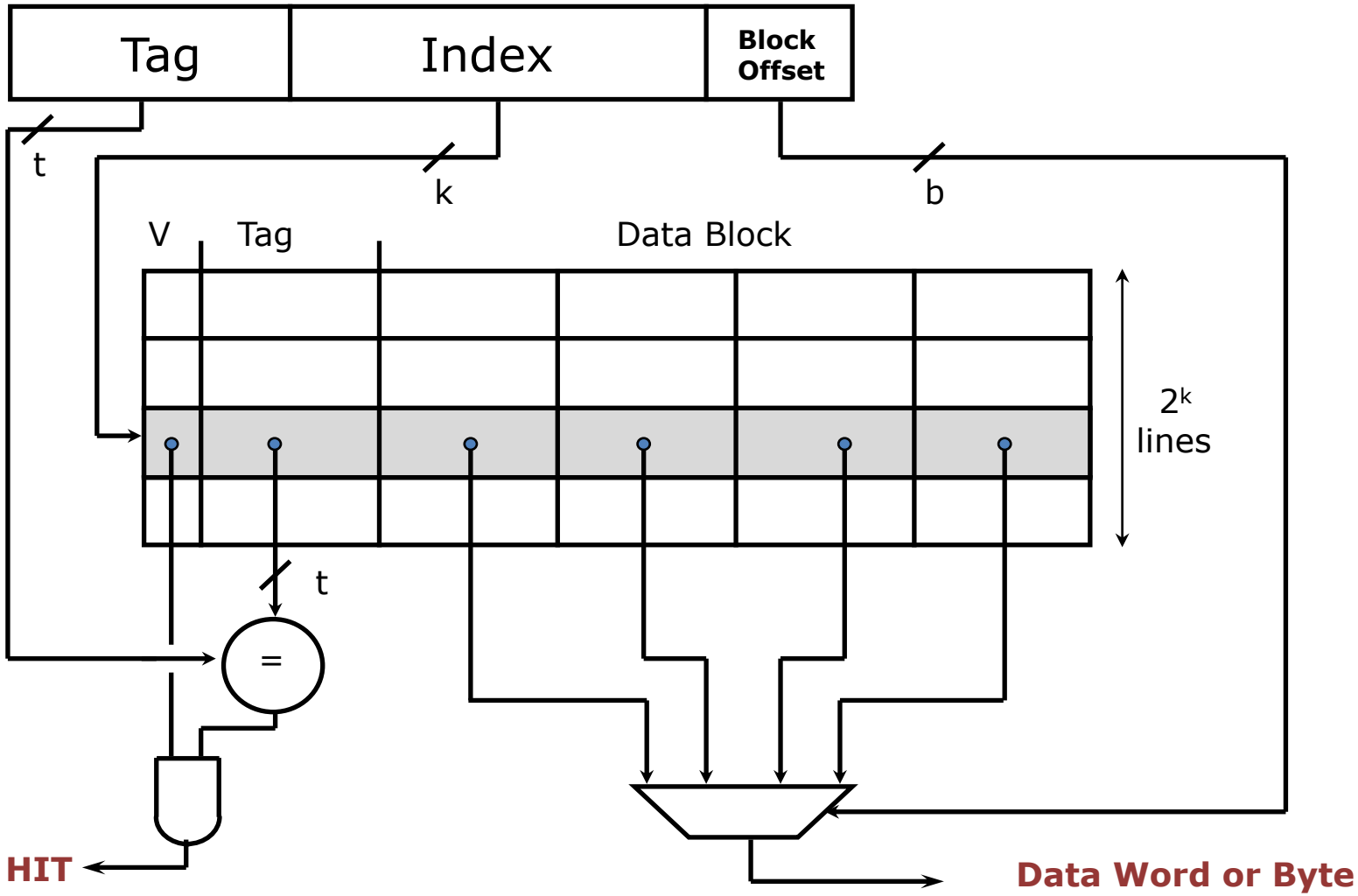
Placement Policy



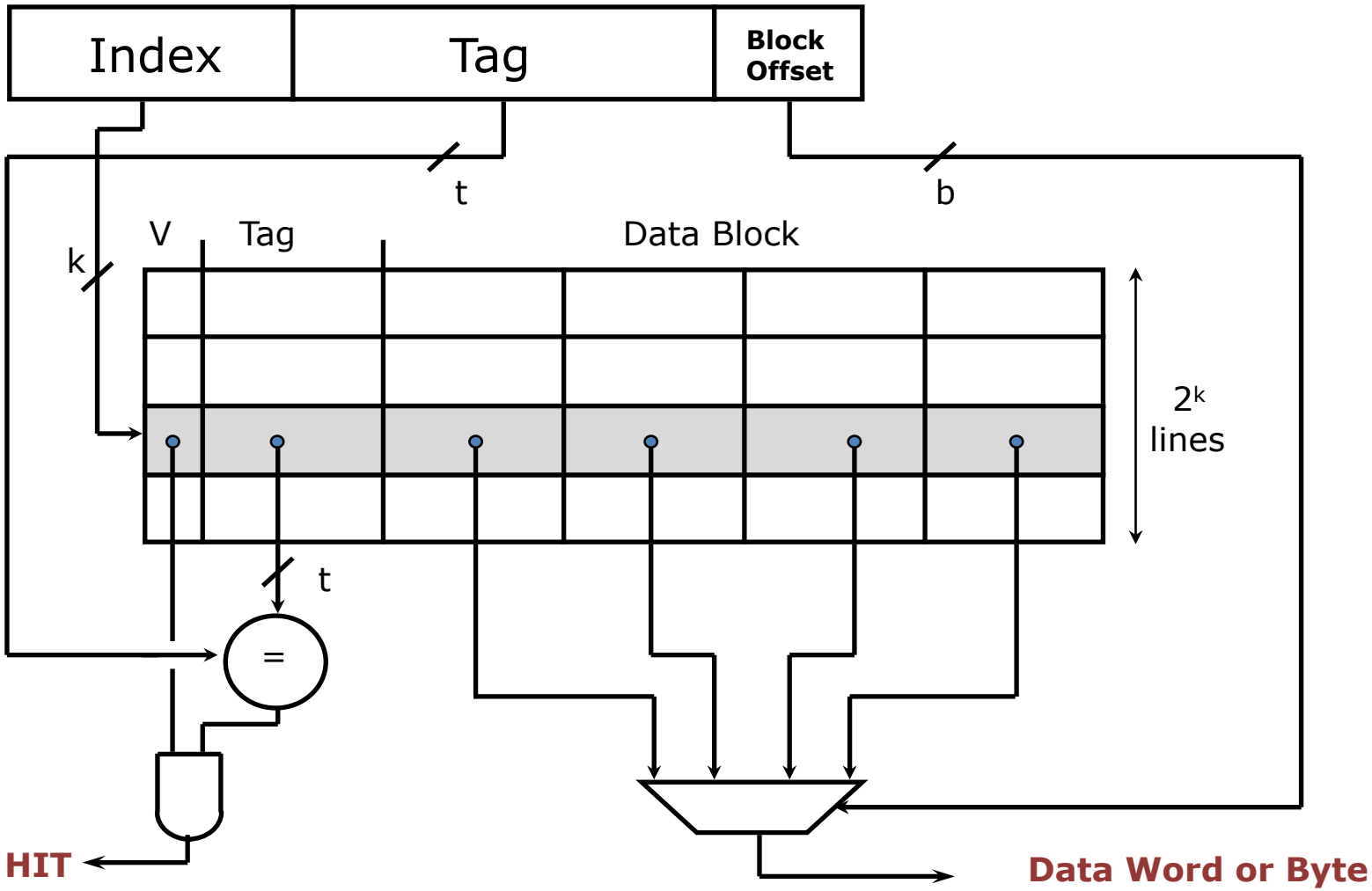
Placement Policy



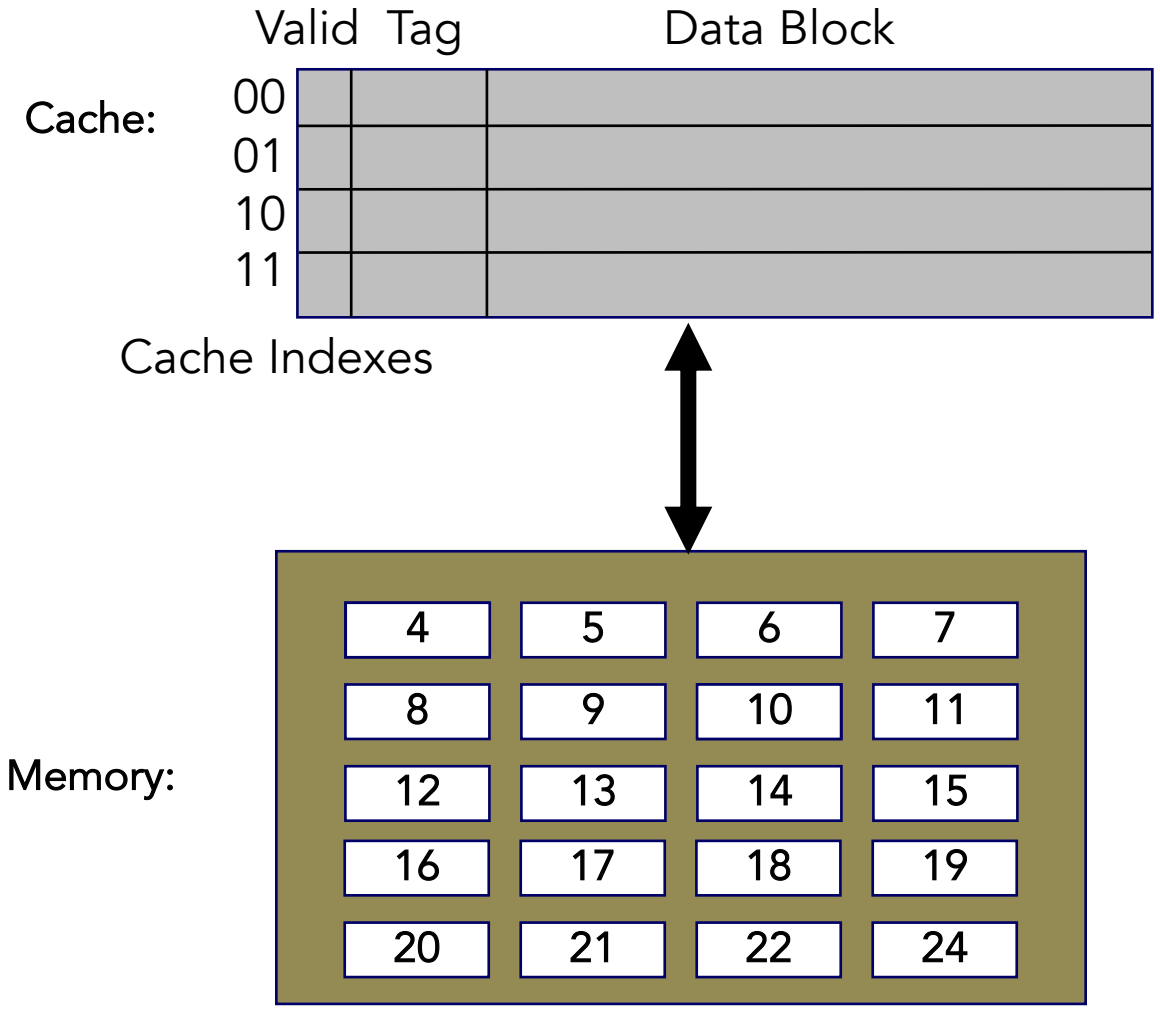
