CSE251: System Programming

13. Cache (2)

Seongil Wi

Department of Computer Science and Engineering

Where Are We?

Register 1

Register 2

On-chip cache (L1 cache)

SRAM (L2 cache)

Processor (CPU) (e.g., Intel I7)

DRAM (Main memory)

Main memory

Secondary storage (Hard disk)

Today's Topic

How we improve the cache performance?

Register 1
Register 2

On-chip cache (L1 cache)

SRAM (L2 cache)

Processor (CPU) (e.g., Intel I7)

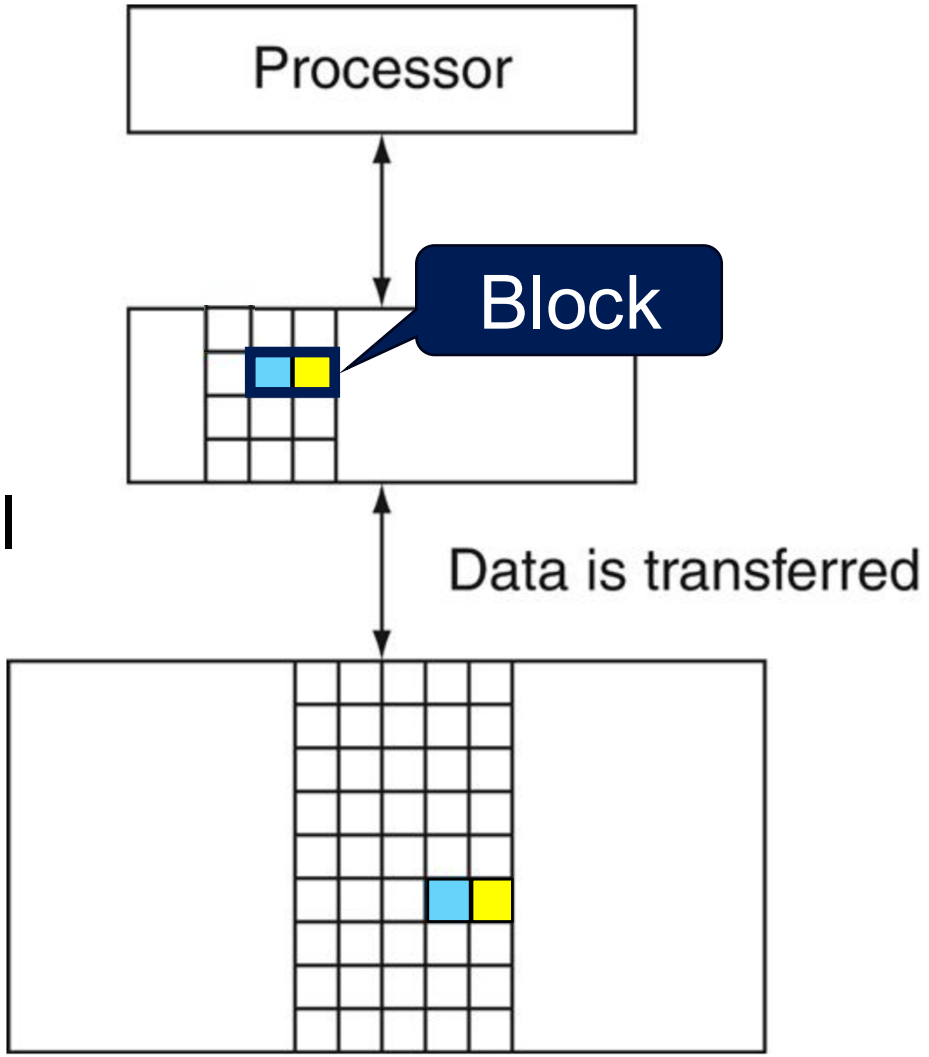
DRAM (Main memory)

Main memory

Secondary storage (Hard disk)

Terms

- Block (a.k.a., line): unit of copying
  - Several words in cache memory
- **Hit**: data requested is in the upper level
  - Hit ratio: hits/accesses
- **Miss**: data requested is not in the upper level
  - Block copied from lower level
  - Miss penalty: time taken to resolve miss
  - Miss ratio: misses/accesses
    = 1 – hit ratio



Improving Cache Performance

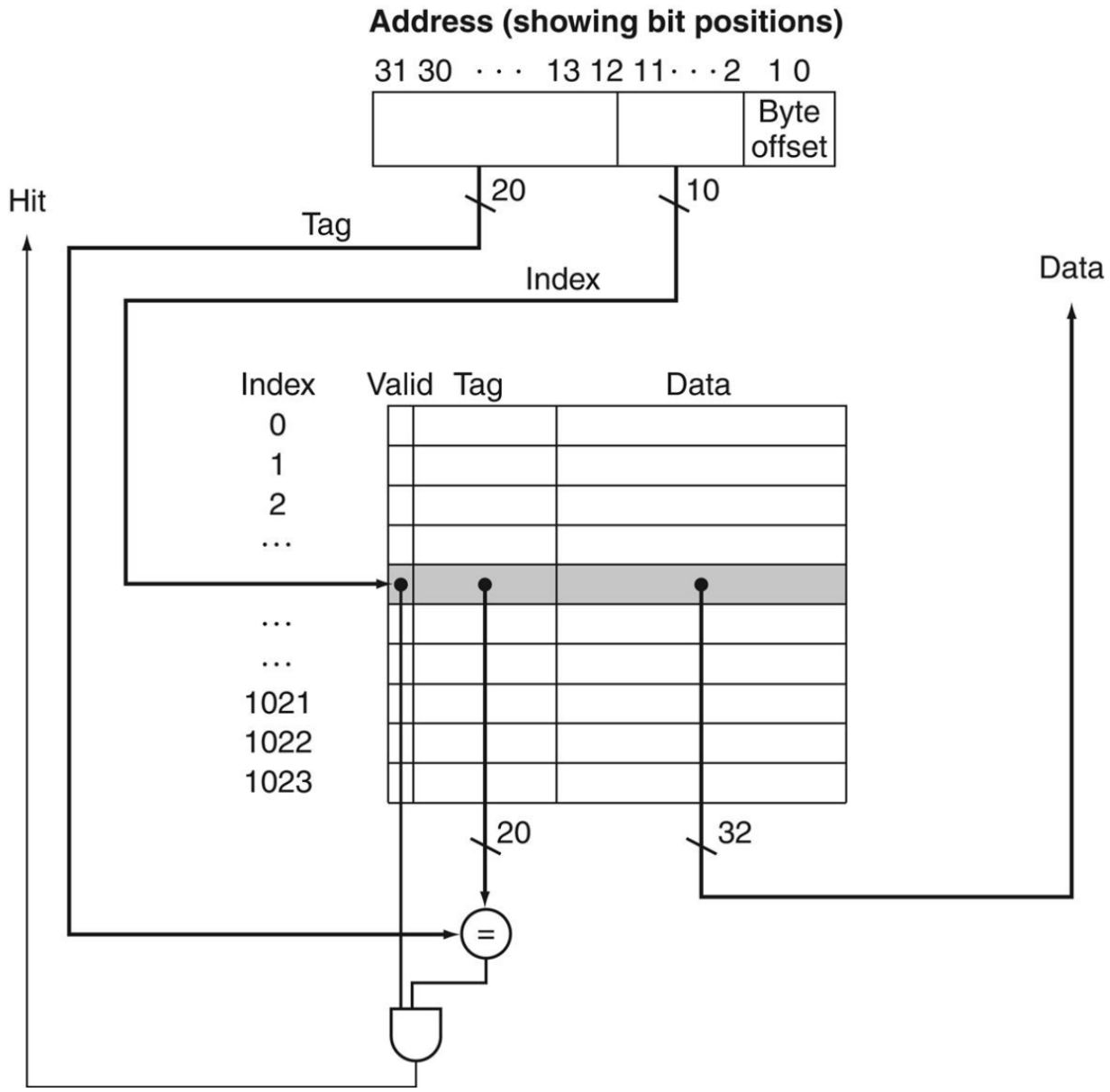
How to Improve the Cache Performance?

- Adjust the block size
- Adjust the cache placement policy
- Adjust the cache replacement policy
- Multilevel caches

How to Improve the Cache Performance?

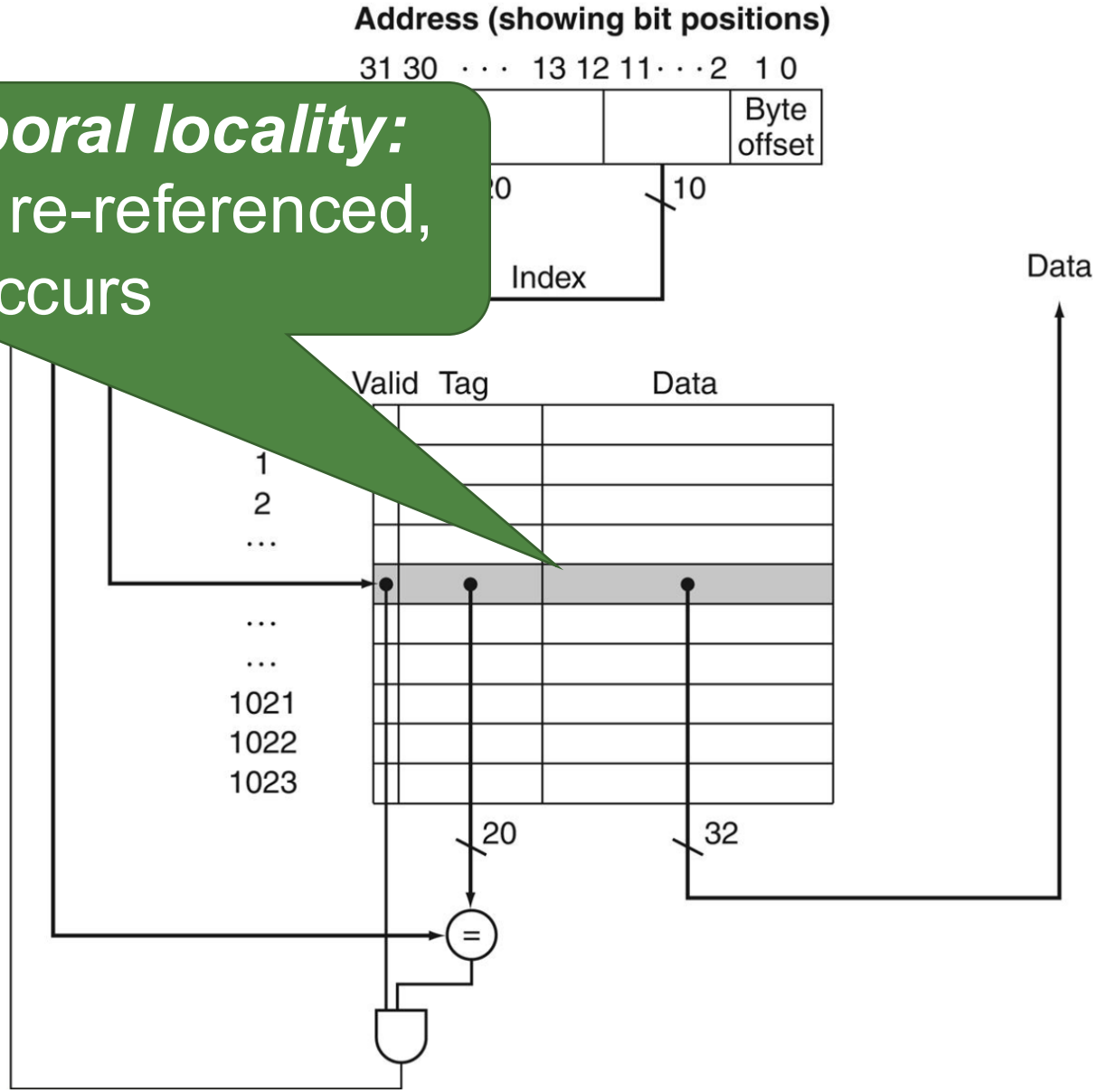
- **Adjust the block size**
- Adjust the cache placement policy
- Adjust the cache replacement policy
- Multilevel caches

Consider about One-word Block



Problem with a One-word Block?

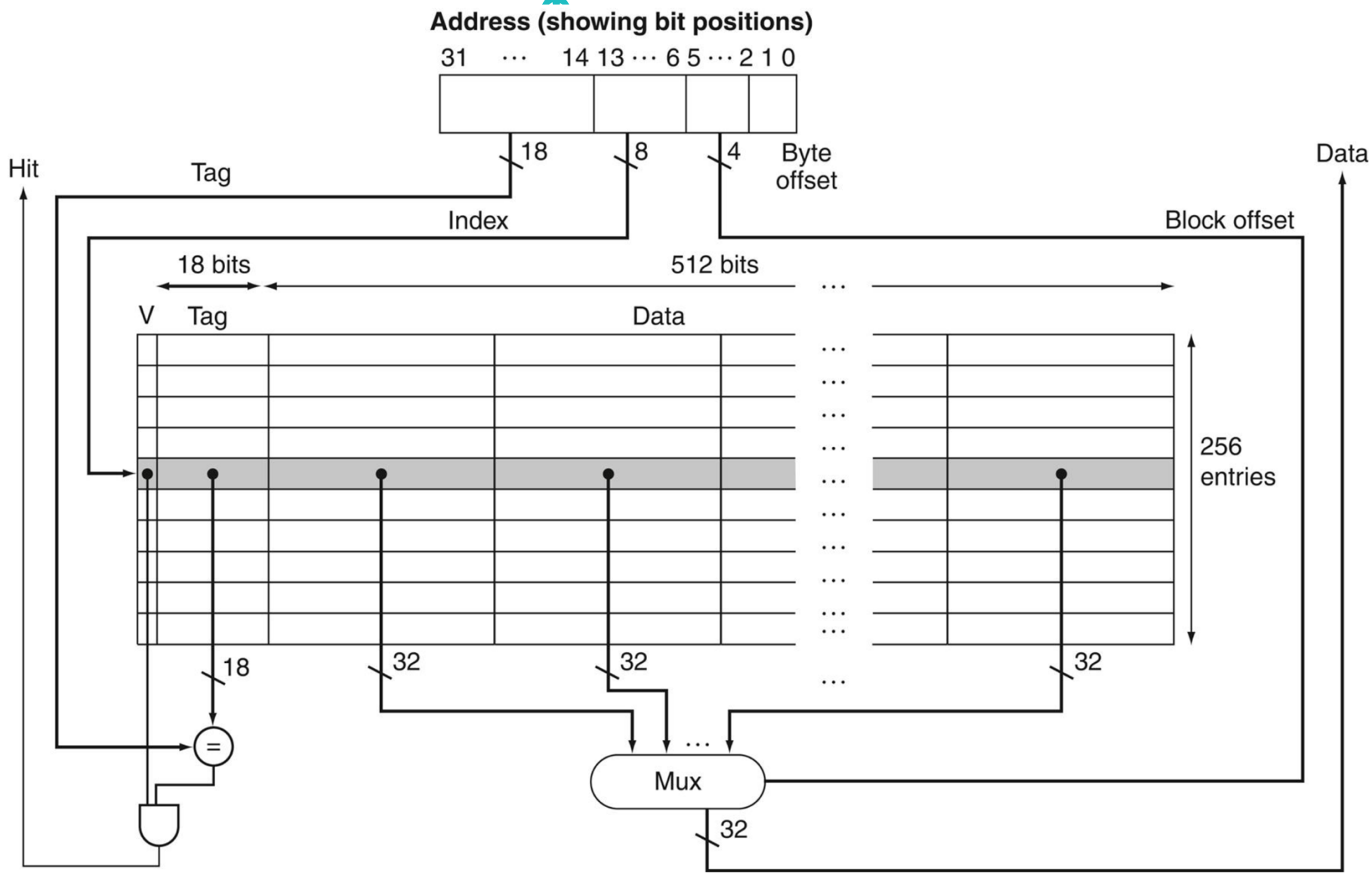
We can exploit temporal locality:
If the same address is re-referenced,
a cache hit occurs



