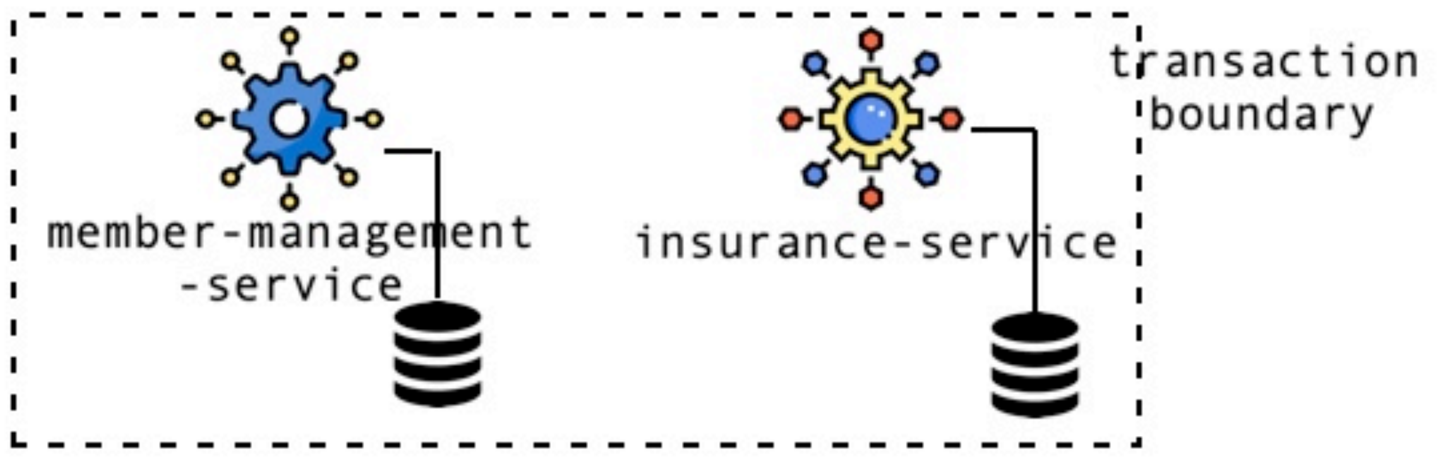
Requirement: Insurance
Issue an Insurance Policy to a New Customer

1. Managing the members is handled by member-management-service, like new, delete, update/edit etc
2. Policy management aspects are handled by insurance-service like, creating/issuing an policy to a member, renewal, changing the policy, withdraw policy etc

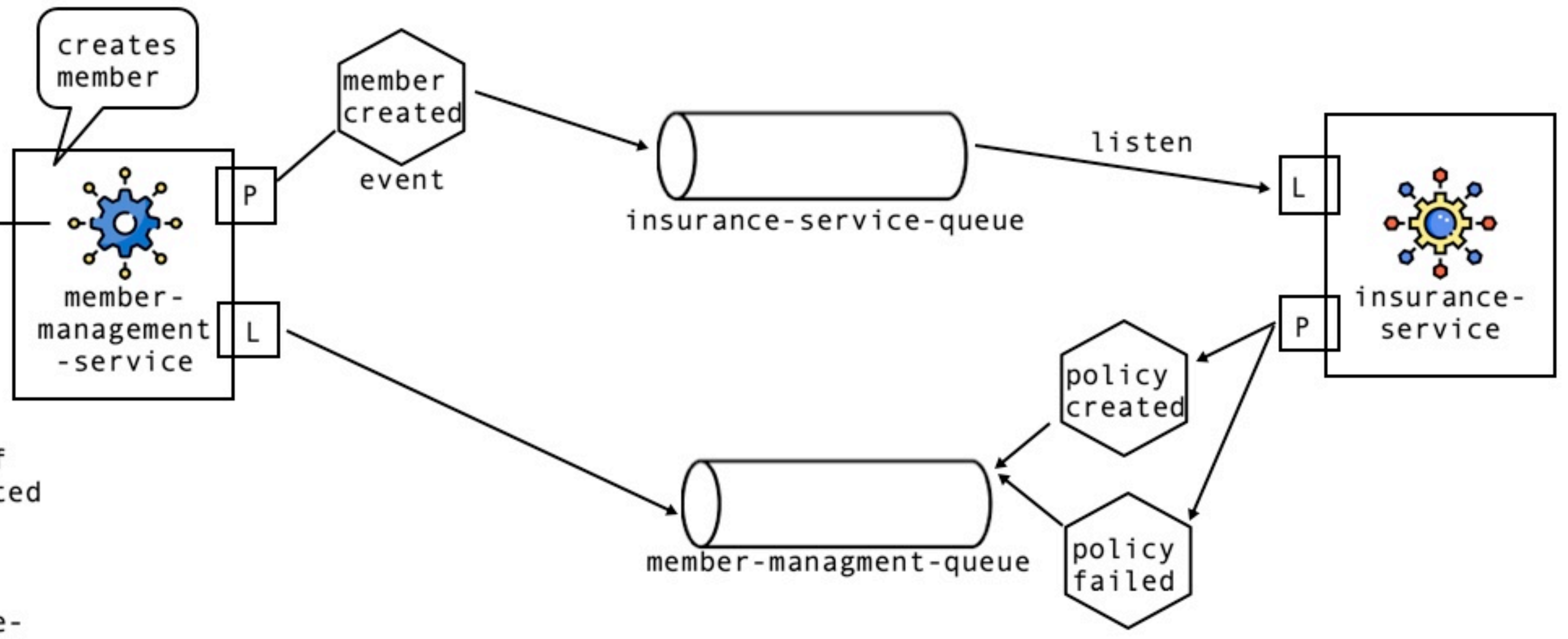
To issue an policy to an new Customer:
1. we need to create a new member by talking to member-management-service
2. then upon adding the new member, we need to talk to insurance-service with that memberid, asking to issue the policy.
hence, it requires an business operation spanning across the microservices. To handle this we can use Saga pattern



Choreography-Based Saga

asynchronous:

received a request for issue policy to new member
data: {
- member
- policy
}



member-management-service
1. creates an new member by making the status of the member as inactive. A new #member is generated by the service. The data will be saved/commited into the database.

