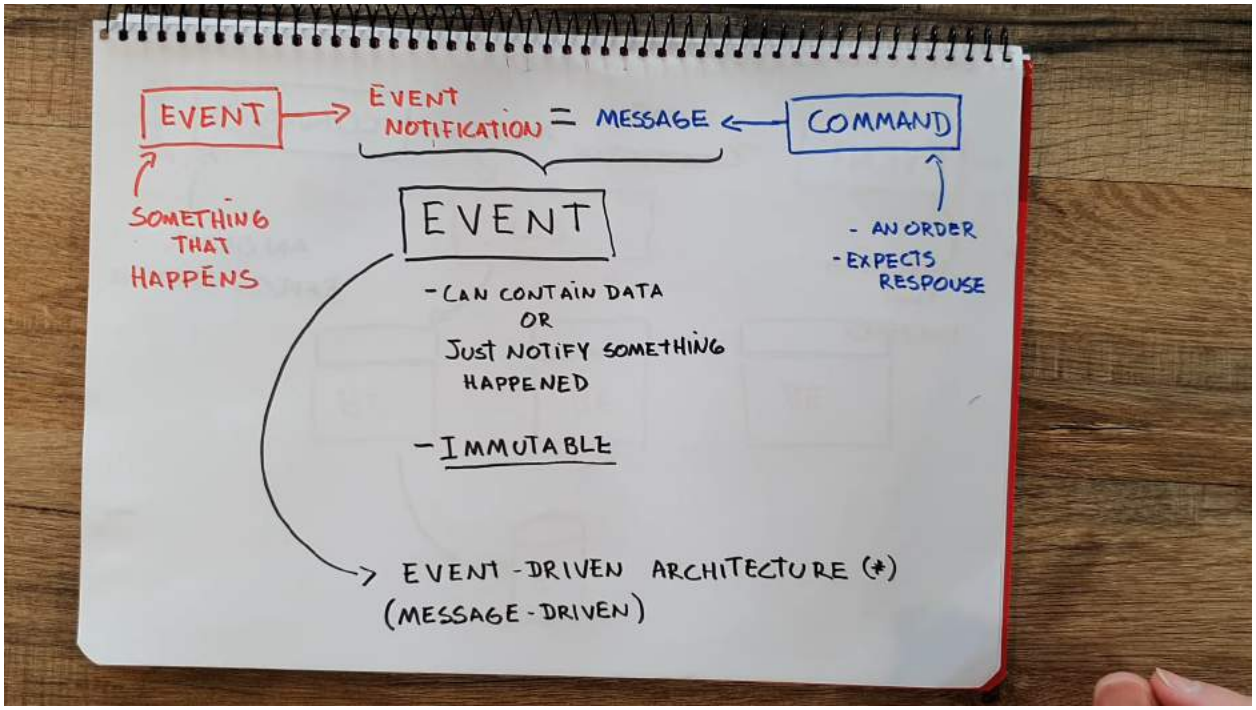
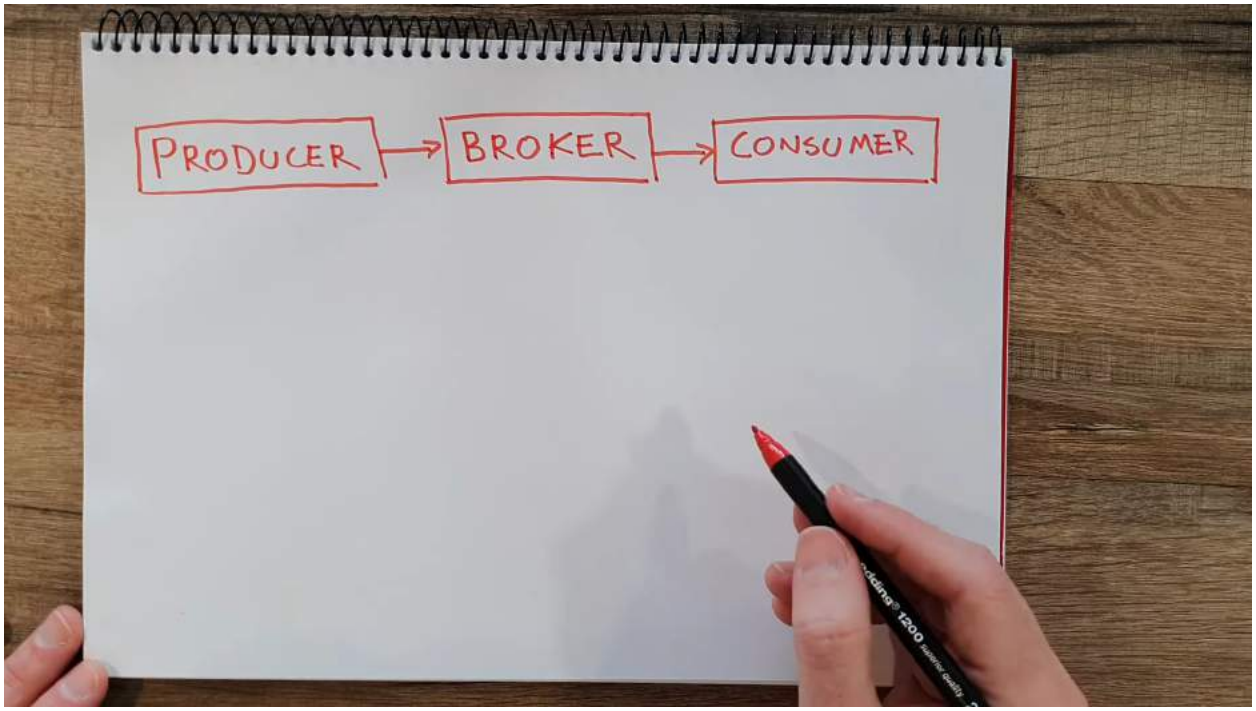


