# Lecture 03- CAP Theorem, Back Of Envelope Calculation, Architecture

# Just to know

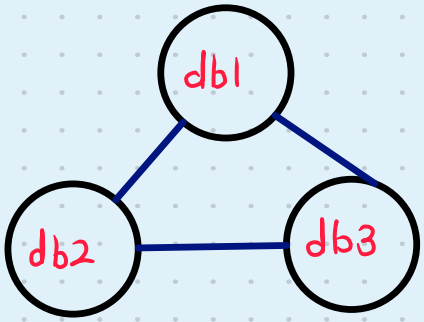
- Abtak hum jitna bhi application develop kar rahe the waha humara focus scalability pr tha.

Scalability kon provide kar raha tha ? → Distributed System

## # CAP Theorem

C: Consistency
A: Availability
P: Partition Tolerance

Suppose there are 3 db nodes, joh agar kuch changes hue toh baki dono ko bata deta hai




Consistency : Client jab bhi data request kare toh usse same data mile at same time, no matter on which node request has land.

Ex: Apna profile photo badla on insta no matter koi apki profile kaha se dekh raha hai (i.e. db node) sbko updated profile show hona

Availability : When client sends request it should get a response doesn't matter if it is stale data (i.e not updated data)

Ex: Instagram khola toh data load hona chahiye fir doesn't matter ki feed old hi hai

Partition Tolerance : Partition yaane suppose apka db jaha hai waha earthquake yaa tsunami aa gayi aur apka database se connection tut gaya

↳ Breaking of connection means partition

Aur apka joh application woh connection tootne ke baad bhi normal kaam kar raha hai that means it has partition Tolerance

↳ In short, our application should always be up or always operate

- Statement of Cap Theorem : No matter what, you will always get maximum of 2 out of these 3 i.e. CA or CP or AP
- Important point of theorem : We will always need partition tolerance and Consistency or Availability are optional.

mandatory

# Why we cannot achieve all 3 at same time

→ Because, partitions are inevitable (cannot be tolerated) because we need partition tolerance as our system is distributed system.

2. If our application cannot tolerate partition then our system is centralized (one failure leads to application down)

3. Therefore you cannot have both during a partition

↳ Because, if one part of the system accepts a write & another part does too (without coordination), then nodes no longer agree

⇒ Consistency lost

↳ If instead you force all partitions to agree before replying, then some partitions must wait (or reject requests) until the network heals

⇒ Availability lost

In One line

⊙ "In a distributed system, when a network partition occurs, you must choose between Consistency (all nodes agree) and Availability (system keeps serving) YOU CANNOT HAVE BOTH AT SAME TIME.

# Consistency Vs Availability

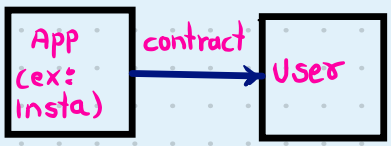
When to prefer consistency : Response upto date hona

Ex : banking applications

When to prefer availability : Response milna important hai doesn't matter if stale data hai

Ex : Instagram

# Availability Numbers




If down time -

- 99% of 365 days : 3.65 days down
- 99.9% of 365 : 8.77 hours
- 99.99% of 365

* Real world application ka no. agar 99 se kaam hai like 98 so it is not good built application

- Har app ka apne users ke saath contract hota hai ki saal mein woh kitne din down ho skti hai
  ↳ yaane no response
- koi application kbhi down nhi hoo skti yaane 100% available hai

Ex : Google, Amazon, Microsoft lies in 99.9 or 99.99%

- Har company chahti hai decimal ke baad maximum 9's laana

Applications are defined in number of 9's ⇒ Availability Numbers

# Back of the Envelope Calculations

This calculations helps in finding how better our application is through QPS (Query Per second) and Storage Unit

helps in improving response time

↳ means any CRUD operation/query on db

↳ helps in finding agar mein ek db le rhi hu and woh hum approx 5 yrs use krenge toh maximum uska storage kya hoga (estimate)