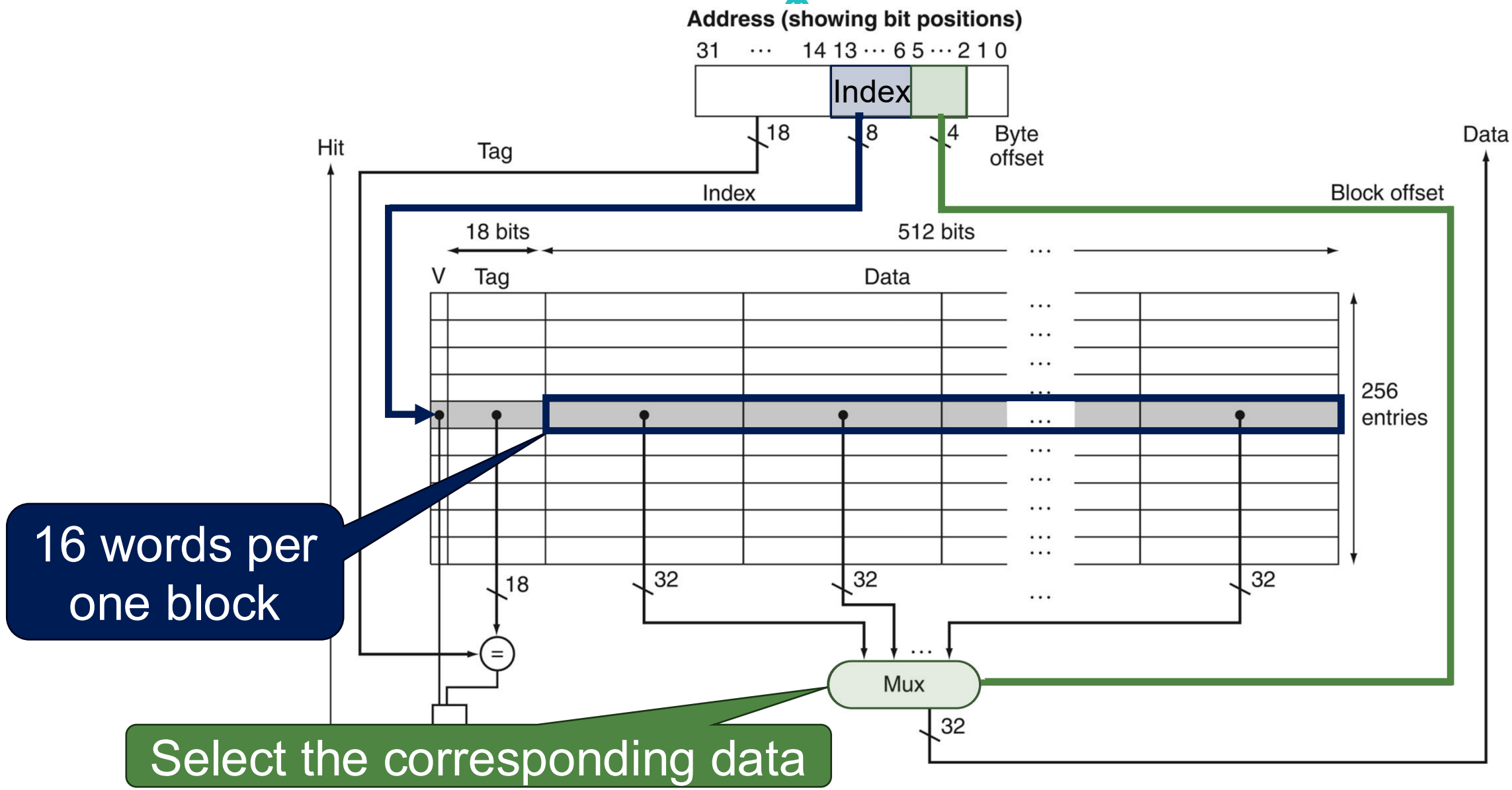
How about spatial locality?

→ Increase the block size!

