CPSC 313: Computer Hardware and Operating Systems

Unit 3: Caching
Managing and Evaluating Writes

Administration

- Quiz 3 is next week (sign up for Quiz 3, viewing, and retake times!)
- Lab 6: due shortly
- Lab 7: out Friday (not as big as Lab 5, but still substantial)
- Tutorial 6: this week
- Drop with W deadline on Friday: We want you in the class! (But... if you're certain for some reason that you want to drop, don't miss the deadline!)

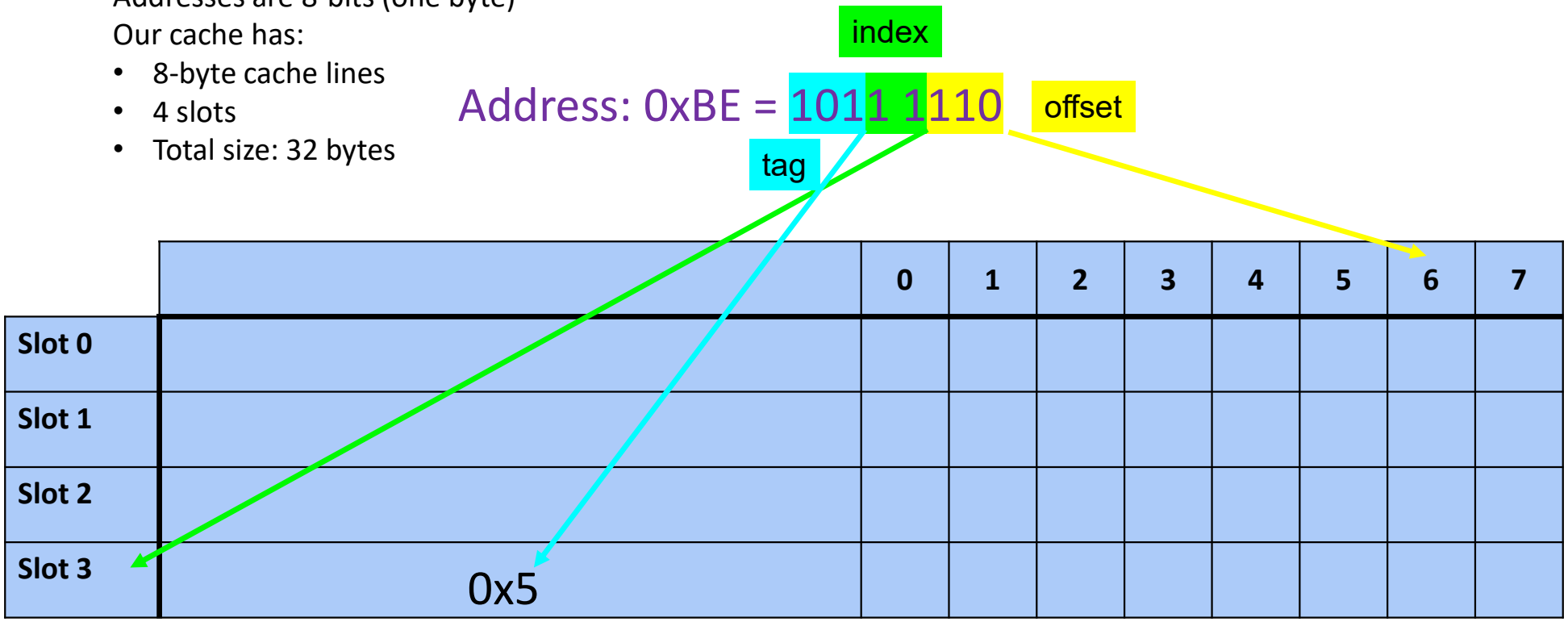
Today

- Learning Objectives
  - Review finding things in a cache.
  - Identify cache metadata requirements.
  - Evaluate cache performance in the presence of writes.
- Reading
  - Section 6.4.5

