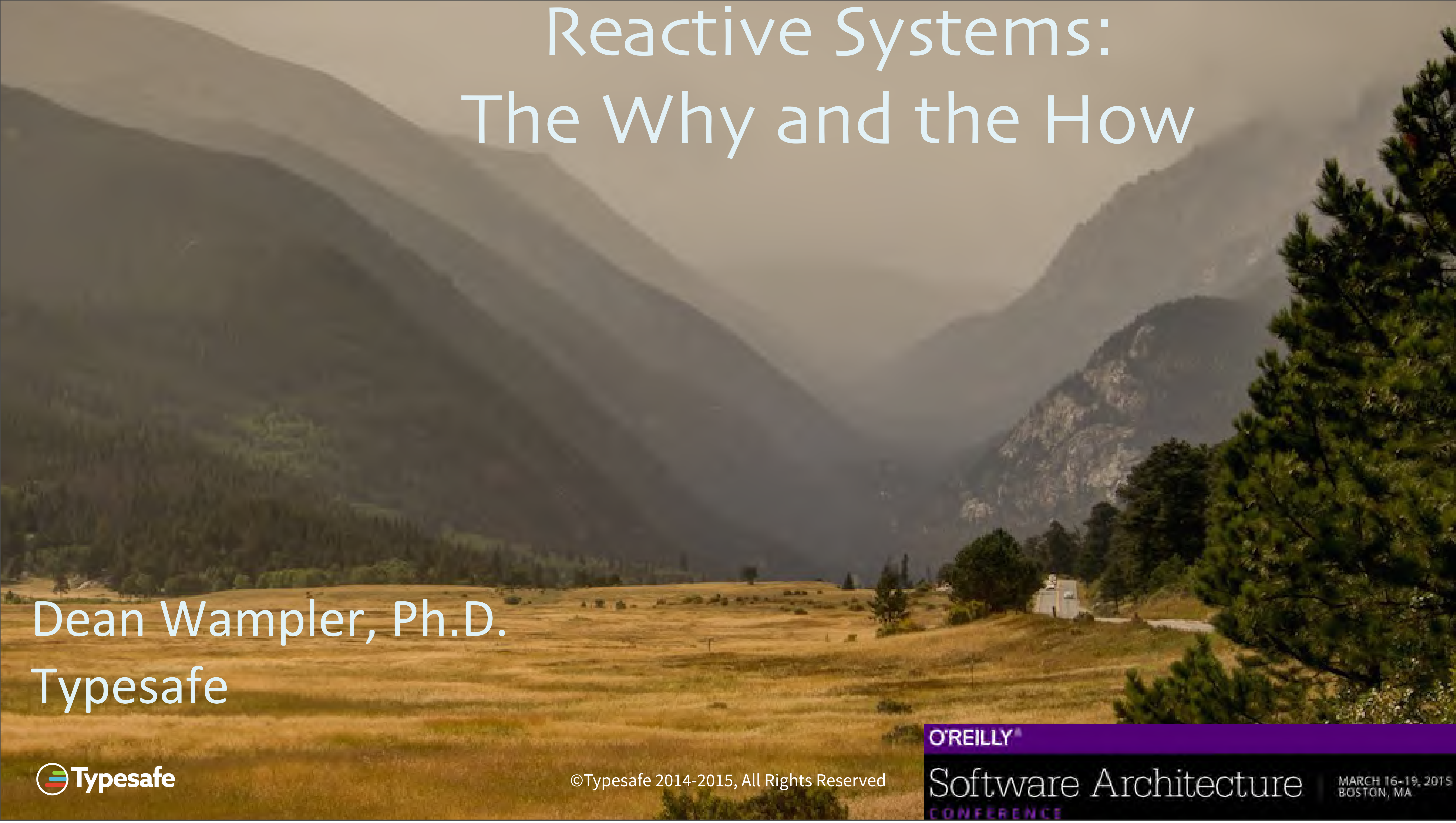
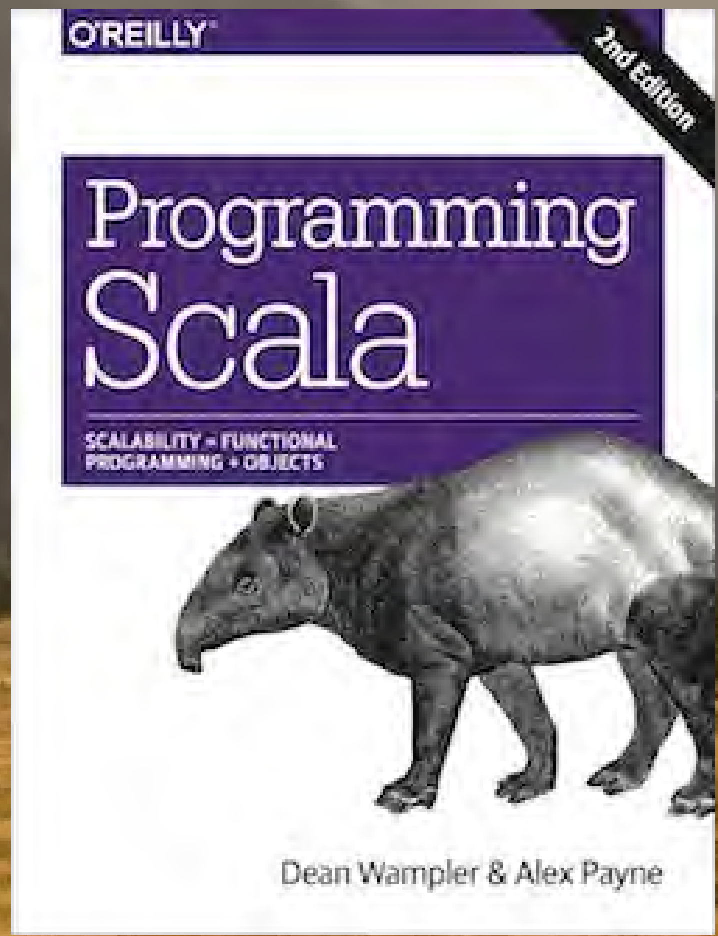
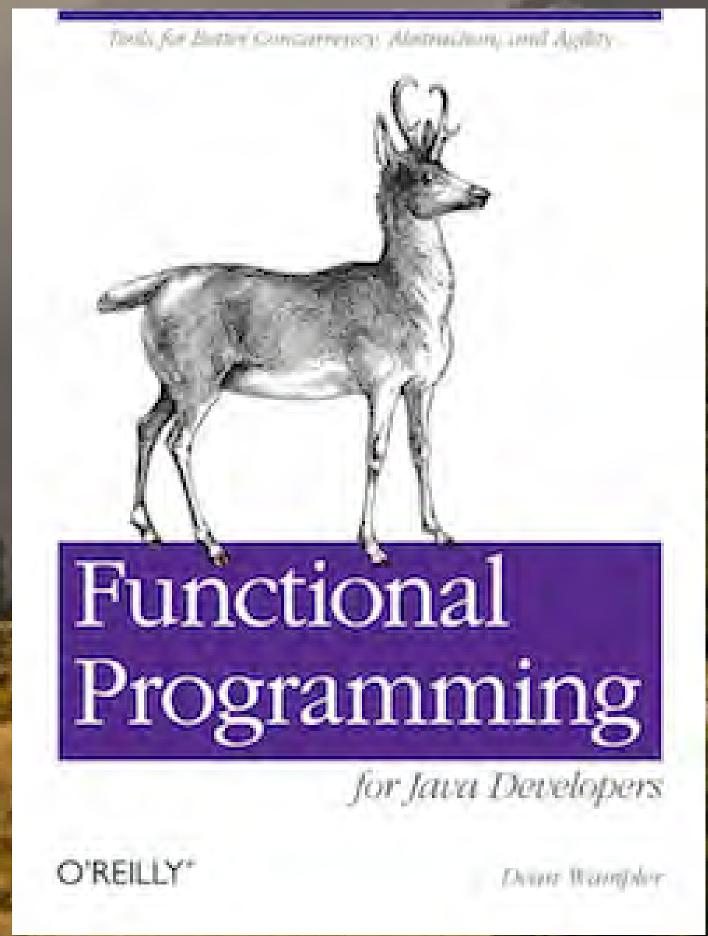


Wednesday, March 18, 15

Photos from Colorado, Sept. 2014.

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





This is my role. 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.

Later today:

Error Handling in Reactive Systems

3:30pm–5:00pm Thursday, 03/19/2015

Reactive and its variants

Location: 304

Tags: reactive

Dean Wampler *(Typesafe)*

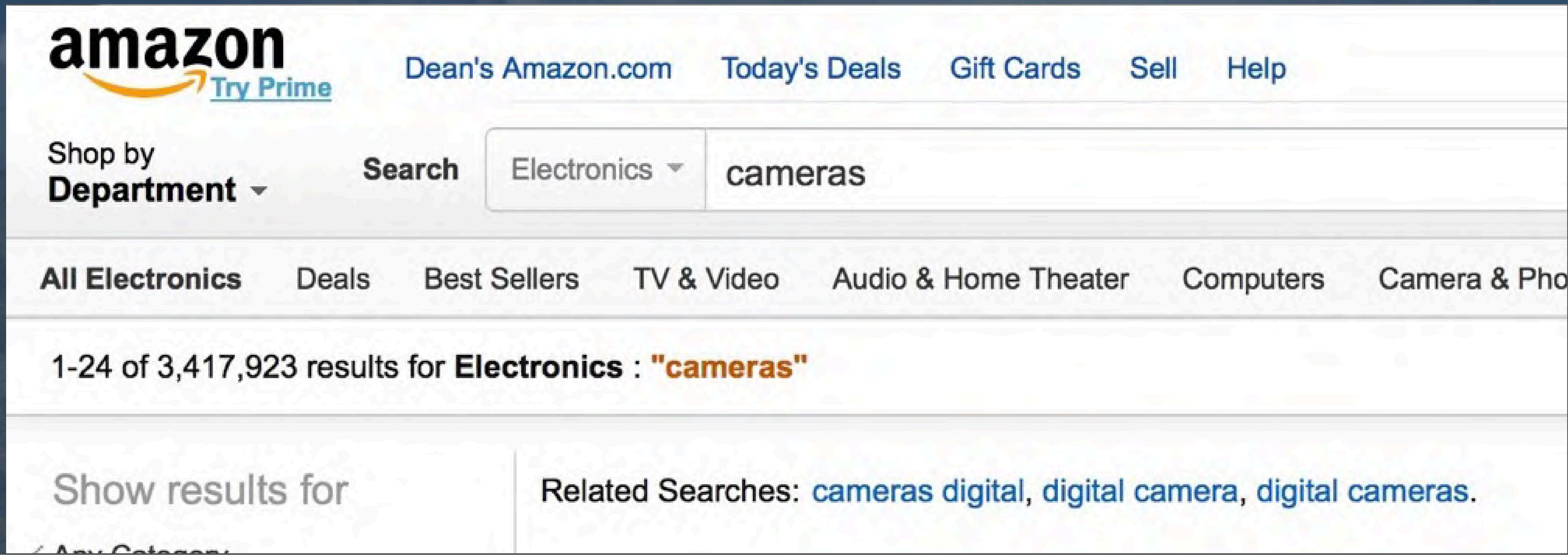
The Reactive Manifesto's "Resilient" trait requires a system to stay responsive when failures happen. I'll discuss how real-world systems do this, starting with in-process techniques in Go, Clojure's core.async, and Reactive Extensions. Next, I'll discuss how some tools prevent common failures in the first place. I'll finish with the Actor model's strategic use of supervisor hierarchies. Read more.

This talk is a general overview of Reactive concepts. I'm going to dive into the "Resilient" trait in more detail later today.

Wednesday, March 18, 15

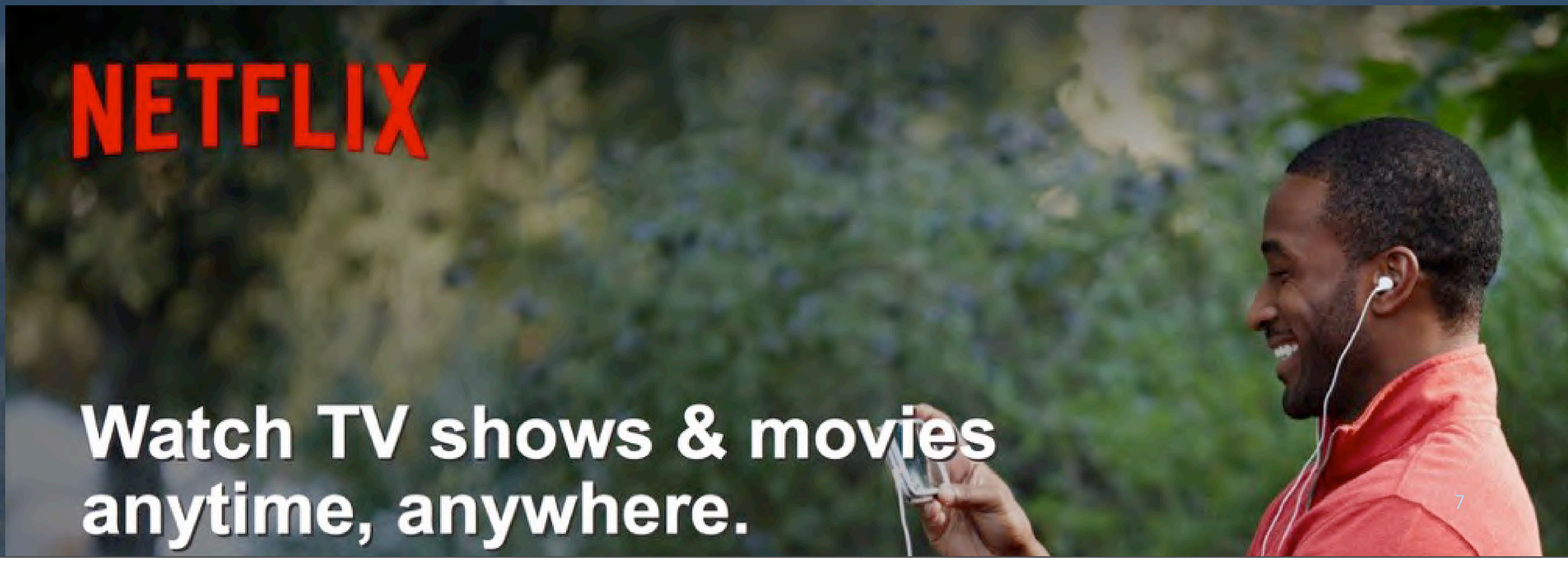
Let’s motivate the notion of “reactive” systems by exploring some common scenarios we see today.

Cyber Monday?



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?



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.

Wednesday, March 18, 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

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

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!

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!

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?

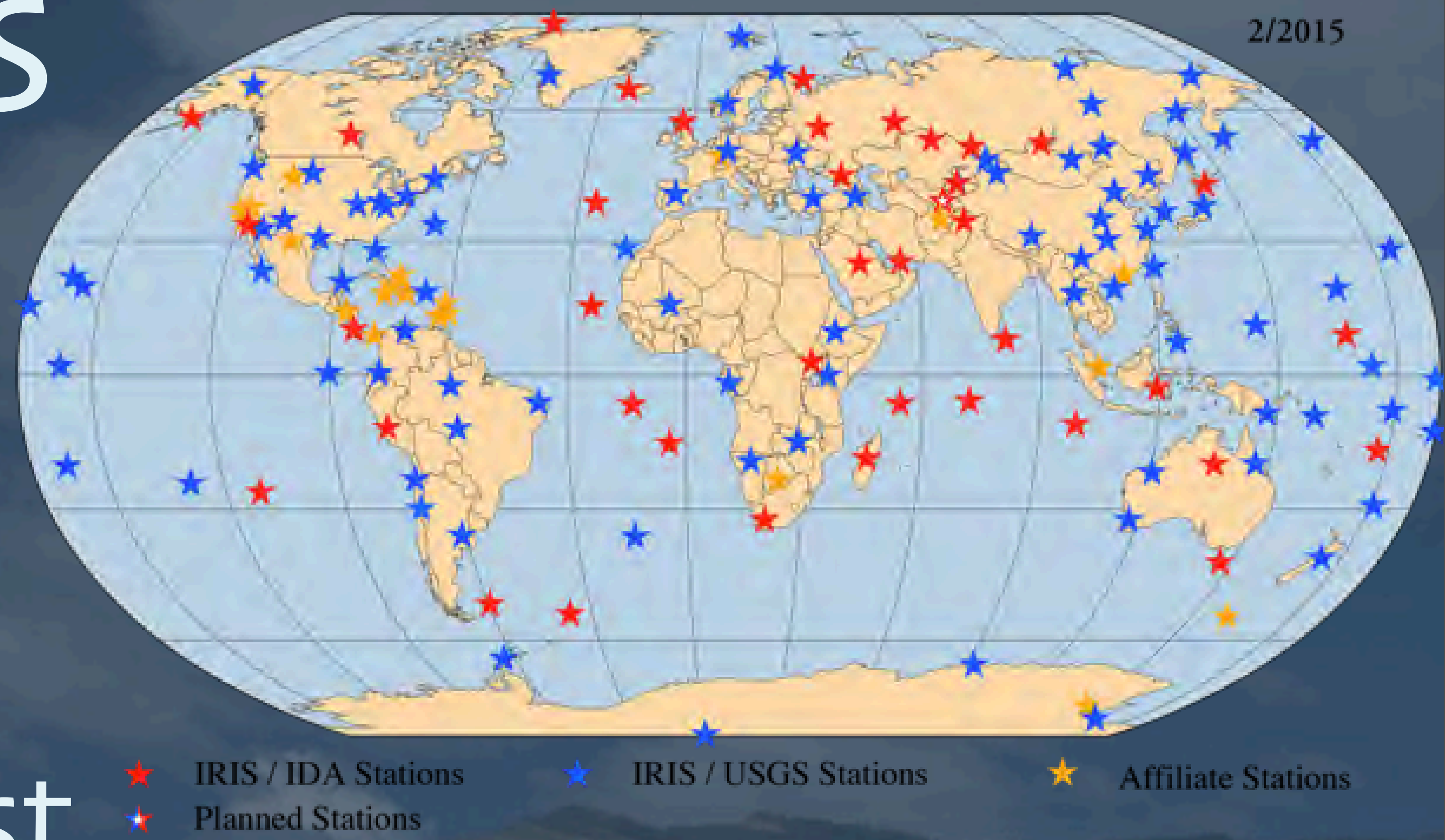
Trucks, Farm Equipment

Connectivity:

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

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

Remote Sensors

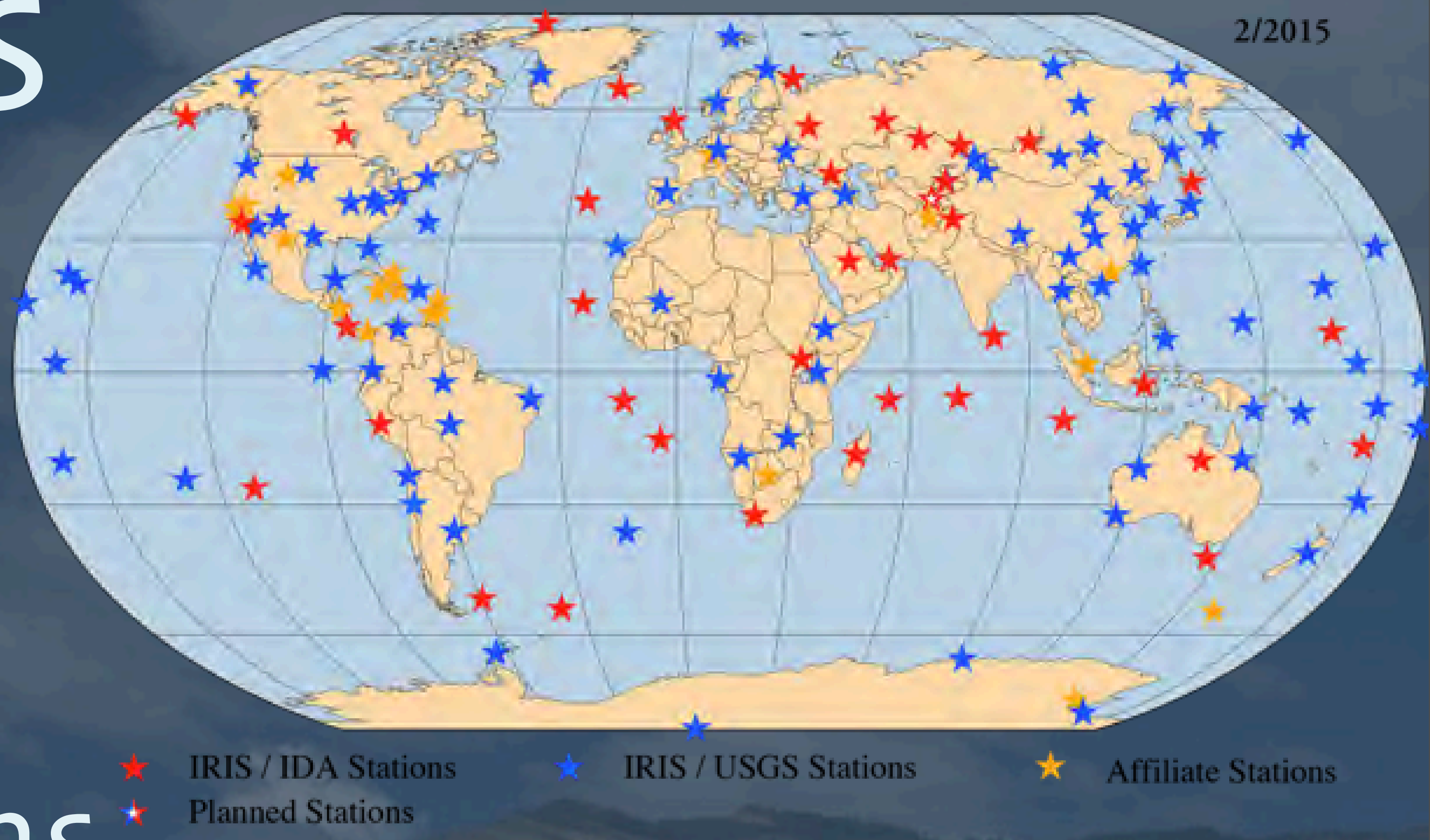


Human to Real-time Responsiveness:

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

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

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?

Some rural areas don't have sufficient wireless data coverage for farm robots to remain to 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?

Wednesday, March 18, 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:

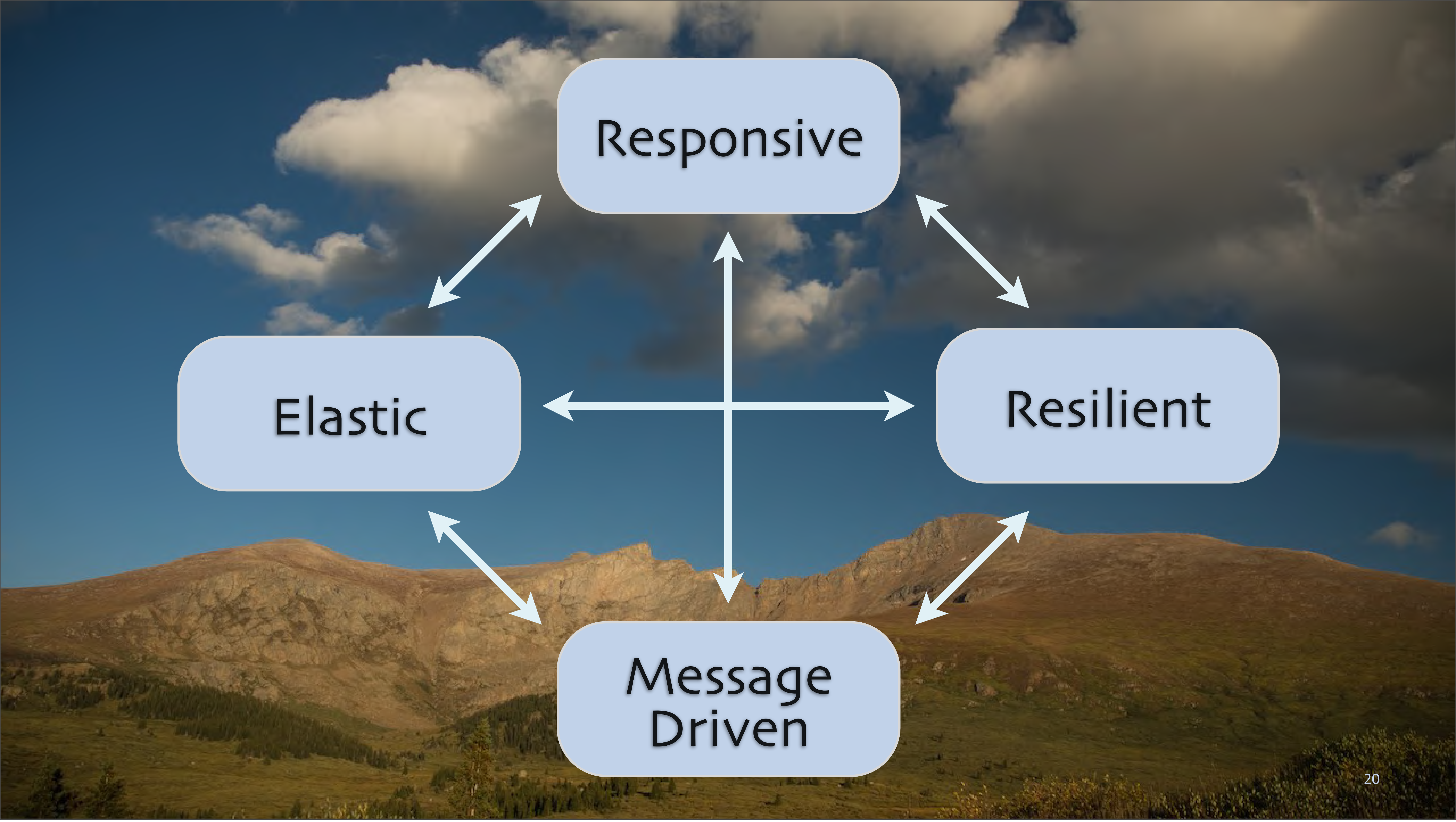
- ToD packet storms
- Fire & break-in detection: automatic notification of authorities

Wednesday, March 18, 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/

Wednesday, March 18, 15

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



Wednesday, March 18, 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.

Wednesday, March 18, 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.

Wednesday, March 18, 15

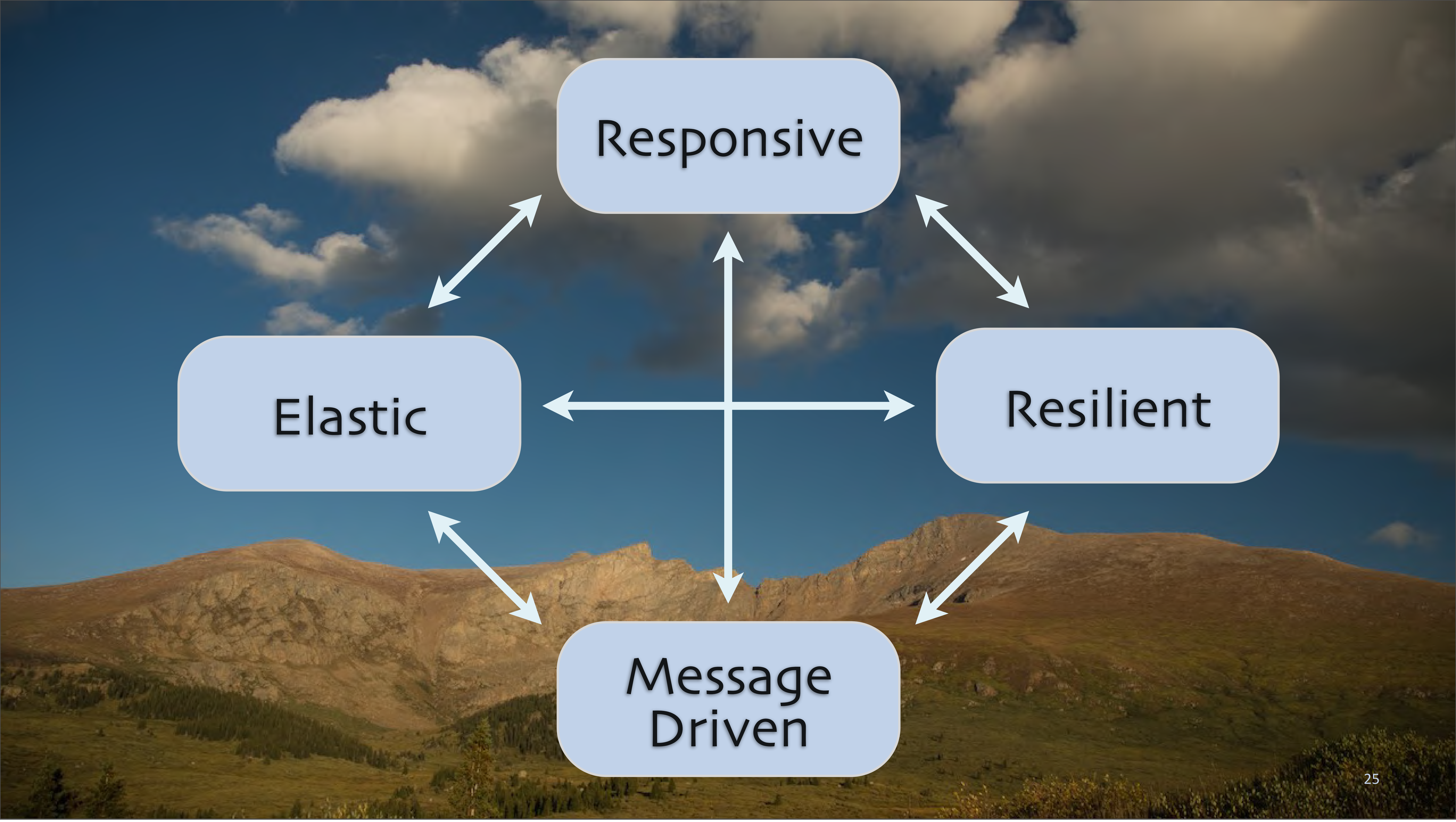
Before discussing them in detail, let’s slay some myths.

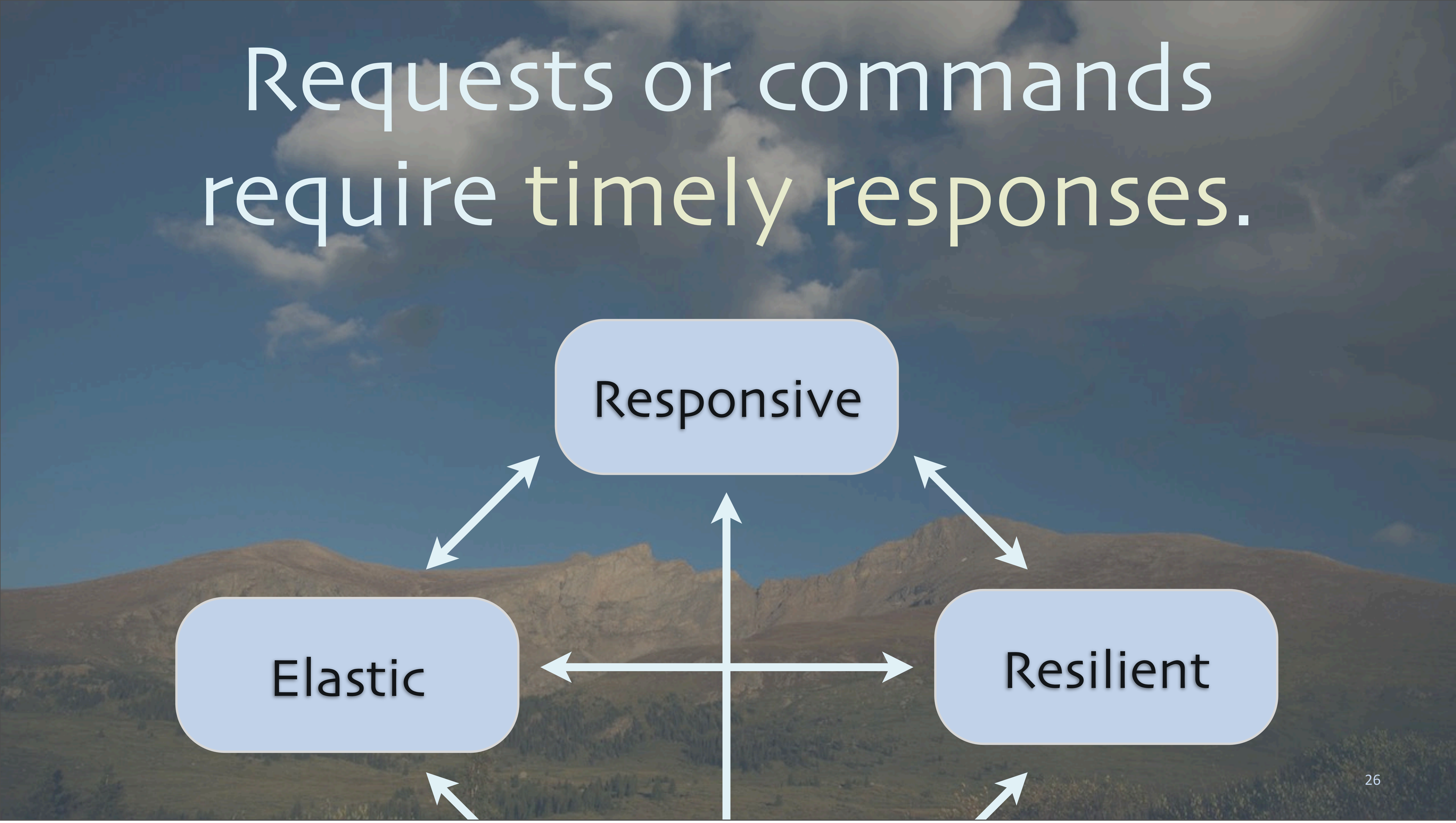
Wednesday, March 18, 15

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

Wednesday, March 18, 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.


Responsive
Elastic
Resilient
Message Driven



Wednesday, March 18, 15

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

Wednesday, March 18, 15


Responsive
Cornerstone of
usability and utility.
28


Responsive
Requires rapid detection of errors and quick responses.
29


Responsive
Requires predictable
response times
and quality of service.
30

Wednesday, March 18, 15

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

Wednesday, March 18, 15

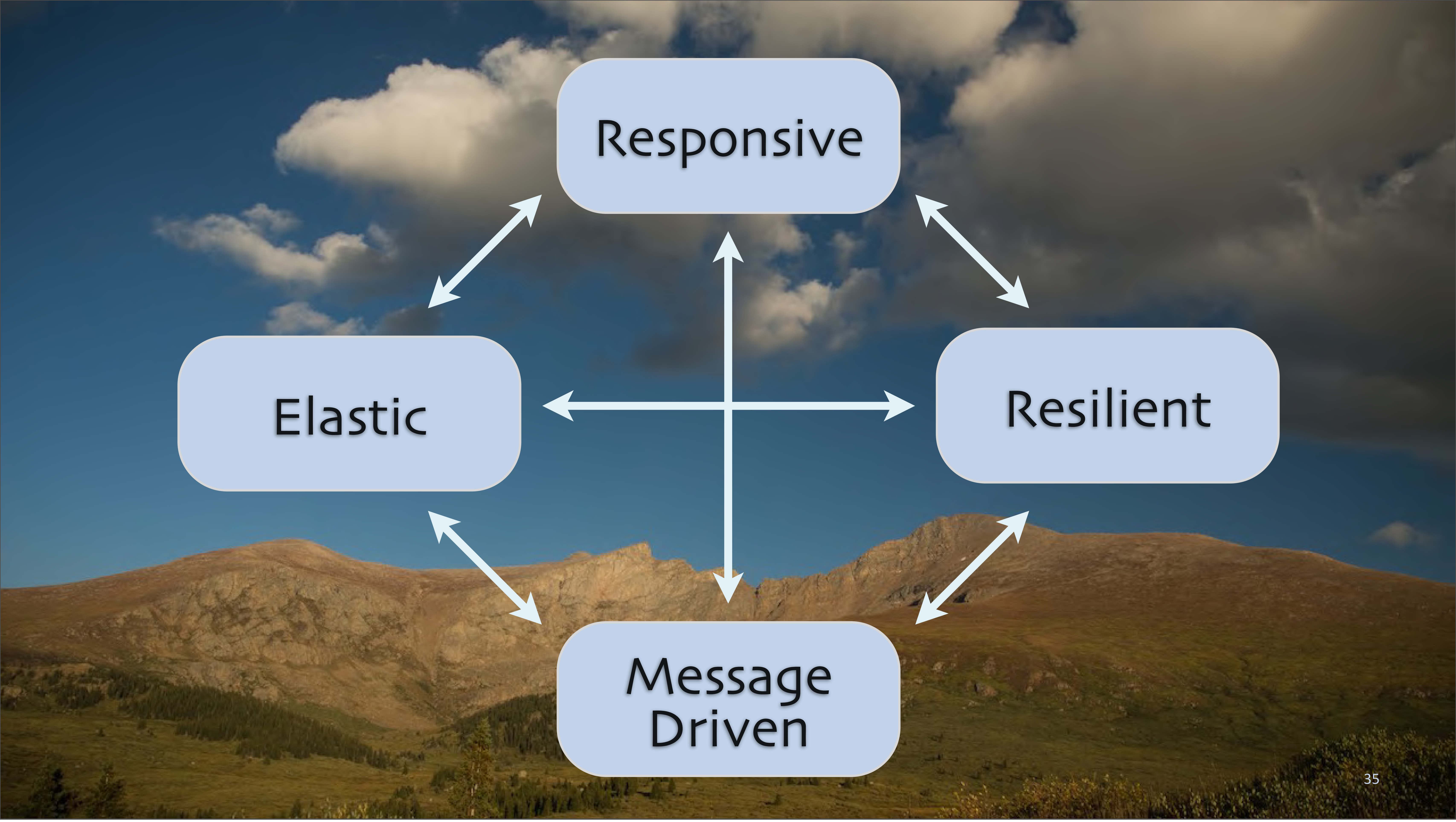
You have to be a clock watcher of sorts.

Example:
Netflix Simian Army

Wednesday, March 18, 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.


Responsive
Elastic
Resilient
Message Driven

Wednesday, March 18, 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.
39

Wednesday, March 18, 15

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

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”.

Wednesday, March 18, 15

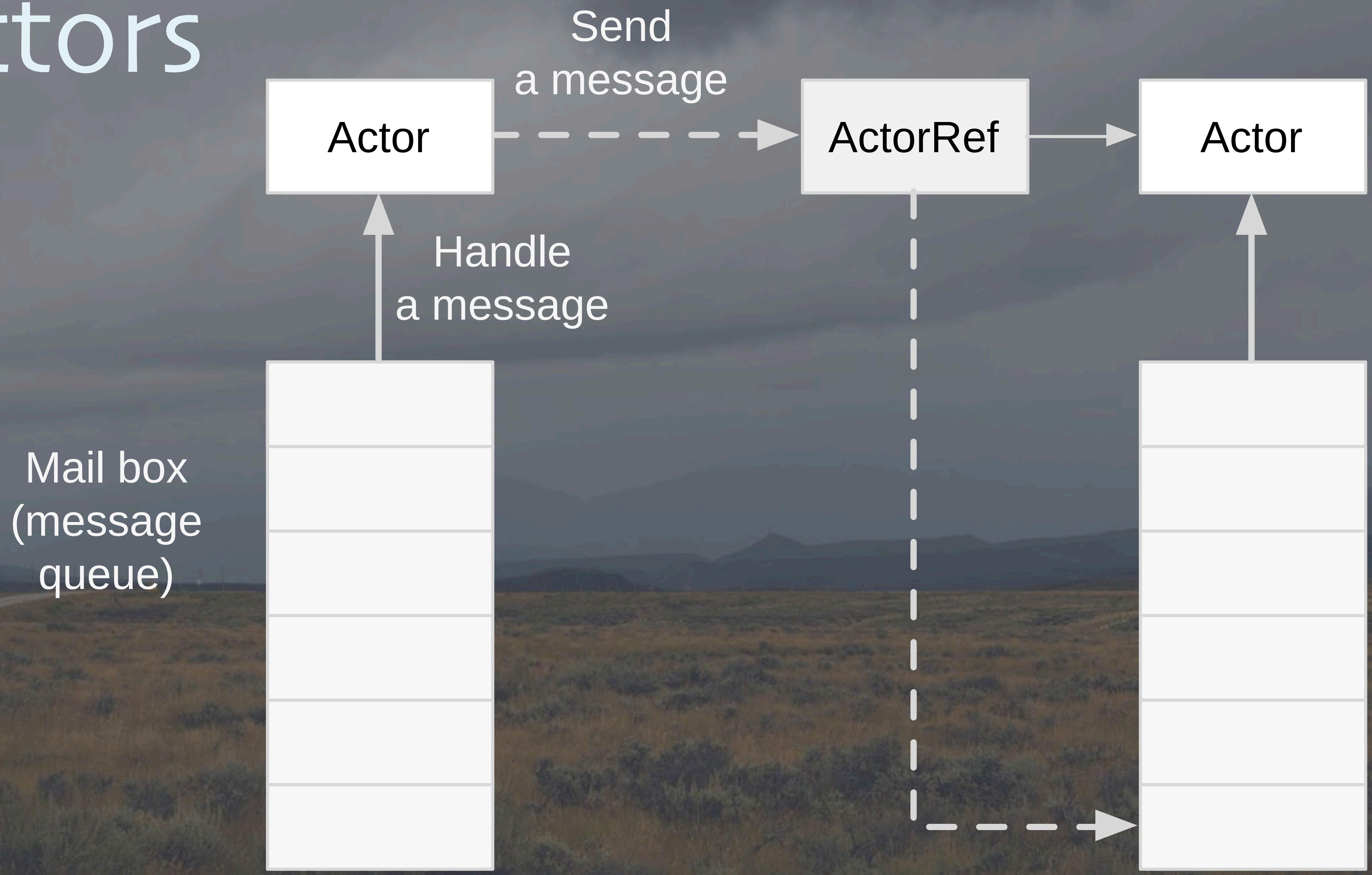
We’ll see an example of what I mean.

Wednesday, March 18, 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.

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.

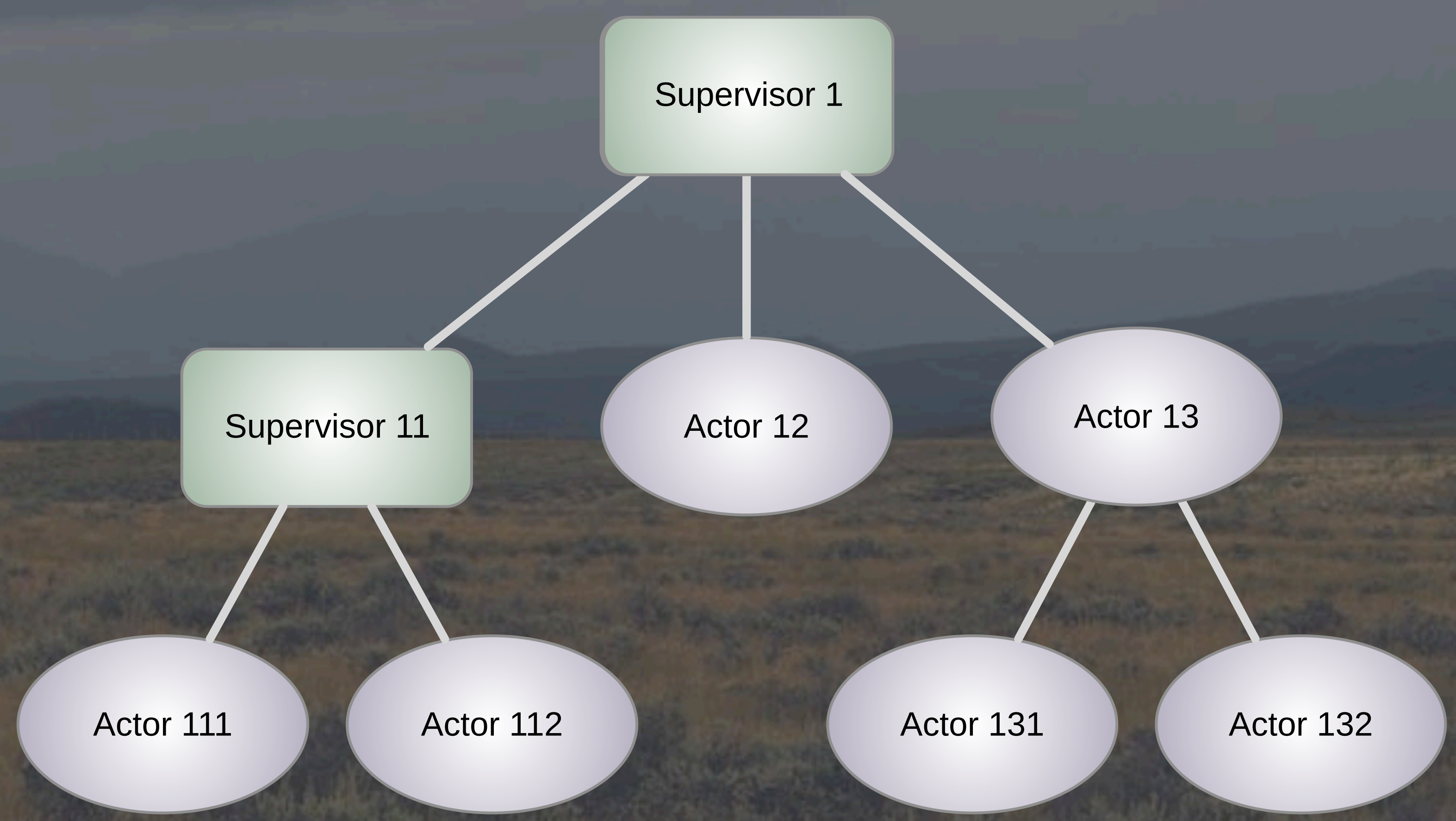
Actors



Wednesday, March 18, 15

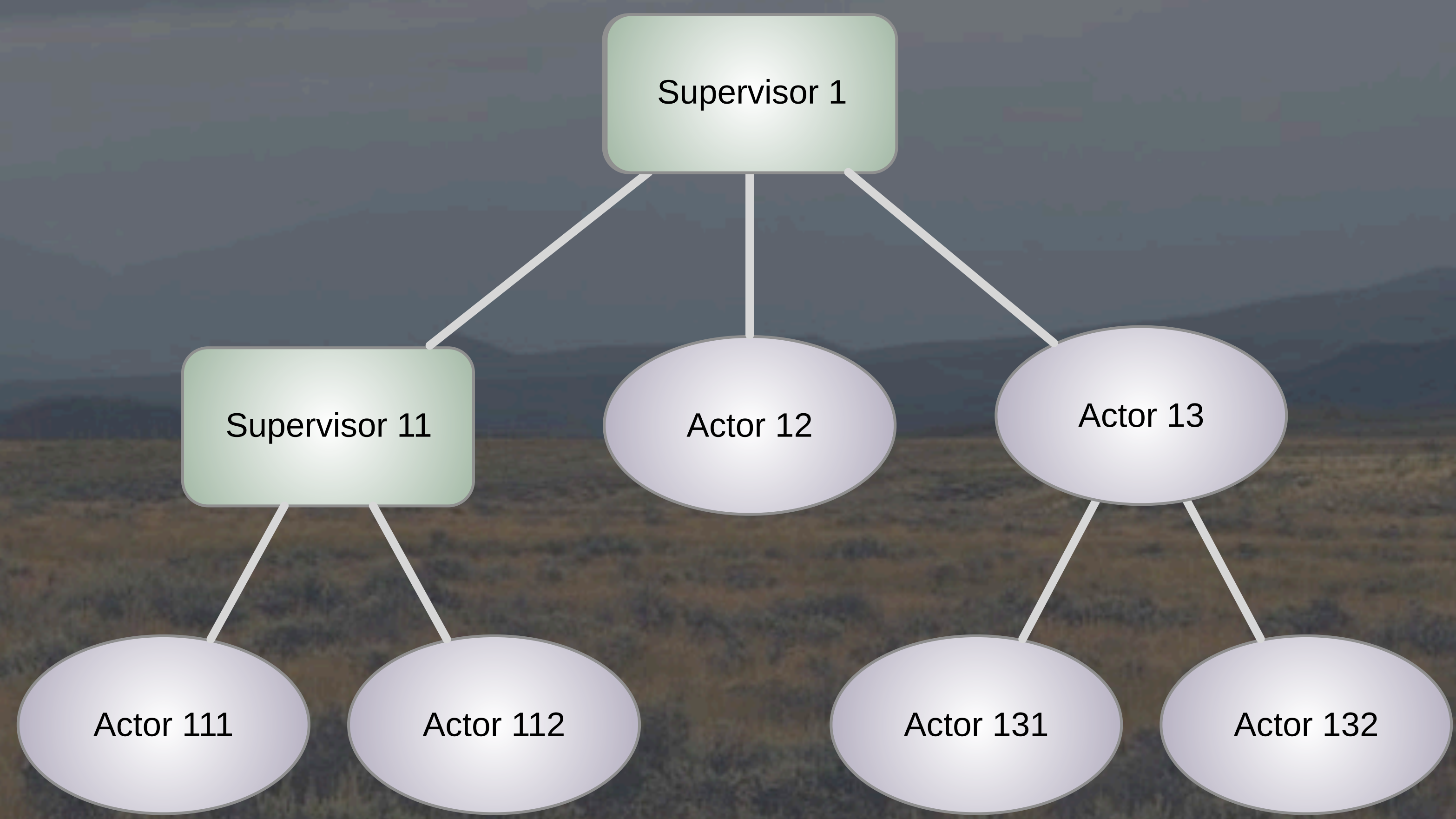
Actors are similar to objects in Smalltalk and similar, message-passing systems; autonomous agents with defined boundaries that communicate through message passing. Actors, though process each message in a threadsafe way, so they are great for concurrency. (This diagram illustrates the Akka implementation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

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

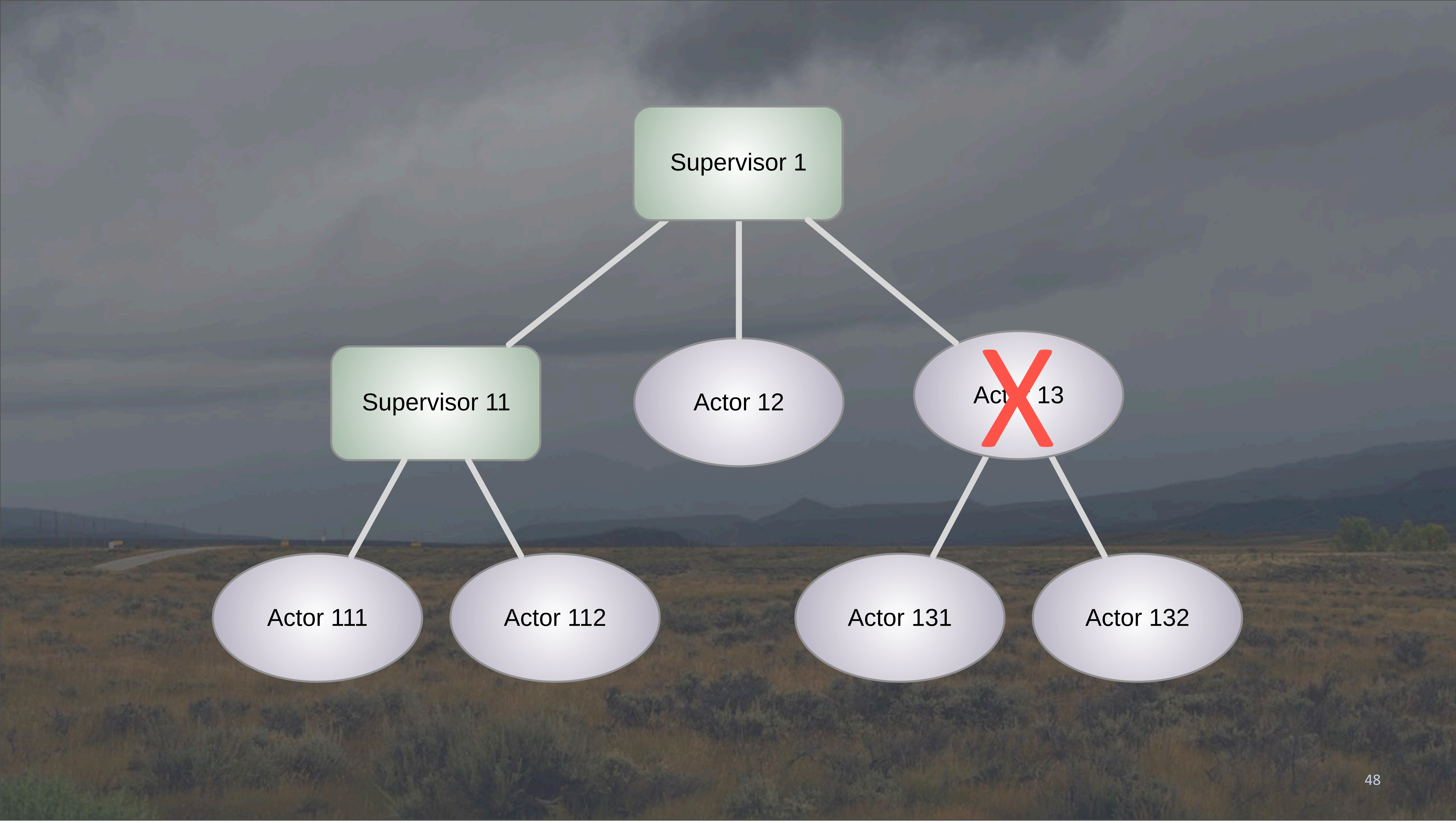


There are lots of so-called Actor systems, but be wary of them unless they have this sophisticated supervision model or something like it (even though the original Actor model of Hewitt, et al., didn’t include supervision like this...).

Generalizes nicely to distributed actor systems.

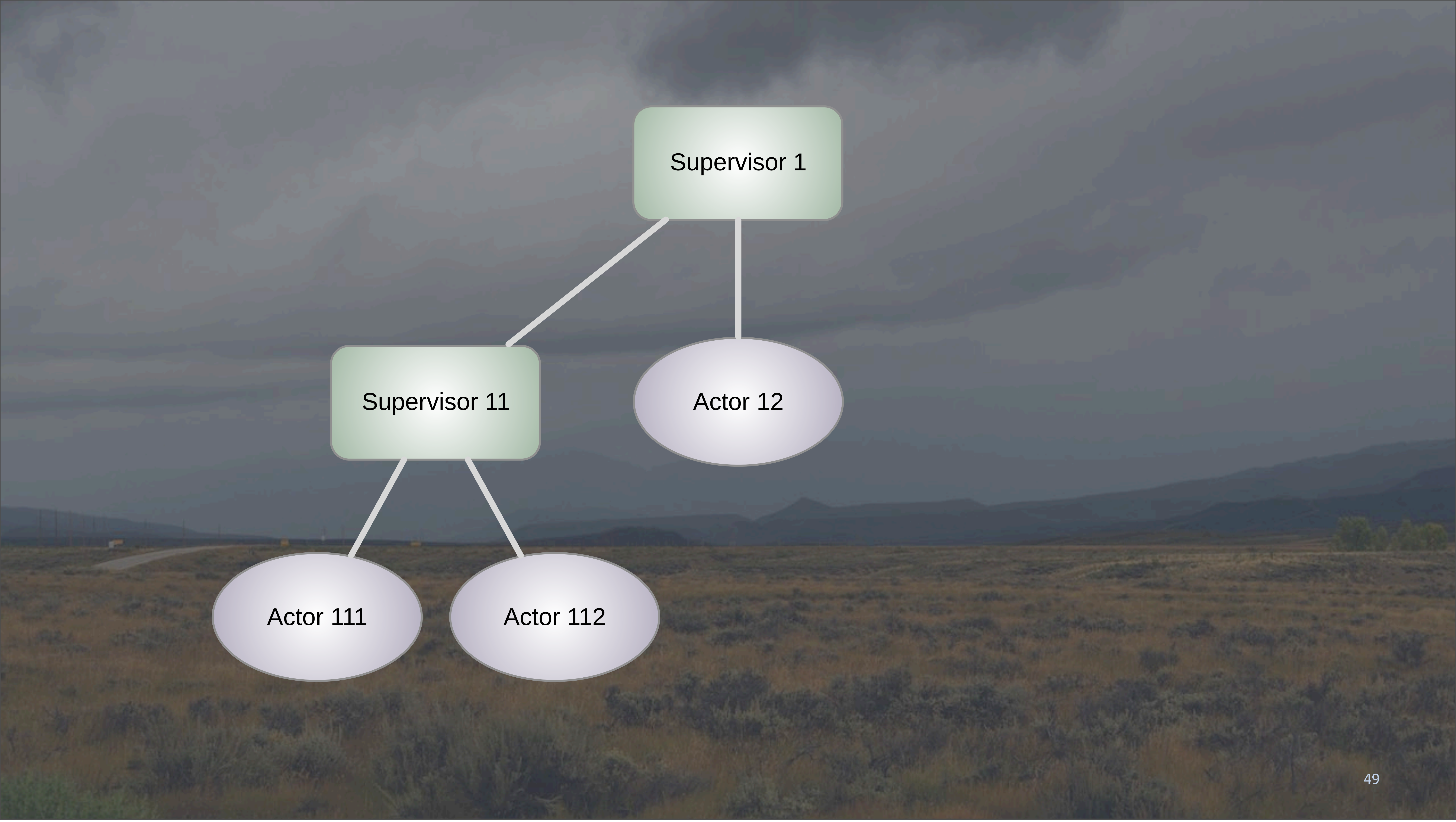


Uniform abstraction when the lines cross process and machine boundaries.



