



# BASE vs ACID

🕒 Created	@May 24, 2023 5:07 PM
👤 Owner	 Din Lester  Din Lester
🏷️ Tags	databases
🌟 Status	Done

## Materials:

- <https://www.youtube.com/watch?v=pomxJOcVcQs>
- <https://www.postgresql.org/docs/current/transaction-iso.html>
- <https://en.wikipedia.org/wiki/ACID>
- [https://en.wikipedia.org/wiki/Two-phase\\_locking](https://en.wikipedia.org/wiki/Two-phase_locking)
- [https://en.wikipedia.org/wiki/Isolation\\_\(database\\_systems\)](https://en.wikipedia.org/wiki/Isolation_(database_systems))
- <https://blog.bitsrc.io/acid-and-base-database-model-fadb156c660f>
- <https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674>
- <https://www.baeldung.com/cs/saga-pattern-microservices>
- <https://www.wallarm.com/what/orchestration-vs-choreography>

CAP theorem says that we cannot pick all 3 states.

So, ACID stands for consistency, BASE - for availability.

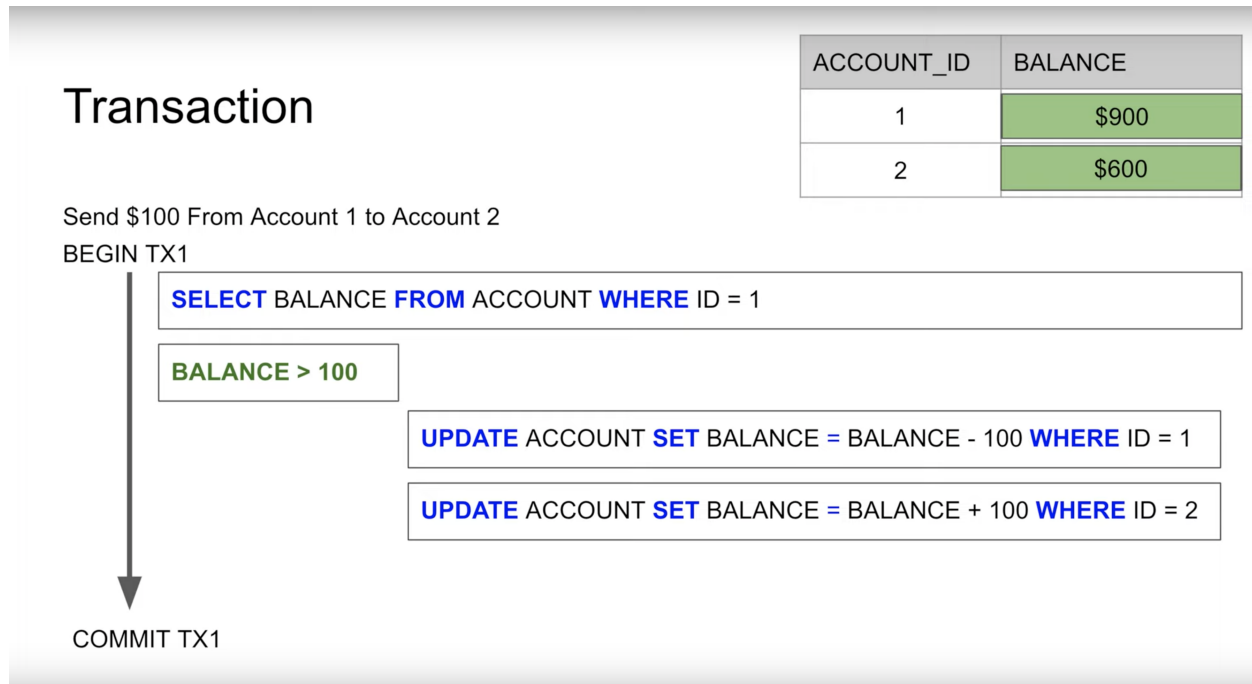
## ACID

Agenda:

- What is transaction ?
- Atomicity
- Consistency

- Isolation
- Durability

## Transaction



## Atomicity

$$\exists o \in N \Rightarrow \forall o \in T \in N$$

$$\nexists o \in N \Rightarrow \forall o \in T \in P$$

N - set for negative operations

P - set for positive operations

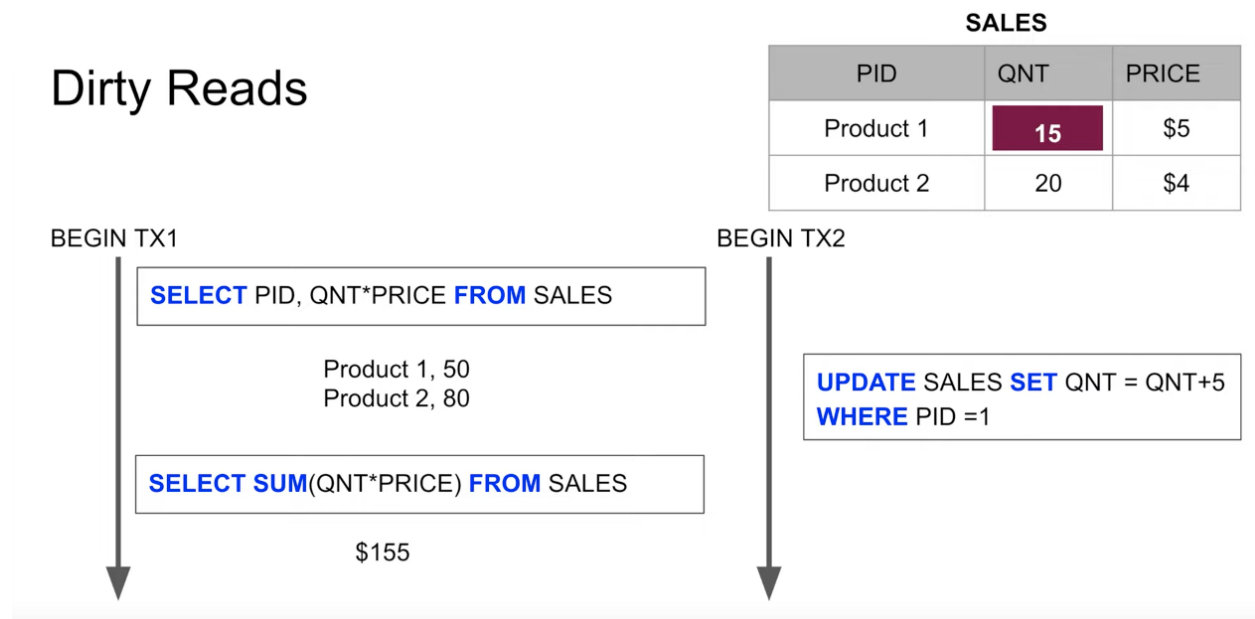
T - set of all operations

## Isolation

Read phenomenas:

## Dirty reads

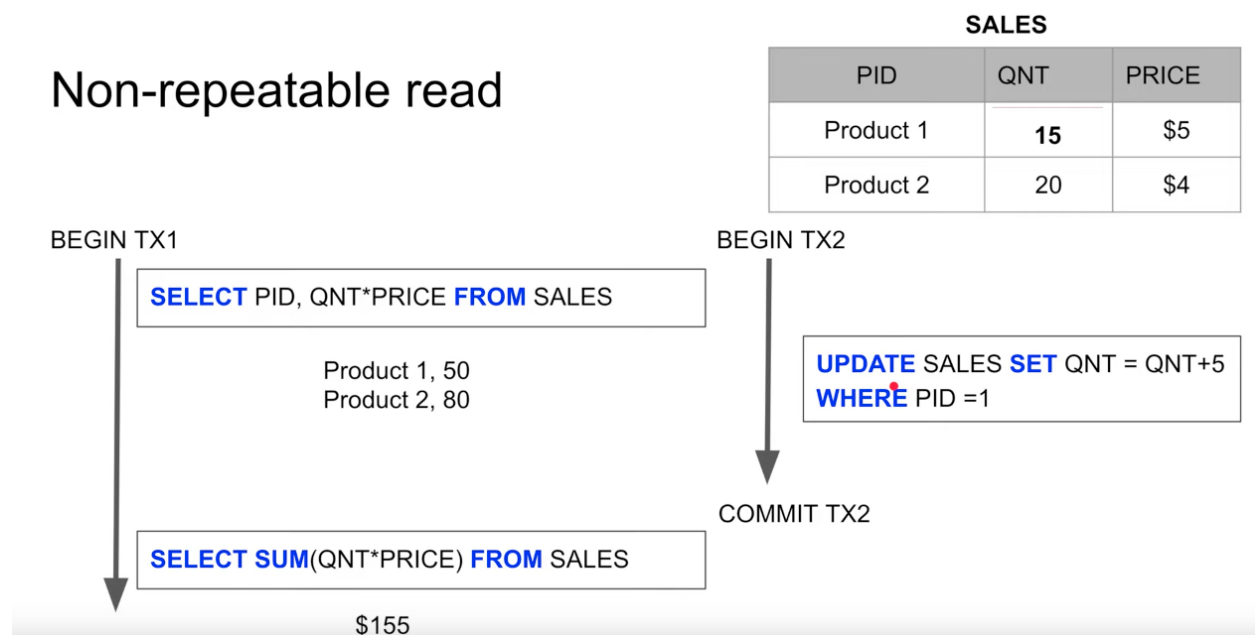
### Dirty Reads



A transaction reads data written by a concurrent uncommitted transaction.

