Reactive Microservices

Lutz Huehnken
@lutzhuehnken

http://www.reactivesystems.eu

Traditional application architectures and platforms are obsolete.

Anne Thomas, Gartner

Modernizing Application Architecture and Infrastructure Primer for 2016.

January 29, 2016

Why talk about Lagom?

Just one framework, but quite inspiring.
A „gateway drug“ to

- **Reactive Systems**
- **Event Sourcing & CQRS**
- **Domain-Driven Design**

Microservices

The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process.

Martin Fowler

Microservice Trade-offs

Again, according to Mr. Martin Fowler

- Distribution
- Eventual Consistency
- Operational Complexity

Microservice Trade-offs

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

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

Microservice Trade-offs

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

All this hype about microservices makes me sad. And not about the concept, but about the name. As I wrote before, “micro” in “microservices” means absolutely nothing. What is worse, it confuses everybody. Again and again I see people focusing on “micro” and falling into nanoservices trap.

Eugene Kalenkovich

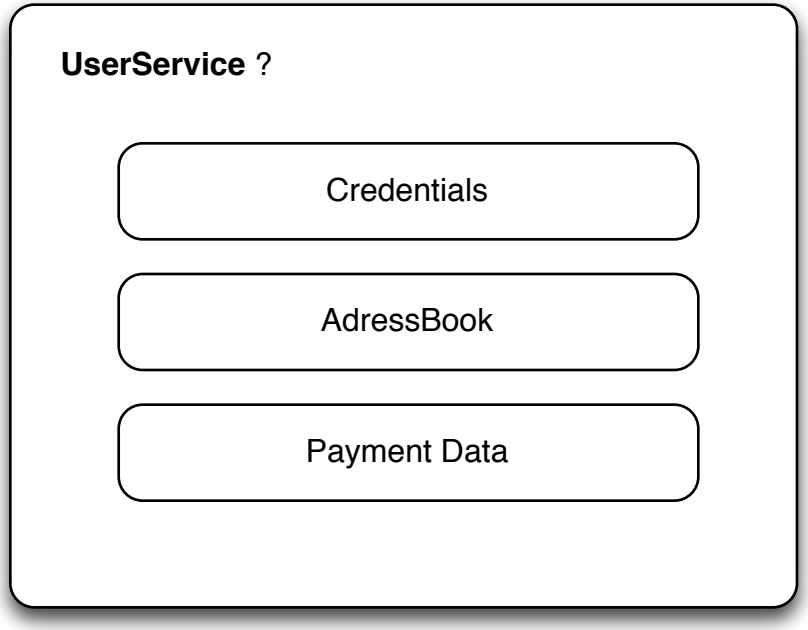
Lagom (pronounced ['lɑ̀ːgɔm]) is a Swedish word meaning "just the right amount". The Lexin Swedish-English dictionary defines lagom as "enough, sufficient, adequate, just right".

https://en.wikipedia.org/wiki/Lagom

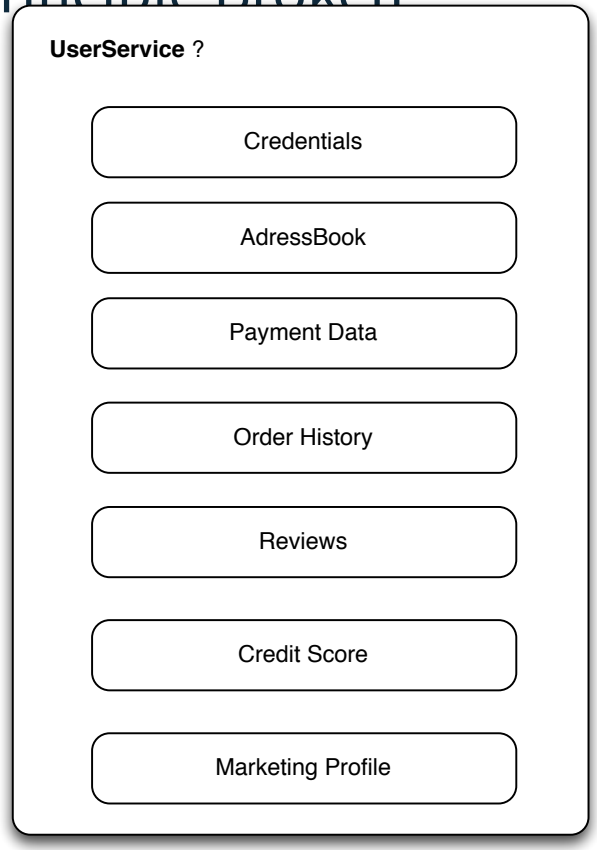
The ideas in Eric Evan's Domain-Driven Design are very useful to us in finding sensible boundaries for our services.