Event: Something happened in the past

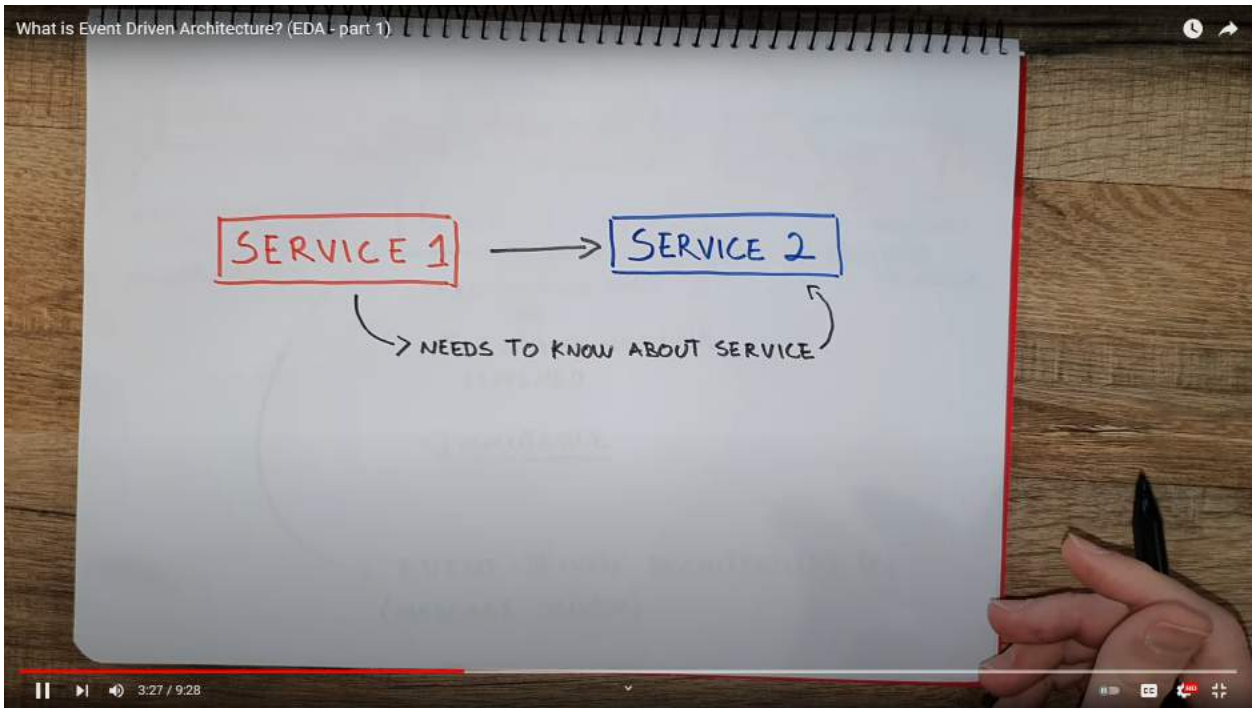
Components

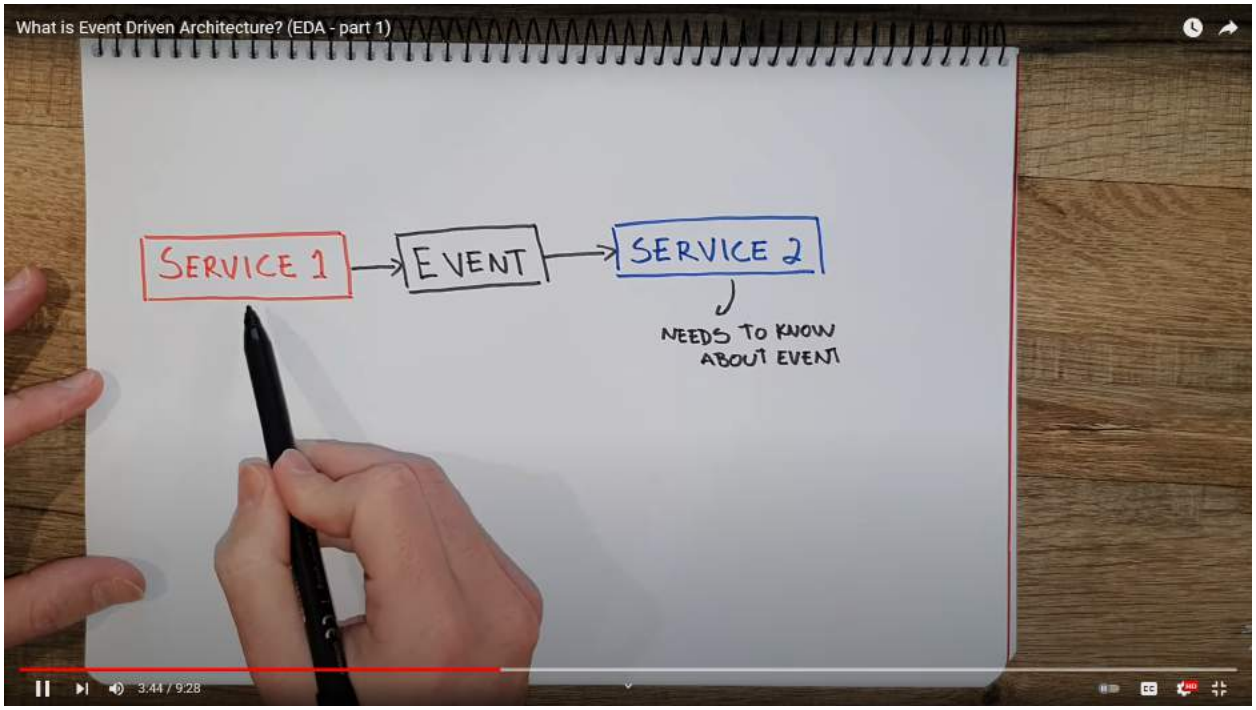
Publish-Subscribe

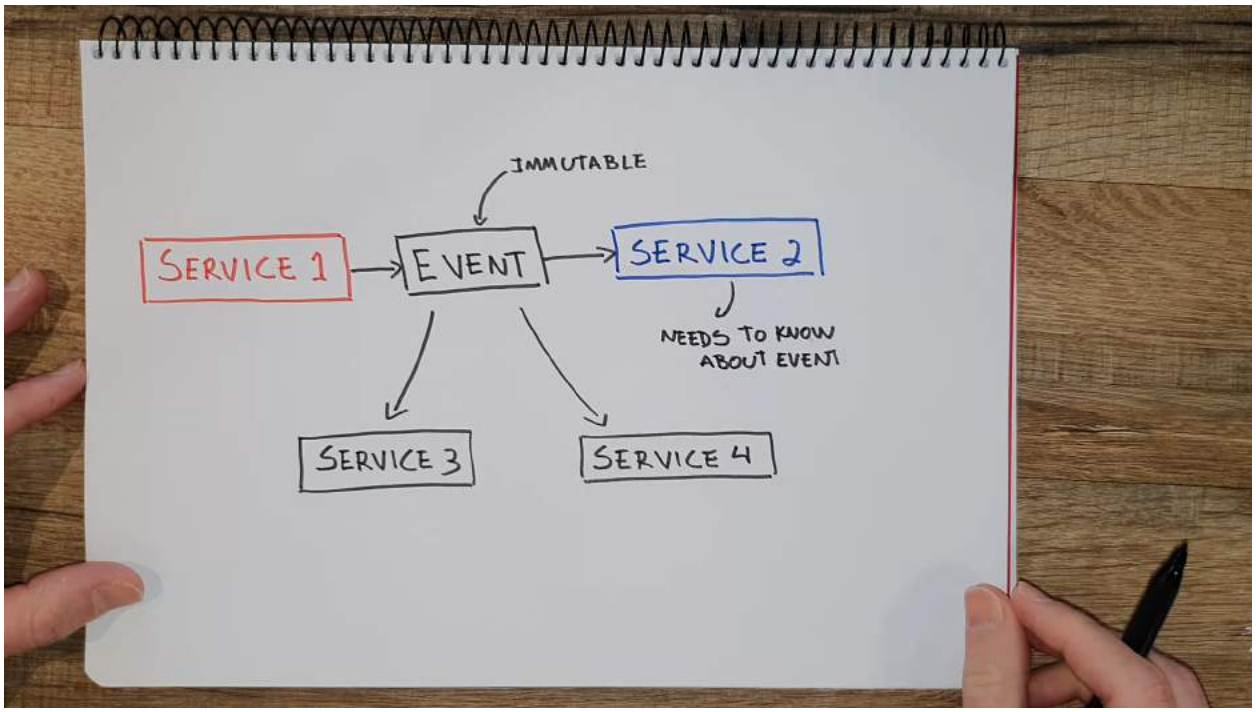


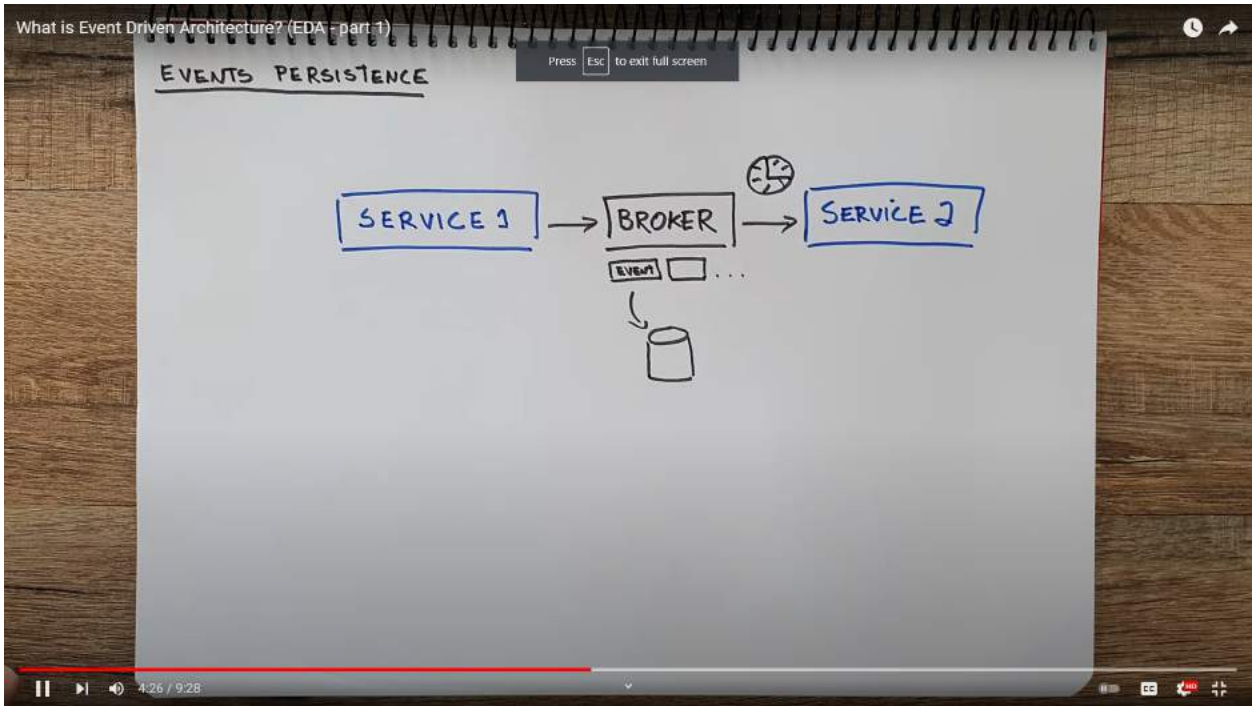
