FROM MICROLITHS TO MICROSYSTEMS

JONAS BONÉR

@JBONER

LIGHTBEND

WE HAVE BEEN SPOILED BY

THE ALMIGHTY

MONOLITH


KNOCK, KNOCK. WHO'S THERE?
REALITY


WE CAN'T MAKE THE HORSE FASTER
(5)
GEO. H. HOLZBOG
JEFFERSONVILLE, IND.
ASSOCIATED
PHOTO CO


WE NEED CARS FOR WHERE WE ARE GOING

BUT DON'T JUST DRINK THE

Kool-Aid®

THINK FOR YOURSELF

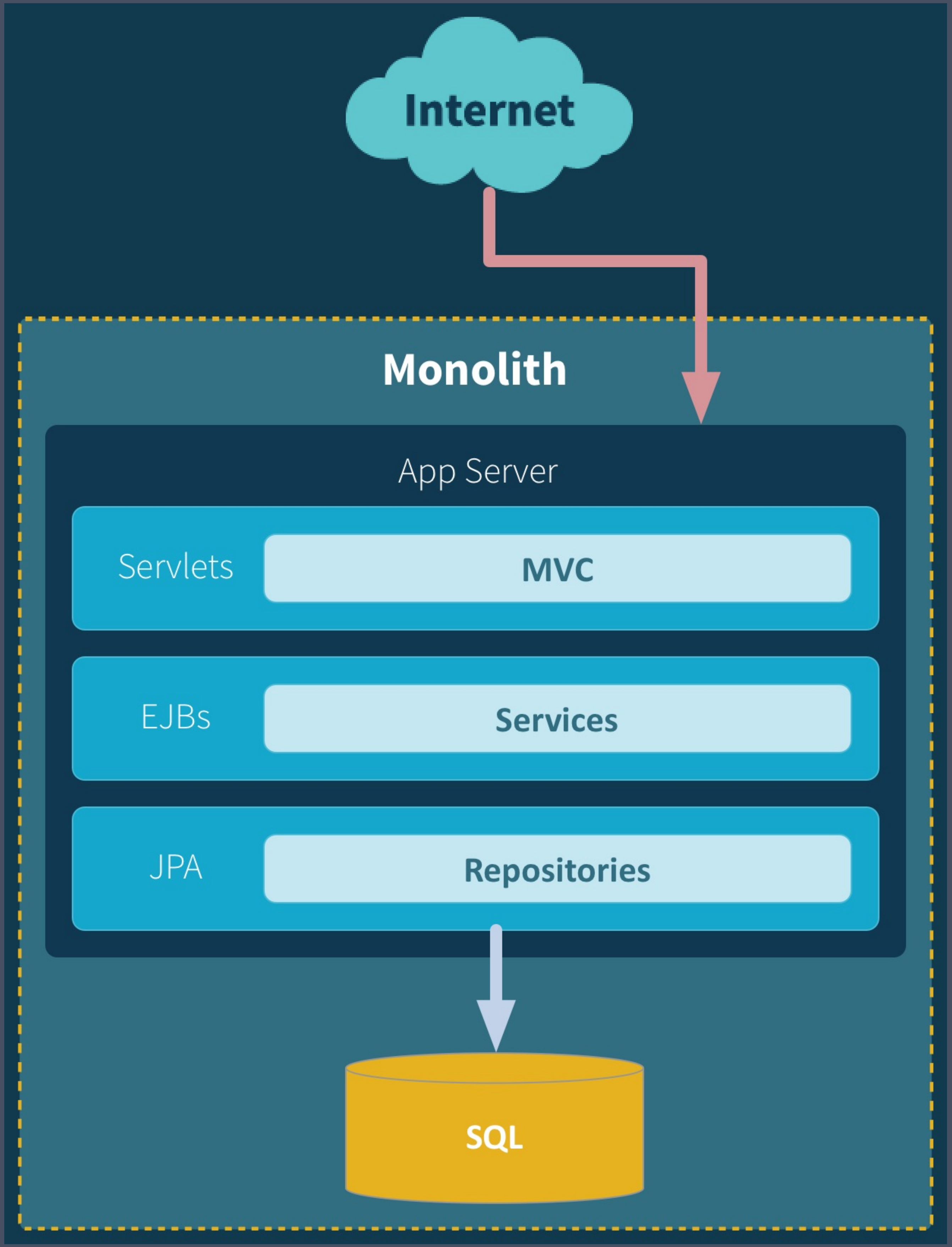
NO ONE WANTS MICROSERVICES

IT'S A NECCESSARY EVIL

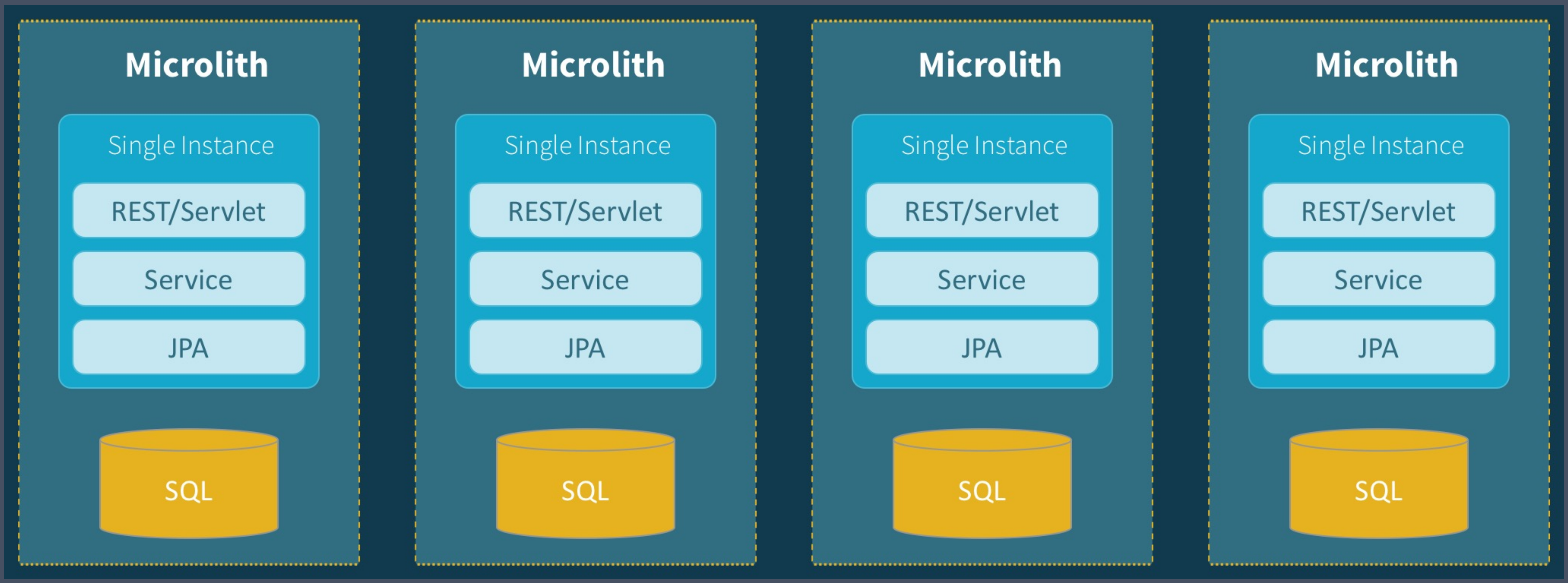
ARCHITECTURAL CONTEXT OF MICROSERVICES:

DISTRIBUTED SYSTEMS

LET'S SAY THAT WE WANT TO SLICE THIS MONOLITH UP



TOO MANY END UP WITH AN ARCHITECTURE LIKE THIS



MICROLITH:

SINGLE INSTANCE MICROSERVICE

-NOT RESILIENT

-NOT ELASTIC

ONE ACTOR IS NO ACTOR.
ACTORS COME IN SYSTEMS.

– CARL HEWITT

MICROSERVICES COME IN SYSTEMS

AS SOON AS WE

EXIT THE SERVICE

WE ENTER A WILD OCEAN OF

NON-DETERMINISM

THE WORLD OF

DISTRIBUTED SYSTEMS

SYSTEMS NEED TO EXPLOIT REALITY

EMBRACE REALITY
AND ITS CONSTRAINTS
SHALL SET YOU FREE

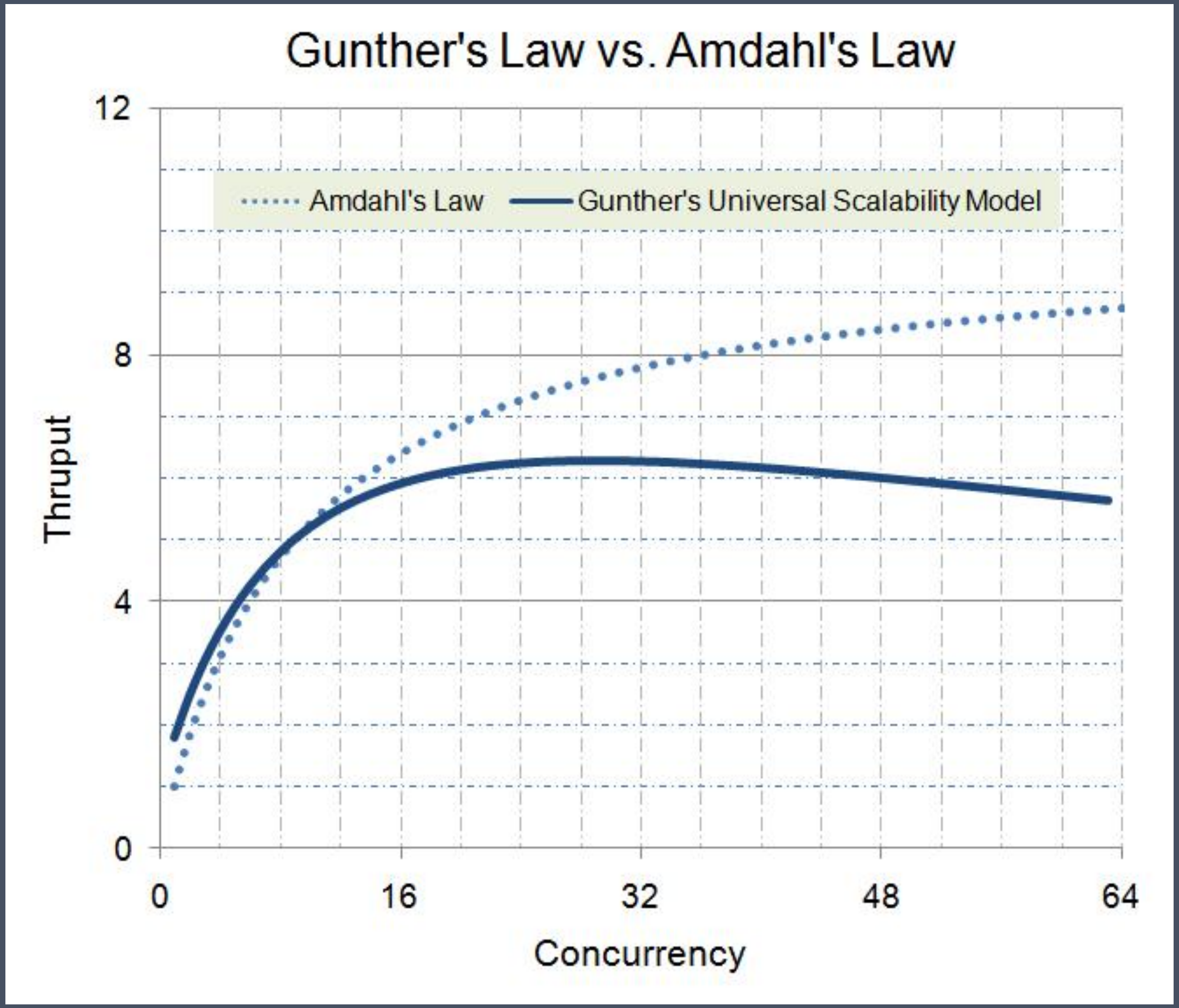
INFORMATION HAS LATENCY

THE CONTENTS OF A MESSAGE ARE ALWAYS FROM THE PAST! THEY ARE NEVER "NOW."

– PAT HELLAND

WE ARE ALWAYS LOOKING INTO THE PAST

THE COST OF MAINTAINING THE ILLUSION OF AN ABSOLUTE NOW



AS LATENCY GETS HIGHER, THE ILLUSION CRACKS EVEN MORE

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

STRIVE TO MINIMIZE

COUPLING & COMMUNICATION

WORDS ARE VERY UNECESSARY.
THEY CAN ONLY DO HARM.
ENJOY THE SILENCE.

- ENJOY THE SILENCE BY MARTIN GORE (DEPECHE MODE)

SILENCE IS NOT ONLY GOLDEN, IT IS SELDOM MISQUOTED.

- BOB MONKHOUSE

WE HAVE TO RELY ON

EVENTUAL CONSISTENCY

BUT RELAX

IT'S HOW THE WORLD WORKS

NO ONE WANTS

EVENTUAL CONSISTENCY.

IT'S A NECESSARY EVIL.

IT'S NOT COOL. IT'S USEFUL.

TWO HELPFUL TOOLS

1. REACTIVE DESIGN
2. EVENTS-FIRST DDD

REACTIVE PROGRAMMING
VS
REACTIVE SYSTEMS

REACTIVE PROGRAMMING CAN HELP US MAKING THE INDIVIDUAL SERVICE INSTANCE HIGHLY PERFORMANT AND EFFICIENT

REACTIVE SYSTEMS CAN HELP US BUILDING DISTRIBUTED SYSTEMS THAT ARE

ELASTIC AND RESILIENT

GO
ASYNCHRONOUS

Viktor Klang
@viktorklang

Following

> "To get consistent & fast response in a real time system, you've to work asynchronously [...] never wait on something happening." @hintjens

RETWEETS 6 LIKES 6

12:44 PM - 21 Sep 2016

ASYNC IO–IS ABOUT NOT BLOCKING THREADS

ASYNC COMM–IS ABOUT NOT BLOCKING REQUESTS

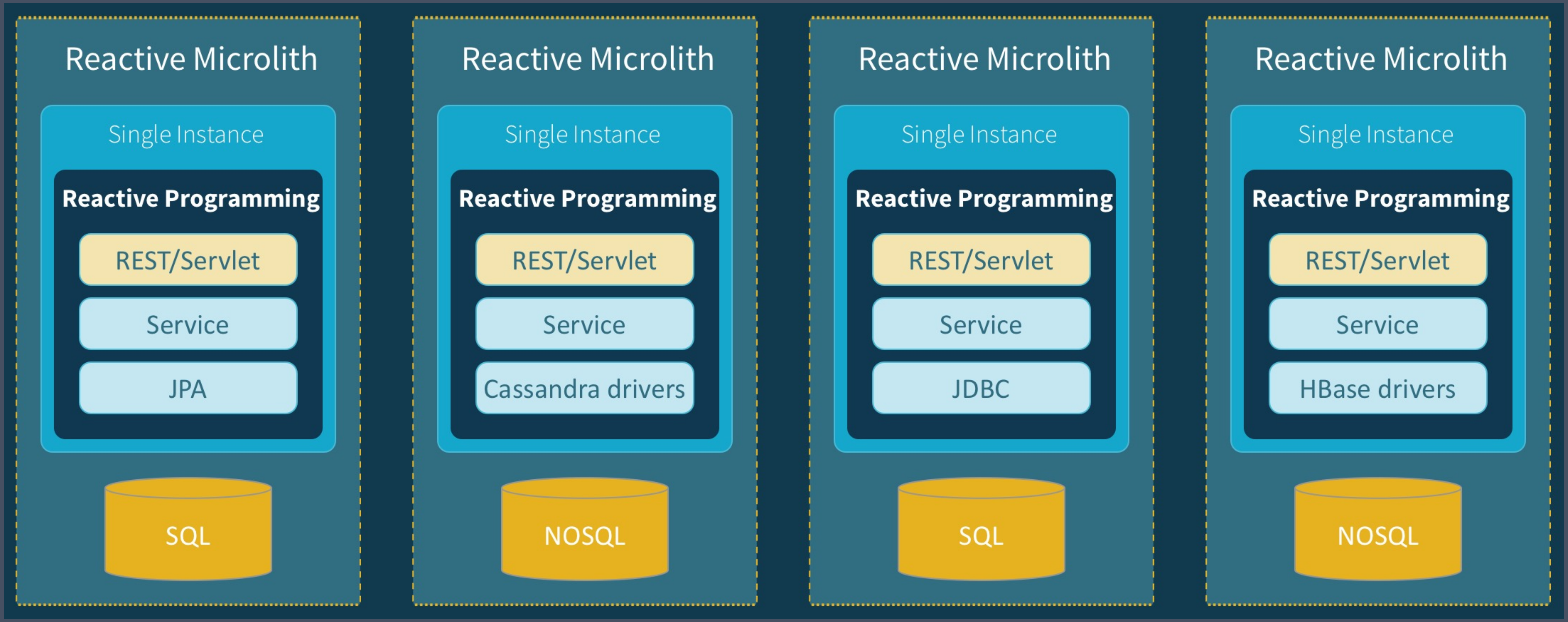
GO ASYNCHRONOUS & NON-BLOCKING

- MORE EFFICIENT USE OF RESOURCES
- MINIMIZES CONTENTION ON SHARED RESOURCES

ALWAYS APPLY BACK-PRESSURE

A FAST SYSTEM SHOULD NOT OVERLOAD A SLOW SYSTEM

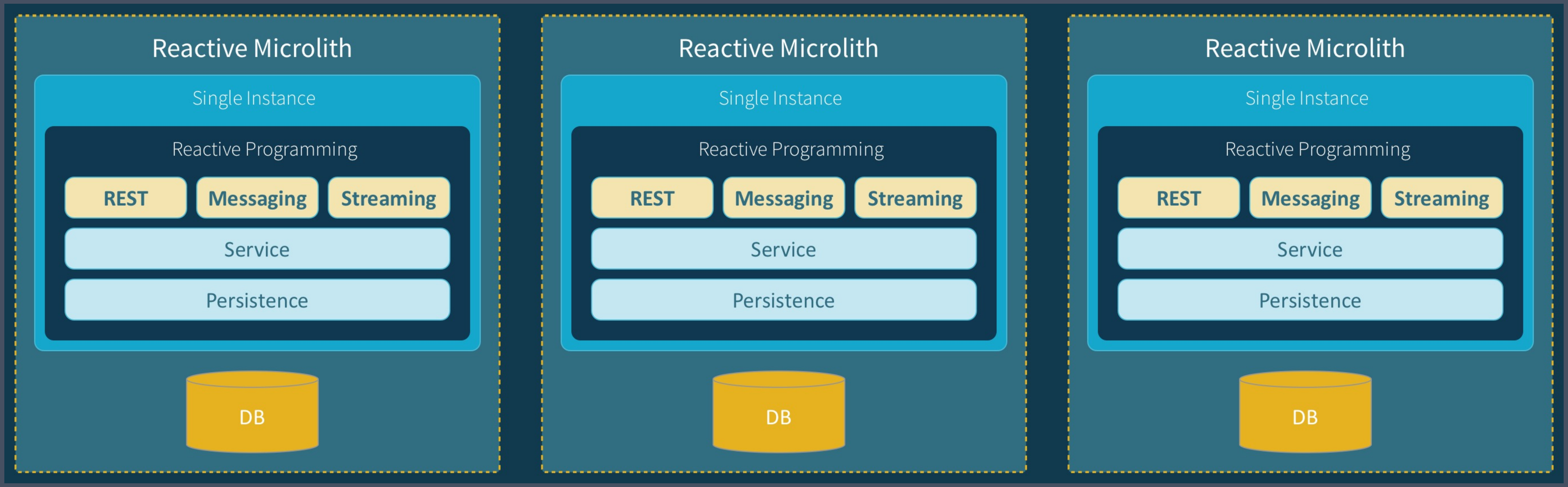
LET'S APPLY REACTIVE PROGRAMMING TO OUR MICROLITHS



WE NEED TO EXTEND OUR MODELS OF

COMMUNICATION

1. ASYNCHRONOUS MESSAGING (N-M)
2. STREAMING (1-1)
3. SYNCHRONOUS REQUEST/ REPLY (1-1)



Reactive Microlith
Single Instance
Reactive Programming
REST
Messaging
Streaming
Service
Persistence
DB
Reactive Microlith
Single Instance
Reactive Programming
REST
Messaging
Streaming
Service
Persistence
DB
Reactive Microlith
Single Instance
Reactive Programming
REST
Messaging
Streaming
Service
Persistence
DB

WE'RE GETTING THERE, BUT WE STILL HAVE A SINGLE INSTANCE MICROSERVICE

- NOT SCALABLE
- NOT RESILIENT

MICROSERVICES COME AS SYSTEMS

EACH MICROSERVICE

NEEDS BE DESIGNED AS

A DISTRIBUTED SYSTEM

A MICROSYSTEM

WE NEED TO MOVE

FROM MICROLITHS TO MICROSYSTEMS

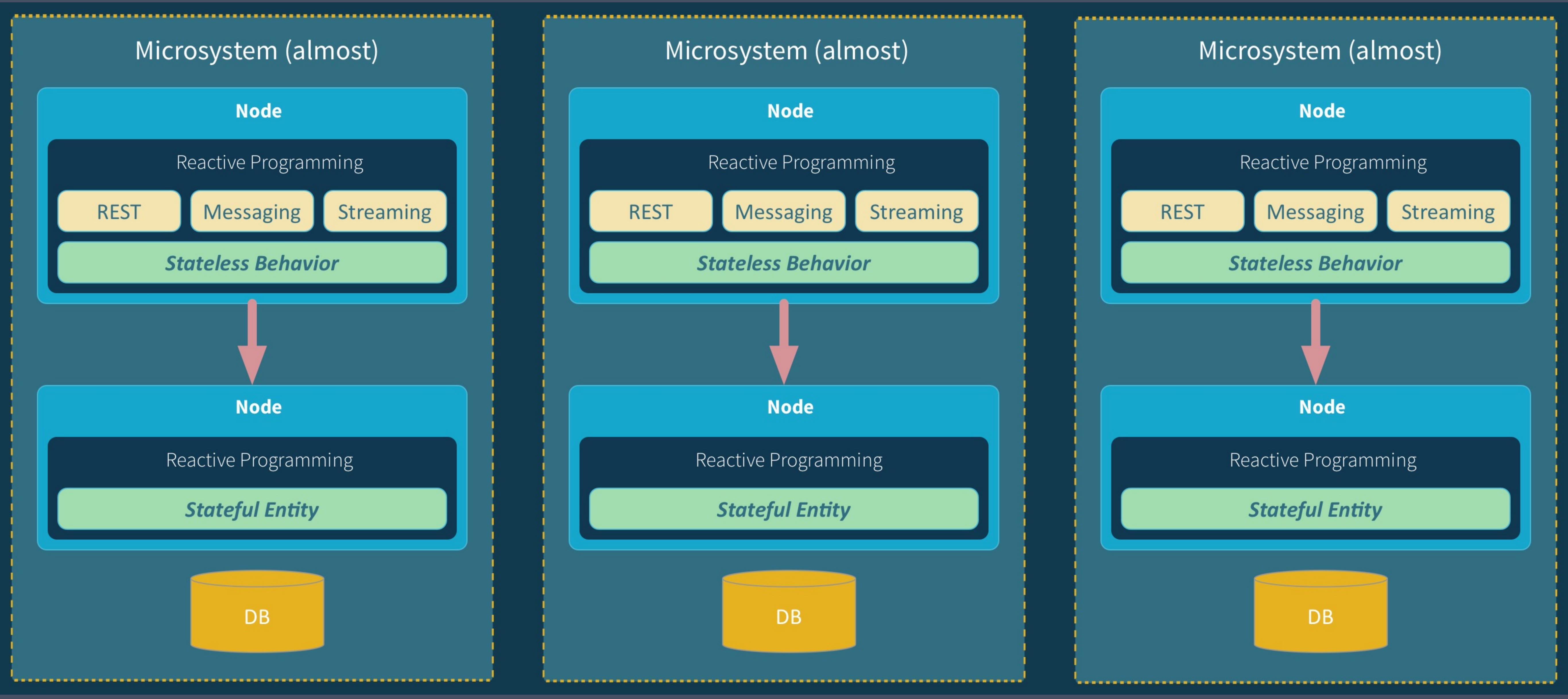
SEPARATE THE

STATELESS BEHAVIOR

FROM THE

STATEFUL ENTITY

TO SCALE THEM INDIVIDUALLY



Microsystem (almost)
Node
Reactive Programming
REST
Messaging
Streaming
Stateless Behavior
Node
Reactive Programming
Stateful Entity
DB
Microsystem (almost)
Node
Reactive Programming
REST
Messaging
Streaming
Stateless Behavior
Node
Reactive Programming
Stateful Entity
DB
Microsystem (almost)
Node
Reactive Programming
REST
Messaging
Streaming
Stateless Behavior
Node
Reactive Programming
Stateful Entity
DB

