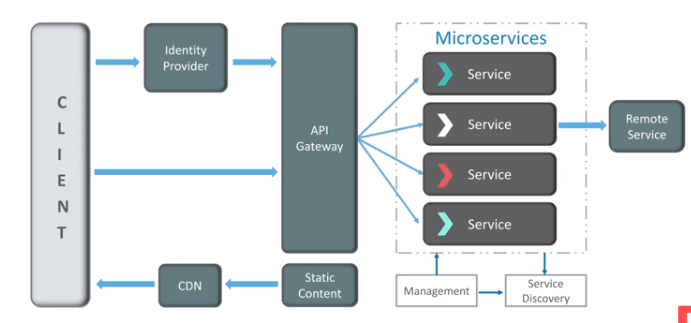
**Micro Services**

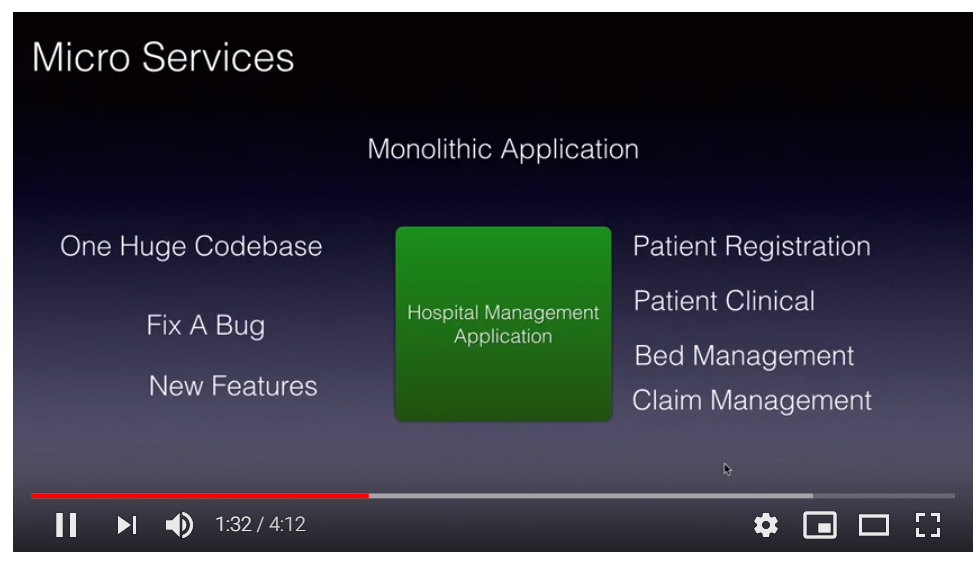
* Monolithic application have too many problems that is

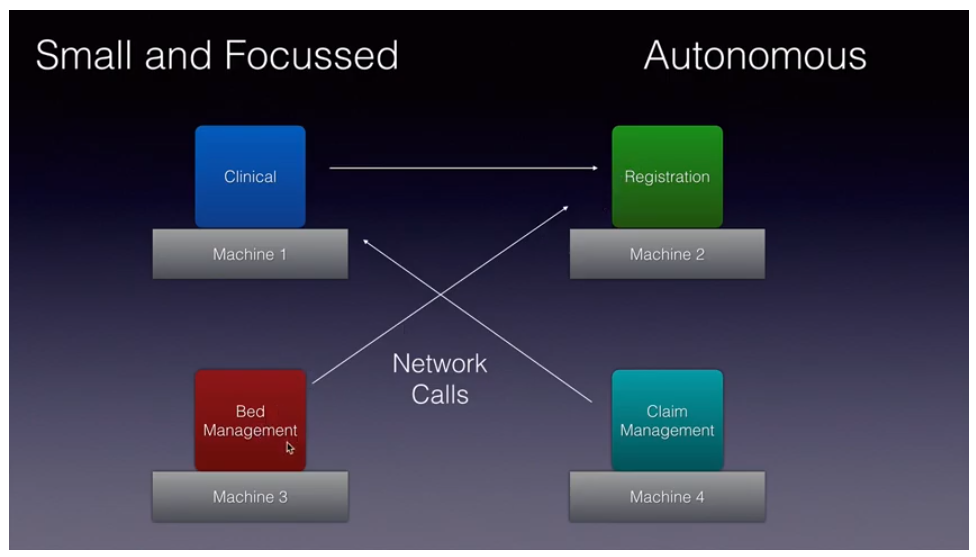
1. large n complex application ,
2. slow development(understand the code and modify)
3. tightly coupled so one service goes down then entire application goes down
4. if we update in one service then we have to deploy all services
5. difficult to adopt new framework.



