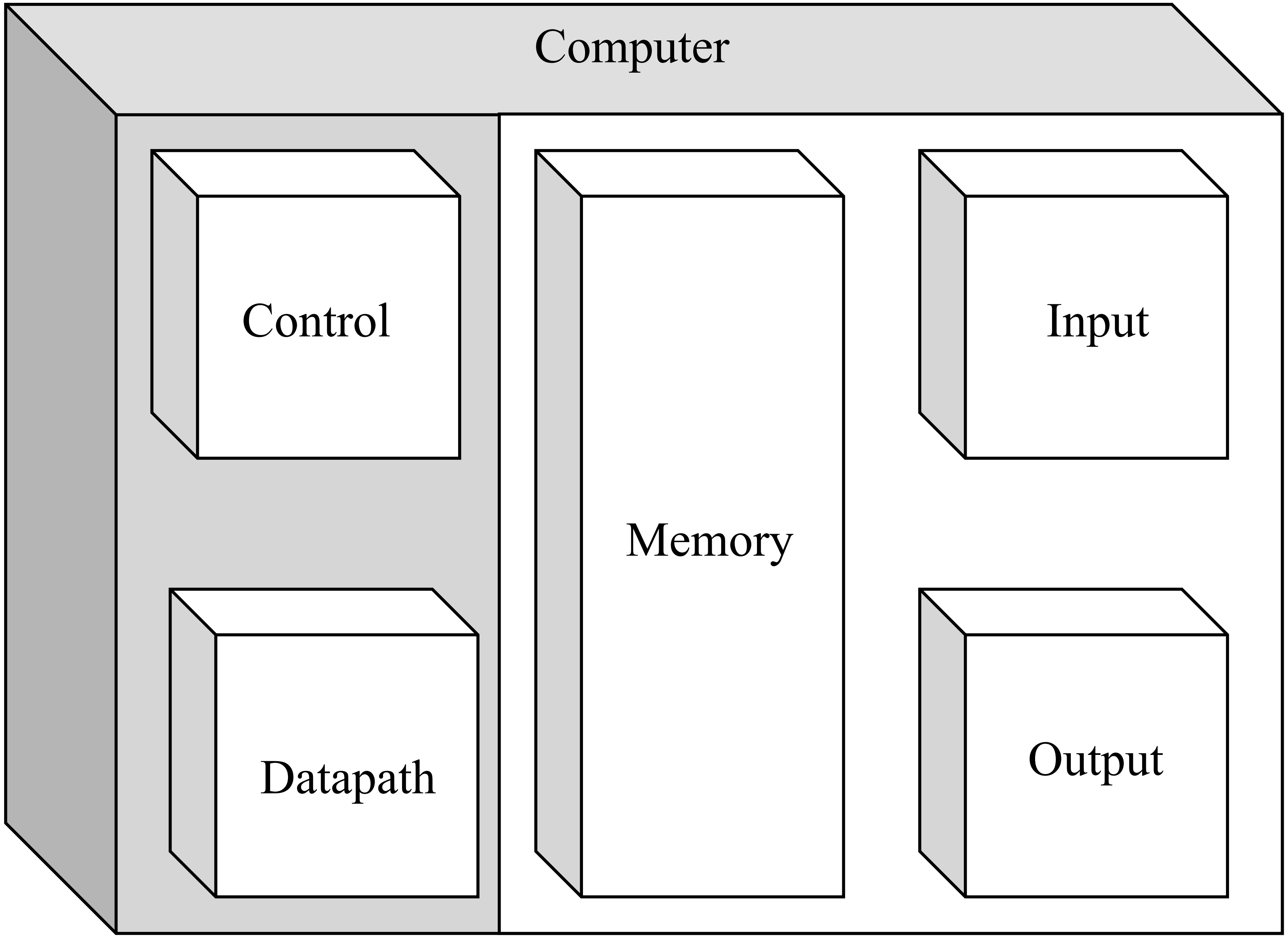
Memory Subsystem Design

Jason Mars

The Memory Subsystem



Memory Locality

Memory Locality

- Memory hierarchies take advantage of memory locality.

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.
- Memories take advantage of two types of locality
  - -- near in time => we will often access the same data again very soon
  - -- near in space/distance => our next access is often very close to our last access (or recent accesses).

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.
- Memories take advantage of two types of locality
  - **Temporal Locality** -- near in time => we will often access the same data again very soon
  - -- near in space/distance => our next access is often very close to our last access (or recent accesses).

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.
- Memories take advantage of two types of locality
  - **Temporal Locality** -- near in time => we will often access the same data again very soon
  - **Spacial Locality** -- near in space/distance => our next access is often very close to our last access (or recent accesses).

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.
- Memories take advantage of two types of locality
    - **Temporal Locality** -- near in time => we will often access the same data again very soon
    - **Spacial Locality** -- near in space/distance => our next access is often very close to our last access (or recent accesses).

(this sequence of addresses exhibits both temporal and spatial locality)
1,2,3,1,2,3,8,8,47,9,10,8,8...

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.
- Memories take advantage of two types of locality
  - **Temporal Locality** -- near in time => we will often access the same data again very soon
  - **Spacial Locality** -- near in space/distance => our next access is often very close to our last access (or recent accesses).

(this sequence of addresses exhibits both temporal and spatial locality)
1,2,3,1,2,3,8,8,47,9,10,8,8...

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.
- Memories take advantage of two types of locality
    - **Temporal Locality** -- near in time => we will often access the same data again very soon
    - **Spacial Locality** -- near in space/distance => our next access is often very close to our last access (or recent accesses).

(this sequence of addresses exhibits both temporal and spatial locality)
1,2,3,1,2,3,8,8,47,9,10,8,8...

Memory Locality

- Memory hierarchies take advantage of memory locality.
- Memory locality is the principle that future memory accesses are near past accesses.
- Memories take advantage of two types of locality
  - **Temporal Locality** -- near in time => we will often access the same data again very soon
  - **Spacial Locality** -- near in space/distance => our next access is often very close to our last access (or recent accesses).

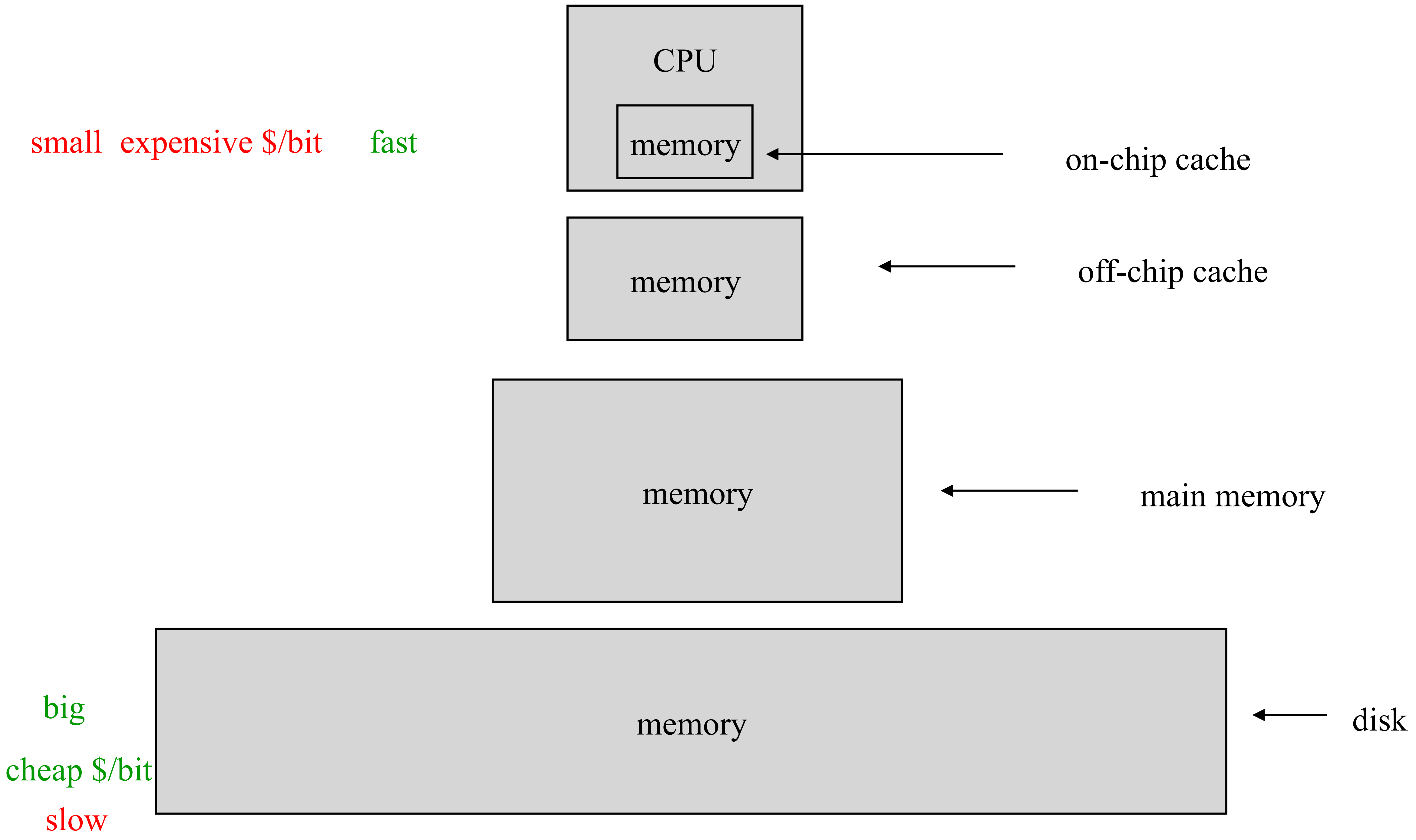
(this sequence of addresses exhibits both temporal and spatial locality)
1,2,3,1,2,3,8,8,47,9,10,8,8...

Locality and Cacheing

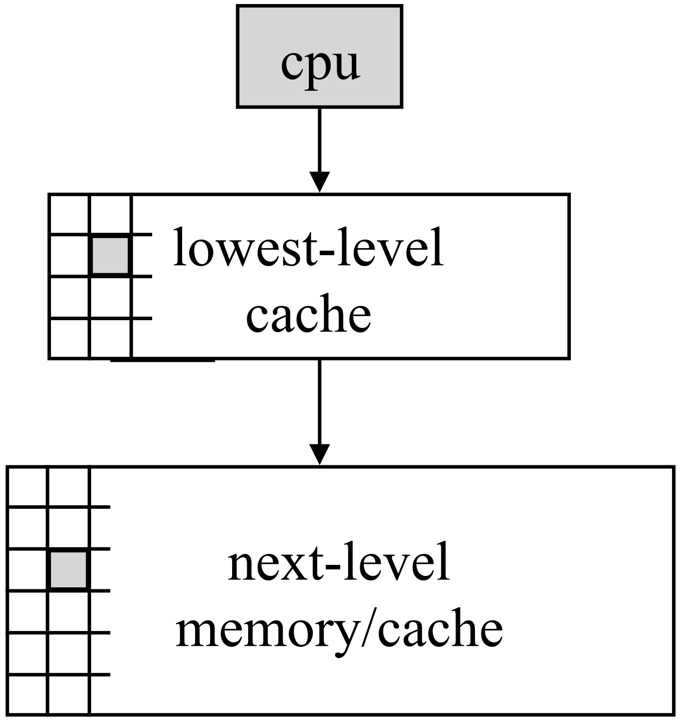
- Memory hierarchies exploit locality by cacheing (keeping close to the processor) data likely to be used again.
- This is done because we can build large, slow memories and small, fast memories, but we can't build large, fast memories.
- If it works, we get the illusion of SRAM access time with disk capacity

SRAM access times are ~1ns at cost of $2000 to $5000 per Gbyte.
DRAM access times are ~70ns at cost of $20 to $75 per Gbyte.
Disk access times are 5 to 20 million ns at cost of $.20 to $2 per Gbyte.

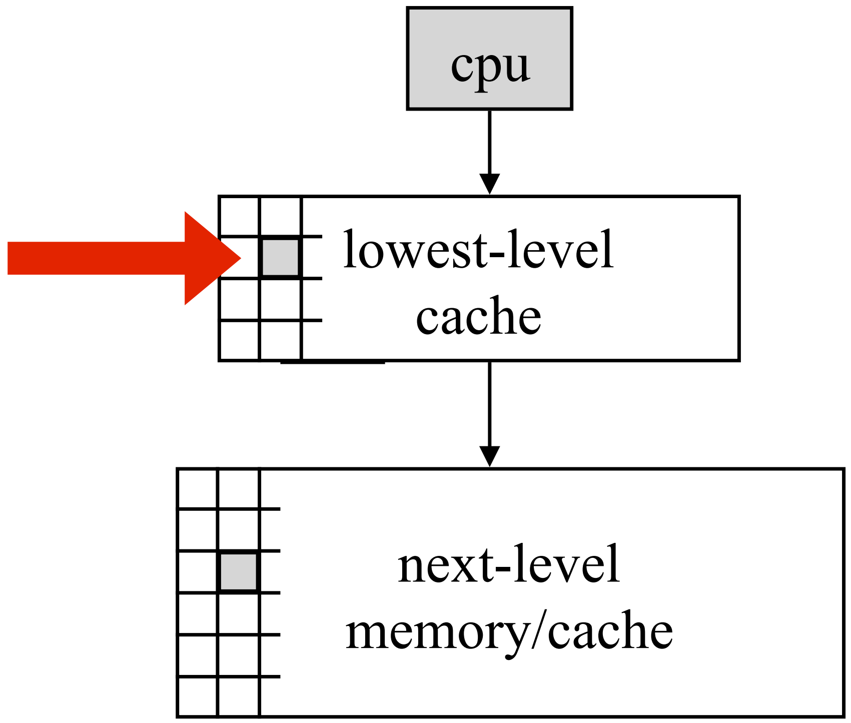
Typical Memory Hierarchy



Cache Fundamentals

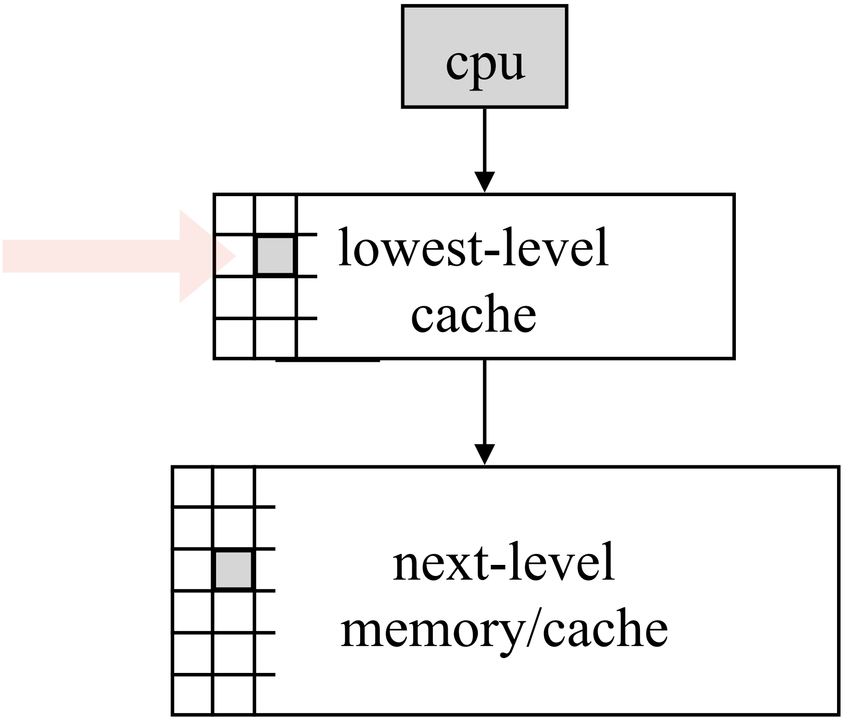


Cache Fundamentals



cache hit -- an access where the data is found in the cache.

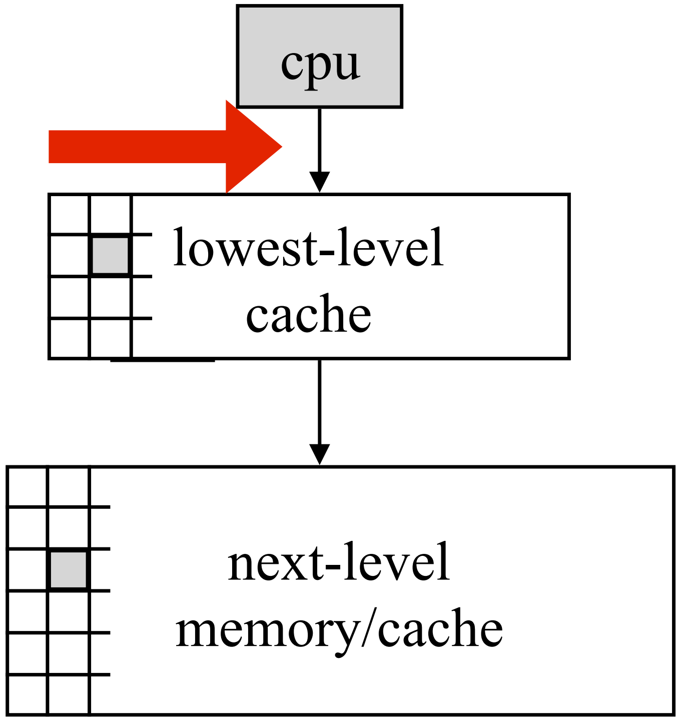
Cache Fundamentals



cache hit -- an access where the data is found in the cache.

cache miss -- an access which isn't

Cache Fundamentals

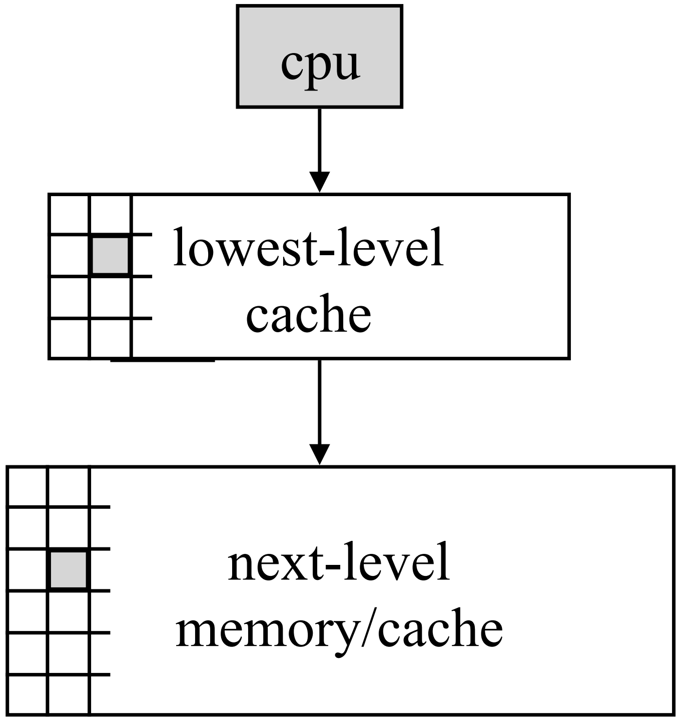


cache hit -- an access where the data is found in the cache.

cache miss -- an access which isn’t

hit time -- time to access the cache

Cache Fundamentals



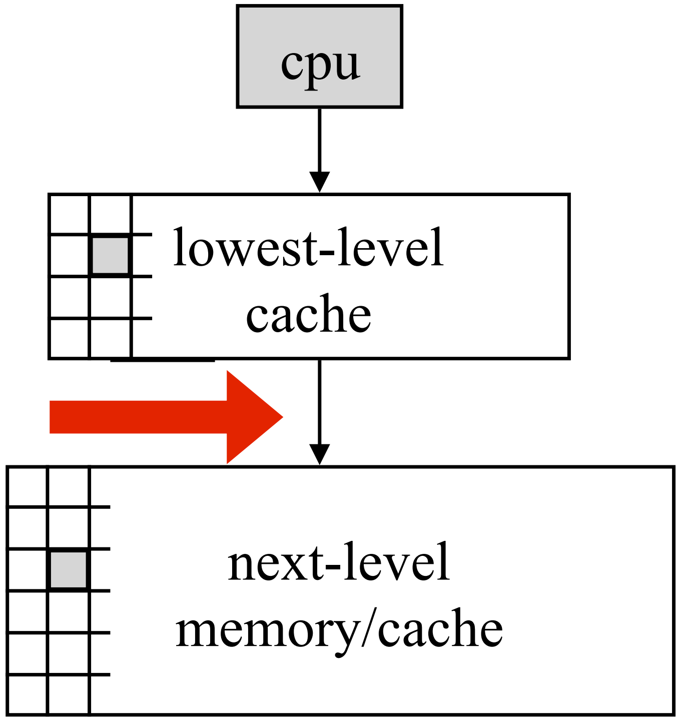
cache hit -- an access where the data is found in the cache.

cache miss -- an access which isn’t

hit time -- time to access the cache

hit ratio -- percentage of time the data is found in the cache

Cache Fundamentals



cache hit -- an access where the data is found in the cache.

