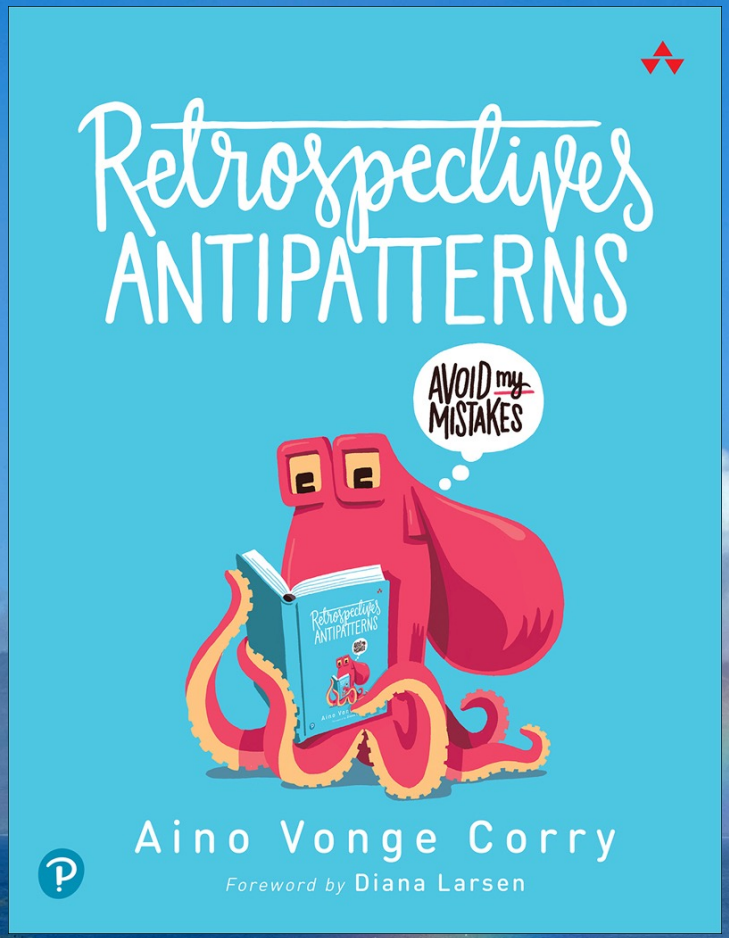
Microservices Retrospective - What we learned (and didn't learn) from Netflix

Adrian Cockcroft | Qcon London | March 2023

@adrianco@mastodon.social

All photos taken by Adrian, mostly in Maui



Retrospective? Time for me to learn some Agile rituals…


From Co-Creator of
NETFLIX
Culture Deck
POWERFUL
Building a Culture of Freedom and Responsibility
Patty McCord
Netflix was extremely agile, without the rituals of Extreme or Agile

Story Points

Retrospectives and Netflix Culture
Netflix in the Cloud – QconSF 2010, Cassandra Summit 2011
Cloud Native Applications 2013
Microservices – Dockercon and MicroXchg Berlin 2016
Why don’t microservices work for some people?

Backlog

Systems for Innovation 2015

NETFLIX

Culture

Seven aspects of Netflix culture

1. Values are what we value
2. High performance
3. Freedom and responsibility
4. Context, not control
5. Highly aligned, loosely coupled
6. Pay top of market
7. Promotions and development

NETFLIX

Systems Thinking

Optimized for agility, minimized processes

Ability to evolve rapidly

Pioneers, happy to be first to do something

Comfortable with ambiguity

Creativity and self discipline combined

Flexibility is more important than efficiency

Steer any pain to where it can be helpful

Netflix in the Cloud

Nov 6, 2010

Adrian Cockcroft
@adrianco #netflixcloud
http://www.linkedin.com/in/adriancockcroft

NETFLIX

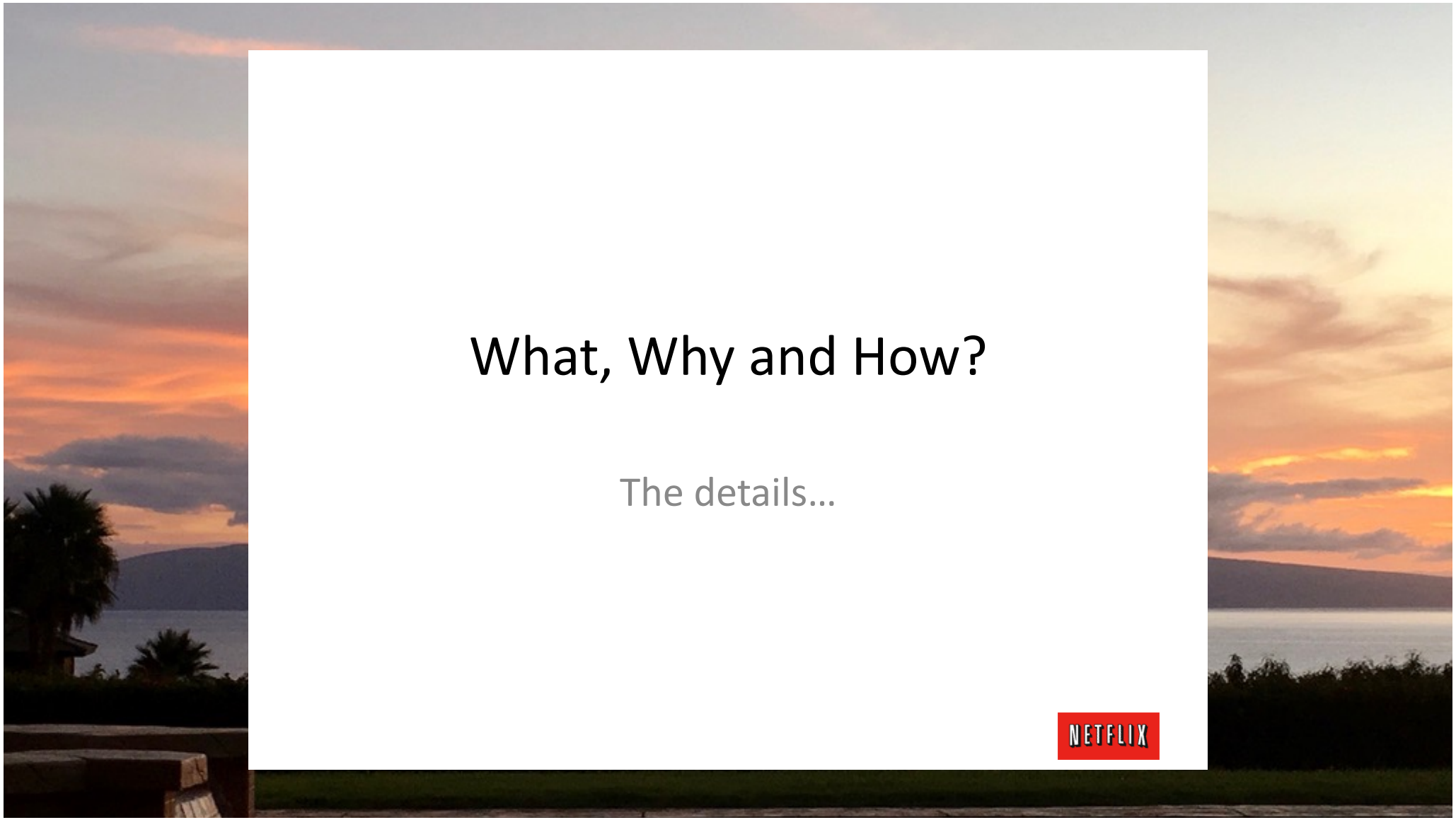
*"The cloud lets its users focus on delivering differentiating business value instead of wasting valuable resources on the **undifferentiated heavy lifting** that makes up most of IT infrastructure."*

