

# Cache Hierarchies

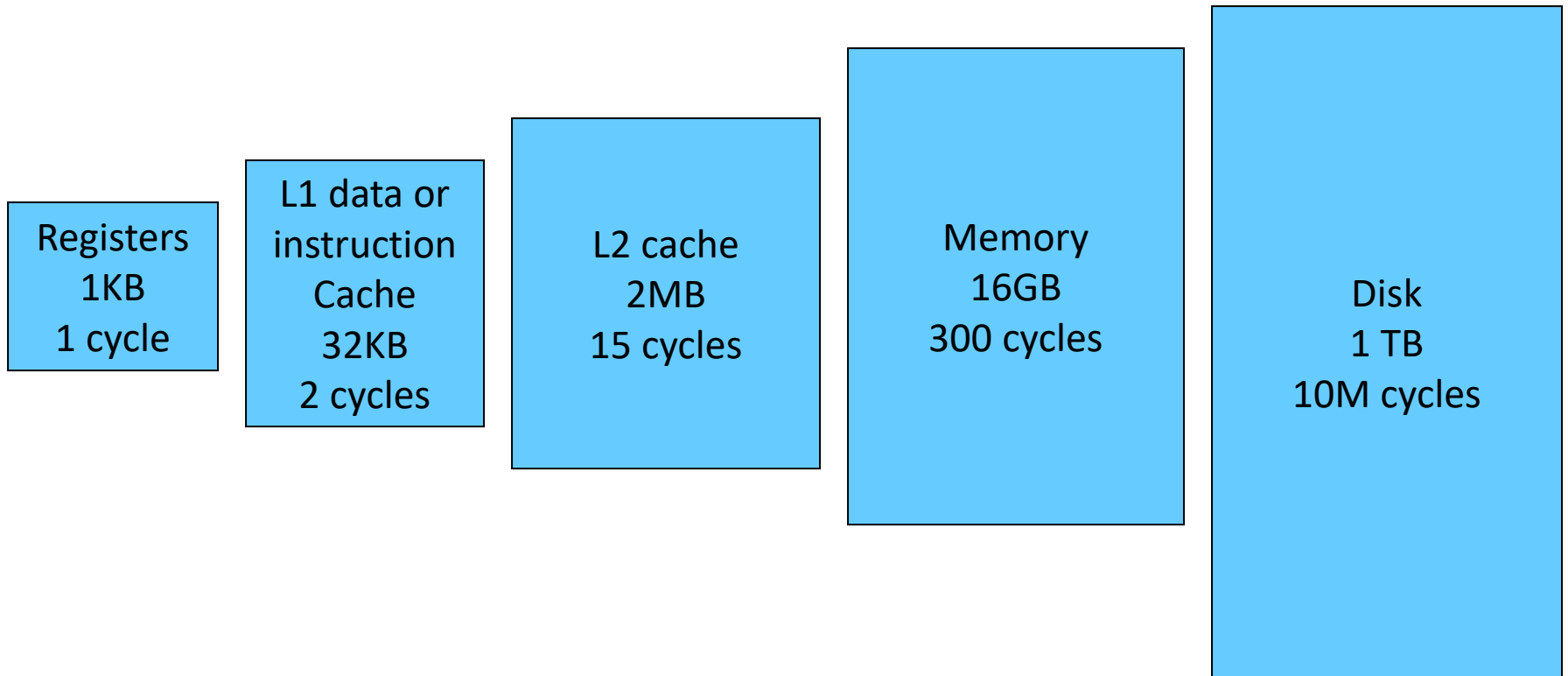
---

- Data and instructions are stored on DRAM chips – DRAM is a technology that has high bit density, but relatively poor latency – an access to data in memory can take as many as 300 cycles today!
- Hence, some data is stored on the processor in a structure called the cache – caches employ SRAM technology, which is faster, but has lower bit density
- Internet browsers also cache web pages – same concept

