

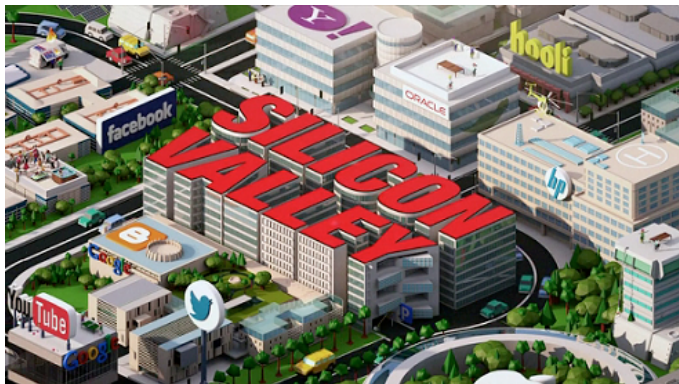
State or Events?

Kenny Bastani
Jakub Pilimon



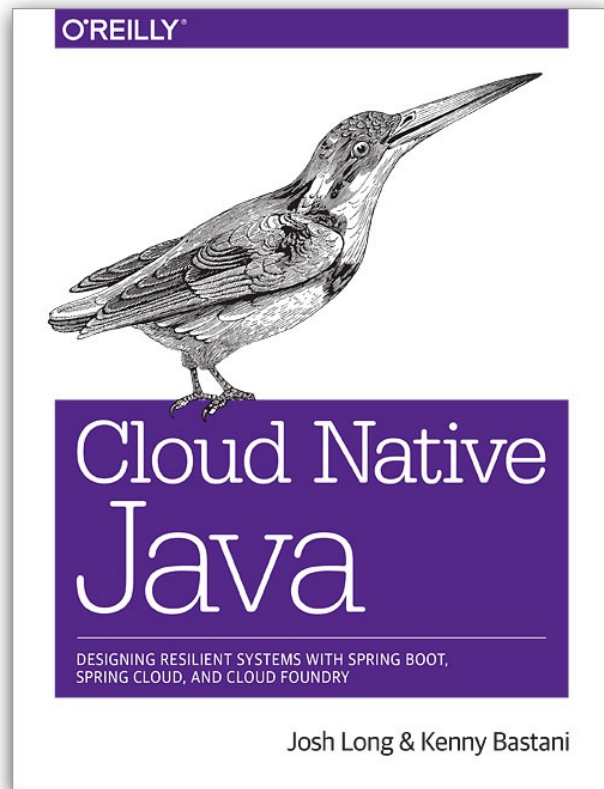
Kenny Bastani

Global CTO @Pivotal



spring 

Pivotal®



About me

Spring Developer Advocate @Pivotal

Interested in

- **Domain-Driven Design**
- **Architecture**
- **Test-Driven Development**

Twitter: @JakubPilimon

Blog: <http://pillopl.github.io>

#dddbyexamples



SZCZEBRZESZYN CHRZĄSZCZYŻEWOSZYCE



The agenda

- Event sourcing, a history
- Why event sourcing?
- Apache Kafka and the immutable log
- Event-driven microservices
- Reference architecture

Event sourcing

A mystery, a history...

Origin story

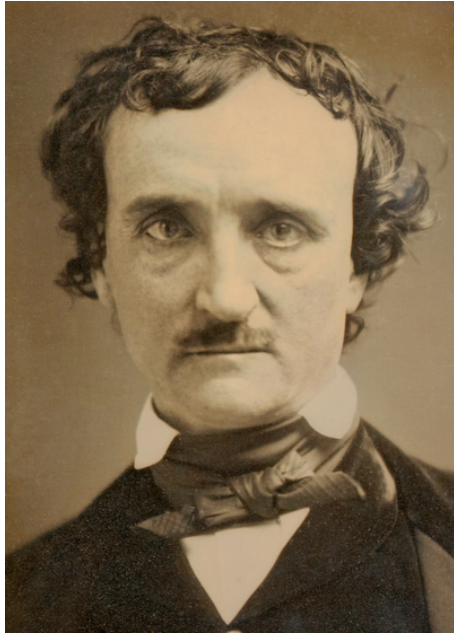
Event Sourcing ensures that all changes to application state are stored as a **sequence of events**. Not only can we query these events, we can also use the **event log** to reconstruct past states.

We can determine the application state at any point in time. We do this by starting with a blank state and **rerunning the events up to a particular time**.

–Martin Fowler

Did Martin Fowler invent event sourcing?

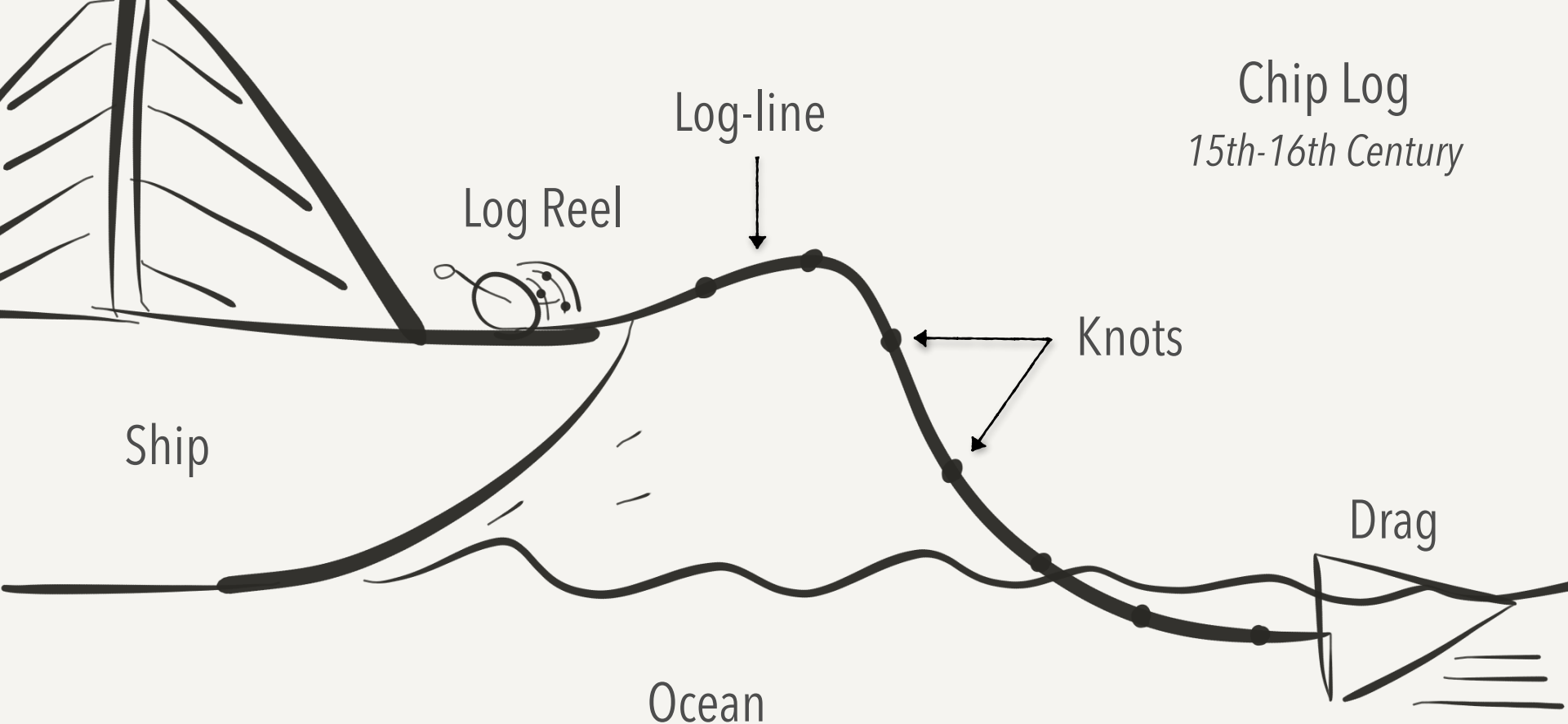
Edgar Allan Poe (1804-1849)



We appreciate **time** by **events** alone. For this reason we define time (somewhat improperly) as the **succession of events**; but the fact itself--that events are our sole means of appreciating time-- [leads to] the erroneous idea that events are time--that the more numerous the events, the longer the time; and the converse.

Did Edgar Allan Poe invent event sourcing?

Lol no...



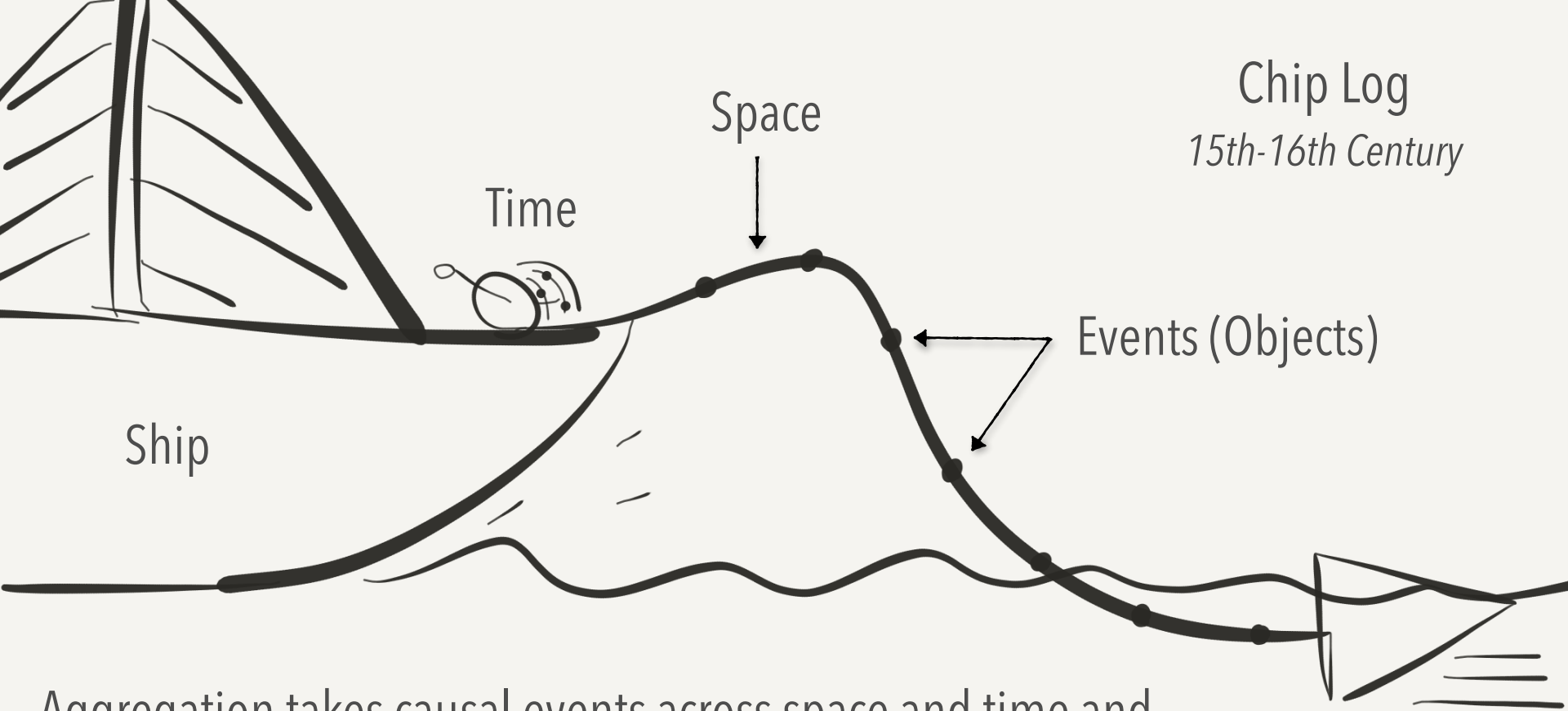
The Poe connection

- Poe was obsessed with the idea of event sourcing for much of his life.
- But in his time, things were commonly measured using a form of event sourcing, like the knots on the log-line for measuring the speed of ships.
- Most software is built on metaphors. The problem is, we've forgotten the metaphors have meaning.

