Lecture: Cache Hierarchies

- Topics: cache access basics/examples

Out-of-Order Loads/Stores

Ld	R1 ← [R2]
Ld	R3 ← [R4]
St	R5 → [R6]
Ld	R7 ← [R8]
Ld	R9←[R10]

What if the issue queue also had load/store instructions?
Can we continue executing instructions out-of-order?

Memory Dependence Checking

Ld	0x abcdef
Ld	
St	
Ld	
Ld	0x abcdef
St	0x abcd00
Ld	0x abc000
Ld	0x abcd00

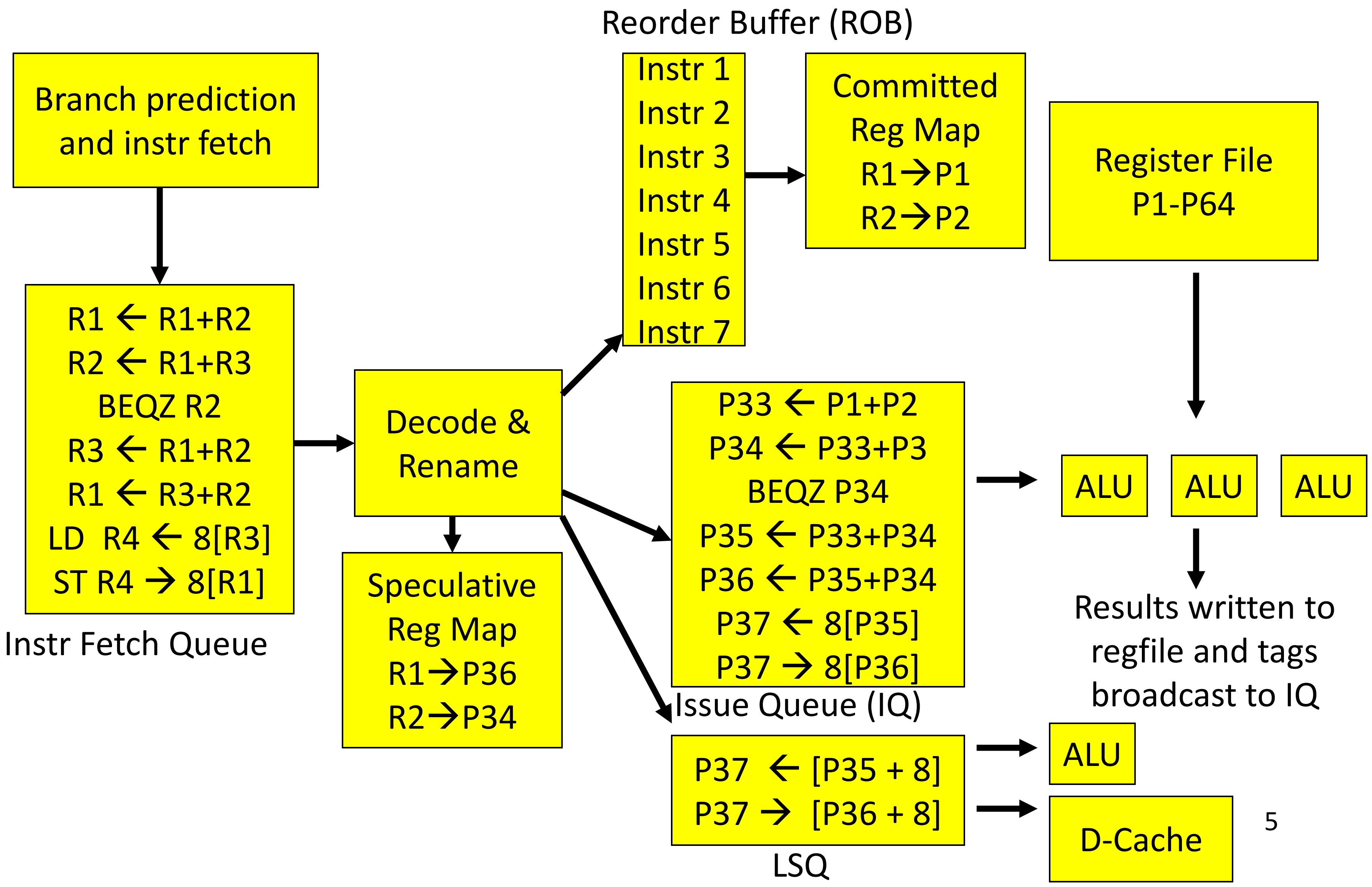
- The issue queue checks for ***register dependences*** and executes instructions as soon as registers are ready
- Loads/stores access memory as well – must check for RAW, WAW, and WAR hazards for memory as well
- Hence, first check for register dependences to compute effective addresses; then check for memory dependences