* Microservice Architecture







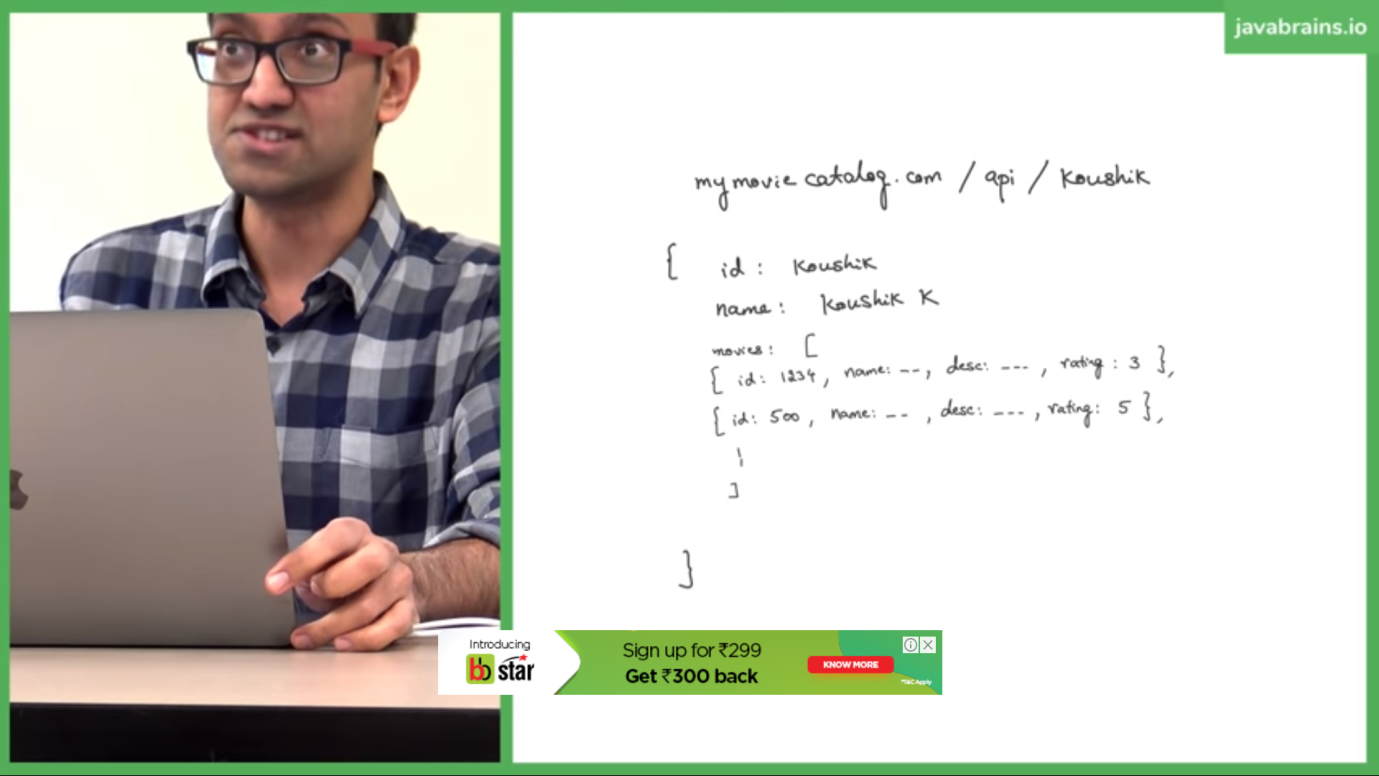
1. Microservices is a form of service oriented architecture style wherein applications are built as a collection of different smaller services rather than one whole app.
2. Collection of small autonomous services, modelled around a business domain and all services communicate stateless through API.
3. Each service is self contained and implements a single business capability.
4. Loosely coupled and independent with each other.
5. We can develop each services in different language.
6. **Management** is responsible for placing services on node, identifying failure and rebalancing services over node and so on.
7. **Service discovery** is used to maintain the list of services and which node that is located on. It enables lookup to find the end point of service.
8. **API Gateway** is the entry point of the client and it will forward the call to appropriate service on the backend. Also it will send the single response or aggregate response.

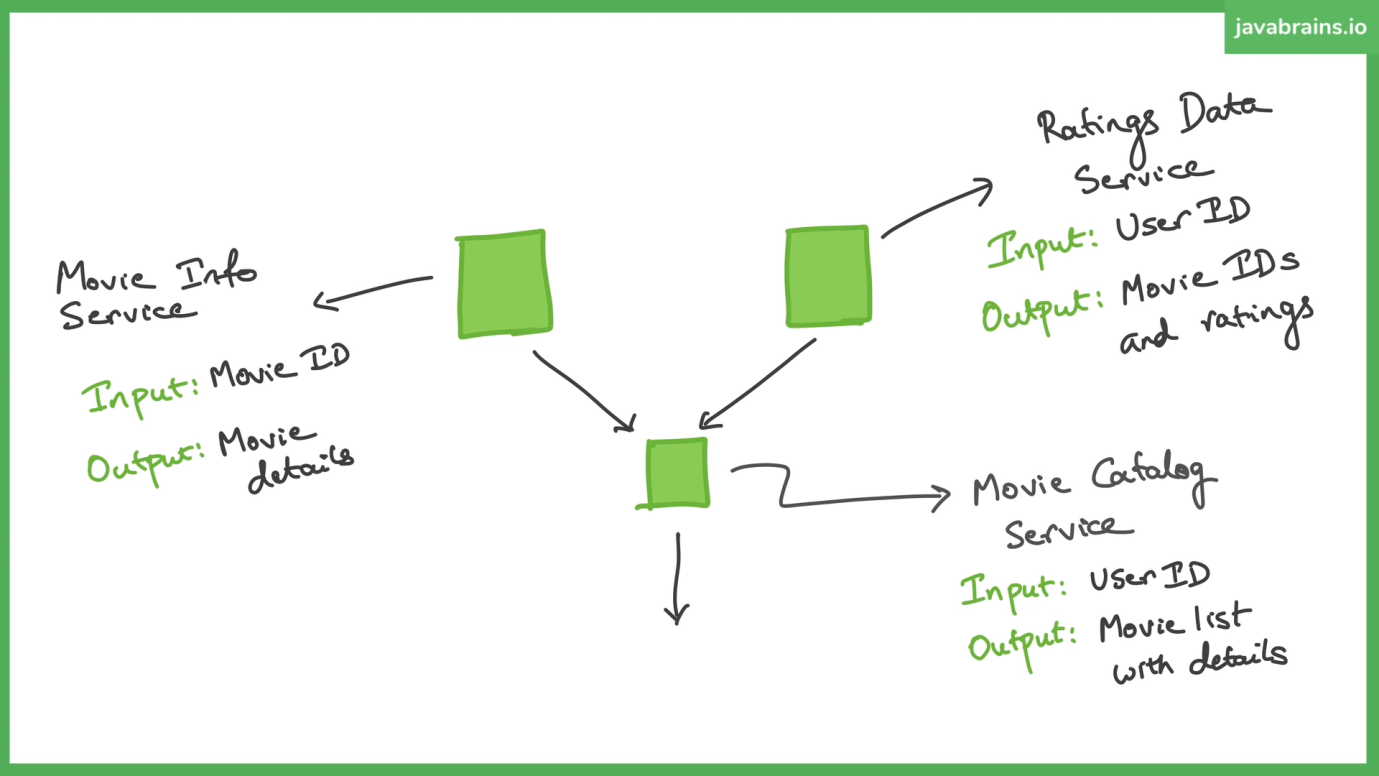
* **Features**

1. Small Focused
2. Loosely Coupled
3. Language Independent
4. Bounded Context(No need to understand the other implementation)

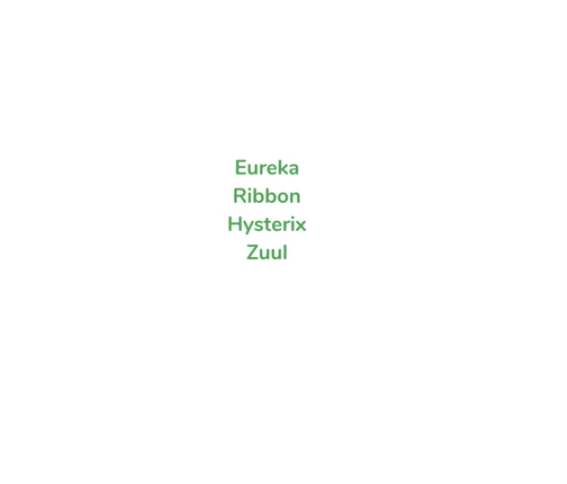
* **Advantages**

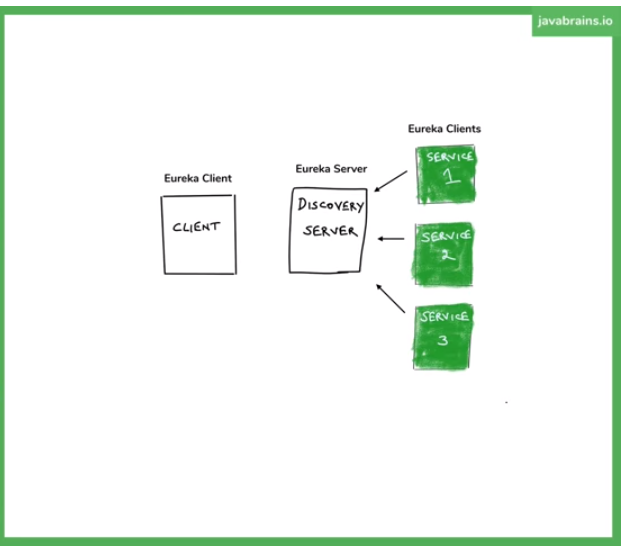
1. Independent development
2. Independent Deployment
3. Fault Isolation(One service got down so only that service will goes down other services will work).
4. Mixed Technology
5. Granular Scaling(can scale one service instead of scaling entire application)