Marginalia, Edgar Allan Poe, 1849

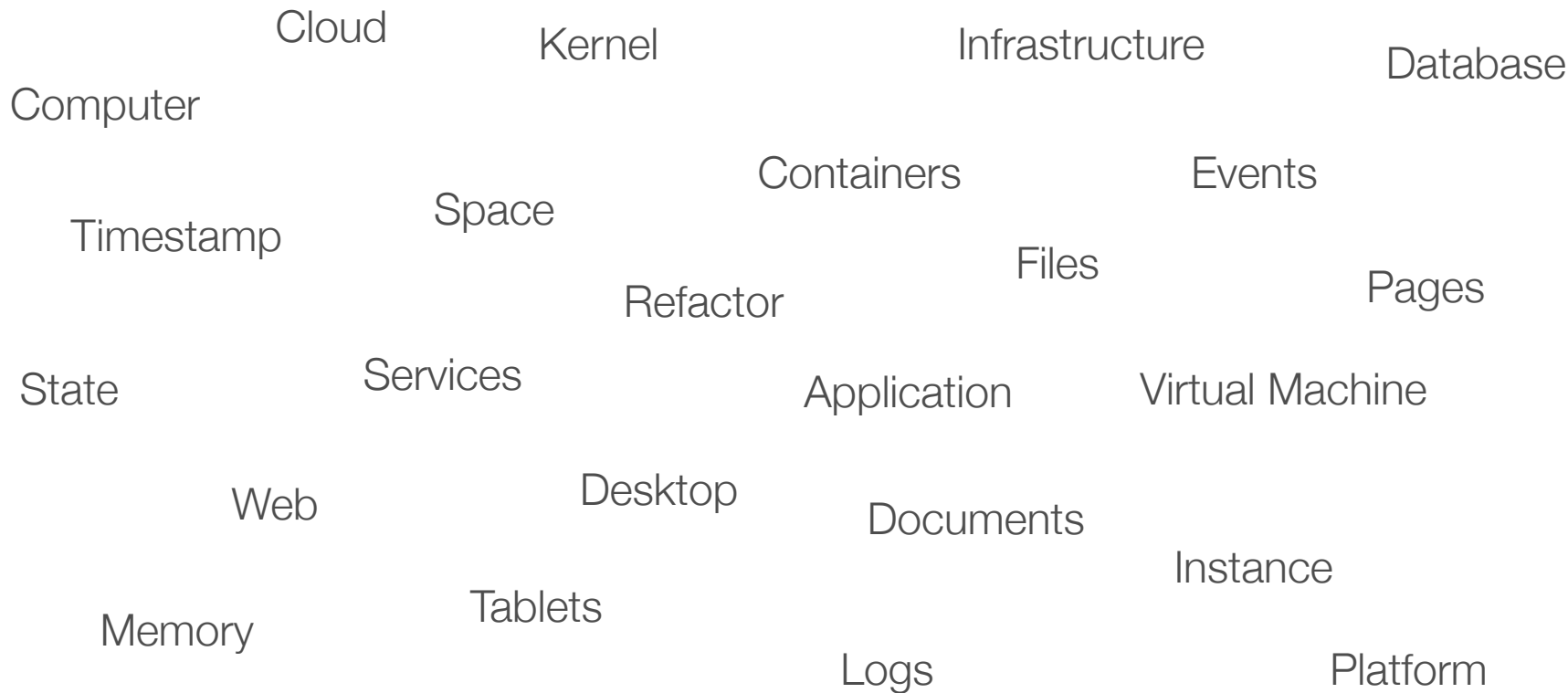
Space is precisely analogous with time. By objects alone we estimate space; and we might as rationally define it “the succession of objects” as time “the succession of events.” But, as before. — The fact, that we have no other means of estimating space than objects afford us — tends to the false idea that objects *are* space — that the more numerous the objects the greater the space; and the converse; and this erroneous impression we should receive in all cases, but for our practical means of correcting it — such as yard measures, and other conventional measures, which resolve themselves, ultimately, into certain natural standards, such as barleycorns, which, after all, we only *assume* to be regular.

Space is precisely analogous with time. By objects alone we estimate space ; and we might as rationally define it “the succession of objects” as time “the succession of events.” But, as before. — The fact, that we have no other means of estimating space than objects afford us — tends to the false idea that objects *are* space — that the more numerous the objects the greater the space ; and the converse ; and this erroneous impression we should receive in all cases, but for our practical means of correcting it — such as yard measures, and other conventional measures, which resolve themselves, ultimately, into certain natural standards,



Aggregation takes causal events across space and time and summarizes it as a single object that's easier to store in memory

All great software is built on metaphors



A word cloud of software-related metaphors. The words are arranged in a roughly circular pattern, with some words appearing more frequently or in larger fonts than others. The words include:

- Cloud
- Kernel
- Infrastructure
- Database
- Events
- Pages
- Virtual Machine
- Instance
- Platform
- Logs
- Tablets
- Memory
- Web
- State
- Services
- Refactor
- Files
- Containers
- Space
- Timestamp
- Computer

The metaphors mean something

- There's an underlying abstraction to our own humanity and it has an intimate connection to the virtual world we are building
- The desktop computer replaced actual desktops by replacing the abstract functionality of the desktop
- The virtual machine replaced the server by virtualizing the abstract functionality of the hardware

But ultimately, it's hard to keep track of all the layers.

Eventually we get lost in abstraction.

All confusing software is built on nonsense

Serverless

Middleware

Microservices

ESB

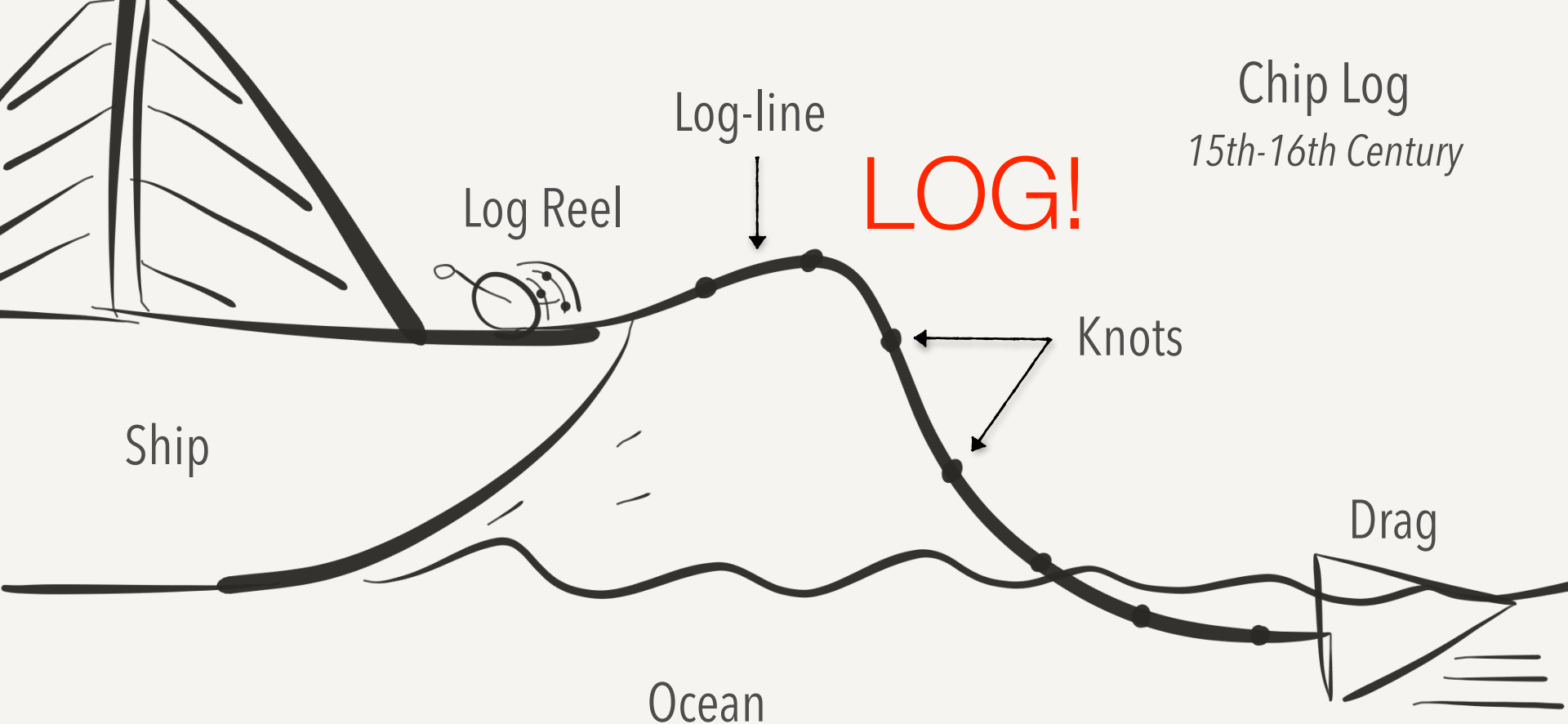
NoSQL

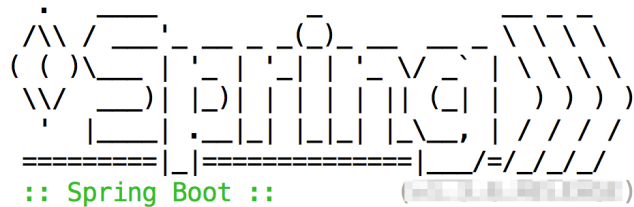
SOA

DevOps

I'm not saying microservices, DevOps,
and serverless are nonsense.

I'm saying that metaphors make good ideas more *durable* to **nonsensical interpretation.**





LOG!

```
2018-02-07 17:05:31.107 INFO [user-service,,,] 55183 --- [main] c.c.c.ConfigServicePropertySourceLocator : Fetching
config from server at: http://localhost:8888
2018-02-07 17:05:31.204 WARN [user-service,,,] 55183 --- [main] c.c.c.ConfigServicePropertySourceLocator : Could not
locate PropertySource: I/O error on GET request for "http://localhost:8888/user-service/development": Connection refused; nested
exception is java.net.ConnectException: Connection refused
2018-02-07 17:05:31.205 INFO [user-service,,,] 55183 --- [main] demo.UserApplication : The following
profiles are active: development
2018-02-07 17:05:31.226 INFO [user-service,,,] 55183 --- [main] ationConfigEmbeddedWebApplicationContext : Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@58d342cd: startup date [Wed Feb 07 17:
05:31 CST 2018]; parent: org.springframework.context.annotation.AnnotationConfigApplicationContext@54613c47
2018-02-07 17:05:32.267 INFO [user-service,,,] 55183 --- [main] o.s.b.f.s.DefaultListableBeanFactory : Overriding
bean definition for bean 'hystrixFeature' with a different definition: replacing [Root bean: class [null]; scope=; abstract=false;
lazyInit=false; autowireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=org.springframework
cloud.netflix.hystrix.HystrixCircuitBreakerConfiguration$HystrixWebConfiguration; factoryMethodName=hystrixFeature; initMethodN
ame=null; destroyMethodName=(inferred); defined in class path resource [org/springframework/cloud/netflix/hystrix/HystrixCircu
itBreakerConfiguration$HystrixWebConfiguration.class]] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; au
towireMode=3; dependencyCheck=0; autowireCandidate=true; primary=false; factoryBeanName=org.springframework.cloud.netflix.hystri
x.HystrixCircuitBreakerConfiguration; factoryMethodName=hystrixFeature; initMethodName=null; destroyMethodName=(inferred); defin
ed in class path resource [org/springframework/cloud/netflix/hystrix/HystrixCircuitBreakerConfiguration.class]]
2018-02-07 17:05:32.358 INFO [user-service,,,] 55183 --- [main] o.s.b.f.s.DefaultListableBeanFactory : Overriding
```

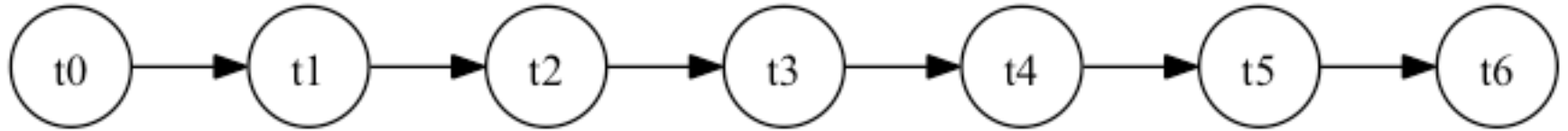

LOG!