Memory Dependence Checking

Ld	0x abcdef
Ld	
St	
Ld	
Ld	0x abcdef
St	0x abcd00
Ld	0x abc000
Ld	0x abcd00

- Load and store addresses are maintained in program order in the Load/Store Queue (LSQ)
- Loads can issue if they are guaranteed to not have true dependences with earlier stores
- Stores can issue only if we are ready to modify memory (can not recover if an earlier instr raises an exception) – happens at commit

The Alpha 21264 Out-of-Order Implementation



Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd		
LD R3 ← [R4]	6		adde		
ST R5 → [R6]	4	7	abba		
LD R7 ← [R8]	2		abce		
ST R9 → [R10]	8	3	abba		
LD R11 ← [R12]	1		abba		

Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd	4	5
LD R3 ← [R4]	6		adde	7	8
ST R5 → [R6]	4	7	abba	5	commit
LD R7 ← [R8]	2		abce	3	6
ST R9 → [R10]	8	3	abba	9	commit
LD R11 ← [R12]	1		abba	2	10

Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd		
LD R3 ← [R4]	6		adde		
ST R5 → [R6]	5	7	abba		
LD R7 ← [R8]	2		abce		
ST R9 → [R10]	1	4	abba		
LD R11 ← [R12]	2		abba		

Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd	4	5
LD R3 ← [R4]	6		adde	7	8
ST R5 → [R6]	5	7	abba	6	commit
LD R7 ← [R8]	2		abce	3	7
ST R9 → [R10]	1	4	abba	2	commit
LD R11 ← [R12]	2		abba	3	5

Problem 4

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. **Assume memory dependence prediction.**