Advantages:

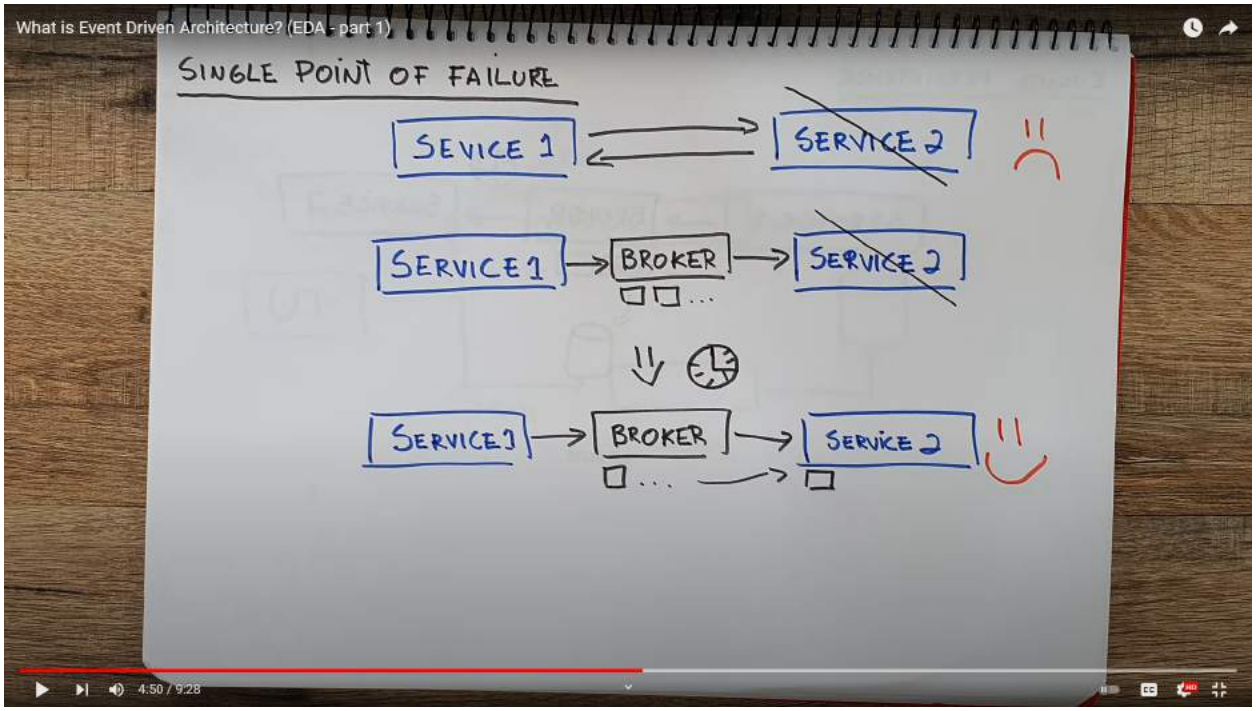
- Decoupling
- Dependency inversion
- Scalbility




What is Event Driven Architecture? (EDA - part 1)
SERVICE 1
EVENT
SERVICE 2
NEEDS TO KNOW ABOUT EVENT
3:44 / 9:28

