Why Reactive?

Operating in an Ever Changing World

Duncan K. DeVore
Typesafe: Senior Engineer
Twitter: @ironfish

Typesafe

What is Reactive?

Typesafe

What is Reactive?

***Reactive* : "Readily responsive to stimulus."[1]**

— Reactive Manifesto

— Published on September 16 2014. (v2.0)

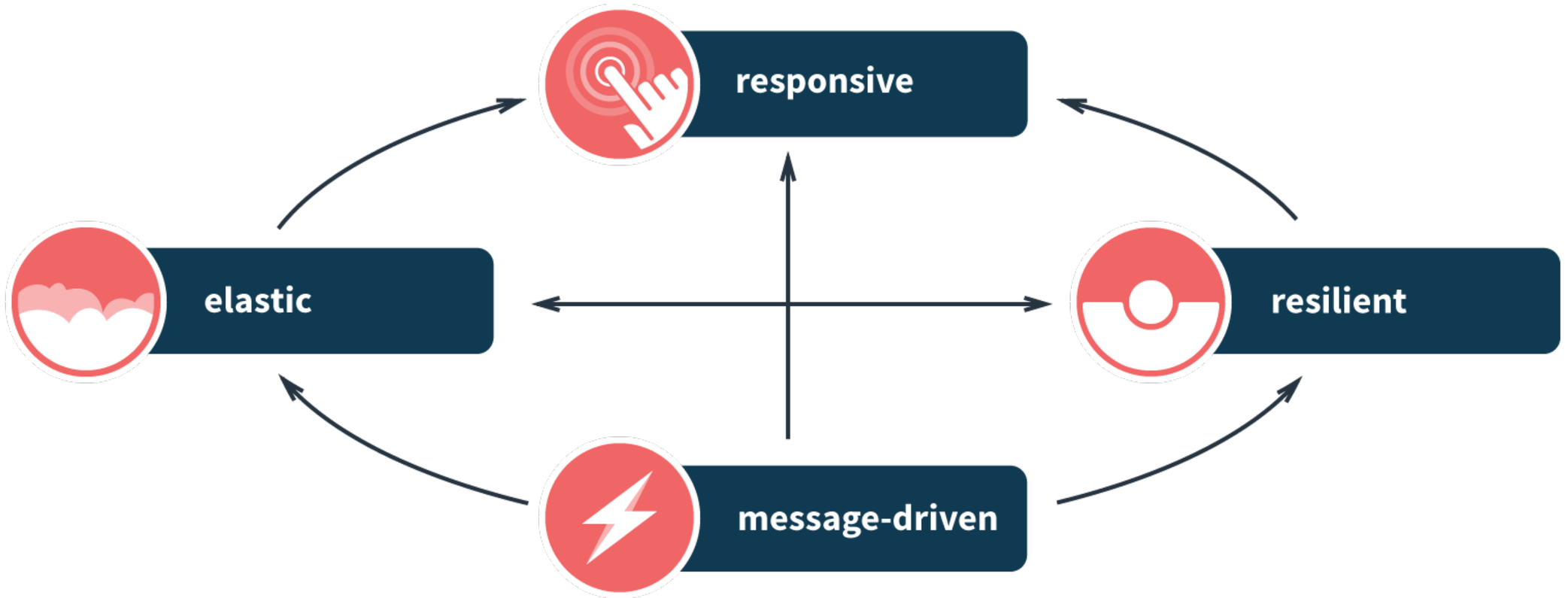
— Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson

— 11K + Signatures

[1] Merriam Webster Dictionary

Typesafe

What is Reactive?



Typesafe

What is Reactive?

Message *: "A verbal, written, or recorded communication sent to or left for a recipient who cannot be contacted directly."*[1]

Driven *: "Operated, moved, or controlled by a specified person or source of power."*[1]

[1] Merriam Webster Dictionary

Typesafe

What is Reactive?

***Elastic* : "Able to return to an original shape or size after being stretched, squeezed, etc.; able to be changed."[1]**

Message-Driven support elasticity.

[1] Merriam Webster Dictionary

Typesafe

What is Reactive?

***Resilient* : "Able to become strong, healthy, or successful again after something bad happens."** [1]

Message-Driven supports resilience.

[1] Merriam Webster Dictionary

What is Reactive?

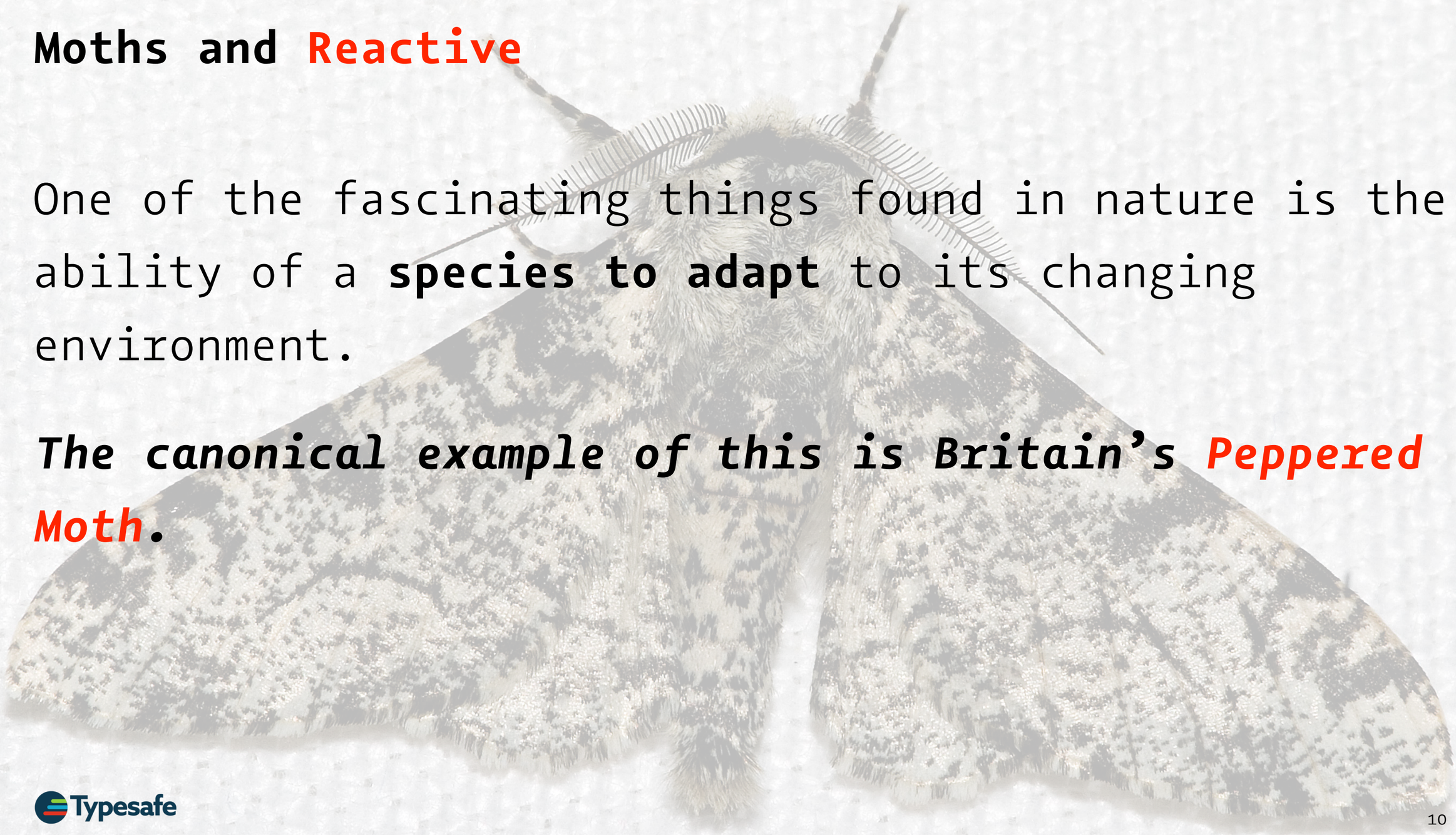
Responsive : "Reacting in a desired or positive way; quick to react or respond."[1]

