AVAILABILITY

Fault tolerance :- How ~~tolerant~~ resistant is a sys. to failures. What happens if a server in sys fails? dB fails? → is sys gonna still be opera^{n}al

Availability → % of a given period of time (eg:- month, year) for which the sys. is atleast opera^{n}al enough to get its primary f^{n}s satisfied

Less availability (LA) → less uptime of a sys. during given time period
- → lose customers (existing)
- → lose the prospect of getting new customers & money lost
- → bad publi.

Sys.s that gotta hv. high availability

↓

any amt. of downtime costs a lot

- → **DUE TO THEIR F^{N}**
  eg:- sys. supportin airplane software
- → **DUE TO THE MAGNITUDE OF PEOPLE ACCESSIN**
  eg:- Youtube, Facebook
- → **CLOUD PROVIDERS**
  All services, platforms (dependent on cloud providers) businesses get heavily affect if cloud provider sys. fails

service ≡ sys.

eg of cloud providers : Azure, AWS, Google Cloud Platform, Oracle

Measuring availability (albt)

abbr. → albt

Availability is measured in terms of % of sys.'s uptime in a given year. (90% albt → sys. is up 90% of the time in a yr)

Most powerful services, sys.s have ~~are~~ ~~to~~ really HA. i.e. 99.9%, 99.99% etc.

Nines → Percentages with the no. 9

eg:- 99% albt → 2 nines of albt

Albt	Downtime / yr
2 nines (99%)	87.7 hr
3 nines (99.9%)	8.8 hr
4 nines (99.99%)	52.6 mins
5 nines (99.999%)	5.3 mins

Highly available sys → albt > 5 nines albt.

SLAs and SLOs (albt. guaranteed explicitly

SLA → Service lvl Agreement → Agreement b/w service b/w service provider and customers / end users of service that guarantee customers amount of albt, error free utility and some other objectives (known as SLOs → service lvl objective), failing which service the service provider pays back the customer some % of their fees

SLA

eg:- AWS (provider) Netflix (user)

SLO 1 → 1) Netflix get 99.9999% avbl.
SLO 2 → 2) Netflix get x errors while usin my service

⇓

If AWS fails to meet SLOs, it returns 10% of the service fees Netflix pays to it every month

NOTE: Achievin higher ~~avb~~ avbty is really difficult and may accompany tradeoffs like

- higher latency
- lower thruput

As a sys. des. gr., you gotta decide which part of sys you want "high avbl.$^{(HA)}$" and which part you don't really need HA.

eg:- Stripe (payment service for business

→ Core services → handlin payments, charging customers

These gotta have high HA else Stripe, as well as its client businesses, platforms lose lotta money

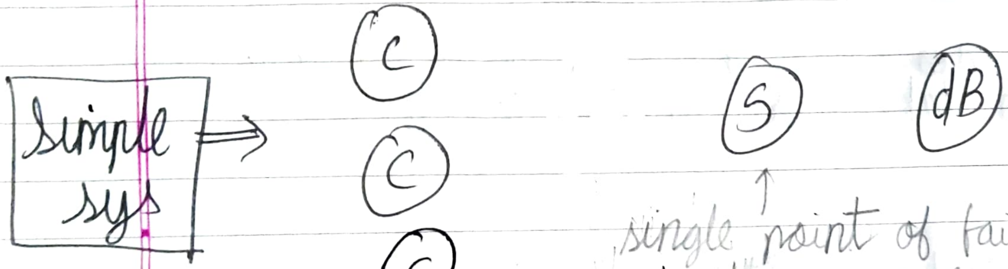
→ Business monitor dashboard → used by client platform to use business stats.

Doesn't really require HA. Something that, if fails, doesn't cause catastrophic losses

How a sys. is made "AVAILABLE"

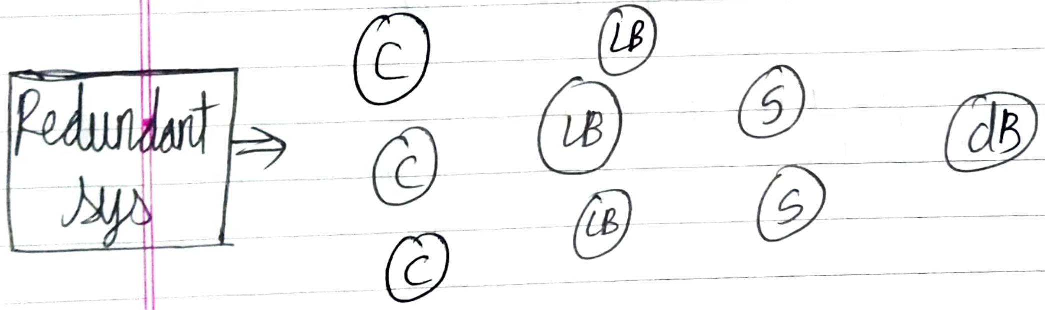
1 Single pts of failure (parts of the sys. ~~is~~ which, on failin cause entire sys. to fail) should be removed → For this we use redundancy while ~~sy~~ designing the sys.

Passive Redundancy → act of duplicatin or triplicatin or multiplyin even more, certain parts of sys.



C - client
LB - load balancer
S - server
dB - database

single point of failure as, if it fails client can never access database so purpose of sys. has failed



Load balancer → to ~~reduce~~ equally distri. load across all servers (prev. 1 server from becomin too overloaded & actin as single pt. of failure

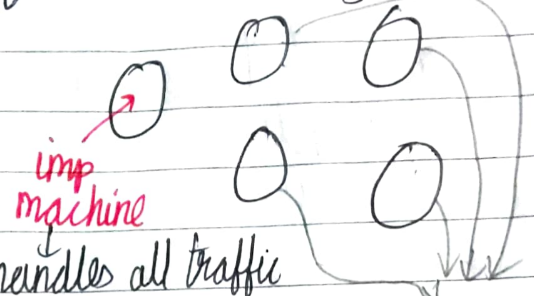
↓

Multiplied load balancers in this sys to avoid 1 LB from gettin too overloaded

eg:- twin engine sys. in airplane → duplicated

Active redundancy - Multip. machines that work together in such a way that only 1 or few of the machines are gonna be typically handlin traffic or doin work ('imp. machine'). If an 'imp machine' fails, the others machines know it failed and they'll take over

eg:-



handles all traffic

if this fails these 4 reconfig themselves to take over handlin traffic

2] Have rigorous processes in place to handle to handle sys failures cuz its possib. that these failures might req. human interven"

∴ Have these backup op"s in mind while sys. des. to get a crashed sys., up in runnin in proper timeframe.