

Reactive Systems: The Why and the How

CJUG May 19, 2015

dean.wampler@typesafe.com

[@deanwampler](https://twitter.com/deanwampler)

©Typesafe 2014-2015, All Rights Reserved

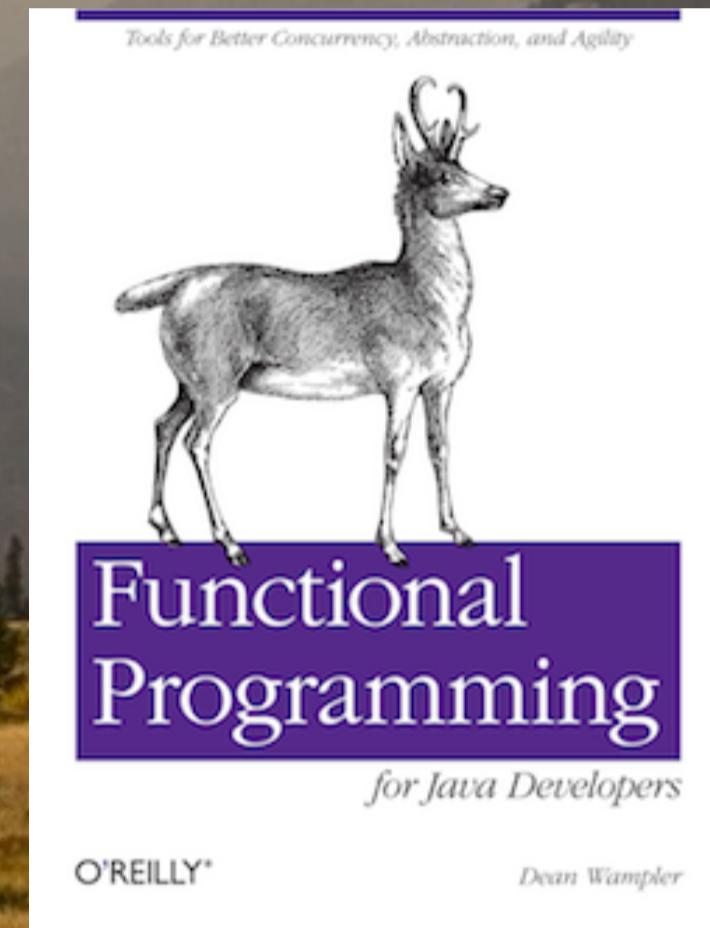
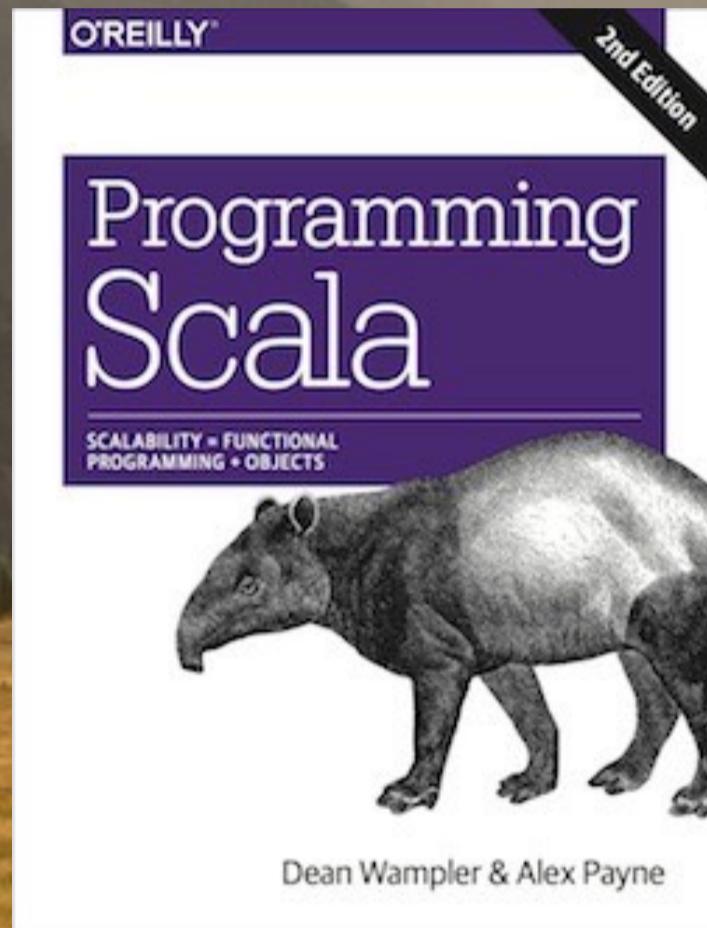


Tuesday, May 19, 15

Photos from Colorado, Sept. 2014.

All photos are copyright (C) 2000–2015, Dean Wampler, except where otherwise noted. All Rights Reserved.

dean.wampler@typesafe.com
polyglotprogramming.com/talks
@deanwampler



Chicago User Groups



- Chicago Area Scala Enthusiasts
- Spark Spark Users
- Chicago Area Hadoop User Group

3

Tuesday, May 19, 15

I founded CASE, I lead the Spark Users group, and I'm a co-organizer of CHUG.

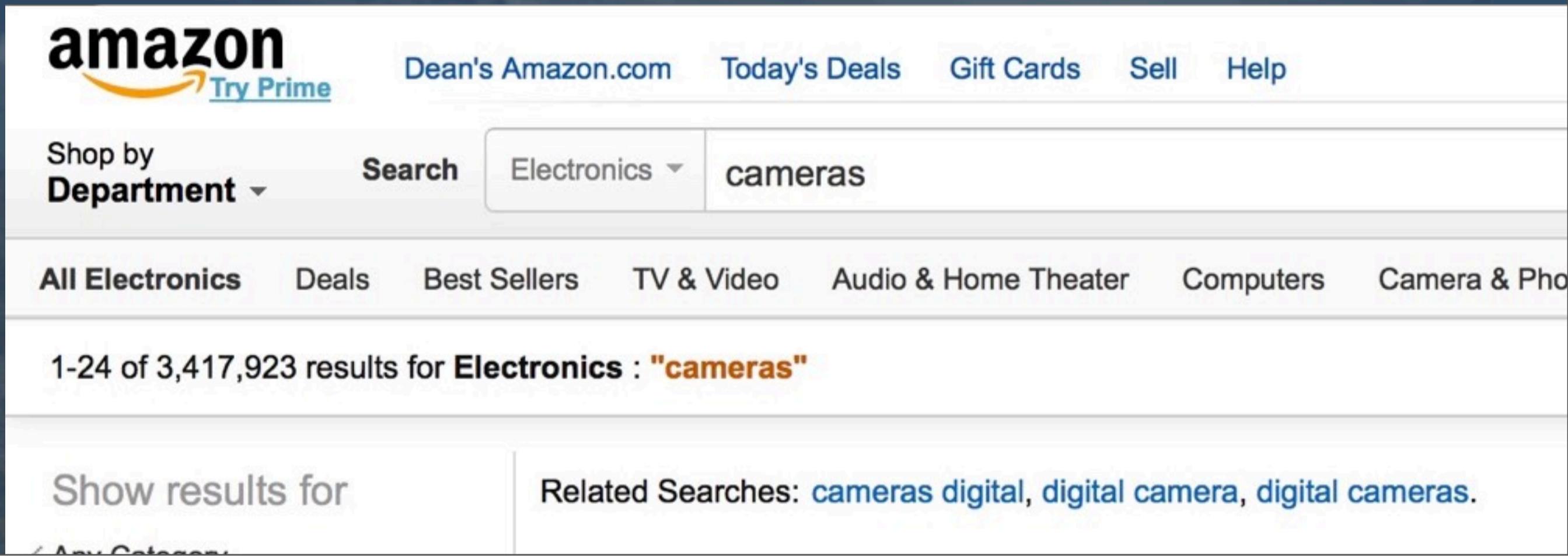
Motivation: eCommerce

4

Tuesday, May 19, 15

Let's motivate the notion of "reactive" systems by exploring some common scenarios we see today.

Cyber Monday?

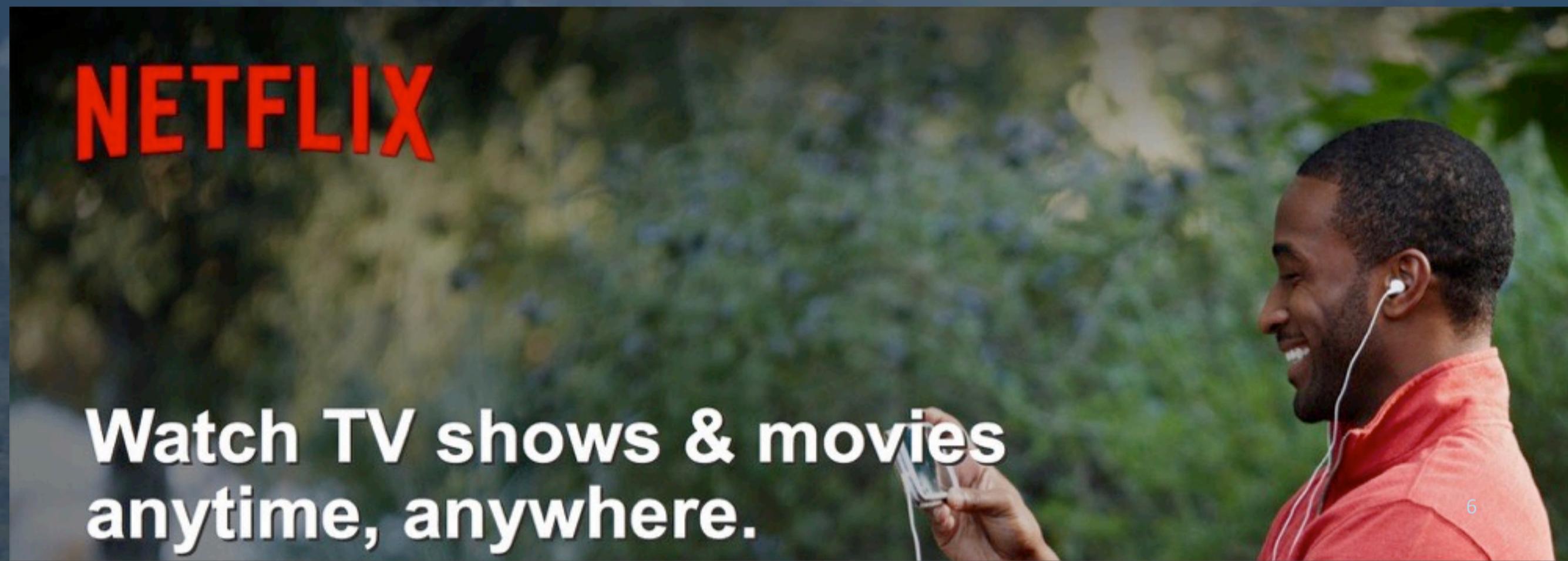


The screenshot shows the Amazon homepage with a search bar containing "Electronics" and "cameras". Below the search bar, there are links for All Electronics, Deals, Best Sellers, TV & Video, Audio & Home Theater, Computers, and Camera & Photo. A message indicates 1-24 of 3,417,923 results for Electronics : "cameras". There are also "Show results for" and "Related Searches" sections.

Tuesday, May 19, 15

Your online store needs to scale up and down with demand. It needs to degrade gracefully if some service components are lost or disconnected by a network partition. For example, if the canonical catalog “disappears” behind a network partition, it’s probably better to continue selling with a stale local copy.
photo: Amazon home page, <http://amazon.com>.

On demand?



Tuesday, May 19, 15

Netflix has extreme scale challenges, but they have become an innovator in building highly resilient, scalable services. What happens when a new season of "House of Cards" is released? Spikes in traffic?
photo: Netflix home page, <http://netflix.com>.



Motivation: Internet of Things

7

Tuesday, May 19, 15

Internet of Things has several categories of applications, each of which has needs that motivate reactive programming.

Medical Devices, IT Systems

Phone home:

- Upload data
 - Usage patterns
 - Predictive diagnostics
- Fetch patches



8

Tuesday, May 19, 15

Medical devices upload test results (e.g., ultrasound images and video) to servers. Med. devices and IT systems send requests for automated updates, send the equivalent of “click-stream” data used to assess usability, etc., and increasingly send metrics used to predict potential HW failures or other service needs.

ultrasound photo: <http://www.usa.philips.com/healthcare-product/HC795054/hd11-xe-ultrasound-system>

switch photo: <http://networklessons.com/switching/introduction-to-vtp-vlan-trunking-protocol/>

Medical Devices, IT Systems

Characteristics:

- Stable to intermittent network connectivity
- One way and two way
- Mixed bandwidth



9

Tuesday, May 19, 15

A mobile scanner might move in and out of WiFi zones, so caching data is necessary. IT appliances are (hopefully) always online. Some data is one way, like diagnostic info for predictive analytics, while data uploads and patch requests need acknowledgements. Bandwidth can vary.

Aircraft Engines

Phone home:

- Upload telemetry
 - Predictive diagnostics
 - Redundant tracking data!



10

Tuesday, May 19, 15

Each engine collects 0.5–1TB of data per flight. Most is currently tossed! Our only clue about the final resting place of Malaysian Airlines Flight 370 was engine telemetry picked up by satellites and used to analyze possible routes.

Trucks, Farm Equipment

GPS Tracking:

- Optimize routing, fuel use, etc.
- Spy on drivers?
- Per plant tracking!



Tuesday, May 19, 15

Track data to optimize routing, minimize fuel use with shortest path and/or delivering heaviest items first. Ensure drivers are obeying the rules of the road and company policies. Some farm equipment planting, watering, and fertilizing gear now tracks data per plant!

UPS truck photo: http://en.wikipedia.org/wiki/United_Parcel_Service

Planter/seeder photo: http://www.deere.com/en_US/products/equipment/planting_and_seeding_equipment/planting_and_seeding_equipment.page

Trucks, Farm Equipment

Connectivity:

- Always along roads
- Intermittent on farms (WiFi in barns?)



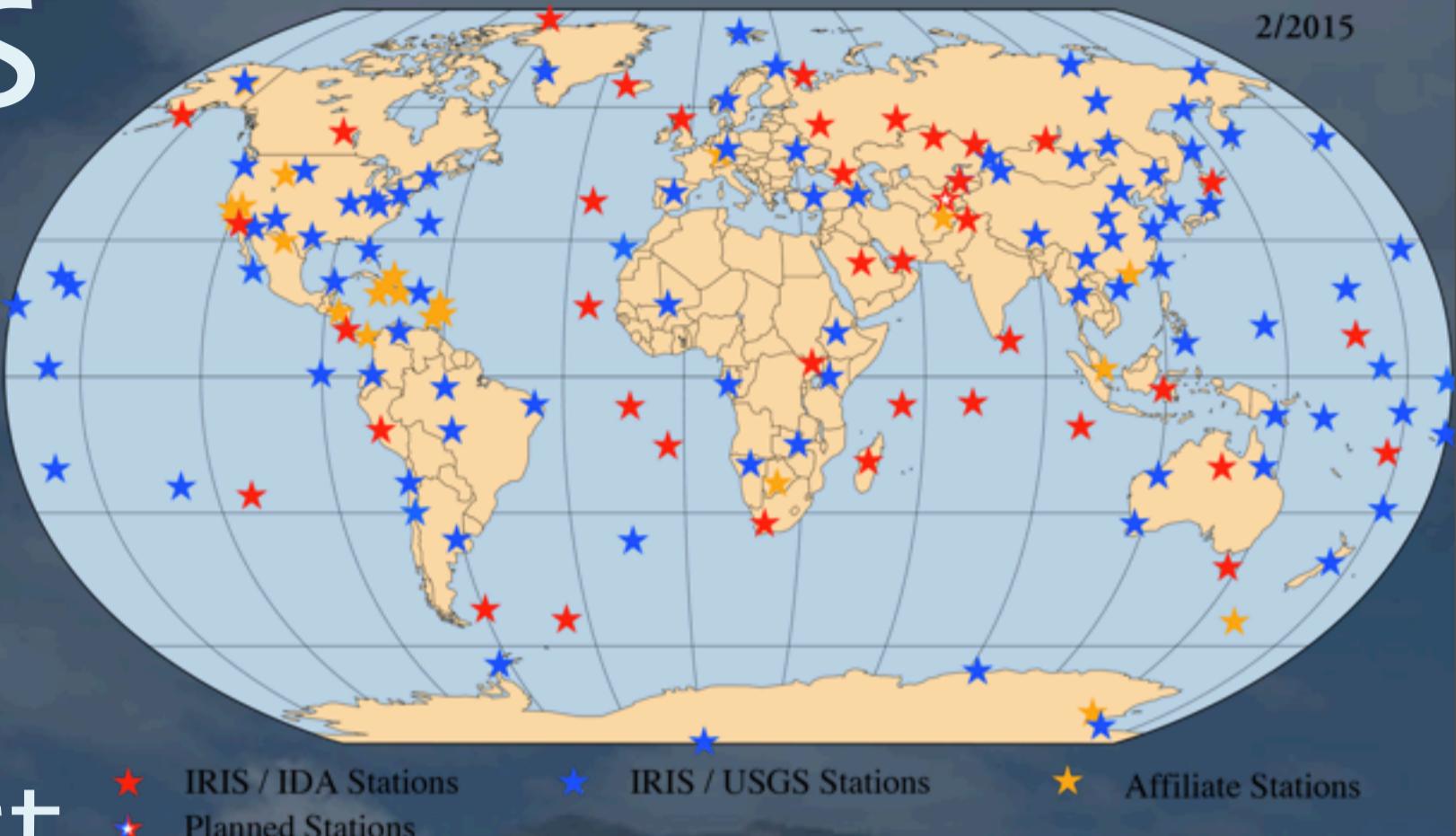
Tuesday, May 19, 15

Some rural areas don't have sufficient wireless data coverage for farm equipment to remain online full time.

Remote Sensors

Human to Real-time
Responsiveness:

- Earthquake, nuclear test sensor networks.
- Climate change monitoring



13

Tuesday, May 19, 15

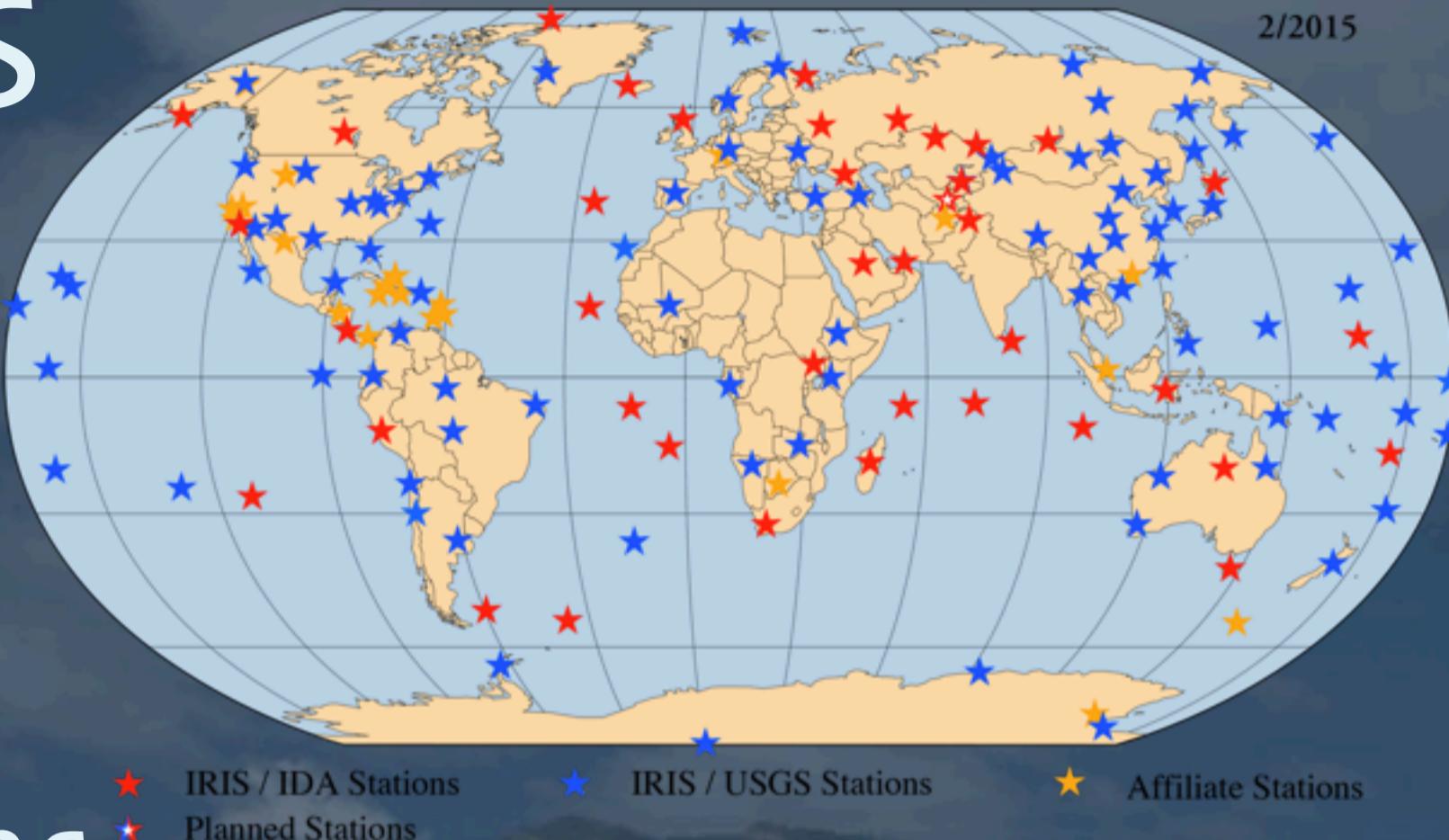
For earthquake sensor networks, you want to get the information to emergency services and community alarm systems in milliseconds.

photo: <http://www.iris.edu/hq/programs/gsn>

Remote Sensors

Characteristics:

- Redundant sensors
- Low-latency connections
- Low-bandwidth requirements



14

Tuesday, May 19, 15

This requires redundant sensors, always on connectivity, and low-latency connections. The amount of data isn't large. Some networks, like monitoring rainfall or glaciers for climate change studies, might be offline except for once-per-year downloads done onsite!

Robotics

Connectivity:

- Two-way, but time of flight matters!
- Autonomous?



Tuesday, May 19, 15

Some rural areas don't have sufficient wireless data coverage for farm robots to remain online full time. The one-way time of flight between Earth and Mars is ~8 minutes.

Quadcopter photo: <http://www.dji.com/product/phantom>

Mars rover photo: http://en.wikipedia.org/wiki/Mars_Exploration_Rover

Health Monitoring

Characteristics:

- Occasional to always-on connectivity
- Detect health emergencies: call for help?



16

Tuesday, May 19, 15

Health monitoring tools are most popular for gathering activity data and some vital signs for analysis later. Some monitors are designed to detect medical emergencies and call for help when needed.

photo: <http://www.fitbit.com/force>

Home Automation

Characteristics:

- ToD packet storms
- Fire & break-in detection:
automatic notification of
authorities



17

Tuesday, May 19, 15

These systems are subject to time of day packet storms, e.g., while everyone in a given time zone is waking up or going to bed.

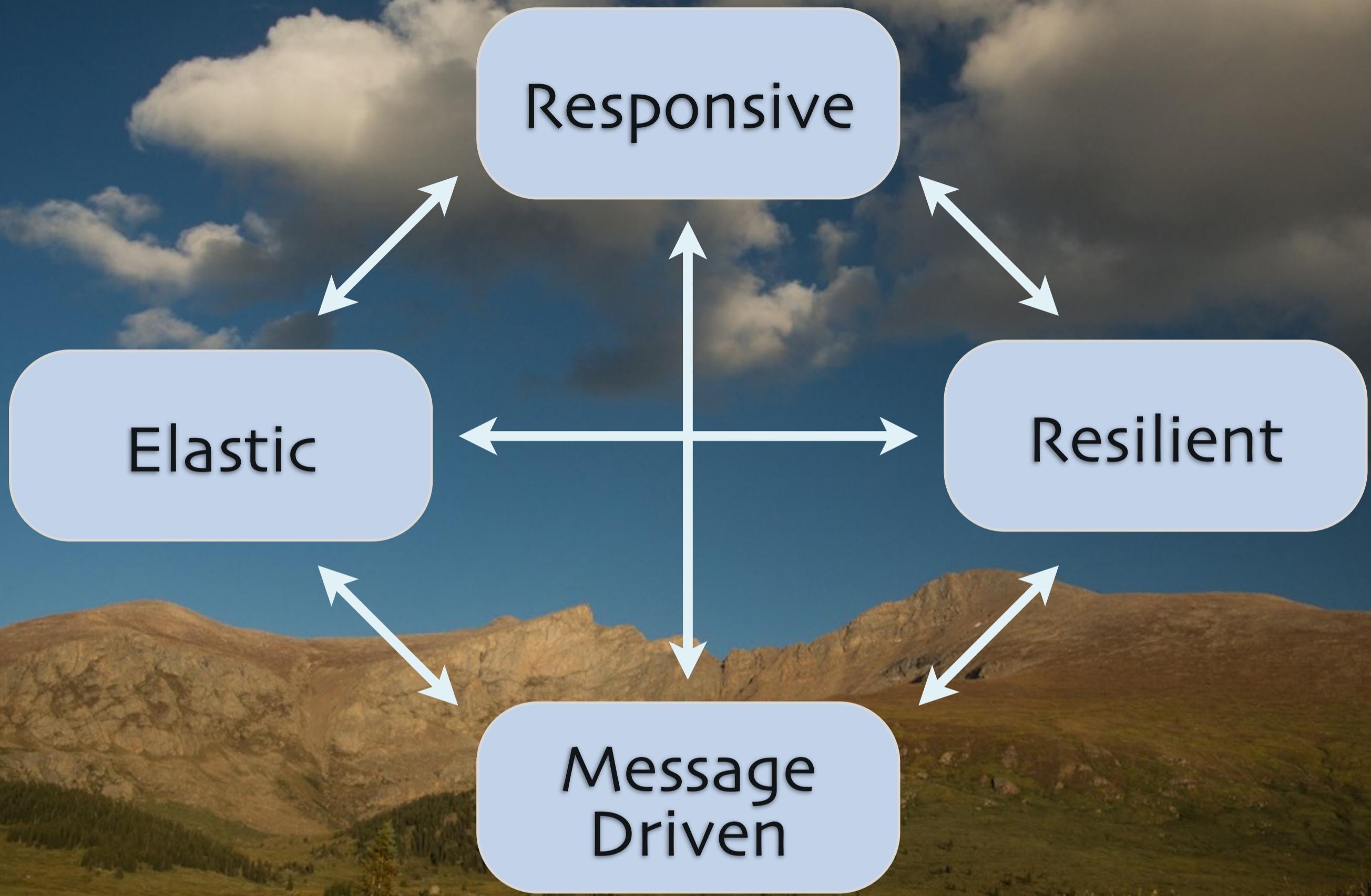
photo: <https://store.nest.com/product/thermostat/>

Reactive Systems

18

Tuesday, May 19, 15

The idea of Reactive Systems emerged to catalog several common characteristics of the systems we have to build to support these scenarios, without over-specifying how these characteristics are satisfied.



29

Tuesday, May 19, 15

The four characteristics or traits of Reactive Systems... as articulated by the Reactive Manifesto, which attempts to codify lessons learned across many projects, industries, and years building highly available, scalable, and reliable systems.

The screenshot shows a web browser window with the title bar "R The Reactive Manifesto" and the URL "www.reactivemanifesto.org". The page content features a large blue diagonal banner on the right side with the text "We Are Reactive". The main heading "The Reactive Manifesto" is centered above a publication date "Published on September 16 2014. (v2.0)". Below the heading, two paragraphs discuss the evolution of software requirements and the challenges faced by organizations.

Organisations working in disparate domains are independently discovering patterns for building software that look the same. These systems are more robust, more resilient, more flexible and better positioned to meet modern demands.

These changes are happening because application requirements have changed dramatically in recent years. Only a few years ago a large application had tens of servers, seconds of response time, hours of offline maintenance and gigabytes of data. Today applications are deployed on everything from mobile devices to cloud-based clusters running thousands of multi-core processors. Users expect millisecond response times and 100% uptime. Data is measured in Petabytes. Today's demands are simply not met by yesterday's software architectures.

Tuesday, May 19, 15

The four characteristics of Reactive Systems... as articulated by the Reactive Manifesto, which attempts to codify lessons learned across many projects, industries, and years building highly available, scalable, and reliable systems.



Myths

21

Tuesday, May 19, 15

Before discussing them in detail, let's slay some myths.



Myths

“This is new.”

22

Tuesday, May 19, 15

The RM attempts to codify lessons learned over many years in many scenarios. It's not new. It doesn't claim to be new.



Myths

“This is
Typesafe marketing.”

23

Tuesday, May 19, 15

While Typesafe's Jonas Bonér was one of the originators of the RM, other originators and contributions include experts in many companies and specialties.

Industry Adoption

https://spring.io/blog/2013/05/13/reactor-a-foundation-for-asynchronous-applications-on-the-jvm

spring

DOCS GUIDES PRO

ENGINEERING

Reactor - a foundation for asynchronous applications on the JVM

ENGINEERING JON BRISBIN MAY 13, 2013 6 COMMENTS

24

Tuesday, May 19, 15

A few industry heavyweights that are embracing Reactive. First, Spring's Reactor

Industry Adoption

The Java Source

Insider News from the Java Team at Oracle!

Recent Posts

Update Your Skills for the 20 Years of Java

Oracle Java 8 ME Embedded + Raspberry Pi + Sensors = IoT World

[« Java 9 Schedule | Main](#)

Reactive Java EE

By Yolande Poirier-Oracle on May 13, 2015

Tuesday, May 19, 15

Even Oracle is embracing Reactive principles for Java EE.