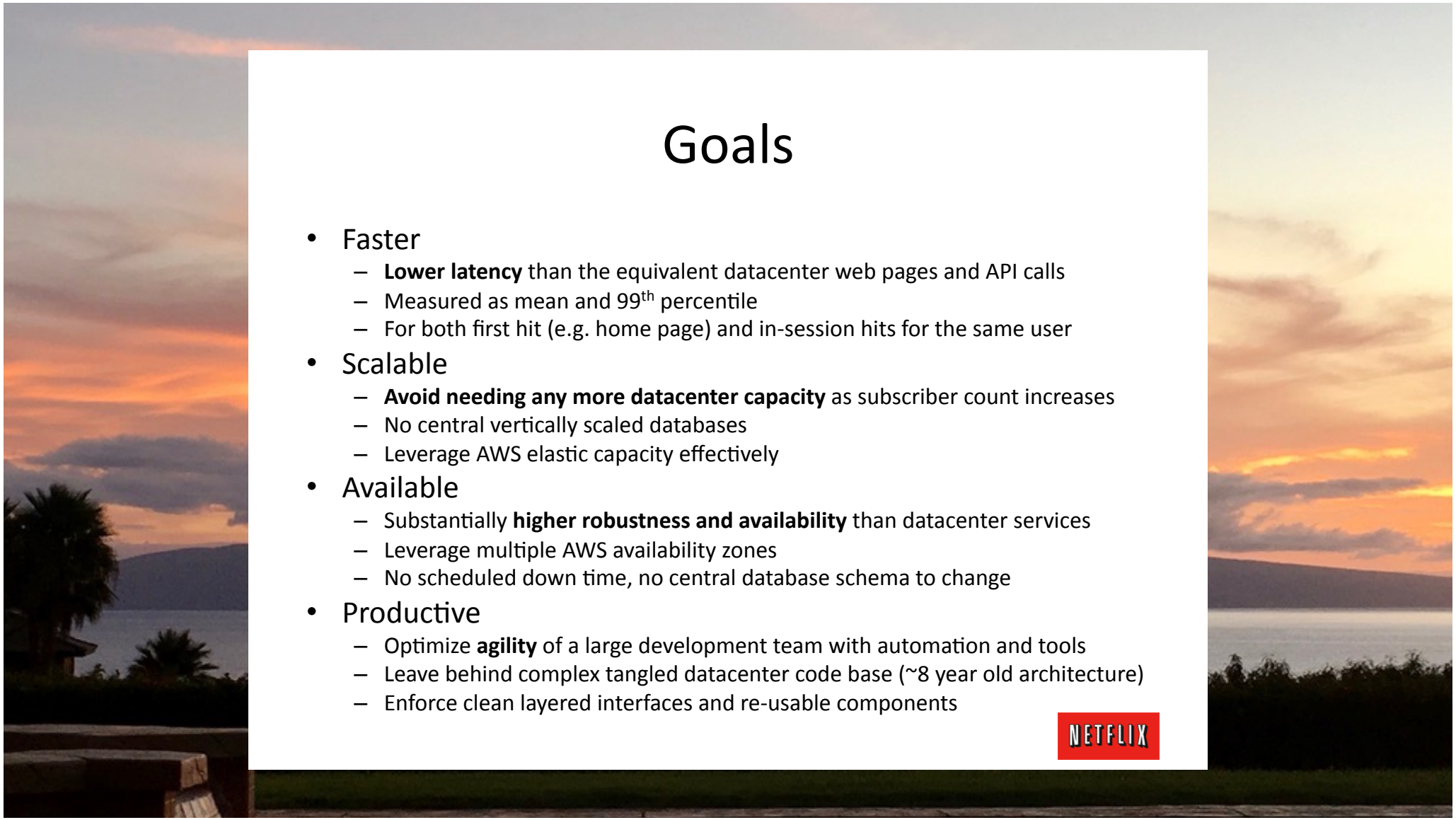
Werner Vogels

Amazon CTO

amazon web services

NETFLIX

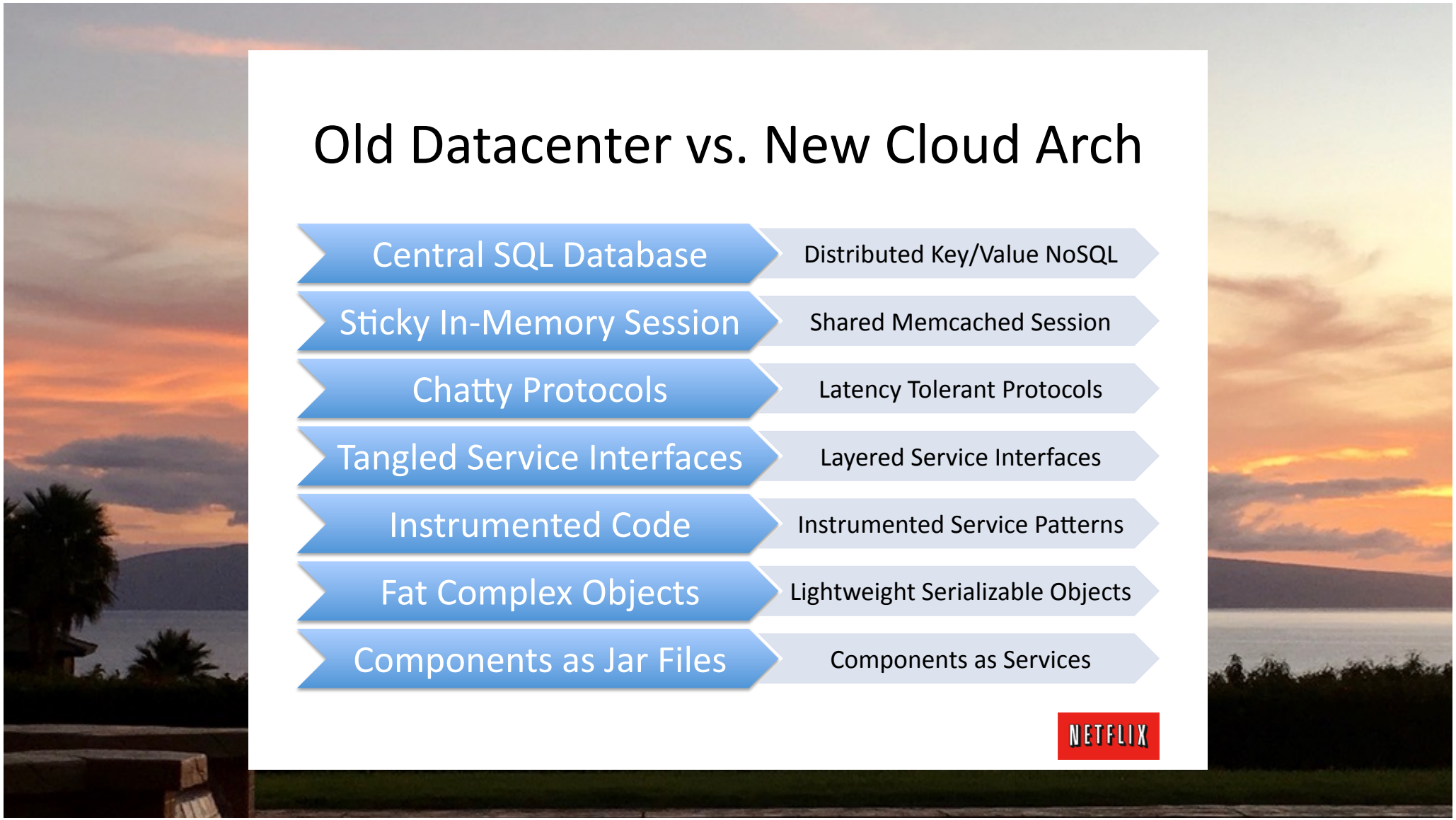

What, Why and How?
The details…
NETFLIX

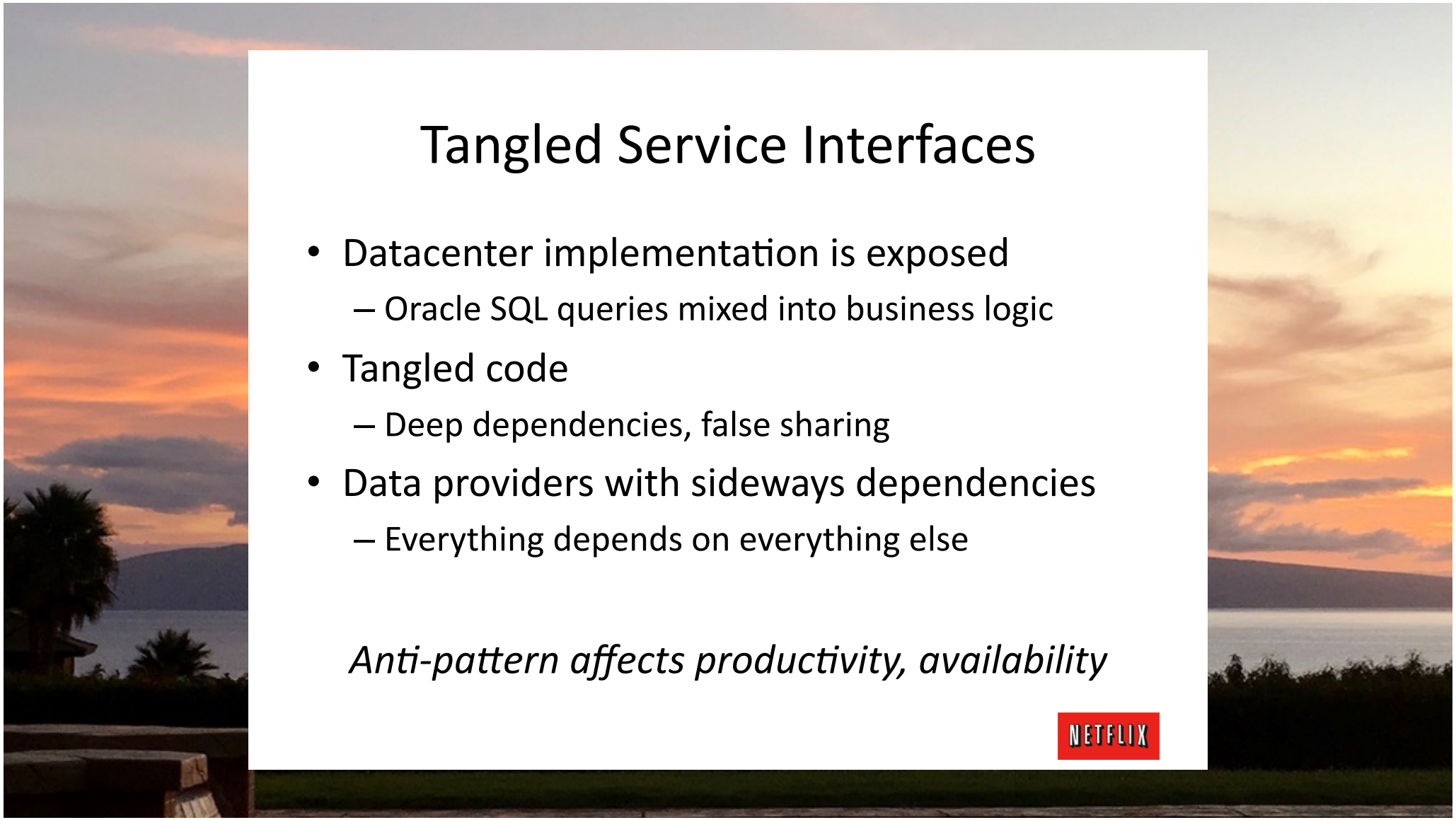


Goals

- Faster
  - **Lower latency** than the equivalent datacenter web pages and API calls
  - Measured as mean and 99th percentile
  - For both first hit (e.g. home page) and in-session hits for the same user
- Scalable
  - **Avoid needing any more datacenter capacity** as subscriber count increases
  - No central vertically scaled databases
  - Leverage AWS elastic capacity effectively
- Available
  - Substantially **higher robustness and availability** than datacenter services
  - Leverage multiple AWS availability zones
  - No scheduled down time, no central database schema to change
- Productive
  - Optimize **agility** of a large development team with automation and tools
  - Leave behind complex tangled datacenter code base (~8 year old architecture)
  - Enforce clean layered interfaces and re-usable components

NETFLIX


Old Datacenter vs. New Cloud Arch
Central SQL Database
Distributed Key/Value NoSQL
Sticky In-Memory Session
Shared Memcached Session
Chatty Protocols
Latency Tolerant Protocols
Tangled Service Interfaces
Layered Service Interfaces
Instrumented Code
Instrumented Service Patterns
Fat Complex Objects
Lightweight Serializable Objects
Components as Jar Files
Components as Services
NETFLIX

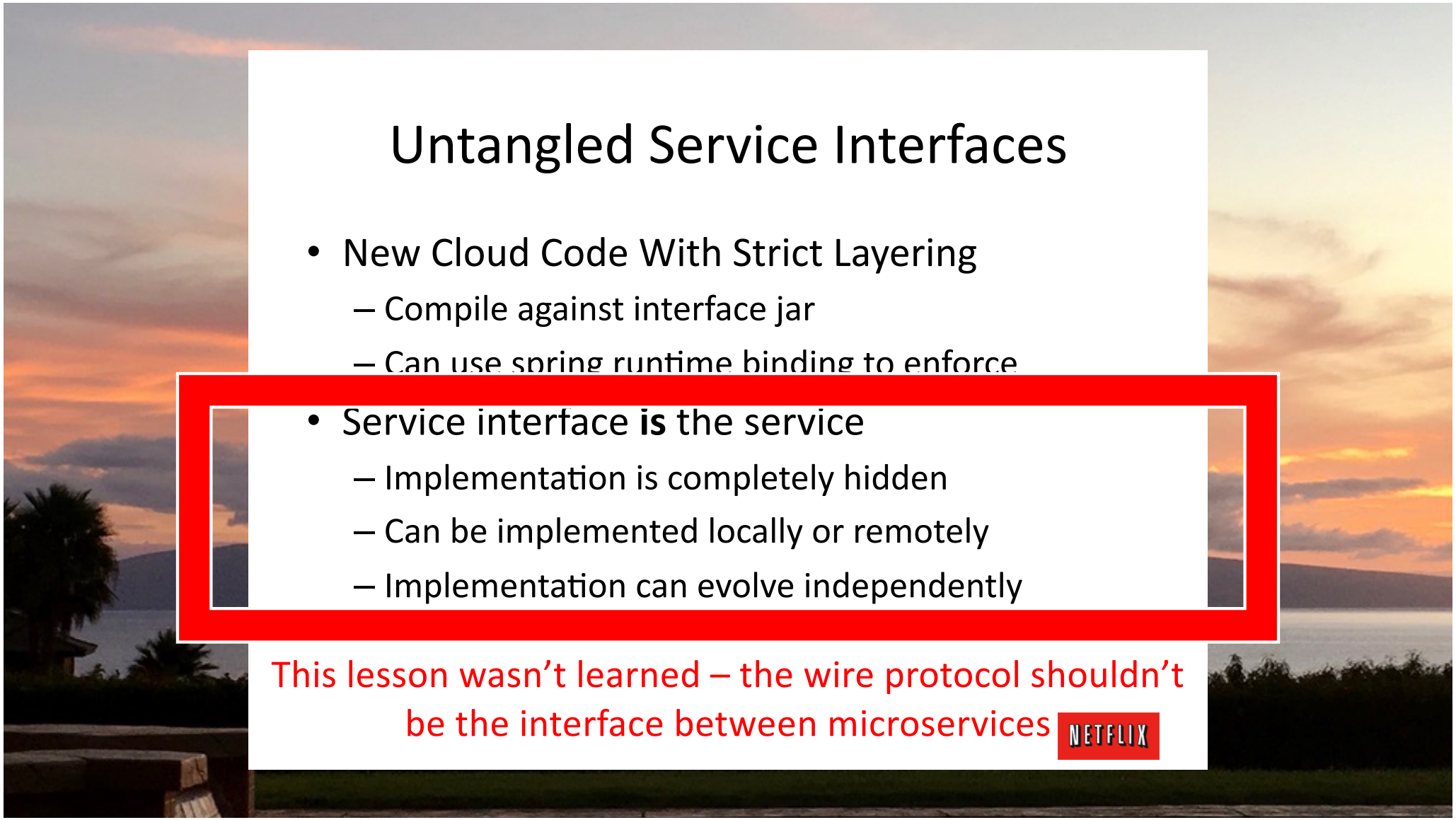


Tangled Service Interfaces

- Datacenter implementation is exposed
  - Oracle SQL queries mixed into business logic
- Tangled code
  - Deep dependencies, false sharing
- Data providers with sideways dependencies
  - Everything depends on everything else

Anti-pattern affects productivity, availability

NETFLIX

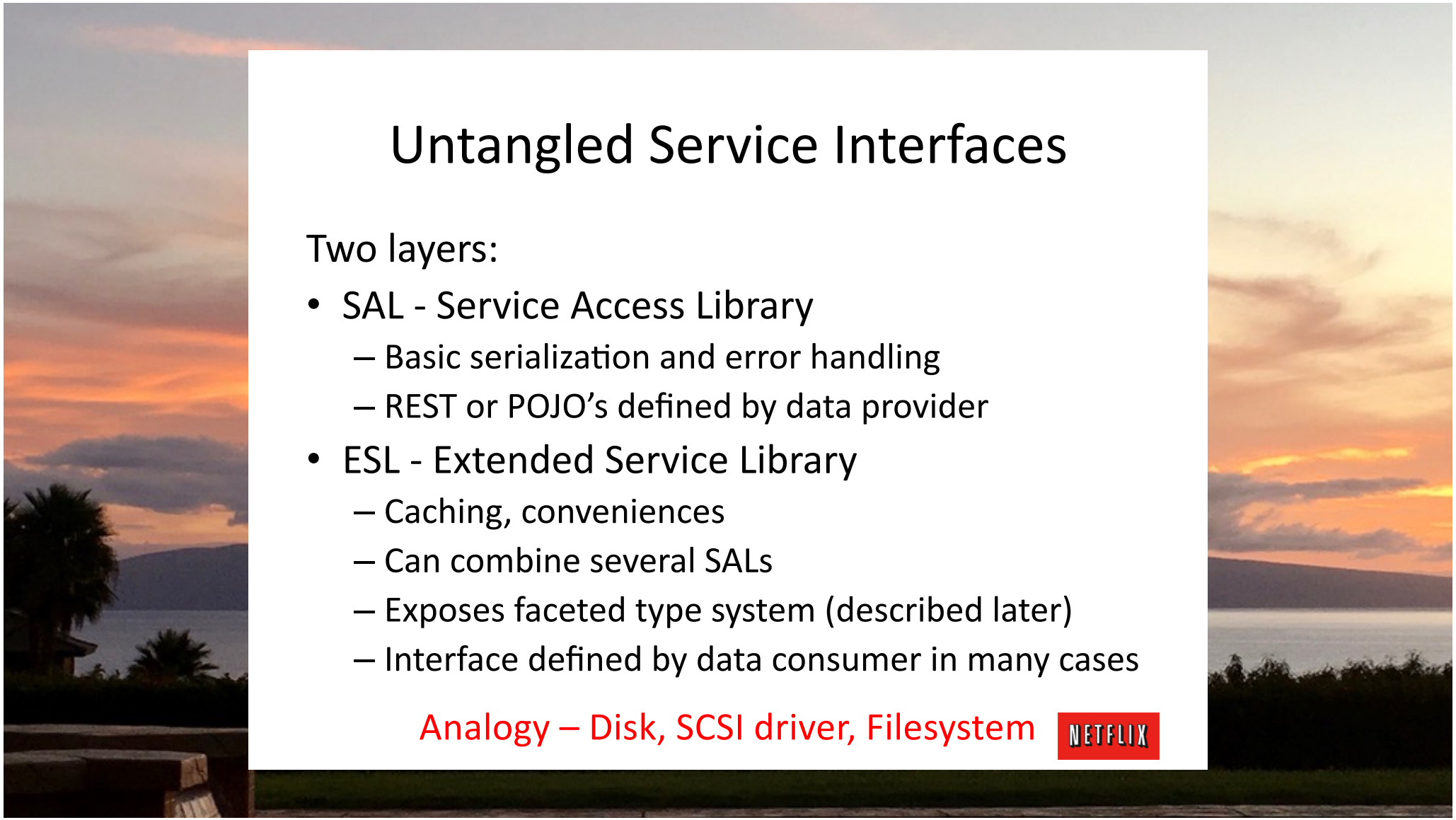


Untangled Service Interfaces

- New Cloud Code With Strict Layering
  - Compile against interface jar
  - Can use spring runtime binding to enforce
- Service interface **is** the service
  - Implementation is completely hidden
  - Can be implemented locally or remotely
  - Implementation can evolve independently

This lesson wasn't learned – the wire protocol shouldn't be the interface between microservices

NETFLIX



Untangled Service Interfaces

Two layers:

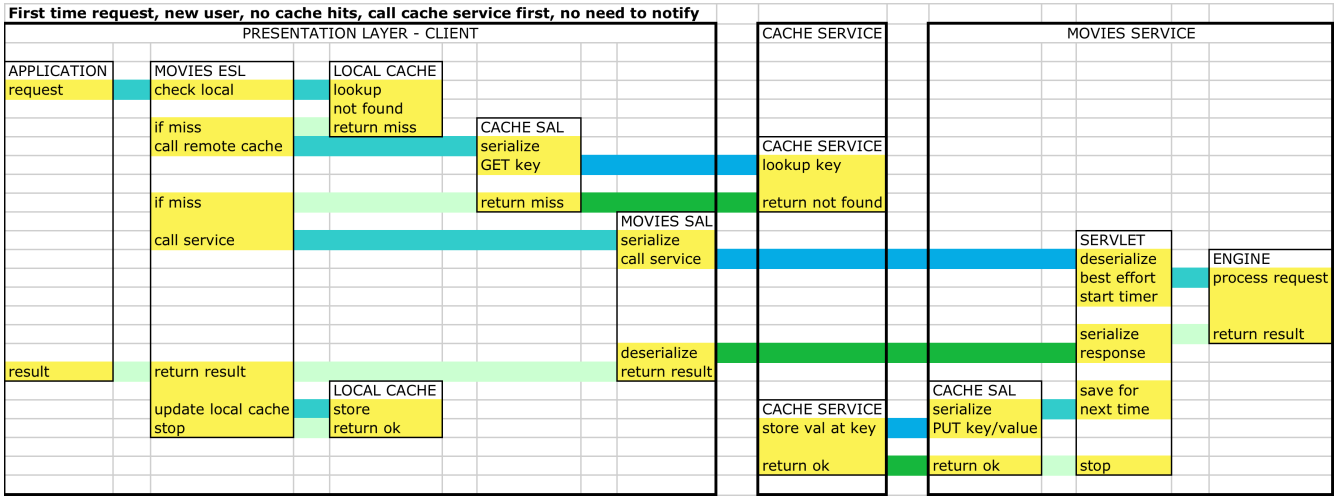
- SAL - Service Access Library
  - Basic serialization and error handling
  - REST or POJO's defined by data provider
- ESL - Extended Service Library
  - Caching, conveniences
  - Can combine several SALs
  - Exposes faceted type system (described later)
  - Interface defined by data consumer in many cases