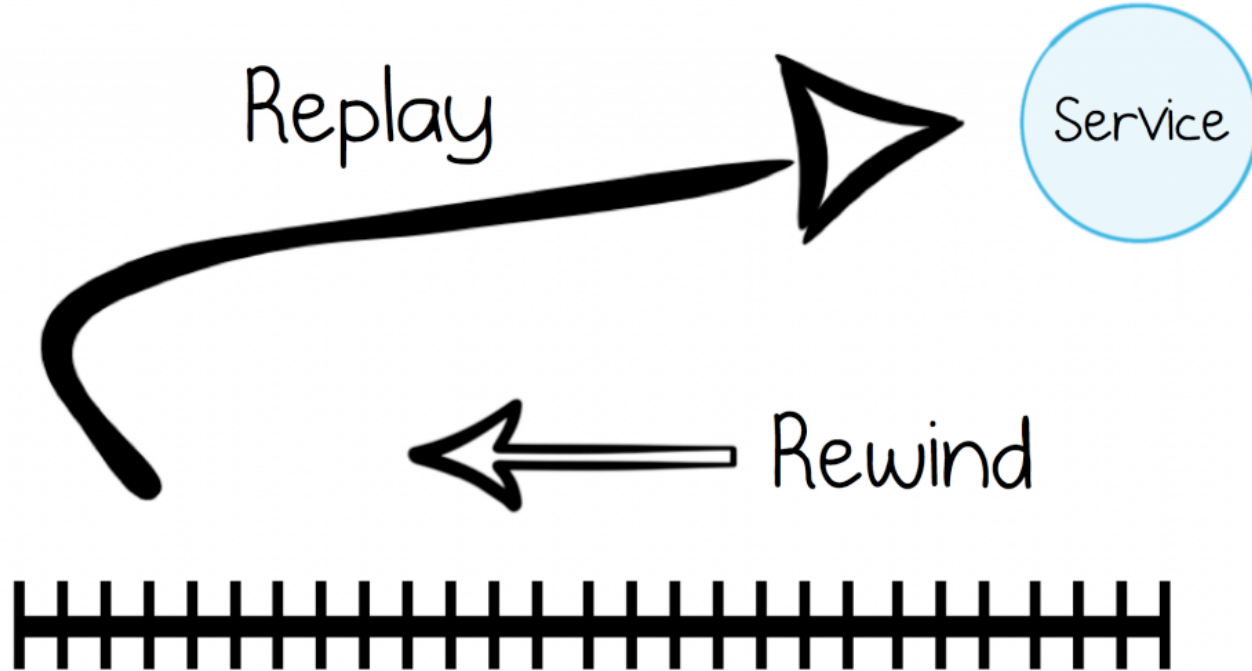
Apache Kafka

The immutable log as the source of truth

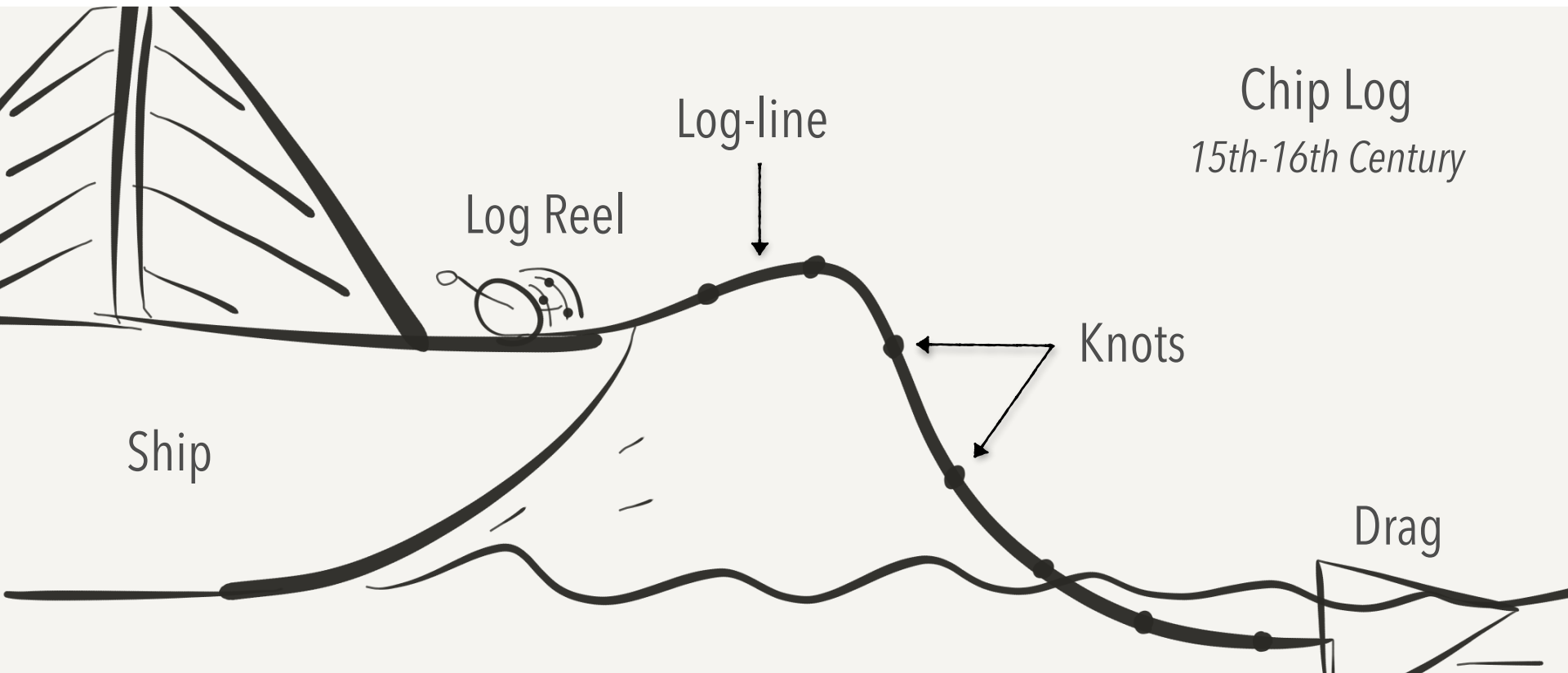
Time-ordered events

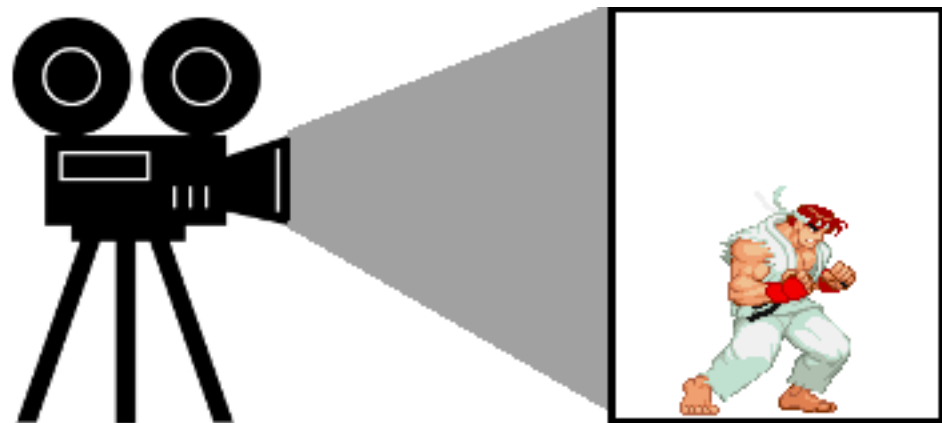


Store state as a log of events



It's the same thing, but way before software

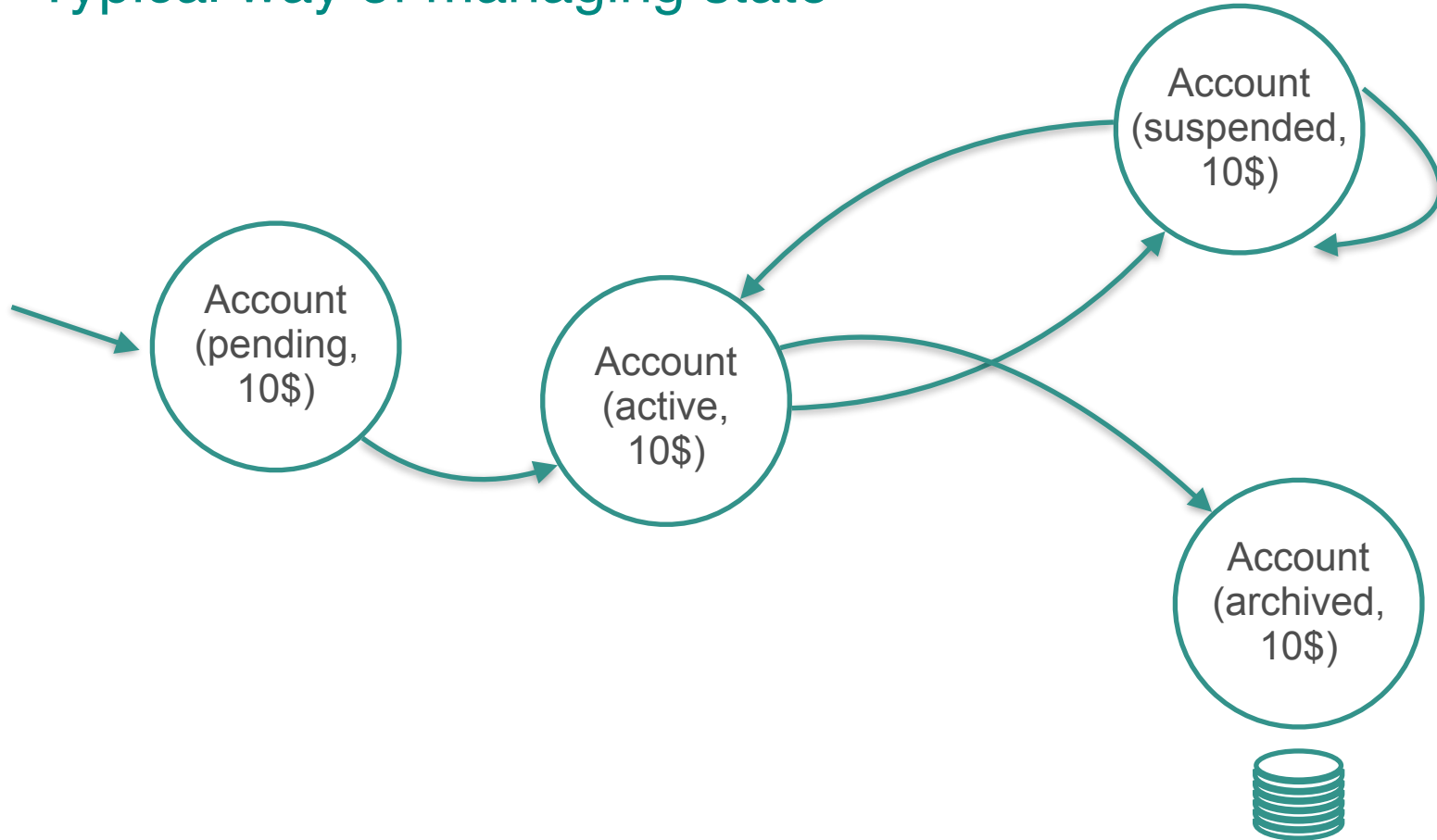




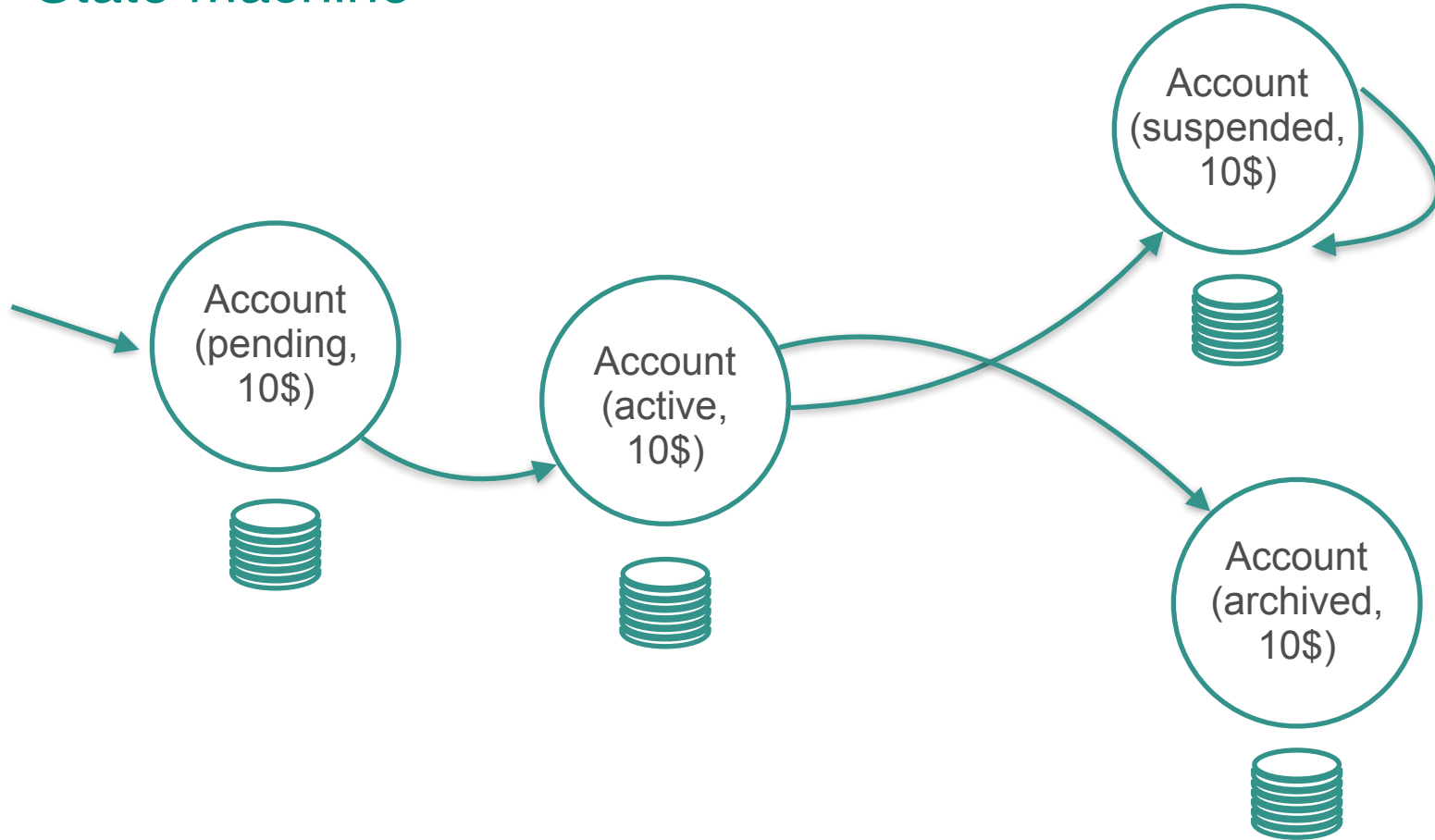
Motivations

What's the benefit of managing state in a distributed system?

