Broken Promises

kiki @ lightbend

@kikisworldrace

Data is dangerous

Microservices are usually required to cooperate to achieve some end goal.

Microservices need to be able to trust each other in order to cooperate.

Microservices cooperate, they are symbiotic but not parasitic.

Microservices have a relationship to each other, but also a relationship with data.

There are different types of data that microservices must deal with.

1. Agent internal-data (aka entities)
2. Persisted Data
3. External Service Data

How data destroys relationships: Data can Lie

1. Lies of omission
2. Lies of commission

How data destroys relationships: Data can be Corrupt

Corrupted by good or bad actors.

How data destroys relationships: Data can be high maintenance.

Hard to secure some, but everyone pays the price.

How data destroys relationships: Data is dumb.

Data is not self-aware.

How data destroys relationships: Data is dumb.

Data is too dumb to drive things. To drive something you need to know direction, how to steer, how to interpret.

How data destroys relationships: Data is dumb.

Data can only inform.

How data destroys relationships: Data is dumb.

Stateless Architecture, another broken promise...

If a microservice cannot rely on data to properly inform, then what?

We need trust.

Microservices be able to make and keep promises.

Promise theory is an engineering framework for coping with uncertainty in information systems.

Promise theory is an engineering framework for coping with uncertainty in information systems.

Promise theory is an engineering framework for coping with uncertainty in information systems.

Promise theory is an engineering framework for coping with uncertainty in information systems.

Uncertainty
Divergence vs Convergence

Uncertainty
Divergence vs Convergence

Uncertainty
Divergence vs Convergence

Promises help you converge.

Obligations aren’t better?

Back to data and winning back trust

Facts

AKA Events

Facts

- Should be immutable
- Should be relevant, or germane to the state at play
- Could be used to assess whether a promise has been kept.

Opinions

AKA Views

Opinions

- Views should be generated as a result of ingesting facts.
- Views should be based on facts.
- Could be used to broadcast whether a promise was kept.

Systems are composed of facts, opinions. States and views.

Give data benevolent masters or stewards capable of making and keeping promises.

What makes a good steward of state?

What makes a good steward of state?

Should be smart, intelligent, knowledgeable

What makes a good steward of state?

Should produce the right, relevant data for its state.

What makes a good steward of state?

Should know what is a good fact vs a bad fact.

Masters need to have clear properly managed boundaries.

Context matters.

Masters need to have clear properly managed boundaries.

Context matters.

Masters should be resistant to corruption.

A good master is autonomous and responsible.

How does a stateful steward repair itself?

How does a stateful steward repair itself?

By rebuilding the state from events.

How does a stateful steward repair itself?

By using compensating actions.

How does a stateful steward repair itself?

Having and executing on info to resolve conflicts.

How does a stateless advisor, view creator, repair?

How does a stateless advisor, view creator, repair?

Rebuild views.

Some views need to be fact based, others do not.

Some views need to be fact based, others do not.