Several Words as One Block

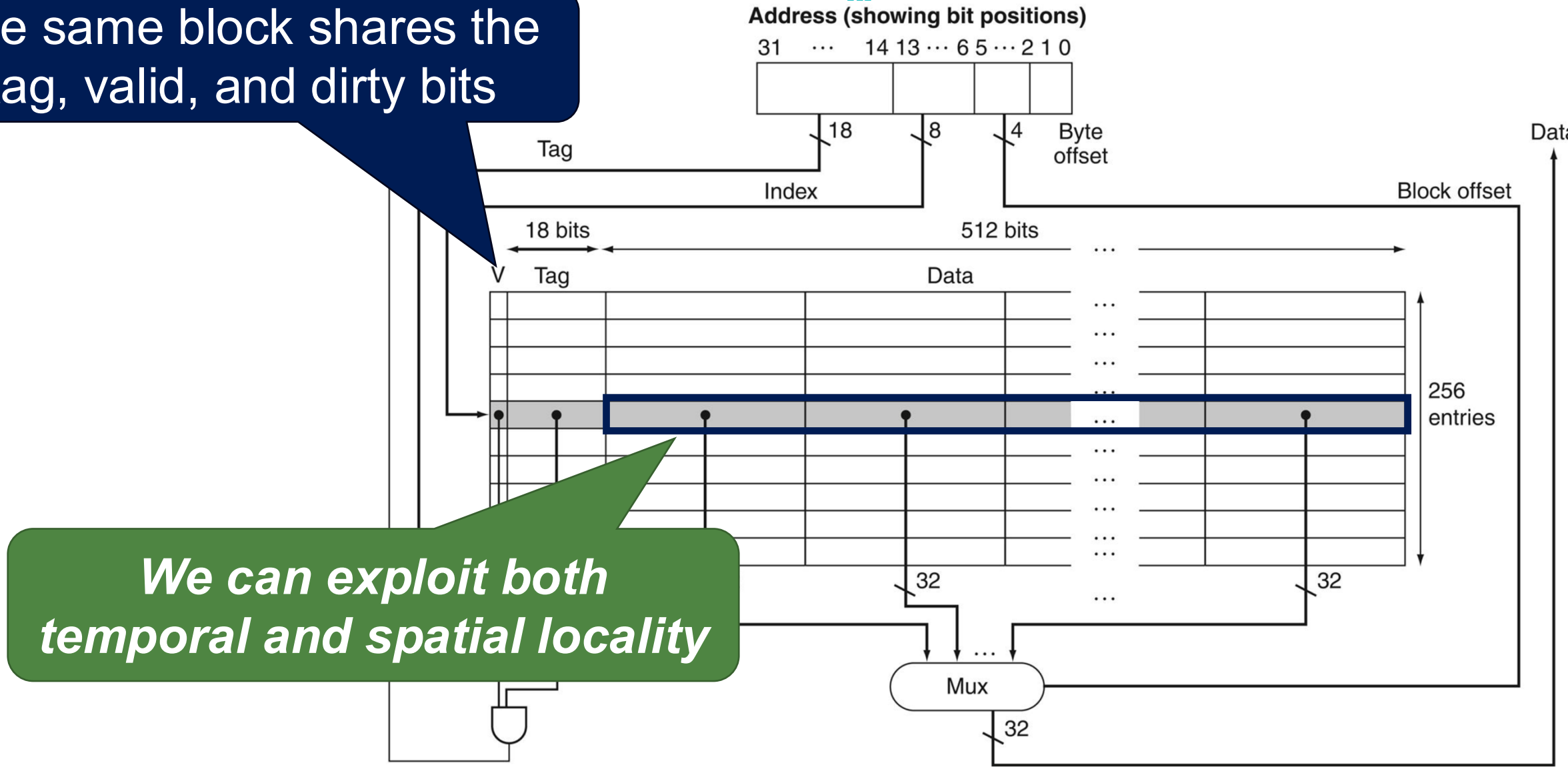


Several Words as One Block



Several Words as One Block

The same block shares the tag, valid, and dirty bits

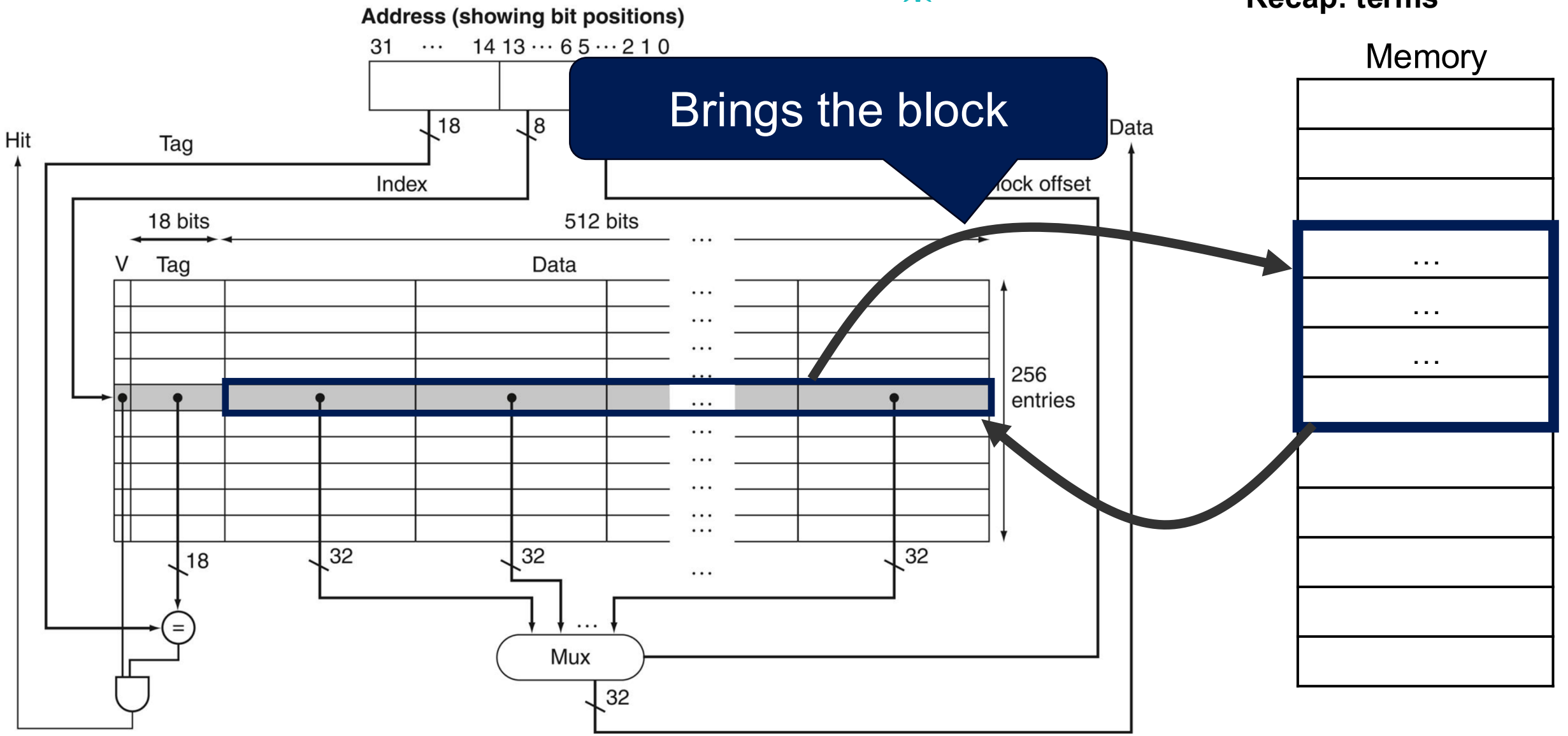


We can exploit both temporal and spatial locality

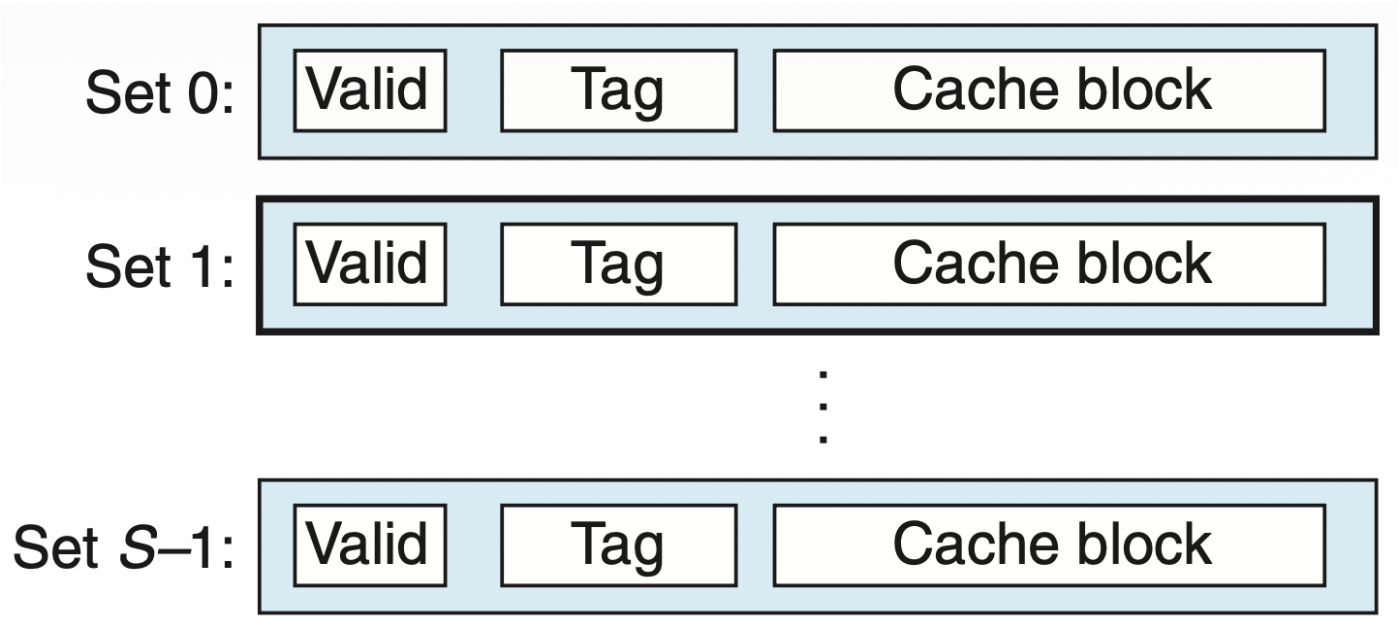
Several Words as One Block

- Block (a.k.a., line): unit of copying
  - Several words in cache memory

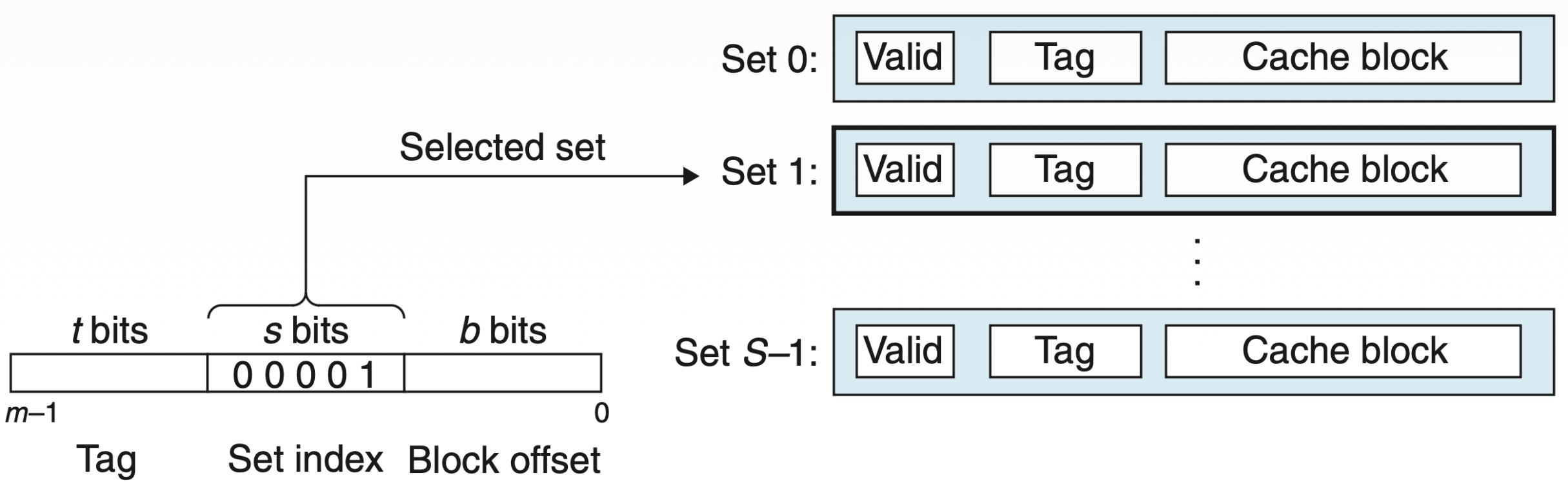
Recap: terms



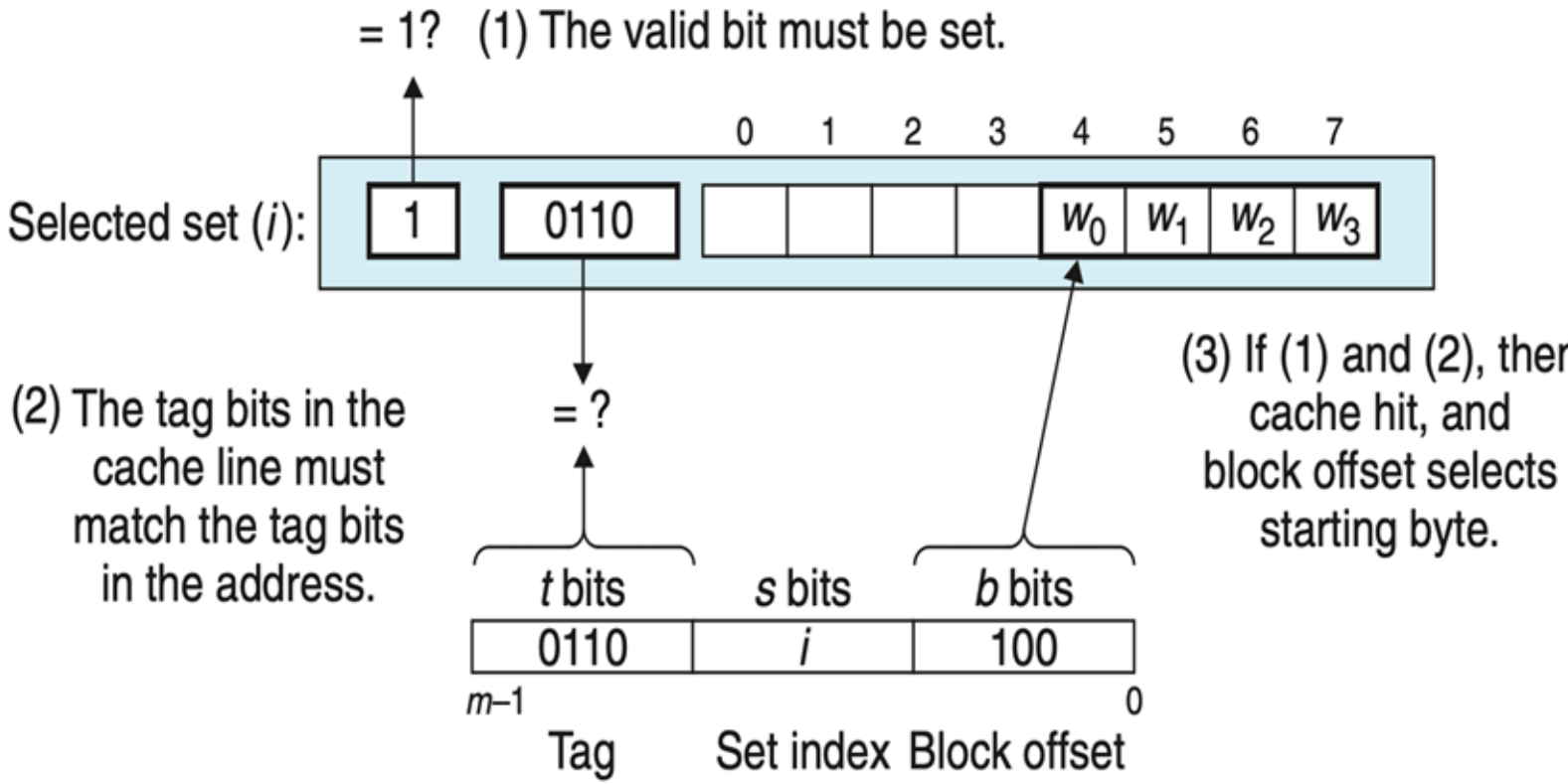
High-level View



High-level View: Block Selection



High-level View: Word Selection



Increasing the Block Size

- **Pros**: tends to decrease miss rate
  - Due to the spatial locality

- **Hit**: data requested is in the upper level
  - Hit ratio: hits/accesses
- **Miss**: data requested is not in the upper level
  - Block copied from lower level
  - Miss penalty: time taken to resolve miss
  - Miss ratio: misses/accesses
    = 1 – hit ratio

Terms

- **Cons**:
  - **The miss penalty increases** because transferring such a large block takes more time
    + α? (Homework4!)

How to Improve the Cache Performance?

- **Adjust the block size**
- Adjust the cache placement policy
- Adjust the cache replacement policy
- Multilevel caches

How to Improve the Cache Performance?

- **Adjust the block size**
- **Adjust the cache placement policy**
- **Adjust the cache replacement policy**
- **Multilevel caches**

Cache Placement Policy

Cache placement policy: where can a block be placed?

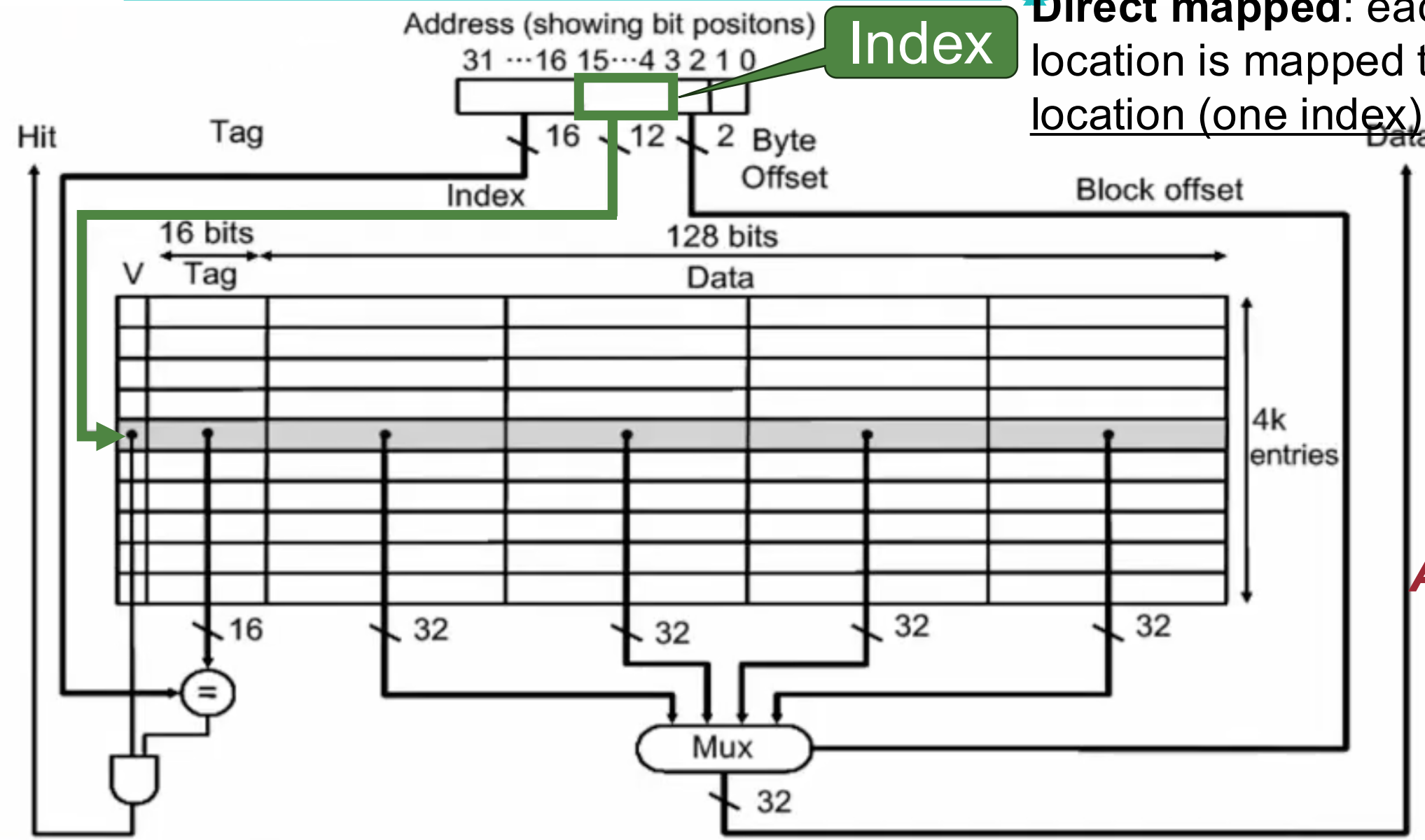
- Direct mapped
- Fully associative
- Set associative

Cache Placement Policy: Direct Mapped

- Address is modulo the number of blocks in the cache

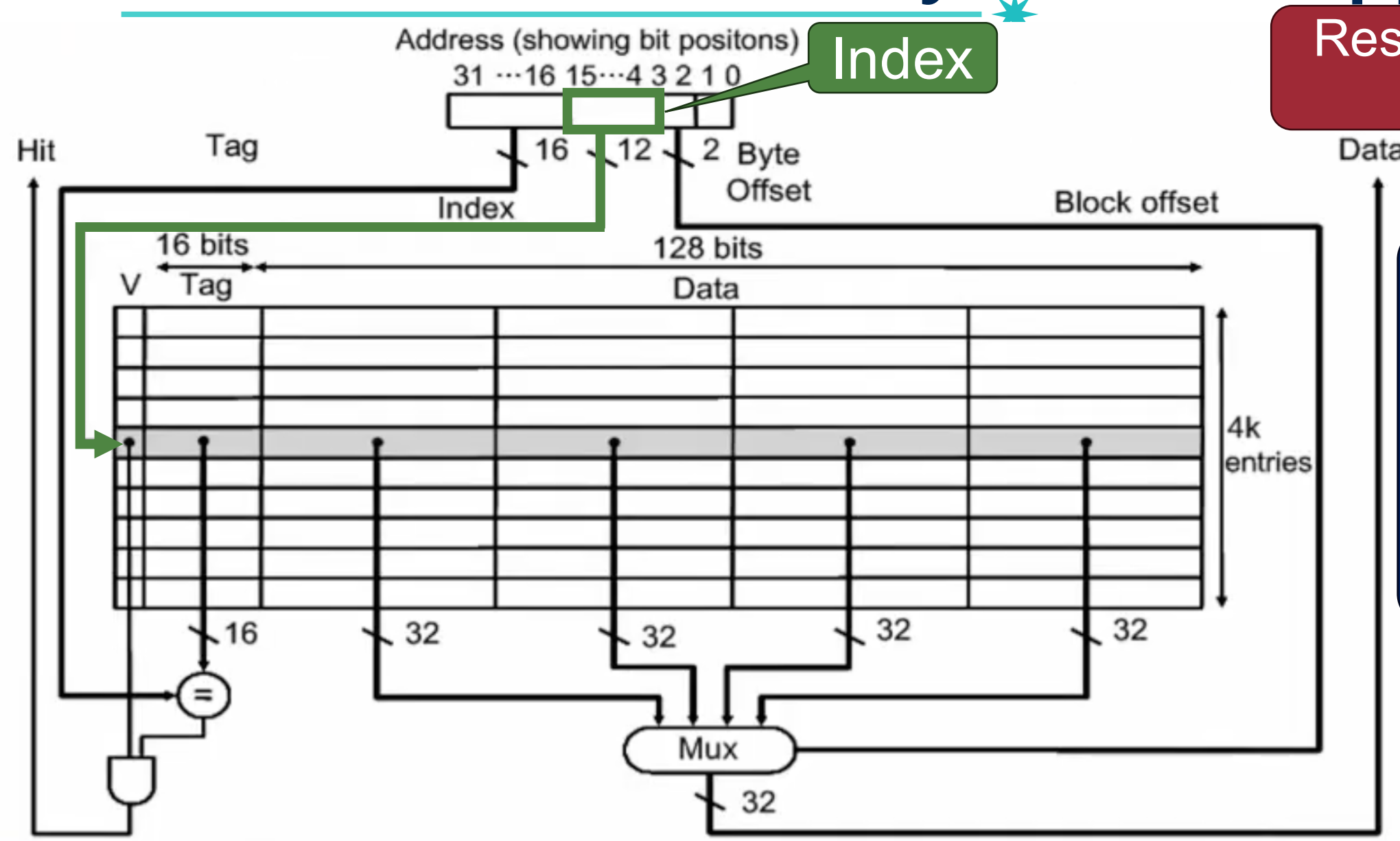
Cache Placement Policy: Direct Mapped

Direct mapped: each *memory* location is mapped to <u>exactly one location (one index)</u> in the *cache*



Any problems?

