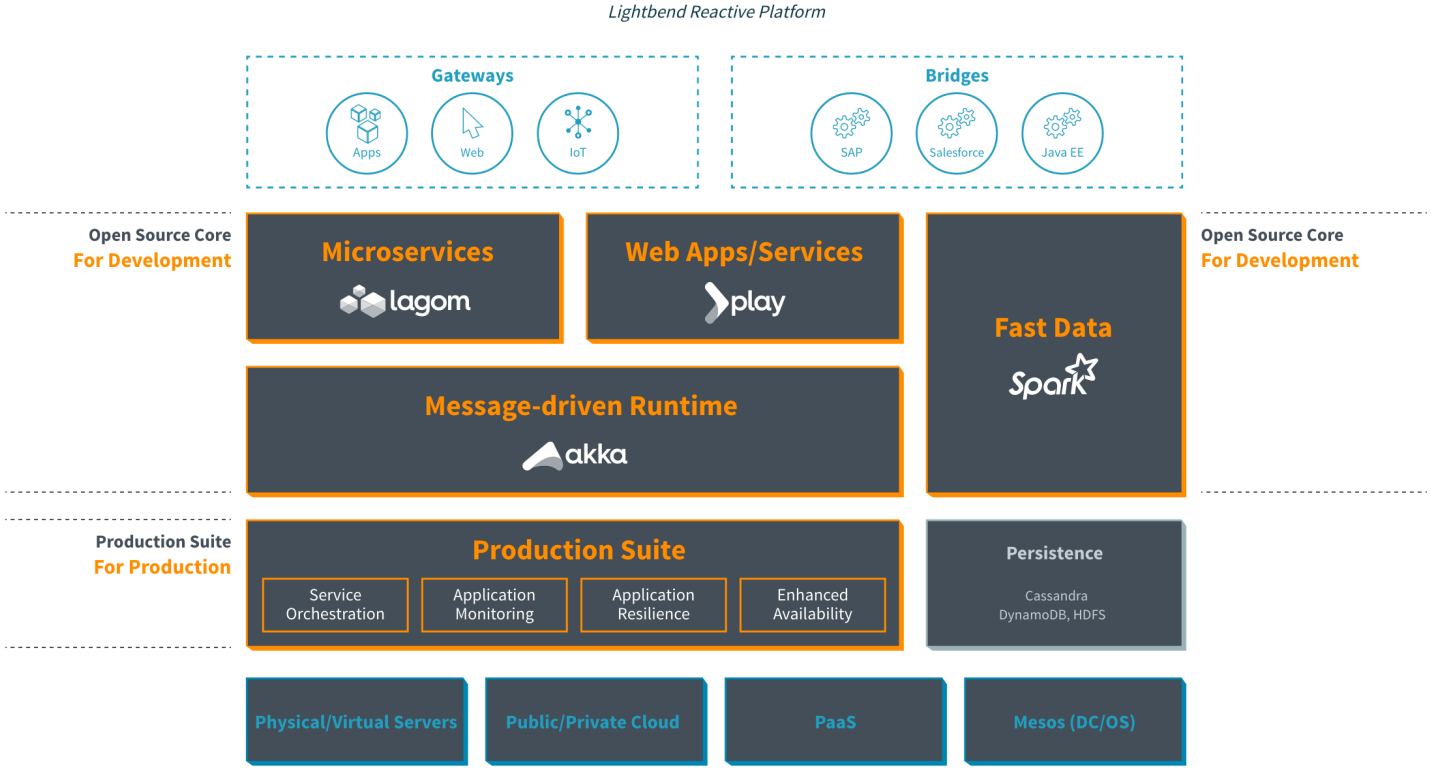
DISTRIBUTED MICROSERVICES

MANAGING CONSISTENCY, STATE AND IDENTITY

SCALA UP NORTH 2017 – VANCOUVER

DUNCAN DEVORE

ABOUT US


Lightbend Reactive Platform
Gateways
Apps
Web
IoT
Bridges
SAP
Salesforce
Java EE
Open Source Core
For Development
Microservices
lagom
Web Apps/Services
play
Fast Data
Spark
Message-driven Runtime
akka
Open Source Core
For Development
Production Suite
For Production
Production Suite
Service Orchestration
Application Monitoring
Application Resilience
Enhanced Availability
Persistence
Cassandra
DynamoDB, HDFS
Physical/Virtual Servers
Public/Private Cloud
PaaS
Mesos (DC/OS)

ABOUT ME

- Senior Software Engineer
- Principal Systems Engineer
- Book : Reactive Application Design (Manning)
- Twitter : @ironfish
- Github : ironfish

MICROSERVICES.

WHAT EXACTLY ARE THEY?