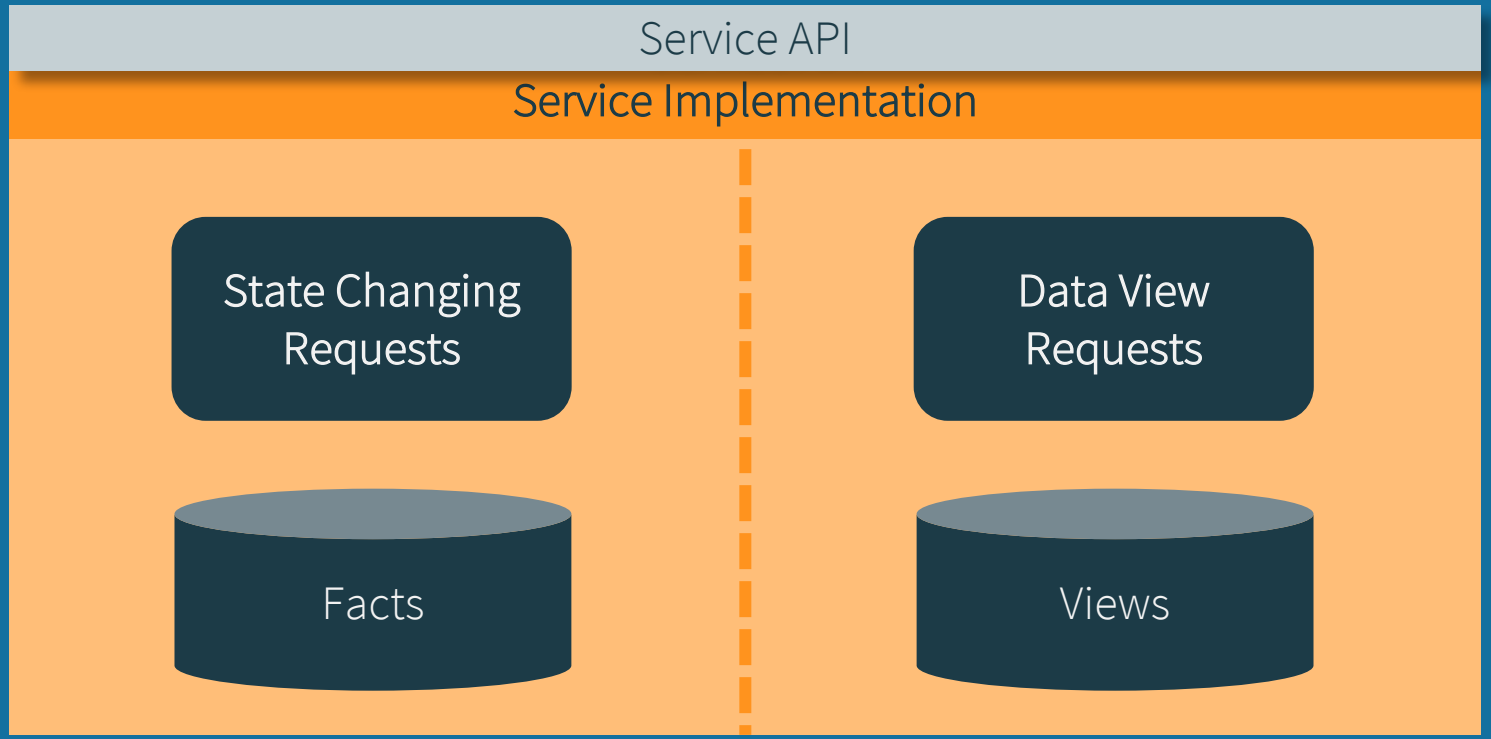
Some views need to be fact based, others do not.

