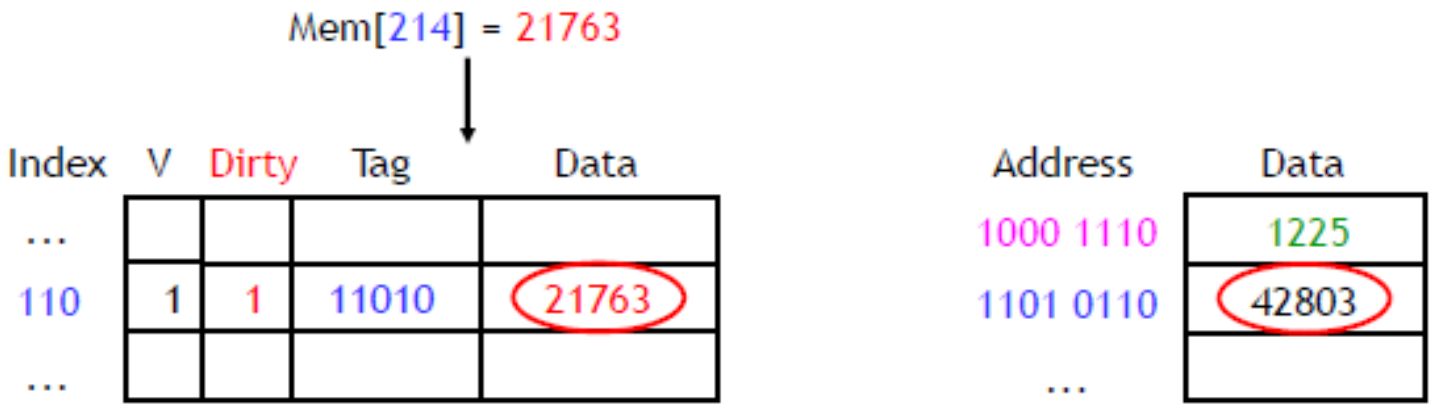
- Review:
  - Write-through?
  - Write-back?

Write-back caches

- In a write-back cache, the memory is not updated until the cache block needs to be replaced (*e.g.*, when loading data into a full cache set).
- For example, we might write some data to the cache at first, leaving it inconsistent with the main memory as shown before.
  - The cache block is marked "dirty" to indicate this inconsistency



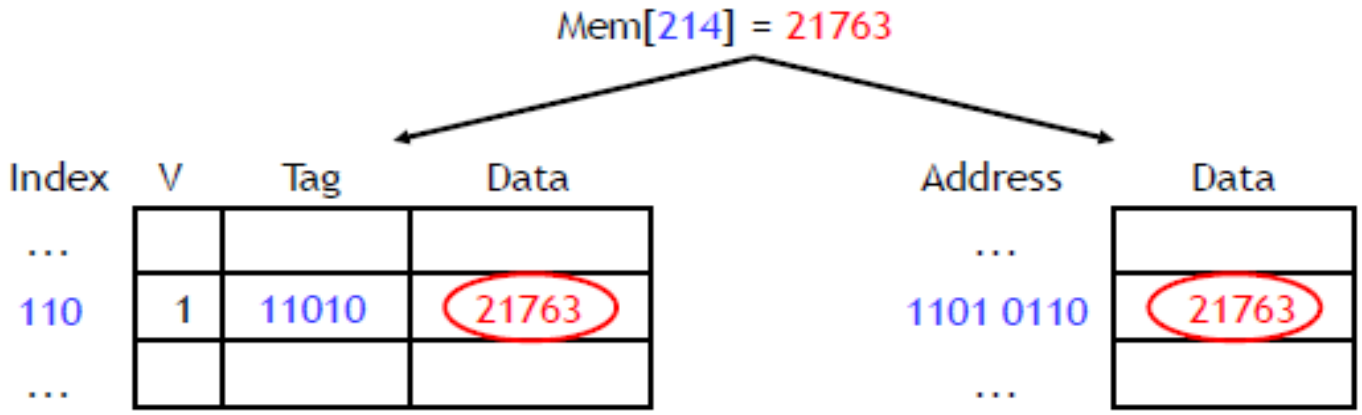
- Subsequent reads to the same memory address?
- Multiple writes to same block?

Write-back cache discussion

- The advantage of write-back caches is that not all write operations need to access main memory, as with write-through caches.
  - If a single address is frequently written to, then it doesn't pay to keep writing that data through to main memory.
  - If several bytes within the same cache block are modified, they will only force one memory write operation at write-back time.

Write-through caches

- A write-through cache forces all writes to update both the cache *and* the main memory.



- This is simple to implement and keeps the cache and memory consistent.
- The bad thing is that forcing every write to go to main memory, we use up bandwidth between the cache and the memory.

Write misses

- A second scenario is if we try to write to an address that is not already contained in the cache; this is called a write miss.
- Let's say we want to store 21763 into Mem[1101 0110] but we find that address is not currently in the cache.

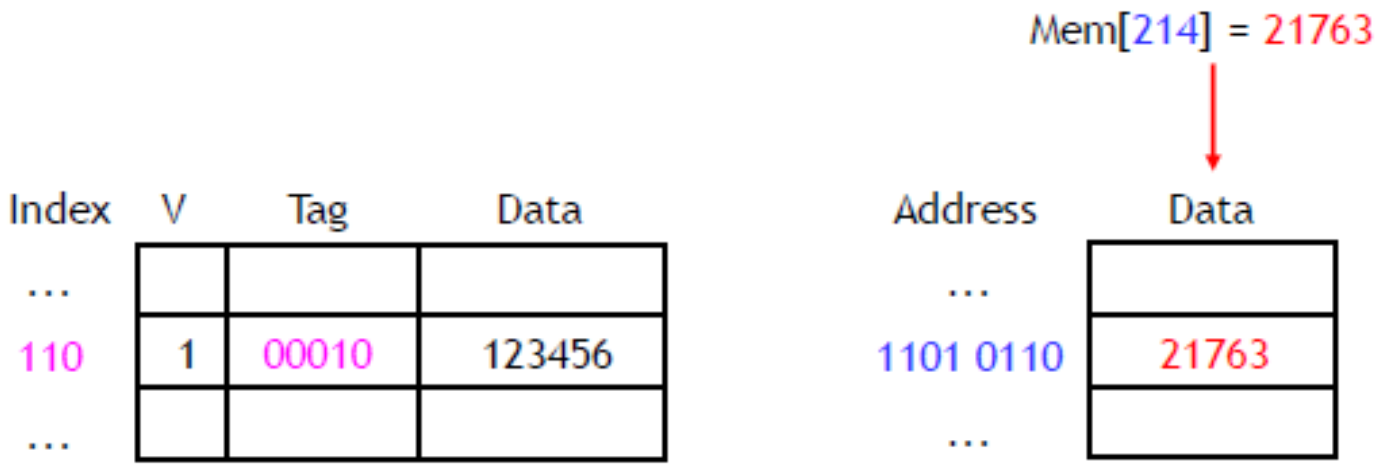
Index	V	Tag	Data
…			
110	1	00010	123456
…			

Address	Data
…	
1101 0110	6378
…	

- When we update Mem[1101 0110], should we *also* load it into the cache?

