Scaling

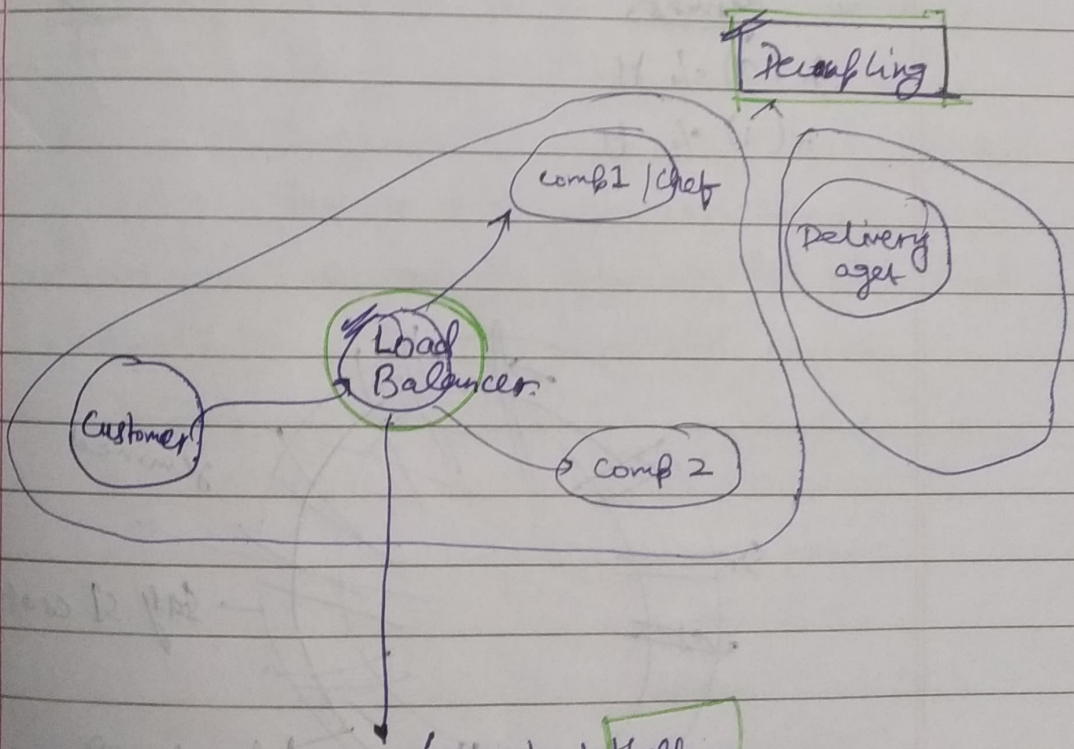
Horizontal

1. Load Balancer required
2. Resilient
3. Network calls (slow)
4. Data inconsistency
5. Scales well

Vertical

Big

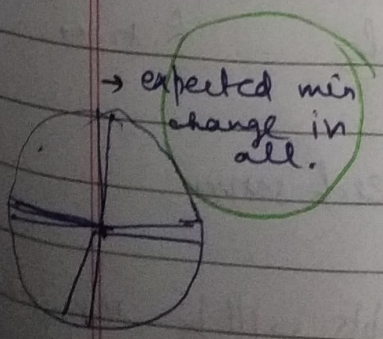
1. -
2. - Single point failure
3. Inter process Comm
4. consistent
5. hardware limit