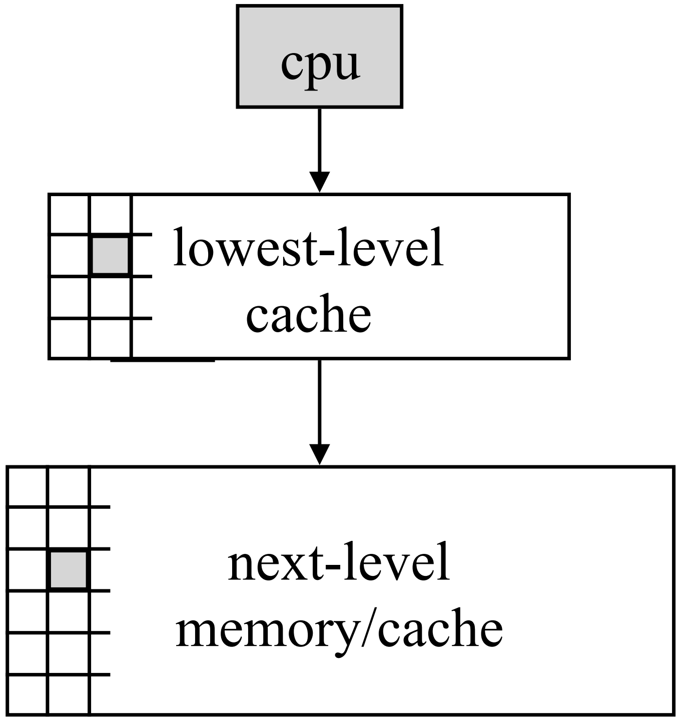
cache miss -- an access which isn't

hit time -- time to access the cache

hit ratio -- percentage of time the data is found in the cache

miss penalty -- time to move data from further level to closer, then to cpu

Cache Fundamentals



cache hit -- an access where the data is found in the cache.

cache miss -- an access which isn't

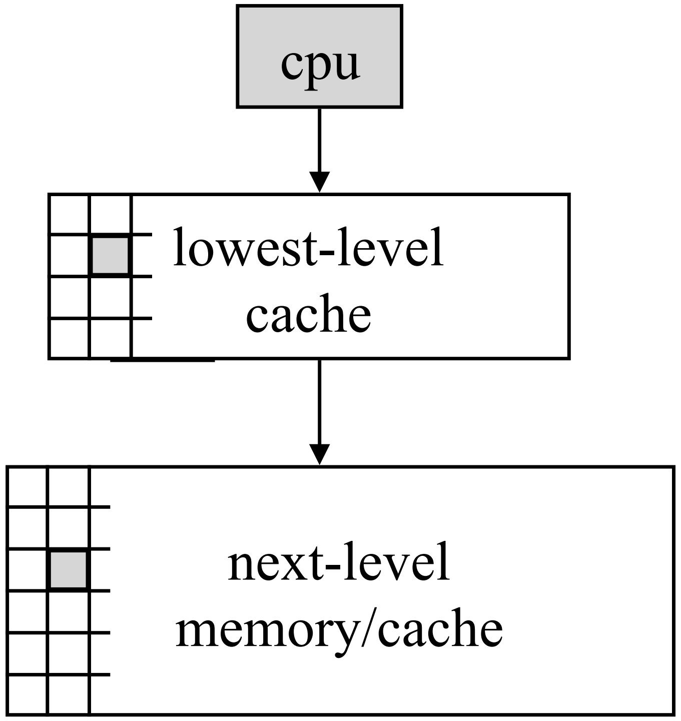
hit time -- time to access the cache

hit ratio -- percentage of time the data is found in the cache

miss penalty -- time to move data from further level to closer, then to cpu

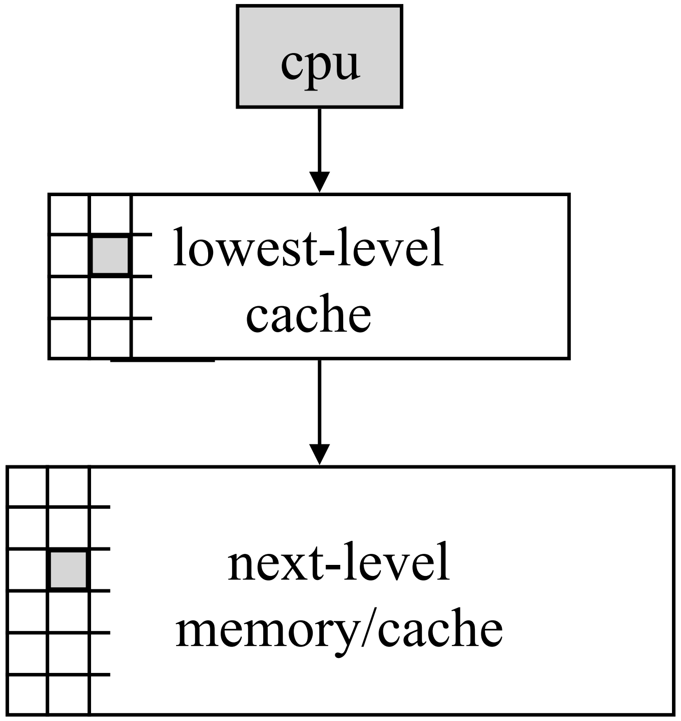
miss ratio -- (1 - hit ratio)

More Fundamentals



More Fundamentals

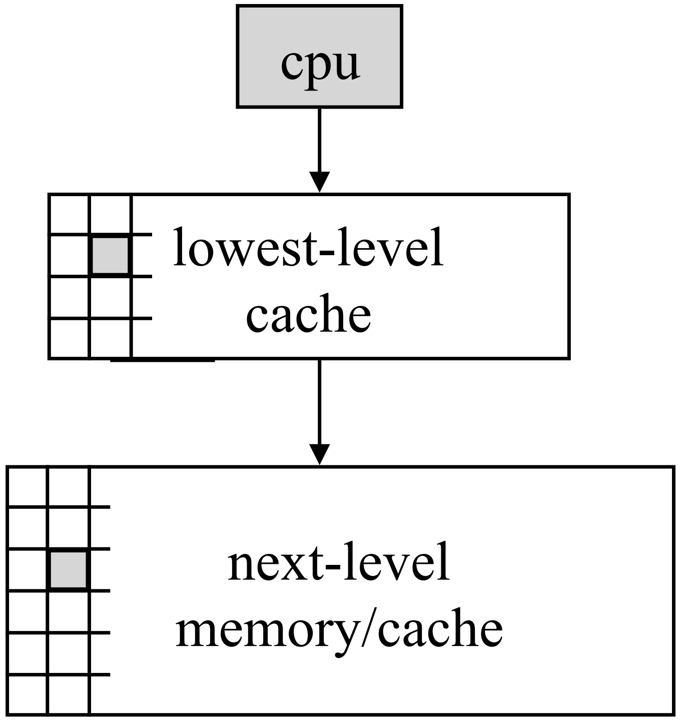
- **cache block size** or **cache line size** -- the amount of data that gets transferred on a cache miss.



More Fundamentals

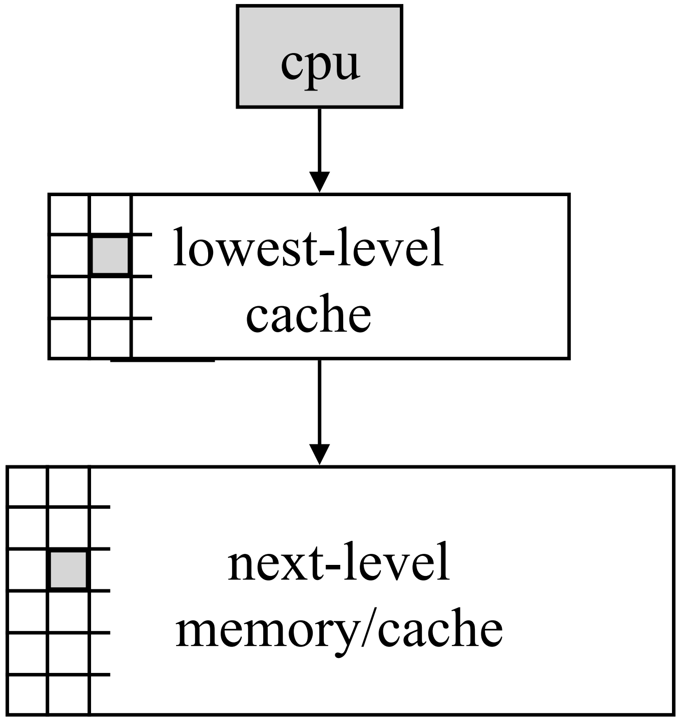
- **cache block size** or **cache line size** -- the amount of data that gets transferred on a cache miss.

- **instruction cache** -- cache that only holds instructions.



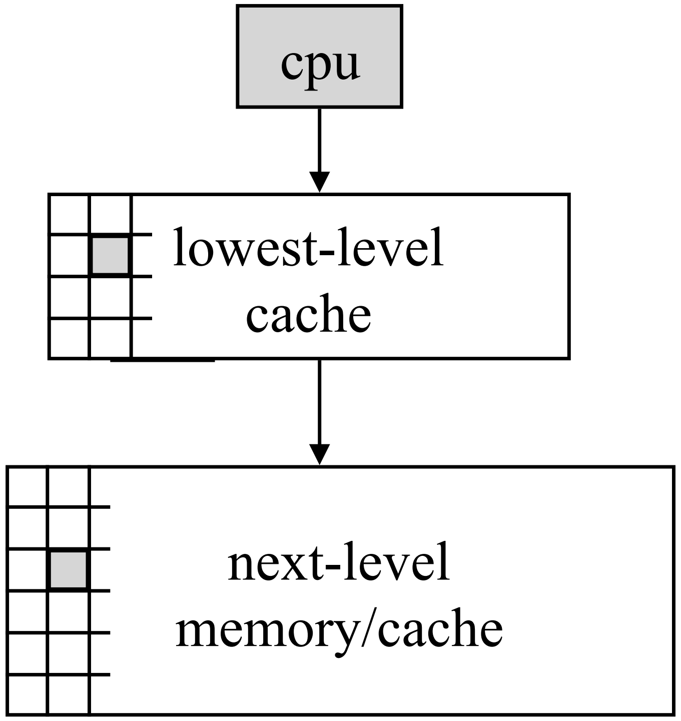
More Fundamentals

- **cache block size** or **cache line size** -- the amount of data that gets transferred on a cache miss.
- **instruction cache** -- cache that only holds instructions.
- **data cache** -- cache that only caches data.



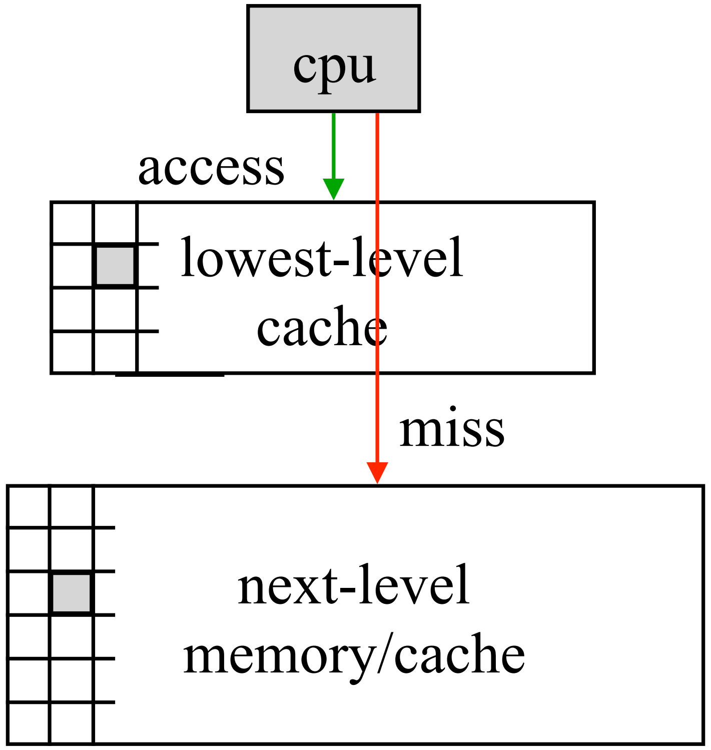
More Fundamentals

- **cache block size** or **cache line size** -- the amount of data that gets transferred on a cache miss.
- **instruction cache** -- cache that only holds instructions.
- **data cache** -- cache that only caches data.
- **unified cache** -- cache that holds both.



Caching Issues

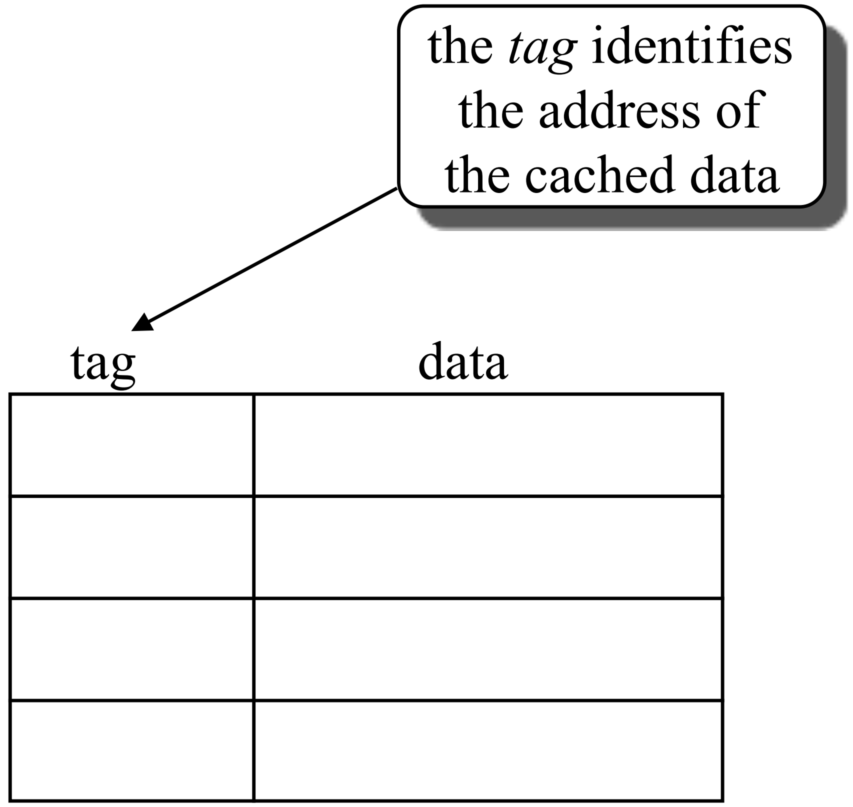
- On a memory access -
  - How do I know if this is a hit or miss?
- On a cache miss -
  - where to put the new data?
  - what data to throw out?
  - how to remember what data this is?



