Serverless Is Interesting But FaaS Is Not Enough

Jonas Bonér
@jboner

Lightbend

Alternative Title

Towards Stateful Serverless

Jonas Bonér
@jbonér

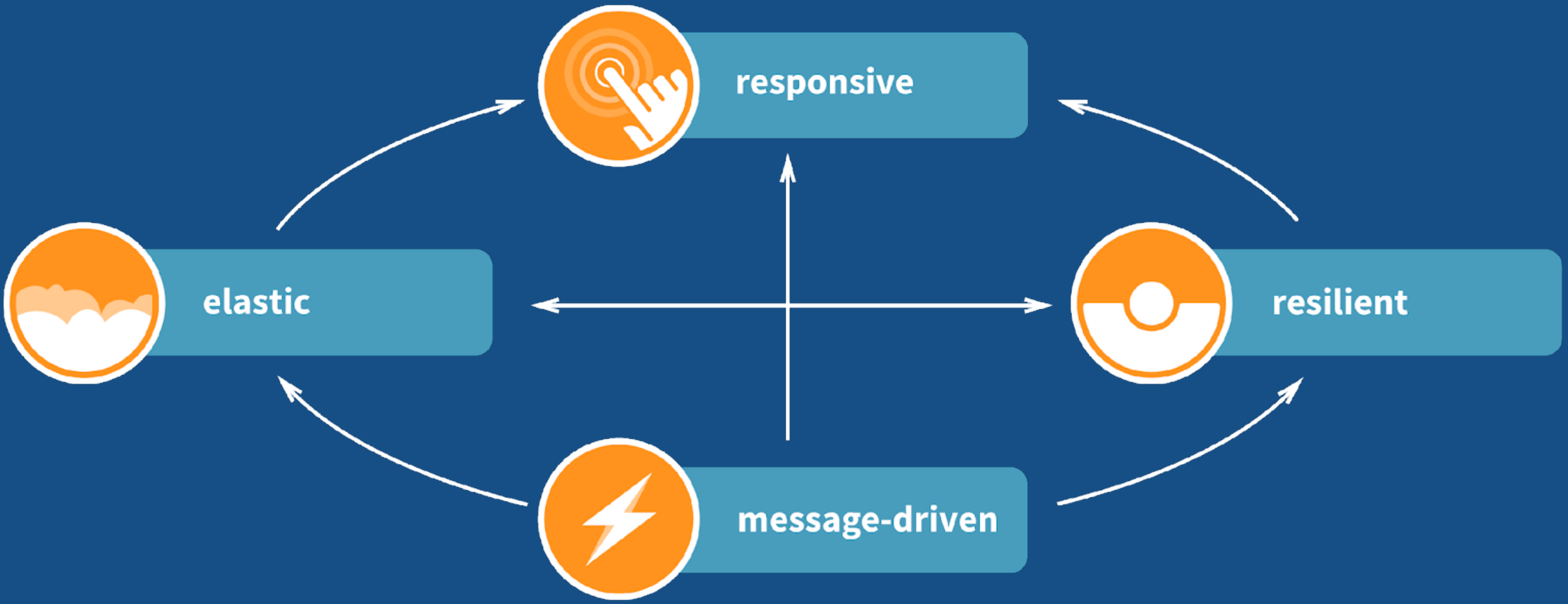
Lightbend

Industry Trends

1. New World: Multicore, Cloud, Mobile, IoT, Big Data, AI
2. Towards Real-time Data-centric Streaming applications
3. Towards a world with automated Operations: Opsless

Reactive Systems

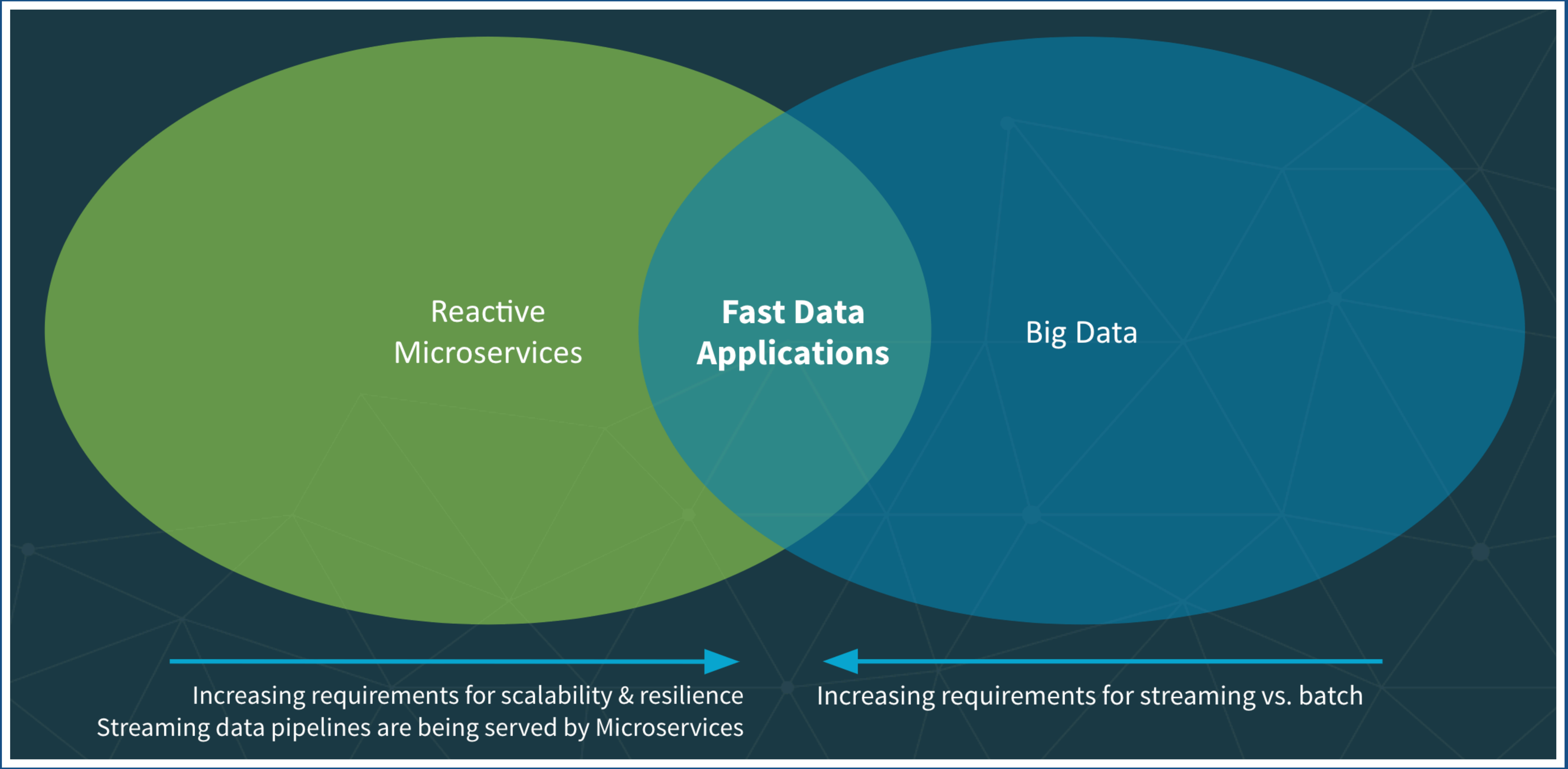
THE RULES OF THE GAME HAVE CHANGED



Reactive Manifesto - reactivemanifesto.org

Towards Fast Data

REAL-TIME, DATA-CENTRIC, EVENT-DRIVEN


Reactive Microservices
Fast Data Applications
Big Data
Increasing requirements for scalability & resilience
Streaming data pipelines are being served by Microservices
Increasing requirements for streaming vs. batch

“We predict that Serverless Computing will grow to dominate the future of Cloud Computing.”

- BERKELEY CS DEPARTMENT

Cloud computing simplified: a Berkeley view on serverless computing

SERVERLESS IS ALL ABOUT THE DEVELOPER EXPERIENCE

SERVERLESS IS ALL ABOUT THE DEVELOPER EXPERIENCE

1. Cost and resource efficient—scale down to zero
2. Pay as you go—scale up on demand
3. Automation—of scale, failure handling, and recovery
4. Supporting the full dev cycle—dev, build, CI, prod

SERVERLESS ≠ FAAS

FaaS = Function-as-a-Service

WHY SHOULD WE LET FAAS HAVE ALL THE FUN?

WHAT'S GOOD WITH FAAS?

WHAT'S GOOD WITH FAAS?

WHAT'S GOOD WITH FAAS?

1. Paved the way for the Serverless Developer Experience

WHAT'S GOOD WITH FAAS?

1. Paved the way for the Serverless Developer Experience
2. Scaling on-demand from 0 to 10000s request, in a cost-efficient manner

WHAT'S GOOD WITH FAAS?

1. Paved the way for the Serverless Developer Experience
2. Scaling on-demand from 0 to 10000s request, in a cost-efficient manner
3. Simplifies delivery of scalable and available applications

WHAT'S GOOD WITH FAAS?

1. Paved the way for the Serverless Developer Experience
2. Scaling on-demand from 0 to 10000s request, in a cost-efficient manner
3. Simplifies delivery of scalable and available applications
4. Encourages good distributed systems design (events-first, loose coupling, etc.)

WHAT'S GOOD WITH FAAS?

1. Paved the way for the Serverless Developer Experience
2. Scaling on-demand from 0 to 10000s request, in a cost-efficient manner
3. Simplifies delivery of scalable and available applications
4. Encourages good distributed systems design (events-first, loose coupling, etc.)
5. Great as integration layer between various (ephemeral and durable) data sources

WHAT'S GOOD WITH FAAS?

1. Paved the way for the Serverless Developer Experience
2. Scaling on-demand from 0 to 10000s request, in a cost-efficient manner
3. Simplifies delivery of scalable and available applications
4. Encourages good distributed systems design (events-first, loose coupling, etc.)
5. Great as integration layer between various (ephemeral and durable) data sources
6. Great for stateless & processing-centric workloads

WHAT'S GOOD WITH FAAS?

1. Paved the way for the Serverless Developer Experience
2. Scaling on-demand from 0 to 10000s request, in a cost-efficient manner
3. Simplifies delivery of scalable and available applications
4. Encourages good distributed systems design (events-first, loose coupling, etc.)
5. Great as integration layer between various (ephemeral and durable) data sources
6. Great for stateless & processing-centric workloads
7. Great as data backbone moving data from A to B, transforming it along the way

USE-CASES FOR FAAS?

USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window

USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window

1. Low traffic applications—enterprise IT services, and spiky workloads

USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window

1. Low traffic applications—enterprise IT services, and spiky workloads
2. Stateless web applications—serving static content form S3 (or similar)

USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window

1. Low traffic applications—enterprise IT services, and spiky workloads
2. Stateless web applications—serving static content form S3 (or similar)
3. Embarrassingly parallel processing tasks—invoked on demand & intermittently, e.g. resizing images, object recognition, log analysis

USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window

1. Low traffic applications—enterprise IT services, and spiky workloads
2. Stateless web applications—serving static content form S3 (or similar)
3. Embarrassingly parallel processing tasks—invoked on demand & intermittently, e.g. resizing images, object recognition, log analysis
4. Orchestration functions—integration/coordination of calls to third-party services

USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window

1. Low traffic applications—enterprise IT services, and spiky workloads
2. Stateless web applications—serving static content form S3 (or similar)
3. Embarrassingly parallel processing tasks—invoked on demand & intermittently, e.g. resizing images, object recognition, log analysis
4. Orchestration functions—integration/coordination of calls to third-party services
5. Composing chains of functions—stateless workflow management, connected via data dependencies

USE-CASES FOR FAAS?

Use-cases where throughput is key rather than low latency and requests can be completed in a short time window

1. Low traffic applications—enterprise IT services, and spiky workloads
2. Stateless web applications—serving static content form S3 (or similar)
3. Embarrassingly parallel processing tasks—invoked on demand & intermittently, e.g. resizing images, object recognition, log analysis
4. Orchestration functions—integration/coordination of calls to third-party services
5. Composing chains of functions—stateless workflow management, connected via data dependencies
6. Job scheduling—CRON jobs, triggers, etc.

WHAT'S BAD WITH FAAS?

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

1. Retains the limitations of the 3-tier architecture

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

1. Retains the limitations of the 3-tier architecture
2. Functions are stateless, ephemeral, and short-lived

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

1. Retains the limitations of the 3-tier architecture
2. Functions are stateless, ephemeral, and short-lived
3. No direct addressability

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

1. Retains the limitations of the 3-tier architecture
2. Functions are stateless, ephemeral, and short-lived
3. No direct addressability
4. No co-location of state and processing

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

1. Retains the limitations of the 3-tier architecture
2. Functions are stateless, ephemeral, and short-lived
3. No direct addressability
4. No co-location of state and processing
5. Limited options for managing and coordinating distributed state

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

1. Retains the limitations of the 3-tier architecture
2. Functions are stateless, ephemeral, and short-lived
3. No direct addressability
4. No co-location of state and processing
5. Limited options for managing and coordinating distributed state
6. Limited options for modelling various consistency guarantees

WHAT'S BAD WITH FAAS?

