Beyond FaaS
Towards Stateful Serverless

Jonas Bonér
@jboner

“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

FaaS

FaaS = Function-as-a-Service

FaaS

IS VISIONARY

FaaS = Function-as-a-Service

FaaS

IS VISIONARY

PAVED THE WAY

FaaS = Function-as-a-Service

FaaS

IS VISIONARY
PAVED THE WAY
JUST THE FIRST STEP

FaaS = Function-as-a-Service

SERVERLESS $\neq$ FAAS

GOOD USE-CASES FOR FAAS?

GOOD USE-CASES FOR FAAS?

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

GOOD 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. Embarrassingly parallel processing tasks—invoked on demand & intermittently, examples include: image processing, object recognition, log analysis
2. Low traffic applications—enterprise IT services, and spiky workloads
3. Stateless web applications—serving static content form S3 (or similar)
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.

FAAS: HARD TO BUILD

GENERAL-PURPOSE APPLICATIONS

FAAS: HARD TO BUILD

GENERAL-PURPOSE APPLICATIONS

1. Functions are stateless, ephemeral, short-lived:
   expensive to lose computational context & rehydrate
2. Durable state is always "somewhere else"
3. No co-location of state and processing
4. No direct addressability—all communication over external storage
5. Limited options for managing & coordinating distributed state
6. Limited options for modelling data consistency guarantees


STATE

We Need Serverless Support For...

- Managing in-memory durable session state across individual requests

  E.g. User Sessions, Shopping Carts, Caching
- Low-latency serving of dynamic in-memory models

  E.g. Serving of Machine Learning Models
- Real-time stream processing

  E.g. Recommendation, Anomaly Detection, Prediction Serving
- Distributed resilient transactional workflows

  E.g. Saga Pattern, Workflow Orchestration, Rollback/Compensating Actions
- Shared collaborative workspaces

  E.g. Collaborative Document Editing, Blackboards, Chat Rooms
- Leader election, counting, voting

  . . . .and other 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. Options for distributed coordination and communication patterns

   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

Technical Requirements

1. Stateful long-lived addressable virtual components

   Actors

2. Options for distributed coordination and communication patterns

   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

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. Options for distributed coordination and communication patterns

   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

3. Options for managing distributed state reliably at scale

   Ranging from strong to eventual consistency (durable/ephemeral)

4. Intelligent adaptive placement of stateful functions

   Physical co-location of state and processing, sharding, and sticky routing

Technical Requirements

1. Stateful long-lived addressable virtual components

   Actors

2. Options for distributed coordination and communication patterns

   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

3. Options for managing distributed state reliably at scale

   Ranging from strong to eventual consistency (durable/ephemeral)

4. Intelligent adaptive placement of stateful functions

   Physical co-location of state and processing, sharding, and sticky routing

5. Predictable performance, latency, and throughput

   In startup time, communication/coordination, and storage of data

Technical Requirements

1. Stateful long-lived addressable virtual components

   Actors

2. Options for distributed coordination and communication patterns

   Pub-Sub, Point-To-Point, Broadcast—CRDTs, Sagas, etc.

3. Options for managing distributed state reliably at scale

   Ranging from strong to eventual consistency (durable/ephemeral)

4. Intelligent adaptive placement of stateful functions

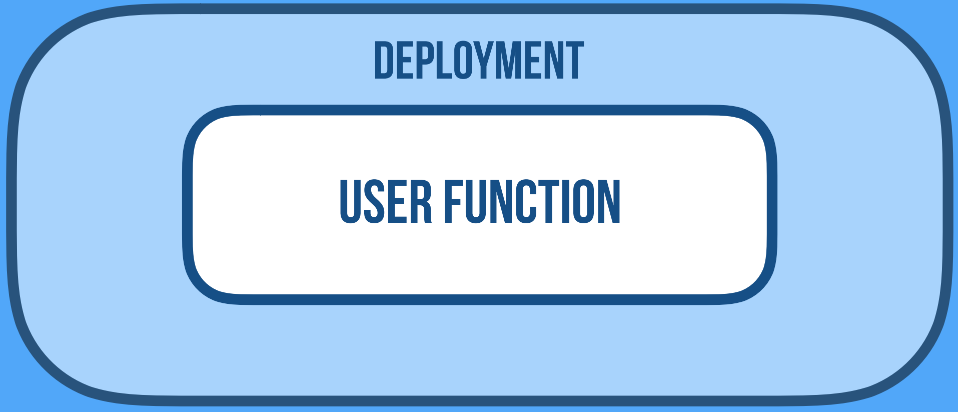
   Physical co-location of state and processing, sharding, and sticky routing

5. Predictable performance, latency, and throughput

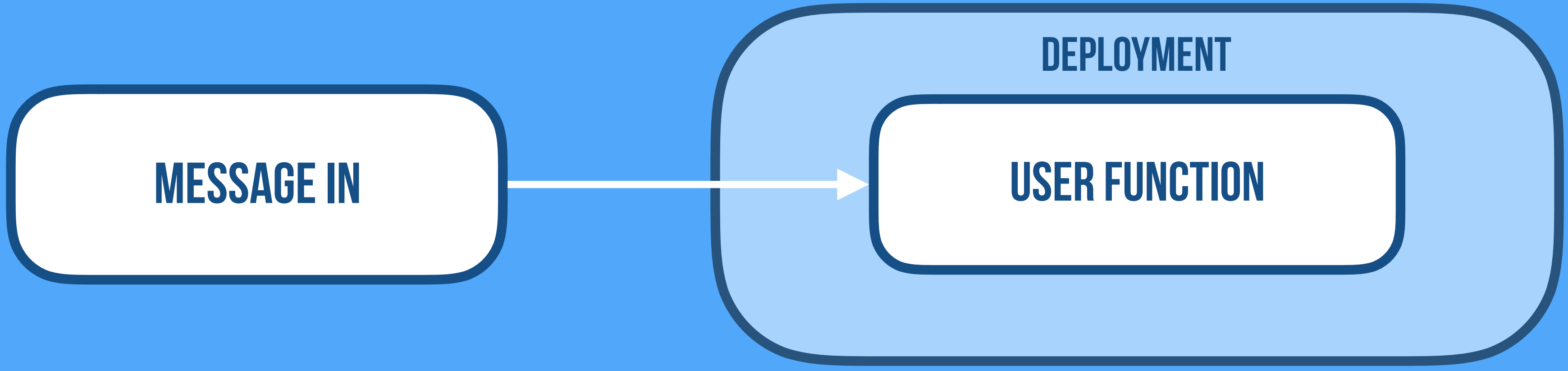
   In startup time, communication/coordination, and storage of data

6. Ways of managing end-to-end guarantees and correctness

FaaS
Abstracting Over Communication

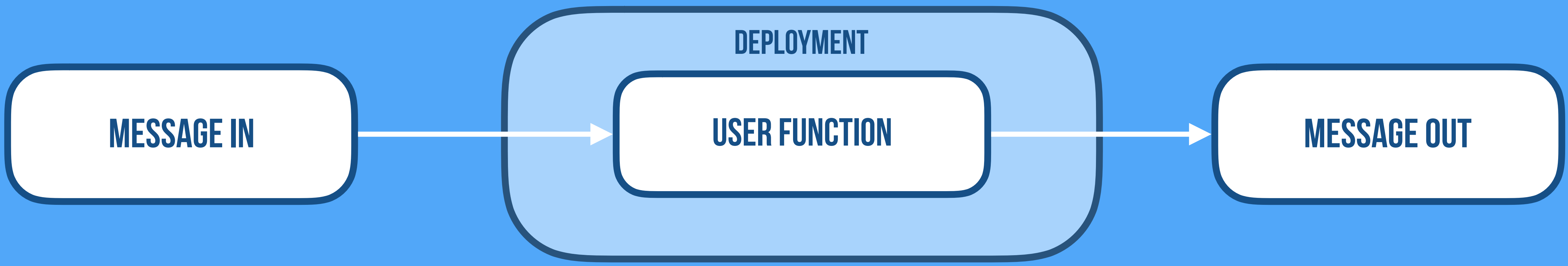


FaaS
Abstracting Over Communication

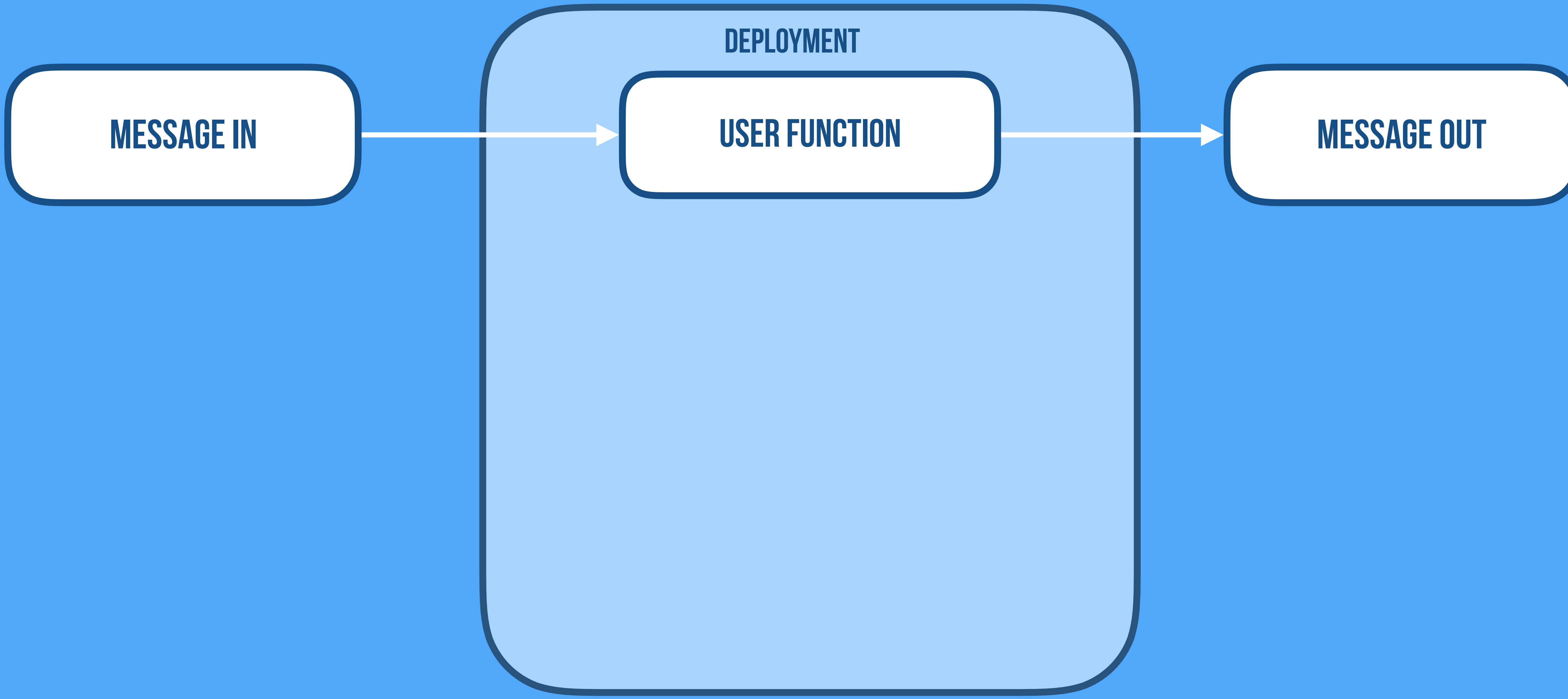


FaaS

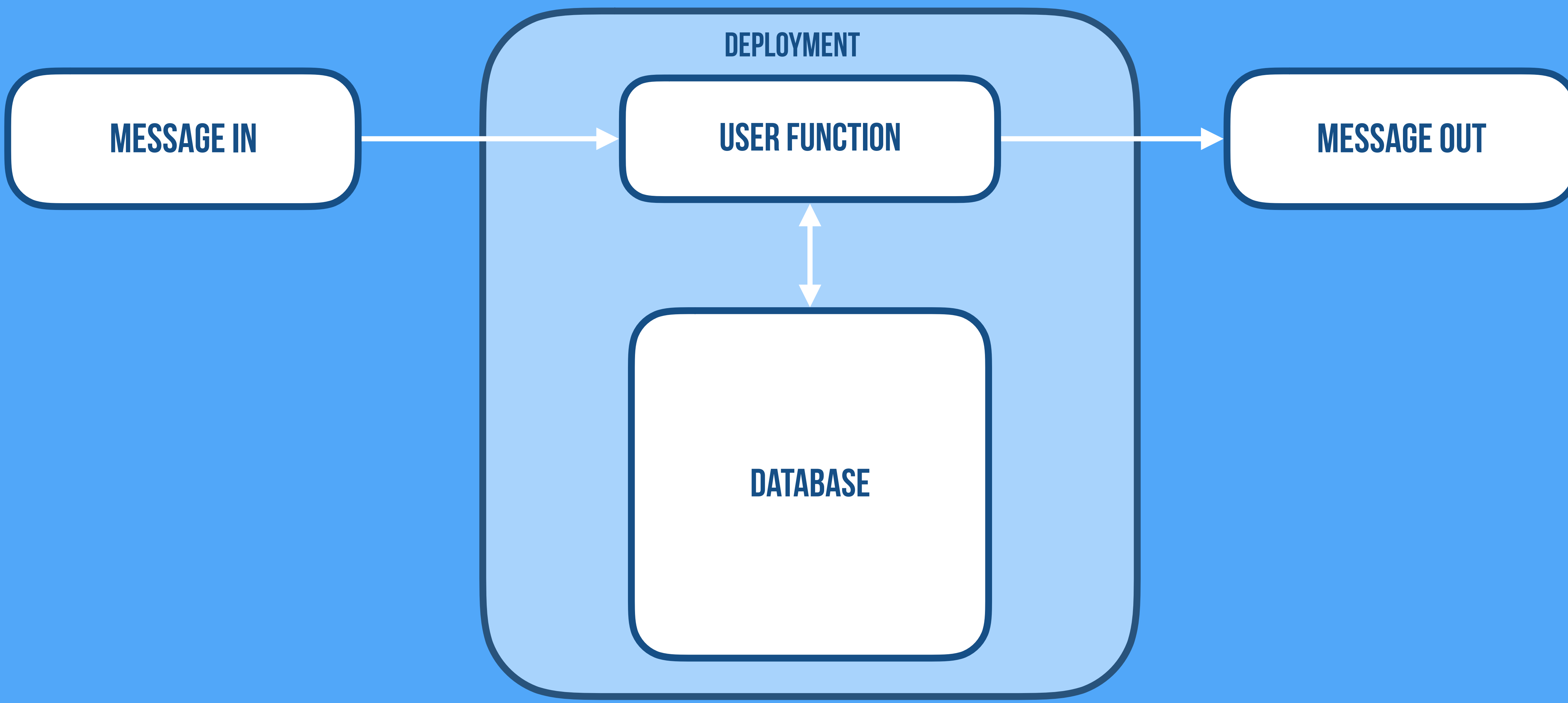
Abstracting Over Communication



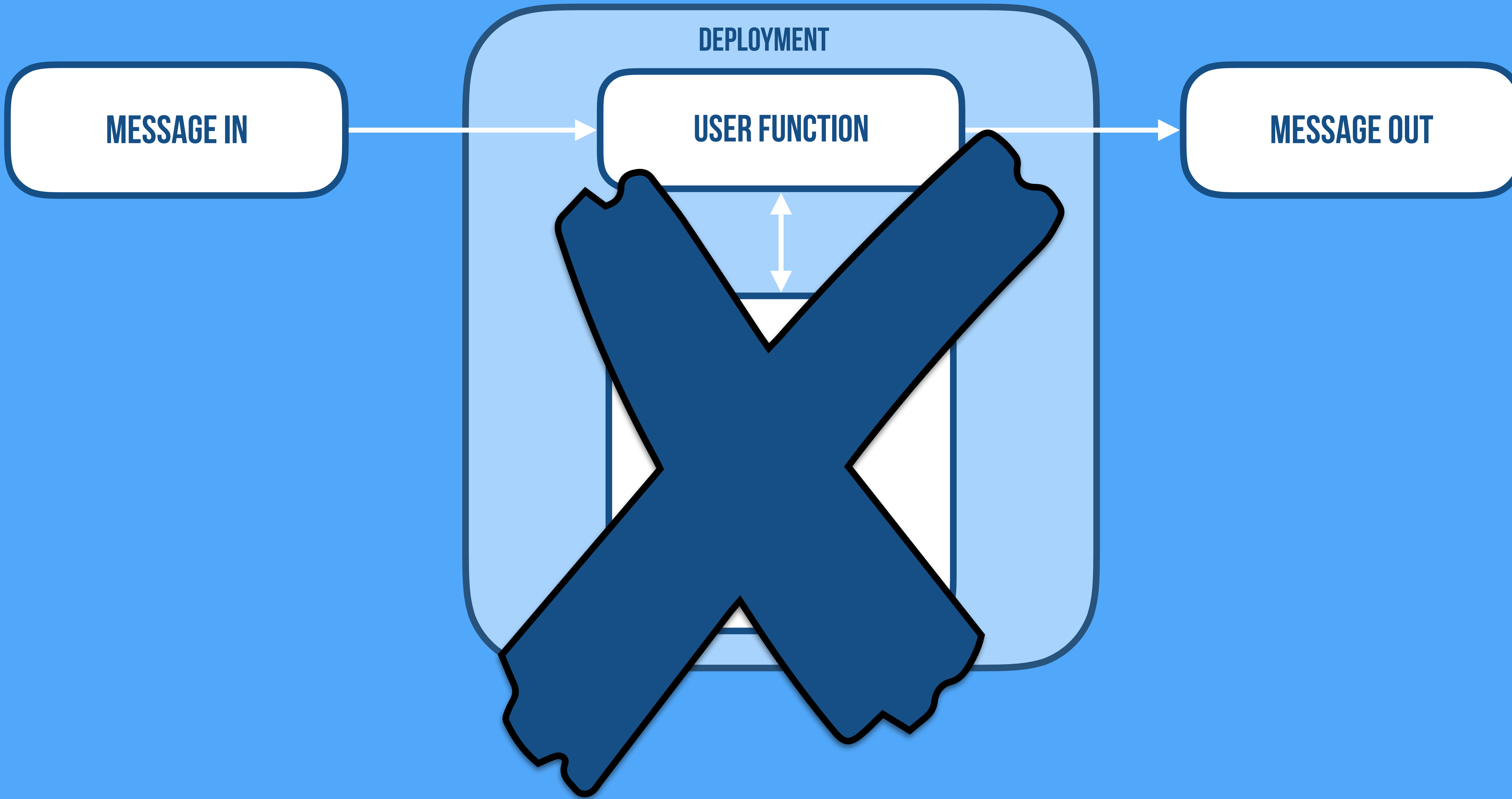
FaaS With CRUD



FaaS With CRUD



Not Serverless
In An Ideal World



UNCONSTRAINED DATABASE ACCESS MAKES IT HARD TO AUTOMATE OPERATIONS

Enter
Stateful
Serverless

Guiding Principles

Guiding Principles

1. Embrace State—Don’t ignore, hide, or delegate it

   Data locality matters. Faster insight into data is a competitive advantage.

Guiding Principles

1. **Embrace State—Don’t ignore, hide, or delegate it**

   Data locality matters. Faster insight into data is a competitive advantage.

2. **Embrace Failure—Unavoidable. Don’t prevent. Manage.**

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

Guiding Principles

1. Embrace State—Don't ignore, hide, or delegate it

   Data locality matters. Faster insight into data is a competitive advantage.

2. Embrace Failure—Unavoidable. Don't prevent. Manage.

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. Embrace Uncertainty—Manage it in the application layer

   End-to-end correctness/stability requires app working in concert w/ infra.

Guiding Principles

1. **Embrace State—Don't ignore, hide, or delegate it**

   Data locality matters. Faster insight into data is a competitive advantage.

2. **Embrace Failure—Unavoidable. Don't prevent. Manage.**

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. **Embrace Uncertainty—Manage it in the application layer**

   End-to-end correctness/stability requires app working in concert w/ infra.

4. **Avoid Needless Consistency**

   Not all data have need the same guarantees. Start with zero, add as needed.

Guiding Principles

1. Embrace State—Don't ignore, hide, or delegate it

   Data locality matters. Faster insight into data is a competitive advantage.

2. Embrace Failure—Unavoidable. Don't prevent. Manage.

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. Embrace Uncertainty—Manage it in the application layer

   End-to-end correctness/stability requires app working in concert w/ infra.

4. Avoid Needless Consistency

   Not all data have need the same guarantees. Start with zero, add as needed.

5. Avoid Needless Coordination and Communication

   Silence is Golden. Favour Eventual Consistency, CALM, CRDTs, ACID 2.0.

Guiding Principles

1. **Embrace State—Don't ignore, hide, or delegate it**

   Data locality matters. Faster insight into data is a competitive advantage.

2. **Embrace Failure—Unavoidable. Don't prevent. Manage.**

   Bulkhead and Contain. Signal and Die. Supervise and Manage.

3. **Embrace Uncertainty—Manage it in the application layer**

   End-to-end correctness/stability requires app working in concert w/ infra.

4. **Avoid Needless Consistency**

   Not all data have need the same guarantees. Start with zero, add as needed.

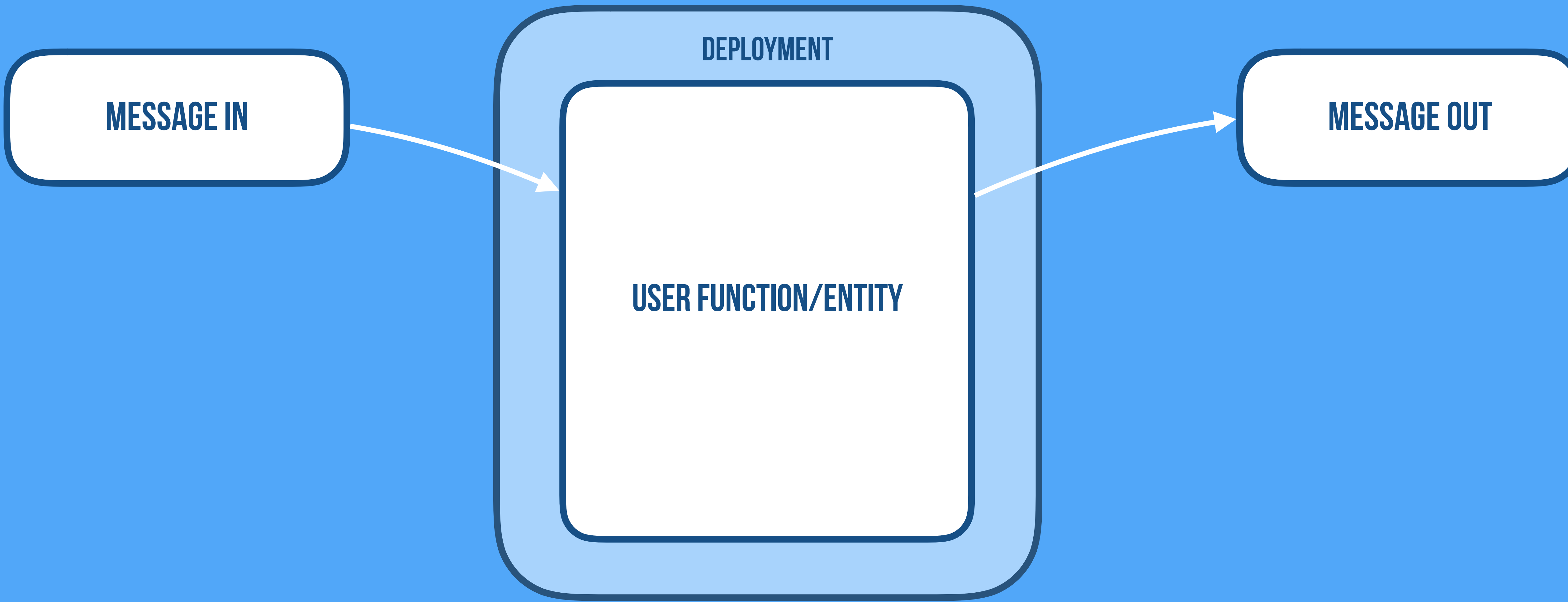
5. **Avoid Needless Coordination and Communication**

   Silence is Golden. Favour Eventual Consistency, CALM, CRDTs, ACID 2.0.

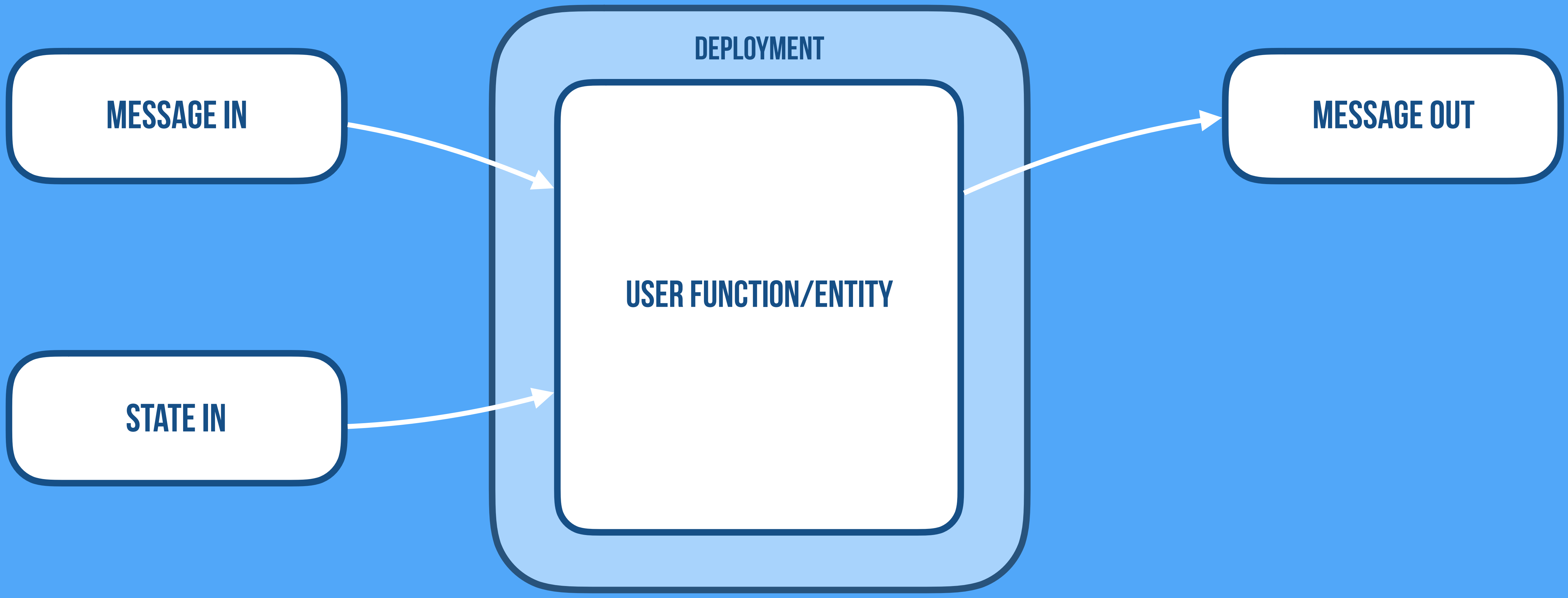
6. **Avoid Coupling in Time and Space**

   Go Async. Don't Block. Location Transparency. Guess/Apologize/Compensate.

Stateful Serverless
Abstracting Over State

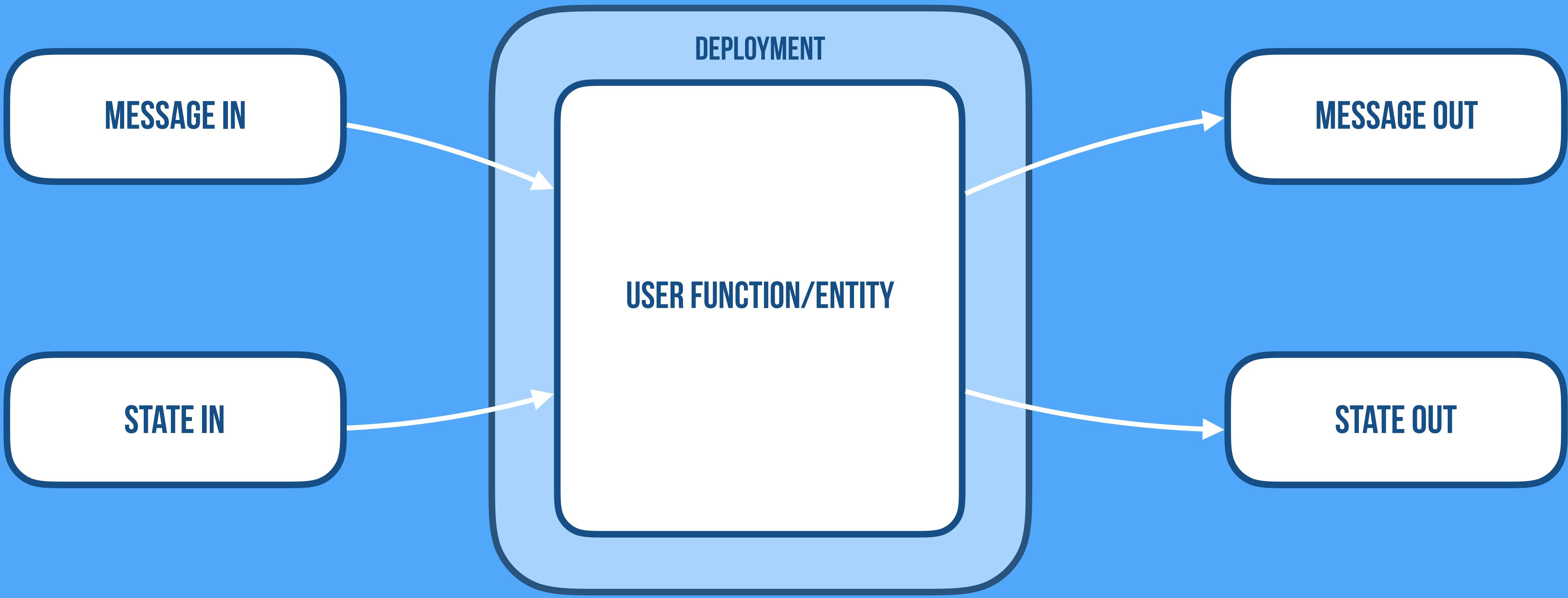


Stateful Serverless
Abstracting Over State



Stateful Serverless

Abstracting Over State



Let Us Use Better Models
For Distributed State

Let Us Use Better Models
For Distributed State

A FEW BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Let Us Use Better Models For Distributed State

A FEW BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Event Sourcing

Let Us Use Better Models For Distributed State

A FEW BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Event Sourcing

CRDTs

Let Us Use Better Models
For Distributed State

A FEW BATTLE-TESTED, YET CONSTRAINED, MODELS ARE:

