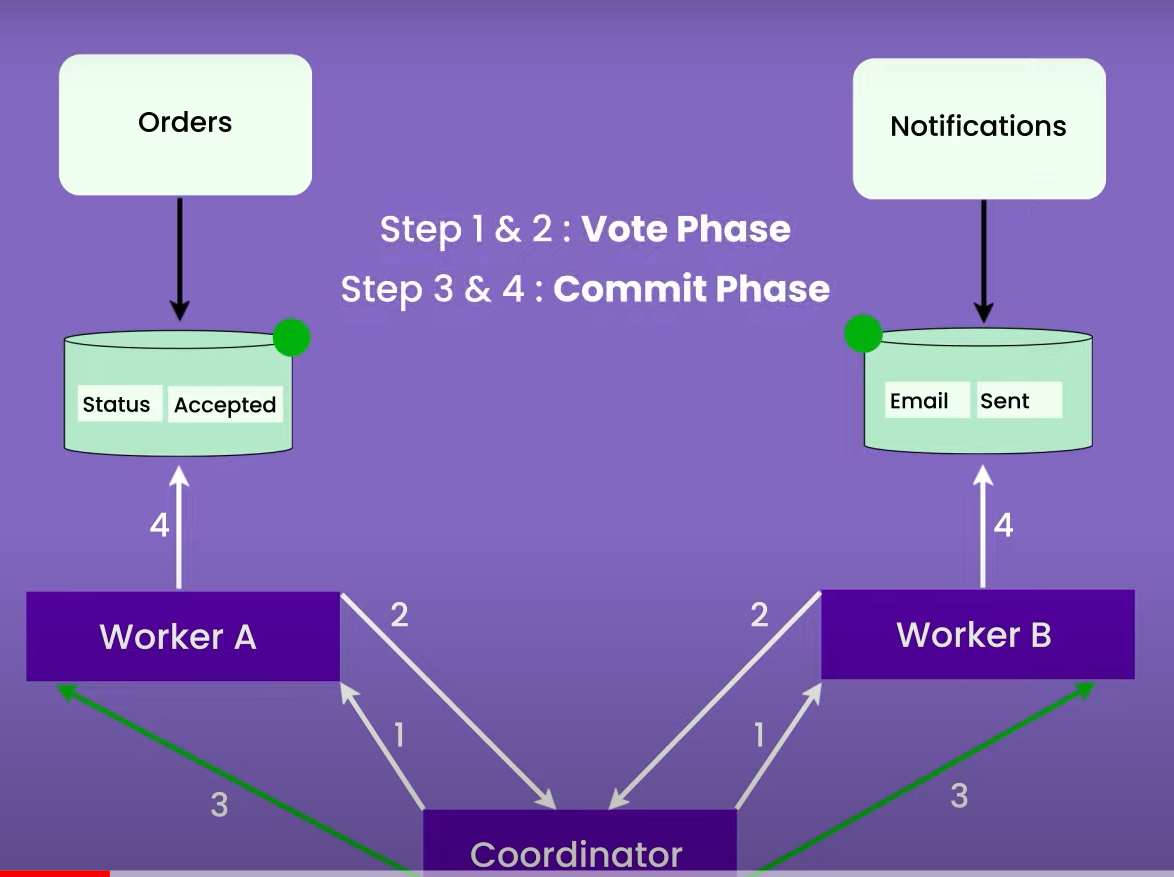
**What is Distributed Transaction**

* Distributed transactions in microservices bring complexity, performance overhead, and scalability challenges. Ensuring data consistency, implementing rollbacks, and handling availability issues can be problematic. Alternative approaches like compensating transactions or distributed sagas are often used to address these challenges while preserving microservices' benefits.