Responsiveness is built on being **elastic** and **resilient**.

[1] Merriam Webster Dictionary

Moths and Reactive

Moths and Reactive



One of the fascinating things found in nature is the ability of a **species to adapt** to its changing environment.

The canonical example of this is Britain's Peppered Moth.

Typesafe

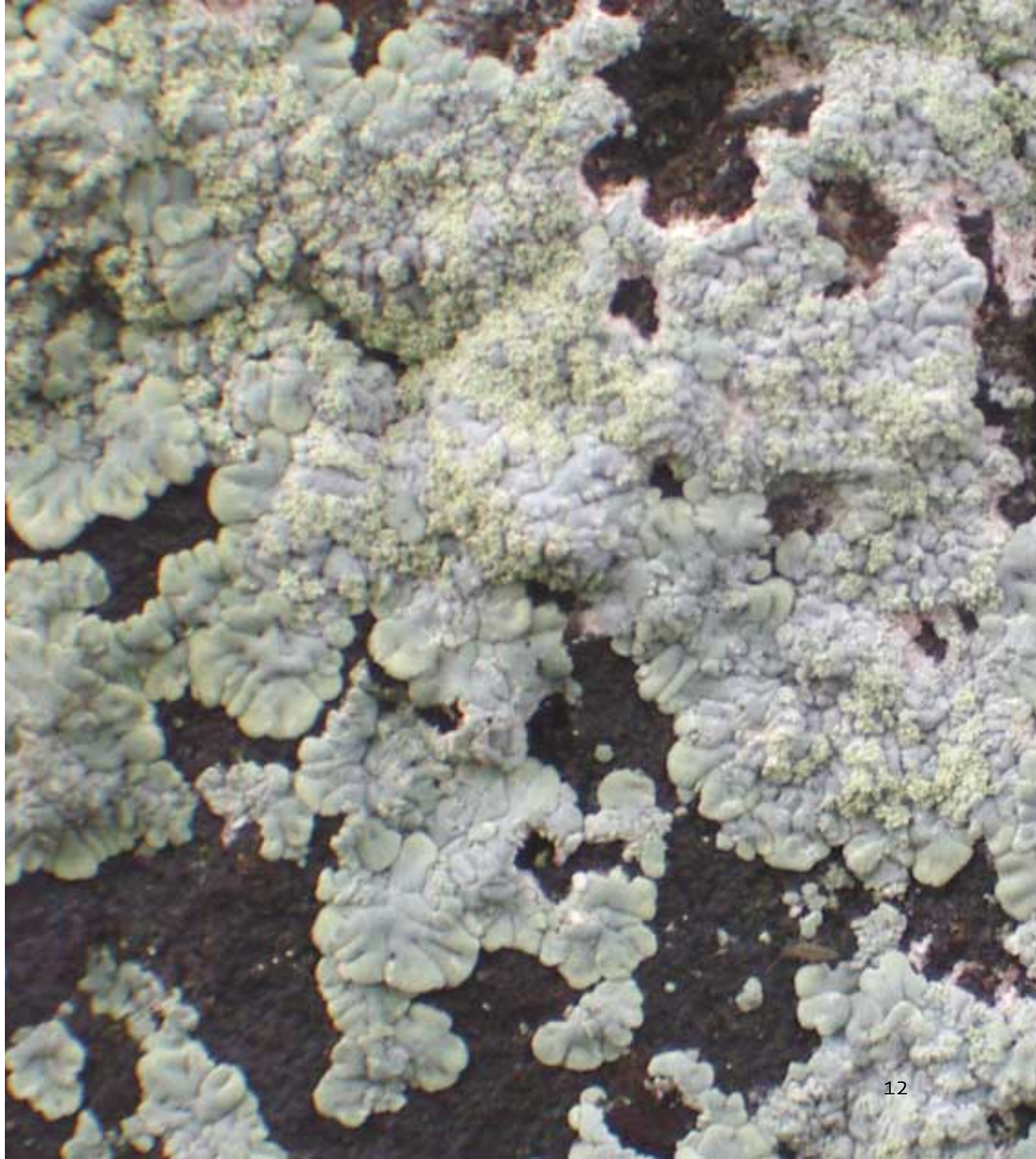
Moths and Reactive

When newly **industrialized** Great Britain became polluted in the nineteenth century, slow-growing, **light-colored** lichens that covered trees **died**, resulting in a **blackening** of the trees bark.

The impact of this was quite profound!

Moths and Reactive

Lichens : "A simple slow-growing plant that typically forms a low crustlike, leaflike, or branching growth on rocks, walls, and trees."



Typesafe

Moths and Reactive

1. Light-colored moths, **camouflaged** and the **majority**
2. Found themselves the **obvious** target of hungry birds
3. Rare dark ones, **blended** into the polluted ecosystem
4. The birds, **changed** from eating **dark** to **light** moths

The dynamics of Britain's moth population changed.