Event Sourcing

CRDTs

Key Value

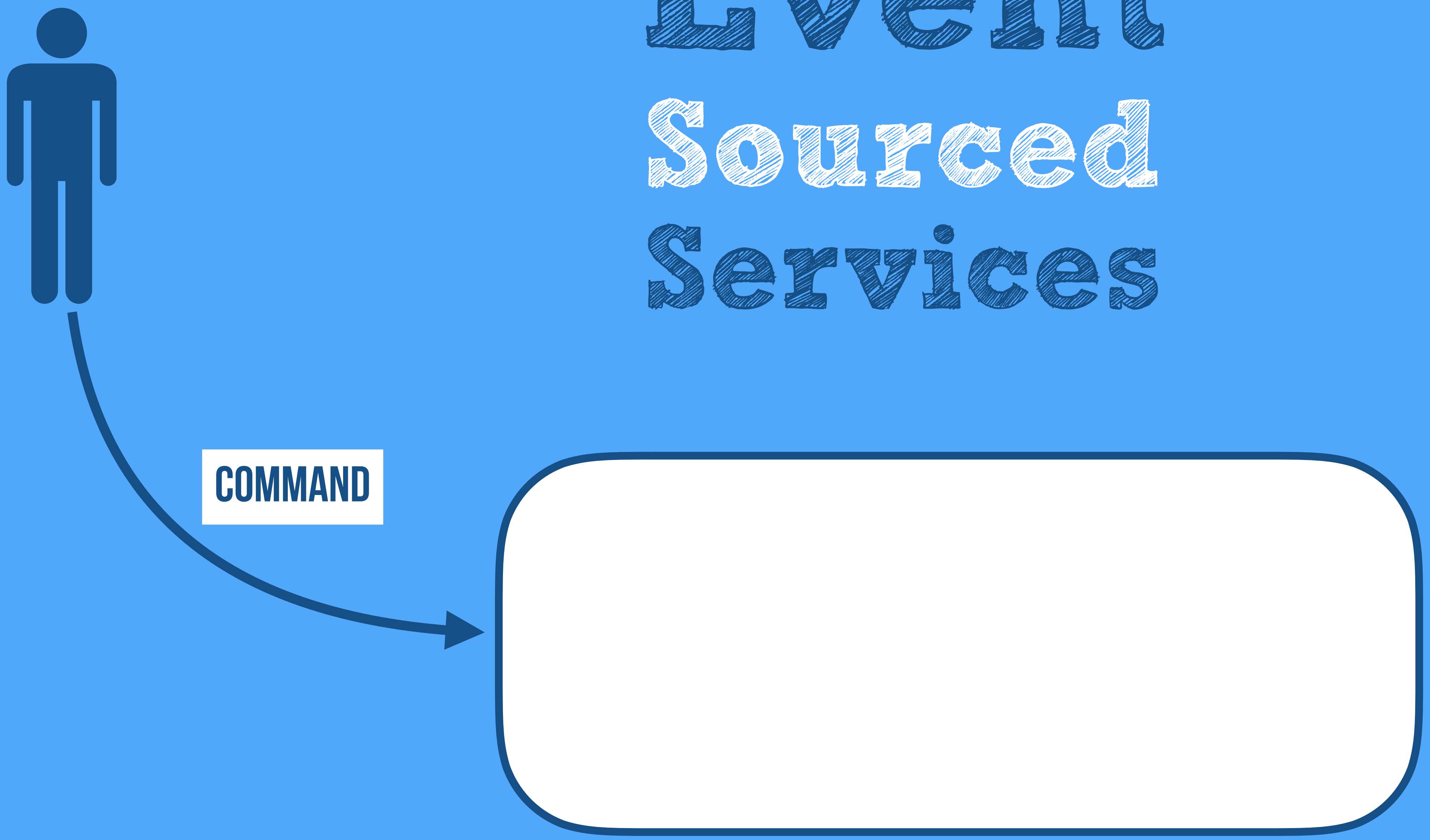
Event Sourced Services

HAPPY PATH

Event
Sourced
Services

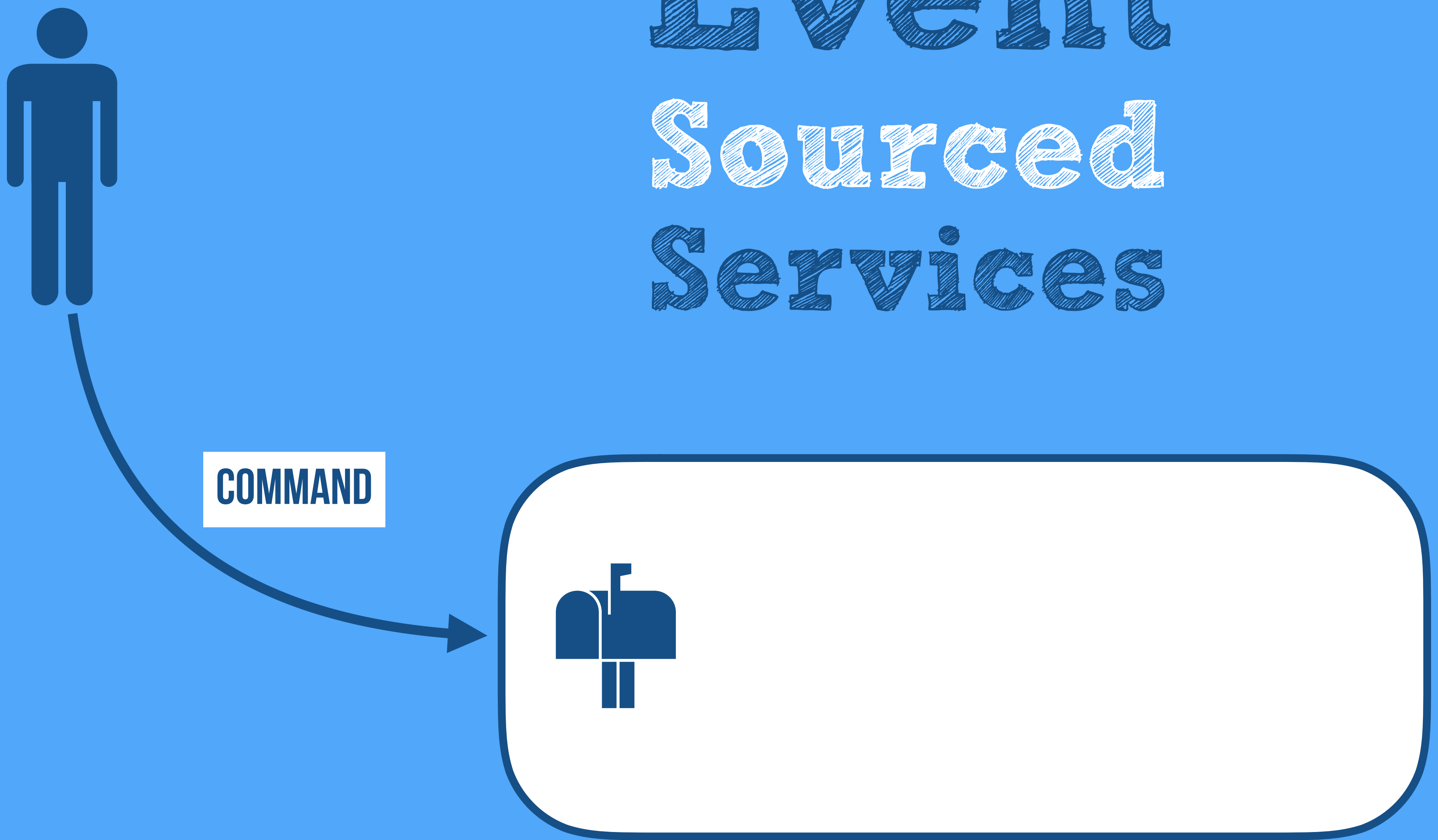
HAPPY PATH

Event Sourced Services



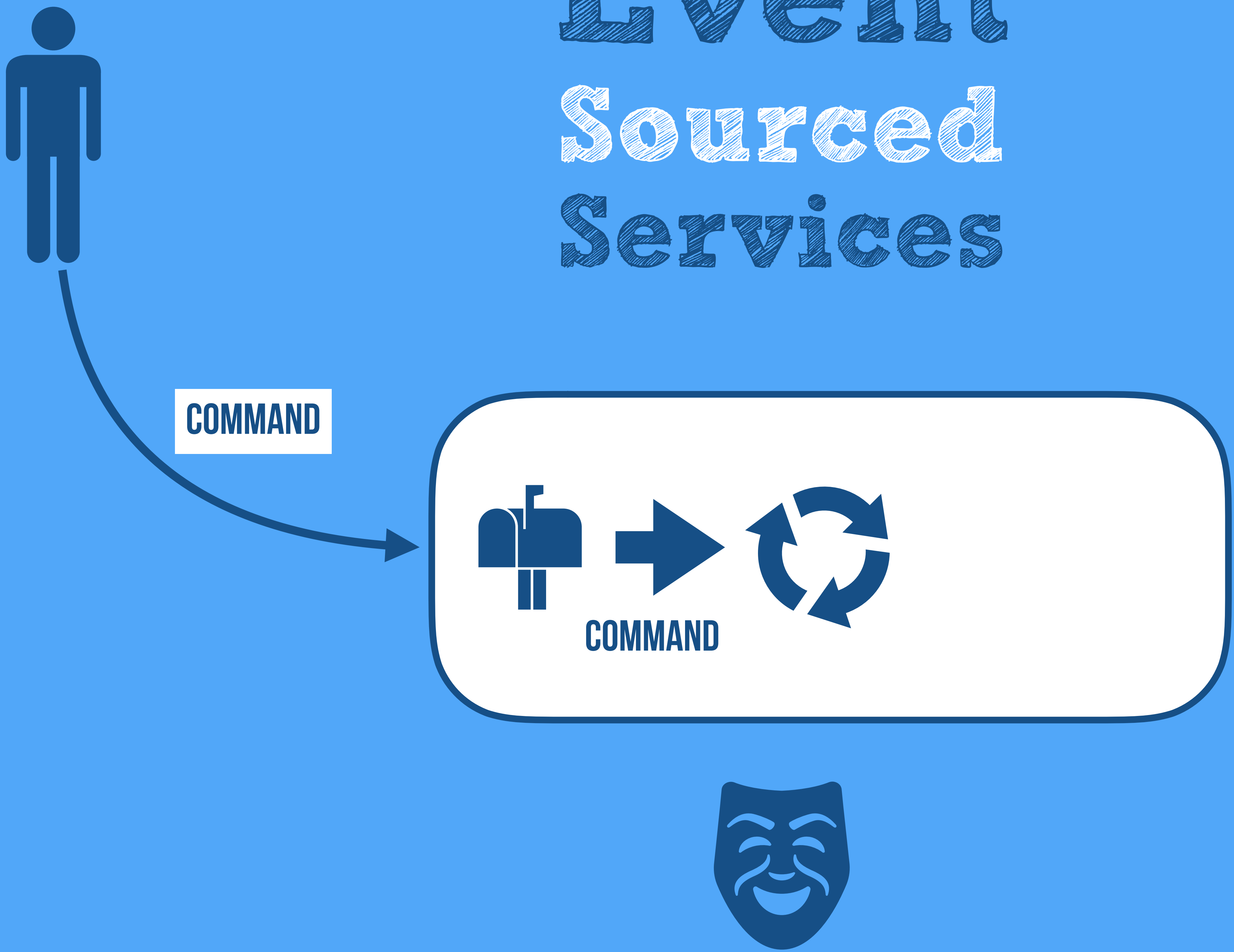
HAPPY PATH

Event Sourced Services



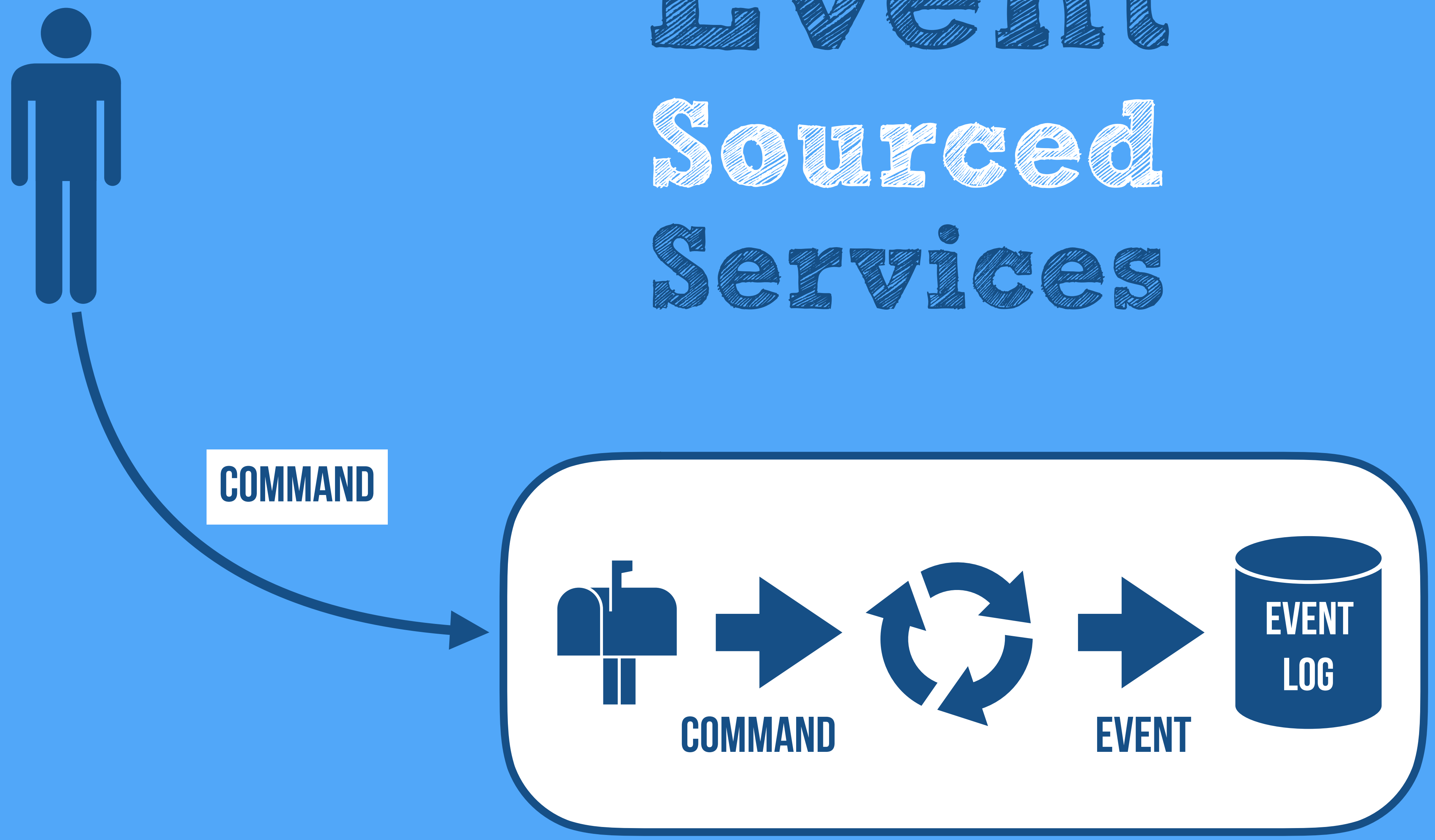
HAPPY PATH

Event Sourced Services

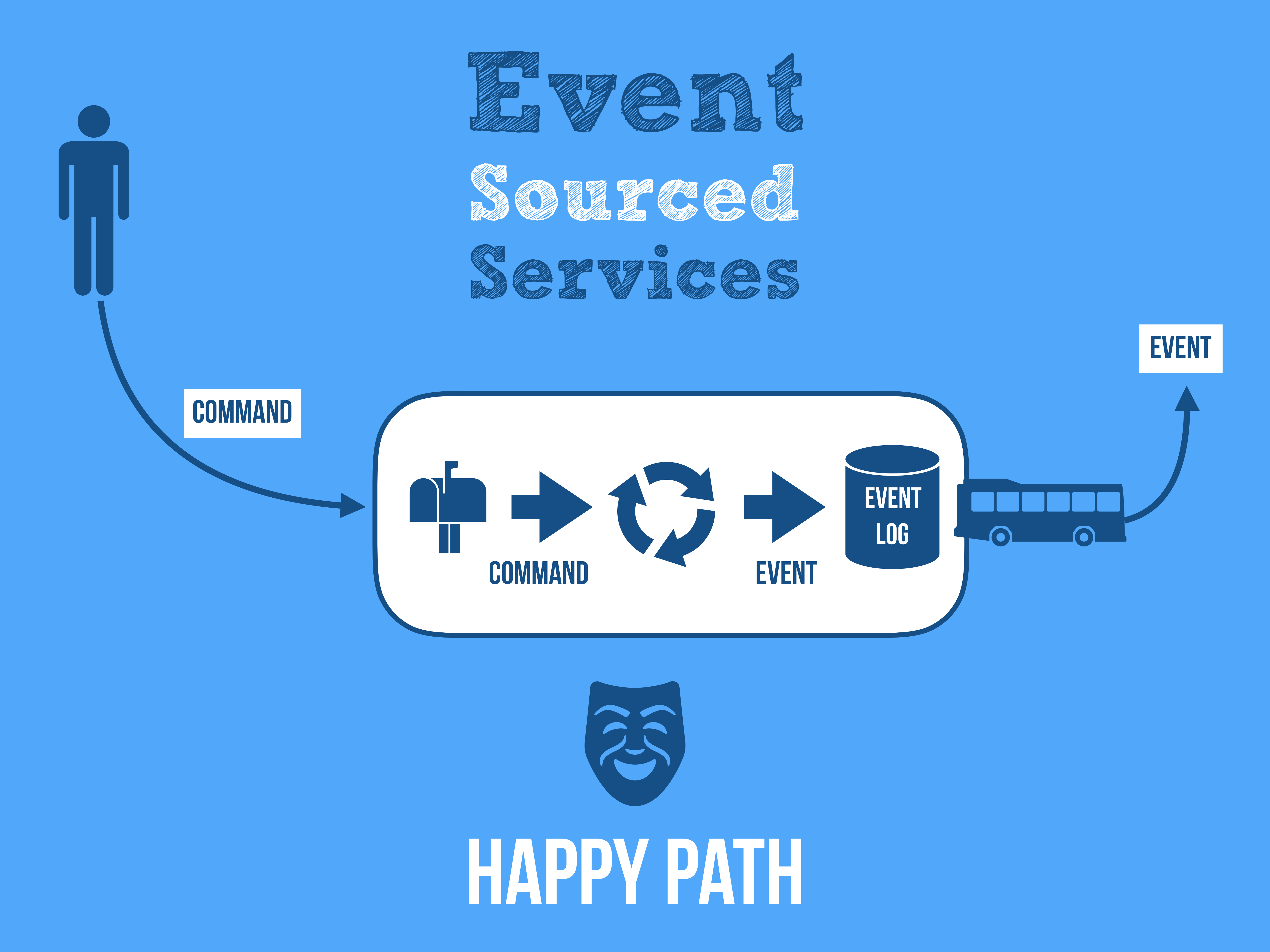


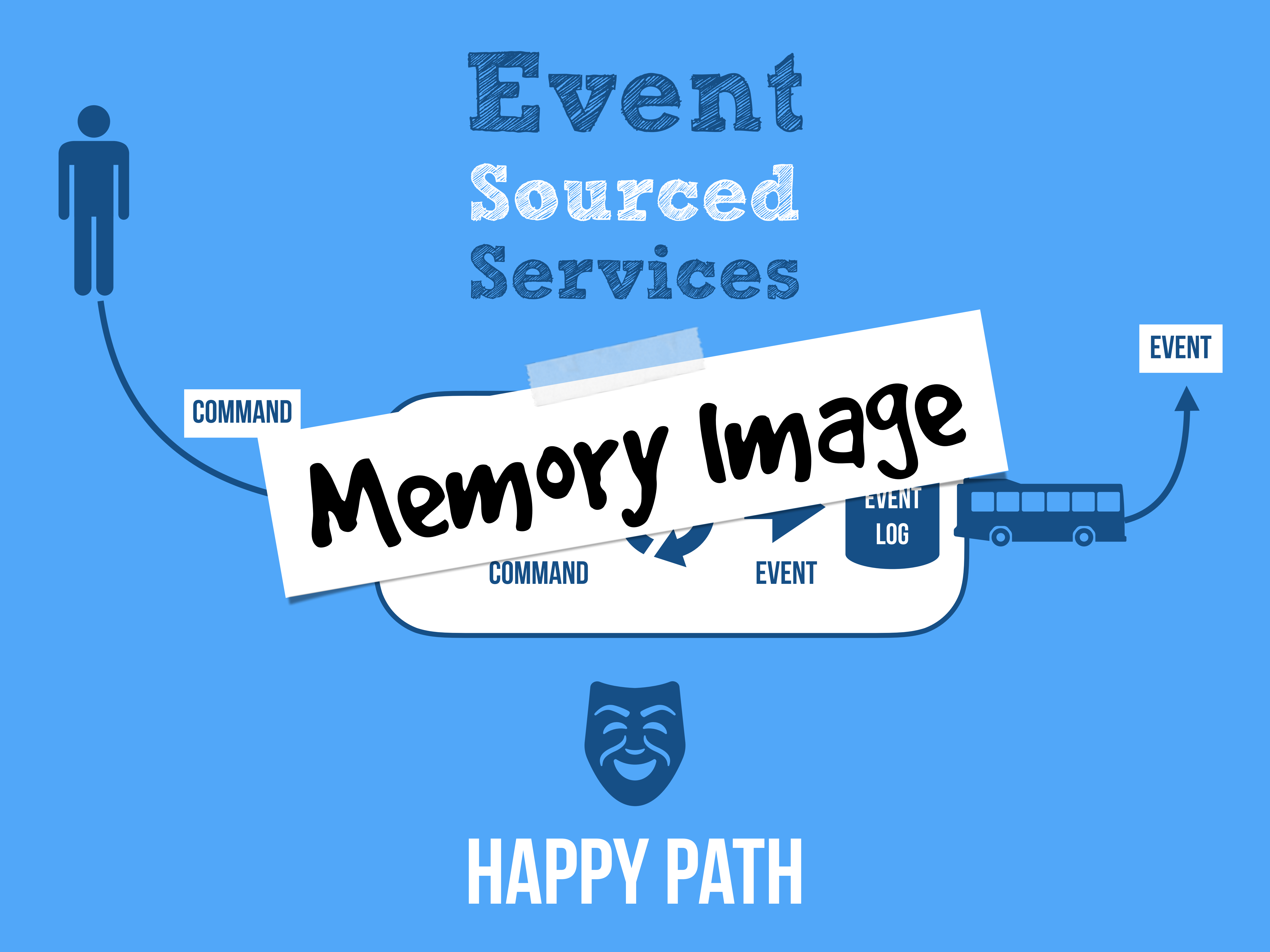
HAPPY PATH

Event Sourced Services



HAPPY PATH


Event
Sourced
Services
COMMAND
EVENT
COMMAND
EVENT
EVENT
LOG
HAPPY PATH


Event
Sourced
Services
COMMAND
Memory Image
EVENT
EVENT
LOG
COMMAND
EVENT
HAPPY PATH

Event Sourced Services

HAPPY PATH

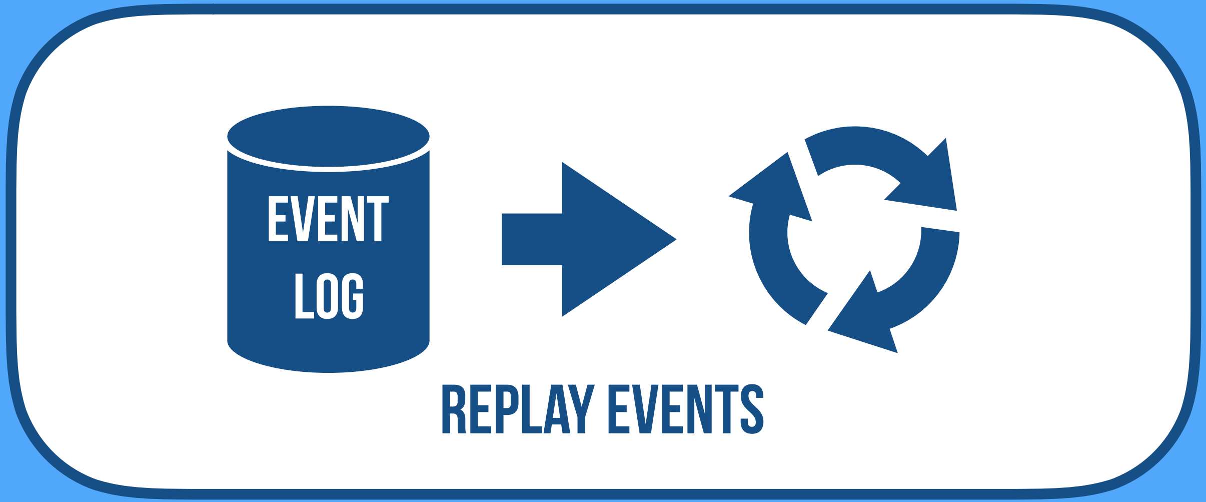
Event
Sourced
Services

SAD PATH, RECOVER FROM FAILURE

Event
Sourced
Services

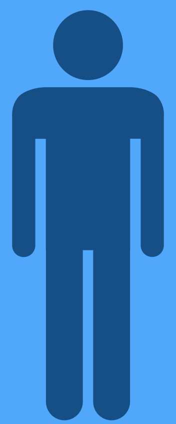
SAD PATH, RECOVER FROM FAILURE

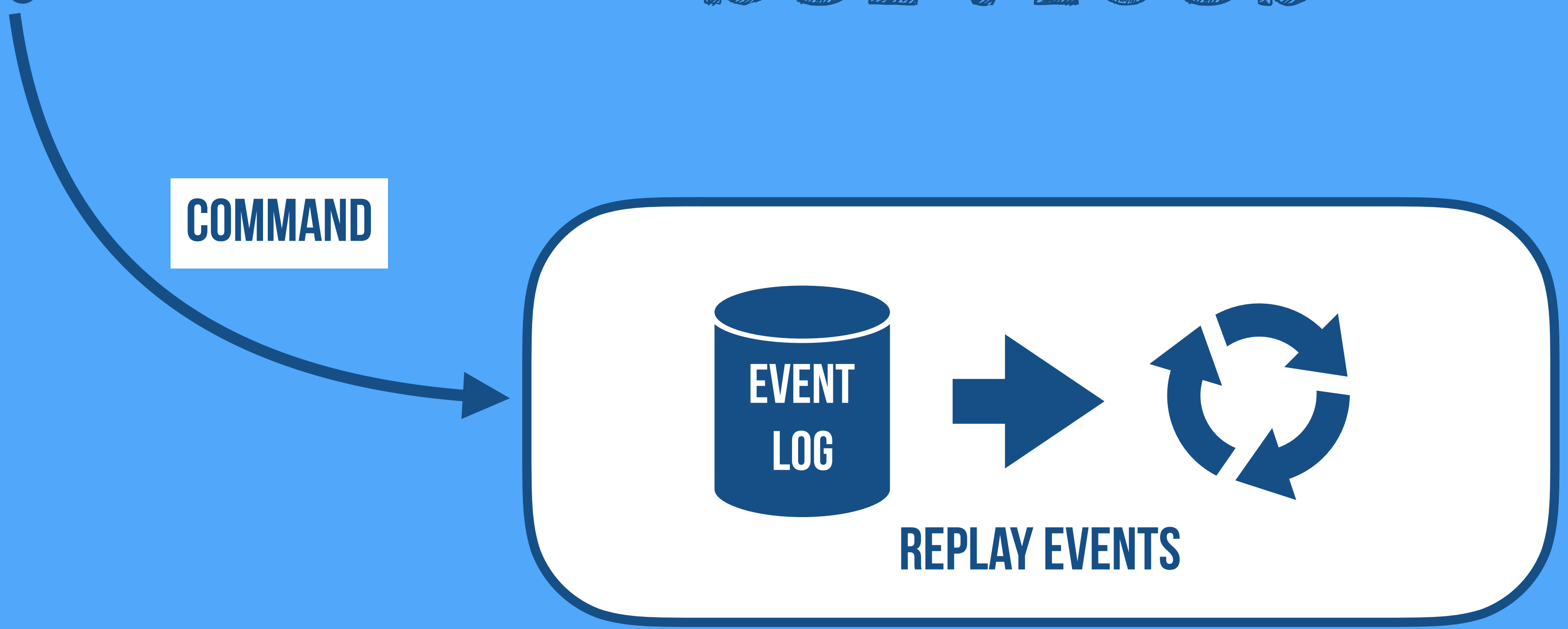
Event
Sourced
Services



SAD PATH, RECOVER FROM FAILURE

Event Sourced Services





SAD PATH, RECOVER FROM FAILURE

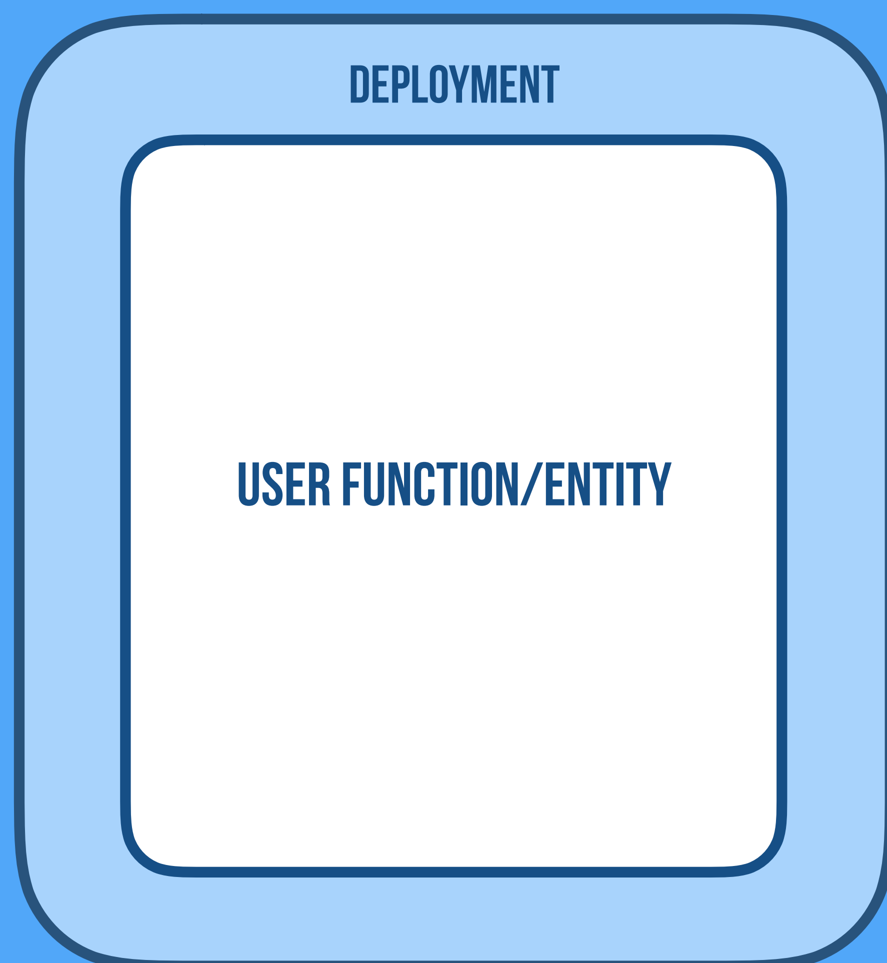
Serverless

Event Sourcing

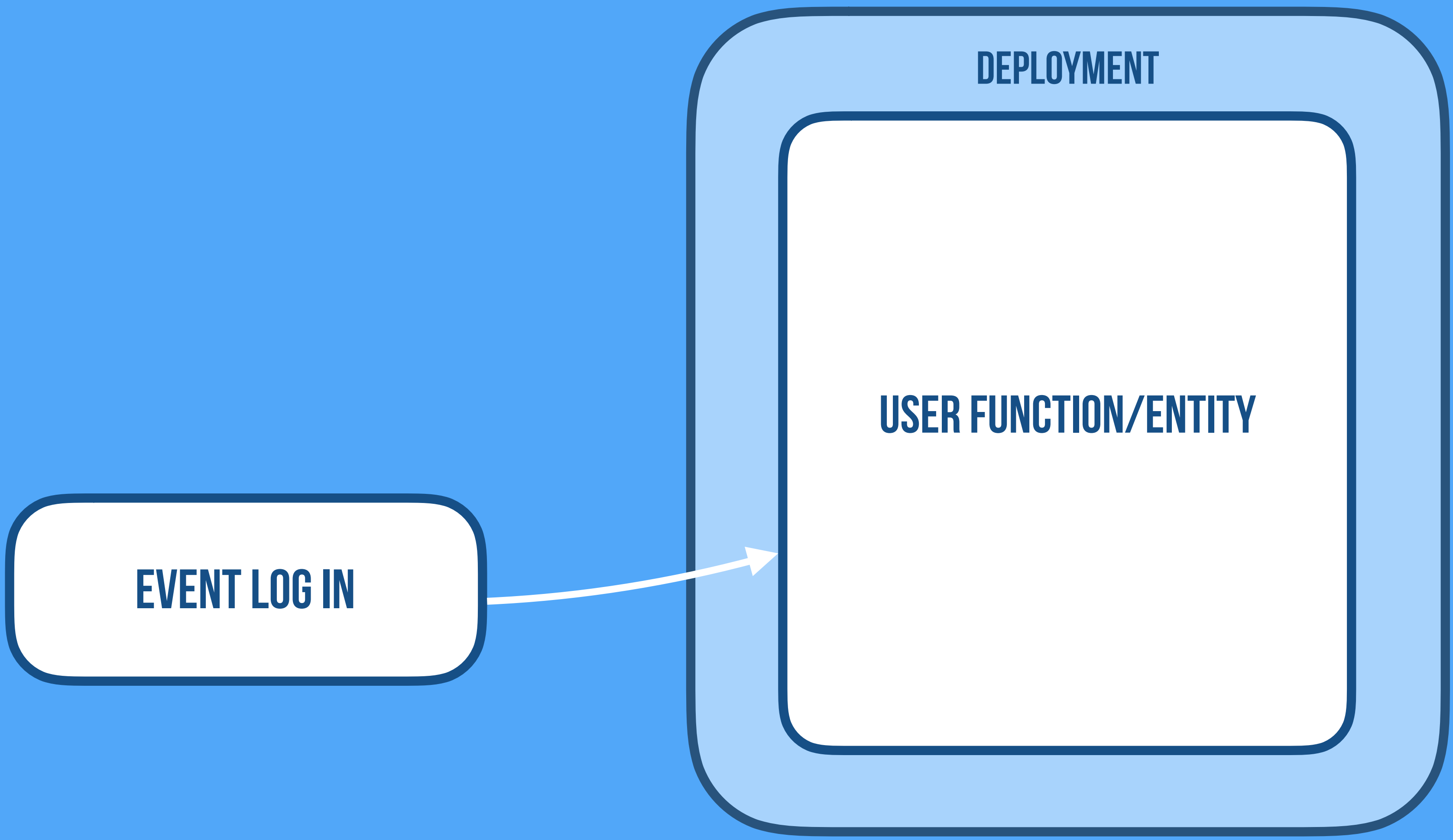
DEPLOYMENT

Serverless

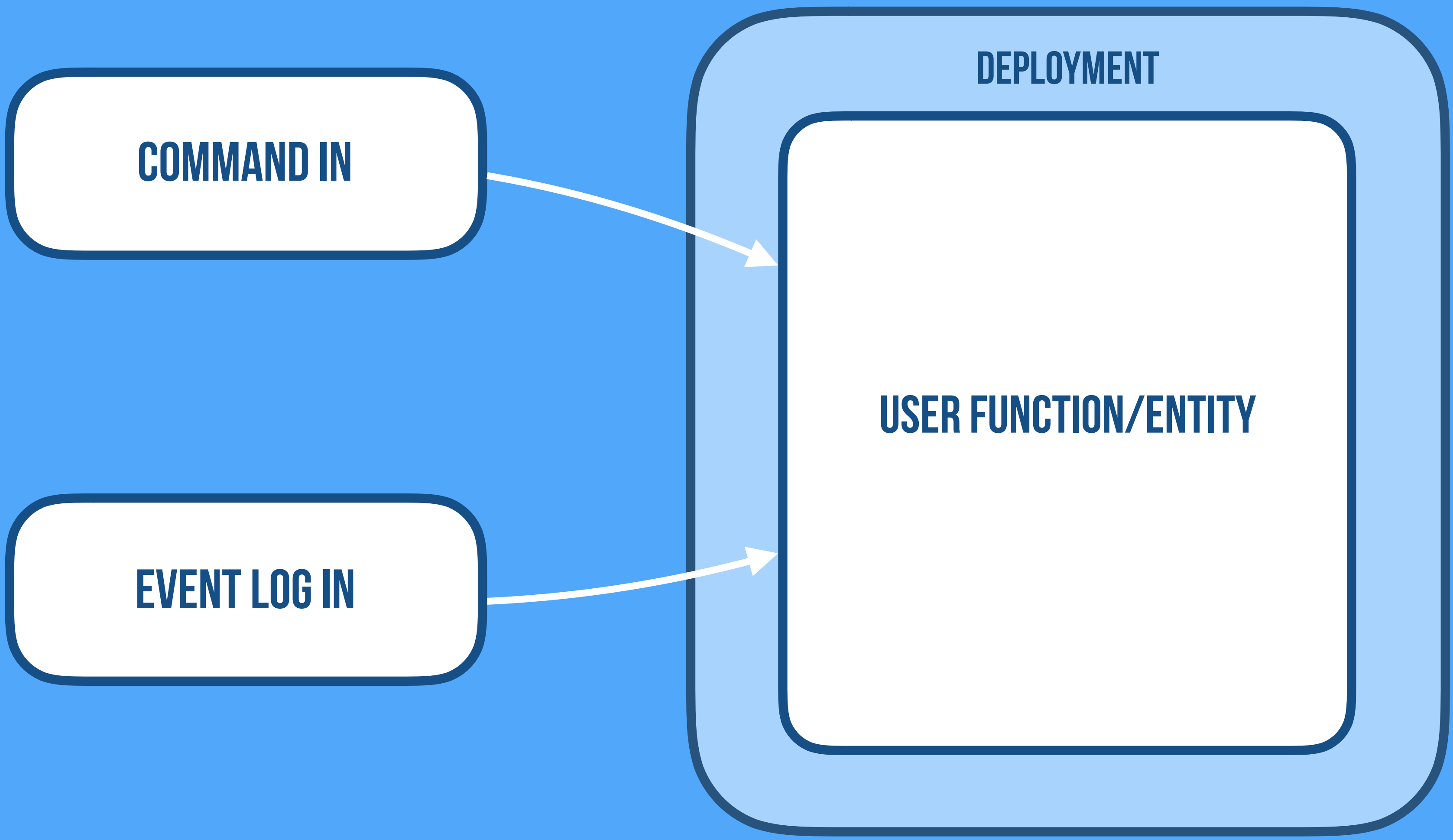
Event Sourcing



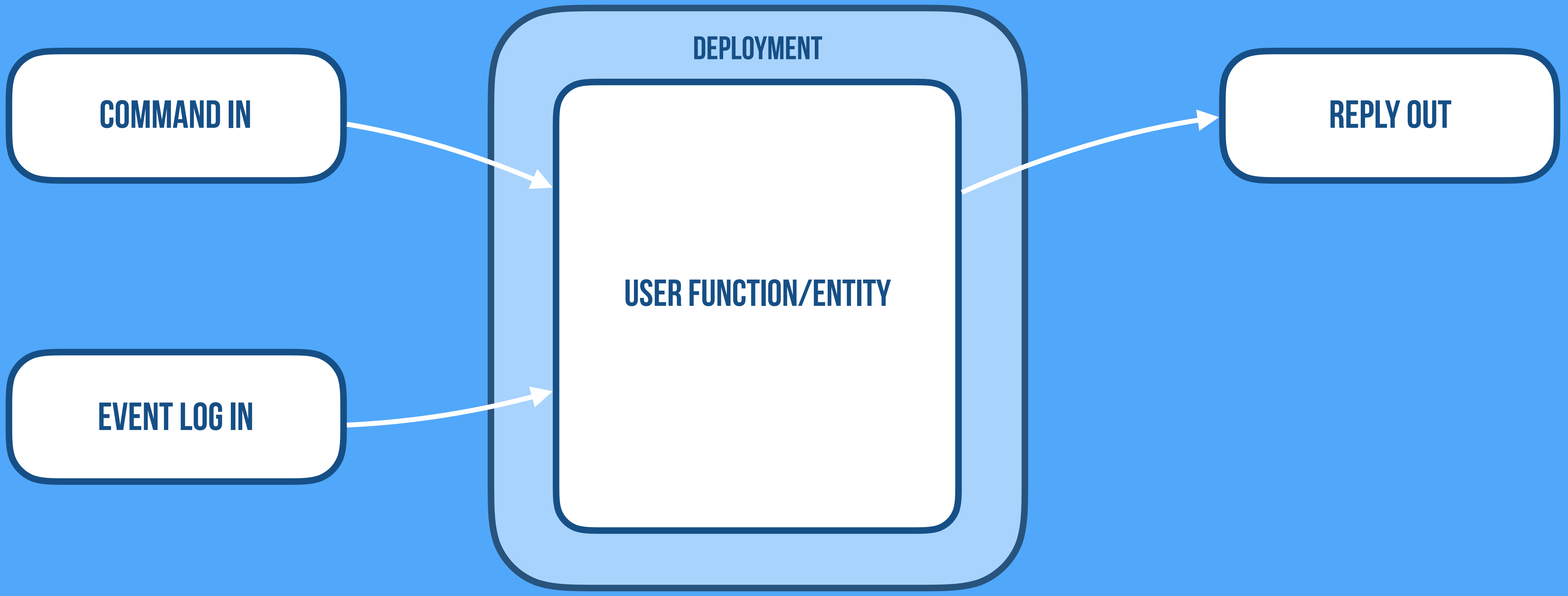
Serverless
Event Sourcing



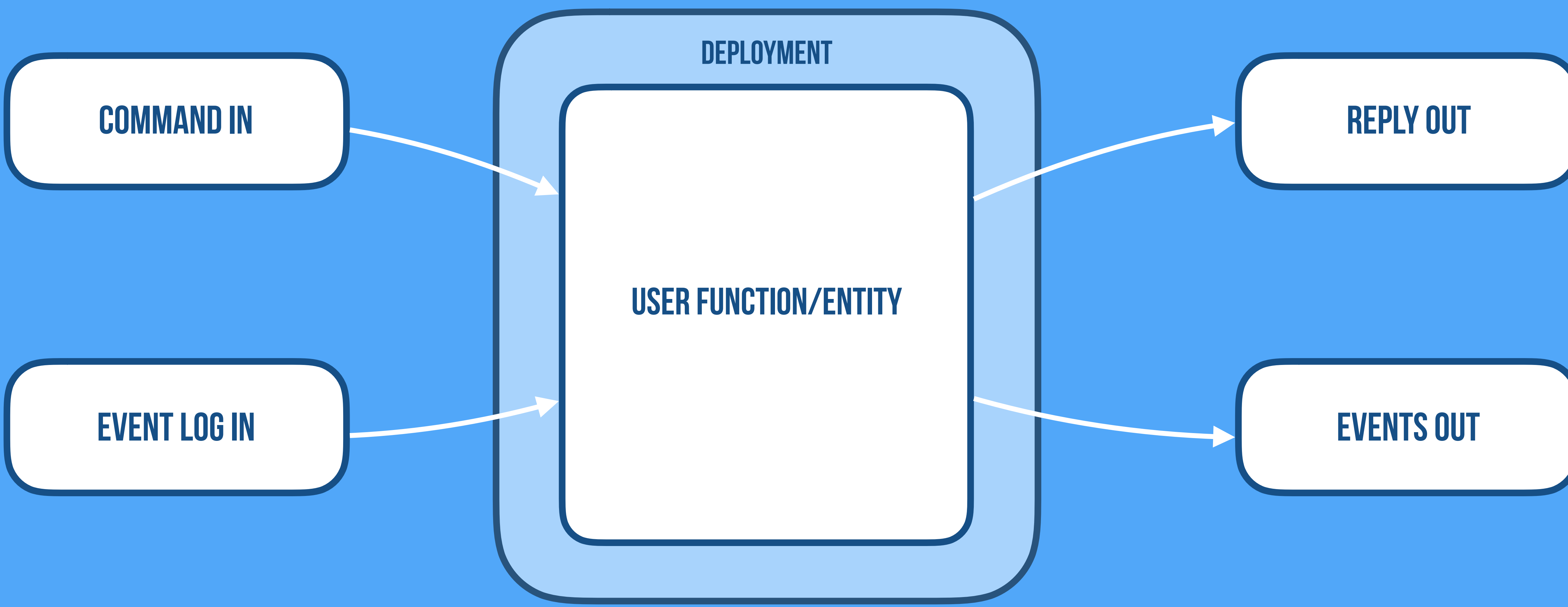
Serverless
Event Sourcing



Serverless
Event Sourcing



Serverless Event Sourcing



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

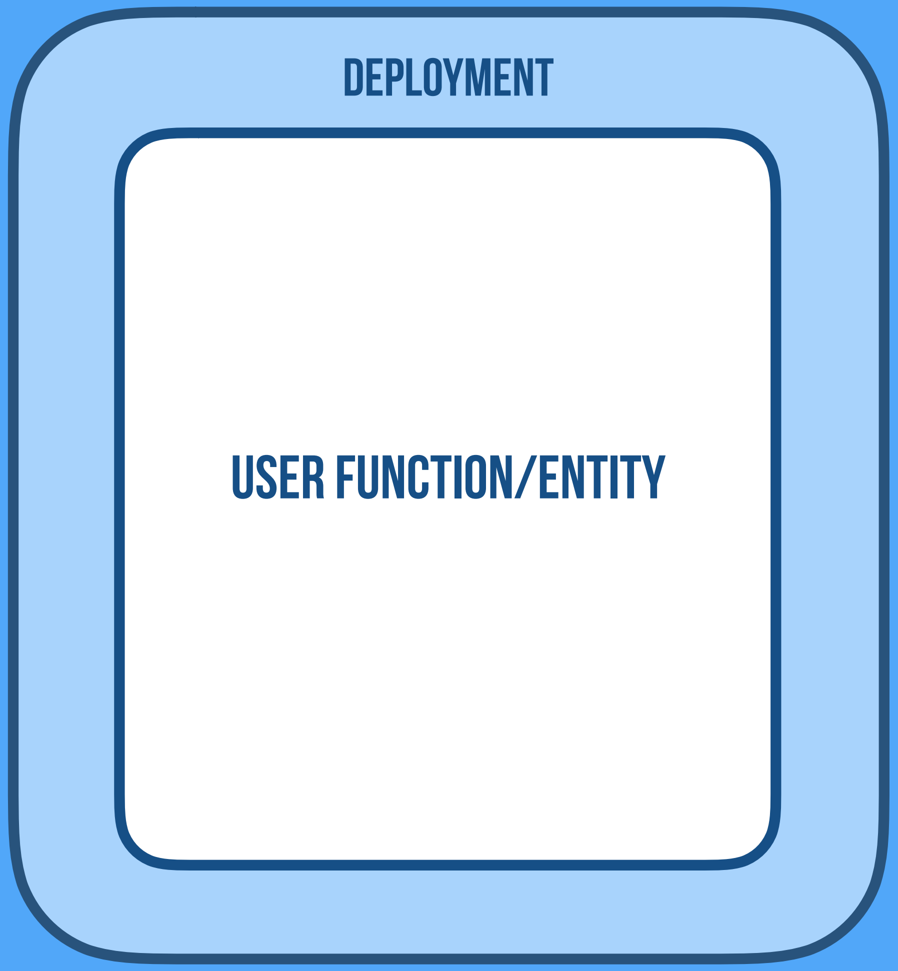
Counters
Registers
Sets
Maps
Graphs
(that all compose)

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

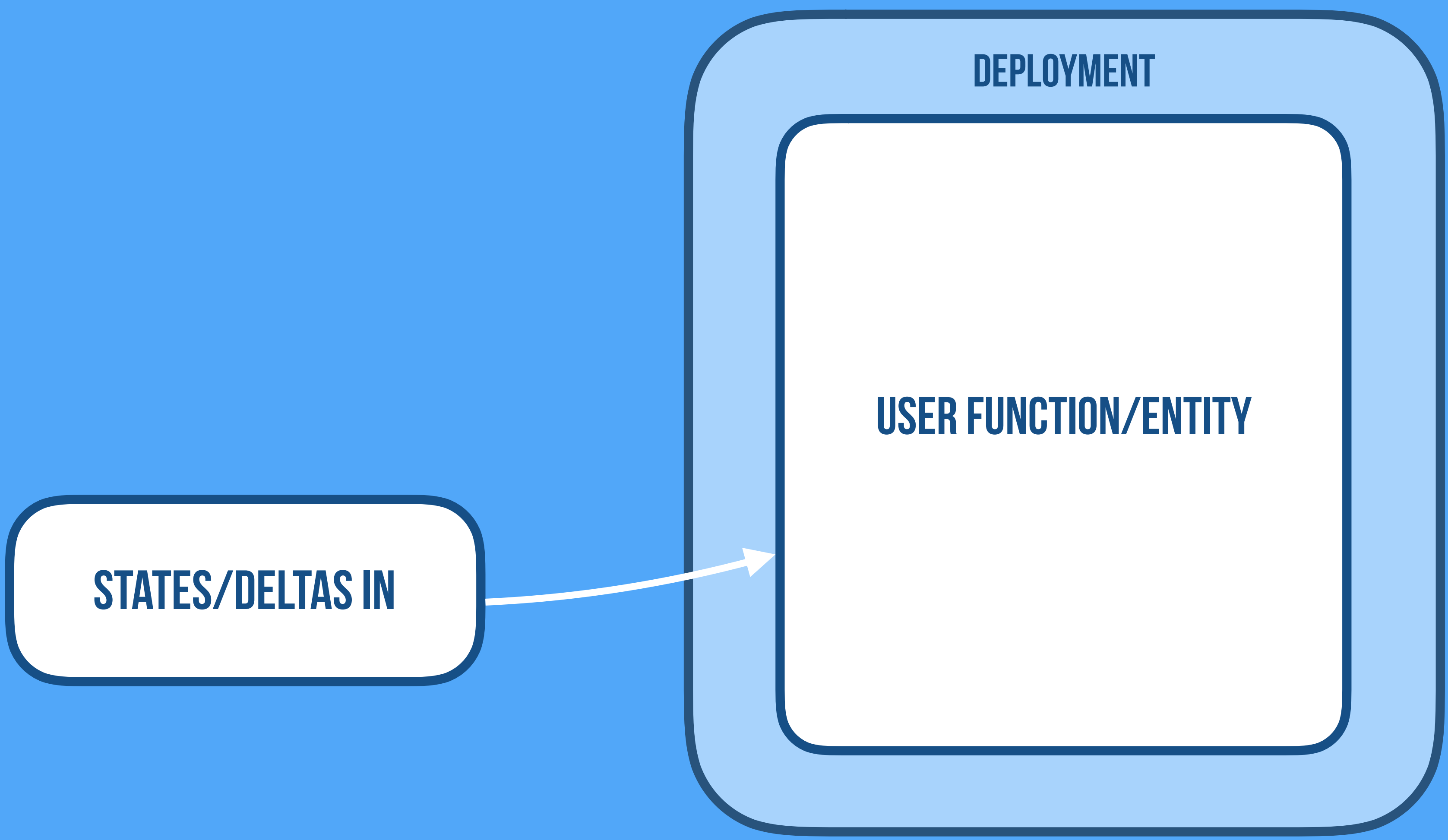
Serverless CRDTs

DEPLOYMENT

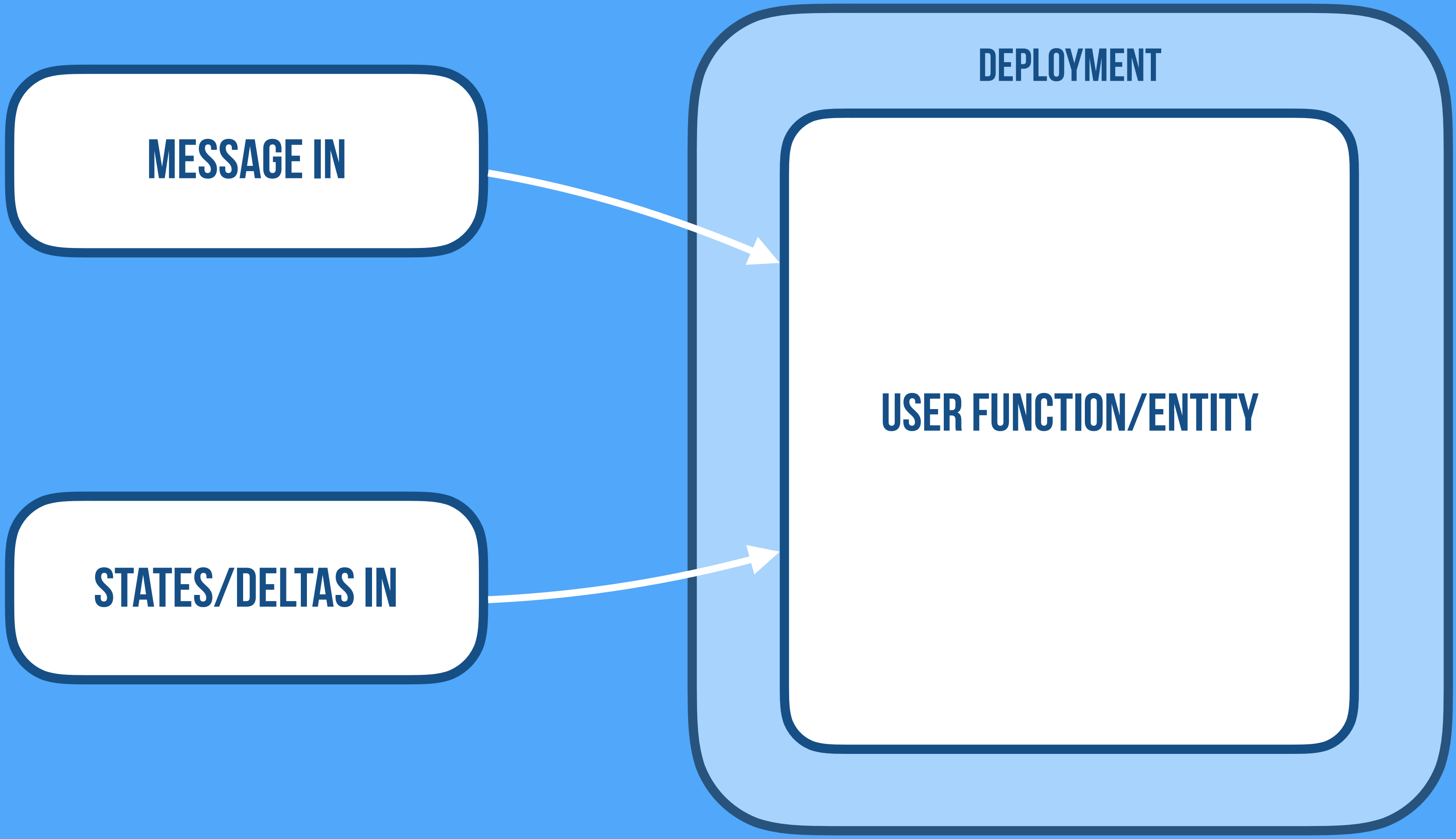
Serverless CRDTs



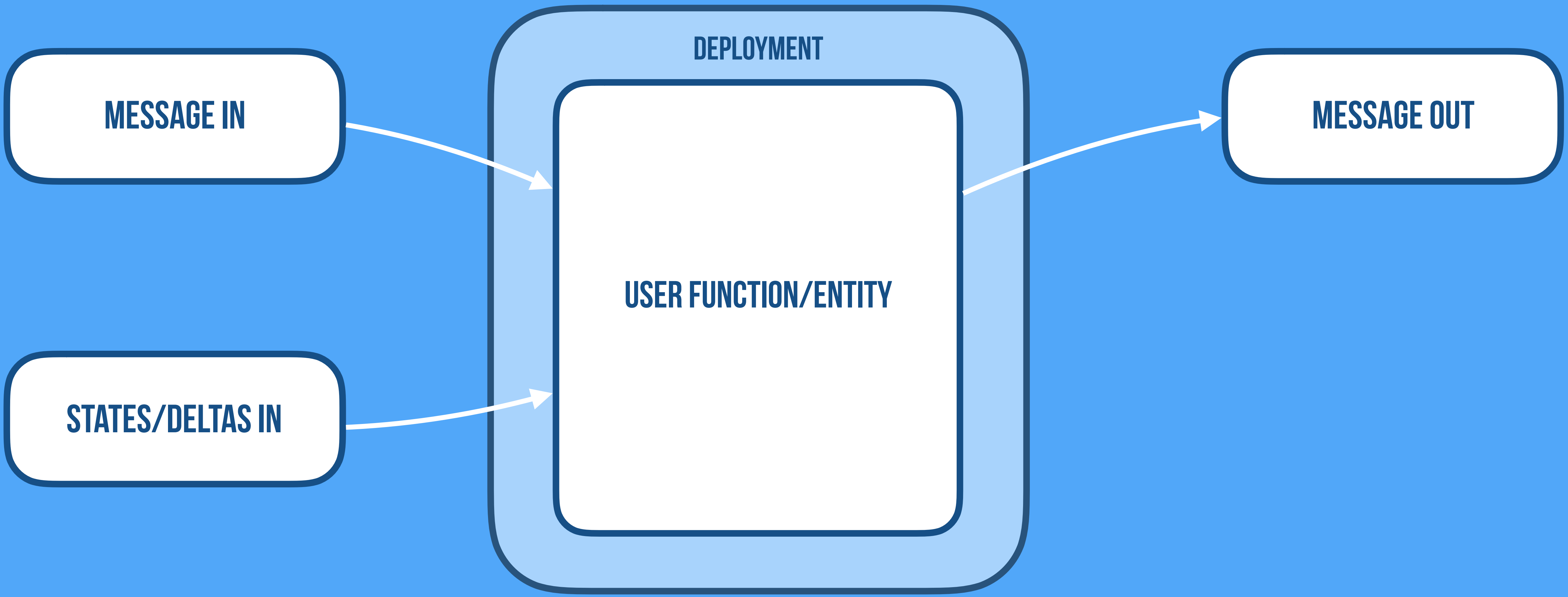
Serverless CRDTs



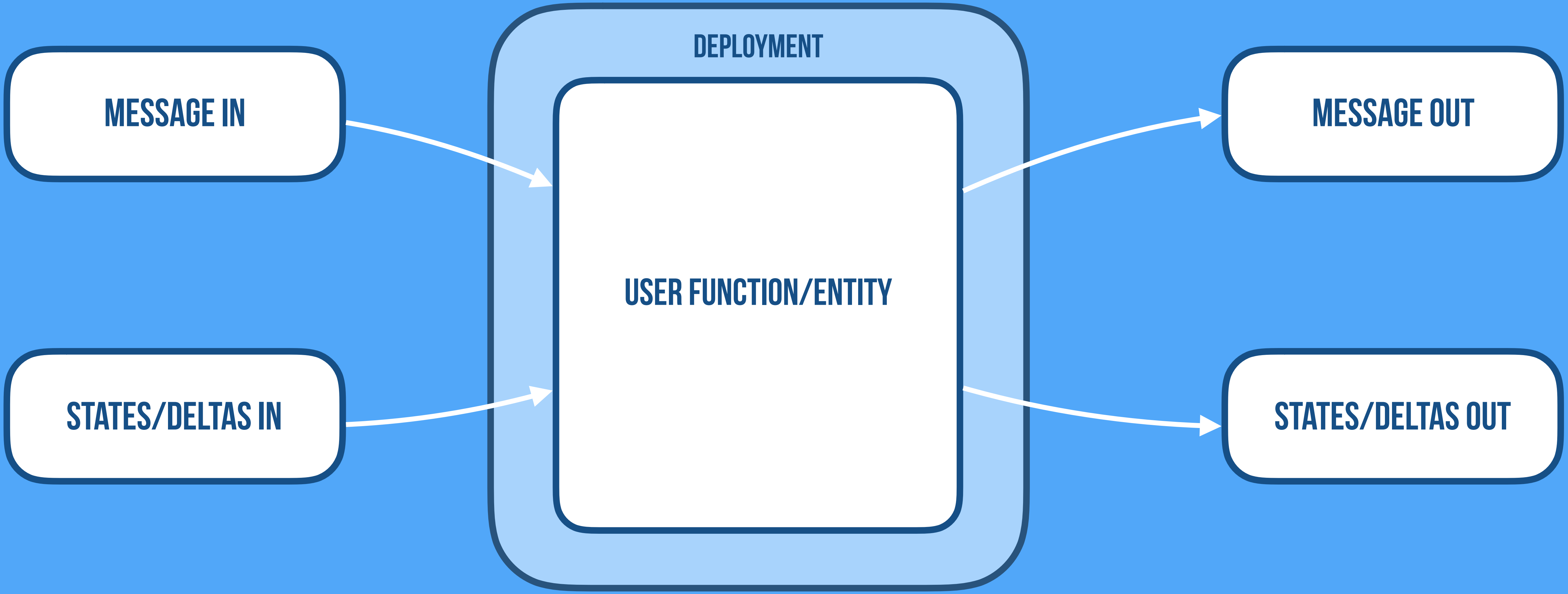
Serverless CRDTs



Serverless CRDTs



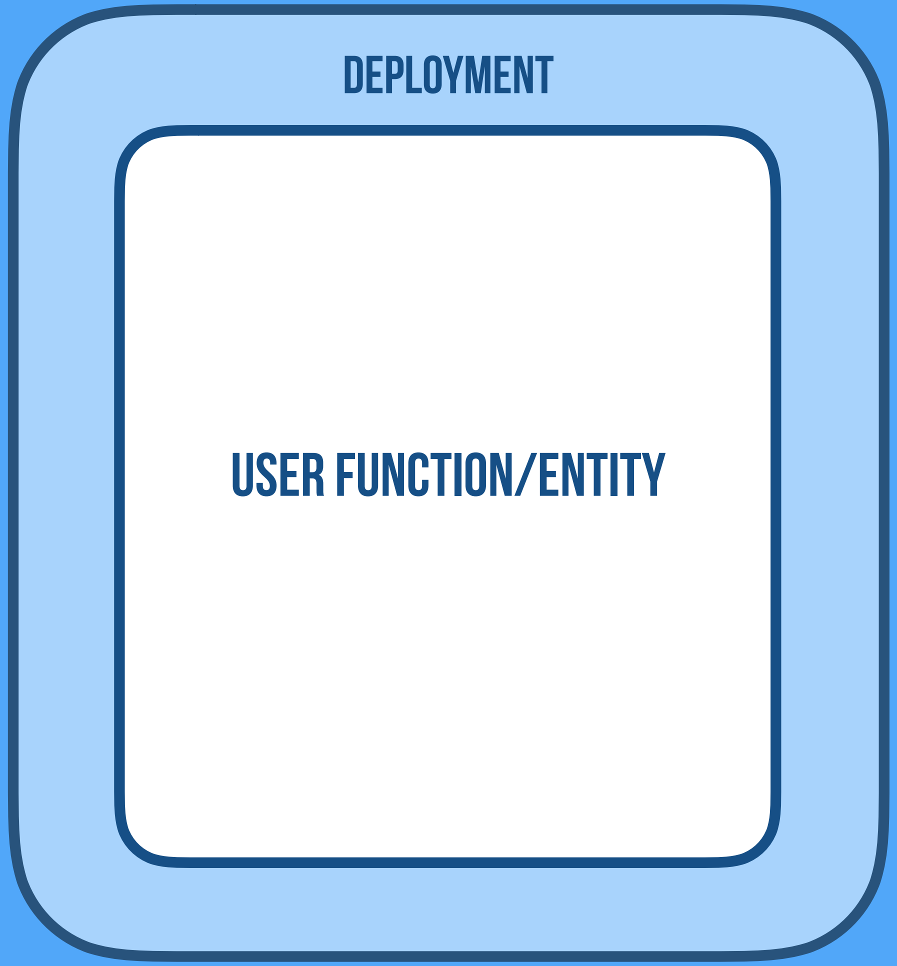
Serverless CRDTs



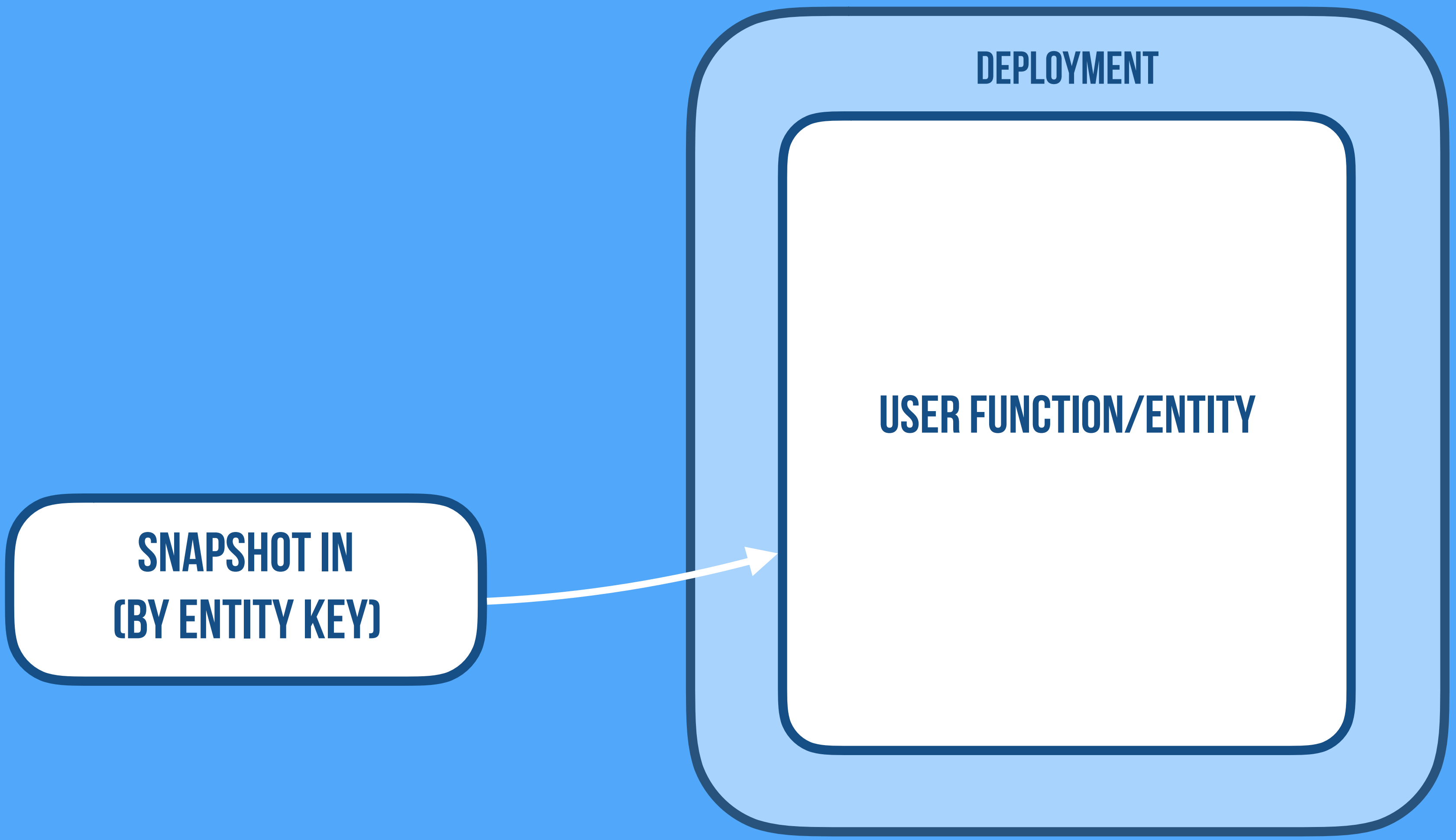
Serverless CRUD

DEPLOYMENT

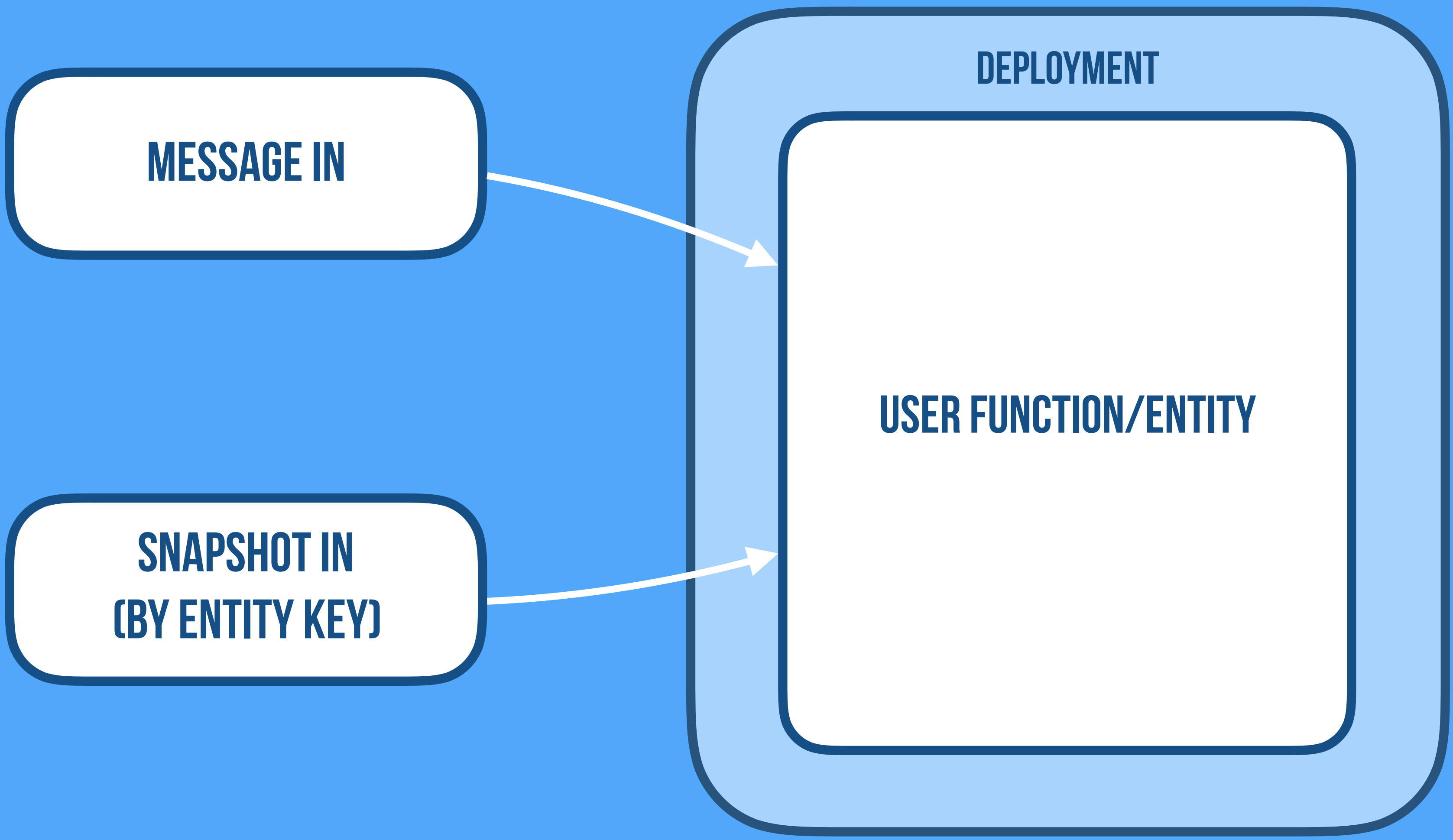
Serverless CRUD



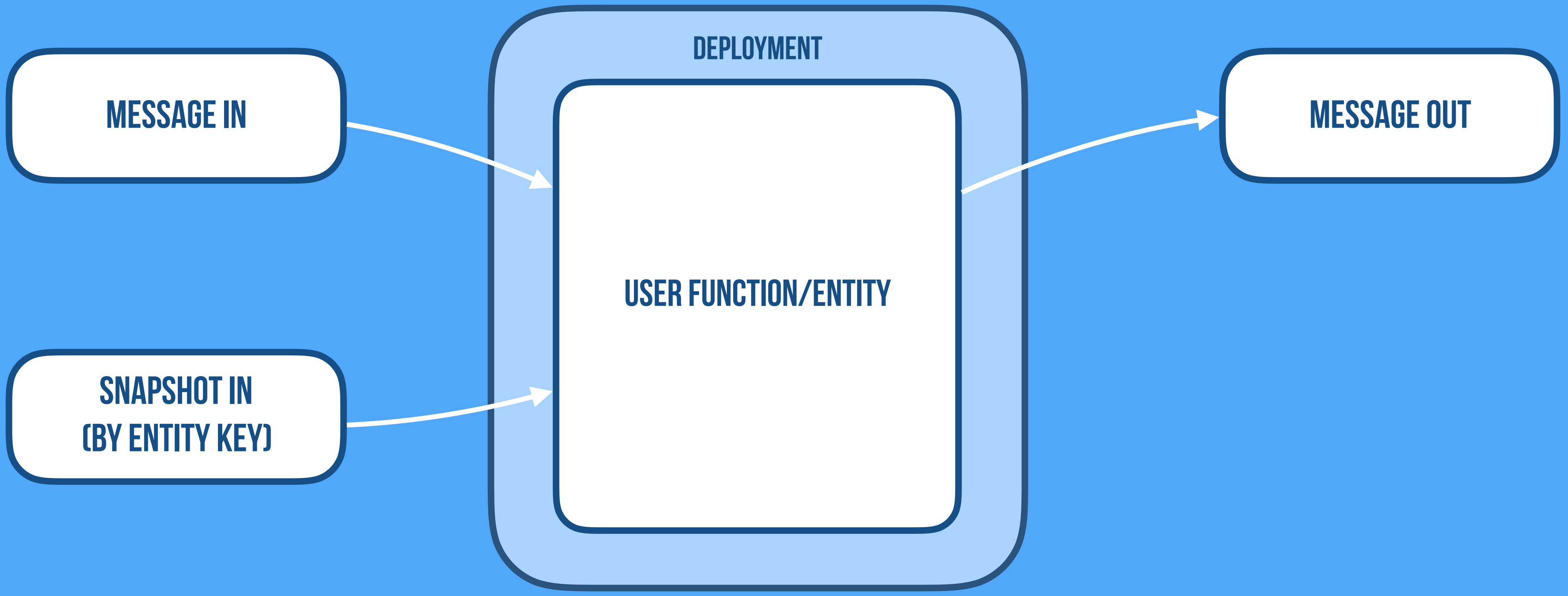
Serverless CRUD



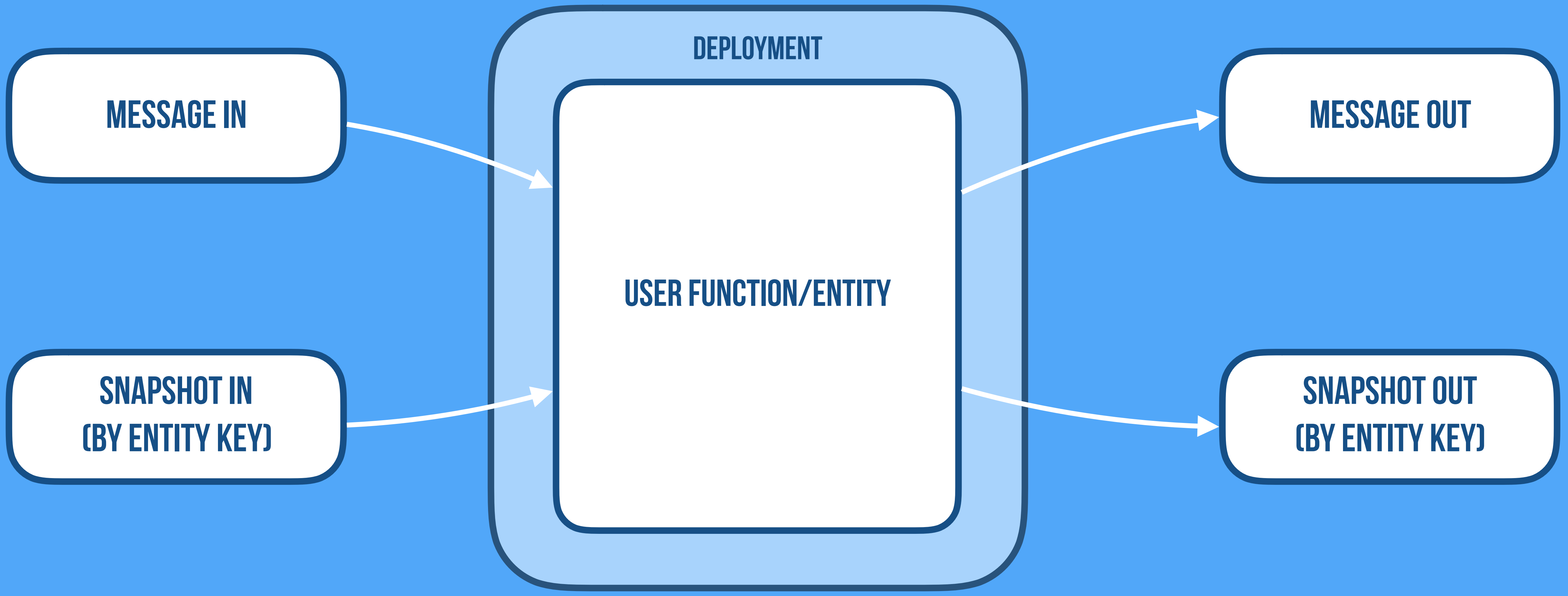
Serverless
CRUD



Serverless CRUD



Serverless CRUD



Introducing

cloudstate

WHAT IS CLOUDSTATE?

https://cloudstate.io

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

- Polyglot: Client libs in JavaScript, Java, Go—Python, .NET, Swift, Scala in the works

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

- Polyglot: Client libs in JavaScript, Java, Go—Python, .NET, Swift, Scala in the works
- PolyState: Powerful state model—support for Event Sourcing, CRDTs, Key Value

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

- Polyglot: Client libs in JavaScript, Java, Go—Python, .NET, Swift, Scala in the works
- PolyState: Powerful state model—support for Event Sourcing, CRDTs, Key Value
- PolyDB: Supporting SQL, NoSQL, NewSQL, and in-memory replication

WHAT IS CLOUDSTATE?

https://cloudstate.io

Two things:

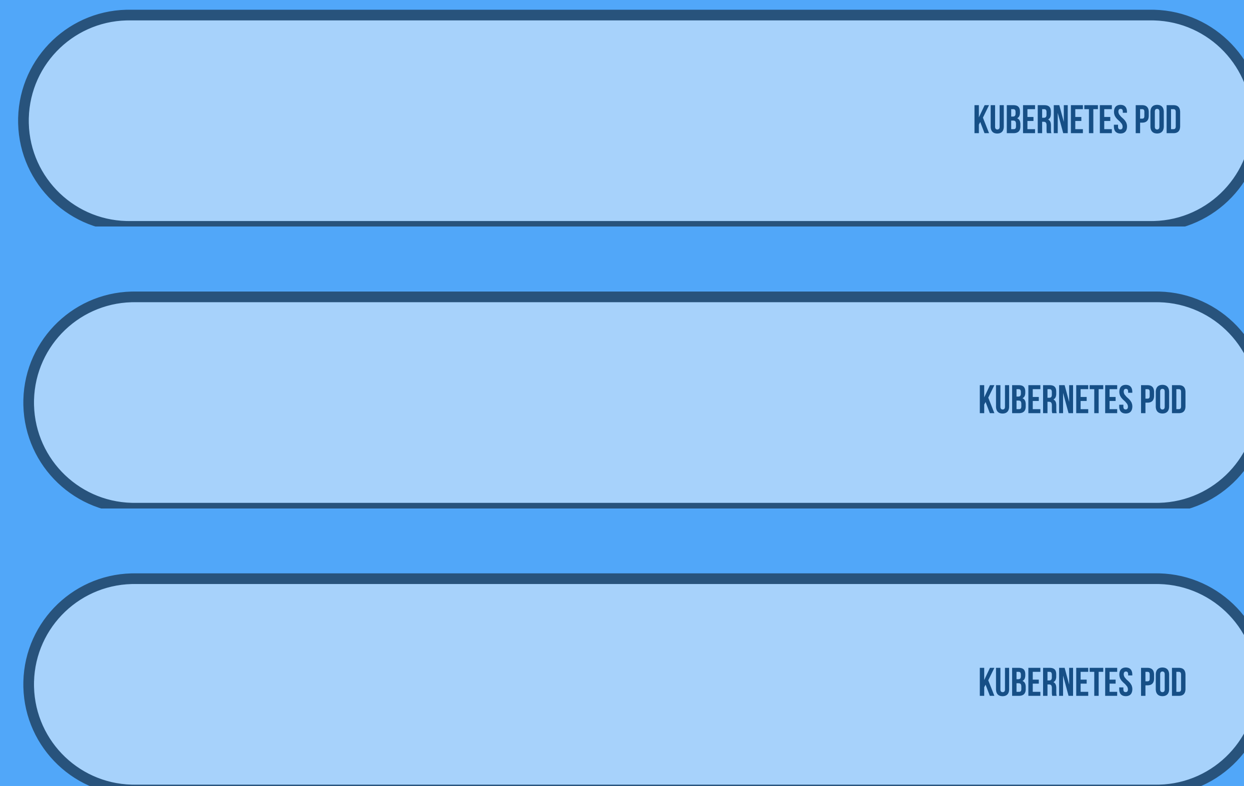
1. Standards Project—defining a specification, protocol, and TCK
2. Reference Implementation—backend + client APIs in different languages

Highlights:

- Polyglot: Client libs in JavaScript, Java, Go—Python, .NET, Swift, Scala in the works
- PolyState: Powerful state model—support for Event Sourcing, CRDTs, Key Value
- PolyDB: Supporting SQL, NoSQL, NewSQL, and in-memory replication
- Open Source leveraging Akka, gRPC, Knative, GraalVM, running on Kubernetes