Direct-Mapped Cache



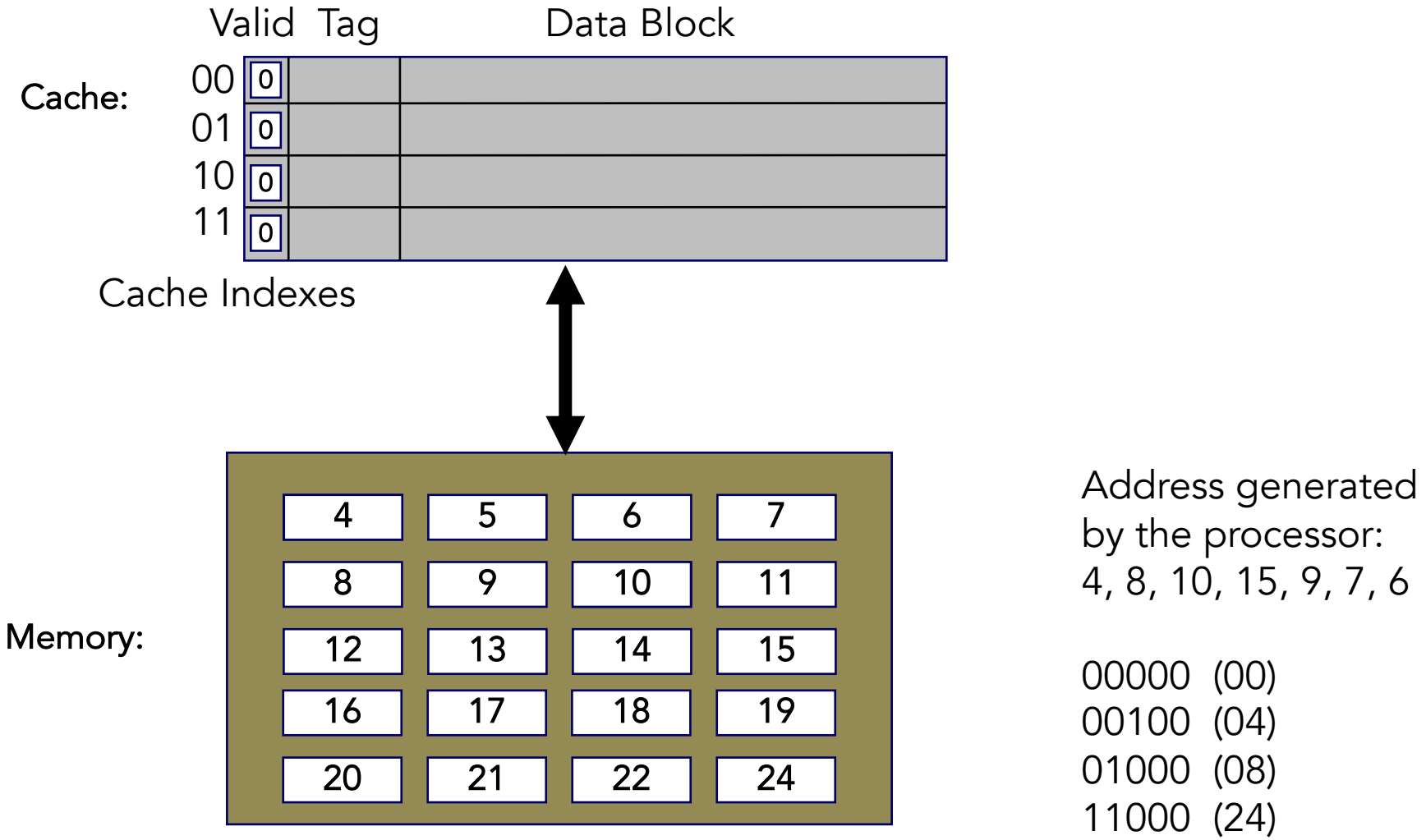
Direct Map Address Selection



Caching Mechanism