SCALING (STATELESS) BEHAVIOR

IS EASY

SCALING (STATEFUL) ENTITIES

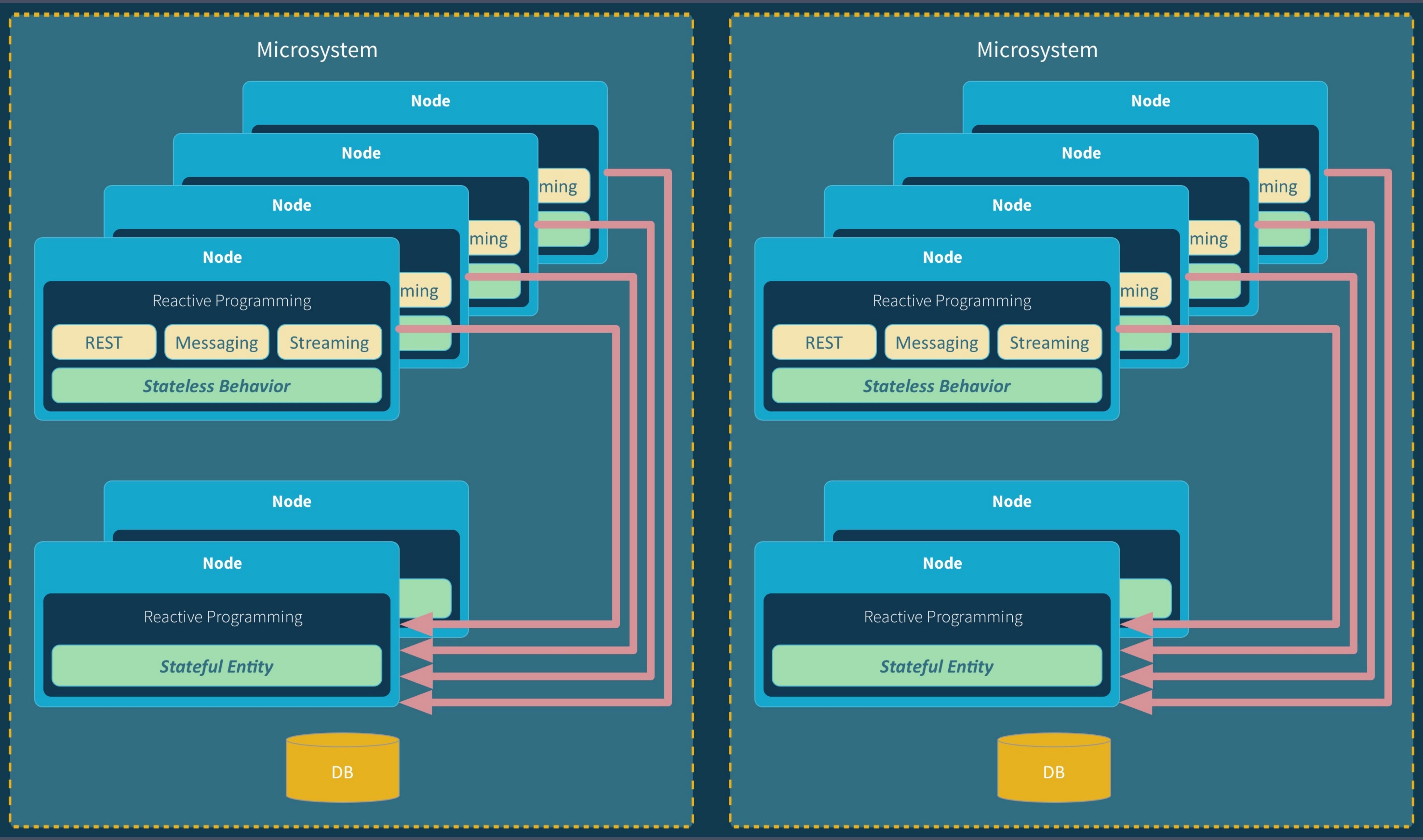
IS HARD


THERE IS NO SUCH THING AS A
"STATELESS" ARCHITECTURE
IT'S JUST SOMEONE ELSE'S PROBLEM

THE ENTITY CAN BECOME AN

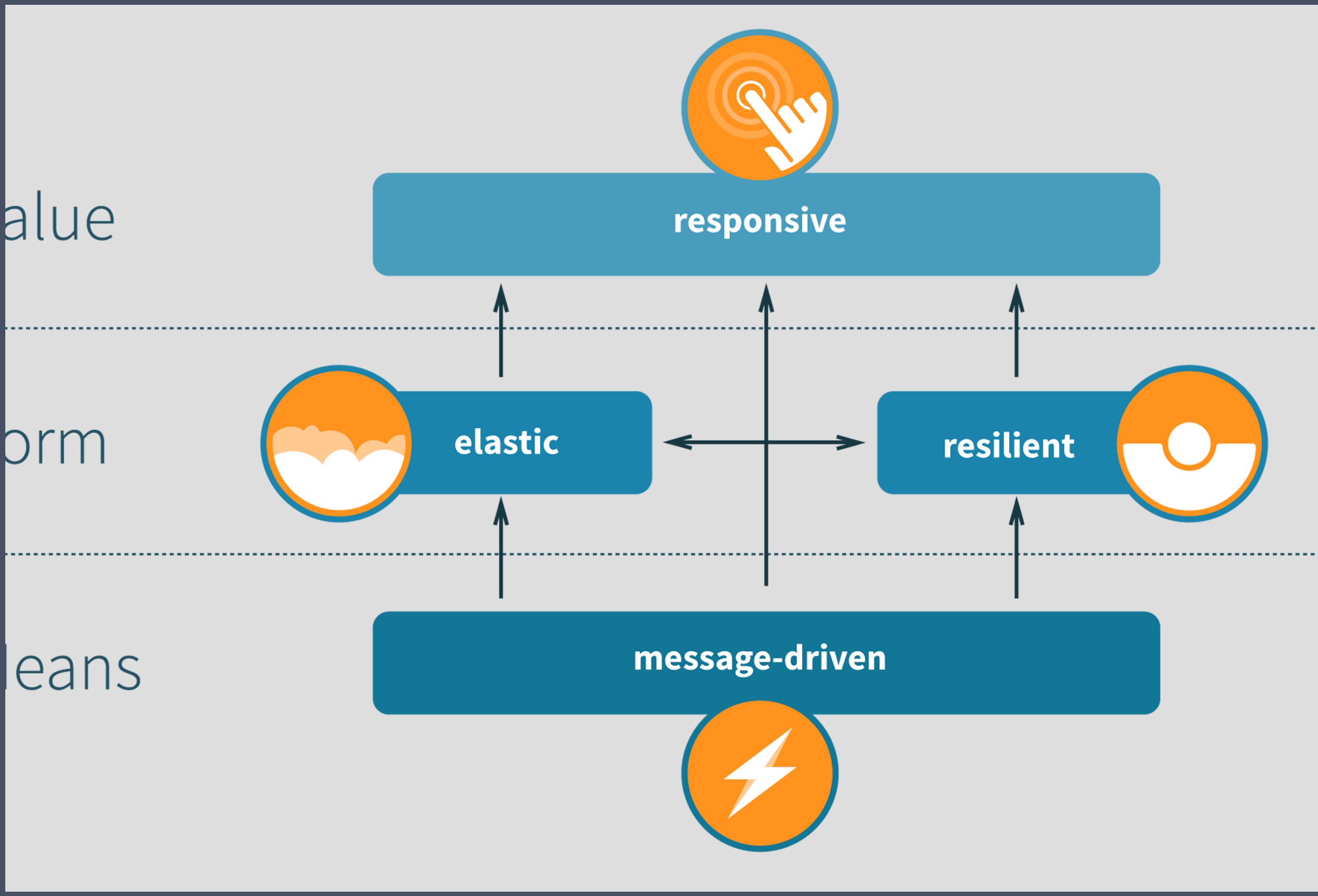
ESCAPE ROUTE FROM REALITY.

A SAFE ISLAND OF DETERMINISM AND STRONG CONSISTENCY


Microsystem
Node
Node
Node
Node
ming
ming
ming
Reactive Programming
REST
Messaging
Streaming
Stateless Behavior
Node
Node
Reactive Programming
Stateful Entity
DB
Microsystem
Node
Node
Node
Node
ming
ming
ming
Reactive Programming
REST
Messaging
Streaming
Stateless Behavior
Node
Node
Reactive Programming
Stateful Entity
DB

REACTIVE SYSTEMS HELPS MAKING THE MICROSERVICE (MICROSYSTEM) RESILIENT AND ELASTICALLY SCALABLE

LEARN MORE AT REACTIVEMANIFESTO.ORG



REACTIVE SYSTEMS ARE BASED ON

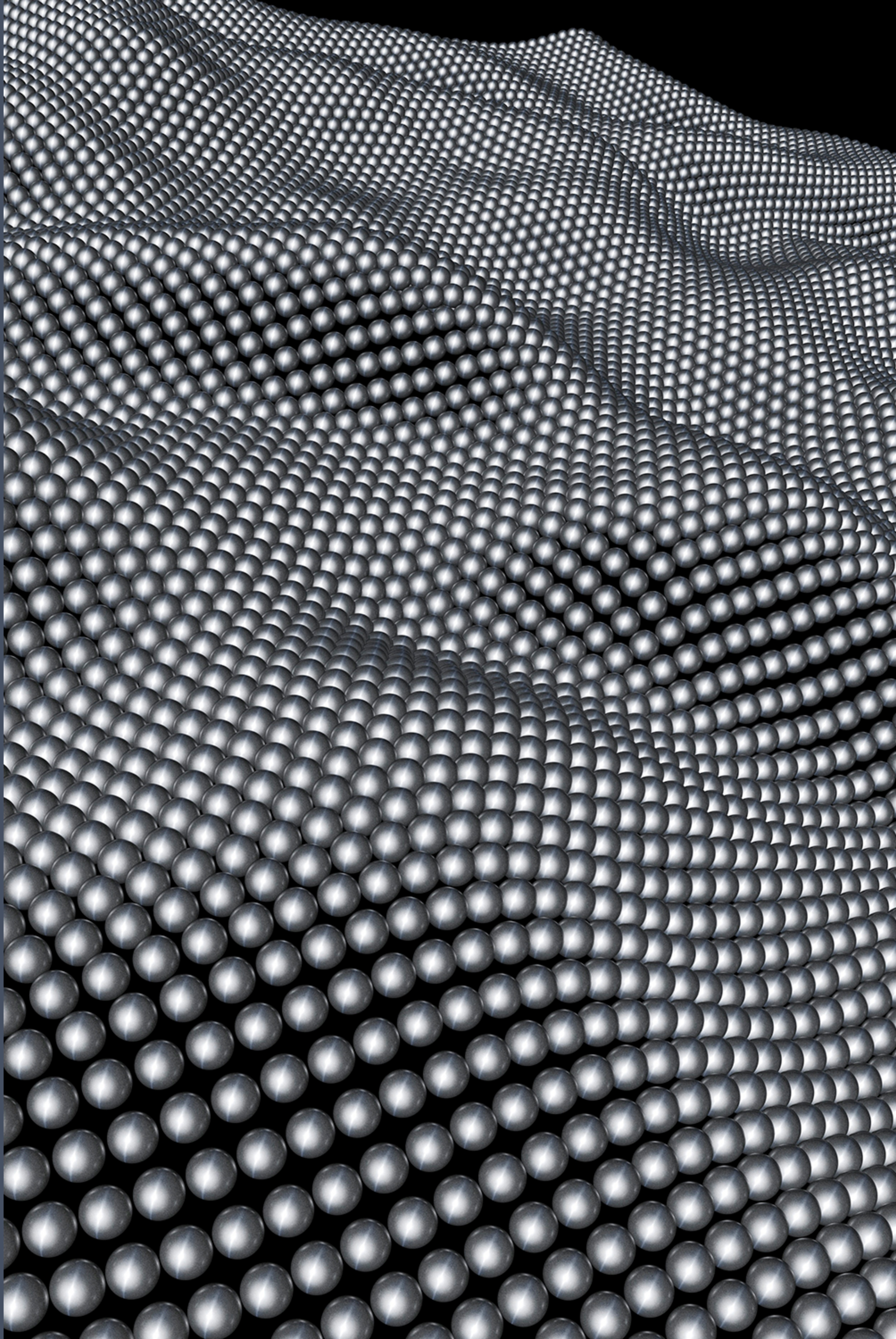
ASYNCHRONOUS MESSAGE-PASSING

ALLOWS DECOUPLING IN

SPACE

AND

TIME



ALLOWS FOR LOCATION TRANSPARENCY

ONE COMMUNICATION ABSTRACTION ACROSS ALL DIMENSIONS OF SCALE

CORE ⇒ SOCKET ⇒ CPU ⇒

CONTAINER ⇒ SERVER ⇒

RACK ⇒ DATA CENTER ⇒

SYSTEM

BUT I'LL TAKE MY TIME ANYWHERE. I'M FREE TO SPEAK MY MIND ANYWHERE. AND I'LL REDEFINE ANYWHERE. ANYWHERE I ROAM. WHERE I LAY MY HEAD IS HOME.

– WHEREVER I MAY ROAM BY LARS ULRICH, JAMES HETFIELD (METALLICA)

THE PATH TOWARDS RESILIENCE

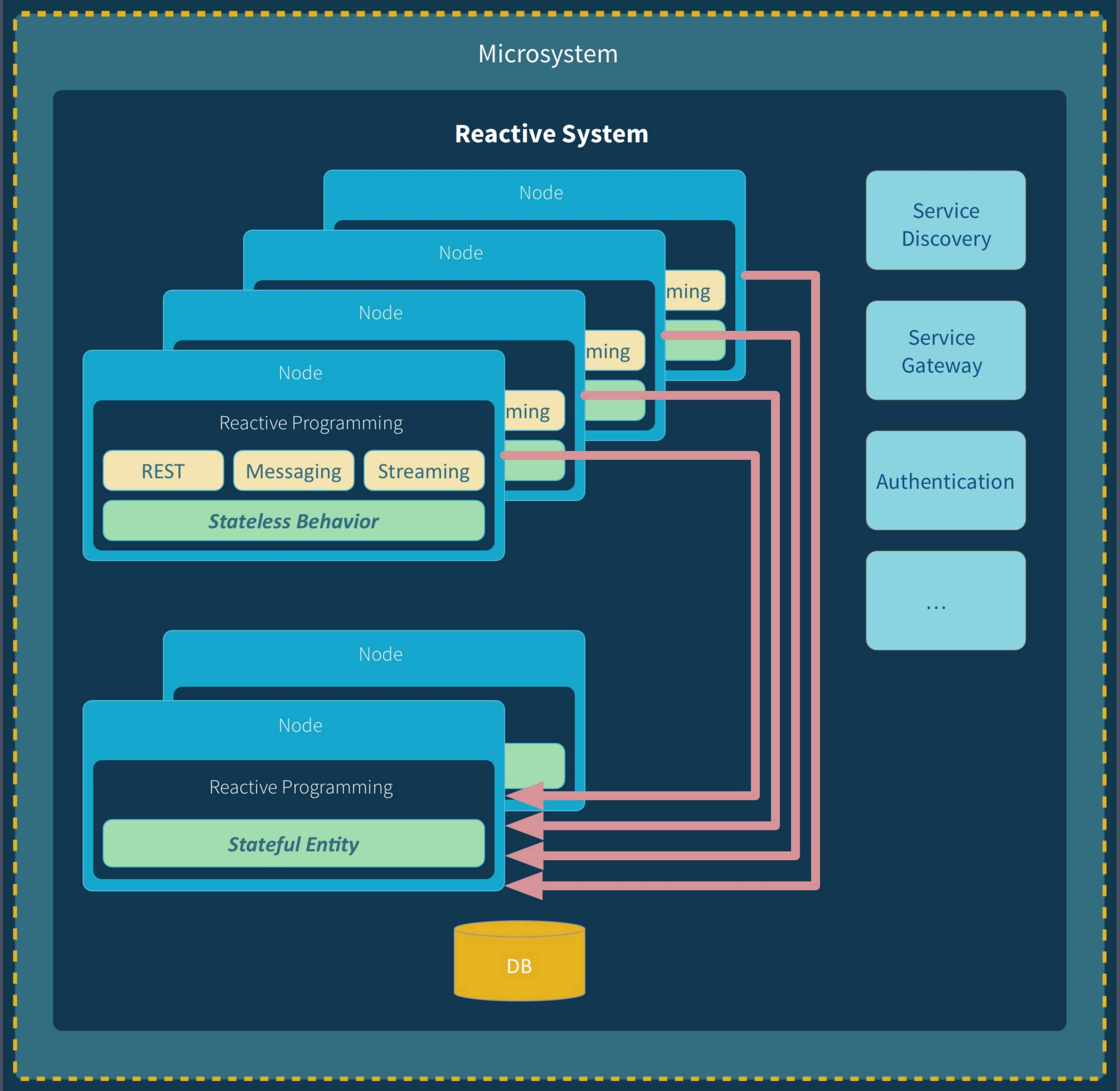
1. DECENTRALIZED ARCHITECTURE
2. BULKHEADING
3. REPLICATION
4. FAILURE DETECTION
5. SUPERVISION

⇒ SELF-HEALING SYSTEMS

THE PATH TOWARDS ELASTICITY

1. DECENTRALIZED ARCHITECTURE
2. EPIDEMIC GOSSIP PROTOCOLS
3. SELF-ORGANIZATION
4. LOCATION TRANSPARENCY

⇒ ELASTIC SYSTEMS



Microsystem
Reactive System
Node
Node
Node
Node
ming
ming
ming
Reactive Programming
REST
Messaging
Streaming
Stateless Behavior
Node
Node
Reactive Programming
Stateful Entity
DB
Service Discovery
Service Gateway
Authentication
...

THINK IN TERMS OF CONSISTENCY BOUNDARIES

INSIDE DATA: OUR CURRENT PRESENT (STATE)

OUTSIDE DATA: BLAST FROM THE PAST (FACTS)

BETWEEN SERVICES: HOPE FOR THE FUTURE (COMMANDS)

— PAT HELLAND, DATA ON THE INSIDE VS DATA ON THE OUTSIDE

PRACTICE

EVENTS-FIRST

DOMAIN-DRIVEN DESIGN

DON'T FOCUS ON THE THINGS–THE NOUNS
FOCUS ON WHAT HAPPENS–THE EVENTS

LET THE

EVENTS DEFINE

THE BOUNDED CONTEXT

EVENTS REPRESENT FACTS

TO CONDENSE FACT FROM THE VAPOR OF NUANCE

- NEAL STEPHENSON, SNOW CRASH

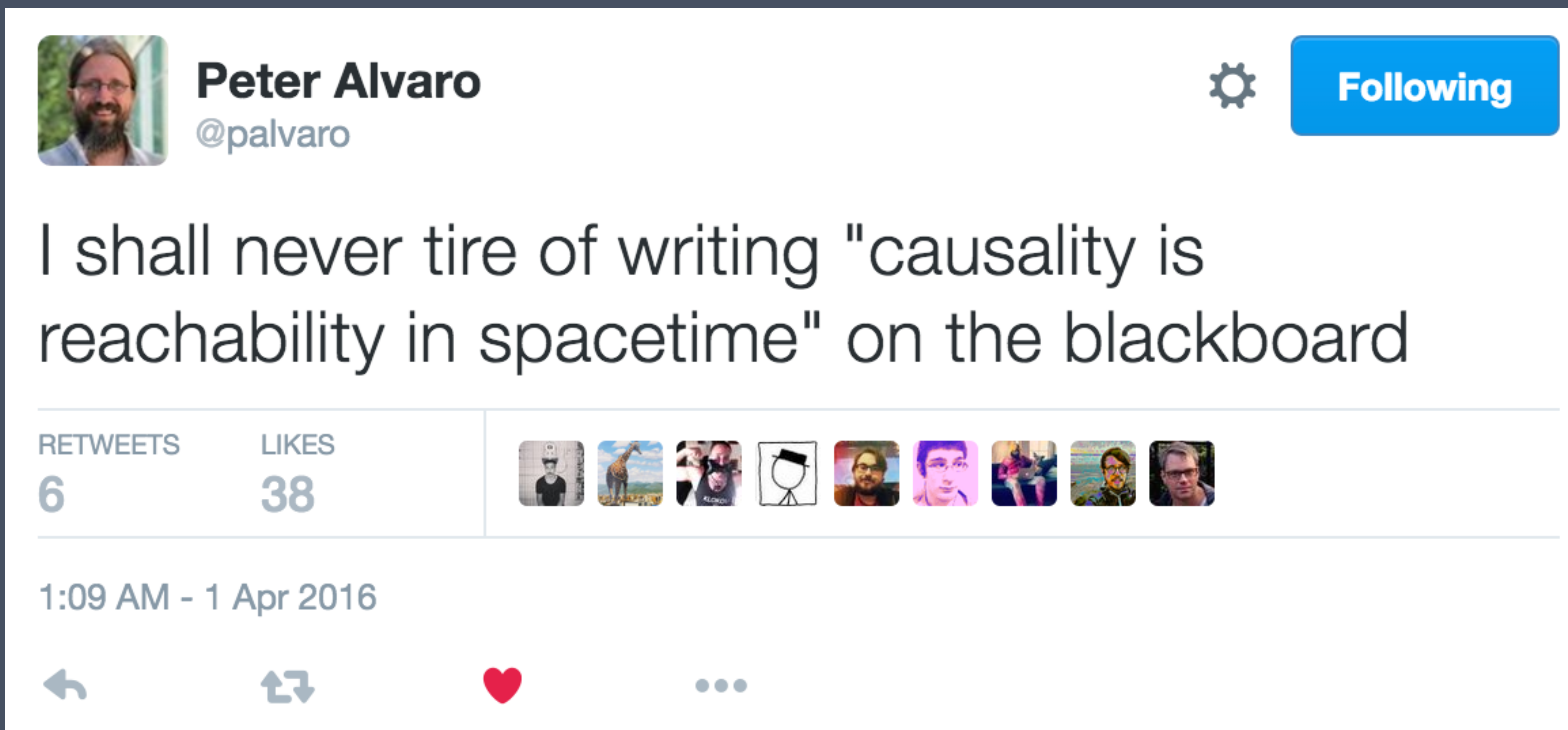
...AND WE ARE NOT TALKING ABOUT

ALTERNATIVE FACTS


WHAT ARE THE
FACTS?

UNDERSTAND HOW FACTS ARE CAUSALLY RELATED

HOW FACTS FLOW IN THE SYSTEM



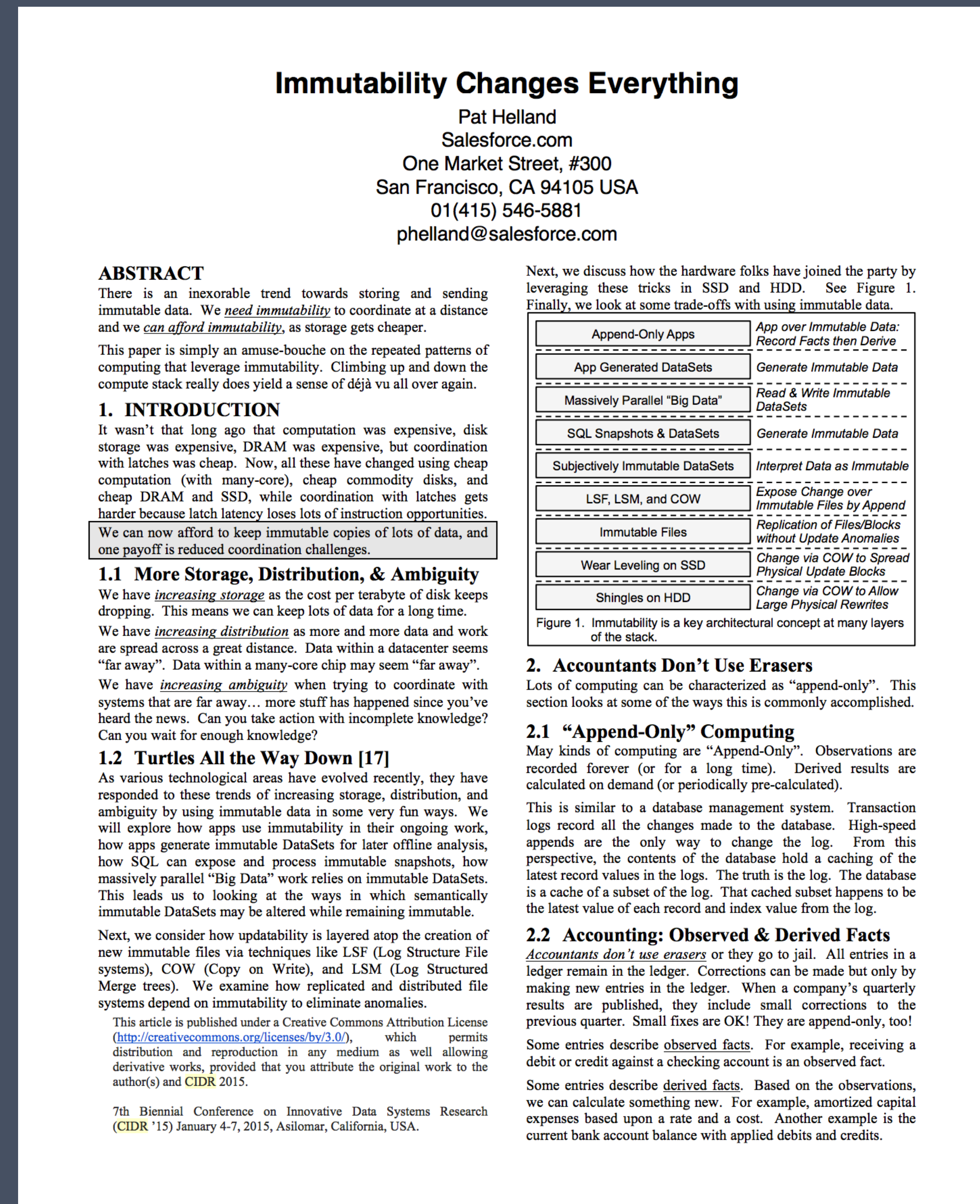
WE NEED TO

CONTAIN MUTABLE STATE &

PUBLISH FACTS

THE TRUTH IS THE LOG.
THE DATABASE IS A CACHE OF A SUBSET OF THE LOG.

– PAT HELLAND



Immutability Changes Everything

Pat Helland
Salesforce.com
One Market Street, #300
San Francisco, CA 94105 USA
01(415) 546-5881
phelland@salesforce.com

ABSTRACT

There is an inexorable trend towards storing and sending immutable data. We *need immutability* to coordinate at a distance and we *can afford immutability*, as storage gets cheaper.

This paper is simply an amuse-bouche on the repeated patterns of computing that leverage immutability. Climbing up and down the compute stack really does yield a sense of déjà vu all over again.

1. INTRODUCTION

It wasn't that long ago that computation was expensive, disk storage was expensive, DRAM was expensive, but coordination with latches was cheap. Now, all these have changed using cheap computation (with many-core), cheap commodity disks, and cheap DRAM and SSD, while coordination with latches gets harder because latch latency loses lots of instruction opportunities.

We can now afford to keep immutable copies of lots of data, and one payoff is reduced coordination challenges.

1.1 More Storage, Distribution, & Ambiguity

We have *increasing storage* as the cost per terabyte of disk keeps dropping. This means we can keep lots of data for a long time.

We have *increasing distribution* as more and more data and work are spread across a great distance. Data within a datacenter seems "far away". Data within a many-core chip may seem "far away".

We have *increasing ambiguity* when trying to coordinate with systems that are far away… more stuff has happened since you've heard the news. Can you take action with incomplete knowledge? Can you wait for enough knowledge?

1.2 Turtles All the Way Down [17]

As various technological areas have evolved recently, they have responded to these trends of increasing storage, distribution, and ambiguity by using immutable data in some very fun ways. We will explore how apps use immutability in their ongoing work, how apps generate immutable DataSets for later offline analysis, how SQL can expose and process immutable snapshots, how massively parallel "Big Data" work relies on immutable DataSets. This leads us to looking at the ways in which semantically immutable DataSets may be altered while remaining immutable.

Next, we consider how updatability is layered atop the creation of new immutable files via techniques like LSF (Log Structure File systems), COW (Copy on Write), and LSM (Log Structured Merge trees). We examine how replicated and distributed file systems depend on immutability to eliminate anomalies.

7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January 4-7, 2015, Asilomar, California, USA.

Next, we discuss how the hardware folks have joined the party by leveraging these tricks in SSD and HDD. See Figure 1. Finally, we look at some trade-offs with using immutable data.

Append-Only Apps	*App over Immutable Data: Record Facts then Derive*
App Generated DataSets	*Generate Immutable Data*
Massively Parallel "Big Data"	*Read & Write Immutable DataSets*
SQL Snapshots & DataSets	*Generate Immutable Data*
Subjectively Immutable DataSets	*Interpret Data as Immutable*
LSF, LSM, and COW	*Expose Change over Immutable Files by Append*
Immutable Files	*Replication of Files/Blocks without Update Anomalies*
Wear Leveling on SSD	*Change via COW to Spread Physical Update Blocks*
Shingles on HDD	*Change via COW to Allow Large Physical Rewrites*

Figure 1. Immutability is a key architectural concept at many layers of the stack.

2. Accountants Don't Use Erasers

Lots of computing can be characterized as "append-only". This section looks at some of the ways this is commonly accomplished.

2.1 "Append-Only" Computing

May kinds of computing are "Append-Only". Observations are recorded forever (or for a long time). Derived results are calculated on demand (or periodically pre-calculated).

This is similar to a database management system. Transaction logs record all the changes made to the database. High-speed appends are the only way to change the log. From this perspective, the contents of the database hold a caching of the latest record values in the logs. The truth is the log. The database is a cache of a subset of the log. That cached subset happens to be the latest value of each record and index value from the log.

2.2 Accounting: Observed & Derived Facts

Accountants don't use erasers or they go to jail. All entries in a ledger remain in the ledger. Corrections can be made but only by making new entries in the ledger. When a company's quarterly results are published, they include small corrections to the previous quarter. Small fixes are OK! They are append-only, too!

Some entries describe observed facts. For example, receiving a debit or credit against a checking account is an observed fact.

Some entries describe derived facts. Based on the observations, we can calculate something new. For example, amortized capital expenses based upon a rate and a cost. Another example is the current bank account balance with applied debits and credits.

CR~~UD~~

IS DEAD


FAVOR
EVENT LOGGING

THE LOG

IS A DATABASE OF THE PAST

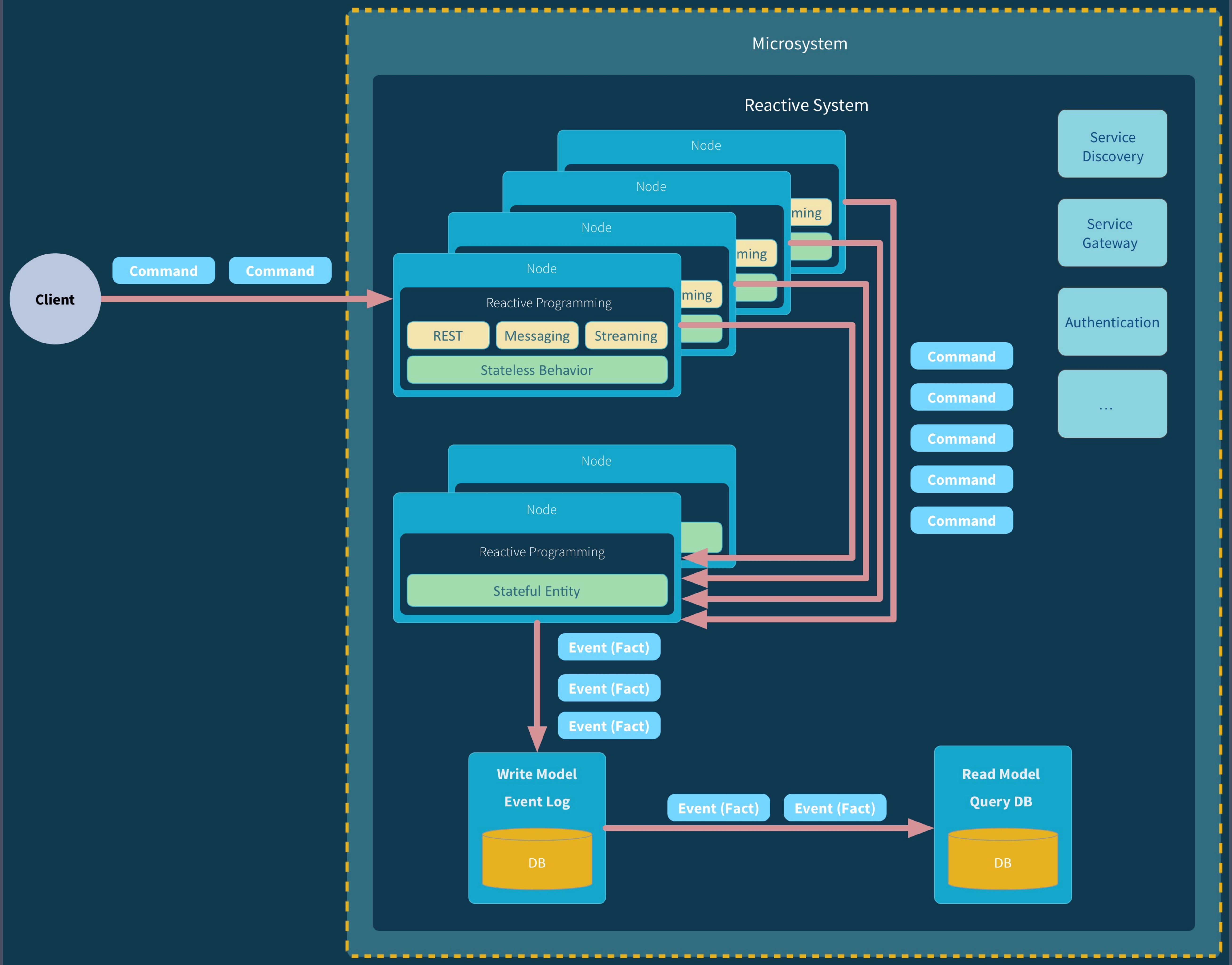
NOT JUST A DATABASE OF THE PRESENT

EVENT LOGGING AVOIDS THE INFAMOUS

OBJECT-RELATIONAL

IMPEDENCE MISMATCH

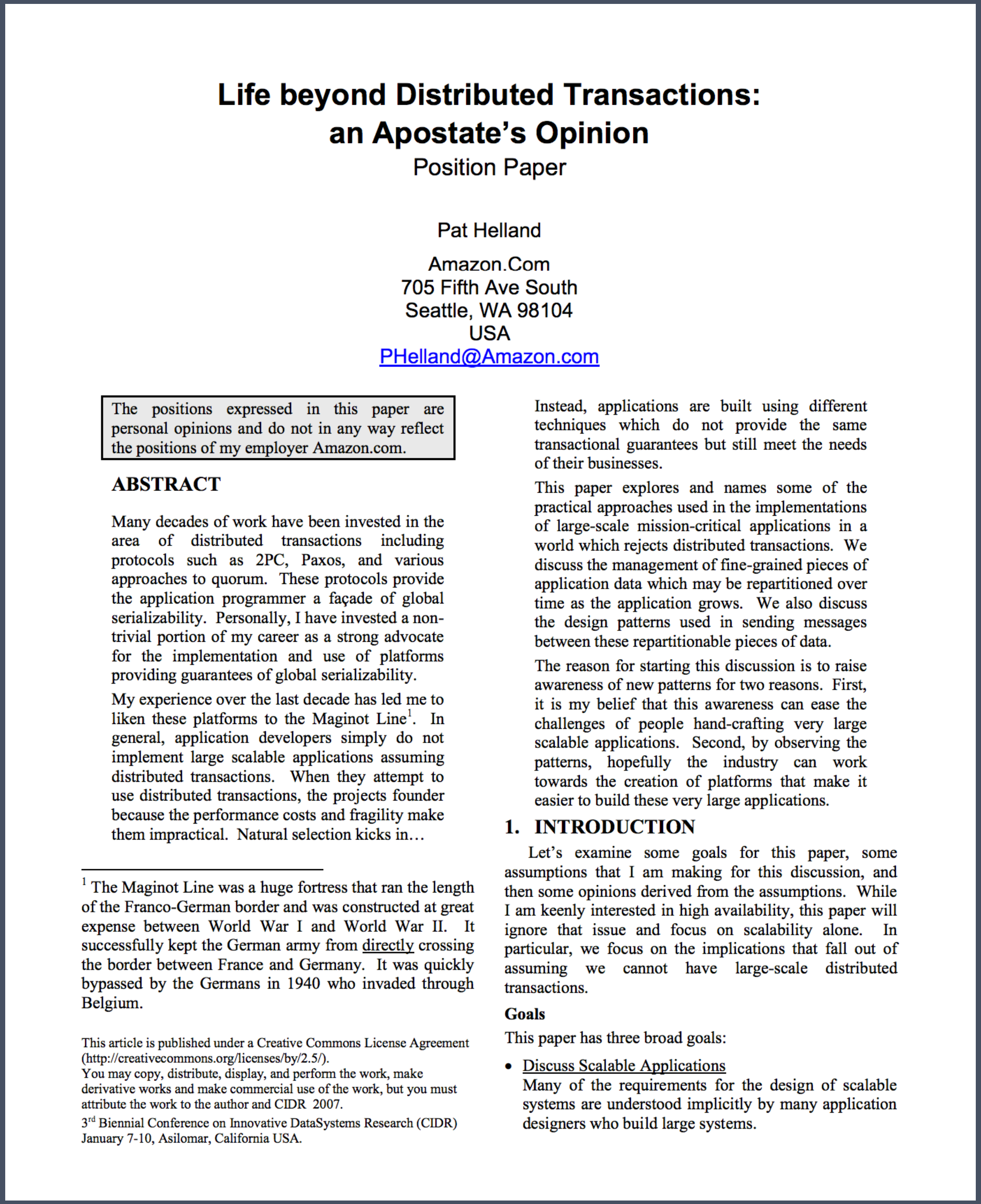
UNTANGLE THE READ & WRITE MODELS WITH CQRS & EVENT SOURCING


Microsystem
Reactive System
Client
Command
Command
Node
Node
Node
Node
Reactive Programming
REST
Messaging
Streaming
Stateless Behavior
ming
ming
ming
Service Discovery
Service Gateway
Authentication
...
Command
Command
Command
Command
Command
Node
Node
Reactive Programming
Stateful Entity
Event (Fact)
Event (Fact)
Event (Fact)
Write Model
Event Log
DB
Event (Fact)
Event (Fact)
Read Model
Query DB
DB

BUT WHAT ABOUT
TRANSACTIONS?

IN GENERAL, APPLICATION DEVELOPERS SIMPLY DO NOT IMPLEMENT LARGE SCALABLE APPLICATIONS ASSUMING DISTRIBUTED TRANSACTIONS.

– PAT HELLAND



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.

USE A PROTOCOL OF

GUESS.
APOLOGIZE.
COMPENSATE.

IT'S HOW THE WORLD WORKS

IN SUMMARY

1. DON'T BUILD MICROLITHS
2. MICROSERVICES COME IN (DISTRIBUTED) SYSTEMS
3. MICROSERVICES COME AS (MICRO)SYSTEMS
4. EMBRACE THE REACTIVE PRINCIPLES
5. EMBRACE EVENT-FIRST DDD & PERSISTENCE
6. PROFIT!

TRY THE

LAGOM

MICROSERVICES FRAMEWORK

POWERED BY AKKA & PLAY

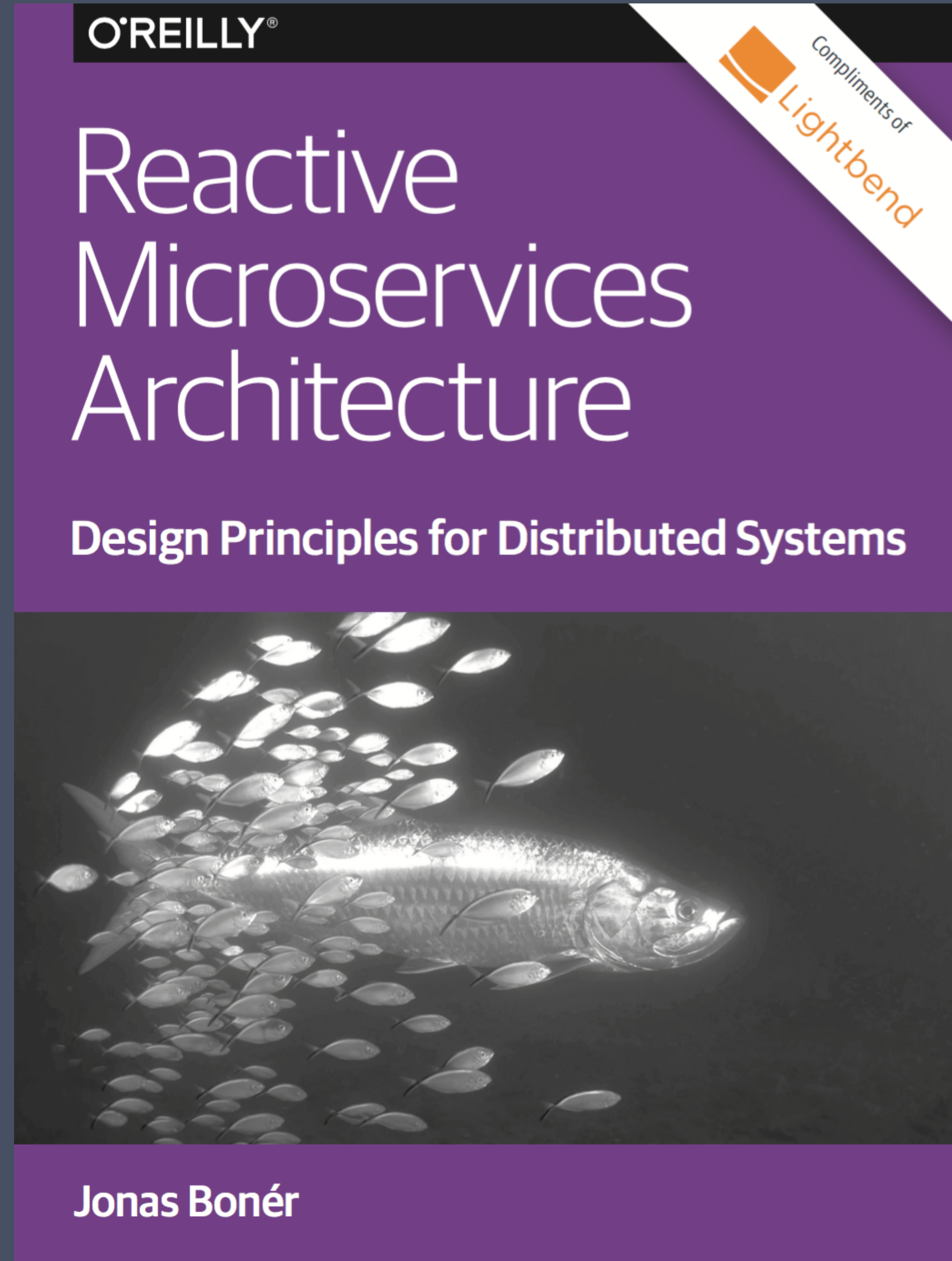
LAGOMFRAMEWORK.COM

lagom

LEARN MORE

DOWNLOAD MY BOOK FOR FREE AT:

BIT.LY/REACTIVE-MICROSERVICES-ARCHITECTURE



THANK

YOU

Lightbend