Good masters aren’t data-driven, they drive the data

Data should be isolated

Isolation; bad for humans, good for data

Promote Isolation & Autonomy by Separating Facts from Views

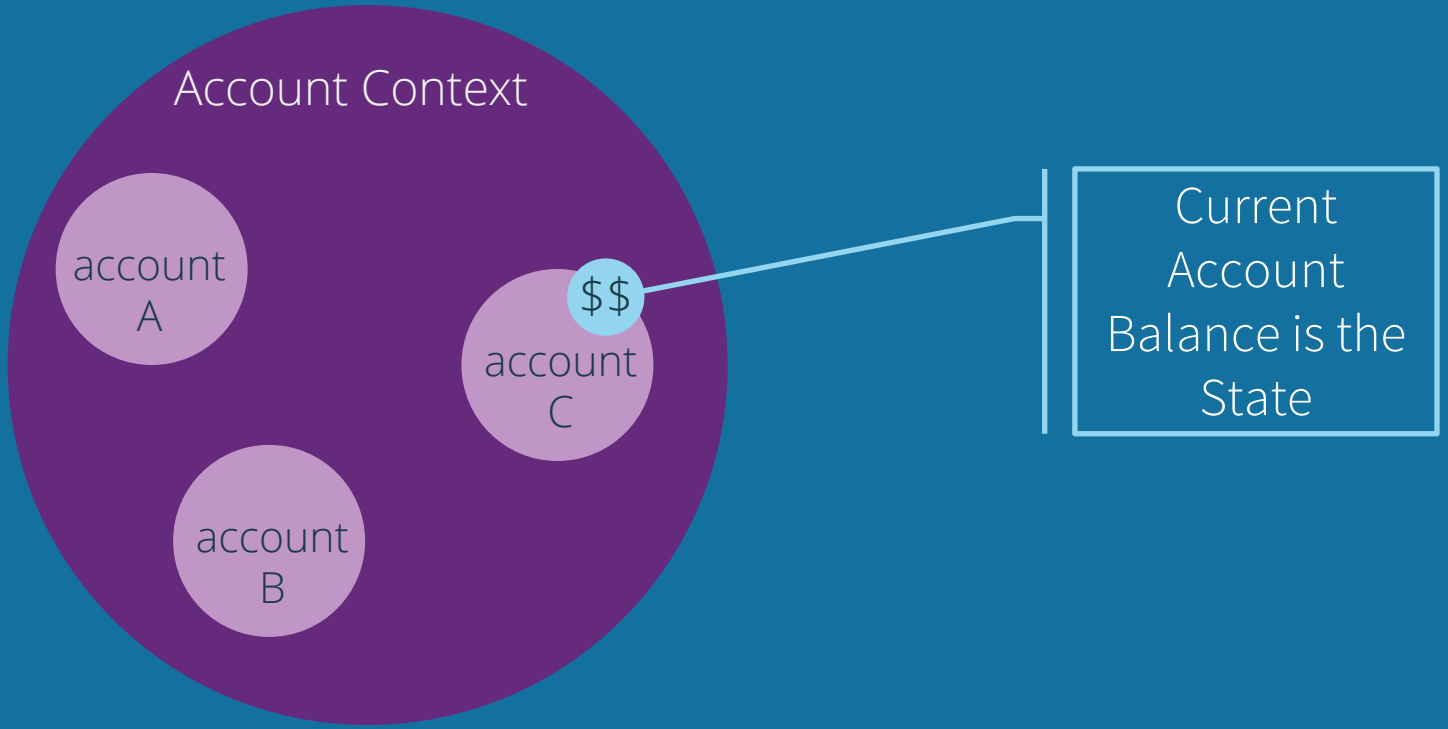


Enter Lagom

Lagom empowers your master stewards through isolation, autonomous services

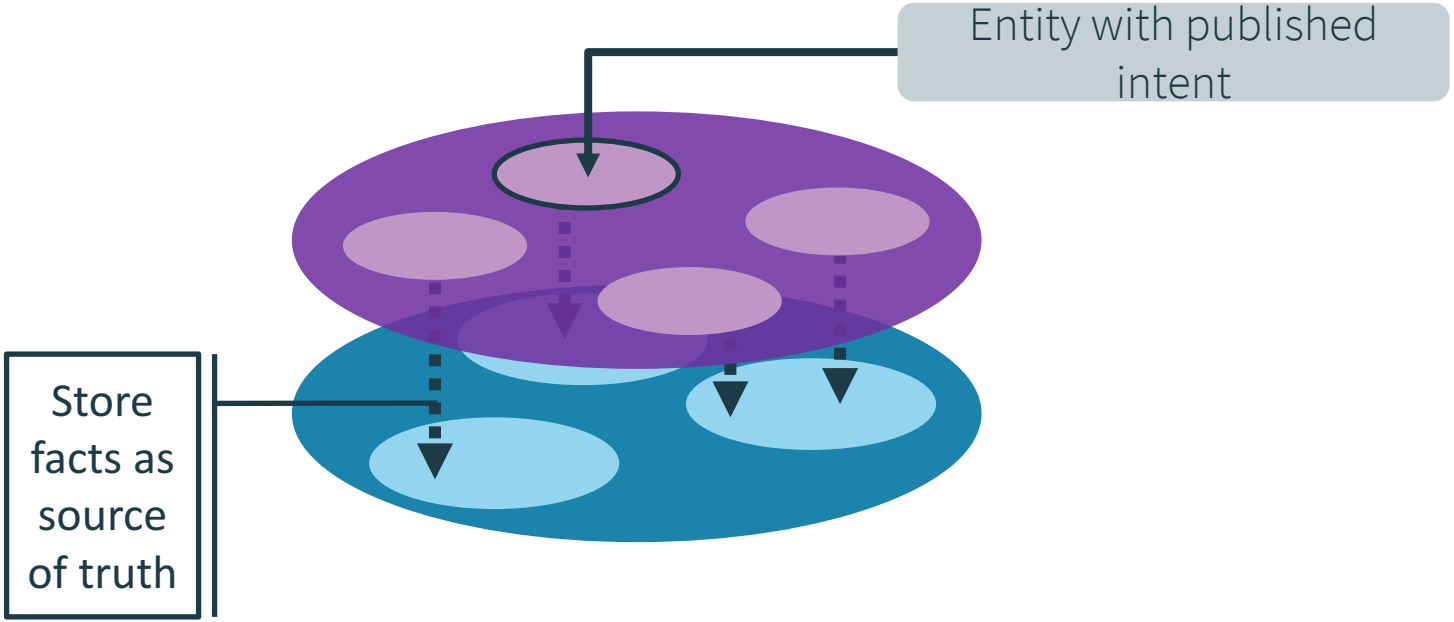
Remember - different types of data that microservices must deal with.