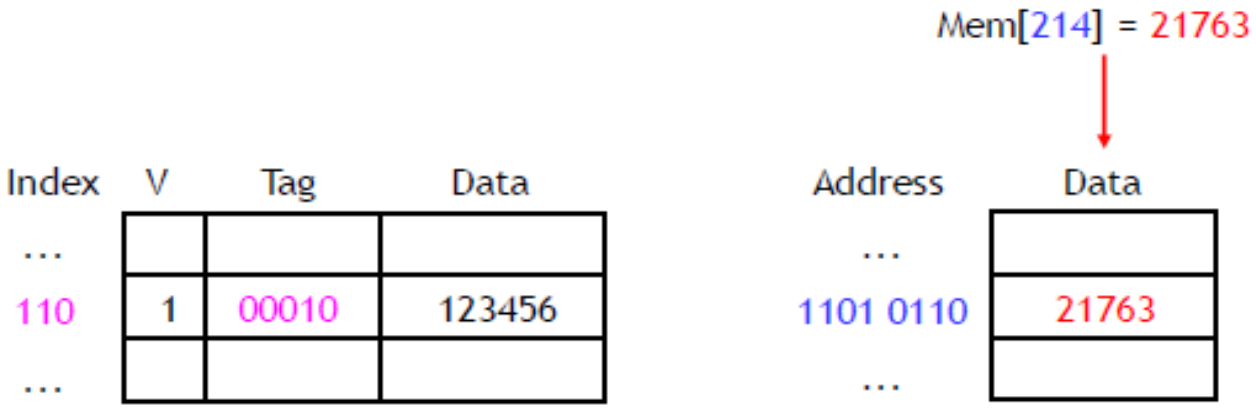
Write around caches (a.k.a. write-no-allocate)

- With a write around policy, the write operation goes directly to main memory *without* affecting the cache.



Write around caches (a.k.a. write-no-allocate)

- With a write around policy, the write operation goes directly to main memory *without* affecting the cache.

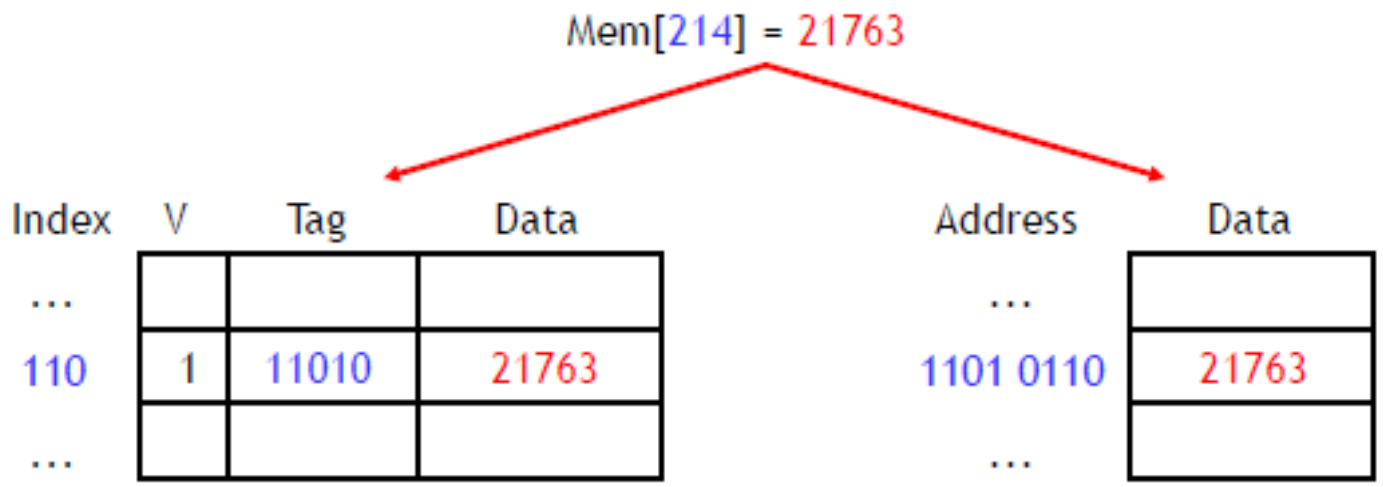


- This is good when data is written but not immediately used again, in which case there's no point to load it into the cache yet.

```
for (int i = 0; i < SIZE; i++)
    a[i] = i;
```

Allocate on write

- An allocate on write strategy would instead load the newly written data into the cache.



- If that data is needed again soon, it will be available in the cache.