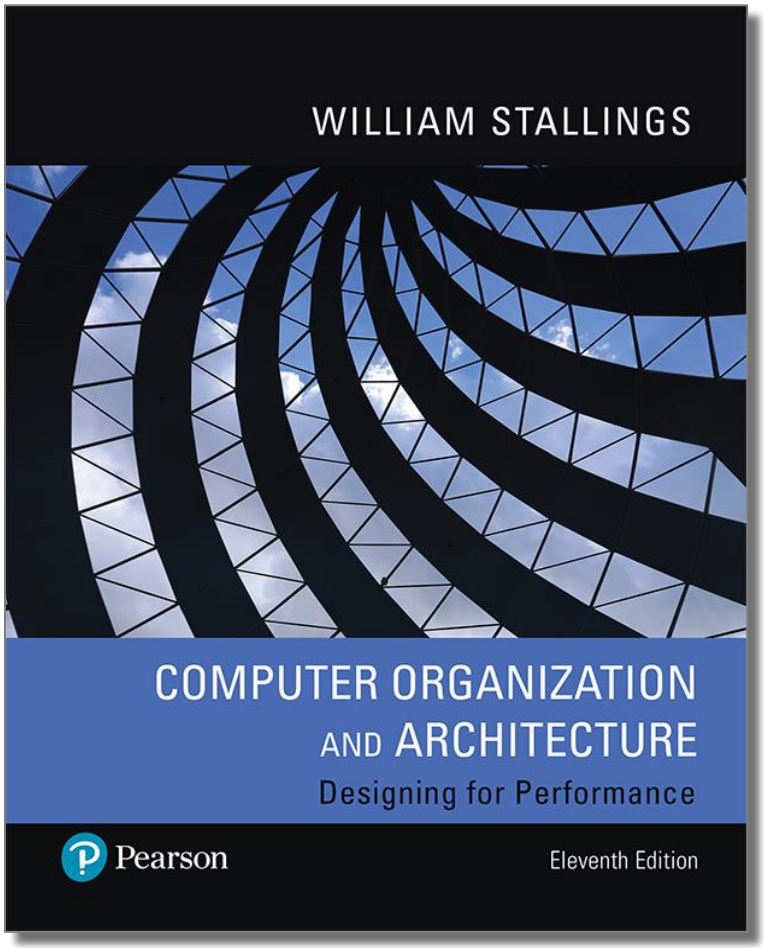
Computer Organization and Architecture
Designing for Performance

11th Edition



Chapter 5

Cache Memory

Table 5.1
Elements of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Replacement Algorithms

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
- For direct mapping there is only one possible line for any particular block and no choice is possible
- For the associative and set-associative techniques a replacement algorithm is needed
- To achieve high speed, an algorithm must be implemented in hardware

The most common replacement algorithms are:

- Least recently used (LRU)
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the most popular replacement algorithm
- First-in-first-out (FIFO)
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- Least frequently used (LFU)
  - Replace that block in the set that has experienced the fewest references
  - Could be implemented by associating a counter with each line