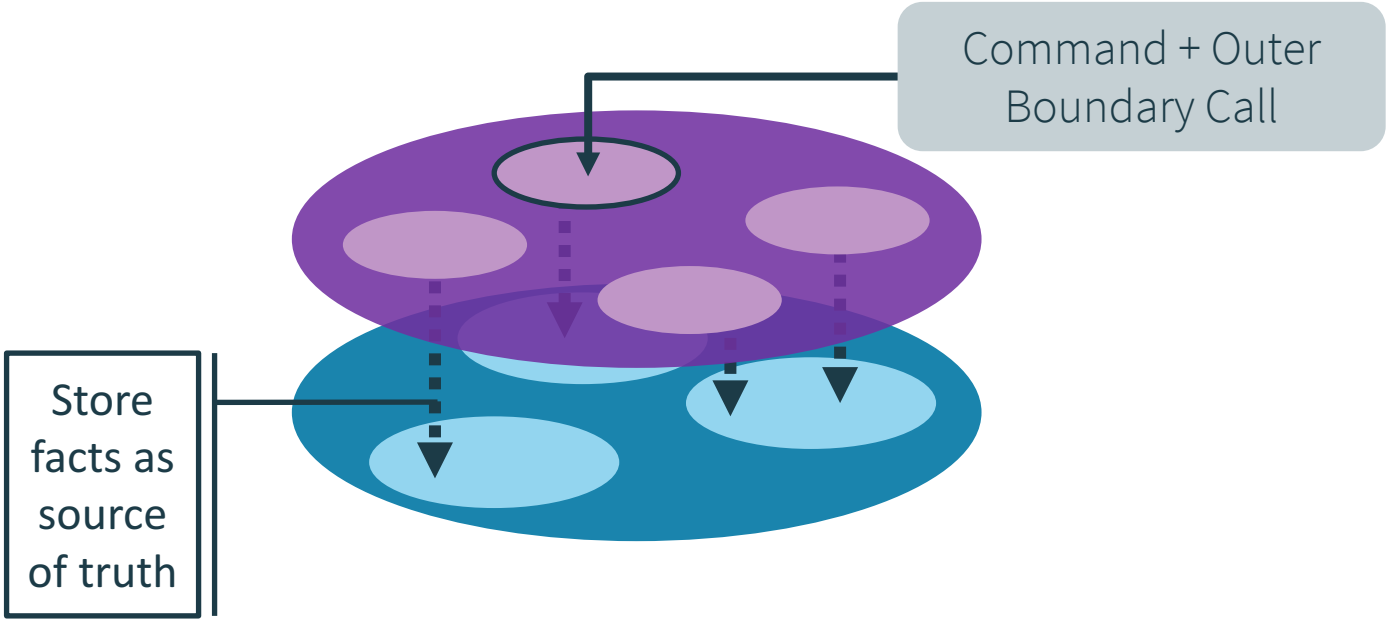
1. Agent internal-data (aka entities)
2. Persisted Data
3. External Service Data