# Memory Hierarchy

---

- As you go further, capacity and latency increase



# Locality

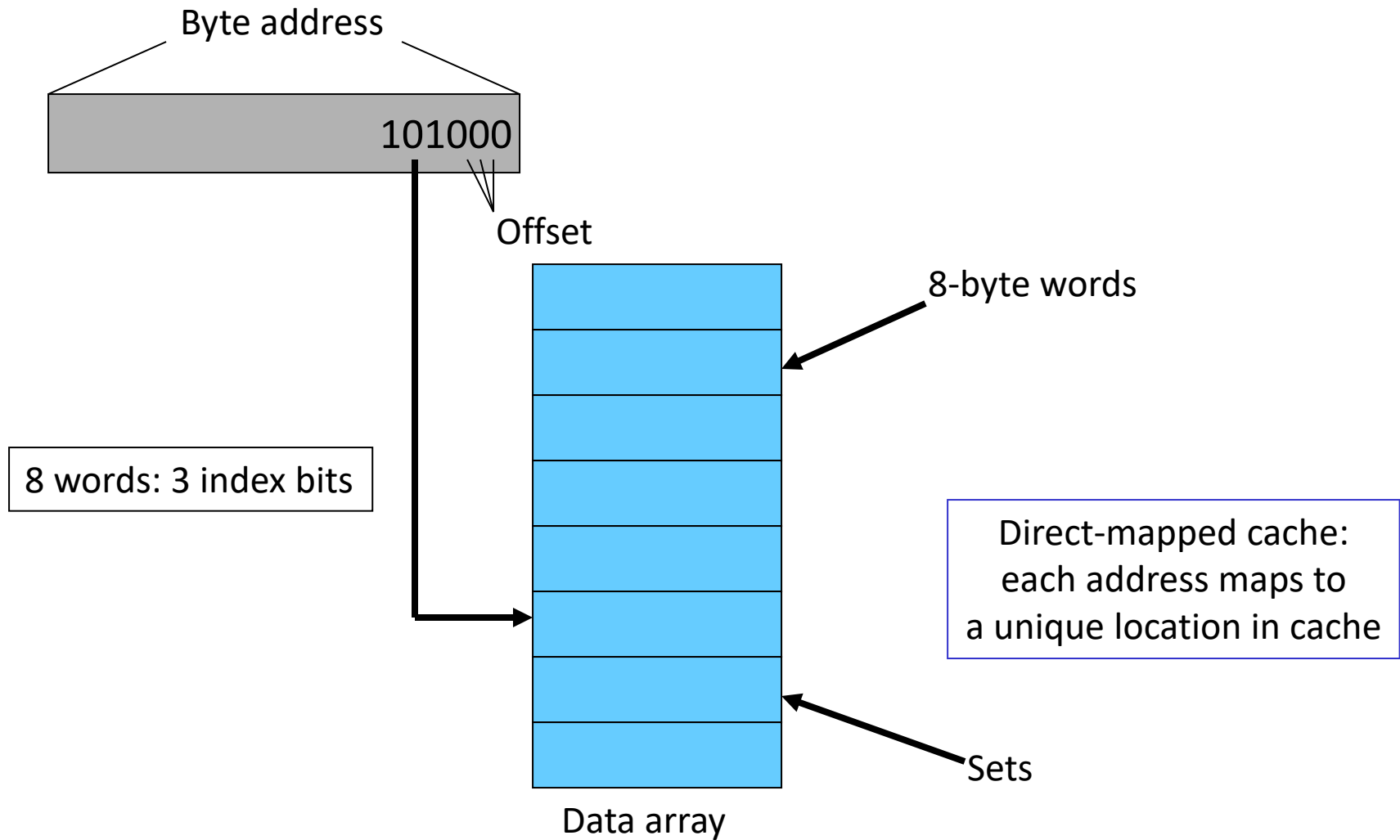
---

- Why do caches work?
  - Temporal locality: if you used some data recently, you will likely use it again
  - Spatial locality: if you used some data recently, you will likely access its neighbors
- No hierarchy: average access time for data = 300 cycles
- 32KB 1-cycle L1 cache that has a hit rate of 95%:  
average access time =  $0.95 \times 1 + 0.05 \times (301)$   
= 16 cycles

