Agenda

1. Appserver Layer
2. Caching
3. Invalidation & Eviction
4. Local Caching

support@scaler.com

+91-7351769231

Scale is large

↳ Read Load ⟶ 90% apps, Read-heavy.

↳ Write Load

#reads ≥ 10^* #writes

Caching helps us handle a large Read-Load

#Writes <= #Reads (99.99% cases)

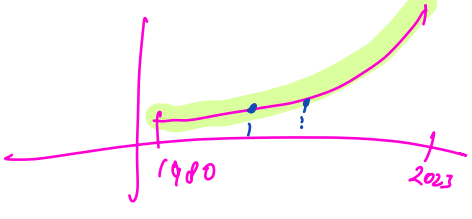
#Writes > #Reads

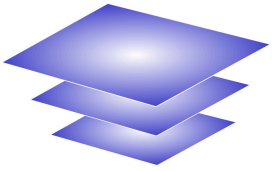
data goes in, but doesn't come out!!

↳ archival

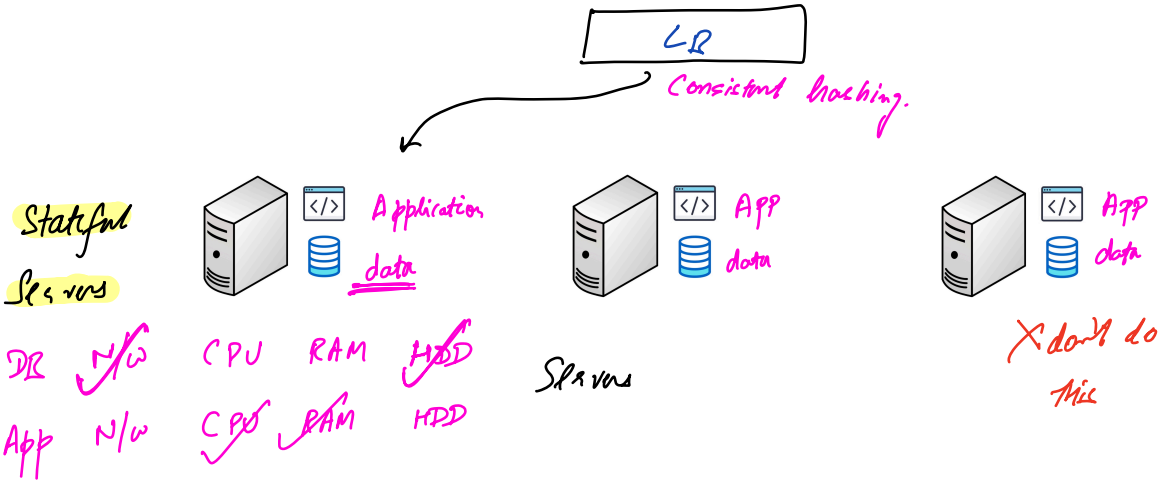
↳ logs

↳ analytics





Appserver Layer



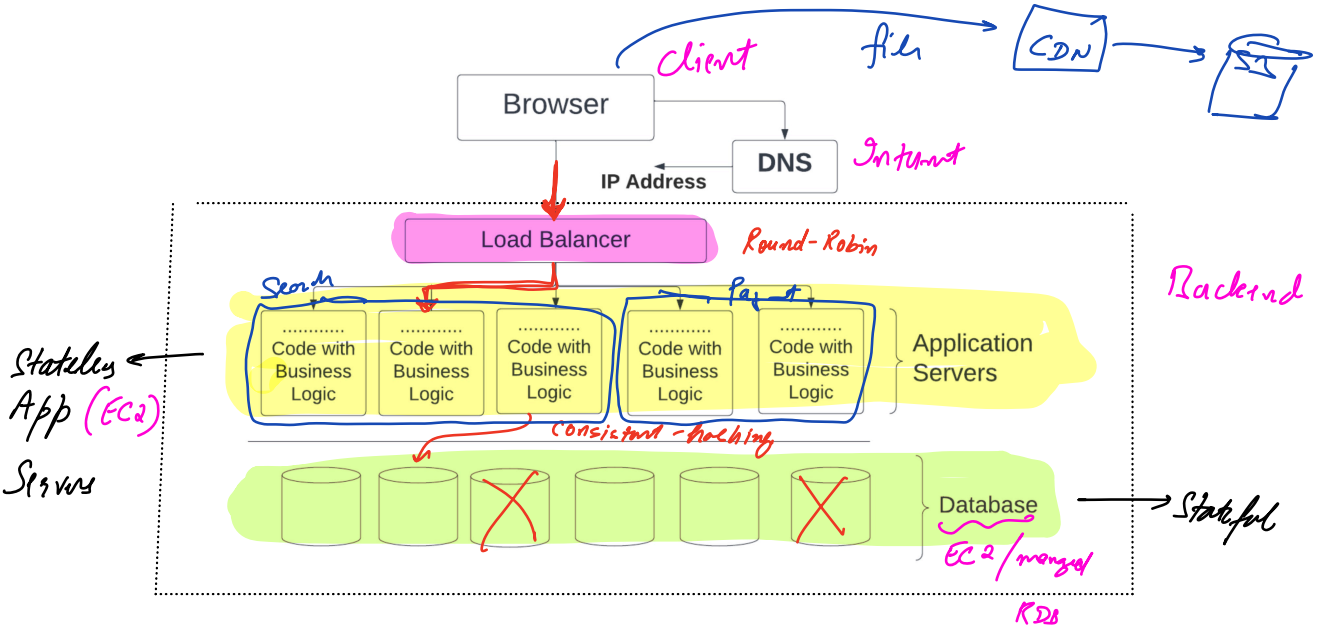
? Is this good design?

1. Data access is fast ∵ data is local.
2. Simple to code & deploy.
3. Transactions / monitoring / security / testing → easier.

(CI/CD) → Continuous Integration & Deployment

? Cons

1. App Code & data is tightly coupled (100 times/day, once per week)
2. Cannot scale things independently. → each server is v.v. expensive



? Which request to which server?

Round Robin

∴ app servers are stateless

Who does the Consistent Hashing? Load Balancer / App Server

DB client library / DB orchestrator

Zookeeper → distributed → keeps track which server configuration

How do the Appservers know how many & which DB servers are available?