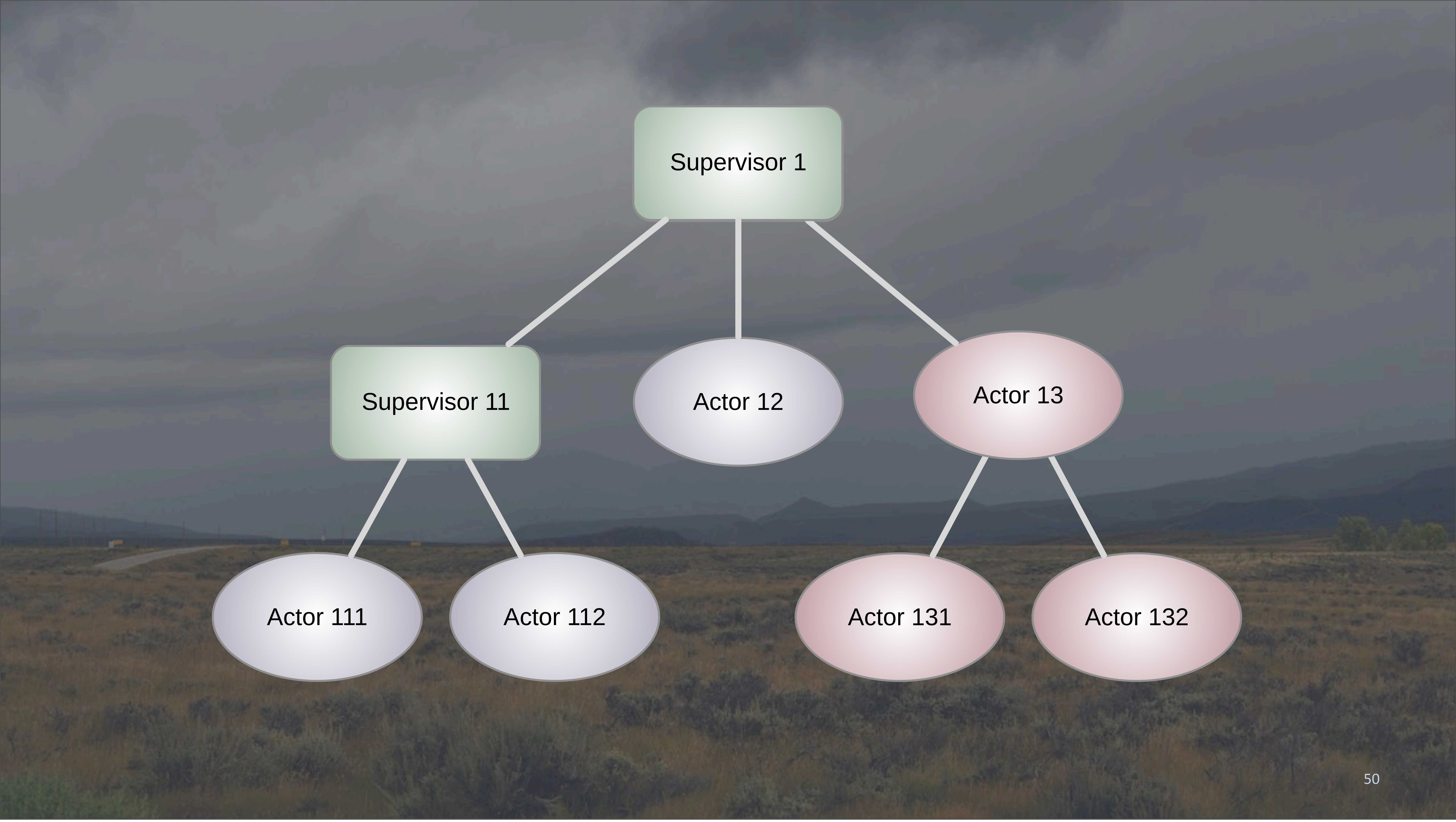
Wednesday, March 18, 15

An actor dies. This one has children, too.



Wednesday, March 18, 15

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



Wednesday, March 18, 15

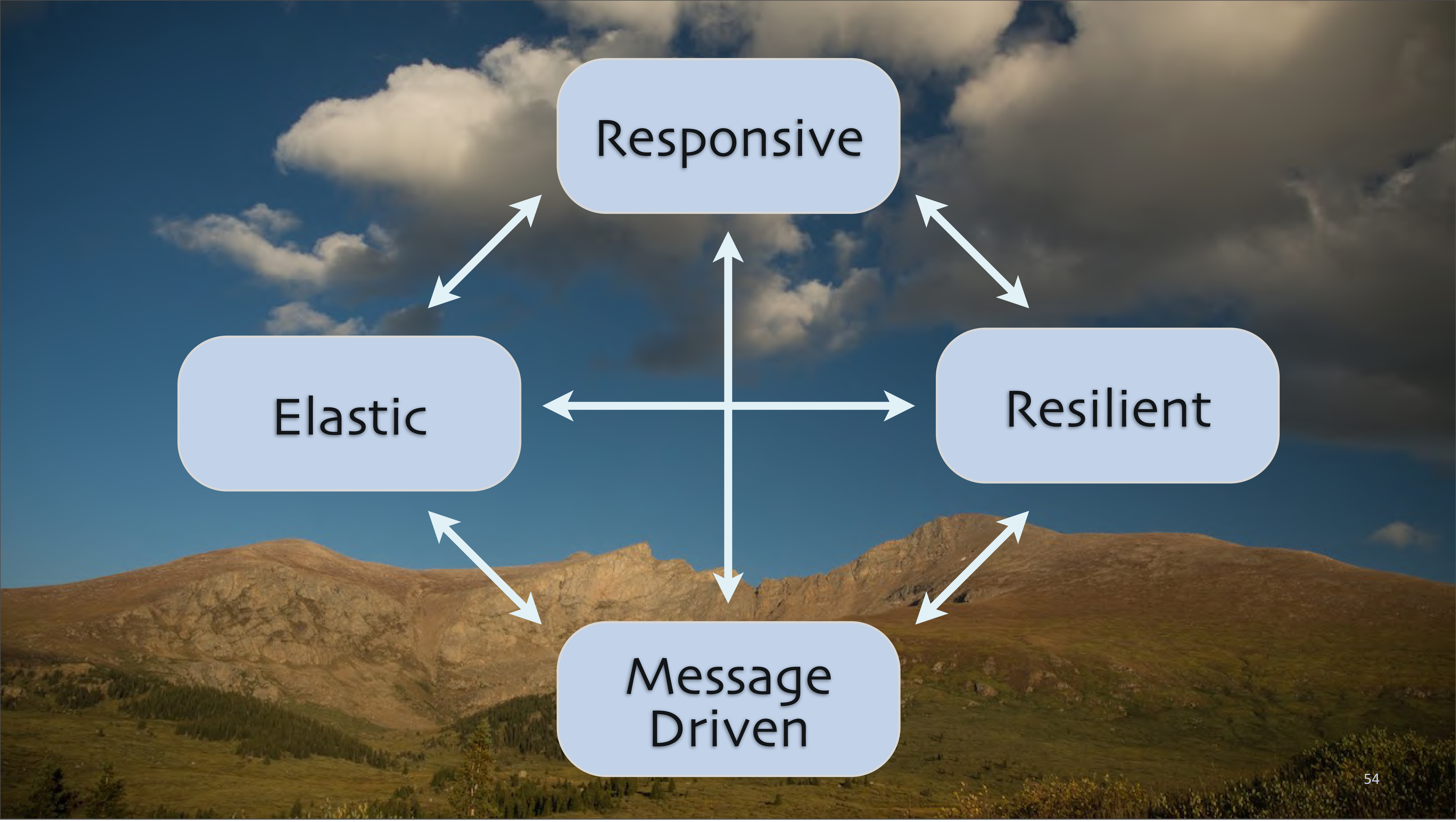
... it reconstructs it.

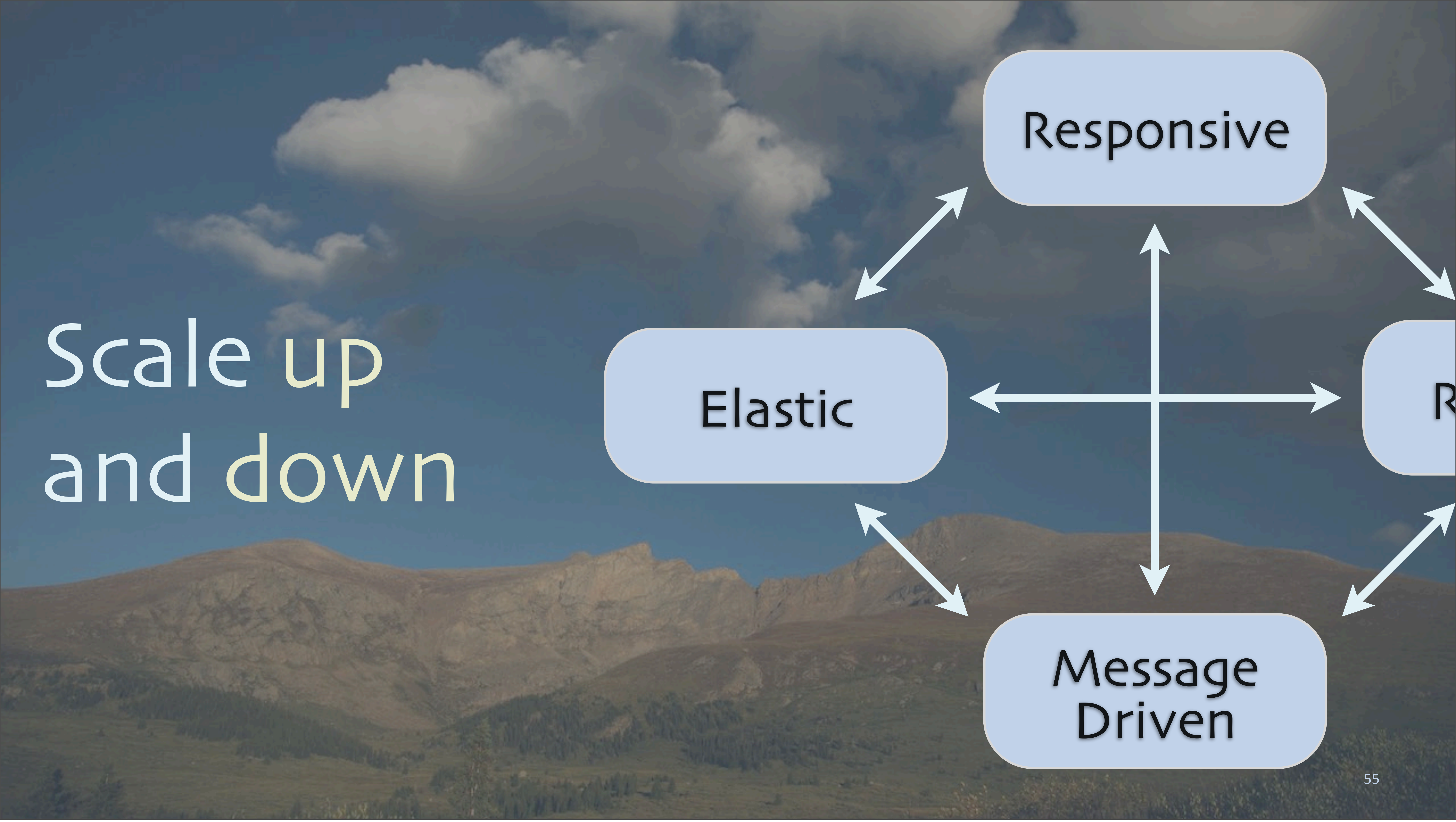
Wednesday, March 18, 15

I discuss this in more depth in my afternoon talk.

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

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


Responsive
Elastic
Resilient
Message
Driven



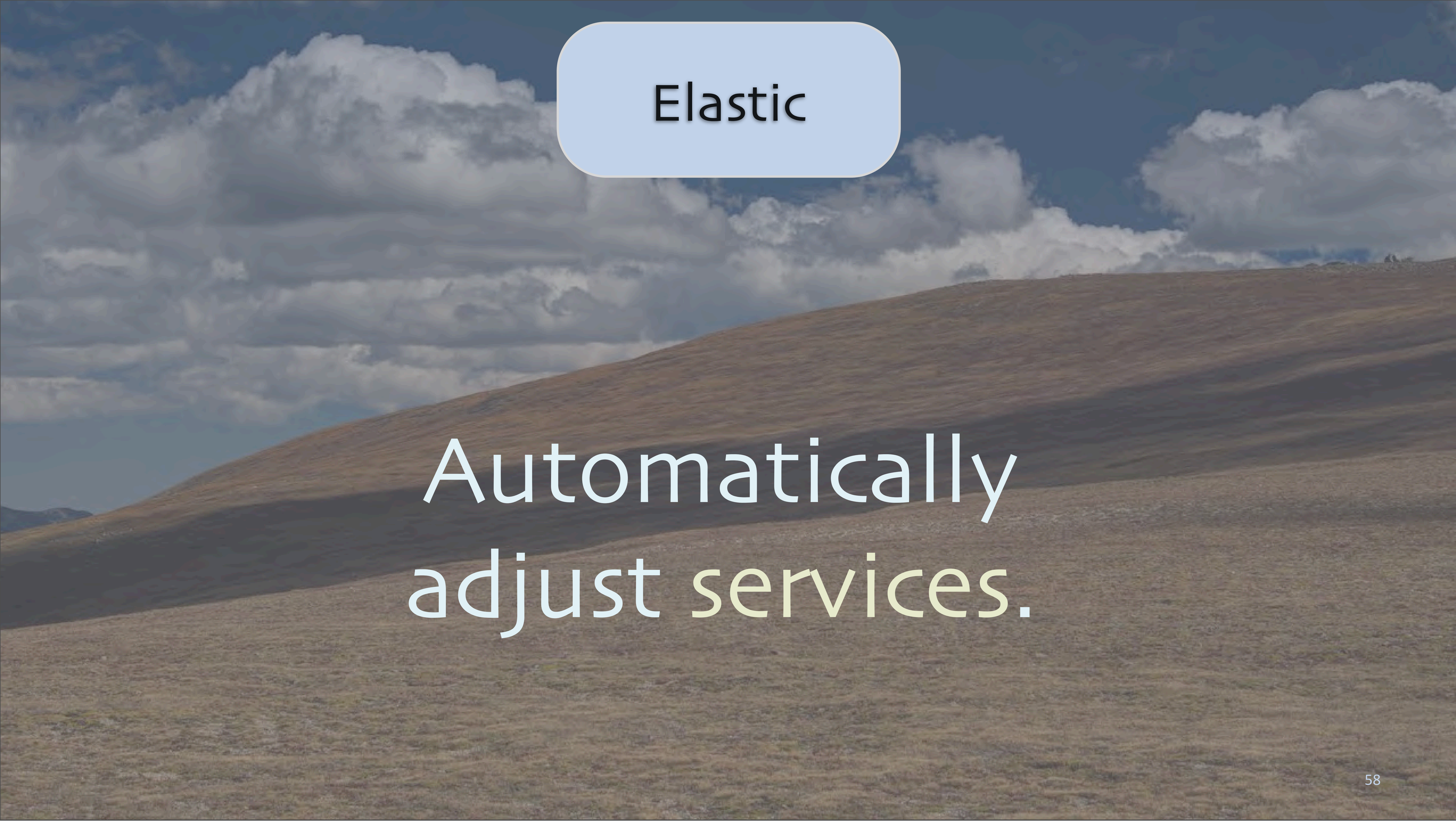
Wednesday, March 18, 15

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

Elastic

Wednesday, March 18, 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.



Wednesday, March 18, 15

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

Wednesday, March 18, 15

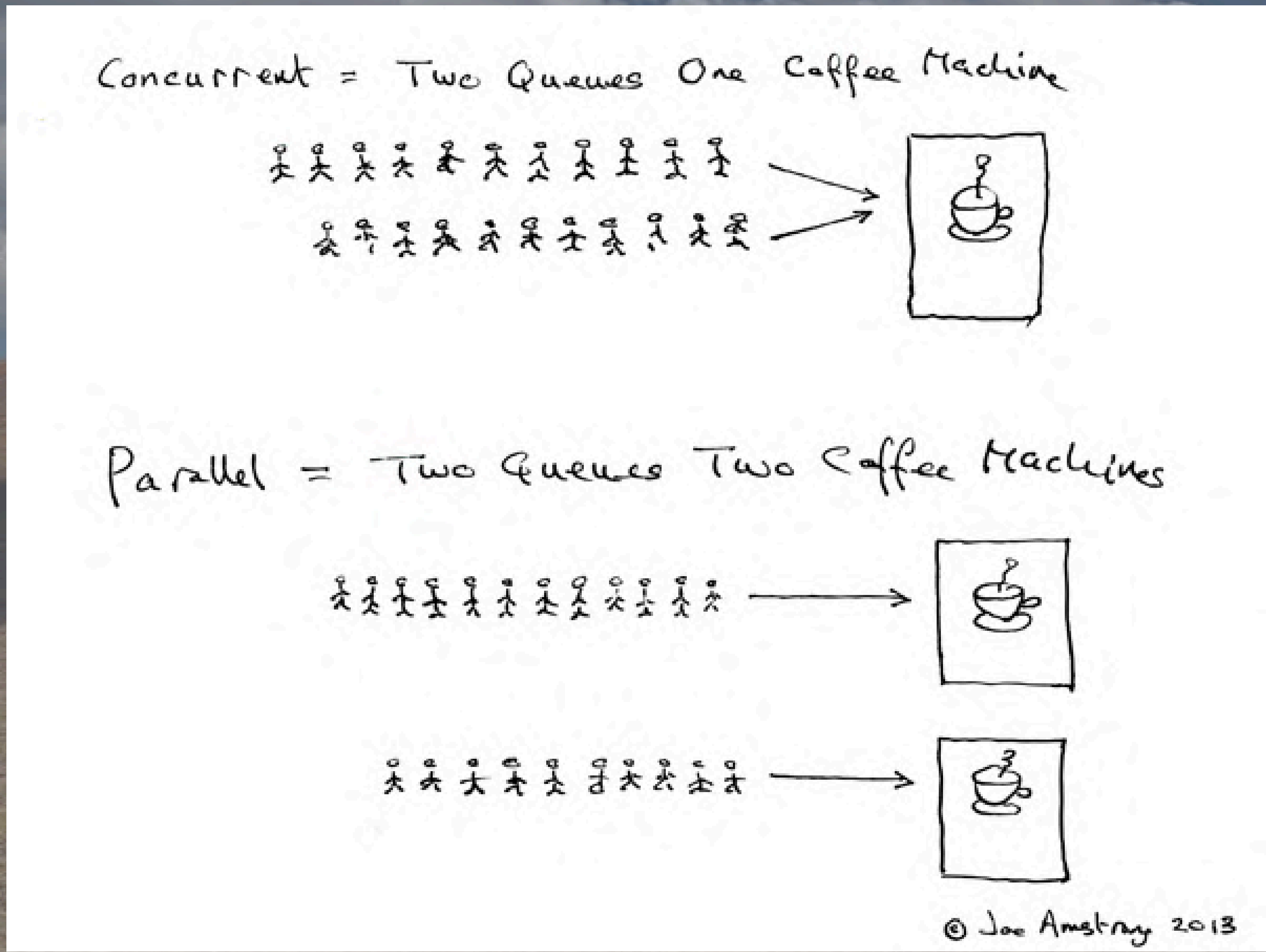
Typically you use redundant services again, across commodity (interchangeable) hardware.

Elastic

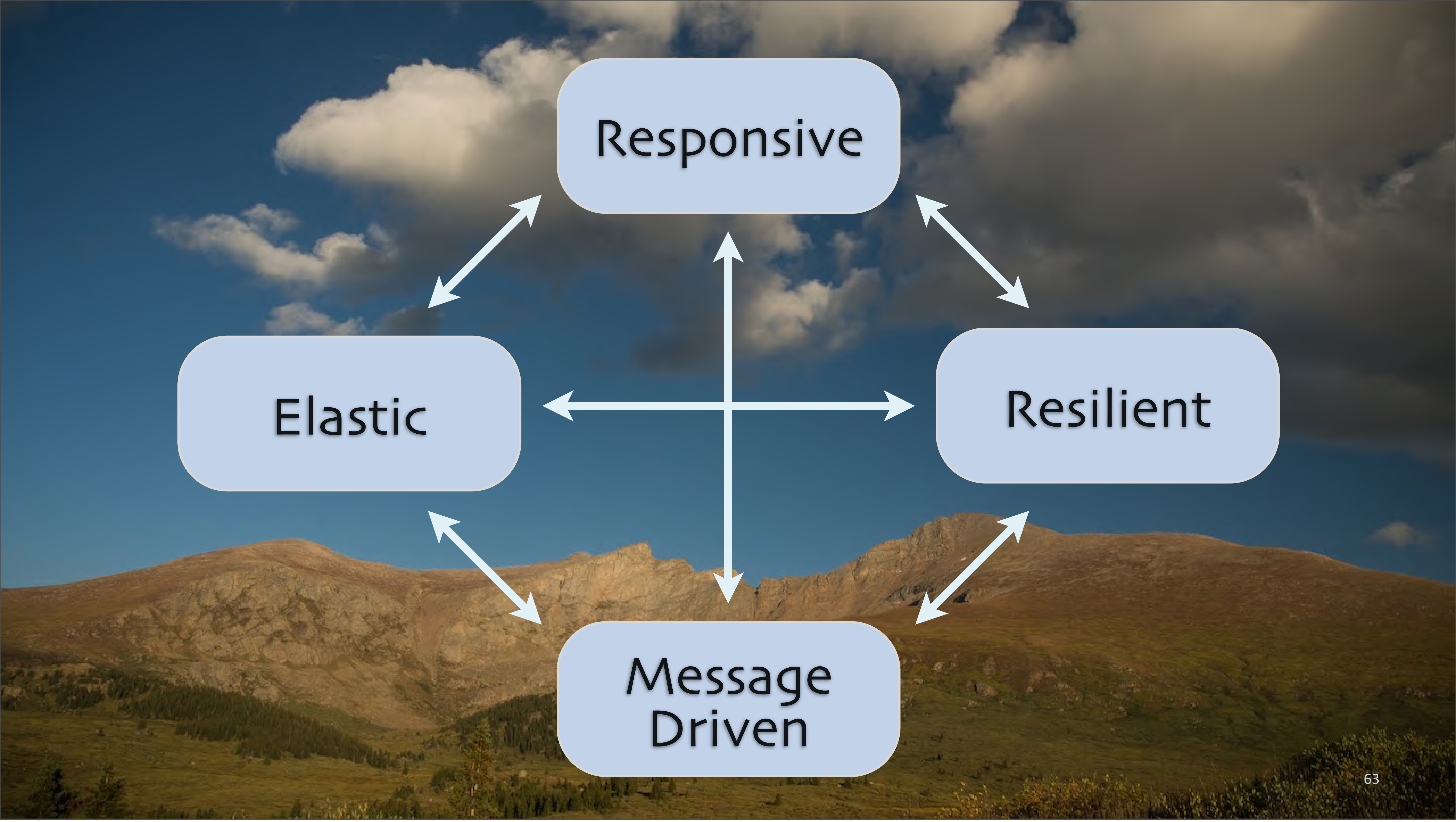
No bottlenecks
or contention points.

So, harder if the nodes hold data!

Elastic



“Concurrent” vs. “parallel”.


Responsive
Elastic
Resilient
Message Driven

Wednesday, March 18, 15

To be responsive to the world around you, you must interact with it through messages.

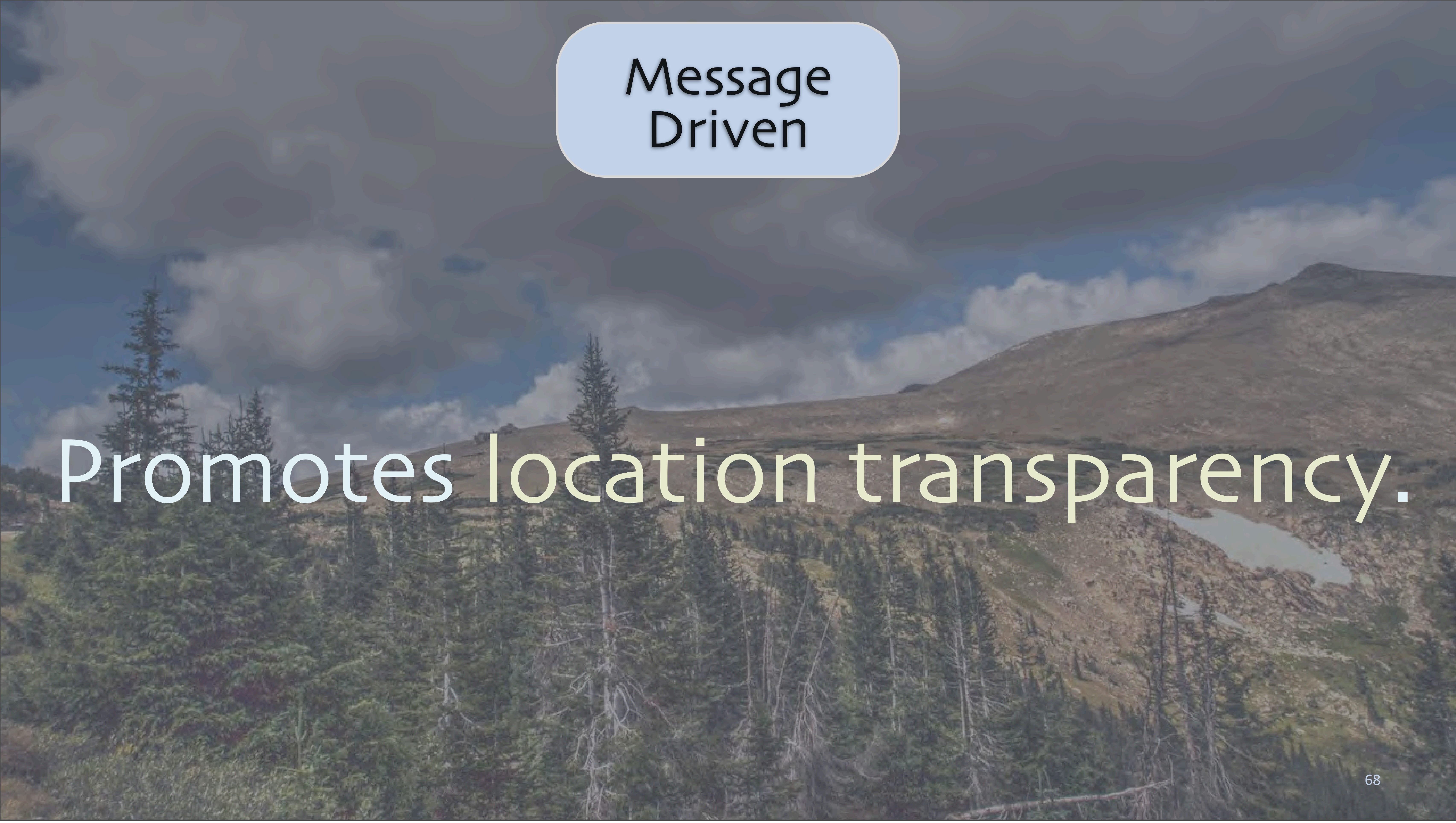

Message
Driven

Wednesday, March 18, 15

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

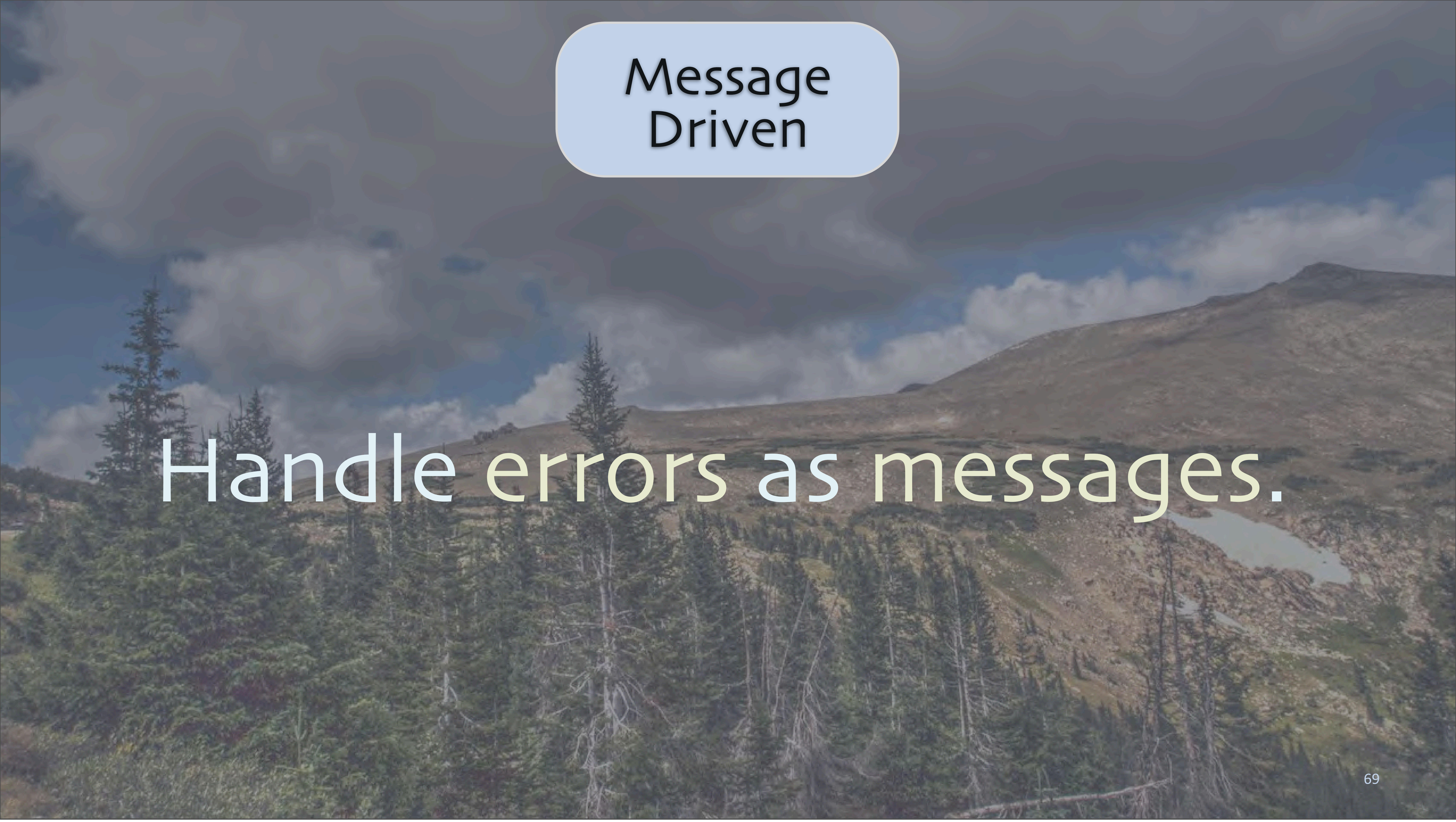
Wednesday, March 18, 15

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.



Wednesday, March 18, 15

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



Wednesday, March 18, 15

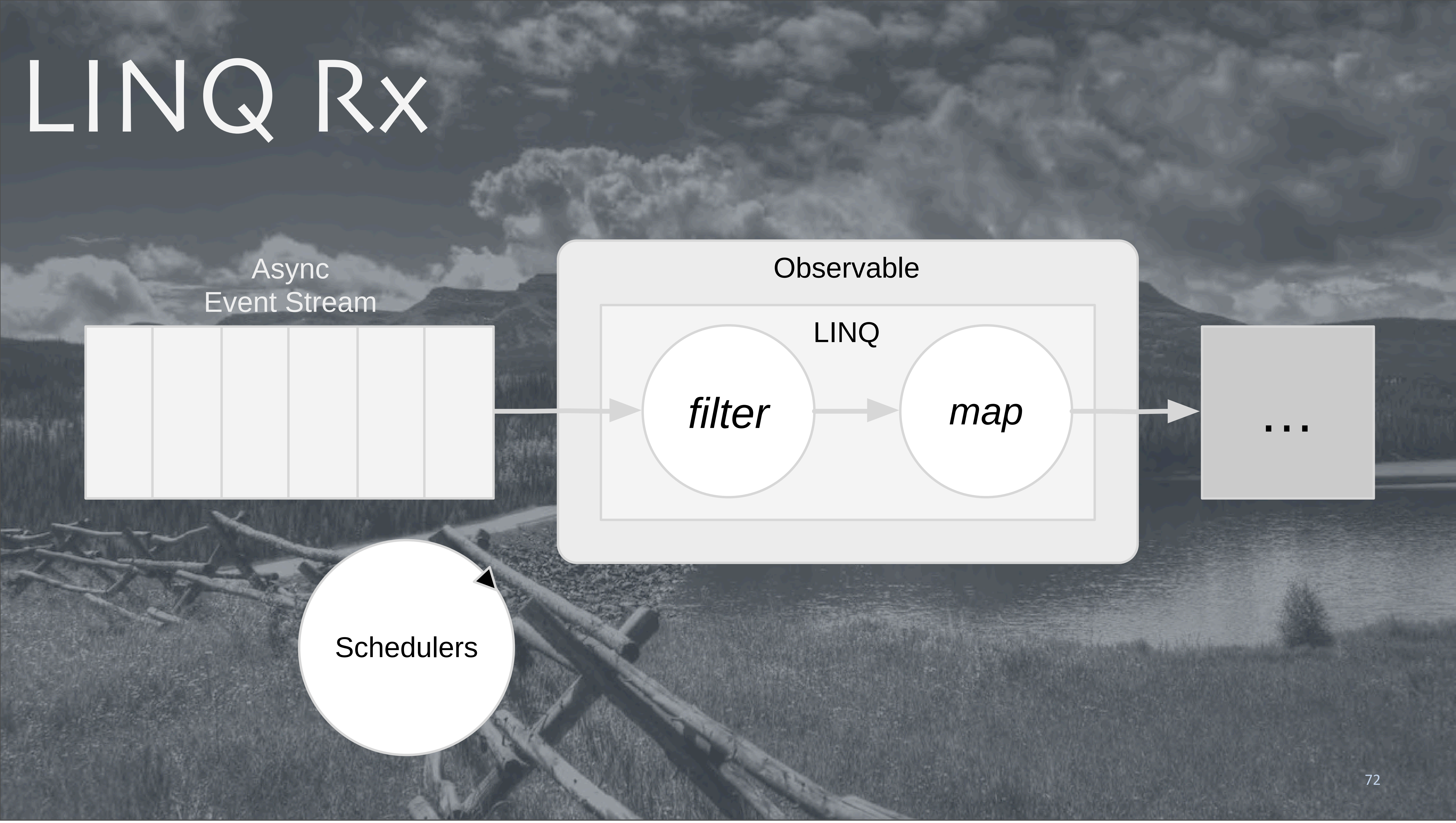
Also, you can use the same message infrastructure to communicate error scenarios as well as normal processing.

Wednesday, March 18, 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.

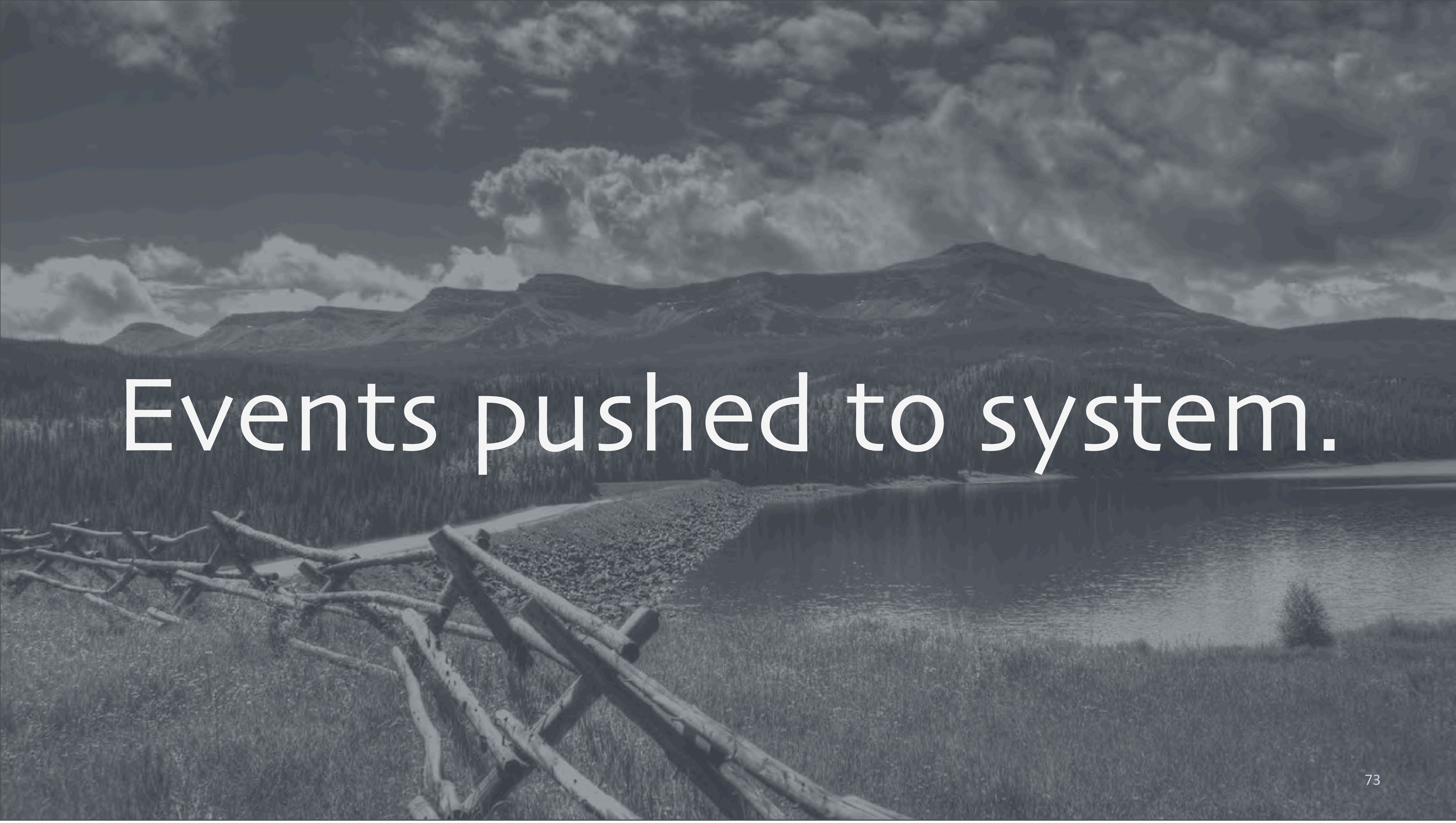
Reactive Extensions (Rx)

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

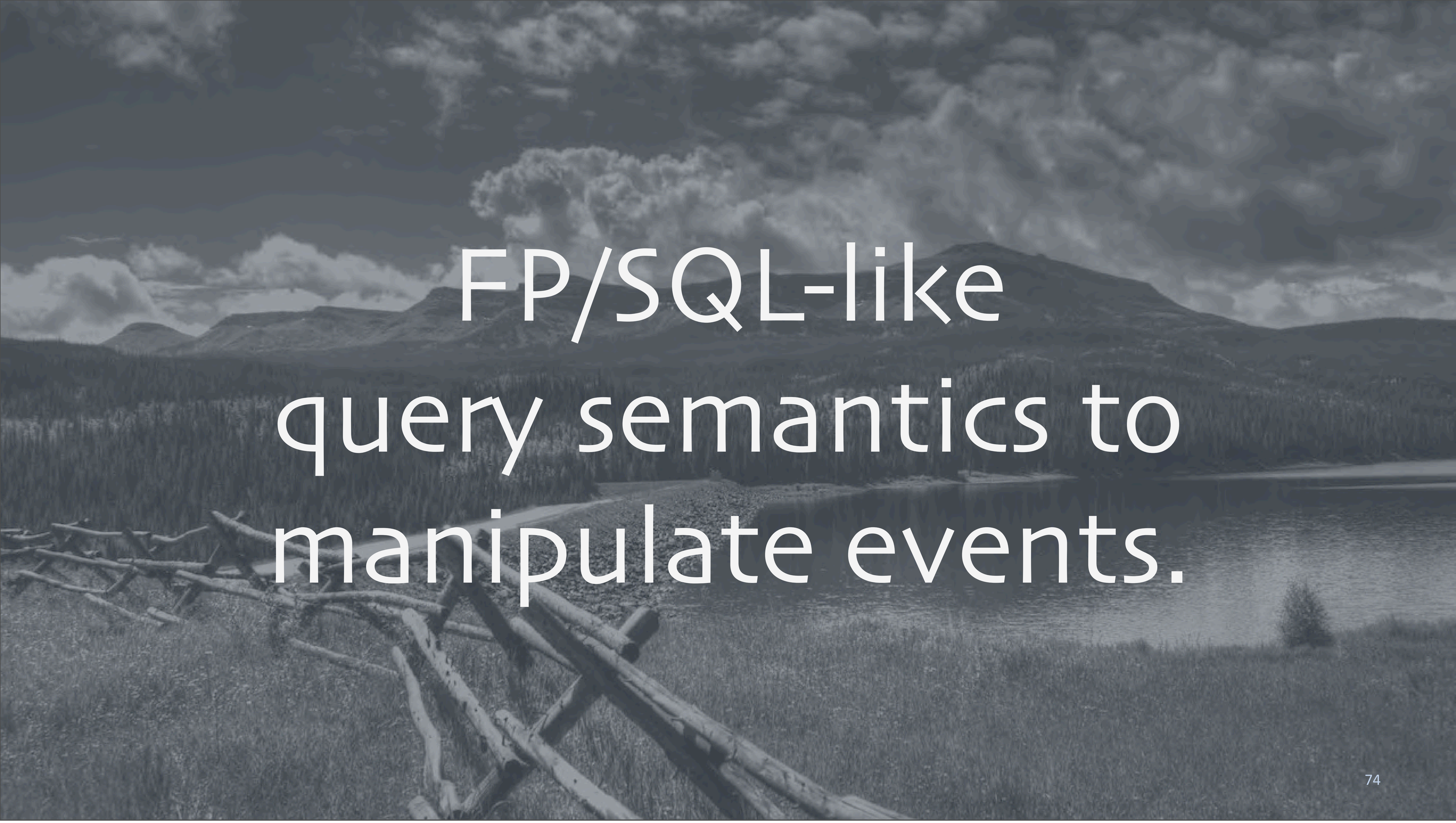


Wednesday, March 18, 15

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.



It’s essentially a push model.


FP/SQL-like
query semantics to
manipulate events.

How: Reactive Tools

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

Wednesday, March 18, 15

We mentioned a few already, let’s fill in some details. This won’t be an exhaustive list.

Wednesday, March 18, 15

CSP - Communicating Sequential Processes.

Functional Programming

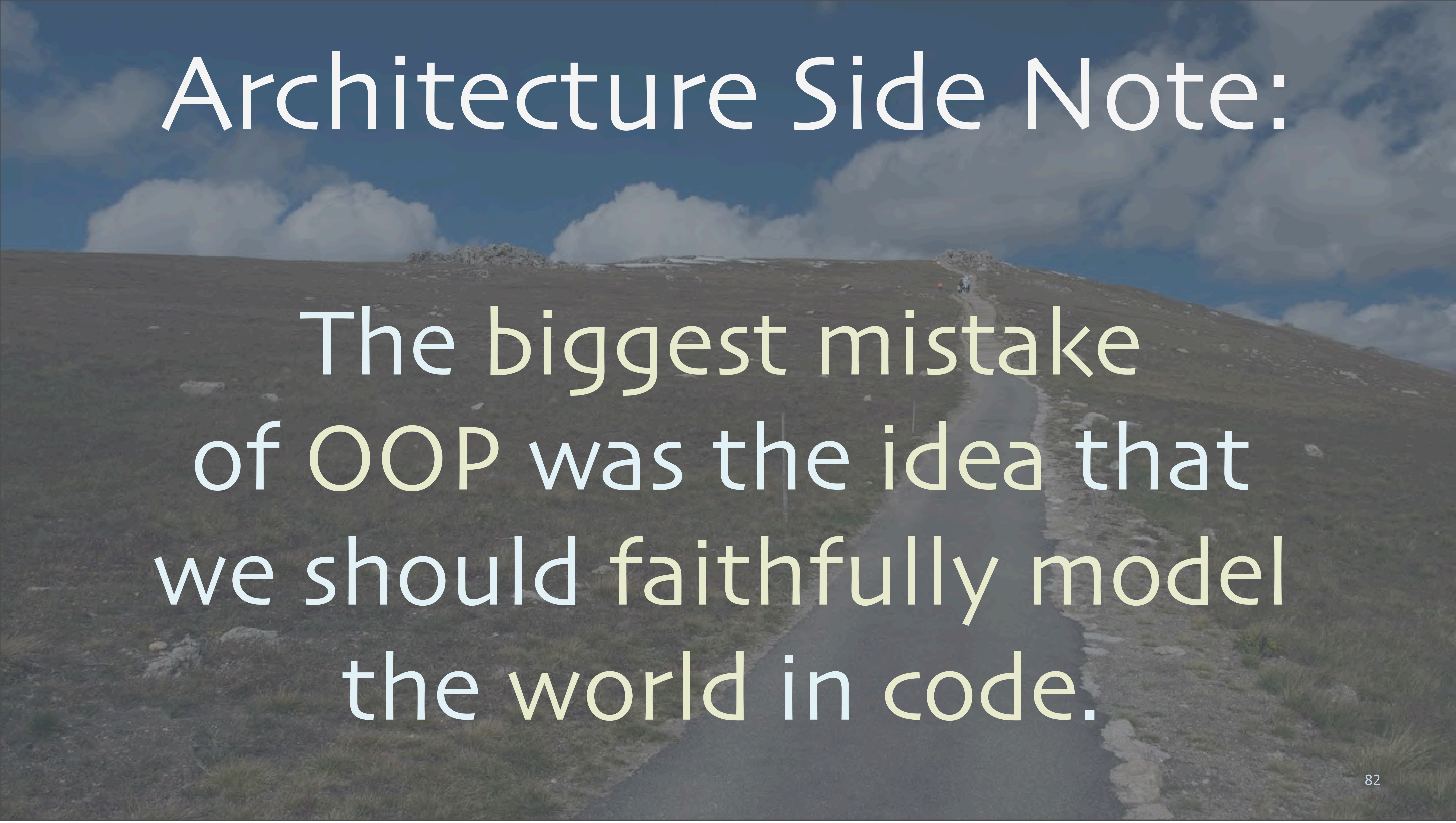

Functional Programming
Prefer immutable values and
side-effect free functions...
79

Functional Programming

... because they eliminate the problems of multithreading.

Why, because all the problems are caused by attempting to coordinate access to shared,mutable state. If the state is no longer mutable, then it's trivial to share. There are many other advantages of FP.

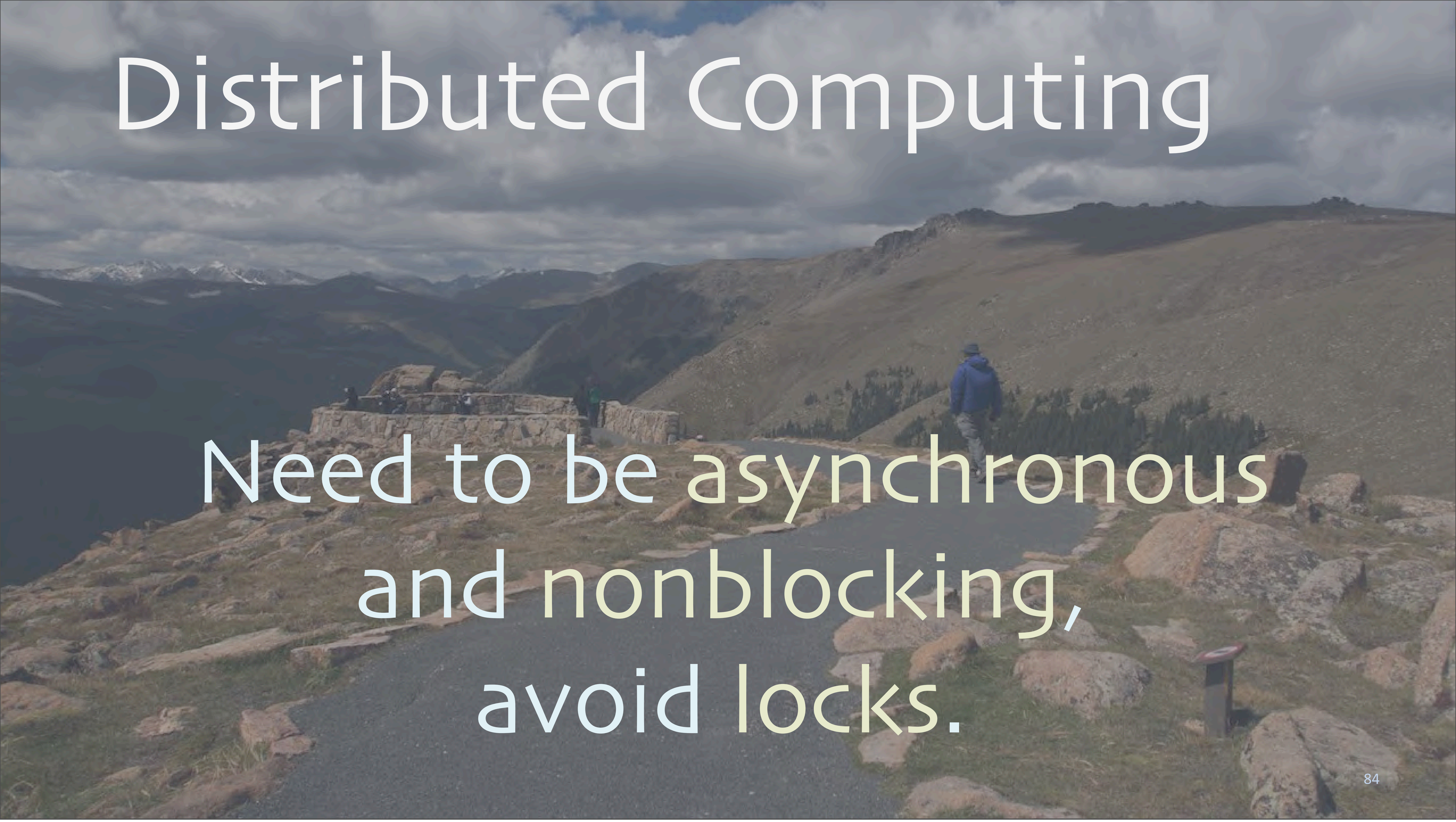
This is Scala’s view, that objects are useful as module constructs, but the code inside should be functional.



Wednesday, March 18, 15

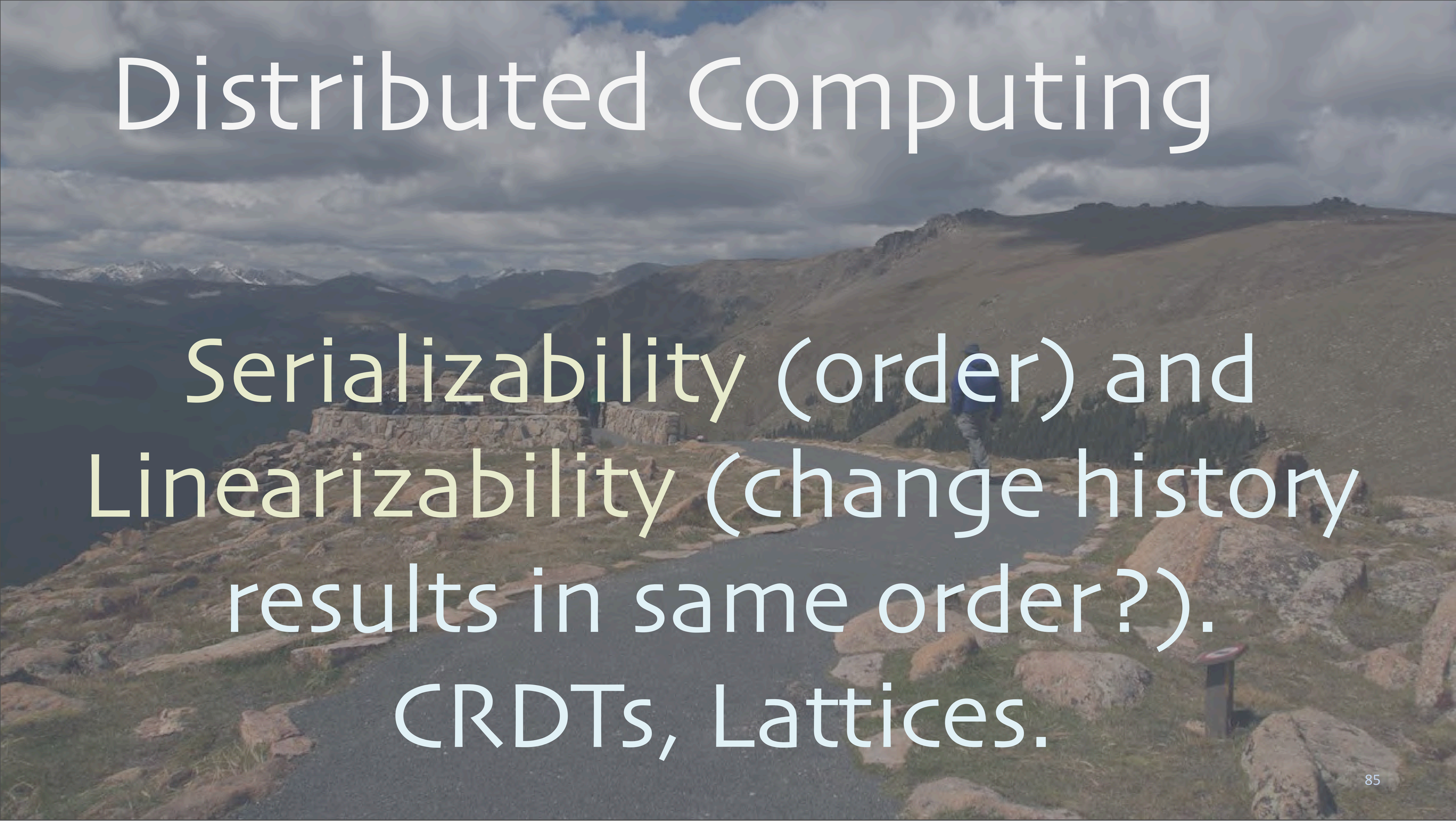
Controversial, but I believe much of our code bloat and inflexibility is actually caused by this mistaken belief. Example: Does a payroll calculator need the concepts of Pay, Deductions, etc.? Or should we just stream numbers through math logic?

Distributed Computing



Wednesday, March 18, 15

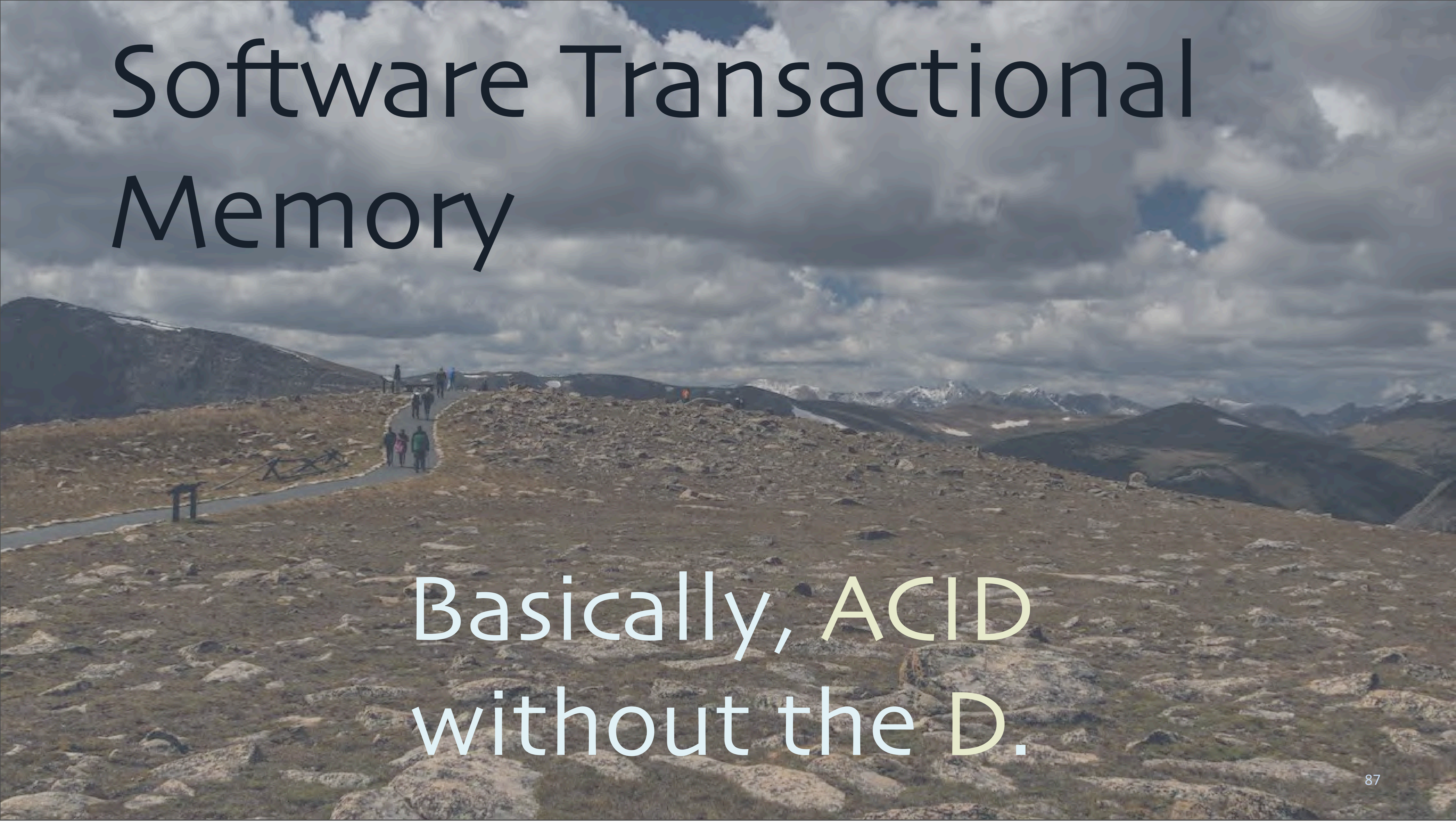
Messaging passing should be asynchronous. Any expensive calculation should be executed async, too, so main threads are not blocked. There are many lock-free algorithms and datastructures now. Locks kill scalability and they are hard to program correctly.



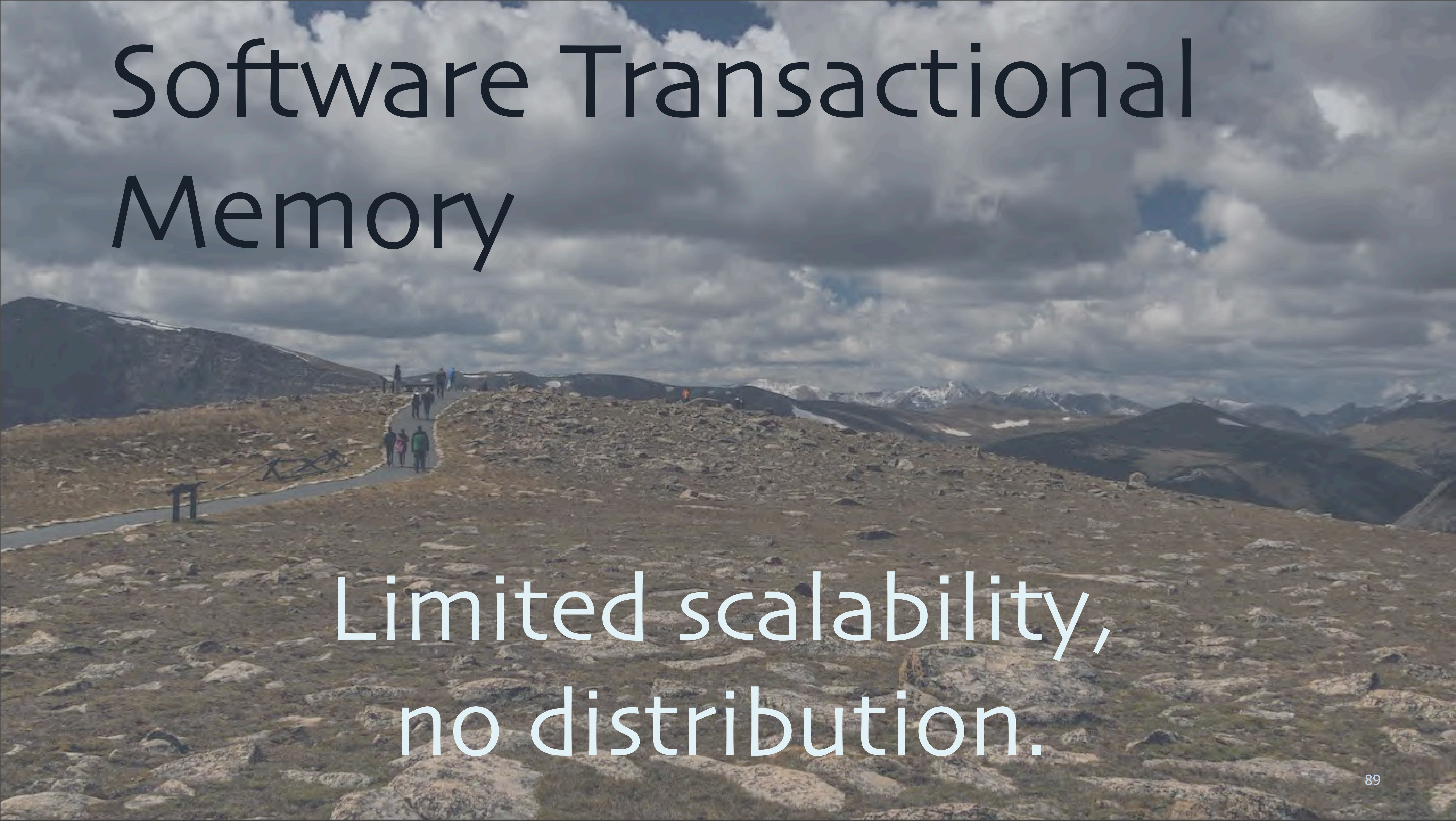
CRDTs - Commutative Replicated Data Types (http://pagesperso-systeme.lip6.fr/Marc.Shapiro/papers/RR-6956.pdf)
Lattices are more general concept applied here.

Wednesday, March 18, 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.
87


Software Transactional Memory
Principled local state mutation through transactions.
88



Wednesday, March 18, 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

Wednesday, March 18, 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.

Wednesday, March 18, 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

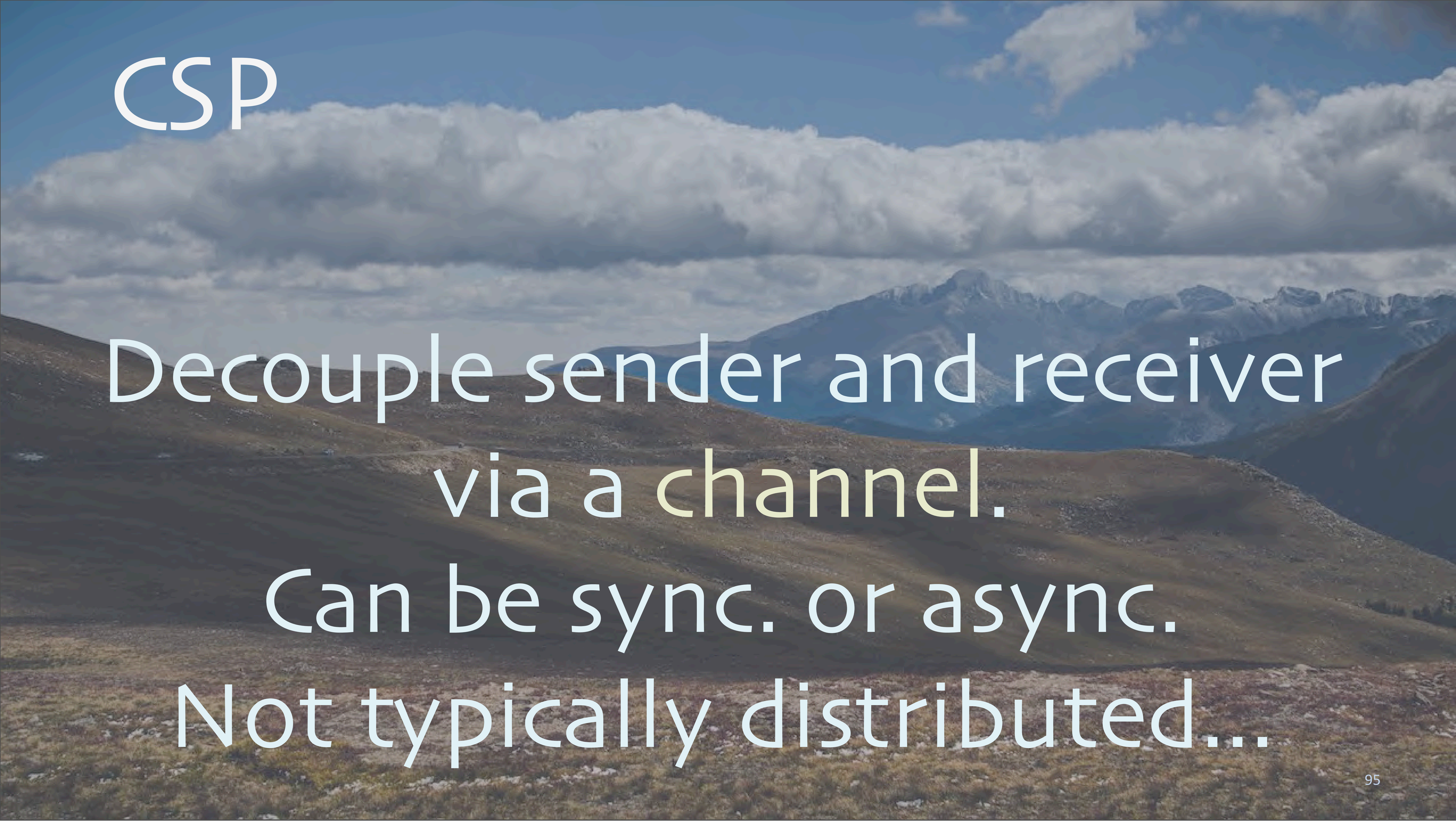
Callbacks invoked when an event is pushed to it.

Wednesday, March 18, 15

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

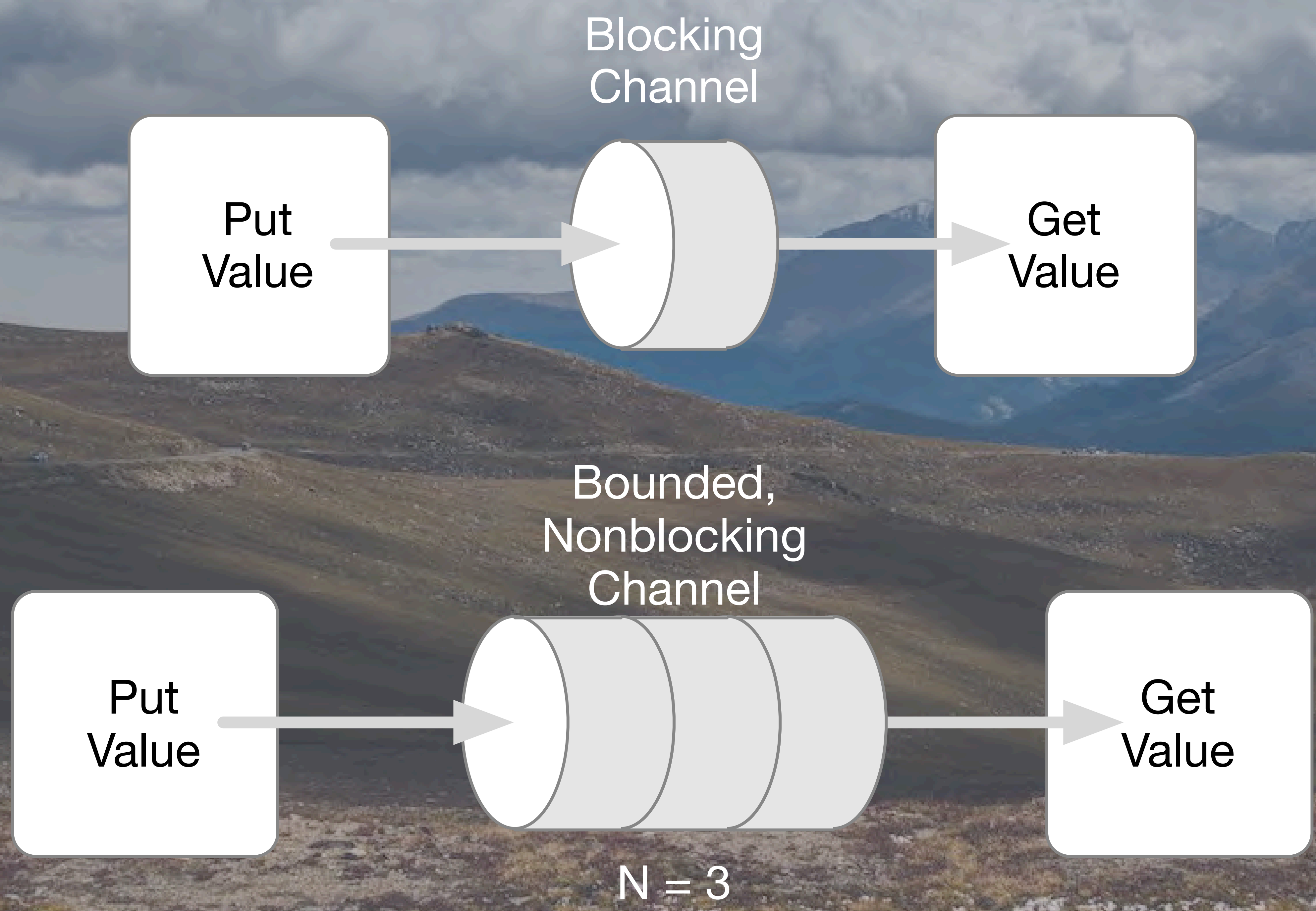
Many of these systems don’t provide facilities for distribution or error recovery.

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.



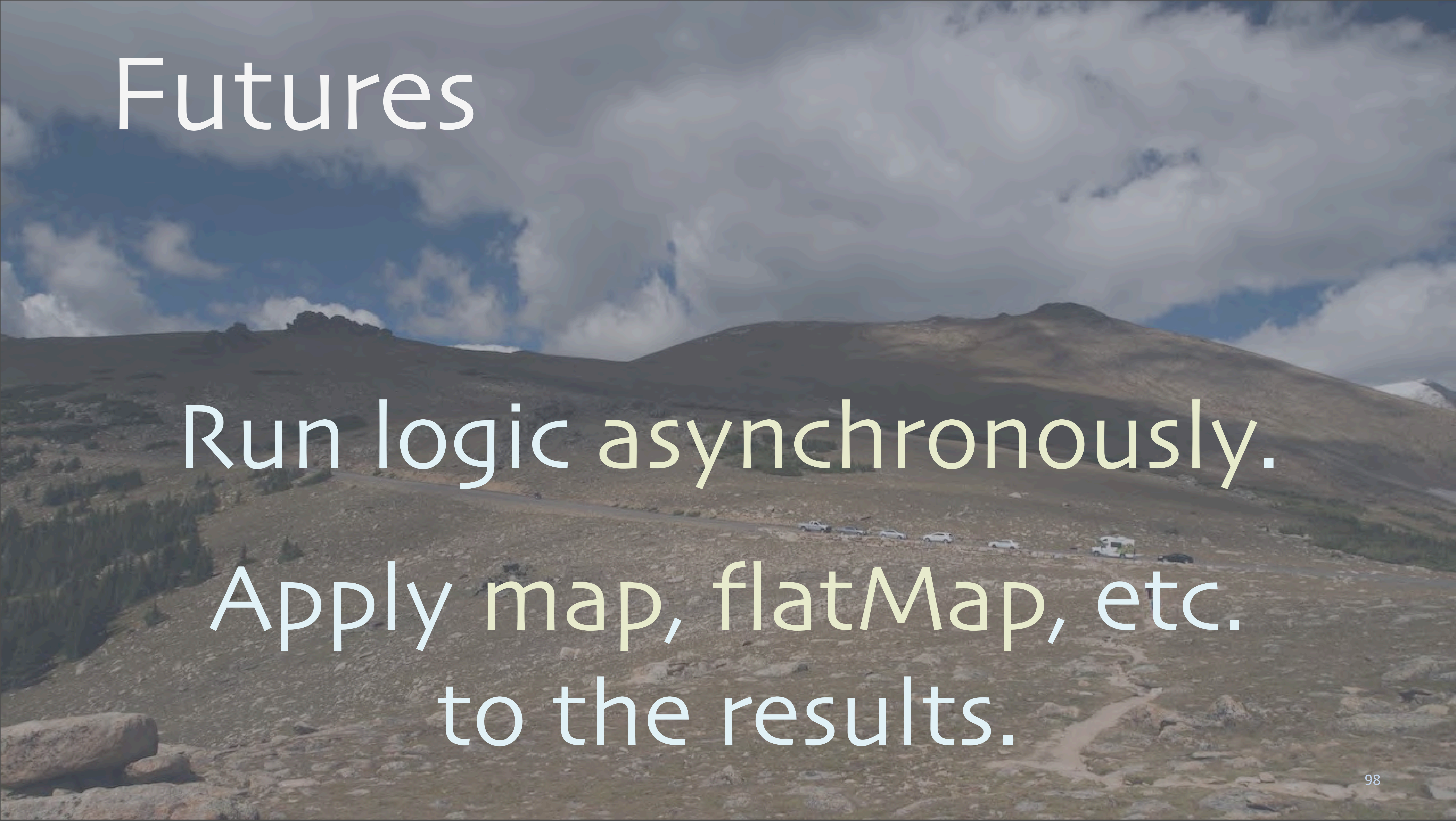
Great abstraction for coordinating exchange of data, with a level of abstraction, the channel. Not really a scalable distributed system in the general sense.

CSP



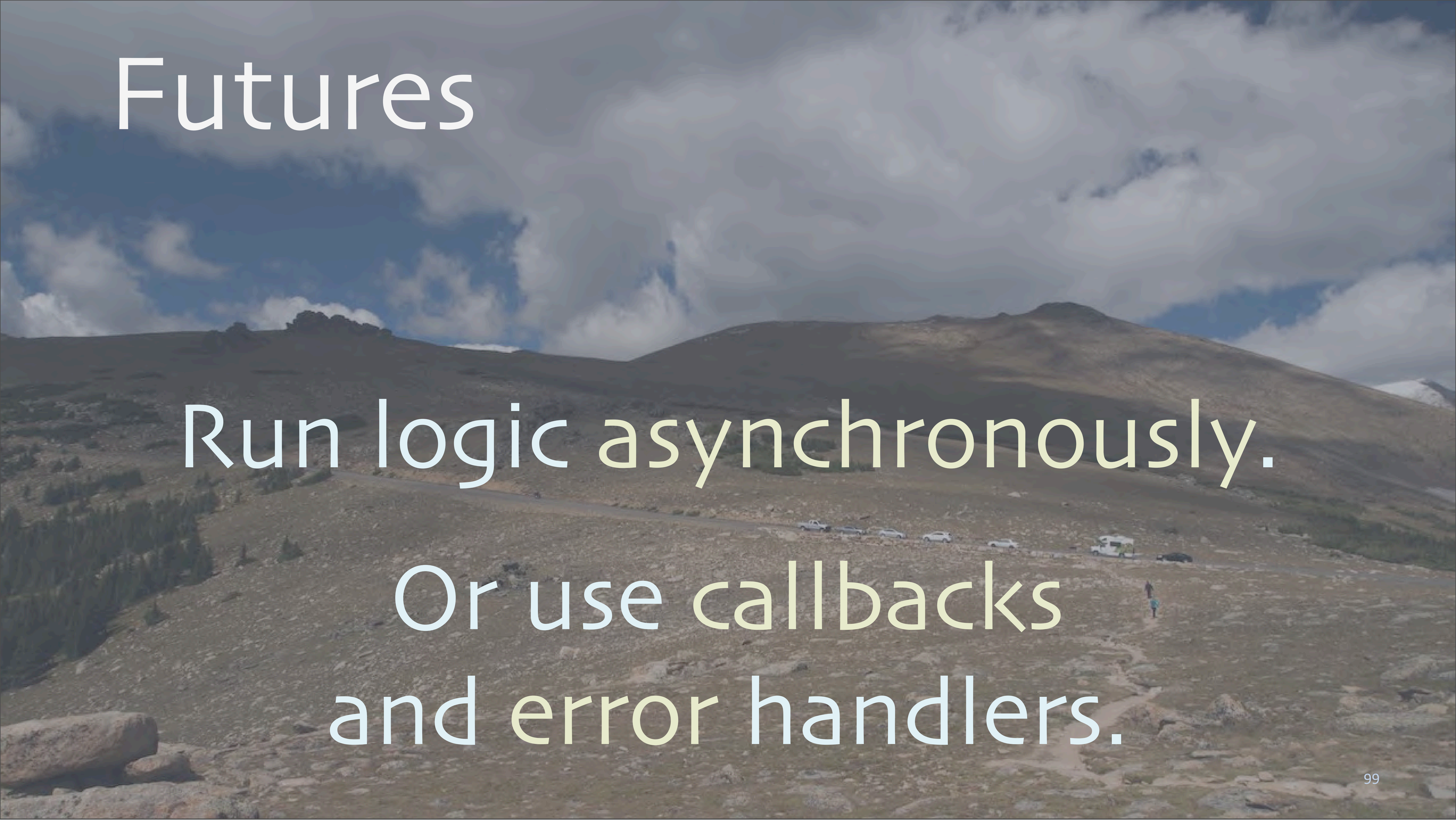
Schematic view of simple CSP interactions. If the channel queue has one slot, then it’s blocking; the “putter” wait for a “getter” to be on the other side. If there are >1 slots, the putter won’t block unless all the slots are full. See my talk on error handling in reactive systems where I discuss CSP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

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.



Wednesday, March 18, 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.

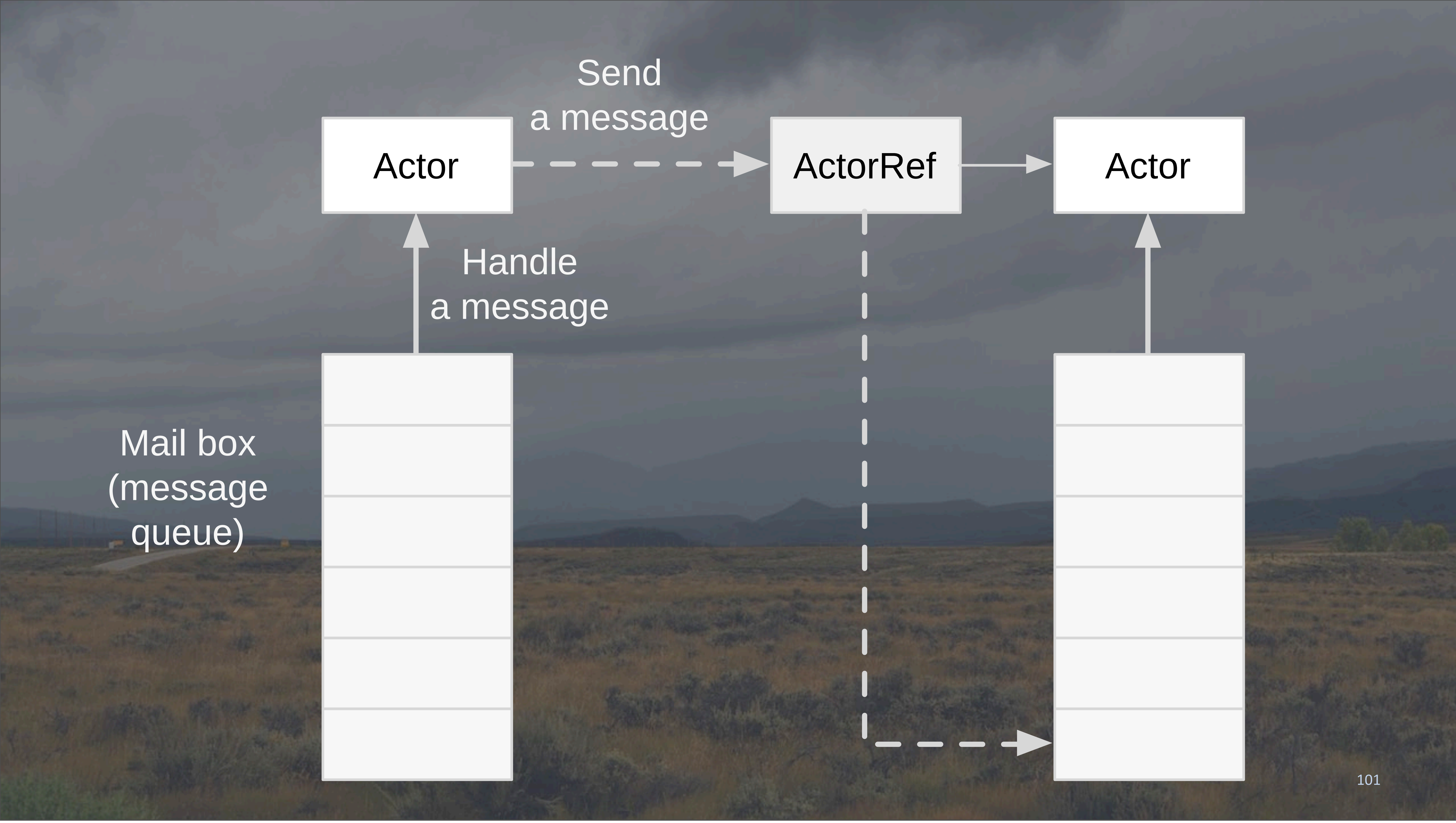


Wednesday, March 18, 15

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.

Wednesday, March 18, 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.



Wednesday, March 18, 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)

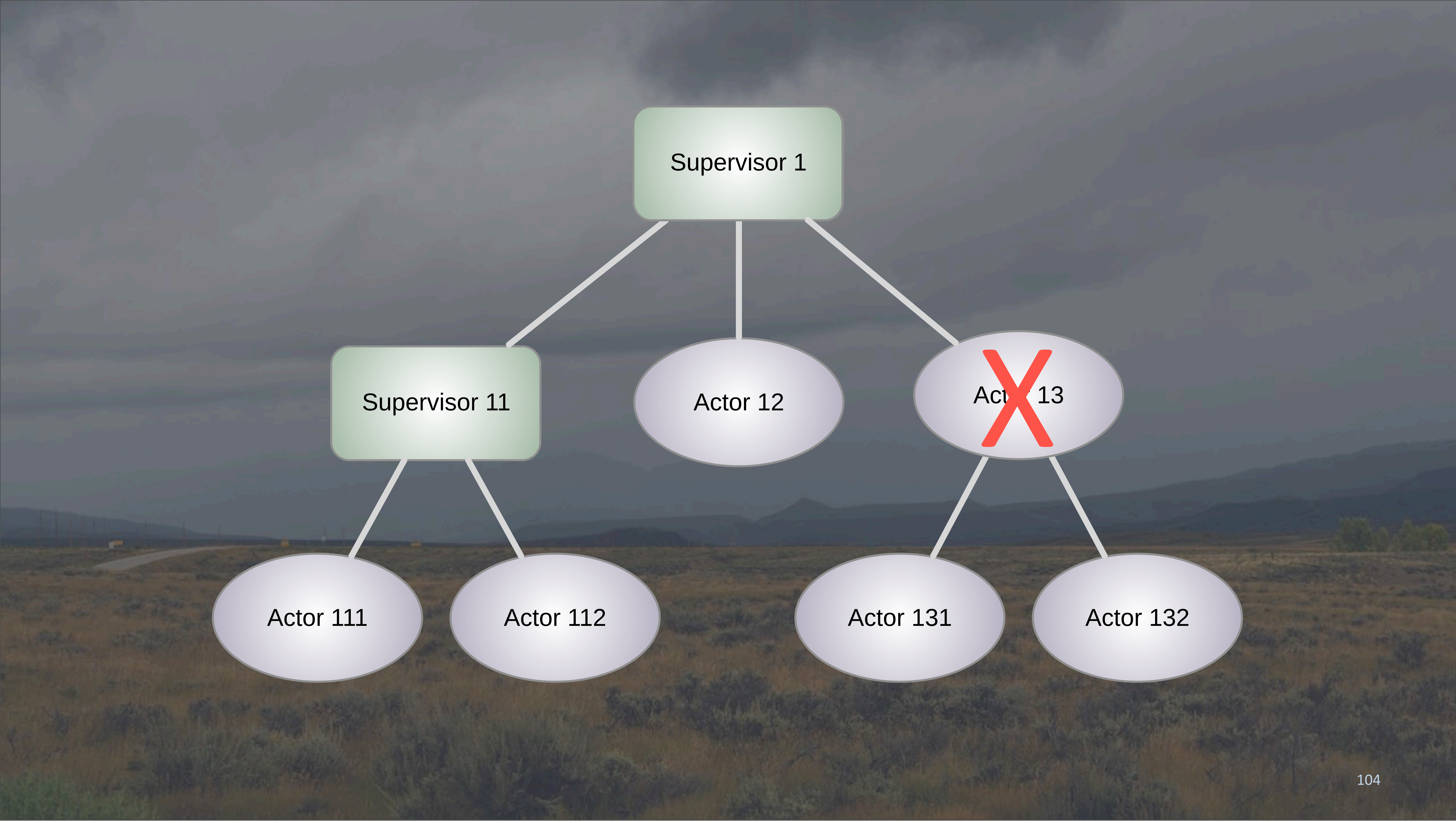
Actor Systems

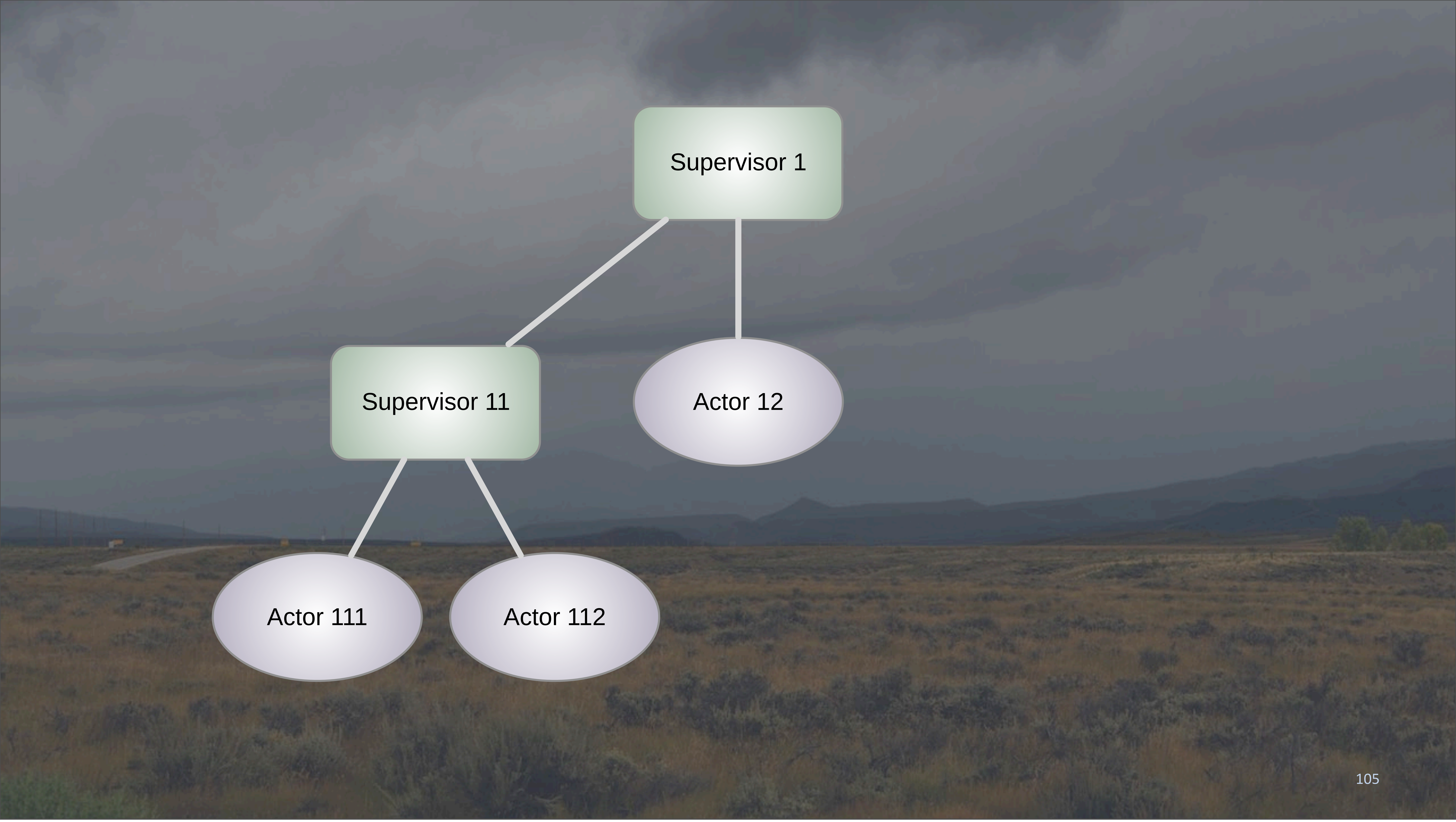
Pioneered by Hewitt, et al. 1973.
Made popular by Erlang,
which introduced Supervision.

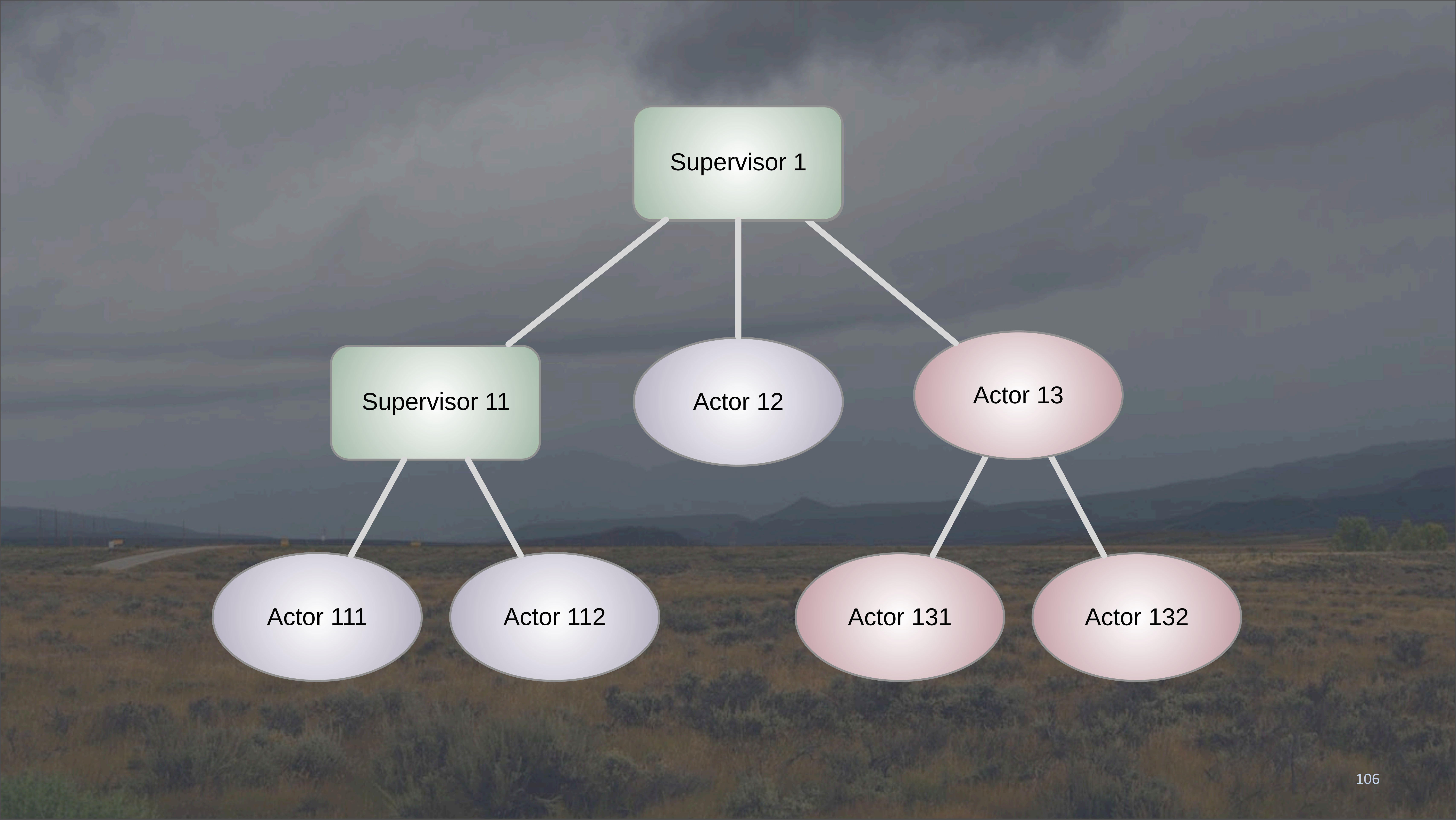
Wednesday, March 18, 15

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).

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).


Supervisor 1
Supervisor 11
Actor 12
Actor 111
Actor 112
Actor 131
Actor 132
104


Supervisor 1
Supervisor 11
Actor 12
Actor 111
Actor 112


Supervisor 1
Supervisor 11
Actor 12
Actor 13
Actor 111
Actor 112
Actor 131
Actor 132

Wednesday, March 18, 15

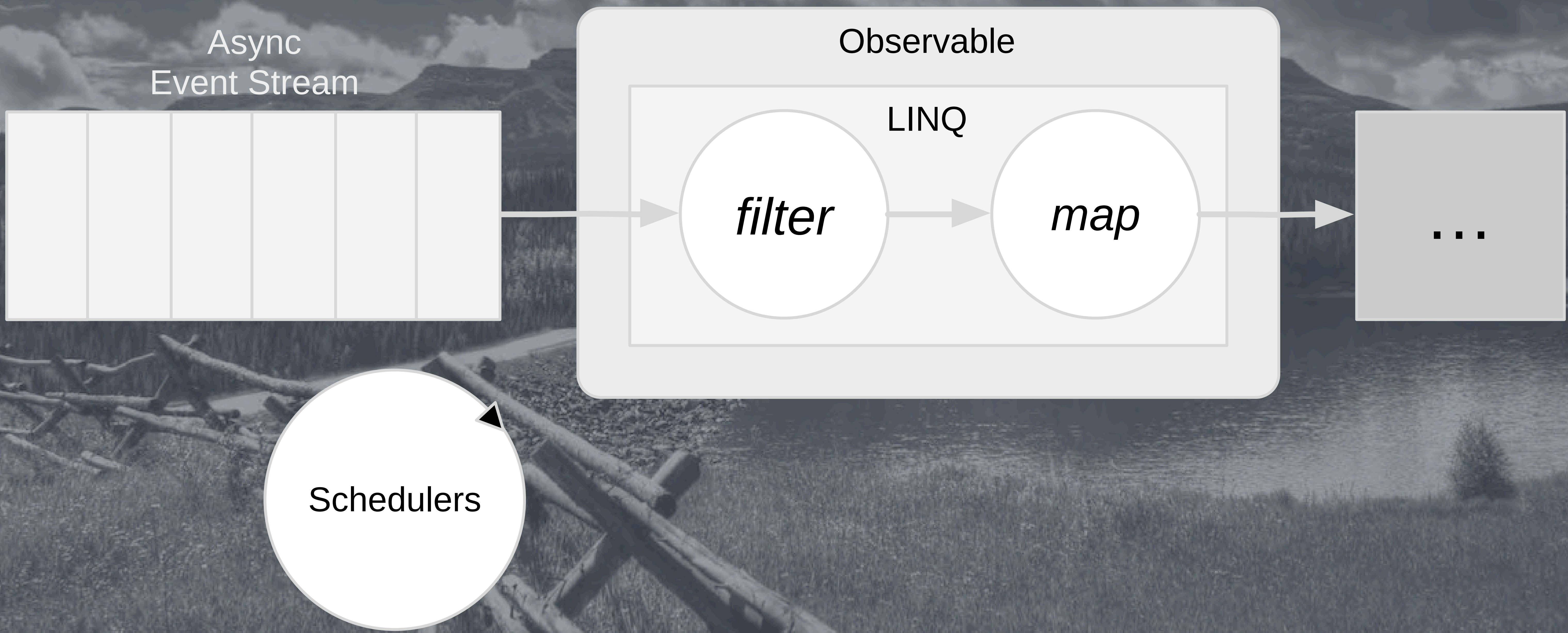
Again, I'll discuss this in some more depth in my other talk.

Clean separation of normal processing from recovery.

Wednesday, March 18, 15

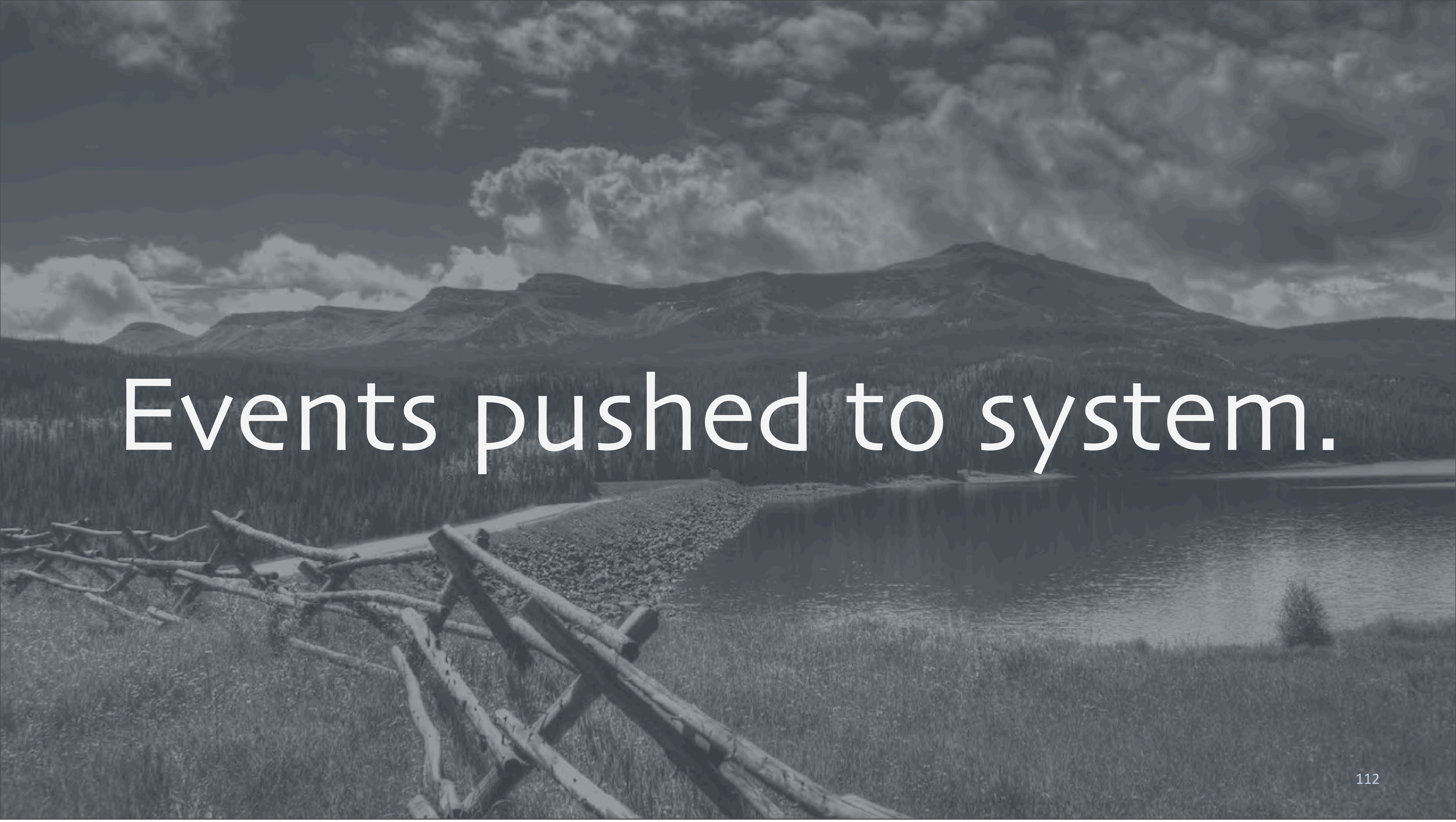
Pioneered by Erik Meijer for .NET. Now ported to several languages, including RxJava (Netflix) and React (JS - Facebook).

LINQ Rx



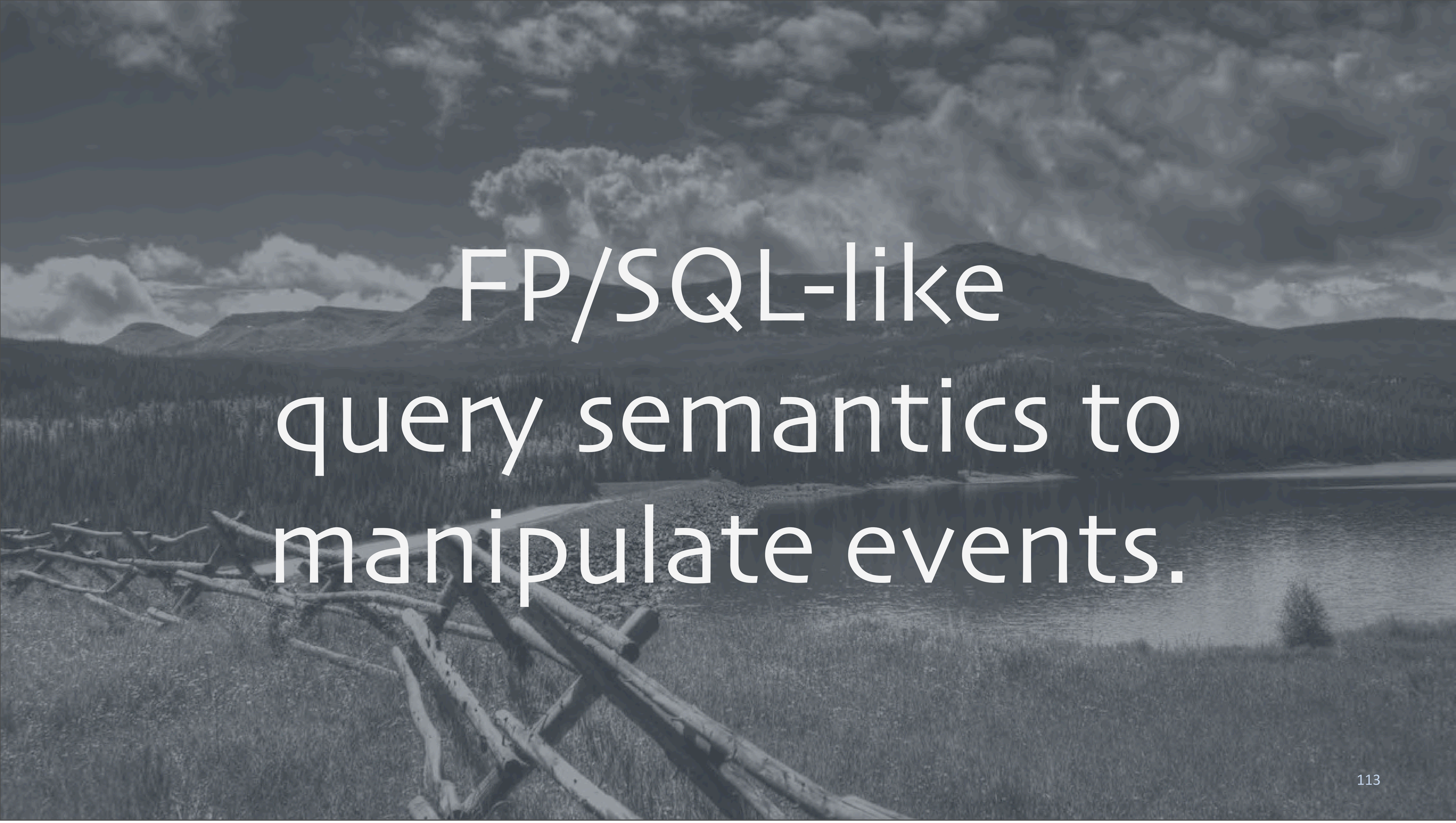
Wednesday, March 18, 15

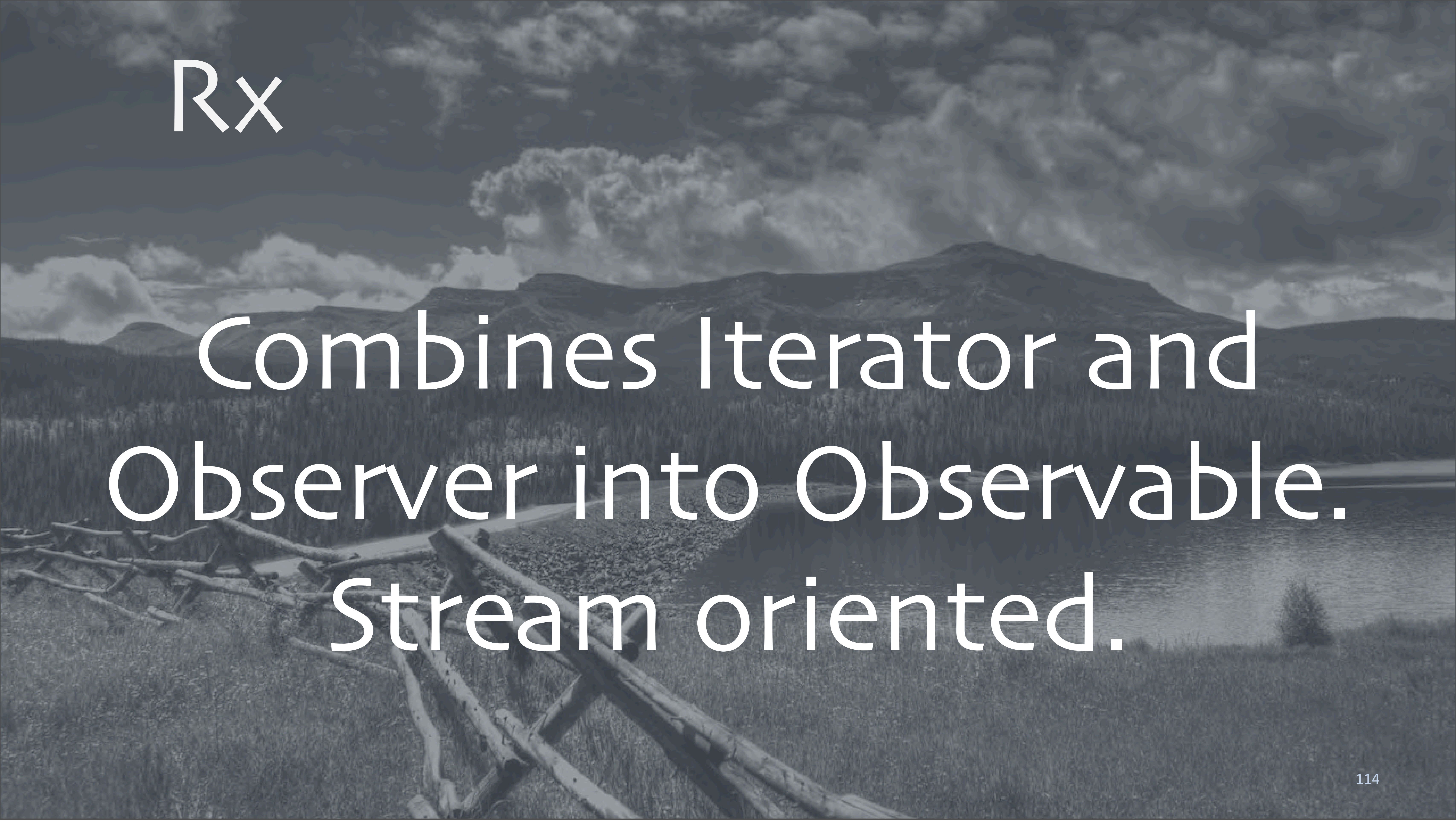
Events are observed (an extension of the observer pattern). Operations like filtering and mapping are provided to work with the stream through LINQ (Language Integrated Query), which uses SQL-like expressions. The Schedulers are used to trigger processing.



Wednesday, March 18, 15

It's essentially a push model.


FP/SQL-like
query semantics to
manipulate events.



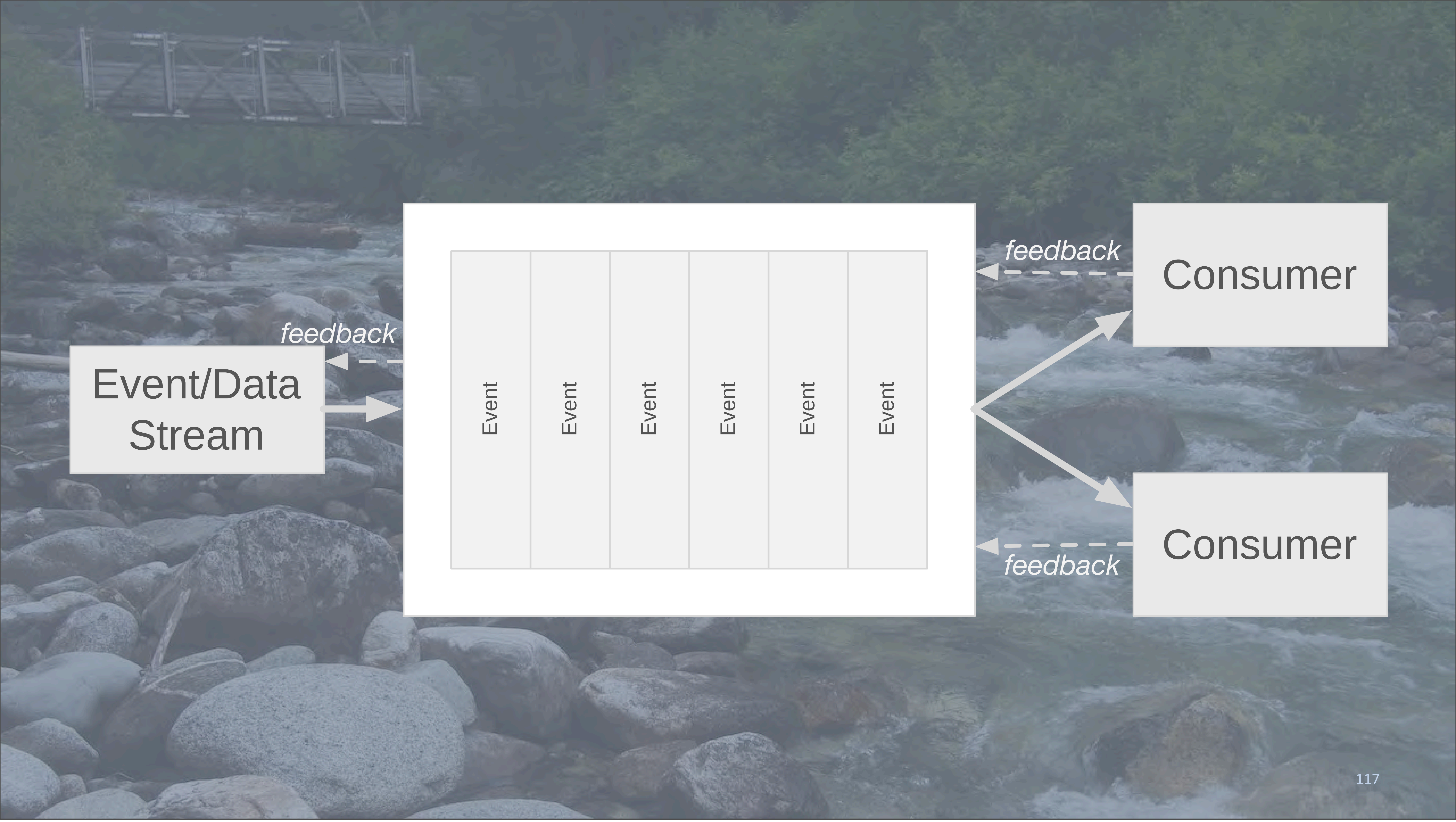
Wednesday, March 18, 15

I didn’t mention this before.

Rx

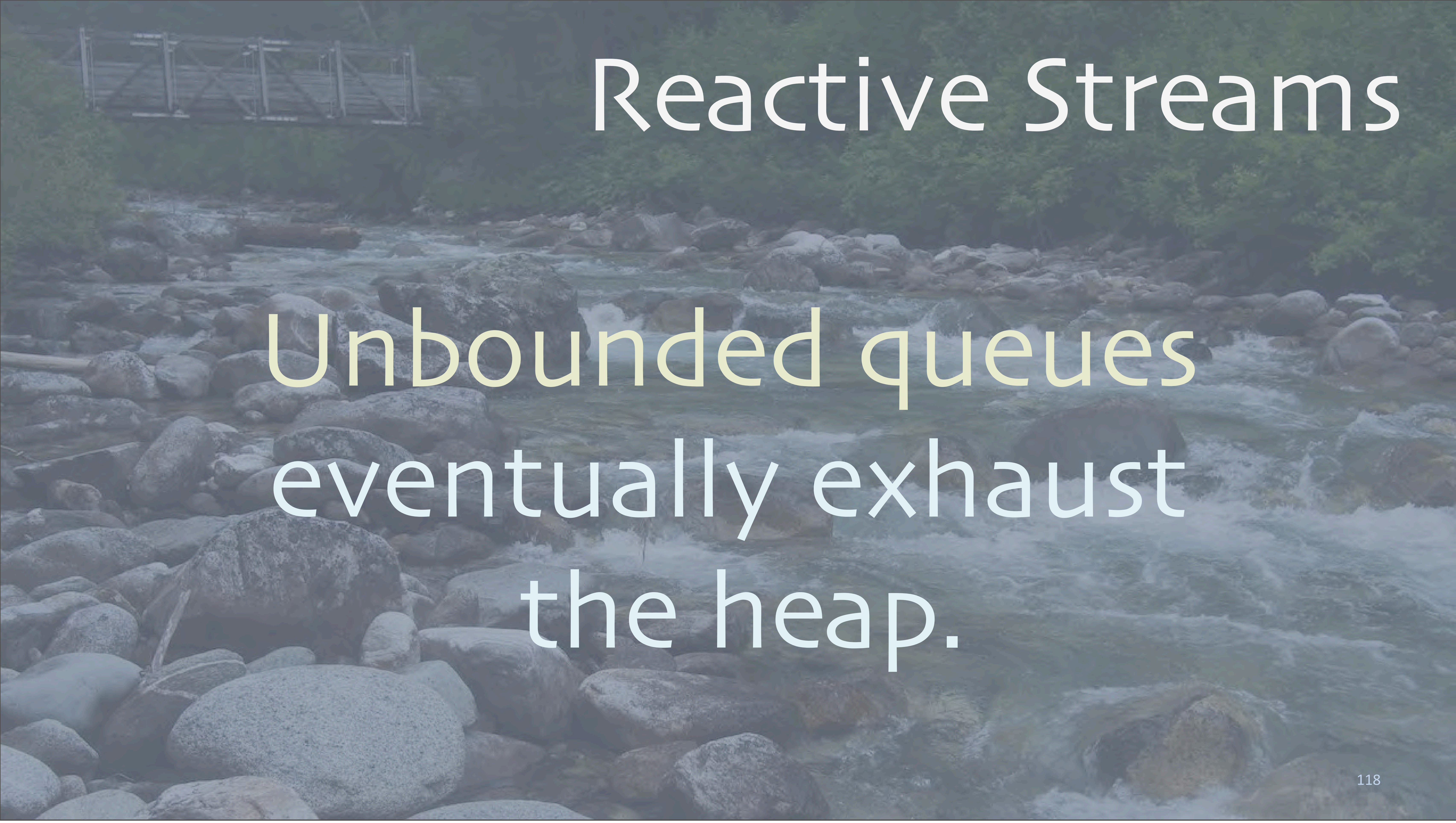
Need to add your own fault-tolerance model.