⊙ We will use example of Instagram and perform Back of envelope calculations

Before starting remember : Power of 2

| | | |
|---|---|---|
| 10 | Thousands | kilobyte |
| 20 | Million | megabyte |
| 30 | Billion | gigabyte |
| 40 | Trillion | terabyte |
| 50 | Quadrillion | petabyte |
| 60 | exabyte | |

Assumptions we need to consider

1. Monthly Active Users : ~2 Billion
   ↳ 60% of the users use insta daily : ~1.2 Billion
   DAU : Daily Active User : 1.2 B

   What users can do -
   a) see feed (1 Feed request = 1 Query)
   b) user posts photo / video / reel (1 post request = 1 Query)

2. Feed Check
   ↳ user check feed 30 times a day (avg)

   Estimation of 5 years -

Feed View QPS = $\frac{1.2B * 30}{86400}$ = ~420k/sec

86400 ↳ 24 hr x 3600 sec

Peek QPS (on holidays/weekends) = 2 * QPS
= 2 * 420k/sec
= ~840k/sec

3. Upload Check
   ↳ User publish 1 picture/reel every day (average)

QPS : Daily Upload req = 1.2B * 1 / 86400 = ~14K/sec

Peak QPS = 2 * 14K/sec = 28K/sec

4. Storage Unit

Assumptions
   ↳ 20% of user → video (~50MB avg)
   ↳ 80% of user → picture (~1MB avg)

- Photos per day = 1.2B * 80% = ~1B
- Storage : 1 Billion * 1MB = $2^{30} * 1 * 2^{20} = 2^{50}$ = ~1PB

(for storage calculation always convert into power of 2)

Video per day = 1.2B * 20% = ~0.25B

Storage : ~0.25B * 50MB = $0.25 * 2^{30} * 50 * 2^{20} = 2^{50} \times 12$ = ~12PB

Total : 12PB + 1PB = (13PB) → per day

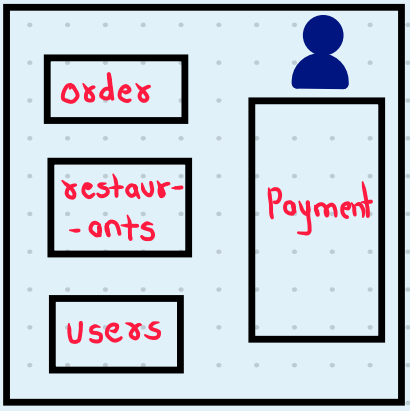
calculation for 5 years : 13PB x 365 days * 5 years = ~24000PB

↳ $24 * 10^3 = 24 \times 2^{10} \times 2^{50} = 24 \times 2^{60}$ = 24 EB (exabyte)

Interview tip -
- Calculations kone padenge and itna detail mein hum batayenge toh interviewer impress ho jaayenga
- Interview mein aage hard question nhi ayenga

# Monolith Vs Microservices

- Monolith applications : saara logic ek hi repo mein
  - ↳ har ek classes / logic / components of application in single file
  - ↳ This is legacy (bohot puraane time se use ho rhe)



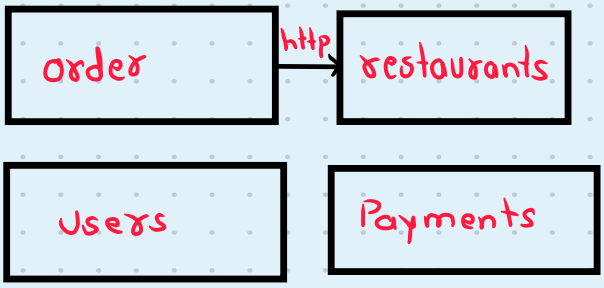

**Advantage**

1) Fast as communication betn two modules are just method call & not http (calls over internet)

**Disadvantage**

1) Not much scalable (can't scale single module)
2) Heavy application (CI/CD pipeline)
3) debug & testing hard

- Microservices : deploy every logic / component in different servers




**Advantage**

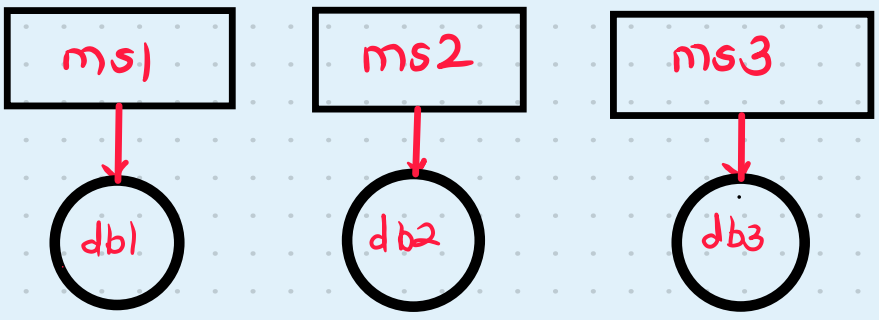
1) scalable
2) Light application
3) easy debug & test

**Disadvantage**

1) slow coz http calls hoga over internet
2) Transaction management
3) multiple microservice ko join krna

- Let's discuss the disadvantage of microservices in detail

1] Transaction Management (ACID Properties) ⟶ In short it means ek toh pura transaction hoga warna issue aagaya toh roleback ho jaana

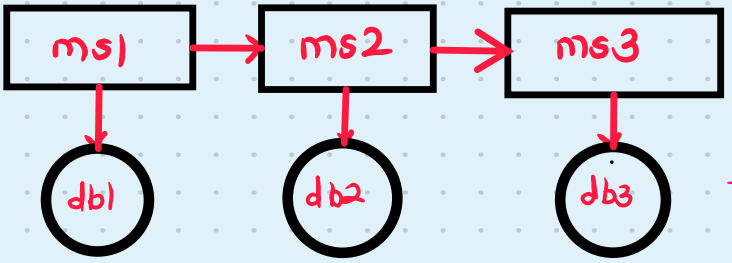



⟶ Yeh monolith mein achieve karna easy hai coz ek hi service hai & uska ek hi db hai

→ Ab har ek ms(microservice) server ka apna db hoga toh woh apas mein communicate kaise karte ki successfull hai yaa nhi

Ex: ms1 pr kuch kaam hua & success ho gaya operation par ms2 par error ke wajah se terminate ho gaya toh ab ms2 → ms1 ko kaise batayenge tu rollback karle

2] Multiple Microservices Join




→ yaha agar humne suppose ms2 ka code change kiya toh humme ms1, ms3 ka bhi check karna padega ka, ki kya koi change required hai

→ isse code mein multiple dependency aati hai

⊙ Ab inn dono problem ka solutions hai unke patterns jisse hum phases of microservice ke haad discuss koenge