SERVING OF STATEFUL FUNCTIONS

SERVING OF STATEFUL FUNCTIONS

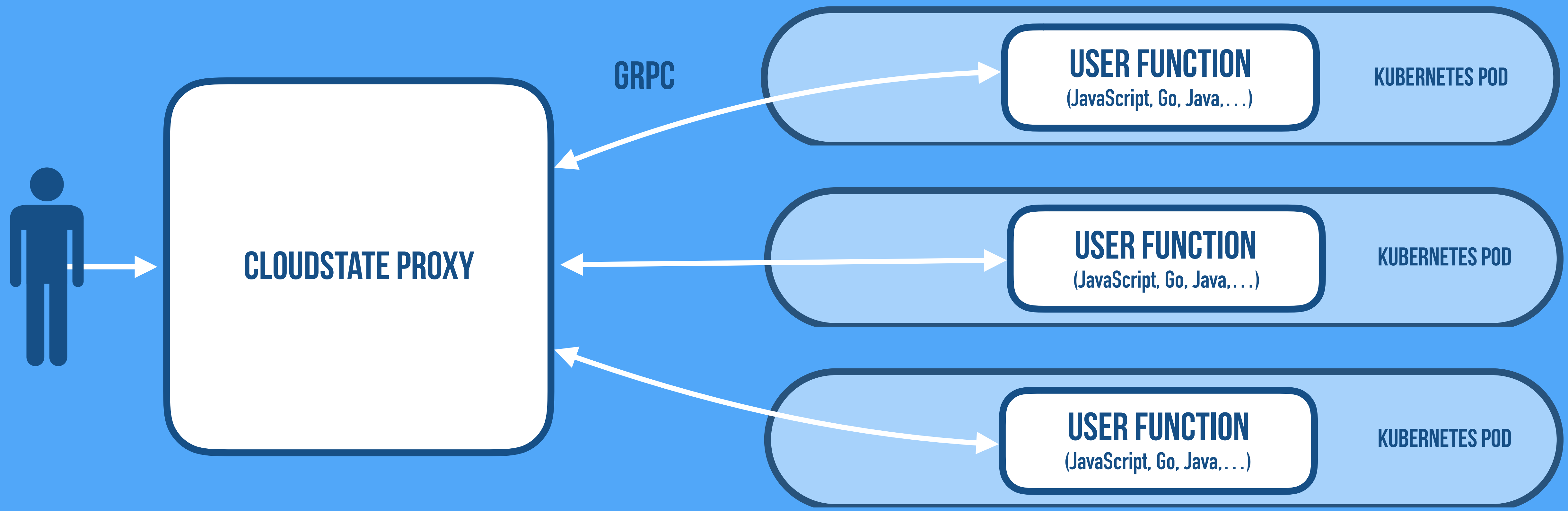


SERVING OF STATEFUL FUNCTIONS

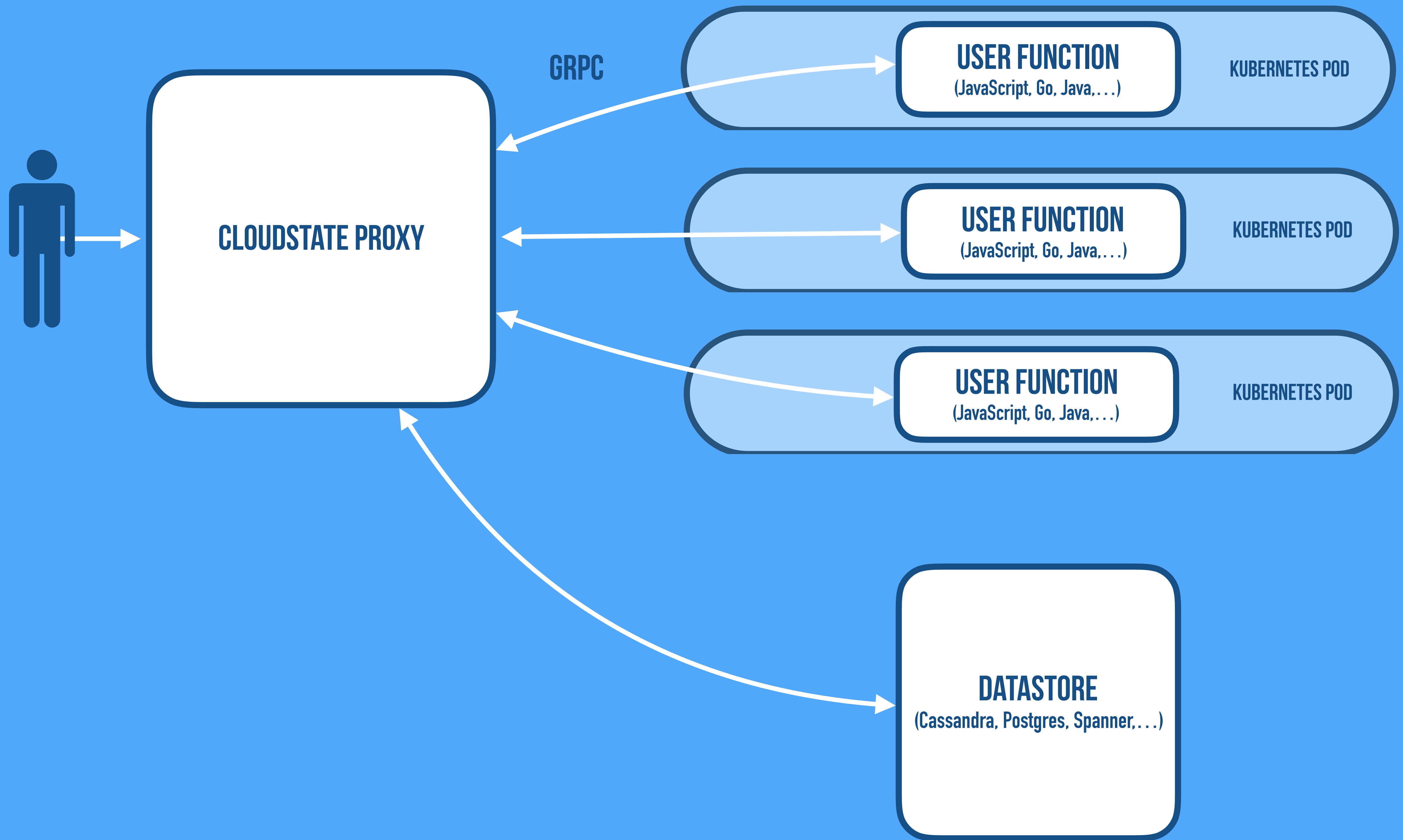
SERVING OF STATEFUL FUNCTIONS

SERVING OF STATEFUL FUNCTIONS

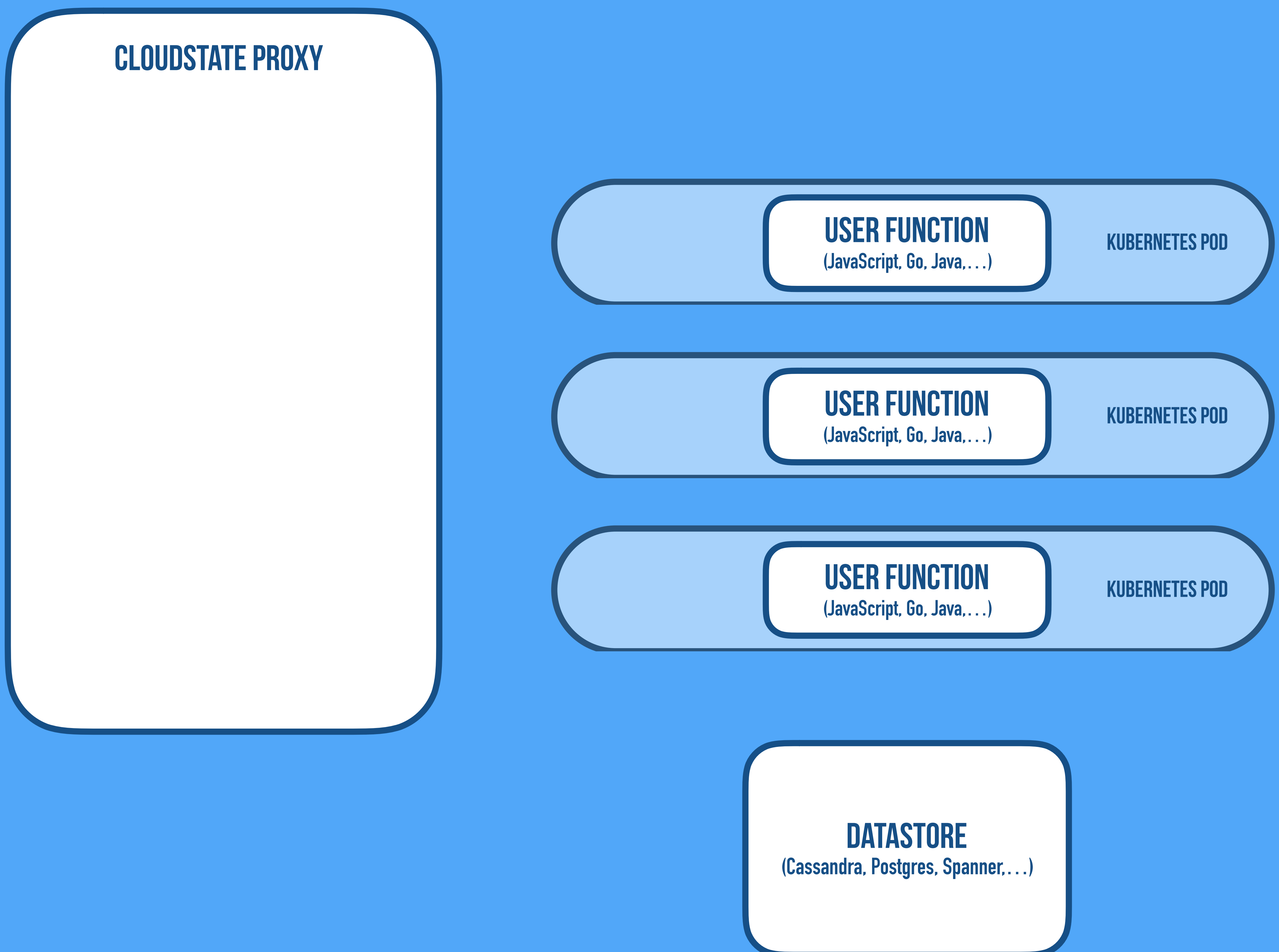
SERVING OF STATEFUL FUNCTIONS



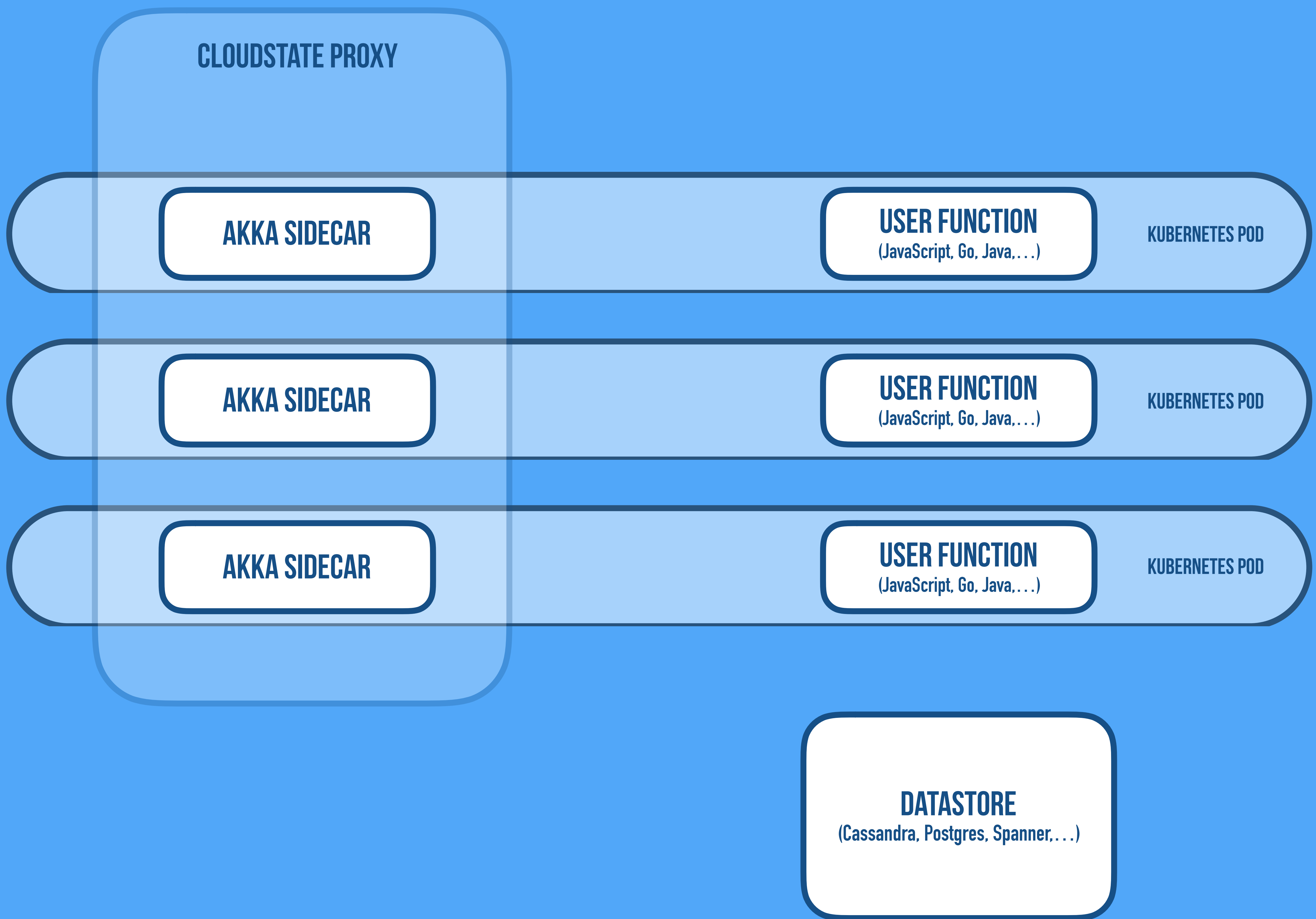
SERVING OF STATEFUL FUNCTIONS



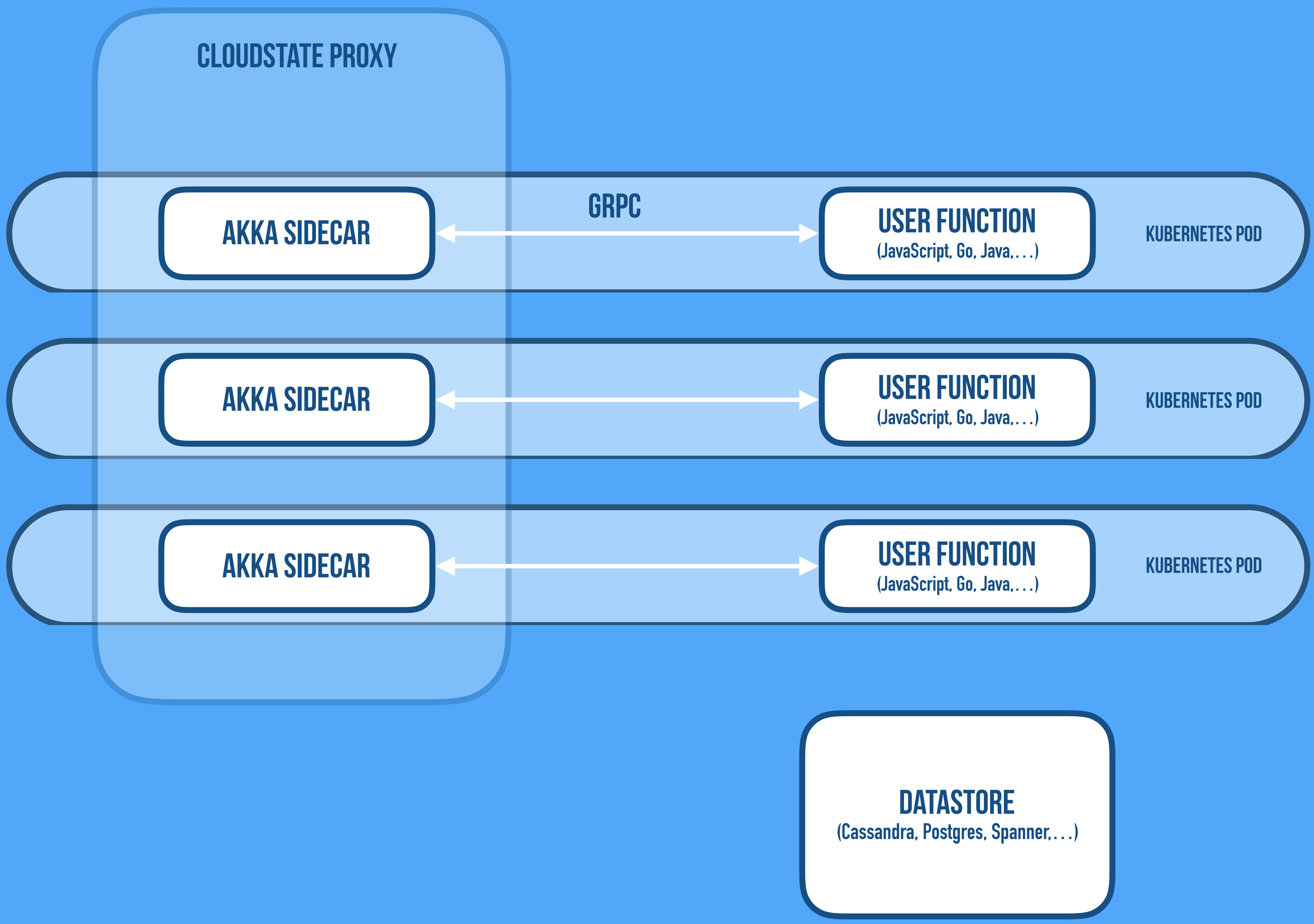
POWERED BY AKKA CLUSTER SIDECARS



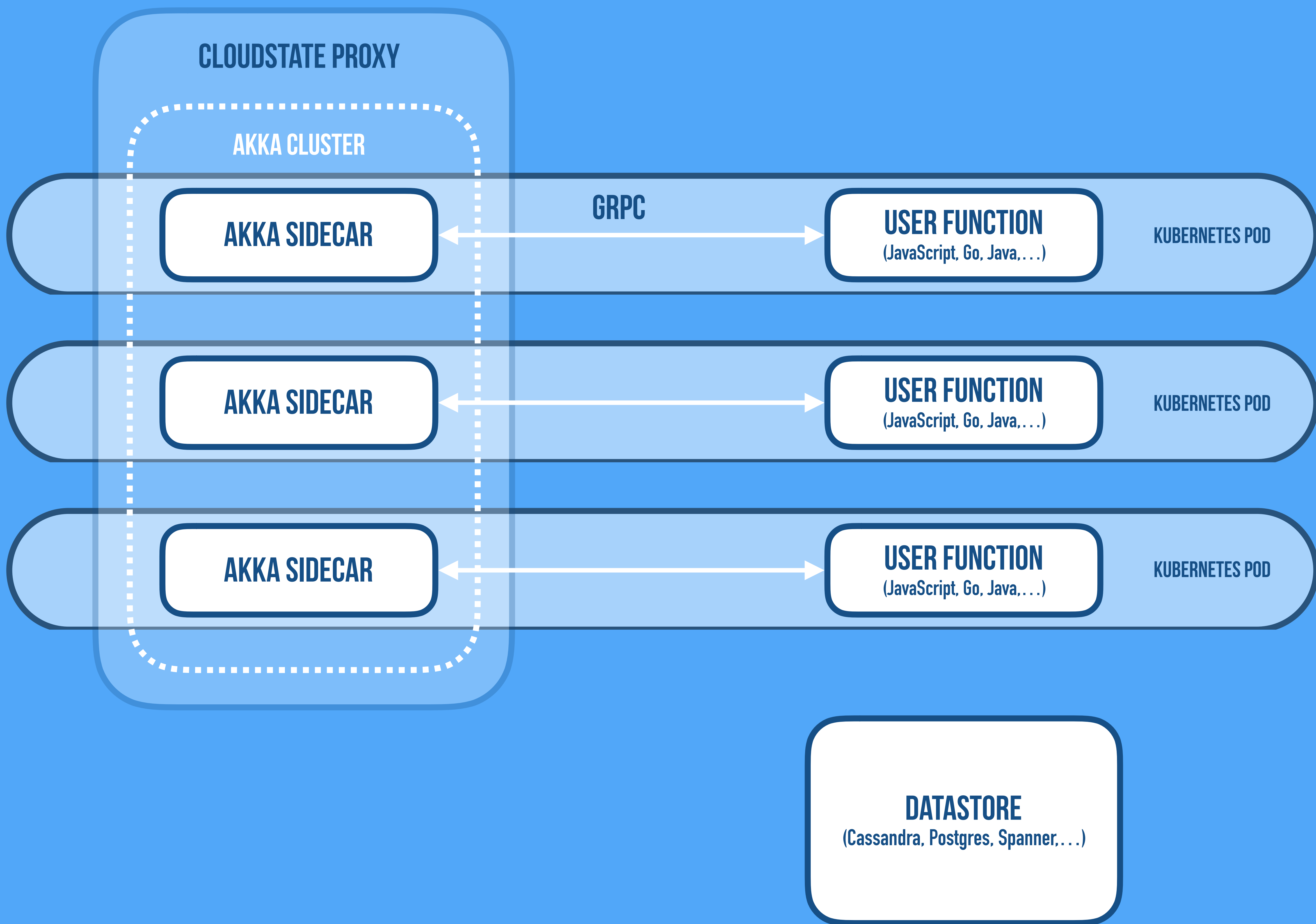
POWERED BY AKKA CLUSTER SIDECARS



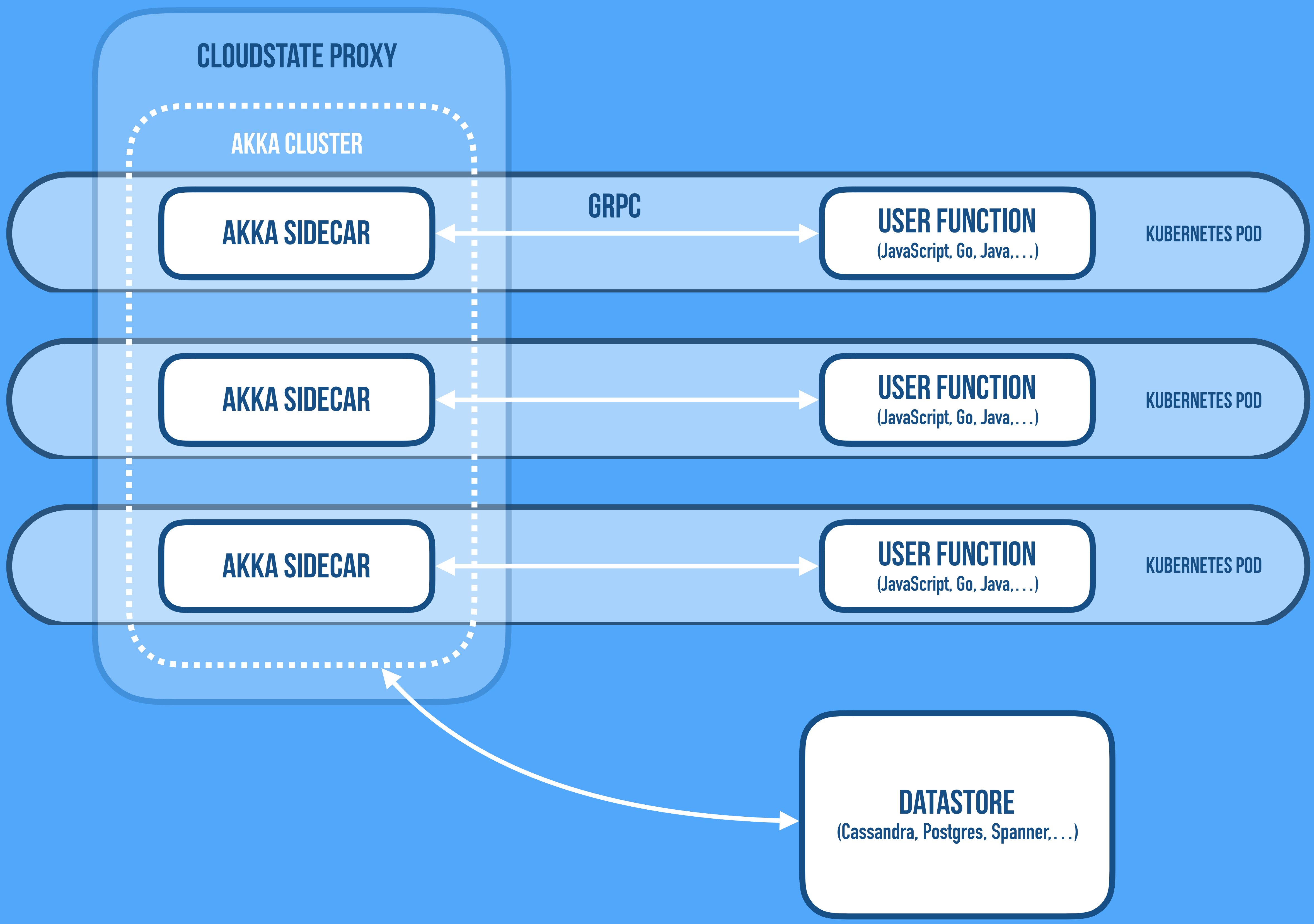
POWERED BY AKKA CLUSTER SIDECARS



POWERED BY AKKA CLUSTER SIDECARS



POWERED BY AKKA CLUSTER SIDECARS