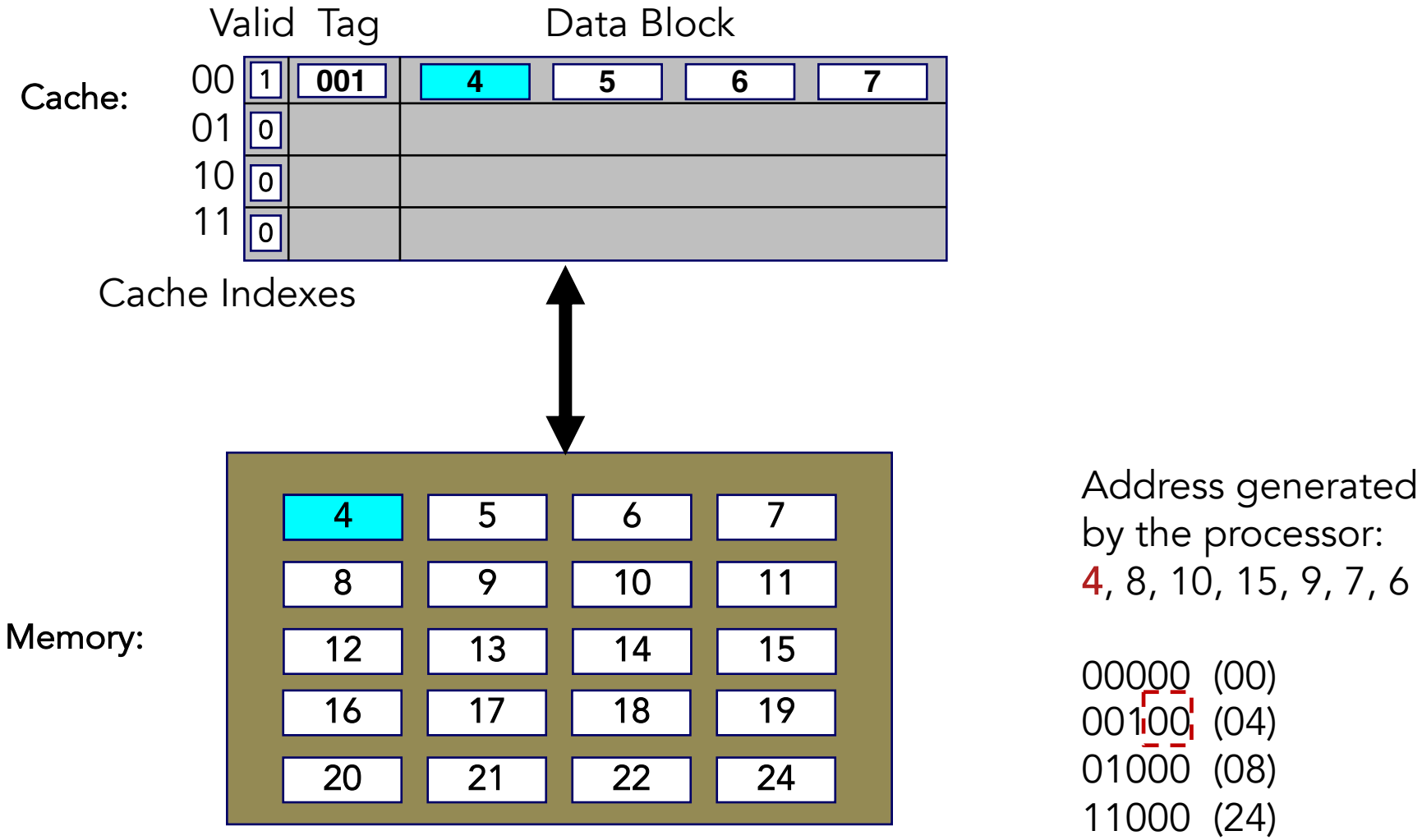
Caching Mechanism



Address generated by the processor:
4, 8, 10, 15, 9, 7, 6

00000 (00)
00100 (04)
01000 (08)
11000 (24)

