Transactions

Systems Analysis & Design

Learning Objectives

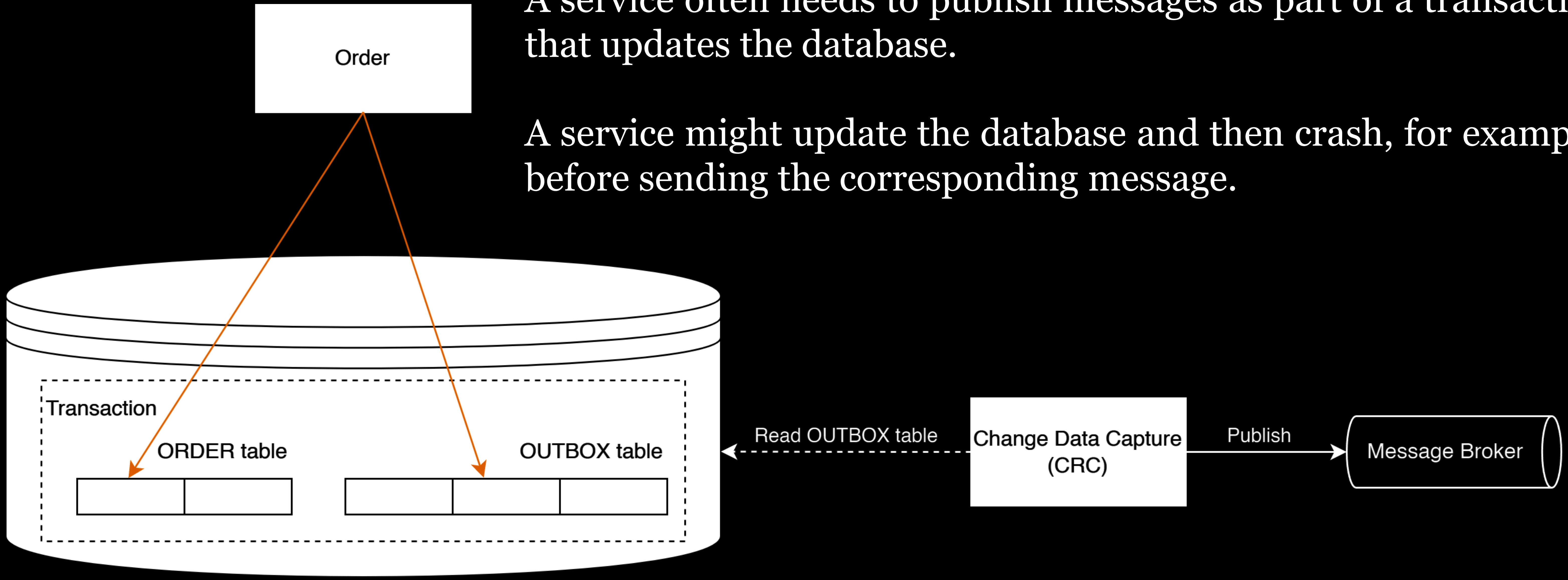
By the end of this session, you will have acquired the following information:

- Outbox Pattern
- Saga Pattern

Outbox Pattern

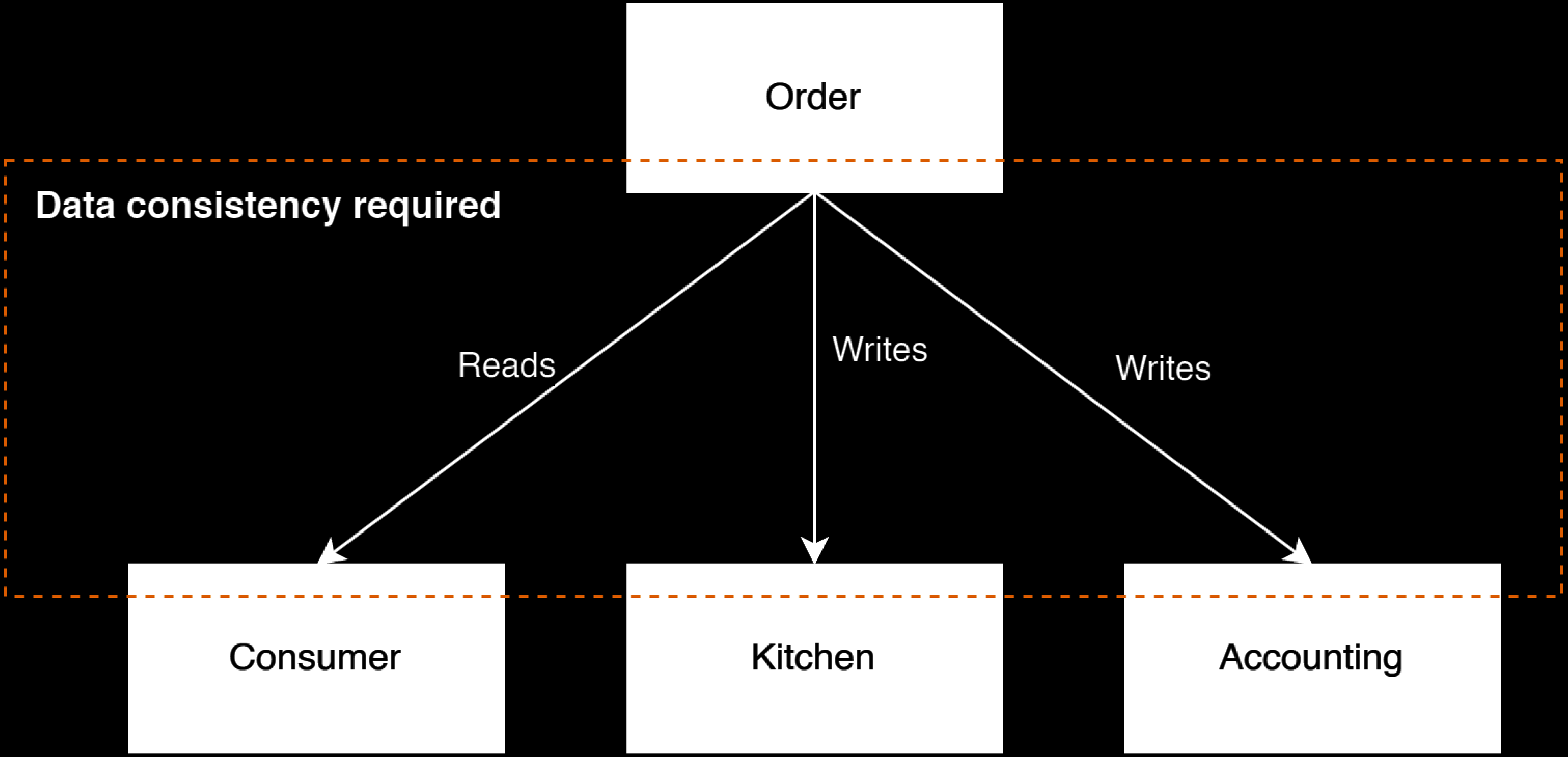
A service often needs to publish messages as part of a transaction that updates the database.

A service might update the database and then crash, for example, before sending the corresponding message.