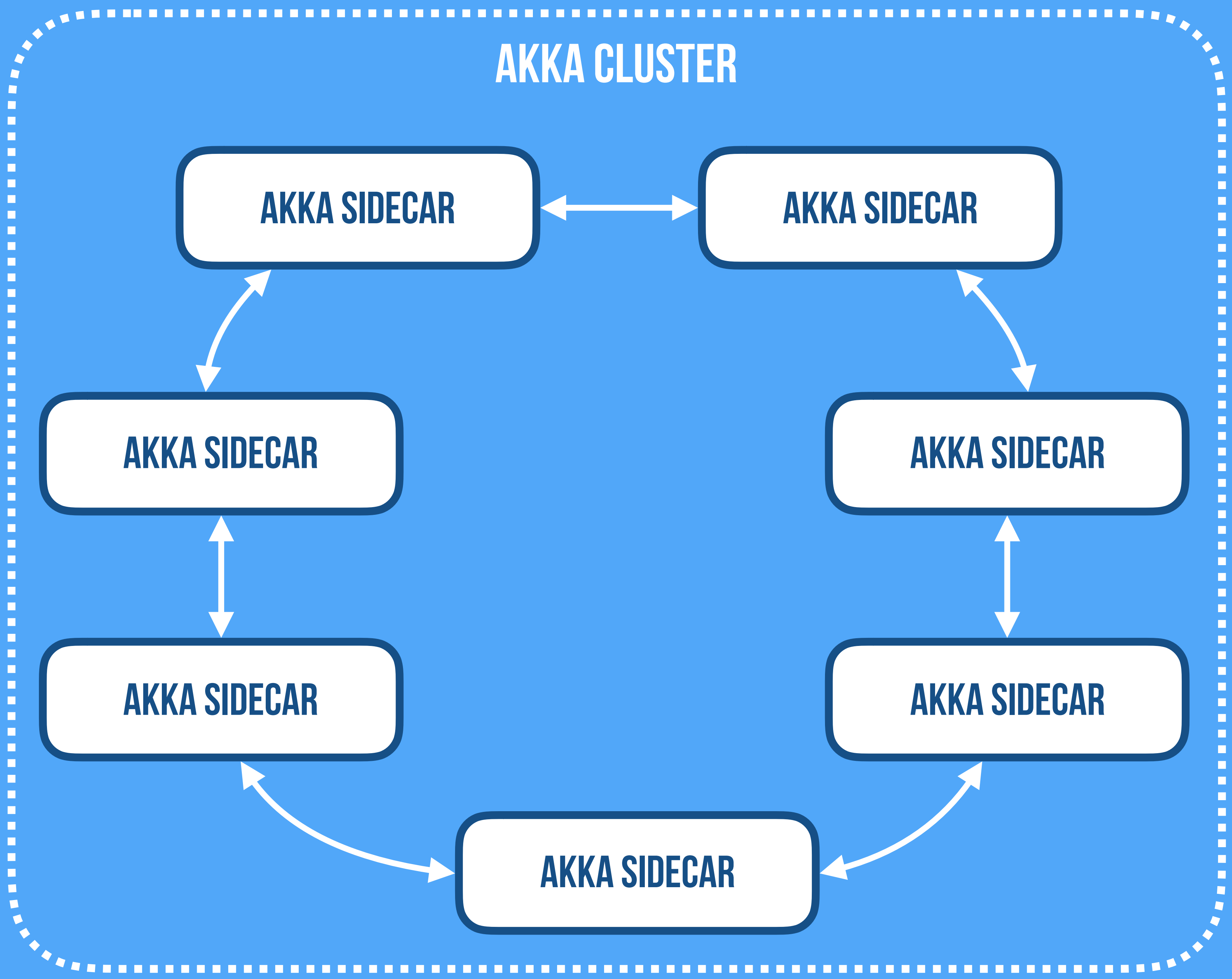


AKKA CLUSTER STATE MANAGEMENT

AKKA CLUSTER STATE MANAGEMENT

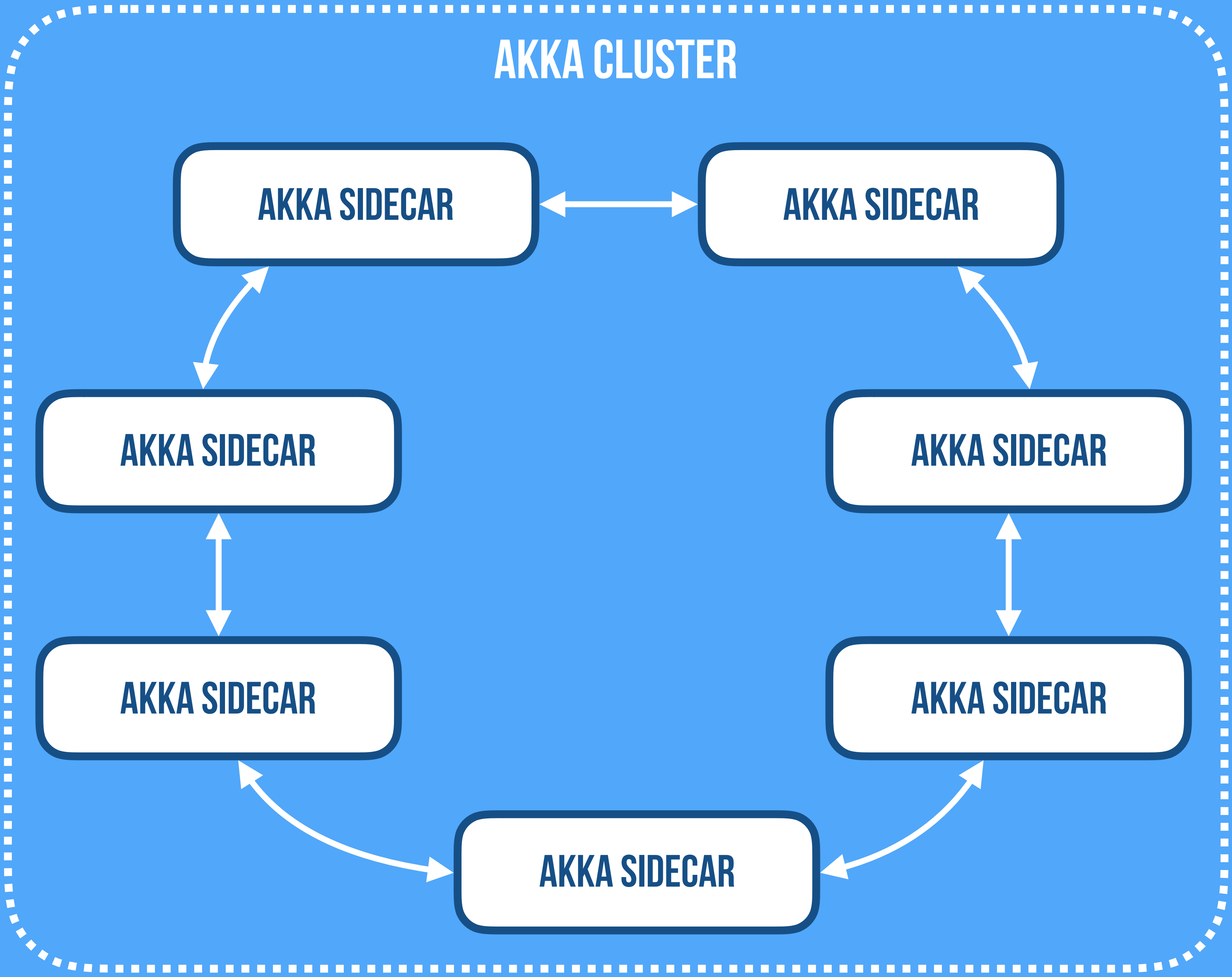
- Actor-based Distributed Runtime

AKKA CLUSTER STATE MANAGEMENT



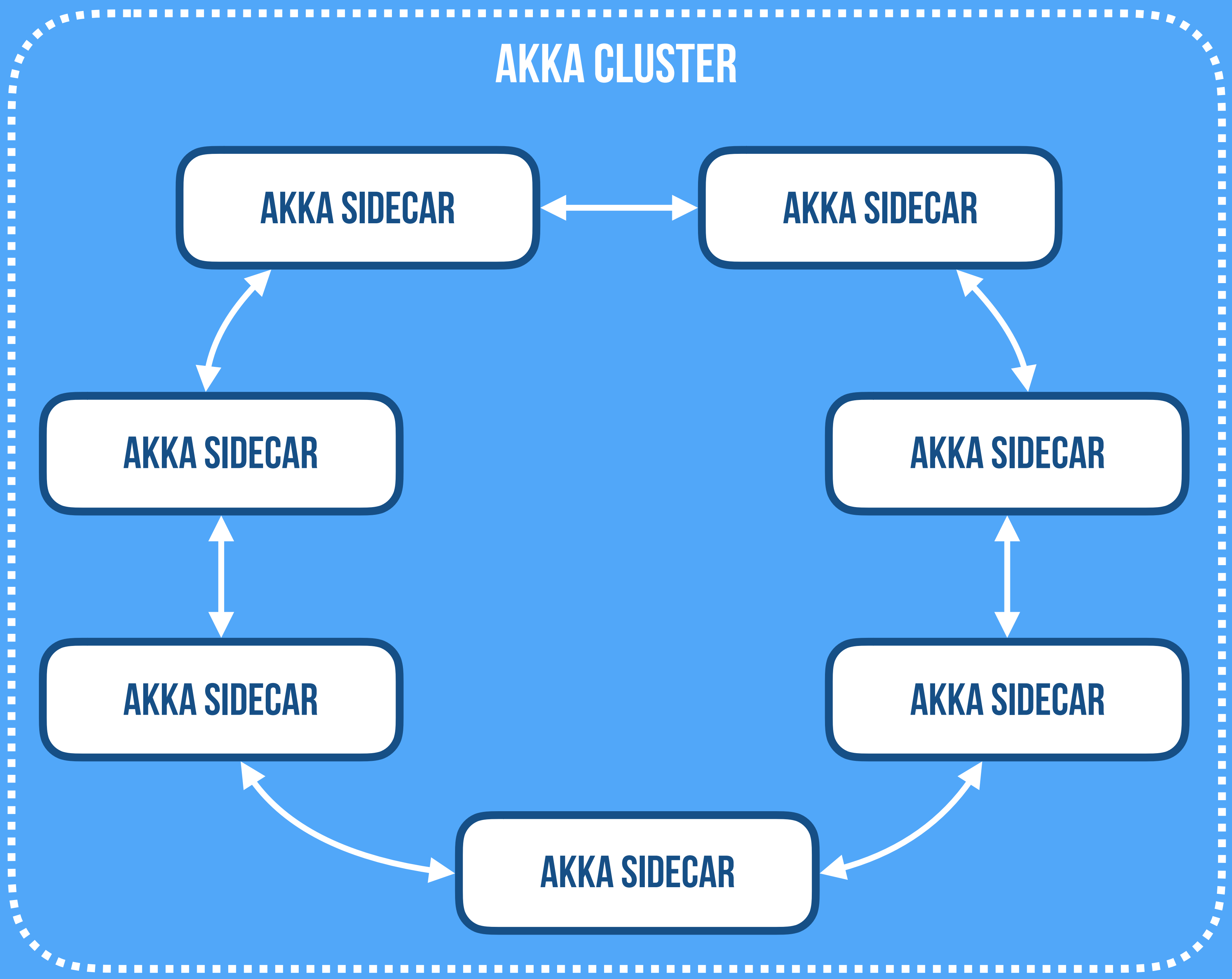
- Actor-based Distributed Runtime
- Dynamo-style Node Ring

AKKA CLUSTER STATE MANAGEMENT



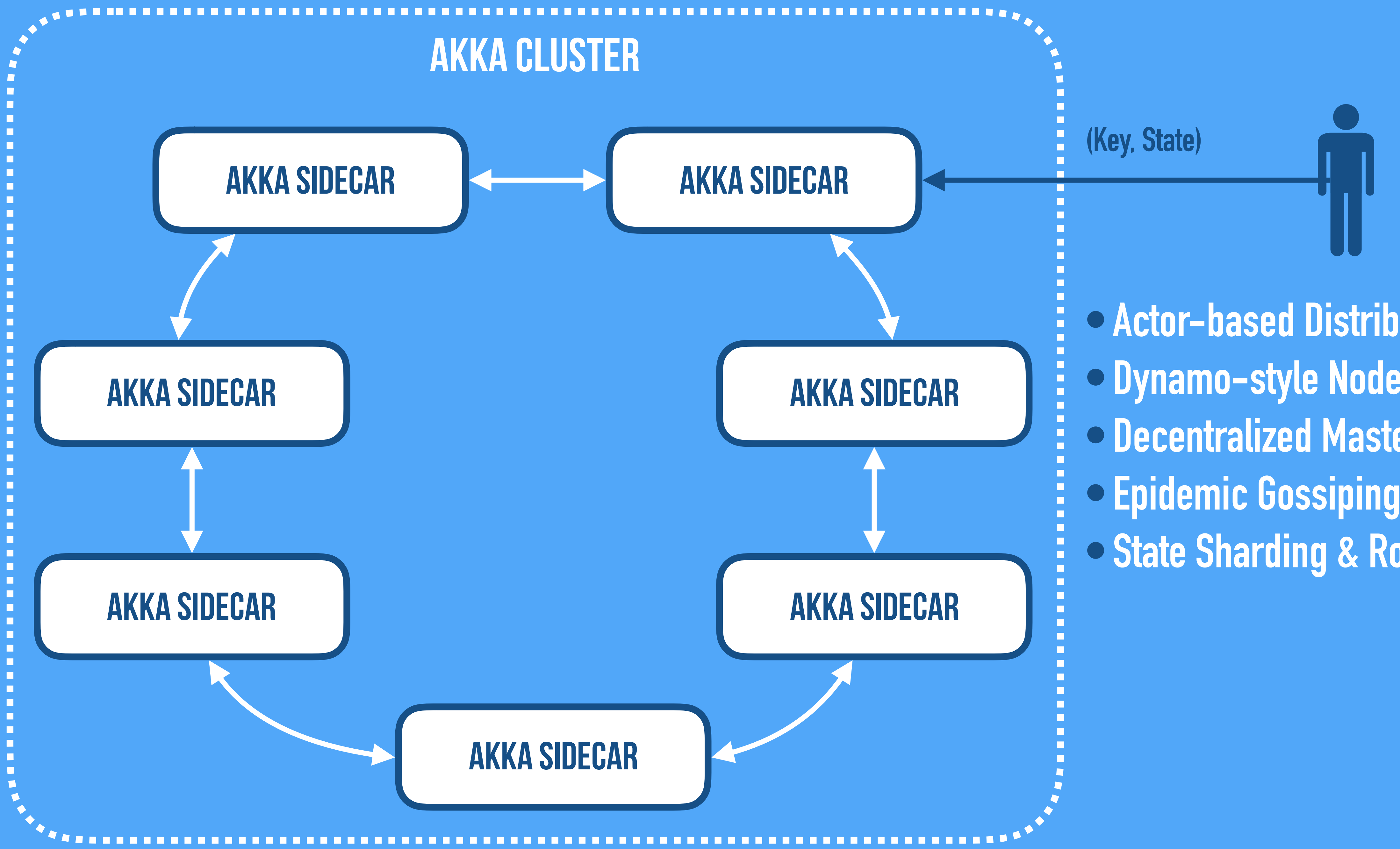
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P

AKKA CLUSTER STATE MANAGEMENT



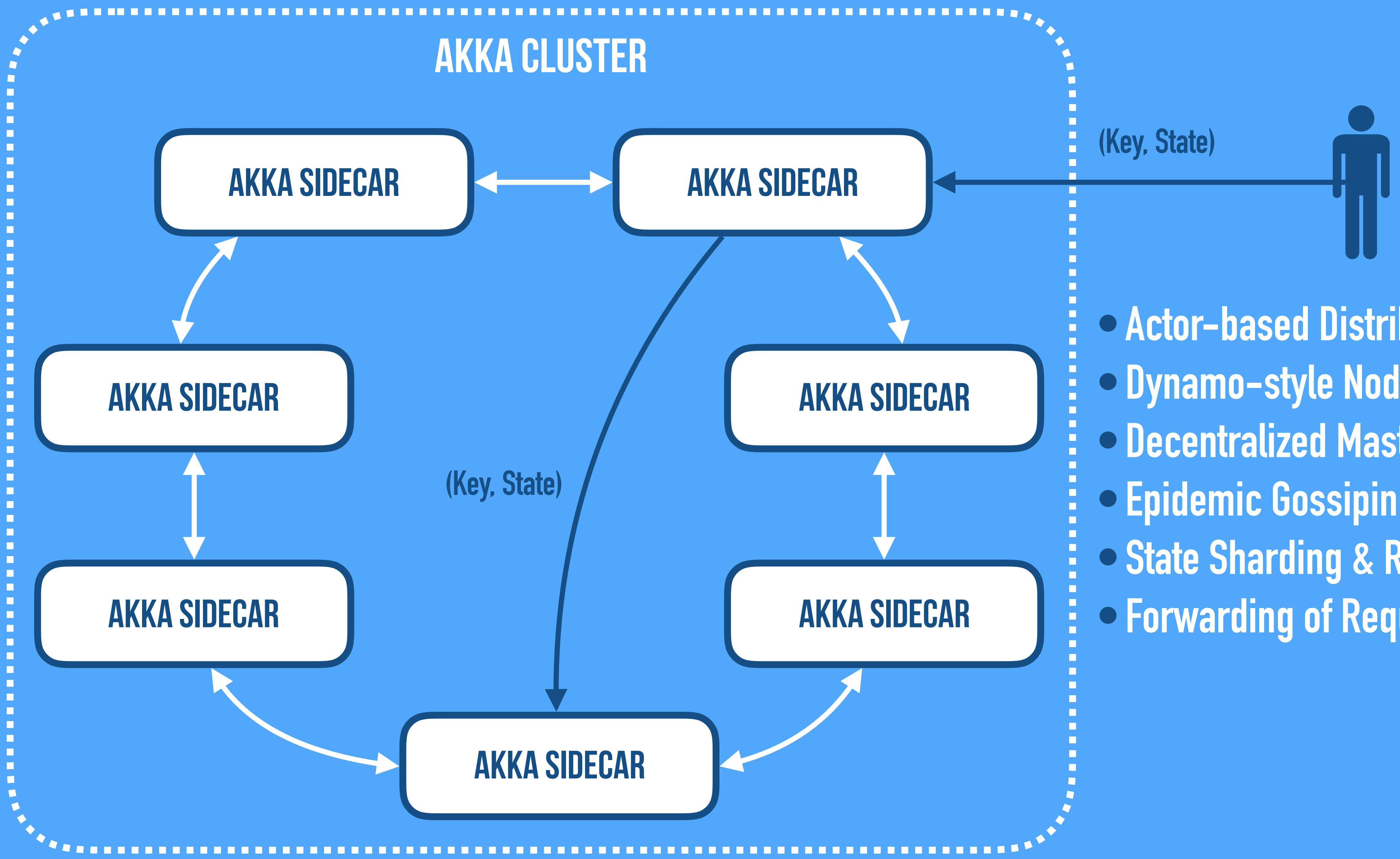
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing

AKKA CLUSTER STATE MANAGEMENT



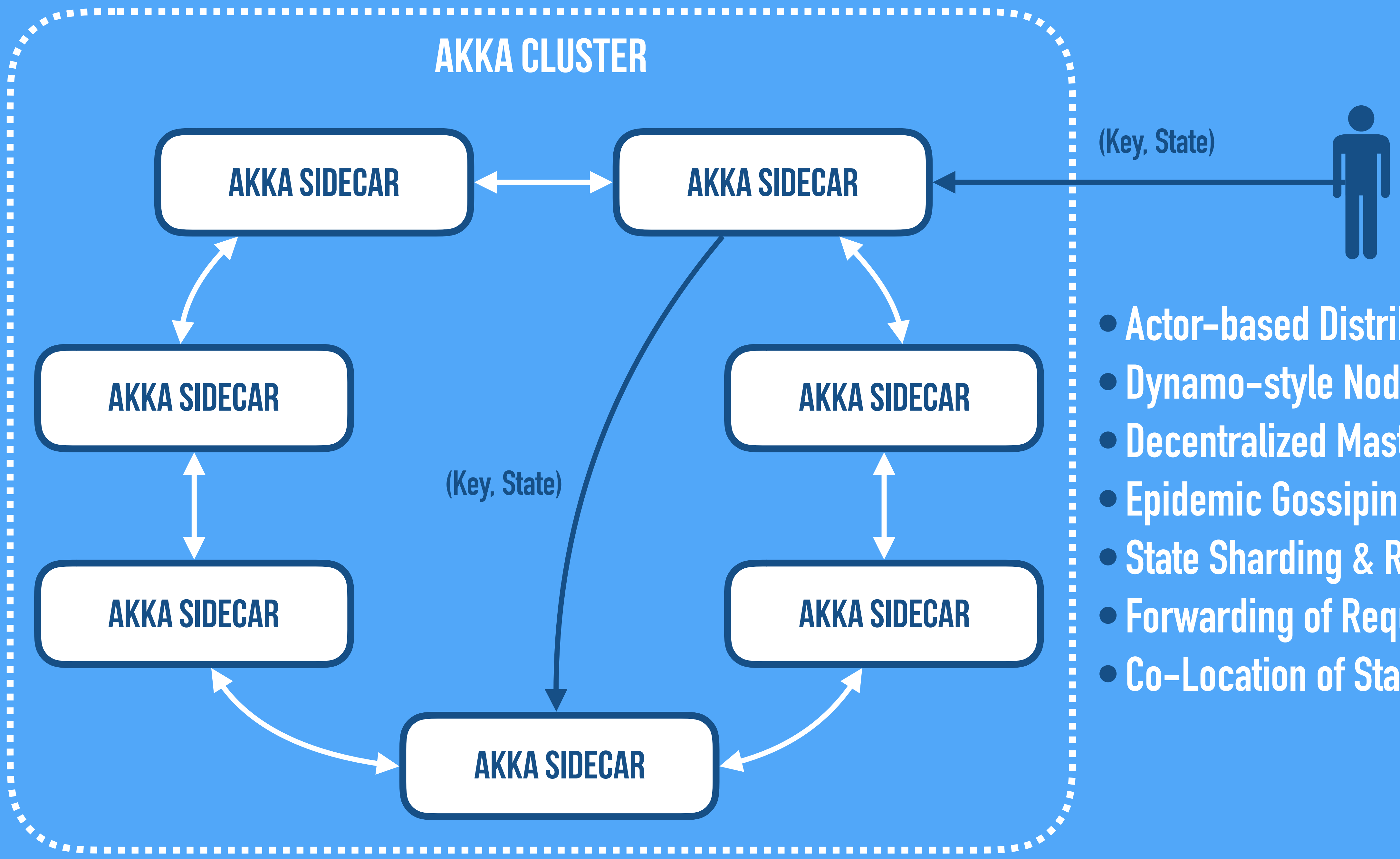
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key

