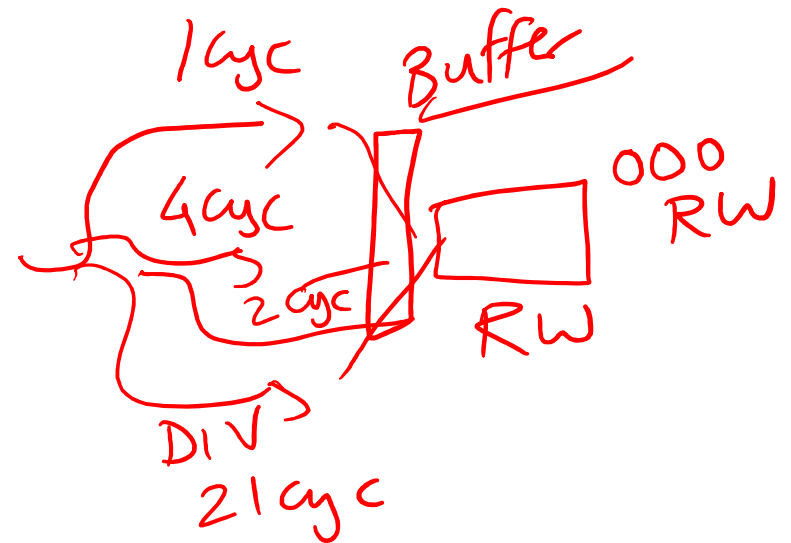


Lecture 22: Out-of-Order, Cache Hierarchies

- Today's topics: HW 8 due tomorrow.
HW 9 posted Thursday, due 4/11

- Out of order processors
- Cache access intro and details

in-order
5-stage



R1 ←

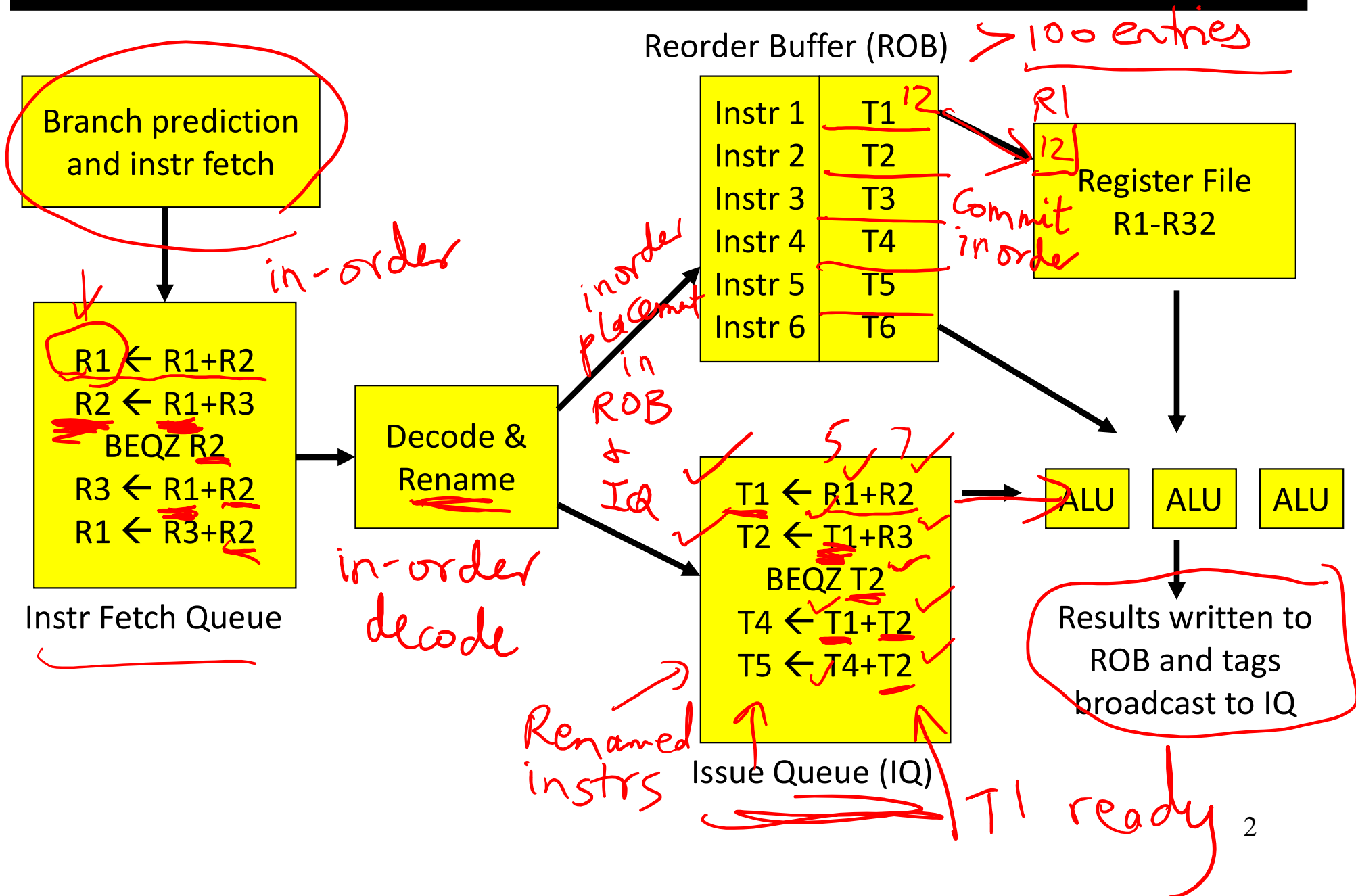
R2 ←

R3 ←

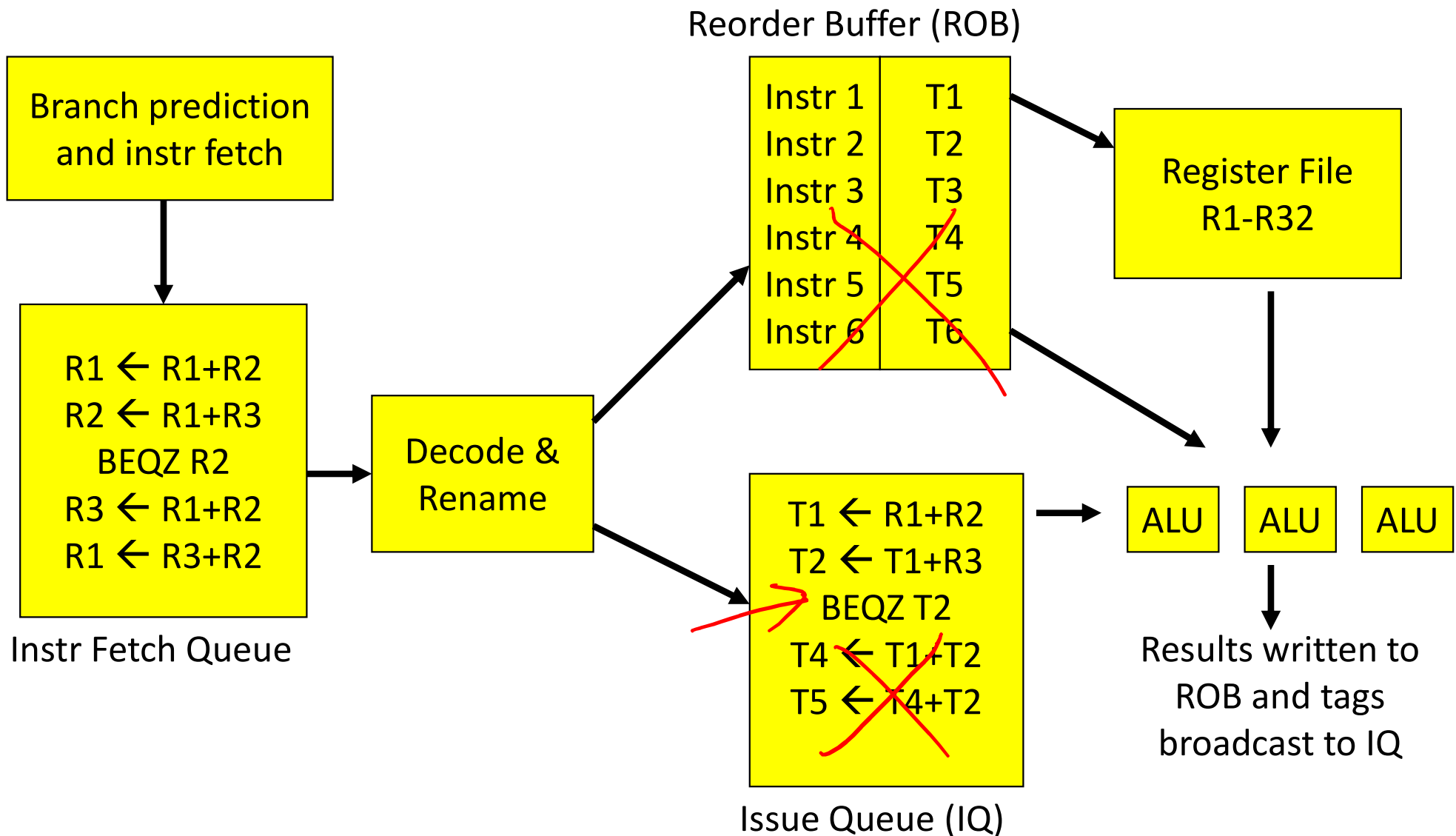
R1 ←

1CP instr level parallelism Superscalar exec = width > 1

An Out-of-Order Processor Implementation



An Out-of-Order Processor Implementation



Example Code

Br pred acc = 90%