DB orchestrator will keep track of this

Making Tea

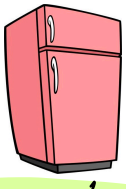
Source → industry

for you → store / shops

? Ingredients

- Water
- Tea leaves
- Milk
- sugar
- ginger
- Lemon grass
- gloves
- cardamom

Super Market

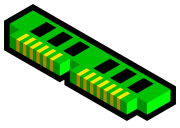


Kitchen / Refrigerator (temporary storage)

- → Source
- → all of the stuff is stored here
- → up-to date / fresh stuff
- → super slow / high latency
- → always needs to be persisted on HDD

- → derived storage
- → typically much smaller
- → can have stale items get rid of them
- → ultra-fast
- → can be in HDD / RAM

Redis / Memcache / ECache



Caching → always layered

client

Browser Cache
- DNS cache
- Cookies (tokens / session ids / theme)
- media cache (images / html / js / css / pdfs)
- local storage
- indexed DB

files → CDN
API req. → DB

Computer Memory
- HDD → level 3
- 100ns
- RAM → level 2
- 500x
- CPU → Processor
  - L3 cache
  - L2 cache
  - L1 cache
  - Registers
- 0.2ns
- flip-flops

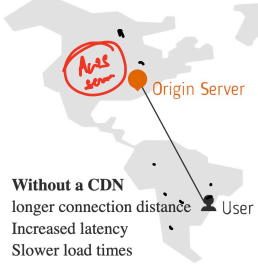
Content Delivery Networks (CDN)

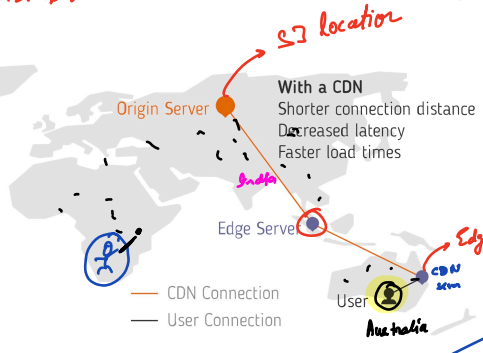
highly distributed

- Cloudflare
- Akamai
- Amazon Cloudfront
- Fastly

70% all internet is secure via http?