A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



the *tag* identifies the address of the cached data

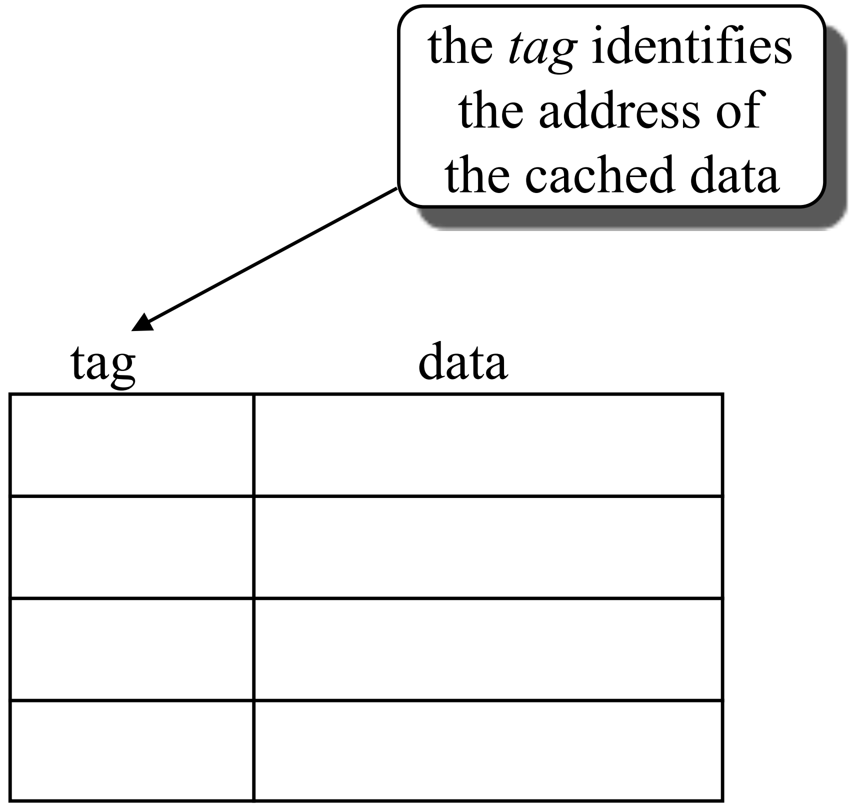
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called _______________

- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



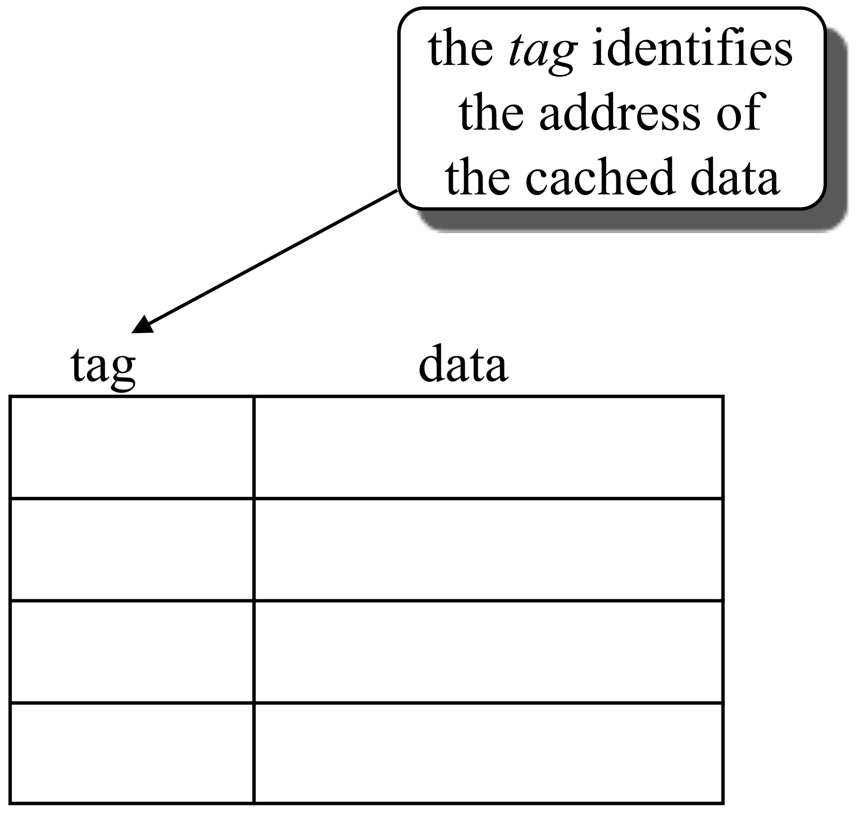
the *tag* identifies the address of the cached data

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

the *tag* identifies the address of the cached data

tag	data
000001	

4 entries, each block holds one word, any block can hold any word.

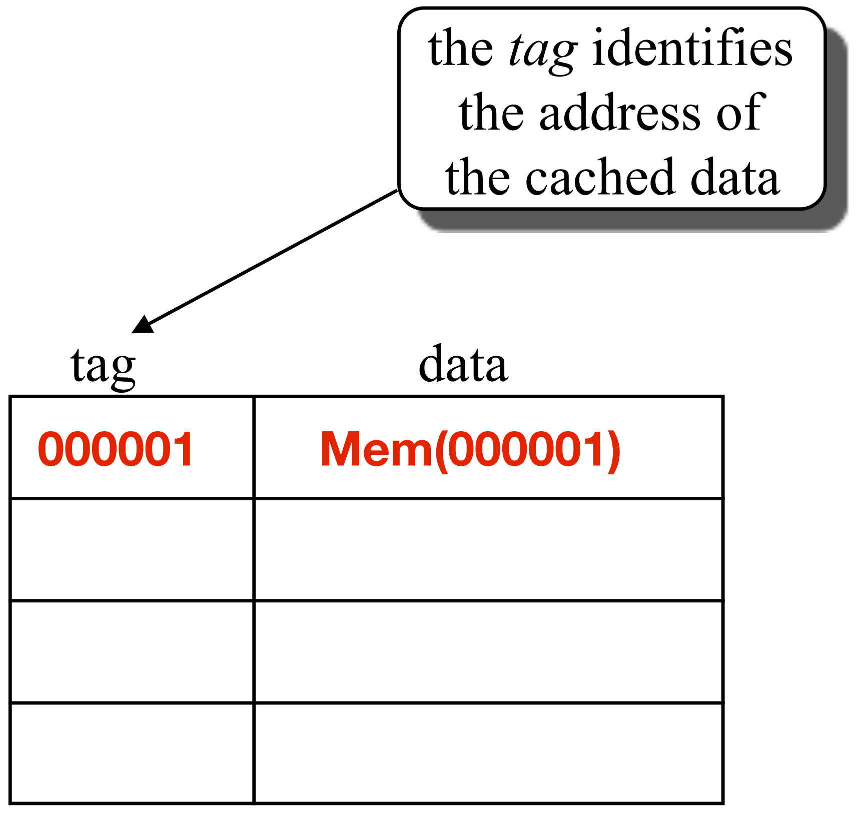
- A cache that can put a line of data anywhere is called **Fully Associative**

- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

the *tag* identifies the address of the cached data

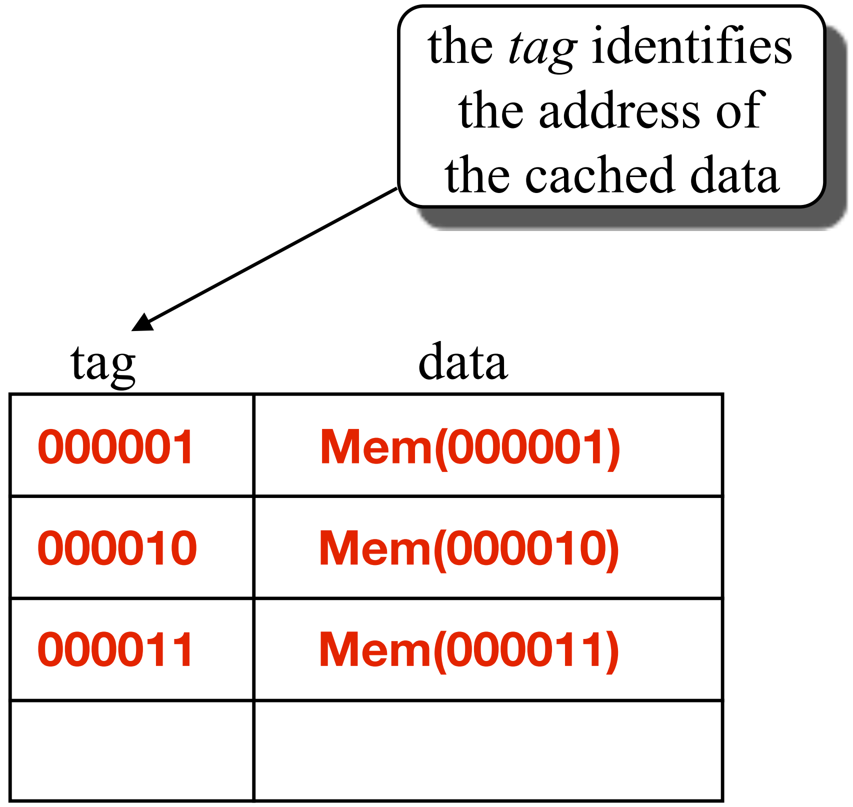
tag	data
000001	**Mem(000001)**
000010	**Mem(000010)**

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

the *tag* identifies the address of the cached data

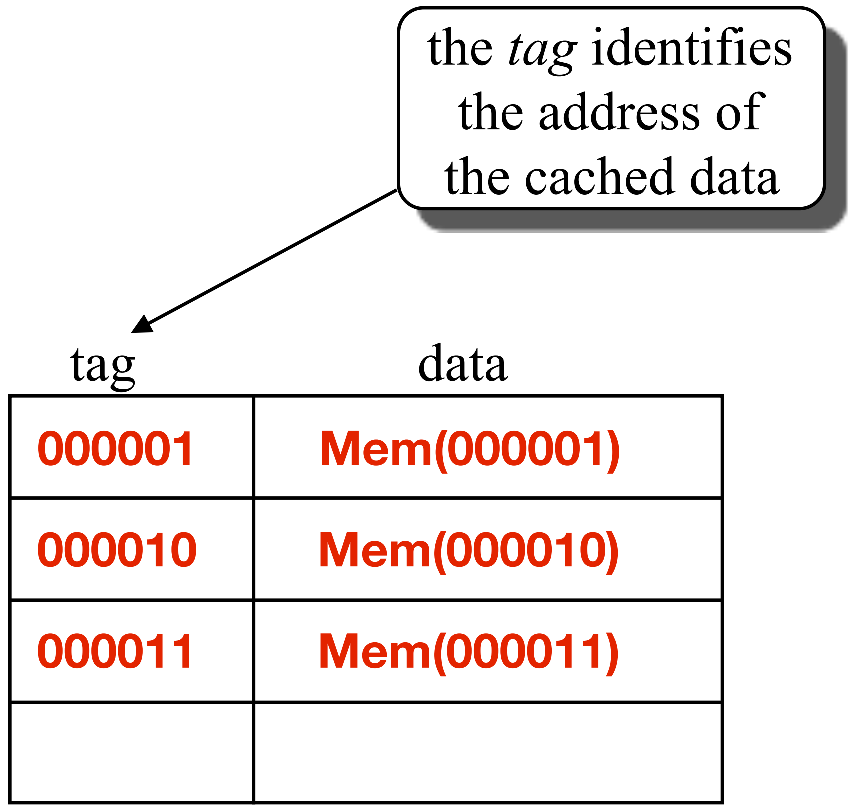
tag	data
000001	**Mem(000001)**
000010	**Mem(000010)**
000011	**Mem(000011)**

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)

4 entries, each block holds one word, any block can hold any word.

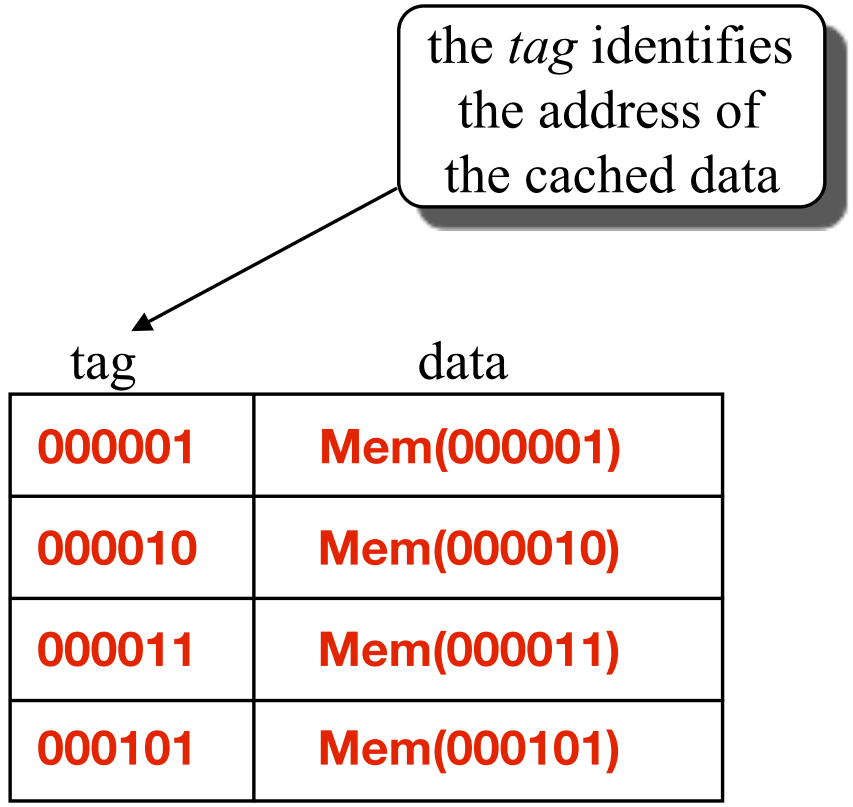
- A cache that can put a line of data anywhere is called **Fully Associative**

- The most popular replacement strategy is LRU (   ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)
000101	Mem(000101)

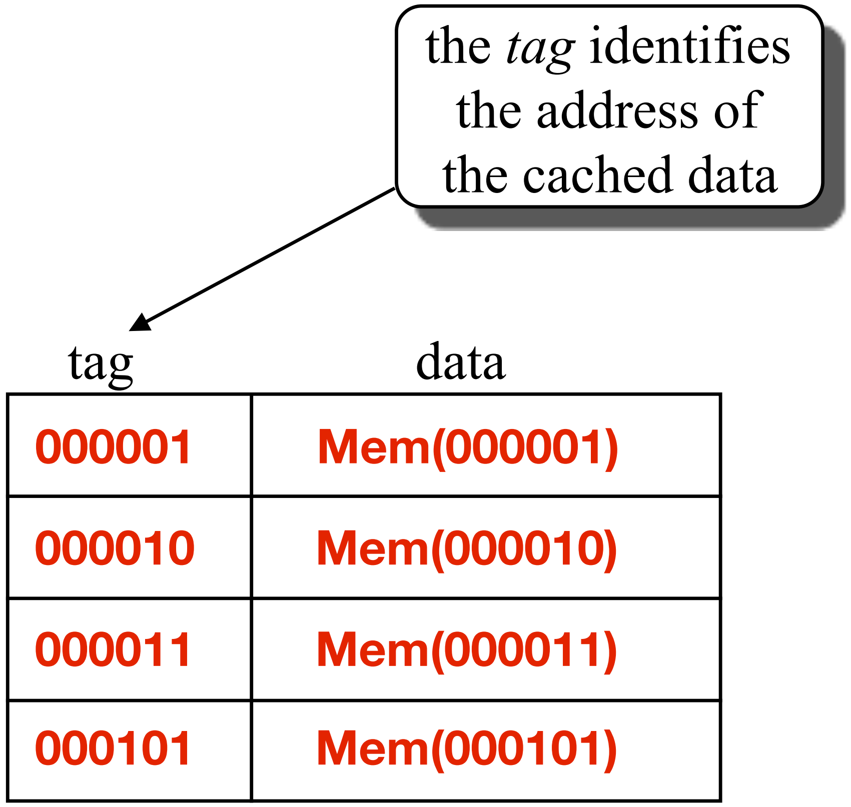
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)
000101	Mem(000101)

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)
000101	Mem(000101)

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**

- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

the *tag* identifies the address of the cached data

tag	data
000001	**Mem(000001)**
000010	**Mem(000010)**
000011	**Mem(000011)**
000101	**Mem(000101)**

4 entries, each block holds one word, any block can hold any word.

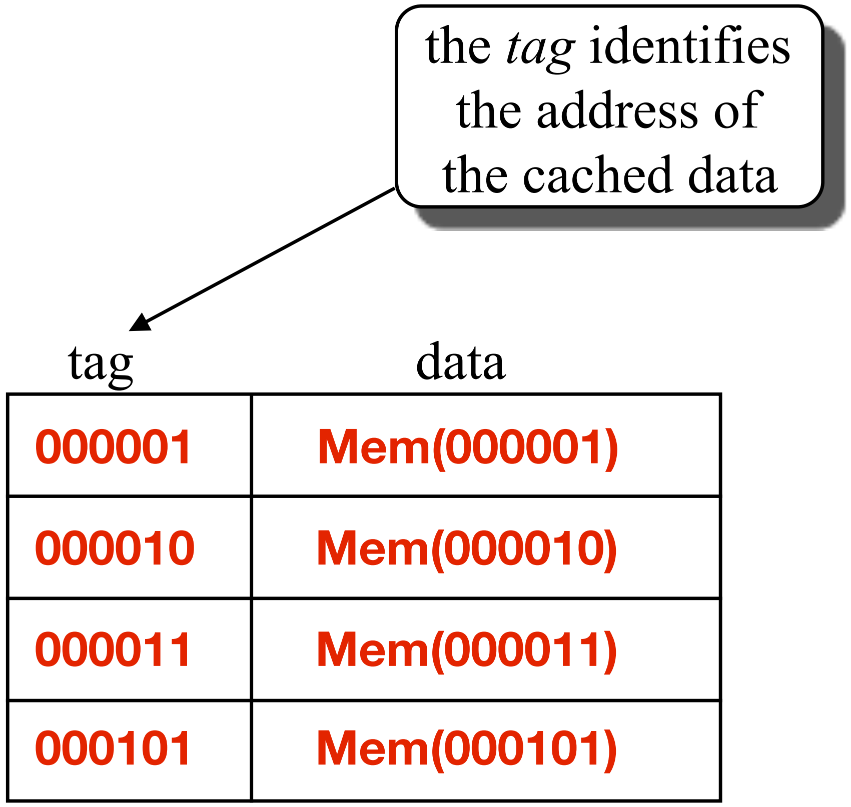
- A cache that can put a line of data anywhere is called **Fully Associative**

- The most popular replacement strategy is LRU (**Least Recently Used**).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)
000101	Mem(000101)

4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU (**Least Recently Used**).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

the *tag* identifies the address of the cached data

tag	data
000001	Mem(000001)
000010	Mem(000010)
000011	Mem(000011)
000101	Mem(000101)

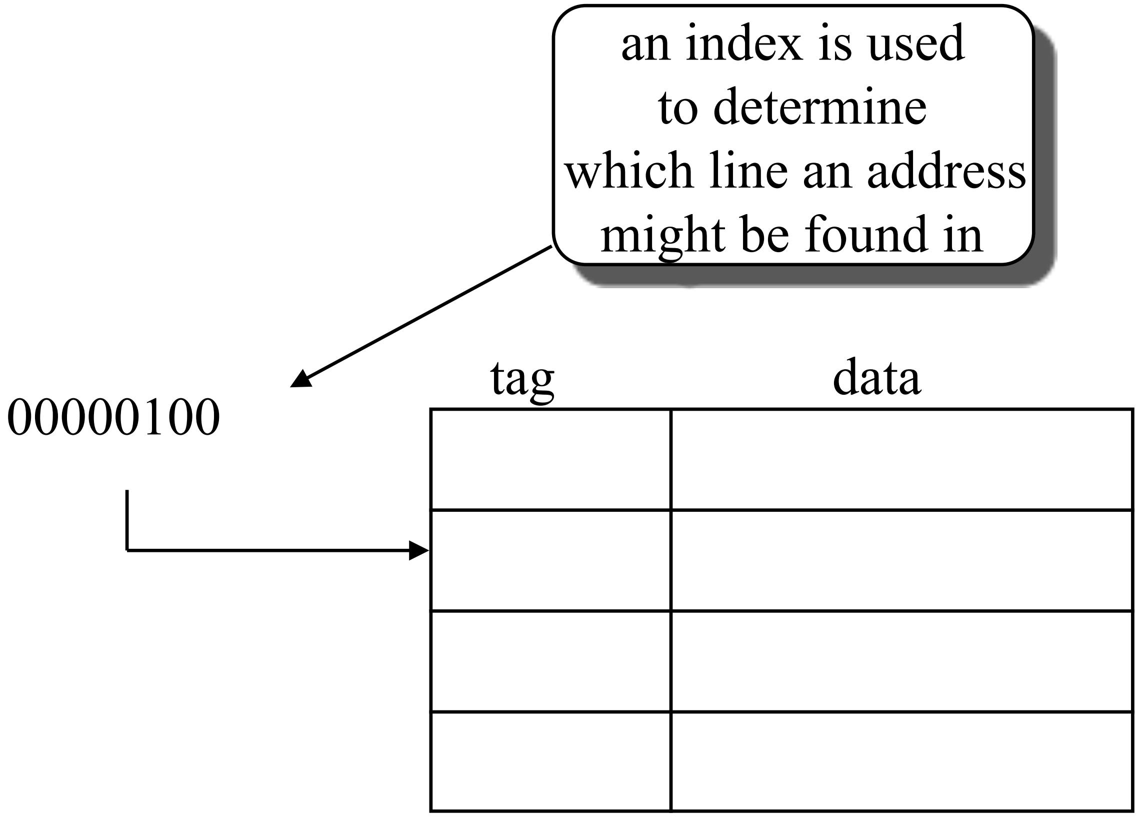
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU (**Least Recently Used**).