Analogy – Disk, SCSI driver, Filesystem

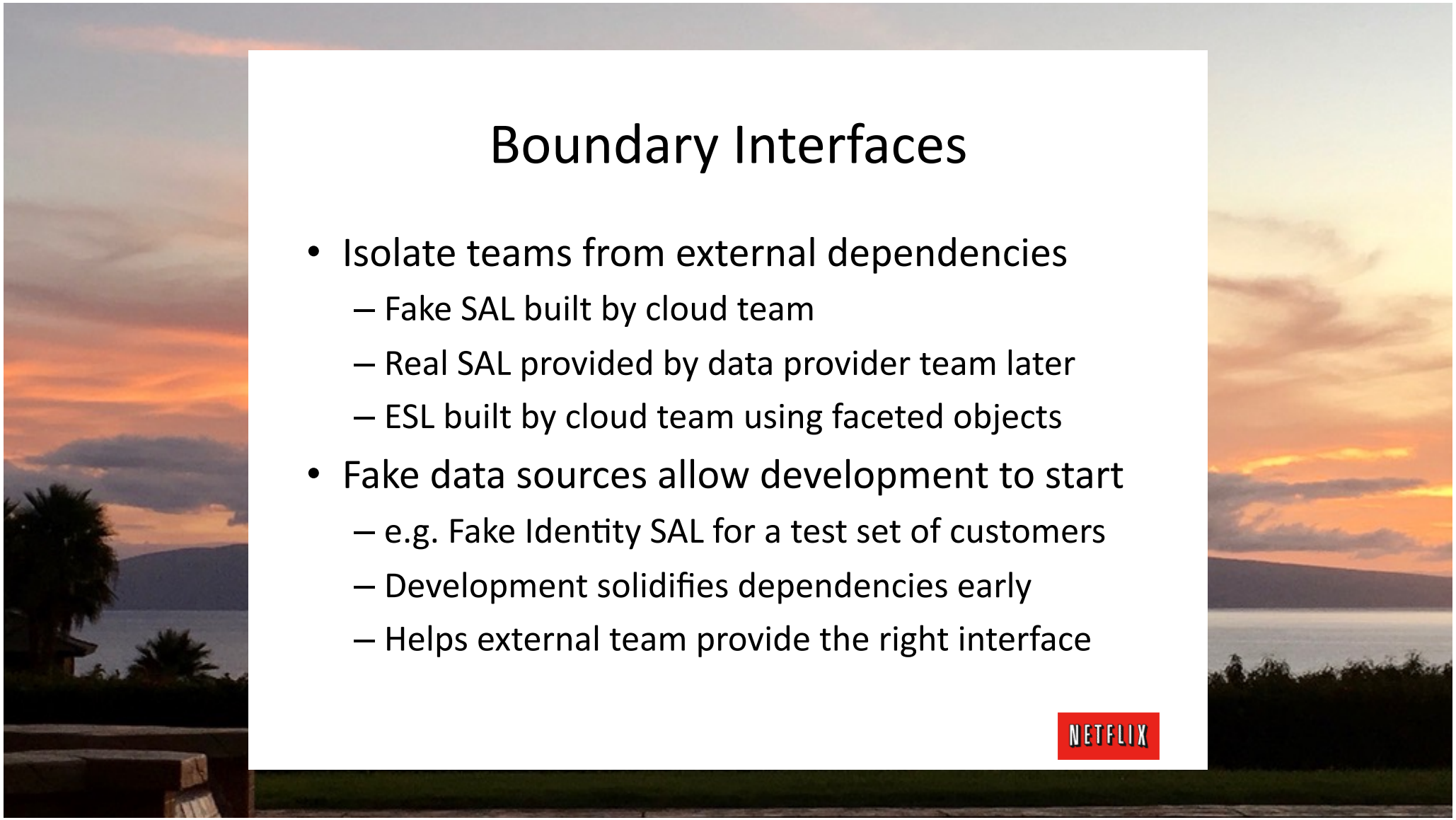
NETFLIX

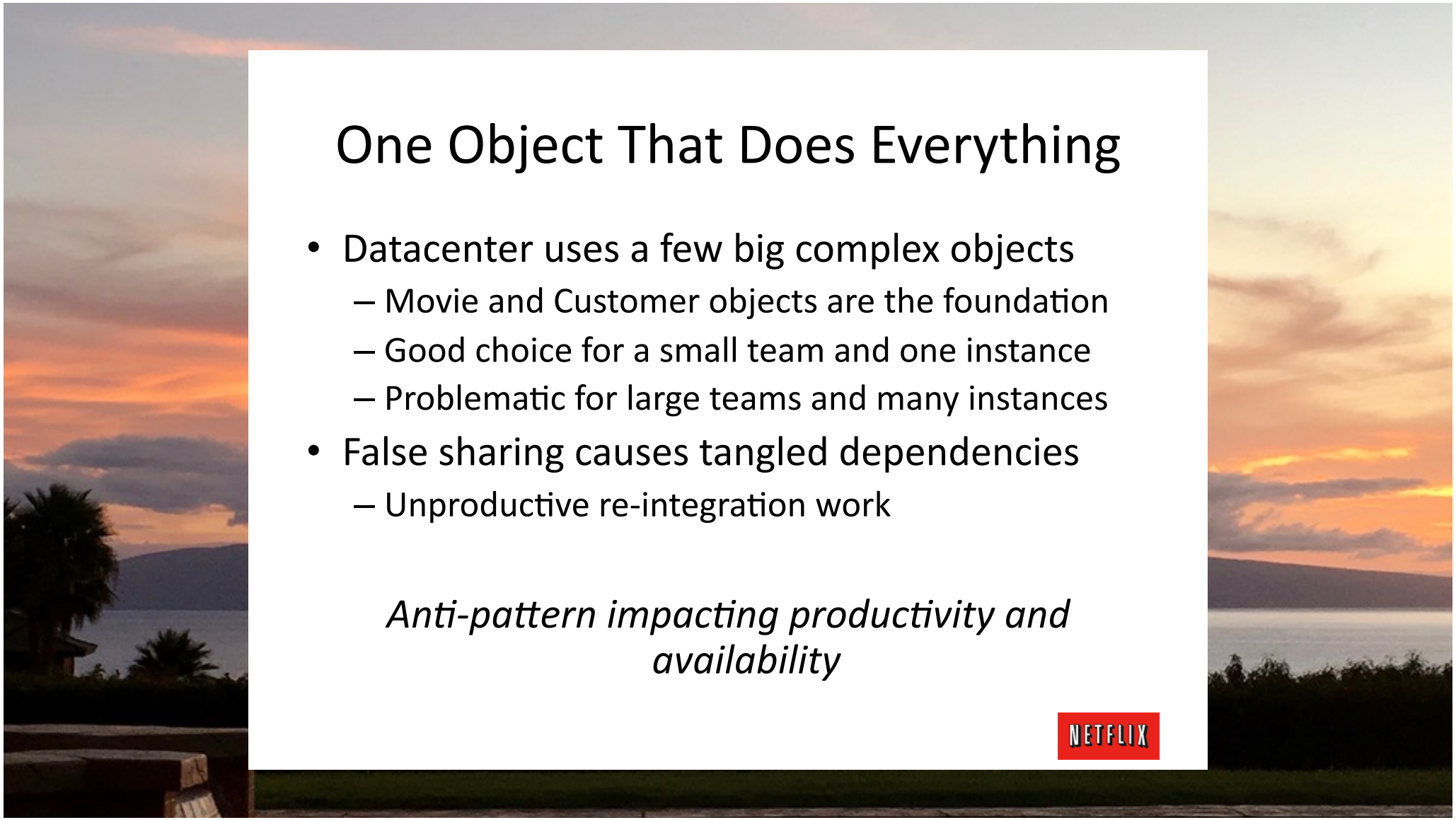
Service Interaction Pattern

Sample Swimlane Diagram



NETFLIX


Boundary Interfaces
• Isolate teams from external dependencies
– Fake SAL built by cloud team
– Real SAL provided by data provider team later
– ESL built by cloud team using faceted objects
• Fake data sources allow development to start
– e.g. Fake Identity SAL for a test set of customers
– Development solidifies dependencies early
– Helps external team provide the right interface
NETFLIX

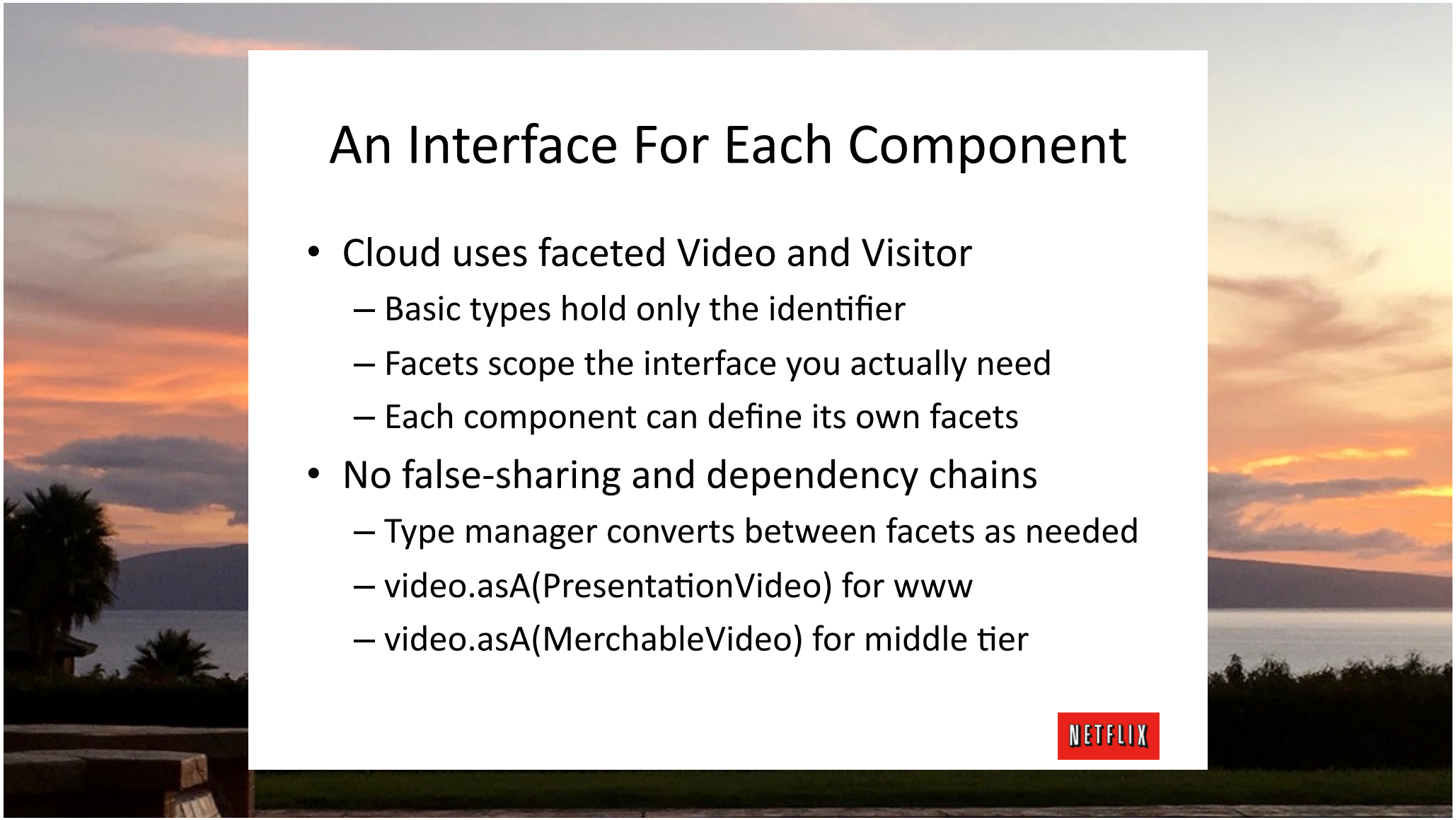


One Object That Does Everything

- Datacenter uses a few big complex objects
  - Movie and Customer objects are the foundation
  - Good choice for a small team and one instance
  - Problematic for large teams and many instances
- False sharing causes tangled dependencies
  - Unproductive re-integration work

Anti-pattern impacting productivity and availability

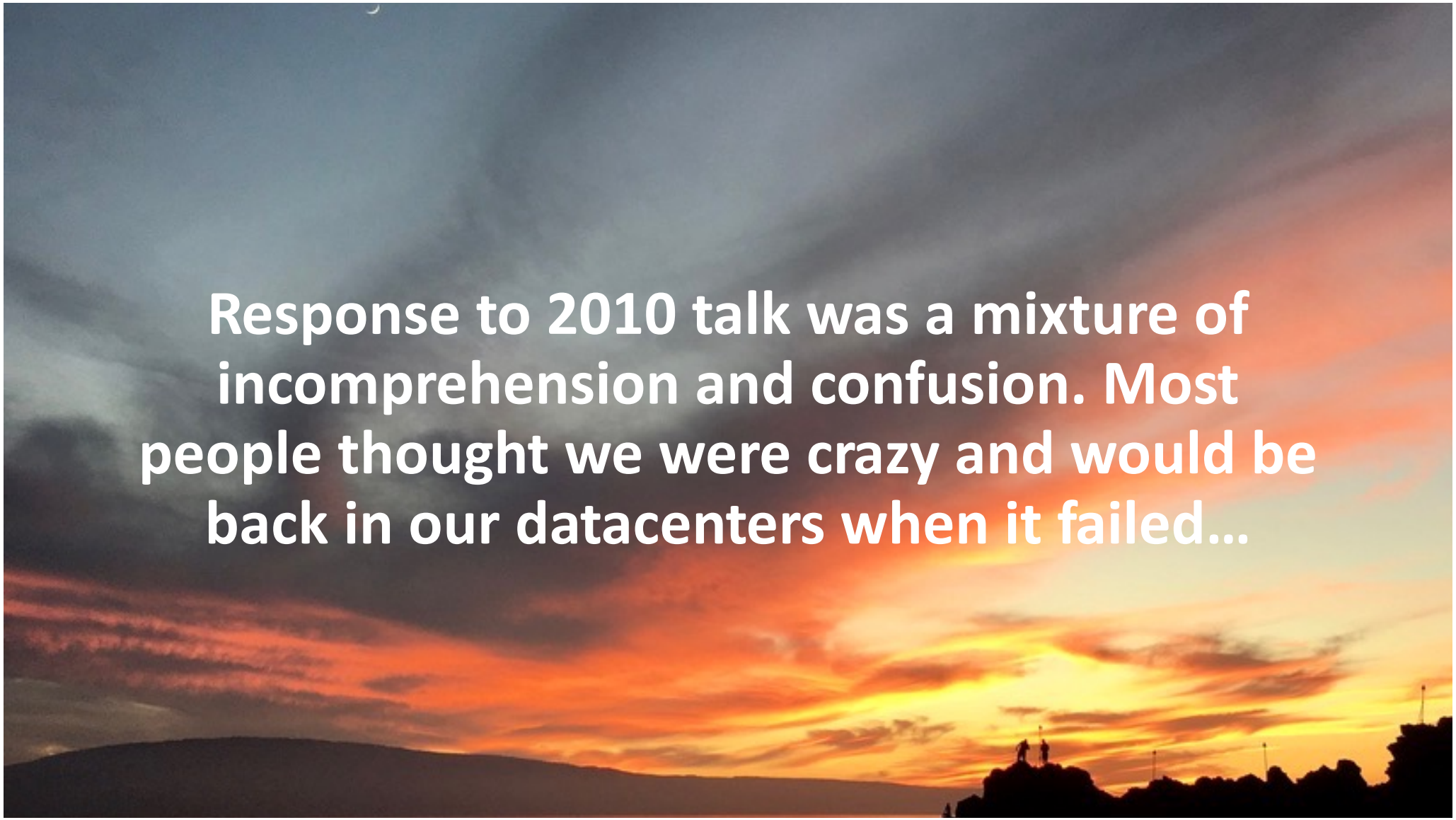
NETFLIX



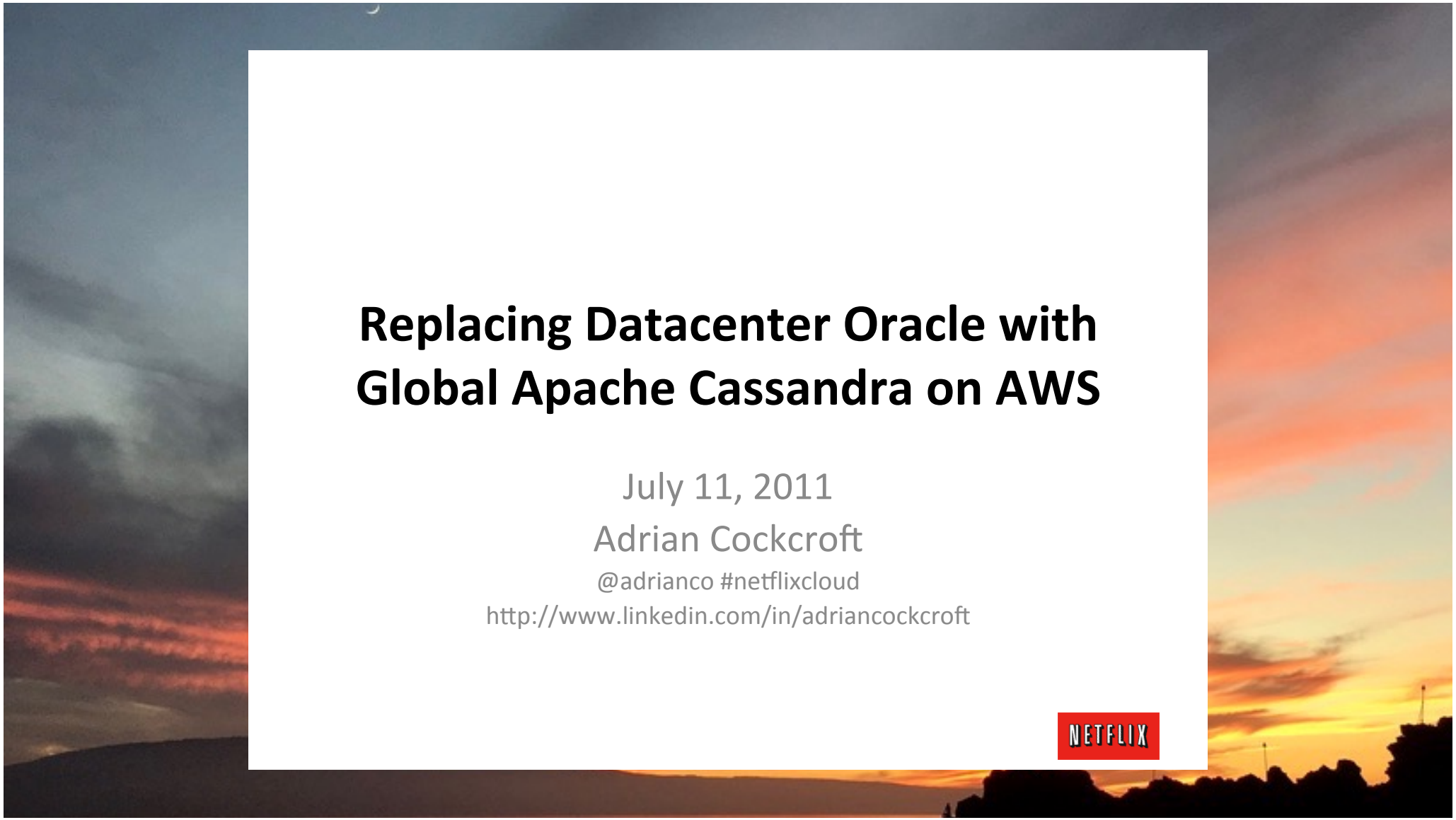
An Interface For Each Component

- Cloud uses faceted Video and Visitor
  - Basic types hold only the identifier
  - Facets scope the interface you actually need
  - Each component can define its own facets
- No false-sharing and dependency chains
  - Type manager converts between facets as needed
  - video.asA(PresentationVideo) for www
  - video.asA(MerchableVideo) for middle tier

NETFLIX


Response to 2010 talk was a mixture of incomprehension and confusion. Most people thought we were crazy and would be back in our datacenters when it failed…