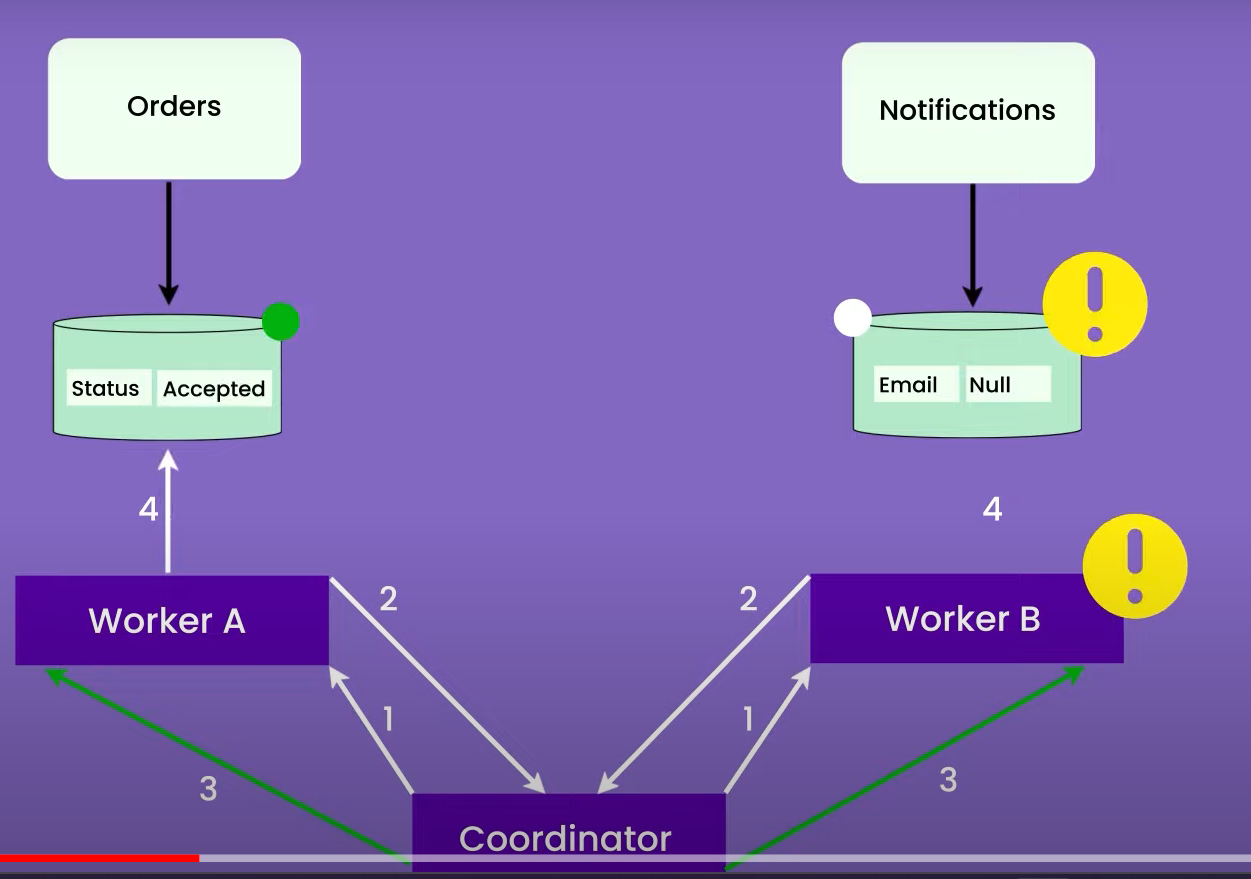
**Types of Distributed Transaction**

1. Two Phase Commit

* It has the Workers and coordinator to handle the transaction with the multiple services
* There are two phases like vote phase and commit phase.
* It take the lock on the table rows once both the service says yes we are ready to commit updates