Industry Adoption

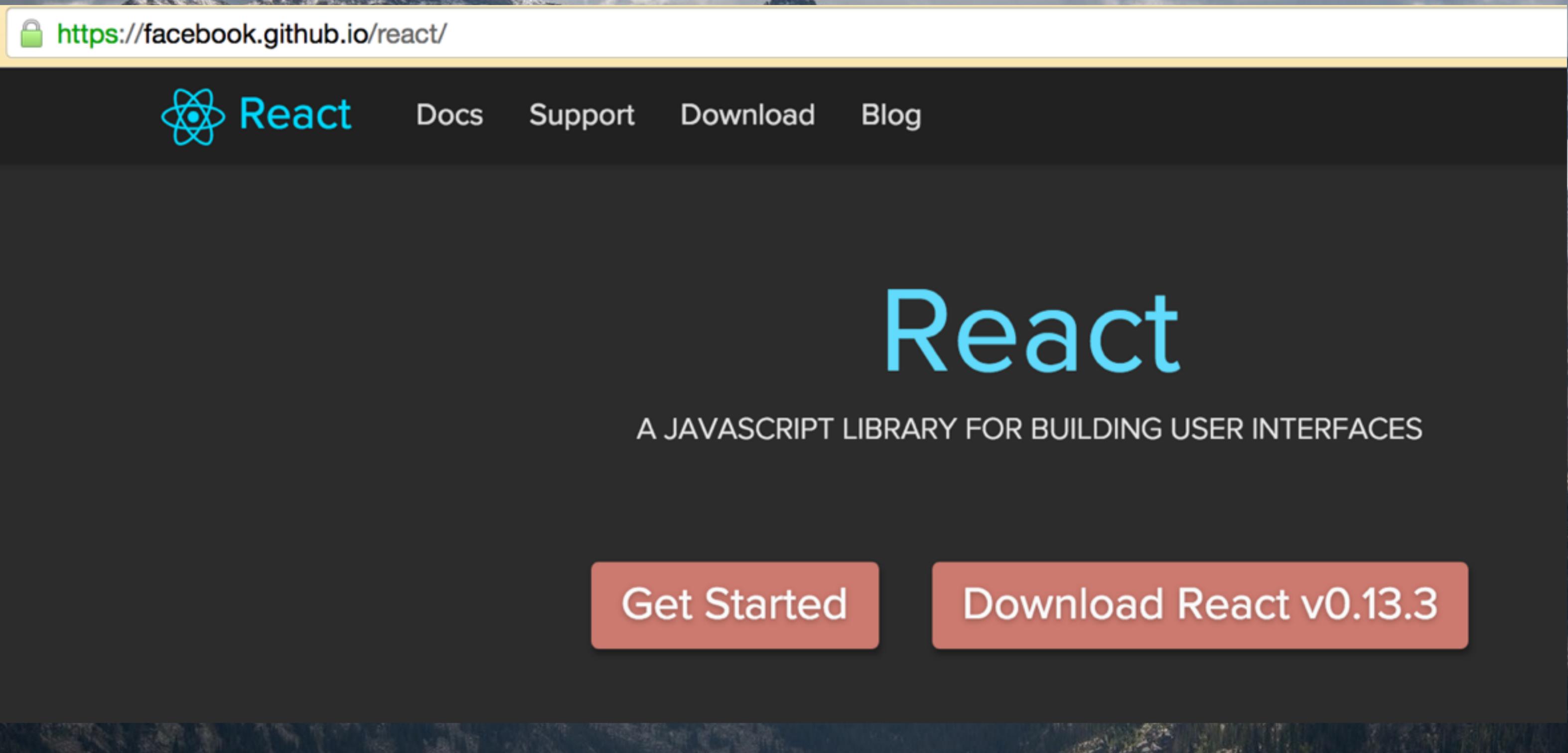


26

Tuesday, May 19, 15

Netflix created RxJava. (We'll define "Reactive Extensions")

Industry Adoption

A screenshot of the React.js website homepage. The page has a dark background featuring a mountain range at the top. At the top left is a browser-style header with a lock icon and the URL "https://facebook.github.io/react/". Below this is a navigation bar with the React logo (an atom symbol) and the word "React" in blue, followed by links for "Docs", "Support", "Download", and "Blog". The main title "React" is prominently displayed in large blue letters, with the subtitle "A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES" in smaller white letters below it. Two orange buttons at the bottom are labeled "Get Started" and "Download React v0.13.3".

<https://facebook.github.io/react/>

 **React** Docs Support Download Blog

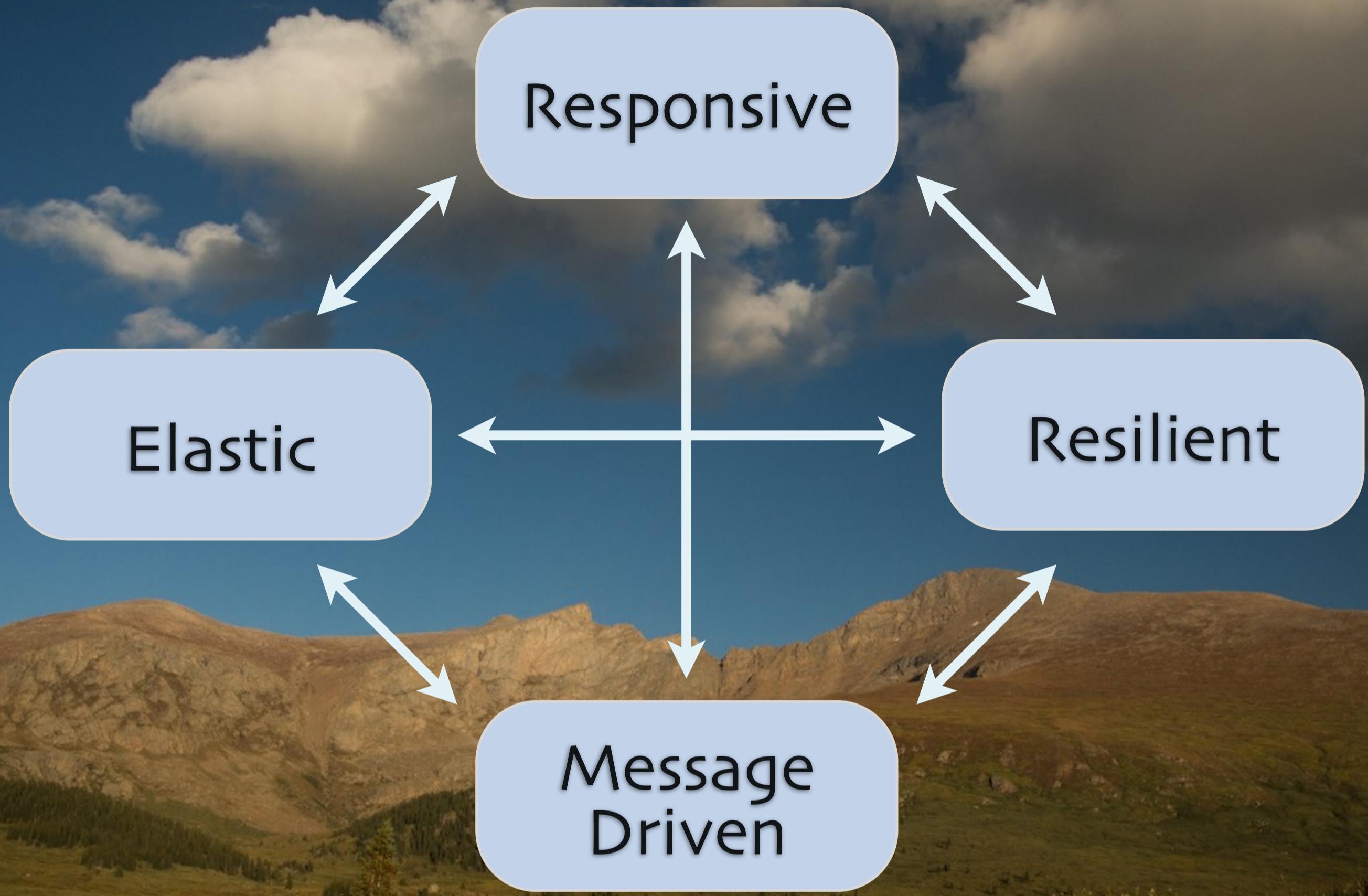
React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

[Get Started](#) [Download React v0.13.3](#)

Tuesday, May 19, 15

Facebook's JS framework.

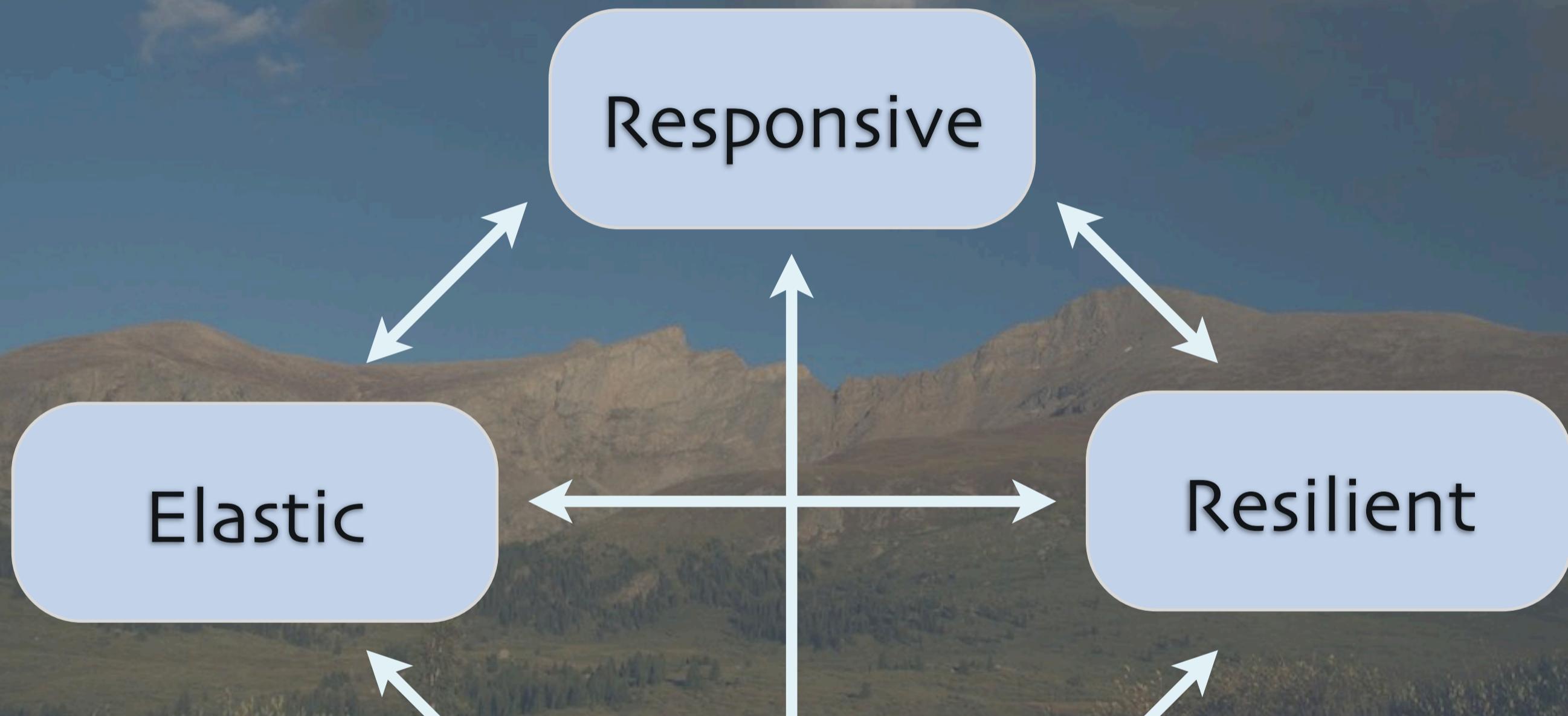


28

Tuesday, May 19, 15

Now let's explore the four traits of Reactive systems.

Requests or commands require timely responses.



29

Tuesday, May 19, 15

What does it mean if a service you rely on fails to respond to requests for service?



Responsive

30

Tuesday, May 19, 15



Responsive

Cornerstone of
usability and utility.



Responsive

Requires rapid
detection of errors
and quick responses.



Responsive

Requires predictable
response times
and quality of service.



Responsive

Requires pre-planned
graceful degradation
of service.

34

Tuesday, May 19, 15

You should plan in advance what level of service you'll provide if (or better, when) certain failure scenarios arise.



Responsive

Awareness of time
is first class.

35

Tuesday, May 19, 15

You have to be a clock watcher of sorts.

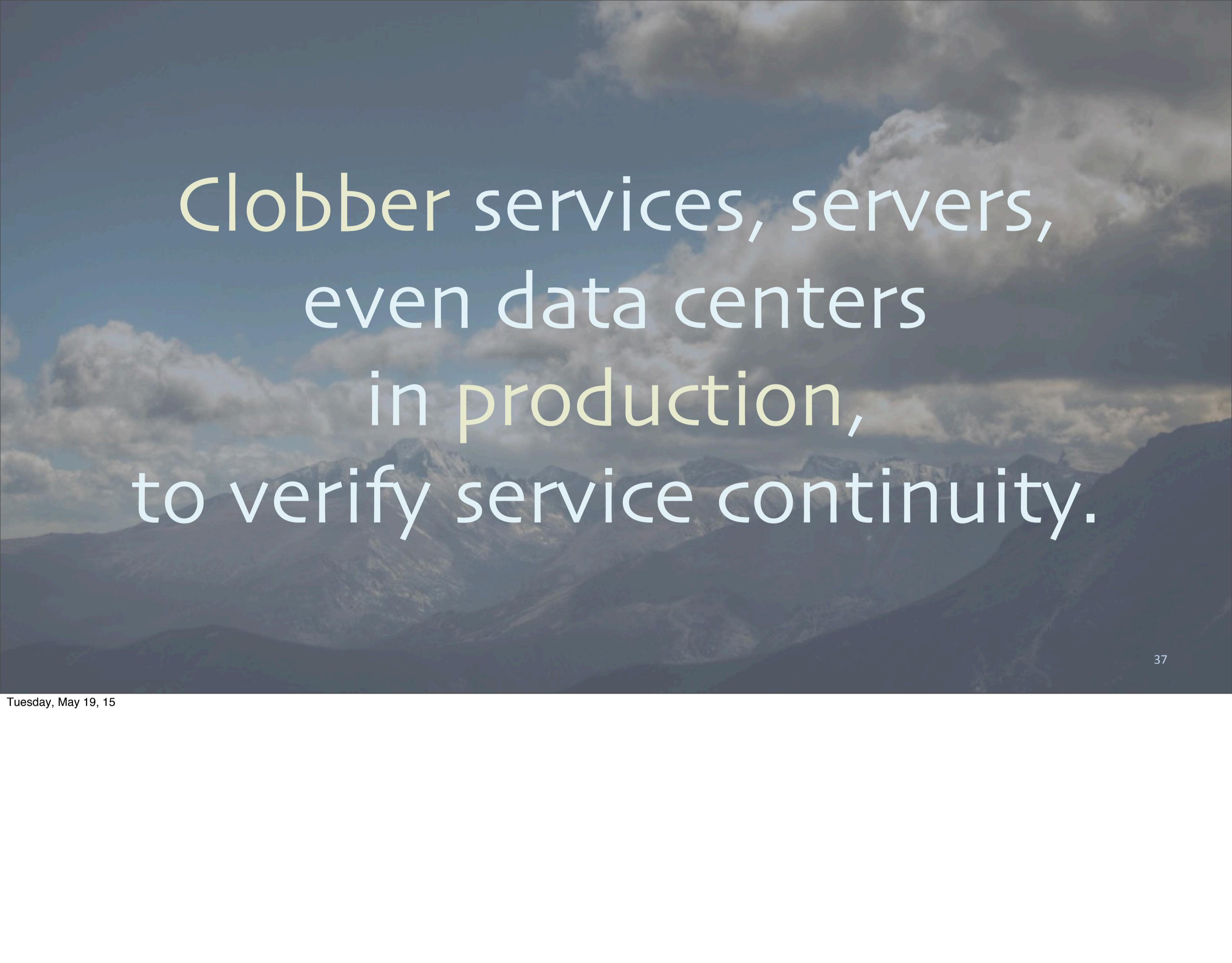
Example: Netflix Simian Army



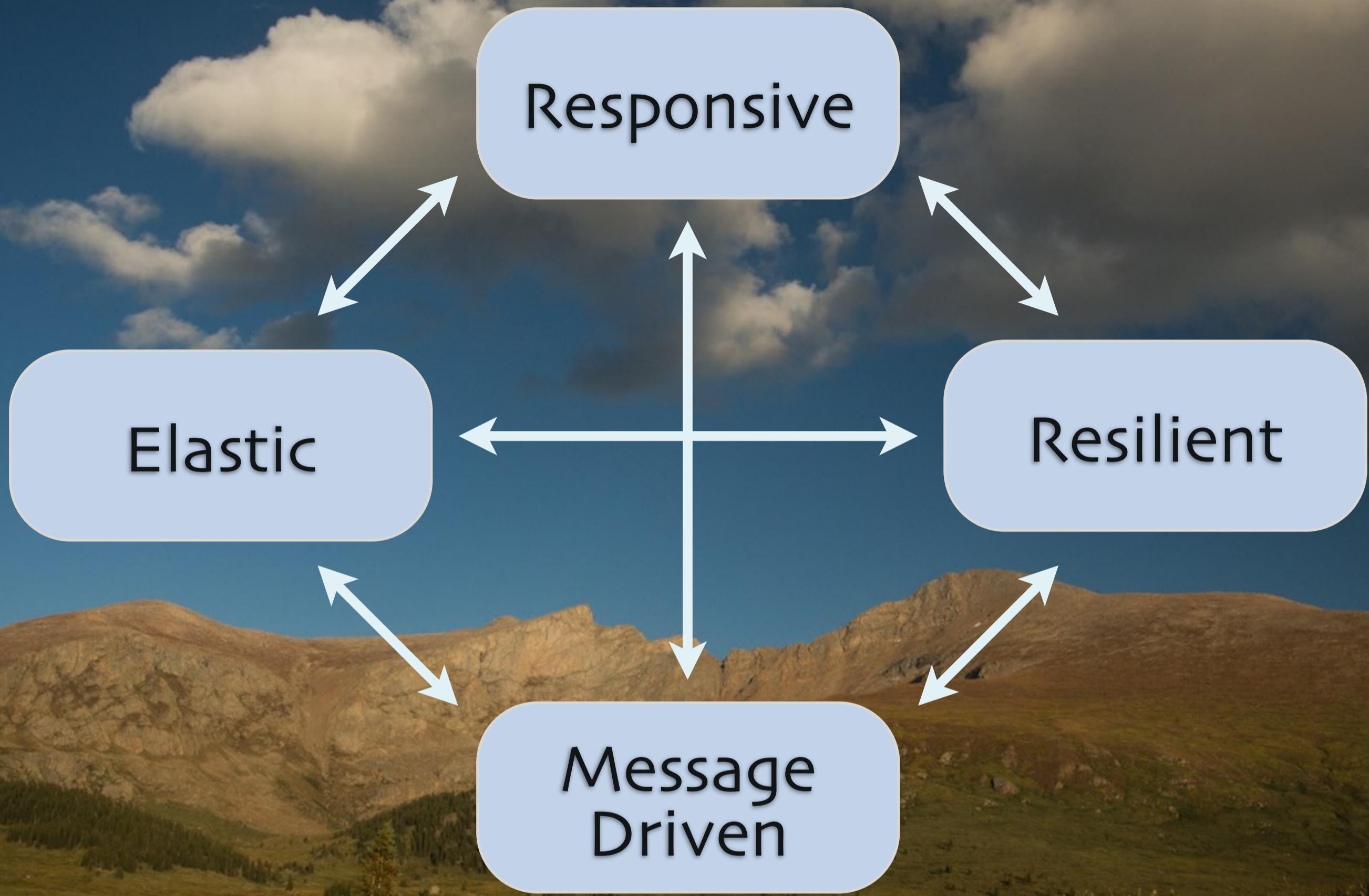
36

Tuesday, May 19, 15

For each trait, I'll cite some good examples of adding the trait. Image from <http://devops.com/features/netflix-the-simian-army-and-the-culture-of-freedom-and-responsibility/>



Clobber services, servers,
even data centers
in production,
to verify service continuity.





39

Tuesday, May 19, 15

Truly resilient systems must treat failures as routine, in some sense of the word, because they are inevitable when the systems are big enough and run long enough.



Resilient



Resilient

Failure is
not disruptive.



Resilient

Failure is
expected.



Resilient

So, failure must be
first class.

43

Tuesday, May 19, 15

A normal part of your domain model, implementation, etc.



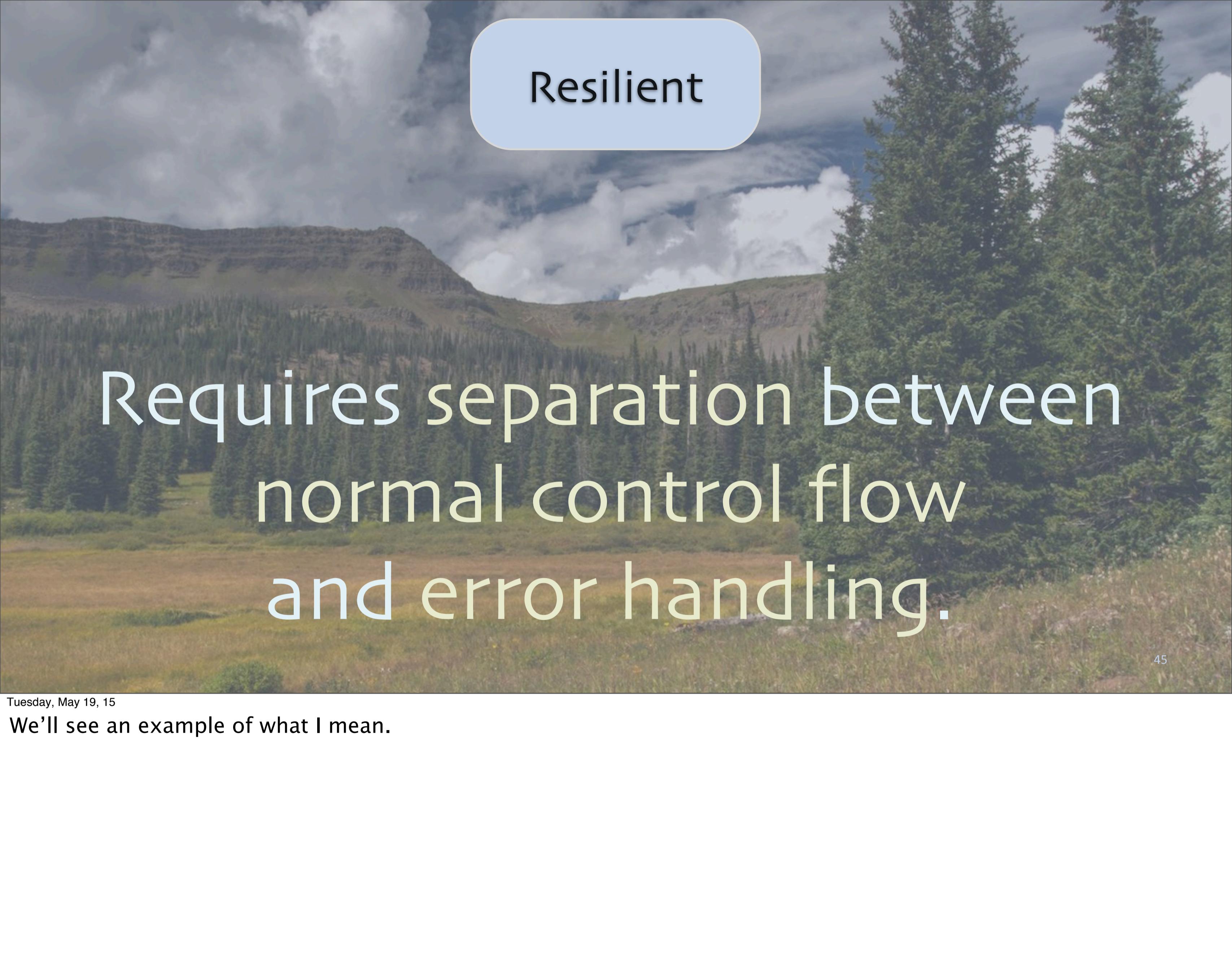
Resilient

Requires replication,
containment, isolation,
and delegation.

44

Tuesday, May 19, 15

Replication – other copies (data and services) replaced lost copies.
Containment and isolation – firewalls stop disaster from spreading.
Delegation – indirection to allow new copies to step into “holes”.

A scenic landscape featuring a range of mountains in the background, covered with green forests. In the foreground, there's a field of dry, yellowish-brown grass. The sky is filled with large, white, billowing clouds.

Resilient

Requires separation between
normal control flow
and error handling.

45

Tuesday, May 19, 15

We'll see an example of what I mean.

Example: Failure-handling in Actor Systems

46

Tuesday, May 19, 15

The Netflix Simian Army could also be cited here.
We'll come back and fill in details of Actor systems shortly. For now, let's focus on error handling.



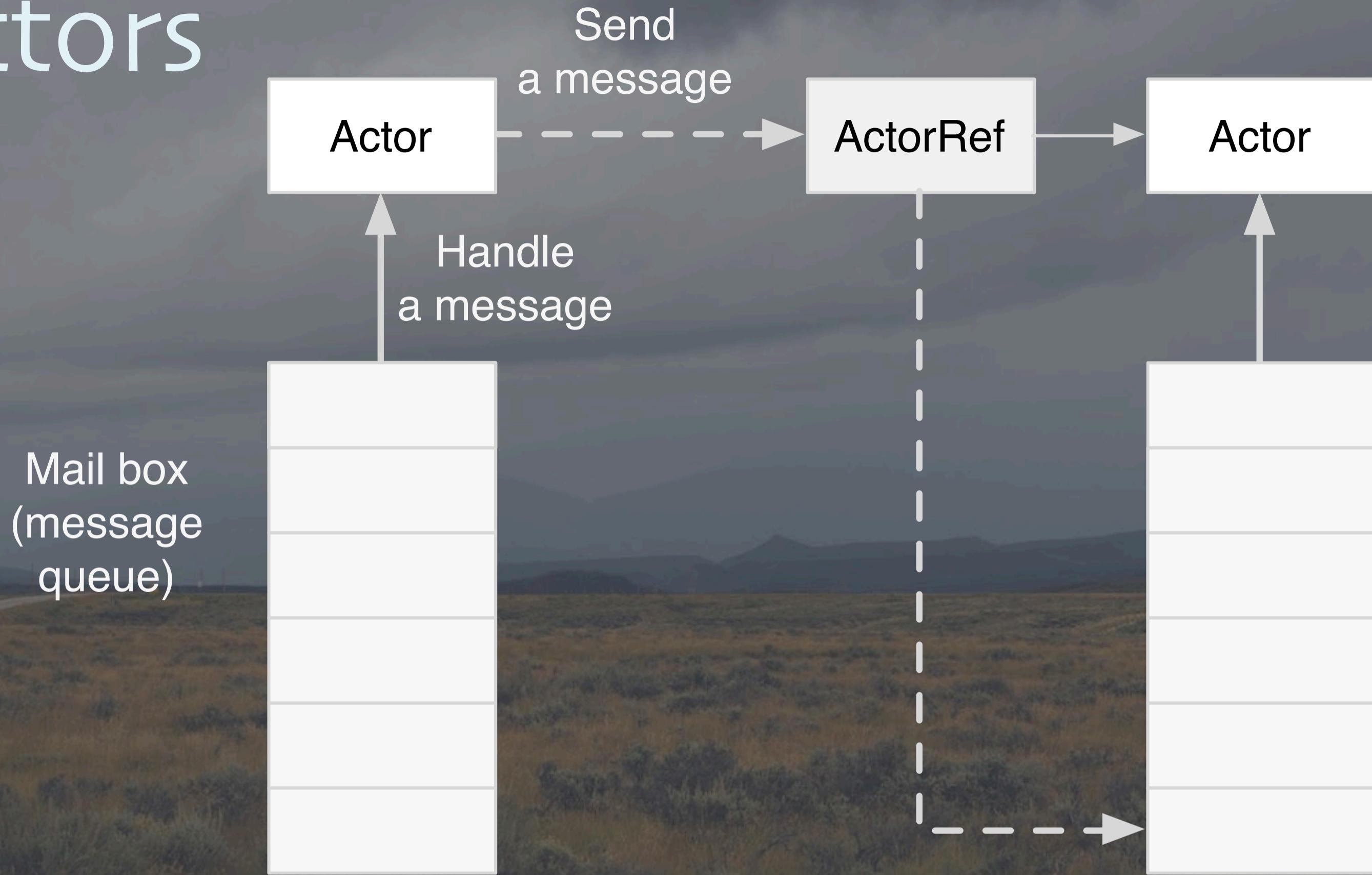
Let it Crash!

47

Tuesday, May 19, 15

Rather than attempt to recover from errors inside the domain logic (e.g., elaborate exception handling), allow services to fail, but with failure detection and reconstruction of those services, plus failover to other replicas.