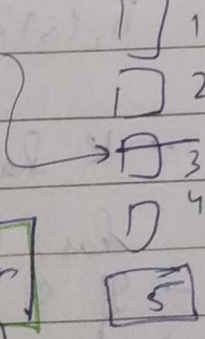


Consistent Hashing

REQUEST ID → r_1

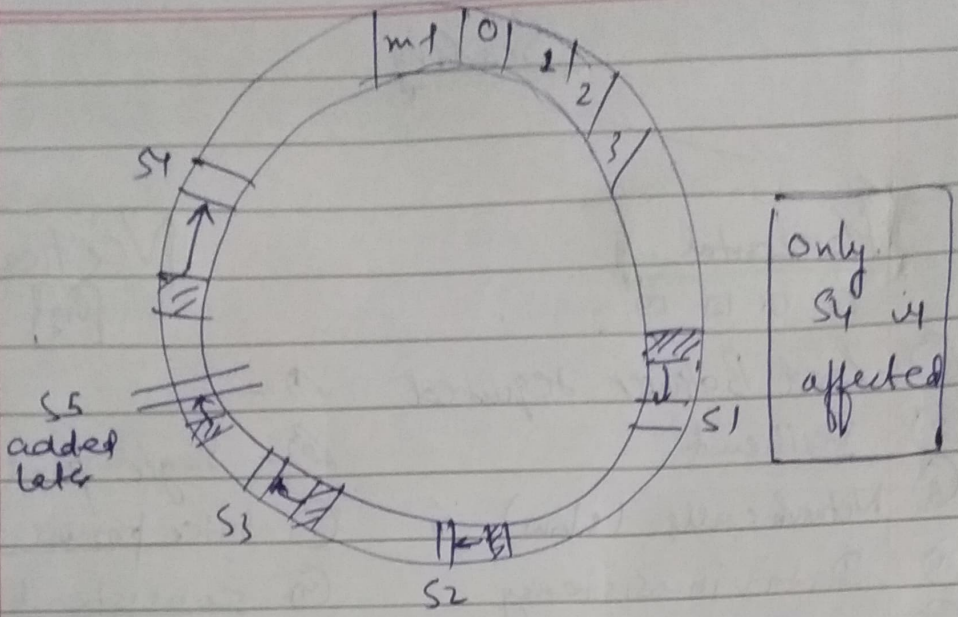
$h(r_1) = x\%n$





When new server added

r_1 may not go to 3

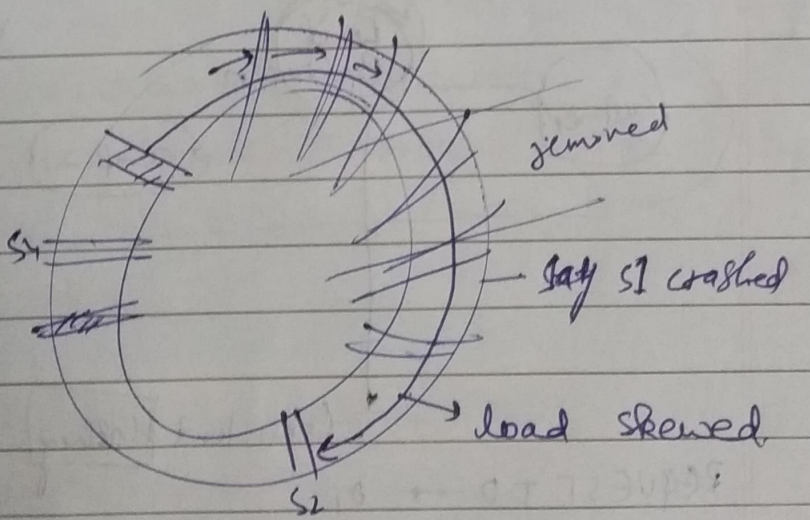


Theoretically → min change = $1/N$

servers

$h(a) \% M$

$h(1) \% M$



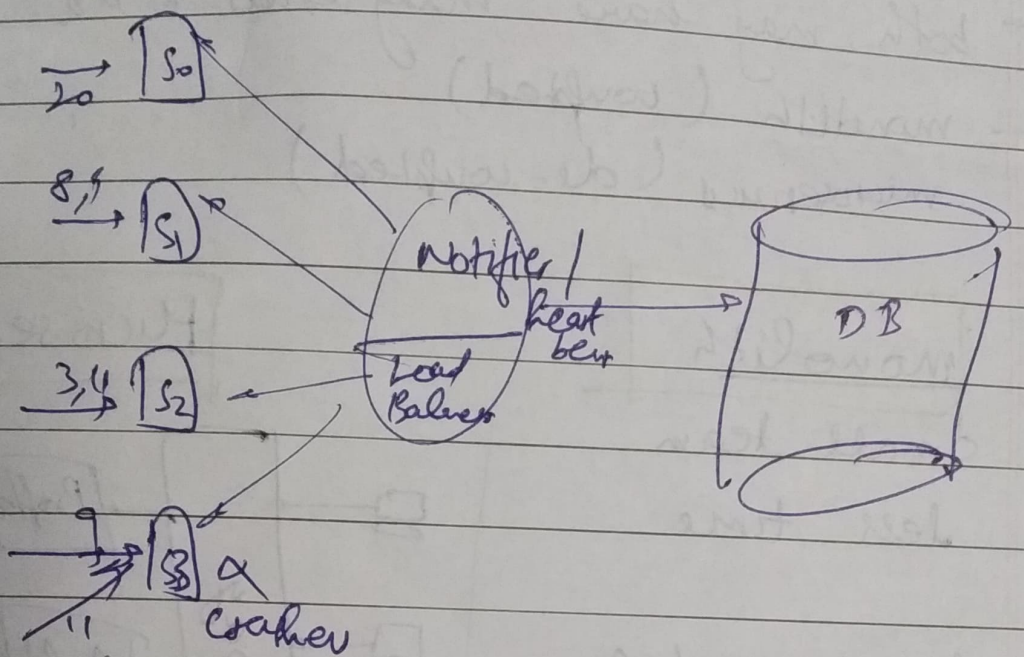
$h_1(a) \% M \rightarrow h_2(\ ) \rightarrow h_3$ ... k times

→ k hash functions i.e. each server has k pts

→ If S1 crashed, multiple pts will be removed

→ distributed uniformly.