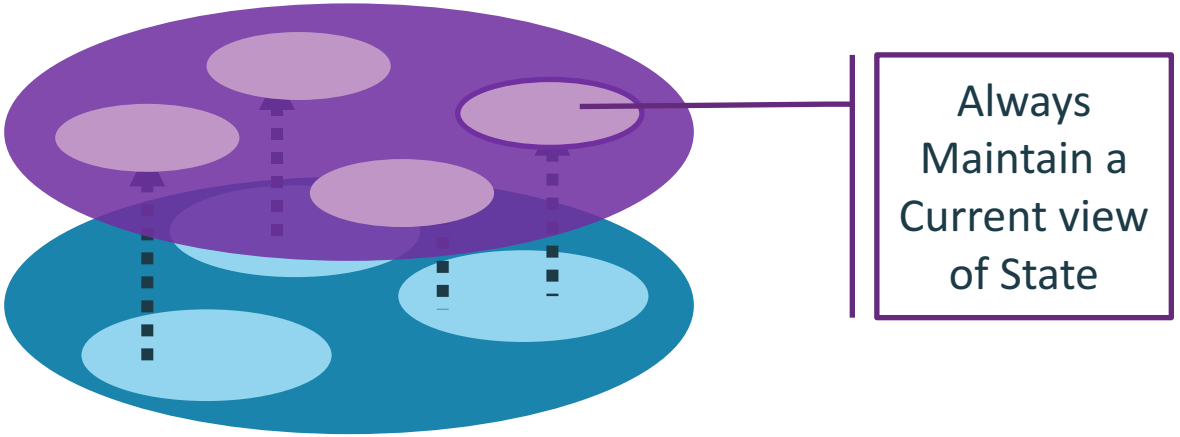
Agent Internal Data

Lagom Encourages and Enables You to Contain Mutable State & Publish Facts


Entity with published intent
Store facts as source of truth


Command + Outer Boundary Call
Store facts as source of truth

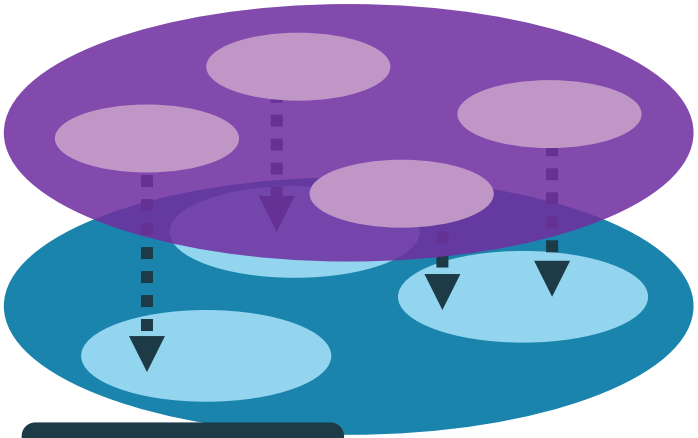
Entities in a Cluster



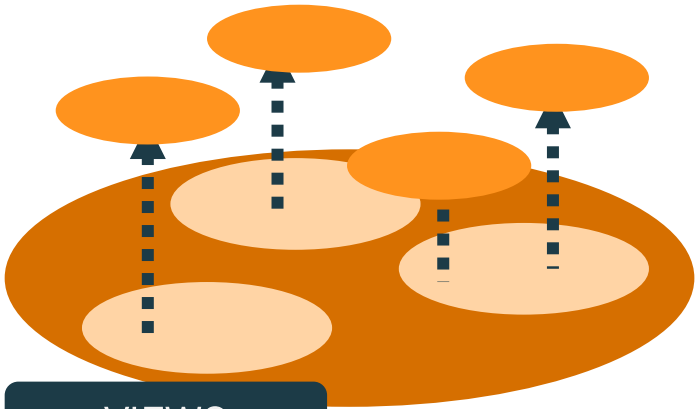
Lagom Allows you specialize in order to have the right properties for stewarding events/views.