0.9 ⁶ BEQ
6 ~~BEQ~~

ILP

+

ooo

Speculation

0.9 x 0.9

0.9 x 0.9 x 0.9

5-stage ⁶

Completion times

with in-order

with ooo

1 ns ³⁰

0.9⁵

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

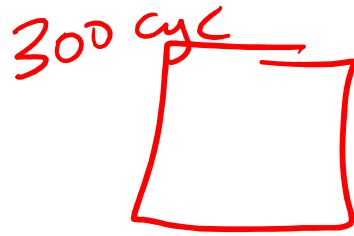
after R4 is produced

after R4 is produced

$$IPC = \frac{8 \text{ instrs}}{10 \text{ cyc}} = 0.8$$

$$IPC = \frac{8 \text{ instrs}}{6 \text{ cyc}} = 1.33$$

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

64KB

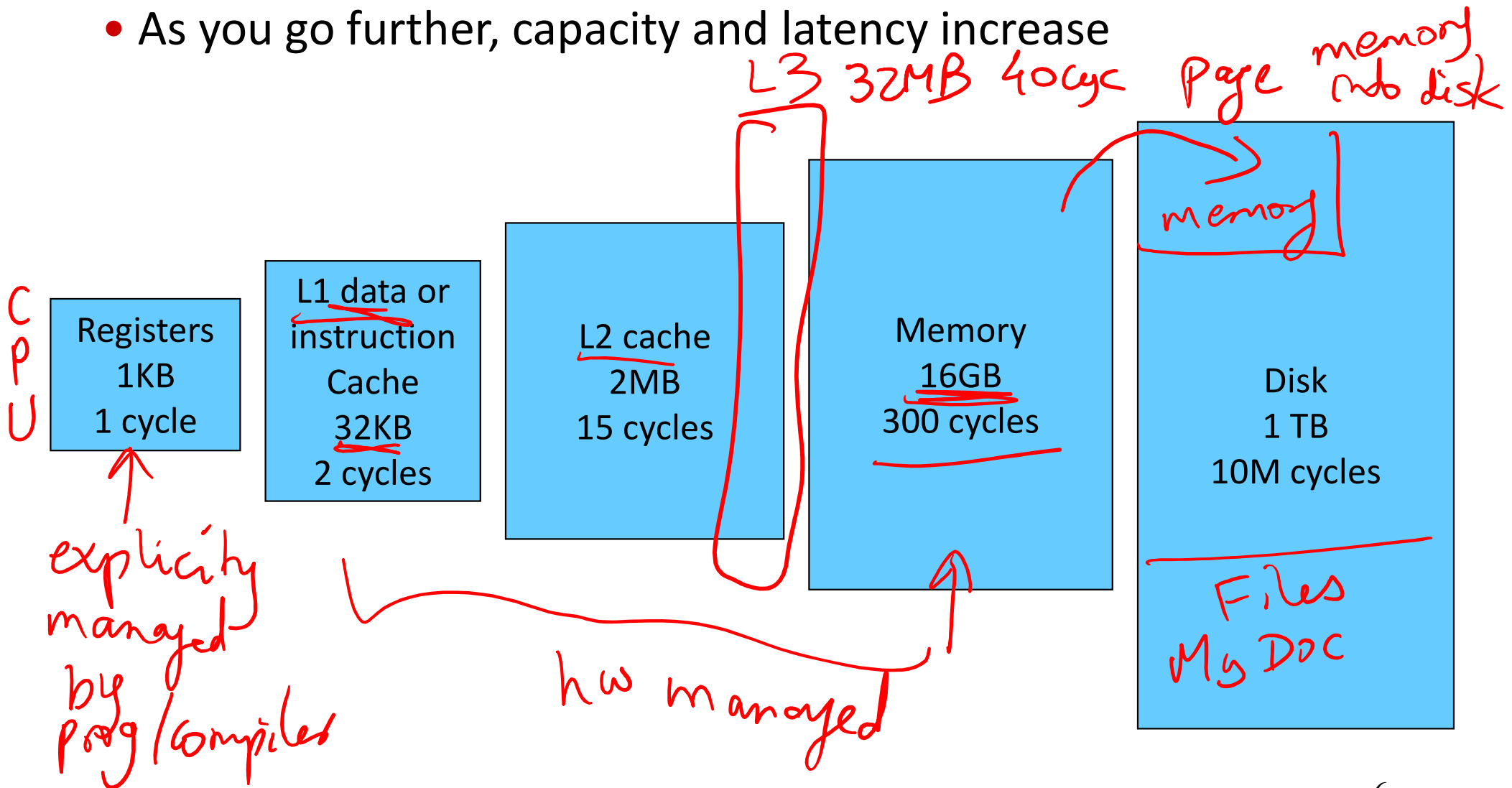
hit rate

90%

locality

Memory Hierarchy

- As you go further, capacity and latency increase



Locality 1-cycle L1 with 90% hit rate
10-cycle L2 with 50% hit rate

100 acc 5 90 L1 hits 90 cyc
5 5 L2 hits 5 x 11 cyc

- 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

90 + 55 + 1555

= 1700 cyc

every 1w

- No hierarchy: average access time for data = 300 cycles / mem access

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

$$\text{average access time} = 0.95 \times 1 + 0.05 \times (301) = 16 \text{ cycles}$$

