Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2024

Lecture 19:
Memory Hierarchy pt 2

Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

Lecture slides adapted from Bill Leahy and Charles Lively of Georgia Tech

Modern memory hierarchy

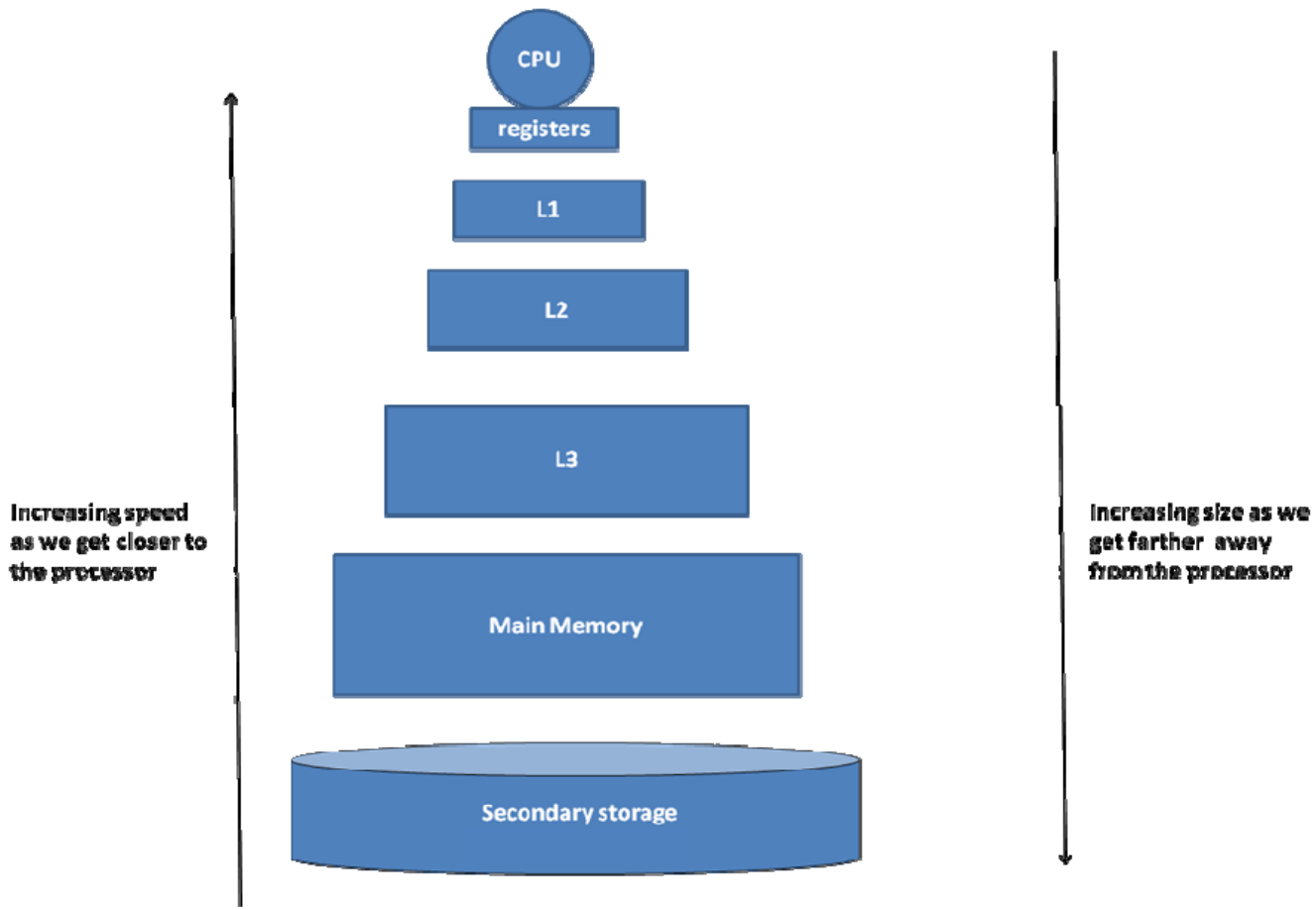


Figure 9.2: The entire memory hierarchy stretching from processor registers to the virtual memory.

Modern memory hierarchy

$AMAT_i = T_i + M_i * AMAT_{i+1}$

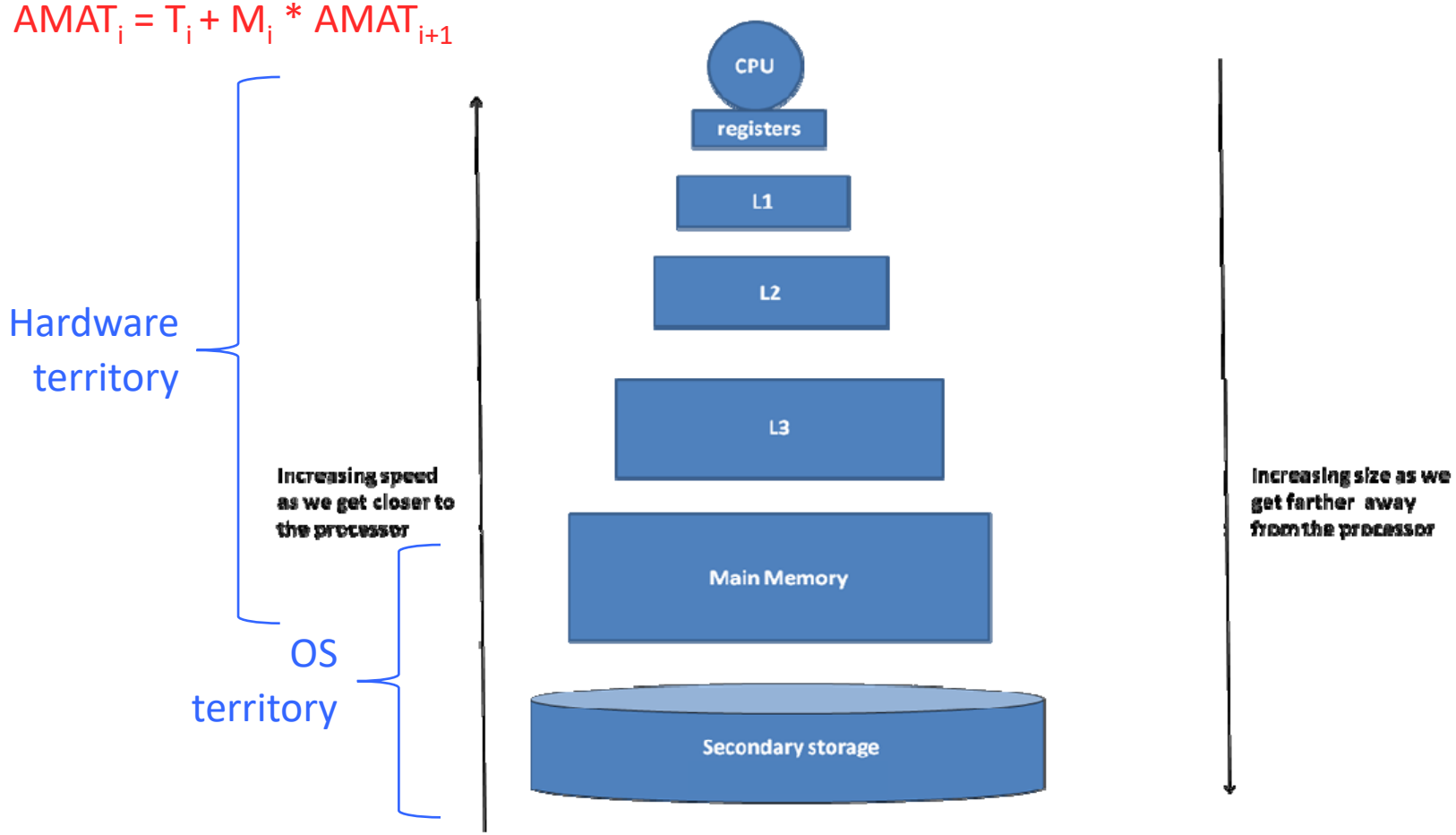
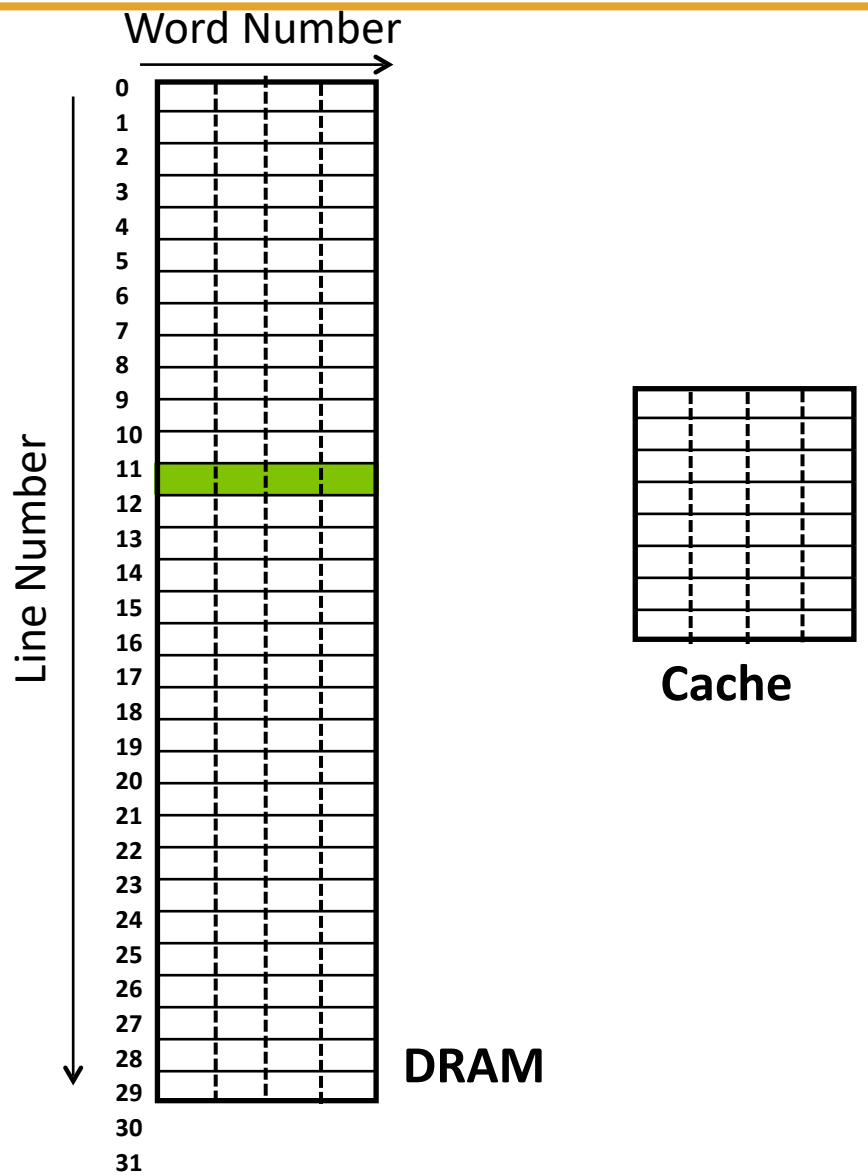


Figure 9.2: The entire memory hierarchy stretching from processor registers to the virtual memory.

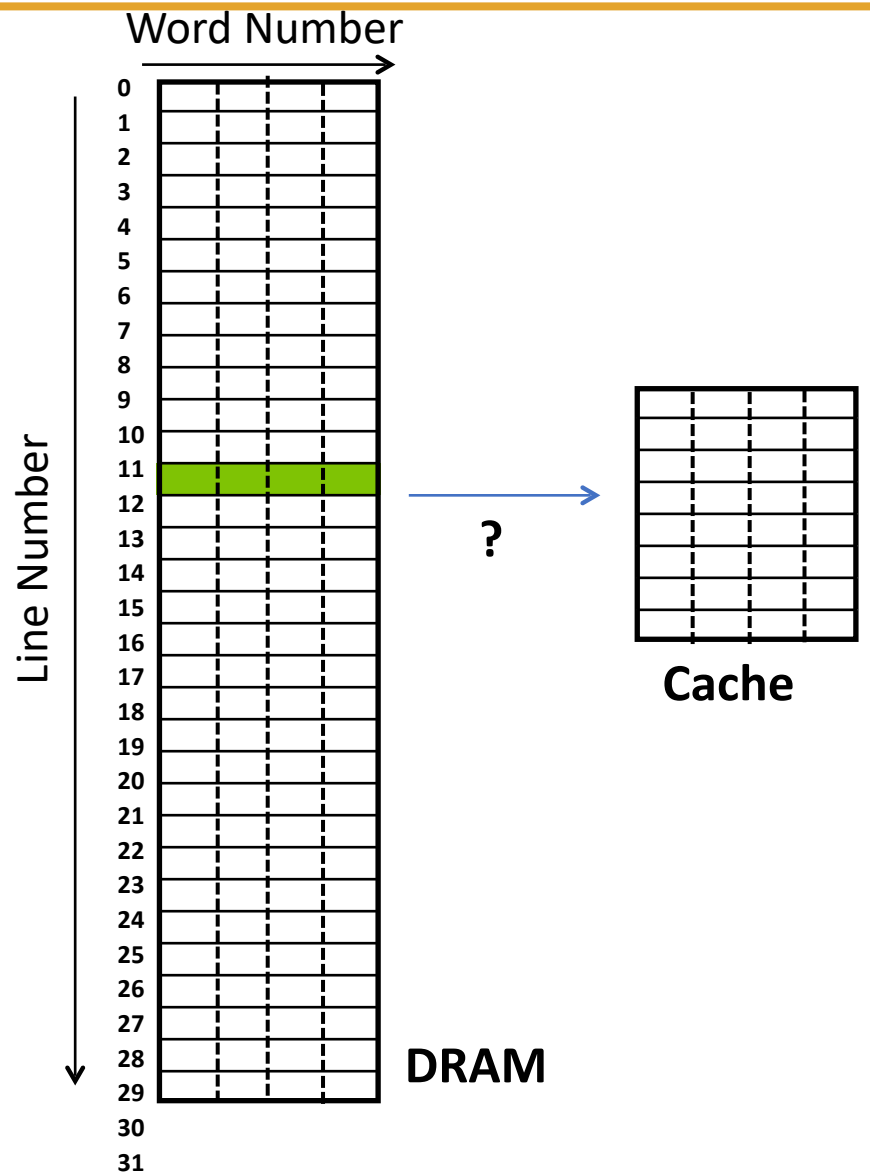
Cache Organizations

- Direct mapped
- Fully associative
- Set associative

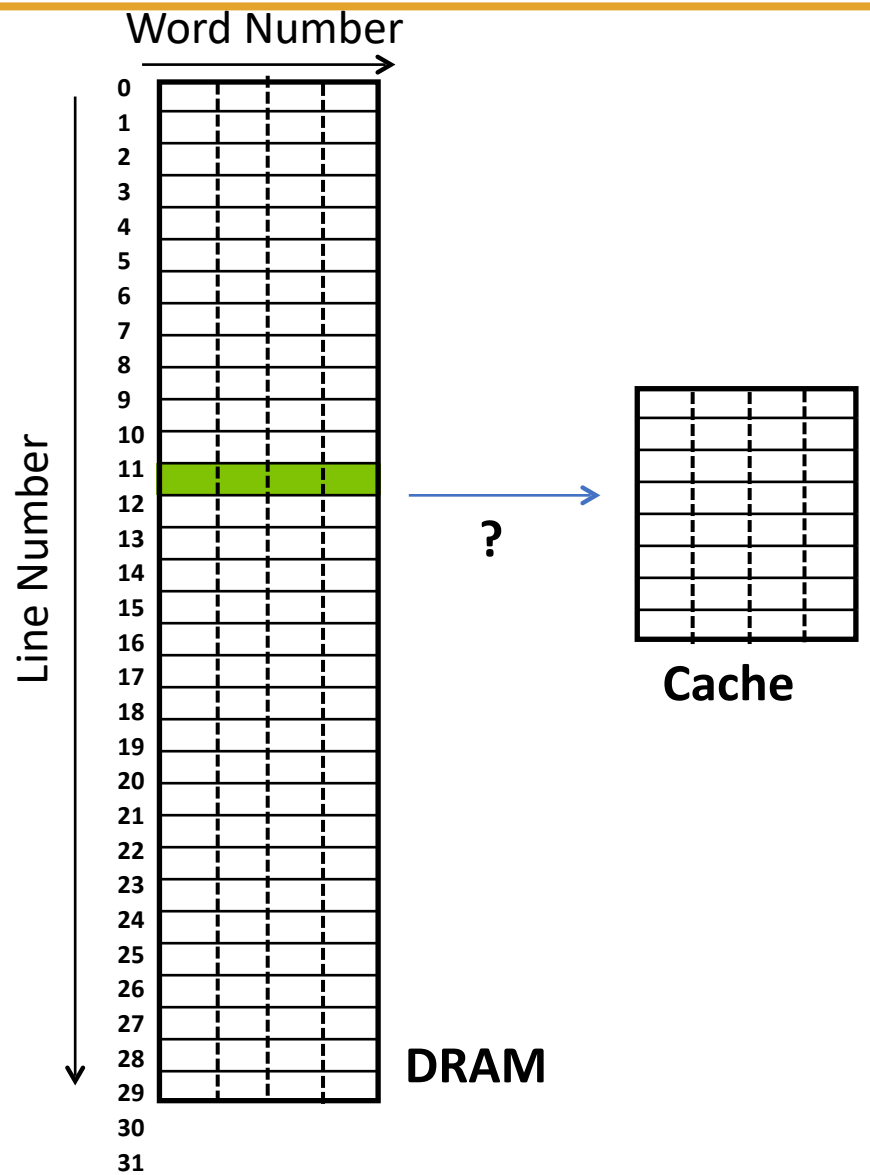
Cache Placement

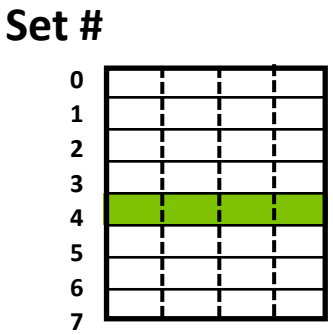


Cache Placement



Cache Placement

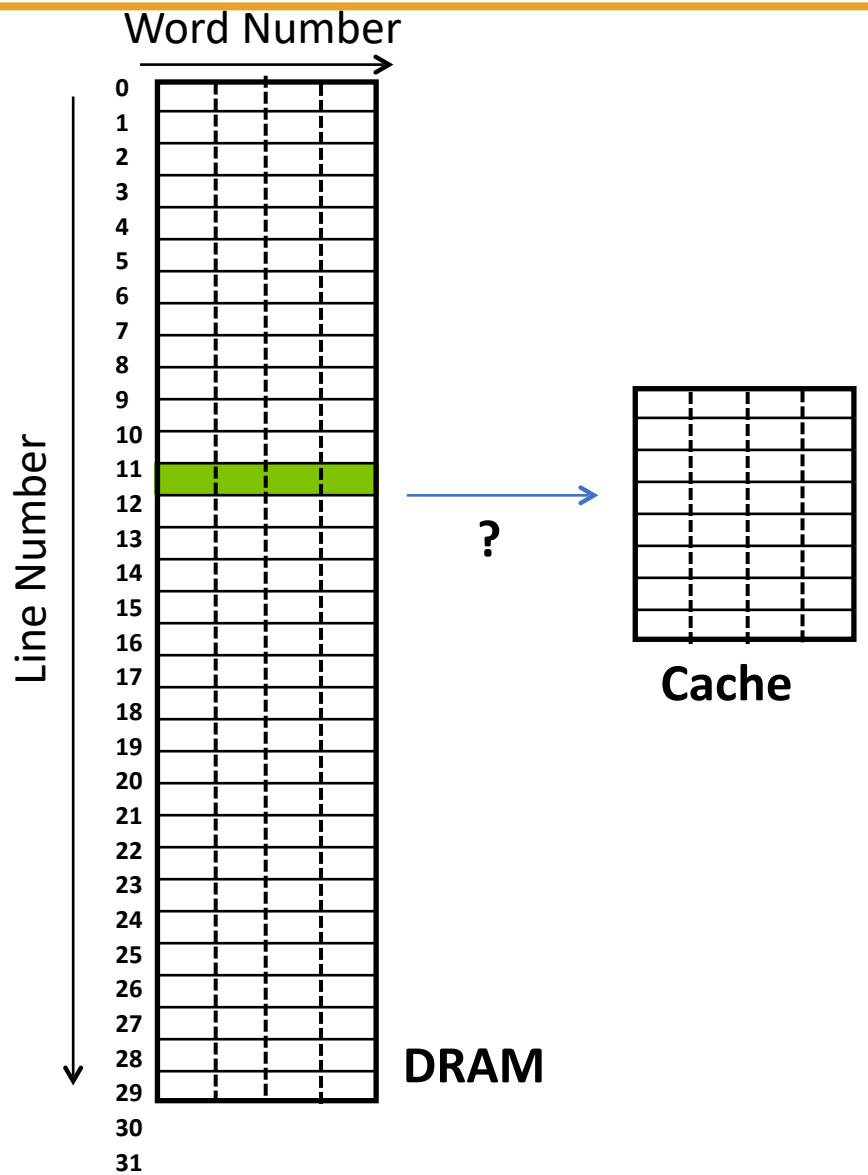


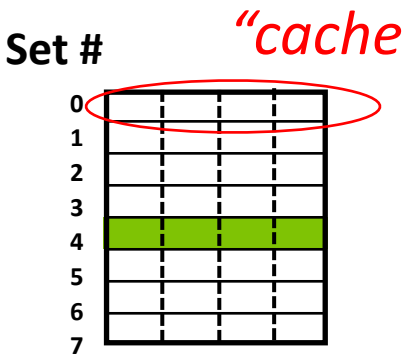


Direct Mapped

only one location

Cache Placement



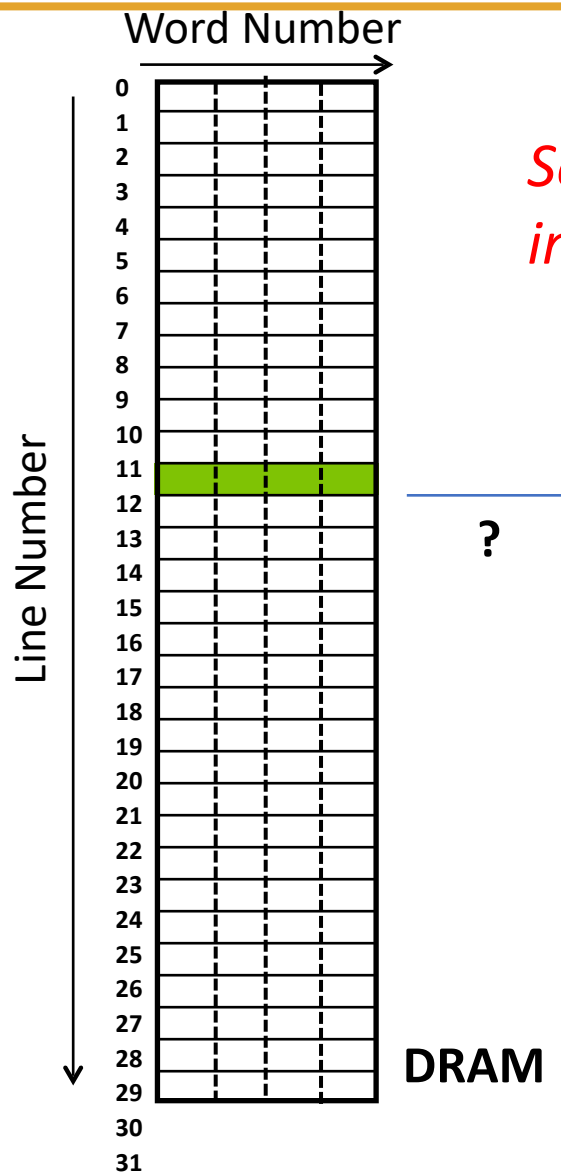


"cache block" or "cache line"

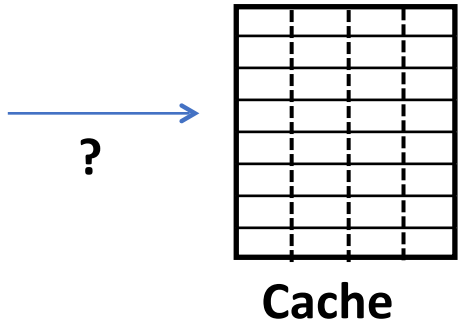
Direct Mapped

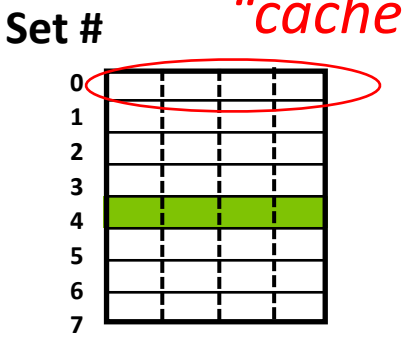
only one location

Cache Placement



Searching is easy but inefficient use of space



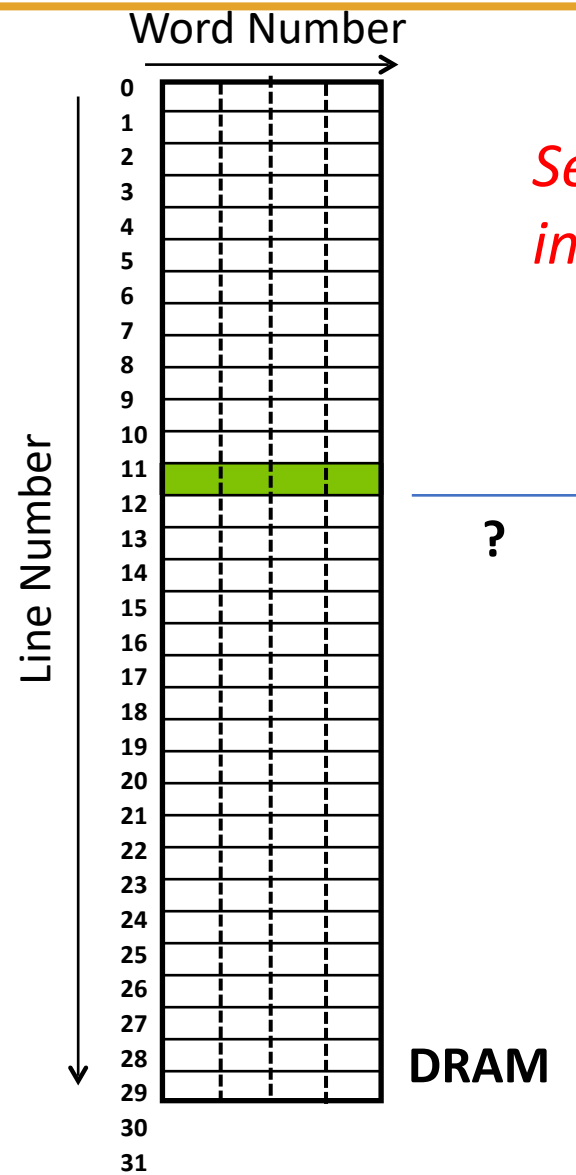


"cache block" or "cache line"

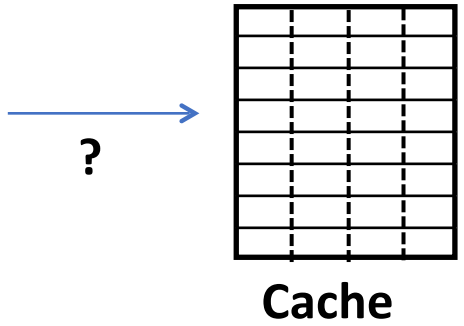
Direct Mapped

only one location

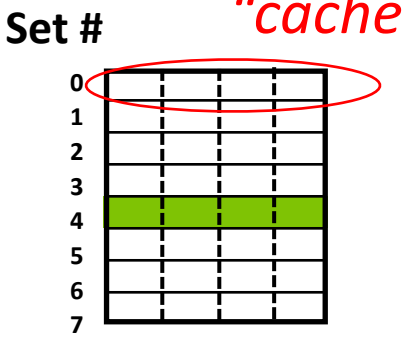
Cache Placement



Searching is easy but inefficient use of space

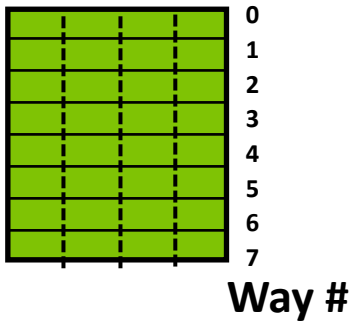


"cache block" or "cache line"



Direct Mapped

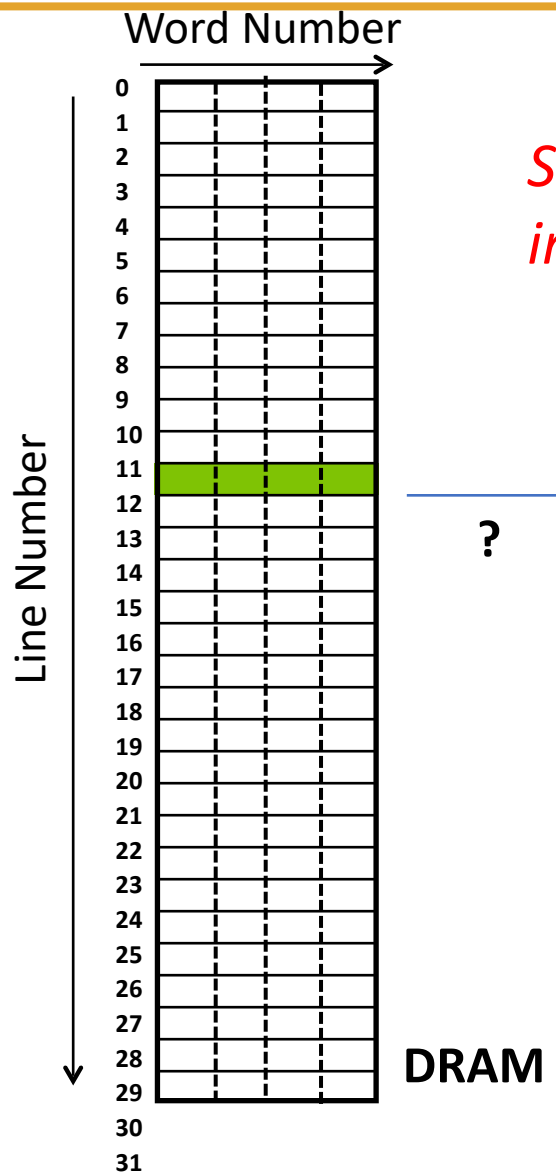
only one location



Fully Associative

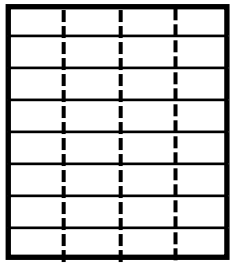
anywhere

Cache Placement



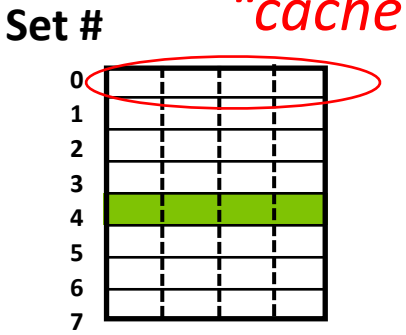
Searching is easy but inefficient use of space

?



Cache

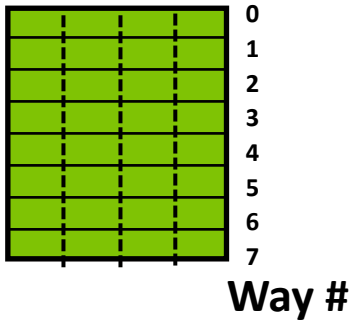
Searching is more expensive



"cache block" or "cache line"

Direct Mapped

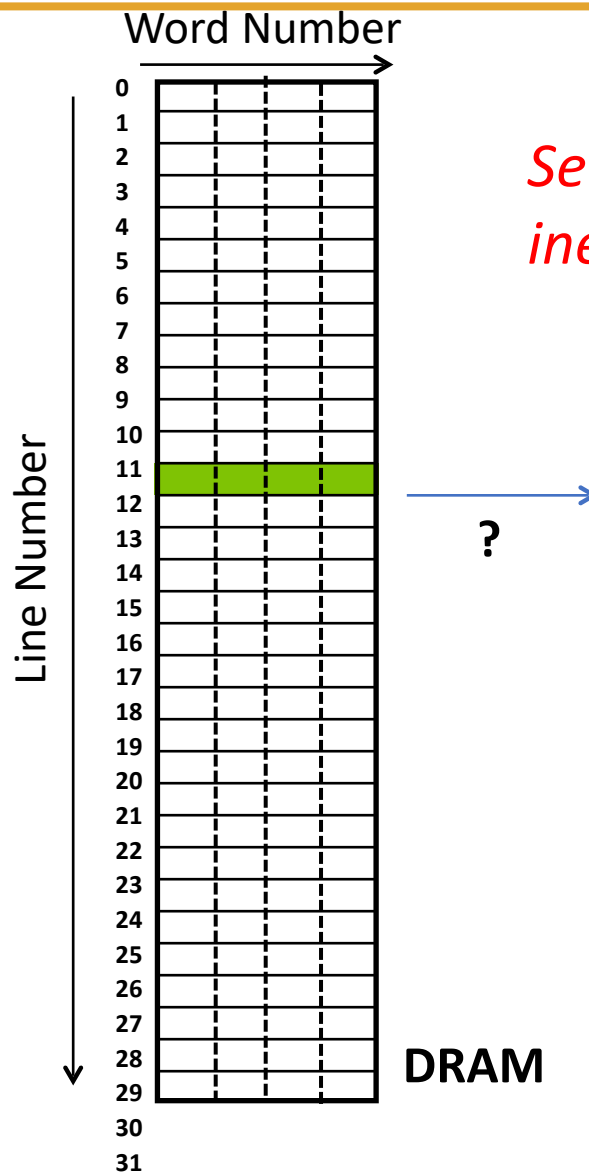
only one location



Fully Associative

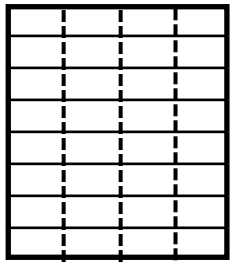
anywhere

Cache Placement



Searching is easy but inefficient use of space

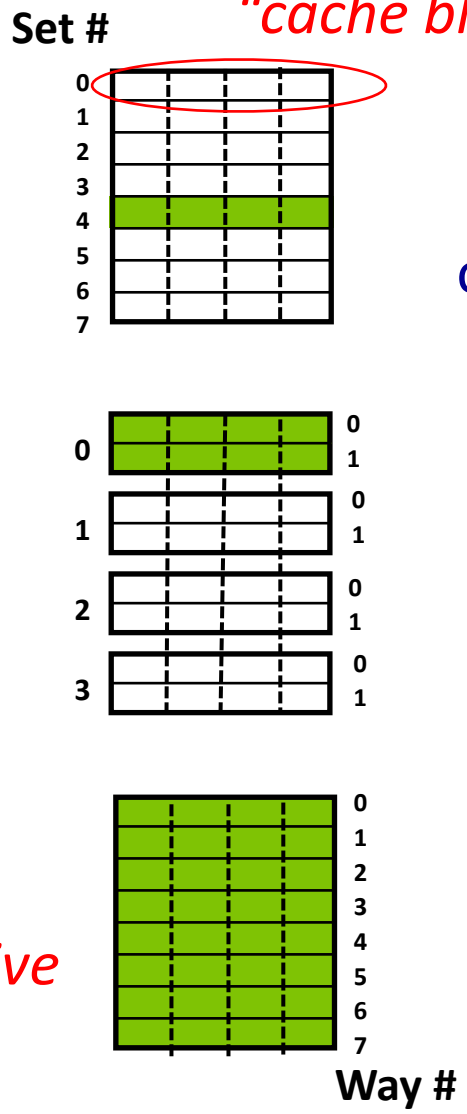
?



Cache

Searching is more expensive

"cache block" or "cache line"



Direct Mapped

only one location

(2-way) Set Associative

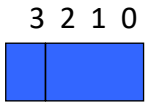
2 locations

Fully Associative

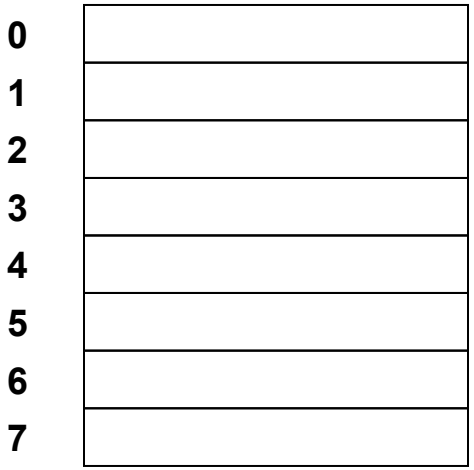
anywhere

Direct mapped cache

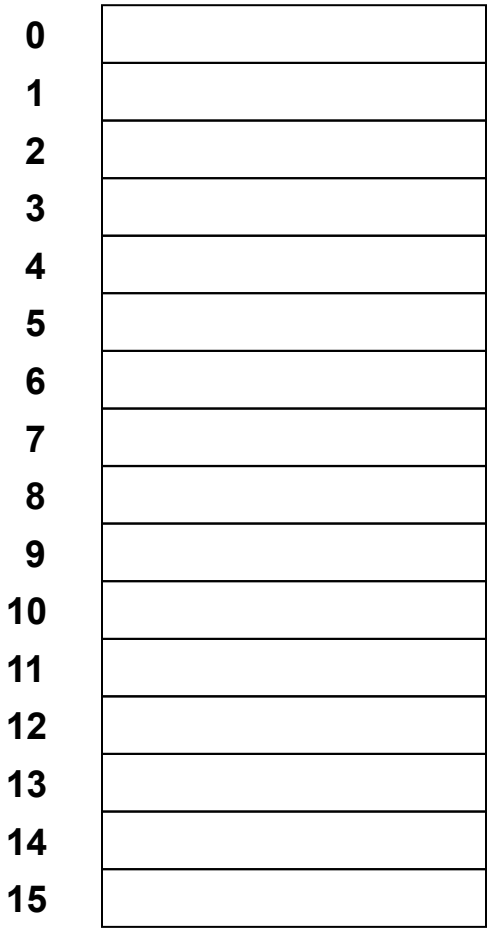
For now we will assume:
1. word-addressable memory
2. each cache block entry is a word



← Example: 4-bit memory address



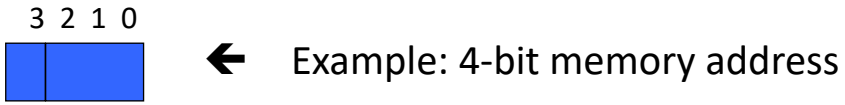
Cache



Main Memory

Direct mapped cache

For now we will assume:
1. word-addressable memory
2. each cache block entry is a word



← Example: 4-bit memory address

0	
1	
2	
3	
4	
5	
6	
7	

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

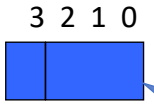
Main Memory

Given a memory address,
What is the numerical value of cache index?

An address' index tells me which **set** of the cache is the cache block's home location

Direct mapped cache

For now we will assume:
1. word-addressable memory
2. each cache block entry is a word



← Example: 4-bit memory address

3-bit cache **index**
($\log_2$ of cache size)

0	
1	
2	
3	
4	
5	
6	
7	

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

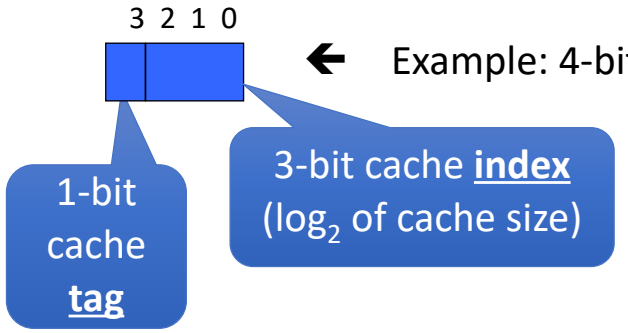
Main Memory

Given a memory address,
What is the numerical value of cache index?

An address' index tells me which **set** of the cache is the cache block's home location

Direct mapped cache

For now we will assume:
1. word-addressable memory
2. each cache block entry is a word



← Example: 4-bit memory address

0	
1	
2	
3	
4	
5	
6	
7	

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

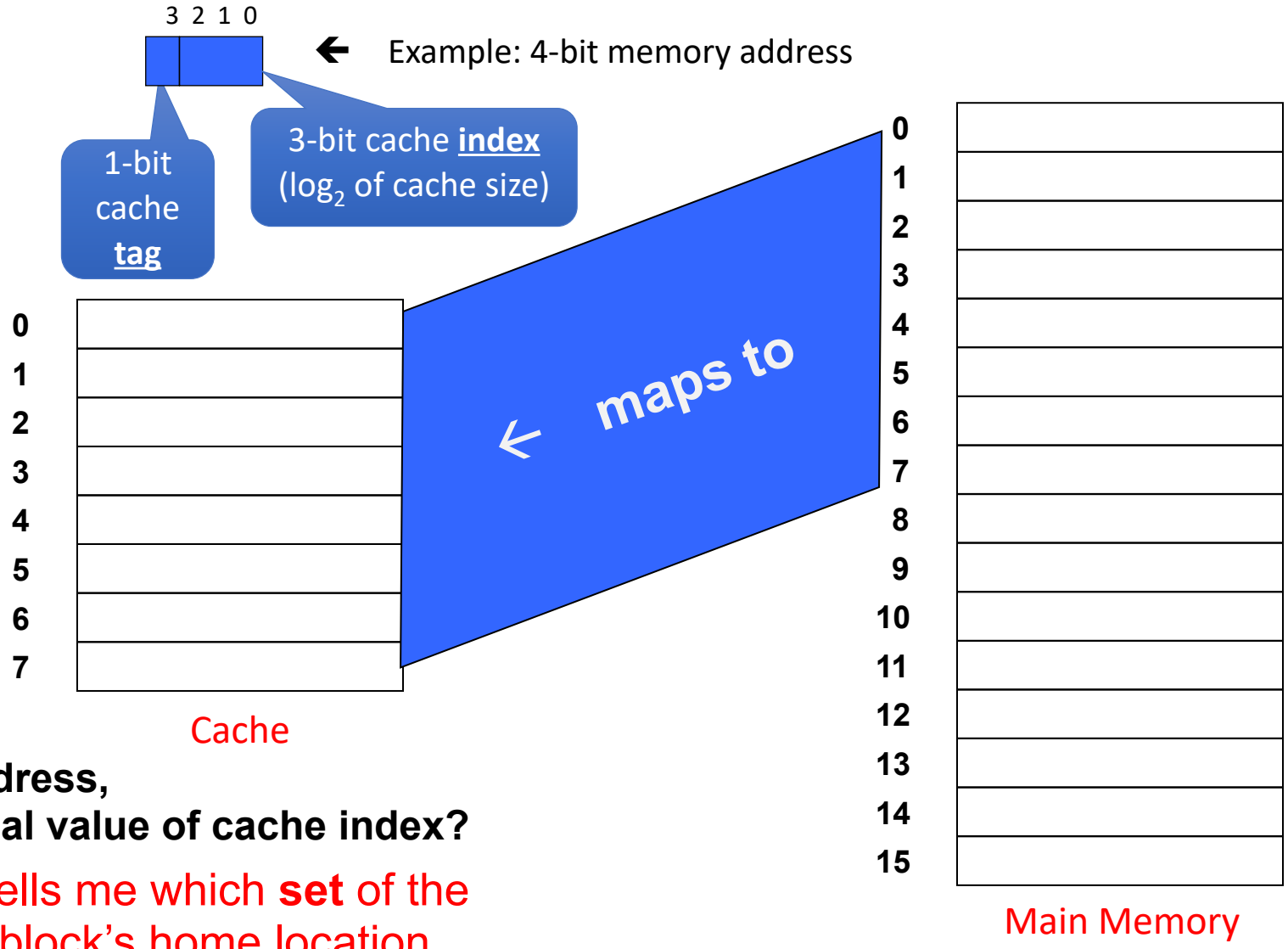
Main Memory

Given a memory address,
What is the numerical value of cache index?

