

Lecture 21: Memory Hierarchy

- Today's topics:
 - Cache organization
 - Cache hits/misses

OoO Wrap-up

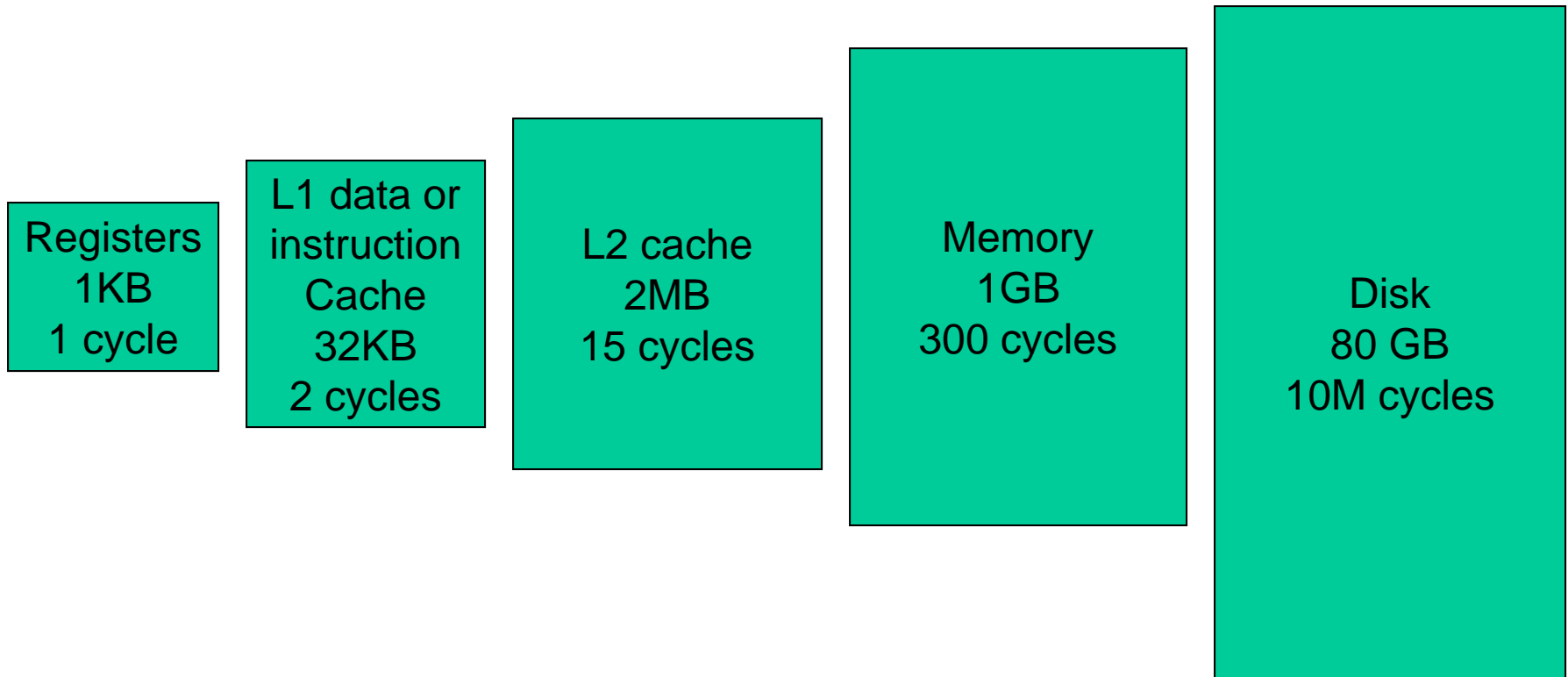
- How large are the structures in an out-of-order processor?
- What are the pros and cons of having larger/smaller structures?

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

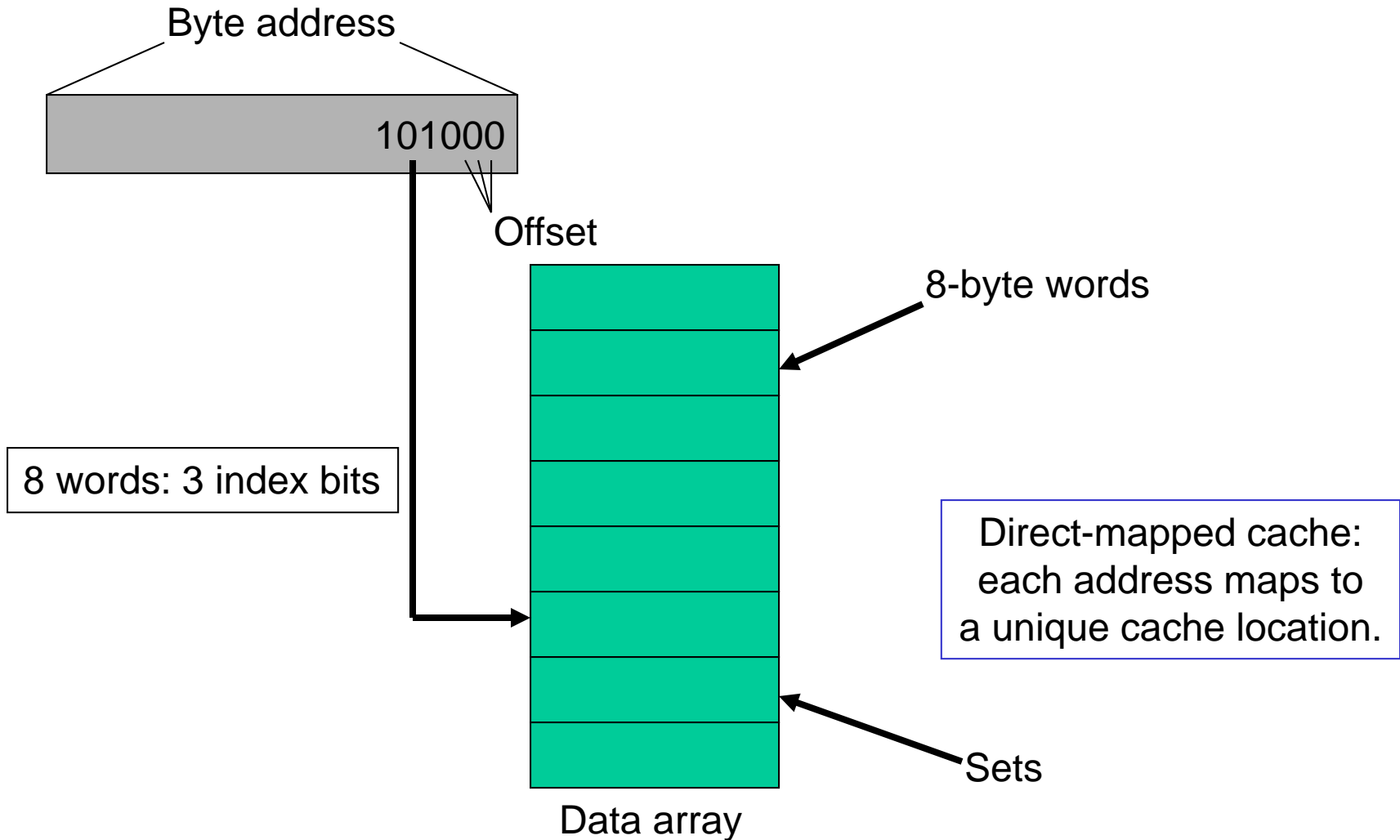
- As you go further, capacity and latency increase



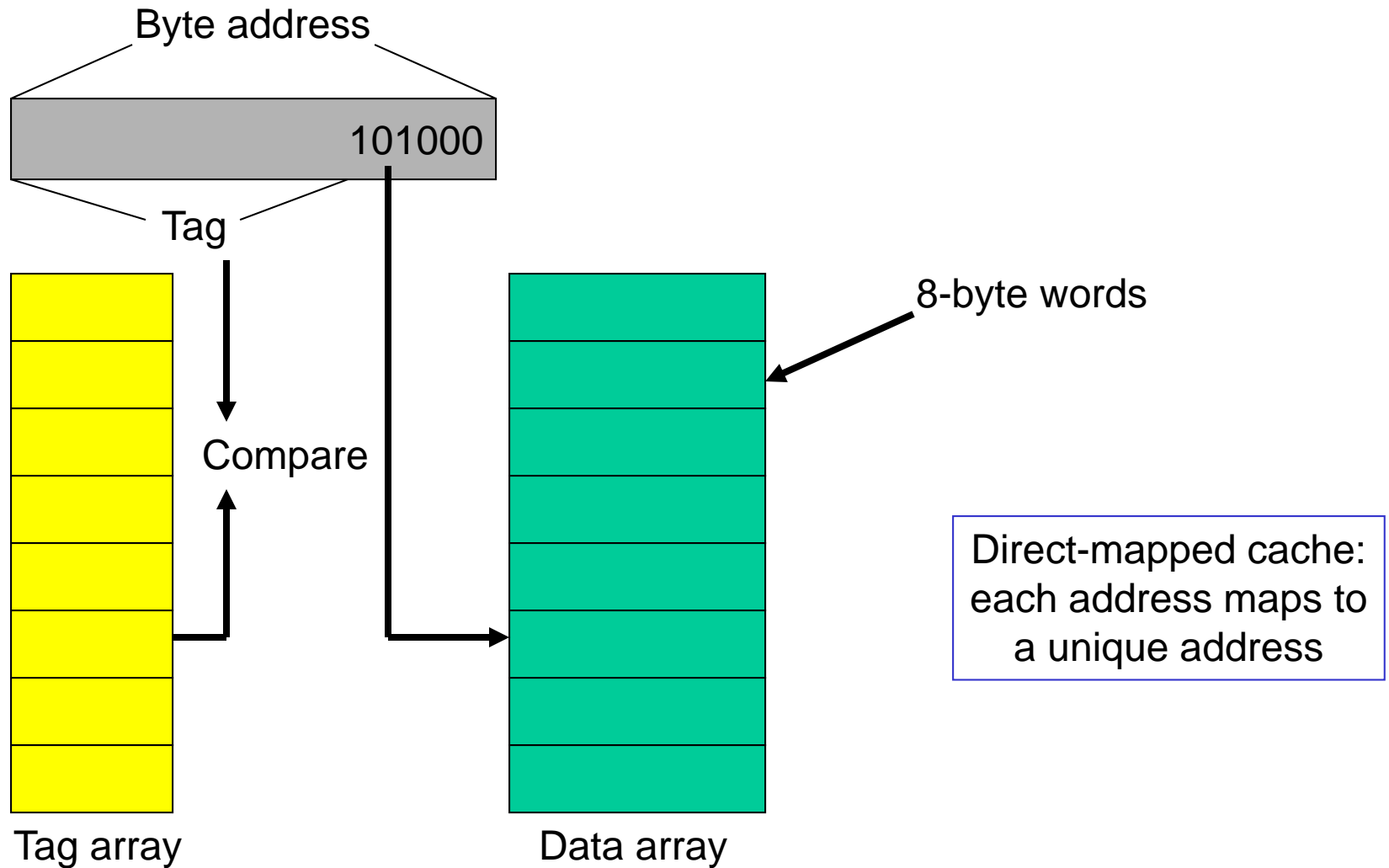
Locality

- 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



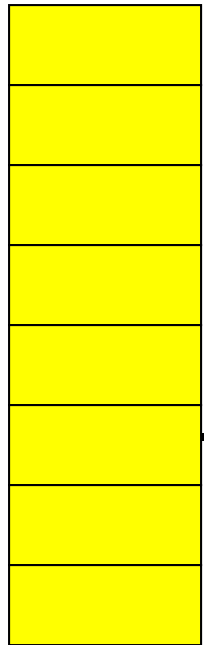