Actors

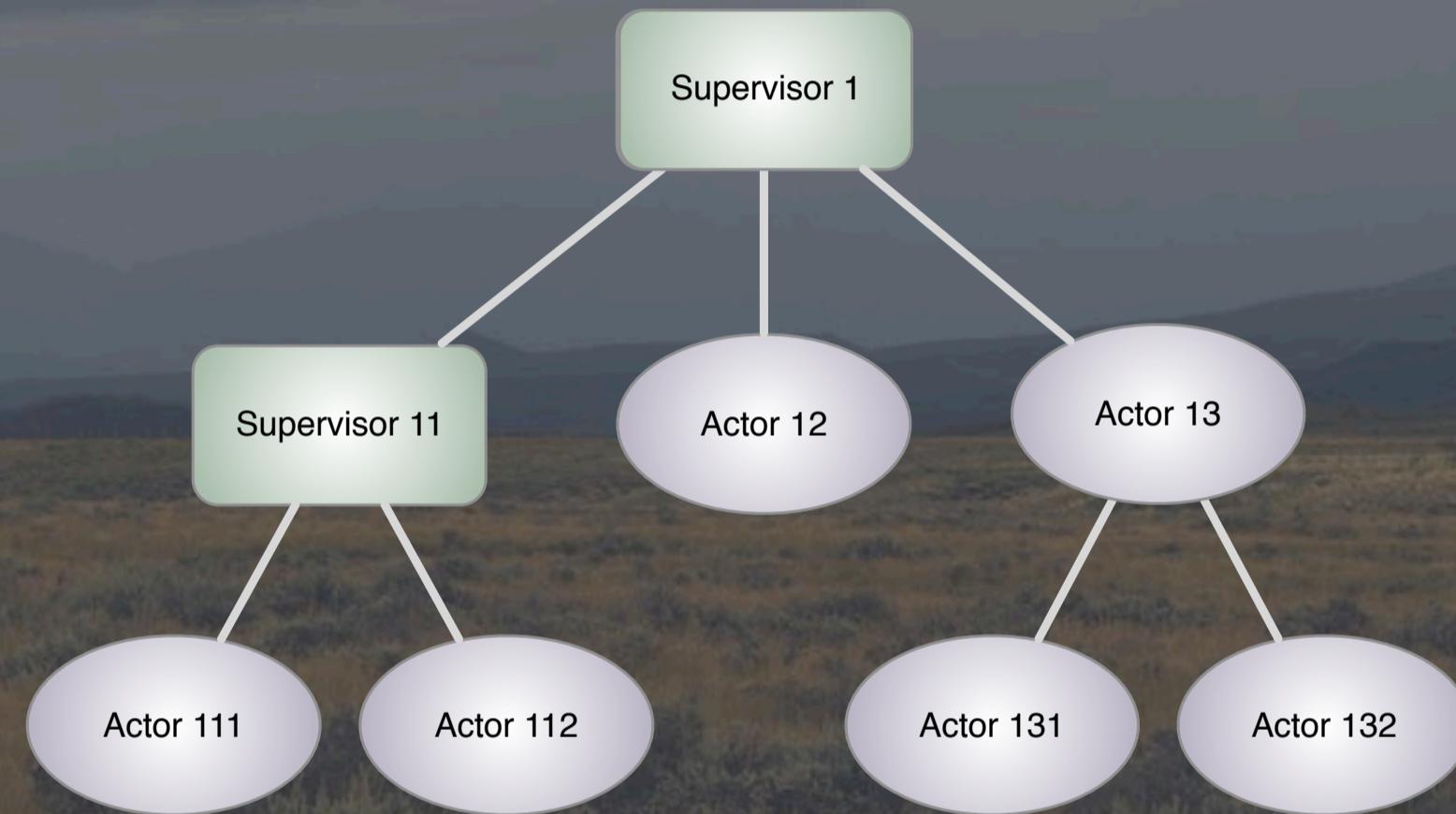


48

Tuesday, May 19, 15

Actors are similar to objects in Smalltalk and similar, message-passing systems; autonomous agents with defined boundaries that communicate through message passing. Actors, though process each message in a threadsafe way, so they are great for concurrency. (This diagram illustrates the Akka implementation – <http://akka.io>)

Erlang introduced supervisors. A hierarchy of actors that manage each “worker” actor’s lifecycle.

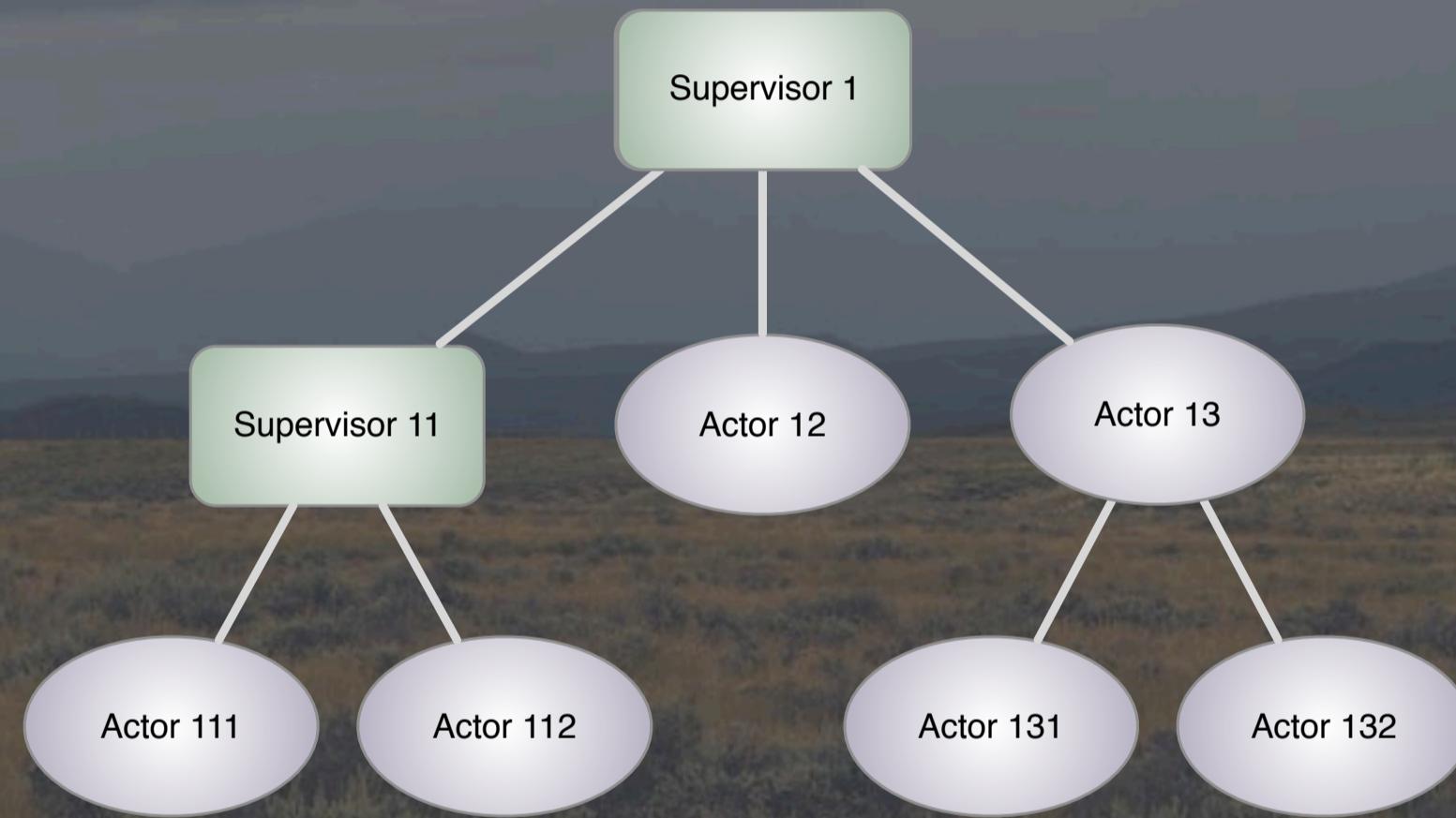


49

Tuesday, May 19, 15

There are lots of so-called Actor systems, but be wary of them unless they have this sophisticated supervision model or something like it (even though the original Actor model of Hewitt, et al., didn’t include supervision like this...).

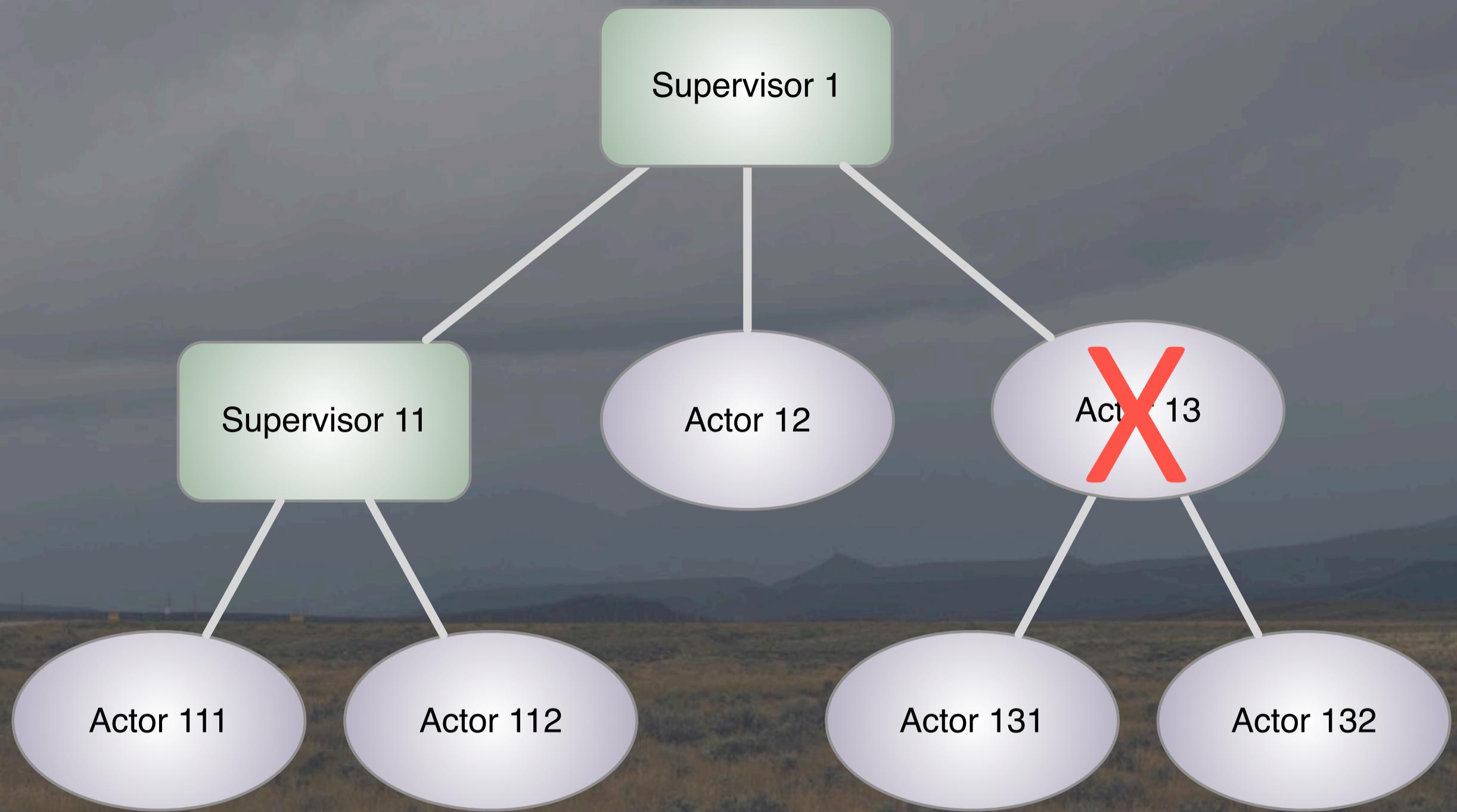
Generalizes nicely to distributed actor systems.



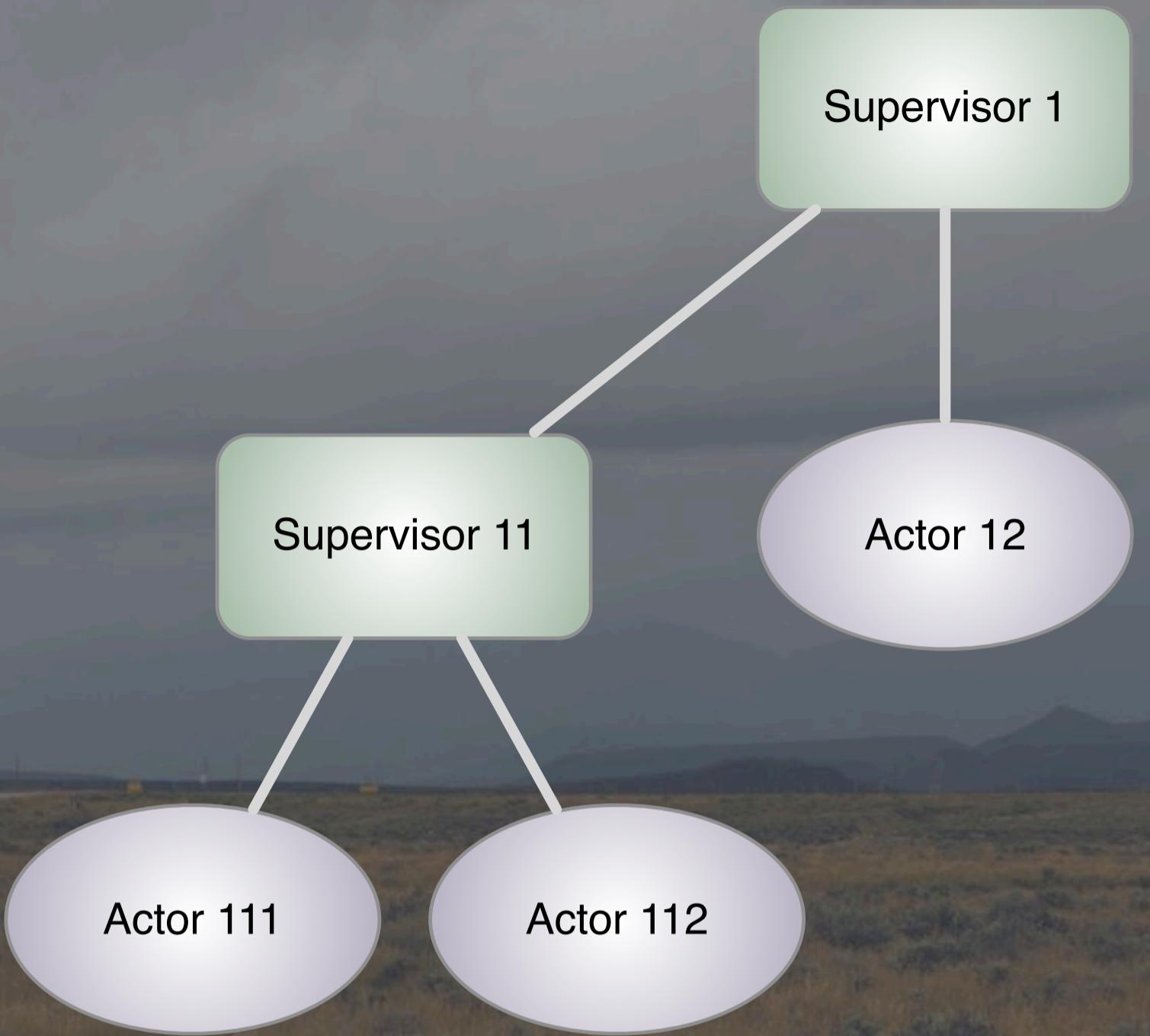
50

Tuesday, May 19, 15

Uniform abstraction when the lines cross process and machine boundaries.



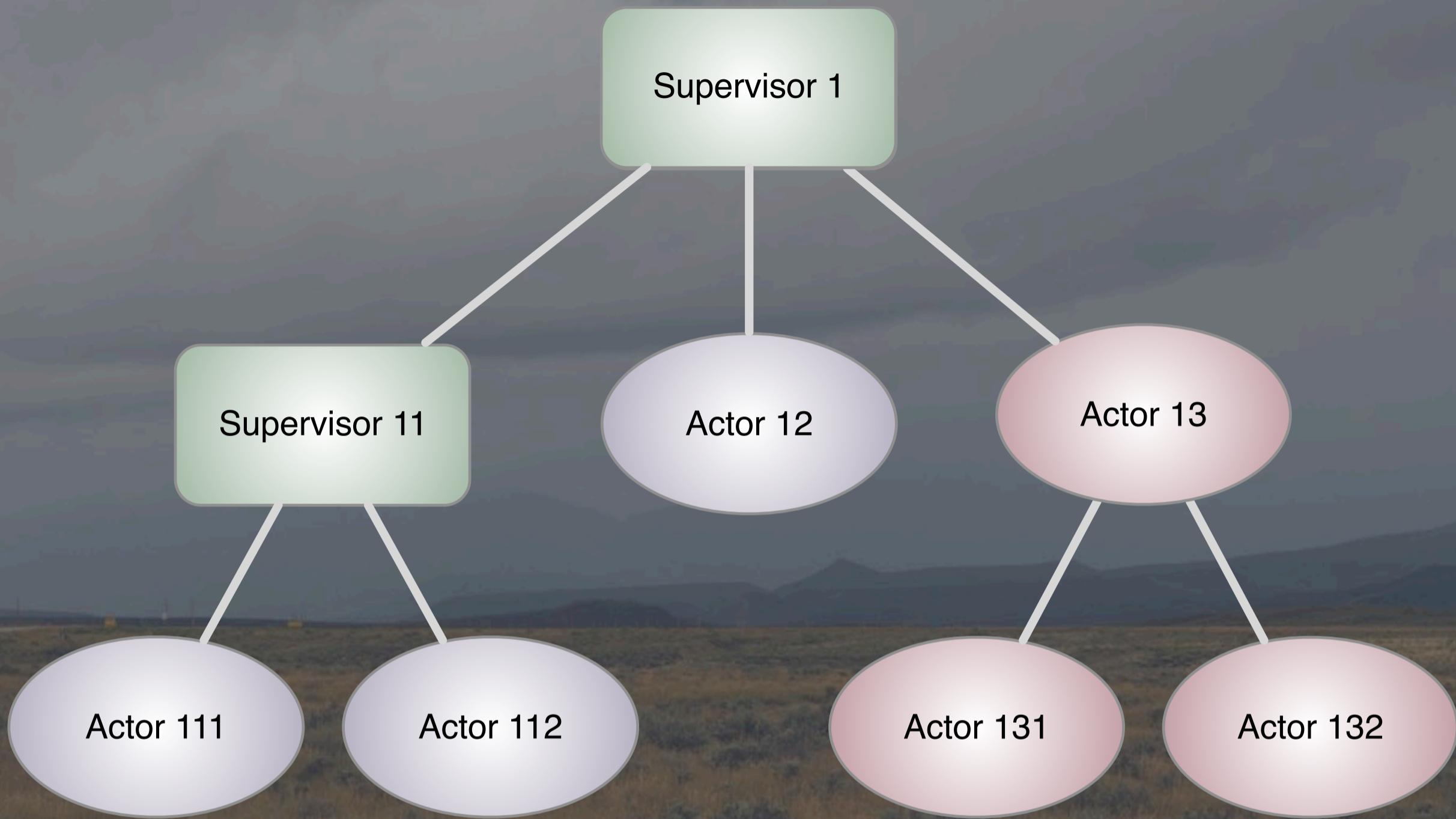
An actor dies. This one has children, too.

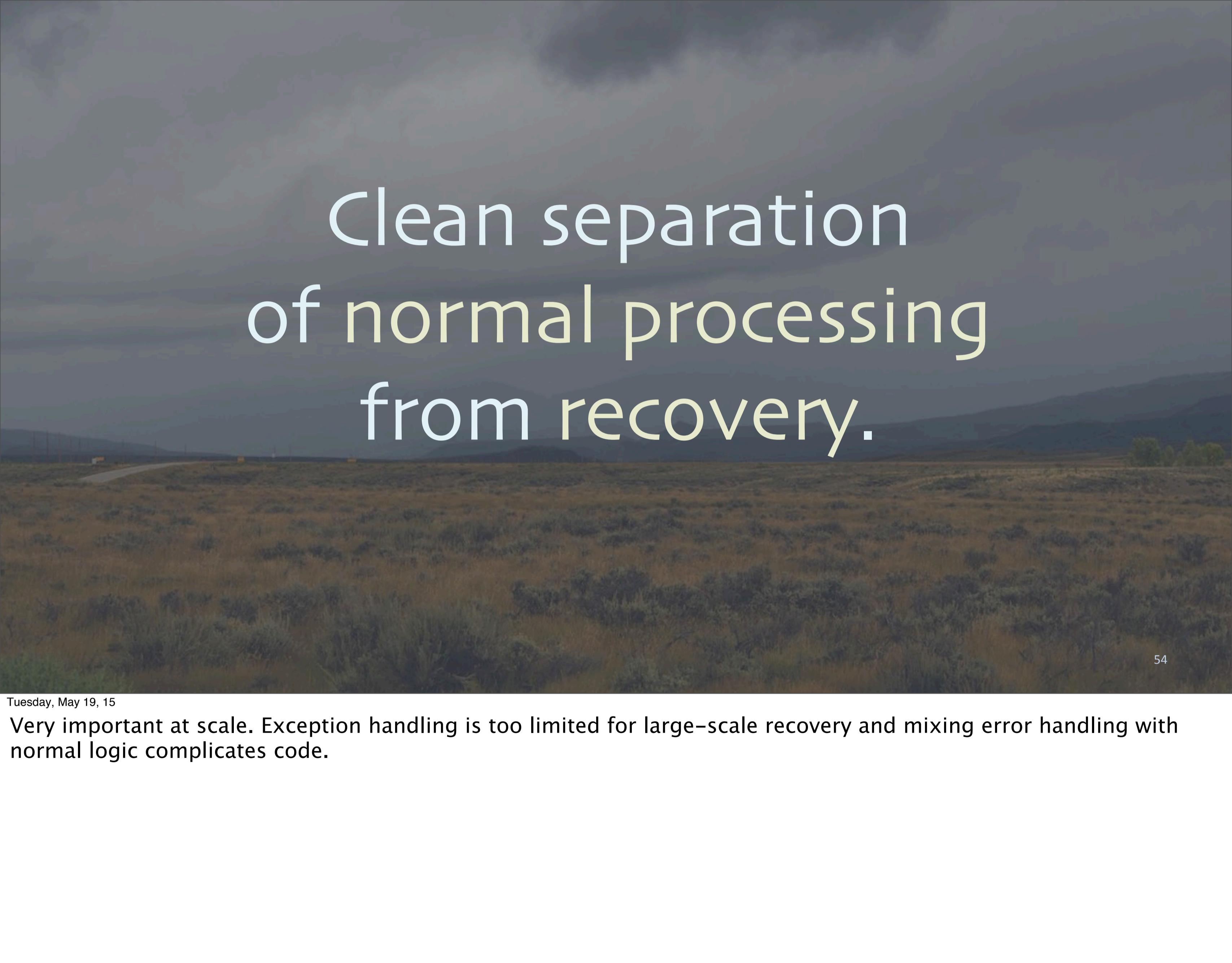


52

Tuesday, May 19, 15

The supervisor tears down the tree of dependent actors, then...



A landscape photograph showing a dry, brownish-yellow field in the foreground, stretching towards a range of mountains in the background under a clear sky.

Clean separation
of normal processing
from recovery.

54

Tuesday, May 19, 15

Very important at scale. Exception handling is too limited for large-scale recovery and mixing error handling with normal logic complicates code.



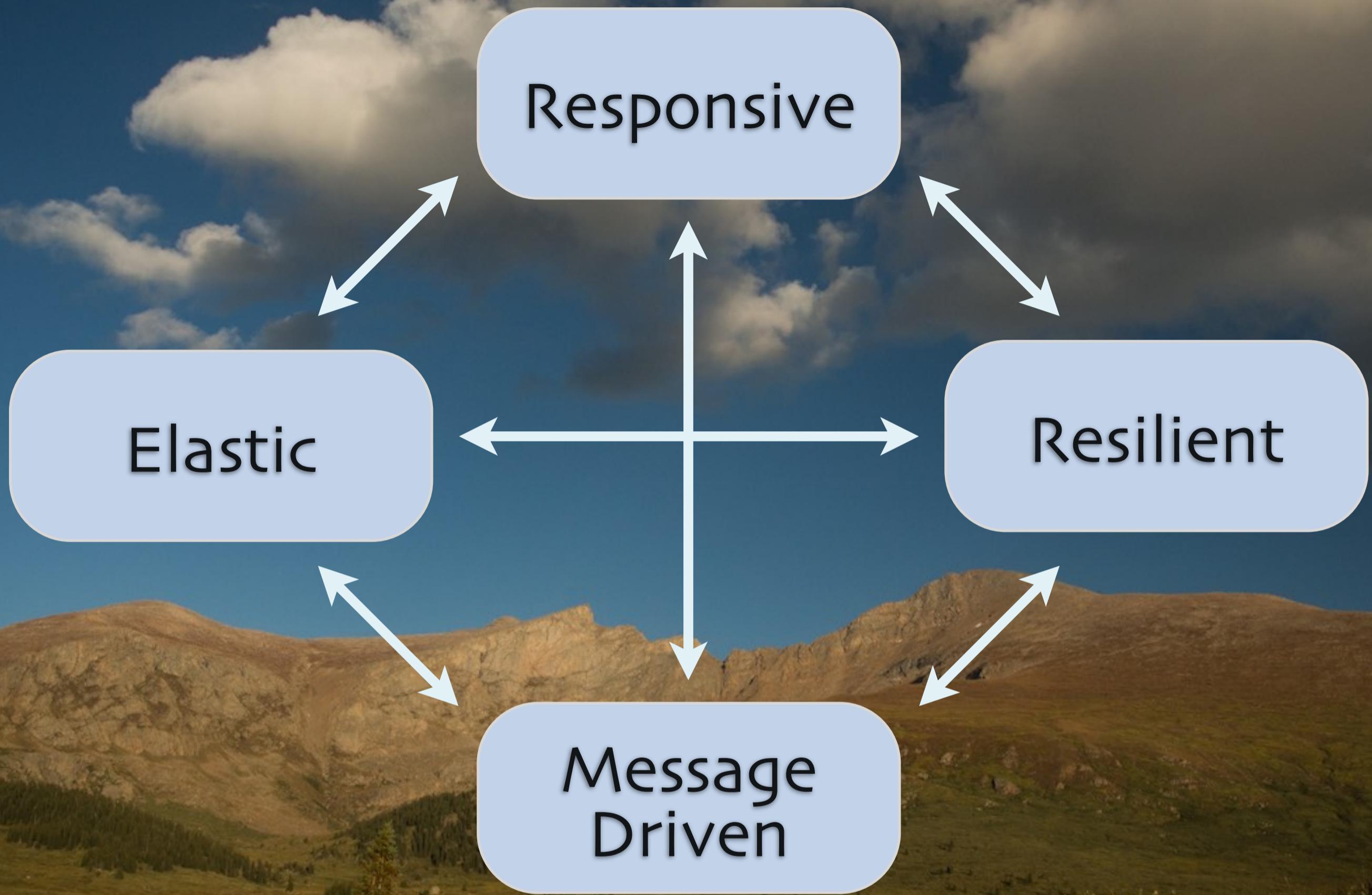
Not using Actors?
Consider Hystrix
from Netflix or similar...