An address' index tells me which **set** of the cache is the cache block's home location

Direct mapped cache

For now we will assume:
1. word-addressable memory
2. each cache block entry is a word

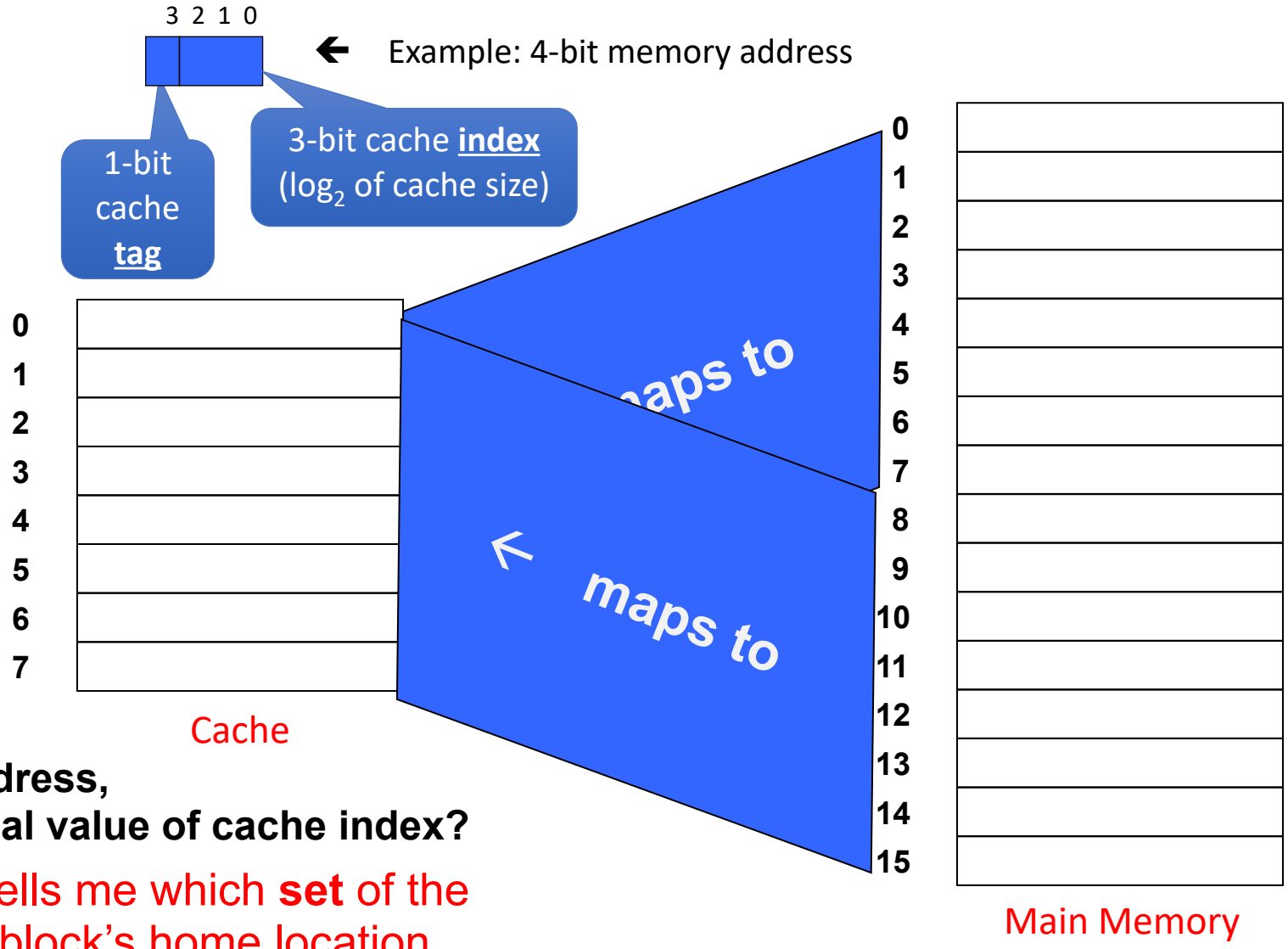


Given a memory address,
What is the numerical value of cache index?

An address' index tells me which **set** of the cache is the cache block's home location

Direct mapped cache

For now we will assume:
1. word-addressable memory
2. each cache block entry is a word

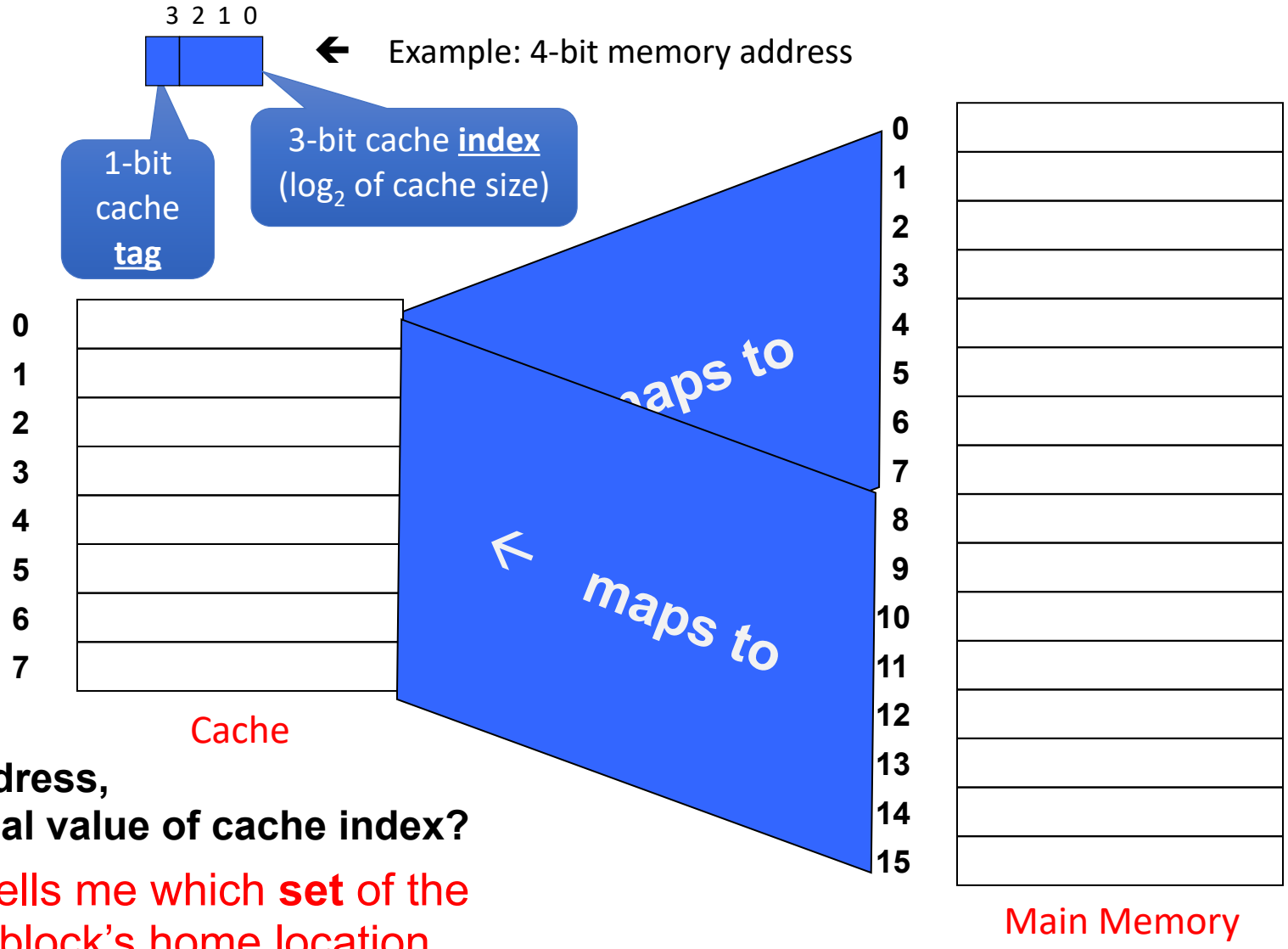


Given a memory address,
What is the numerical value of cache index?

An address' index tells me which **set** of the cache is the cache block's home location

Direct mapped cache

For now we will assume:
1. word-addressable memory
2. each cache block entry is a word

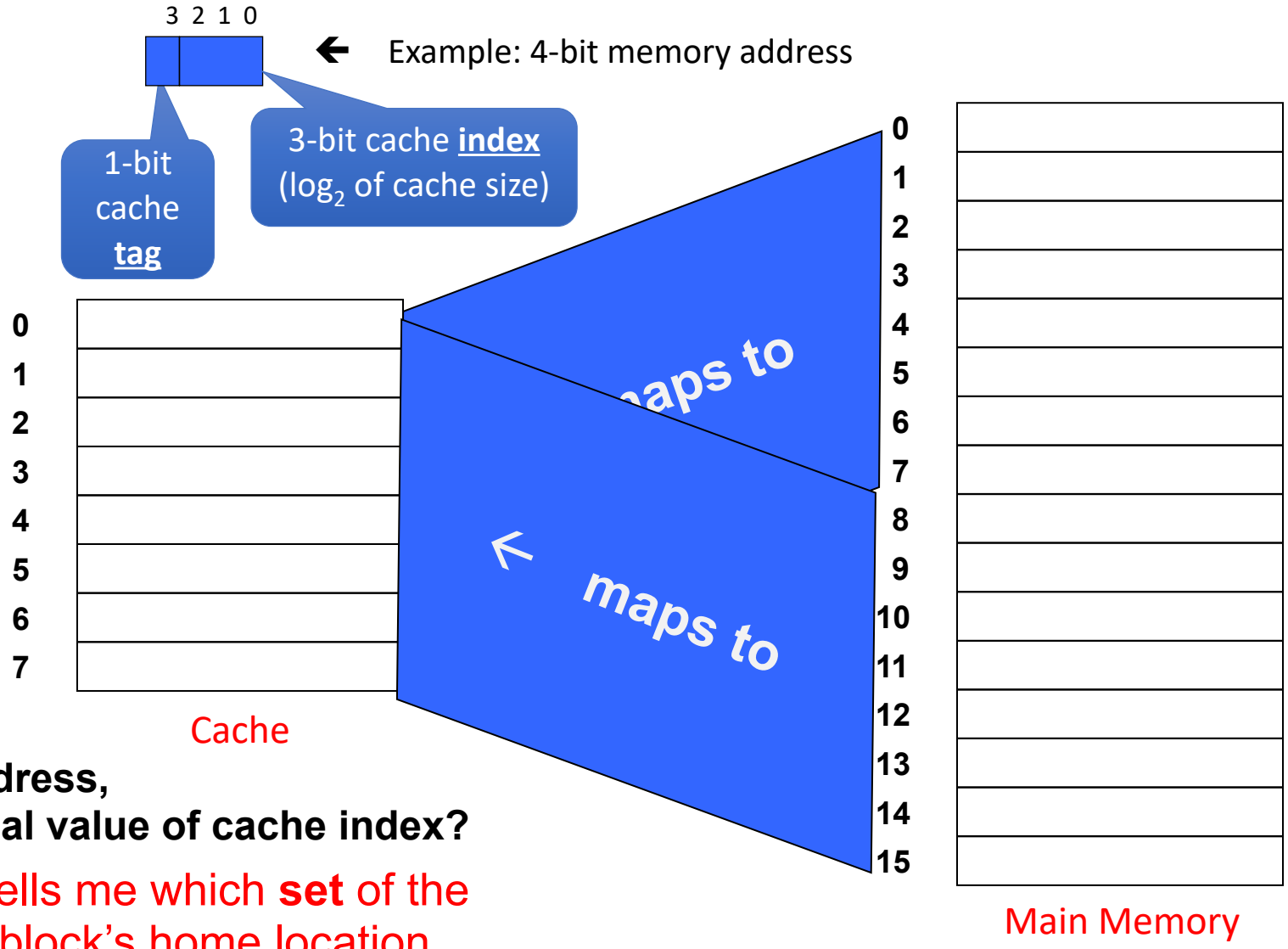


Given a memory address,
What is the numerical value of cache index?

An address' index tells me which **set** of the cache is the cache block's home location

Direct mapped cache

For now we will assume:
1. word-addressable memory
2. each cache block entry is a word



Given a memory address,
What is the numerical value of cache index?

An address' index tells me which **set** of the cache is the cache block's home location

Types of misses

Types of misses

- Compulsory
  - first time an address is requested

Types of misses

- Compulsory
  - first time an address is requested
- Capacity
  - block used to be in cache but was removed because cache got full

Types of misses

- Compulsory
  - first time an address is requested
- Capacity
  - block used to be in cache but was removed because cache got full
- Conflict
  - block used to be in cache but was removed because target cache set got full

Types of misses

- Compulsory
  - first time an address is requested
- Capacity
  - block used to be in cache but was removed because cache got full
- Conflict
  - block used to be in cache but was removed because target cache set got full

Known as the 3 C's!

Types of misses

- Compulsory
  - first time an address is requested
- Capacity
  - block used to be in cache but was removed because cache got full
- Conflict
  - block used to be in cache but was removed because target cache set got full

Known as the 3 C's!

Types of misses

- Compulsory
  - first time an address is requested
- Capacity
  - block used to be in cache but was removed because cache got full
- Conflict
  - block used to be in cache but was removed because target cache set got full

Known as the 3 C's!

Types of misses

- Compulsory
  - first time an address is requested
- Capacity
  - block used to be in cache but was removed because cache got full
- Conflict
  - block used to be in cache but was removed because target cache set got full

Known as the 3 C's!

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses

0	mem loc 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses

These were compulsory misses.

The data were referenced for the first time.

0	mem loc 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits

These were compulsory misses.

The data were referenced for the first time.

0	mem loc 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

These were compulsory misses.

The data were referenced for the first time.

0	mem loc 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

These were compulsory misses.

The data were referenced for the first time.

0	mem loc ~~0~~
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

0	mem loc ~~0~~ 8
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

This is also a compulsory miss.

There are two memory addresses mapping to the same cache entry. Must replace the old one (0)

0	mem loc ~~0~~ 8
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

This is now a conflict miss. Cache block 0 used to be in the cache, but was replaced because the previous access to block 8 maps to the same set

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits

miss

3 2 1 0

	9

4-bit memory address

	Cache
0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

	Memory
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits

miss

3 2 1 0

9

9 = 1 0 0 1

4-bit memory address

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits

miss

3 2 1 0

9

9 = 1 0 0 1

4-bit memory address

3-bit index

9 MOD 8 is 1

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits

miss

3 2 1 0

9

9 = 1 0 0 1

4-bit memory address

3-bit index

9 MOD 8 is 1

index = memory_address *mod* cache_size

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

misses hits miss

3 2 1 0

	9

9 = 1 0 0 1

4-bit memory address

3-bit index

9 MOD 8 is 1

0	mem loc ~~0~~ ~~8~~ 0
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

index = memory_address *mod* cache_size

*** For a direct-mapped cache!**

How do we disambiguate data?

	data
0	mem loc ~~0~~ 8
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

How do we disambiguate data?

We use the part of the memory address that was *not* used as cache index as the tag to label the data in the cache

	data
0	mem loc ~~0~~ 8
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

How do we disambiguate data?

We use the part of the memory address that was *not* used as cache index as the tag to label the data in the cache

	tag	data
0	1	mem loc ~~0~~ 8
1	0	mem loc 1
2	0	mem loc 2
3	0	mem loc 3
4		empty
5		empty
6		empty
7		empty

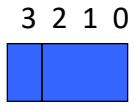
Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

How do we disambiguate data?

We use the part of the memory address that was *not* used as cache index as the tag to label the data in the cache



← 4-bit memory address

	tag	data
0	1	mem loc ~~0~~ 8
1	0	mem loc 1
2	0	mem loc 2
3	0	mem loc 3
4		empty
5		empty
6		empty
7		empty

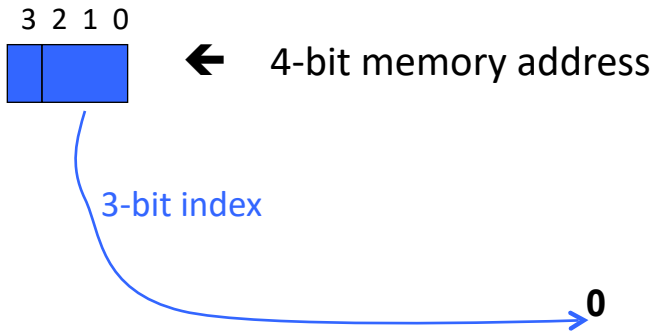
Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

How do we disambiguate data?

We use the part of the memory address that was *not* used as cache index as the tag to label the data in the cache



	tag	data
0	1	mem loc ~~0~~ 8
1	0	mem loc 1
2	0	mem loc 2
3	0	mem loc 3
4		empty
5		empty
6		empty
7		empty

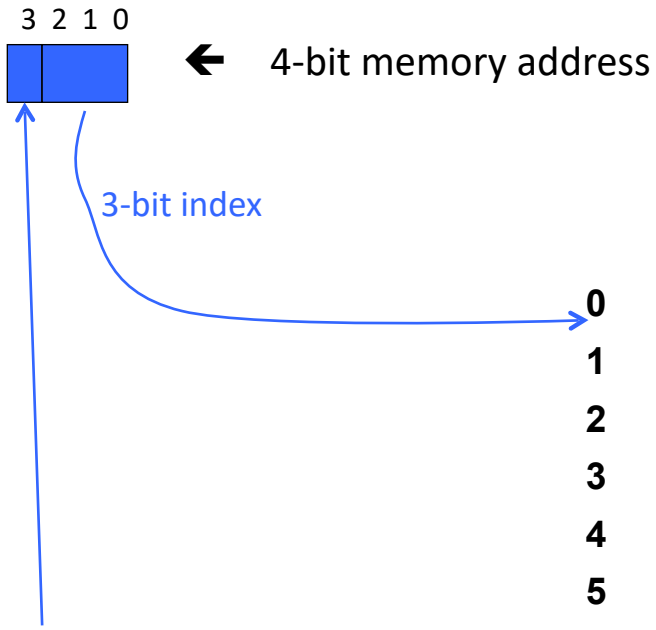
Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

How do we disambiguate data?

We use the part of the memory address that was *not* used as cache index as the tag to label the data in the cache



This remaining bit determines if data in cache set 0 corresponds to memory address 0 or 8

	tag	data
0	1	mem loc ~~0~~ 8
1	0	mem loc 1
2	0	mem loc 2
3	0	mem loc 3
4		empty
5		empty
6		empty
7		empty

Cache

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

Memory

How do we know data is valid?

- At power-up, a cache may contain garbage!
- Tags help disambiguate, not validate

	tag	data
0	1	loc 8
1	0	loc 1
2	0	loc 2
3	0	loc 3
4	X	empty
5	X	empty
6	X	empty
7	X	empty

How do we know data is valid?

- At power-up, a cache may contain garbage!
- Tags help disambiguate, not validate

	valid	tag	data
0	1	1	loc 8
1	1	0	loc 1
2	1	0	loc 2
3	1	0	loc 3
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

How do we know data is valid?

- At power-up, a cache may contain garbage!
- Tags help disambiguate, not validate

	valid	tag	data
0	1	1	loc 8
1	1	0	loc 1
2	1	0	loc 2
3	1	0	loc 3
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Fields in a Direct Mapped Cache

Valid	Tag	Data

How do we know data is valid?

- At power-up, a cache may contain garbage!
- Tags help disambiguate, not validate

	valid	tag	data
0	1	1	loc 8
1	1	0	loc 1
2	1	0	loc 2
3	1	0	loc 3
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Fields in a Direct Mapped Cache

Valid	Tag	Data

Metadata

Real data

Interpreting memory addresses

Memory address

Cache Tag	Cache Index

- index = memory addr *mod* cache size
  - MemAddr = 8 ➔ index = 0 * Assuming previous
  - MemAddr = 0 ➔ index = 0 example of 8 cache sets
  - The tag bits will help decide if the block currently present in set 0 is address 0 or 8 !

- number of tag bits = memory addr size - cache index size

Session ID: dag2200

What does the cache entry contain?

You have a 64-entry direct-mapped cache with 16-bit word addresses and word-sized (16-bit) cache blocks.

30% A. 1 bit valid flag, 6 bit tag, 10 bit data

43% B. 1 bit valid flag, 10 bit tag, 16 bit data

24% C. 1 bit valid flag, 10 bit tag, 6 bit data

4% D. 2 bit valid flag, 12 bit tag, 16 bit data

Interpreting memory addresses

Memory address

Cache Tag	Cache Index

- index = memory addr *mod* cache size
  - MemAddr = 8 ➔ index = 0
  - MemAddr = 0 ➔ index = 0
  - The tag bits will help decide if the block currently present in set 0 is address 0 or 8 !

- number of tag bits = memory addr size - cache index size

Why not the other way around?

➔ use the low-order bits for cache tag?

Index first, tag last?

Address:

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

index tag

Index first, tag last?

Address:

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

index tag

	valid	tag	data
0	1	0	loc 0
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 0

Index first, tag last?

Address:

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

index tag

	valid	tag	data
0	1	0	loc 0
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 0

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 1

Index first, tag last?

Address:

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

index tag

	valid	tag	data
0	1	0	loc 0
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 0

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 1

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	1	0	loc 2
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 2

Index first, tag last?

Address:

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

index tag

	valid	tag	data
0	1	0	loc 0
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 0

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 1

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	1	0	loc 2
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 2

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	1	loc ~~2~~ 3
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 3

Index first, tag last?

Address:

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

index tag

	valid	tag	data
0	1	0	loc 0
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 0

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 1

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	1	0	loc 2
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 2

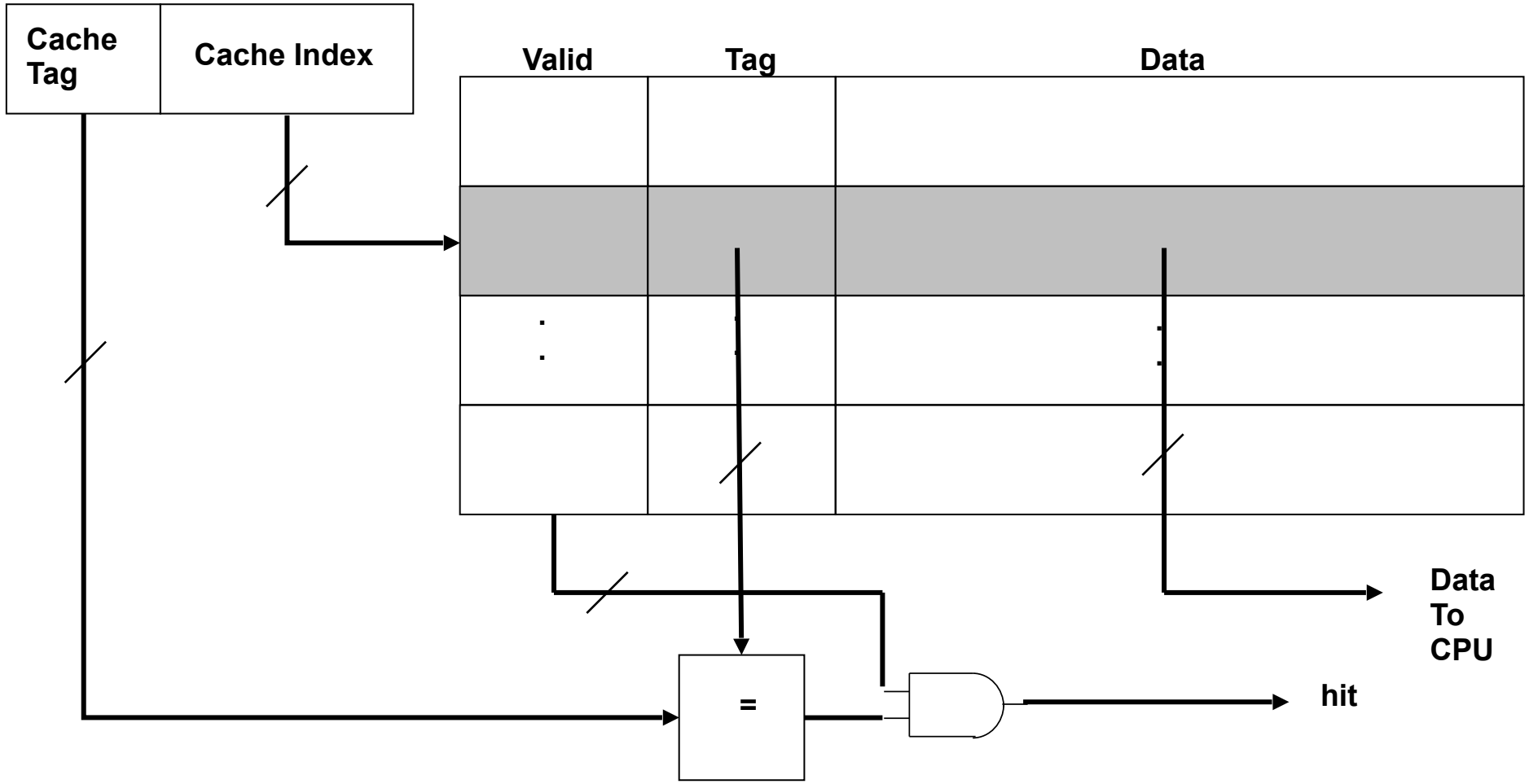
	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	1	loc ~~2~~ 3
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access address 3

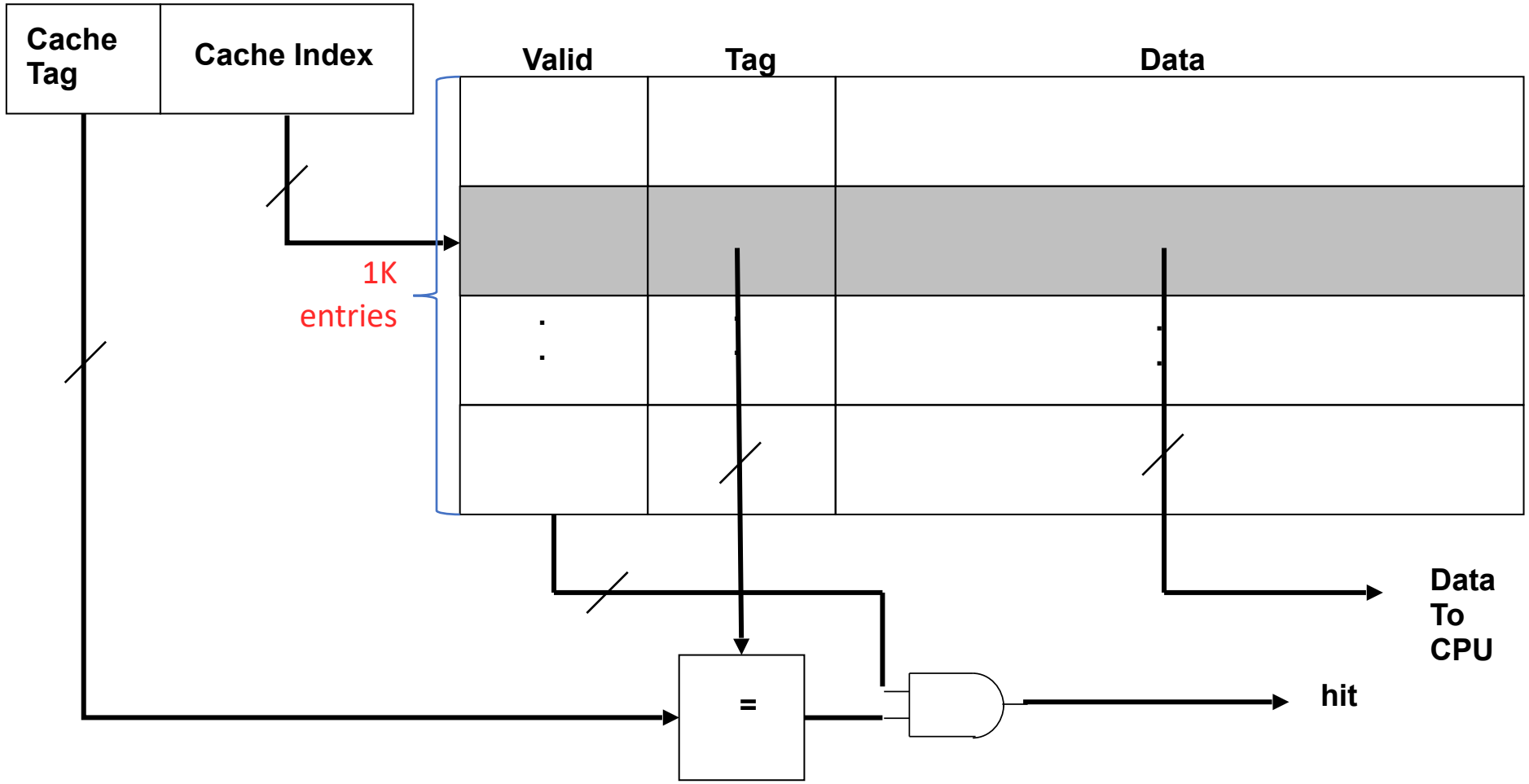
Cache occupancy if we switch index and tag is BAD!!

➔ loss of spatial locality

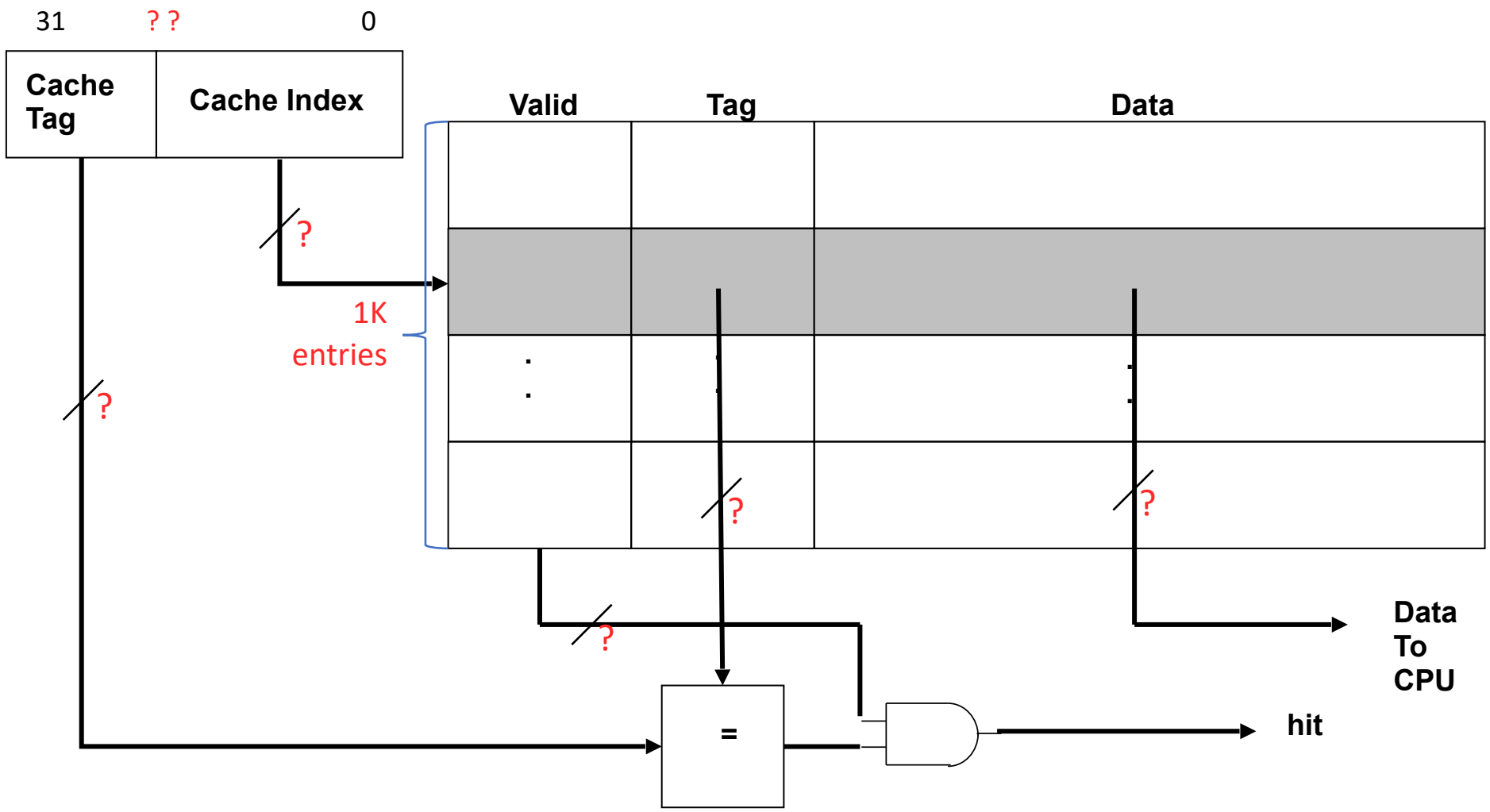
Hardware



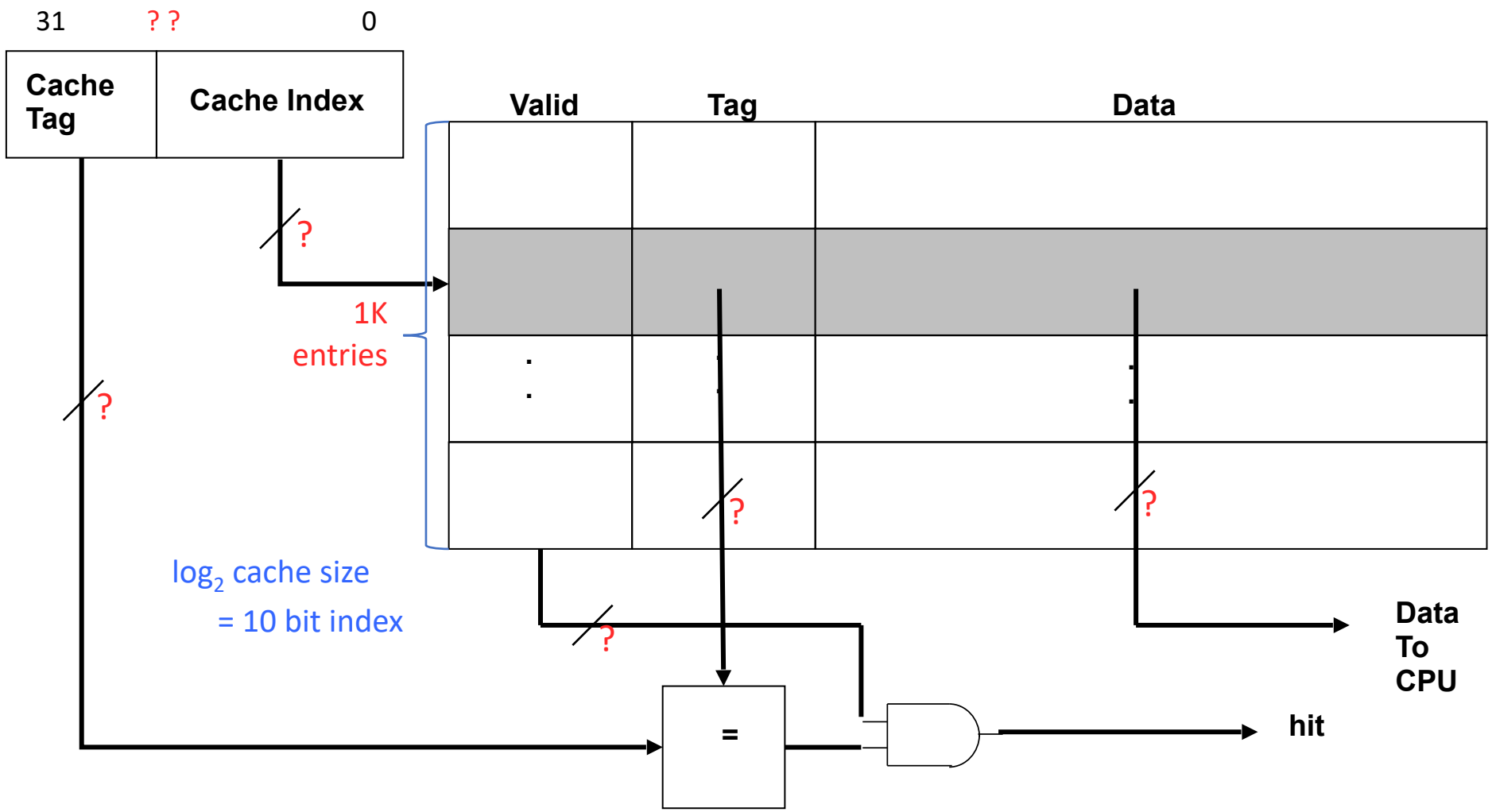
Hardware


Cache Tag
Cache Index
Valid
Tag
Data
1K entries
=
Data To CPU
hit

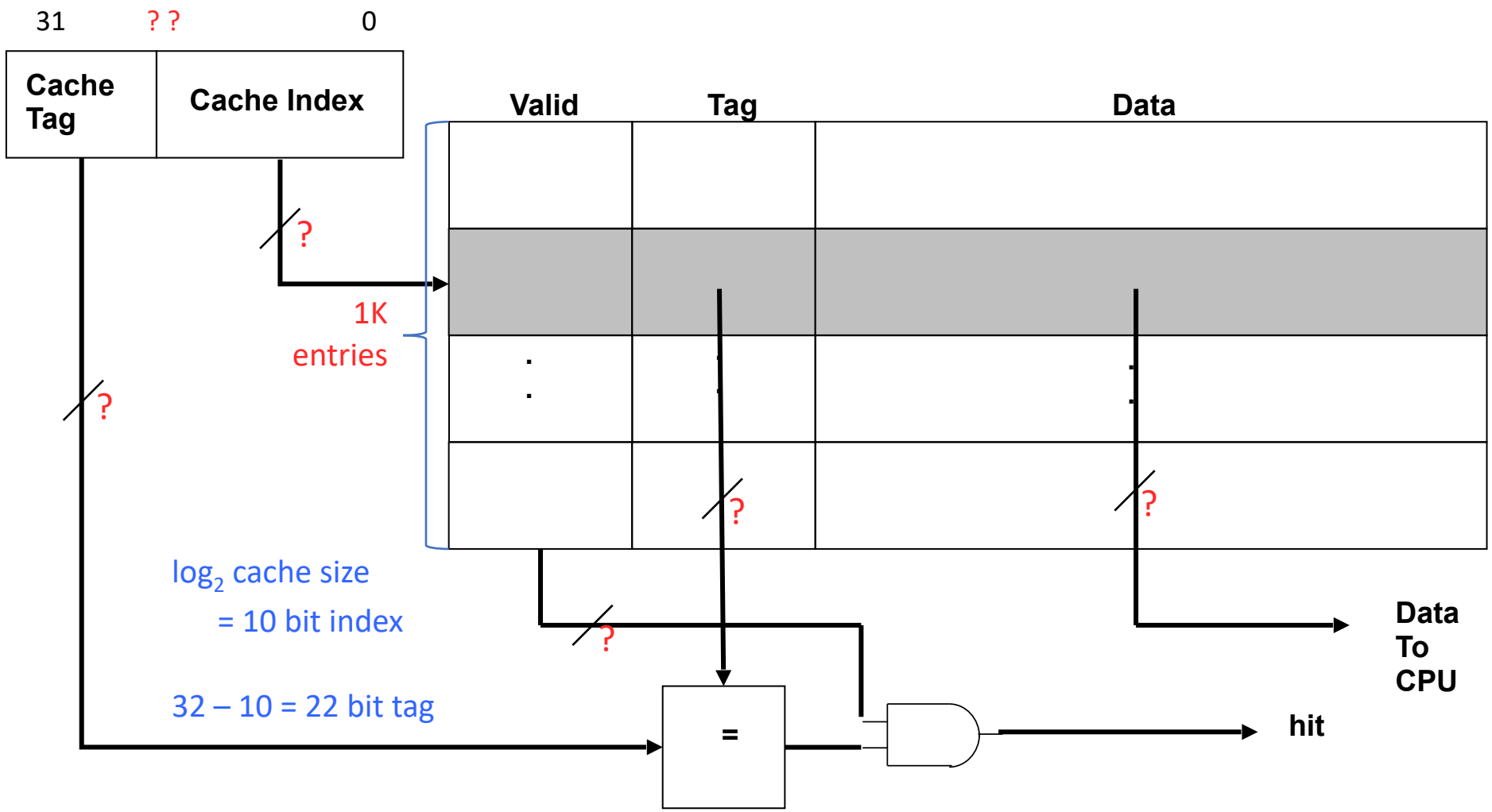
Hardware


31
? ?
0
Cache Tag
Cache Index
Valid
Tag
Data
?
1K entries
?
?
?
?
Data To CPU
=
hit

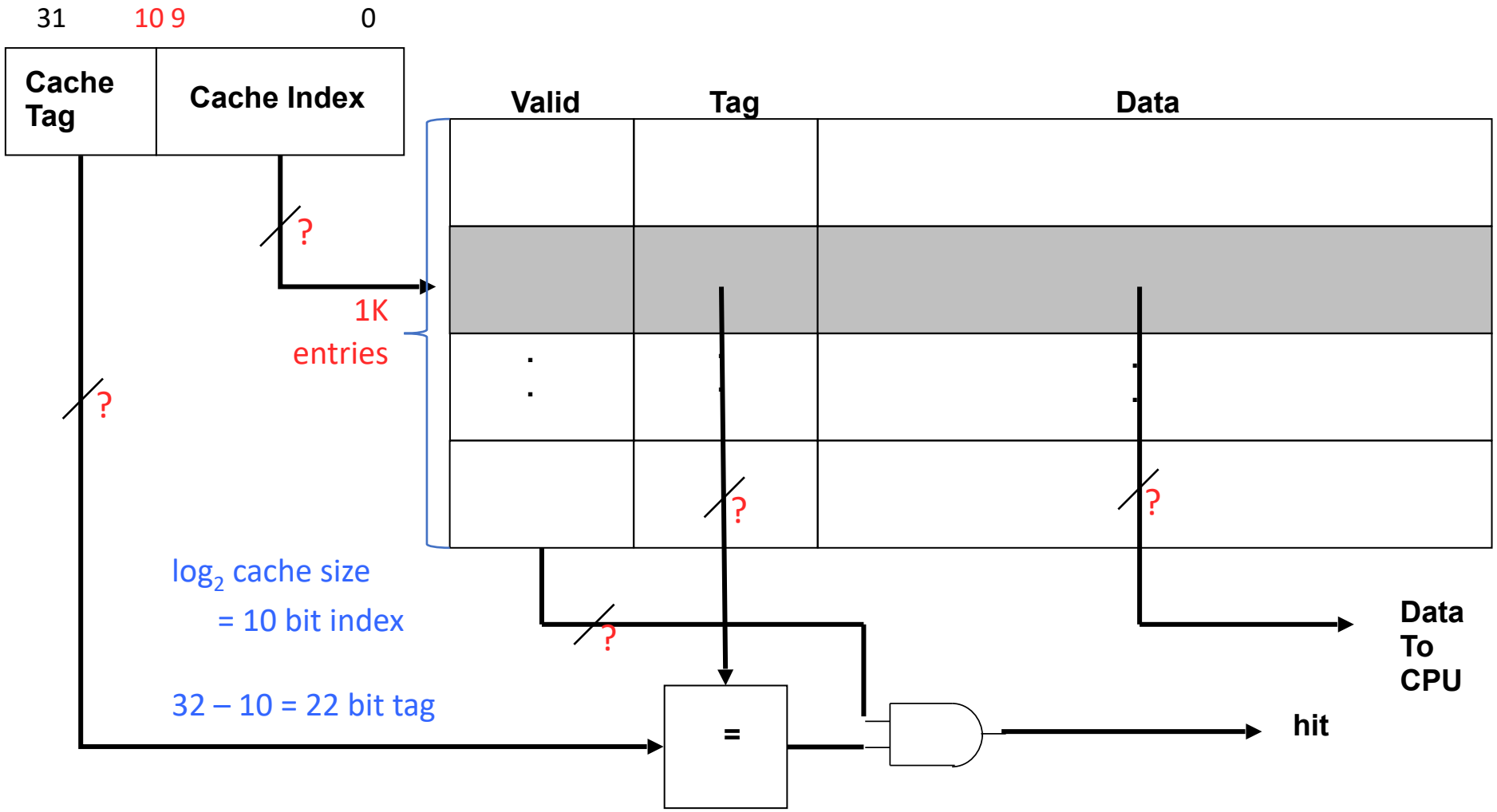
Hardware


31
? ?
0
Cache Tag
Cache Index
Valid
Tag
Data
?
1K entries
?
?
?
?
log2 cache size
= 10 bit index
Data To CPU
=
hit

