mojaloop foundation

PI-12 Scalability PoC update

January 2021

Miguel de Barros - Modusbox

Pedro Barreto - Crosslake

Technical Objectives

Re-designed transfers and participants interactions using a distributed transaction

Built a distributed event-based transaction system based on the single responsibility principle - split transfers and participants

Implemented Event-Sourcing and CQRS

Milestones

1st - Prove scalability according to guiding principles

2nd - Introduce Event-Sourcing and CQRS

3rd - Report out document

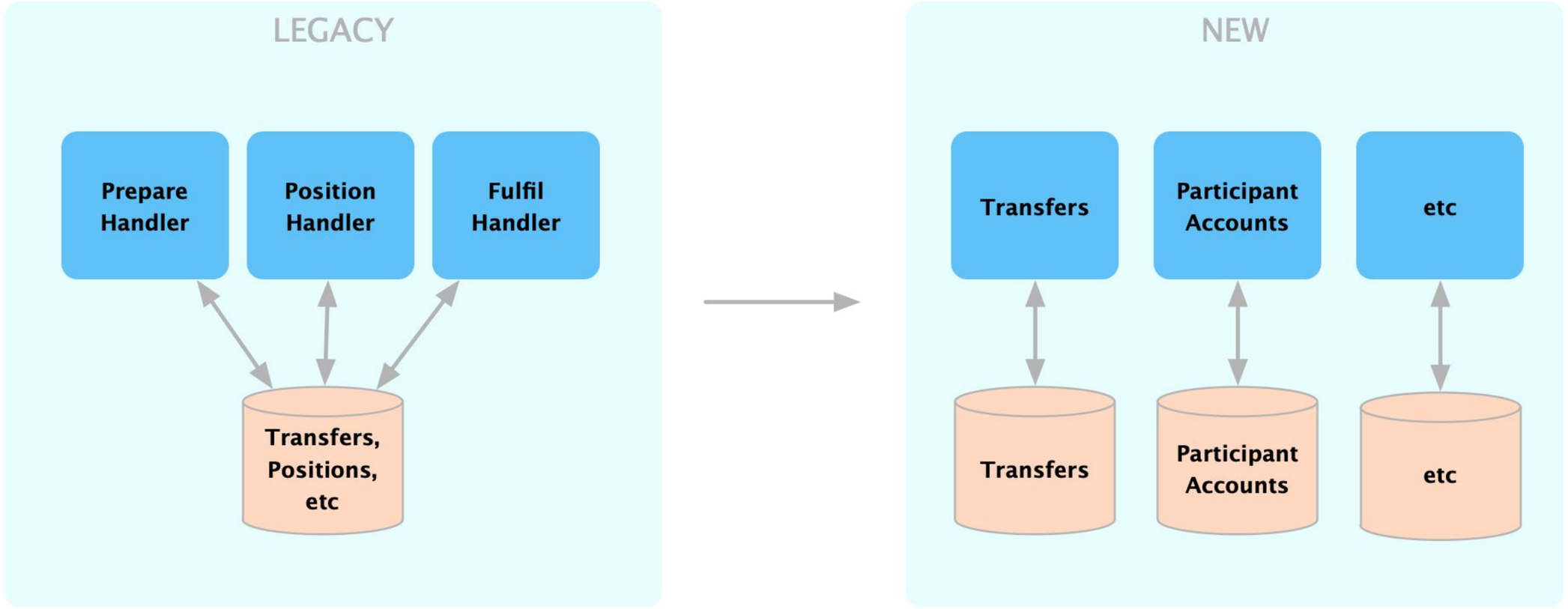
4th - Production ready

How it works

?

Single responsibility

Complete separation of logic and data



mojaloop foundation

Event Driven

“It’s really become clear to me in the last couple of years that we need a new building block, and that is the Domain Events”

