Eg: Instagram ka Back of Envelope Calc Karenge.

Uske fhele not!.

Power of 2 → 2^10 :- Thousand

2^{10} → Thousands: Kilobyte.

2^{20} → millions: Megabyte.

2^{30} → Billion: Gegabyte.

2^{40} → Trillion: ~~PetaByte~~ Tera Byte.

2^{50} → Quadrillions: ~~Terabyte~~ Peta Byte.

2^{60} → Exabyte.

Assumptions:-

① Monthly active users ~ 2 Billion.

60% of these users use Insta Daily: ~1.2 Billion.

DAU:- Daily Active User : 1.2B

User See feed (1 feed request → 1 query)

User Post Photo/Video/Reel (1 Post req - 1 query)

User publish one pictur / Reel a day (Avg)

User check feed 30 times a day (Average)

We will make estimate of 5 years

Feed View QPS:

QPS: Daily Feed Request: $1.2B * 30 / (24 * 60 * 1 sec) = \sim 420k/sec$.

Peak QPS: $2 * QPS = \sim 840k/sec$.

UPLOAD QPS:

QPS: Daily upload req: $1.2B * 1 / 86,400 = \sim 14k/sec$.

Peak QPS: $2 * QPS = \sim 28k/sec$.

Storage units:

Assumptions:

20% of users: Video (50mb)

80% of users: Photo (1mb)

PHOTOS:

photos/day: $1.2B * 80\% = \sim 1B$.

1 Billion * 1mb: $2^{30} * 1 * 2^{20} = 2^{50} = \sim 1PB$

VIDEOS:

$1.2B * 20\% = \sim 0.25B * 50mb = 0.25 * 2^{30} * 50 * 2^{20}$

$2^{50} * 12 = \sim 12PB$.

Total: $12PB + 1PB = 13PB$

for 5 years

$13PB/day * 365 days * 5 years = \sim 24000PB$

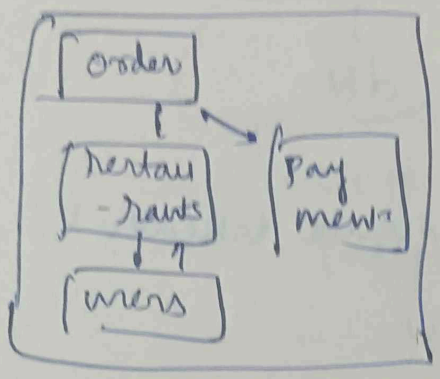
$24 * 10^5 = 24 * 2^{10} * 2^{50}$

$= 24EB$ ~ Exabyte.

Monolith and Microservices

Monolith → Sab kuch ek mai h.

eg:- Zomato



Adv

→ Isme agar order aur Payment ko communicate karna h to simple fn call lagega, jisse ye fast hota h.

Disadv

(1) Scalability

↳ Agar aaplog log raha h. orders badh rahe aur order server ko badhana h to aap individually order ko ni badha sakte, aapko pure ko replicate karna padega.

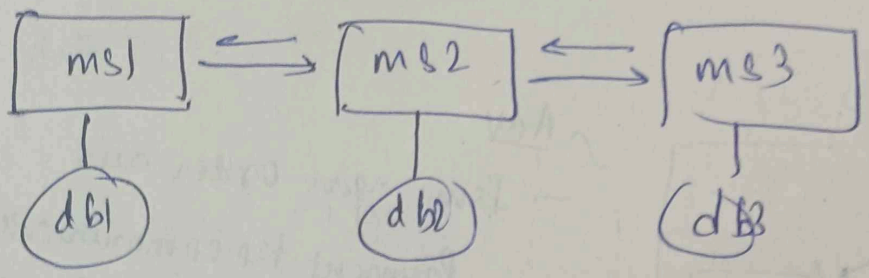
(2) Heavy

↳ Agar koi bhi change karke deploy kiya to kaafi heavy rahega.

(3) Debugging and Testing

→ Debugging and Testing is difficult in Monolith architecture.

Microservices

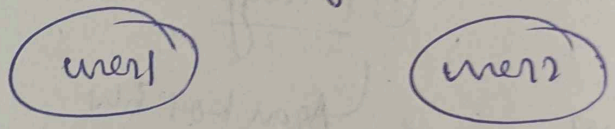


Adv → Jo Monolith ke disadv wo iske adv h.

Disadv → (i) Slow h thora.

(ii) Transaction management

↳ eg:- Banking ~~System~~ System me payment ka eg lete h



monolith me cheeze phones mai hoti, to agar phone1 mai hi rah a user phone2 me dikkat aaya to, acid me phone1 ko uss transaction ko revert karne ko kehna thora mushkil ho jata h.

Ya to payment hoga ya to ni hoga

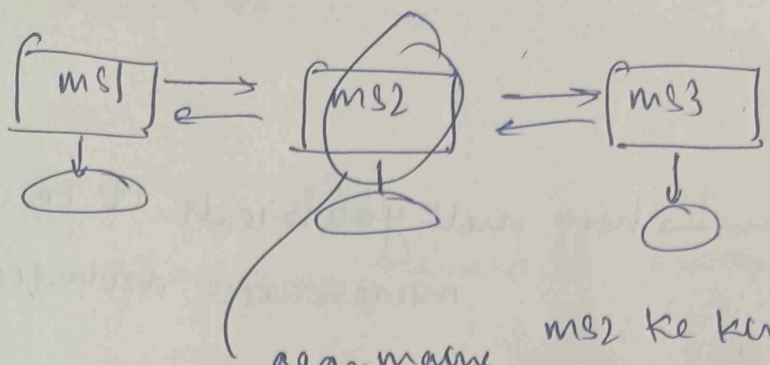
⇓

matlab acid koi scene ni hoga jaha user1 se debit ho gaya par user2 pe credit ni hoga,

⇓

transaction mai koi dikkat aayi to pura transaction cancel.

(iii) Multiple microservice scan



agar maine kahi to mujhe ye maake sure karna padega ki baaki ke microservices ke changes ke karna hai ke naa chalijaye.

MS2 ke kisi code mai changes

Different Phases of a microservice

(i) Decomposition - (monolith se microservice mai convert karna)

(ii) Database (Shared DB or Unique DB)

(iii) Communication (API / Event driven System)

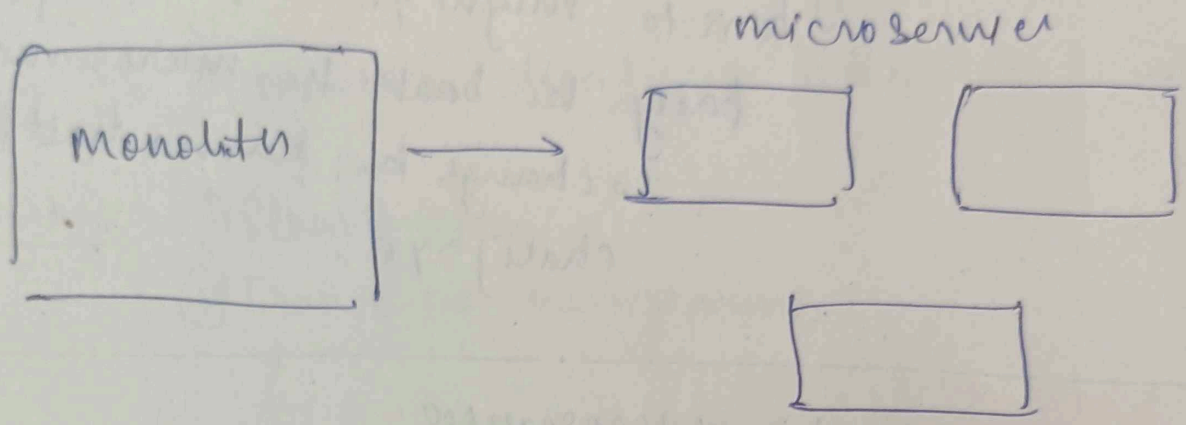
(iv) Deployment (How will your app will be deployed / CI/CD.

(v) Observability (How will we monitor our applicaiton)

Ye DevOps wale dekhte h.

Decomposition

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



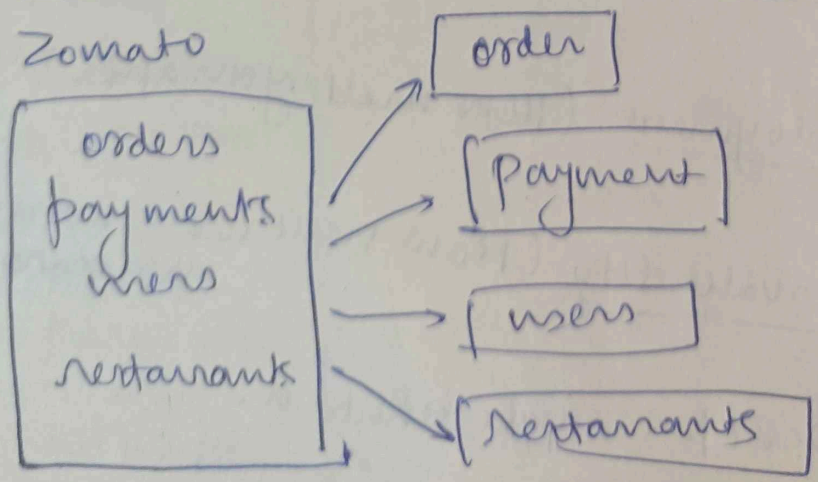
Design patterns of Decomposition:

(i) Decomposition by Business Logic

(ii) ~~Dec~~ " " Sub Domain.

(i) Decomposition by Business Logic

eg: Zomato



matlab jitne entities h unko alag alag bana do.

Disadvantage:

→ Developer should be aware of entire business.

(ii) Decomposition by Sub Domains

Do Tarike alag alag endpoints honge na.

www.zomato.com/orders/ → orders/place-order
orders/select-order
:

→ to un sabka ek microservice

www.zomato.com/payments → payments/initiate
" / make-payment
:

Un sabka ek microservice.

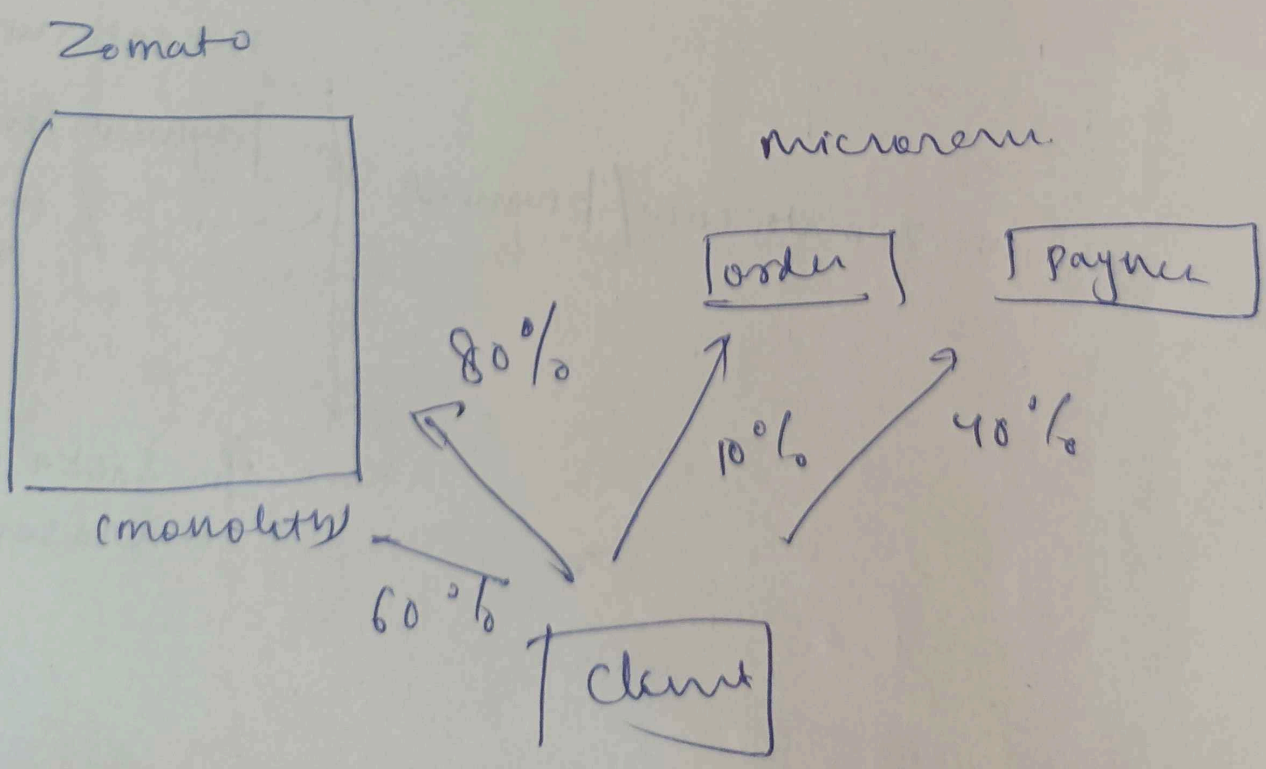
Strangler Pattern

↳ dekha aapne koi application monolith mai run ho rahi h, jisme user aa rahe

⇓

to aapko monolith mai convert karne ke liye aisa nai kar sakte ki, 2 mahine system off karke unn ~~mono~~ microservice kar rahe

aapko rather monolith banate rehna, aur dheere dheere customer ko waha direct karna h.



Jaise ki Monolith h pura zomato,

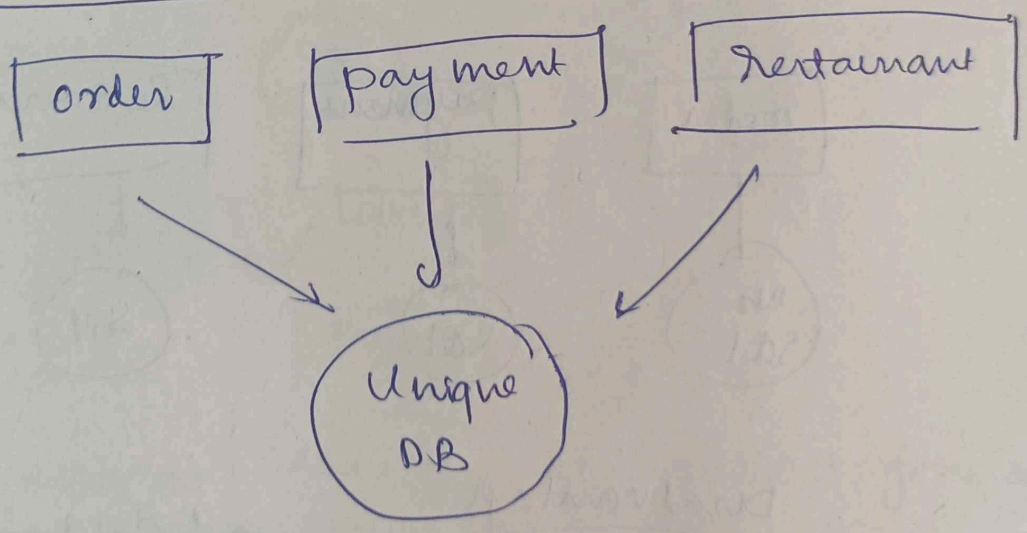
par maan lijie se order aur payment ka microservice bana diya to

monolith	micro
100%	0%
95%	5%
92%	8%

Aise karke dheere dheere micro pe users ko redirect karte jaunga.

(11) Databases

(a) Shared ~~Unique~~ DB



Haan koi data ek hi DB mai daal rahe h.

Advantages:

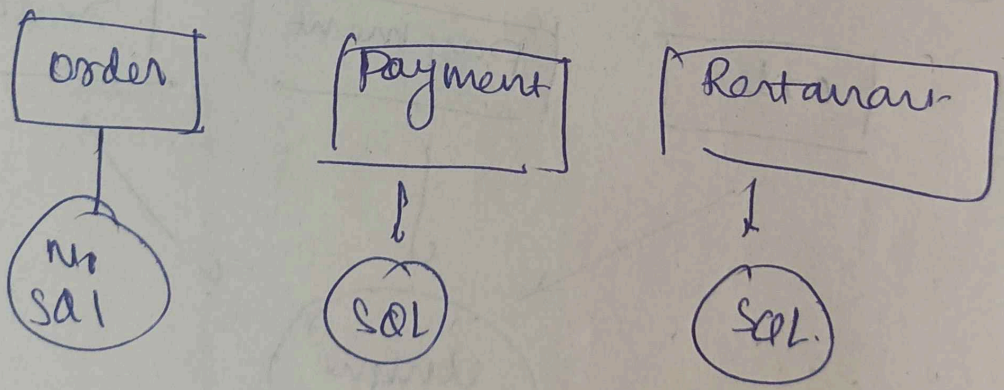
→ Simple to carry operations
(eg: JOIN (SQL))

→ Transactions manage (ACID)
↳ matlab agar koi process
to use revert back karna kafi
aasan h.

Disadvantages:

- Cannot be scaled properly.
→ either can be Sql or No SQL

(b) Unique DB



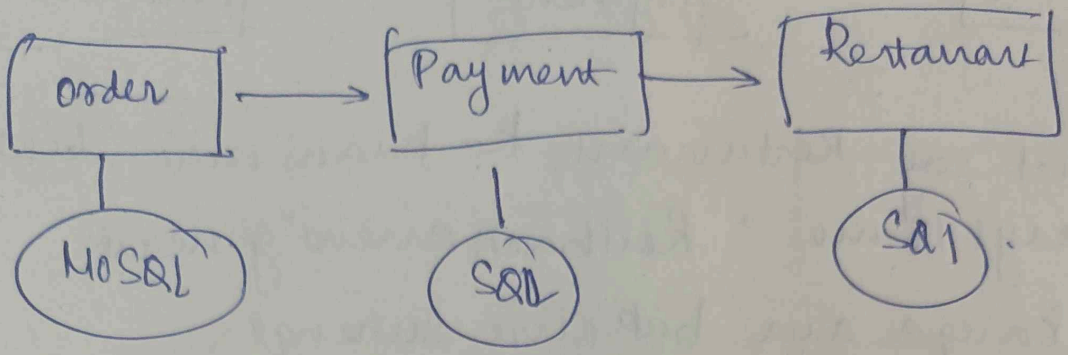
Disadvantages

→ Operations jaise JOIN hard ho jate h
(SQL → (QRS)

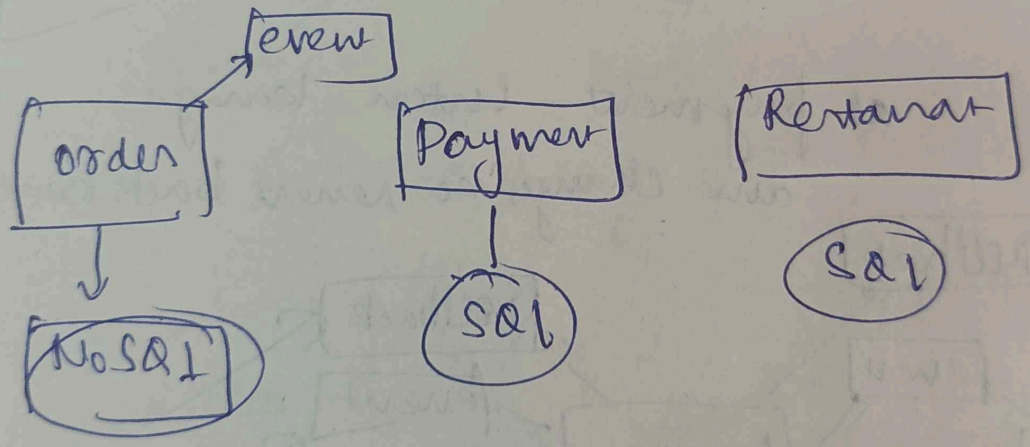
→ Transaction Mangement
↳ SQL → (SDCA)

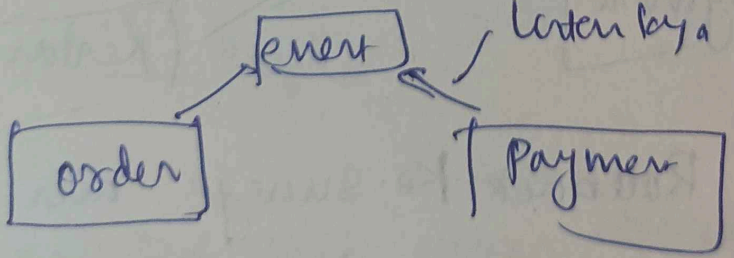
SAGA

① Event Driven System

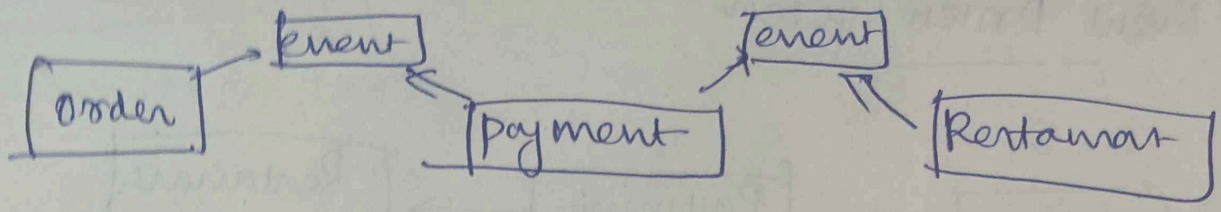


Jaise hi ek microservice apna kaam kar lega, to wo ek event generate karega aur baki use sunge.