Sam Newman, „Building Microservices“, p. 38

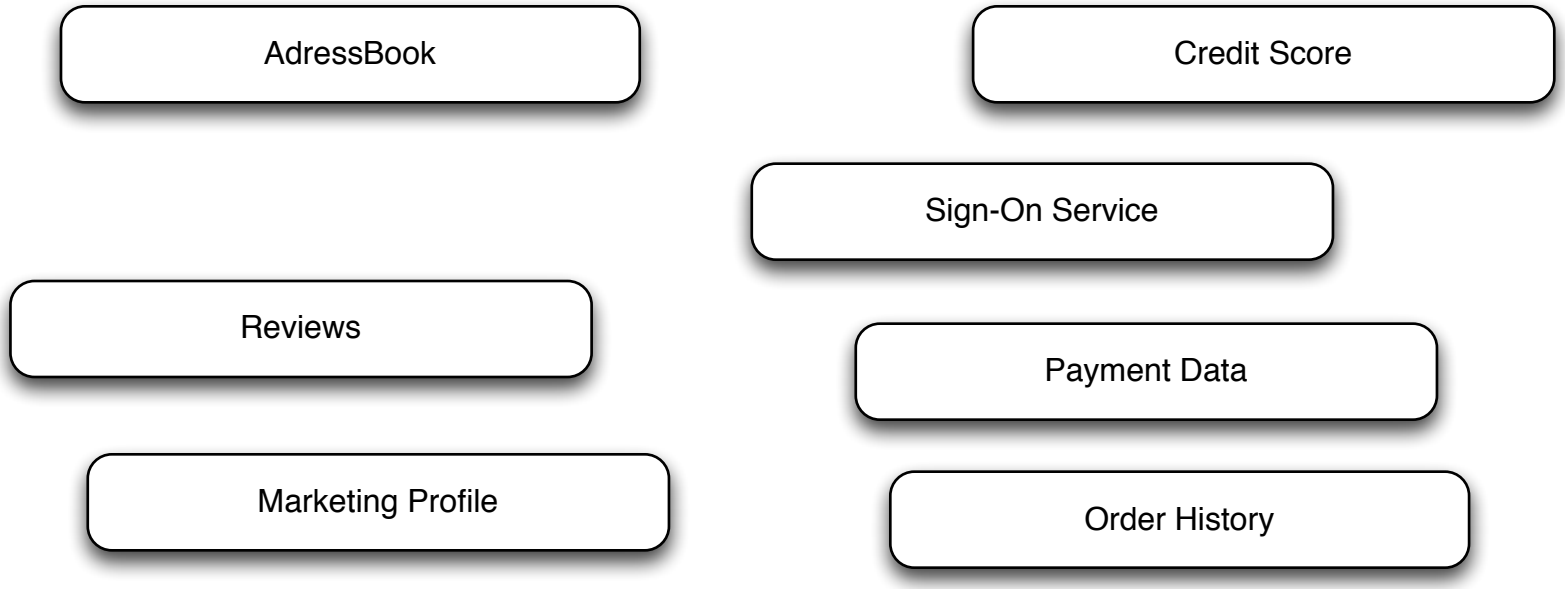
Service Boundaries



Single Responsibility Principle Broken



Bounded Contexts



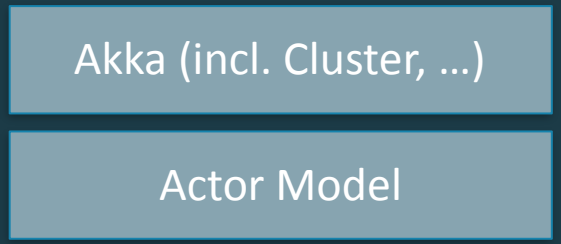
Different Levels of Abstraction

Actor Model

- Everything is an actor
- Each actor has an address
- When an actor handles a message, it can
  - create new actors
  - send messages to other actors
  - change the behavior for handling the next message

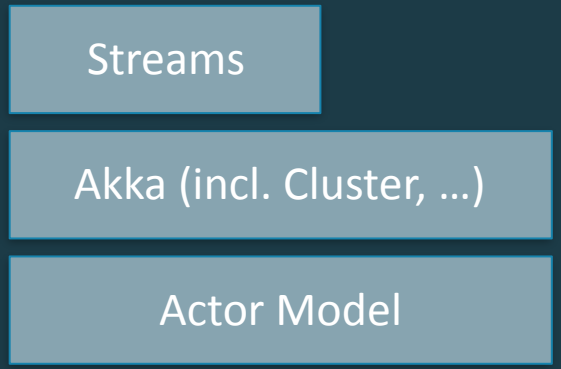
Akka

Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM.



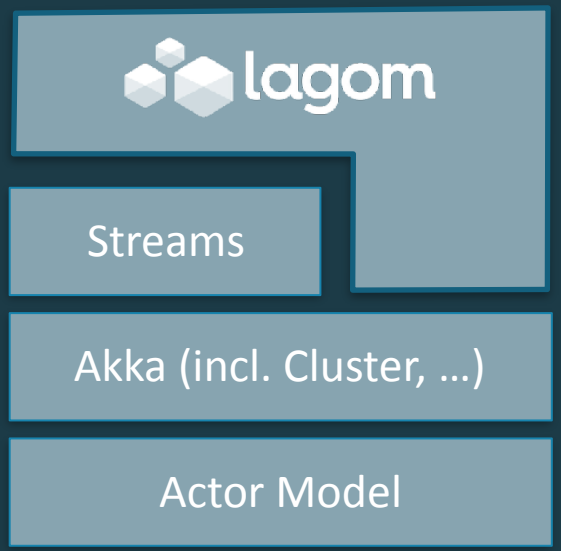
Akka Streams

An intuitive and safe way to formulate stream processing setups such that we can then execute them efficiently and with bounded resource usage.



Lagom

A framework for creating microservice-based systems



Example: The Service API

„Service“ as a first-class concept

- Service Descriptors abstract from transport
- Services have a typed interface
- Can be treated like objects, can be injected etc.
- Service Client includes Service Lookup, Circuit Breaker

„Service“ as a first-class concept

```
public interface HelloService extends Service {
   ServiceCall<NotUsed, String, String> sayHello();
   ...
}

interface ServiceCall<Id, Request, Response> {
  CompletionStage<Response> invoke(Id id, Request
request);
}
```

Async

Built-in support for asynchronous streaming

```
ServiceCall<NotUsed, Source<String, ?>,
Source<String, ?>> sayHello();
```

Not strict, but a stream
(materialized as WebSocket)

Developer Experience

Development tooling

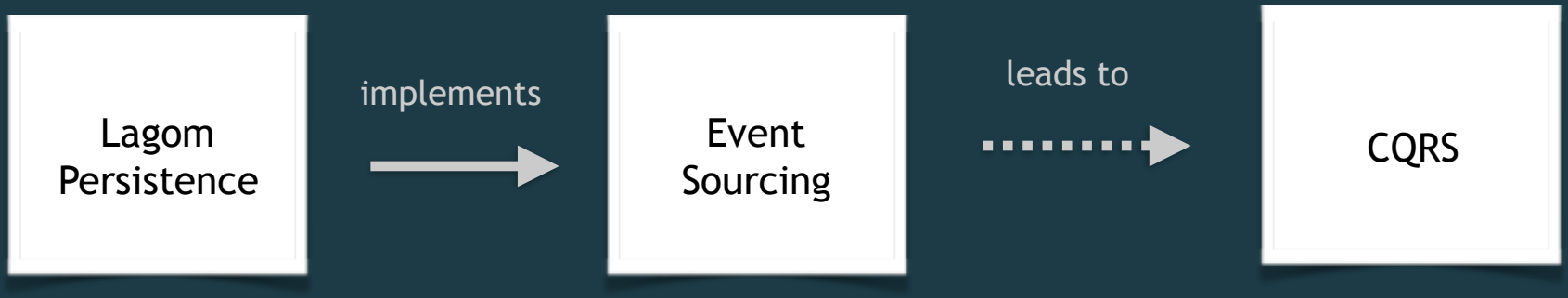
- Run all microservices at once
- Embedded Cassandra DB
- Embedded Kafka Message Bus
- Inter-service communication via service locator
- Hot reloading

Lagom Persistence

A Rail-Guided Approach

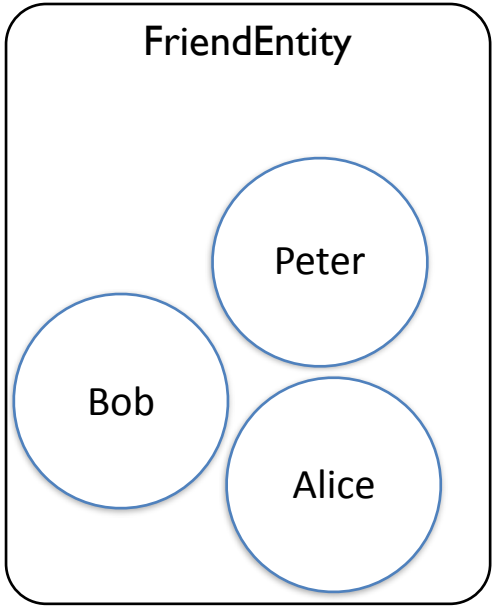
- A guided approach to Event Sourcing and CQRS
- Required structure provides clarity and simplicity
- Cassandra given as default data store
- ReadSideProcessor provided for read side

Lagom Persistence



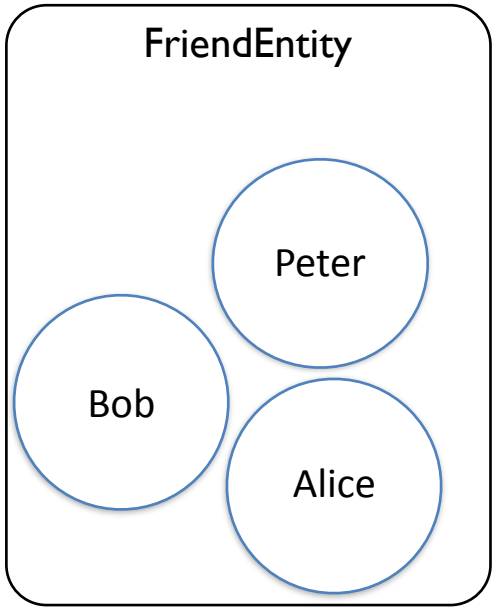
Not covered: More Domain Driven Design, NoSQL Databases

Aggregates



- We implement our **aggregates** as **`PersistentEntities`**
- **`PersistentEntities`** will receive **commands**
- Triggered by a command, **`PersistentEntities`** will change their state
- Example: Add a friend, remove a friend

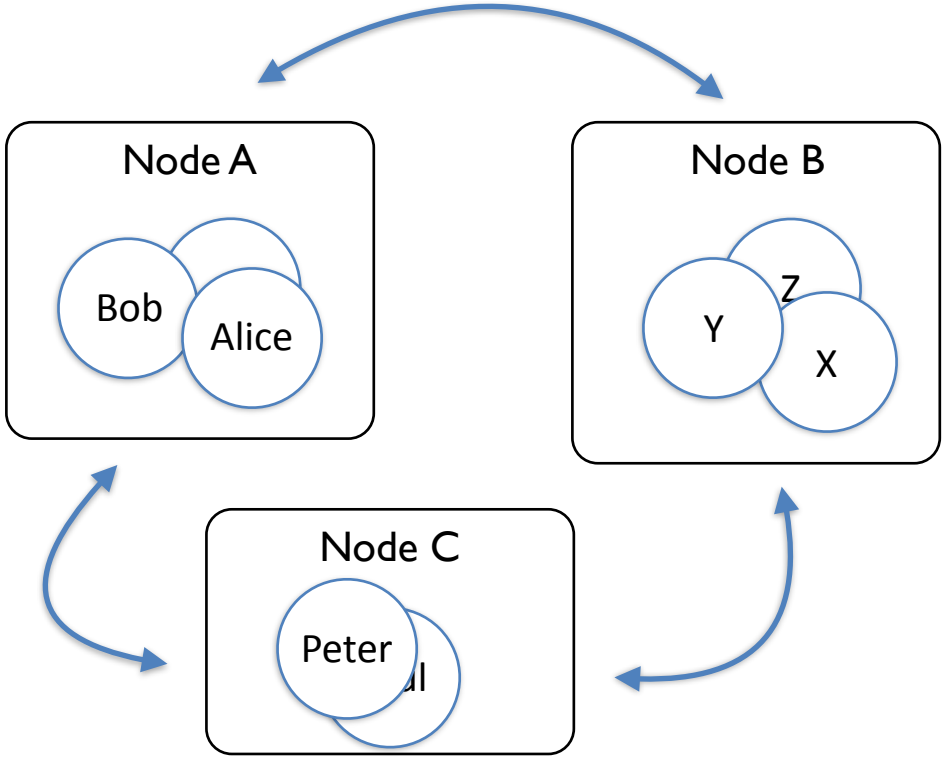
Memory Image



- We keep all* our `PersistentEntities` in memory!
- We have now moved from a CRUD approach to a Memory Image approach
- See http://martinfowler.com/bliki/MemoryImage.html

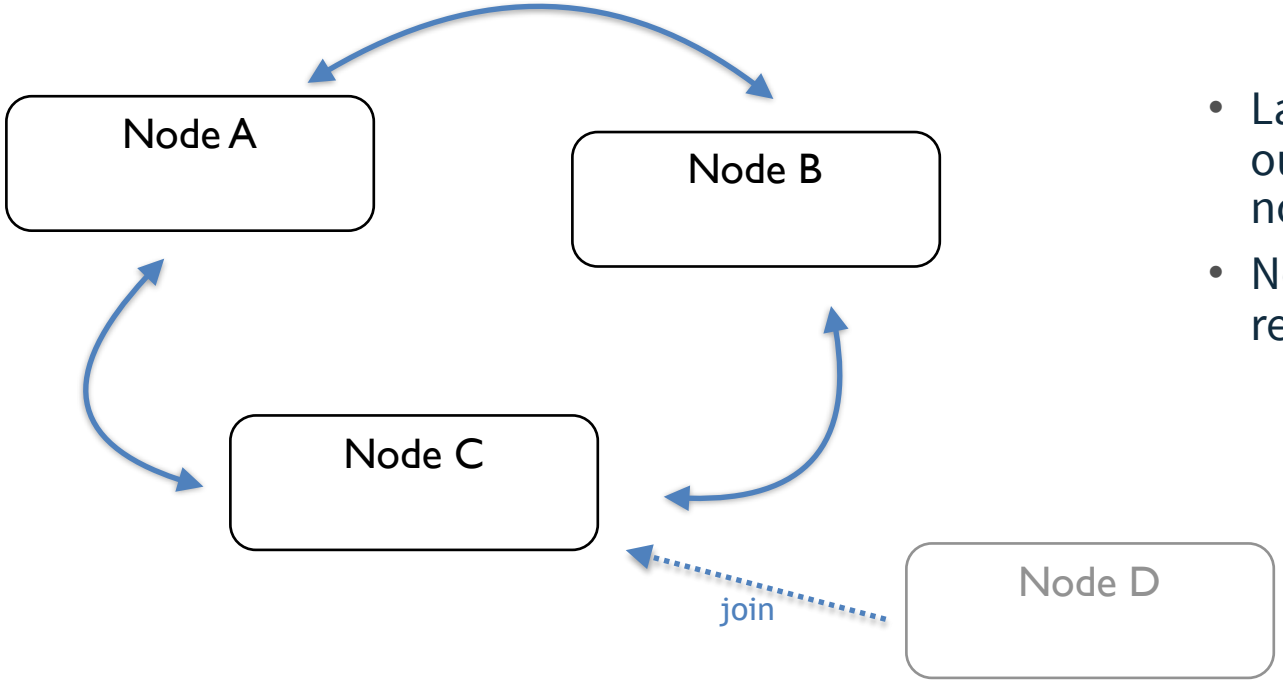
(*) or a working set, entities will be passivated and activated as needed

Lagom Cluster



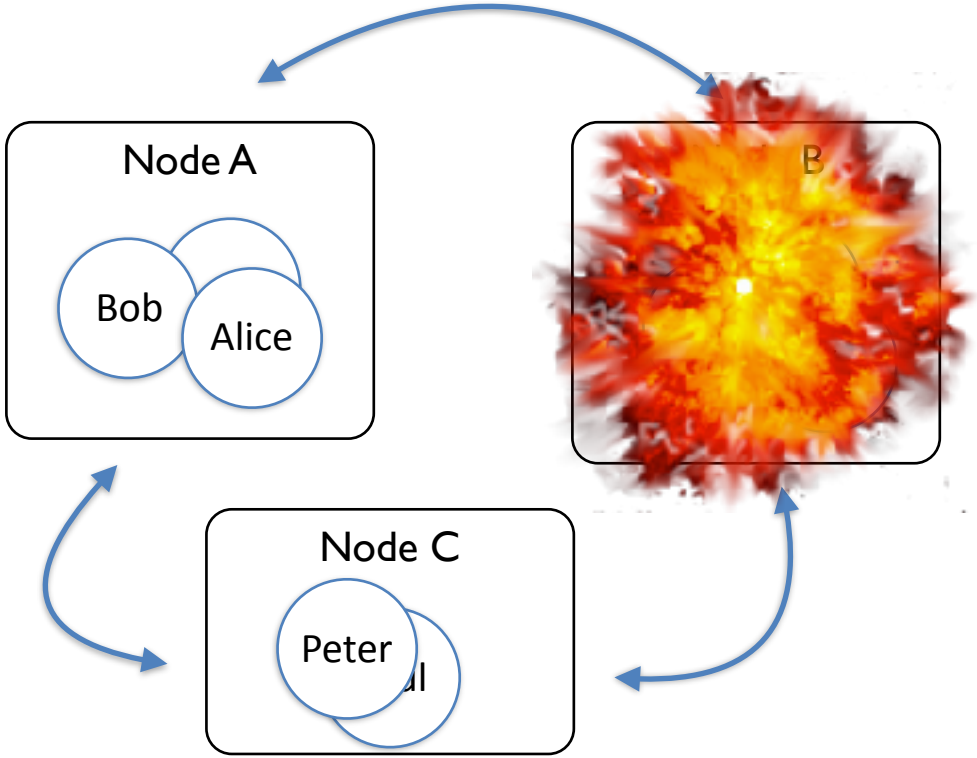
- Lagom allows you to scale out by distributing your **`PersistentEntities`** in the cluster (Cluster Sharding)

Lagom Cluster



- Lagom allows you to scale out by forming a cluster of nodes
- Nodes can be added and removed dynamically

Lagom Persistence

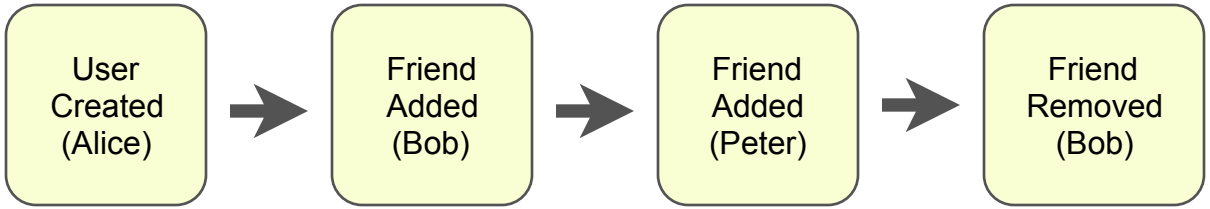


- But how does our data survive a system crash?
- We log all the state changes!

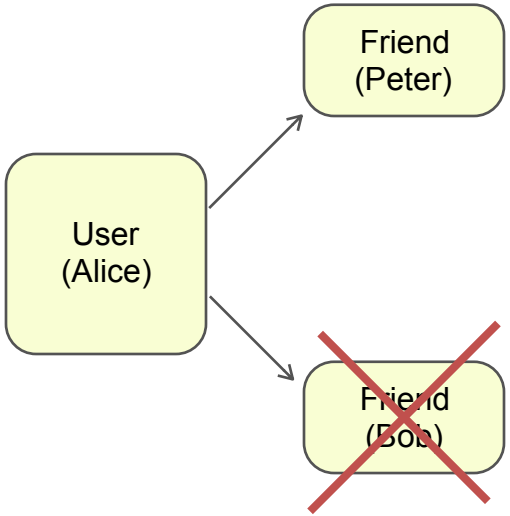
Event Sourcing - storing deltas

- Every state change is materialized in an Event
- All events are stored in an Event Log
- Current state is constructed by replaying all events

Event Sourcing - Storing Deltas



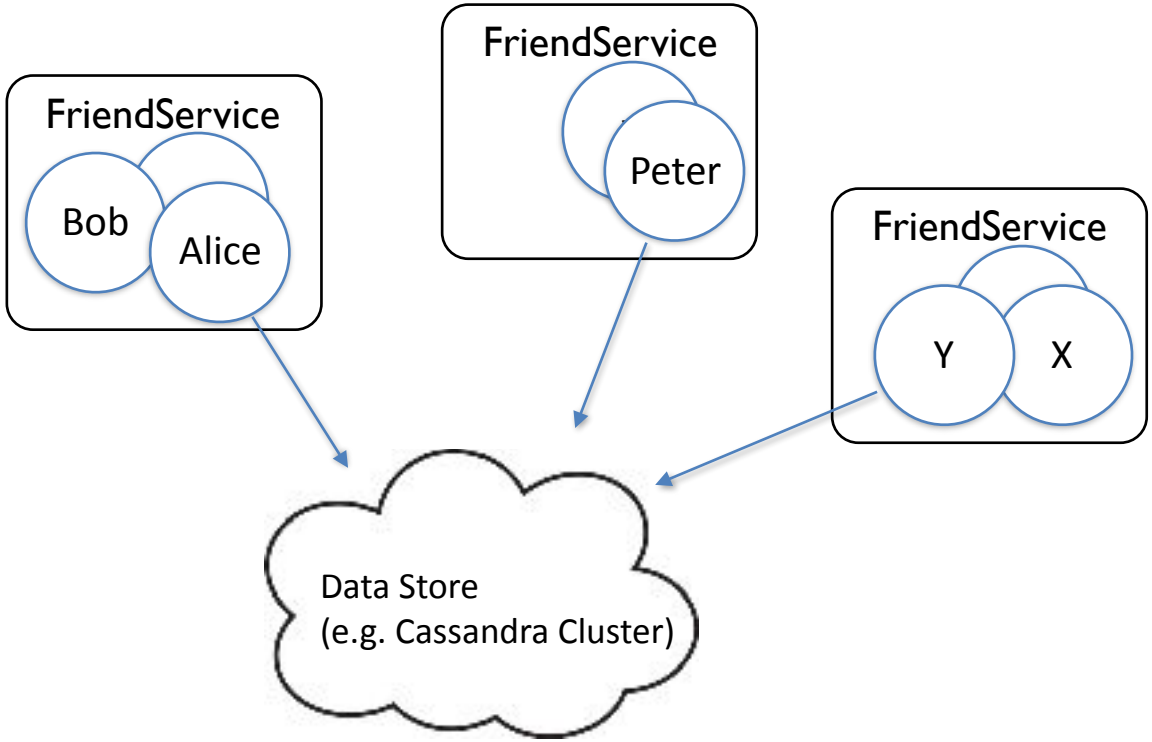
Traditional Way



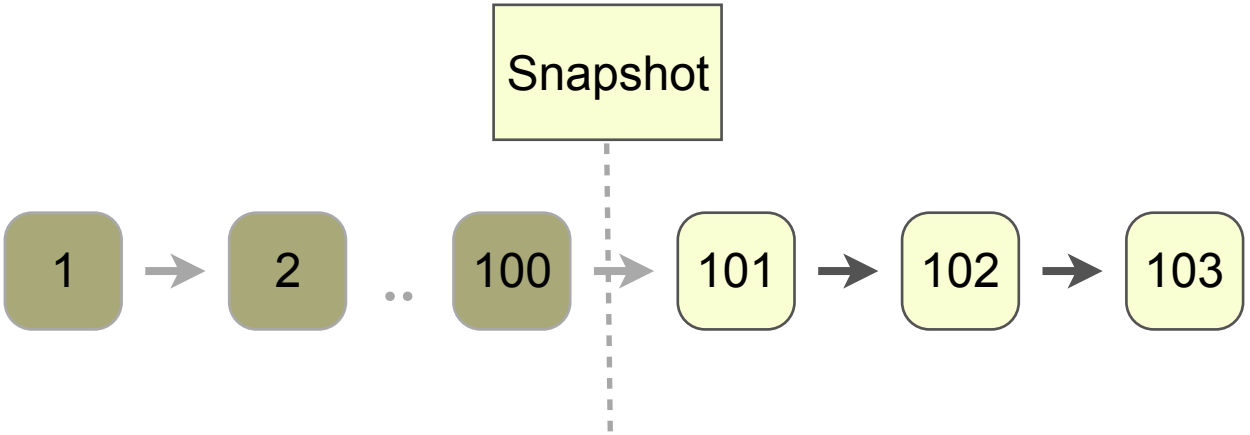
Event Sourcing - benefits

- No object-relational impedance mismatch
- Bullet-proof auditing and historical tracing
- Support future ways of looking at data
- Performance and scalability
- Testability

Persistent Entity



Event Sourcing - Snapshots



Event Sourcing with Lagom Persistence

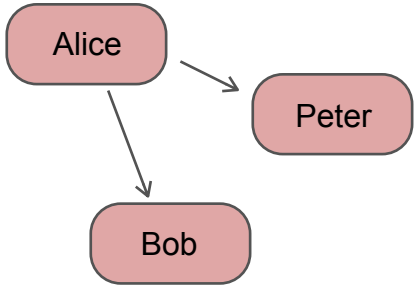
- Keep all data in memory!
  - *Optional: Only working set, by using passivation/ activation*
- Store all state changes as events
- Replay all events of an actor to recreate it
  - *Optional: Start from snapshot*
- Scale out with Lagom Cluster and scalable data store

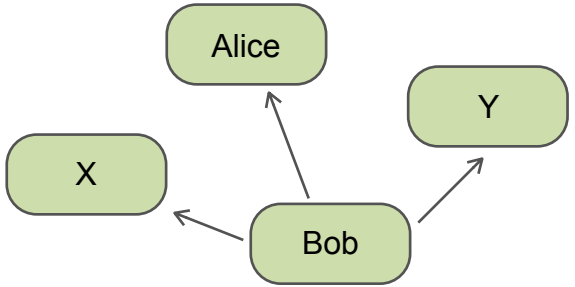
Read-Side

UserCreated(Alice)

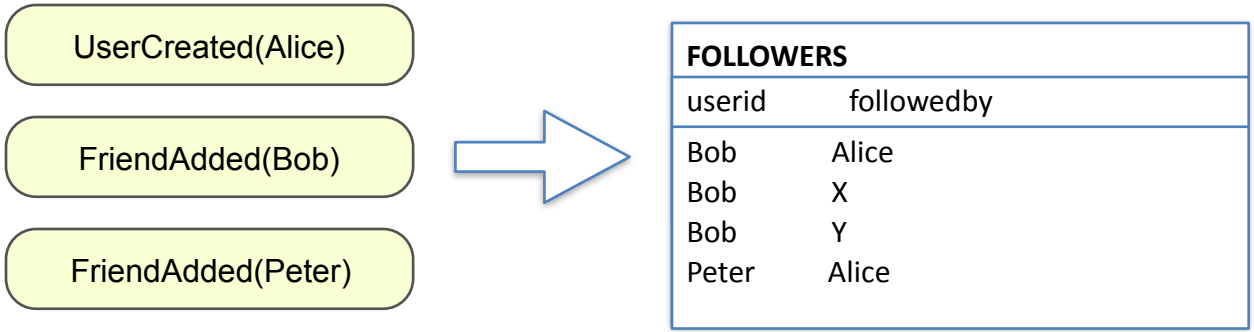
FriendAdded(Bob)

FriendAdded(Peter)





Read-Side


UserCreated(Alice)
FriendAdded(Bob)
FriendAdded(Peter)
FOLLOWERS
userid followedby
Bob Alice
Bob X
Bob Y
Peter Alice

Read side is derived from event log

- Events forwarded to read side to build different representations
    - ElasticSearch
    - SQL, de-normalized
    - Stream Processor / Analytics
    - BI
    - OLAP
    - …

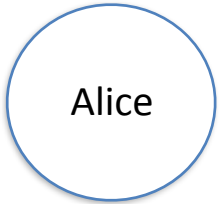
Read side cont’d

- Read side can be discarded and re-created.
- The „book of record“ is the event log.
- Read side can be scaled out by creating copies - it’s read only.

- *Btw, you can get a status for an identified entity on the write side, too.*

Consistency

1 - Actor



2 - Journal / Event Store

3 - Read Side

FOLLOWERS	
userid	followedby
Bob	Alice
Bob	X
Bob	Y
Peter	Alice

Consistency

1 - Actor

- A **Persistent Entities** defines an Aggregate Root
- Aggregate Root is the Transactional Boundary
- Strong consistency within an Aggregate Root
- Commands are executed sequentially on the latest state
- No limit to scalability

Consistency

2 - Journal / Event Store

- Depending on implementation / configuration
- Popular choice: Cassandra
- „Tunable Consistency“
- Proper use of quorum ensures consistency

Consistency

3 - Read Side

FOLLOWERS	
userid	followedby
Bob	Alice
Bob	X
Bob	Y
Peter	Alice

- Will not be updated immediately, but deferred
- Not much different from queries in interactive applications

Event Sourcing with Lagom Persistence revisited

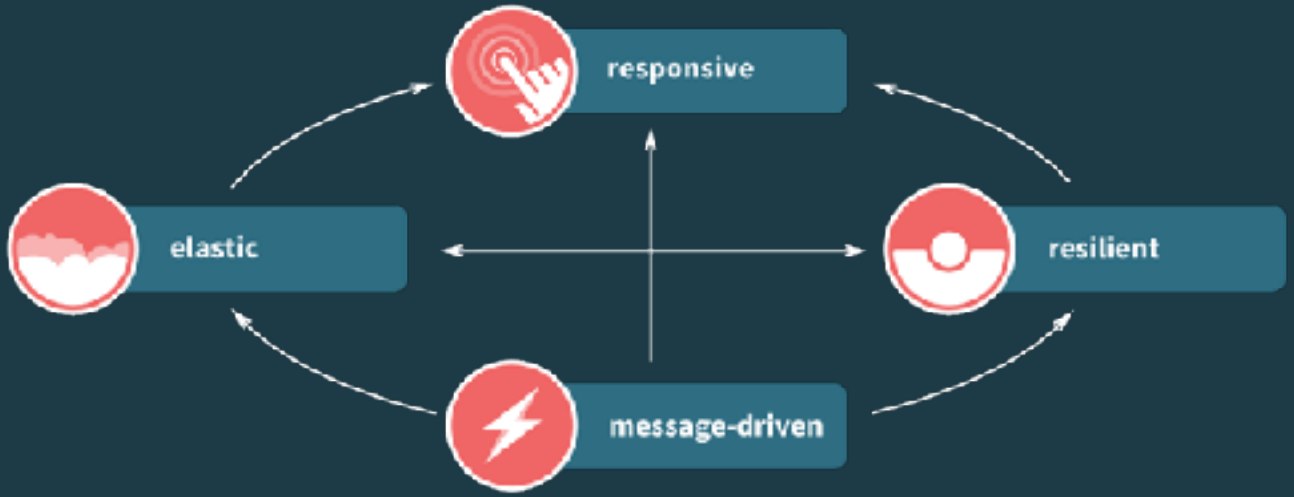
- Keep all data in memory!
- Store all state changes as events
- Replay all events of an actor to recreate it
- Strong consistency for Actor (aggregate) and Journal
- Eventual Consistency for Read Side

Event Sourcing might not be right for everything

- In this case, don’t use Lagom Persistence
- You can use whatever data store you like
- Beware of blocking APIs (JDBC..)
- For Cassandra, you can use the CassandraSession from the Persistence module

Resilience

Reactive



Reactive

- Asynchronous I/O
- Asynchronous communication as first class
  - Reactive Streams over WebSockets support
  - Message Bus (based on Kafka)

Reactive

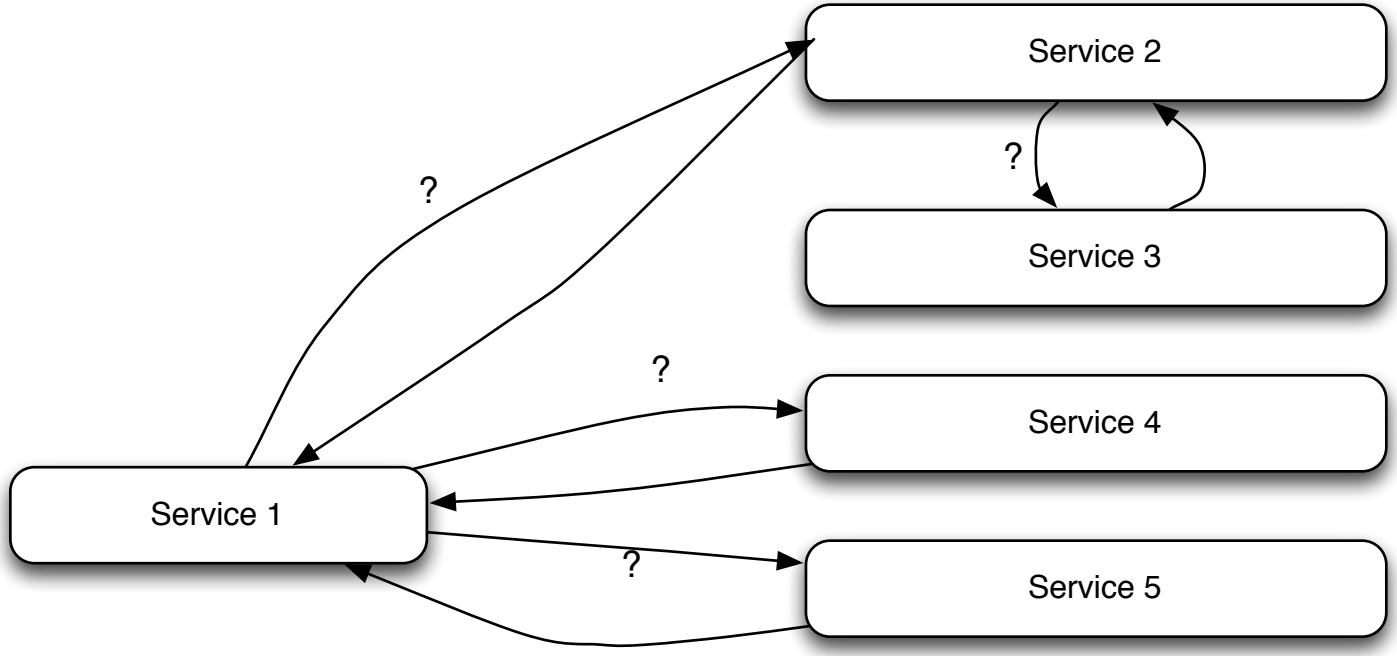
- Distributed by default
  - Built-on Akka Clustering, Sharding, Persistence
- Circuit-Breaker Pattern built-in