<<Eric Evans, 2009>>

Event Driven

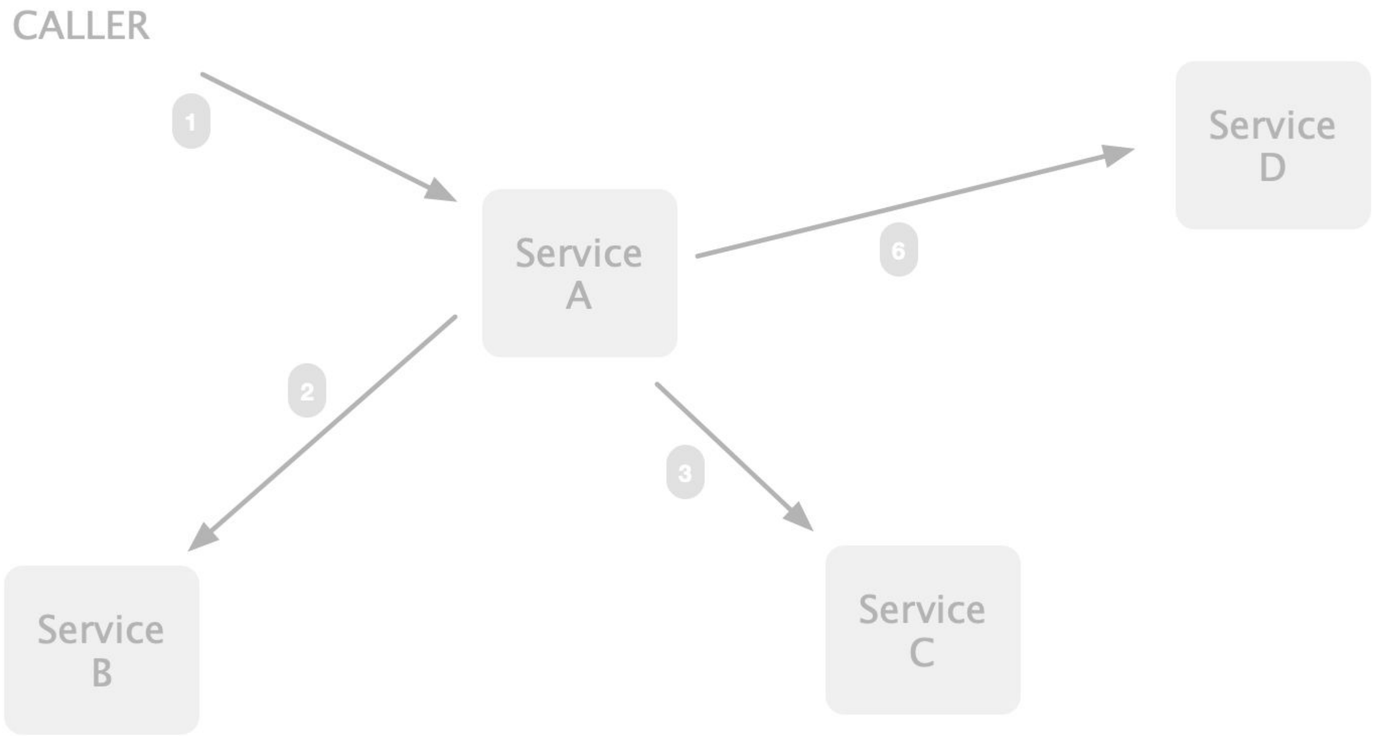
Orchestration

vs

Choreography

Event Driven

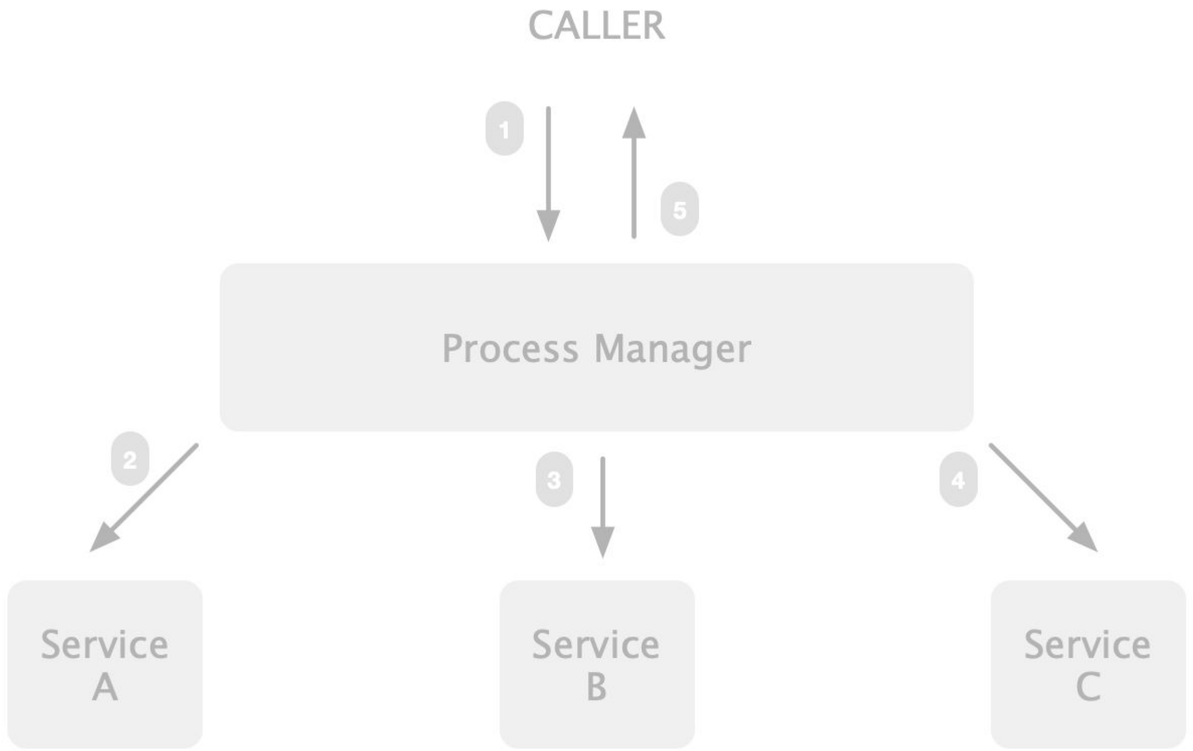
Orchestration Flavour 1 - God service (sync or async)



mojaloop foundation

Event Driven

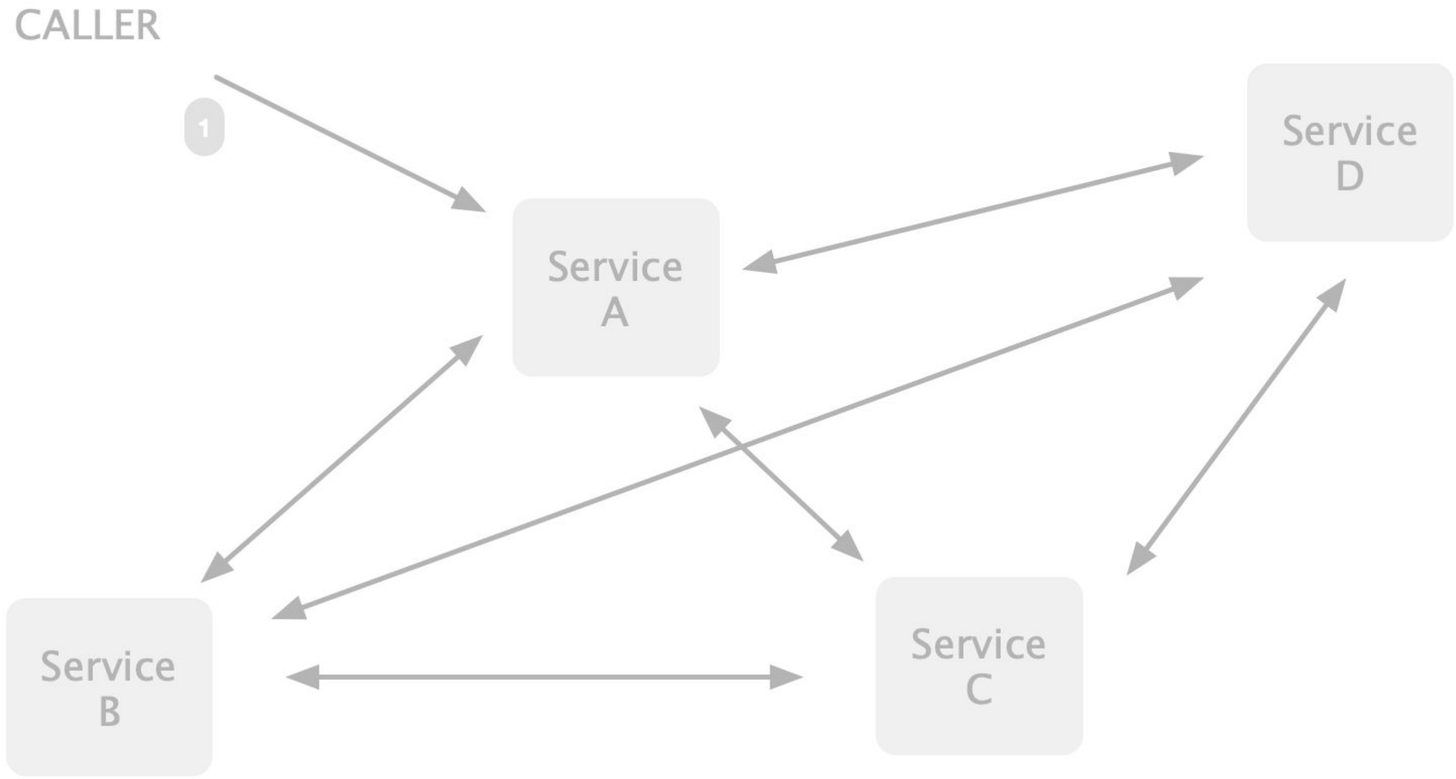
Orchestration Flavour 2 - Process Manager (sync or async)



mojaloop foundation

Event Driven

Orchestration Flavour 3 - Mesh (sync or async)