Cache Placement Policy: Direct Mapped



Results in lots of misses!

```
0x00011010
0x00101010
0x00011010
0x00101010
0x00011010
...
```

Memory access sequence

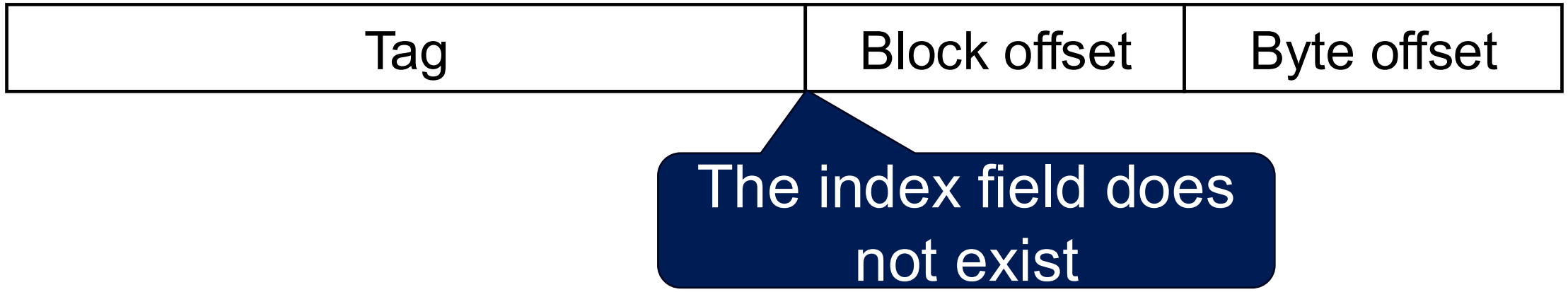
Direct Mapped: Summary

- Address is modulo the number of blocks in the cache

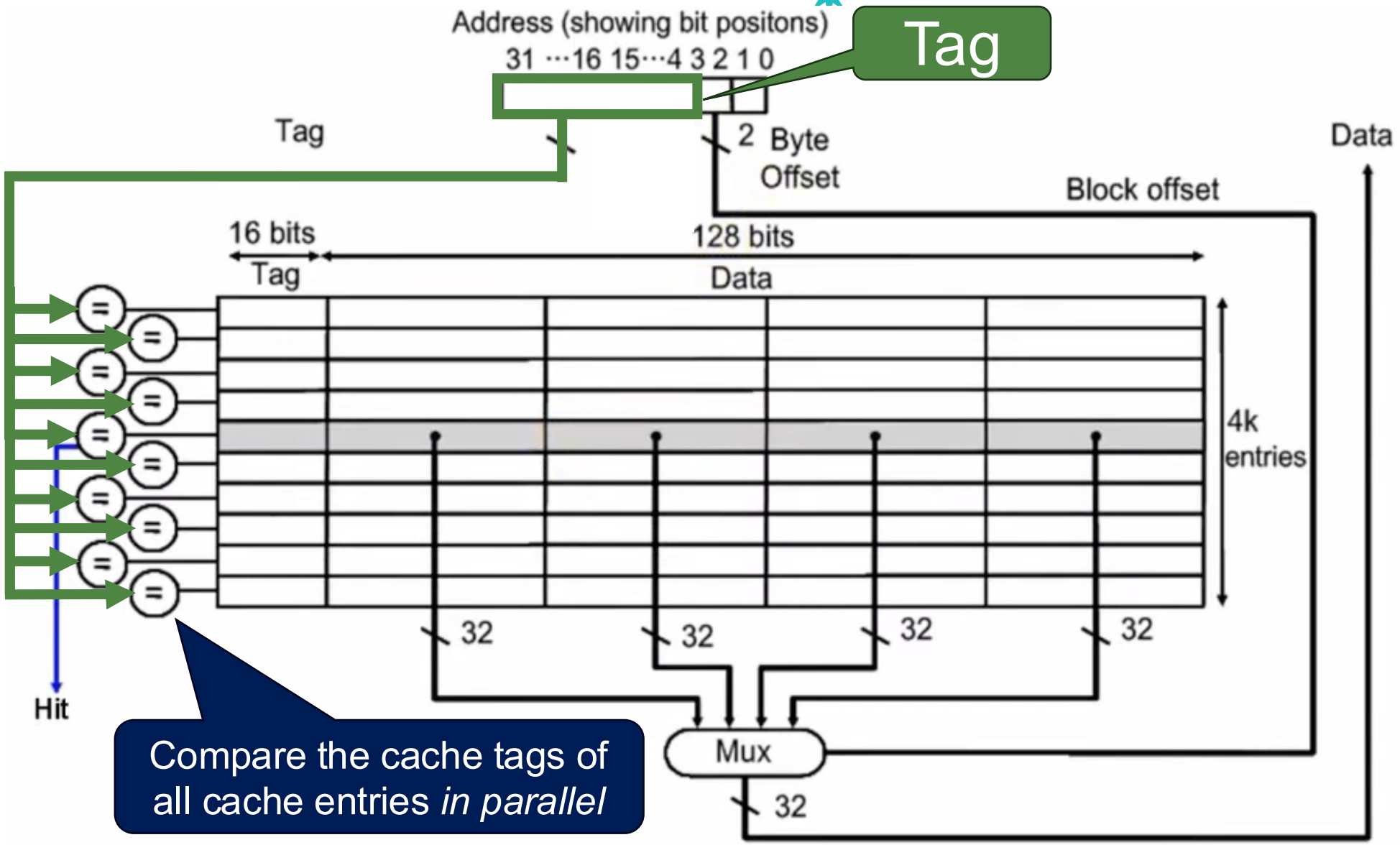
- **Disadvantage**: imagine two frequently-accessed variables. What if their addresses have the same index bits?
  - Such addresses are in conflict in the cache
  - Can't hold both in the cache at once
  - Result in lots of misses (bad!)

Cache Placement Policy: Fully Associative

- No cache index
- Block can be placed **in any location in the cache**
- Requires all entries to be searched at once (compare the cache tags of all cache entries in parallel)

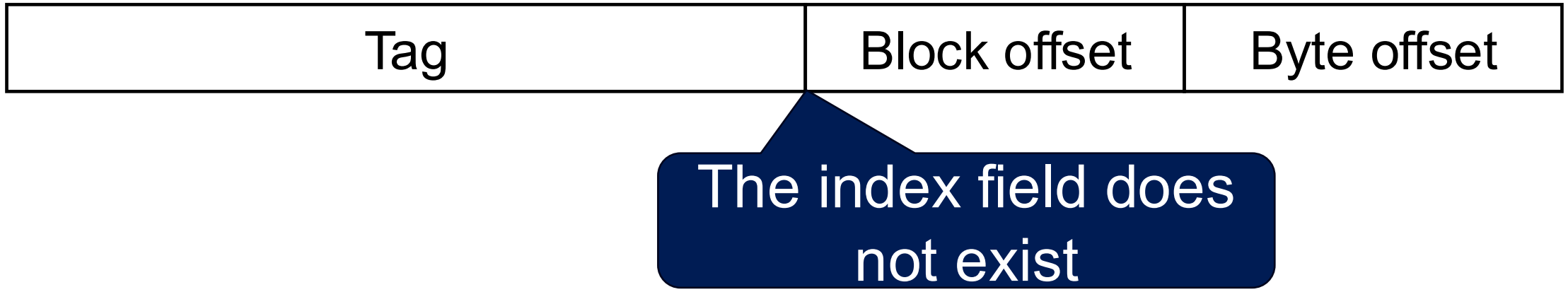


Cache Placement Policy: Fully Associative



Fully Associative: Summary

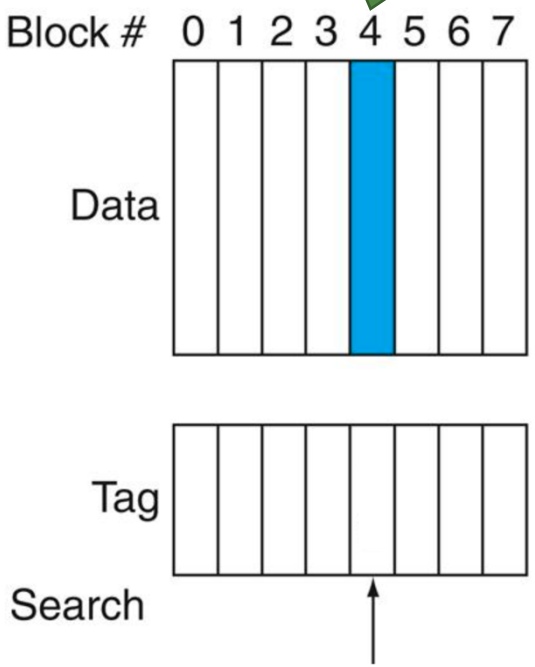
- No cache index
- Block can be placed **in any location in the cache**
- Requires all entries to be searched at once (compare the cache tags of all cache entries in parallel)



- **Disadvantage**: Comparator per entry (expensive hardware)

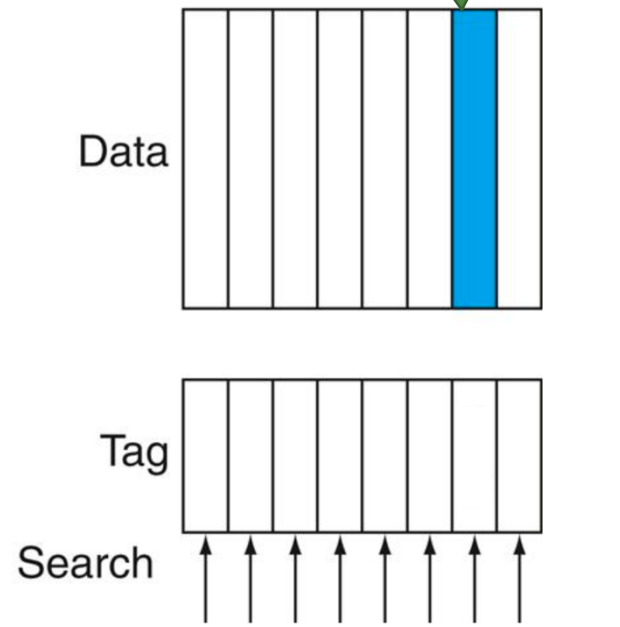
Associativity – Access to Address 12

12 mod 8 = 4



Direct mapped

Any location

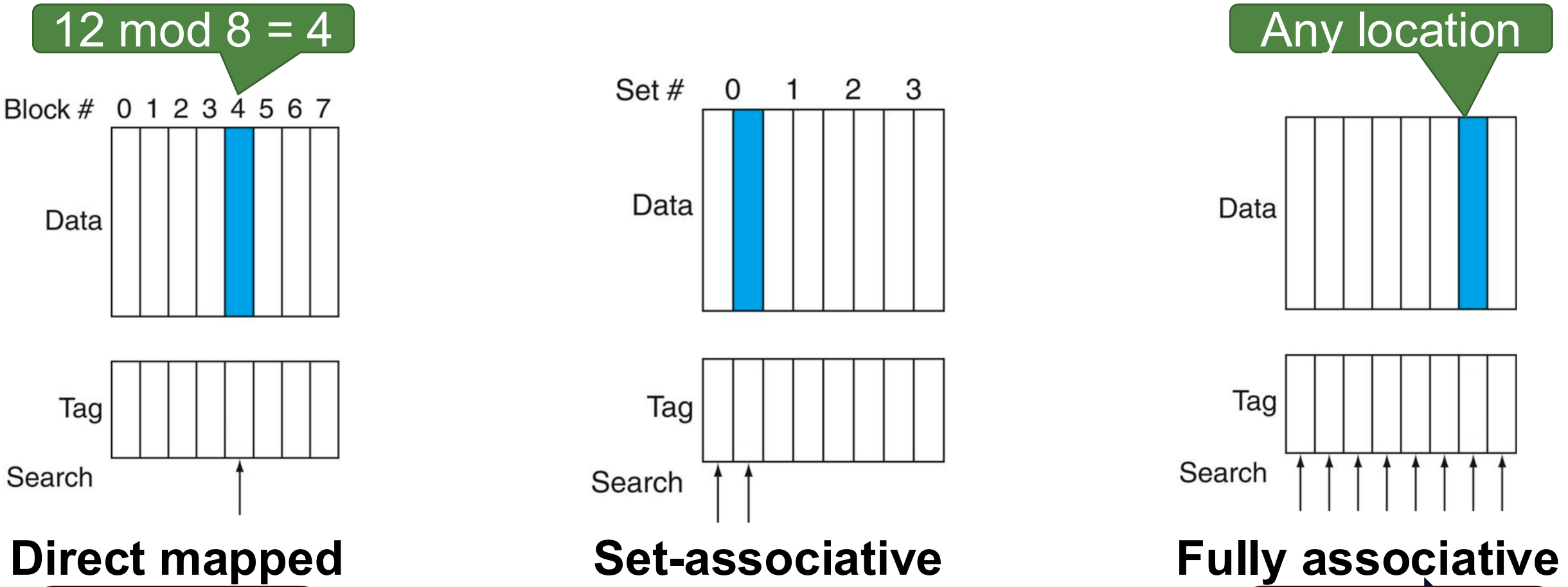


Fully associative

Increasing associativity (shrink index & expand tag)

