

# Designing Events First Microservices

Jonas Bonér

@jboner



Lightbend

# WHY SHOULD YOU CARE ABOUT EVENTS?

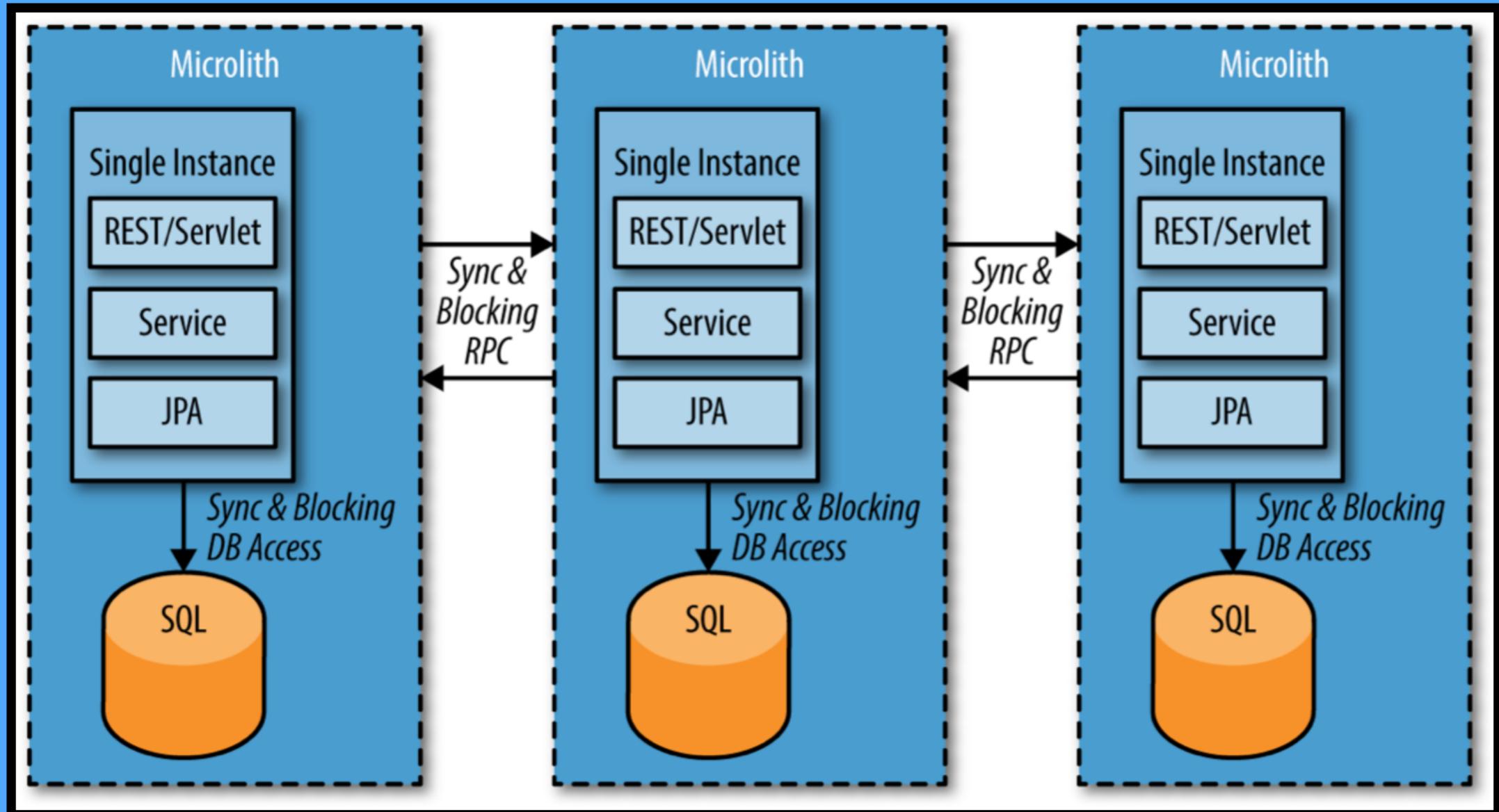
1. Events **DRIVE AUTONOMY**
2. Events **HELP REDUCE RISK**
3. Events **HELP YOU MOVE FASTER**
4. Events **INCREASE LOOSE COUPLING**
5. Events **INCREASE STABILITY**
6. Events **INCREASE SCALABILITY**
7. Events **INCREASE RESILIENCE**
8. Events **INCREASE TRACEABILITY**
9. Events **ALLOW FOR TIME-TRAVEL**

# WHY NOW?

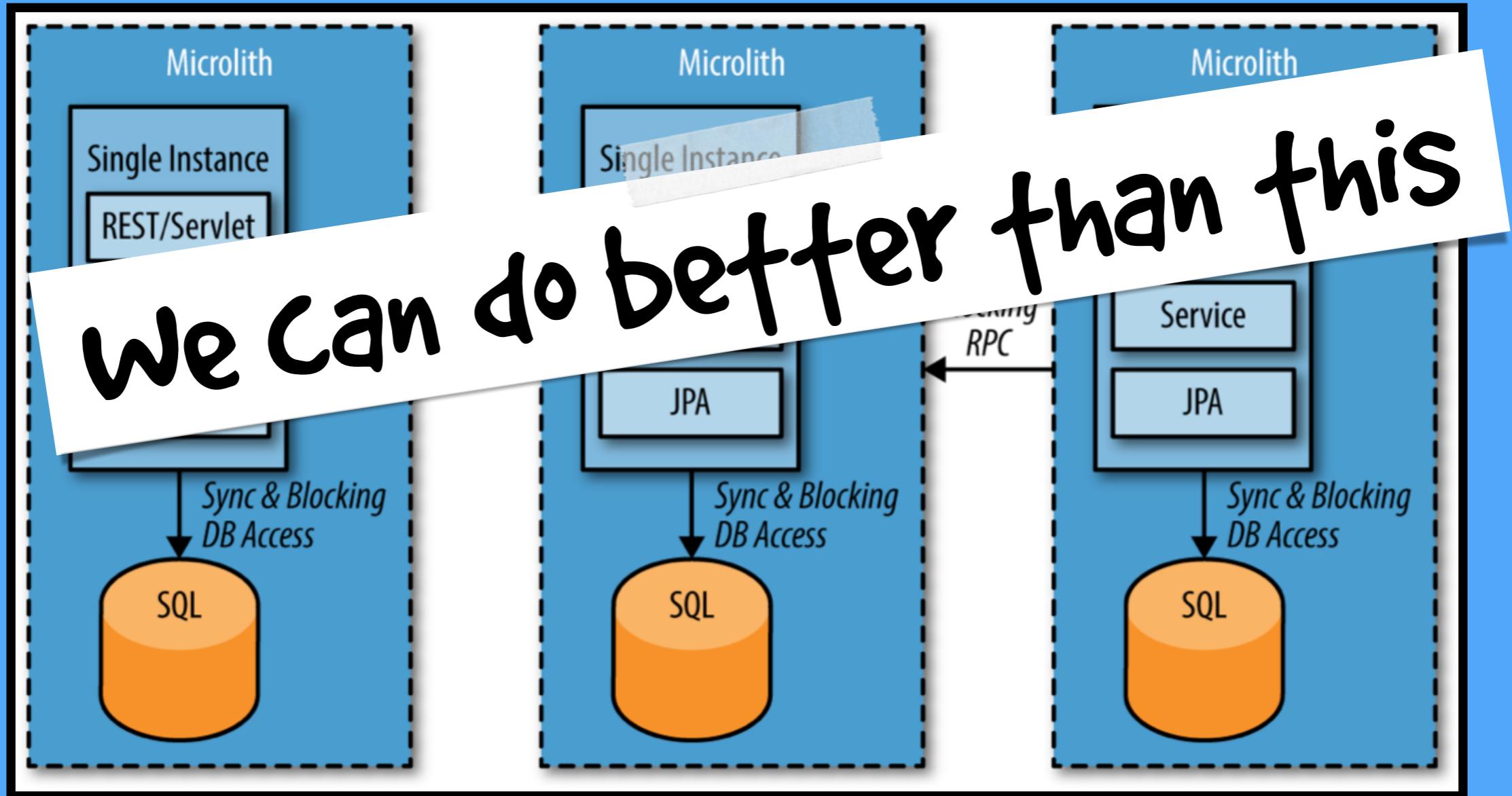
1. Cloud and multicore architectures
2. Microservices and distributed systems
3. Data-centric applications – Data in motion
4. “We want more, of everything, and we want it now.” -Your Customers

**SO, YOU WANT TO DO  
MICROSERVICES?**

# MAKE SURE YOU DON'T END UP WITH MICROLITHS



# MAKE SURE YOU DON'T END UP WITH MICROLITHS



Events First  
Domain Driven  
Design

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

**- GREG YOUNG**



**DON'T FOCUS ON THE THINGS**

The Nouns

The Domain Objects



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

The Nouns

The Domain Objects



# **FOCUS ON WHAT HAPPENS**

The Verbs

The Events

**WHAT IS AN  
EVENT?**

# 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**

A silhouette of a detective wearing a fedora hat and a trench coat, smoking a pipe and holding a magnifying glass.

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

Commands

Expectations

Contracts

Control Transfer

## \* FACTS

→ State

→ History

→ Causality

→ Notifications

→ State Transfer

# Event Driven Design

## \* INTENTS

→ Communication

Commands

Expectations

→ Contracts

→ Control Transfer

## \* FACTS

→ State

Events

Causality

→ Notifications

→ State Transfer

# 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)

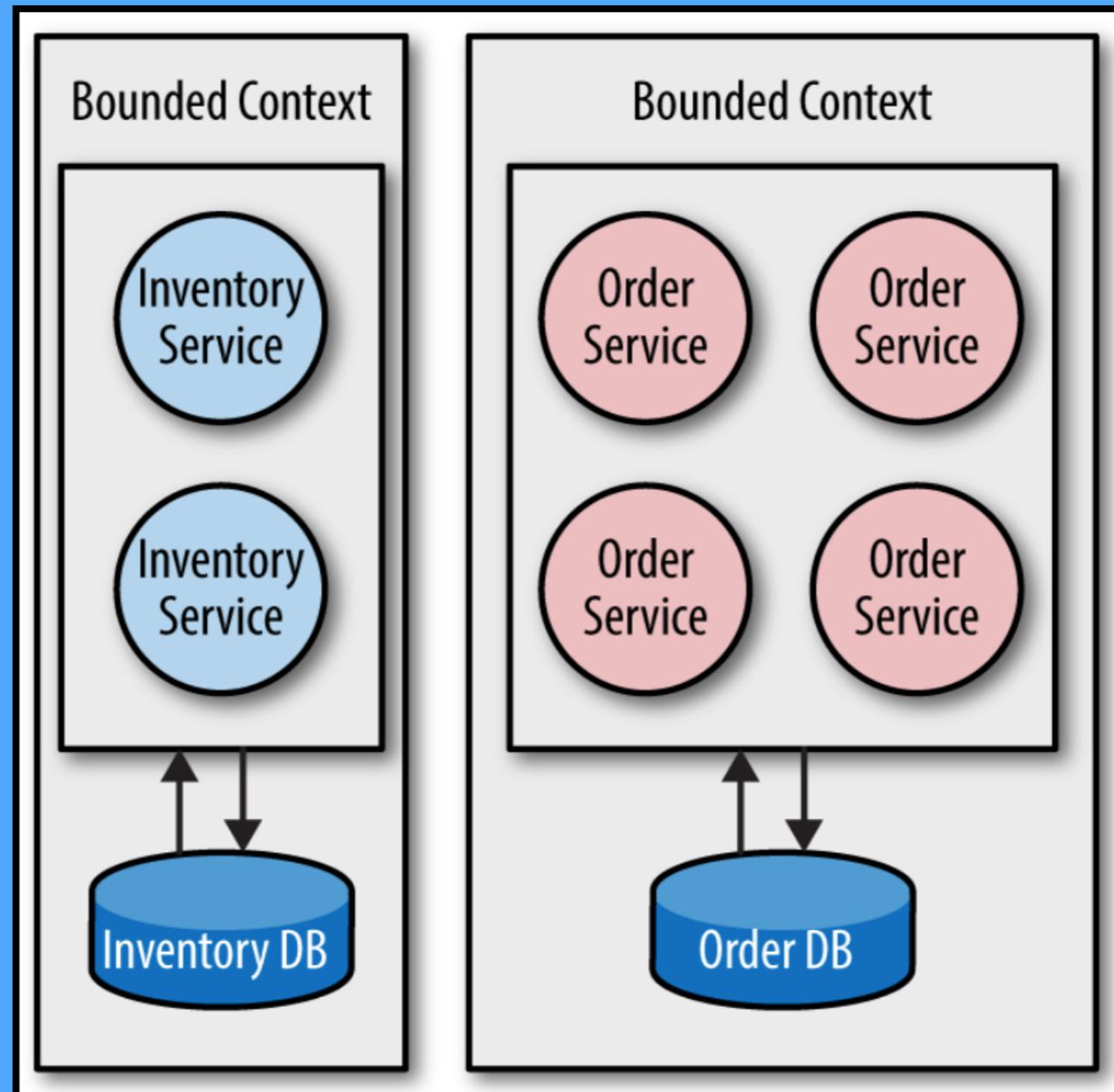
# COMMANDS vs EVENTS

- 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



# Event Driven Services

# Event Driven Services

**1. RECEIVE & REACT (or not)**

**TO FACTS\* that are coming its way**

# Event Driven Services

**1. RECEIVE & REACT (or not)**

**TO FACTS\* that are coming its way**

**2. PUBLISH NEW FACTS\* ASYNCHRONOUSLY**

**to the rest of the world**

# Event Driven Services

**1. RECEIVE & REACT (or not)**

**TO FACTS\* that are coming its way**

**2. PUBLISH NEW FACTS\* ASYNCHRONOUSLY**

**to the rest of the world**

**3. INVERT THE CONTROL FLOW**

**to minimize coupling and increase autonomy**

# Event Driven Services

**1. RECEIVE & REACT (or not)**

**TO FACTS\* that are coming its way**

**2. PUBLISH NEW FACTS\* ASYNCHRONOUSLY**

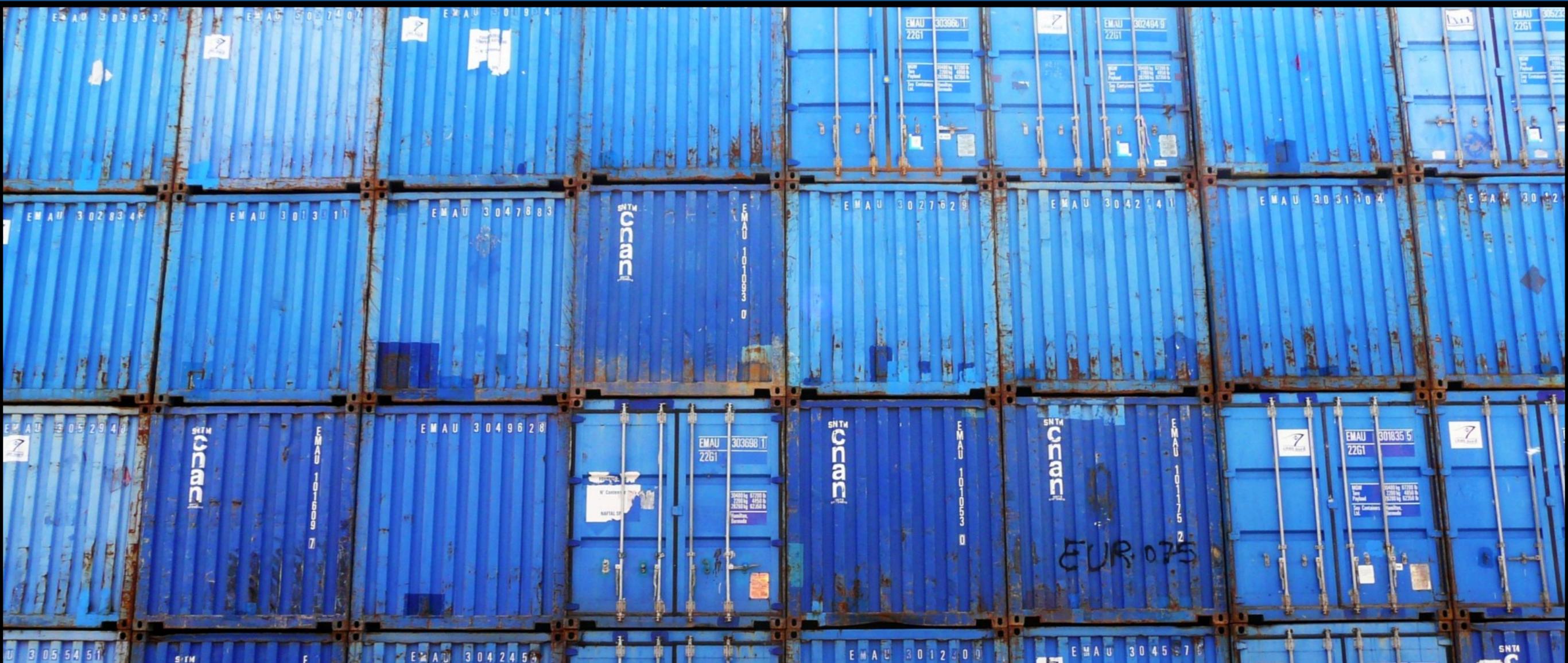
**to the rest of the world**

**3. INVERT THE CONTROL FLOW**

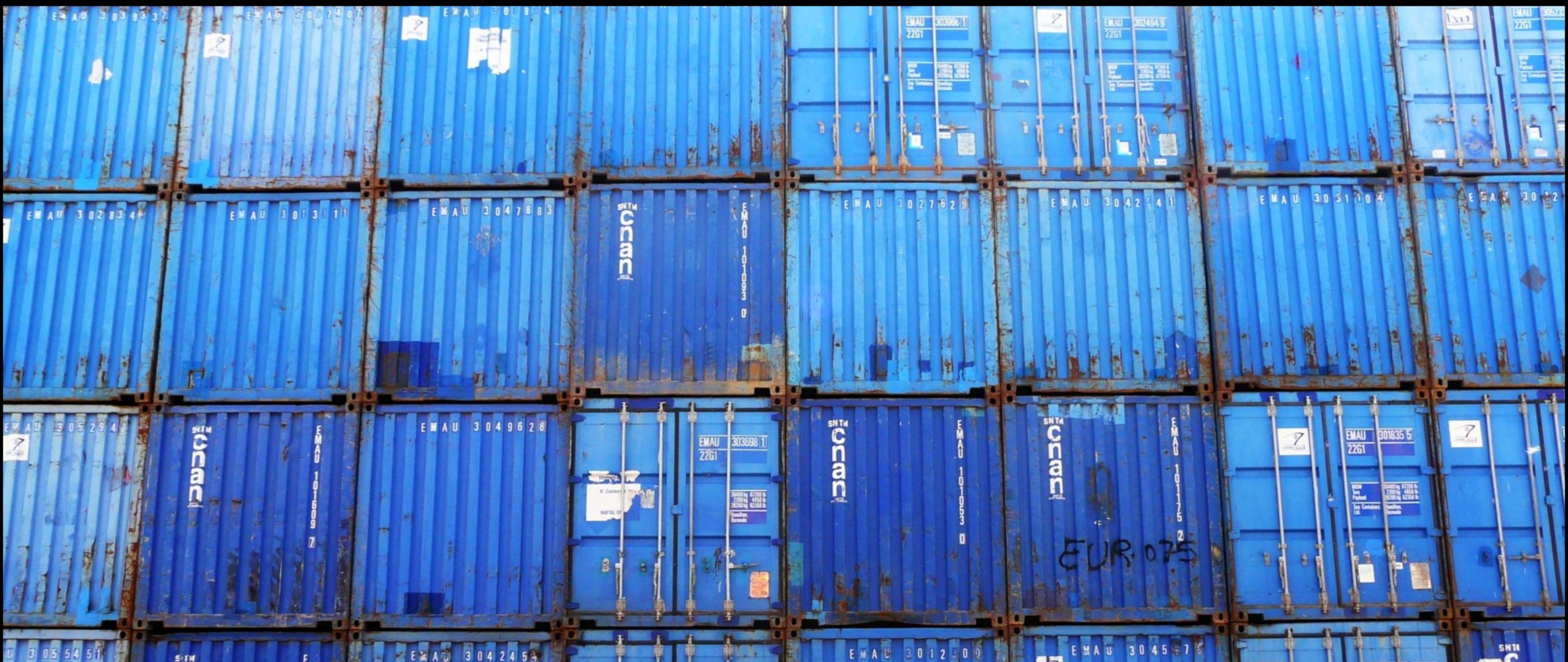
**to minimize coupling and increase autonomy**

