Reactive Microsystems

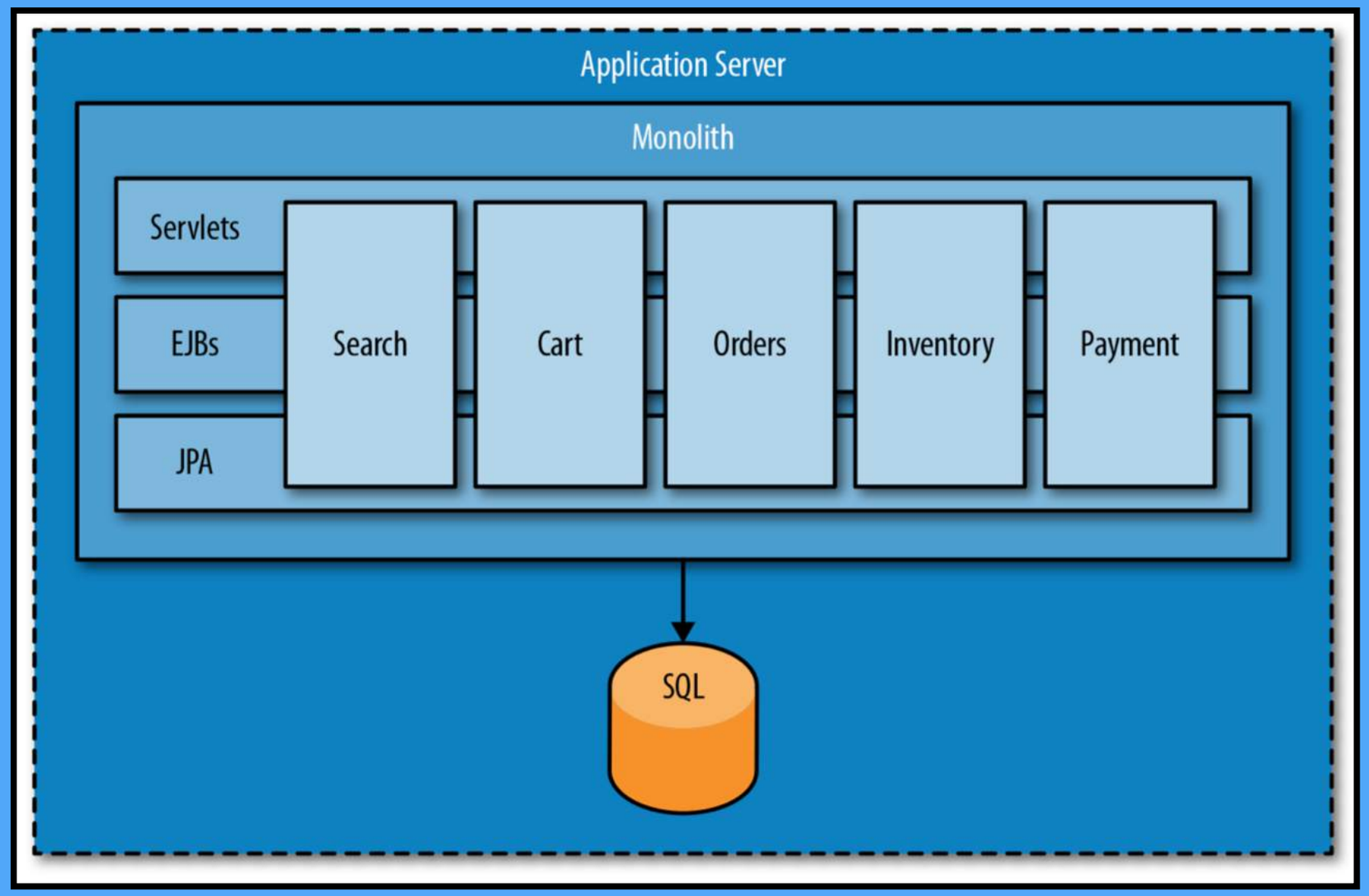
The Evolution of Microservices at Scale

Jonas Bonér
@jboner

Lightbend

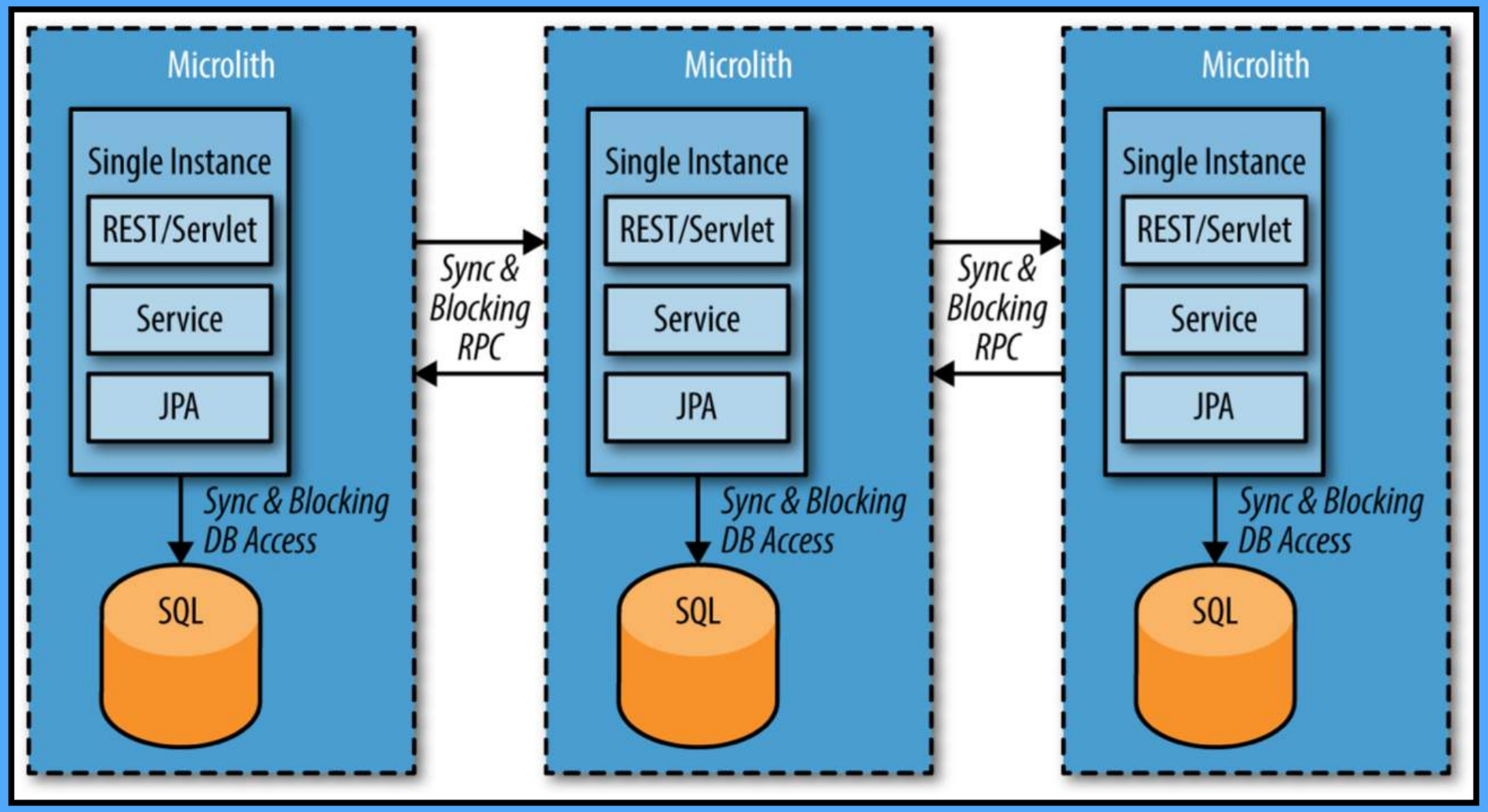
Microservices Are Hyped

SO YOU WANT TO STRANGLE THE Almighty Monolith AND MOVE TOWARDS MICROSERVICES?



DO NOT SETTLE FOR

Microliths



Microlith

Single instance microservice,
communicating over blocking protocols

Microlith

Sin microservice,
over blocking protocols

Not Resilient

Microlith

Not Resilie

Not Scalable

Event Driven Architecture To The Rescue

"Event-Driven Architecture (EDA) is a design paradigm in which a software component executes in response to receiving one or more event notifications."

- GARTNER

The Nature of Events

The Nature of Events

- Events represent FACTS OF INFORMATION
  - ➡ FACTS ARE IMMUTABLE
  - ➡ FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW

The Nature of Events

- Events represent FACTS OF INFORMATION
  - ➡ FACTS ARE IMMUTABLE
  - ➡ FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW
- Events/Facts CAN BE DISREGARDED/IGNORED

The Nature of Events

- Events represent FACTS OF INFORMATION
  - ➡ FACTS ARE IMMUTABLE
  - ➡ FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW
- Events/Facts CAN BE DISREGARDED/IGNORED
- Events/Facts CAN NOT BE RETRACTED (once accepted)

The Nature of Events

- Events represent FACTS OF INFORMATION
  - ➡ FACTS ARE IMMUTABLE
  - ➡ FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW

- Events/Facts CAN BE DISREGARDED/IGNORED
- Events/Facts CAN NOT BE RETRACTED (once accepted)
- Events/Facts CAN NOT BE DELETED (once accepted)
  - ➡ Might be needed for LEGAL OR MORAL REASONS

The Nature of Events

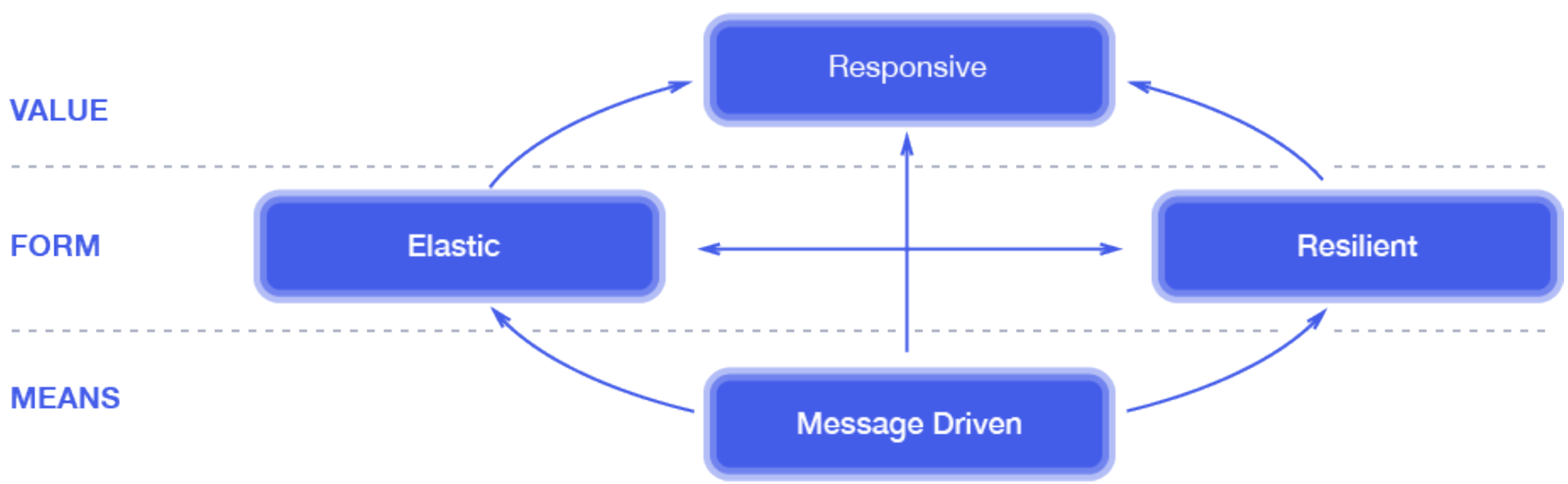
- Events represent FACTS OF INFORMATION
  - ➡ FACTS ARE IMMUTABLE
  - ➡ FACTS ACCRUE - KNOWLEDGE CAN ONLY GROW

- Events/Facts CAN BE DISREGARDED/IGNORED
- Events/Facts CAN NOT BE RETRACTED (once accepted)
- Events/Facts CAN NOT BE DELETED (once accepted)
  - ➡ Might be needed for LEGAL OR MORAL REASONS
- Events/Facts (new) CAN INVALIDATE existing Facts

Event Driven Services

1. RECEIVE and REACT (or not) TO FACTS, that are coming its way
2. PUBLISH NEW FACTS (as events) to the rest of the world
3. INVERT THE CONTROL FLOW to minimize coupling and increase autonomy

