# **Saga Design Pattern ->** It is used to maintain data consistency in microservices architecture.

# 

***Problem Statement ->*** Assume scenario of Online Shopping like amazon or flipkart where you build application on microservices Architecture.

In this scenario you have

1. Order Microservices
2. Product Microservices
3. Cart microservices
4. Payment Microservices
5. Shipment Microservices

Now When ***User placed one Order*** Ther Request will be completed by Calling All microservices.

**Step 1** -> When User Place One Order it will go to Order Microservices.

**Step 2** -> Order Microservices will take the request, save it to local MS DB and send it to Product MS.

**Step 3** -> Product microservice take the product, save it to local DB and decrease the product count from overall product count and send it to cart microservice.

**Step 4** -> Now in Cart Microservice if User wants to proceed then it will proceed to payment MS.

**Step 5** -> Now in Payment MS if Payment fails how we can roll back the the transaction from each MS as each MS maintains their own DB .

With SAGA design pattern we can achieve Transition in all Microservices.

## ***SAGA*** -> In a simple term we can say SAGA is a sequence of local transactions.

***Important Points -***

1. Each transaction to a particular service is treated as a local transaction. So we have a sequence of local transactions.
2. Each local transaction (saga) updates data within a single service.
3. This design pattern can be sequential or parallel. Means synchronous or asynchronous.
4. This design pattern is preferred for long running transactions.
5. Mainly 2 types of implementation of SAGA pattern mostly used.

***1. Event Based approach / Choreography Based approach.***

***2. Orchestration based approach / Command Based***

## **Event Based approach / Choreography Based approach. ->** In Event based approach

**Each** microservices do their task on the basis of the event means

1. Each microservices will listen the Event
2. Do the business operation, save required details in DB
3. Then trigger the event.
4. Another MS will list that step 3 event and start processing .

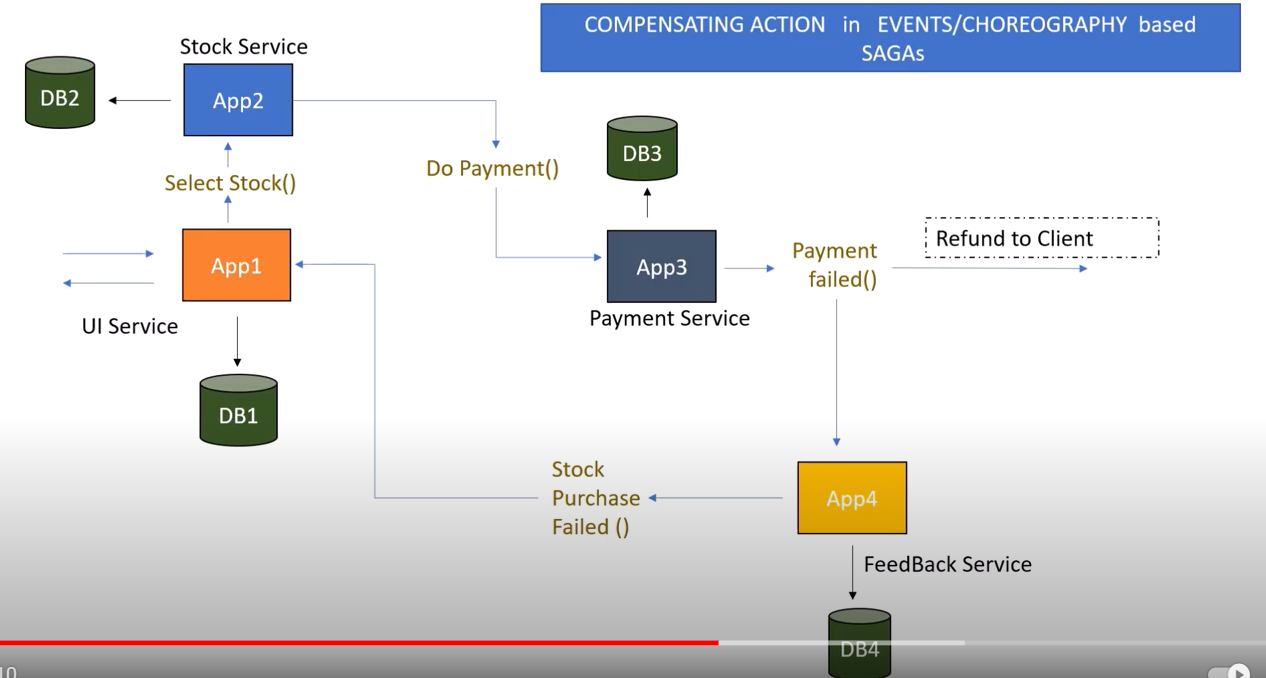
Ex Happy Flow ->

1. Order MS -> createOrderEvent -> now Order MS will send event -> processOrderEvent
2. Product MS -> will listen/subscribe event -> processOrderEvent-> Product MS do some business operation -> send event -> itemsAddedToCartEvent
3. Cart MS -> will listen/subscribe event -> itemsAddedToCartEvent> Cart MS do some business operation -> send event -> proceedToCheckoutEvent
4. Payment MS -> will listen/subscribe event -> proceedToCheckoutEvent> Payment MS do some business operation -> send event -> paymentDoneEvent
5. Order Fulfilment MS -> will listen/subscribe event -> paymentDoneEvent-> Order Fulfilment MS -> do some business operation -> send event -> orderPlacedSuccessEvent
6. User MS -> will listen/subscribe event -> orderPlacedSuccessEvent> User Fulfilment MS -> do some business operation