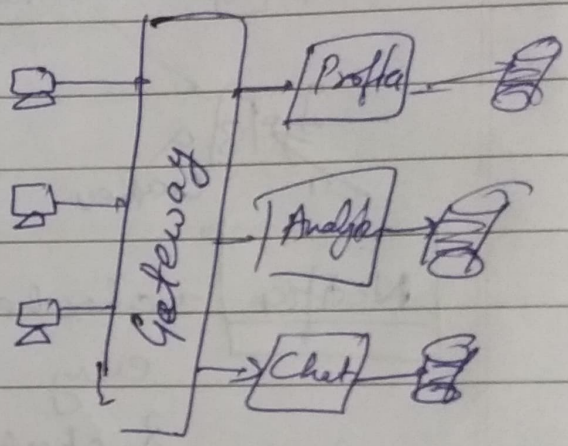
Messages / Task Queue



Notifier → checks if a server is live every 10-15 sec & checks if a req is not stuck / taking too much time it so it distributes the tasks of that server to other servers

Monolith vs Microservice

- both may have many servers & db
- monolith (coupled)
  microservice (de-coupled)

monolith	Microservice
- small team less time	
- less moving parts less complex easy deployment	
- less duplication	- easy scalability.
- faster	- easy for new code - parallel development
→ new coder need more context & is difficult.	- Need skilled developer
→ small changes, needful deployment & monitored.	
→ since everything is touching it.	
→ Single point failure.	

ACID

Backup Master → Slave

→ write first at master

→ read

Database Sharding

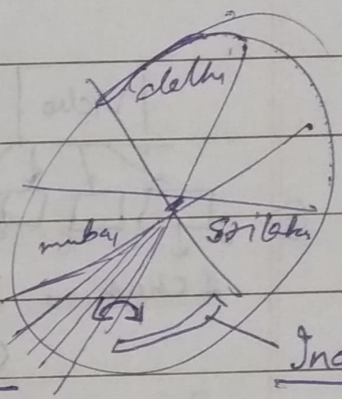
- horizontal Partioning → depend on one key which is attribute of data we're storing to partion data
- Vertical Partition → uses column to partition

DB Servers → Consistency Availability.

- Horizontal Partitioning ex.