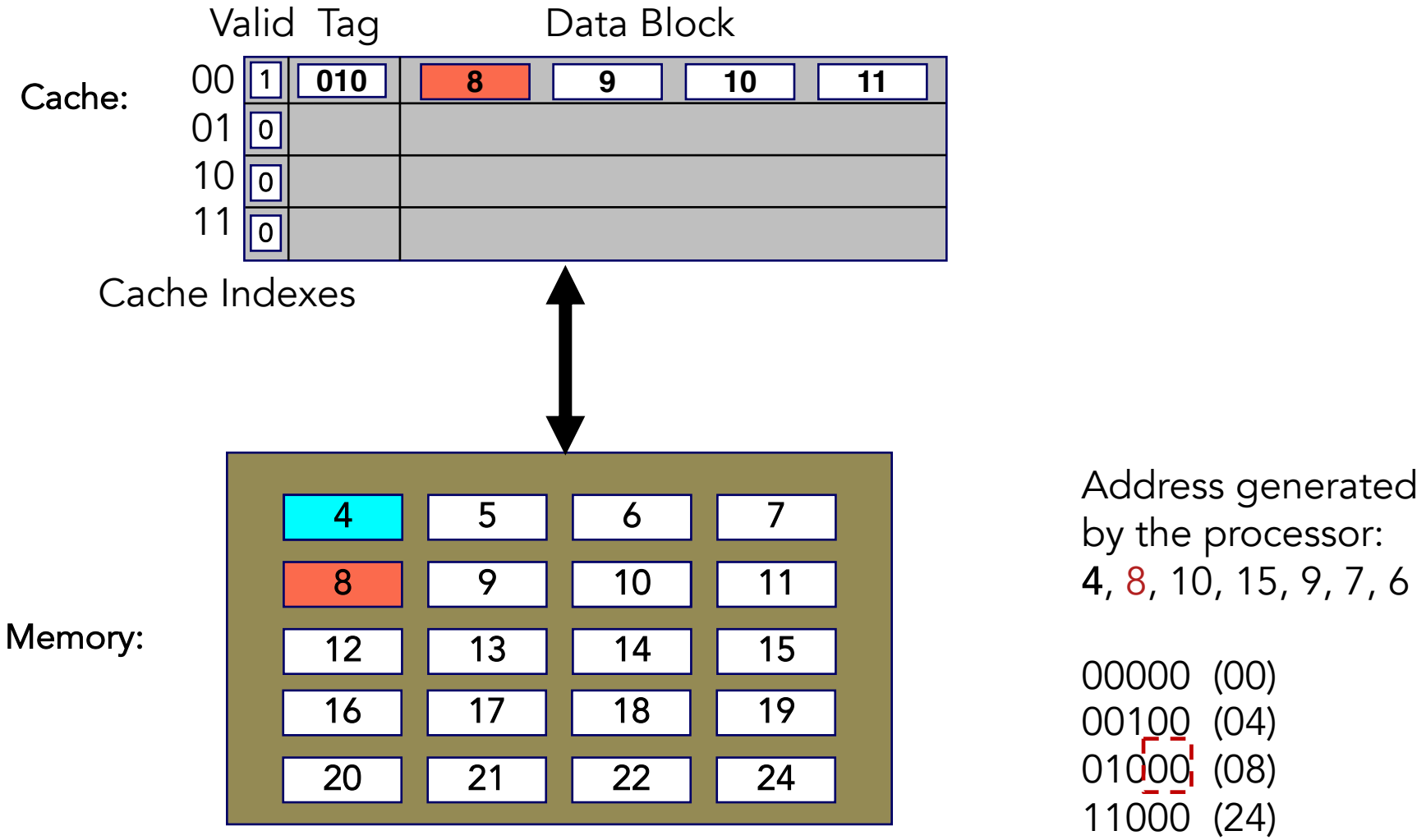
Caching Mechanism



Address generated by the processor:
4, 8, 10, 15, 9, 7, 6

00000 (00)
00100 (04)
01000 (08)
11000 (24)

