

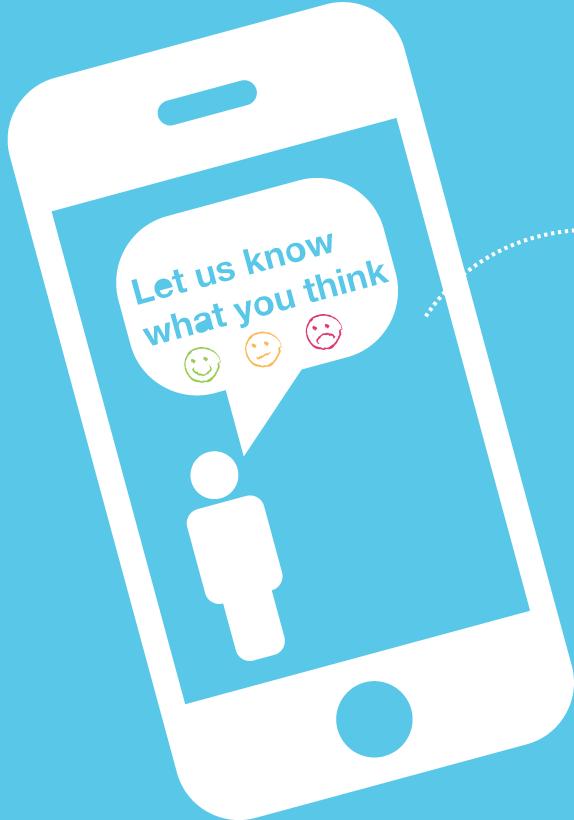


Managing Consistency, State and Identity in Distributed Microservices

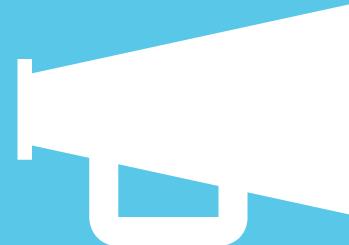
Speaker Duncan DeVore



Join the conversation #scaladays

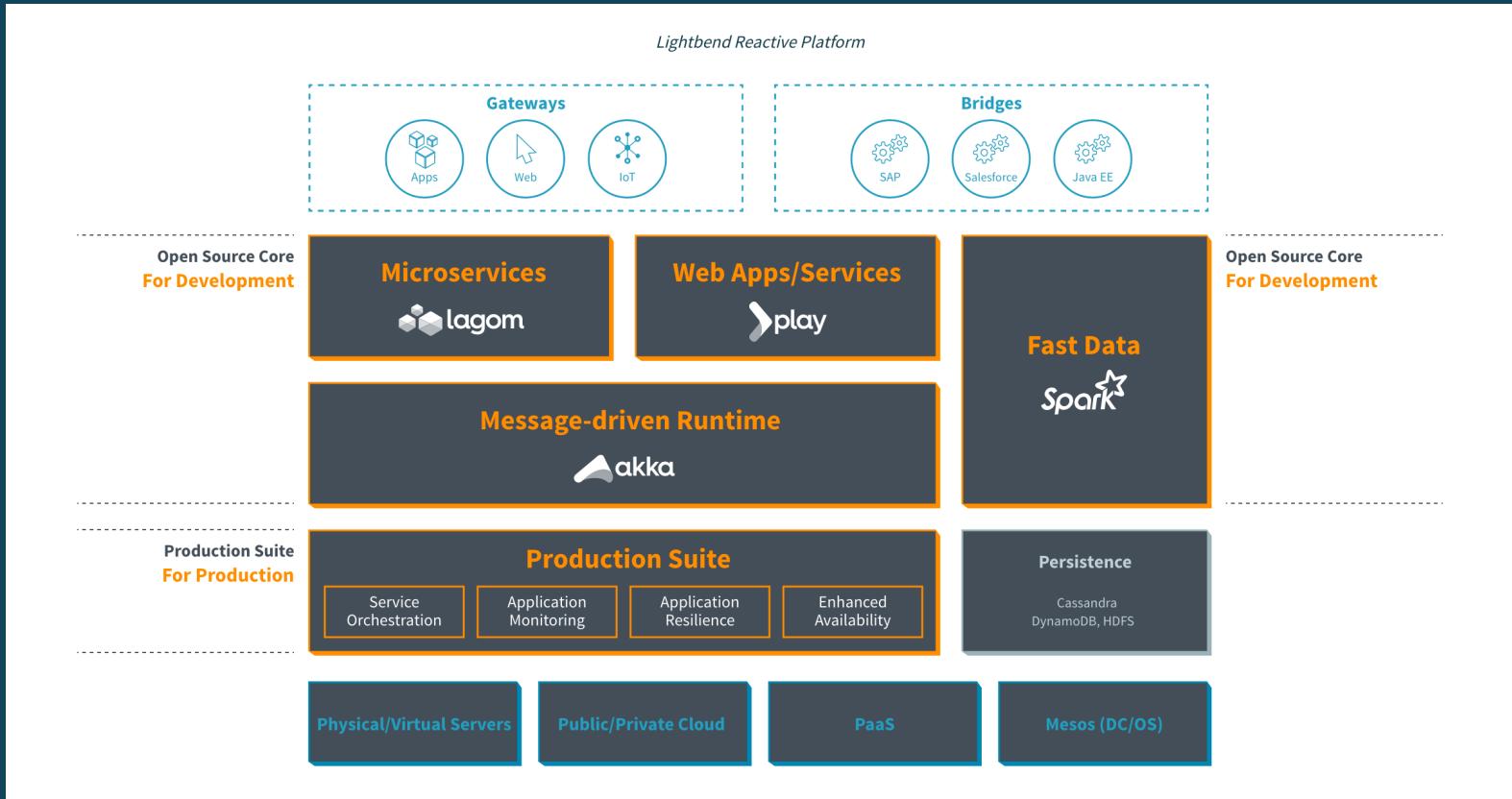


Please use the
Scala Days app
to rate sessions.



Join the conversation #scaladays

About Us



About Me

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



Lightbend

Microservices.

What exactly are they?



Lightbend

*“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



Lightbend

The Architectural Context of Microservices: Distributed Systems



Lightbend

Another Definition

A *system* of autonomous
collaborative distributed
services



Lightbend

Autonomy

From Greek Auto-nomos:

auto meaning self

nomos meaning law

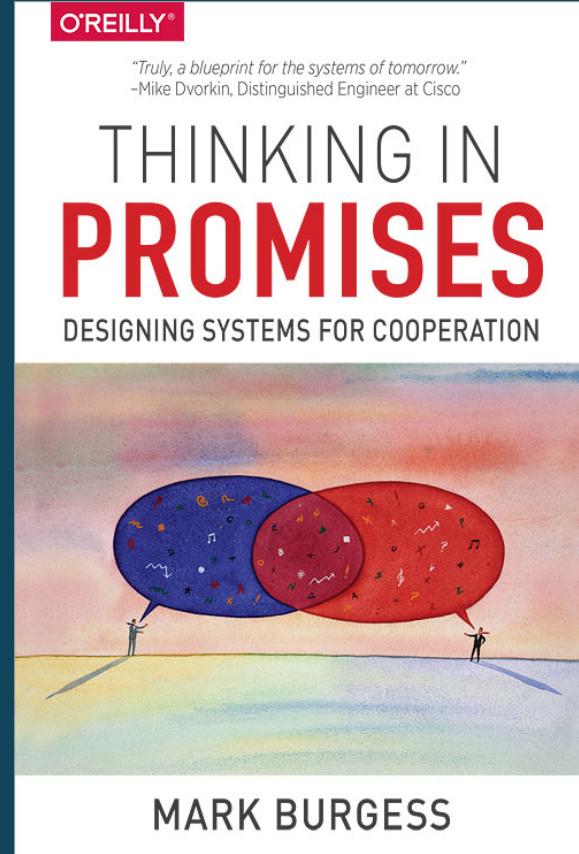


Lightbend

Leading the way: Promise Theory



Lightbend



Think in Promises
not Obligations



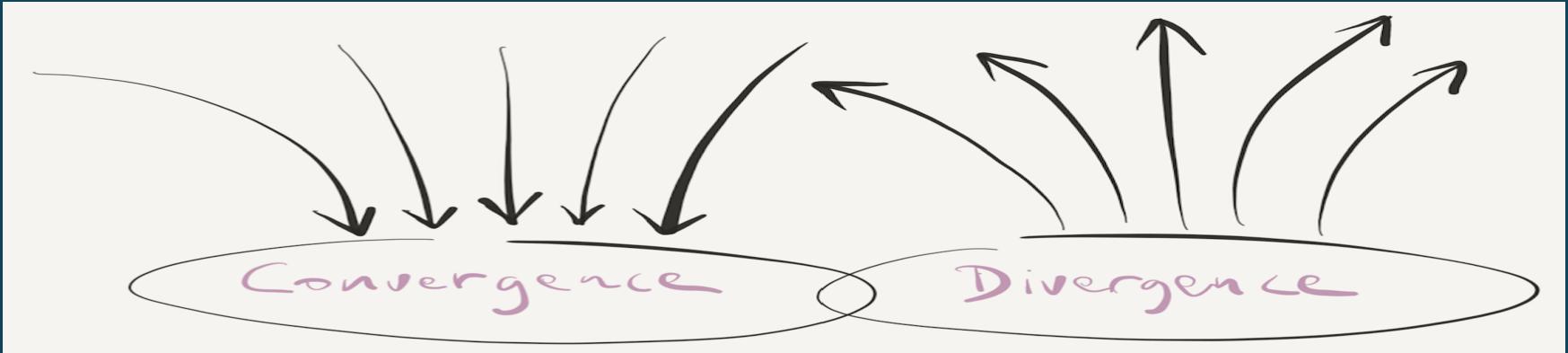
Lightbend

“Autonomy makes information local, leading to greater certainty and stability”

- In Search for Certainty by Mark Burgess



Lightbend



Obligations **diverge** into unpredictable outcomes from **definite** beginnings \Rightarrow decreased certainty