ex on loc

ex on users



Problem

Indexing

① Joins

② Consistency

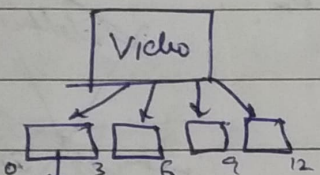
Case Study

(1) Netflix

① Different format video quality (f)
- high → .mp4
- med → .avi
- low

② Resolution — TV, laptop, phone (R)
- 1020p
- 720p

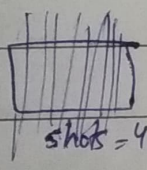
f x R → Total Videos for 1 Video



⊗ chunks to process above

1 chunk = 1 f, 1 R

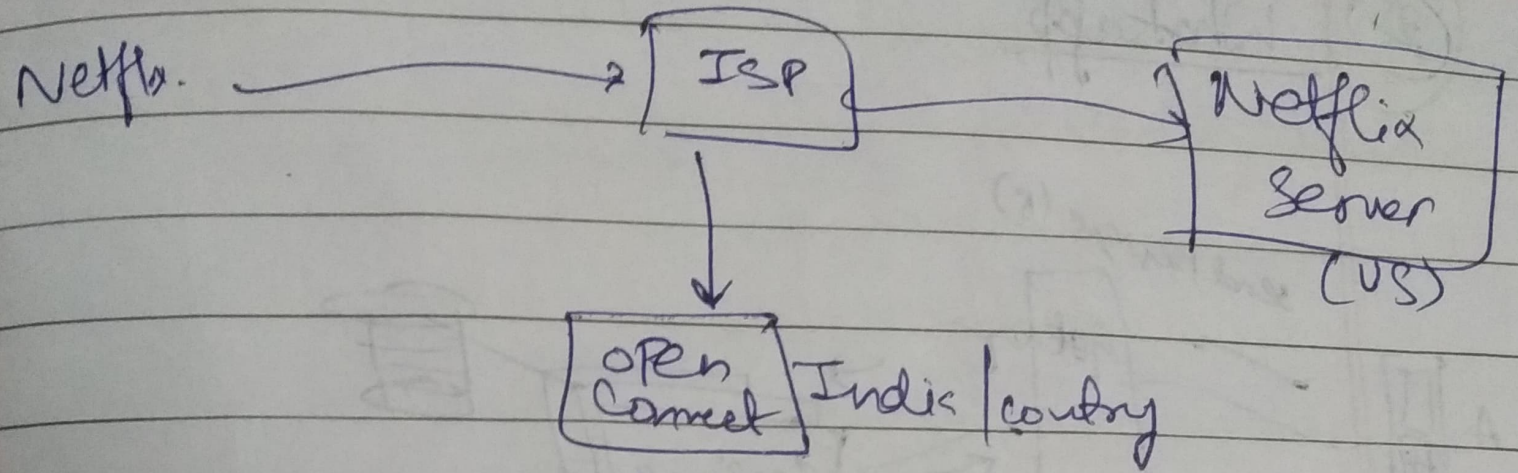
Breaking chunks on basis of Scenes > Timestamps



→ If you arbitrarily go to pts, seen netflix predicts this is a sparsely movie so not send whole chunk, but data user has asked for (shots)

→ Dense → Chunk

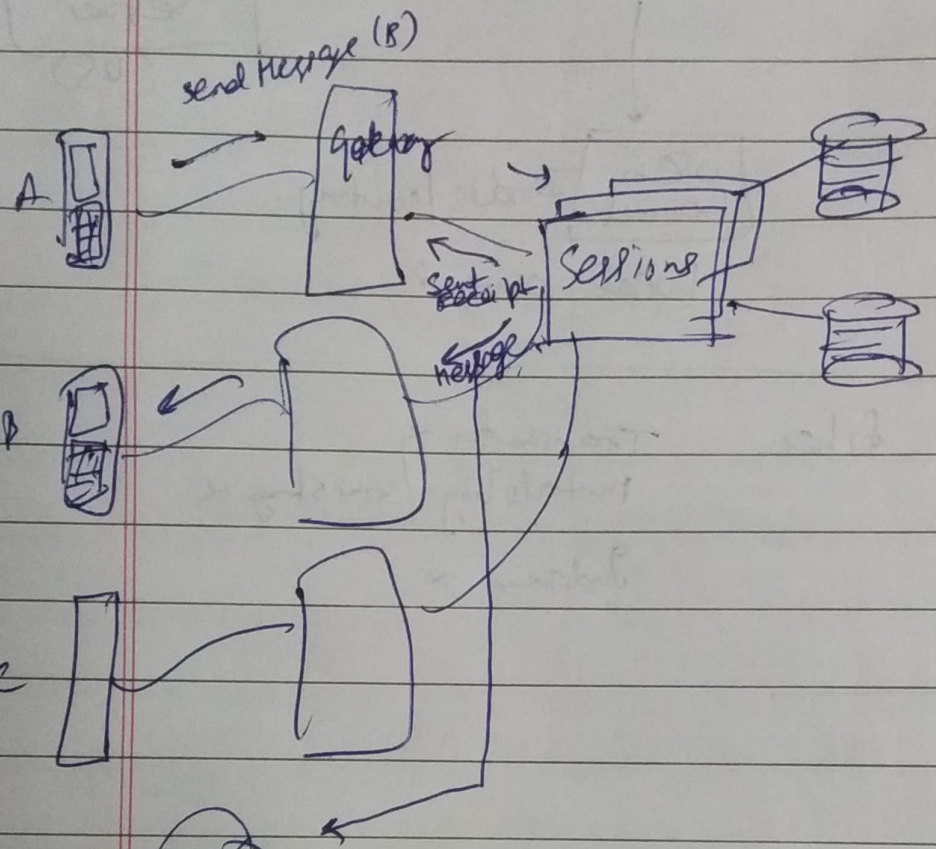
→ Data (Amazon S3) → cheap



Images - files

Transaction ×
Mutability / consistency ×
Indexes ×

(2) Whatsapp



http → client server only.

long polling → ask if U have any messages (but not Real Time)

TCP web sockets (wss)

Cache

① To save network calls

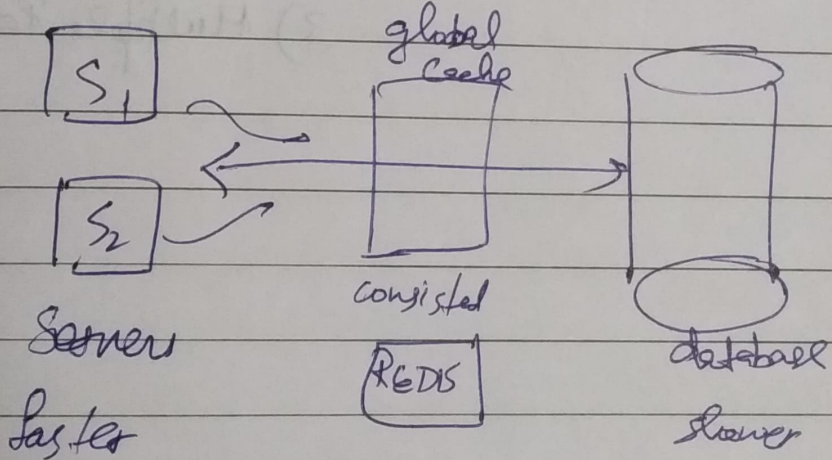
- can be expensive
- context production

Cache Policies

① LRU ② FIFO ③ LI-

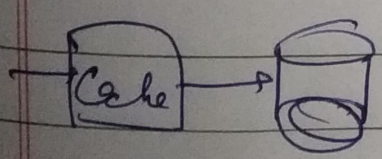
Thrashing →

Cache where to place



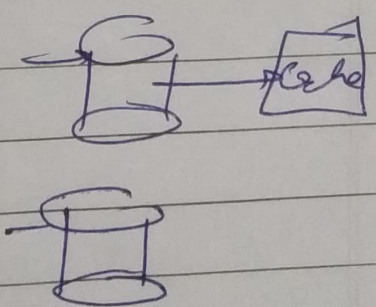
Cache

Write-through



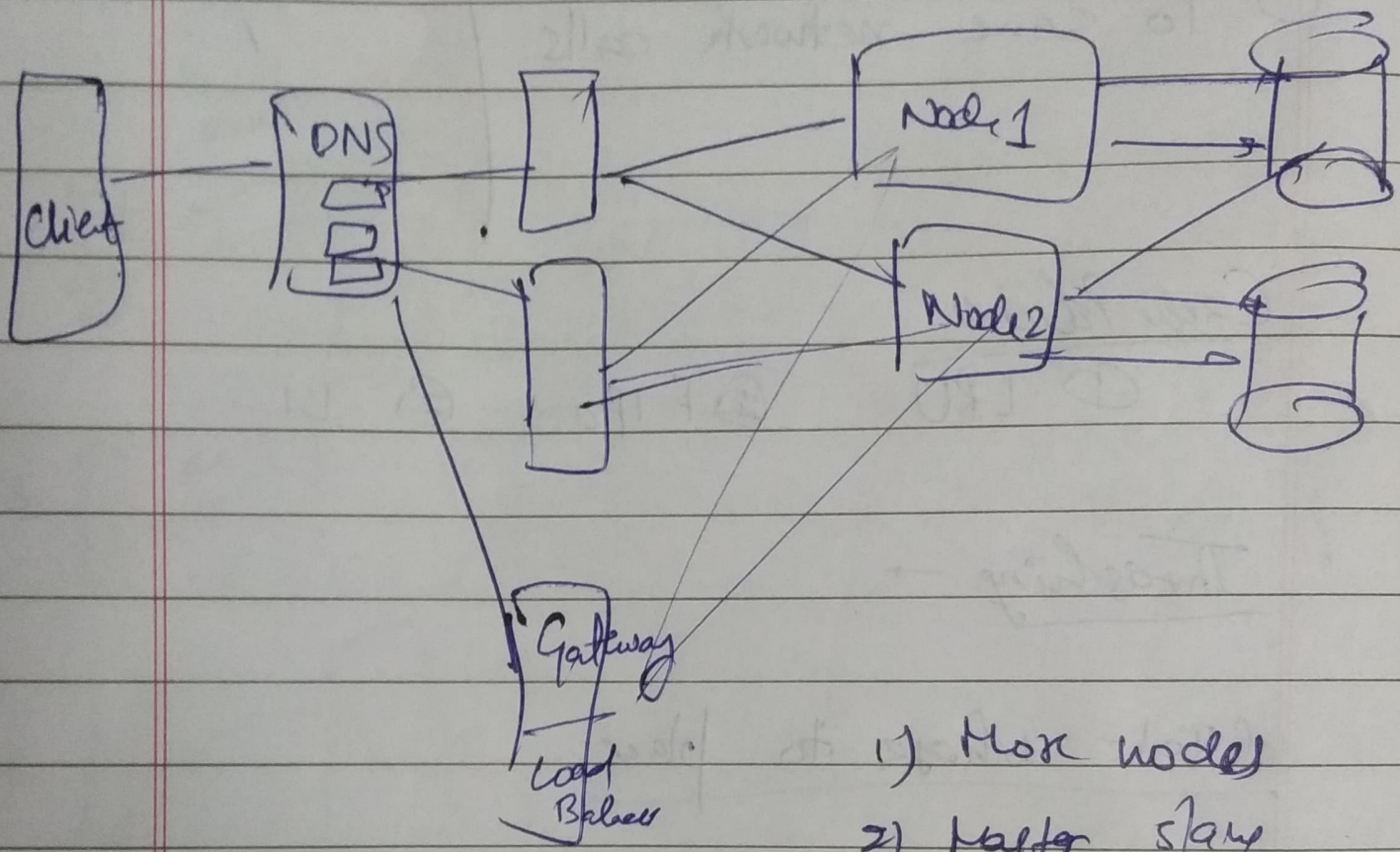
for
- not critical data

Write Back



- critical data

Avoid Single point of failure



1) More nodes
2) Master slave
3) Multiple Regions

No SQL (RDBMS) - Mongo

1. Insertion & Retrieval (can be cheaper)
   - whole block
   - no foreign key
2. Schema is easily changeable
3. Horizontal partitioning inbuilt.

---

4. ACID not guaranteed
5. Read time, are comparatively slow. — eg. age
6. No join predefined

Database optimization

B+ Tree

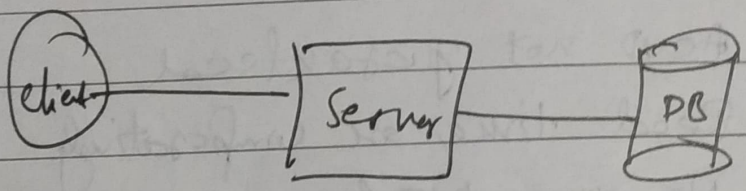
O(logN) Read

O(logN) write

(LOG) Linked List

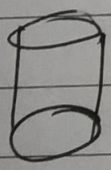
Read O(N)

Write O(1)



(write O(1) log)

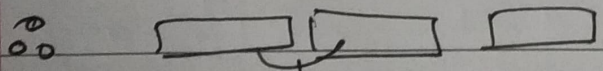
Persist → (Next, we convert data to Bt or Sorted Array so to Read O(logN))



10K records

→ So we're to sort 10K + 6 record

+6

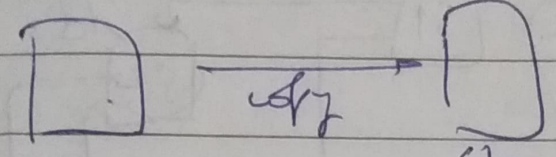


Sorted chunks .... still mergeSort — when?

4 (chunk) × $\log_2 6$ (size of chunks) ⇒ 12

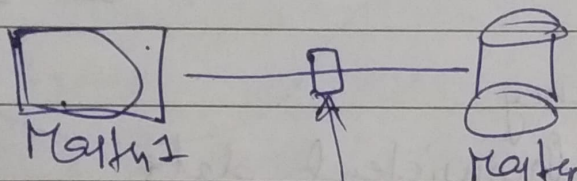
$\log_2 24 = 5$ — Better !!

Bloom filter – false true ✓
false false ✗

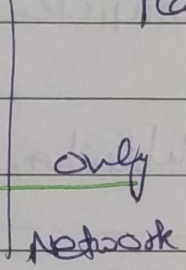


sync
or
Async

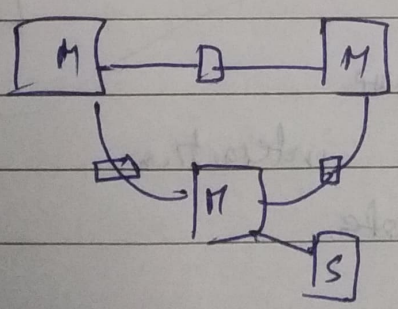
or



- Write on master only



- ~~If~~ Master 1 fail → Master 2 will take over & vice versa.
- but if network h/w. then fails
  - Split Brain Problem

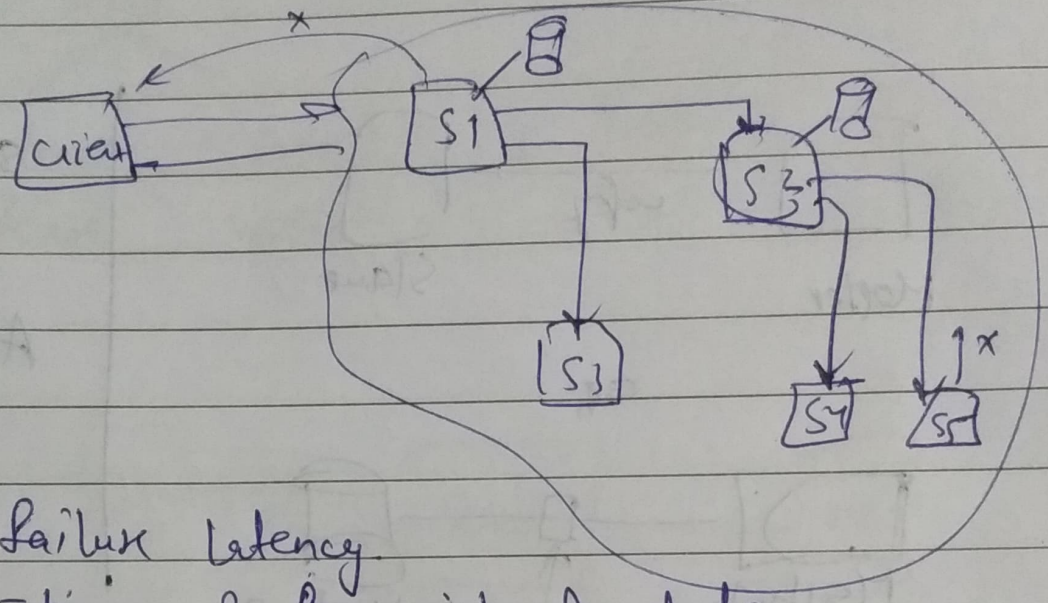


Roll back

2PC (2 are in Sync)

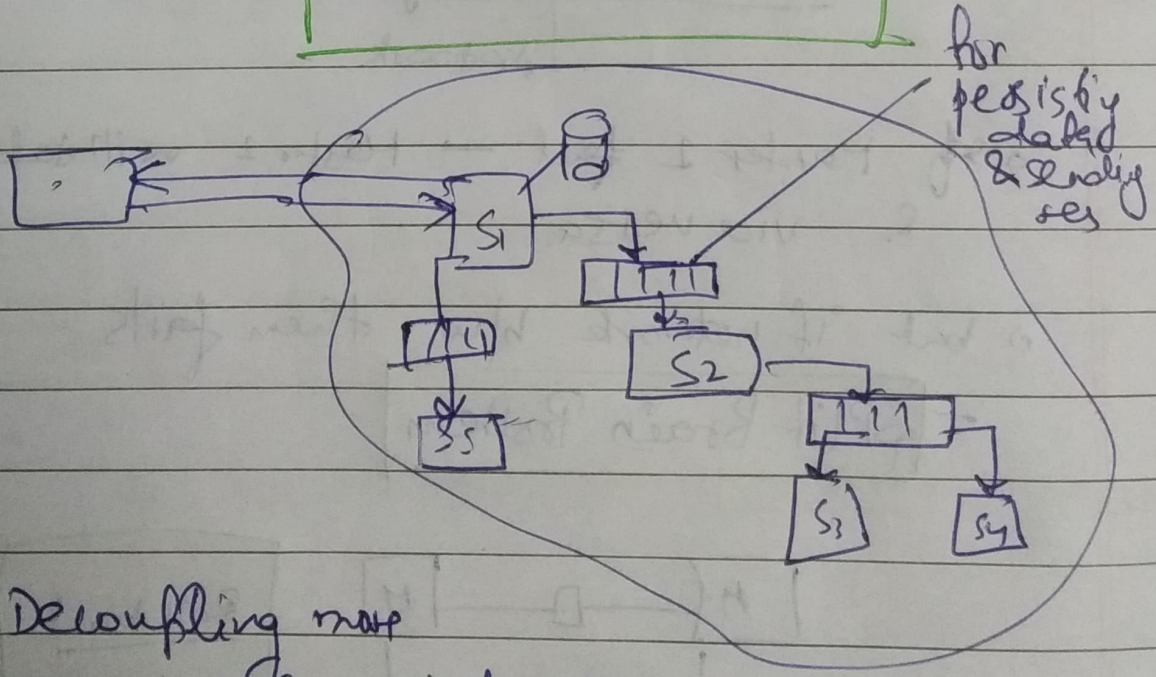
(req, res)