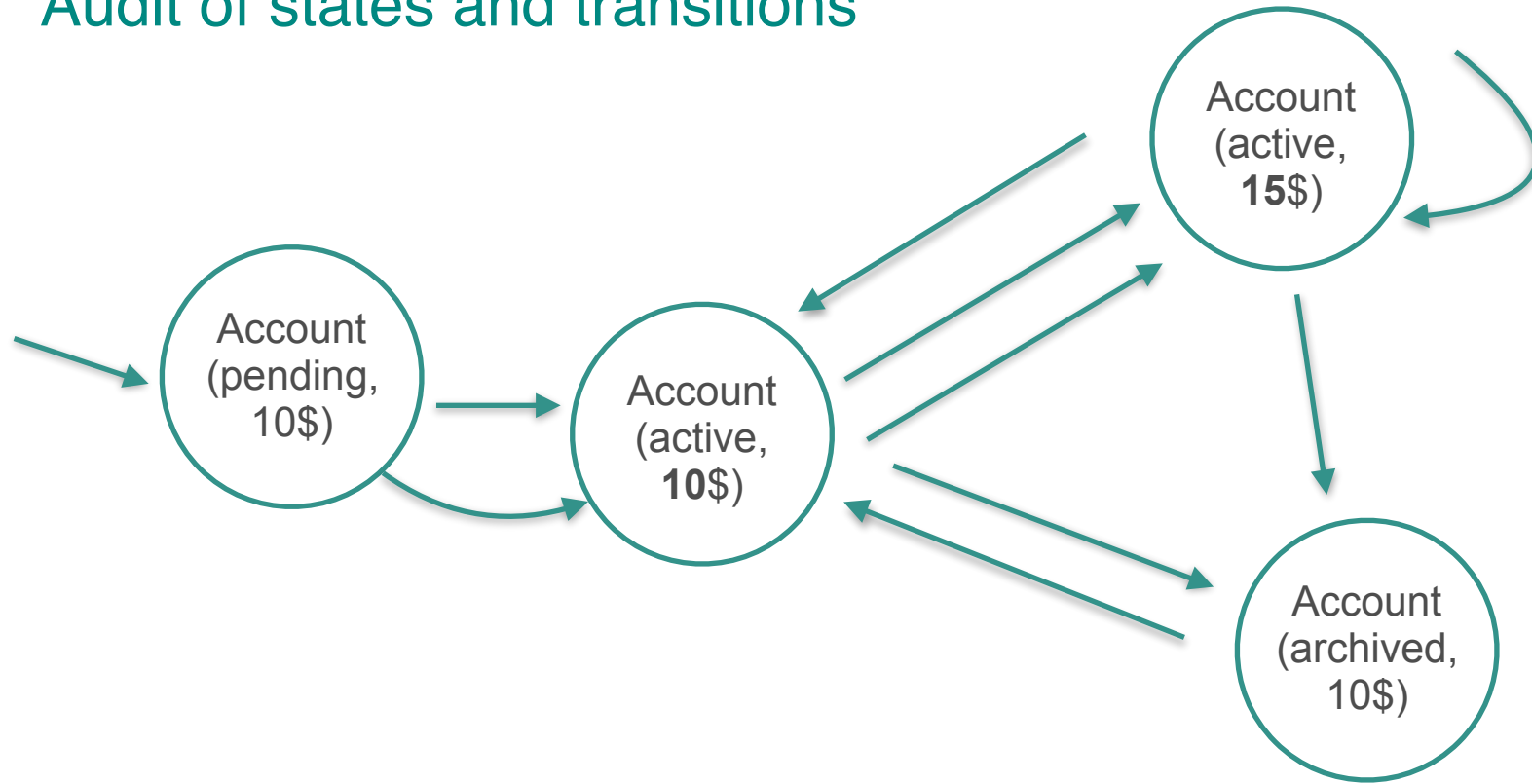
Typical way of managing state



State machine



Audit of states and transitions



Event Sourcing

- ▶ **Aggregates** can be used to generate the **consistent state** of any object
- ▶ It provides an **audit trail** that can be replayed to generate the state of an object from any point in time
- ▶ It provides the many inputs necessary for analyzing data using **event stream processing**
- ▶ It enables the use of **compensating transactions** to rollback events leading to an inconsistent application state

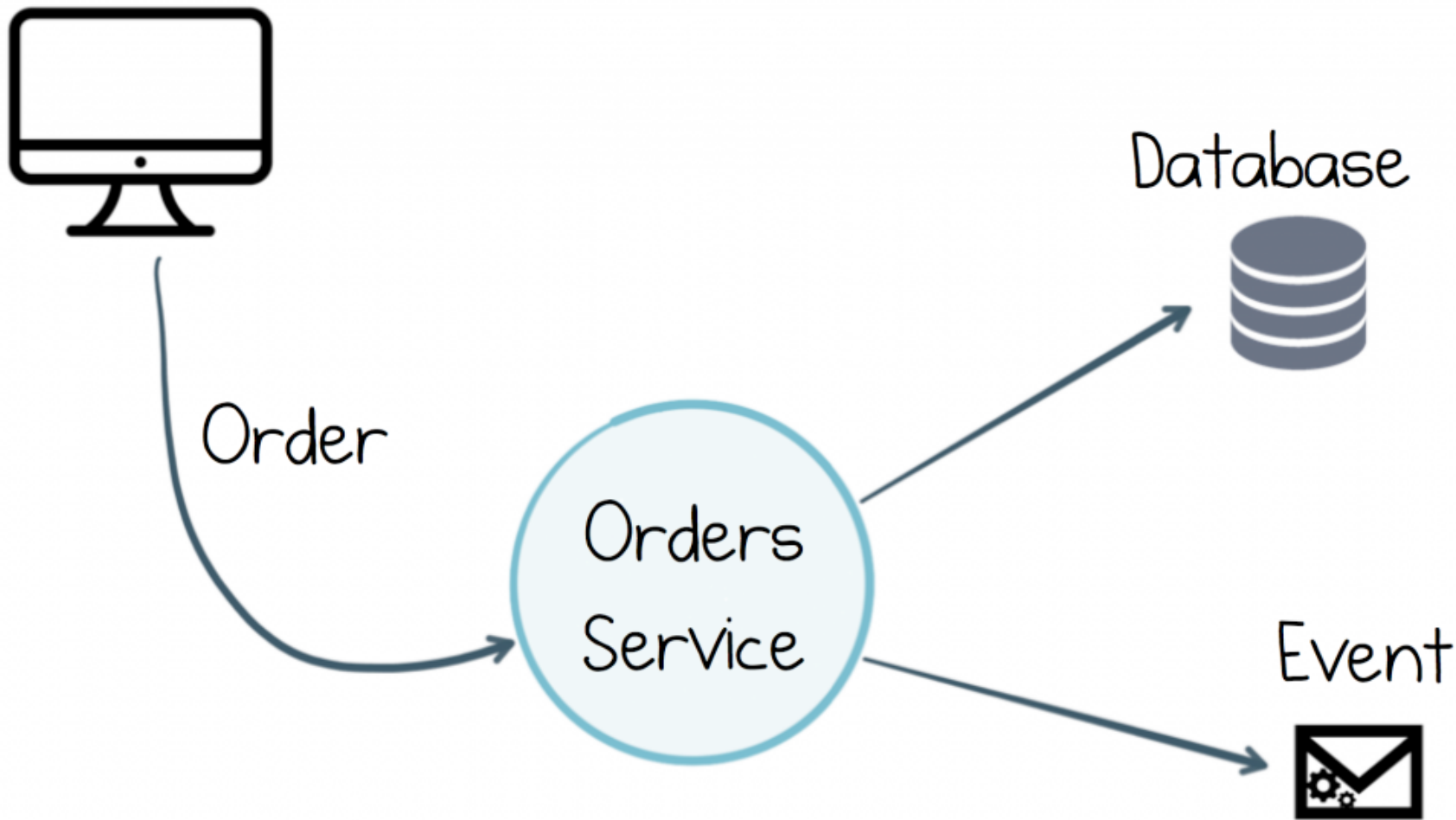


So why aren't we doing it everywhere?

Because storing state as an immutable log of events for domain data didn't make sense in a time B.C.