**\*Facts == Immutable Events**

# Mutable State Is Fine But Needs To Be Contained And Non Observable



# Publish Facts To Outside World

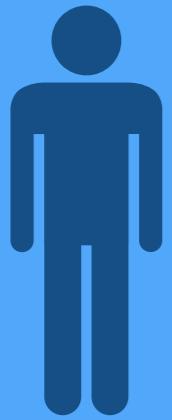


# The Aggregate

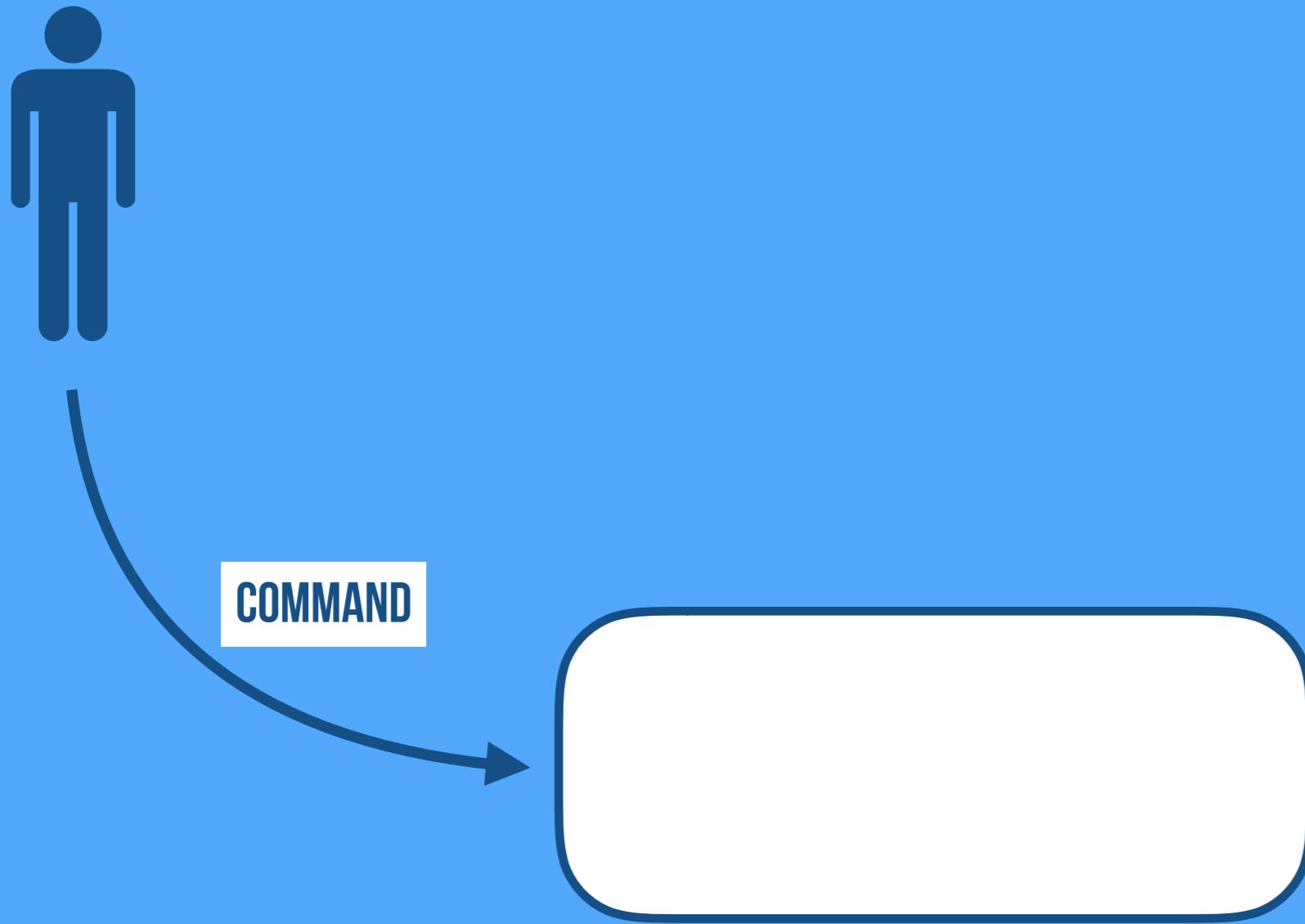
- \* Maintains INTEGRITY & CONSISTENCY
- \* Is our UNIT OF CONSISTENCY
- \* Is our UNIT OF FAILURE
- \* Is our UNIT OF DETERMINISM
- \* Is fully AUTONOMOUS