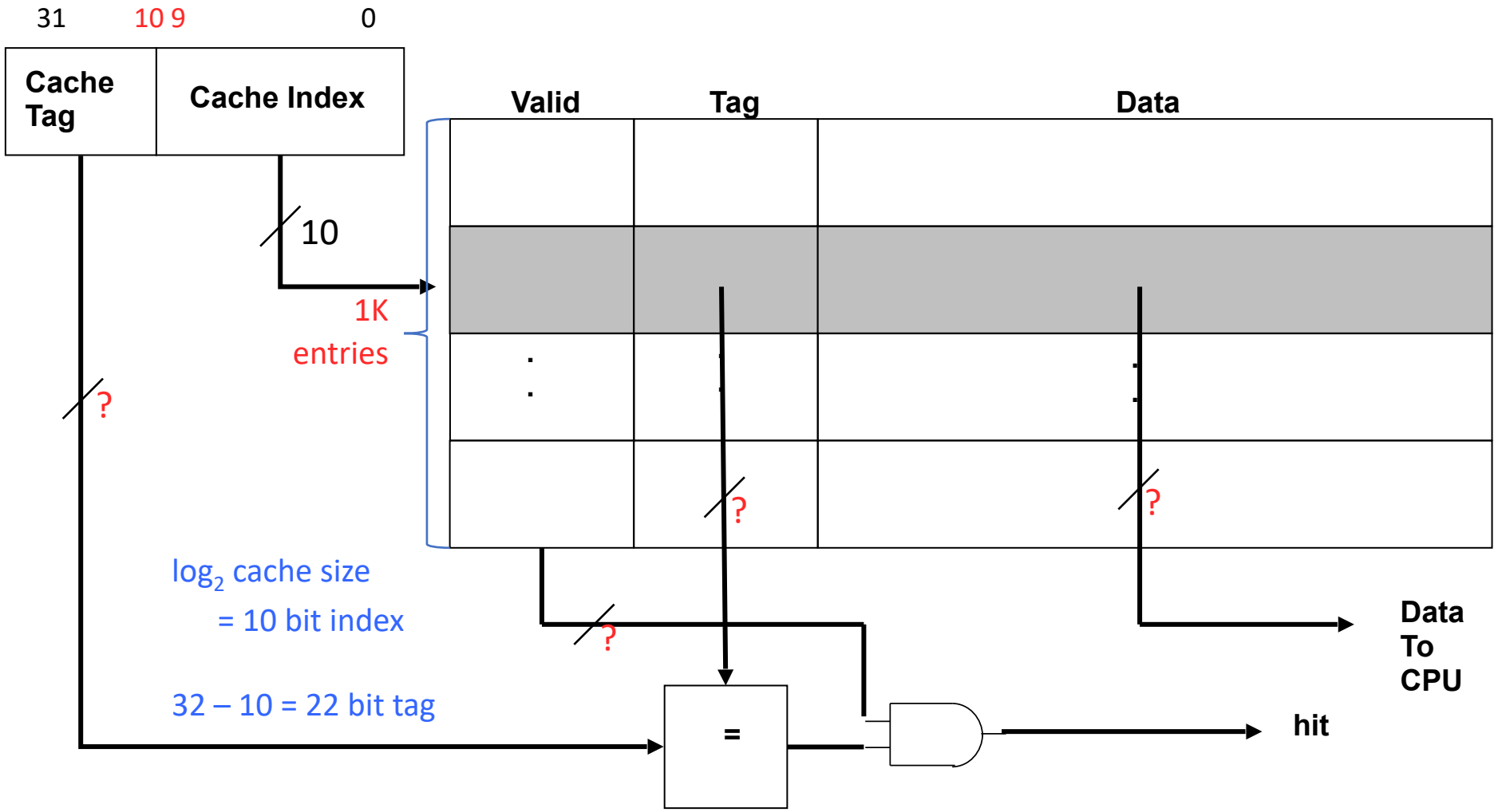
Hardware


31
? ?
0
Cache Tag
Cache Index
Valid
Tag
Data
?
1K entries
?
?
?
?
log2 cache size
= 10 bit index
32 – 10 = 22 bit tag
=
Data To CPU
hit

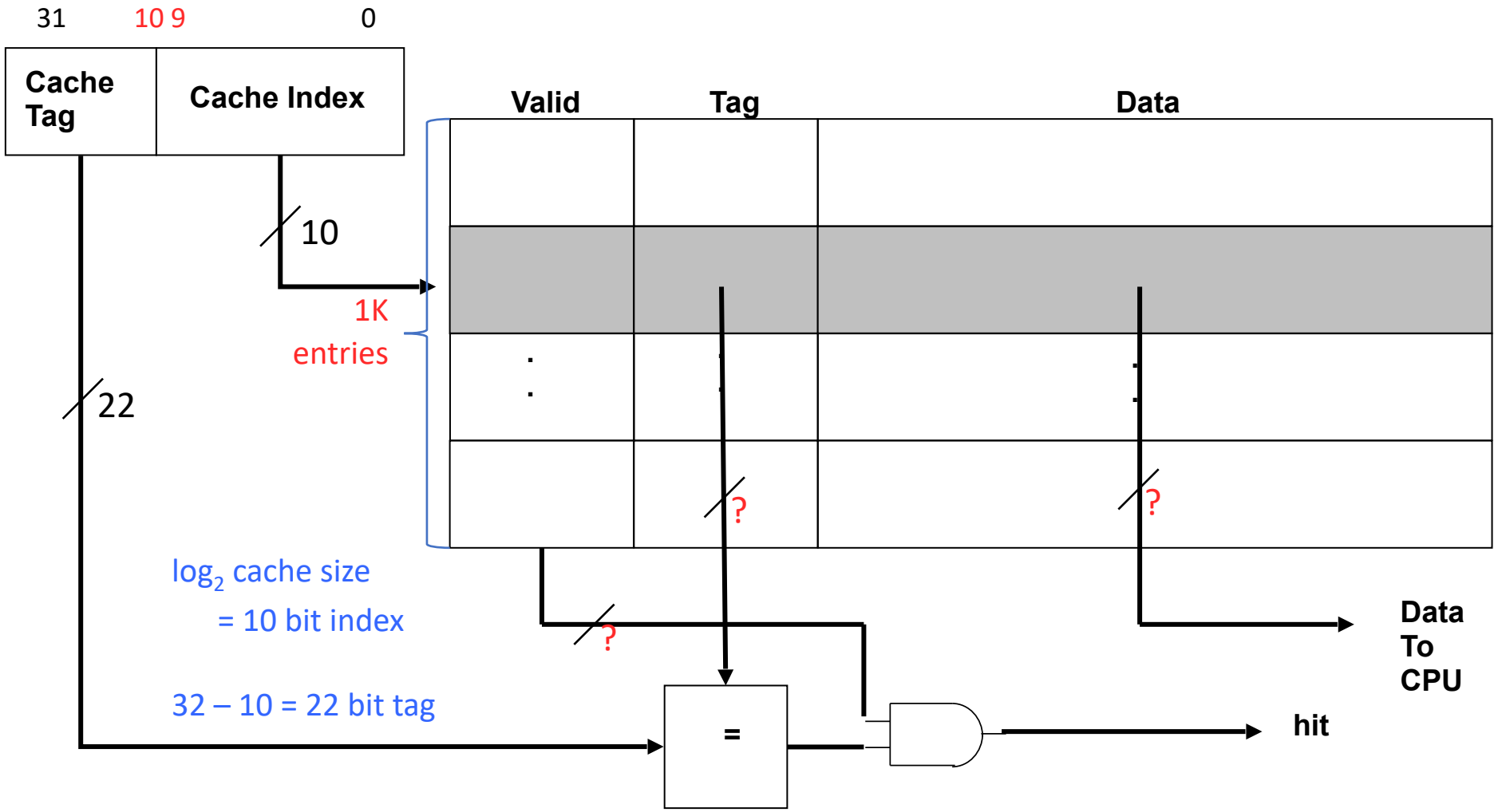
Hardware


31
10 9
0
Cache Tag
Cache Index
Valid
Tag
Data
?
1K entries
?
?
?
?
log2 cache size
= 10 bit index
32 – 10 = 22 bit tag
=
Data To CPU
hit

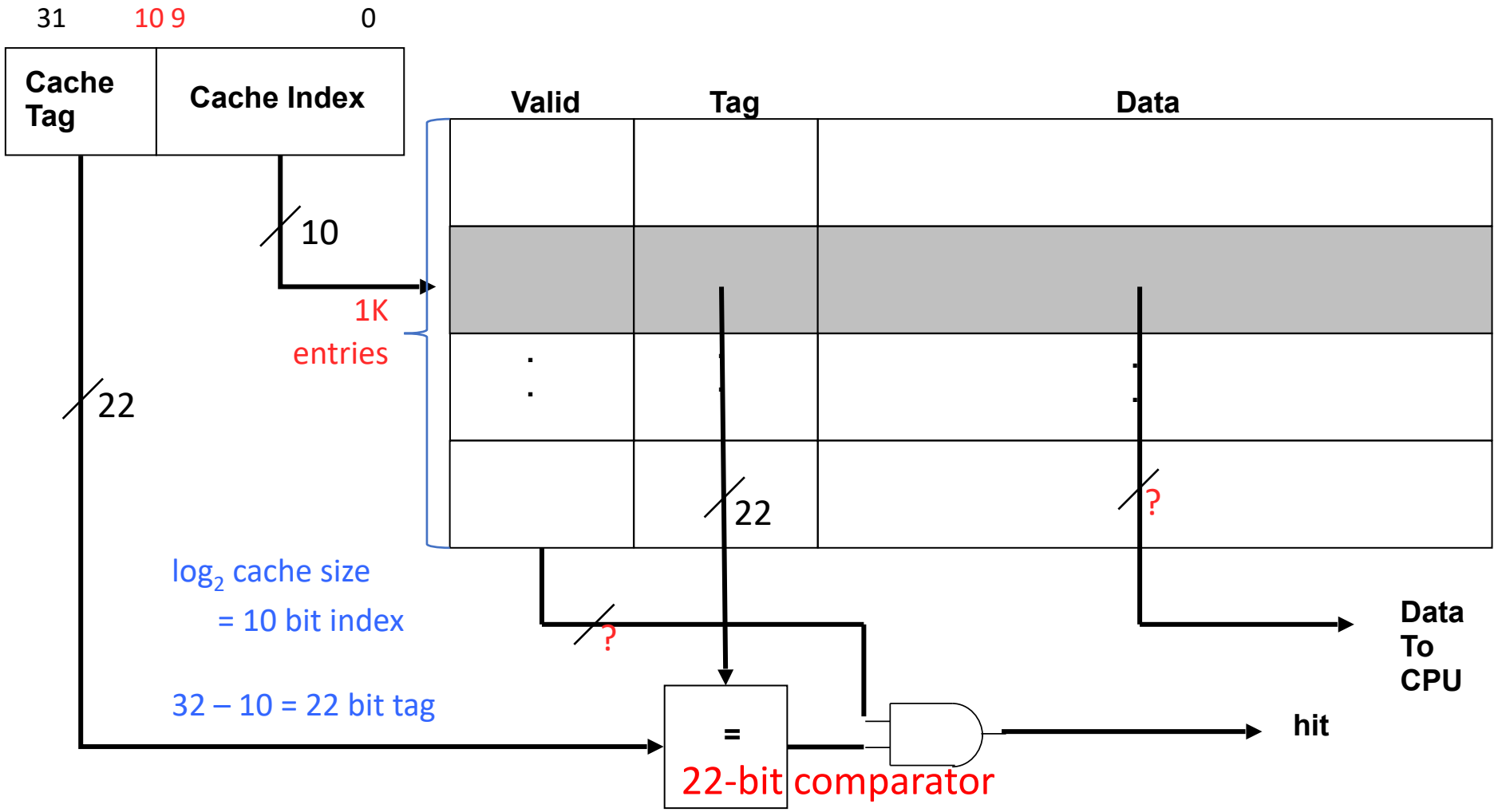
Hardware


31
10 9
0
Cache Tag
Cache Index
Valid
Tag
Data
10
1K entries
?
?
?
?
log2 cache size
= 10 bit index
32 – 10 = 22 bit tag
=
Data To CPU
hit

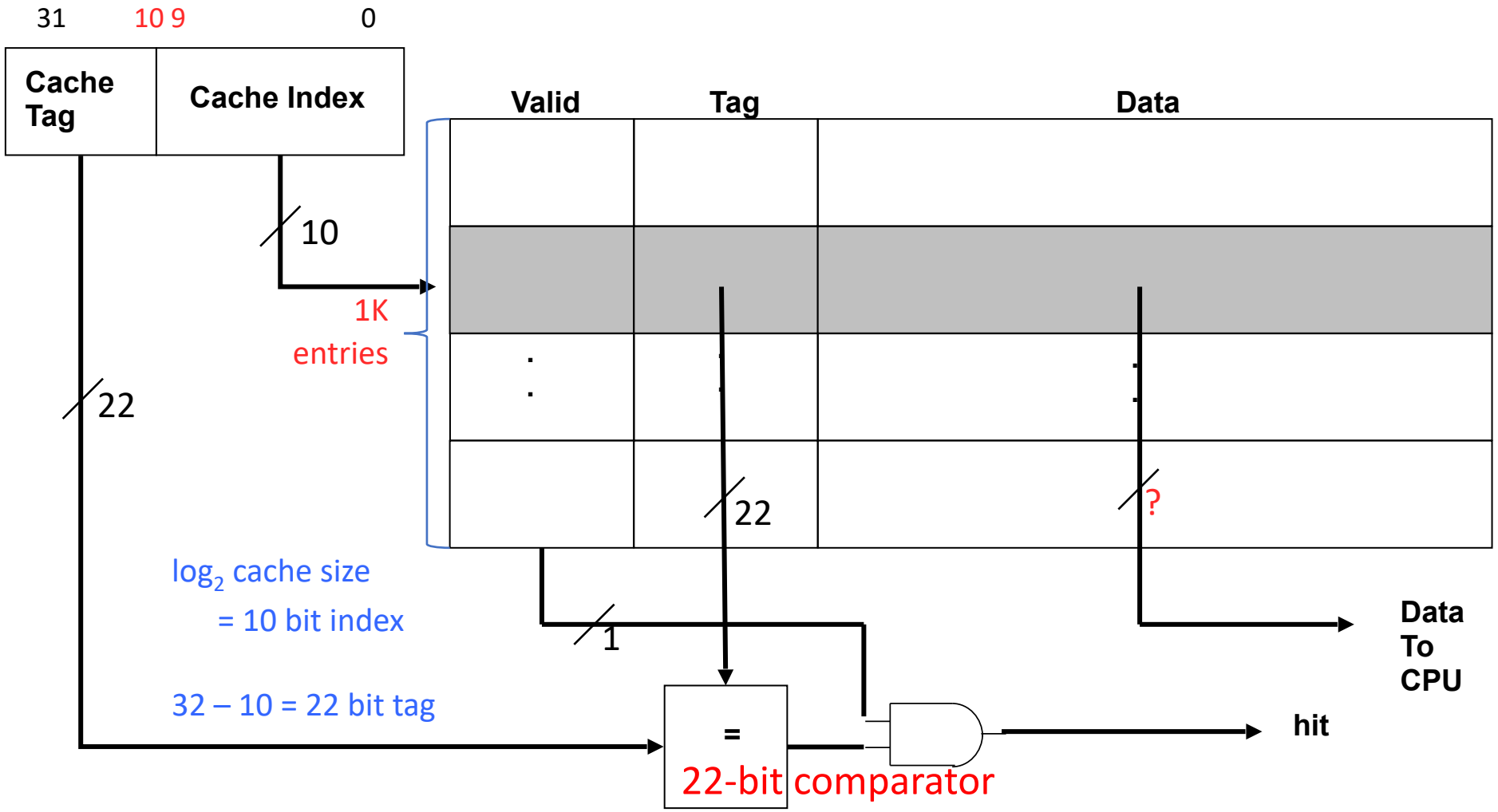
Hardware


31
10 9
0
Cache Tag
Cache Index
Valid
Tag
Data
10
1K entries
22
?
?
?
log2 cache size
= 10 bit index
32 – 10 = 22 bit tag
=
Data To CPU
hit

Hardware


31
10 9
0
Cache Tag
Cache Index
Valid
Tag
Data
10
1K entries
22
22
?
?
log2 cache size
= 10 bit index
32 – 10 = 22 bit tag
=
22-bit comparator
hit
Data To CPU

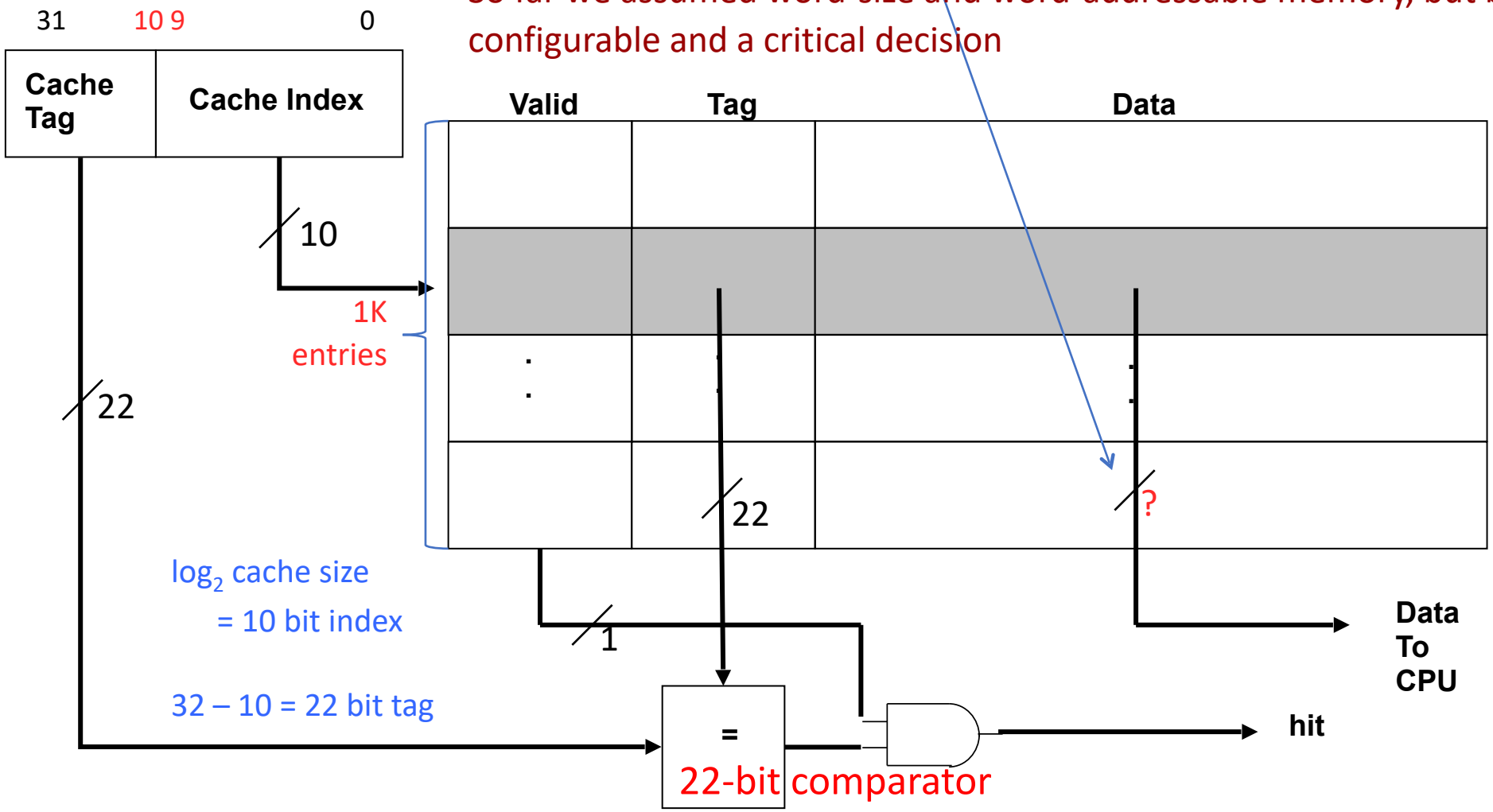
Hardware


31
10 9
0
Cache Tag
Cache Index
Valid
Tag
Data
10
1K entries
22
22
?
1
log2 cache size
= 10 bit index
32 – 10 = 22 bit tag
=
22-bit comparator
hit
Data To CPU

Hardware

What should the data (i.e., cache block size) be?

So far we assumed word-size and word-addressable memory, but block size configurable and a critical decision



$\log_2$ cache size = 10 bit index

32 – 10 = 22 bit tag

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.

Memory address

Cache Tag	Cache Index

Cache

V	Tag	Data
		4 bytes
...	...	...

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.

Memory address

Cache Tag	Cache Index	Byte offset

Cache

V	Tag	Data
		4 bytes
...	...	...

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

Cache

V	Tag	Data
		4 bytes
...	...	...

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Consider only real data
→ $64k/4=2^{14}$ rows in cache

Memory address

Cache Tag	Cache Index	Byte offset

Cache

V	Tag	Data
		4 bytes
…	…	…

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

Consider only real data
→ $64k/4=2^{14}$ rows in cache

Cache

V	Tag	Data
		4 bytes
…	…	…

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

Consider only word address for lookup

Consider only real data → $64k/4=2^{14}$ rows in cache

Cache

V	Tag	Data
		4 bytes
…	…	…

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

Use for lookup

Consider only word address for lookup

Consider only real data → $64k/4=2^{14}$ rows in cache

Cache

V	Tag	Data
		4 bytes
…	…	…

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

31 16 15 2 1 0

-16- -14- -2-

Use for lookup

Consider only word address for lookup

Consider only real data → 64k/4=2^{14} rows in cache

Cache

V	Tag	Data
		4 bytes
…	…	…

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

31 16 15 2 1 0

-16- -14- -2-

Use for lookup

Consider only word address for lookup

Consider only real data → $64k/4=2^{14}$ rows in cache

Metadata (overhead)

Cache

V	Tag	Data
		4 bytes
…	…	…

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

31 16 15 2 1 0

-16- -14- -2-

Use for lookup

Consider only word address for lookup

Consider only real data → 64k/4=2^{14} rows in cache

Metadata (overhead)

Cache

V	Tag	Data
		4 bytes
…	…	…

48 47 32 31 0

-1- -16- -32-

Example

Let us consider the design of a direct-mapped cache for a realistic memory system.

- Assume that the CPU generates a 32-bit byte memory address (i.e., byte-addressable memory).
- Each memory word contains 4 bytes.
- A memory access brings a full word into the cache.
- The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache set containing one word of data.
- Compute the additional storage space needed for the valid bits and the tag fields of the cache.

Memory address

Cache Tag	Cache Index	Byte offset

31 16 15 2 1 0

-16- -14- -2-

Use for lookup

Consider only word address for lookup

Consider only real data → 64k/4=2^{14} rows in cache

Metadata (overhead)

Cache

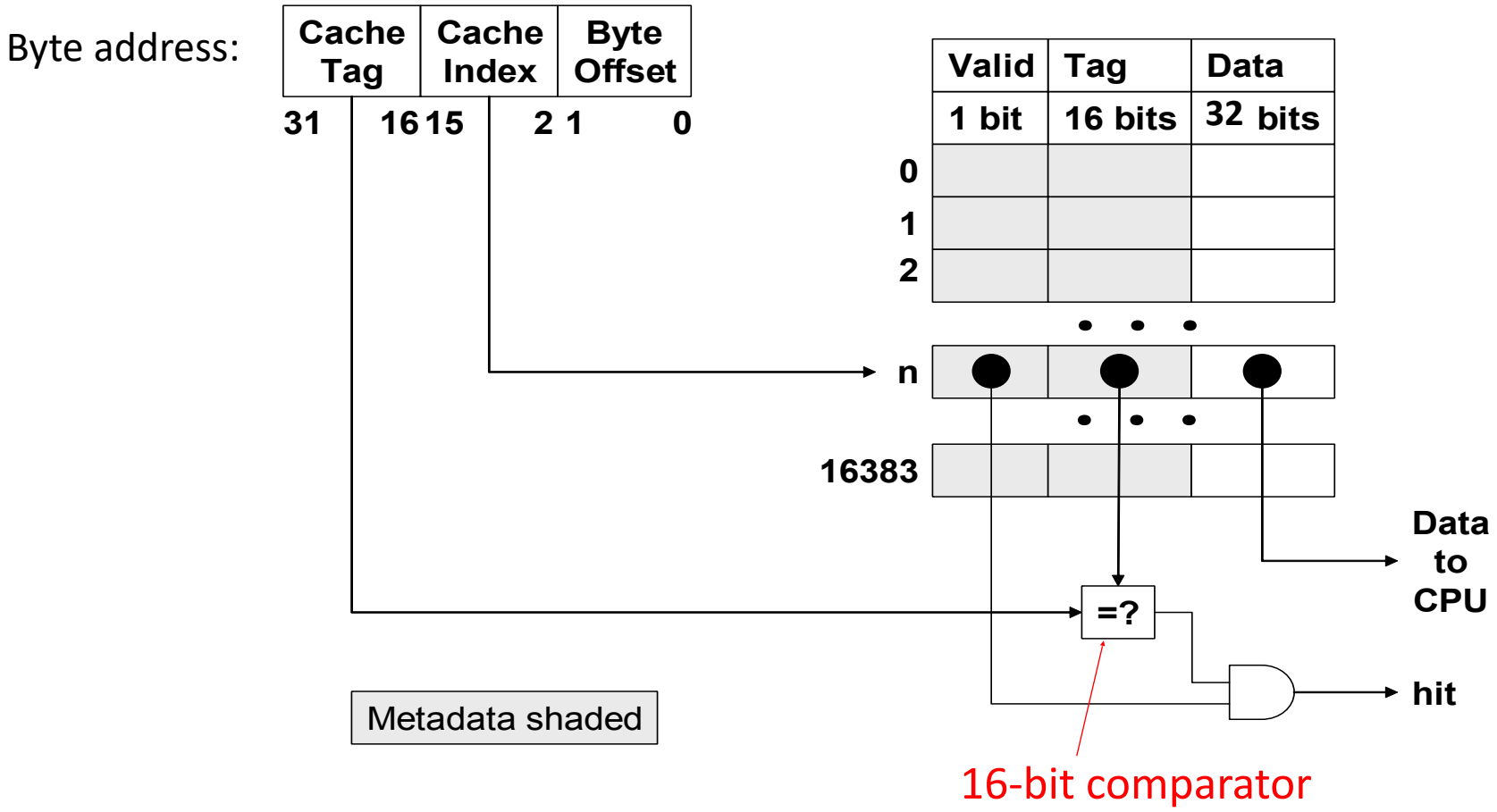
V	Tag	Data
		4 bytes
...	...	...

48 47 32 31 0

-1- -16- -32-

2^{14} * (1 + 16) additional bits for metadata

Memory address interpretation when single cache block contains multiple bytes



Session ID: dag2200

A direct-mapped cache

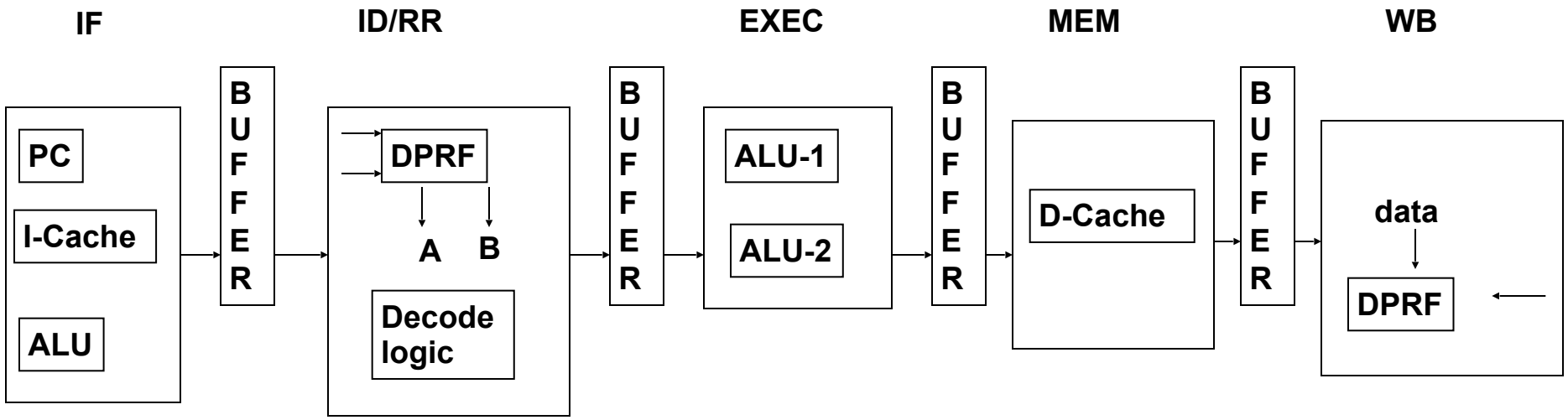
A. Has a many-to-one mapping between memory and cache locations
B. Allows a memory location to be cached wherever there is space in the cache
C. Is so-called because there is a directory associated with the contents of the cache
D. Is usually much smaller than any other type of cache organization

Session ID: dag2200

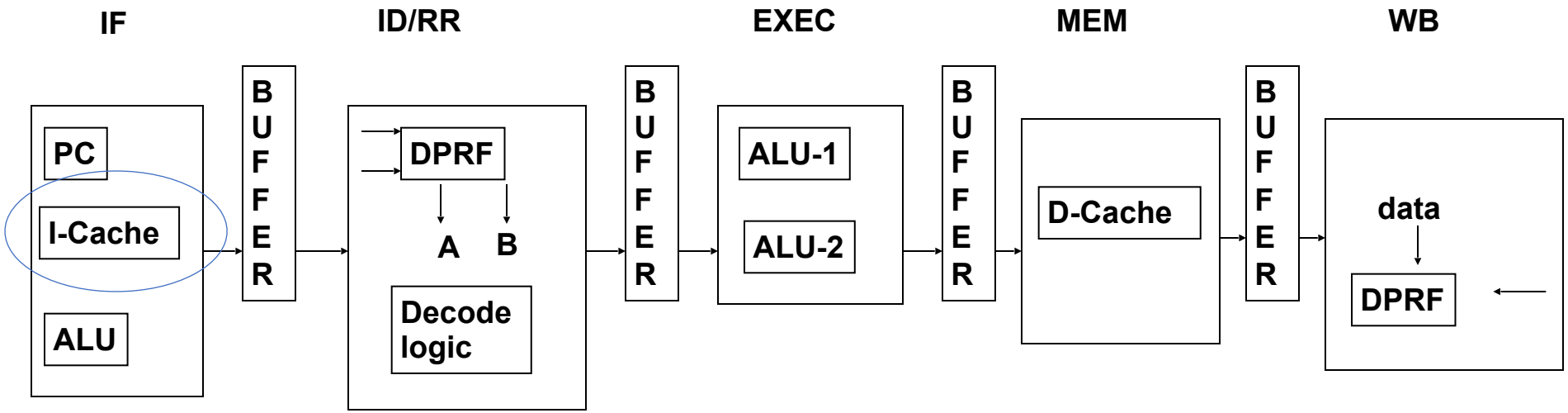
In a direct-mapped cache with a t-bit tag

A. There is one 1-bit tag comparator for each cache block
B. There is one t-bit tag comparator for each cache block
C. There is one 1-bit tag comparator for the entire cache
D. There is one t-bit tag comparator for the entire cache

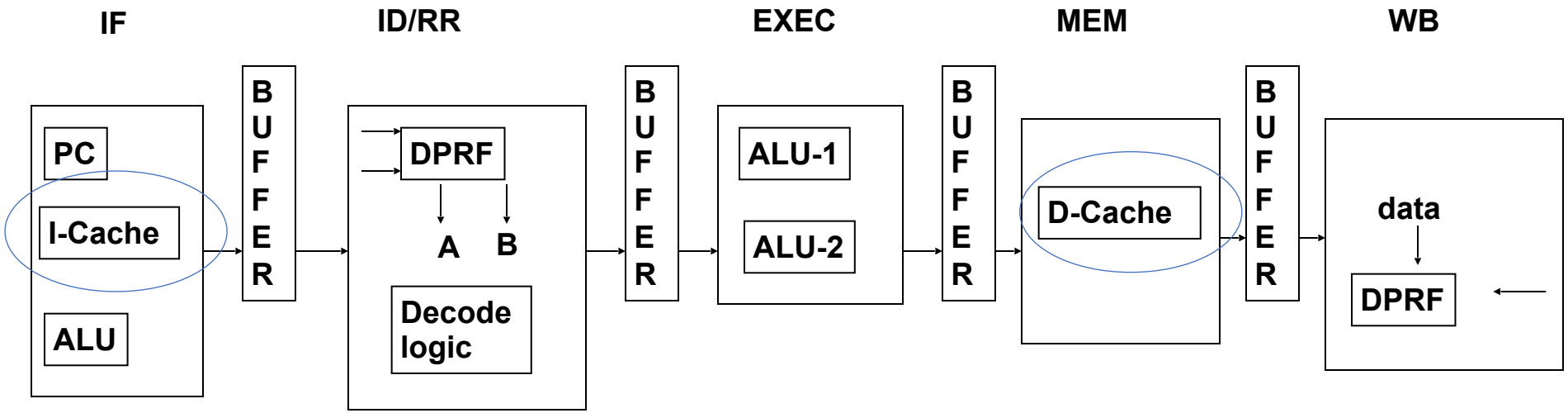
Pipelined processor with caches

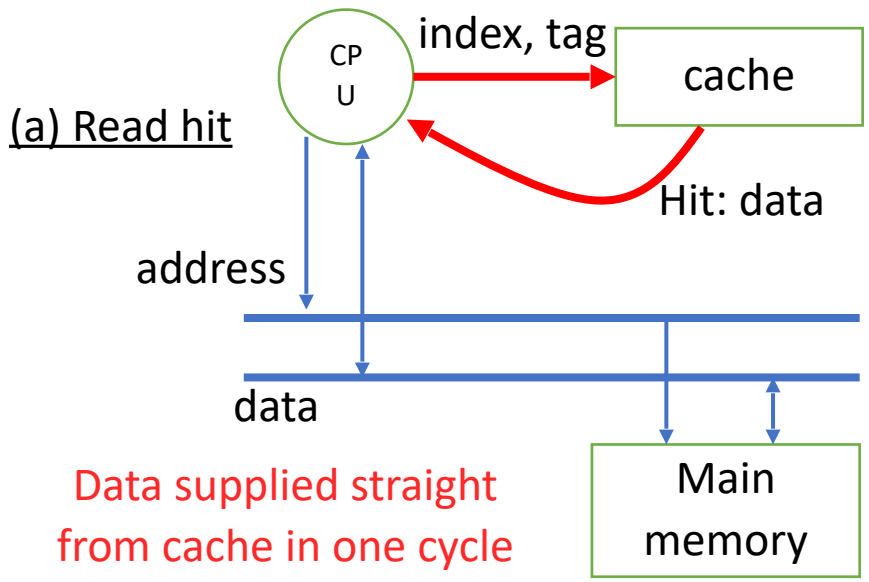


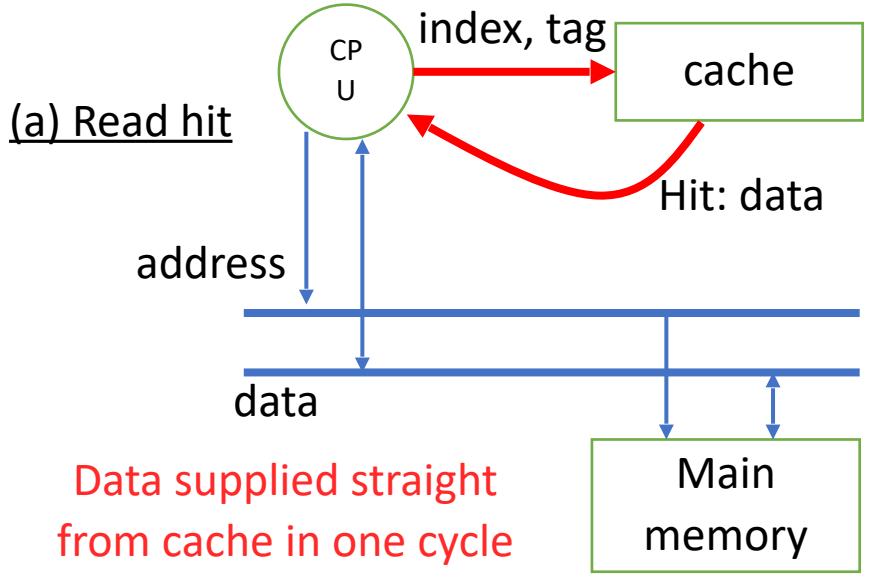
Pipelined processor with caches

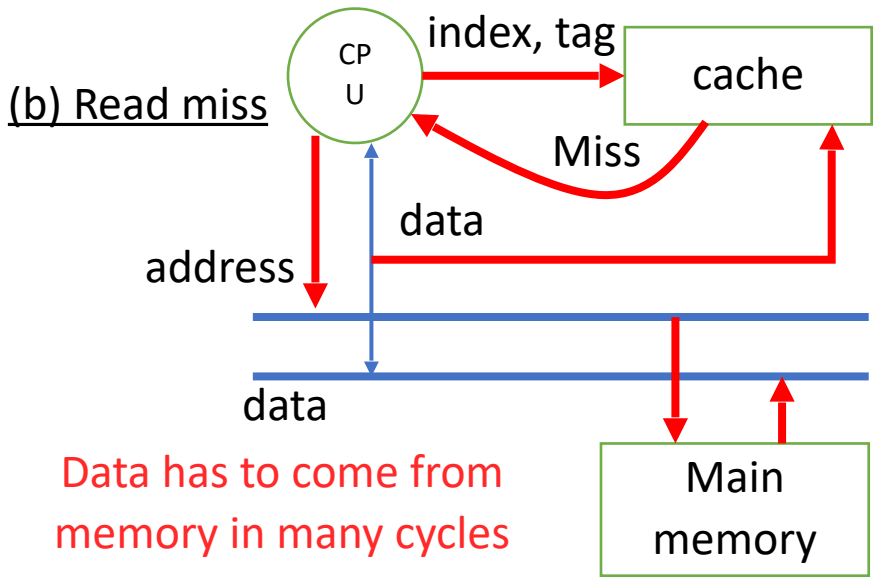


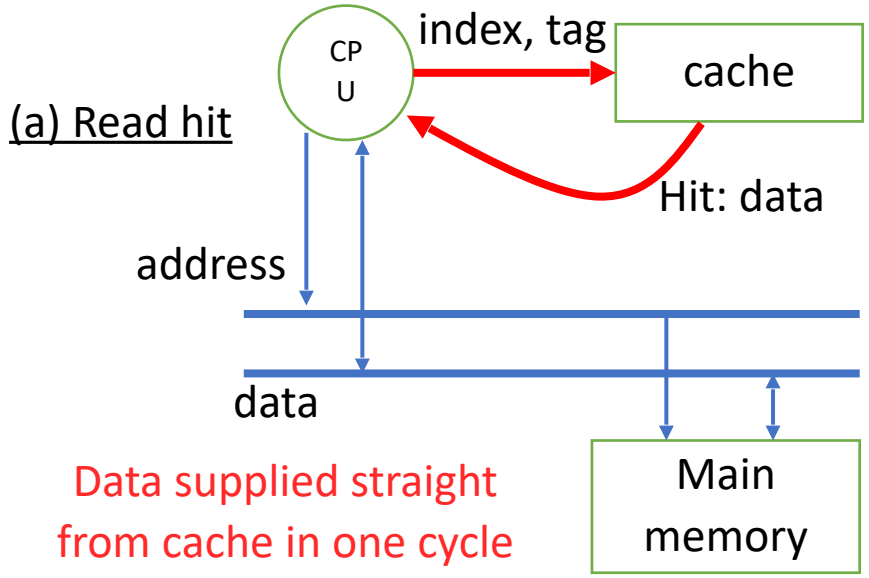
Pipelined processor with caches

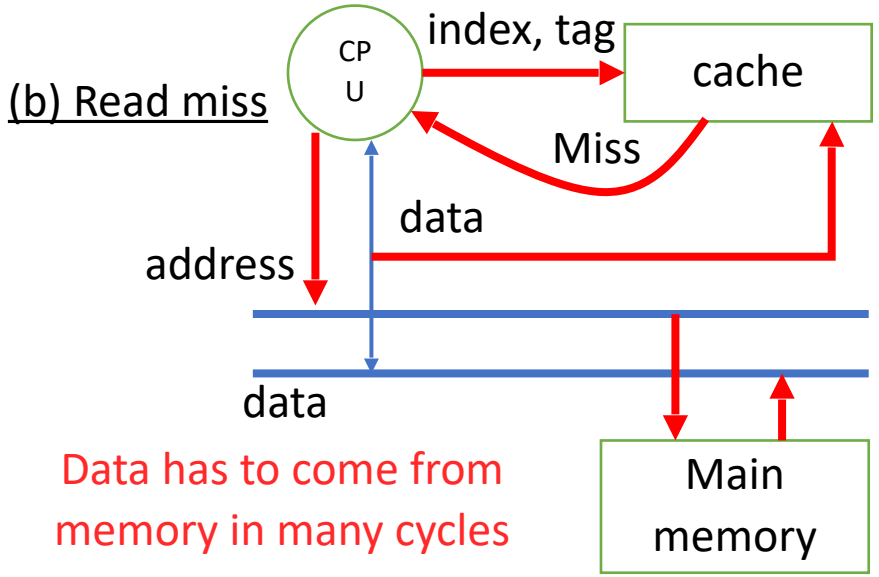


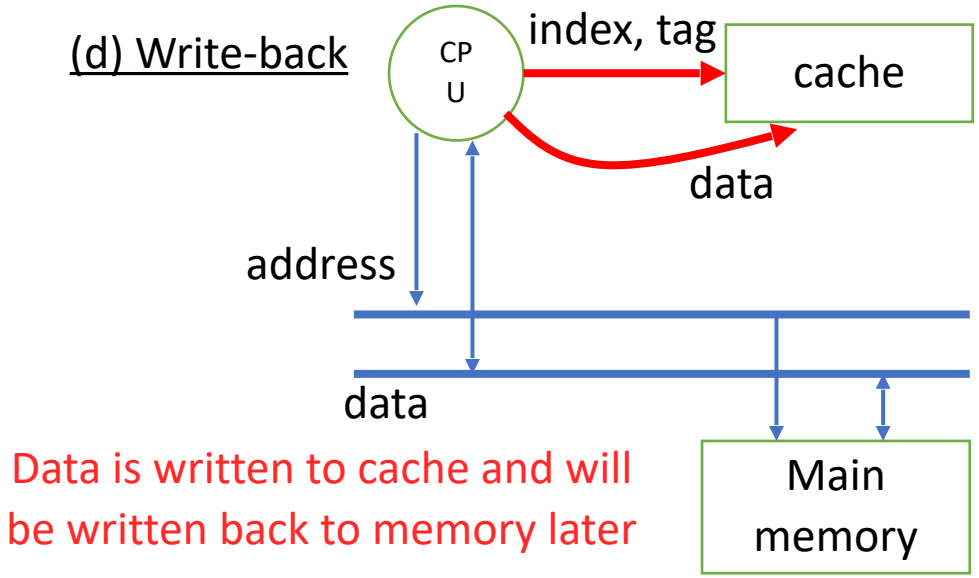

(a) Read hit
CP
U
index, tag
cache
Hit: data
address
data
Main
memory
Data supplied straight
from cache in one cycle

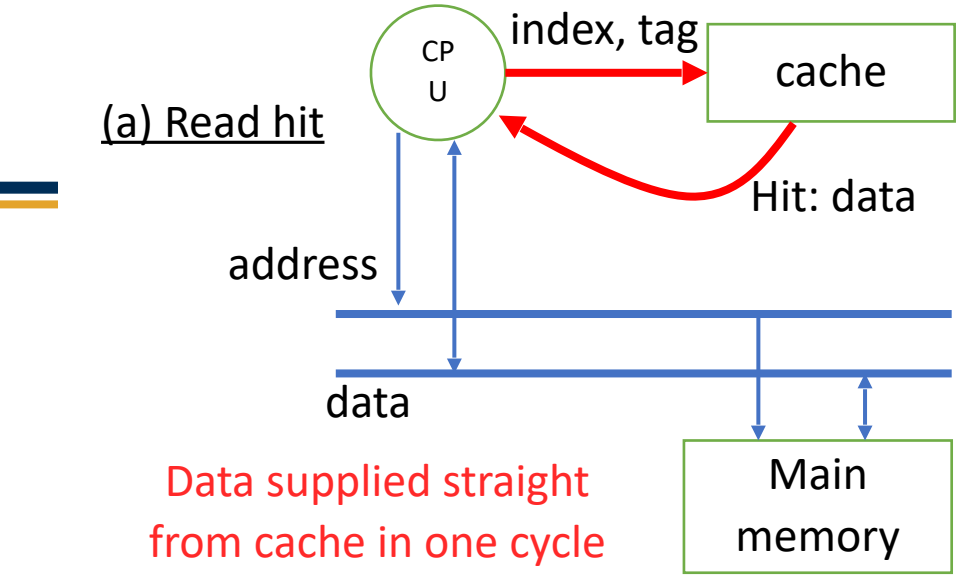

(a) Read hit
CP
U
index, tag
cache
Hit: data
address
data
Main
memory
Data supplied straight
from cache in one cycle