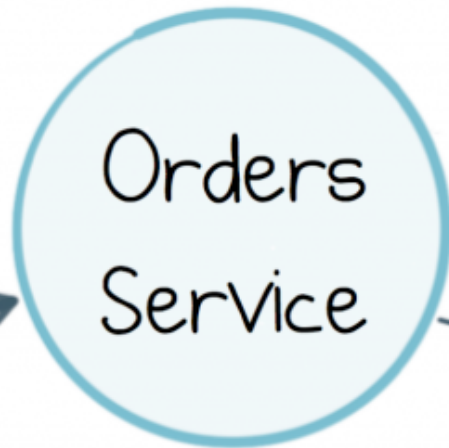
B.C. = Before Cloud

– Andrew Clay Shafer



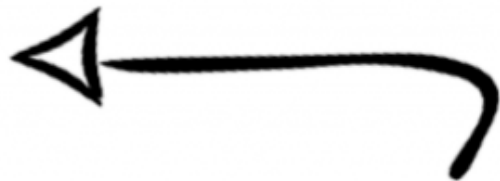


Order



Orders
Service

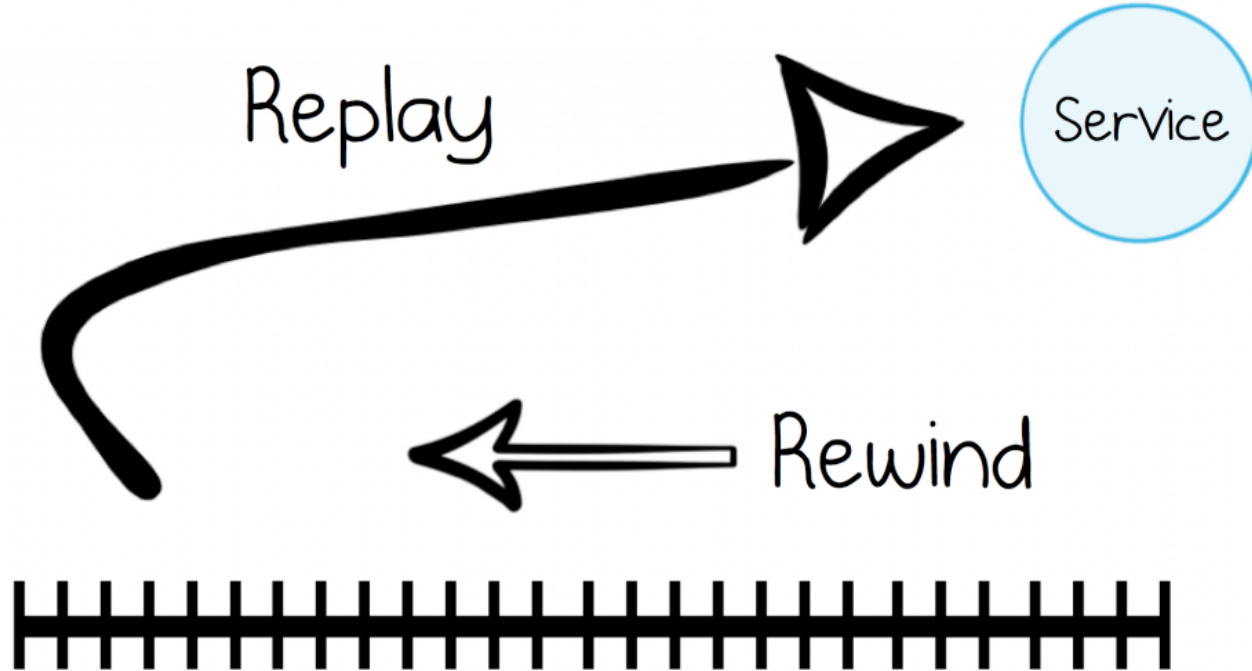
Derive state solely
from the events



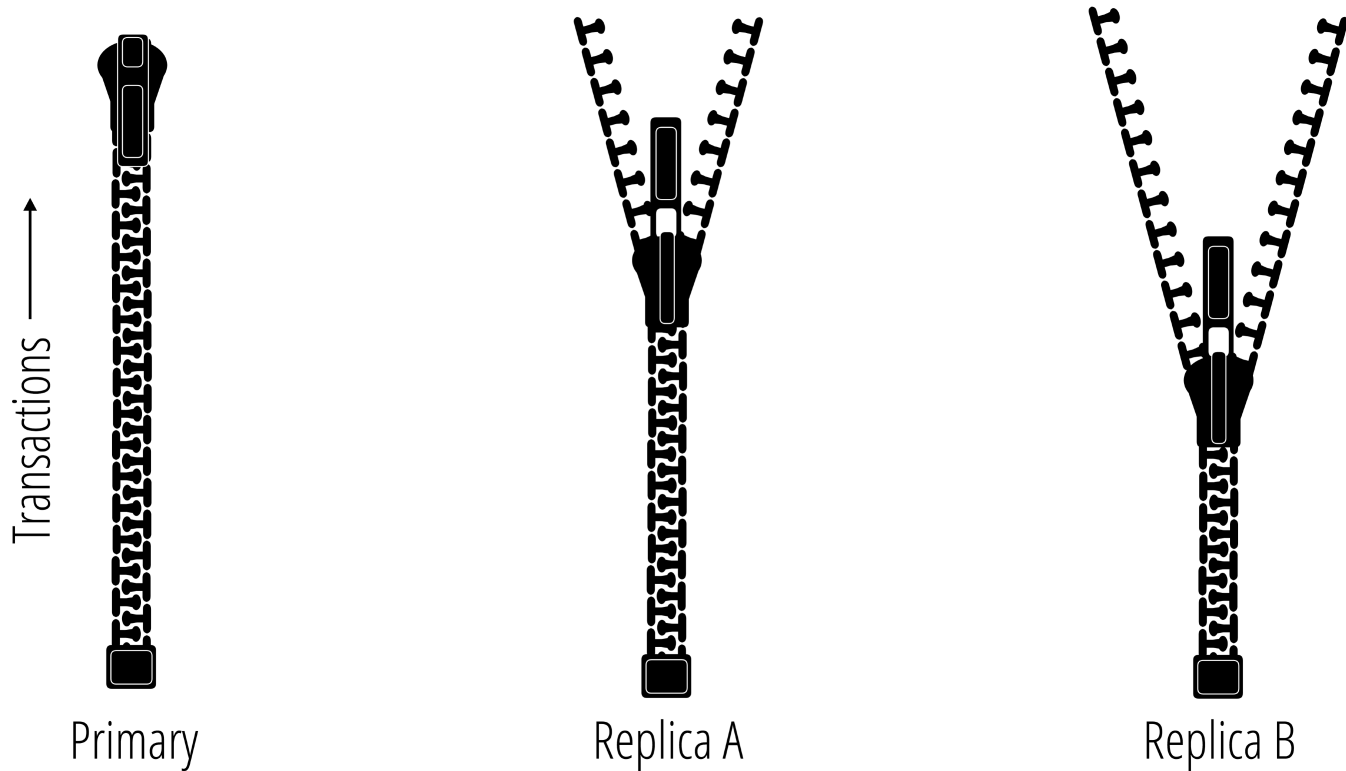
Event



The Immutable Log



The Immutable Log



Event Sourcing and GDPR

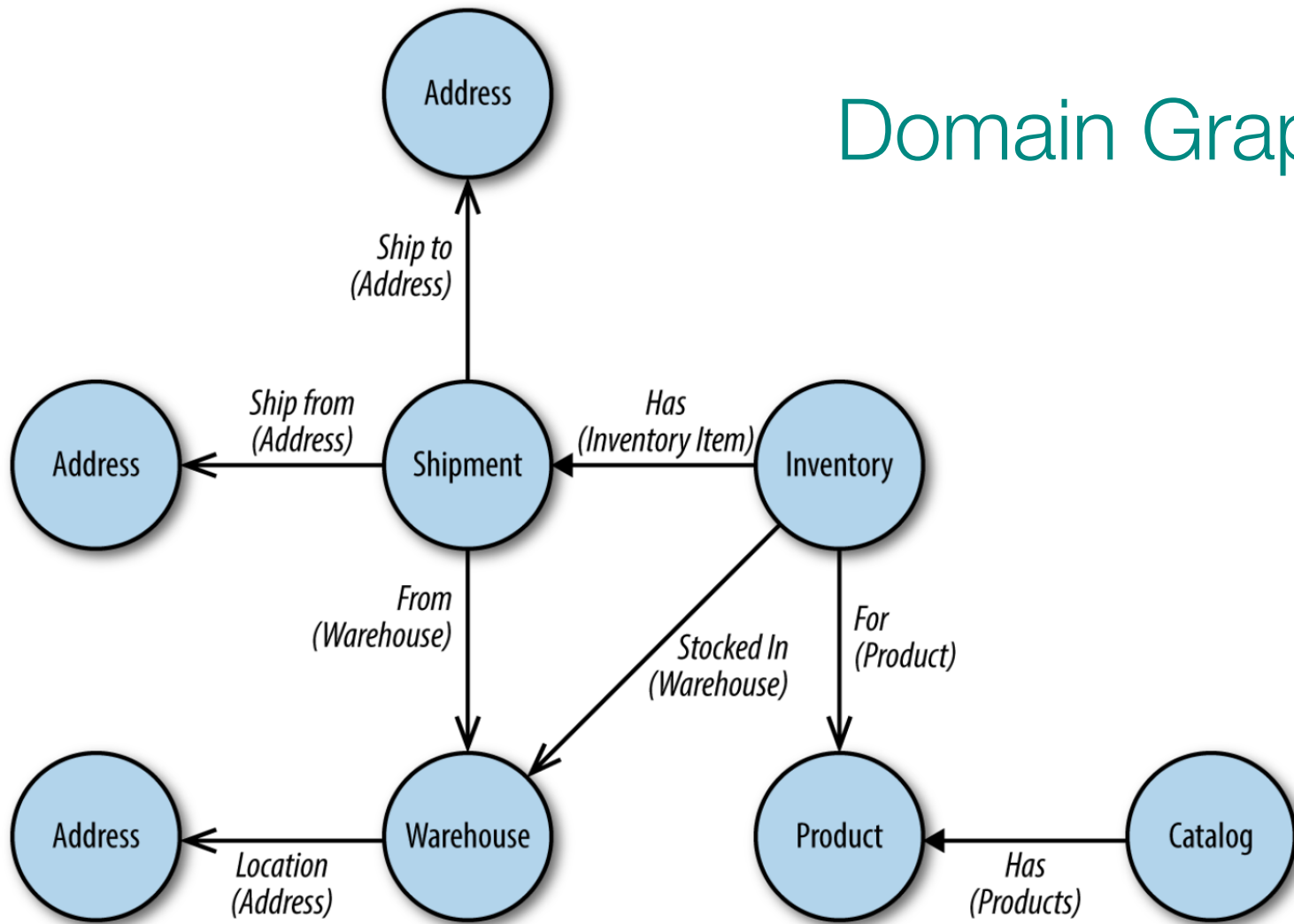
What if somebody requests to have their immutable data removed from a log?



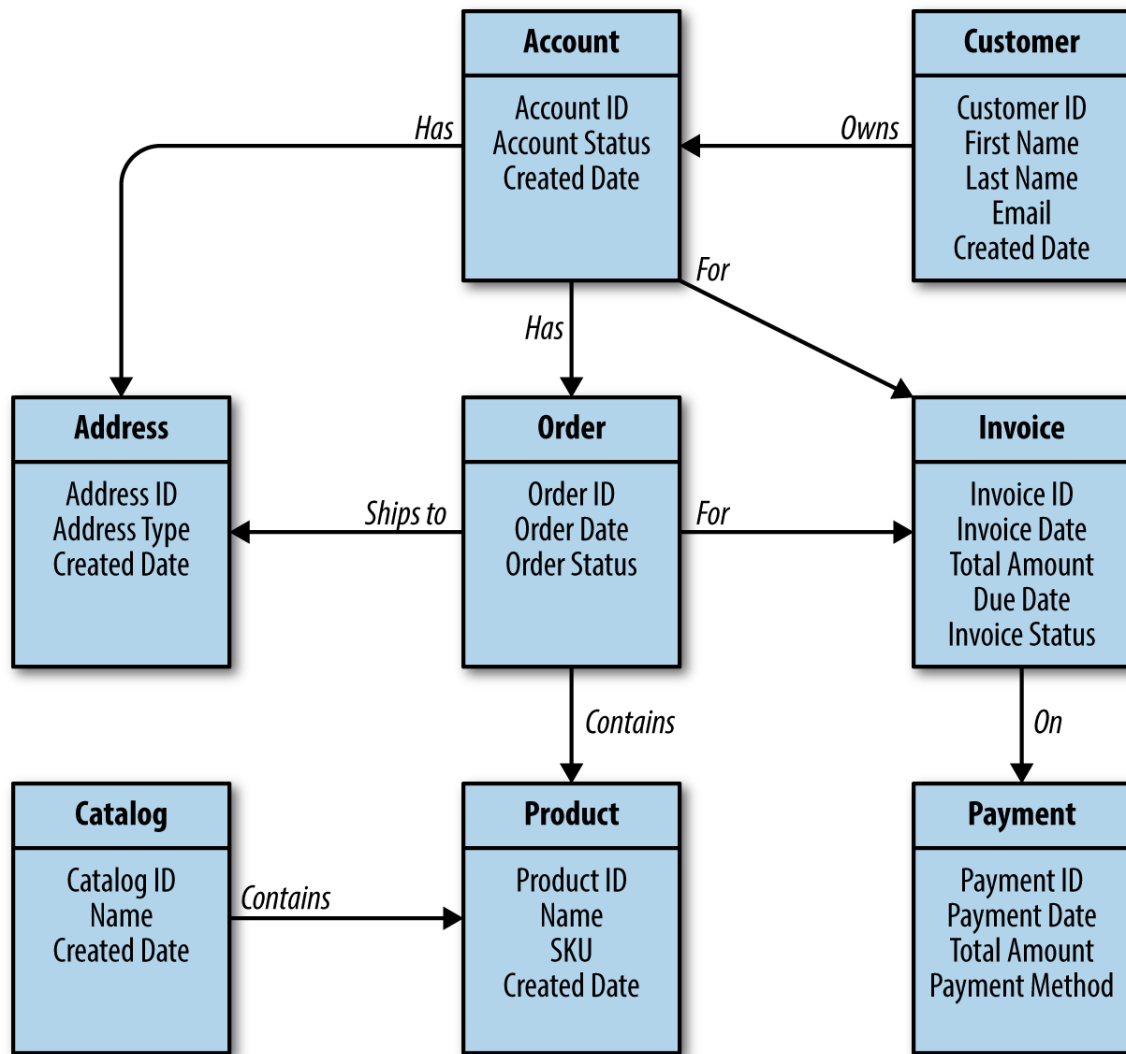
How about event-driven (micro)services?

There are no foreign key constraints between (micro)services.