AKKA CLUSTER STATE MANAGEMENT



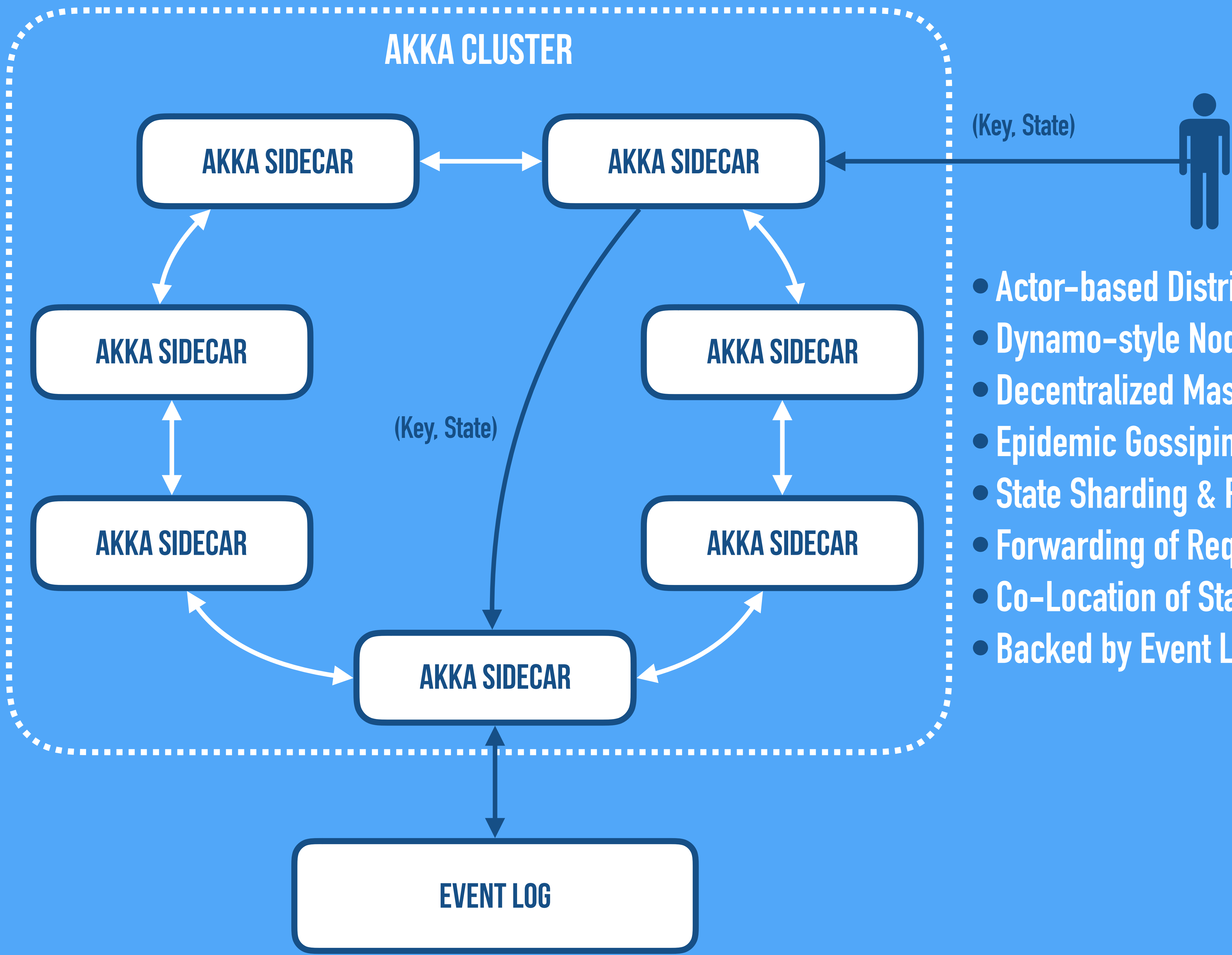
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)

AKKA CLUSTER STATE MANAGEMENT



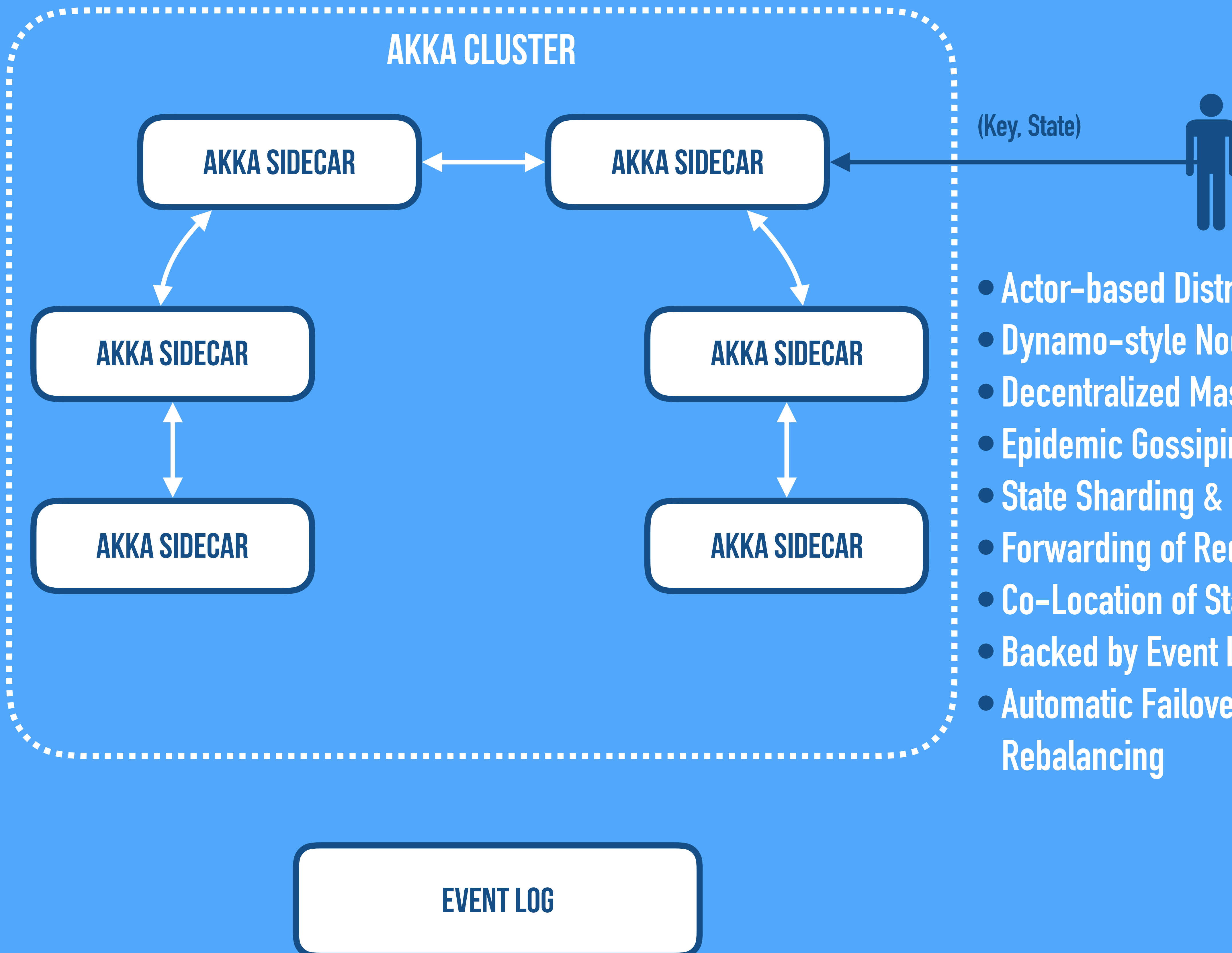
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing

AKKA CLUSTER STATE MANAGEMENT



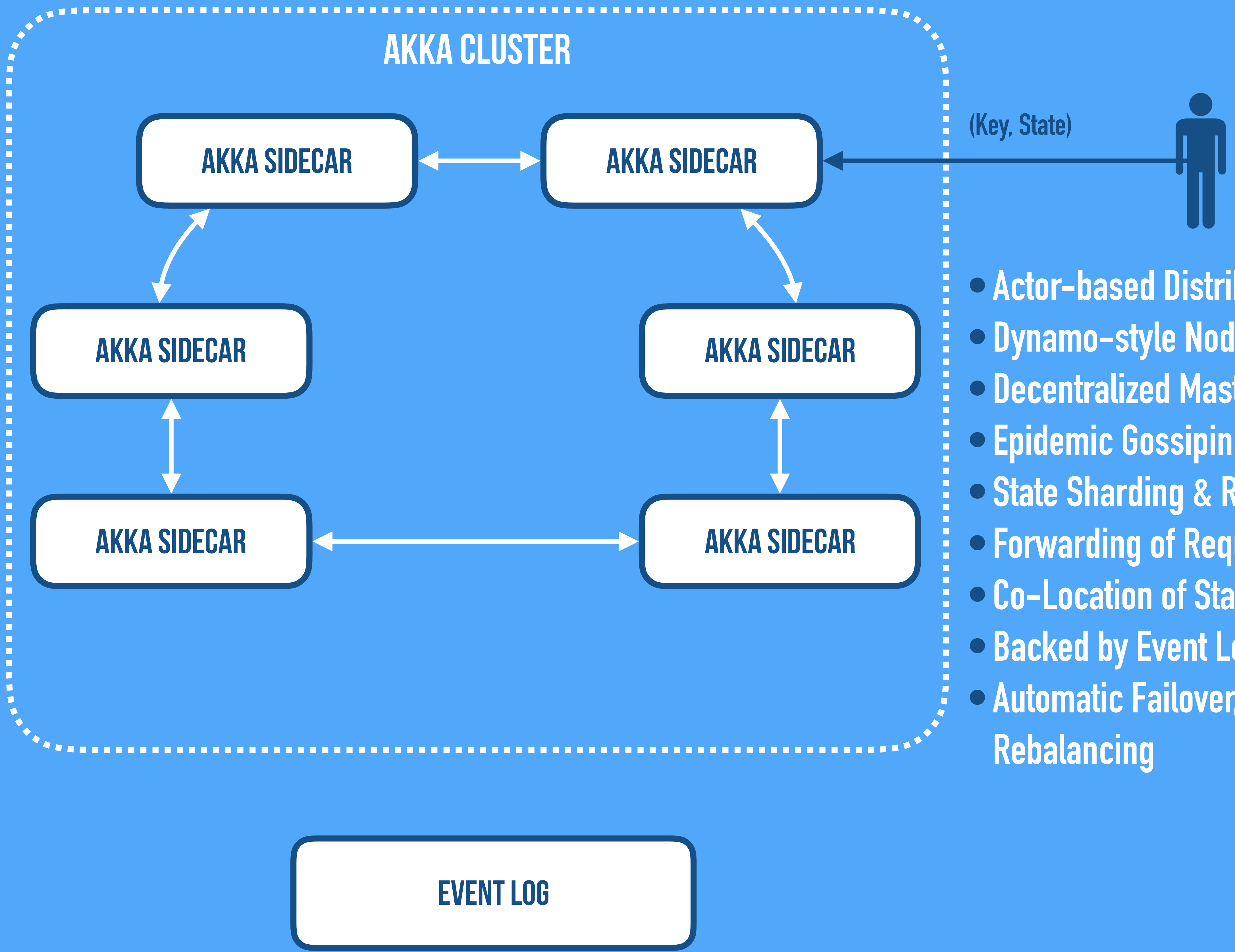
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log

AKKA CLUSTER STATE MANAGEMENT



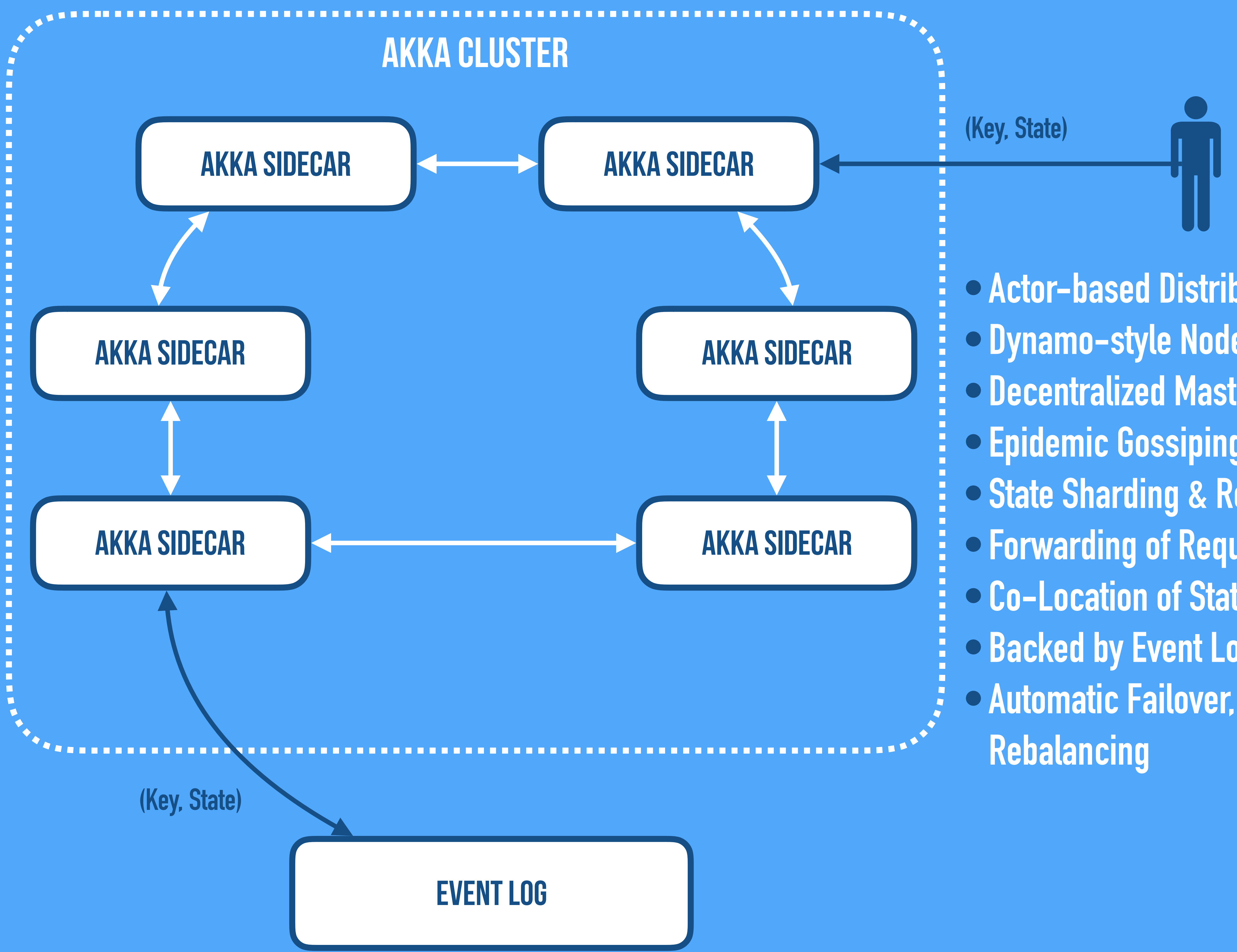
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

AKKA CLUSTER STATE MANAGEMENT



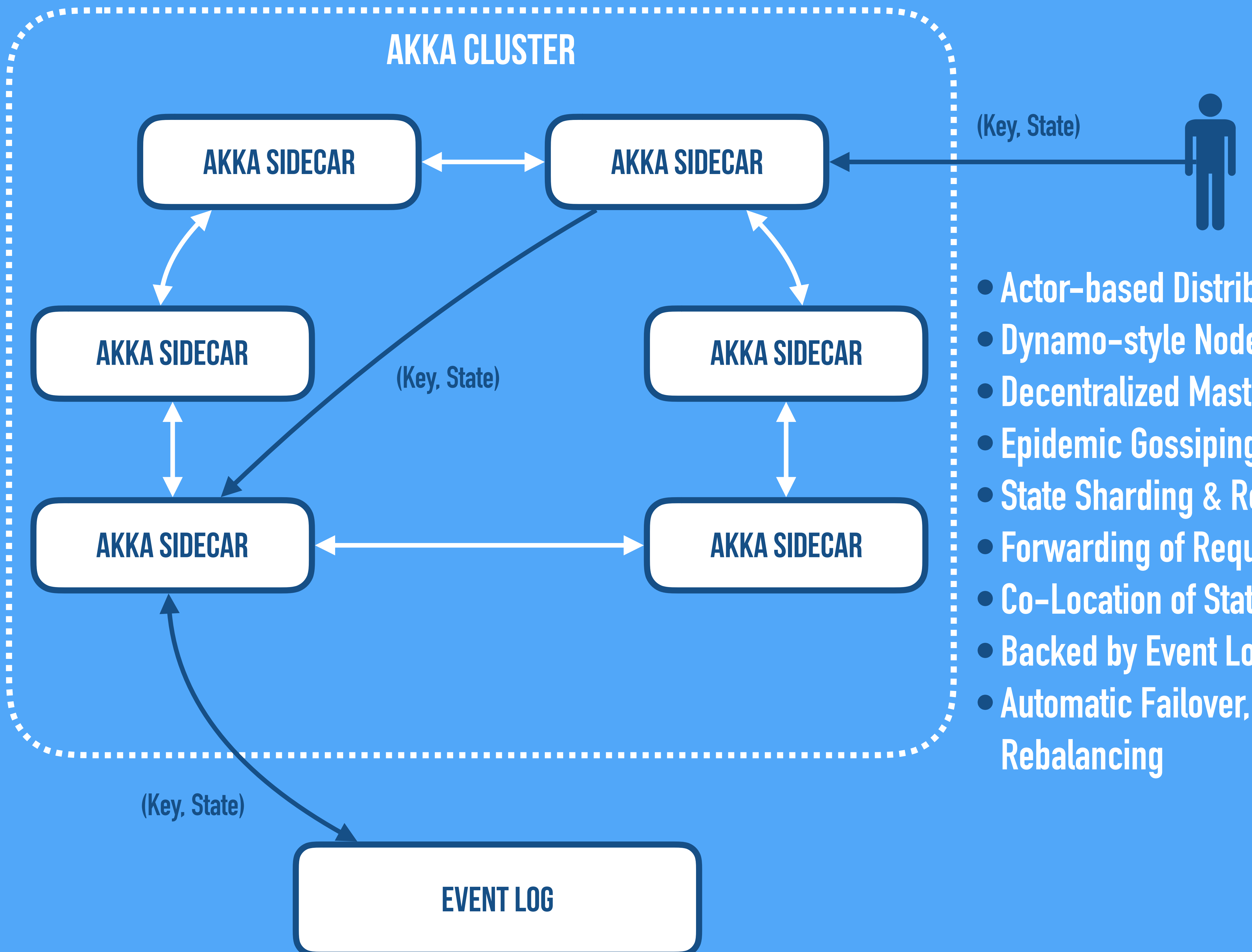
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

AKKA CLUSTER STATE MANAGEMENT



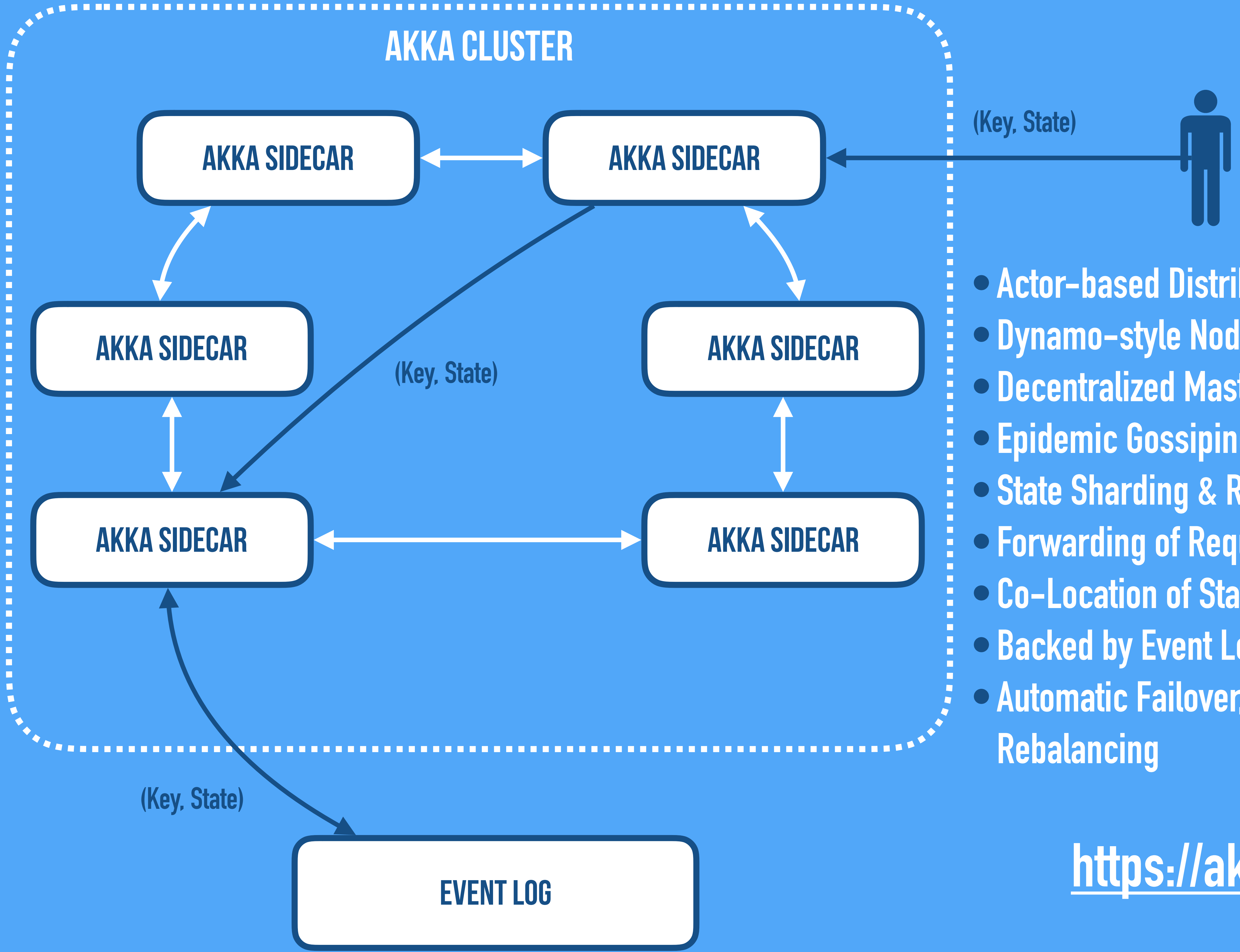
- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

AKKA CLUSTER STATE MANAGEMENT



- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

AKKA CLUSTER STATE MANAGEMENT



- Actor-based Distributed Runtime
- Dynamo-style Node Ring
- Decentralized Masterless P2P
- Epidemic Gossiping, Self-healing
- State Sharding & Routing on Entity Key
- Forwarding of Requests (if needed)
- Co-Location of State & Processing
- Backed by Event Log
- Automatic Failover, Rehydration, and Rebalancing

https://akka.io

In Summary

In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud

In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let's not stop here

In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let's not stop here
3. Serverless 2.0 needs a runtime & programming model for general-purpose application development

In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let's not stop here
3. Serverless 2.0 needs a runtime & programming model for general-purpose application development
4. We can't ignore/delegate the hardest thing: State

In Summary

1. The promise of Serverless is revolutionary and will grow to dominate the future of Cloud
2. FaaS is a great first step, but let's not stop here
3. Serverless 2.0 needs a runtime & programming model for general-purpose application development
4. We can't ignore/delegate the hardest thing: State
5. We think that Cloudstate shows the way