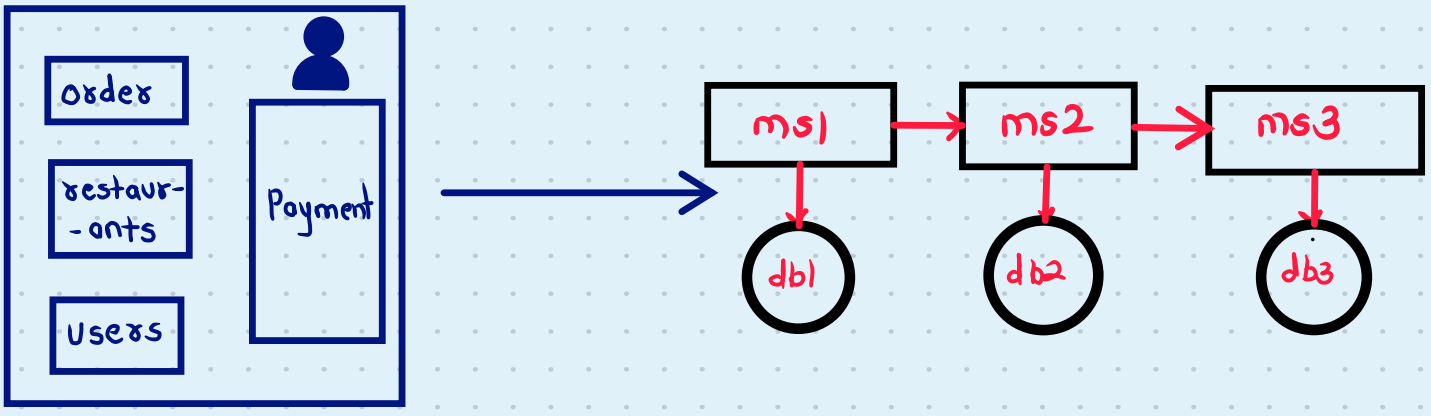
# Different Phases of Microservices (imp interview question)

- Phases during conversion of monolith (mL) to microservices (ms)

P-I] Decomposition → how will you breakdown your application (mL → ms)
P-II] Database → shared DB or unique DB for different ms in application
P-III] Communication → how ms will communicate : api / event driven system
P-IV] Deployment → how will be depoyed } handled by DevOps
P-V] Observability → how we will monitor }

1] Decomposition Phase

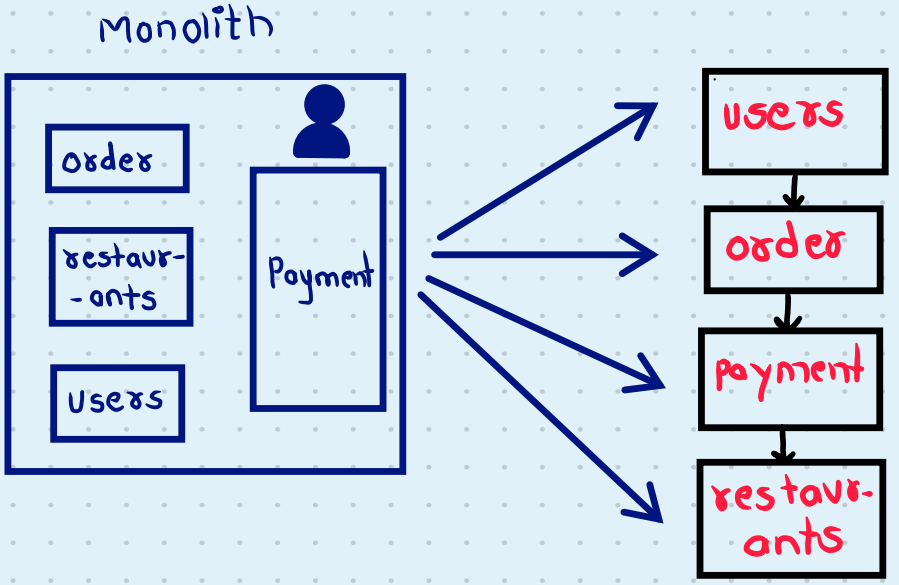
- If you have monolith → how will you break it to a microservice architecture




- Now we have design patterns to understand this thing of how things will work

① a) Decomposition by business logic } Yeh batayenga divide konse
b) Decomposition by subdomain } base par karna hai

a] Decomposition by Business Logic



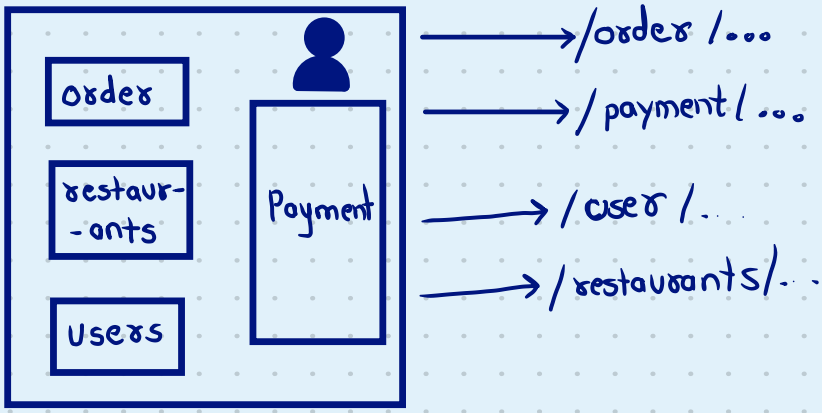

- divided the monolith application based on business logic
- yaane saare business components of zomato divided in parts

Disadvantage
- developer should be aware of entire business
- confusion of keeping components in which section

Ex: Agar menu component bhi aaya toh usse new entity / order / restaurants inmein se kaha rakhna

b] Decomposition by Sub-domain

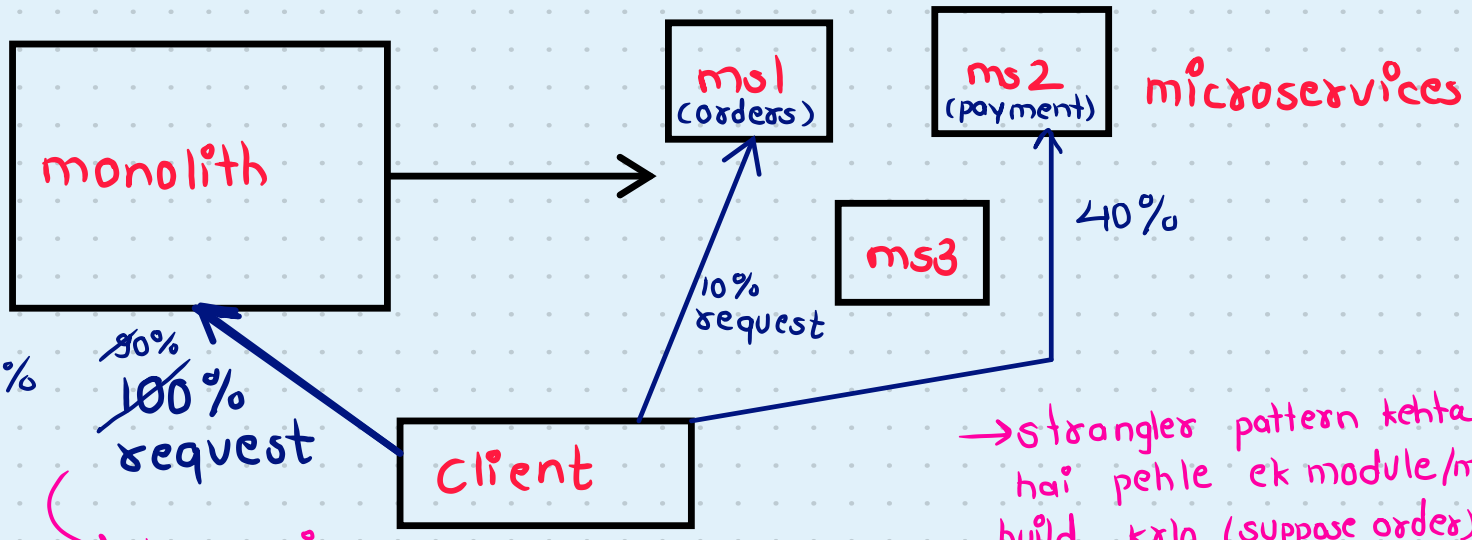
- Humare applications mein already kuch domains bane honge toh uss hisaab se hi divide karo
- jaise kuch requests
  - /orders → 1 entity (1 ms)
  - /payment → 1 ms
  - /user/add → 1ms
  - /restaurants → 1ms




② Strangler Pattern → Yeh batayenga karonge kaise (How will u do it)

- Humne decide toh koliya ki part kaise divide honge par monolith application deployed hai server par ... toh microservices ko pehle pura deploy krdu & then shift traffic pr monolith saara handle nhi kar paayenga.
- Toh strangler pattern kehta hai ki hum humare monolith ko dheere dheere microservices mein convert kara

Strangler means step by step




→ Ab hum isse kaam kaam karenge and microservices par badheynge

→ strangler pattern kehta hai pehle ek module/ms build krlo (suppose order)

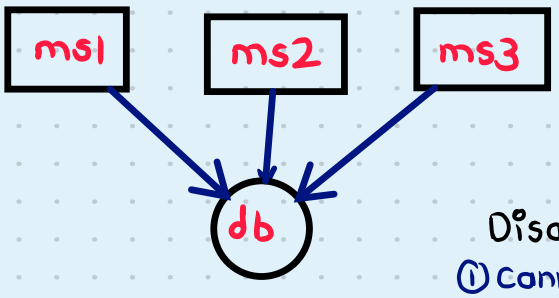
→ Ab 100% of api's req joh monolith ke server par jaati hai usko reduce kro to 90% & 10% ms pr bhejo

→ Ab isse divide krne ke bhi 2 ways hai
① by no. of apis (10% api on ms)
② by no. of users (jaise 10% user on ms)

## 2] Database Phase

- Each microservice can have → shared db / unique db

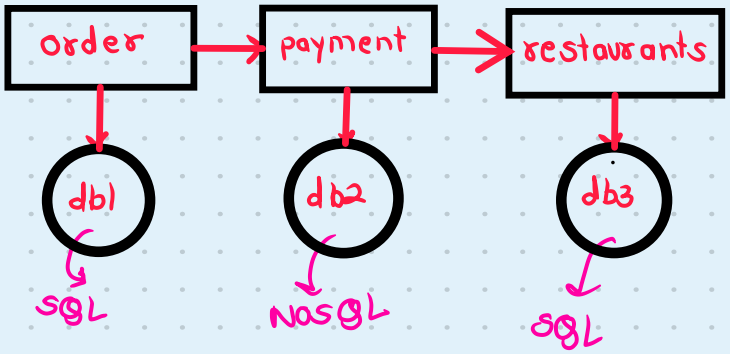
### a) Shared DB




Advantage
① simple to carry operation
② JOINS (SQL) - easily
③ Transaction management (ACID)

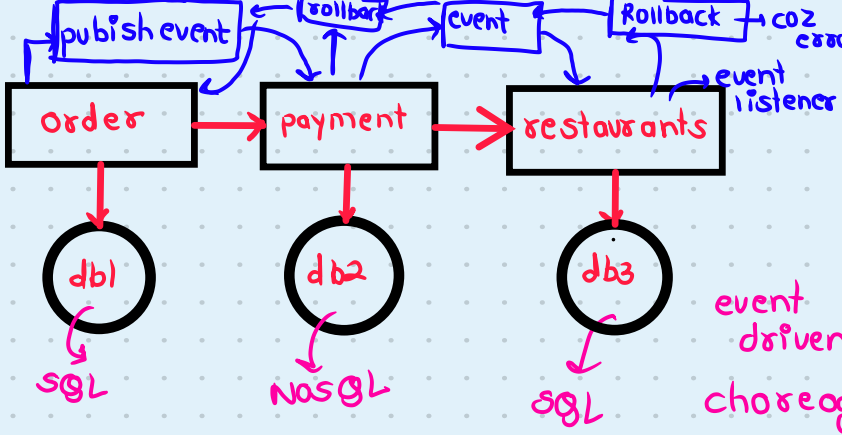
Disadvantage
① cannot be scaled properly
② limitation of either SQL/NoSQL (multiple options X)