Typesafe

Moths and Reactive

Typesafe

Moths and Reactive

Typesafe

Moths and Reactive

One **mutation** from a single ancestor caused **increased dark** pigment, called melanism[2], allowing the peppered moth to **survive**.

This ability to react on-the-fly is what a reactive system does.

[2] Science AAAS

Typesafe

Moths and Reactive

Message-Driven : the mutation was the message

A mutation is the changing of the **structure of a gene**, resulting in a **variant** form that may be **transmitted** to subsequent generations."[1]

[1] Merriam Webster Dictionary

Typesafe

Moths and Reactive

Elastic : load came in two forms.

1. **Unexpected**, increased pollution
2. **Resultant**, the lack of camouflage

The **mutation** baked into the Peppered moths DNA allowed the moth population to **rebound**.

Moths and Reactive

Resilient : failure came in one form.

1. The **light-colored** moths were easy targets

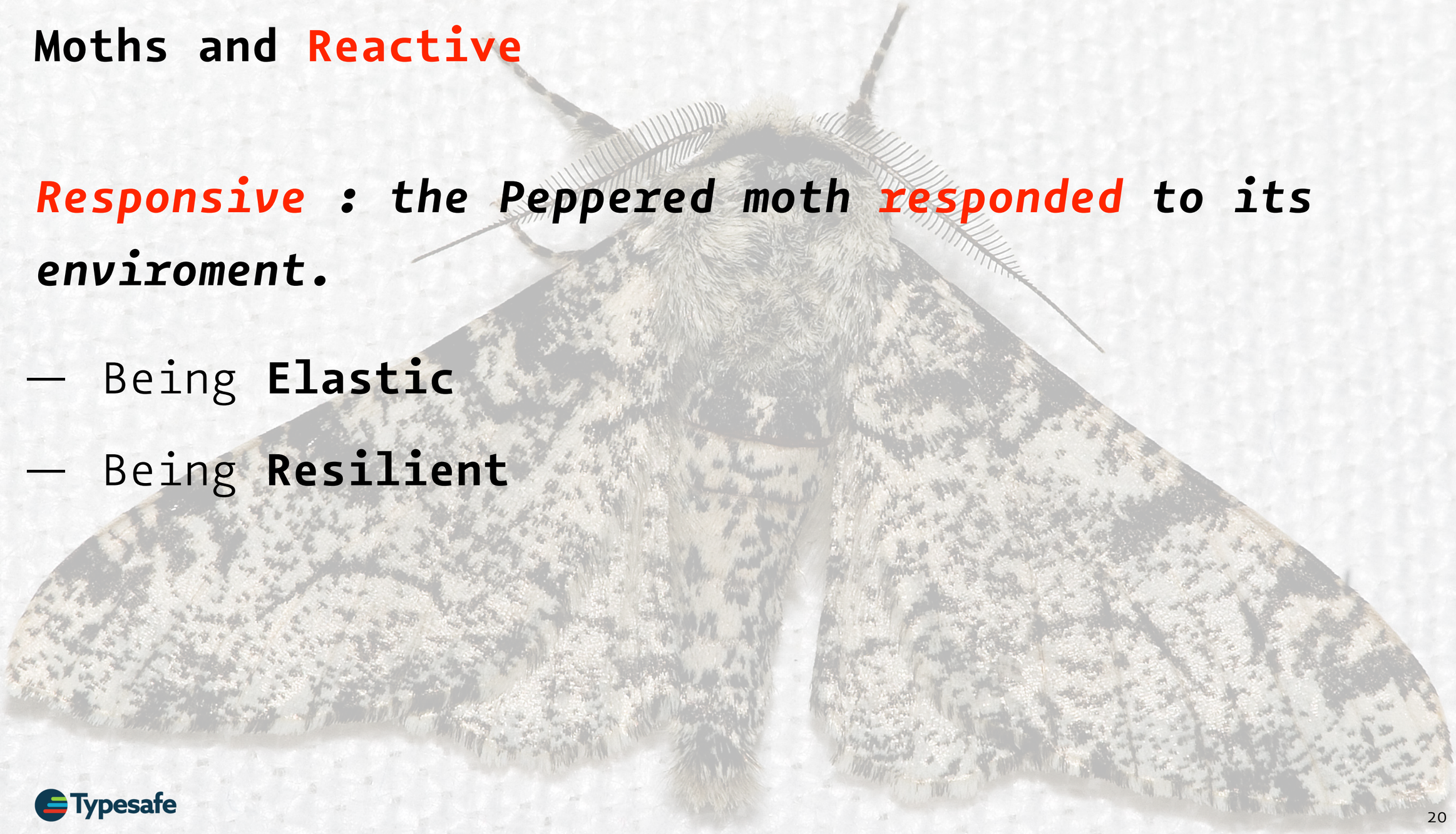
The **mutation** allowed the moth population to **respond** to this **failure**.

Moths and Reactive

Responsive : the Peppered moth responded to its enviroment.

— Being **Elastic**

— Being **Resilient**



Typesafe

Business and Reactive

Business and Reactive

The Internet has had a significant impact on business.

— The US wanted a **robust**, **fault-tolerant** computer network

— A series of memos by **J.C.R. Licklider** of MIT, 1962

— Envisioned a **globally** interconnected network of computers

— Access data and programs from **anywhere** in the world

Business and Reactive

A new computer model, Distributed Systems came into being.

— It represented a **shift** in the computing paradigm

— Before, the model was **large**, **expensive** mainframe systems

— Affectionately referred to as **Big Iron**

— Mainframes used a **centralized** computing model

— Focusing on **efficiency**, **local scalability**, and **reliability**

Business and Reactive

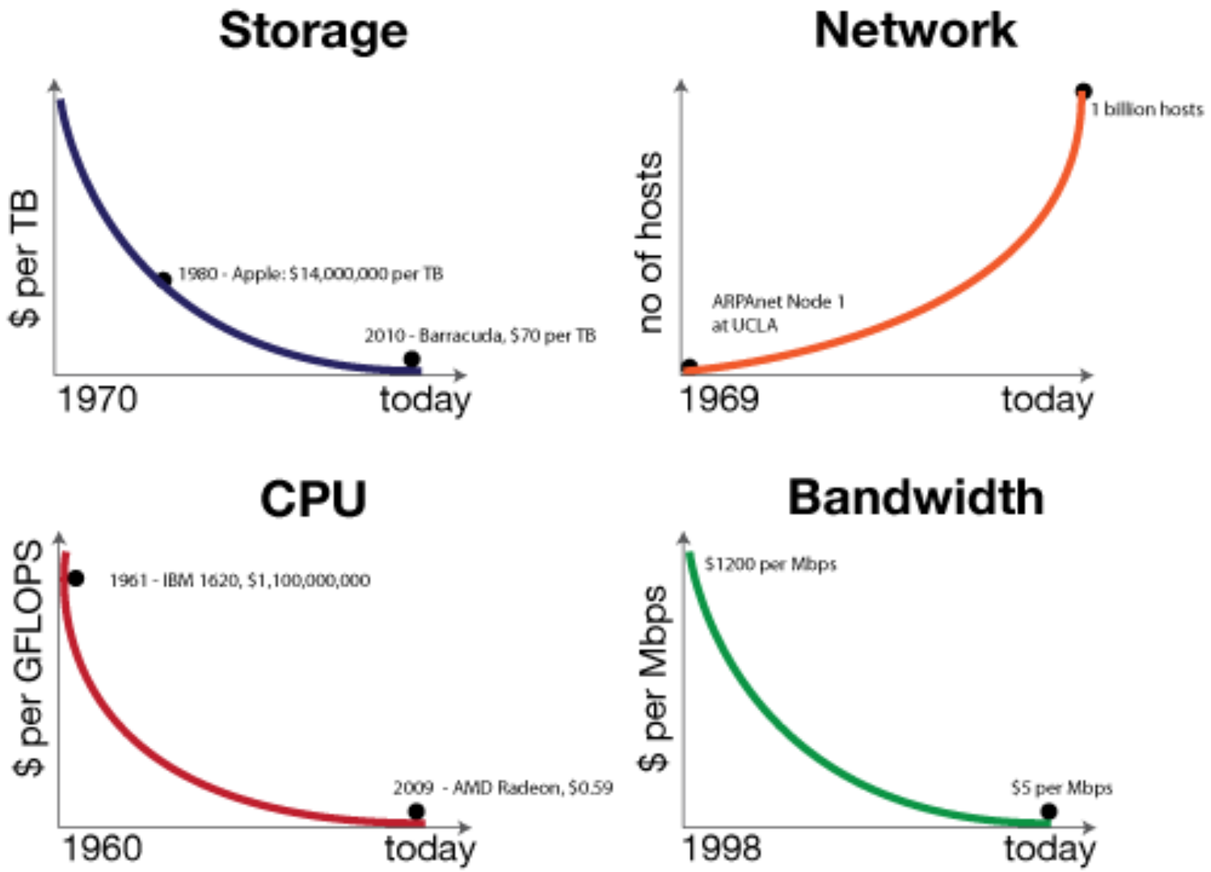
Distributed gave way to what we know as Cloud Computing.

- A more powerful **less expensive** computing solution
- Cloud computing represents another **paradigm** shift
- Changing the way we **reason** about computer applications
- Distributed systems focus on the **technical** details
- Cloud computing focuses on the **economics** side

Business and Reactive

The state of **storage**, **CPU**, and **bandwidth** compared to the number of **network nodes**.

Notice the increase in use and the decrease in cost from the 70's![3]



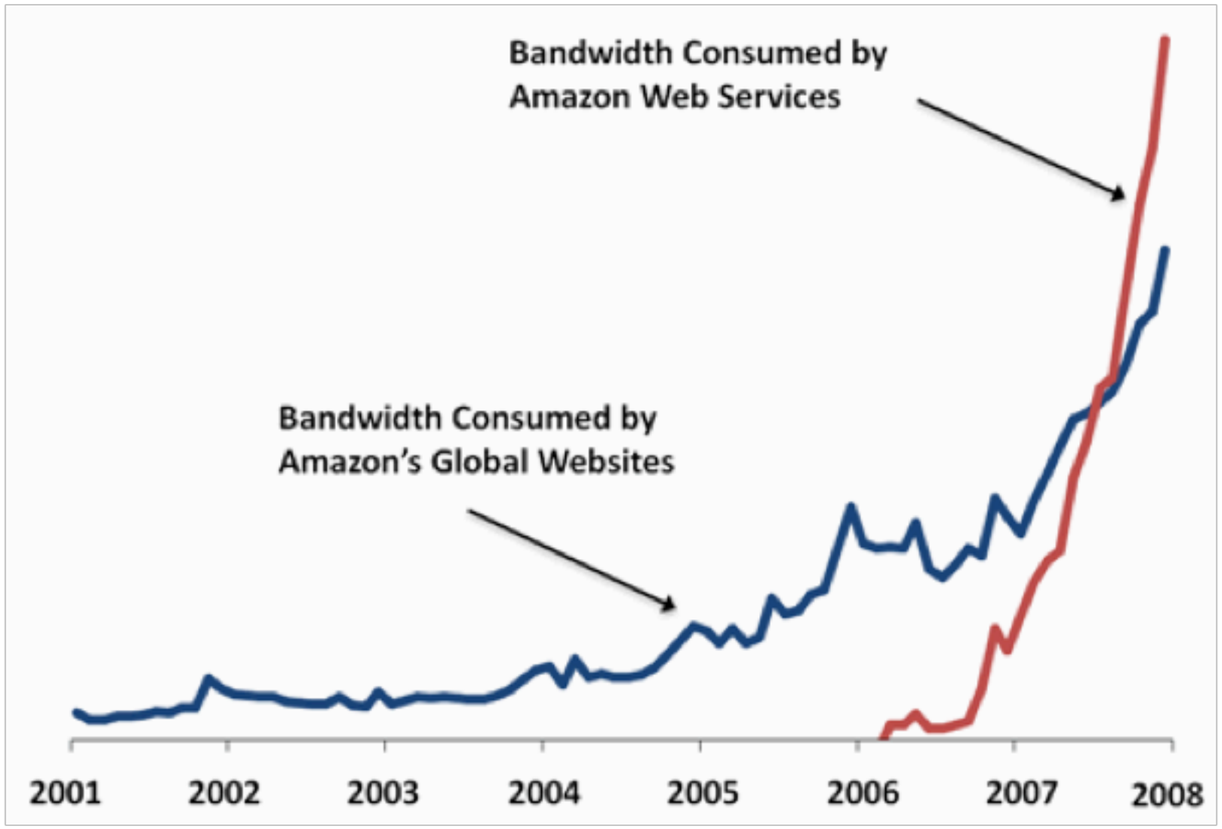
[3] image from Oreilly Radar

Business and Reactive

This new landscape of **distributed cloud computing** represents a **dramatic change** for the modern programmer.

Much like the Industrial Revolution of the nineteenth century did for the Peppered moth.

Business and Reactive



As a result, many companies have begun to rethink their value proposition:

— In January of 2008 Amazon announced that Amazon Web Services now **consume more bandwidth** than their entire global network of retail services, as shown in this figure from Amazon Blogs.[4]

[4] image from Amazon Blogs

Typesafe

Business and Reactive

What is Amazon?

— An online **bookstore**?

— A provider of **Cloud Services**?

Business and Reactive

"As user expectations of sub-second performance, spikes in application load, demands to run on multi-core hardware for parallelism, and data needs expand into the petabytes...,[5]

...modern applications must embrace these changes by incorporating reactive behavior into their DNA."[5]

[5] Reactive Application Development

Programmers and Reactive

Programmers and Reactive

Message-Driven : Key Ideas

- Based on **asynchronous** communication
- Sender/recipient **not affected** by **message propagation**
- You can design your system in **isolation**
- No **worries** how the transmission of messages occurs
- **Message-driven** communication => a **loosely** coupled design
- Provides **scalability**, **resilience** and **responsiveness**

Programmers and Reactive

Message-Driven : Never Block

- blocking incurs a high wake-up cost
- Roughly 650 ns (on Haswell MBP15)
- Can run out of threads if blocking
- Don’t use a single threaded runtime

Programmers and Reactive

Message-Driven : Go Async

- Actors
- Conflict Free Resolution Datasets (CRDTs)
- Reactive Streams
- Futures

Programmers and Reactive

Elasticity : Key Ideas

- System stays **responsive** under varying workload
- Actively scale **up/down**, **in/out** based upon usage
- Saves **money** on unused computing power
- Ensures the **servicing** of **growth** in user base

Programmers and Reactive

Elasticity : Never Block

— Universal Scalability Law

— **Contention**; waiting for queues or shared resources.

— **Coherency**; the delay for data to become consistent

It's been proven that **blocking** of any **kind**, **anywhere** in the system will **measurably** impact scale

Programmers and Reactive

Elasticity : Use Location Transparency

- **Explicit** distributed computing
- **Local** communication is an optimization
- **Locality** of data
- **Async** message passing

Programmers and Reactive

Elasticity : Never Share Mutable State

If **multiple threads** access the same **mutable** state variable without synchronization:

— **Your program is broken**

Programmers and Reactive

Elasticity : Share Nothing

- Embrace **Immutability**
- Immutable collections
- Defensive copies
- Mutability in **local thread-safe** state **only**

Typesafe

Programmers and Reactive

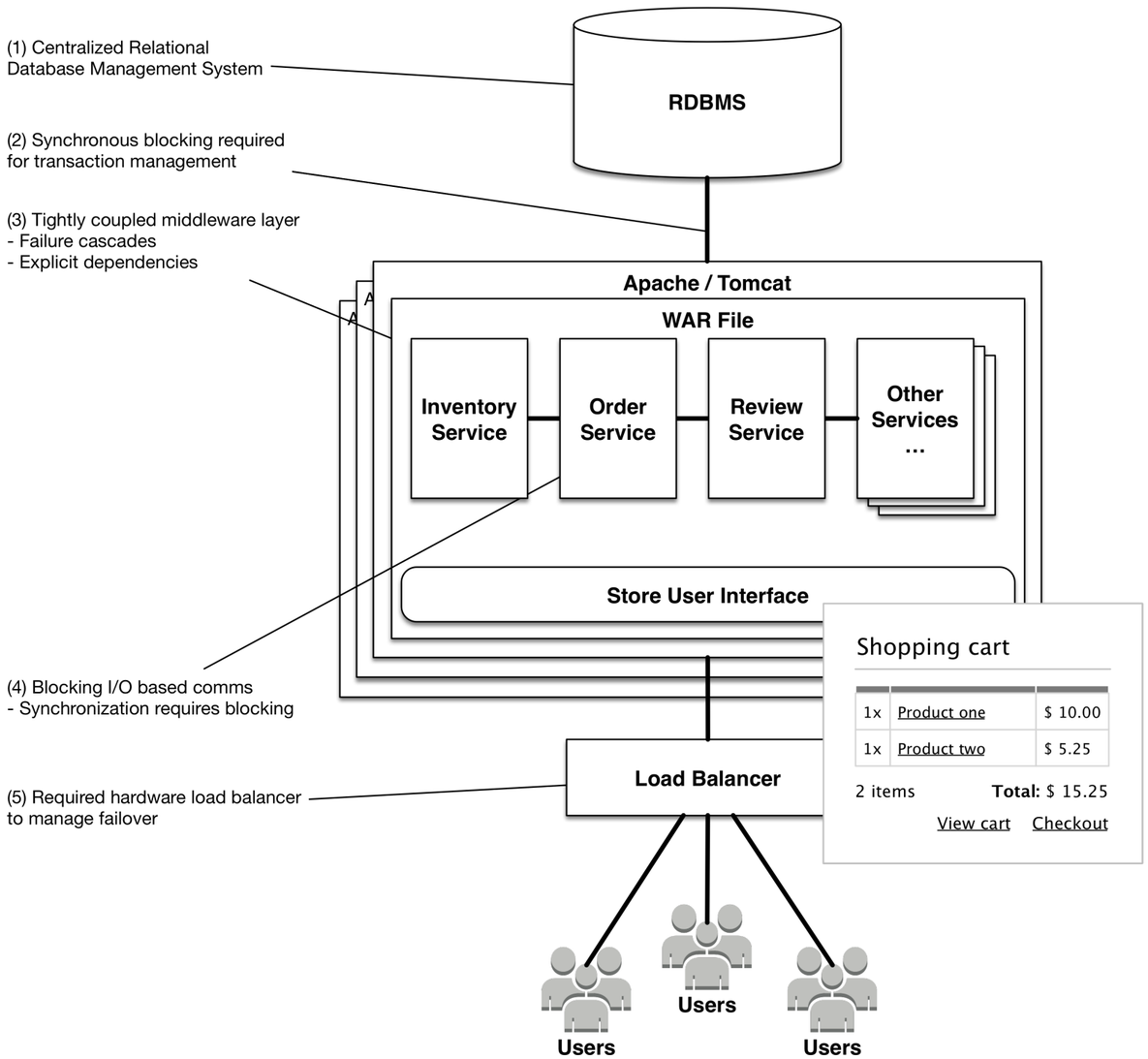
Resilience : Key Ideas

- System stays responsive in the **face of failure**
- Failure is **expected/embraced**; systems exist in **isolation**
- A **single** point of failure **remains** just that
- The system **responds** appropriately
- Strategies for **restarting** or **re-provisioning**
- **Seamless** to the overall systems

Programmers and Reactive

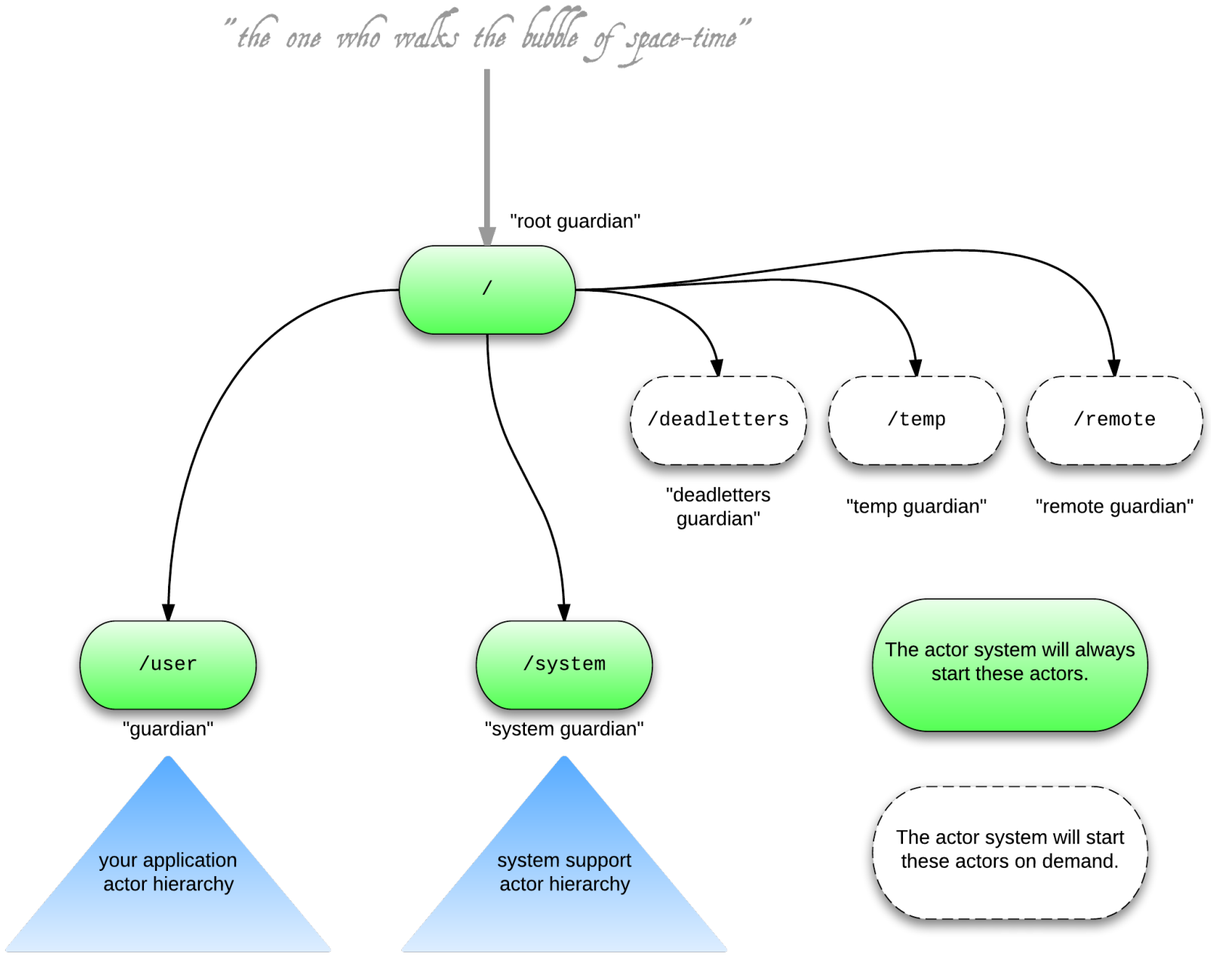
Resilience : No Monoliths

1. Centralized RDBMS
2. Synchronous blocking
3. Tightly coupled middleware
4. Blocking I/O based comms
5. Required load balancer



Typesafe

Programmers and Reactive



Resilience : Use Supervision

1. **Resume** the subordinate
2. **Restart** the subordinattruee
3. **Stop** the subordinate
4. **Escalate** the failure

Typesafe

Programmers and Reactive

Resilience : Use Bulkheads

1. Isolate failure
2. Compartmentalize
3. Manage failure locally
4. Avoid cascading failures

Typesafe

Programmers and Reactive

Responsiveness : Key Ideas

- **Cornerstone** of usability and utility
- The system **responds** promptly if at all possible
- Means that **problems** may be **detected** quickly
- Means that **problems** are **dealt** with effectively
- Built on top of **elasticity** and **resilience**

Monitoring and Reactive

Monitoring and Reactive

There are dozens of monitoring solutions that attempt to keeps tabs on whats going on. However distributed systems are different.

- **No single thread** to trace a path
- **Async** boundaries increase **non-determinism**
- **Business** rules are **distributed** and often **implicit**
- **Volume** of metric data required can be **enormous**
- **Aggregation** of metrics **required**
- Time to **detect** and **repair** often unacceptable

Monitoring and Reactive

Reactive monitoring like security requires an onion skin (layered) approach.

- **Local** metric capture
- **Aggregation** of distributed **local** metrics
- **Control** based on **aggregation** reasoning

Monitoring and Reactive

- **Local** metric capture
- **Configurable** metric capture
- **Class**, **instance** and **group** support
- Source **must** be **optimized** for runtime
- **Self** monitoring constructs
- **feedback** loop control and **tracing**
- **Backoff** algorithms

Monitoring and Reactive

Local capture requires handoff to aggregator(s)

- **Aggregators** should be composable to provide **granularity**
- **Aggregators** should be configurable based on **business** rules
- **Aggregators** must not impact **local** metric capture
- **Aggregators** must be **optimized** for time to **detect**
- Provides **situational awareness**

Monitoring and Reactive

Aggregators are for reasoning, control is required auto-scaling

- **Control** must be configurable
- **Control** should be ingorant of **why**
- **Control** requires **outside** knowledge
- Application **design** and **deployment** affect control
- **Control** must not impede **production**

Monitoring and Reactive

Typesafe Monitoring focuses on local metric capture

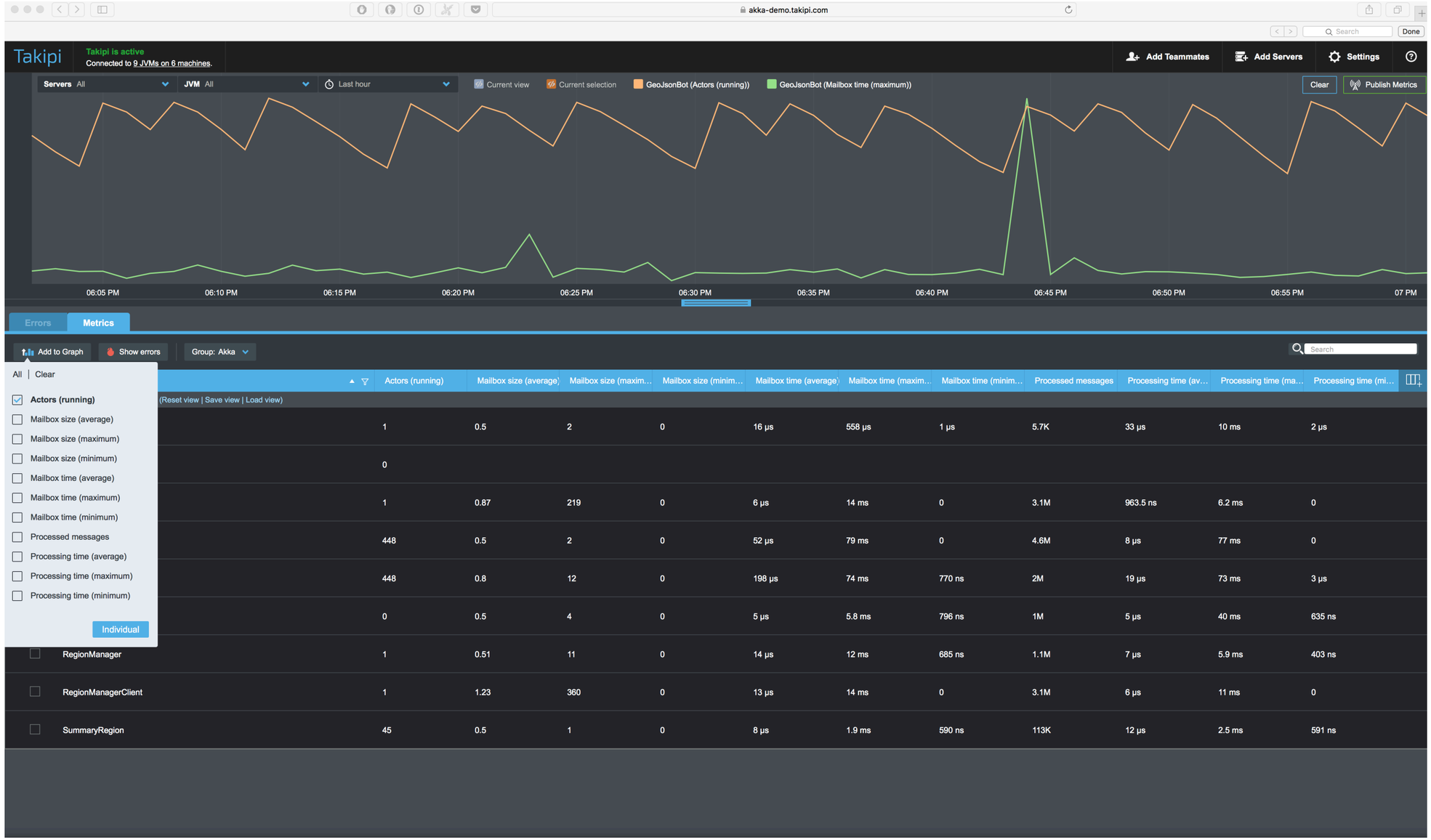
- Built from the **ground** up
- Highly **optimized** for runtime
- Supports **actors** by **class**, **instance** and **group**.
- Supports **wildcards** structures for **class** and **path**
- Local **aggregation** supported
- Actor **tracing** supported.
- **Futures**, **streams** and **circuit breaker** support coming soon