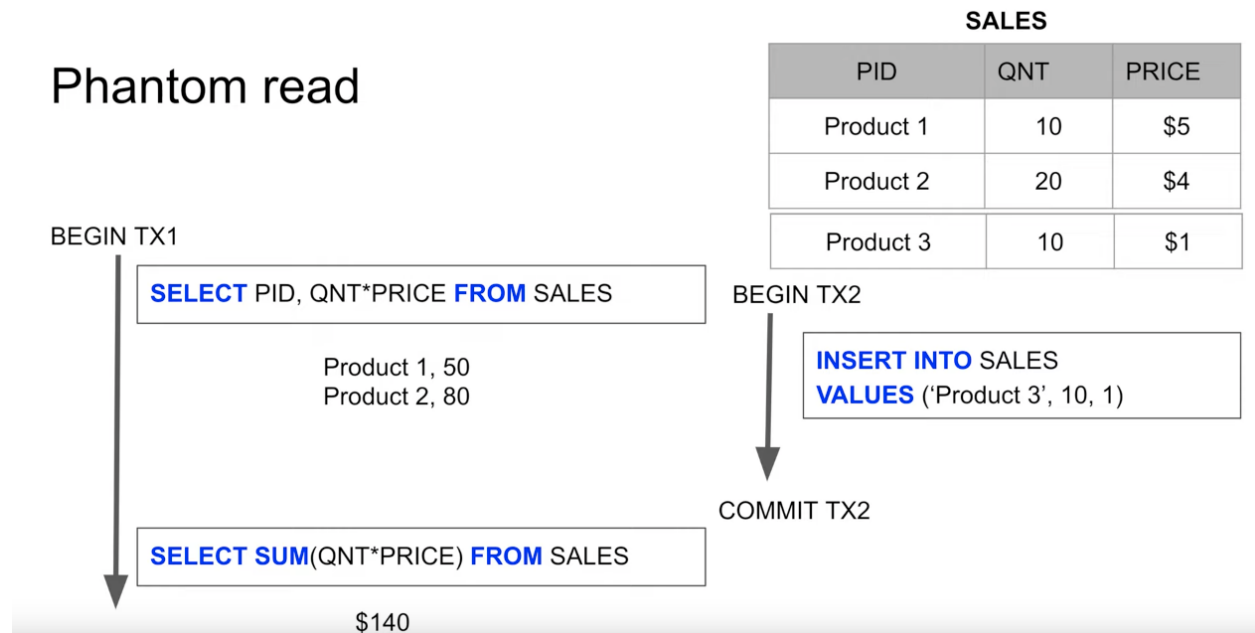
## Nonrepeatable Read

### Non-repeatable read



A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).

## Phantom Read



A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction.

## Isolation - Isolation Levels for inflight transaction

- **Read uncommitted**
  - No Isolation, any change from the outside is visible to the transaction
- **Read committed**
  - Each query in a transaction only sees committed stuff
- **Repeatable Read**
  - Each query in a transaction only sees committed updates at the beginning of transaction
- **Serializable**
  - Transactions are serialized.

Repeatable read, e.g. in Cassandra, is made by versioning. Also, commonly it's just a Shared Lock

What isolation level solves ?

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Allowed, but not in PG	Possible	Possible
Read committed	<b>Not possible</b>	Possible	Possible
Repeatable read	<b>Not possible</b>	<b>Not possible</b>	Allowed, but not in PG
Serializable	<b>Not possible</b>	<b>Not possible</b>	<b>Not possible</b>

Some notes about technical realization of isolation:

**Locks:** If user A is running a transaction that has to read a row of data that user B wants to modify, user B must wait until user A's transaction completes. Two-phase locking is often applied to guarantee full isolation.

**Versioning:** When user A's transaction requests data that user B is modifying, the database provides A with the version of that data that existed when user

B started his transaction. User A gets a consistent view of the database even if other users are changing data. One implementation, namely snapshot isolation, relaxes the isolation property.