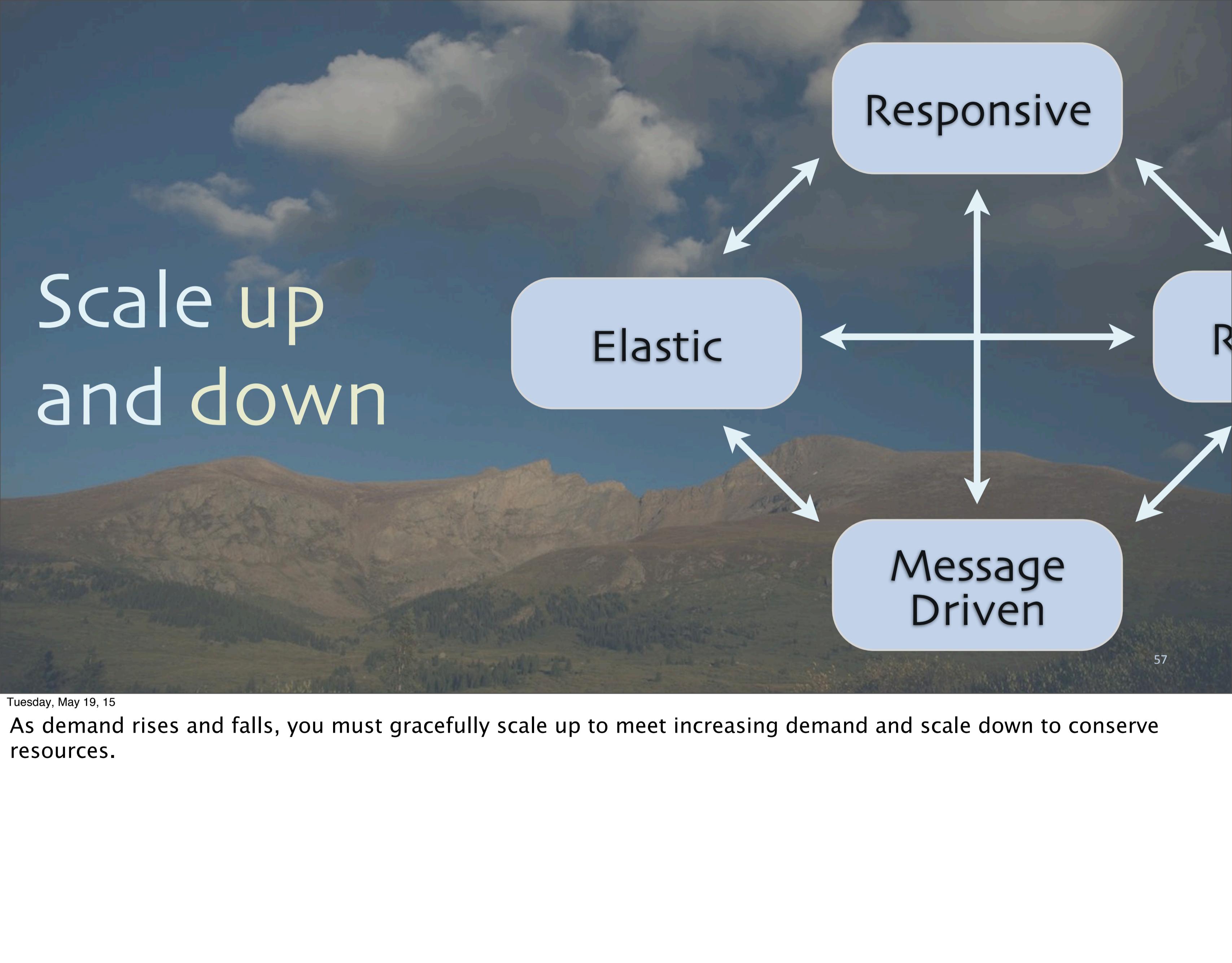
<https://github.com/Netflix/Hystrix>

55

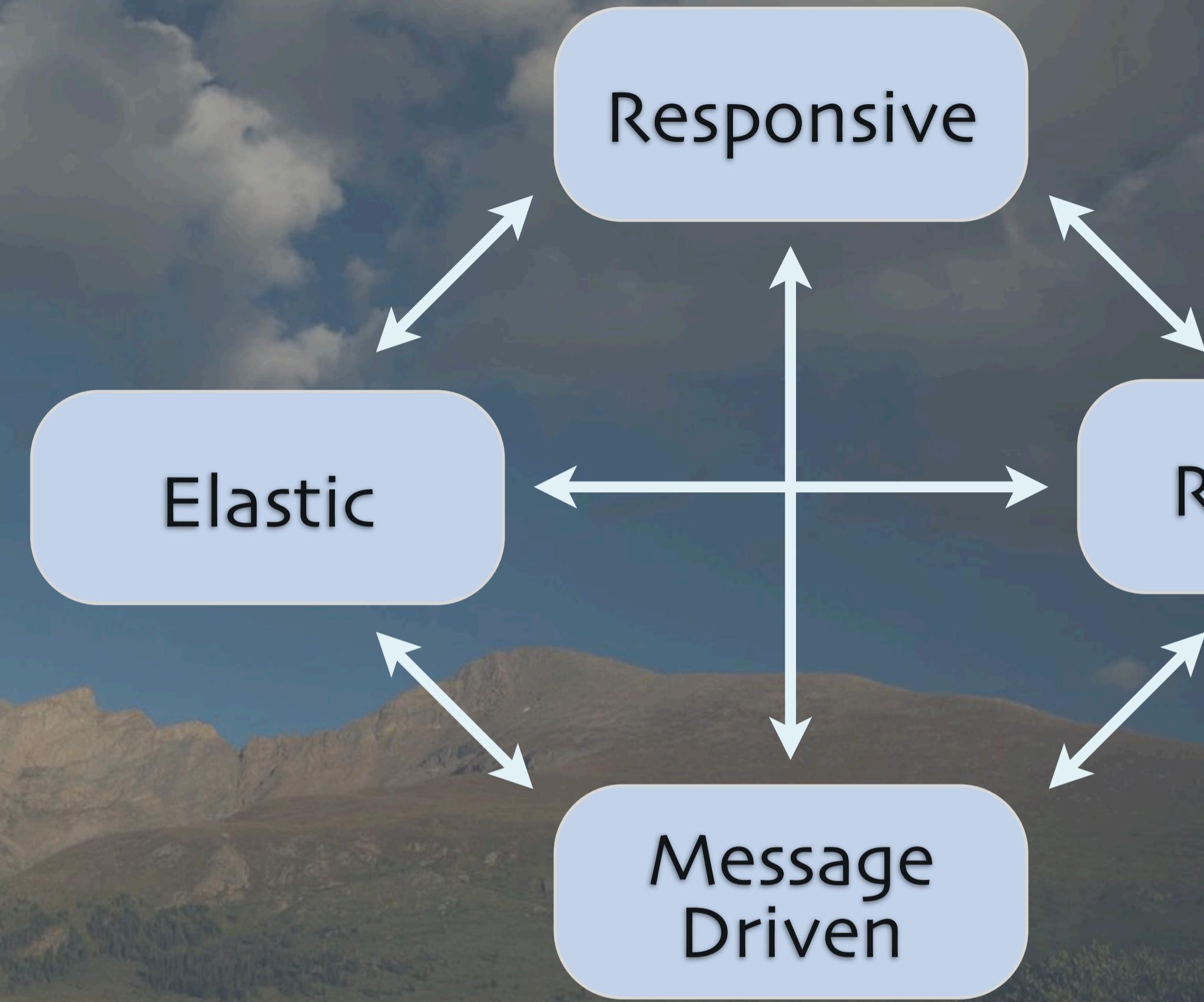
Tuesday, May 19, 15

I believe every environment must solve this problem one way or another.





Scale up and down



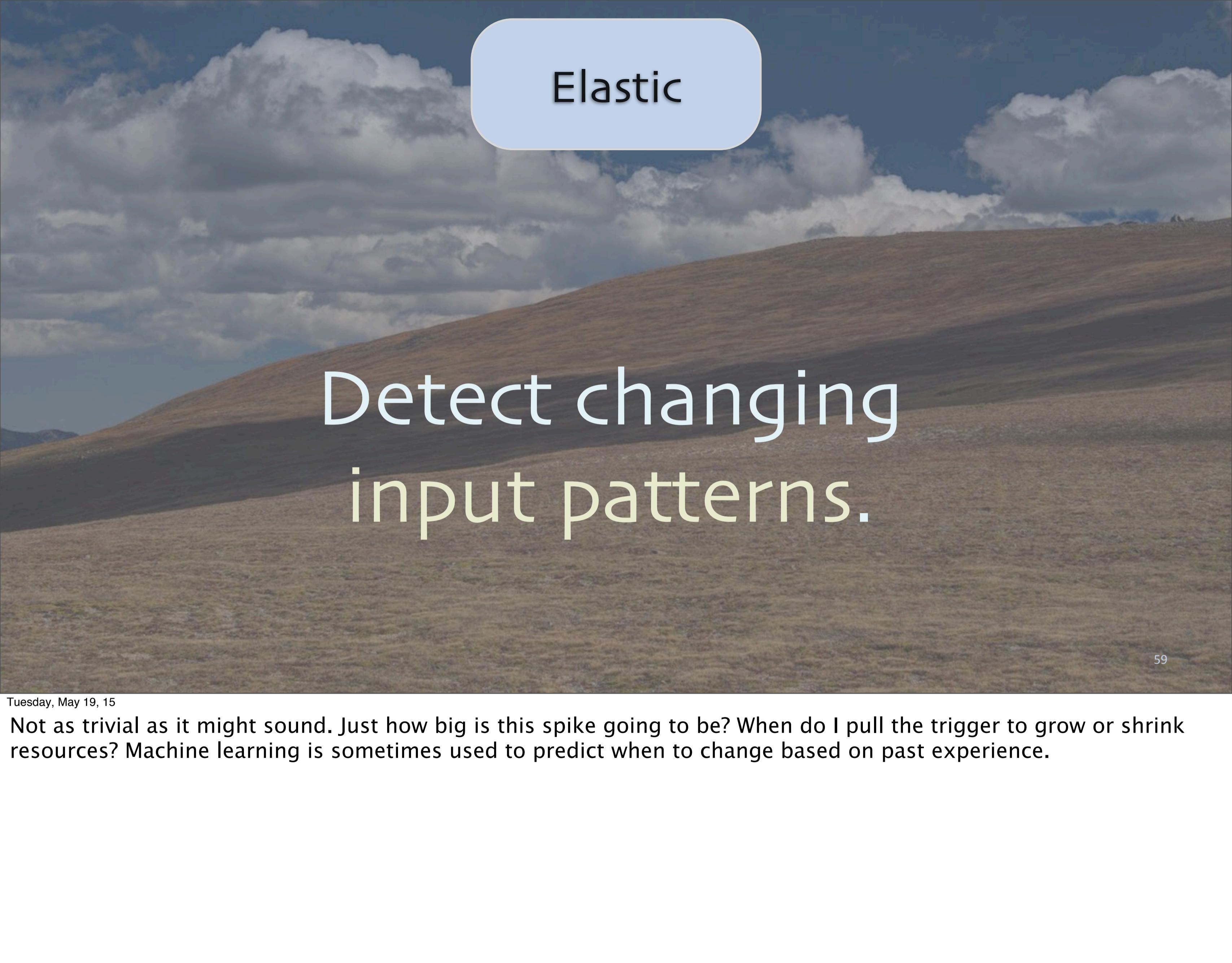
57

Tuesday, May 19, 15

As demand rises and falls, you must gracefully scale up to meet increasing demand and scale down to conserve resources.



Elastic



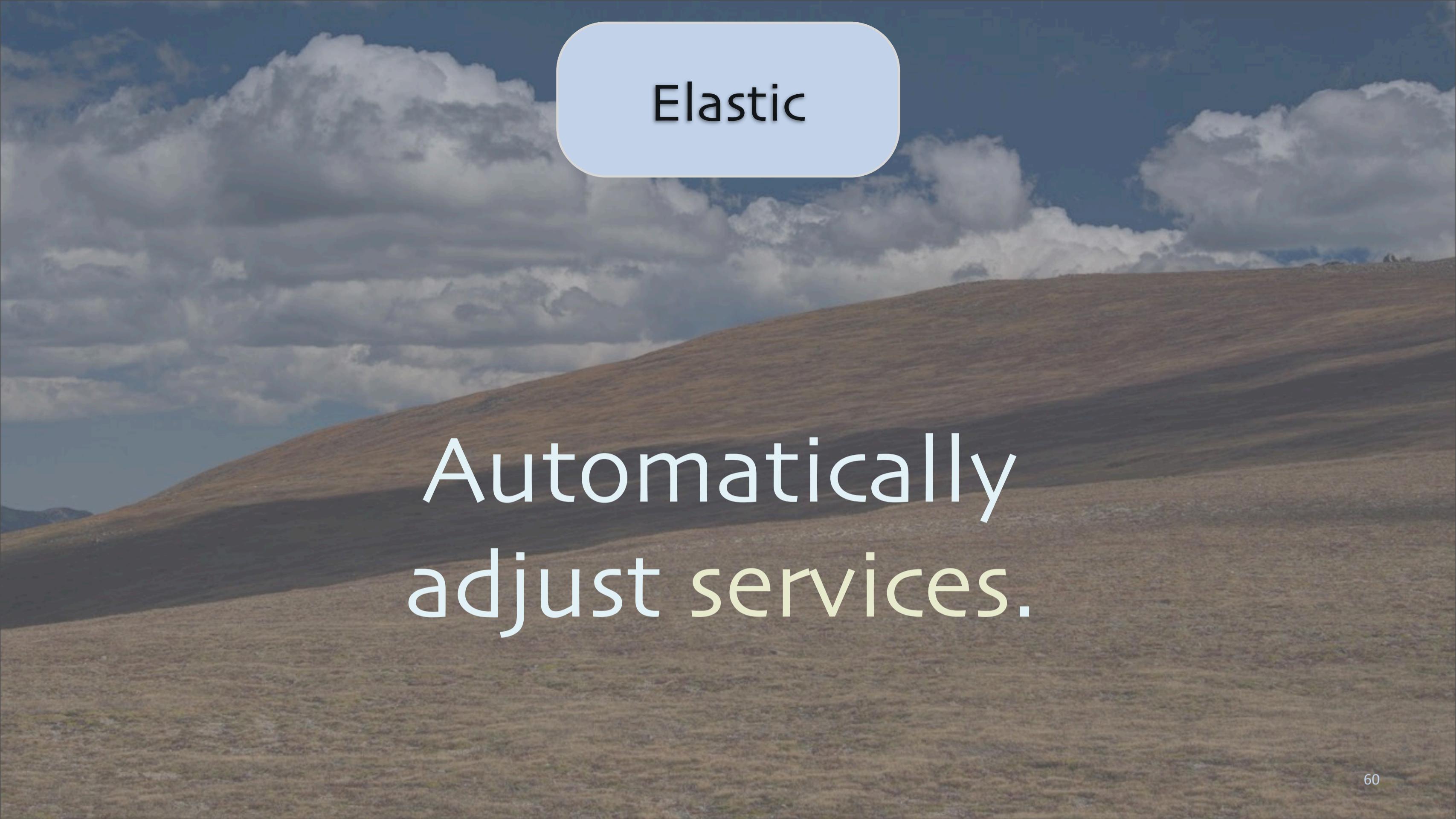
Elastic

Detect changing
input patterns.

59

Tuesday, May 19, 15

Not as trivial as it might sound. Just how big is this spike going to be? When do I pull the trigger to grow or shrink resources? Machine learning is sometimes used to predict when to change based on past experience.



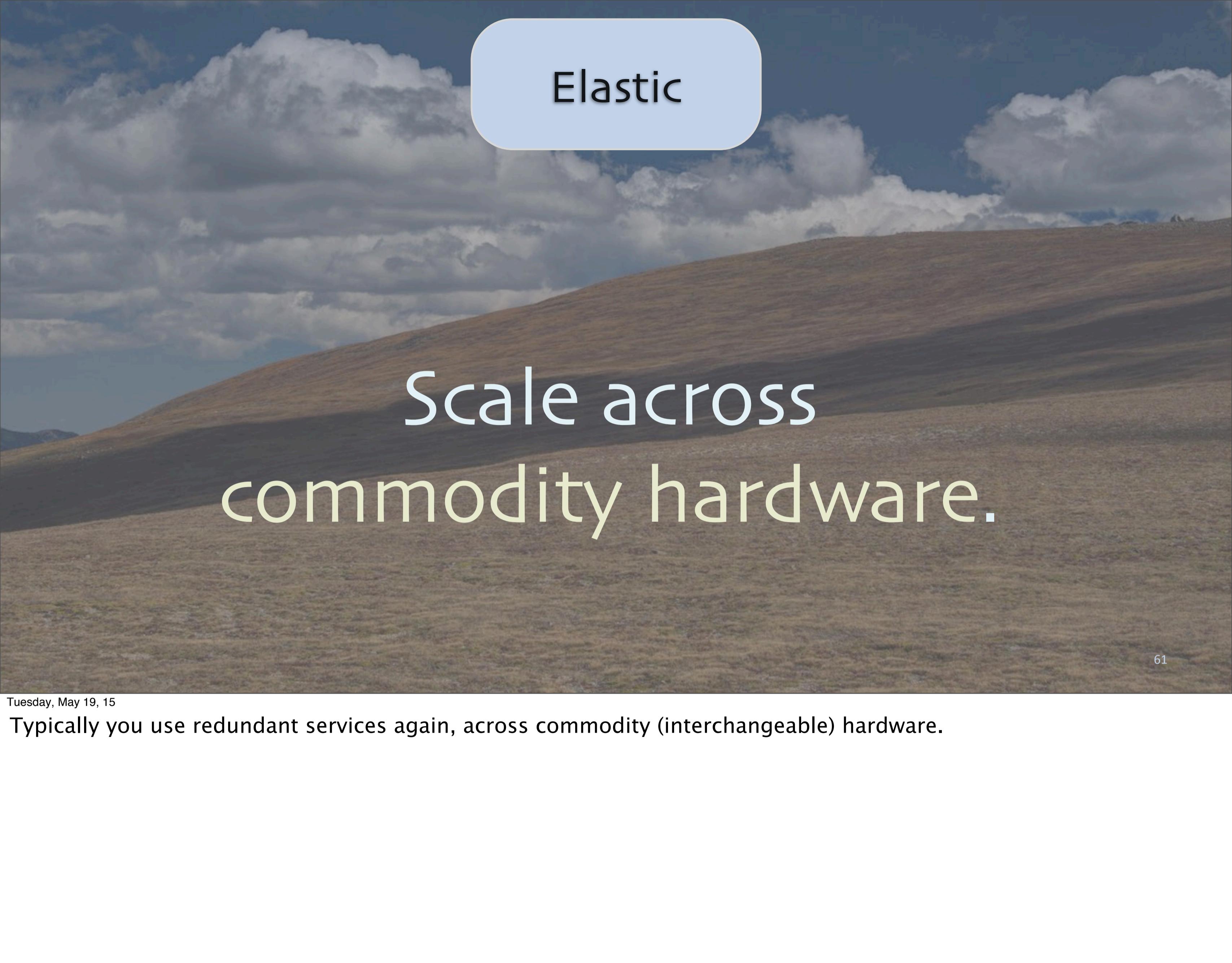
Elastic

Automatically
adjust services.

60

Tuesday, May 19, 15

Human intervention has to be minimal for this to really work.



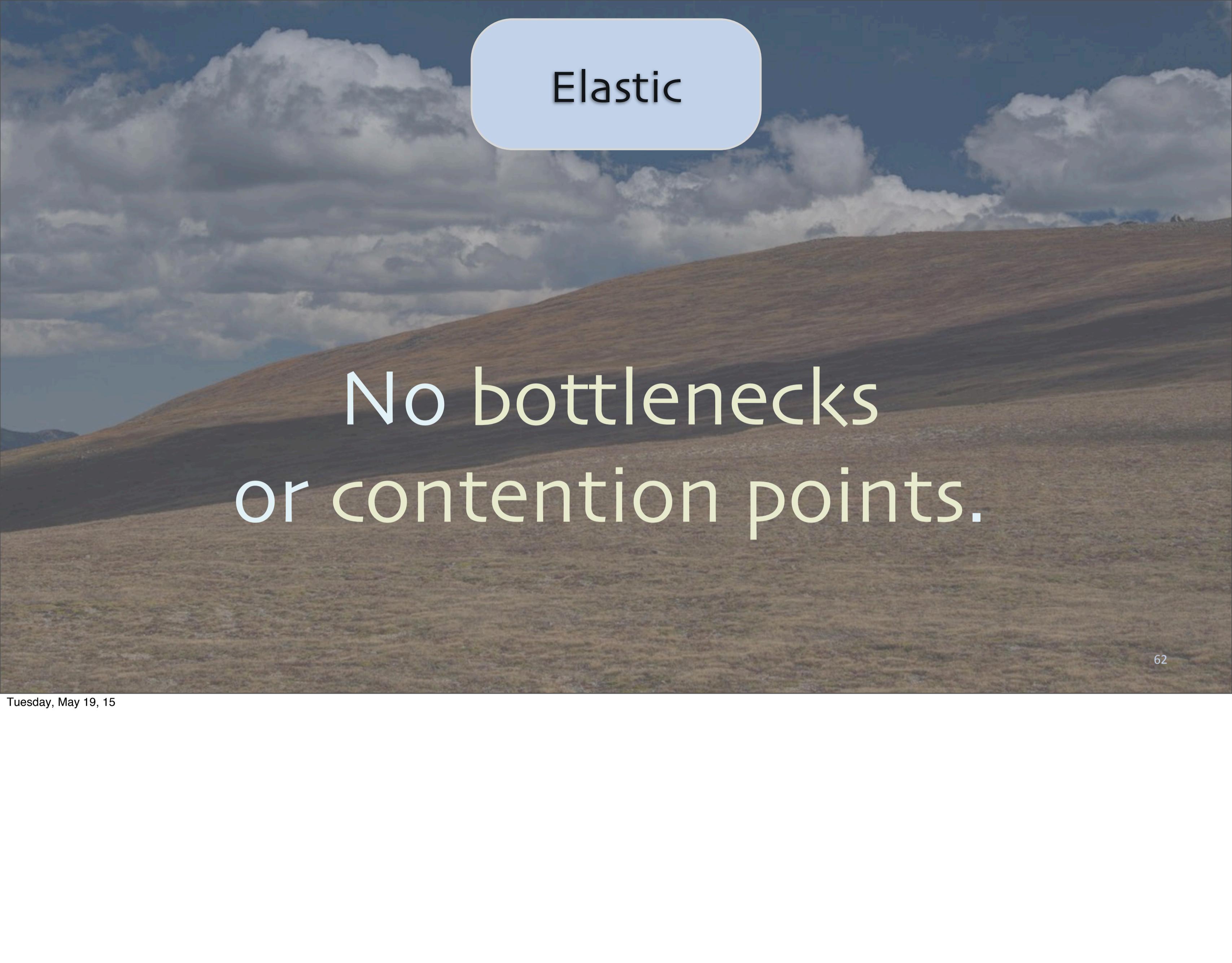
Elastic

Scale across
commodity hardware.

61

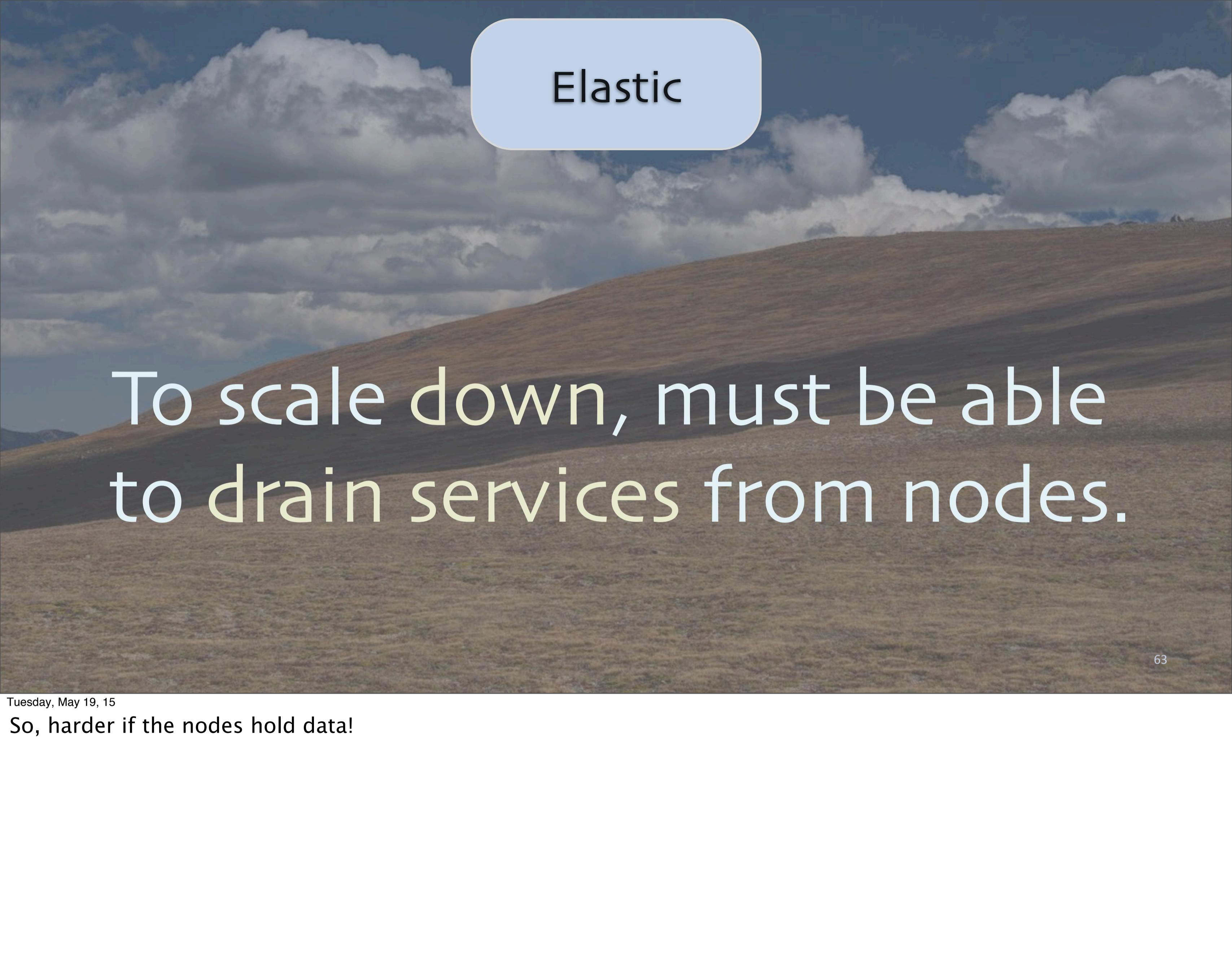
Tuesday, May 19, 15

Typically you use redundant services again, across commodity (interchangeable) hardware.



Elastic

No bottlenecks
or contention points.



Elastic

To scale down, must be able
to drain services from nodes.

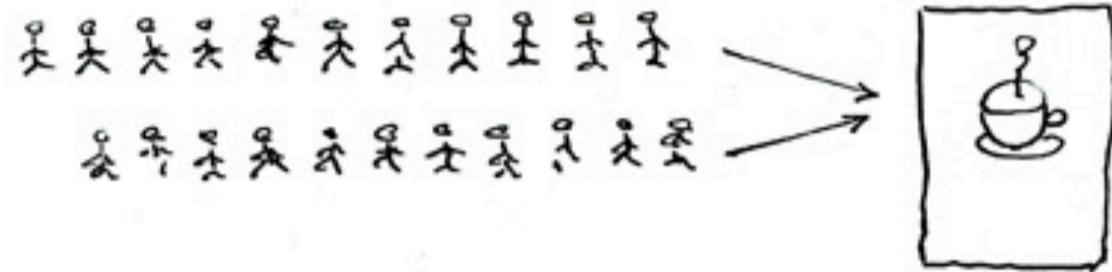
63

Tuesday, May 19, 15

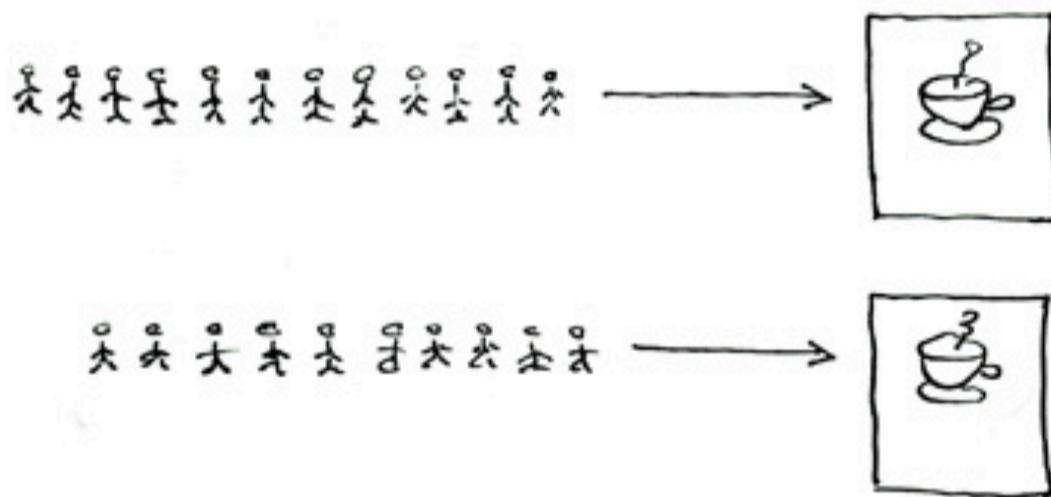
So, harder if the nodes hold data!

Elastic

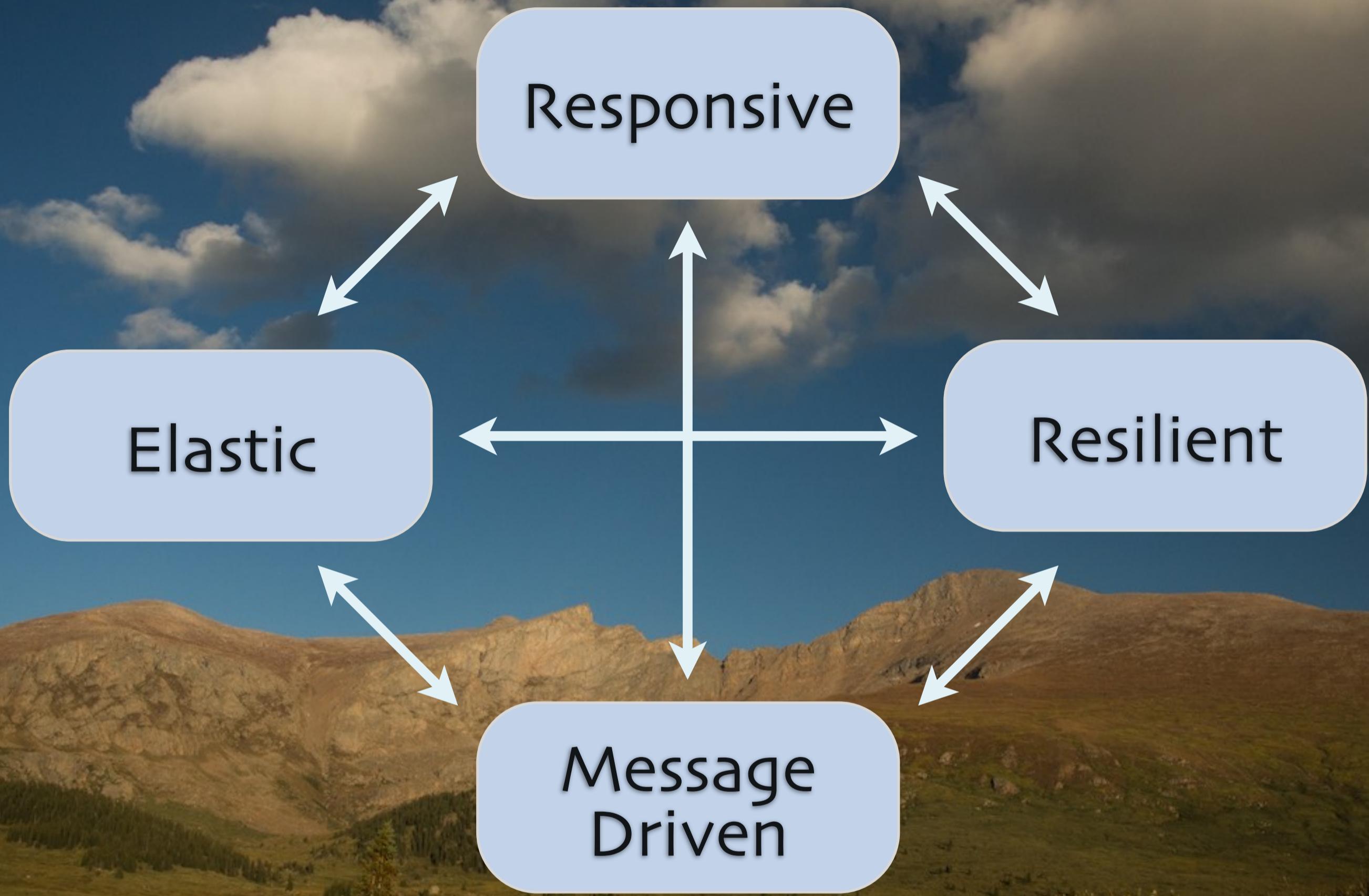
Concurrent = Two Queues One Coffee Machine



Parallel = Two Queues Two Coffee Machines



© Joe Armstrong 2013



```
graph TD; A([Elastic]) <--> B([Resilient]); A <--> C([Message Driven]); B <--> C
```

Elastic

Resilient

Message
Driven

To react, you must
be message driven.



