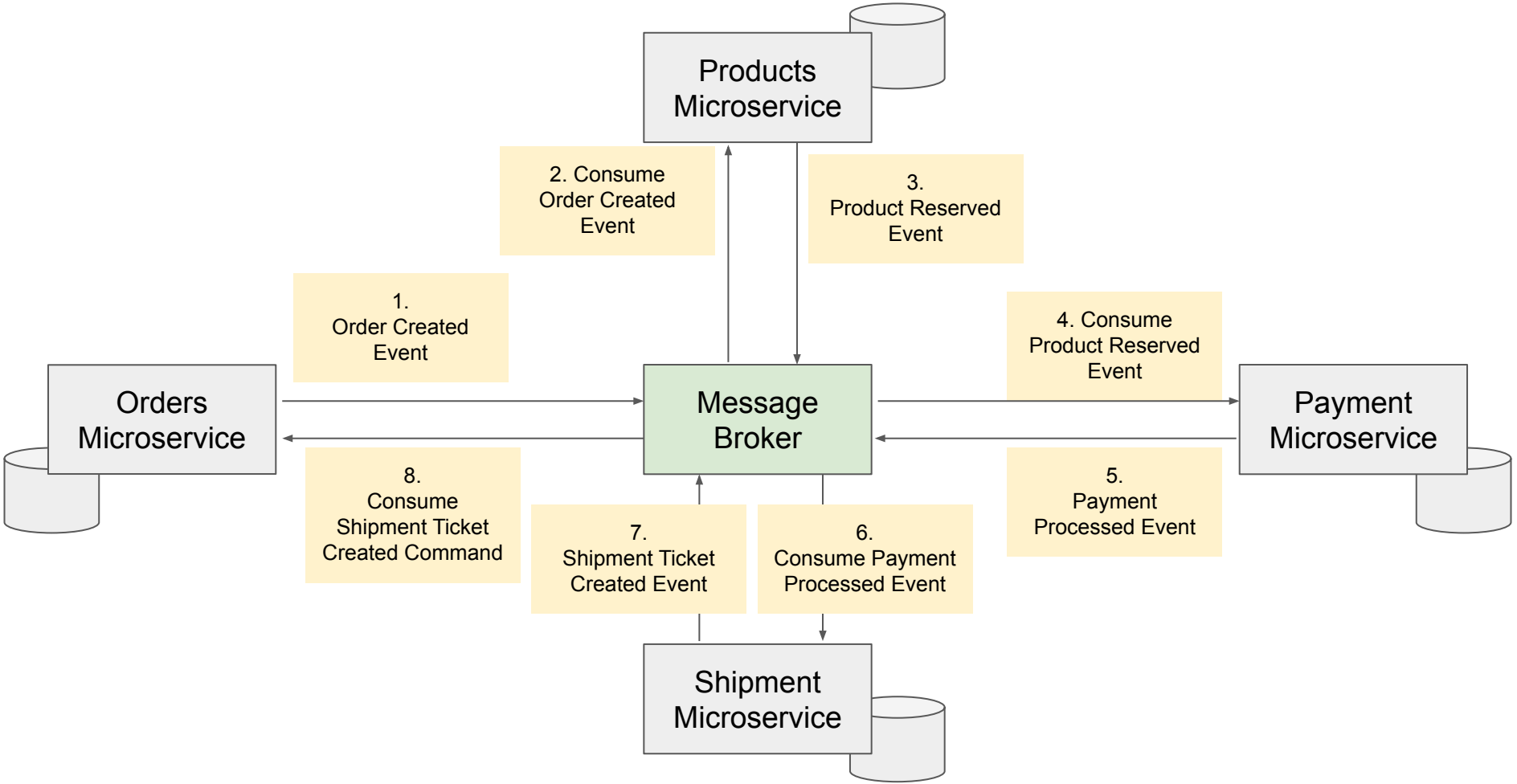
SAGA
Design Pattern

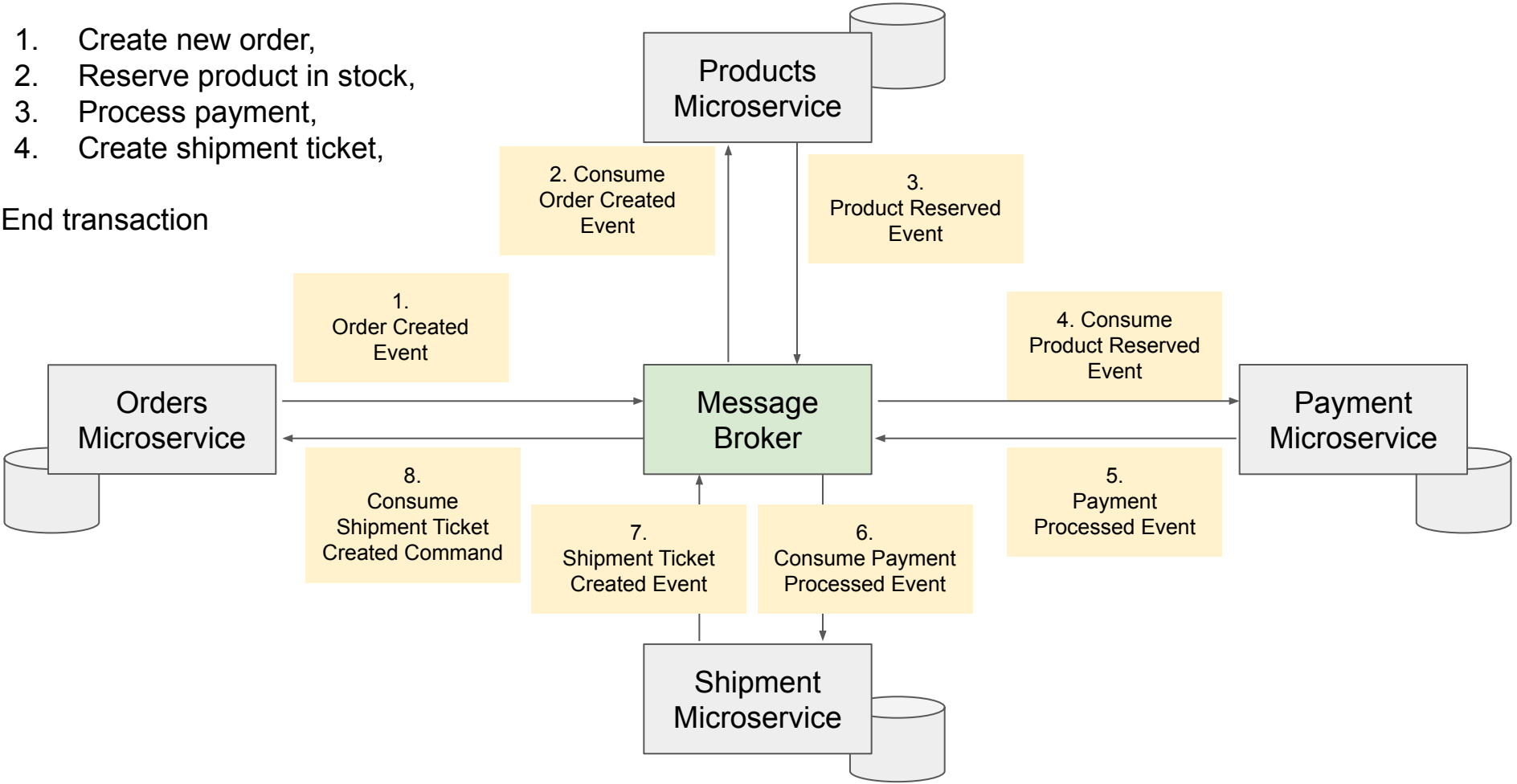
Choreography-Based Saga


Products Microservice
2. Consume Order Created Event
3. Product Reserved Event
1. Order Created Event
4. Consume Product Reserved Event
Orders Microservice
Message Broker
Payment Microservice
8. Consume Shipment Ticket Created Command
7. Shipment Ticket Created Event
6. Consume Payment Processed Event
5. Payment Processed Event
Shipment Microservice