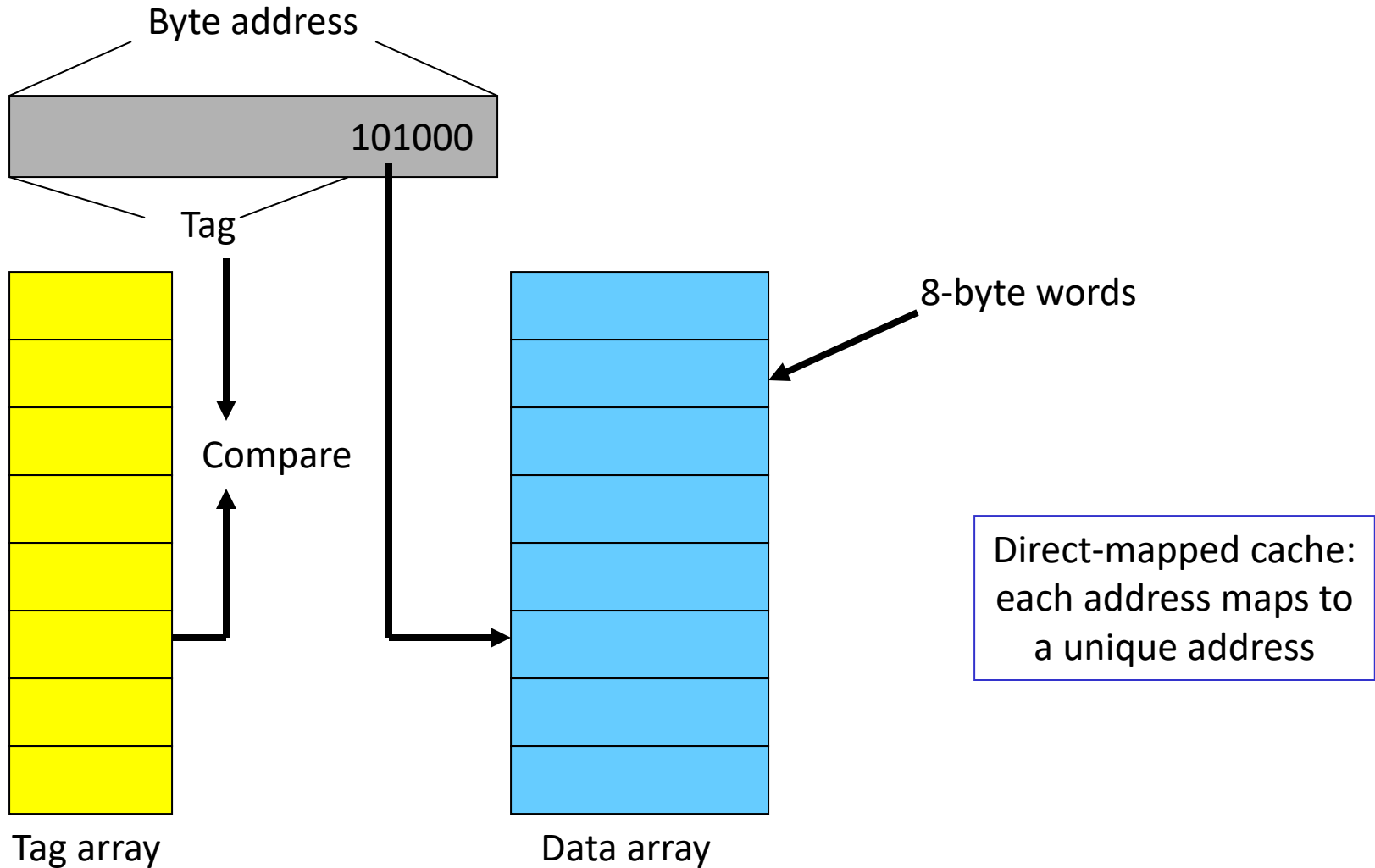
# Accessing the Cache

---

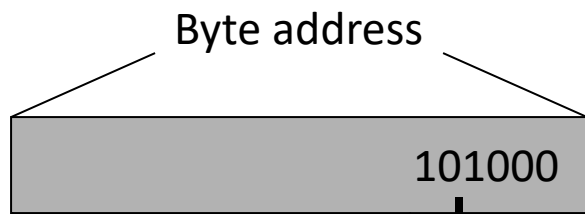
# Accessing the Cache



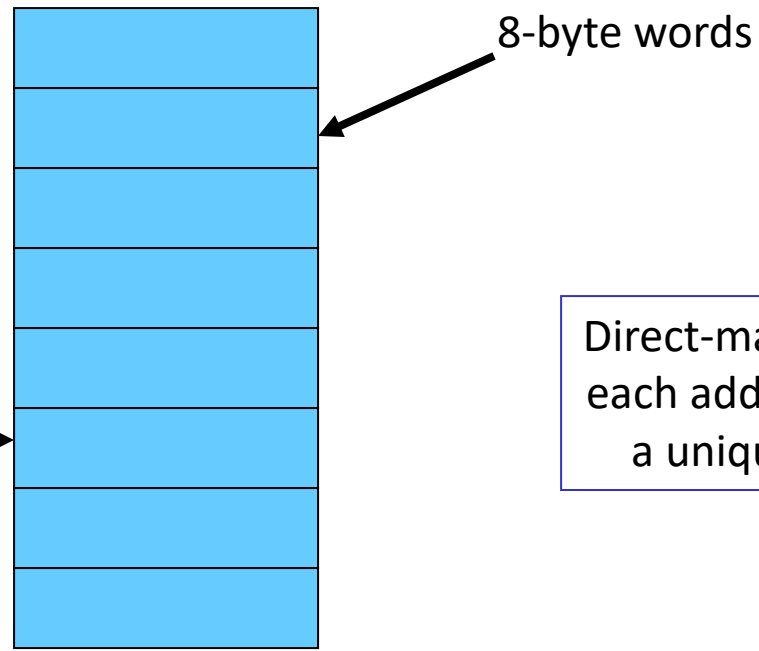
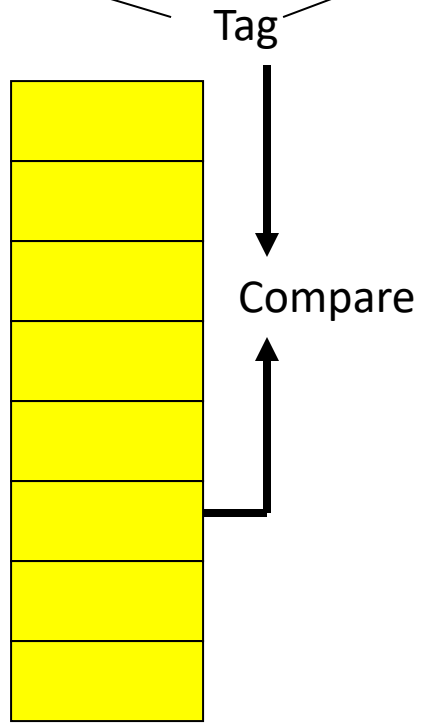
# The Tag Array