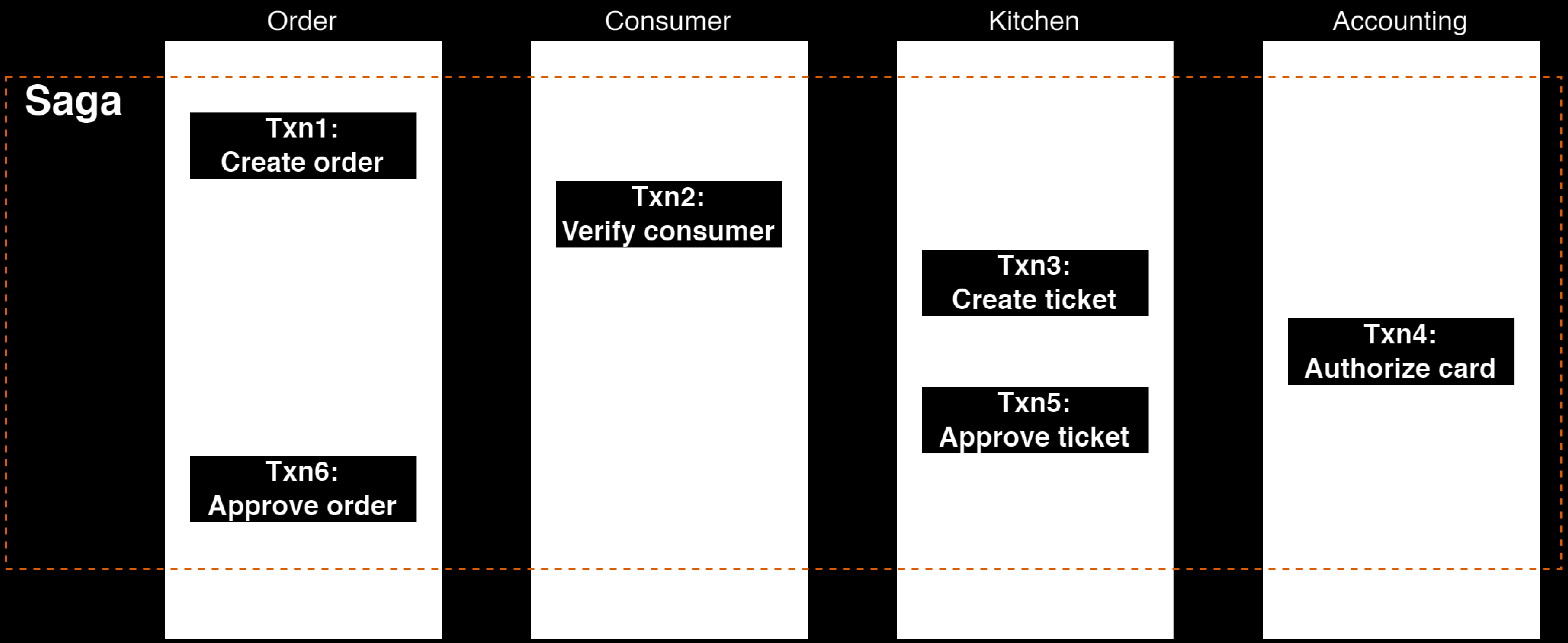


Saga Pattern

The `createOrder()` operation reads from the Consumer Service and updates data in the Order Service, Kitchen Service, and Accounting Service.



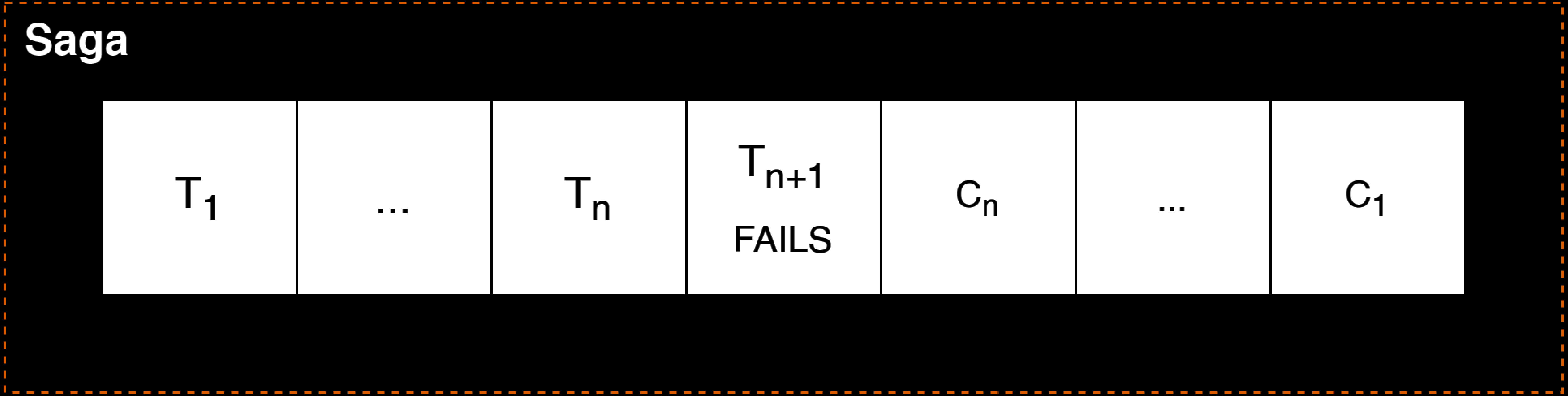
The `createOrder()` operation is implemented by a saga that consists of local transactions in several services.