### b) Unique DB




Disadvantage
① Joins not possible (soln : CQRS)
② Transaction management (soln : SAGA)

PS : Payments waala service directly, order ke database ko call nhi kar skta, order service ko bolna padega to call

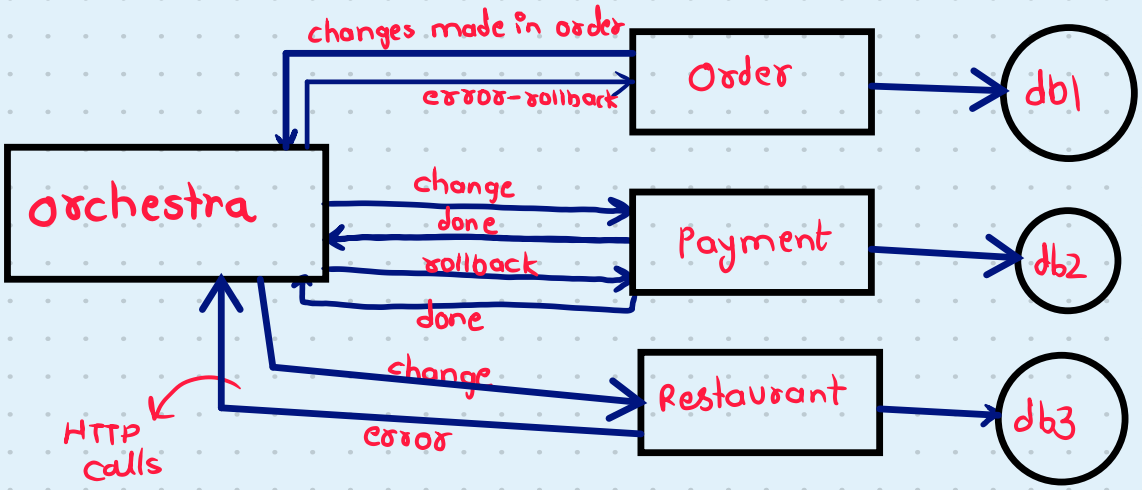



event driven/ choreography

- SAGA Pattern
  - ↳ soln of transaction mgmt problem - agar koi txn fail ho gyi toh baakiyon ke kaise pata chalega to roll back

SAGA Pattern states : jaise hi koi changes db mein ho toh ek event create karo & pass to next microservice. Agar txn successful hai toh pass forward otw backward to ask them to rollback

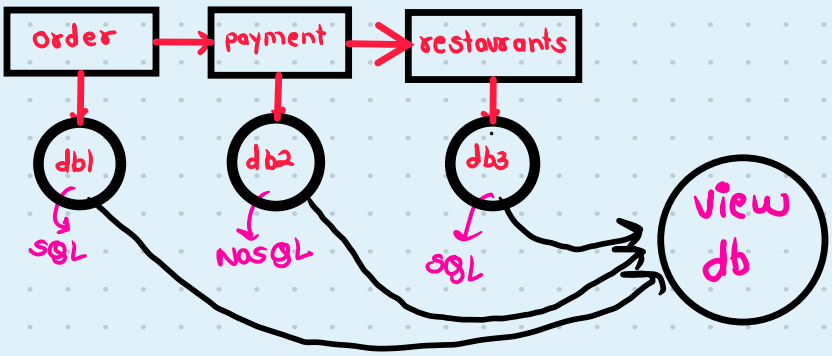
- Agar event driven system complicated lag rhi hai toh SAGA has one more option → orchestration




- Inshort it says, kuch bhi change hoga mujhe batao I will forward
- → error aaya? I will ask others to rollback

- CQRS (Command Query Request Separation)
  - ↳ soln of joins problem - different server par deployed database par join ops kaise lagaye

CQRS kehta hai - separate karo
- ↳ Command - write operation (update, delete, insert)
- ↳ Query - read (select)




→ ek alag view db banao jismein saare tables present ho aur query ke results usmein so doh.