Reducing miss rate by more flexible placement

Introduction to Set-associative Cache



Direct mapped | **Set-associative** | **Fully associative**

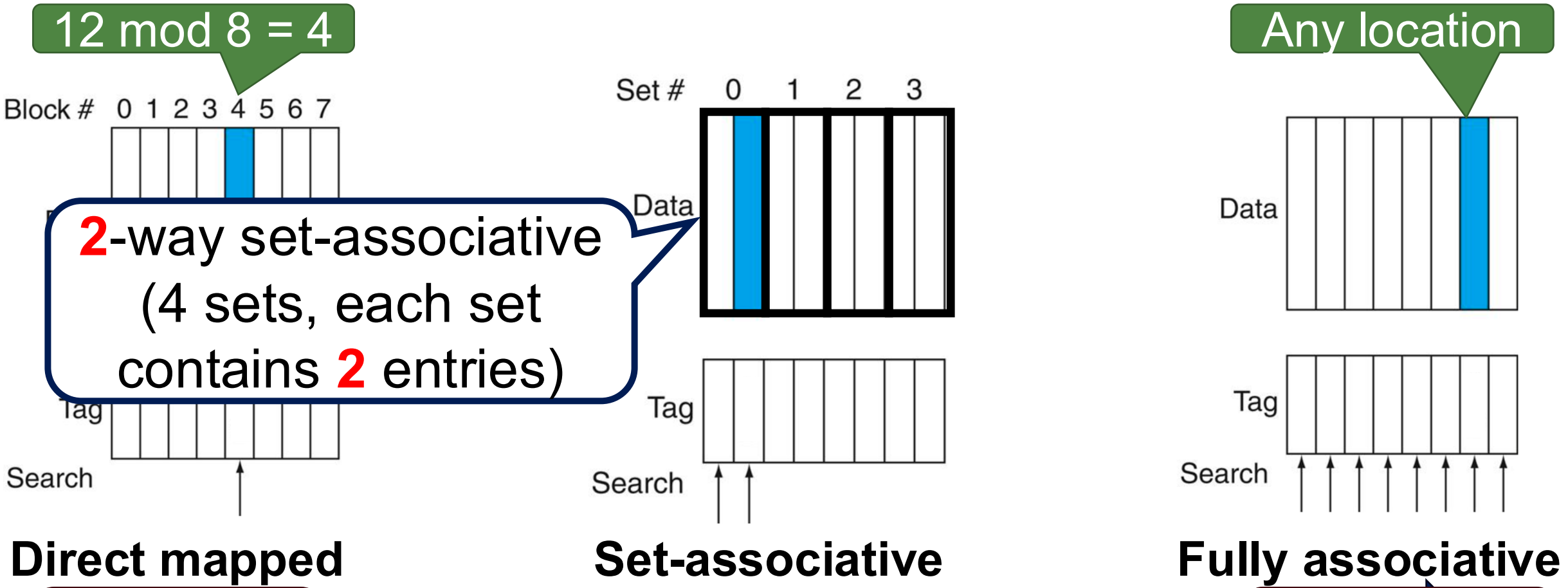
Miss rate ↑

easing associativity (shrink index & expand tag

Reducing miss rate by more flexible placement

HW complexity ↑

Introduction to Set-associative Cache



Miss rate ↑

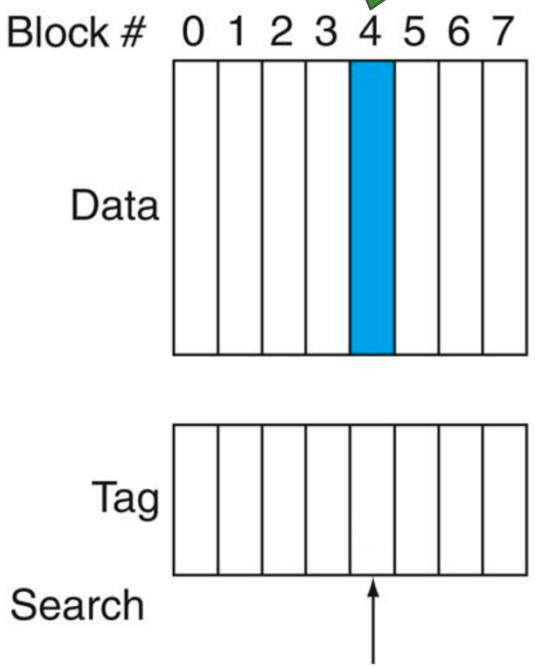
easing associativity (shrink index & expand tag

Reducing miss rate by more flexible placement

HW complexity ↑

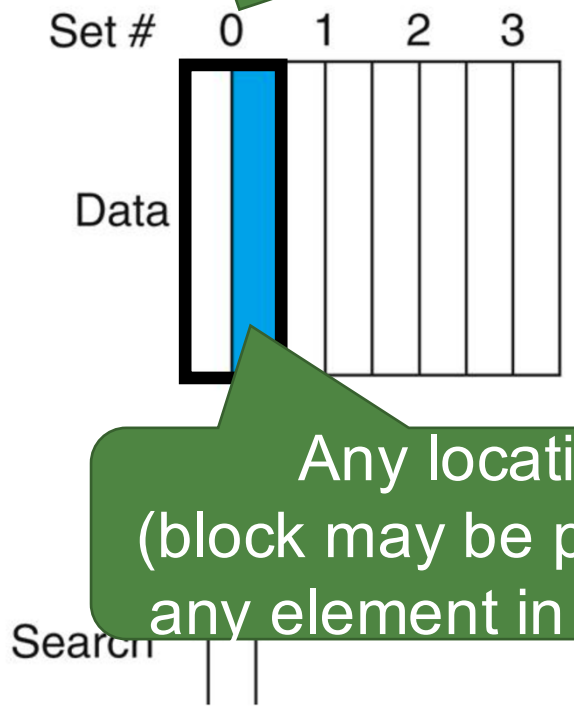
Introduction to Set-associative Cache

12 mod 8 = 4



Direct mapped

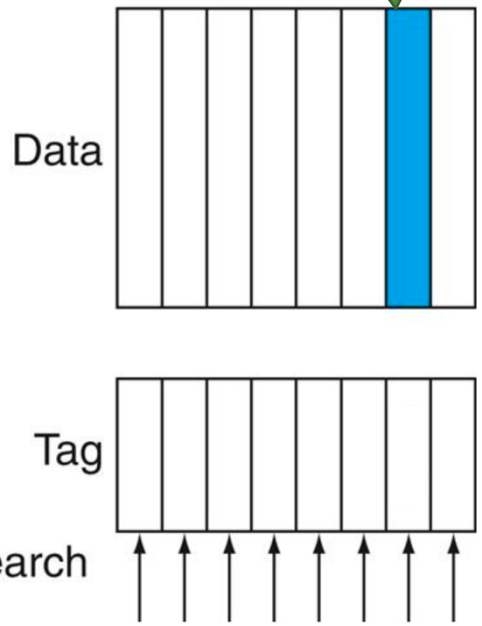
12 mod 4 = 0



Any location
(block may be placed in any element in the set)

Set-associative

Any location



Fully associative

Miss rate ↑

easing associativity (shrink index & expand tag

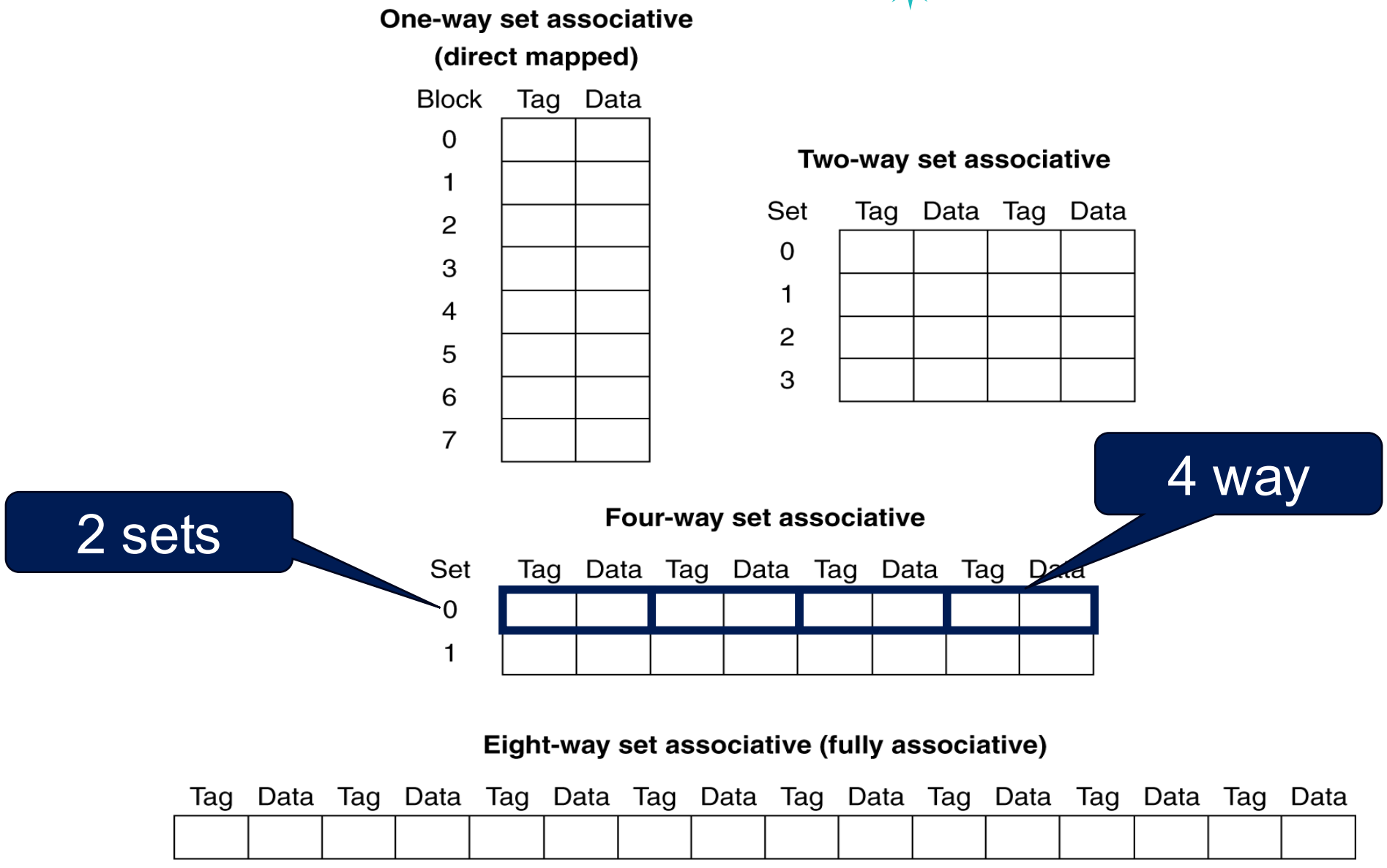
Reducing miss rate by more flexible placement

HW complexity ↑

Cache Placement Policy: *N*-way Set Associative

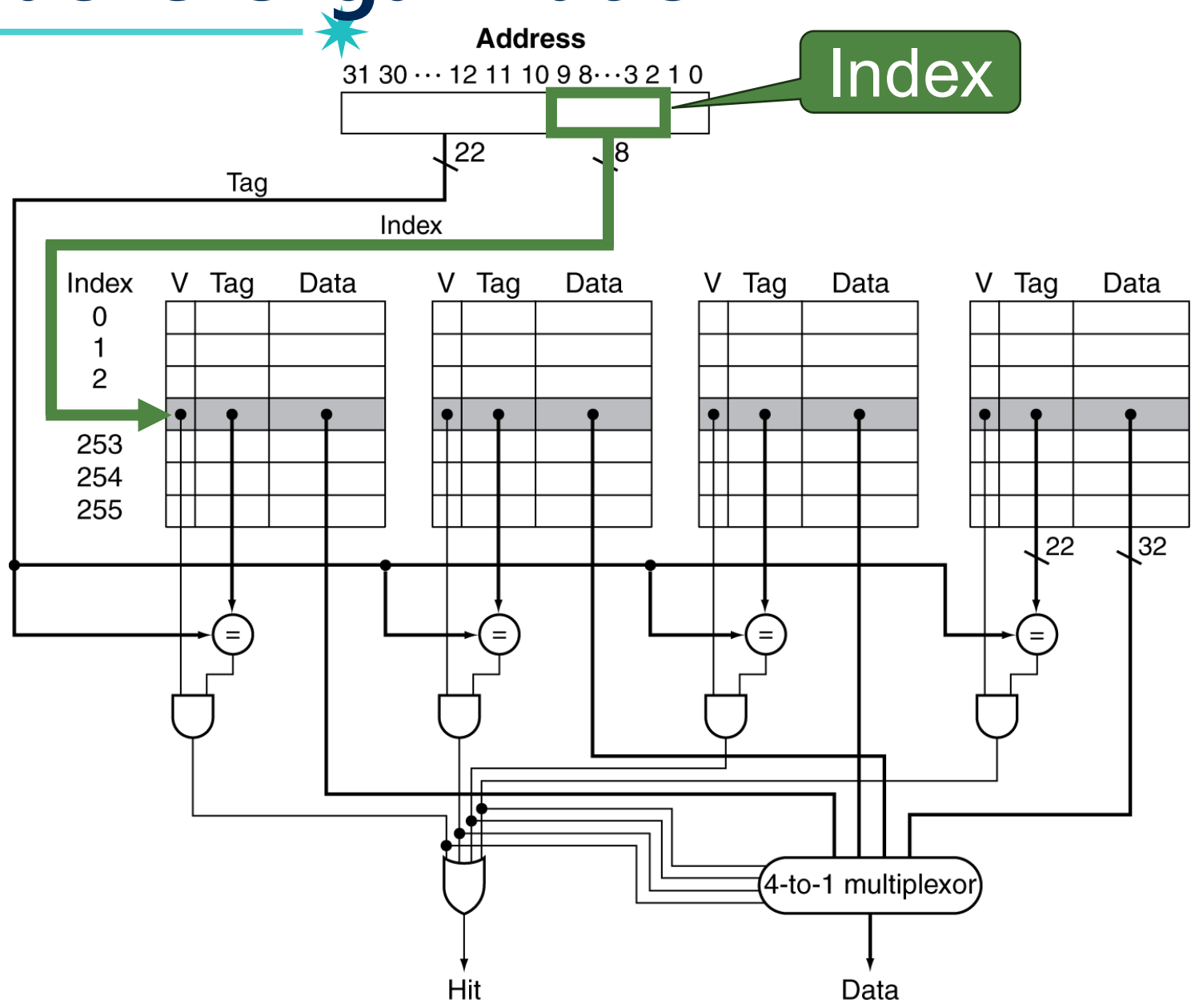
- Allow a given block to go in a target set
  - Position of memory block = (block #) mod (# of sets in the cache)
- Each set contains *N* entries
- Search all entries *in a given set* at once
  - *N* comparators (less expensive)