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

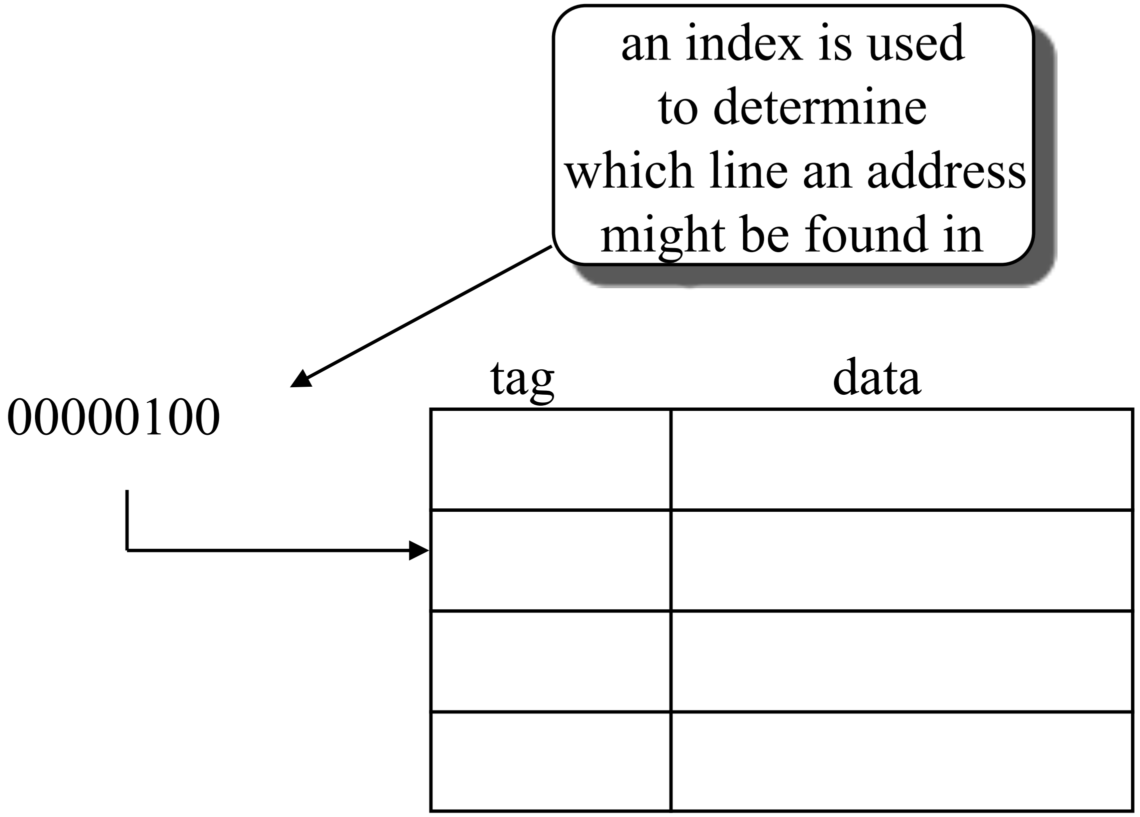
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called _________________.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100


an index is used to determine which line an address might be found in

00000100

tag

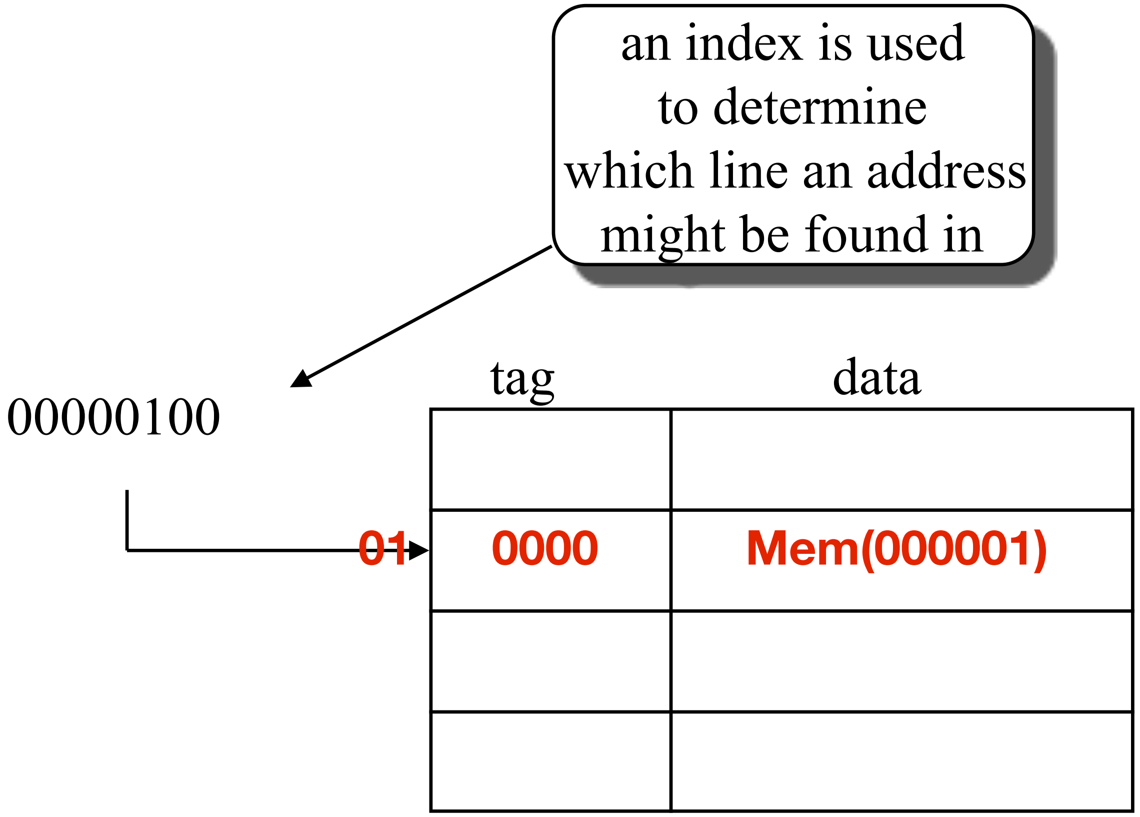
data

4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

	tag	data
01	0000	Mem(000001)

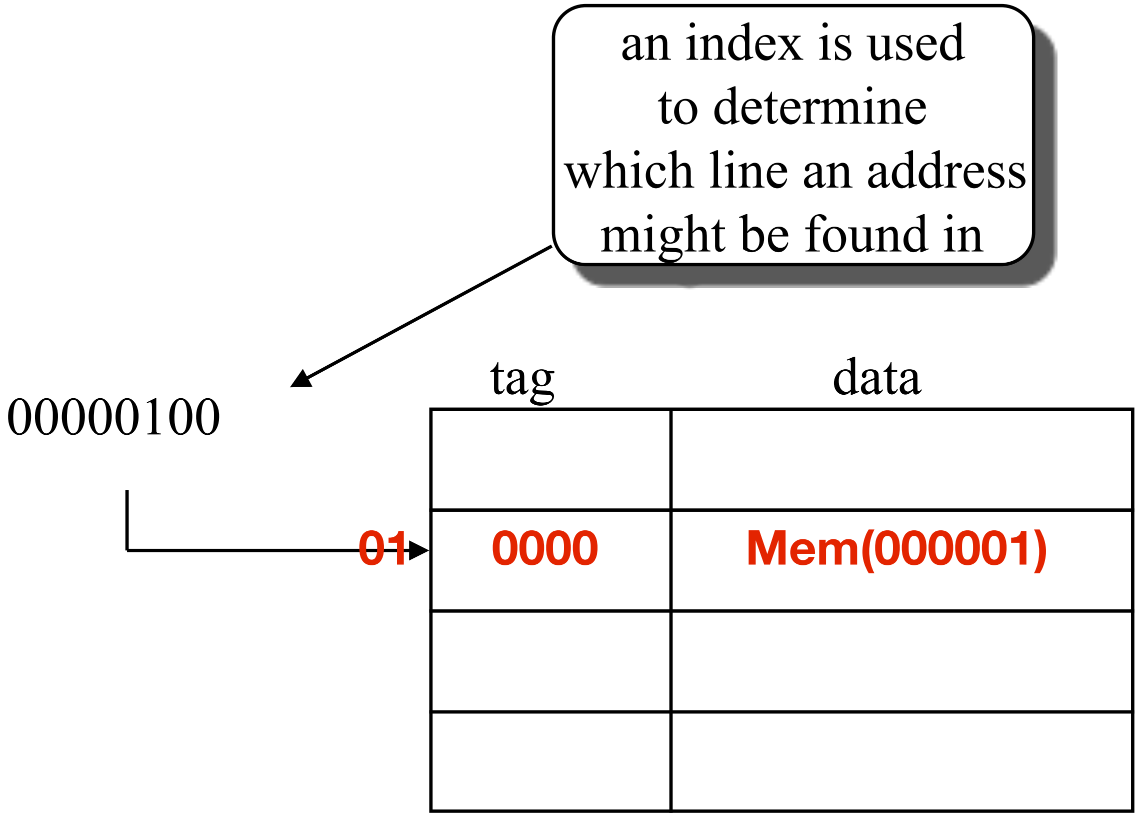
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

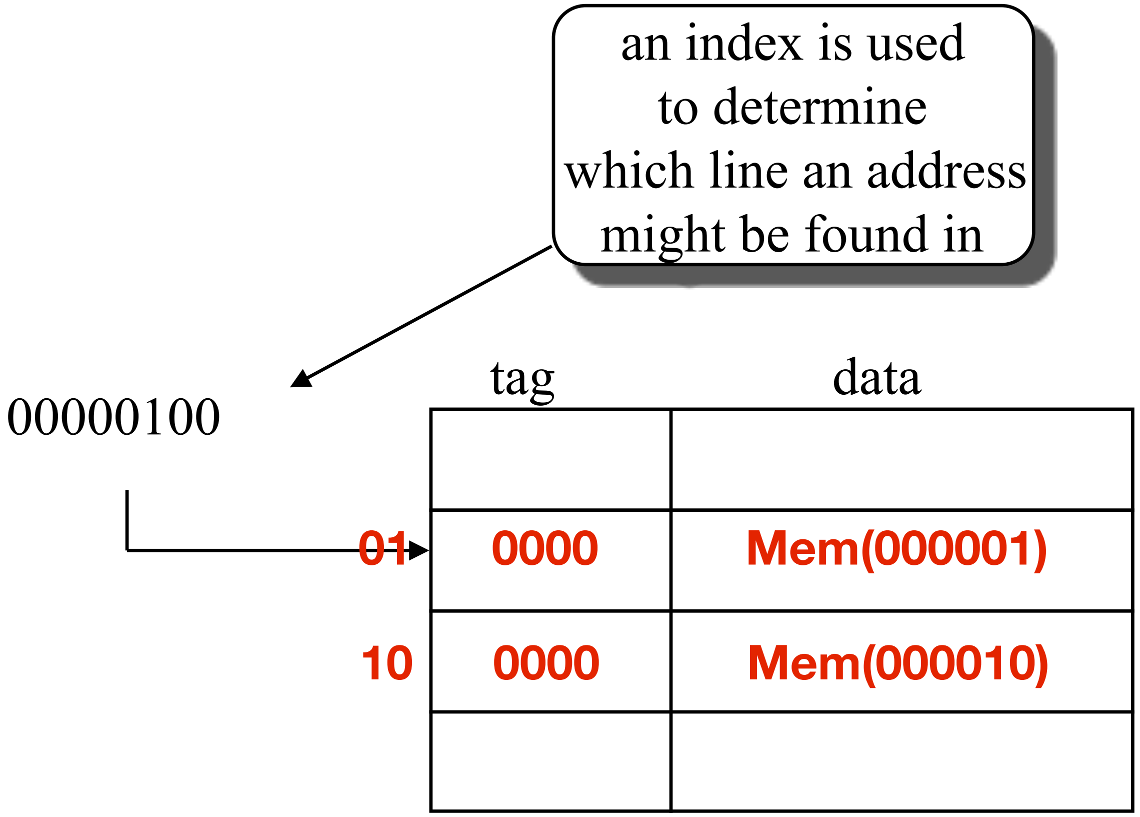
	tag	data
01	0000	Mem(000001)

4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

	tag	data
01	0000	Mem(000001)
10	0000	Mem(000010)

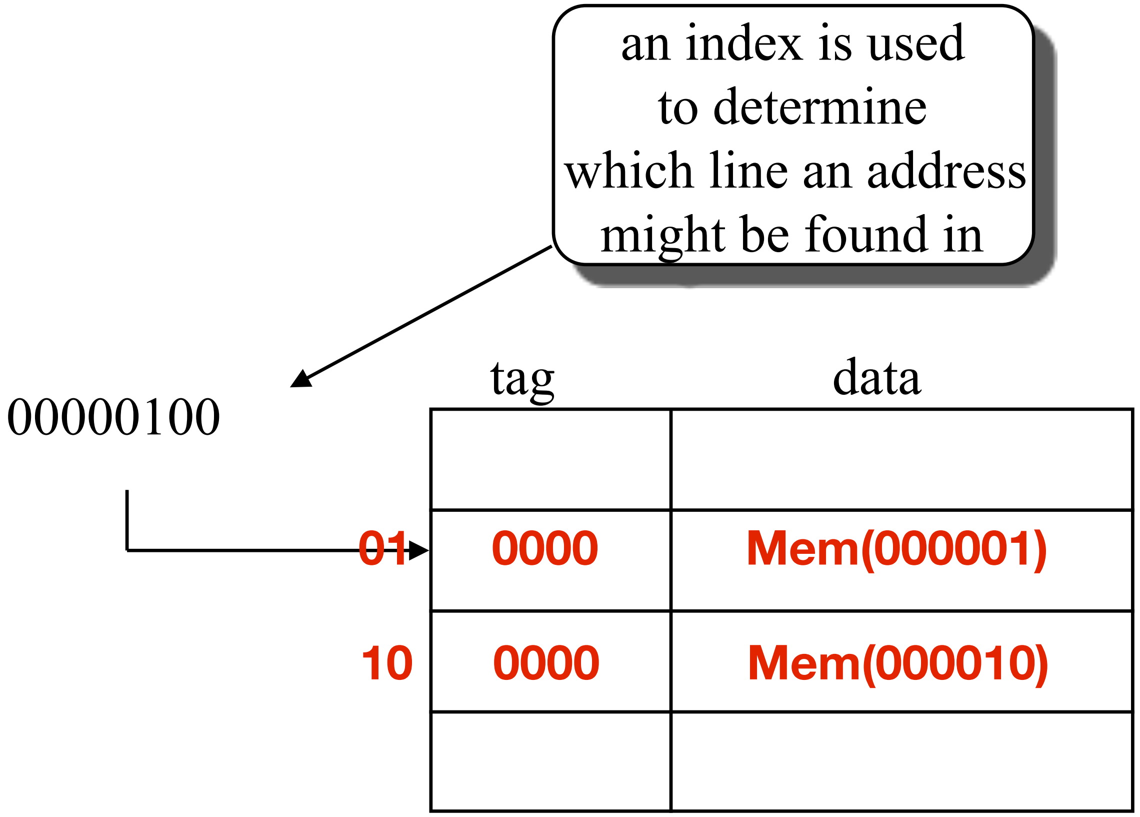
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

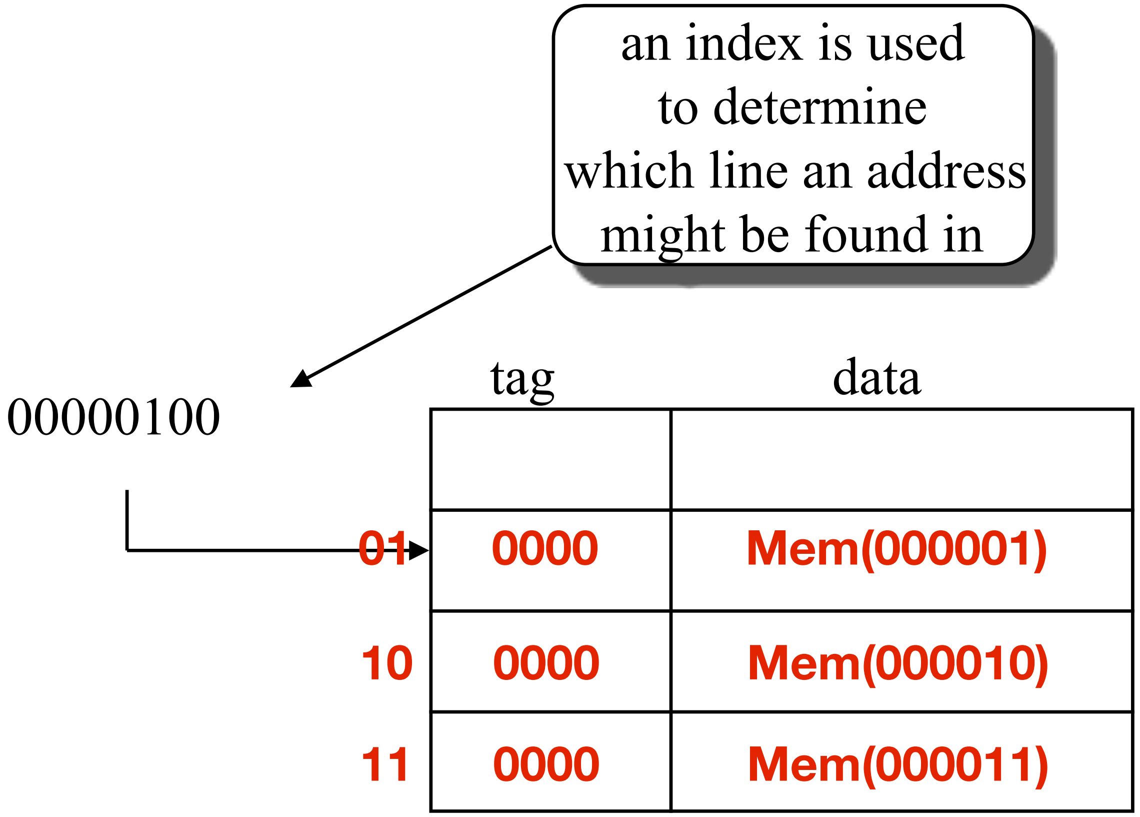
	tag	data
01	0000	Mem(000001)
10	0000	Mem(000010)