Hard to build general-purpose applications

1. Retains the limitations of the 3-tier architecture
2. Functions are stateless, ephemeral, and short-lived
3. No direct addressability
4. No co-location of state and processing
5. Limited options for managing and coordinating distributed state
6. Limited options for modelling various consistency guarantees
7. Limited options for managing durable state, that is scalable and available

What We Want

- The Serverless DX—but for general-purpose applications, including modern Fast Data and Reactive systems
- Stateful functions—complementing stateless functions, expanding the toolbox and supported use-cases
- The cost efficiencies of FaaS—while allowing the user to dial in trade-offs (related to cost, SLOs, use-cases)

Support For Use Cases Like

- Training and Serving of Machine Learning Models
  - Any dynamic in-memory model that needs to build up and served with low latency
- Real-time Distributed Stream Processing
  - E.g. Real-time Prediction/Recommendation Serving, Anomaly Detection
- User Sessions, Shopping Carts, Caching
  - Managing in-memory, yet durable, session state across individual requests
- Transaction Management
  - Saga Pattern, Workflow Orchestration, Rollback/Compensating Actions
- Shared Collaborative Workspaces
  - Collaborative Document Editing, Blackboards, Chat Rooms, etc.
- Leader Election
  - . . . and other standard distributed systems patterns/protocols for coordination

Technical Requirements

Technical Requirements

1. Stateful long-lived addressable virtual components (actors)

Technical Requirements

1. Stateful long-lived addressable virtual components (actors)
2. Wide range of options for distributed coordination & communication patterns

Technical Requirements

1. Stateful long-lived addressable virtual components (actors)
2. Wide range of options for distributed coordination & communication patterns
3. Options for managing distributed state reliably at scale, ranging from strong to eventual consistency (durable/ephemeral)

Technical Requirements

1. Stateful long-lived addressable virtual components (actors)
2. Wide range of options for distributed coordination & communication patterns
3. Options for managing distributed state reliably at scale, ranging from strong to eventual consistency (durable/ephemeral)
4. Intelligent adaptive placement of stateful functions (co-location)

Technical Requirements

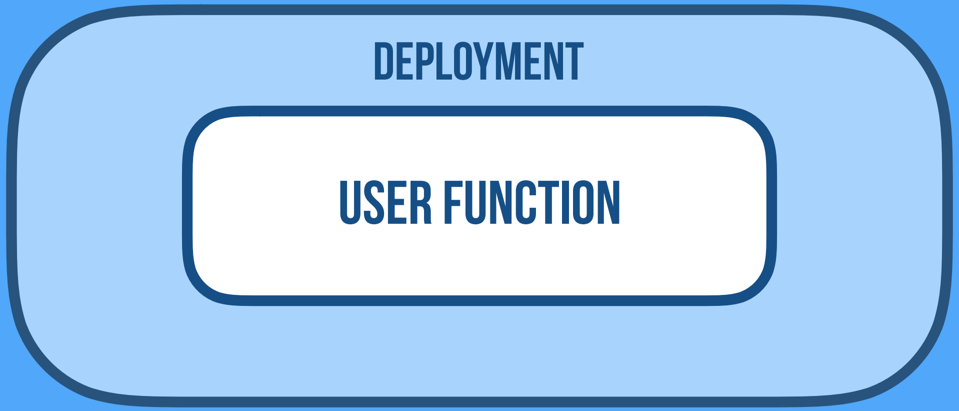
1. Stateful long-lived addressable virtual components (actors)
2. Wide range of options for distributed coordination & communication patterns
3. Options for managing distributed state reliably at scale, ranging from strong to eventual consistency (durable/ephemeral)
4. Intelligent adaptive placement of stateful functions (co-location)
5. Ways of managing end-to-end guarantees and correctness

Technical Requirements

1. Stateful long-lived addressable virtual components (actors)
2. Wide range of options for distributed coordination & communication patterns
3. Options for managing distributed state reliably at scale, ranging from strong to eventual consistency (durable/ephemeral)
4. Intelligent adaptive placement of stateful functions (co-location)
5. Ways of managing end-to-end guarantees and correctness
6. Predictable performance, latency, and throughput—in startup time, communication/coordination, and storage of data