mojaloop foundation

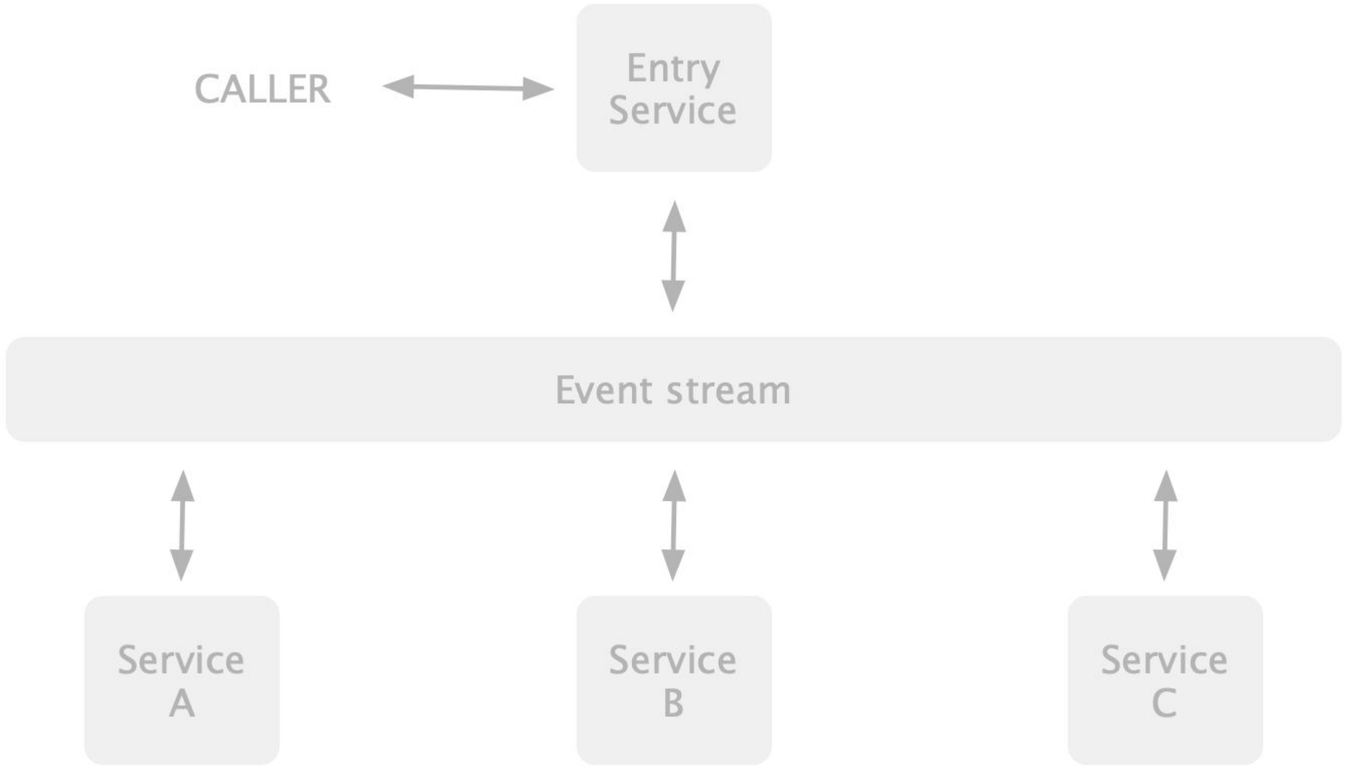
Event Driven

Orchestration Flavour 4 - Enterprise Service Bus

mojaloop foundation

Event Driven

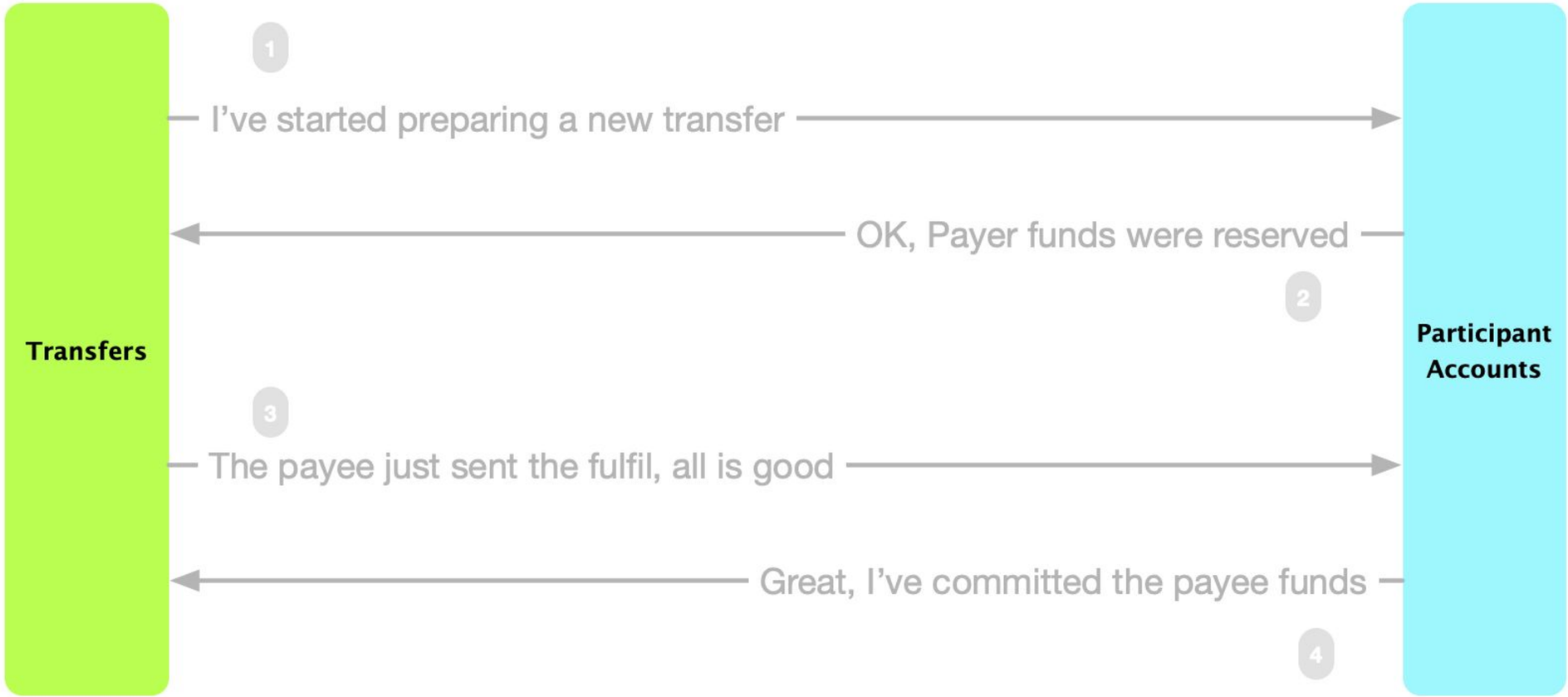
Choreography - With events (The one we chose!)