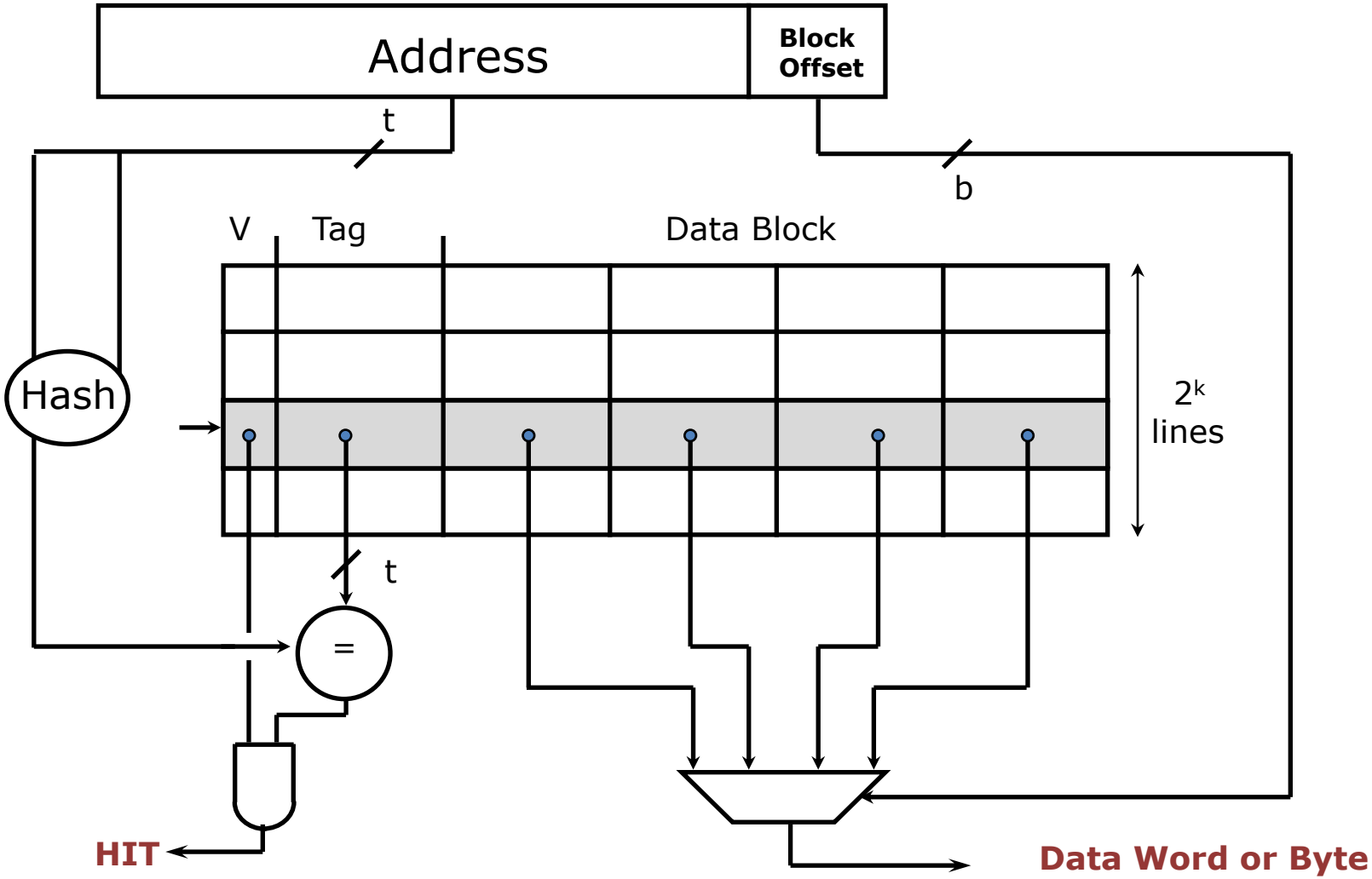
Caching Mechanism



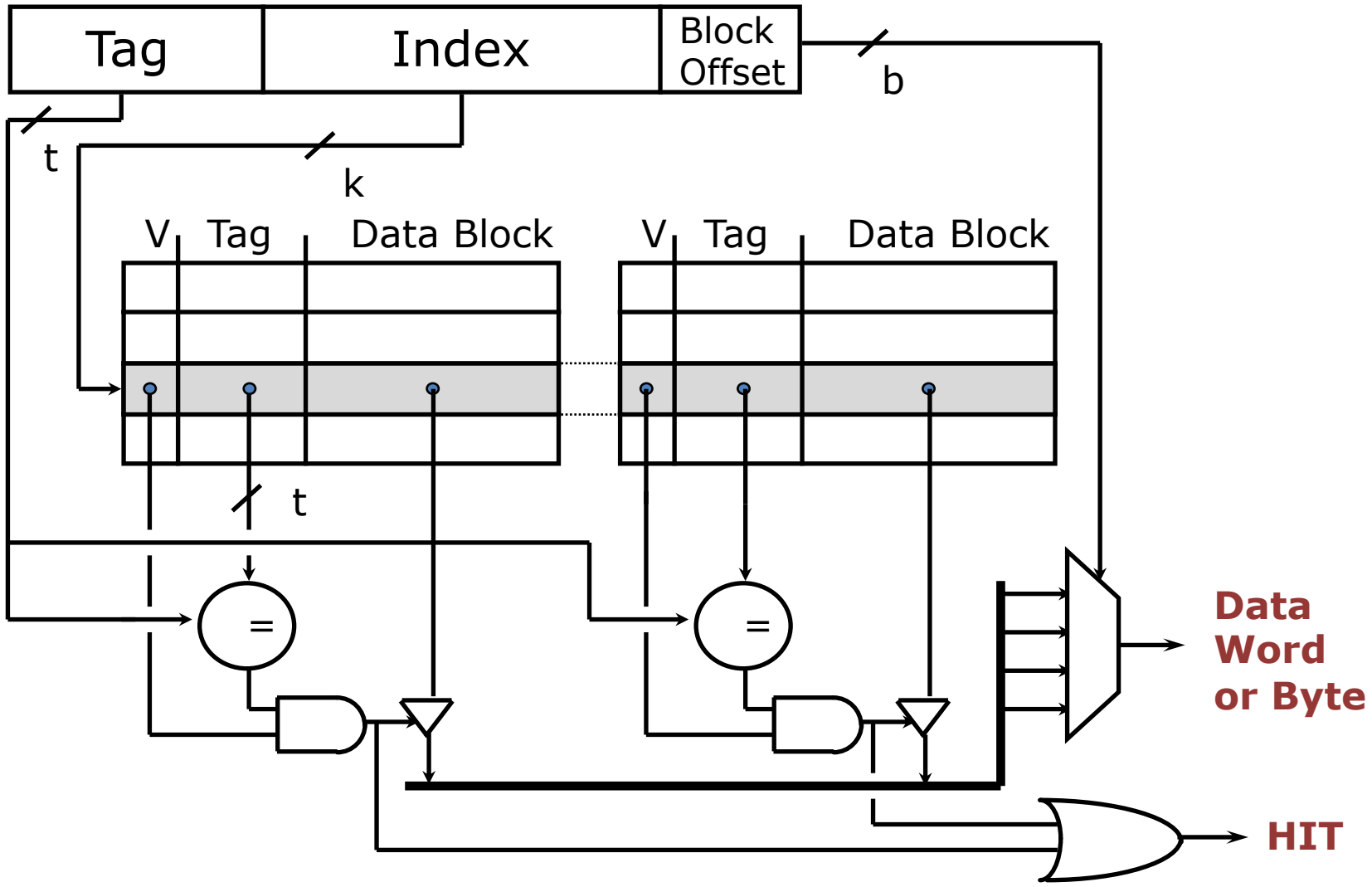
Address generated by the processor:
4, 8, 10, 15, 9, 7, 6

00000 (00)
00100 (04)
01000 (08)
11000 (24)