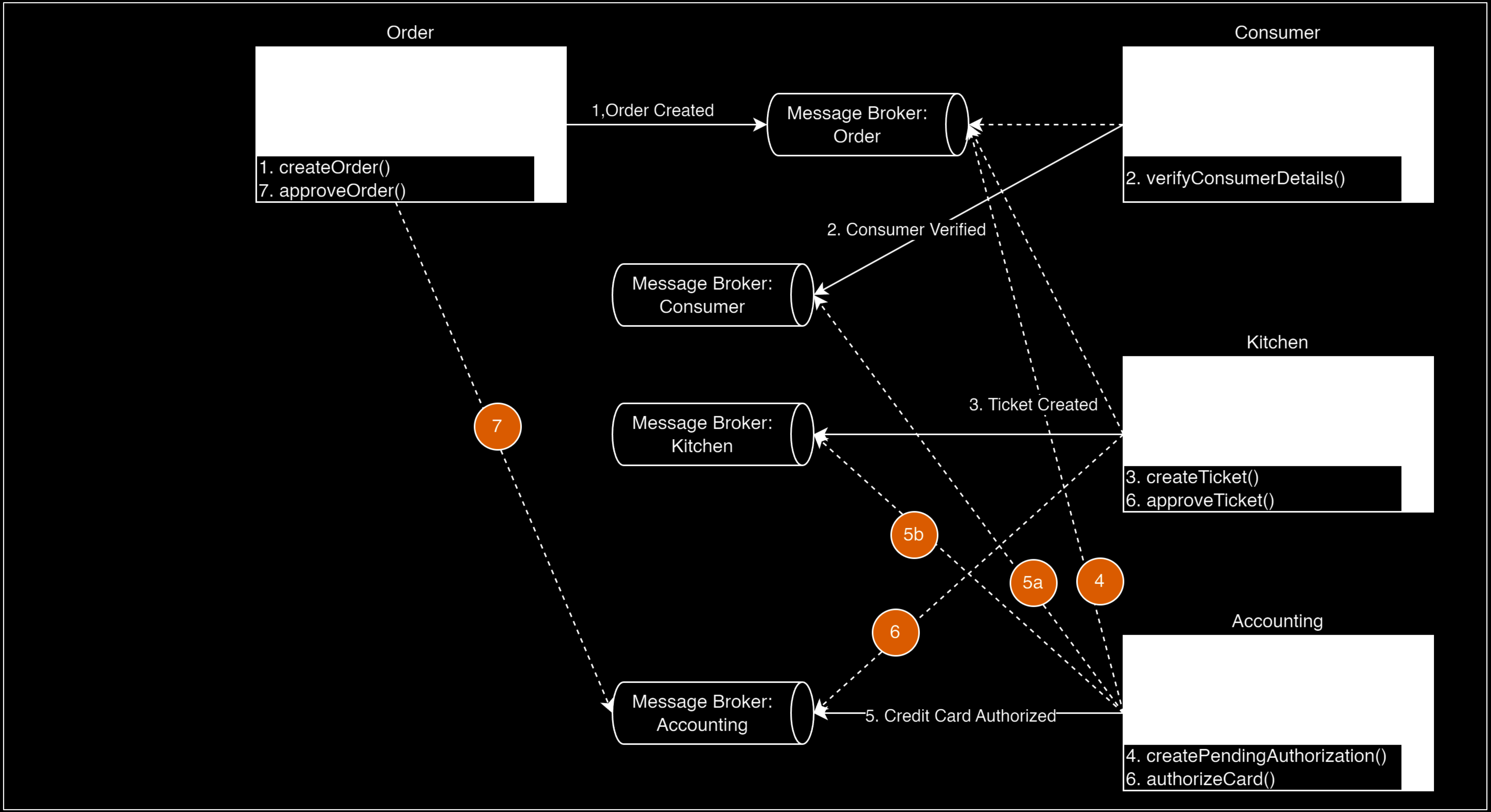
When a step of a saga fails due to a business rule violation, the saga must explicitly undo the updates made by previous steps by executing compensating transactions.

The changes made by $T_1 \ldots T_n$ have been committed

The compensating transactions undo the changes made by $T_1 \ldots T_n$



Step	Service	Transaction	Compensating Transaction
1	Order	`createOrder()`	`rejectOrder()`
2	Consumer	`verifyConsumerDetails()`	–
3	Kitchen	`createTicket`	`rejectTicket()`
4	Accounting	`authorizedCreditCard`	–
5	Kitchen	`approveTicket()`	–
6	Order	`approveOrder()`	–


Order
1. createOrder()
7. approveOrder()
1,Order Created
Message Broker: Order
Consumer
2. verifyConsumerDetails()
2. Consumer Verified
Message Broker: Consumer
Kitchen
3. Ticket Created
Message Broker: Kitchen
3. createTicket()
6. approveTicket()
7
5b
5a
4
6
Accounting
Message Broker: Accounting
5. Credit Card Authorized
4. createPendingAuthorization()
6. authorizeCard()

Successful Order Fulfillment Saga

1. The Order Service creates an `Order` in the `APPROVAL_PENDING` state and publishes an `OrderCreated` event.
2. Consumer Service consumes the `OrderCreated` event, verifies that the consumer can place the order, and publishes a `ConsumerVerified` event.
3. Kitchen Service consumes the `OrderCreated` event, validates the `Order`, creates a `Ticket` in a `CREATE_PENDING` state, and publishes the `TicketCreated` event.
4. Accounting Service consumes the `OrderCreated` event and creates a `CreditCardAuthorization` in a `PENDING` state.
5. Accounting Service consumes the `TicketCreated` and `ConsumerVerified` events, charges the consumer's credit card, and publishes the `CreditCardAuthorized` event.
6. Kitchen Service consumes the `CreditCardAuthorized` event and changes the state of the `Ticket` to `AWAITING_ACCEPTANCE`.
7. Order Service receives the `CreditCardAuthorized` events, changes the state of the `Order` to `APPROVED`, and publishes an `OrderApproved` event.

Further Resources

- Managing Transactions with Sagas