Scaling Your App with Backend Caching Strategies

Muhammad Attia


Allow myself to introduce...

Who Am I ?

Muhammad Attia

- Lead Software Engineer @Elm
- Mentor @Manara & AdpList

Previously

- Senior Software Engineer @Careem
- Senior / Lead Software Engineer @_VOIS
- Senior Software Engineer @Andela
- Software Engineer @Asset Technology Group

Contact With Me :-

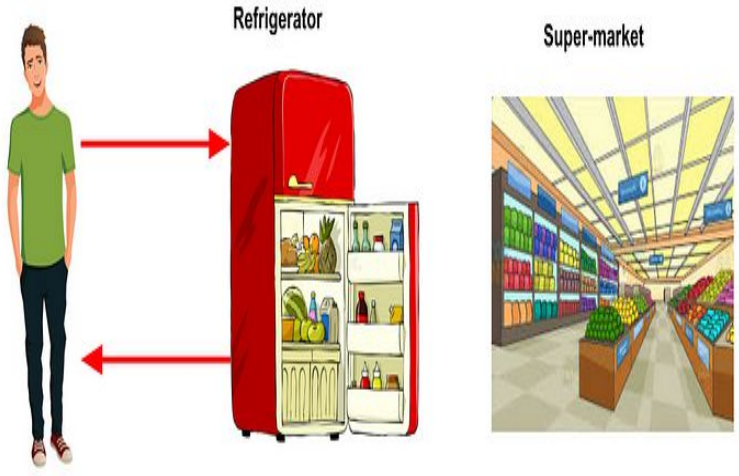
Agenda

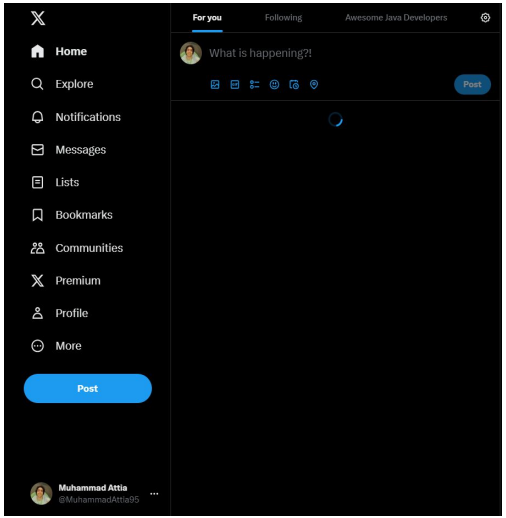
- Introduction
- What is a cache?
- Why do we need a cache?
- Where does the cache exist?
- When to use cache?
- Backend Caching Strategies
- Demo Time
- Final thoughts

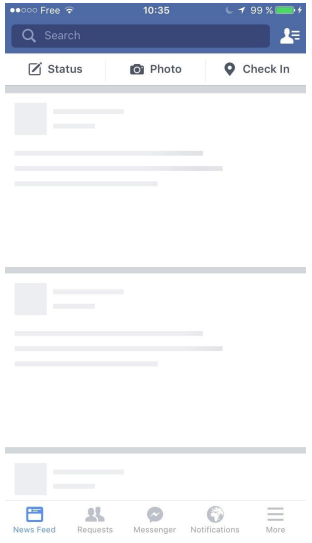
INTRODUCTION

Introduction

- Imagine you cooking daily and needing different ingredients, But imagine having to go to the supermarket every day for these supplies. That's both a hassle and a time consuming.
- Instead of daily supermarket trips, Caching is like having a stocked fridge to save you from frequent trips to the supermarket for your favorite ingredients.
- Have you ever wondered why text loads before high-quality images on a slow internet connection, and why well-visited sites load faster than new ones? Do you know why this happens? **The answer is Caching.**







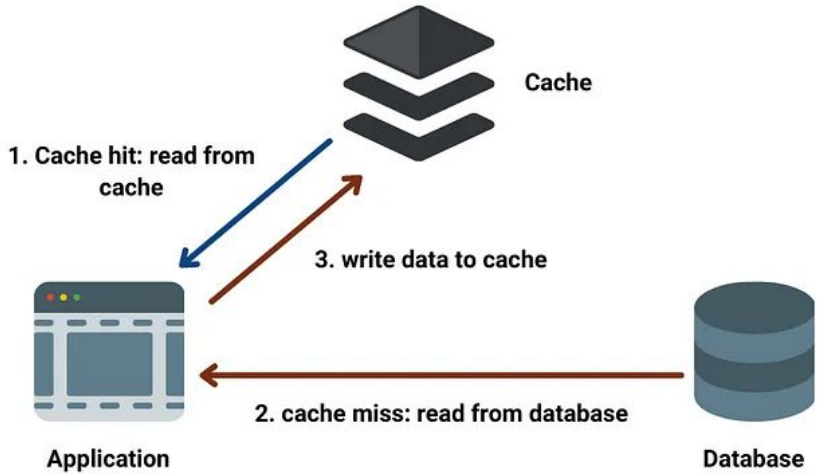
What is cache ?

What is cache ?

- **Caching** is the process of temporarily storing copies of data, so it can be accessed more quickly.
- Technique for improving performance, reducing cost, and increasing the scalability of a system by storing frequently accessed data in fast data-storage component called cache!

How Caching Works?

- The application reads the data directly from the cache
- If the data is in the cache (cache hit),it's returned to the client .
- Otherwise (cache miss), the application retrieves the data from th data and then writes it to the cache.



Why do we need cache ?

Why do we need cache ?

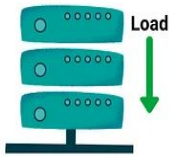
Caching is an essential technique for optimizing the performance, scalability, and availability applications in many different areas, including web applications, databases, media streaming, e-commerce, gaming, cloud computing, and mobile applications.

By caching frequently accessed data closer to the user or application:-

- **Improve Application Performance.**
- **Scalability improved.**
- **Availability enhanced.**
- **Reduce load on backend / data Store which lead to Costs optimization (cache save a cash $$$)**

Improve Application Performance

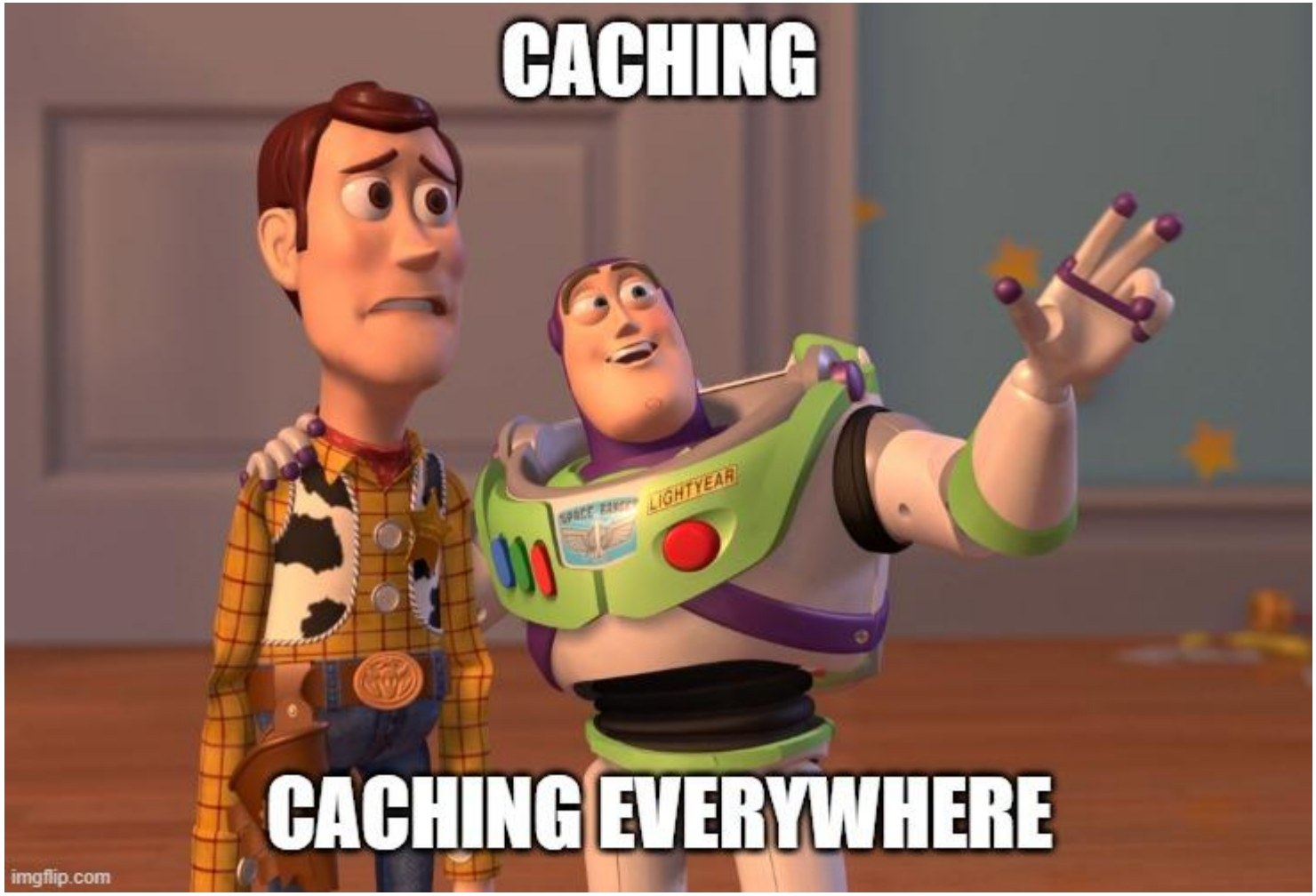
Reduce Database Cost

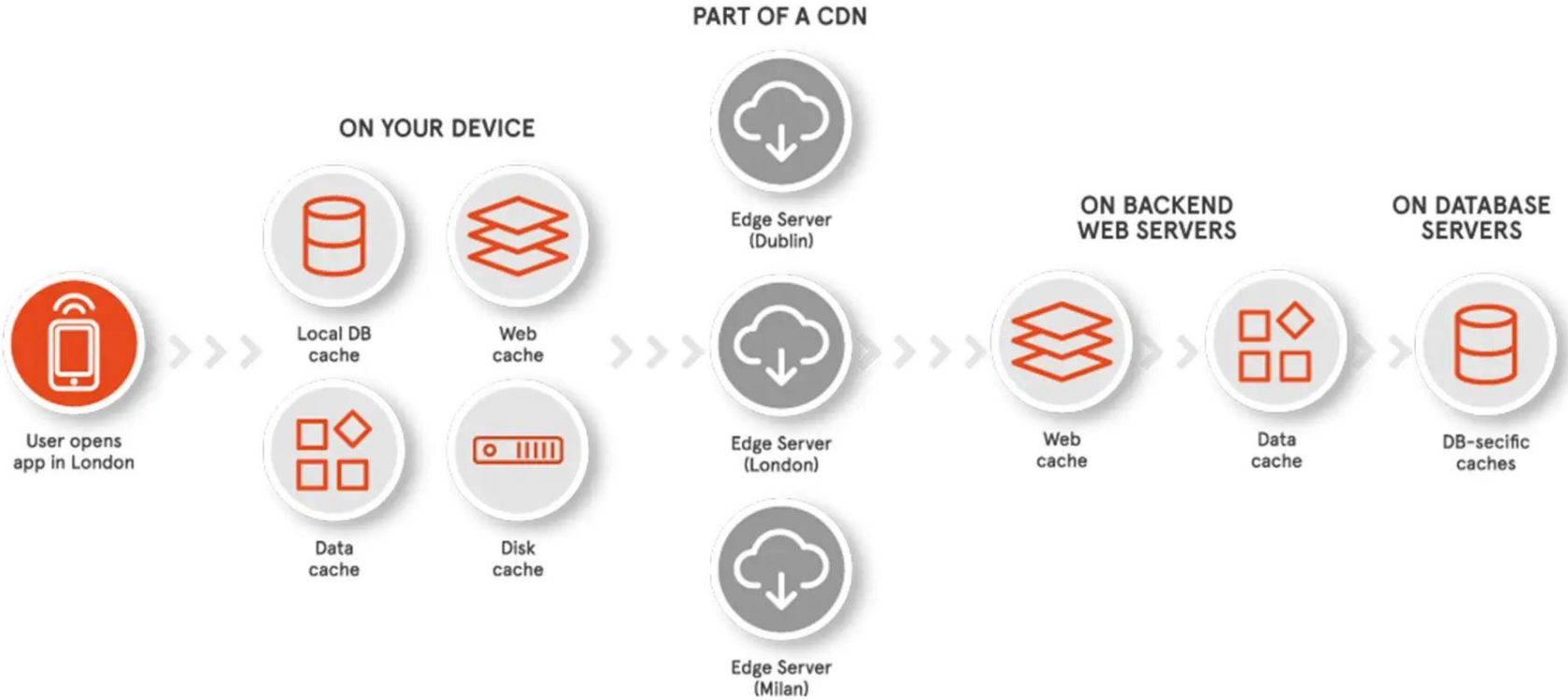


Reduce load on Backend

Increase Read Throughput (IOPS)

Where the cache exist ?


CACHING
LIGHTYEAR
CACHING EVERYWHERE
imgflip.com


PART OF A CDN
ON YOUR DEVICE
ON BACKEND WEB SERVERS
ON DATABASE SERVERS
User opens app in London
Local DB cache
Web cache
Data cache
Disk cache
Edge Server (Dublin)
Edge Server (London)
Edge Server (Milan)
Web cache
Data cache
DB-secific caches

When to use cache

When to use cache?

- Frequently used data.
- Data calculated from costly calculations.
- Data fetched from potential resource bottlenecks
- Scenarios that need strict performance SLAs.

Backend Caching Strategies

Backend Caching Strategies

🔍 **Read Strategies:**

- Cache Aside (Lazy Loading)
- Read Through (Eager Loading)
- Refresh Ahead

📝 **Write Strategies:**

- Write Through (Synchronous)
- Write behind/back (Asynchronous)
- Write Around

Read Strategies

Cache Aside

How it works:

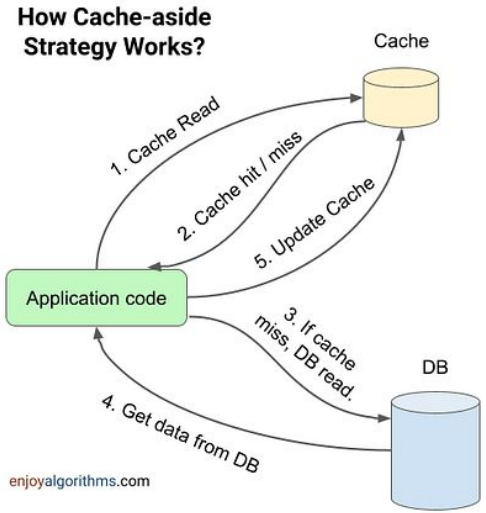
- The application reads the data directly from the cache
- If the data is in the cache (hit),it's returned to the client .
- Otherwise (cache miss), the application retrieves the data from the database and then writes it to the cache.

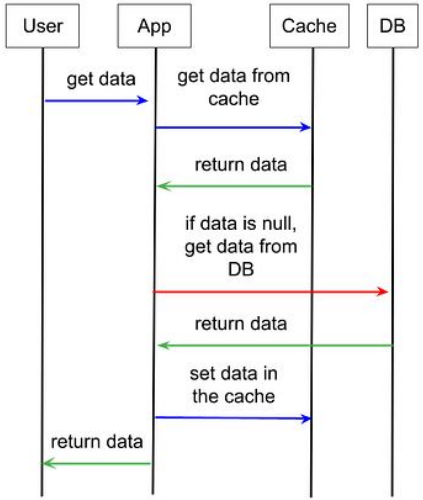
Pros:

- adapt for read-heavy workloads.
- Systems using cache-aside are resilient to cache failures.
- The cache contains only data that the application actually requests, which helps keep the cache size cost-effective.
- database and cache are decoupled and can handle different data models.

Cons:-

- some overhead is added to the initial response time because additional round trips to the cache and database are needed.
- Data might become stale if someone updates the database without writing to the cache. (Hence, Cache-aside is usually used alongside other strategies)





Read Through

How it works:

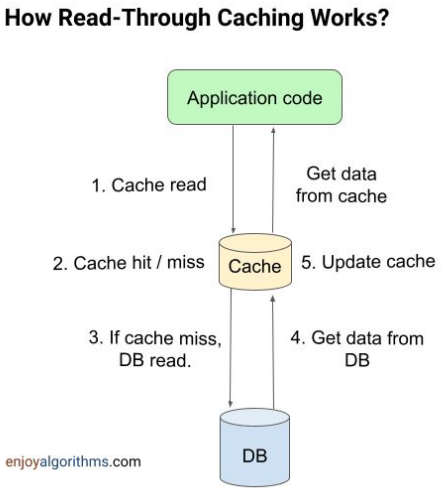
- Like read aside, but the application interacts only with the cache
- Check the cache, retrieve from the database, add data into the cache and return it to application

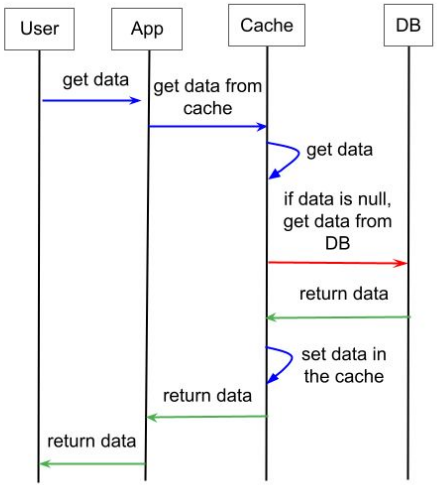
Pros:

- read-through cache provides an automatic way to ensure data consistency.
- simplify application code

Cons:

- Clearly, since every time you need to retrieve some data you have to pass through the cache, we've just introduced a Single Point of Failure: if for some reason the Cache is unavailable, your application won't be able to retrieve such data from the DB.
- require a plugin for getting the data.
- We've also introduced coupling between the Cache and the DB.





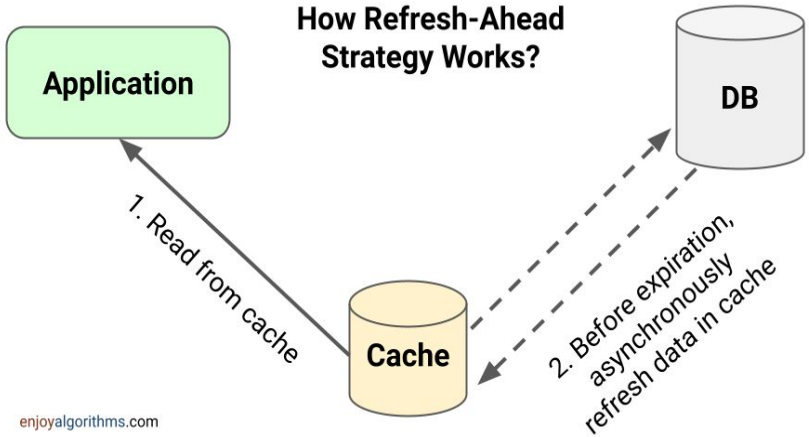
Refresh Ahead

- In cache-aside or read-through patterns, when cached data expires (TTL).it's reloaded from the database on the next read request, causing a cache miss and increased read latency.

How it works?

- The goal of refresh-ahead pattern is to configure cache to asynchronously reload (refresh) the most recent version of data from database before the expiration (or close to its expiration).

Use Case: prediction apps like weather app



Write Strategies

Write Through

How it works:

- The application writes data to the cache.
- The cache write immediately the data to the database.

Usage:

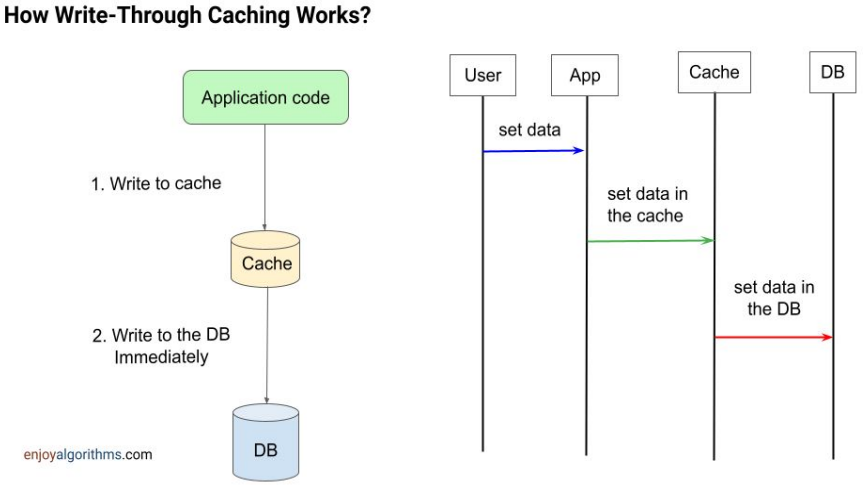
- When data consistency is critical.

Pros:

- Since all data writings pass through the cache, we won't have stale data,
- our application will always be in the most updated state.
- no data loss in case of cache crash.

Cons:

- higher latency for write operations.
- Coupling between cache and datasource, If the cache fails or is unavailable, we will lose the data we're trying to write.



Write Back/Behind

How it works:

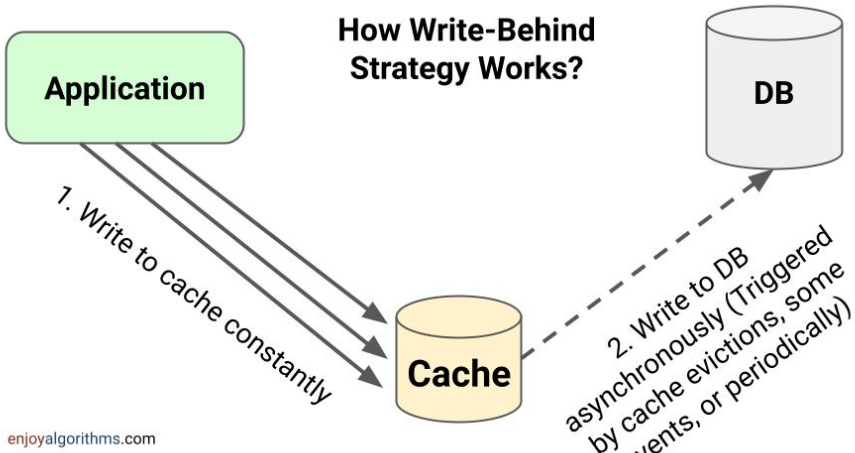
- Writes to cache first, async write to DB later.
- The application writes data to the cache
- The cache writes the data to the database asynchronously

Pros:

- adapt to write-heavy workloads (low latency).
- This strategy is ideal when write performance is more critical than data consistency, and the risk of data loss is acceptable.
- less load on the database.
- tolerant to database failures

Cons:

- data can be lost if the cache crash.



Write Around

How it works:

- Writes bypass the cache and go directly to the DB.
- The application writes data directly to the database .
- Only the read data get into the cache.

Usage:

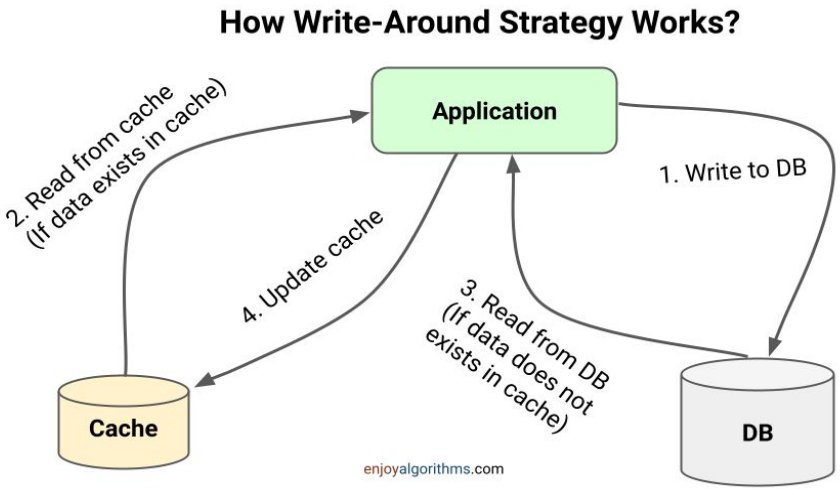
- When written data won't immediately be read back from cache.
- If you have data written once and read rarely (i.e. real time logs), write around could be a good fit.

Pros:

- adapt for data written once and read less frequently (the cache store only re-reads data)
- database and cache are decoupled.

Cons:

- reading recently written data gives a miss (high latency).
- Less consistency.



Real-Life Use Case:

- There always need to combine read and write strategies to achieve the business need.
  - **Cache Aside + Write Through / Cache Aside + Write Around** : This ensures consistent cache/DB sync while allowing read heavy.
  - **Read Through + Write Back**: This abstracts the DB and handles heavy write traffic well by delaying sync. However, it risks larger data loss if the cache goes down before syncing the buffered writes to the database (we can use **kafka** to avoid this).

- **Depending on the nature of the application we develop, we can ask the following questions before choosing a caching Strategy:**
  - **How is the load on read vs write in the system?** Is the system read heavy or write heavy?
  - **How often does the data need to be recent?** Freshness of data
  - **How I will invalidate cached data when it becomes outdated?**. Consider strategies such as time-based TTL, manual invalidation, or event-based invalidation triggered by data changes. This is the most challenging and important part of implementing caching successfully.
    “There are only two hard things in Computer Science: cache invalidation and naming things”

  - **What if the cache is full ?** Which eviction policies I will use to handle cache capacity limitations. Common strategies include Least Recently Used (LRU), Least Frequently Used (LFU), and First In First Out (FIFO) based eviction.

Demo Time

Final thoughts

Final thoughts

- Caching is your system's invisible hero, providing much-needed speed while reducing load and Cost. But it's not a one-size-fits-all solution.
- Each application might need a different caching strategy based on its unique requirements.
- Caching is not only a method for architecture and design, but it's also a general idea for solving performance problems.
- Like any technology, caching has its pros and cons.

  Pros:
  - **Speed**: Caching makes your system faster by reducing data fetching time.
  - **Reduced Server Load and Cost**: Caching reduces the load on your database or primary servers which reduce the Cost (Cach Save Cash).
  - **User Experience**: Quick response times lead to happier users!

  Cons:
  - **Complexity**: Implementing caching adds an extra layer of complexity to system design.
  - **Cache Invalidation:** Cache data can become outdated, leading to data inconsistency, Determining when to refresh or clear cache can be challenging.
- **When to not use cache**
  - **There is area of improvement, for example the original source of data is slow** (e.g. a query that does complex JOINs in a relational database.)
  - **The data doesn't need to change for each request** (e.g. caching real-time sensor data that your car needs when it's in the self-driving mode or live medical data from patients, checkout and payment not good ideas.).

Resources

1. https://aws.amazon.com/caching/
2. https://d0.awsstatic.com/whitepapers/Database/database-caching-strategies-using-redis.pdf
3. https://codeahoy.com/2017/08/11/caching-strategies-and-how-to-choose-the-right-one/
4. https://www.enjoyalgorithms.com/blog/caching-system-design-concept
5. https://blog.bytebytego.com/p/ep54-cache-systems-every-developer
6. https://zubialevich.blogspot.com/2018/08/caching-strategies.html

Thank you