Practice Reactive Design



Go Never Async Block

- Efficient use of resources
- Minimizes contention

Always Apply
Back Pressure

A FAST system

Should NOT OVERLOAD

A SLOW system

Always Apply
Back Pressure

A FAST system

Should NOT OVERLOAD

A SLOW system

Microservices Come In Systems

Event Stream

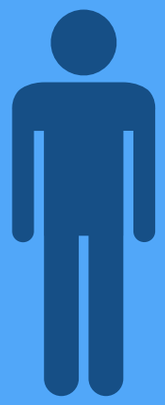
USE THE

Event Stream

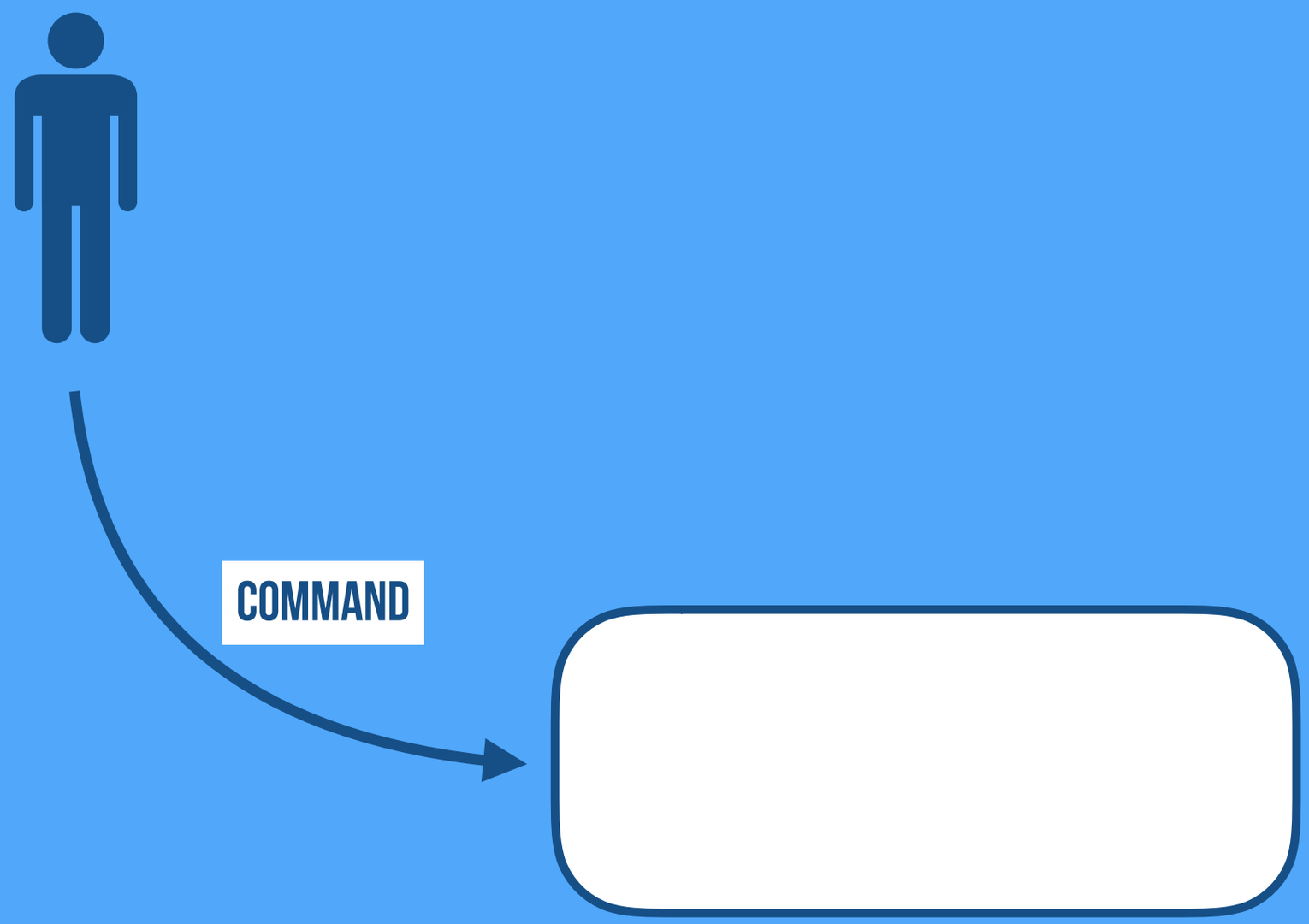
AS THE INTEGRATION FABRIC

Event Driven
Services

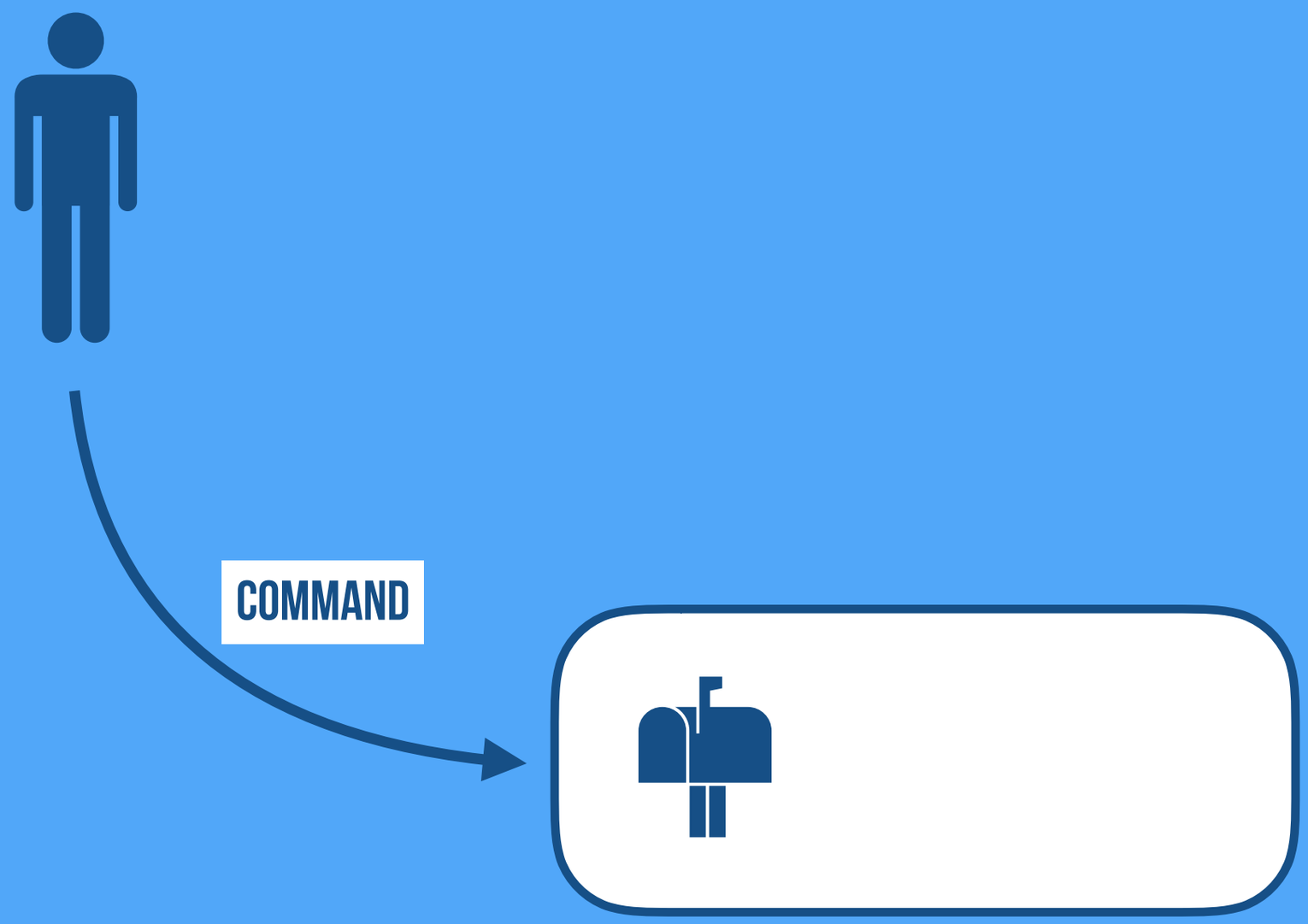
Event Driven Services



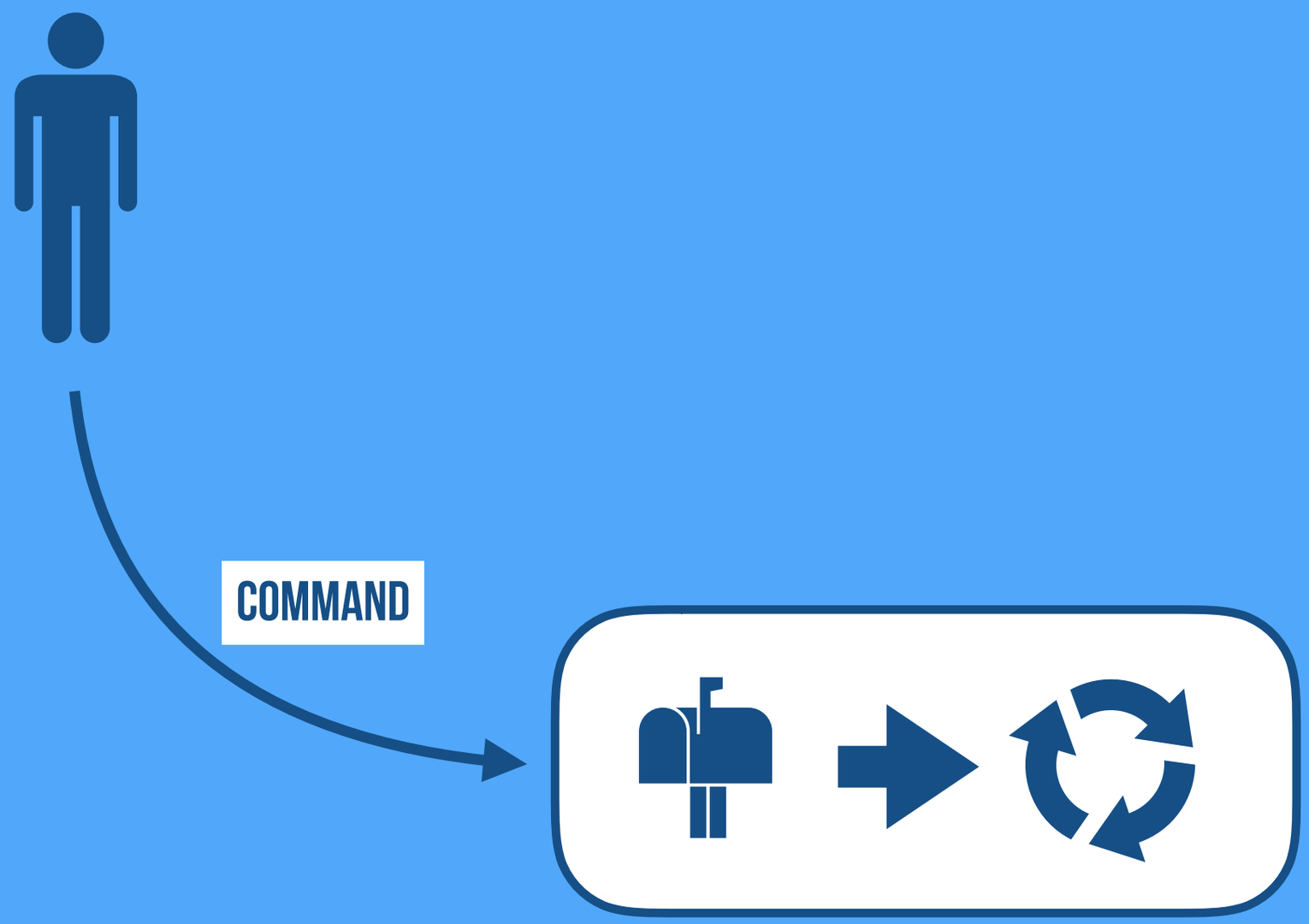
Event Driven Services



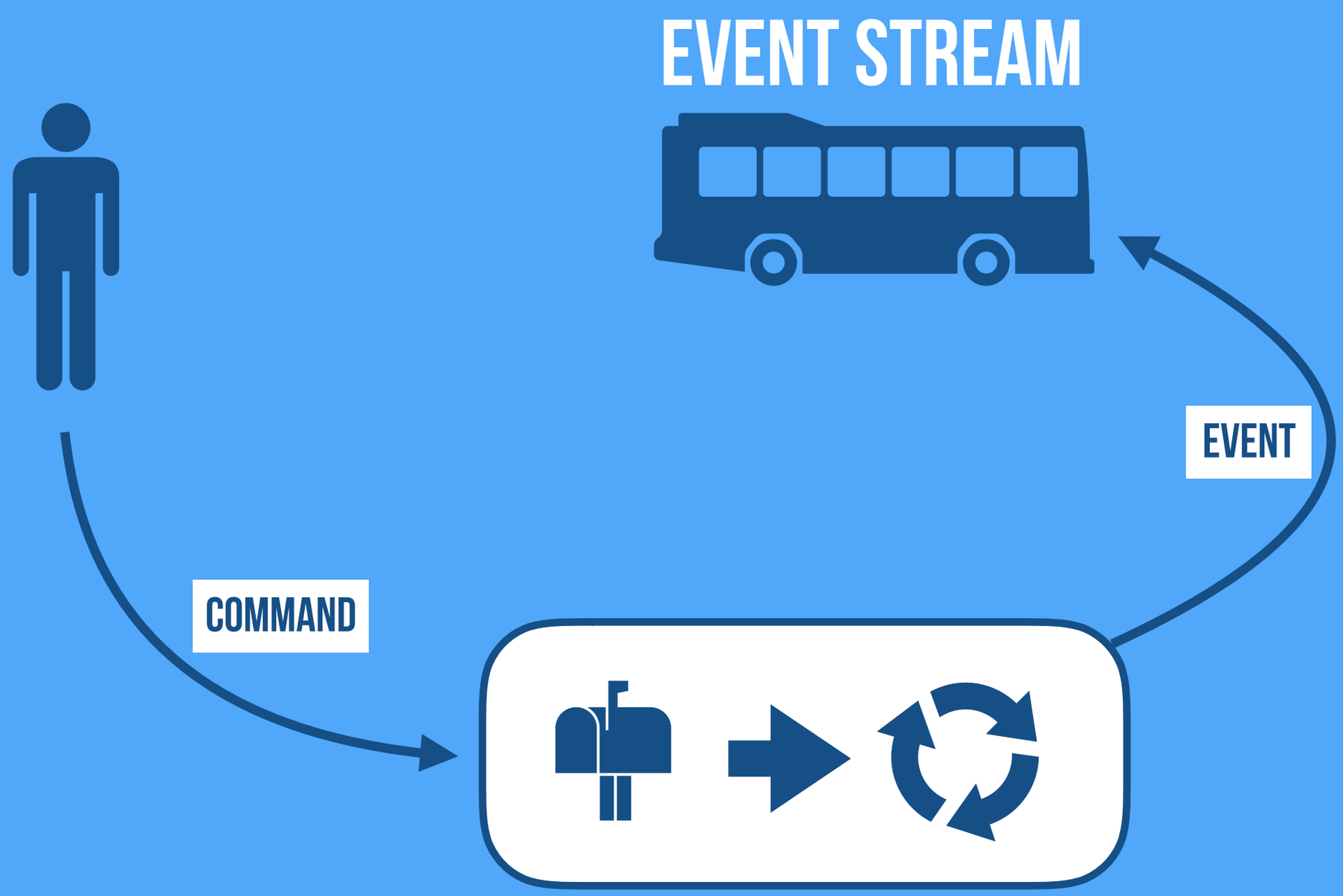
Event Driven Services



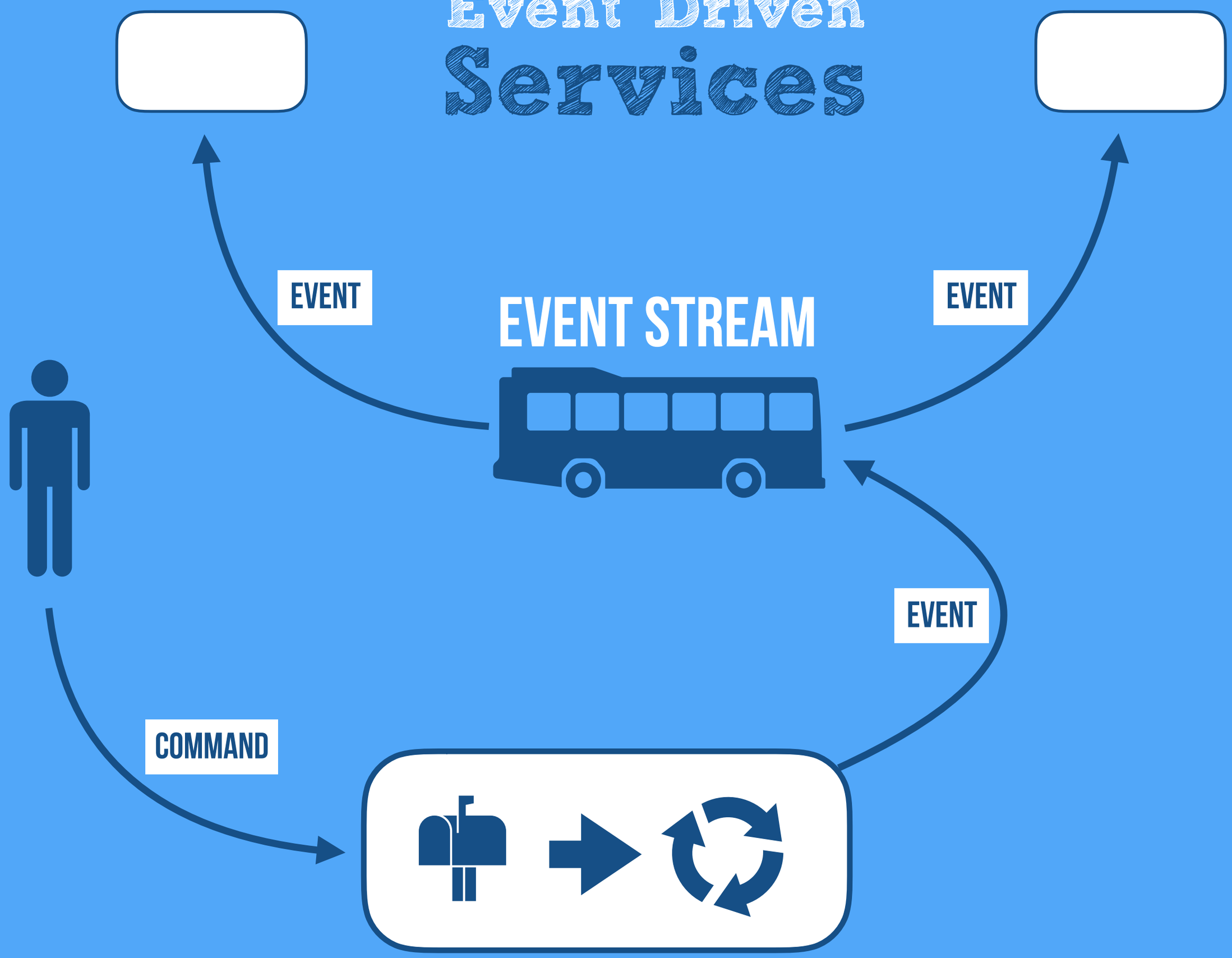
Event Driven Services