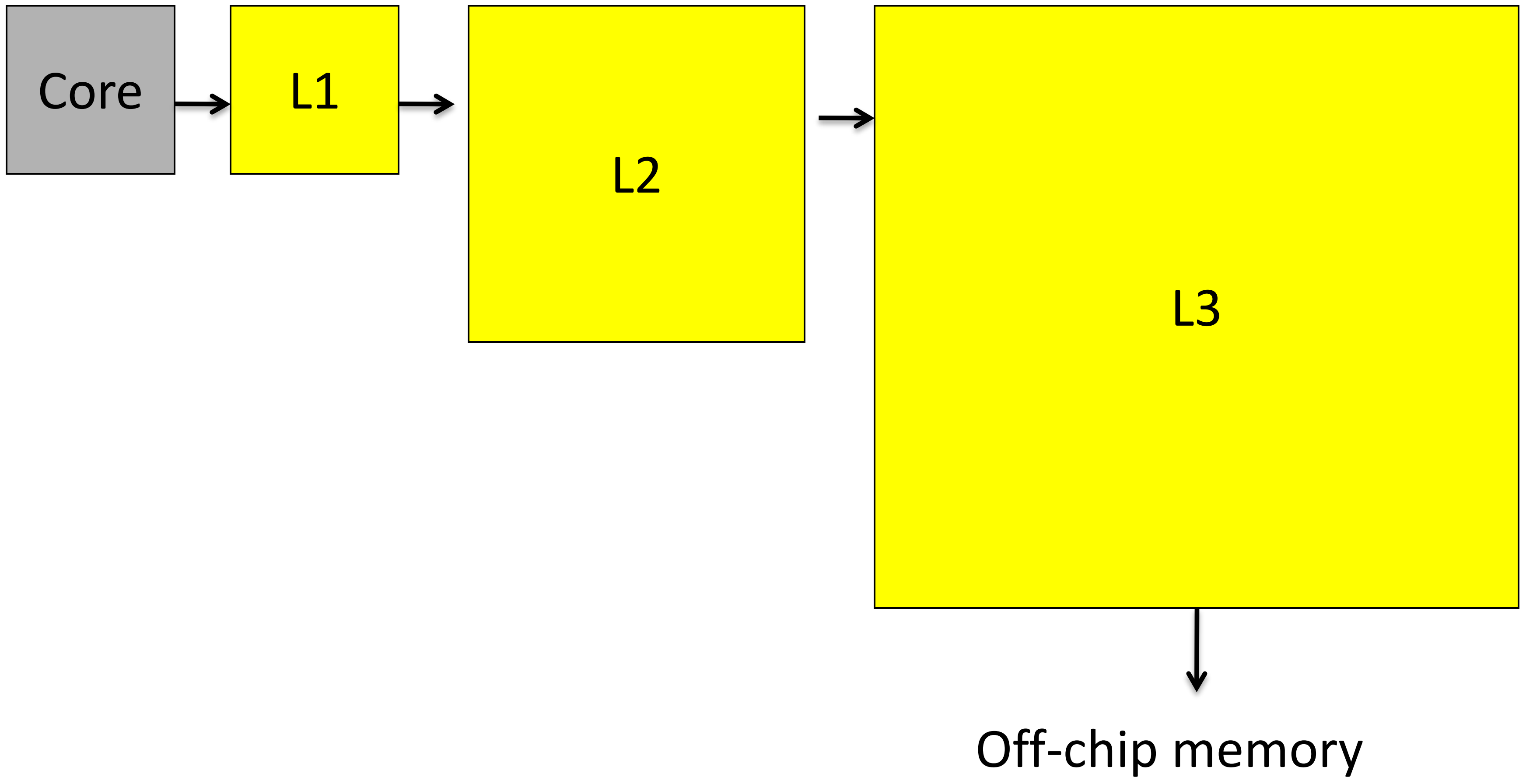
	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd		
LD R3 ← [R4]	6		adde		
ST R5 → [R6]	4	7	abba		
LD R7 ← [R8]	2		abce		
ST R9 → [R10]	8	3	abba		
LD R11 ← [R12]	1		abba		

Problem 4

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. **Assume memory dependence prediction.**

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd	4	5
LD R3 ← [R4]	6		adde	7	8
ST R5 → [R6]	4	7	abba	5	commit
LD R7 ← [R8]	2		abce	3	4
ST R9 → [R10]	8	3	abba	9	commit
LD R11 ← [R12]	1		abba	2	3/10

