Apache Kafka vs. Traditional Middleware (MQ, ETL, ESB)

Friends, Enemies or Frenemies?

Kai Waehner
Technology Evangelist
kontakt@kai-waehner.de
LinkedIn
@KaiWaehner
www.confluent.io
www.kai-waehner.de

Agenda

1. Traditional Middleware
2. Event Streaming Platform
3. Enemies
4. Friends
5. Frenemies

Apache Kafka vs. Traditional Middleware

Agreement:

Kafka is the **de facto standard** for ...

- messaging at scale!
- decoupling of microservices!
- reliable, lightweight stream processing!

Controversial discussion:

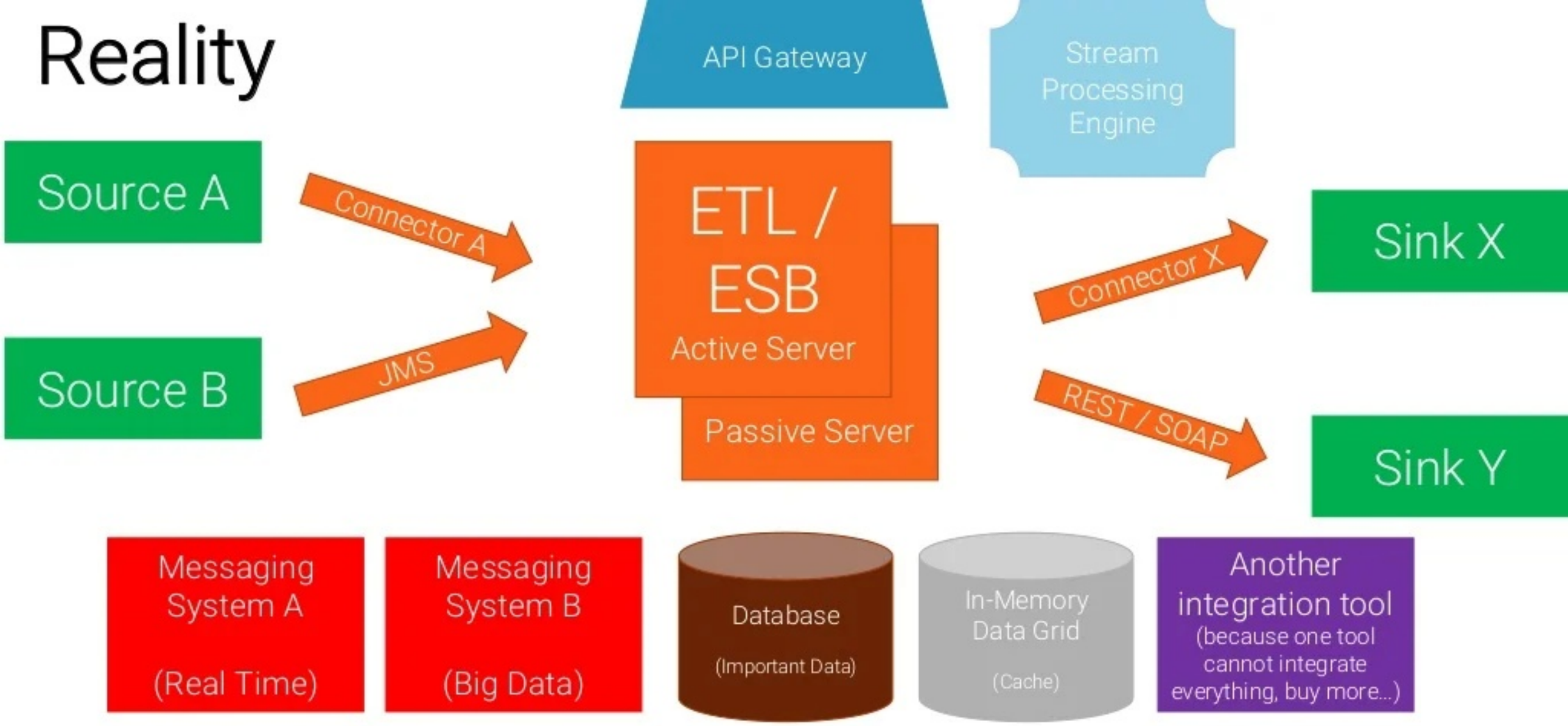
Agenda

1. **Traditional Middleware**
2. Event Streaming Platform
3. Enemies
4. Friends
5. Frenemies

Dream


Source A
Source B
Connector A
JMS
ETL / ESB
Product
XYZ
Connector X
REST / SOAP
Sink X
Sink Y

Reality



Build your custom integration layer… At least 4-5 different products or open source frameworks

Challenges with current integration architectures (MQ / ETL / ESB)

Zoo of technologies

Integration platform (Extract-Transform-Load / Enterprise service Bus) + additional "optional" components

Messaging System (Message Queue, Peer-to-Peer)

In-Memory Cache or Data Grid

Database

Streaming Engine

API Gateway

Architectures with limited scalability and availability

No end-to-end scalability (only parts are built for high volume of messages and high throughput)

No native, built-in scalability (you cannot just add a broker to the running system)

Broker-per-scenario (e.g. hundreds of MQ clusters)

Active-passive clustering

Downtime for maintenance, upgrades, configuration changes

No backwards-compatibility

Tight coupling

Platform-centric integration implementation (→ no separation of concerns, vendor lock-in)

Synchronous communication via request-response (SOAP / REST / gRPC / ...)

No handling of backpressure or unavailable consumers (push-based messaging queues)

The World has Changed

Mobile

Cloud

Microservices

Internet of Things

Machine Learning

Business Digitalization Trends are Driving the Need to Process Events at a whole new Scale, Speed and Efficiency

?

#NoMQ #NoESB #NoETL can handle this well!

Tight coupling, limited scale, zoo of components

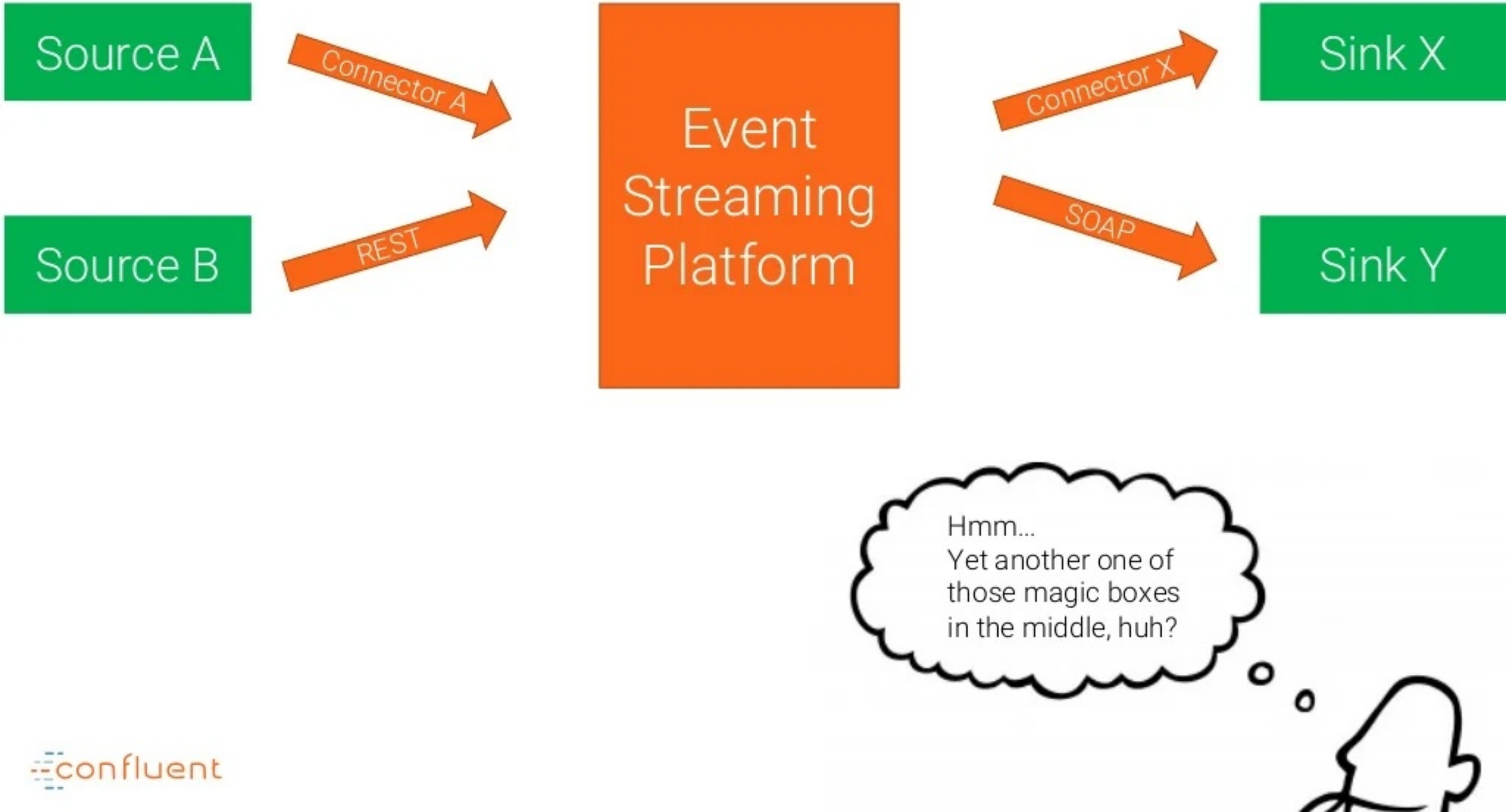
You need to think event-based to realize these use cases!

You need to leverage a single, scalable and loosely coupled platform

Event Streaming Platform!

Agenda

1. Traditional Middleware
2. **Event Streaming Platform**
3. Enemies
4. Friends
5. Frenemies


Source A
Source B
Connector A
REST
Event Streaming Platform
Connector X
SOAP
Sink X
Sink Y
Hmm...
Yet another one of those magic boxes in the middle, huh?

What is an event?

Events

SOMETHING HAPPENED

confluent

Events

A Sale

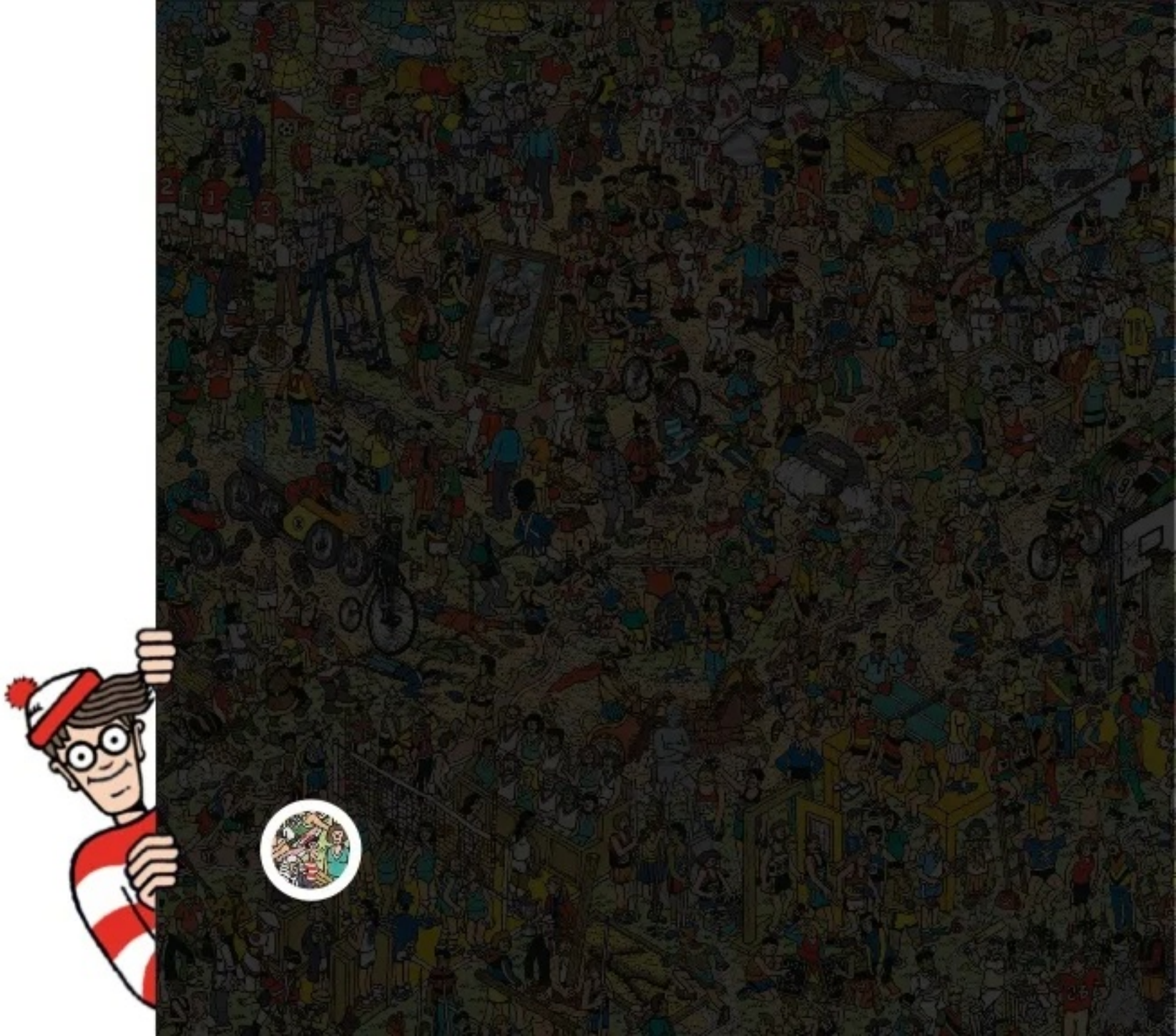
An Invoice

A Trade

A Customer Experience

Where are they?

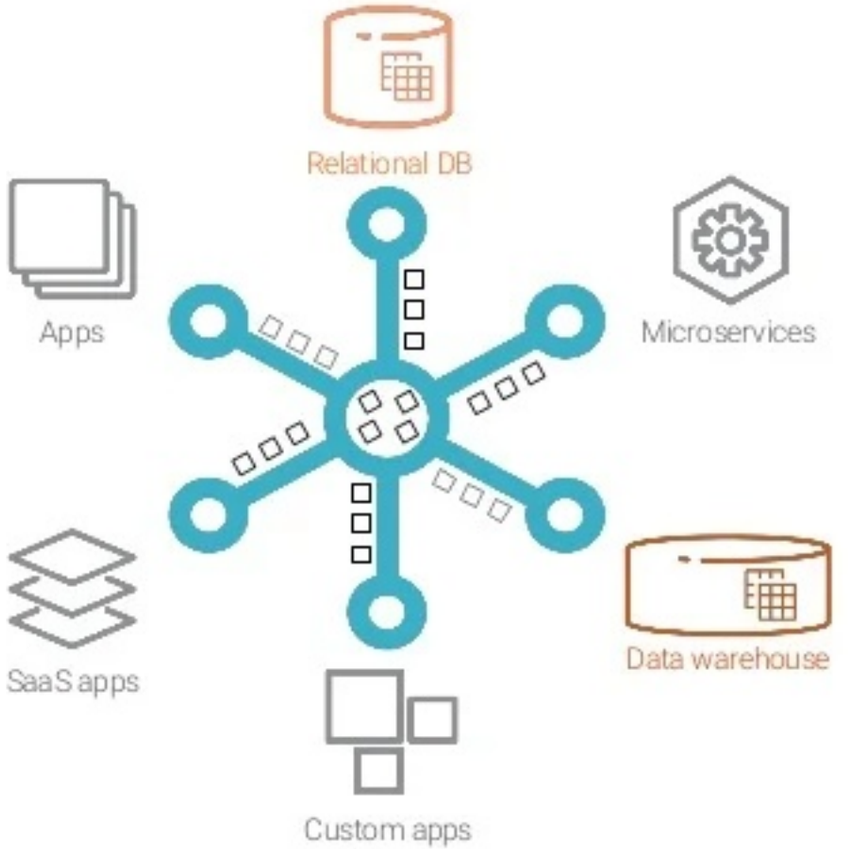
Events haven’t had a proper home in infrastructure or in code. They are implicit.



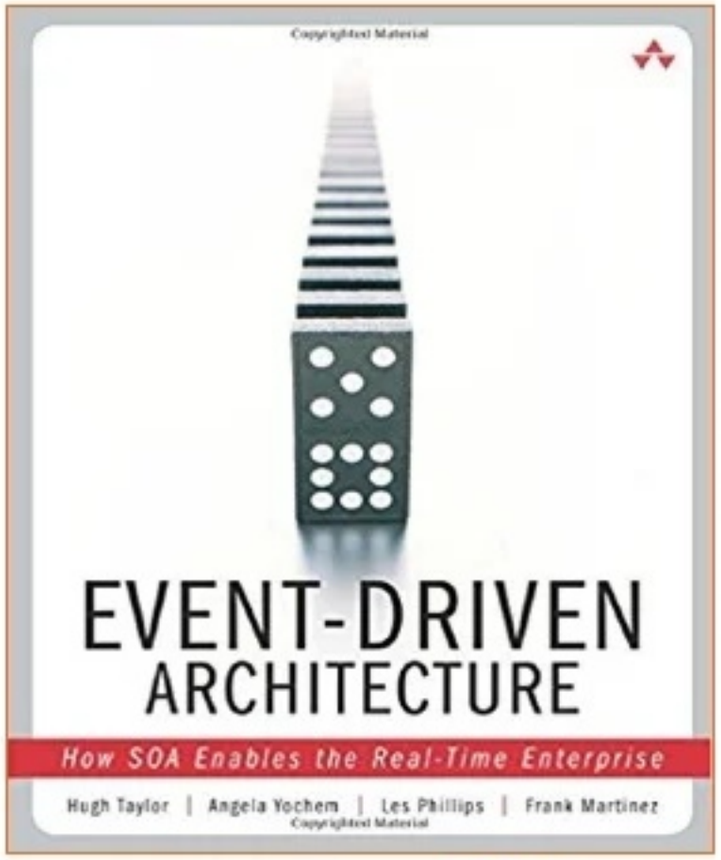
Implementing an Event-driven Architecture Requires a Paradigm Shift

From data represented in static tables and accessed with RPC...

...to data represented as streams of events




Haven’t we seen all
this before?



(Published in 2009)

TIBCO™

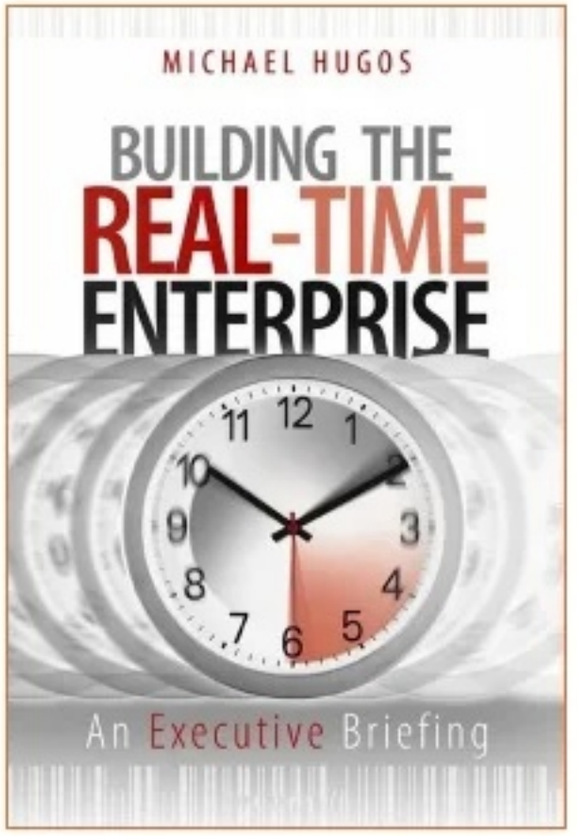
Informatica™

IBM

solace

RabbitMQ

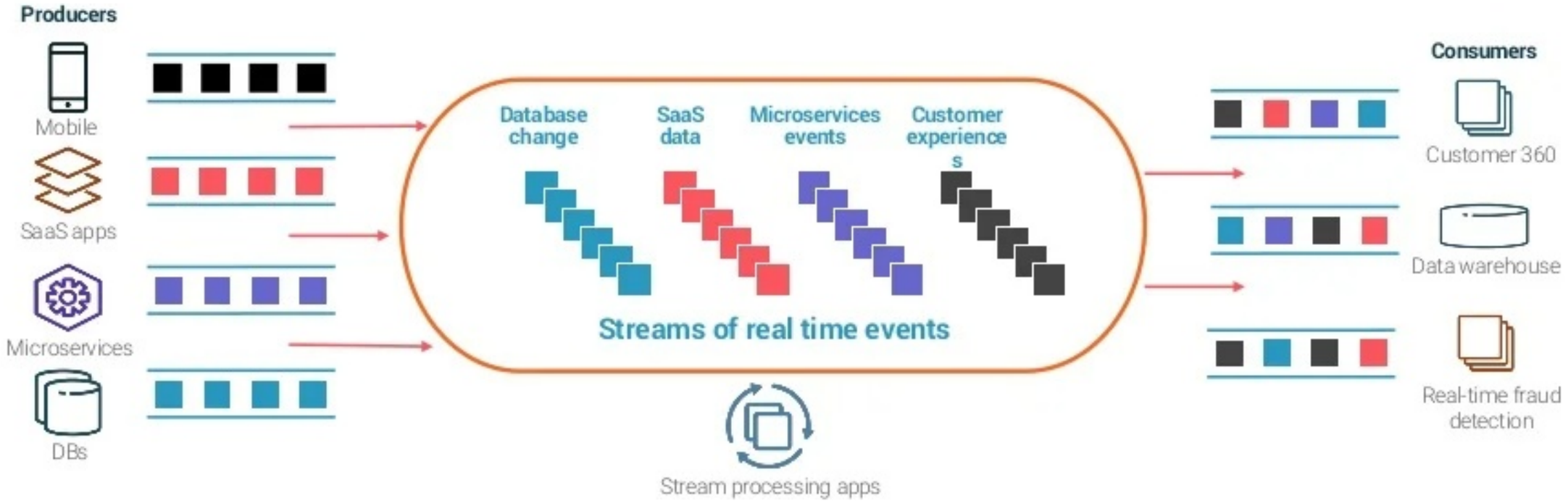
ActiveMQ



(Published in 2004)

What's different this time around?

A Streaming Platform is the Underpinning of an Event-driven Architecture



Ubiquitous connectivity
Globally scalable platform for all event producers and consumers

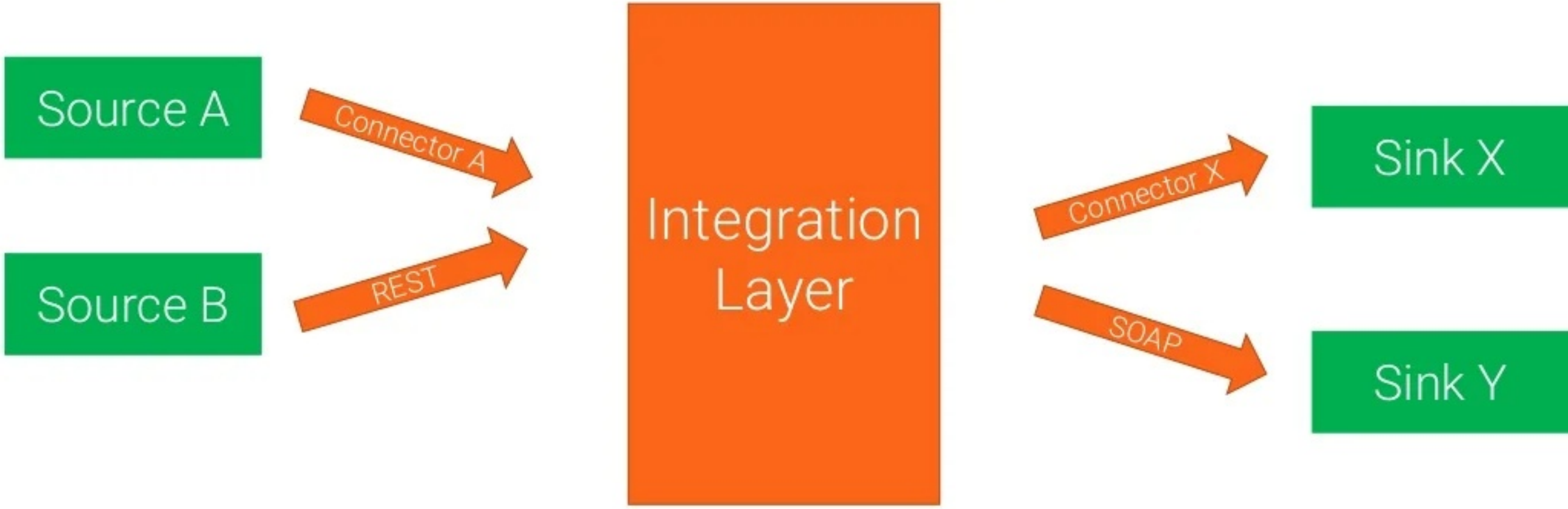
Immediate data access
Data accessible to all consumers in real time

Single System of record
Persistent storage to enable reprocessing of past events

Continuous queries
Stream processing capabilities for in-line data transformation

Agenda

- Traditional Middleware
- Event Streaming Platform
- Enemies
- Friends
- Frenemies



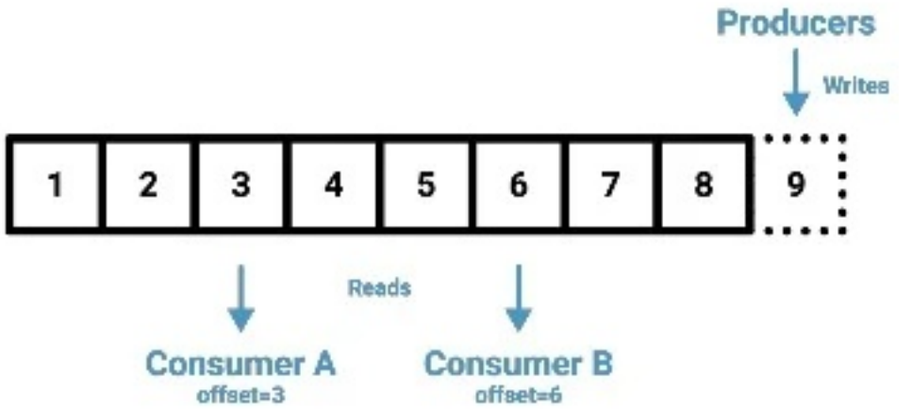
Middleware: Message Queue… ETL… Enterprise Service Bus…
Event Streaming Platform: Apache Kafka.
Spoilt for Choice!

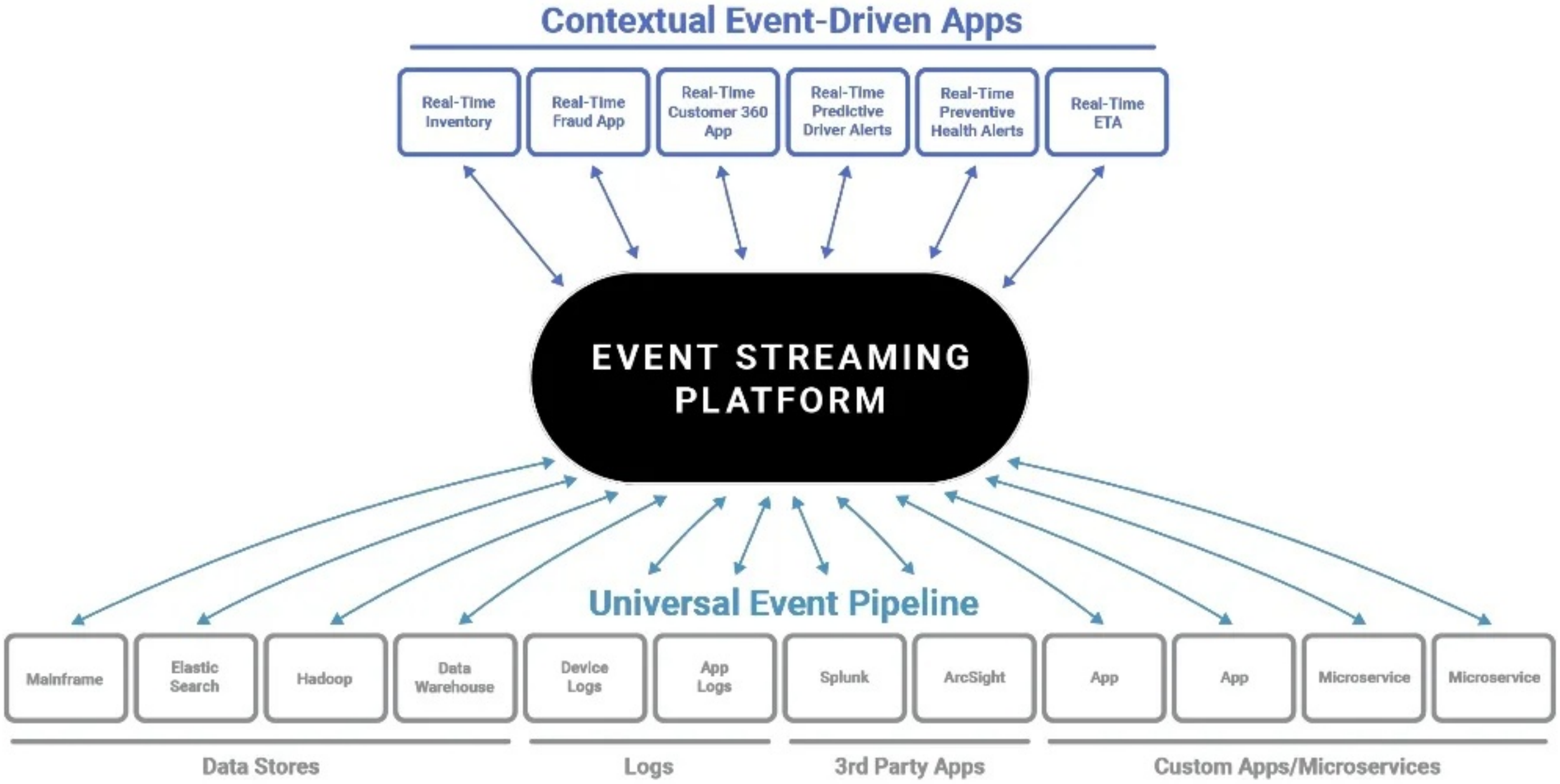
Why do you need an Event Streaming Platform? Remember:

The World has Changed!

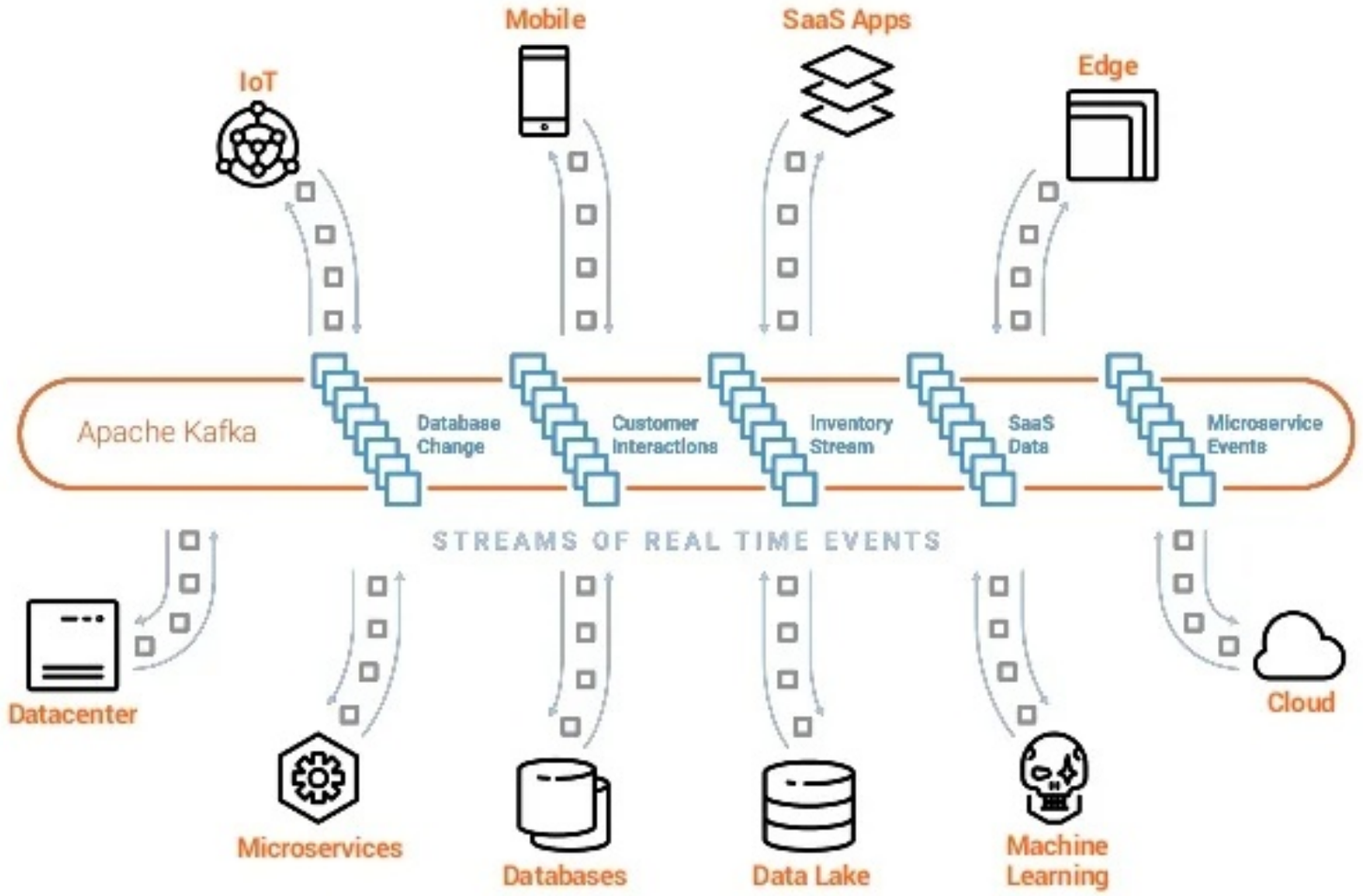
Business Digitalization Trends are Driving the Need to Process Events at a whole new Scale, Speed and Efficiency

Event Streaming Paradigm




Contextual Event-Driven Apps
Real-Time Inventory
Real-Time Fraud App
Real-Time Customer 360 App
Real-Time Predictive Driver Alerts
Real-Time Preventive Health Alerts
Real-Time ETA
EVENT STREAMING PLATFORM
Universal Event Pipeline
Mainframe
Elastic Search
Hadoop
Data Warehouse
Device Logs
App Logs
Splunk
ArcSight
App
App
Microservice
Microservice
Data Stores
Logs
3rd Party Apps
Custom Apps/Microservices

Apache Kafka: The De-facto Standard for Real-Time Event Streaming



- Global-scale
- Real-time
- Persistent Storage
- Stream Processing

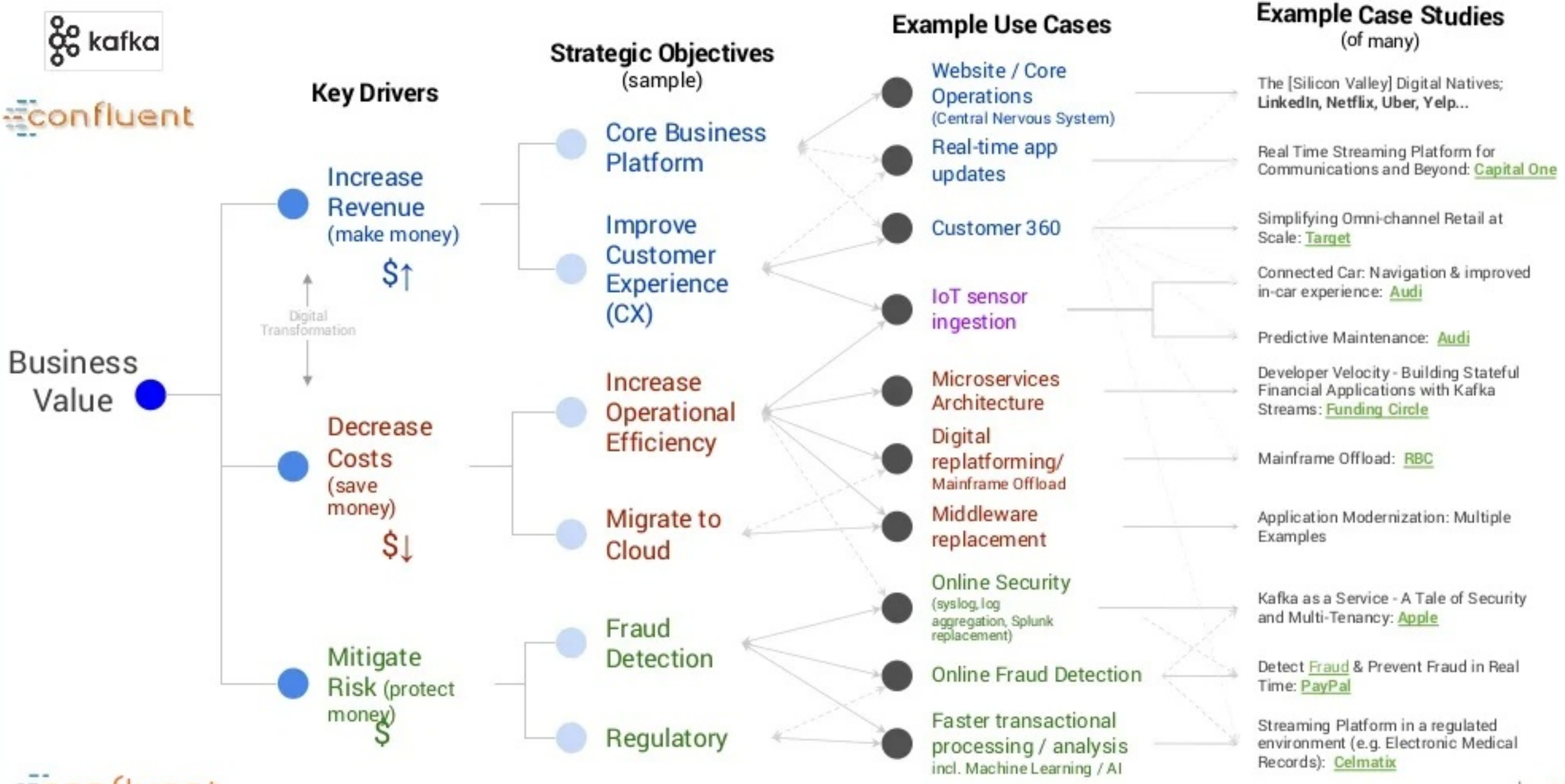
Apache Kafka at Scale at Tech Giants

> 4.5 trillion messages / day

> 6 Petabytes / day

"You name it"

Event Streaming Platform - Value per Use Case



Why Apache Kafka instead of traditional middleware?

Event Streaming Platform

- The core is event-based
- Supports real time stream processing
- But also Fire-and-Forget, Publish / Subscribe, Request-Response / RPC, Batch, …

Single infrastructure

- Messaging, storage, processing
- Extreme Scale and throughput

Reliability and zero downtime (“built for failure”)

- High availability
- Rolling upgrades and dynamic configuration changes
- Backwards compatibility

Decoupling of clients

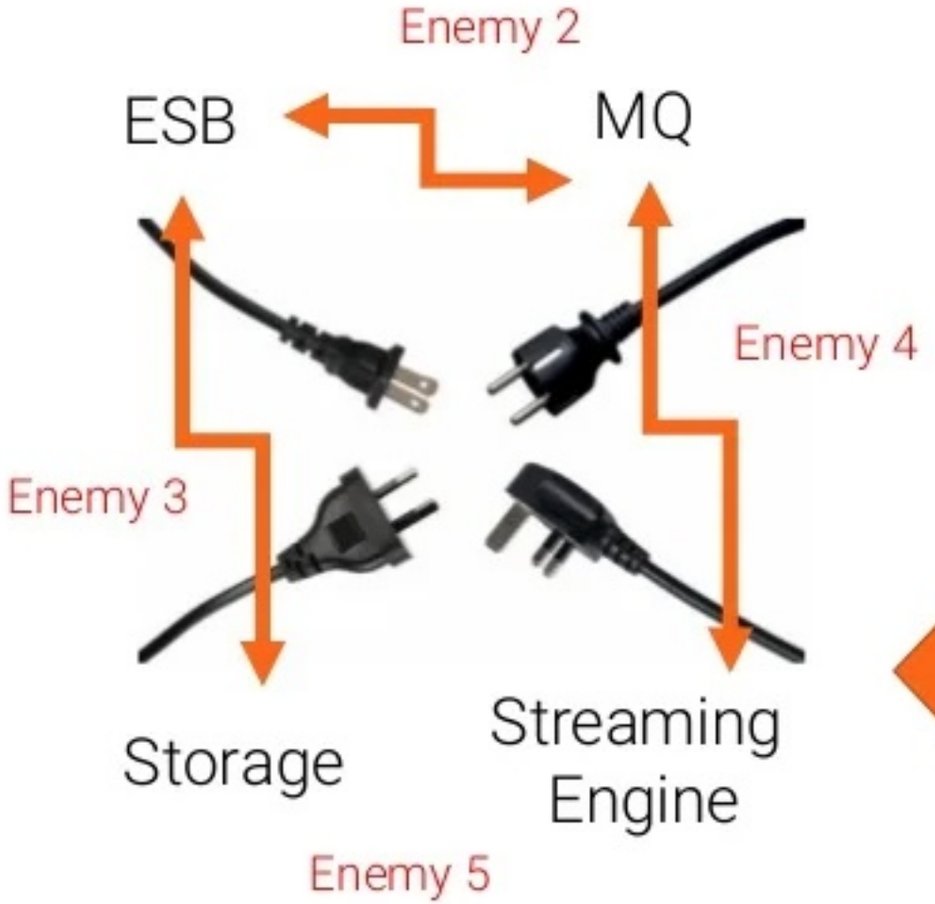
- Agile microservices
- Dumb pipes, smart endpoints
- Handling backpressure

No vendor lock-in

Why Apache Kafka instead of traditional middleware?

"Eat your own dog good!"

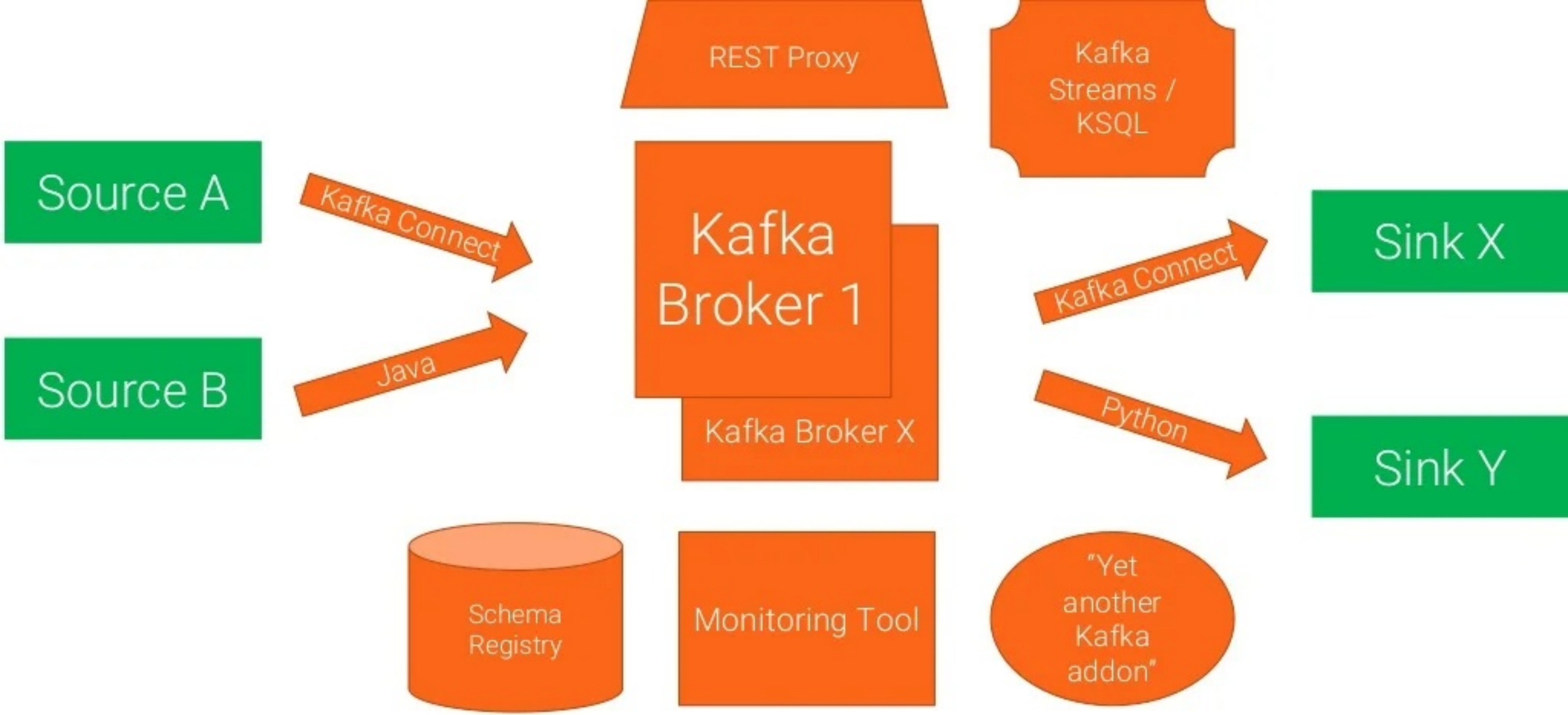
Enemies! .... More components, clusters, technologies means more conflicts, incompatibility, operations burden!



VS.

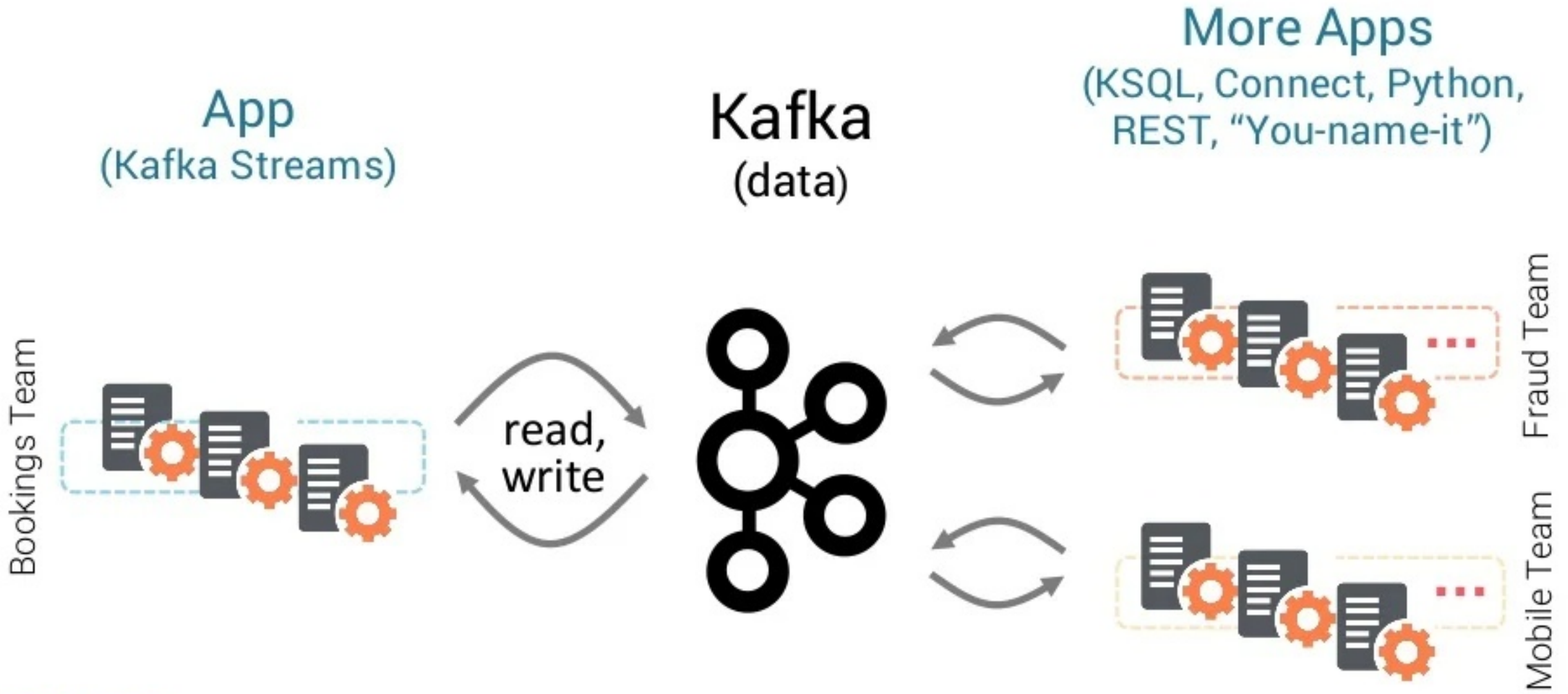
Enemy 1

Messaging: Kafka Core
Storage: Kafka Core
Caching: Kafka Core
Real-Time, Batch: Kafka Clients
Integration: Kafka Connect
Stream Processing: Kafka Streams / KSQL
Request-Response: REST Proxy
"Eat your own dog good"



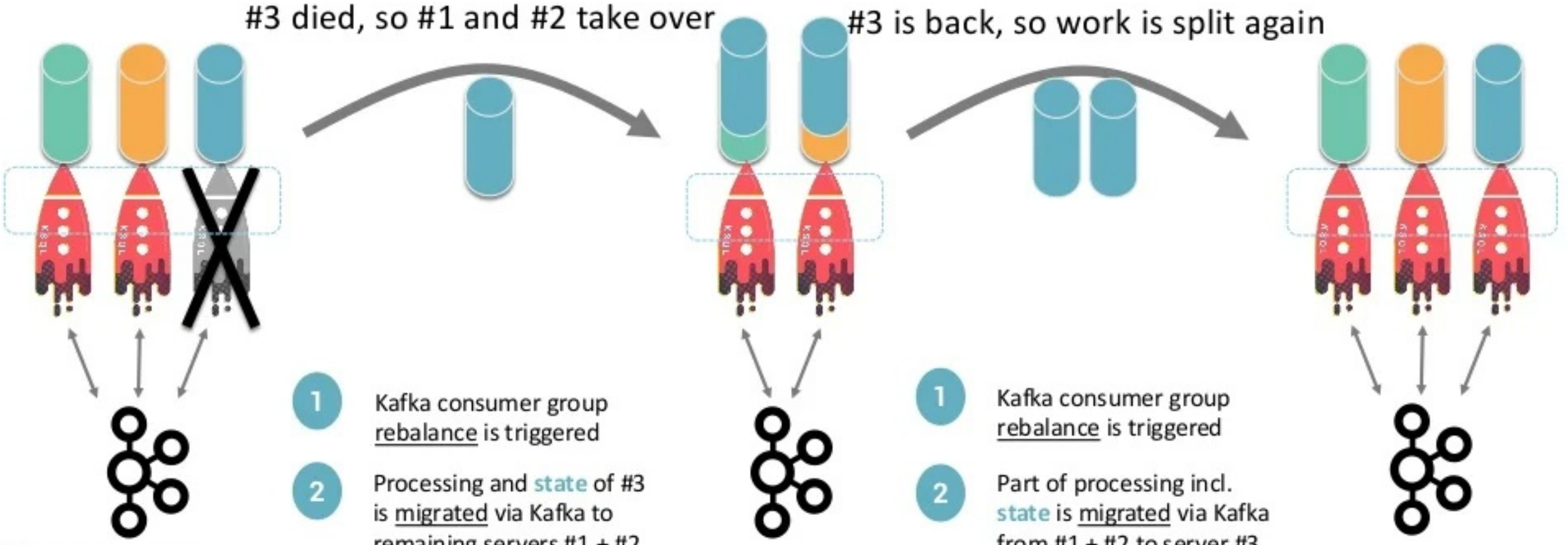
No need for another infrastructure, cluster, database… High availability and scale handled by Kafka Topics!

Independent, scalable, reliable components



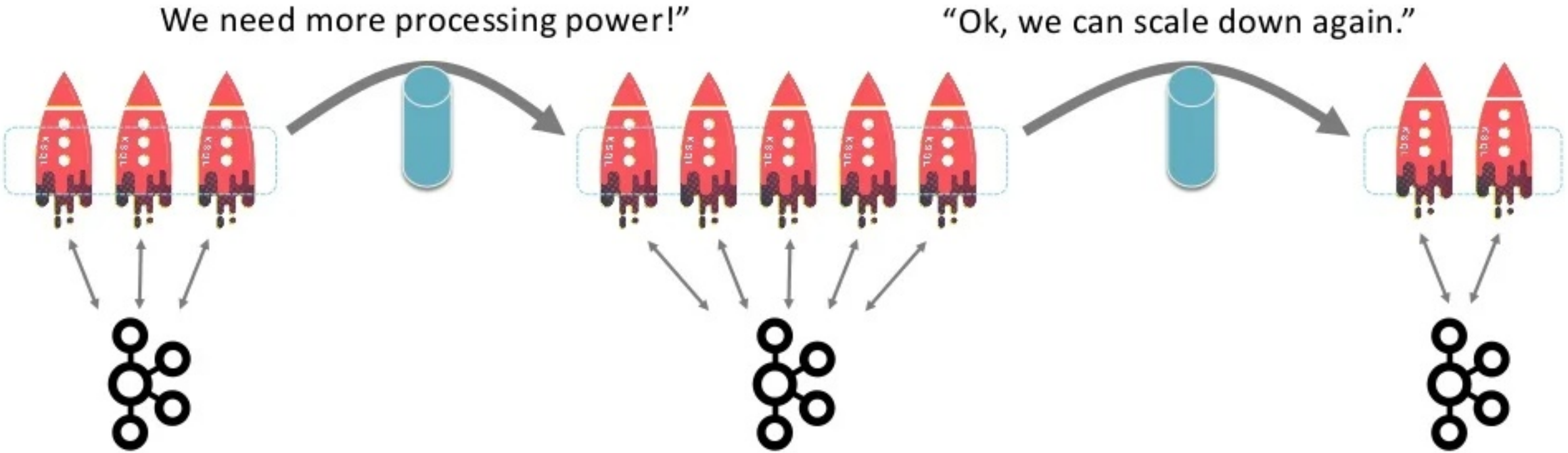
Example - Kafka Streams / KSQL : Fault-tolerance, powered by Kafka

Processing fails over automatically, without data loss or miscomputation.



Example - Kafka Streams / KSQL: Elasticity and Scalability, powered by Kafka

You can add, remove, restart servers during live operations.



Don’t use your ESB knowledge with Kafka!

Recreating ESB antipatterns with Kafka

MAY 2018

HOLD

Kafka is becoming very popular as a messaging solution, and along with it, Kafka Streams is at the forefront of the wave of interest in streaming architectures. Unfortunately, as they start to embed Kafka at the heart of their data and application platforms, we're seeing some organizations **recreating ESB antipatterns with Kafka** by centralizing the Kafka ecosystem components — such as connectors and stream processors — instead of allowing these components to live with product or service teams. This reminds us of seriously problematic ESB antipatterns, where more and more logic, orchestration and transformation were thrust into a centrally managed ESB, creating a significant dependency on a centralized team. We're calling this out to dissuade further implementations of this flawed pattern.

ThoughtWorks®

https://www.thoughtworks.com/radar/techniques/recreating-esb-antipatterns-with-kafka

Agenda

- Traditional Middleware
- Event Streaming Platform
- Enemies
- **Friends**
- Frenemies

Friends!

Plenty of integration options between Kafka and traditional middleware

- Kafka Connect connectors
  (JMS, IBM MQ, RabbitMQ, etc.)
- JMS Client
  (Kafka-native JMS Implementation)
- ESB or ETL tools
  with their own connectors
- Kafka's Client APIs
  (like Java, .NET, Go, Python, Javascript)
- REST Proxy
- Etc.

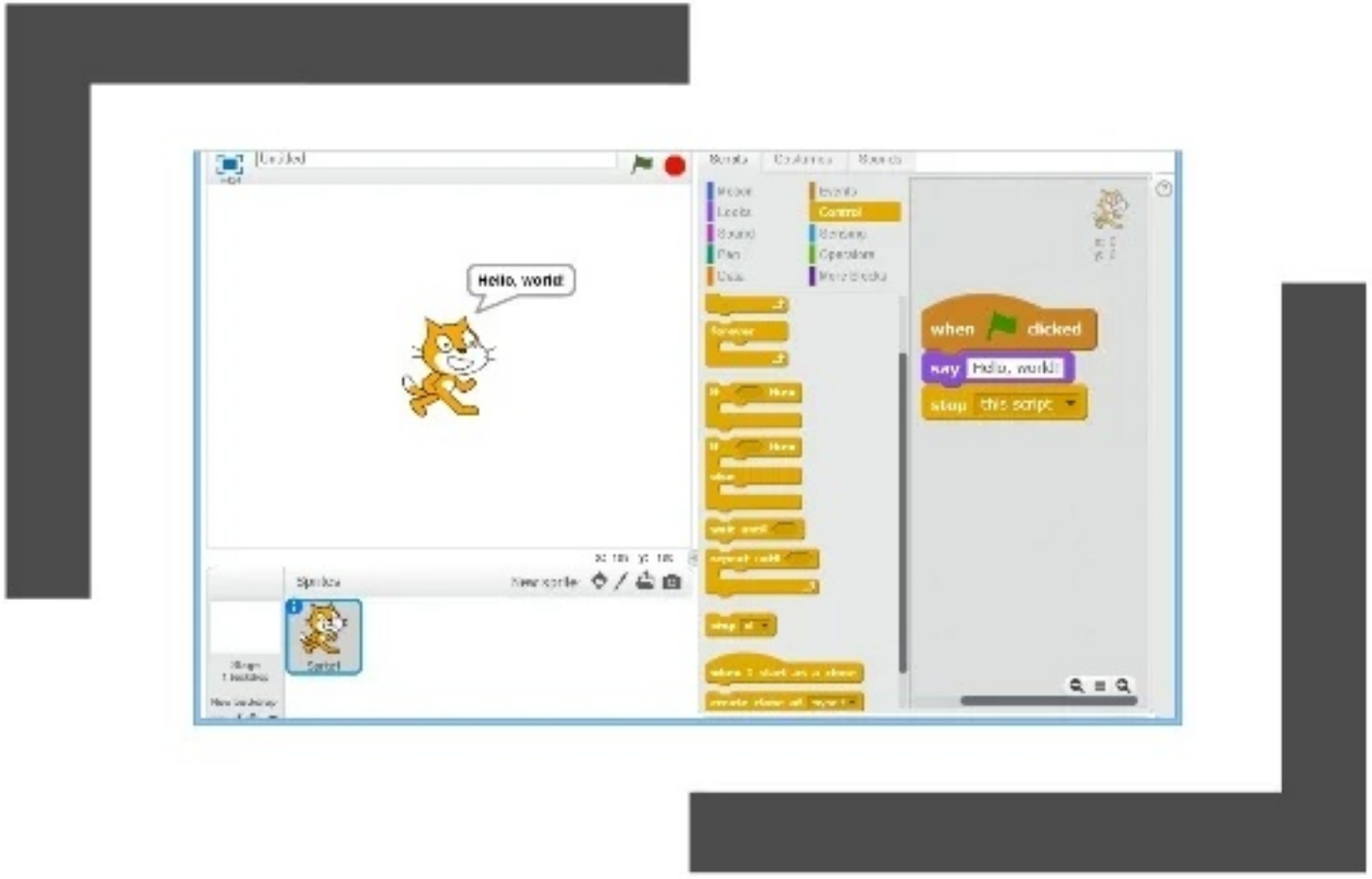
Why friends?

Kafka is NOT THE ALLROUNDER for every single problem!

Maybe you don't need a scalable, reliable, distributed system, but "just":

- Integration with legacy components (Cobol, Edifact, …)
- Point-to-point messaging with active / passive HA for ("a few") messages
- Very specific messaging solution for "less than millisecond performance"
- API Management
- "Real" Batch Processing (Hadoop, Flink, Informatica, …)
- …

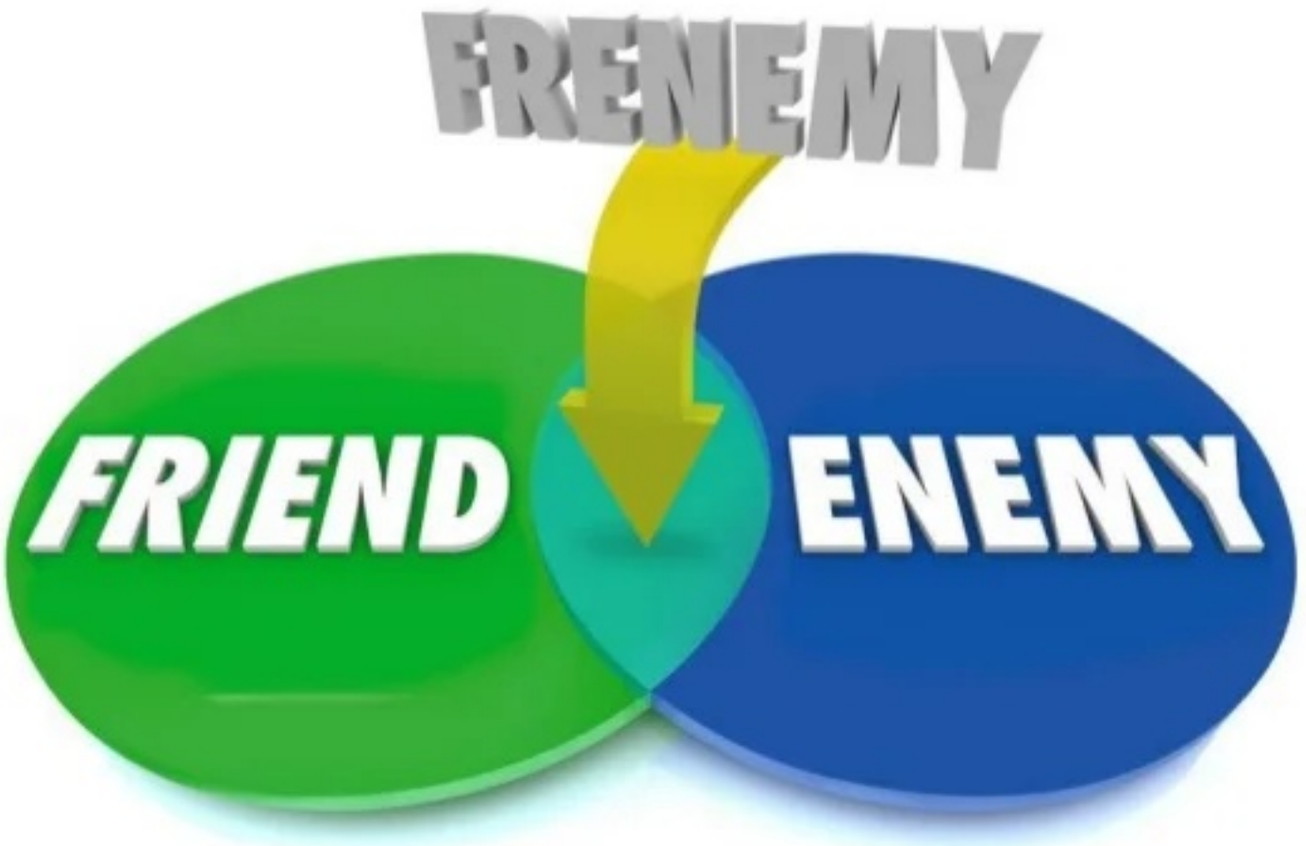
Choose the right integration technology



- Visual coding for complex graphical mappings
- Integration components for complex legacy standard software and protocols
- (SAP BAPI / iDoc, Cobol, Edifact, SOAP / WS*, etc.)

Agenda

- Traditional Middleware
- Event Streaming Platform
- Enemies
- Friends
- Frenemies



Big Bang fails for critical 24/7 deployments

No!

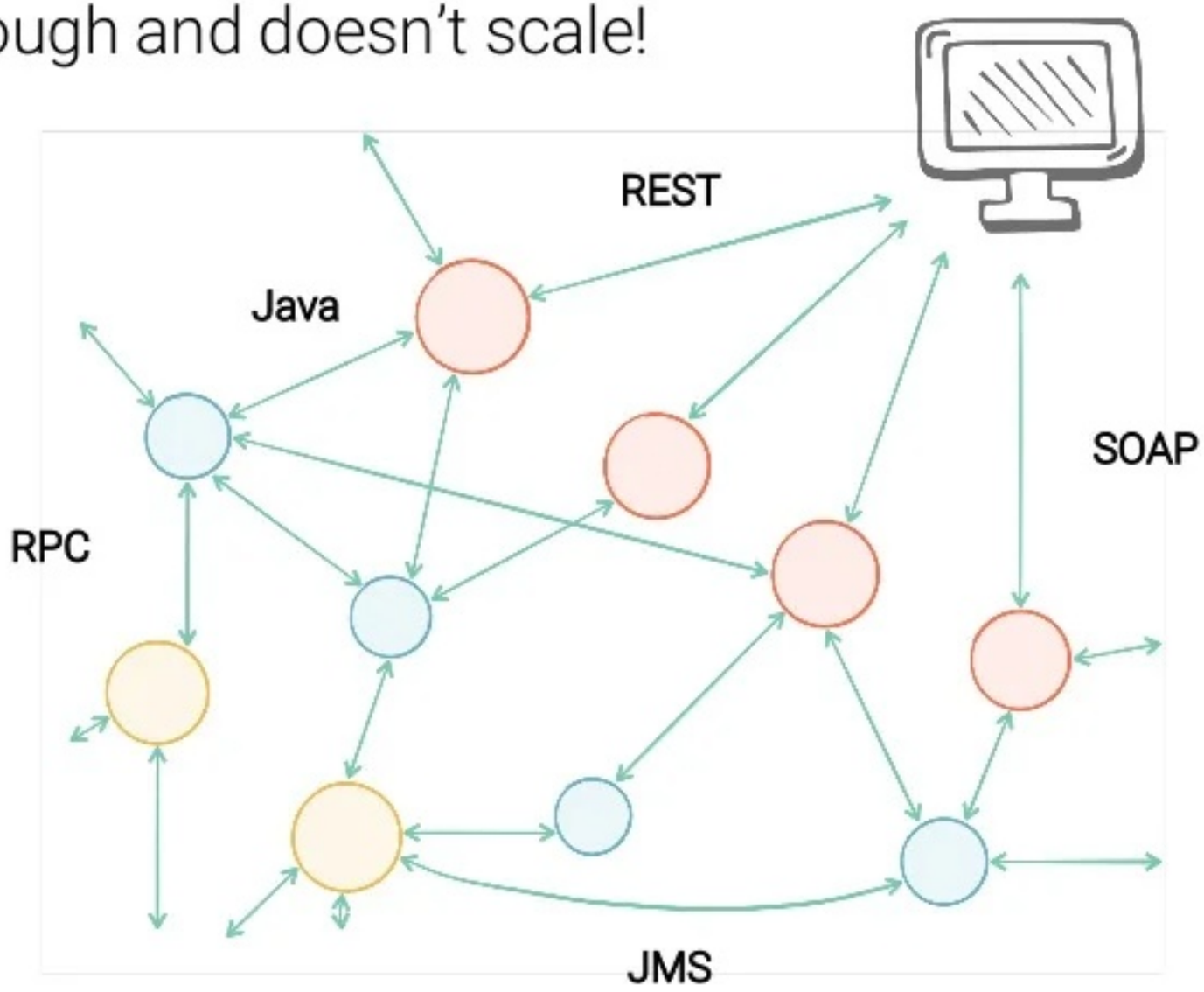
Some technologies never die… Cough…

Can we ever get away from all legacy applications?

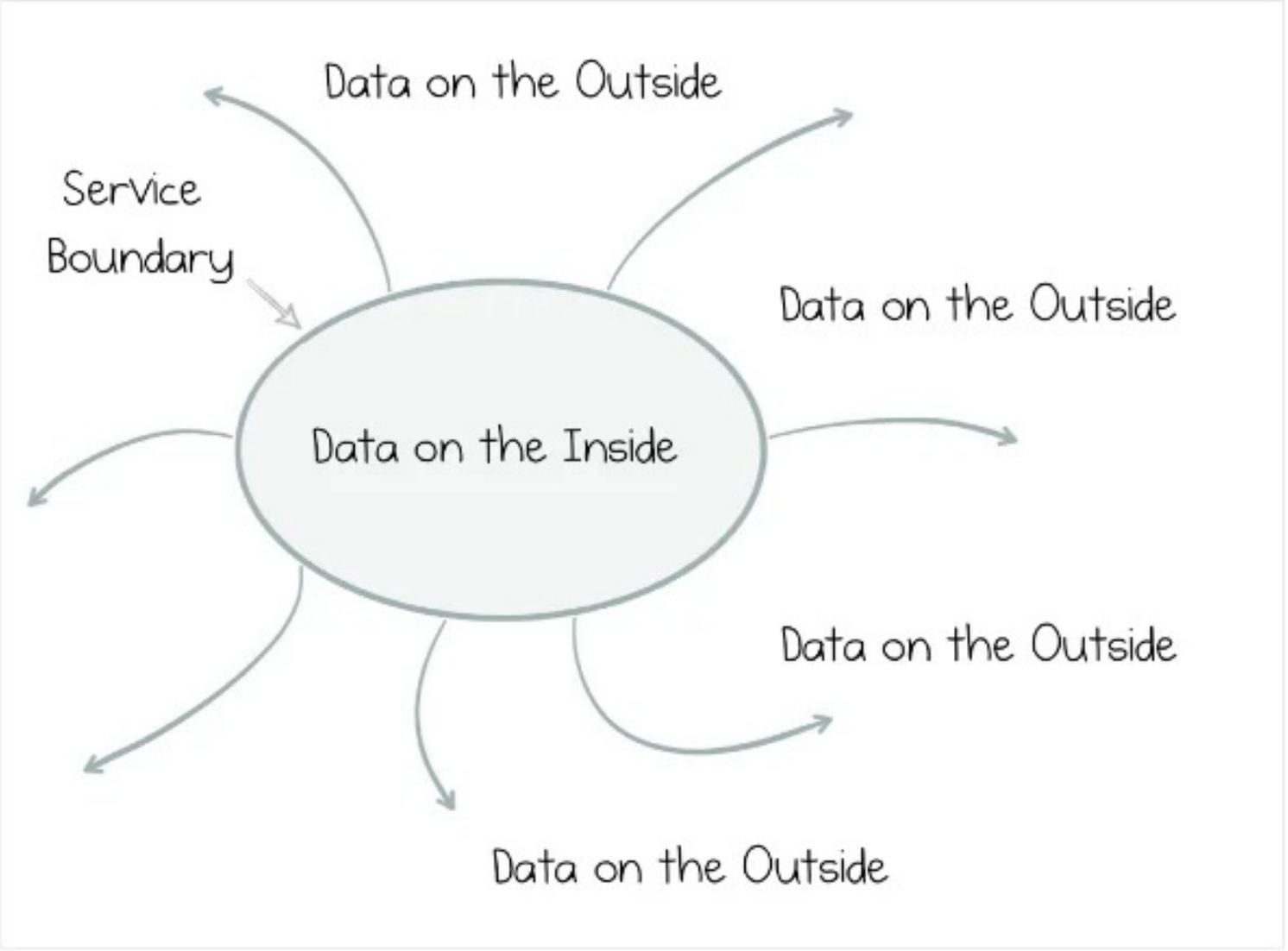
Probably not!

RPC / Request-Response… You can use the ESB for it. BUT:

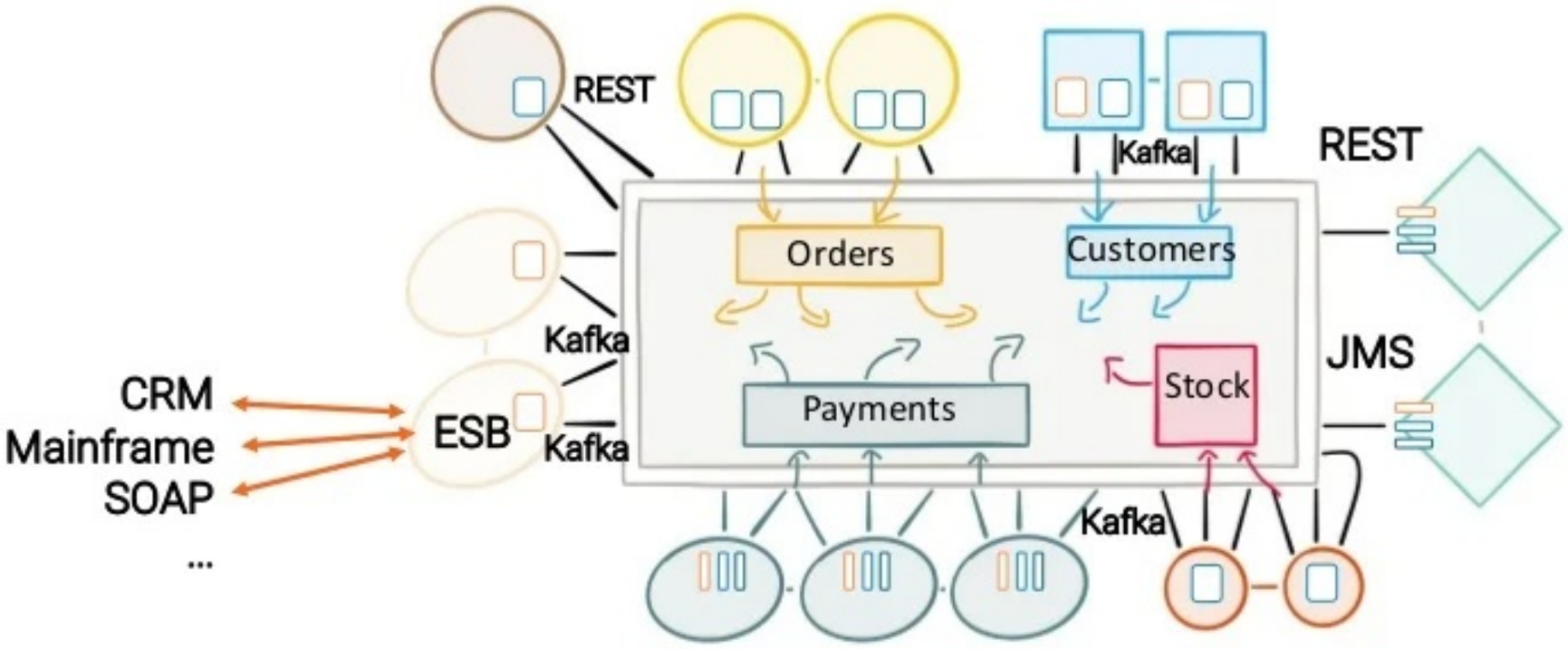
… it often isn't enough and doesn't scale!



Embrace data that lives and flows between services



An Event Streaming Platform gives services independence



Freedom to tap into and manage shared data

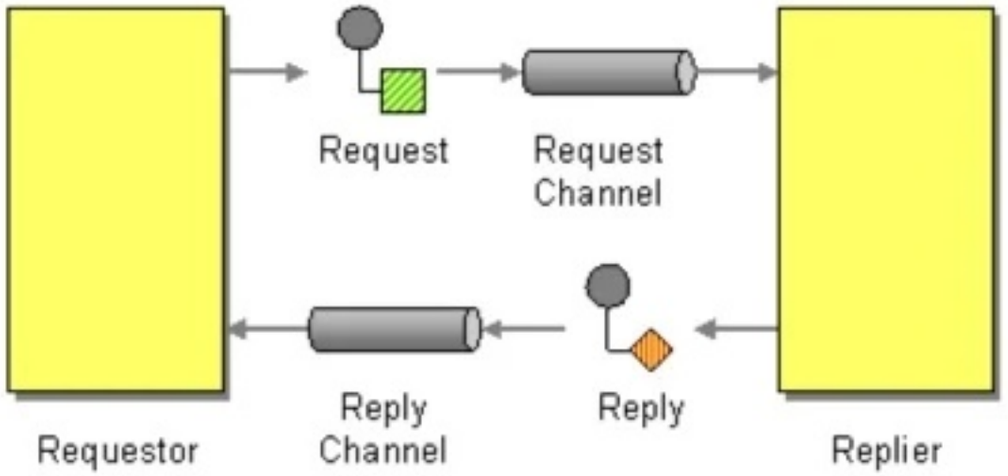
If you really need Request-Response with Kafka → A Correlation ID is your friend!

Request-Reply

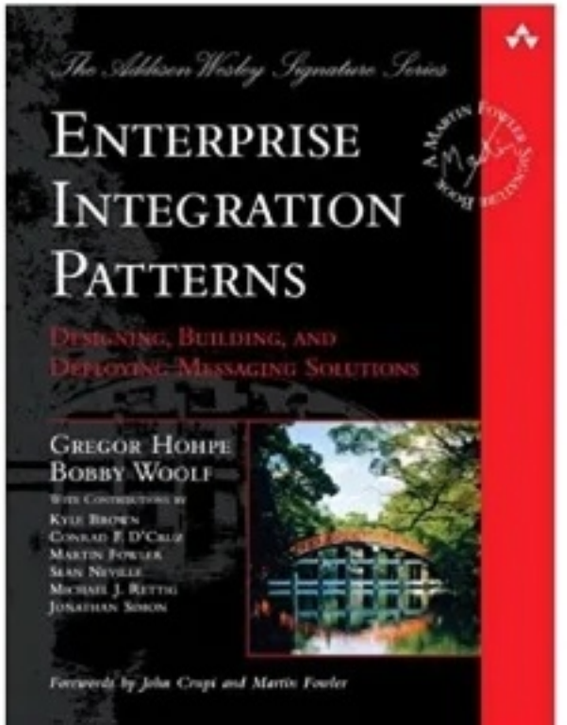
MESSAGING PATTERNS » MESSAGE CONSTRUCTION » REQUEST-REPLY

When two applications communicate via *Messaging*, the communication is one-way. The applications may want a two-way conversation.

When an application sends a message, how can it get a response from the receiver?

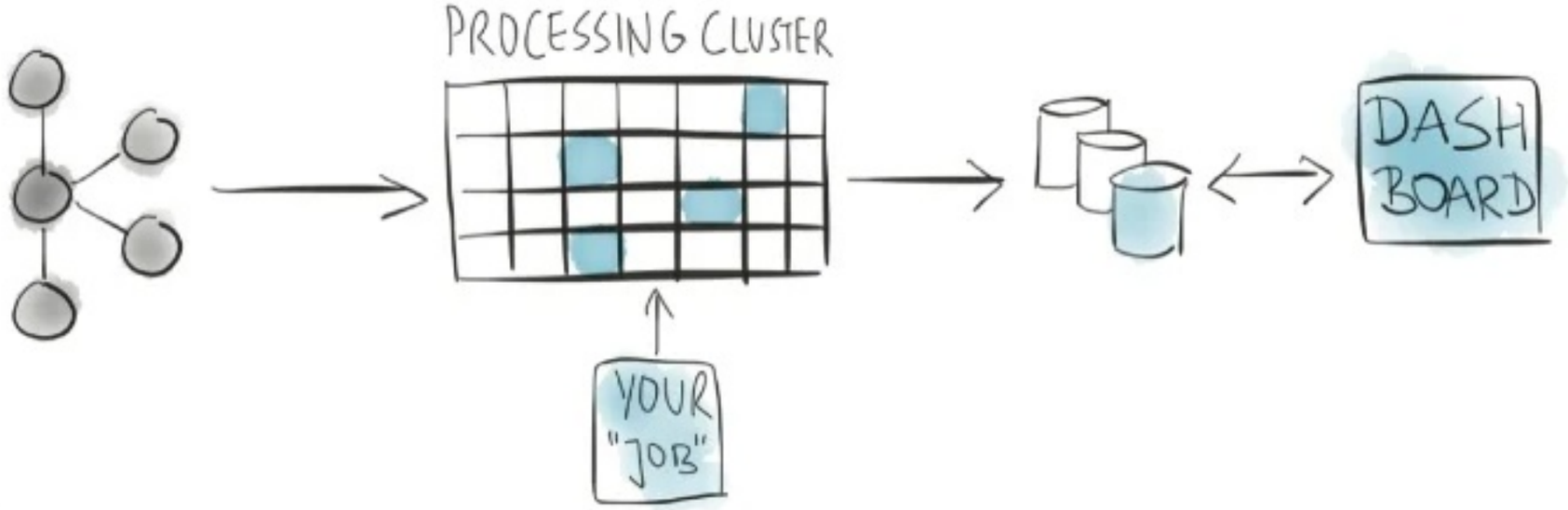


Send a pair of *Request-Reply* messages, each on its own channel.

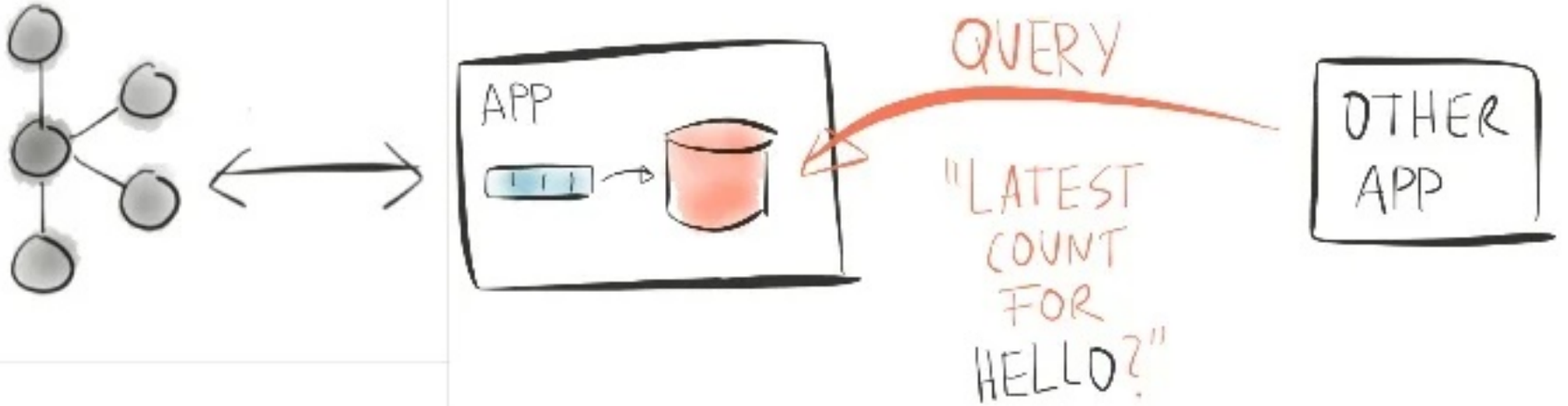


If you need to combine RPC with Streaming: Interactive Queries (IQ) (Kafka Streams)

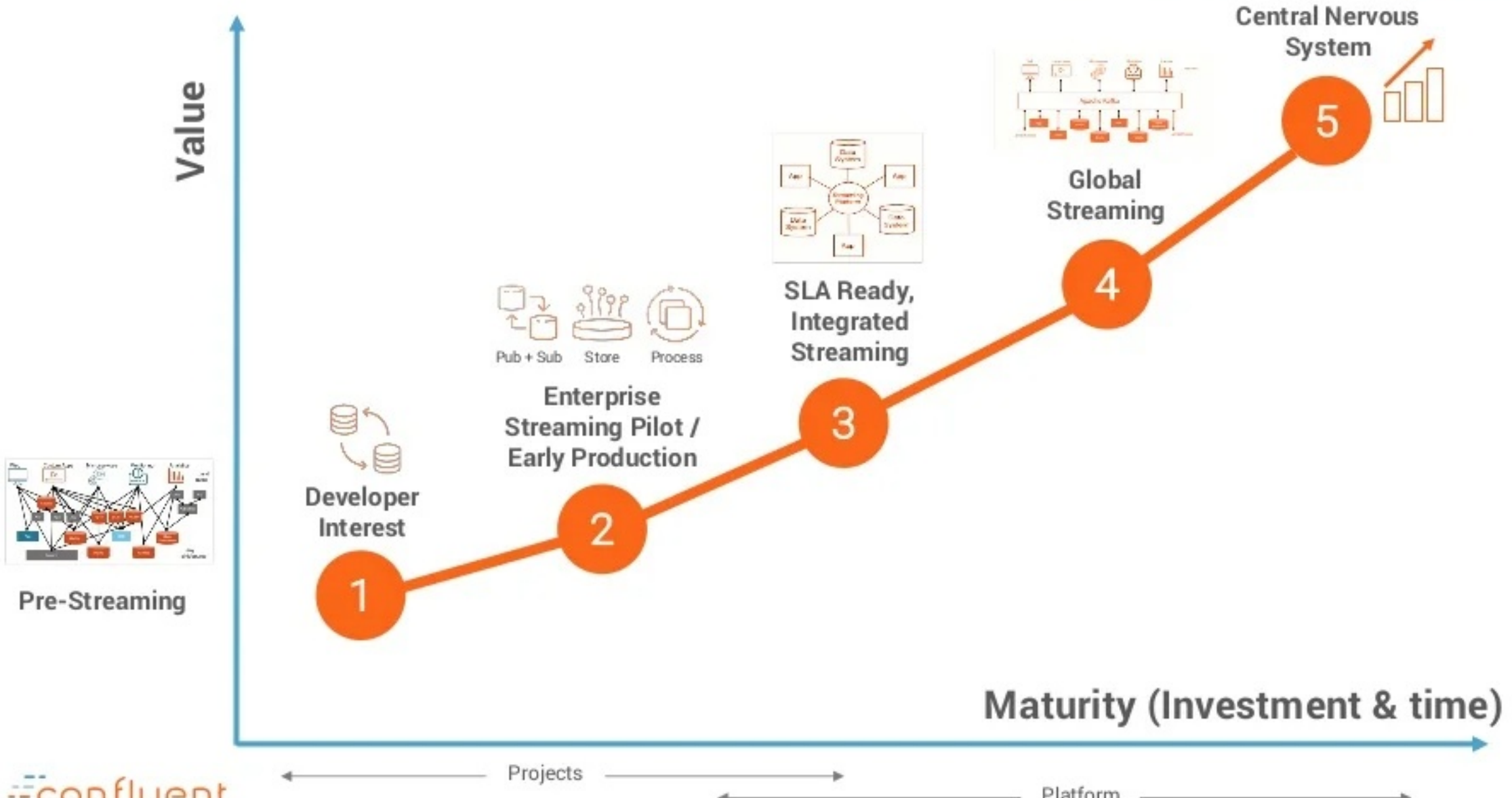
Before IQ ☹



With IQ ☺



Confluent's Streaming Maturity Model - where are you?



Frenemies?

How to integrate
the old and new world?

Mainframe offloading to Apache Kafka

Transaction Data

Date	Amount
1/27/2017	$4.56
1/22/2017	$32.14

Transaction Description

Vendor	Description
Starbucks	Coffee
Walmart	Blu-Ray

Integration via
- Kafka Connect (JMS, MQ)
- REST Proxy
- 3rd party CDC tool
- Etc.

kafka

confluent

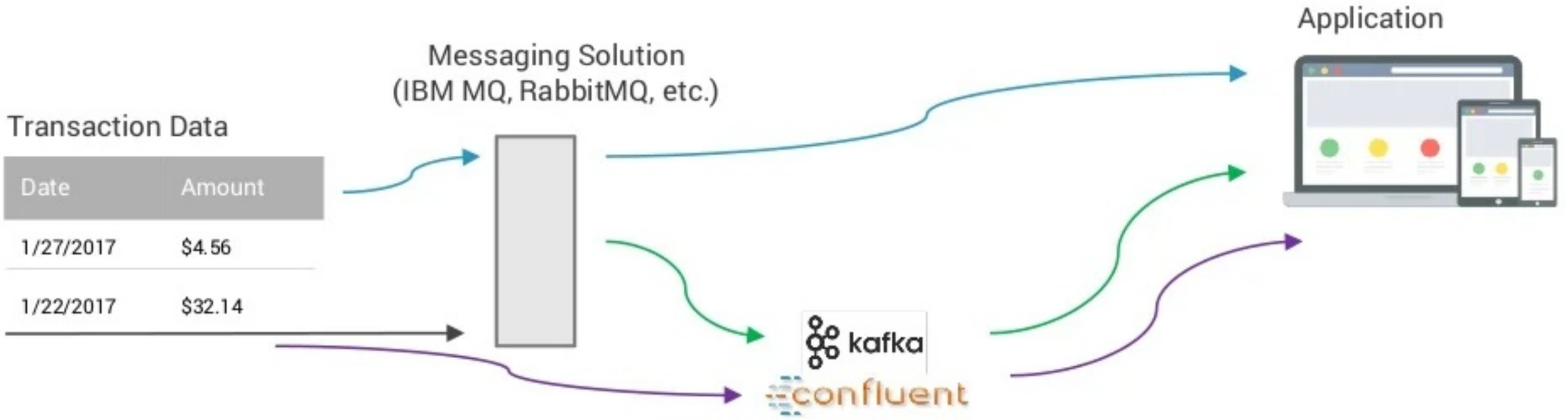
Website

Client profiles

Mainframe MIPS = $$

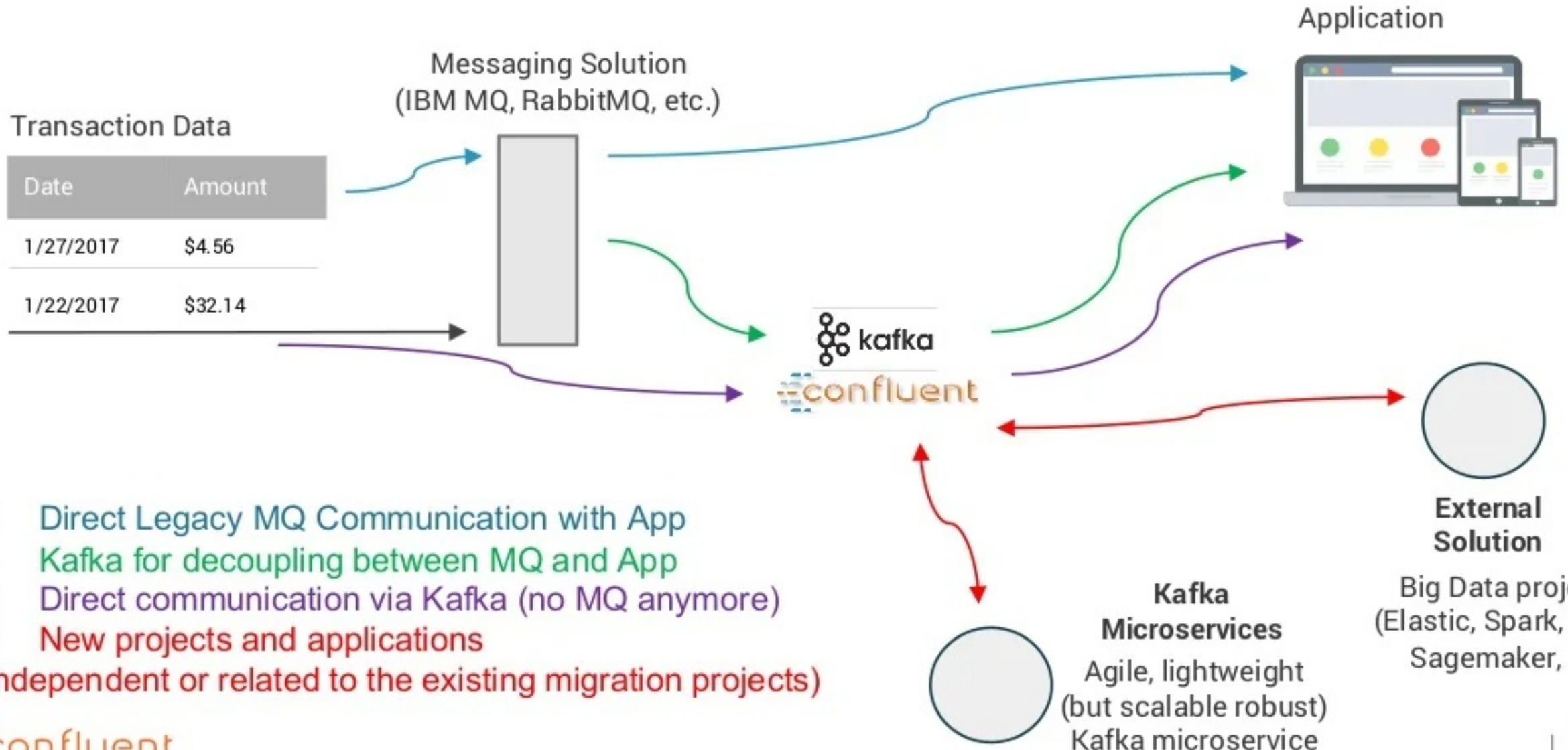
Ability to migrate back if not working well

MQ Integration (and later Replacement)



1) Legacy MQ Communication with App
2) Kafka for decoupling between MQ and App
3) Direct communication via Kafka (no MQ anymore)

New, Innovative Projects and Applications



1) Direct Legacy MQ Communication with App
2) Kafka for decoupling between MQ and App
3) Direct communication via Kafka (no MQ anymore)
4) New projects and applications

(independent or related to the existing migration projects)

Use the right tool for the job (and combine them where it makes sense!)

This is just the beginning of a new era! Confluent Vision for Kafka:

Global

Automated **disaster recovery**

Global applications with **geo-awareness**

Infinite

Efficient and infinite data with **tiered storage**

Unlimited horizontal scalability for single clusters

Faster elastic scaling for brokers and partition

Elastic

Easy Container-based orchestration and management

Faster **elastic scaling** when adding brokers and partitions

Cloud-native Apache Kafka

Questions? Feedback? Let's connect!

Kai Waehner

Technology Evangelist

kontakt@kai-waehner.de
@KaiWaehner
www.confluent.io
www.kai-waehner.de
LinkedIn