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



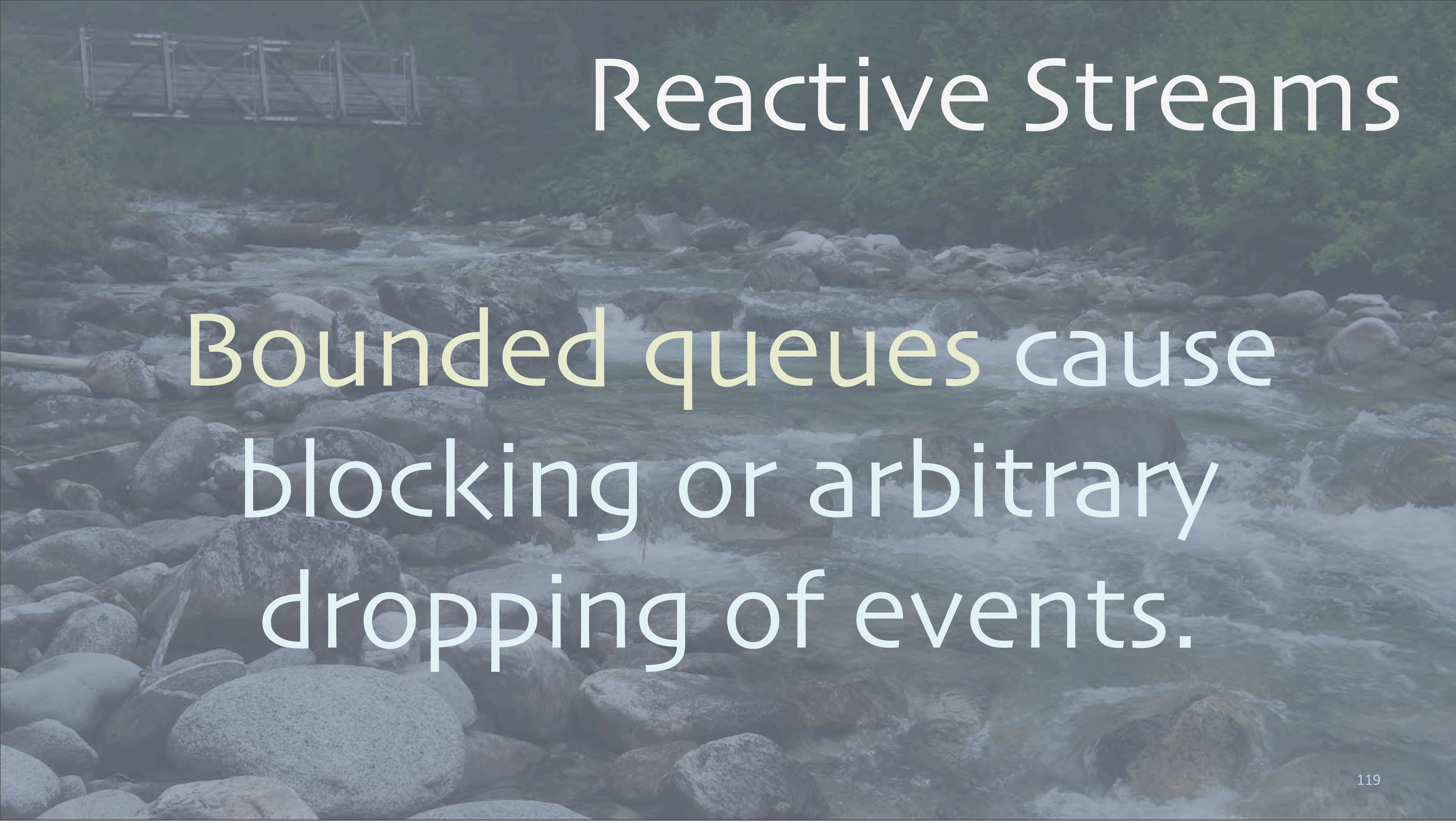
Wednesday, March 18, 15

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.



Wednesday, March 18, 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.



Wednesday, March 18, 15

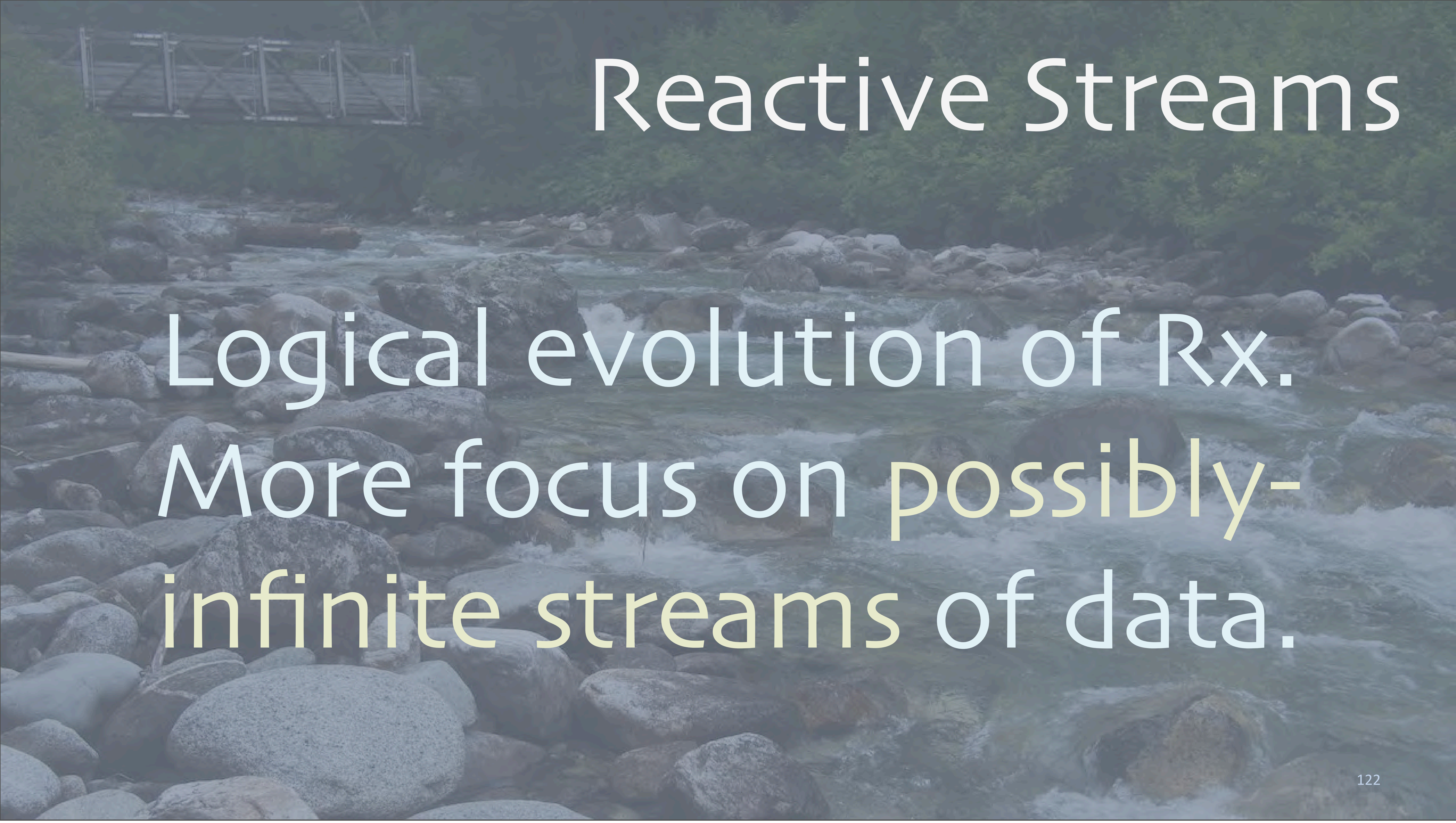
Bounded queues avoid heap exhaustion, but force arbitrary dropping of events or blocking.

Wednesday, March 18, 15

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

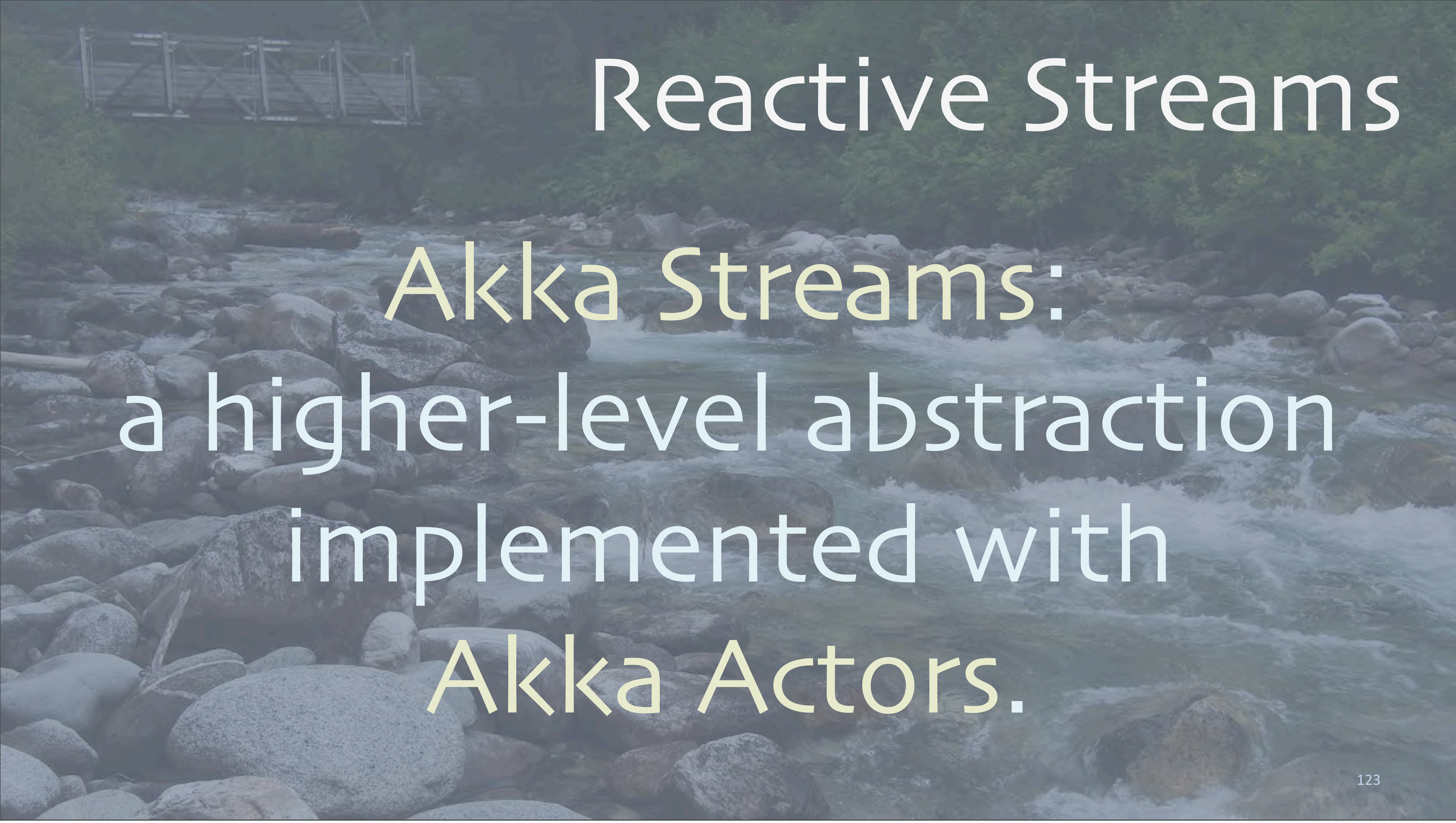
Wednesday, March 18, 15

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.



Wednesday, March 18, 15

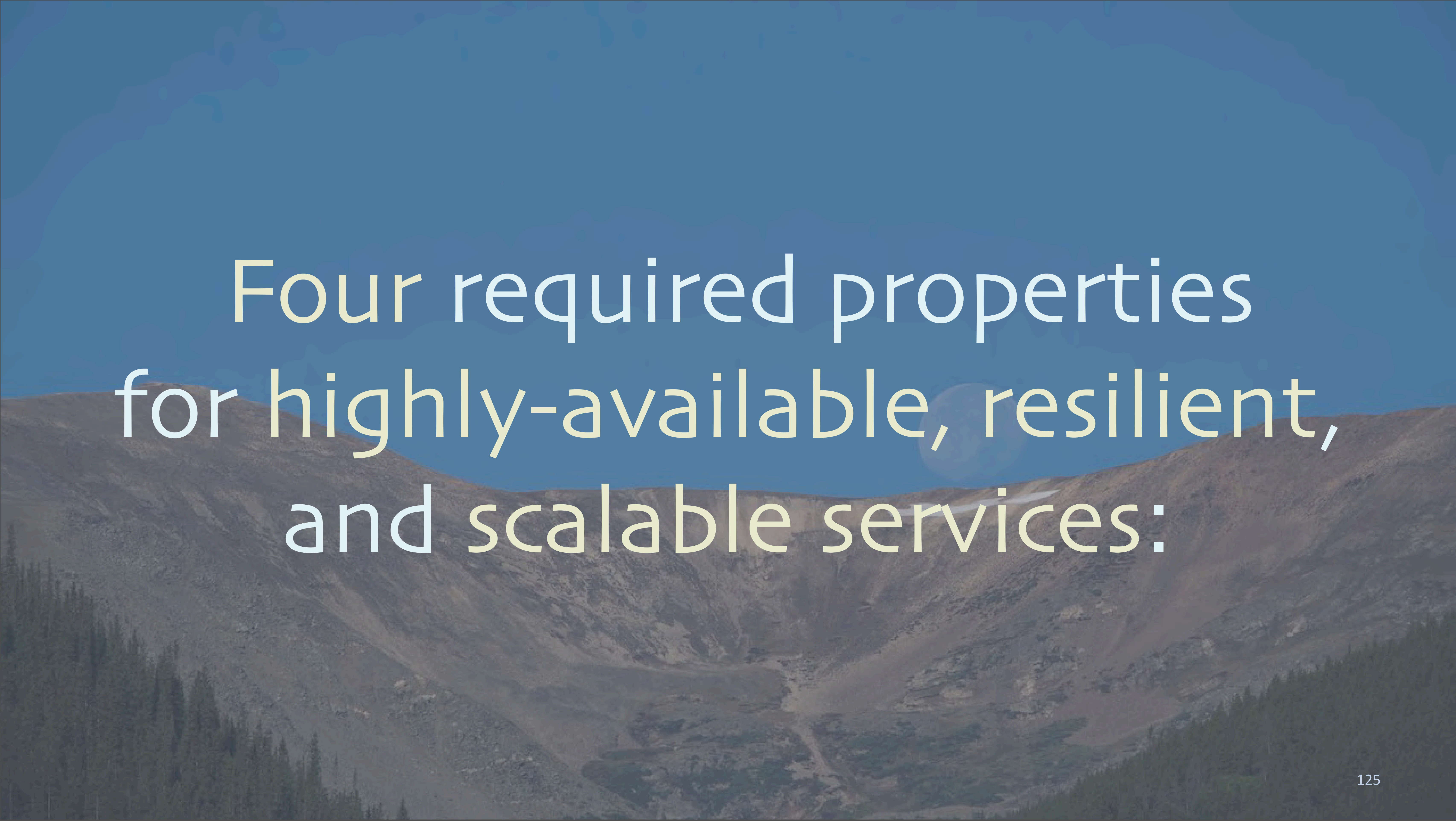
Like Rx, RS uses functional transformations to manipulate the data. It puts slightly more emphasis on the idea of streams (possibly infinite) rather than an event loop.

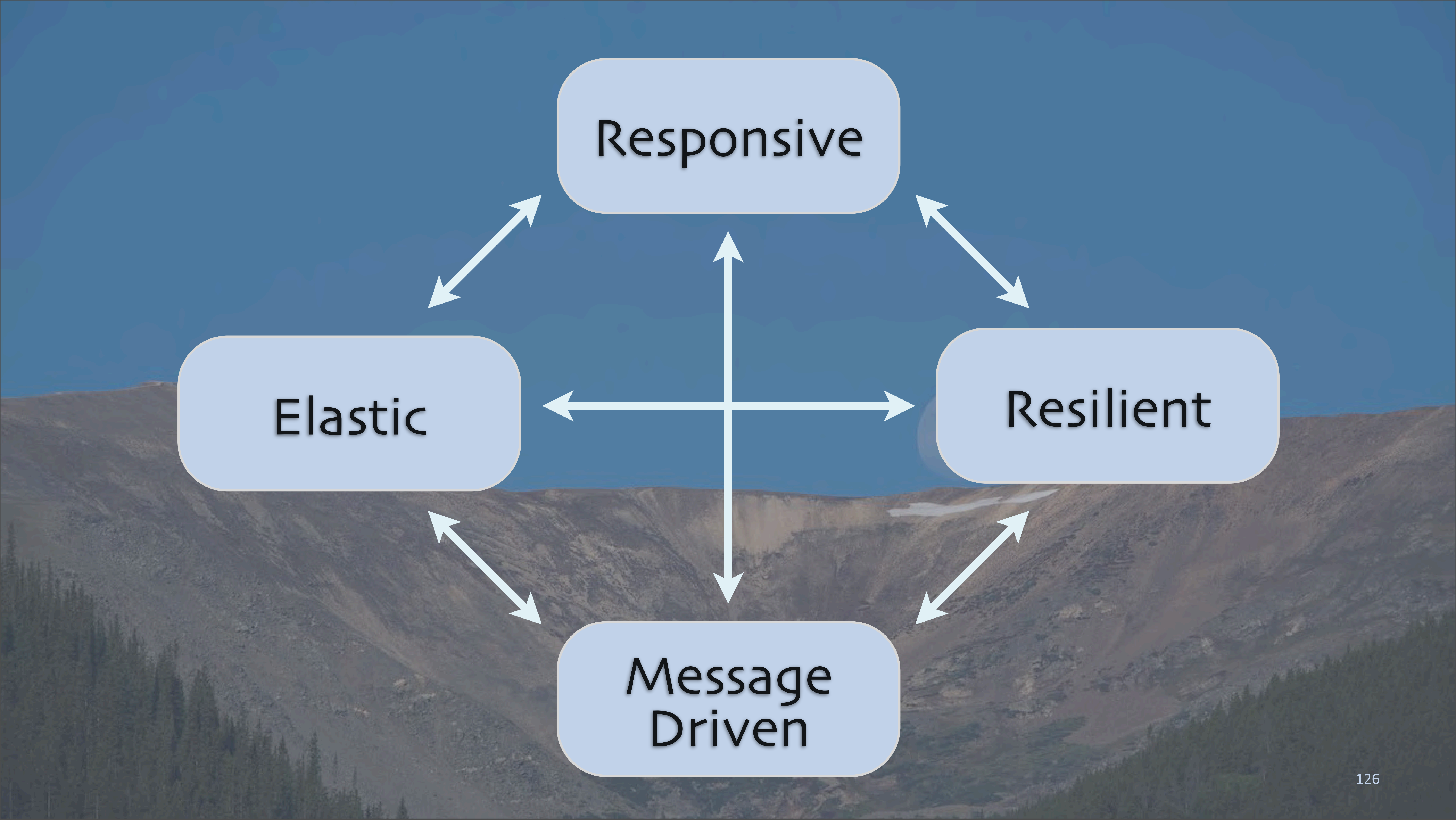


Wednesday, March 18, 15

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

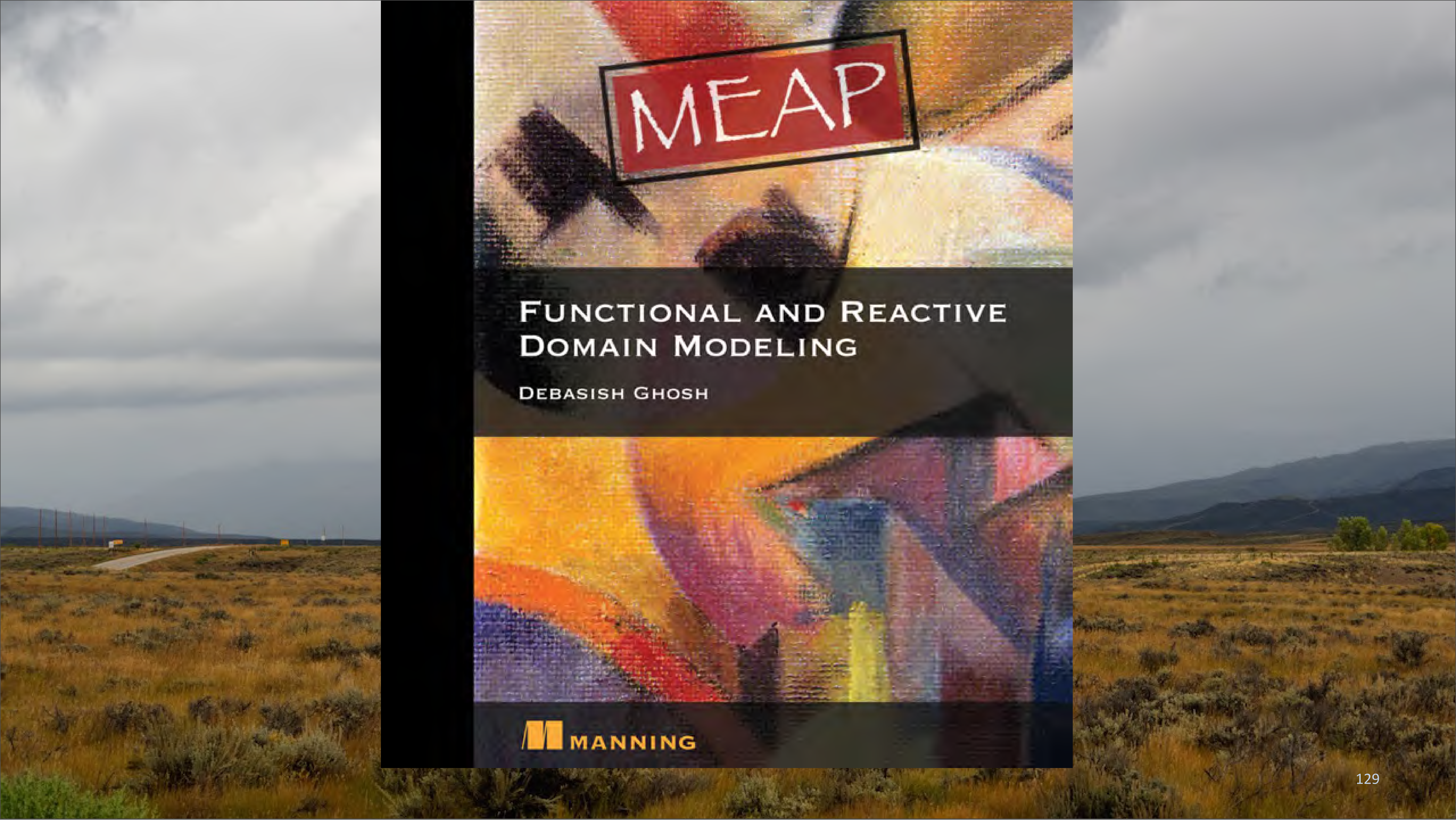

Four required properties
for highly-available, resilient,
and scalable services:
125


Responsive
Elastic
Resilient
Message Driven

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

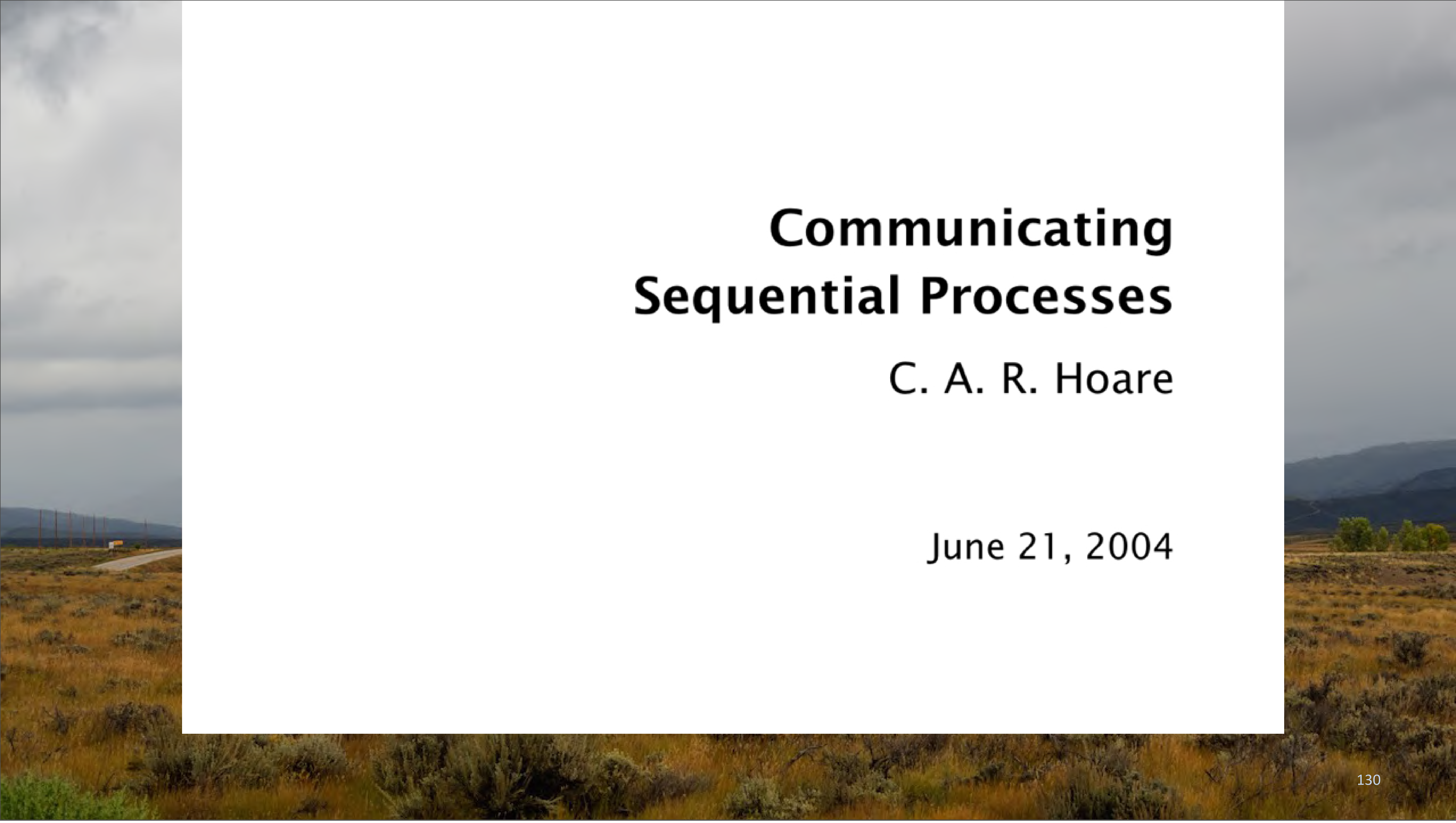
Wednesday, March 18, 15

See also links earlier in the presentation.



Wednesday, March 18, 15

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



Wednesday, March 18, 15

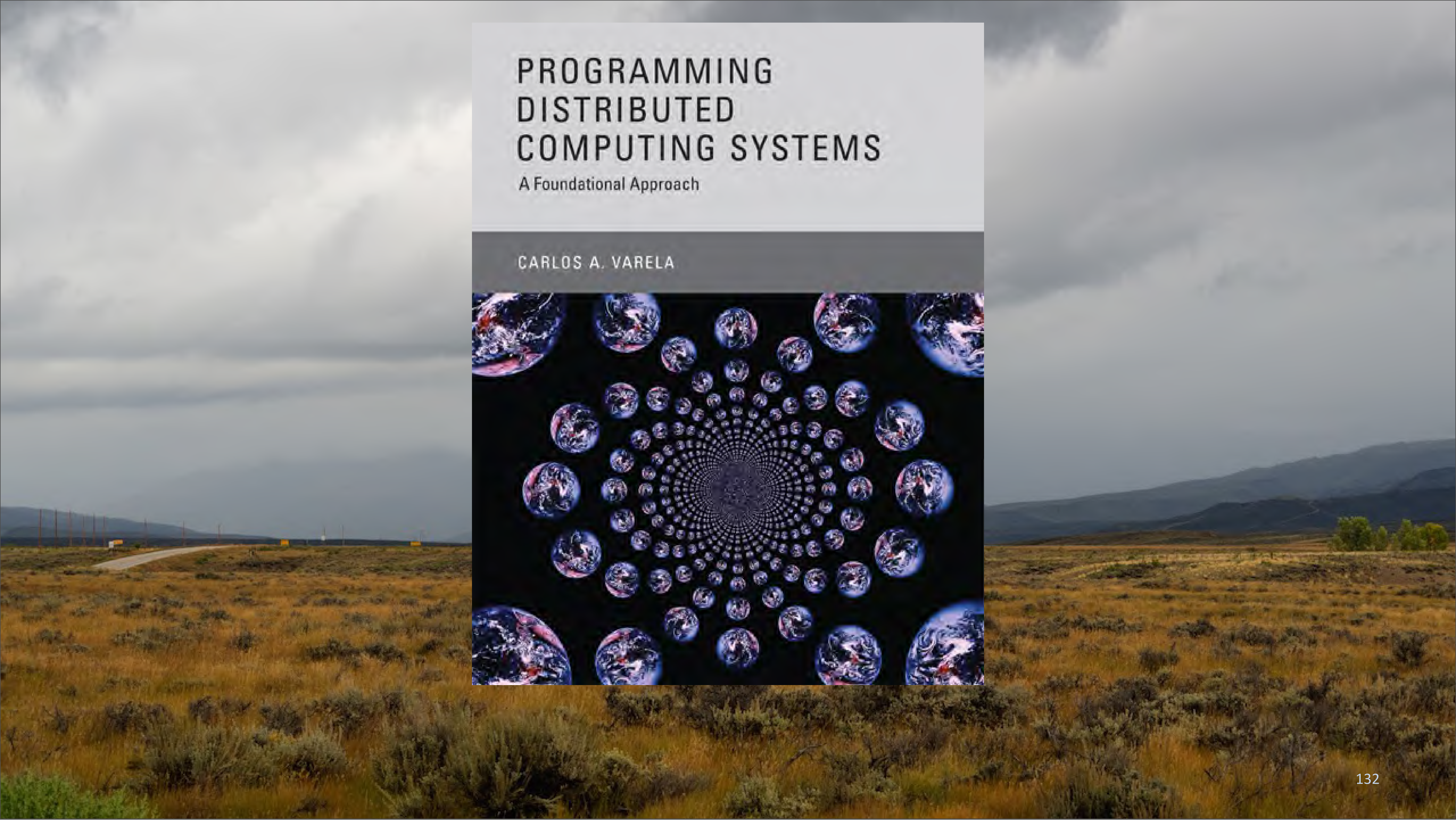
Hoare’s book on CSP, originally published in ’85 after CSP had been significantly evolved from a programming language to a theoretical model with a well-defined calculus. The book itself has been subsequently refined. The PDF is available for free.

