<https://github.com/ashishps1/awesome-system-design-resources>

<https://github.com/DreamOfTheRedChamber/system-design-interviews/tree/master>

<https://github.com/donnemartin/system-design-primer>

1. 𝗙𝘂𝗻𝗱𝗮𝗺𝗲𝗻𝘁𝗮𝗹𝘀 𝗼𝗳 𝗦𝘆𝘀𝘁𝗲𝗺 𝗗𝗲𝘀𝗶𝗴𝗻
2. Scalability
3. Fault tolerance
4. Load balancing
5. Availability
6. Consistency
7. Latency
8. Throughput
9. Partition Tolerance
10. CAP Theorem
11. ACID Properties
12. Network Protocols
13. Monitoring and Analytics
14. Security
15. **Content Delivery Network (CDN): https://lnkd.in/gjJrEJeH**  
    [A Content Delivery Network (CDN) is a geographically distributed group of servers that cache content close to end users](https://www.cloudflare.com/learning/cdn/what-is-a-cdn/). [It allows for the quick transfer of assets needed for loading Internet content, including HTML pages, JavaScript files, stylesheets, images, and videos](https://www.cloudflare.com/learning/cdn/what-is-a-cdn/). [The majority of web traffic today is served through CDNs, including traffic from major sites like Facebook, Netflix, and Amazon](https://www.cloudflare.com/learning/cdn/what-is-a-cdn/).

A CDN is not the same as a web host. [While a CDN does not host content and can’t replace the need for proper web hosting, it helps cache content at the network edge, which improves website performance1](https://www.cloudflare.com/learning/cdn/what-is-a-cdn/). [Many websites opt for CDNs to reduce hosting bandwidth, help prevent interruptions in service, and improve security1](https://www.cloudflare.com/learning/cdn/what-is-a-cdn/).

[The primary benefits of using a CDN can be broken down into four components](https://www.cloudflare.com/learning/cdn/what-is-a-cdn/)

1. Improving website load times
2. Reducing bandwidth costs
3. Increasing content availability and redundancy
4. Improving website security

[A CDN works by caching content (such as images, videos, or webpages) in proxy servers that are located closer to end users than origin servers](https://www.cloudflare.com/learning/cdn/what-is-caching/). [This makes the content delivery on the web faster and more efficient](https://www.cloudflare.com/learning/cdn/cdn-benefits/).

CDNs take the burden of serving static media off of your application servers (which are typically optimized for serving dynamic pages rather than static media) and provide geographic distribution. Overall, your static assets will load more quickly and with less strain on your servers (but a new strain of business expense).

In a typical CDN setup, a request will first ask your CDN for a piece of static media, the CDN will serve that content if it has it locally available (HTTP headers are used for configuring how the CDN caches a given piece of content). If it is not available, the CDN will query your servers for the file and then cache it locally and serve it to the requesting user (in this configuration they are acting as a read-through cache).

Can you design a CDN?

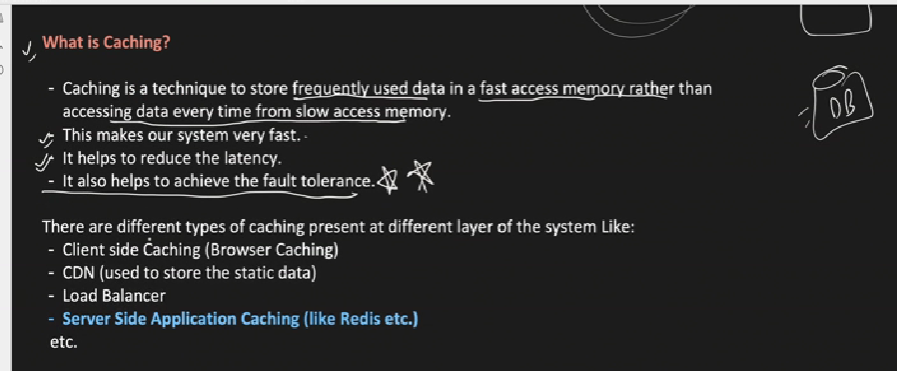
<https://www.akamai.com/glossary/what-is-a-cdn#:~:text=Akamai's%20CDN%20includes%20origin%20servers,content%20stored%20on%20original%20servers>.

