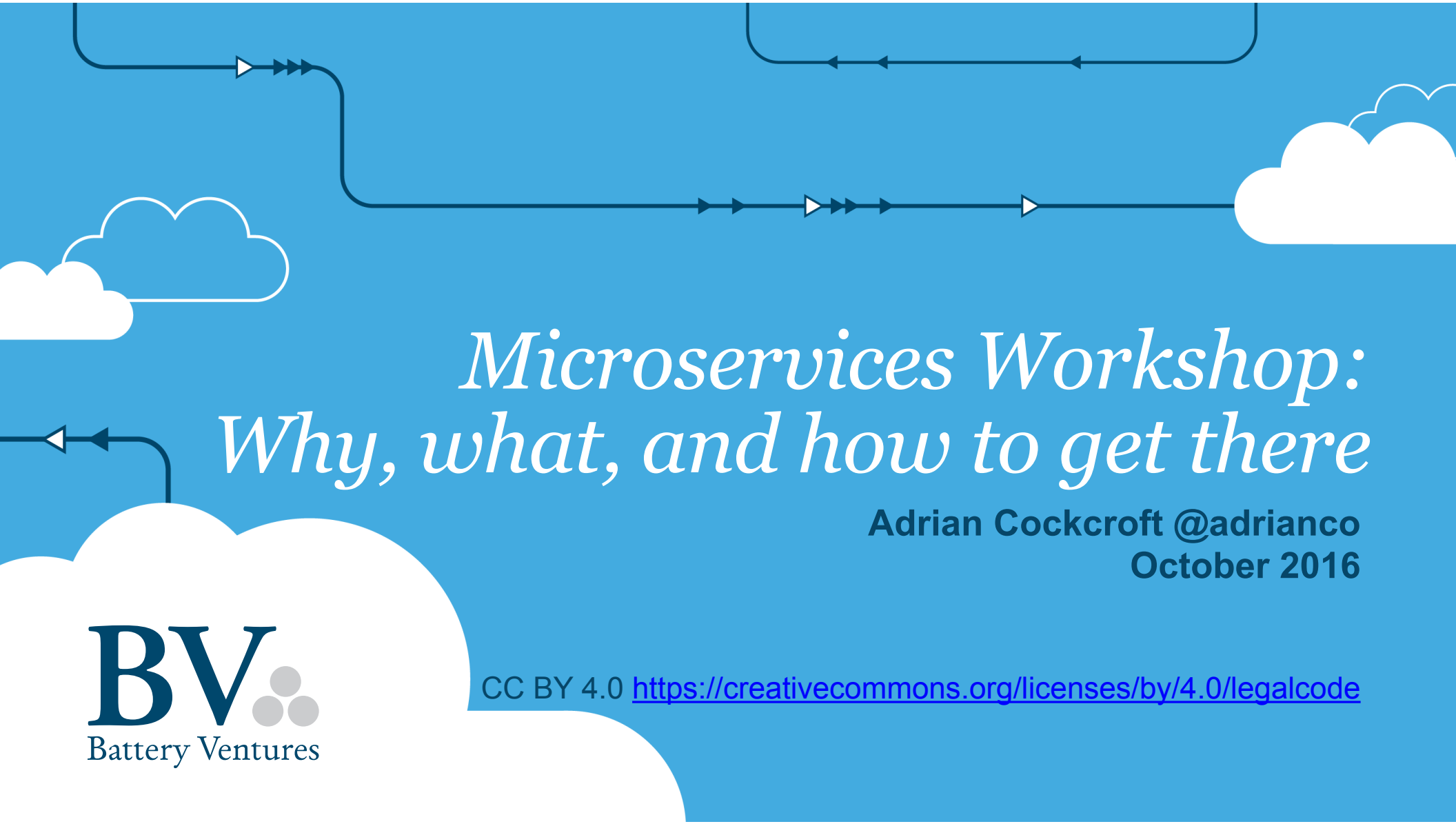
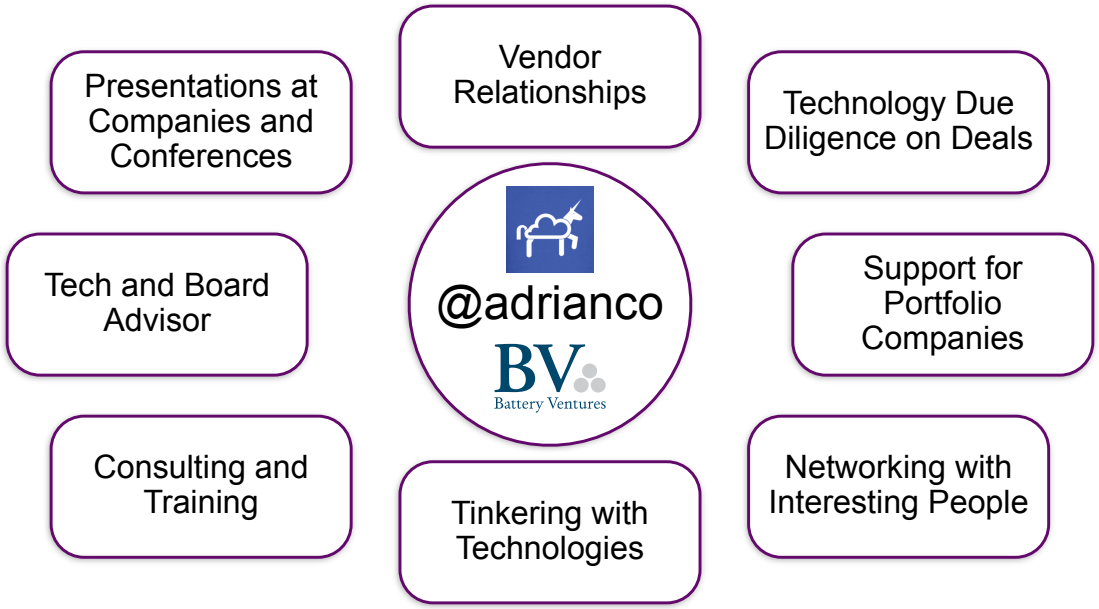


Microservices Workshop: Why, what, and how to get there

Adrian Cockcroft @adrianco

October 2016

What does @adrianco do?



Previously: Netflix, eBay, Sun Microsystems, Cambridge Consultants, City University London - BSc Applied Physics

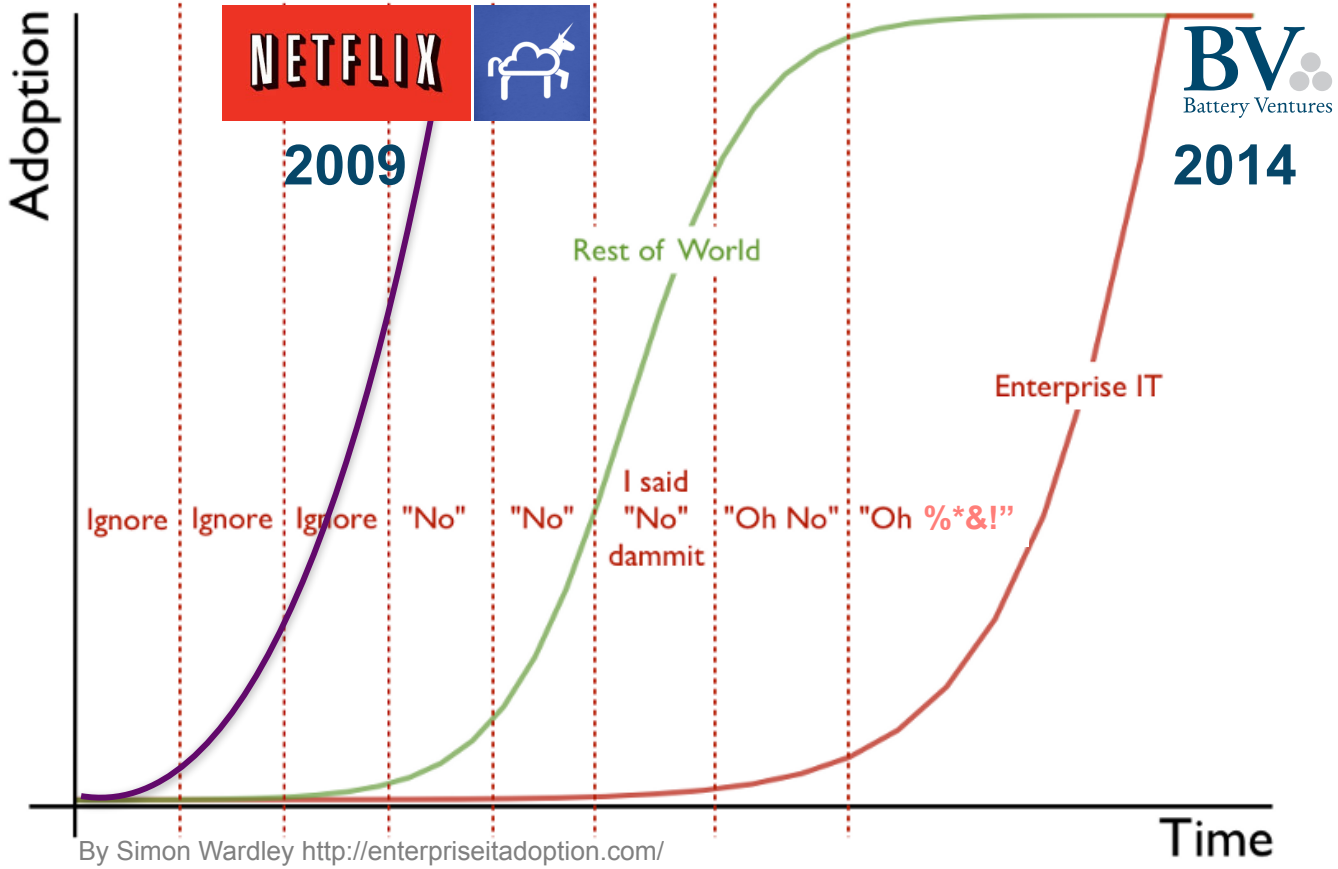
Topics

Scene Setting
State of the Cloud
What Changes?
Product Processes
Microservices
State of the Art
Segmentation
What's Missing?

Monitoring
Challenges
Migration
Response Times
Serverless
Lock-In
Teraservices
Wrap-Up

Setting the Scene

Why am I here?



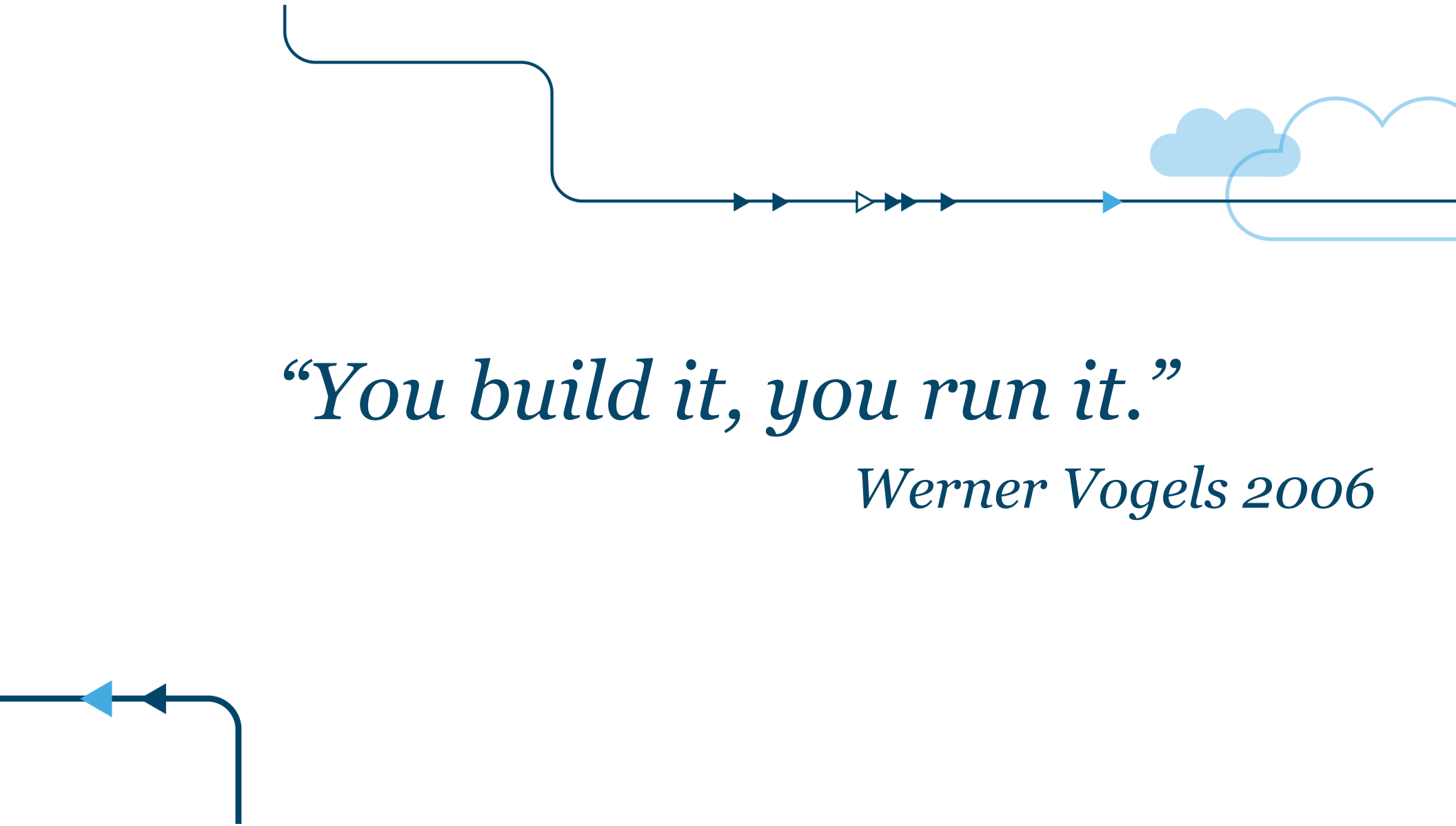
@adrianco's job at the intersection of cloud and Enterprise IT, looking for disruption and opportunities.

Disruptions in 2016 coming from server-less computing and teraservices.

Typical reactions to my Netflix talks…

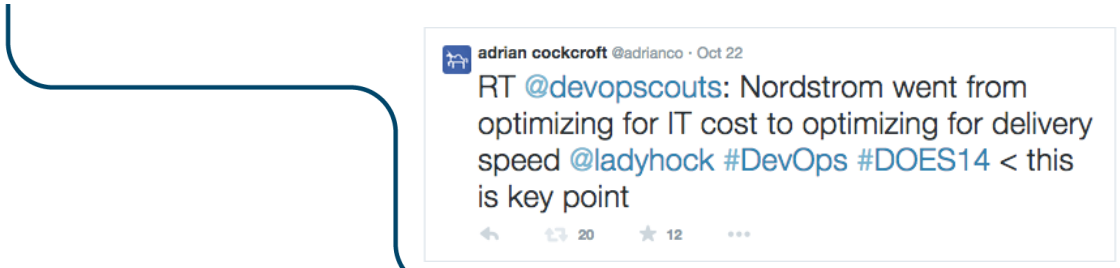
What I learned from my time at Netflix

- *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*



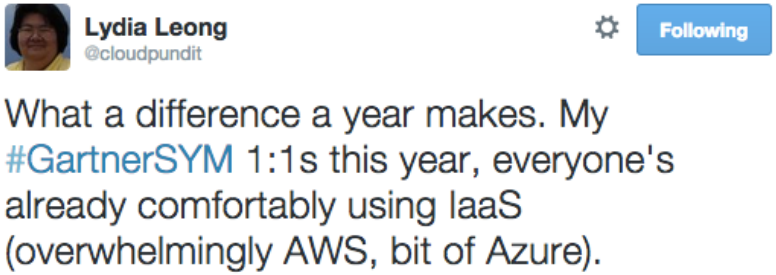
“You build it, you run it.”

Werner Vogels 2006



In 2014 Enterprises finally embraced public cloud and in 2015 began replacing entire datacenters.

Oct 2014



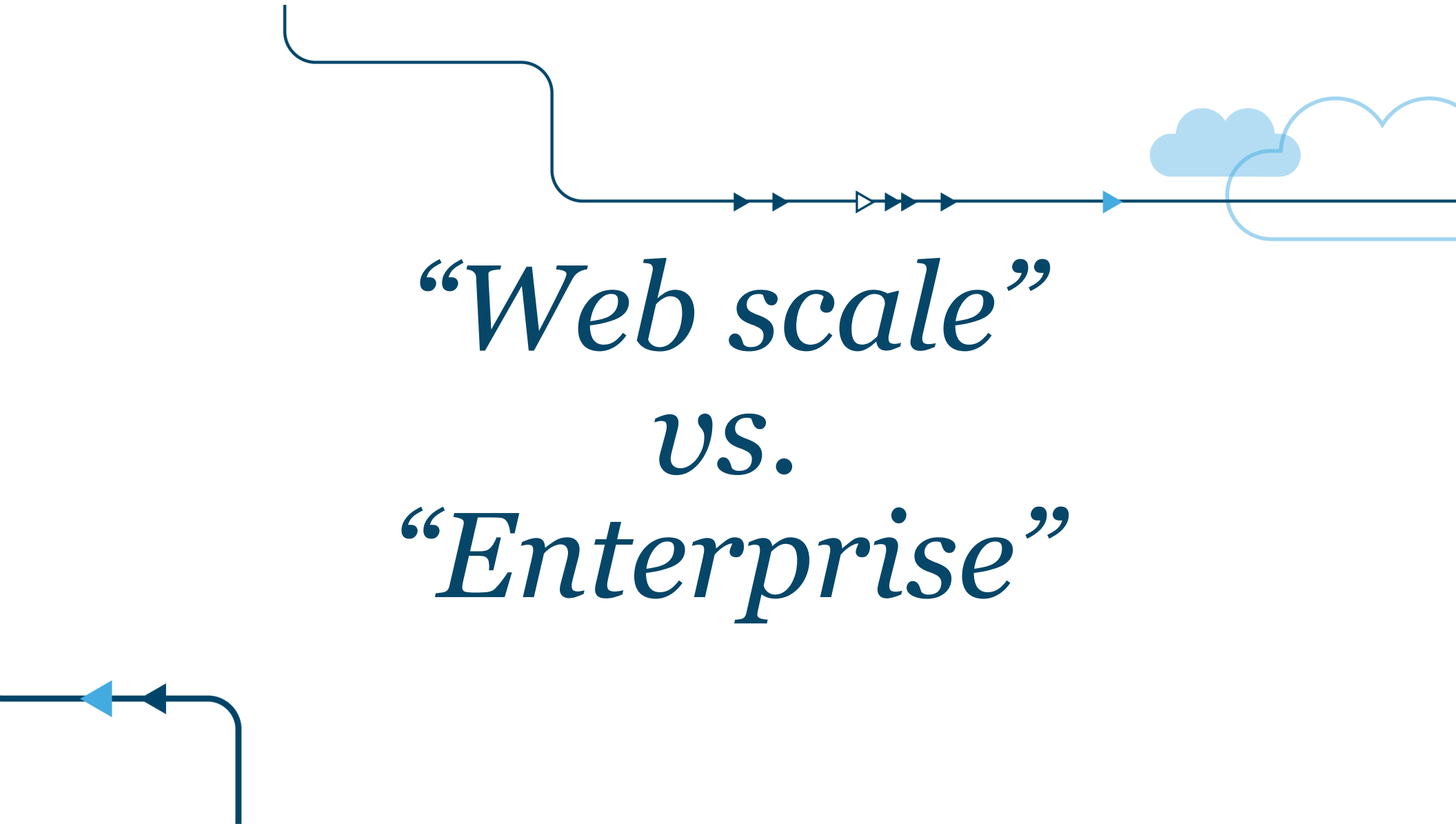
Oct 2015

Key Goals of the CIO?
Align IT with the business
Develop products faster
Try not to get breached

Security Blanket Failure

Insecure applications hidden behind firewalls make you feel safe until the breach happens…

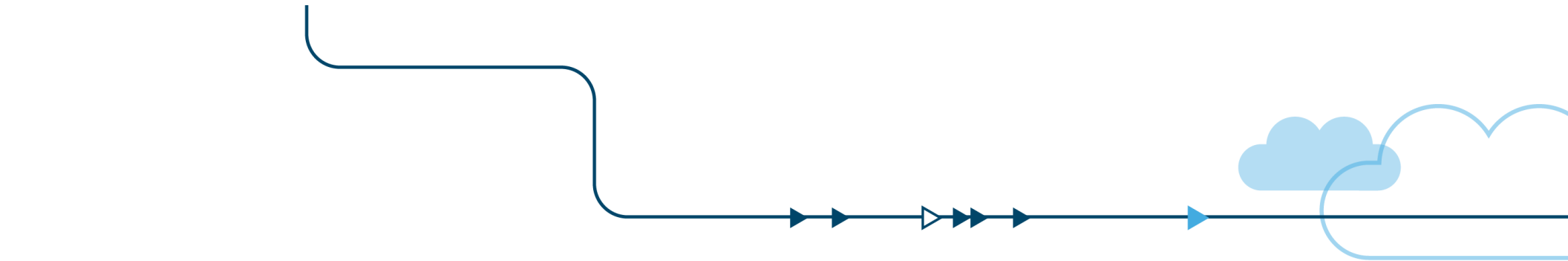
http://peanuts.wikia.com/wiki/Linus'_security_blanket



“Web scale” vs. “Enterprise”

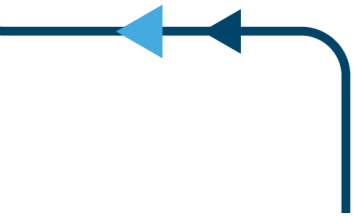
“Webscale”

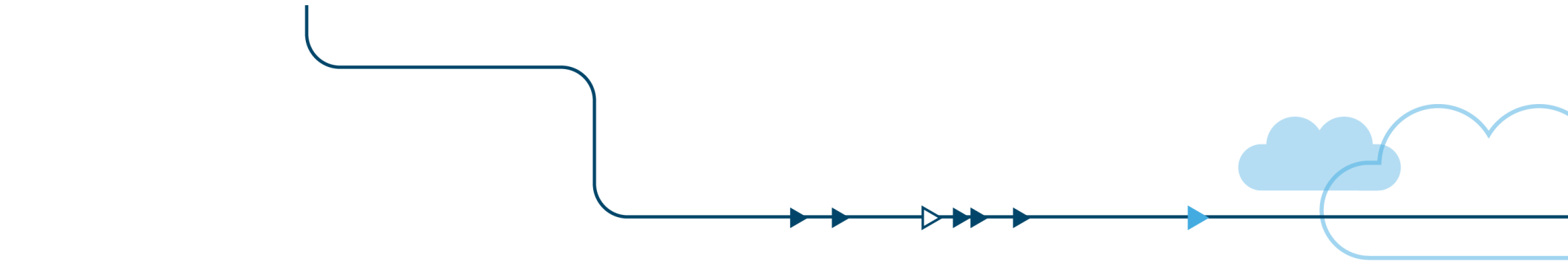
Freedom and responsibility
High trust



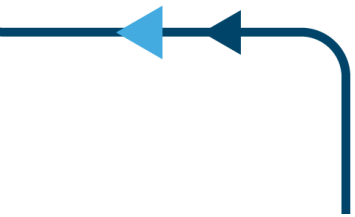
“Enterprise”

Bureaucracy and blame
Low trust





How can everyone get speed, low cost, and better usability?



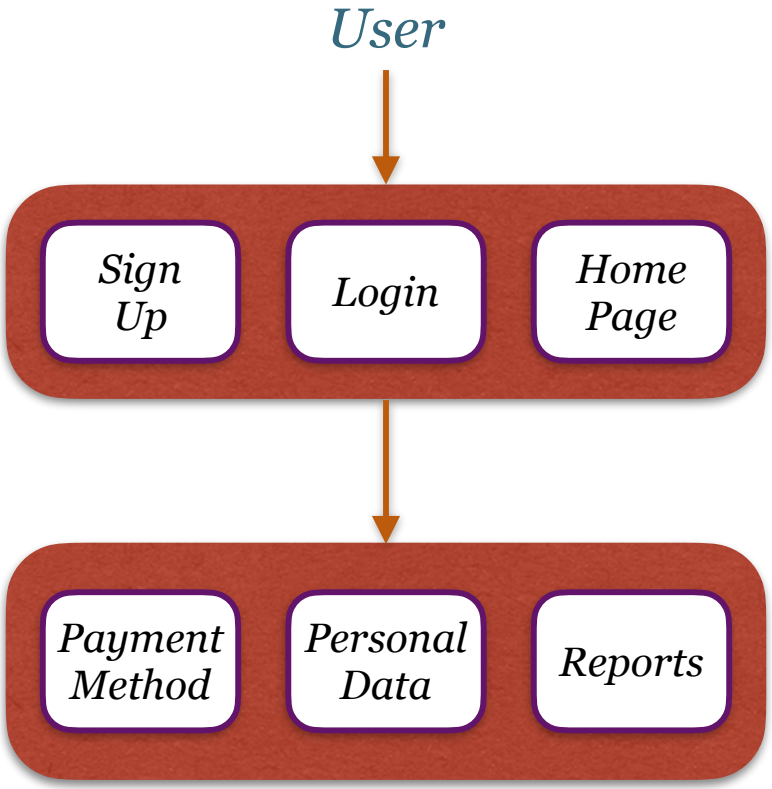
Mixed methods: Disaggregation into microservices helps!