2. publish a “member-created” event to insurance-service by wrapping #member and policy to created

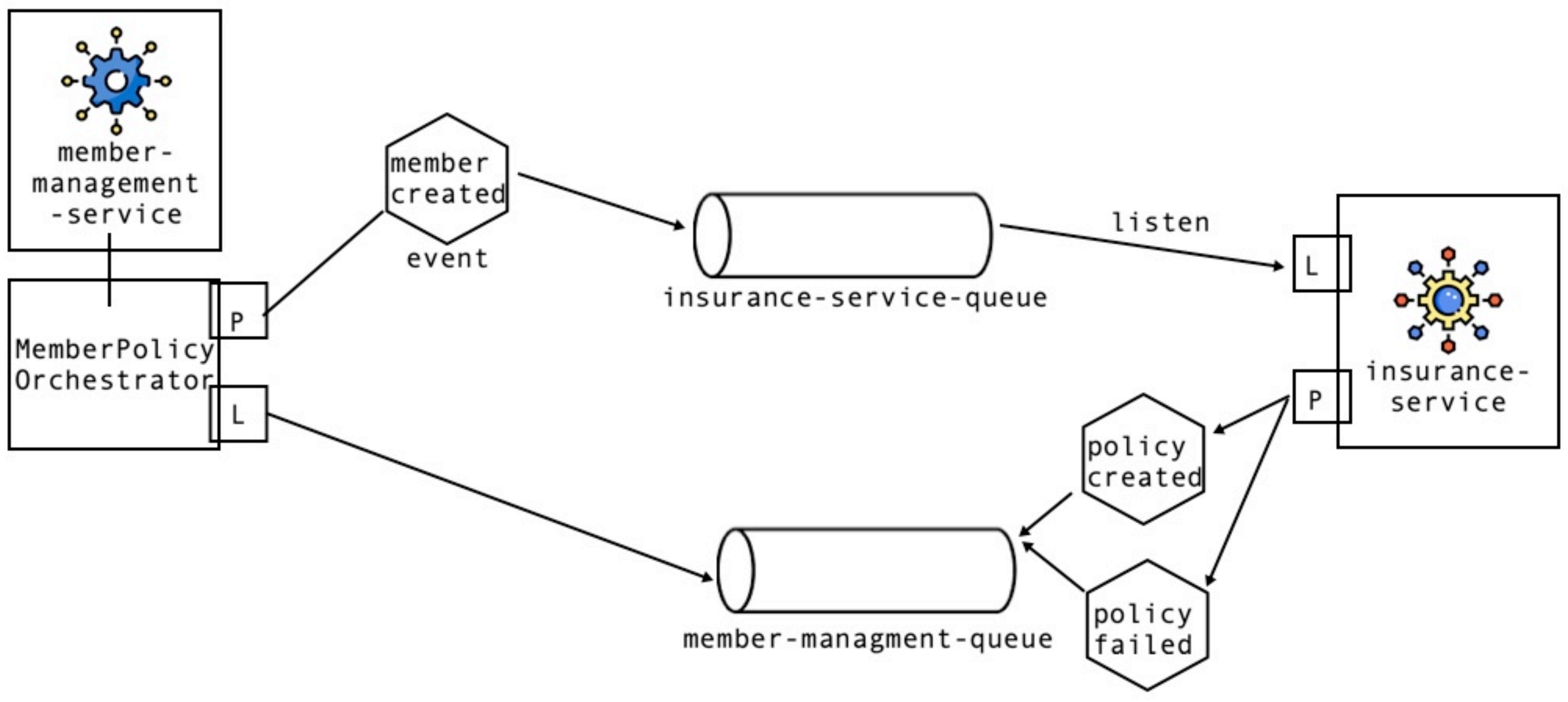
7. upon recieving an event in the member-queue by the member-management-service “Listener”

it checks whether the event is of what Type:
if it is policy-created eventType:
  then update the member record with status: active and send the response to the user with policy and memberno

if it is policy-failed eventType:
  perform compensating transaction in deleting the member that is created in past (rollback the system). and communicate the failure to the client.

insurance-service
3. The insurance-service has a listener listens for the member-created events from the queue.
4. upon an event has been received in the new-policy-queue with event as: member-created, it receives the event, extracts the data and performs operation of creating and issuing the policy
5. if all the business rules are passed and if the policy has been created and issued/assigned to that member by saving/commiting the data into database. it publishes an event “policy-created” with #policy into member-queue
6. incase creating an policy has been failed, due to violation or failures in business rules or error while performing operation, it creates an “policy-failed” event indicating it and publishes the event into the member-queue. and rollbacks the local transaction.

Orchestration-Based Saga
In the orchestration based saga pattern, a single orchestrator is responsible for managing the overall transaction status.


member-management-service
MemberPolicy Orchestrator
P
L
member created event
insurance-service-queue
listen
insurance-service
policy created
policy failed
member-managment-queue