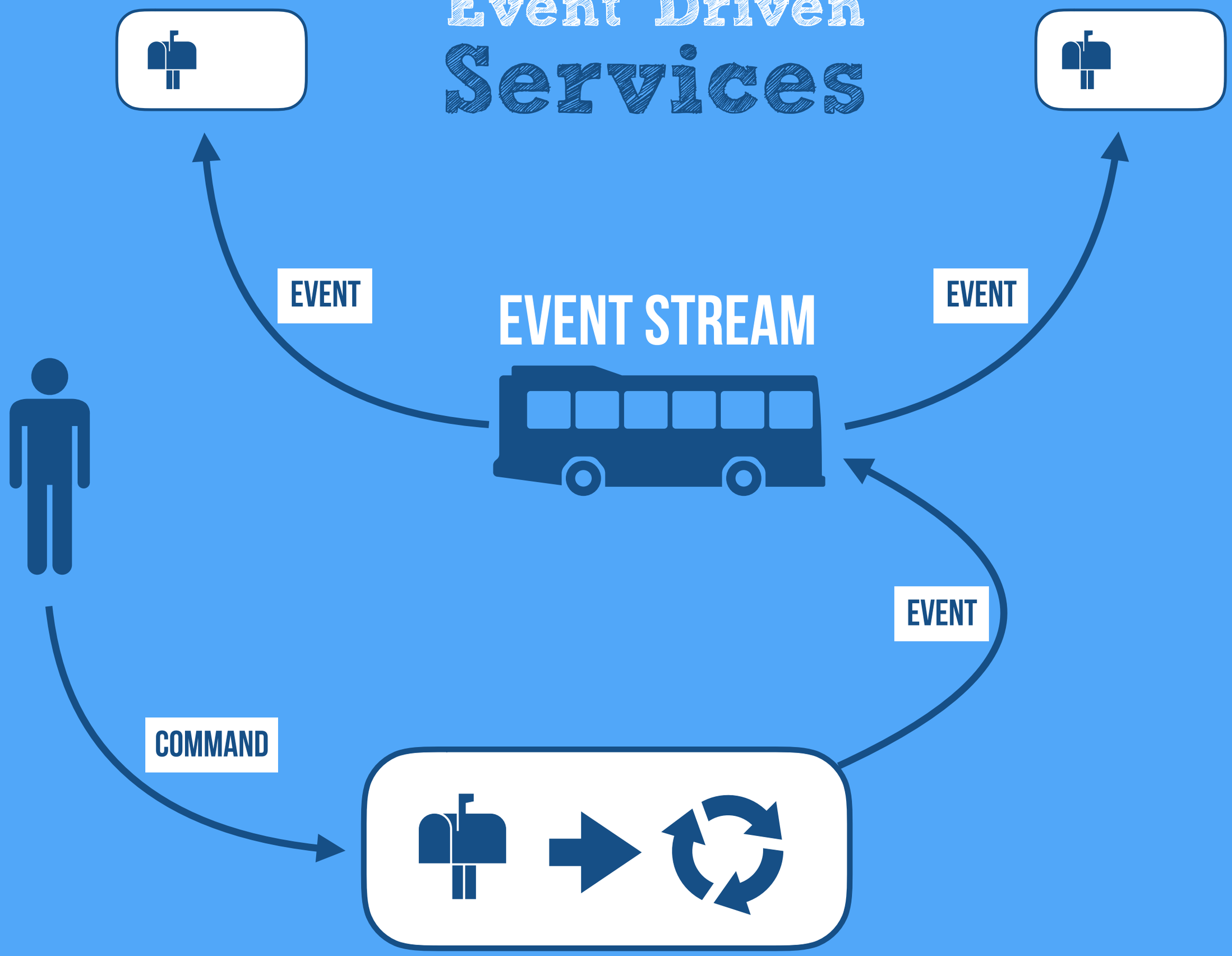
Event Driven Services



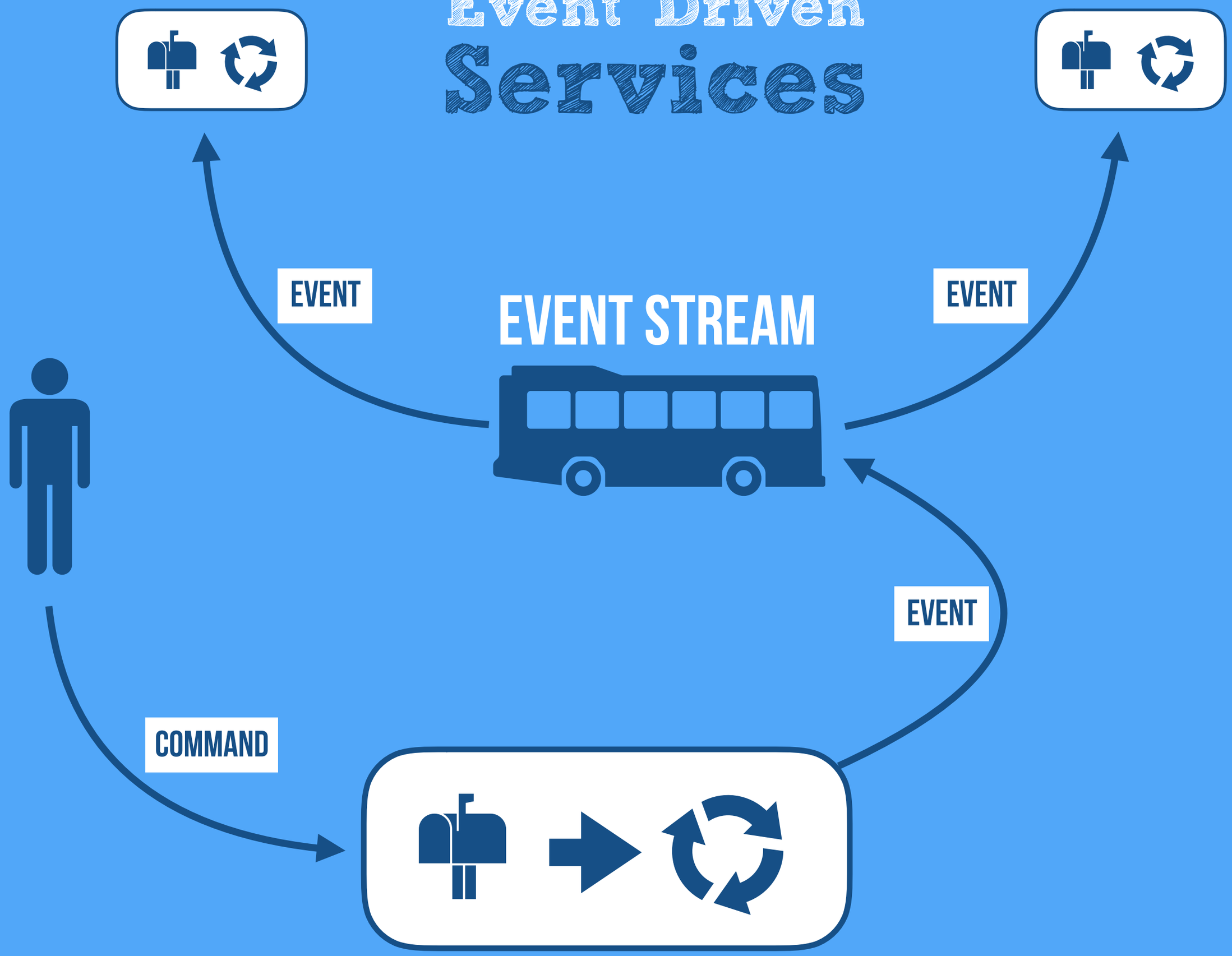
Event Driven Services



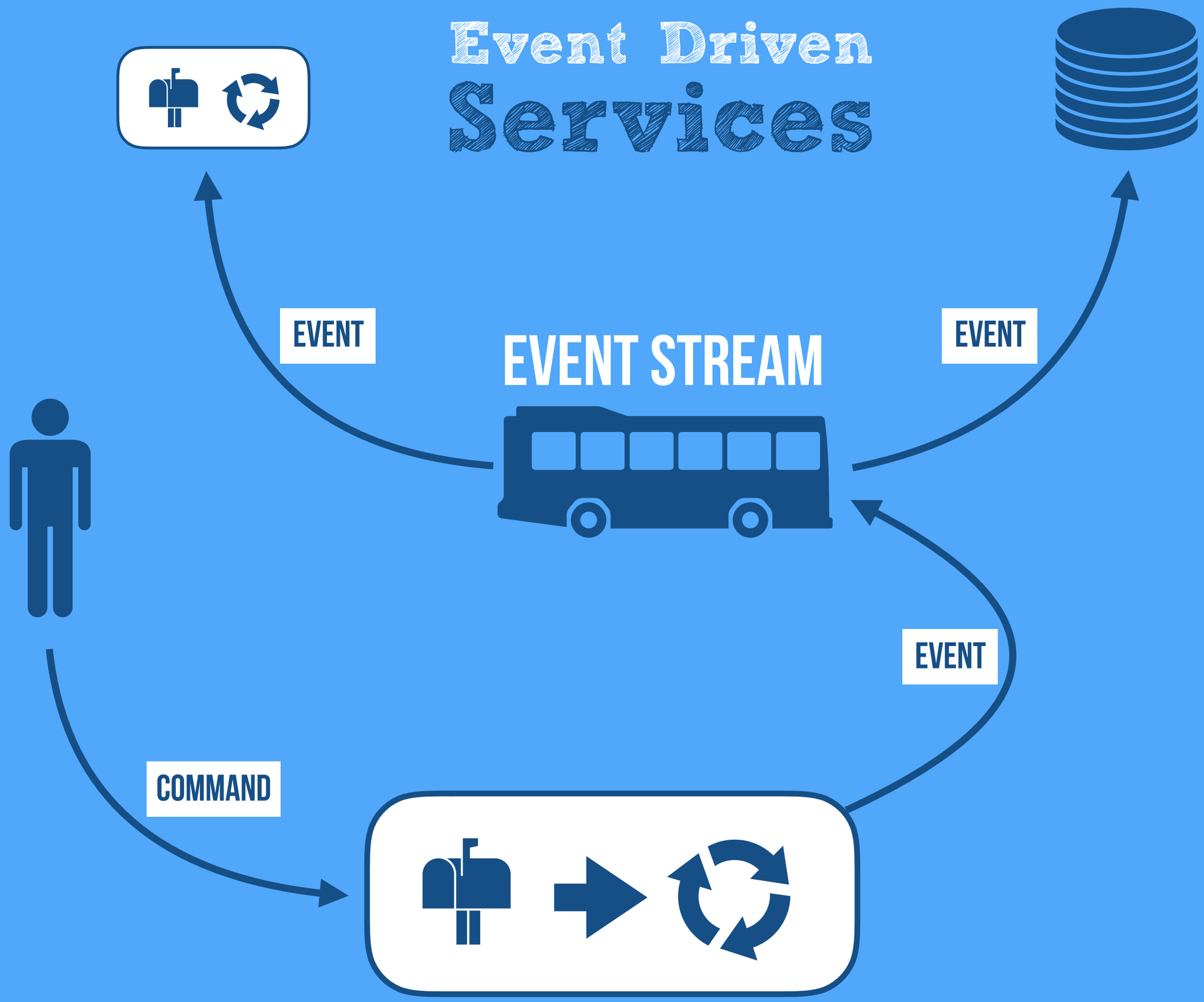
Event Driven Services



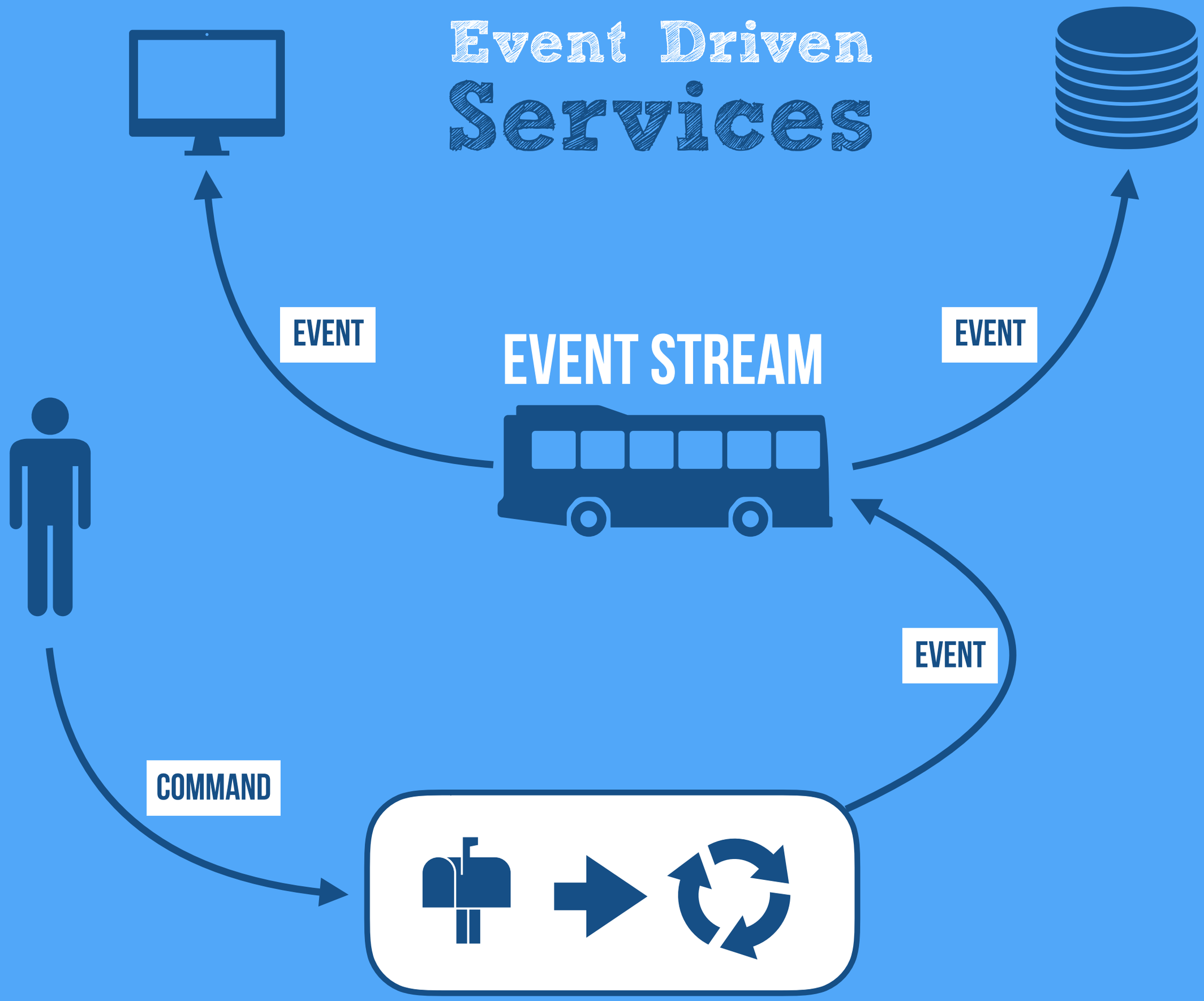
Event Driven Services



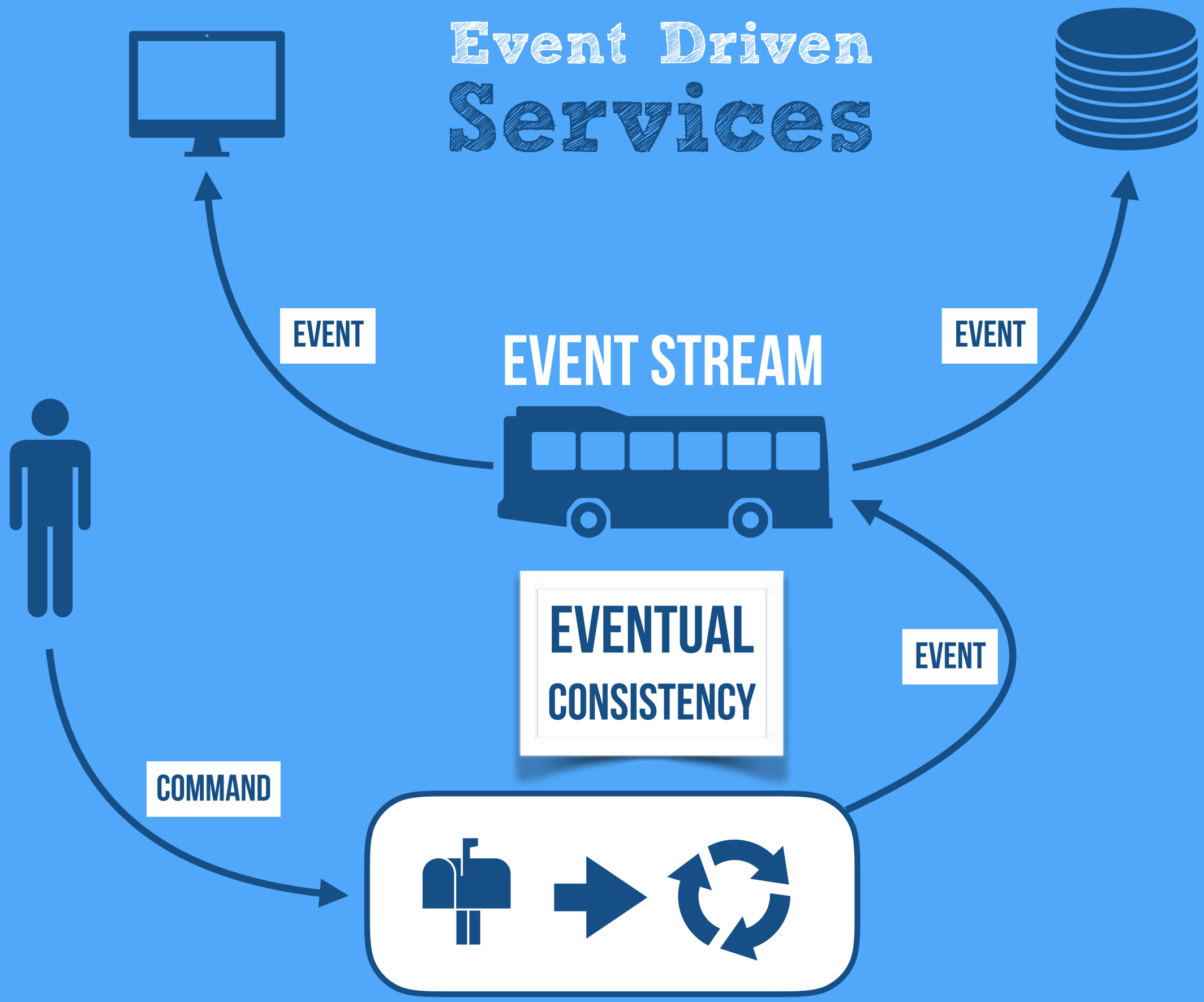
Event Driven Services



Event Driven Services



Event Driven Services



Eventual Consistency

NO ONE WANTS

Eventual Consistency

IT'S A NECESSARY EVIL

Information Has Latency

Information Is Always From the Past

Welcome To The Wild Ocean Of Non Determinism Distributed Systems

We Need To Model Uncertainty

"In a system which cannot count on distributed transactions, the management of uncertainty must be implemented in the business logic."