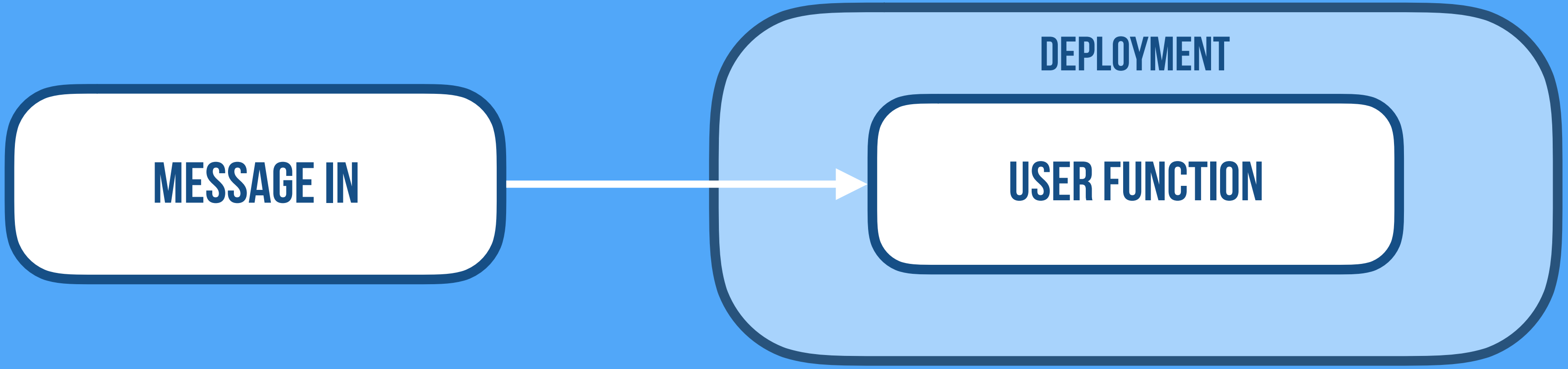
FaaS

Serverless 1.0



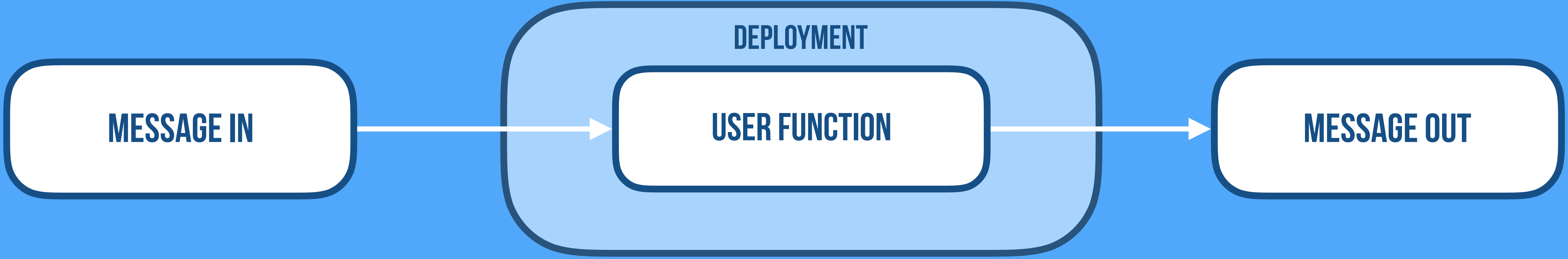
FaaS

Serverless 1.0

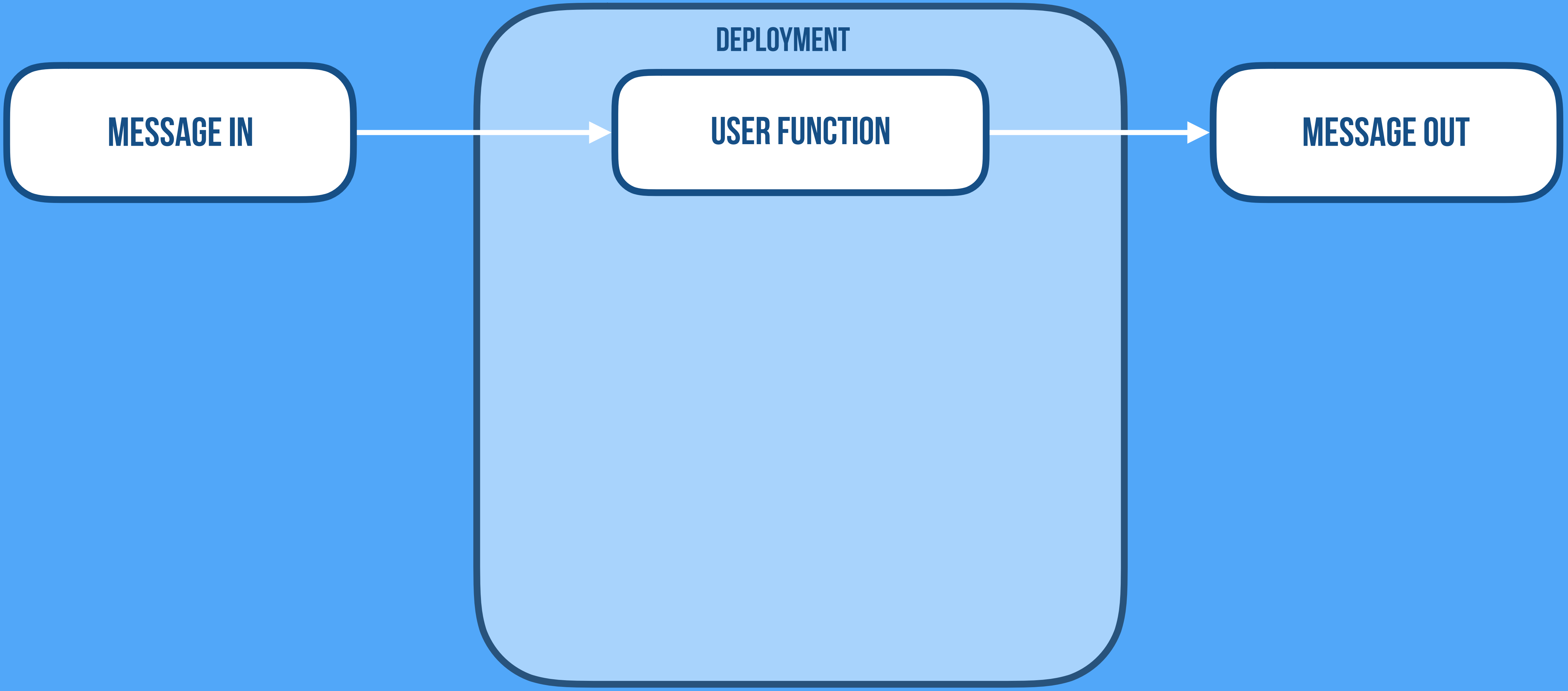


FaaS

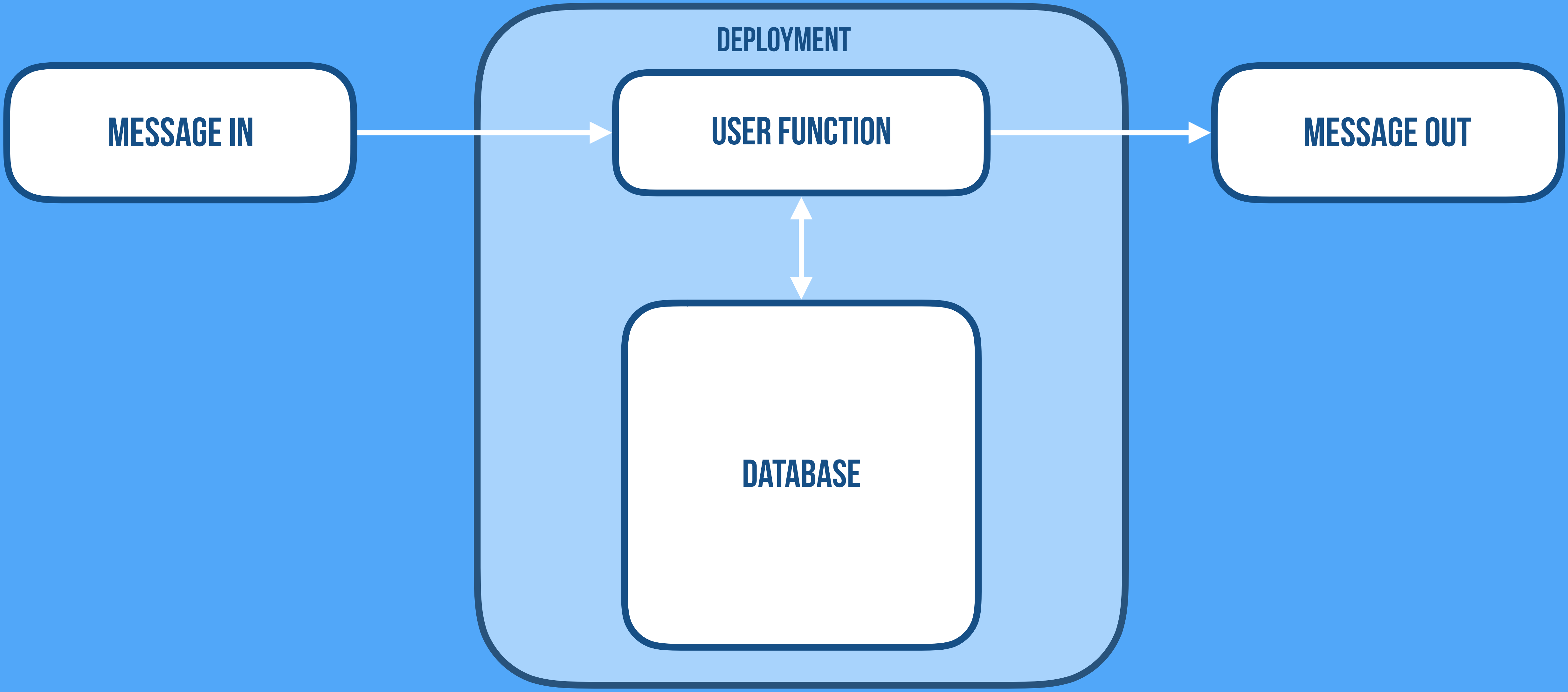
Serverless 1.0



FaaS With CRUD

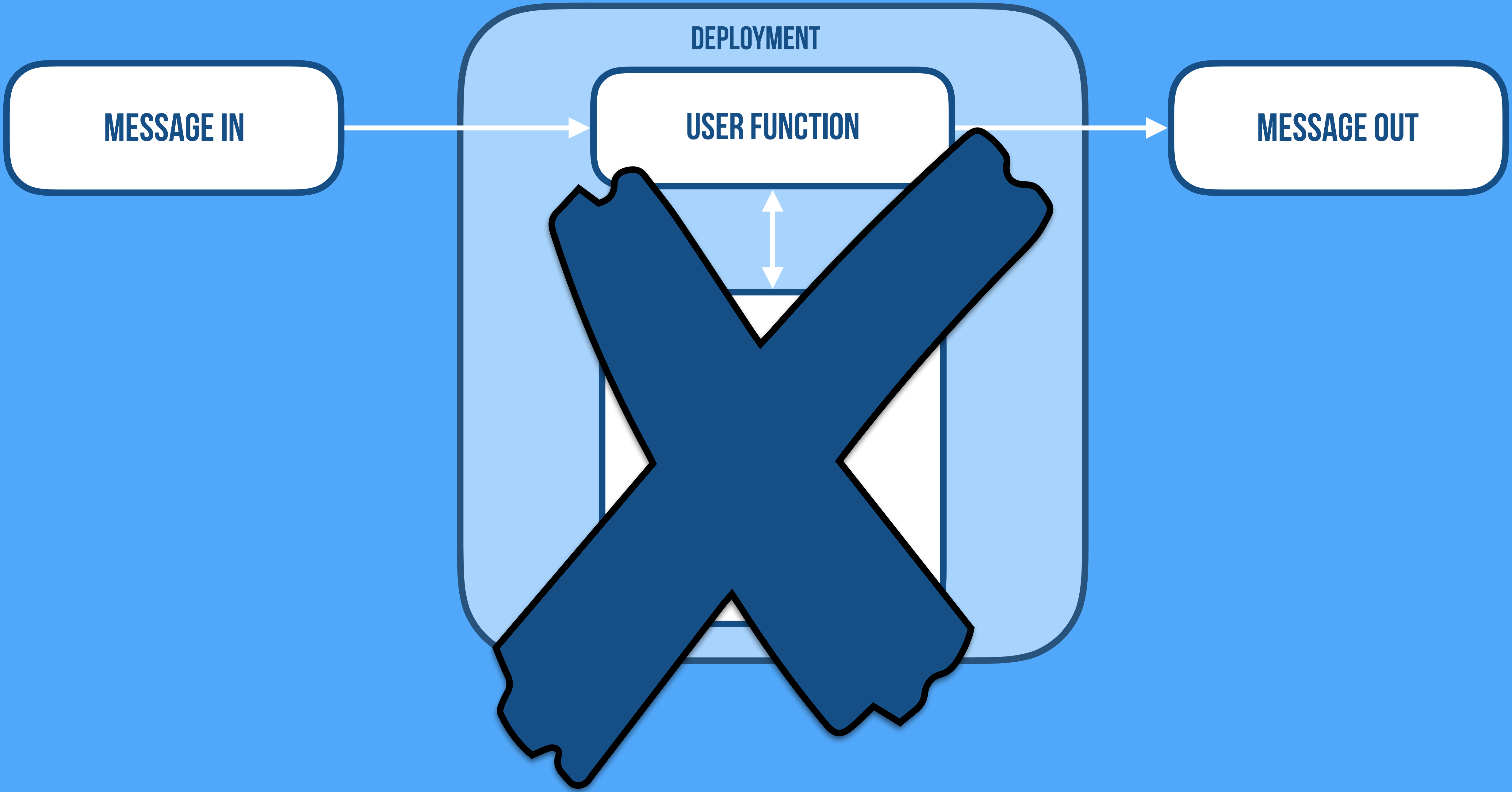


FaaS With CRUD



Not Serverless

In An Ideal World

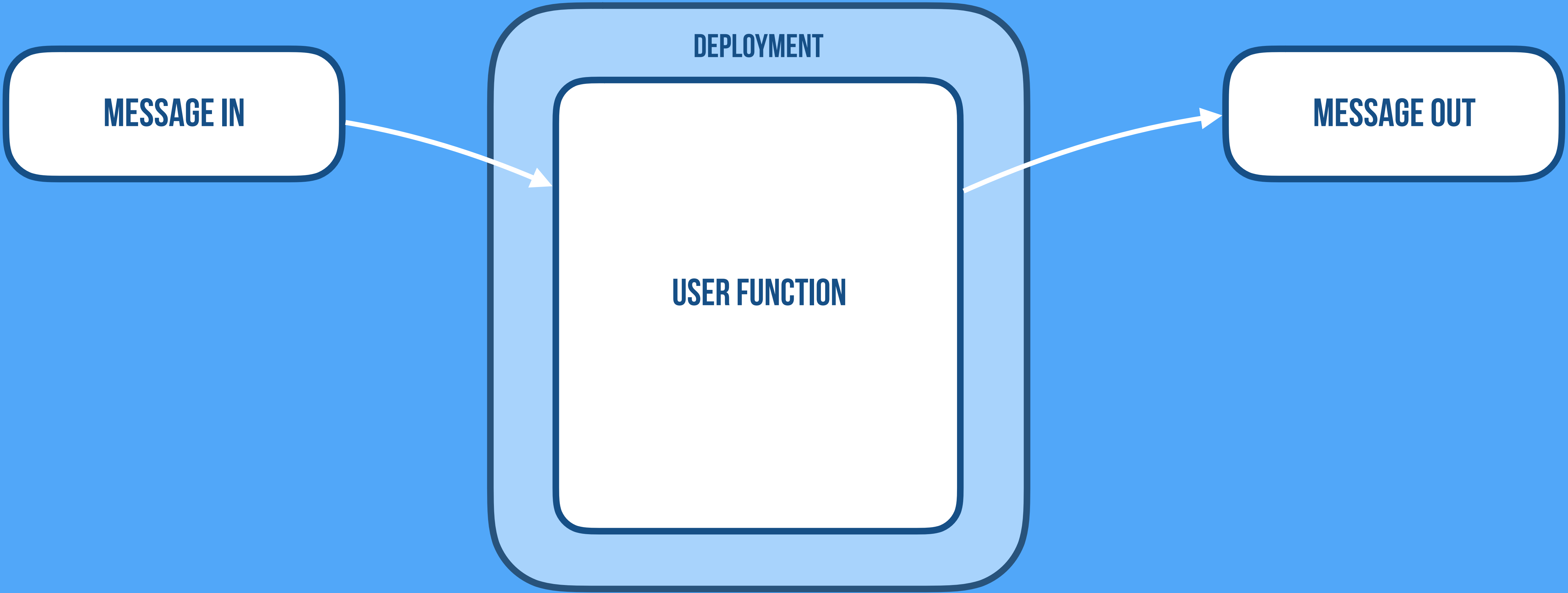


UNCONSTRAINED DATABASE ACCESS MAKES IT HARD TO AUTOMATE OPERATIONS

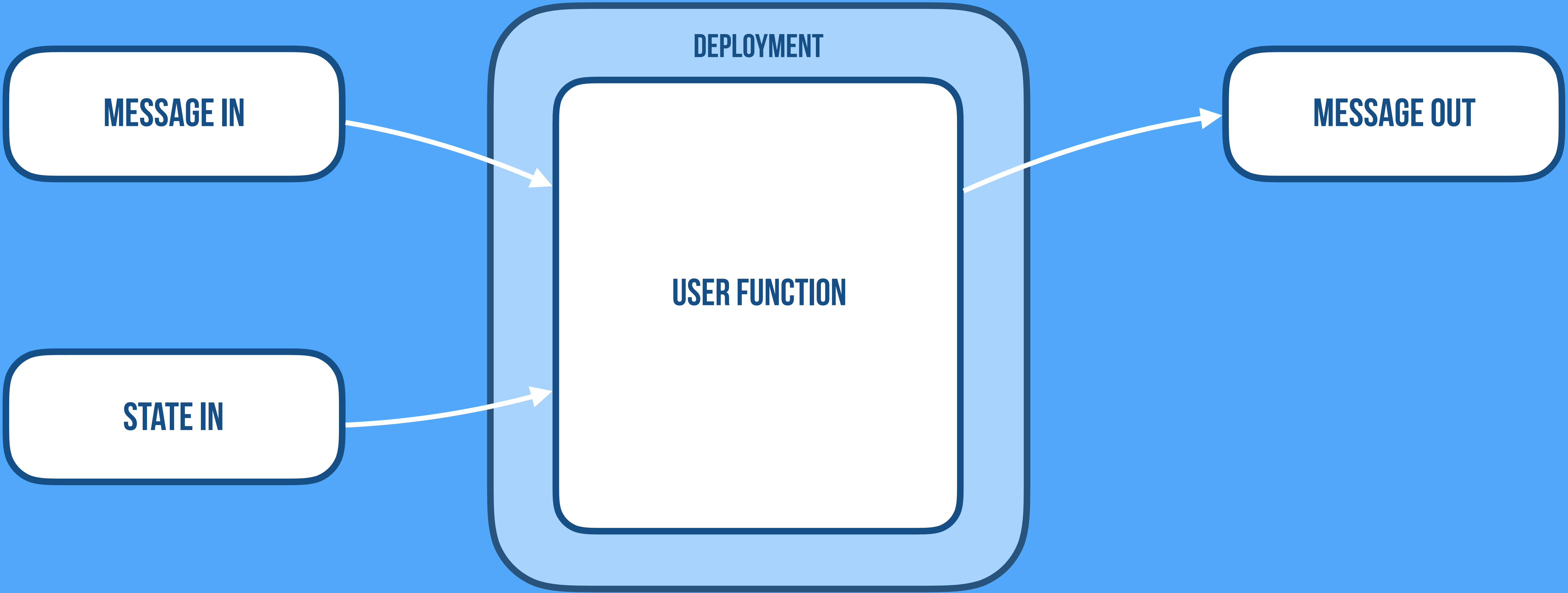
“Constraints liberate, liberties constrain.”

- RUNAR BJARNASON

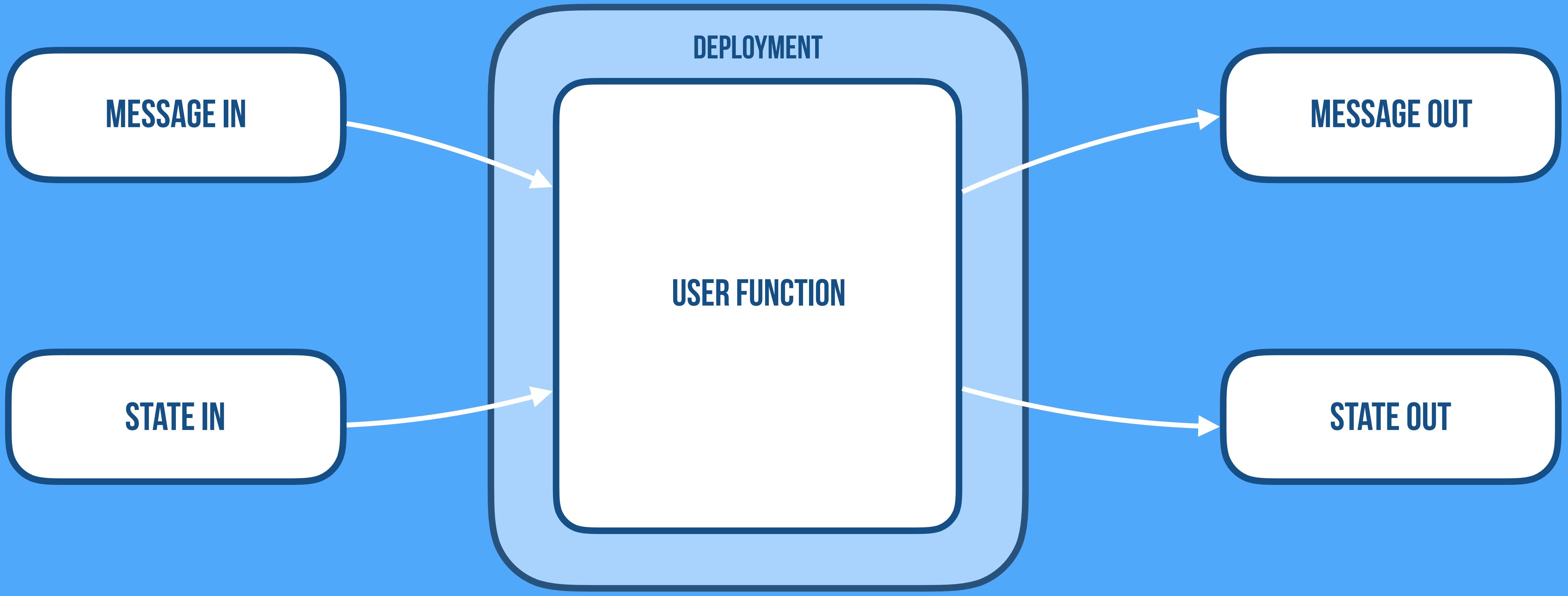
Stateful Serverless



Stateful Serverless



Stateful Serverless



We Need Better Models For Distributed State

We Need Better Models For Distributed State

A COUPLE OF BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

We Need Better Models For Distributed State

A COUPLE OF BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Event Sourcing & **CRDTs**

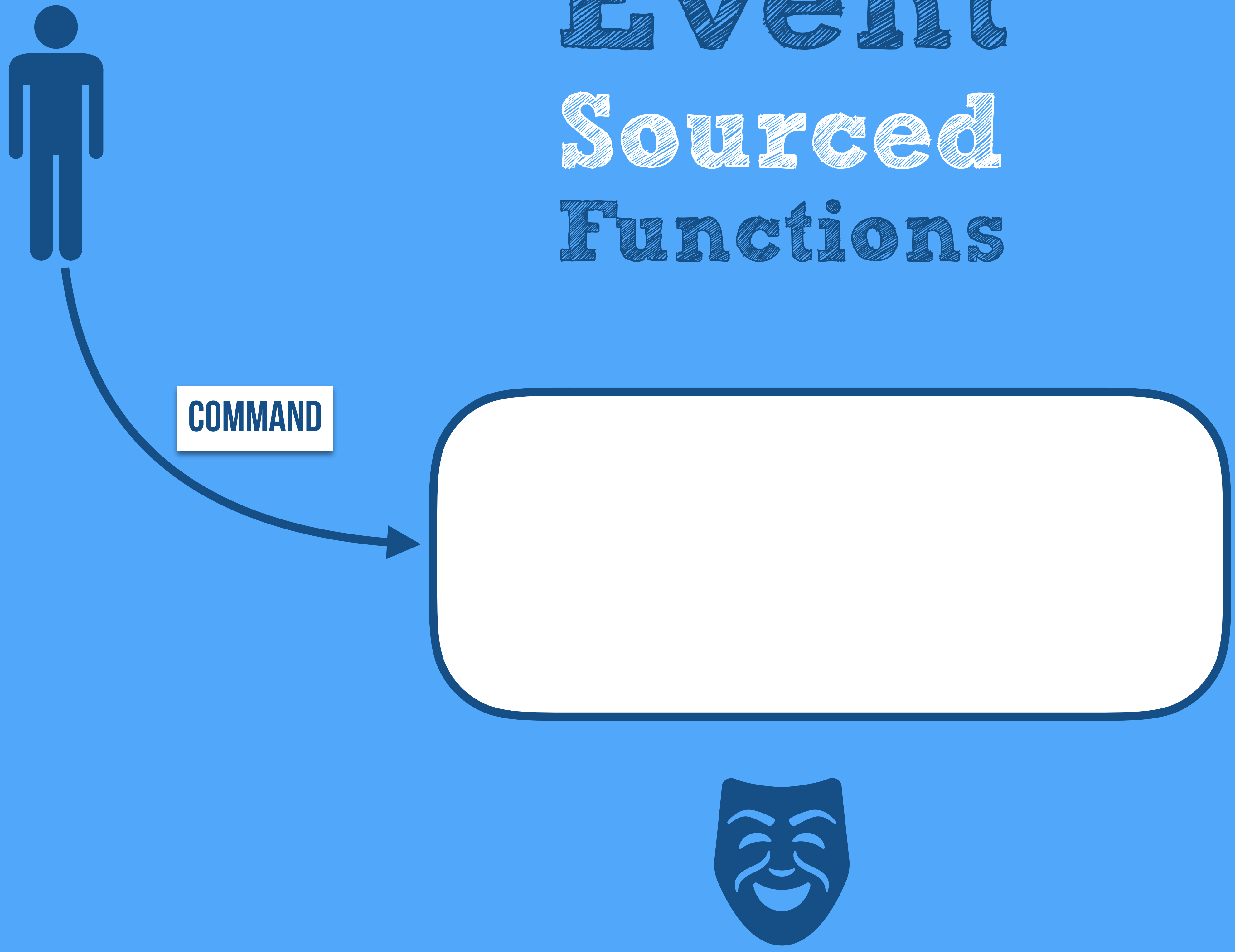
Event Sourced Functions

HAPPY PATH

Event Sourced Functions

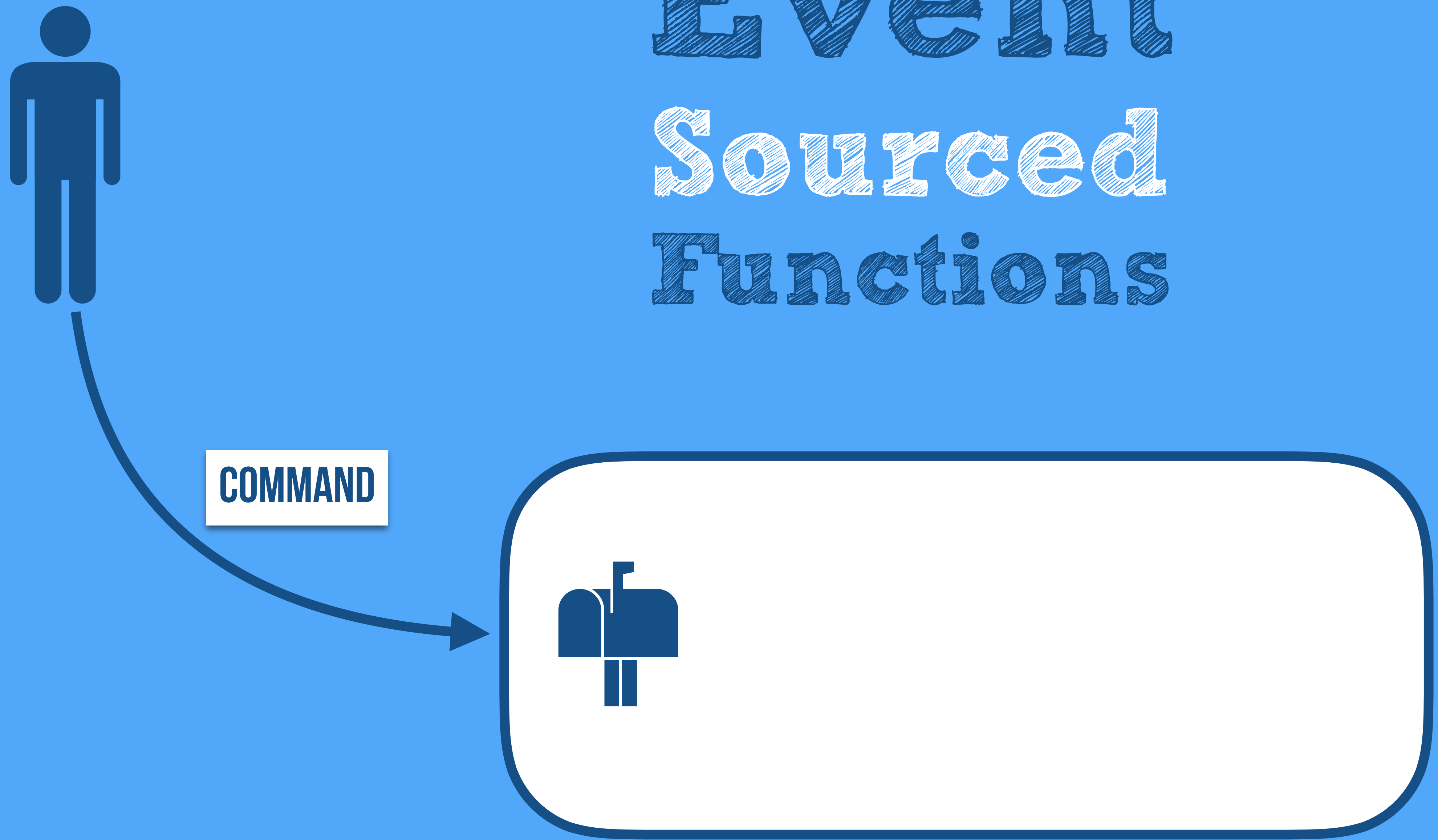
HAPPY PATH

Event Sourced Functions



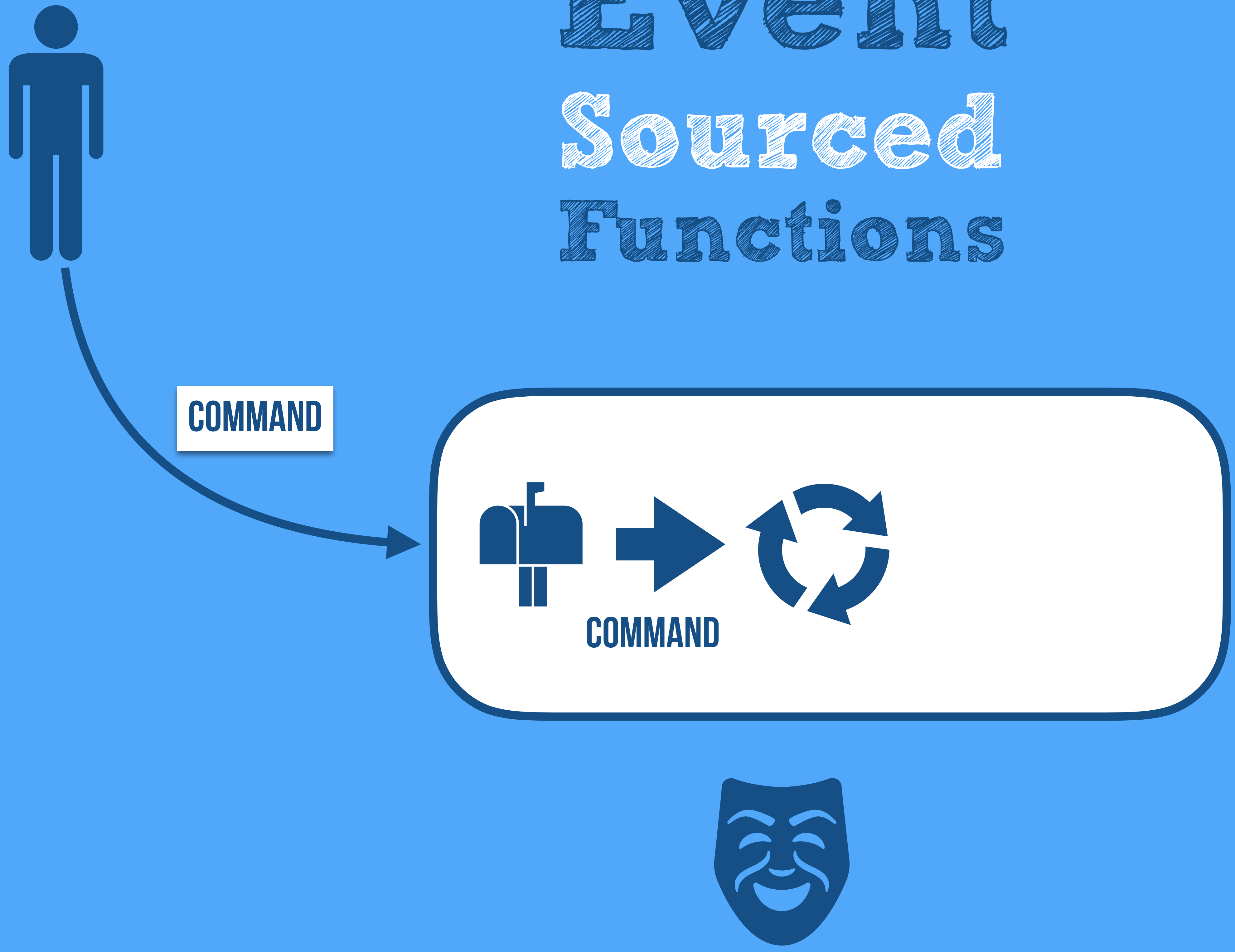
HAPPY PATH

Event Sourced Functions



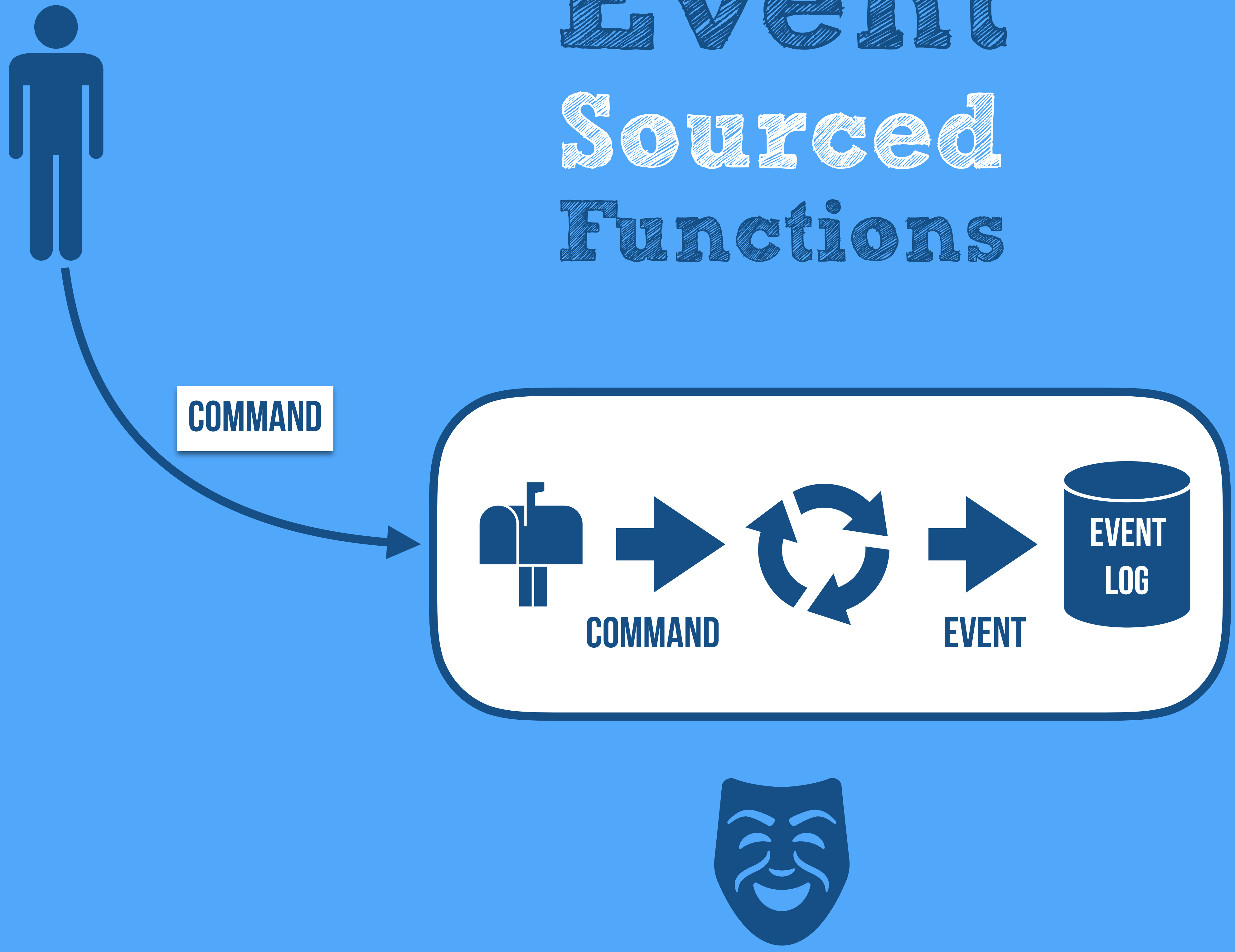
HAPPY PATH

Event Sourced Functions



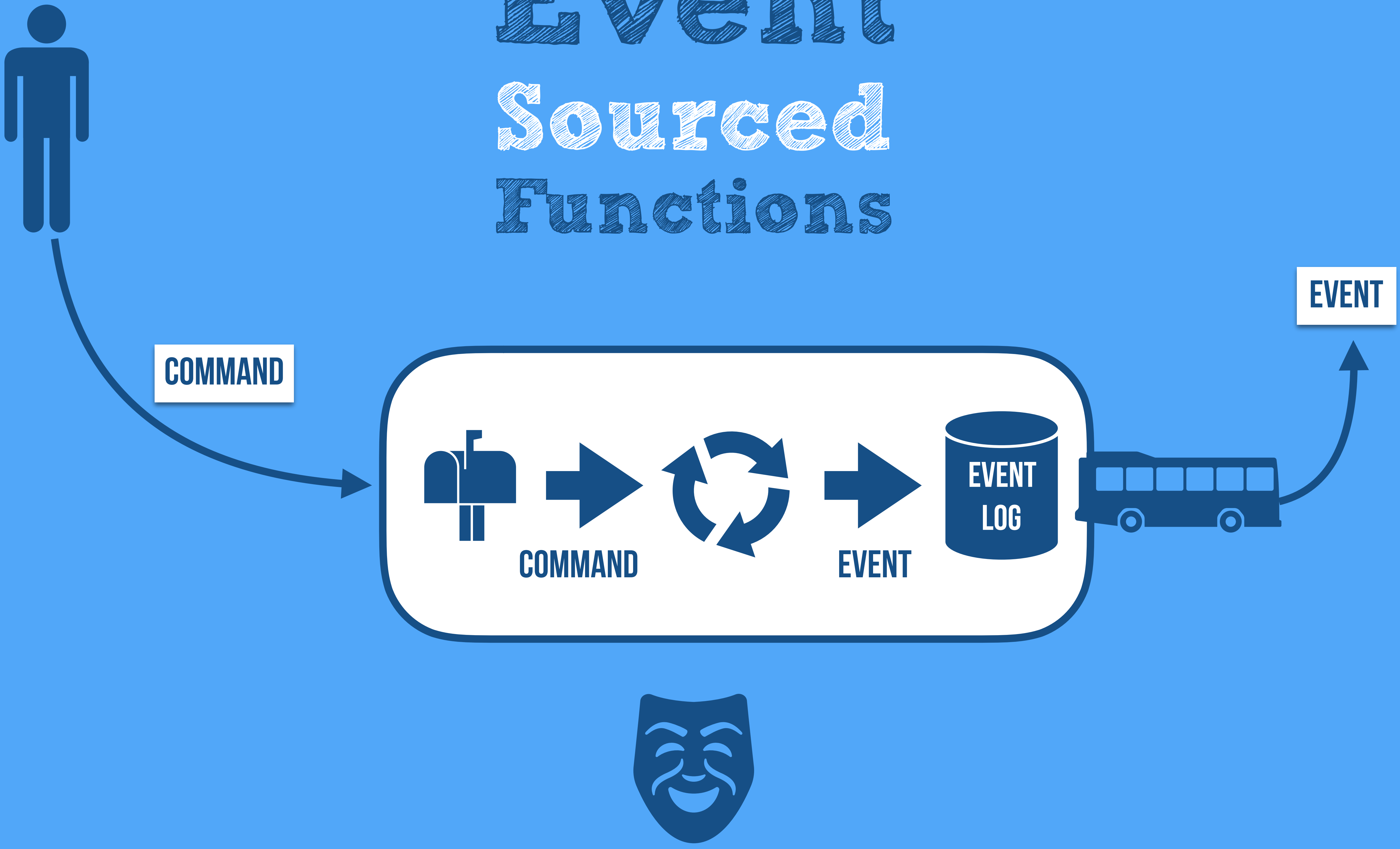
HAPPY PATH

Event Sourced Functions



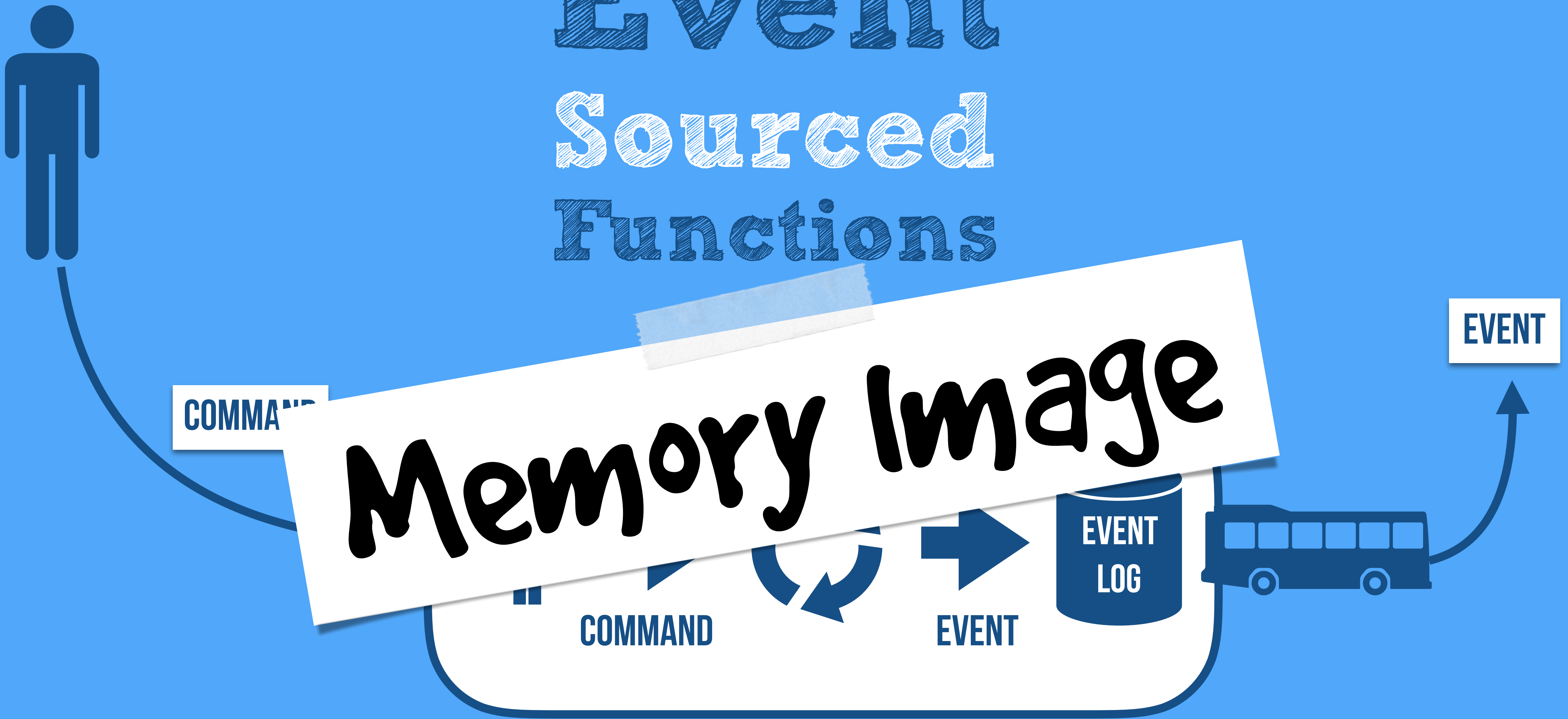
HAPPY PATH

Event Sourced Functions



HAPPY PATH

Event Sourced Functions



HAPPY PATH

Event

Sourced

Functions

HAPPY PATH

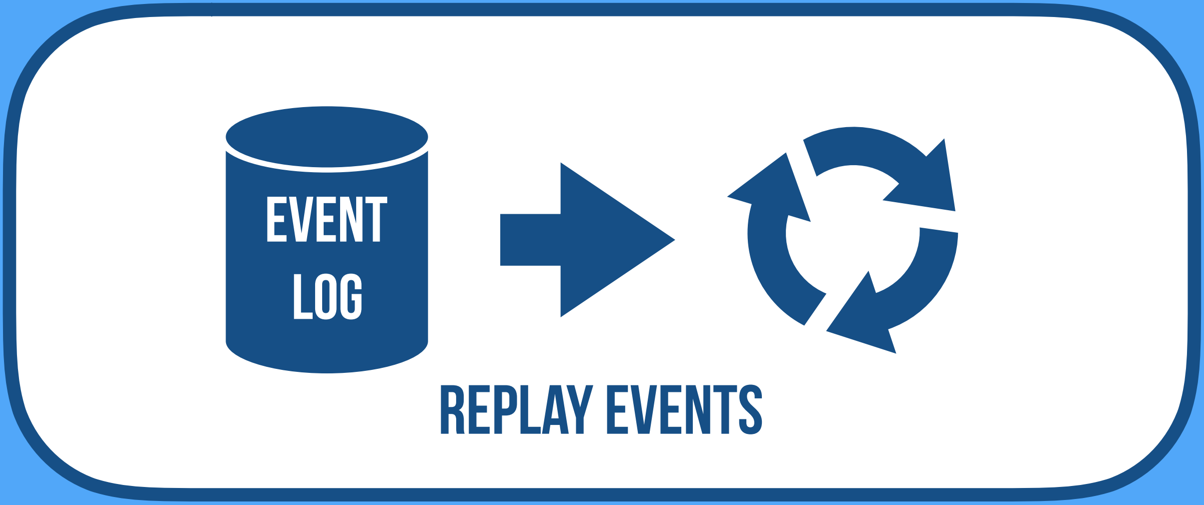
Event Sourced Functions

SAD PATH, RECOVER FROM FAILURE

Event Sourced Functions

SAD PATH, RECOVER FROM FAILURE

Event Sourced Functions



SAD PATH, RECOVER FROM FAILURE

ACID 2.0

ACID 2.0

Associative

Batch-insensitive
(grouping doesn't matter)

$a+(b+c)=(a+b)+c$

ACID 2.0

Associative

Batch-insensitive

(grouping doesn't matter)

$a+(b+c)=(a+b)+c$

Commutative

Order-insensitive

(order doesn't matter)

$a+b=b+a$

ACID 2.0

Associative

Batch-insensitive

(grouping doesn't matter)

$a+(b+c)=(a+b)+c$

Commutative

Order-insensitive

(order doesn't matter)

$a+b=b+a$

Idempotent

Retransmission-insensitive

(duplication does not matter)

$a+a=a$

CONFLICT-FREE REPLICATED DATA TYPES

Convergent & Commutative Replicated Data Types - Shapiro et. al. 2011

CONFLICT-FREE REPLICATED DATA TYPES

CRDT

ACID 2.0
Strong Eventual Consistency
Replicated & Decentralized
Always Converge Correctly
Monotonic Merge Function
Highly Available & Scalable

Convergent & Commutative Replicated Data Types - Shapiro et. al. 2011

CONFLICT-FREE REPLICATED DATA TYPES

CRDT

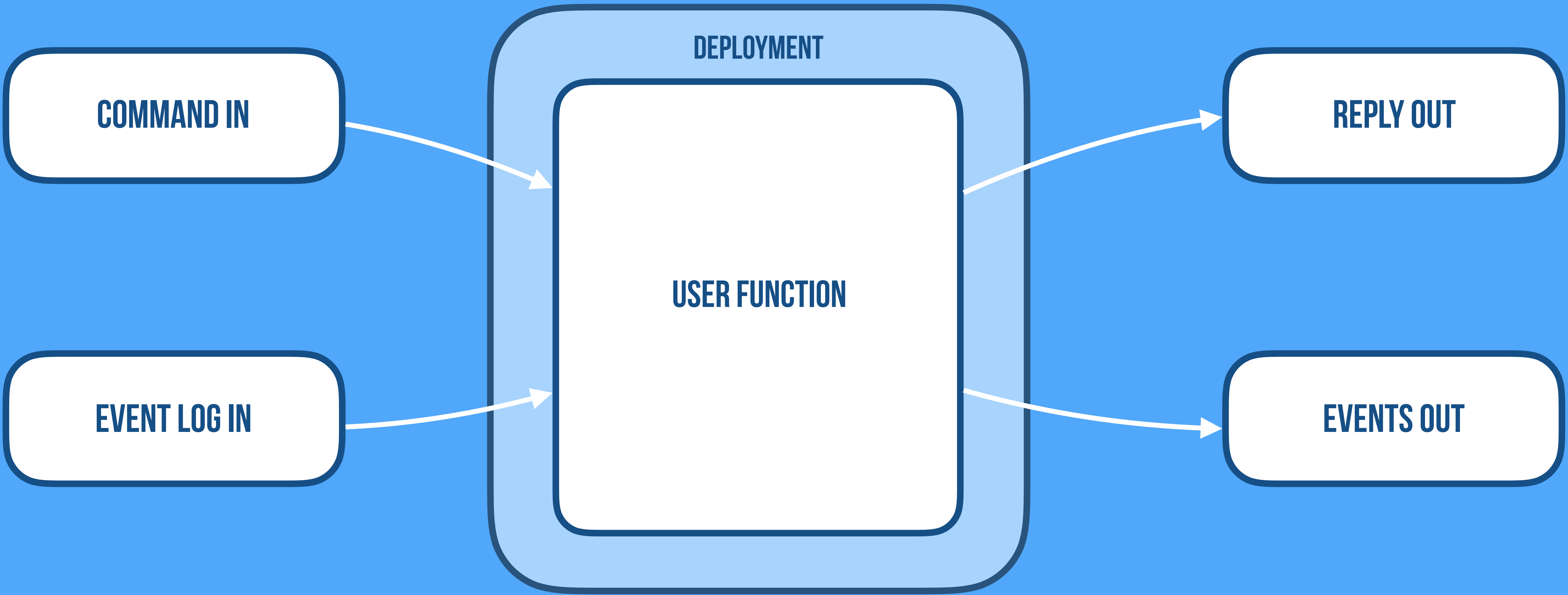
ACID 2.0
Strong Eventual Consistency
Replicated & Decentralized
Always Converge Correctly
Monotonic Merge Function
Highly Available & Scalable