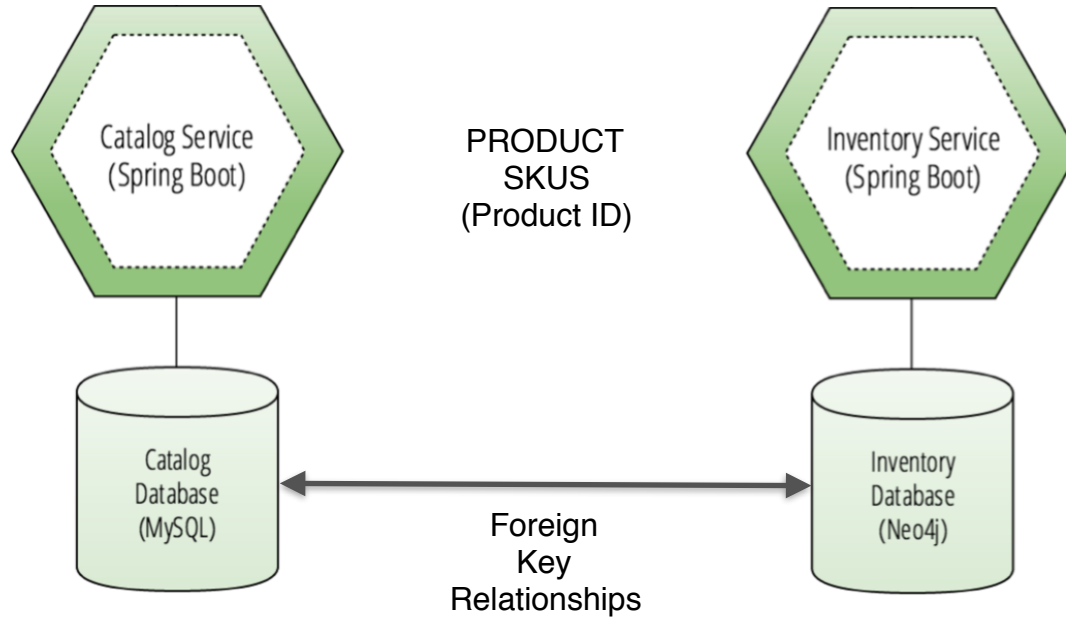
Domain Graph



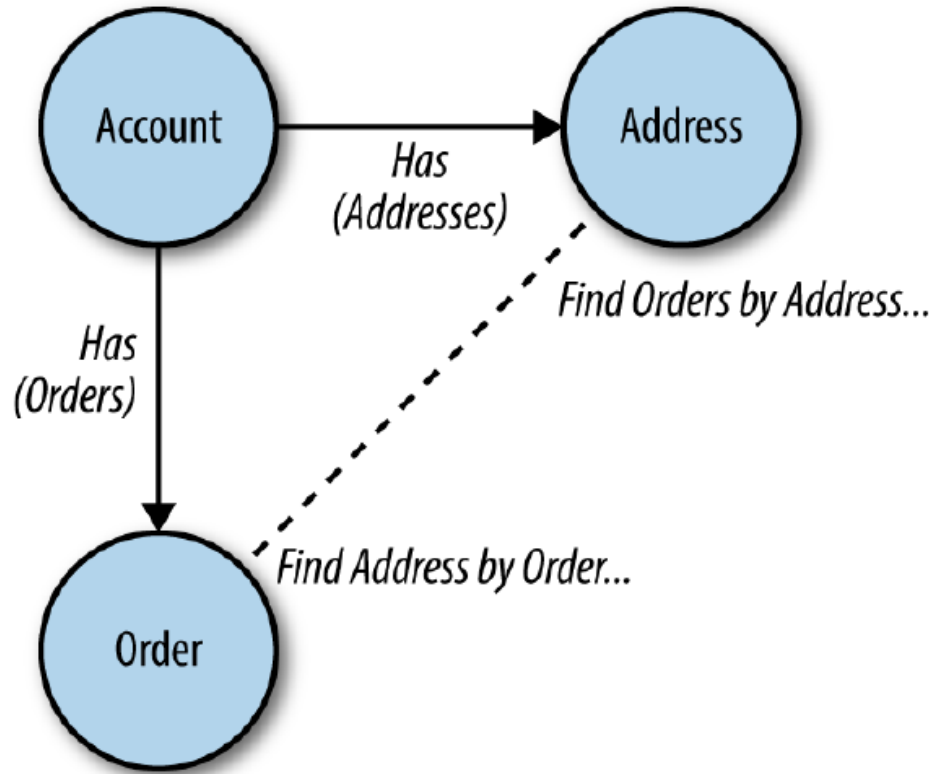
Domain Relationships



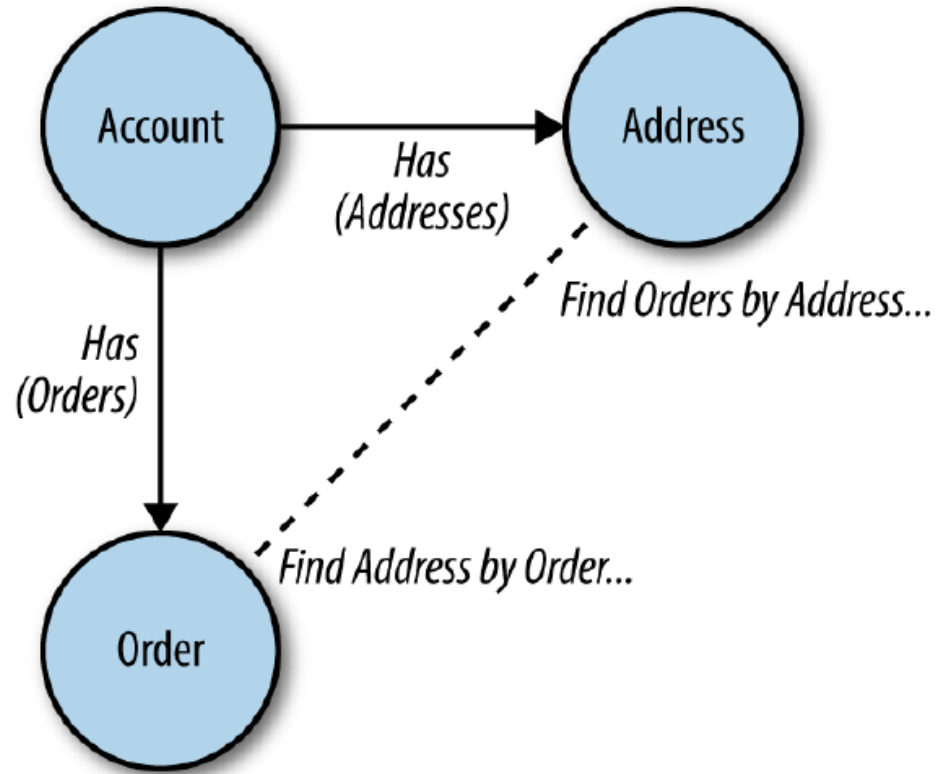
No foreign key constraints across services



Distributed joins are slow (reads)

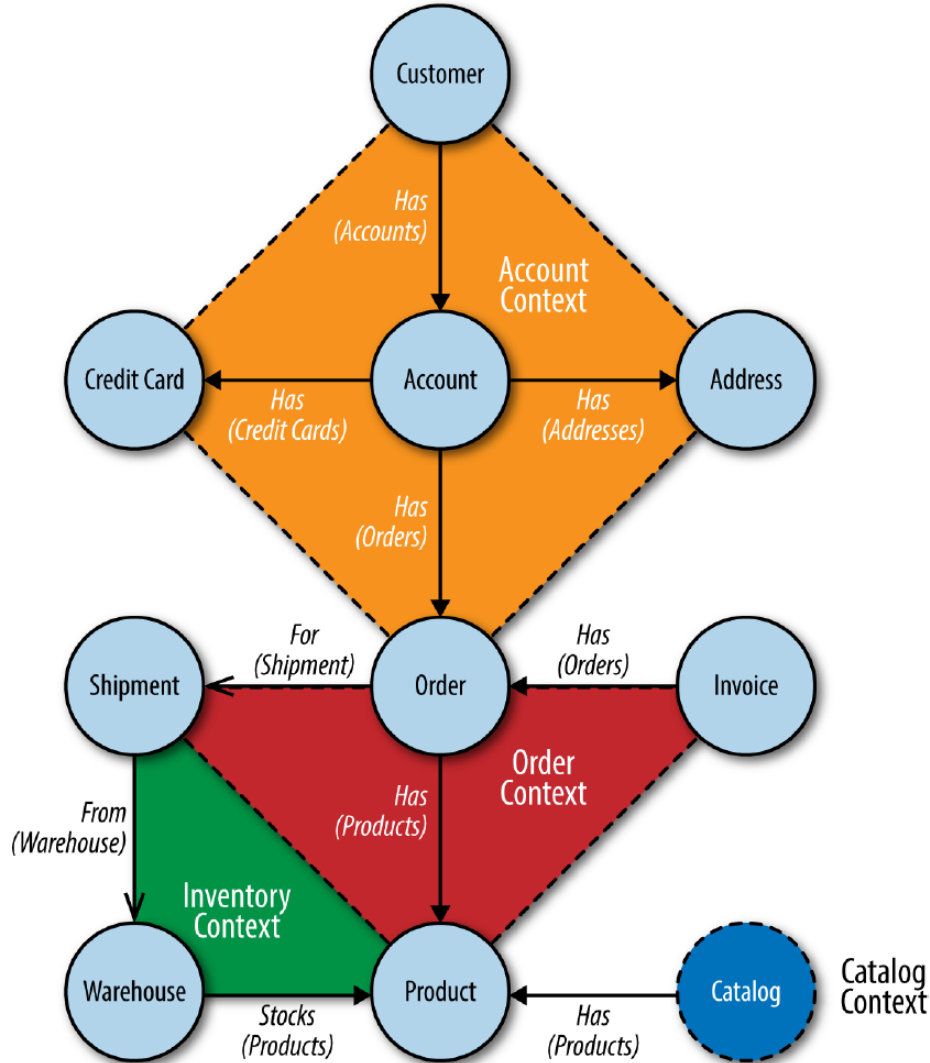


Distributed transactions are brittle (writes)



(Micro)service joins

- Services are organized using domain-driven design
- Services cannot own domain data from two different bounded contexts
- Complex joins (reads) are inevitable
- Avoid distributed transactions (write)



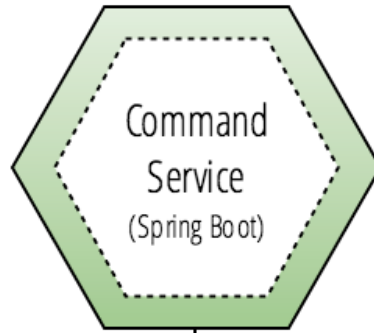
What is CQRS?

CQRS (Command Query Responsibility Segregation) is a pattern for separating handlers for commands and queries.

Handlers synchronize by turning events into materialized views.

How do I do CQRS in a distributed system?

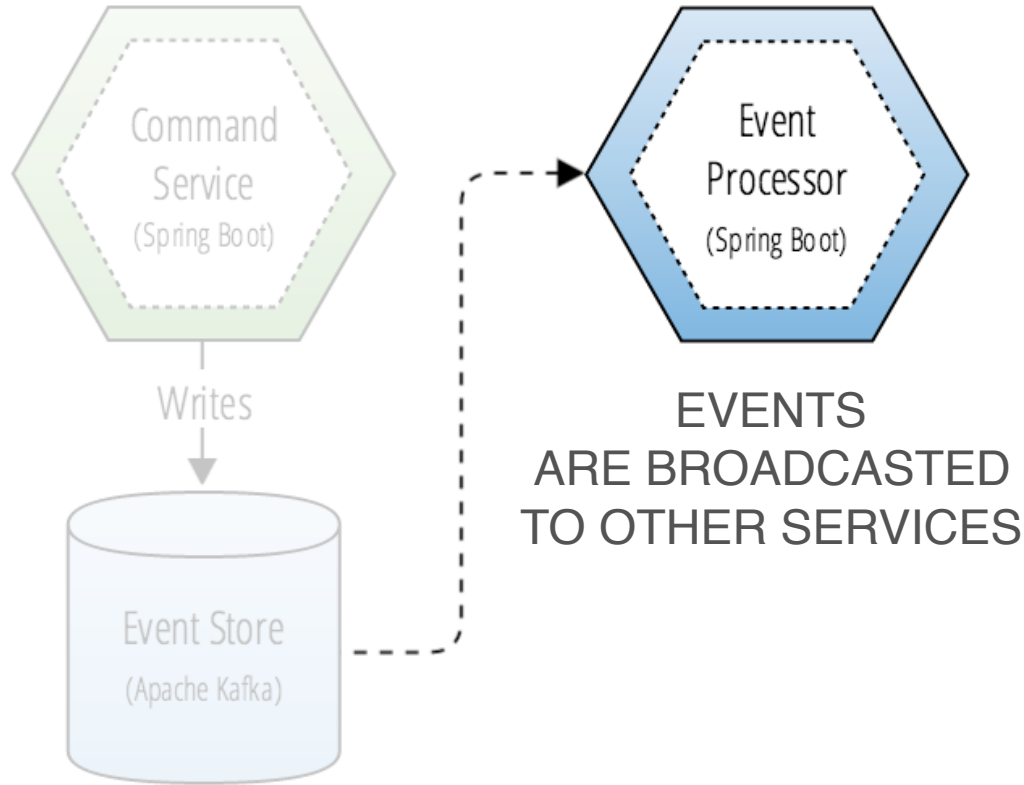
Let's look at a system architecture that uses
Spring Boot