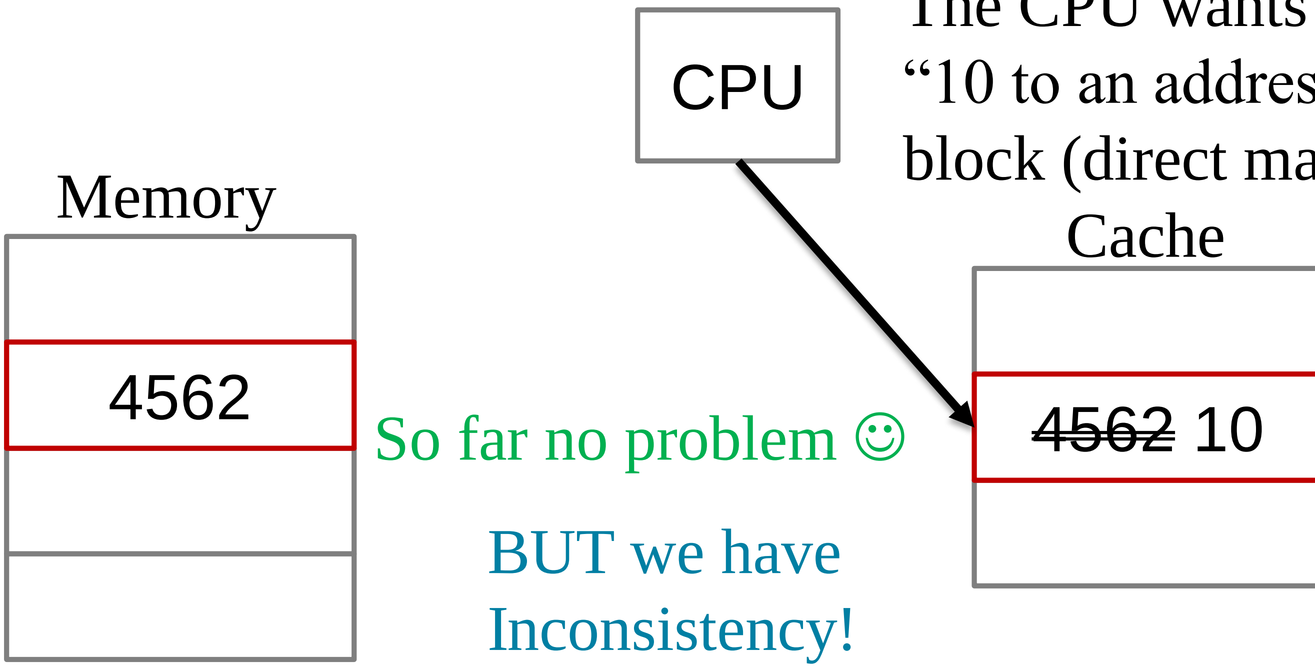
Table 5.1
Elements of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Writing to Cache

- Once we determine the cache block to replace, we may face a problem!!!

The CPU wants to store "10 to an address in the red block (direct mapping)

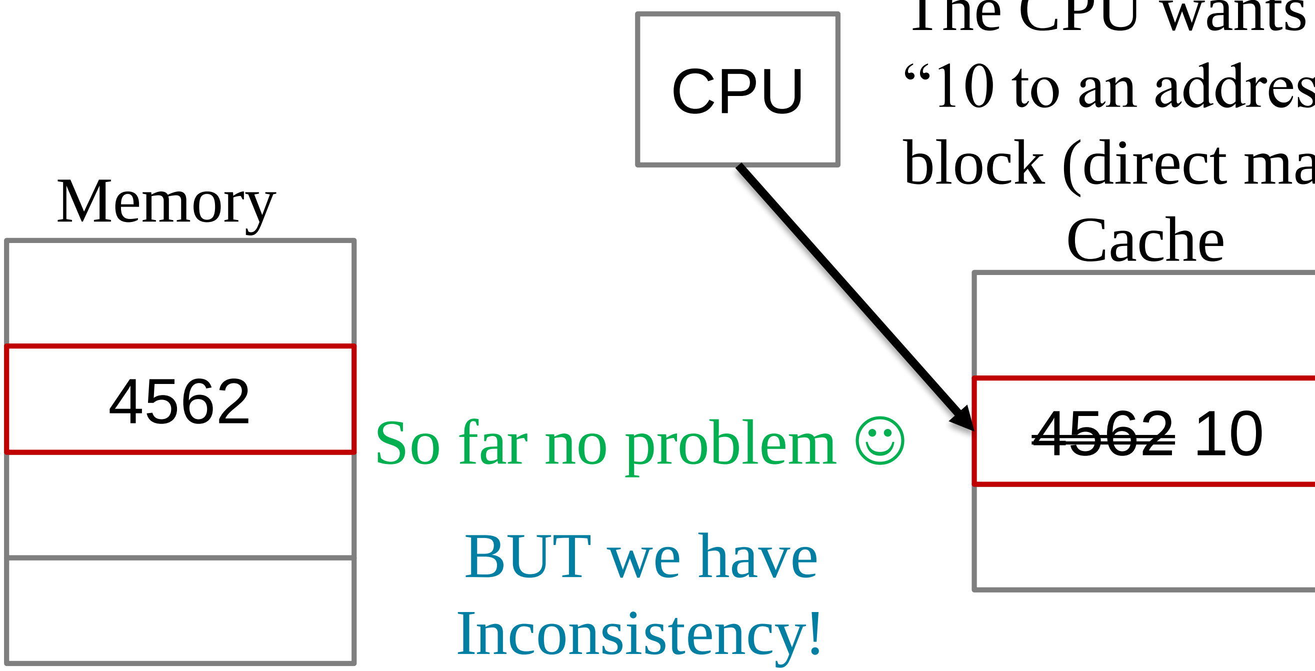


Is it a problem?

Writing to Cache

- Once we determine the cache block to replace, we may face a problem!!!

The CPU wants to store "10 to an address in the red block (direct mapping)



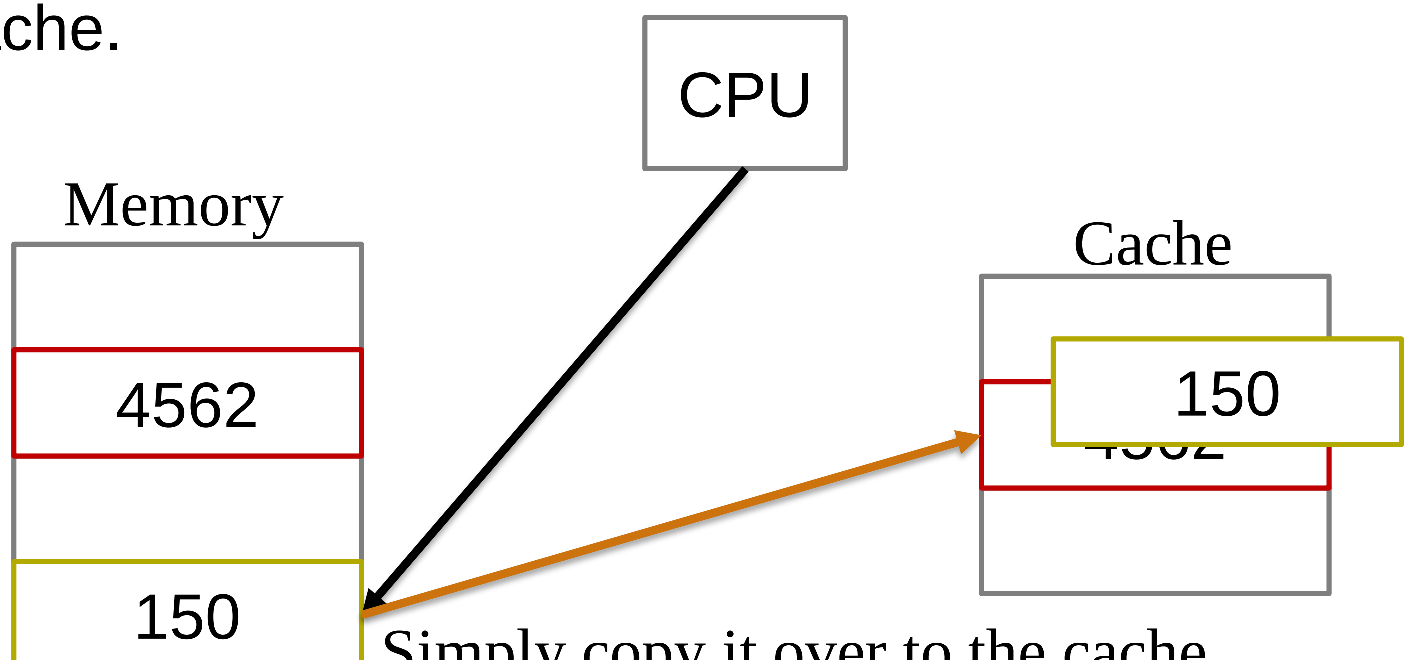
So far no problem ☺

BUT we have Inconsistency!

Is it a problem?

Writing to Cache – Scenario 1

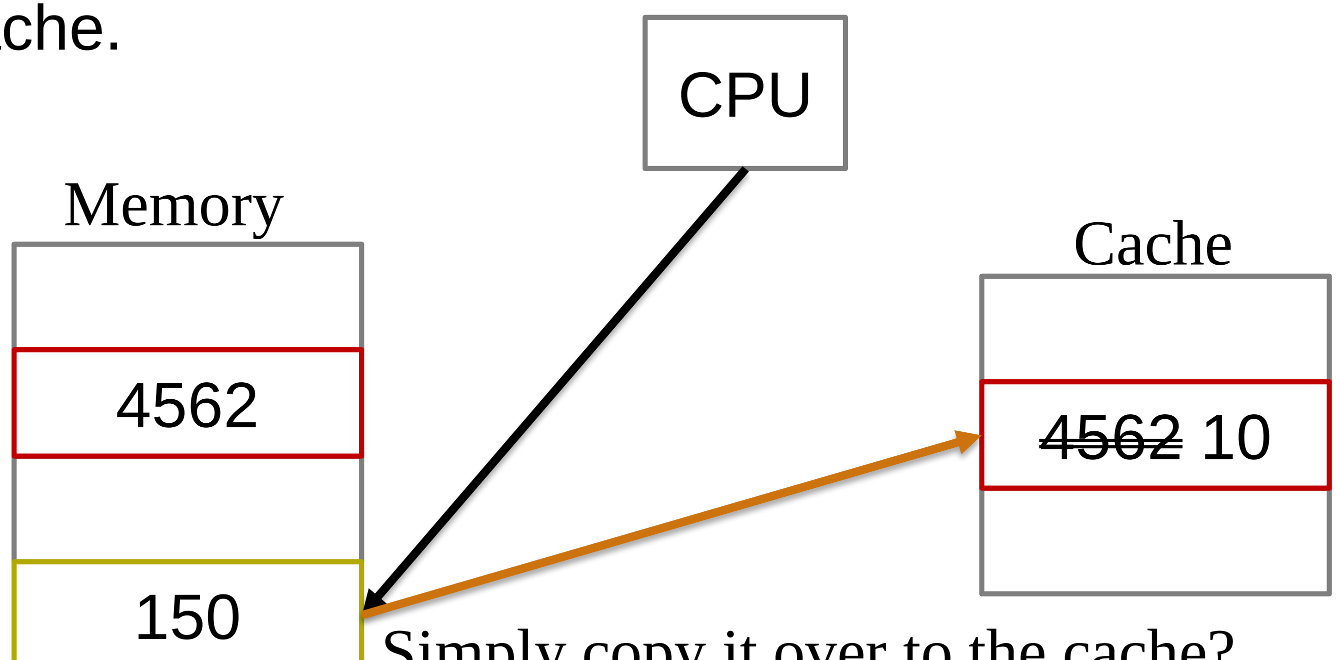
- Suppose the processor needs to access the yellow block which is directly mapped to the same red block in the cache.



Simply copy it over to the cache.
No problem!

Writing to Cache – Scenario 2

- Suppose the processor needs to access the yellow block which is directly mapped to the same red block in the cache.



Simply copy it over to the cache?
What about the red blocks? They are inconsistent.
Value of "10" will be LOST!

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

If at least one write operation has been performed on a word in that line of the cache then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

There are two problems to contend with:

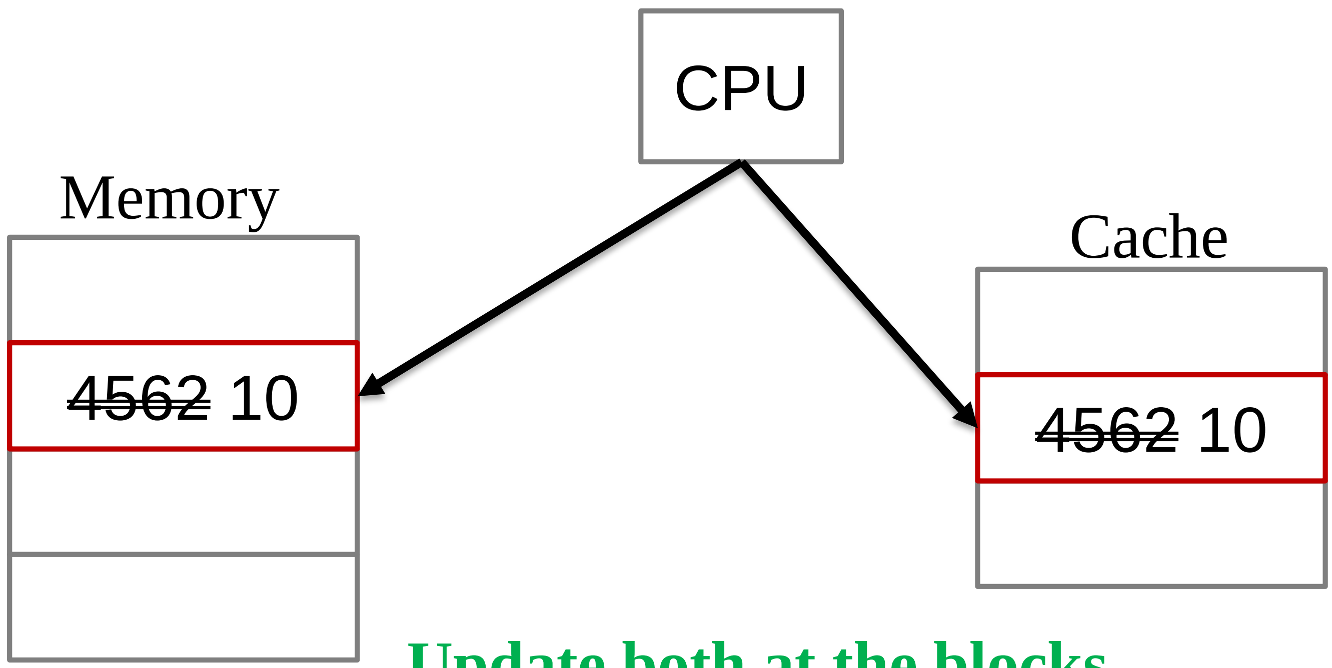
More than one device may have access to main memory

A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

Pearson

Write Through

- The CPU wants to store “10 to an address in the red block (direct mapping)



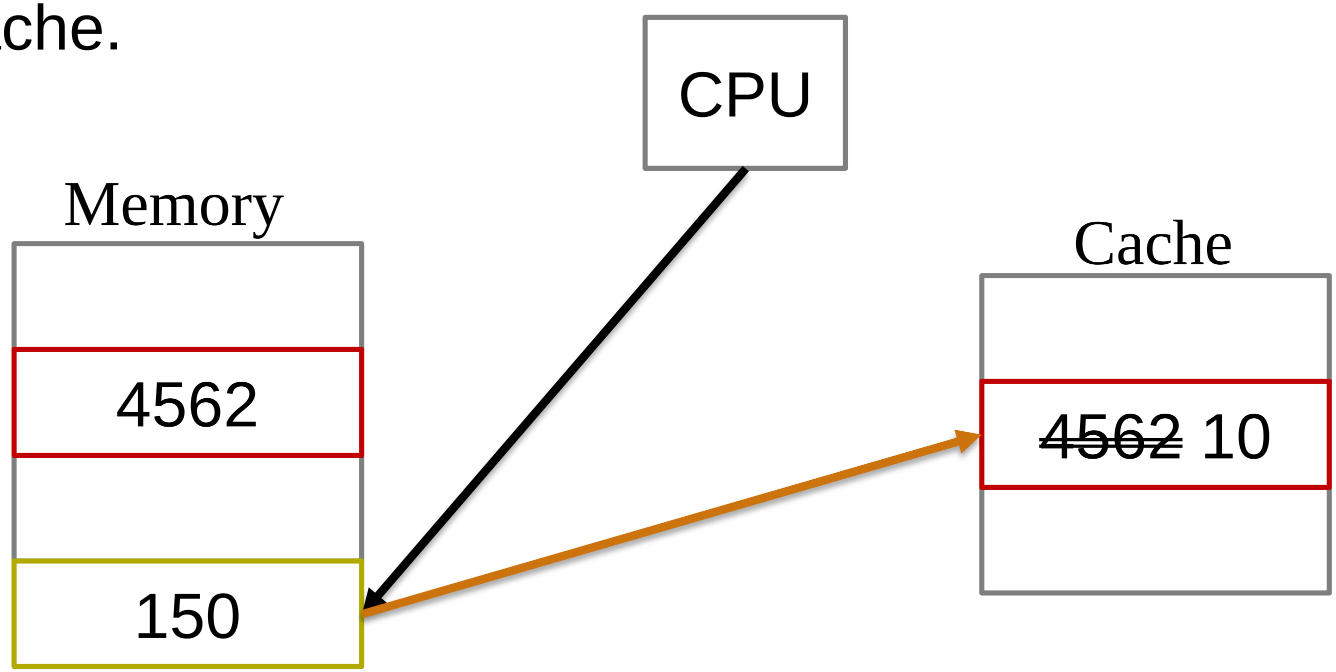
Update both at the blocks.
All blocks will always be consistent.

Write Through

- Write through
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck

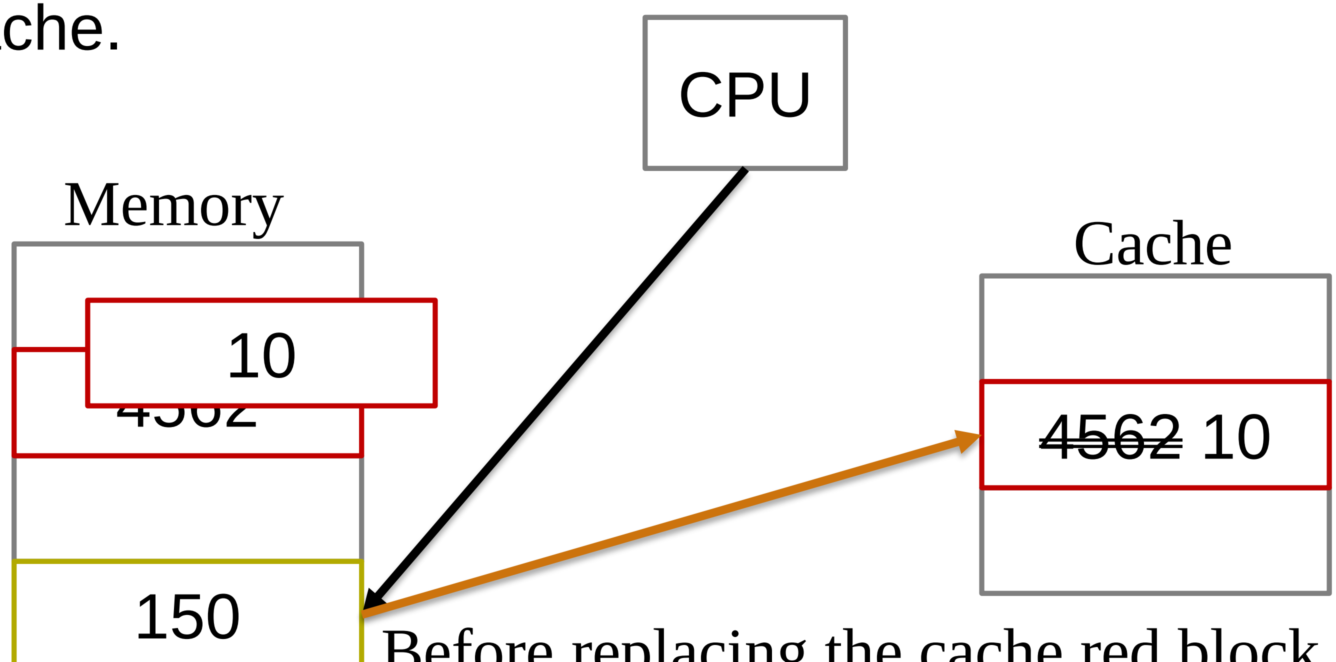
Write Back

- Suppose the processor needs to access the yellow block which is directly mapped to the same red block in the cache.



Write Back

- Suppose the processor needs to access the yellow block which is directly mapped to the same red block in the cache.