mojaloop foundation

Event Driven

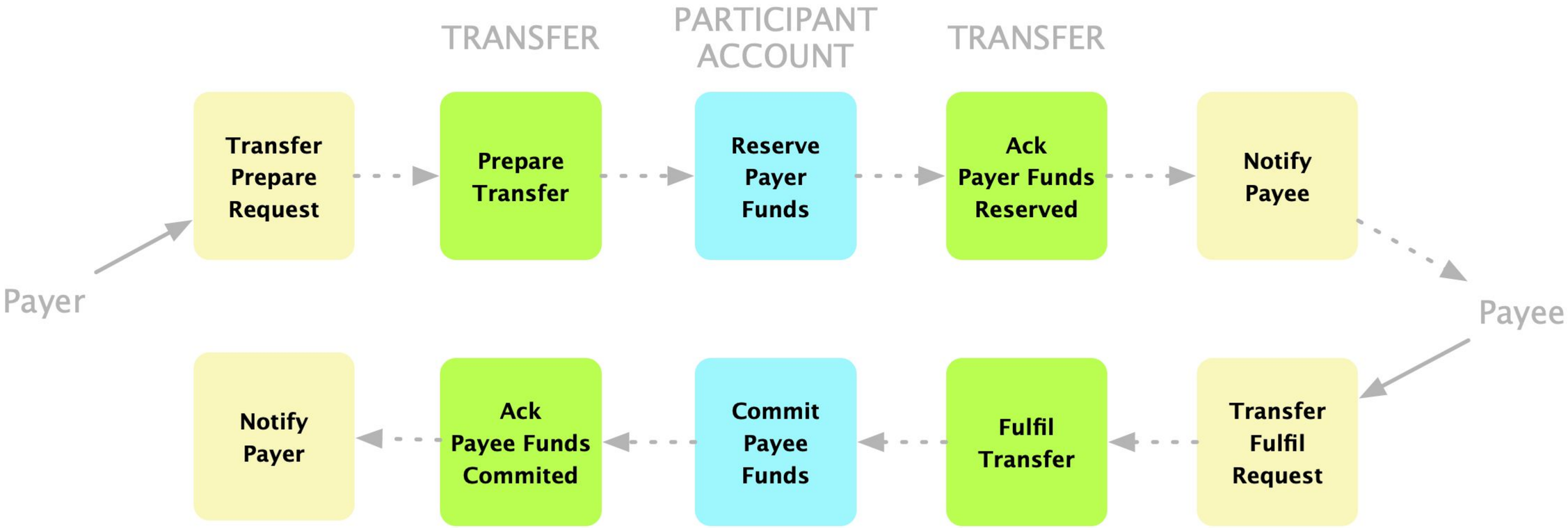
How services talk to each other… they don't!



mojaloop foundation

Event Driven

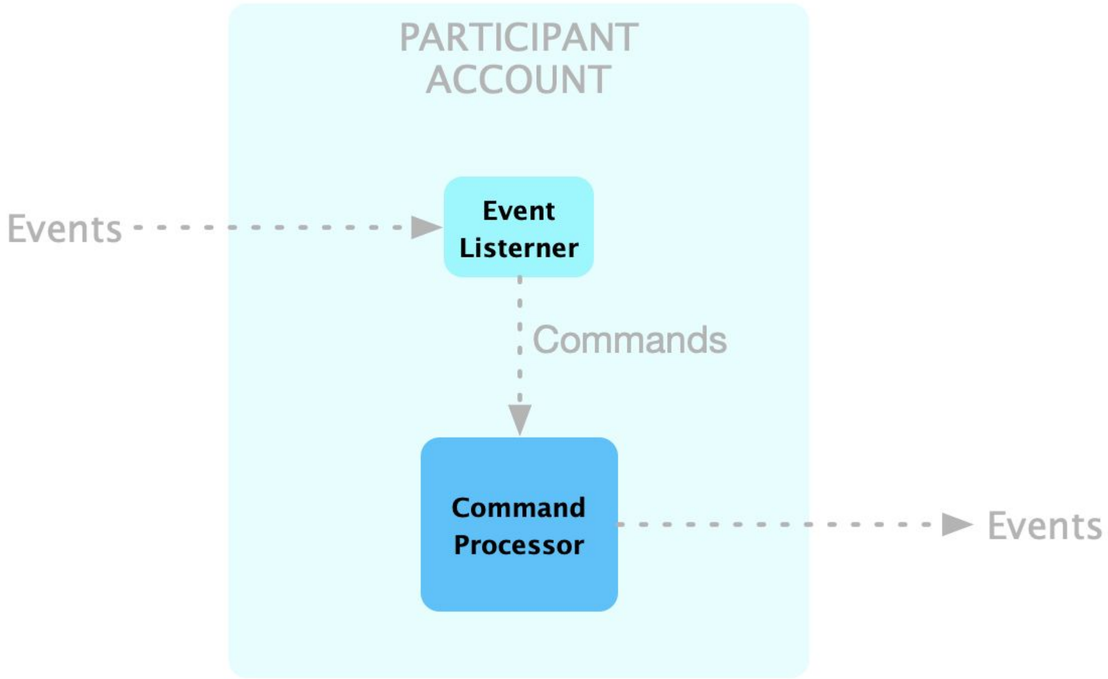
Overview



mojaloop foundation

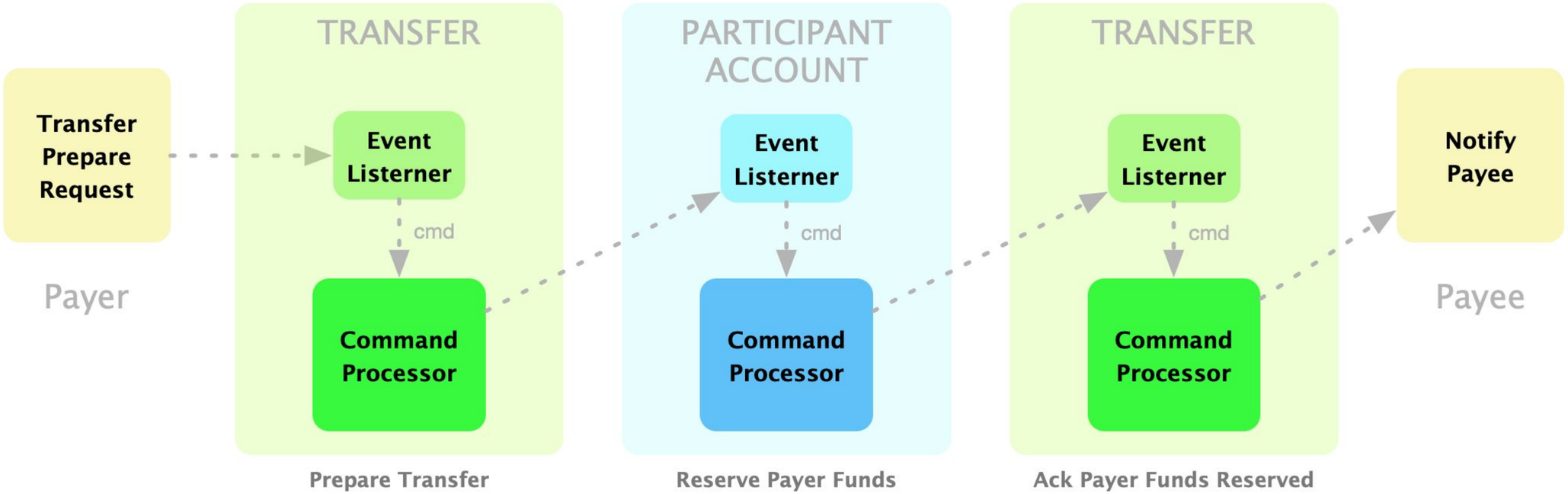
Event Driven

Inside each subsystem



Event Driven

How it all fits together



mojaloop foundation

Event-Sourcing

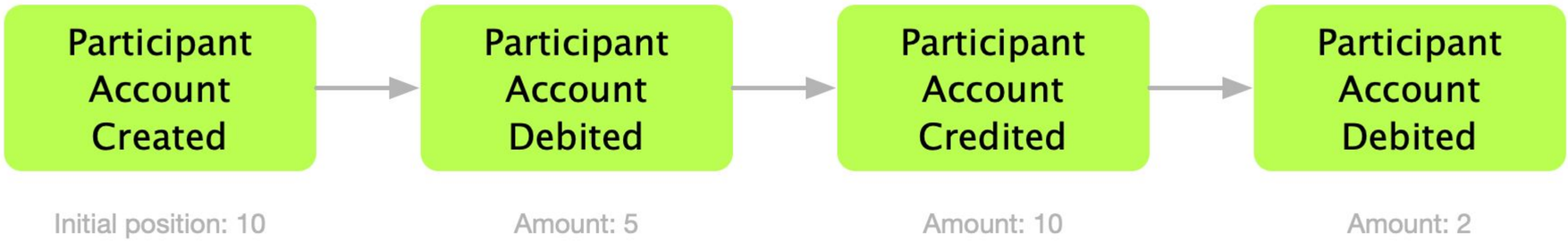
A different way of generating and storing system state

Business logic creates meaningful events

instead of directly changing the state

Event-Sourcing

Life cycle of the Participant Account entity

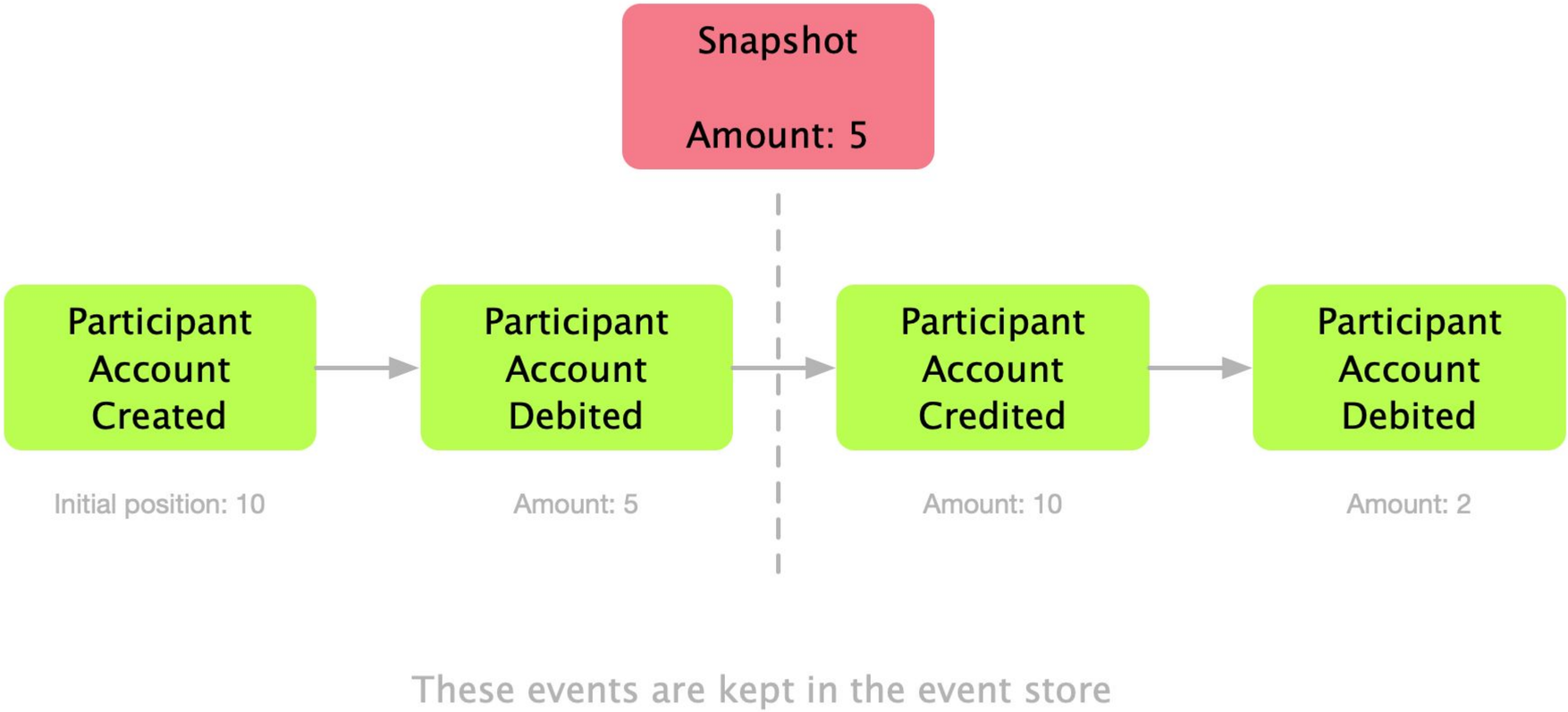


These events are kept in the event store

mojaloop foundation

Event-Sourcing

State loading and snapshotting



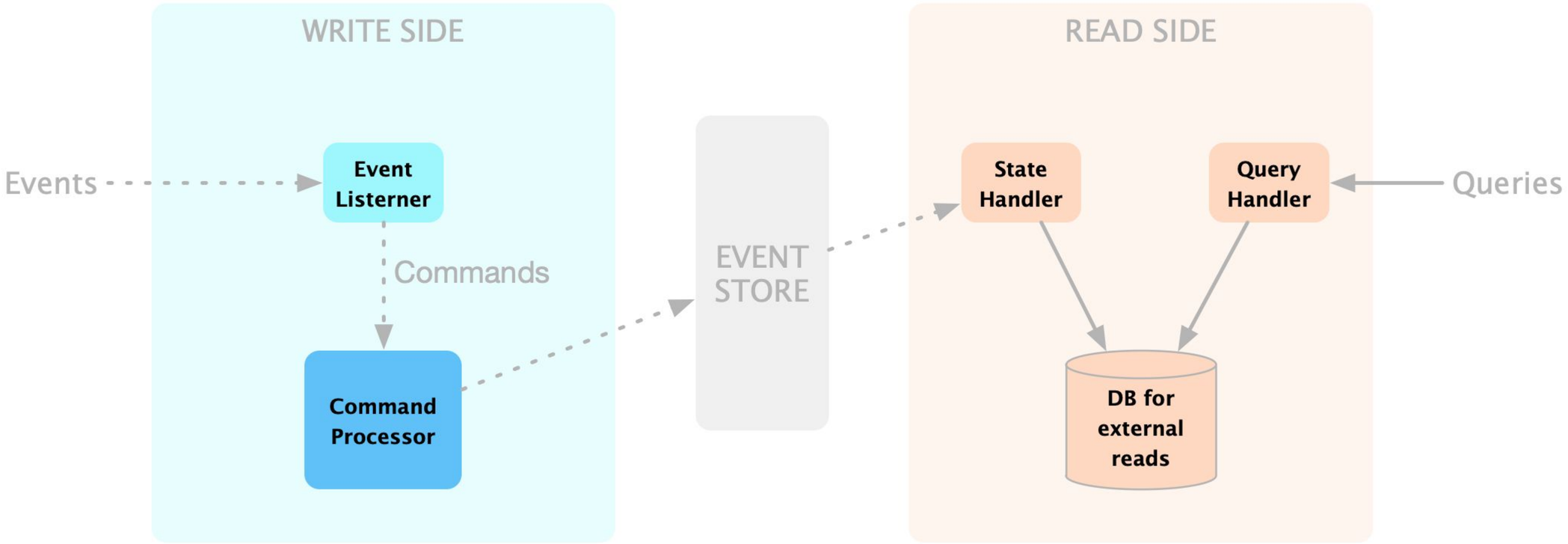
Event-Sourcing

It’s not all roses

How to do complex queries?