"TRADITIONAL APPLICATION ARCHITECTURES AND PLATFORMS ARE OBSOLETE"

-- GARTNER

Lightbend

ONE DEFINITION

Supports multiple autonomous teams organized to scale the development where the teams can develop, deploy and manage their services independently

THE ARCHITECTURAL CONTEXT OF MICROSERVIES:

DISTRIBUTED SYSTEMS

ANOTHER DEFINITION

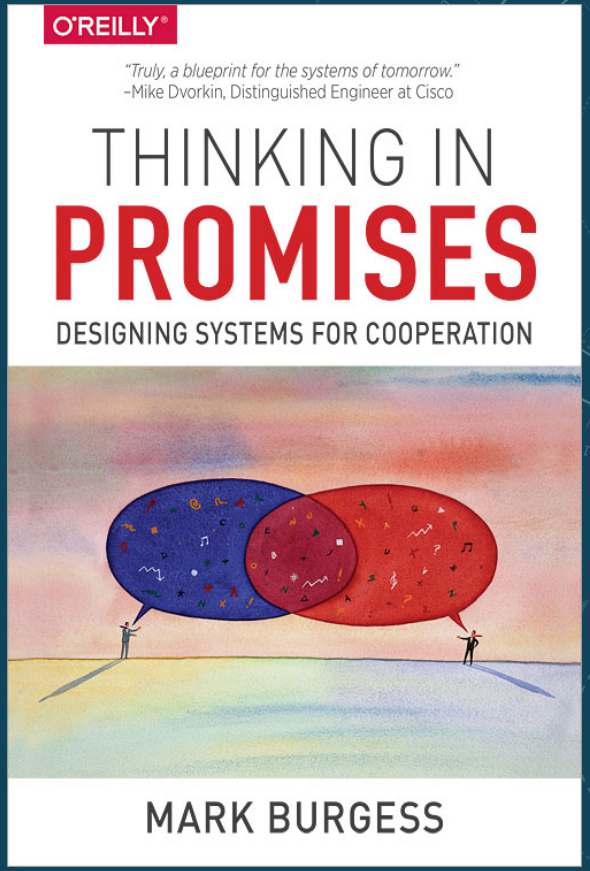
A system of autonomous collaborative distributed services

AUTONOMY

FROM GREEK AUTO-NOMOS:
AUTO MEANING SELF
NOMOS MEANING LAW

Leading the way:

Promise Theory

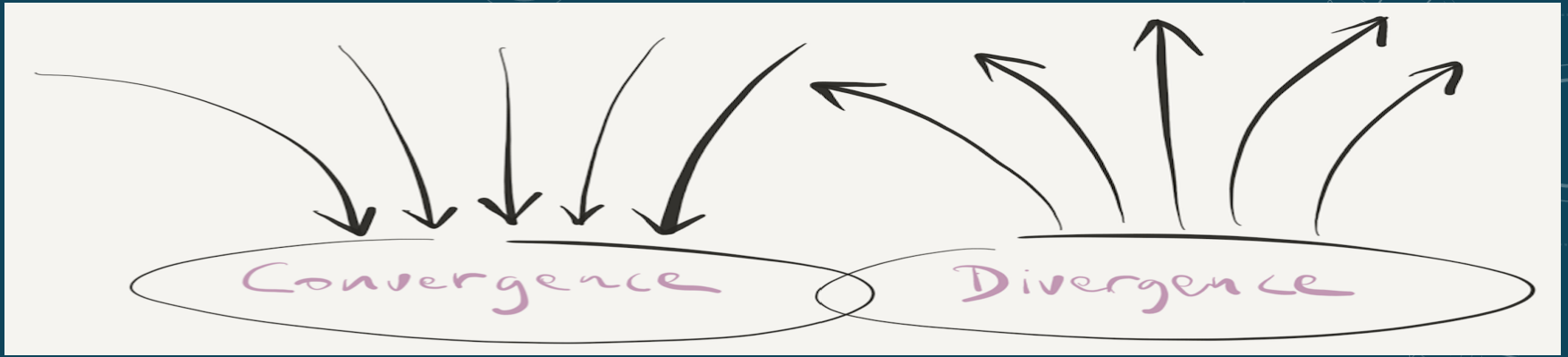


Lightbend

THINK IN PROMISES NOT OBLIGATIONS

"AUTONOMY MAKES INFORMATION LOCAL, LEADING TO GREATER CERTAINTY AND STABILITY"

- IN SEARCH FOR CERTAINTY BY MARK BURGESS



OBLIGATIONS DIVERGE INTO UNPREDICTABLE OUTCOMES FROM DEFINITE BEGINNINGS ⇒ DECREASED CERTAINTY

PROMISES CONVERGE TOWARDS A DEFINITE OUTCOME FROM UNPREDICTABLE BEGINNINGS ⇒ IMPROVED CERTAINTY

Lightbend

Meri Williams
@Geek_Manager

Follow

"Promises are not guarantees. Guarantees are not possible" @markburgess_osl #operability #OIO16

RETWEETS 10 LIKES 9

11:27 AM - 19 Sep 2016

Lightbend

AN AUTONOMOUS SERVICE CAN ONLY PROMISE ITS OWN BEHAVIOR

CONSISTENT STATE.