Less than a year later…

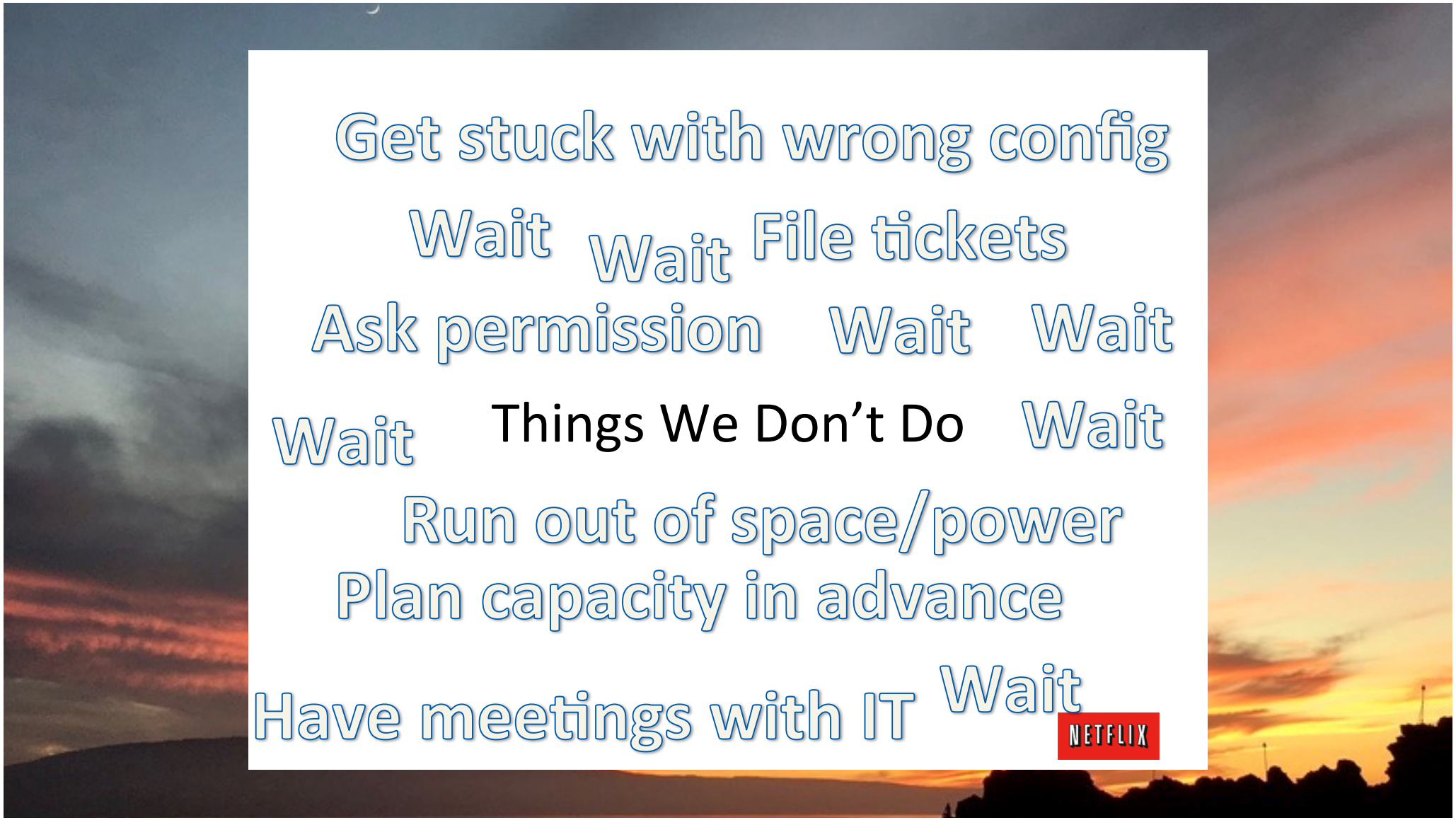


Replacing Datacenter Oracle with Global Apache Cassandra on AWS

July 11, 2011

Adrian Cockcroft
@adrianco #netflixcloud
http://www.linkedin.com/in/adriancockcroft

NETFLIX


Get stuck with wrong config
Wait
Wait
File tickets
Ask permission
Wait
Wait
Wait
Things We Don't Do
Wait
Run out of space/power
Plan capacity in advance
Have meetings with IT
Wait
NETFLIX

Netflix could not build new datacenters fast enough

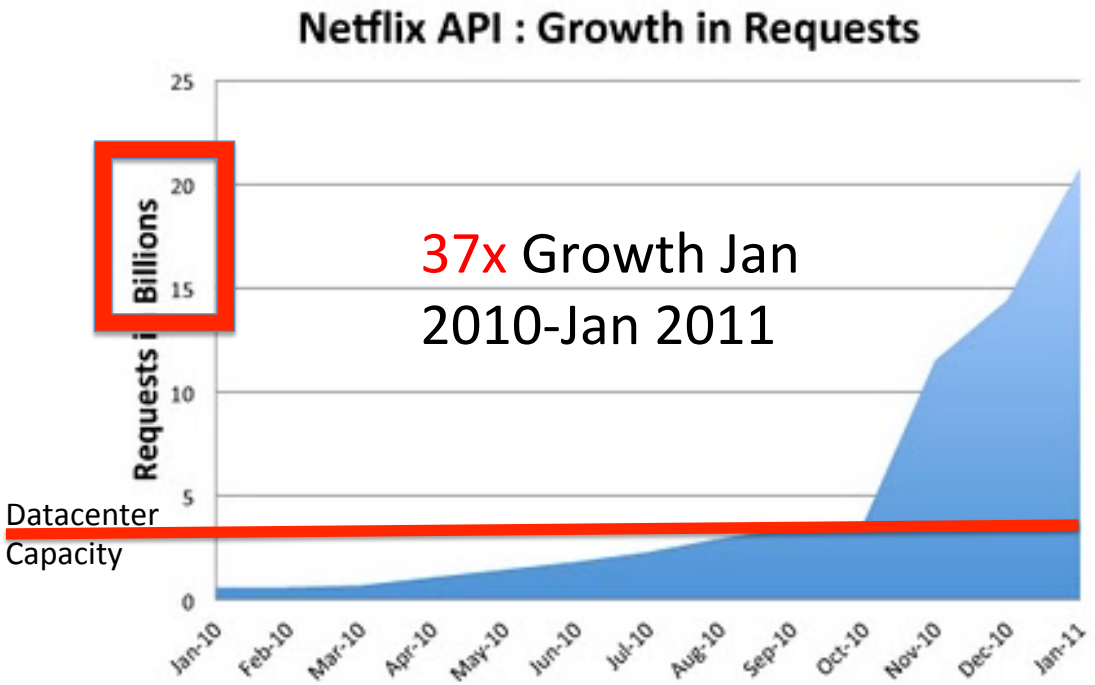
Capacity growth is accelerating, unpredictable

Product launch spikes - iPhone, Wii, PS3, XBox

NETFLIX

Out-Growing Data Center

http://techblog.netflix.com/2011/02/redesigning-netflix-api.html



NETFLIX

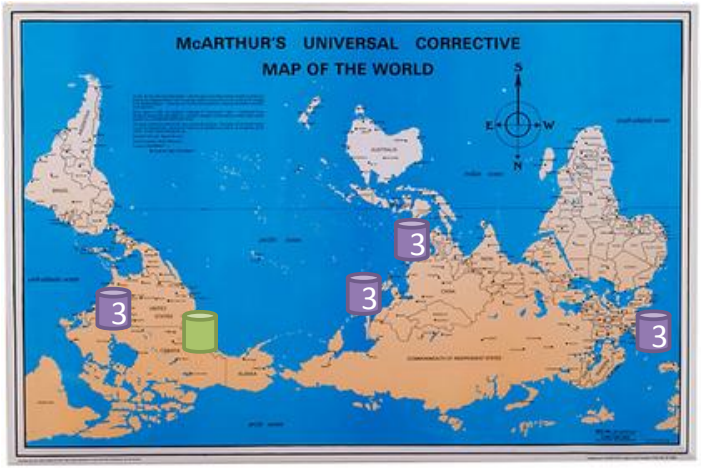
High Availability

- Cassandra stores 3 local copies, 1 per zone
  - Synchronous access, durable, highly available
  - Read/Write One fastest, least consistent - ~1ms
  - Read/Write Quorum 2 of 3, consistent - ~3ms
- AWS Availability Zones
  - Separate buildings
  - Separate power etc.
  - Close together

Actually 10-100km apart

Remote Copies

- Cassandra duplicates across AWS regions
  - Asynchronous write, replicates at destination
  - Doesn't directly affect local read/write latency
- Global Coverage
  - Business agility
  - Follow AWS…
- Local Access
  - Better latency
  - Fault Isolation



NETFLIX

Chaos Monkey

- Make sure systems are resilient
  - Allow any instance to fail without customer impact
- Chaos Monkey hours
  - Monday-Thursday 9am-3pm random instance kill
- Application configuration option
  - Apps now have to opt-out from Chaos Monkey
- Computers (Datacenter or AWS) randomly die
  - Fact of life, but too infrequent to test resiliency

Architecture design control
Be sure you can auto-scale down!

NETFLIX


Response to 2011 progress was that Netflix was a Unicorn, and while it might work for us, it wasn’t relevant to others


18 Months later in 2013
Early use of Cloud Native and
Microservices terminology

Cloud Native Applications: What Changed?

February 2013

Adrian Cockcroft

@adrianco #netflixcloud @NetflixOSS

http://www.linkedin.com/in/adriancockcroft

Cloud Native?

Agile Infrastructure

NetflixOSS – Cloud Native On-Ramp

August 25th, 2006

AWS EC2 Launched

Infrastructure as a Service

Forklift Old Code to the Cloud

Faster than datacenter deployments but fragile

New Anti-Fragile Patterns

Micro-services

Chaos engines

HA systems composed from ephemeral components

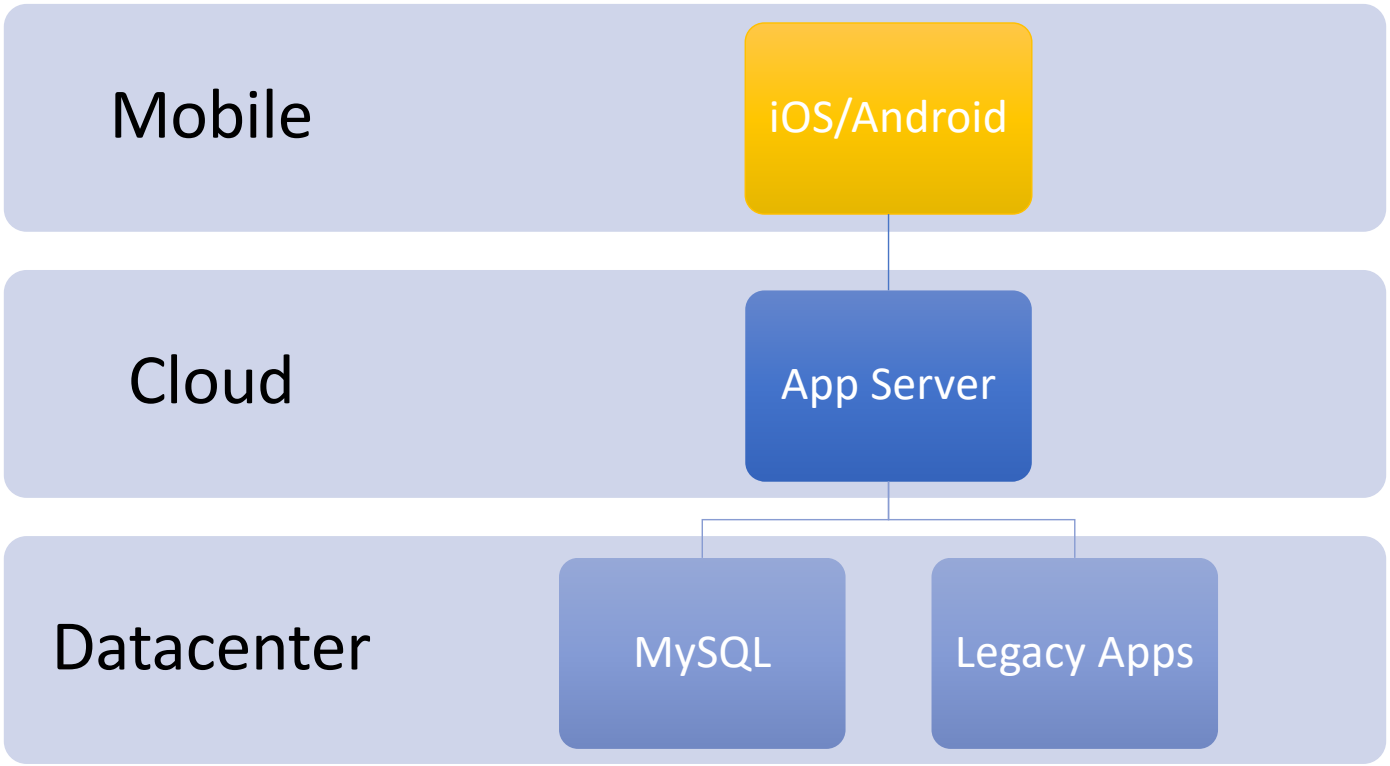
Open Source Platforms

Techniques spread rapidly

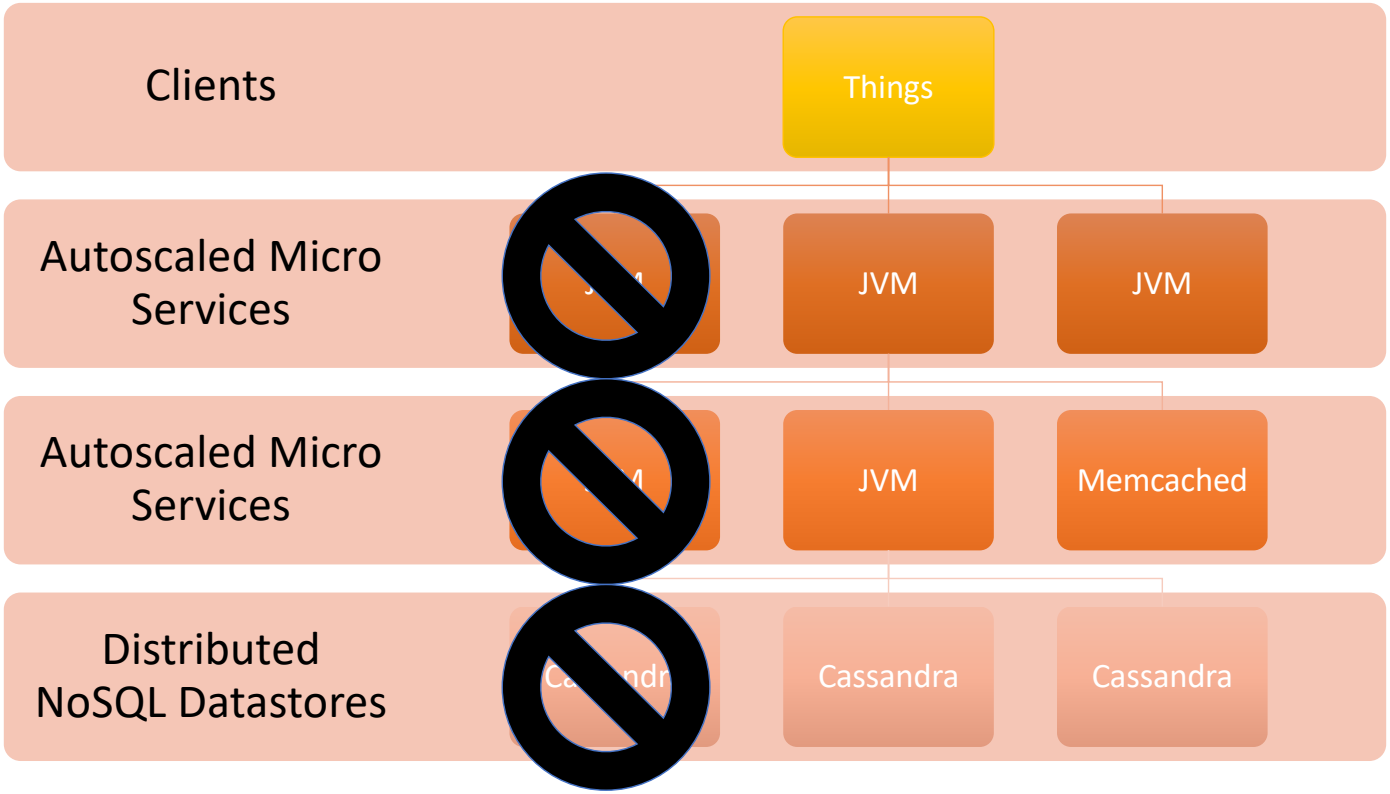
Totally new development patterns and tools

(Slow enterprise adoption)

Hybrid Cloud Architecture



Cloud Native Architecture



Cloud Native

Master copies of data are cloud resident

Everything is dynamically provisioned

All services are ephemeral

Cloud native meant that the application code was optimized to only run in the cloud
The CNCF later co-opted the term to mean code for running clouds…

Availability

Is it running yet?