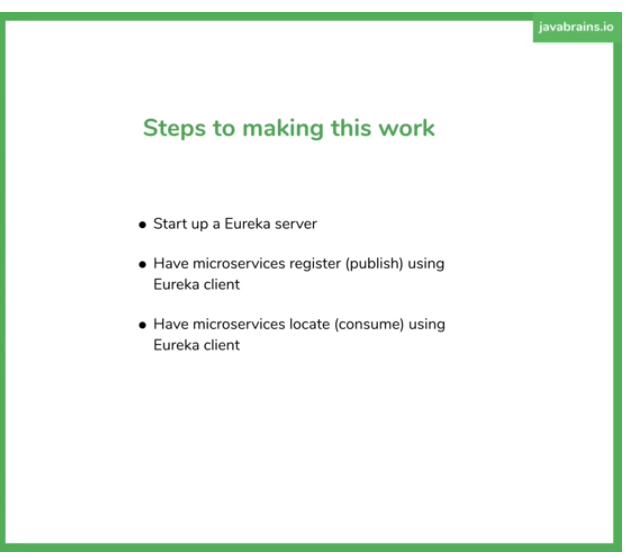




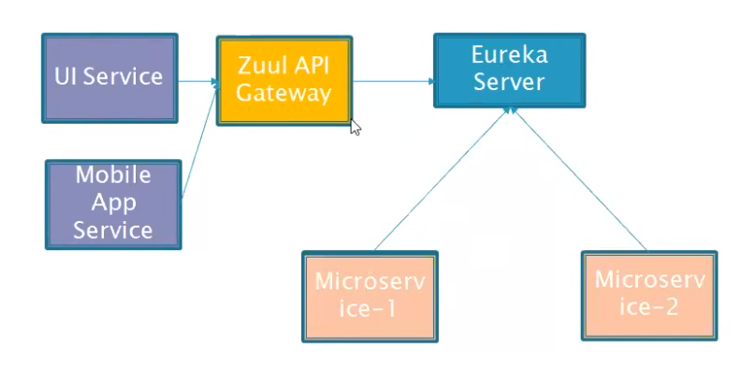


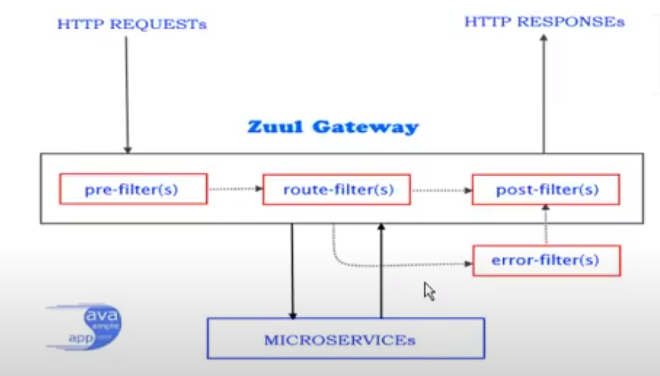


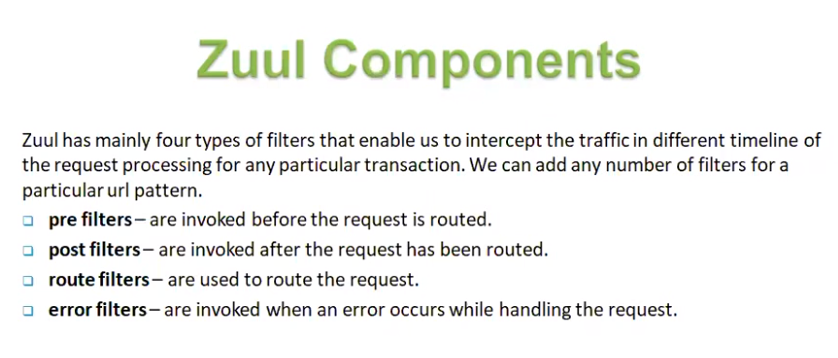




Zuul Gateway







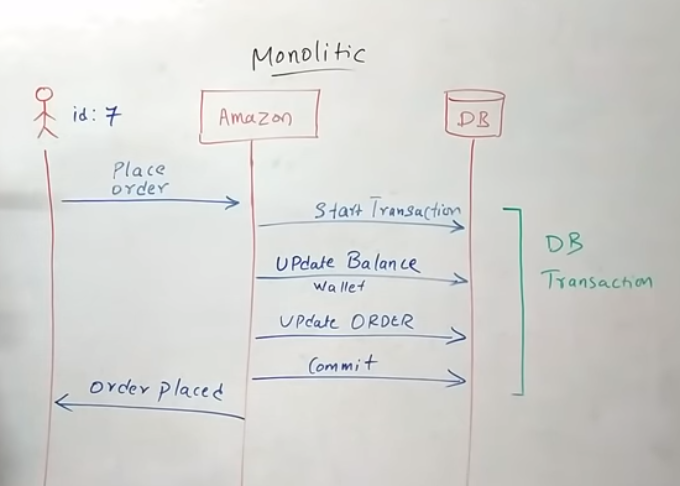
**Hystrix Circuit Breaker**

->It’s used to return some message for circuit breaker. It means if any microservices has goes down then it