Edge → close to user





Scaler
- Backend Server → Code (AWS)
- S3 → file storage (AWS)

GET scaler.com/logo.svg → server to file stored in

s3.aws.com/scaler/...svg
- S3 → cheap for storage, expensive for access
- S3 → ultra slow

cloudfront/scaler
- Source
- s3.aw...
- Slow Lit...

https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

? How to get the IP of the nearest CDN server?

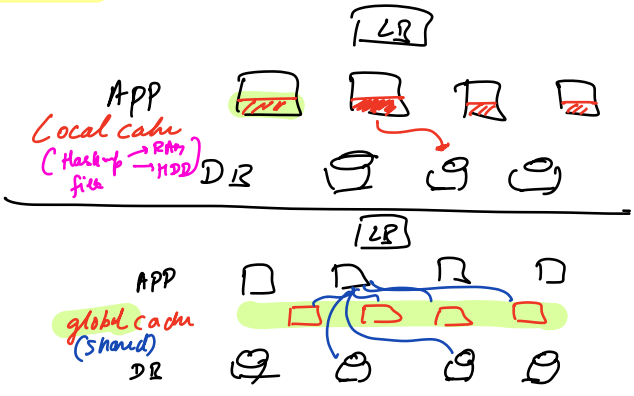
1. GeoDNS → low penetration
2. Any Cast → hack/trick for CDNs
   - try to act as their own Geo DNS

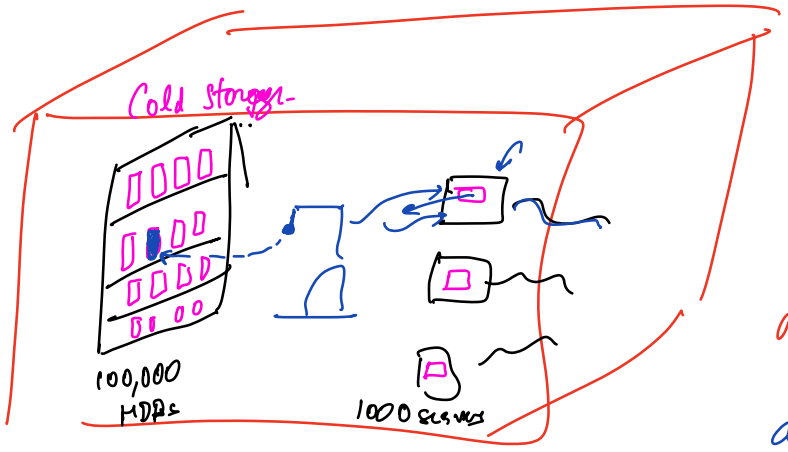
https://www.cloudflare.com/en-gb/learning/cdn/glossary/anycast-network/
https://constellix.com/news/anycast-vs-geodns

→ backend ↓

Local Cache

Global Cache

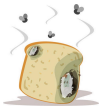



Cold storage
100,000
HDDs
1000 servers
Glacier
Amazon
data center

10:19 → 10:30

Challenges with Caching

Stale Data



↓ Invalidate it

Cache is not the source of truth → DB

data in DB is updated
data in cache is not updated

Tiny

Cache can become full

to add new data
↳ remove some other data (Eviction)

Cache Invalidation

TTL (time - to live)

→ Cache can have stale entries
→ some of the cache might be unused

{Eventual Consistency ✓ 90%

1. {Eventual Consistency
2. fast reads (10,000/s)
3. fast writes (100/s)