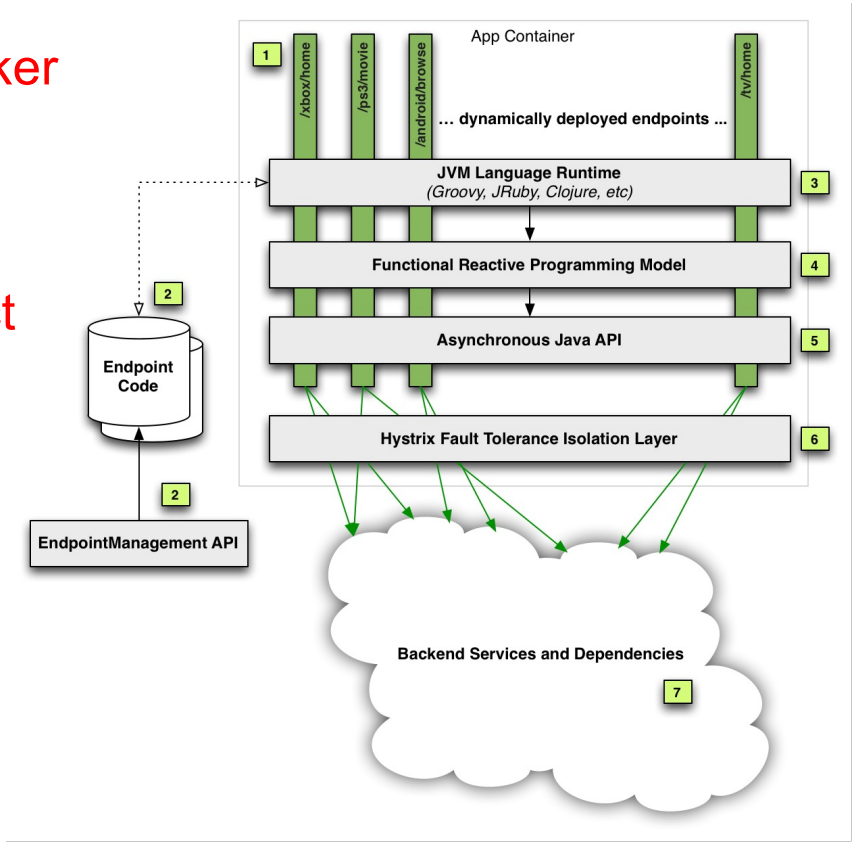
How many places is it running in?

How far apart are those places?

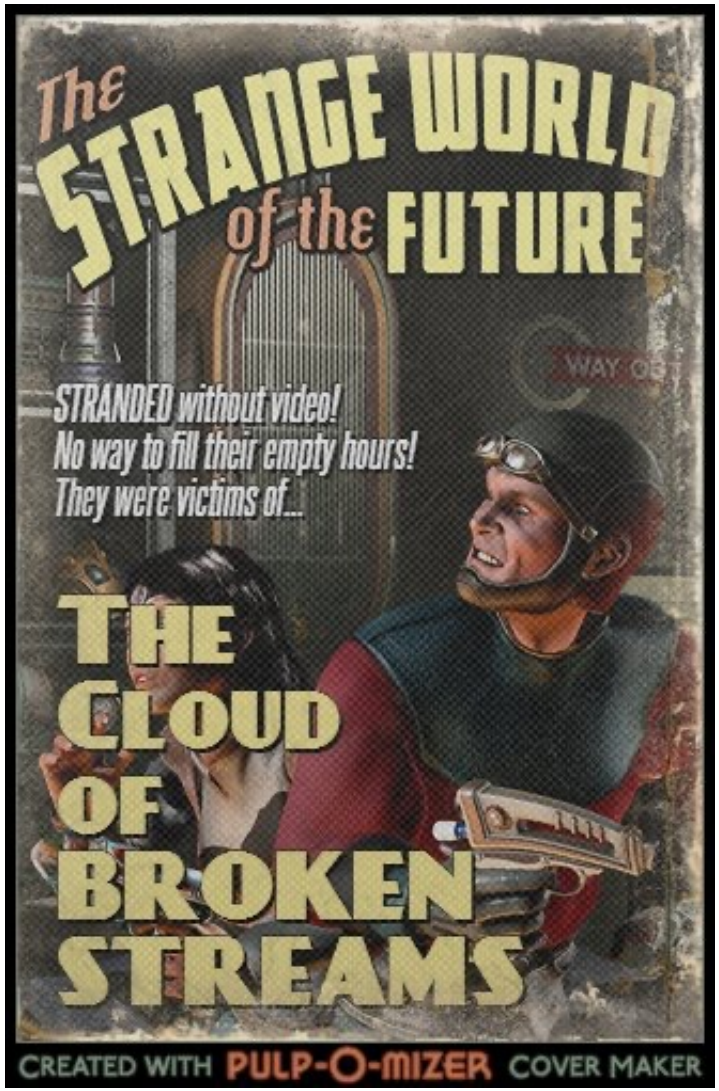
Antifragile Development Patterns

Functional Reactive with Circuit Breakers and Bulkheads

Hystrix circuit breaker library was an influential and successful Netflix open source project



NETFLIX OSS


The STRANGE WORLD of the FUTURE
WAY OUT
STRANDED without video!
No way to fill their empty hours!
They were victims of...
THE CLOUD OF BROKEN STREAMS
CREATED WITH PULP-O-MIZER COVER MAKER

Configuration Management

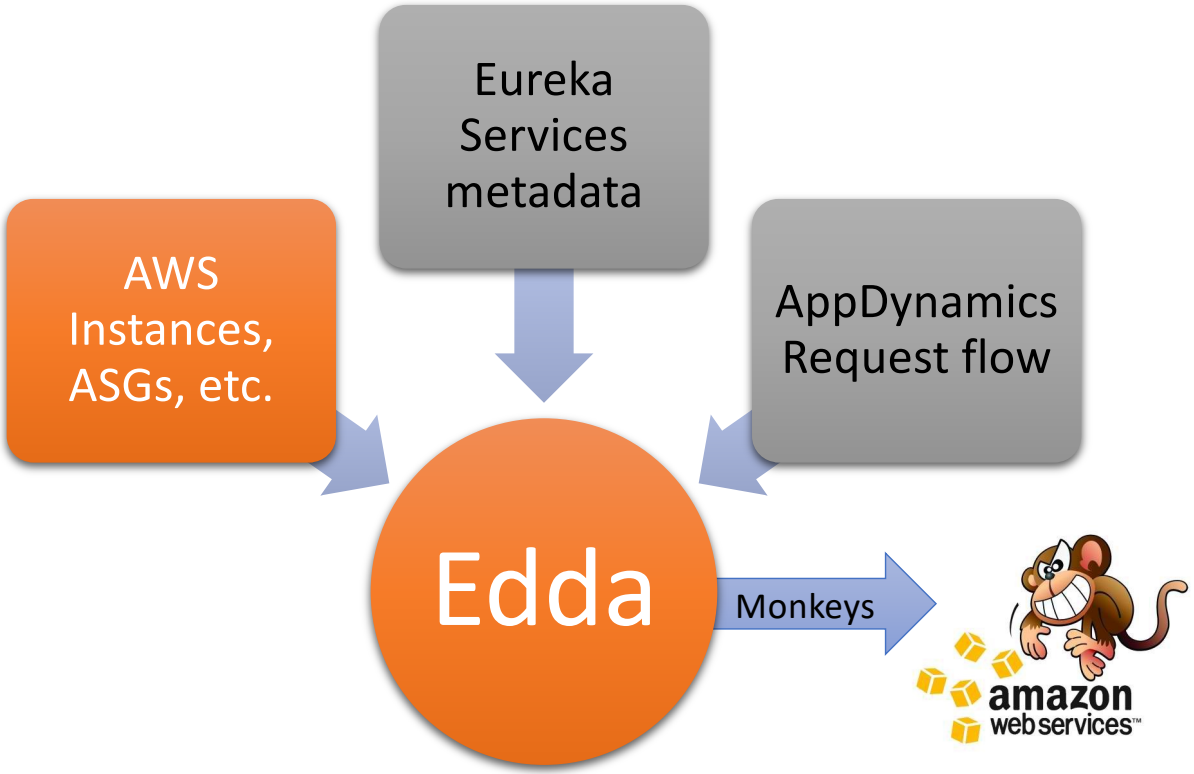
Datacenter CMDB’s woeful

Cloud is a model driven architecture

Dependably complete

Edda – Configuration History

http://techblog.netflix.com/2012/11/edda-learn-stories-of-your-cloud.html



NETFLIX
OSS

Edda Query Examples

Find any instances that have ever had a specific public IP address

```
$ curl "http://edda/api/v2/view/instances;publicIpAddress=1.2.3.4;_since=0"
 ["i-0123456789","i-012345678a","i-012345678b”]
```

Show the most recent change to a security group

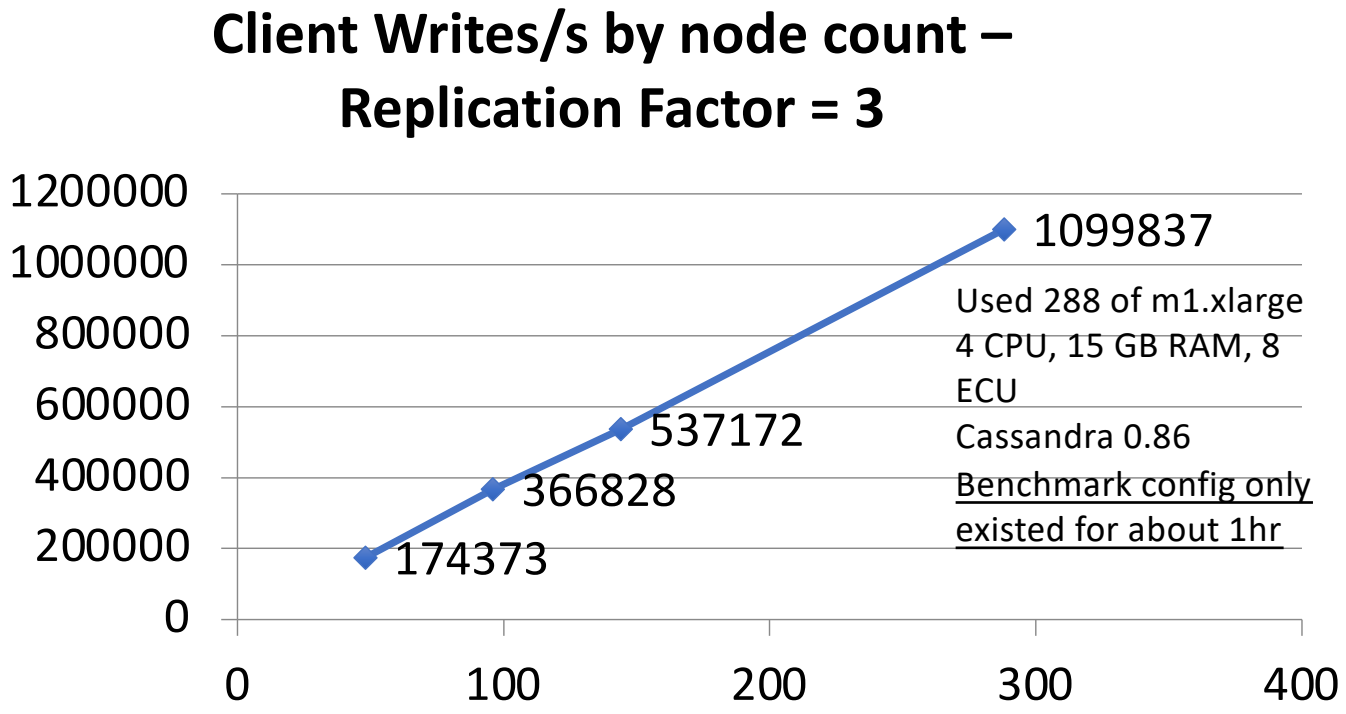
```
$ curl "http://edda/api/v2/aws/securityGroups/sg-0123456789;_diff;_all;_limit=2"
--- /api/v2/aws.securityGroups/sg-0123456789;_pp;_at=1351040779810
+++ /api/v2/aws.securityGroups/sg-0123456789;_pp;_at=1351044093504
@@ -1,33 +1,33 @@
 {
…
        "ipRanges" : [
          "10.10.1.1/32",
          "10.10.1.2/32",
+         "10.10.1.3/32",
-         "10.10.1.4/32"
…
 }
```

Not sure if this lesson was learned… but the ability to query the entire history of your infrastructure is very powerful

Frictionless infrastructure experiments!

Scalability from 48 to 288 nodes on AWS

http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html



This benchmark was used by Datastax in a “Million Writes Per Second” ad campaign

A Cloud Native Open Source Platform

Open source has become a common way to build a technology brand and influence the industry

Three Questions

Why is Netflix doing this?

How does it all fit together?

What is coming next?

Platform Evolution

Netflix ended up several years ahead of the industry, but it's not a sustainable position

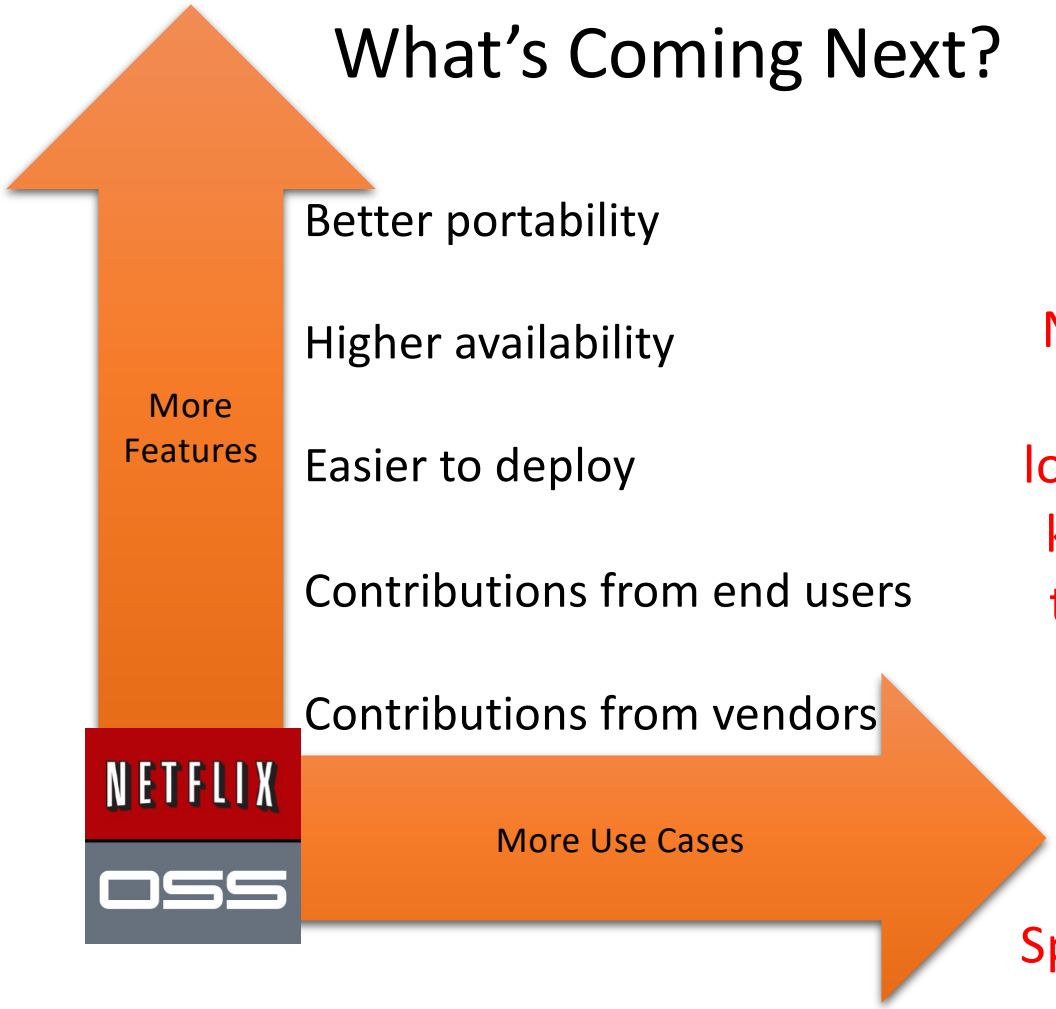
Making it easy to follow

Exploring the wild west each time

vs. laying down a shared route


Establish our solutions as Best Practices / Standards
Hire, Retain and Engage Top Engineers
Goals
Build up Netflix Technology Brand
Benefit from a shared ecosystem

What's Coming Next?



NetflixOSS was much too hard to deploy for too long. Lack of investment in key internal components that weren't ready to be shared in time

We learned from the problems and launch of Spinnaker a few years later was much better

Functionality and scale now, portability coming

Moving from parts to a platform in 2013

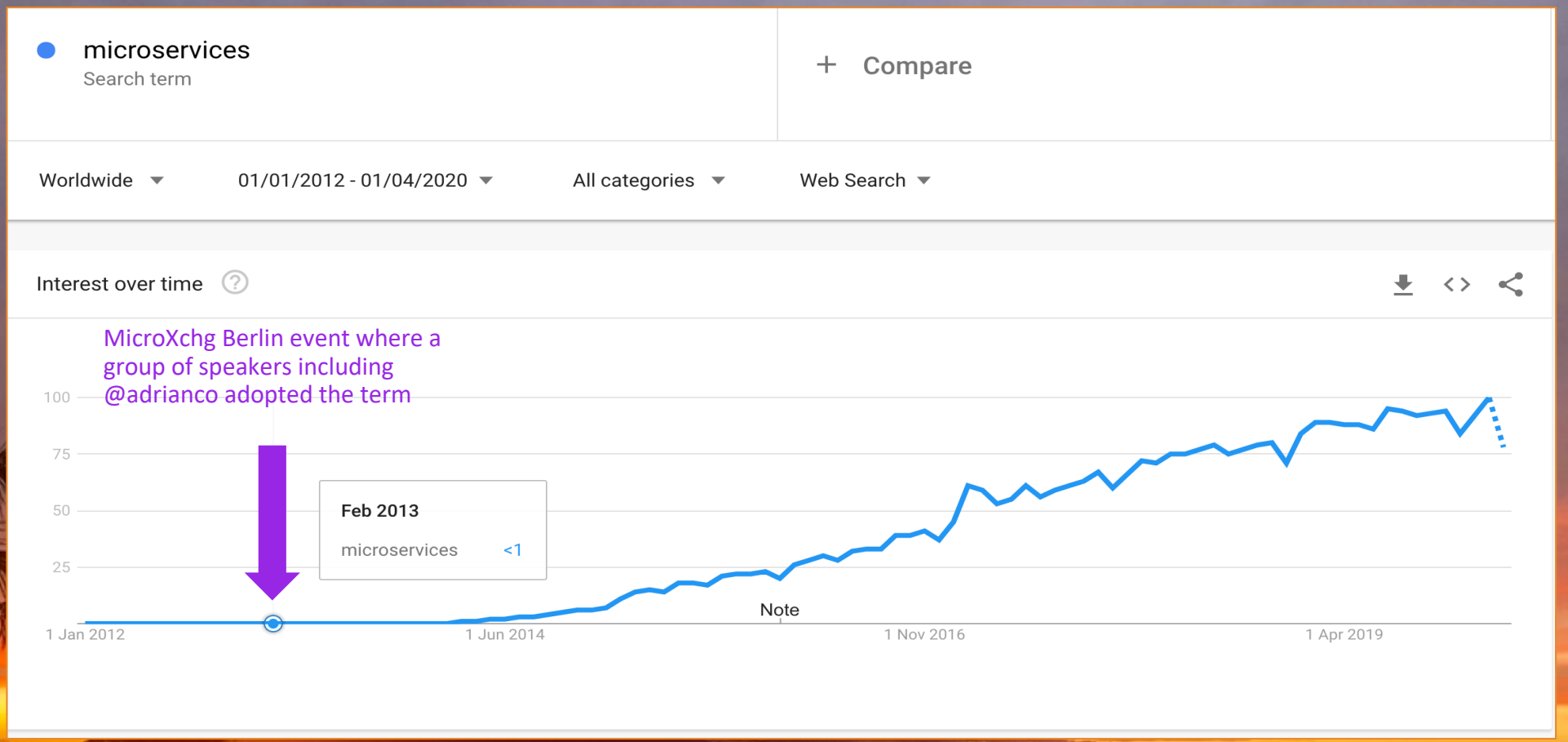
Netflix is fostering an ecosystem

Rapid Evolution - Low MTBIAMSH

(Mean Time Between Idea And Making Stuff Happen)


Adrian left Netflix in January 2014
and joined Battery Ventures just as
Docker containers began to take off as a
microservices pattern…

Trends: Microservices

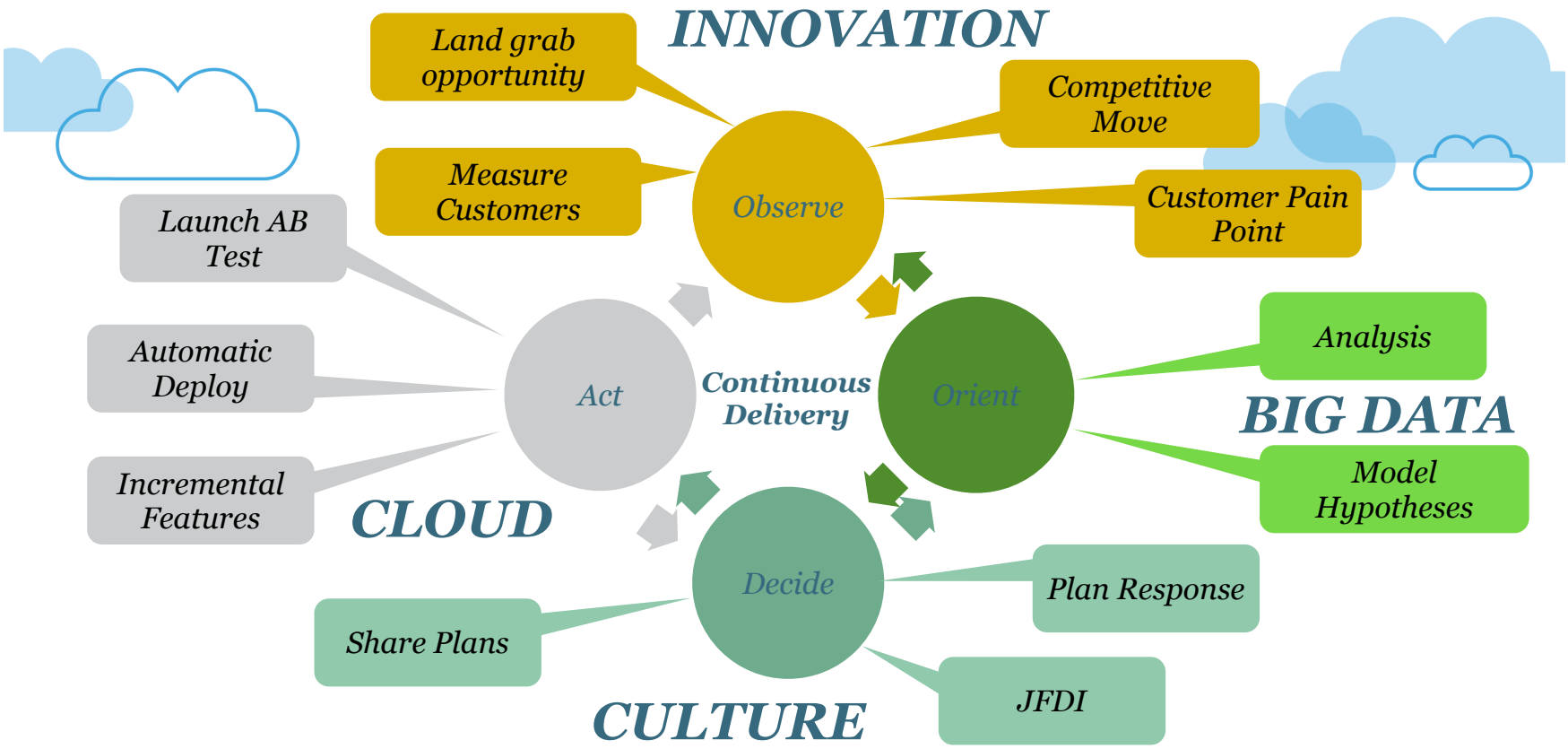



Microservices: Why, What and How
Adrian Cockcroft @adrianco
Technology Fellow - Battery Ventures
October 2016
BV
Battery Ventures

Typical reactions to my Netflix talks…

What I learned from my time at Netflix

- *Speed wins in the marketplace*
- *Remove friction from product development*
- *High trust, low process, no hand-offs between teams*
- *Freedom and responsibility culture*
- *Don't do your own undifferentiated heavy lifting*
- *Use simple patterns automated by tooling*
- *Self service cloud makes impossible things instant*


INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Launch AB Test
Automatic Deploy
Incremental Features
Act
CLOUD
Continuous Delivery
Orient
Analysis
BIG DATA
Model Hypotheses
Decide
Share Plans
Plan Response
JFDI
CULTURE

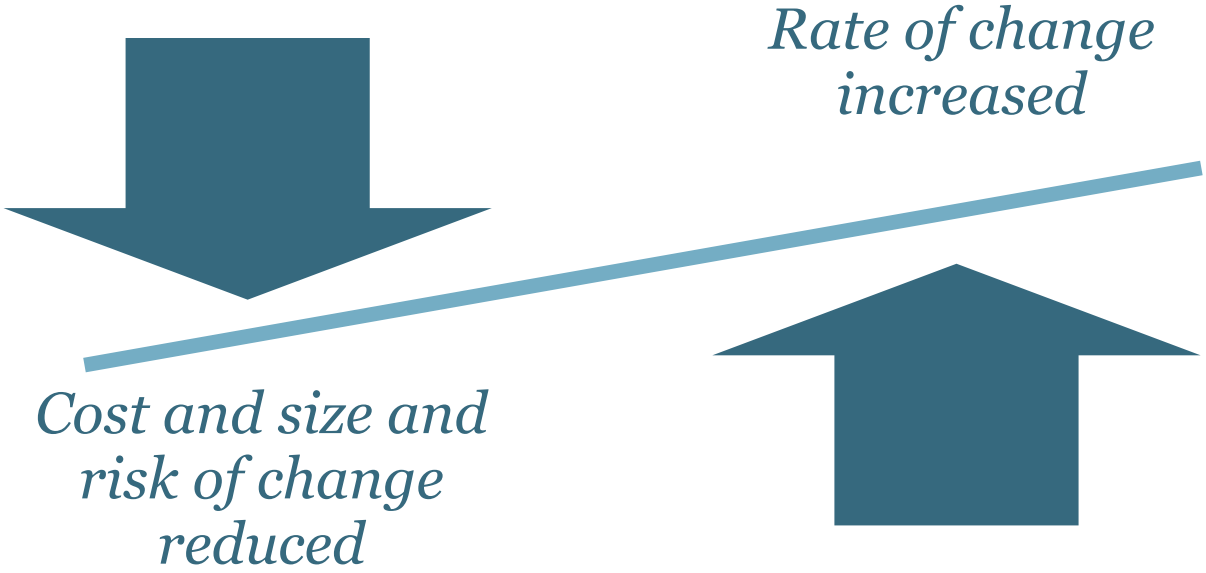
Netflix was accused of “Ruining the cloud” because we didn’t use Chef/Puppet to update our code, we just started a new instance

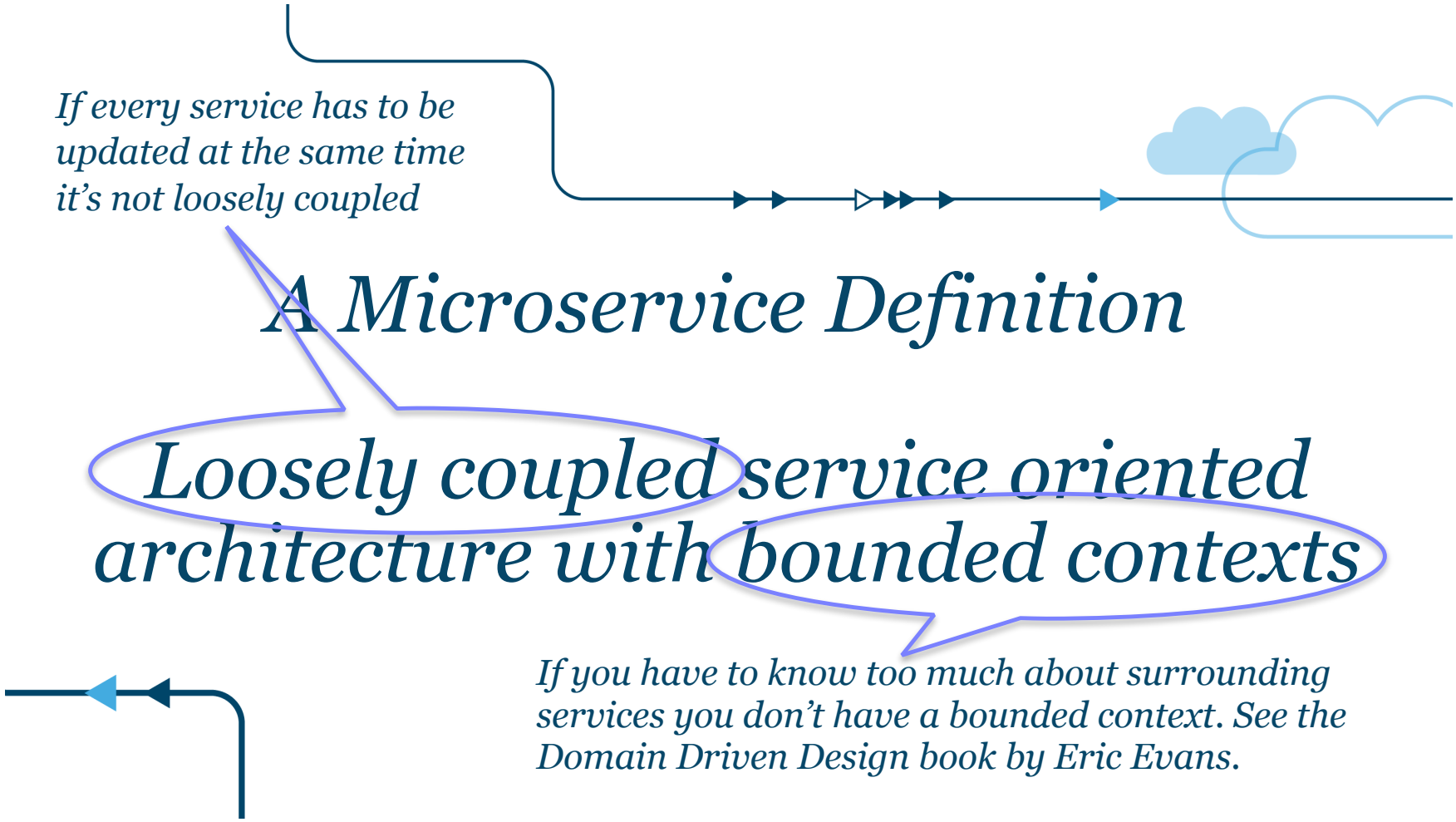
Non-Destructive Production Updates

- *“Immutable Code” Service Pattern*
  - *Existing services are unchanged, old code remains in service*
  - *New code deploys as a new service group*
  - *No impact to production until traffic routing changes*
- *A|B Tests, Feature Flags and Version Routing control traffic*
  - *First users in the test cell are the developer and test engineers*
  - *A cohort of users is added looking for measurable improvement*

The Netflix model used version-aware routing between services, another idea that seemed to get lost along the way…

What Happened?



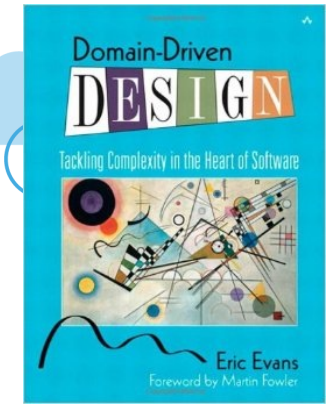

If every service has to be updated at the same time it's not loosely coupled
A Microservice Definition
Loosely coupled service oriented architecture with bounded contexts
If you have to know too much about surrounding services you don't have a bounded context. See the Domain Driven Design book by Eric Evans.

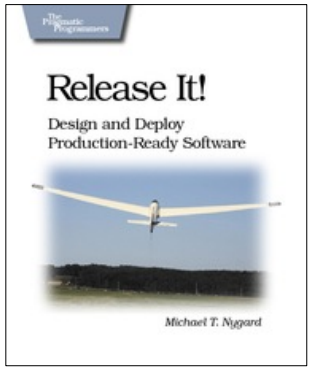
Separate Concerns with Microservices

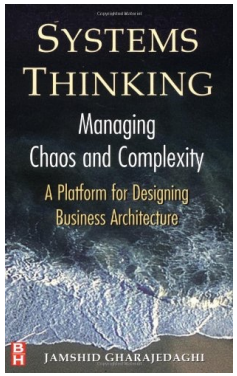
- *Invert Conway's Law – teams own service groups and backend stores*
- *One "verb" per single function micro-service, size doesn't matter*
- *One developer independently produces a micro-service*
- *Each micro-service is it's own build, avoids trunk conflicts*
- *Deploy in a container: Tomcat, AMI or Docker, whatever…*
- *Stateless business logic. Cattle, not pets.*
- *Stateful cached data access layer using replicated ephemeral instances*

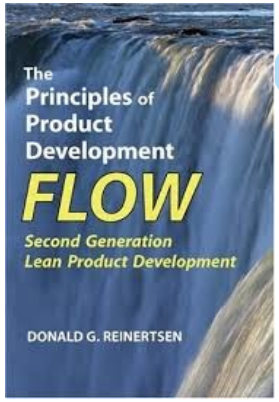
http://en.wikipedia.org/wiki/Conway's_law

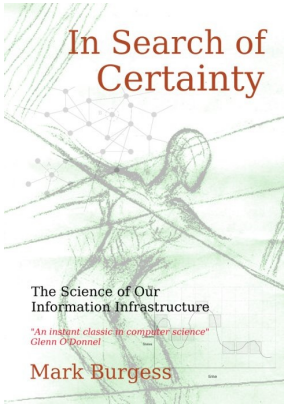
Inspiration

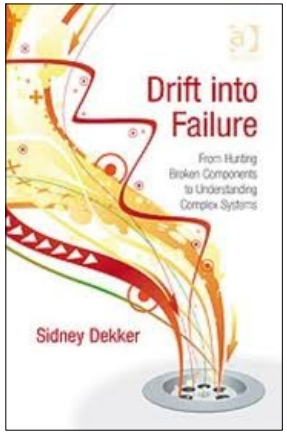


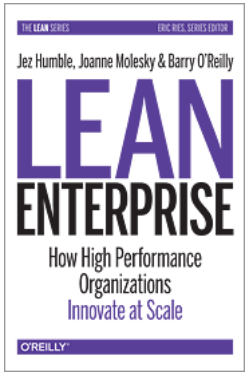


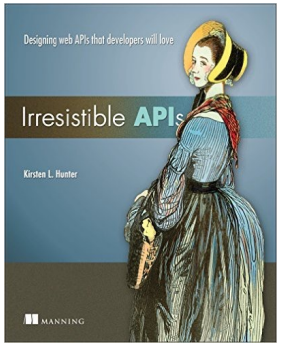


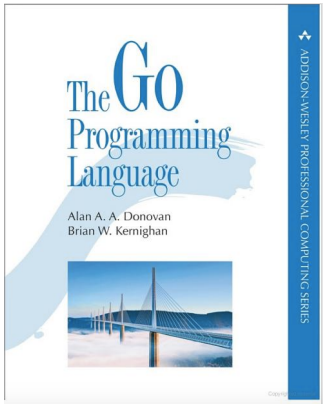


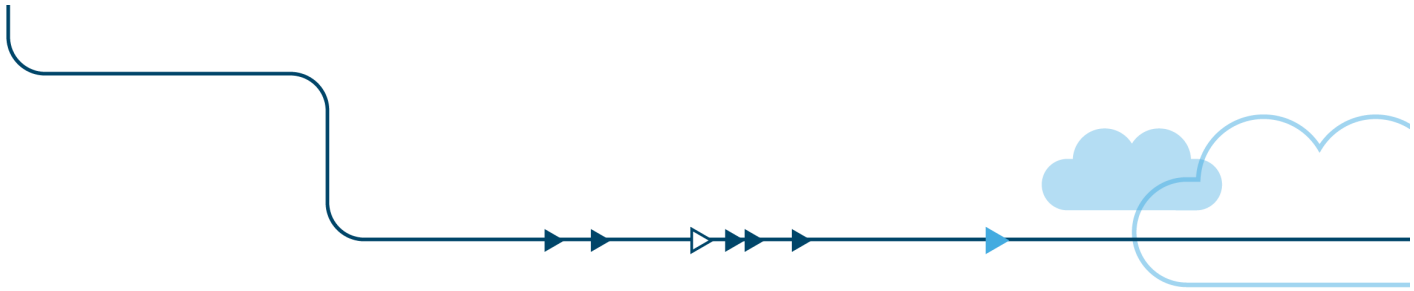




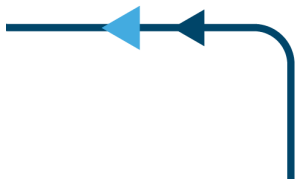








What's Missing?



Advanced Microservices Topics

Failure injection testing
Versioning, routing
Binary protocols and interfaces
Timeouts and retries
Denormalized data models
Monitoring, tracing
Simplicity through symmetry

Timeouts and Retries

Connection timeout vs. request timeout confusion

Usually setup incorrectly, global defaults

Systems collapse with "retry storms"

Timeouts too long, too many retries

Services doing work that can never be used

@adrianco
BV
Battery Ventures

Timeouts and Retries

Bad config: Every service defaults to 2 second timeout, two retries

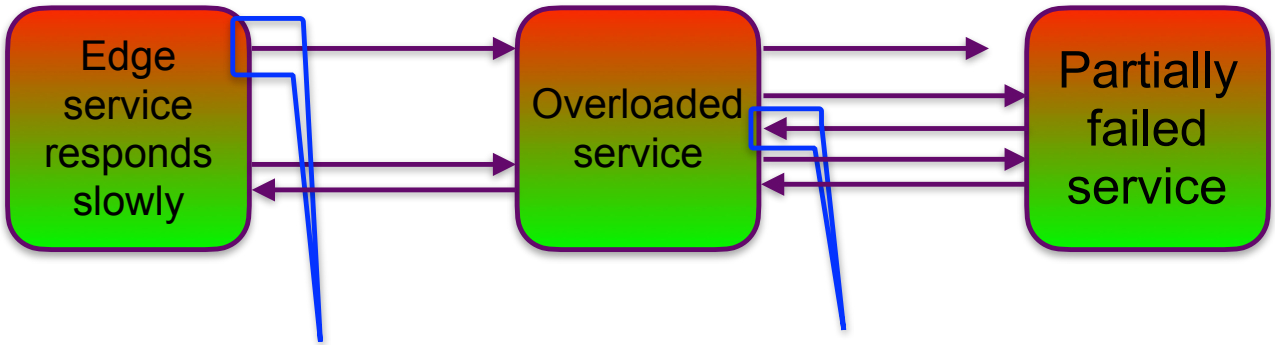
If anything breaks, everything upstream stops responding

Retries add unproductive work

@adrianco
BV
Battery Ventures

Timeouts and Retries

Bad config: Every service defaults to 2 second timeout, two retries



First request from Edge timed out so it ignores the successful response and keeps retrying. Middle service load increases as it's doing work that isn't being consumed

@adrianco
BV
Battery Ventures

Timeout and Retry Fixes

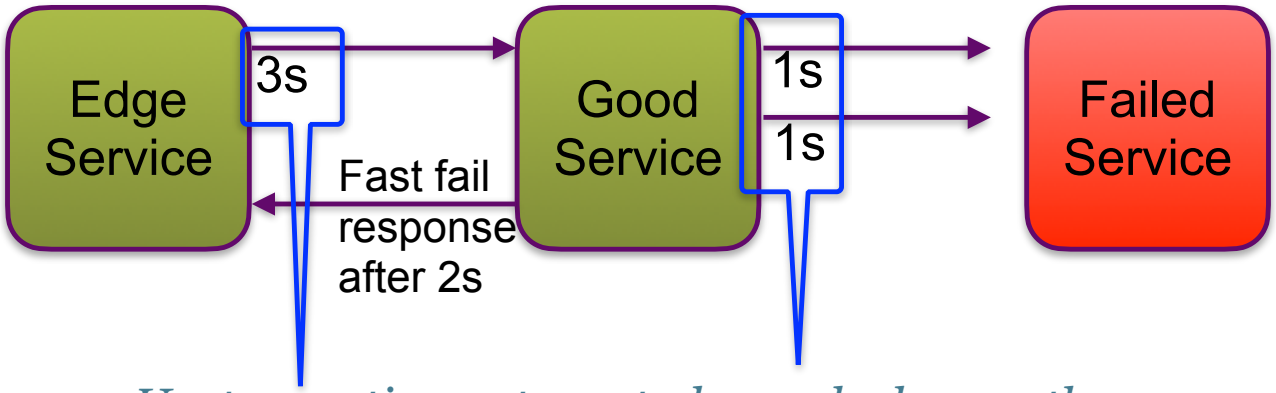
Cascading timeout budget
Static settings that decrease from the edge
or dynamic budget passed with request

How often do retries actually succeed?
Don't ask the same instance the same thing
Only retry on a different connection

@adrianco
BV
Battery Ventures

Timeouts and Retries

Budgeted timeout, one retry



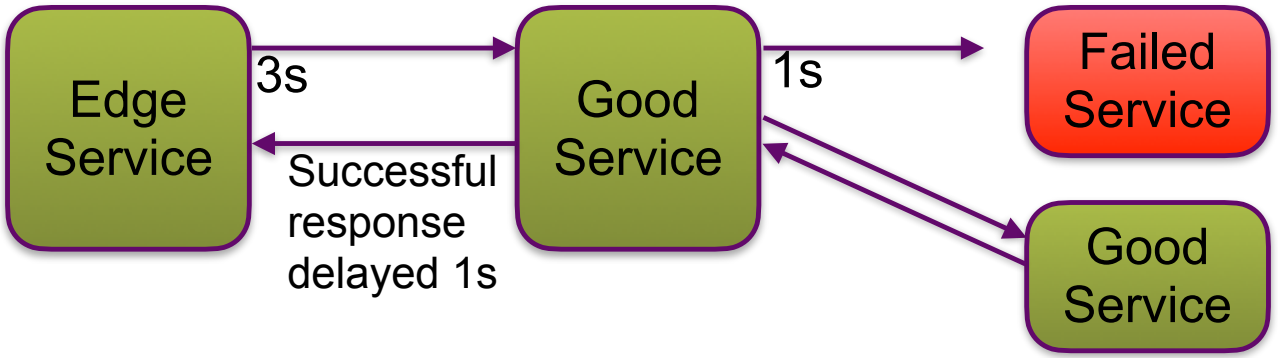
*Upstream timeout must always be longer than total downstream timeout * retries delay*

No unproductive work while fast failing

@adrianco
BV
Battery Ventures

Timeouts and Retries

Budgeted timeout, failover retry

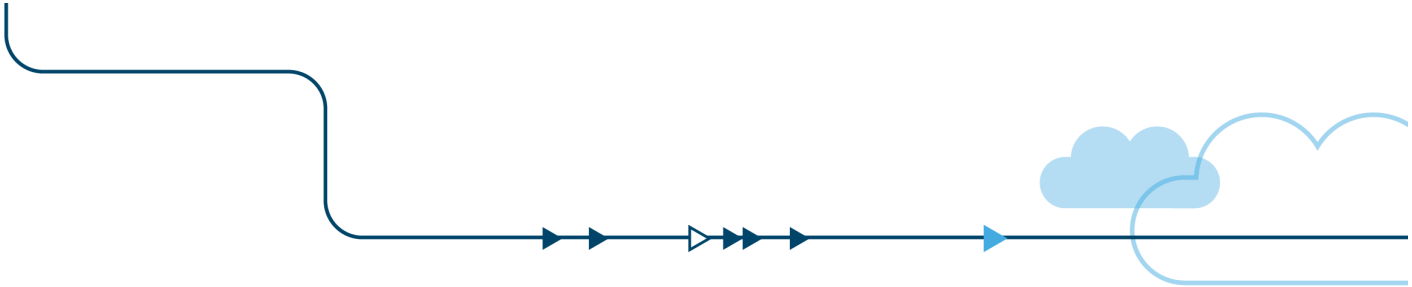


For replicated services with multiple instances never retry against a failed instance

No extra retries or unproductive work

These lessons weren't learned, most microservices frameworks still suffer from retry storms as far as I can tell

@adrianco
BV Battery Ventures



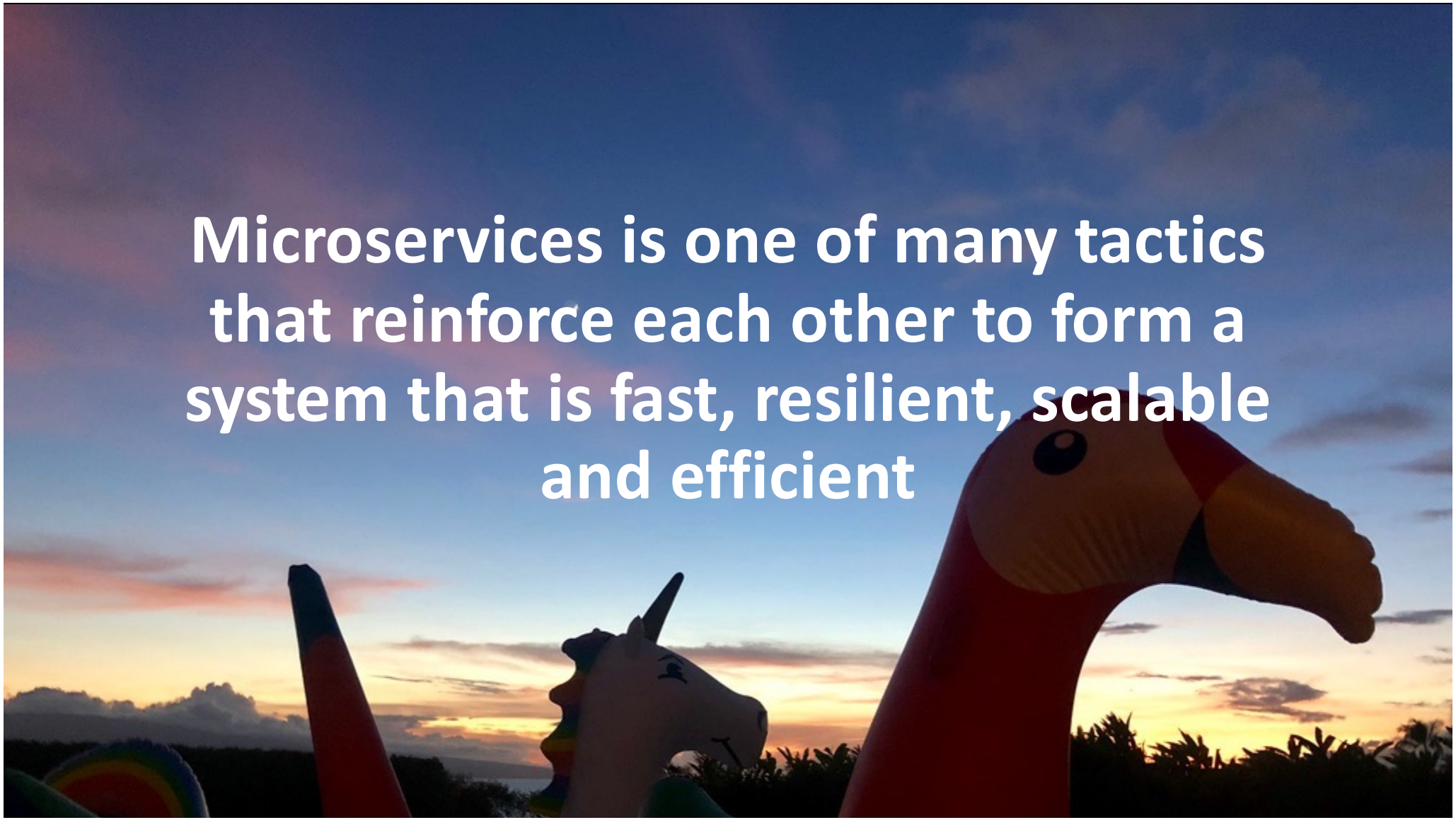
"We see the world as increasingly more complex and chaotic because we use inadequate concepts to explain it. When we understand something, we no longer see it as chaotic or complex."

Jamshid Gharajedaghi - 2011
Systems Thinking: Managing Chaos and Complexity: A Platform for Designing Business Architecture

@adrianco
BV
Battery Ventures

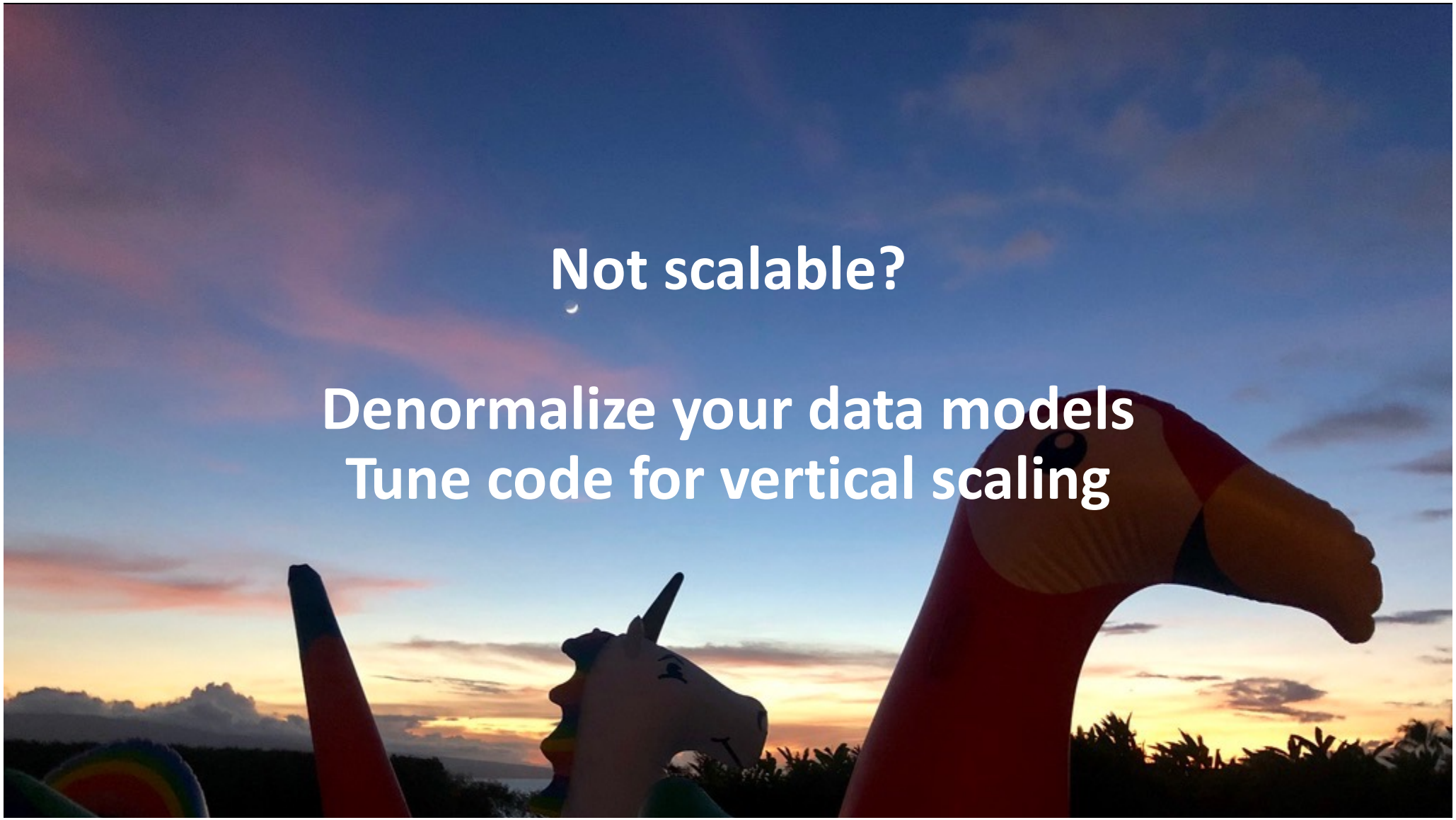

We tried microservices and it didn’t work!