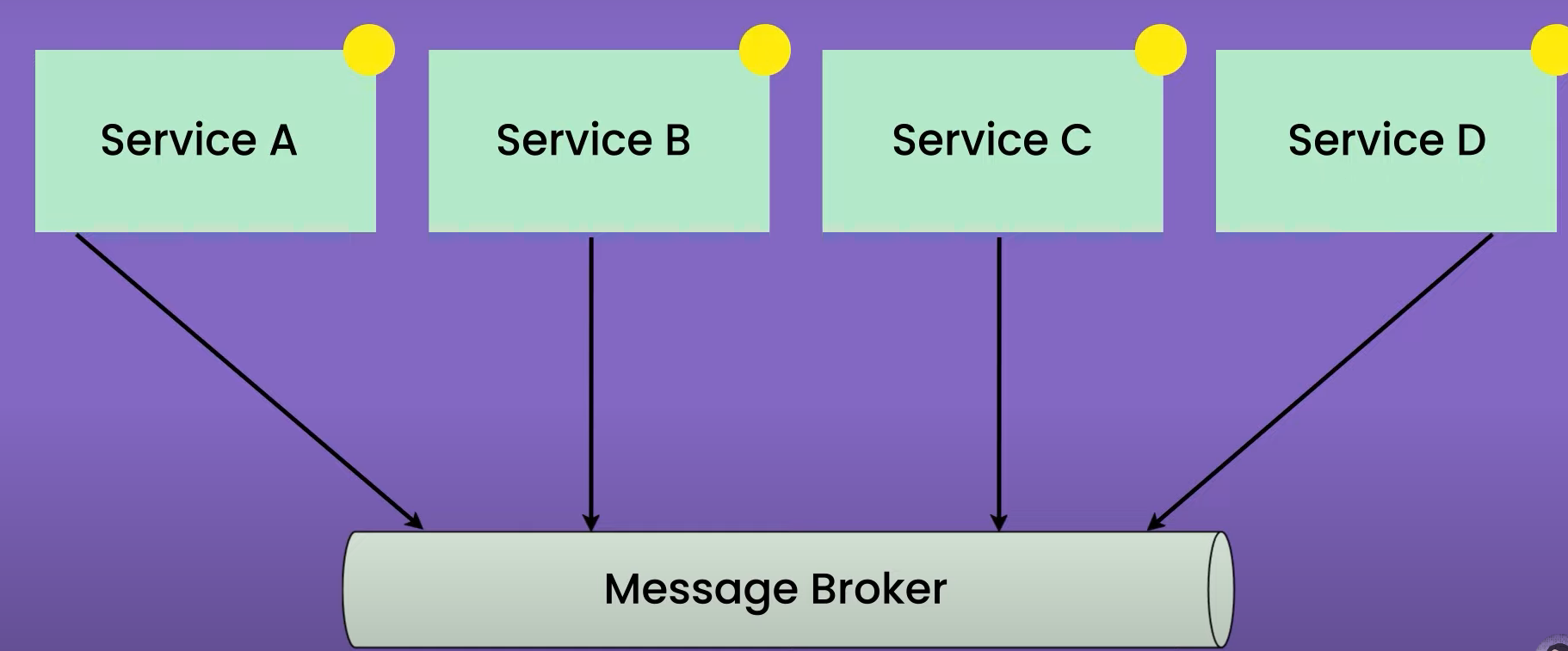
* But the problem comes when we have millons of transection to commit in DB and having loc on the table multiple time is risky and increase the latency of the system.
* Also it is risky to hold on the tables for longer period of time
* This problem is also call Long Living transaction problem
* Anther problem is like if any of the service goes down in this case the whole system will faill
* Not good for growing data