The Theory and Practice of Concurrency

A.W. Roscoe

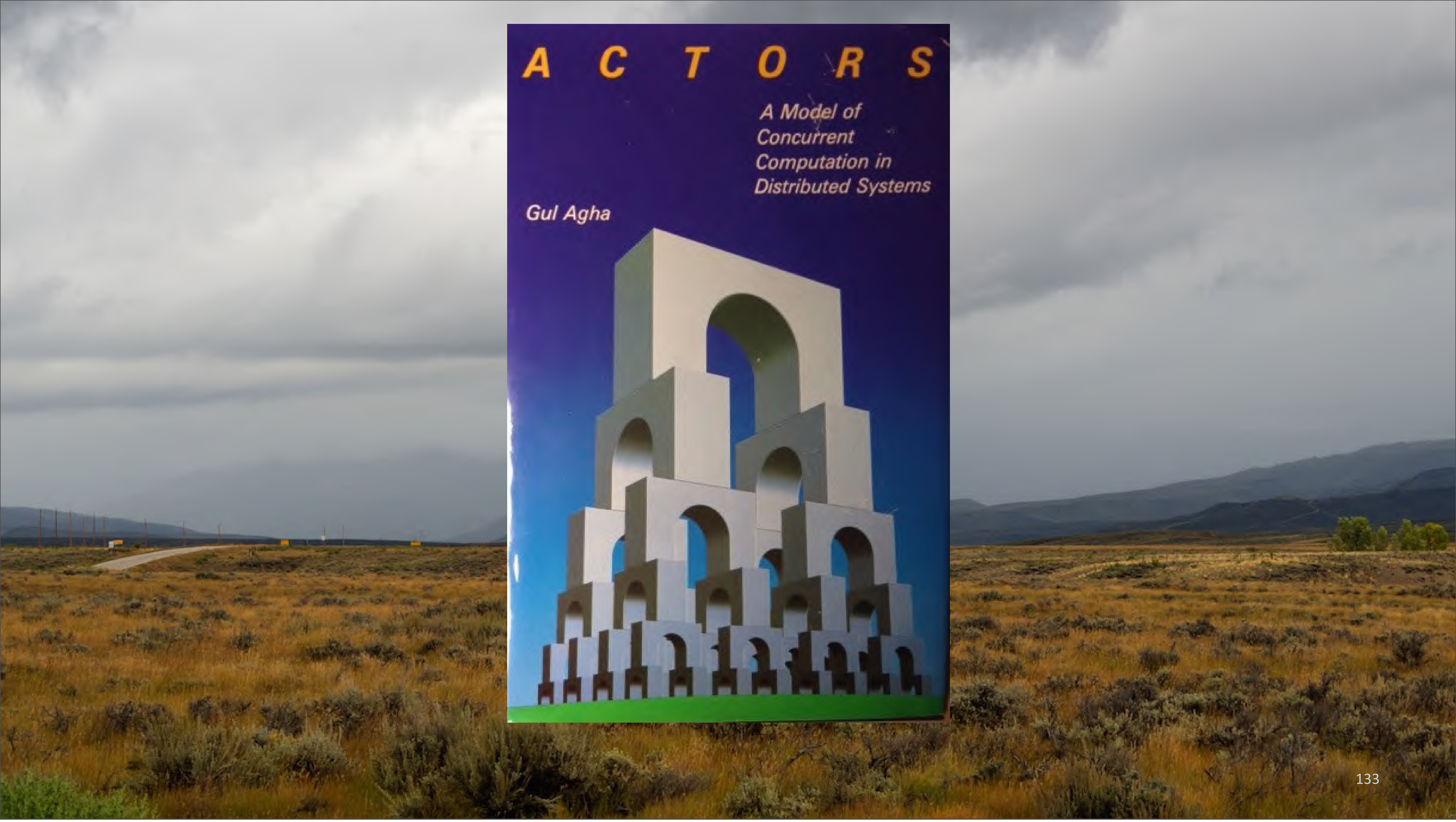
Published 1997, revised to 2000 and lightly revised to 2005.

Modern treatment of CSP. Roscoe helped transform the original CSP language into its more rigorous, process algebra form, which was influenced by Milner's Calculus of Communicating Systems work. This book's PDF is available free. The treatment is more accessible than Hoare's book.



Wednesday, March 18, 15

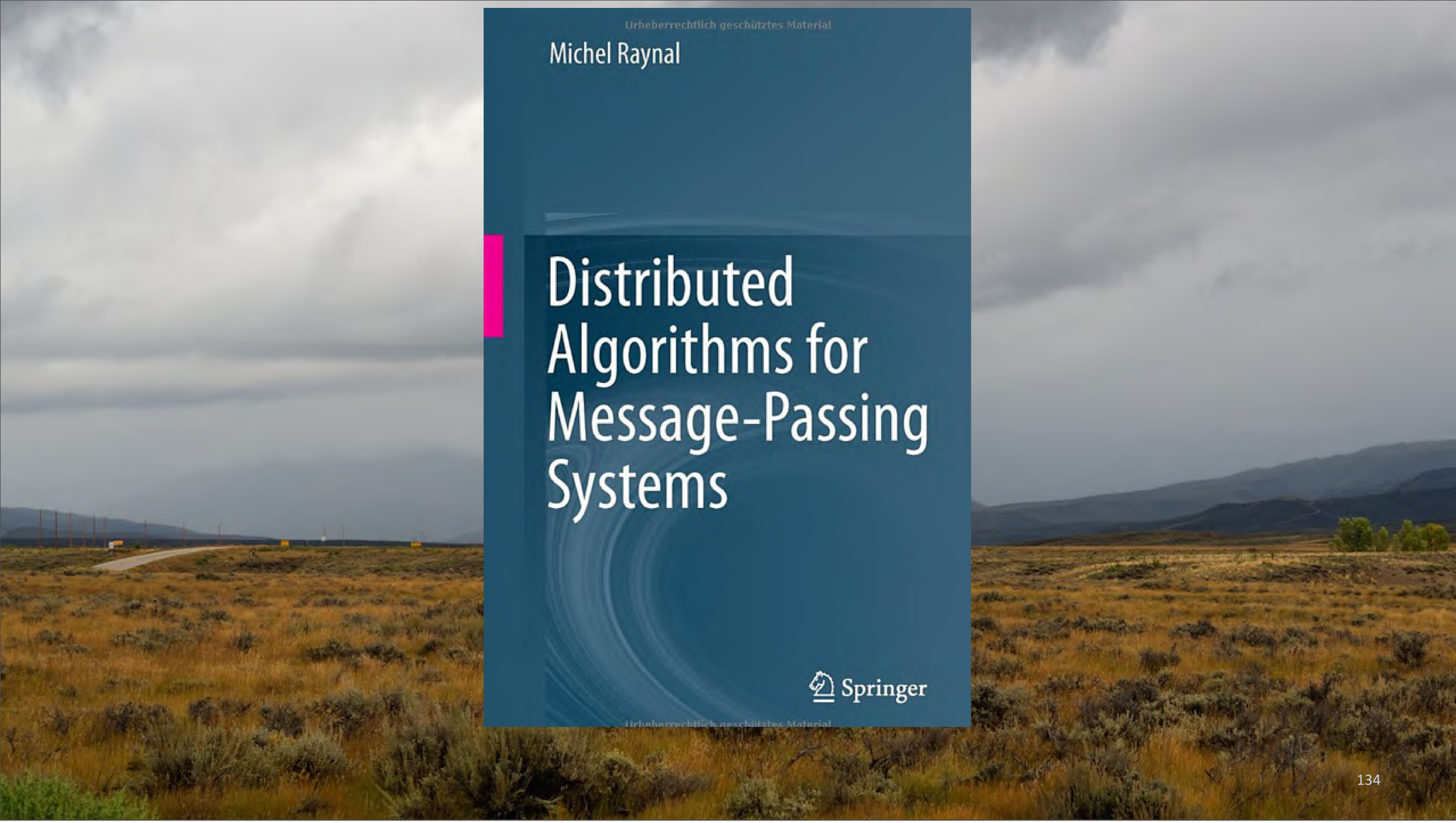
A survey of theoretical models of distributed computing, starting with a summary of lambda calculus, then discussing the pi, join, and ambient calculi. Also discusses the actor model. The treatment is somewhat dry and could use more discussion of real-world implementations of these ideas, such as the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



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 .

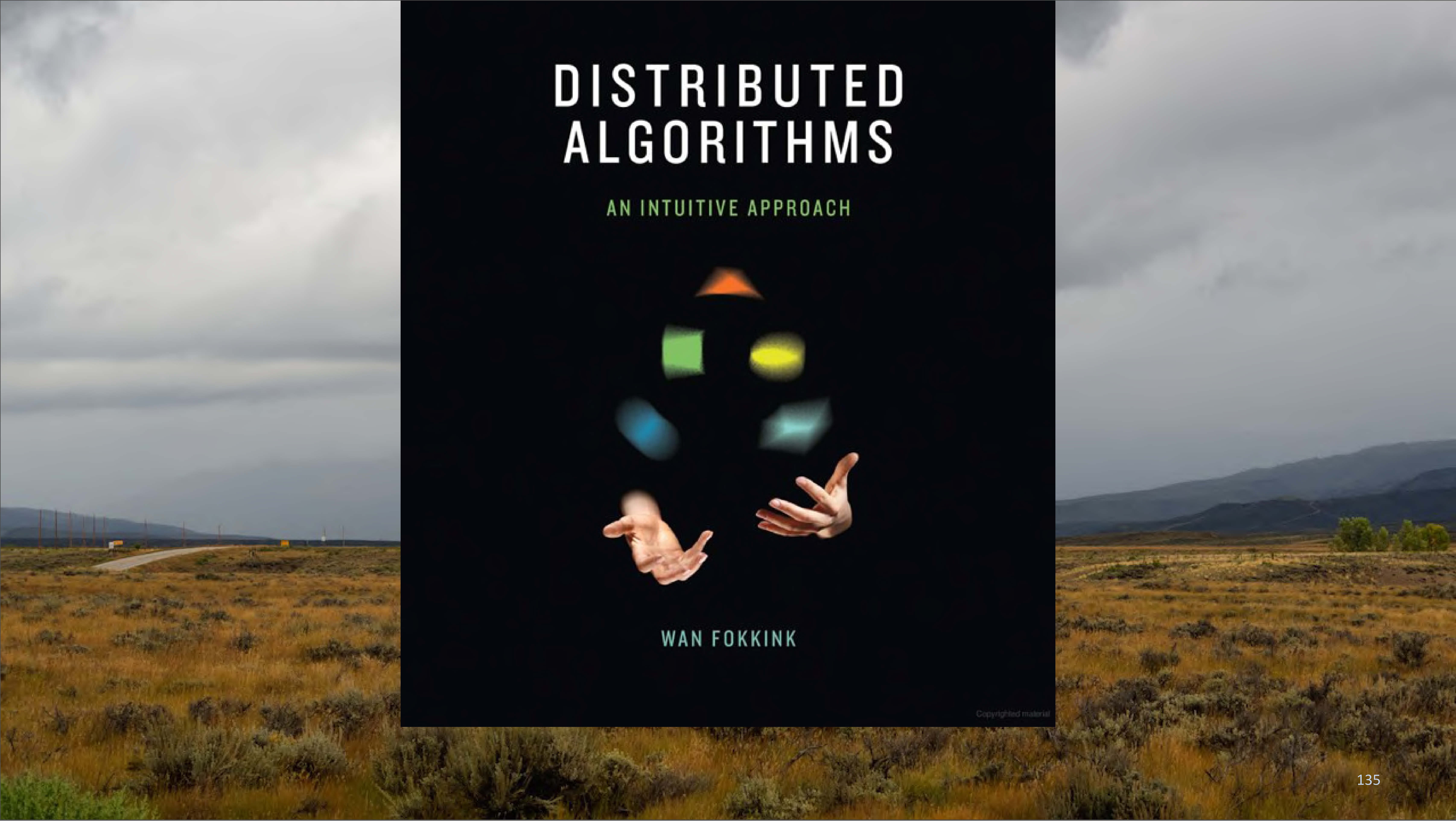
His thesis, http://dspace.mit.edu/handle/1721.1/6952, the basis for his book,http://mitpress.mit.edu/books/actors

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

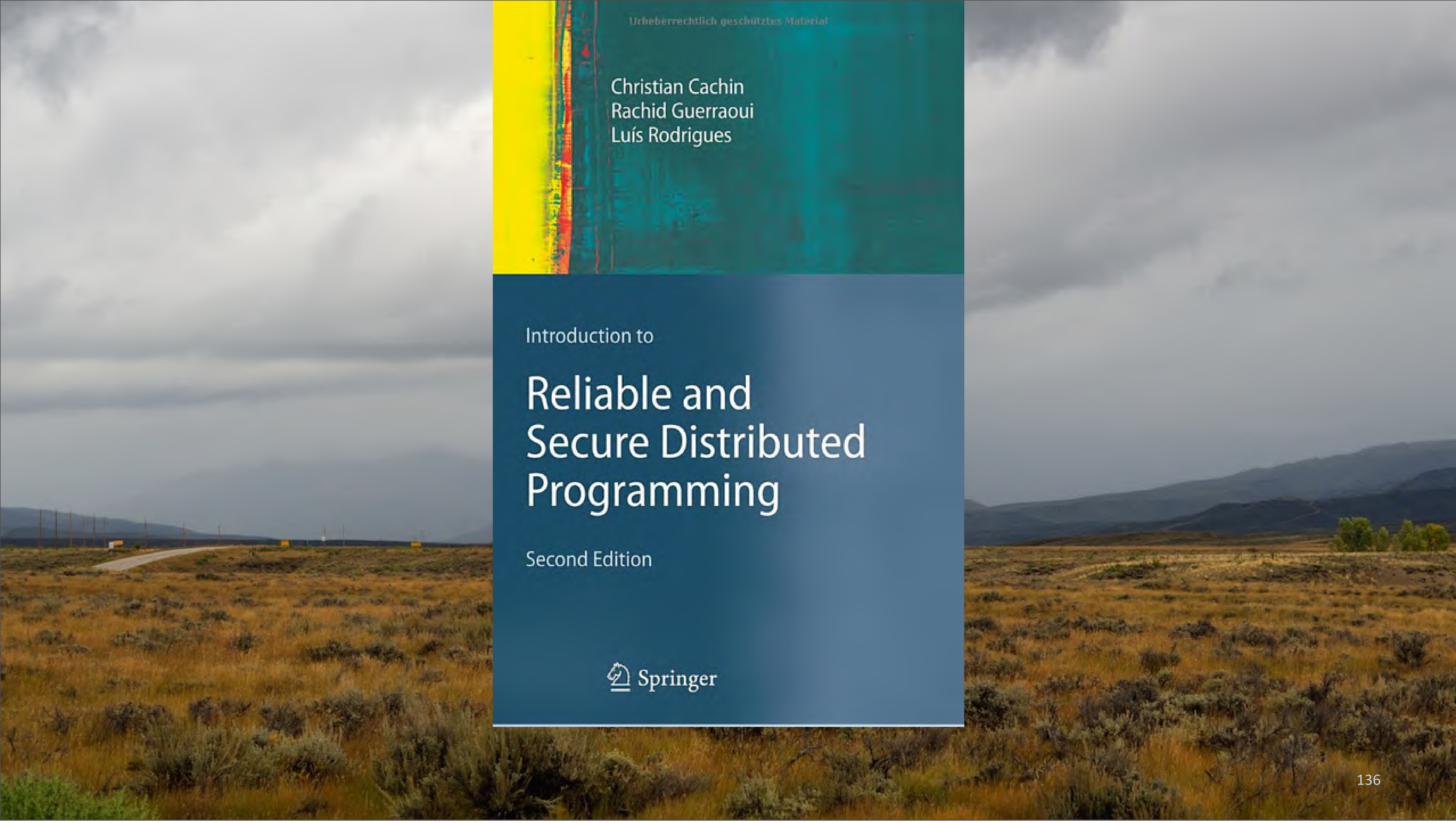


Wednesday, March 18, 15

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

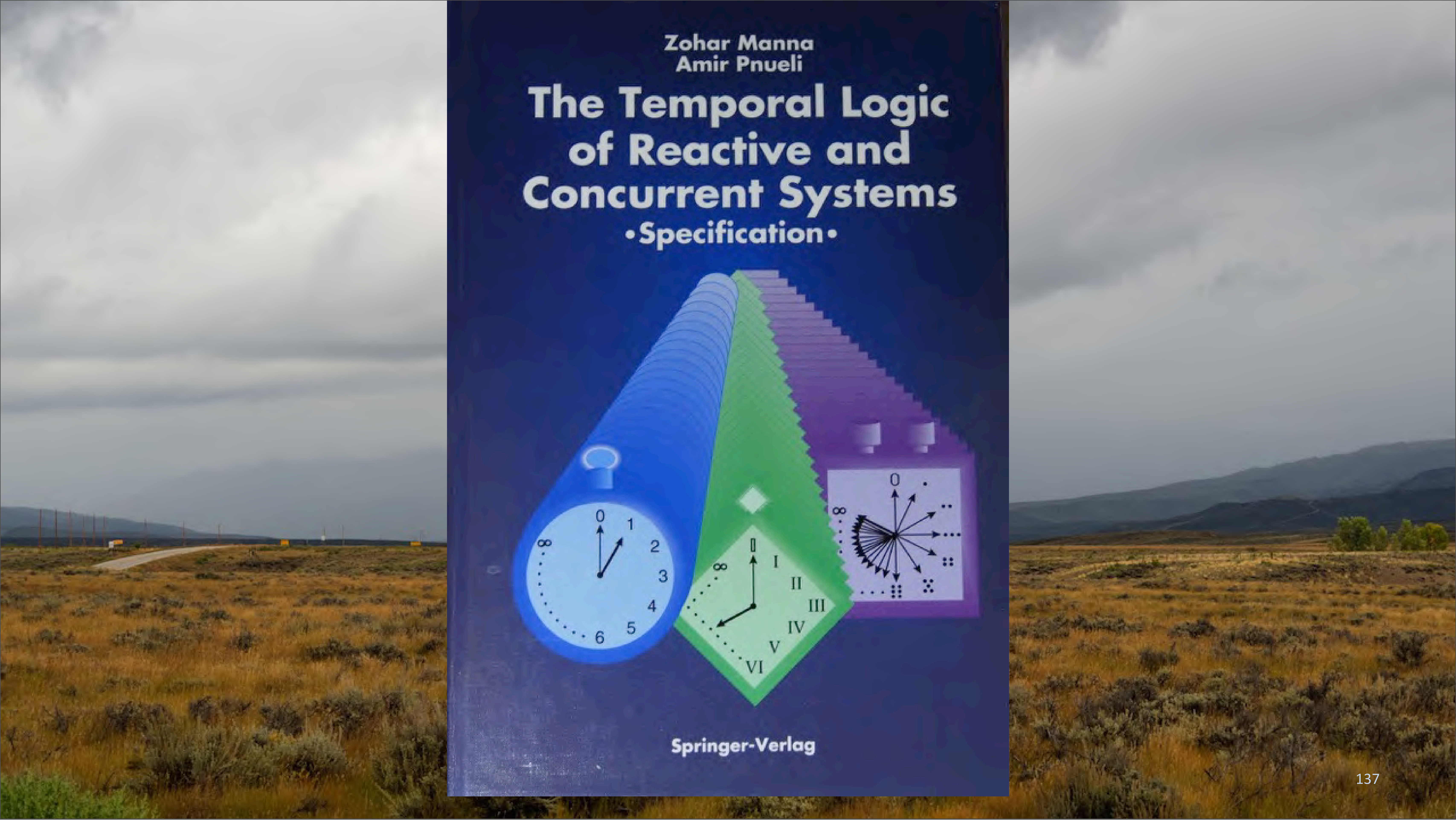


A less comprehensive and formal, but more intuitive approach to fundamental algorithms.

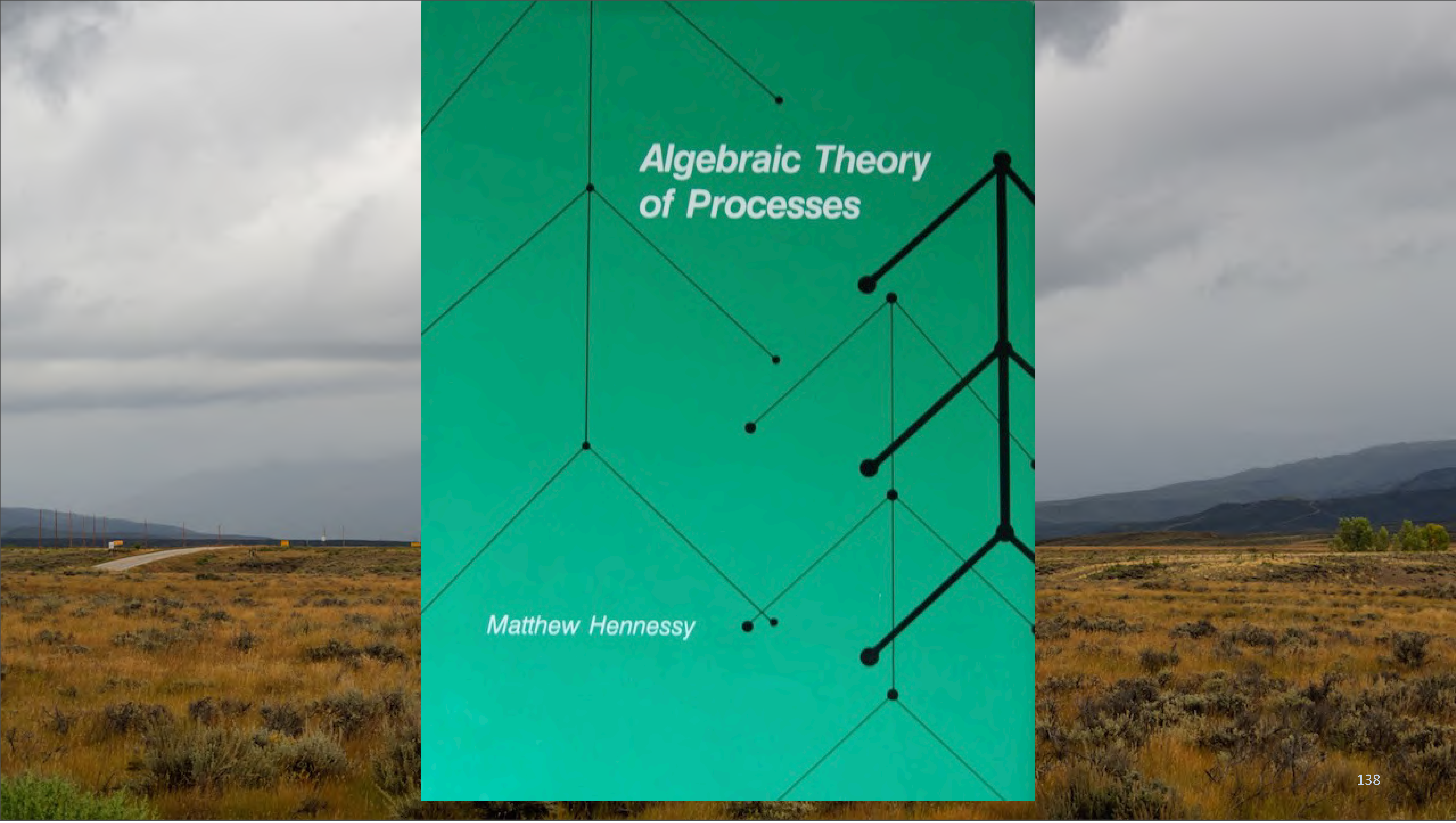


Wednesday, March 18, 15

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

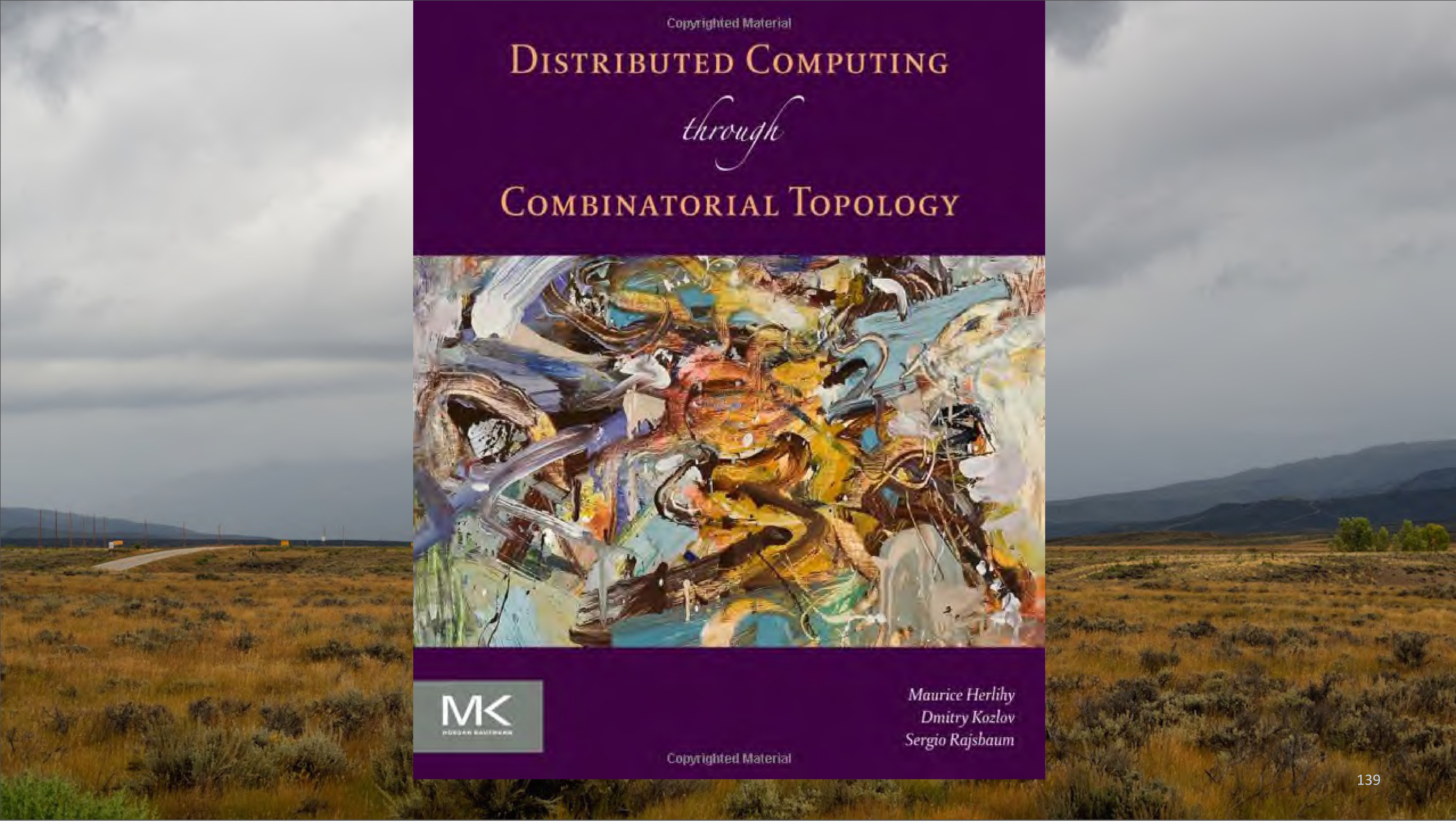


1992: Yes, “Reactive” isn’t new ;) This book is lays out a theoretical model for specifying and proving “reactive” concurrent systems based on temporal logic. While its goal is to prevent logic errors, It doesn’t discuss handling failures from environmental or other external causes in great depth.



Wednesday, March 18, 15

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



Wednesday, March 18, 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.

Wednesday, March 18, 15

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

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