1. **Caching:** [**https://medium.com/must-know-computer-science/system-design-caching-acbd1b02ca01**](https://medium.com/must-know-computer-science/system-design-caching-acbd1b02ca01)

[**https://massivetechinterview.blogspot.com/2017/04/system-design-cache.html**](https://massivetechinterview.blogspot.com/2017/04/system-design-cache.html)

[**https://www.udemy.com/course/system\_design\_lld\_hld/learn/lecture/41910268#overview**](https://www.udemy.com/course/system_design_lld_hld/learn/lecture/41910268#overview)

A cache is like short-term memory which has a limited amount of space. It is typically faster than the original data source.

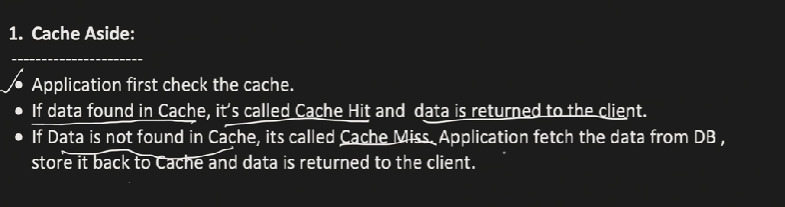


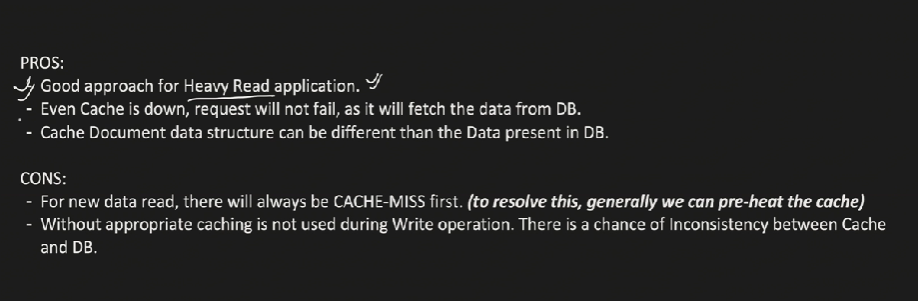
Fast Access Memory: - RAM

Slow Access Memory: - reading from Hard disk, Database.

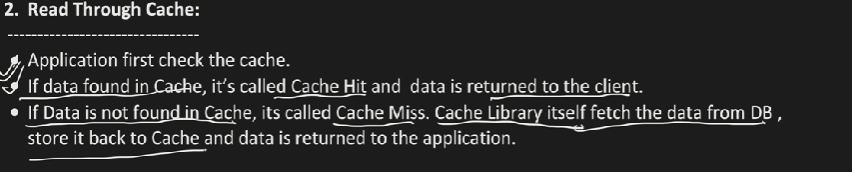
* Caching consists of  
  1. Pre-calculating results (e.g. the number of visits from each referring domain for the previous day)  
  2. Pre-generating expensive indexes (e.g. suggested stories based on a user’s click history)  
  3. Storing copies of frequently accessed data in a faster backend (e.g. Memcache instead of PostgreSQL.