# Event Driven Services

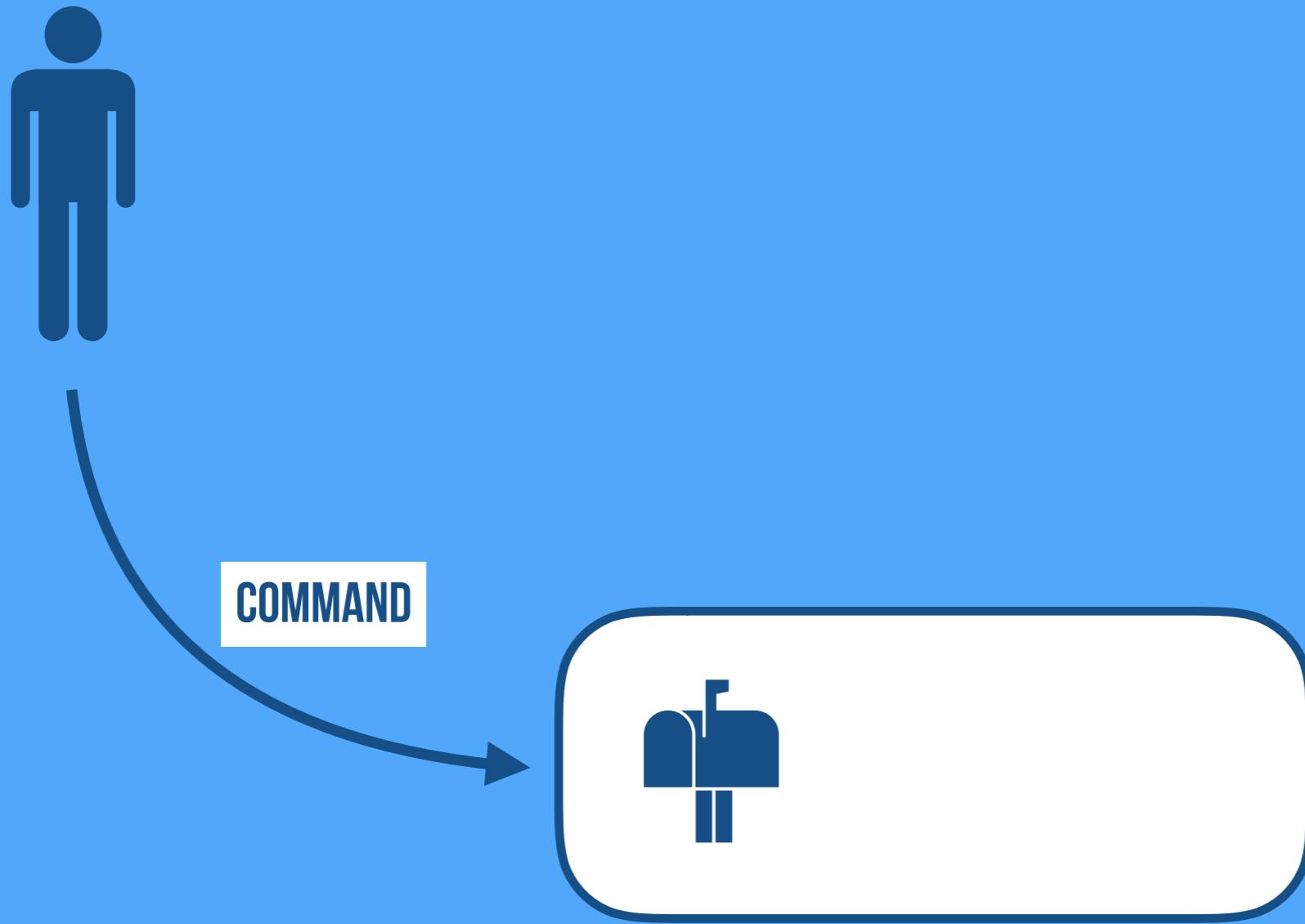
# Event Driven Services



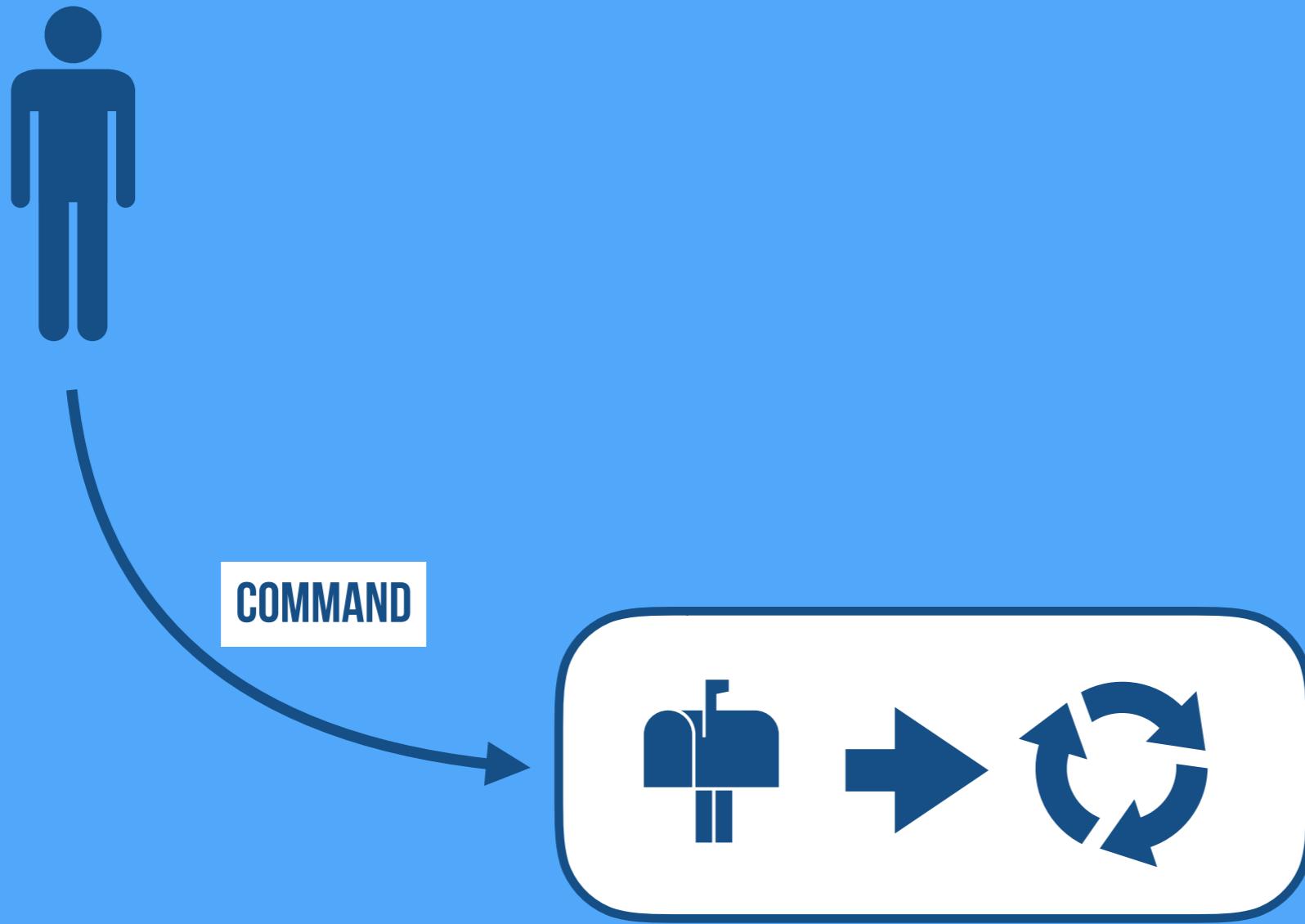
# Event Driven Services



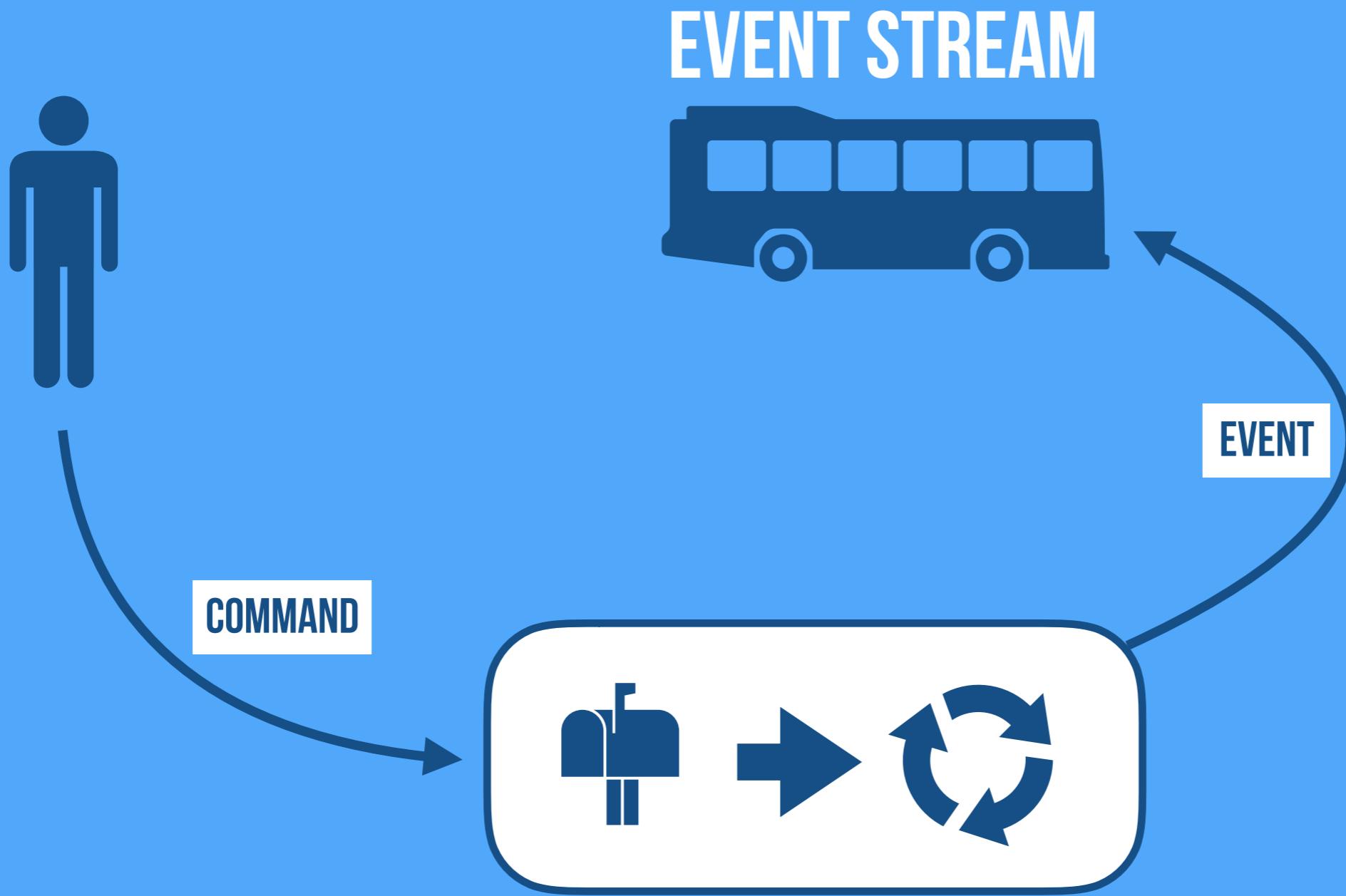
# Event Driven Services



# Event Driven Services



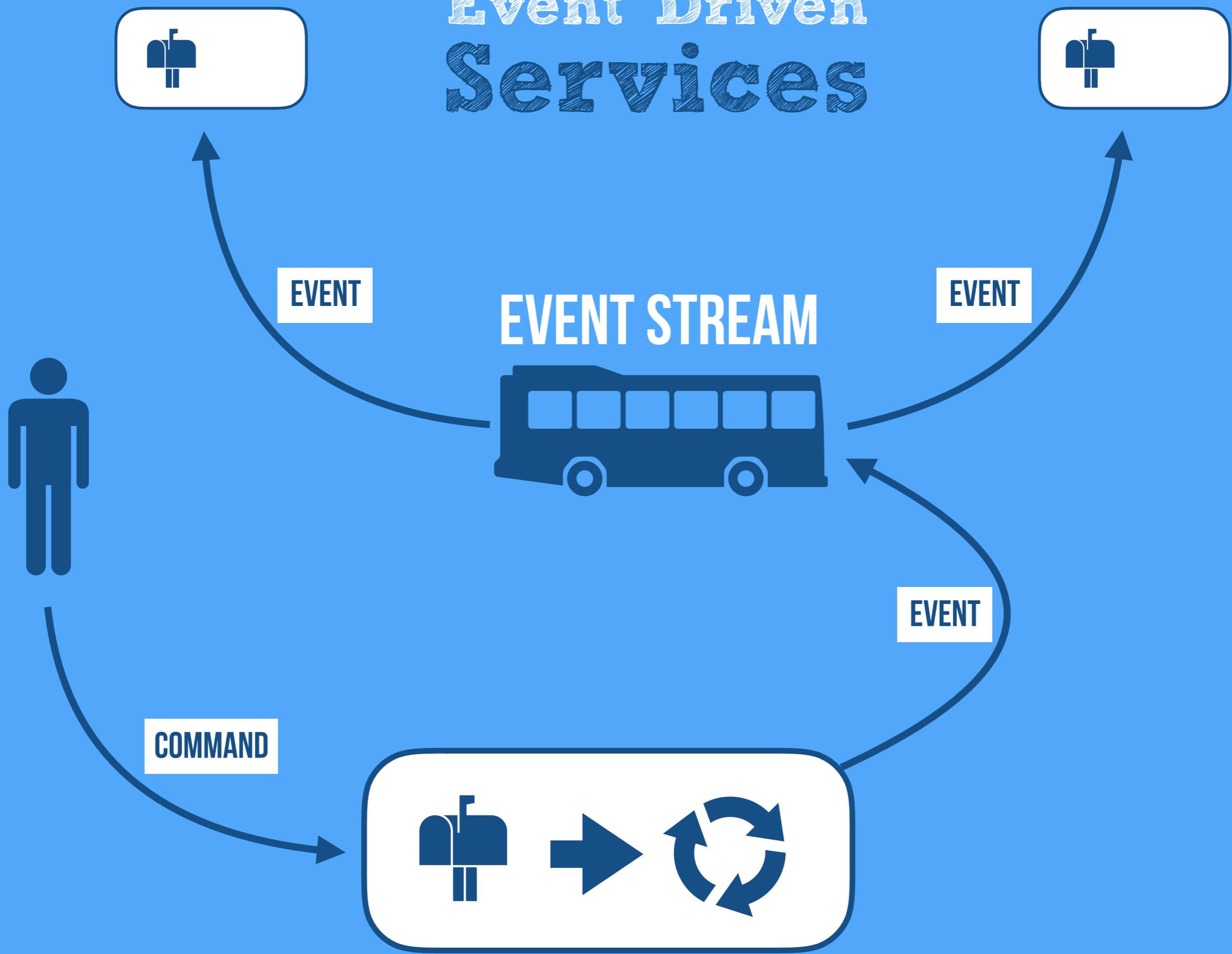
# Event Driven Services



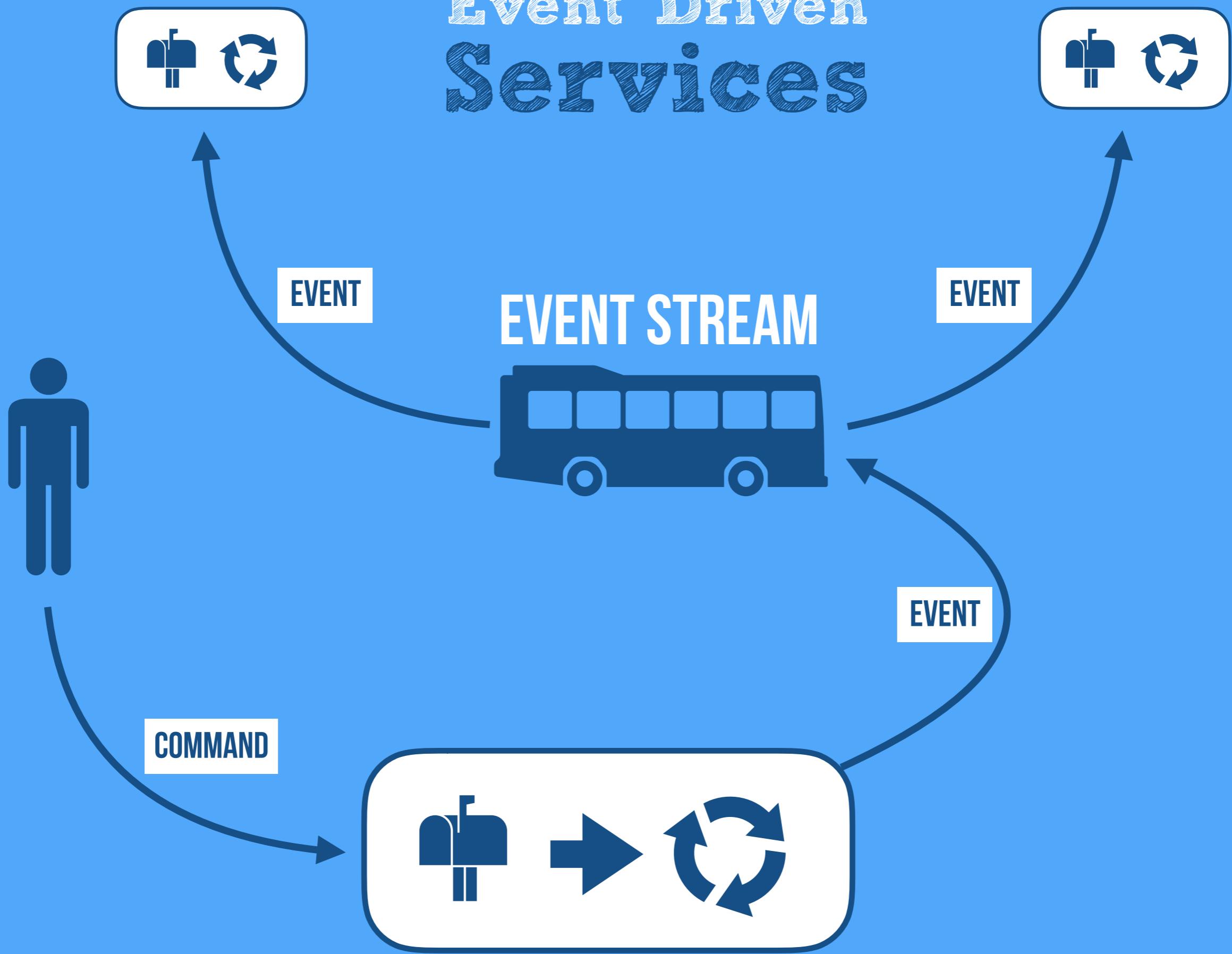
# Event Driven Services



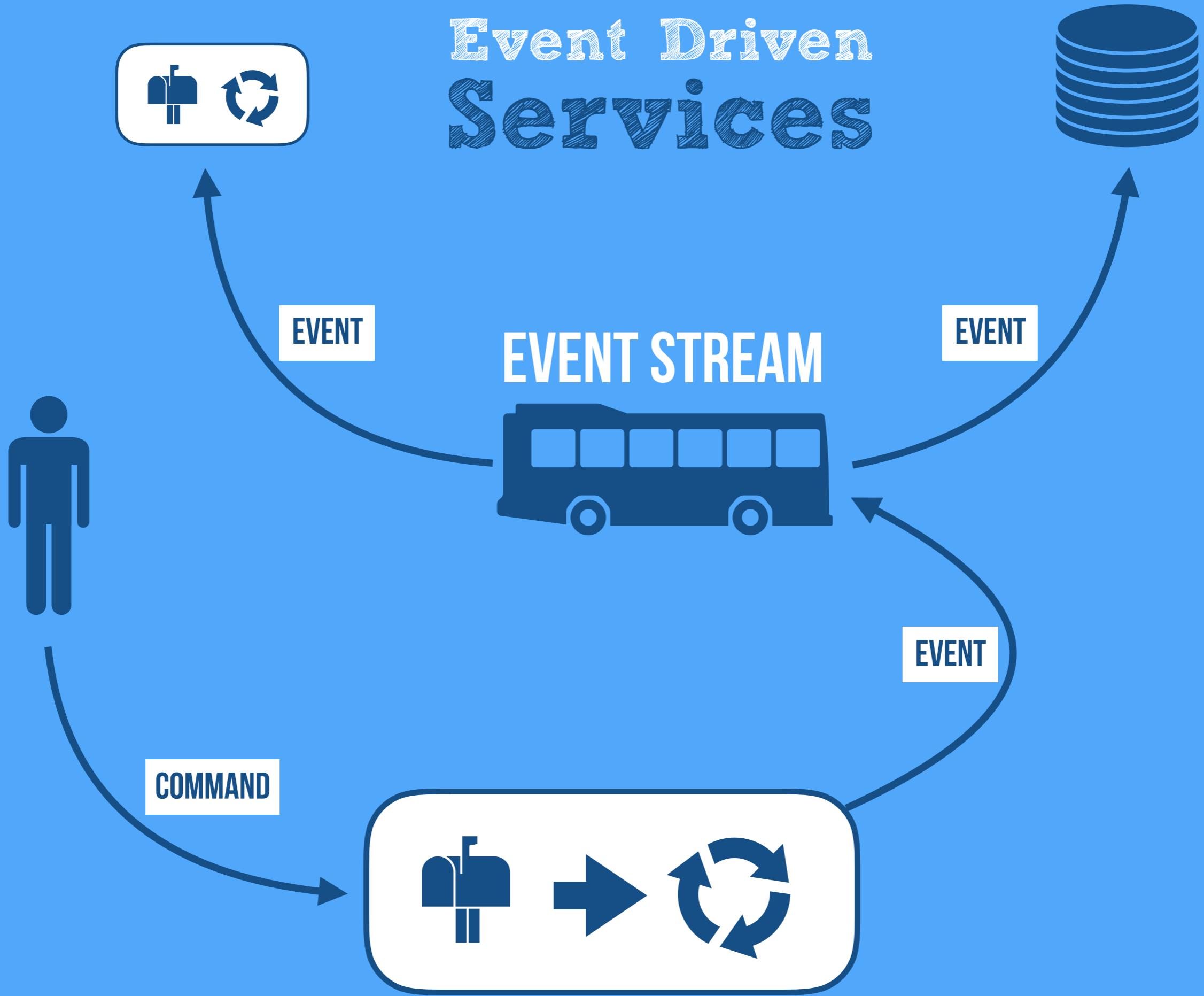
# Event Driven Services



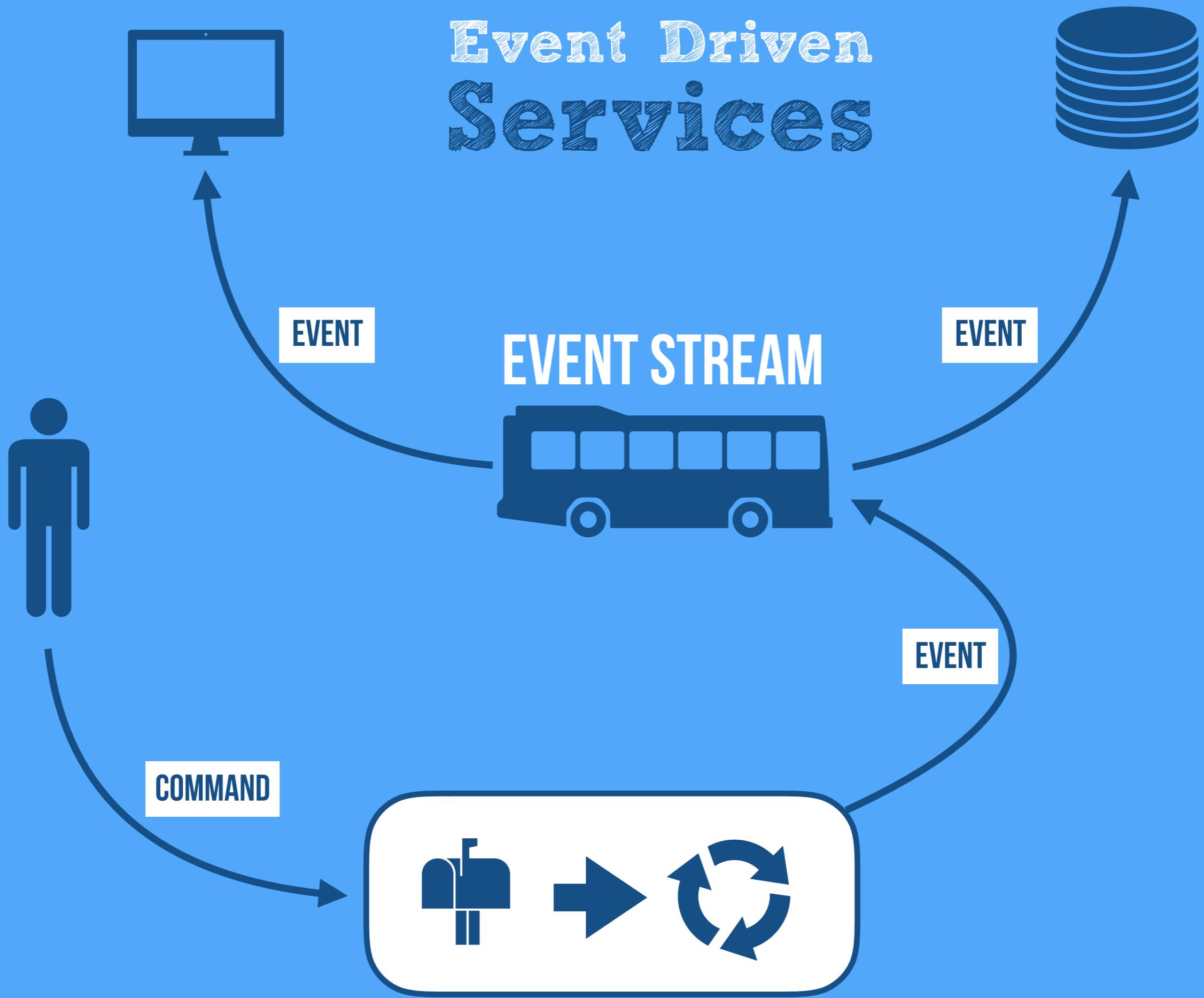
# Event Driven Services



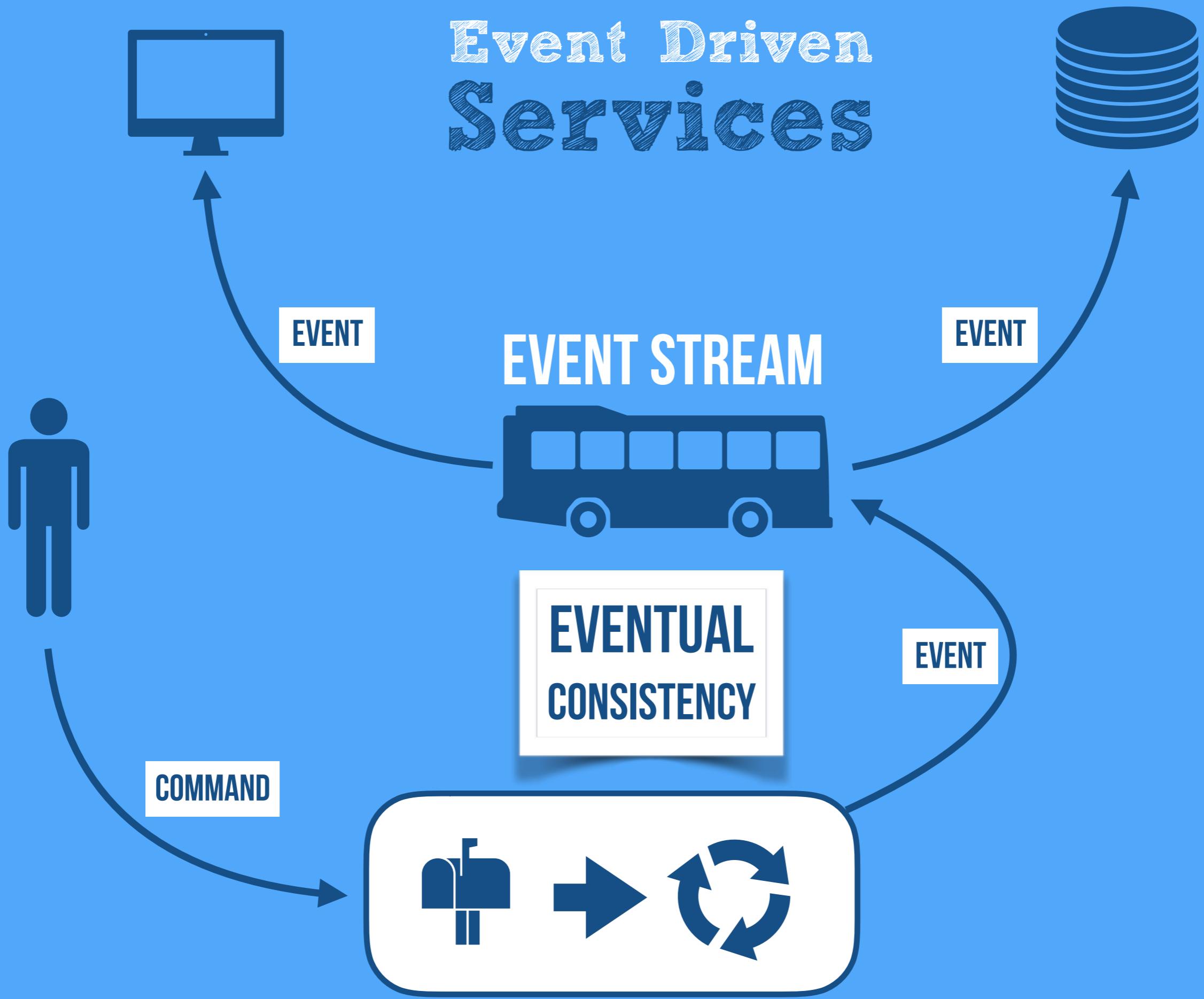
# Event Driven Services



# Event Driven Services



# Event Driven Services



USE THE  
**Event**  
**Stream**

USE THE  
**Event  
Stream**  
AS THE COMMUNICATION FABRIC

USE THE  
Event  
Stream

USE THE  
**Event**  
**Stream**  
AS THE INTEGRATION FABRIC

USE THE  
Event  
Stream

USE THE  
**Event**  
**Stream**  
AS THE REPLICATION FABRIC

USE THE  
**Event**  
**Stream**

USE THE  
**Event**  
**Stream**  
AS THE CONSENSUS FABRIC

USE THE  
Event  
Stream

USE THE  
**Event**  
**Stream**  
AS THE PERSISTENCE FABRIC

# The Problem With CRUD Services

# The Problem With CRUD Services

- \* **CRUD is FINE for totally ISOLATED data**

# The Problem With CRUD Services

- \* **CRUD is FINE for totally ISOLATED data**
- \* **But, CROSS CRUD services CONSISTENCY**