# Example Access Pattern



Assume that addresses are 8 bits long  
How many of the following address requests are hits/misses?  
4, 7, 10, 13, 16, 68, 73, 78, 83, 88, 4, 7, 10...

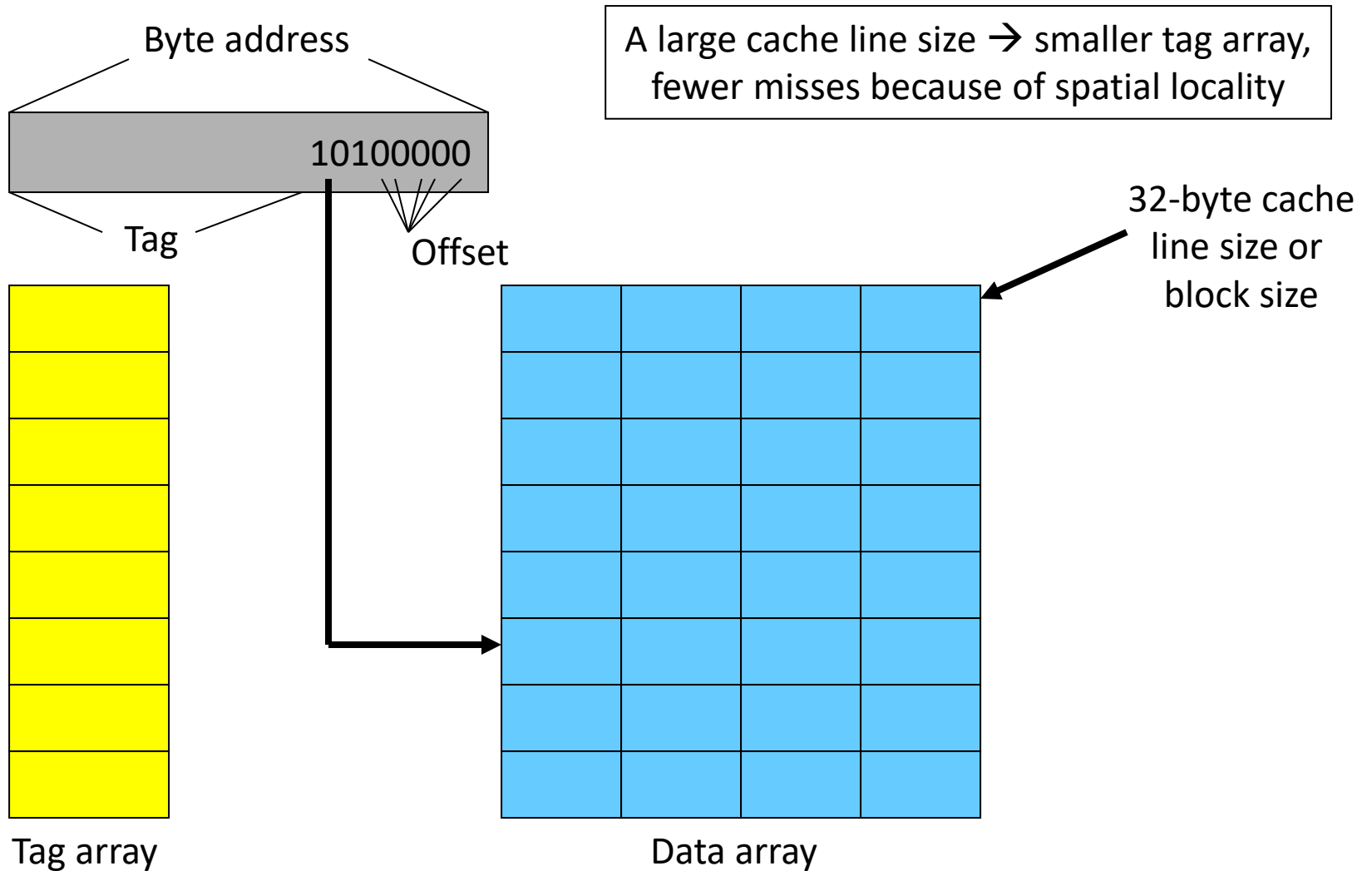


Direct-mapped cache:  
each address maps to  
a unique address

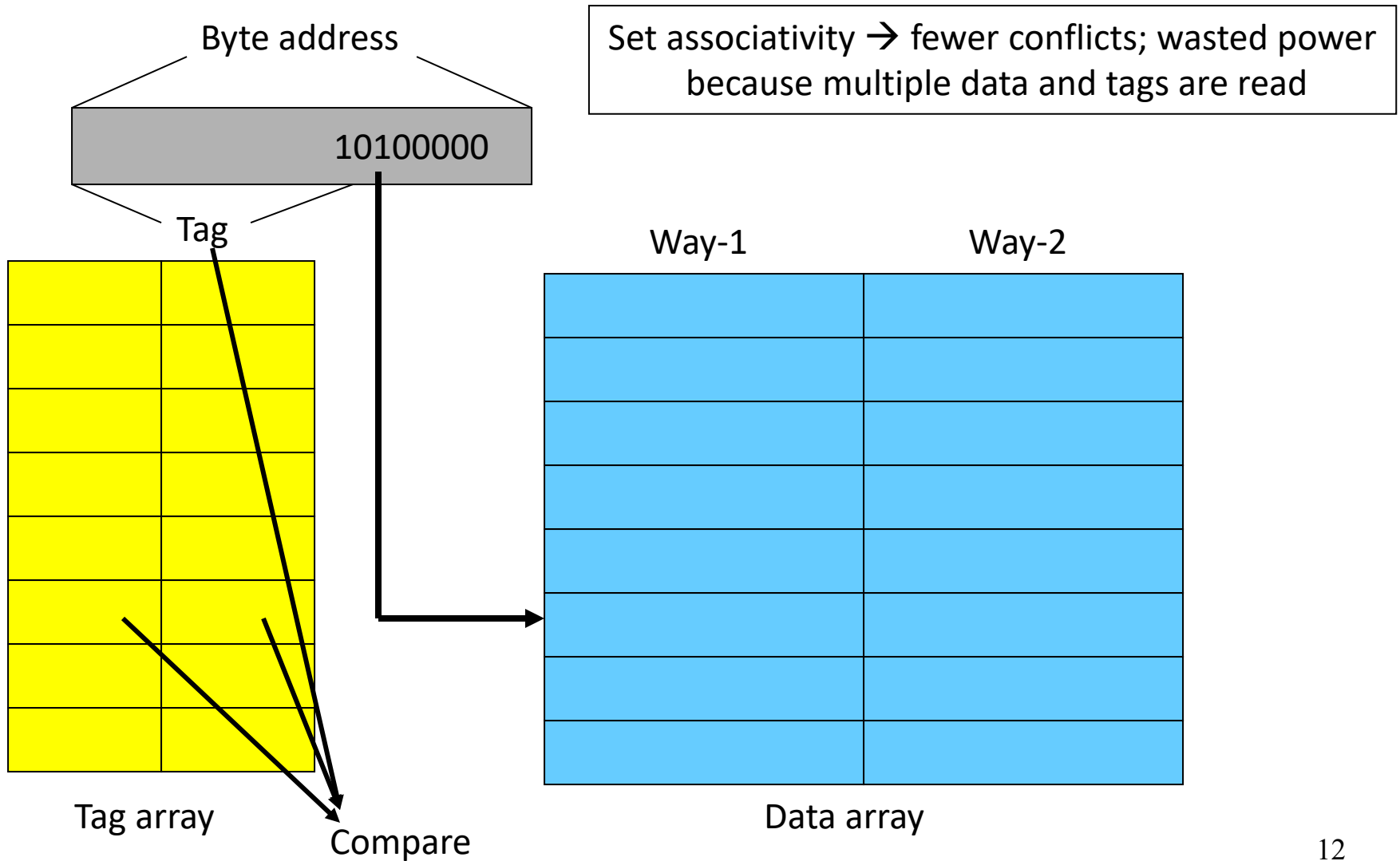
Tag array

Data array

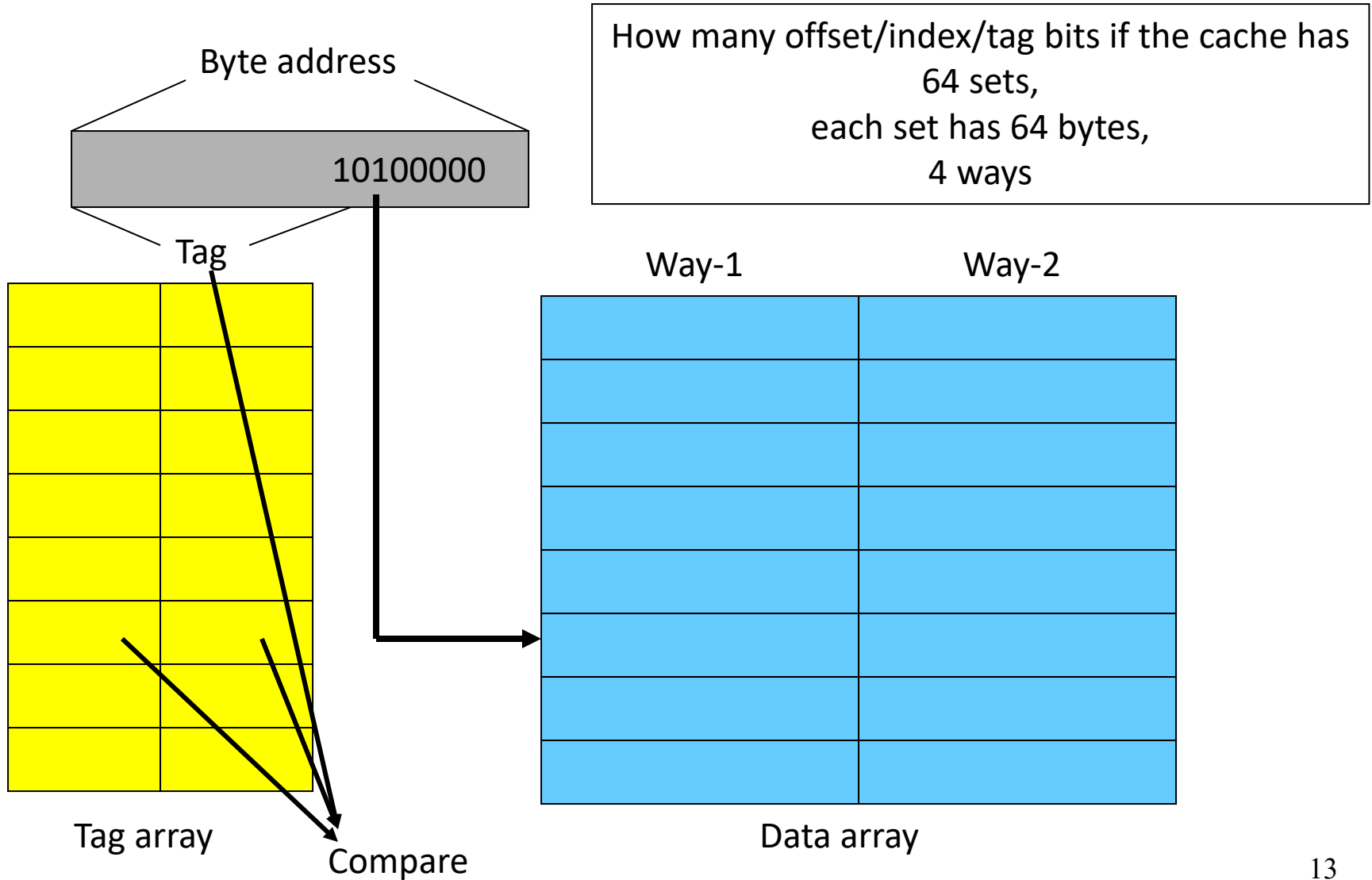
# Increasing Line Size



# Associativity



# Associativity



# Example

---

- 32 KB 4-way set-associative data cache array with 32 byte line sizes
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?

# Example

---

- 32 KB 4-way set-associative data cache array with 32 byte line sizes
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?

Cache size = #sets x #ways x blocksize

Index bits =  $\log_2(\text{sets})$

Offset bits =  $\log_2(\text{blocksize})$

Addr width = tag + index + offset

# Example 1

---

- 32 KB 4-way set-associative data cache array with 32 byte line sizes

cache size = #sets x #ways x block size

- How many sets? 256
- How many index bits, offset bits, tag bits?

8	5	19
$\log_2(\text{sets})$	$\log_2(\text{blksize})$	addrsize-index-offset

- How large is the tag array?  
tag array size = #sets x #ways x tag size  
= 19 Kb = 2.375 KB

## Example 2

---

- A pipeline has CPI 1 if all loads/stores are L1 cache hits  
40% of all instructions are loads/stores  
85% of all loads/stores hit in 1-cycle L1  
50% of all (10-cycle) L2 accesses are misses  
Memory access takes 100 cycles  
What is the CPI?