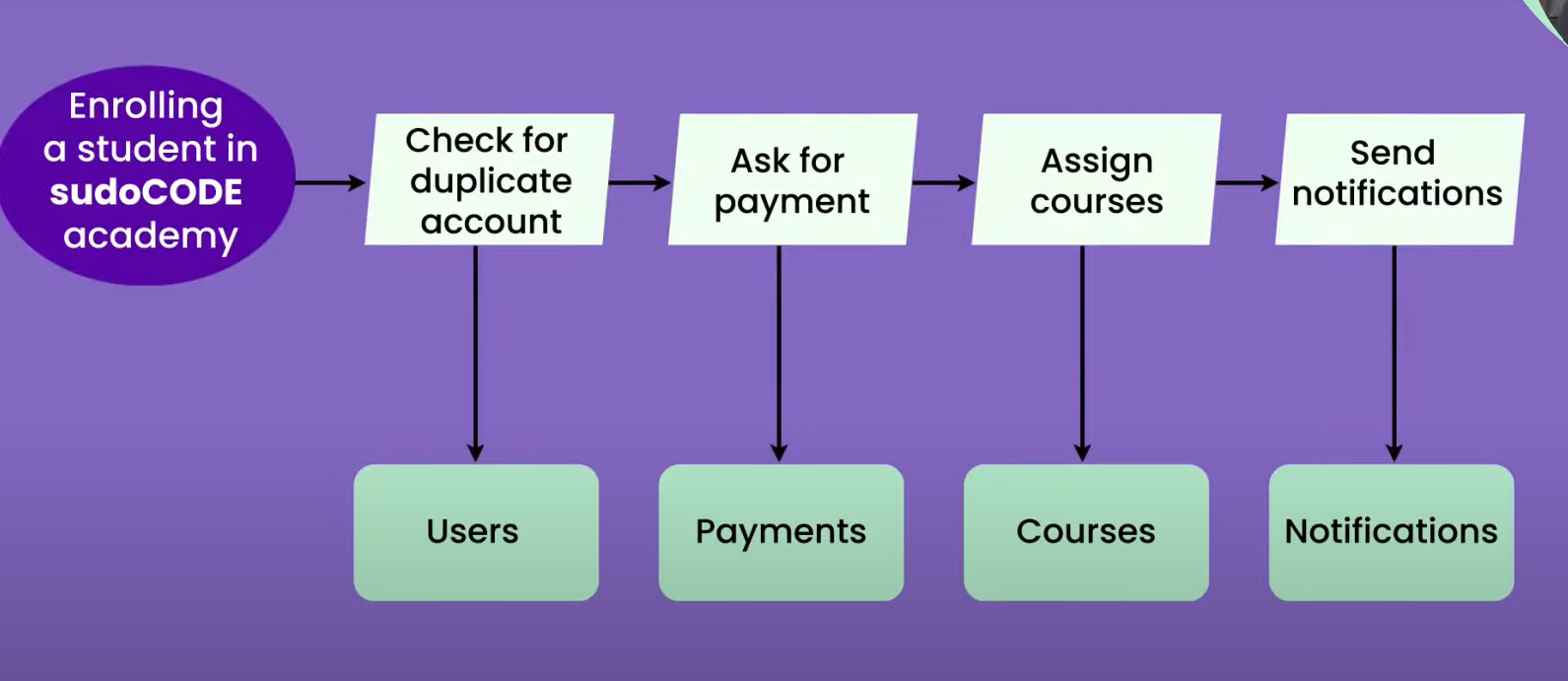
**Solution :**

1. **Saga pattern**

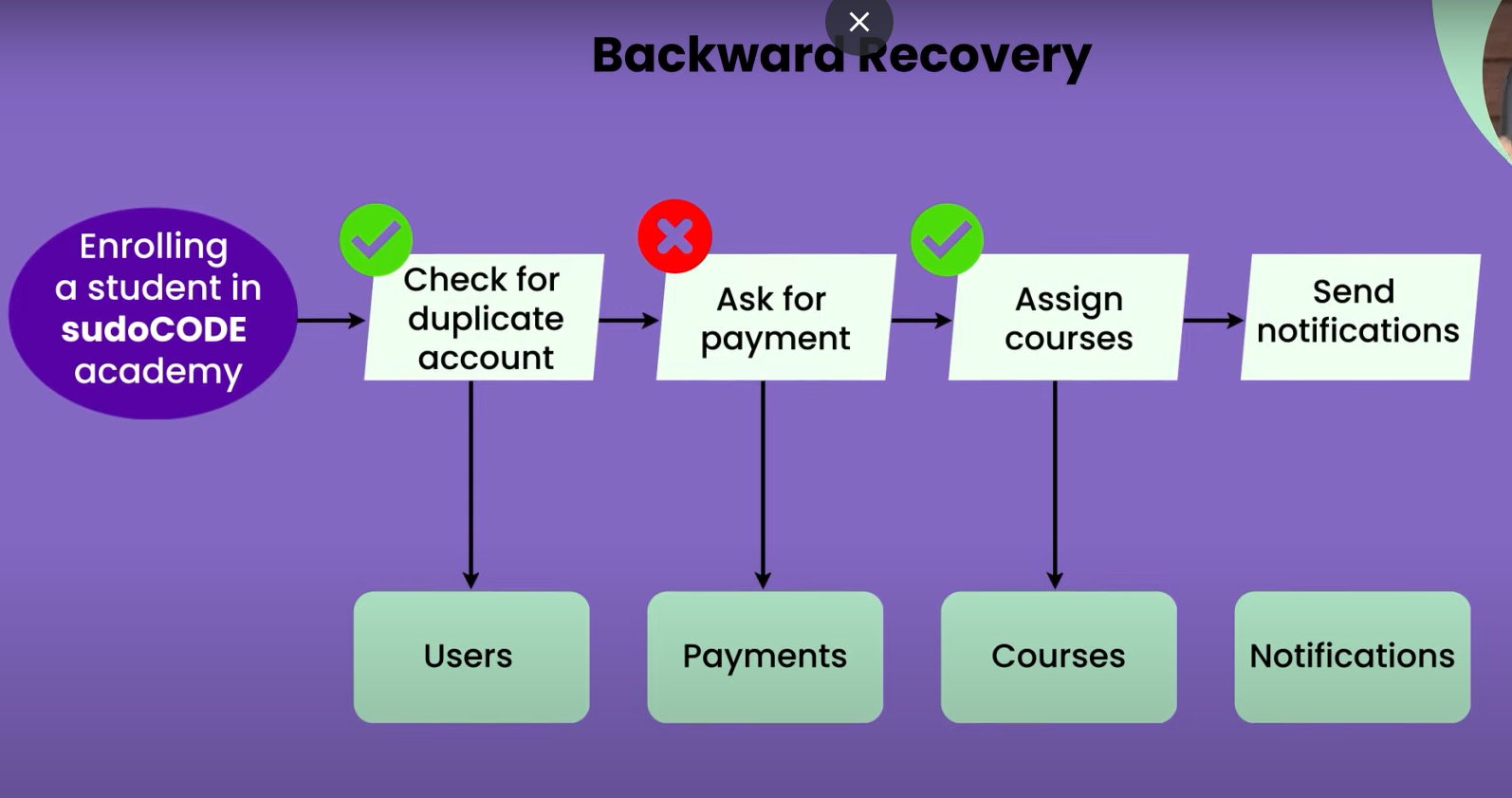
* Saga is a sequence of transaction that update each service and publish a message and event to trigger the next transaction step