Begin Transaction

1. Create new order,
2. Reserve product in stock,
3. Process payment,
4. Create shipment ticket,

End transaction



Initial Flow:

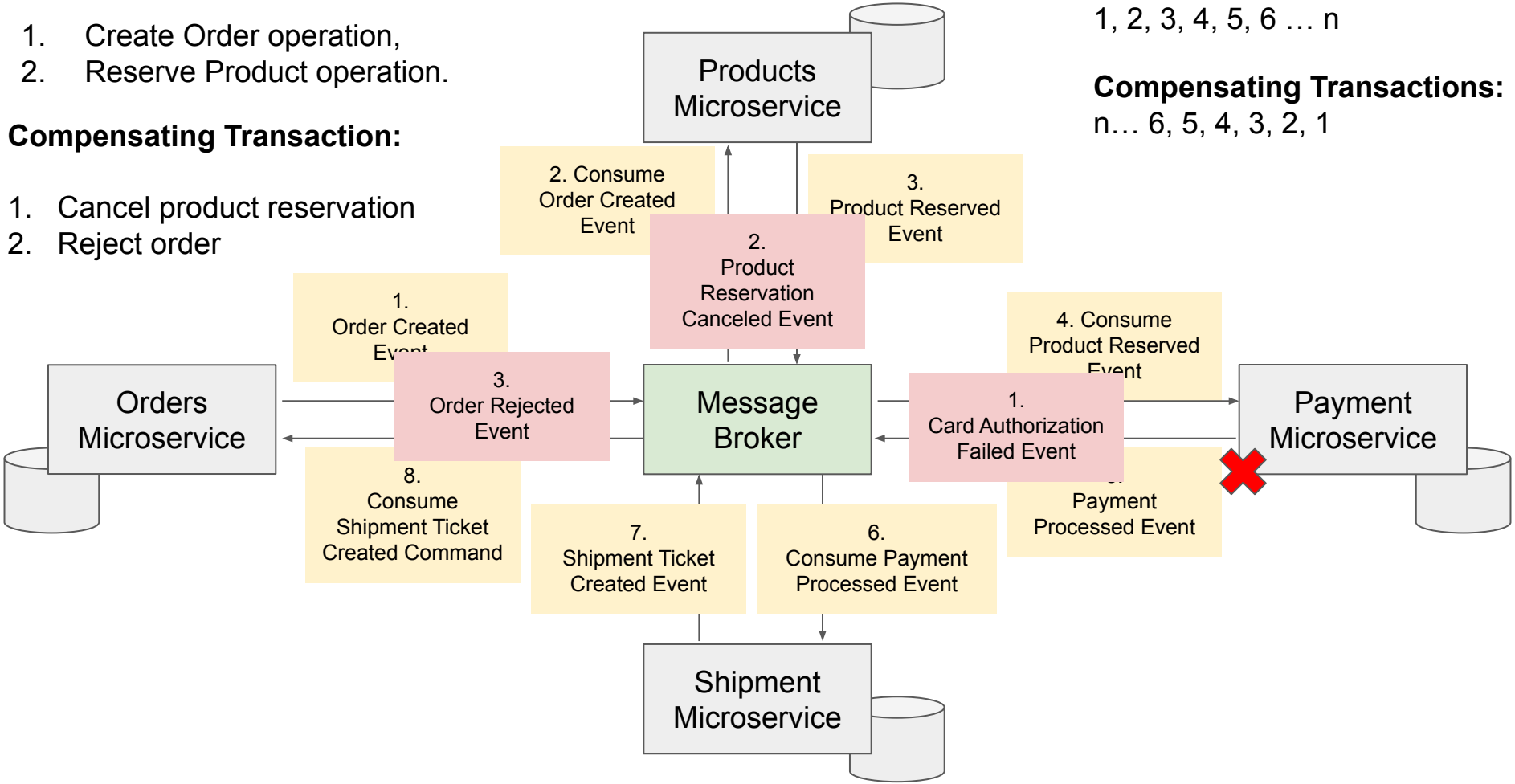
1. Create Order operation,
2. Reserve Product operation.