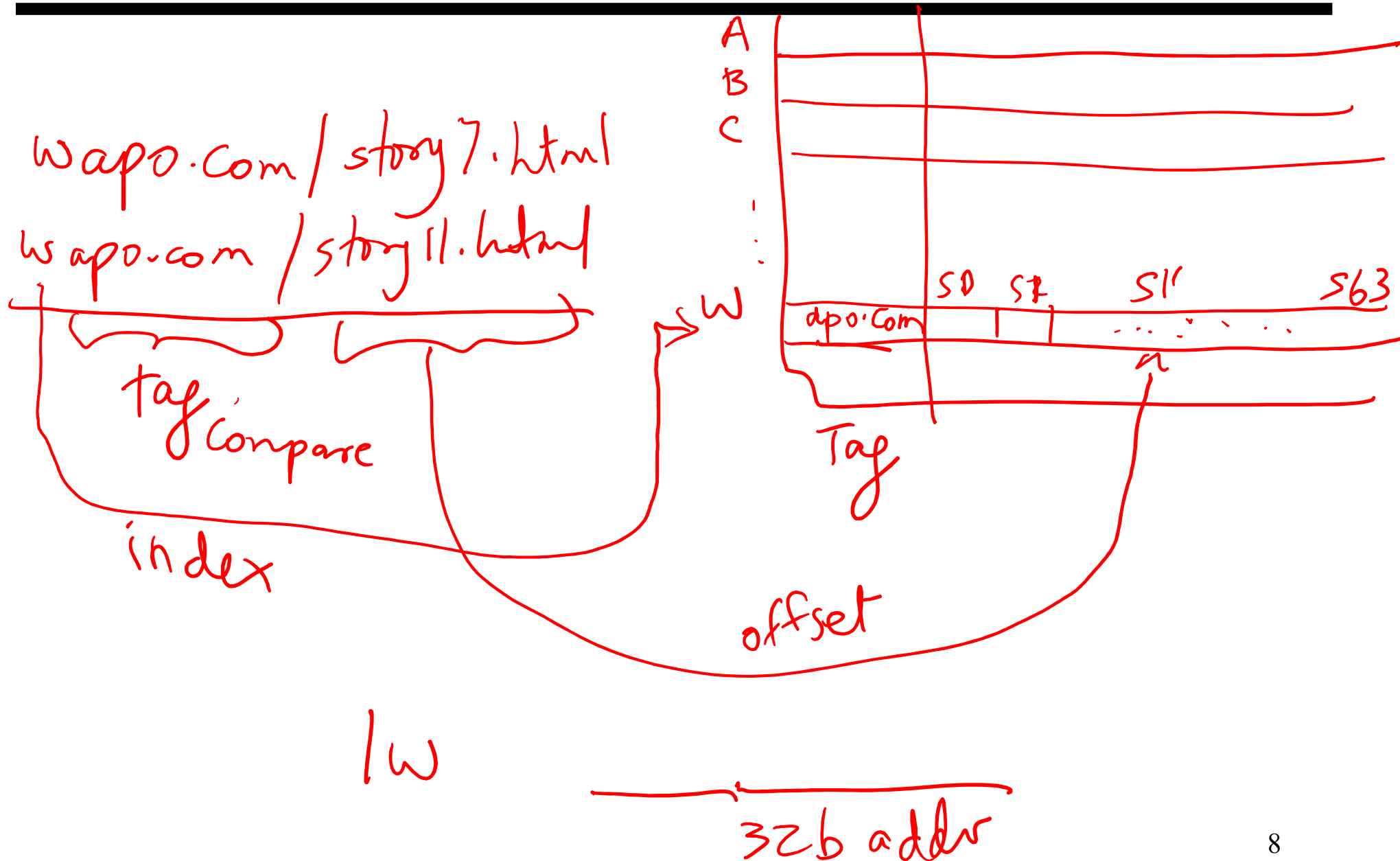
100 mem acc

95 - 1 cycle

5 - 301 cycles each