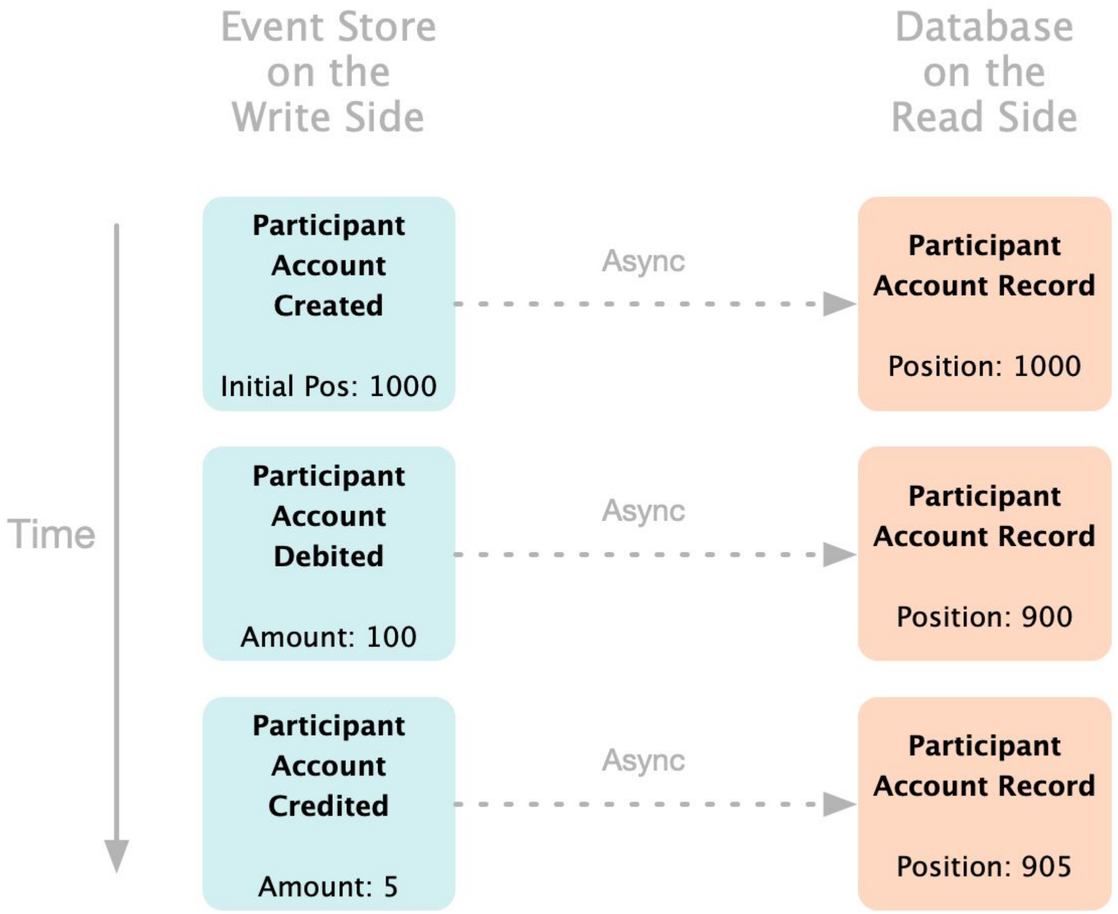
Imagine going through thousands or millions of events?