The Tag Array



Example Access Pattern



Tag



Compare

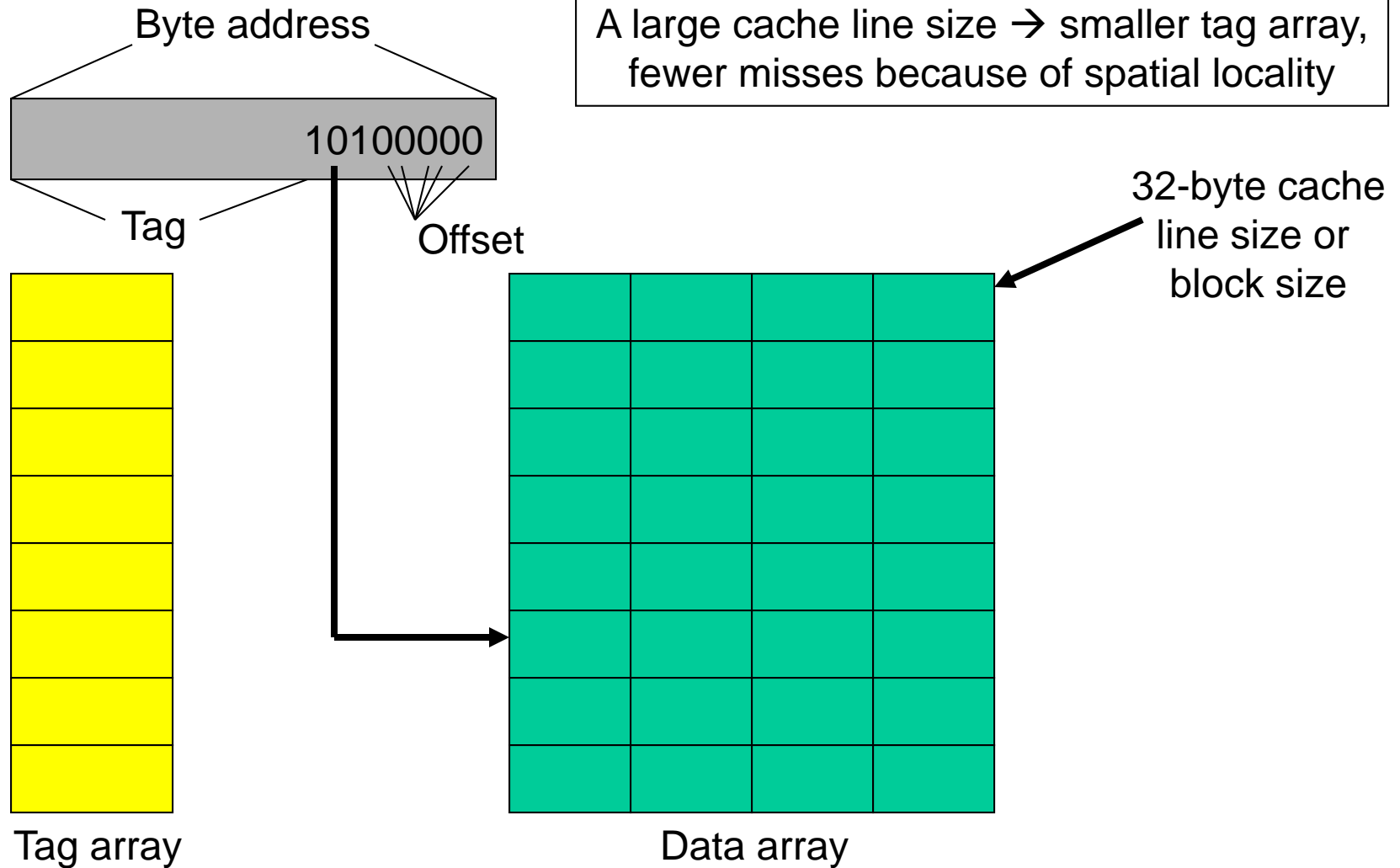


8-byte words

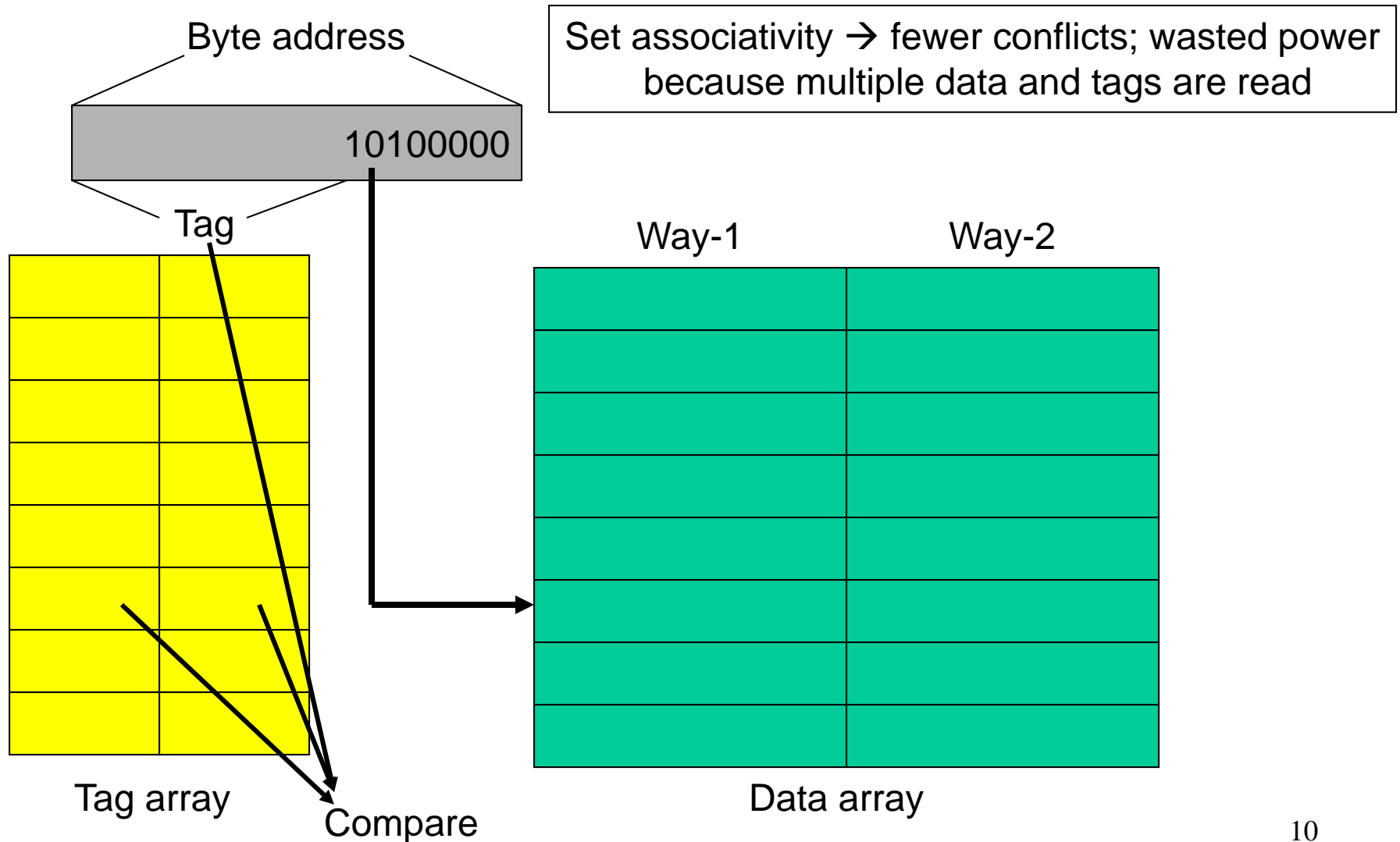
Assume that addresses are 8 bits long
How many of the following address requests
are hits/misses?
4, 7, 10, 13, 16, 68, 73, 78, 83, 88, 4, 7, 10...