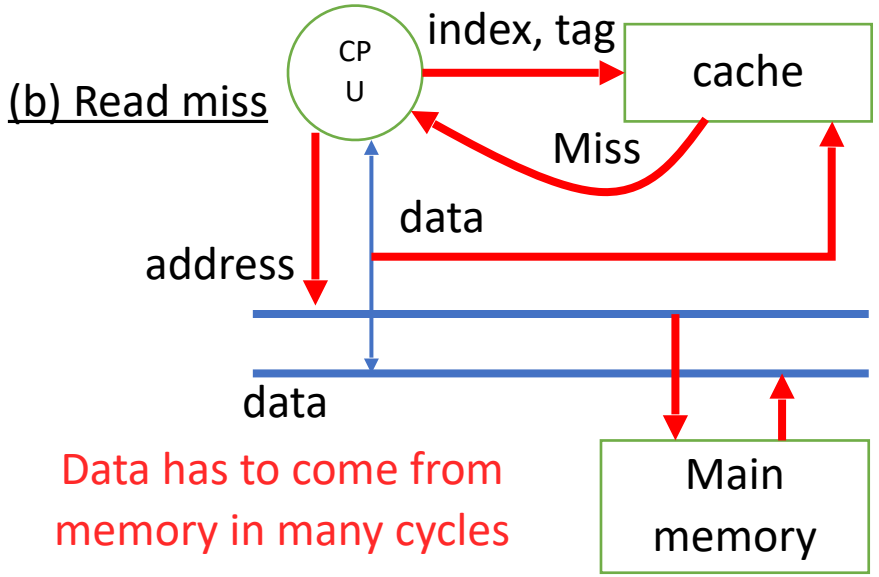

(b) Read miss
CP
U
index, tag
cache
Miss
data
address
data
Main
memory
Data has to come from
memory in many cycles

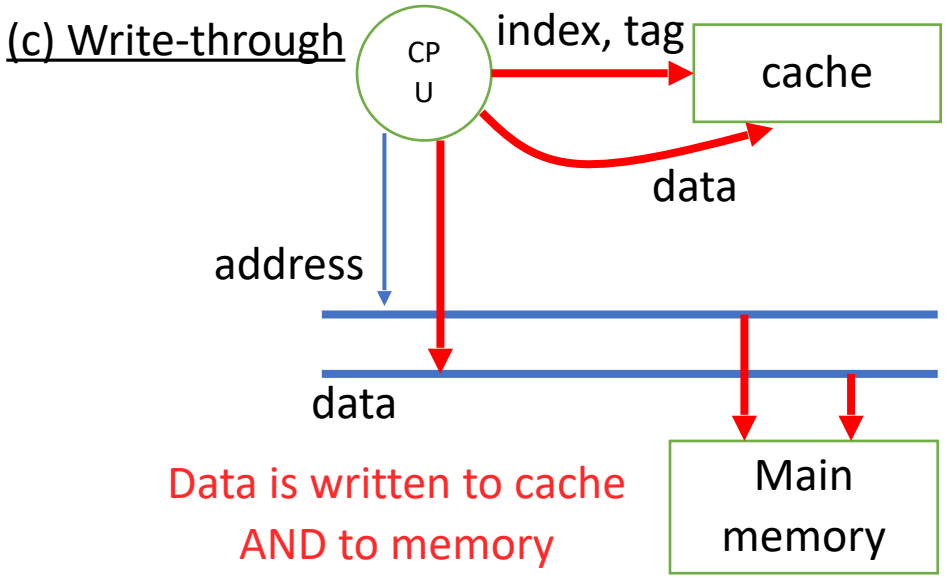

(a) Read hit
CP
U
index, tag
cache
Hit: data
address
data
Main
memory
Data supplied straight
from cache in one cycle

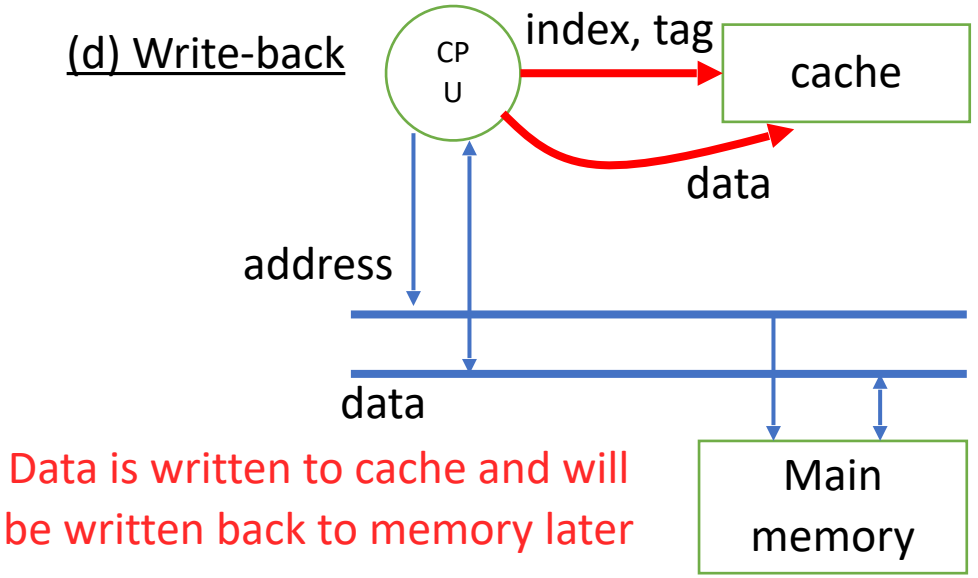

(b) Read miss
CP
U
index, tag
cache
Miss
data
address
data
Main
memory
Data has to come from
memory in many cycles

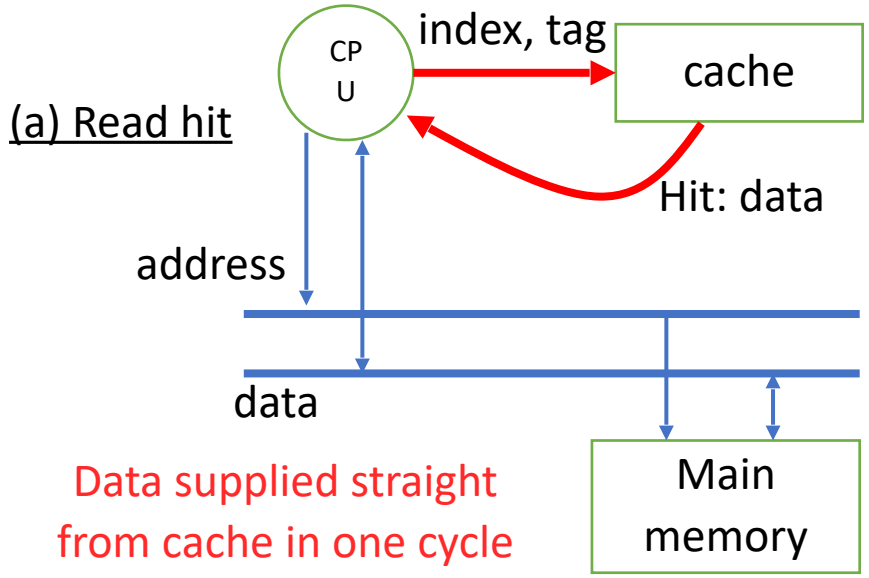

(d) Write-back
CP
U
index, tag
cache
data
address
data
Main
memory
Data is written to cache and will
be written back to memory later

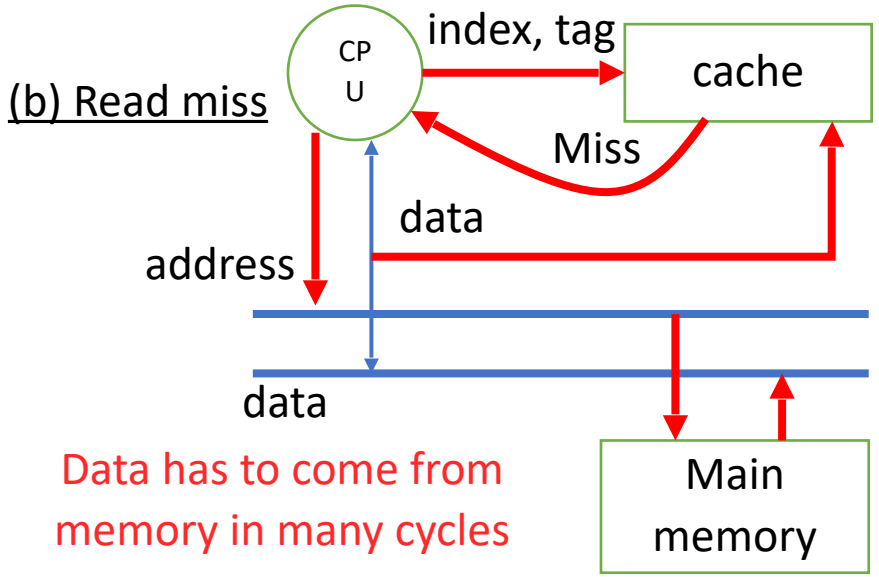

(a) Read hit
CP
U
index, tag
cache
Hit: data
address
data
Main
memory
Data supplied straight
from cache in one cycle

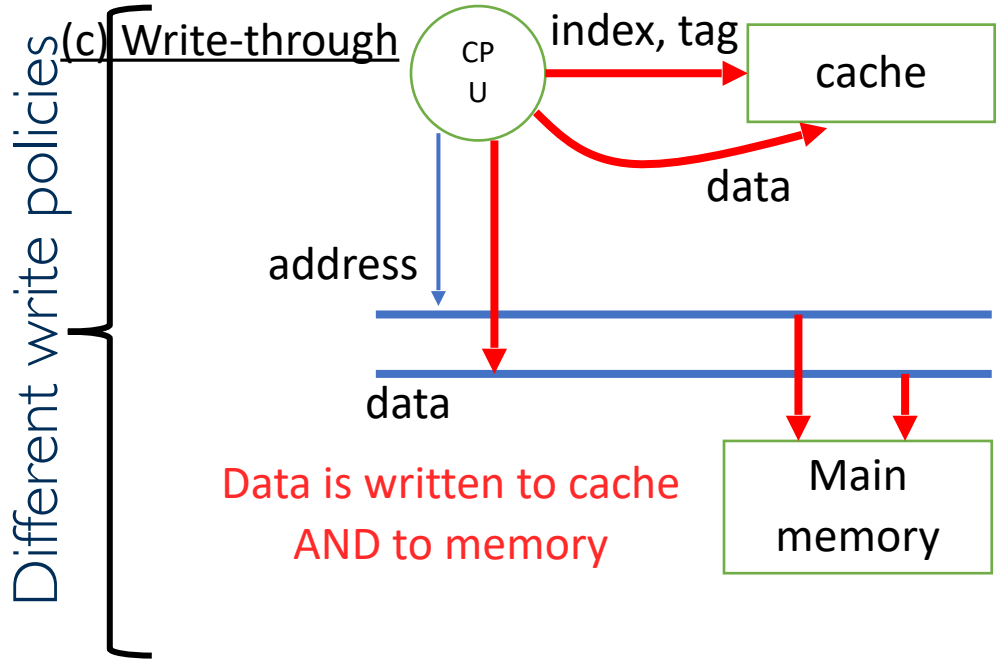

(b) Read miss
CP
U
index, tag
cache
Miss
data
address
data
Main
memory
Data has to come from
memory in many cycles

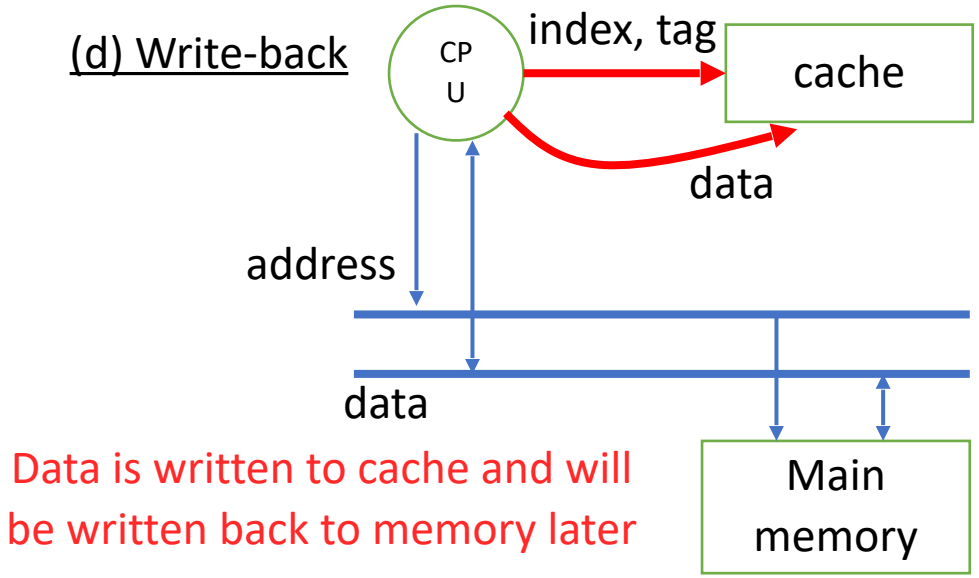

(c) Write-through
CP
U
index, tag
cache
data
address
data
Main
memory
Data is written to cache
AND to memory


(d) Write-back
CP
U
index, tag
cache
data
address
data
Main
memory
Data is written to cache and will
be written back to memory later

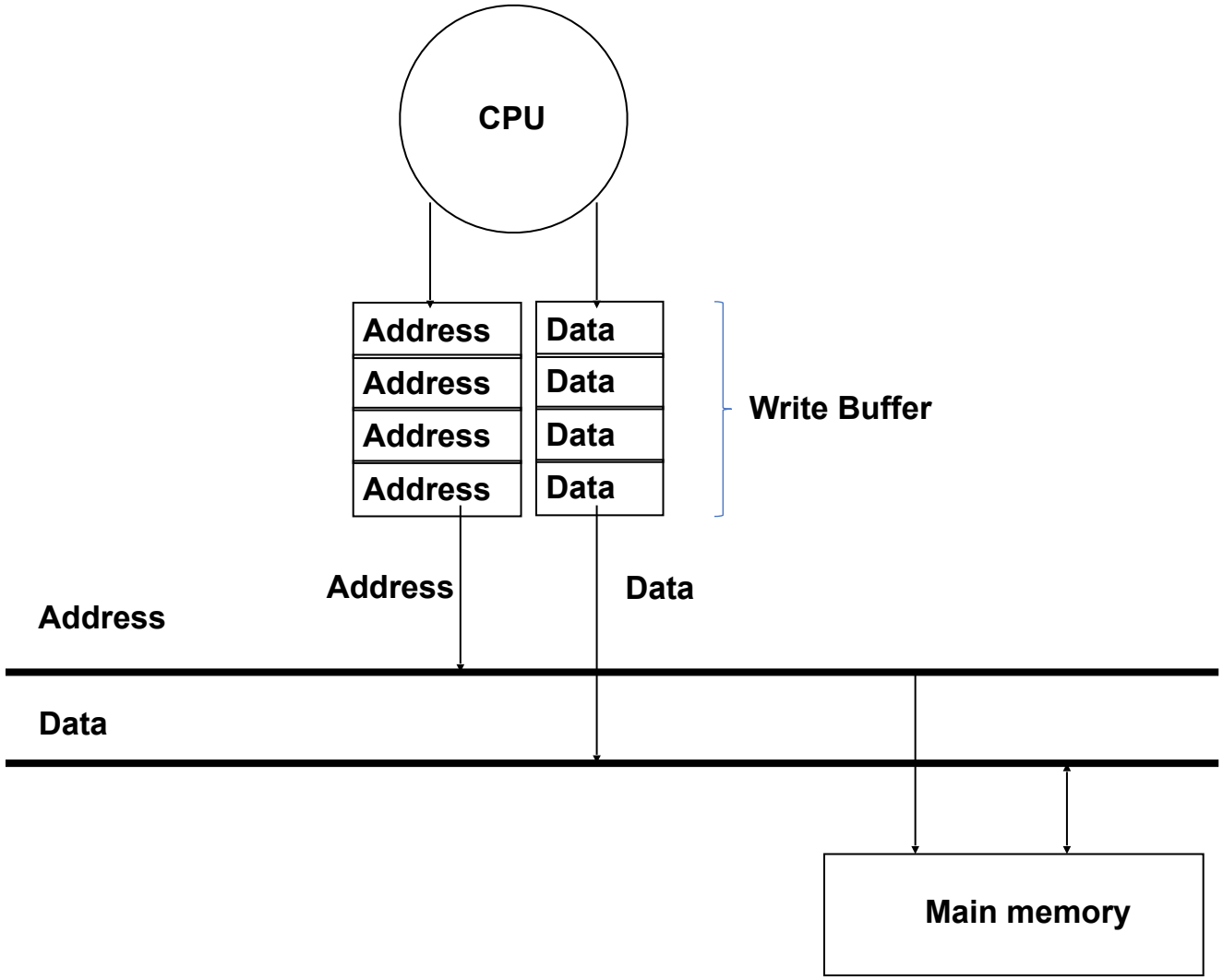

(a) Read hit
CPU
index, tag
cache
Hit: data
address
data
Main memory
Data supplied straight from cache in one cycle


(b) Read miss
CPU
index, tag
cache
Miss
data
address
data
Main memory
Data has to come from memory in many cycles

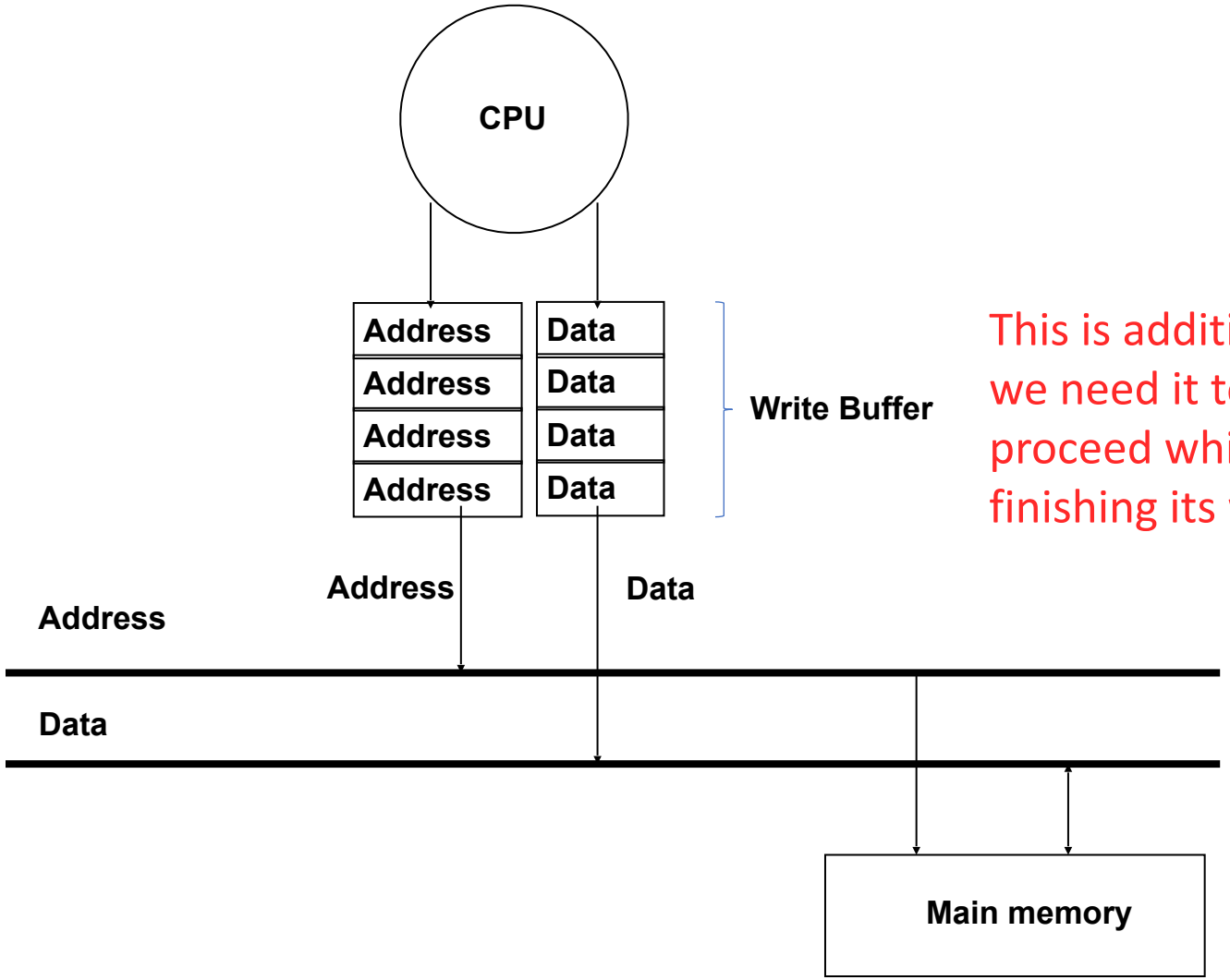

Different write policies
(c) Write-through
CPU
index, tag
cache
data
address
data
Main memory
Data is written to cache AND to memory


(d) Write-back
CPU
index, tag
cache
data
address
data
Main memory
Data is written to cache and will be written back to memory later

Write-through operation

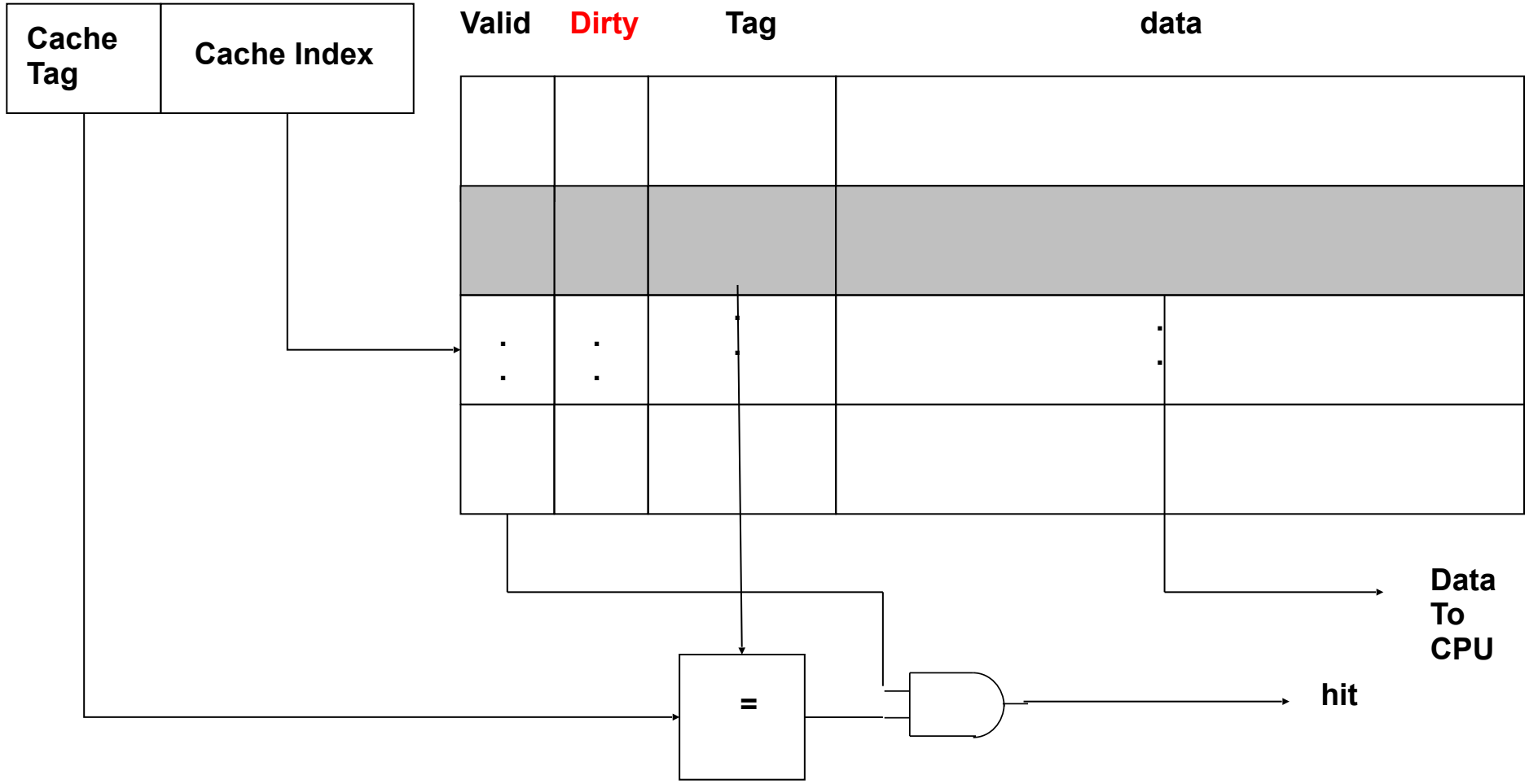


Write-through operation

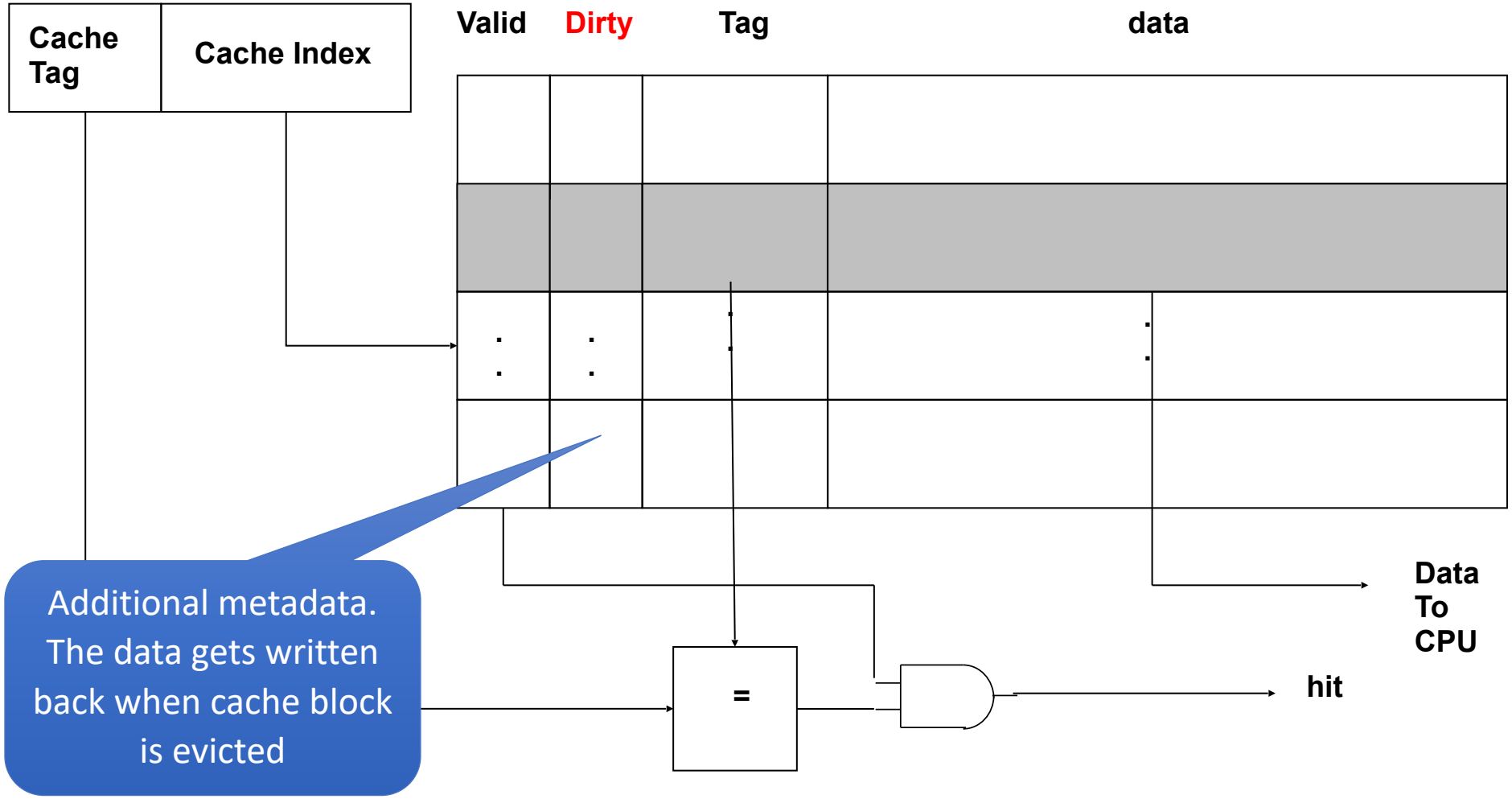


This is additional hardware – we need it to allow the CPU to proceed while the memory is finishing its write cycle

Write-back cache

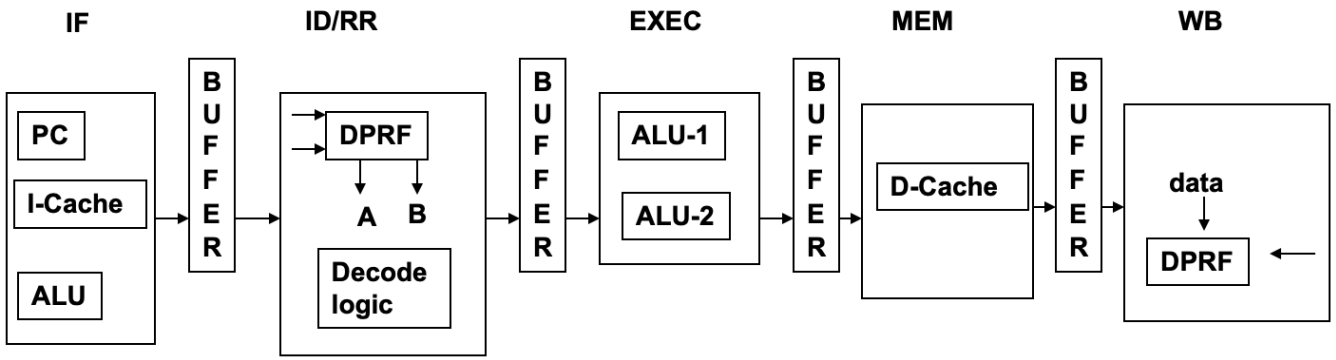


Write-back cache



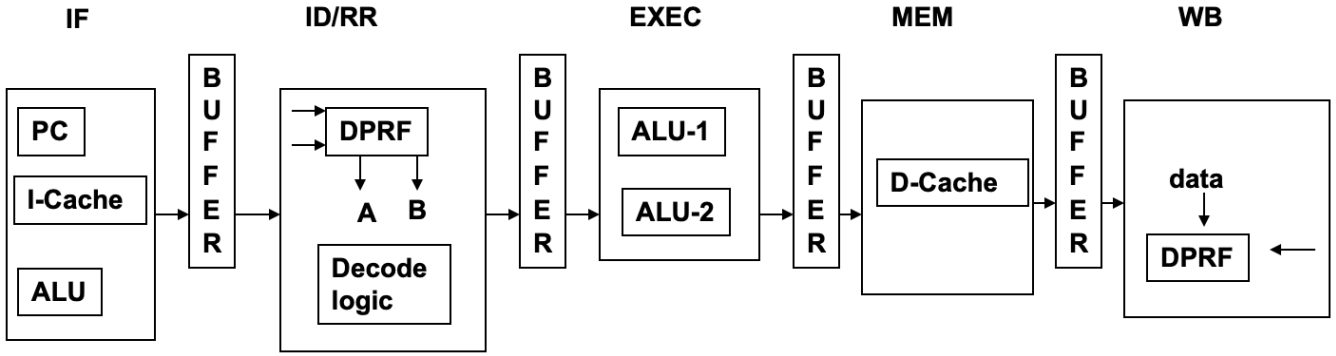
Read miss stalls

```
I1: ld  r1,a        ;r1 <- memory at a
I2: add r3,r4,r5    ;r3 <- r4 + r5
I3: and r6,r7,r8    ;r6 <- r7 & r8
I4: add r2,r4,r5    ;r2 <- r4 + r5
I5: add r2,r1,r2    ;r2 <- r1 + r2
```



Read miss stalls

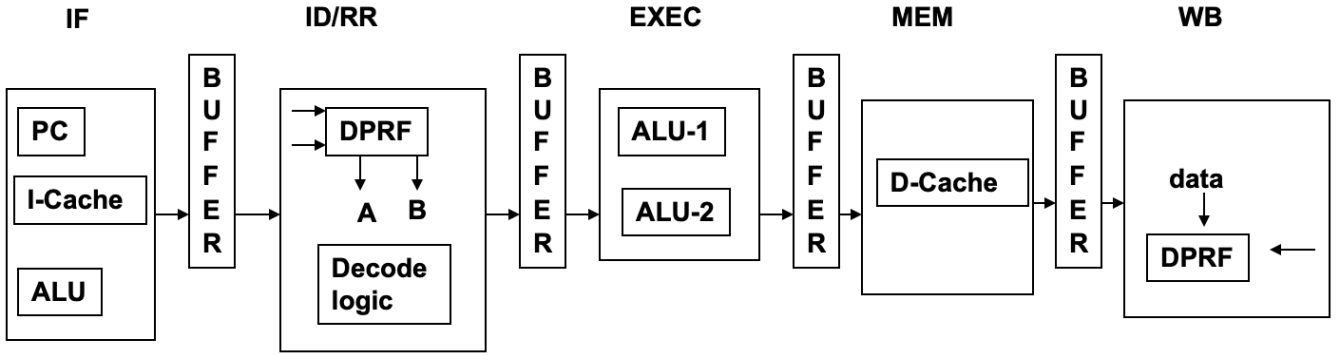
```
I1: ld   r1,a        ;r1 <- memory at a
I2: add r3,r4,r5  ;r3 <- r4 + r5
I3: and r6,r7,r8  ;r6 <- r7 & r8
I4: add r2,r4,r5  ;r2 <- r4 + r5
I5: add r2,r1,r2  ;r2 <- r1 + r2
```



Read miss stalls

```
I1: ld   r1,a        ;r1 <- memory at a
I2: add r3,r4,r5  ;r3 <- r4 + r5
I3: and r6,r7,r8  ;r6 <- r7 & r8
I4: add r2,r4,r5  ;r2 <- r4 + r5
I5: add r2,r1,r2  ;r2 <- r1 + r2
```

- We can treat a read-cache-miss in MEM in a similar fashion as we did previously with registers and the busy bits



Execution time with caches

Execution time with caches

Execution time = N * CPI_{Avg} * cycle time

Execution time with caches

Execution time = N * CPI_{Avg} * cycle time

$CPI_{eff} = CPI_{Avg} + \text{Memory-stalls}_{Avg}$

Execution time with caches

~~Execution time = N * CPI_{Avg} * cycle time~~

$CPI_{eff} = CPI_{Avg} + \text{Memory-stalls}_{Avg}$

Execution time = N * CPI_{eff} * cycle time

Execution time with caches

~~Execution time = N * CPI_{Avg} * cycle time~~

$CPI_{eff} = CPI_{Avg} + \text{Memory-stalls}_{Avg}$

Execution time = N * CPI_{eff} * cycle time

Execution time = N * (CPI_{Avg} + M-stalls_{Avg}) * cycle time

Execution time with caches

~~Execution time = N * CPI_{Avg} * cycle time~~

$CPI_{eff} = CPI_{Avg} + \text{Memory-stalls}_{Avg}$

Execution time = N * CPI_{eff} * cycle time

Execution time = N * (CPI_{Avg} + M-stalls_{Avg}) * cycle time

$\text{Memory-stalls}_{Avg}$ = $\text{misses per instruction}_{Avg}$ * $\text{miss-penalty}_{Avg}$

Execution time with caches

~~Execution time = N * CPI_{Avg} * cycle time~~

$CPI_{eff} = CPI_{Avg} + \text{Memory-stalls}_{Avg}$

Execution time = N * CPI_{eff} * cycle time

Execution time = N * (CPI_{Avg} + M-stalls_{Avg}) * cycle time

$\text{Memory-stalls}_{Avg}$ = $\text{misses per instruction}_{Avg}$ * $\text{miss-penalty}_{Avg}$

Total memory stalls = N * $\text{Memory-stalls}_{Avg}$

Session ID: dag2200

The effective CPI is…

Average CPI = 1.5

Average cache miss per instruction = 3%

Miss penalty = 20

A. 1.8
B. 2.1
C. 21.5
D. 7.5

Session ID: dag2200

The effective CPI is…

Average CPI = 1.5

Average cache miss per instruction = 3%

Miss penalty = 20

A. 1.8
B. 2.1
C. 21.5
D. 7.5

$CPI_{eff} = 1.5 + (3\% * 20) = 1.5 + 0.6 = 2.1$ CPI

Example

- Consider a pipelined processor that has an average CPI of 1.8 without accounting for memory stalls.
  - I-Cache has a hit ratio of 95%
  - D-Cache has a hit ratio of 98%
- Assume that memory reference instructions account for 30% of all the instructions executed.
  - 80% are loads
  - 20% are stores
- On average
  - read-miss penalty is 20 cycles
  - write-miss penalty is 5 cycles

Compute the effective CPI of the processor accounting for the memory stalls

Solution

Cost of instruction misses:

= I-cache miss ratio * read miss penalty

= (1 – 0.95) * 20 = 1 cycle per instruction

Solution

Cost of instruction misses:

= I-cache miss ratio * read miss penalty

= (1 – 0.95) * 20 = 1 cycle per instruction

Cost of data read misses:

= % memory reference instructions

* fraction that are loads * D-cache miss ratio * read miss penalty

= 0.3 * 0.8 * (1 – 0.98) * 20 = 0.096 cycles per instruction

Solution

Cost of instruction misses:

= I-cache miss ratio * read miss penalty

= (1 – 0.95) * 20 = 1 cycle per instruction

Cost of data read misses:

= % memory reference instructions

* fraction that are loads * D-cache miss ratio * read miss penalty

= 0.3 * 0.8 * (1 – 0.98) * 20 = 0.096 cycles per instruction

Cost of data write misses:

= % memory reference instructions

* fraction that are stores * D-cache miss ratio * read miss penalty

= 0.3 * 0.2 * (1 – 0.98) * 5 = 0.006 cycles per instruction

Solution

Cost of instruction misses:

= I-cache miss ratio * read miss penalty

= (1 – 0.95) * 20 = 1 cycle per instruction

Cost of data read misses:

= % memory reference instructions

* fraction that are loads * D-cache miss ratio * read miss penalty

= 0.3 * 0.8 * (1 – 0.98) * 20 = 0.096 cycles per instruction

Cost of data write misses:

= % memory reference instructions

* fraction that are stores * D-cache miss ratio * read miss penalty

= 0.3 * 0.2 * (1 – 0.98) * 5 = 0.006 cycles per instruction

$CPI_{eff} = CPI_{avg}$ + cost of I-cache misses + cost of D-cache misses

= 1.8 + 1 + (0.096 + 0.006) = 2.902

How to improve cache efficiency

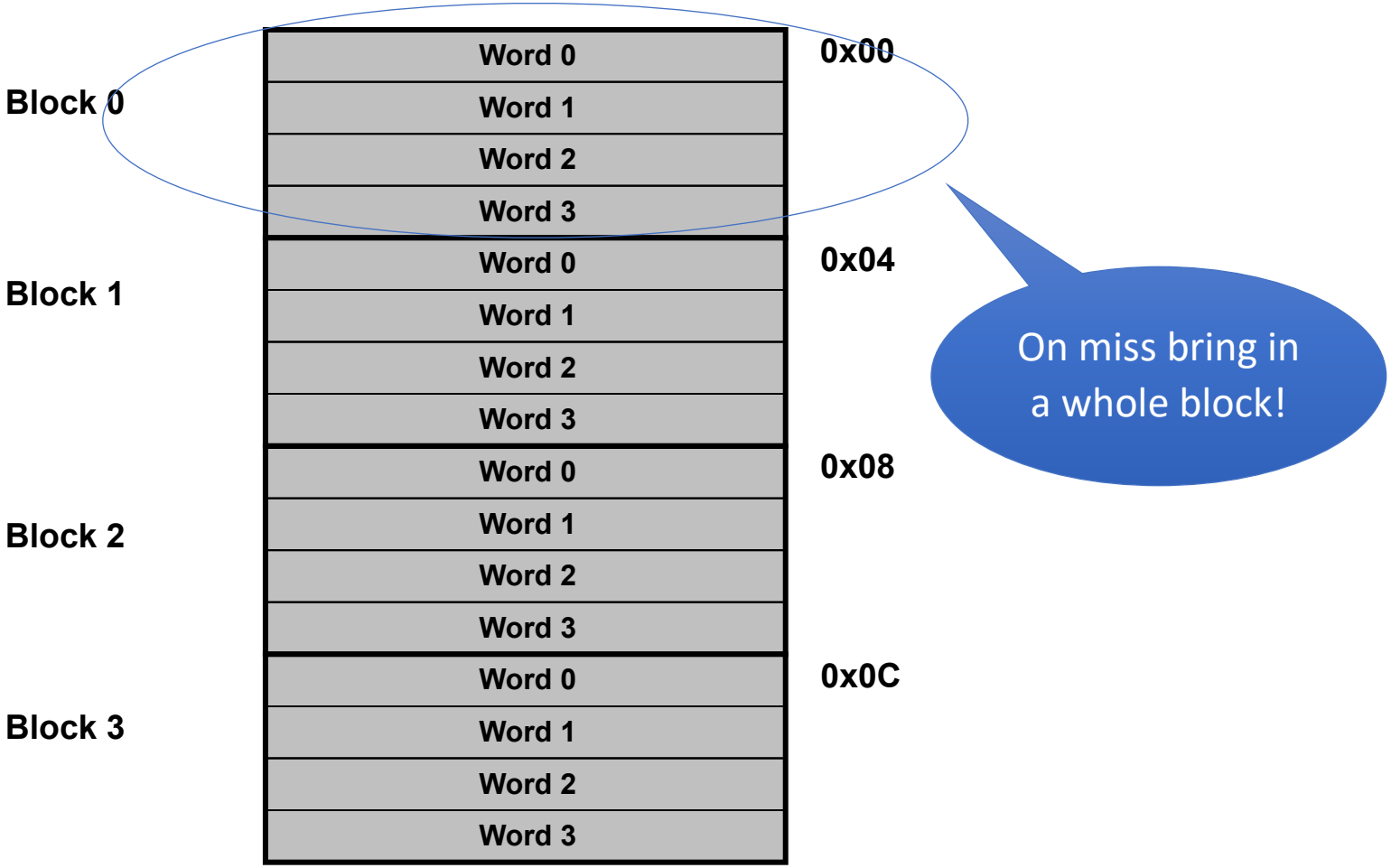
How to improve cache efficiency

- Exploit spatial locality
  - Bring more from memory into cache at a time

How to improve cache efficiency

- Exploit spatial locality
  - Bring more from memory into cache at a time
- Better organization
  - Exploit working set concept