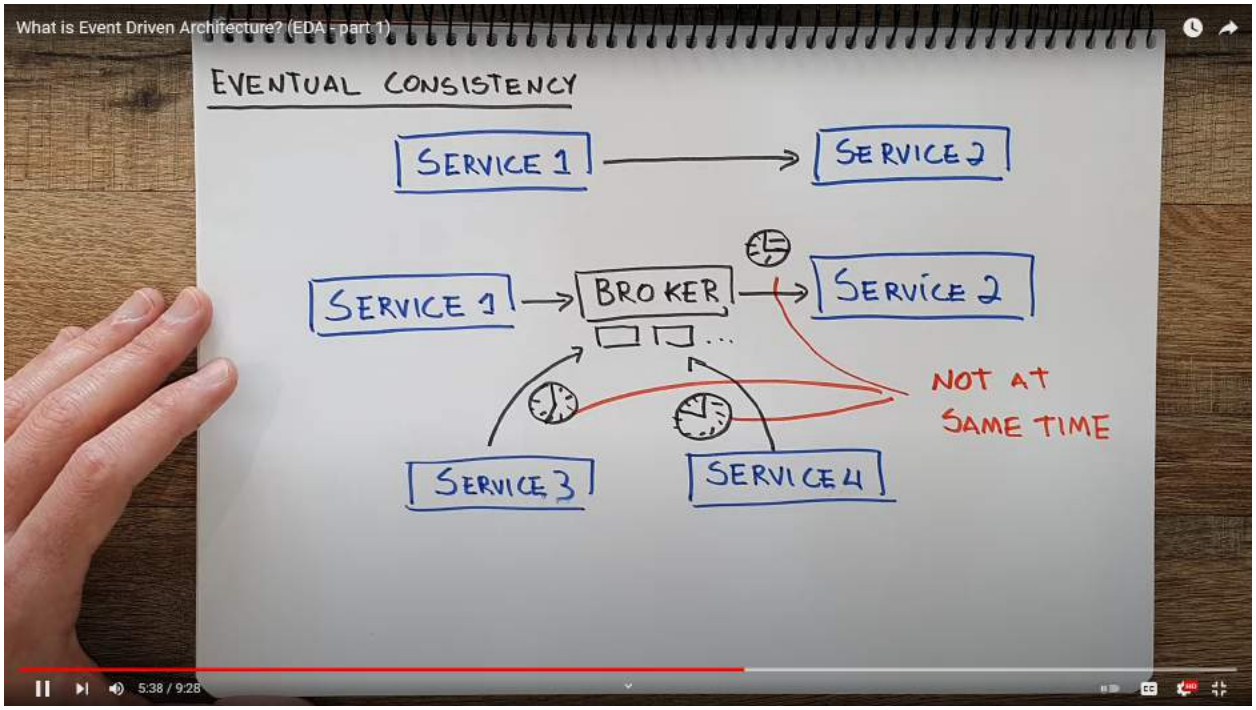

IMMUTABLE
SERVICE 1
EVENT
SERVICE 2
NEEDS TO KNOW ABOUT EVENT
SERVICE 3
SERVICE 4

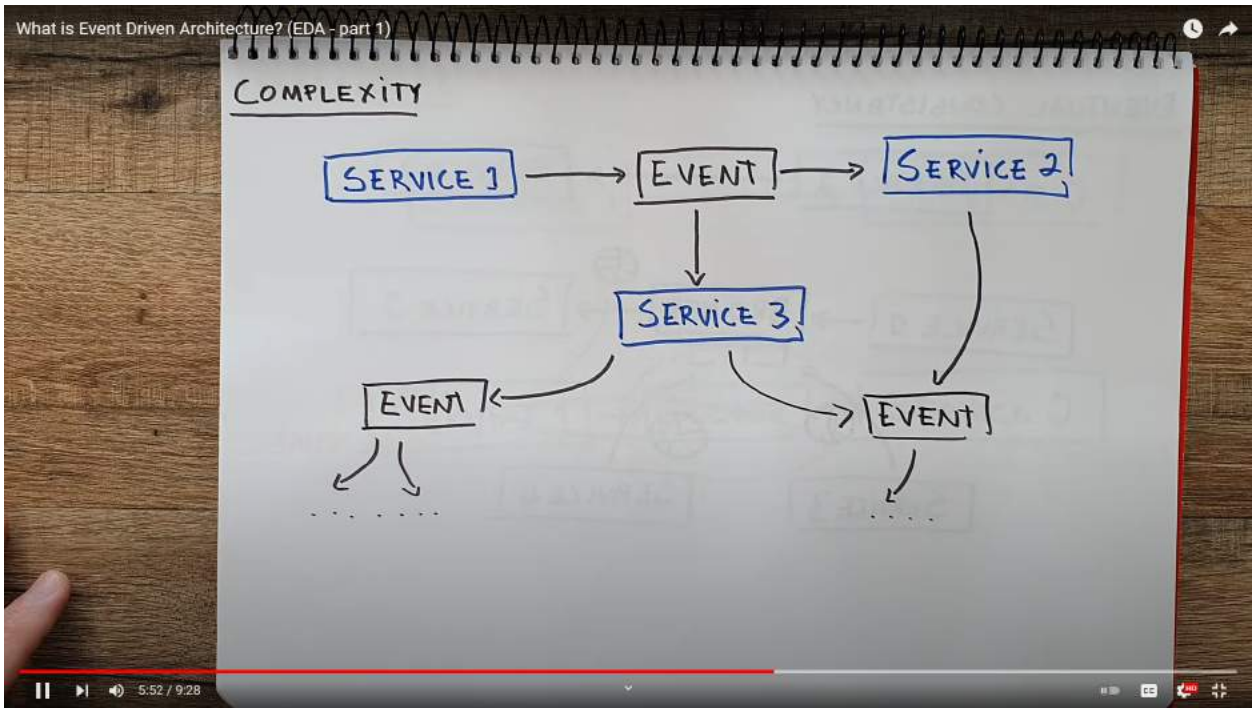


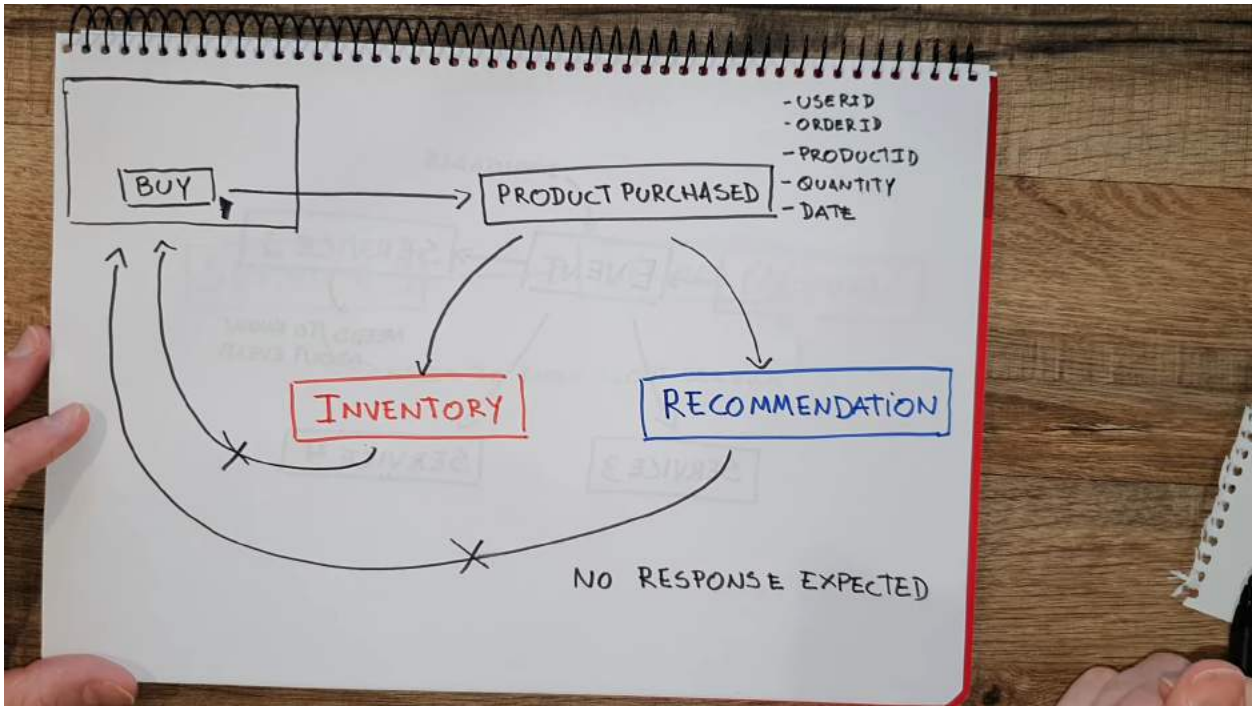


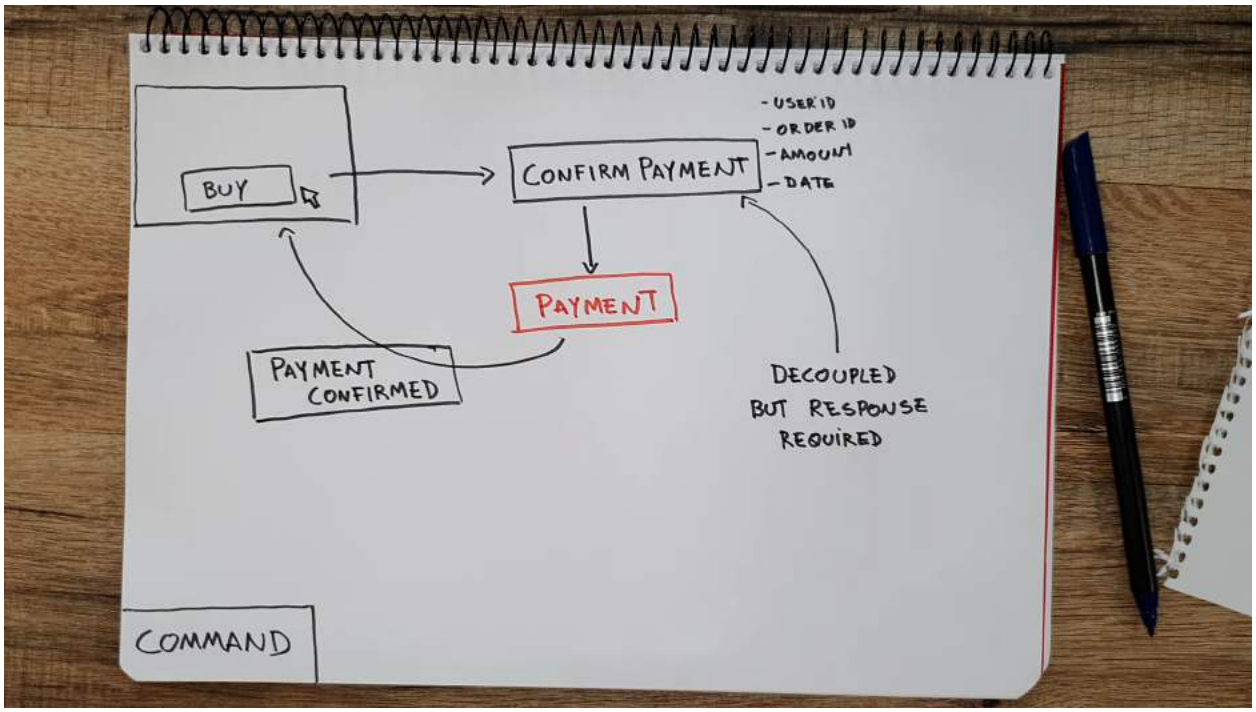
Cons: Performance

Now the broker is going to sit in between services