Compensating Transaction:

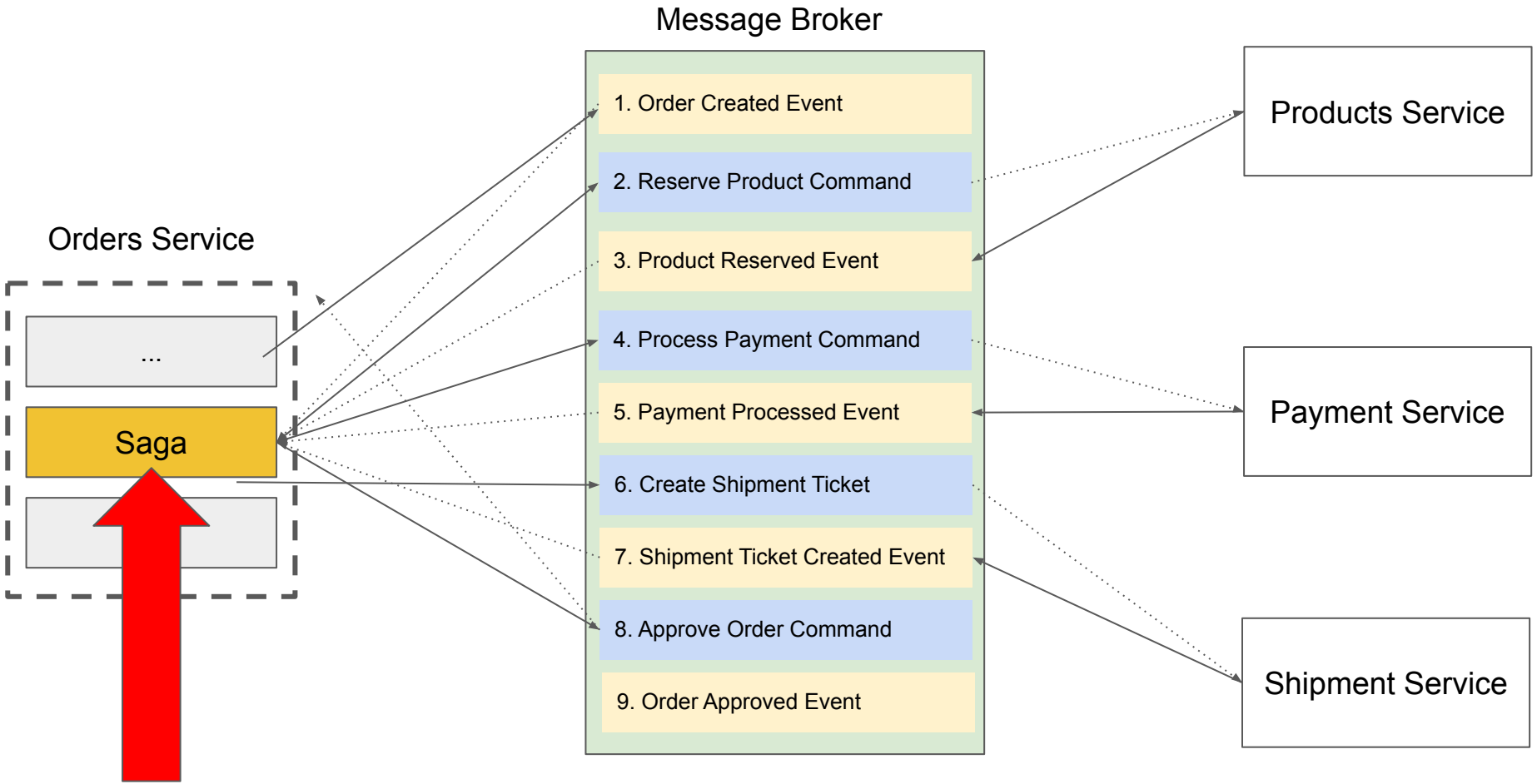
1. Cancel product reservation
2. Reject order