- PAT HELLAND

Life Beyond Distributed Transactions - An Apostate's Opinion, Pat Helland (2007)

Exploit Reality

WE NEED TO

Exploit Reality

IN OUR DESIGN

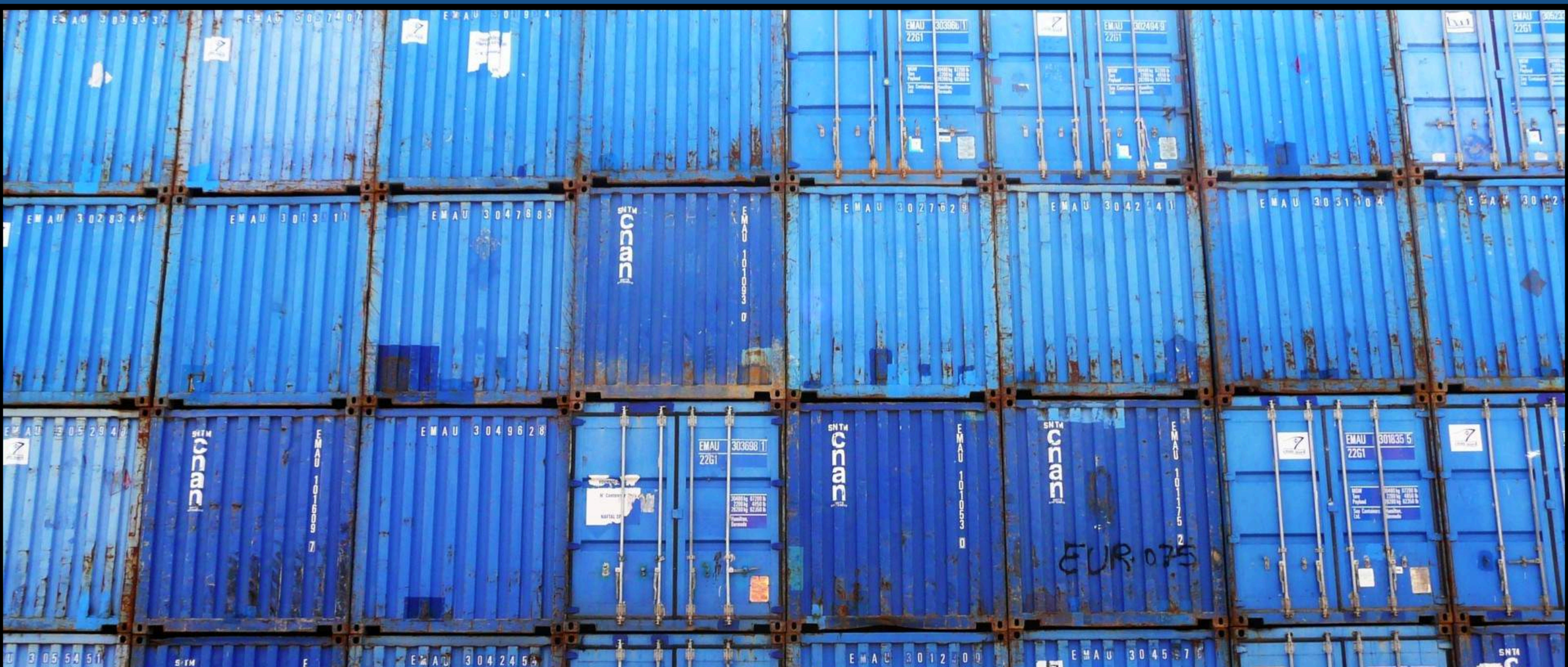
Events Can Lead To Greater

Certainty

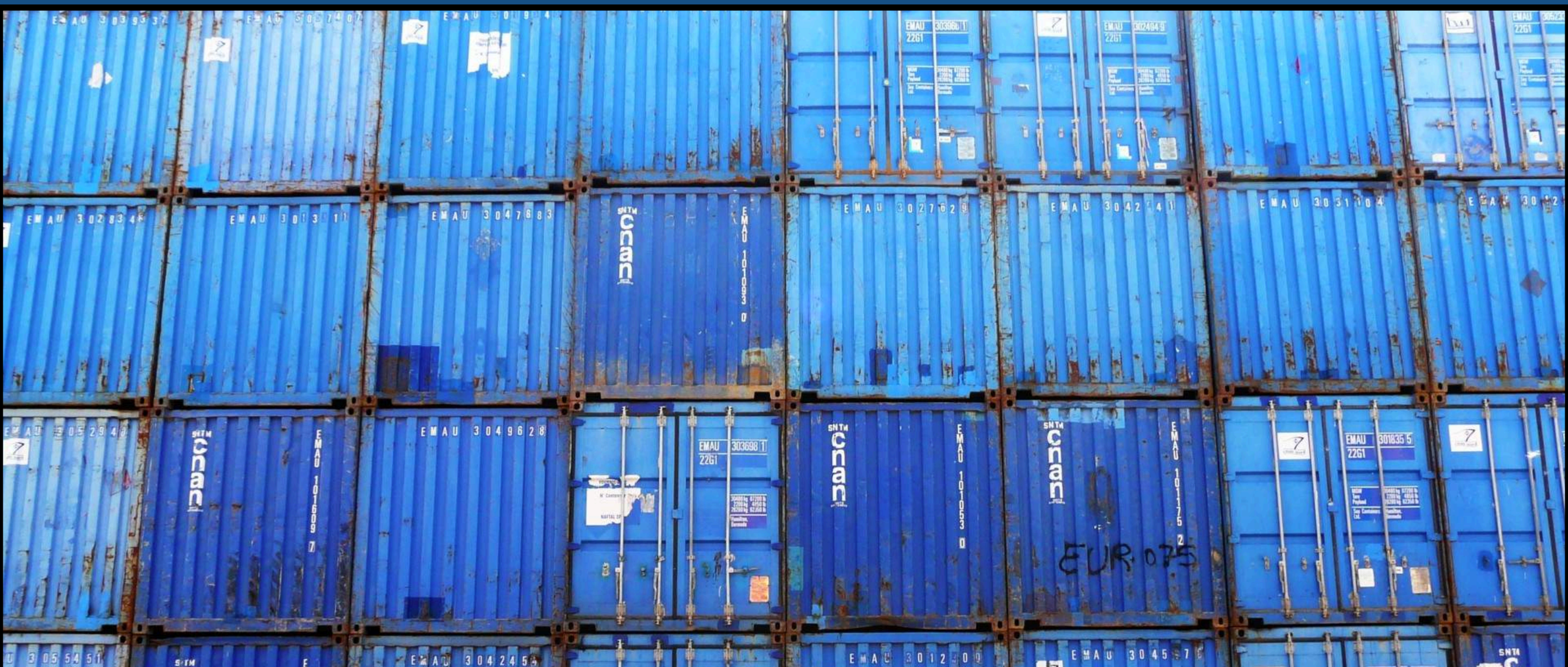
Events Can Help Us Craft Autonomous Islands Of Determinism

Mutable State Needs To Be Contained

And Non Observable



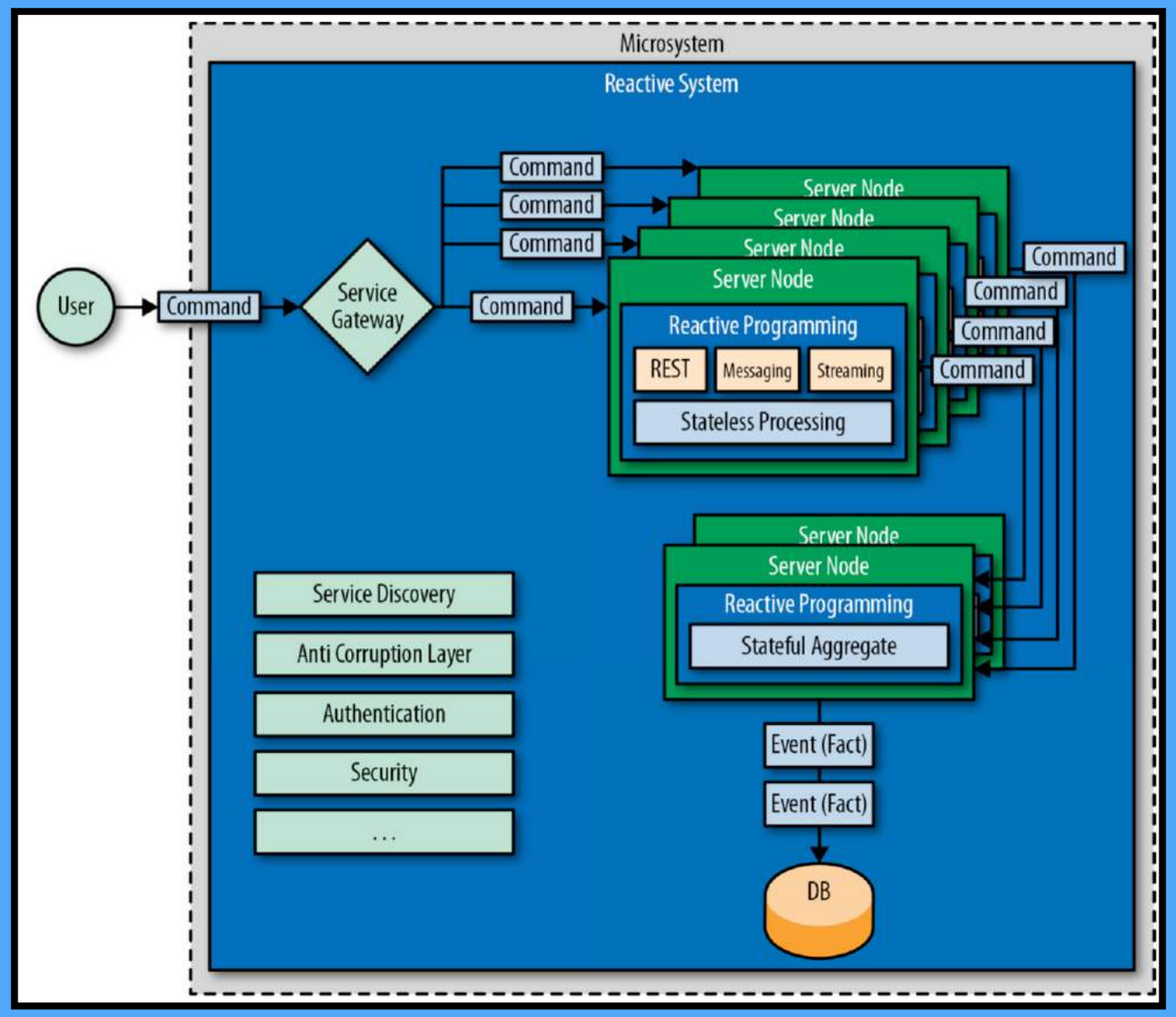
Publish Facts
To Outside World



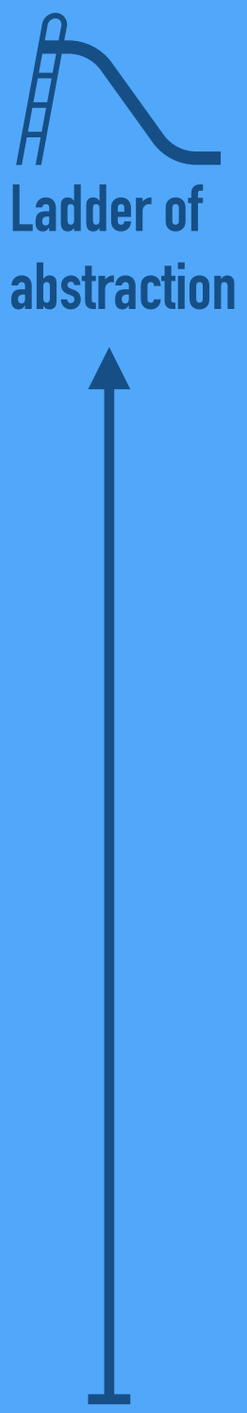
Microservices Come As Systems

Each microservice needs to be designed as a distributed system

A MICROSYSTEM

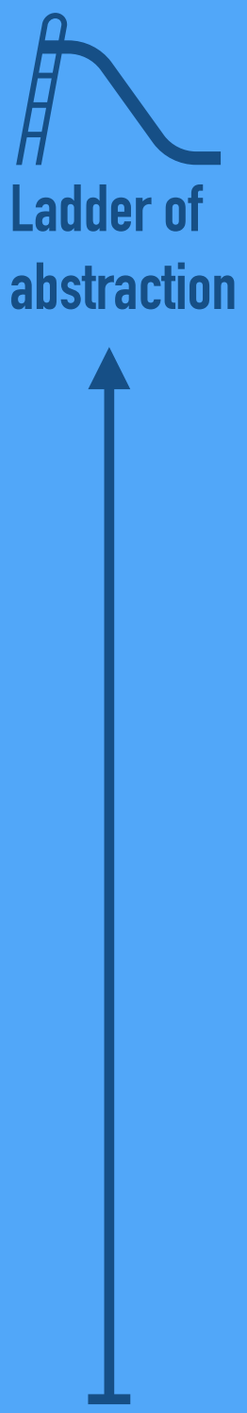


Event Driven Microservices
Powered By
Reactive Systems Fabric

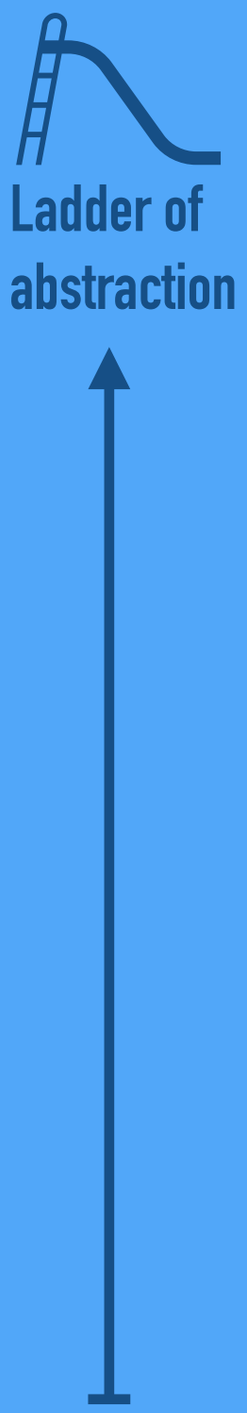


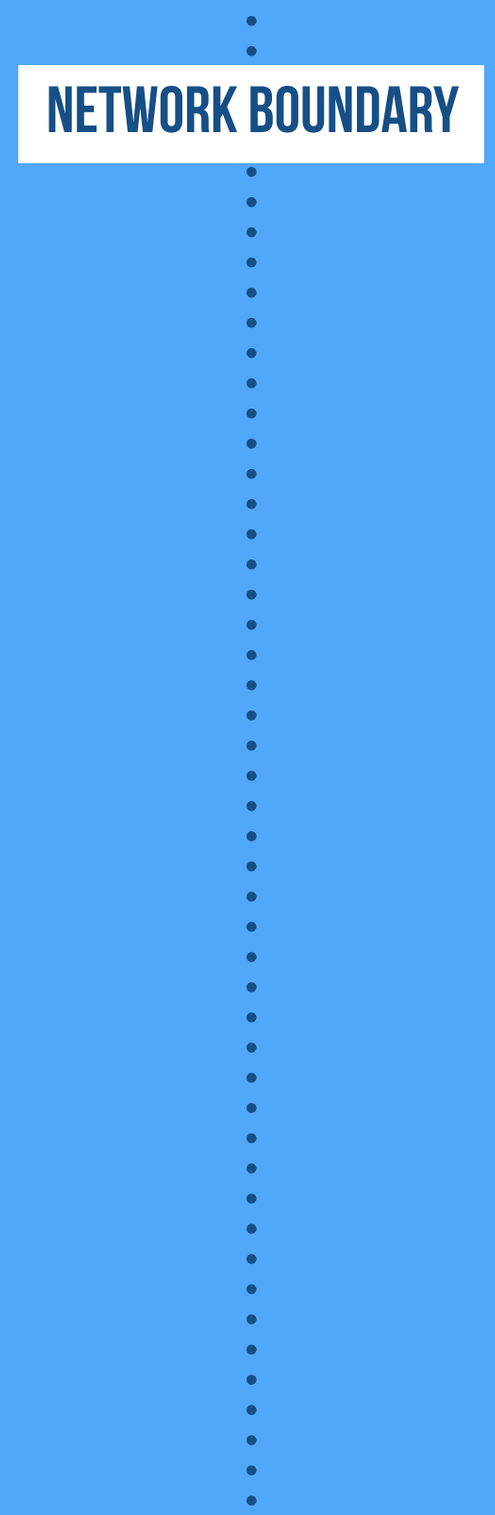
Event Driven Microservices
Powered By
Reactive Systems Fabric

NETWORK BOUNDARY

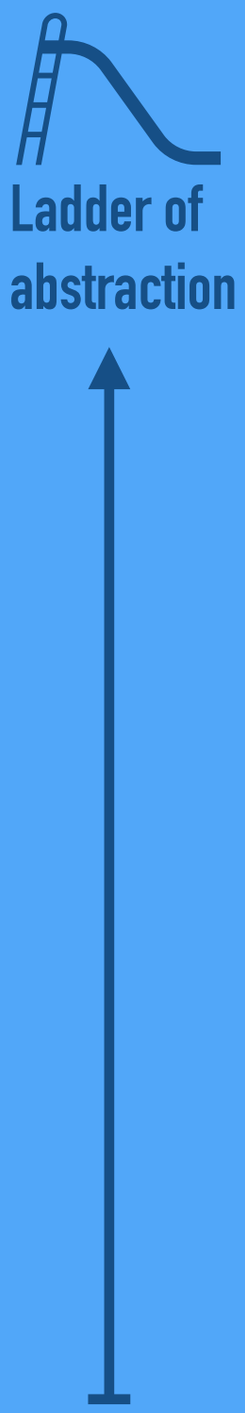


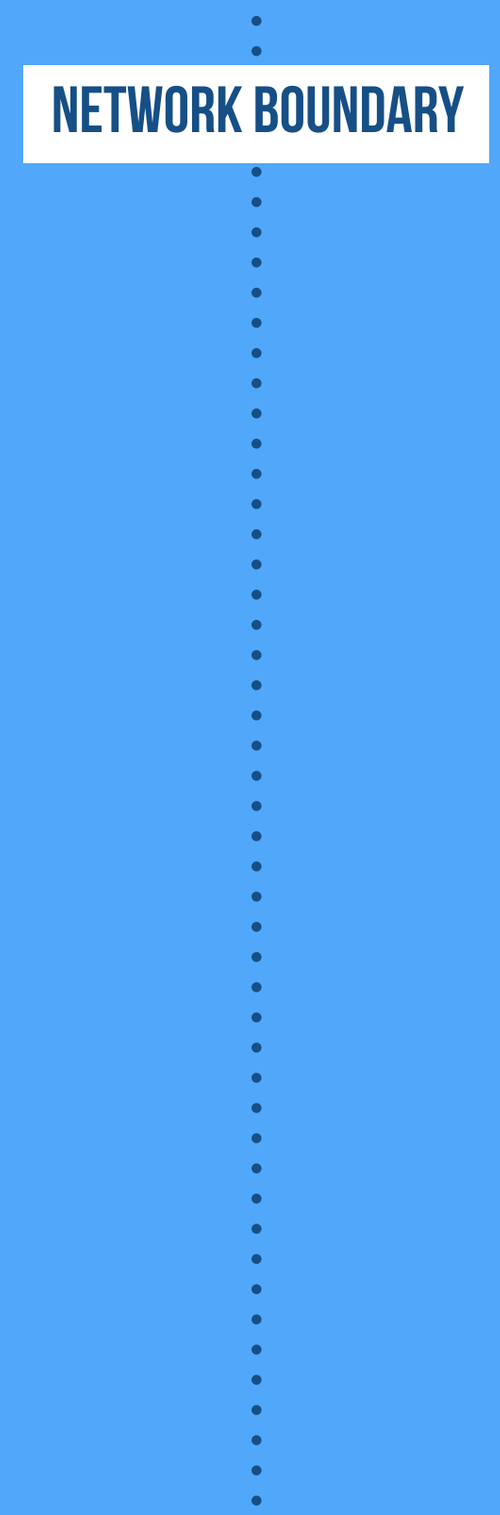
Event Driven Microservices
Powered By
Reactive Systems Fabric





Event Driven Microservices
Powered By
Reactive Systems Fabric





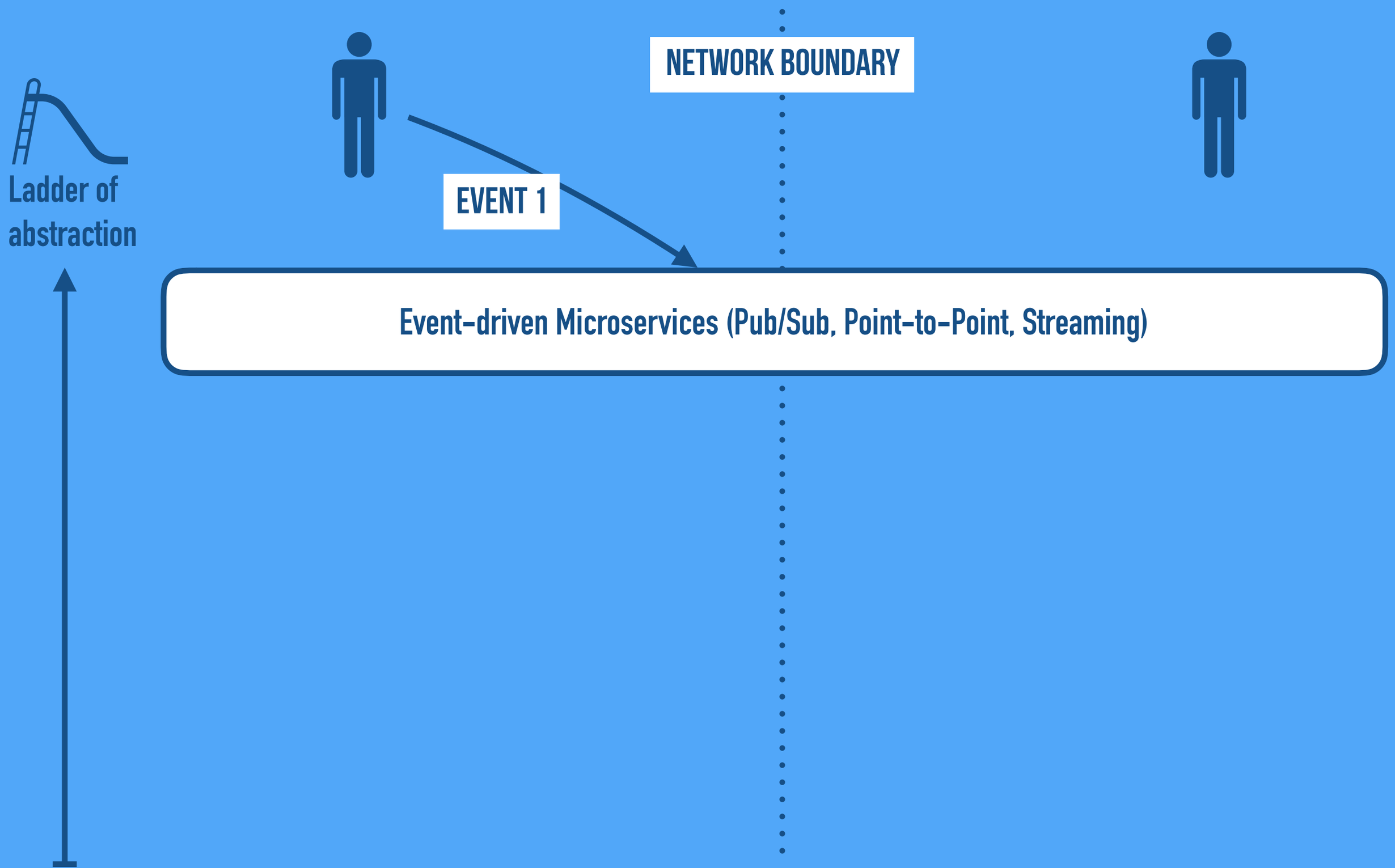
Event Driven Microservices
Powered By
Reactive Systems Fabric

Ladder of abstraction

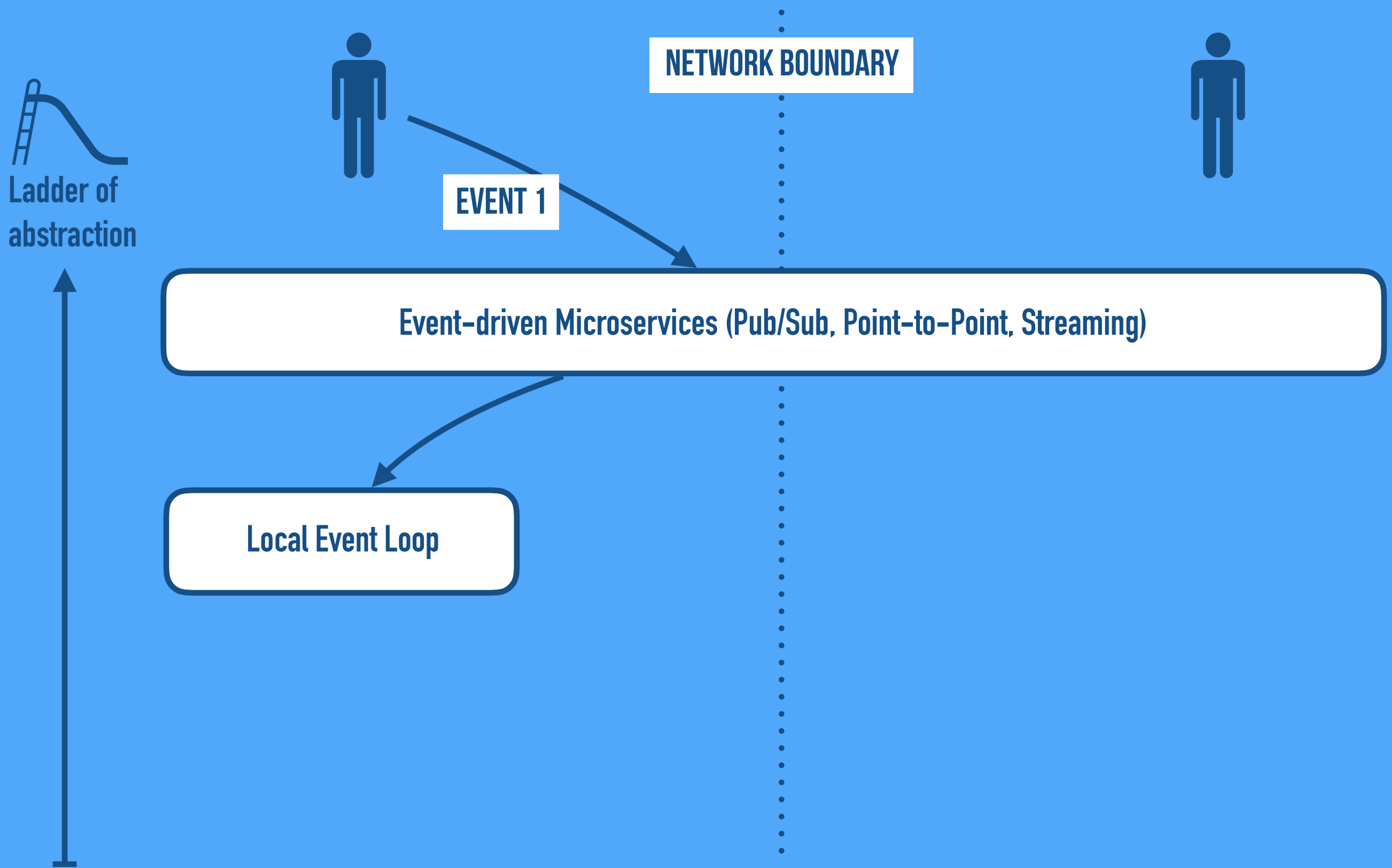
NETWORK BOUNDARY

Event-driven Microservices (Pub/Sub, Point-to-Point, Streaming)

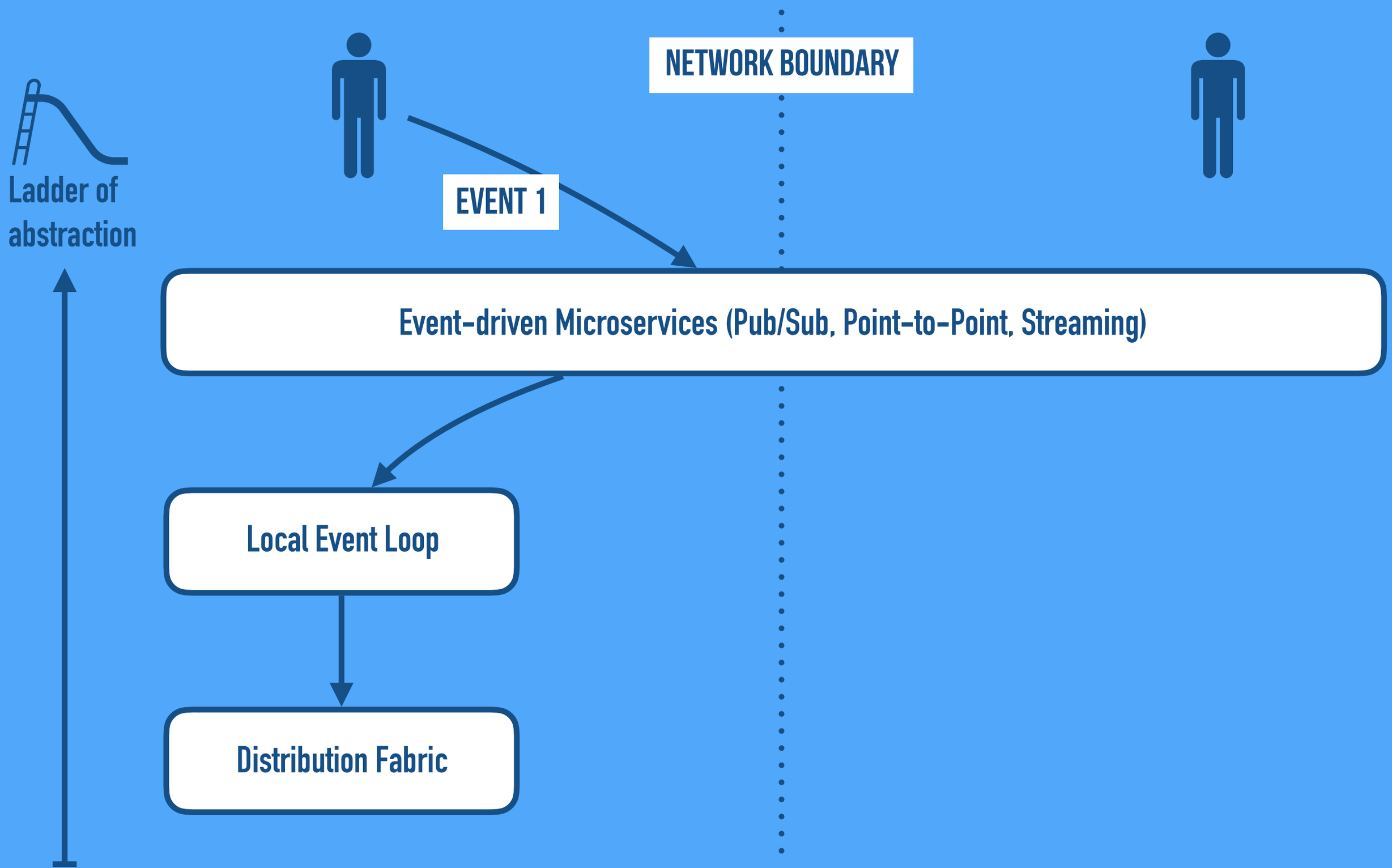
Event Driven Microservices
Powered By
Reactive Systems Fabric