## Consistency

### Consistency in Data

Pictures			Picture_Likes	
ID (PK)	BLOB	LIKES	USER (PK)	PICTURE_ID (PK)(FK)
1	xx	2	Jon	1
2	xx	1	Edmond	1
			Jon	2

DB is Invariant. When we have multiple replicas, we may encounter inconsistent reads due to the time it takes to clone data to each replica.

## Durability

Committed transactions must be persisted in a durable non-volatile storage

**Check for transaction log if data from RAM not write to SSD/HDD**

It's happening because DB is optimizing write queries and don't write straight to the disk

## BASE

Agenda:

- Basically Available
- Soft State
- Eventually consistency

I came up with [the BASE] acronym with my students in their office earlier that year. I agree it is contrived a bit, but so is "ACID" -- much more than people realize, so we figured it was good enough.

- Eric Brewer

## Basically Available

This property covers 2 terms in the BASE and is a major factor in making the database highly available. Against waiting for the transactions to process BA said that all the operations should happen at the same time in an ideal space. So each transaction will not wait for other to finish thus making 0 delay within the transaction phase

In addition, we ensure that the system will respond in the event of any partitions.

## Eventual Consistency

Let's take a look at **E** firstly, because **S** is a derivative.

Since transactions are not waiting for each other to finish and not letting them in isolated will cause a multi-direction flow of data entry on the same record. Considering at some time same record will stop updating as there might be no request comes to update will lead to consistency

Simplier, it indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

## Soft State

The state of the system is always soft which means that it can change over time even if there are no reads or writes as the system keeps changing the data to make it consistent.

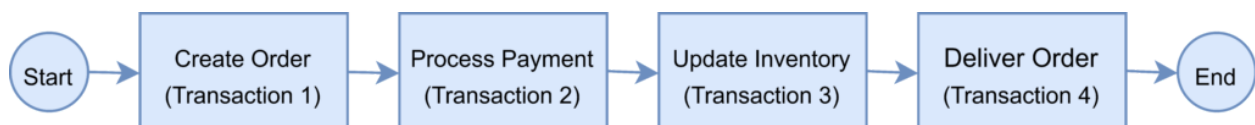
Indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.

## [Additional topic] Distributed transaction in microservices

Imagine we have a DB per service, so how handle ACID in this distributed app ?

### Distributed Transaction

As an example, we take an e-commerce site with microservice architecture

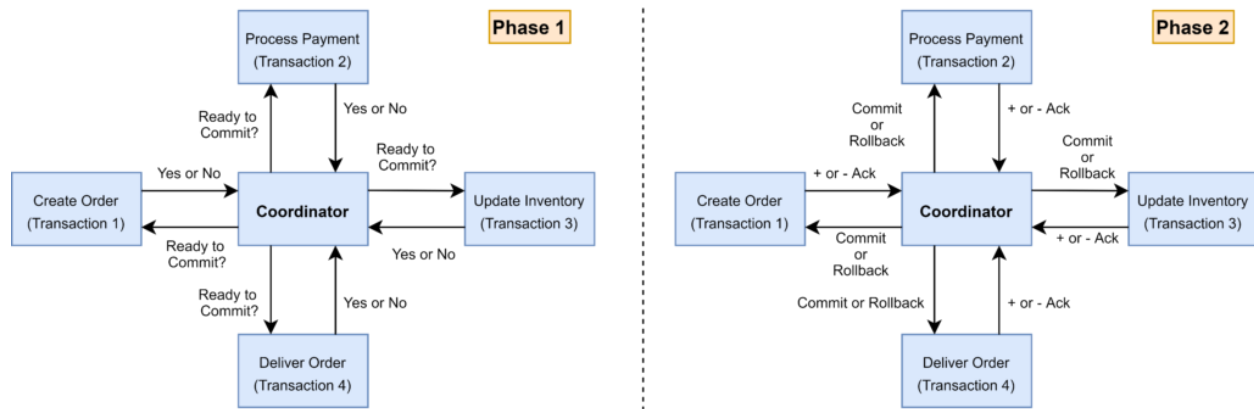


To ensure a successful order processing service, all four microservices must complete the individual local transactions. If any of the microservices fail to complete its local transaction, all the completed preceding transactions should roll back to ensure data integrity.