Spatial Locality



(Re)Interpreting memory address

S = Size of cache; B = Block size; L = sets in cache

Cache Tag	Cache Index	Block Offset
t	**n**	**b**

$b = log_2 B$

$L = S/B$

$n = log_2 L$

$t = a – (b+n)$

(Re)Interpreting memory address

S = Size of cache; B = Block size; L = sets in cache

Cache Tag	Cache Index	Block Offset
t	**n**	**b**

$b = log_2 B$

$L = S/B$

$n = log_2 L$

$t = a - (b+n)$

(Re)Interpreting memory address

S = Size of cache; B = Block size; L = sets in cache

Cache Tag	Cache Index	Block Offset
t	**n**	**b**

$b = log_2 B$

$L = S/B$

$n = log_2 L$

$t = a - (b+n)$

(Re)Interpreting memory address

S = Size of cache; B = Block size; L = sets in cache

Cache Tag	Cache Index	Block Offset
t	**n**	**b**

$b = log_2 B$

$L = S/B$

$n = log_2 L$

$t = a - (b+n)$

(Re)Interpreting memory address

S = Size of cache; B = Block size; L = sets in cache

Cache Tag	Cache Index	Block Offset
t	n	b

$b = log_2 B$

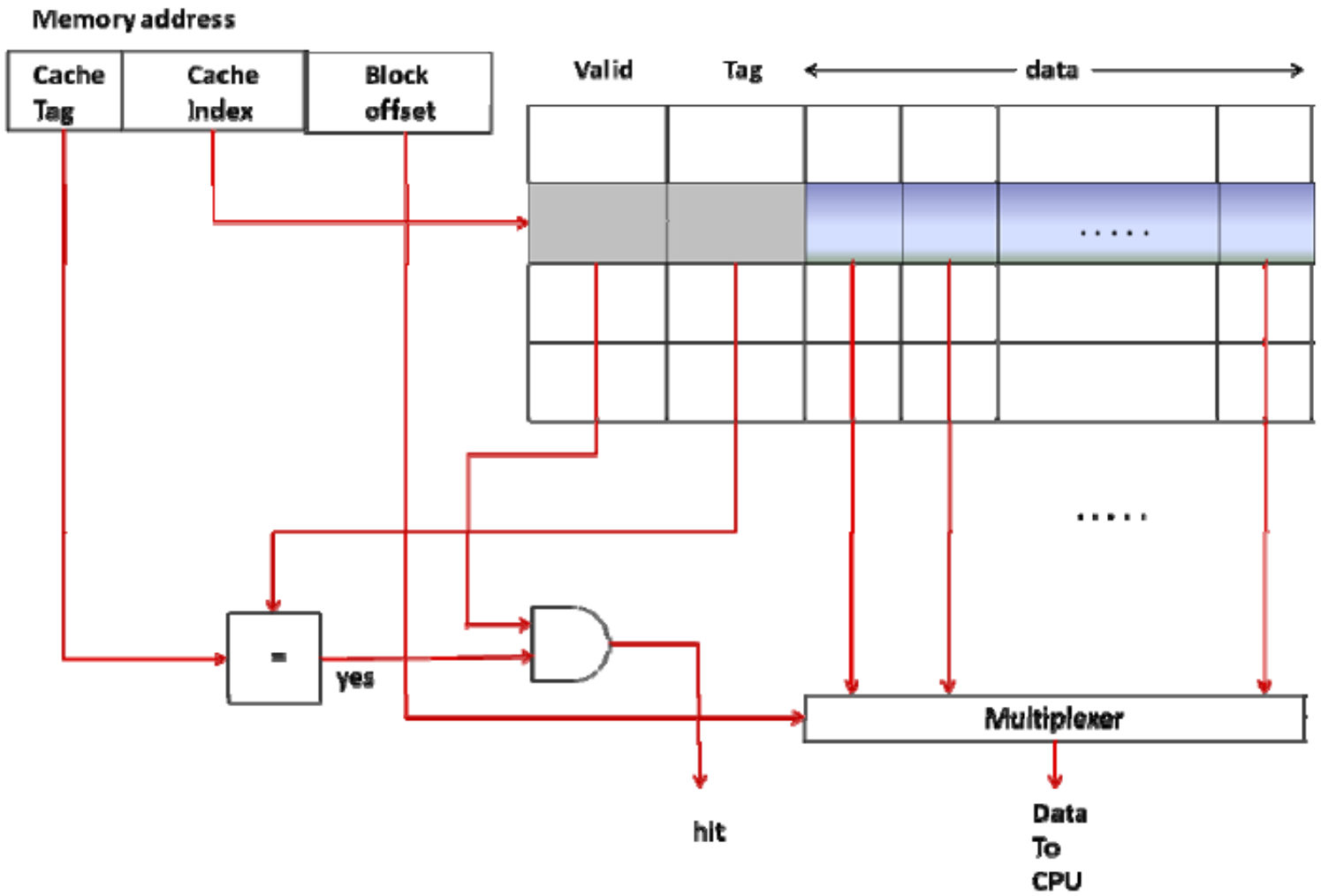
$L = S/B$

$n = log_2 L$

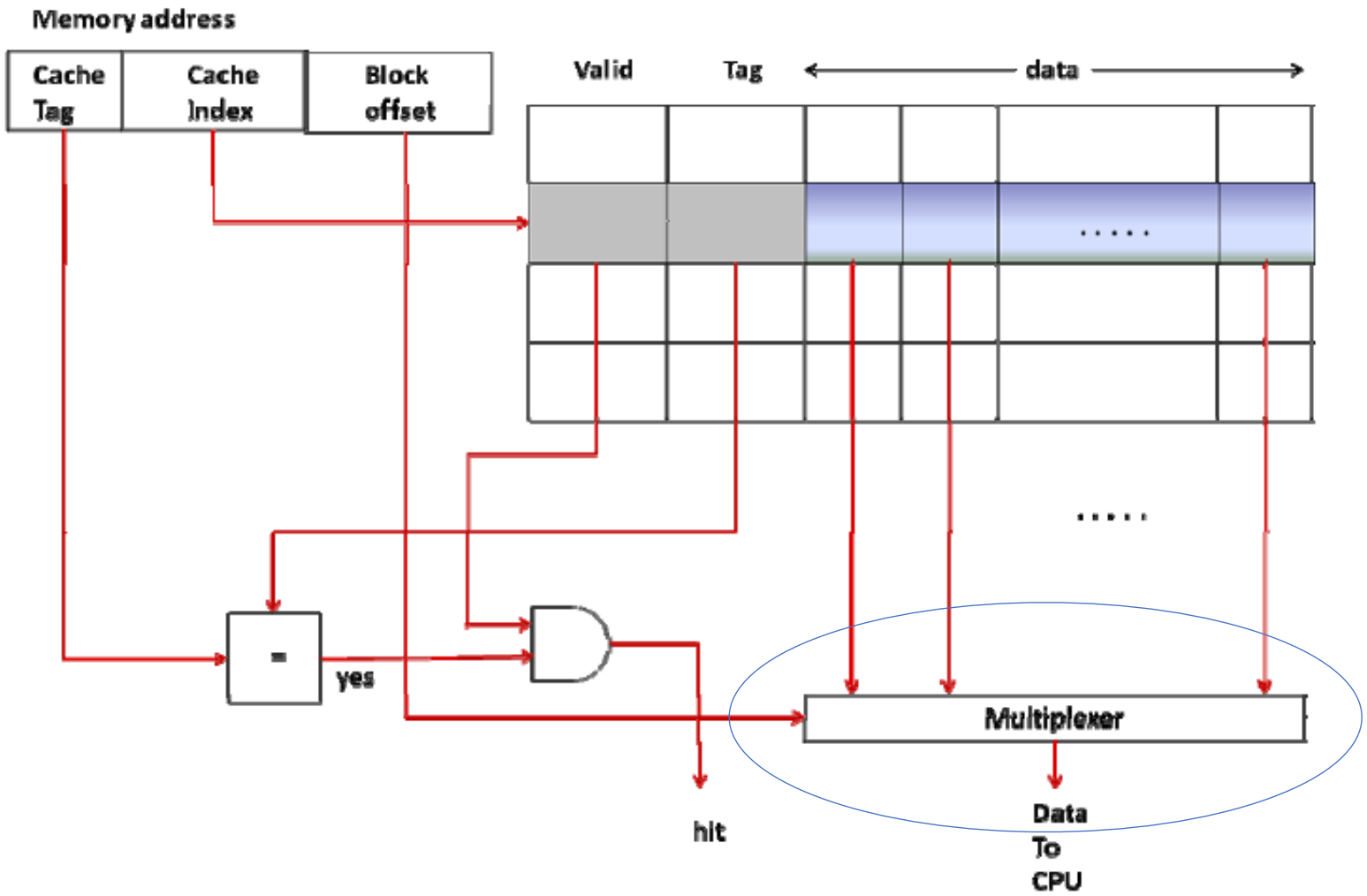
$t = a - (b+n)$

This is really just the word (or byte) number within the block

Multi-word cache organization

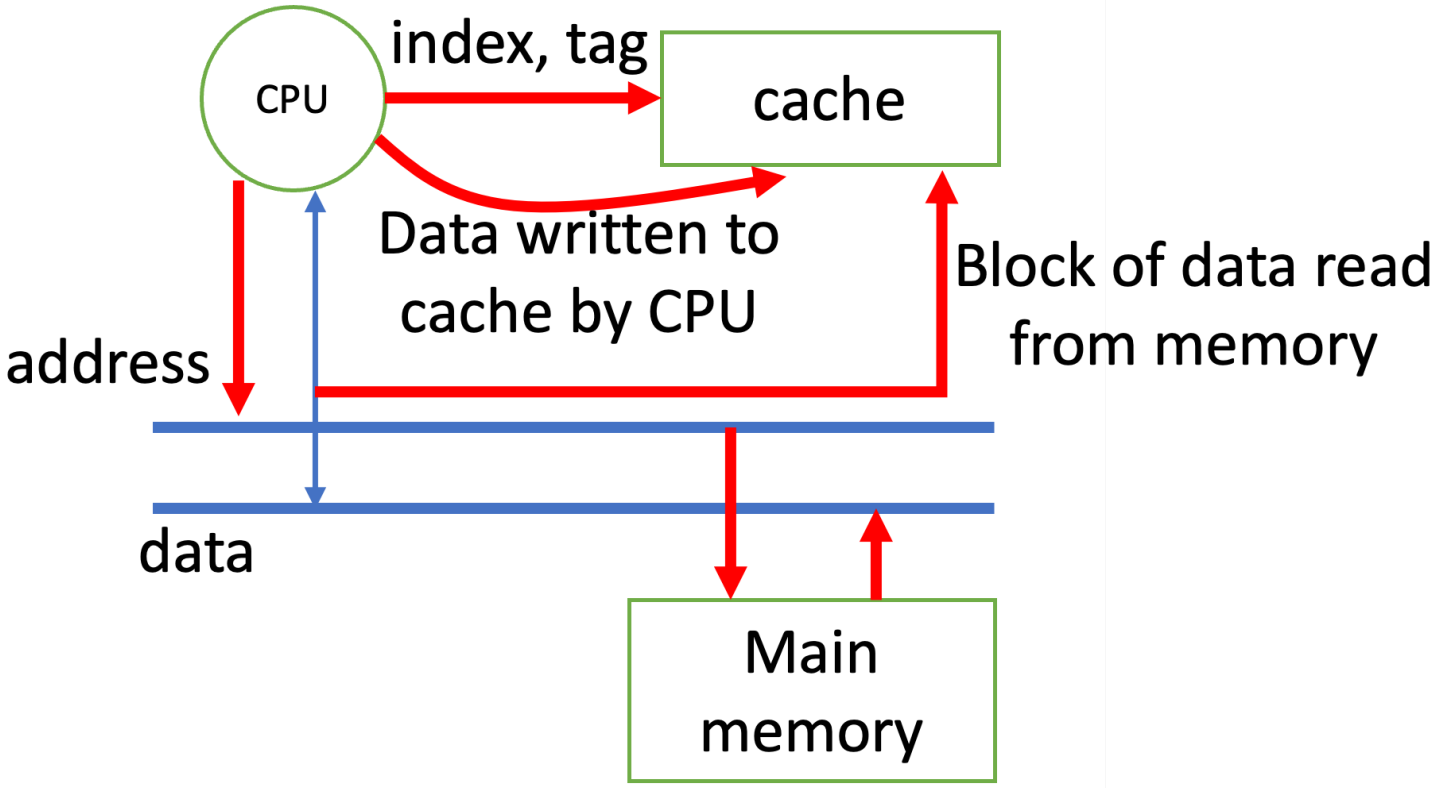


Multi-word cache organization



Write miss with multi-word cache block

The missing block is first copied from memory into the cache; only then do we update the specific word being written



Remember: the processor's interface to memory is word-sized

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Blk off

Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

Multi-word cache block example

Direct-mapped cache

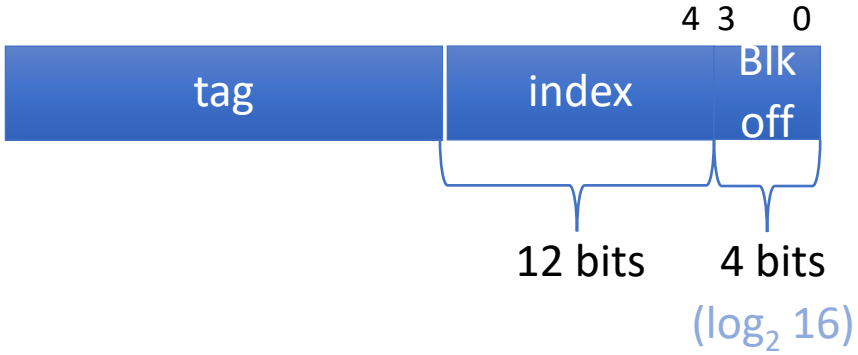
- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

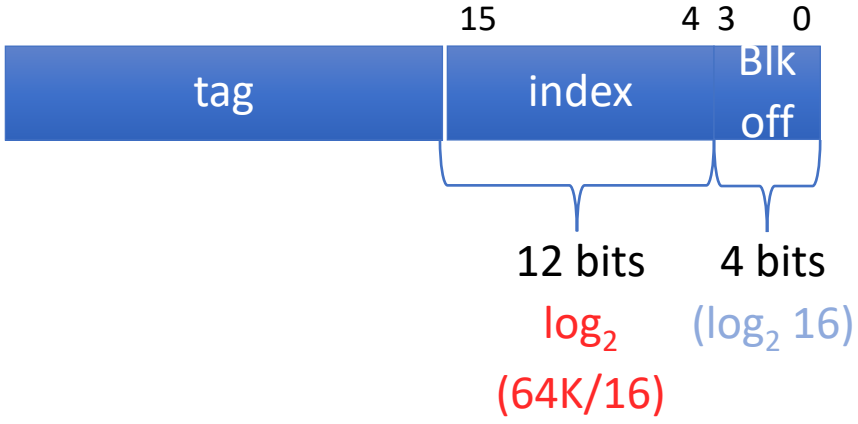


Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

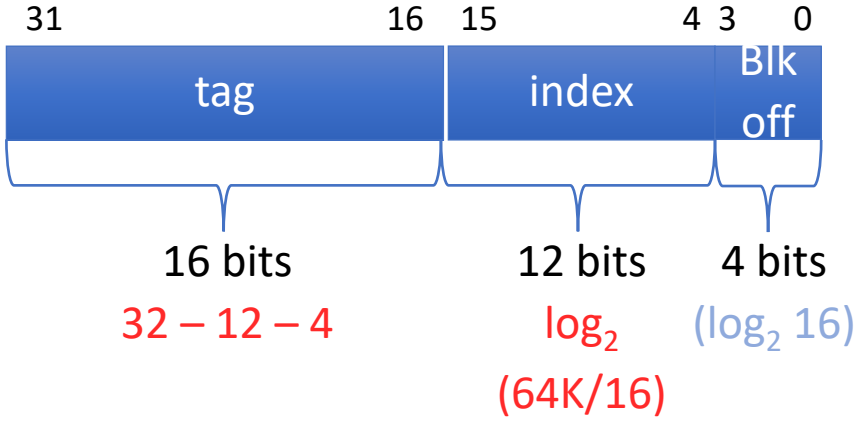


Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

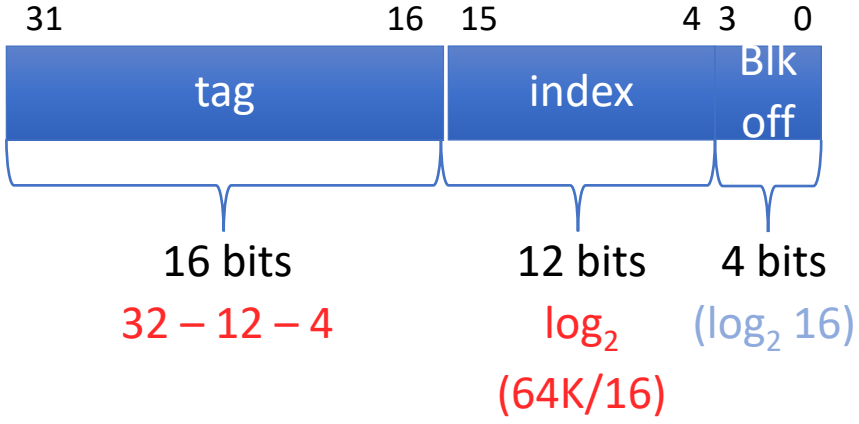


Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache



	valid	dirty	tag	word 0	word 1	word 2	word 3
0	1	0	0001				
1	1	1	0021				
2							
…							
4094							
4095							

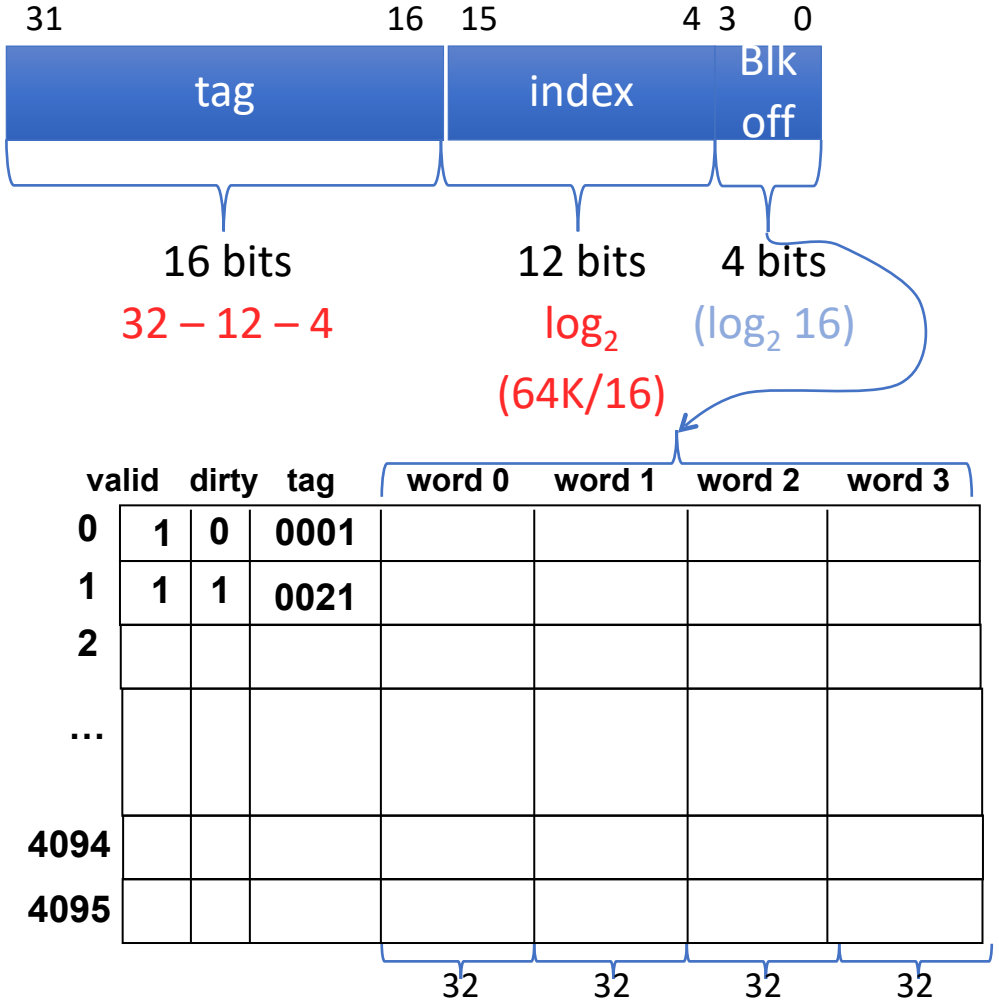
Draw cache structure

Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

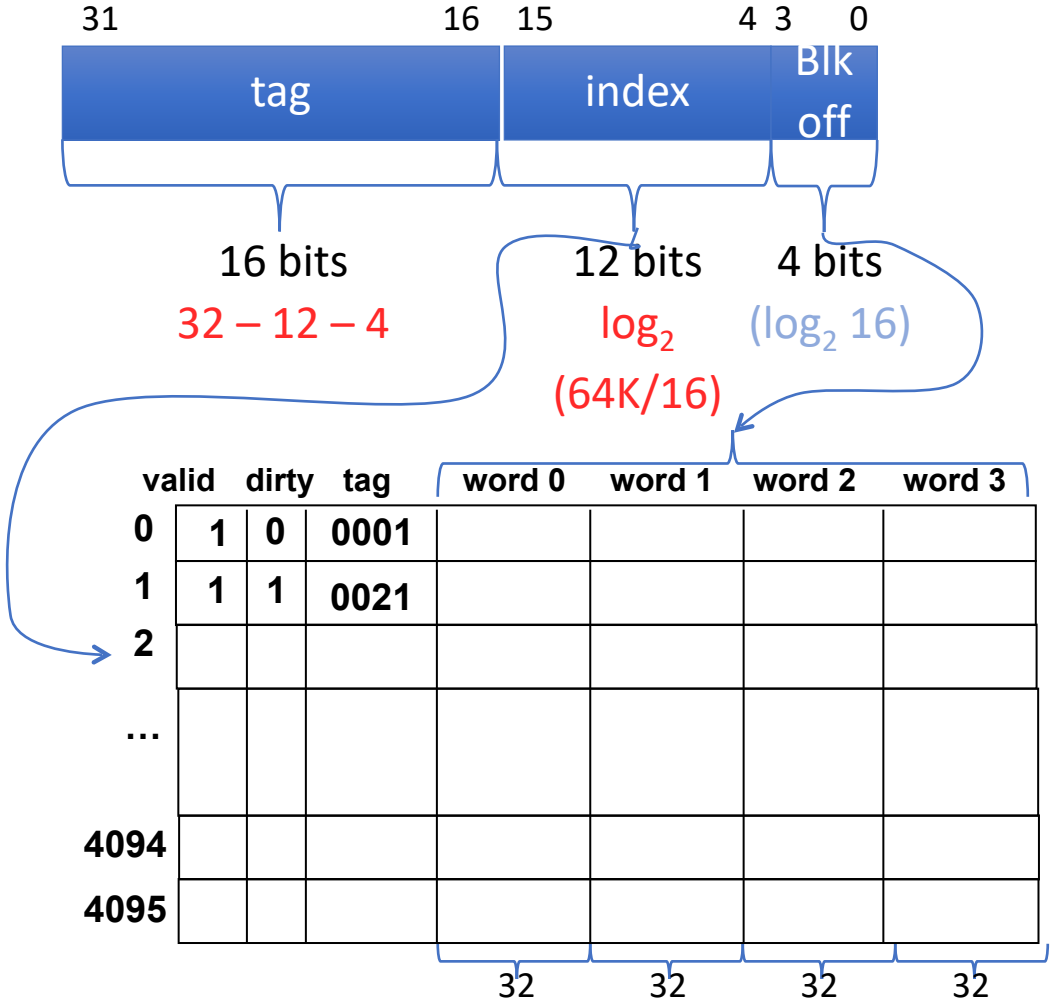


Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

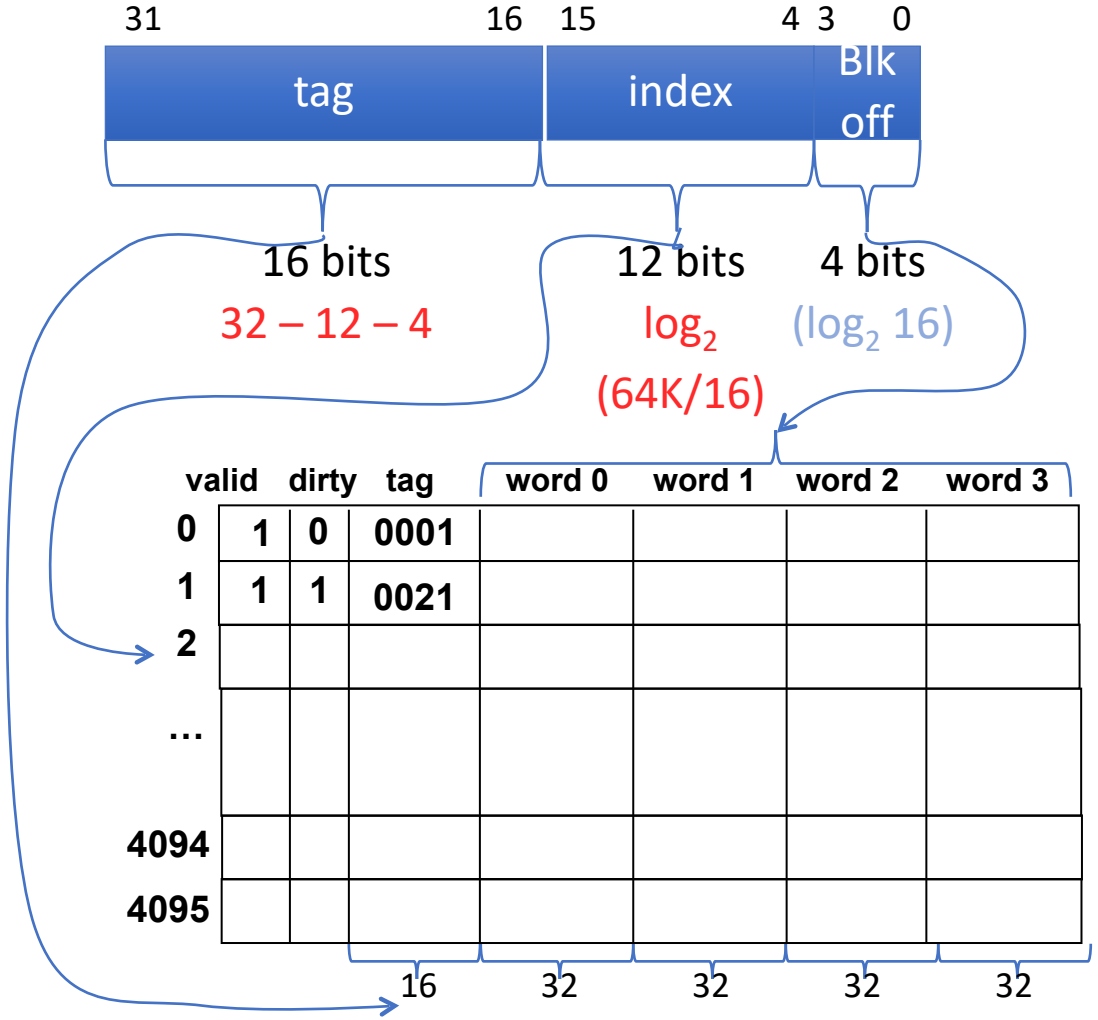


Multi-word cache block example

Direct-mapped cache

- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure

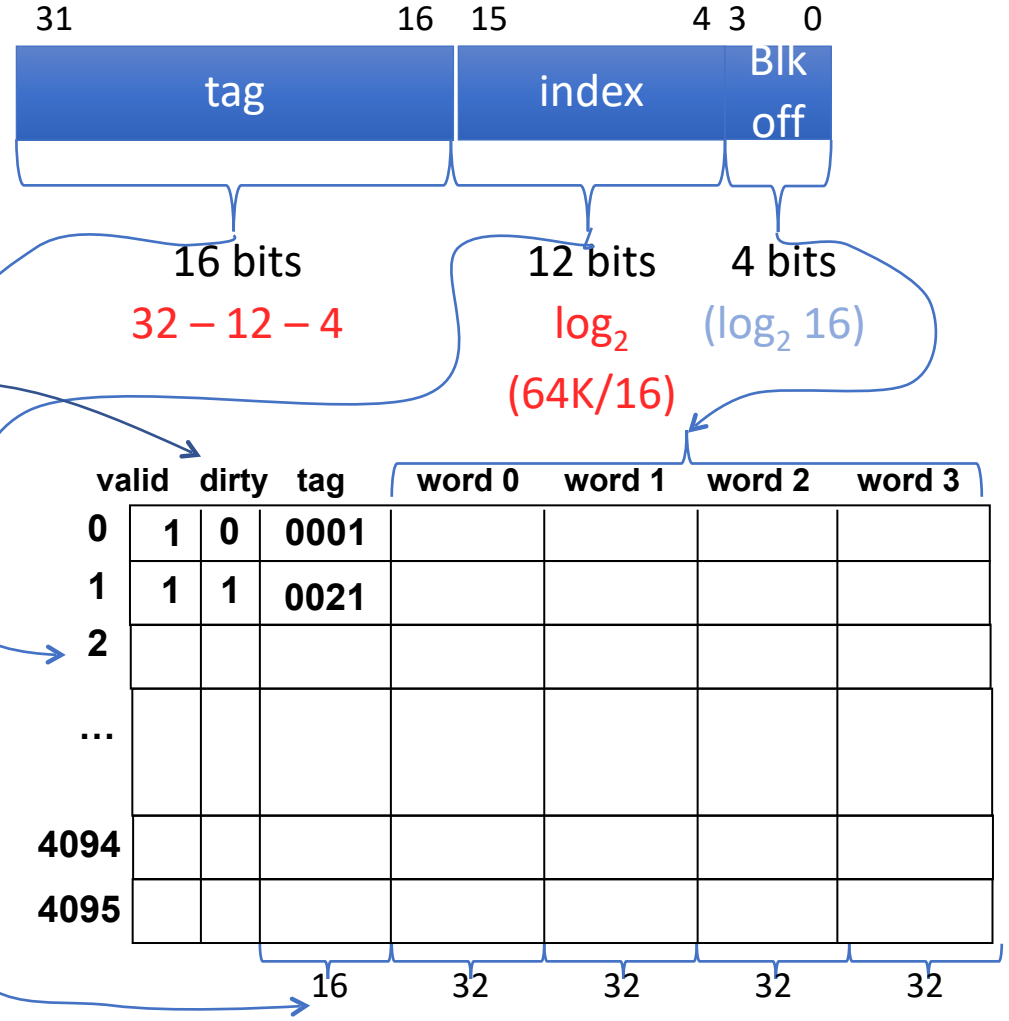


Multi-word cache block example

Direct-mapped cache

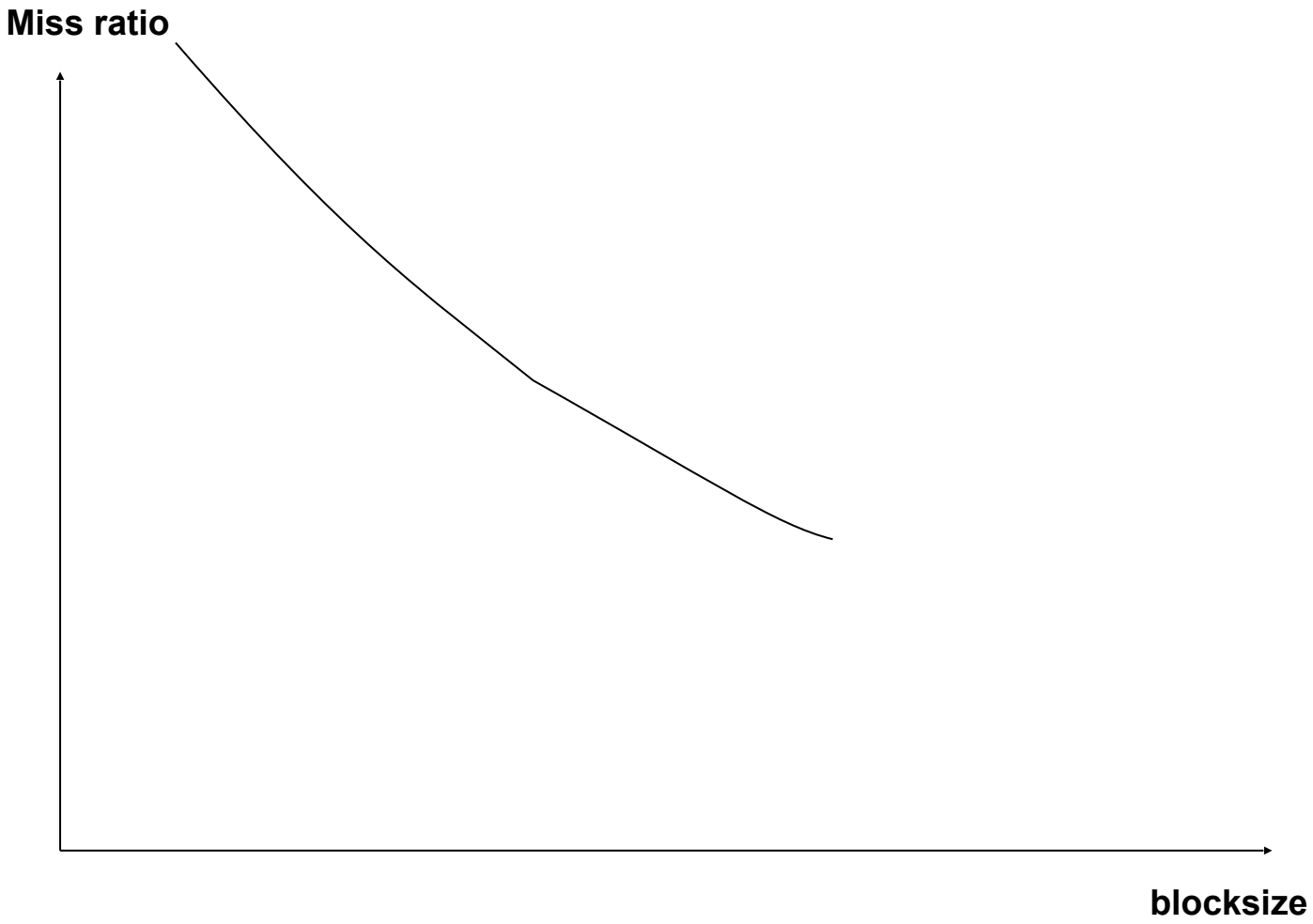
- 32-bit byte-addressable memory address
- Each memory word contains 4 bytes
- Block size = 4 words (16 bytes)
  - A memory access brings in a block
- 64K byte write-back cache

Draw cache structure



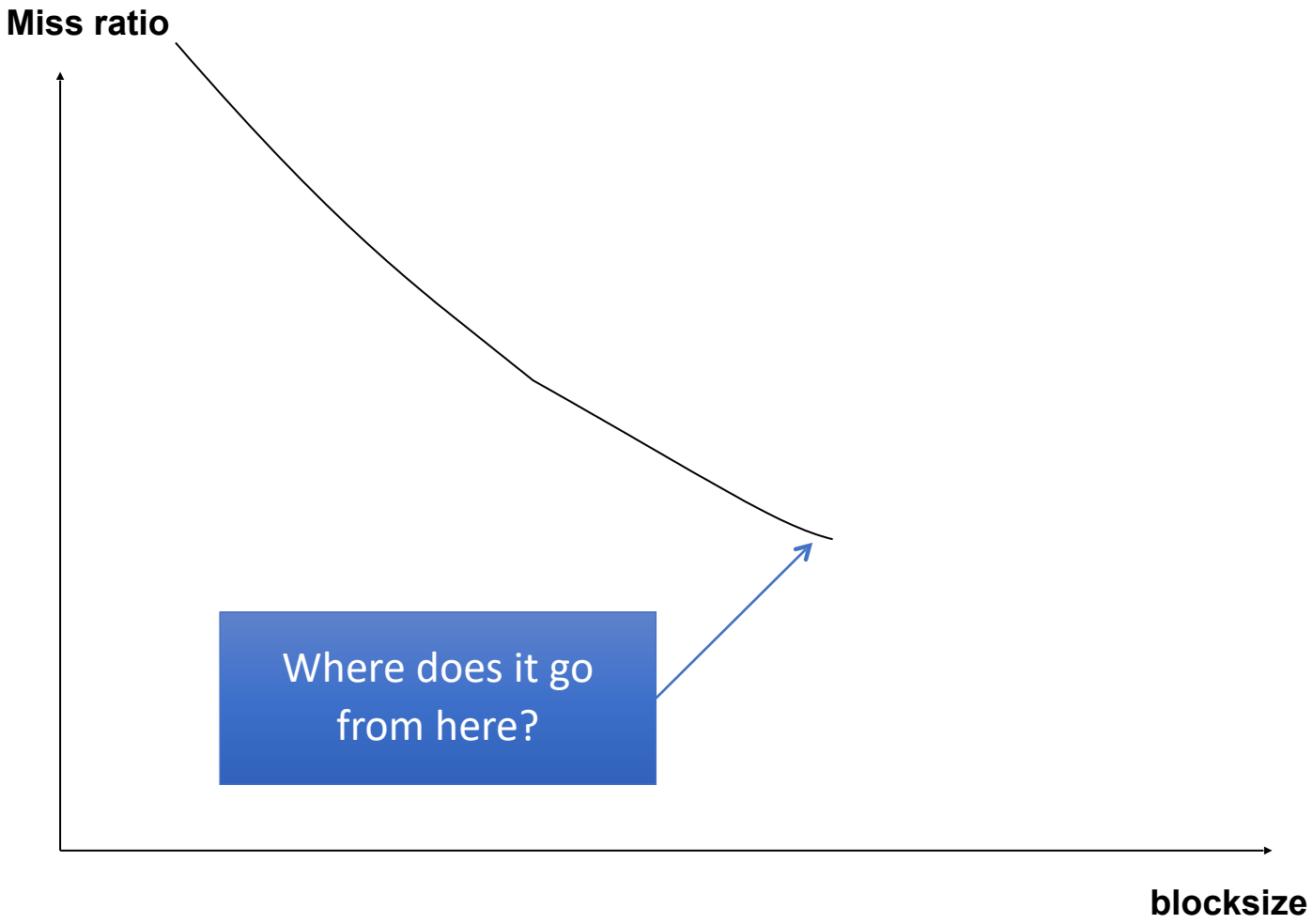
Increased block size?

- Exploits more spatial locality
- Reduces miss ratio



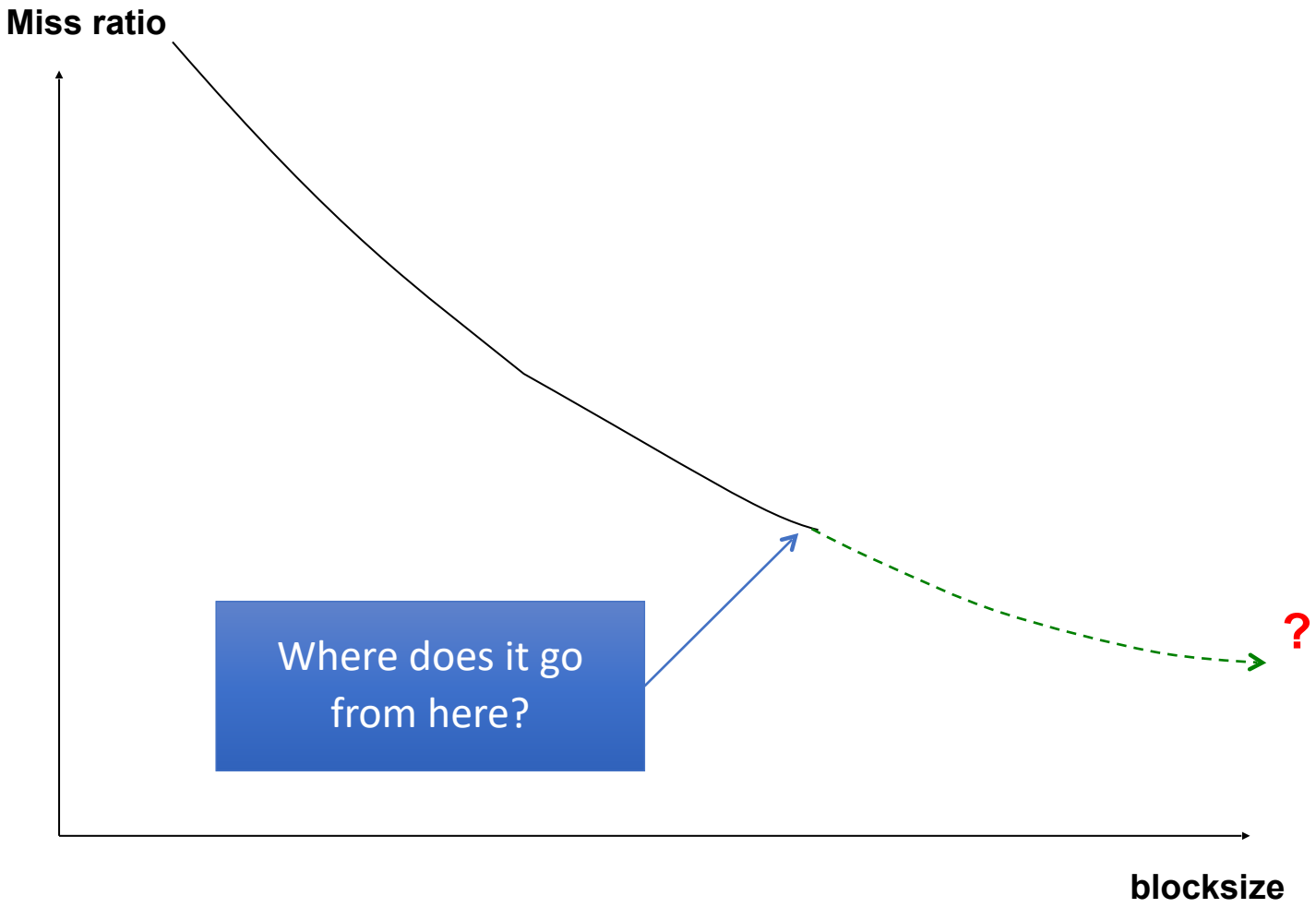
Increased block size?

- Exploits more spatial locality
- Reduces miss ratio



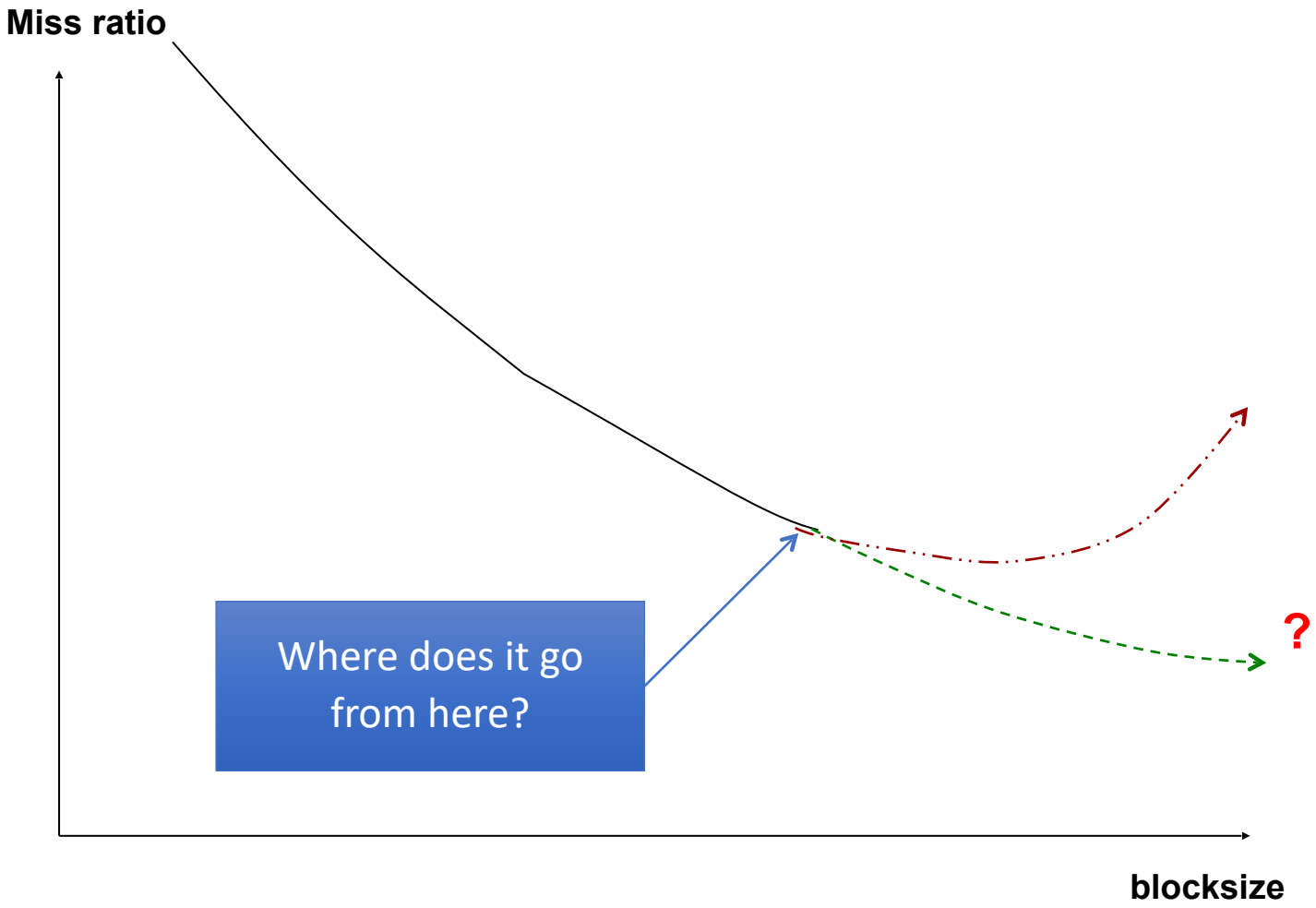
Increased block size?

- Exploits more spatial locality
- Reduces miss ratio



Increased block size?