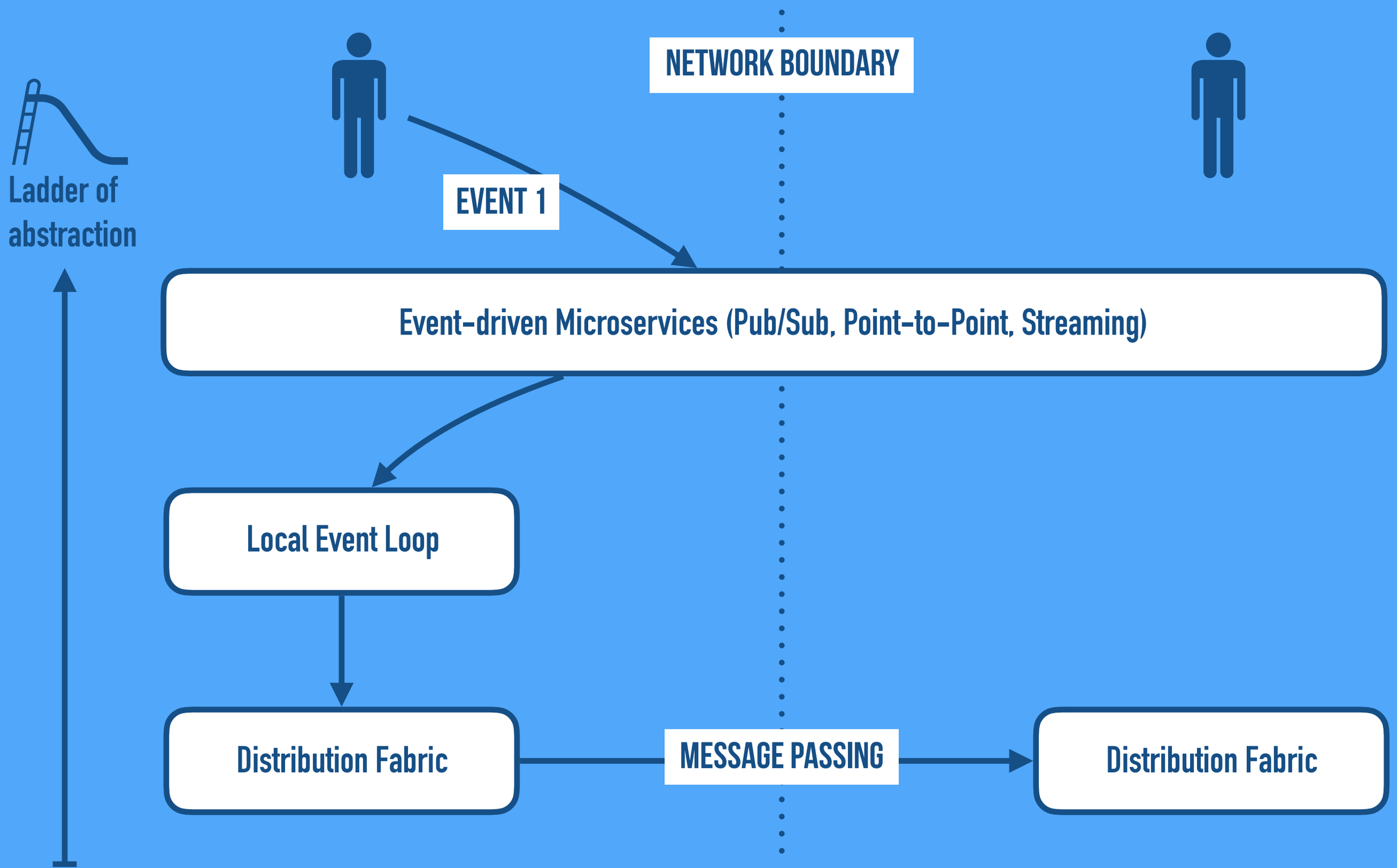
Event Driven Microservices
Powered By
Reactive Systems Fabric



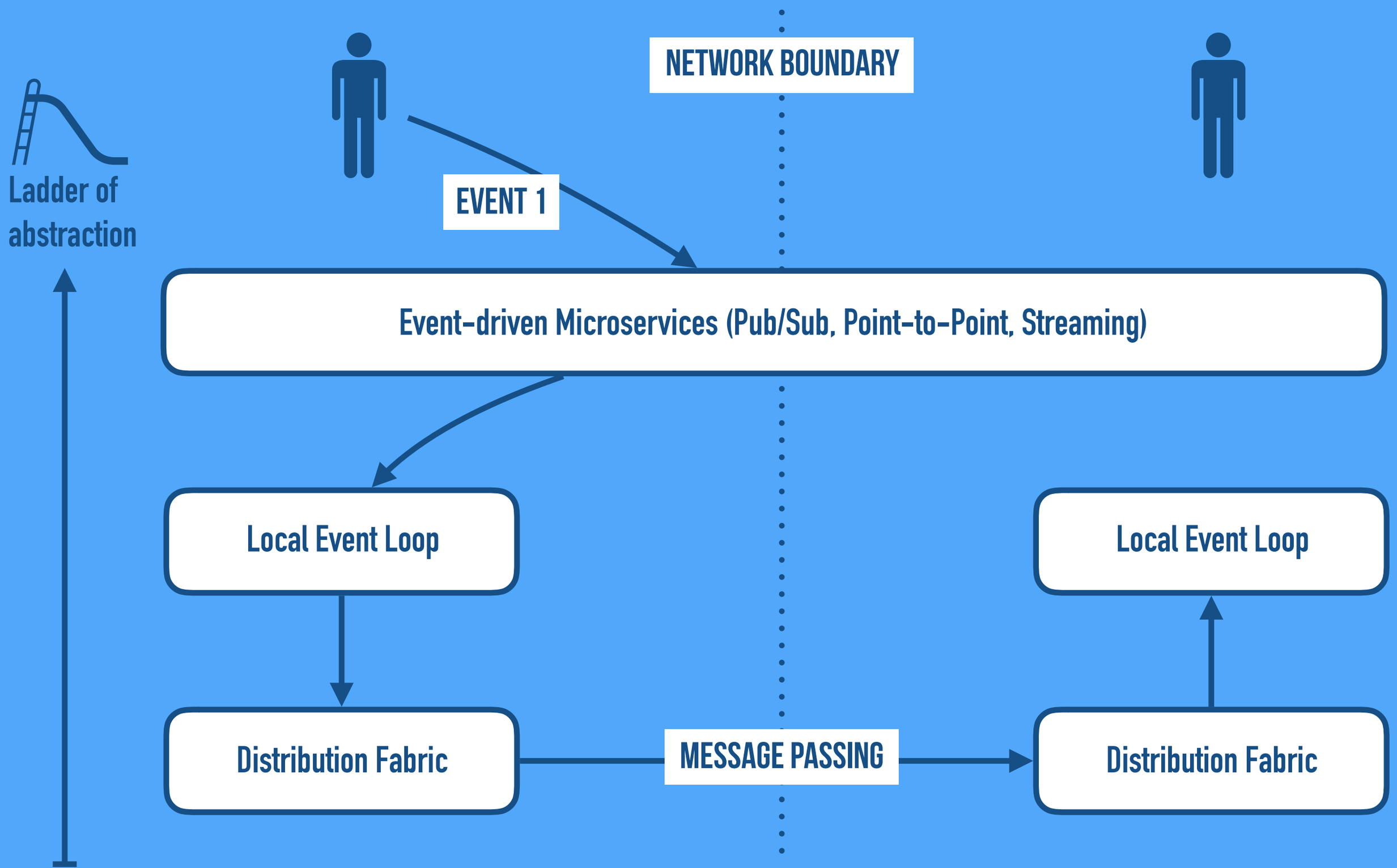
Event Driven Microservices
Powered By
Reactive Systems Fabric



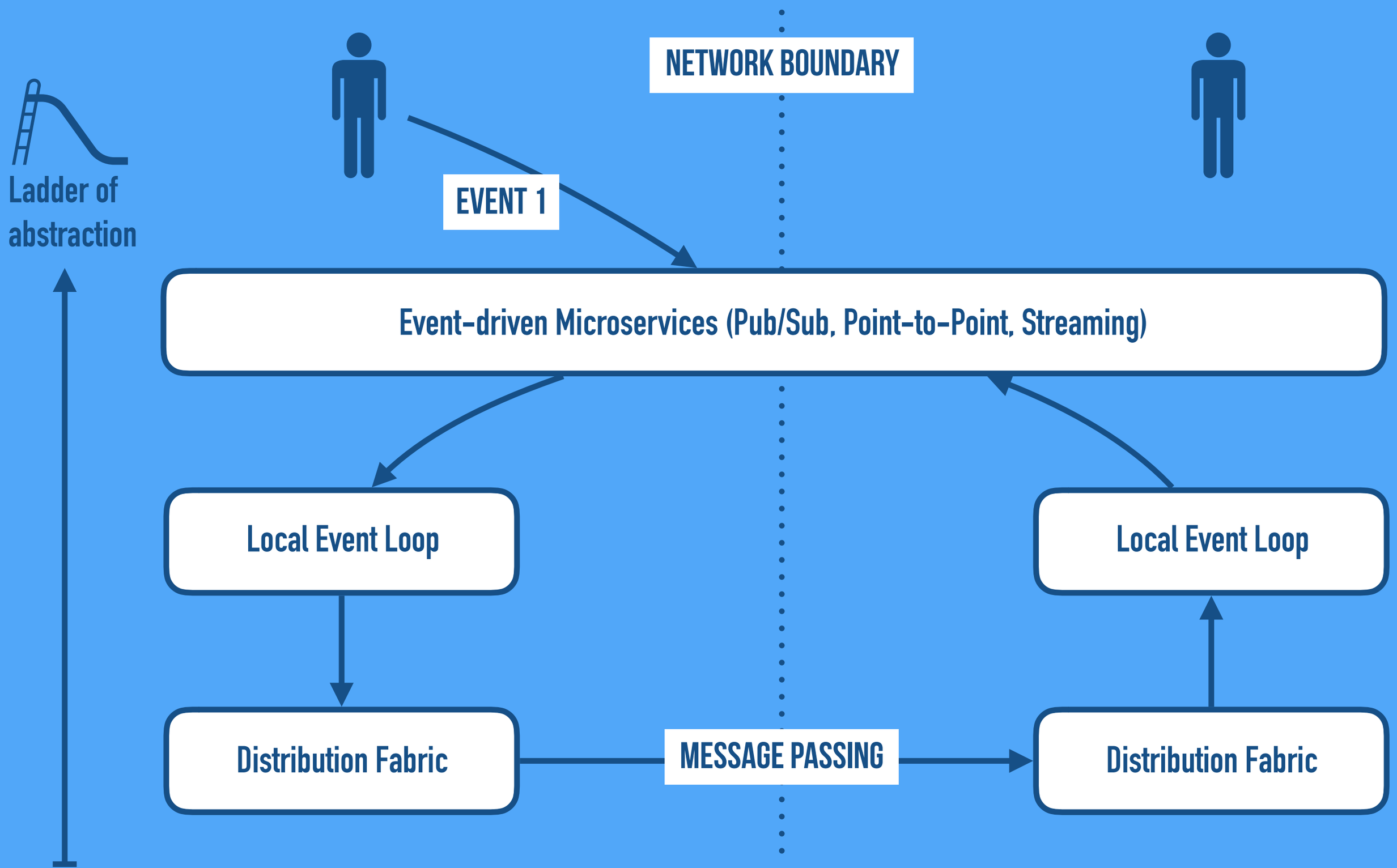
Event Driven Microservices
Powered By
Reactive Systems Fabric



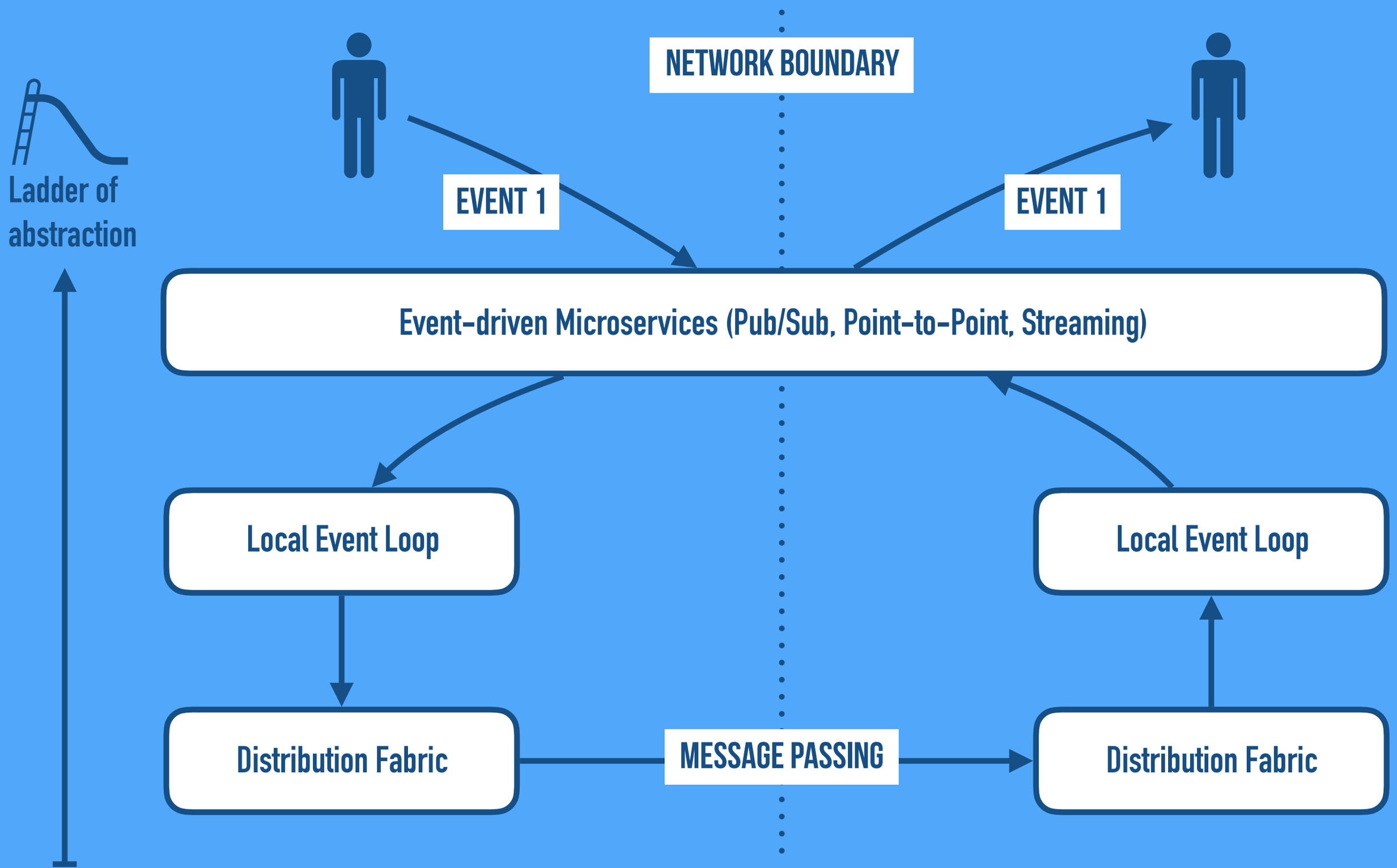
Event Driven Microservices
Powered By
Reactive Systems Fabric