What is Event Driven Architecture? (EDA - part 1)
EVENTUAL CONSISTENCY
SERVICE 1
SERVICE 2
SERVICE 1
BROKER
SERVICE 2
NOT AT SAME TIME
SERVICE 3
SERVICE 4
5:38 / 9:28

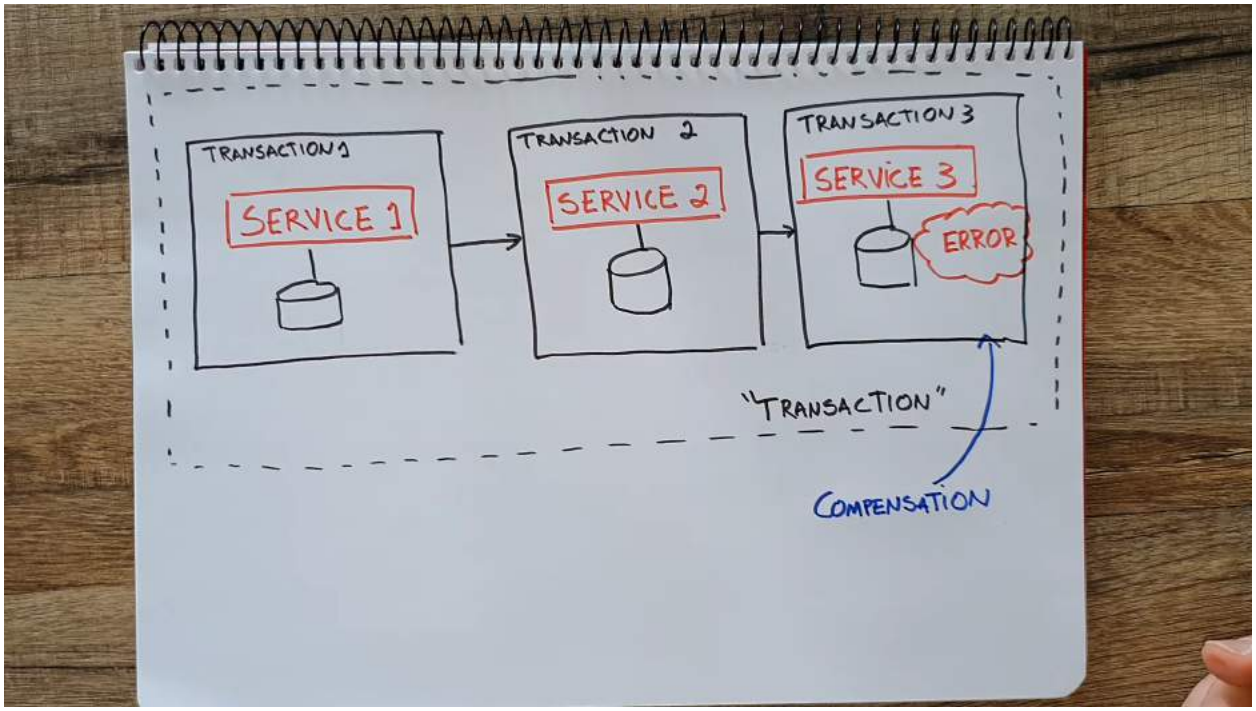

What is Event Driven Architecture? (EDA - part 1)
COMPLEXITY
SERVICE 1
EVENT
SERVICE 2
SERVICE 3
EVENT
EVENT
5:52 / 9:28