Write Through

1. Immediate Consistency → No stale data whatsoever!!!
2. fast reads (10K/s)
3. slow writes (10/s)

Write Back

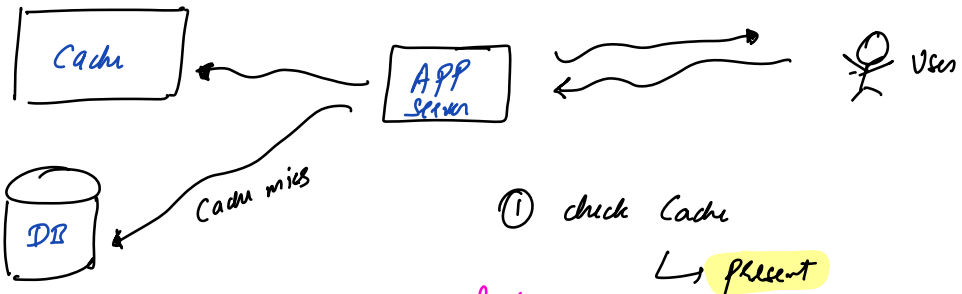
1. No Consistency → data loss!!
2. fast reads (10K/s)
3. ultra-fast writes (10K/s)

Write Around (v. similar to TTL)

1. {Eventual Consistency
2. fast reads (10K/s)
3. fast writes (100/s)

TTL — Pull

Time to Live ⇒ for how much time do we consider the data as sufficiently fresh (usable)



① check Cache
- ↳ present
  - ↳ TTL is okay → return from cache
  - ↳ TTL has expired
    - ↳ cache will reply as "missing" → remove it
- ↳ absent (cache miss)
  - ① fetch from DB
  - ② Response
  - ③ Async → is sent into Cache
    Record the timestamp

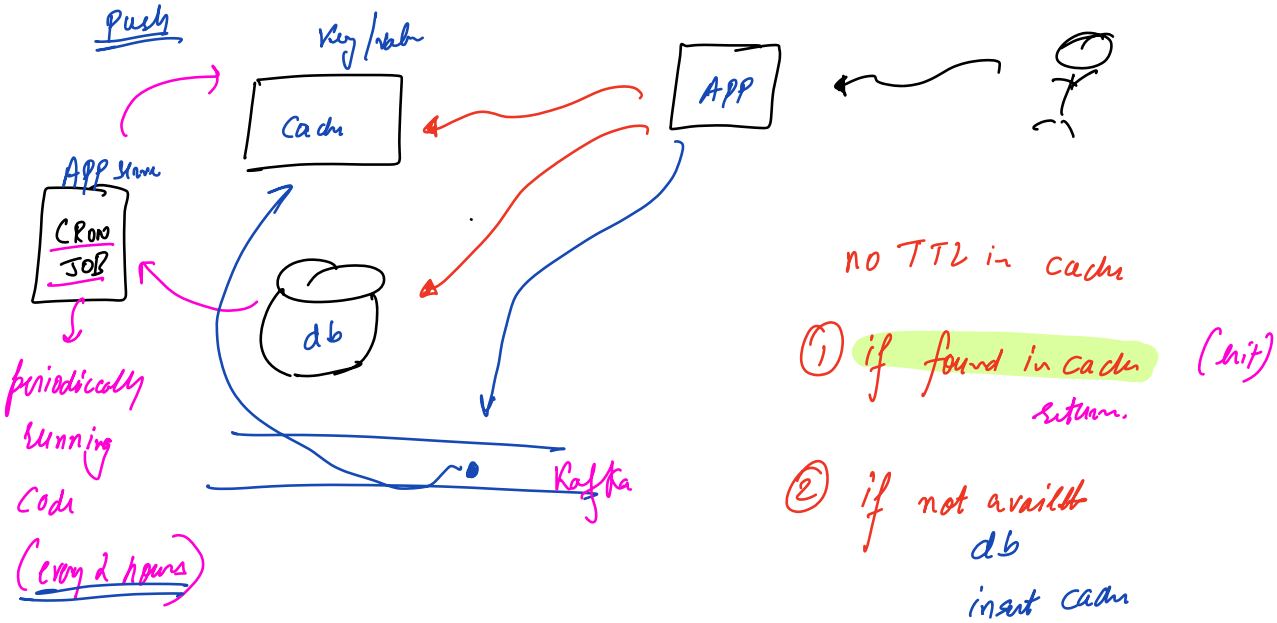
Cache

when did we last fetch from DB

Key	Value	Timstop	TTL
user-1-book	{ = }	10:48 PM	2 hours

Write → just to db

Write - Around



no TTL in cache

① if found in cache (hit)
return.

