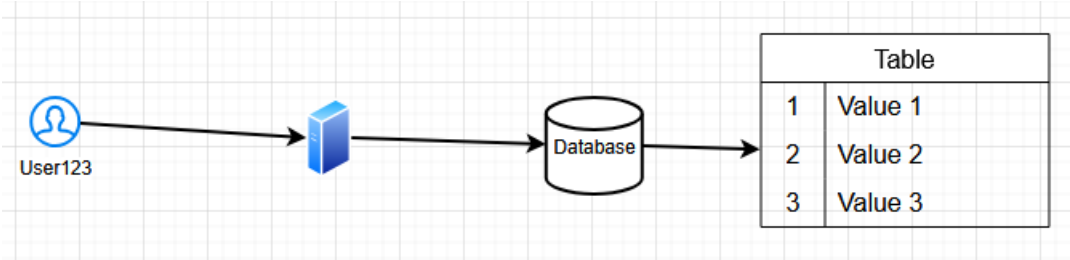
SAGA
To overcome issues with 2PC and 3PC we have Saga

When we have a long series of movies, we can call it a Saga like Twilight saga

A long running transaction thats why it is called as a Saga

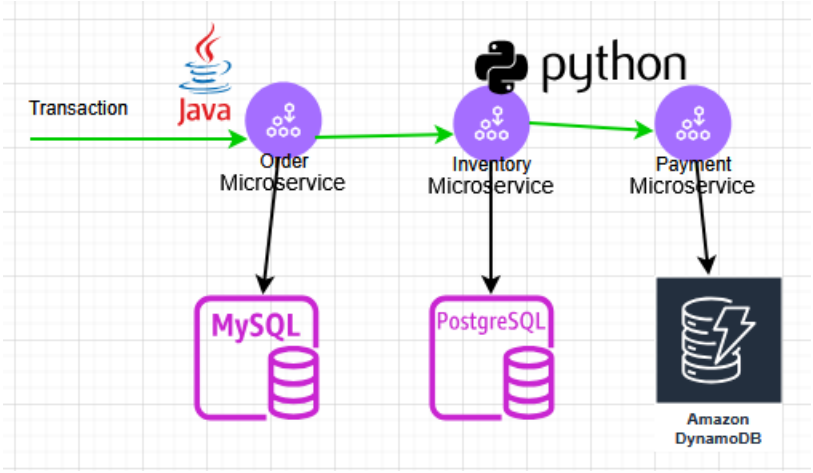
Two kinds of application:
Say we have a Monolithic for e-commerce. Single server and a single database and Microservices

Single server application:



Say we have Order table, Inventory table, Payment table. When a Customer places an order, we got to update all these 3 tables. When all 3 tables are updated, then the order is placed. When all transactions are done then it is a successful case. The transaction can fail at any level, Order, Inventory or Payment. If it fails at any level, we perform a rollback of the lengthy transactions. Thats why it is called as a Saga because they are lengthy transactions.

Microservices:
We break-down into different services, handled by different servers
Order service, which will be handling the order data.
Inventory service, handling inventory data
Payment service, handling payment data



All three services are independent so we can use Java Springboot on one machine, Python based application on another machines etc. We are getting good scalability but we are making our system complex to talk to each. Now our transaction has to go through multiple services to update tables. Picture shows the transaction flow, goes from one service to another. In a happy scenario, we get an Ok response back to the User. Lets say we are moving through different services in parallel and if one of the service fails but the other two succeeds. Only Inventory database is not updated but Payment was successful then what happens? It could happen in any other booking applications also. It's a challenging task to capture failures when there are especially multiples services. How will be reverse the order? What SAGA suggests is, with every operation, we will have a rollback compensation function() attached to each of the services. When things go wrong, we will call the Compensation function to revert.

1. Create order
    - Compensation: cancel this order function()
2. Update inventory
    - Compensation: Undo or Redo the update
3. Charge payment
    - Compensation: Do the refund, credit back the amount

Say Update inventory fails, we will trigger Compensation in both Order and Payment automatically

When you place an order, we send an email. Then if one of the services fail, they send another email that order is canceled. That's the downfall of this SAGA design. So we first send a success notification then if something fails, we send unsuccessful notification since notifications cannot be rolled back. Unlike 2PC or 3PC, where things work in a sequence, in SAGA, things can work in parallel.

Drawbacks:
- Some operations cant be reverted like sending notifications, emails. It can add operational cost of sending email again.

Homework: 2 ways of implementing Saga: Choreography (hint: every dancer is dancing without any instructor), Orchestration (hint: there is an instructor in front of the orchestra and he is managing the orchestration).