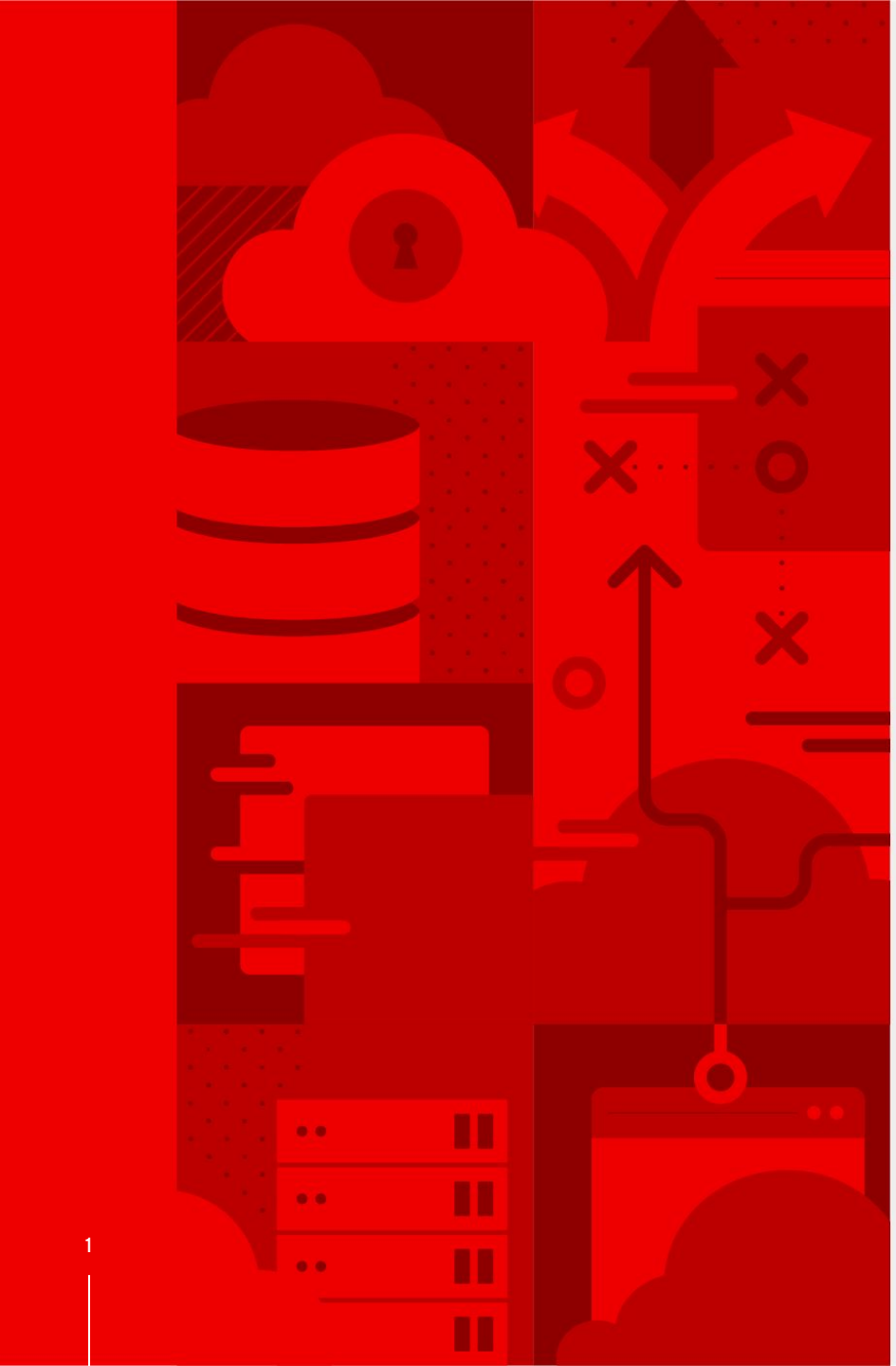


Integration with microservices, events & APIs: What's next?

Marius Bogoevici

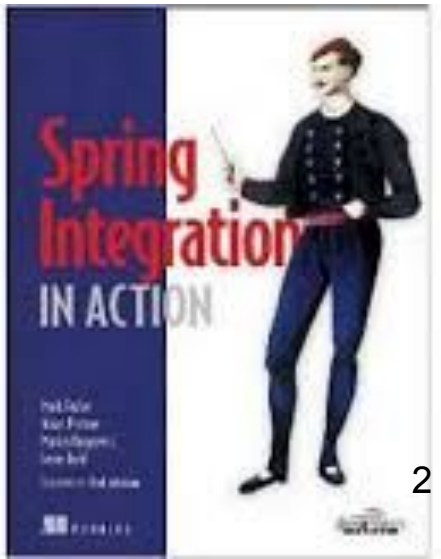
Principal Specialist Solution Architect

Chicago - May 21, 2019

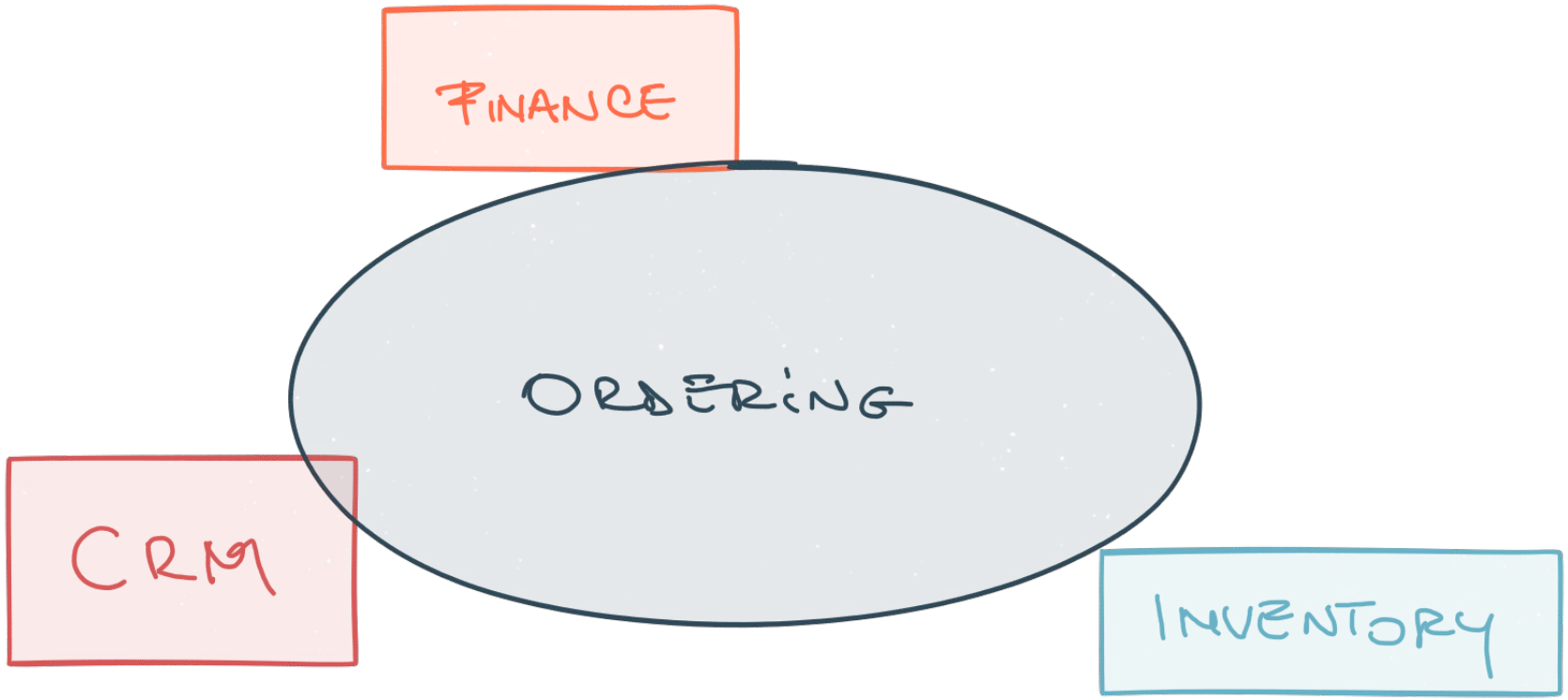
Red Hat

Marius Bogoevici

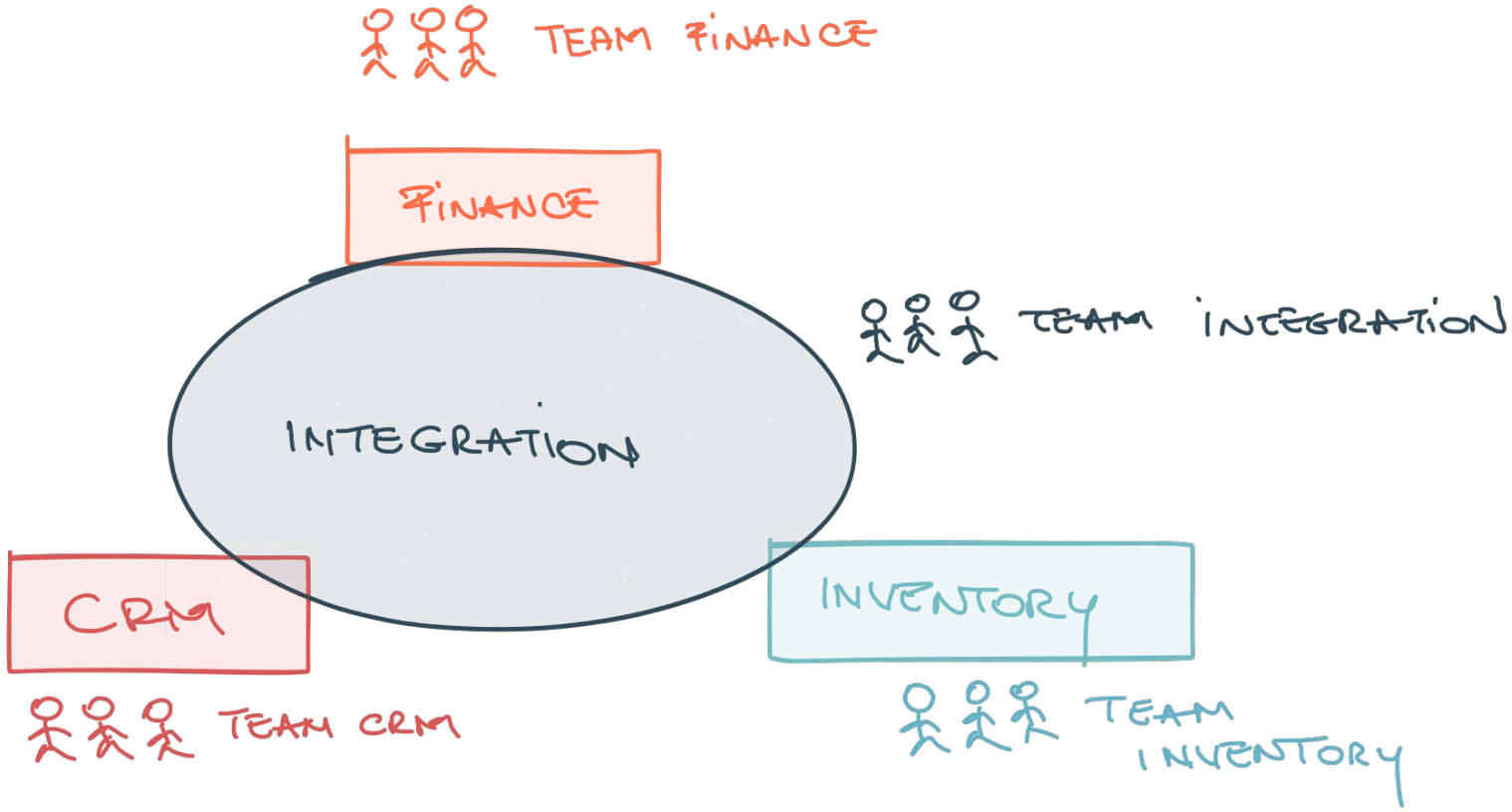
- Principal Specialist Solutions Architect at Red Hat
  - Specialize in Integration/Messaging/Data Streaming
- OSS contributor since 2008
  - Spring Integration
  - JBoss ecosystem
  - Spring XD, Spring Integration Kafka
  - Former Spring Cloud Stream project lead
- Co-author “Spring Integration in Action”, Manning, 2012



Integration: Systems of Systems

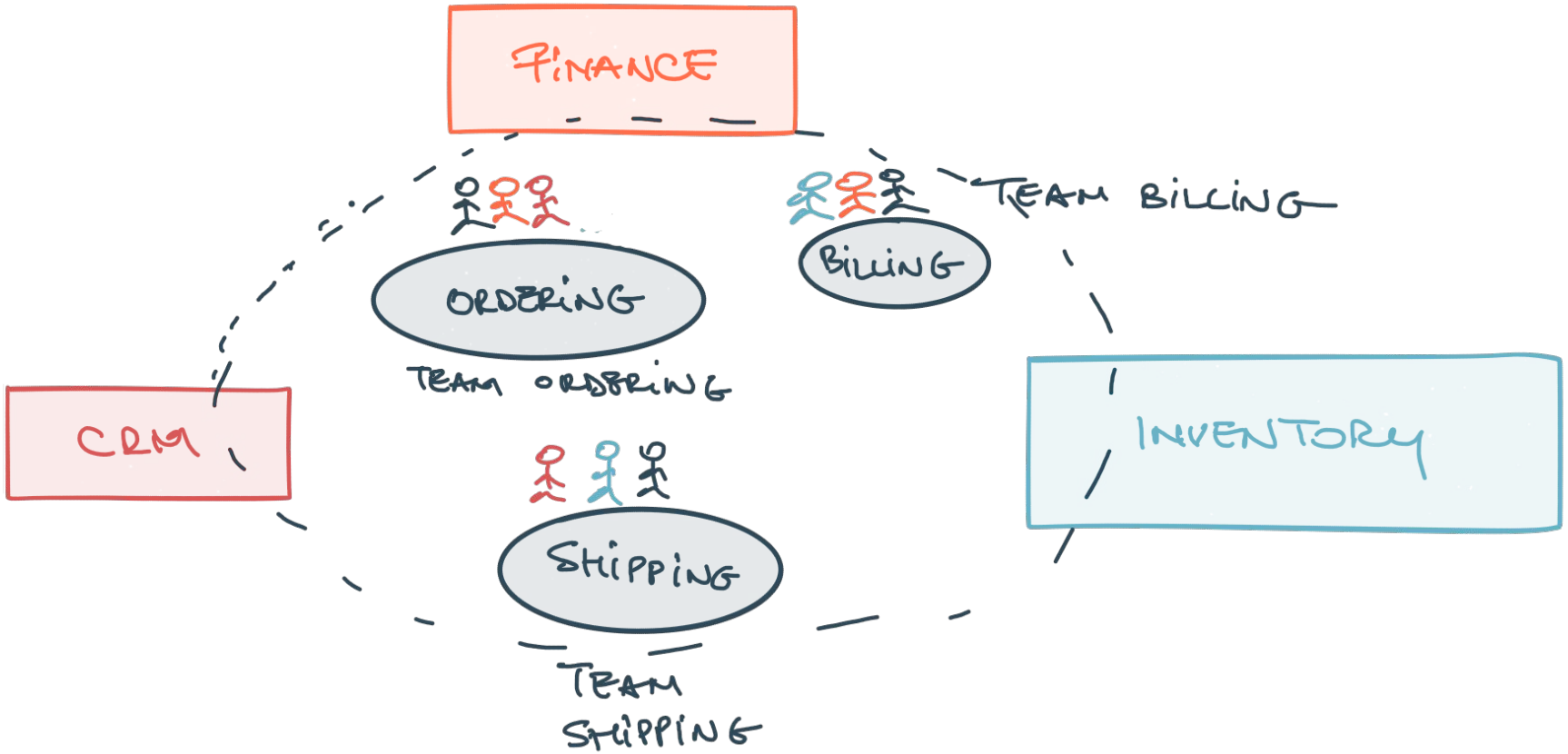


From traditional integration ...



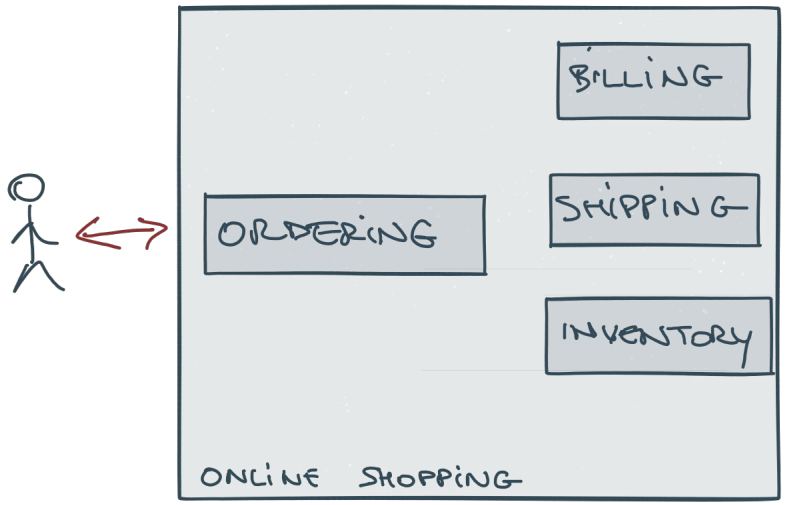
Red Hat

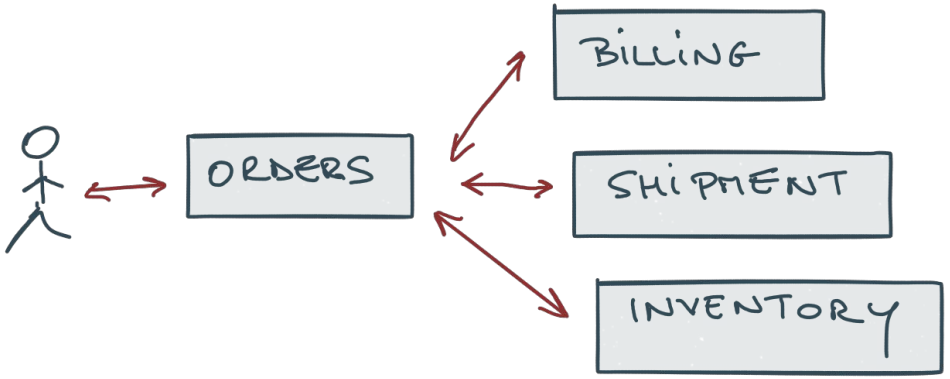
... to agile integration ...



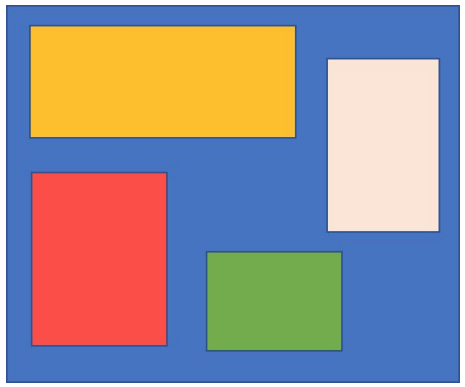
Red Hat

... and finally microservices

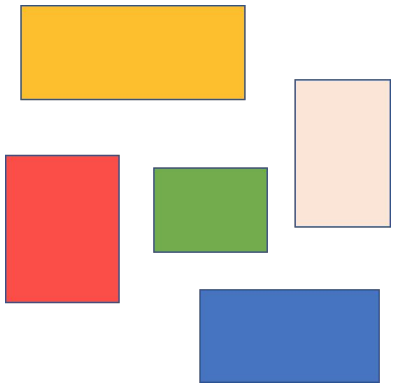




Why microservices?



Monolith



Microservices

Operational efficiency

Fast value delivery

Red Hat

Martin Fowler

01 July 2015

Translations: Japanese · Korean

Microservices provide benefits…

- Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.

- Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.

- Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.

…but come with costs

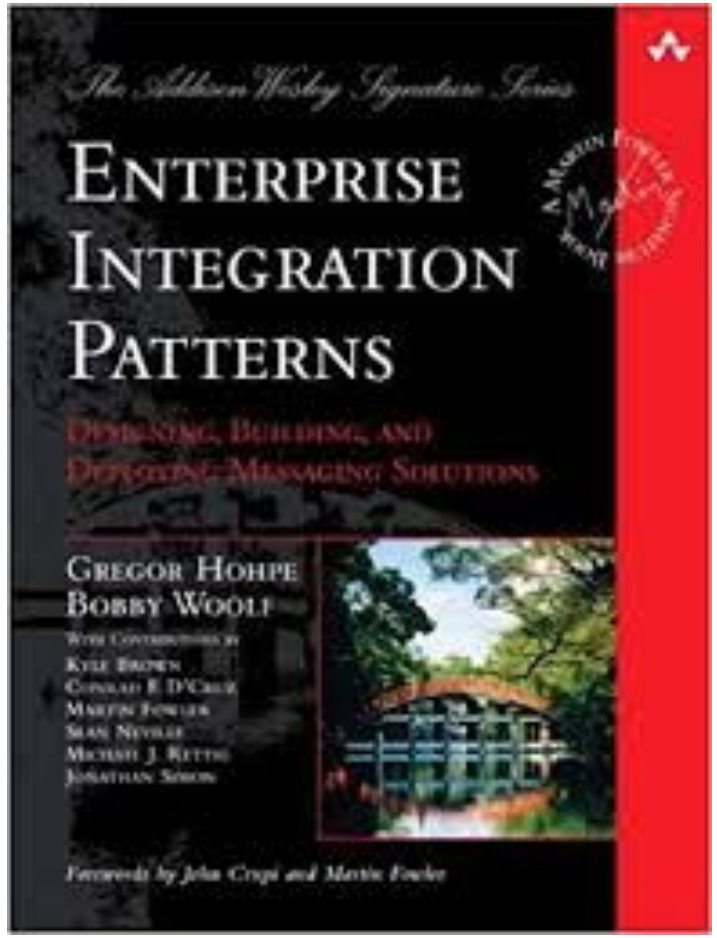
- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.

- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.

- Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

https://martinfowler.com/articles/microservice-trade-offs.html

Adopting **microservices** means dealing with the **inherent complexity** of **distributed systems**



Gary Olliffe
@garyolliffe

Following

Replying to @n0ther_guy @ghohpe @bobby_woolf

So true... great book, it's my patterns "bible"...in 2003 the name was right, we used these patterns to integrate systems... in 2019 we use them to compose distributed systems.

4:25 PM - 18 Mar 2019

Gregor
@ghohpe

Following

Replying to @n0ther_guy @mariusbogoevici and 2 others

Orchestration is an important part of integration. The #eaipatterns book is really about async messaging whereas orchestration is generally stateful. One day, that be the second volume. Work in progress here, incl. description of difference here: enterpriseintegrationpatterns.com/patterns/conve…

11:19 PM - 2 Apr 2019

Red Hat

Today's focus: how **microservices** run and **communicate**

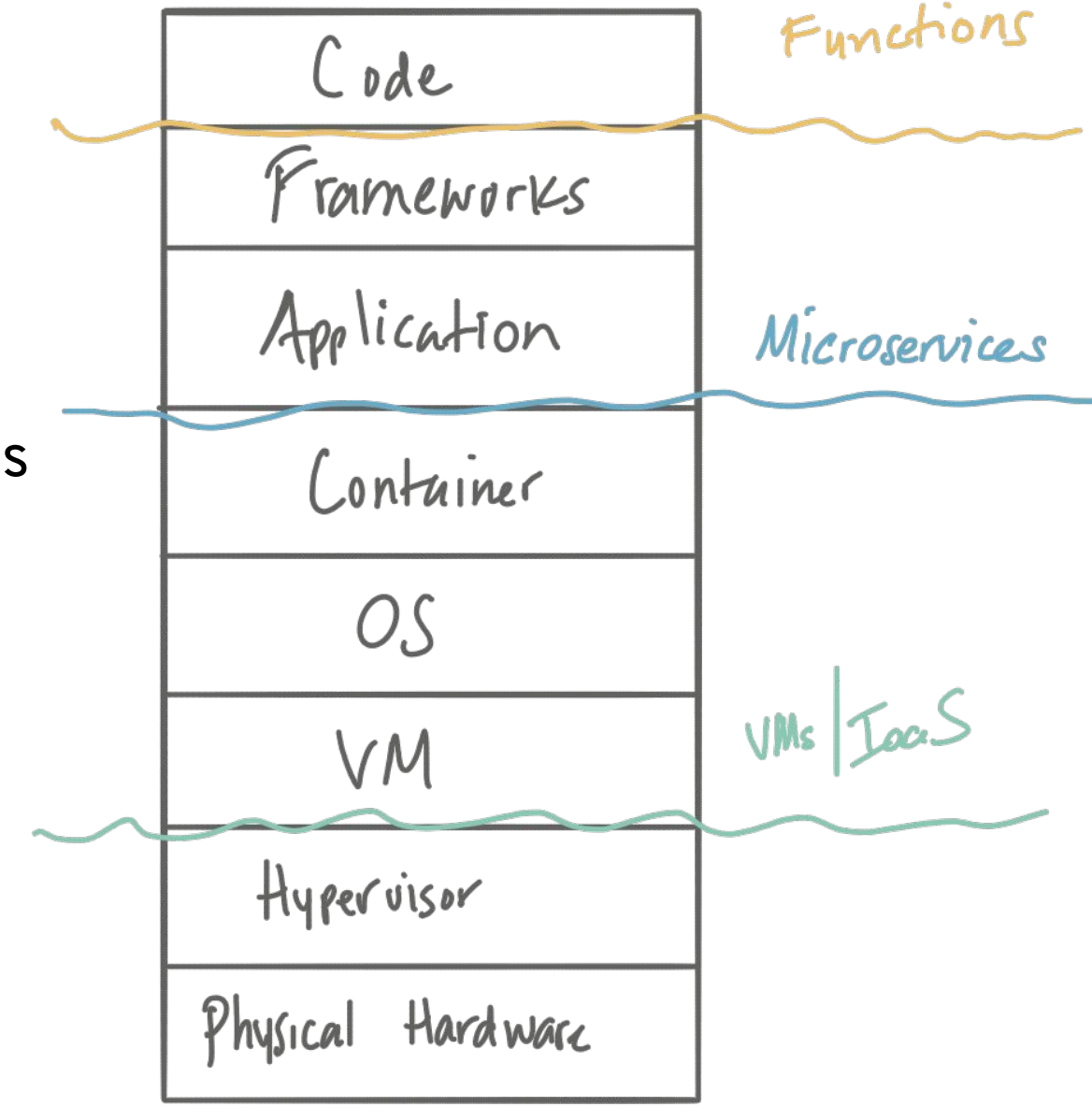
There are other challenges: **security, observability**, etc.

Today's focus: how **microservices** run and **communicate**

Running microservices at scale

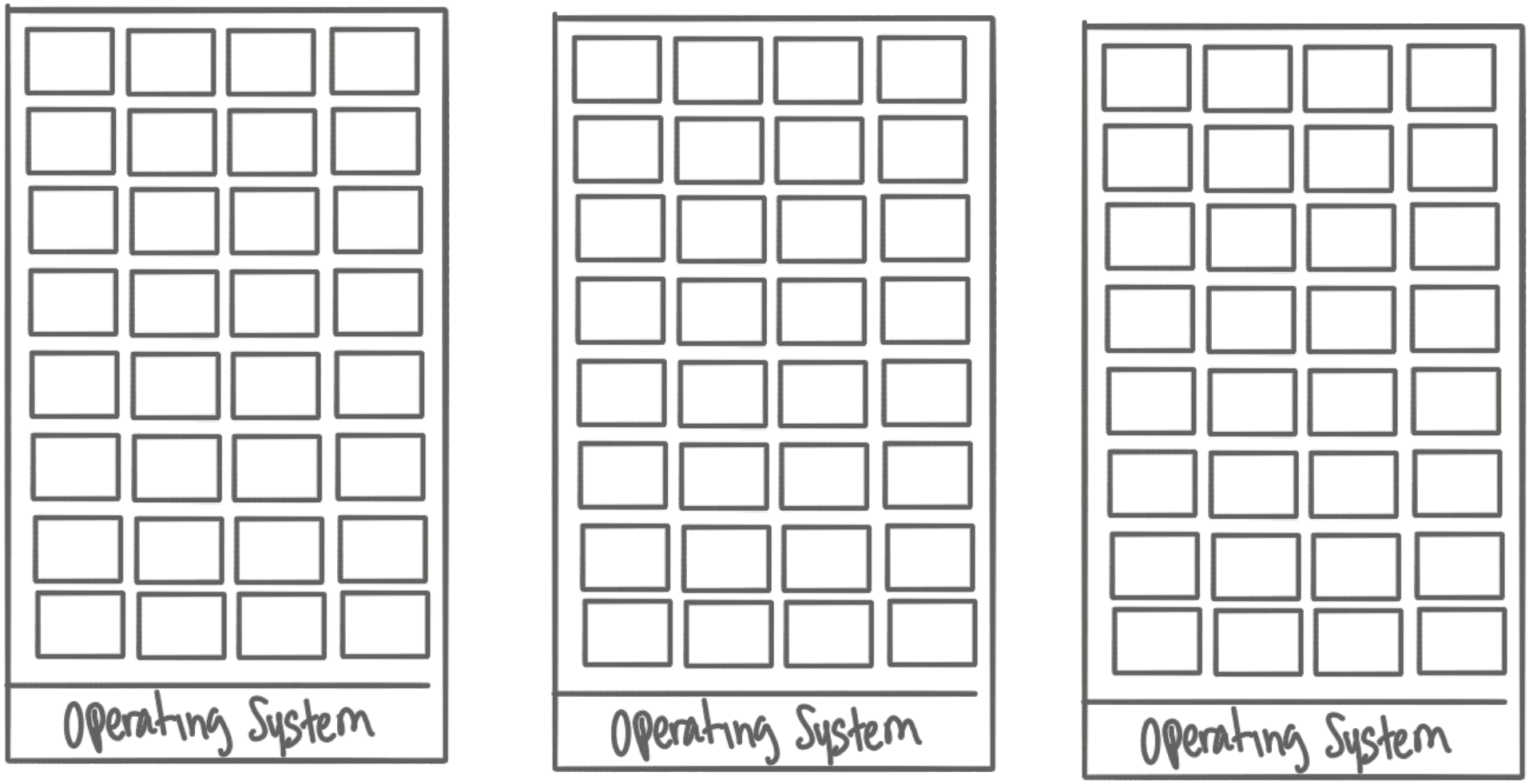
Containerization

- Reduce overhead in running services
- Higher density/utilization gains
- Portable across deployment platforms
- Rich ecosystem (see Kubernetes!)
- Shorter-lived



Red Hat

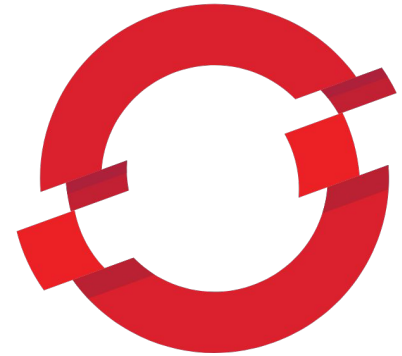
Microservices in containers:
Increasing agility, isolation, utilization



Red Hat

Orchestrating containers on cloud native platforms

- Use a **platform** that makes running apps reliable, transparent and boring
- In-built resource management
  - Memory, CPU, disk
- Elastic scaling
- Monitoring and failover
  - Health, logging, metrics
- Routing and load balancing
- Rolling upgrades and CI/CD
- Namespacing



OPENSHIFT

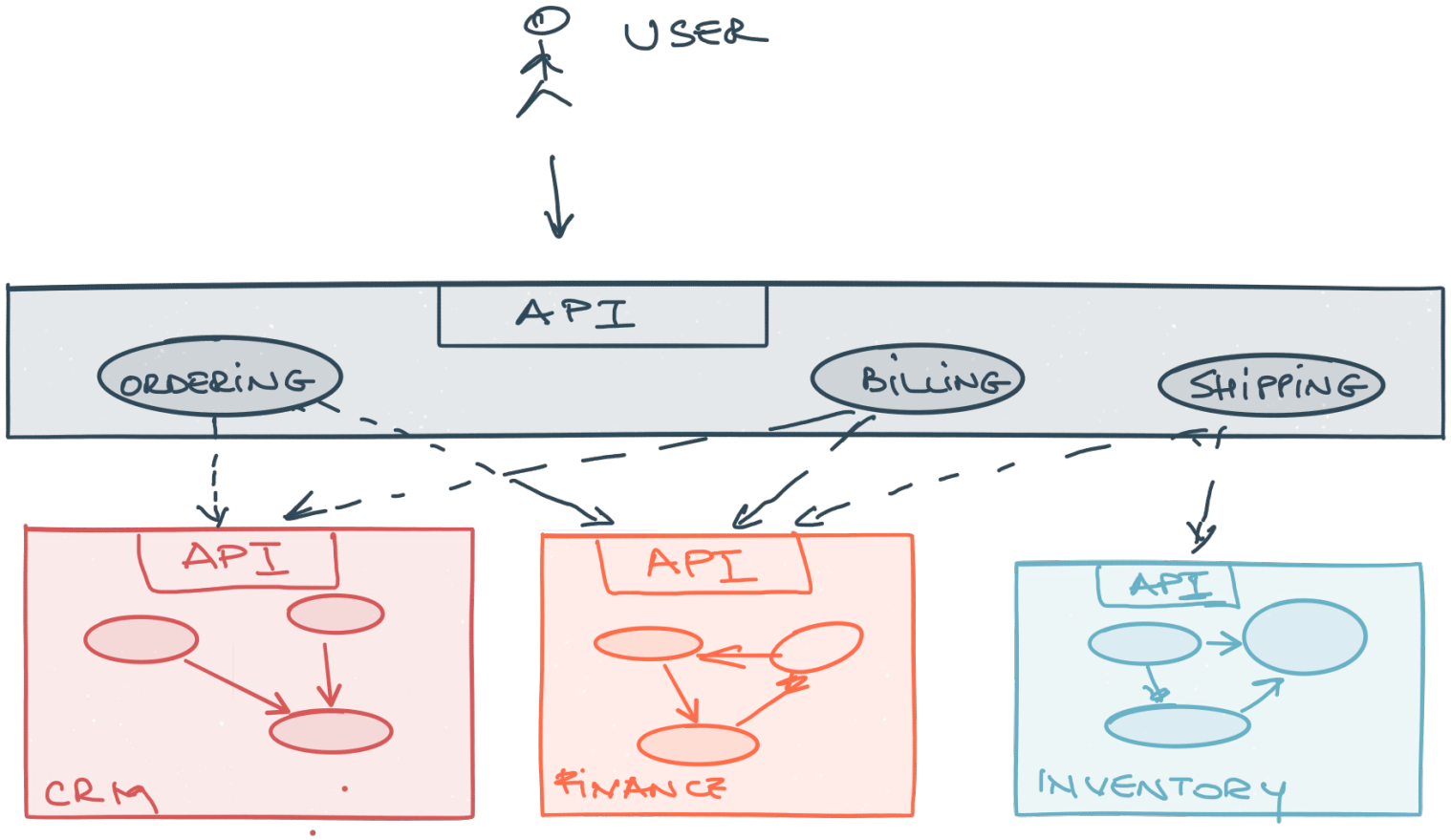
Red Hat

Insight: focus on **delegating to the platform** all responsibilities that are **not intrinsic to the application**

Integrating microservices: how they **communicate**

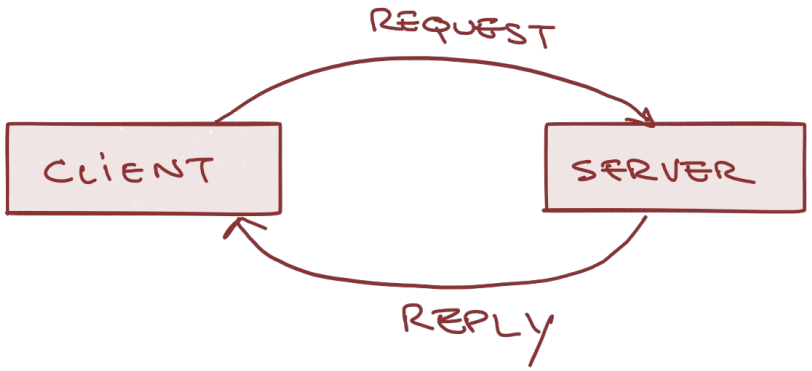
System-level integration via APIs:

- bounded contexts and ubiquitous language

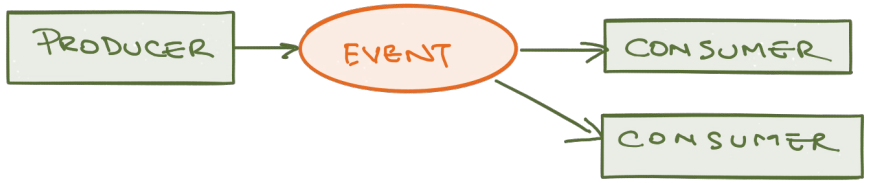


Red Hat

Request-reply vs. event-driven



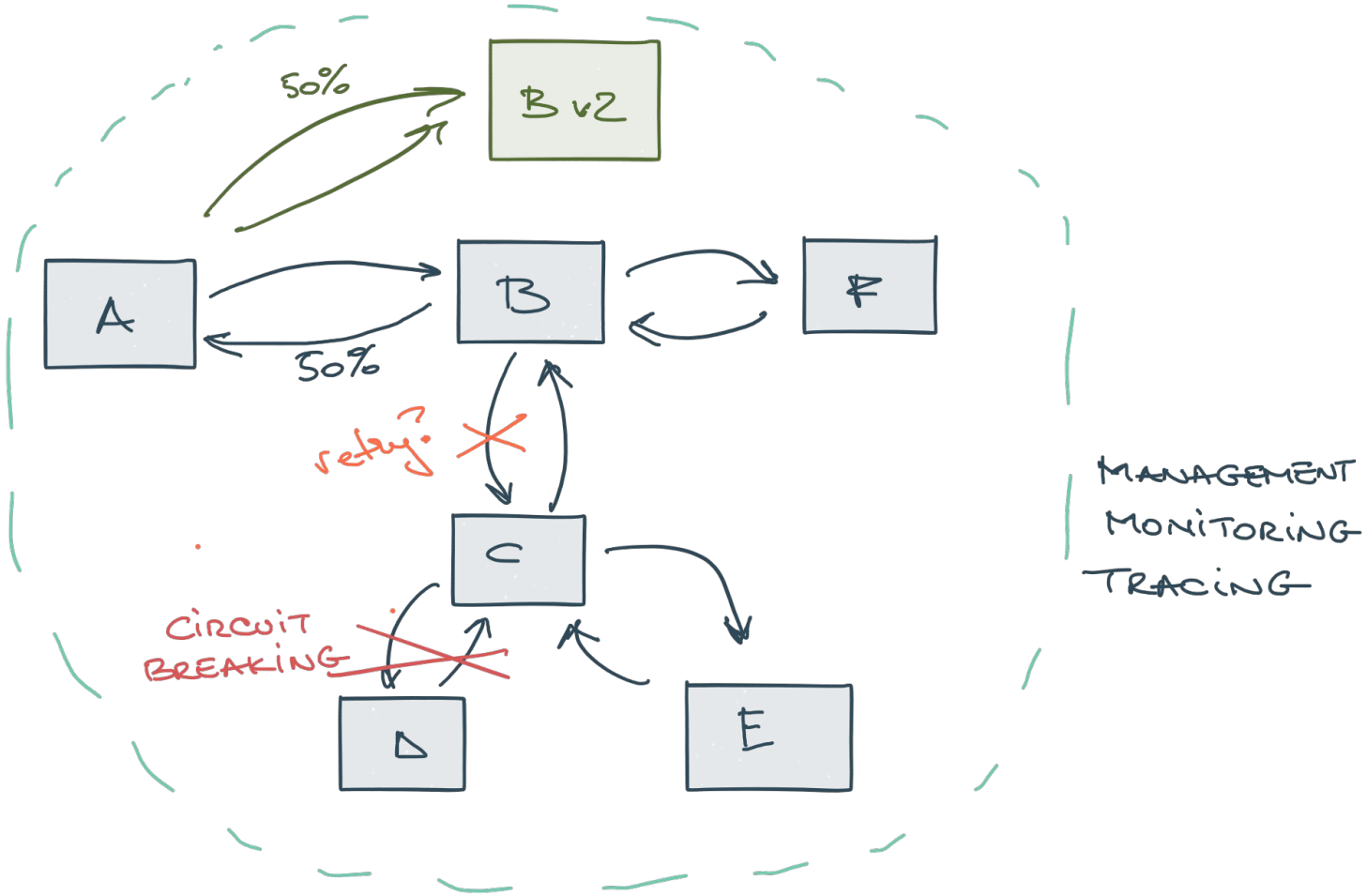
Synchronous & ephemeral
Low composability
Simplified model
Low tolerance to failure
Best practices evolved as REST



Asynchronous and persistent
Decoupled
Highly composable
Complex model
High tolerance to failure
Best practices are still evolving

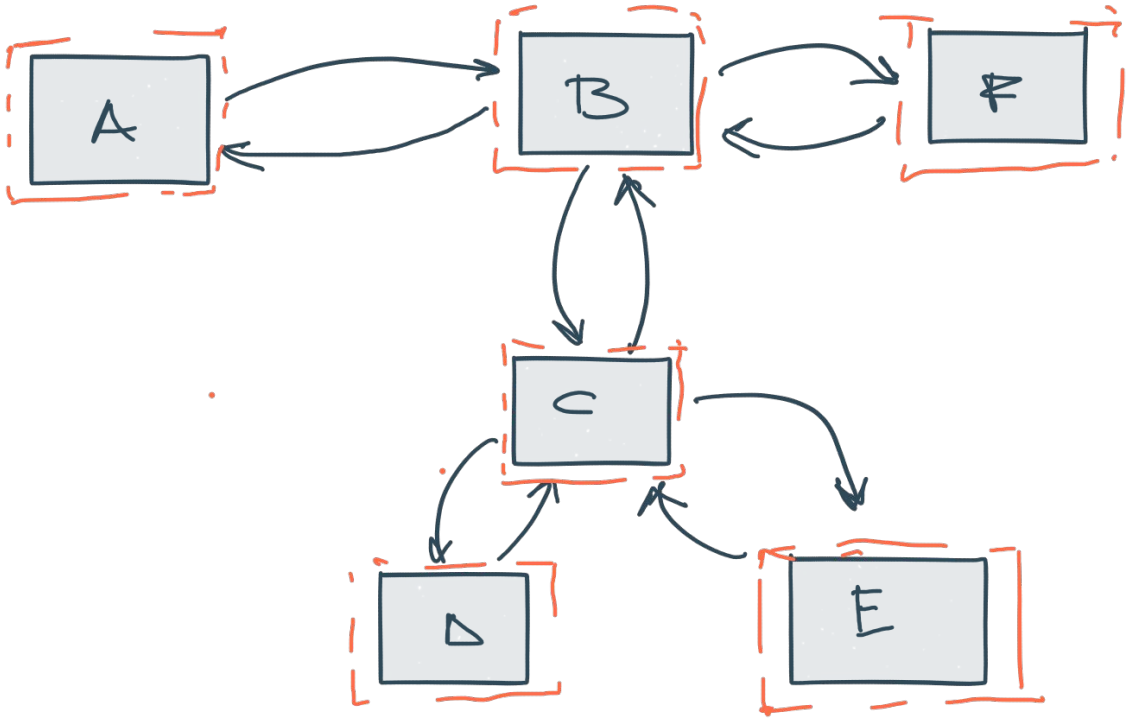
Modern RESTful integration with Service Mesh

How simple are RESTful microservices?



Red Hat

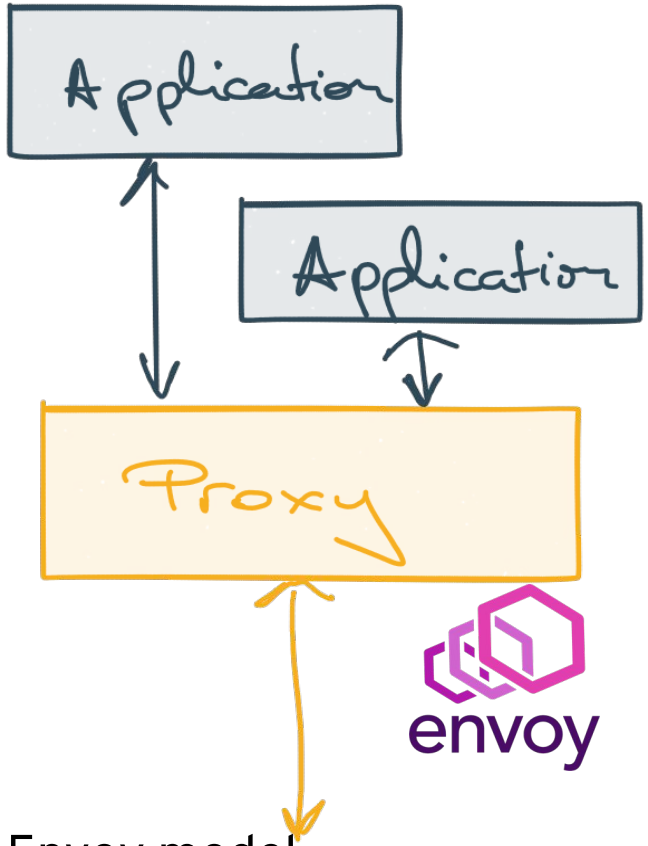
Solution: enhancing applications



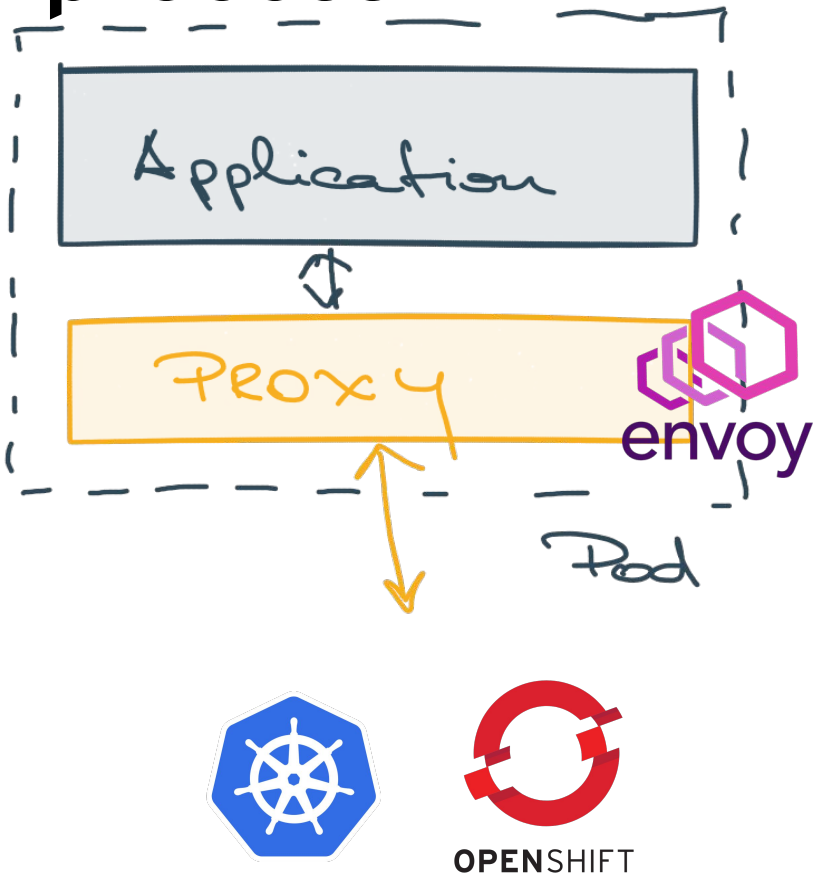
Red Hat

Enhancing apps: in-process/out-of-process

Enhance the application itself
Drawback: model does not extend outside Spring/Java
Each application needs to be enhanced individually



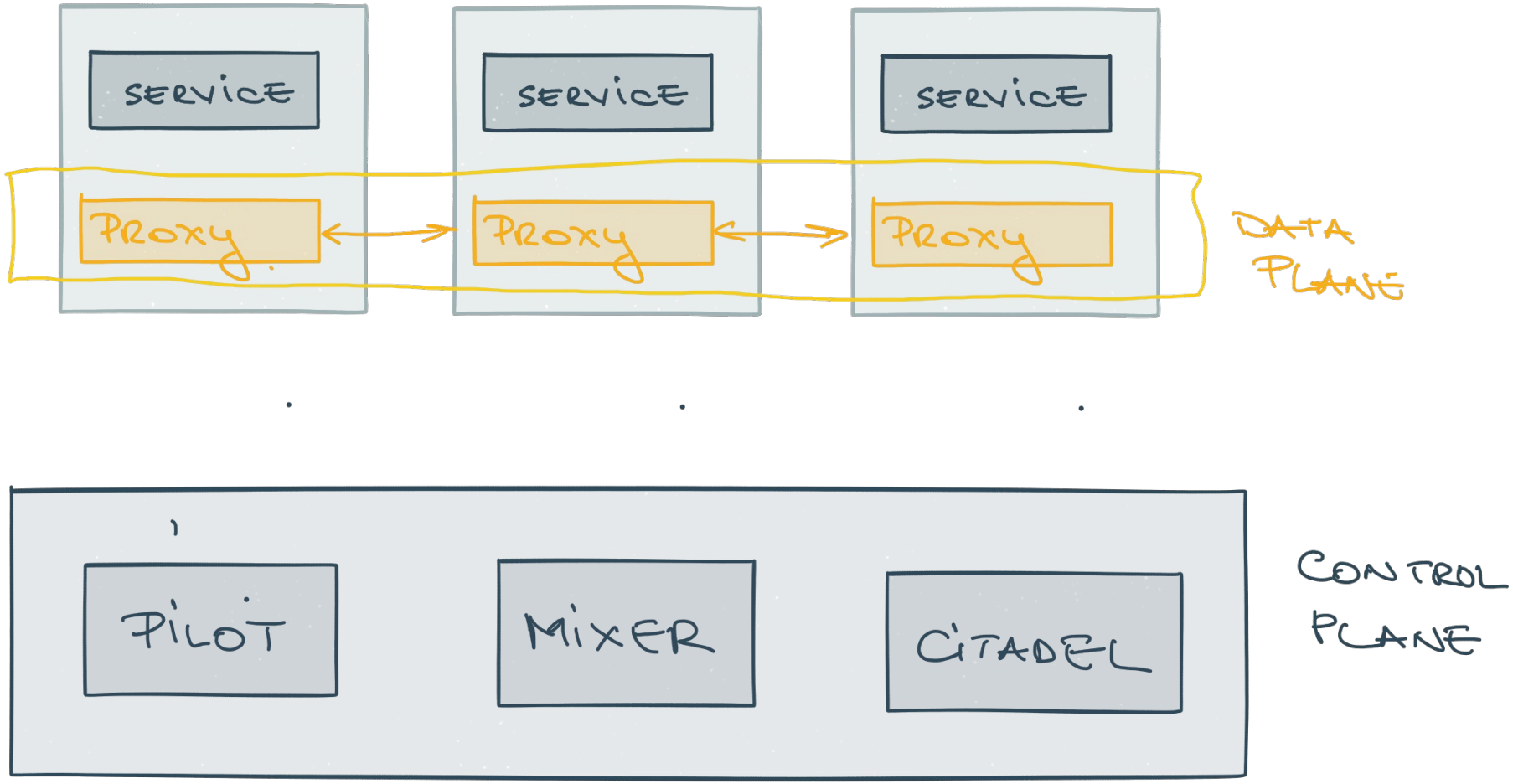
Envoy model
Out-of process binary proxy for HTTP/HTTP2
Polyglot support, app agnostic



Pod injection
Polyglot support, app agnostic

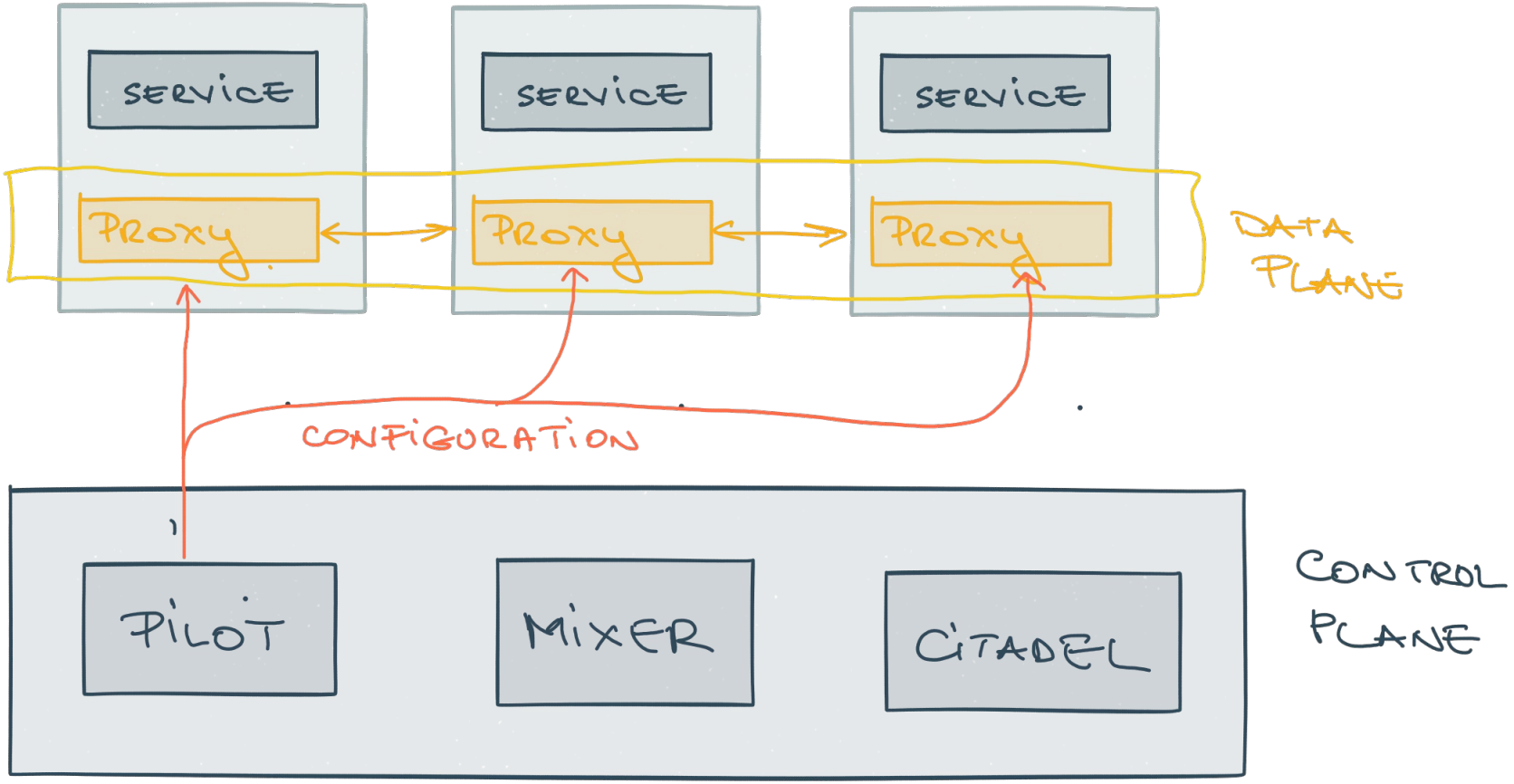
Red Hat

Istio

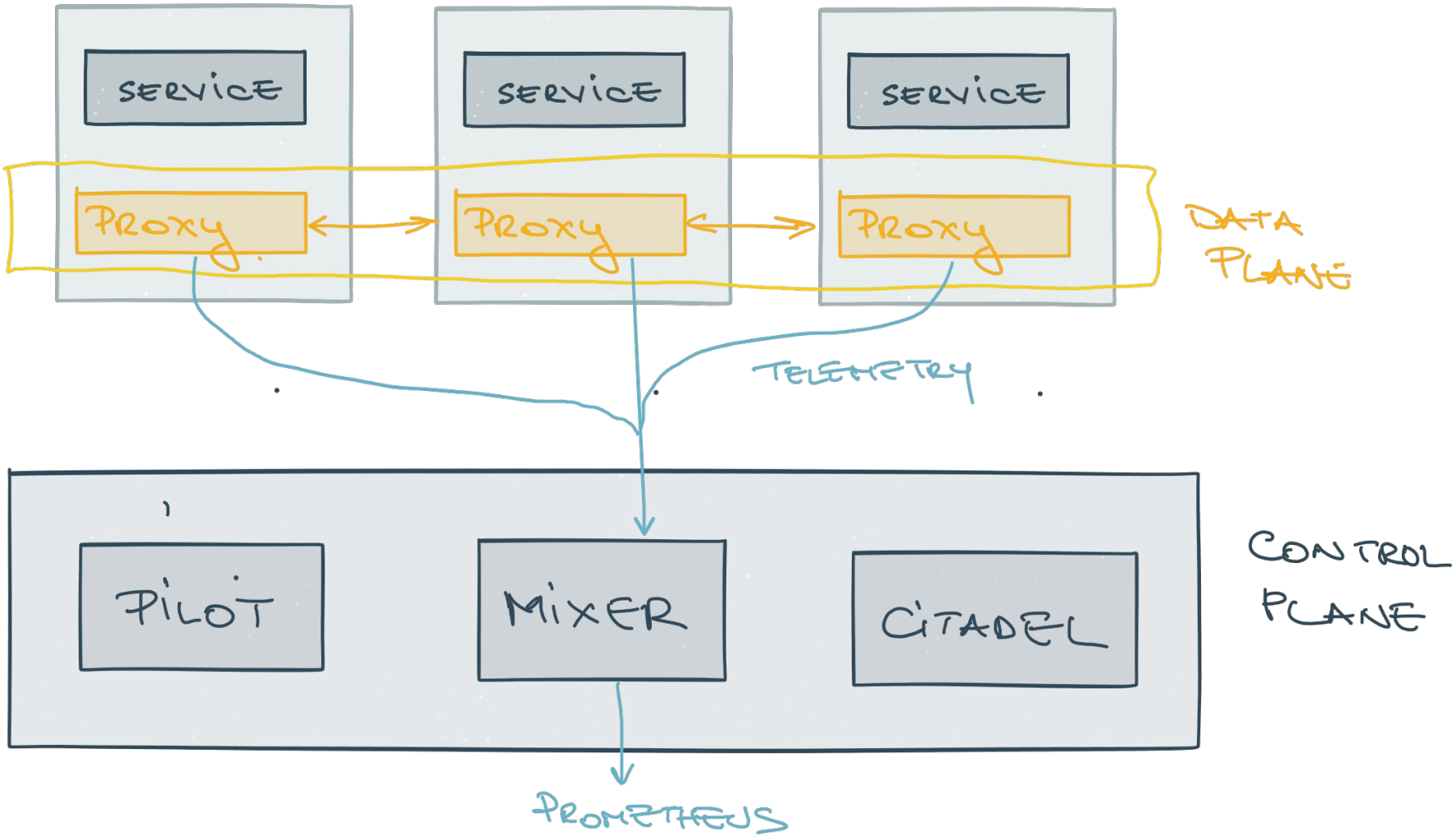


Red Hat

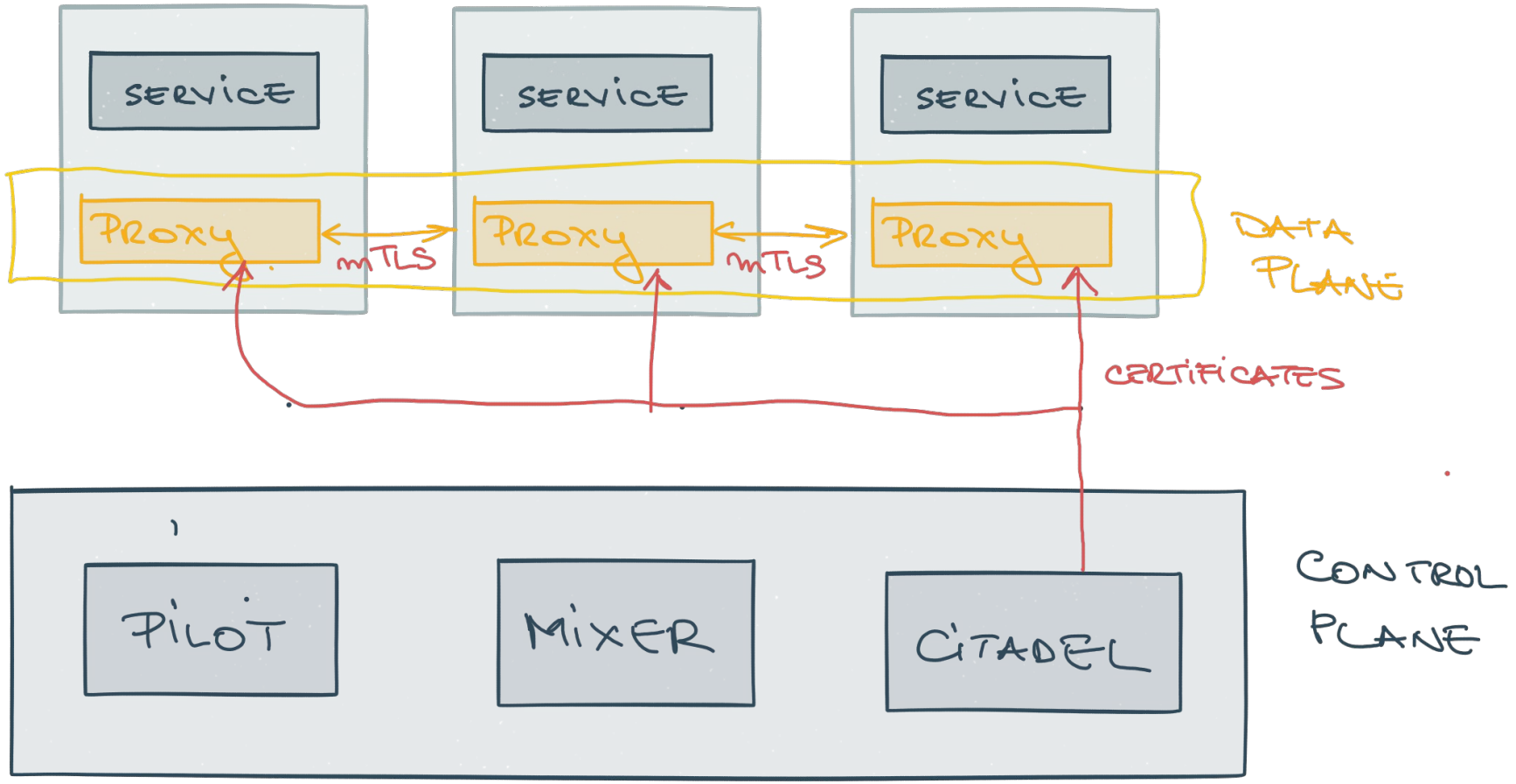
Istio: Pilot



Istio: Mixer

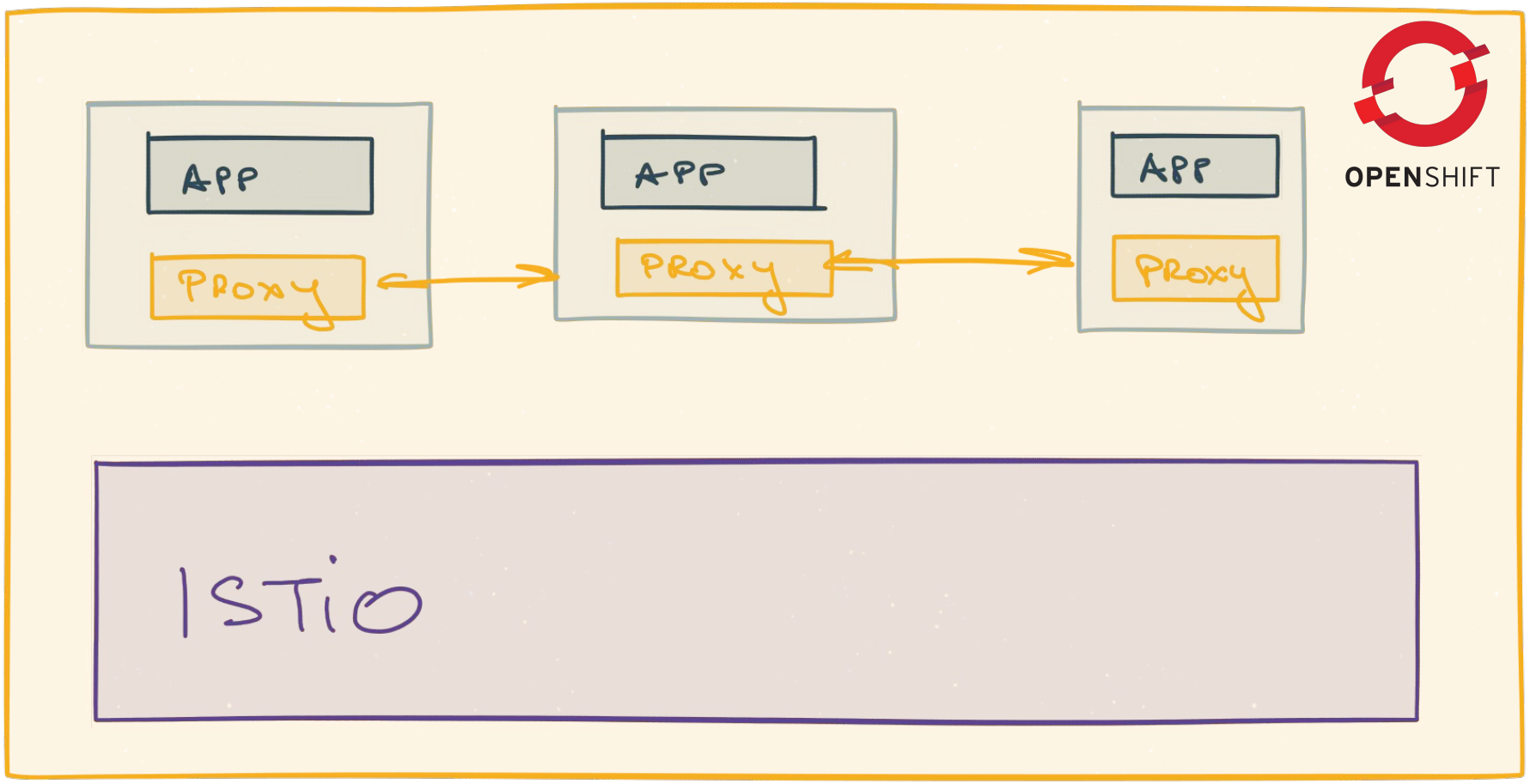


Istio: Citadel



Red Hat

Moving complexity into the platform



Red Hat

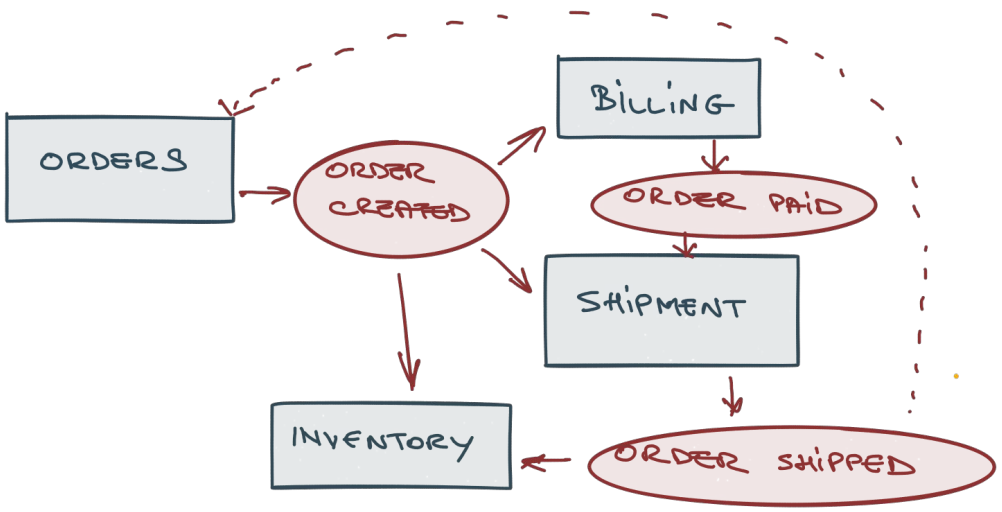
Event-driven and streaming architectures

What is an event?

- Action or occurrence, something that happened in the past
  - ‘Order created’, ‘user logged in’, ‘
- Event characteristics:
  - Immutable
  - Optionally persistent
  - Shareable
- Event types: [1]
  - Notification
  - State Transfer (Command)
  - Event-Sourcing/CQRS

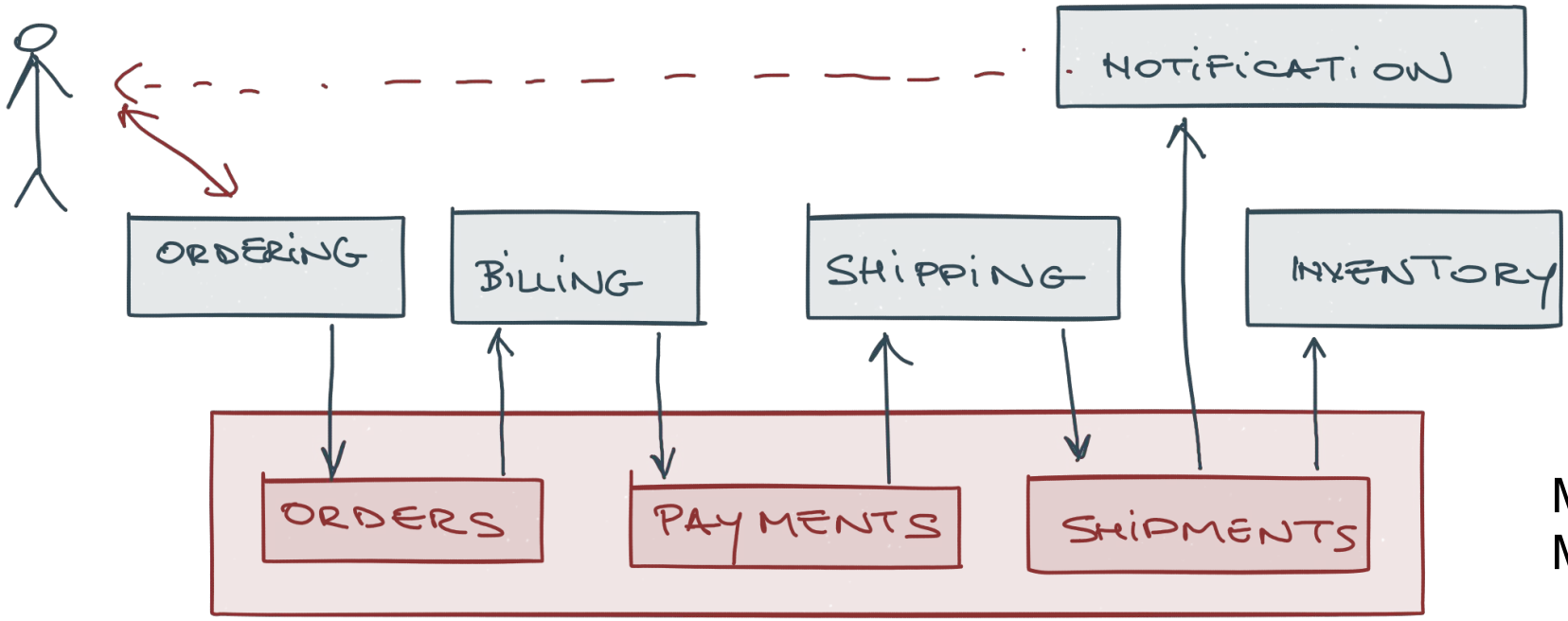
[1] https://martinfowler.com/articles/201701-event-driven.html

Designing systems with events



- EDA: event-centric approach in system design
  - Treating events as part of your domain model
  - Designing components as event handlers and emitters
- EDA is aligned with the goals of domain-driven design
  - Enforce isolation and decoupling between bounded contexts
  - Properly designed events can create an expressive ubiquitous language
- EDA creates highly observable and extensible systems
- Event storming: events-first design

Event-driven microservices



Applications

Messaging Middleware

Events in the digital business

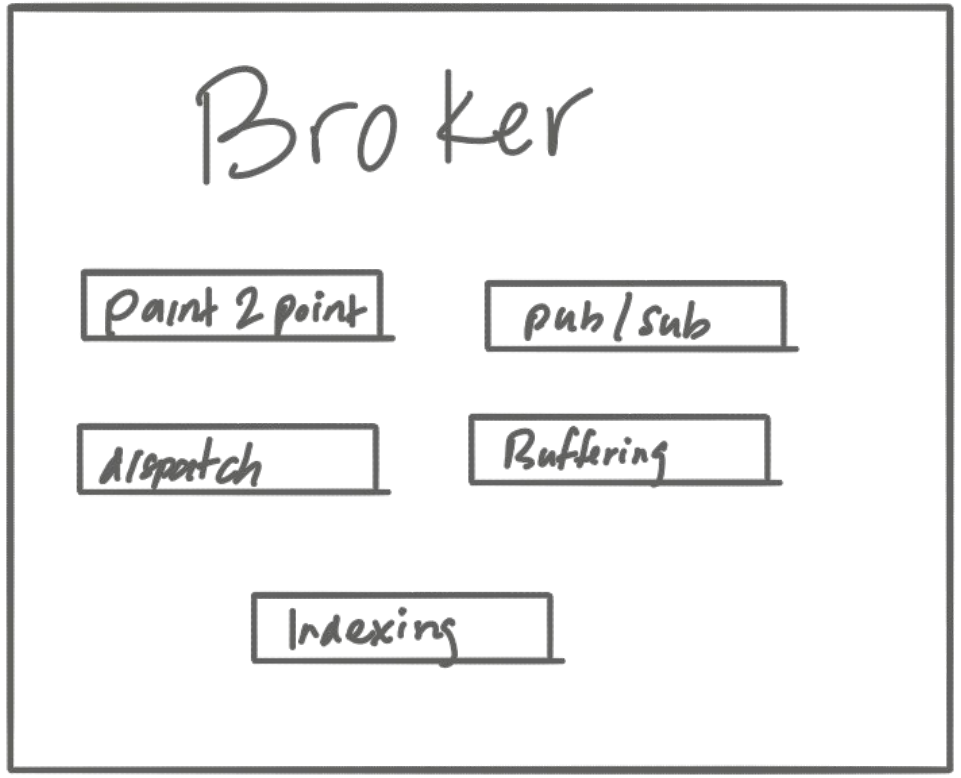
- We live in an event-driven world (literally), and that impacts how we do business
- Next-generation digital business is about agility and experimentation
  - Shifting focus from analyzing the status quo to understanding the change in progress
  - Blurring the distinction between events and data
  - Architectural focus shifting from data-centric to event-driven
- Increased importance of bottom-up approaches in business event design
  - Complex event processing driven by experimentation, analytics, machine learning
  - Emphasis on readiness to observe and collect events before ascribing them a business meaning

Event-driven architectures reduce friction

- From a technical standpoint:
  - Building robust and resilient distributed architectures
- From a development process standpoint
  - High composability encourage **agility** and **experimentation**
- From a business standpoint:
  - Aligning digital business with the real world

Evolution of messaging infrastructure: from traditional brokers …

- Publish subscribe semantics (vs queuing)
- Subscribers receive events at their own pace
- High utilization of consumers, regardless of event publish
- Persistent vs non-persistent
- Example: ActiveMQ, RabbitMQ, etc



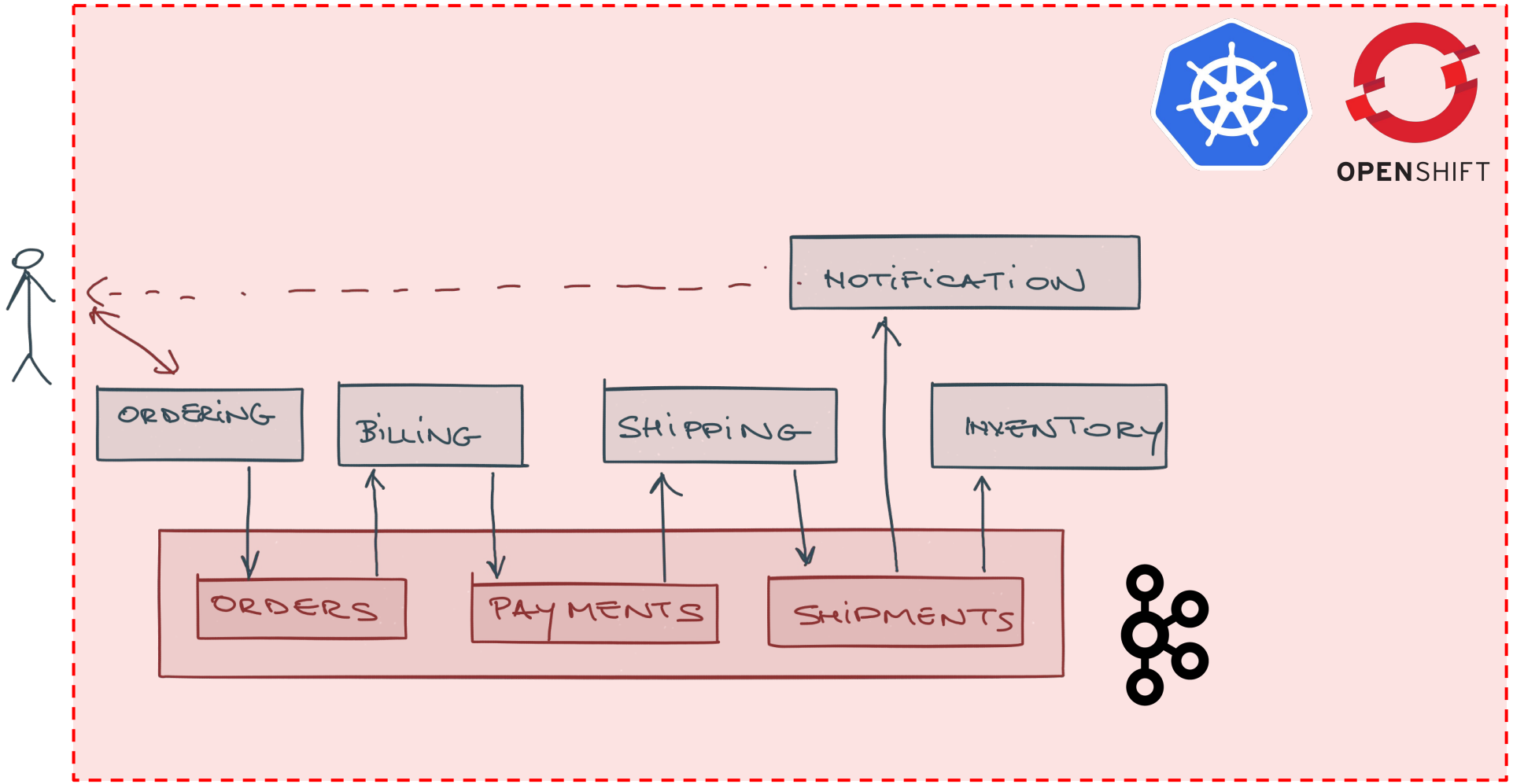
… to streaming

- Decentralized processing
- Move indexing and bookkeeping to consumers
- Make fundamental data structure first class citizen (log data structure)
- Replication and failover part of the protocol
- Example: Apache Kafka, Kinesis, etc

Traditional messaging vs streaming

Traditional messaging	Log/Streaming Systems
• Advantage in: individual message exchanges (transactionality, acknowledgment, error handling/DLQs), P2P/competing consumer support • Publish-subscribe support with limitations) • No replay support	• Advantage in: long-term persistence, replay, semantic partitioning, large publisher/subscriber imbalances, replay and late-coming subscribers • Weak support for individual message acknowledgment, p2p/competing consumers

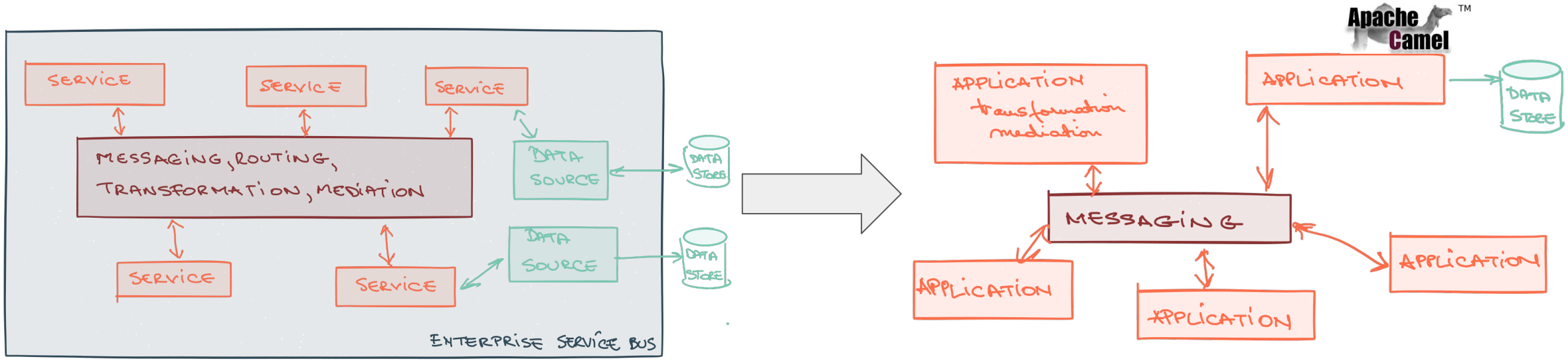
Messaging: utility service & event streams



Red Hat

Agile Integration with Microservices, Containers and Events

From ESBs to agile integration

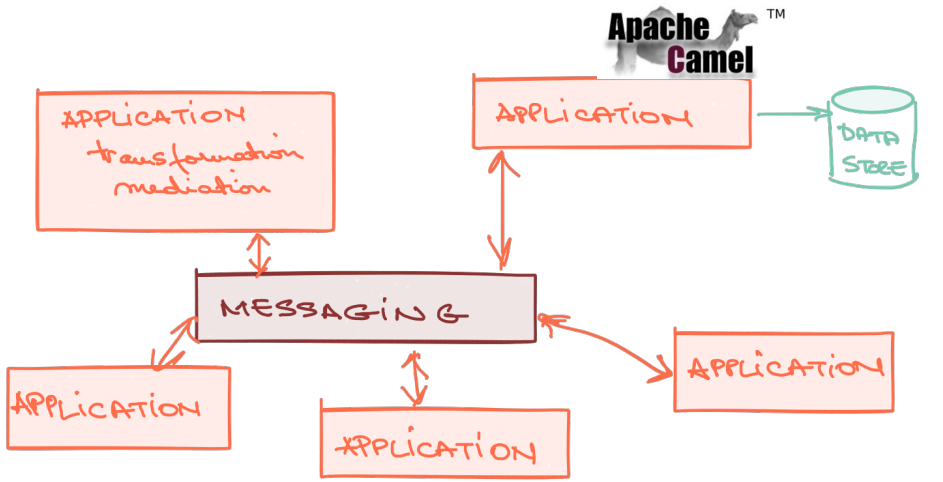


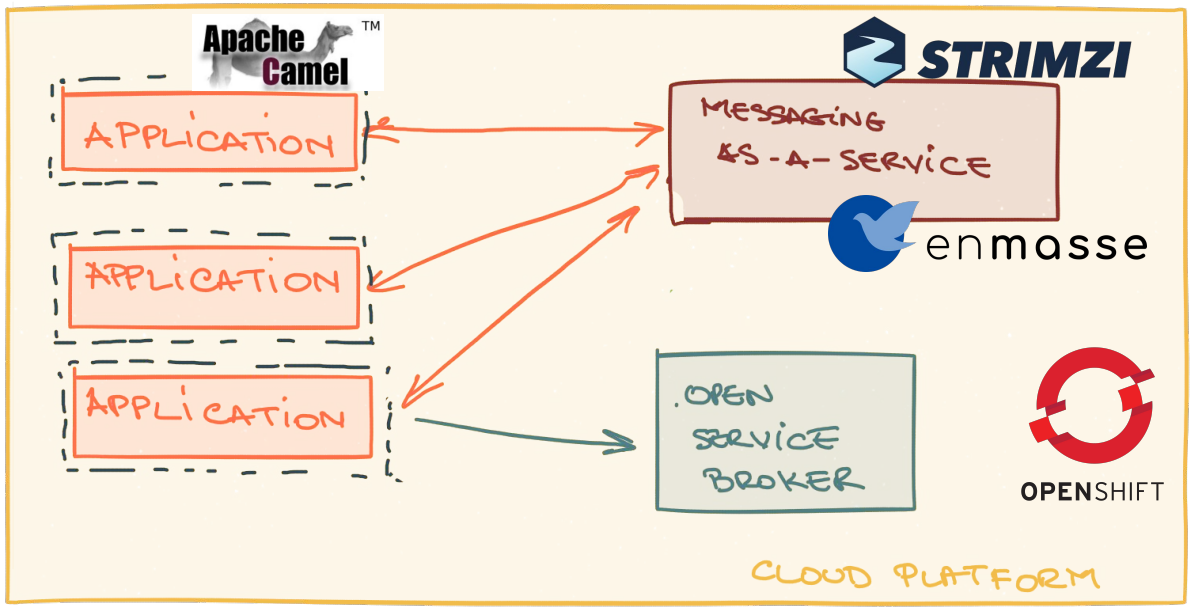
Optimized for utilization
Centralized, tightly coupled
Mixing logic with infrastructure

Optimized for agility
Decentralized, decoupled
Separate messaging middleware from logic

Red Hat

Modern enterprise integration: agile, decentralized, cloud-native





Optimized for agility
Decentralized, decoupled
Separate messaging middleware from logic

Preserves benefits of agility while optimizing resource utilization
Clear separation of concerns between compute and data infrastructure and application logic

Red Hat

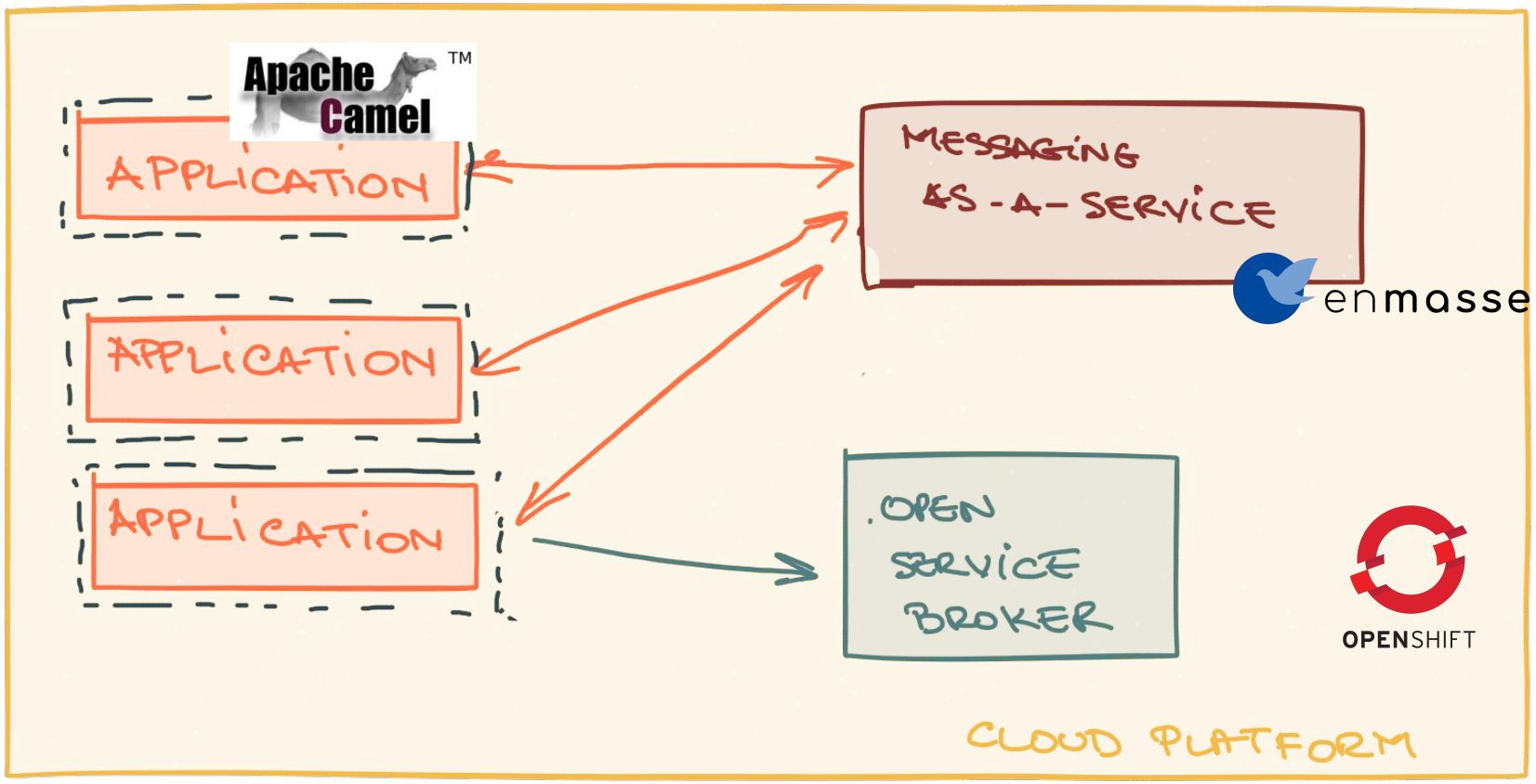
Enterprise integration patterns for microservices

- Originally designed for building integrated solutions out of siloed enterprise systems
- Applicable to general-purpose event-driven interaction
- Very well suited for building event-oriented distributed systems (aka event-driven microservices) - e.g. with Apache Camel

Enterprise Integration and Streaming

- Perpetual data and event “streams” as a first class citizen
- Data in aggregate vs individual messages
- Small services working together to interpret large numbers of streams
- Data in perpetual motion
- Eventual consistency as data synchronization pattern
- Examples: Apache Camel, Kafka Streams, stream-processing frameworks

Modern enterprise integration:
agile, decentralized, cloud-native



Red Hat

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat