Agenda:-

i) What is HLD, why HLD is important?

ii) Monolith
- advantages
- disadvantages

iii) What are micro-services?

iv) Comm b/w micro-services

v) Distributed transactions

vi) Advantages & Disadvantages of micro-services

vii) When to use what?

Sachin → 2004 → Shipkart.com
↓
[Ecommerce]

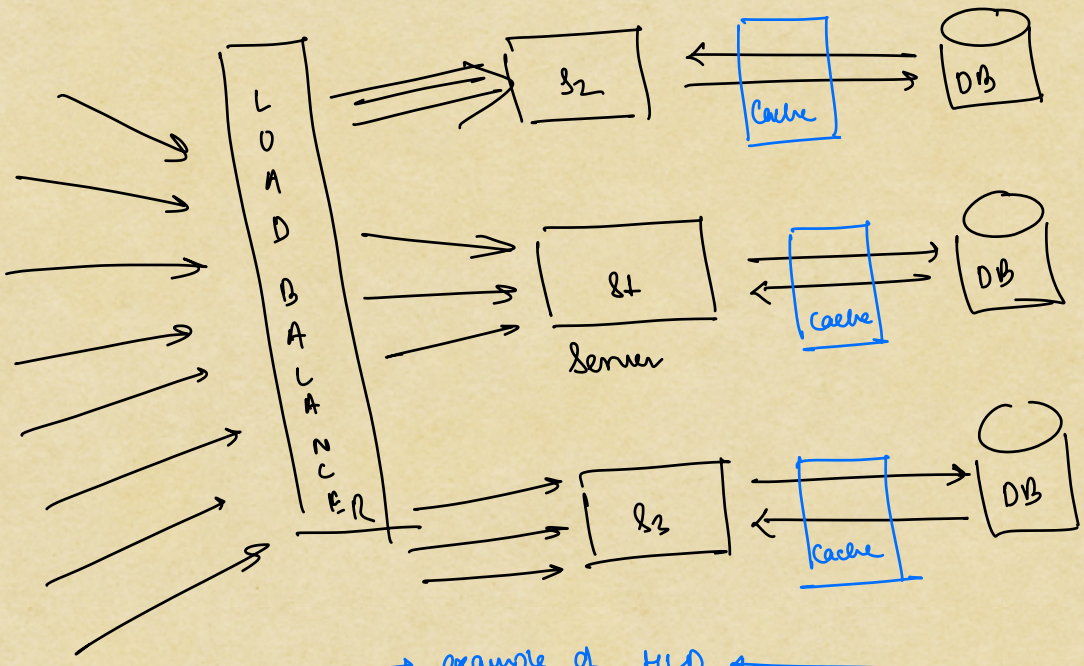
⇒ modules

* Payment
* Catalogue
* Cart
* Account management
* Order
* Shipping | Tracking | Returns | LMS (last mile service)
* Search

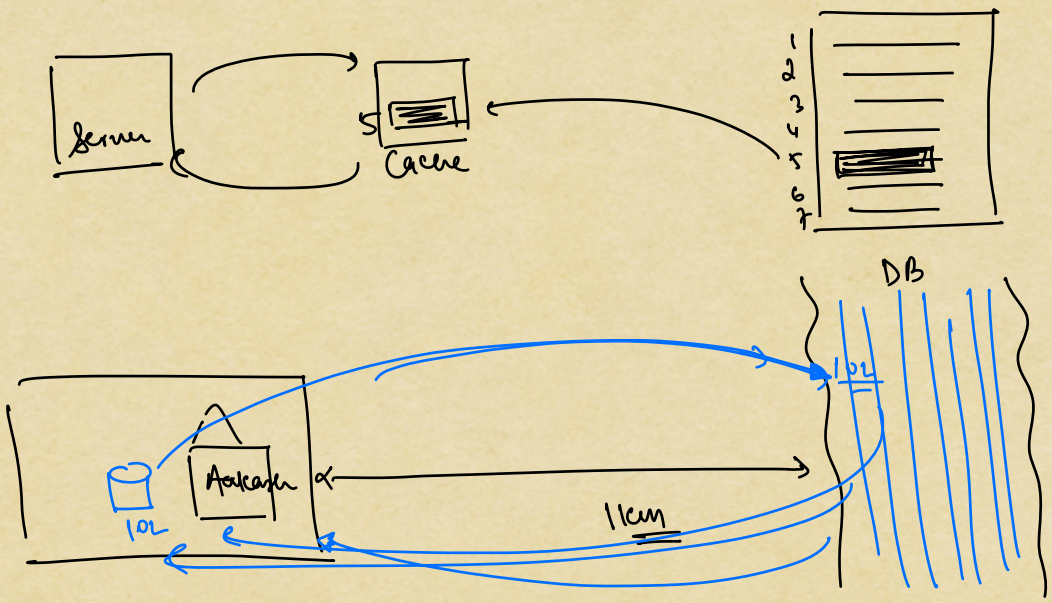
⇒ What is HLD?

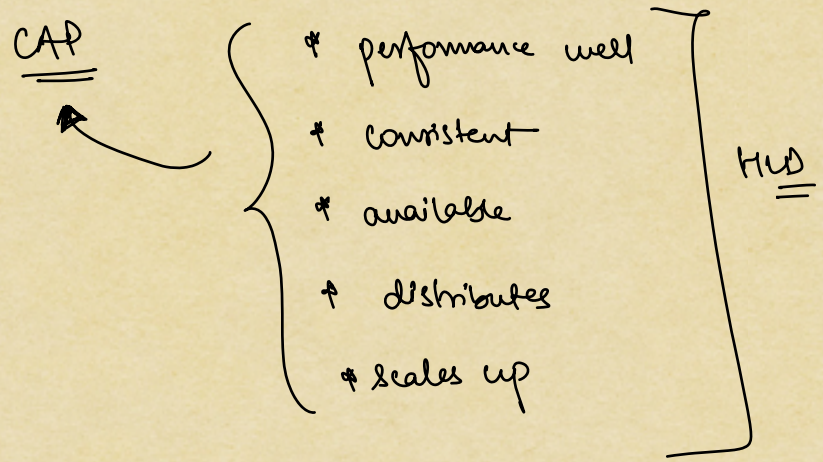
High level Design

* bird's eye view
* Overall architecture



→ example of HLD ←



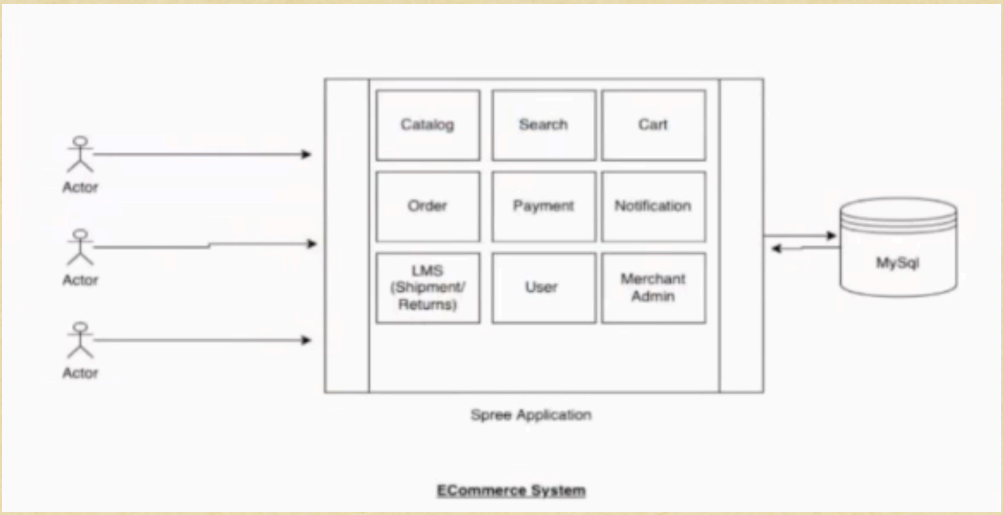


=> Popular ecommerce frameworks :-

i) Magento → PHP

ii) ATh → Java

iii) Spree → Ruby on Rails



2004 ⟶ 2 orders/day

⋮

2008 ⟶ 2000 orders/day

Vertical Scaling

↑ DB / ↑ Server

↓

10 TB

↓

100 TB

↓

1000 TB

↓

10K TB.

* costly
* not available easily
* brings limitation
* single pt of failure

Horizontal Scaling

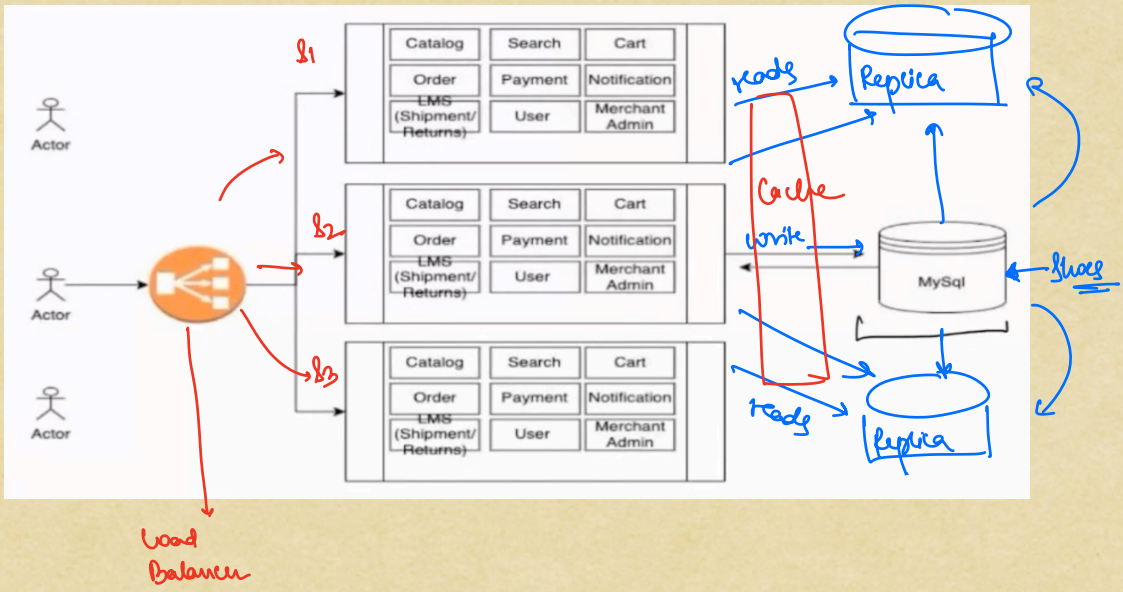
Add more machines

1 — 10 TB

10 — 10 TB * 10 ⇒ 100

100 — 10 TB ⇒ 1000

* cheaper
* easily available because commoditised
* no single pt of failure



Advantages of monolith :

i) Easy to understand and implement

ii) End to end testing is easy

iii) Able to maintain with a small team

iv) Easy to develop for quick launch

monolith
↓
small codebase
+ team
(10-12 ppl)

2008 ⟶ 2000 orders/day

2018 ⟶ 2K orders/min

* huge codebase
* 500+ ppl

* too many features.

Drawbacks of monolith:- → 10.8 MB, 1 cr lines, 1.2 lakh classes

* Developer onboarding → complex code
* Making changes are difficult → tightly coupled code and minor changes can bring entire thing down
* Selective scaling is not possible.
* Inefficient use of resources.
* Barrier to new tech stack
* Difficult to debug
* Takes a long time to develop new features
* [Big Ball of mud] ← nobody knows end to end everything

Monolith: An appn consisting of one system where all the codebase, business logic, DB are interconnected and dependent on each other

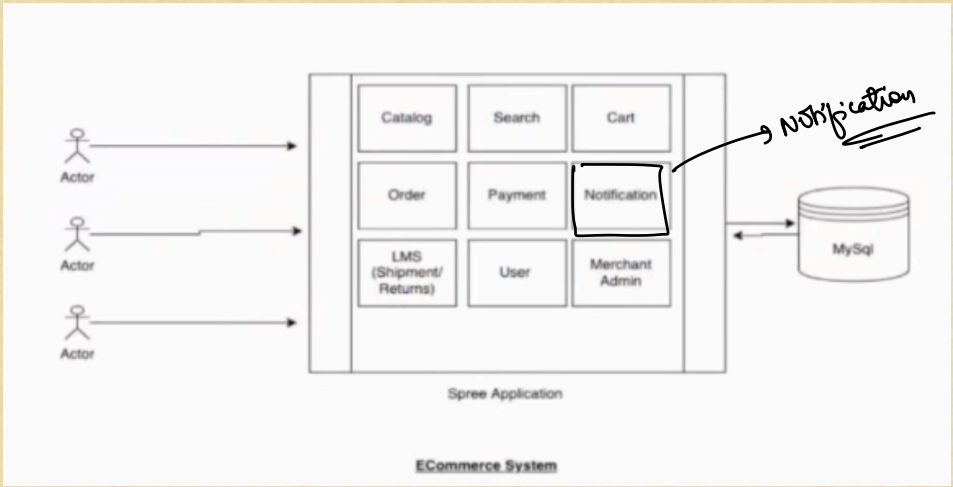
→ CTO → monolith to micro-services

↓

Chief Technology Officer

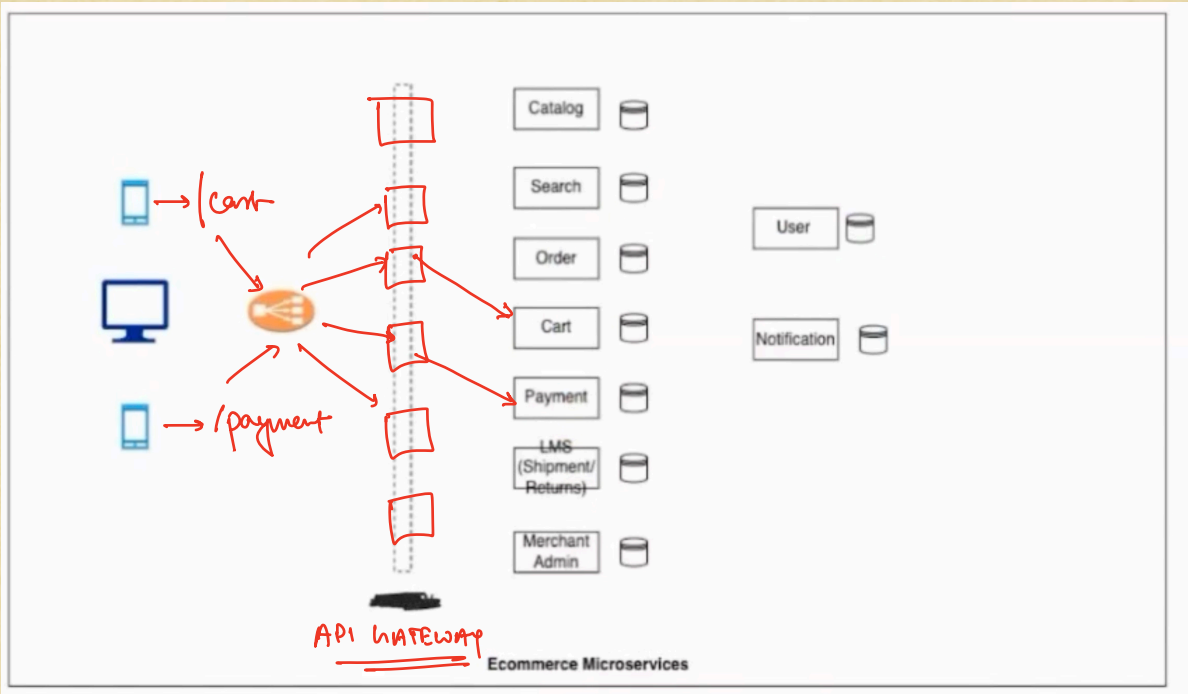
⇒ Moving from monolith to micro-services:-

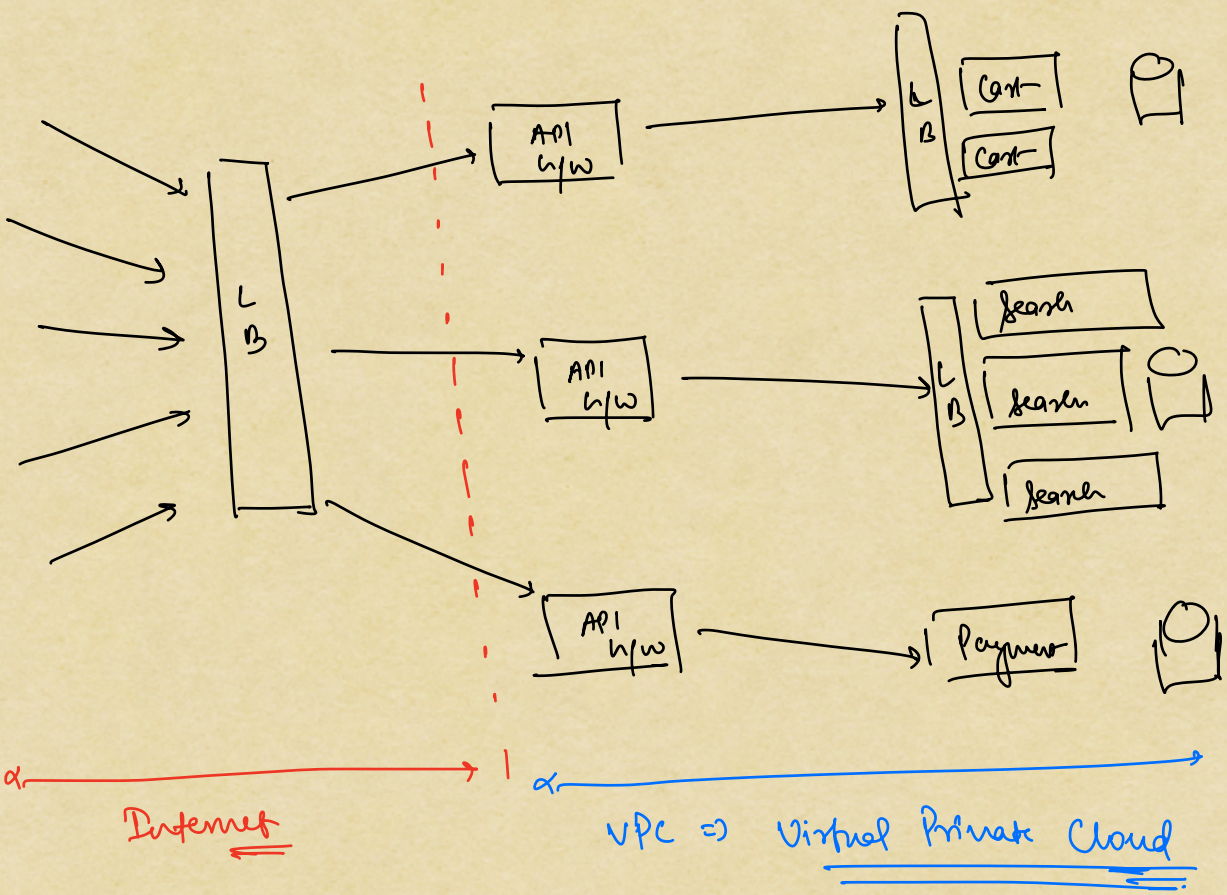
→ Decompose the modules



→ Decompose modules:-

* Based on scale
* Not that tightly coupled
* Business Unit (BU)