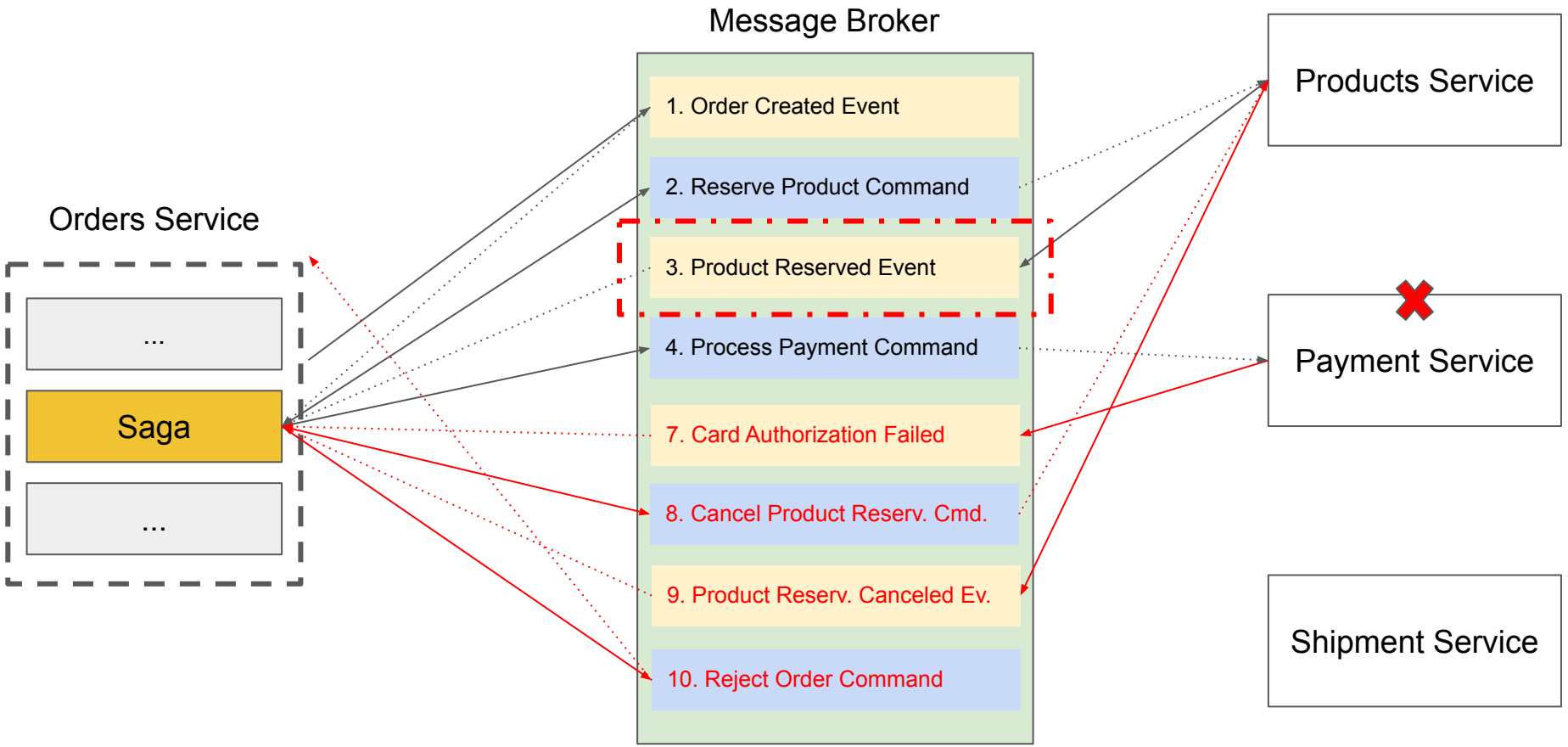
Initial Flow:
1, 2, 3, 4, 5, 6 … n

Compensating Transactions:
n… 6, 5, 4, 3, 2, 1

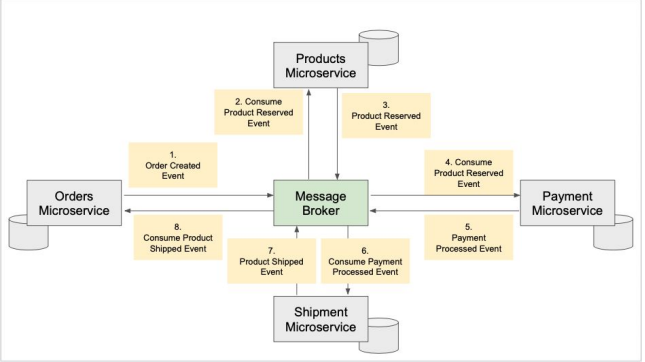


Orchestration-Based Saga


Message Broker
1. Order Created Event
2. Reserve Product Command
3. Product Reserved Event
4. Process Payment Command
5. Payment Processed Event
6. Create Shipment Ticket
7. Shipment Ticket Created Event
8. Approve Order Command
9. Order Approved Event
Products Service
Payment Service
Shipment Service
Orders Service
...
Saga


Message Broker
Products Service
1. Order Created Event
2. Reserve Product Command
Orders Service
3. Product Reserved Event
...
4. Process Payment Command
Payment Service
Saga
7. Card Authorization Failed
8. Cancel Product Reserv. Cmd.
...
9. Product Reserv. Canceled Ev.
Shipment Service
10. Reject Order Command

Which Saga pattern to use?



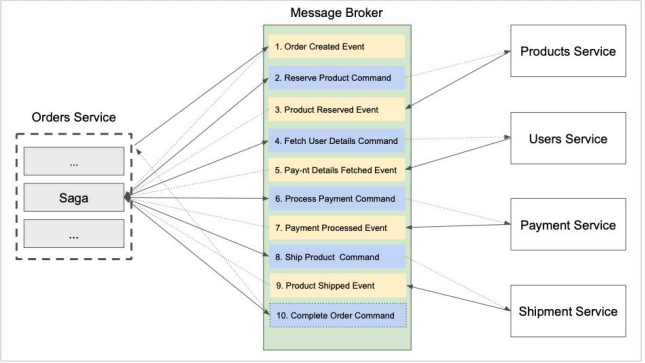
Choreography-based Saga

Benefits:

- Good for simple workflows that require few microservices and don't need a coordination logic,
- Doesn't require additional service implementation and maintenance to coordinate transactions,
- Doesn't introduce a single point of failure, since the responsibilities are distributed across the saga participants.

Drawbacks:

- Workflow can become confusing when adding new steps and Microservices. The more steps in a transaction, the more difficult it is to track which saga participants listen to which commands.
- There's a risk of cyclic dependency between saga participants because they have to consume each other's commands.
- Integration testing is difficult because all services must be running to simulate a transaction.



Orchestration-based Saga

Benefits:

- Good for complex workflows involving many Microservices,
- Easier to control the flow of activities,
- Doesn't introduce cyclic dependencies, because the orchestrator unilaterally depends on the saga participants,
- Saga participants don't need to know about commands for other participants. Clear separation of concerns simplifies business logic.

Drawbacks:

- Additional design complexity requires an implementation of a coordination logic.
- There's an additional point of failure, because the orchestrator manages the complete workflow.