Ex UnhappyFlow ->

1. Order MS -> createOrderEvent -> now Order MS will send event -> processOrderEvent
2. Product MS -> will listen/subscribe event -> processOrderEvent-> Product MS do some business operation -> send event -> itemsAddedToCartEvent
3. Cart MS -> will listen/subscribe event -> itemsAddedToCartEvent> Cart MS do some business operation -> send event -> proceedToCheckoutEvent
4. Payment MS -> will listen/subscribe event -> proceedToCheckoutEvent> Payment MS do some business operation -> send event -> ***paymentFailedEvent***
5. Cart MS -> will listen/subscribe event -> ***paymentFailedEvent***> Cart MS do some business operation -> roll back the DB -> send event -> checkOutFailuerEvent
6. Product MS -> will listen/subscribe event -> checkOutFailuerEvent> Product MS do some business operation -> roll back the DB -> send event -> orderProcessingFailuerEvent
7. Order MS -> will listen/subscribe eventorderProcessingFailuerEvent->roll back the DB -> Order is failed, try again after some time.

**Digram**

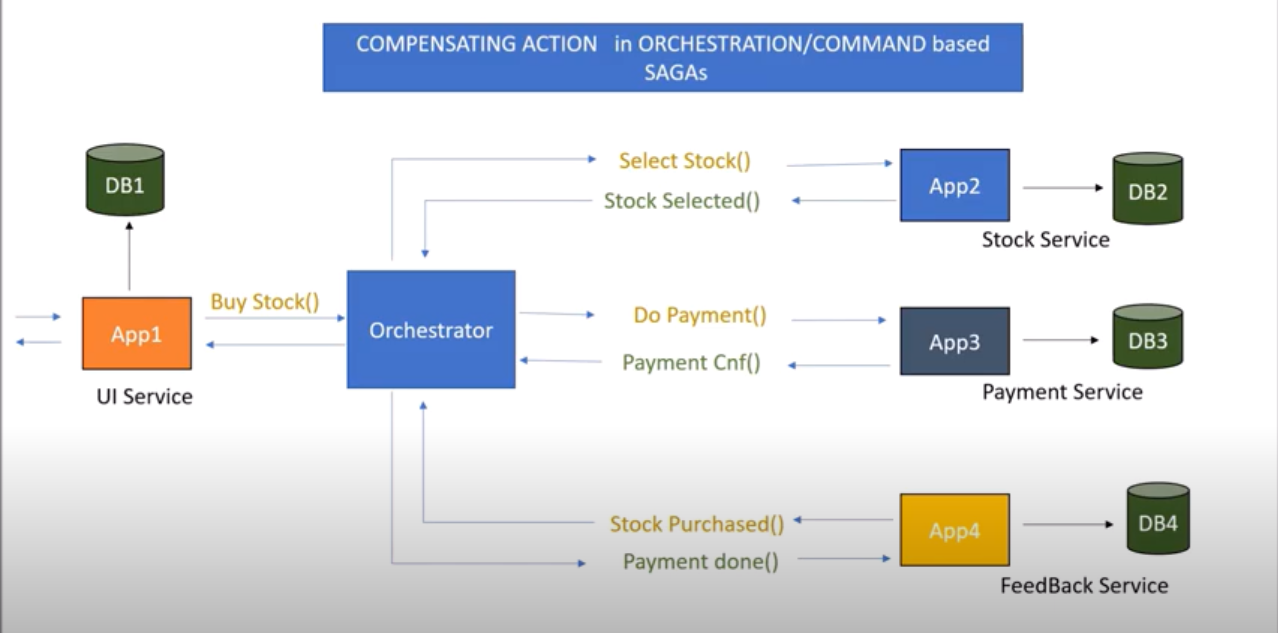


## ***Orchestration based approach / Command Based***

In Orchestration Based approach the MS will not interact with each other directly instead of that we have common orchestrator to perform such action

Ex ->

1. From UI one trigger will happen after clicking on the place order button.
2. From that point request will go to Orchestrator.
3. Now from point 2 it will do command processOrder()
4. Now processOrder() command will be listen by Order MS -> do some business logic -> trigger selectProduct() command
5. After getting selectProduct() command orchestrator will trigger command -> addToCart()
6. Like that process will happen



**Digram**

