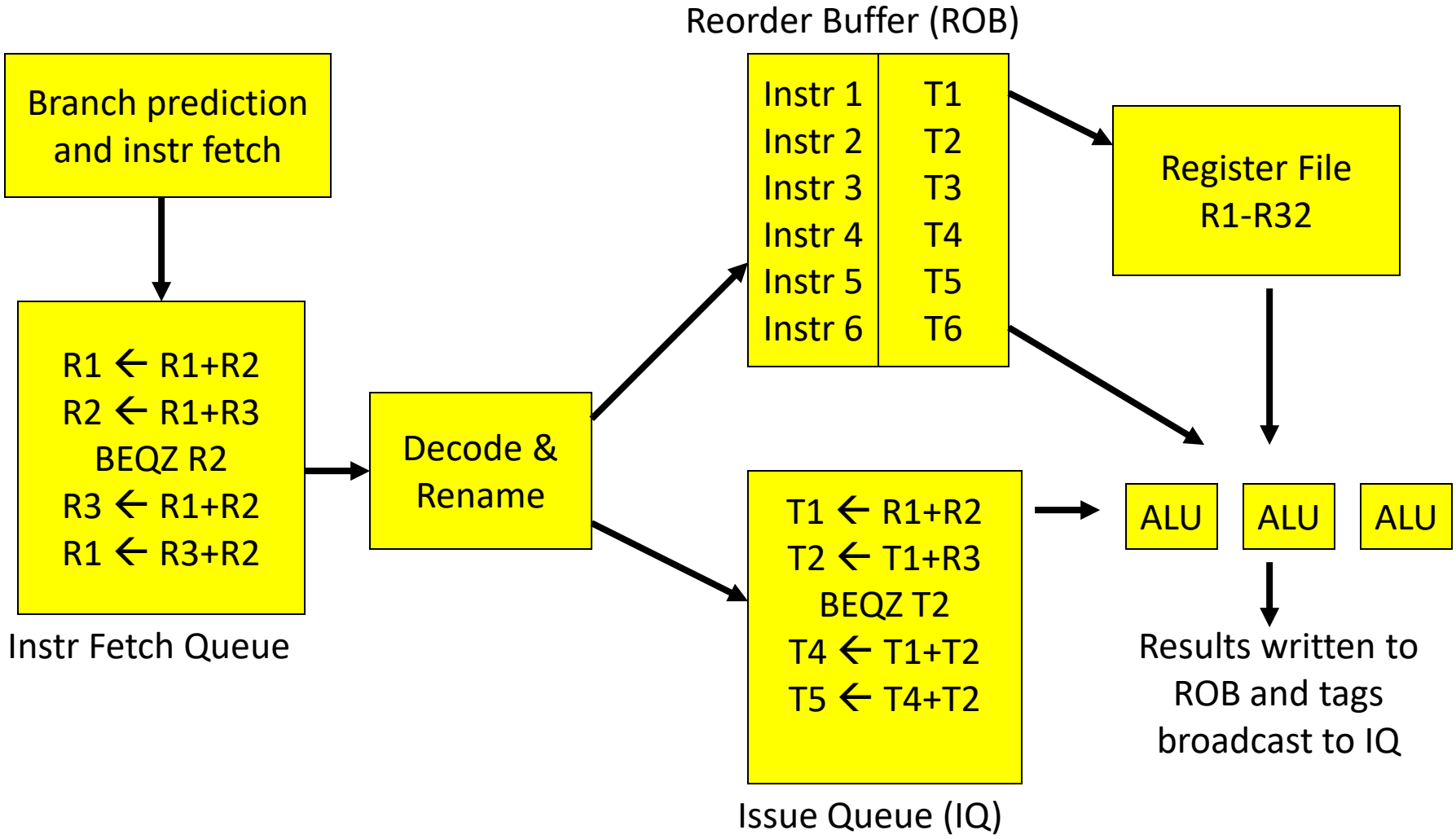
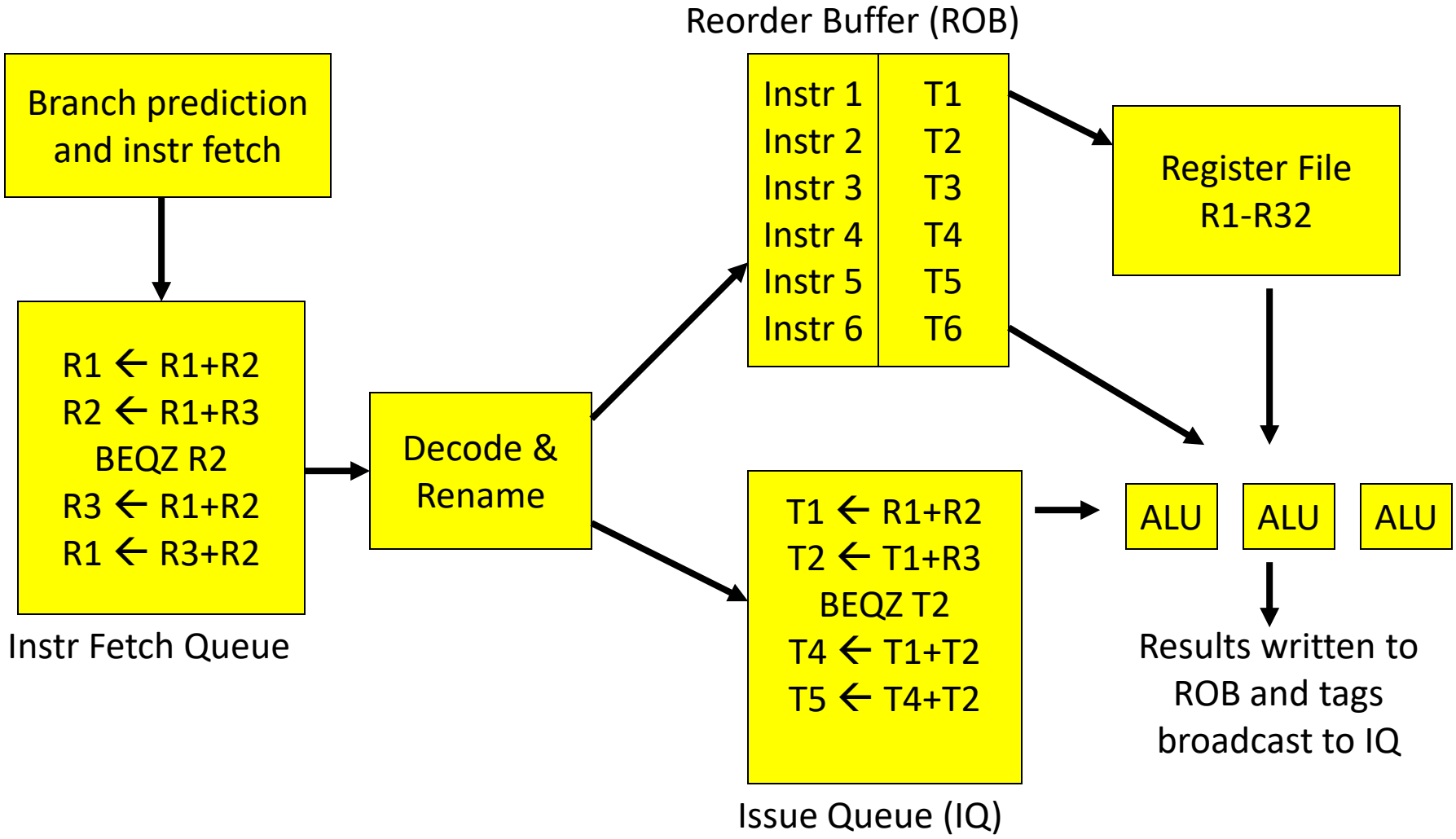
Lecture 22: Out-of-Order, Cache Hierarchies

- Today's topics:
  - Out of order processors
  - Cache access intro and details

An Out-of-Order Processor Implementation



An Out-of-Order Processor Implementation



Example Code

Completion times	with in-order	with ooo
ADD R1, R2, R3	5	5
ADD R4, R1, R2	6	6
LW R5, 8(R4)	7	7
ADD R7, R6, R5	9	9
ADD R8, R7, R5	10	10
LW R9, 16(R4)	11	7
ADD R10, R6, R9	13	9
ADD R11, R10, R9	14	10

Cache Hierarchies

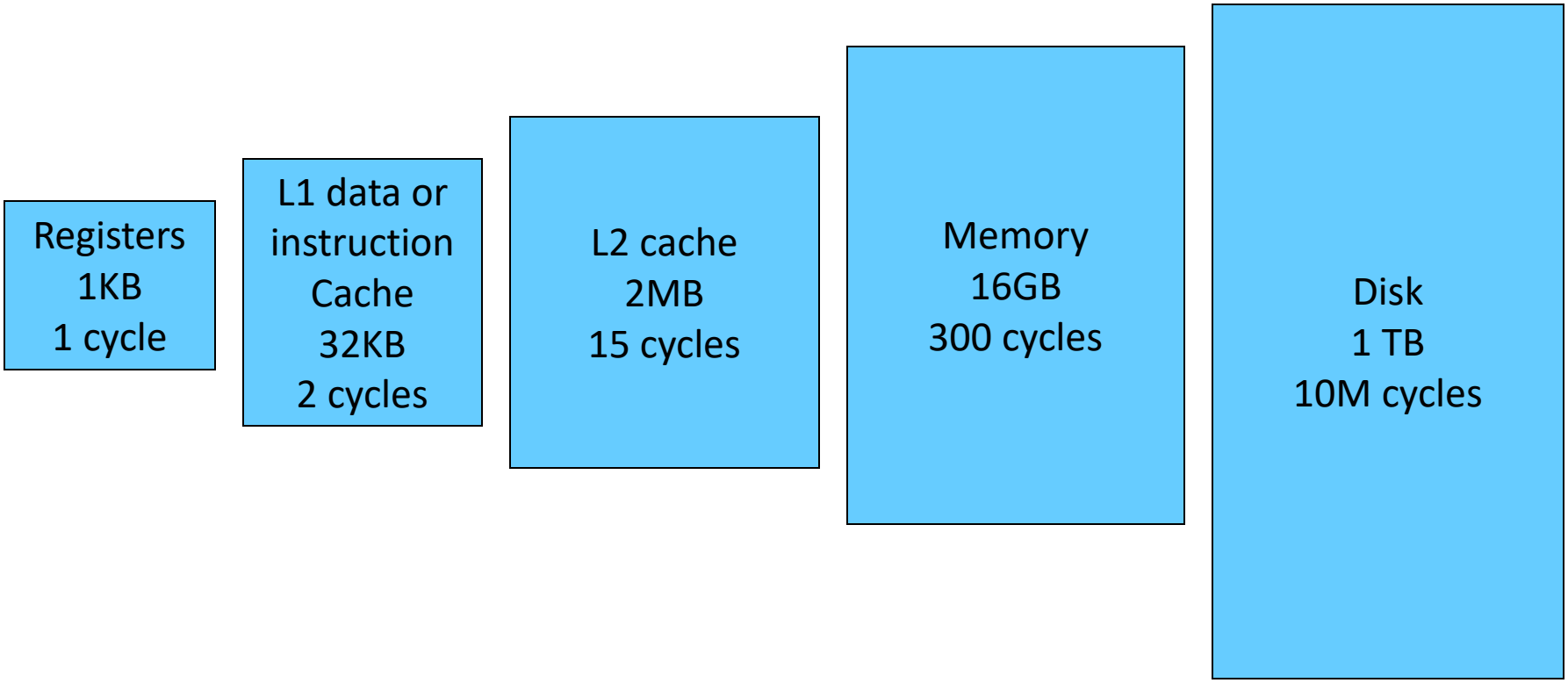
- Data and instructions are stored on DRAM chips – DRAM is a technology that has high bit density, but relatively poor latency – an access to data in memory can take as many as 300 cycles today!

- 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

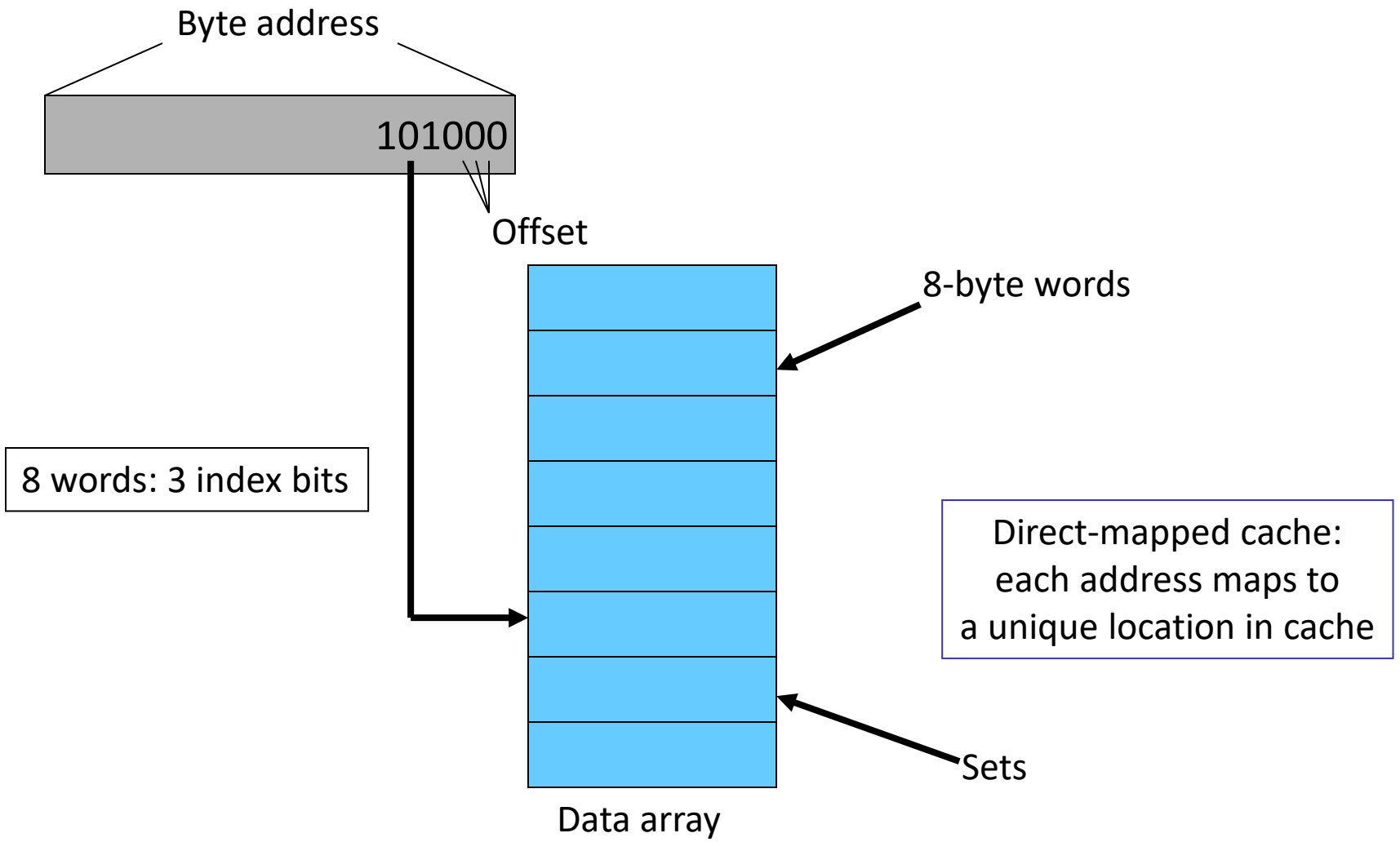
- 32KB 1-cycle L1 cache that has a hit rate of 95%:

  average access time = 0.95 x 1 + 0.05 x (301)

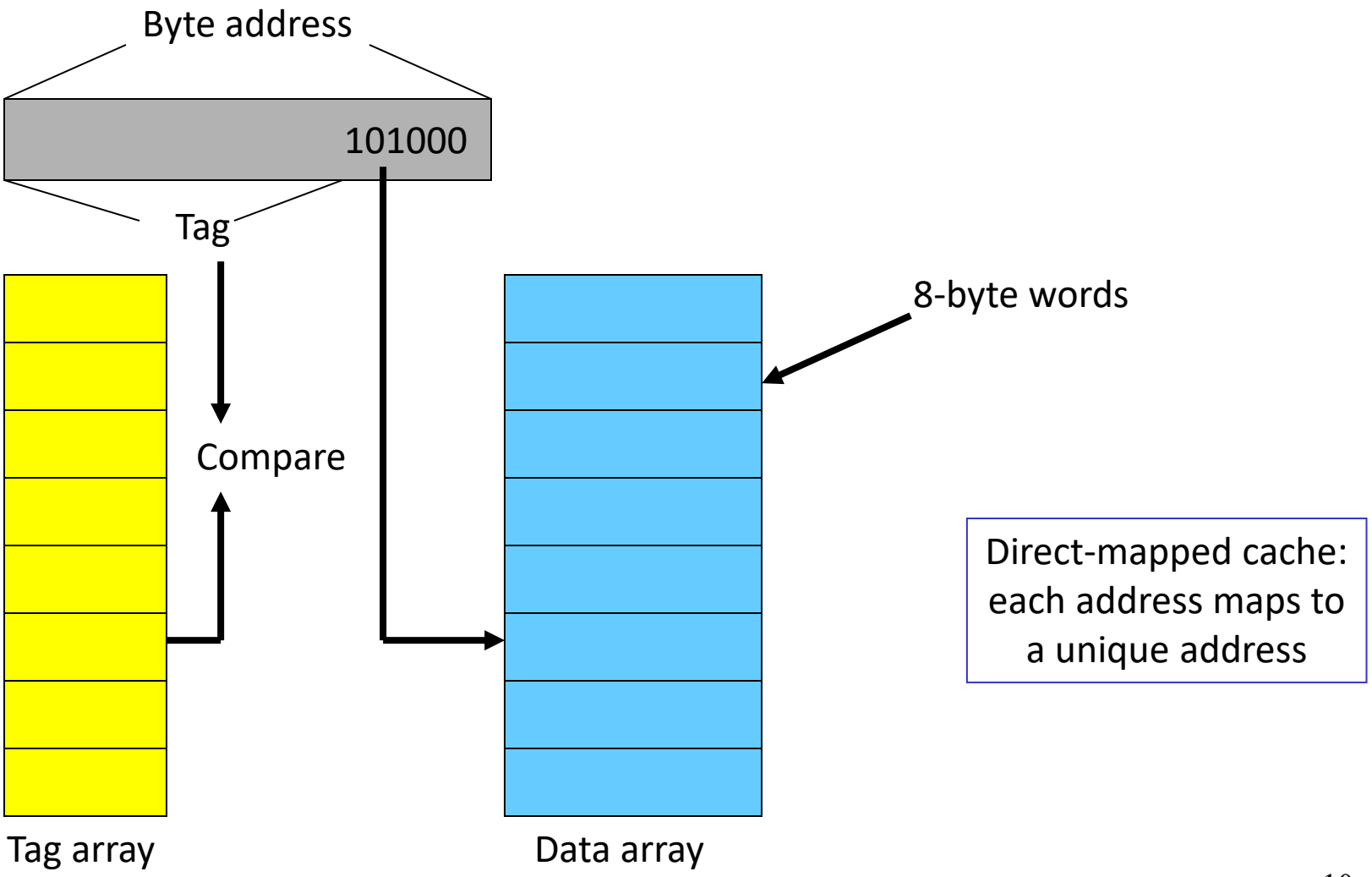
  = 16 cycles

Accessing the Cache

Accessing the Cache

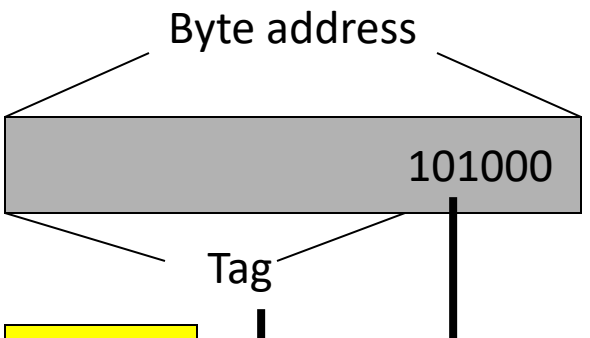


The Tag Array

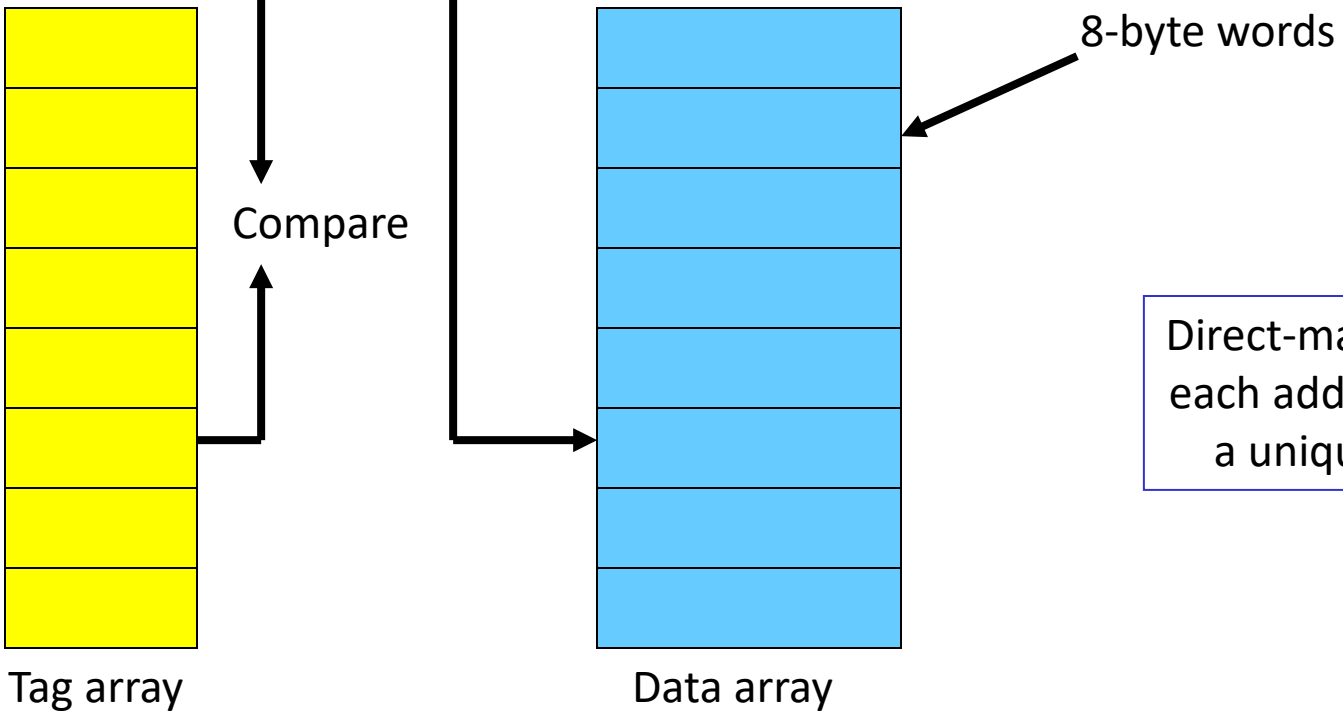


Direct-mapped cache: each address maps to a unique address

Example Access Pattern

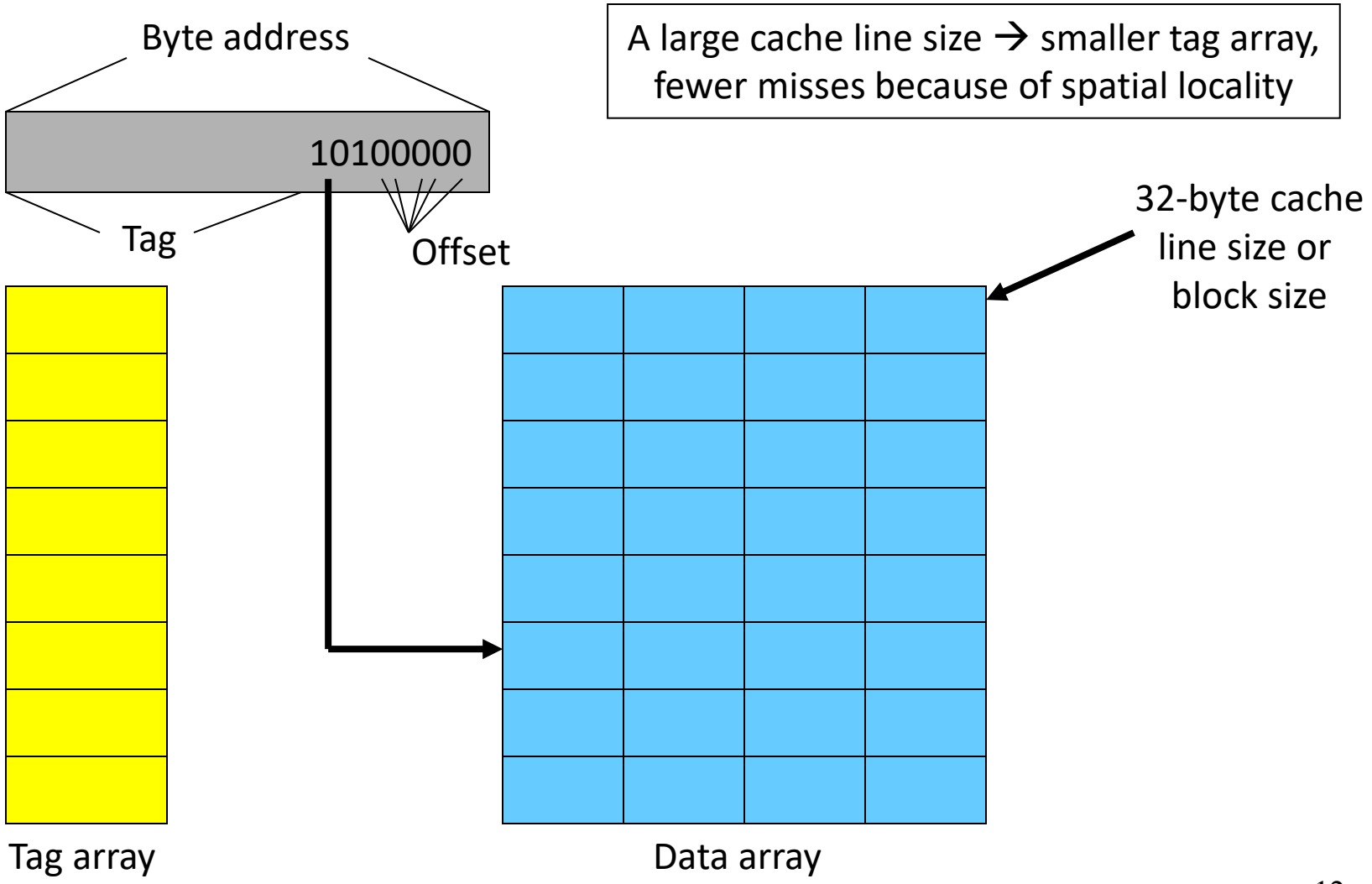


Assume that addresses are 8 bits long
How many of the following address requests are hits/misses?
4, 7, 10, 13, 16, 68, 73, 78, 83, 88, 4, 7, 10…