4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

	tag	data
01	0000	Mem(000001)
10	0000	Mem(000010)
11	0000	Mem(000011)

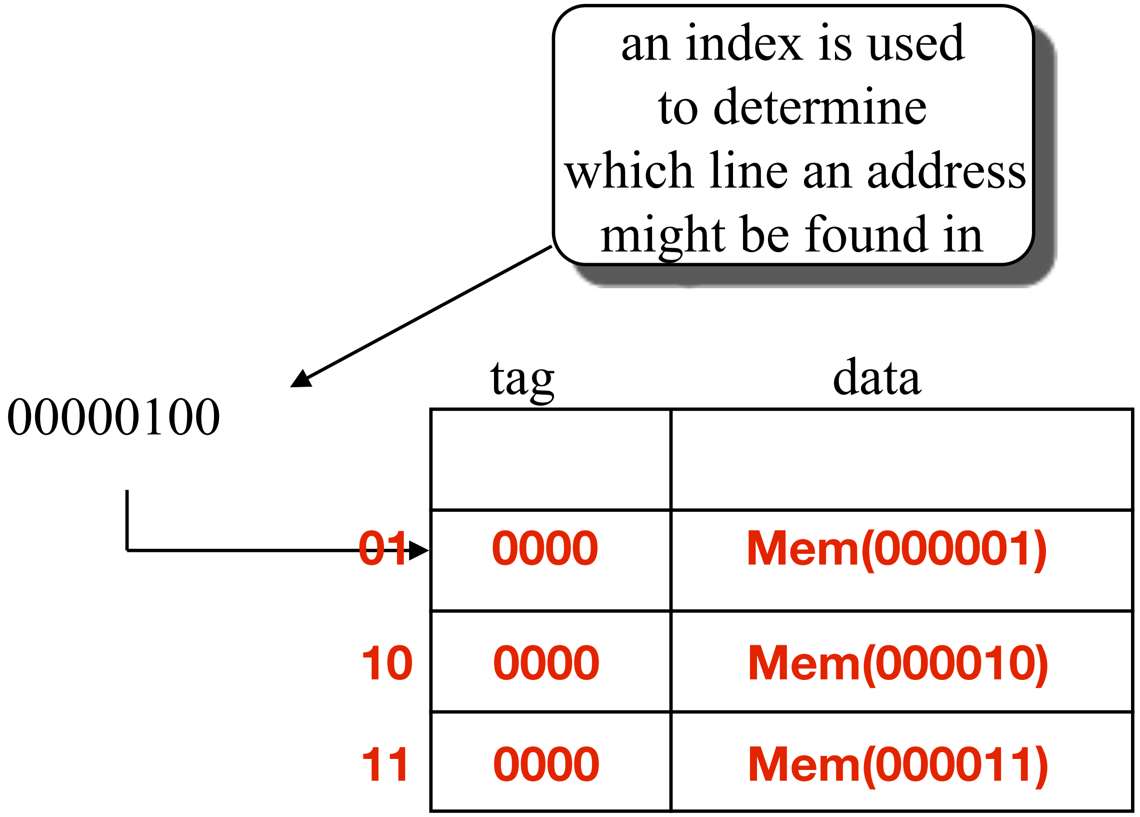
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

	tag	data
01	0000	Mem(000001)
10	0000	Mem(000010)
11	0000	Mem(000011)

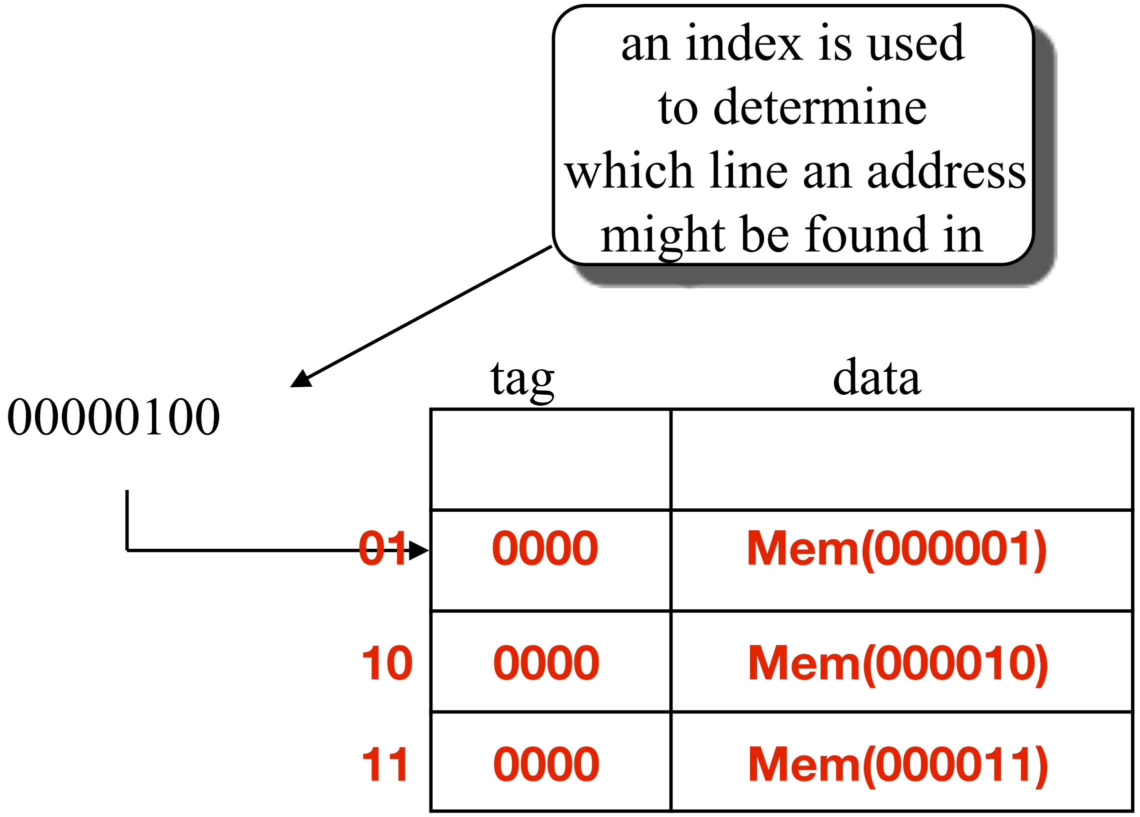
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

	tag	data
01	0000	Mem(000001)
10	0000	Mem(000010)
11	0000	Mem(000011)

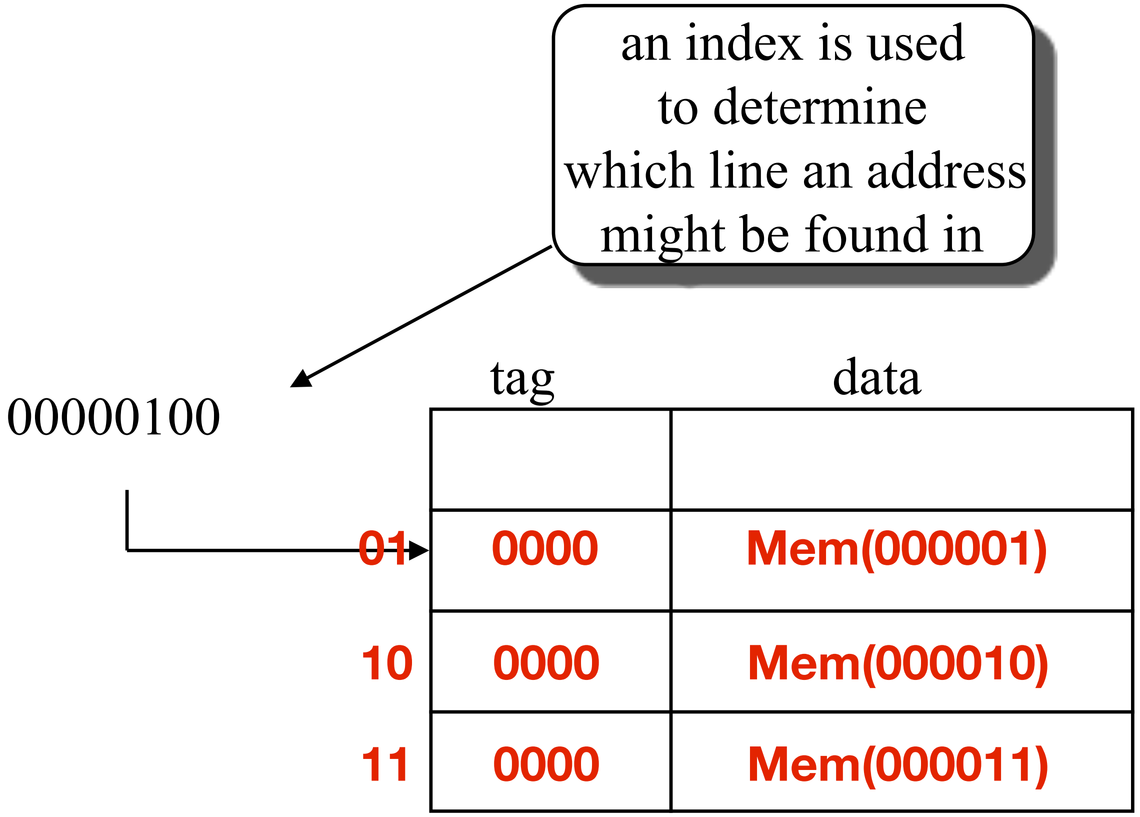
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100


an index is used to determine which line an address might be found in
00000100

	tag	data
01	0000	Mem(000001)
10	0000	Mem(000010)
11	0000	Mem(000011)

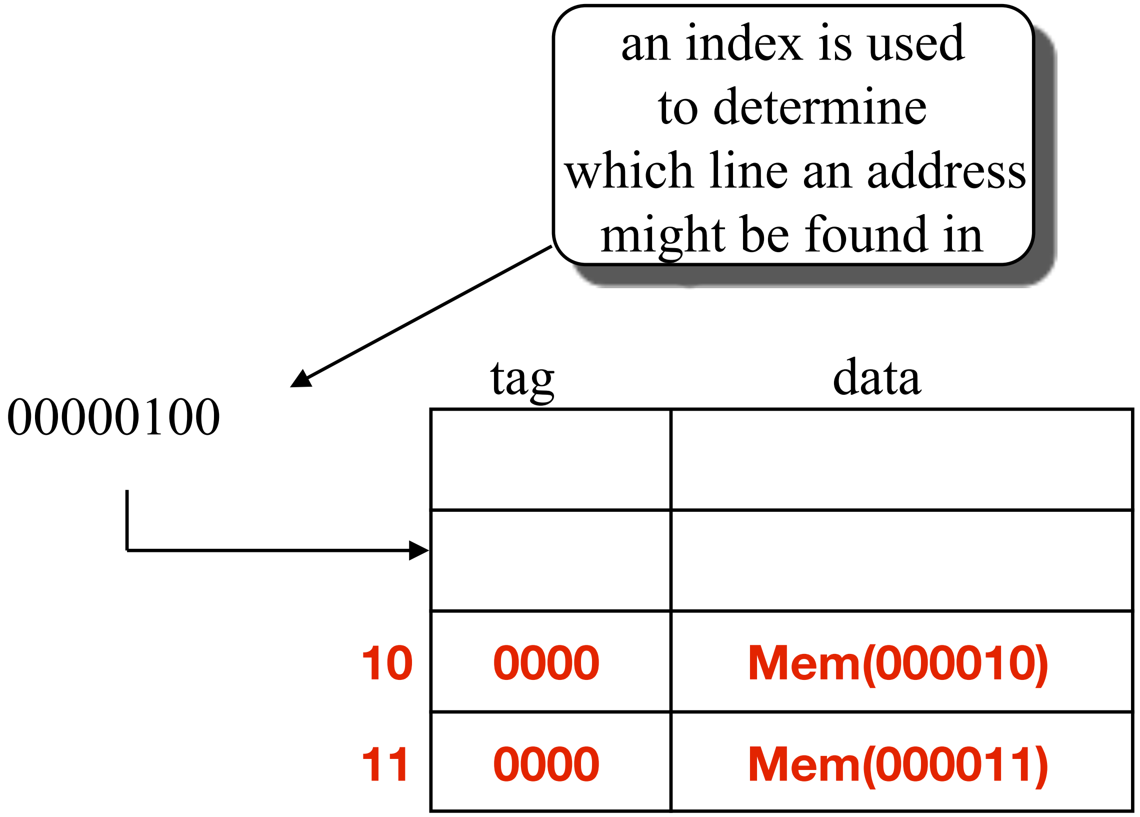
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

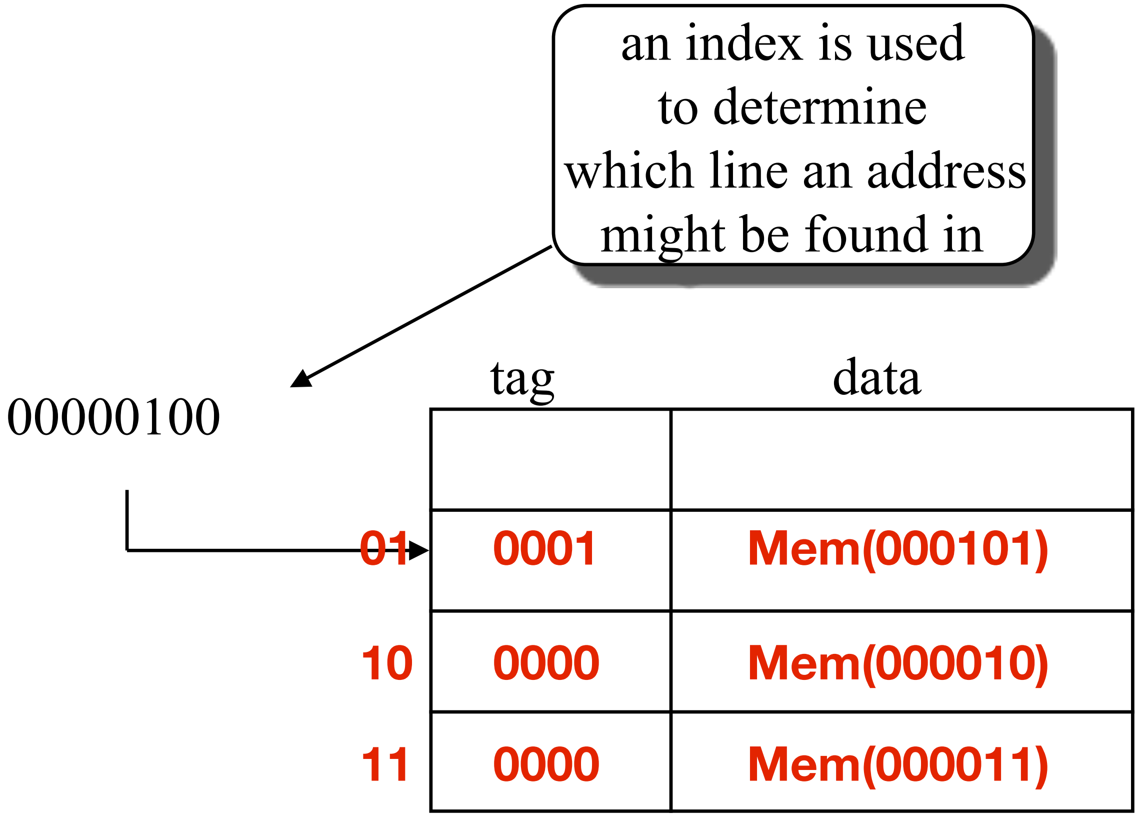
	tag	data
10	0000	Mem(000010)
11	0000	Mem(000011)

4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



an index is used to determine which line an address might be found in

00000100

	tag	data
01	0001	Mem(000101)
10	0000	Mem(000010)
11	0000	Mem(000011)

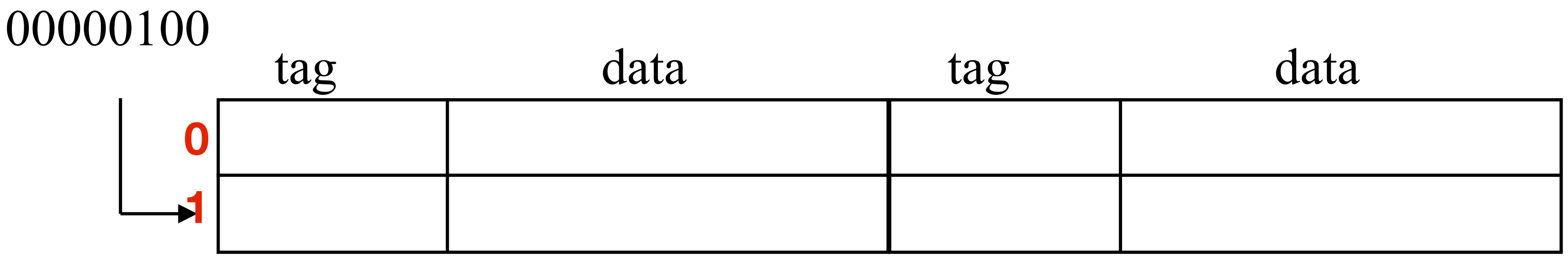
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