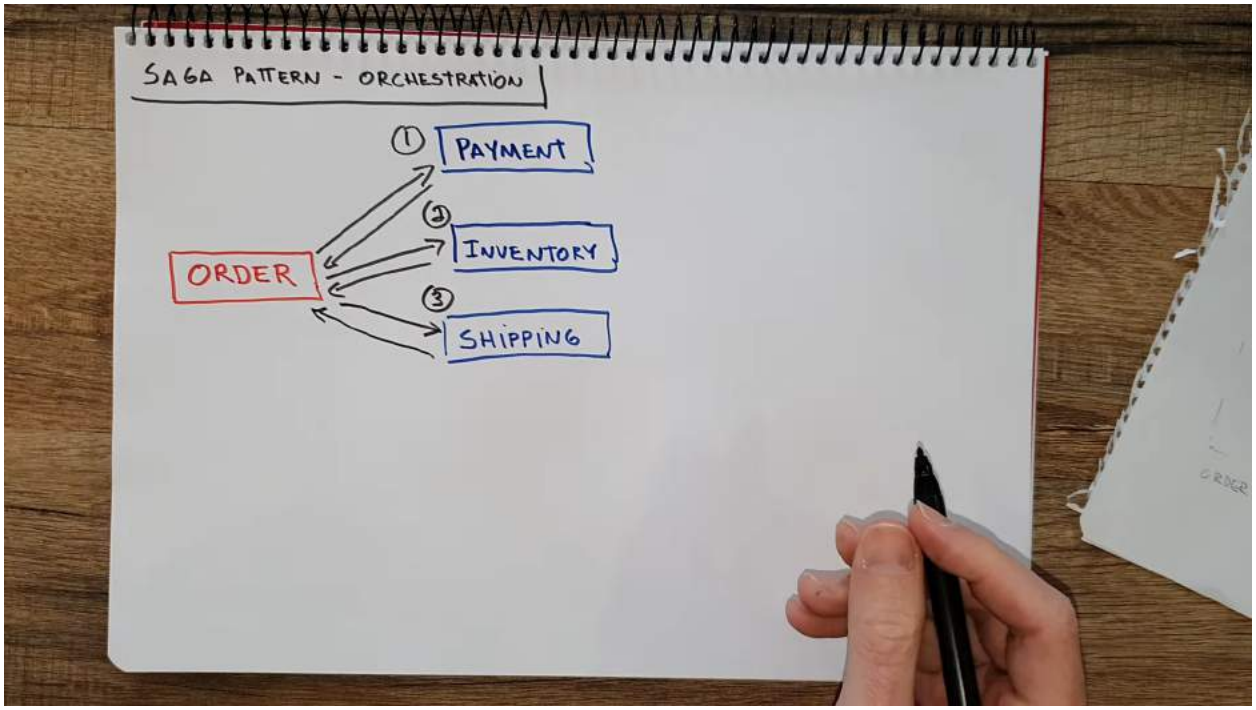

BUY
PRODUCT PURCHASED
- USERID
- ORDERID
- PRODUCTID
- QUANTITY
- DATE
INVENTORY
RECOMMENDATION
NO RESPONSE EXPECTED

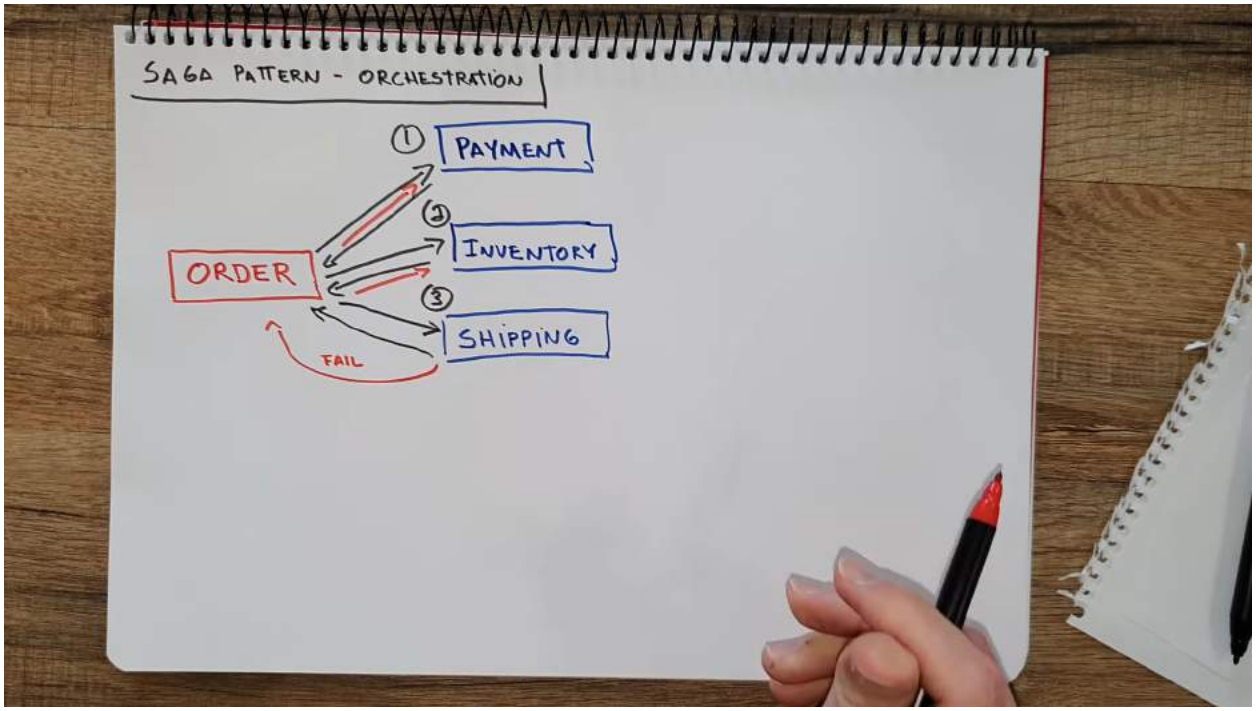

BUY
CONFIRM PAYMENT
- USER ID
- ORDER ID
- AMOUNT
- DATE
PAYMENT
PAYMENT CONFIRMED
DECOUPLED BUT RESPONSE REQUIRED
COMMAND

Saga pattern in Microservices: To make transactions over MC look like ACID-compliant
We can do it in two ways:

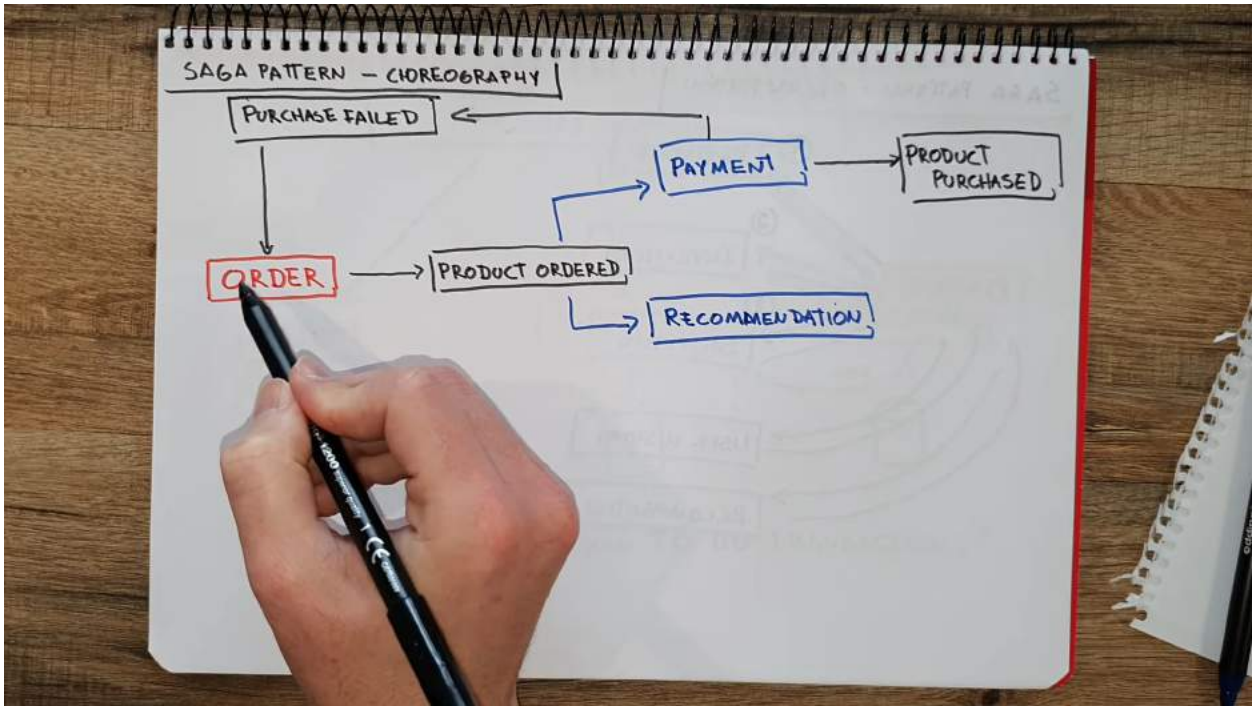
1. Orchestration – We need a central service to receive the response from all the other services
2. Coreography