Microservices



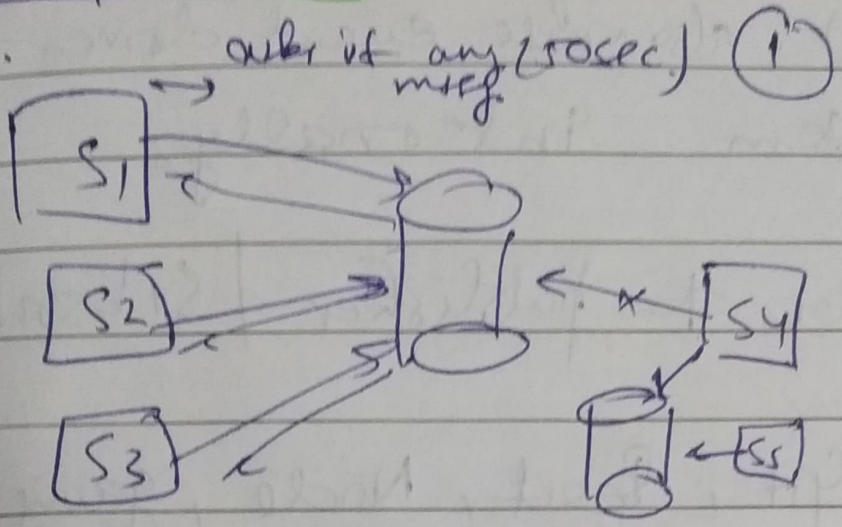
failure latency.
- time & Inconsistent data.

To solve — Publisher, Subscriber



① Decoupling more
② easy interface interaction
③ Consistent data.

Antipattern



Message Queue

1. push data / msg

for large systems

2. expensive / coders

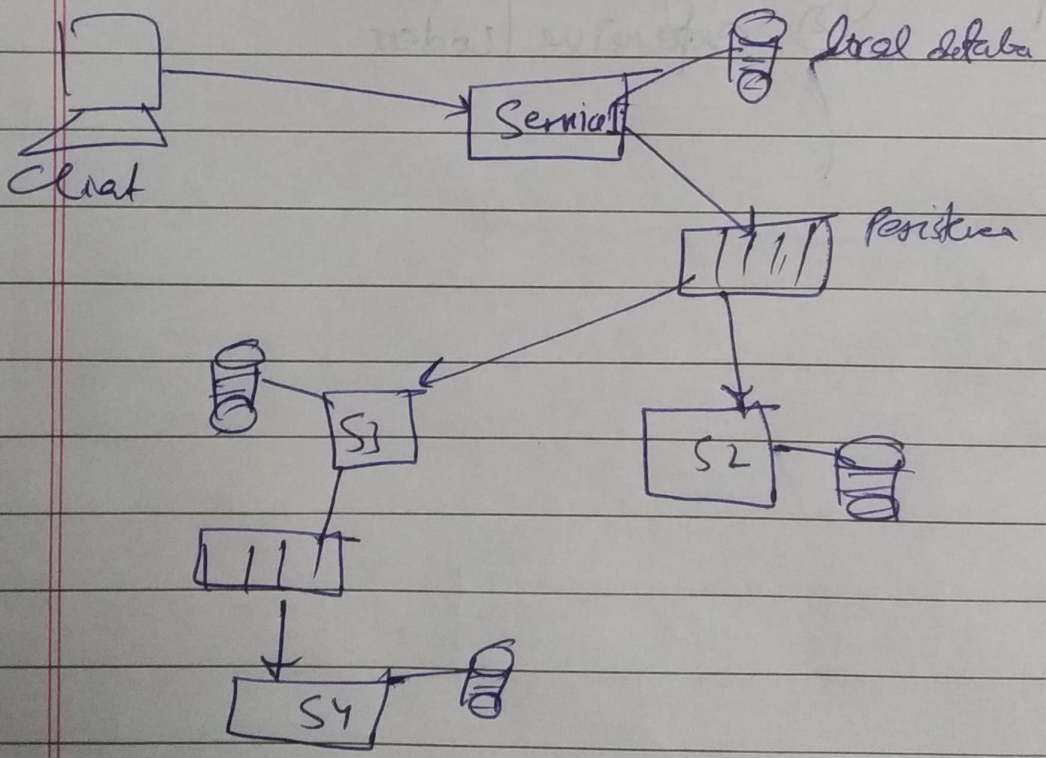
(req, res) Vs Event Driven System

- (req, res) : may also use event driven system internally.

— Similar to publisher / subscriber

- ex Git, React, Node, gaming system (Head shot)

— less flexible



— less consistent / data