Promises converge towards a **definite outcome** from **unpredictable** beginnings \Rightarrow 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



Lightbend

Consistent State.

What about it?

Sounds like a problem



Lightbend

Are we
ignoring
something
important?



Lightbend

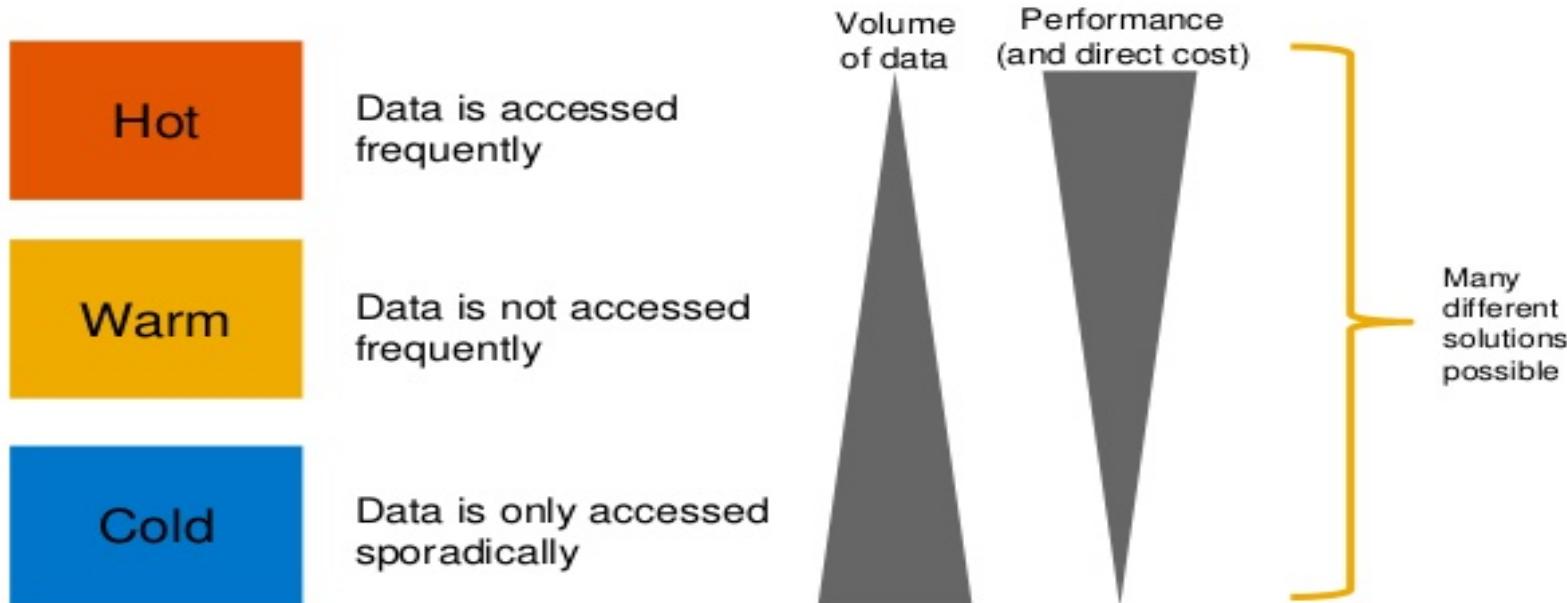
One Definition

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.



Lightbend

Balancing Data Temperature and Costs



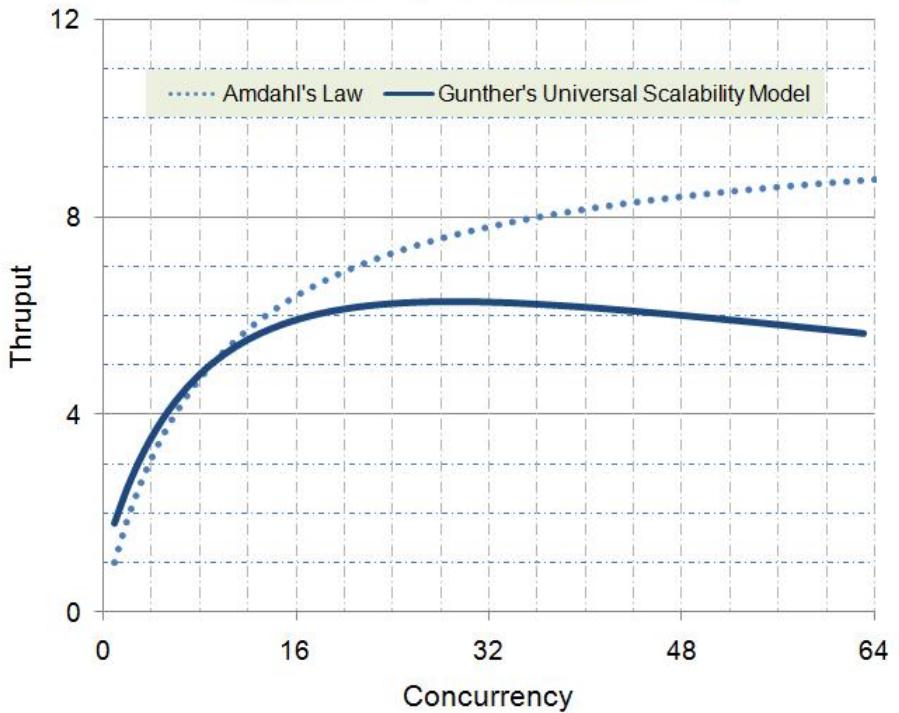
Lightbend

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



Lightbend

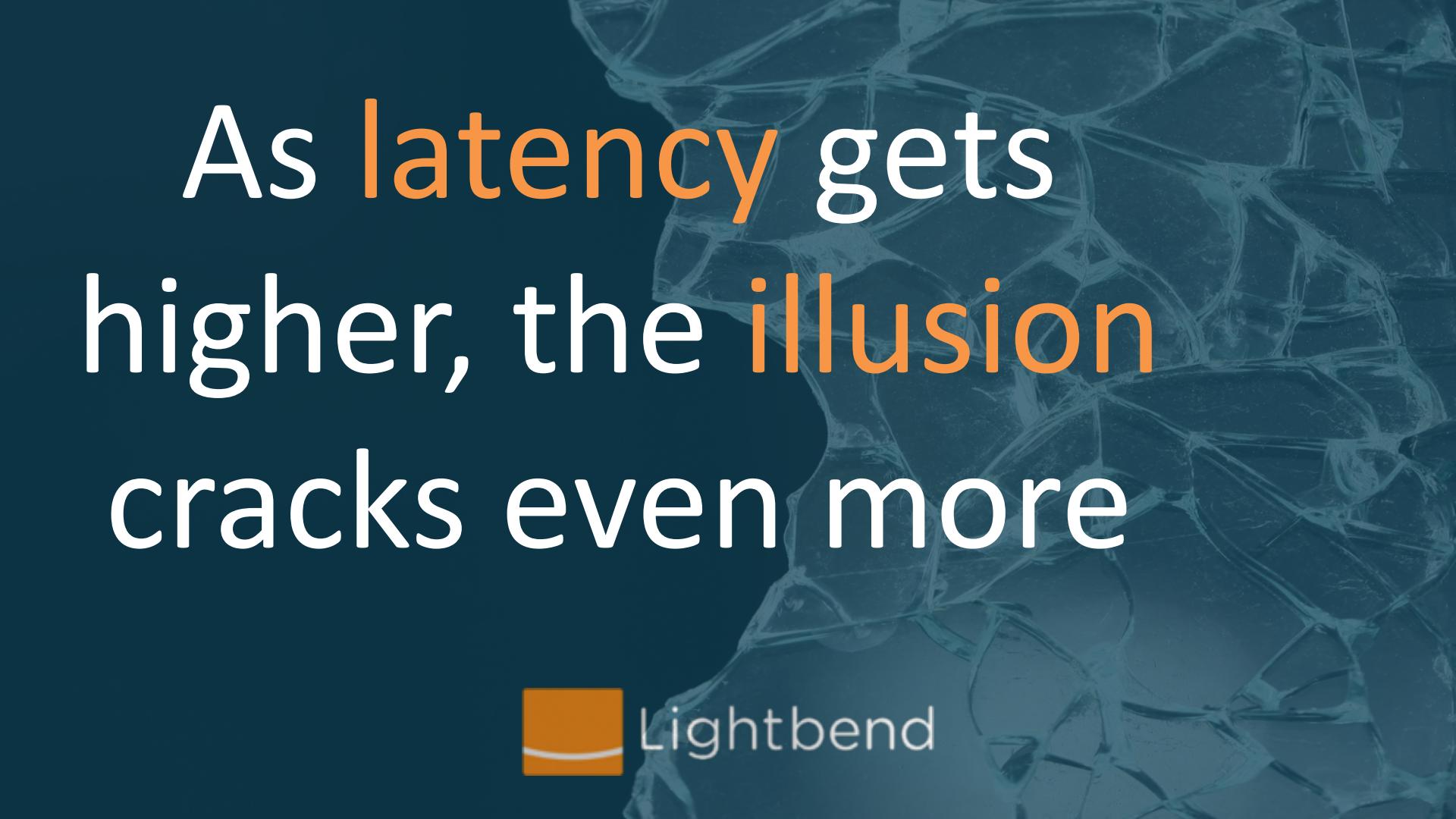
Gunther's Law vs. Amdahl's Law



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



Lightbend

The Perfect Storm



Lightbend

The Island 😊



Lightbend

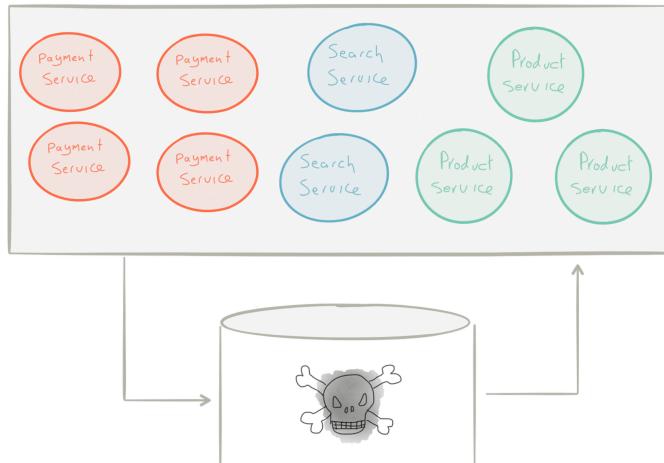
- Inside data: Our current present
- Outside Data: Blast from the past
- Between Services: Hope for the future
 - Data on the inside vs Data on the Outside, Pat Helland



Lightbend

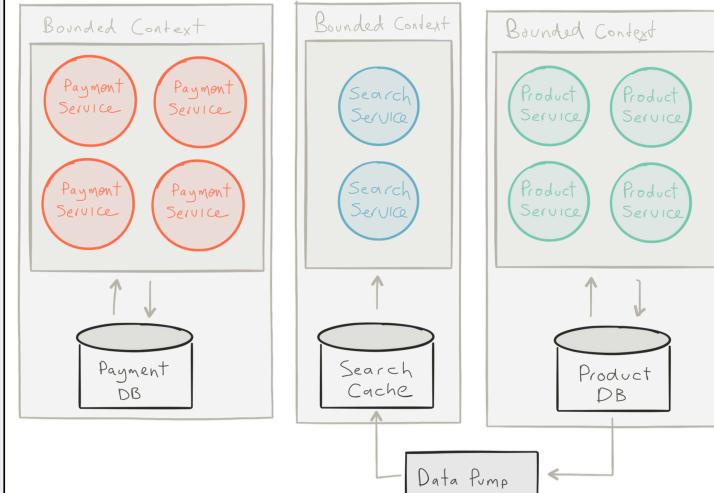
Own your state. Exclusively

A MONOLITH DISGUISED AS A MICROSERVICE ...



... IS STILL A MONOLITH.

A MICROSERVICE OWNS ITS DATA!





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



Lightbend

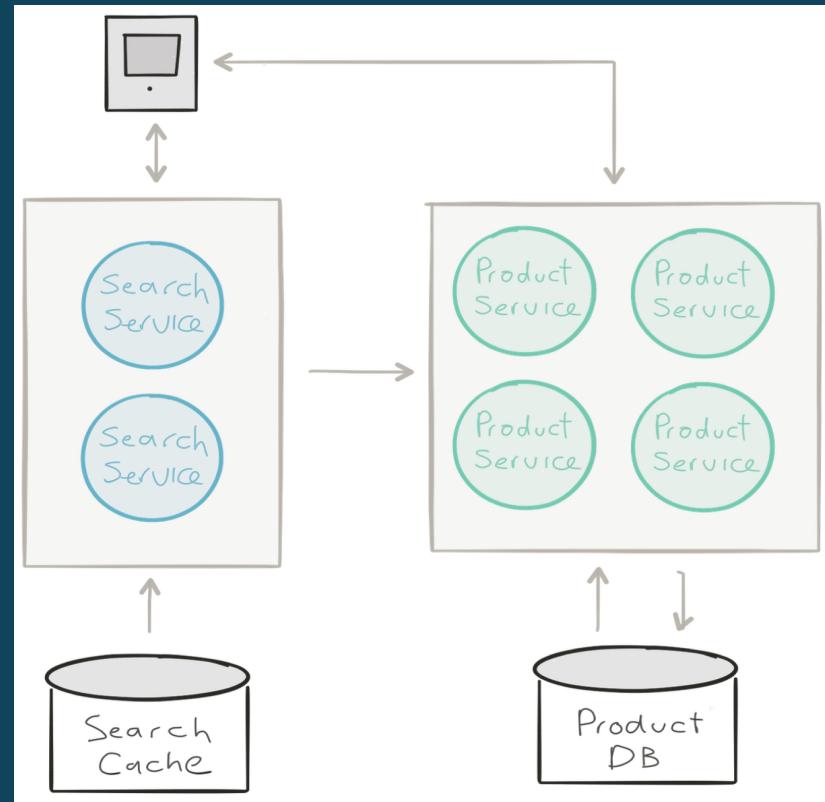


Think in terms of
Consistent Boundaries



Lightbend

Bounded Contexts



Lightbend

*We have to rely on
Eventual Consistency*

But don't be surprised

It's how the world works



Lightbend

*No one wants
Eventual Consistency*

It is a necessary evil



Lightbend

Composable Identity.

The meaning of life?



Lightbend

Polyglot Persistence



Lightbend



*“The truth is the log.
The database is a
cache of a subset of
the log.”*

--Pat Helland



Lightbend

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 3 @ \$12 added.
5. Item 4 @ \$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



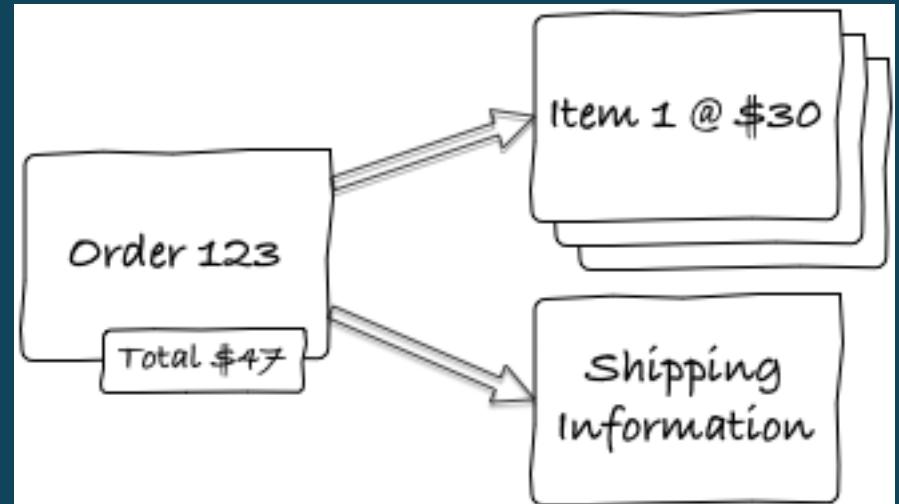
Lightbend

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 can't 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



Lightbend

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**



Lightbend

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**



Lightbend

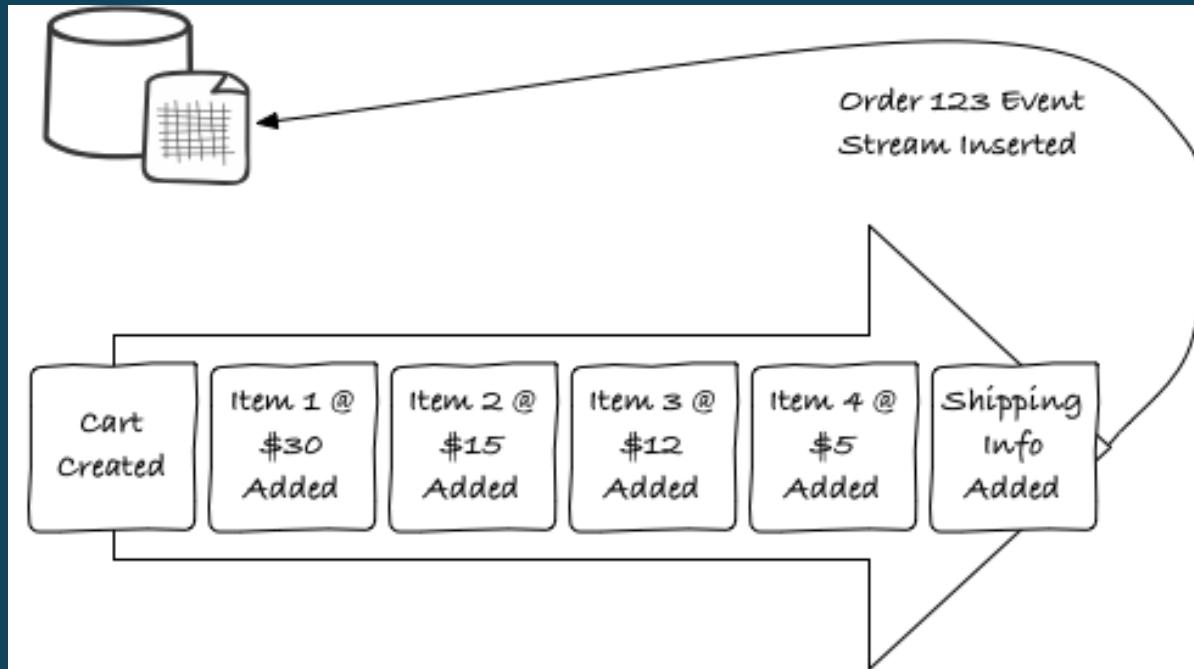
The Reactive Shopping Cart

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



Lightbend

The Reactive Shopping Cart



Lightbend

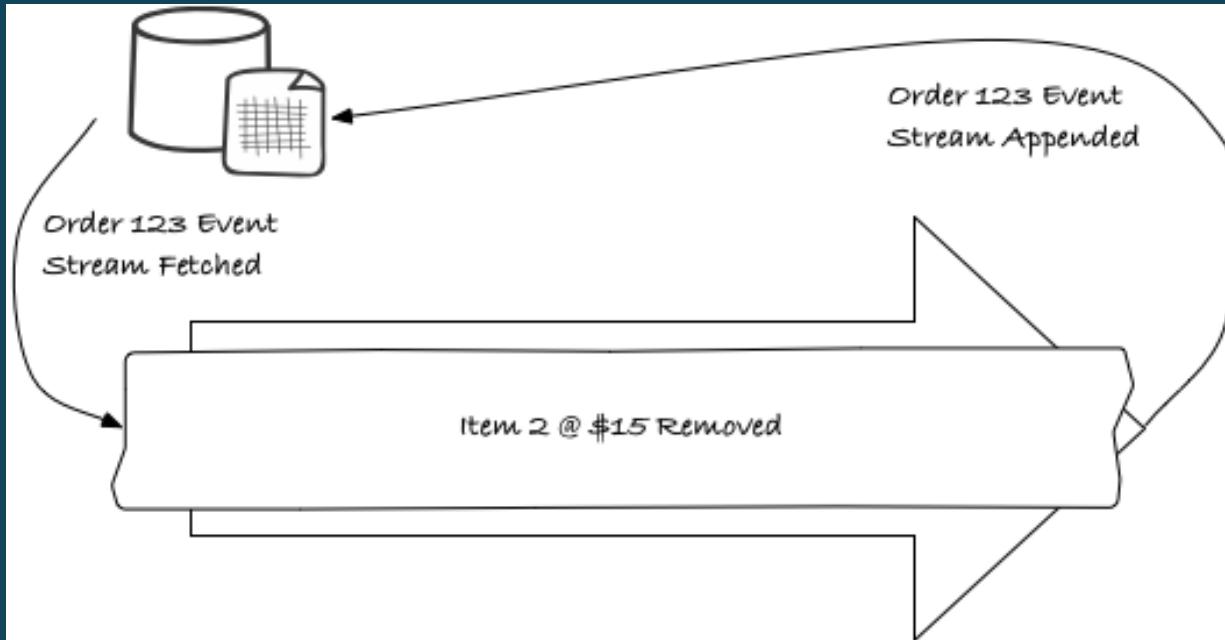
The Reactive Shopping Cart

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



Lightbend

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!**



Lightbend

Business Implications

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



Lightbend

**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 facade 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 Line¹. 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 use of replicated data and application data which may be partitioned 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.

This article is published under a Creative Commons License Agreement (<http://creativecommons.org/licenses/by/2.5/>). You may copy, distribute, display, and perform the work, make derivative works and make commercial use of the work, but you must attribute the work to the author and CIDR 2007.

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

You need to separate

The stateless part - the behavior

from

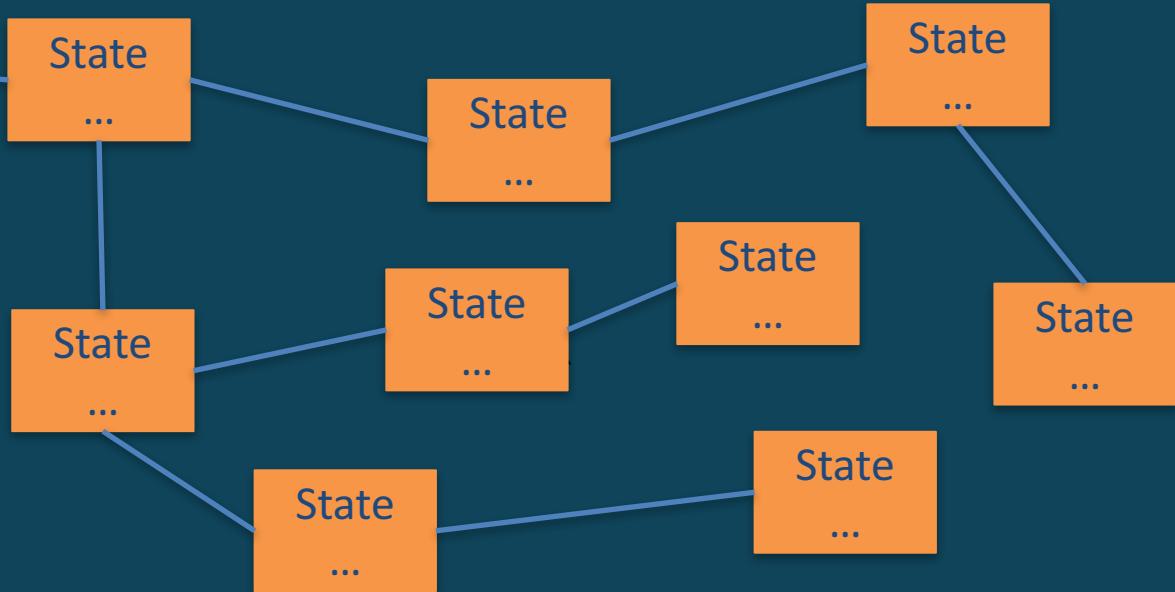
The stateful part - the knowledge



Lightbend

System Composable Identity

State Changes
Yield
Identity



Lightbend

Cheers!



Lightbend



Please

Remember to
rate this session

Thank you!



Join the conversation #scaladays



Join the conversation #scaladays



 Join the conversation #scaladays