# CACHING

1] Caching in algos → To avoid doing same ops. especially computationally complex ops. that take a lotta time multiple times
∴ caching used to improve time complexity (T.C.)

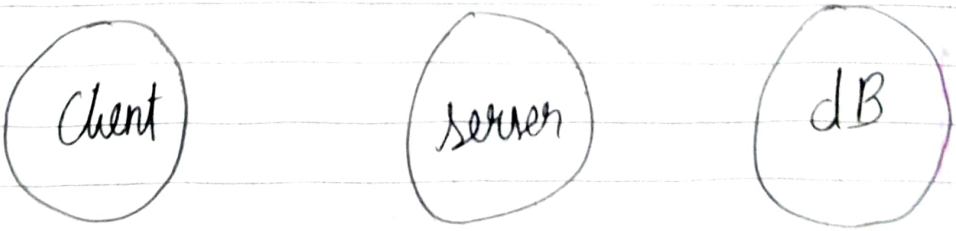
2] Caching in sys. → Used to reduce (improve) the latency of a sys. (speeds up the sys.)

3] CACHING → Storing data in a locn that's diff from the original data locn s.t. it's faster to access this data frm memory

It's a way to design a sys. s.t. if we were originally using I/ops. or data transfers that take a lotta time (eg:- Network requests) the sys. is designed is s.t. we do diff types of ops. or data transfers that are faster.

I Caching in diff places in a sys.

eg:-




Regular op. → client makes net. req. to server to get some data from dB and then data transferred from dB → server → client

1) Caching at client level - Client caches a data value s.t. it no longer needs to go to server to retrieve it

2) Caching at server level - You want client to always interact with server but maybe the server doesn't need to go to dB for data retrieval each time
∴ Go the dB once → get data → cache in server → transfer data to client as per net. req

3) Cache in b/w components - eg:- cache in b/w server and dB

not in our control when we des. sys. → 4) Caching at hardware level - eg:- CPU caches - make it faster to retrieve data from memory.

5) Caching occurs by default, at many other such kinda lvls. of sys.

II

## Advantages / Utility of Caching

Caching speeds up sys → 1) Avoid large magnitudes of net. reqs. → If the client wants to access the same data again and again, no new net req. (client → server → dB → server → client) done each time as we cache reqd. data so client doesn't need to go to dB each time
Caching done at client or server lvl.

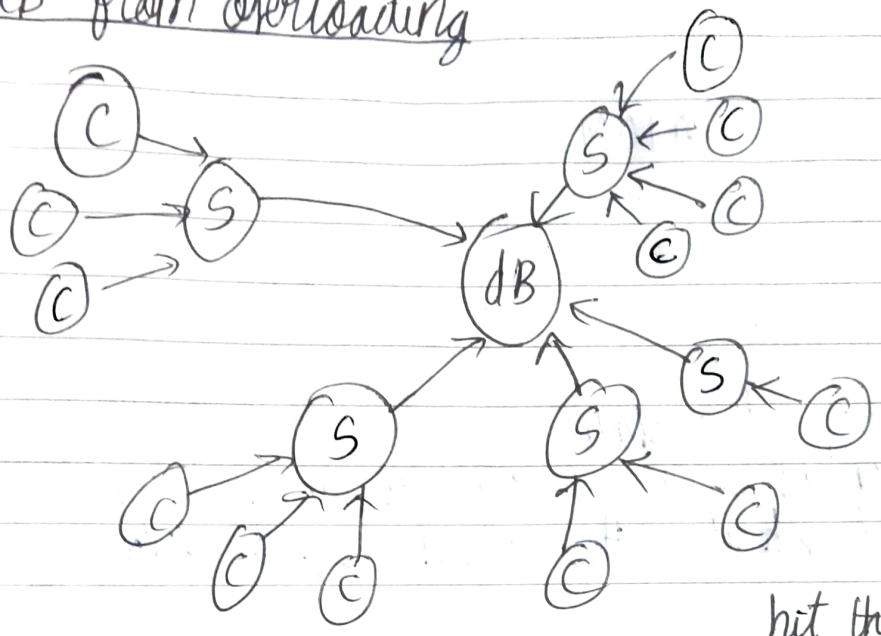
Caching speeds up sys. → 2) Avoid redoing a computationally long, bad TC, complex operation each time
eg:- say on net. req. from client → server,

everytime, this time - consumin algo runs. To avoid performin this op multip times, caching is done.