Spectrum of Associative



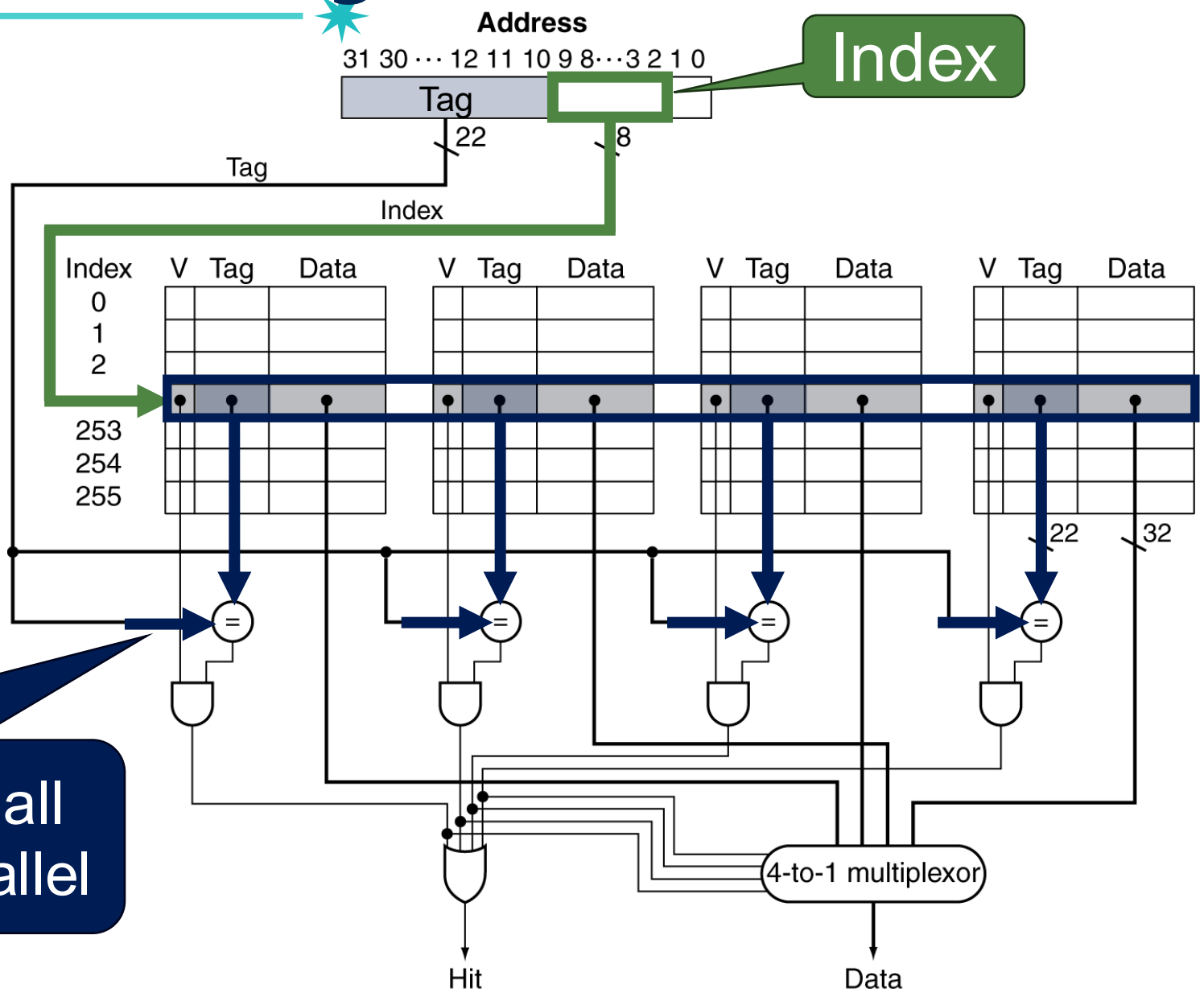
Set Associative Cache Organization

- 1024 blocks
- 4-way set associativity
- 1 block = 1 word



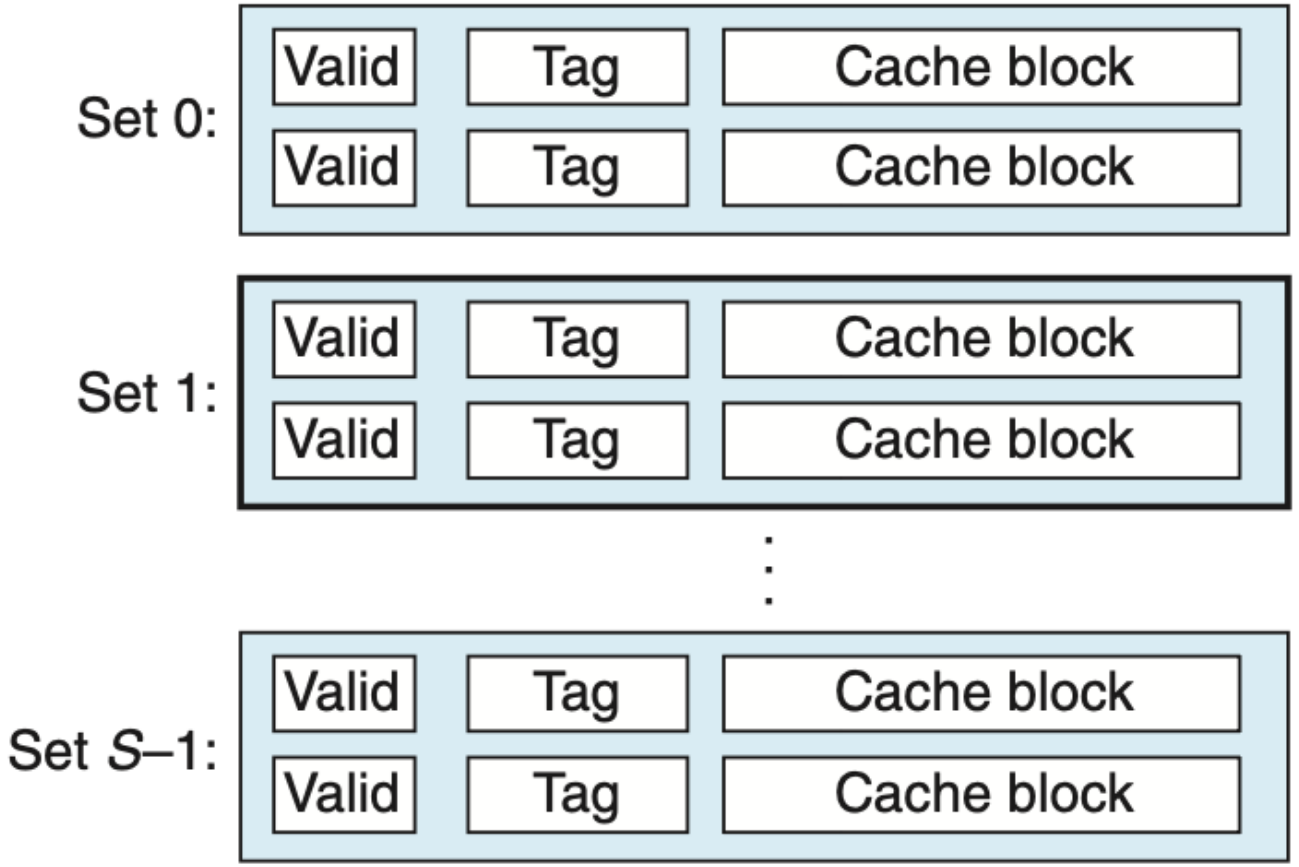
Set Associative Cache Organization

- 1024 blocks
- 4-way set associativity
- 1 block = 1 word

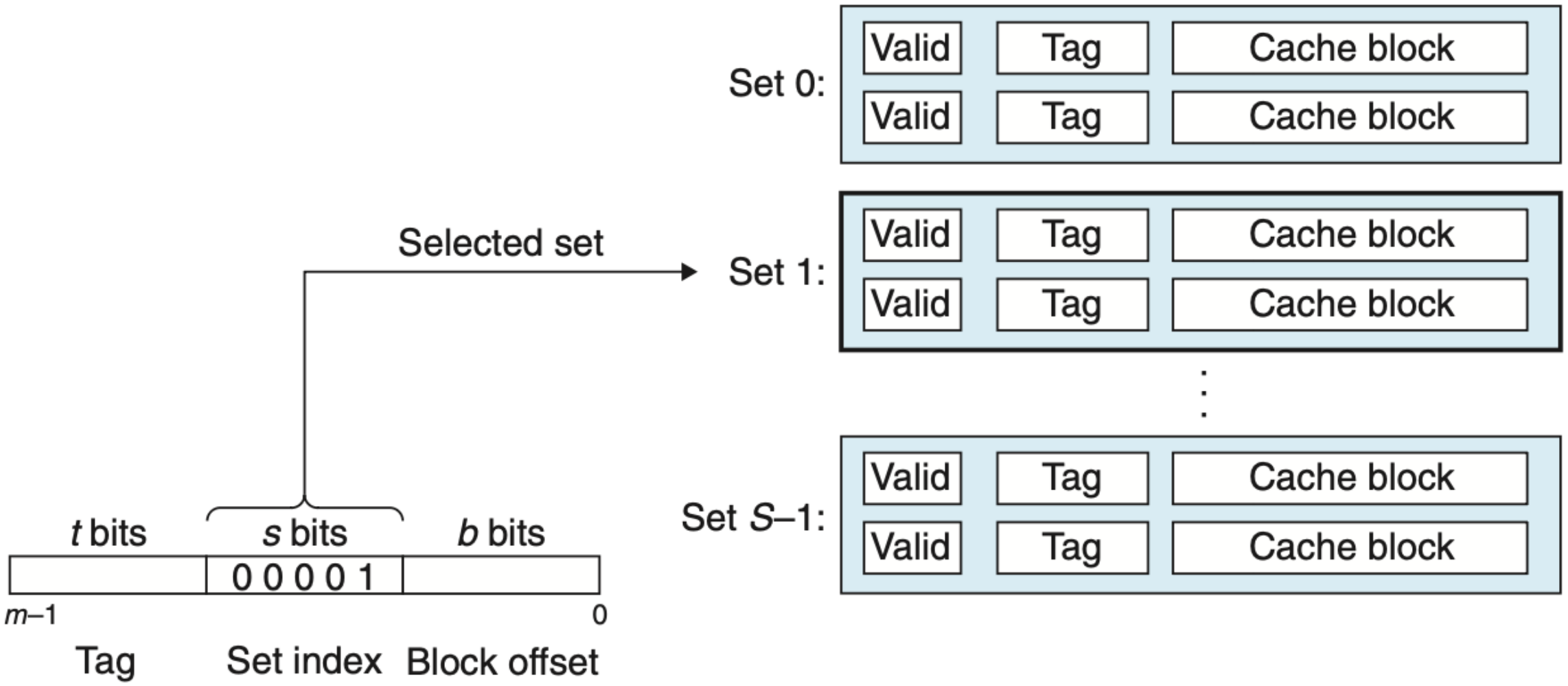


Compare the cache tags of all entries **in a given set** in parallel

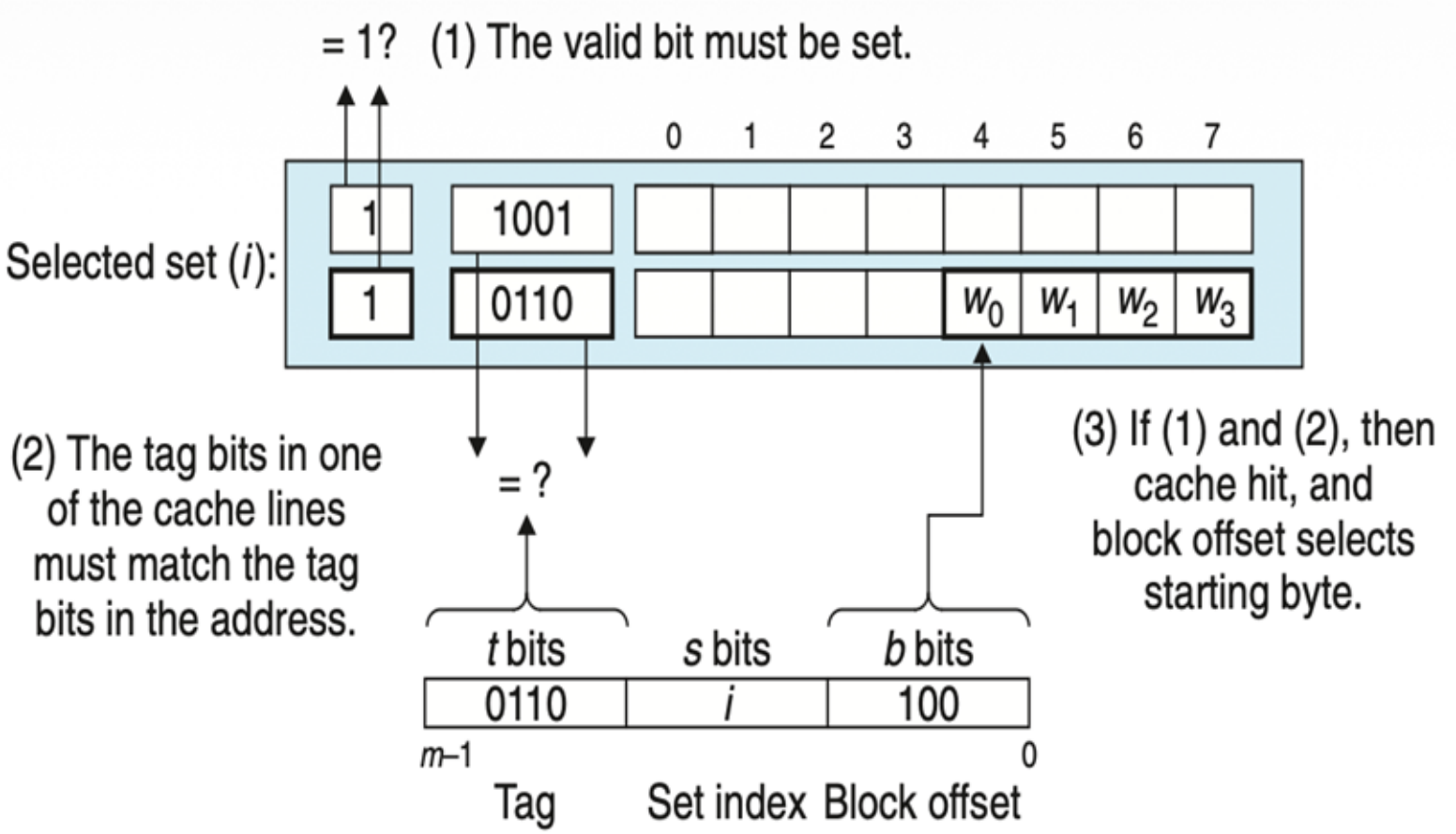
High-level View: Set Associative Cache



High-level View: Set Associative Cache



High-level View: Set Associative Cache



Direct Mapped: Summary

- Increasing associativity (shrink index & expand tag) reduce miss rate by more flexible placement

- **Disadvantage**: ? (Homework4!)