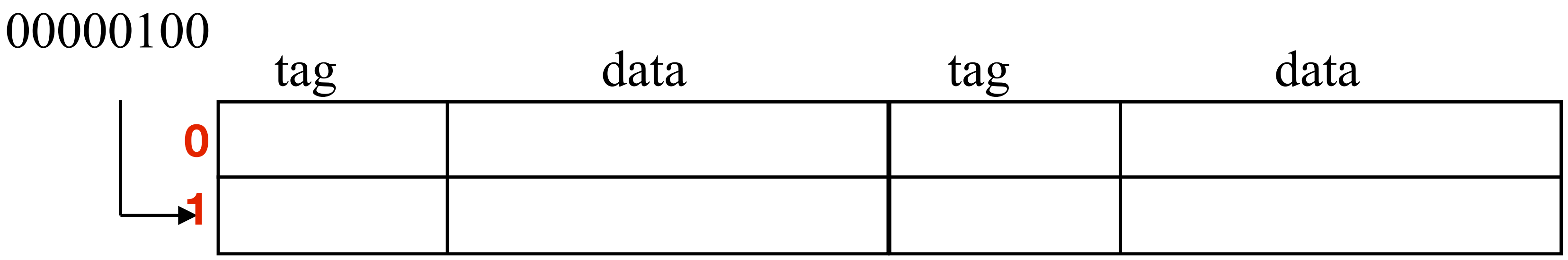
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called _________________.
- The cache lines/blocks that share the same index are a cache _____________.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



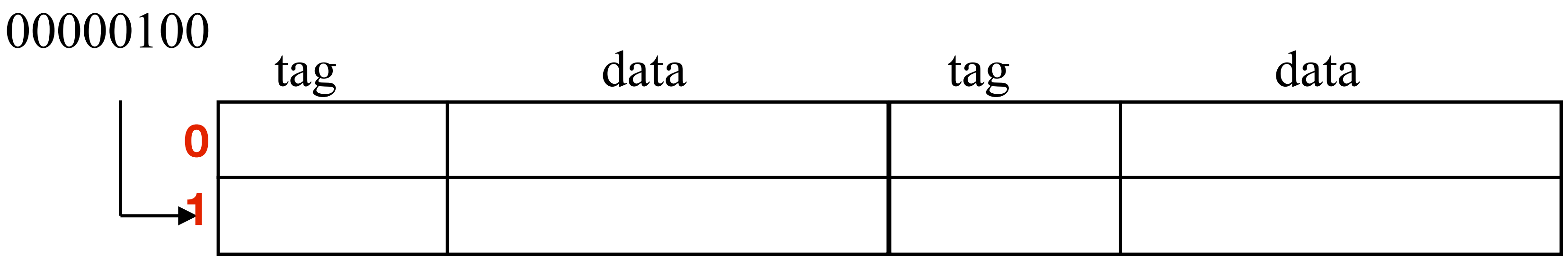
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called n-way set-associative ________________.
- The cache lines/blocks that share the same index are a cache ____________.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



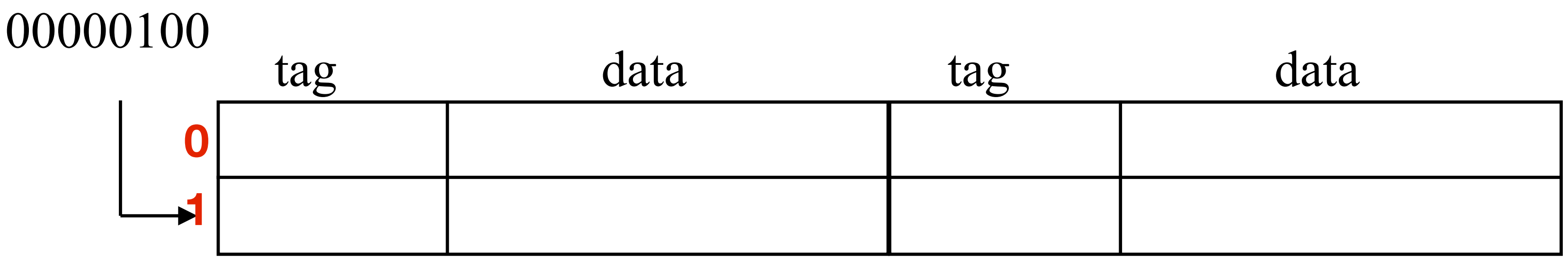
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



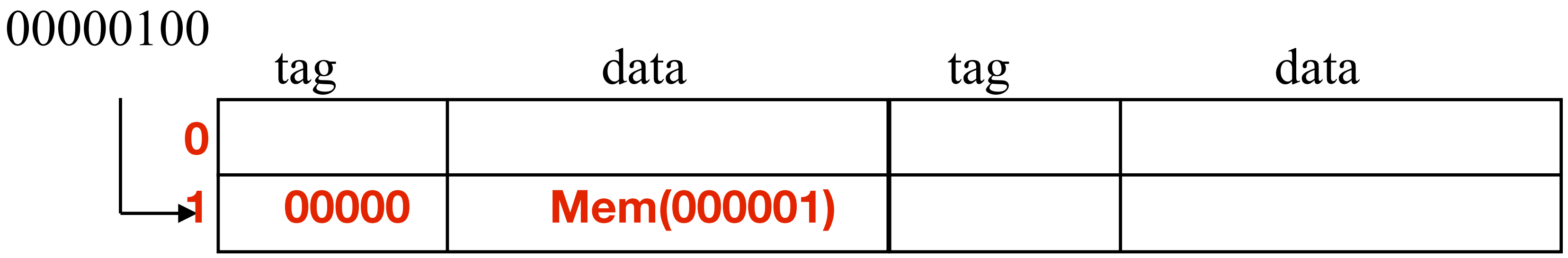
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

<u>address string:</u>

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	data	tag	data
0				
1	00000	Mem(000001)		

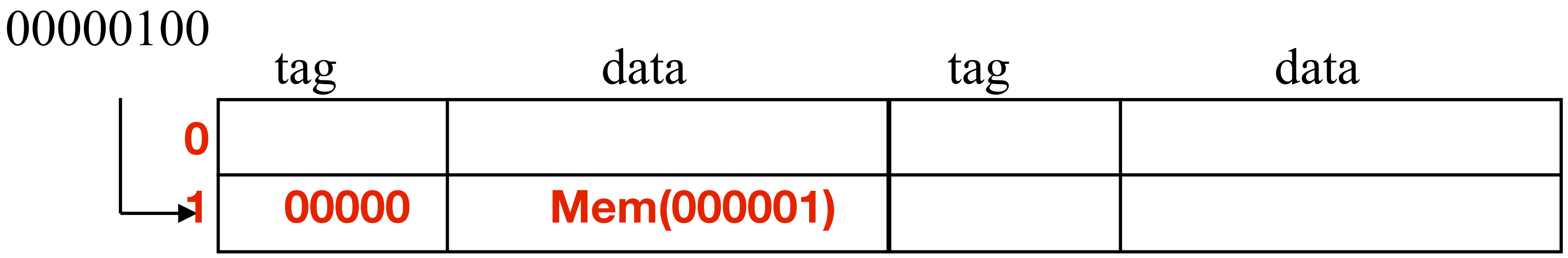
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

<u>address string:</u>

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



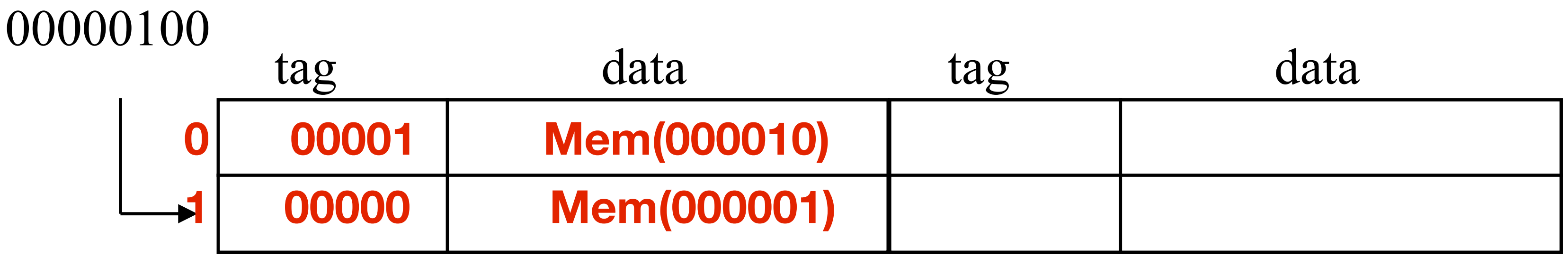
	tag	data	tag	data
0				
1	00000	Mem(000001)		

4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called n-way set-associative.
- The cache lines/blocks that share the same index are a cache set.

A Set Associative Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	data	tag	data
0	00001	Mem(000010)		
1	00000	Mem(000001)		

4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

00000100

	tag	data	tag	data
0	00001	Mem(000010)		
1	00000	Mem(000001)		

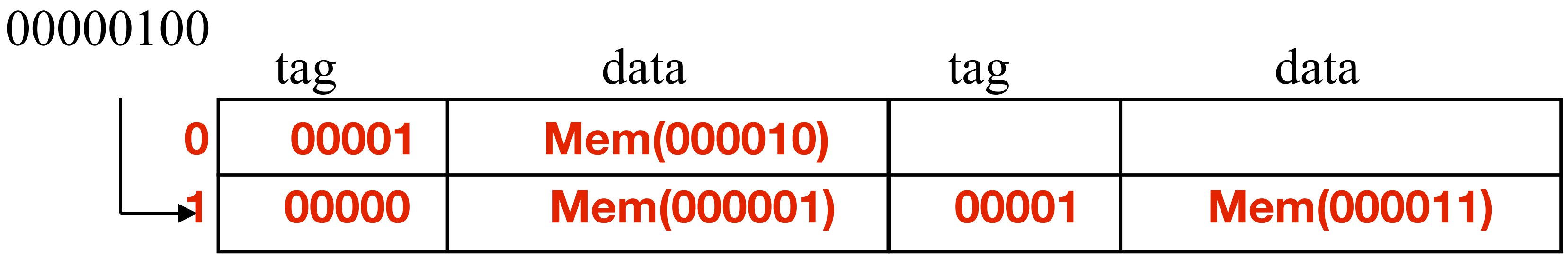
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	data	tag	data
0	00001	Mem(000010)		
1	00000	Mem(000001)	00001	Mem(000011)

4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called n-way set-associative.
- The cache lines/blocks that share the same index are a cache set.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

00000100

	tag	data	tag	data
0	00001	Mem(000010)		
1	00000	Mem(000001)	00001	Mem(000011)

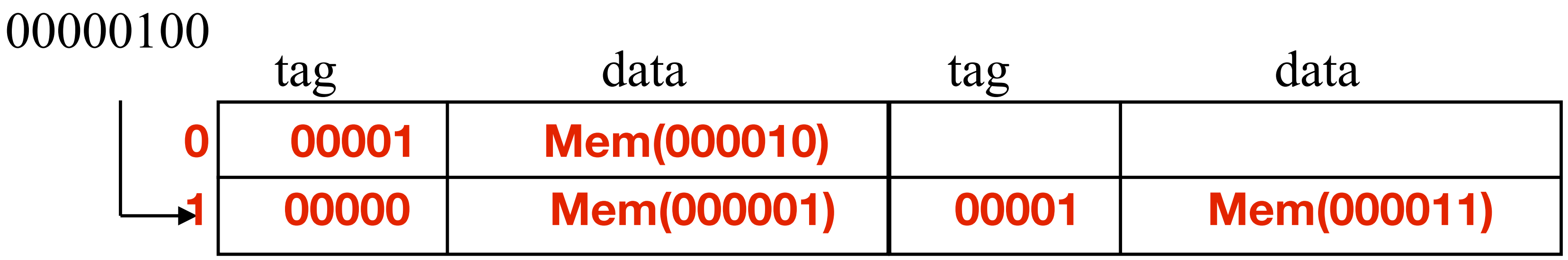
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	data	tag	data
0	00001	Mem(000010)		
1	00000	Mem(000001)	00001	Mem(000011)

4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called n-way set-associative.
- The cache lines/blocks that share the same index are a cache set.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

00000100

	tag	data	tag	data
0	00001	Mem(000010)		
1	00000	Mem(000001)	00001	Mem(000011)

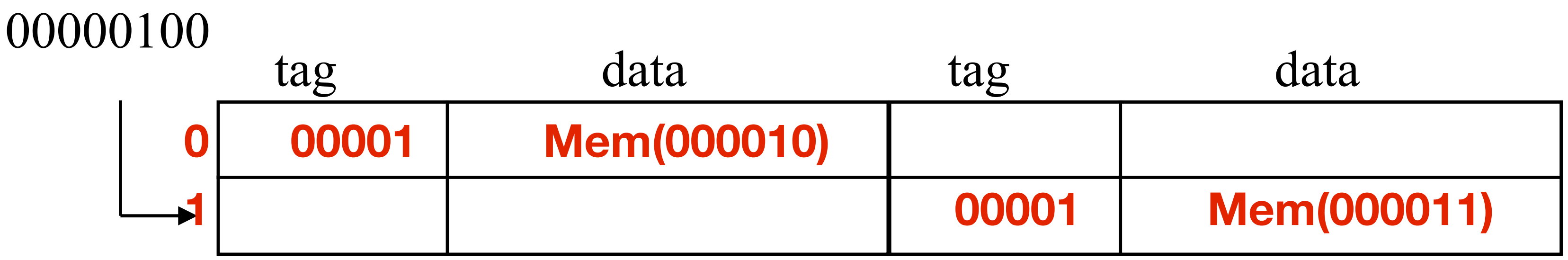
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	data	tag	data
0	00001	Mem(000010)		
1			00001	Mem(000011)

4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called n-way set-associative.
- The cache lines/blocks that share the same index are a cache set.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

00000100

	tag	data	tag	data
0	**00001**	**Mem(000010)**		
1	**00010**	**Mem(000101)**	**00001**	**Mem(000011)**

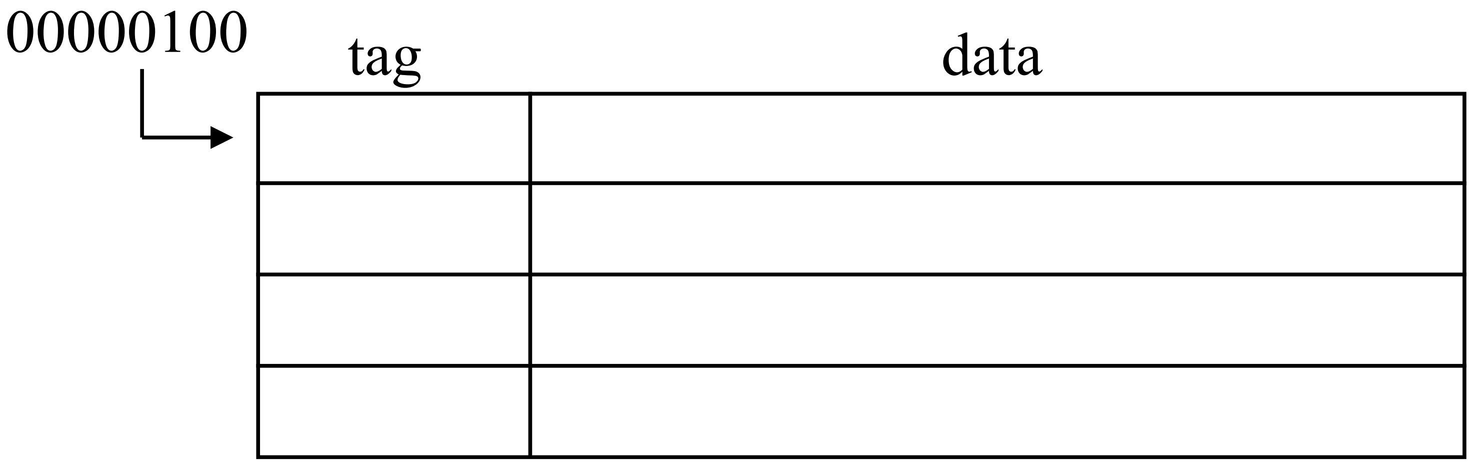
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



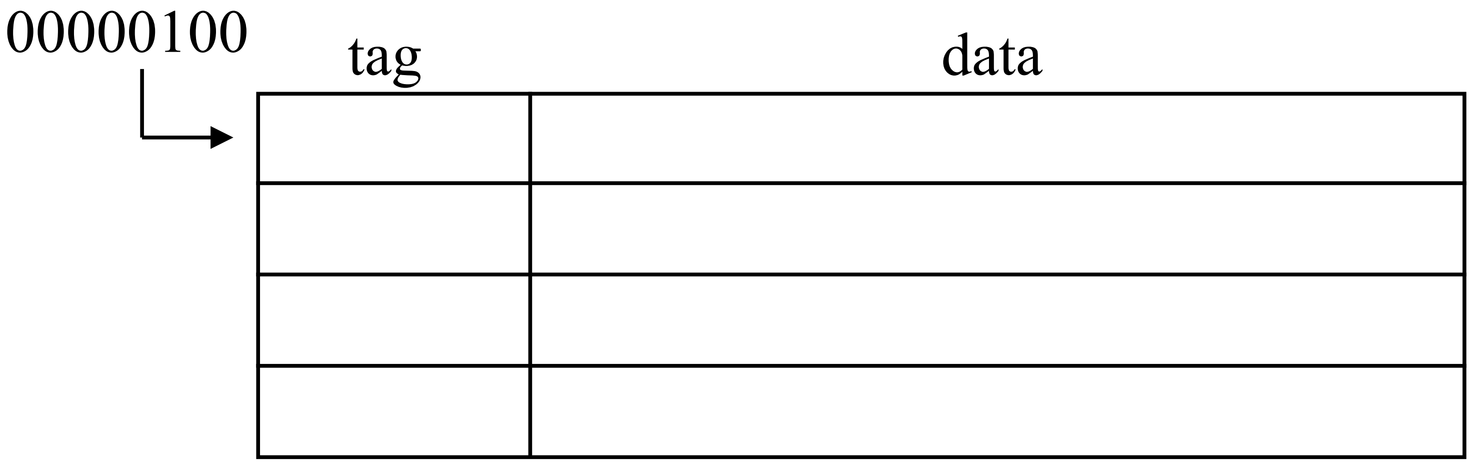
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of _______________.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



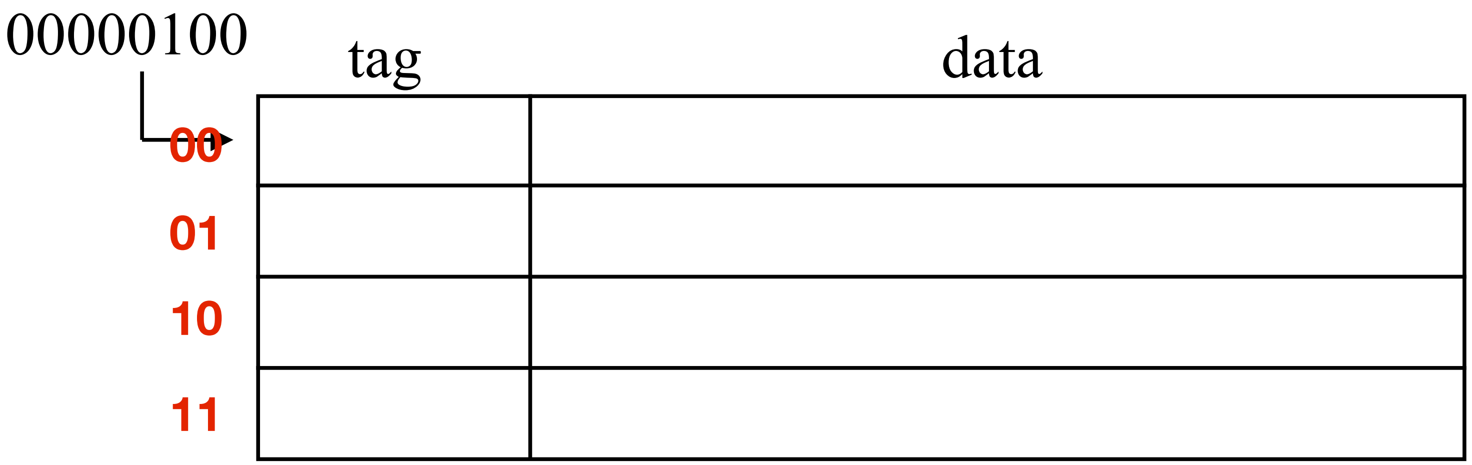
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



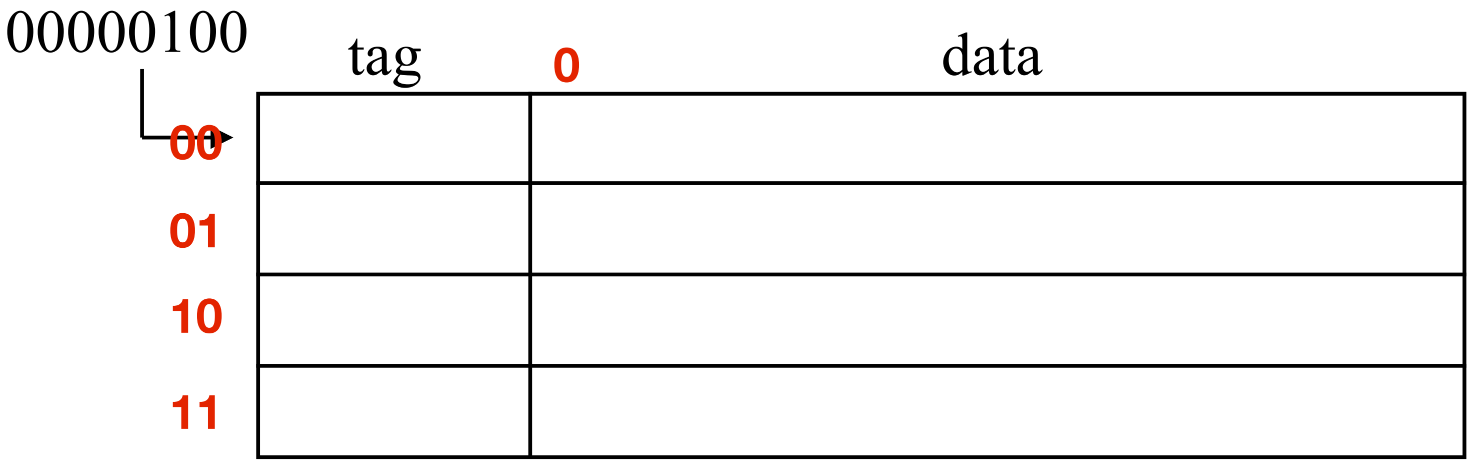
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	0 data
00		
01		
10		
11		

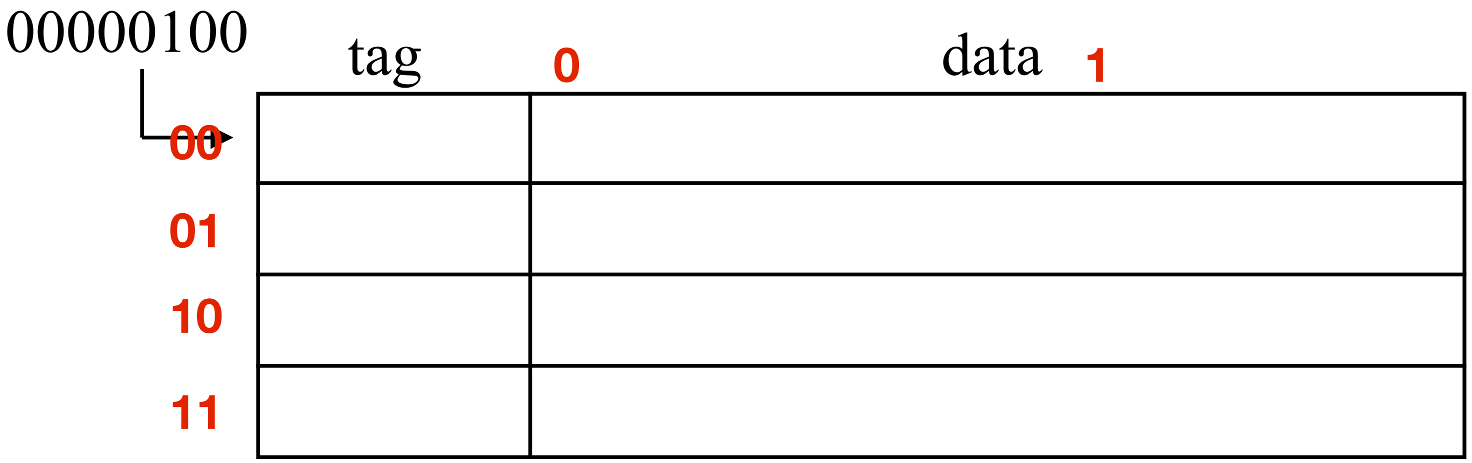
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

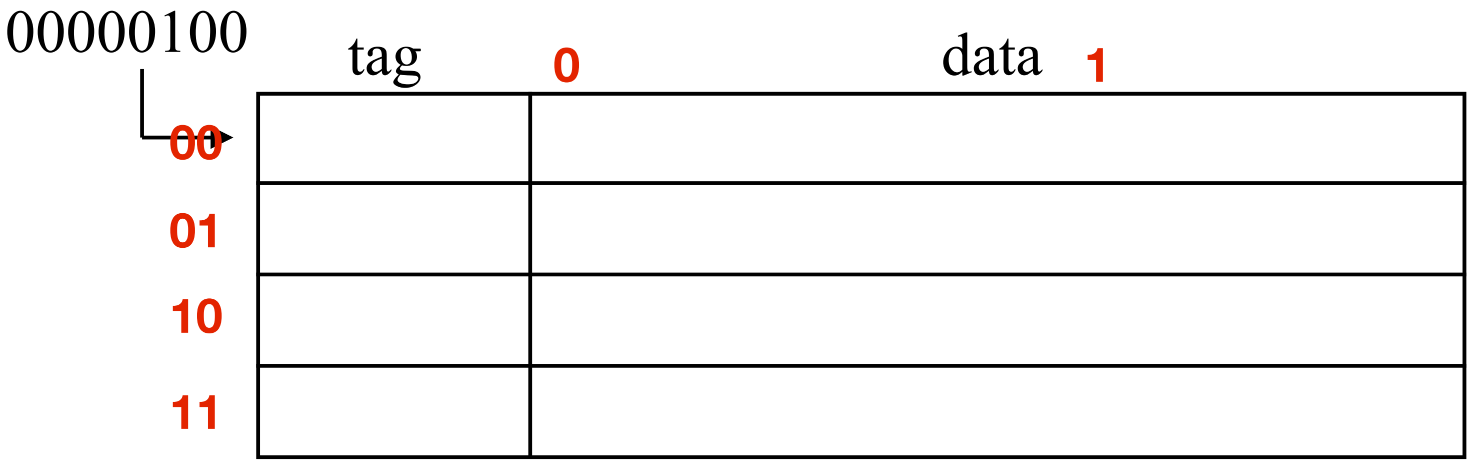


4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of Spacial Locality.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



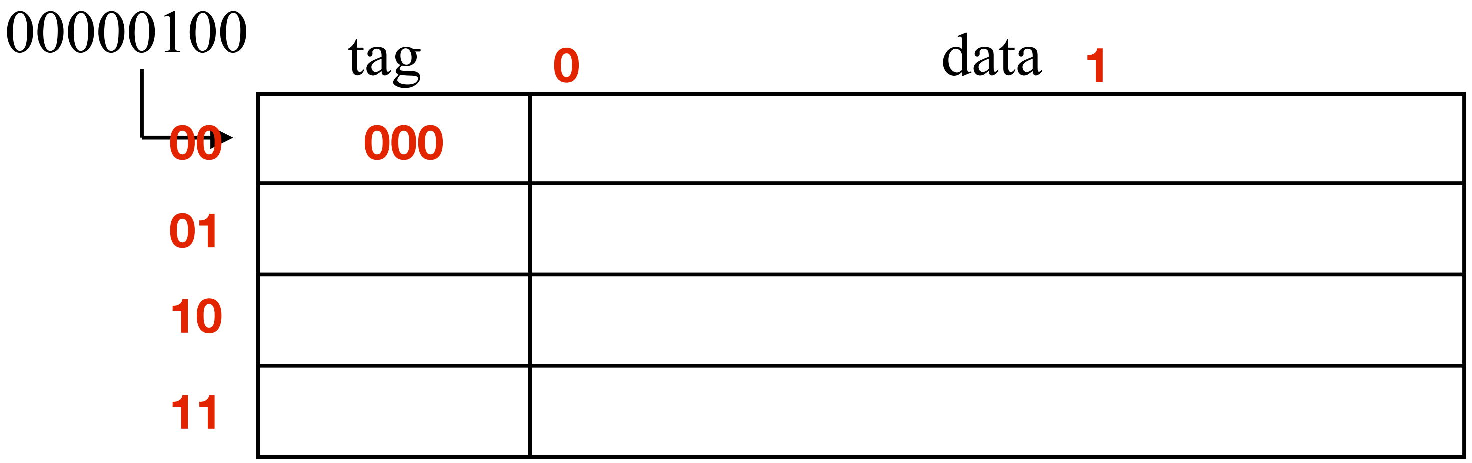
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



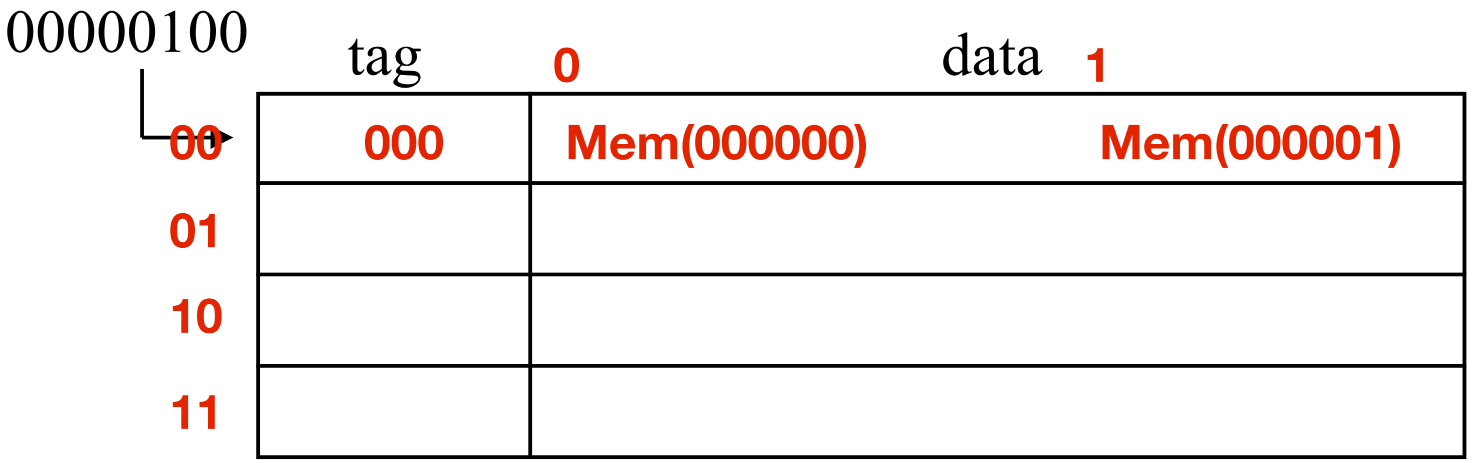
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of Spacial Locality.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

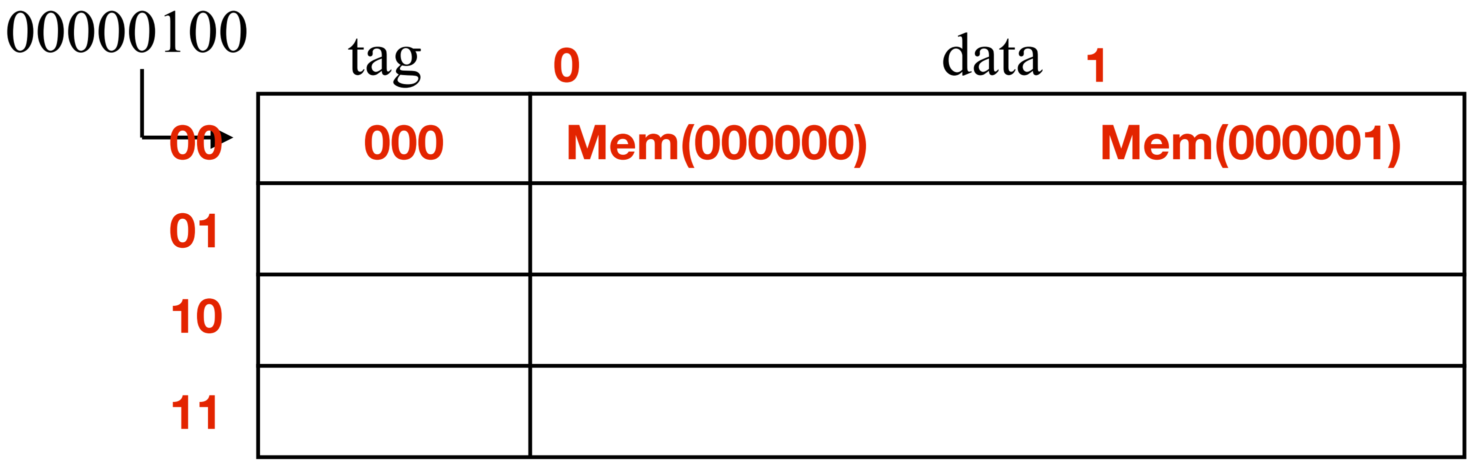
	tag	data	
		0	1
00	000	Mem(000000)	Mem(000001)
01			
10			
11			

4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of Spacial Locality.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	data	
		0	1
00	000	Mem(000000)	Mem(000001)
01			
10			
11			

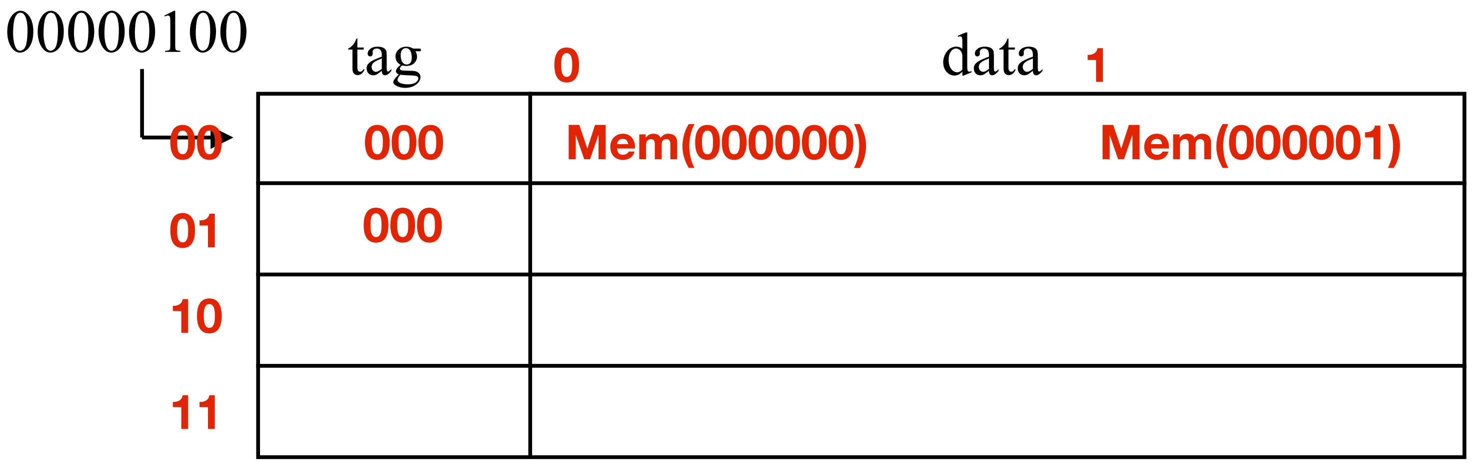
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of Spacial Locality.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



00000100

	tag	data	
		0	1
00	000	Mem(000000)	Mem(000001)
01	000		
10			
11			

4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of Spacial Locality.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