## Example 2

---

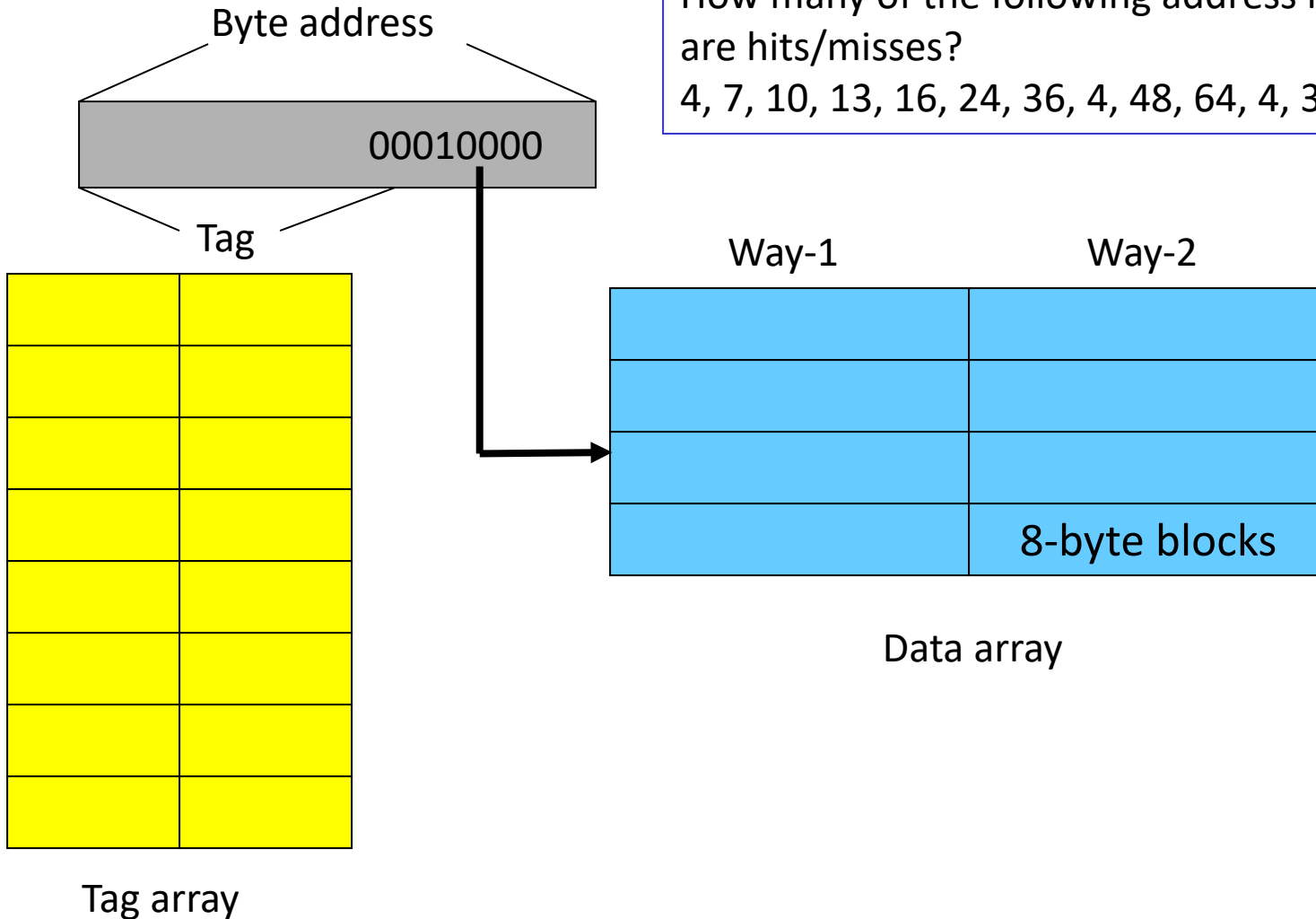
- A pipeline has CPI 1 if all loads/stores are L1 cache hits  
40% of all instructions are loads/stores  
85% of all loads/stores hit in 1-cycle L1  
50% of all (10-cycle) L2 accesses are misses  
Memory access takes 100 cycles  
What is the CPI?