# The Problem With CRUD Services

- \* **CRUD is FINE for totally ISOLATED data**
- \* **But, CROSS CRUD services CONSISTENCY**  
→ Is HARD → No JOINS

# The Problem With CRUD Services

- \* **CRUD is FINE for totally ISOLATED data**
- \* **But, CROSS CRUD services CONSISTENCY**
  - Is HARD → No JOINS
  - Has AD-HOC & WEAK GUARANTEES

# The Problem With CRUD Services

- \* **CRUD is FINE for totally ISOLATED data**
- \* **But, CROSS CRUD services CONSISTENCY**
  - Is HARD → No JOINS
  - Has AD-HOC & WEAK GUARANTEES



**“In general, application developers simply  
do not implement large scalable applications  
assuming distributed transactions”**

**- PAT HELLAND**

**“Two-phase commit is the  
anti-availability protocol.”**

**- PAT HELLAND**



**STRONG**  
**Consistency**  
**IS THE WRONG DEFAULT**  
**IN DISTRIBUTED SYSTEMS**

**what can we do?**

**consistency**

**CONSISTENCY IS THE WRONG DEFAULT  
IN DISTRIBUTED SYSTEMS**

WE HAVE TO RELY ON  
Eventual  
Consistency

WE HAVE TO RELY ON  
Eventual  
Consistency  
BUT RELAX—IT'S HOW THE WORLD WORKS

WE NEED TO  
Embrace  
Reality

WE NEED TO  
Embrace  
Reality  
NOT FIGHT IT



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

Events Can Lead To Greater  
Certainty

**“An autonomous component can only promise its own behavior.”**

**“Autonomy makes information local, leading to greater certainty and stability.”**

**- MARK BURGESS**

# Events Can Help Us Craft Autonomous Islands Of Determinism





# Inside Data

OUR CURRENT PRESENT → STATE

# **Inside Data**

**OUR CURRENT PRESENT ➔ STATE**

# **Outside Data**

**BLAST FROM THE PAST ➔ EVENTS/FACTS**

# Inside Data

OUR CURRENT PRESENT → STATE

# Outside Data

BLAST FROM THE PAST → EVENTS/FACTS

# Between Services

HOPE FOR THE FUTURE → COMMANDS

# A system of microservices is a never ending stream towards convergence



A system of microservices is a  
never ending stream towards convergence

**There Is No Now**



Resilience  
is by  
Design



Photo courtesy of FEMA/Joselyne Augustino

EVENTS CAN HELP US  
Manage  
Failure  
INSTEAD OF TRYING TO AVOID IT

# REQUIREMENTS FOR A Sane Failure Model

**FAILURES NEED TO BE**

- 1. CONTAINED—AVOID CASCADING FAILURES**
- 2. REIFIED—AS EVENTS**
- 3. SIGNALLED—ASYNCHRONOUSLY**
- 4. OBSERVED—BY 1-N**
- 5. MANAGED—OUTSIDE FAILED CONTEXT**

# Event Based Persistence

You can use **CRUD**  
Together with **EVENT STREAMS**  
To get an internally consistent **MATERIALIZED VIEW**

You can use **CRUD**

Together with **EVENT STREAMS**

To get an internally consistent **MATERIALIZED VIEW**

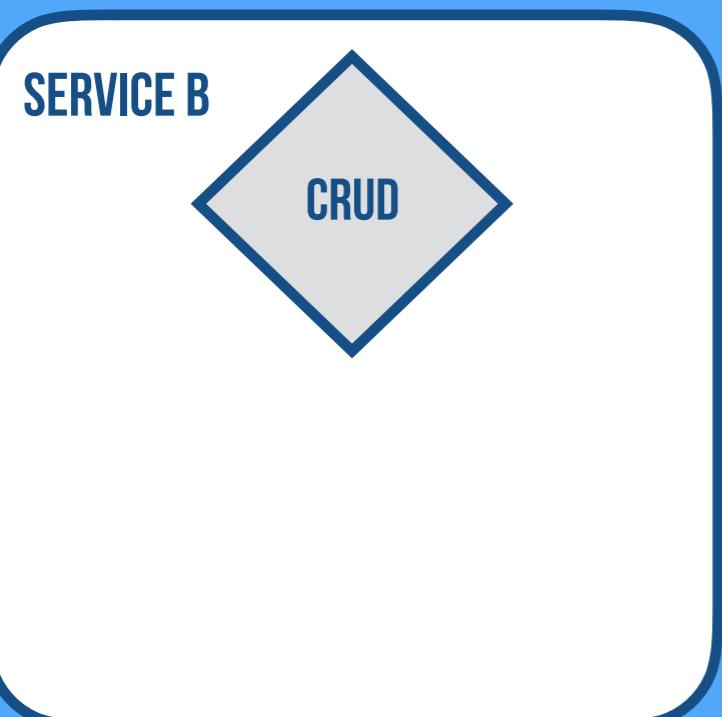
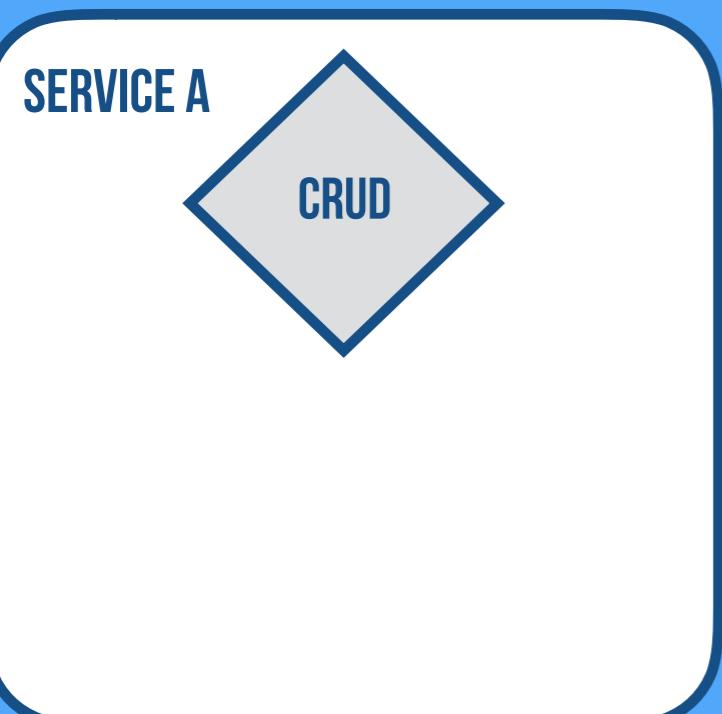
**SERVICE A**

**SERVICE B**

You can use **CRUD**

Together with **EVENT STREAMS**

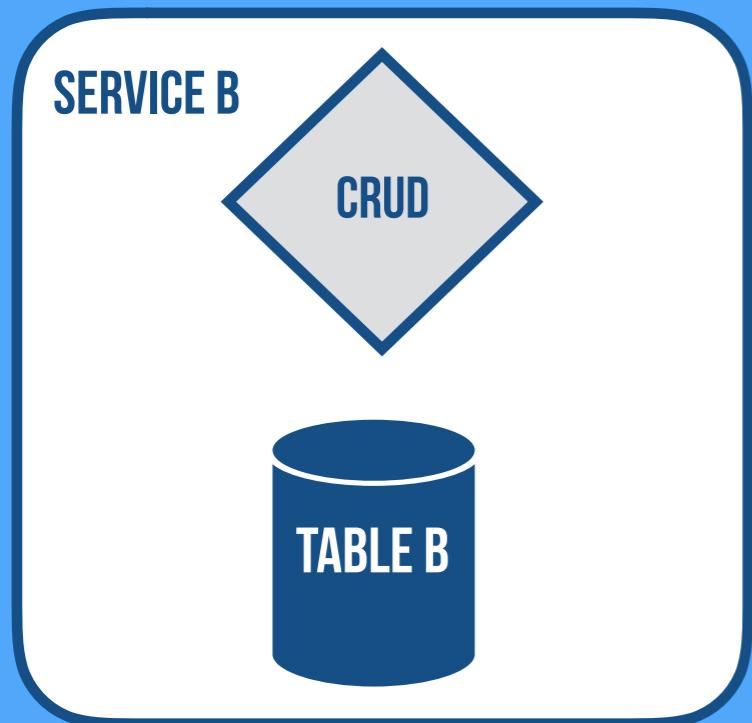
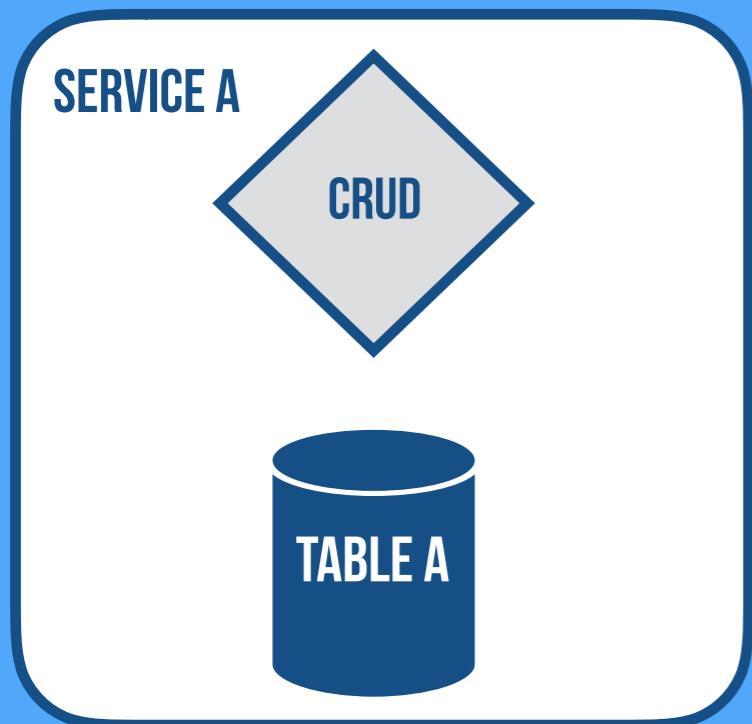
To get an internally consistent **MATERIALIZED VIEW**



You can use **CRUD**

Together with **EVENT STREAMS**

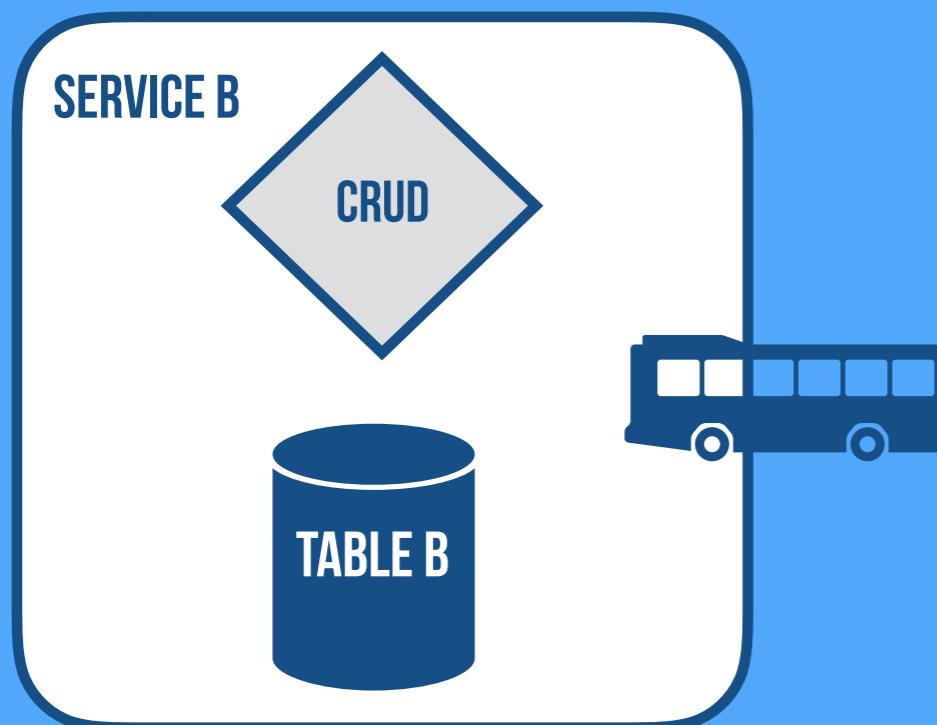
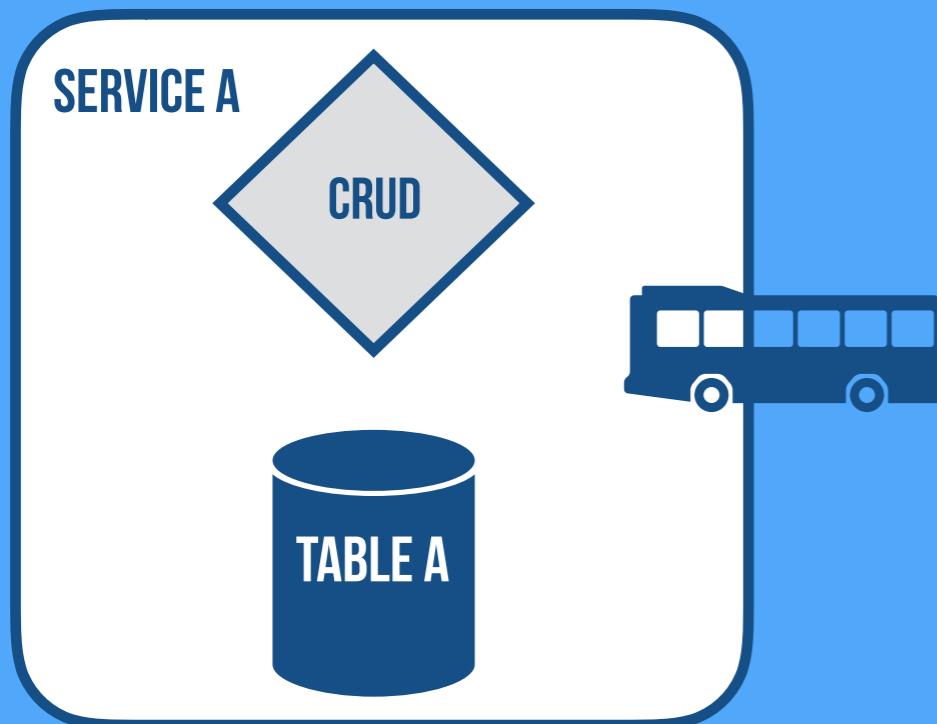
To get an internally consistent **MATERIALIZED VIEW**



You can use **CRUD**

Together with **EVENT STREAMS**

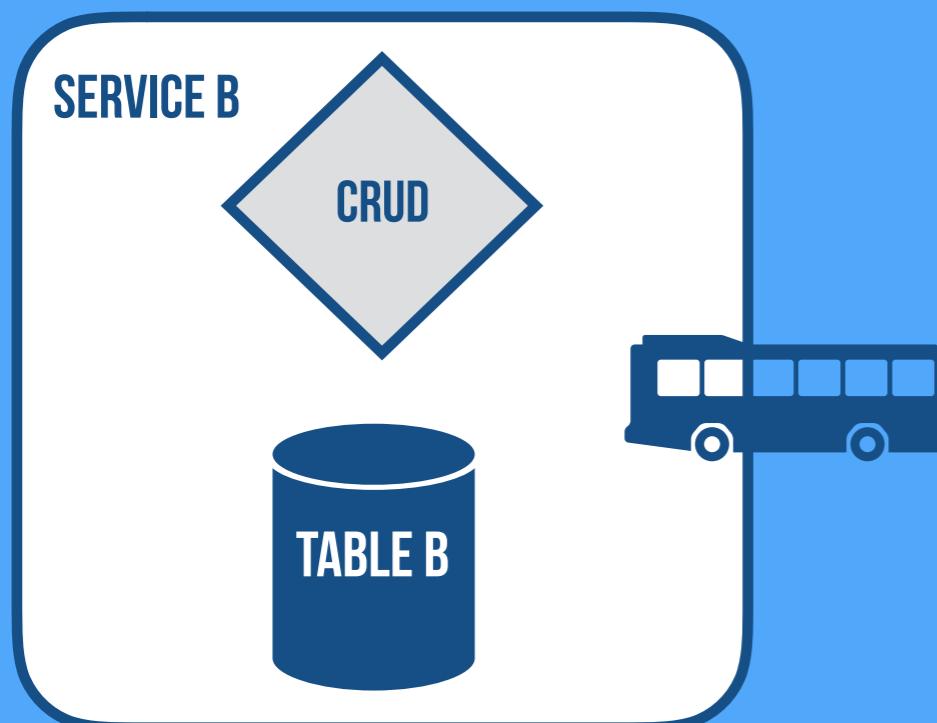
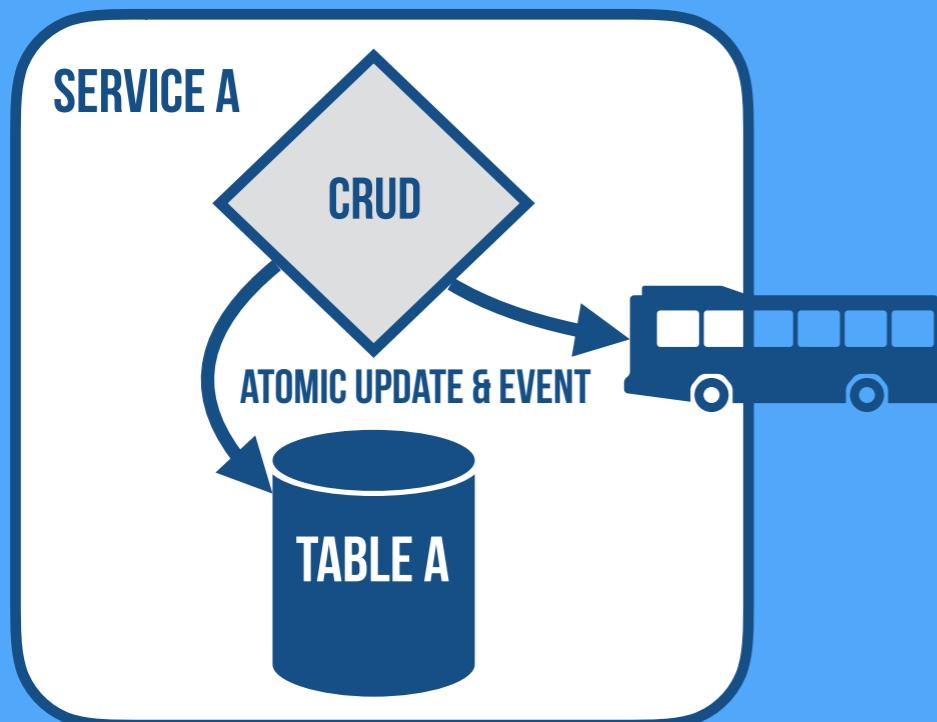
To get an internally consistent **MATERIALIZED VIEW**



You can use **CRUD**

Together with **EVENT STREAMS**

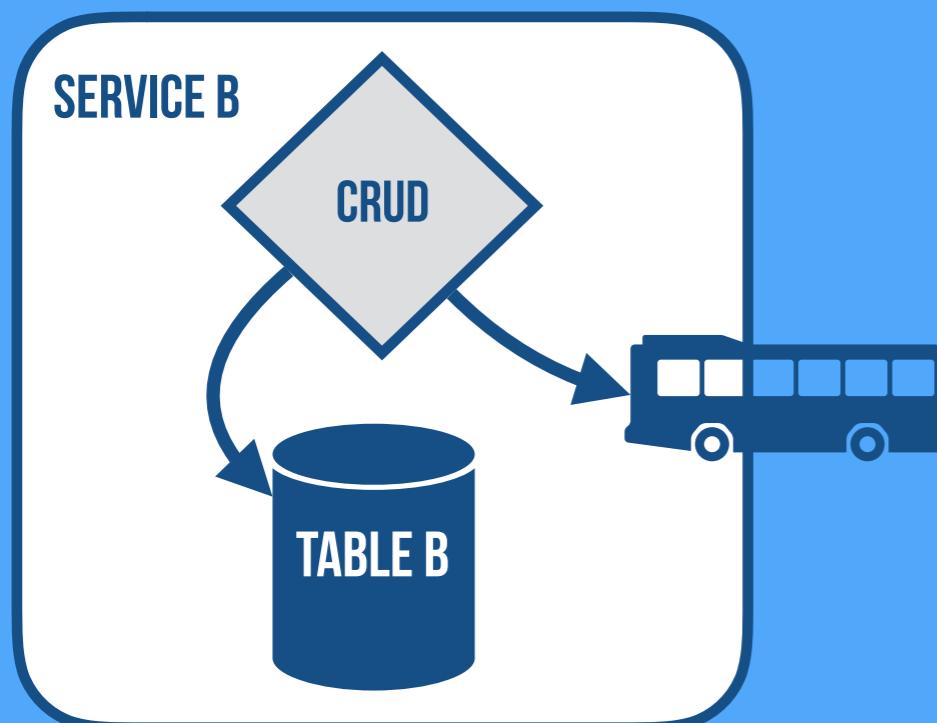
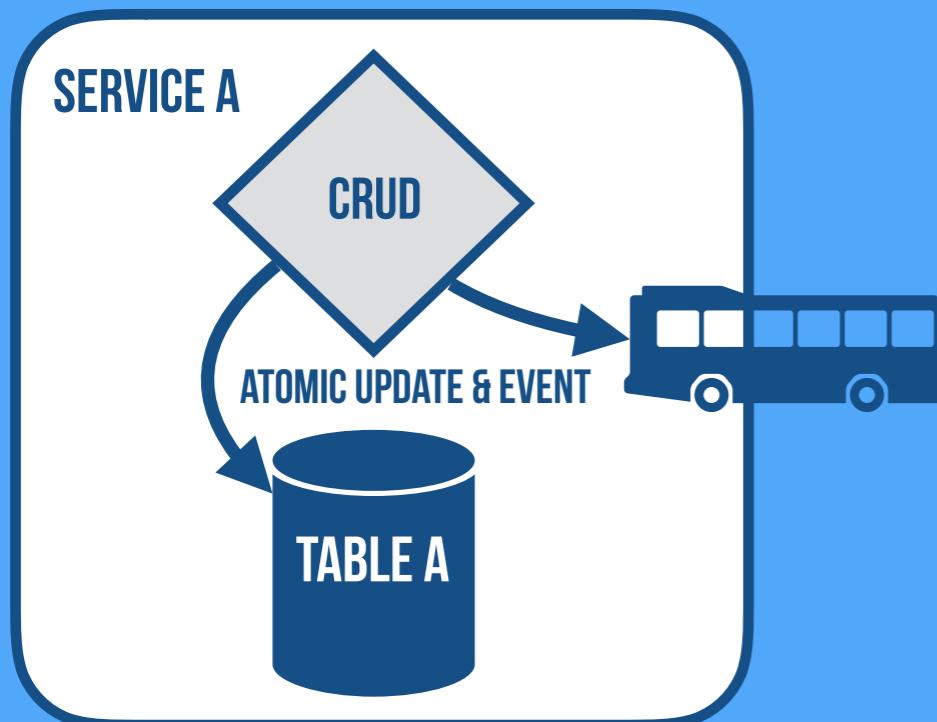
To get an internally consistent **MATERIALIZED VIEW**



You can use **CRUD**

Together with **EVENT STREAMS**

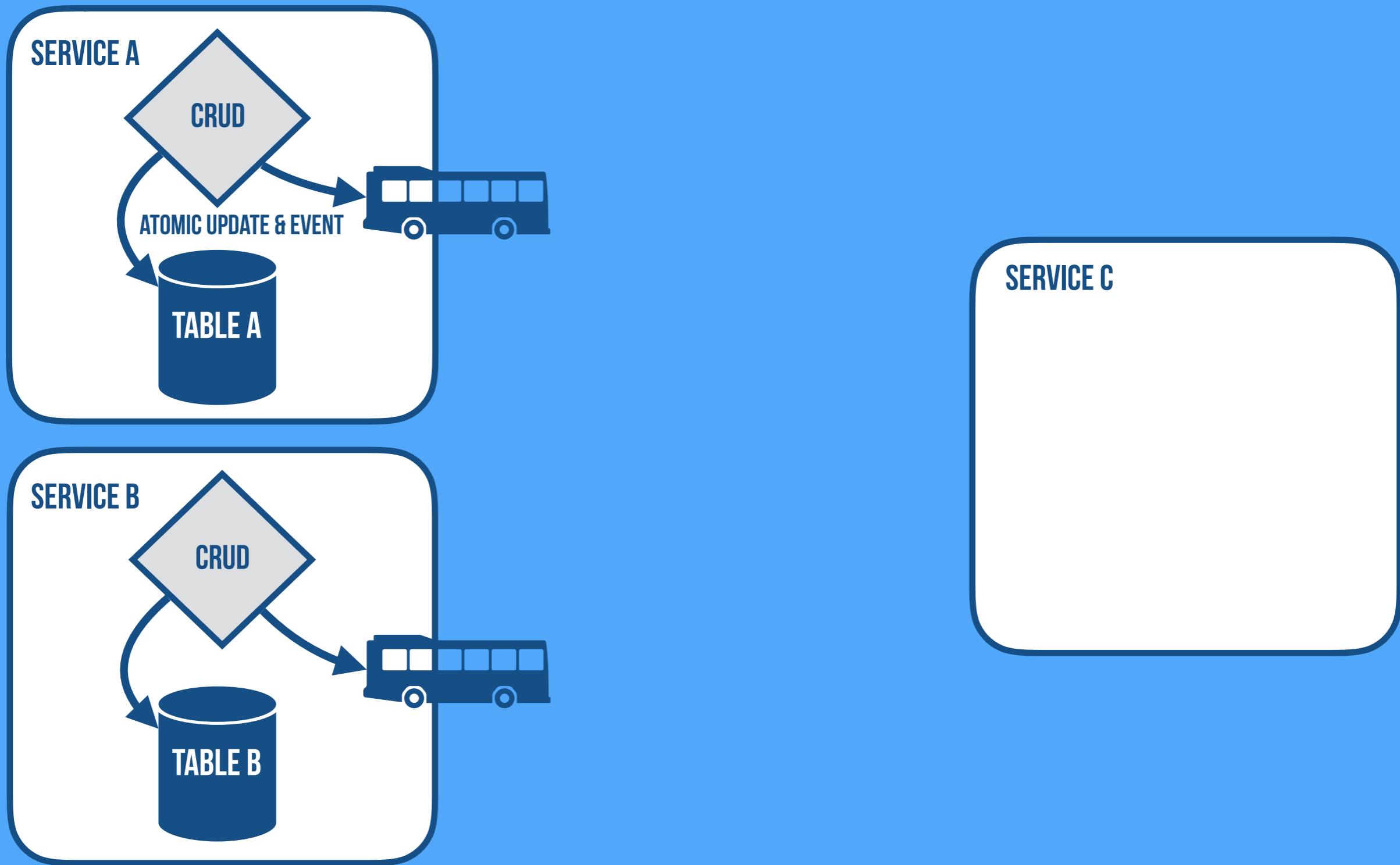
To get an internally consistent **MATERIALIZED VIEW**



You can use **CRUD**

Together with **EVENT STREAMS**

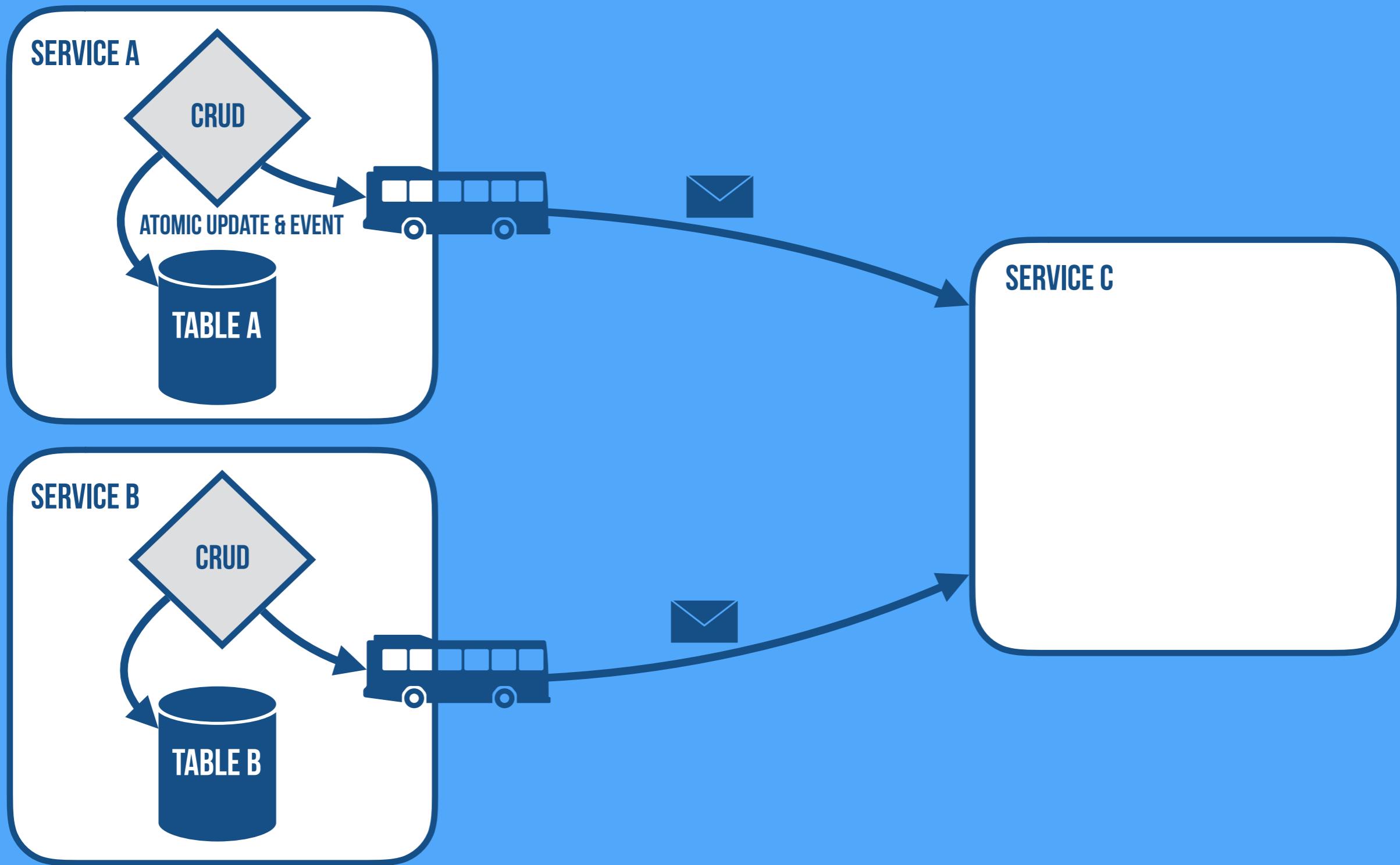
To get an internally consistent **MATERIALIZED VIEW**



You can use **CRUD**

Together with **EVENT STREAMS**

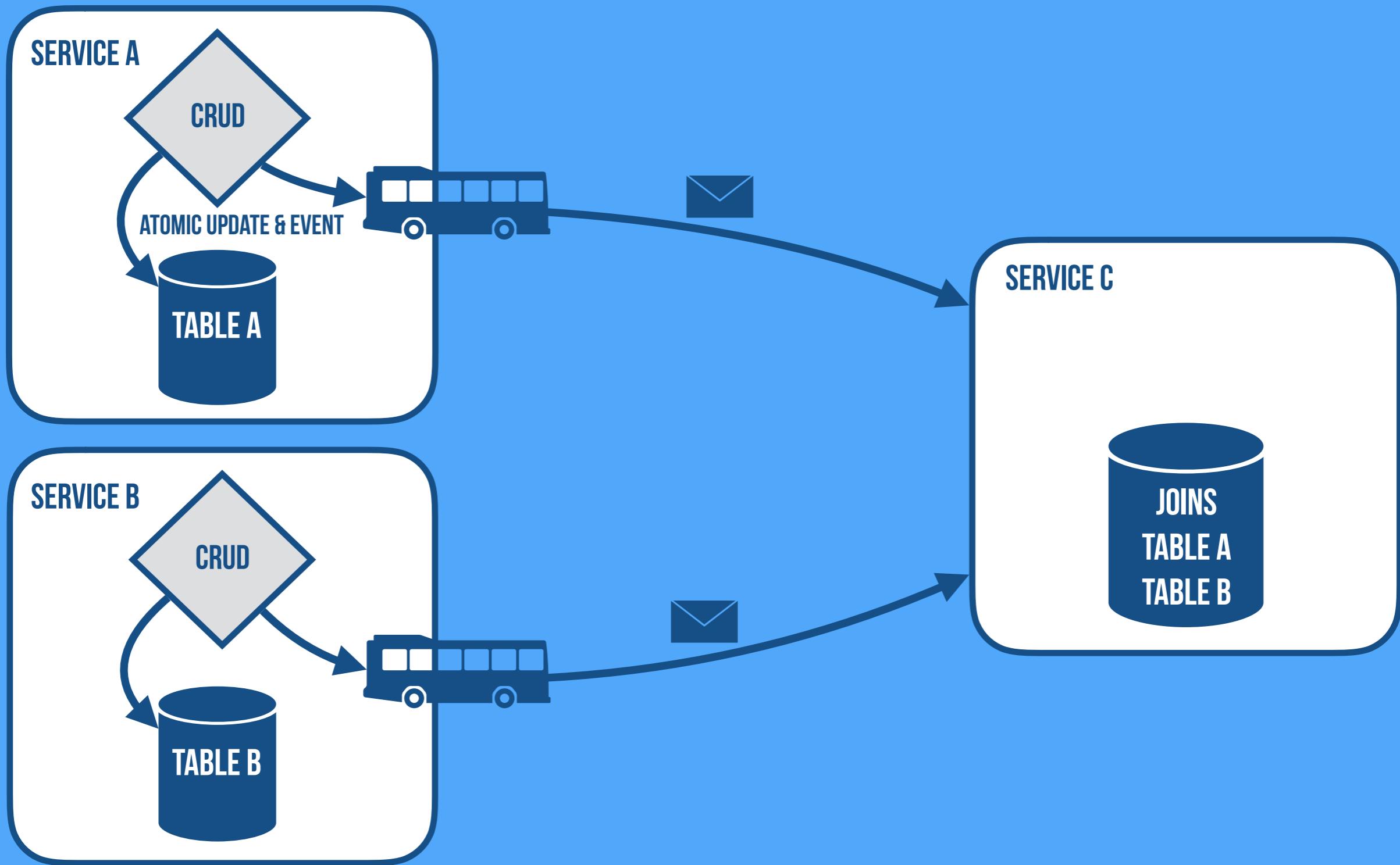
To get an internally consistent **MATERIALIZED VIEW**