WHAT ABOUT IT?
SOUNDS LIKE A PROBLEM

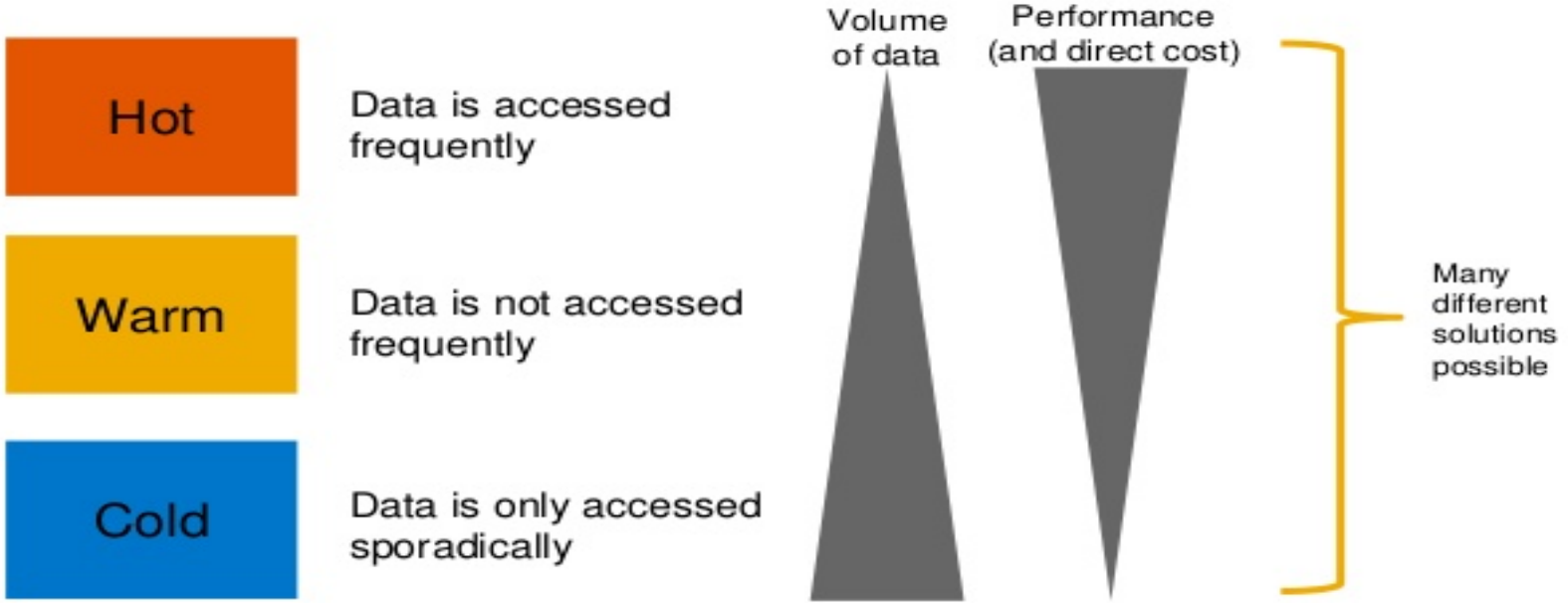
ARE WE IGNORING SOMETHING IMPORTANT?

Lightbend

ONE DEFINITION

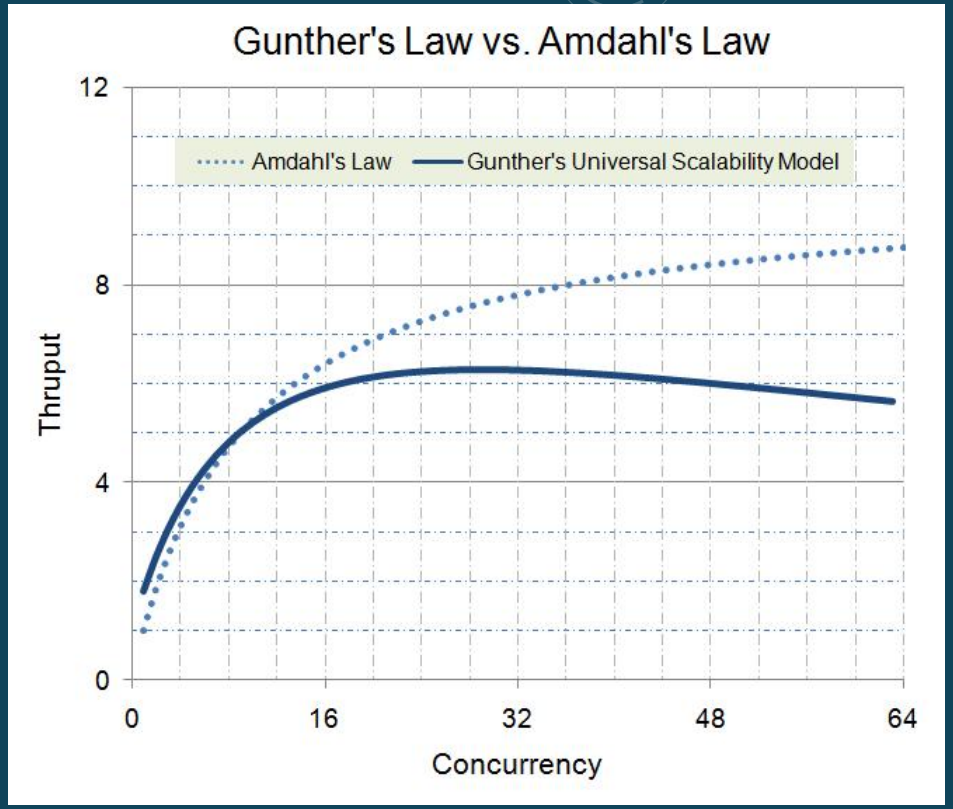
Stateful means the computer or program keeps track of the state of interaction, usually by setting values in a storage field designated for that purpose.

Balancing Data Temperature and Costs



Lightbend

Regardless of storage medium our view of state is always of the past!



The cost of maintaining the illusion of an absolute now

Lightbend

As latency gets higher, the illusion cracks even more

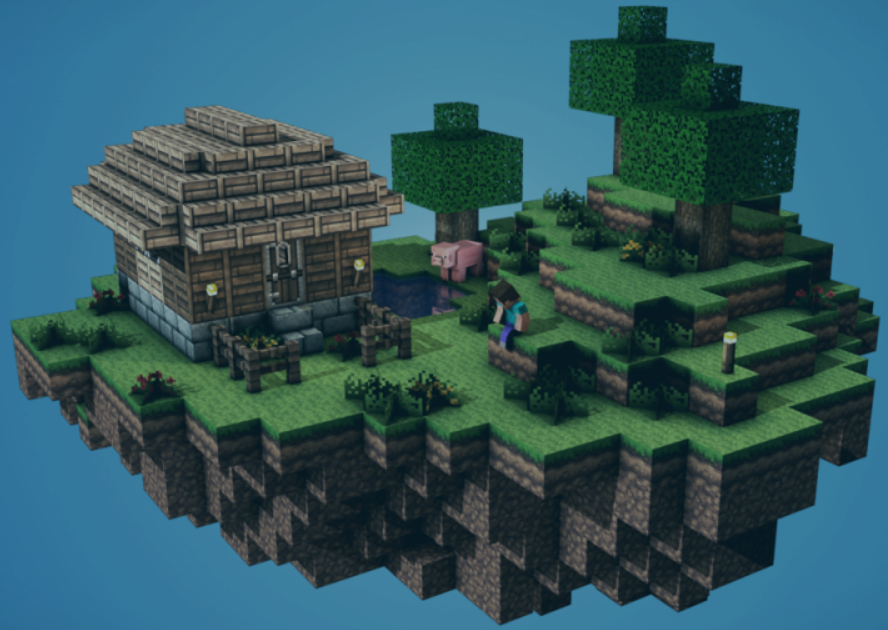
Lightbend

"In a distributed system, you can know where the work is done or you can know when the work is done but you can't know both"

--Pat Helland

The Perfect Storm

The Island ☺

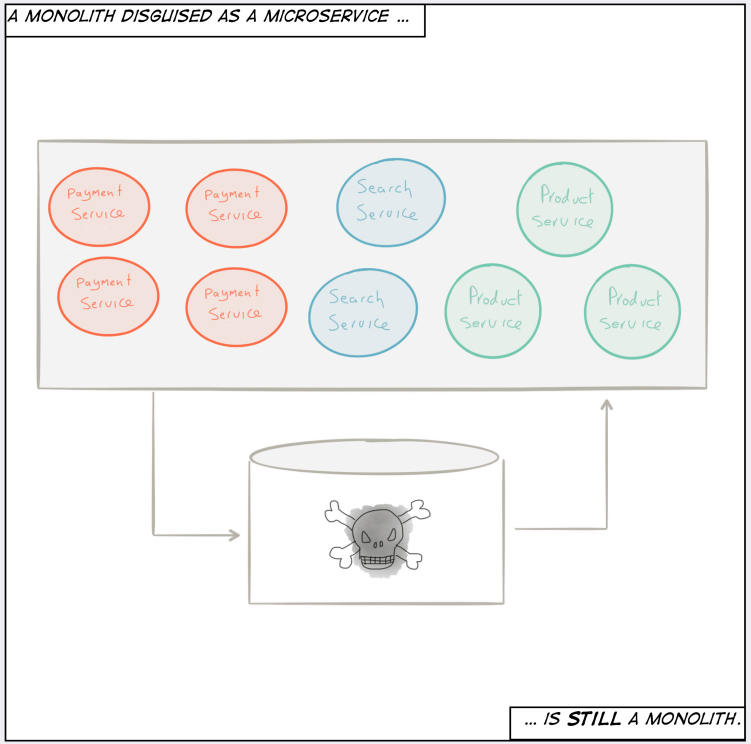


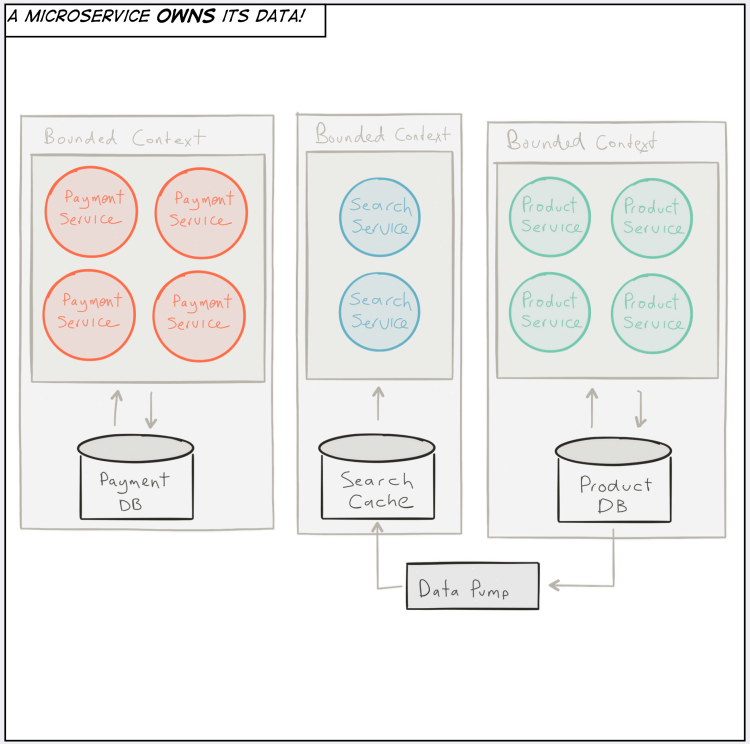
Lightbend

- Inside data: Our current present
- Outside Data: Blast from the past
- Between Services: Hope for the future

-- Data on the inside vs Data on the Outside, Pat Helland

OWN YOU STATE. EXCLUSIVELY



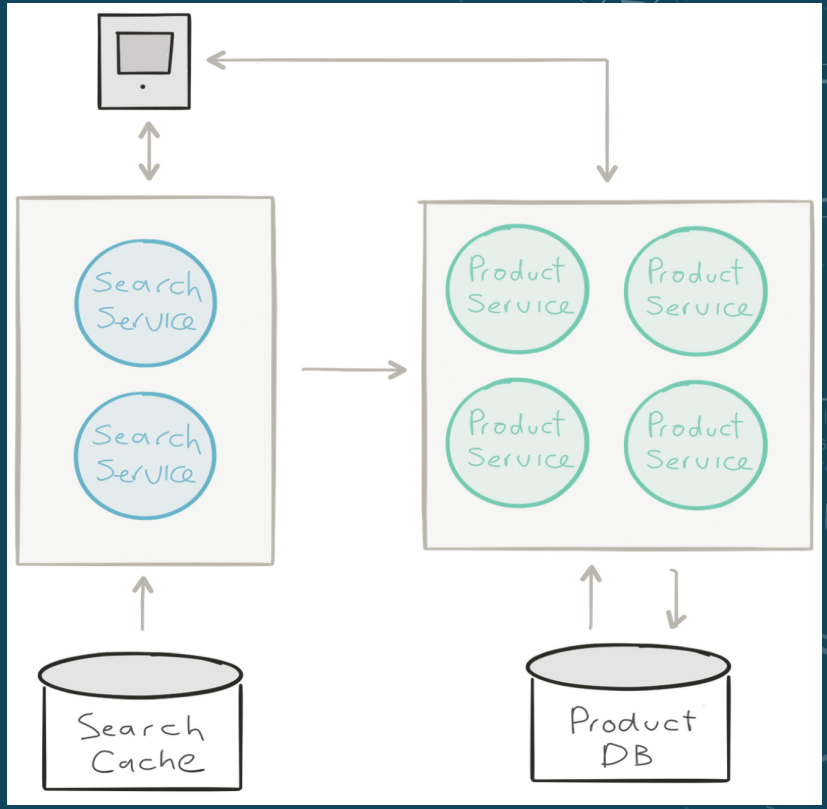


There is no such thing as a *"stateless"* architecture. It's just someone else's problem of an absolute now

Lightbend


Think in terms of
Consistent Boundaries
Lightbend

Bounded Contexts



Lightbend

We have to rely on

Eventual Consistency

But don't be surprised

It's how the world works

No one wants

Eventual Consistency

It is a necessary evil

COMPOSABLE IDENTITY.

THE MEANING OF LIFE?

Polyglot Persistence

"The truth is the log. The database is a cache of a subset of the log."

--Pat Helland

FAVOR EVENT LOGGING

Date	Comment	Change	Balance
1/1/2012	Deposit from 3300	+10000.00	10000.00
1/3/2012	Check 1	-4000.00	6000.00
1/4/2012	ATM withdraw	-3.00	5997.00
1/11/2012	Check 2	-5.00	5992.00
1/12/2012	Deposit from 3301	+2000.00	7992.00

Shopping Cart

CRUD

vs

Event Sourcing

Lightbend

The CRUD Shopping Cart

1. Cart created
2. Item 1 @ $30 added.
3. Item 2 @ $15 added.
4. Item 4 @ $12 added.
5. Item 5 @ $5 added.
6. Shipping information added.
7. Total @ $62 generated.
8. Order 123 inserted

Lightbend

The CRUD Shopping Cart

Now at some time in the future before the order is shipped, the customer changes their mind and wants to delete an item.

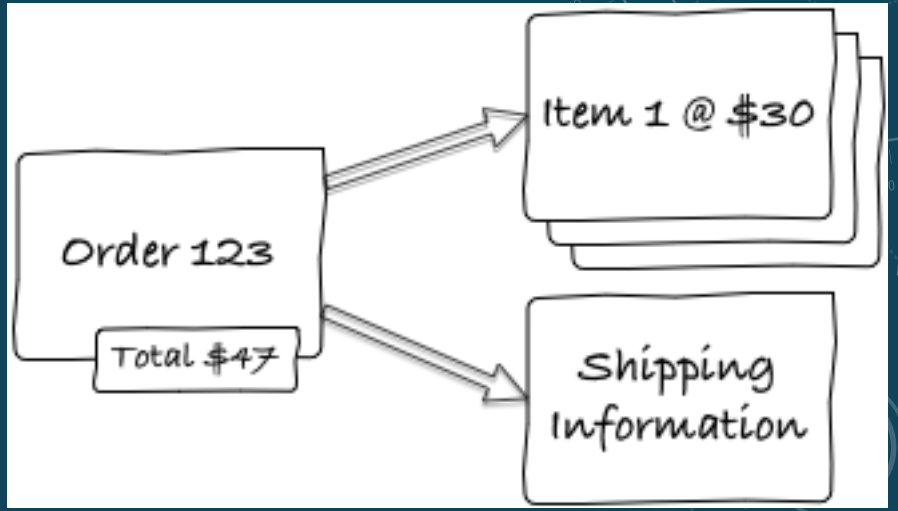
1. Order 123 fetched
2. Item 2 @ $15 removed
3. Total @ $47 regenerated
4. Order 123 updated

The CRUD Shopping Cart

This is the current state persisted

The result of these transactions:

- the current state of the order
- is 3 items with a total of $47



Lightbend

The CRUD Shopping Cart

Now the manager asks the development team to produce a report of all orders where customers have removed items. Since only the current state of the data is recorded this cant be done.

- The development team will add in a future sprint
- Once added it will only work from that point forward
- Substantial implications to the value of the data

ORMS & Static State Models

- Work well in most situations, come at a fairly large cost
- Query and persist current state to database
- Tightly coupled domain and data model
- Can lead to leaky abstraction
- Can lead to an anemic domain model
- Lossy and the intent of the user is not captured

Lightbend

Static State Models

- Represent change between two points
- Commonly referred to as Deltas
- In static state models Deltas are implicit
- They are left to frameworks such as an ORM
- ORMs save state, calculate diffs, update backing model
- As a result much of the intent or behavior is lost
- Auditing is almost always explicit

THE REACTIVE SHOPPING CART

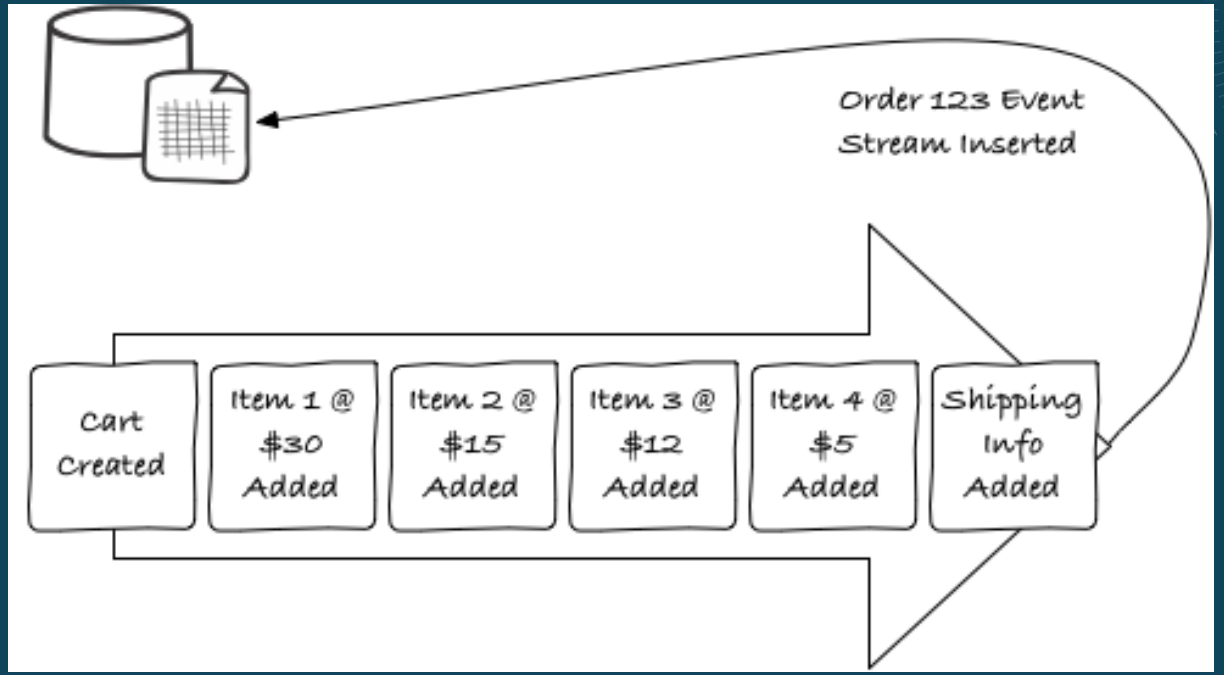
Events

- Events are notifications
- They report on something that has already happened
- As such, events cannot be rejected
- An event would be something like:
  - `ClientRegistered`
  - `ClientLocaleChanged`

The Reactive Shopping Cart

1. Cart created
2. Item 1 @ $30 added.
3. Item 2 @ $15 added.
4. Item 3 @ $12 added.
5. Item 4 @ $5 added.
6. Shipping information added.
7. Total @ $62 generated.
8. Order 123 event stream inserted

The Reactive Shopping Cart



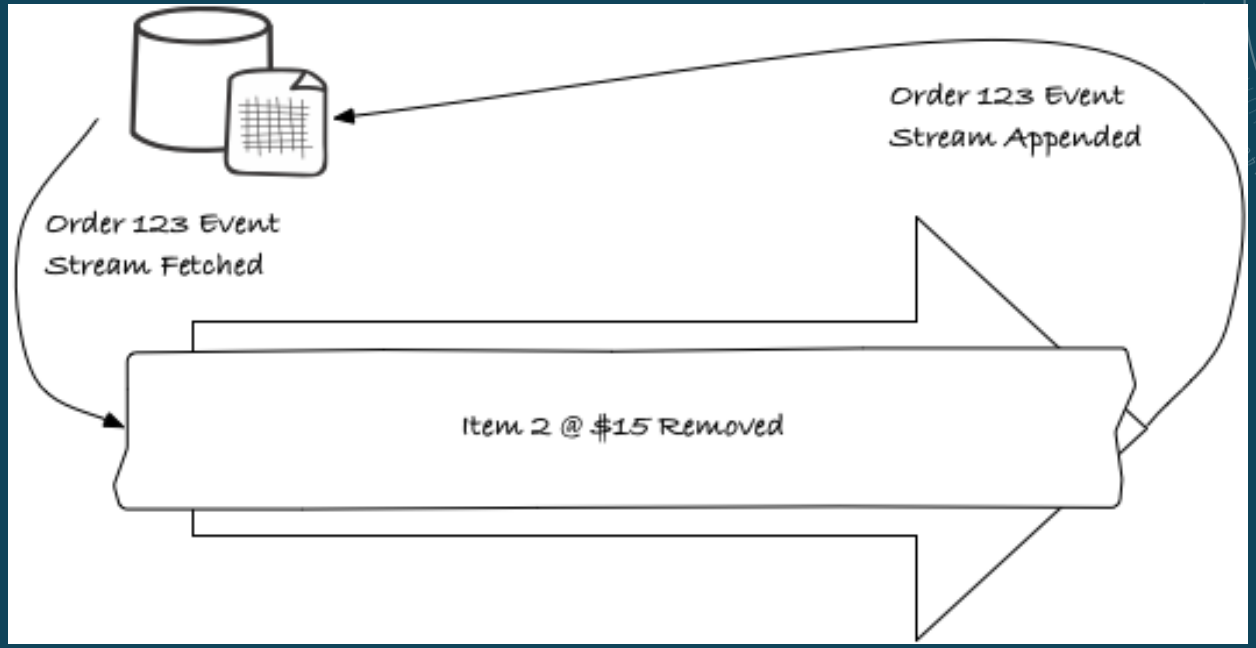
Lightbend

The Reactive Shopping Cart

Now at some time in the future before the order is shipped, the customer changes their mind and wants to delete an item.

1. Order 123 event stream fetched
2. Item 2 @ $15 removed event
3. Order 123 event stream appended

The Reactive Shopping Cart



Lightbend

The Reactive Shopping Cart

By replaying the event stream the object can be returned to the last known state.

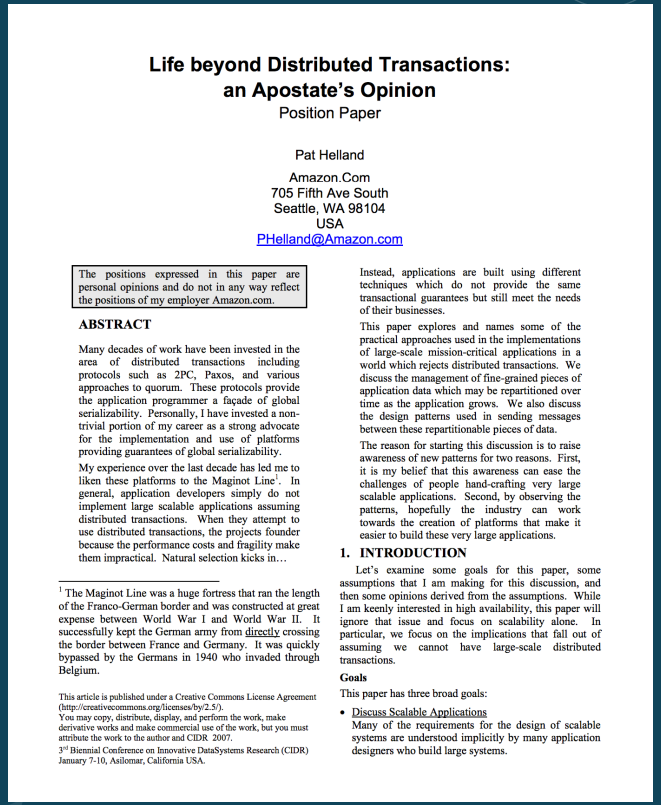
- There is a structural representation of the object
- It exists by replaying previous transactions
- Data is not persisted structurally
- It is a series of events
- No coupling between current state in the domain and storage

Technology Implications

- The storage system becomes an additive only architecture
- Append-only architectures distribute
- Far fewer locks to deal with
- Horizontal Partitioning is difficult for a relational model
- What key do you partition on in a complex relational model?
- When using an Event Store there is only 1 key!

Business Implications

- Criteria is tracked from inception as an event stream
- You can answer questions form the origin of the system
- You can answer questions not asked yet!
- Natural audit log



Life beyond Distributed Transactions: an Apostate's Opinion

Position Paper

Pat Helland

Amazon.Com
705 Fifth Ave South
Seattle, WA 98104
USA
PHelland@Amazon.com

The positions expressed in this paper are personal opinions and do not in any way reflect the positions of my employer Amazon.com.

ABSTRACT

Many decades of work have been invested in the area of distributed transactions including protocols such as 2PC, Paxos, and various approaches to quorum. These protocols provide the application programmer a façade of global serializability. Personally, I have invested a non-trivial portion of my career as a strong advocate for the implementation and use of platforms providing guarantees of global serializability.

My experience over the last decade has led me to liken these platforms to the Maginot Line[1]. In general, application developers simply do not implement large scalable applications assuming distributed transactions. When they attempt to use distributed transactions, the projects founder because the performance costs and fragility make them impractical. Natural selection kicks in… Instead, applications are built using different techniques which do not provide the same transactional guarantees but still meet the needs of their businesses.

This paper explores and names some of the practical approaches used in the implementations of large-scale mission-critical applications in a world which rejects distributed transactions. We discuss the management of fine-grained pieces of application data which may be repartitioned over time as the application grows. We also discuss the design patterns used in sending messages between these repartitionable pieces of data.

The reason for starting this discussion is to raise awareness of new patterns for two reasons. First, it is my belief that this awareness can ease the challenges of people hand-crafting very large scalable applications. Second, by observing the patterns, hopefully the industry can work towards the creation of platforms that make it easier to build these very large applications.

1. INTRODUCTION

Let's examine some goals for this paper, some assumptions that I am making for this discussion, and then some opinions derived from the assumptions. While I am keenly interested in high availability, this paper will ignore that issue and focus on scalability alone. In particular, we focus on the implications that fall out of assuming we cannot have large-scale distributed transactions.

Goals

This paper has three broad goals:

- Discuss Scalable Applications
  Many of the requirements for the design of scalable systems are understood implicitly by many application designers who build large systems.

[1] The Maginot Line was a huge fortress that ran the length of the Franco-German border and was constructed at great expense between World War I and World War II. It successfully kept the German army from directly crossing the border between France and Germany. It was quickly bypassed by the Germans in 1940 who invaded through Belgium.

3rd Biennial Conference on Innovative DataSystems Research (CIDR) January 7-10, Asilomar, California USA.

You need to separate

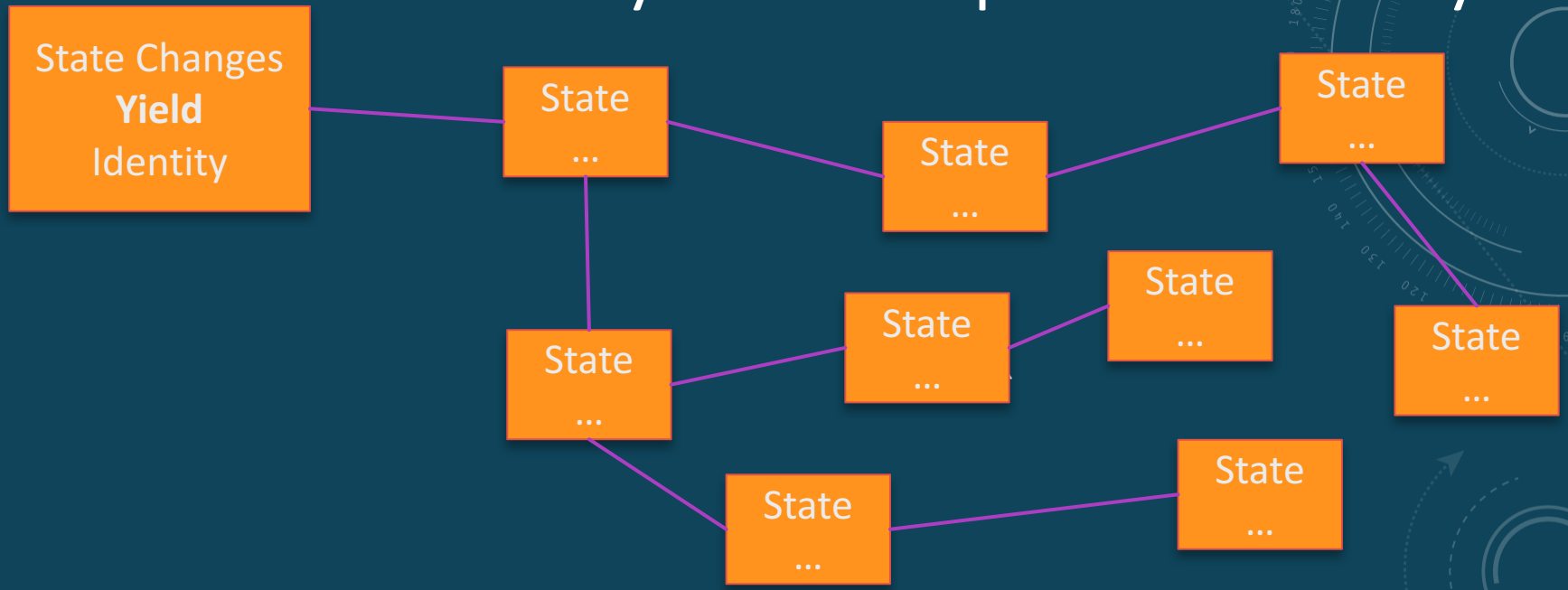
The stateless part - the behavior

from

The stateful part - the knowledge

Lightbend

System Composable Identity



Lightbend

Cheers!