will return some message stating that service is down, Service will be back shortly.

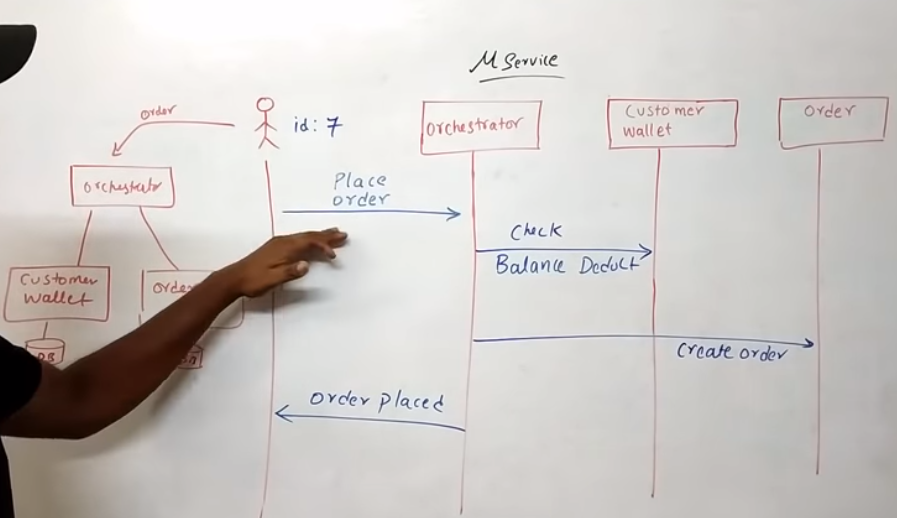
# Spring cloud ribbon with eureka

* Used to load balancing at client side.

**Transaction in Microservices using Sagas**

****

**Distributed Transaction**

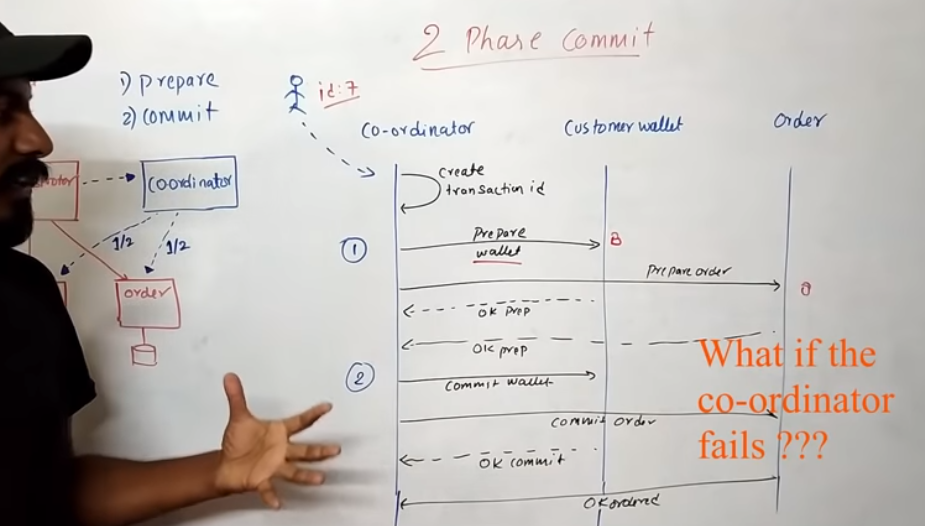
****

Two ways to handle distributed transaction.