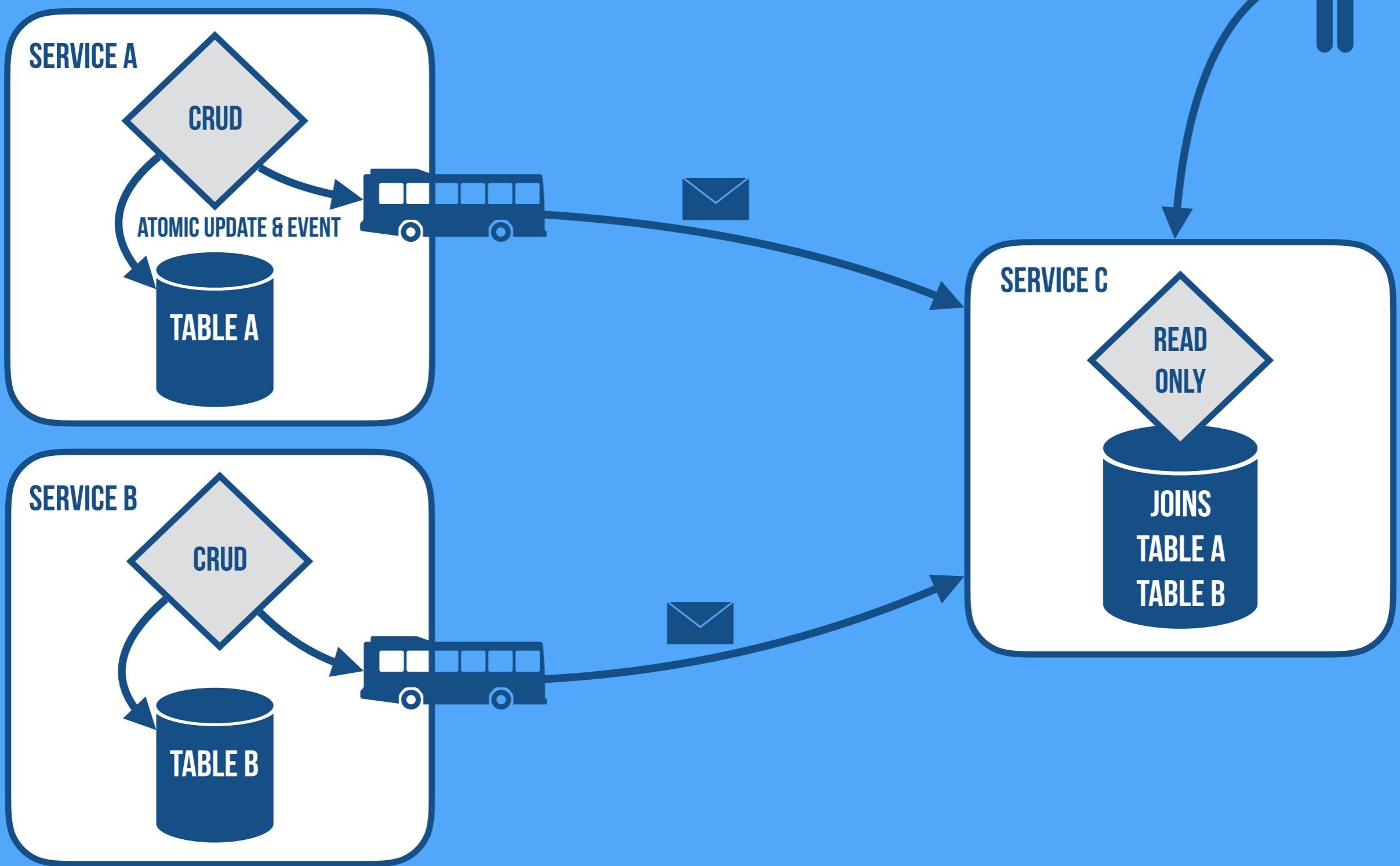
You can use **CRUD**

Together with **EVENT STREAMS**

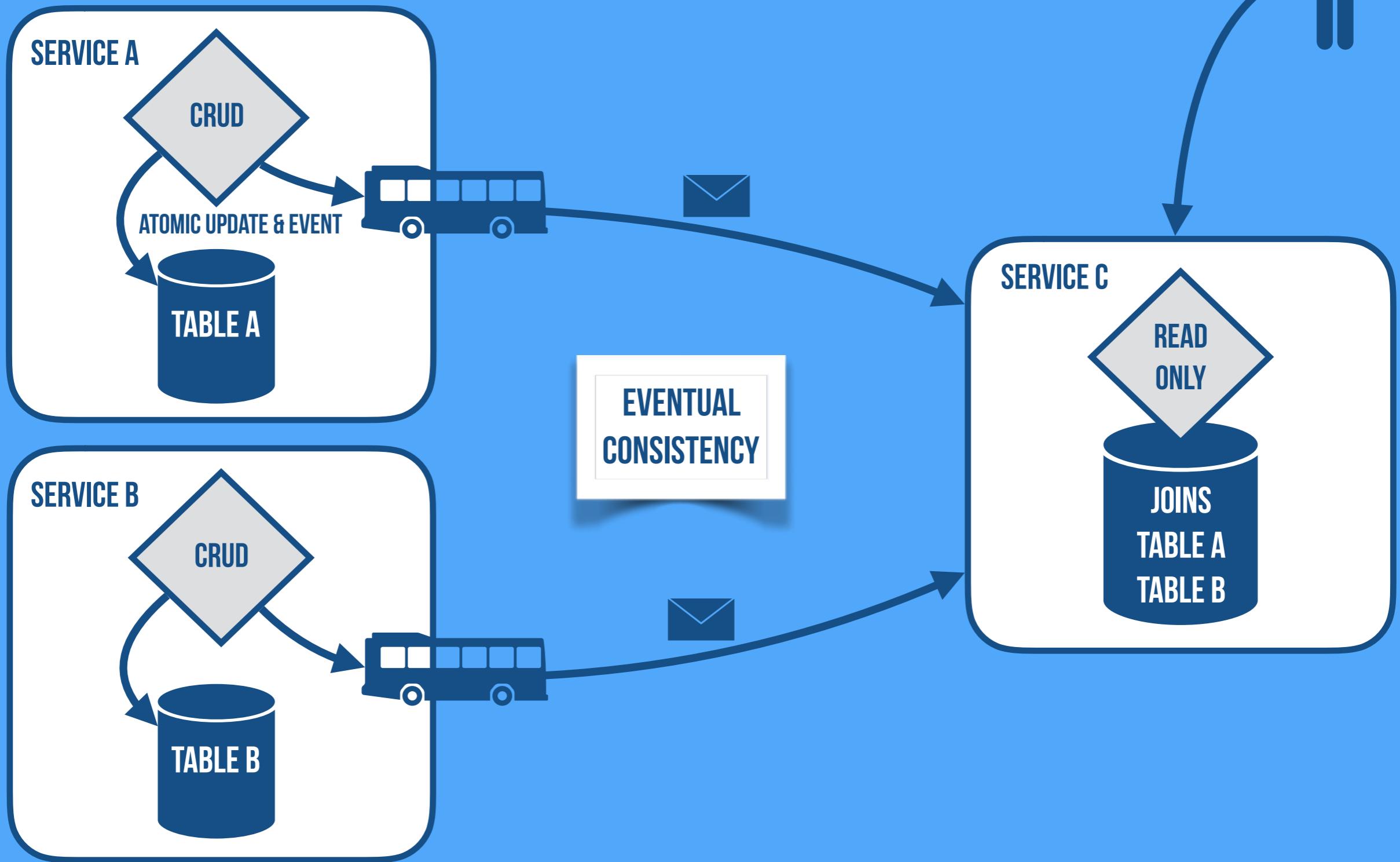
To get an internally consistent **MATERIALIZED VIEW**



# You can use CRUD Together with EVENT STREAMS To get an internally consistent MATERIALIZED VIEW



# You can use CRUD Together with EVENT STREAMS To get an internally consistent MATERIALIZED VIEW



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

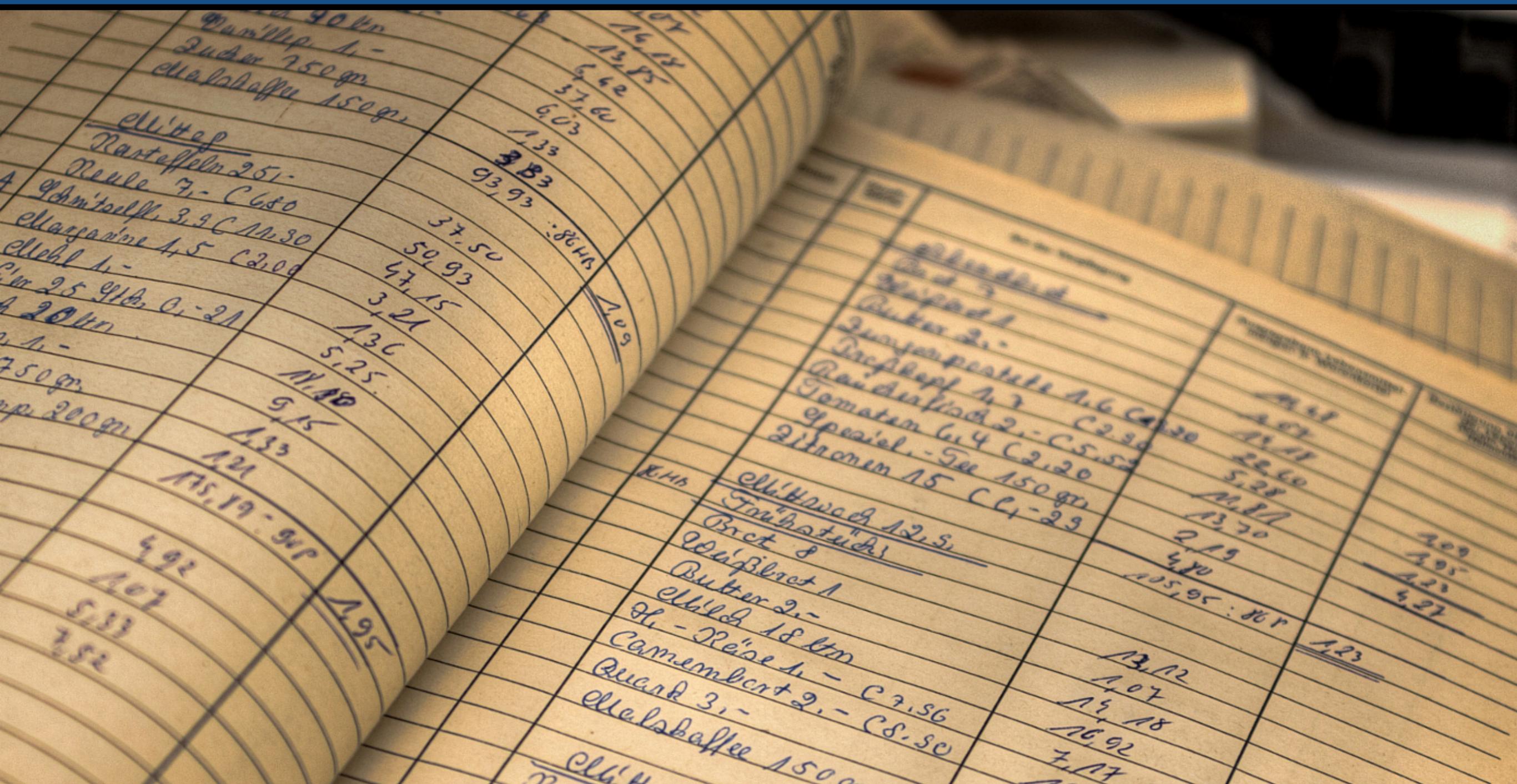
**- JIM GRAY**

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

**- PAT HELLAND**

# Event Logging

## The Bedrock



# Event Sourcing

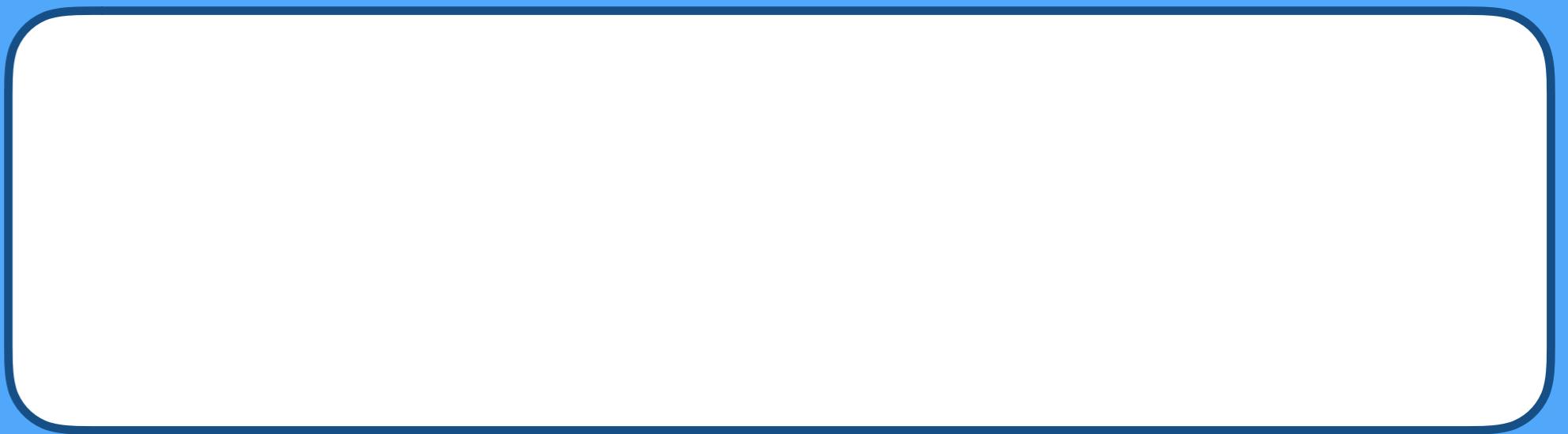
## A Cure For the Cardinal Sin

# 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 - RECOVER 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 - RECOVER 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 - RECOVER FROM FAILURE



1) Rehydrate Events  
from Event Log



2) Update internal  
component state

# Event Sourced Services



HAPPY PATH



SAD PATH - RECOVER FROM FAILURE



# Event Sourcing

# Event Sourcing

- \* One single **SOURCE OF TRUTH** with **ALL HISTORY**

# Event Sourcing

- \* One single **SOURCE OF TRUTH** with **ALL HISTORY**
- \* Allows for **MEMORY IMAGE** (**Durable In-Memory State**)

# Event Sourcing

- \* One single **SOURCE OF TRUTH** with **ALL HISTORY**
- \* Allows for **MEMORY IMAGE** (**Durable In-Memory State**)
- \* Avoids the **OBJECT-RELATIONAL MISMATCH**

# Event Sourcing

- \* One single **SOURCE OF TRUTH** with **ALL HISTORY**
- \* Allows for **MEMORY IMAGE** (**Durable In-Memory State**)
- \* Avoids the **OBJECT-RELATIONAL MISMATCH**
- \* Allows others to **SUBSCRIBE TO STATE CHANGES**

# Event Sourcing

- \* One single **SOURCE OF TRUTH** with **ALL HISTORY**
- \* Allows for **MEMORY IMAGE** (**Durable In-Memory State**)
- \* Avoids the **OBJECT-RELATIONAL MISMATCH**
- \* Allows others to **SUBSCRIBE TO STATE CHANGES**
- \* Has good **MECHANICAL SYMPATHY**  
**(Single Writer Principle etc.)**

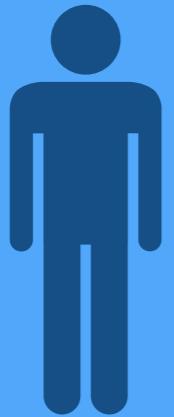
# **Disadvantages Of Using Event Sourcing**

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

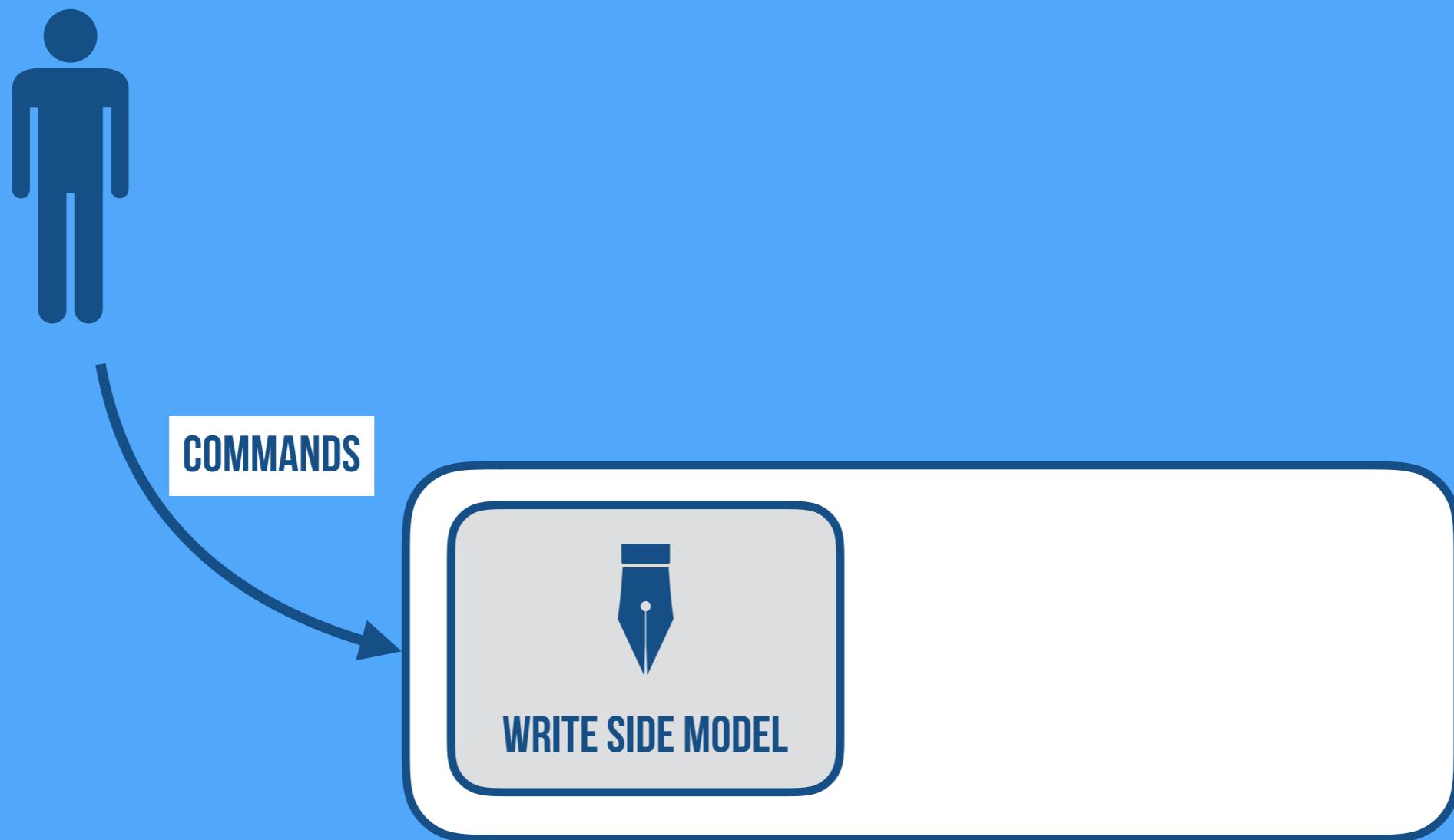
Untangle Your  
Read and Write Models  
**With cQRS**

# CQRS

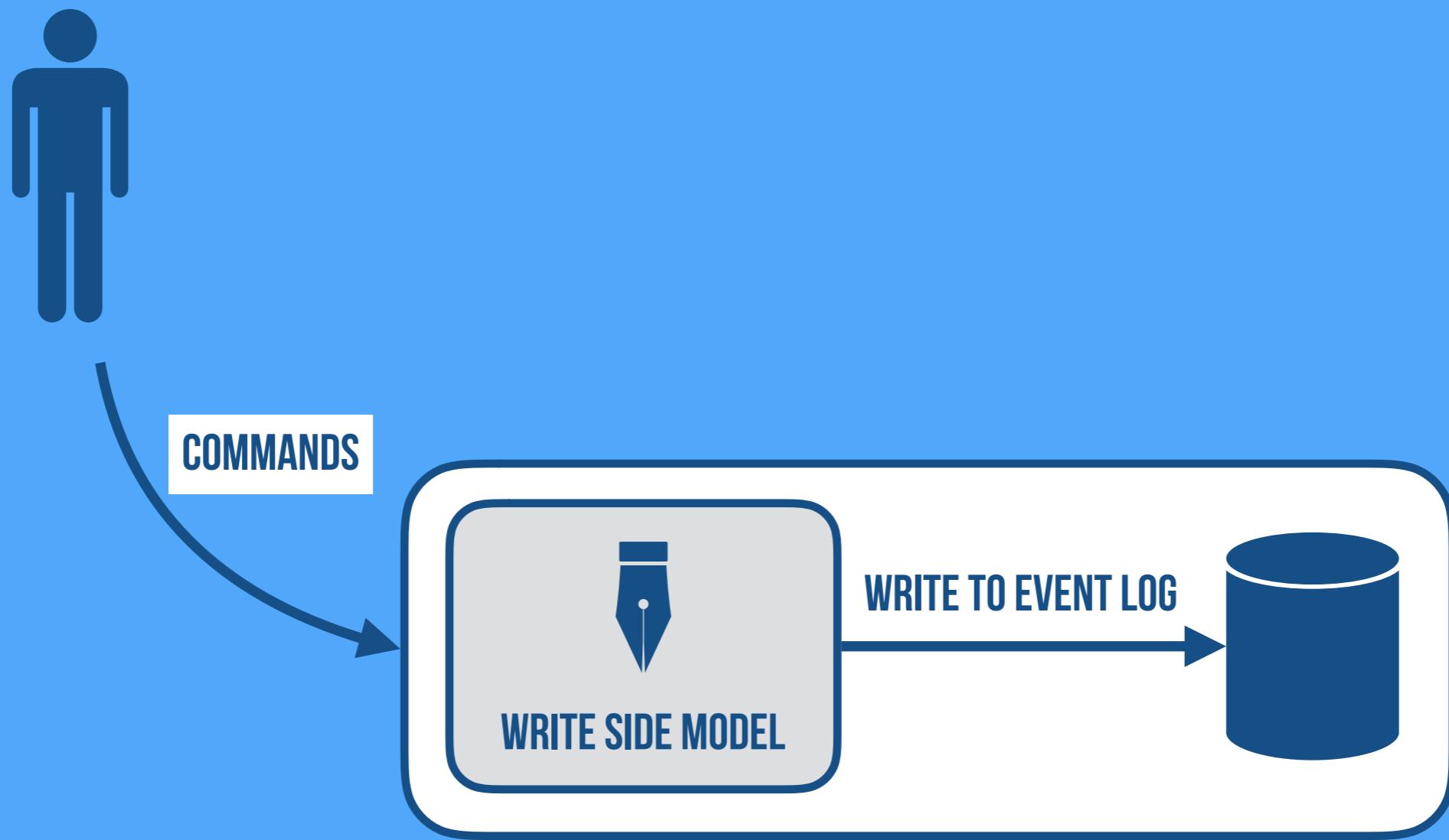
# CQRS



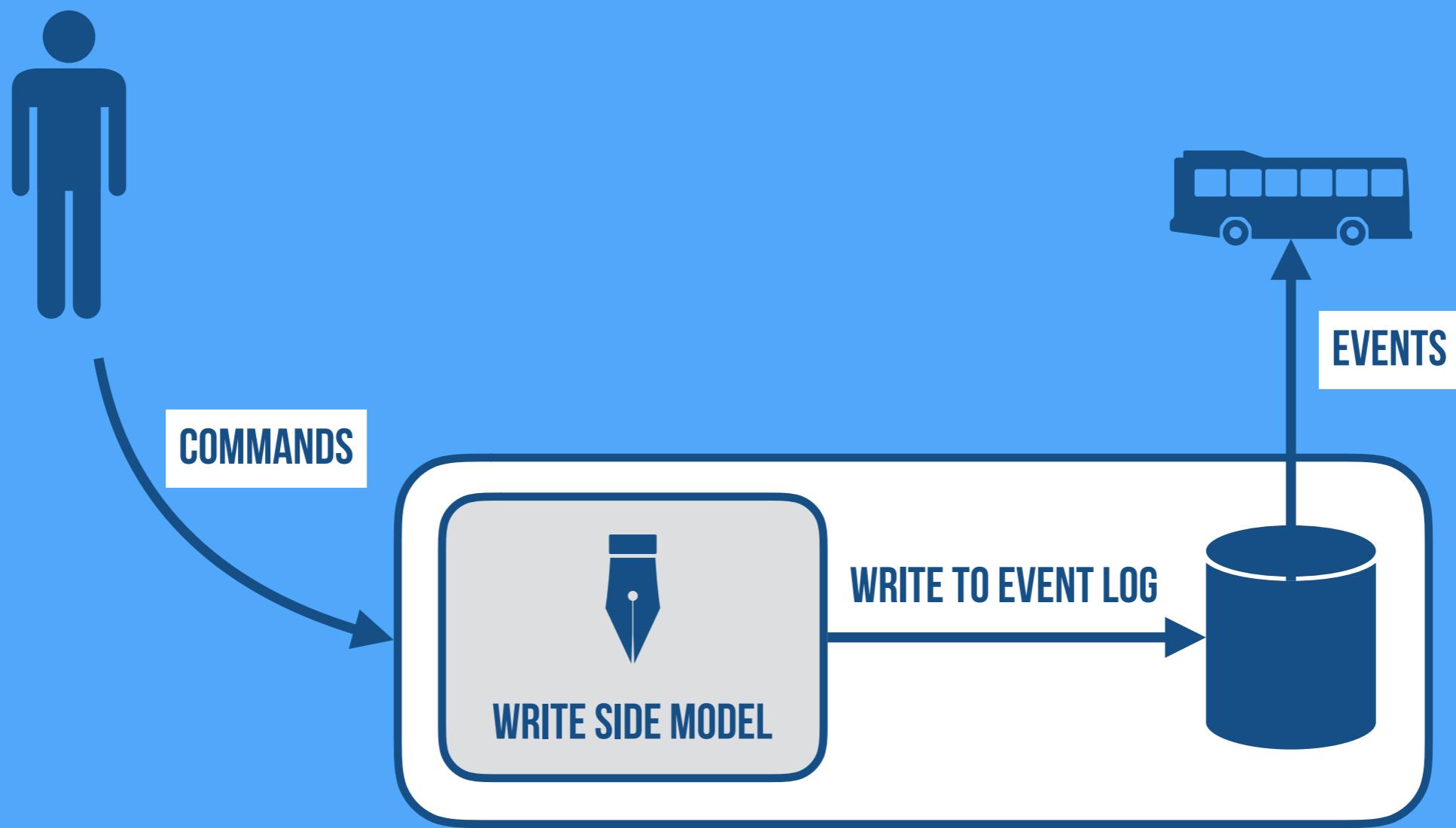
# CQRS



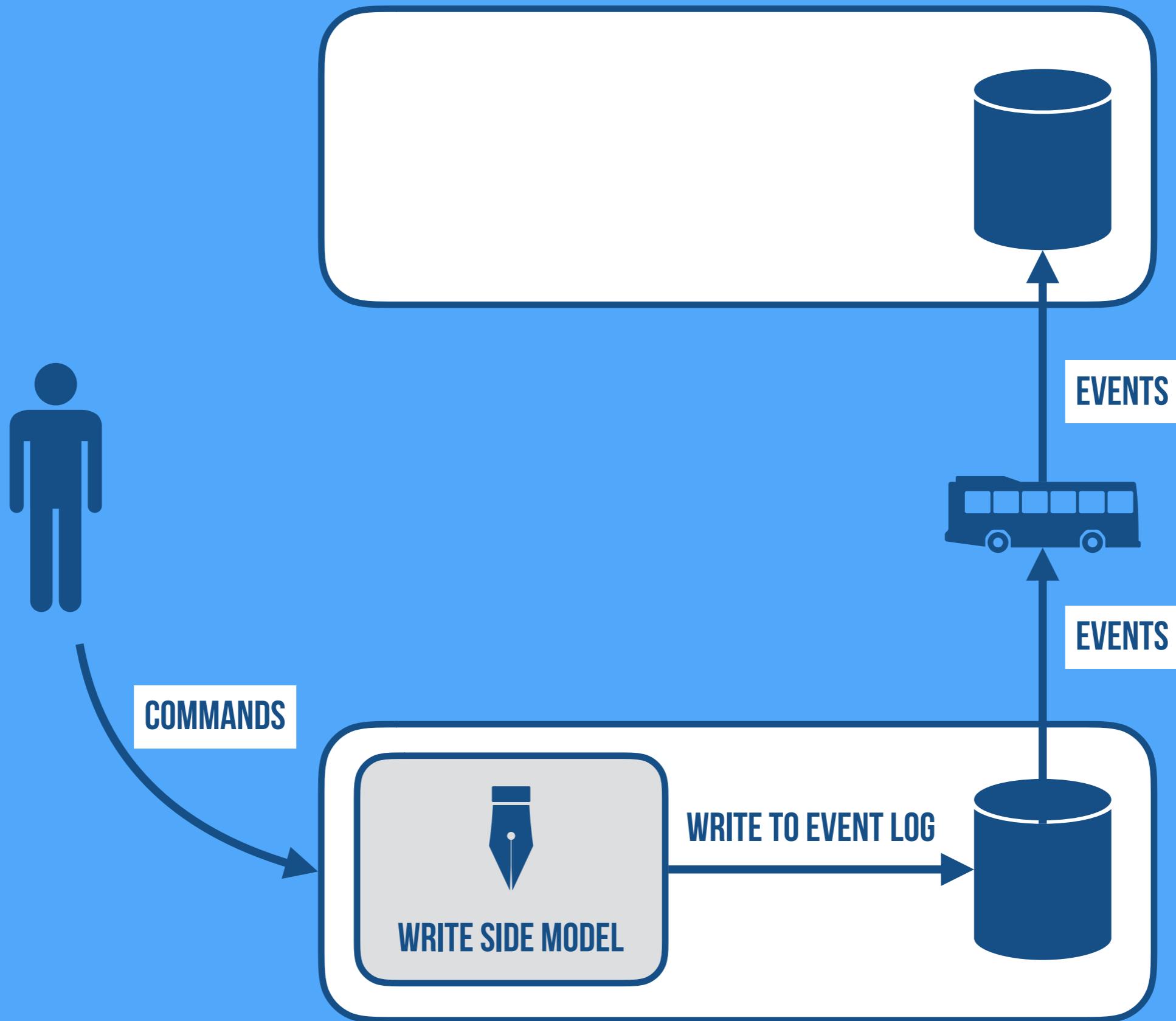
# CQRS



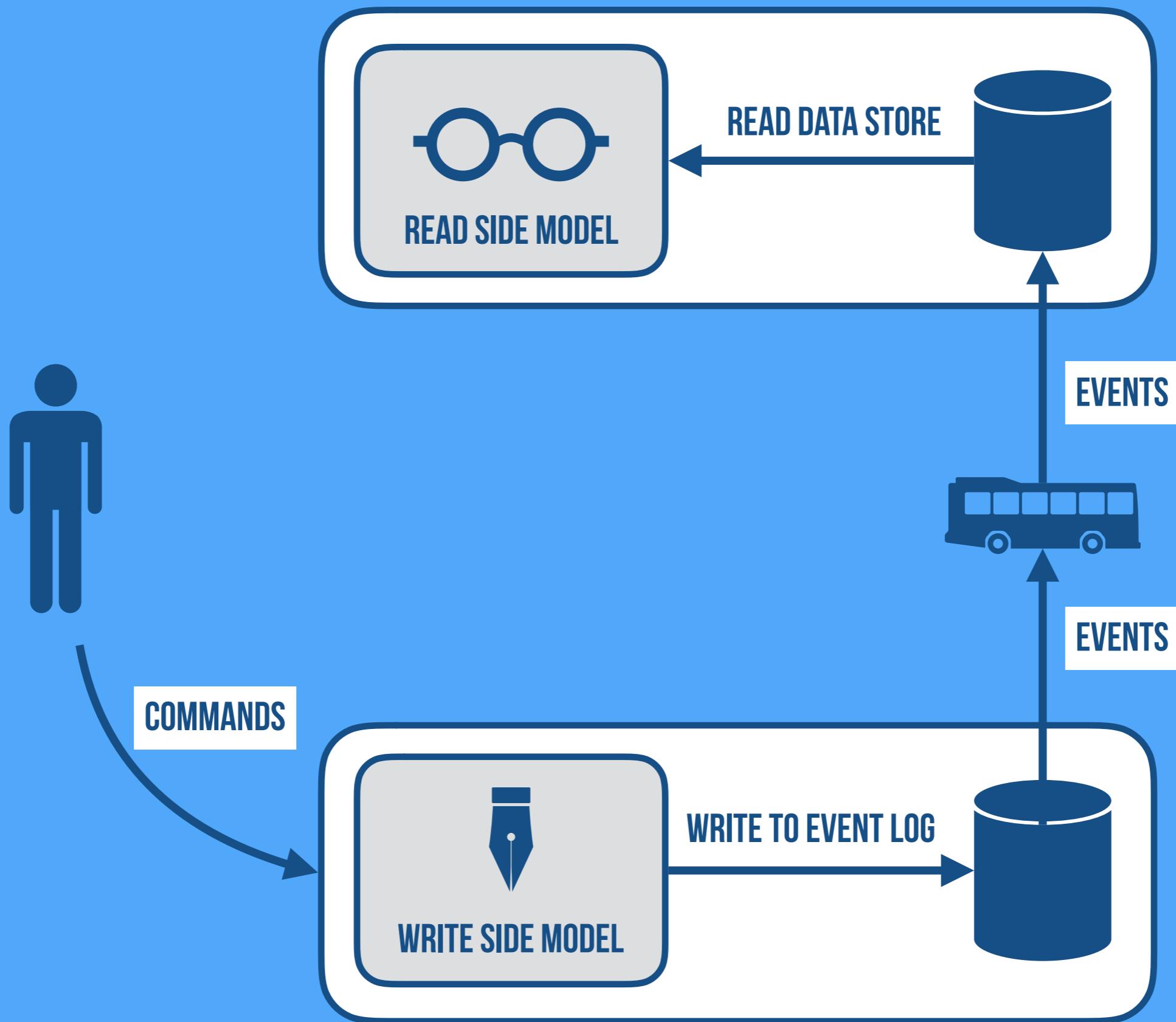
# CQRS



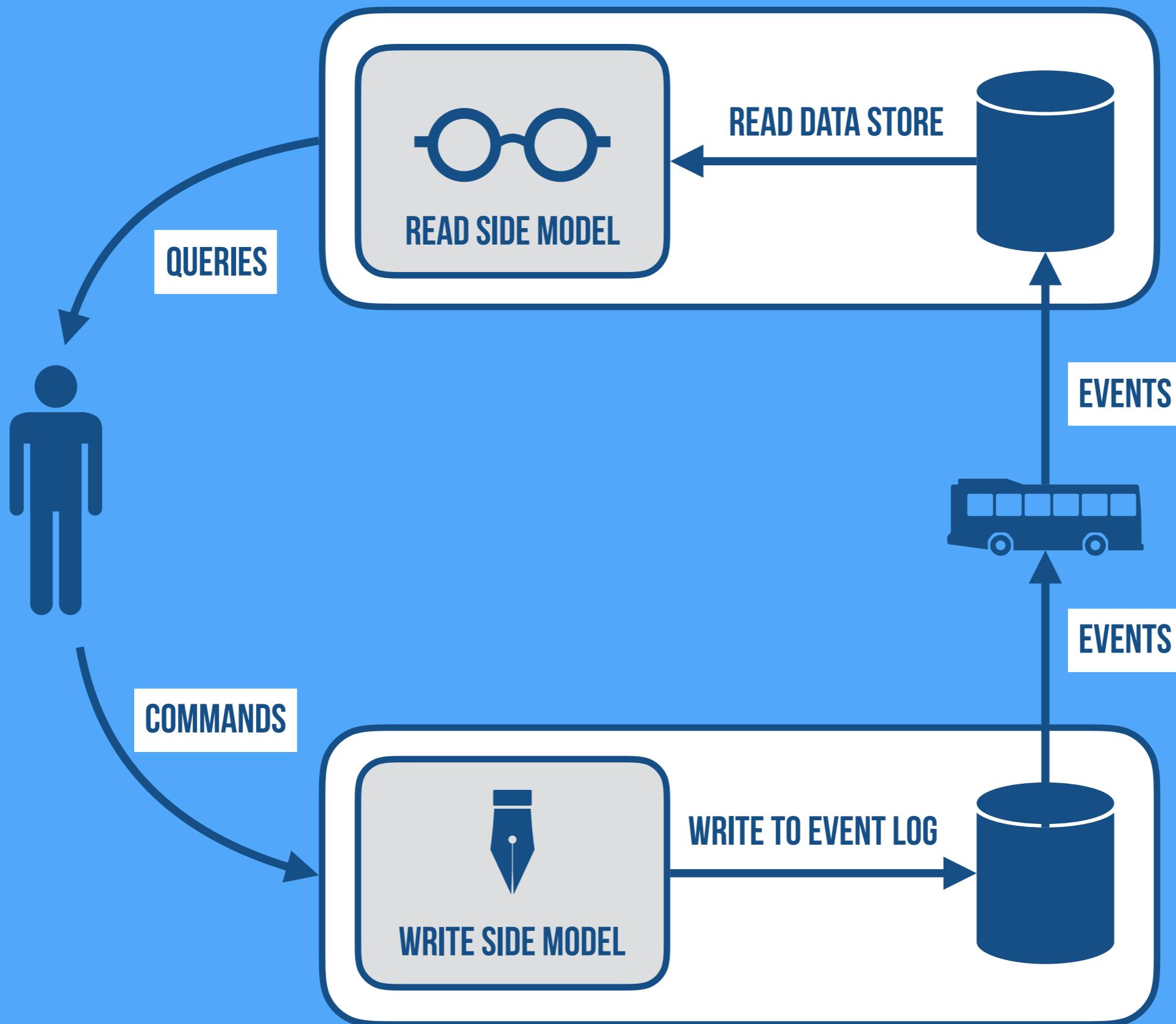
# CQRS



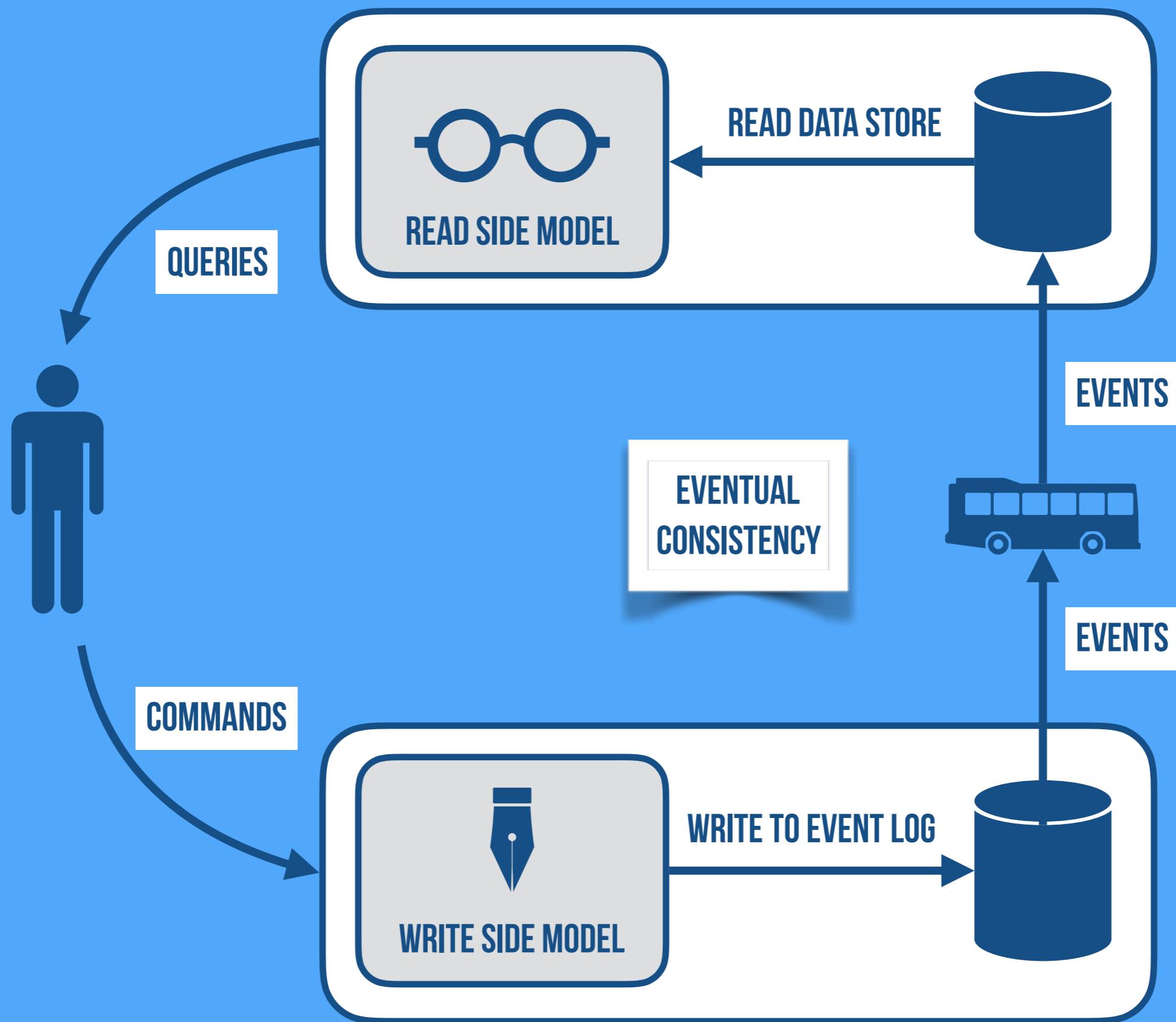
# CQRS



# CQRS



# CQRS



# **Advantages Of Using CQRS**

- \* Temporal DECOUPLING**
- \* Increased RESILIENCE**
- \* Increased SCALABILITY**
- \* Allows for MULTIPLE EVENT SUBSCRIBERS**
- \* Allows for POLYGLOT PERSISTENCE**

# **Disadvantages Of Using CQRS**

- \* **MORE INFRASTRUCTURE** to manage
- \* **2 DATA MODELS** (or more) to maintain
- \* **EVENTUAL Consistency**

Events  
Allow Us To Manage  
Time

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

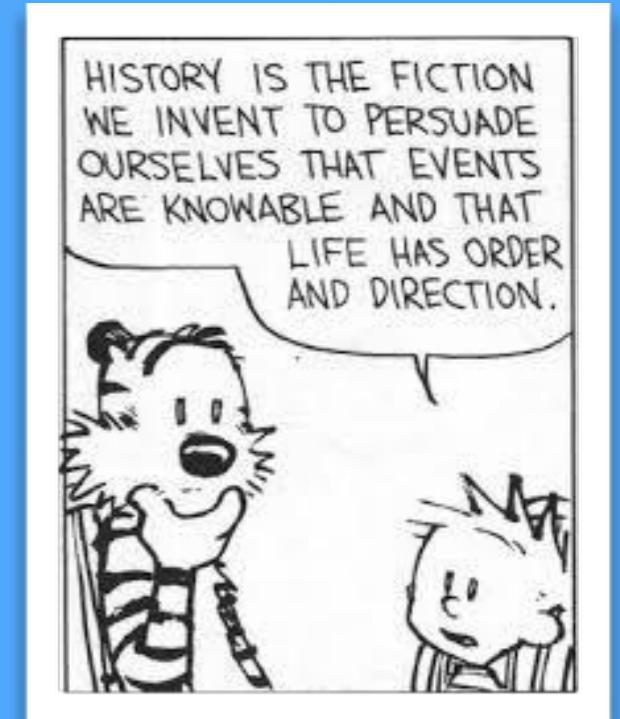
**- GREG YOUNG**

# Event Sourcing Allows Us To Model Time

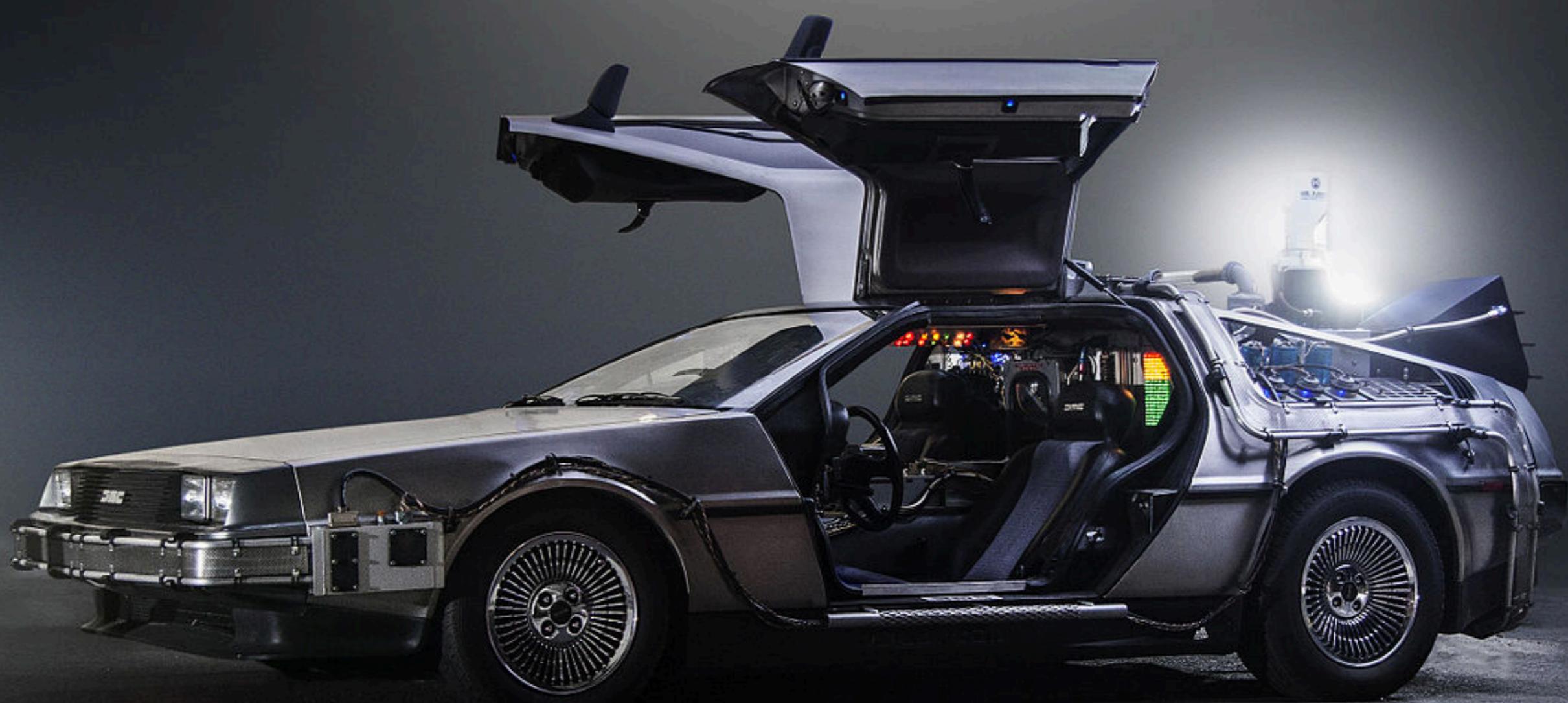
- \* Event is a SNAPSHOT IN TIME
- \* Event ID is an INDEX FOR TIME
- \* Event Log is our FULL HISTORY

The DATABASE OF OUR PAST

The PATH TO OUR PRESENT



Event Sourcing  
Allows For  
Time Travel



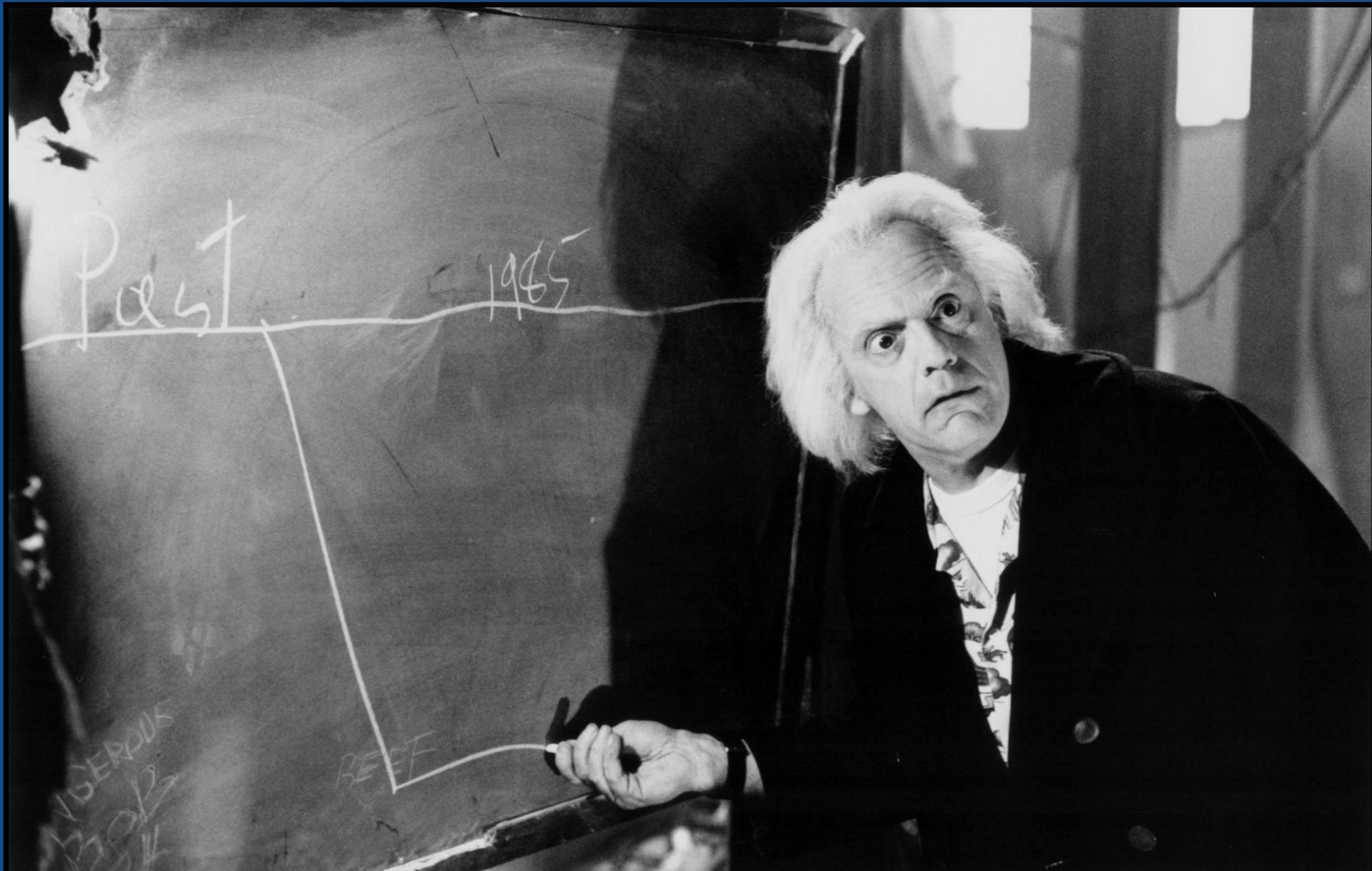
# Event Sourcing Allows For Time Travel



# 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

**EVENTS-FIRST DESIGN** helps you to:

- \* **REDUCE RISK** when **MODERNIZING** applications
- \* **MOVE FASTER** towards a **RESILIENT** and **SCALABLE** architecture
- \* **DESIGN AUTONOMOUS** services
- \* **BALANCE CERTAINTY** and **UNCERTAINTY**

# Key Takeaways

**EVENTS-FIRST DESIGN** helps you to:

- \* REDUCE RISK when MODERNIZING applications
- \* MOVE FASTER towards a RESILIENT and SCALABLE architecture
- \* DESIGN AUTONOMOUS services
- \* BALANCE CERTAINTY and UNCERTAINTY

**EVENT LOGGING** allows you to:

- \* AVOID CRUD and ORM
- \* TAKE CONTROL of your system's HISTORY
- \* TIME-TRAVEL
- \* BALANCE STRONG and EVENTUAL consistency



akka

<http://akka.io>

# Learn More

Download my latest book for free at:  
[bit.ly/reactive-microsystems](https://bit.ly/reactive-microsystems)



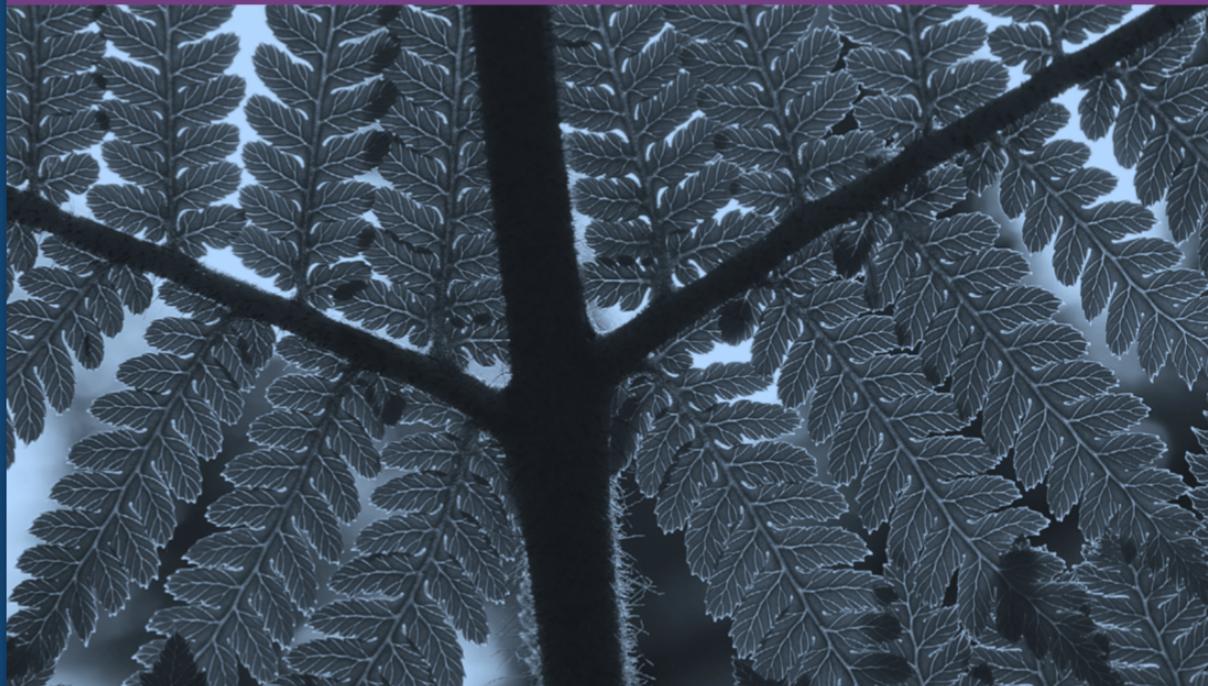
Lightbend

O'REILLY®

Compliments of  
Lightbend

# Reactive Microsystems

The Evolution of Microservices at Scale



Jonas Bonér