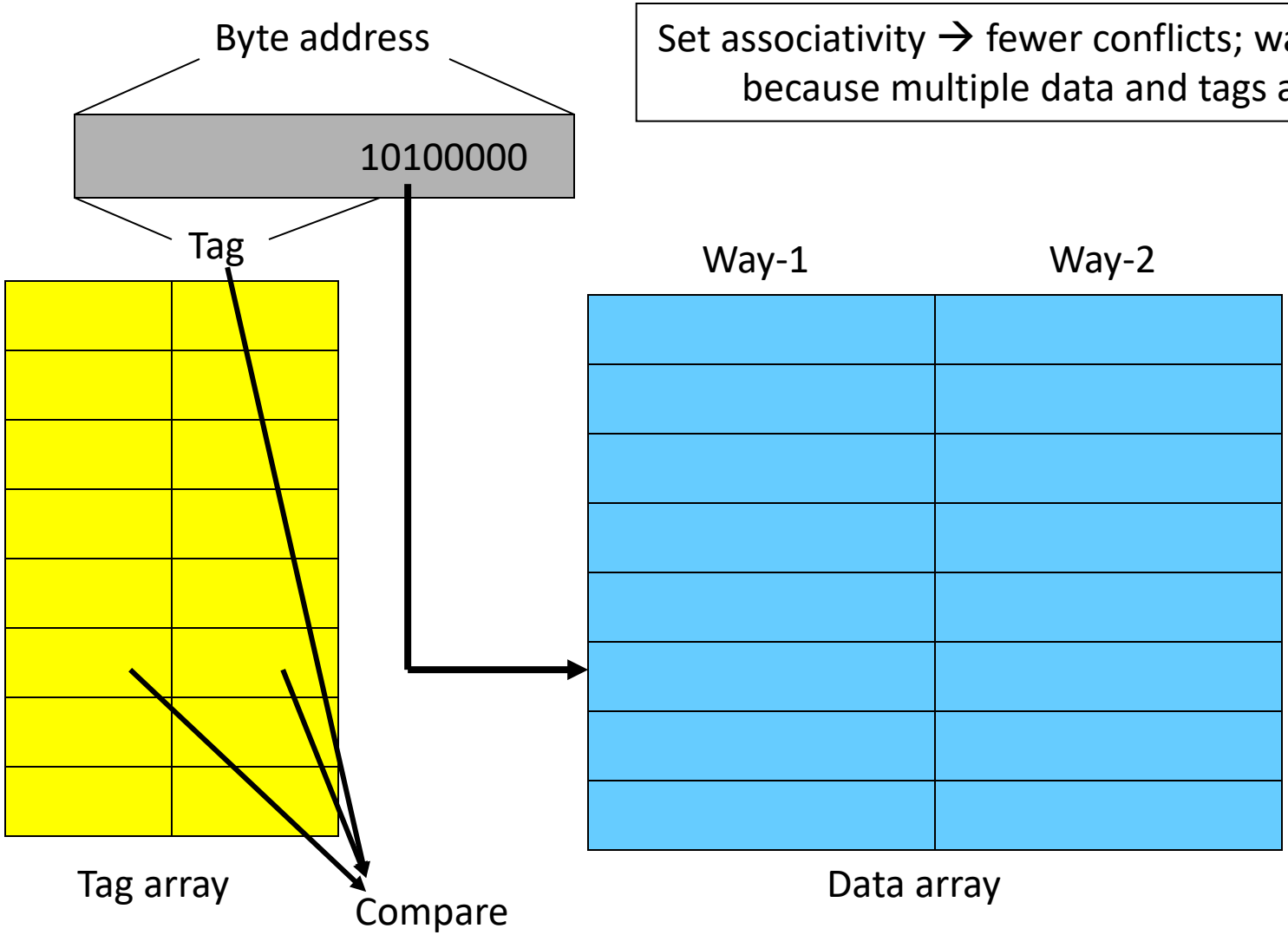


Direct-mapped cache: each address maps to a unique address

Increasing Line Size



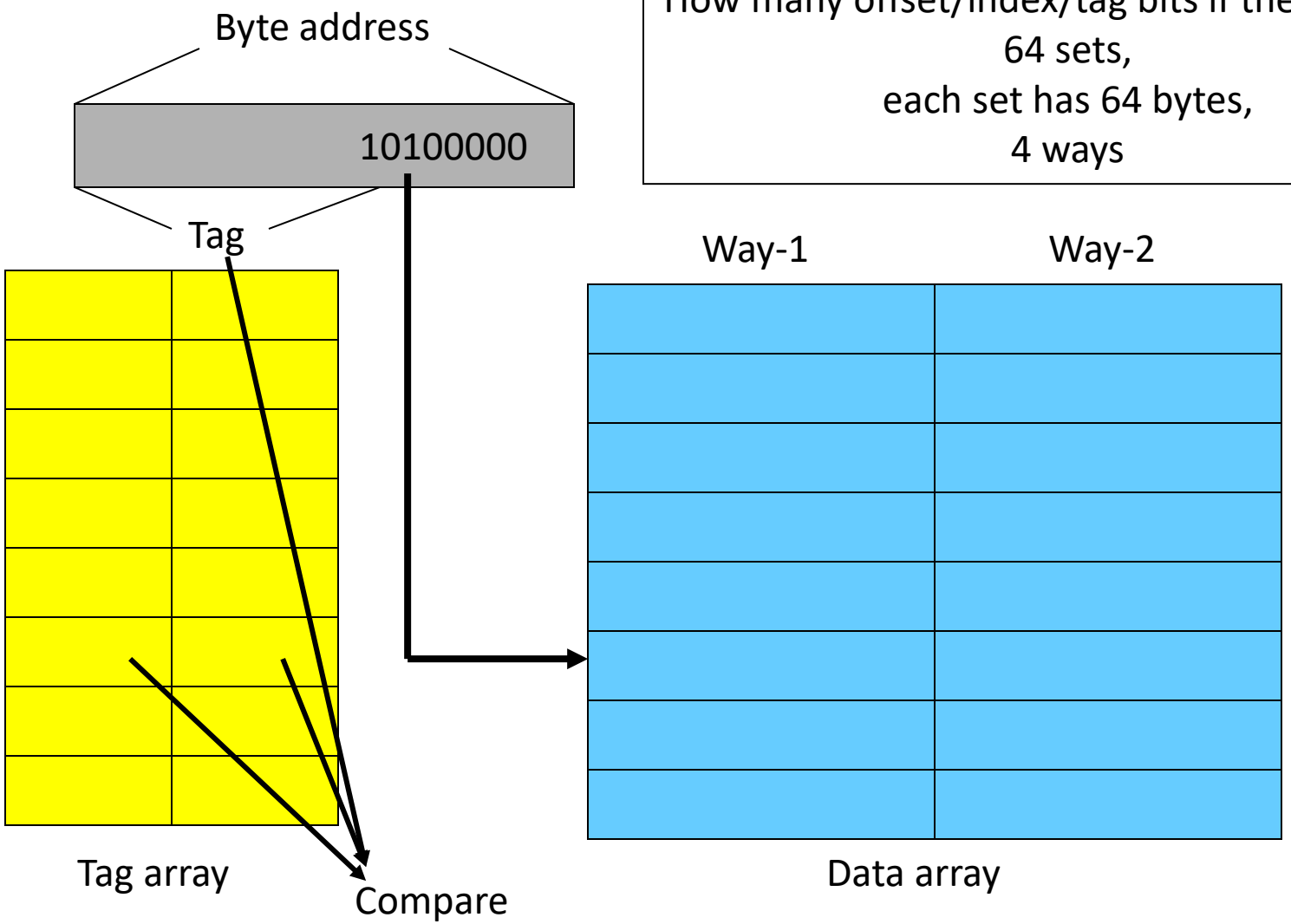
Associativity



Set associativity → fewer conflicts; wasted power because multiple data and tags are read

Associativity

How many offset/index/tag bits if the cache has
64 sets,
each set has 64 bytes,
4 ways



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?

Cache size = #sets x #ways x blocksize
Index bits = $\log_2$(sets)
Offset bits = $\log_2$(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 (log$_2$(sets)) 5 (log$_2$(blksize)) 19 (addrsize-index-offset)

- How large is the tag array?

  tag array size = #sets x #ways x tag size

  = 19 Kb = 2.375 KB