**So, how to maintain ACID in this scenario ?**

### Two-phase commit





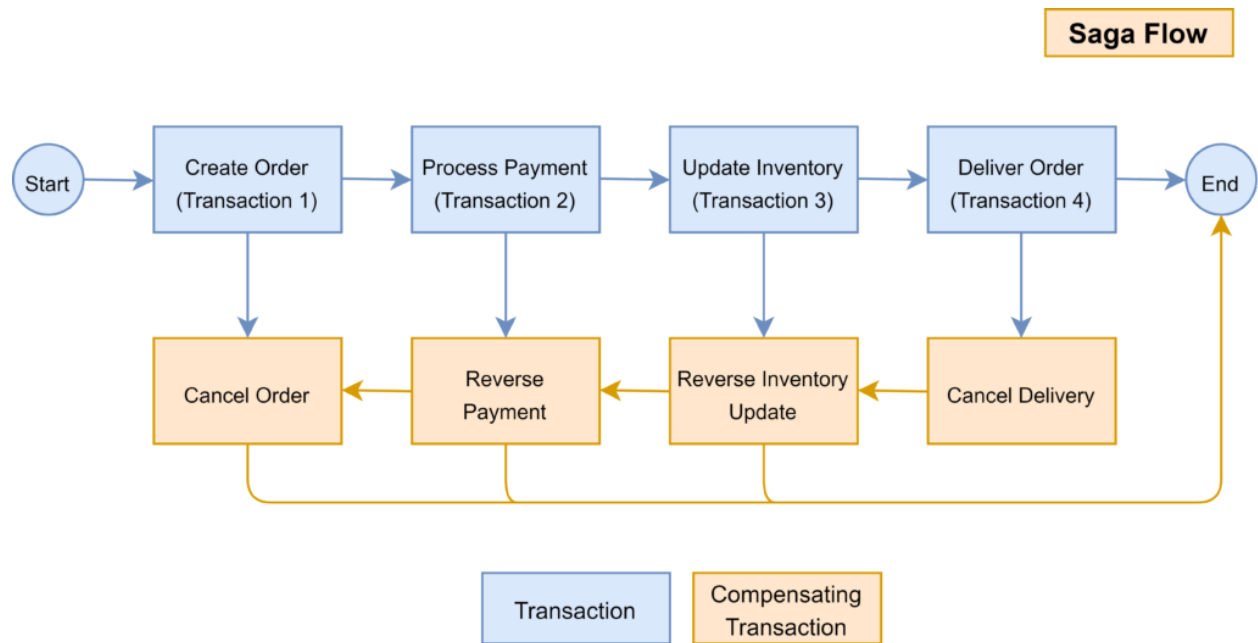
1. **Prepare Phase (phase 1)** – The coordinator asks the participating nodes whether they are ready to commit the transaction. The participants returned with a *yes* or *no*.
2. **Commit Phase (phase 2)** – If all the participating nodes respond affirmatively in phase 1, the coordinator asks all of them to commit. If at least one node returns negative, the coordinator asks all participants to roll back their local transactions.

### What problems can arrive ?

1. The coordinator is a single point of failure.
2. All services are waiting while the slowest one completes the transaction
3. Hard to scale
4. The two-phase commit protocol is not available in NoSQL databases. This means that in microservice architecture where one or more services rely on NoSQL databases, the two-phase commit cannot be utilized.