Example Monolith:

Monolithic application

Complex mix of queries

Monolithic "kitchen sink" database



Because one part of the monolithic application and database holds sensitive data all of it is subject to the most rigorous policies

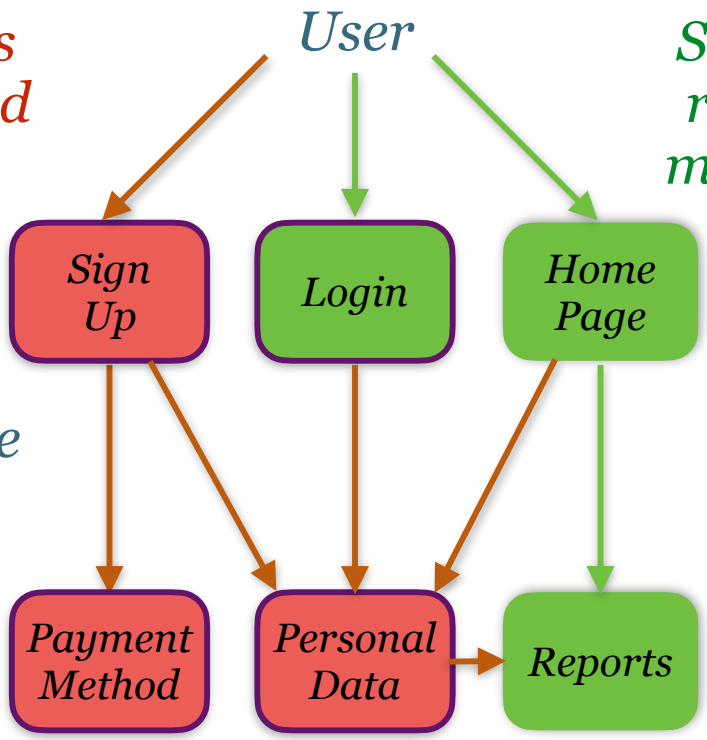
Microservices version:

Segregated team owns secure data sources and infrequent updates

Microservices separation of concerns

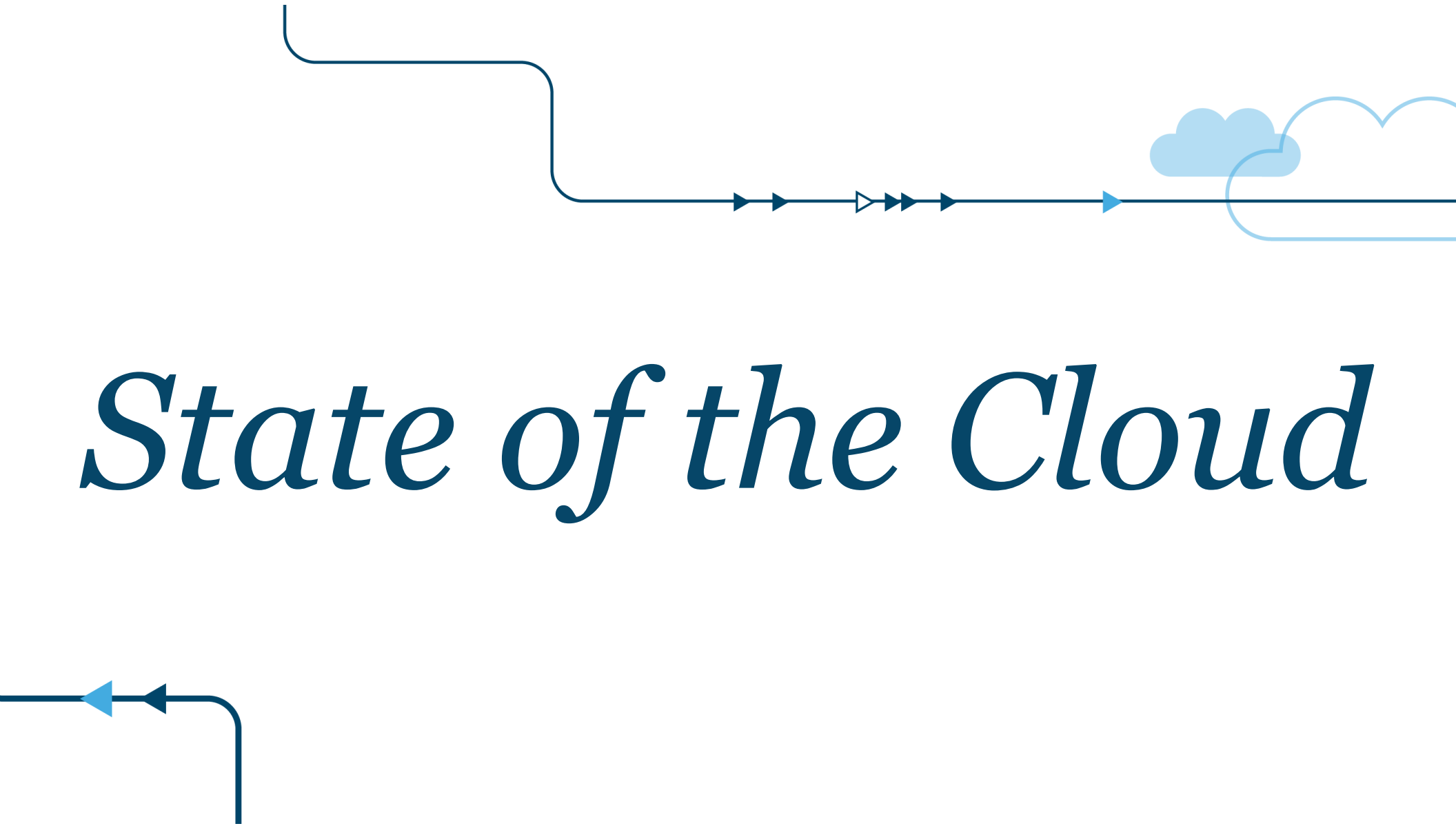
Isolated single purpose connections

Optimized datastores

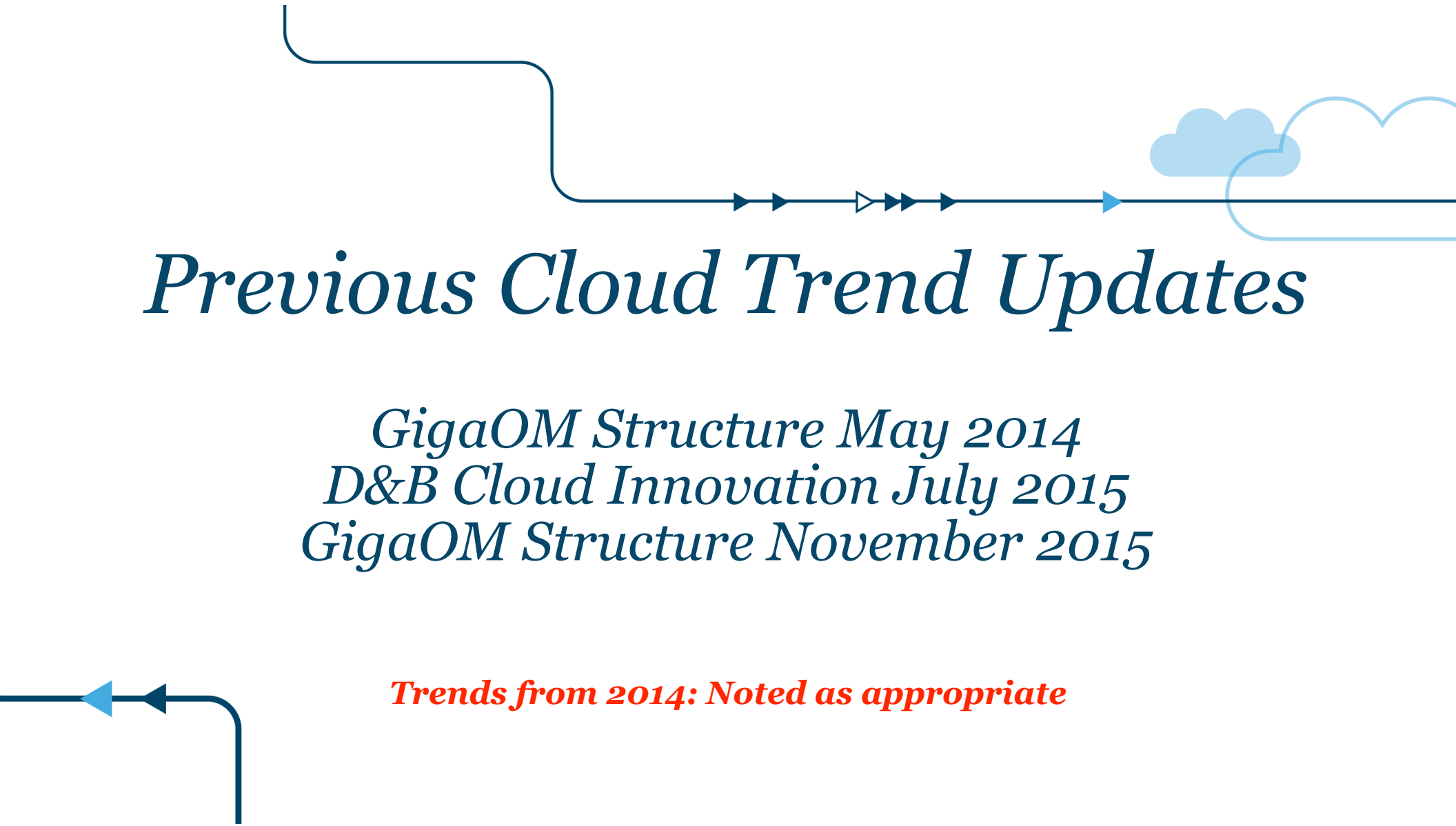


Segregated team owns rapid improvement of most common use cases

Because each microservice can conform to the appropriate policy, demands for agility can be separated from requirements for security



State of the Cloud

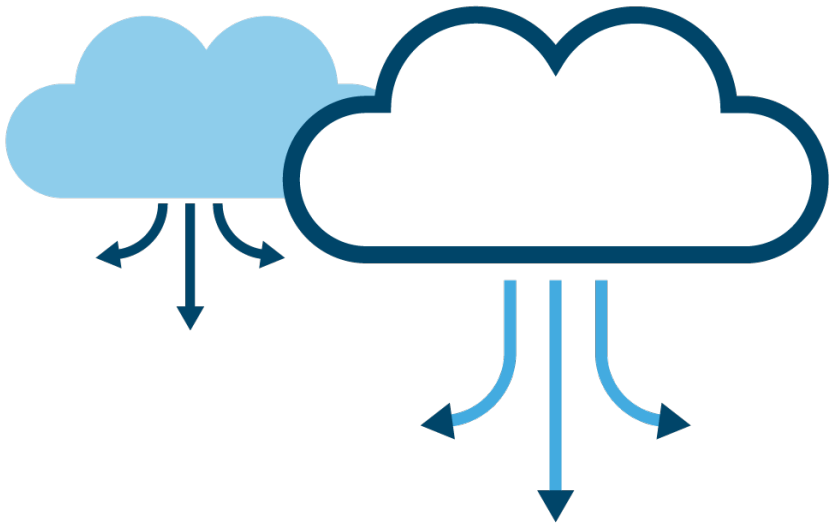


Previous Cloud Trend Updates

GigaOM Structure May 2014
D&B Cloud Innovation July 2015
GigaOM Structure November 2015

Trends from 2014: Noted as appropriate

Cloud Ecosystem Matures



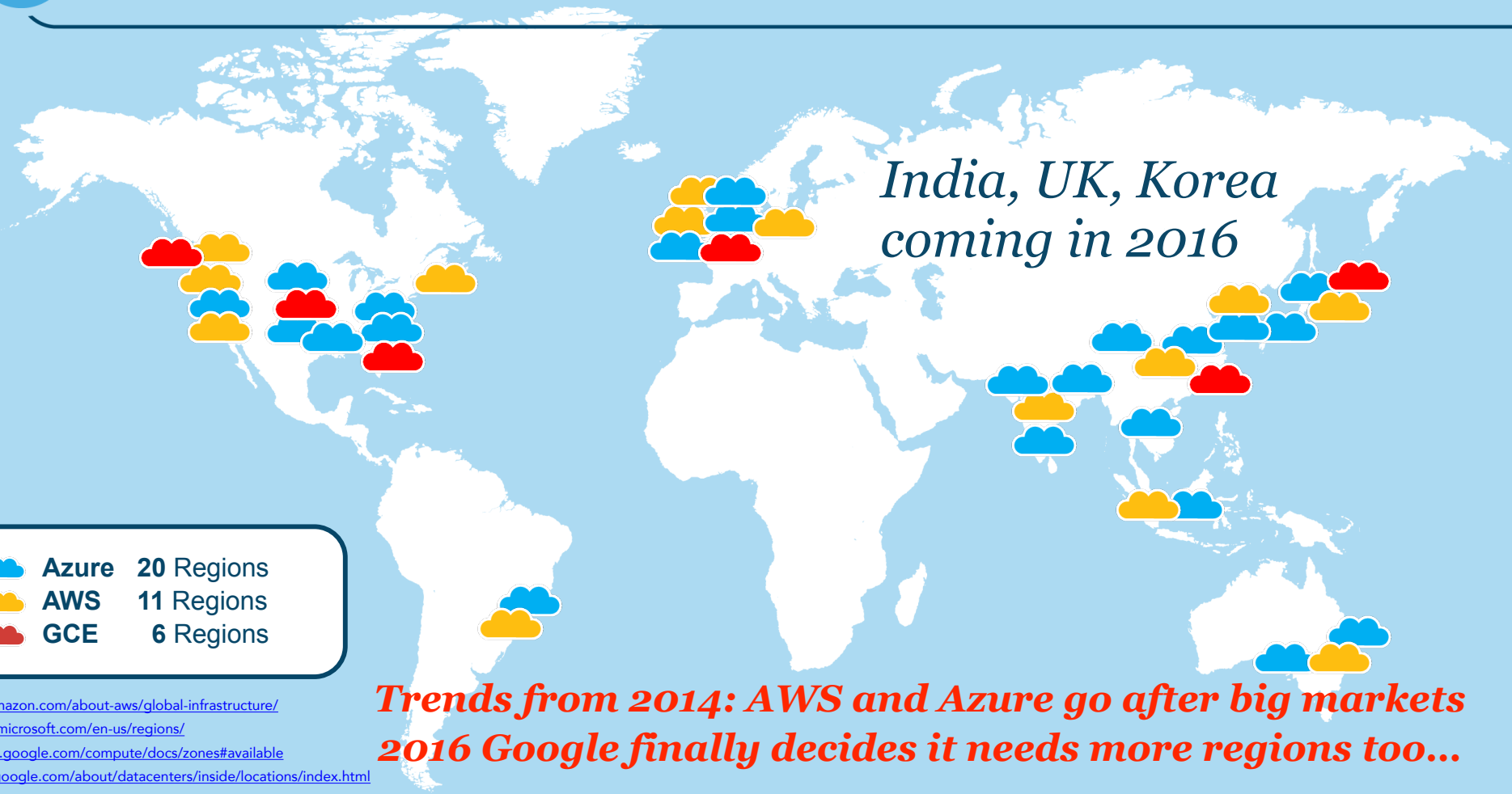
Staying Power
Support
Scale
Location

Trends from 2014: Even more emphasis on location

Safe Bets

Trends from 2014: AWS capacity share vs. Azure still growing
AWS growing 60% year on year, Enterprise share growing fast
Azure providing strong Linux and Open Source support

The Global Land-Grab



http://aws.amazon.com/about-aws/global-infrastructure/
http://azure.microsoft.com/en-us/regions/
https://cloud.google.com/compute/docs/zones#available
http://www.google.com/about/datacenters/inside/locations/index.html

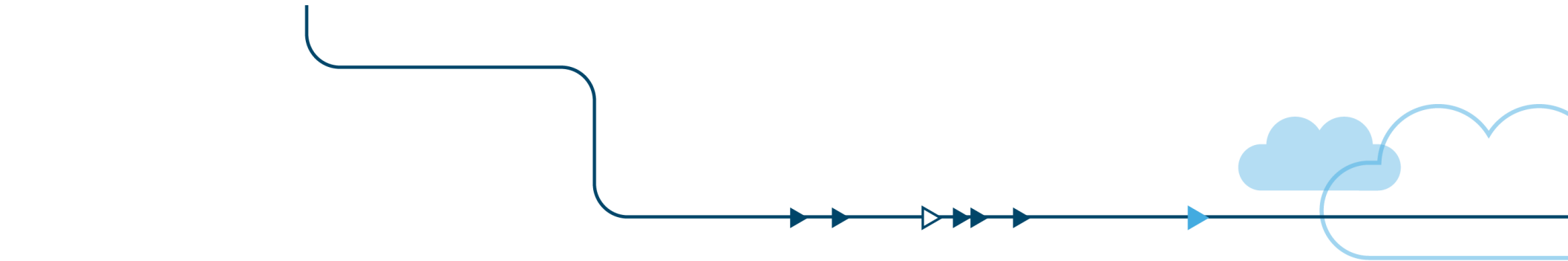
Trends from 2014: AWS and Azure go after big markets
2016 Google finally decides it needs more regions too...

Who is challenging VMware for in-house cloud automation?

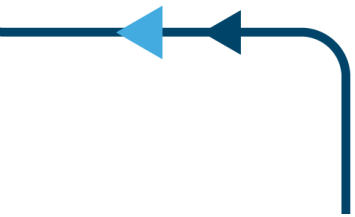
Questions we keep hearing about cloud

Challenges for Private Cloud

Stability
Functionality
Total Cost of Ownership



Some enterprise vendor responses to cloud and container ecosystem growth...

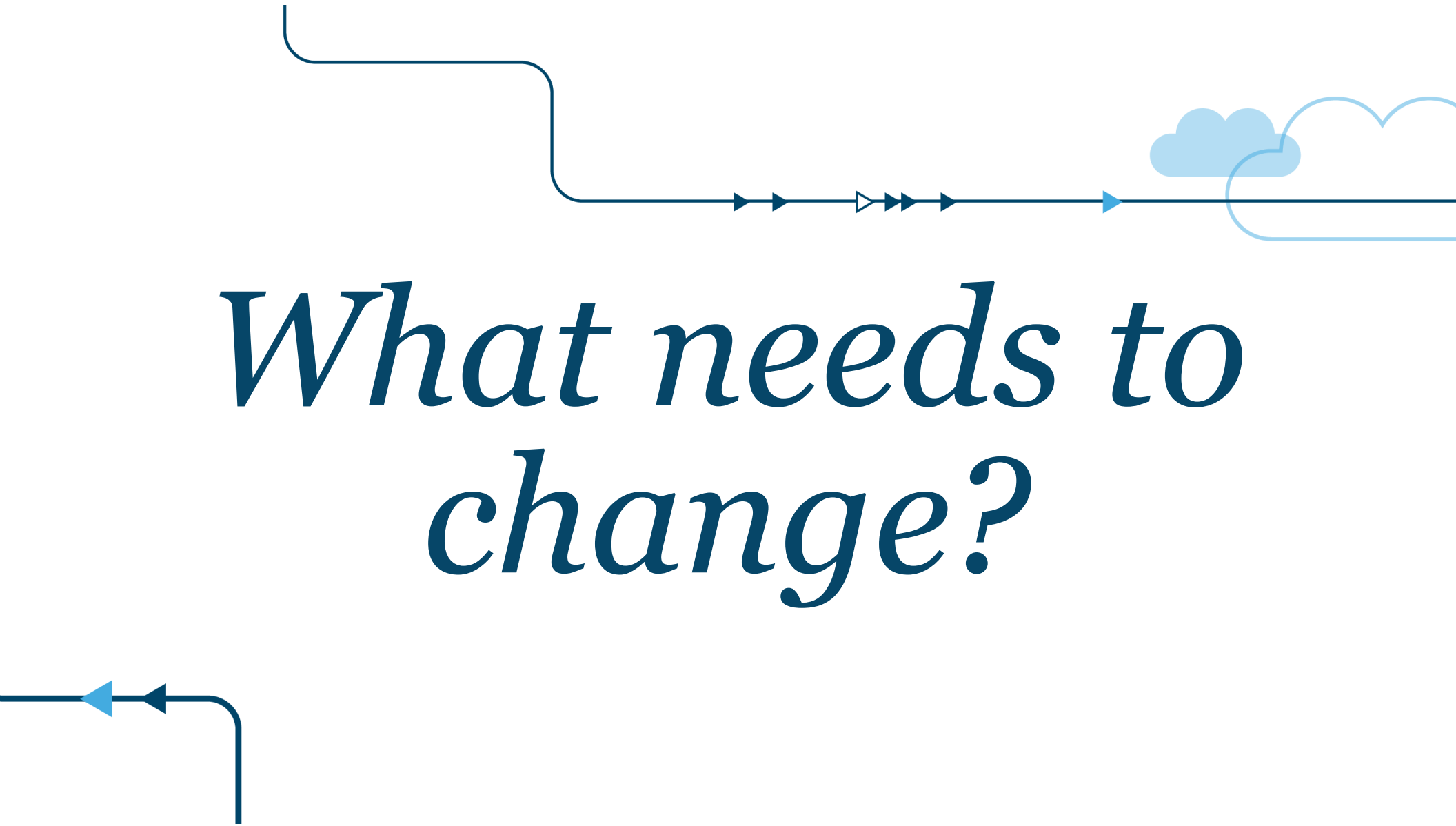


DELL *The ship is sinking, let's re-brand as a submarine!*

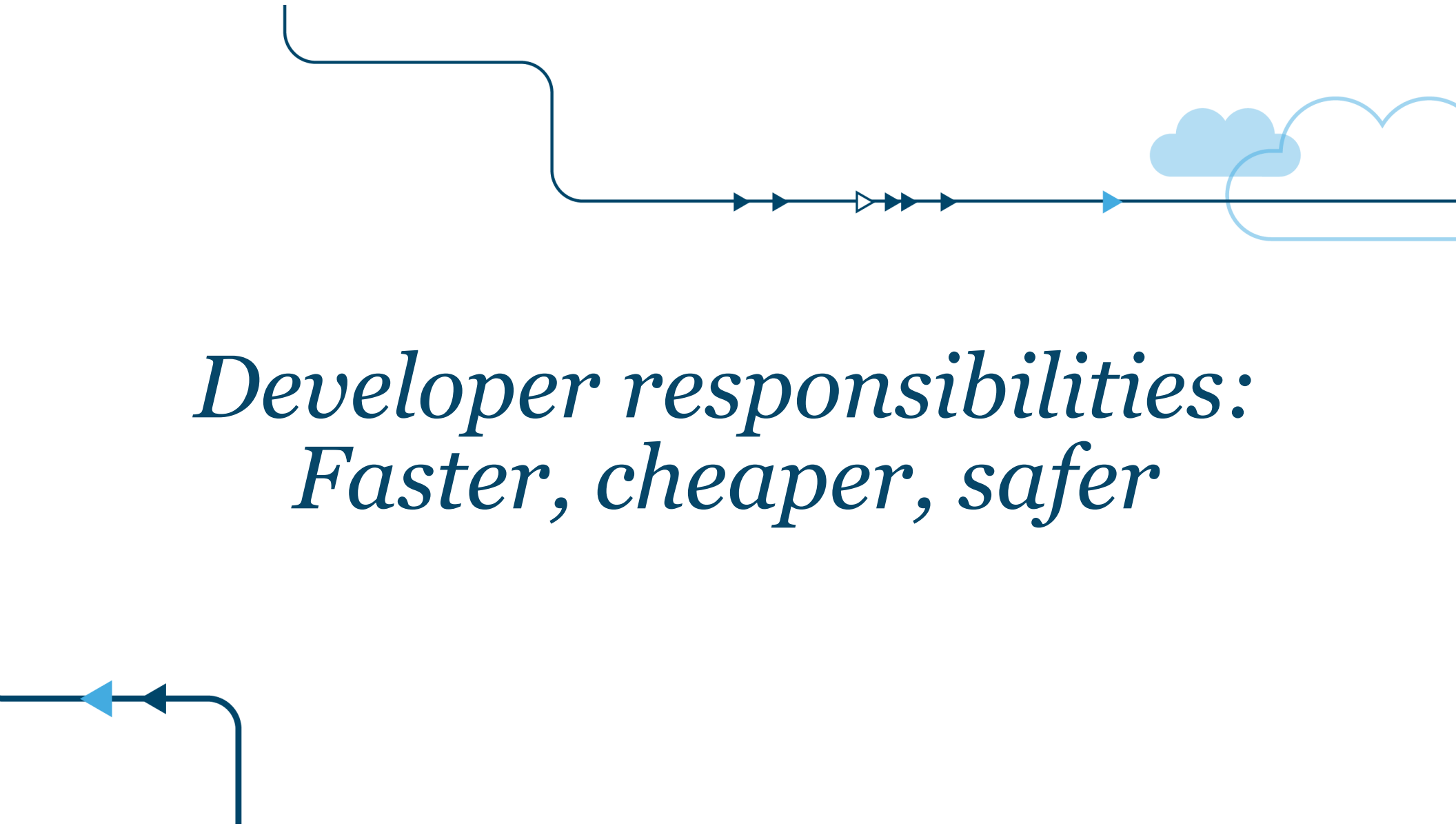
EMC² *The ship is sinking, let's merge with a submarine!*

Look! we cut our ship in two really quickly!

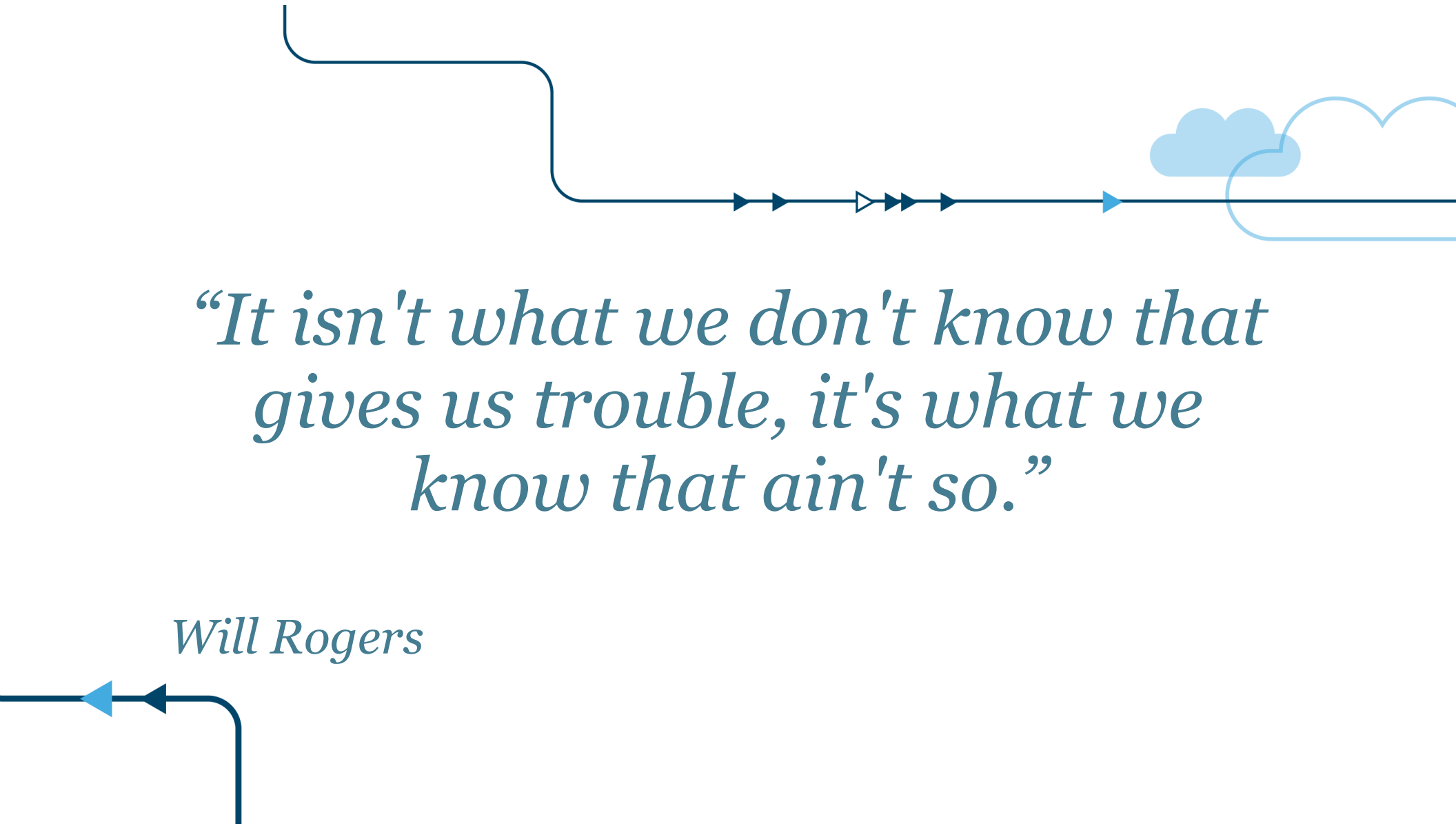
Hewlett Packard Enterprise



What needs to change?

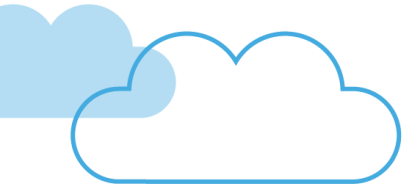


Developer responsibilities: Faster, cheaper, safer

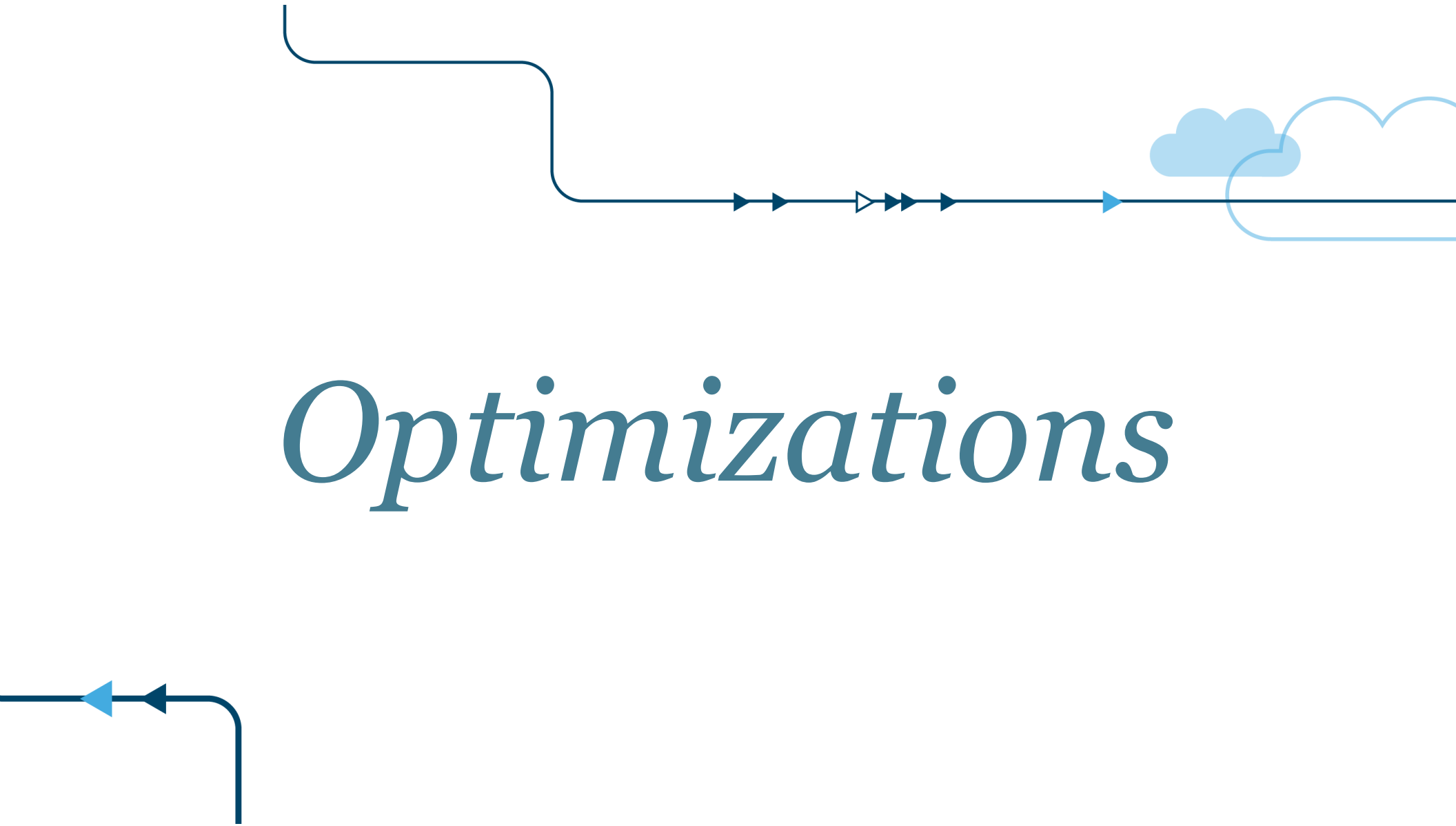


“It isn't what we don't know that gives us trouble, it's what we know that ain't so.”

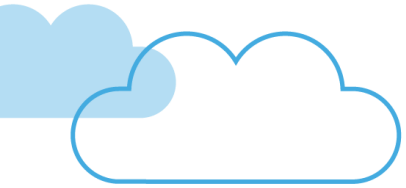
Will Rogers



Assumptions



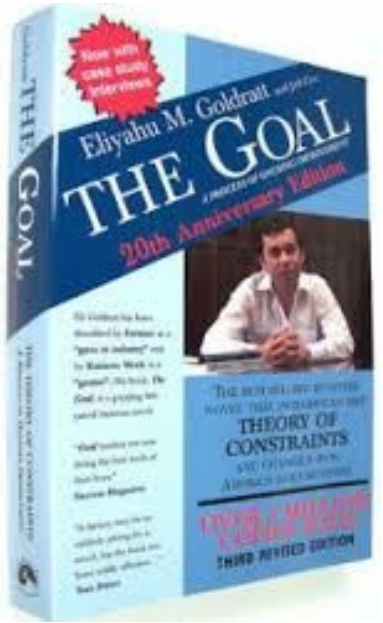
Optimizations



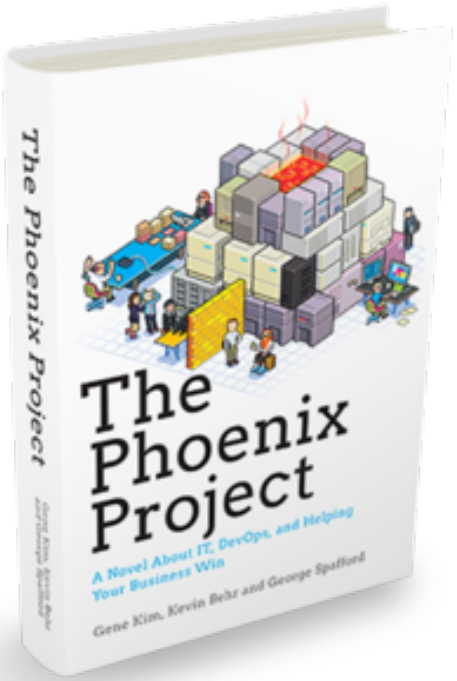

Assumption: Process prevents problems

Organizations build up slow complex "Scar tissue" processes

"This is the IT swamp draining manual for anyone who is neck deep in alligators."



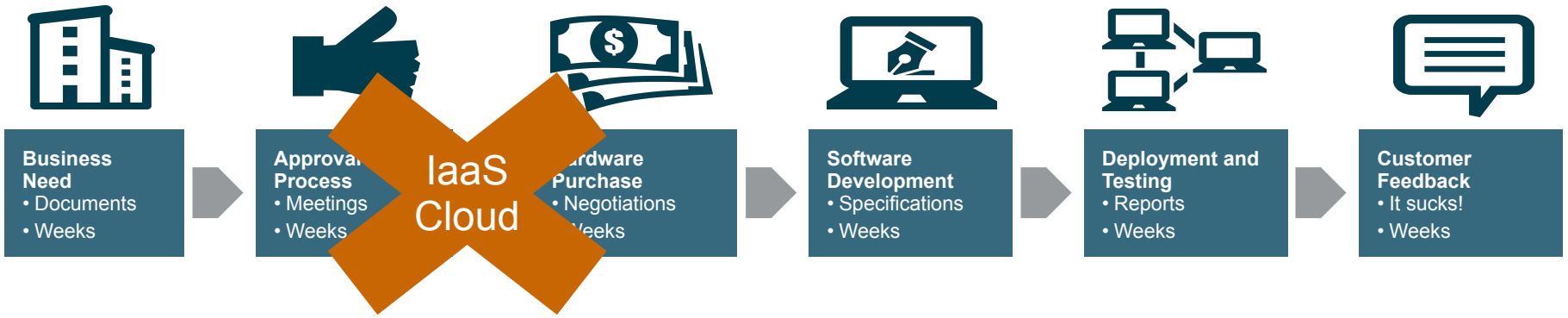
1984



2014

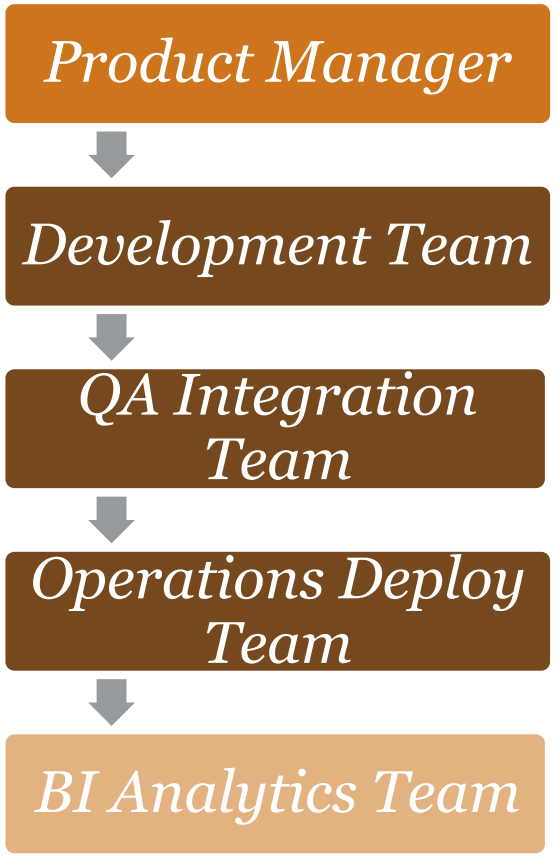
Product Development Processes

Waterfall Product Development



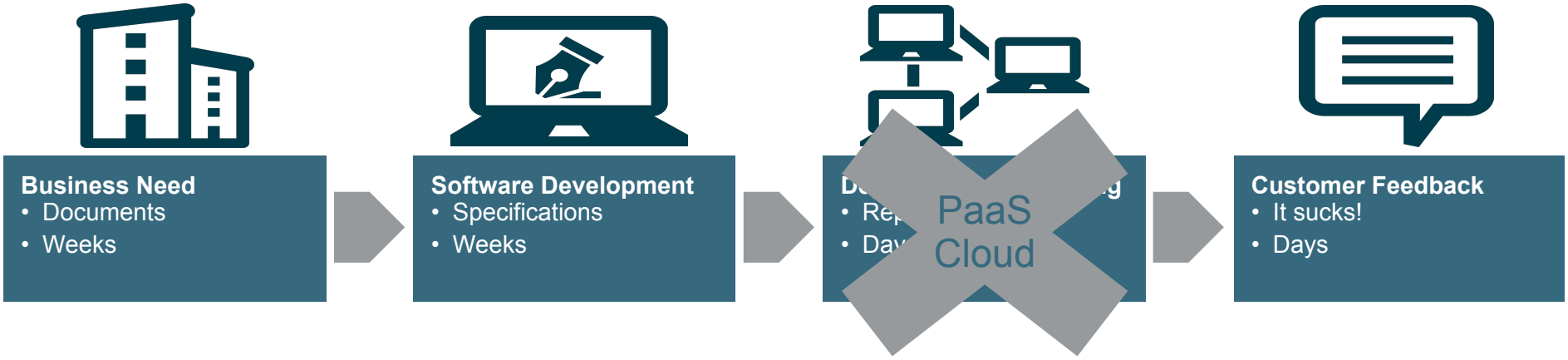
Hardware provisioning is undifferentiated heavy lifting – replace it with IaaS

Process Hand-Off Steps for Agile Development on IaaS



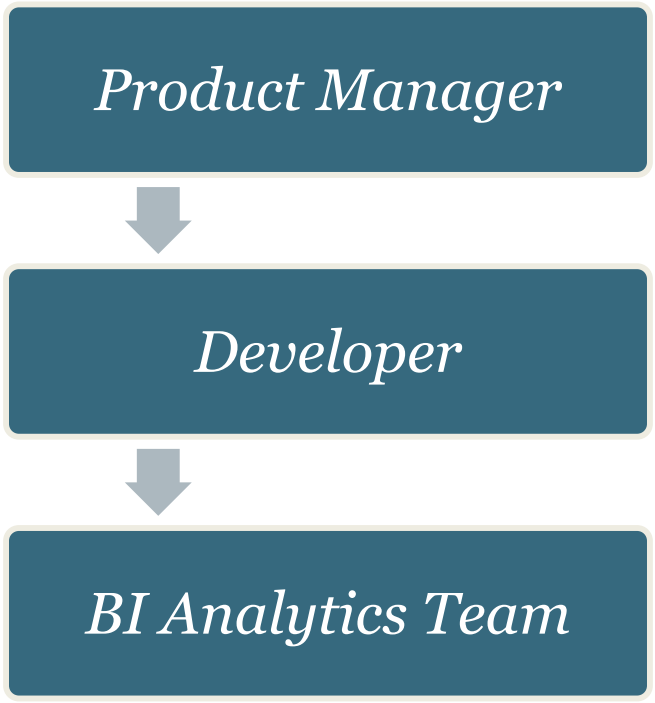
IaaS Agile Product Development

amazon web services™ | Windows Azure | SOFTLAYER an IBM Company | openstack CLOUD SOFTWARE | etc…

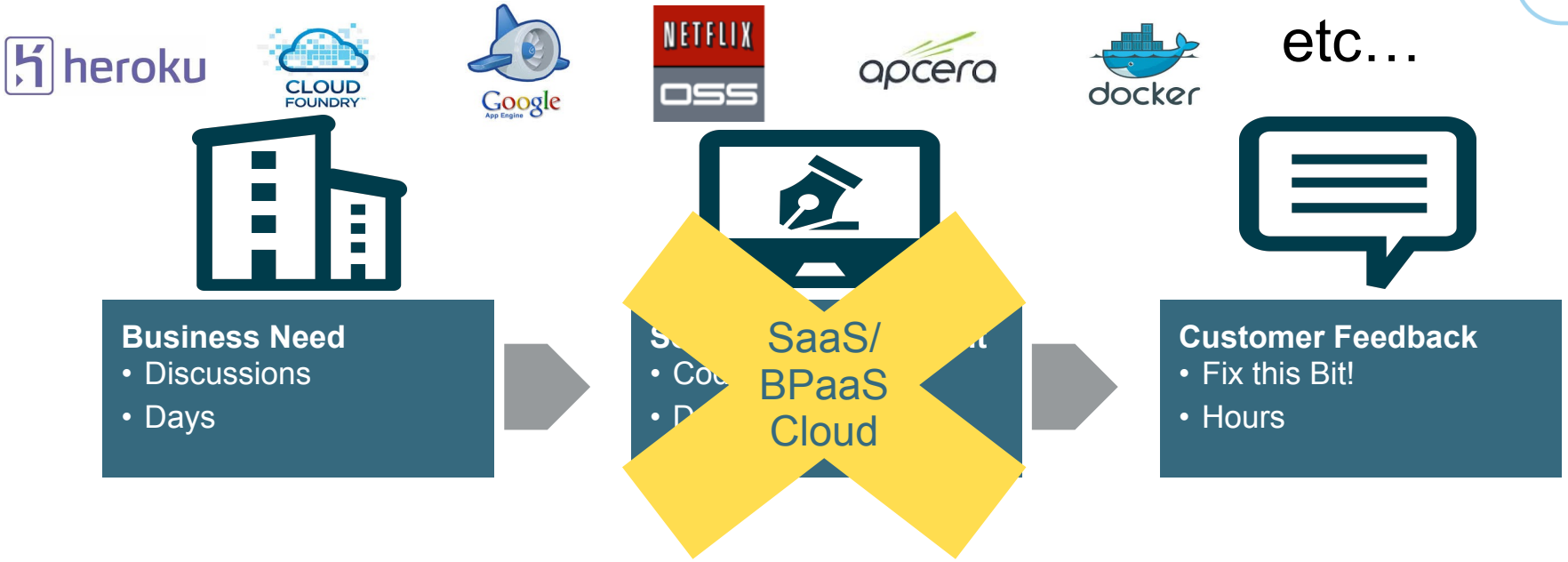


Software provisioning is undifferentiated heavy lifting – replace it with PaaS

Process for Continuous Delivery of Features on PaaS



PaaS CD Feature Development



Building your own business apps is undifferentiated heavy lifting – use SaaS

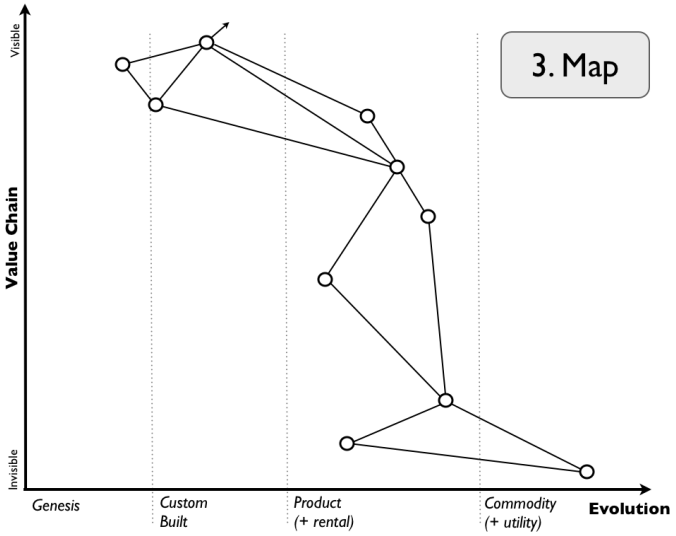
SaaS Based Business Application Development

and thousands more...

Business Need
- GUI Builder
- Hours

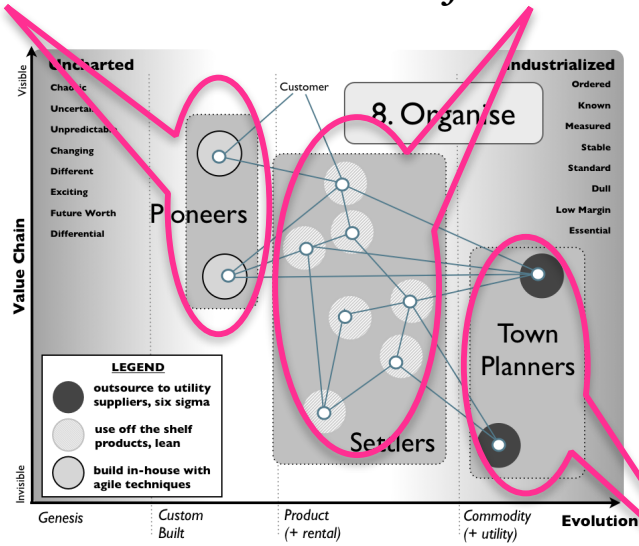
Customer Feedback
- Fix this bit!
- Seconds

Value Chain Mapping



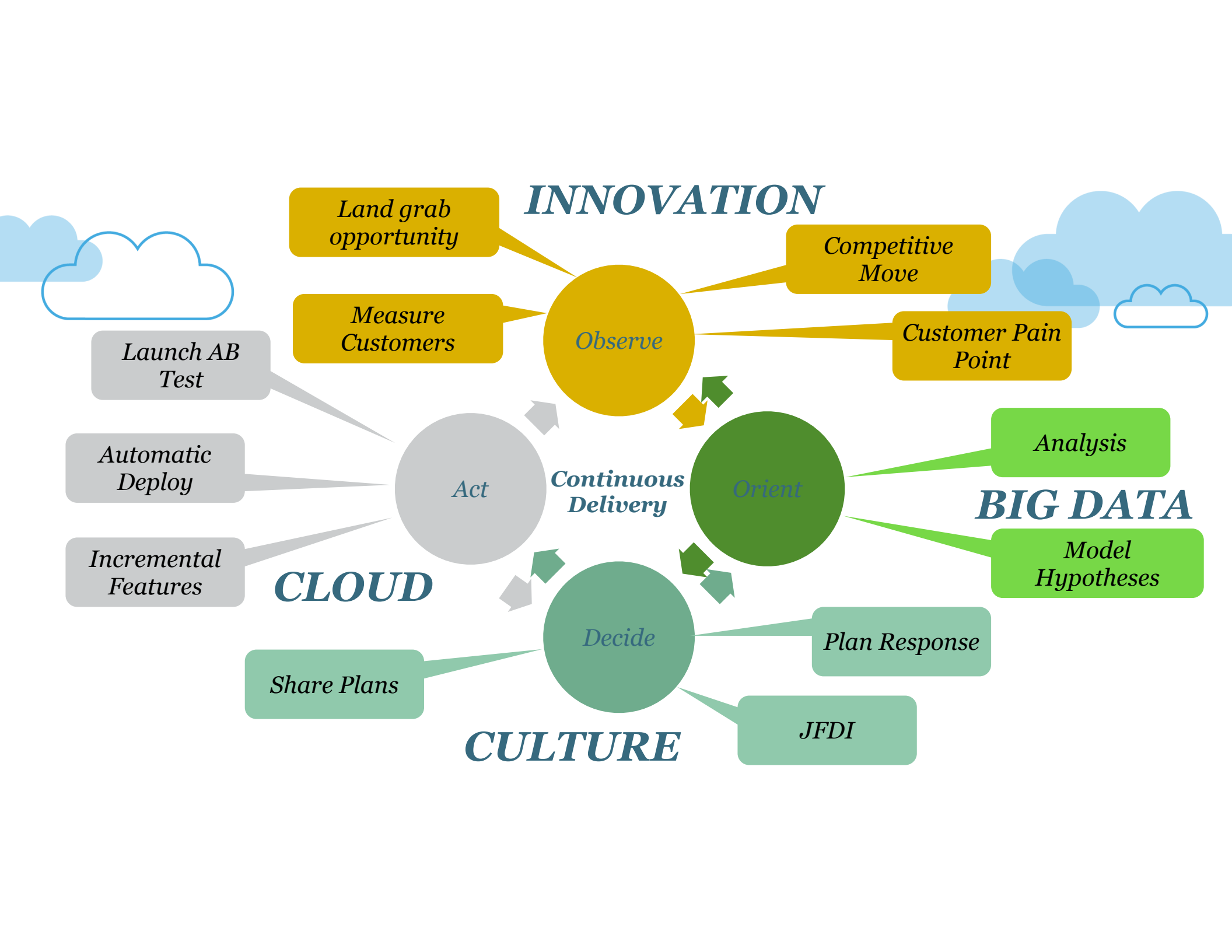
Your unique product - Agile

Best of breed as a Service - Lean

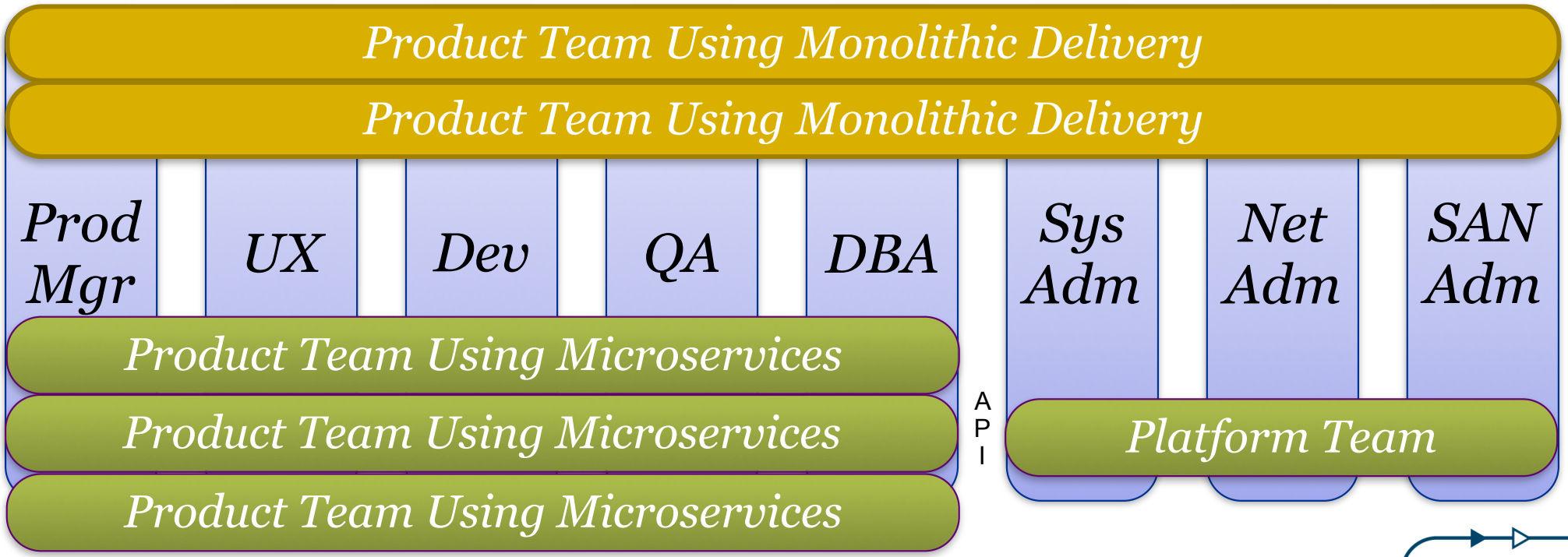


Undifferentiated utility suppliers - 6sigma

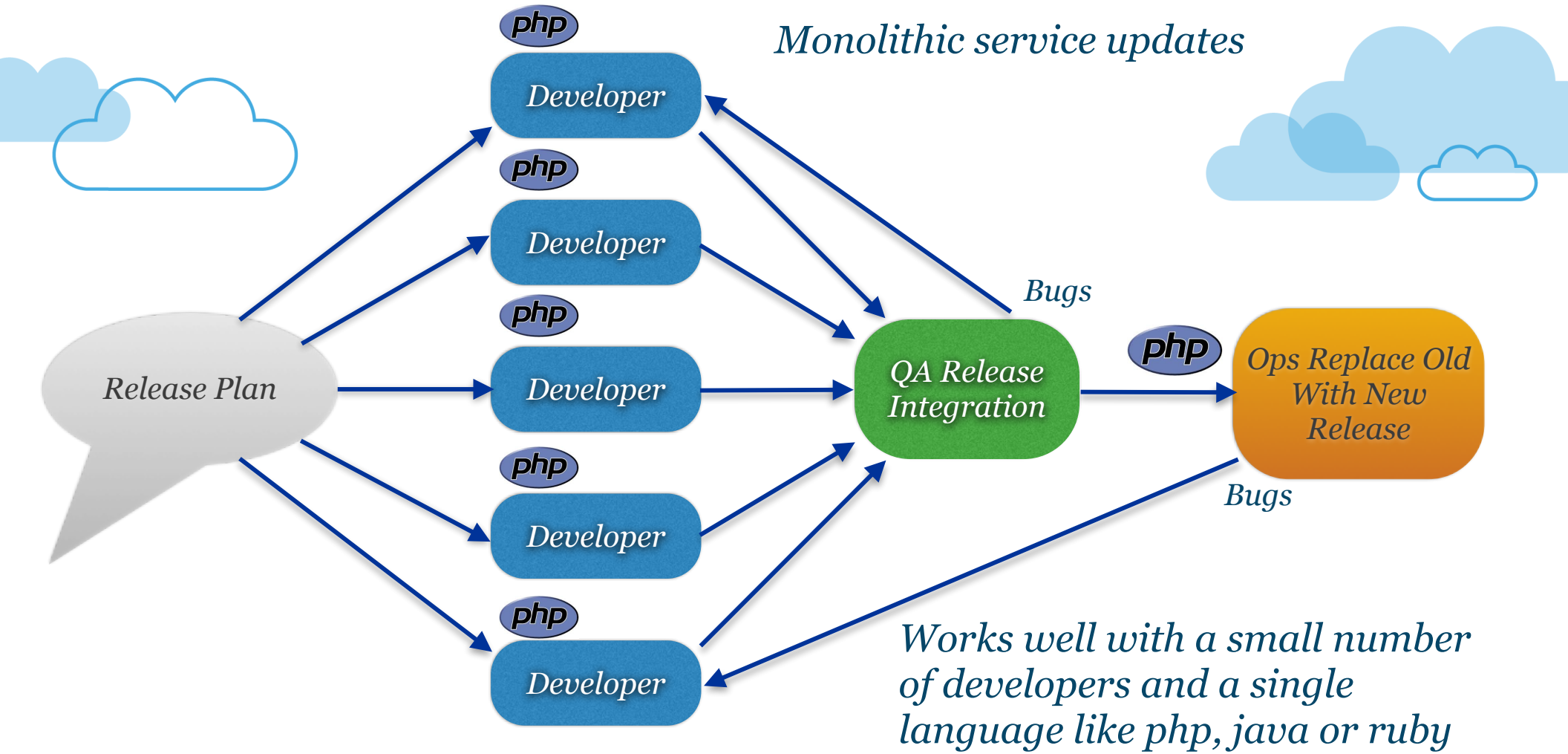
Simon Wardley http://blog.gardeviance.org/2014/11/how-to-get-to-strategy-in-ten-steps.html
Related tools and training http://www.wardleymaps.com/

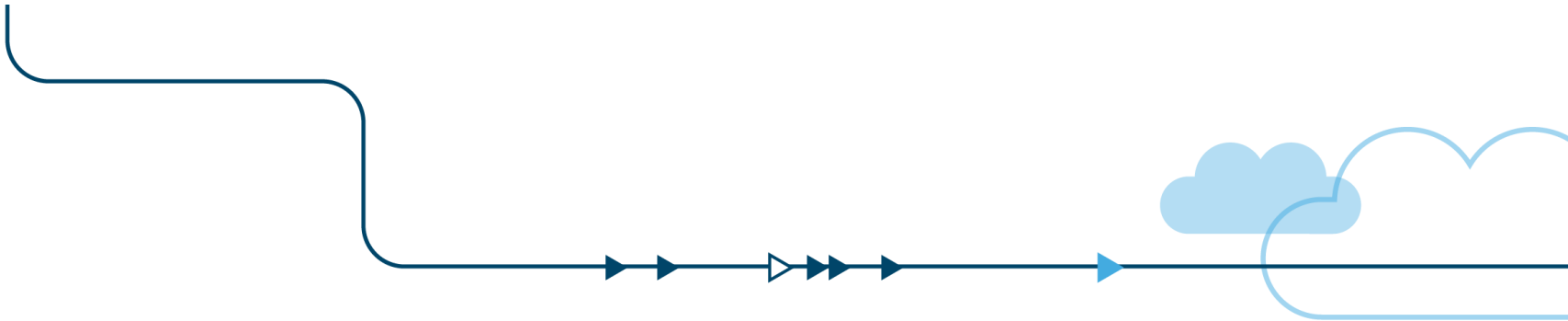

INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Launch AB Test
Automatic Deploy
Incremental Features
Act
Continuous Delivery
Orient
Analysis
BIG DATA
Model Hypotheses
CLOUD
Decide
Plan Response
Share Plans
JFDI
CULTURE

Breaking Down the SILOs

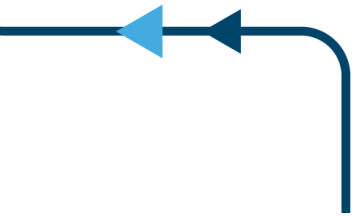


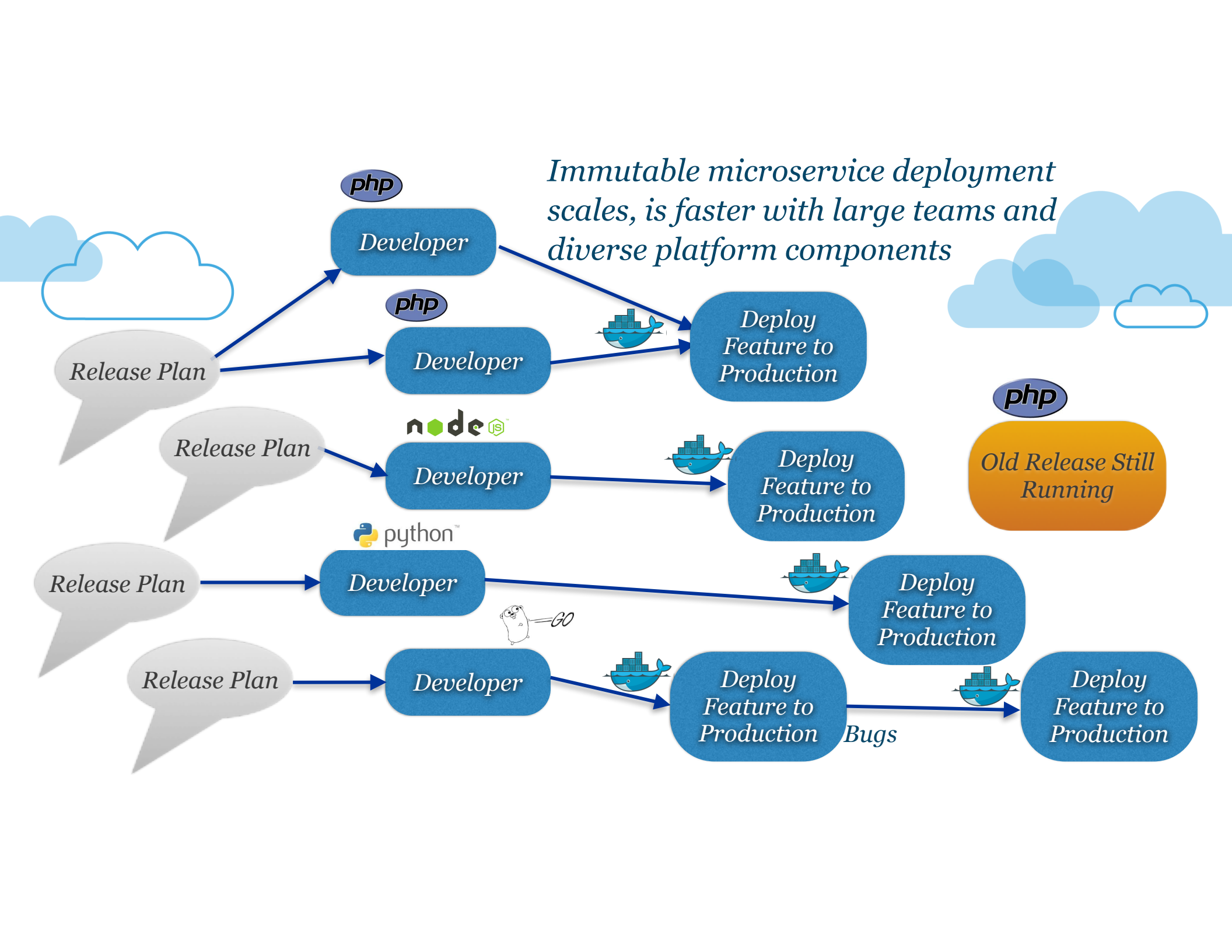
Re-Org from project teams to product teams

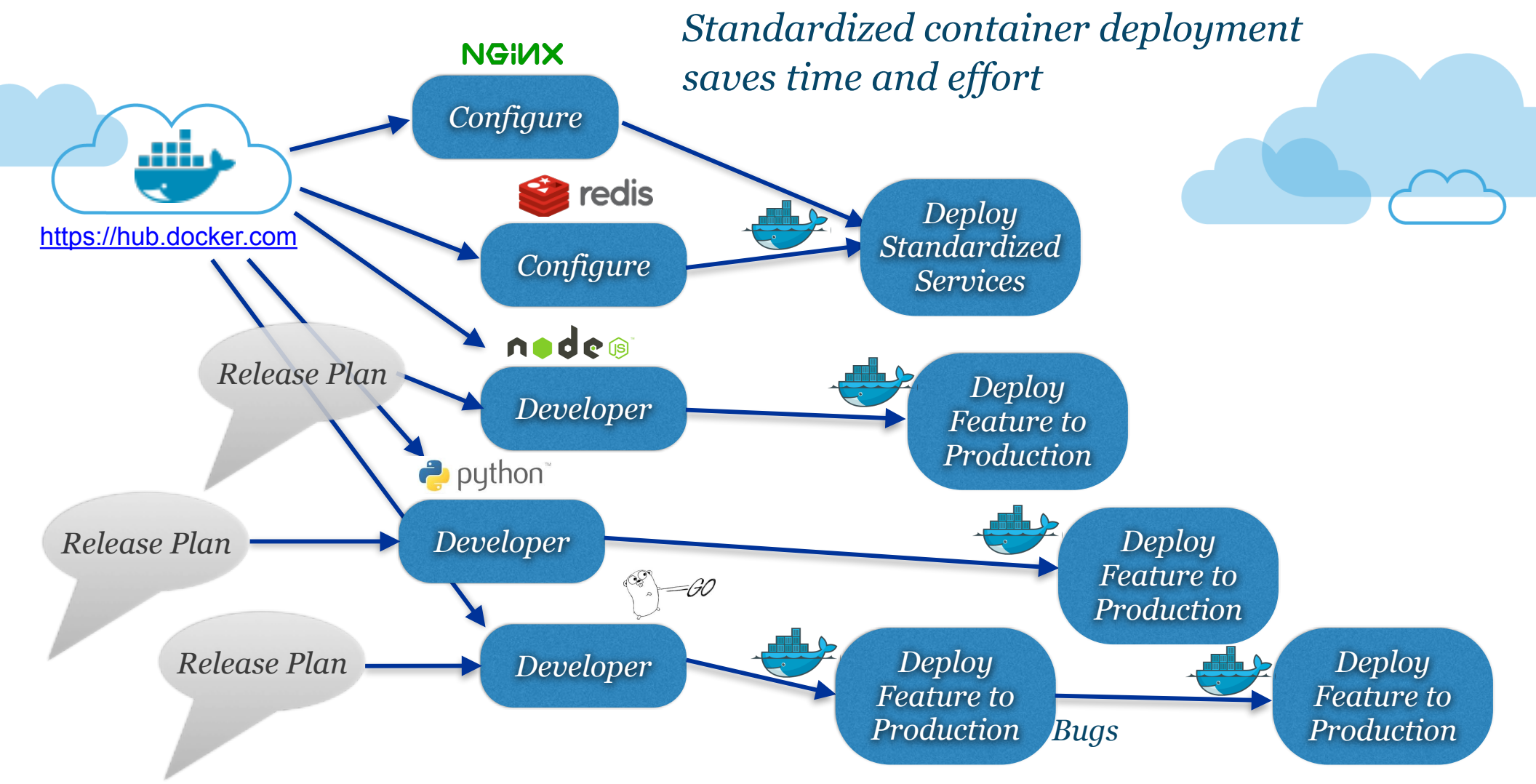

Monolithic service updates
php
Developer
php
Developer
php
Developer
php
Developer
php
Developer
Release Plan
QA Release Integration
Bugs
php
Ops Replace Old With New Release
Bugs
Works well with a small number of developers and a single language like php, java or ruby



Use monolithic apps for small teams, simple systems and when you must, to optimize for efficiency and latency

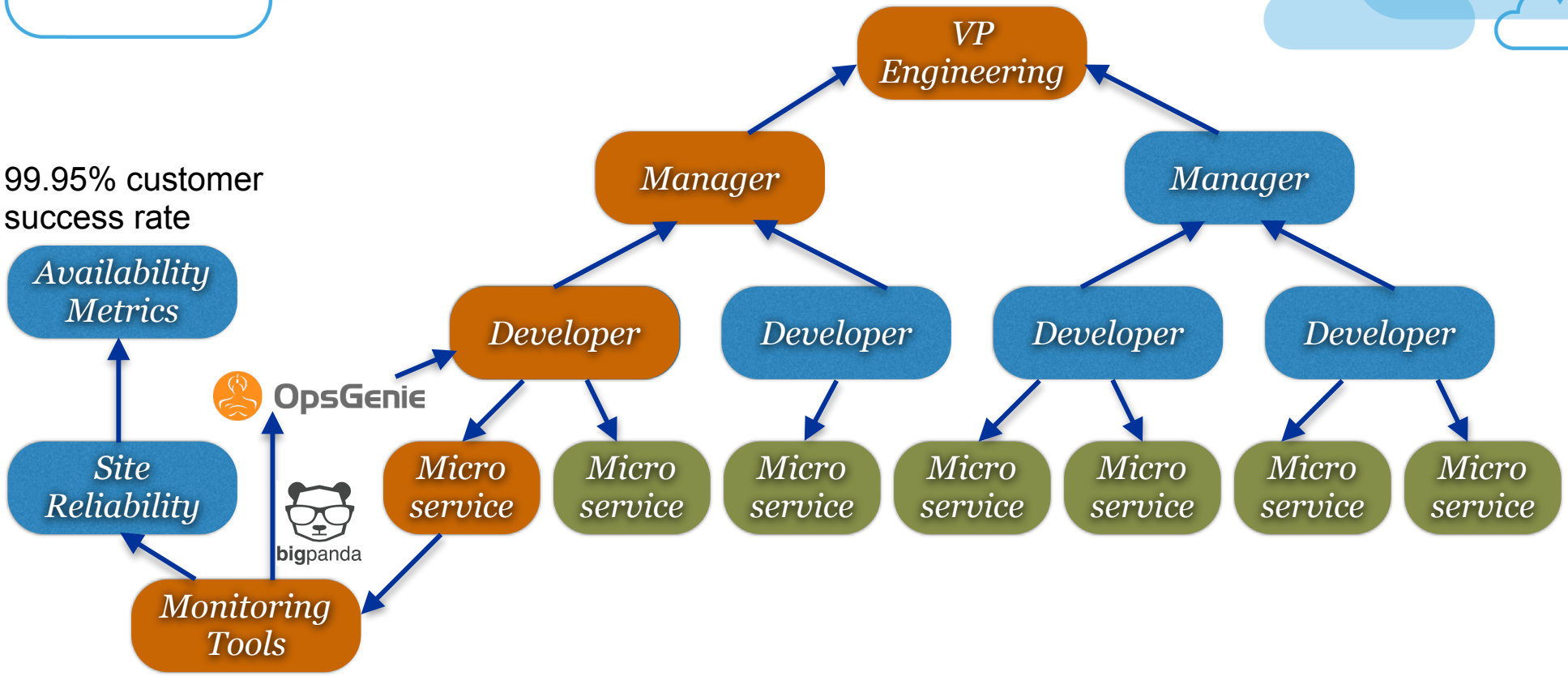



Immutable microservice deployment scales, is faster with large teams and diverse platform components
php
Developer
Release Plan
php
Developer
Deploy Feature to Production
Release Plan
node
Developer
Deploy Feature to Production
php
Old Release Still Running
Release Plan
python
Developer
Deploy Feature to Production
GO
Release Plan
Developer
Deploy Feature to Production
Bugs
Deploy Feature to Production


Standardized container deployment saves time and effort
NGINX
Configure
https://hub.docker.com
redis
Configure
Deploy Standardized Services
node
Release Plan
Developer
Deploy Feature to Production
python
Release Plan
Developer
Deploy Feature to Production
GO
Release Plan
Developer
Deploy Feature to Production
Bugs
Deploy Feature to Production

Run What You Wrote

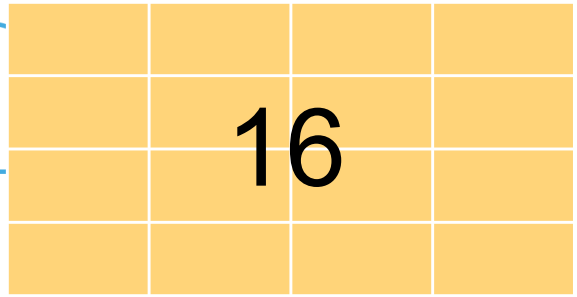
99.95% customer success rate



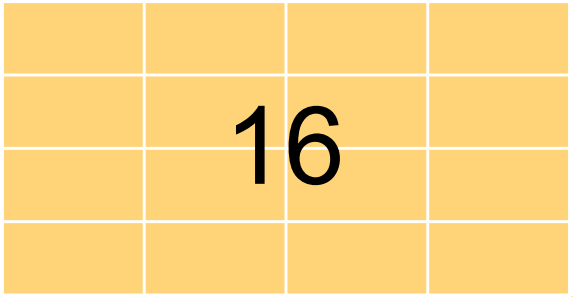
Non-Destructive Production Updates

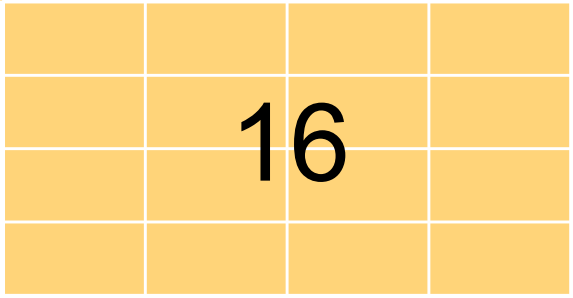
- *"Immutable Code" Service Pattern*
  - *Existing services are unchanged, old code remains in service*
  - *New code deploys as a new service group*
  - *No impact to production until traffic routing changes*
- *A|B Tests, Feature Flags and Version Routing control traffic*
  - *First users in the test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

Deliver four features every four weeks



Bugs! Which feature broke?
Need more time to test!
Extend release to six weeks?

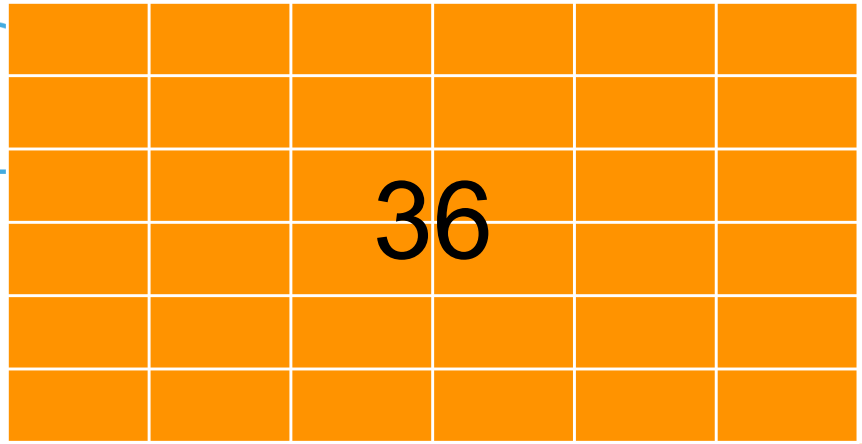




Work In Progress = 4
Opportunity for bugs: 100% (baseline)
Time to debug each: 100% (baseline)

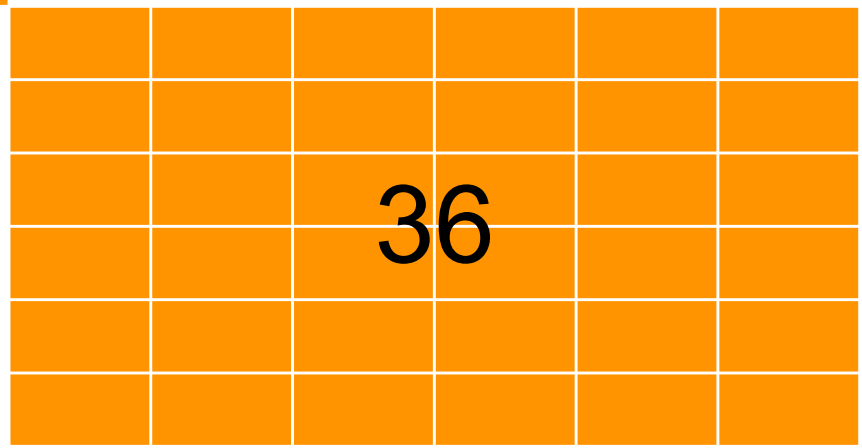
But: risk of bugs in delivery increases with interactions!

Deliver six features every six weeks



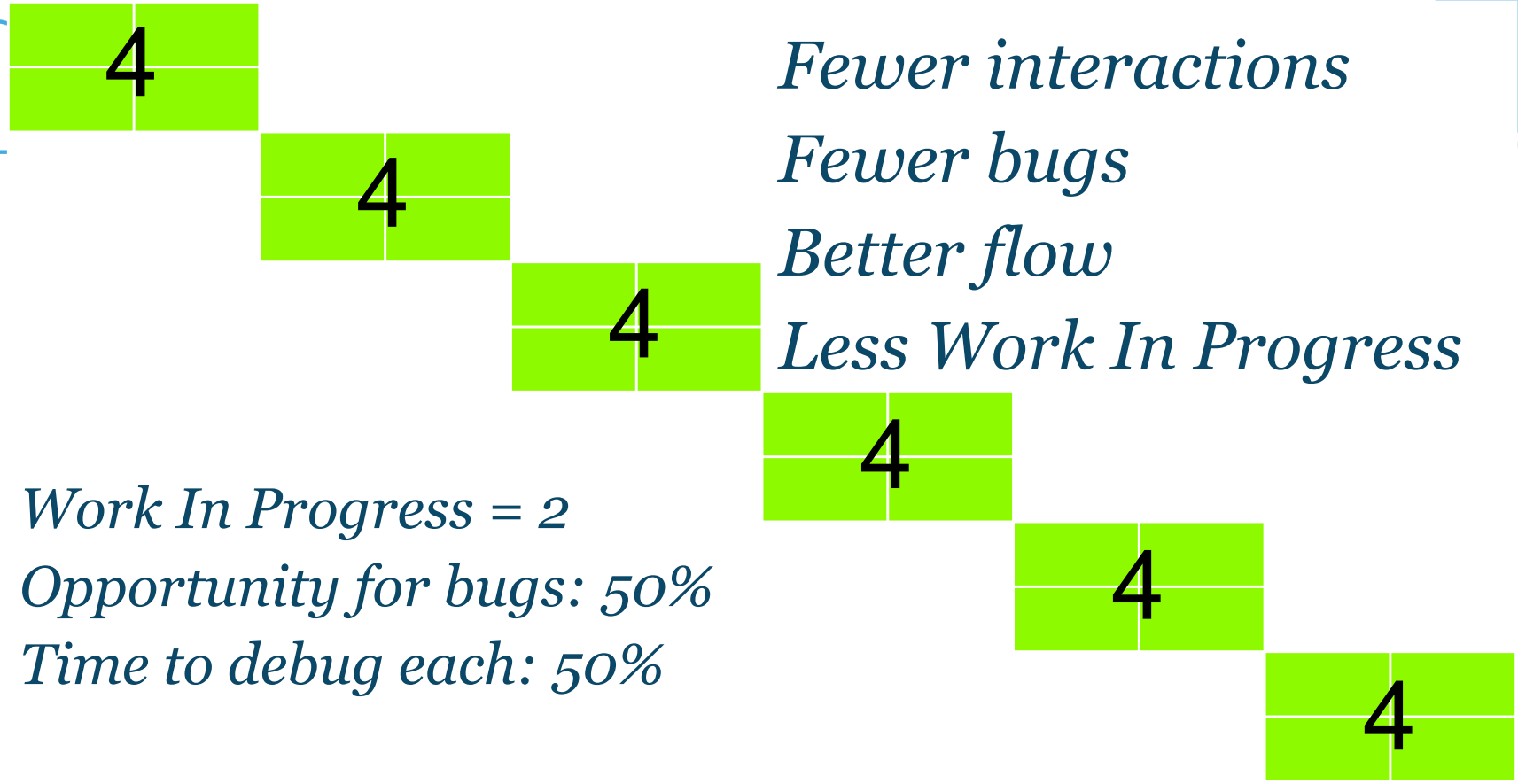
More features
What broke?
More interactions
Even more bugs!!

Work In Progress = 6
Individual bugs: 150%
Interactions: 150%?



Risk of bugs in delivery increased to 225% of original!

Deliver two features every two weeks



Fewer interactions

Fewer bugs

Better flow

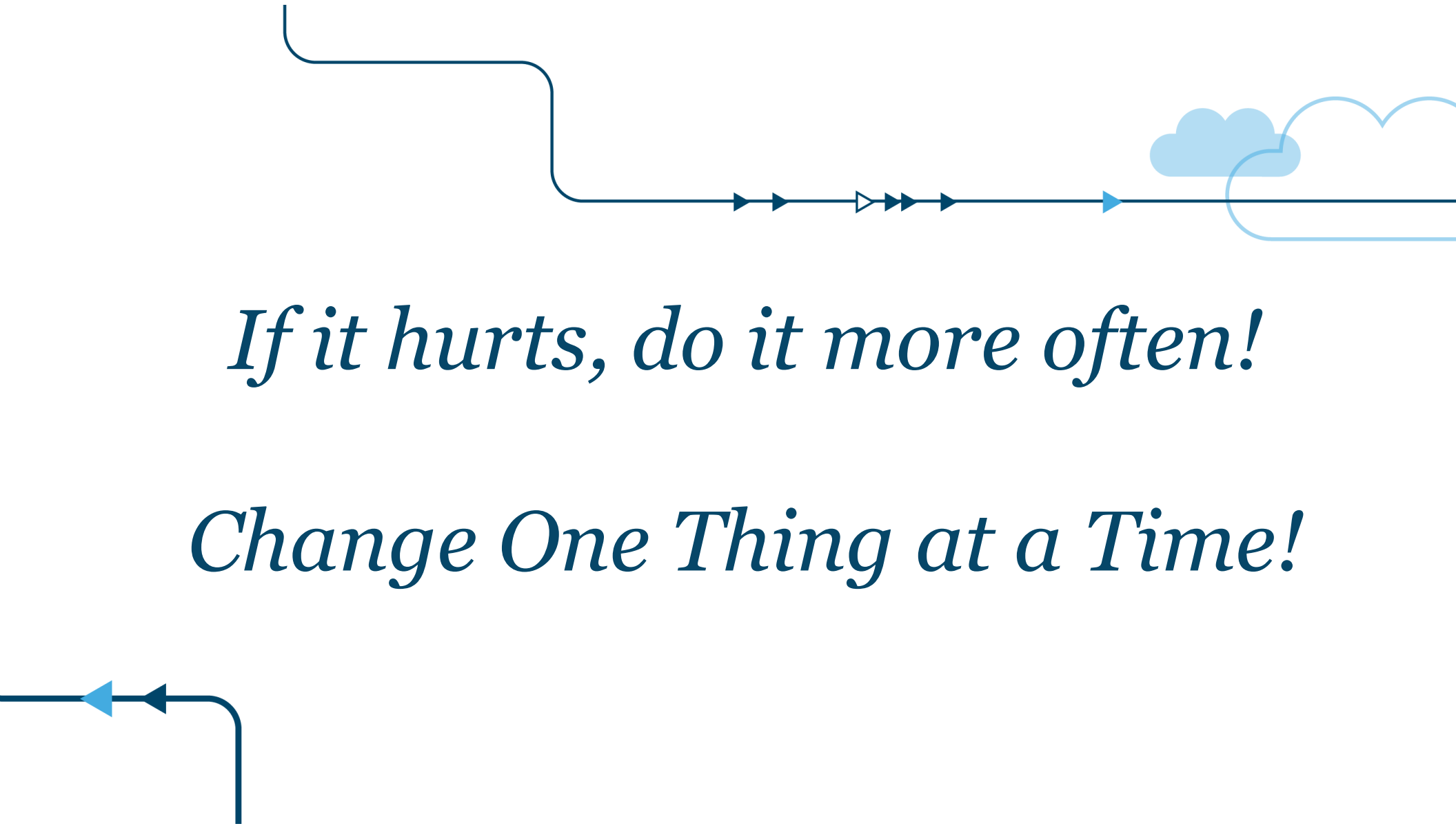
Less Work In Progress

Work In Progress = 2

Opportunity for bugs: 50%

Time to debug each: 50%

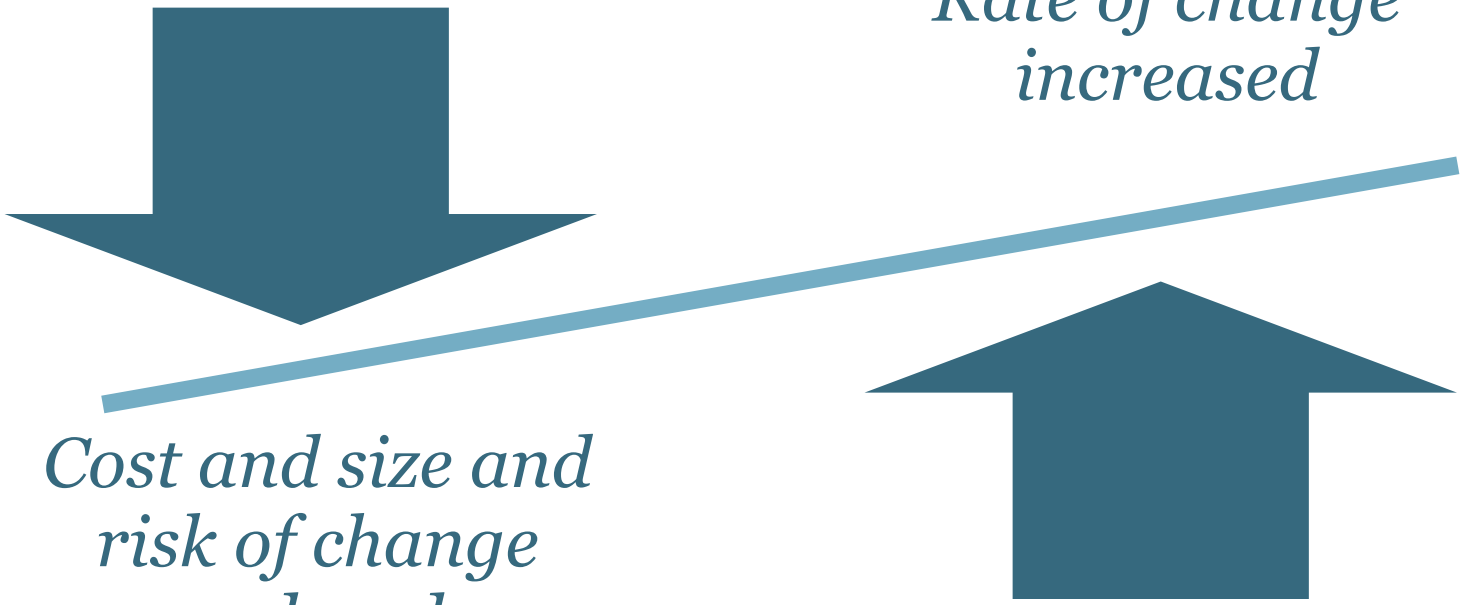
Complexity of delivery decreased by 75% from original



If it hurts, do it more often!

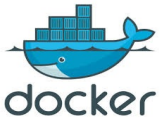
Change One Thing at a Time!

What Happened?

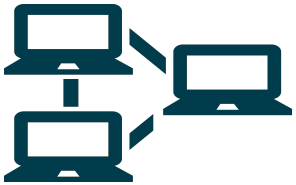


Low Cost of Change Using Docker

Developers
- Compile/Build
- Seconds

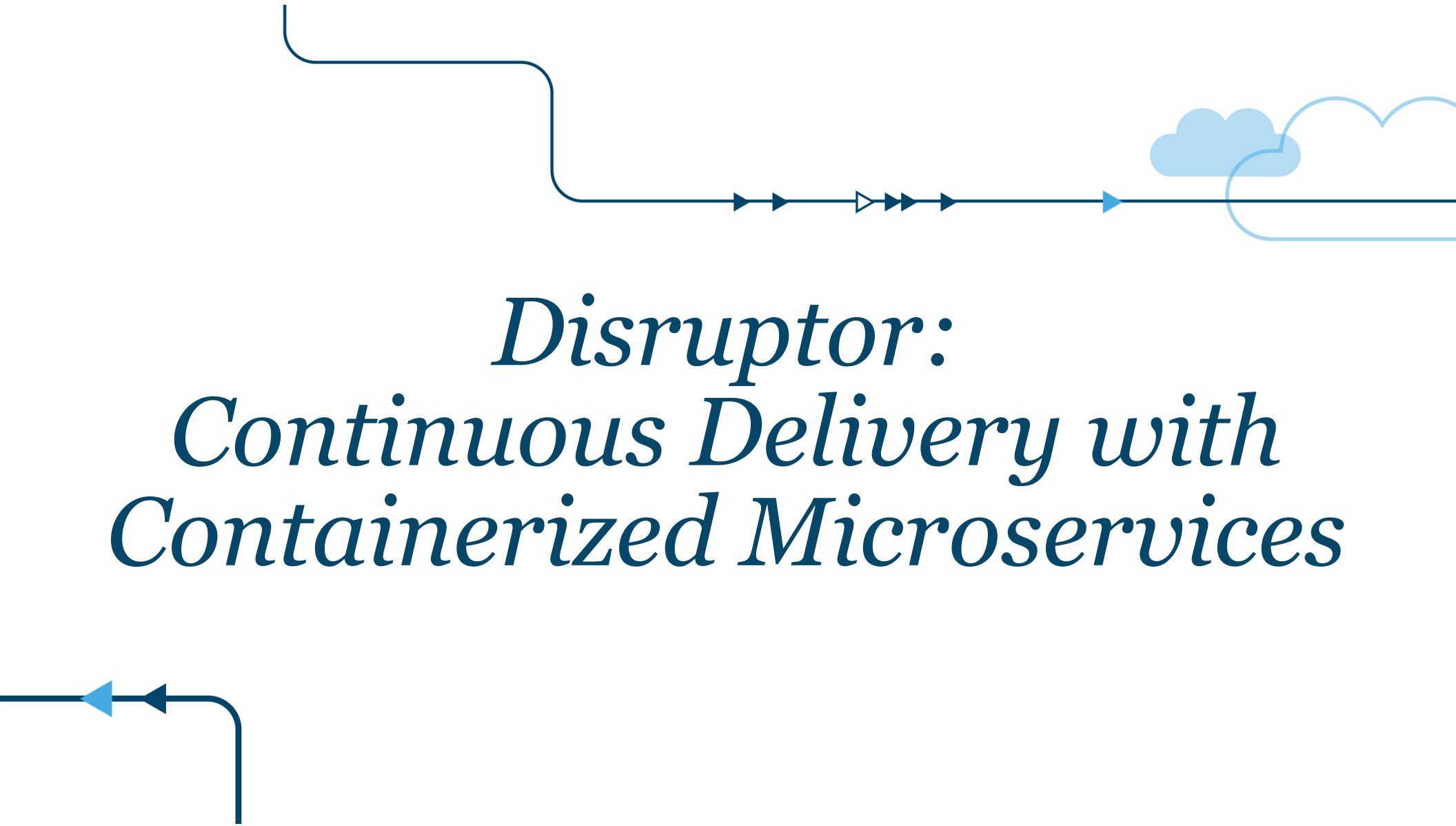


Extend container
- Package dependencies
- Seconds



Deploy Container
- Docker startup
- Seconds

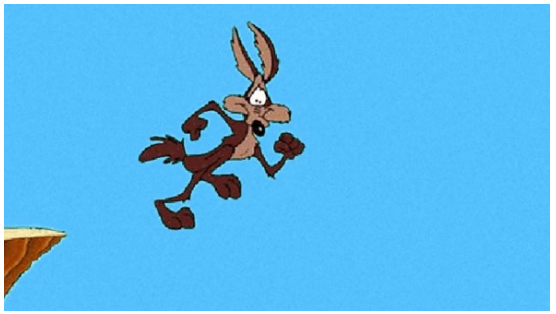
Fast tooling supports continuous delivery of many tiny changes

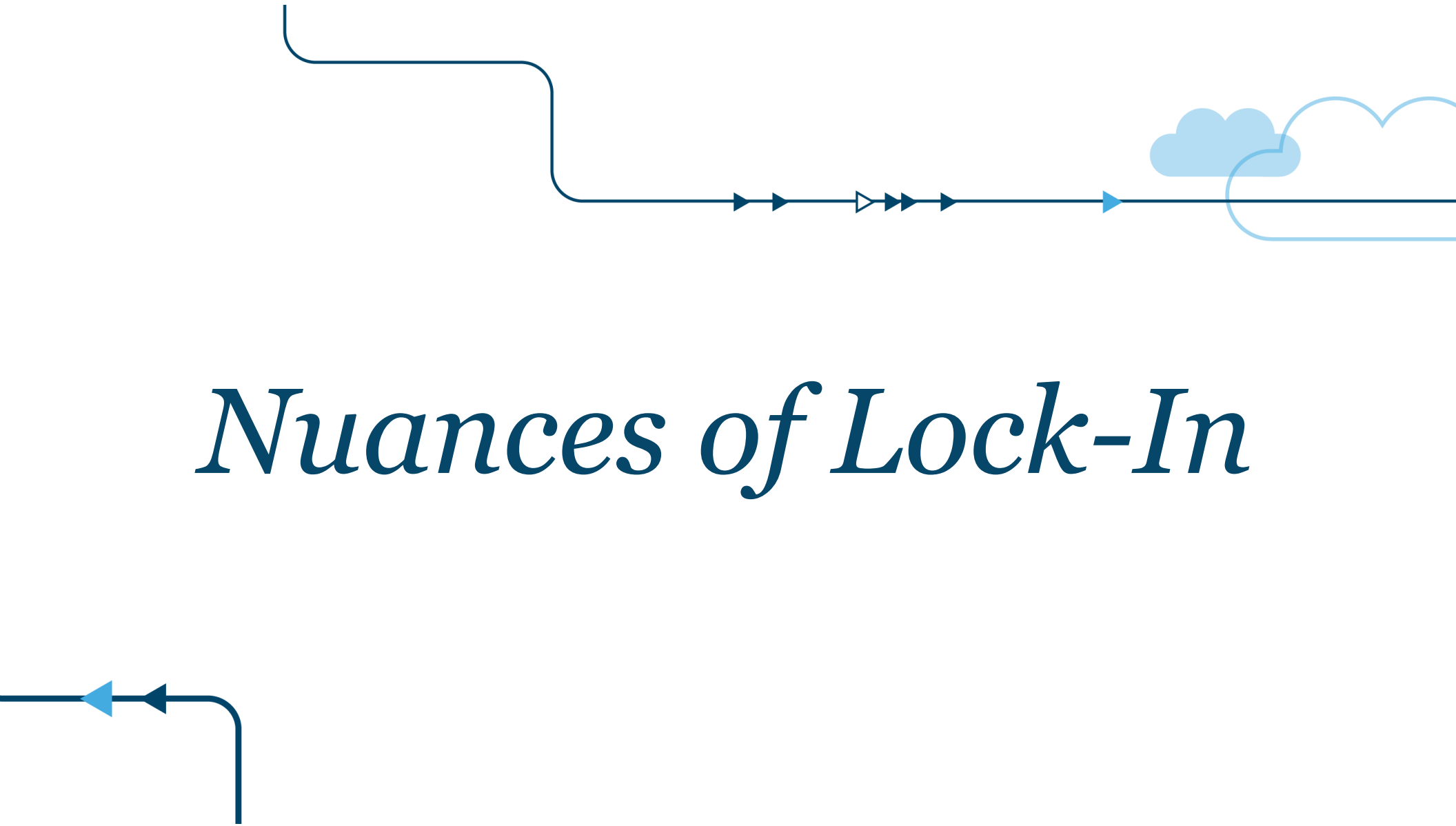


Disruptor: Continuous Delivery with Containerized Microservices

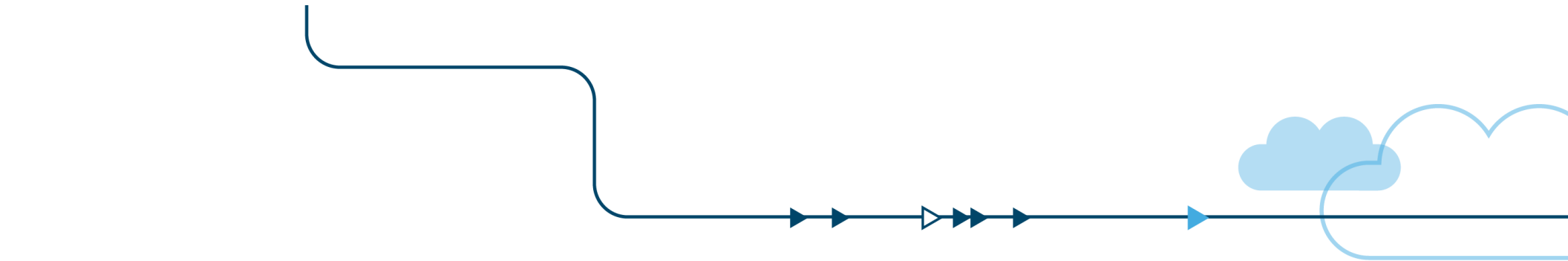
It’s what you know that isn’t so

- *Make your assumptions explicit*
- *Extrapolate trends to the limit*
- *Listen to non-customers*
- *Follow developer adoption, not IT spend*
- *Map evolution of products to services to utilities*
- *Re-organize your teams for speed of execution*

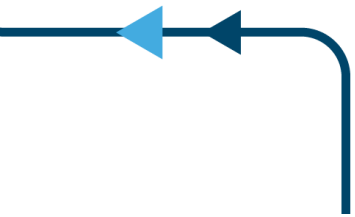


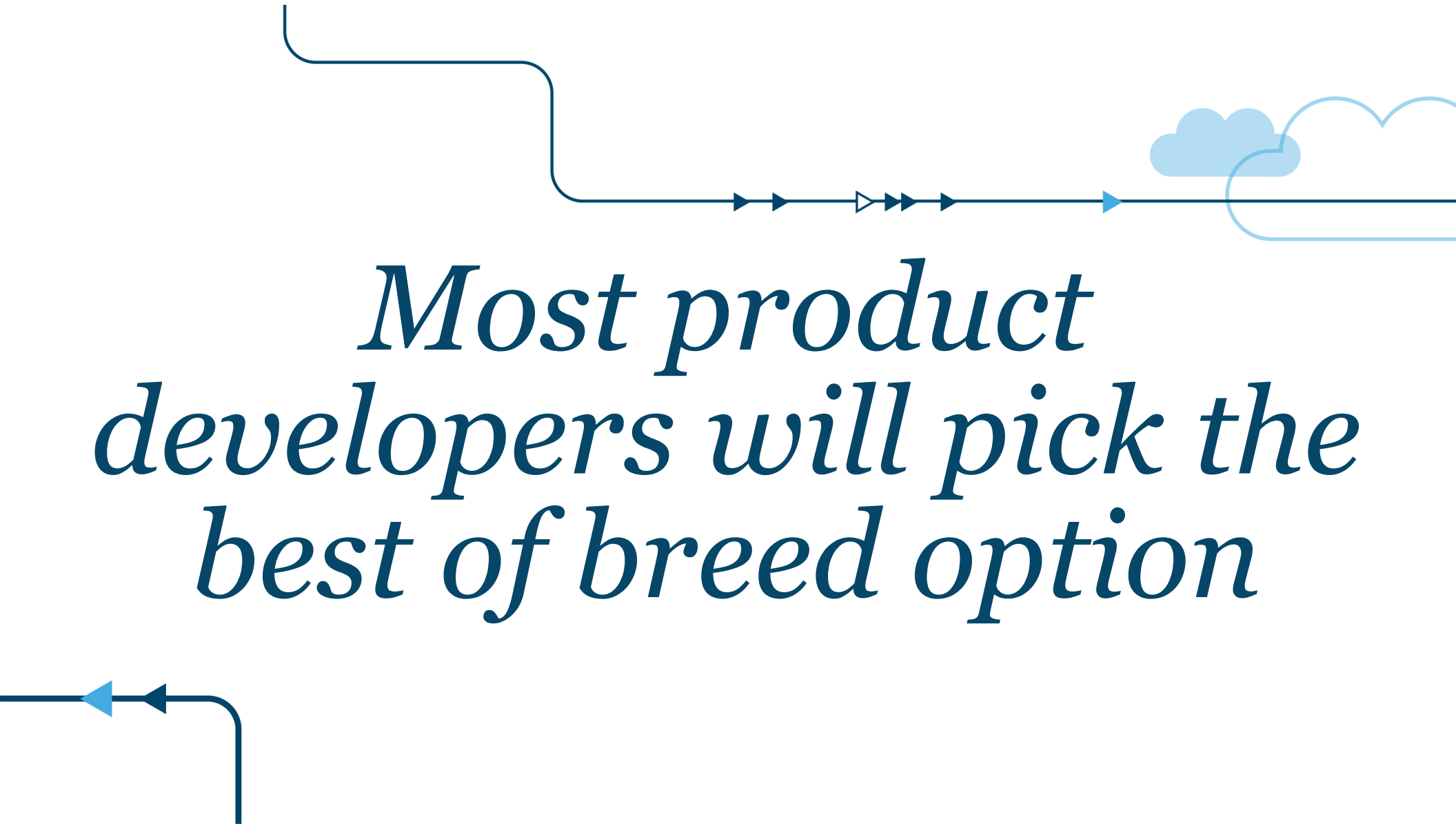


Nuances of Lock-In

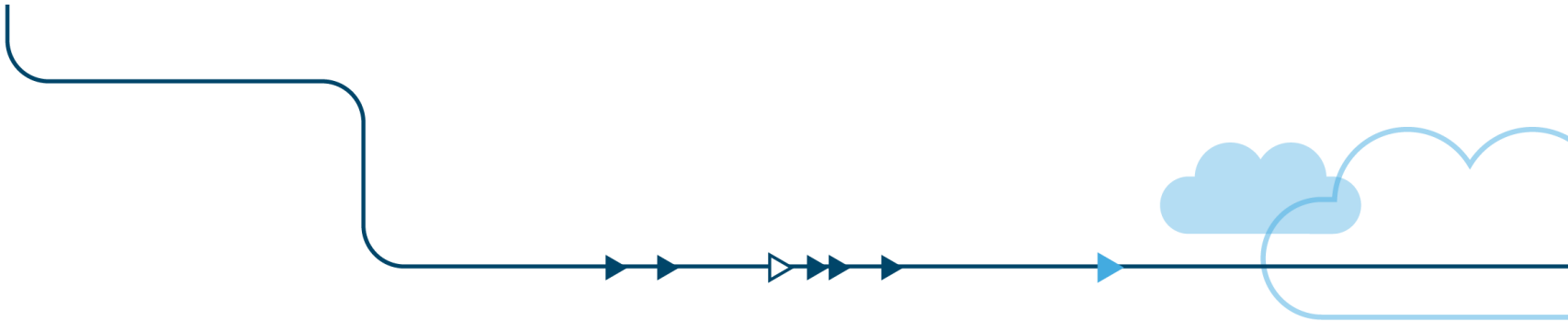


Most IT Ops people will try to avoid lock-in



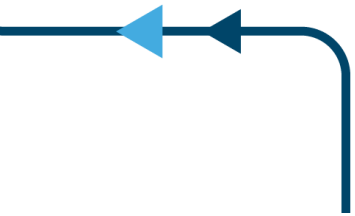


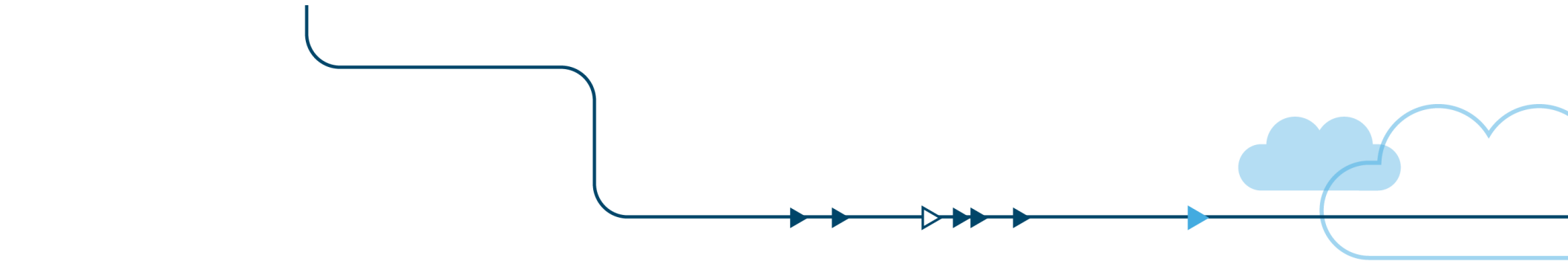
Most product developers will pick the best of breed option



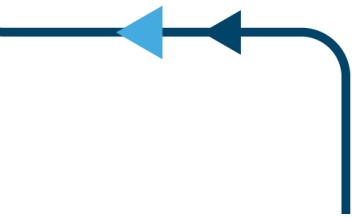
Analogy:
Dev's get to go on dates and get married
Ops get to run the family and get divorced

DevOps gets the whole lifecycle!





DevOps to the rescue!



https://www.youtube.com/watch?v=7g3uqSzWVZs

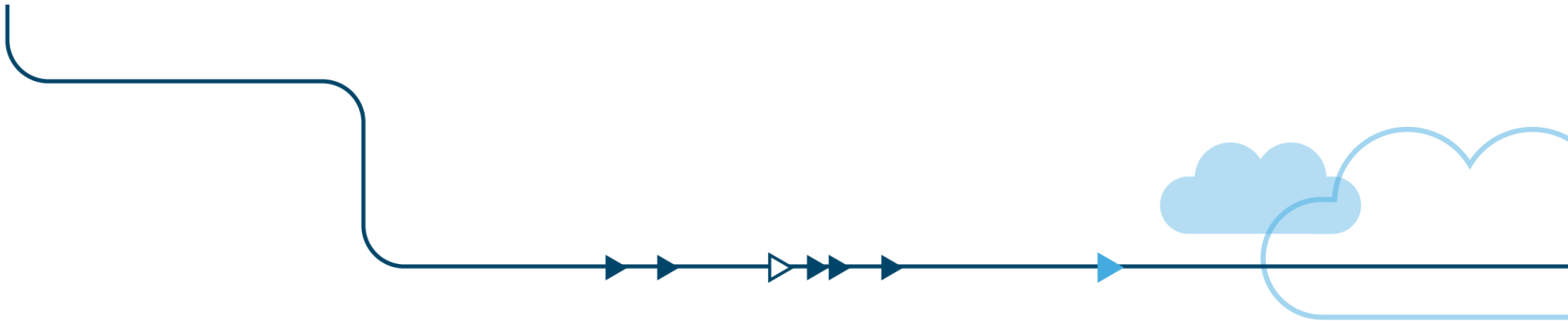
Erik Rogneby @erikrogneby · Mar 9

@botchagalupe John, do you mind if we thrown your ignite **Deming** 14 point video up on our DevOps page? Brilliant! youtube.com/watch?v=7g3uqS…

YouTube

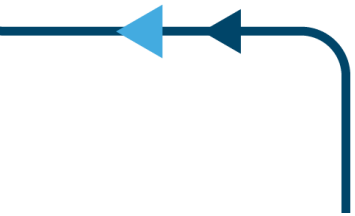
John Willis' Ignite talk: "Devops and Dr Deming's 14 Points" (Veloc...

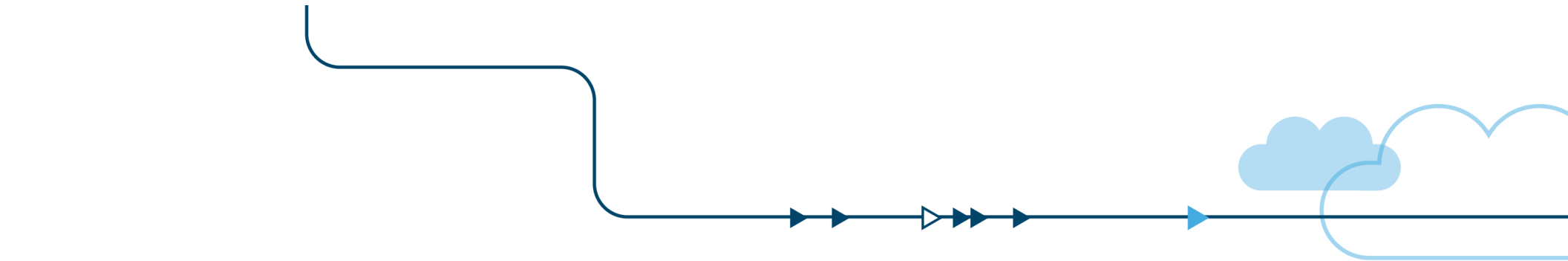
This a rapid paced 20 slide Ignite presentation about Edward Deming and his 14 points. More specifically we will analyze each one of his 14 points as to how ...



"End the practice of awarding business on the basis of a price tag. Instead, minimize total cost. Move toward a single supplier for any one item, on a long-term relationship of loyalty and trust."

W. Edwards Deming - 4th Point

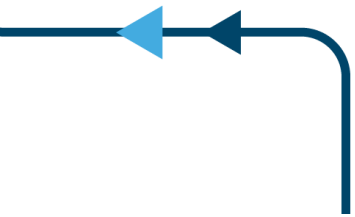


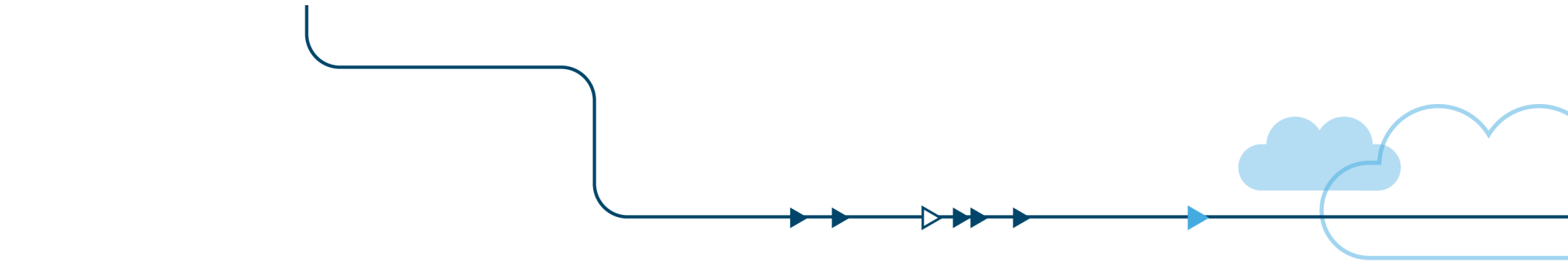


"End the practice of awarding business on the basis of a price tag. Instead, minimize total cost. Move toward a single supplier for any one item, on a ~~long-term~~ relationship of ~~loyalty and trust.~~"

dysfunctional *exploitation and abuse*

How did we end up here?

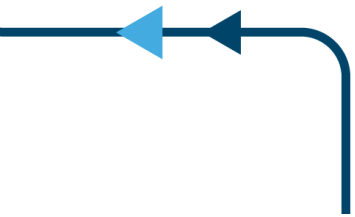


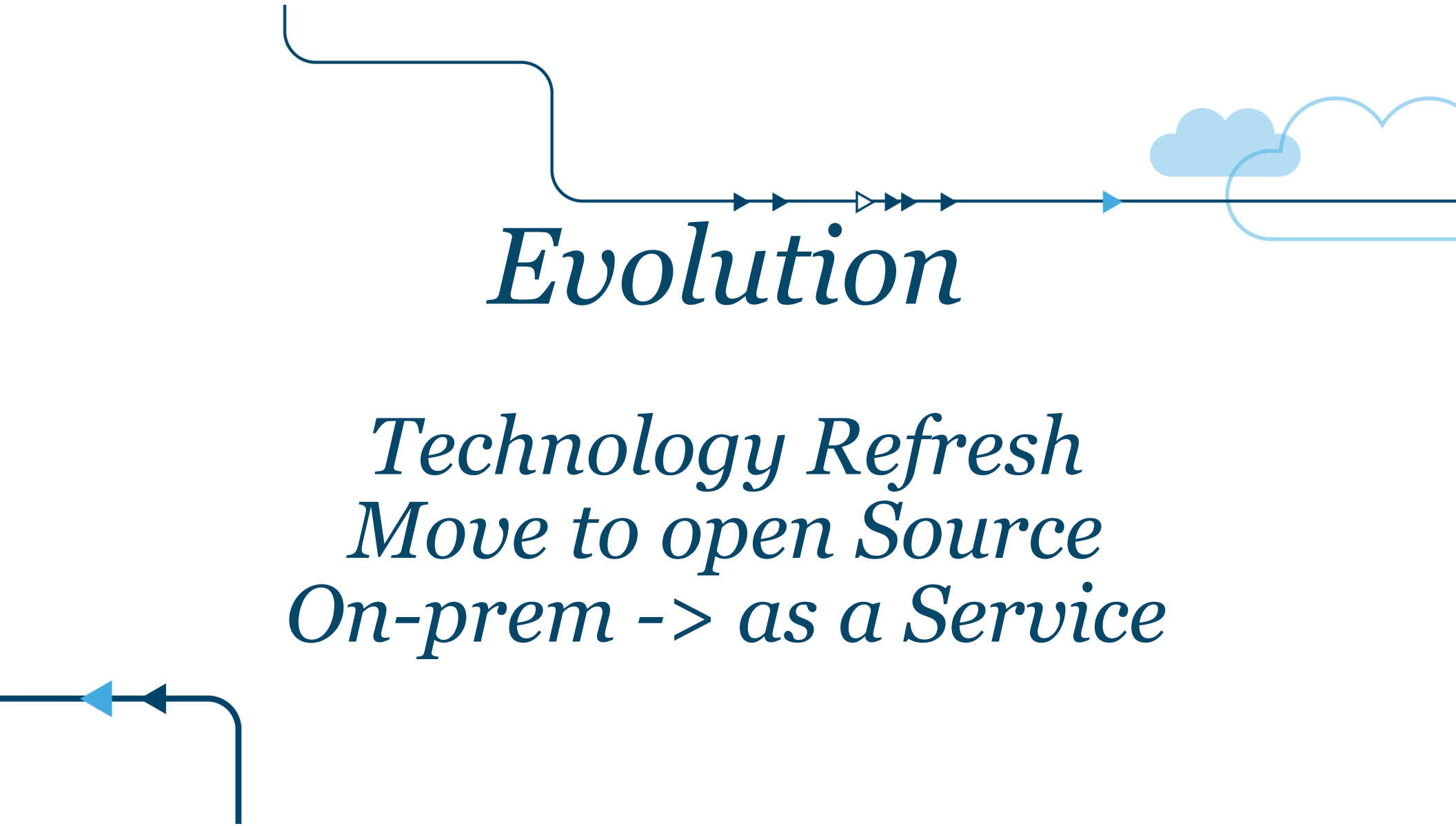


Project vs. Product

Leads to lock-in

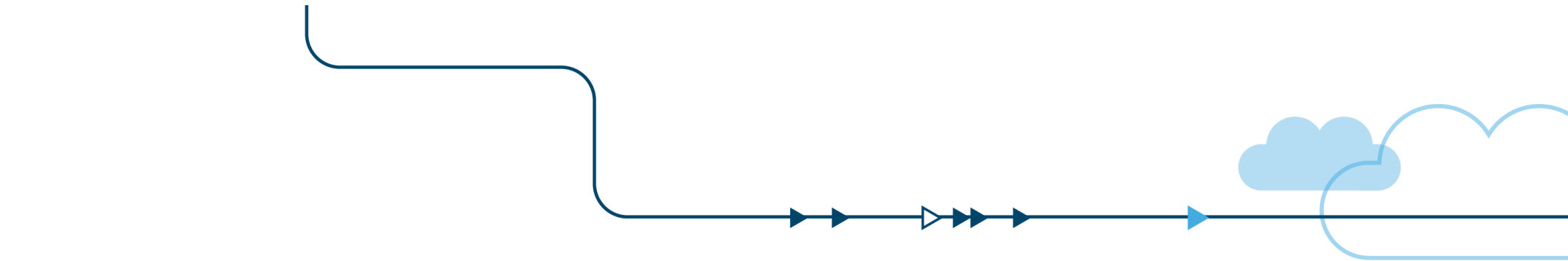
Evolves to follow best of breed





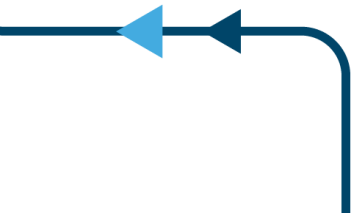
Evolution

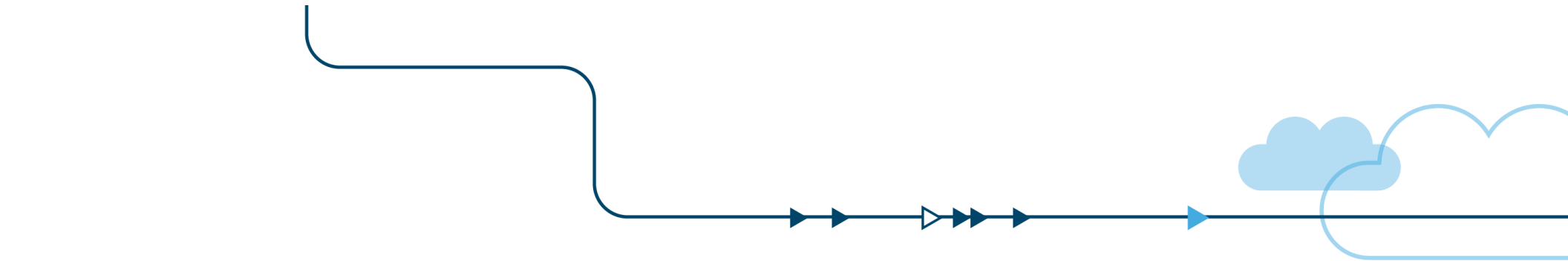
Technology Refresh
Move to open Source
On-prem -> as a Service



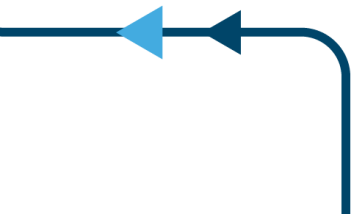
Best of breed is now OSS and as a Service

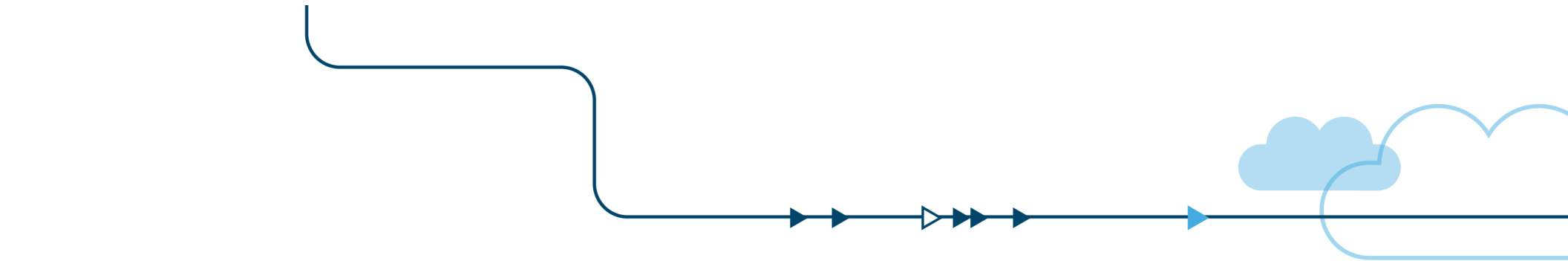
Less inherent lock-in





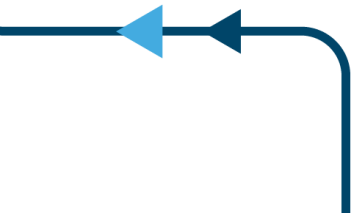
What kinds of lock-in are there?

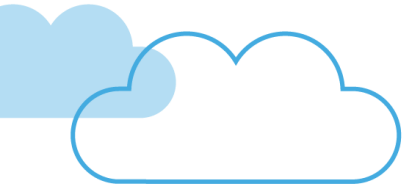




Business lock-in

Hardest to escape…





Legal lock-in

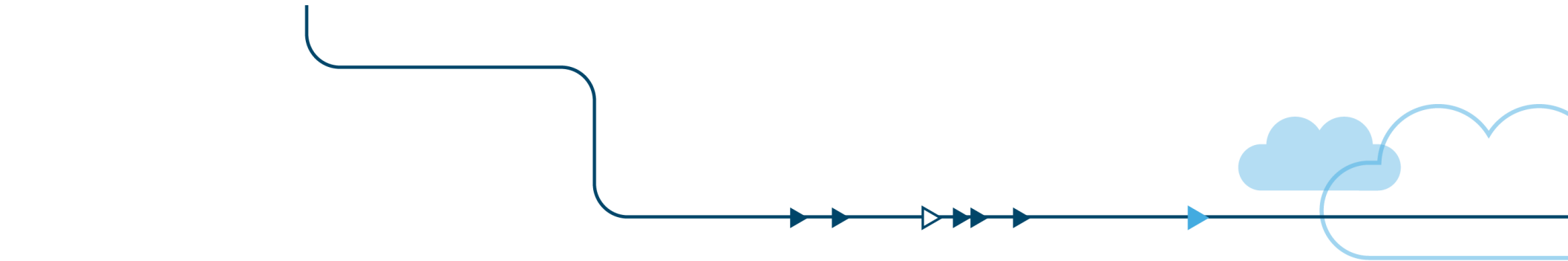
e.g. compliance with laws that exclude alternatives based on jurisdiction or certification

Financial lock-in

e.g. budget spent in advance on long term deal with a vendor

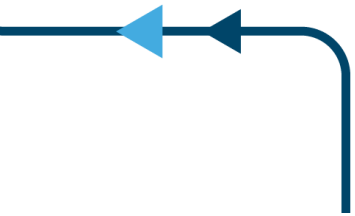
Contractual lock-in

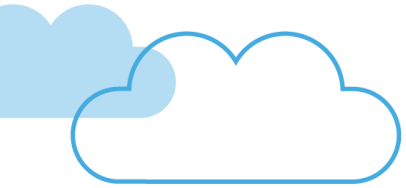
e.g. partnership or investment deal with one vendor prevents using alternatives



Technology lock-in

Possible to escape given time and work...





Implementation

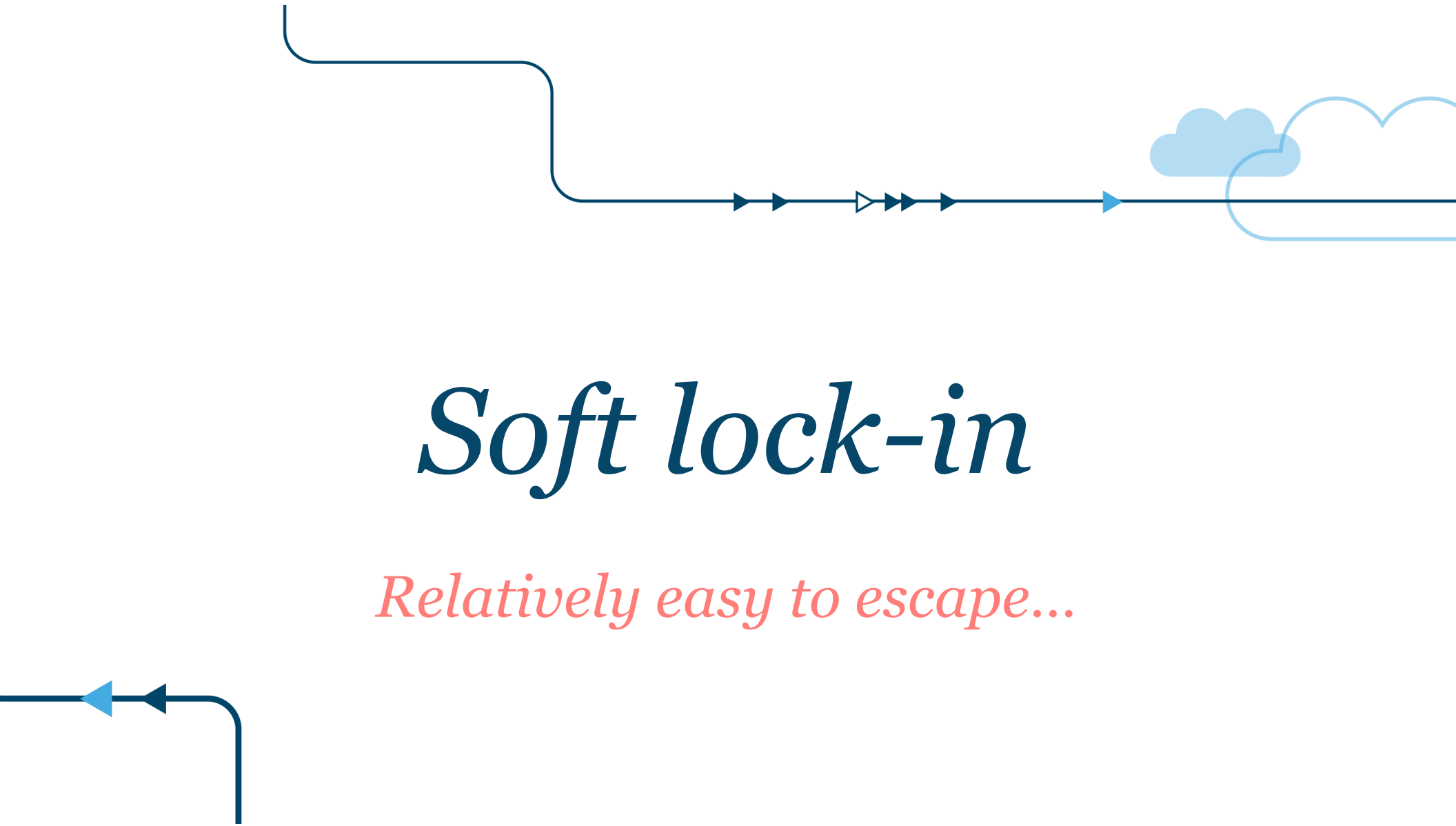
e.g. interface is the same but behavior is different

Proximity lock-in

e.g. chatty clients don't work unless they are co-located with their server

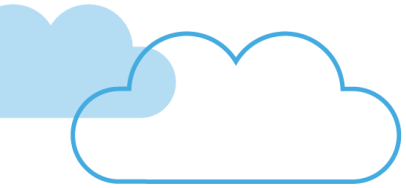
Topology lock-in

e.g. quorum based availability (C, Riak) needs three zones/datacenters per region*



Soft lock-in

Relatively easy to escape...



Interface lock-in

e.g. different APIs that get the same result, easy to hide behind an abstraction layer

Web service lock-in

Interface lock-in, but remote access unlocks ability to migrate applications

Query syntax lock-in

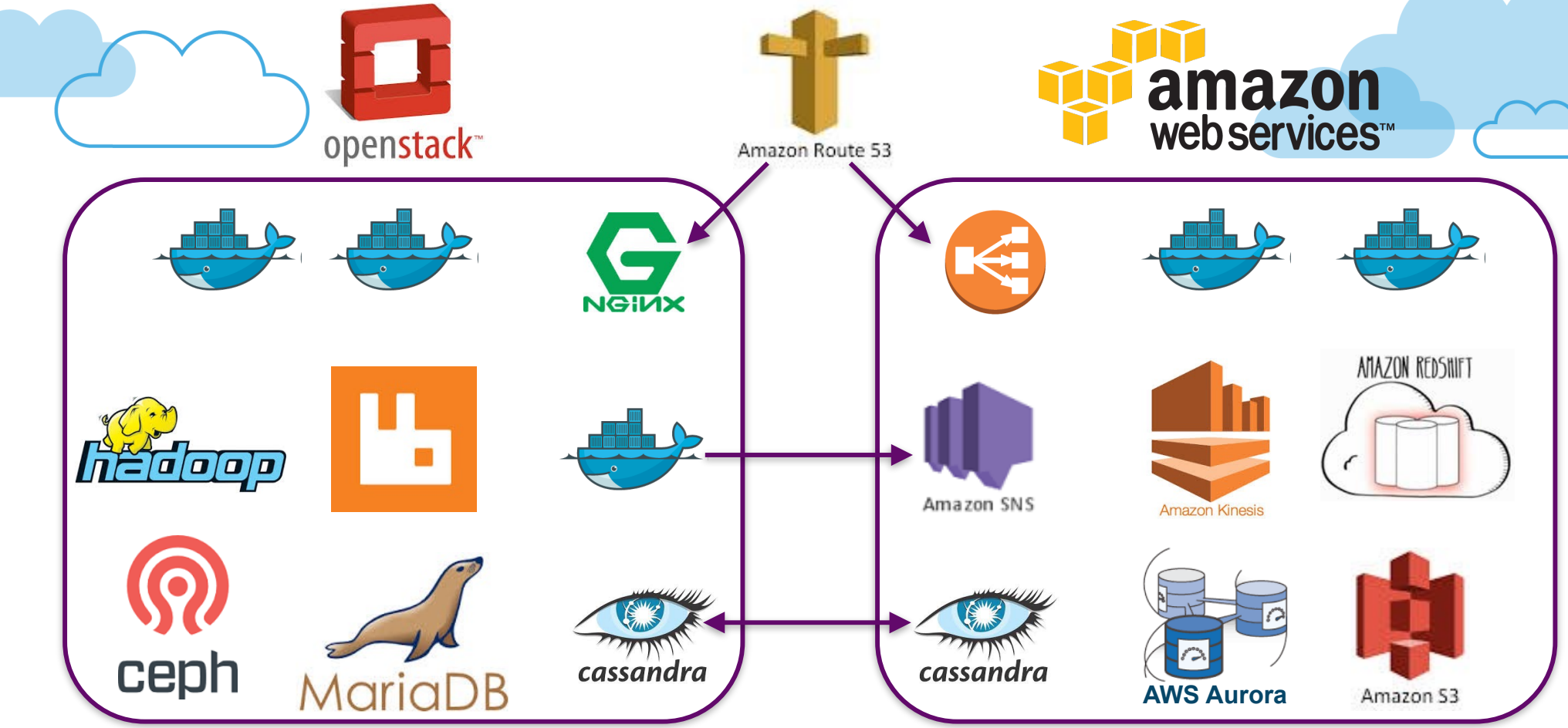
e.g. SQL variants for different databases

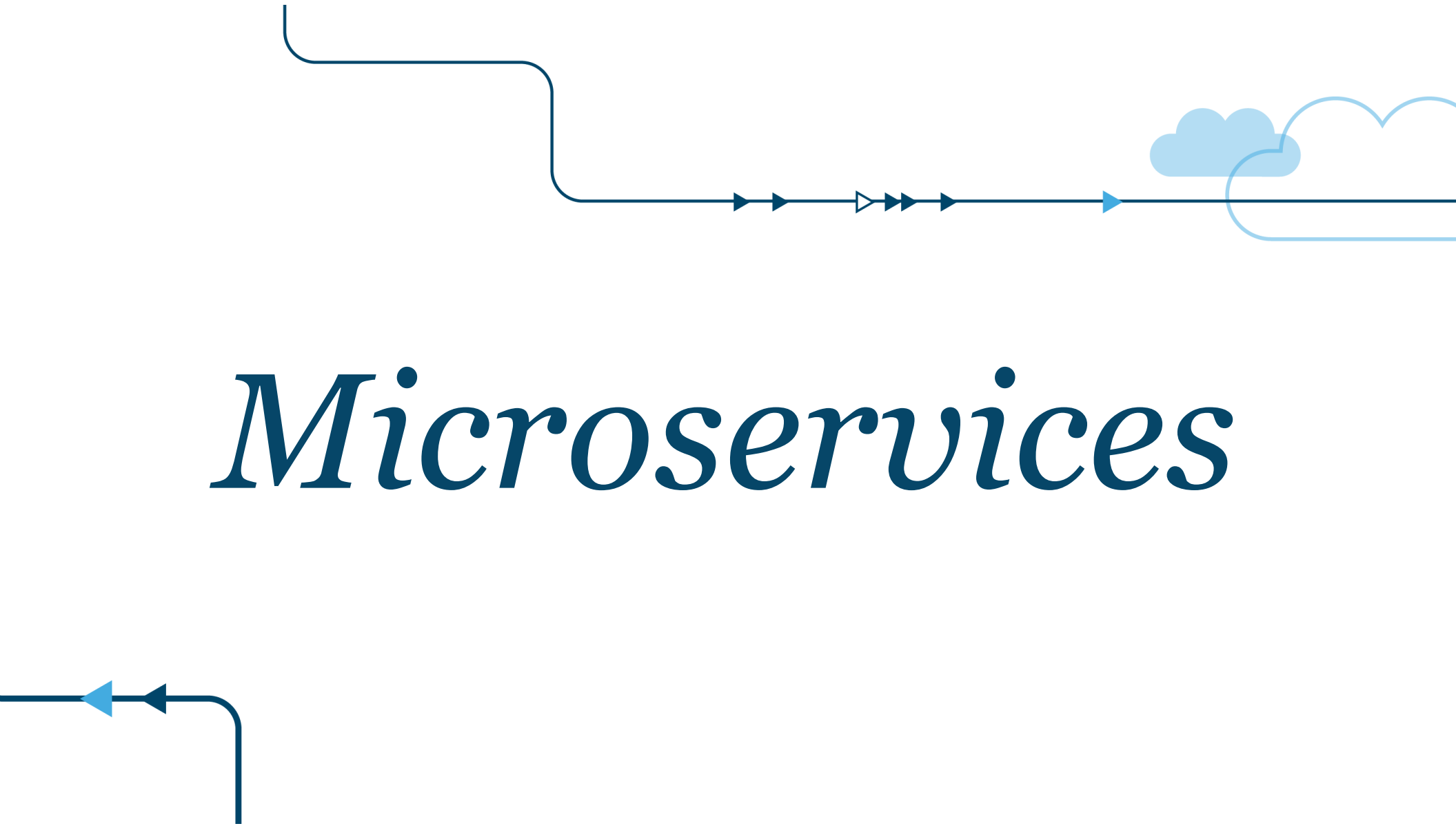
Data gravity lock-in

e.g. lots of data to move or duplicate

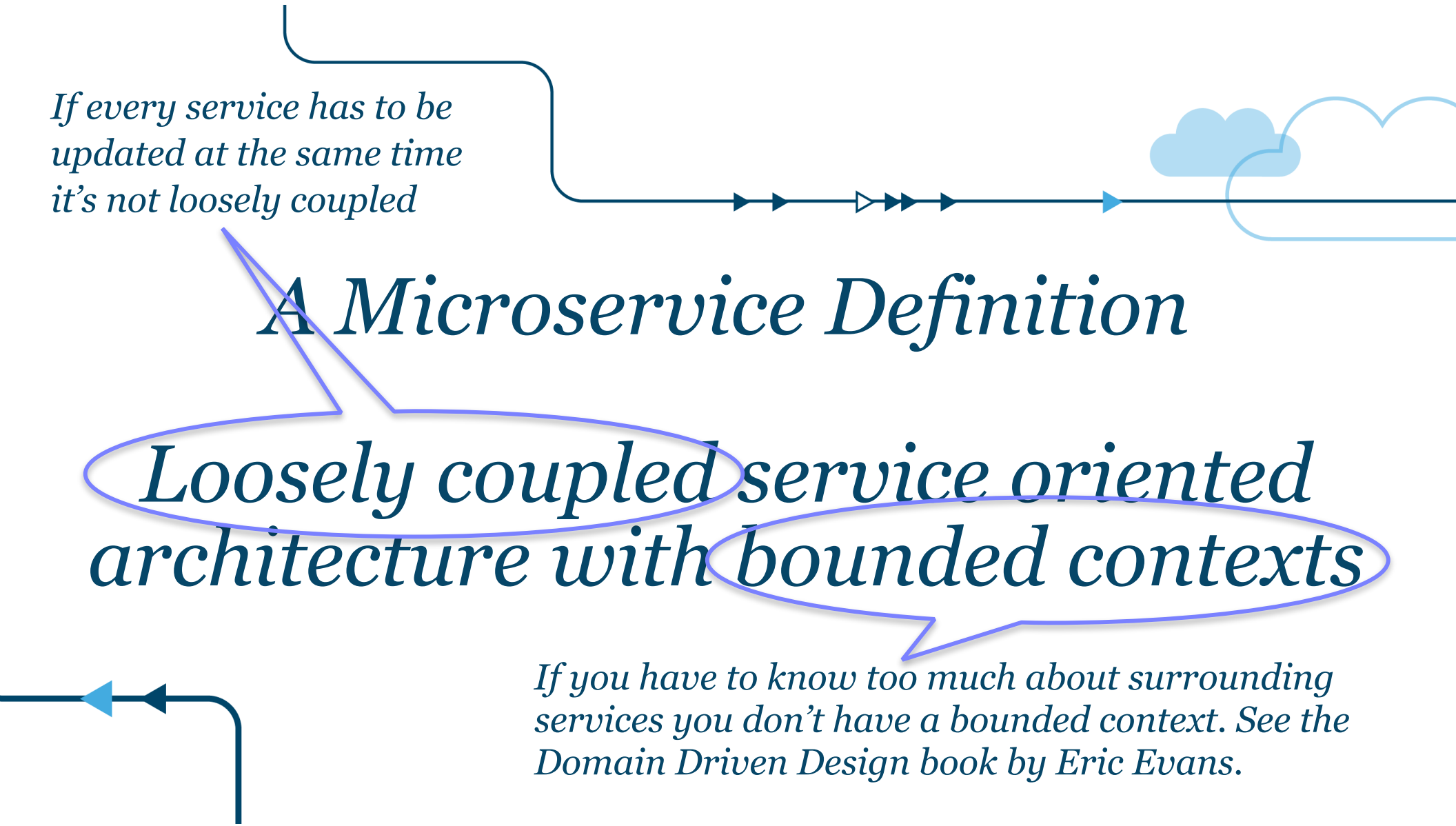
Cloud native microservices

Example for discussion



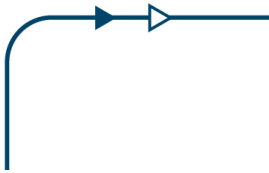


Microservices


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.

Coupling Concerns

- *Conway's Law - organizational coupling*
- *Centralized Database Schemas*
- *Enterprise Service Bus - centralized message queues*
- *Inflexible Protocol Versioning*

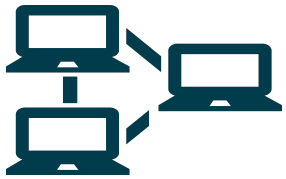


http://en.wikipedia.org/wiki/Conway's_law

Speeding Up The Platform

Datacenter Snowflakes
- Deploy in months
- Live for years

Virtualized and Cloud
- Deploy in minutes
- Live for weeks



Container Deployments
- Deploy in seconds
- Live for minutes/hours

Lambda Deployments
- Deploy in milliseconds
- Live for seconds

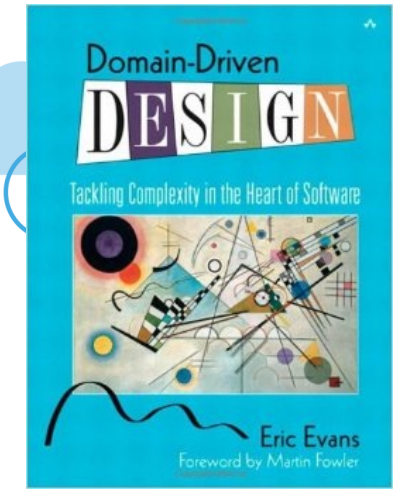
AWS Lambda is leading exploration of serverless architectures in 2016

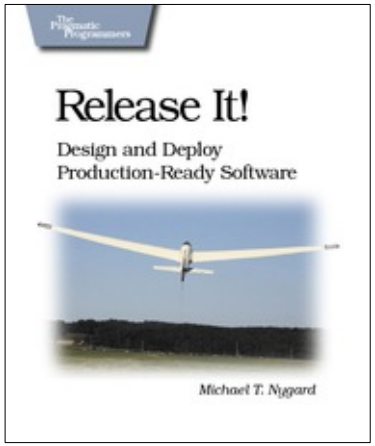
Separate Concerns with Microservices

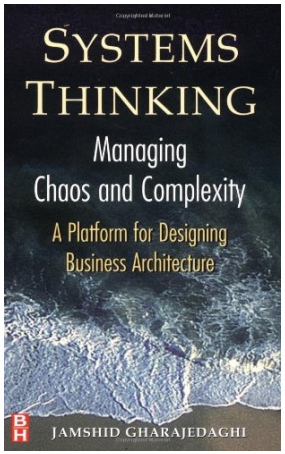
- *Invert Conway's Law – teams own service groups and backend stores*
- *One "verb" per single function micro-service, size doesn't matter*
- *One developer independently produces a micro-service*
- *Each micro-service is it's own build, avoids trunk conflicts*
- *Deploy in a container: Tomcat, AMI or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

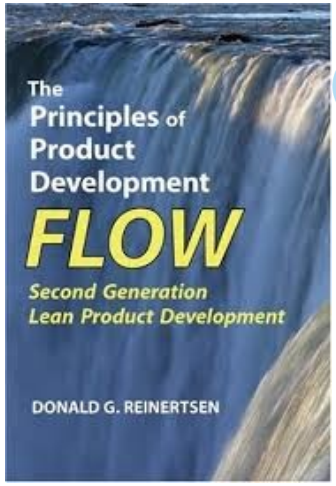
http://en.wikipedia.org/wiki/Conway's_law

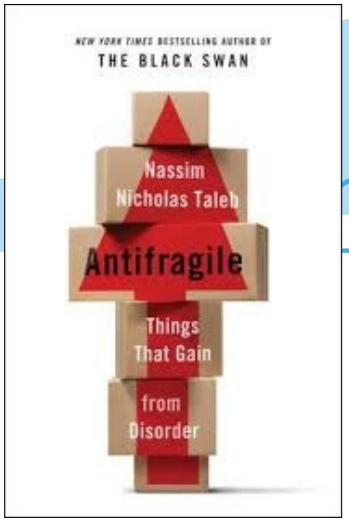
Inspiration

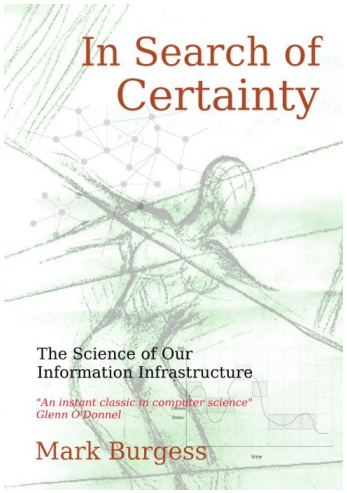


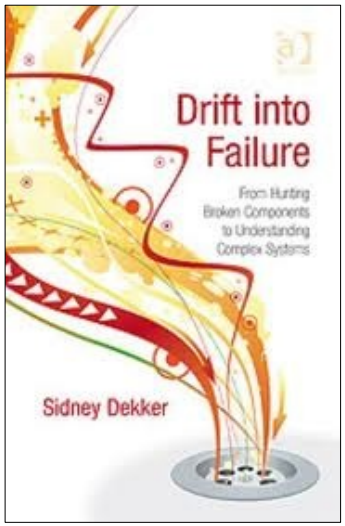


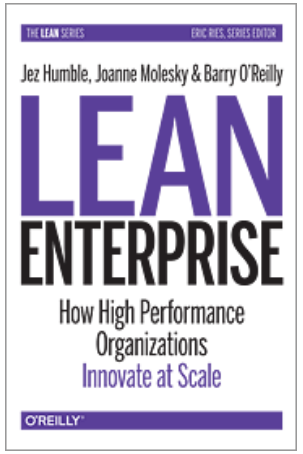


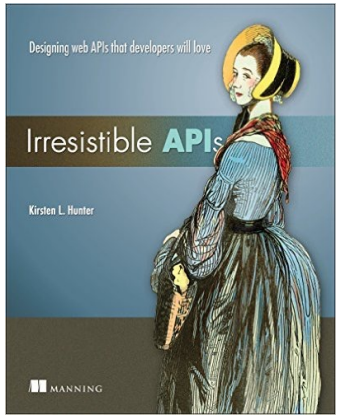


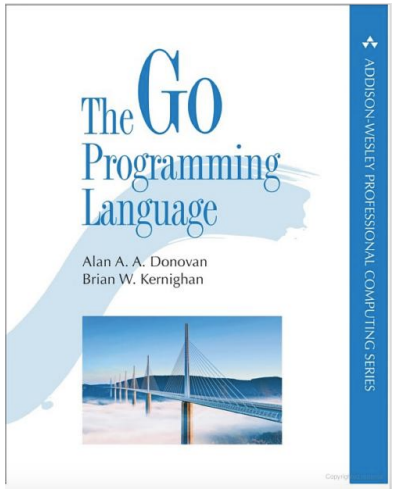












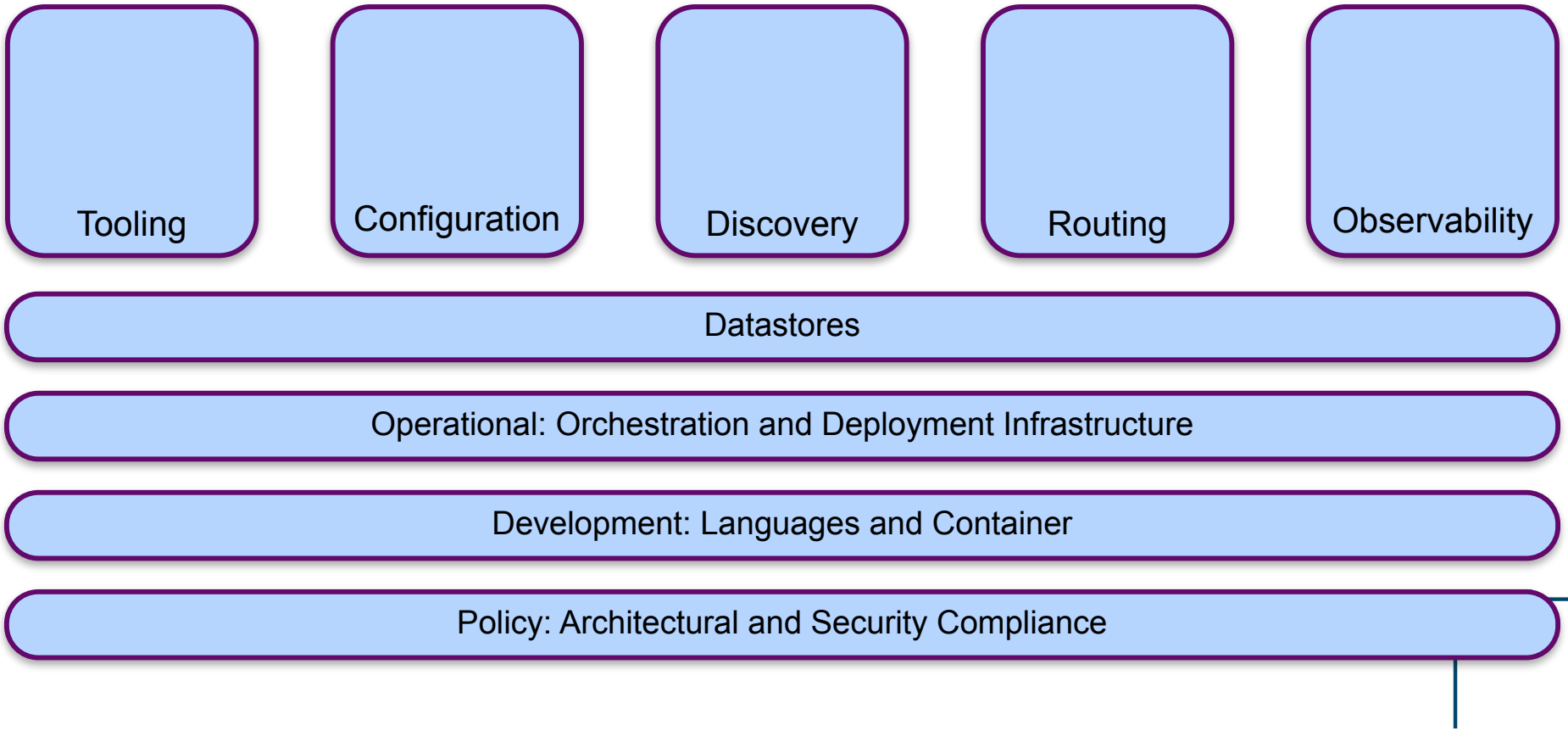
State of the Art in Web Scale Microservice Architectures


AWS Re:Invent : Asgard to Zuul https://www.youtube.com/watch?v=p7ysHhs5hI0
Resiliency at Massive Scale https://www.youtube.com/watch?v=ZfYJHtVL1_w
Microservice Architecture https://www.youtube.com/watch?v=CriDUYtfrjs
New projects for 2015 and Docker Packaging https://www.youtube.com/watch?v=hi7BDAtjfKY
Spinnaker deployment pipeline https://www.youtube.com/watch?v=dwdVwE52KkU
http://www.infoq.com/presentations/spring-cloud-2015


http://www.infoq.com/presentations/Twitter-Timeline-Scalability
http://www.infoq.com/presentations/twitter-soa
http://www.infoq.com/presentations/Zipkin
http://www.infoq.com/presentations/scale-gilt

Go-Kit https://www.youtube.com/watch?v=aL6sd4d4hxk
http://www.infoq.com/presentations/circuit-breaking-distributed-systems
https://speakerdeck.com/mattheath/scaling-micro-services-in-go-highload-plus-plus-2014

Microservice Architectures



NETFLIX OSS Microservices

Spinnaker
SpringCloud

Tooling

Edda
Archaius

Configuration

Eureka
Prana

Discovery

Denominator
Zuul
Ribbon

Routing

Hystrix
Pytheus
Atlas

Observability

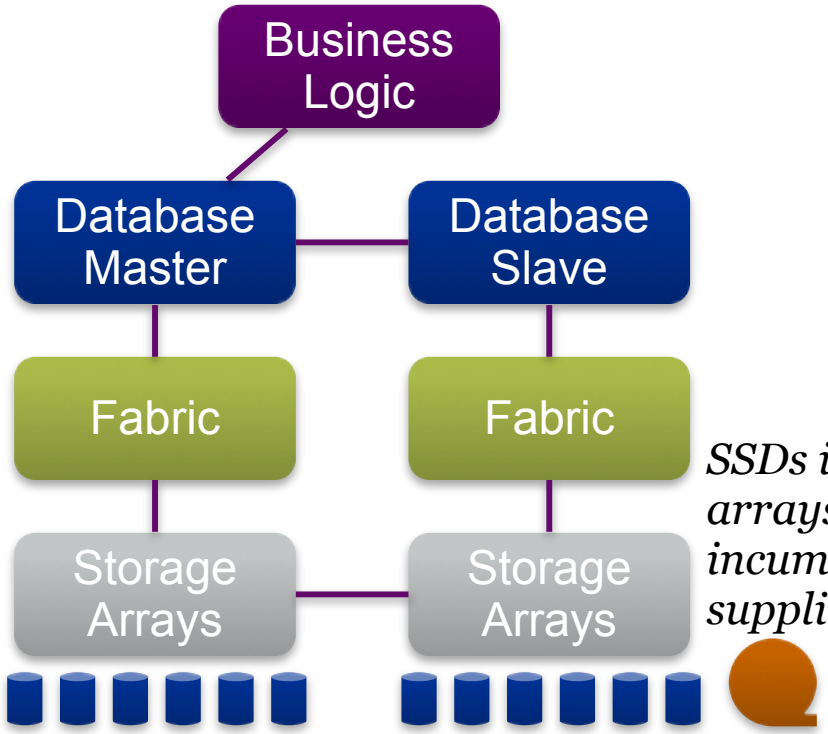
Ephemeral datastores using Dynomite, Memcached, Astyanax, Staash, Priam, Cassandra

Orchestration with Autoscalers on AWS, Titus exploring Mesos & ECS for Docker

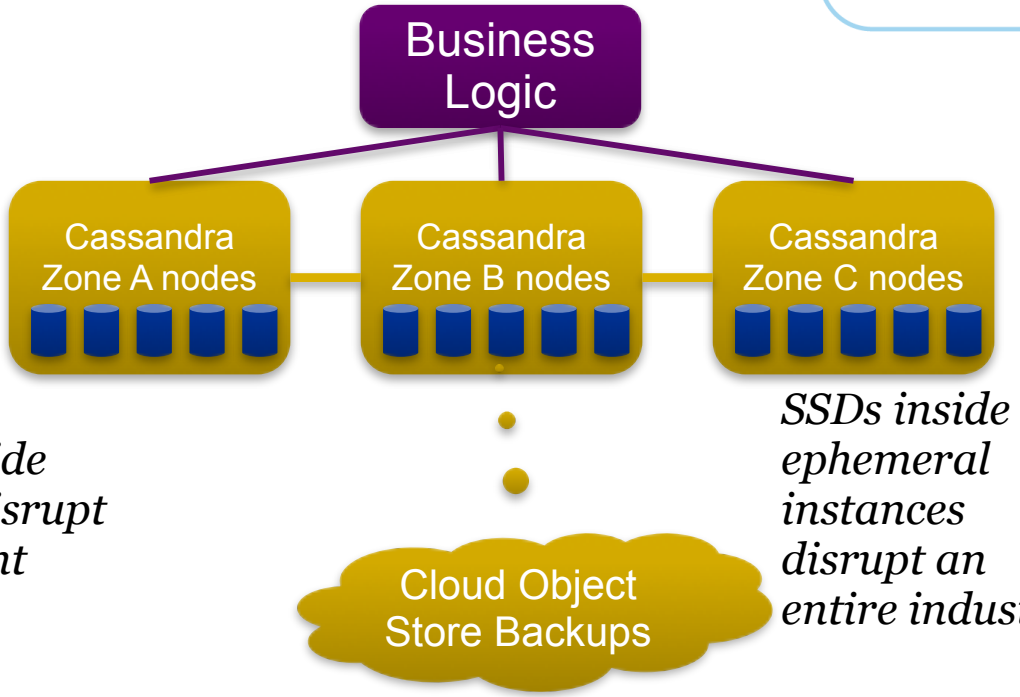
Development using Java, Groovy, Scala, Clojure, Python with AMI and Docker Containers

Policy via the Simian Army - Chaos Monkey, Chaos Gorilla, Conformity Monkey, Security Monkey

Cloud Native Storage



SSDs inside arrays disrupt incumbent suppliers



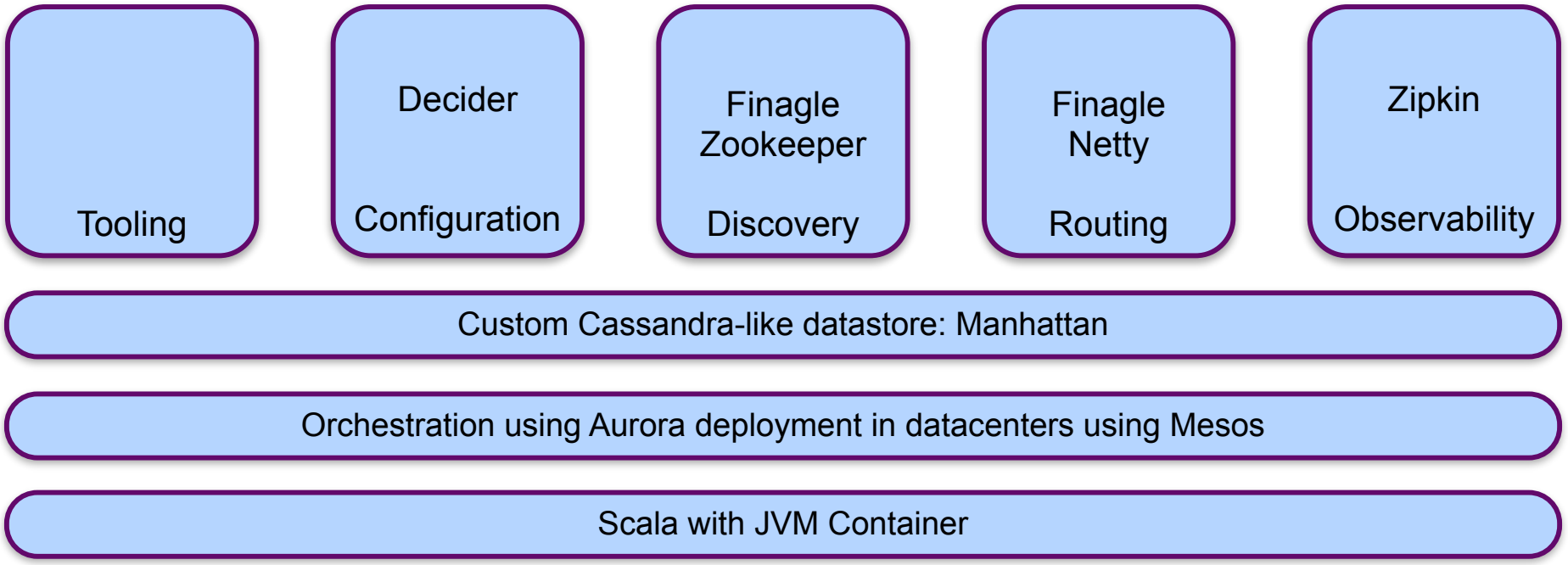
SSDs inside ephemeral instances disrupt an entire industry

NetflixOSS Uses Priam to create Cassandra clusters in minutes

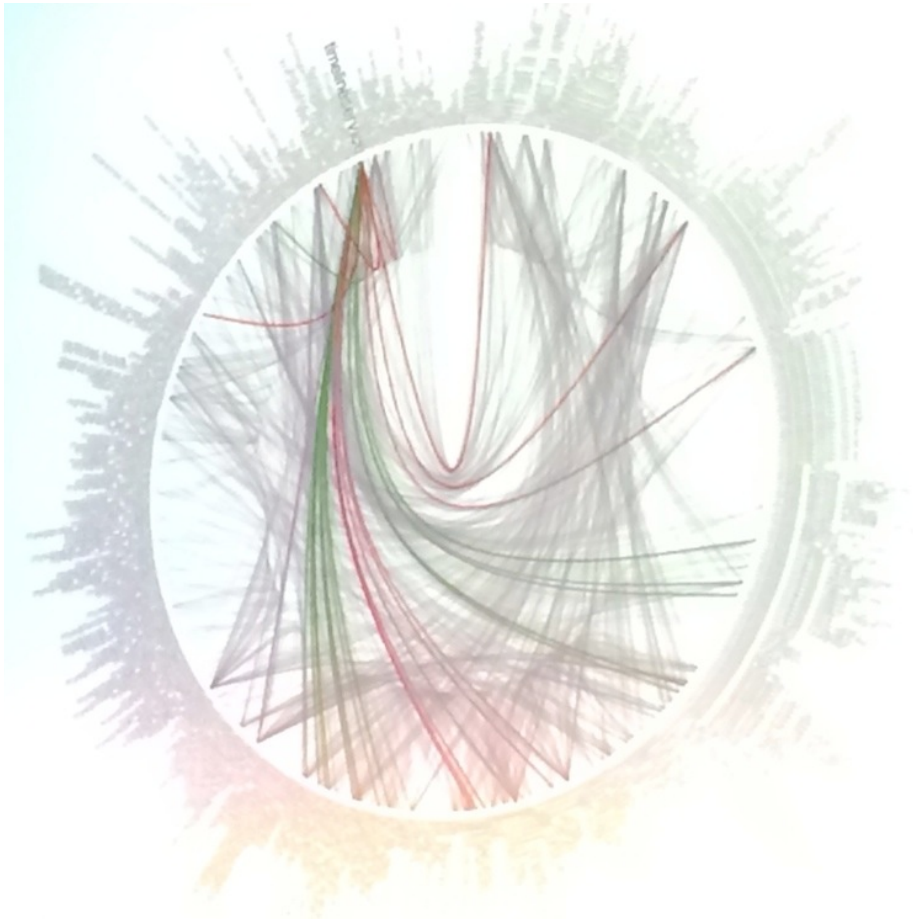

NETFLIX OSS
Netflix Open Source Software Center
Repositories
Powered By NetflixOSS
These companies are using and contributing to Netflix OSS Components
Email netflixoss@netflix.com to have your logo here.
RAPID7
BONOBOS
KNEWTON
rainforest
waze
nirmata
IBM
vennetics
swrve
KIXEYE
yammer
FullContact
flipkart.com
globus genomics
RIOT GAMES
coursera
yelp
Hotels.com
MORTAR
AnsWerS
YAHOO!
EUCALYPTUS
StumbleUpon
Maginatics
UserEvents
bazaarvoice:
OpenSCG
SUNCORP GROUP

NETFLIX

Twitter Microservices



Focus on efficient datacenter deployment at scale



Gilt Microservices

Ion Cannon
SBT
Rake

Tooling

Decider

Configuration

Finagle
Zookeeper

Discovery

Akka
Finagle
Netty

Routing

Zipkin

Observability

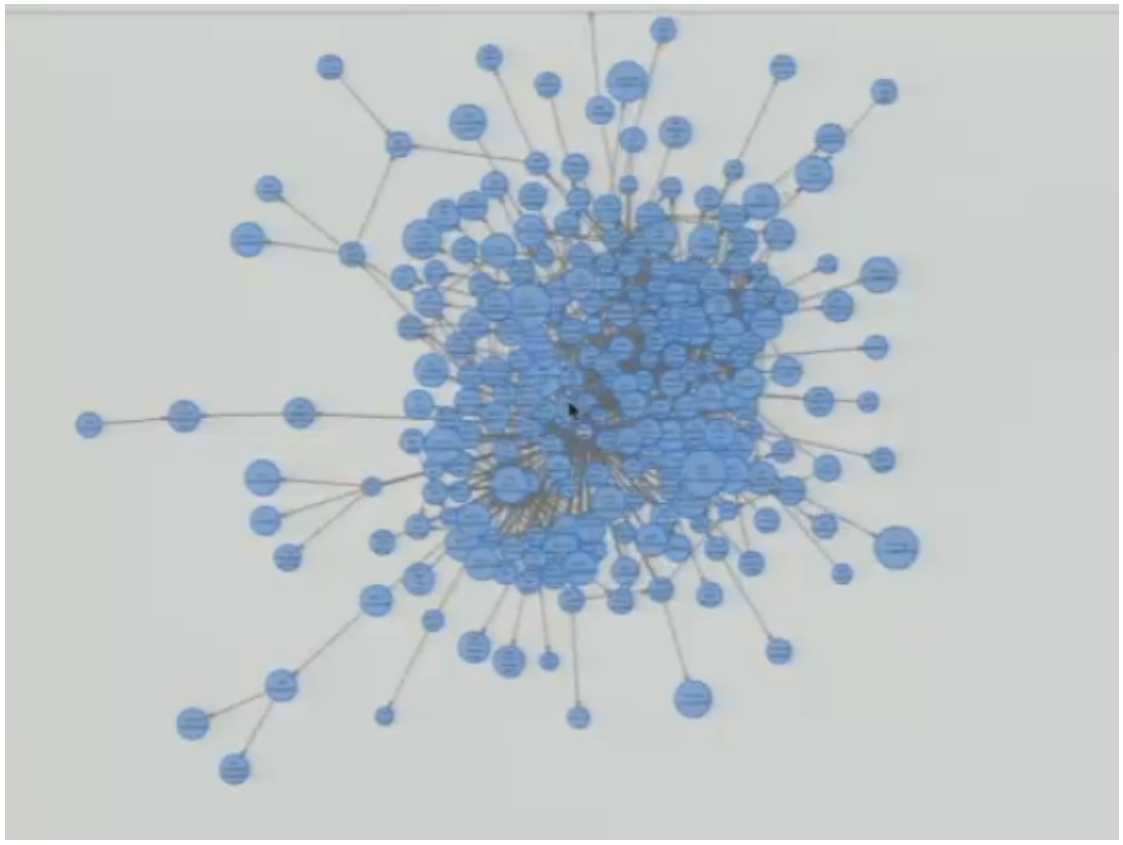
Datastores per Microservice using MongoDB, Postgres, Voldemort

Deployment on AWS

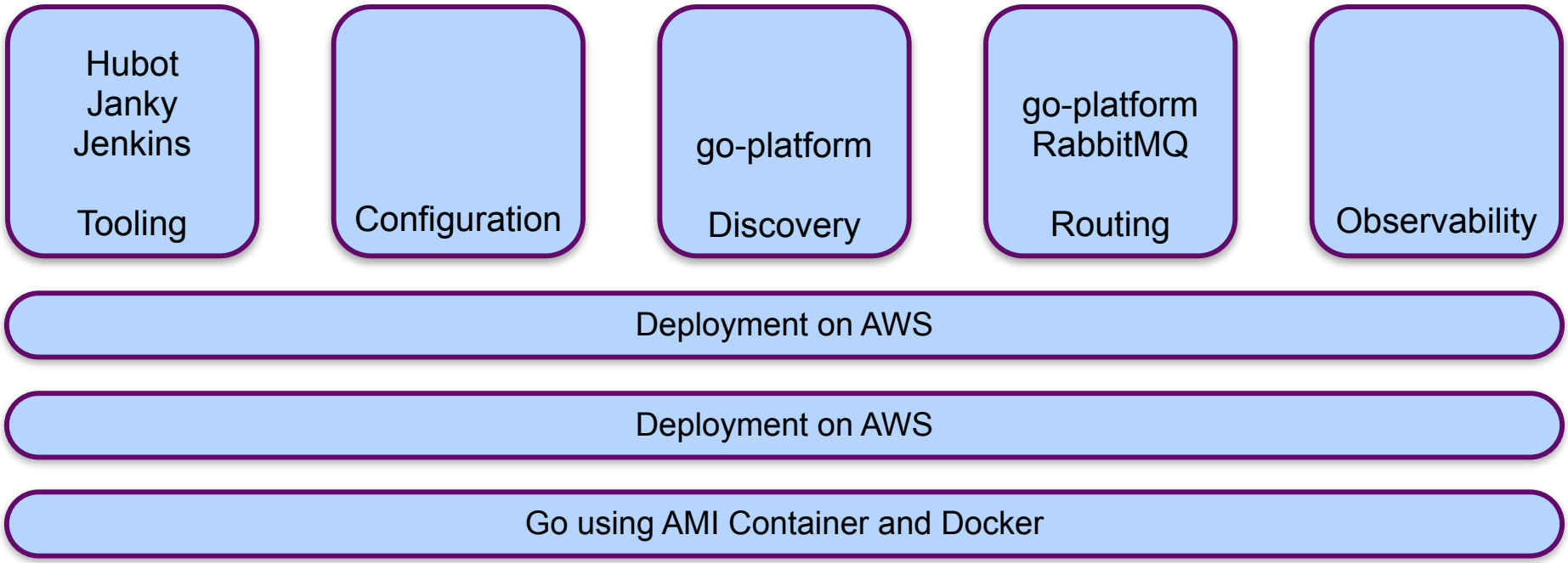
Scala and Ruby with Docker Containers

Focus on fast development with Scala and Docker

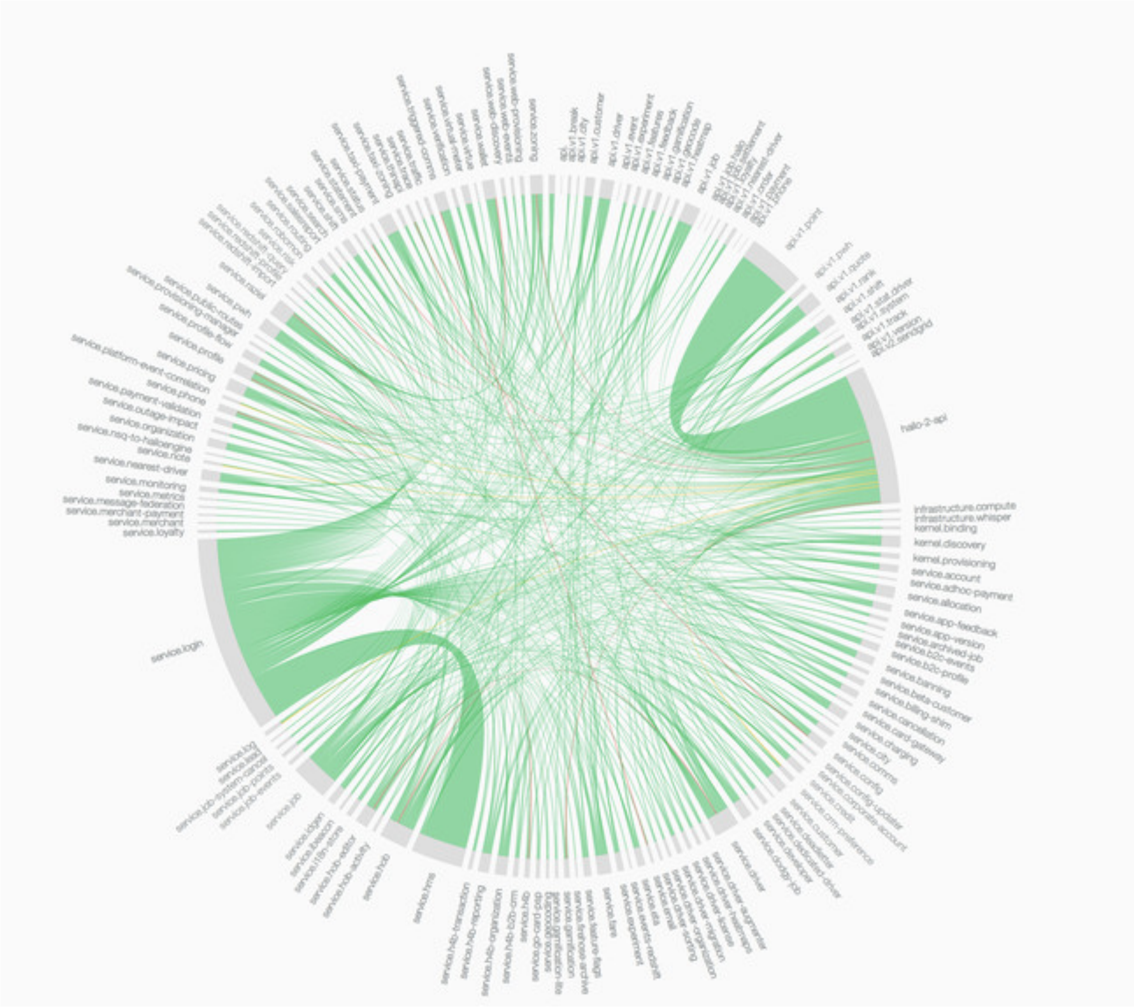
GILT



Hailo Microservices



See: go-micro and https://github.com/peterbourgon/gokit


hailo-2-api
infrastructure.compute
kernel.discovery
kernel.provisioning
service.account
service.allocation
service.login

Next Generation Applications

Fill in the gaps, rapidly evolving ecosystem choices

Docker
Desktop
Spinnaker
Terraform

Tooling

Wasabi
LaunchDarkly
Habitat

Configuration

Etcd
Eureka
Consul

Discovery

Compose
Linkerd
Weave

Routing

OpenTracing
Prometheus
Hystrix

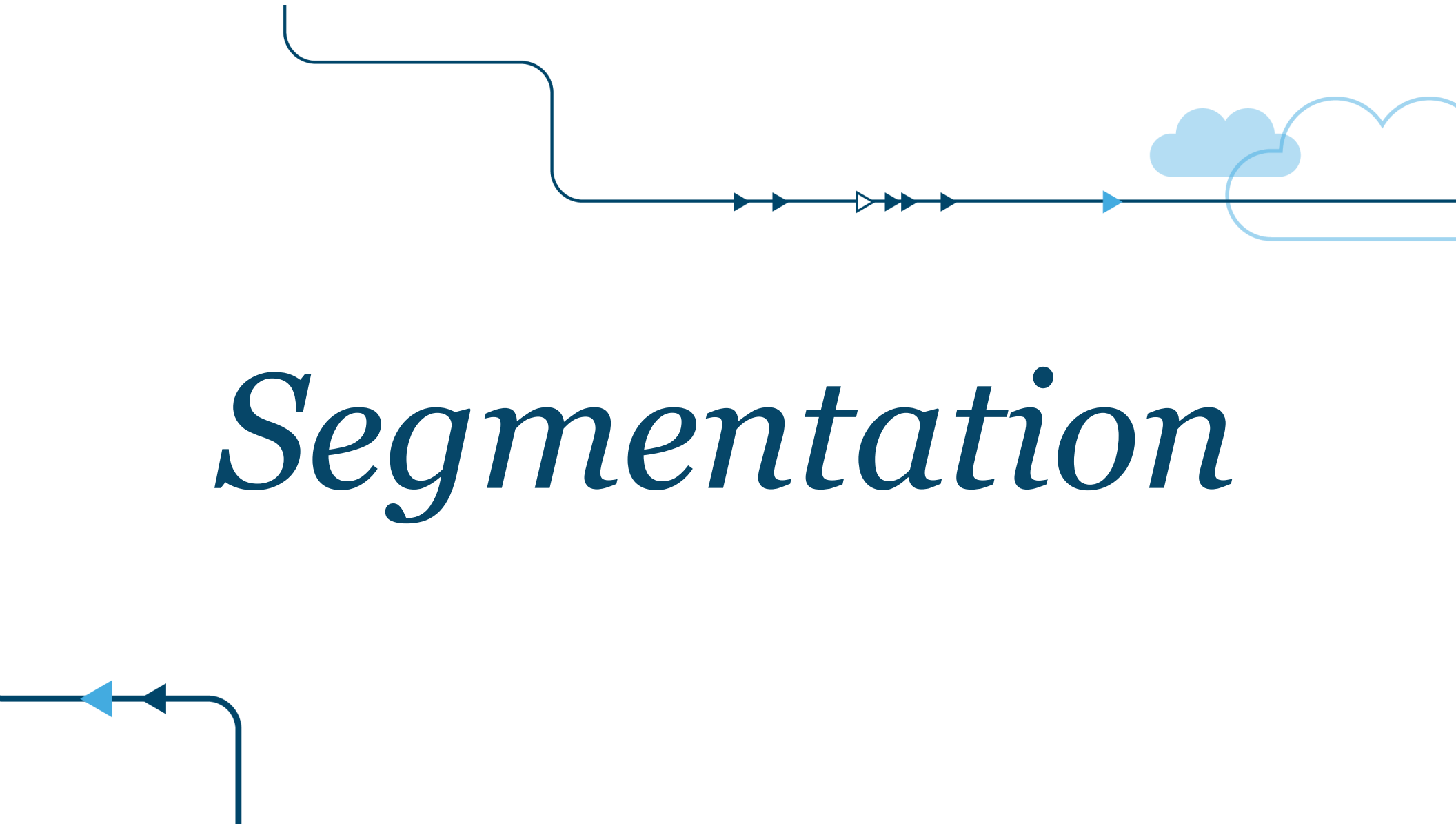
Observability

Datastores: Orchestrated, Distributed Ephemeral e.g. Cassandra, or DBaaS e.g. DynamoDB

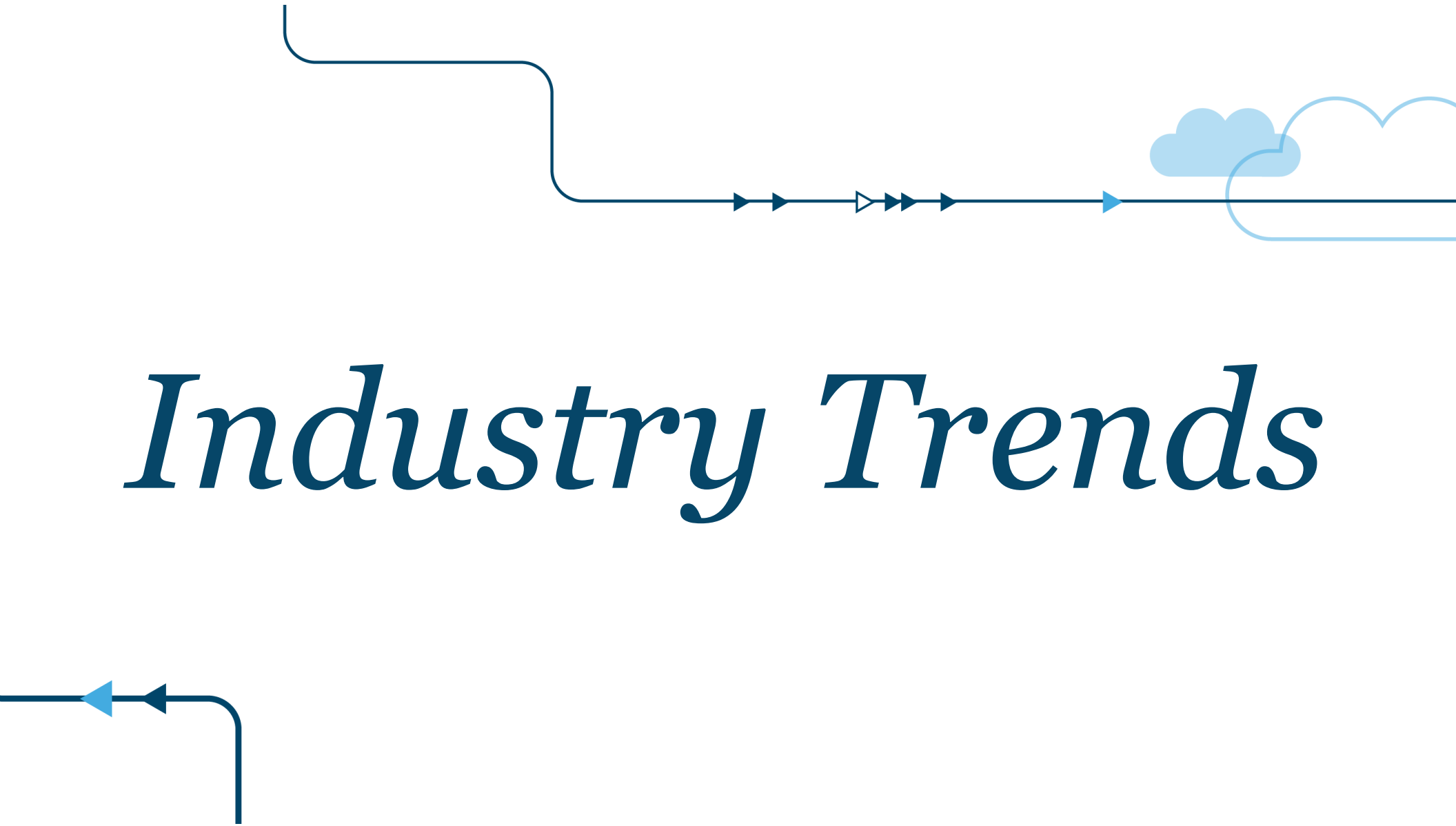
Operational: Mesos, Kubernetes, Swarm, Nomad for private clouds. ECS, GKS for public

Development: Containers and languages e.g. Docker, Lambda, Node.js, Go, Rust

Policy: Security compliance e.g. Vault, Docker Content Trust. Scaffolding e.g. Cloud Foundry