Start with 1000 instructions

1000 cycles            (includes all 400 L1 accesses)  
+ 400 (ld/st) x 15% x 10 cycles (the L2 accesses)  
+ 400 x 15% x 50% x 100 cycles (the mem accesses)  
= 4,600 cycles  
CPI = 4.6

# Example 3

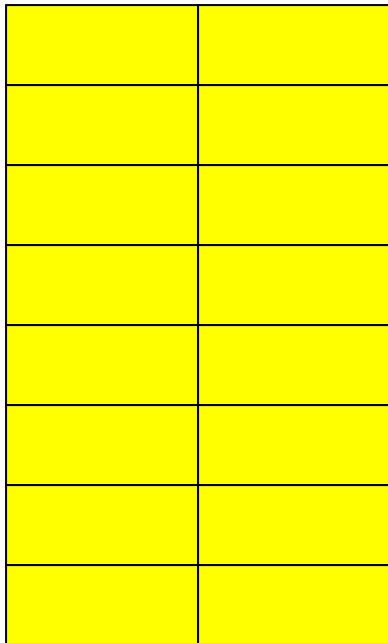
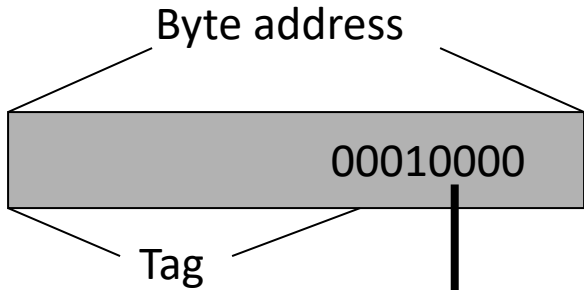
Assume that addresses are 8 bits long  
How many of the following address requests  
are hits/misses?  
4, 7, 10, 13, 16, 24, 36, 4, 48, 64, 4, 36, 64, 4



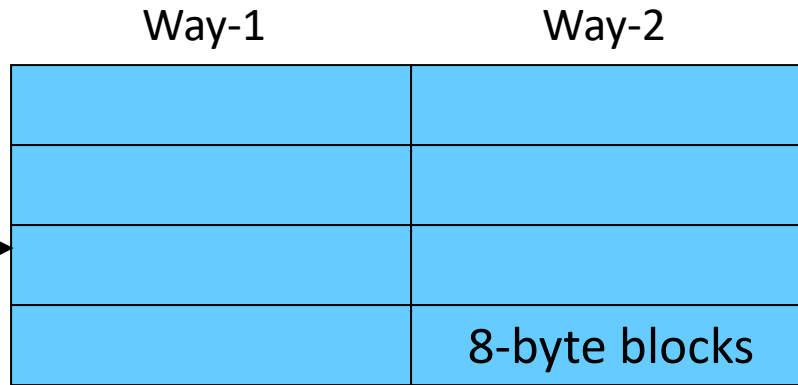
# Example 3

Assume that addresses are 8 bits long  
How many of the following address requests  
are hits/misses?

4, 7, 10, 13, 16, 24, 36, 4, 48, 64, 4, 36, 64, 4  
M H M H M M M H M M H M M M



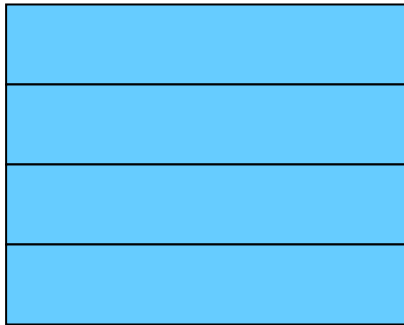
Tag array



Data array

# Example 0b

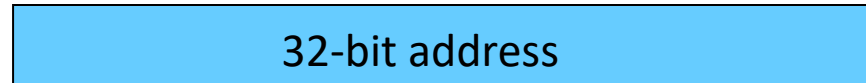
Show how the following addresses map to the cache and yield hits or misses. The cache is direct-mapped, has 16 sets, and a 64-byte block size. Addresses: 8, 96, 32, 480, 976, 1040, 1096



.  
.  
.



Offset = address % 64 (address modulo 64, extract last 6)  
Index = address/64 % 16 (shift right by 6, extract last 4)  
Tag = address/1024 (shift address right by 10)



	22 bits tag	4 bits index	6 bits offset	
8:	0	0	8	M
96:	0	1	32	M
32:	0	0	32	H
480:	0	7	32	M
976:	0	15	16	M
1040:	1	0	16	M
1096:	1	1	8	M

# Cache Misses

---

- On a write miss, you may either choose to bring the block into the cache (write-allocate) or not (write-no-allocate)
- On a read miss, you always bring the block in (spatial and temporal locality) – but which block do you replace?
  - no choice for a direct-mapped cache
  - randomly pick one of the ways to replace
  - replace the way that was least-recently used (LRU)
  - FIFO replacement (round-robin)

# Writes

---

- When you write into a block, do you also update the copy in L2?
  - write-through: every write to L1 → write to L2
  - write-back: mark the block as dirty, when the block gets replaced from L1, write it to L2
- Writeback coalesces multiple writes to an L1 block into one L2 write
- Writethrough simplifies coherency protocols in a multiprocessor system as the L2 always has a current copy of data

# Types of Cache Misses

---

- Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache
- Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache
- Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache