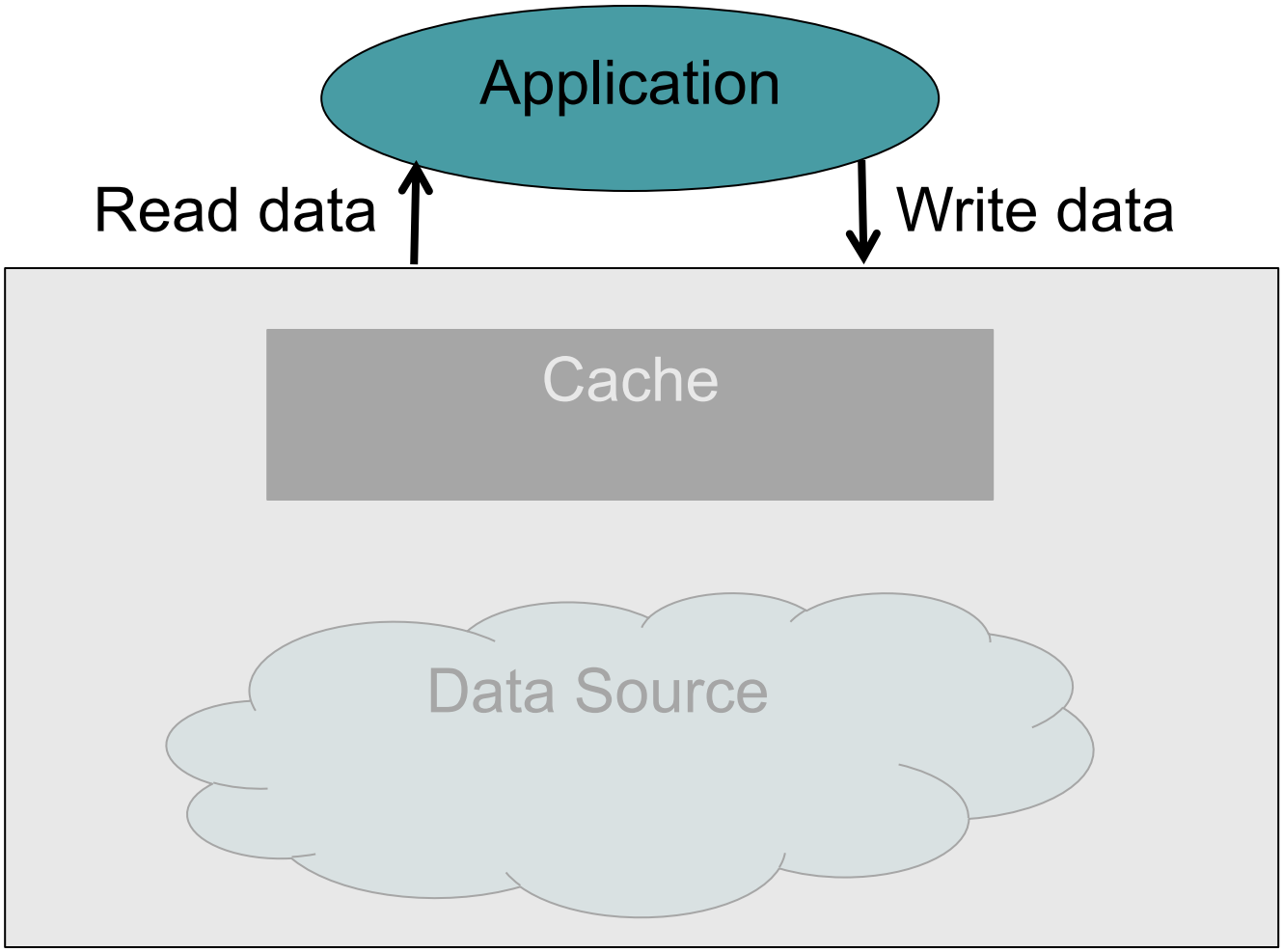
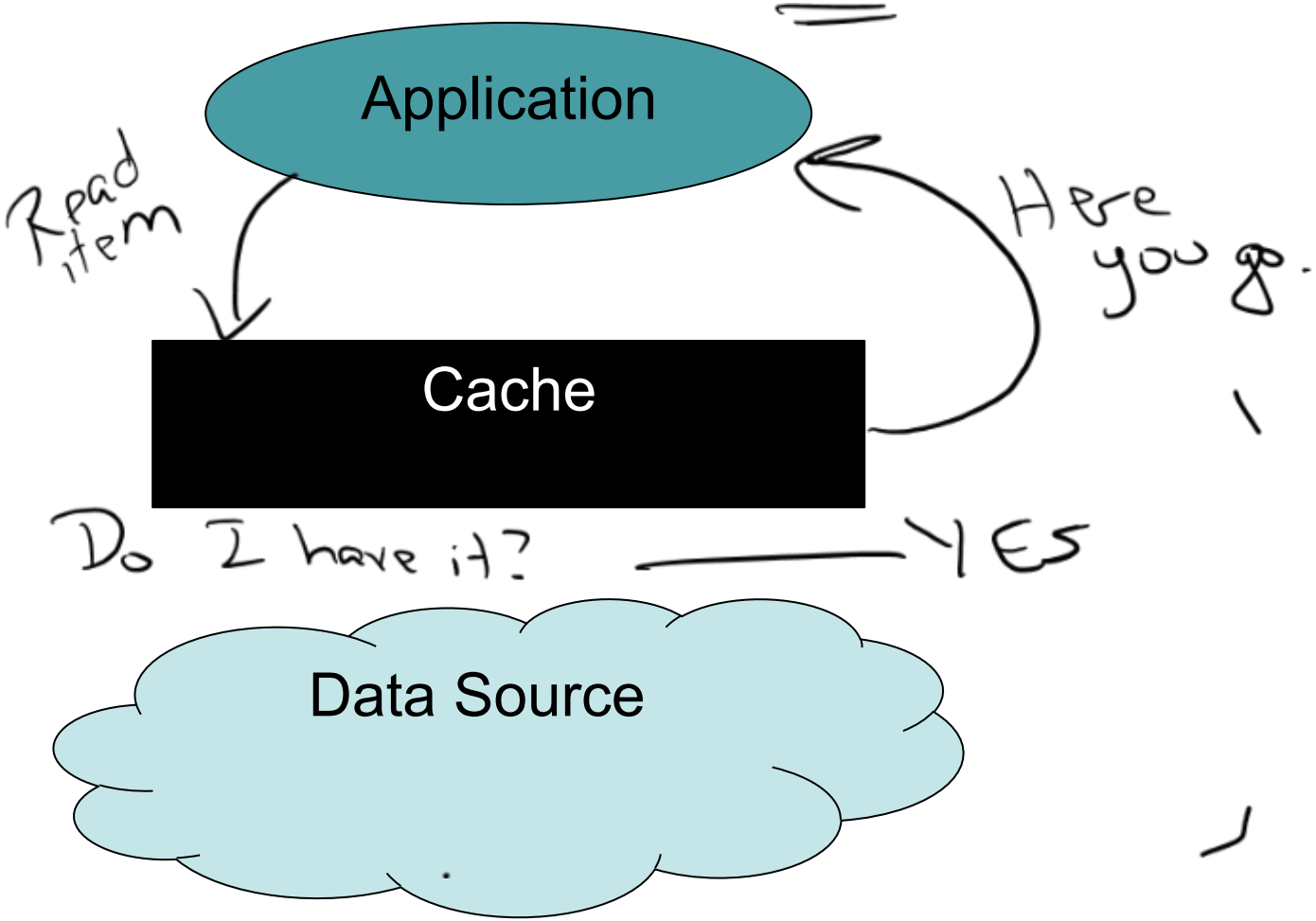
Caching

- Learning Objectives
  - Identify location of common caches in the IO path.
  - Discuss issues particular to write caches
  - Define
    - Cache block
    - Cache slot
    - Cache hit/miss rate
    - Replacement policy
    - Write-back
    - Write-through

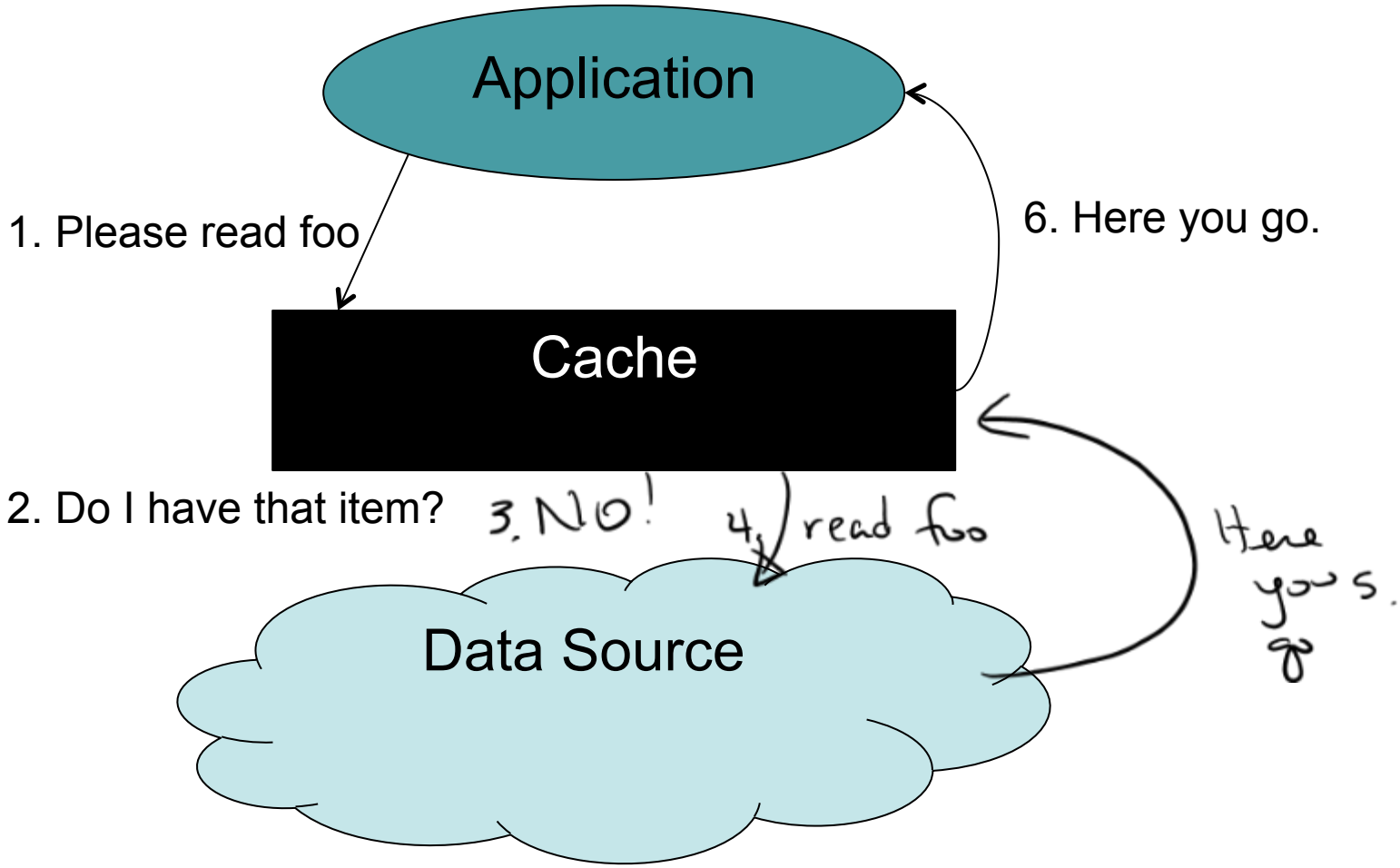
An Abstract Cache



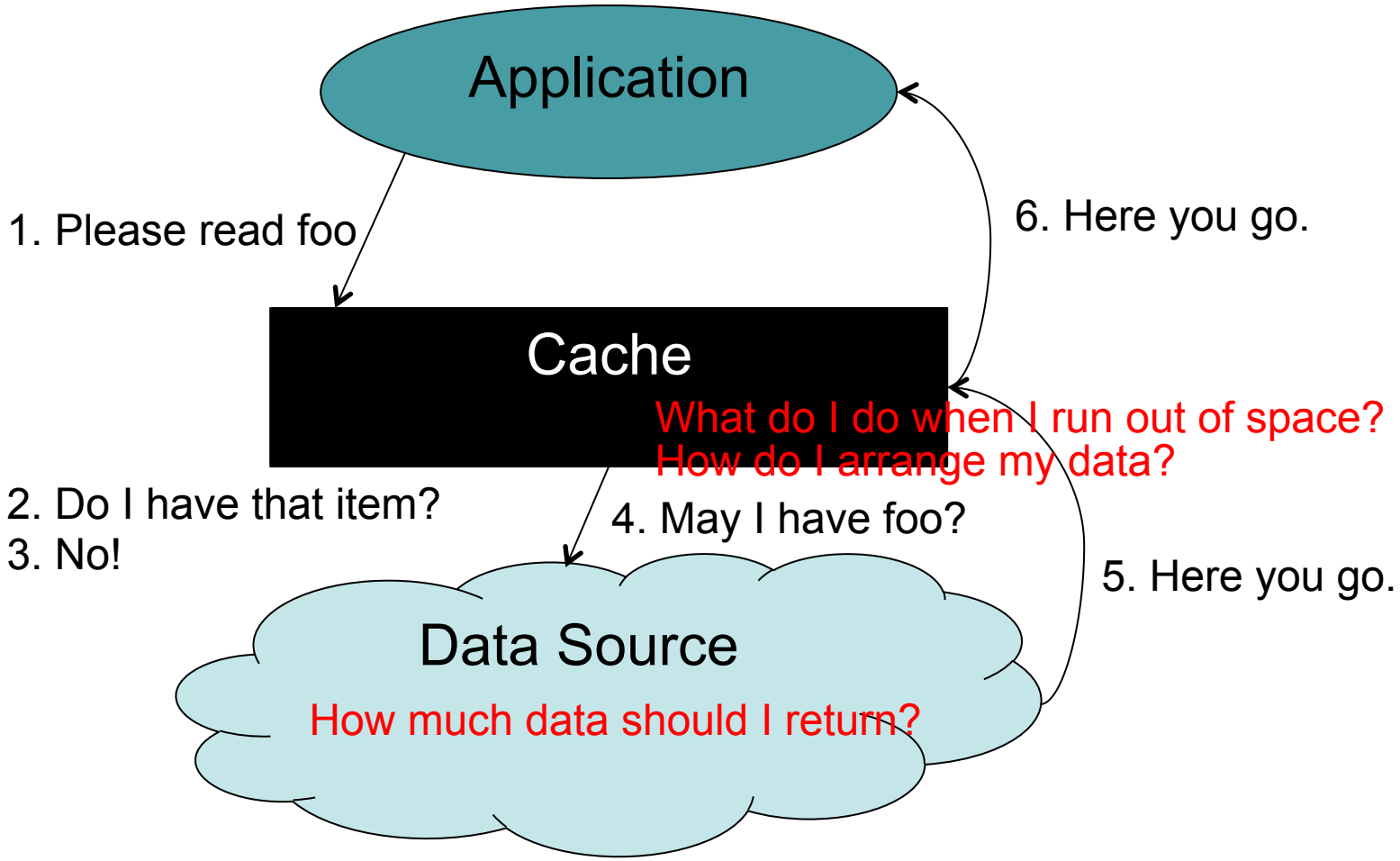
A Read Cache Hit



A Read Cache Miss



A Read Cache Miss -- Decisions



How much data should I return?

- Most storage devices have a native data access and/or transmission size, e.g., disk block (4 KB).
- Caches also have a native size.
  - E.g., Broadwell we talked about in the last video has 64-byte cache lines.
  - Implication: Data moves in and out of caches in 64-byte chunks.
- Block size: the unit in which data is stored in a cache.
  - HW caches: 64 bytes
  - File system caches: 4 KB
  - Object caches: size of the object

What do I do when I fill up?

- A cache has a limited capacity.
- At some point, the application will fill the cache and request another item.
- Caching the new item requires evicting some other item.
- What item do I evict?
  - We need an eviction policy
  - The available decisions here vary between hardware and software.

Eviction Software

- In a perfect world, we'd like to evict the item that is least valuable.
- In the real world, we don't know what that item is.
- Practically all software caches try to approximate this ideal.
  - LRU: Least-recently-used – find the item that has been unused the longest and get rid of that.
  - LFU: Least-frequently-used – find the item that has been used less frequently and get rid of that.
  - Clock: Used in virtual memory systems to approximate LRU, take CS161 for details.
  - Something tuned to known access patterns.

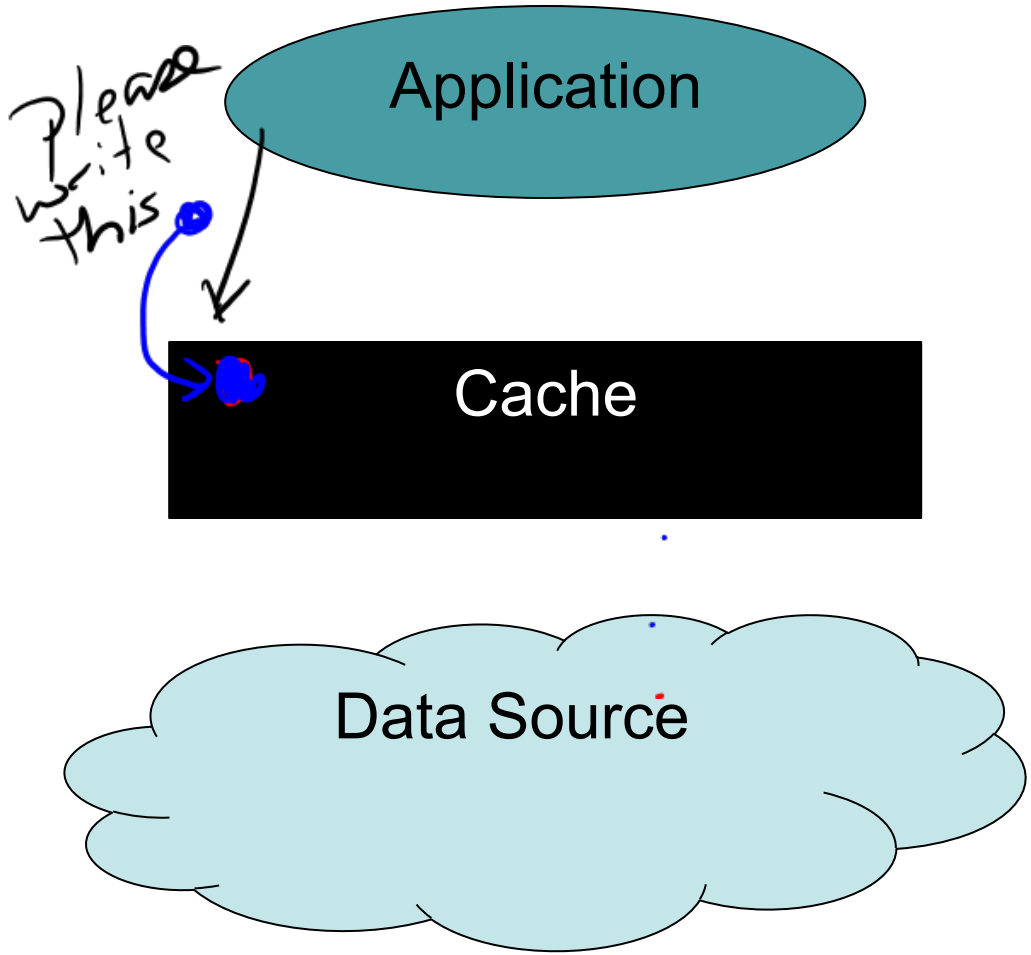
Eviction: Hardware

- Software algorithms are lovely, but are not nearly fast enough for most hardware, e.g., processor, caches.
- Cache is comprised of some number of slots.
- For any particular item, limit the number of possible slots in which it can be placed. Call the number of such slots A. Let n be the total number of slots in the cache.
  - A = 1: Direct mapped: each object can live in exactly one slot in the cache, so you have no choice but to evict the item in that slot.
  - A > 1, A << n: A-way set associative: an object can live in one of A slots; A is typically 2, 4, or 8. On eviction, choose randomly from among the A slots.
  - A = n: Fully associative: an object can live in any slot (like a software cache).
- This answers the, “How do I organize my cache?” question