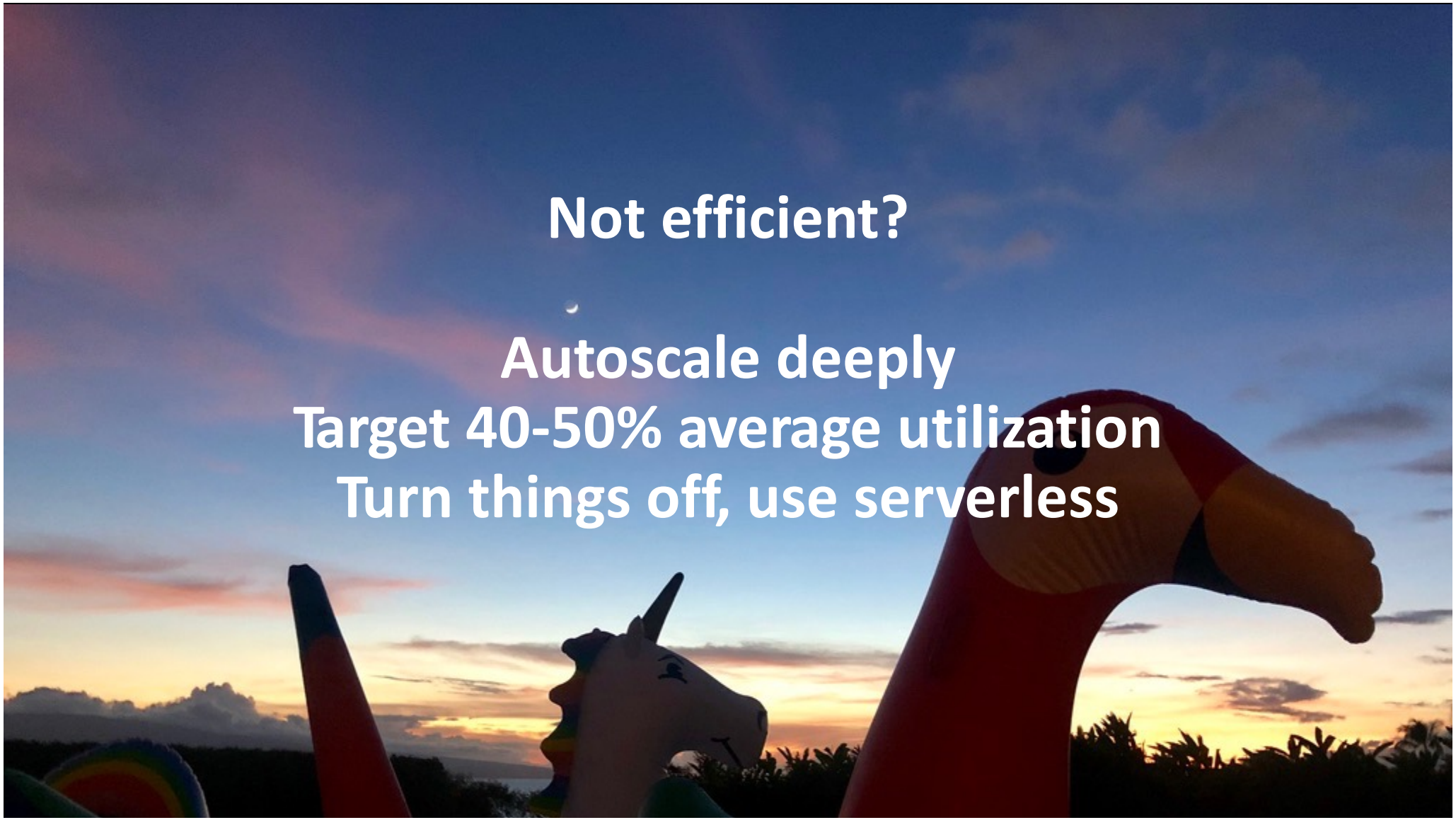

Are you a unicorn or a donkey with a carrot tied to it’s forehead?

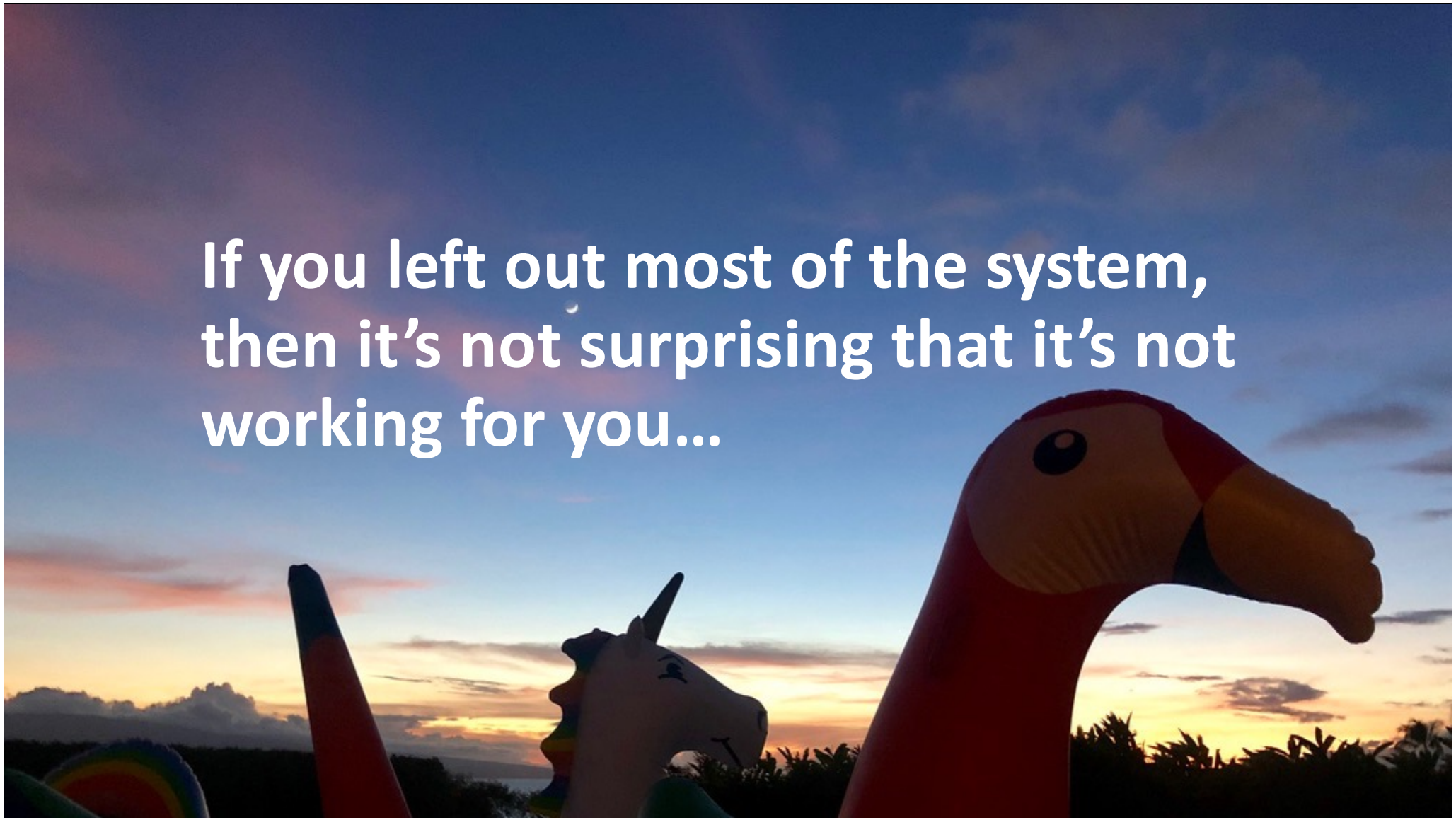

Microservices is one of many tactics that reinforce each other to form a system that is fast, resilient, scalable and efficient


Too slow?
Stop using chatty/inefficient protocols
Pick your system boundaries better


Not resilient?
Fix retry storms
Put developers on call
Use chaos engineering to leverage AZs properly


Not scalable?
Denormalize your data models
Tune code for vertical scaling


Not efficient?
Autoscale deeply
Target 40-50% average utilization
Turn things off, use serverless


If you left out most of the system, then it’s not surprising that it’s not working for you…


Backlog

Adrian Cockcroft - @adrianco

Systems Thinking at Netflix and Beyond

Netflix systematically drives higher talent density via company culture optimized for "fully formed adults"

Netflix is running an Olympic sports team, not a family

“We can’t copy Netflix because it has all those superstar engineers, we don’t have the people”

Fortune 100 CTO after a Netflix presentation - 2013

“We hired them from you, and got out of their way…”

Adrian Cockcroft - in response

"Purposeful systems, representing the systems view of development, assumes plurality in all three dimensions: function, structure, and process."

Jamshid Gharajedaghi - 2011
Systems Thinking: Managing Chaos and Complexity: A Platform for Designing Business Architecture

See also W. Ross Ashby's Law of Requisite Variety
Simplification and efficiency drives can go too far

If you want to build a ship,
don't drum up the people
to gather wood, divide the
work, and give orders.
Instead, teach them to yearn
for the vast and endless sea.

-Antoine De Saint-Exupery,
Author of The Little Prince

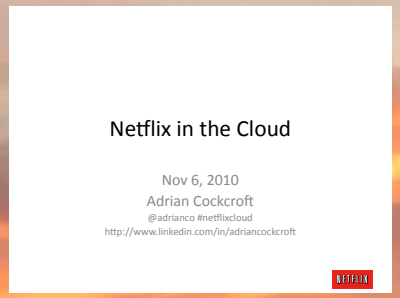
NETFLIX

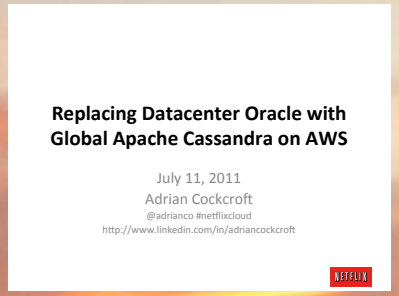
"The basic thesis of this article is that organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations."

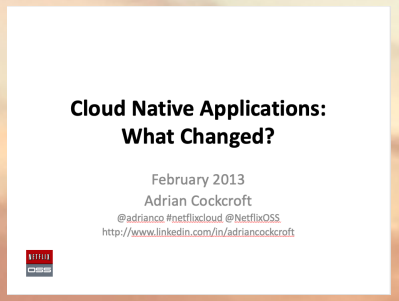
Melvin E. Conway - Conway's Law
How Do Committees Invent? - 1968

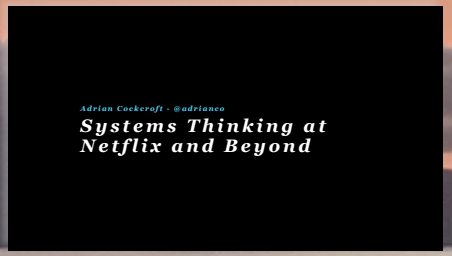
Ruth and Mel Conway 2019

This talk features excerpts from these decks

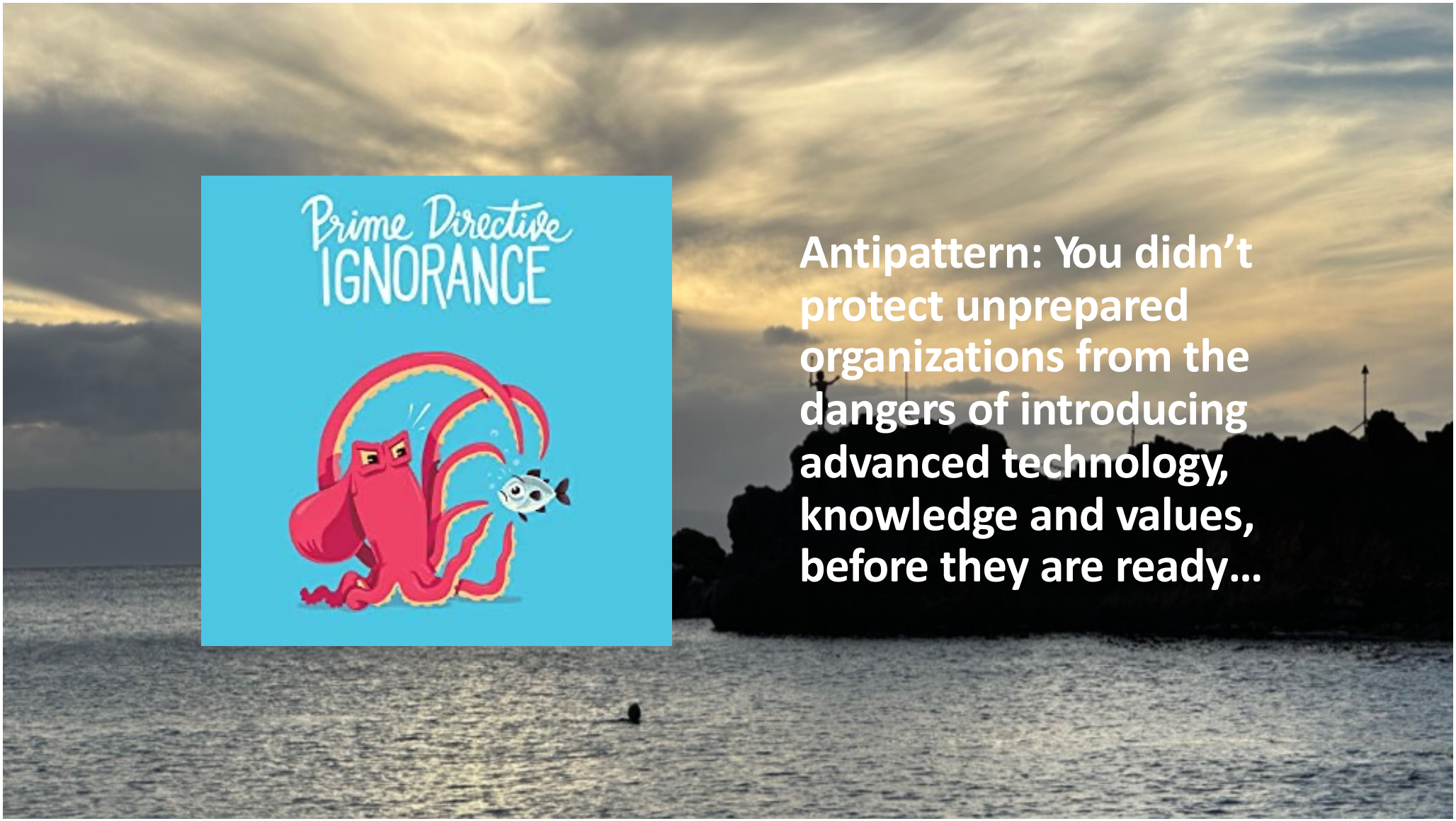








https://www.slideshare.net/adrianco - Netflix era decks
https://www.slideshare.net/adriancockcroft - Battery Ventures decks
https://github.com/adrianco/slides - AWS decks and later including a pdf of these slides!
https://www.youtube.com/@adriancockcroft



Antipattern: You didn't protect unprepared organizations from the dangers of introducing advanced technology, knowledge and values, before they are ready…


Microservices Retrospective - What we learned (and didn't learn) from Netflix
Adrian Cockcroft | Qcon London | March 2023
@adrianco@mastodon.social