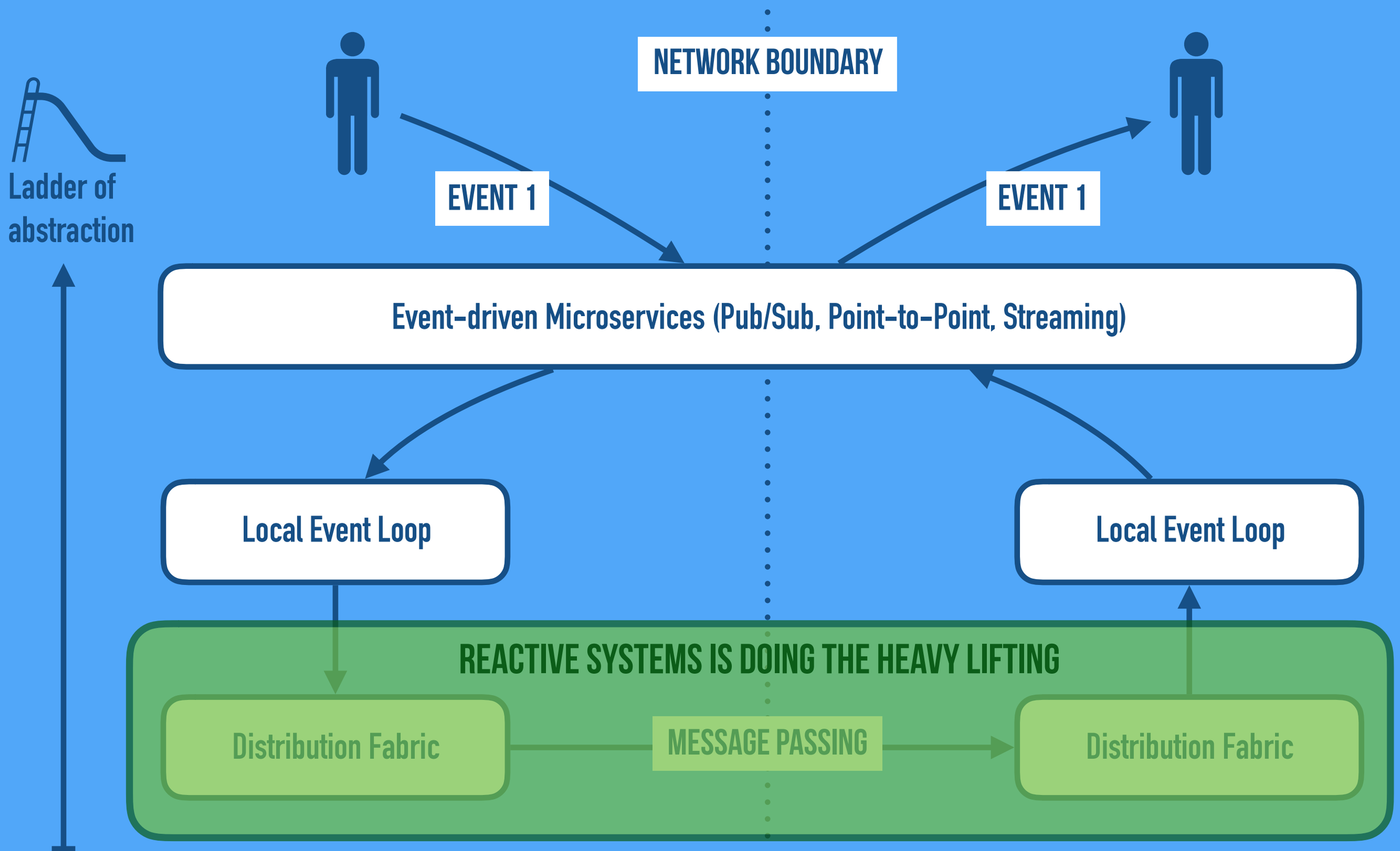
Event Driven Microservices
Powered By
Reactive Systems Fabric



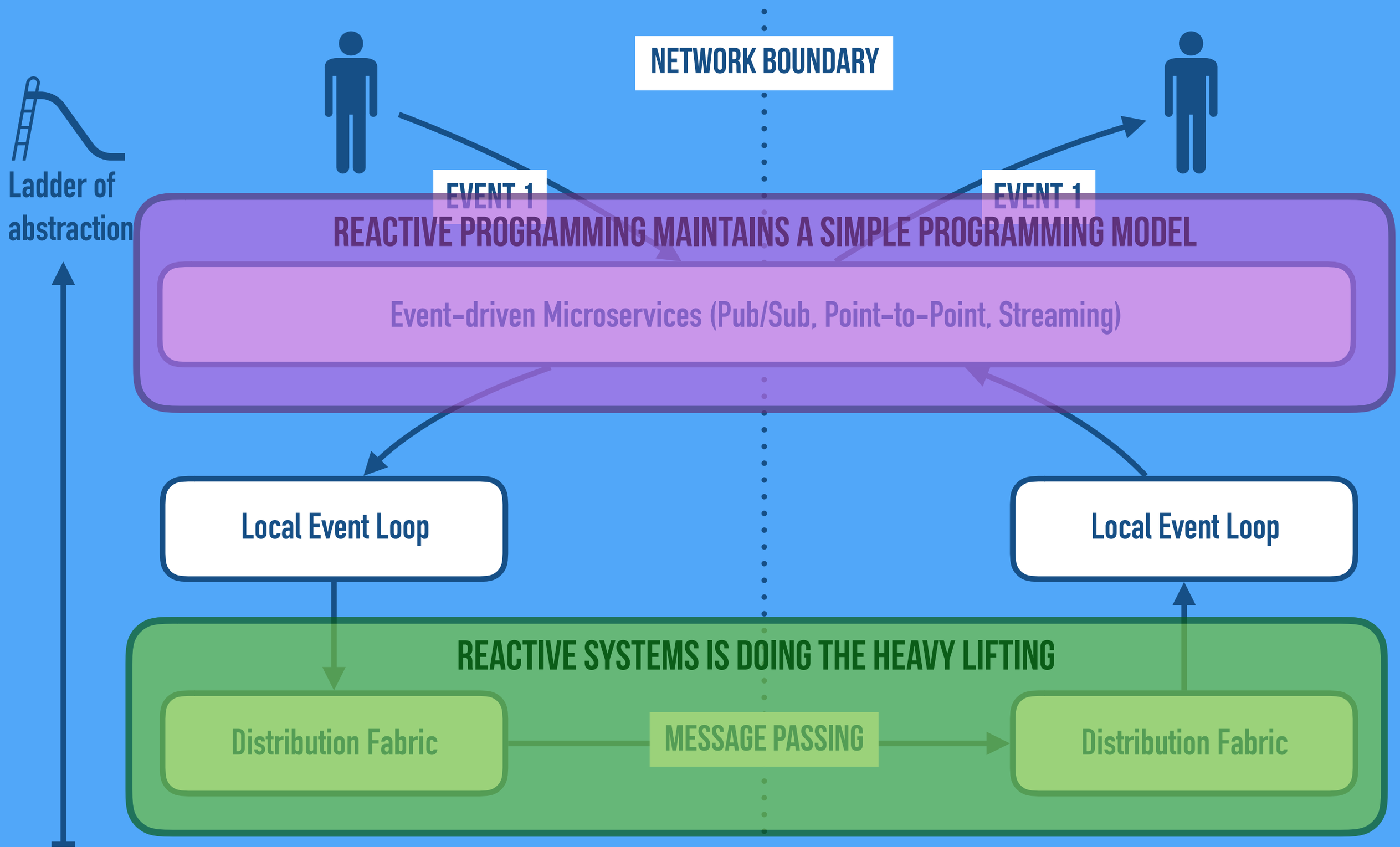
Event Driven Microservices
Powered By
Reactive Systems Fabric



Event Driven Microservices
Powered By
Reactive Systems Fabric



Event Driven Microservices
Powered By
Reactive Systems Fabric



Events First Domain Driven Design

“When you start modeling events, it forces you to think about the behavior of the system. As opposed to thinking about the structure of the system.”

- GREG YOUNG

A Decade of DDD, CQRS, Event Sourcing, Greg Young (Presentation from 2016)

- **DON'T FOCUS ON THE THINGS**

  The Nouns

  The Domain Objects

- **FOCUS ON WHAT HAPPENS**

  The Verbs

  The Events

Mine the Facts


Event Storming

Event Driven Design

Event Driven Design

✳ INTENTS

- ➡ Communication
- ➡ Conversations
- ➡ Expectations
- ➡ Contracts
- ➡ Control Transfer

Event Driven Design

✳ INTENTS

- ➡ Communication
- ➡ Conversations
- ➡ Expectations
- ➡ Contracts
- ➡ Control Transfer

✳ FACTS

- ➡ State
- ➡ History
- ➡ Causality
- ➡ Notifications
- ➡ State Transfer

Event Driven Design

✳ INTENTS

- ➡ Communication
- ➡ Expectations
- ➡ Contracts
- ➡ Control Transfer

commands

✳ FACTS

- ➡ State
- ➡ History
- ➡ Causality
- ➡ Notifications
- ➡ State Transfer

Event Driven Design

✳ INTENTS

- ➡ Communication
- ➡ Expectations
- ➡ Contracts
- ➡ Control Transfer

Commands

✳ FACTS

- ➡ State
- ➡ Causality
- ➡ Notifications
- ➡ State Transfer

Events

Event Driven Design

Event Driven Design

- **COMMANDS**
  - ➡ Object form of **METHOD/ACTION REQUEST**
  - ➡ **IMPERATIVE:** `CreateOrder`, `ShipProduct`

Event Driven Design

- **COMMANDS**
  - ➡ Object form of **METHOD/ACTION REQUEST**
  - ➡ **IMPERATIVE:** `CreateOrder`, `ShipProduct`
- **REACTIONS**
  - ➡ Represents **SIDE-EFFECTS**

Event Driven Design

- **COMMANDS**
  - ➡ Object form of **METHOD/ACTION REQUEST**
  - ➡ **IMPERATIVE:** `CreateOrder`, `ShipProduct`
- **REACTIONS**
  - ➡ Represents **SIDE-EFFECTS**
- **EVENTS**
  - ➡ Represents something that **HAS HAPPENED**
  - ➡ **PAST-TENSE:** `OrderCreated`, `ProductShipped`

COMMANDS vs EVENTS

COMMANDS vs EVENTS

1. All about intent

1. Intentless

COMMANDS vs EVENTS

1. All about intent
2. Directed

1. Intentless
2. Anonymous

COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination

1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe

COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination
4. Models personal communication

1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe
4. Models broadcast (speakers corner)

COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination
4. Models personal communication
5. Distributed focus

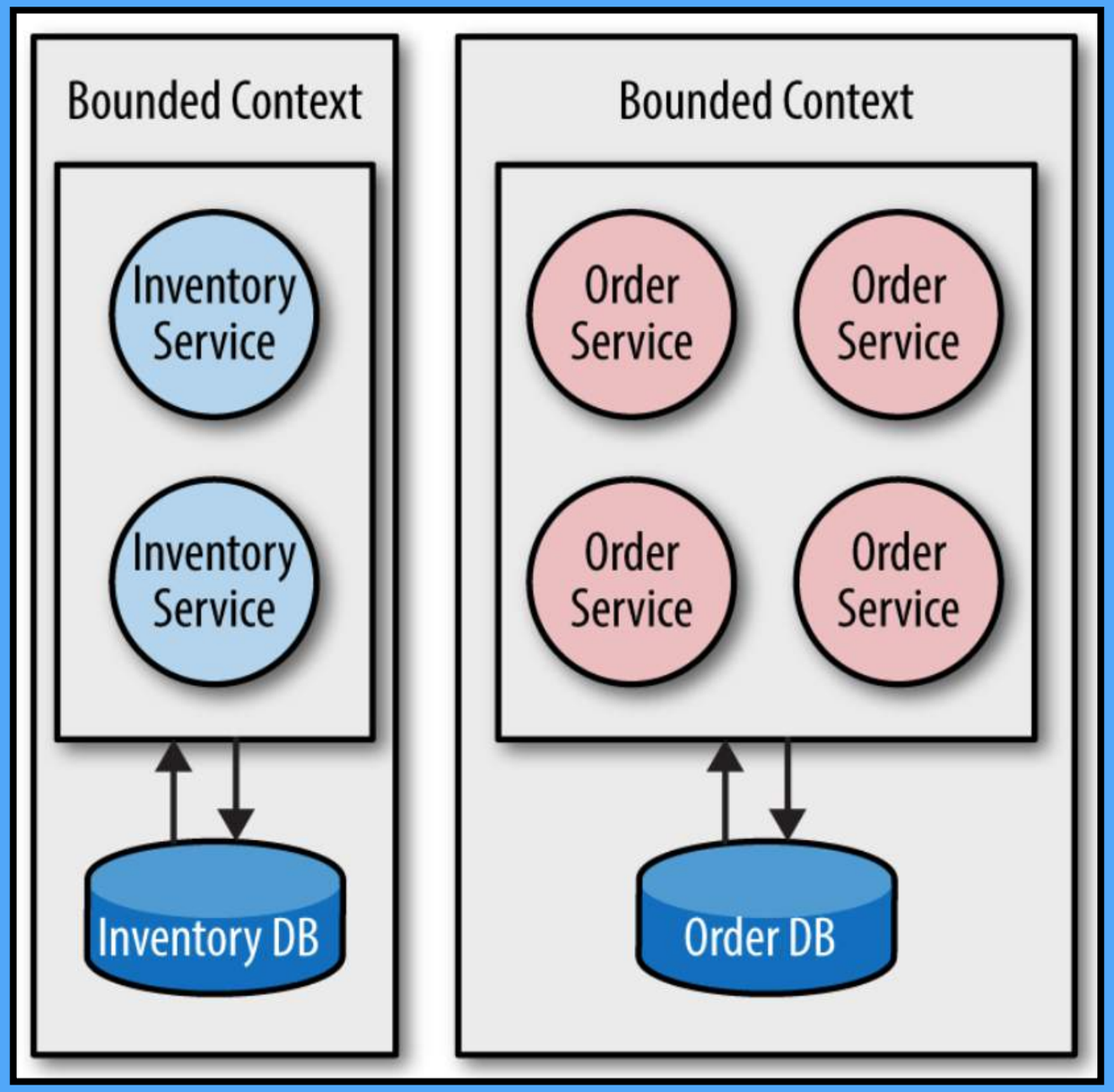
1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe
4. Models broadcast (speakers corner)
5. Local focus

COMMANDS vs EVENTS

1. All about intent
2. Directed
3. Single addressable destination
4. Models personal communication
5. Distributed focus
6. Command & Control

1. Intentless
2. Anonymous
3. Just happens – for others (0–N) to observe
4. Models broadcast (speakers corner)
5. Local focus
6. Autonomy

Let the Events Define the Bounded Context



ARE WE DONE NOW?

Perhaps. We have come a long way.
But events can also be used for:

- ➡ PERSISTENCE
- ➡ MANAGING TIME

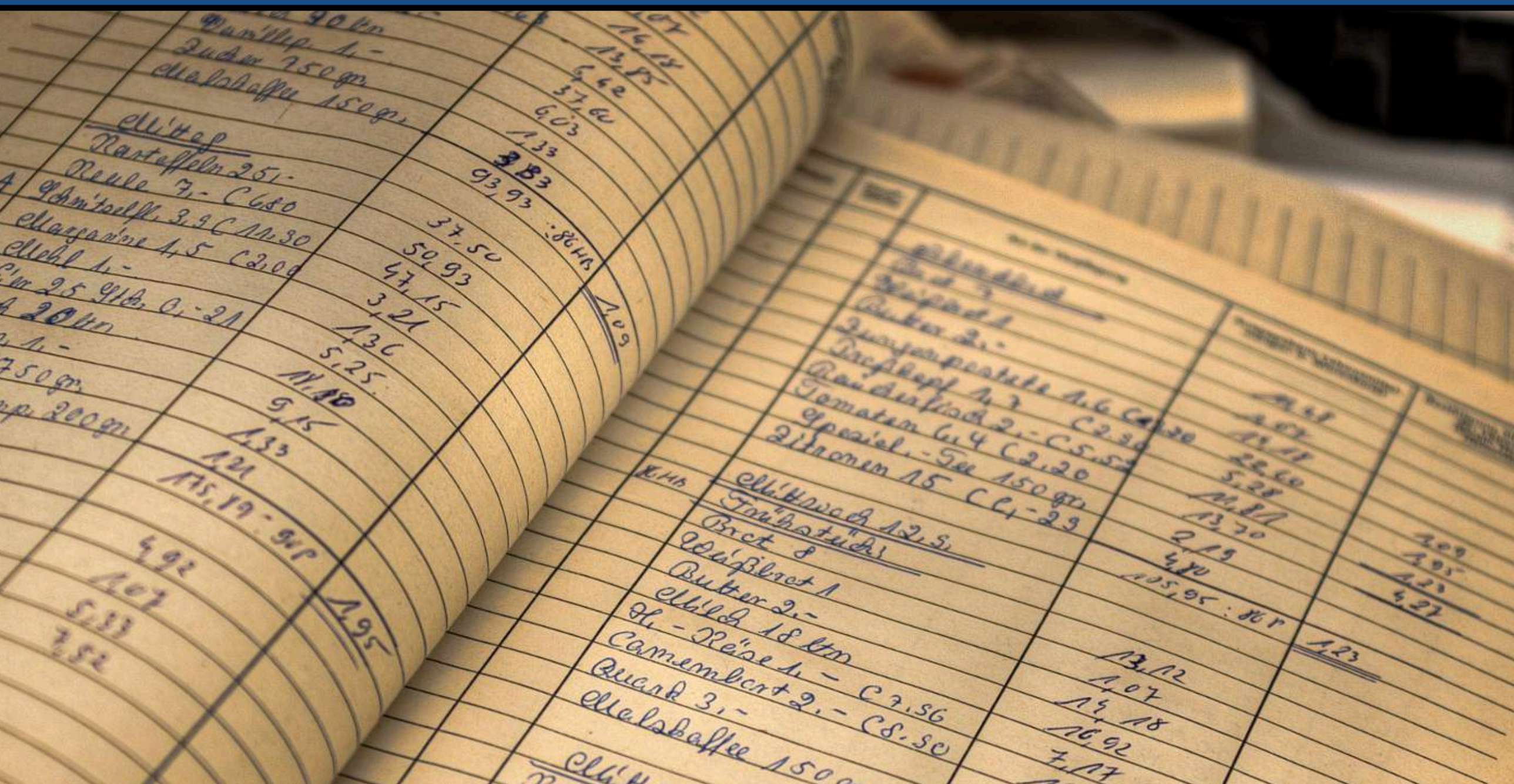
Event Based Persistence

“Update-in-place strikes systems designers as a cardinal sin: it violates traditional accounting practices that have been observed for hundreds of years.”