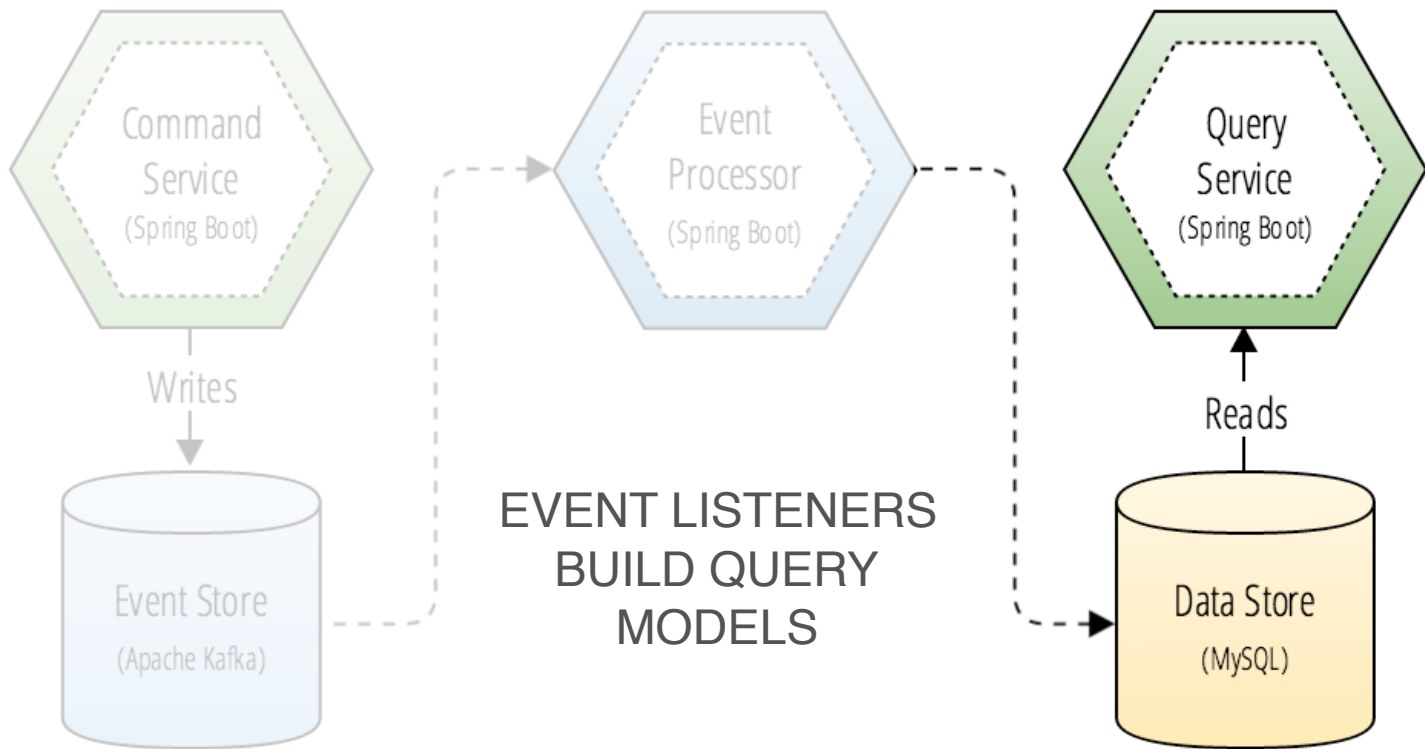


Writes

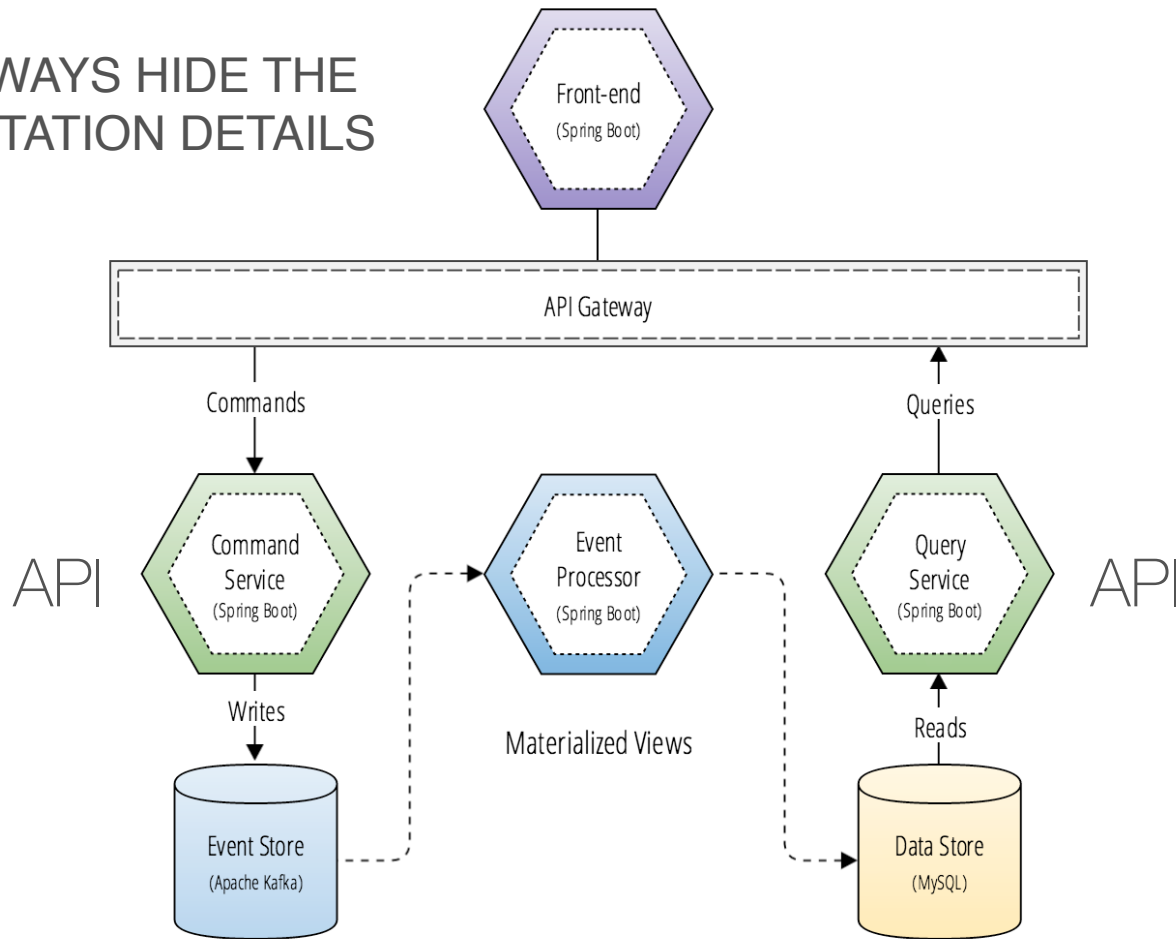


COMMANDS
GENERATE
EVENTS

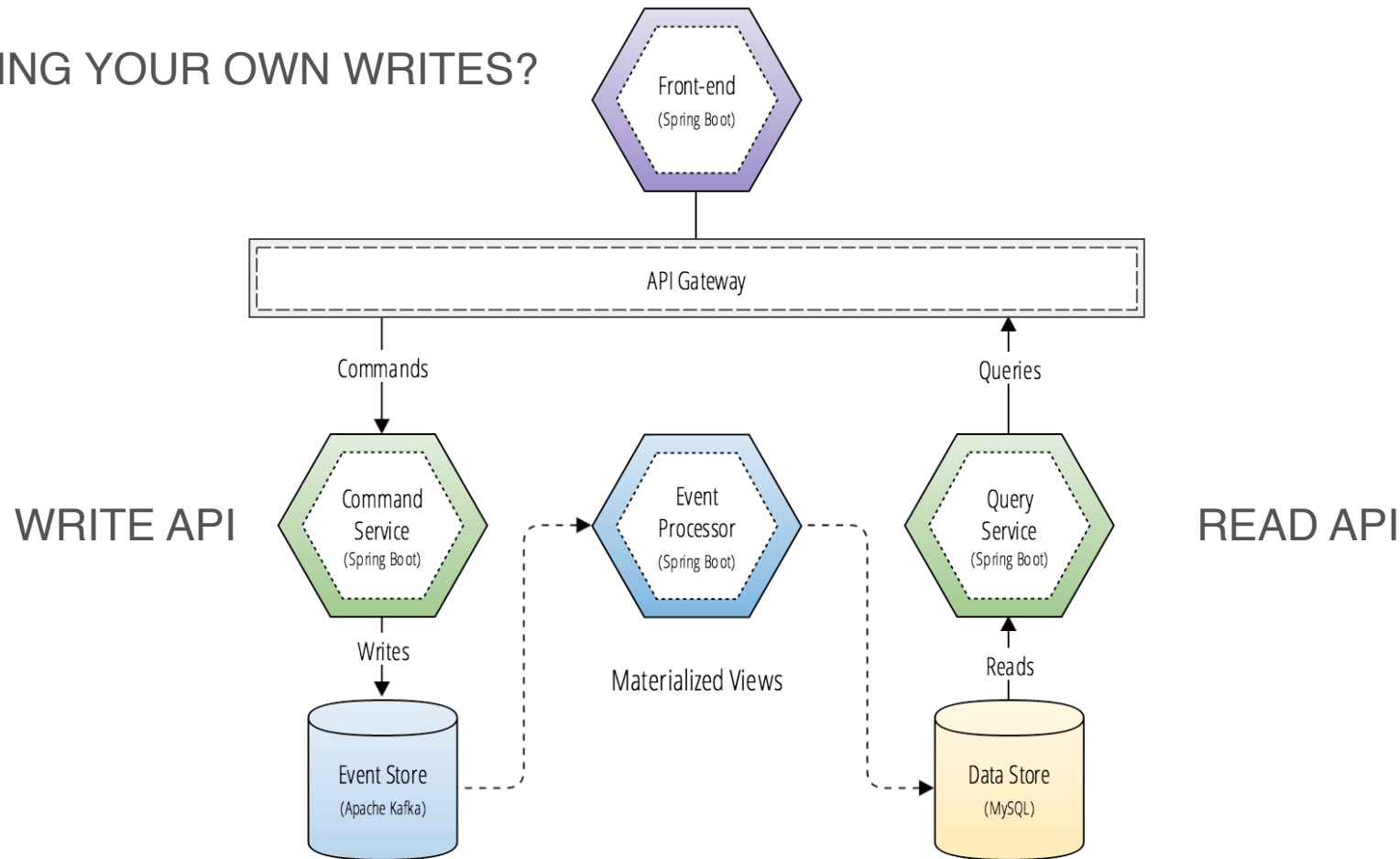




API GATEWAYS HIDE THE IMPLEMENTATION DETAILS



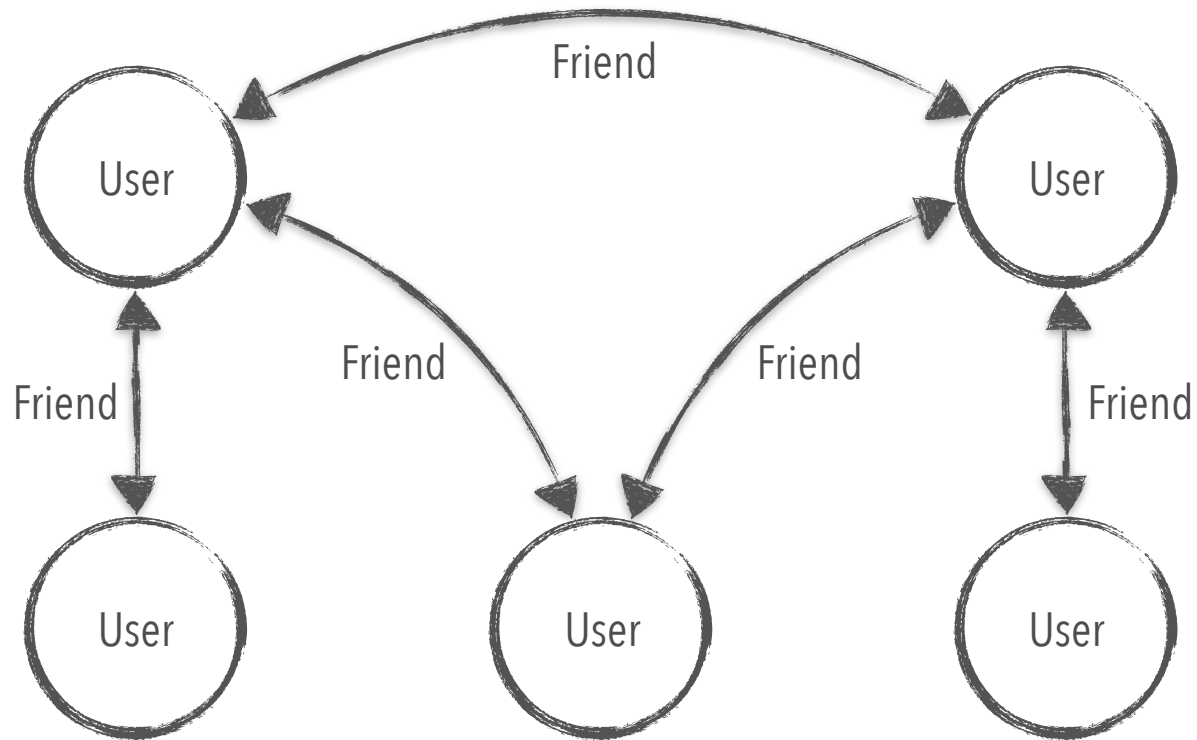
READING YOUR OWN WRITES?



Reference architecture

Creating a social network as microservices

How do you do friend of a friend queries with microservices?