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

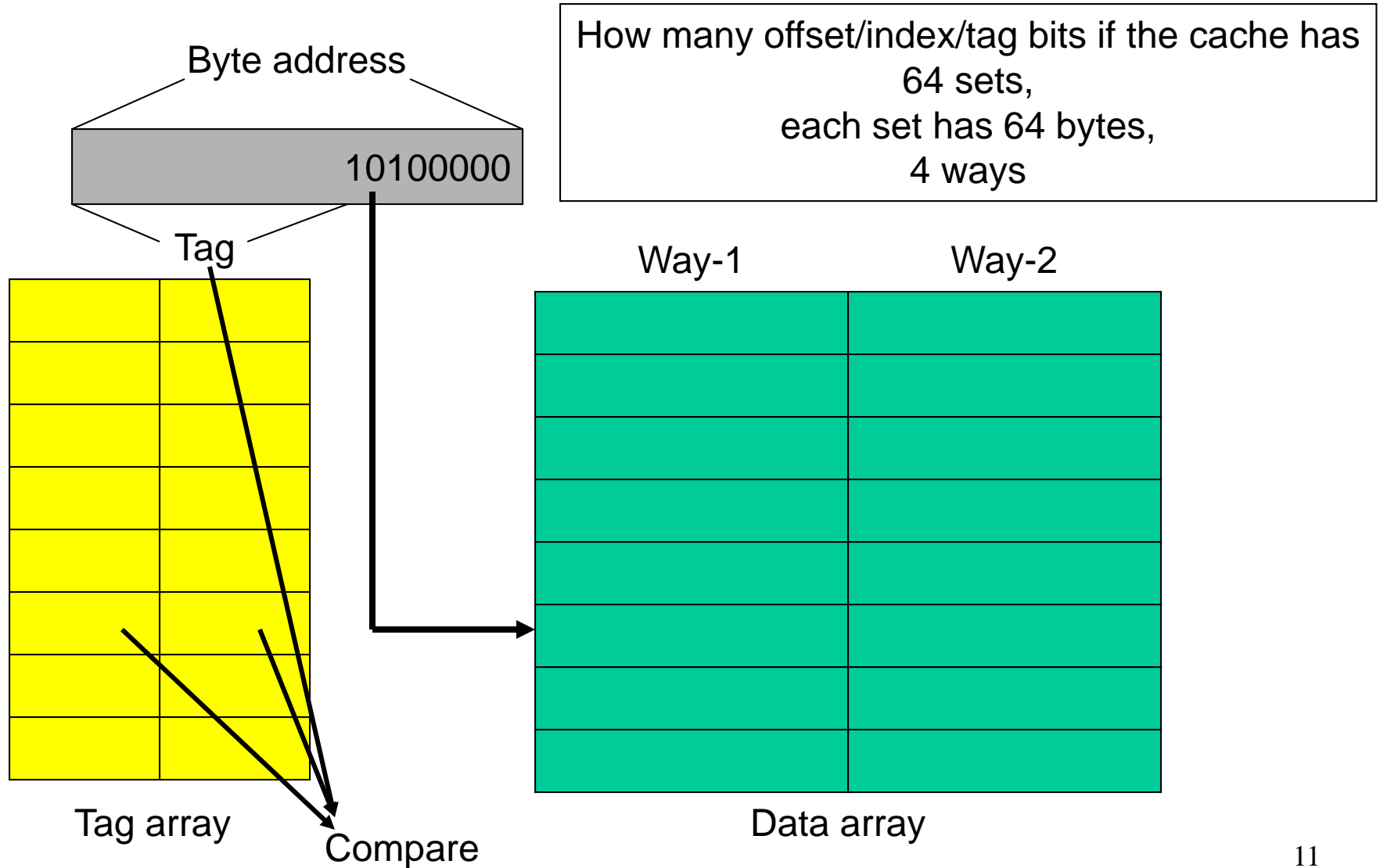
Increasing Line Size



Associativity



Associativity



Example 1

- 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 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
- 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 (l/s) 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

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

Title

- Bullet