Message
Driven



Message
Driven

Asynchronous
message passing.

68

Tuesday, May 19, 15

It can't be command and control. Blocking while waiting for a response fails to scale. (See Amdahl's Law)



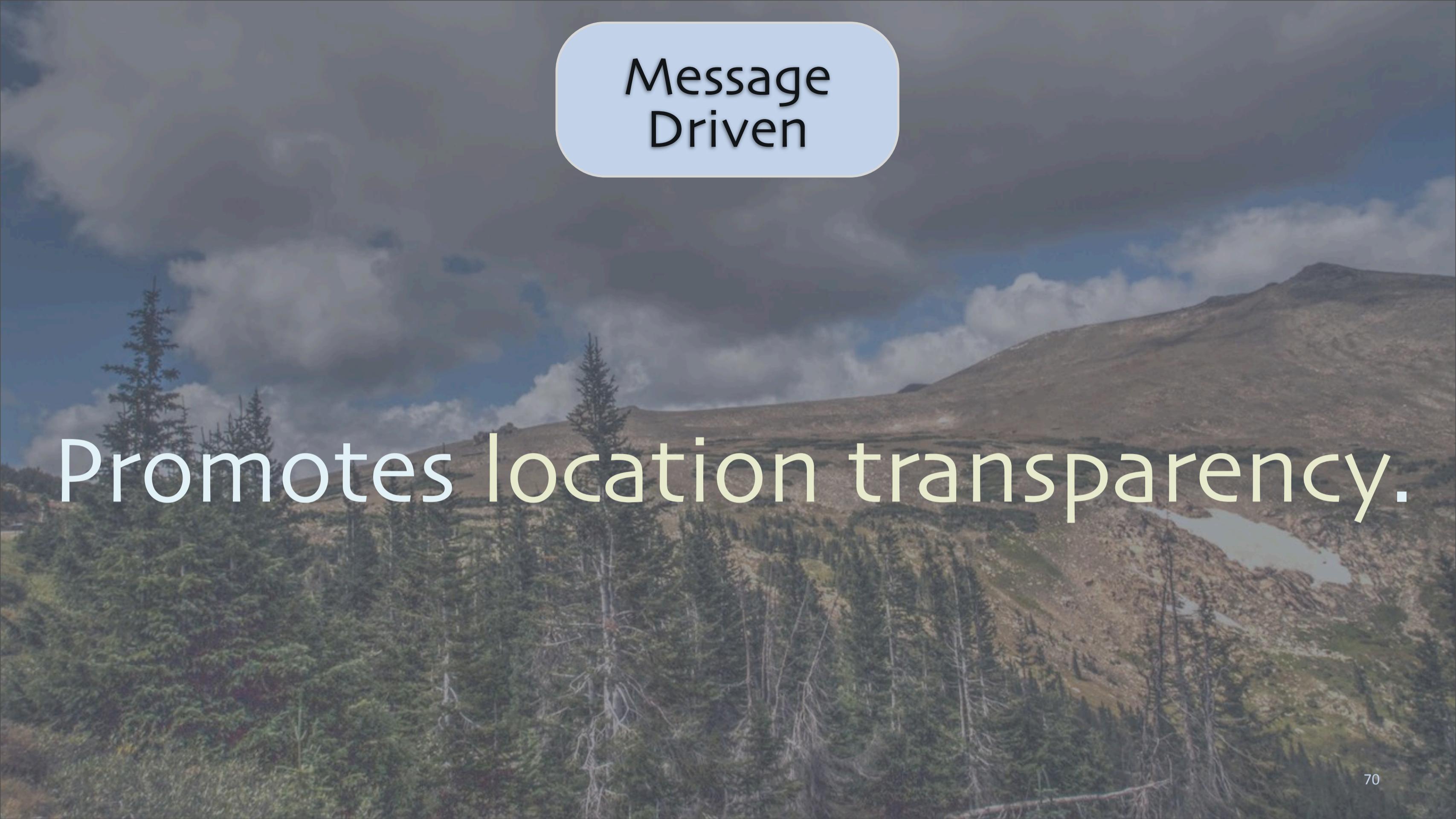
Message Driven

Defines boundaries,
promotes loose coupling
and isolation.

69

Tuesday, May 19, 15

Clear separation between components (whether or not the messages cross process boundaries), which encourages effective decomposition into focused services that are isolated from each other and loosely coupled.



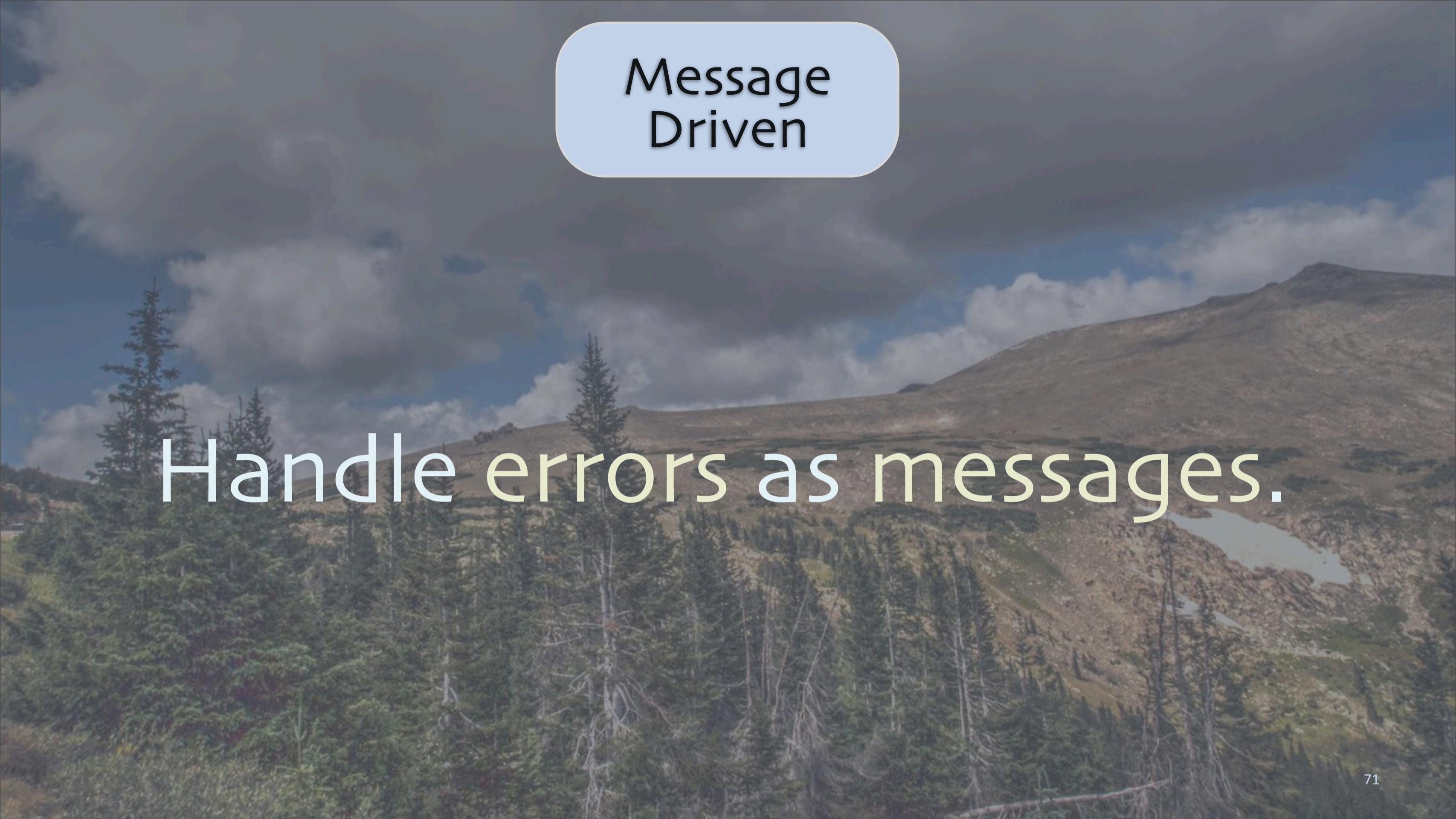
Message
Driven

Promotes location transparency.

70

Tuesday, May 19, 15

Source and receiver can change, so services can be migrated to adapt to changing load dynamics.



Message
Driven

Handle errors as messages.

71

Tuesday, May 19, 15

Also, you can use the same message infrastructure to communicate error scenarios as well as normal processing.



Message Driven

Promotes global management
and flow control
through back pressure.

72

Tuesday, May 19, 15

Think of the messages as forming a stream. If a common implementation infrastructure is used, it's possible to monitor and manage traffic flow. Back pressure is the idea of communication between sender and receiver to control the rate of flow. We'll return to it.

Reactive Extensions (Rx)

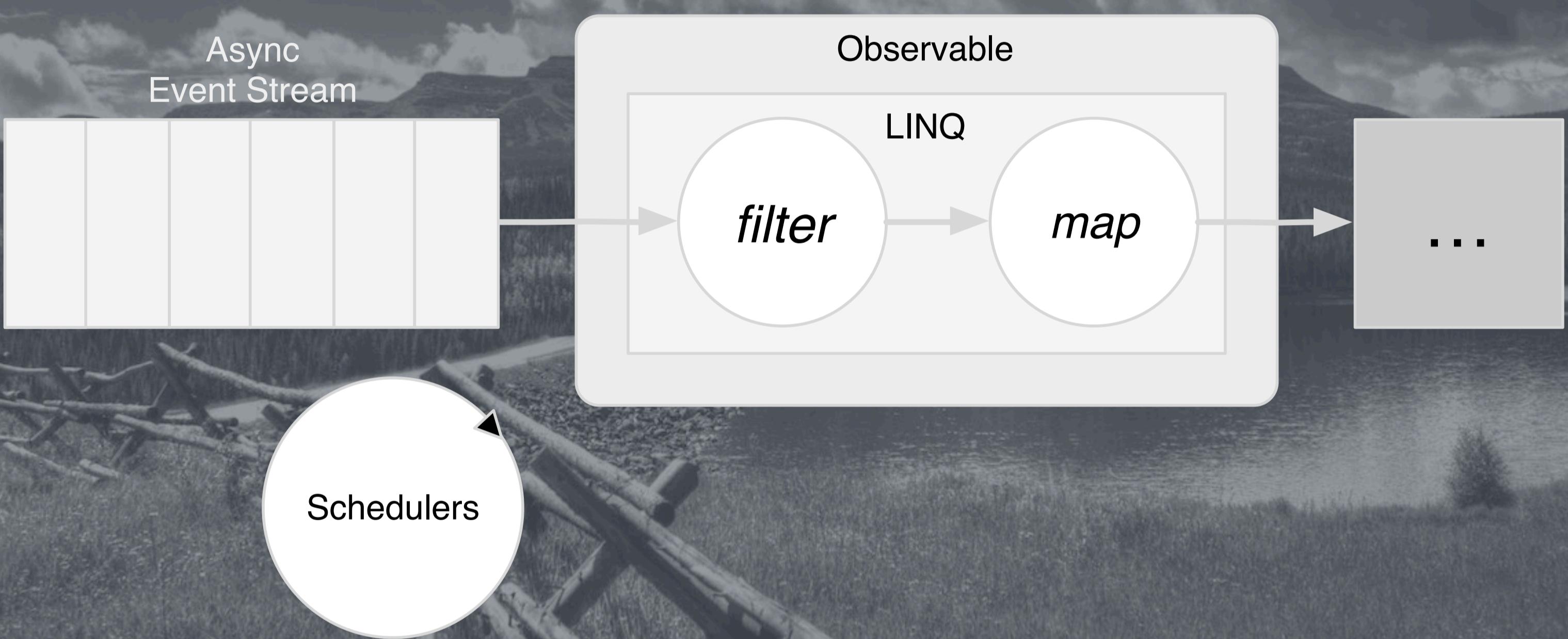


73

Tuesday, May 19, 15

Rx was pioneered by Erik Meijer for .NET. Now ported to several languages, including RxJava (Netflix) and React (JS - Facebook).

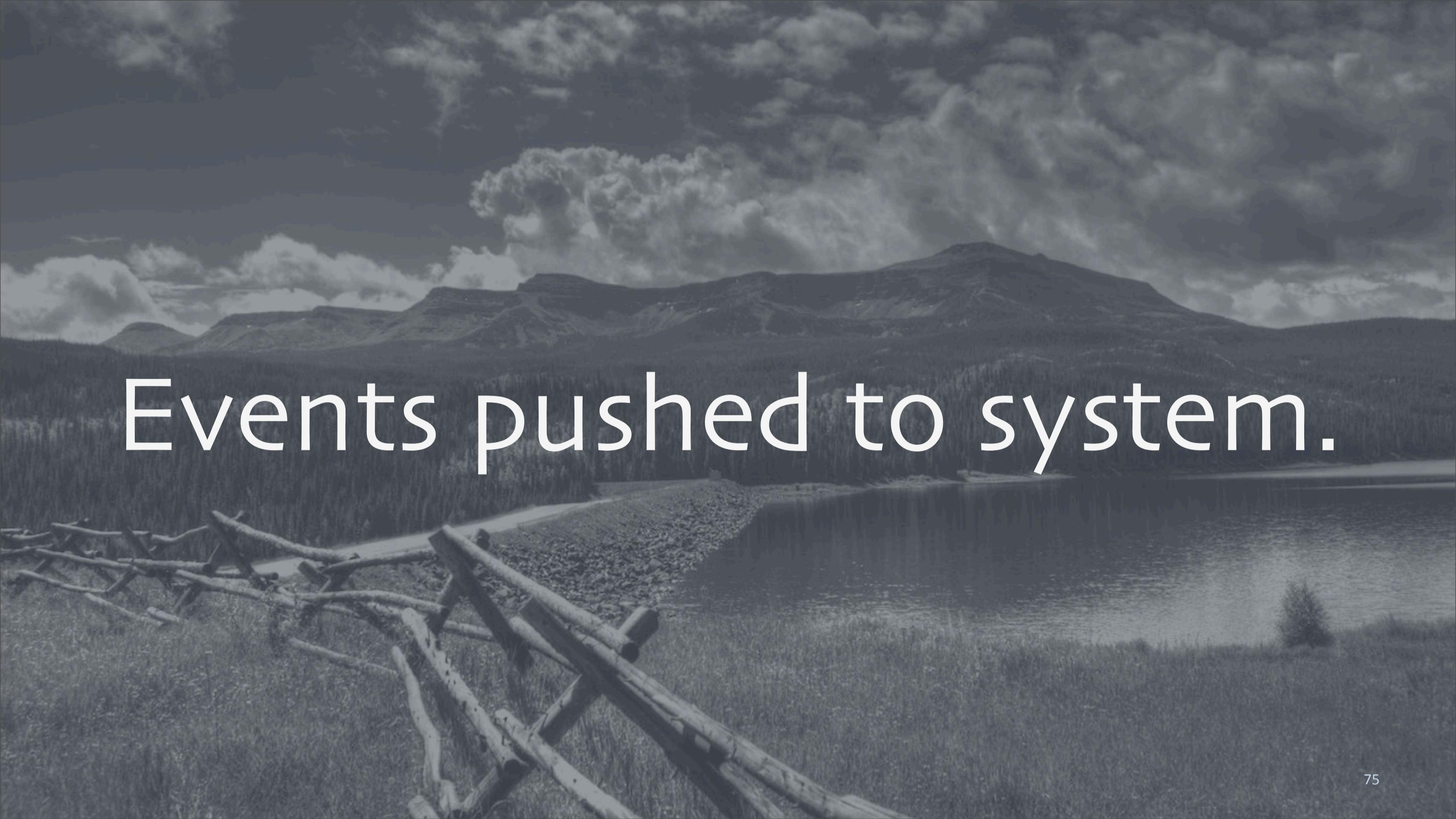
LINQ Rx



74

Tuesday, May 19, 15

Events are observed (an extension of the observer pattern). Operations like filtering and mapping are provided to work with the stream through LINQ (Language Integrated Query), which uses SQL-like expressions. The Schedulers are used to trigger processing.

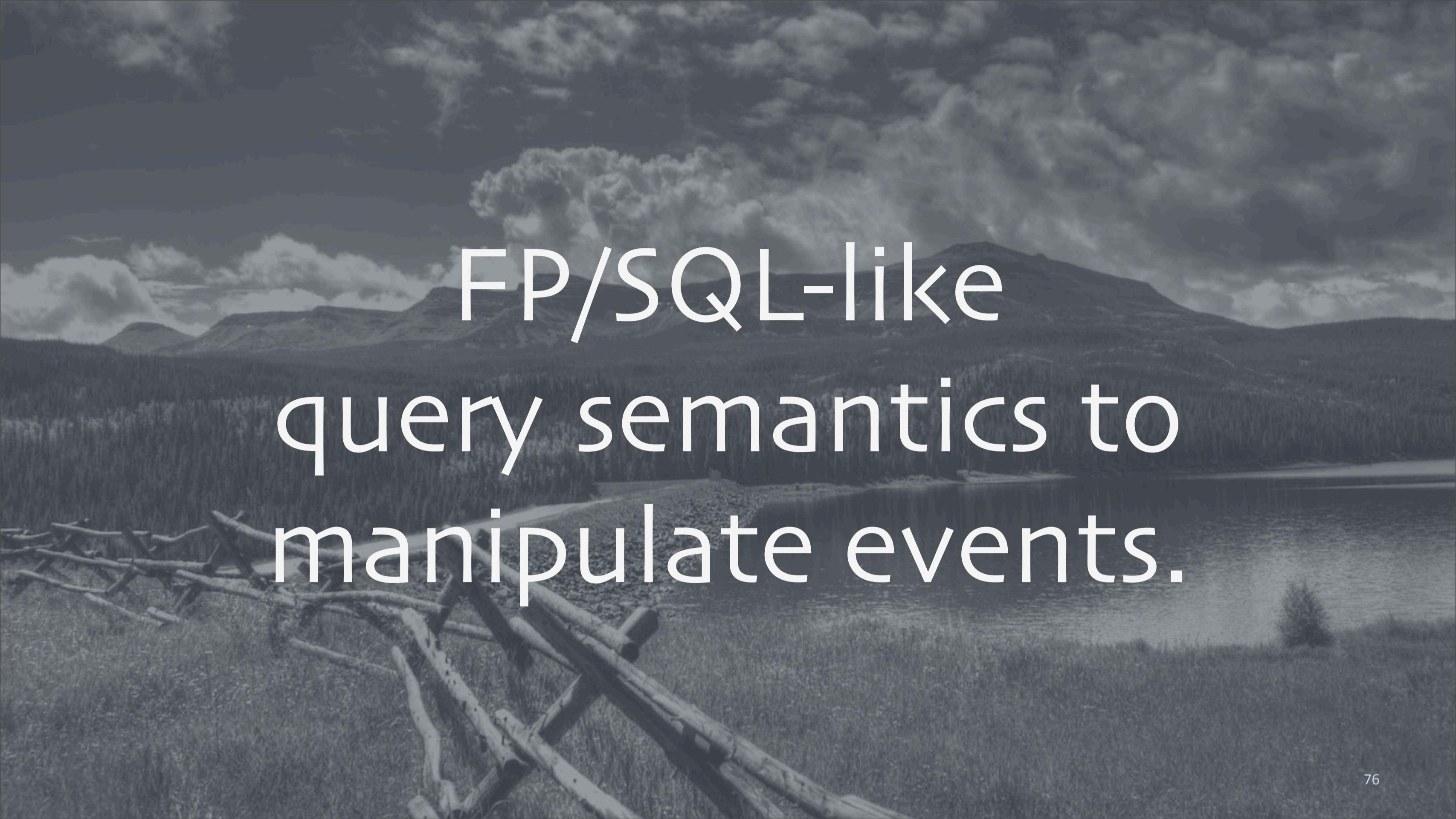


Events pushed to system.

75

Tuesday, May 19, 15

It's essentially a push model.



FP/SQL-like
query semantics to
manipulate events.

How: Reactive Tools



77

Tuesday, May 19, 15

We mentioned a few already, let's fill in some details. This won't be an exhaustive list.
Hat tip to Jamie Allen at Typesafe for some of these ideas.

How: Reactive Tools

- Functional Programming
- Distributed Computing “Laws”
- Software Transactional Mem.
- Event Loops

How: Reactive Tools

- CSP
- Futures
- Actors - Erlang or Akka
- Rx and variants
- Reactive Streams

Functional Programming



80

Tuesday, May 19, 15

Functional Programming

Prefer immutable values and
side-effect free functions...

Functional Programming

... because they eliminate the problems of multithreading.

Functional Programming

Objects - suitable for modules.
Functions - for everything else.

83

Tuesday, May 19, 15

This is Scala's view, that objects are useful as module constructs, but the code inside should be functional.

Architecture Side Note:

The biggest mistake of OOP was the idea that we should faithfully model the world in code.

84

Tuesday, May 19, 15

Controversial, but I believe much of our code bloat and inflexibility is actually caused by this mistaken belief. Example: Does a payroll calculator need the concepts of Pay, Deductions, etc.? Or should we just stream numbers through math logic?

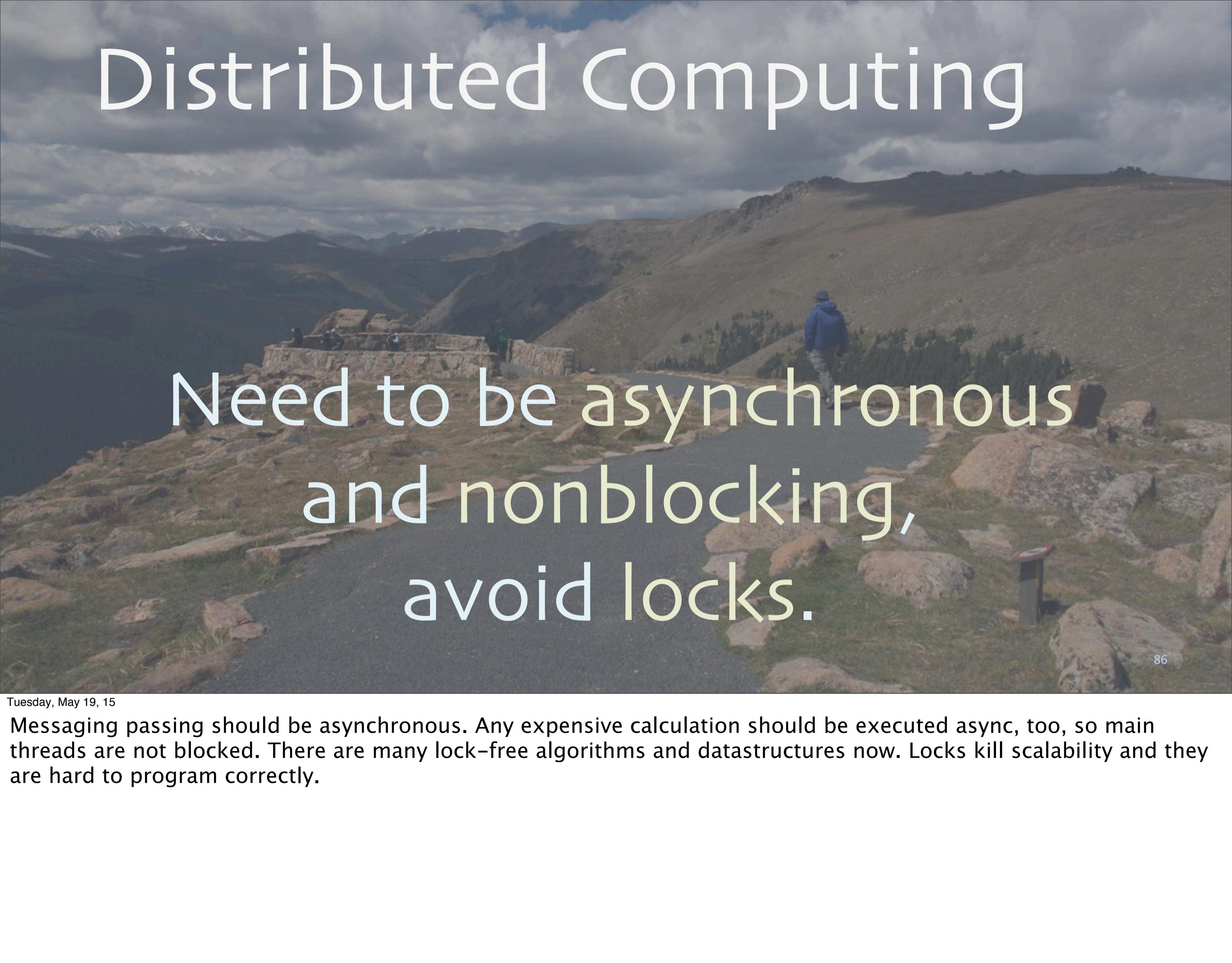
Distributed Computing



85

Tuesday, May 19, 15

Distributed Computing

A scenic mountain landscape with a hiker on a trail.

Need to be asynchronous
and nonblocking,
avoid locks.

86

Tuesday, May 19, 15

Messaging passing should be asynchronous. Any expensive calculation should be executed async, too, so main threads are not blocked. There are many lock-free algorithms and datastructures now. Locks kill scalability and they are hard to program correctly.

Distributed Computing

Serializability (order) and
Linearizability (change history
results in same order?).
CRDTs, Lattices.

87

Tuesday, May 19, 15

CRDTs – Commutative Replicated Data Types (<http://pagesperso-systeme.lip6.fr/Marc.Shapiro/papers/RR-6956.pdf>)
Lattices are more general concept applied here.

Software Transactional Memory



88

Tuesday, May 19, 15

Popularized first in Hardware, then in Software by Haskell. Now used in persistent datastructures in many languages.
Great description of STM by Simon Peyton-Jones, from the O'Reilly Book Beautiful Code, <http://research.microsoft.com/en-us/um/people/simonpj/papers/stm/#beautiful>

Software Transactional Memory

Basically, ACID
without the D.

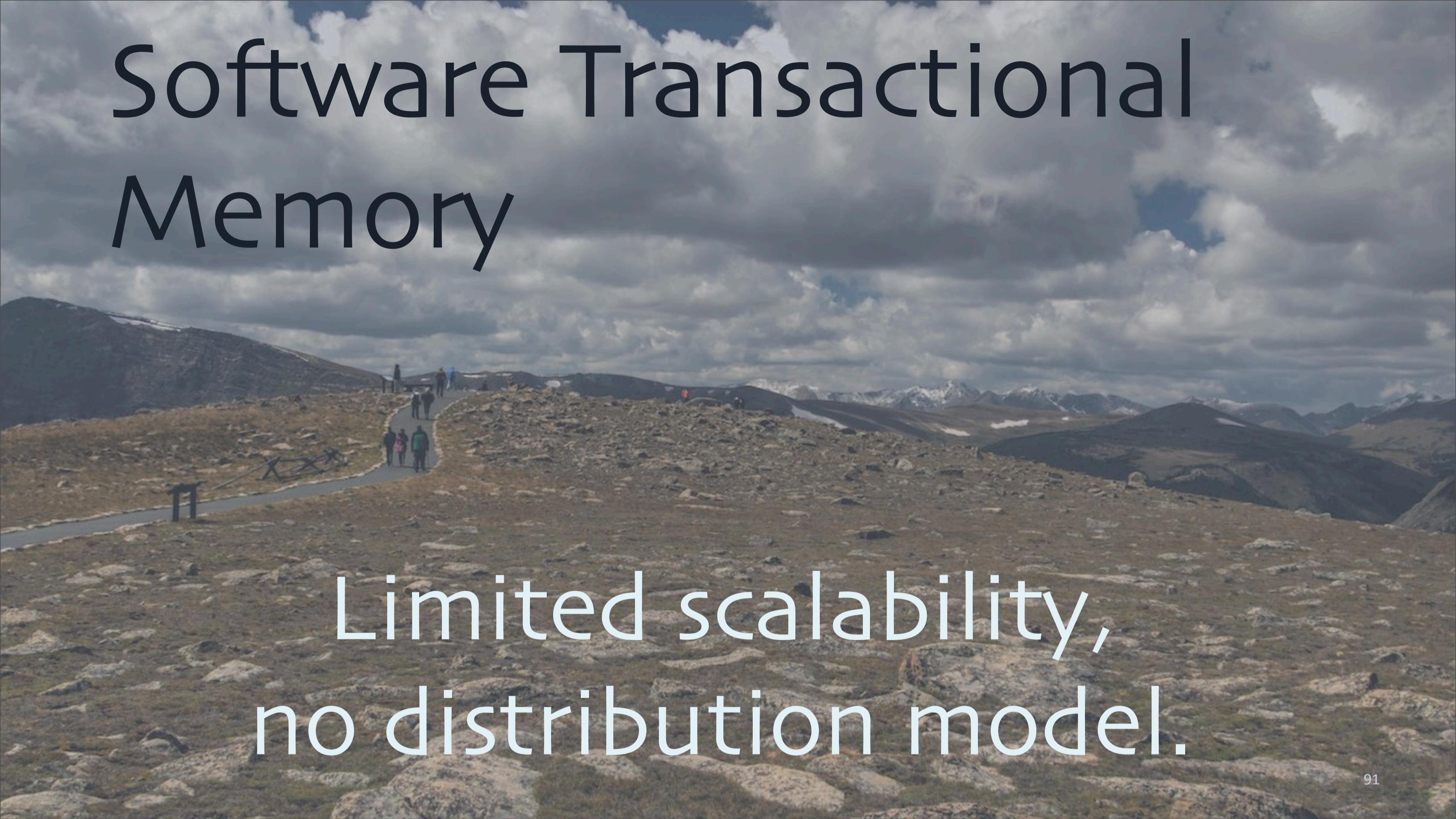
89

Software Transactional Memory

Principled local state
mutation through transactions.

90

Software Transactional Memory



Limited scalability,
no distribution model.

91

Tuesday, May 19, 15

A very powerful tool for avoiding local locks and unprincipled mutation, but not a tool that scales to the global challenges we're discussing here.

Event Loops

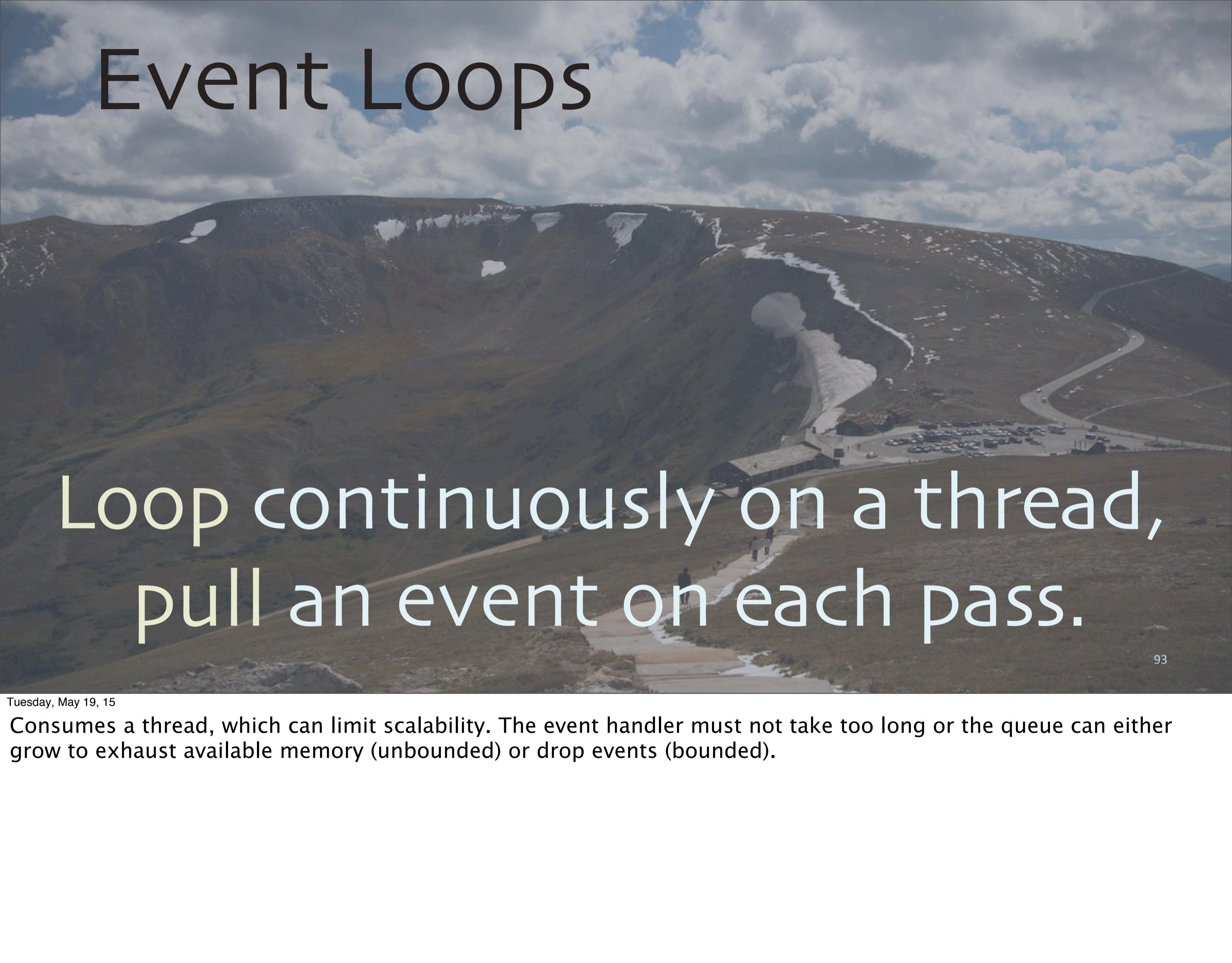


92

Tuesday, May 19, 15

The standard technique for message/event driven programming. Usually pull based, for something that loops continuously pulling events off a queue, or push based with callbacks.

Event Loops

A scenic view of a winding road through a mountainous landscape. The mountains are covered in patches of snow and green vegetation. A small town or parking area is visible at the base of the mountains. The sky is filled with white clouds.

Loop continuously on a thread,
pull an event on each pass.

93

Tuesday, May 19, 15

Consumes a thread, which can limit scalability. The event handler must not take too long or the queue can either grow to exhaust available memory (unbounded) or drop events (bounded).

Event Loops

A scenic view of a winding road through a mountainous landscape. The mountains are covered in patches of snow and green vegetation. A road curves through the valley, leading towards a parking lot and some buildings at the base of the mountain. The sky is filled with white and grey clouds.

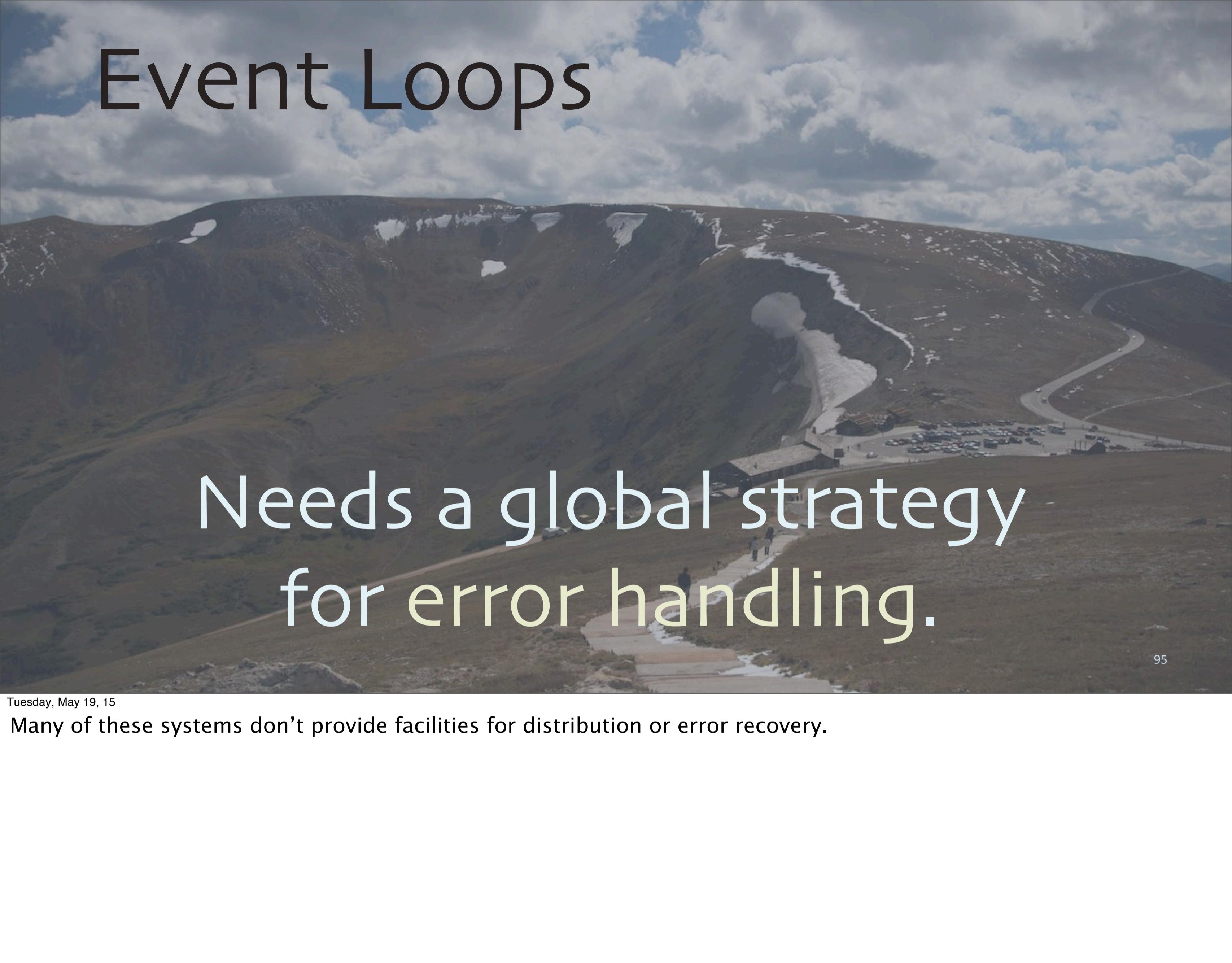
Callbacks invoked when
an event is pushed to it.

94

Tuesday, May 19, 15

Can use threads more efficiently, but callback hell is sometimes a problem.

Event Loops

A scenic view of a winding road through a mountainous landscape. The mountains are covered in patches of snow and green vegetation. A road curves through the valley, leading towards a parking lot and a building. The sky is filled with white and grey clouds.

Needs a global strategy
for error handling.

95

Tuesday, May 19, 15

Many of these systems don't provide facilities for distribution or error recovery.

Communicating Sequential Processes

96

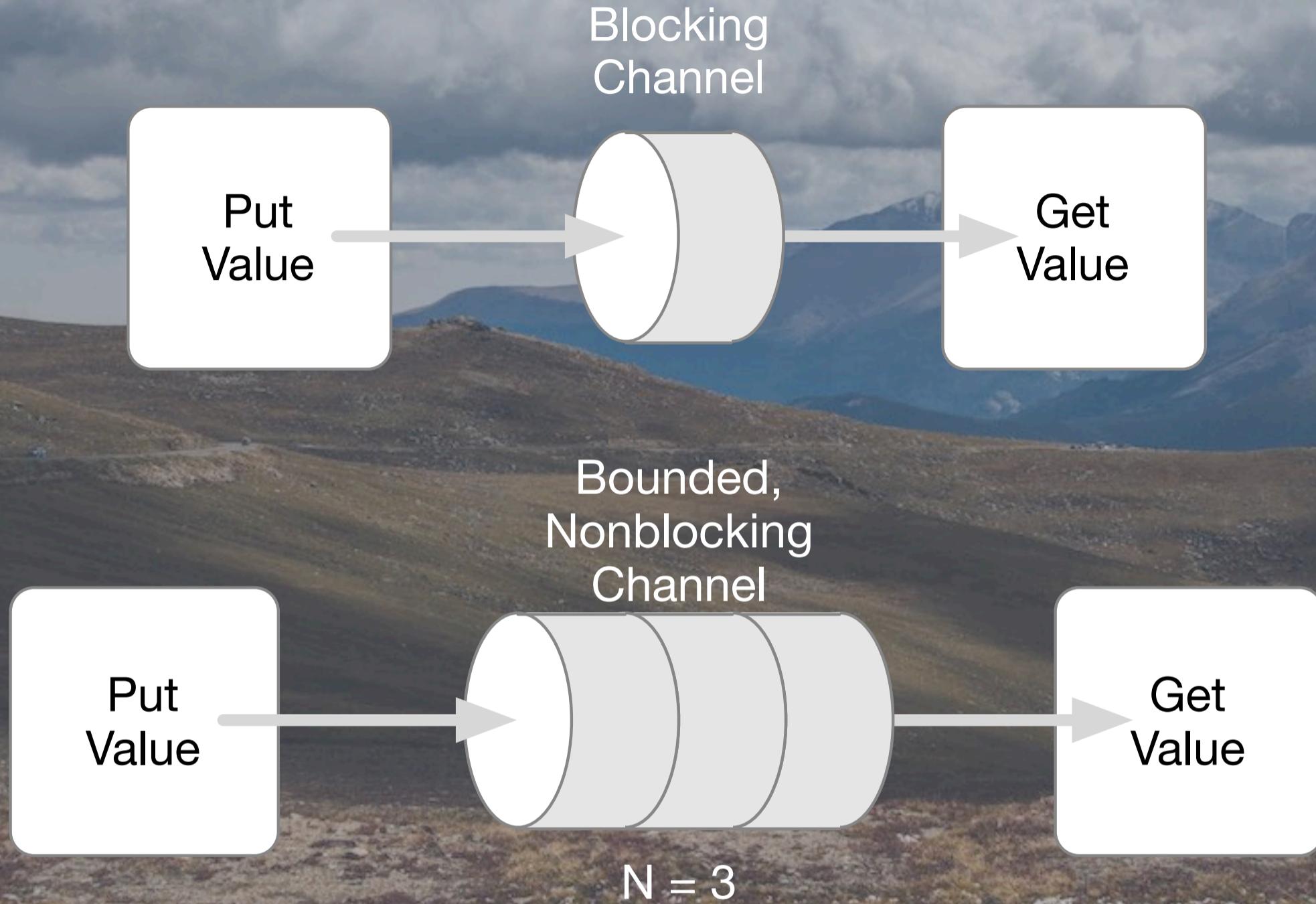
Tuesday, May 19, 15

Communicating Sequential Processes – The first mathematical model of distributed computing. It has evolved somewhat and it's still popular in Clojure and Go, for example.



Decouple sender and receiver
via a channel.
Can be sync. or async.
Not typically distributed...

CSP



98

Tuesday, May 19, 15

Schematic view of simple CSP interactions. If the channel queue has one slot, then it's blocking; the “putter” wait for a “getter” to be on the other side. If there are >1 slots, the putter won’t block unless all the slots are full. See my talk on error handling in reactive systems where I discuss CSP in more detail. (I’ll discuss CSP vs. Actors in more depth in my other talk.)

Futures



99

Tuesday, May 19, 15

Fill more or less the same niche as CSP. That is, most Futures and CSP systems cover the same scope of concurrency control, which is somewhat fine-grained as opposed to strategic.

Futures

Run logic asynchronously.

Apply map, flatMap, etc.
to the results.

100

Tuesday, May 19, 15

Like using the UNIX shell to fire a process in the background, but you either define callbacks to handle the success or failure (a more procedure-oriented approach) or use functional operations like map, etc. to process the results on success.

Futures

Run logic asynchronously.

Or use callbacks
and error handlers.

101

Tuesday, May 19, 15

Like using the UNIX shell to fire a process in the background, but you either define callbacks to handle the success or failure (a more procedure-oriented approach) or use functional operations like map, etc. to process the results on success.

Actor Systems

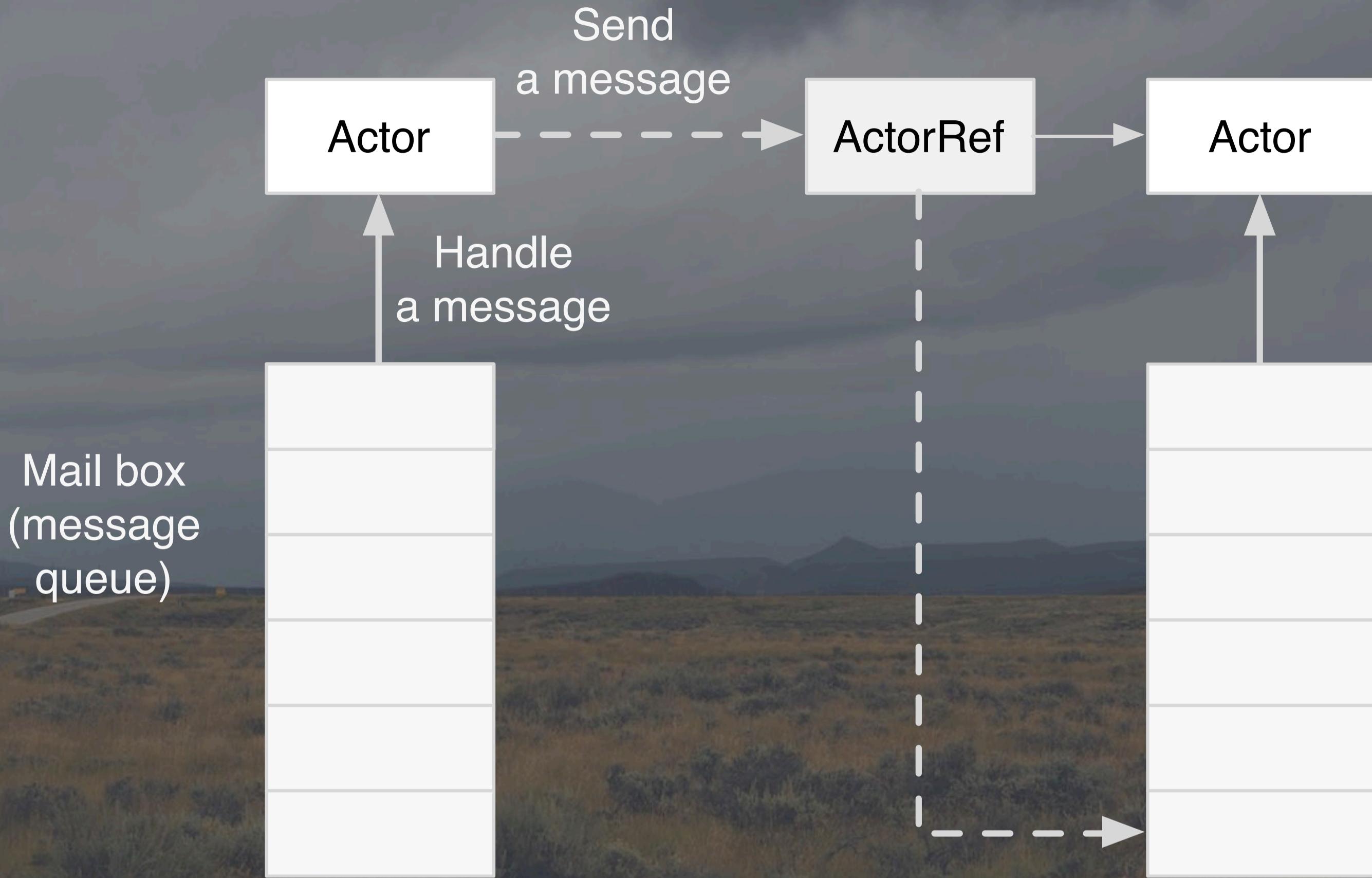
Let it Crash!

102

Tuesday, May 19, 15

We briefly visited this before.

Rather than attempt to recover from errors inside the domain logic (e.g., elaborate exception handling), allow services to fail, but with failure detection and reconstruction of those services, plus failover to other replicas.



103

Tuesday, May 19, 15

Actors are similar to objects in Smalltalk and similar systems; autonomous agents with defined boundaries that communicate through message passing. Actors, though process each message in a threadsafe way, so they are great for concurrency. (This diagram illustrates the Akka implementation – <http://akka.io>)

Actor Systems

Pioneered by Hewitt, et al. 1973.
Made popular by Erlang,
which introduced Supervision.

104

Tuesday, May 19, 15

Erlang is a simple language with actor semantics baked in. It has been used to create extremely reliable telecom switches, databases (e.g., Riak), and other services (e.g., GitHub).

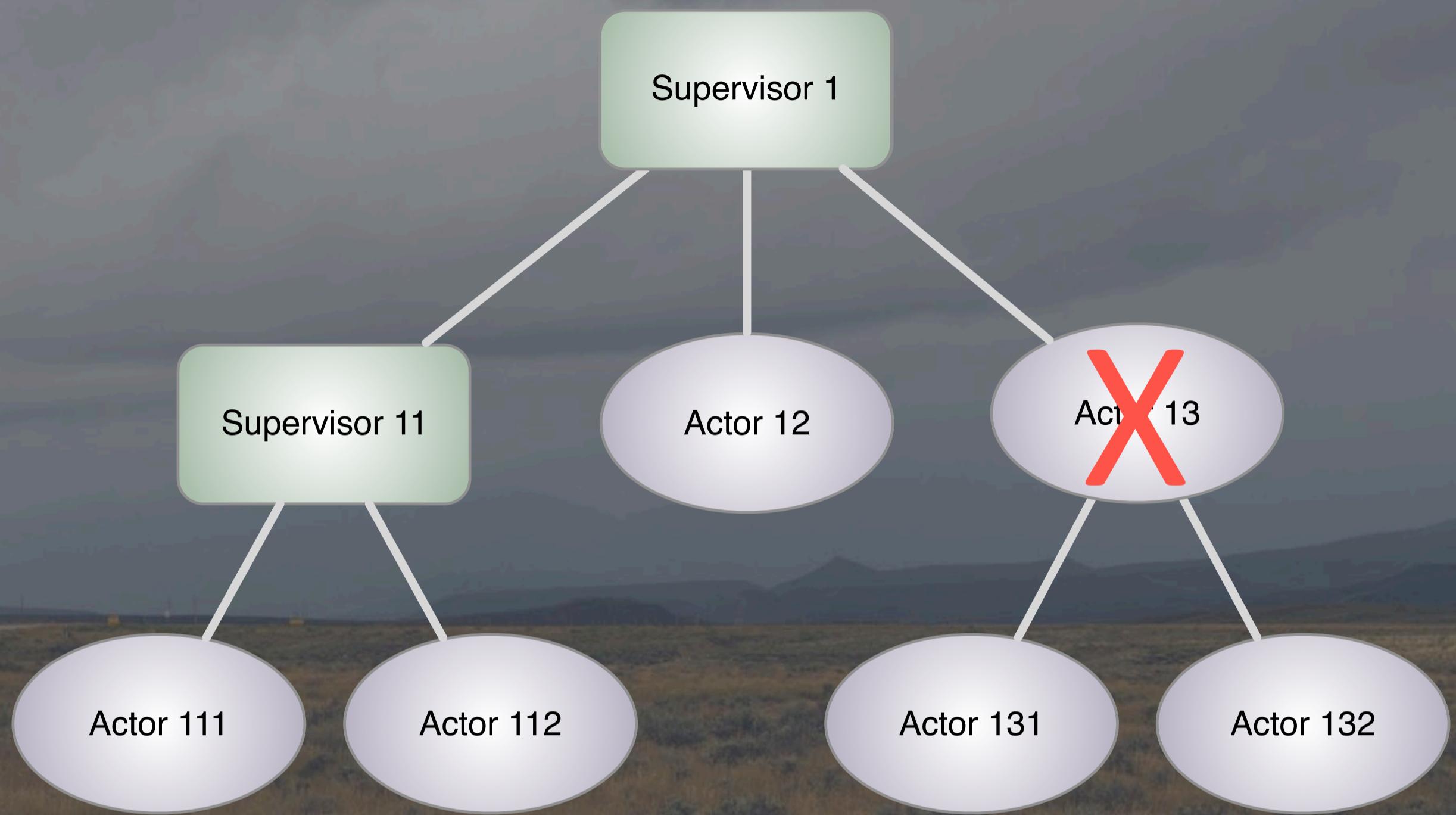
Actor Systems

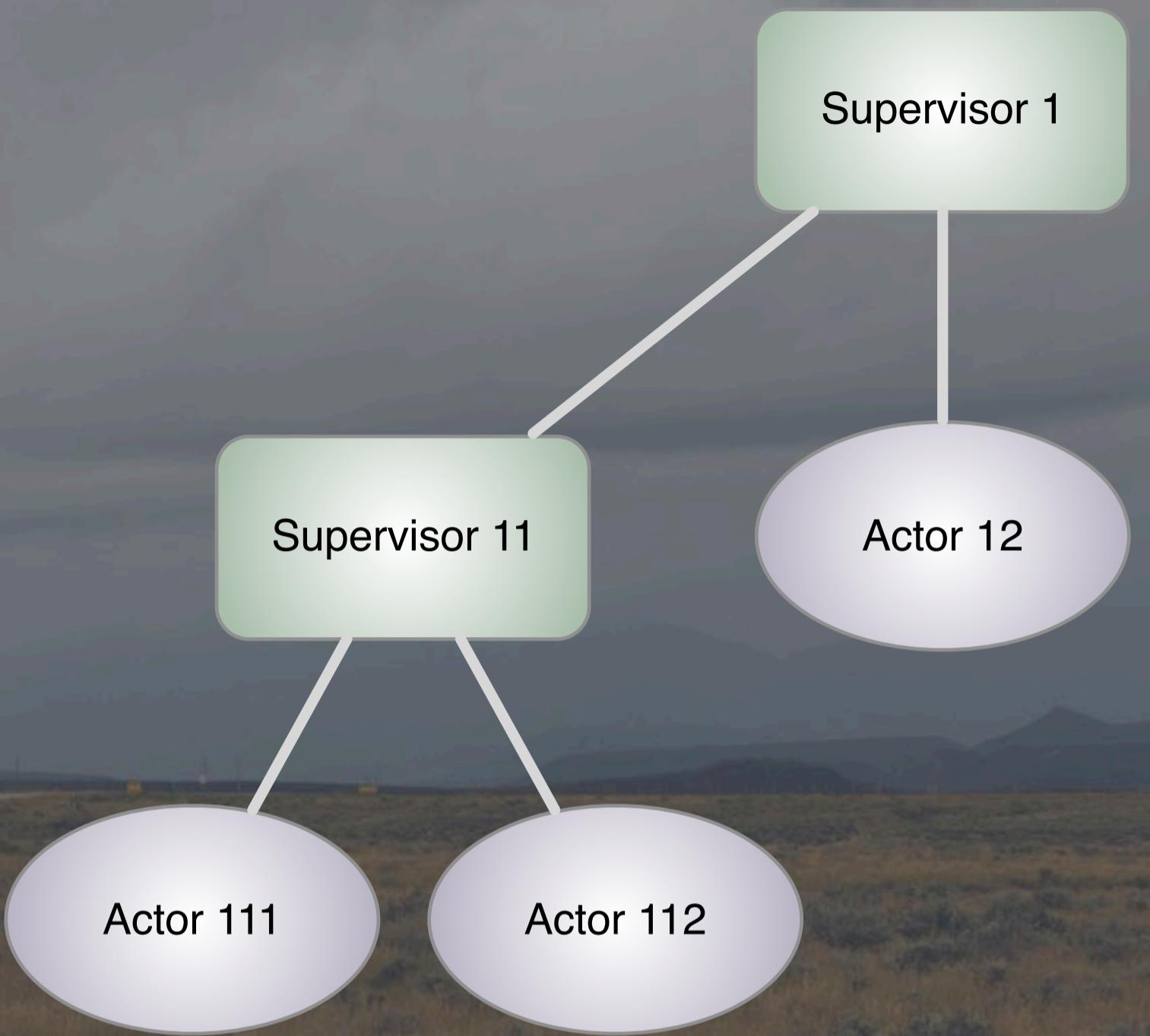
Distribution is a
natural extension.