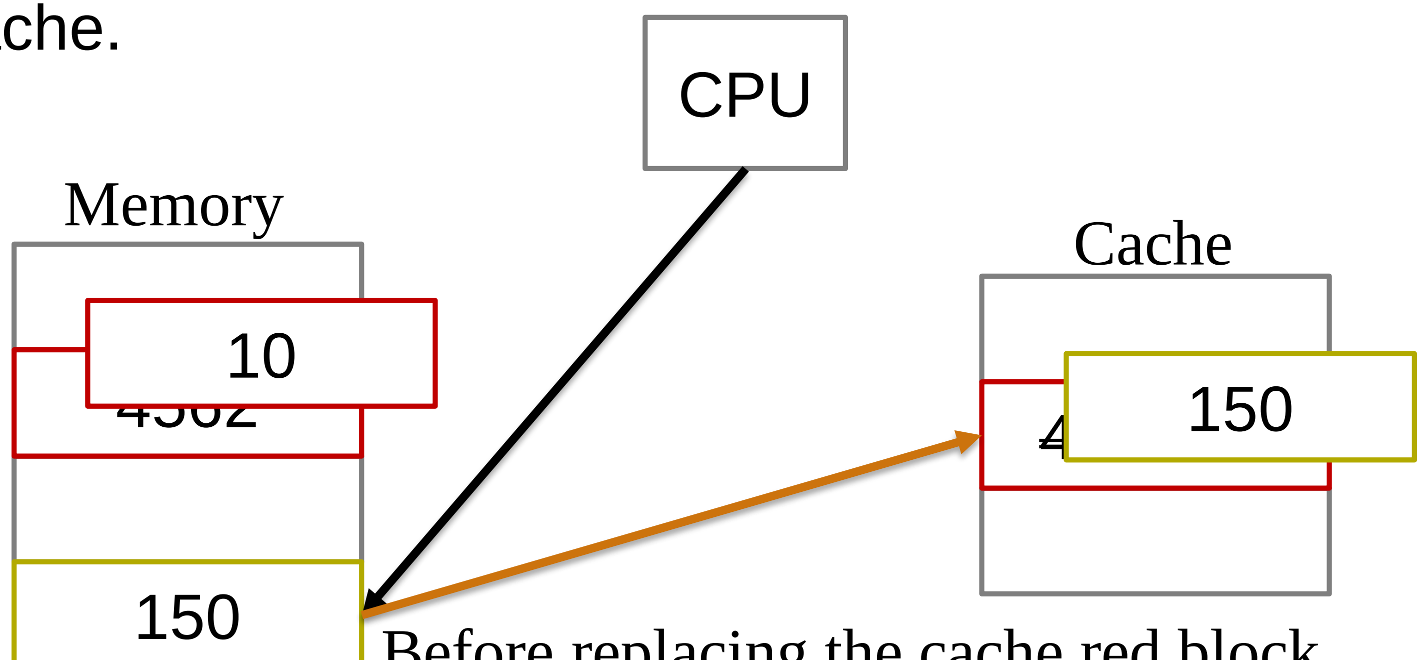


Before replacing the cache red block

Update the red block in the main memory!

Write Back

- Suppose the processor needs to access the yellow block which is directly mapped to the same red block in the cache.



Before replacing the cache red block

Update the red block in the main memory!

Write Back

- Write back
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

Write Miss Alternatives

- If the processor wants to write in a block that is not in the cache, this is called **Write Miss**
- There are two alternatives in the event of a write miss at a cache level:
  - Write allocate
    - The block containing the word to be written is fetched from main memory (or next level cache) into the cache and the processor proceeds with the write cycle
  - No write allocate
    - The block containing the word to be written is modified in the main memory and not loaded into the cache
- Either of these policies can be used with either write through or write back
- No write allocate is most commonly used with write through
- Write allocate is most commonly used with write back

Table 5.1
Elements of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Line Size

When a block of data is retrieved and placed in the cache not only the desired word but also some number of adjacent words are retrieved

As the block size increases the hit ratio will at first increase because of the principle of locality

As the block size increases more useful data are brought into the cache

The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced

Two specific effects come into play:

- Larger blocks reduce the number of blocks that fit into a cache
- As a block becomes larger each additional word is farther from the requested word

Table 5.1
Elements of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	**(We will not cover)**
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Copyright