DATA TYPES

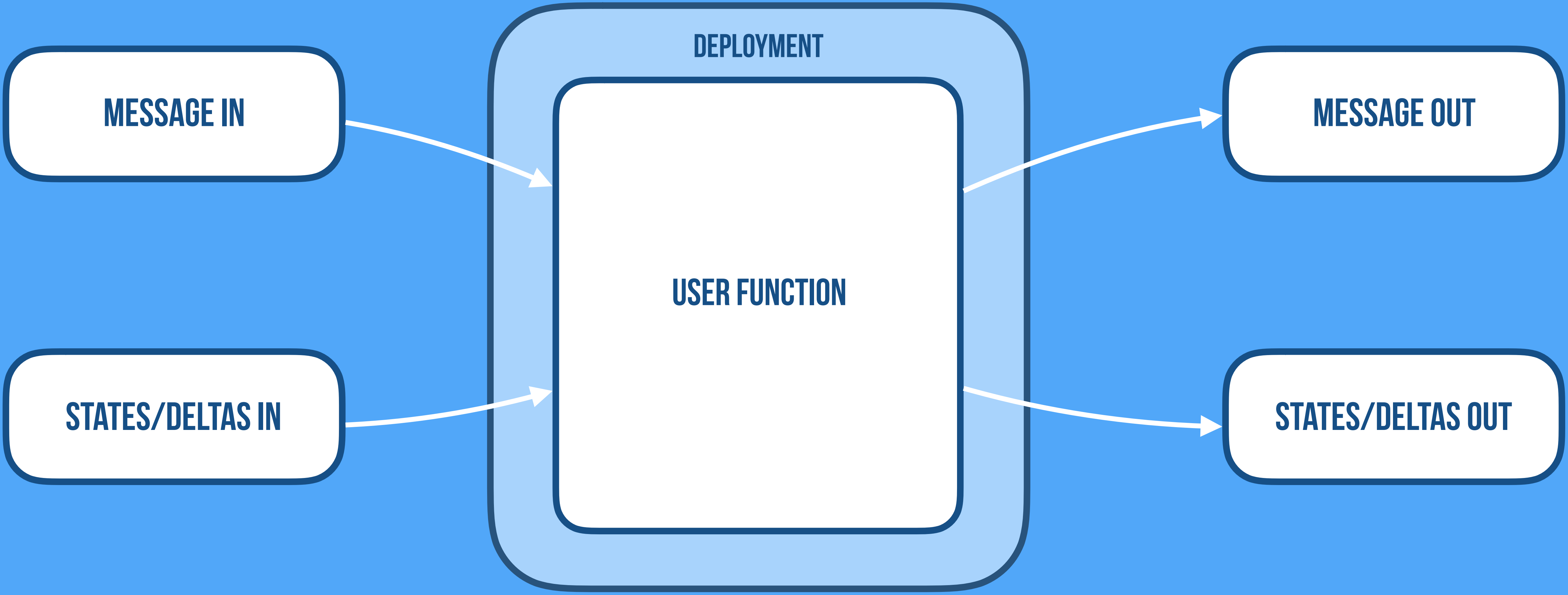
Counters
Registers
Sets
Maps
Graphs
(that all compose)

Convergent & Commutative Replicated Data Types - Shapiro et. al. 2011

Serverless Event Sourcing



Serverless CRDTs



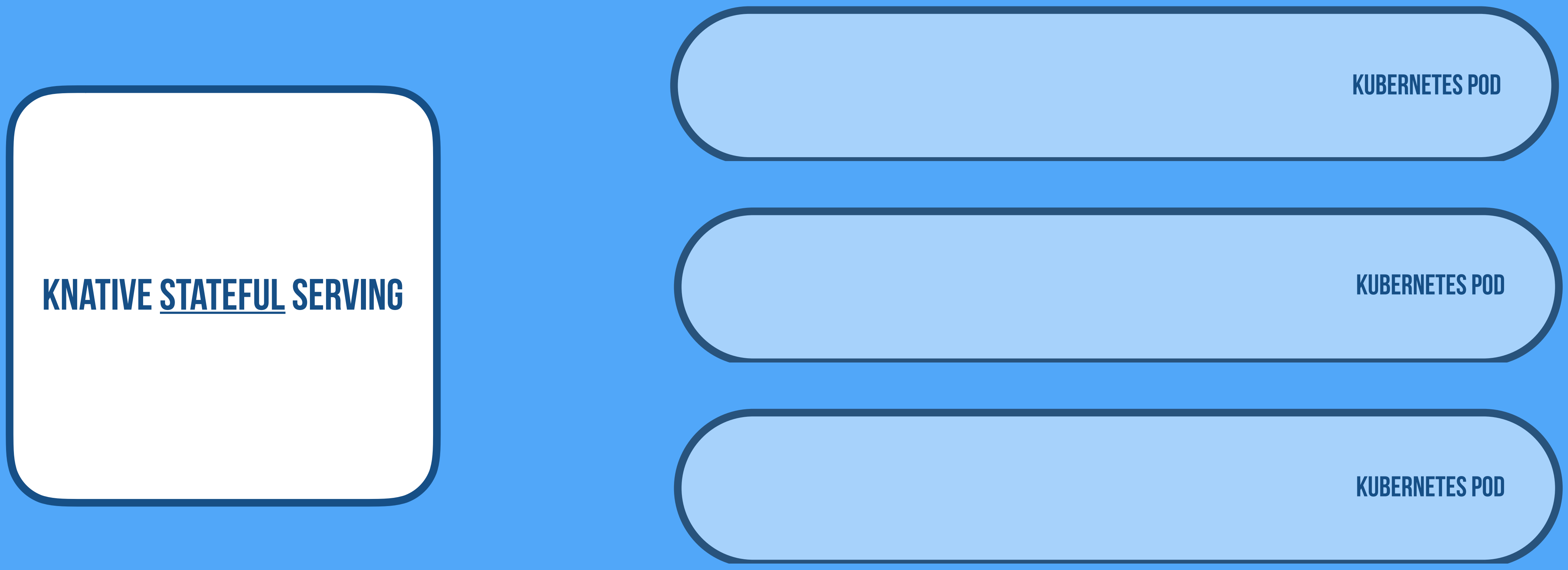
SO, WHAT ARE WE DOING ABOUT IT?

SERVING OF STATEFUL FUNCTIONS

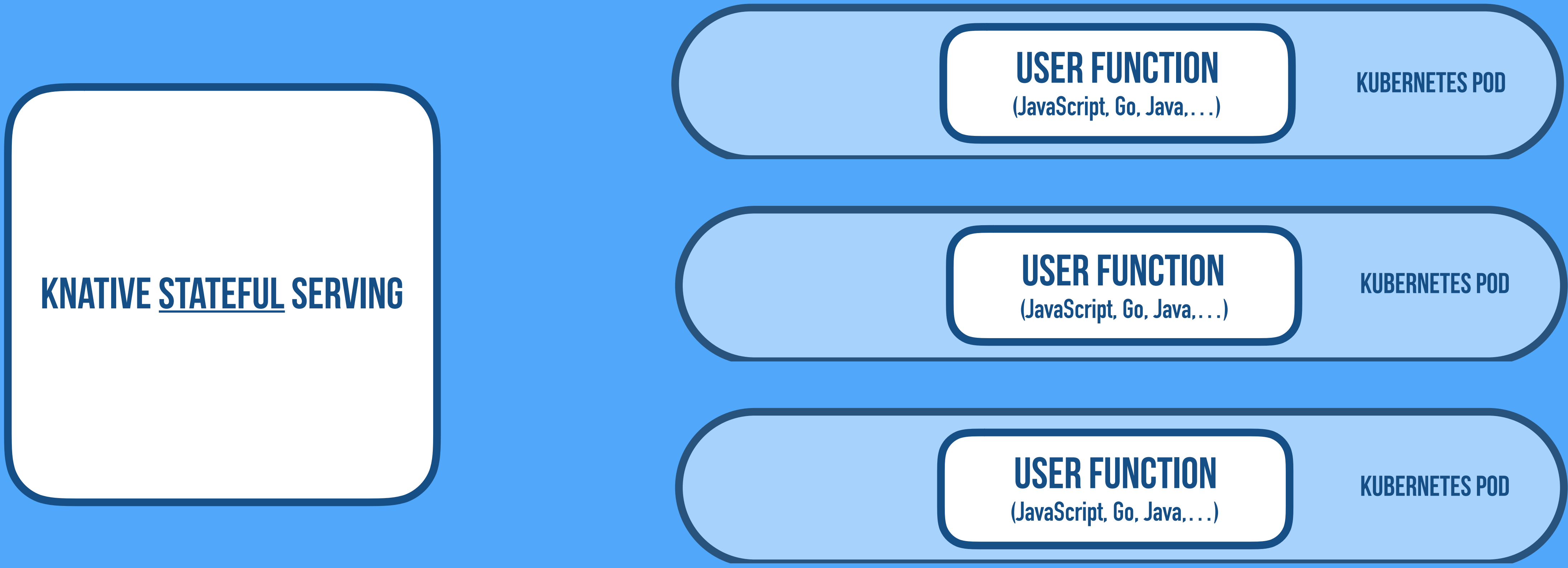
SERVING OF STATEFUL FUNCTIONS

KNATIVE STATEFUL SERVING

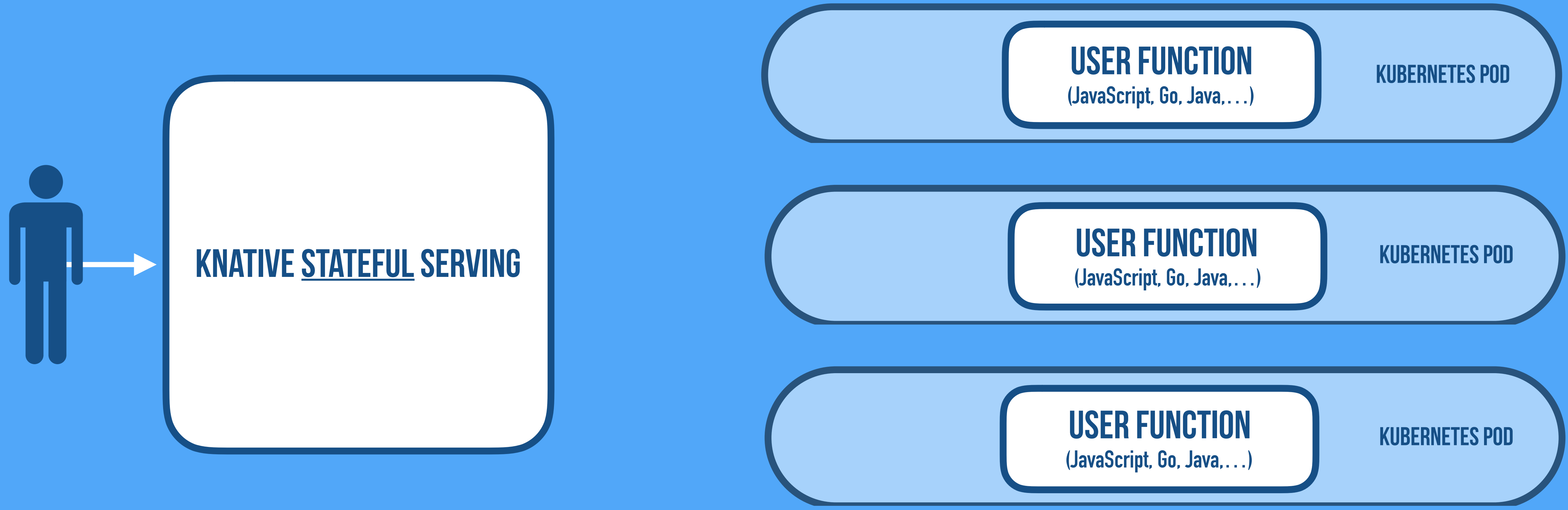
SERVING OF STATEFUL FUNCTIONS



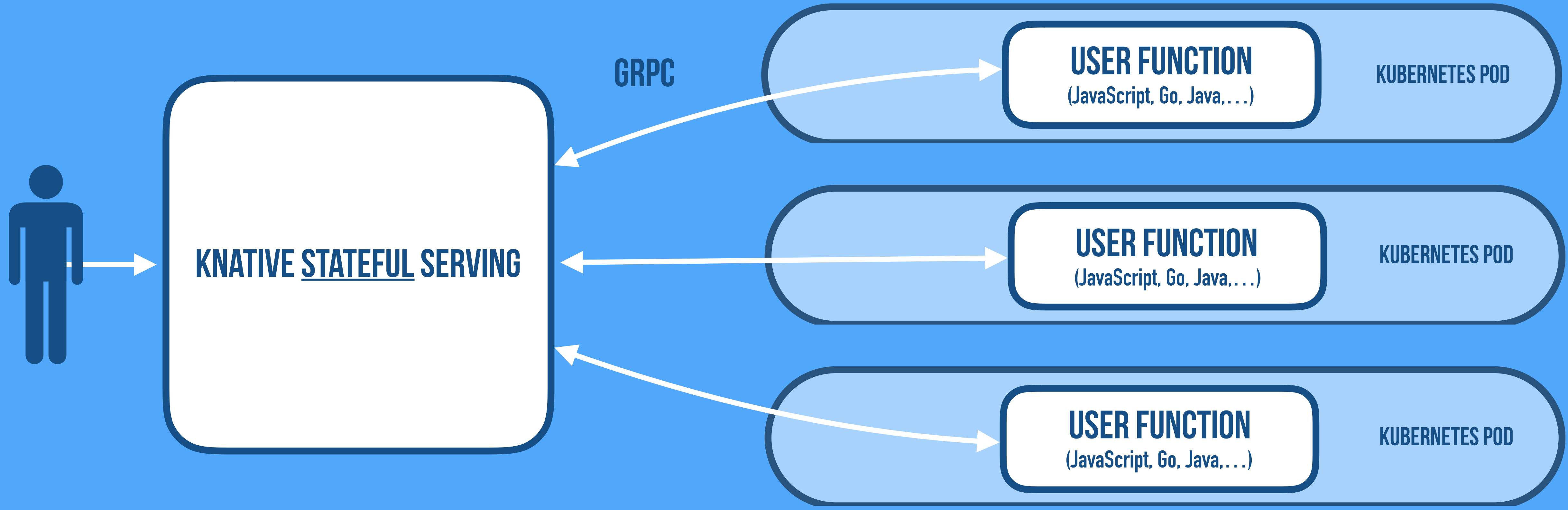
SERVING OF STATEFUL FUNCTIONS



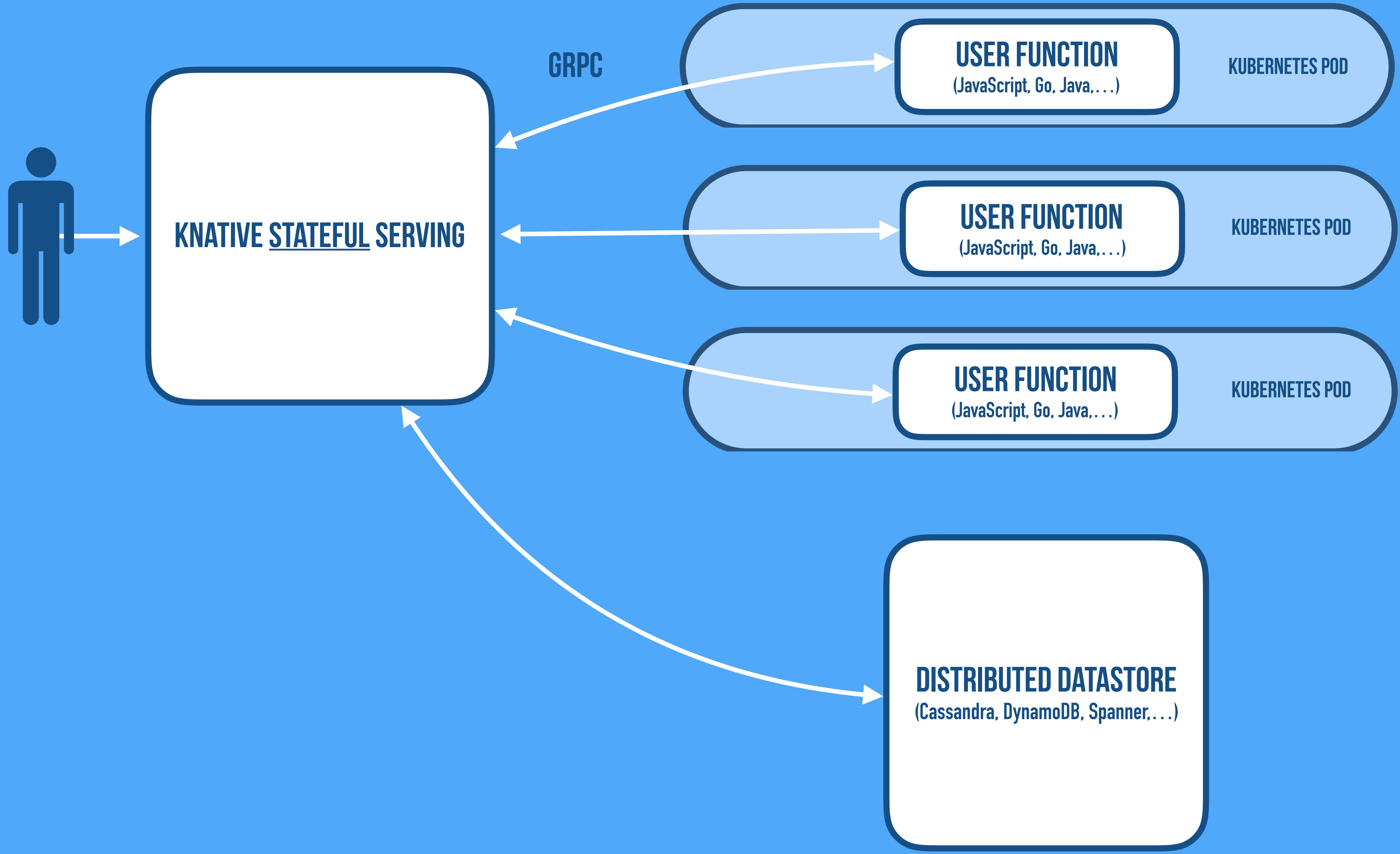
SERVING OF STATEFUL FUNCTIONS



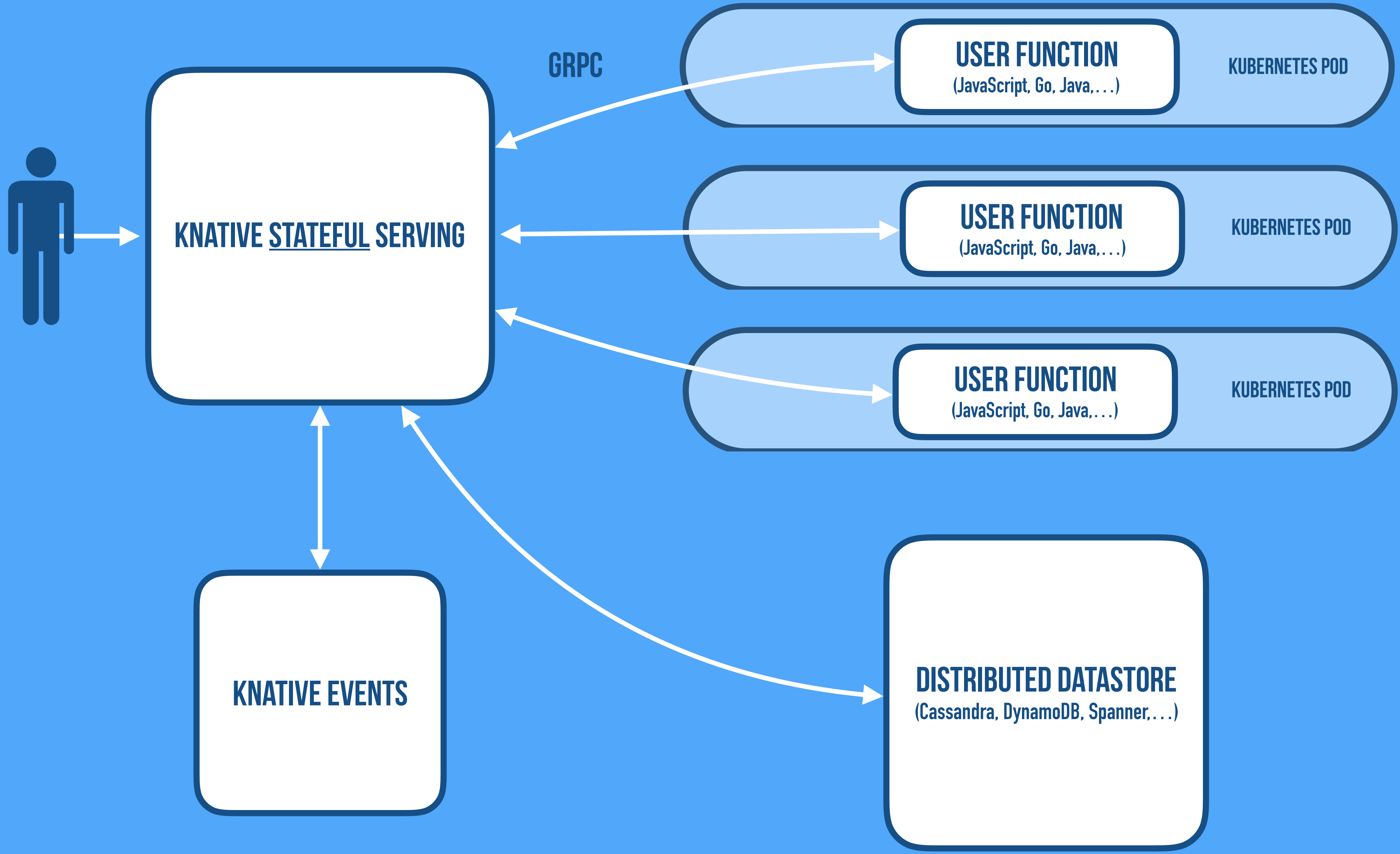
SERVING OF STATEFUL FUNCTIONS



SERVING OF STATEFUL FUNCTIONS



SERVING OF STATEFUL FUNCTIONS



WHAT IS AKKA?

WHAT IS AKKA?

- Cloud Native, Reactive, Distributed Systems Runtime
  - Implementation of the Actor Model—Concurrency and Distribution
  - Decentralized, Self-Organizing, Peer-to-peer Service Mesh
  - Autonomous, Self-healing, Event-driven Services

WHAT IS AKKA?

- Cloud Native, Reactive, Distributed Systems Runtime
  - Implementation of the Actor Model—Concurrency and Distribution
  - Decentralized, Self-Organizing, Peer-to-peer Service Mesh
  - Autonomous, Self-healing, Event-driven Services

- High Performance, High Throughput, and Low Latency
  - Communication: Point-to-Point, Pub-Sub, and Streaming
  - Protocols: HTTP, TCP, Aeron (UDP), Kafka, Reactive Streams, gRPC

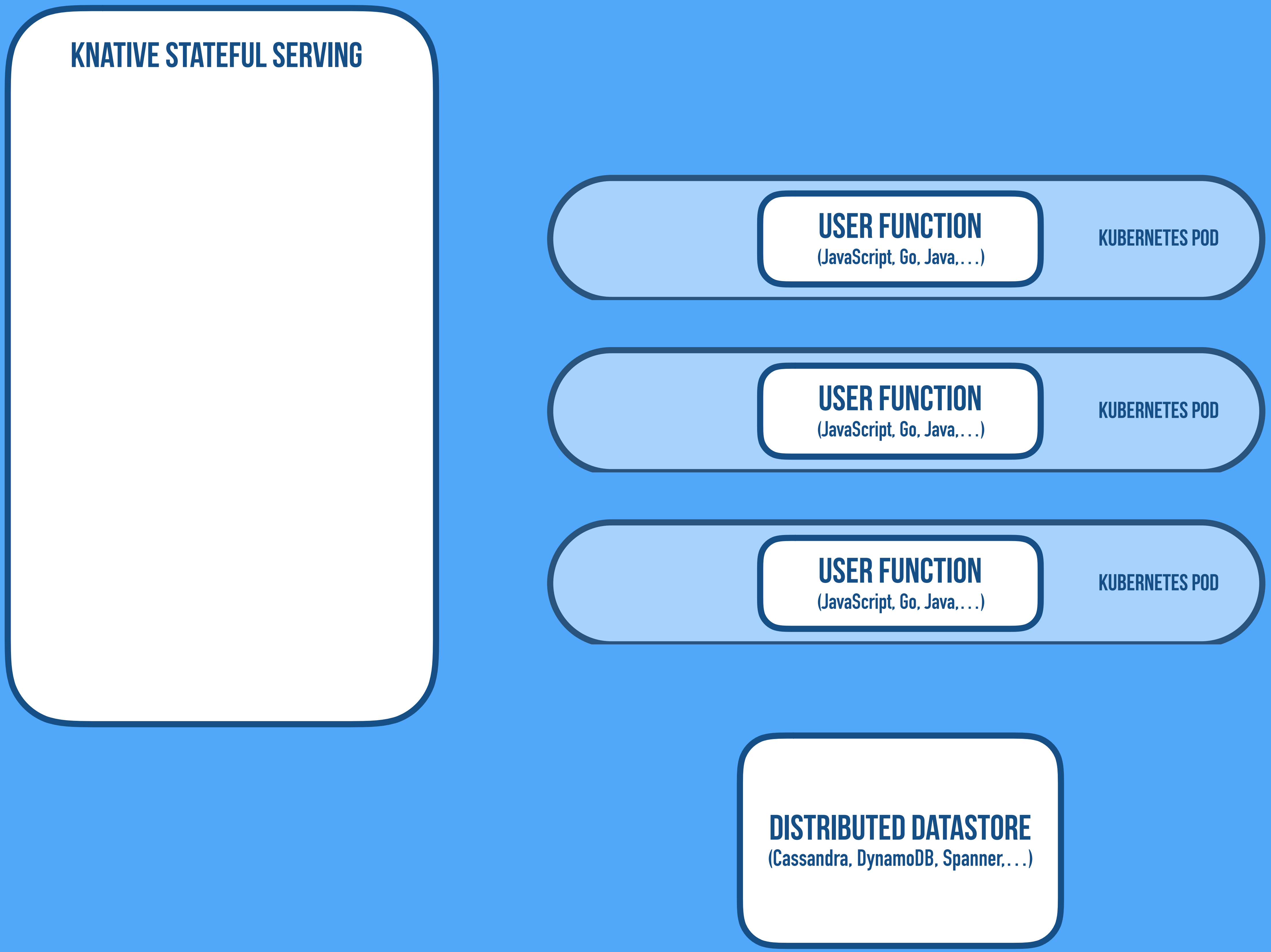
WHAT IS AKKA?

- Cloud Native, Reactive, Distributed Systems Runtime
  - Implementation of the Actor Model—Concurrency and Distribution
  - Decentralized, Self-Organizing, Peer-to-peer Service Mesh
  - Autonomous, Self-healing, Event-driven Services
- High Performance, High Throughput, and Low Latency
  - Communication: Point-to-Point, Pub-Sub, and Streaming
  - Protocols: HTTP, TCP, Aeron (UDP), Kafka, Reactive Streams, gRPC
- Distributed State Management
  - CRDTs, Event Sourcing, CQRS
  - Multi-Datacenter Clustering and Log Replication

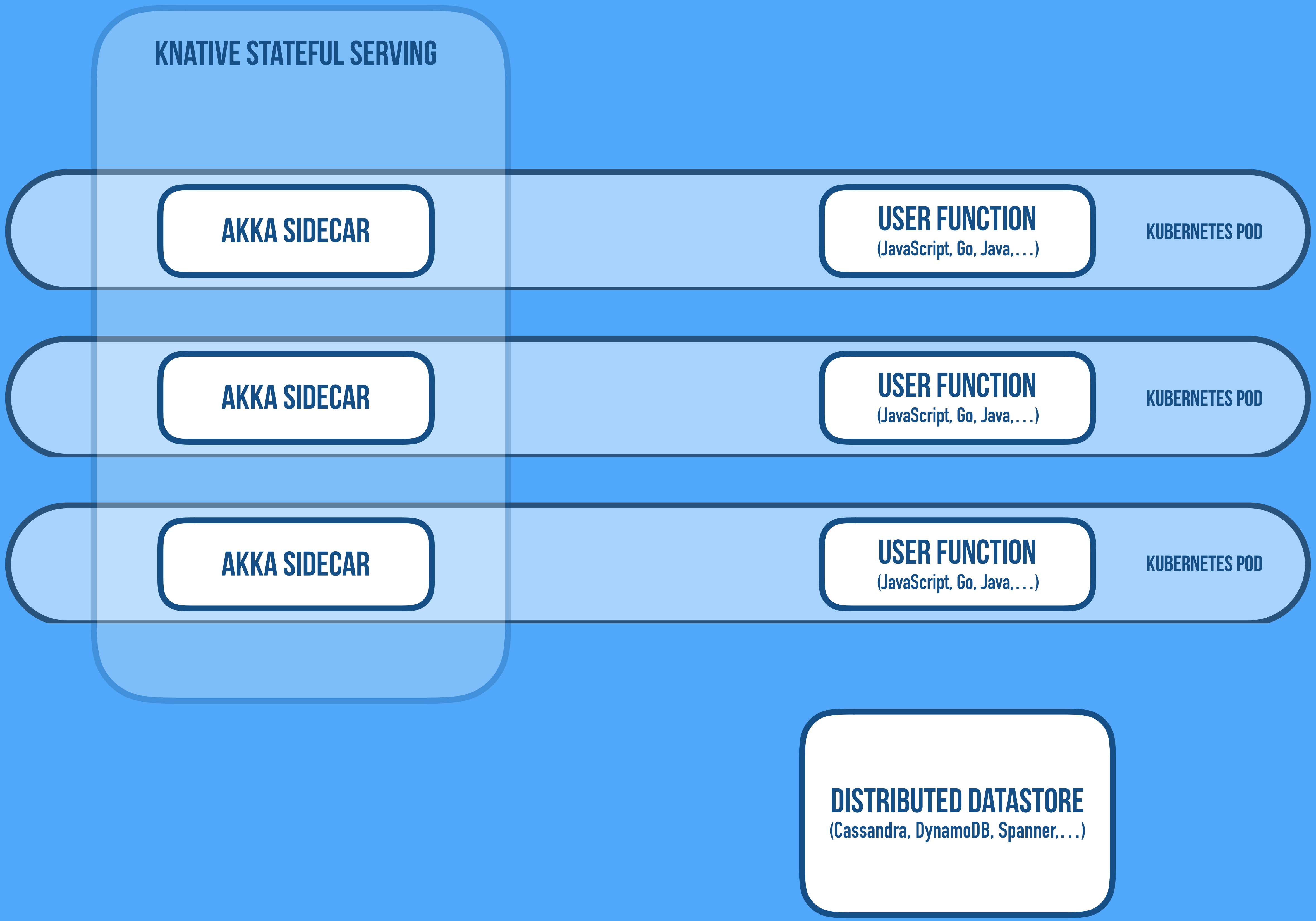
WHAT IS AKKA?