**Cache Strategies**

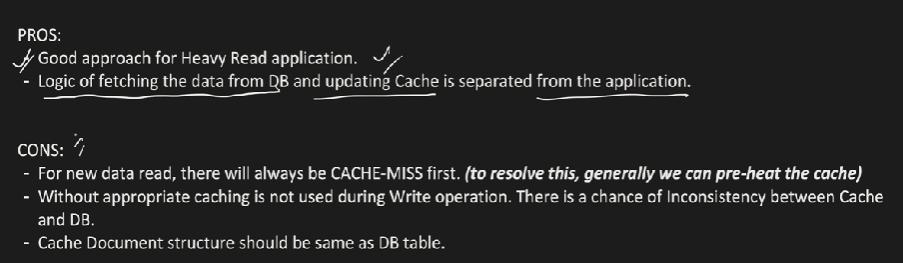
* 1. **Cache Aside: - **

****

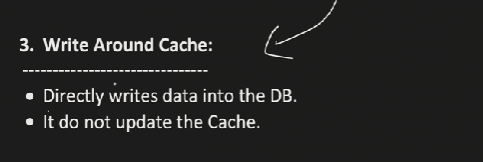
* 1. **Read Through Cache: -**

****

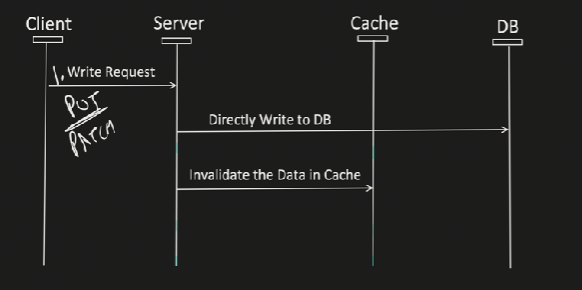
In read through cache if there is a cache miss, cache service itself is responsible for writing the data to DB, but in cache Aside app server is responsible for writing the cache miss data to DB.

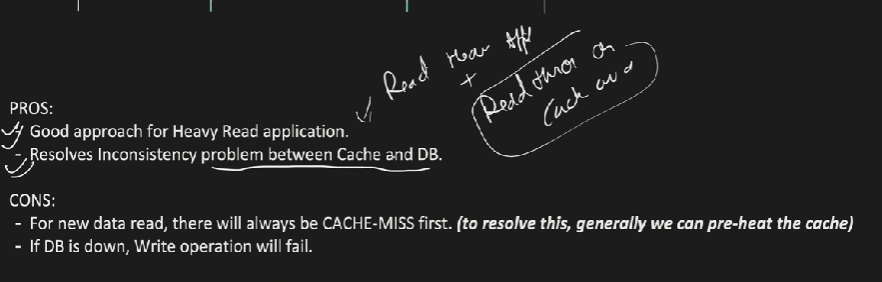
****

* 1. **Write Around Cache**

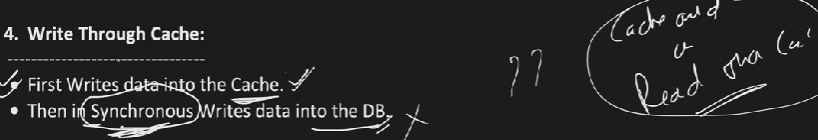
****

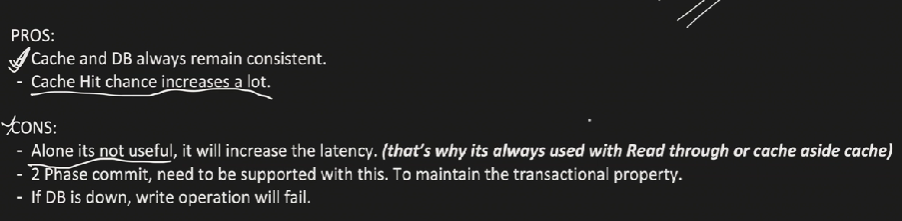
It doesn’t update the cache but marks the cache as cache invalidate.



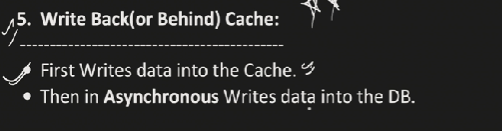


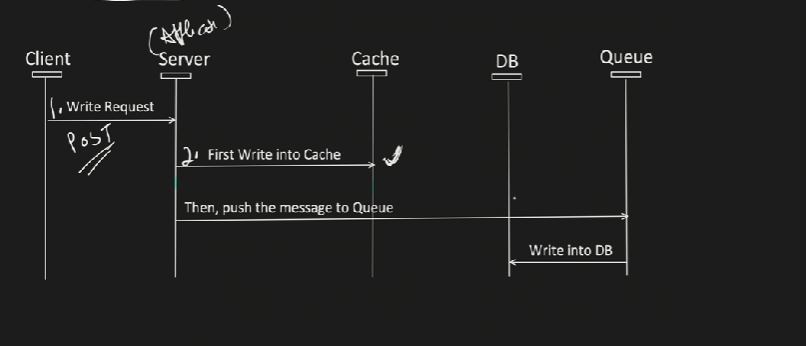
**Write Through Cache**

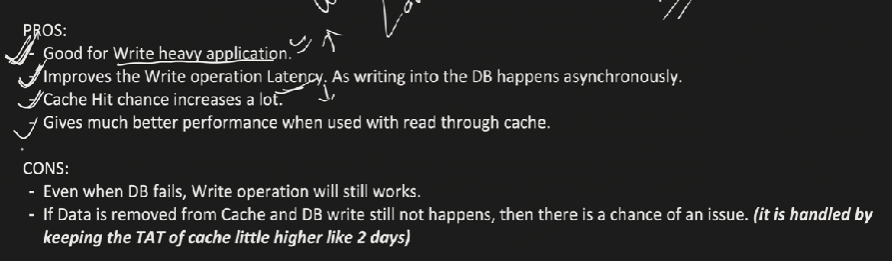


****

**Write Back Cache: -**

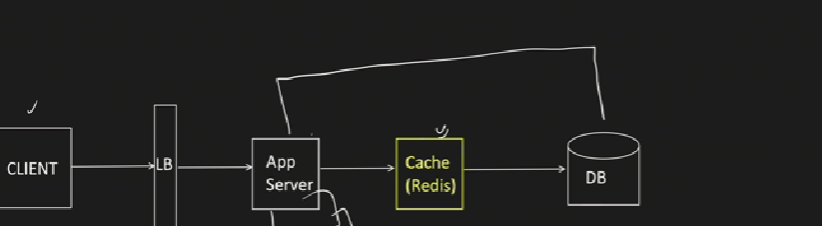
****

****

****

**Cache Types: -**

**Server-Side Cache**



Why do we need Distributed Caching?

1. To avoid Single Point of Failure.
2. To mitigate Scalability issues.

Distributed Caching can be achieved using Consistent Hashing technique.

𝟮) 𝗨𝗻𝗱𝗲𝗿𝘀𝘁𝗮𝗻𝗱 𝗰𝗼𝗺𝗺𝗼𝗻 𝗱𝗲𝘀𝗶𝗴𝗻 𝗽𝗮𝘁𝘁𝗲𝗿𝗻𝘀  
Microservices  
Sharding  
Event sourcing  
Circuit Breaker  
Reverse proxy  
Backpressure  
CQRS (Command Query Responsibility Segregation)  
Object Pool  
  
𝟯) 𝗙𝗮𝗺𝗶𝗹𝗶𝗮𝗿𝗶𝘁𝘆 𝘄𝗶𝘁𝗵 𝗱𝗶𝗳𝗳𝗲𝗿𝗲𝗻𝘁 𝘁𝘆𝗽𝗲𝘀 𝗼𝗳 𝗱𝗮𝘁𝗮𝗯𝗮𝘀𝗲𝘀  
Relational databases  
NoSQL databases  
Distributed key-value stores  
Document databases  
Graph databases  
Time-series databases  
  
𝟰) 𝗙𝗮𝗺𝗶𝗹𝗶𝗮𝗿𝗶𝘁𝘆 𝘄𝗶𝘁𝗵 𝗱𝗶𝗳𝗳𝗲𝗿𝗲𝗻𝘁 𝘁𝘆𝗽𝗲𝘀 𝗼𝗳 𝗱𝗶𝘀𝘁𝗿𝗶𝗯𝘂𝘁𝗲𝗱 𝘀𝘆𝘀𝘁𝗲𝗺𝘀 𝗮𝗻𝗱 𝗮𝗹𝗴𝗼𝗿𝗶𝘁𝗵𝗺𝘀  
Merkle Tree  
Consistent Hashing  
Read Repair  
Gossip Protocol  
Bloom Filter  
Heartbeat  
CAP and PACELC Theorems  
  
𝟱) 𝗟𝗲𝗮𝗿𝗻 𝗵𝗼𝘄 𝘁𝗼 𝗰𝗼𝗺𝗺𝘂𝗻𝗶𝗰𝗮𝘁𝗲 𝘆𝗼𝘂𝗿 𝘁𝗵𝗼𝘂𝗴𝗵𝘁 𝗽𝗿𝗼𝗰𝗲𝘀𝘀  
Start with the problem statement  
Break down the problem  
Use diagrams and sketches  
Discuss trade-offs and constraints  
Explain your reasoning  
Be prepared to answer questions and provide alternatives  
Be open to feedback  
  
𝟲) 𝗛𝗼𝘄 𝘁𝗼 𝗮𝗻𝘀𝘄𝗲𝗿 𝗮 𝘀𝘆𝘀𝘁𝗲𝗺 𝗱𝗲𝘀𝗶𝗴𝗻 𝗾𝘂𝗲𝘀𝘁𝗶𝗼𝗻 𝗶𝗻 𝗮𝗻 𝗶𝗻𝘁𝗲𝗿𝘃𝗶𝗲𝘄?  
Step 1: Requirements clarification  
Step 2: Back-of-the-envelope estimation  
Step 3: System interface definition  
Step 4: Defining the data model  
Step 5: High-level design  
Step 6: Detailed design  
Step 7: Identifying and resolving bottlenecks  
  
𝟳) 𝗧𝗼𝗽 𝗦𝘆𝘀𝘁𝗲𝗺 𝗗𝗲𝘀𝗶𝗴𝗻 𝗜𝗻𝘁𝗲𝗿𝘃𝗶𝗲𝘄 𝗤𝘂𝗲𝘀𝘁𝗶𝗼𝗻𝘀  
Design Facebook Messenger  
Design Youtube  
Design Facebook’s Newsfeed  
Design an API Rate Limiter  
Design Twitter  
Design Dropbox or Google Drive  
Design a Web Crawler  
Design Twitter Search  
Design a URL Shortening service like TinyURL  
Design Instagram  
Designing Yelp or Nearby Friends  
Design Ticketmaster

3) Distributed Caching: https://lnkd.in/g7WKydNg  
  
4) Latency vs Throughput: https://lnkd.in/g\_amhAtN  
  
5) CAP Theorem: https://lnkd.in/g3hmVamx  
  
6) Load Balancing: https://lnkd.in/gQaa8sXK  
  
7) ACID Transactions: https://lnkd.in/gMe2JqaF  
  
8) SQL vs NoSQL: https://lnkd.in/g3WC\_yxn  
  
9) Consistent Hashing: https://lnkd.in/gd3eAQKA  
  
10) Database Index: https://lnkd.in/gCeshYVt  
  
11) Rate Limiting: https://lnkd.in/gWsTDR3m  
  
12) Microservices Architecture: https://lnkd.in/gFXUrz\_T  
  
13) Strong vs Eventual Consistency: https://lnkd.in/gJ-uXQXZ  
  
14) REST vs RPC: https://lnkd.in/gN\_\_zcAB  
  
15) Batch Processing vs. Stream Processing: https://lnkd.in/g4\_MzM4s  
  
16) HeartBeat: https://lnkd.in/g4x7sMrF  
  
17) Circuit Breaker: https://lnkd.in/gCxyFzKm  
  
18) Idempotency: https://lnkd.in/gPm6EtKJ  
  
19) Database Scaling: https://lnkd.in/gAXpSyWQ  
  
20) Data Replication: https://lnkd.in/gVAJxTpS  
  
21) Data Redundancy: https://lnkd.in/gNN7TF7n  
  
22) Database Sharding: https://lnkd.in/gMqqc6x9  
  
23) Proxy Server: https://lnkd.in/gi8KnKS6  
  
24) Domain Name System (DNS): https://lnkd.in/gkMcZW8V  
  
25) Message Queues: https://lnkd.in/gTzY6uk8  
  
26) WebSockets: https://lnkd.in/g76Gv2KQ  
  
27) Consensus Algorithms: https://lnkd.in/ggc3tFbr  
  
28) Fault Tolerance: https://lnkd.in/dVJ6n3wA  
  
29) Failover: https://lnkd.in/dihZ-cEG  
  
30) Gossip Protocol: https://lnkd.in/gfPMtrJZ  
  
31) API Gateway: https://lnkd.in/gnsJGJaM  
  
32) Distributed Locking: https://lnkd.in/gRxNJwWE  
  
33) Distributed Tracing: https://lnkd.in/d6r5RdXG  
  
34) Bloom Filters: https://lnkd.in/dt4QbSUz  
  
35) Checksum: https://lnkd.in/ghNc5pfn  
  
36) API Design: https://lnkd.in/ghYzrr8q  
  
37) Serverless Architecture: https://lnkd.in/gQNAXKkb  
  
38) Concurrency vs Parallelism: https://lnkd.in/gSKUm2Nh  
  
39) Disaster Recovery: https://lnkd.in/g8rnr3V3  
  
40) Service Discovery: <https://lnkd.in/gjnrYkyF>

**How to Approach System Design**