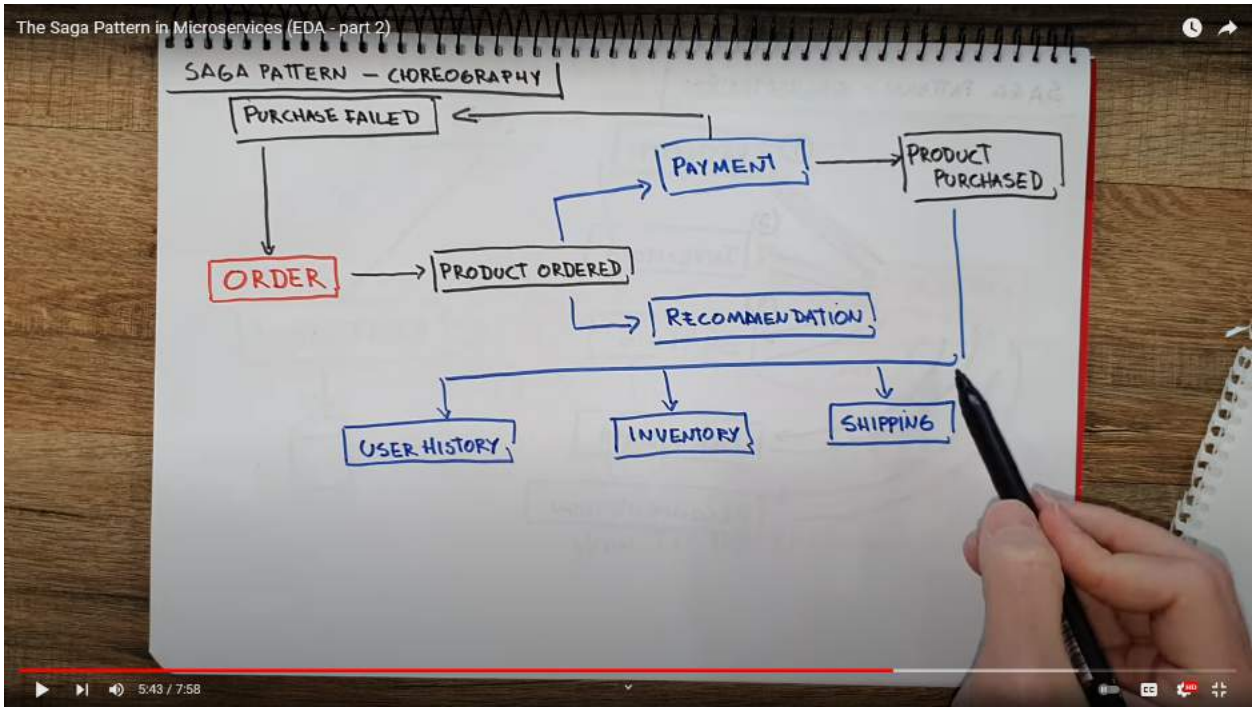


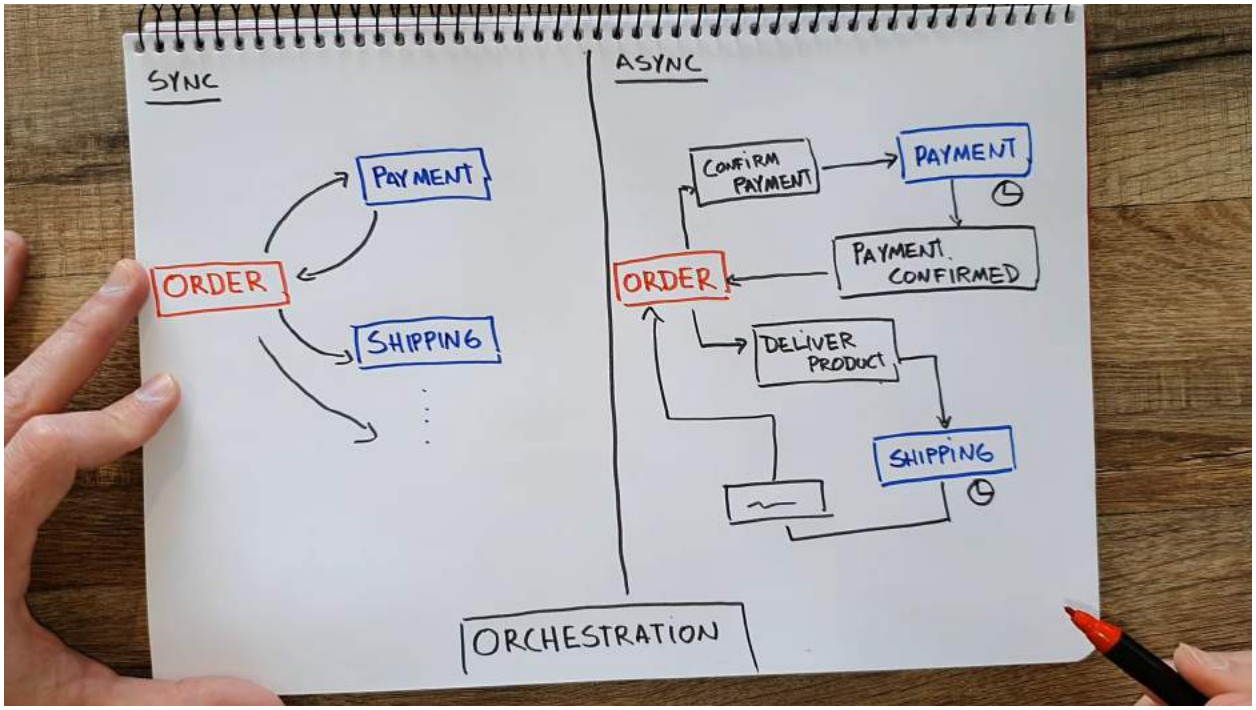




Choreography: Events


SAGA PATTERN – CHOREOGRAPHY
PURCHASE FAILED
PAYMENT
PRODUCT PURCHASED
ORDER
PRODUCT ORDERED
RECOMMENDATION

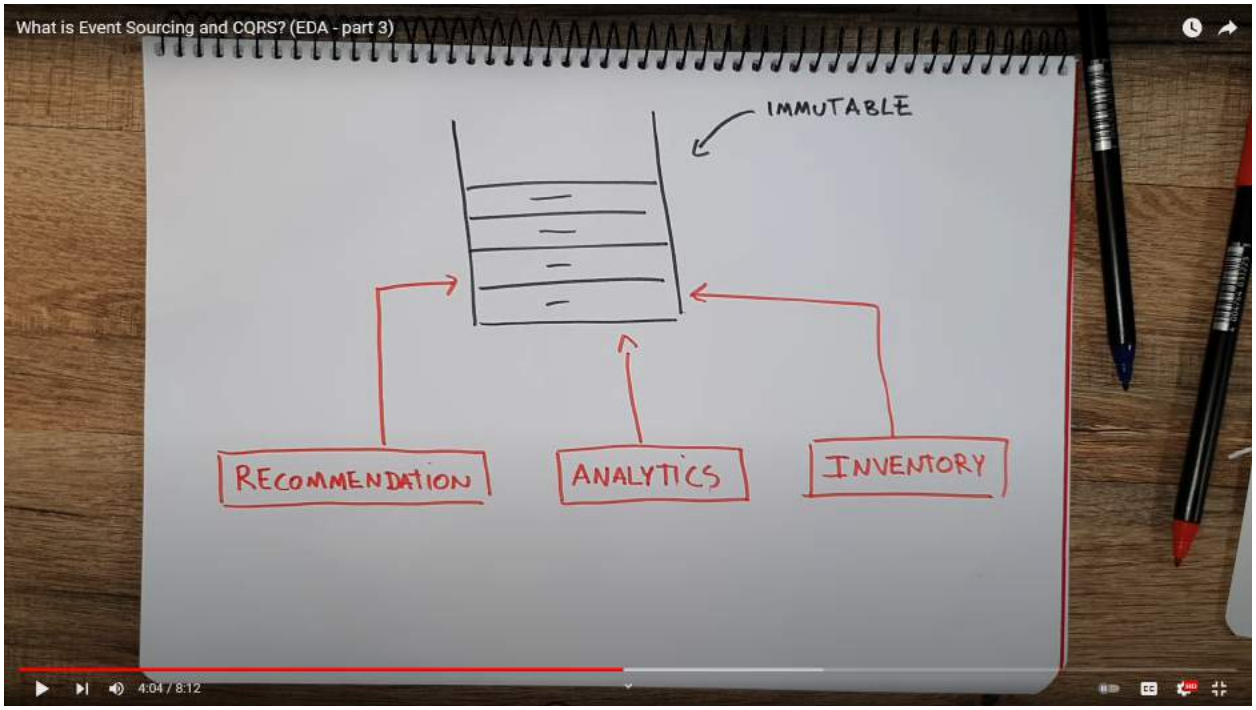

The Saga Pattern in Microservices (EDA - part 2)
SAGA PATTERN – CHOREOGRAPHY
PURCHASE FAILED
PAYMENT
PRODUCT PURCHASED
ORDER
PRODUCT ORDERED
RECOMMENDATION
USER HISTORY
INVENTORY
SHIPPING
5:43 / 7:58

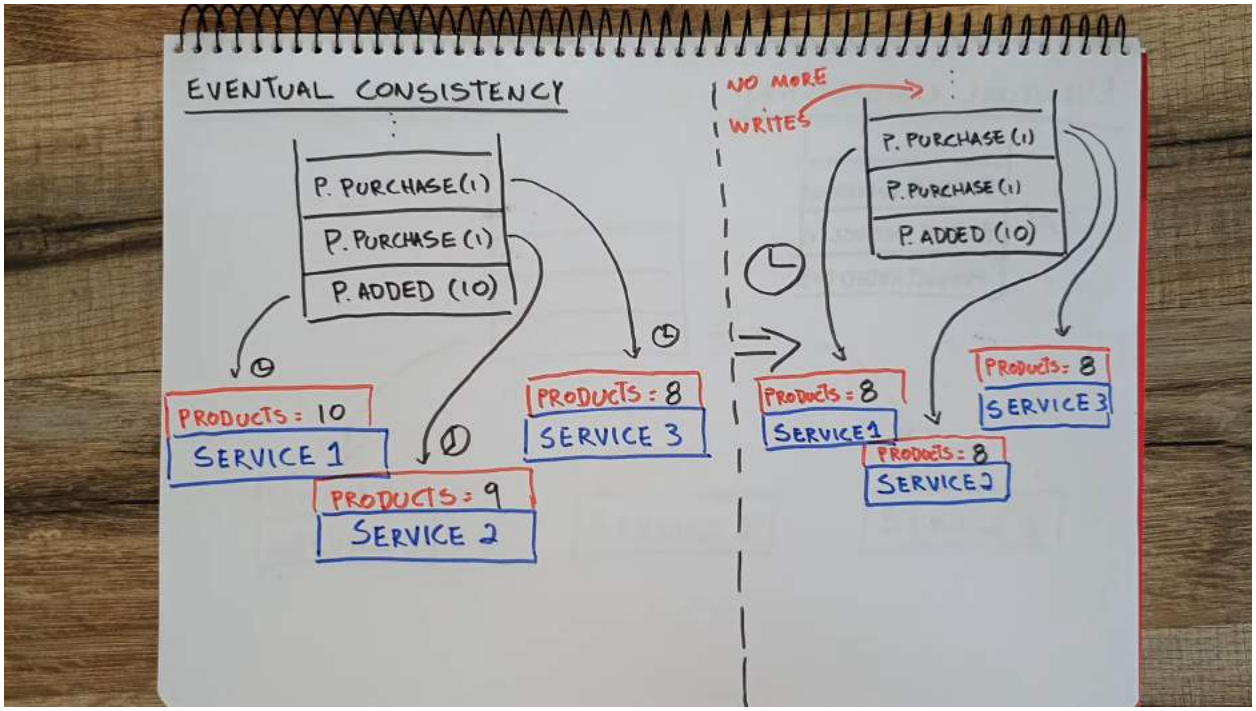


Event Sourcing **gives us a new way of persisting application state as an ordered sequence of events**. We can selectively query these events and reconstruct the state of the application at any point in time.
These events here **are facts that have happened and can not be altered** — in other words, they must be immutable.

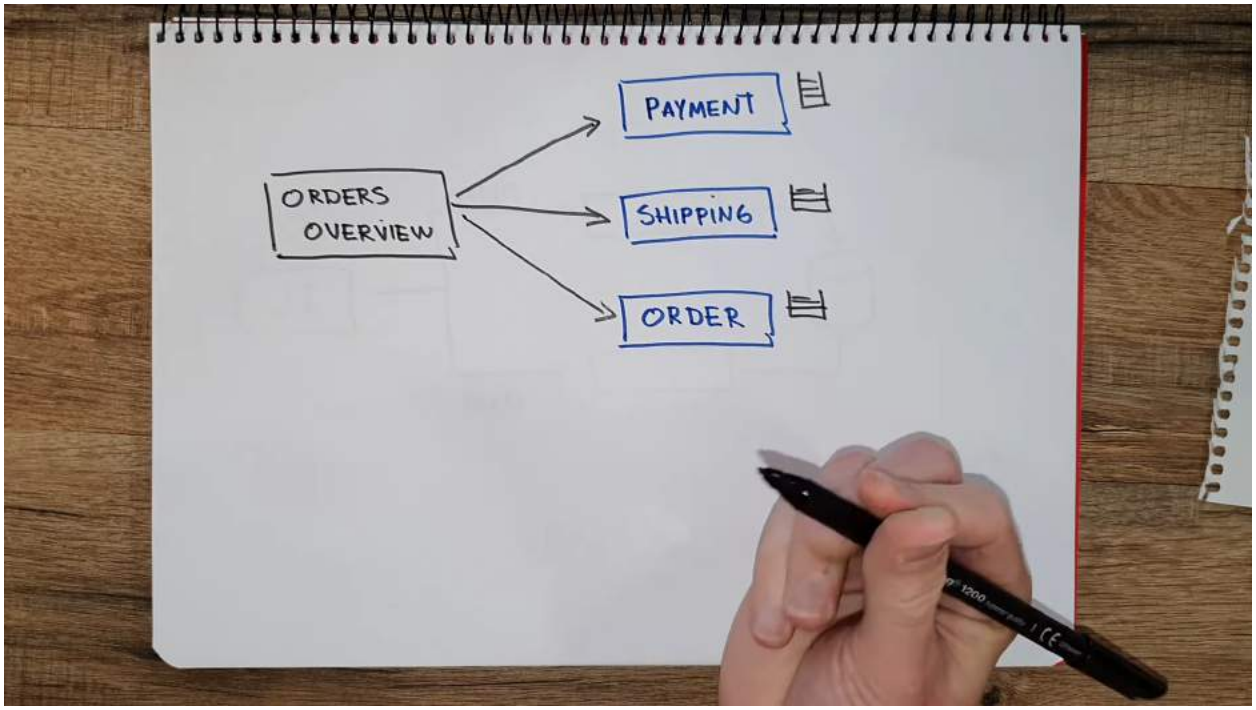
Replaying all the events from the past is not feasible, so we need to have a snapshot at a point in time

It helps to query Temporal data, used in audits





CQRS: Command Query Responsibility Segregation



Event Replay: Helps in debugging the issue