- JIM GRAY

The Transaction Concept, Jim Gray (1981)

Event Logging
The Bedrock



“The truth is the log.
The database is a cache
of a subset of the log.”

- PAT HELLAND

Immutability Changes Everything, Pat Helland (2015)

The Log

A Database Of the Past
Not Just the Present

Event Sourcing

A Cure For the Cardinal Sin

Event Sourced Services

Event Sourced Services

HAPPY PATH

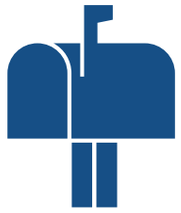
Event Sourced Services

HAPPY PATH

1) Receive and verify Command
("ApprovePayment")

Event Sourced Services

HAPPY PATH

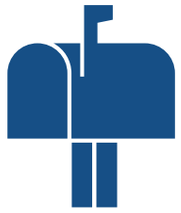


1) Receive and verify Command
("ApprovePayment")

2) Create new Event
("PaymentApproved")

Event Sourced Services

HAPPY PATH



1) Receive and verify Command ("ApprovePayment")

2) Create new Event ("PaymentApproved")

3) Append Event to Event Log

Event Sourced Services

HAPPY PATH

1) Receive and verify Command ("ApprovePayment")

2) Create new Event ("PaymentApproved")

3) Append Event to Event Log

4) Update internal component state

Event Sourced Services

HAPPY PATH

1) Receive and verify Command ("ApprovePayment")

2) Create new Event ("PaymentApproved")

3) Append Event to Event Log

4) Update internal component state

5) Run side-effects (approve the payment)

Event Sourced Services

HAPPY PATH

1) Receive and verify Command ("ApprovePayment")

2) Create new Event ("PaymentApproved")

3) Append Event to Event Log

4) Update internal component state

5) Run side-effects (approve the payment)

SAD PATH - RECOVERING FROM FAILURE

Event Sourced Services

HAPPY PATH

1) Receive and verify Command ("ApprovePayment")

2) Create new Event ("PaymentApproved")

3) Append Event to Event Log

4) Update internal component state

5) Run side-effects (approve the payment)

SAD PATH - RECOVERING FROM FAILURE

1) Rehydrate Events from Event Log

Event Sourced Services

HAPPY PATH

1) Receive and verify Command ("ApprovePayment")

2) Create new Event ("PaymentApproved")

3) Append Event to Event Log

4) Update internal component state

5) Run side-effects (approve the payment)

SAD PATH - RECOVERING FROM FAILURE

1) Rehydrate Events from Event Log

2) Update internal component state

Event Sourced Services

Memory Image

HAPPY P

...ymentApproved")

...) Append Event to Event Log

4) Update internal component state

5) Run side-effects (approve the payment)

SAD PATH - RECOVERING FROM FAILURE

1) Rehydrate Events from Event Log

2) Update internal component state

BENEFITS OF USING

Event Sourcing

BENEFITS OF USING
Event Sourcing

- One single SOURCE OF TRUTH with ALL HISTORY

BENEFITS OF USING
Event Sourcing

- One single SOURCE OF TRUTH with ALL HISTORY
- Allows for MEMORY IMAGE - durable in-memory state

BENEFITS OF USING
Event Sourcing

- One single SOURCE OF TRUTH with ALL HISTORY
- Allows for MEMORY IMAGE - durable in-memory state
- Avoids the OBJECT-RELATIONAL IMPEDENCE MISMATCH

BENEFITS OF USING
Event Sourcing

- One single SOURCE OF TRUTH with ALL HISTORY
- Allows for MEMORY IMAGE - durable in-memory state
- Avoids the OBJECT-RELATIONAL IMPEDENCE MISMATCH
- Allows others to SUBSCRIBE TO STATE CHANGES

BENEFITS OF USING
Event Sourcing

- One single SOURCE OF TRUTH with ALL HISTORY
- Allows for MEMORY IMAGE - durable in-memory state
- Avoids the OBJECT-RELATIONAL IMPEDENCE MISMATCH
- Allows others to SUBSCRIBE TO STATE CHANGES
- Mechanical SYMPATHY (Single Writer Principle etc.)

Disadvantages
Of Using Event Sourcing

- **UNFAMILIAR** model
- **VERSIONING** of events
- **DELETION** of events (legal or moral reasons)

Events

Allow Us To Manage

Time

“Modeling events forces you to have a temporal focus on what’s going on in the system. Time becomes a crucial factor of the system.”

- GREG YOUNG

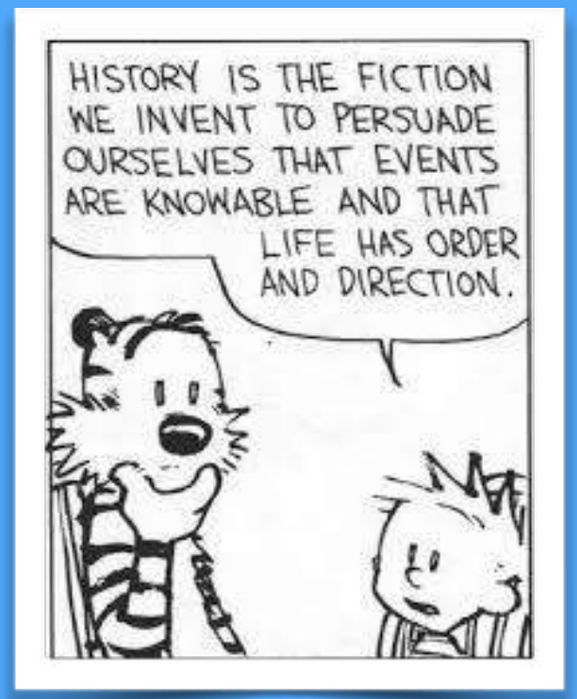
A Decade of DDD, CQRS, Event Sourcing, Greg Young (Presentation from 2016)

Event Sourcing
Allows Us To
Model Time

- Event is a **SNAPSHOT** IN TIME
- Event ID is an **INDEX** FOR TIME
- Event Log is our FULL **HISTORY**

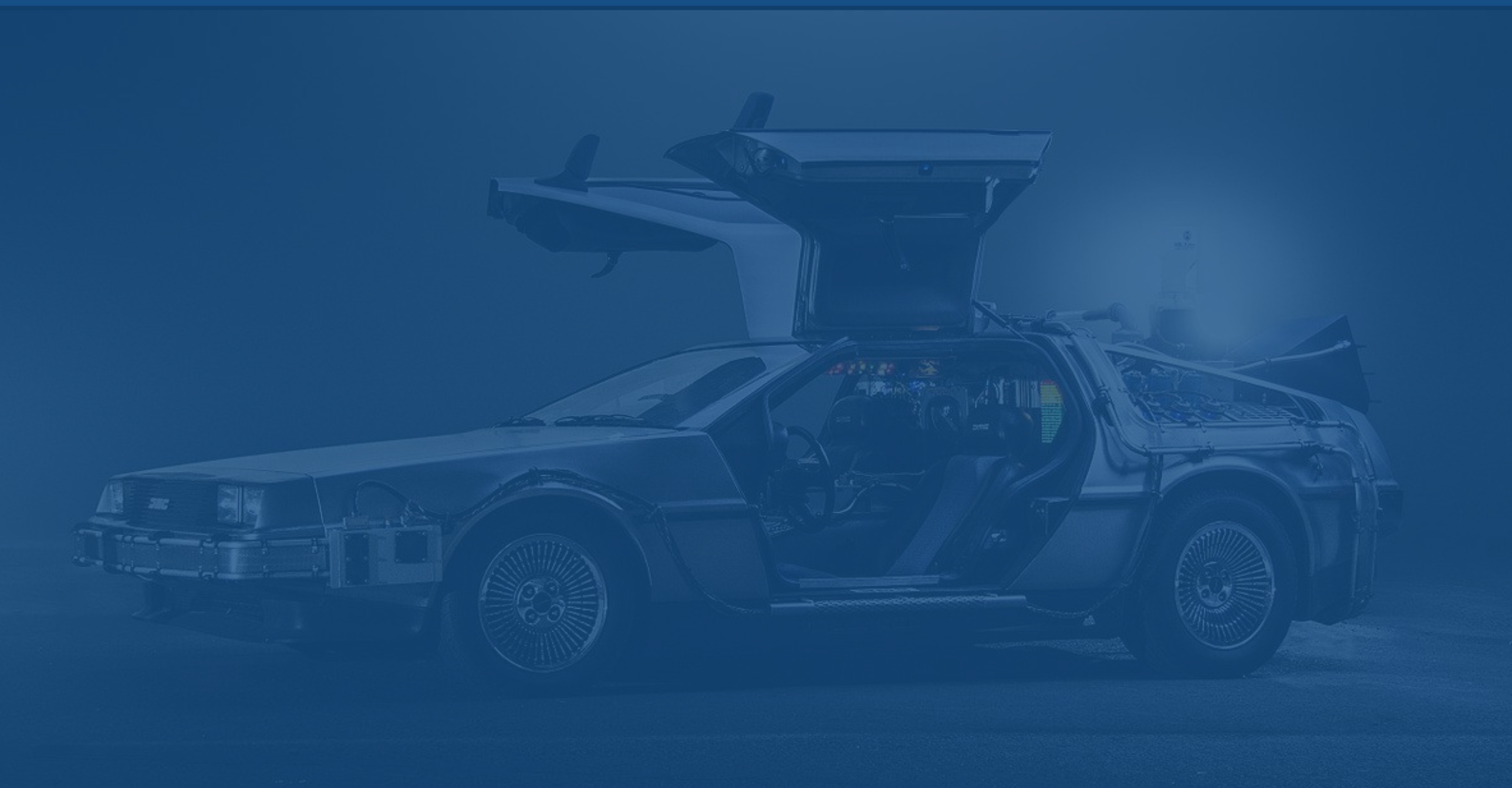
  The DATABASE OF OUR **PAST**

  The PATH TO OUR **PRESENT**



Event Sourcing
Allows For
Time Travel

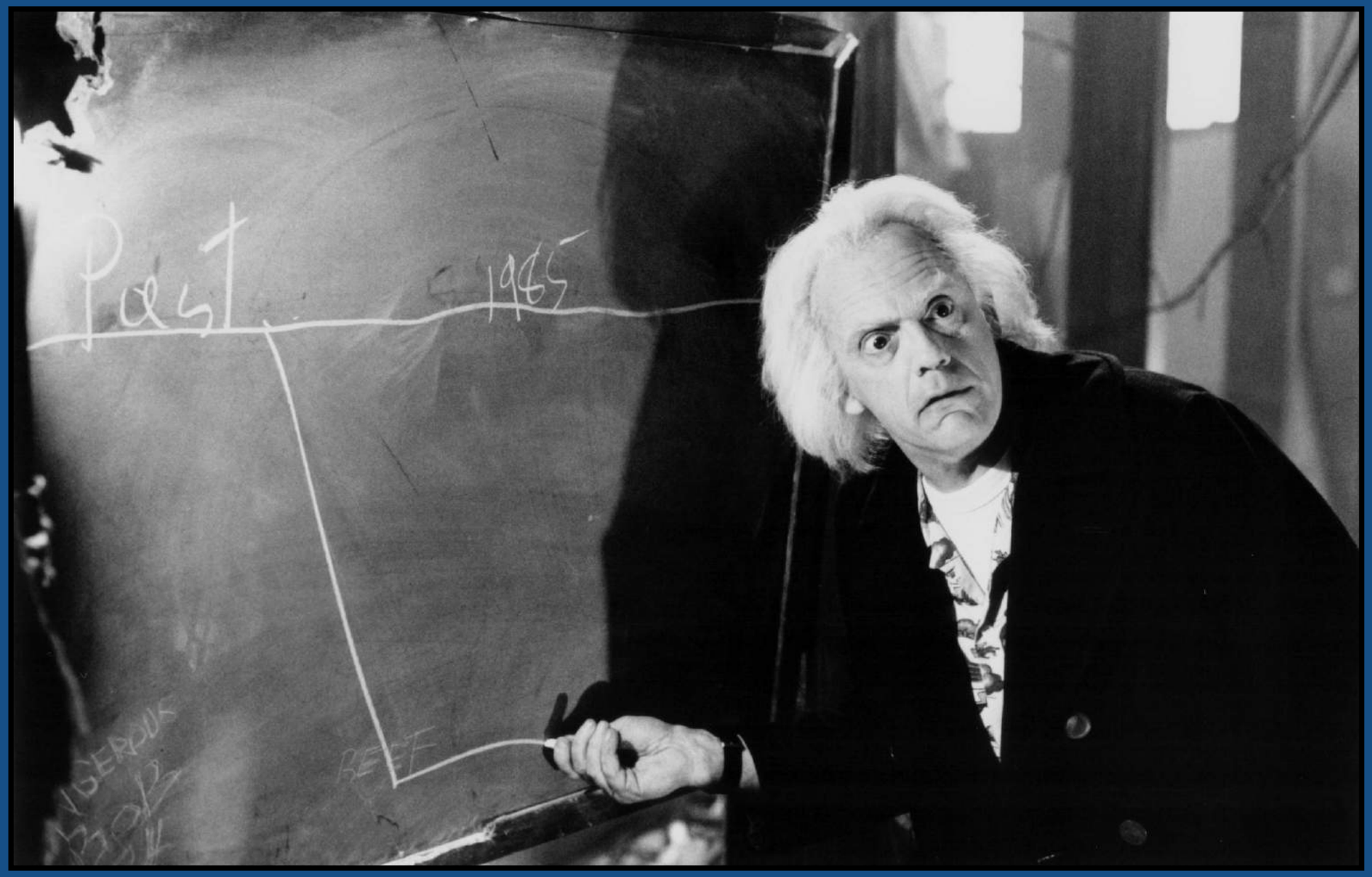
Event Sourcing Allows For Time Travel



Event Sourcing Allows For Time Travel

- ✳ Replay the log FOR HISTORIC DEBUGGING
- ✳ Replay the log FOR AUDITING & TRACEABILITY
- ✳ Replay the log ON FAILURE
- ✳ Replay the log FOR REPLICATION

We Can Even Fork the Past



...Or Join Two Distinct Pasts

Key Takeaways

Key Takeaways

1. Don't build Microliths

Key Takeaways

1. Don't build Microliths
2. Microservices come in (distributed) systems

Key Takeaways

1. Don't build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems

Key Takeaways

1. Don't build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles

Key Takeaways

1. Don't build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events-first Domain Driven Design

Key Takeaways

1. Don't build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events-first Domain Driven Design
6. Embrace Event-driven Architecture

Key Takeaways

1. Don't build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events-first Domain Driven Design
6. Embrace Event-driven Architecture
7. Embrace Event-driven Persistence

Key Takeaways

1. Don't build Microliths
2. Microservices come in (distributed) systems
3. Microservices come as (micro)systems
4. Embrace the Reactive Principles
5. Embrace Events-first Domain Driven Design
6. Embrace Event-driven Architecture
7. Embrace Event-driven Persistence
8. Profit!!!

Learn More

Download my latest book for free at:

bit.ly/reactive-microsystems