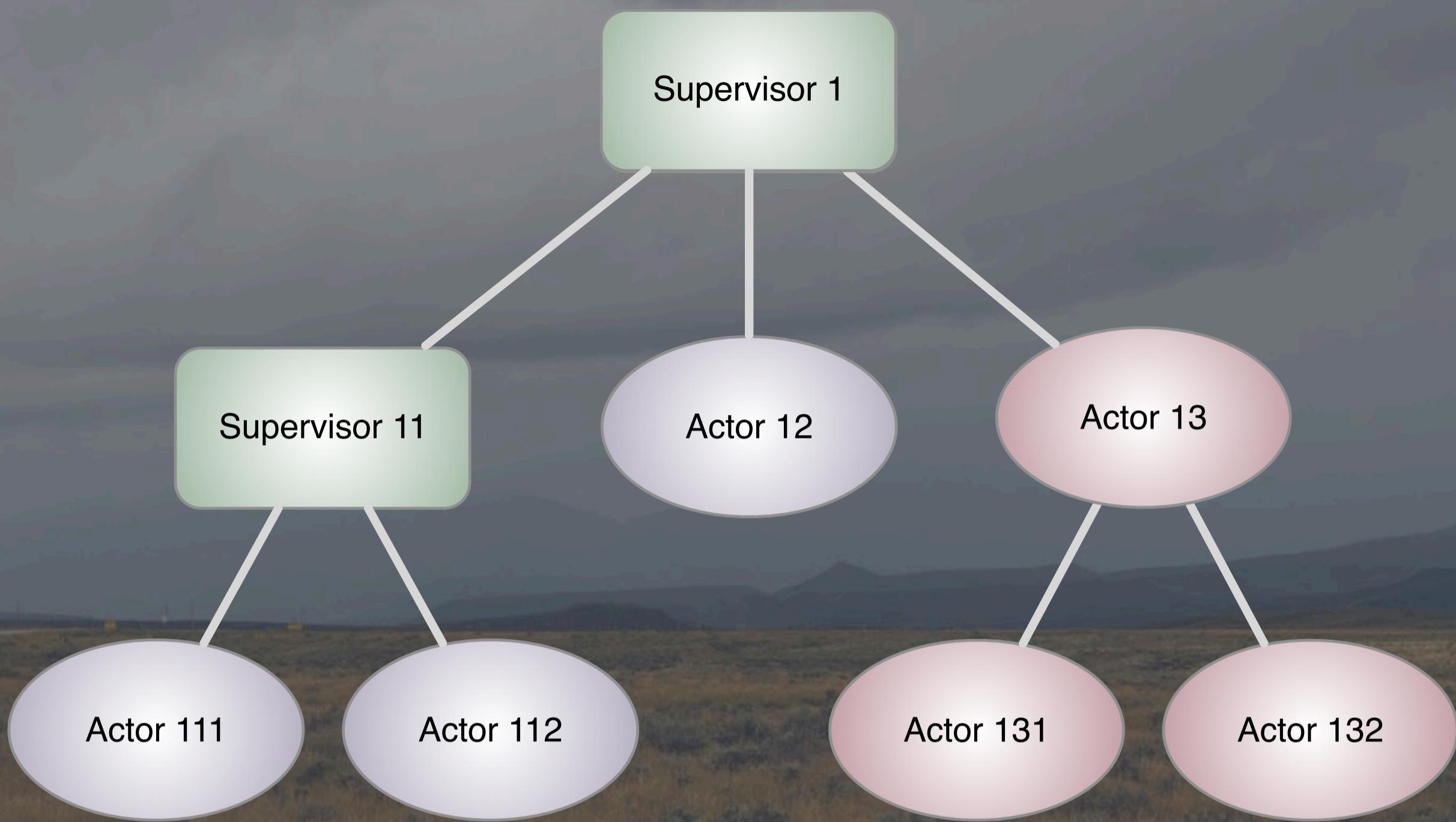
105

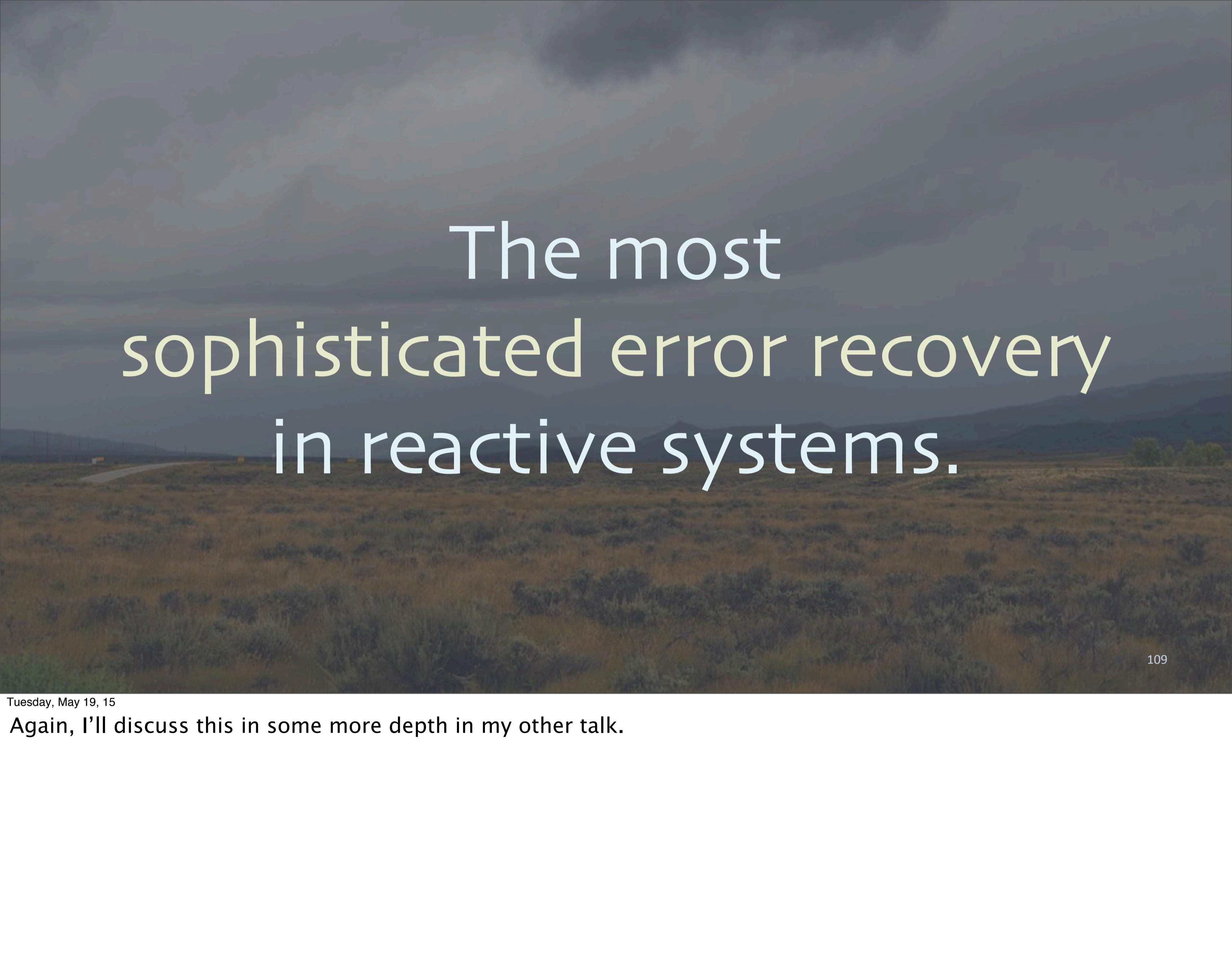
Tuesday, May 19, 15

Erlang is a simple language with actor semantics baked in. It has been used to create extremely reliable telecom switches, databases (e.g., Riak), and other services (e.g., GitHub).







A landscape photograph showing a dry, grassy field in the foreground, leading to a range of mountains in the background under a cloudy sky.

The most
sophisticated error recovery
in reactive systems.

109

Tuesday, May 19, 15

Again, I'll discuss this in some more depth in my other talk.

A landscape photograph showing a dry, grassy field in the foreground, leading to a range of mountains in the background under a clear sky.

Clean separation
of normal processing
from recovery.

110

A landscape photograph showing a dry, grassy field in the foreground, leading to a range of mountains in the background under a clear sky.

State mutation “firewalls”.
Supports location transparency.

Reactive Extensions (Rx)

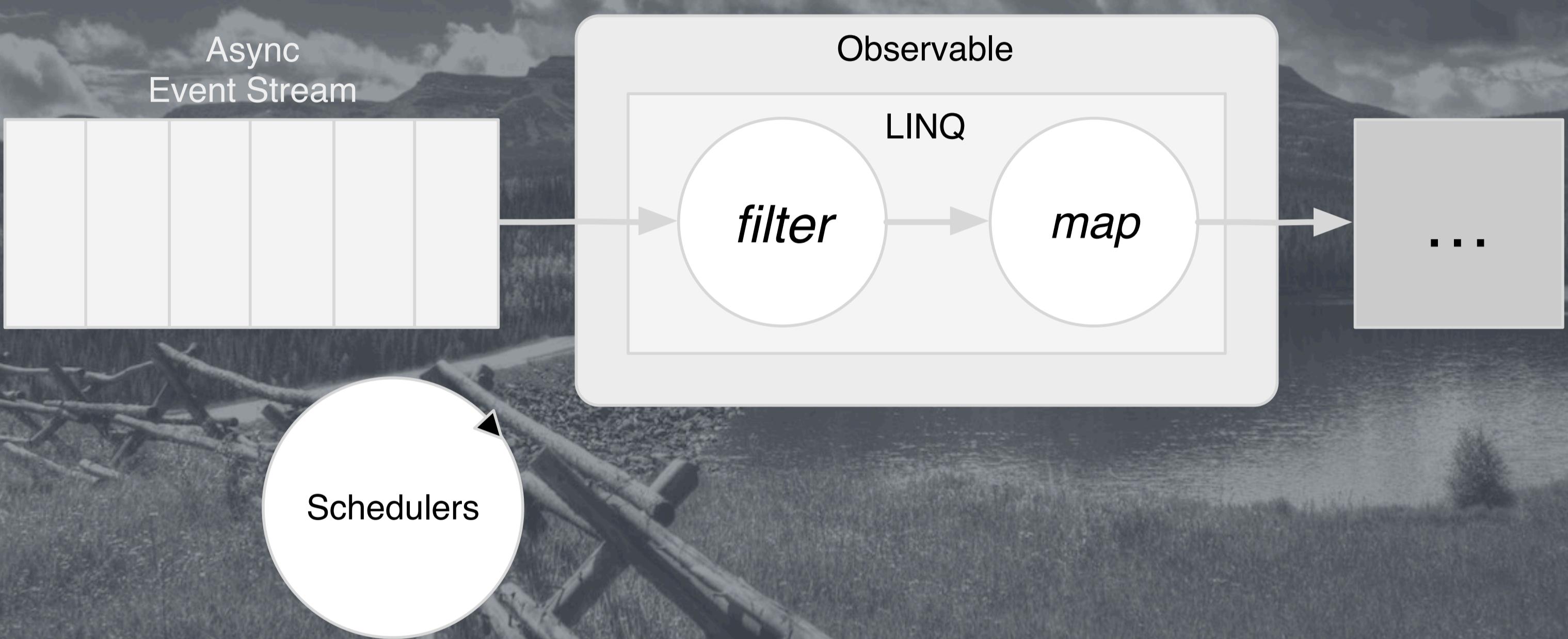


112

Tuesday, May 19, 15

Pioneered by Erik Meijer for .NET. Now ported to several languages, including RxJava (Netflix) and React (JS – Facebook).

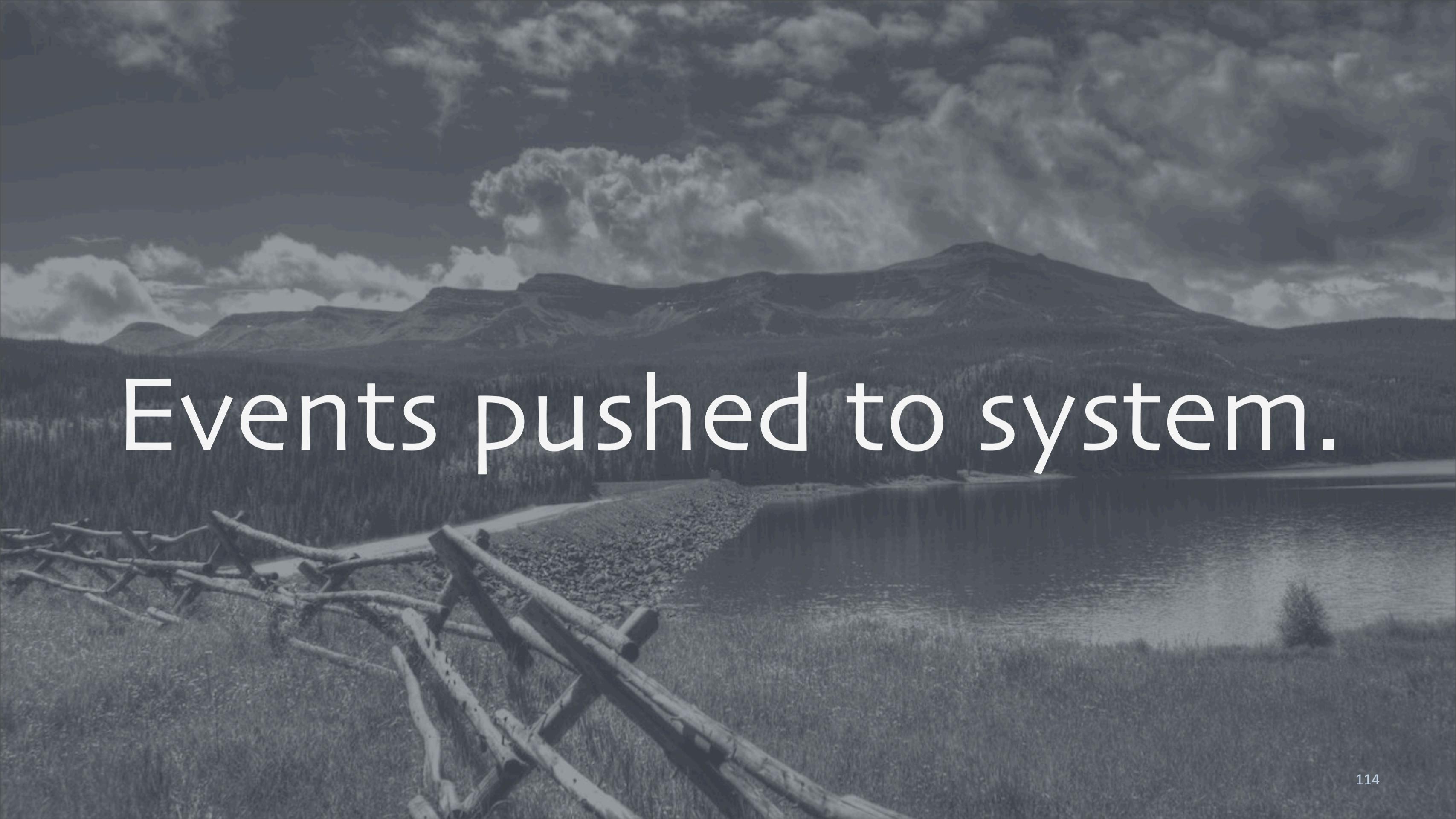
LINQ Rx



113

Tuesday, May 19, 15

Events are observed (an extension of the observer pattern). Operations like filtering and mapping are provided to work with the stream through LINQ (Language Integrated Query), which uses SQL-like expressions. The Schedulers are used to trigger processing.



Events pushed to system.

114

Tuesday, May 19, 15

It's essentially a push model.

A black and white photograph of a landscape featuring a range of mountains in the background under a cloudy sky. In the foreground, there's a grassy area with some fallen logs or branches. A dirt path or road leads towards the mountains.

FP/SQL-like
query semantics to
manipulate events.

Rx

Combines Iterator and
Observer into Observable.
Stream oriented.

116

Tuesday, May 19, 15

I didn't mention this before.

Rx

Need to add your own
fault-tolerance model.

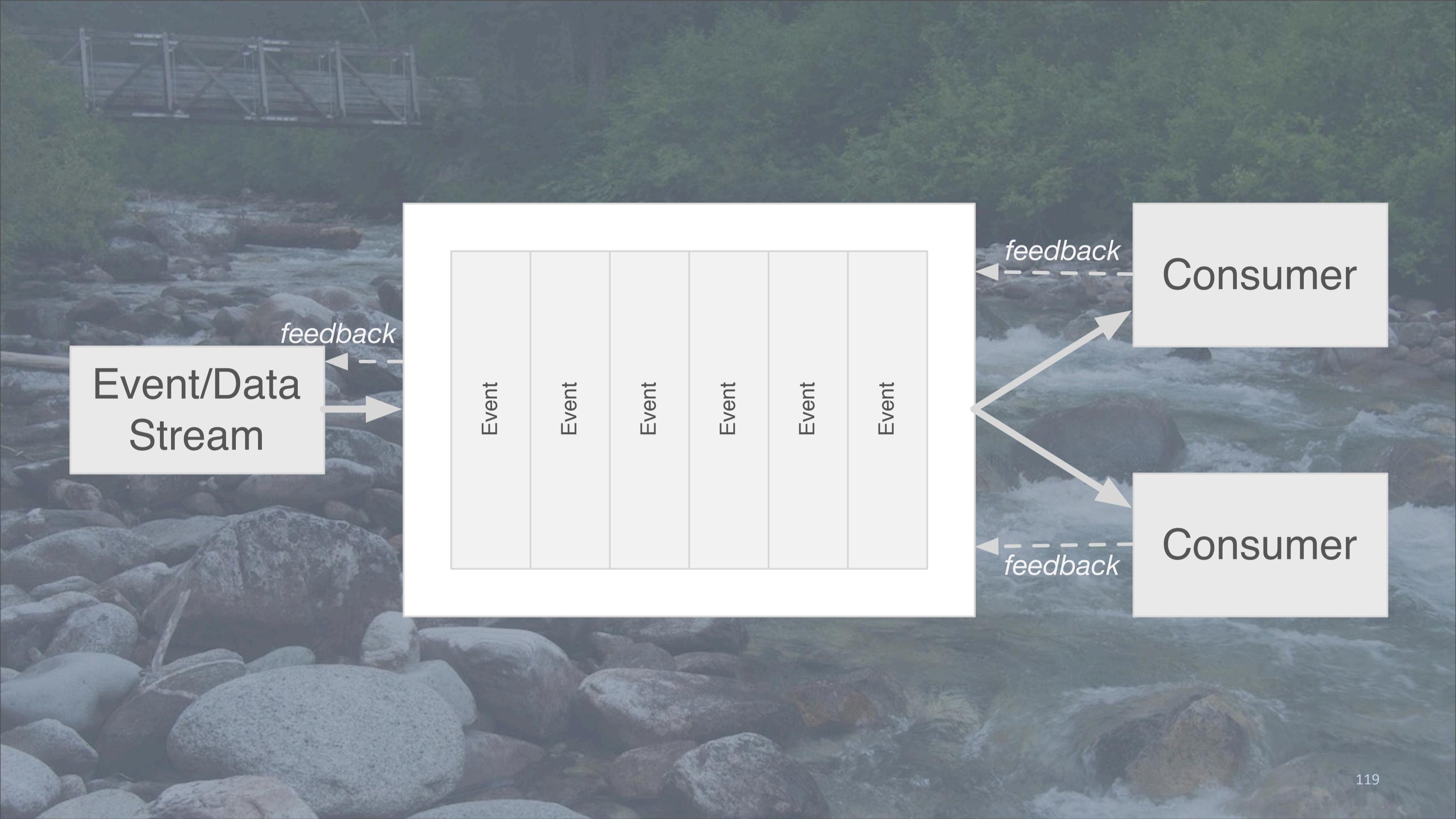
Reactive Streams

reactive-streams.org

118

Tuesday, May 19, 15

A standard with many implementations for streaming systems with truly “reactive” behavior.
photo: Bridge Creek, North Cascades National Park, Washington State (not Colorado ;)



119

Tuesday, May 19, 15

A stream of events from some upstream producer to one or more downstream consumers. Typically, queues are used for buffering, since for asynchrony the production and consumption can't be in lock step, that is synchronized! But what happens if the queue is unbounded? Or bounded? That's where the feedback comes in.

Reactive Streams

Unbounded queues
eventually exhaust
the heap.

120

Tuesday, May 19, 15

Any rare, low-probability event will eventually happen for a system that's big enough or runs long enough. Any unbounded queue will eventually grow to consume all memory.

Reactive Streams

Bounded queues cause
blocking or arbitrary
dropping of events.

121

Tuesday, May 19, 15

Bounded queues avoid heap exhaustion, but force arbitrary dropping of events or blocking.

Reactive Streams

Solution: Back pressure
where the producer and
consumer negotiate.

122

Tuesday, May 19, 15

Back pressure, where the producer and consumer negotiate the flow rate dynamically, is the only way to avoid these scenarios.

Reactive Streams

Back pressure
allows strategic
management of
event flows.

123

Tuesday, May 19, 15

With a system using backpressure for all flows, it's possible to add global flow control and also strategically decide when you must drop events or take other action.

Reactive Streams

Logical evolution of Rx.
More focus on possibly-infinite streams of data.

124

Tuesday, May 19, 15

Like Rx, RS uses functional transformations to manipulate the data. It puts slightly more emphasis on the idea of streams (possibly infinite) rather than an event loop.



Reactive Streams

Akka Streams: a higher-level abstraction implemented with Akka Actors.

125

Tuesday, May 19, 15

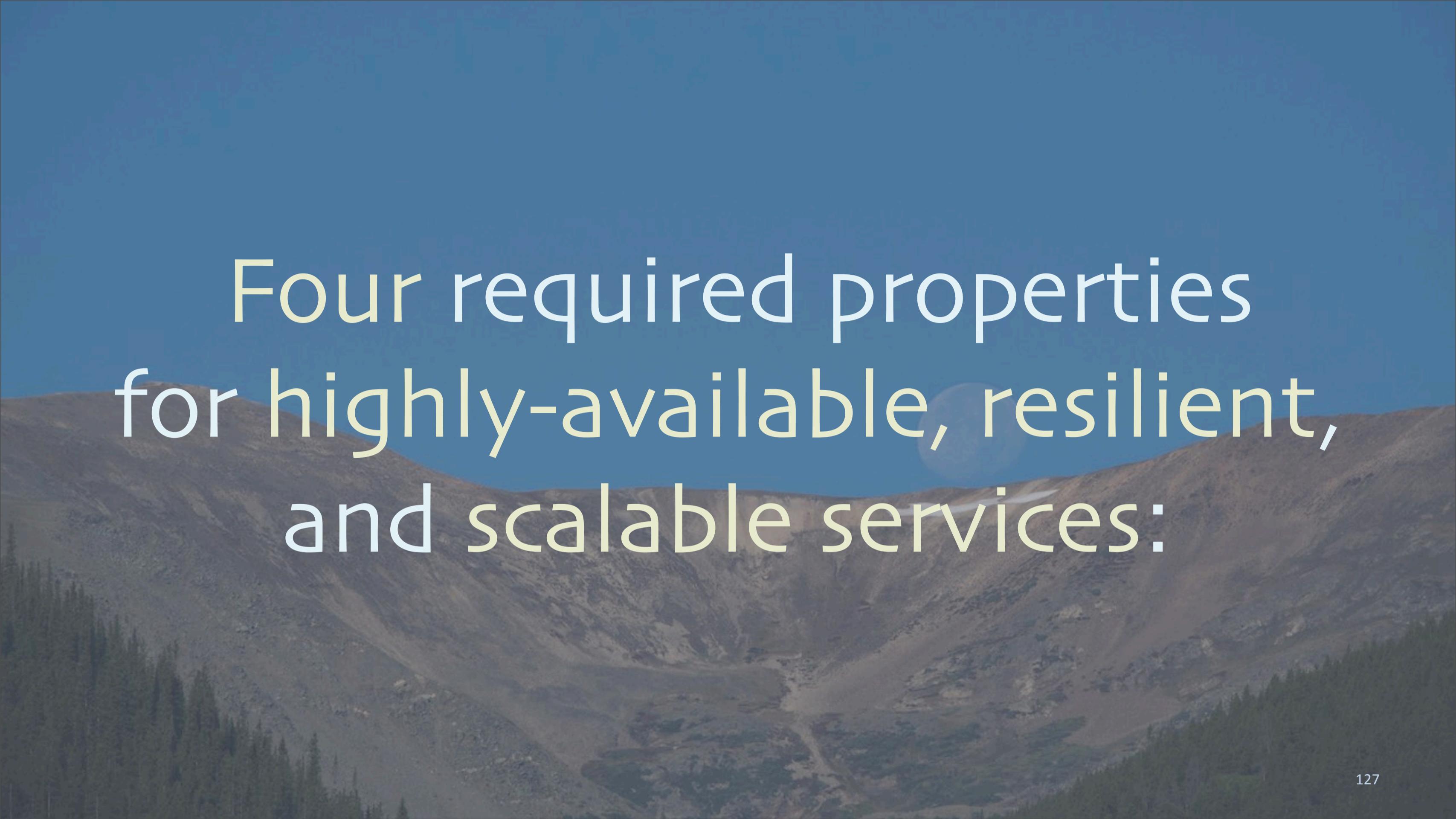
We're realizing that Actors are a low-level primitive and Actor systems can become unwieldy. Typesafe thinks that higher-level abstractions, like reactive streams, implemented on low-level concurrency systems, like Actors, will be the way to go for most future systems. Other implementations of RS include RxJava.

Recap

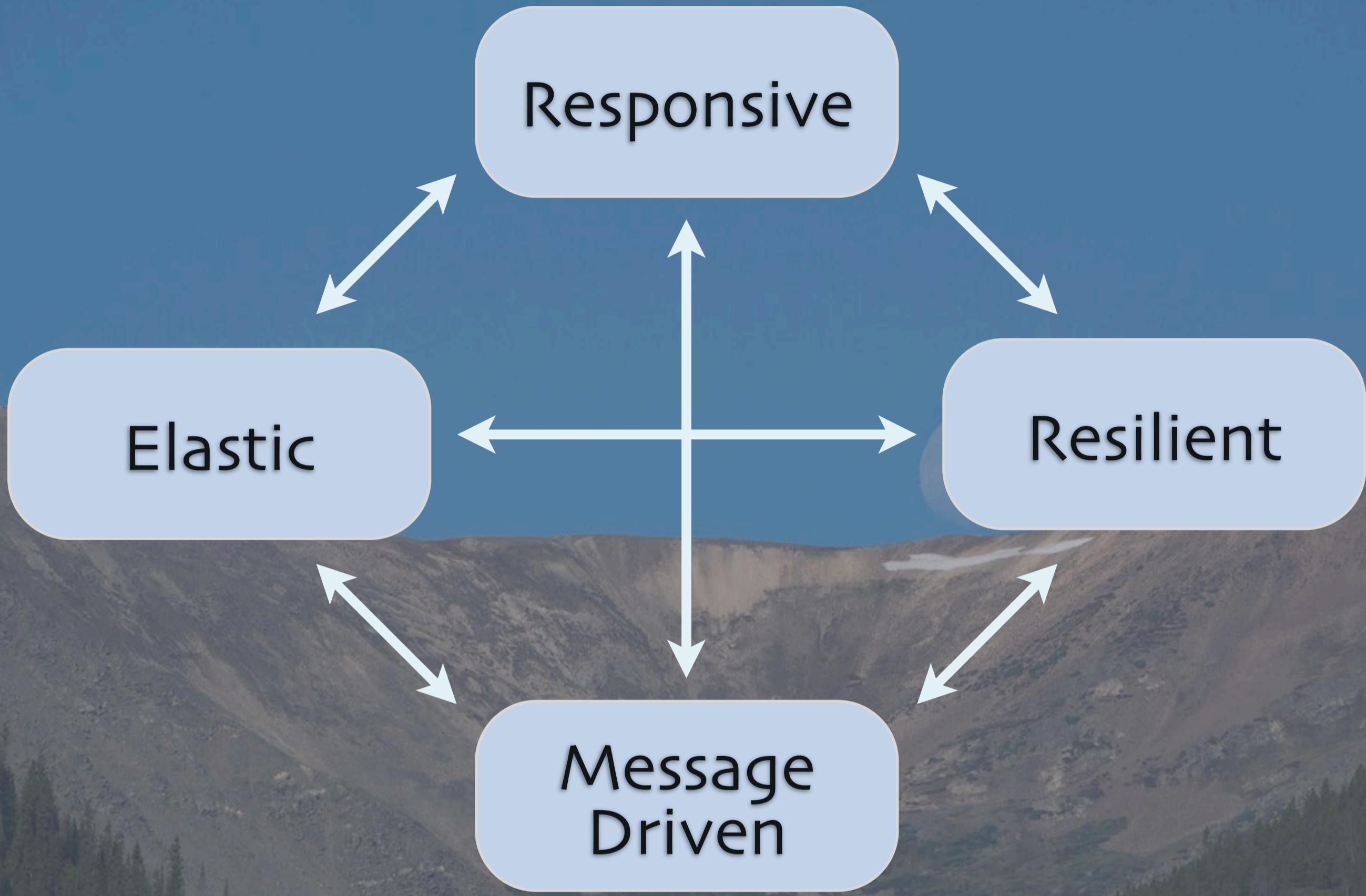


126

Tuesday, May 19, 15



Four required properties
for highly-available, resilient,
and scalable services:





Typesafe

Reactive

<http://typesafe.com/reactive-big-data>
dean.wampler@typesafe.com

©Typesafe 2014-2015, All Rights Reserved

Tuesday, May 19, 15

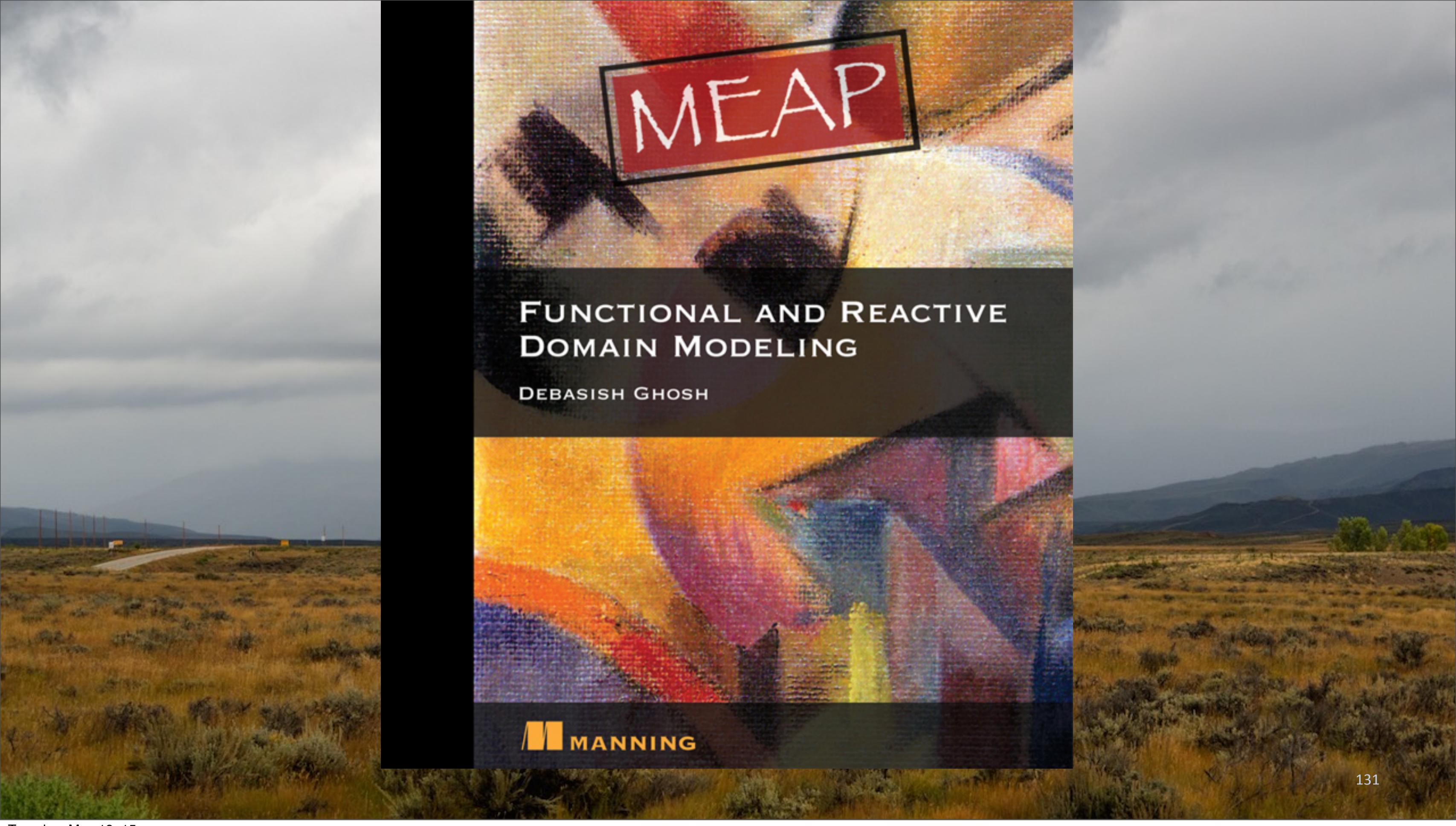
My primary role at Typesafe is addressing the Big Data market. We're now rolling out commercial support for Spark in non-Hadoop environments and we have other projects in the works. Talk to me if you're interested in what we're doing.