/cart
/payment
Catalog
Search
Order
Cart
Payment
LMS (Shipment/ Returns)
Merchant Admin
User
Notification
API GATEWAY
Ecommerce Microservices


L B
API G/W
Cart
Cart
API G/W
Search
Search
Search
API G/W
Payment
Internet
VPC => Virtual Private Cloud

Advantages of micro-services:-

i) Independent components => failure isolation.

ii) Easy to add new features.

iii) Faster development

iv) Selective scaling

v) Onboarding of developers becomes easy

vi) Small team, clear ownership

vii) Easy to debug

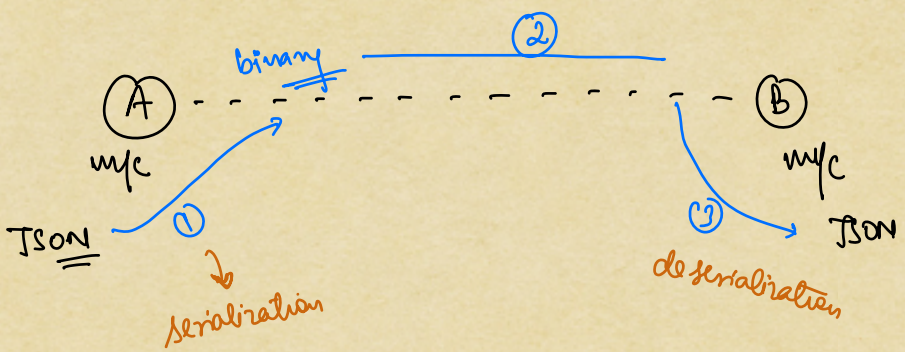
viii) Easy to introduce new tech stack

Communications b/w micro-services:-

i) REST API [HTTP] → slow due to multiple steps involved

POST /order

Payload
JSON



2: RPC => Remote Procedural Calls

↓

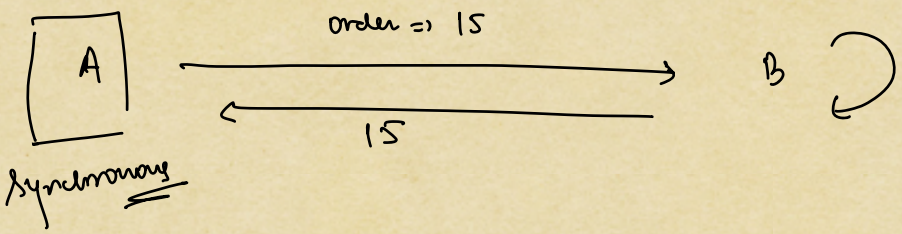
Protobuf → binary

↓

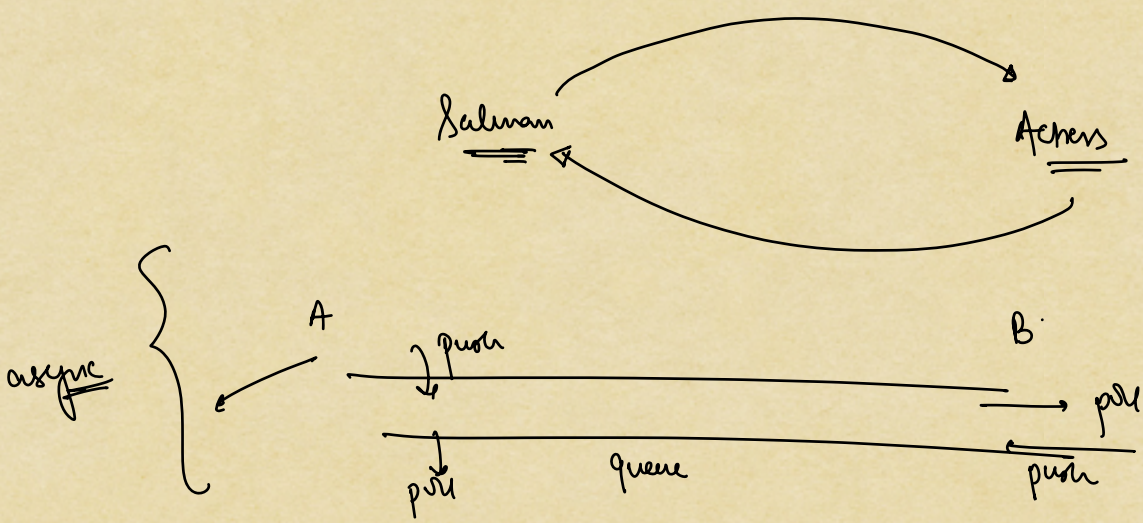
faster than REST API

gRPC => Google RPC

3 Events



Main Pyar kiya



async / event ⇒ message queues

→ Kafka, RabbitMQ,

Aws sqs, Tibco, ...

* Observer design pattern:

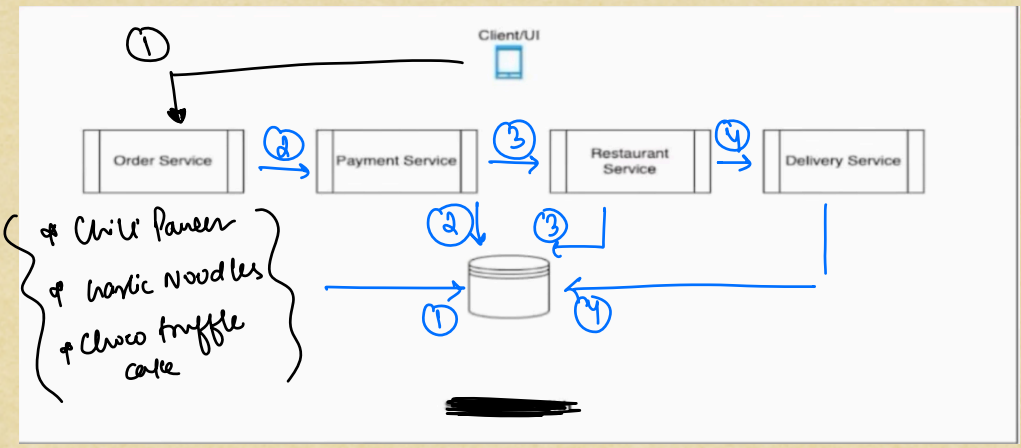
pub/sub model

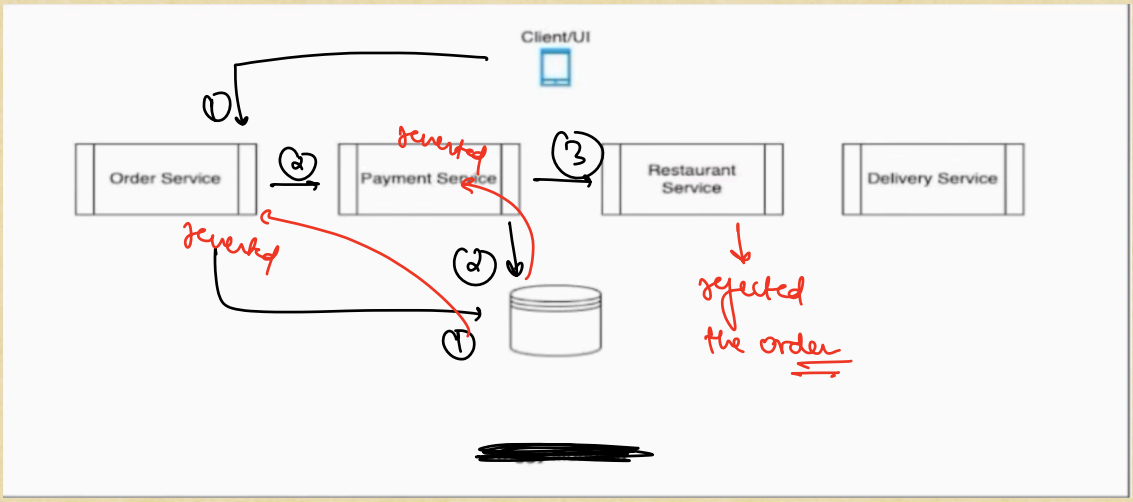
publishers

subscribers

Distributed Transactions:

Food Delivery ⇒ Ziggy



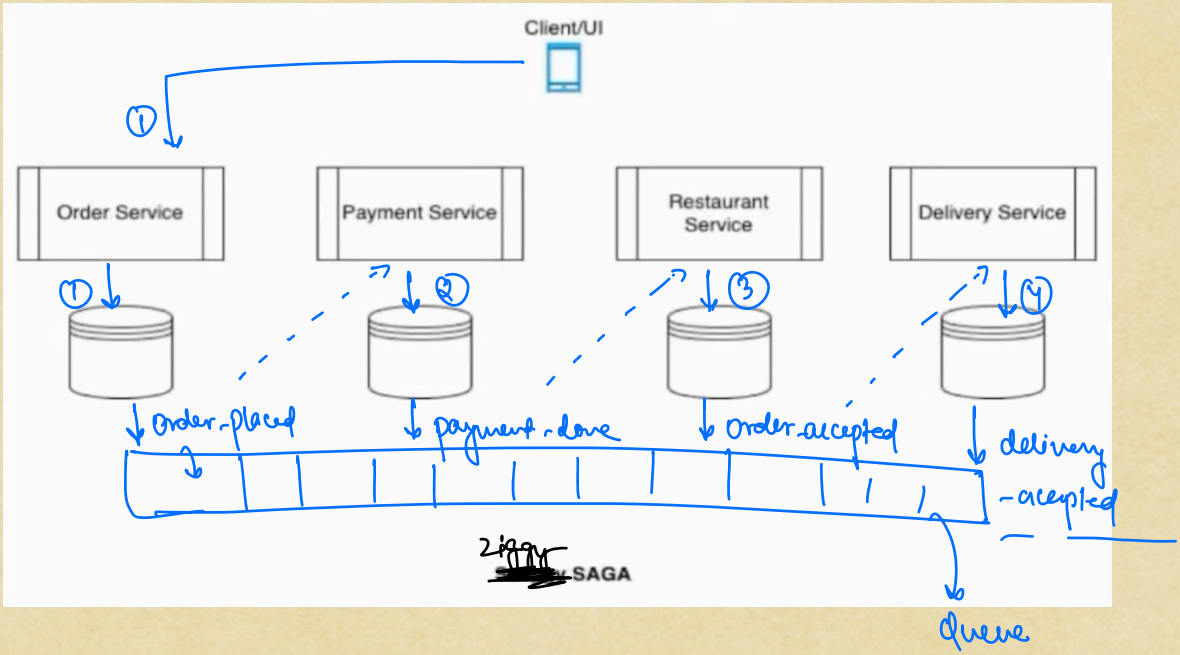


Saga Design Pattern

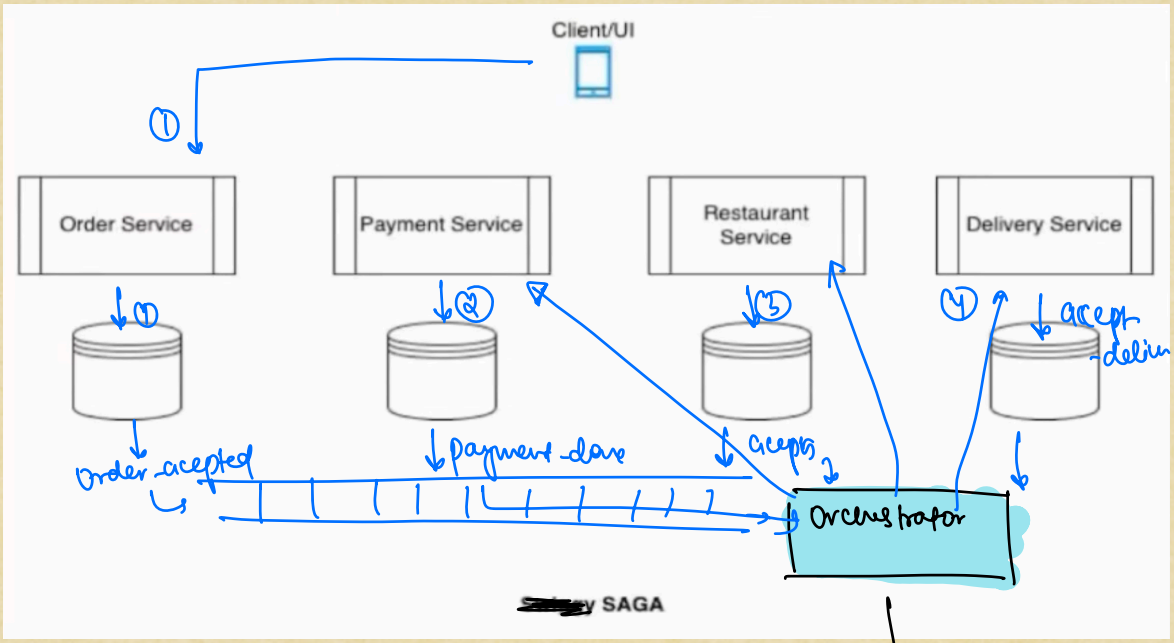
i) Orchestrator → someone is controlling the flow

ii) Choreography → performs on its own

Choreography :-



Orchestrator :-



Create replications to handle load and remove single point of failure

Producer

- Manager
- Architect
- Senior
- Junior

0-3 → SDE: 1 → 20-35 LPA

3-6 → SDE: 2 → 35-60 LPA

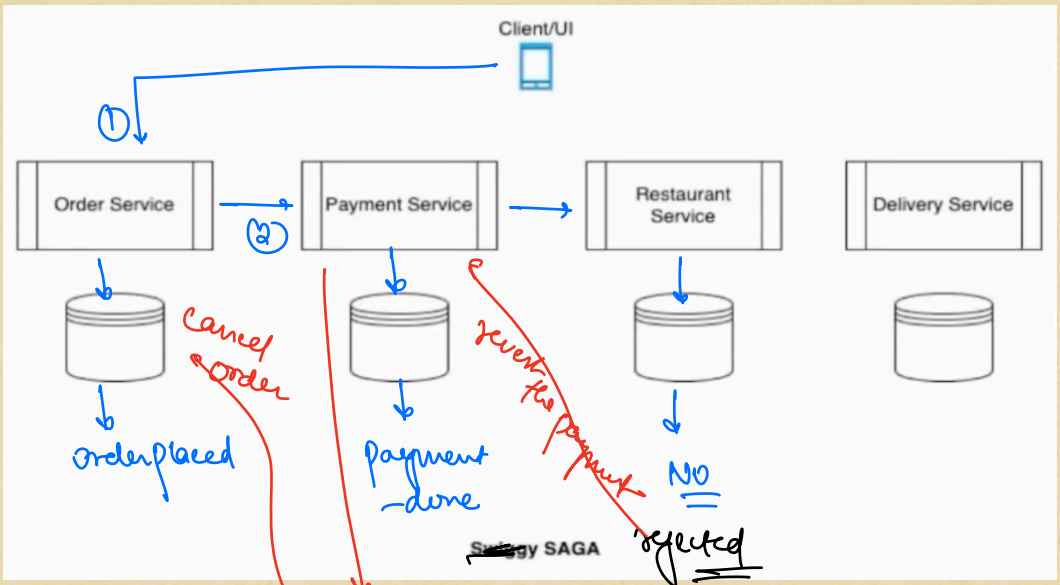
7-10 → SDE: 3 → 60-90 LPA

10+ → Manager / Architect → 1 Cr+

Handling failures:

* Compensating transactions :

taking care of expected failures.



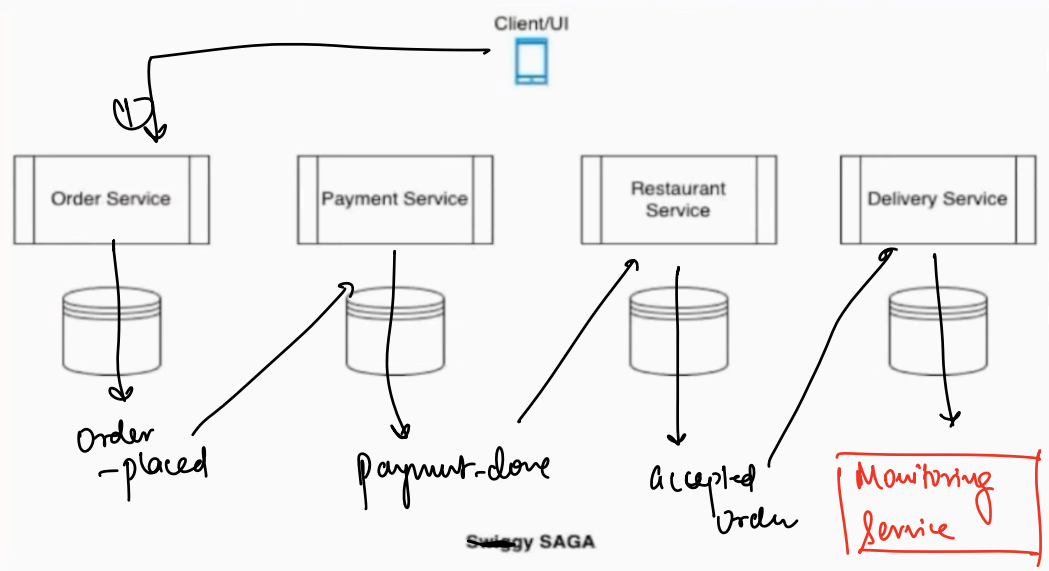
NO/REJECT

negation events

for expected negation events we create compensating transaction to revert changes.

* Self compensating transactions :-

handling unexpected events



↓

triggers compensating transactions after some delay / timeout / failed healthcheck

Disadvantages of micro-services :-

* Difficult to build, need highly skilled engineers.
* Heavy dependency on devops.
* Lot of monitoring required
* N/w hops => increase in latency

=> When to use monolith

- Small team
- Simple appn
- Low scale
- Quick launch
- Low experience

=> When to use micro services

- High scale
- Complex features
- Big & talented engg. team
- devops team
- High scalability. availability
- Faster development

{ Start with monolith but with the intention of moving to μ services }