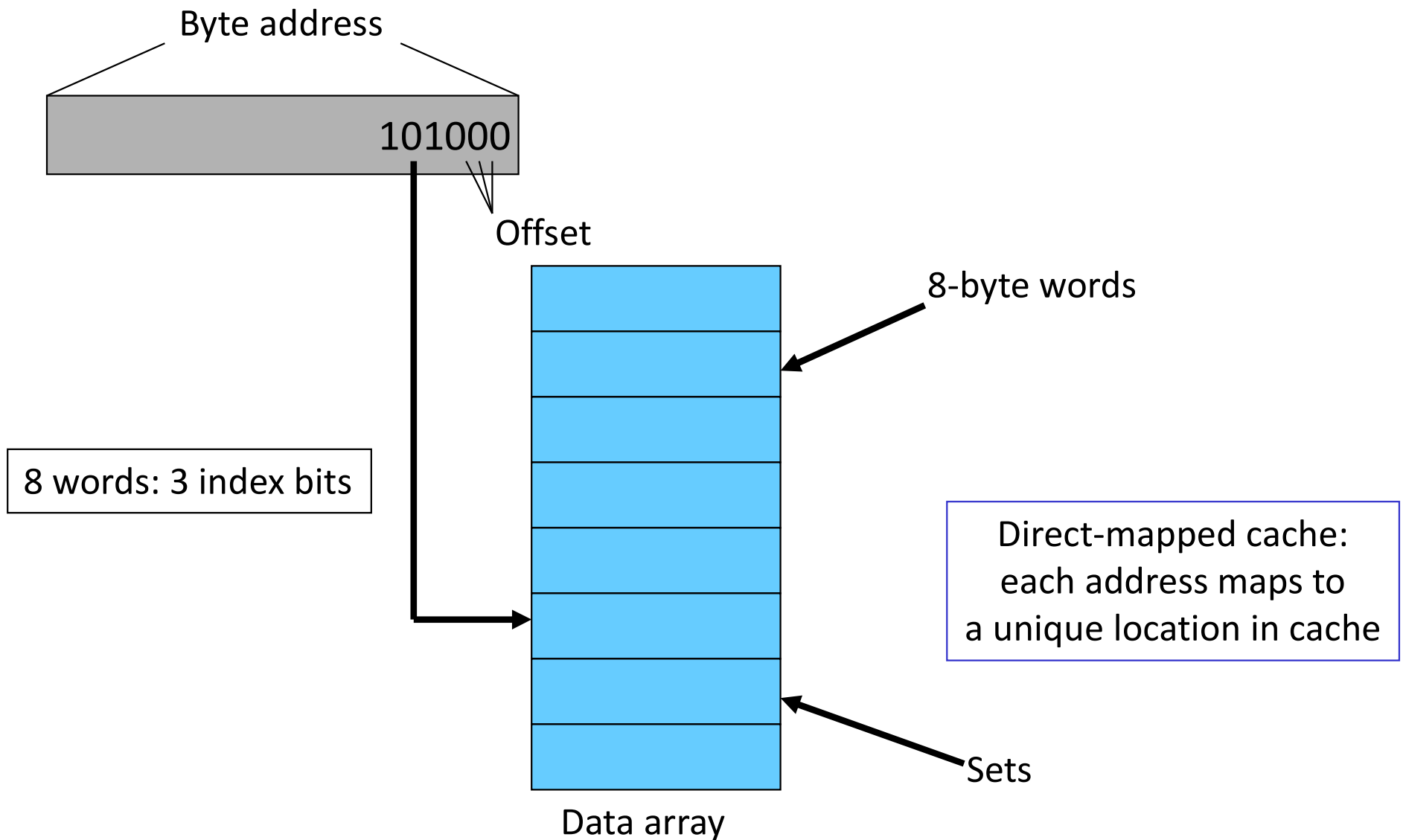
Total time = 95 + 5 x 301 = 1600 cyc
time per mem acc = 16 cyc