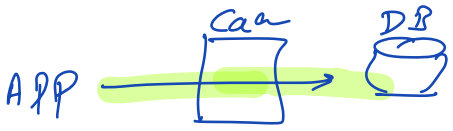
② if not available
db
insert cache

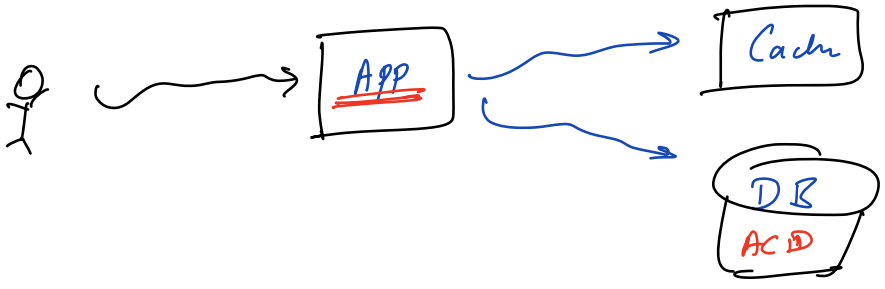
CRON

```
Select * from bookmarks
where updated_at > (now() - 2 hours)
```
→ update in cache.

Writes → just to db

Write-Through





① check cache
- ↳ present → return
- ↳ absent
  - ↳ read from DB
  - ↳ insert into cache

} Read

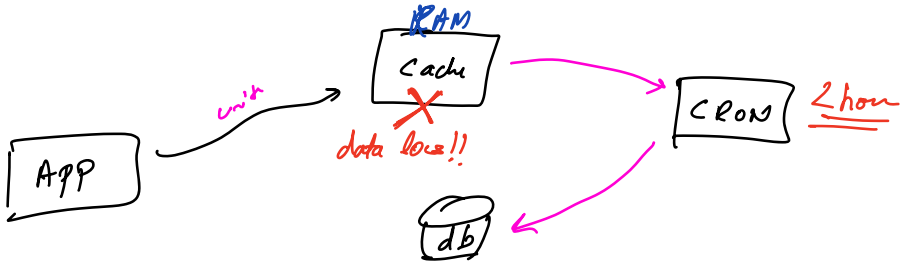
Writes

write to both DB & Cache → atomic transaction

if DB ✓ cache ✗ → roll back
DB ✗ cache ✓ → & retry
DB ✓ cache ✓ → response success

} 2PC (two-phase Commit)
high latency
(writes are slower)

Write - back



write → simply write to cache

if cache crashes before sync → data loss

Youtube → View count 1k / 1M / 100M / 10K

twitter

Cache Eviction

Similar to Page Replacement Policies

Strong Assumption: if a piece of data has been accessed recently / frequently in past → most likely to be accessed in near future.
↳ temporal locality

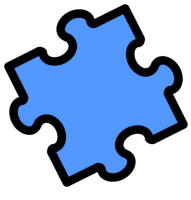
First in First Out — not that good
Simple Queue

Least Recently Used — 89 %
Priority : timestamp
→ priority Queue (min-heap)
→ LL + HashMap → preffered

Last in First Out — v.v. rare → special case → H/w

Least Frequently Used — 9 %
priority : counter
→ Similar to LRU implementation.

Some-shitty-website-that-noone-uses.com
↳ 3 visitors in past 10 years.



For next class

Code Judge

evaluates
↳ ✓
↳ WA
↳ TLE
↳ MLE

? Local caching of test data on the app server. How to invalidate on the update?