The Cache Hierarchy



Problem 1

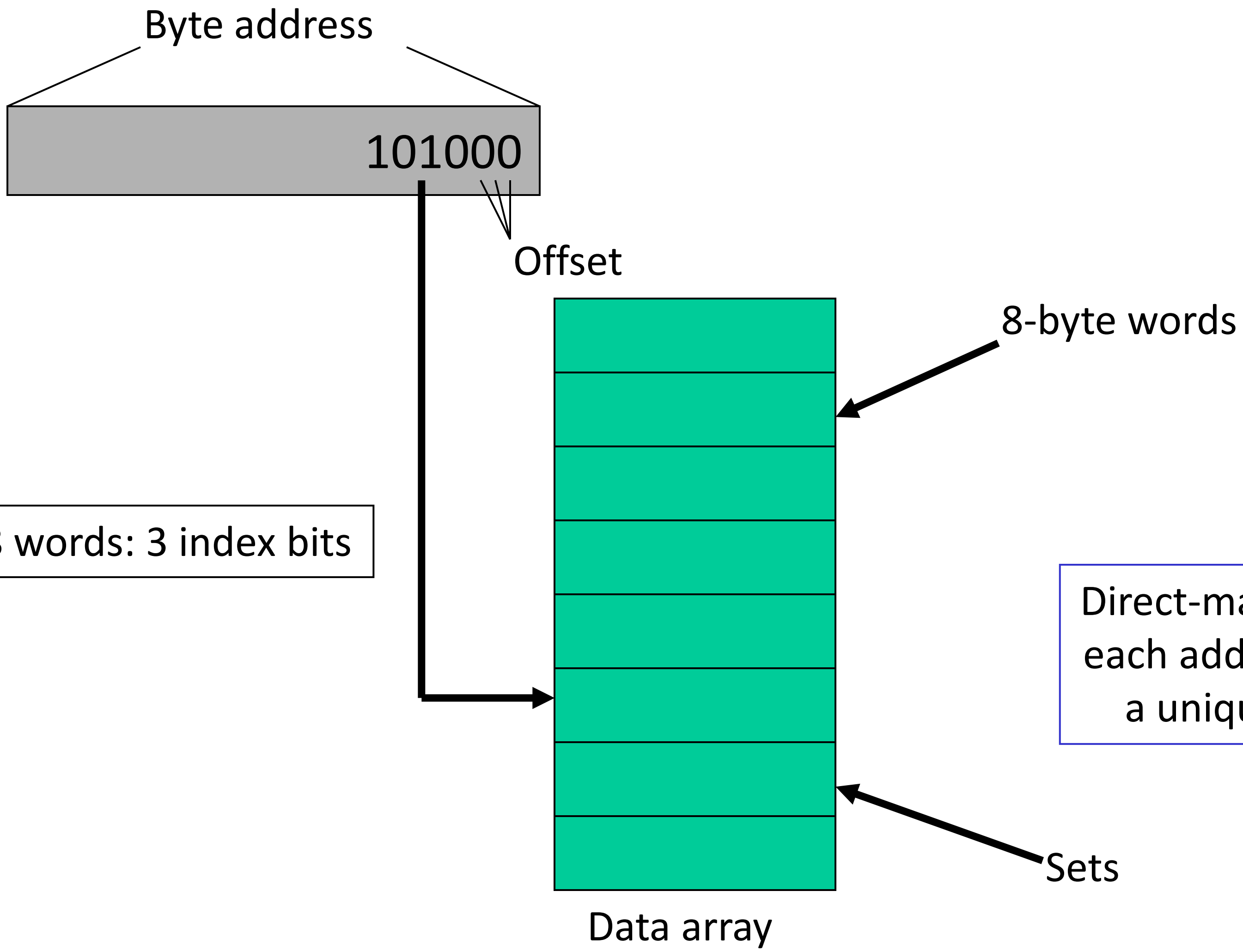
- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

With L3: 1000 + 10x20 + 30x10 + 300x5 = 3000
Without L3: 1000 + 10x20 + 10x300 = 4200