CQRS to the rescue



mojaloop foundation

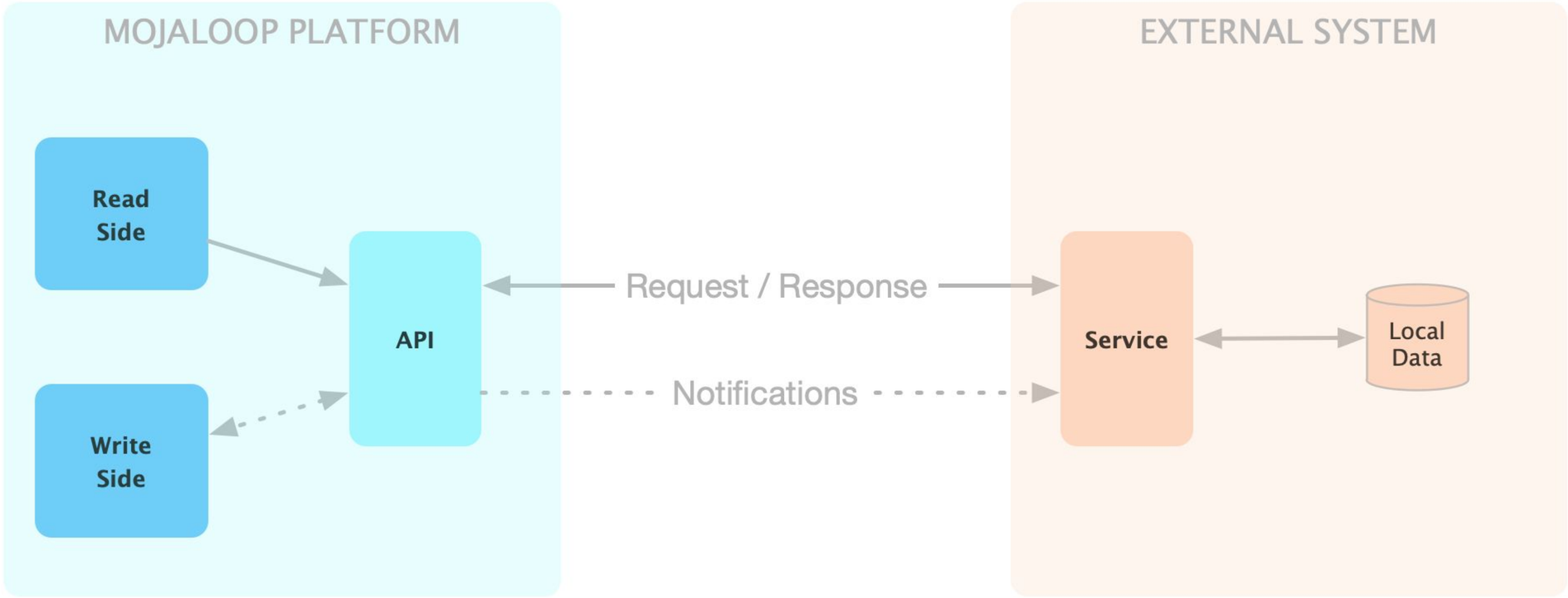
CQRS to the rescue



mojaloop foundation

Integrations

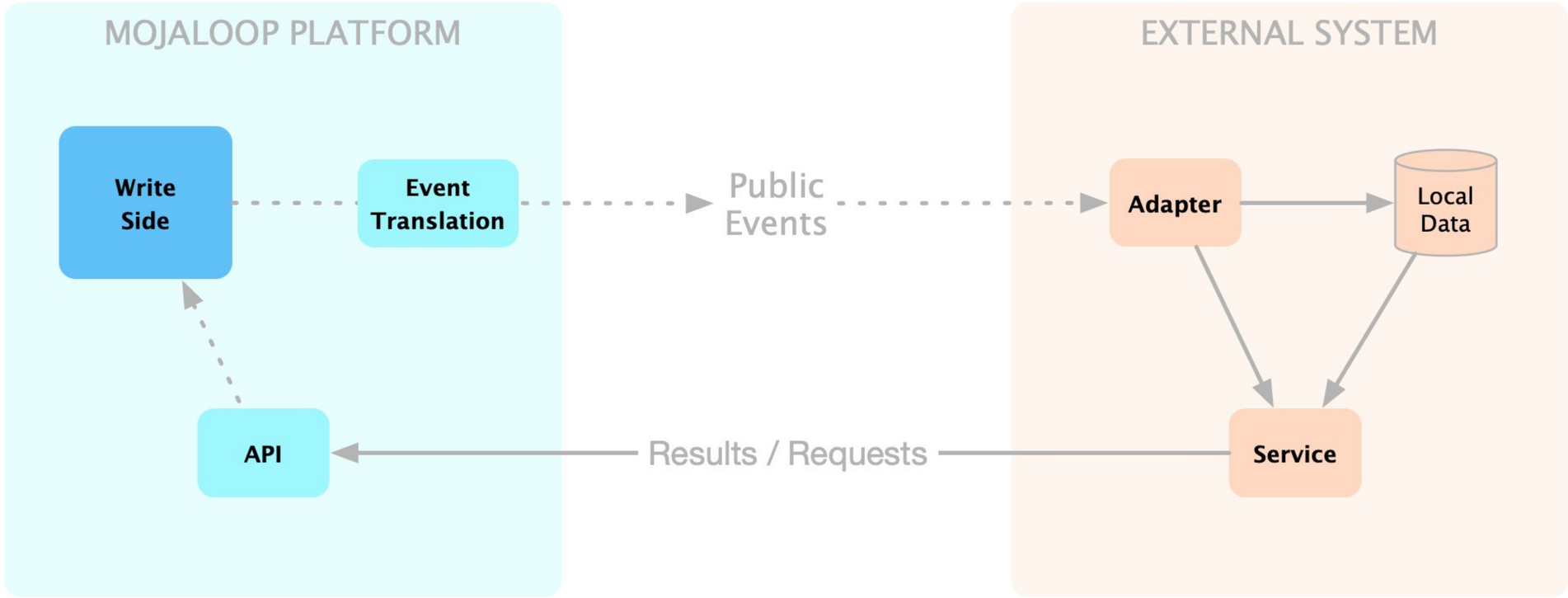
Typical API integration



mojaloop foundation

Integrations

Streaming consumer



Disadvantages of this architecture

Complexity

Initially it is not as intuitive

Advantages of this architecture

Fully encapsulated domain code that only exposes behaviour

Loosely coupled components that can scale and evolve independently

Separation between execution logic and queries, each can be optimised independently utilizing fit-for-purpose and best-of-breed technologies

Highly available

https://community.mojaloop.io/t/performance-scalability-poc-report/187

Where we are since PI-10?

~~Properly test CQRS and Event Sourcing and execute load tests~~

~~Write Read-side CQRS handlers to keep necessary DB Data up to date~~

~~Test scalability of PoC upto 5000 FTPS~~

~~“Performance Scalability PoC Report”~~

Production Ready Feasibility Study

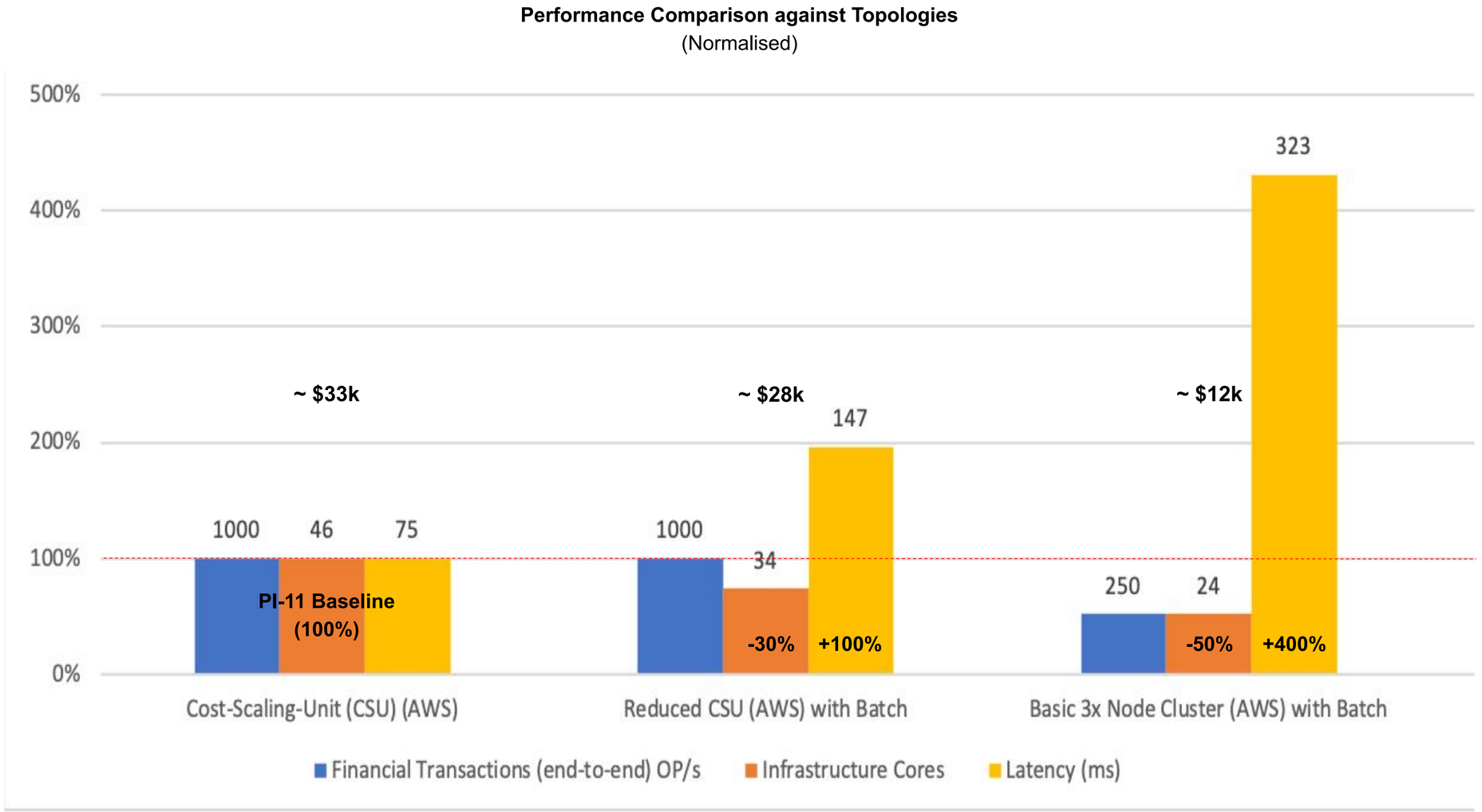
What was done in PI-12

1. Performance Scalability PoC Report was published for community feedback ← https://community.mojaloop.io/t/performance-scalability-poc-report/187
2. Completed Snapshot Handling mechanism for Participants ← Similar to “debouncing”
3. Designed separation of Participant and Account entities to increase the systems scalability. ← Currently we “lock” on Participants, this will enable us to “lock” on a specific account for each currency, and this is also the foundation for the “virtualizing” (or partitioning) of the account position.
4. Optimising the operational costs of running the PoC
   a. Implemented batch processing of messages ← Optimised network and IO usage
   b. Tested batch processing against 3 different topologies: CSU*, Basic 3-Node Cluster, Laptop

mojaloop foundation

Notes:
* Cost Scale Unit (**CSU**): the minimum application and infrastructure configuration required for a self-hosted environment (i.e. infrastructure, scaling, etc.) to achieve 1000 FTP/s - Ref Performance Scalability PoC Report - https://community.mojaloop.io/t/performance-scalability-poc-report/187
* Debouncing - https://en.wiktionary.org/wiki/debounce