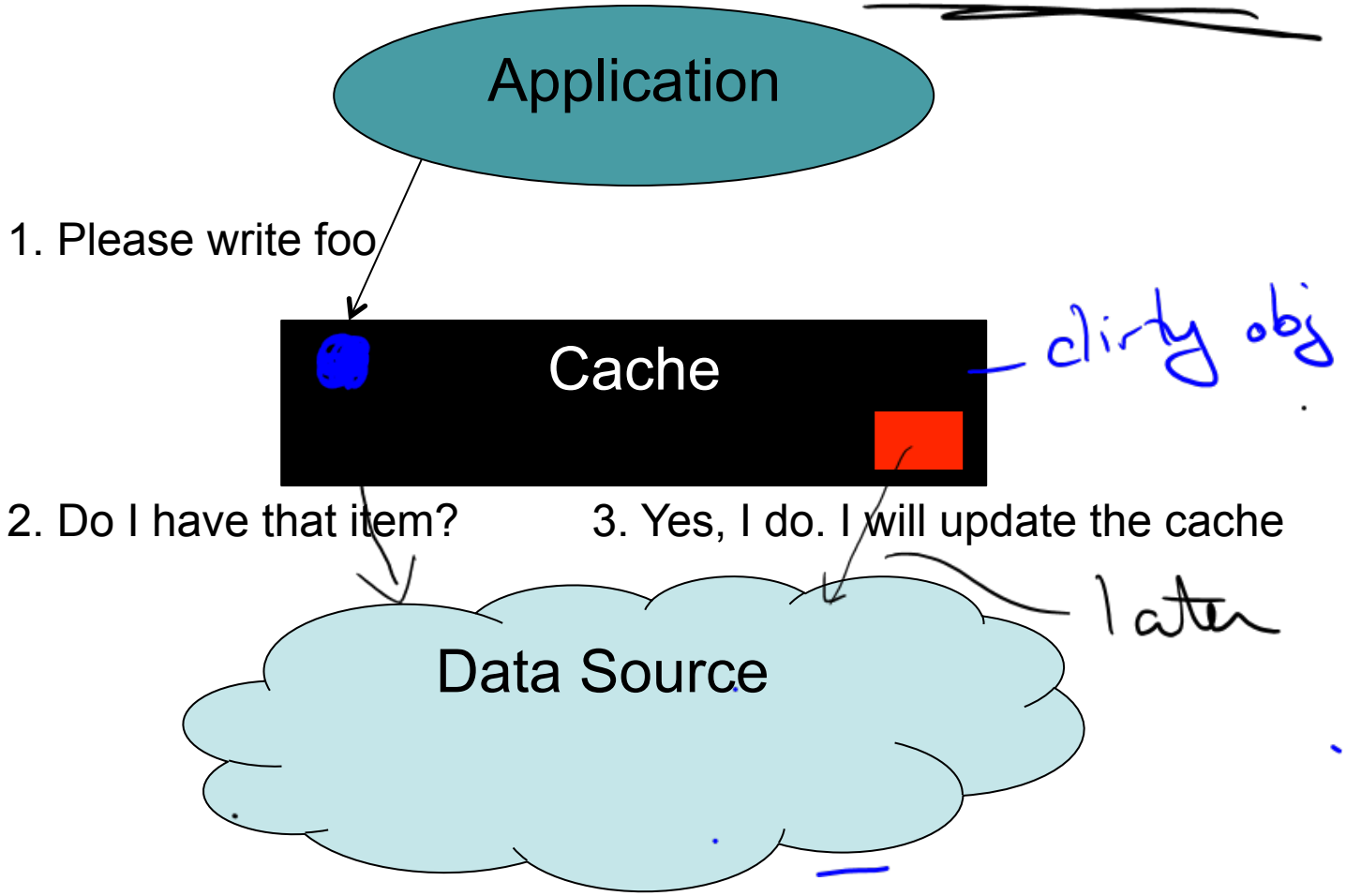
Types of Cache Misses

- Compulsory:
  - On first access to an object, you take a miss; there is little you can do about it.
- Capacity:
  - You are touching more data than can fit in the cache.
  - If your cache were larger, you would have fewer misses.
- Conflict:
  - You have something other than a fully associative cache and are taking misses because there is another item occupying the slots you need.

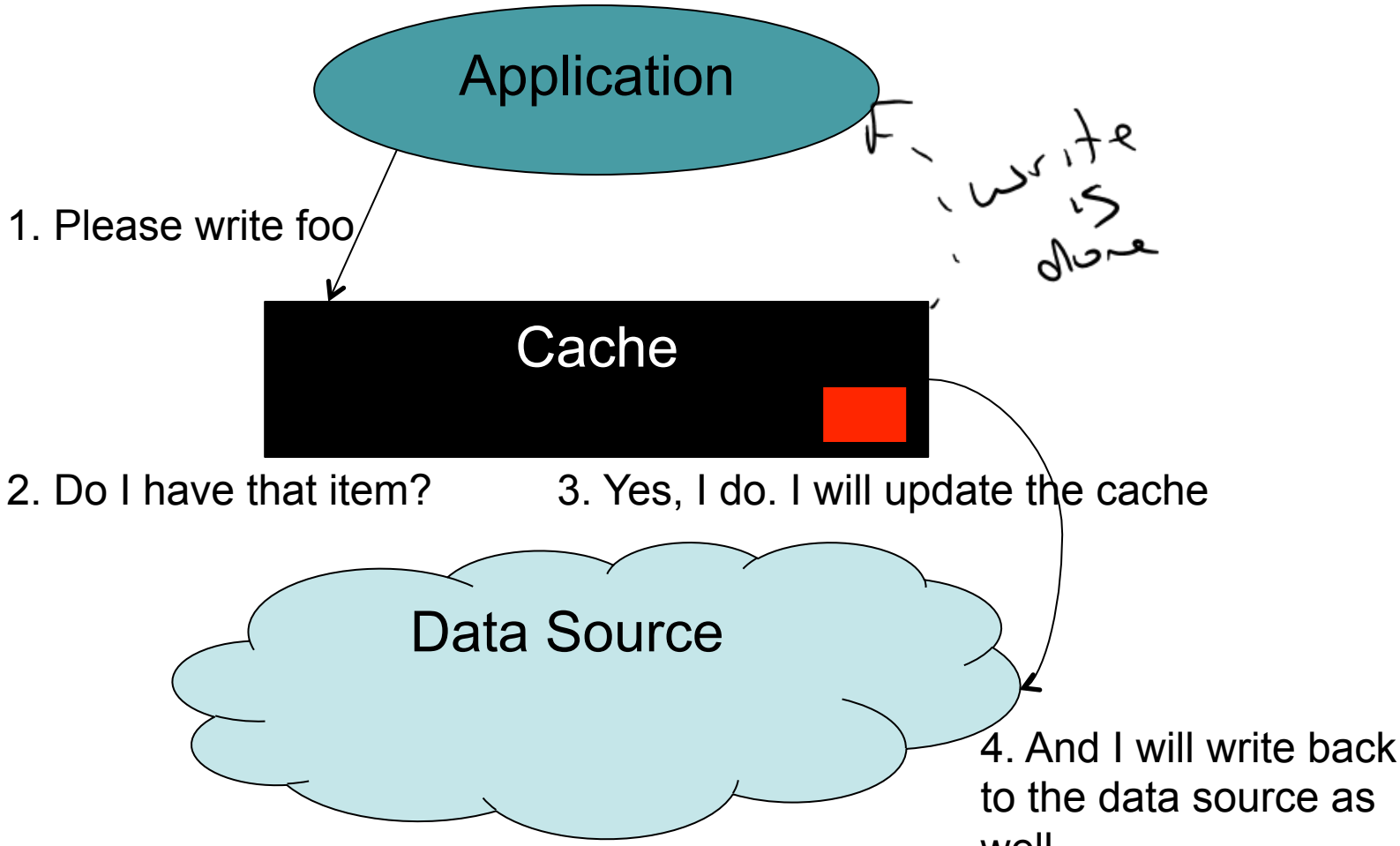
A Write Cache Hit



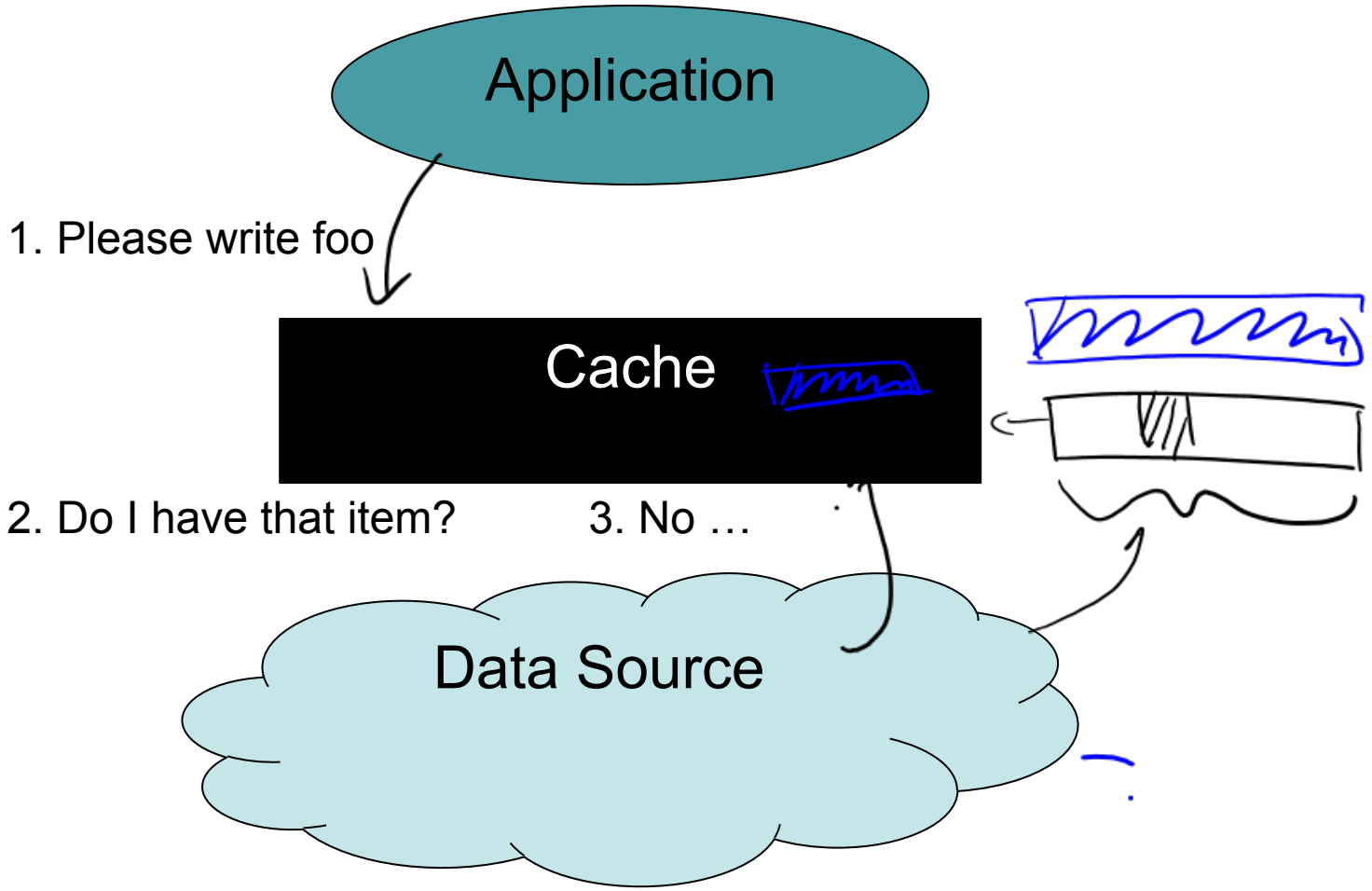
A Write Cache Hit – Write Back



A Write Cache Hit – Write Through



A Write Cache Miss



Evaluating a Cache

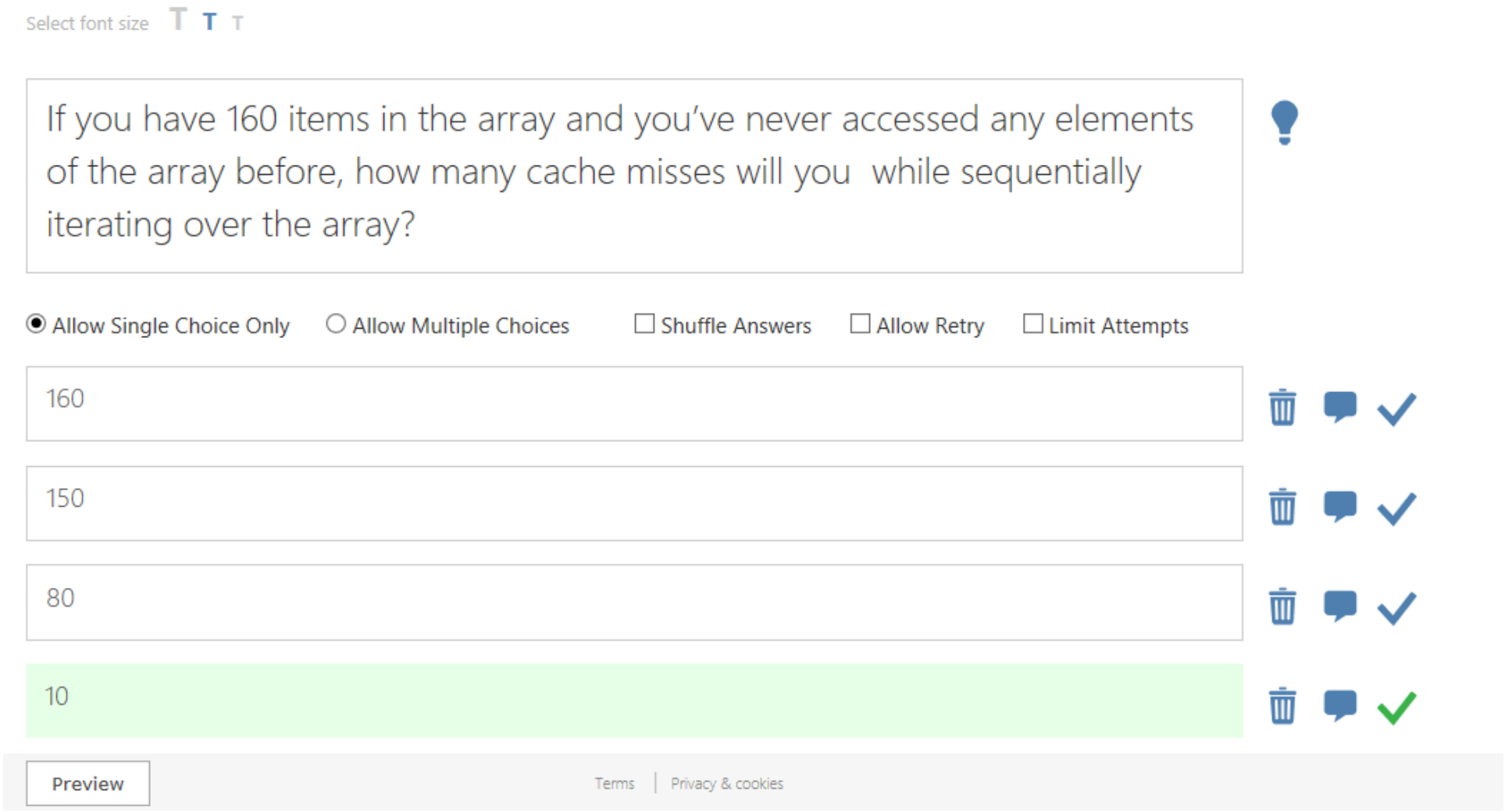
- Hits are much better than misses!
- We measure the efficiency of a cache in terms of its cache hit rate:
  - # cache hits / # cache accesses
  - # cache hits / (# cache hits + # cache misses)
- Example:
  - I access my cache 1000 times and have 400 hits.
    - My cache hit rate is 400/1000 = 40%
- Good performance requires high cache hit rates.

Computing Average Access Time

- Let's say that it takes 1 time unit to access your cache and 100 time units to access the data source.
  - 10% hit rate: 90% * 100 + 10% * 1 = 90.1 time units/access
  - 50% hit rate: 50% * 100 + 50% * 1 = 50.5 time units/access
  - 90% hit rate: 10% * 100 + 90% * 1 = 10.9 time units/access
  - 99% hit rate: 1% * 100 + 99% * 1 = 1.99 time units/access
- The ratio between access time for the cache and the data source dictates just how good your hit rate needs to be to obtain good performance from a cache.

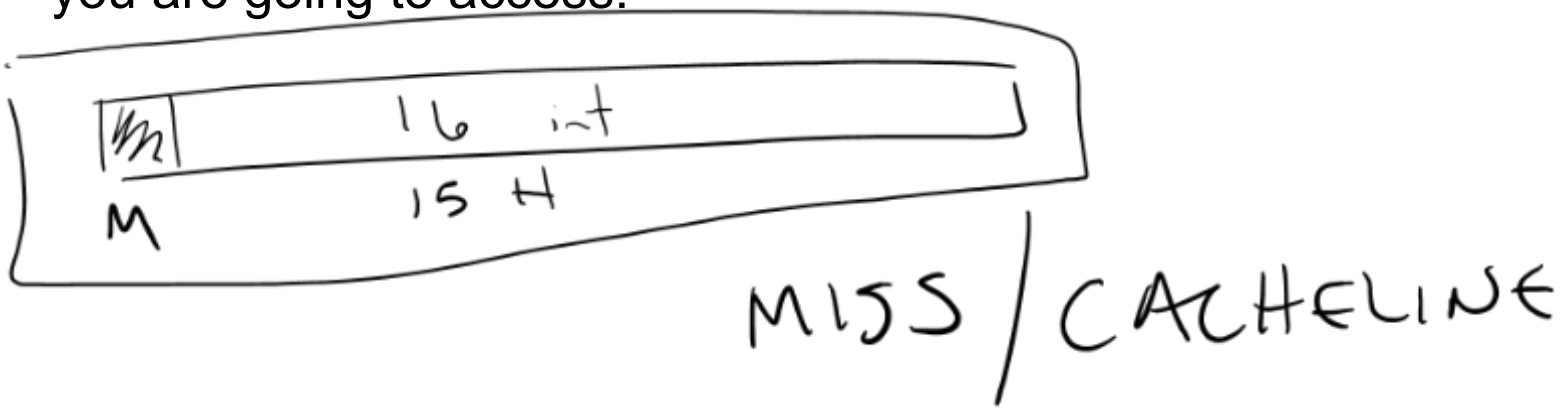
More than one way to get a hit …

- If you touch the same item more than once, you get a hit, but there is another way to get a hit.
- Think about the fact that your cache is organized in blocks…
- Consider this:
  - Let's say you are accessing an array of 4-byte integers.
  - A cache line is 64 bytes. (16 integers)
- Here is the question:
  - Let's say that you have 160 items in the array and you've never accessed it before, how many cache misses will you take?


Select font size T T T
If you have 160 items in the array and you've never accessed any elements of the array before, how many cache misses will you while sequentially iterating over the array?
Allow Single Choice Only
Allow Multiple Choices
Shuffle Answers
Allow Retry
Limit Attempts
160
150
80
10
Preview
Terms
Privacy & cookies

How many cache misses?

- The answer is about 10!
- Why?
  - When you miss on the first array element, the cache will bring in an entire block (called a cache line in hardware).
  - And since arrays are laid out contiguously, the rest of the values in the cache line just happen to be the next elements you are going to access!



Wrapping Up

- Caches have some number of slots.
- A cache block occupies a slot.
- The associativity of a cache tells you in which slot(s) a block can reside.
- If the slot a cache block needs is optimized, the eviction policy determines which block gets evicted.
- We measure cache efficiency in terms of hit rate.
- The miss rate is just 1 – hit rate.
- If writes go all the way through to the data source, we call that a write-through cache.
- If writes can stay in the cache, we call the written cache blocks dirty and the cache is write-back.