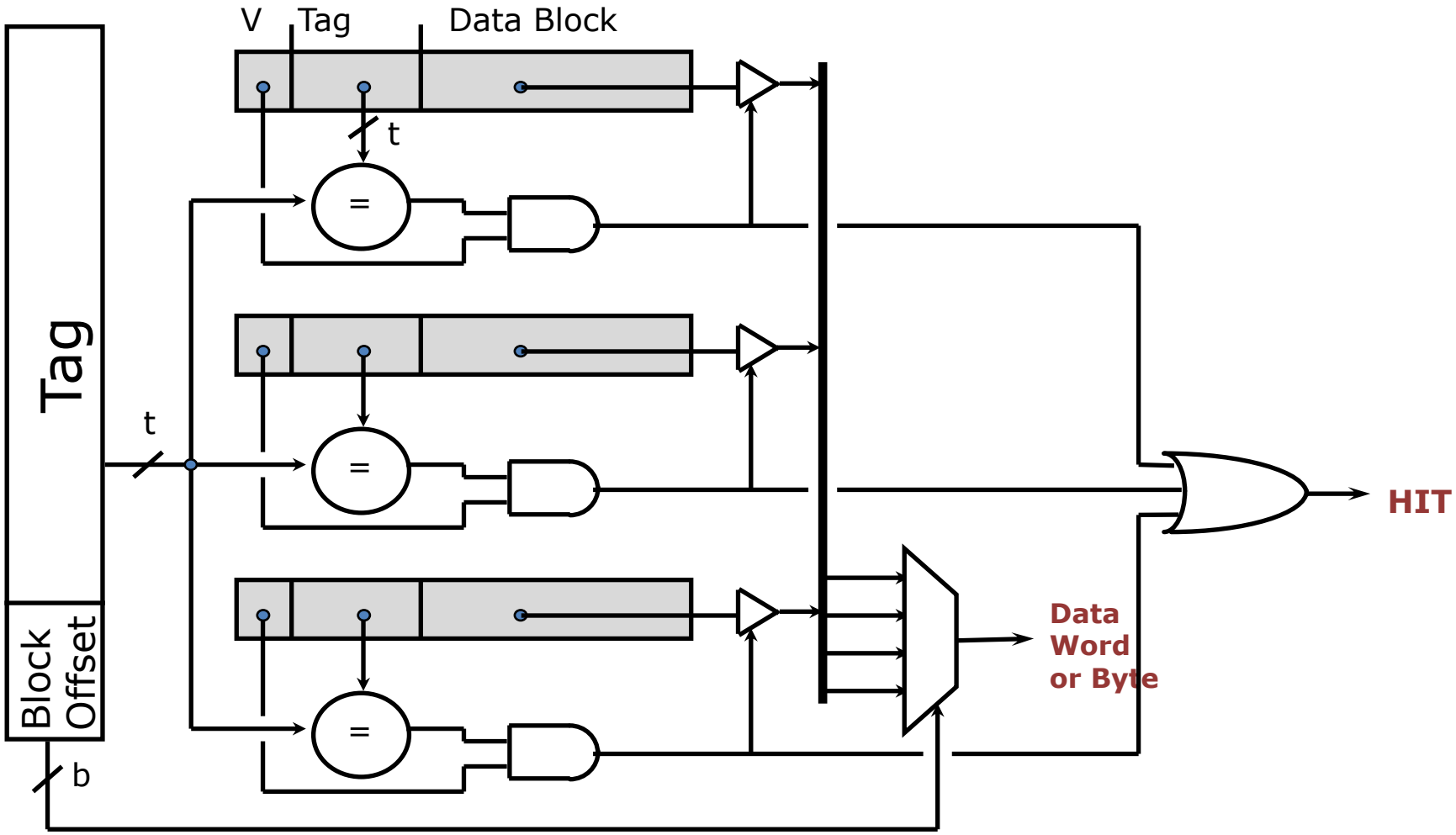
Hashed Address Selection



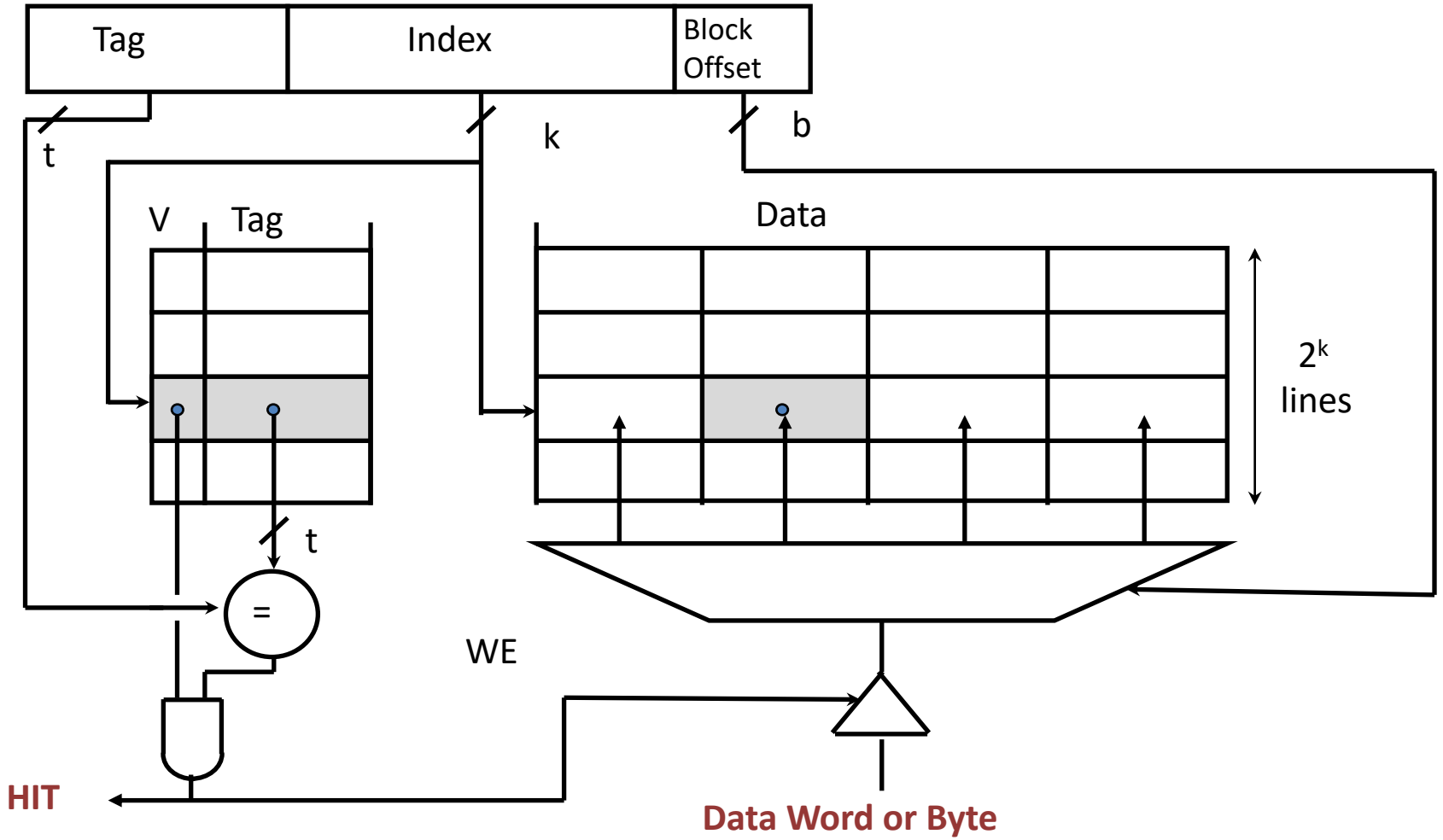
2-Way Set-Associative Cache



Fully Associative Cache

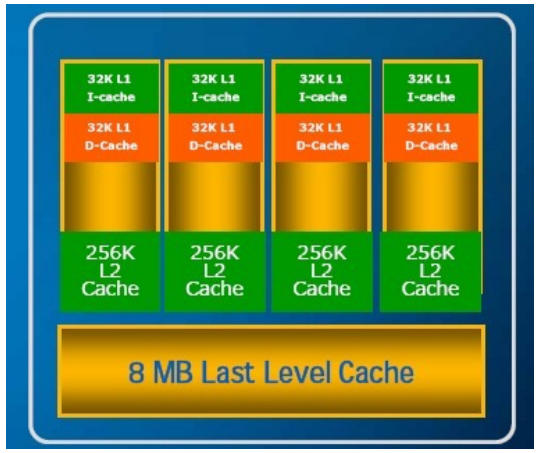


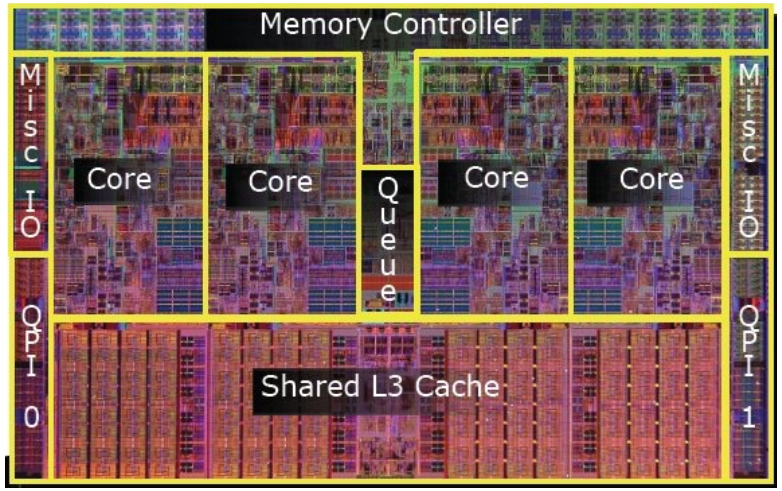
Write Performance



System Example

- Intel Core i7 processors
  - 20 to 24 pipeline stages
  - Performance
    - Max. CPU clock rate 1.06 GHz to 3.33 GHz
  - Cache
    - L1 cache 64 KB per core
    - L2 cache 256 KB per core
    - L3 cache 4 MB to 24 MB shared