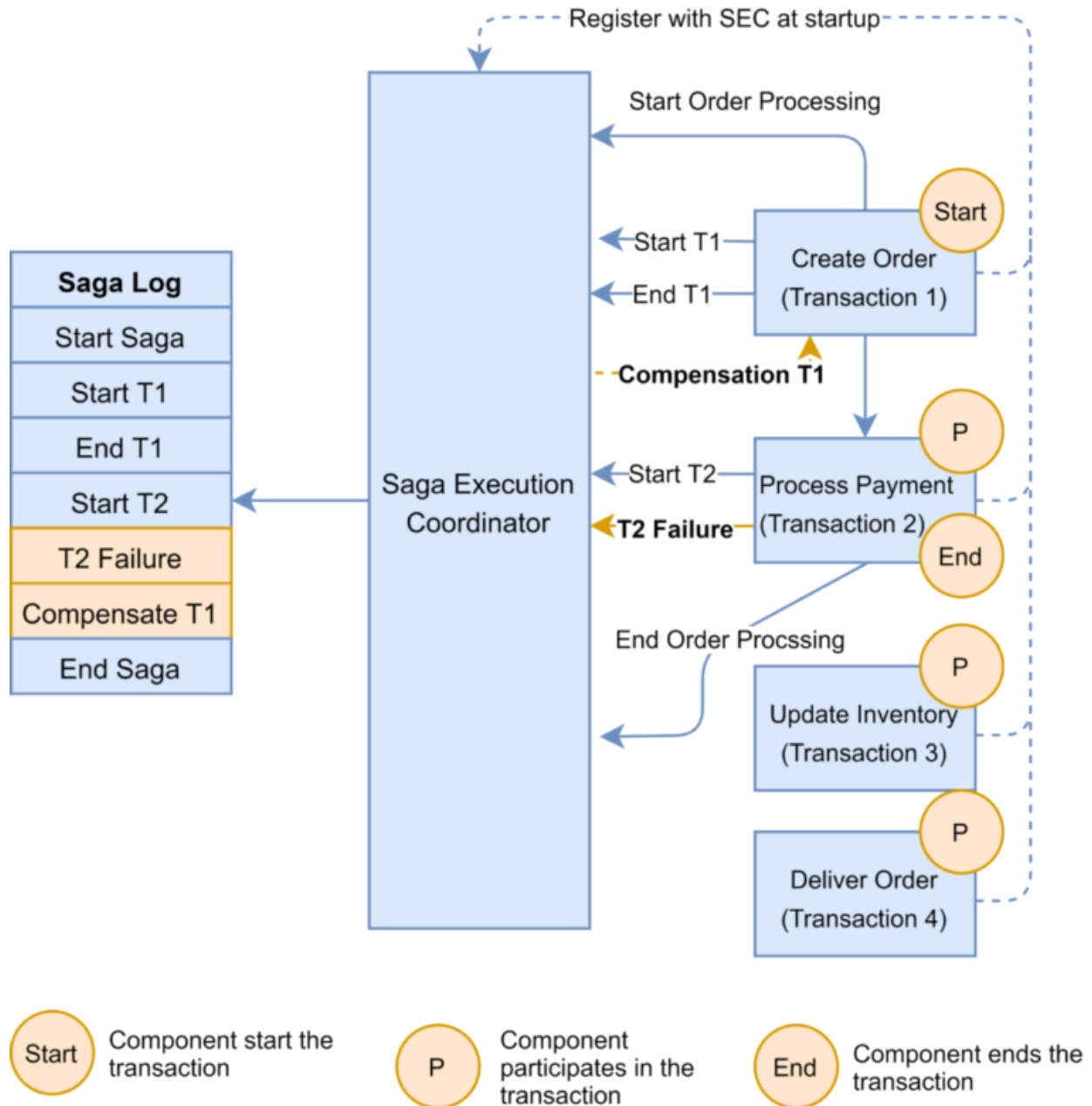
## SAGA pattern

Every operation has rollback

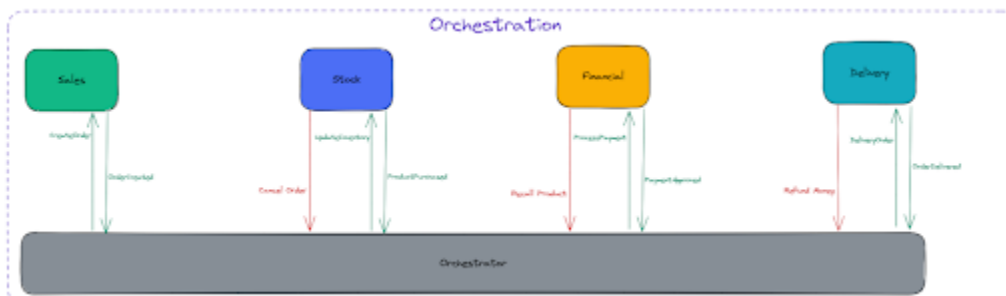


There are 2 implementations of this pattern:

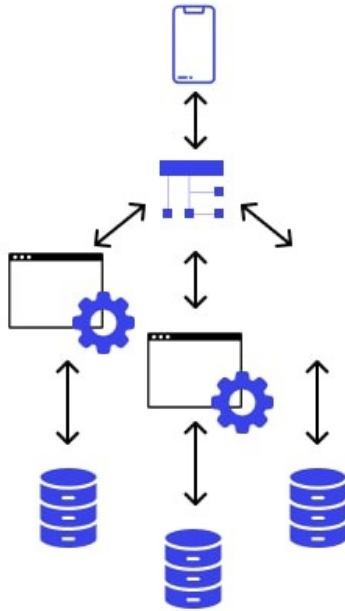
### **SAGA Choreograph**



## SAGA Orchestration



## Orchestration



## Choreography