Accessing the Cache

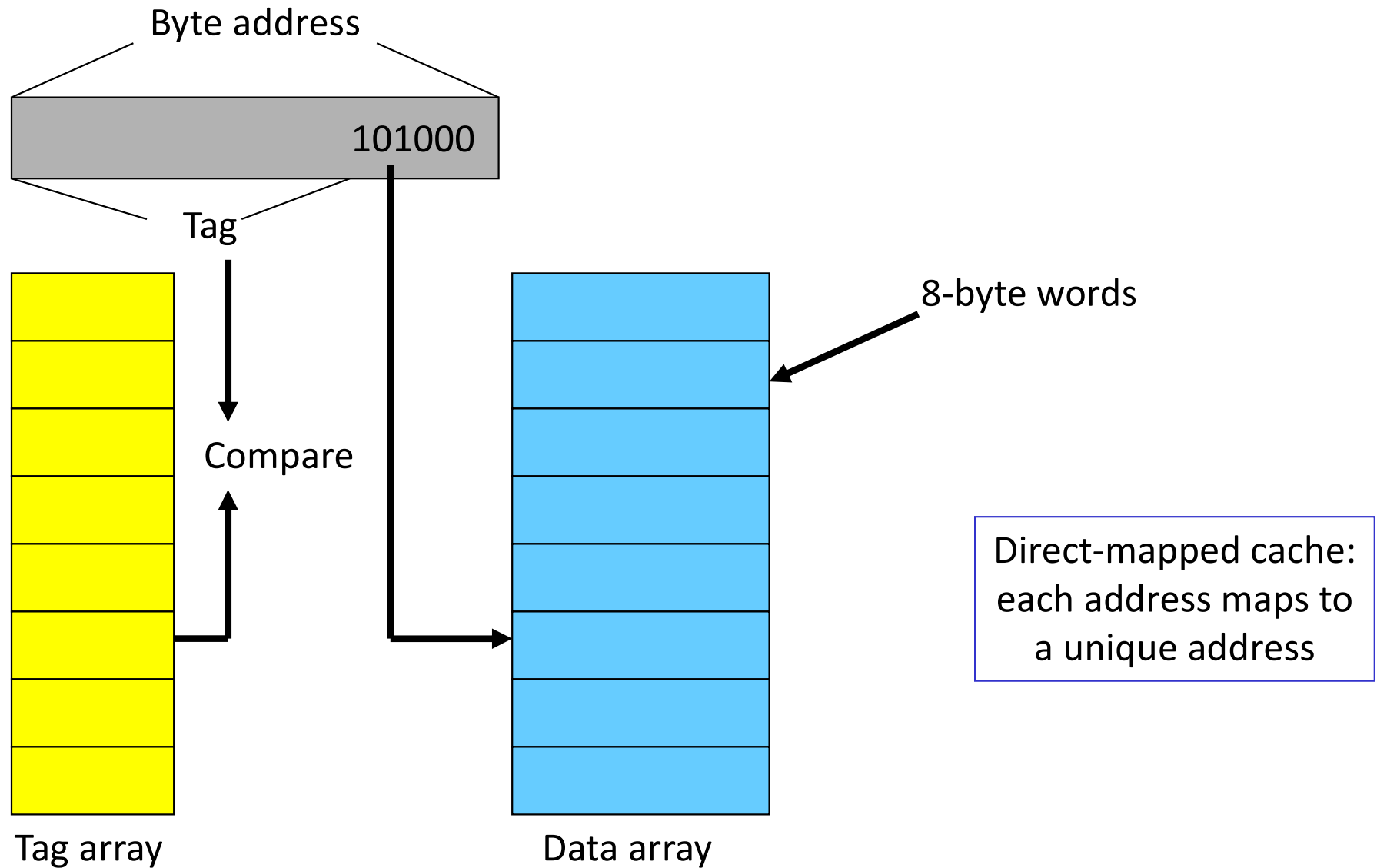


Accessing the Cache

Accessing the Cache



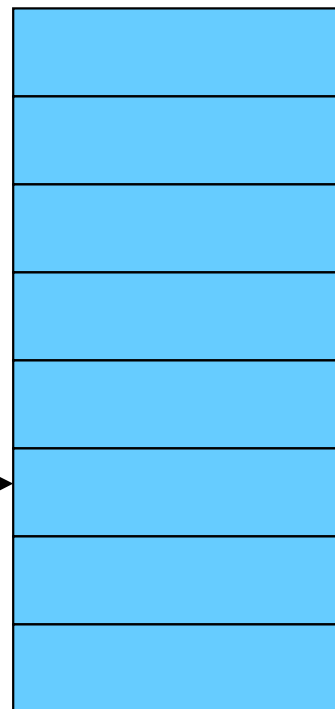
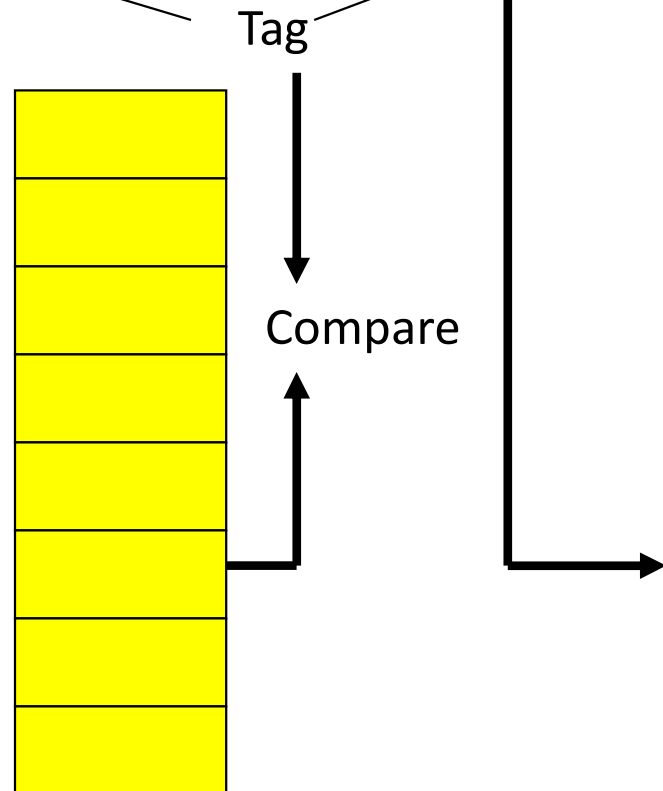
The Tag Array



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...