Resilience

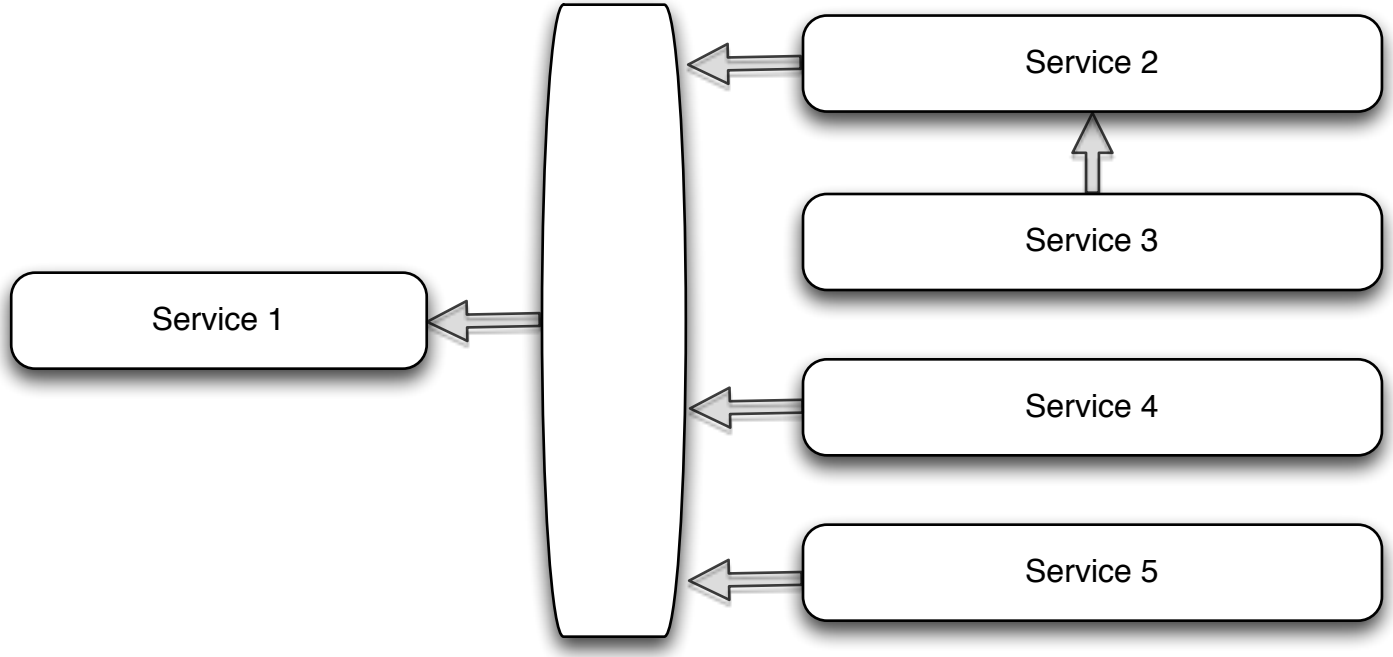
Based on Akka Cluster

- „Node Ring“ with Gossip protocol
- No Single Point of Failure, no Master
- Nodes can leave / join dynamically
- Network partition handling

Authority..



vs. Autonomy



Don’t Ask, Tell

Communication patterns in Microservices

- Query
- Event

By providing a messaging API (and an embedded message bus in development) Lagom encourages the „autonomy“ (event-based) approach.

Without resilience, nothing else matters.
If your beautiful, production-grade, elastic, scalable, highly concurrent, non-blocking, asynchronous, highly responsive and performant application isn't running, then you're back to square one.
It starts and ends with resilience.

Jonas Bonér, CTO

Operations

Deployment

- Create bundles (or Docker images) with a single command
- Needs orchestration tool that provides Service Locator and supports Akka Cluster (node discovery).
- ConductR (commercial product) works seemlessly, open source projects for etcd, ZooKeeper, Consul, go to these for Kubernetes, Docker Swarm..

Try it out

- Getting started: lightbend.com/lagom
- Examples: lightbend.com/activator/templates
- Contribute: https://github.com/lagom
- Communicate:
  - https://groups.google.com/forum/#!forum/lagom-framework
  - https://gitter.im/lagom/lagom
- Lightbend Proof of Concept Program: lightbend.com/company/contact

Read this book

(free, registration required)

https://www.lightbend.com/reactive-microservices-architecture