Summary: Cache Placement Policy

Cache placement policy: where can a block be placed?

- Direct mapped
- Fully associative
- Set associative

How to Improve the Cache Performance?

- **Adjust the block size**
- **Adjust the cache placement policy**
- **Adjust the cache replacement policy**
- **Multilevel caches**

How to Improve the Cache Performance?

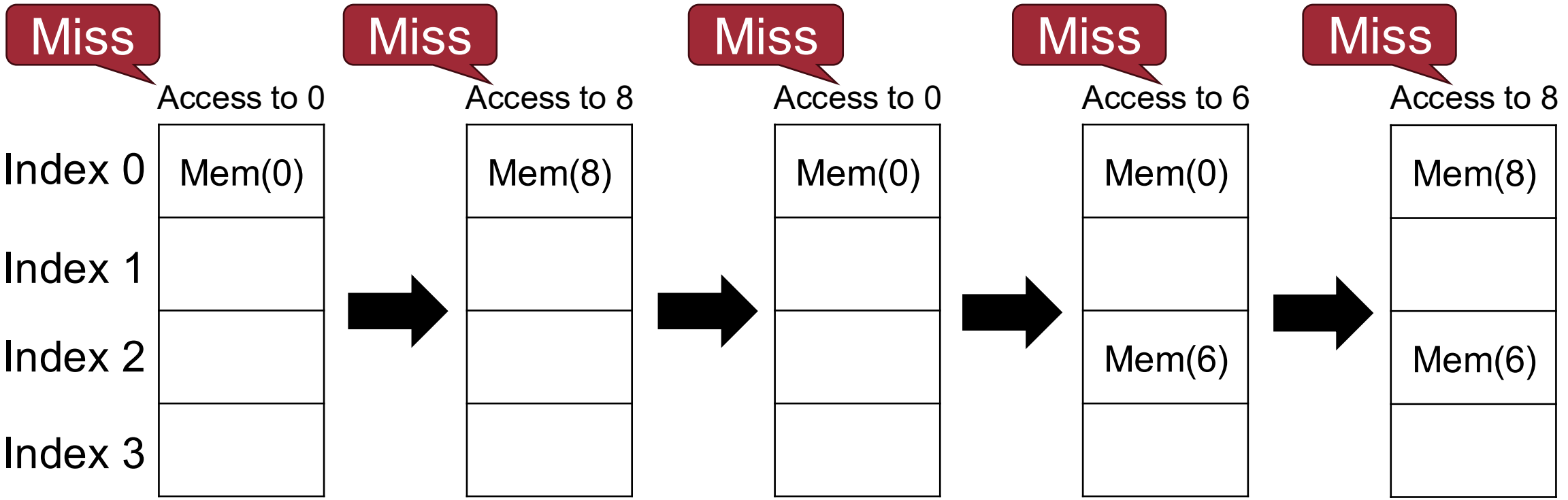
- Adjust the block size
- Adjust the cache placement policy
- Adjust the cache replacement policy
- Multilevel caches

Motivation Example: Cache Replacement Policy

- Block access sequence: 0, 8, 0, 6, 8
- 4 Blocks
- Block size: 4 bytes (1 word)

Motivation Example: Cache Replacement Policy

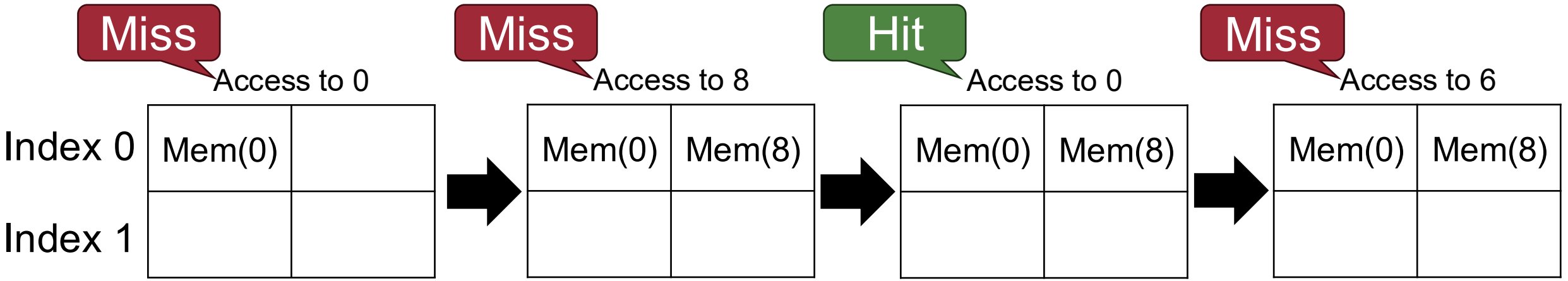
- Block access sequence: 0, 8, 0, 6, 8
- 4 Blocks
- Block size: 4 bytes (1 word)



Direct mapped

Motivation Example: Cache Replacement Policy

- Block access sequence: 0, 8, 0, 6, 8
- 4 Blocks
- Block size: 4 bytes (1 word)

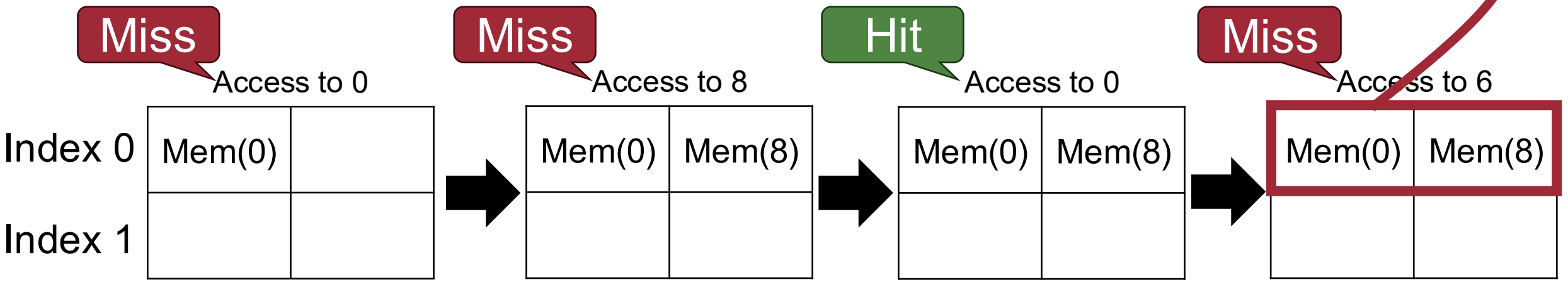


2-way set associative

Motivation Example: Cache Replacement Policy

- Block access sequence: 0, 8, 0, 6, 8
- 4 Blocks
- Block size: 4 bytes (1 word)

Which block should be evicted?



2-way set associative

Which Block Should be Evicted on a Cache Miss?

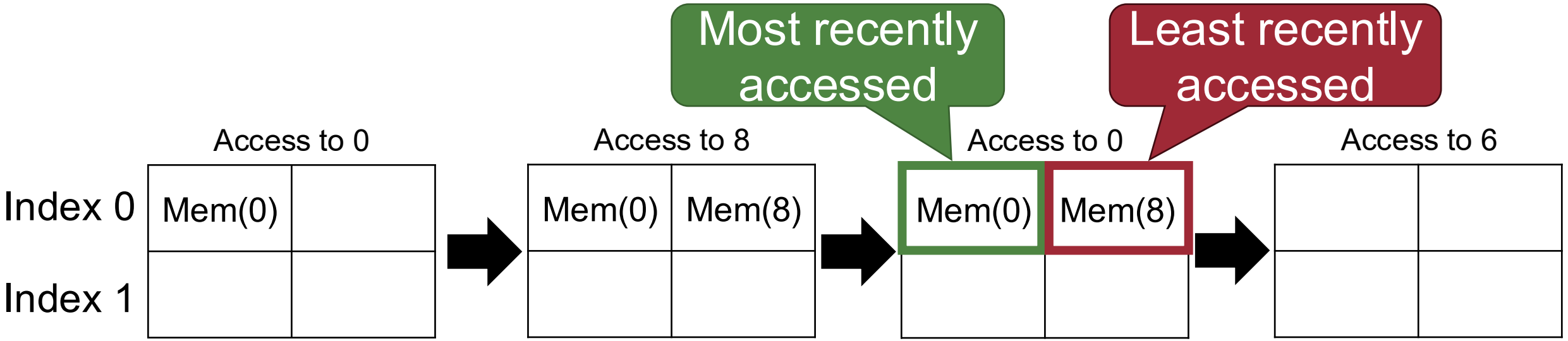
- Direct mapped: no choice
- Set associative & fully associative: follow the **replacement policy**

Replacement Policy (Determined which block gets evicted)

- **Least Recently Used (LRU)**: replace the one NOT used (accessed) for the longest time

Replacement Policy (Determined which block gets evicted)

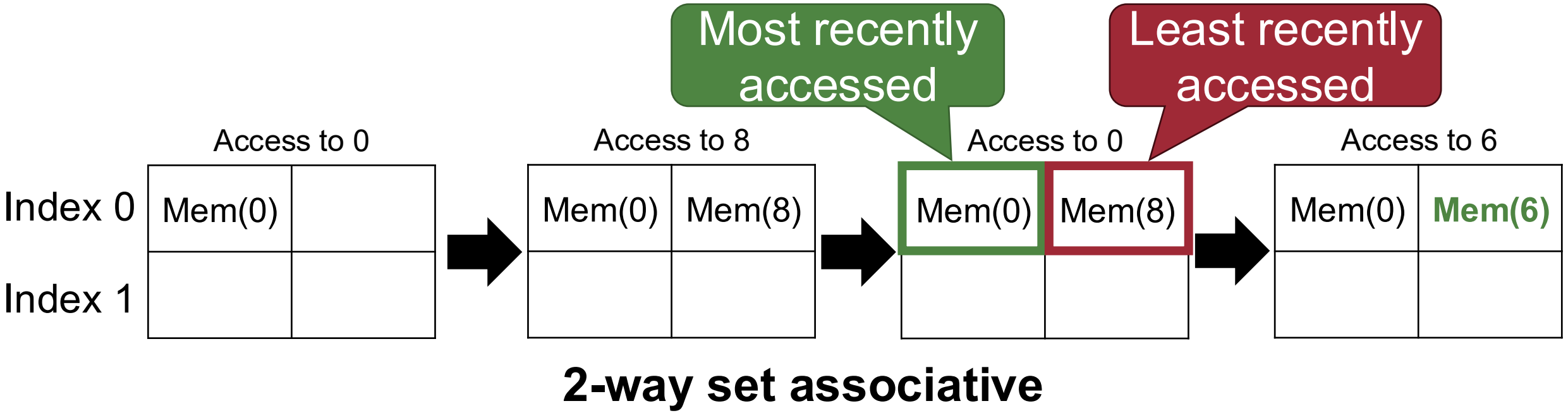
- **Least Recently Used (LRU)**: replace the one NOT used (accessed) for the longest time
  - Temporal locality of access is considered
  - Need a reference history information



2-way set associative

Replacement Policy (Determined which block gets evicted)

- **Least Recently Used (LRU)**: replace the one NOT used (accessed) for the longest time
  - Temporal locality of access is considered
  - Need a reference history information



2-way set associative

Replacement Policy (Determined which block gets evicted)

- **Least Recently Used (LRU)**: replace the one NOT used (accessed) for the longest time
  - Temporal locality of access is considered
  - Need a reference history information

- **Random**
  - Gives approximately the same performance as LRU for high associativity

+ **First In First Out (FIFO)**: replace the block with long-lived in the cache

+ **Least Frequently Used (LFU)**: replace the block with fewest reference

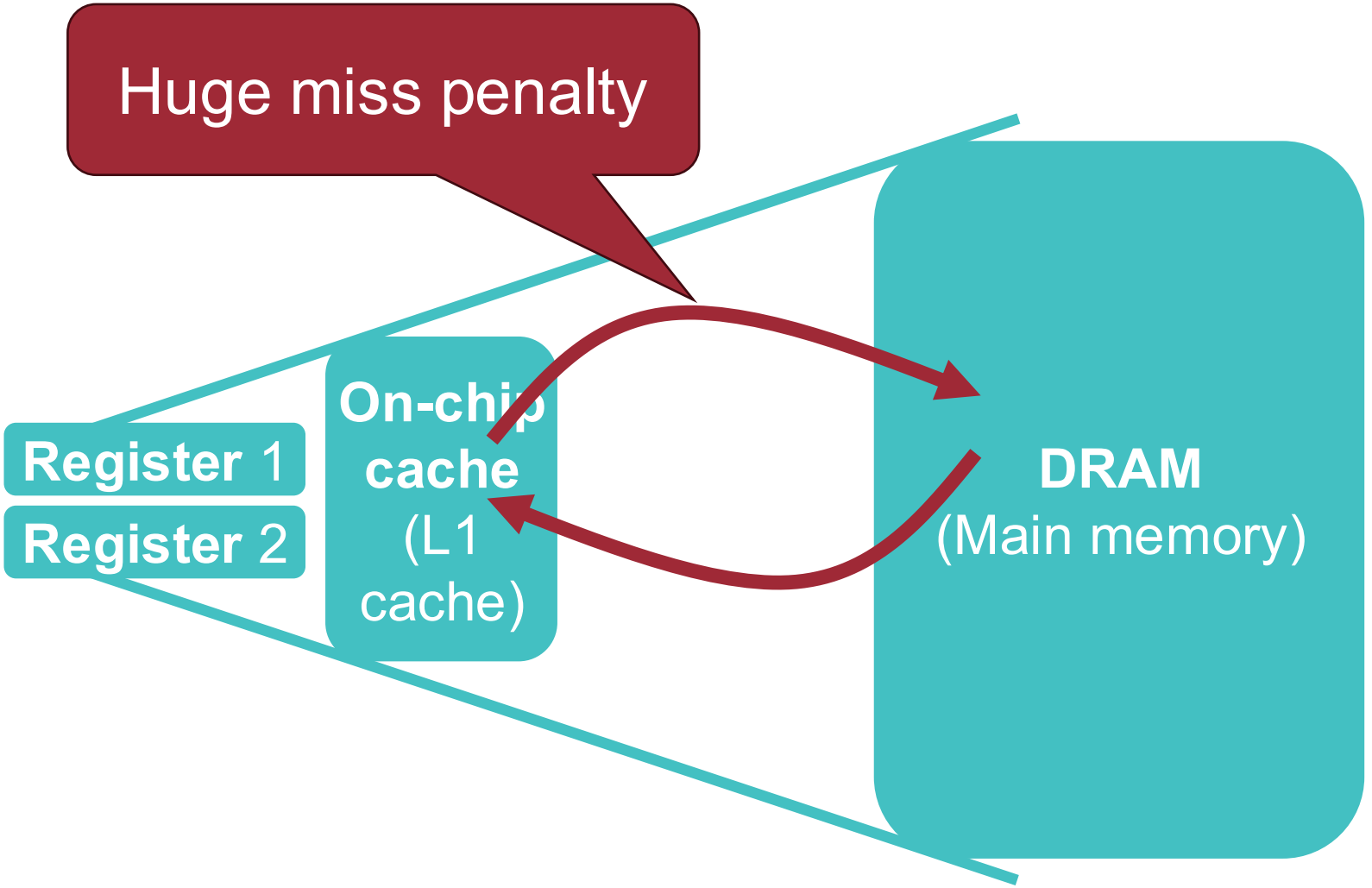
How to Improve the Cache Performance?

- Adjust the block size
- Adjust the cache placement policy
- Adjust the cache replacement policy
- Multilevel caches

How to Improve the Cache Performance?

- Adjust the block size
- Adjust the cache placement policy
- Adjust the cache replacement policy
- Multilevel caches

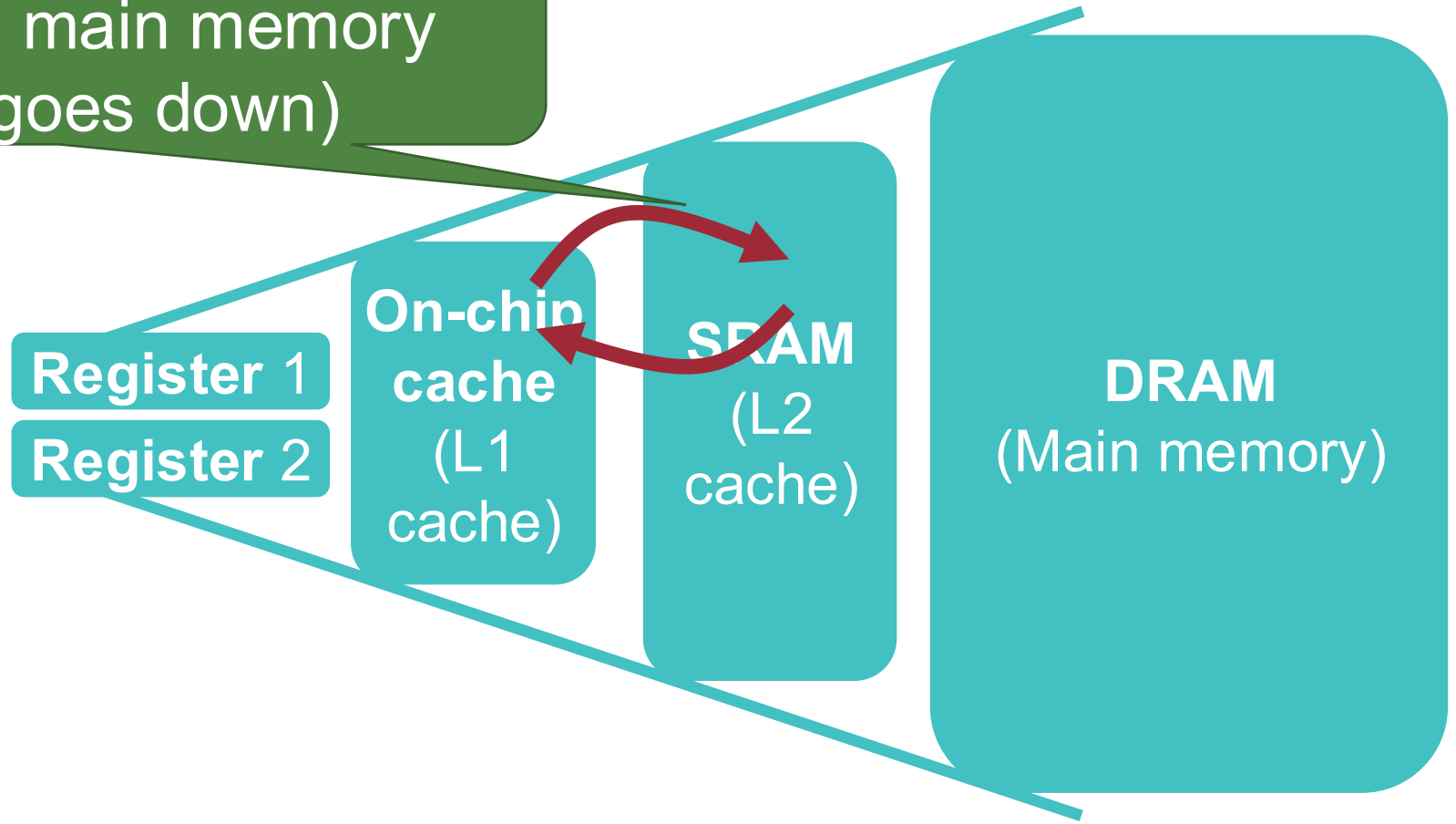
Multilevel Caches



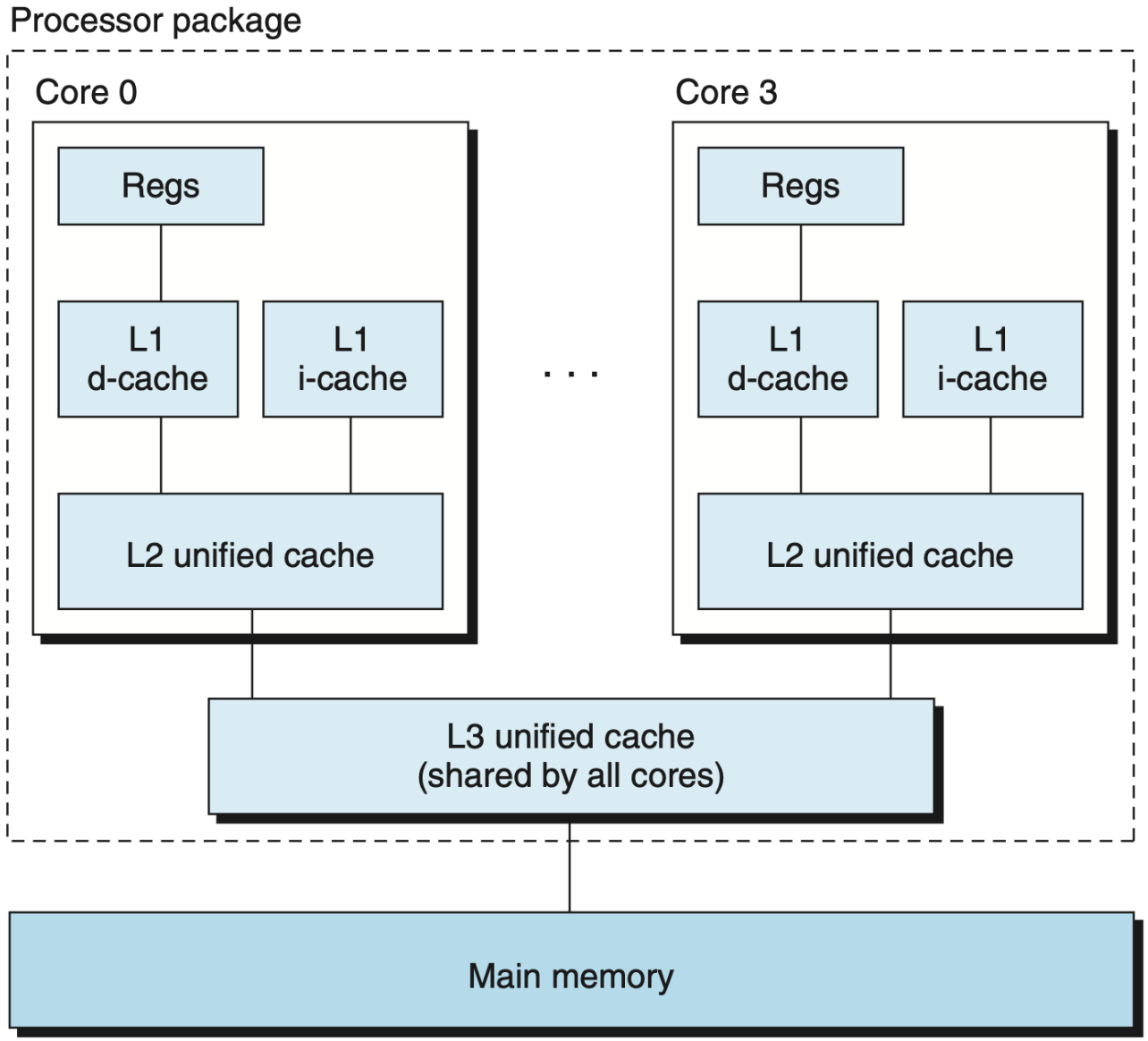
Multilevel Caches

- Add a second level (L2) cache

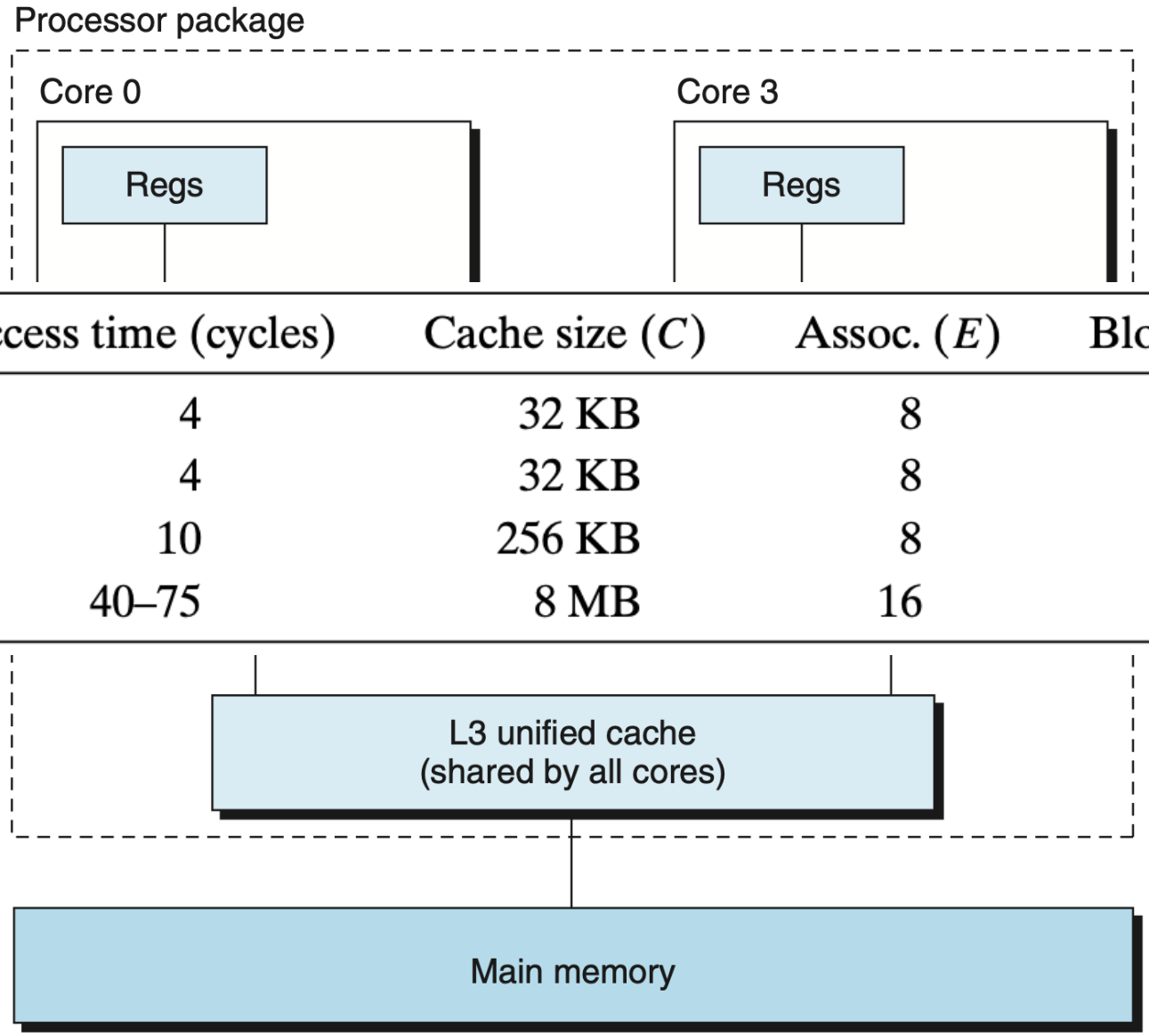
Larger and slower than the L1 cache, but sill faster than main memory (Miss penalty goes down)



Intel Core i7 Cache Hierarchy



Intel Core i7 Cache Hierarchy



Cache type	Access time (cycles)	Cache size (C)	Assoc. (E)	Block size (B)	Sets (S)
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	10	256 KB	8	64 B	512
L3 unified cache	40–75	8 MB	16	64 B	8,192

Summary: How to Improve the Cache Performance?

- **Adjust the block size**
  - When the cache block size is too small or too large, cache shows poor performance
- **Adjust the cache placement policy**
  - Direct mapped, fully associative, set associative
  - A fully associative cache theoretically shows best performance, but it is complex
- **Adjust the cache replacement policy**
  - Generally, LRU usually shows best performance
- **Multilevel caches**
  - Nowadays, there are more than one level of cache

Question?