References

130

Tuesday, May 19, 15

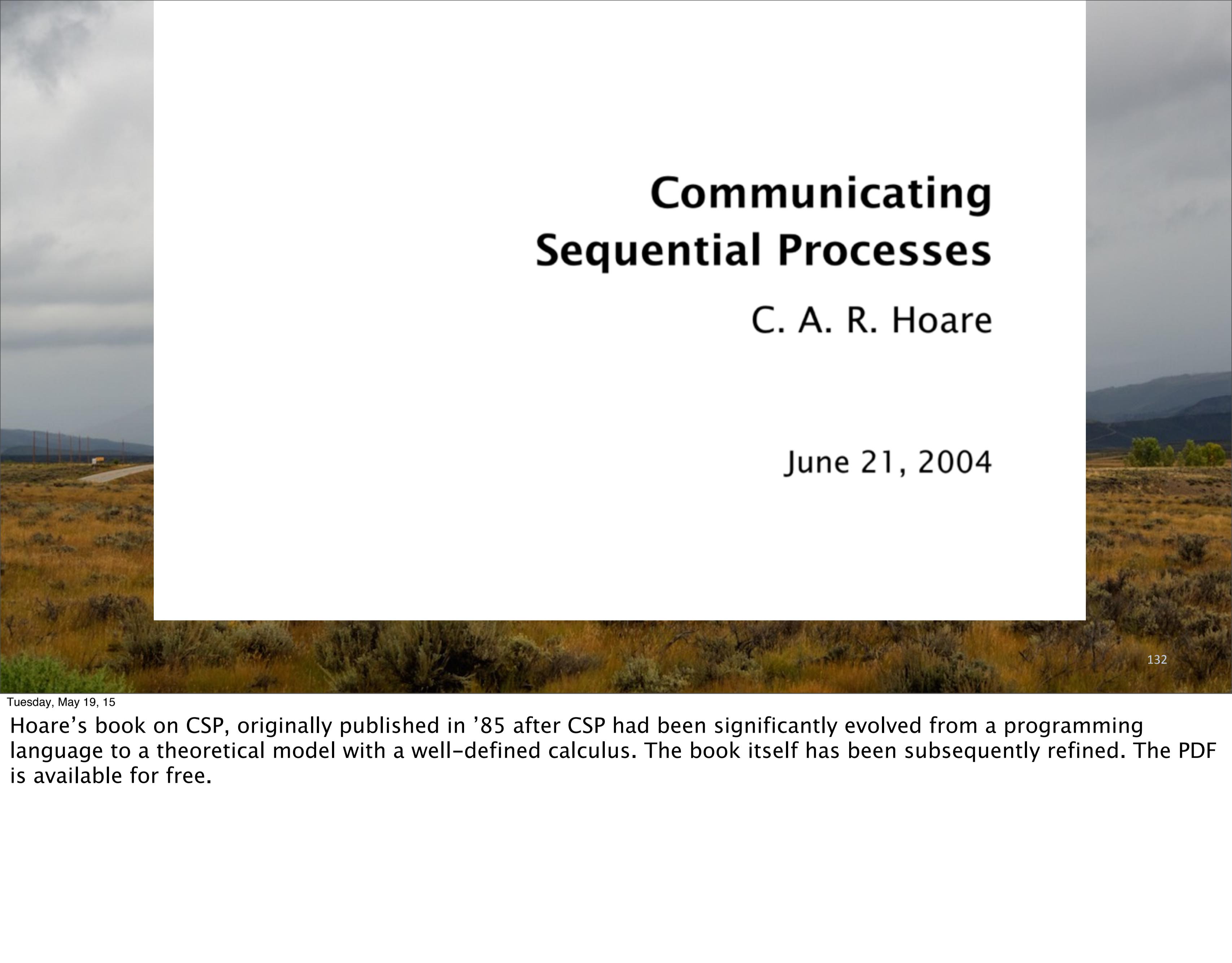
See also links earlier in the presentation.



131

Tuesday, May 19, 15

Lots of interesting practical ideas for combining functional programming and reactive approaches to class Domain-Driven Design by Eric Evans.



Communicating Sequential Processes

C. A. R. Hoare

June 21, 2004

132

Tuesday, May 19, 15

Hoare's book on CSP, originally published in '85 after CSP had been significantly evolved from a programming language to a theoretical model with a well-defined calculus. The book itself has been subsequently refined. The PDF is available for free.

The Theory and Practice of Concurrency

A.W. Roscoe

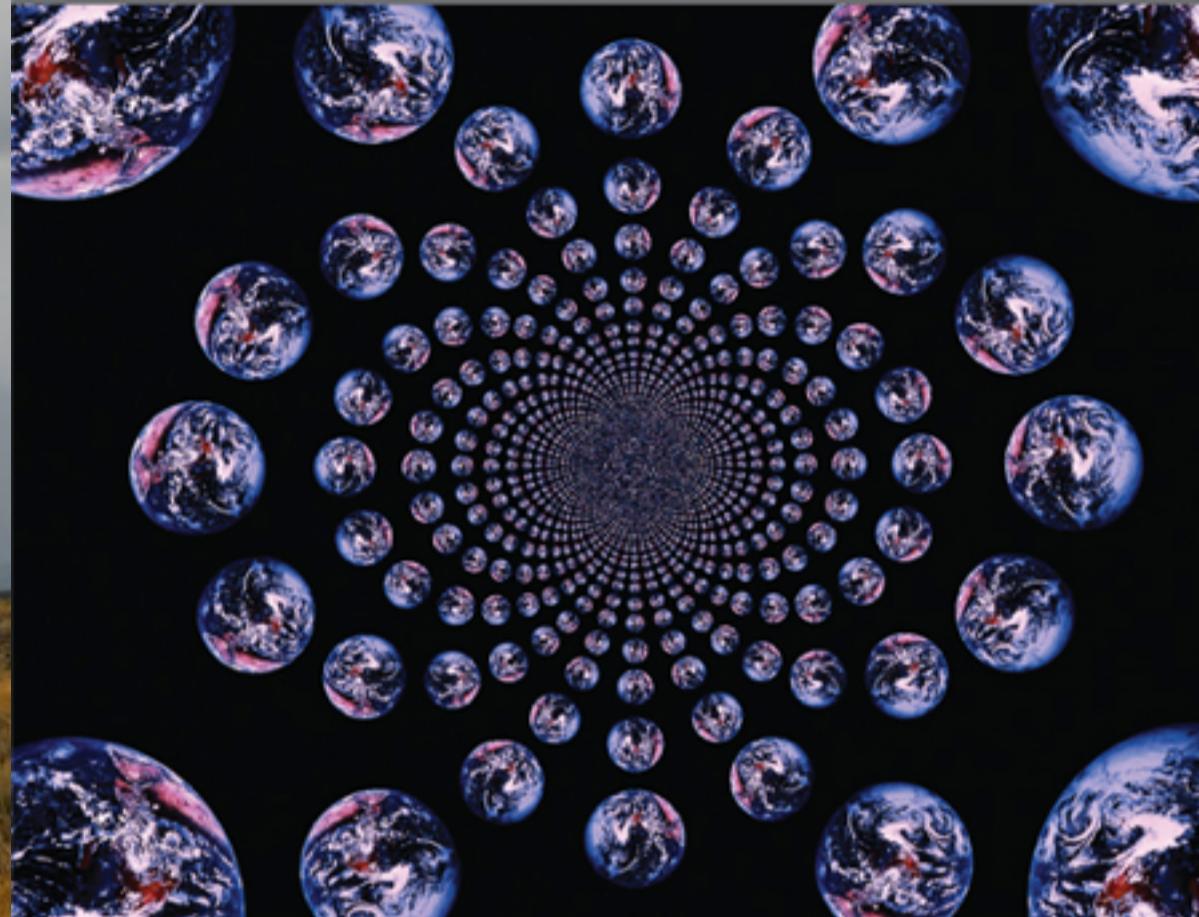
Published 1997, revised to 2000 and lightly revised to 2005.

The original version is in print in April 2005 with Prentice-Hall (Pearson).
This version is made available for personal reference only. This version is
copyright (©) Pearson and Bill Roscoe.

PROGRAMMING DISTRIBUTED COMPUTING SYSTEMS

A Foundational Approach

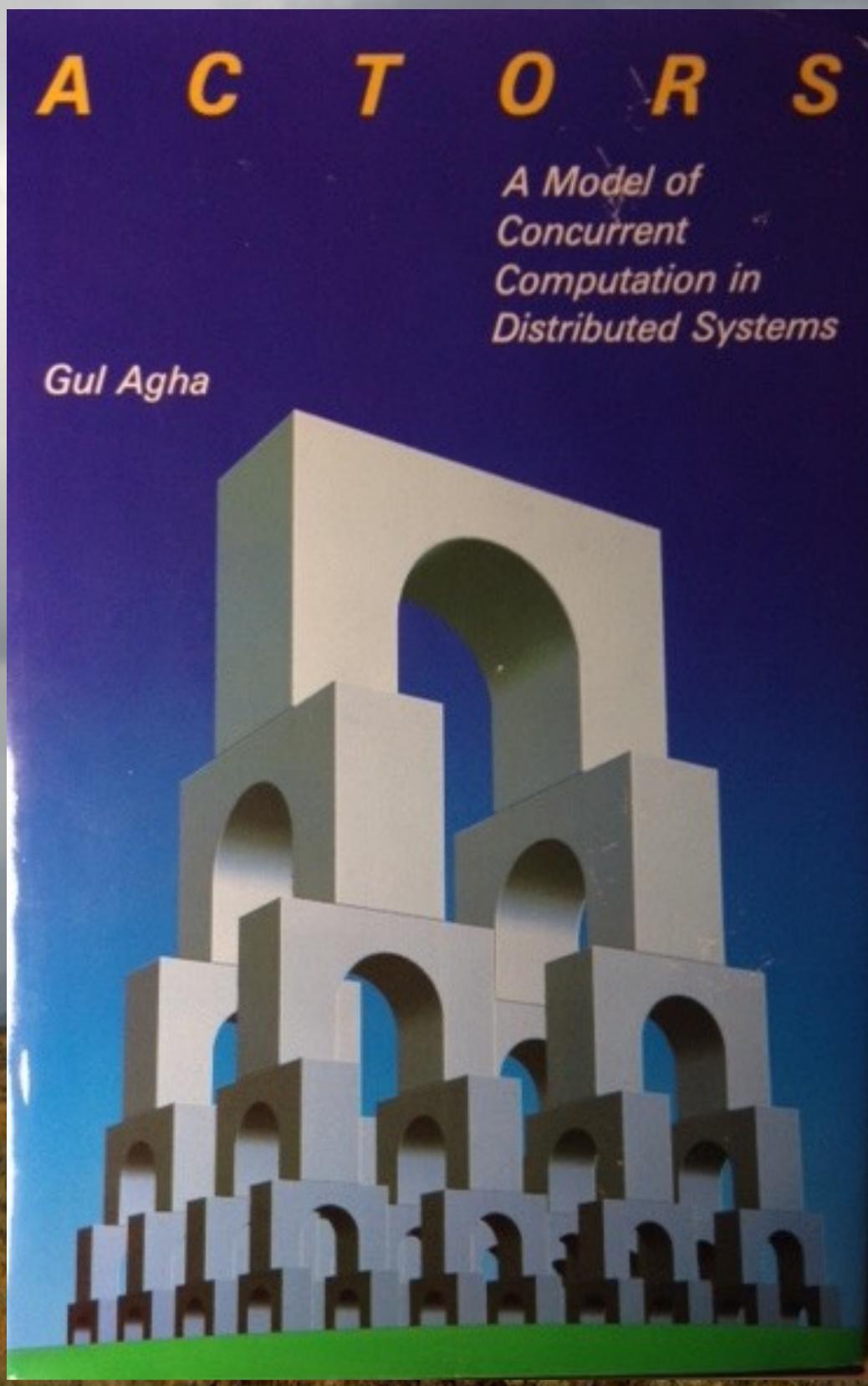
CARLOS A. VARELA



134

Tuesday, May 19, 15

A survey of theoretical models of distributed computing, starting with a summary of lambda calculus, then discussing the pi, join, and ambient calculi. Also discusses the actor model. The treatment is somewhat dry and could use more discussion of real-world implementations of these ideas, such as the Actor model in Erlang and Akka.



135

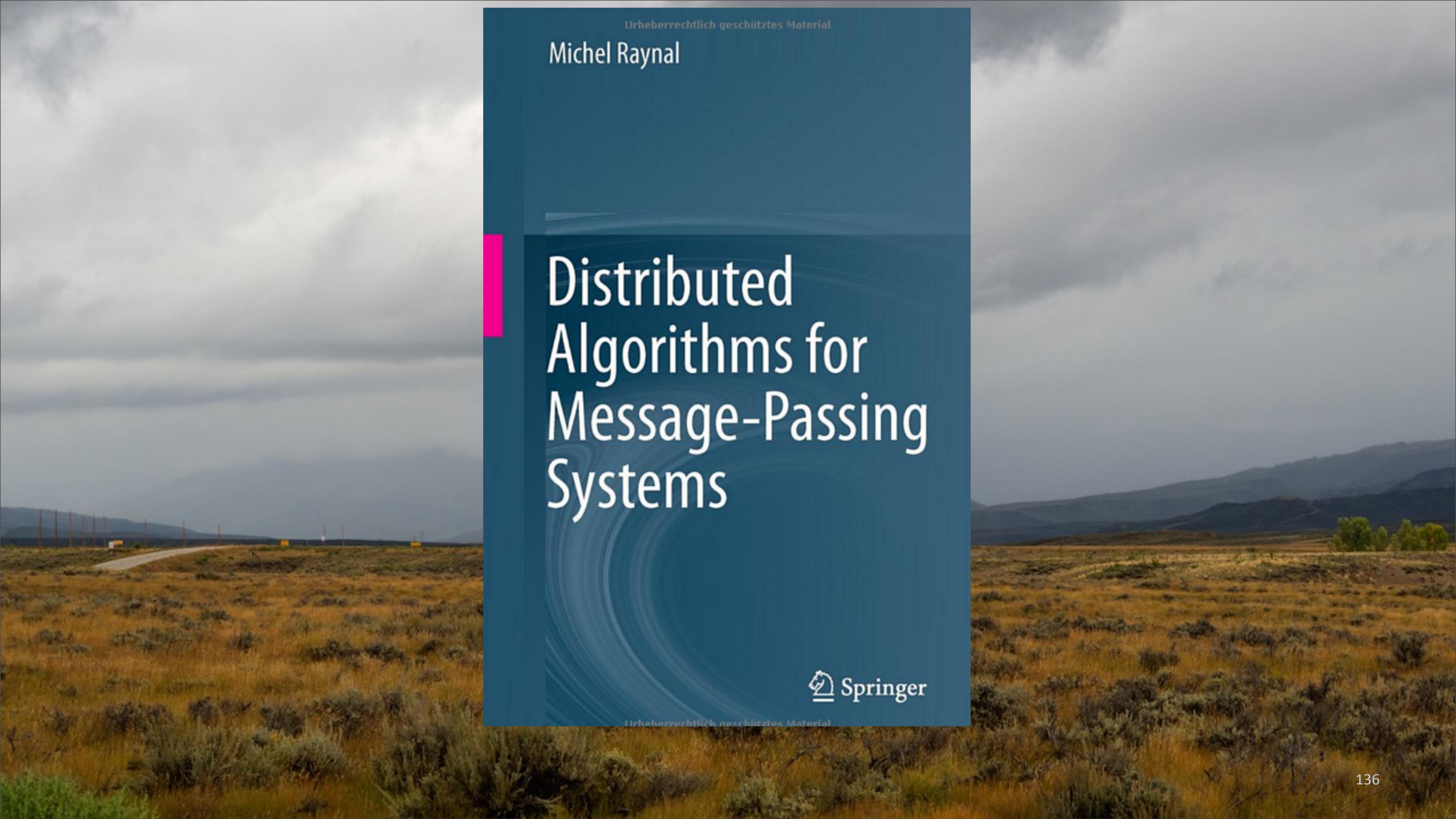
Tuesday, May 19, 15

Gul Agha was a grad student at MIT during the 80s and worked on the actor model with Hewitt and others. This book is based on his dissertation.

It doesn't discuss error handling, actor supervision, etc. as these concepts .

His thesis, <http://dspace.mit.edu/handle/1721.1/6952>, the basis for his book,<http://mitpress.mit.edu/books/actors>

See also Paper for a survey course with Rajesh Karmani, <http://www.cs.ucla.edu/~palsberg/course/cs239/papers/karmani-agha.pdf>



136

Tuesday, May 19, 15

Survey of the classic graph traversal algorithms, algorithms for detecting failures in a cluster, leader election, etc.

DISTRIBUTED ALGORITHMS

AN INTUITIVE APPROACH



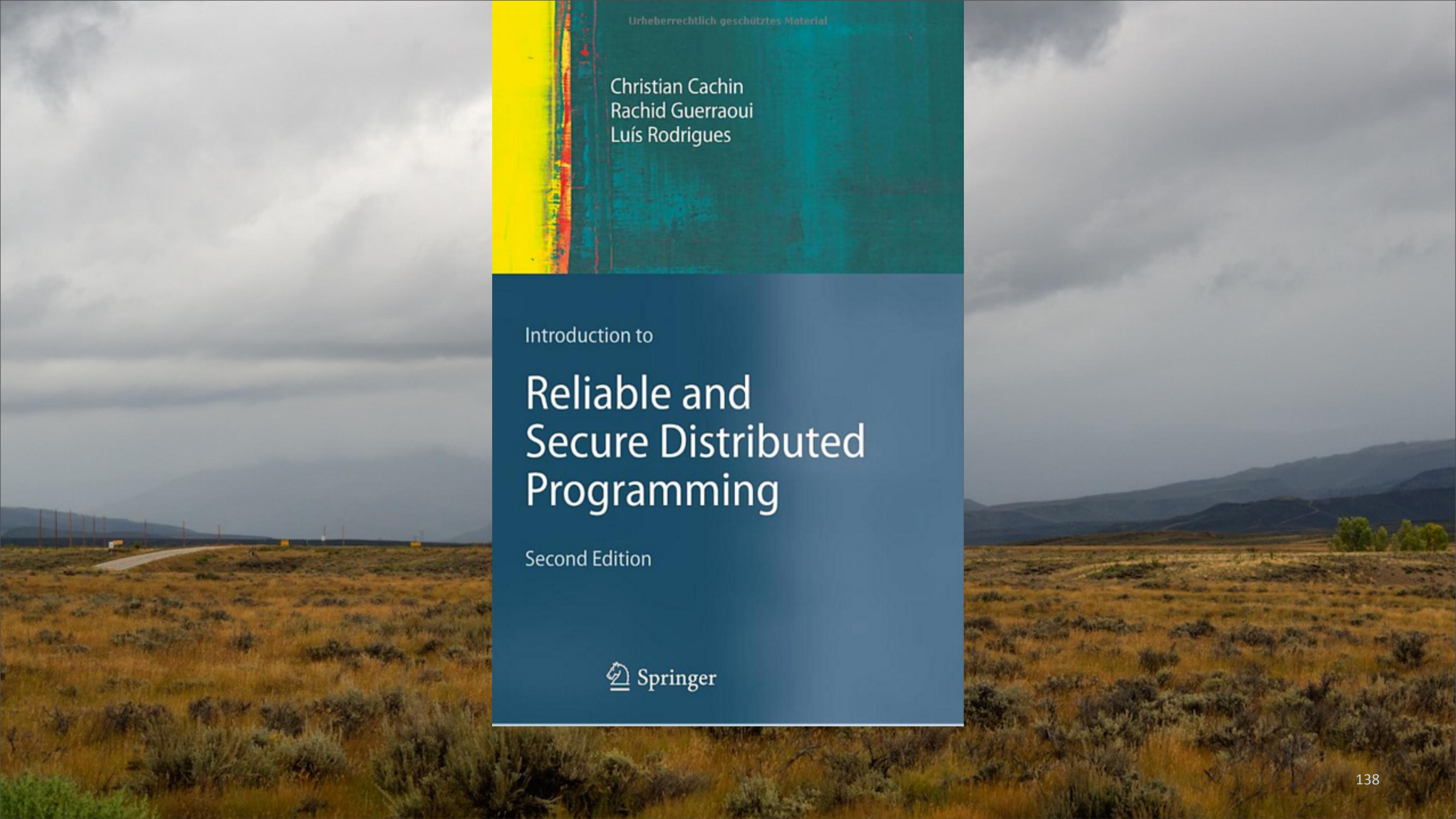
WAN FOKKINK

Copyrighted material

137

Tuesday, May 19, 15

A less comprehensive and formal, but more intuitive approach to fundamental algorithms.



138

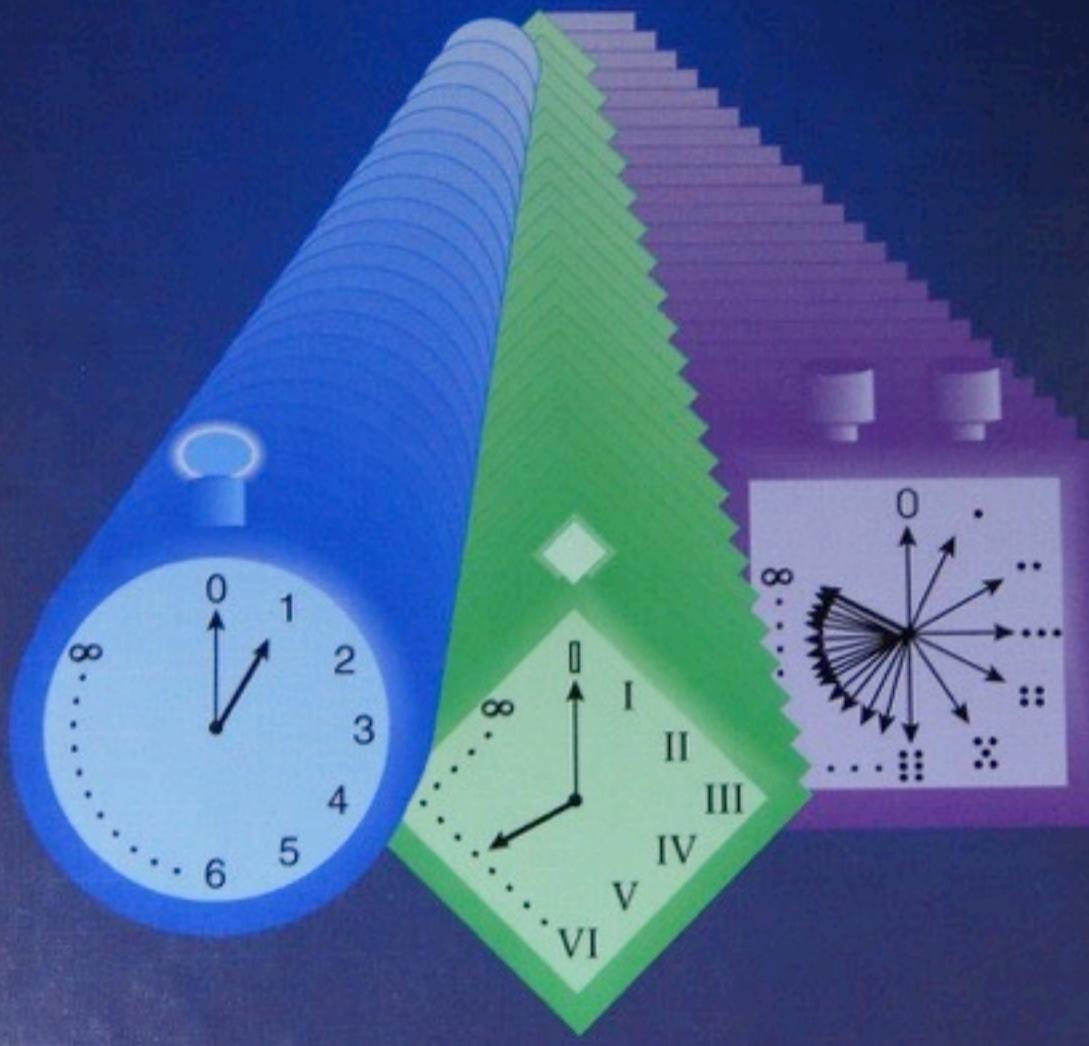
Tuesday, May 19, 15

Comprehensive and somewhat formal like Raynal's book, but more focused on modeling common failures in real systems.

Zohar Manna
Amir Pnueli

The Temporal Logic of Reactive and Concurrent Systems

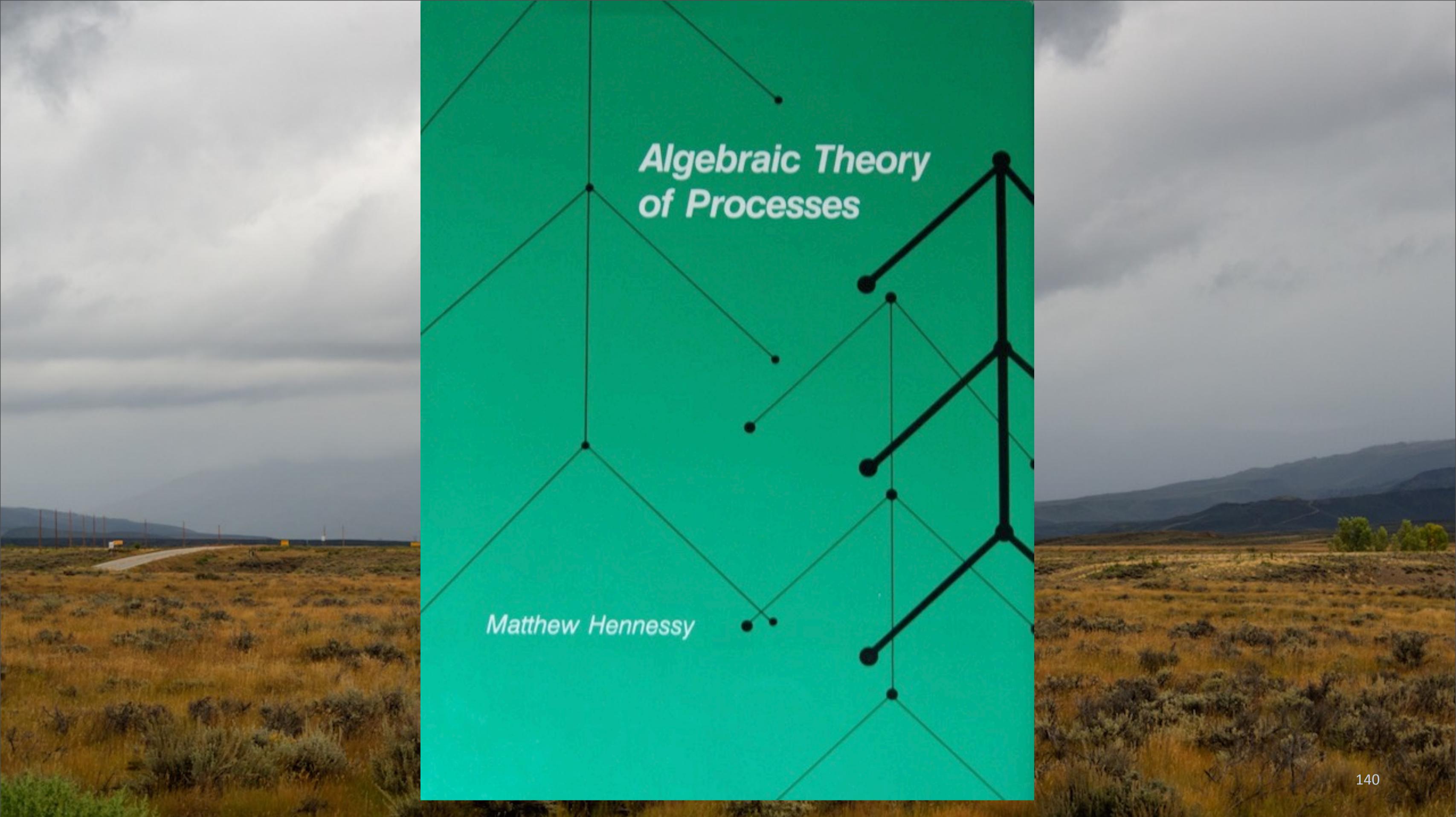
•Specification•



139

Tuesday, May 19, 15

1992: Yes, “Reactive” isn’t new ;) This book is lays out a theoretical model for specifying and proving “reactive” concurrent systems based on temporal logic. While its goal is to prevent logic errors, It doesn’t discuss handling failures from environmental or other external causes in great depth.



140

Tuesday, May 19, 15

1988: Another treatment of concurrency using algebra. It's not based on CSP, but it has similar constructs.

DISTRIBUTED COMPUTING *through* COMBINATORIAL TOPOLOGY



Copyrighted Material

Maurice Herlihy
Dmitry Kozlov
Sergio Rajsbaum

Tuesday, May 19, 15

A recent text that applies combinatorics (counting things) and topology (properties of geometric shapes) to the analysis of distributed systems. Aims to be pragmatic for real-world scenarios, like networks and other physical systems where failures are practical concerns.



Engineering a Safer World

Systems Thinking Applied
to Safety

Nancy G. Leveson

ENGINEERING SYSTEMS



142

Tuesday, May 19, 15

<http://mitpress.mit.edu/books/engineering-safer-world>

Farther afield, this book discusses safety concerns from a systems engineering perspective.