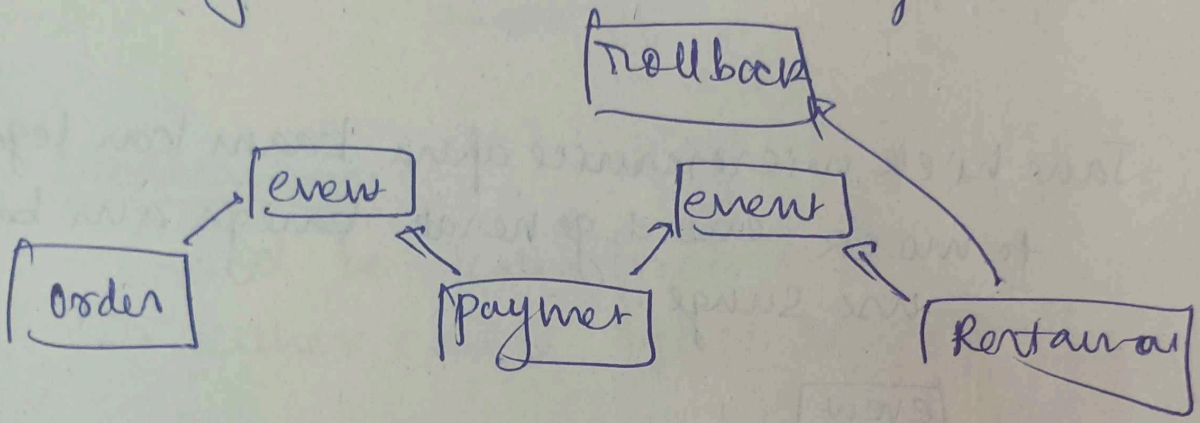




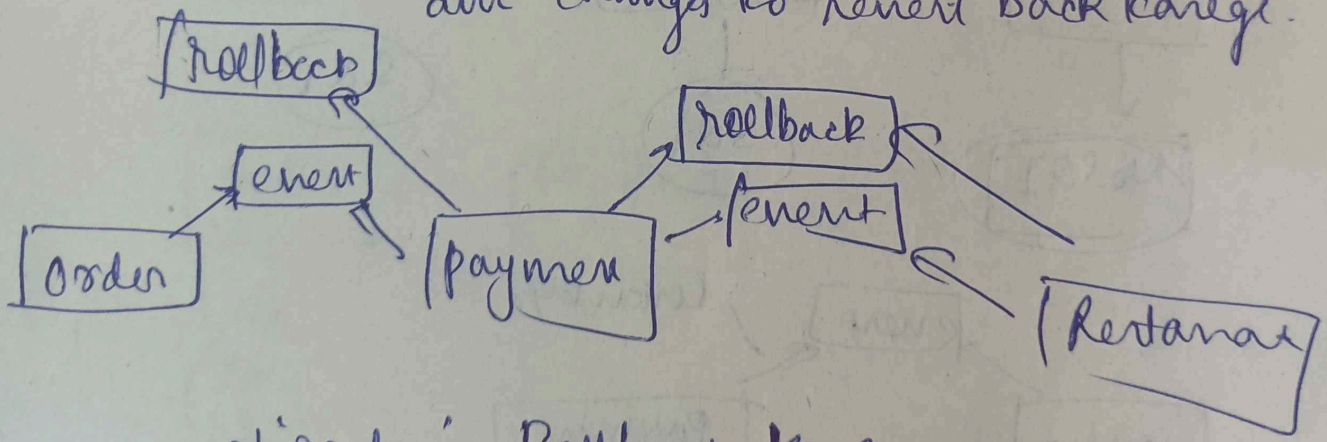
Payment ne kaam kiya to ek event generate karega.



ab let say Restaurant ke kaam mai dikkat aayi to wo 'Rollback event' generate karega aur baki ure sunenge.

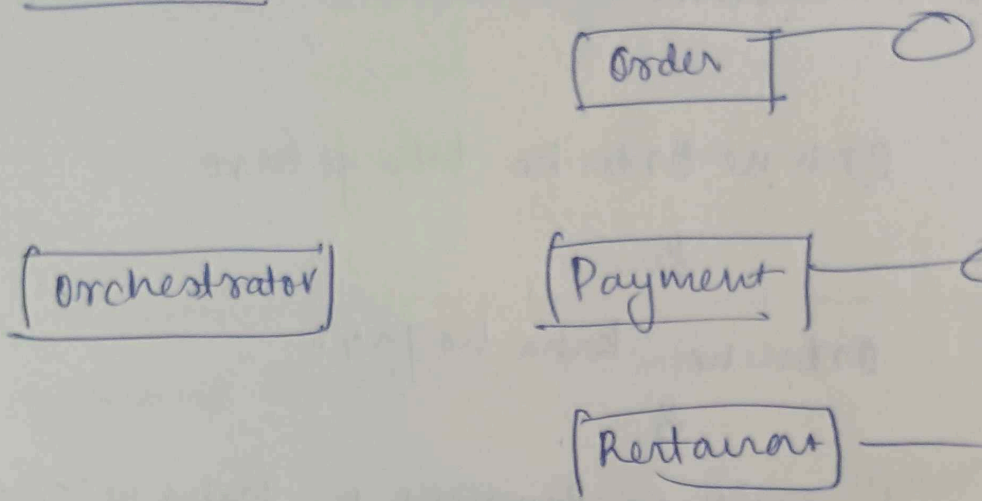


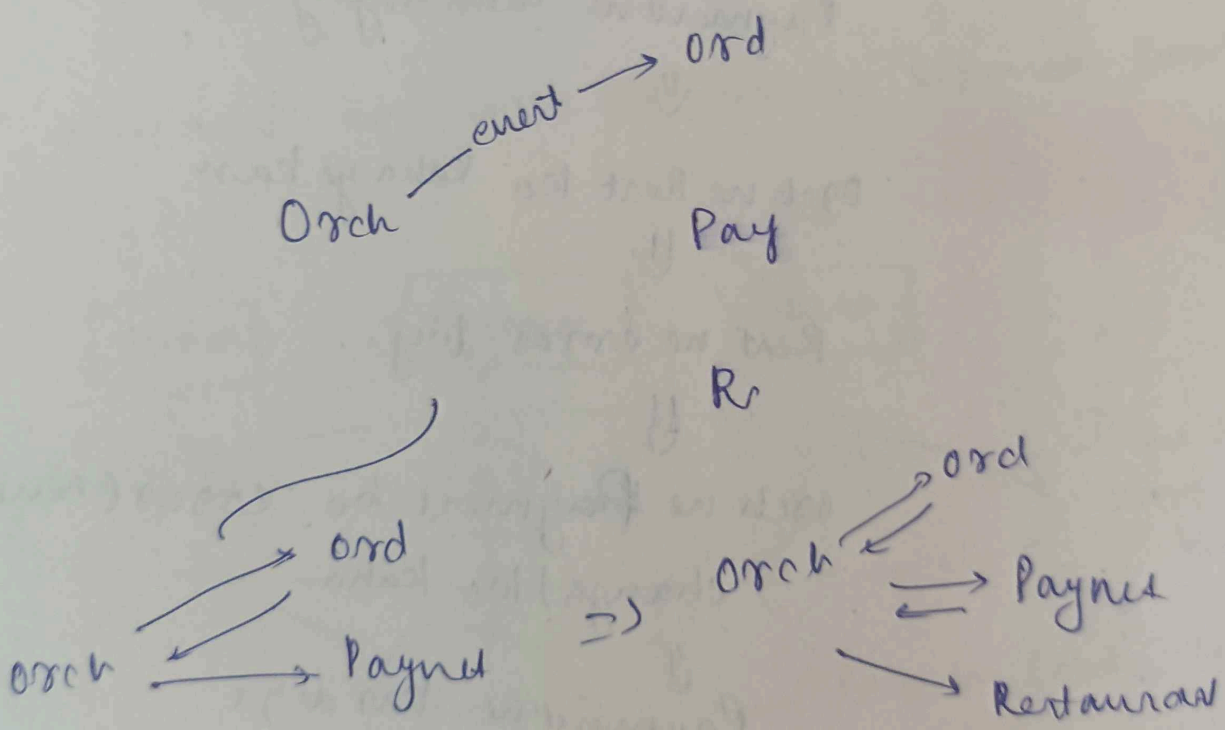
ab payment listen karega.
aur changes ko revert back karega.



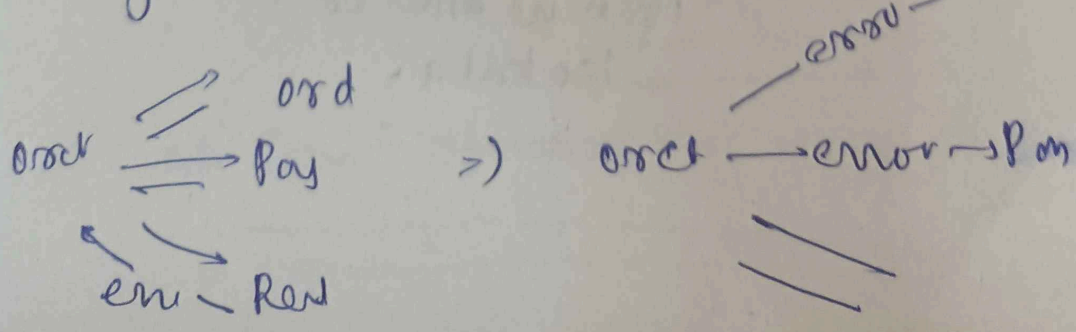
ab 'Order' Rollback ko sunega aur
cheezo ko revert back karega.

(b) Orchestration (API call based)





aga Rest mai error aaya to



Yaha par "orchestrator" saari cheeze yaad rakhta h,

Orch ne Order ko kaha ye karo

⇓

Order ne kaha ho gaya.

⇓

orch ne Payment ko kaha ye kar

⇓

Payment ne kaha ho gaya.

⇓

Orch ne Rent ko kaha ye kar

⇓

Rent ne error diy.

⇓

orch ne Payment ko error (rement changes) ko kaha

⇓

Payment ne kar diy.

⇓

orch ne order ko rement karne ko kaha.

Event-based one-faster, as async h, but harder to build

CQRS (Command Query Request Separation)

Command → write operation (like update, delete, insert)

Query → read operation.

apne read aur write operation ko separate karde