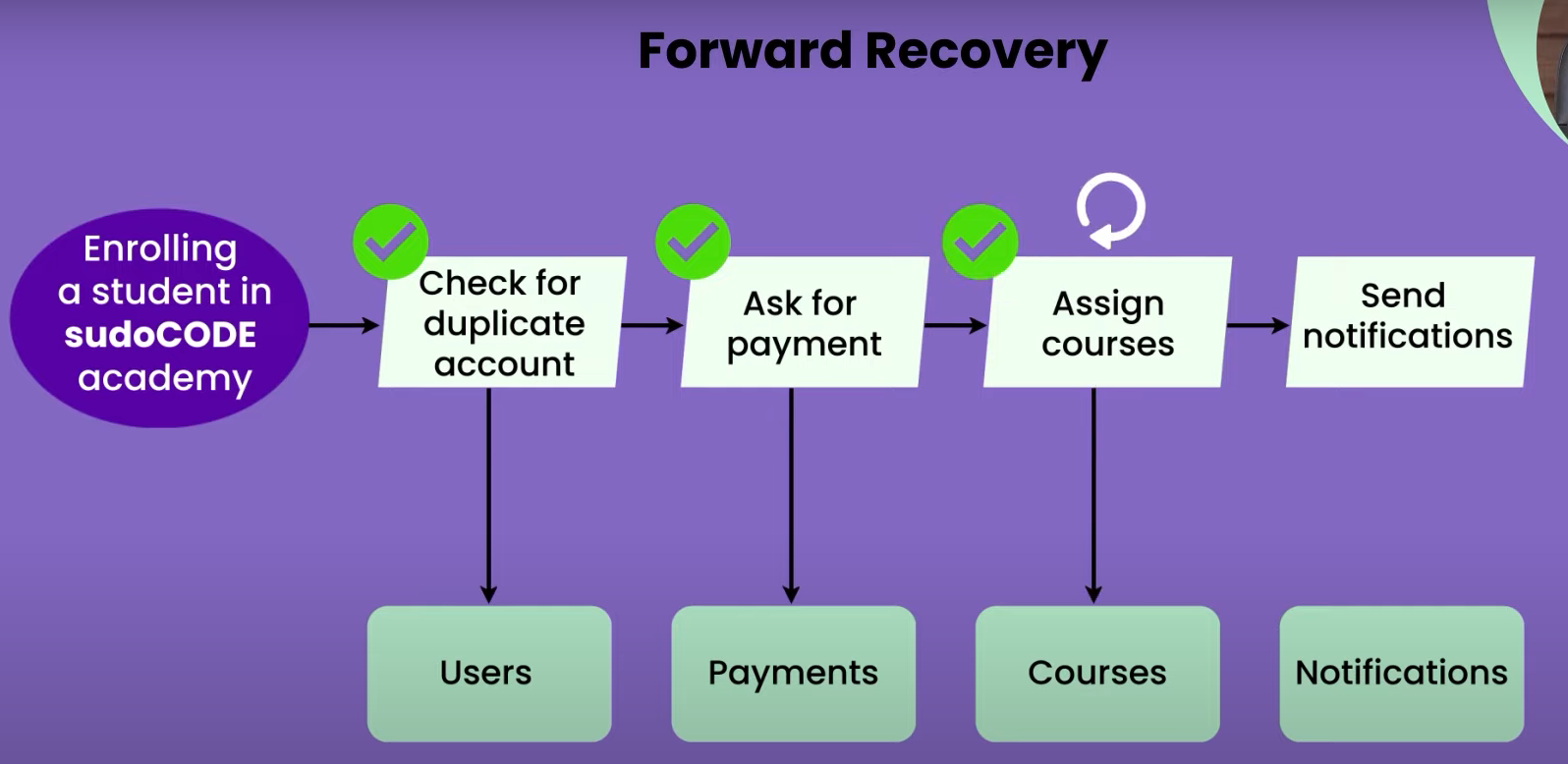
* Lets say you have the Enrolment student system where some of the funcnality like below



* And in the middle of transaction payment phase get failed, so what we can do in such scenario.
* Backward Recovery



* Here in this case we go back and revert the transaction like assignment of courses Payment and remove the profile and send the notification recreate the profile again.
* **Forward Recovery**

****

* In this case we can retry the Payment phase and if it is successful then we can go head and complete the profile
* **Combination of both forward and backward Recovery**
* Here we can do the forward recovery first and if the traction is not able to complete then we can do backward recovery.
* It has two approaches of saga pattern

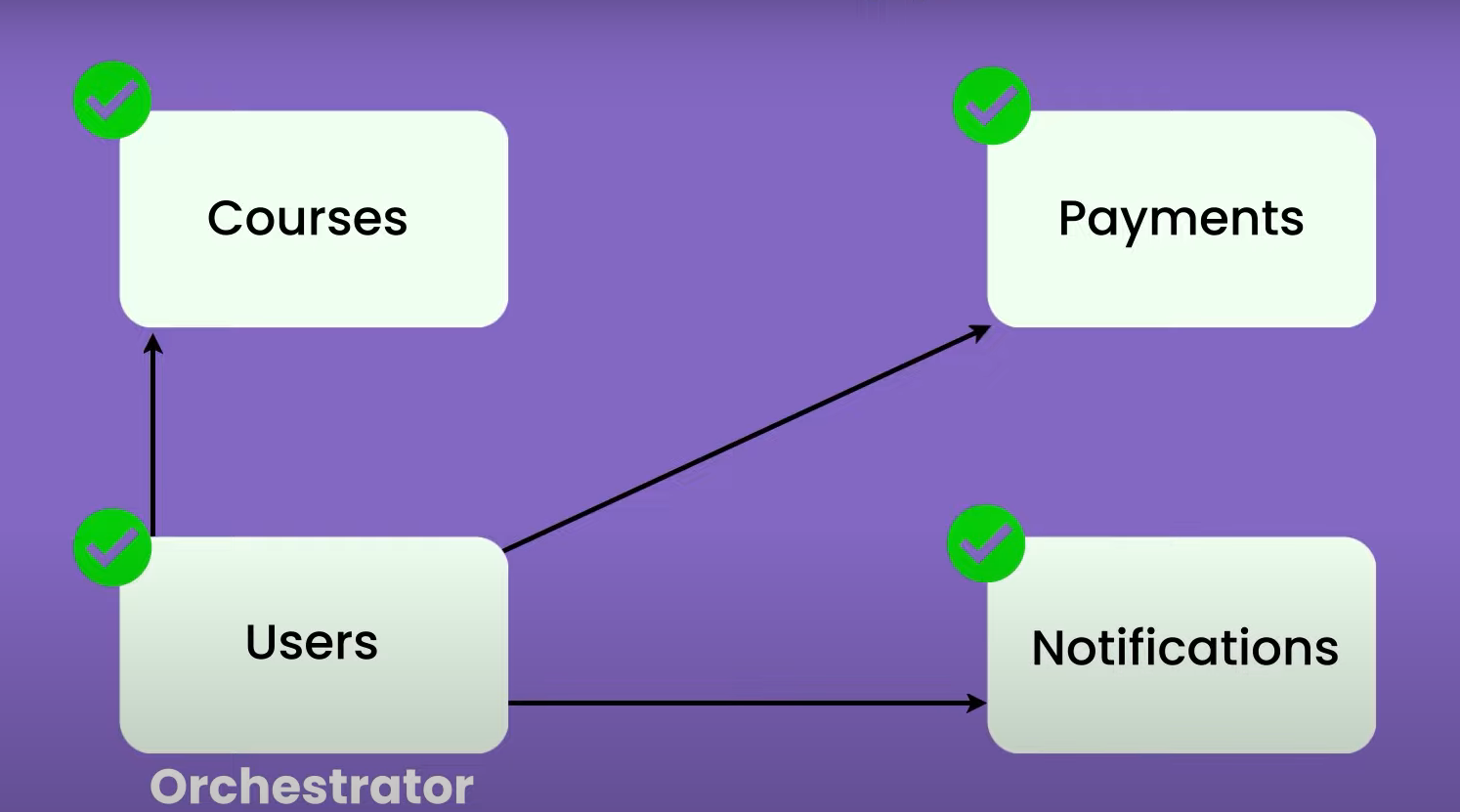
1) Orchestration approach

2) Choreography approach

* Orchestration approach :
* I t fallows command and follow approach
* Like a music orchestration a music person give the instruction to group of people to when to stat playing music.