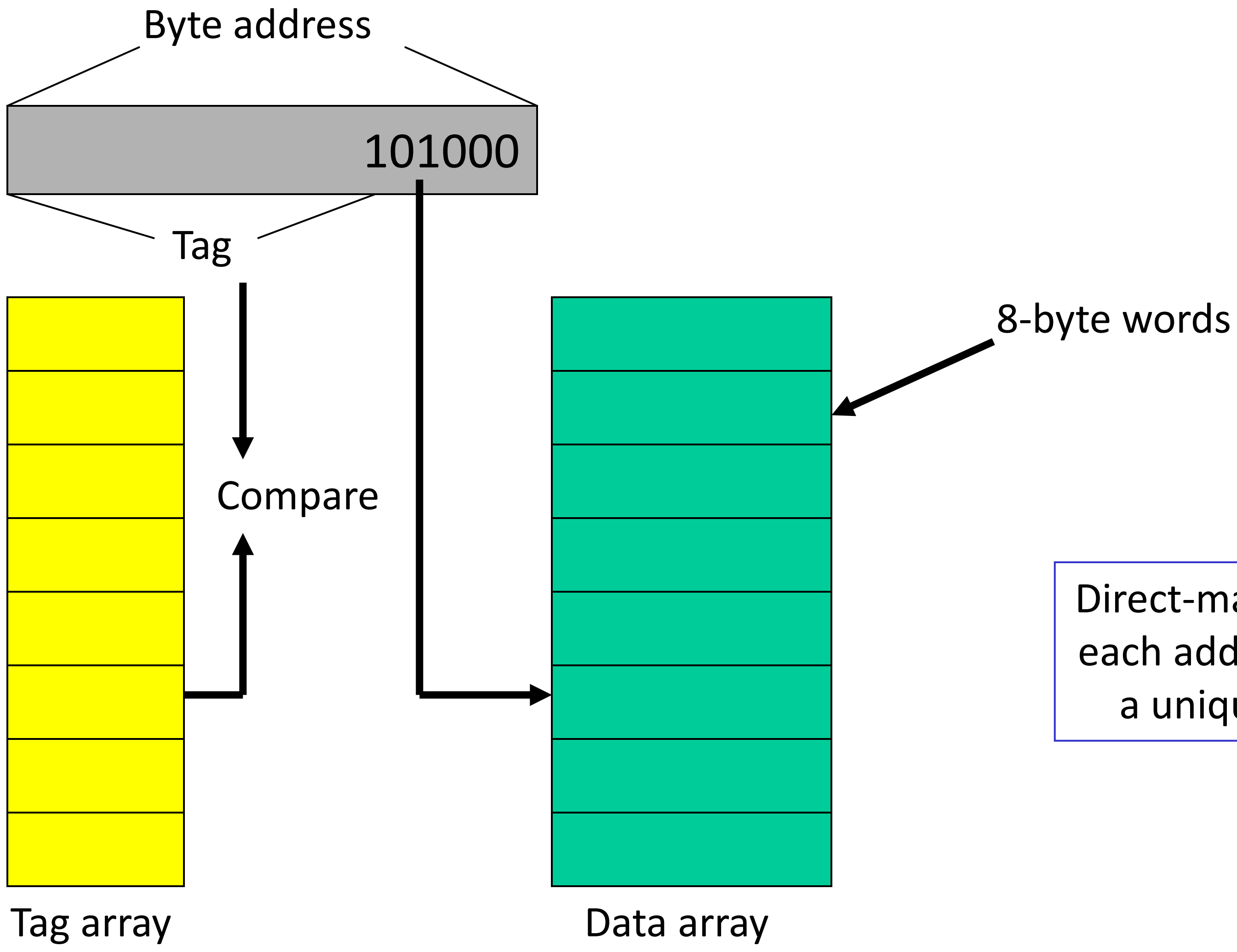
Accessing the Cache



8 words: 3 index bits

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

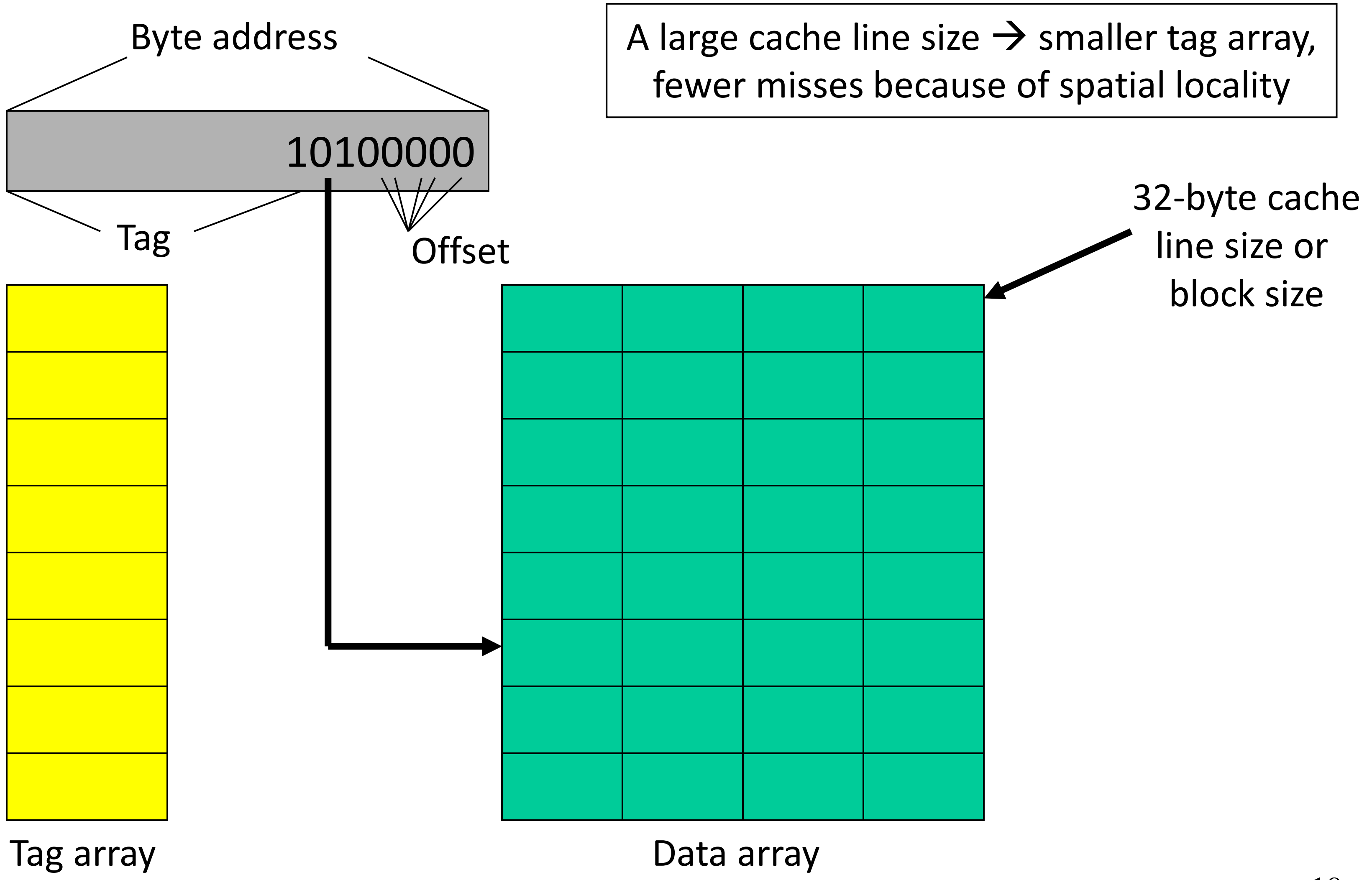
The Tag Array



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

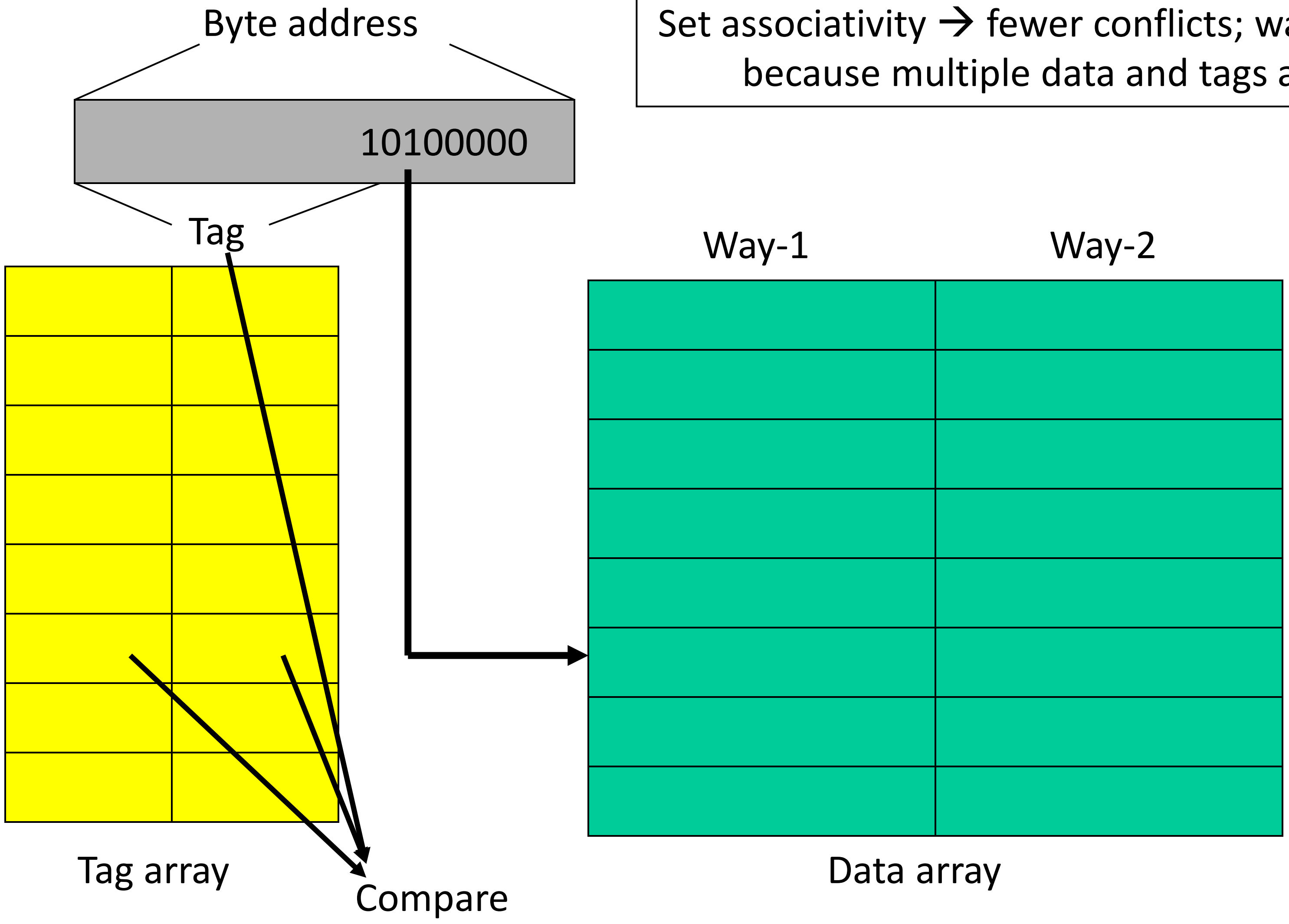
Increasing Line Size

A large cache line size → smaller tag array, fewer misses because of spatial locality



Associativity

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



Problem 2

- Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A

Problem 2

- Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH MM H MM HM HMM M M M

Problem 3

- Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A

Problem 3

- Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH M MH MM HM HMM M H M

Problem 4

- 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?

Equations:
Data array size (cache size) = #sets x #ways x blocksize
Tag array size = #sets x #ways x tagsize
Index bits = $\log_2$ (#sets)
Offset bits = $\log_2$ (blocksize)
Tag bits + index bits + offset bits = address width

Problem 4

- 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address
- How many sets? 64
- How many index bits (6), offset bits (6), tag bits (28)?
- How large is the tag array (28 Kb)?

Problem 5

- 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address
- How many sets? How many ways?
- How many index bits, offset bits, tag bits?
- How large is the tag array?

Problem 5

- 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address
- How many sets (1) ? How many ways (128) ?
- How many index bits (0), offset bits (6), tag bits (34) ?
- How large is the tag array (544 bytes) ?