PI-12 Results

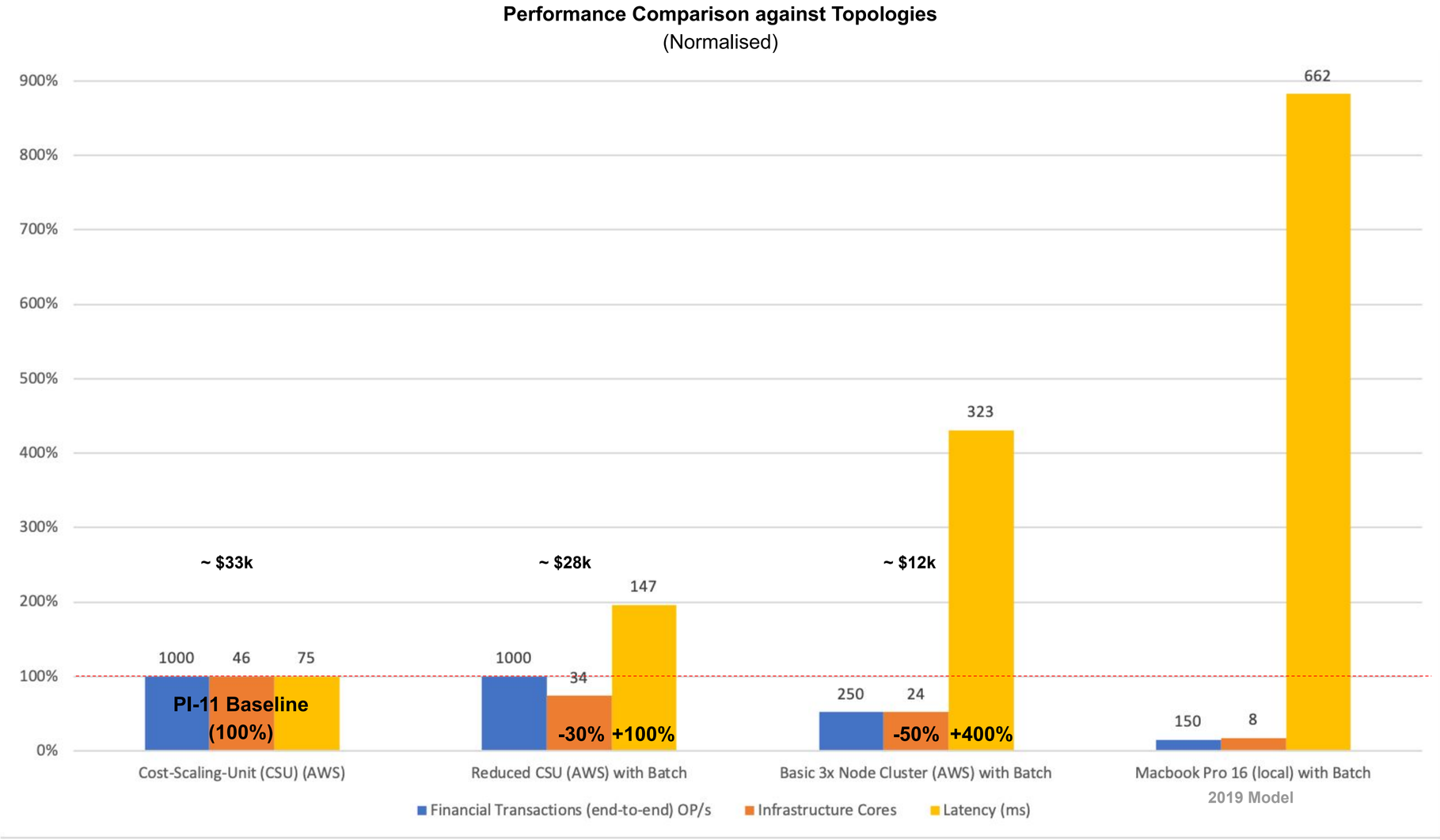


Notes:

* Cost Scale Unit (**CSU**): the minimum application and infrastructure configuration required for a self-hosted environment (i.e. infrastructure, scaling, etc.) to achieve 1000 FTP/s - Ref Performance Scalability PoC Report - https://community.mojaloop.io/t/performance-scalability-poc-report/187
* Estimated $ costs are indicative values for a 12 month period, AWS Cost calculated using On-demand instances in EU-Ireland region - https://calculator.aws/
* Labeled % figures are approximate / rounded

mojaloop foundation

PI-12 Results with Local included



Notes:

* Cost Scale Unit (**CSU**): the minimum application and infrastructure configuration required for a self-hosted environment (i.e. infrastructure, scaling, etc.) to achieve 1000 FTP/s - Ref Performance Scalability PoC Report - https://community.mojaloop.io/t/performance-scalability-poc-report/187
* Estimated $ costs are indicative values for a 12 month period, AWS Cost calculated using On-demand instances in EU-Ireland region - https://calculator.aws/
* Labeled % figures are approximate / rounded

mojaloop foundation

Future enhancements

1. Optimize Batch processing ← Optimize processing throughput ~ initial implementation completed, but there is more improvements that can be made here to take advantage of this processing mode (i.e. see point 2 and 3 below).
2. Separation of Participant Metadata, Account and End-point Information ← Optimize data persistence, & message sizes ~ in-progress
3. Kafka Message Encoding-Parsing ← Reduce encoding/parsing overhead by adopting Protobuf or Avro
4. Virtual positions (sharding the position through different participant handlers) ← Partition FSP position for Parallelism ~ dependency on 2
5. Using a faster language like Java for the Command Handlers ← Can this be considered?
6. Tiger Beetle ← How can we take advantage of this?

PI-12 Conclusions

1. Performance Scalability PoC Report
   a. The proposed architecture and PoC is feasible ← Allot of the work has been done on Happy Paths, effort is needed on unhappy paths and compensating mechanisms.
   b. The PoC can run in parallel with existing Mojaloop Platform components ← e.g. Account Lookup Service, Quoting Service, Central-Settlements, etc
   c. Upgrade and migration of existing systems is possible, but will need support from existing implementers and hub operators ← Refer to report document on how this can be achieved
2. Design is stable, we need to get the rest of the code production ready ← Not enough dedicated resources
3. PoC is performant and scalable across multiple topologies ← Cloud, On-Prem, Laptops

Recap

1. Better horizontal scaling by segregating the different functions across different services and datastores
2. Separate transfers and participants lead to easier partition of datastores
3. Better maintainability / extensibility
4. This design shifts the burden from a shared DB to distributed event streaming, separate read stores and caches - which are horizontally scalable by design
5. Reduced latency and faster response times
6. Retrofitting is possible and a migration/bootstrapping of data is feasible

Summary

The ask was scalability and performance...

... we have, an **abstracted architecture** that is modular and **doesn’t dependent on a single tech**, where **everything can be replaced**, was built for **marketplace extensibility** and can be deployed in multiple of ways, from a laptop to a highly scalable multi geography production cluster

Next Steps

We think this architecture

should be the foundation

for Mojaloop 2.0!

Q&A

Code
https://github.com/mojaloop/poc-architecture

Performance and Scalability PoC Report
https://community.mojaloop.io/t/performance-scalability-poc-report/187

mojaloop
foundation