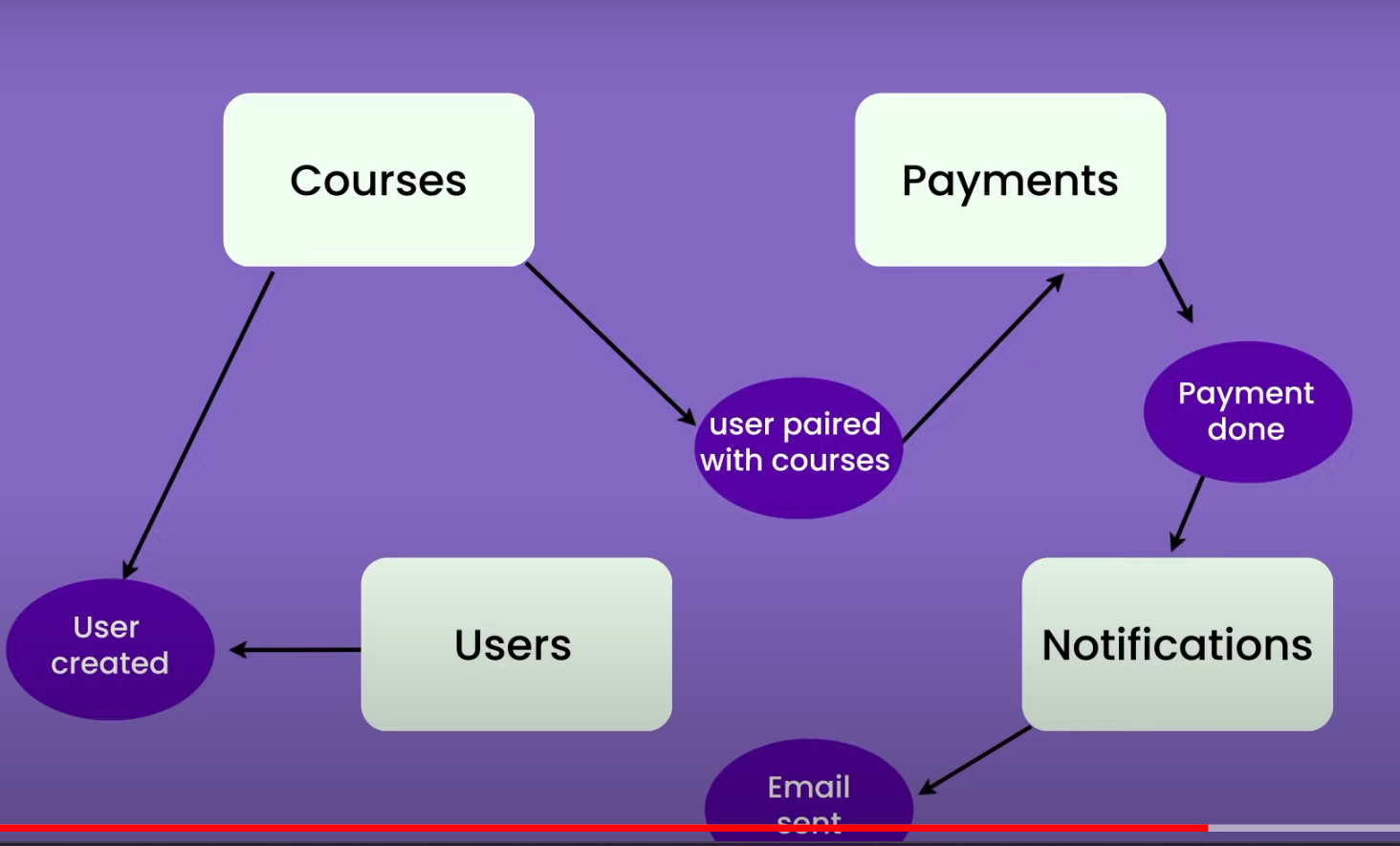
* In orchestrator approach one of the service take the leadership(orchestrator) to control the transaction management like user service.



* This service will ask all the other service to do there job and return to me .
* Choreography approach :
* It follows trust but verify approach (like I trust you will do better on stage but need to verify it before you go on stage by practice and rehearsal).



* In Choreography approach there is not controller or orchestrator.
* The services are independed and only depend on events.
* Services are communicate using event in MQ or streaming platform



Pros and cron :

* Need very good observability and monitoring system in of any of the service goes down
* Co-relation/requestId is used to figure out problem is associated with service
* This Id will help to find the logs for each of the service.