Microservices & Apache Kafka®

Part 1 – The Data Dichotomy: Rethinking the way we treat data and services

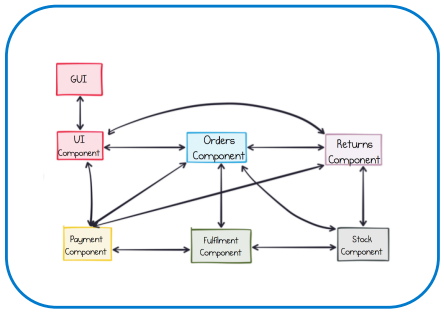
Series Schedule

- **Session 1: The Data Dichotomy: Rethinking the way we treat data and services**
- Session 2: Building Event-Driven Services with Apache Kafka
- Session 3: Putting the ‘Micro’ into Microservices with Stateful Stream Processing

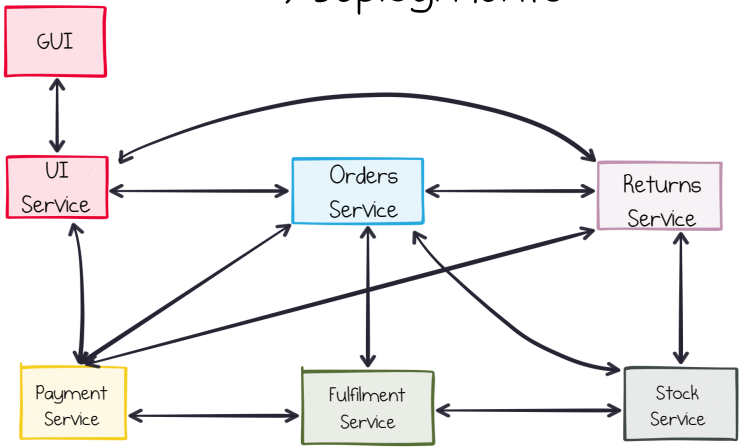
What are microservices really about?

Splitting the Monolith?

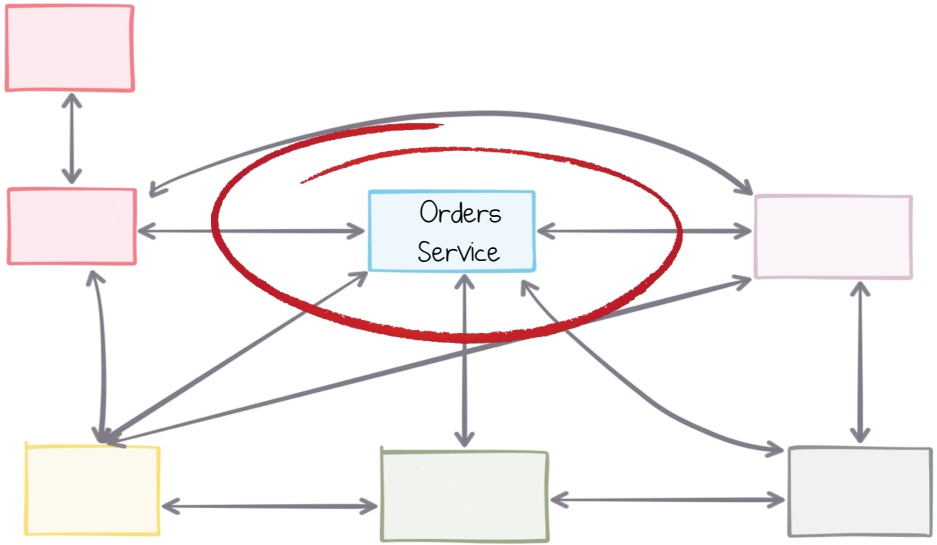
Single Process
/Code base
/Deployment

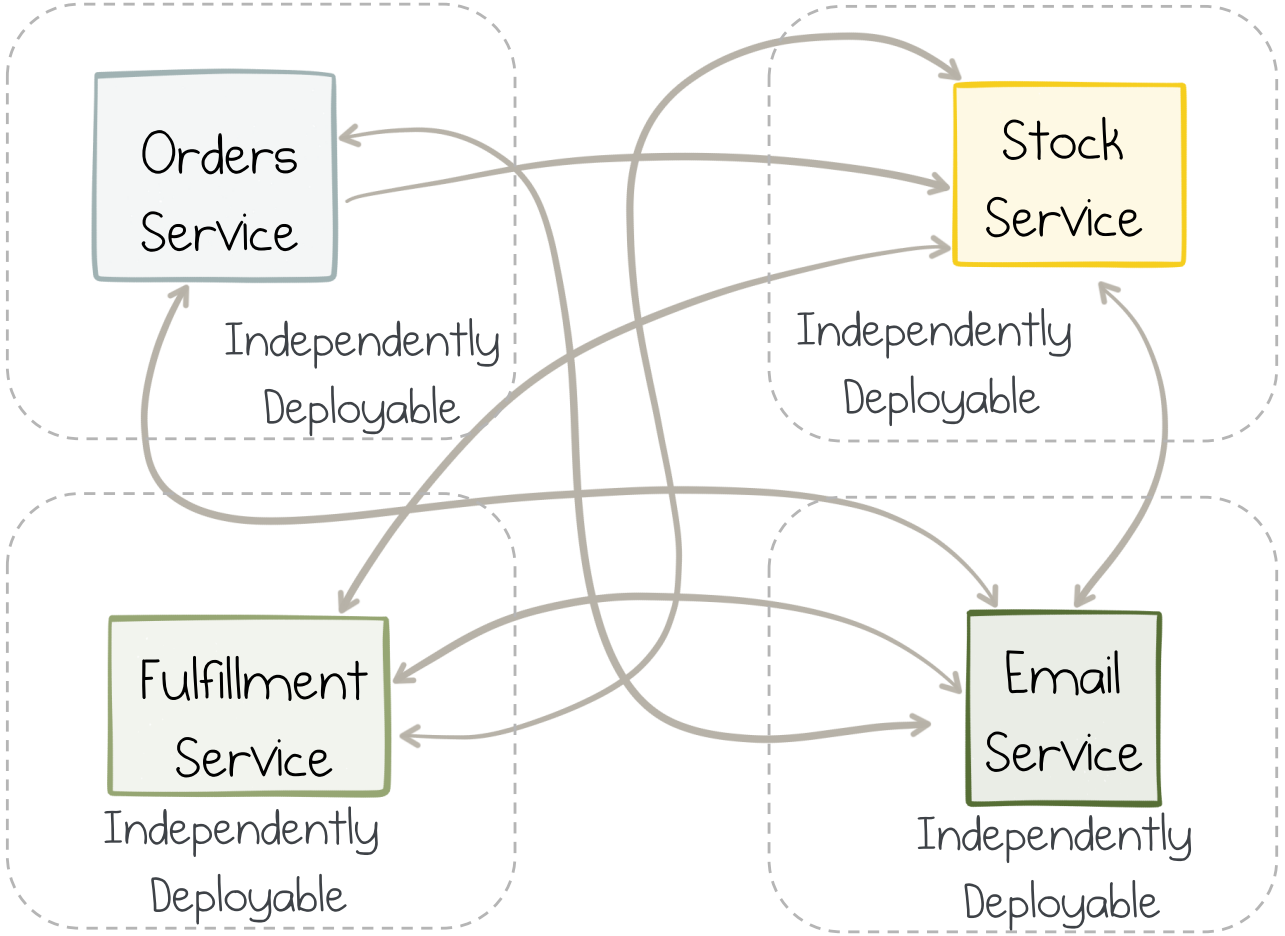


Many Processes
/Code bases
/Deployments



Autonomy?



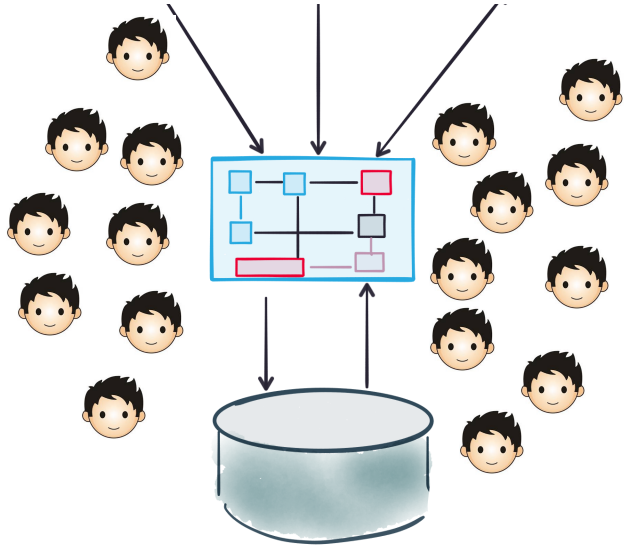

Orders Service
Independently Deployable
Stock Service
Independently Deployable
Fulfillment Service
Independently Deployable
Email Service
Independently Deployable

Independence is where services get their value

Allows Scaling

but not *just* in terms of data/load/throughput...

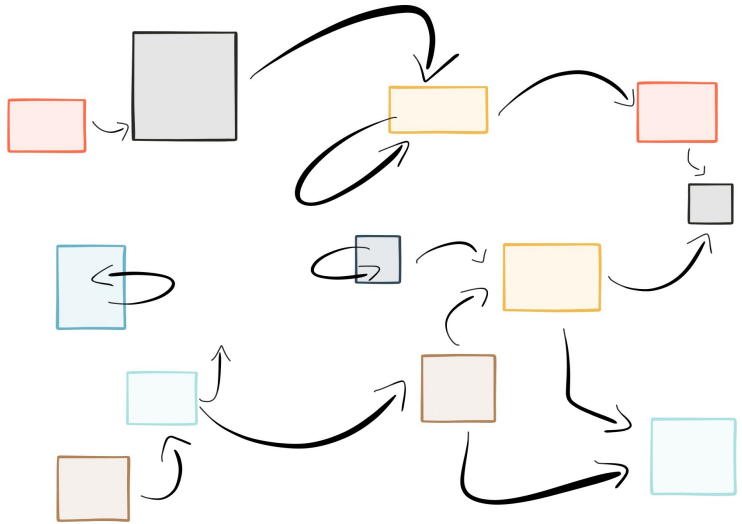
Scaling in terms of people



What happens when we grow?

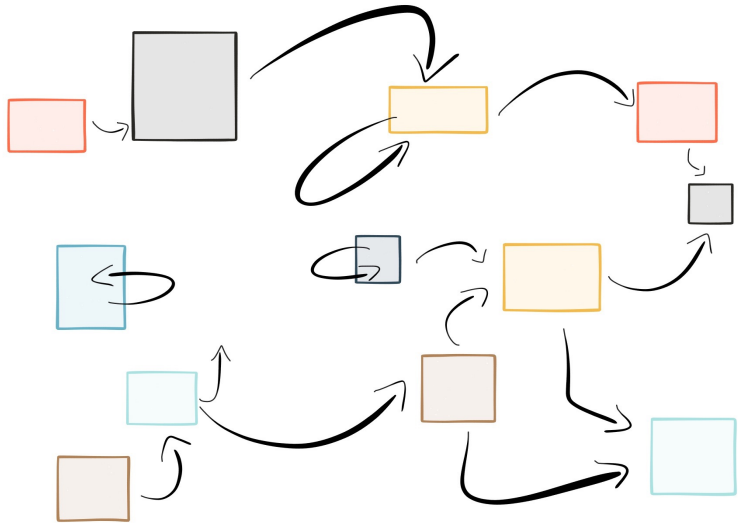
Companies are inevitably a collection of applications

They must work together to some degree

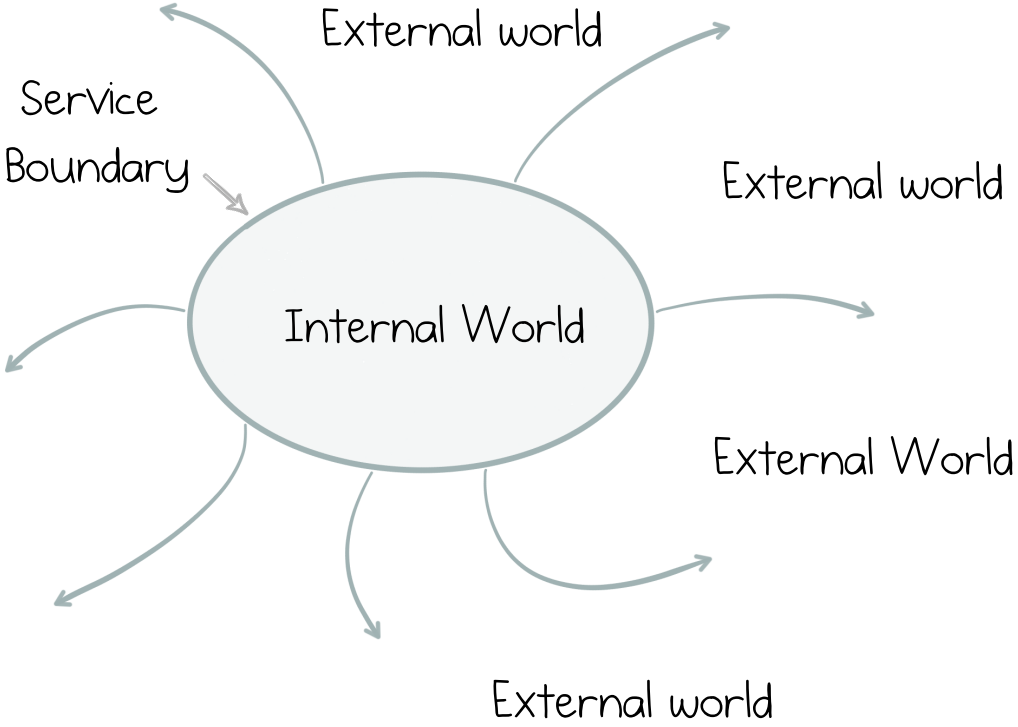


Interconnection is an afterthought

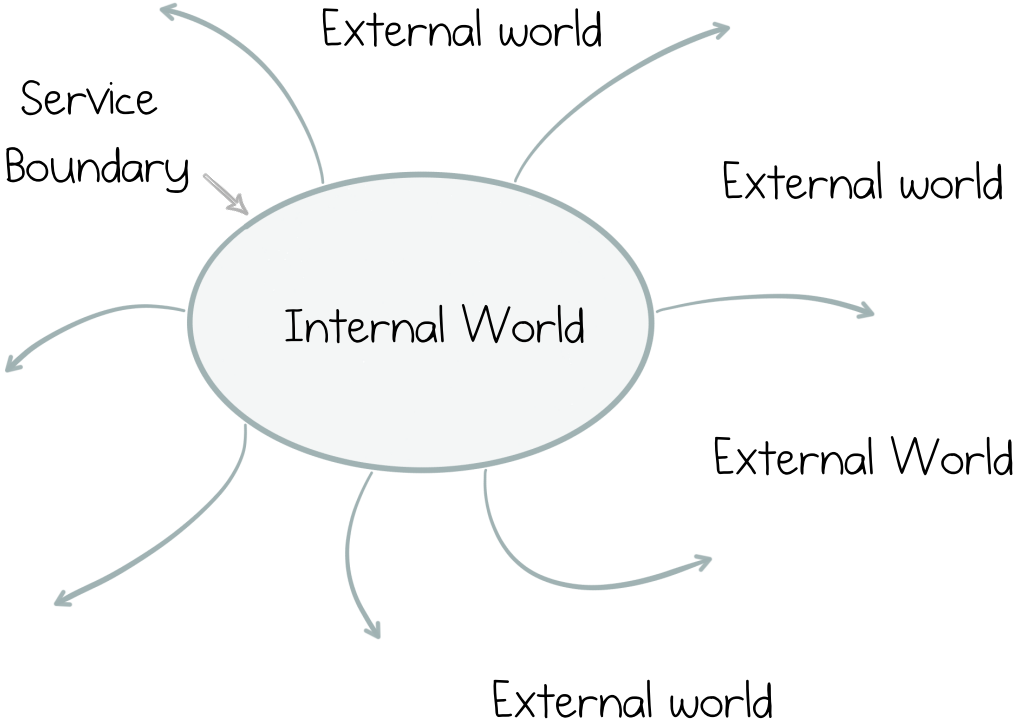
FTP / Enterprise Messaging etc



Services Force Us To Consider The External World

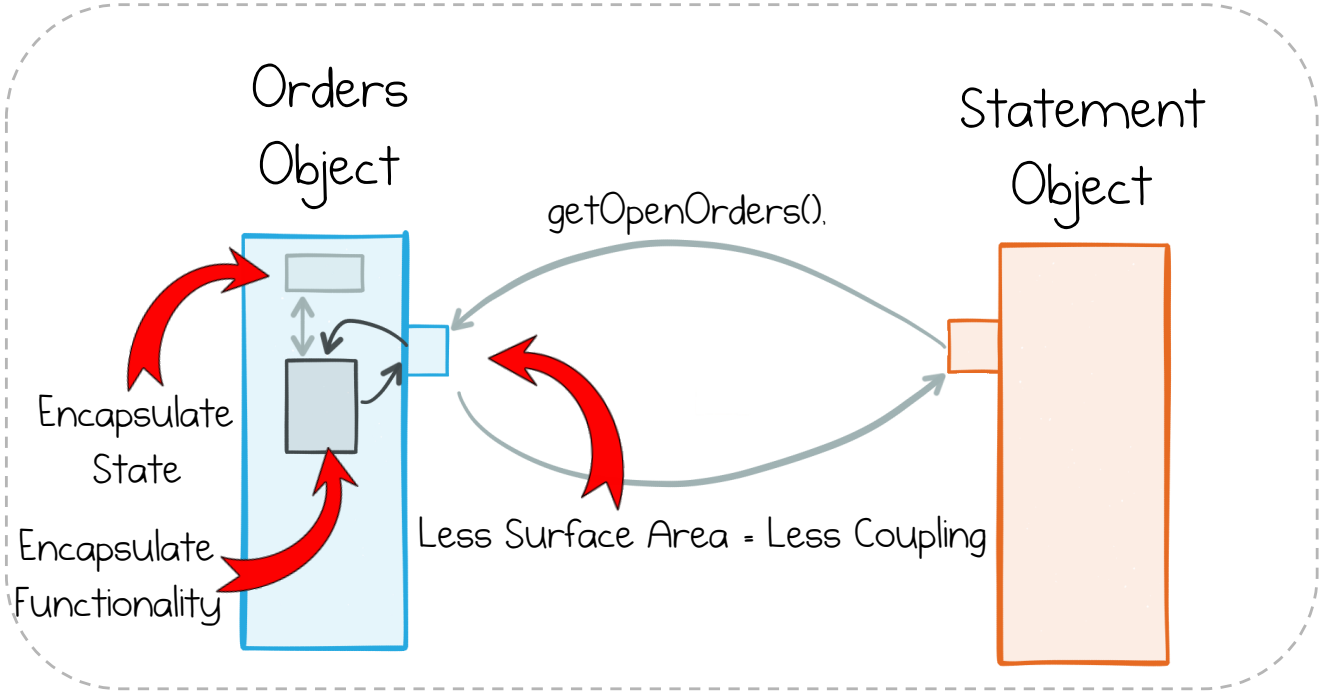


External World is something we should Design For

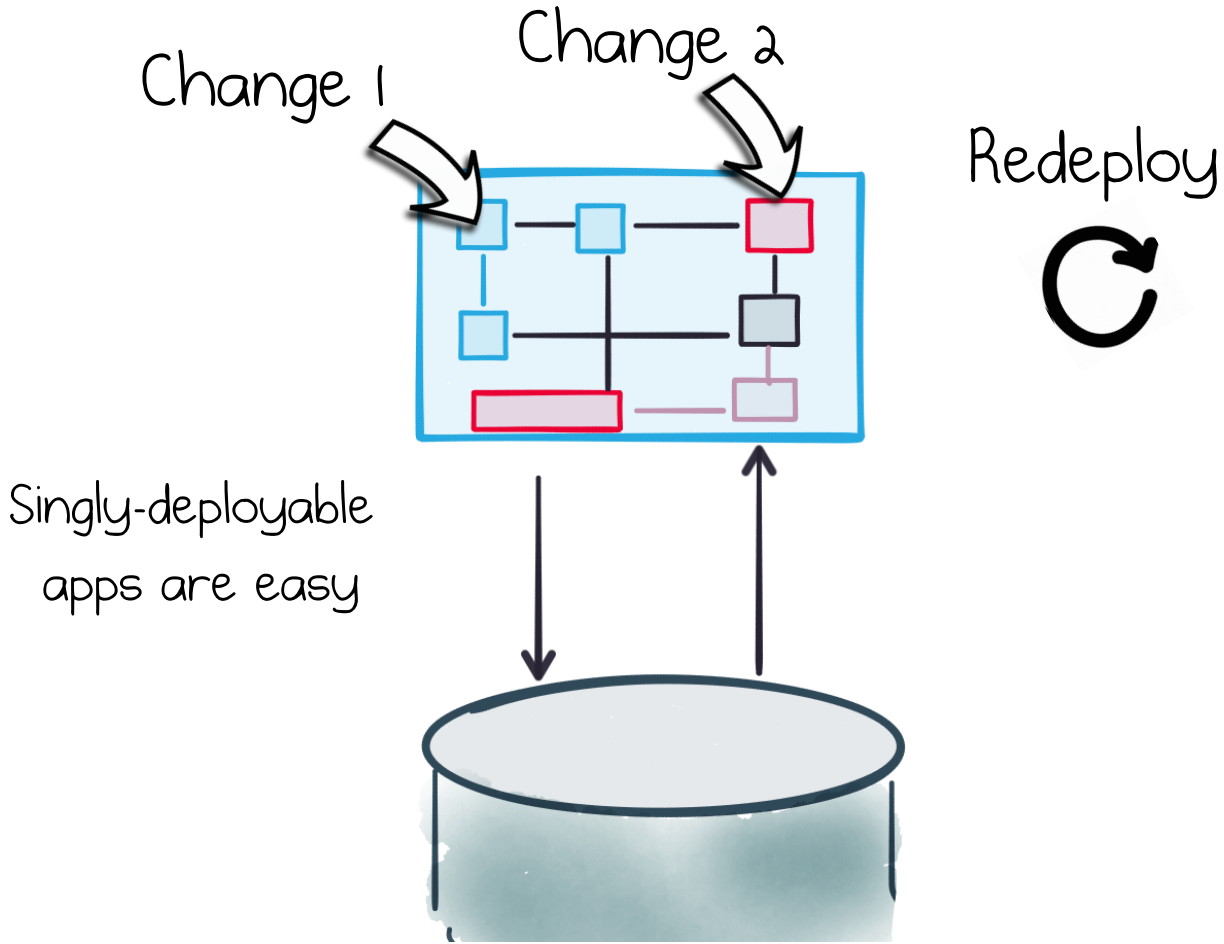


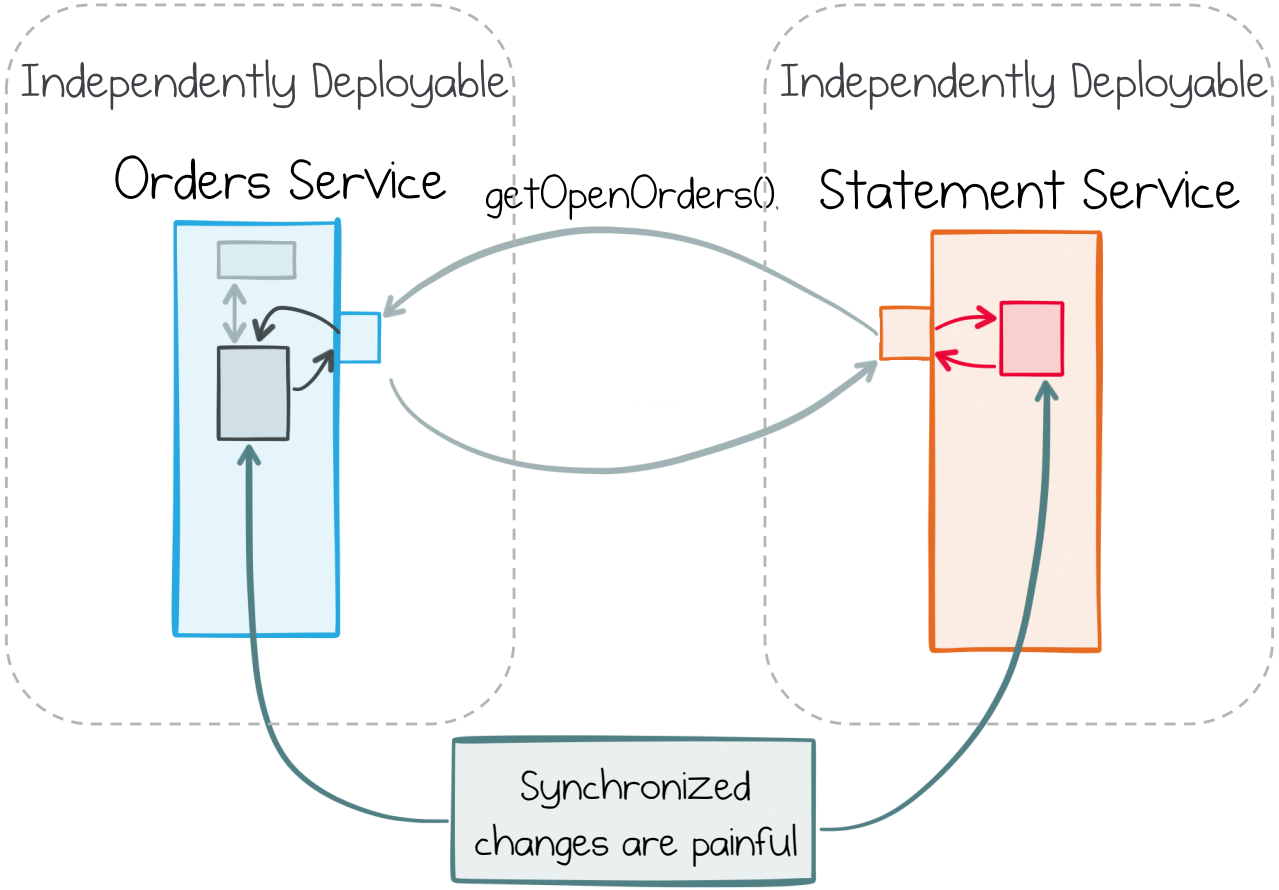
But this independence comes at a cost $$$

Consider Two Objects in one address space



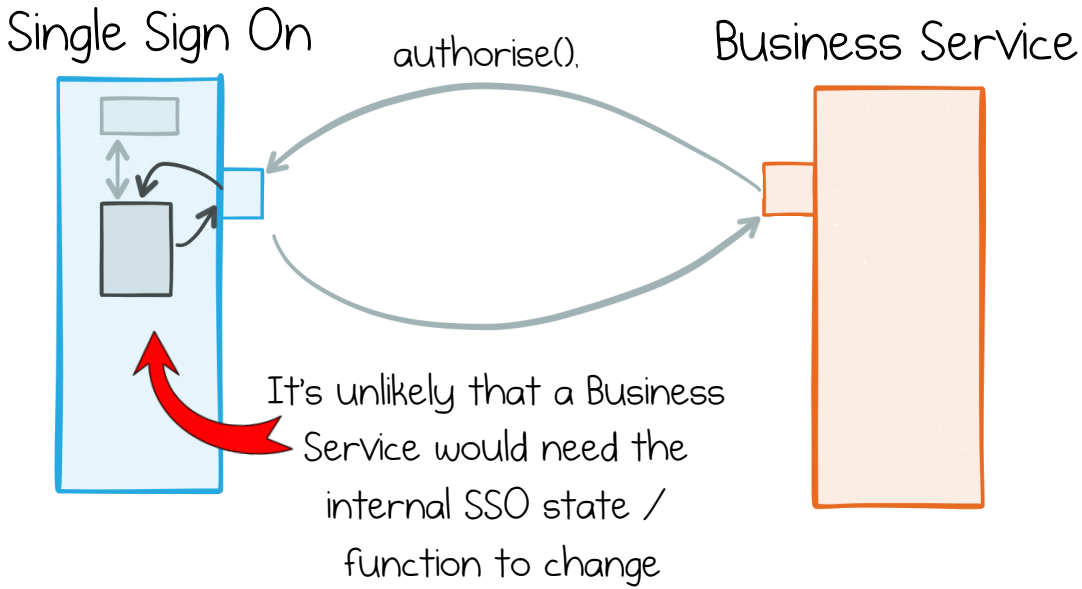
Encapsulation => Loose Coupling


Change 1
Change 2
Redeploy
Singly-deployable
apps are easy


Independently Deployable
Orders Service
getOpenOrders()
Independently Deployable
Statement Service
Synchronized
changes are painful

Services work best where requirements are isolated in a single bounded context

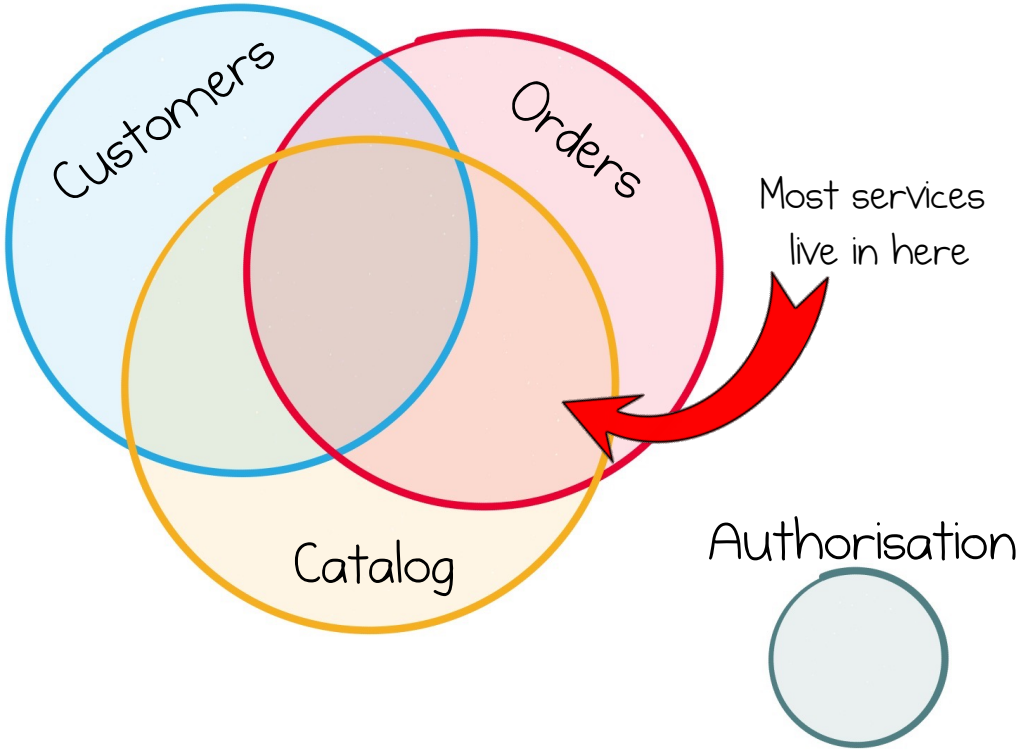
Single Sign On



SSO has a tightly bounded context

But business services are different

Most business services share the same core stream of facts.



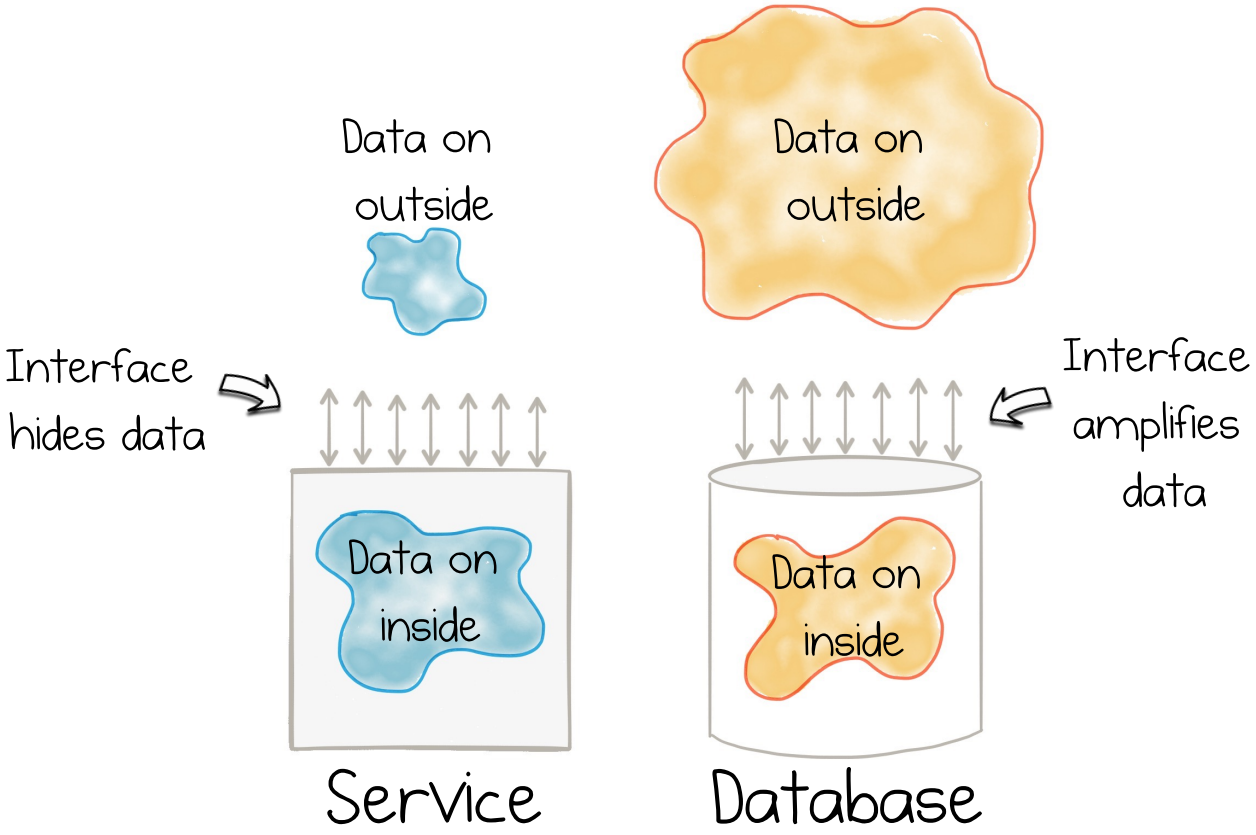
The futures of business services are far more tightly intertwined.

We need encapsulation to hide internal state. Be loosely coupled.

But we need the freedom to slice & dice shared data like any other dataset

But data systems have little to do with encapsulation

Databases amplify the data they hold



THE DATA DICHOTOMY

Data systems are about exposing data.

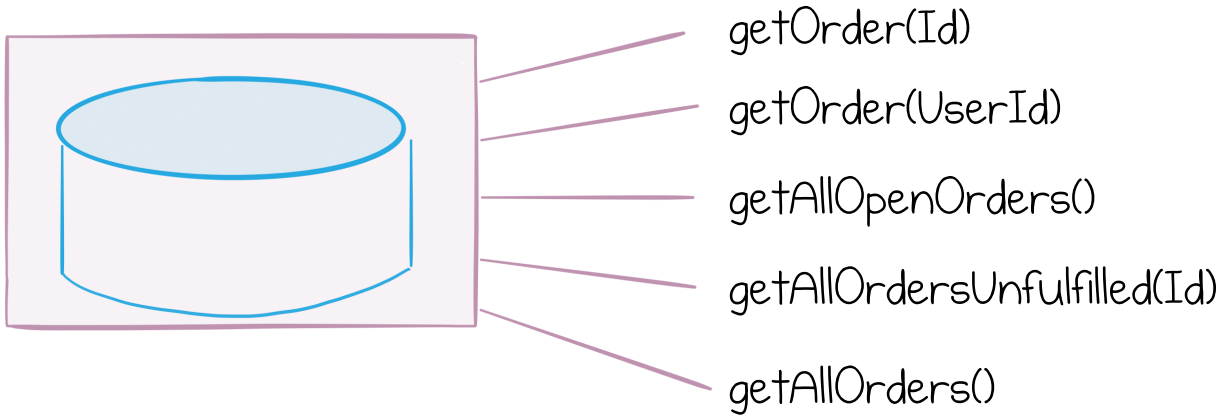
Services are about hiding it.

This affects services in one of two ways

Either (1) we constantly add to the interface, as datasets grow

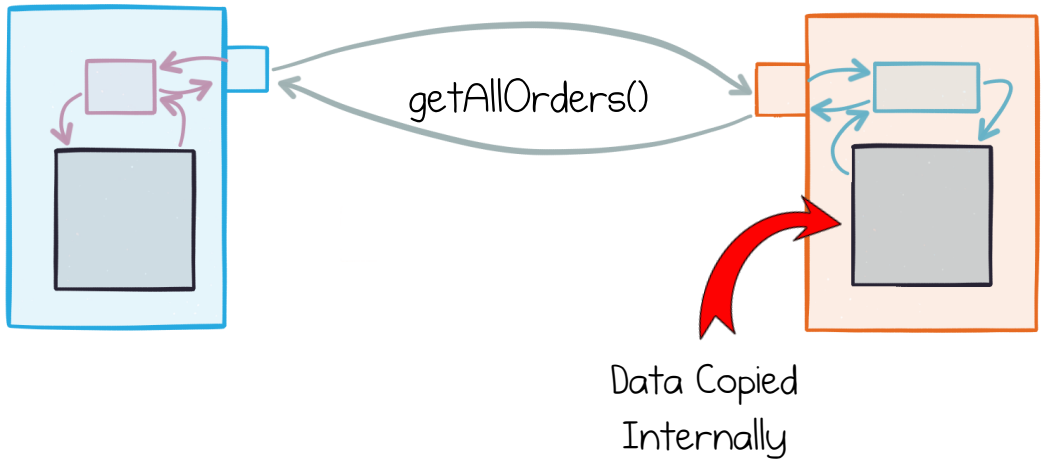
getOpenOrders(
fulfilled=false,
deliveryLocation=CA,
orderValue=100,
operator=GreatherThan)

(1) Services can end up looking like kookie home-grown databases



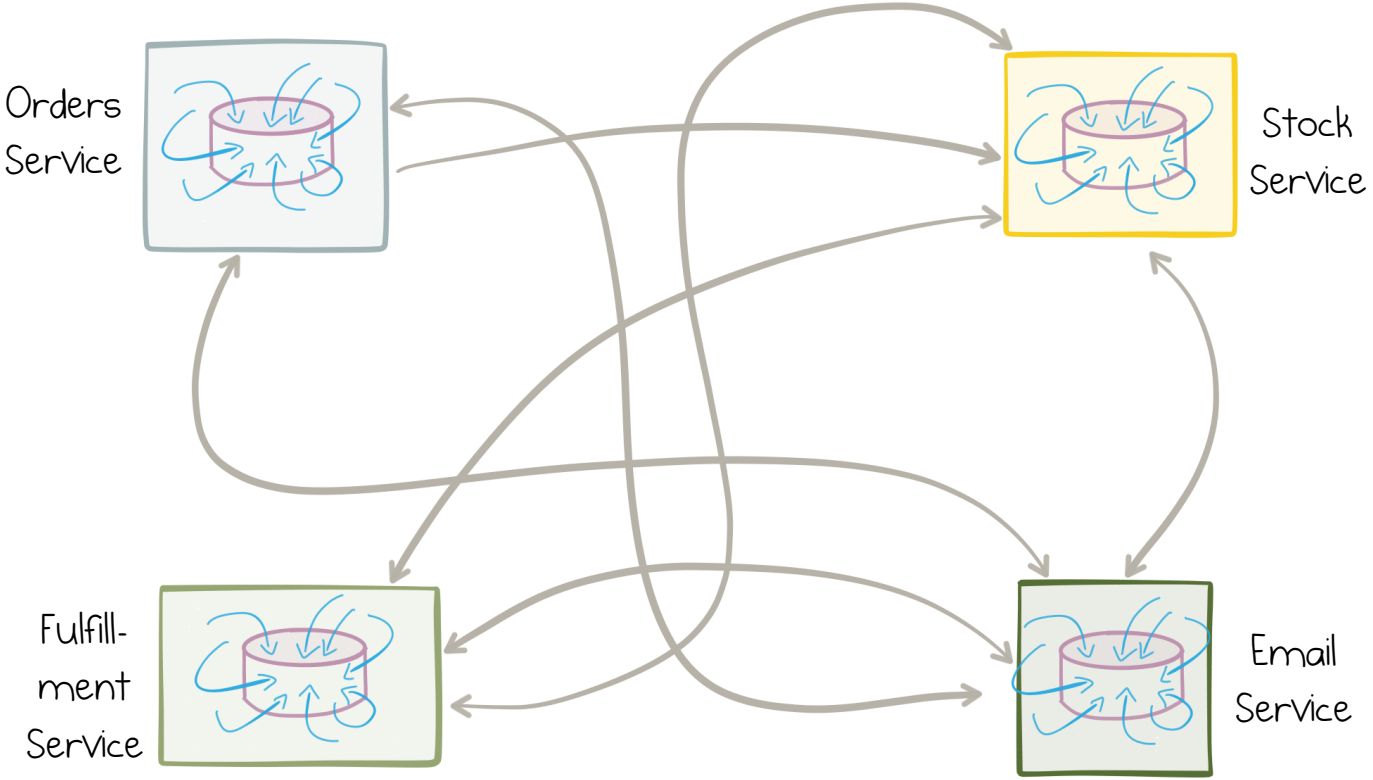
...and DATA amplifies this service boundary problem

(2) Give up and move whole datasets en masse



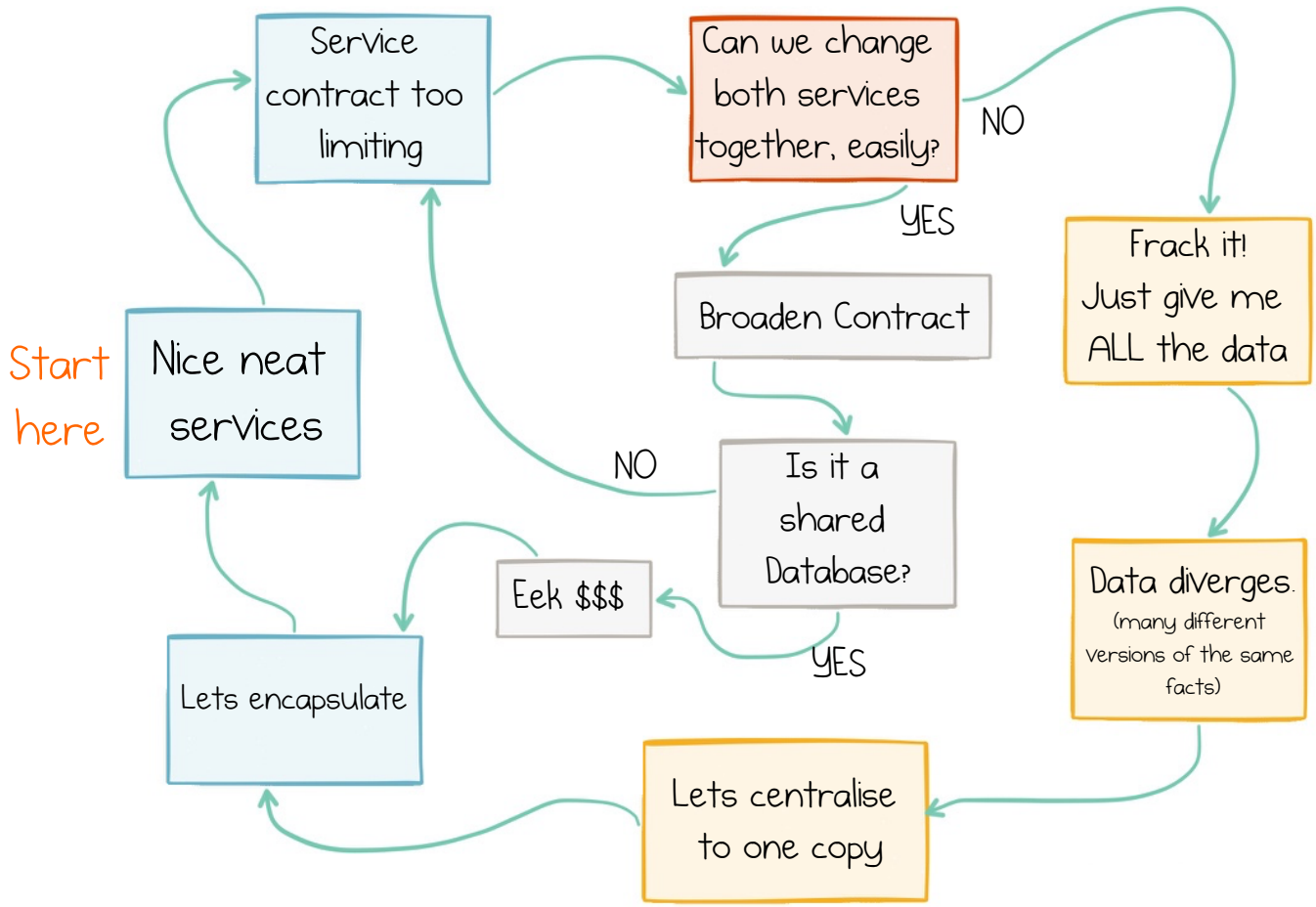
This leads to a different problem

Data diverges over time

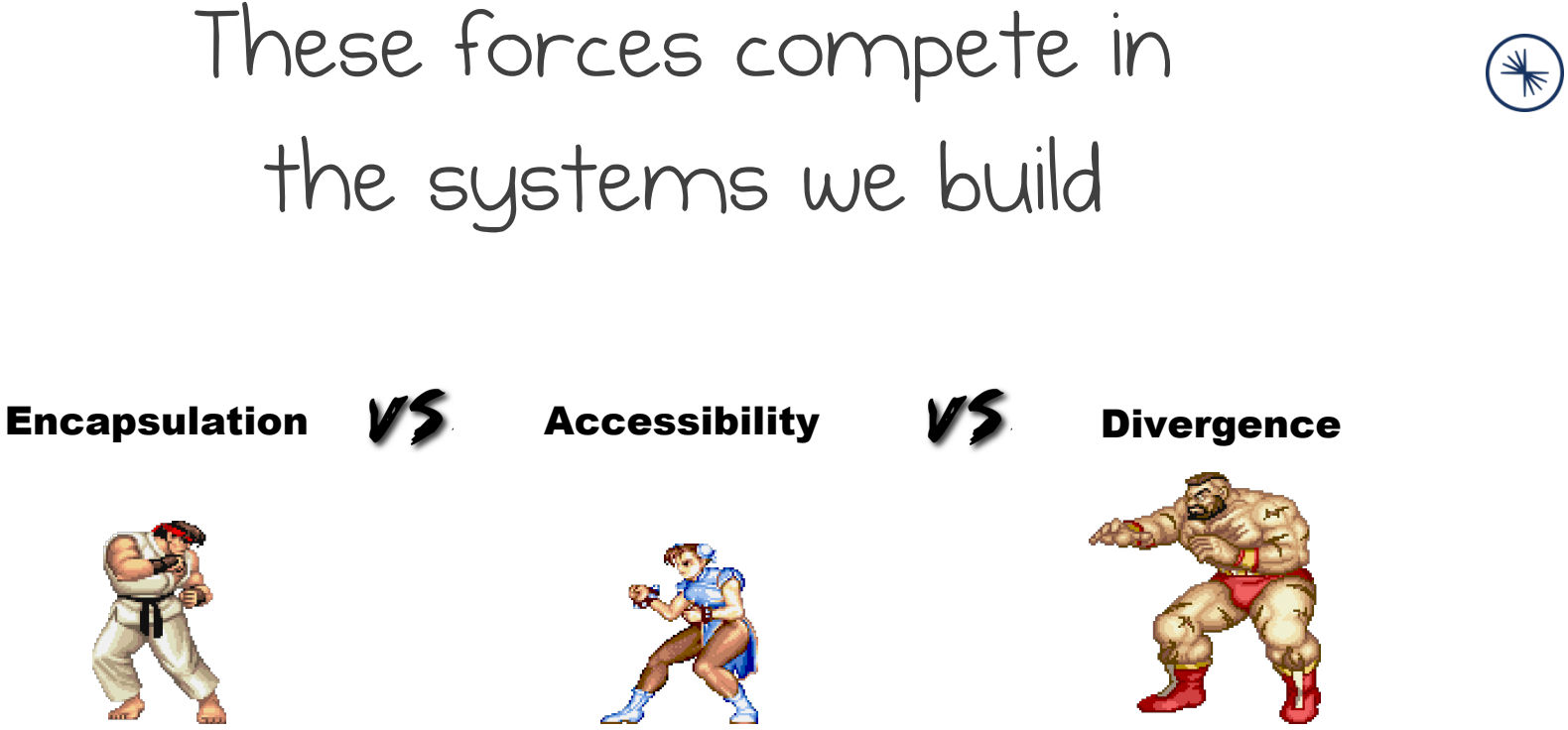


The more
mutable copies,
the more data will
diverge over time

Cycle of inadequacy:

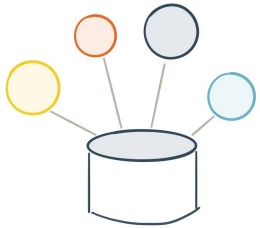


These forces compete in the systems we build

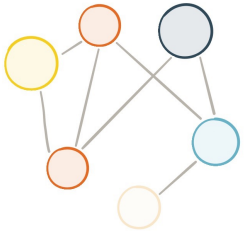


No perfect solution

Shared database



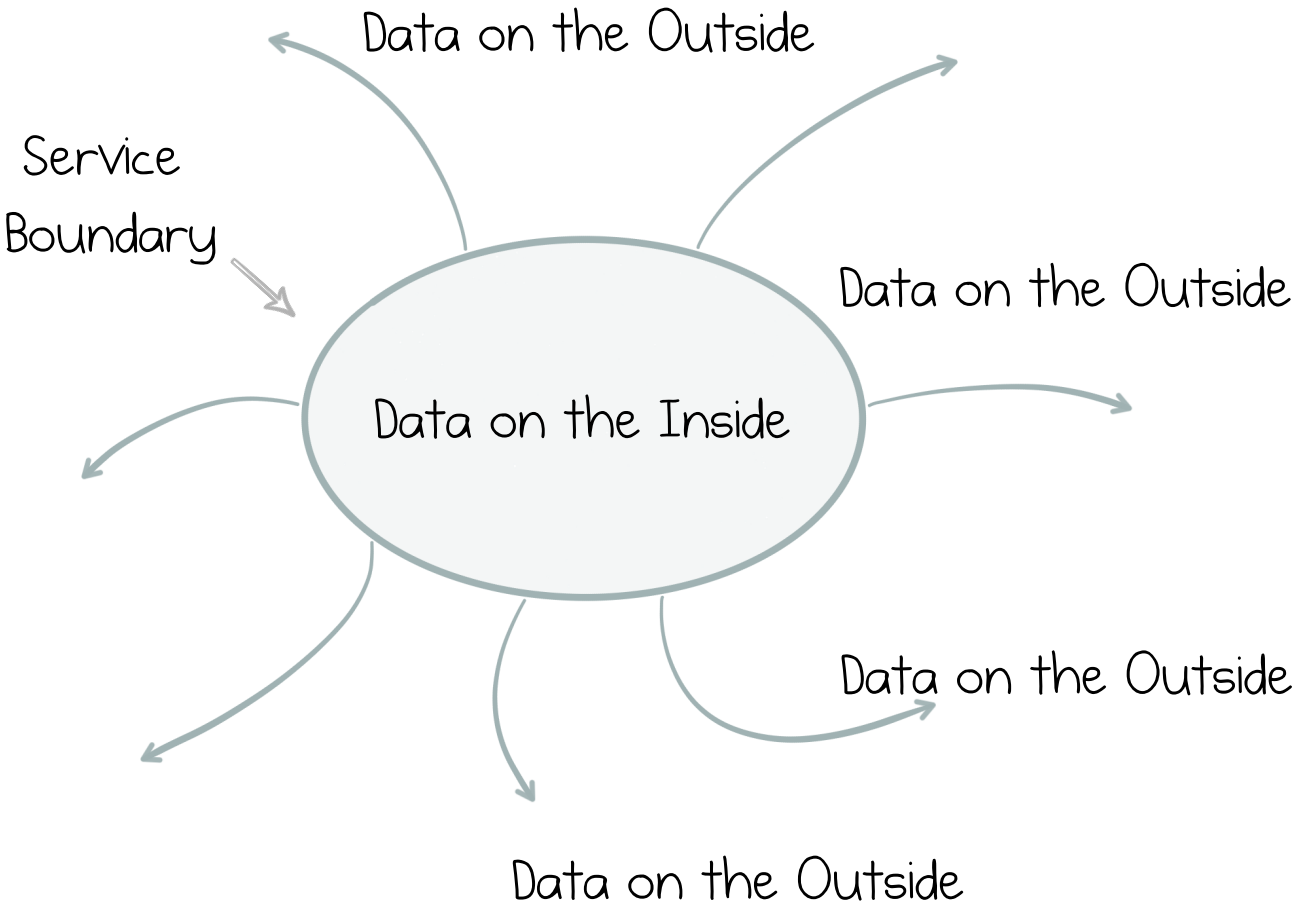
Service Interfaces



Better Accessibility

Better Independence

Is there a better way?


Data on the Outside
Service
Boundary
Data on the Outside
Data on the Inside
Data on the Outside
Data on the Outside

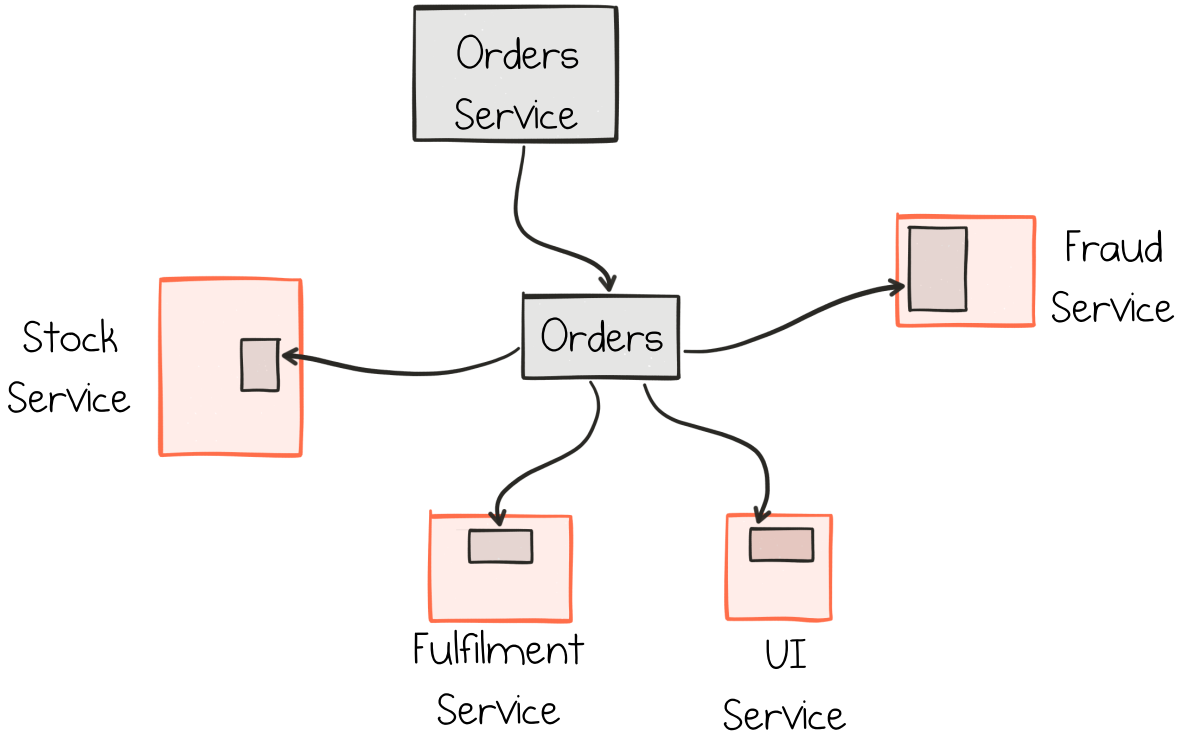
Make data-on-the-outside a 1st class citizen

Separate Reads & Writes with Immutable Streams

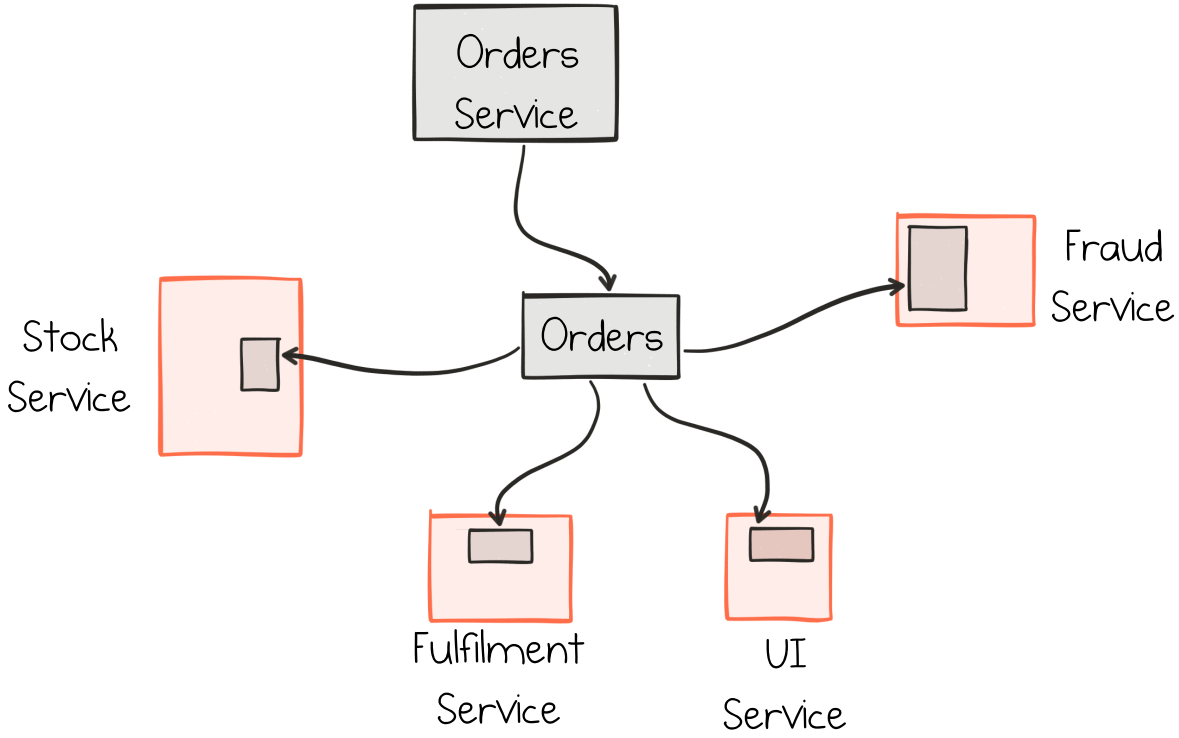
CQRS

- Writes => "Normalized" into each service
- Reads => "Denormalized" into Services
- One canonical set of streams

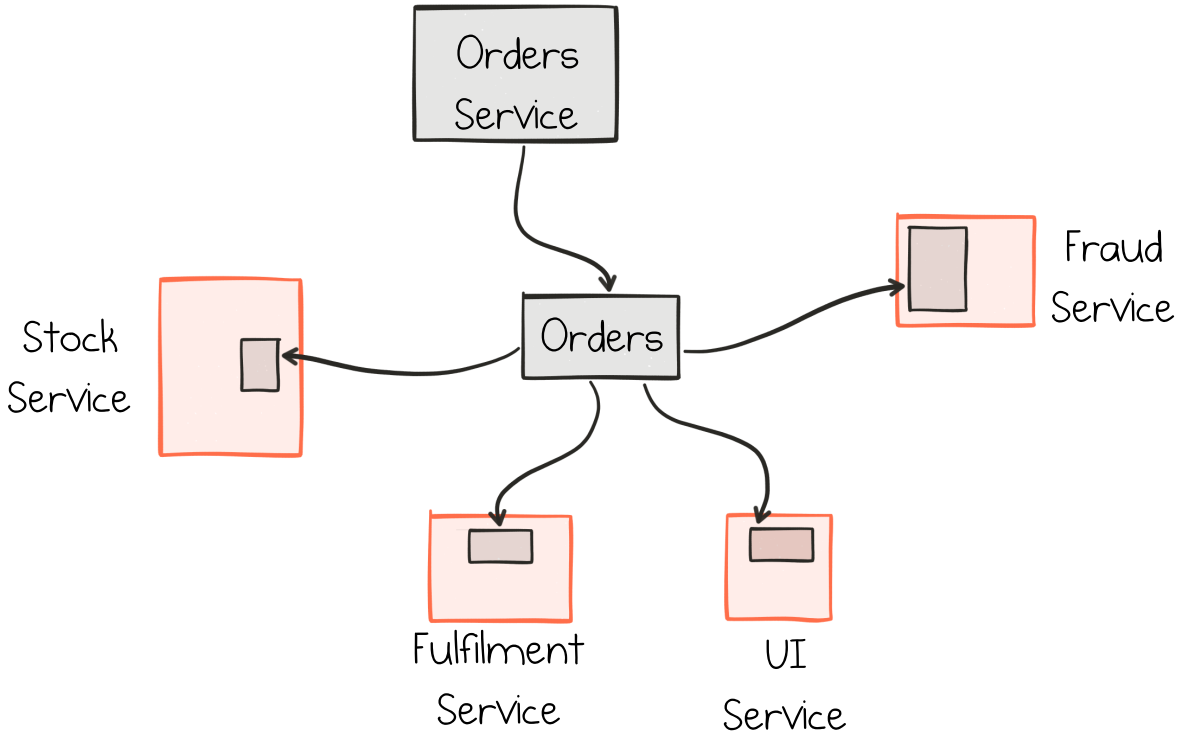
Writes enter via the Orders Service (which manages the workflow of orders)



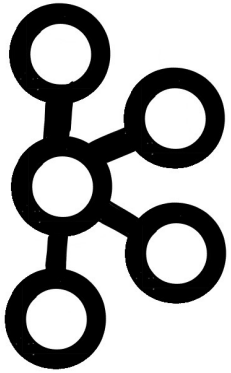
Readers use streams, materialized in each service



Important: data is not retained, it's just a view over those centralized streams

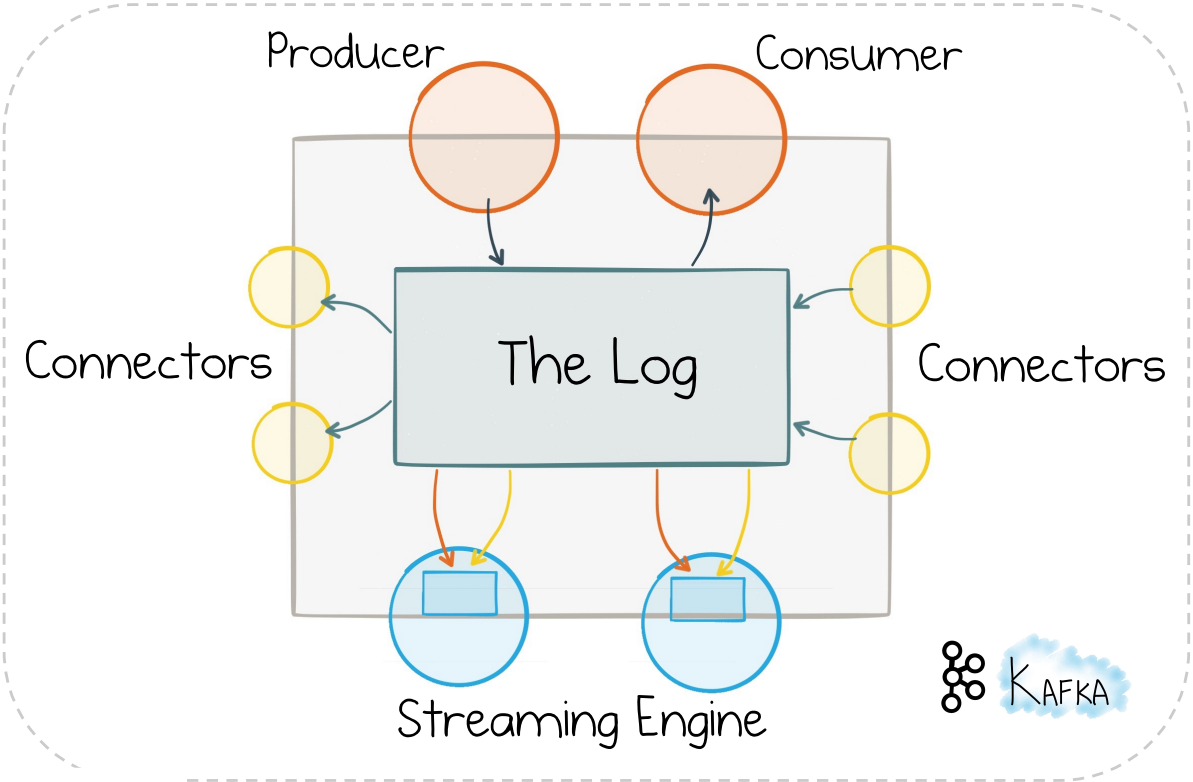


Use a Stream Processing tool-kit



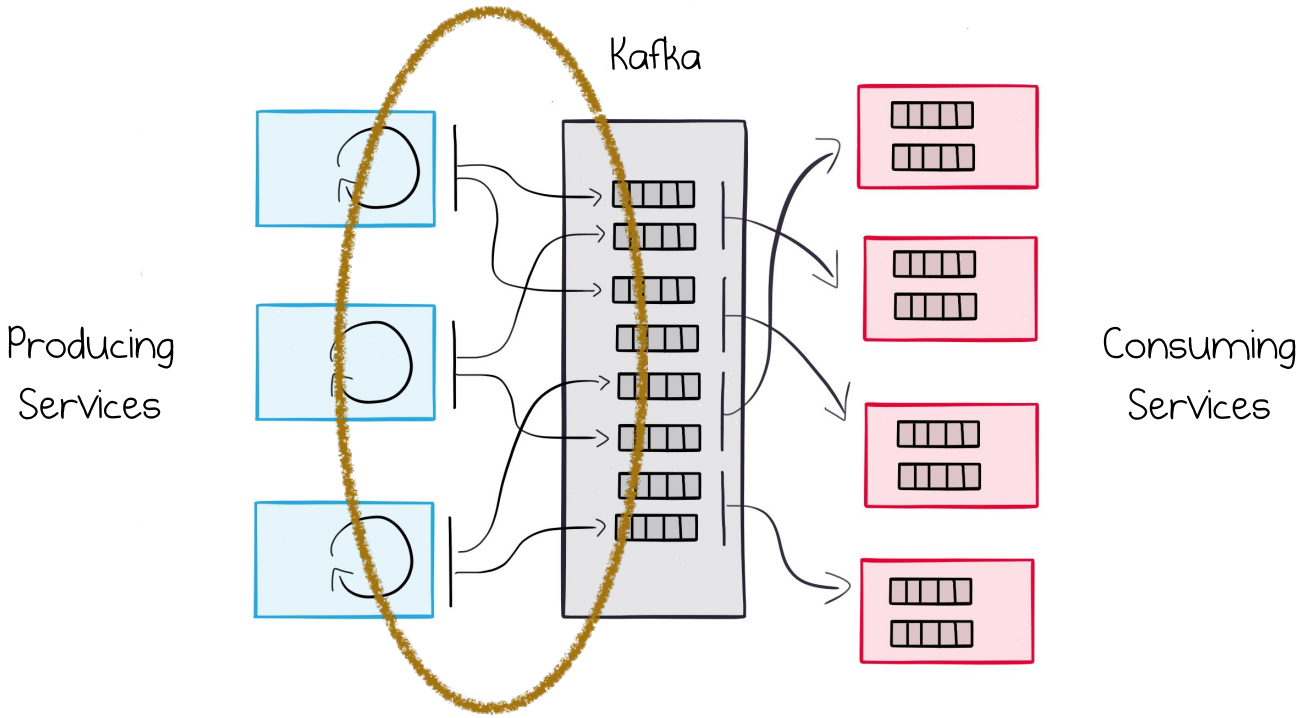
Kafka is a Streaming Platform

Kafka: a Streaming Platform

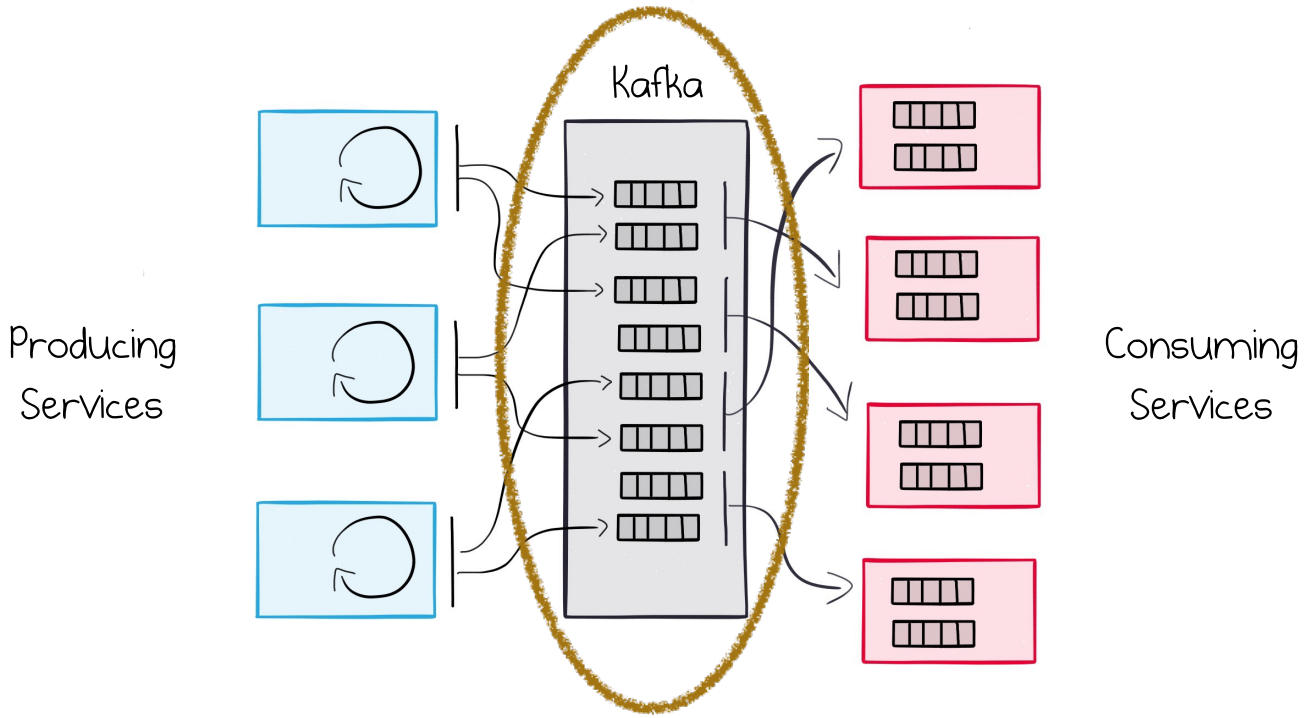


The Log

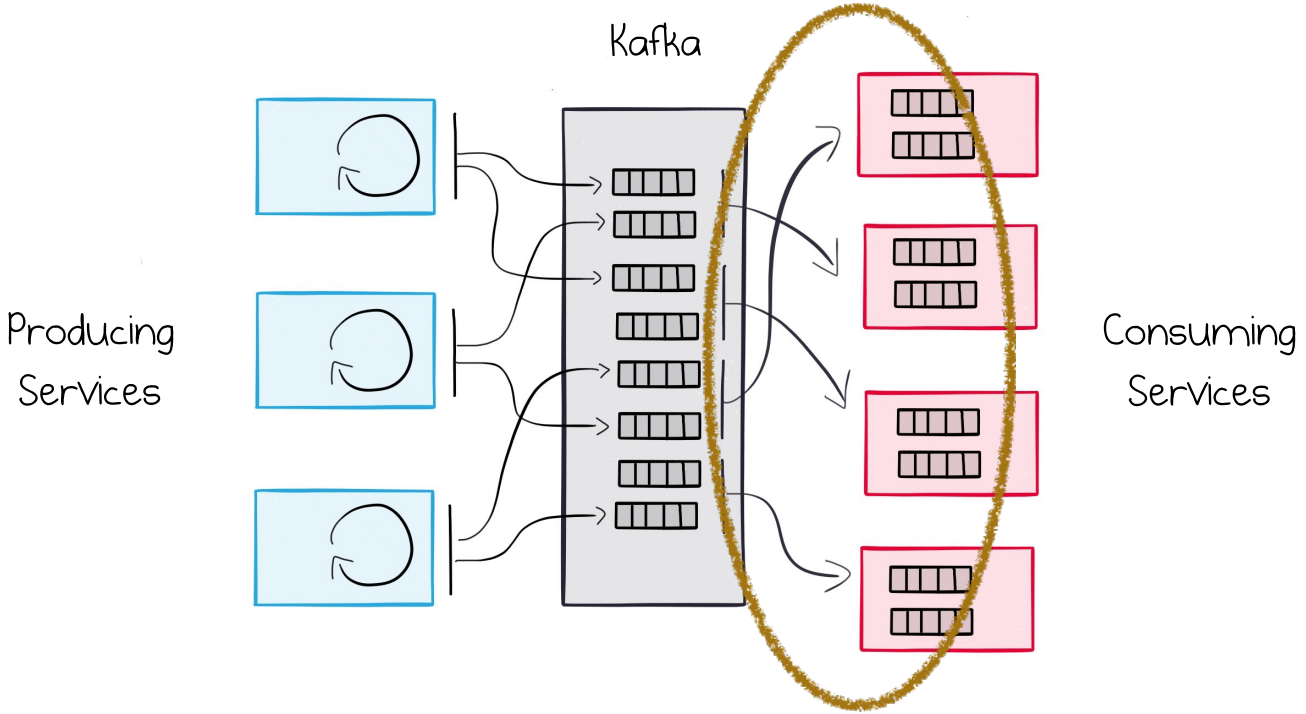
Shard on the way in



Each shard is a queue

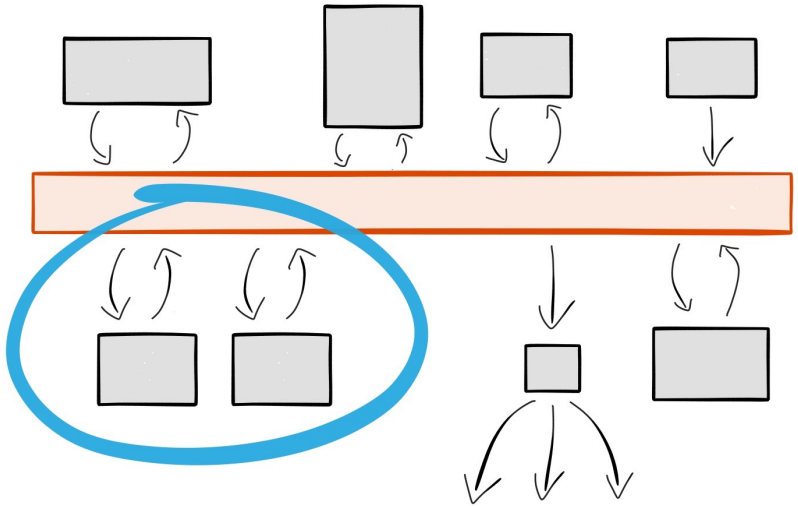


Consumers share load

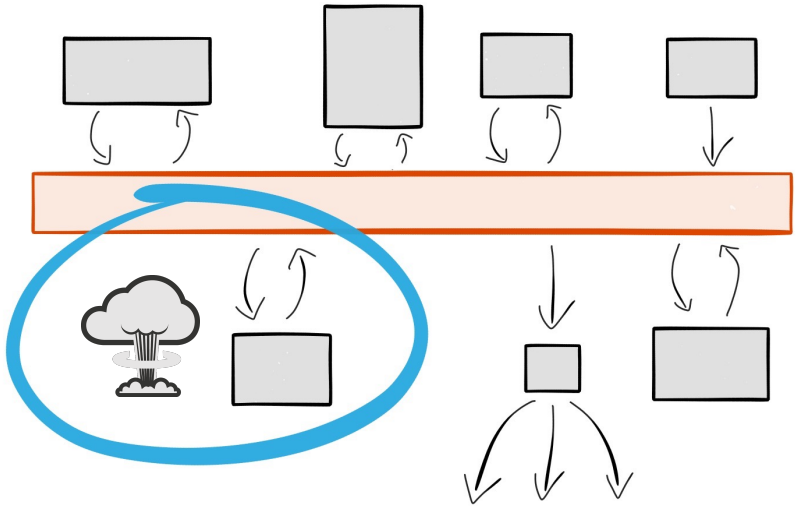


Scaling becomes a concern of the broker, not the service

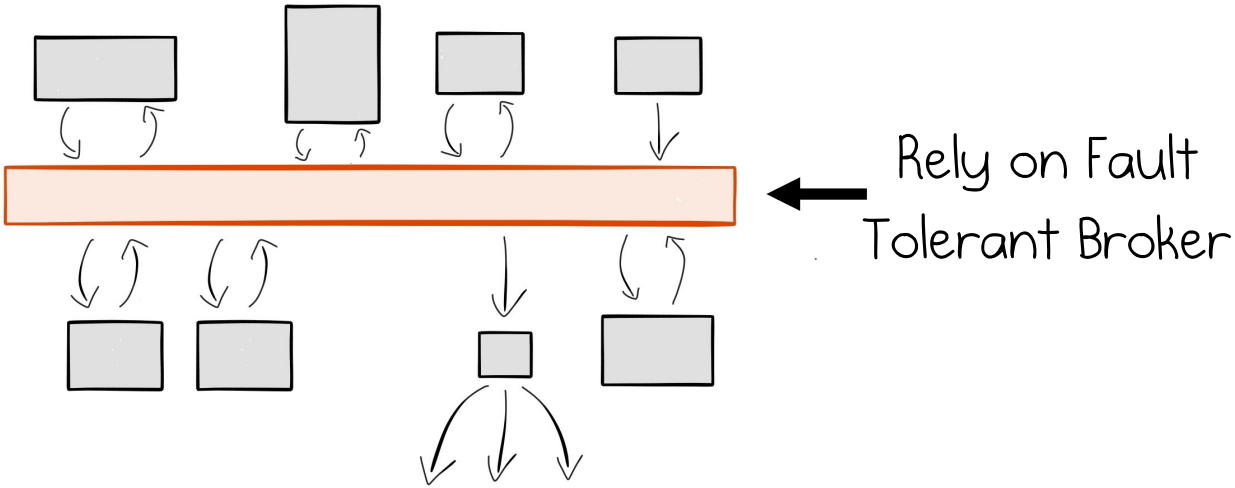
Load Balanced Services



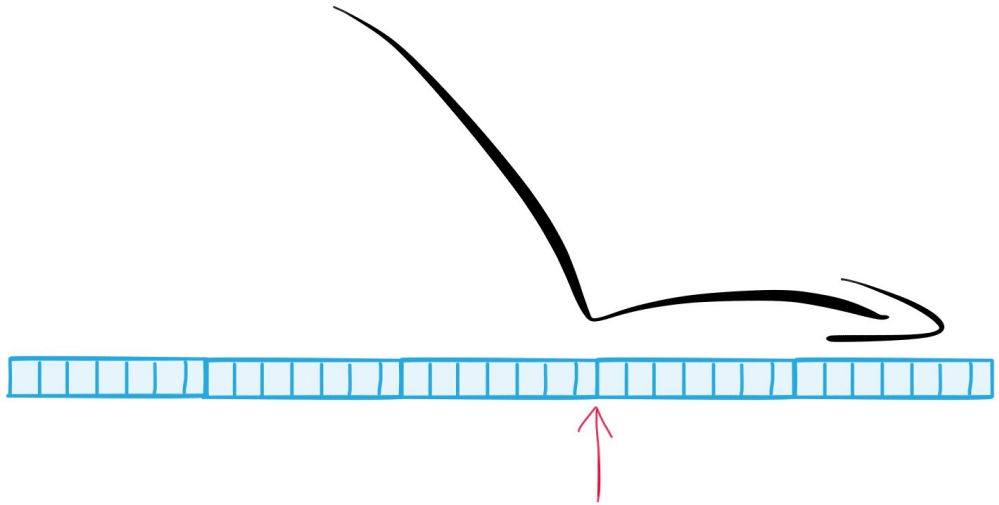
Fault Tolerant Services



Build 'Always On' Services

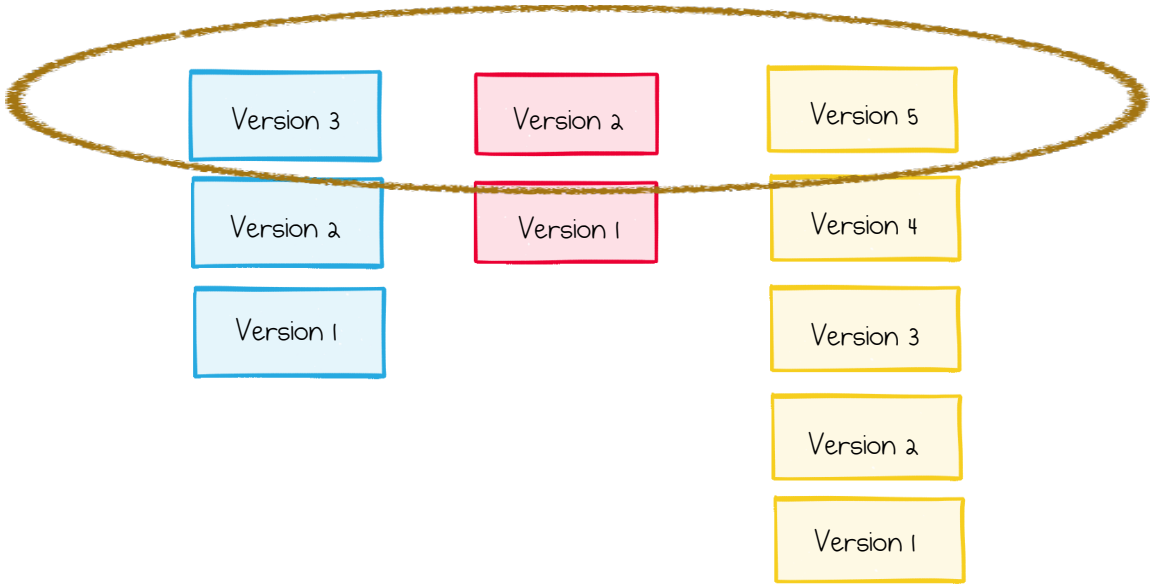


Reset to any point in the shared narrative



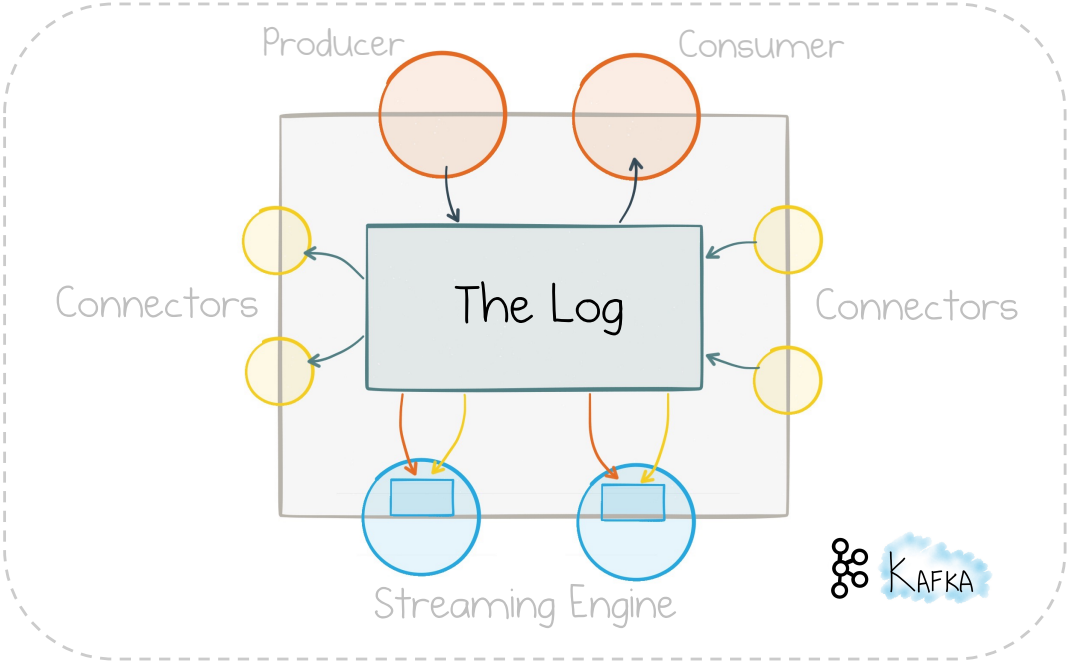
Compacted Log

(retains only latest version)



Service Backbone

Scalable, Fault Tolerant, Concurrent, Strongly Ordered, Retentive

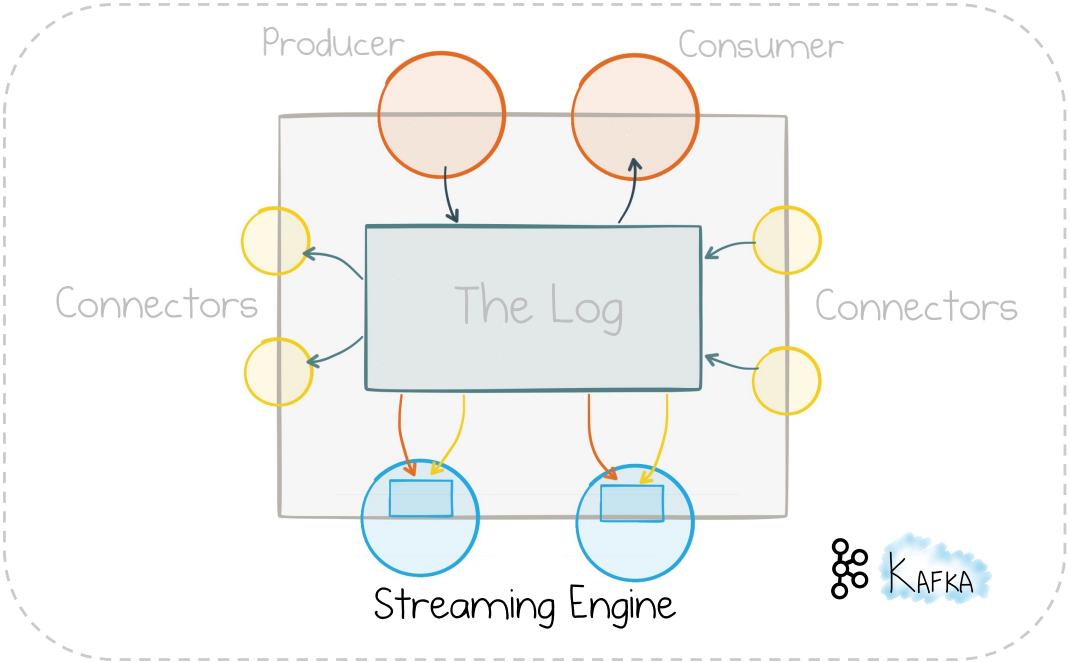


A place to keep the data-on-the-outside as a immutable narrative

Now add Stream Processing

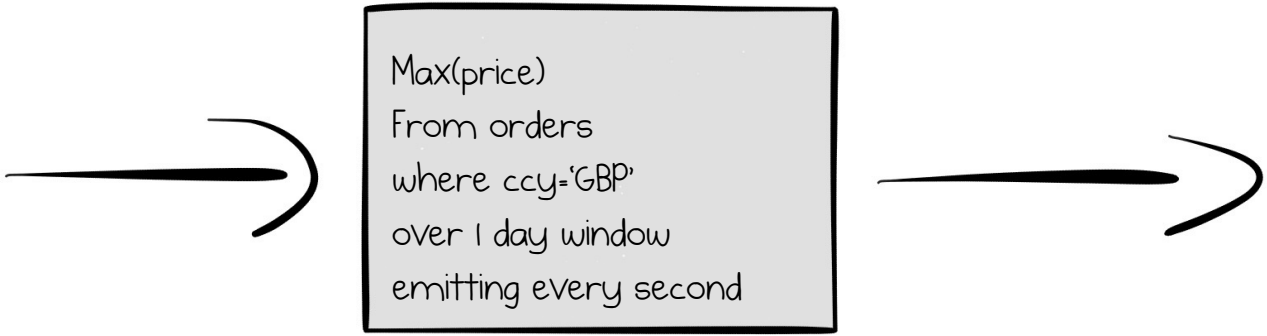
Kafka's Streaming API

A general, embeddable streaming engine

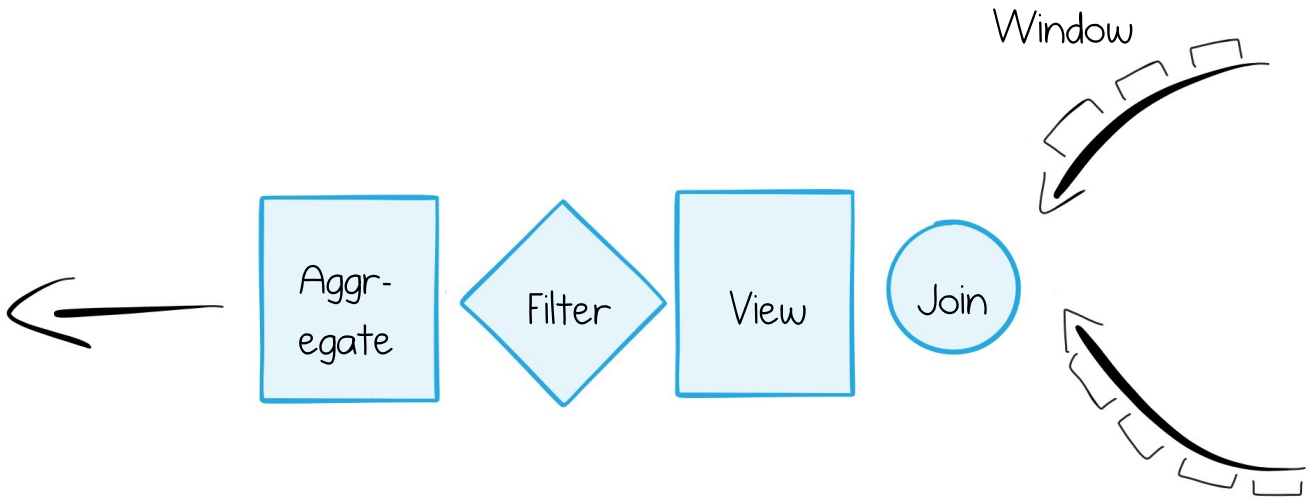


What is Stream Processing?

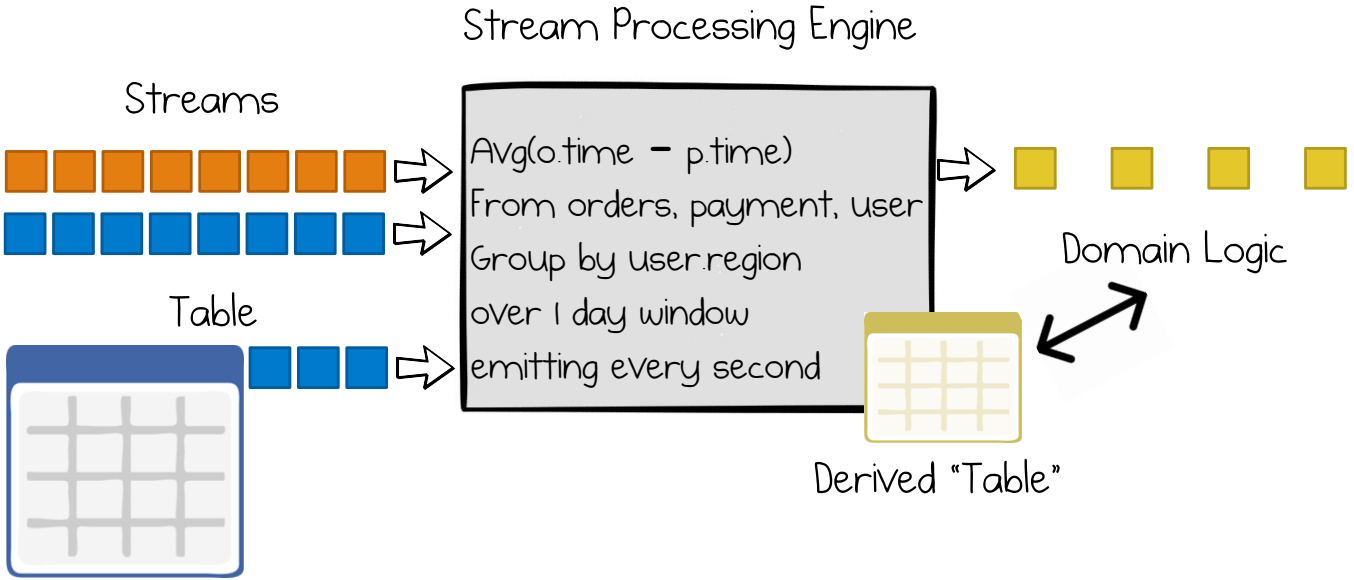
DB Engine designed to process streams



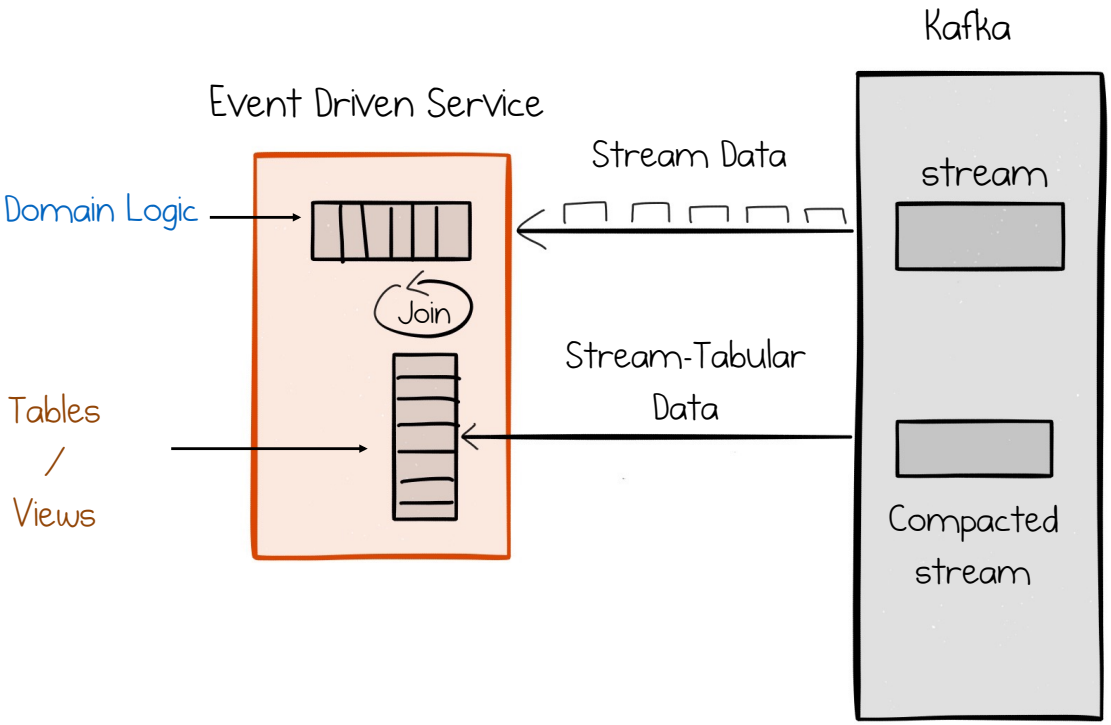
Features: similar to database query engine



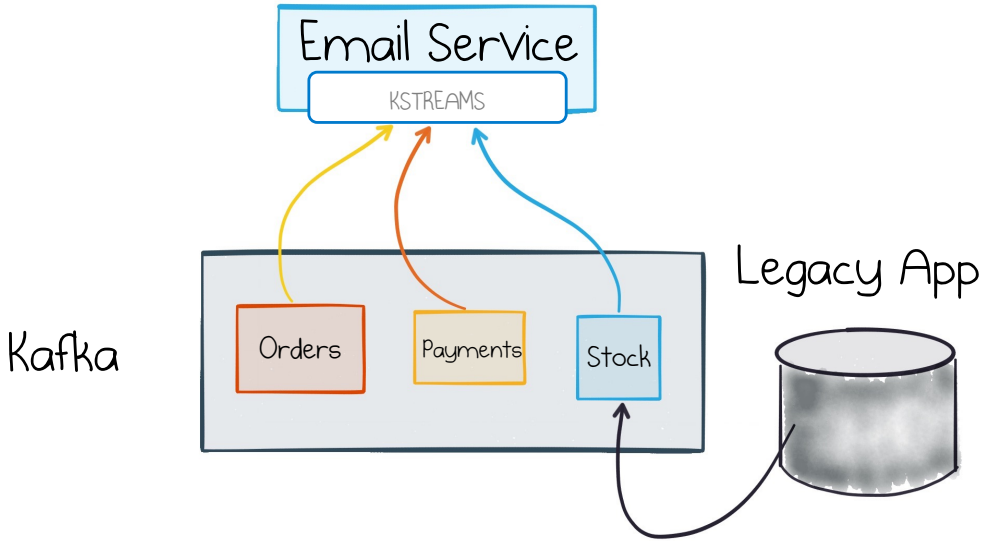
Stateful Stream Processing



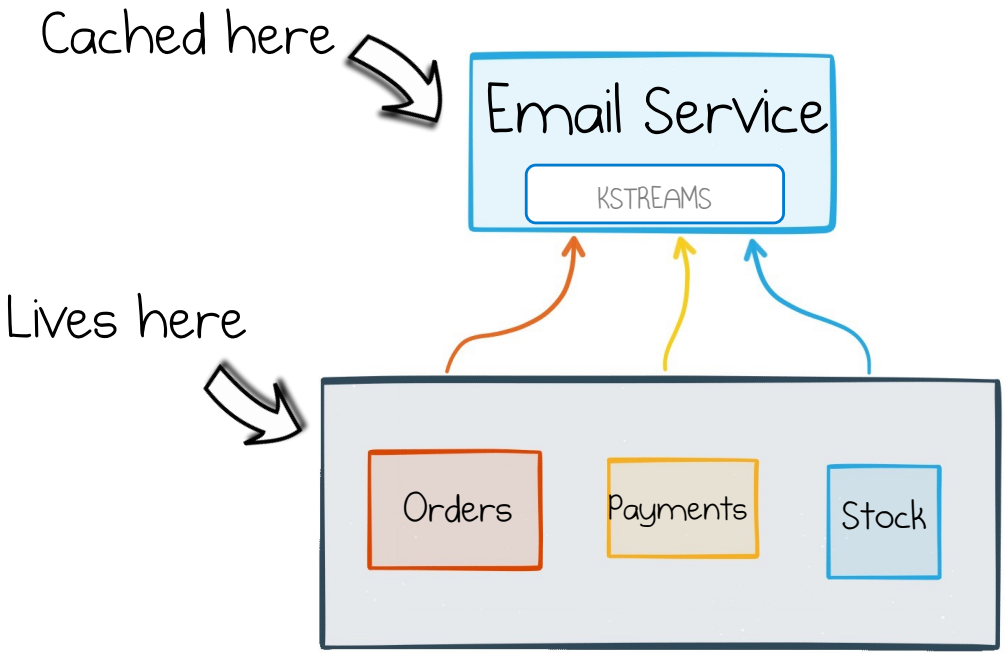
Stateful Stream Processing



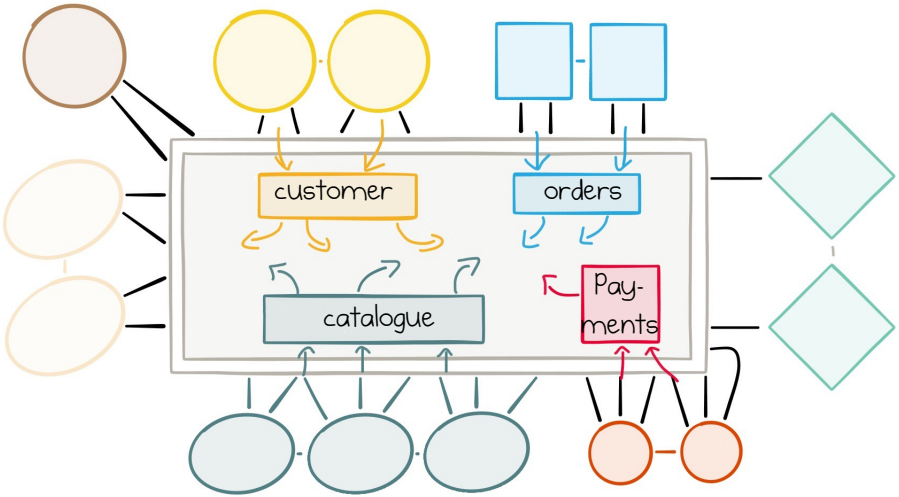
Join shared streams from many other services



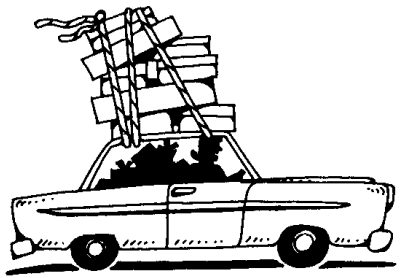
Shared State is only cached in the service, so there is no way to diverge



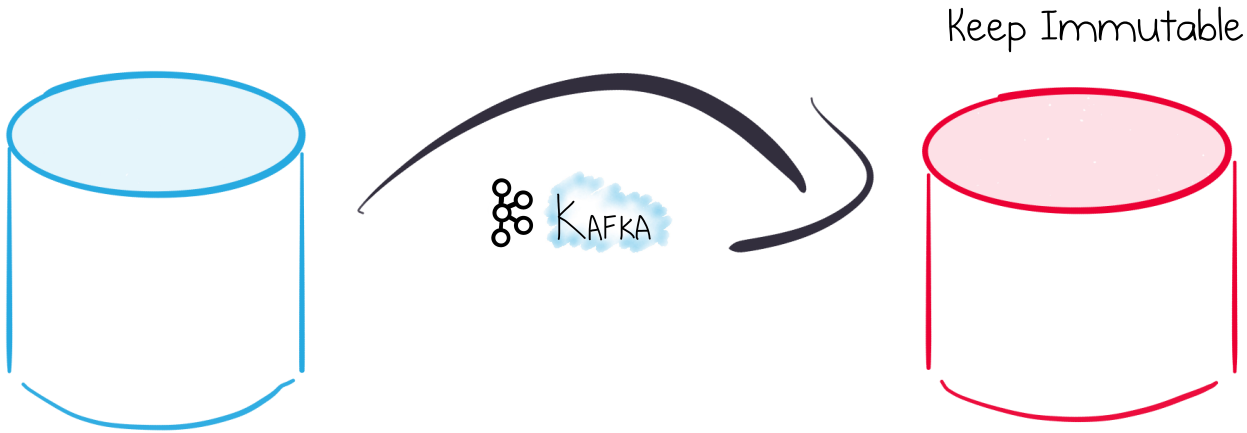
Tool for accessing shared, retentive streams



But sometimes you have to move data

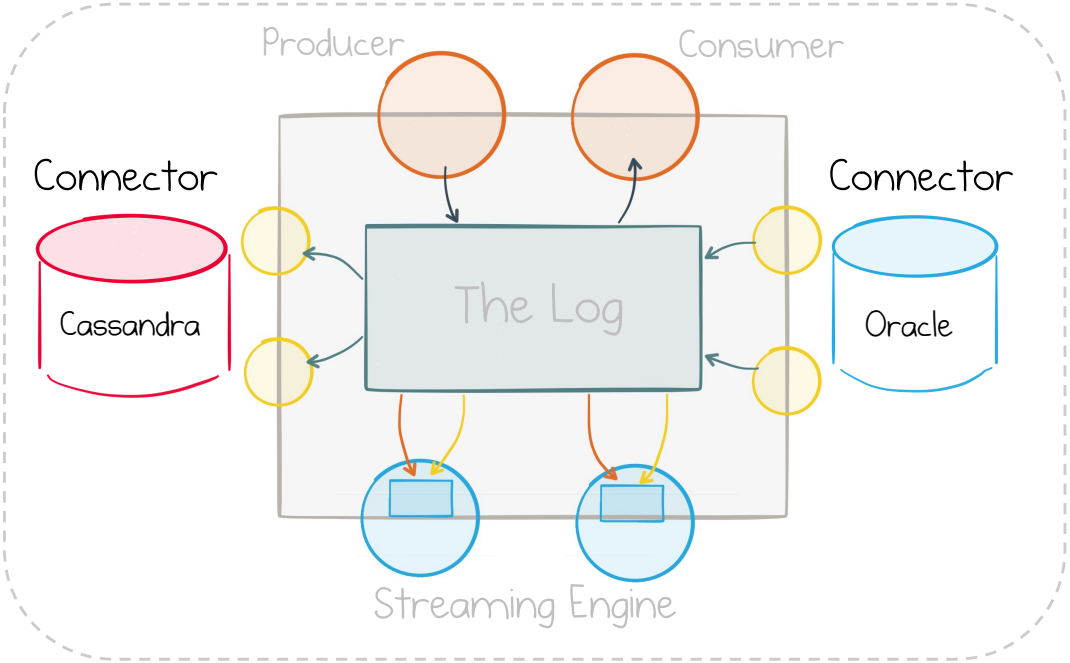


Replicate it, so both copies are identical



Take only the data you need today

Confluent's Connectors make this easier



So...

When building services consider more than just REST

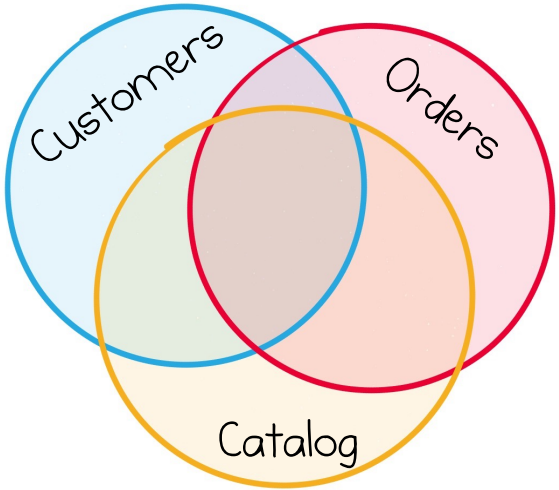
Remember:

THE DATA DICHOTOMY

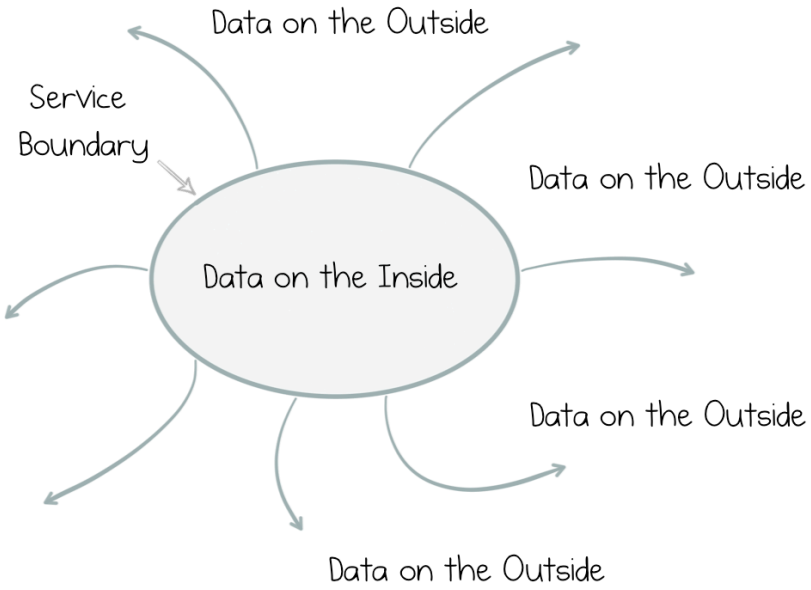
Data systems are about exposing data.

Services are about hiding it.

Shared data is a reality for most organisations

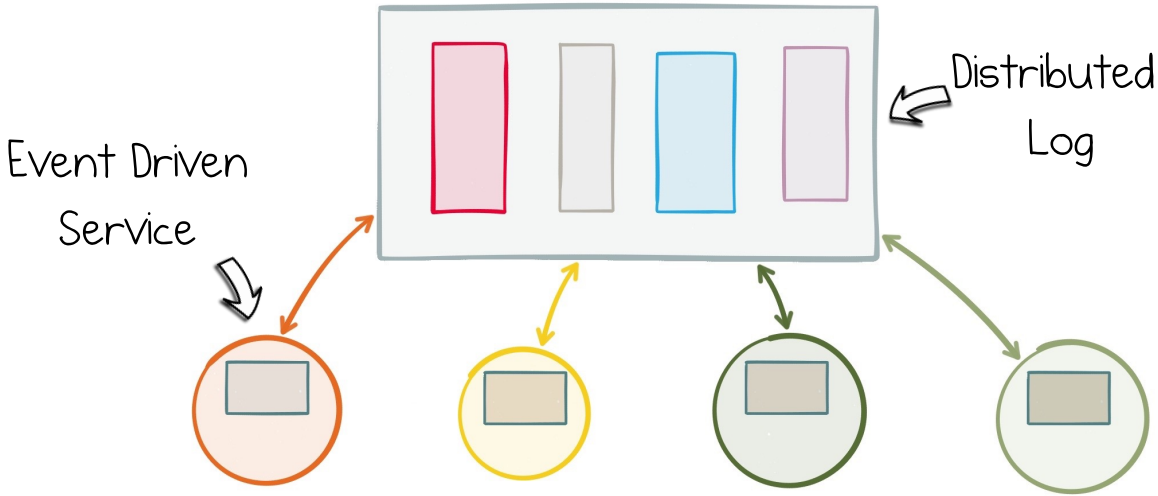


Embrace the data that both lives and flows between services



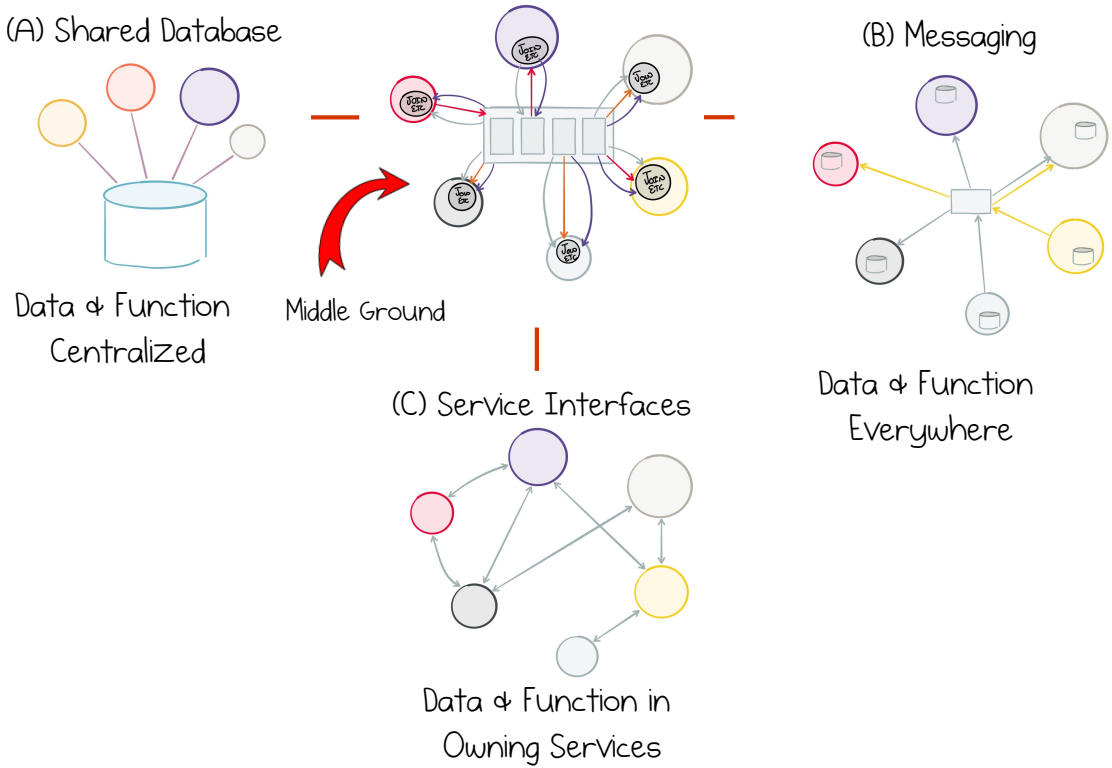
Avoid data services that have complex / amplifying interfaces

Share Data via Immutable Streams



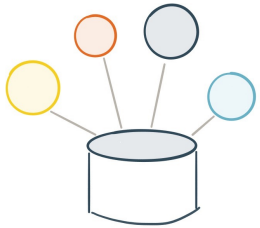
Embed Function into each service

Middle-ground between shared database, messaging and service interfaces



Balance the Data Dichotomy

Shared database



Service Interfaces

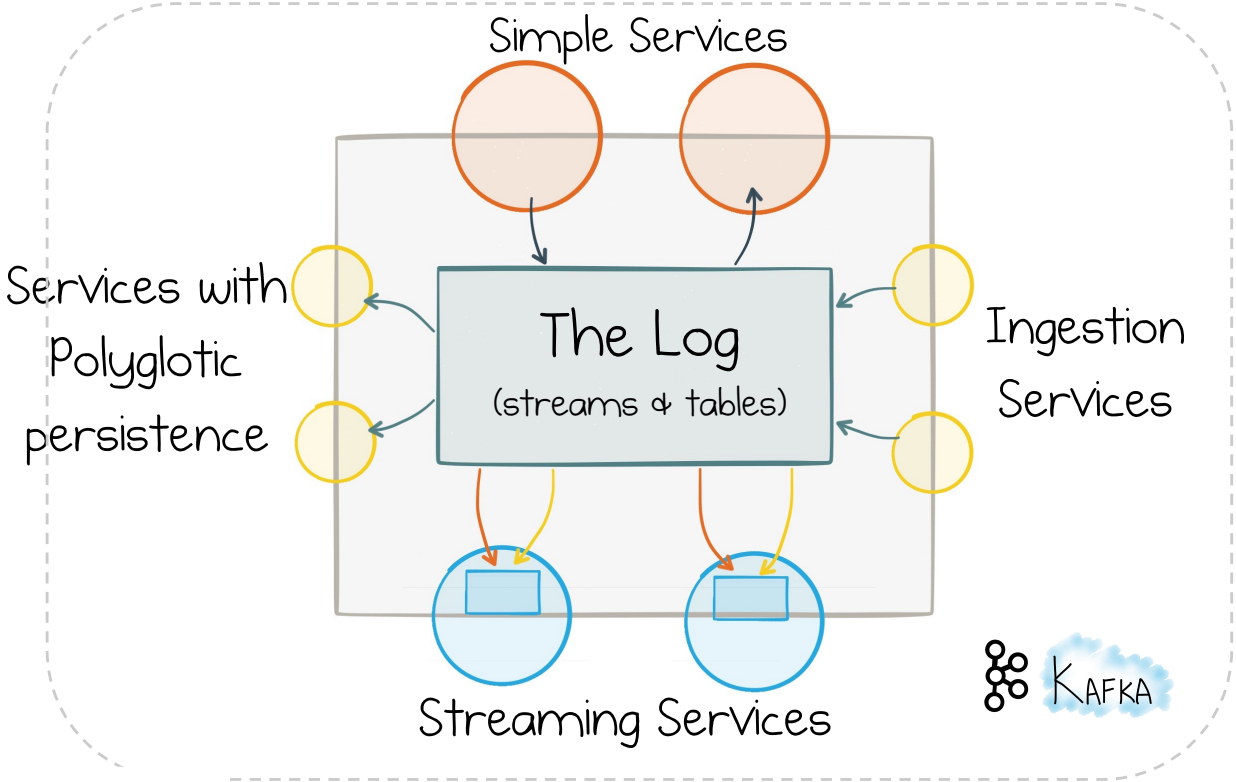
Better Accessibility

Better Independence

RIDER Principals

- Reactive - Leverage Asynchronicity. Focus on the now.
- Immutable - Build an immutable, shared narrative.
- Decentralized - No GOD services. Receiver driven.
- Evolutionary - Use only the data you need today.
- Retentive - Rely on the central event log, indefinitely.

Event Driven Service Backbone



Tune in next time

- How do we actually build these things?
- Putting the micro into microservices

Stay in touch!

Online Talks

cnfl.io/online-talks