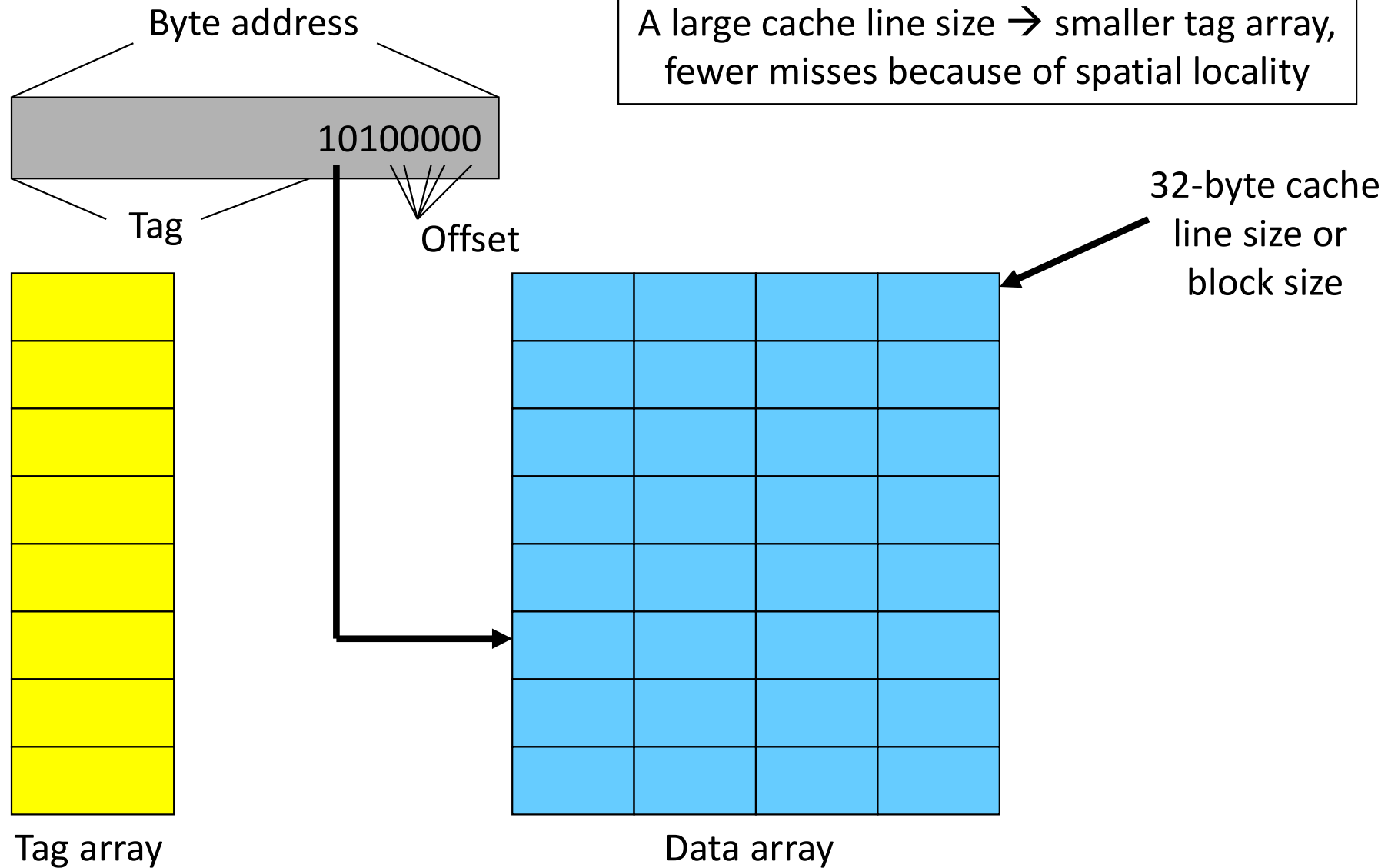
8-byte words

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

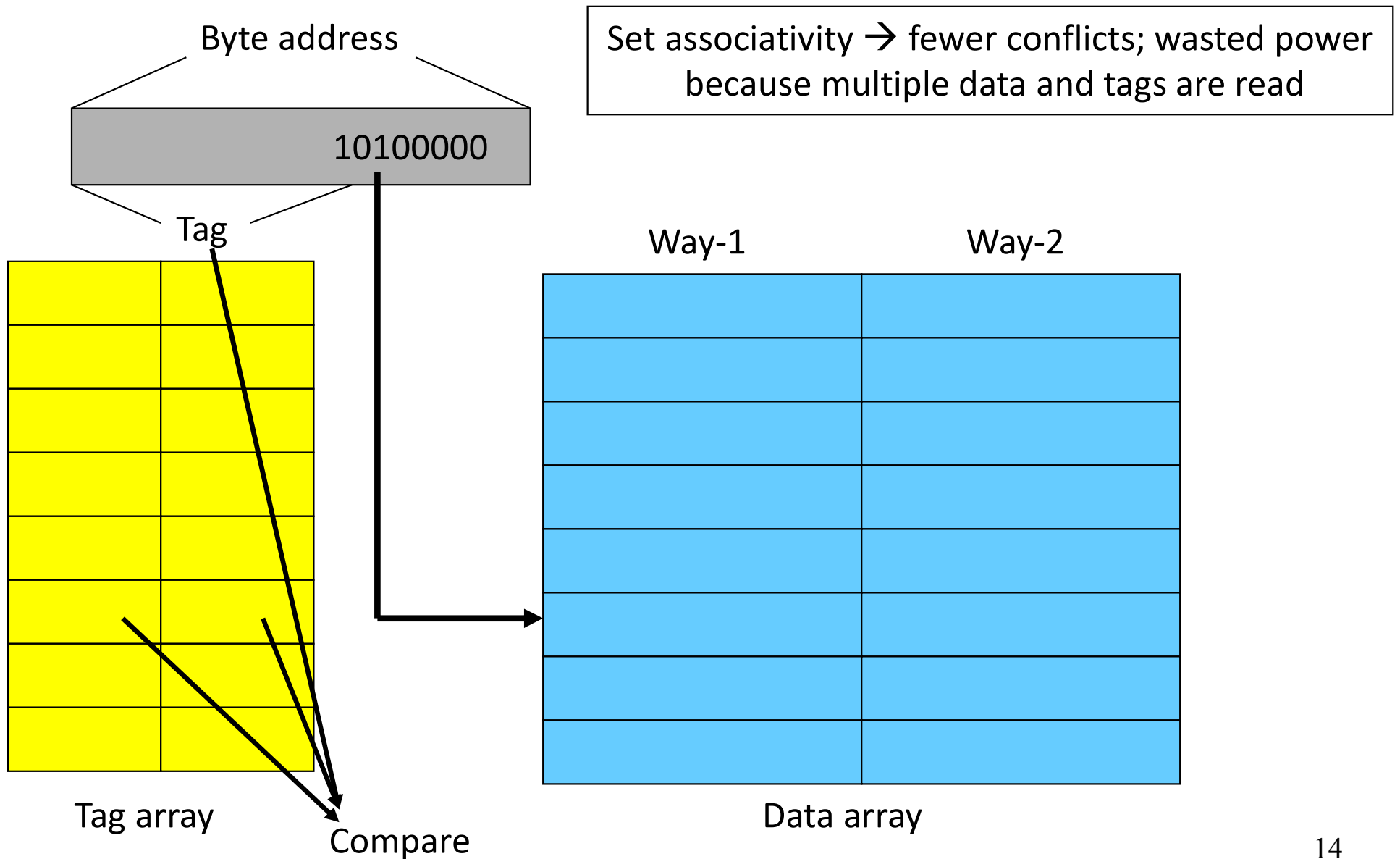
Tag array

Data array

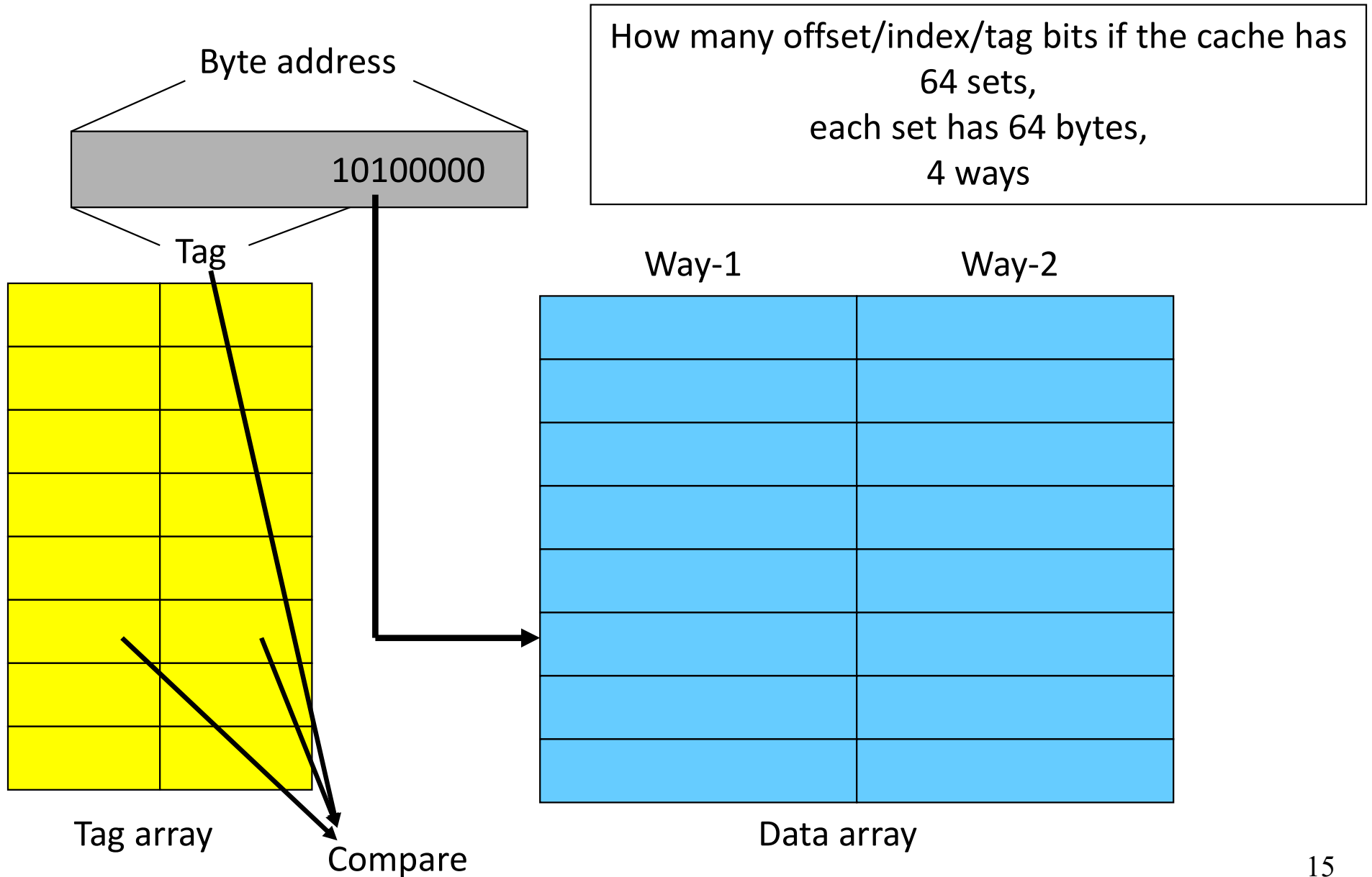
Increasing Line Size



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?

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
$\log_2(\text{sets})$	$\log_2(\text{blksize})$	addrsiz-index-offset

- How large is the tag array?

tag array size = #sets x #ways x tag size
= 19 Kb = 2.375 KB