medium.datadriveninvestor.com

# System Design Interview Prep: All you need to know about Caching

*Wasek Rahman*

8-10 minutes

---

Here is the one stop guide on Caching for System Design Interviews. You don’t have to go through hours of videos.

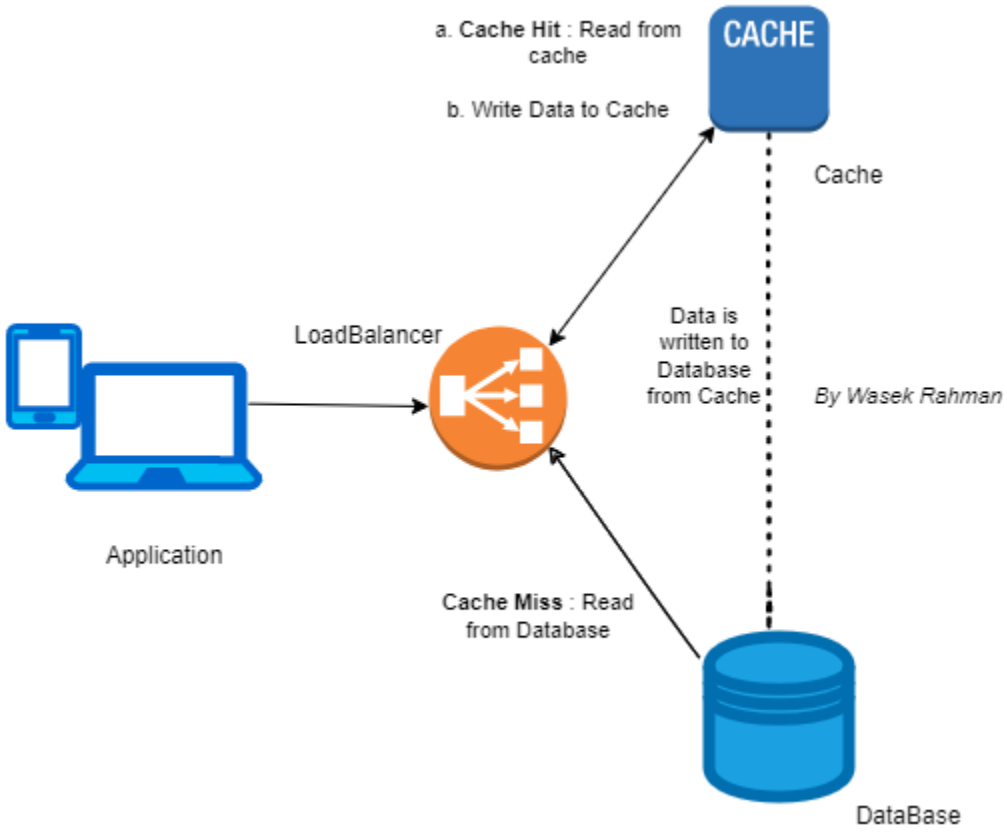



Image by Wasek Rahman

Every system design interview has a few common concepts that you must know before giving the interview. One of those concepts

is Caching.

Caching is a concept that is used almost everywhere in the tech world. But we will focus on caching in web applications for interview preparation purposes.

## What do you mean by caching?

Caching is the process of storing the results of a request at a different location other than the main storage unit such as the Database.

Basically, the cache is a temporary storage for files and data such that it's faster to access this data from this new location.

In simple terms, Caching allows you to efficiently reuse previously retrieved or computed data.

## How does this work?

It's quite simple actually.

Let's look at a normal scenario.

1. User A hits a URL through the browser.
2. Your webpage loads for a couple of seconds.
3. Data from the database is fetched and displayed to your browser.

Now if the User A reloads the same page, it will do the same. It will repeat steps 1 to 3.

Here's the question to ask yourself. Does it seem efficient and logical to keep fetching the same data from the database? I don't think so. We can definitely do better.

This is where we can apply caching.

So let's see how it would be like.

1. User A hits a URL through the browser.
2. Your webpage loads for a couple of seconds.
3. The server looks in your cache for the data.
4. If data is not found; Data from the database is fetched and stored in cache.
5. If data is found; Data from the cache is directly fetched.
6. The data is displayed to your browser.

Now if User A is loading the webpage for the first time, it will take a couple of seconds to load as the data will be fetched from the database. However, once the data is loaded, no matter how many times you reload or go to that same URL, it will load much faster. This is because the server no longer has to fetch the data from the database and instead, the results are retrieved from the cache memory.

## Caching Benefits

Caching benefits everyone. The developers/owner of the web application as well as the users.

**Lower network costs:** Data can be cached in different network points. The closer the data is to the User, the lower the cost it will take to retrieve that data.

**Faster loading:** Caching enables data to be retrieved faster because it is no longer necessary to go beyond the caching memory. Caches maintained close to the user, like the browser cache, can make this retrieval nearly instantaneous.

**Data Availability:** With certain policies, caching can be used to serve data to end users even when it may be unavailable, such as if the database fails for a short period of time.

**Reduce the Load on the Backend:** Offloading the same request traffic from the main server to caching server would reduce the backend load.

## Eviction Policy

Unlike databases, a cache memory will have limited space. After all, it is a temporary storage system. You can expect it to hold 4GB of data. In other words, what do we do when our cache memory is full? This is where eviction policies come into place.

A cache eviction algorithm is a way of deciding which element to evict when the cache is full.

The top two eviction policies you should be aware of for your interview is —

**Least Recently Used (LRU):** It defines the policy to evict elements from the cache to make room for new elements when the cache is full. It does that by **discarding the least recently** used items first. Say when you search for something in Google, it will be stored in your cache as the system will expect you to hit the same URL again. However, after visiting a few other URLs, the ones that are most recently used will remain in the cache. In such use cases, the cached data that is not used very recently will be removed to save space.

**Advantages:**

1. Is nearest to the most optimal algorithm
2. Selects and removes elements that are not used recently

**Disadvantages:**

1. Only mild success rate since often it is more important how often an element was accessed than when it was last accessed

**Least Frequently Used (LFU):** It defines the policy to evict elements from the cache to make room for new elements when the cache is full. It does that by **discarding the least frequently** used items first. Say when you search for something in Google, it will be stored in your cache as the system will expect you to hit the same URL again. However, after visiting a few other URLs, the ones that are most frequently used will remain in the cache. In such use cases, the cached data that is not visited recently will be removed to save space.

**Advantages:**

1. Takes age of the element into account
2. Takes reference frequency of the element into account
3. Works better under high load when quickly a lot of elements is requested (less false eviction)

**Disadvantages:**

1. A frequently accessed element will only be evicted after lots of misses
2. More important to have invalidation on elements that can change

**Still confused?**

**LFU** only cares about the most frequently used elements. So, if you visit a same URL 100 times 5 days ago, and you visited different other URLs after that but less than 100 times, the one you visited 100 times will remain in the cache.

**LRU** only cares about the recently accessed elements. So, it

doesn't matter how many times you visited a URL before. It will only store the one you most recently visited and remove the old visits. Google Chrome uses this mechanism. Check your history and see if you can view 6 months old data. They are all removed.

## What are the different types of caching?

The popular caching types that you should be aware of for your interviews are :

**Application server cache** We can cache the data directly in the Application Layer. Every time a request is made to the service, it will return local, cached data quickly if it exists. If it is not in the cache, it will query the data from the database.

**Global caches** In Global Caches, the same single cache space is used for all the nodes. Each of the application nodes queries the cache in the same way as a local one would be.

**Distributed cache** The cache is usually broken up using a consistent hashing algorithm, and each of its nodes owns part of the cached data. If a requesting node is searching for a certain piece of data, it can easily use the hashing function to locate information from the distributed cache to decide if the data is available.

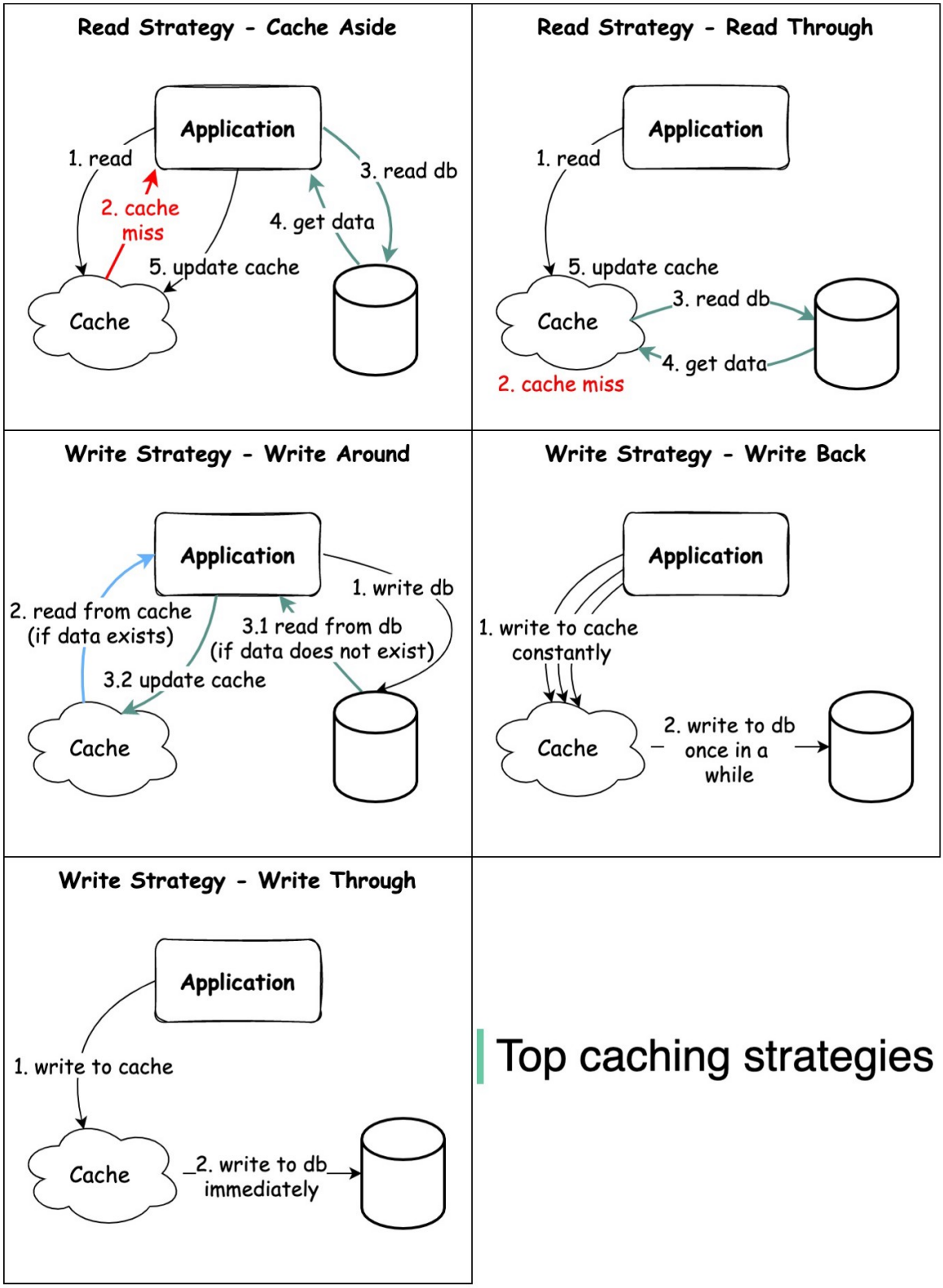
## What are the top cache strategies?

Read data from the system:

1. Cache aside
2. Read through

Write data to the system:

1. Write around
2. Write back
3. Write through




## Top caching strategies

Source : www.twitter.com/alexxubyte

## Different data types use different Cache

Different caching technology serves well to different data types due to their internal design. As a result, it is highly important to understand what data type you are caching and what method would fit that data type best.

***Key-Value Pair:*** Suppose you have a URL which gives you details of items. Let's say ***xyz.com/item/1*** will return a JSON from the database such as

```
{ 'item_id' : '1',
  'name': 'laptop',
  'price' : 350
}
```

So in order to cache this data, we can use a key-value store like

```
{ '1' : { 'item_id' : '1',
  'name': 'laptop',
  'price' : 350 }
}
```

Now, everytime the URL will hit ***/item/1*** **,** it will look for '1' in the cache and return that data.

Storing simple string key to string value is almost supported by any cache. Example: Redis, Memcached, Hazelcast, Couchbase etc.

***Static File Cache:*** Suitable to cache images, files, static files like — css or javascript files. Example: Akamai CDN, Memcached to a certain extent.
If you want to see an example of how and where CDN is used,

please check this writing —

***Object Store:*** Suitable to store unmodifiable objects like HTTP Response Object, database result set, rendered html page etc. Example: Memcached.

***In-Memory Caching:*** Suitable to store any key value or objects directly accessible through run time memory in the same node. Example: HashMap, Guava Cache etc, Hibernate & MySQL query caching.