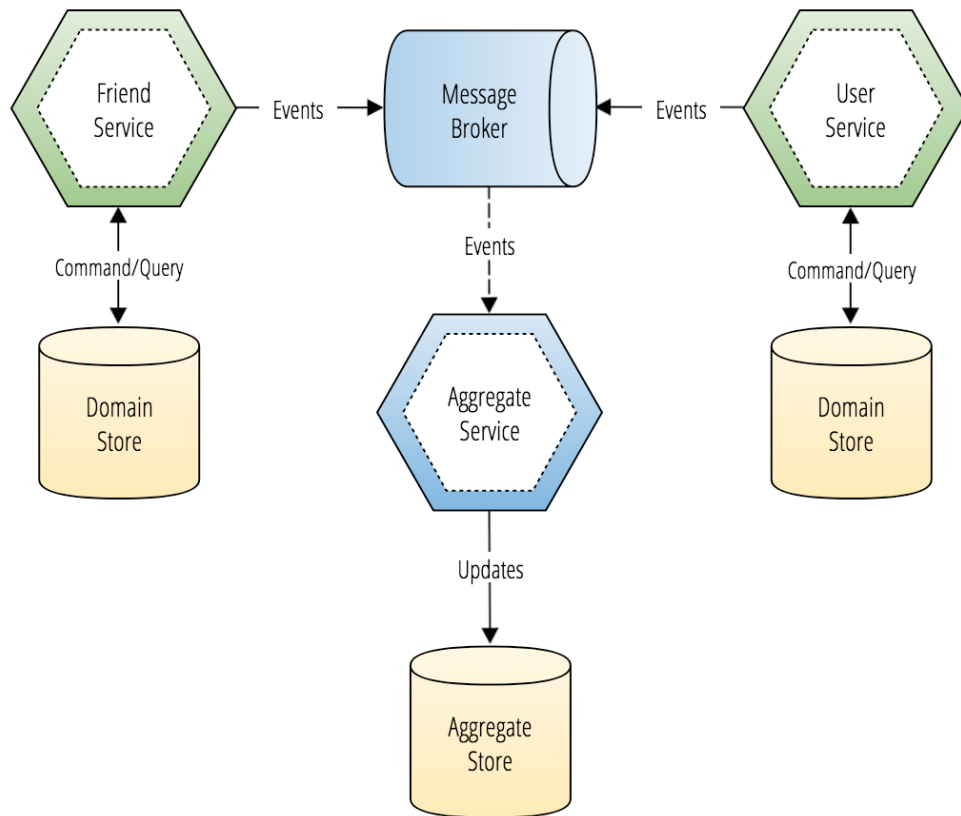


ices

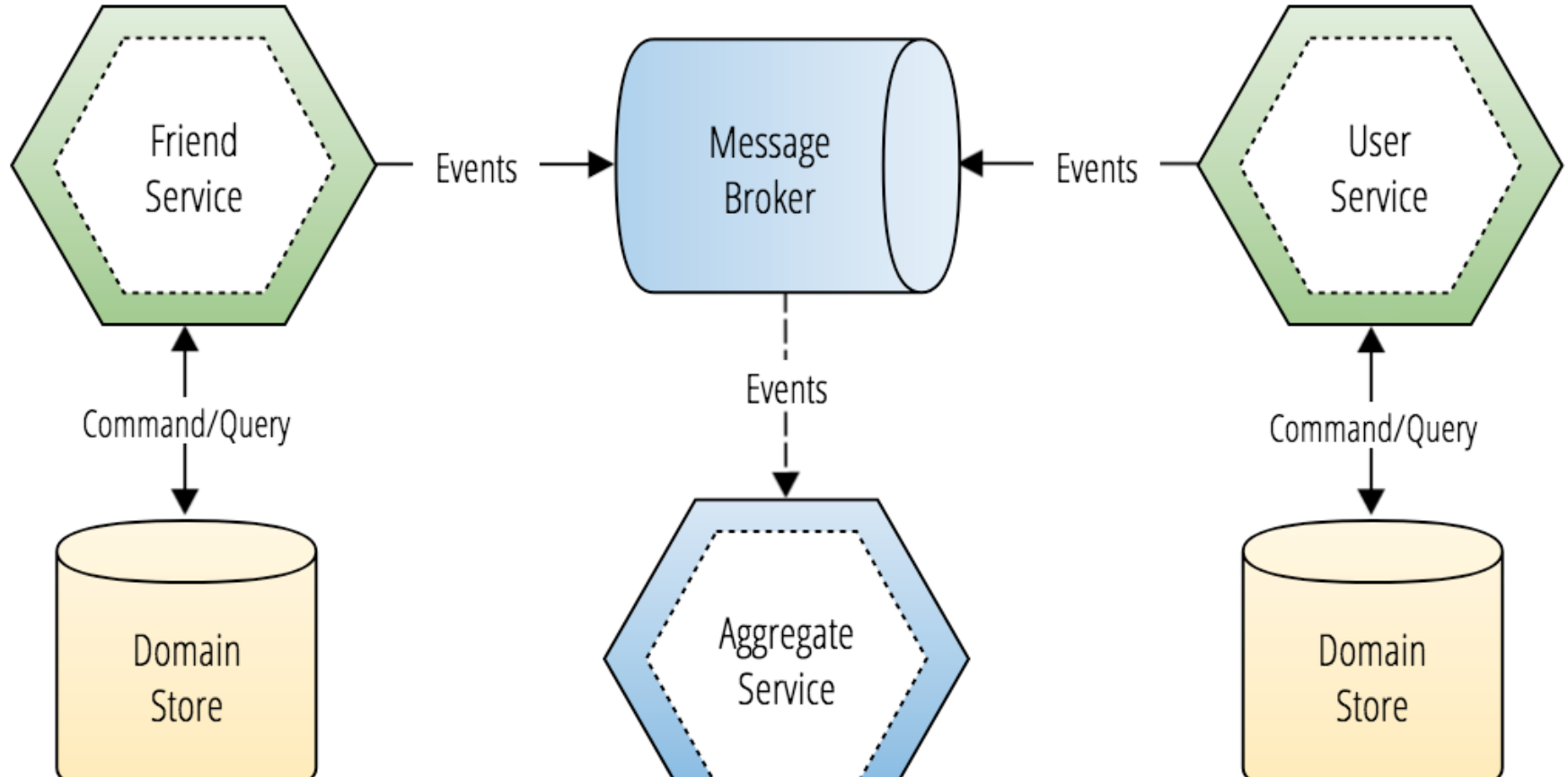
Day 2 of building microservices...

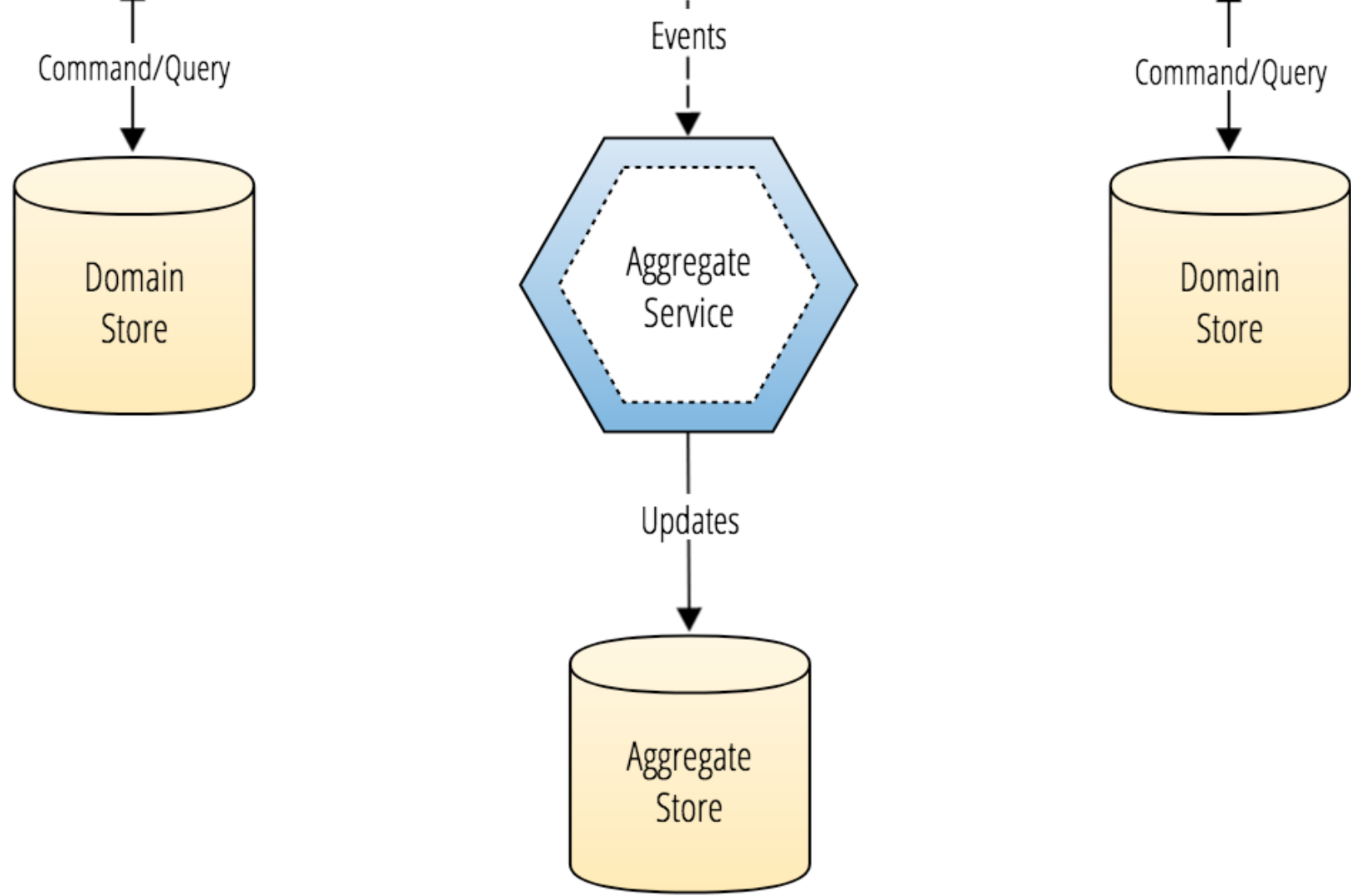


Materialized views



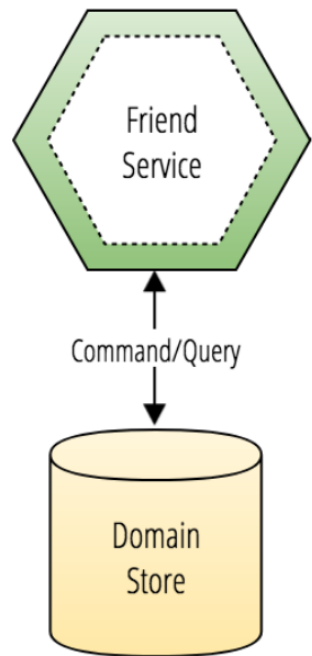
Create aggregate views from event data





Domain Services

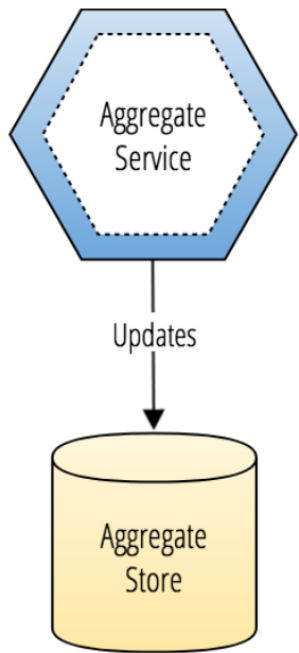
Domain services are microservices that own the *system of record* for a portion of the application's domain.



- Manage the storage of domain data that it owns.
- Produce the API contract for the domain data that it owns.
- Produce events when the state of any domain data changes.
- Maintain relationship integrity to domain data owned by other services.

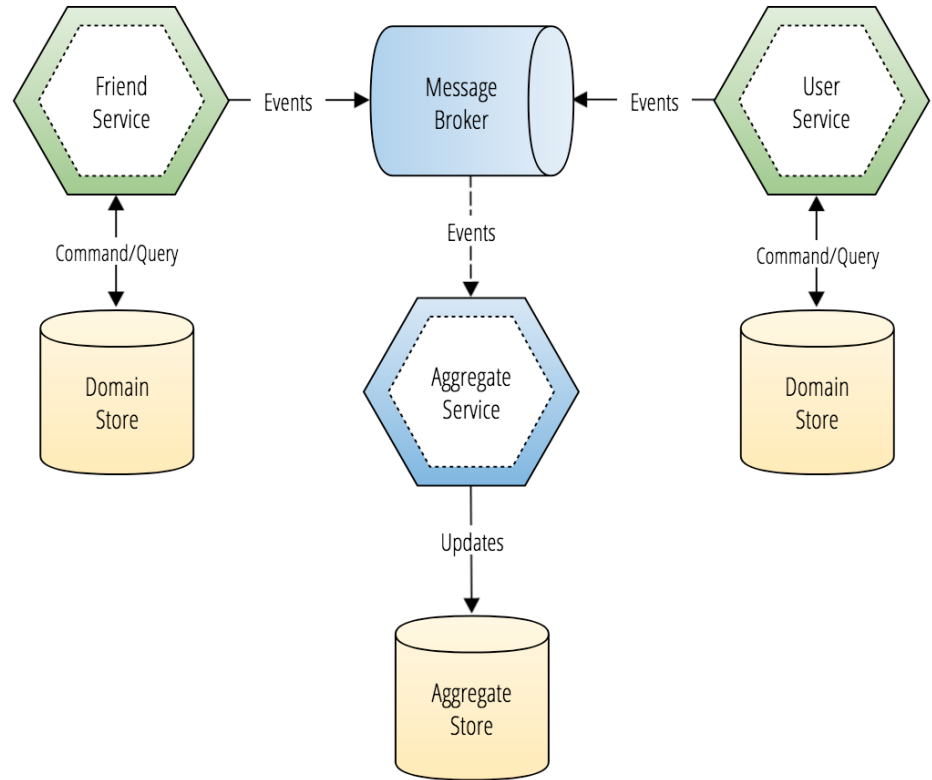
Aggregate Services

Aggregate services are microservices that replicate eventually consistent views of domain data owned by separate *domain services*.



- Subscribe to domain events emitted by separate domain services.
- Maintain an ordered immutable event log for events it receives.
- Create connected query projections of distributed domain data.
- Provide performant read-access to complex views of domain data.

Open Source Example



Thanks!

Q/A

<https://github.com/kbastani/event-sourcing-microservices-basics>

<https://github.com/ddd-by-examples/all-things-cqrs>

<https://github.com/ddd-by-examples/event-source-cqrs-sample>