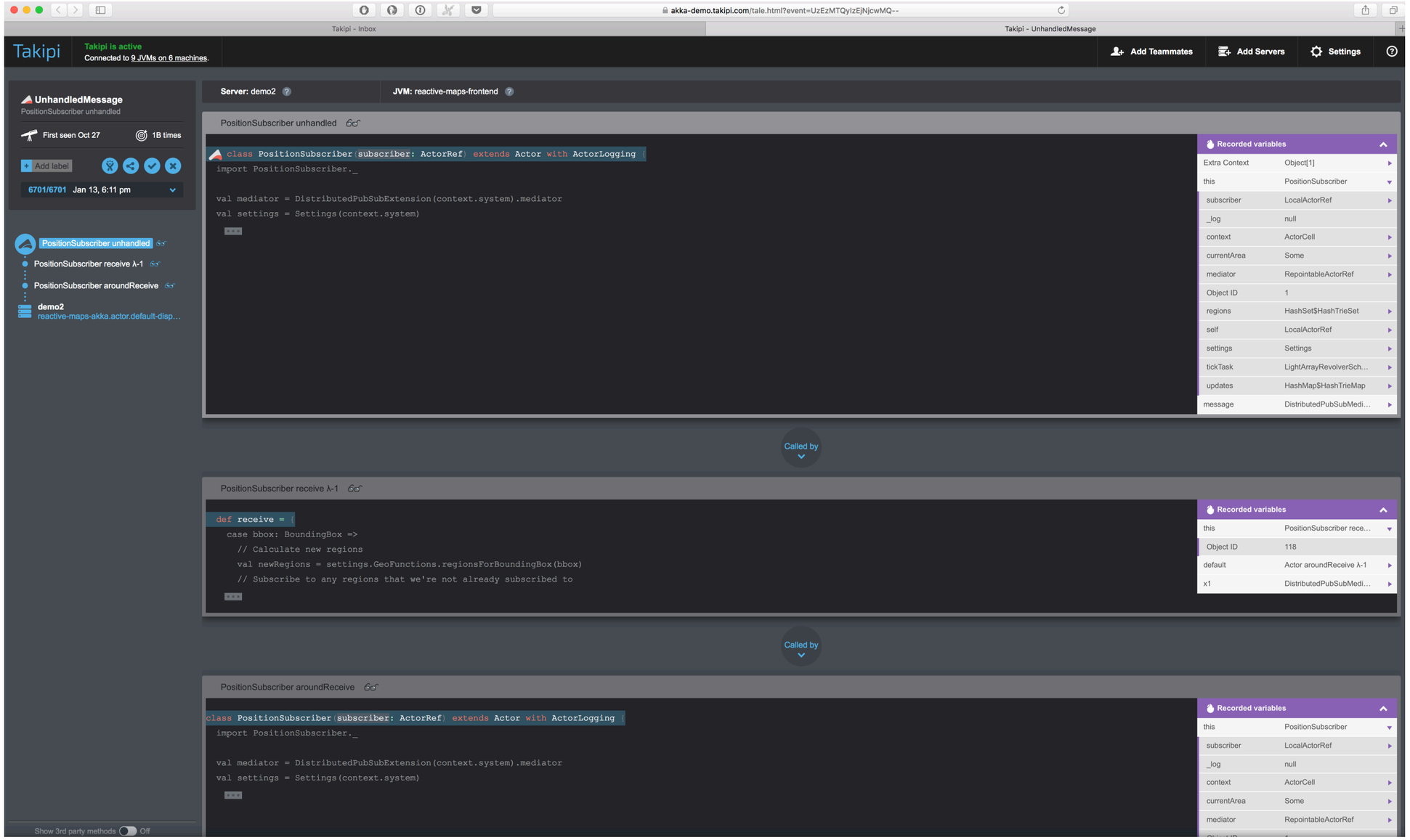
Monitoring and Reactive

Typesafe ConductR focuses on aggregation and control

- **Automated** cluster seed node establishment
- Dynamic service **discovery**
- **Scalable** rolling updates
- No **single point** of failure
- Load balancing at **scale**
- Automated node failure & network partition **resolution**
- **Auto-scaling** support coming soon

Typesafe


akka-demo.takipi.com
Takipi
Takipi is active
Connected to 9 JVMs on 6 machines.
Add Teammates
Add Servers
Settings
Servers All
JVM All
Last hour
Current view
Current selection
GeoJsonBot (Actors (running))
GeoJsonBot (Mailbox time (maximum))
Clear
Publish Metrics
06:05 PM
06:10 PM
06:15 PM
06:20 PM
06:25 PM
06:30 PM
06:35 PM
06:40 PM
06:45 PM
06:50 PM
06:55 PM
07 PM
Errors
Metrics
Add to Graph
Show errors
Group: Akka
Search
All | Clear
Actors (running)
Mailbox size (average)
Mailbox size (maximum)
Mailbox size (minimum)
Mailbox time (average)
Mailbox time (maximum)
Mailbox time (minimum)
Processed messages
Processing time (average)
Processing time (maximum)
Processing time (minimum)
Individual
(Reset view | Save view | Load view)
Actors (running)
Mailbox size (average)
Mailbox size (maxim…
Mailbox size (minim…
Mailbox time (average)
Mailbox time (maxim…
Mailbox time (minim…
Processed messages
Processing time (av…
Processing time (ma…
Processing time (mi…
1 0.5 2 0 16 µs 558 µs 1 µs 5.7K 33 µs 10 ms 2 µs
0
1 0.87 219 0 6 µs 14 ms 0 3.1M 963.5 ns 6.2 ms 0
448 0.5 2 0 52 µs 79 ms 0 4.6M 8 µs 77 ms 0
448 0.8 12 0 198 µs 74 ms 770 ns 2M 19 µs 73 ms 3 µs
0 0.5 4 0 5 µs 5.8 ms 796 ns 1M 5 µs 40 ms 635 ns
RegionManager 1 0.51 11 0 14 µs 12 ms 685 ns 1.1M 7 µs 5.9 ms 403 ns
RegionManagerClient 1 1.23 360 0 13 µs 14 ms 0 3.1M 6 µs 11 ms 0
SummaryRegion 45 0.5 1 0 8 µs 1.9 ms 590 ns 113K 12 µs 2.5 ms 591 ns


akka-demo.takipi.com/tale.html?event=UzEzMTQyIzEjNjcwMQ--
Takipi - Inbox
Takipi - UnhandledMessage
Takipi
Takipi is active
Connected to 9 JVMs on 6 machines.
Add Teammates
Add Servers
Settings
UnhandledMessage
PositionSubscriber unhandled
First seen Oct 27
1B times
Add label
6701/6701 Jan 13, 6:11 pm
PositionSubscriber unhandled
PositionSubscriber receive λ-1
PositionSubscriber aroundReceive
demo2
reactive-maps-akka.actor.default-disp…
Server: demo2
JVM: reactive-maps-frontend
PositionSubscriber unhandled
class PositionSubscriber(subscriber: ActorRef) extends Actor with ActorLogging {
import PositionSubscriber._
val mediator = DistributedPubSubExtension(context.system).mediator
val settings = Settings(context.system)
Recorded variables
Extra Context Object[1]
this PositionSubscriber
subscriber LocalActorRef
_log null
context ActorCell
currentArea Some
mediator RepointableActorRef
Object ID 1
regions HashSet$HashTrieSet
self LocalActorRef
settings Settings
tickTask LightArrayRevolverSch…
updates HashMap$HashTrieMap
message DistributedPubSubMedi…
Called by
PositionSubscriber receive λ-1
def receive = {
case bbox: BoundingBox =>
// Calculate new regions
val newRegions = settings.GeoFunctions.regionsForBoundingBox(bbox)
// Subscribe to any regions that we're not already subscribed to
Recorded variables
this PositionSubscriber rece…
Object ID 118
default Actor aroundReceive λ-1
x1 DistributedPubSubMedi…
Called by
PositionSubscriber aroundReceive
class PositionSubscriber(subscriber: ActorRef) extends Actor with ActorLogging {
import PositionSubscriber._
val mediator = DistributedPubSubExtension(context.system).mediator
val settings = Settings(context.system)
Recorded variables
this PositionSubscriber
subscriber LocalActorRef
_log null
context ActorCell
currentArea Some
mediator RepointableActorRef
Show 3rd party methods Off

Results of Non-Reactive

Results of Non-Reactive?

"In the new world, it is not the big fish which eats the small fish, it's the fast fish which eats the slow fish"[6]

[6] Klaus Schwab

Typesafe

Results of Non-Reactive?

"Everyone has a plan 'till they get punched in the mouth"[7]

[7] Mike Tyson

Typesafe

The End

Typesafe