Intel Nehalem

System Example

- Intel Core i7 processors
  - 20 to 24 pipeline stages
  - Performance
    - Max. CPU clock rate 1.06 GHz to 3.33 GHz
  - Cache

Level	Capacity	Associativity (ways)	Line Size (bytes)	Access Latency (clocks)	Access Throughput (clocks)	Write Update Policy
First Level Data	32 KB	8	64	4	1	Writeback
Instruction	32 KB	4	N/A	N/A	N/A	N/A
Second Level	256KB	8	64	101	Varies	Writeback
Third Level (Shared L3)[2]	8MB	16	64	35-40^{+2}	Varies	Writeback

Caching Example

```
addi   x2, x0, 413
lw     x4, 0(x2)
```

Index	V	Tag	Data
…			
1101	0	01 0010	700
…			

Address	Data
…	…
00 1001 1101	311
…	…
01 1001 1101	513
…	…

Caching Example

```
addi   x2, x0, 413
lw     x4, 0(x2)
```

Index	V	Tag	Data
…			
1101	0	01 0010	700
…			

Address	Data
…	…
00 1001 1101	311
…	…
01 1001 1101	513
…	…

1. Mem[413] = Mem[01 1001 1101] = 513
2. V = 0 → Miss

Caching Example

```
addi   x2, x0, 413
lw     x4, 0(x2)
```

Index	V	Tag	Data
…			
1101	1	00 1001	311
…			

Address	Data
…	…
00 1001 1101	311
…	…
01 1001 1101	513
…	…

1. Mem[413] = Mem[01 1001 1101] = 513
2. V = 1 → Hit?
3. Tag mismatch 01 1001 ≠ 00 1001

Caching Example

```
addi   x2, x0, 413
lw     x4, 0(x2)
```

Index	V	Tag	Data
…			
1101	1	01 1001	513
…			

Address	Data
…	…
00 1001 1101	311
…	…
01 1001 1101	513
…	…

1. Mem[413] = Mem[01 1001 1101] = 513
2. V = 1 → Hit?
3. Tag mismatch 01 1001 = 01 1001 → Hit

Caching Principles

- Cache contains copies of some of Main Memory
  - Those storage locations recently used
    - When Main Memory address A is referenced in CPU
    - Cache checked for a copy of contents of A
  - If found, cache hit
    - Copy used
    - No need to access Main Memory
  - If not found, cache miss
    - Main Memory accessed to get contents of A
    - Copy of contents also loaded into cache

Cache Performance Metrics

- Cache miss rate
  - Number of cache misses divided by number of accesses
- Cache hit time
  - Time between sending address and data returning from cache
- Cache miss latency
  - Time between sending address and data returning from next-level cache/memory
- Cache miss penalty
  - Extra processor stall caused by next-level cache/memory access

Average Memory Access Time

- Average Memory Access Time (AMAT)
  - AMAT = Hit time + (Miss rate x Miss penalty)
  - Memory stall cycles = Memory accesses x miss rate x miss penalty
  - CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time
  - *CPI = ideal CPI + average stalls per instruction*

Average Memory Access Time

- Average Memory Access Time (AMAT)
  - AMAT = Hit time + (Miss rate x Miss penalty)
  - Memory stall cycles = Memory accesses x miss rate x miss penalty
  - CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time
  - *CPI = ideal CPI + average stalls per instruction*

The cache hit ratio is 96% and the hit time is 1 cycle, but the miss penalty is 20 cycles.

Average Memory Access Time

- Average Memory Access Time (AMAT)
  - AMAT = Hit time + (Miss rate x Miss penalty)
  - Memory stall cycles = Memory accesses x miss rate x miss penalty
  - CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time
  - *CPI = ideal CPI + average stalls per instruction*

The cache hit ratio is 96% and the hit time is 1 cycle, but the miss penalty is 20 cycles.

AMAT = 1 + (0.04 * 20) = 1.8 cycles

Comparative Study

- Increasing block size can improve hit rate due to spatial locality, but transfer time increases
- Assume we have two cache structures with the following specifications
  - Single cycle hit times
  - Memory accesses take 15 cycles
  - Memory bus is 8-bytes wide
    - For example a 16-byte memory access takes 18 cycles
      - 1 (send address) + 15 (memory access) + 2 (two 8-byte transfers)
  - Which of the two cache configuration is better?

	Cache configuration 1	Cache configuration 2
Block size	32-bytes	64-bytes
Miss rate	5%	4%

Comparative Study

- Assume we have two cache structures with the following specifications
  - Single cycle hit times
  - Memory accesses take 15 cycles
  - Memory bus is 8-bytes wide
  - Which of the two-cache configuration is better?

	Cache configuration 1	Cache configuration 2
Block size	32-bytes	64-bytes
Miss rate	5%	4%

Cache configuration 1
Miss Penalty = 1 + 15 + 32B/8B = 20 cycles
AMAT = 1 + (.05 * 20) = 2

Comparative Study

- Assume we have two cache structures with the following specifications
  - Single cycle hit times
  - Memory accesses take 15 cycles
  - Memory bus is 8-bytes wide
  - Which of the two cache configuration is better?

	Cache configuration 1	Cache configuration 2
Block size	32-bytes	64-bytes
Miss rate	5%	4%

Cache configuration 2
Miss Penalty = 1 + 15 + 64B/8B = 24 cycles
AMAT = 1 + (.04 * 24) = 1.96

Memory Impact on Performance

- Example: cold cache, 4-byte words, 4-word cache blocks

```
int sum_by_rows(int array[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += array[i][j];
    return sum;
}
```

- Miss rate = 1/4 = 25%

Memory Impact on Performance

- Example: cold cache, 4-byte words, 4-word cache blocks

```
int sum_by_columns(int array[M][N])
{
      int i, j, sum = 0;

      for (j = 0; j < N; j++)
            for (i = 0; i < M; i++)
                  sum += array[i][j];
      return sum;
}
```

- Miss rate = 100%*

* Assuming that array does not fit into memory

Multi-level Cache

- Having L1 and L2 Caches
- AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$
  - Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$
- AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ × Miss Penalty$_{L2}$)

Multi-level Cache

- Assumptions:
  - L1 hit time = 1 cycle, L1 hit rate = 90%
  - L2 hit time = 8 cycles,
    L2 miss penalty= 100 cycles,
    L2 hit rate = 90%

Multi-level Cache

- Assumptions:
  - L1 hit time = 1 cycle, L1 hit rate = 90%
  - L2 hit time = 8 cycles,
    L2 miss penalty= 100 cycles,
    L2 hit rate = 90%
- Access time = L1 hit time + (L2 hit time + L2 miss penalty * (1 - L2 hit rate)) * L1 miss rate

  = 1 + ((8 + 100*0.1) *(1-0.9))

  = 1 + (18* 0.1) = 2.8 clock cycles

Multi-level Cache

- A processor with a base CPI of 1.0 assuming all memory references hit in the L1 cache and a clock rate of 1 GHz. The main memory access time is 100 ns. Suppose the miss rate per instruction is 5%
  - What is the effective CPI?
  - How much faster will the processor run if it has a secondary cache (with 20-ns access time) that reduces the miss rate to 3%?

Multi-level Cache

- A processor with a base CPI of 1.0 assuming all memory references hit in the L1 cache and a clock rate of 1 GHz. The main memory access time is 100 ns. Suppose the miss rate per instruction is 5%
  - What is the effective CPI?
    - Miss penalty to main memory = 100 ns = 100 cycles.
    - Total CPI = Base CPI + Memory-stall cycles per instruction
    - CPI = 1.0 + 5% x 100 = 6.0

Multi-level Cache

- A processor with a base CPI of 1.0 assuming all memory references hit in the L1 cache and a clock rate of 1 GHz. The main memory access time is 200 ns. Suppose the miss rate per instruction is 5%
  - How much faster will the processor run if it has a secondary cache (with 20-ns access time) that reduces the miss rate to 3%?
    - Access time = L1 hit time + (L2 hit time + L2 miss penalty * (1 - L2 hit rate))* L1 miss rate

Improving Cache Performance

- Average memory access time =

  Hit time + Miss rate x Miss penalty

- To improve performance:
  - Reduce the hit time
  - Reduce the miss rate (e.g., larger cache)
  - Reduce the miss penalty (e.g., L2 cache)
- What is the simplest design strategy?
  - Biggest cache that doesn't increase hit time past 1-2 cycles (approx 8-32KB in modern technology)

Effect of Cache on Performance

- Larger cache size
  - Reduces conflict misses
  - Hit time will increase
- Higher associativity
  - Reduces conflict misses
  - May increase hit time
- Larger block size
  - Reduces compulsory misses
  - Exploit burst transfers in memory and on buses
  - Increases miss penalty and conflict misses

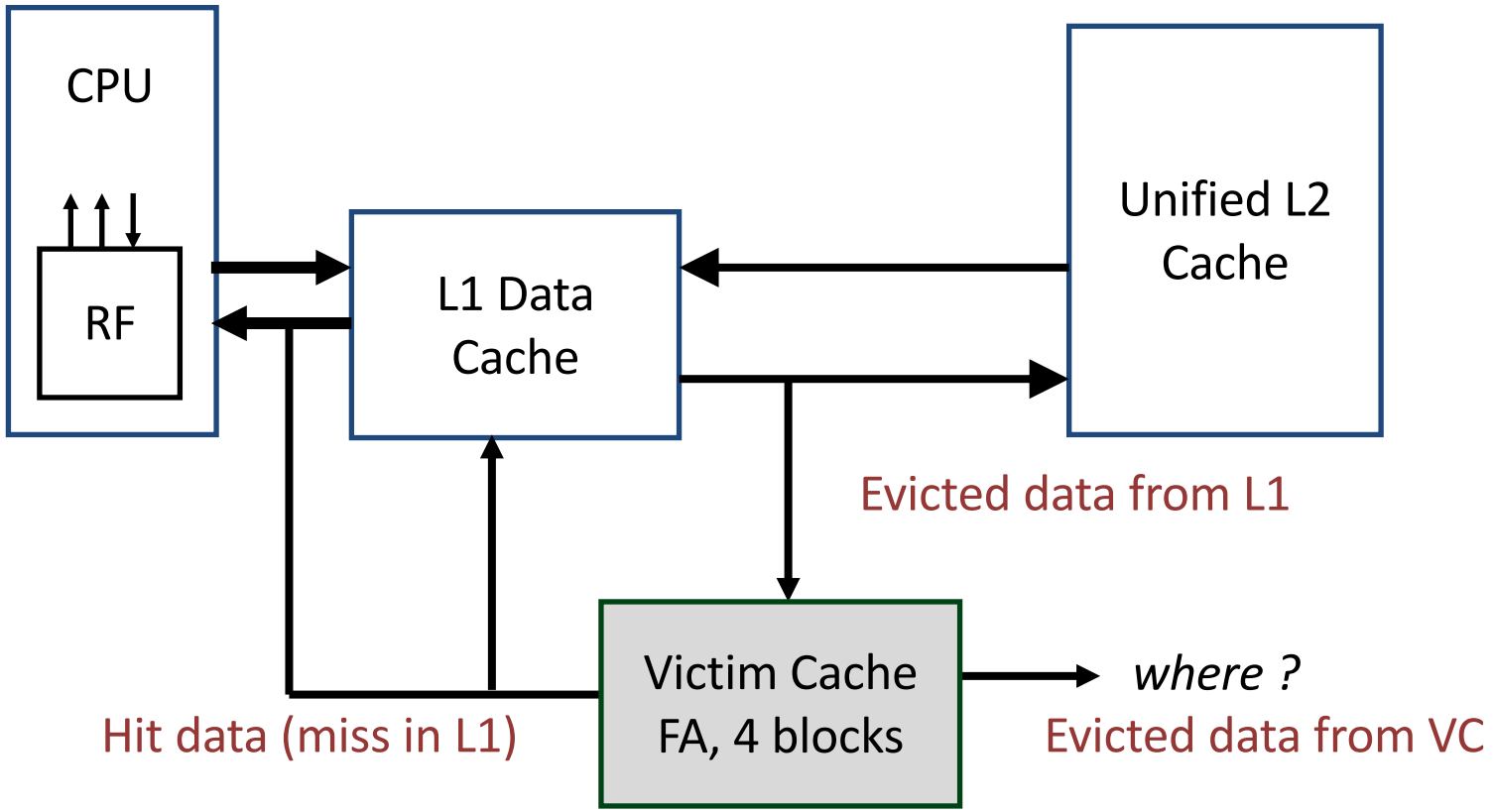
Replacement Policy

- Which block from a set should be evicted?
  - Random
  - Least Recently Used (LRU)
    - LRU cache state must be updated on every access
    - True implementation only feasible for small sets (2-way)
    - Pseudo-LRU binary tree often used for 4-8 way
  - First In, First Out (FIFO) a.k.a. Round-Robin
    - Used in highly associative caches
  - Not Least Recently Used (NLRU)
    - FIFO with exception for most recently used block or blocks

Reducing Write Hit Time

- Problem: Writes take two cycles in memory stage, one cycle for tag check plus one cycle for data write if hit
- Solutions
  - Design data RAM that can perform read and write in one cycle, restore old value after tag miss
  - Fully-associative (CAM Tag) caches: Word line only enabled if hit
  - Pipelined writes: Hold write data for store in single buffer ahead of cache, write cache data during next store's tag check

Victim Caches



- Victim cache is a small associative back up cache, added to a direct

Victim Caches

- Victim cache is a small associative back up cache, added to a direct
  - Mapped cache, which holds recently evicted lines
    1. First look up in direct mapped cache
    2. If miss, look in victim cache
    3. If hit in victim cache, swap hit line with line now evicted from L1
    4. If miss in victim cache, L1 victim -> VC, VC victim->?
  - Fast hit time of direct mapped but with reduced conflict misses

Next Learning Module

- Advanced Memory Operations