3) dB optimized

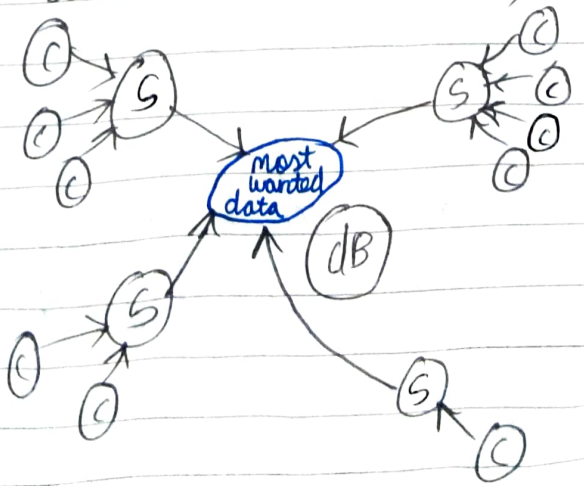
## Prev. dB from overloading

eg:-




Multip clients across multip servers hit the dB with the same net. req. eg:- Each client makes indiv. net. req. to server to get Instagram profile of a popular celeb.

Indiv. req. are fast so speed is not the concern. We use cachin to prev. same data being read from dB $10^6$ times (or somethin like that)
∴ a cache is created, outside dB, for that data




If we don't cache, too many req.s to dB for this data + in gen. other data accumulate and overload dB

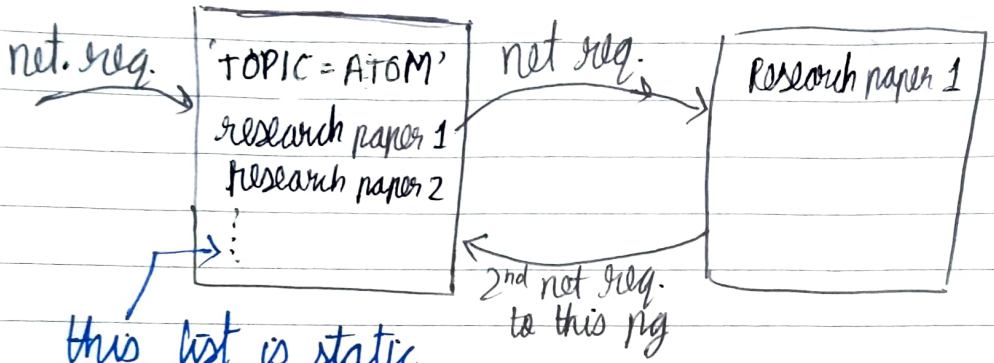
k-V = key value

Date: M T W T F S S

III

## CACHING in ac^n

1) Reloadin static (immutable) components or content in a website
eg:- ResearchGate : every once a yr papers published on site




content so if we cache it after 1st net req. to this pg of website, the 2nd net req. to list pg will be a lot faster

2) Runnin code on algoexp. (AE)

i) Running testin code takes ≈ 1 sec.
ii) for customer code → runnin testin code reqd.
for AE soln code → w.r.t this code is correct code ie- it'll past all test cases so no need to run, test

Cache stored → server level
OR
→ detached from server in an indep. comp.

Redis (popular in-memory dB) ⇒ a k-V store used for caching.

All AE solns. codes pre-stored with a unique 'key' in this Redis cache

Page

Date:
M T W T F S S

## Caching in ac^n

When a code is called to run on AE → http req. made to AE servers → req (bunch of bytes) hashed down to a single signature (key)

Cache with Redis (k-v) store is checked whether this key is pre-stored

if pre-stored → code called is AE soln code. no need to run. Takes < 1 sec to show result

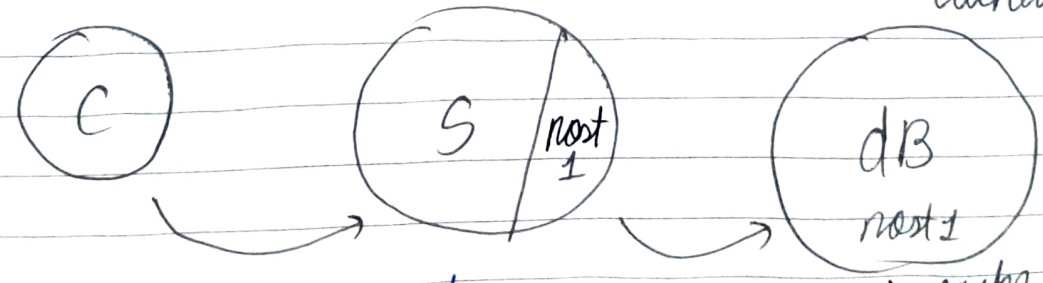
if not pre-stored → Code called is customer's code so run it. Takes 1 sec. to give result

Until now we discussed cachin for a sys. that needs to read data.

IV Cachin and Sys. where users can read, write, edit data.

eg:- Readin, writin, editin posts of FB or LinkedIN

1) When you write a post to FB, say that using net reqs. that post is → stored in dB
→ cached in server




2 sources of truth for given post → cache in server
→ dB

2) When you now wanna update your post, dependin on type of cachin in server : post stored in the

JS server.js > app.get('/nocache/index.html') callback > database.get('index.html', page) callback

```
const database = require('./database');
const express = require('express');

const app = express();
const cache = {};

app.get('/nocache/index.html', (req, res) => {
  database.get('index.html', page => {
    res.send(page);
  });
});

app.get('/withcache/index.html', (req, res) => {
  if ('index.html' in cache) {
    res.send(cache['index.html']);
    return;
  }

  database.get('index.html', page => {
    cache['index.html'] = page;
    res.send(page);
  });
});

app.listen(3001, function() {
  console.log('Listening on port 3001!');
});
```

JS database.js > ...

```
const database = {
  ['index.html']: '<html>Hello World!</html>',
};

module.exports.get = (key, callback) => {
  setTimeout(() => {
    callback(database[key]);
  }, 3000);
};
```

Date:
M T W T F S S

dB → updated instantly
dB → updated async. ly.

| Write-thru cache | Write-back cache |
|---|---|
| 1) dB and cache both updated instantly when you edit post | 1st cache is updated instantly<br>Then dB is updated async. ly<br>→ updated after a period of time eg. 5 hrs, 30 days etc.<br>→ once cache is full and you gotta evict stuff outta cache |
| 2) Takes time as you still gotta go to dB and come back for an edit | Takes less time as only cache update at the instant of edit |
| 3) Edit secure in 2 sources of truth at the instant of edit | Edit secured only in cache at instant of edit<br>∴ if somethin happens (eg:- delete) to our cache before dB is async. ly updated, we lose the edit. |

V

## Staleness

eg:- YouTube comments sec$^n$.

You don't wanna update data asyncly as it might happen that:

- → someone posts a comment (SERVER 1)
- → other person reads it frm diff. server (SERVER 2)
- → 1st person edits comment (but dB not updated and his server's cache updated asyncly yet)
- → other person still sees original comment (unedited) and replies to unedited comment (STALE DATA)

Cache in each server accessing YouTube dB

as SERVER 1's cache ↓ don't want this to happen by SERVER 2

Cache in server 1 is STALE ∵ it hasn't been updated properly

Soln: Make cache outside servers into a common comp. for them all outside dB
∴ Each time cache is updated from server 1, server 2 accesses this latest updated cached data

NOTE ∴ While sys. des. ing just see if that part of sys. is ok with staleness or not and accordingly choose how you gonna cache the data

eg:- YouTube
- → view count on vids → lil bit of staleness is fine
- → comment sec$^n$ → even a lil bit of staleness is NOT fine

Date:
M T W T F S S

VI

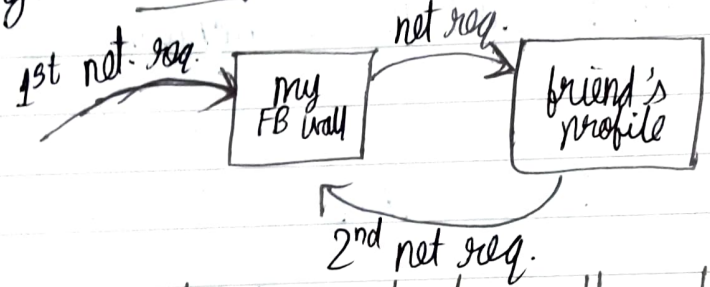
NOTE: WHEN TO USE CACHING?

✓ 1) Static (immutable) data → Cachin is easy to implement, fast while using

eg:- Research papers website

X 2) Dynamic (mutable) data → Cachin is tricky to implement as data stored at 2 diff loca^n's and you gotta make sure data in both loca^n's is in sync.

eg:- Facebook (FB) wall → not cached




On Facebook website, the data you see on your wall after 1st and 2nd net req. to FB wall page is diff.

✓ You're doin only 1 single thing eg: either of readin or writin data

X You're doin multip things → readin, writin, editin data.

✓ You don't care abt staleness of data or if you can invalidate (get rid of) stale data in caches in a distributed manner when dealing with a distributed sys.

VII

# CACHING - EVICTION POLICIES

We might wanna evict (delete) data frm cache for several reasons. eg: data is stale, cache's storage is full.

To evict data from cache, we follow certain rules a.k.a evicⁿ policies

1) LRU policy → Least Recently Used (LRU) pieces of data in a cache are removed if you fr some way of tracking what pieces of data are LRU. cuz we care least abt em.

2) LFU policy → Least Freq. ly Used (LFU) pieces of data evicted

3) LIFO or FIFO → Data ejected on a Last In First Out or First In First Out basis

4) other ways.

Choice of evicⁿ policy depends on kind of product you wanna do sys. des. for