1. Two phase commit

Advantages : strong consistency

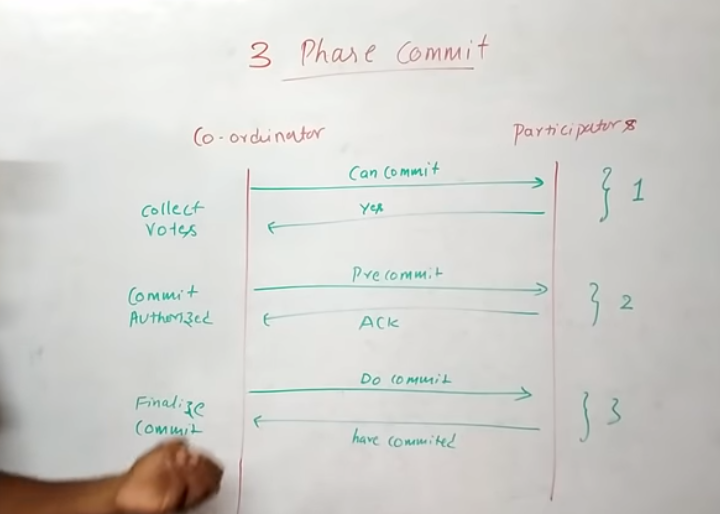
Disadvantages : coordinator fails, Locking time is more



1. 3 phase commit

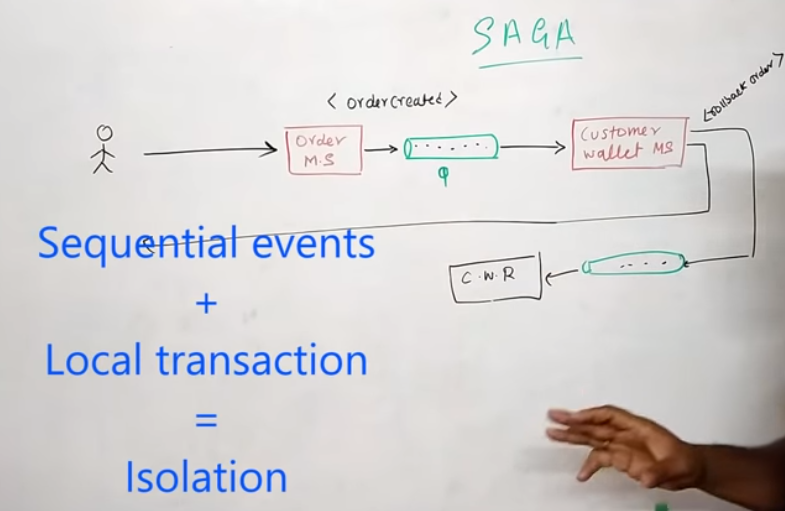
Advantages : strong consistency

Disadvantages : Locking time is more



1. SAGA

* Works as Asynchronous



**Service Mesh(Microservices design pattern)**

* **Suppose I am migrating from our monolith application to microservices application so we face many many challenges.**
* **We have to do analysis that how many modules we are using and based on that we can create microservice for each module and put services like authentication,authorization,cicuit breaker,eureka server/client,distributed transaction and so on. After that we have to containerization using docker to deploy from dev to prod env.**
* **So I pulling out all the repeatative services and putting into single component and that will embedded with the container. this is called sidecar proxy.**
* **It’s a mechanism or platform by which do not have the write any code to handle those challenges like service discovery , load balancing, multi protocol support and so on..**
* **All will take care by service mesh for example Istio+envoy**