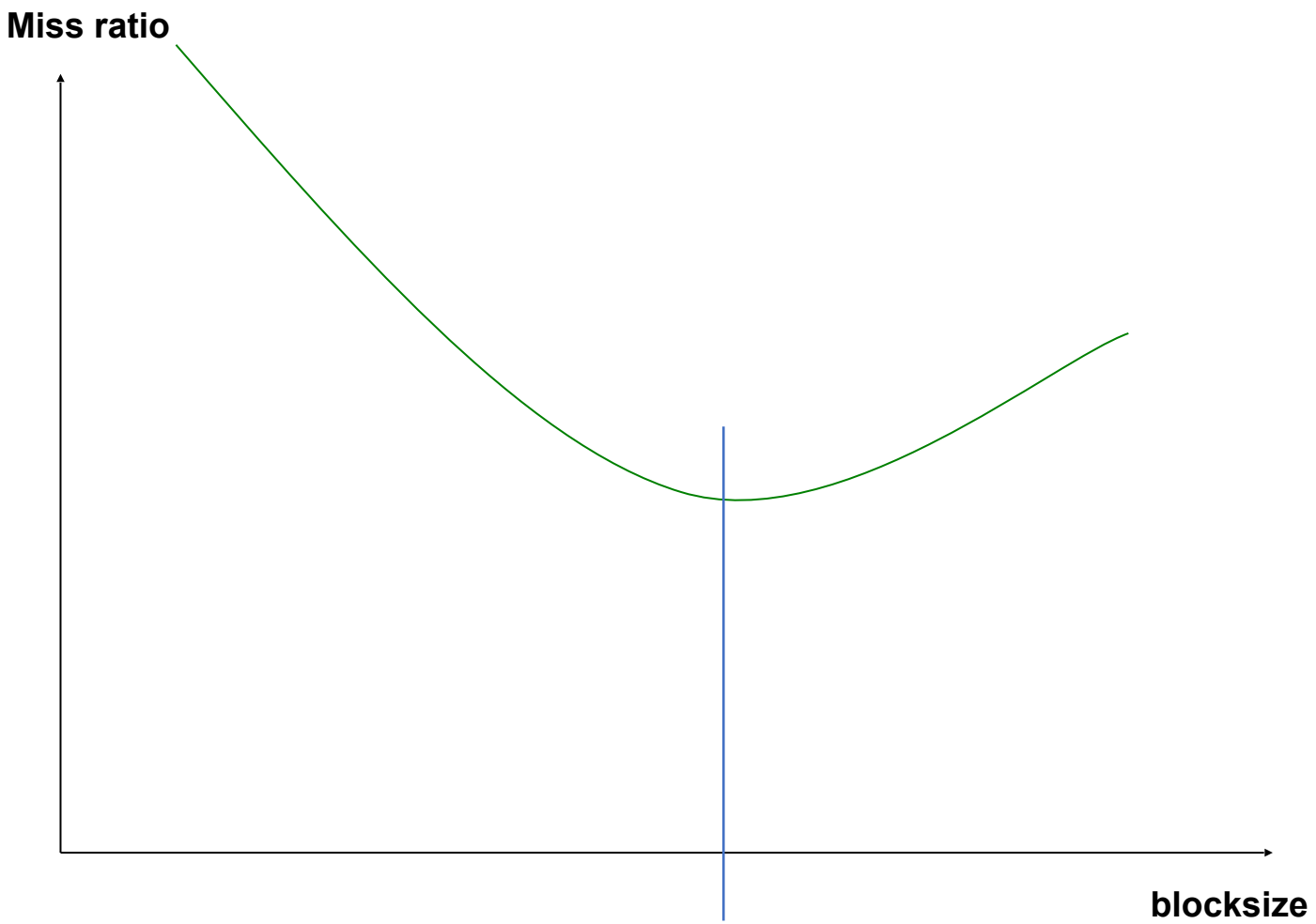
- Exploits more spatial locality
- Reduces miss ratio



Increased block size?

There is a point where things get worse

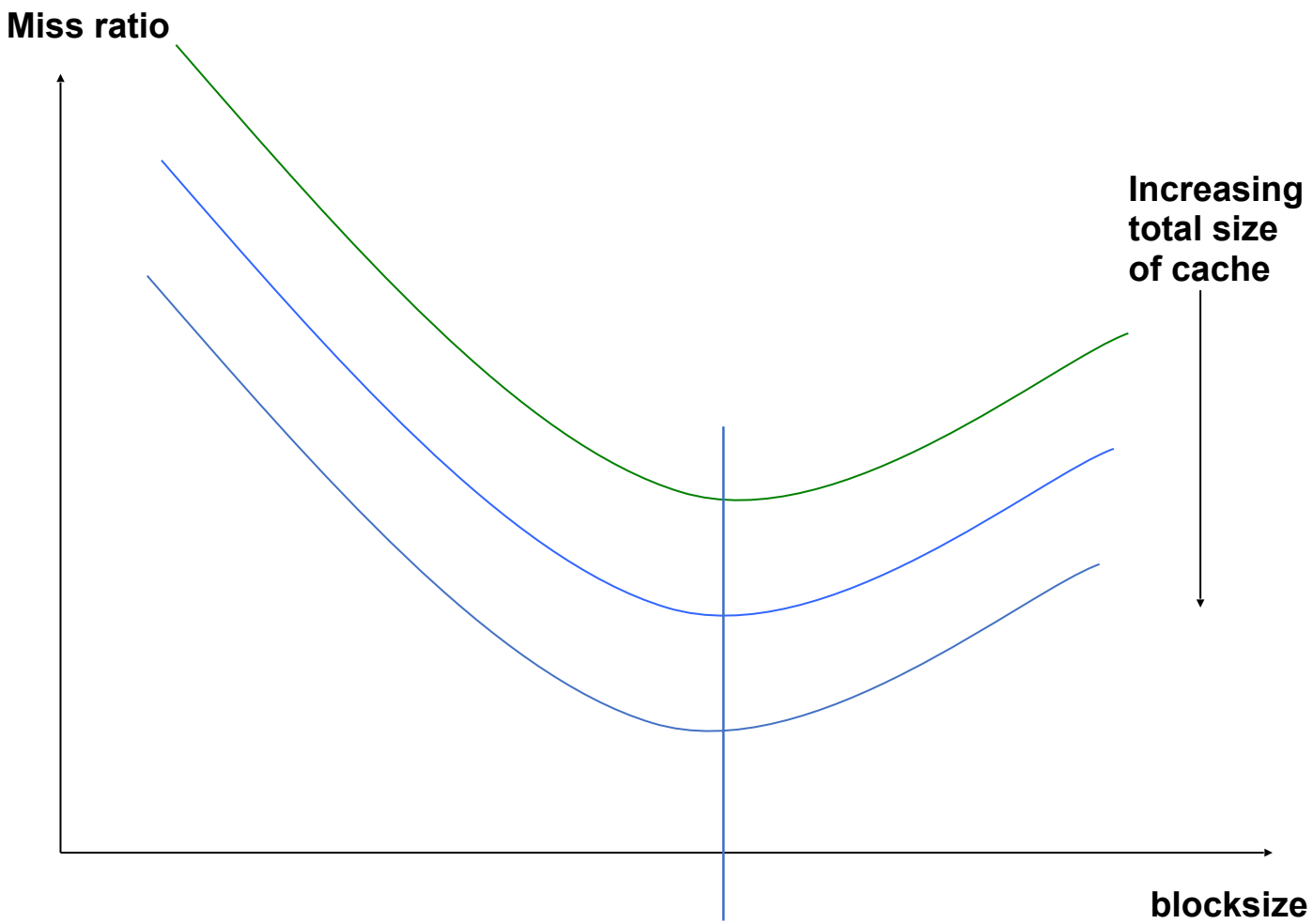
1. We reduce effective cache capacity by bringing in too much data (beyond useful spatial locality)
2. When the working set changes, larger blocks have to be fetched
   - Memory can only transfer so fast and it can become the bottleneck



Increased block size?

There is a point where things get worse

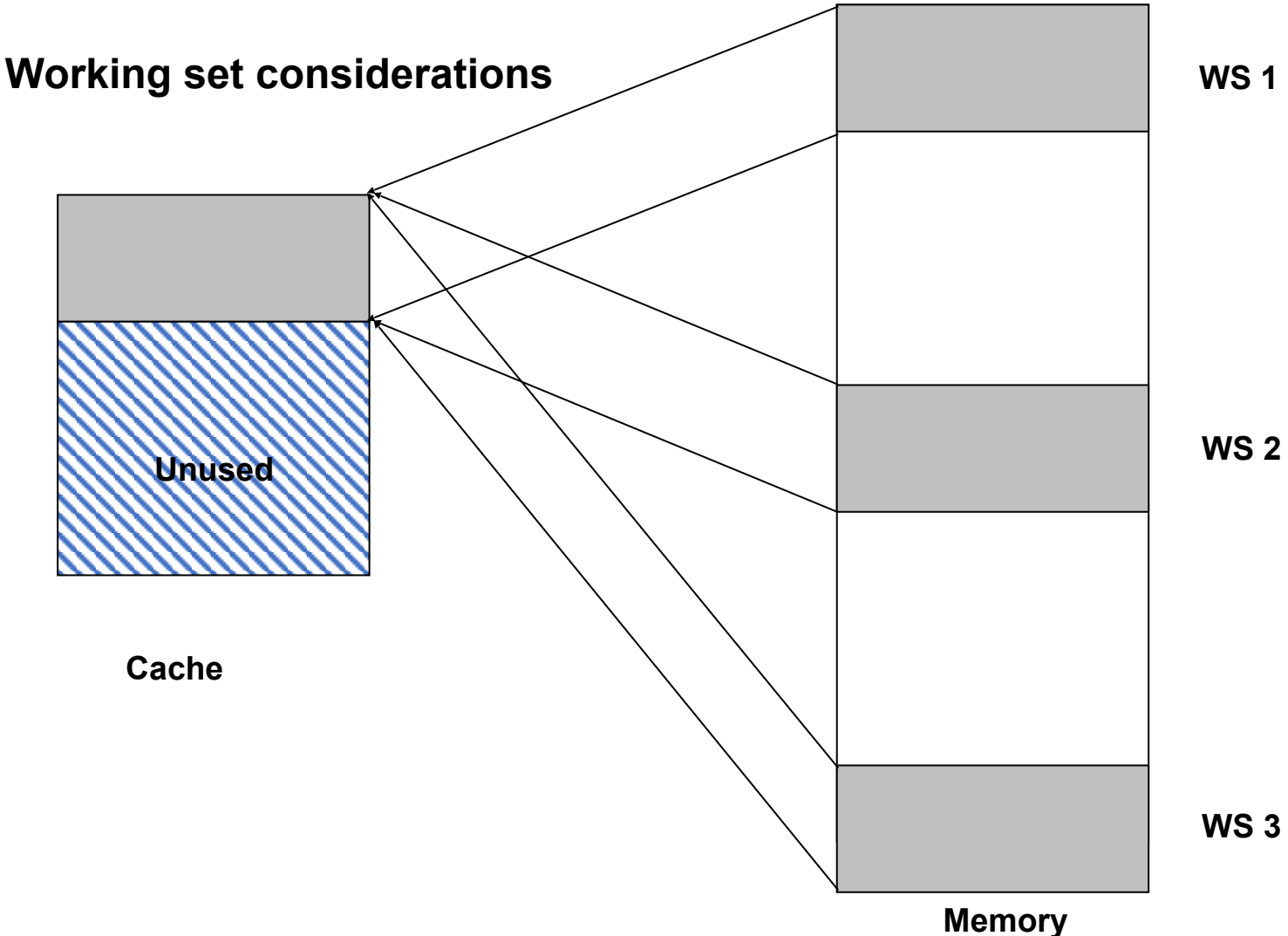
1. We reduce effective cache capacity by bringing in too much data (beyond useful spatial locality)
2. When the working set changes, larger blocks have to be fetched
   - Memory can only transfer so fast and it can become the bottleneck



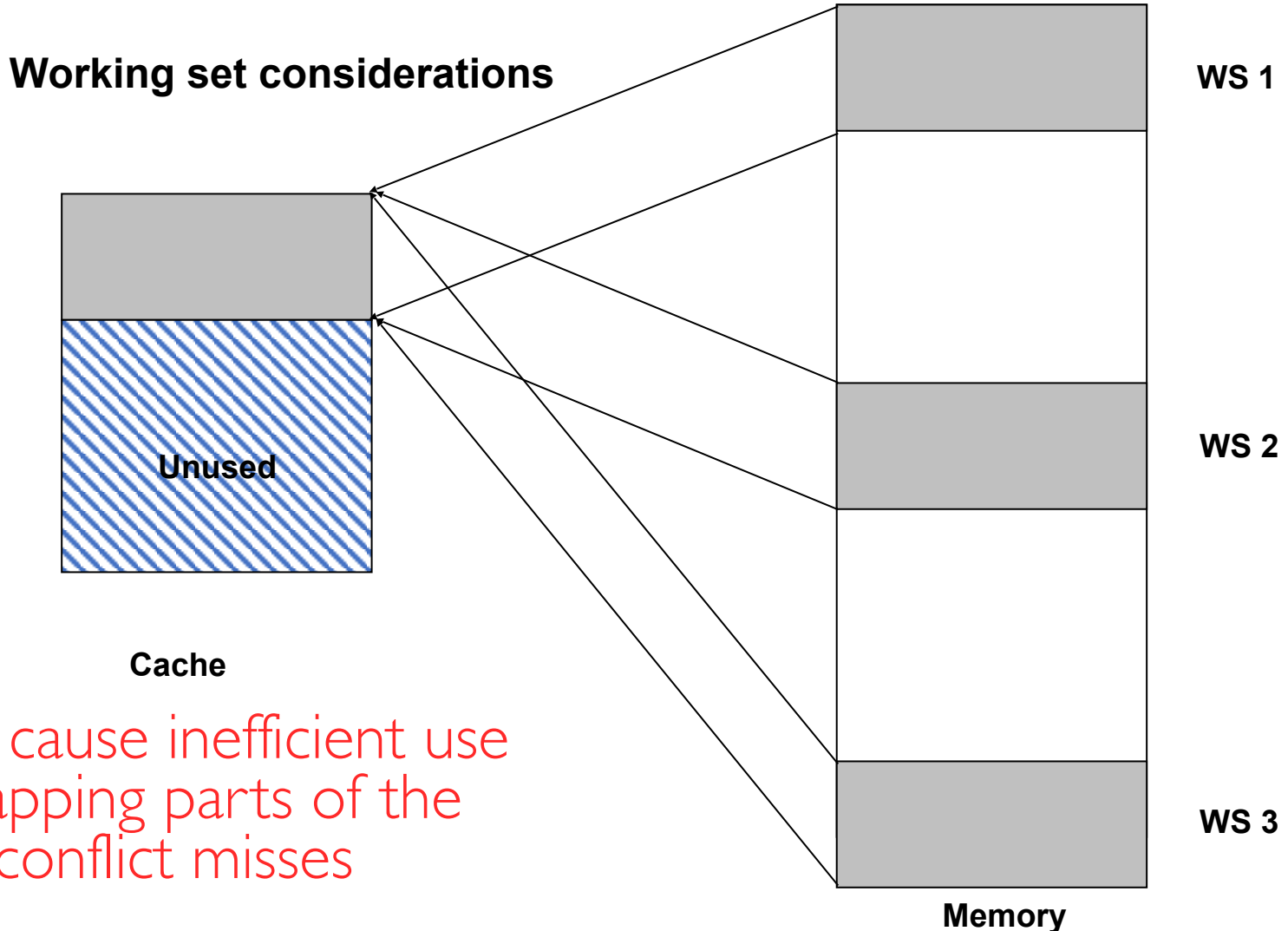
How to improve cache efficiency

- Exploit spatial locality
  - Bring more from memory into cache at a time
- Better organization
  - Exploit working set concept

Working set considerations



Working set considerations



Direct mapping can cause inefficient use of cache with overlapping parts of the working set due to conflict misses

What would be best?

- Allow any memory block to be brought into any cache block

What would be best?

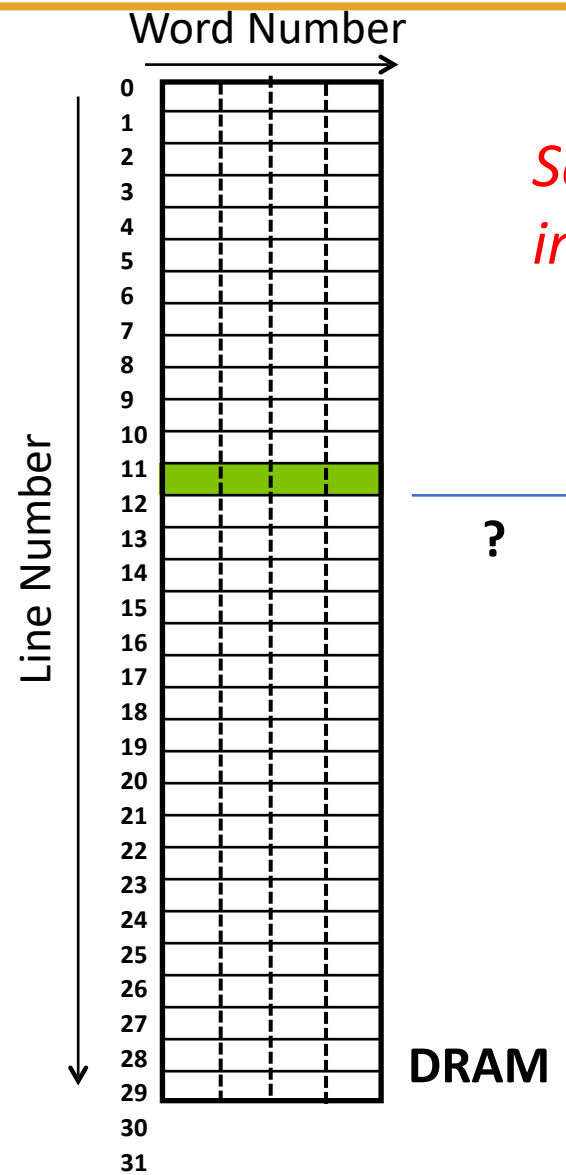
- Allow any memory block to be brought into any cache block
- This is similar to the ability of mapping any virtual page to any available physical page frame

What would be best?

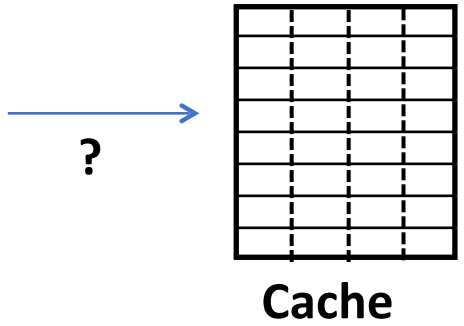
- Allow any memory block to be brought into any cache block
- This is similar to the ability of mapping any virtual page to any available physical page frame

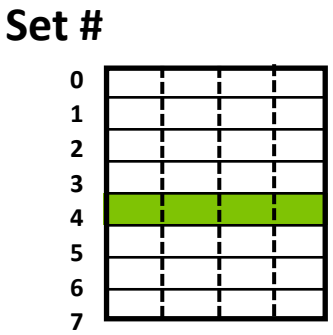
➔ Fully associative mapping

Cache Placement

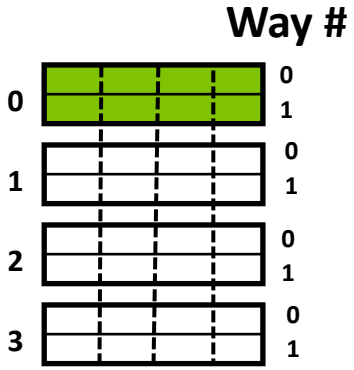


Searching is easy but inefficient use of space



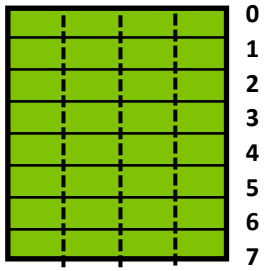


Direct Mapped
one location



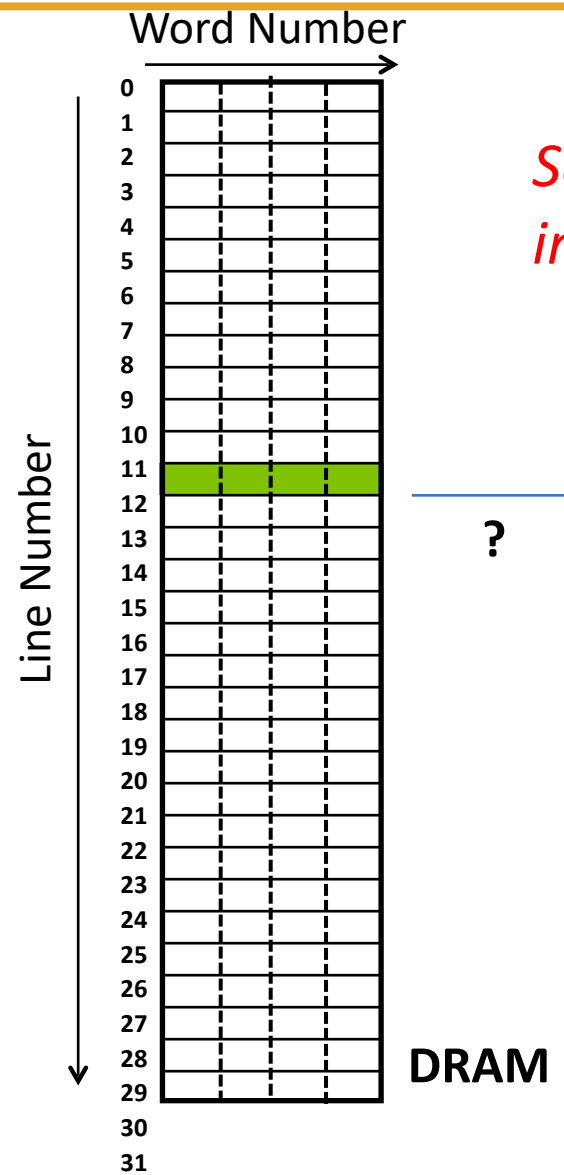
(2-way) Set Associative
2 locations

Searching is more expensive



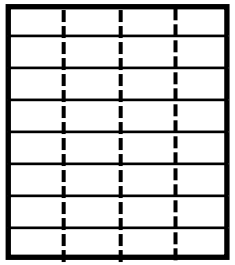
Fully Associative
anywhere

Cache Placement



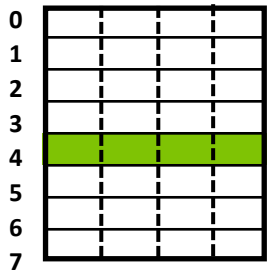
Searching is easy but inefficient use of space

?



Cache

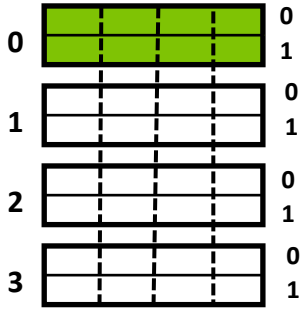
Set #



Direct Mapped

one location

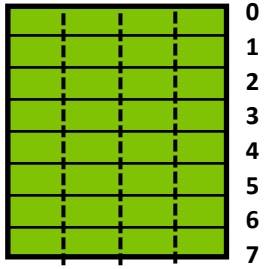
Way #



(2-way) Set Associative

2 locations

Searching is more expensive



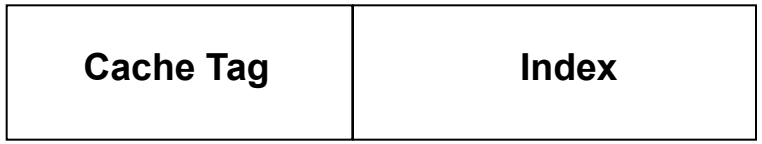
Fully Associative

anywhere

Address interpretation in FA cache

Cache Tag	Index

Address interpretation in FA cache



- No splitting memory addresses into "index" and "tag"

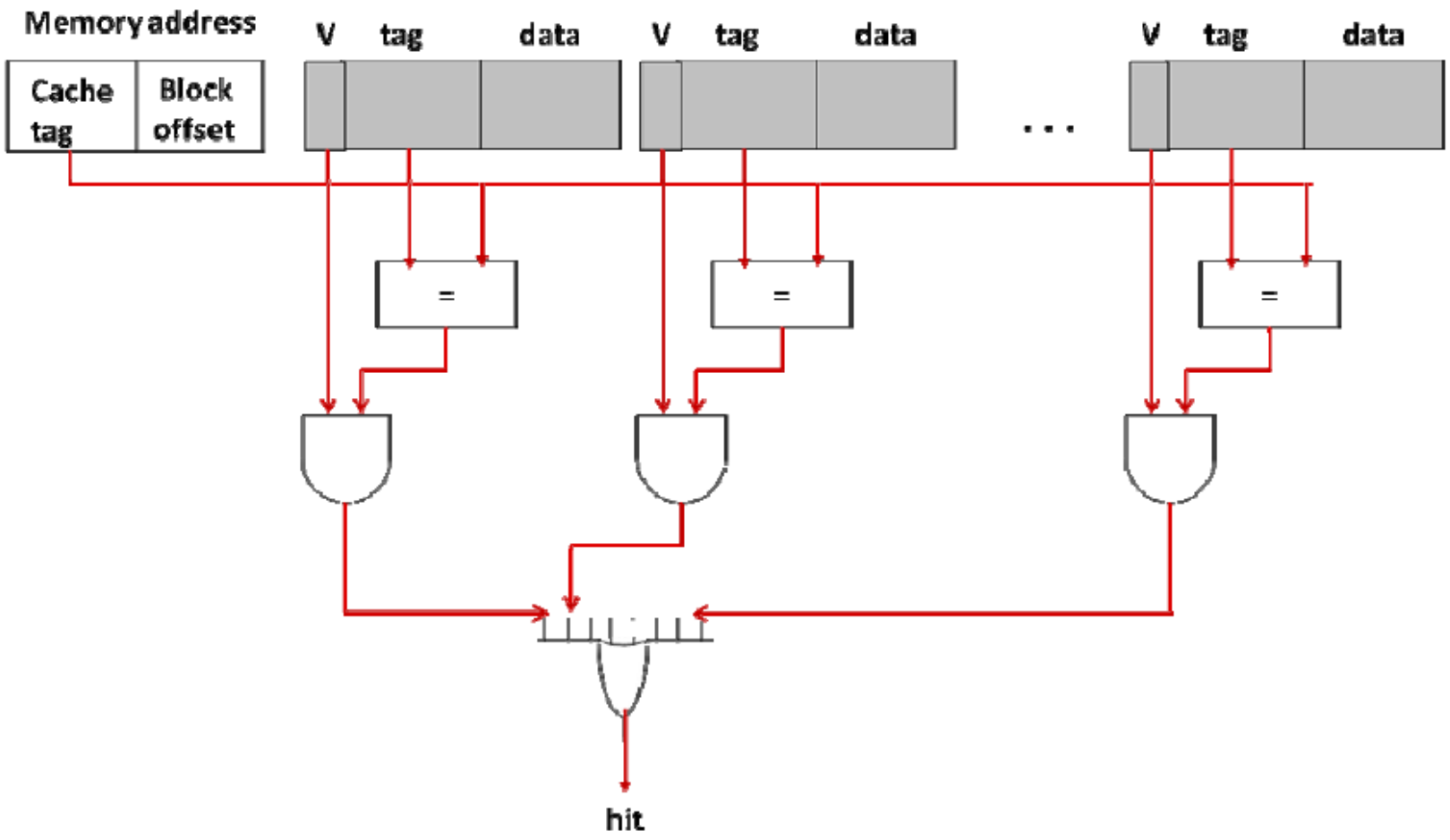
Address interpretation in FA cache

Cache Tag	Index

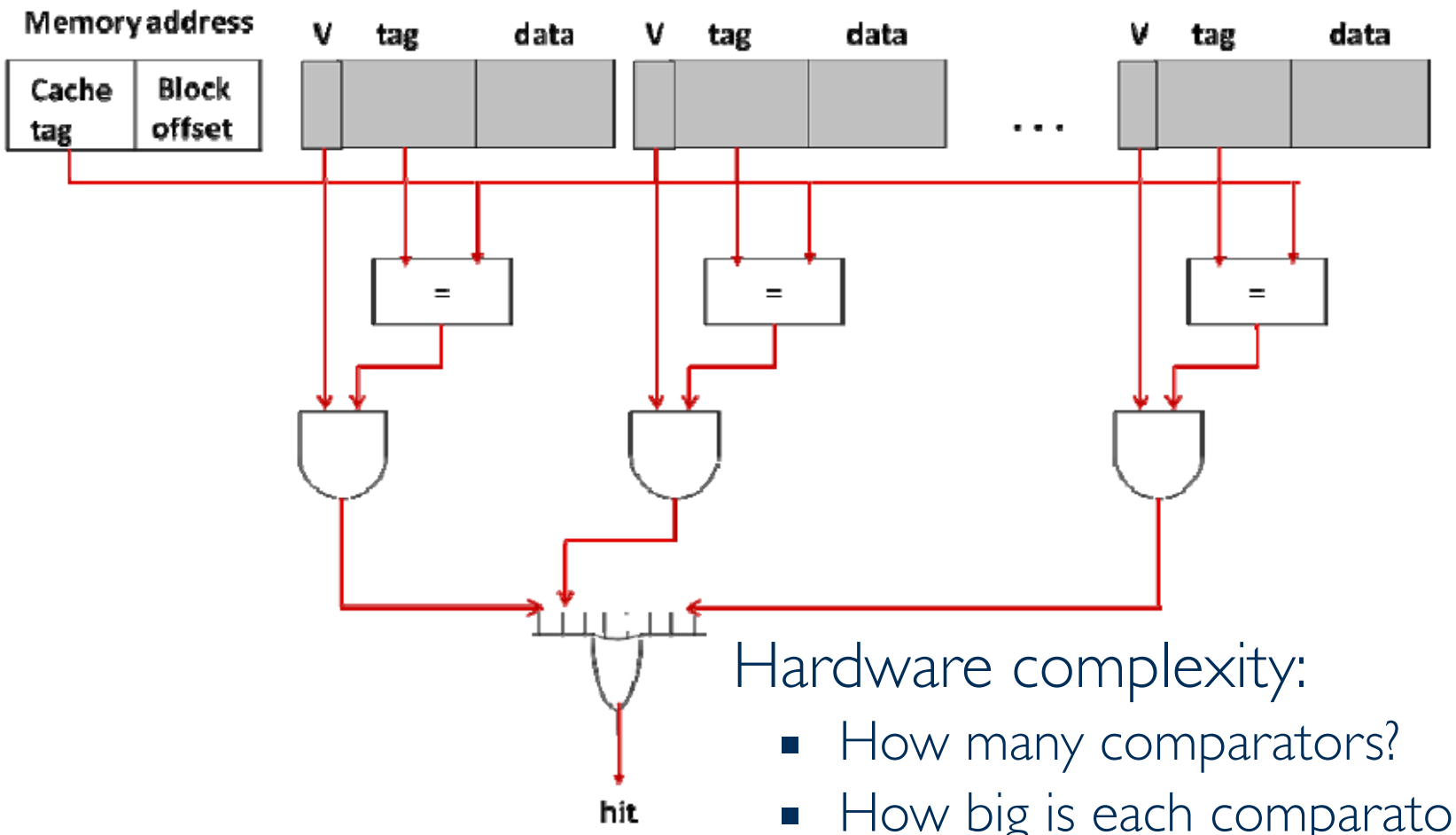
- No splitting memory addresses into "index" and "tag"
- It all becomes tag!

Cache Tag

Fully associative cache circuitry



Fully associative cache circuitry



Hardware complexity:

- How many comparators?
- How big is each comparator?

Cache organizations

- Fully associative cache ➔
  - Too much hardware complexity
  - Most flexible

- Direct mapped cache ➔
  - Least hardware complexity
  - Least flexible

Cache organizations

- Fully associative cache ➔
  - Too much hardware complexity
  - Most flexible

- Direct mapped cache ➔
  - Least hardware complexity
  - Least flexible

- Can we do better? Is there a compromise?

Cache organizations

- Fully associative cache ➔
  - Too much hardware complexity
  - Most flexible

- Direct mapped cache ➔
  - Least hardware complexity
  - Least flexible

- Can we do better? Is there a compromise?
- Yes! It's called a set-associative cache

Cache organizations

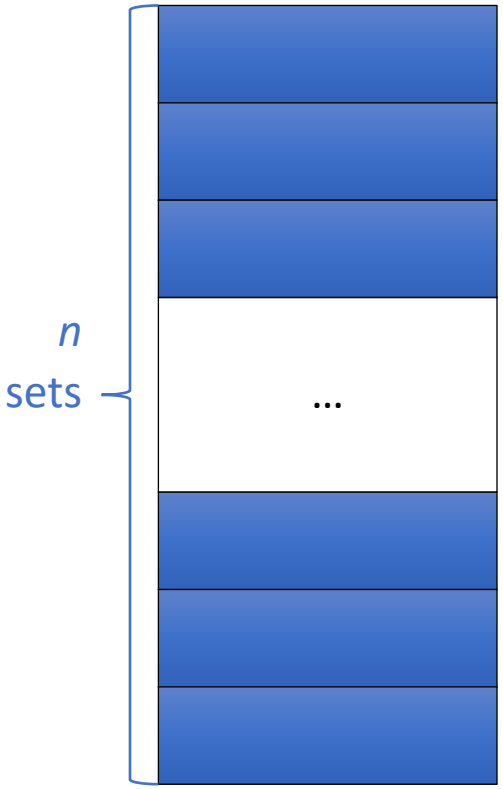
- Fully associative cache ➔
  - Too much hardware complexity
  - Most flexible

- Direct mapped cache ➔
  - Least hardware complexity
  - Least flexible

- Can we do better? Is there a compromise?
- Yes! It's called a set-associative cache
- Direct-mapped and fully-associative caches are cases of a set-associative cache on opposite ends of the spectrum!

Generalization?

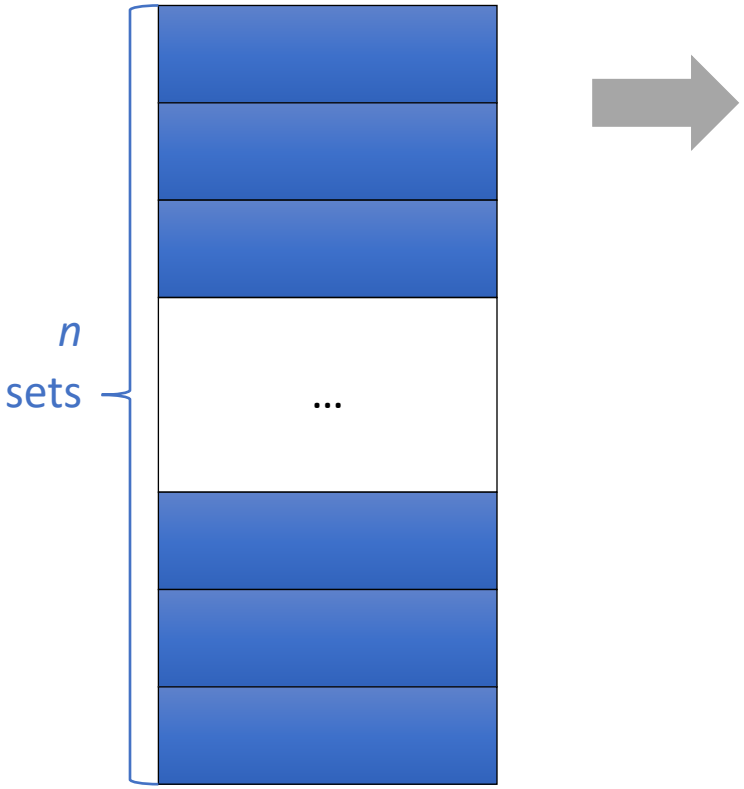
Direct Mapped



Fully Associative

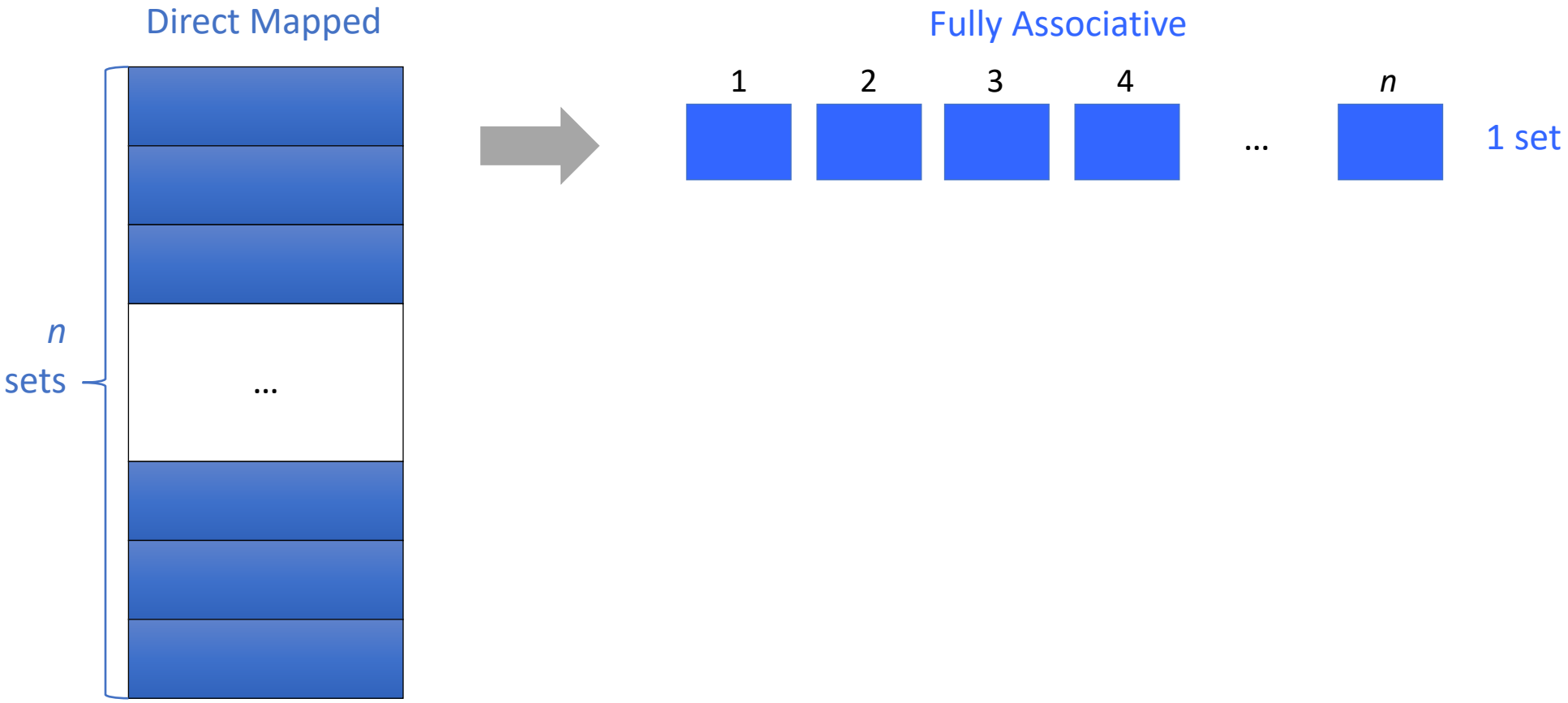
Generalization?

Direct Mapped

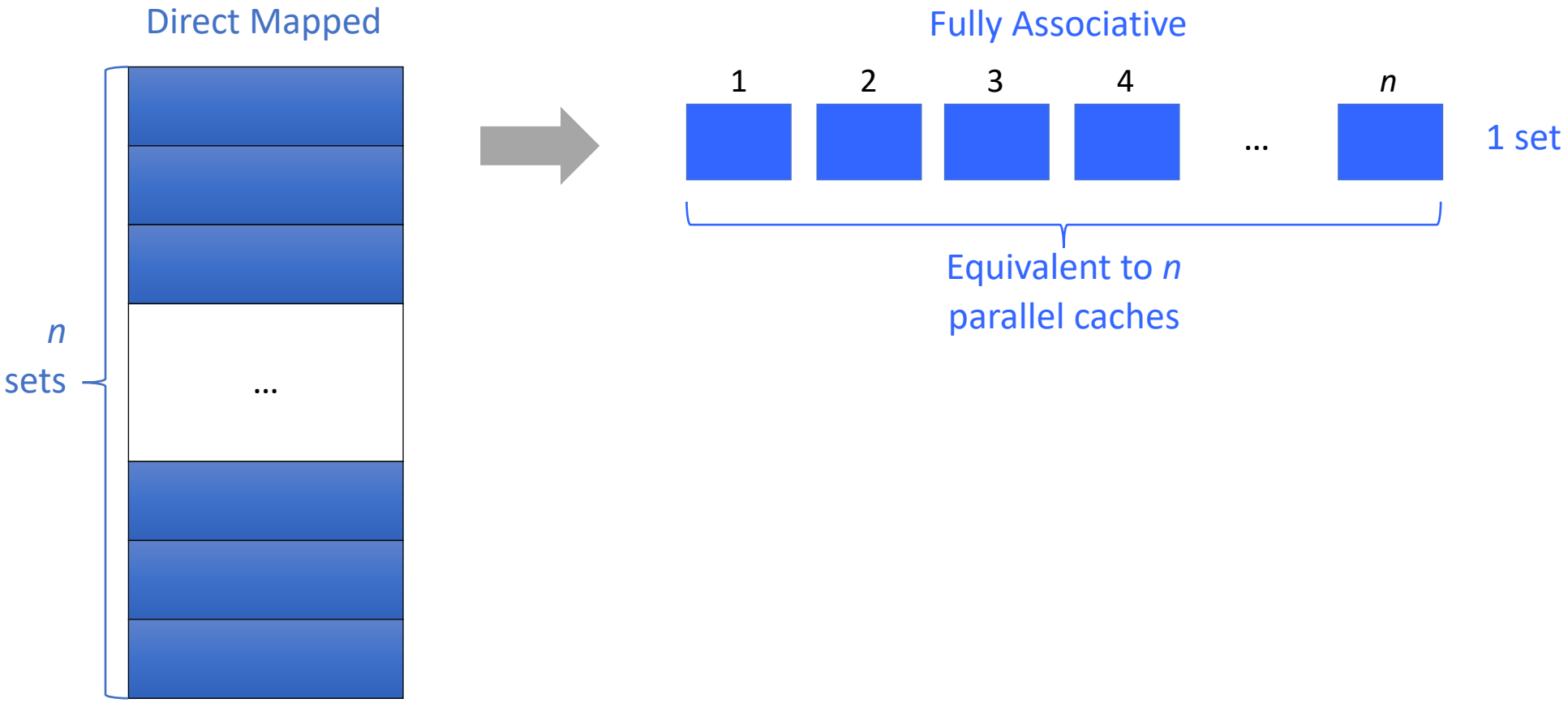


Fully Associative

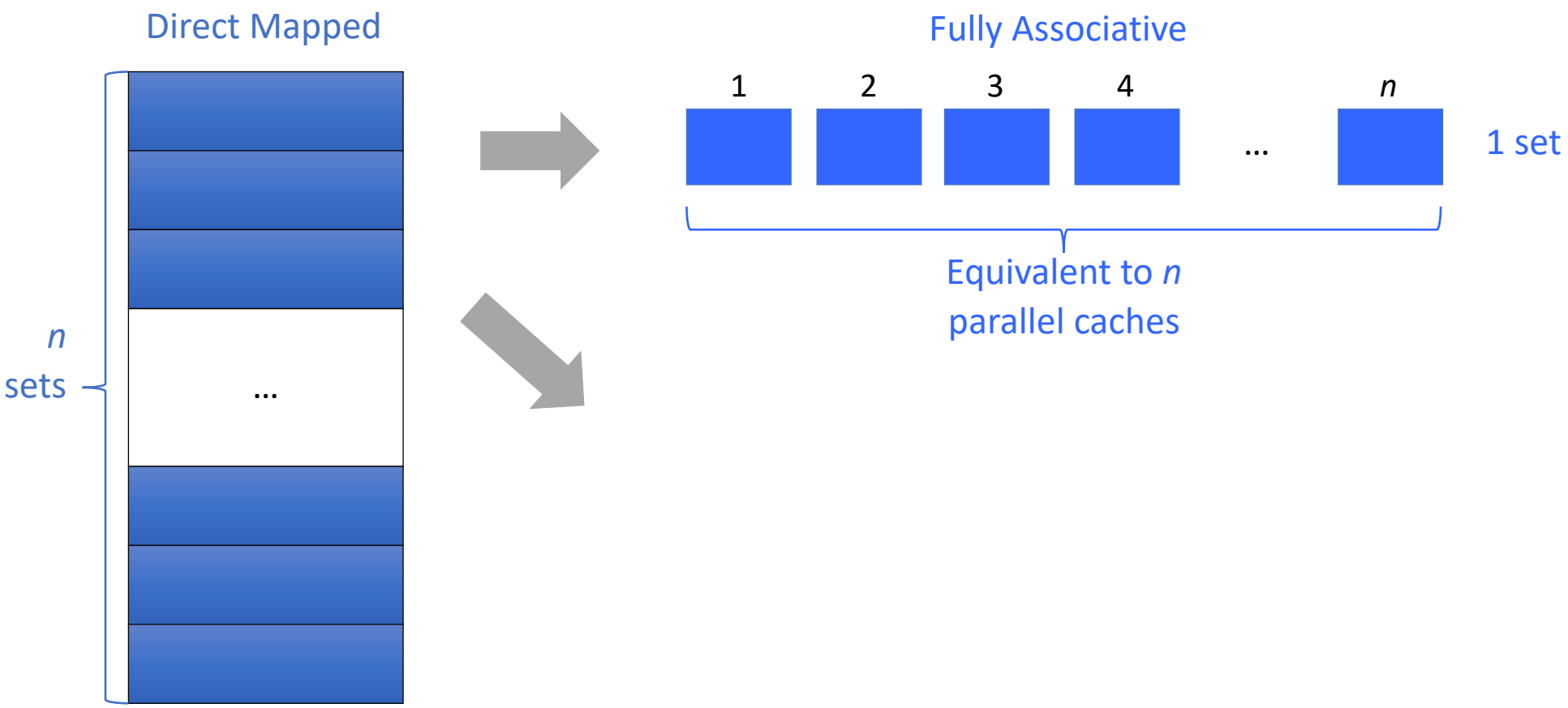
Generalization?



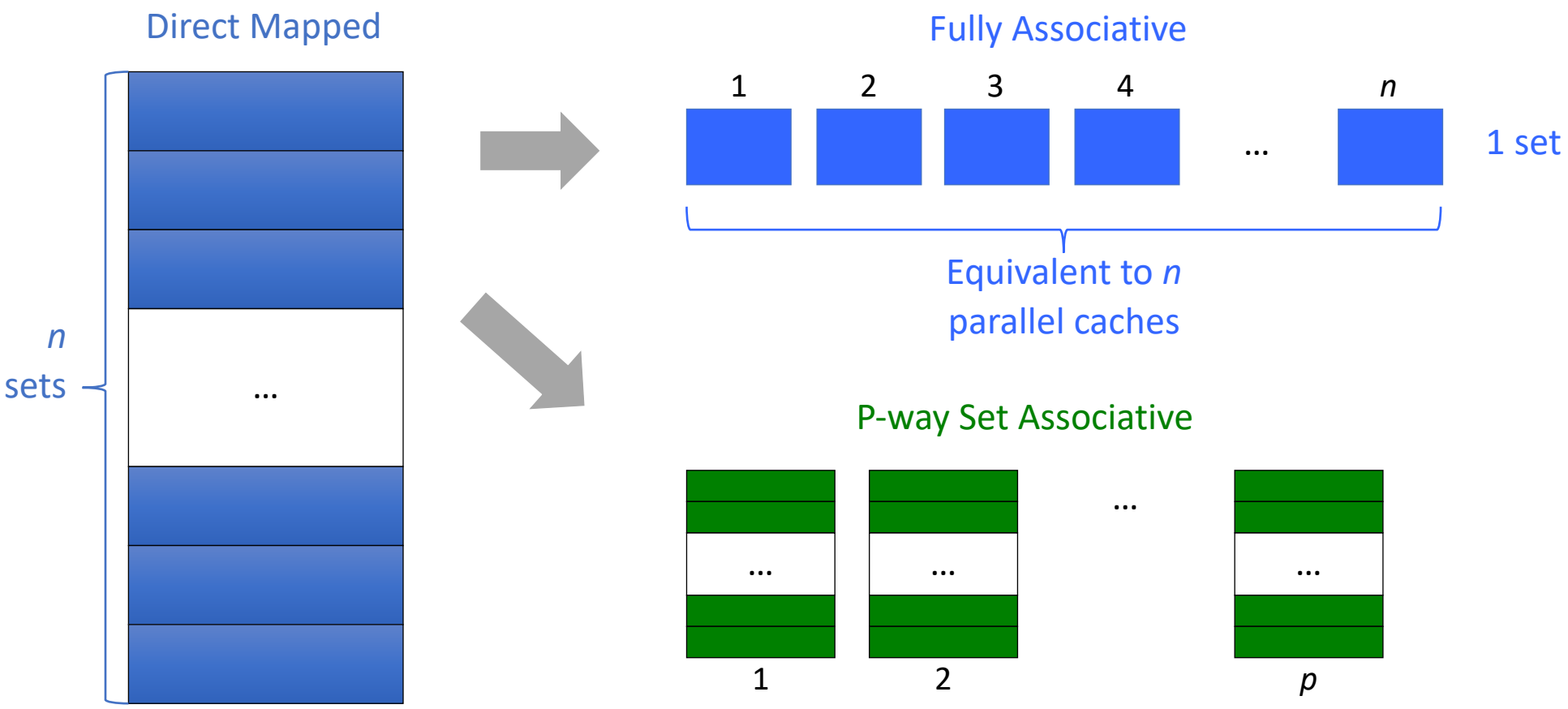
Generalization?



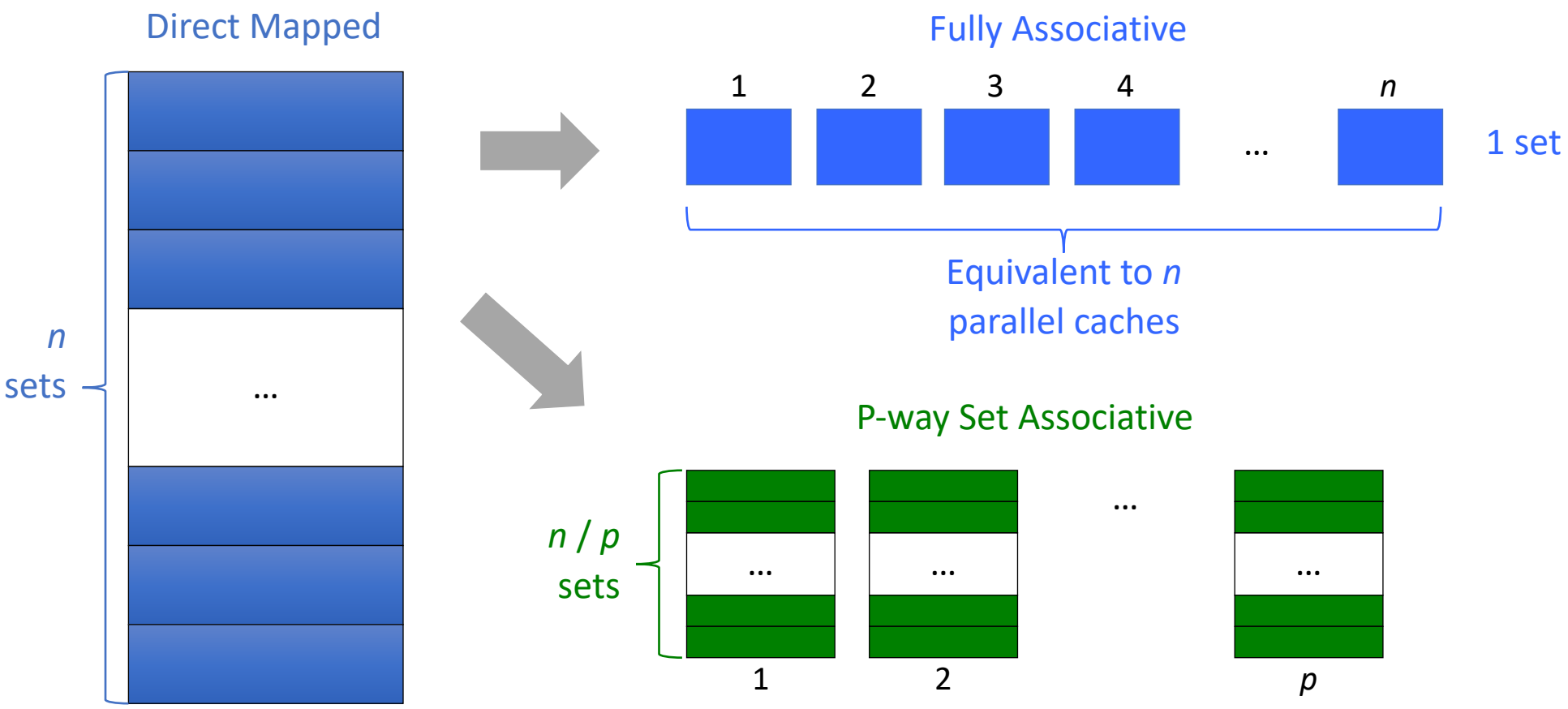
Generalization?



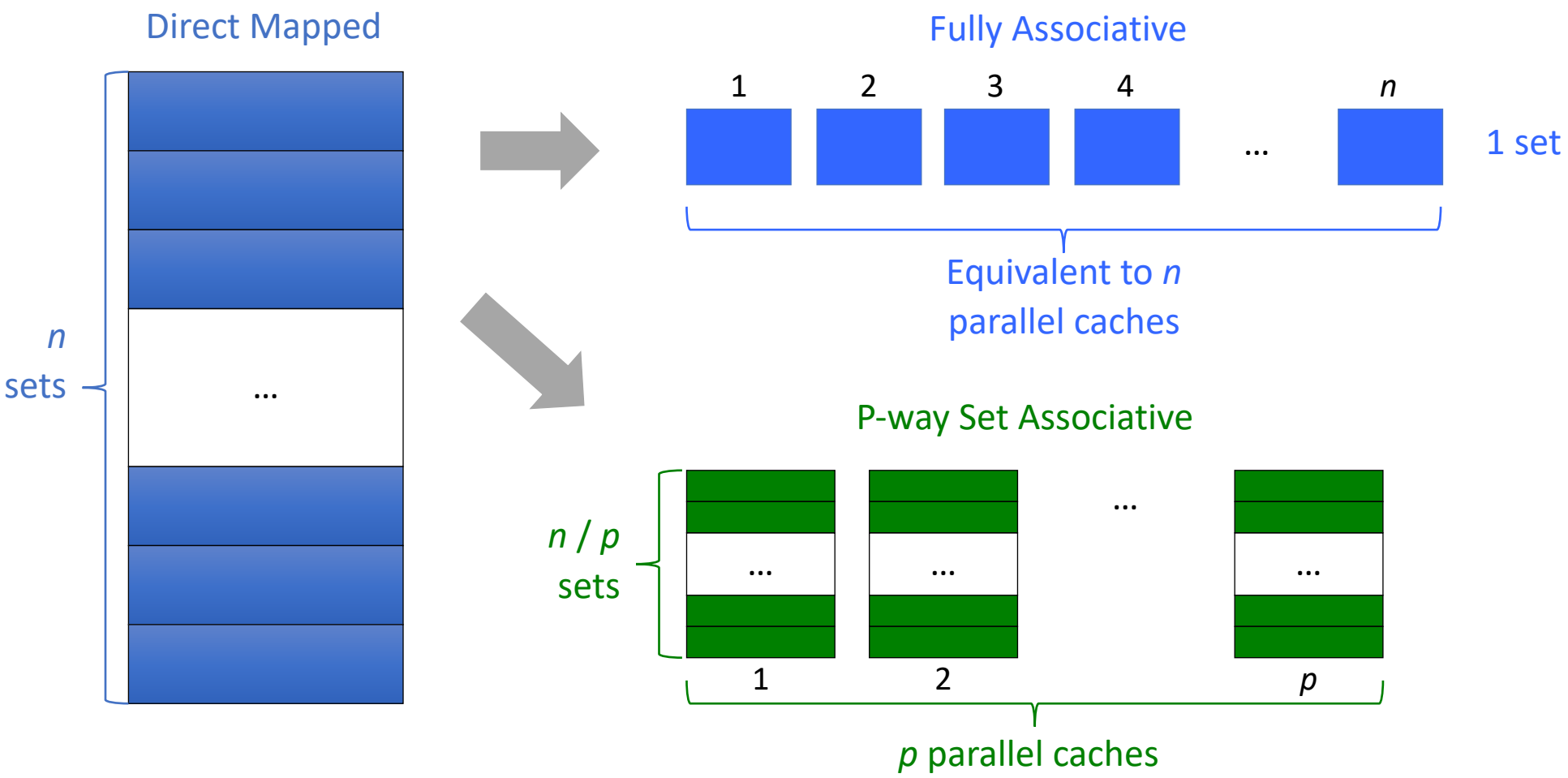
Generalization?



Generalization?



Generalization?



Iso-capacity cache comparison

Iso-capacity cache comparison

	V	Tag	data
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

(a) Direct-mapped cache

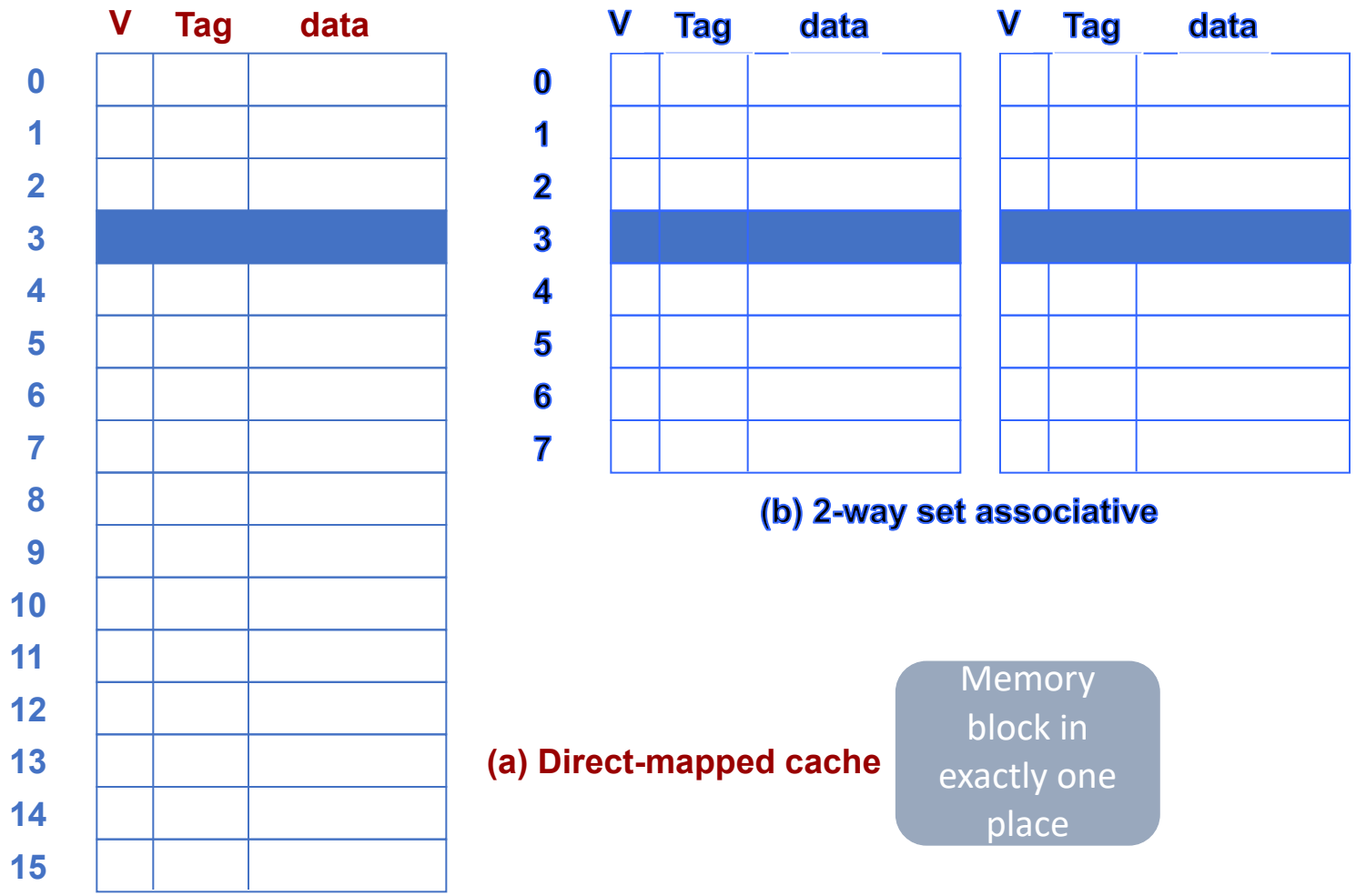
Iso-capacity cache comparison

	V	Tag	data
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

(a) Direct-mapped cache

Memory block in exactly one place

Iso-capacity cache comparison

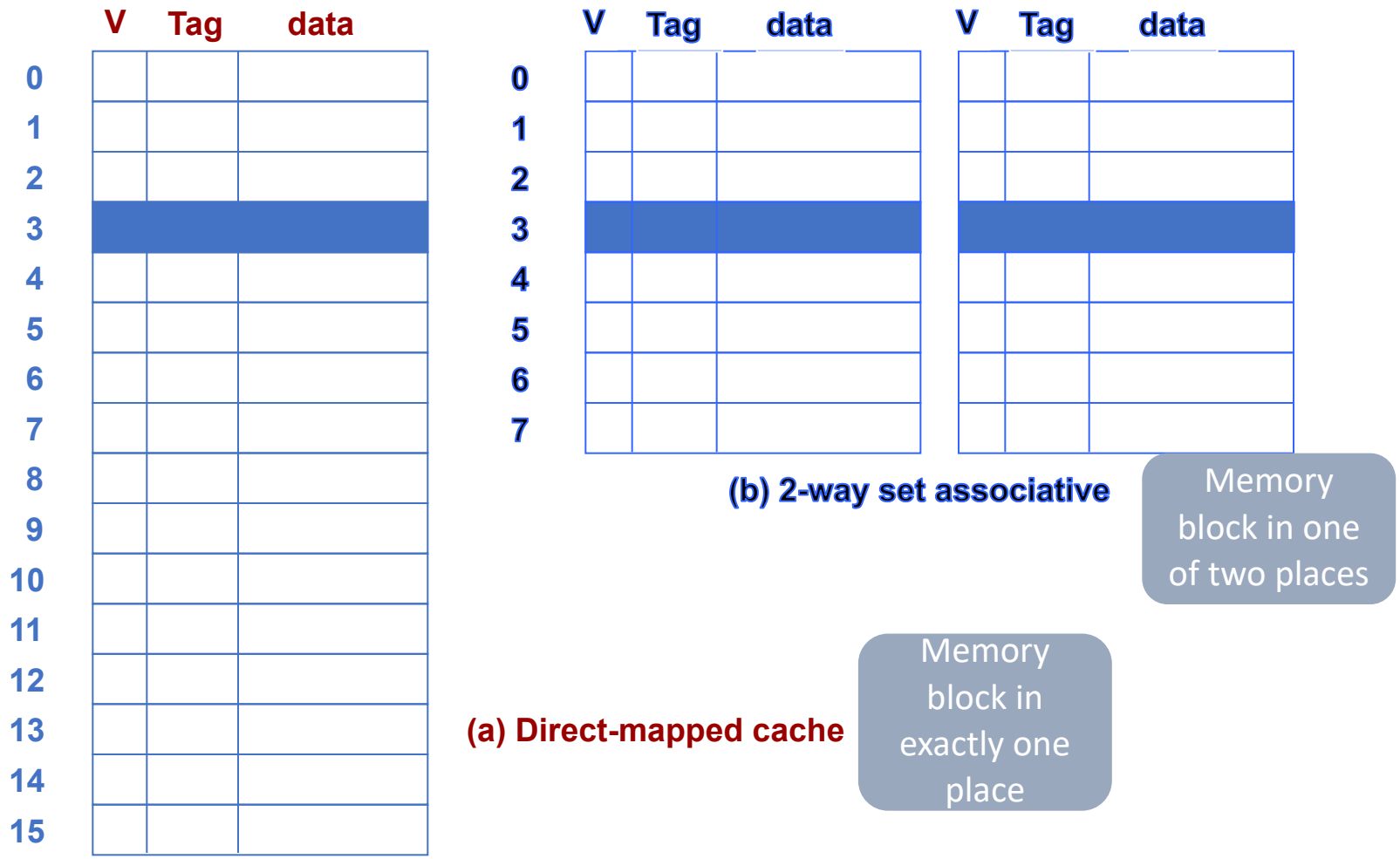


(b) 2-way set associative

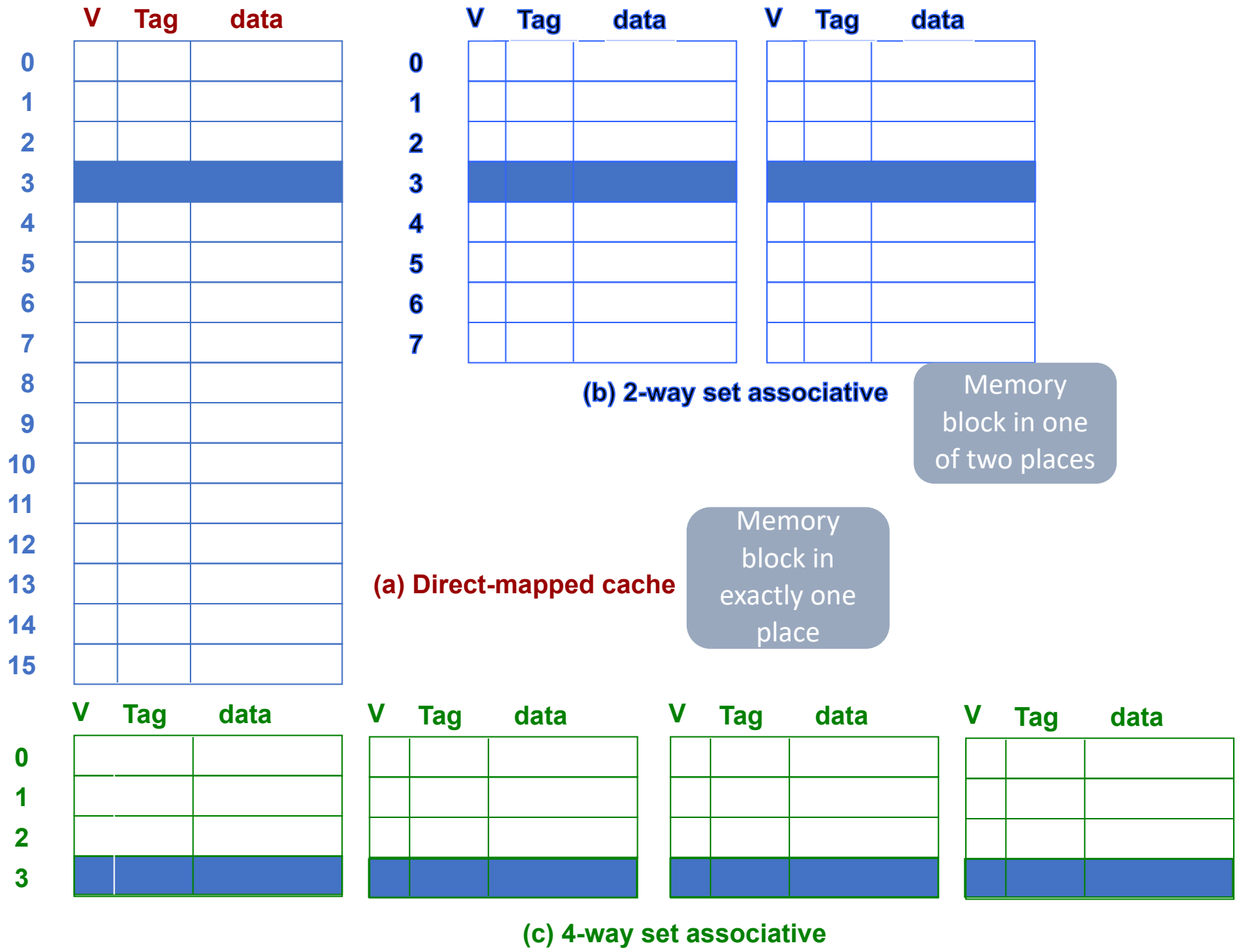
(a) Direct-mapped cache

Memory block in exactly one place

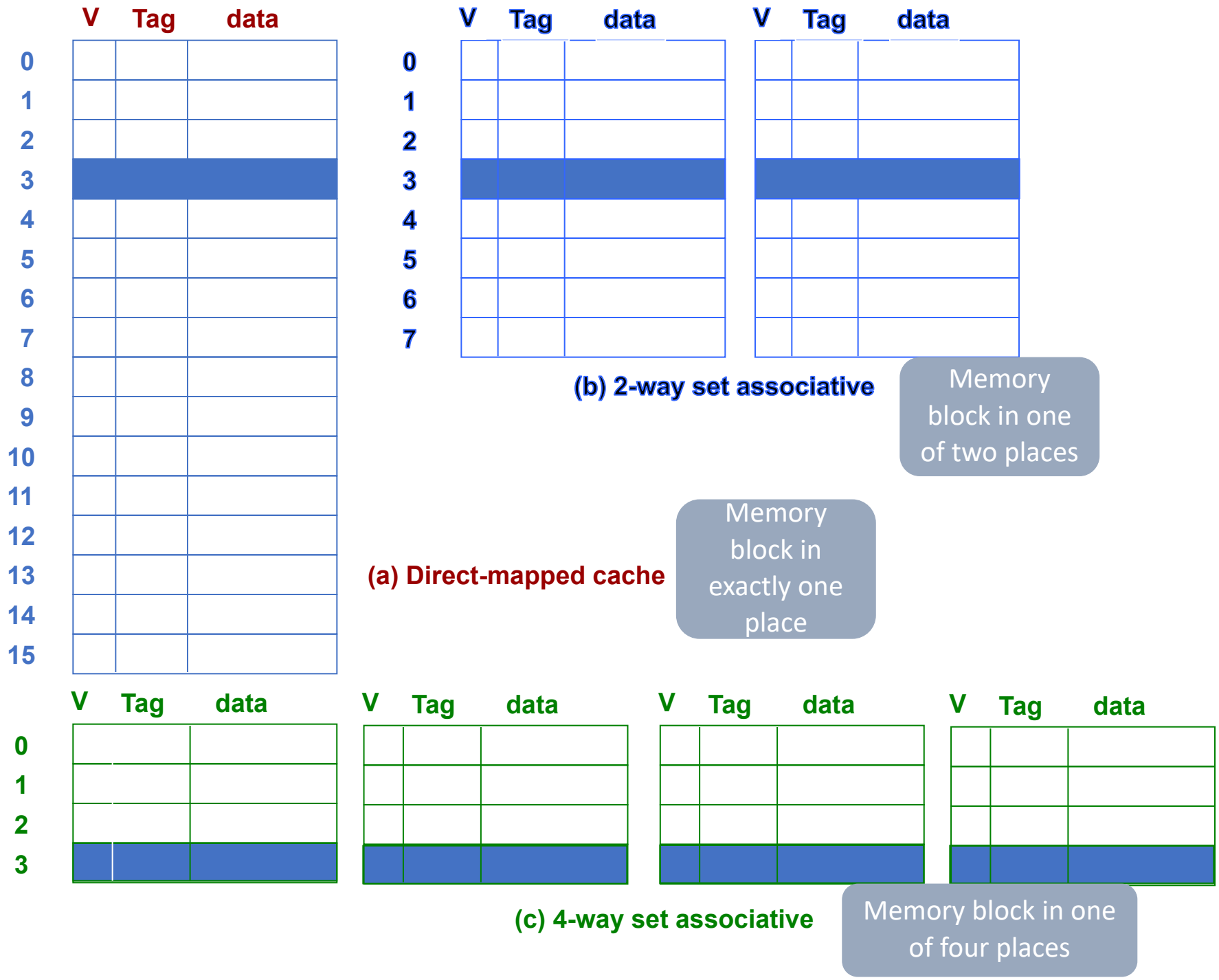
Iso-capacity cache comparison



Iso-capacity cache comparison



Iso-capacity cache comparison



Terminology clarification

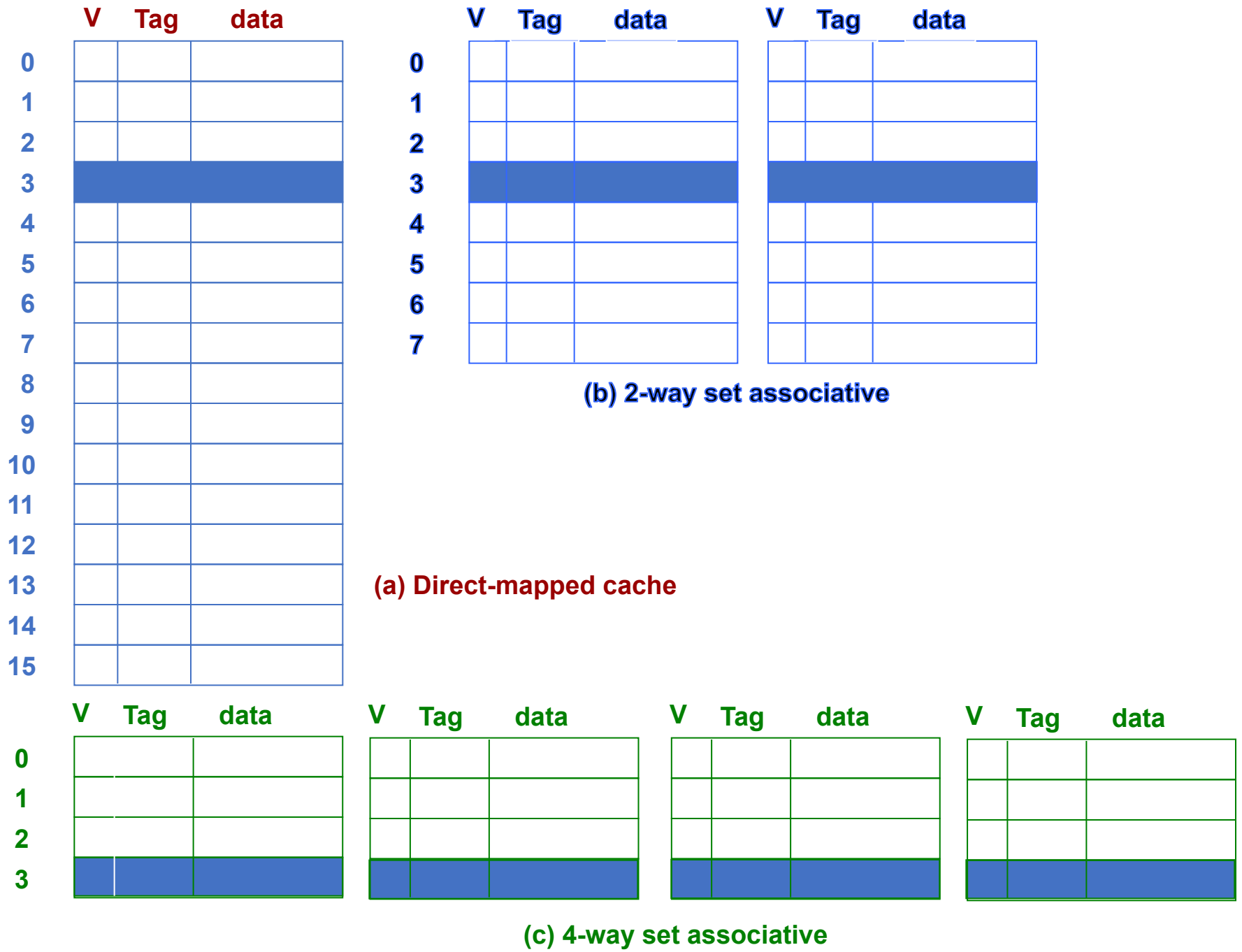
- Unfortunate but…
  - Four different ways of saying the same thing
    - Cache line
    - Cache block
    - Cache entry
    - Cache element

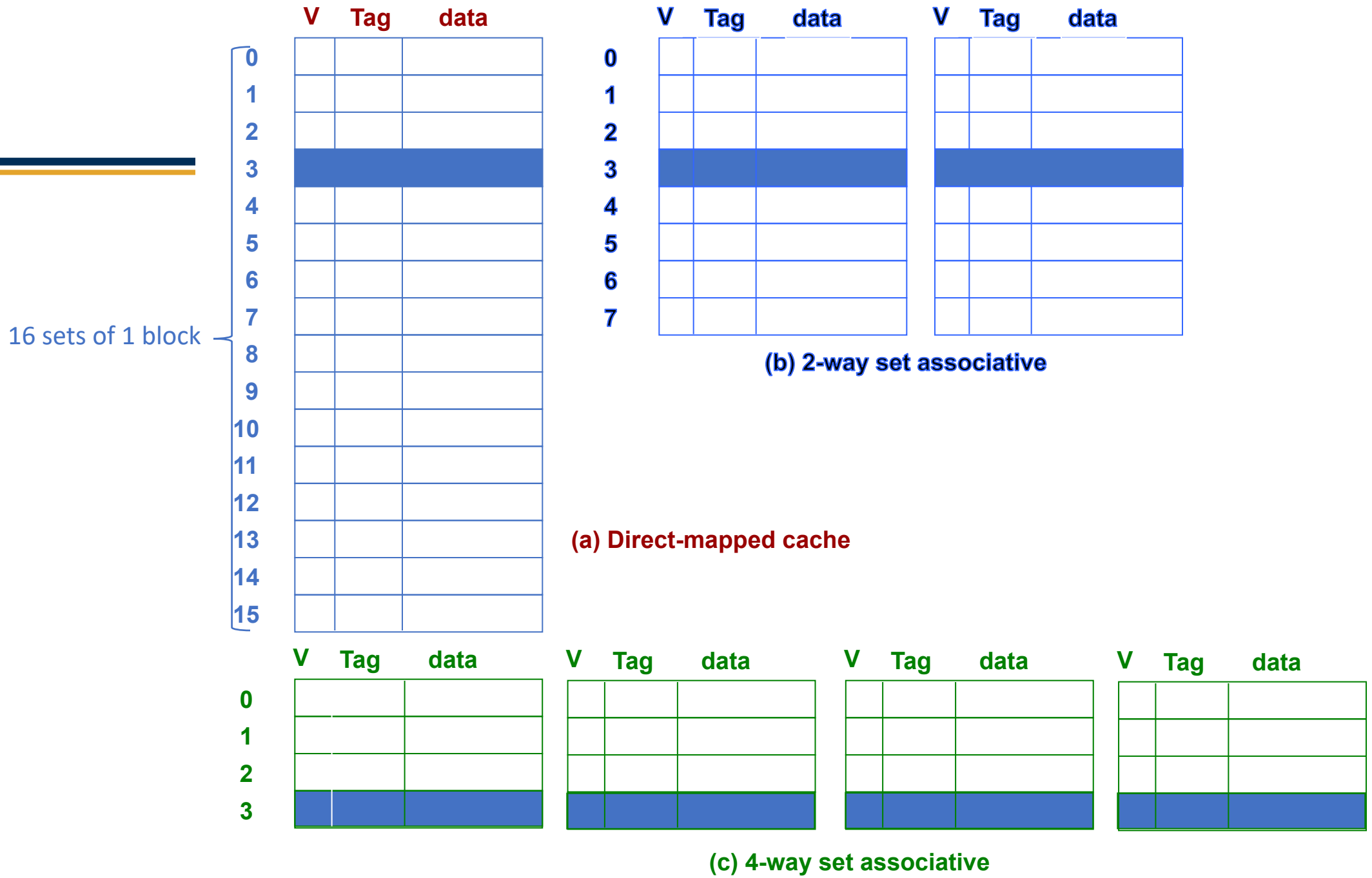
Terminology clarification

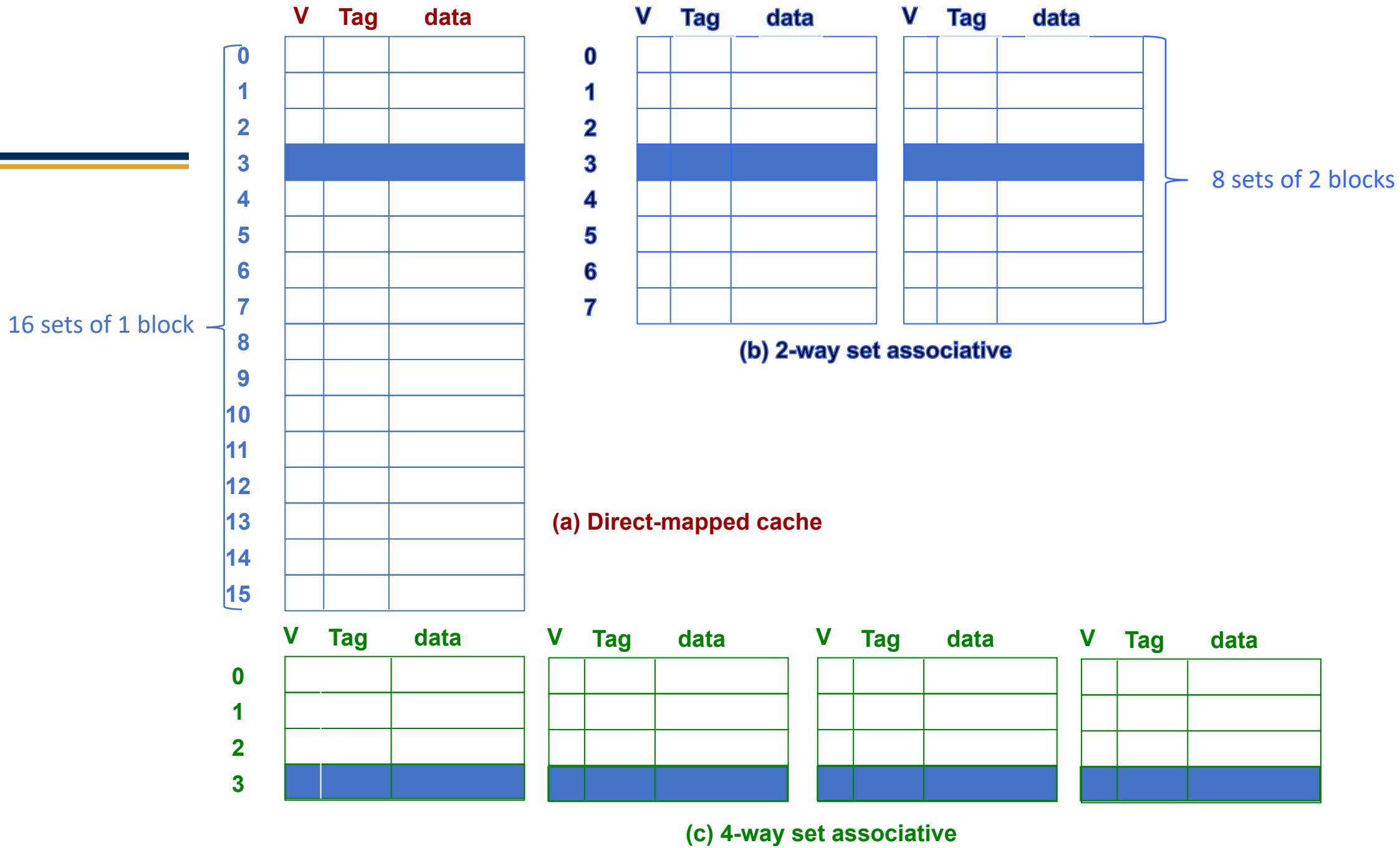
- Unfortunate but…
  - Four different ways of saying the same thing
    - Cache line
    - Cache block
    - Cache entry
    - Cache element
  - All mean the same thing…the basic unit of data transferred into/out of the cache at a time
  - The textbook has a couple of typos in which it erroneously implies cache set is also synonymous to cache block. In addition, from chapter 9.11.3 onwards, almost every occurrence of the term "cache line" should have been "cache set". A cache line is synonymous to a cache block, and a cache set corresponds to a single cache line only in the case of direct-mapped caches!

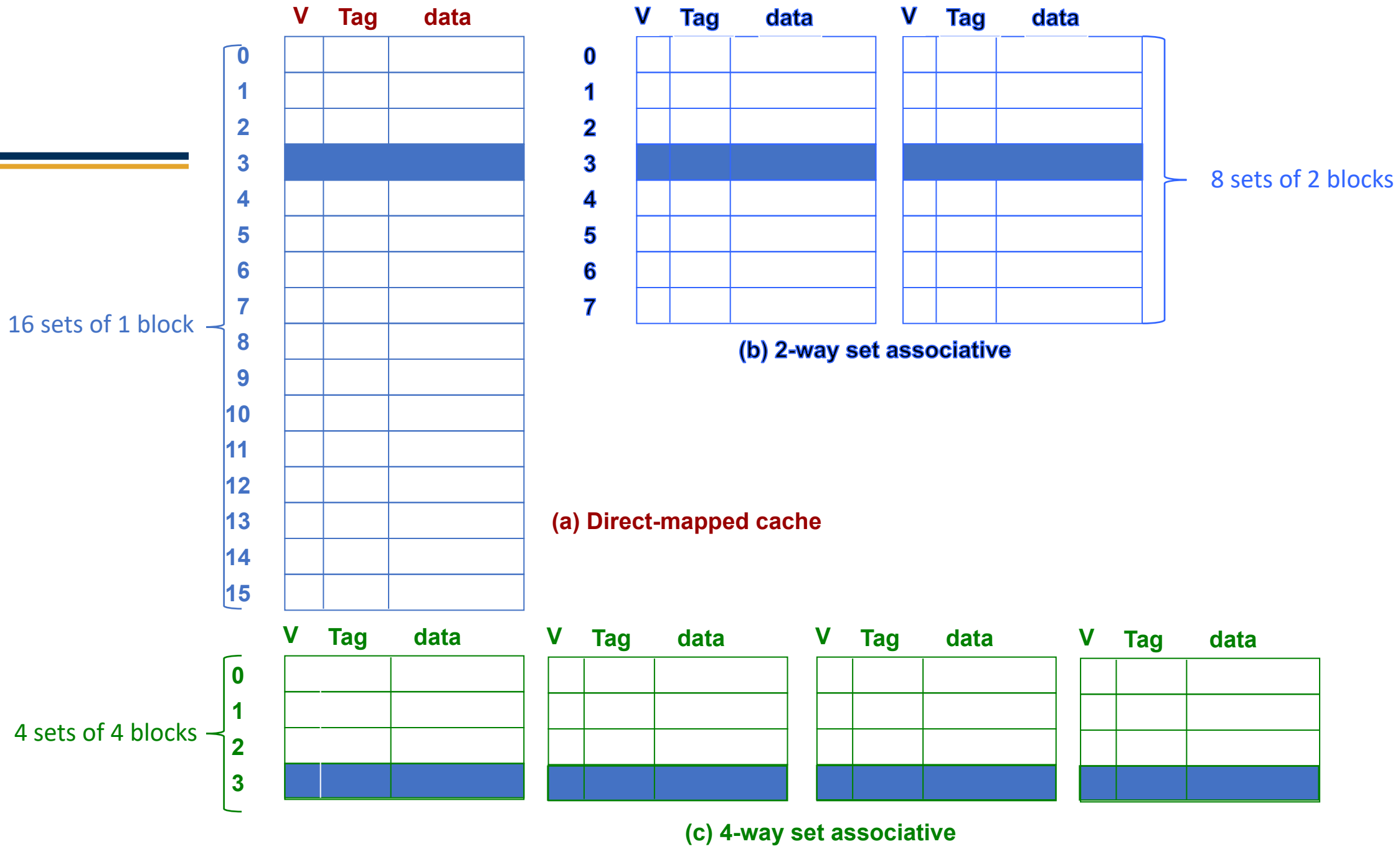
Terminology clarification

- Unfortunate but…
  - Four different ways of saying the same thing
    - Cache line
    - Cache block
    - Cache entry
    - Cache element
  - All mean the same thing…the basic unit of data transferred into/out of the cache at a time
  - The textbook has a couple of typos in which it erroneously implies cache set is also synonymous to cache block. In addition, from chapter 9.11.3 onwards, almost every occurrence of the term "cache line" should have been "cache set". A cache line is synonymous to a cache block, and a cache set corresponds to a single cache line only in the case of direct-mapped caches!
- A cache set is a "row" in the cache. The number of blocks per set is determined by the type of the cache
  - Direct mapped: n sets, 1 block
  - P-way set associative: n/p sets, p blocks
  - Fully associative: 1 set, n blocks

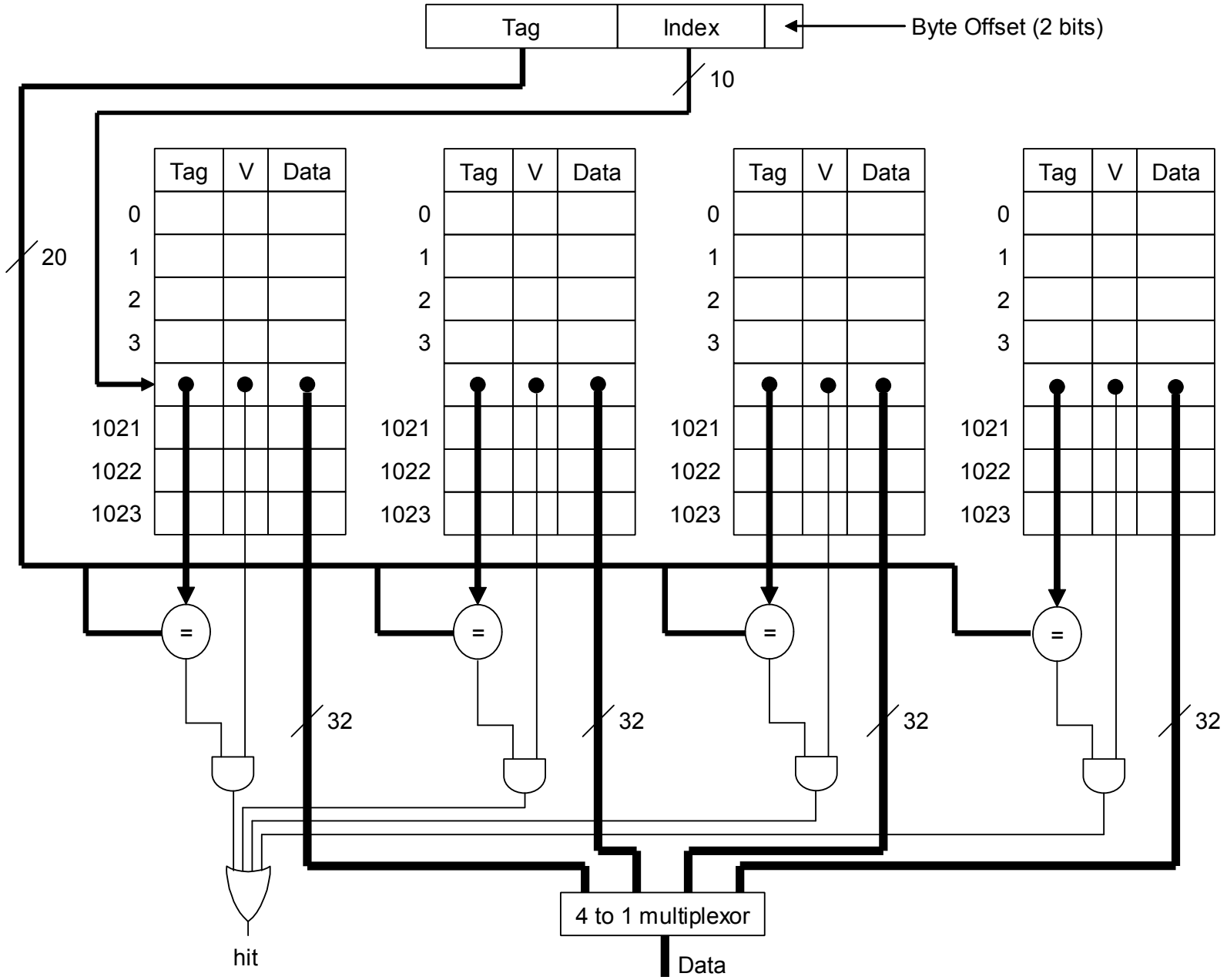

V
Tag
data
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
(a) Direct-mapped cache
V
Tag
data
V
Tag
data
0
1
2
3
4
5
6
7
(b) 2-way set associative
V
Tag
data
V
Tag
data
V
Tag
data
V
Tag
data
0
1
2
3
(c) 4-way set associative


V
Tag
data
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16 sets of 1 block
(a) Direct-mapped cache
(b) 2-way set associative
(c) 4-way set associative


V
Tag
data
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16 sets of 1 block
(a) Direct-mapped cache
V
Tag
data
V
Tag
data
0
1
2
3
4
5
6
7
8 sets of 2 blocks
(b) 2-way set associative
V
Tag
data
V
Tag
data
V
Tag
data
V
Tag
data
0
1
2
3
(c) 4-way set associative


V
Tag
data
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16 sets of 1 block
(a) Direct-mapped cache
8 sets of 2 blocks
(b) 2-way set associative
4 sets of 4 blocks
(c) 4-way set associative

4-way SA cache organization

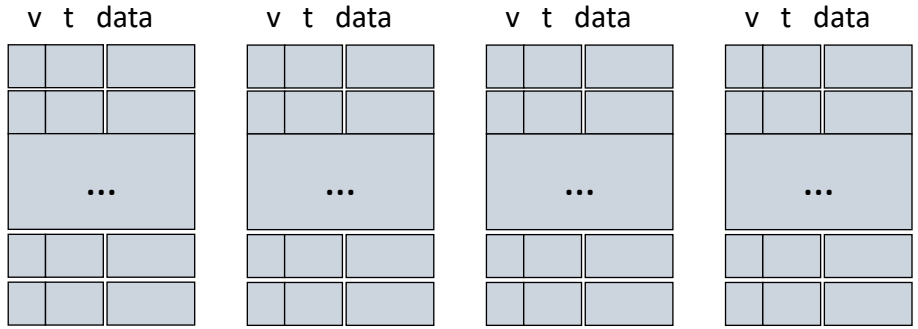


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

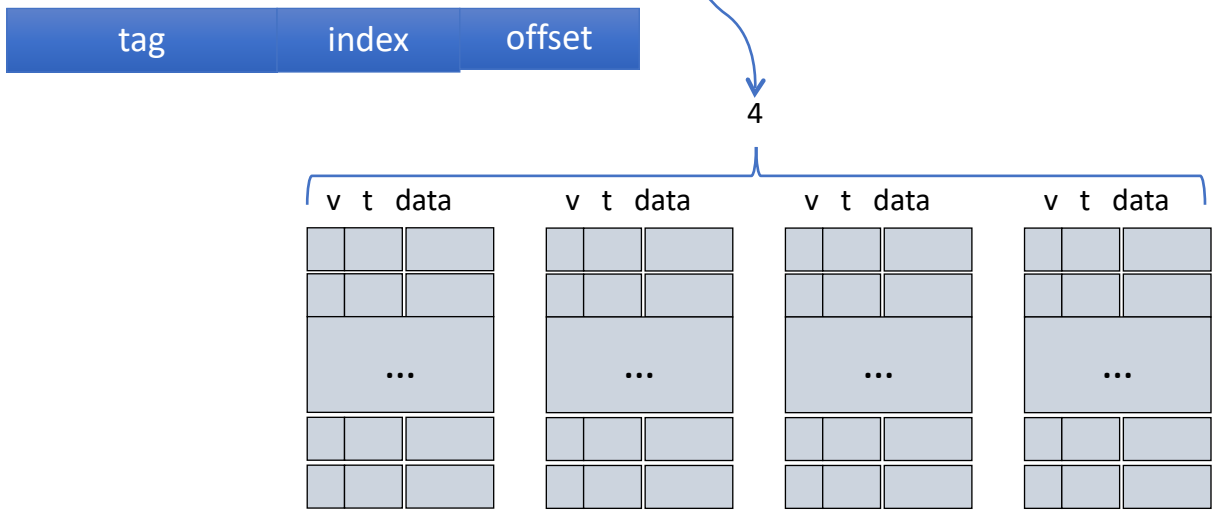


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

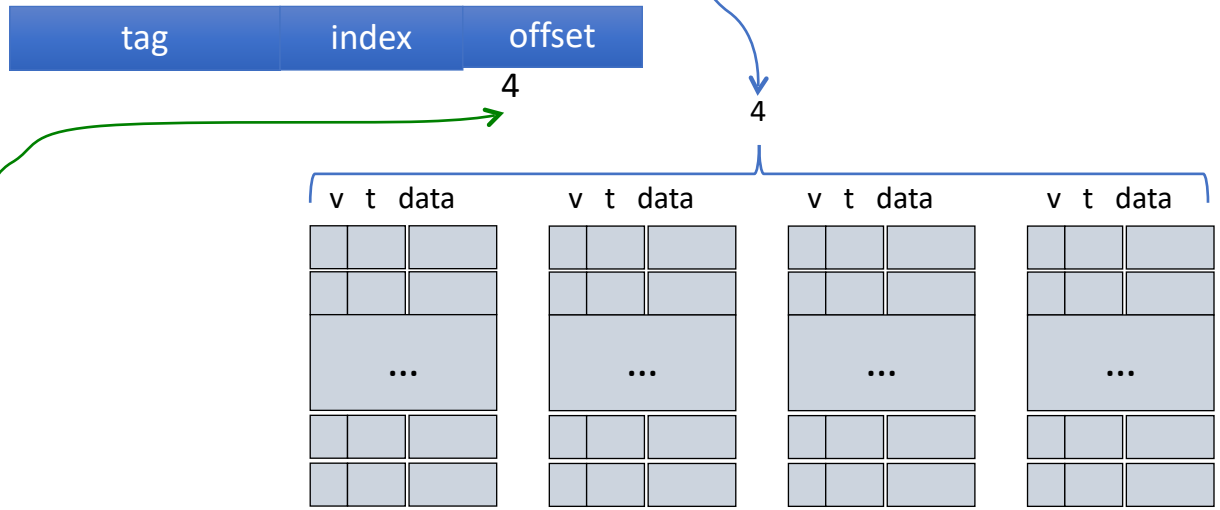


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)



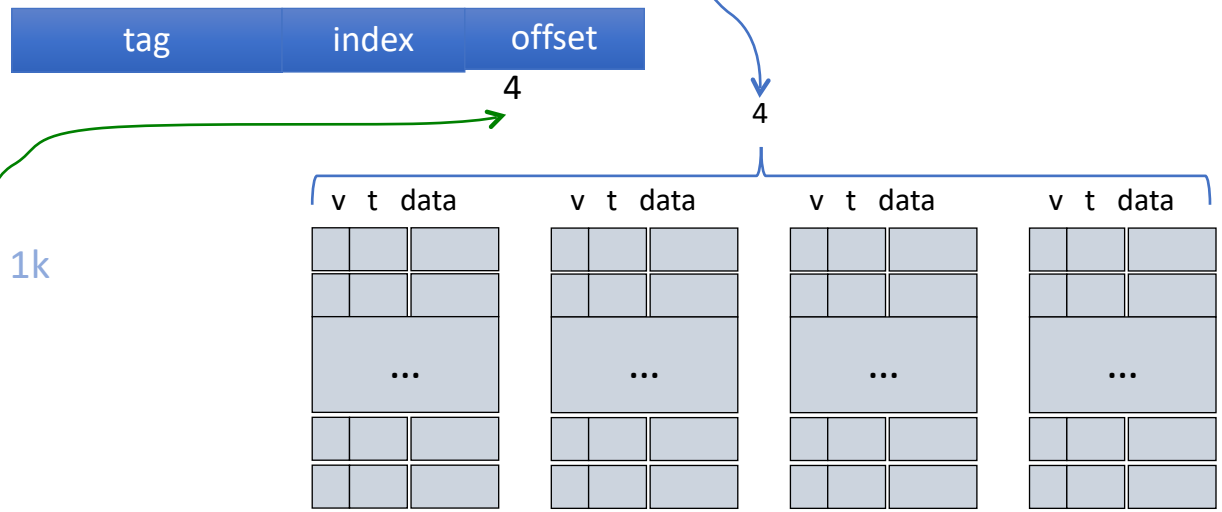
Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

64K/4/16 = 1k

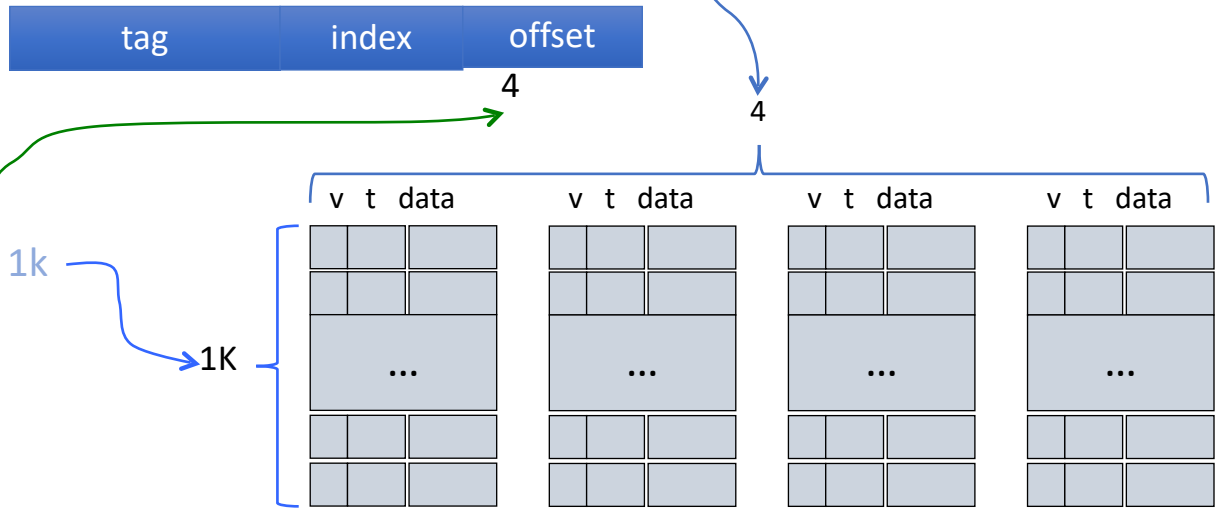


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

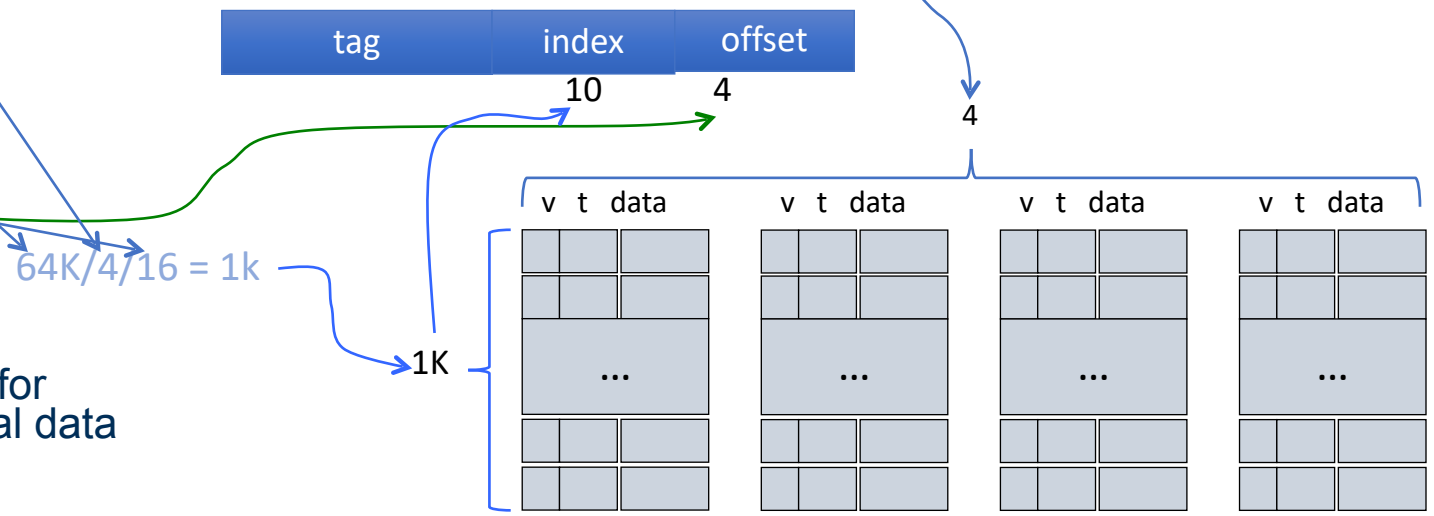


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

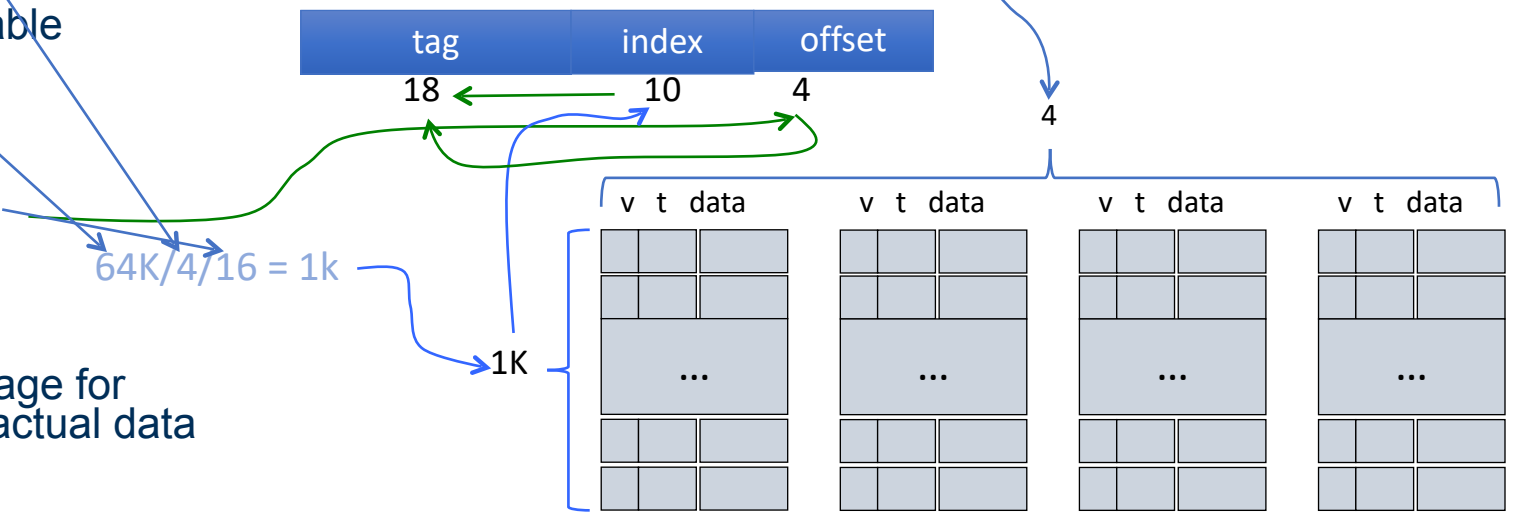


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

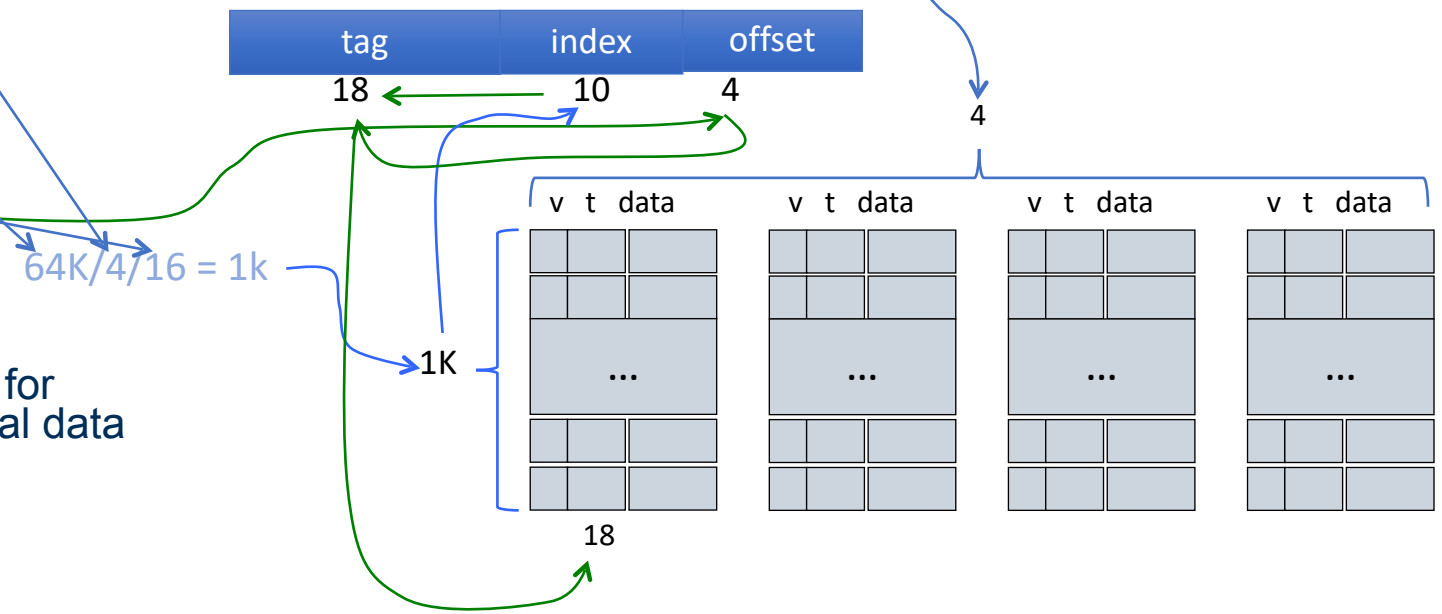


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)



Example: 4-way set associative cache

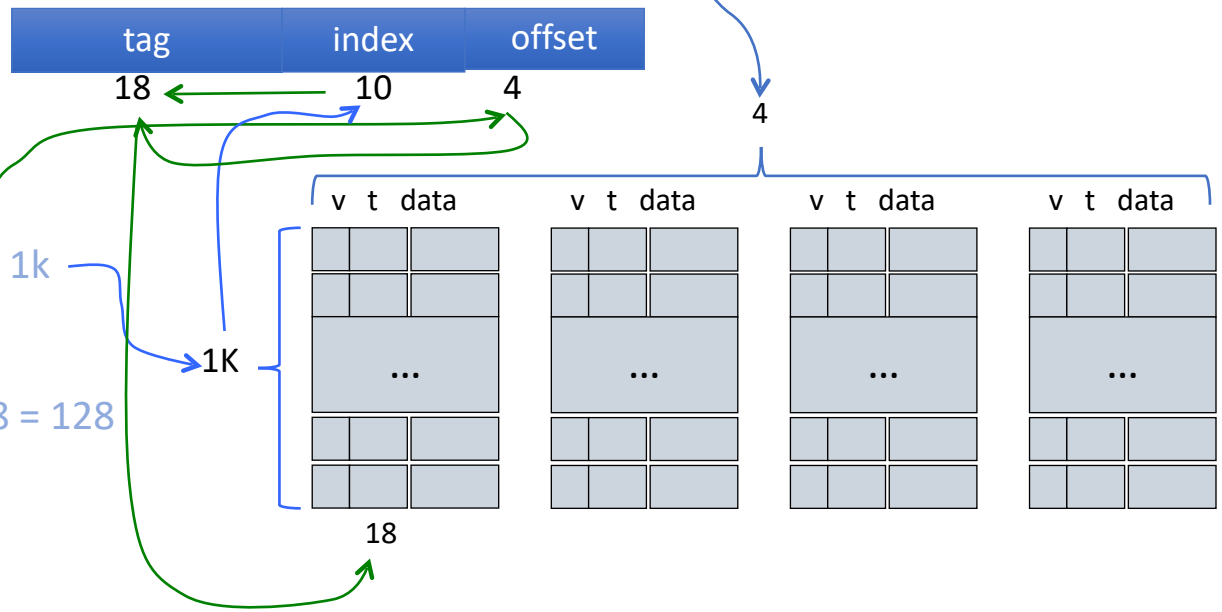
- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

64K/4/16 = 1k

16*8 = 128



Example: 4-way set associative cache

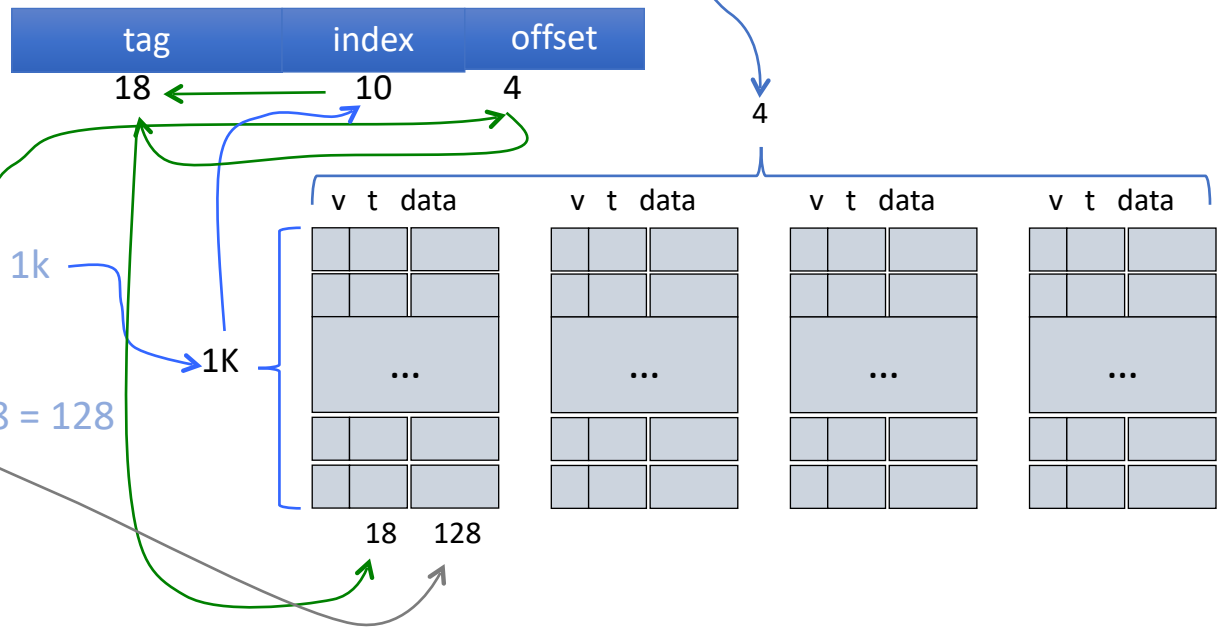
- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

64K/4/16 = 1k

16*8 = 128



Example: 4-way set associative cache

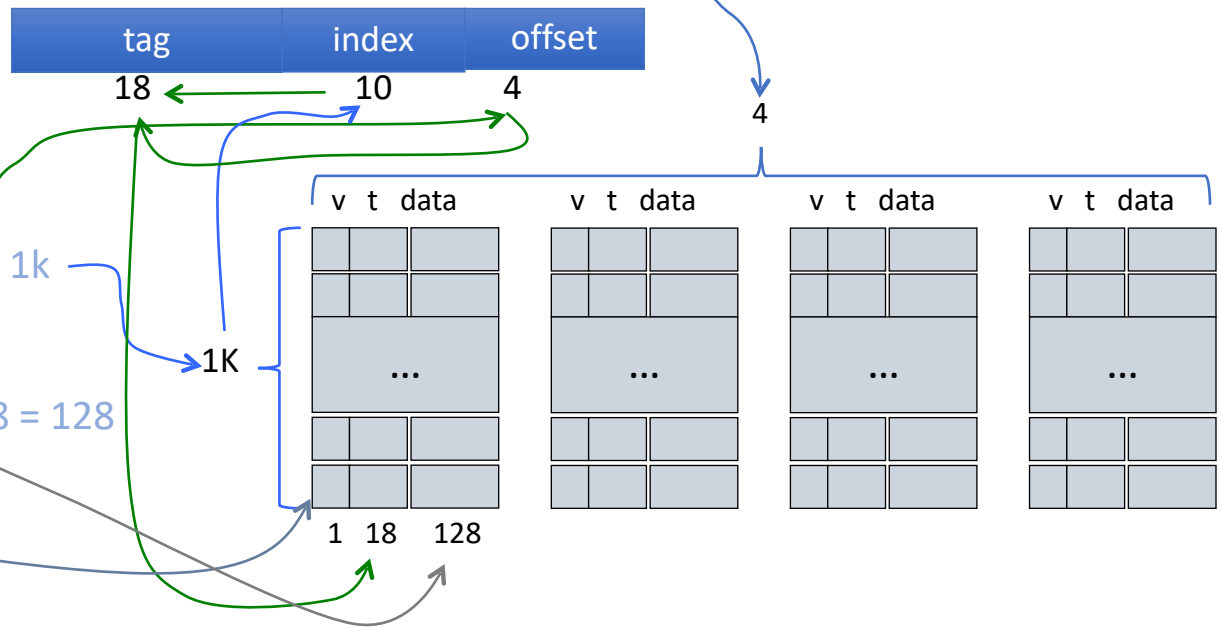
- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

64K/4/16 = 1k

16*8 = 128



Example: 4-way set associative cache

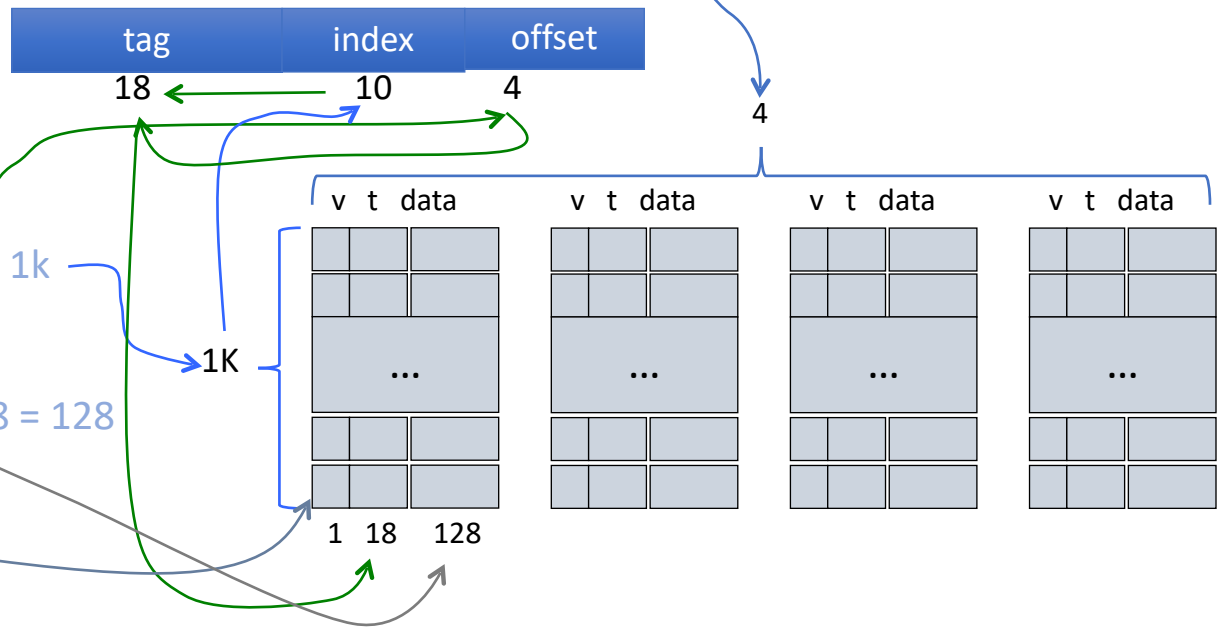
- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

64K/4/16 = 1k

16*8 = 128



(1+18+128) * 1K * 4 / 8 = 75,264 bytes

Example: 4-way set associative cache

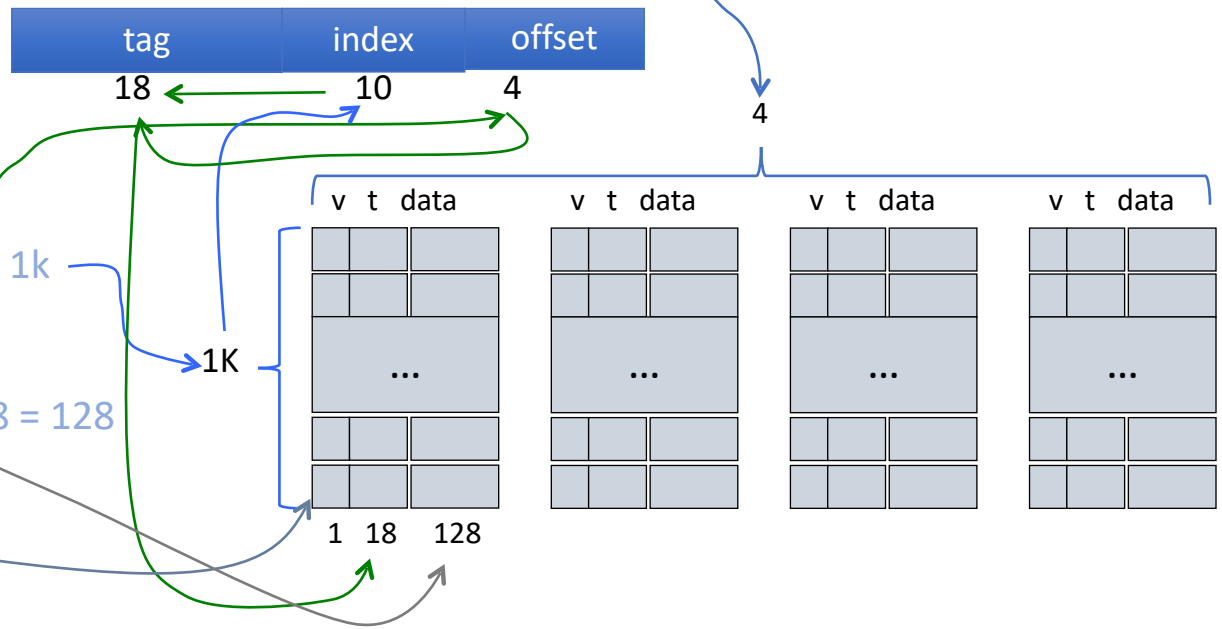
- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

64K/4/16 = 1k

16*8 = 128



(1+18+128) * 1K * 4 / 8 = 75,264 bytes

Hardware complexity?

Example: 4-way set associative cache

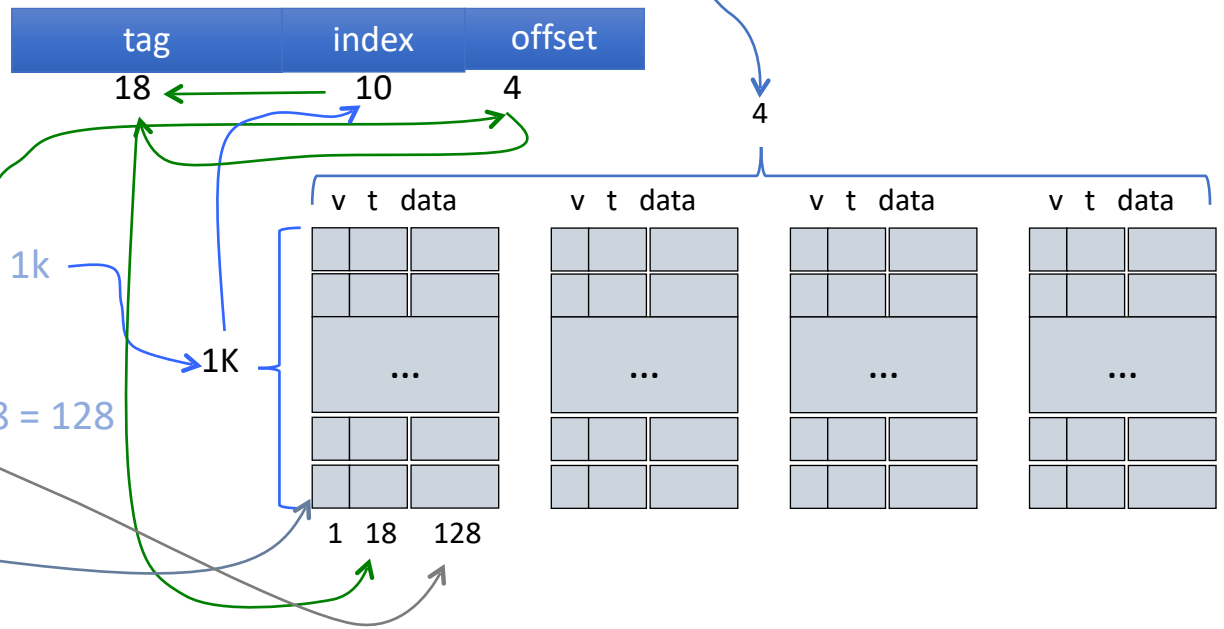
- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.

- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

64K/4/16 = 1k

16*8 = 128



(1+18+128) * 1K * 4 / 8 = 75,264 bytes

Hardware complexity?

4 18-bit comparators

Session ID: dag2200

In a fully associative cache …

…with 64K bytes of data, 64 bytes / block and a *t*-bit tag

19% A. There are four *t*-bit tag comparators

30% B. There are 64 *t*-bit tag comparators

43% C. There are 1k *t*-bit tag comparators

8% D. There is one *t*-bit tag comparator for the entire cache

Session ID: dag2200

In a fully associative cache …

…with 64K bytes of data, 64 bytes / block and a *t*-bit tag

19% A. There are four *t*-bit tag comparators

30% B. There are 64 *t*-bit tag comparators

43% C. There are 1k *t*-bit tag comparators

8% D. There is one *t*-bit tag comparator for the entire cache

FA caches have one comparator for each block/line/entry in the cache.
That means 64K/64 = 1K is the number of comparators.

Session ID: dag2200

In a 4-way set associative cache, …

…with 64K bytes of data, 64 bytes / block, with a *t*-bit tag

49% A. There are four *t*-bit tag comparators

29% B. There are 64 *t*-bit tag comparators

17% C. There are 1k *t*-bit tag comparators

4% D. There is one *t*-bit tag comparator for the entire cache

What about cache replacement policy?

	C0			C1		
	V	Tag	data	V	Tag	data
0						
1						
2						
3						
4						
5						
6						
7						

What about cache replacement policy?

	C0			C1		
	V	Tag	data	V	Tag	data
0						
1						
2						
3						
4						
5						
6						
7						

- What kind of cache is this?

What about cache replacement policy?

	C0			C1		
	V	Tag	data	V	Tag	data
0						
1						
2						
3						
4						
5						
6						
7						

- What kind of cache is this? 2-way set associative

What about cache replacement policy?

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

- What kind of cache is this? 2-way set associative

What about cache replacement policy?

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

- What kind of cache is this? 2-way set associative
- How many LRU bits do we need?

What about cache replacement policy?

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

- What kind of cache is this? 2-way set associative
- How many LRU bits do we need? Just one.

What about cache replacement policy?

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

- What kind of cache is this? 2-way set associative
- How many LRU bits do we need? Just one.
- What happens on every memory access?

What about cache replacement policy?

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

- What kind of cache is this? 2-way set associative
- How many LRU bits do we need? Just one.
- What happens on every memory access? Set LRU to 0/1 if we read from/store in C0/C1

What about cache replacement policy?

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

- What kind of cache is this? 2-way set associative
- How many LRU bits do we need? Just one.
- What happens on every memory access? Set LRU to 0/1 if we read from/store in C0/C1
- So what do we do with a 4-way set associative cache?

LRU replacement in a 4-way cache

	C0			C1			C2			C3			LRU
	V	Tag	data	V	Tag	data	V	Tag	data	V	Tag	data	
0													c1 -> c3 -> c0 -> c2
1													c0 -> c2 -> c1 -> c3
2													c2 -> c3 -> c0 -> c1
3													c3 -> c2 -> c1 -> c0

LRU replacement in a 4-way cache

	C0			C1			C2			C3			LRU
	V	Tag	data	V	Tag	data	V	Tag	data	V	Tag	data	
0													c1 -> c3 -> c0 -> c2
1													c0 -> c2 -> c1 -> c3
2													c2 -> c3 -> c0 -> c1
3													c3 -> c2 -> c1 -> c0

- Do we need a state machine for each cache line?

LRU replacement in a 4-way cache

	C0			C1			C2			C3			LRU
	V	Tag	data	V	Tag	data	V	Tag	data	V	Tag	data	
0													c1 -> c3 -> c0 -> c2
1													c0 -> c2 -> c1 -> c3
2													c2 -> c3 -> c0 -> c1
3													c3 -> c2 -> c1 -> c0

- Do we need a state machine for each cache line?
- Using as many state machines as the number of rows in the cache is a lot of hardware

LRU replacement in a 4-way cache

	C0			C1			C2			C3			LRU
	V	Tag	data	V	Tag	data	V	Tag	data	V	Tag	data	
0													c1 -> c3 -> c0 -> c2
1													c0 -> c2 -> c1 -> c3
2													c2 -> c3 -> c0 -> c1
3													c3 -> c2 -> c1 -> c0

- Do we need a state machine for each cache line?
- Using as many state machines as the number of rows in the cache is a lot of hardware
- Each state machine ➔ 4! States ➔ 5 bit state register

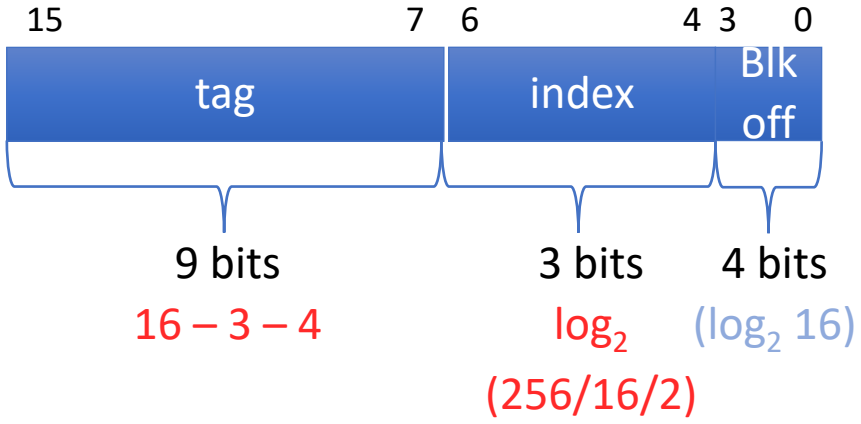
Example

- 16-bit byte-addressable memory
- 2-way 256-byte cache
- 16-byte cache blocks

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

Example

- 16-bit byte-addressable memory
- 2-way 256-byte cache
- 16-byte cache blocks



	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2							
3							
4							
5							
6							
7							

0xF123 → 1111 0001 0010 0011

Offset: 0011

Index: 010

Tag: 1111 0001 0 → 0x1E2

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx				1
3							
4							
5							
6							
7							

0xF123 → 1111 0001 0010 0011

Offset: 0011

Index: 010

Tag: 1111 0001 0 → 0x1E2

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx				1
3							
4							
5							
6							
7							

0x0252 → 0000 0010 0101 0010

Offset: 0010

Index: 101

Tag: 0000 0010 0 → 0x004

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx				1
3							
4							
5				1	0x004	zzz	0
6							
7							

0x0252 → 0000 0010 0101 0010

Offset: 0010

Index: 101

Tag: 0000 0010 0 → 0x004

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx				1
3							
4							
5				1	0x004	zzz	0
6							
7							

0x11A0 → 0001 0001 1010 0000

Offset: 0000

Index: 010

Tag: 0001 0001 1 → 0x023

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x023	xxx	0
3							
4							
5				1	0x004	zzz	0
6							
7							

0x11A0 → 0001 0001 1010 0000

Offset: 0000

Index: 010

Tag: 0001 0001 1 → 0x023

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x023	xxx	0
3							
4							
5				1	0x004	zzz	0
6							
7							

0xF120 → 1111 0001 0010 0000

Offset: 0000

Index: 010

Tag: 1111 0001 0 → 0x1E2

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x023	xxx	0
3							
4							
5				1	0x004	zzz	0
6							
7							

0xF120 → 1111 0001 0010 0000

Offset: 0000

Index: 010

Tag: 1111 0001 0 → 0x1E2

Cache Hit!

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x023	xxx	1
3							
4							
5				1	0x004	zzz	0
6							
7							

0xF120 → 1111 0001 0010 0000

Offset: 0000

Index: 010

Tag: 1111 0001 0 → 0x1E2

Cache Hit!

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x023	xxx	1
3							
4							
5				1	0x004	zzz	0
6							
7							

0xB020 → 1011 0000 0010 0000

Offset: 0000

Index: 010

Tag: 1011 0000 0 → 0x160

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x023	xxx	1
3							
4							
5				1	0x004	zzz	0
6							
7							

Replace way 1's block!

0xB020 → 1011 0000 0010 0000

Offset: 0000

Index: 010

Tag: 1011 0000 0 → 0x160

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x160	yyy	1
3							
4							
5				1	0x004	zzz	0
6							
7							

Replace way 1's block!

0xB020 → 1011 0000 0010 0000

Offset: 0000

Index: 010

Tag: 1011 0000 0 → 0x160

Example

Access stream from CPU:

0xF123

0x0252

0x11A0

0xF120

0xB020

	C0			C1			
	V	Tag	data	V	Tag	data	LRU
0							
1							
2	1	0x1E2	xxx	1	0x160	yyy	0
3							
4							
5				1	0x004	zzz	0
6							
7							

Replace way 1's block!

0xB020 → 1011 0000 0010 0000

Offset: 0000

Index: 010

Tag: 1011 0000 0 → 0x160