Stateful Agents

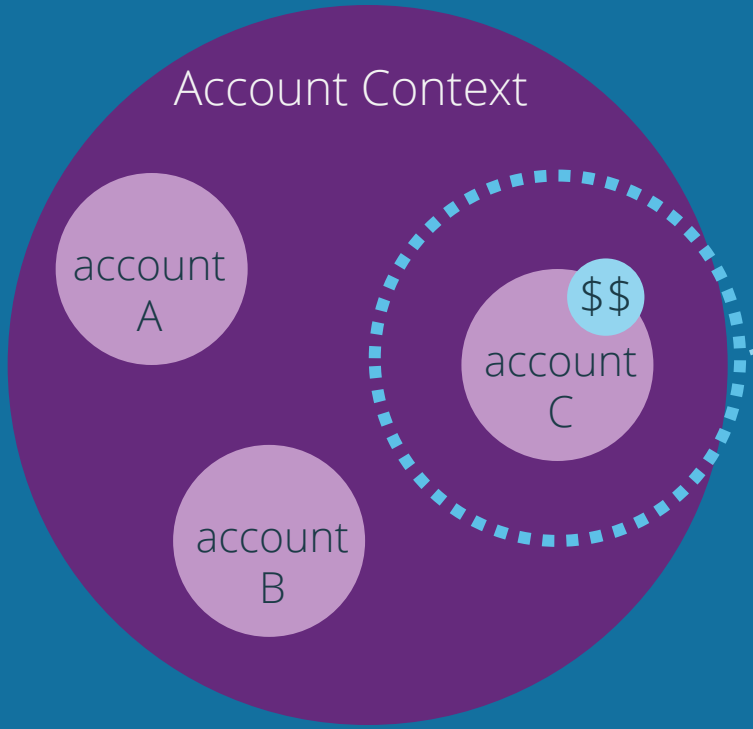
Stateless Agents



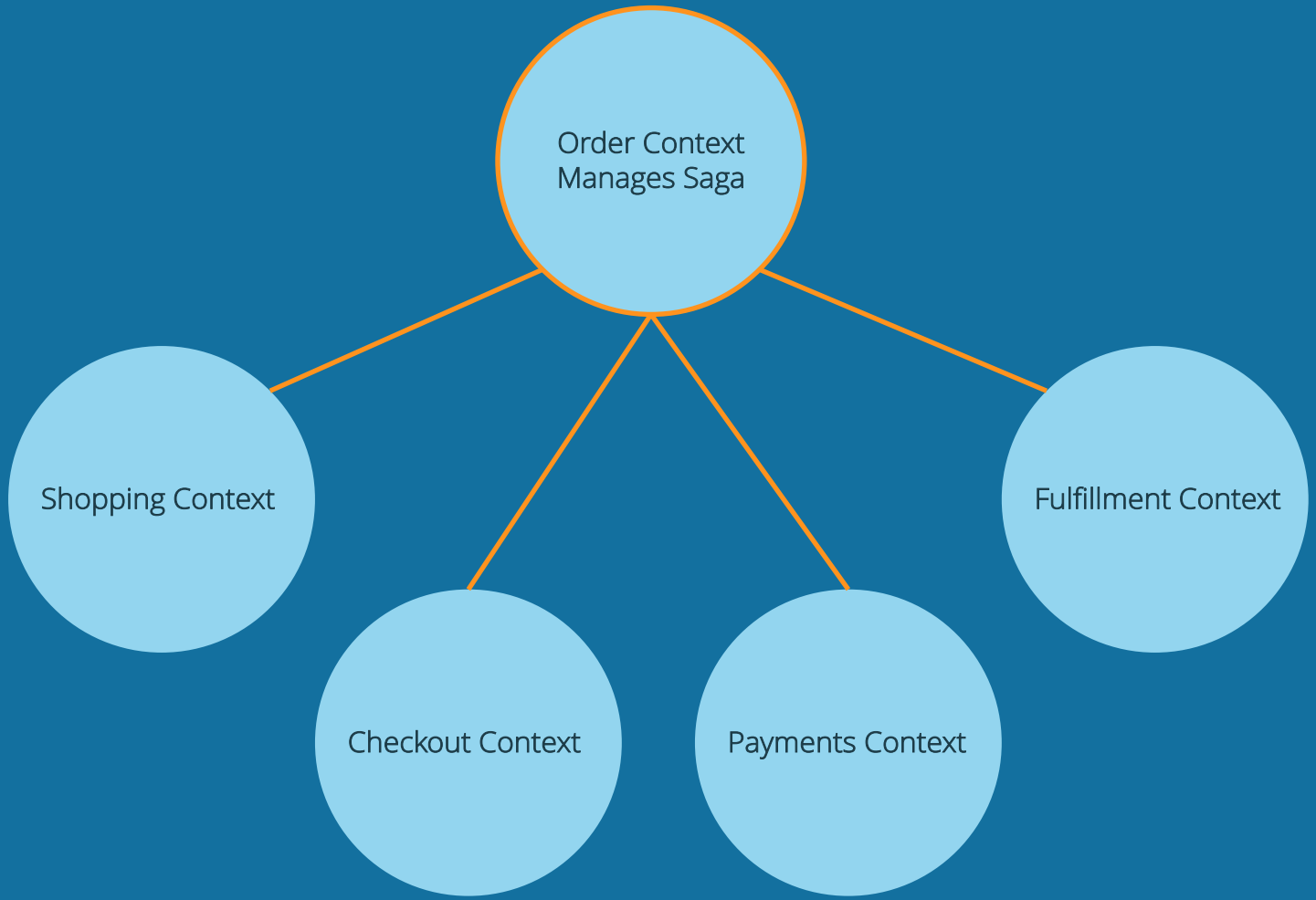
FACTS



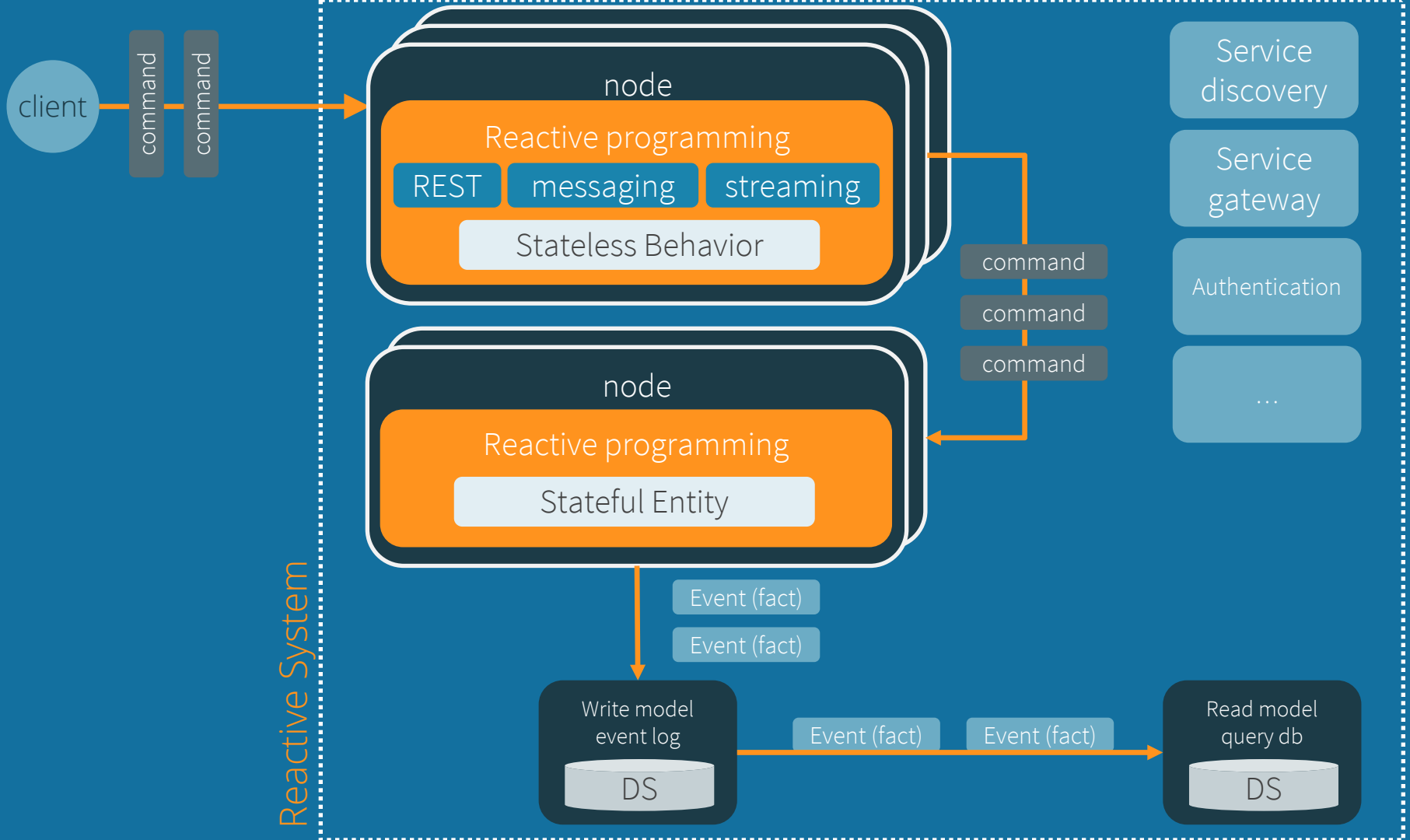
VIEWS

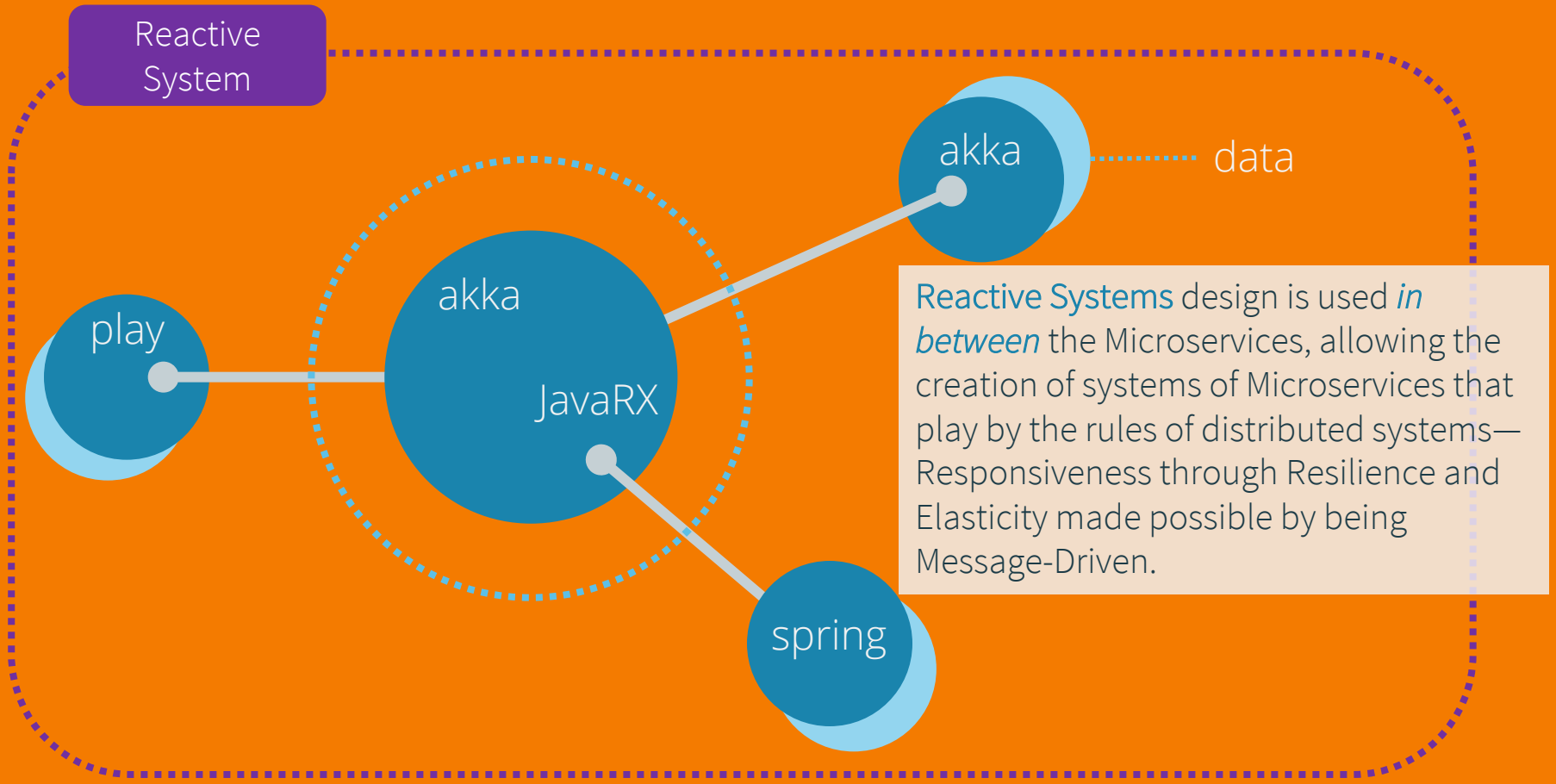


You can do transactions in the boundary of your persistent entity


Order Context
Manages Saga
Shopping Context
Checkout Context
Payments Context
Fulfillment Context

Lagom uses asynchronous messaging, asynchronous IO & distributed persistence patterns


client
command
command
node
Reactive programming
REST
messaging
streaming
Stateless Behavior
node
Reactive programming
Stateful Entity
command
command
command
Service discovery
Service gateway
Authentication
...
Event (fact)
Event (fact)
Write model event log
DS
Event (fact)
Event (fact)
Read model query db
DS
Reactive System



Reactive Systems design is used *in between* the Microservices, allowing the creation of systems of Microservices that play by the rules of distributed systems—Responsiveness through Resilience and Elasticity made possible by being Message-Driven.

Summary

Data can hurt relationships if you let it.

Build trust by managing data with specialized stewards.

Empower your entities to keep their promises.

Managing state requires care, awareness.

Distributed systems require a level up on the care.

Summary

https://github.com/kikiya/wallet-exercise