300,000 people live
heavy caching

? How is facebook's newsfeed calculated, and how can you make it very fast?

2 B users
↳ update frq
↳ read frq.

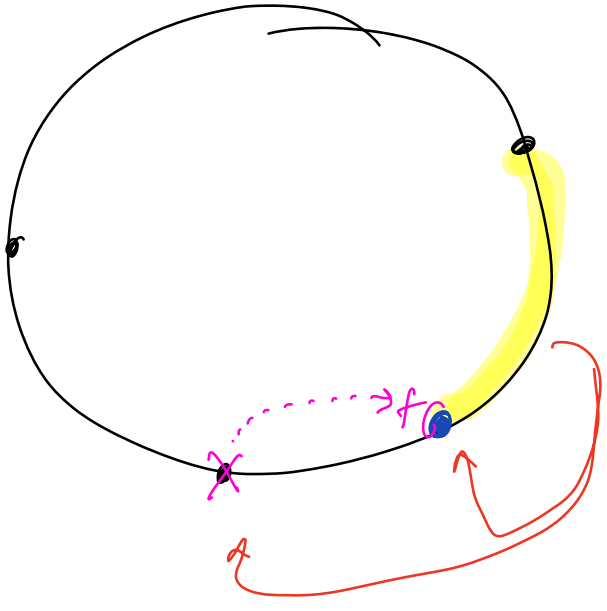
Unix → daemon process
↳ background

ls → list directory

mv → moving

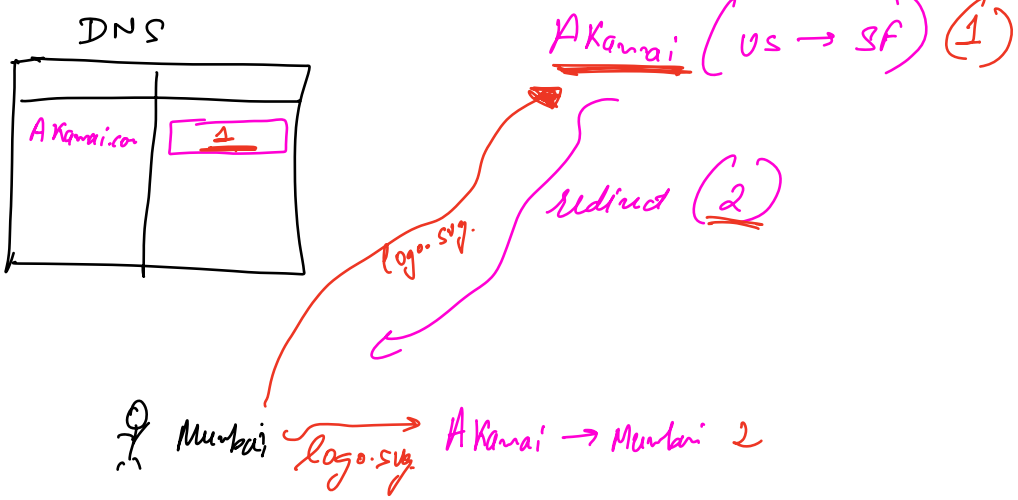
Cron → periodic tasks

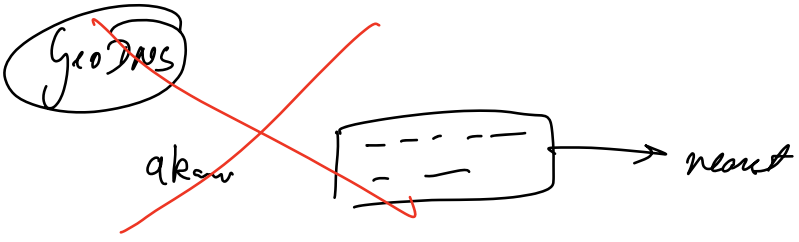
Cron job → periodic task



Stand-creation

Any Cast


DNS
Akamai.com
1
Akamai (US → SF) (1)
redirect (2)
logo.svg
Mumbai
logo.svg
Akamai → Mumbai 2


GeoDNS
akam
react