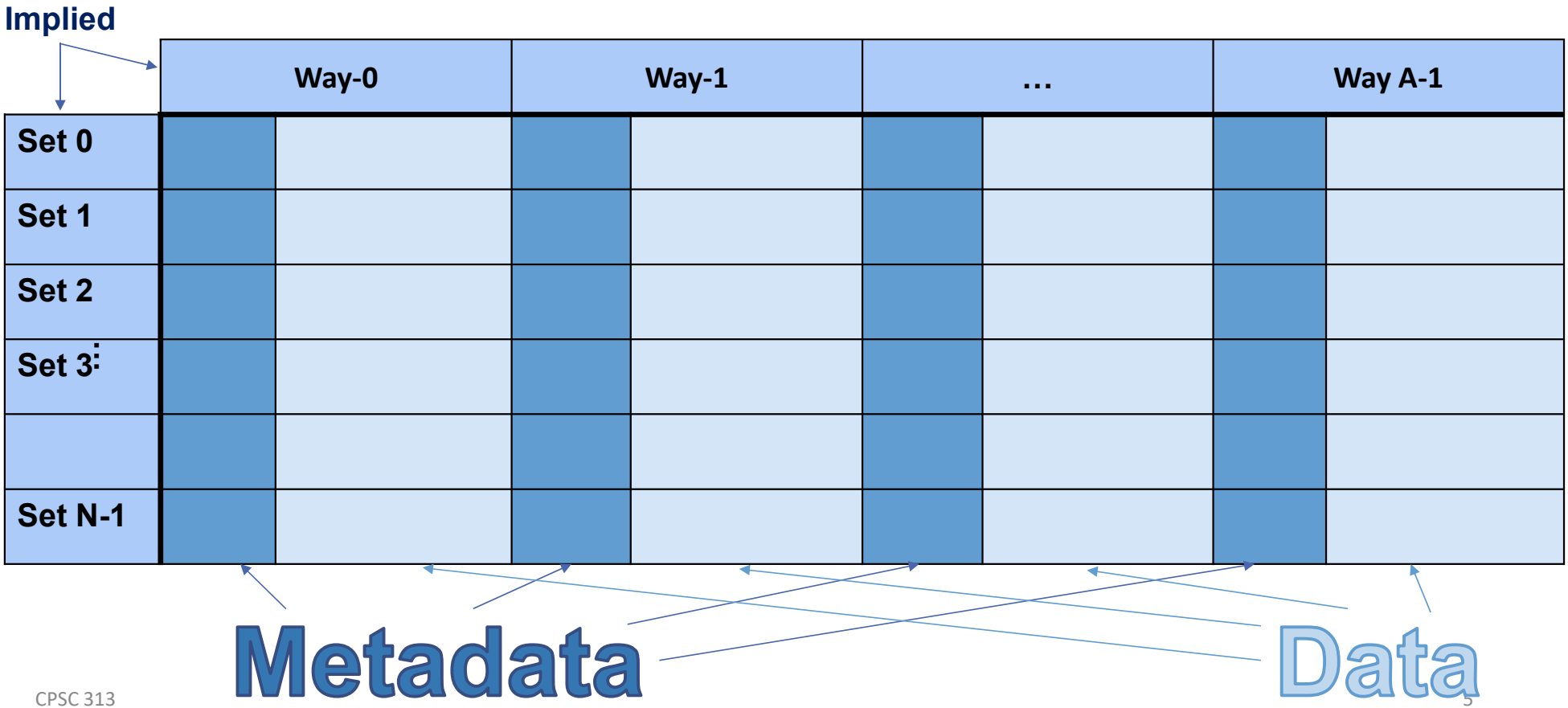
Part 1: Recall our Teeny Tiny Cache

Addresses are 8-bits (one byte)
Our cache has:

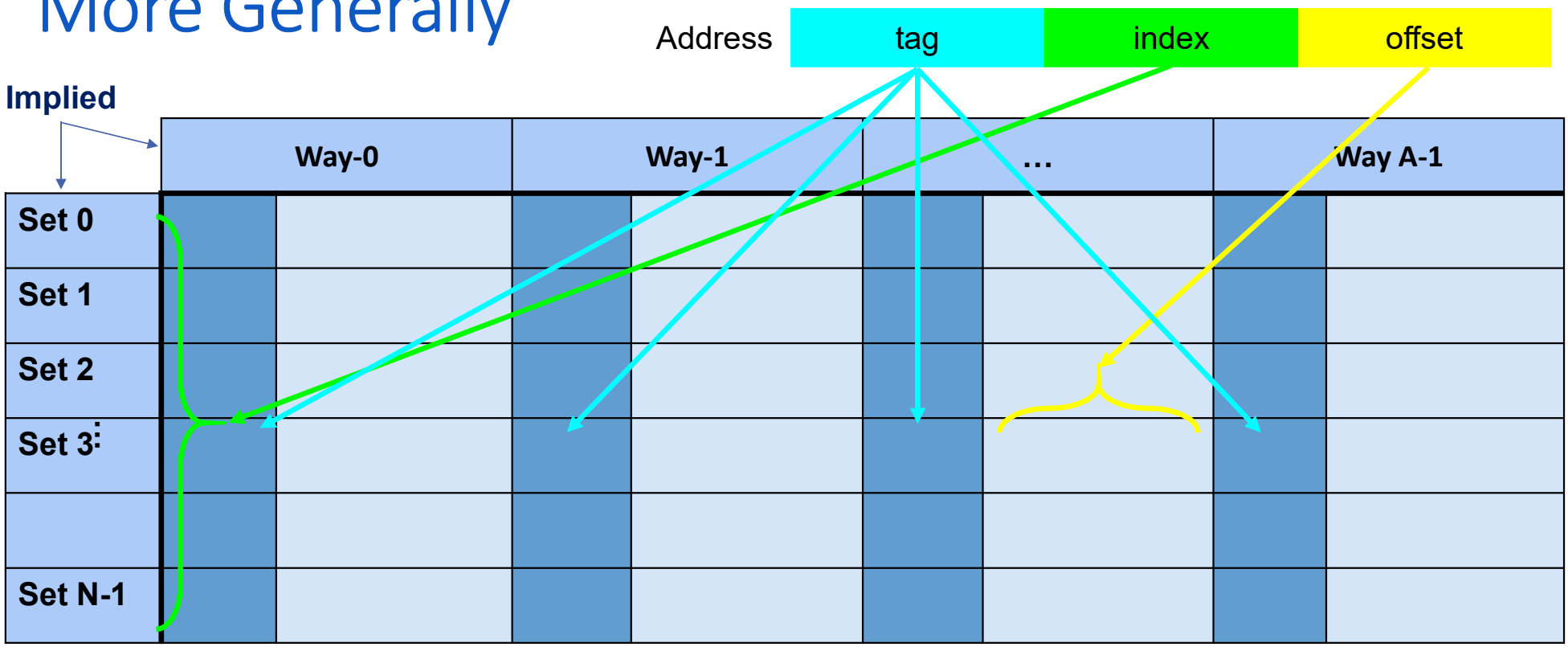
- 8-byte cache lines
- 4 slots
- Total size: 32 bytes



More Generally



More Generally



Metadata

Data

Anatomy of Addresses and Cache Lines

- Address
  - offset — selects starting byte in a cache line — # bits = $\log_2$ (cache line size)
  - index — selects a cache set — # bits = $\log_2$ (number of sets in cache)
  - tag — identifies line in a set using associative lookup — # bits = Everything left in address

- Cache Line Metadata
  - Valid bit: 1 iff line contains valid data (if 0: none of the other data/metadata is meaningful)
  - Tag: used to determine precisely which address is stored in a cache line: for a hit, the tag bits in the address must match the tag bits in a slot's metadata in the set indicated by the address's index bits
  - (New for writeback): Dirty bit: 1 iff line contains *dirty* data; i.e., updates that have not be sent to data source
  - (As needed for replacement policy): Replacement metadata, e.g., LRU information

Cache Example (likely skipped in class)

- 4 KB cache
  - Direct-mapped
  - 32-byte cache lines
  - 16-bit addresses

	Number of Bits	Starting Bit	Ending Bit
Offset			
Index			
Tag			

Cache Example

- 4 KB cache
  - Direct-mapped
  - 32-byte cache lines
  - 16-bit addresses

Remember we number bits starting at 0!

	Number of Bits	Starting Bit	Ending Bit
Offset	5	0	4
Index			
Tag			

Cache Example

- 4 KB cache
  - Direct-mapped
  - 32-byte cache lines
  - 16-bit addresses

The cache is direct mapped, so the number of sets is the size of the cache divided by the cacheline size:

4 KB / 32 bytes = $2^{12} / 2^5 = 2^7$ (128)

	Number of Bits	Starting Bit	Ending Bit
Offset	5	0	4
Index			
Tag			

Cache Example

- 4 KB cache
  - Direct-mapped
  - 32-byte cache lines
  - 16-bit addresses

The cache is direct mapped, so the number of sets is the size of the cache divided by the cacheline size:

4 KB / 32 bytes = $2^{12} / 2^5 = 2^7$ (128)

	Number of Bits	Starting Bit	Ending Bit
Offset	5	0	4
Index	7	5	11
Tag			

Cache Example

- 4 KB cache
  - Direct-mapped
  - 32-byte cache lines
  - 16-bit addresses

The cache is direct mapped, so the number of sets is the size of the cache divided by the cacheline size:

4 KB / 32 bytes = $2^{12} / 2^5 = 2^7$ (128)

Be careful! You cannot just add to get the ending bit!

	Number of Bits	Starting Bit	Ending Bit
Offset	5	0	4
Index	7	5	11
Tag			

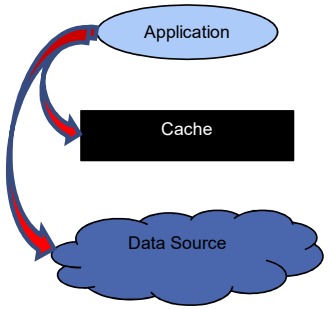
Cache Example

- 4 KB cache
  - Direct-mapped
  - 32-byte cache lines
  - 16-bit addresses

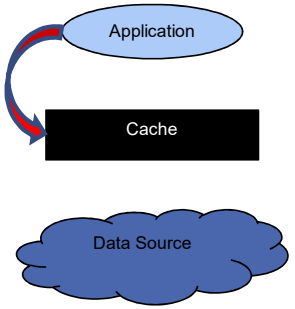
Our addresses are 16 bits and we have “used” 5 for the offset and 7 for the index; how many are “left”?

	Number of Bits	Starting Bit	Ending Bit
Offset	5	0	4
Index	7	5	11
Tag	4	12	15

Part 2: Write Performance: Hits

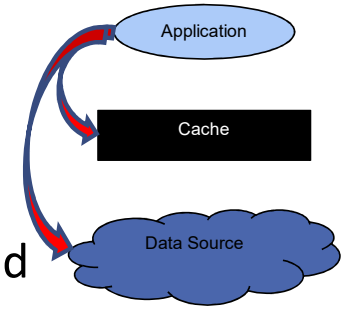


- **Write Through** cache:
  - Data is written to both the cache and the source.
  - Latency ~= latency to write data source (*assume the cache and source writes happen in parallel, so do not add their times*)

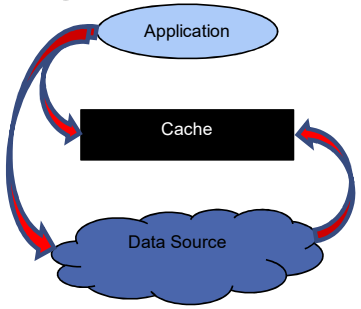


- **Write Back** cache
  - Data is written **only** to the cache, but …
  - The dirty bit in the cache line metadata is set to 1
  - Latency ~= latency to write to the cache

Write Through Cache – Write Miss

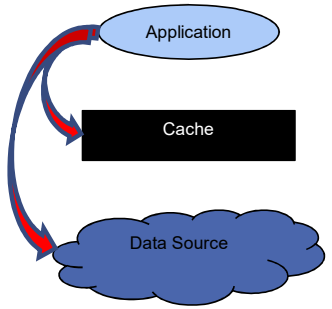


- **No Write Allocate**
  - Data is written to the source
  - Cache is checked (or we wouldn't know it was a miss!) but unchanged
- **Write Allocate**
  - Acts like a read miss; read cache line from the source into the cache
  - Strange combination: write-through is usually for when simplicity is the goal
  - Benefits workloads where read locality mirrors write locality



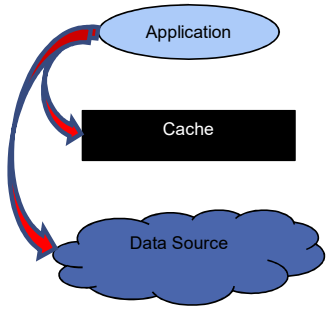
Example 1

- **Write-Through, No-Write-Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write source == 20 ns
- Latency of:
  - Write hit?
  - Write miss?



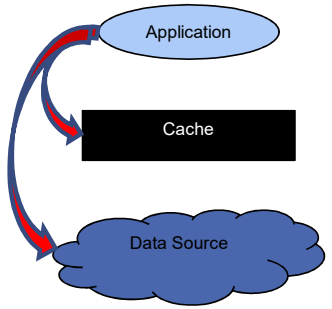
Example 1

- **Write-Through, No-Write-Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write source == 20 ns
- Latency of:
  - Write hit? 20 ns (hides the 2ns access to the cache)
  - Write miss?



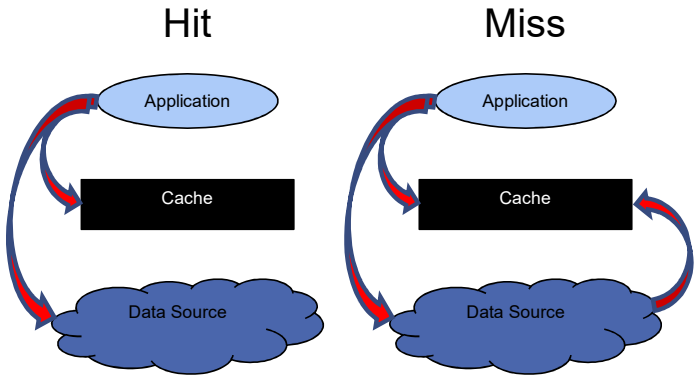
Example 1

- **Write-Through, No-Write-Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write source == 20 ns
- Latency of:
  - Write hit? 20 ns (hides the 2ns access to the cache)
  - Write miss? 20 ns (*also* hides the 2ns access to the cache)



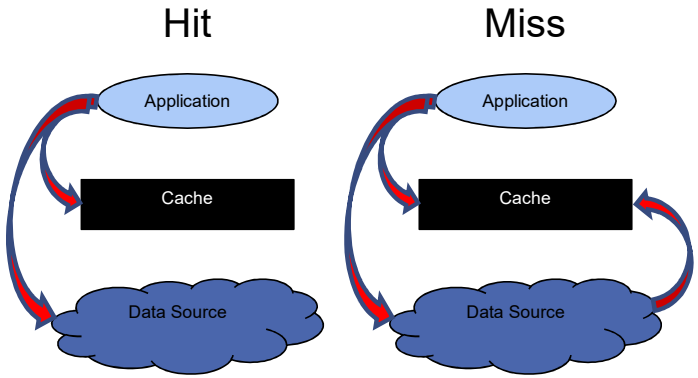
Example 2

- **Write-Through, Write Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write source == 20 ns
- Latency of:
  - Write hit?
  - Write miss?

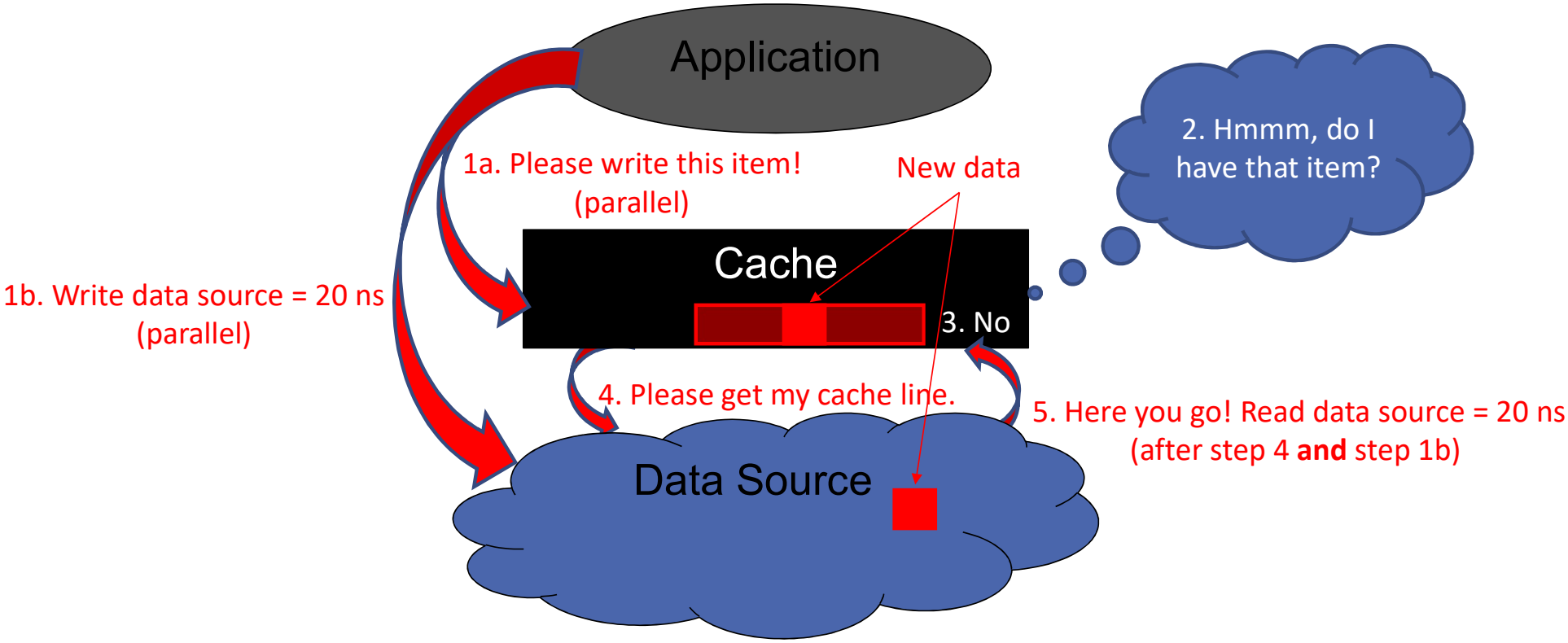


Example 2

- **Write-Through, Write Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write source == 20 ns
- Latency of:
  - Write hit? 20 ns
  - Write miss?

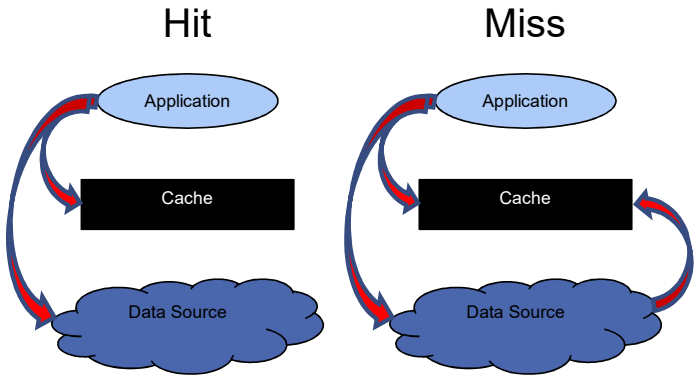


Write Through: Miss – Write Allocate



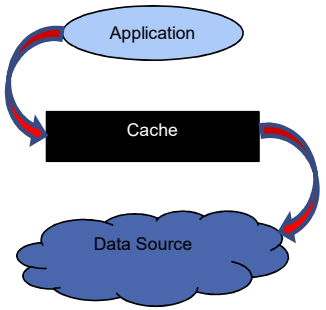
Example 2

- **Write-Through, Write Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write source == 20 ns
- Latency of:
  - Write hit? 20 ns
  - Write miss? 40 ns
    (data source time dominates; 1 write + 1 read)



Write Back Cache – Write Miss (No Allocate)

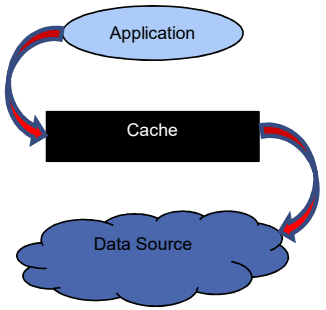
- **No Write Allocate**
  - Write goes to data-source; cache is unchanged
  - Strange combination: writes to same block will miss again!



- If basic latency is:
  - Latency to read/write cache == 2 ns
  - Latency to read/write data source == 20 ns
- Latency of write miss?

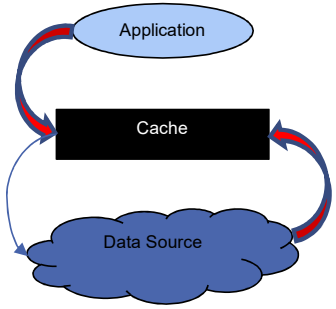
Write Back Cache – Write Miss (No Allocate)

- **No Write Allocate**
  - Write goes to data-source; cache is unchanged
  - Strange combination: writes to same block will miss again!



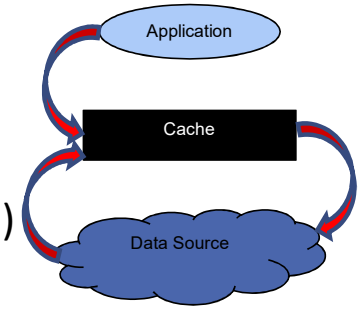
- If basic latency is:
  - Latency to read/write cache == 2 ns
  - Latency to read/write data source == 20 ns
- Latency of write miss? 22ns

Write Back Cache – Write Miss (Allocate)



- **Write Allocate**
  - Evict nothing (rare!) or evict a **clean** cache slot:

    latency ~= 1 data source read + 1 cache write



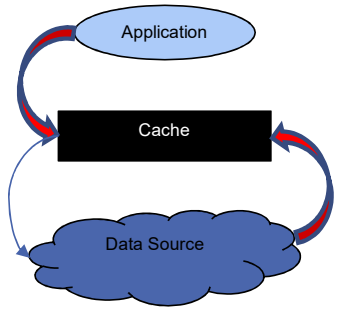
  - Evict a **dirty** cache slot:

    latency ~= 2 data source read/writes + 1 cache write
    (Discover the miss; write the evicted cache line; read/update the new cache line.)

Example 3

- **Write-Back, Write-Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write data source == 20 ns
- Latency of:
  - Write hit?
  - Write miss; clean line evicted?
  - Write miss; dirty line evicted?

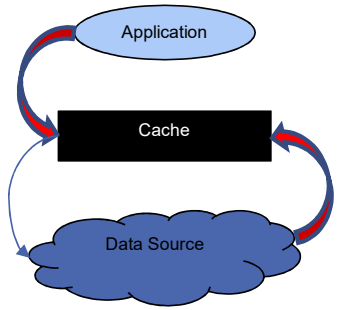
Miss
(clean line evicted)



Example 3

- **Write-Back, Write-Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write data source == 20 ns
- Latency of:
  - Write hit? 2 ns
  - Write miss; clean line evicted?
  - Write miss; dirty line evicted?

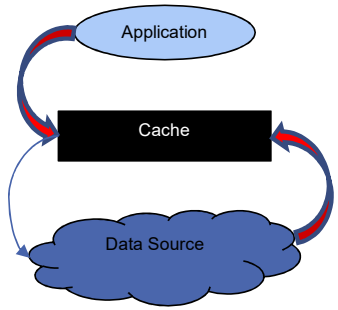
Miss
(clean line evicted)



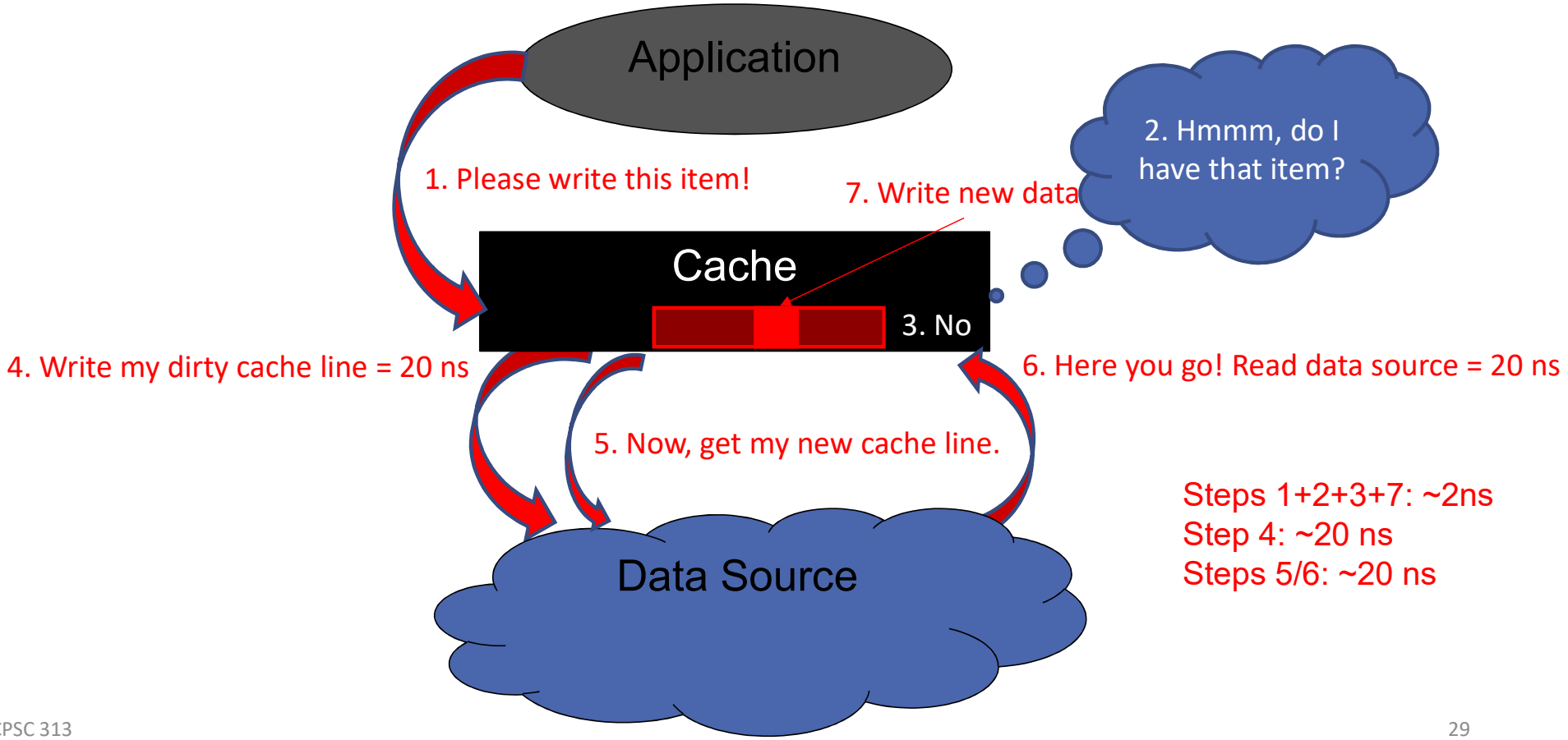
Example 3

- **Write-Back, Write-Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write data source == 20 ns
- Latency of:
  - Write hit? 2 ns
  - Write miss; clean line evicted? 22 ns
  - Write miss; dirty line evicted?

Miss
(clean line evicted)



Write-Back: Miss – Write Allocate (Dirty)

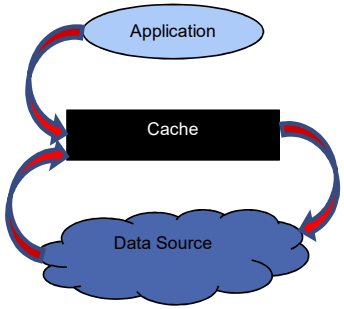


Steps 1+2+3+7: ~2ns
Step 4: ~20 ns
Steps 5/6: ~20 ns

Example 3

- **Write-Back, Write-Allocate** Cache
  - Latency to read/write cache == 2 ns
  - Latency to read/write data source == 20 ns
- Latency of:
  - Write hit? 2 ns
  - Write miss; clean line evicted? 22 ns
  - Write miss; dirty line evicted? 42 ns

Miss
(dirty line evicted)



Improving write-miss latency

- Note that none of the designs so far improve write miss latency!
  - What property of a *write* (vs. a read) means it may not need to wait?
  - What if we had a place to hold the write until the update completes?
- **Write Buffer**
  - Small, associative buffer
  - On a write miss, the processor writes data into the buffer.
    - Asynchronously (i.e., while new instructions execute), the buffer does whatever is necessary to complete the write.
  - Subsequent reads must check the buffer first.
  - More common with write-back, write-allocate caches

Now, you get to practice …

- In-class exercise: Write Caching (notice that you need to do multiple instances of some problems).

- Notice too that the variants we ask about are the common ones; you are also responsible for being able to answer questions about uncommon variants!

Wrapping Up

- Computing miss times can be tricky.
- Remember the difference between writeback and writethrough.
- For writeback, don’t forget about the possibility of dirty cache lines!