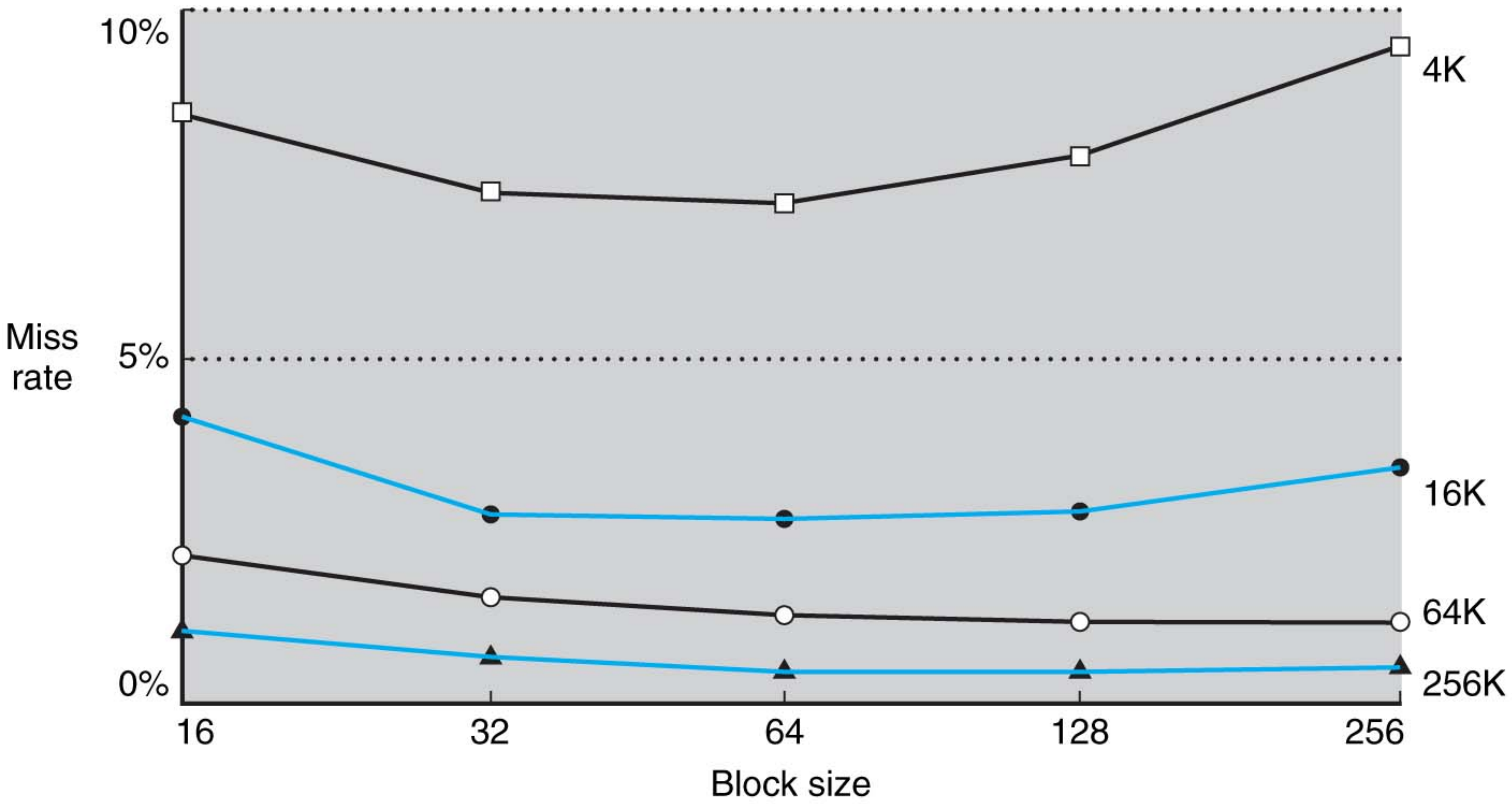
00000100

	tag	data	
		0	1
00	000	Mem(000000)	Mem(000001)
01	000	Mem(000010)	Mem(000011)
10			
11			

4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of Spacial Locality.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Block Size and Miss Rate



Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

128 x 32 = 4096 bytes = 4mb

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

128 x 32 = 4096 bytes = 4mb

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

131072 bytes / 512 = 256 bytes/set

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

128 x 32 = 4096 bytes = 4mb

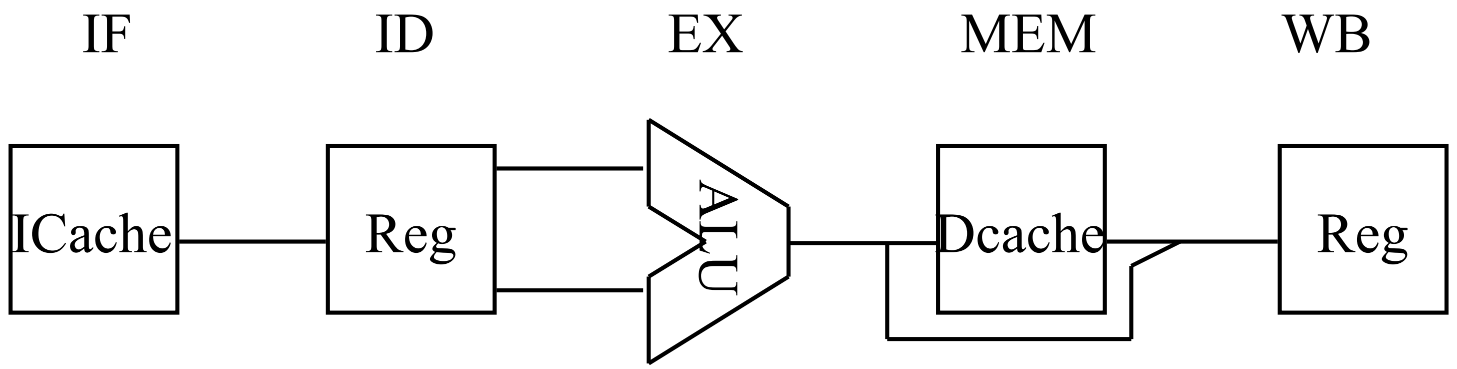
-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

131072 bytes / 512 = 256 bytes/set

256 bytes / 64 byte = 4 blocks/set = 4-way

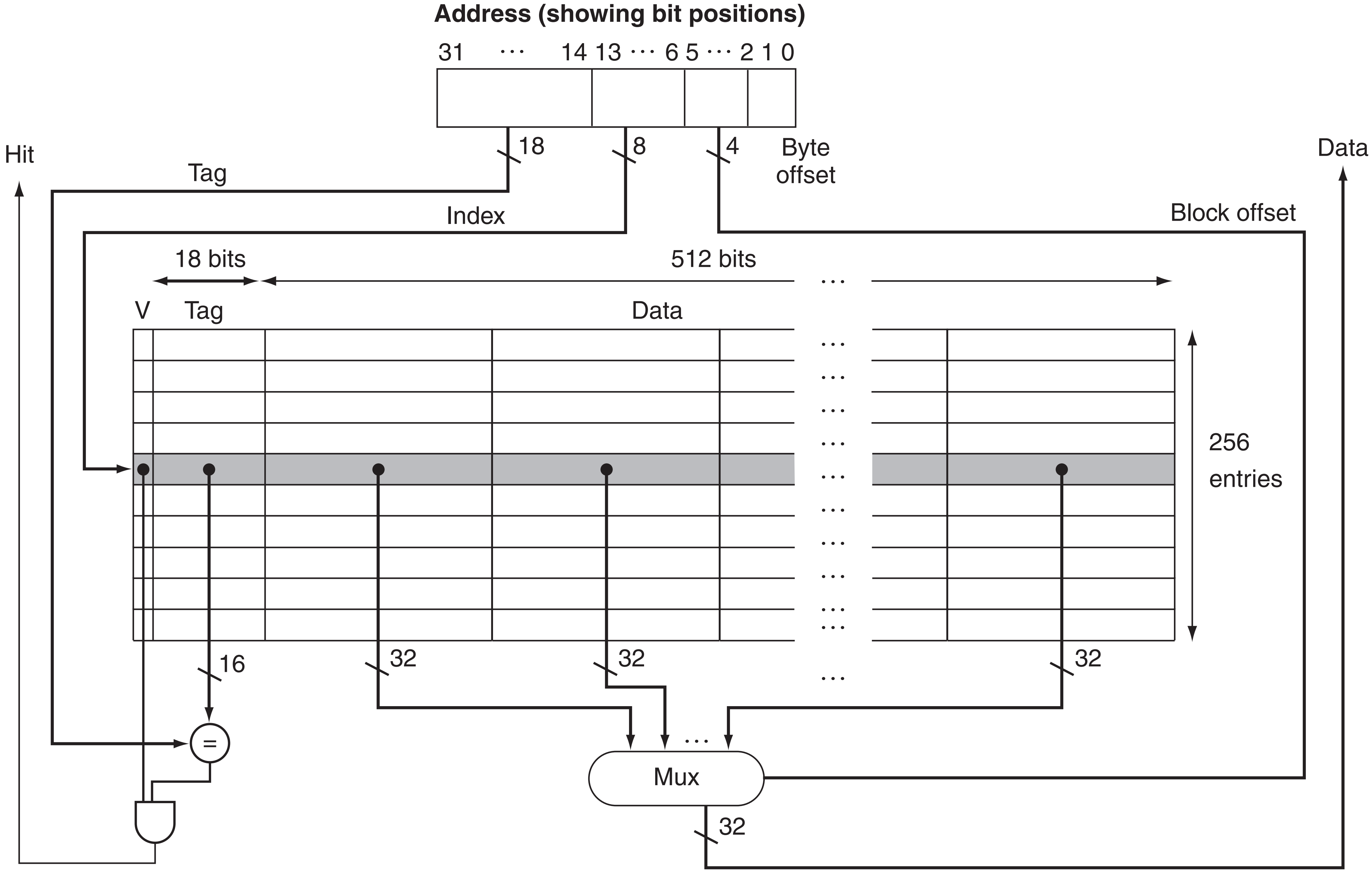
A Cache Access

- 1. Use index and tag to access cache and determine hit/miss.
- 2. If hit, return requested data.
- 3. If miss, select a cache block to be replaced, and access memory or next lower cache (possibly stalling the processor).
  - load entire missed cache line into cache
  - return requested data to CPU (or higher cache)
- 4. If next lower memory is a cache, goto step 1 for that cache.



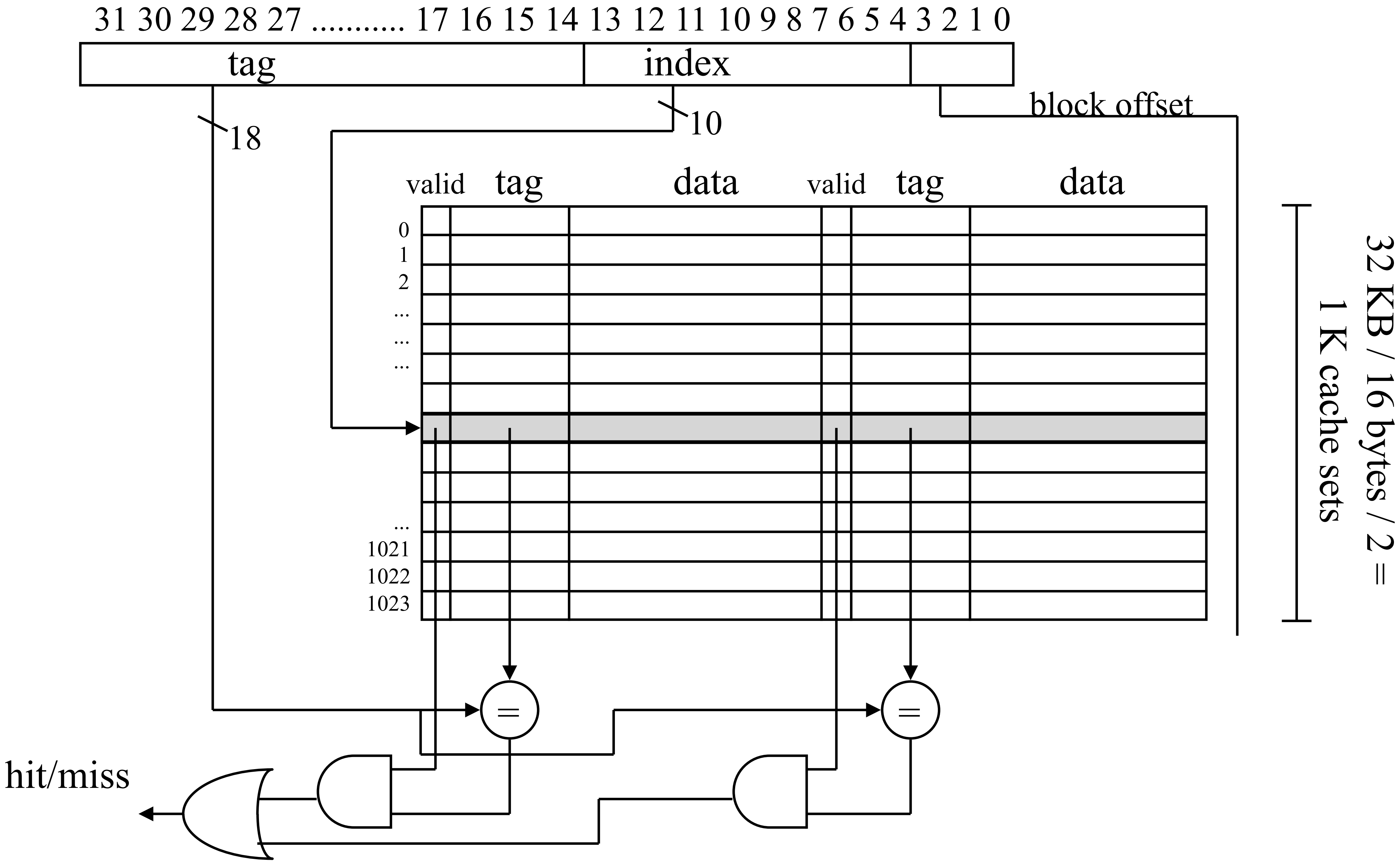
Accessing a Sample Cache

- 16 KB cache, direct-mapped, 64-byte cache block size



Accessing a Sample Cache

- 32 KB cache, 2-way set-associative, 16-byte block size



Associative Caches

- Higher hit rates, but...
- longer access time (longer to determine hit/miss, more muxing of outputs)
- more space (longer tags compared to DM)
  - 2-way extra 1 bit
  - 4-way extra 2 bits

Handling Stores

- Keep memory and cache identical?
- => all writes go to both cache and main memory
- => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
- Make room in cache for store miss?
- => on a store miss, bring written line into the cache
- => on a store miss, ignore cache

Handling Stores

- Keep memory and cache identical?
- **Write-through** => all writes go to both cache and main memory
- => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
- Make room in cache for store miss?
- => on a store miss, bring written line into the cache
- => on a store miss, ignore cache

Handling Stores

- Keep memory and cache identical?
- **Write-through** => all writes go to both cache and main memory
- **Write-back** => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
- Make room in cache for store miss?
- => on a store miss, bring written line into the cache
- => on a store miss, ignore cache

Handling Stores

- Keep memory and cache identical?
- **Write-through** => all writes go to both cache and main memory
- **Write-back** => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
- Make room in cache for store miss?
- **Write-allocate** => on a store miss, bring written line into the cache
- => on a store miss, ignore cache

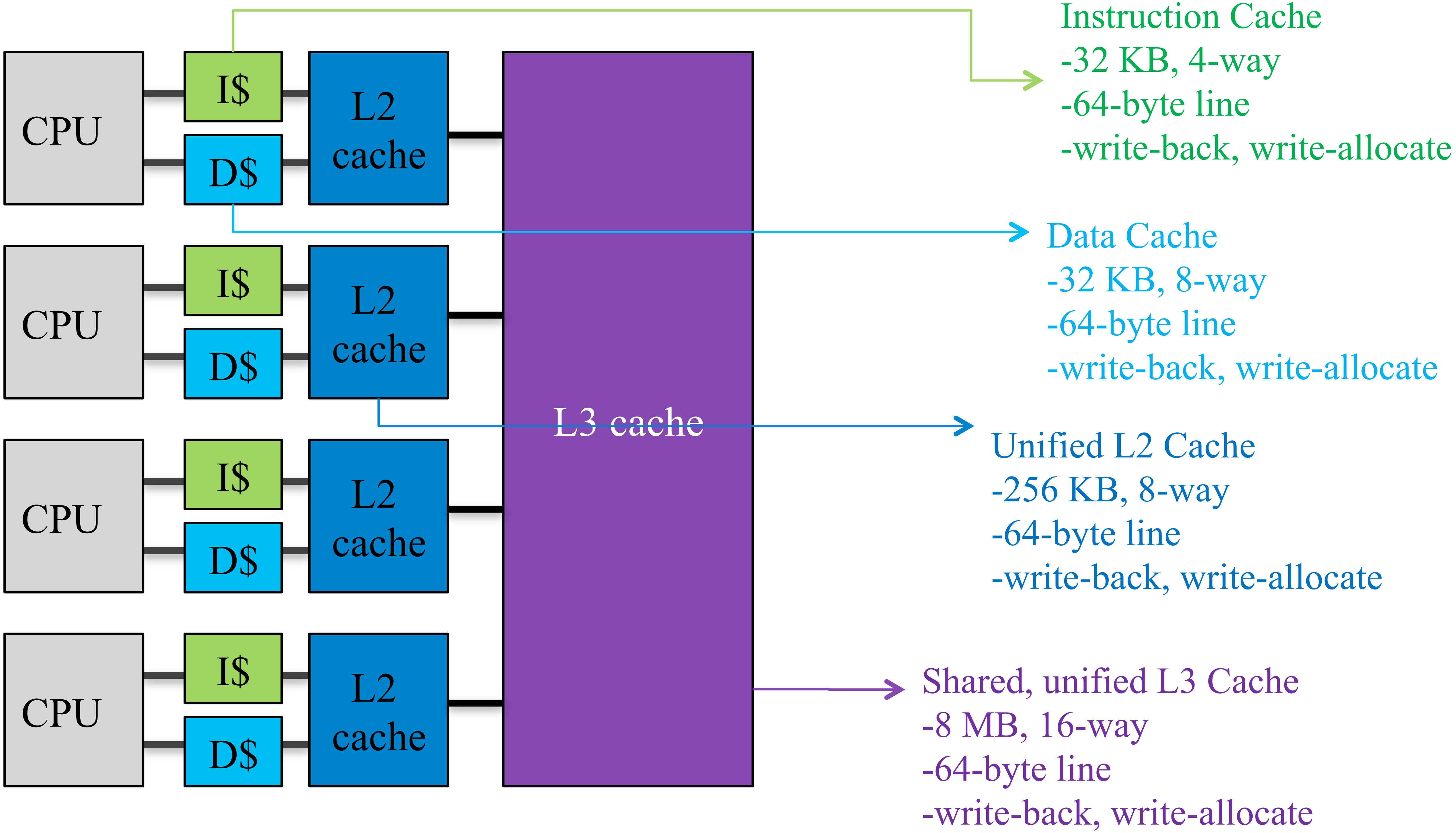
Handling Stores

- Keep memory and cache identical?
- **Write-through** => all writes go to both cache and main memory
- **Write-back** => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
- Make room in cache for store miss?
- **Write-allocate** => on a store miss, bring written line into the cache
- **Write-around** => on a store miss, ignore cache

The Three C's

- Compulsory (or cold-start) misses
    - first access to the data.
- Capacity misses
    - we missed only because the cache isn't big enough.
- Conflict misses
    - we missed because the data maps to the same line as other data that forced it out of the cache.

Modern Caches



Instruction Cache
-32 KB, 4-way
-64-byte line
-write-back, write-allocate

Data Cache
-32 KB, 8-way
-64-byte line
-write-back, write-allocate

Unified L2 Cache
-256 KB, 8-way
-64-byte line
-write-back, write-allocate

Shared, unified L3 Cache
-8 MB, 16-way
-64-byte line
-write-back, write-allocate

Key Points

Key Points

- Caches give illusion of a large, cheap memory with the access time of a fast, expensive memory.

Key Points

- Caches give illusion of a large, cheap memory with the access time of a fast, expensive memory.
- Caches take advantage of memory locality, specifically temporal locality and spatial locality.

Key Points

- Caches give illusion of a large, cheap memory with the access time of a fast, expensive memory.

- Caches take advantage of memory locality, specifically temporal locality and spatial locality.

- Cache design presents many options (block size, cache size, associativity, write policy) that an architect must combine to minimize miss rate and access time to maximize performance