- Cloud Native, Reactive, Distributed Systems Runtime
  - ➡ Implementation of the Actor Model—Concurrency and Distribution
  - ➡ Decentralized, Self-Organizing, Peer-to-peer Service Mesh
  - ➡ Autonomous, Self-healing, Event-driven Services
- High Performance, High Throughput, and Low Latency
  - ➡ Communication: Point-to-Point, Pub-Sub, and Streaming
  - ➡ Protocols: HTTP, TCP, Aeron (UDP), Kafka, Reactive Streams, gRPC
- Distributed State Management
  - ➡ CRDTs, Event Sourcing, CQRS
  - ➡ Multi-Datacenter Clustering and Log Replication
- Find out more at: akka.io

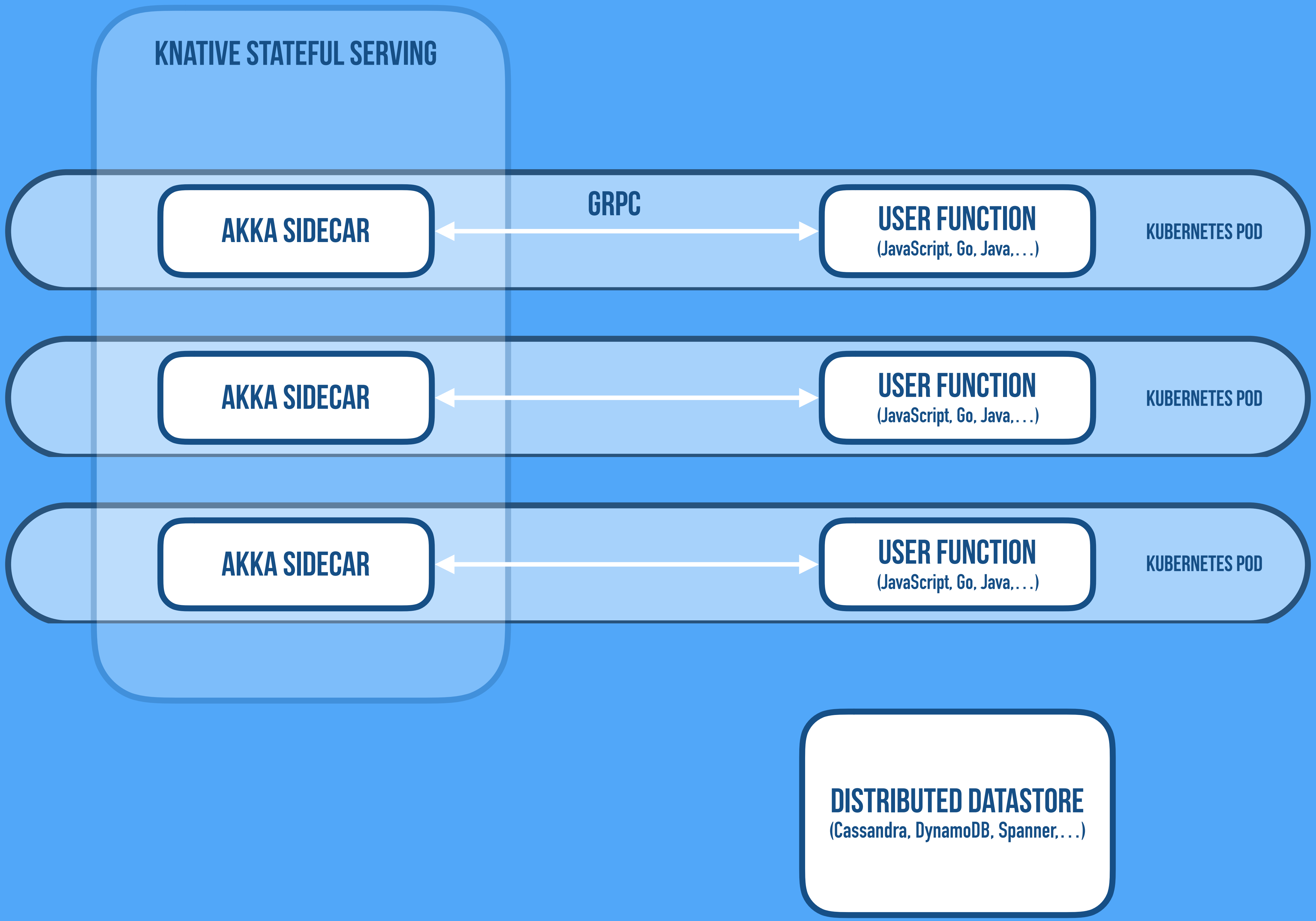
POWERED BY AKKA CLUSTER SIDECARS



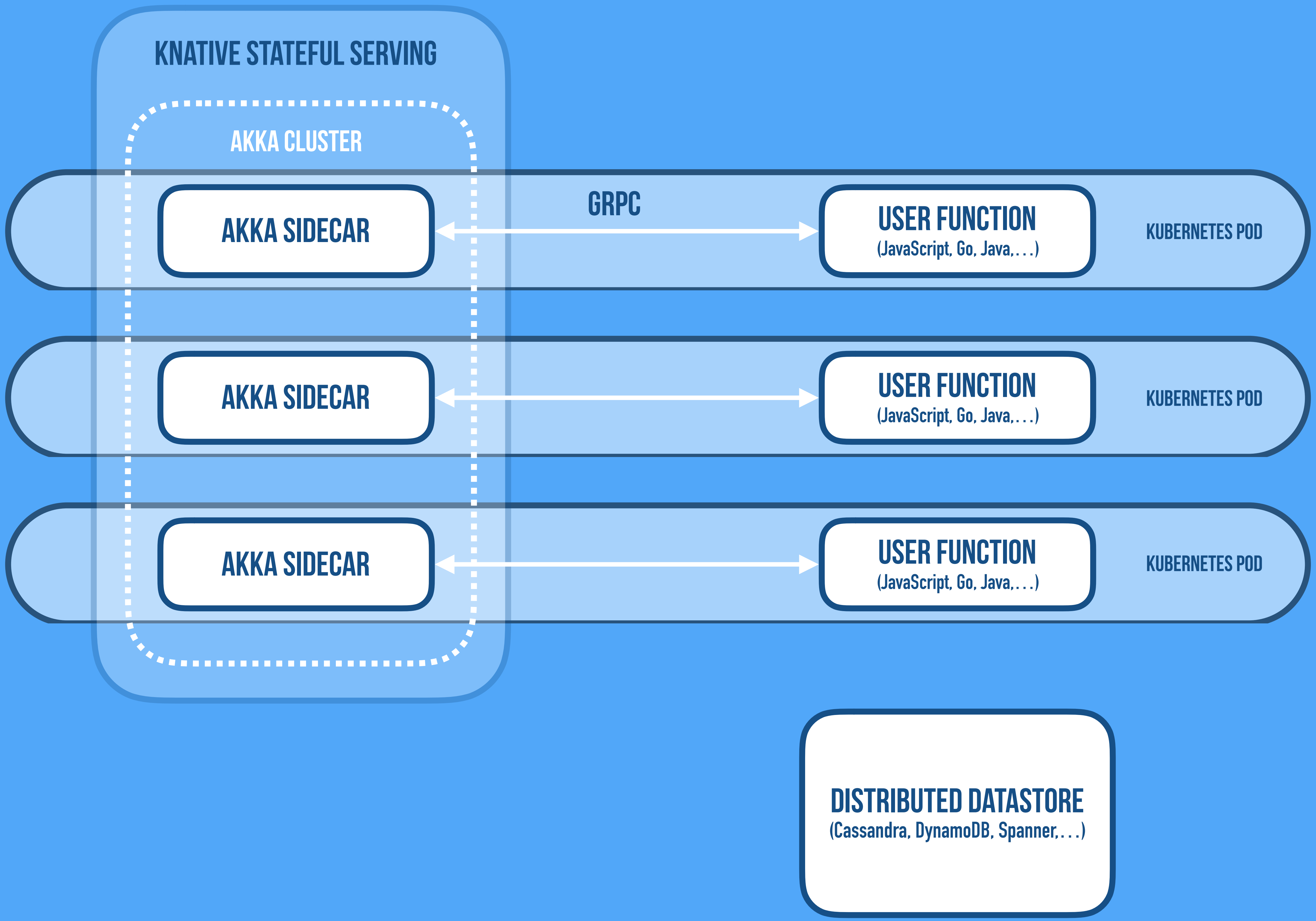
POWERED BY AKKA CLUSTER SIDECARS



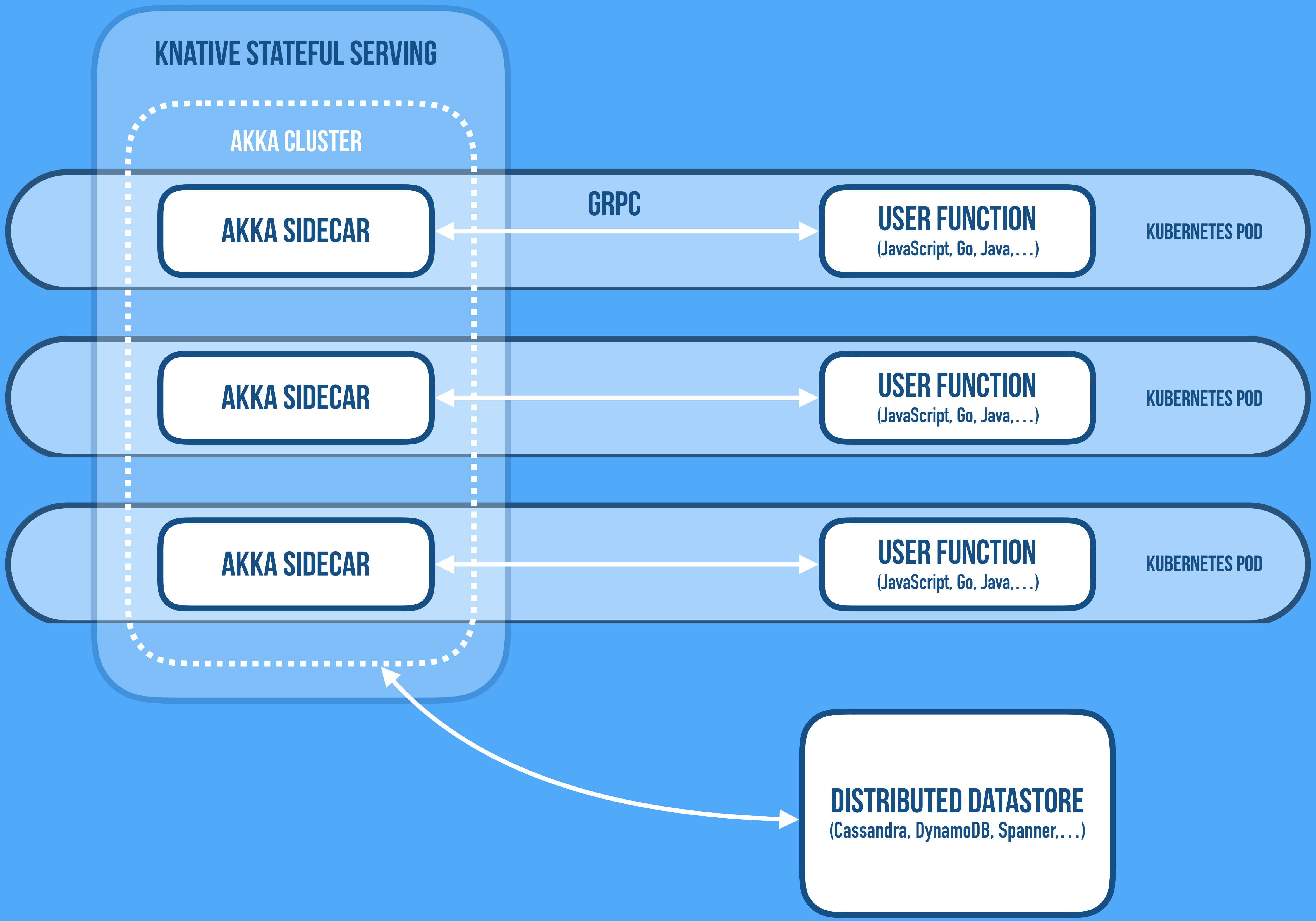
POWERED BY AKKA CLUSTER SIDECARS



POWERED BY AKKA CLUSTER SIDECARS



POWERED BY AKKA CLUSTER SIDECARS



In Summary

1. The Serverless DX is revolutionary and will grow to dominate the future of Cloud Computing
2. FaaS is a good first step, but with limited addressable use-cases
3. Serverless 2.0 needs a runtime & programming model for general-purpose application development
4. We have started building it with Knative, Akka, and gRPC
5. We need your help

Get Involved

bit.ly/stateful-serverless-intro

github.com/lightbend/stateful-serverless

Thank You

Jonas Bonér

@jboner

Lightbend