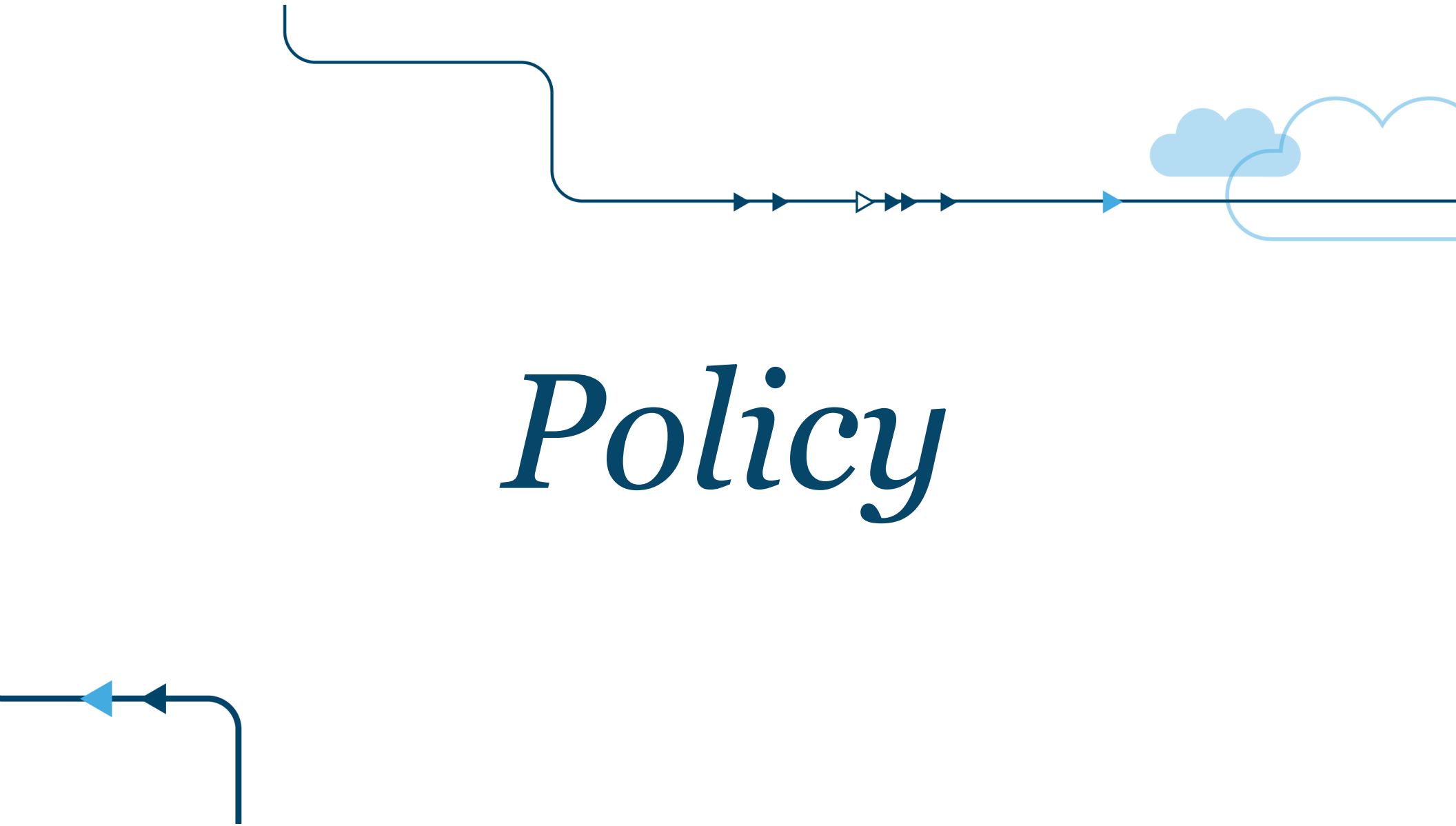
Segmentation



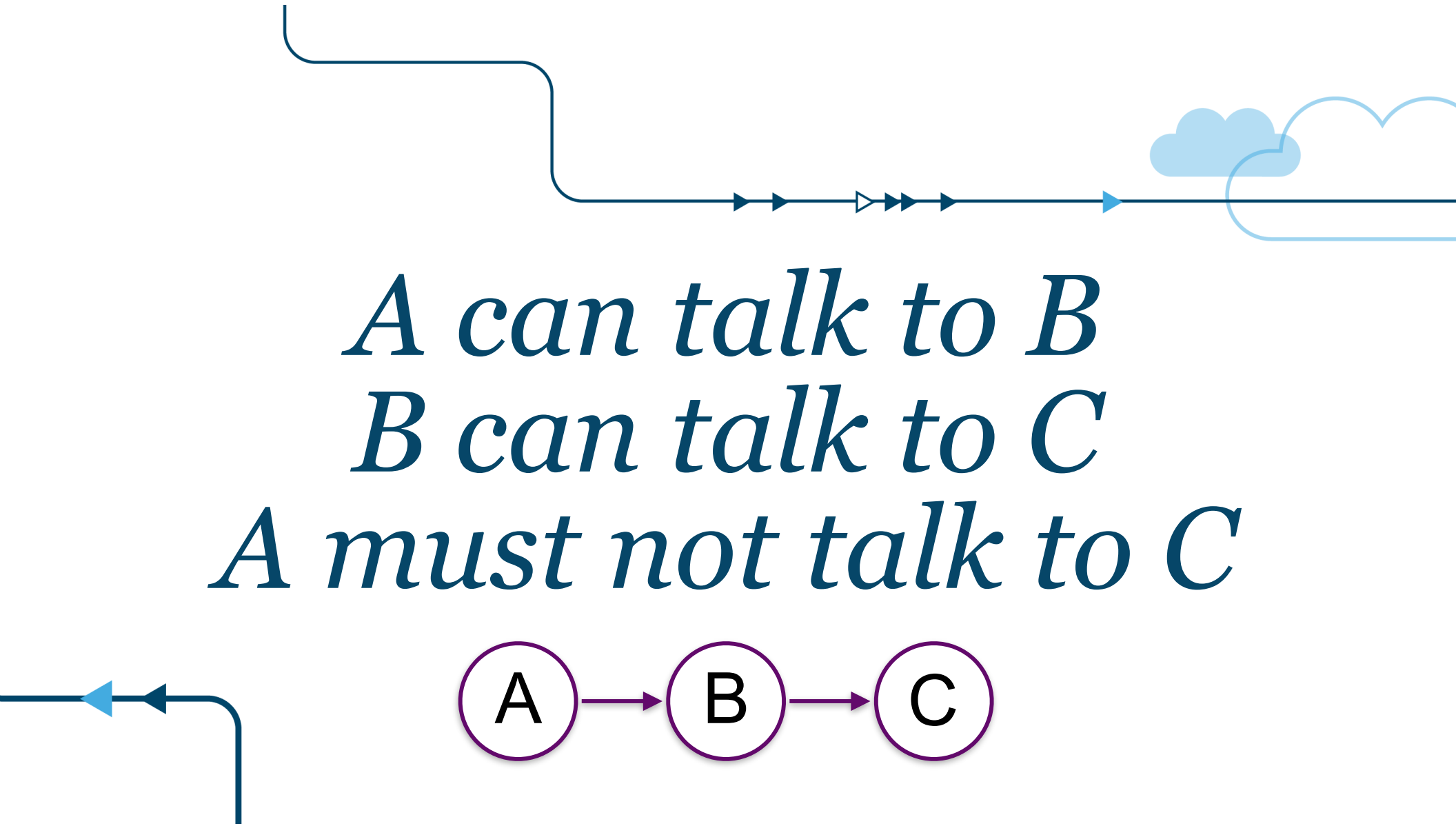
Industry Trends

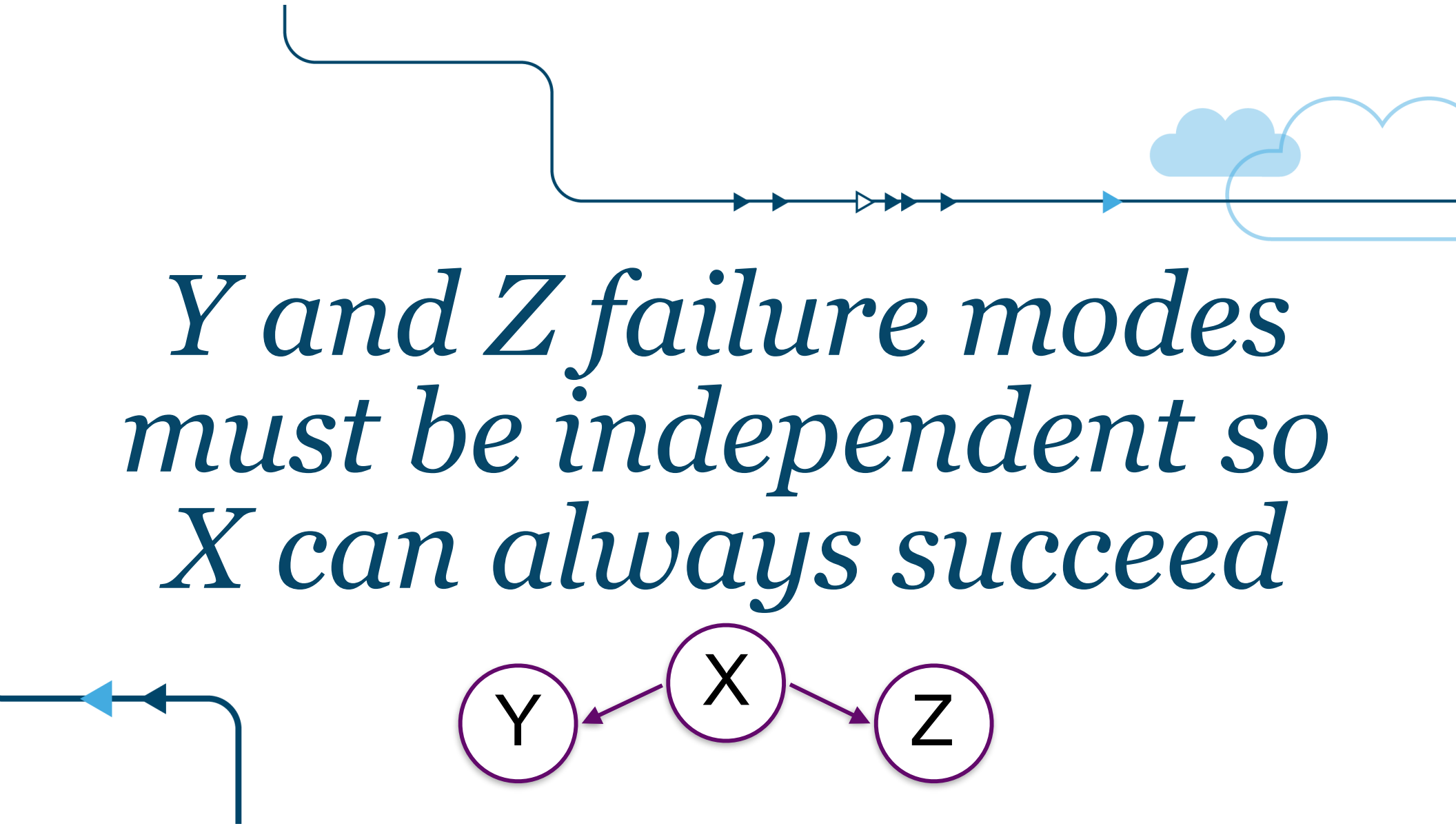
Airgaps closing
Industrial IoT
Security blanket perimeter firewalls
Datacenter to cloud transitions
New systems of engagement

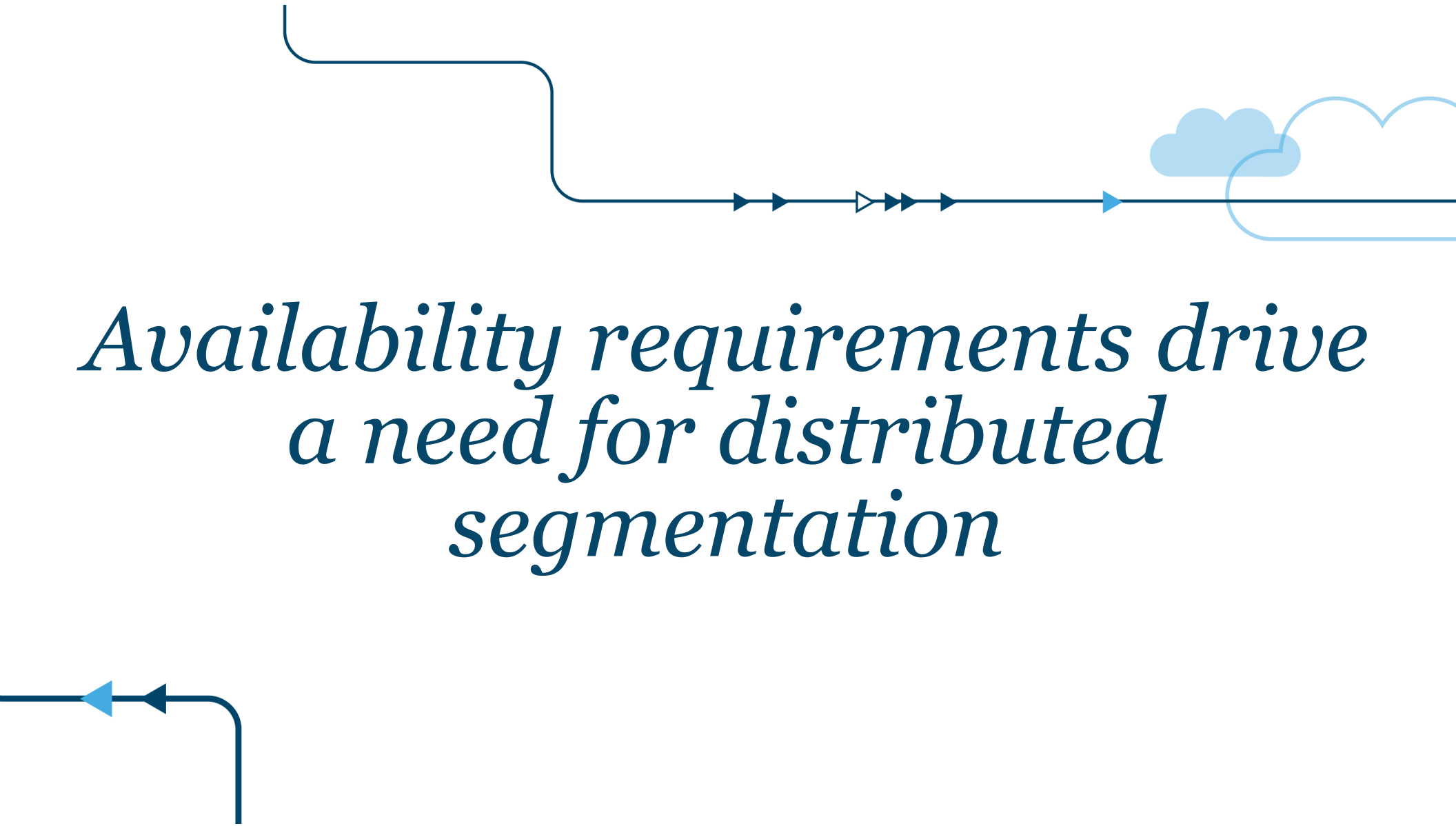
http://peanuts.wikia.com/wiki/Linus'_security_blanket



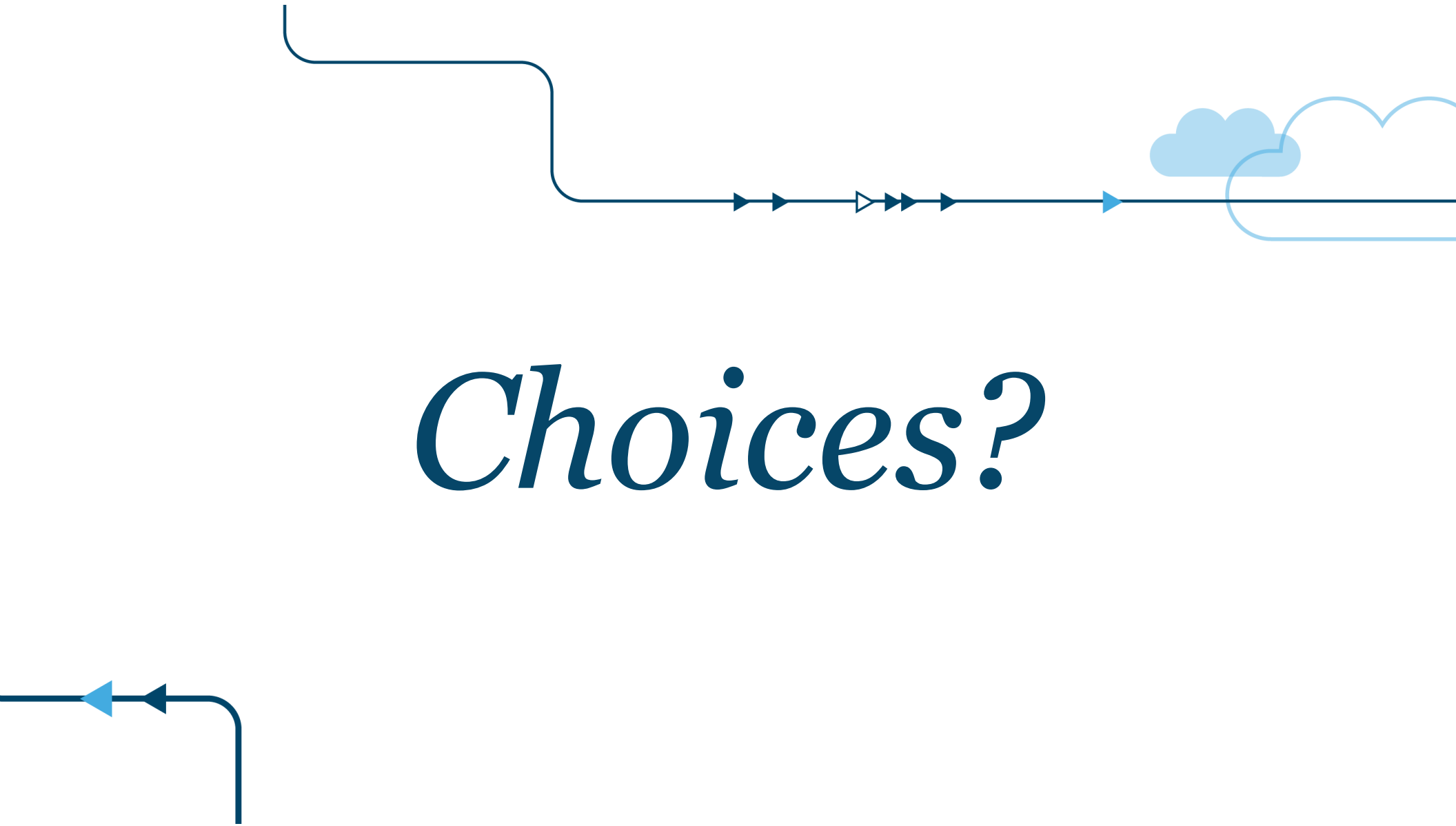
Policy


A can talk to B
B can talk to C
A must not talk to C
A
B
C

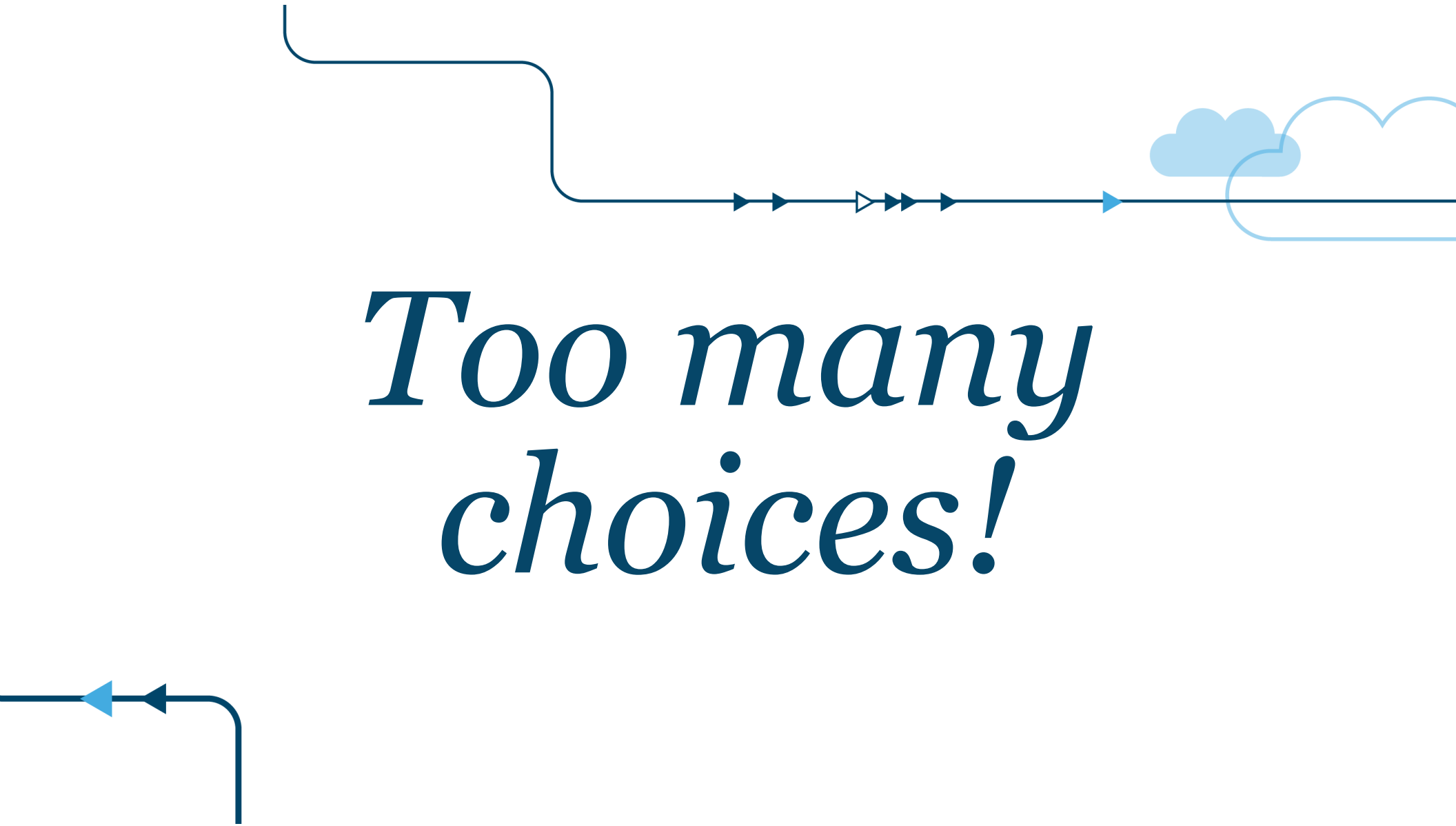

Y and Z failure modes must be independent so X can always succeed
X
Y
Z



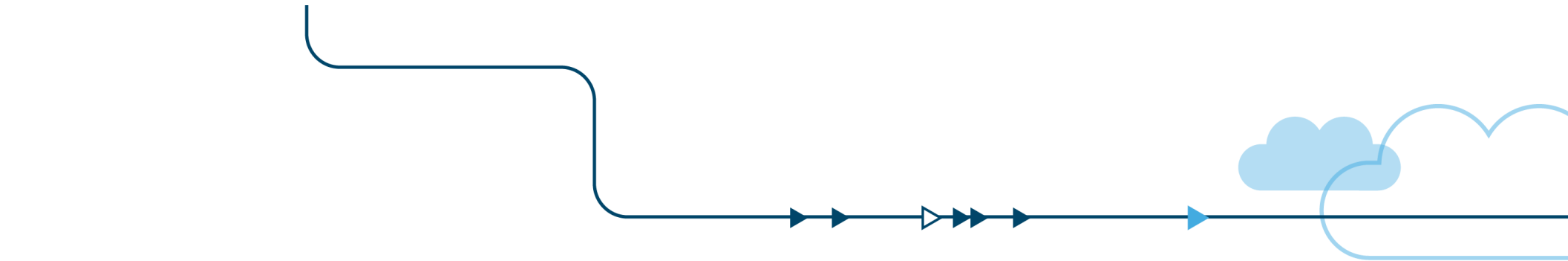
Availability requirements drive a need for distributed segmentation



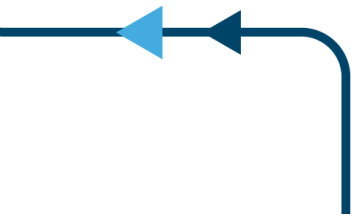
Choices?

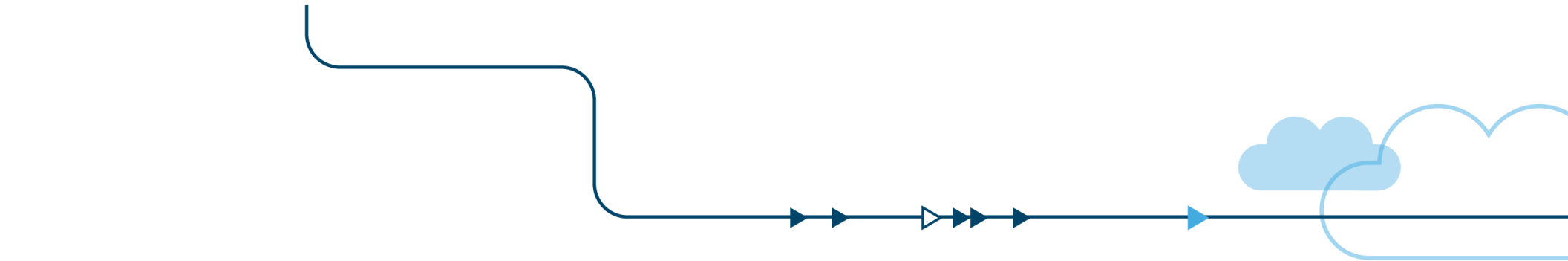


Too many choices!

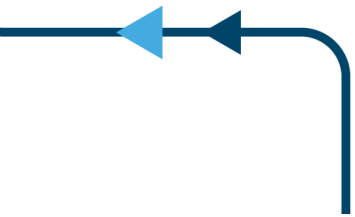


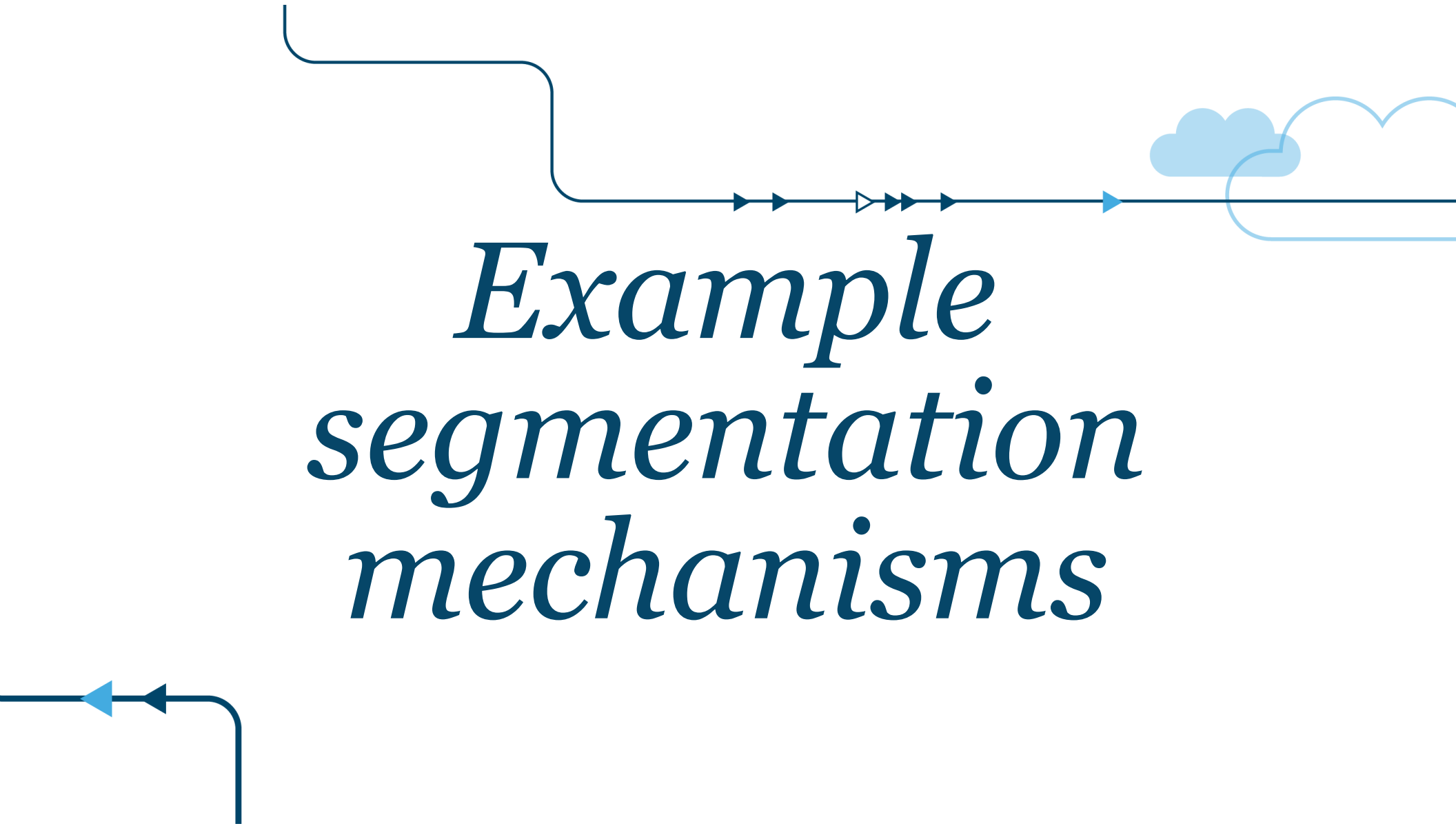
Over-reliance on one mechanism leads to abuse...





Lack of coordination across many mechanisms leads to fragility

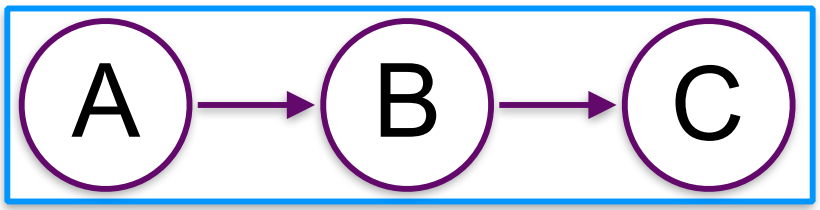




Example segmentation mechanisms

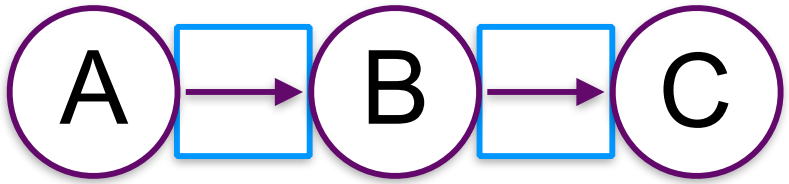
Accounts and Roles

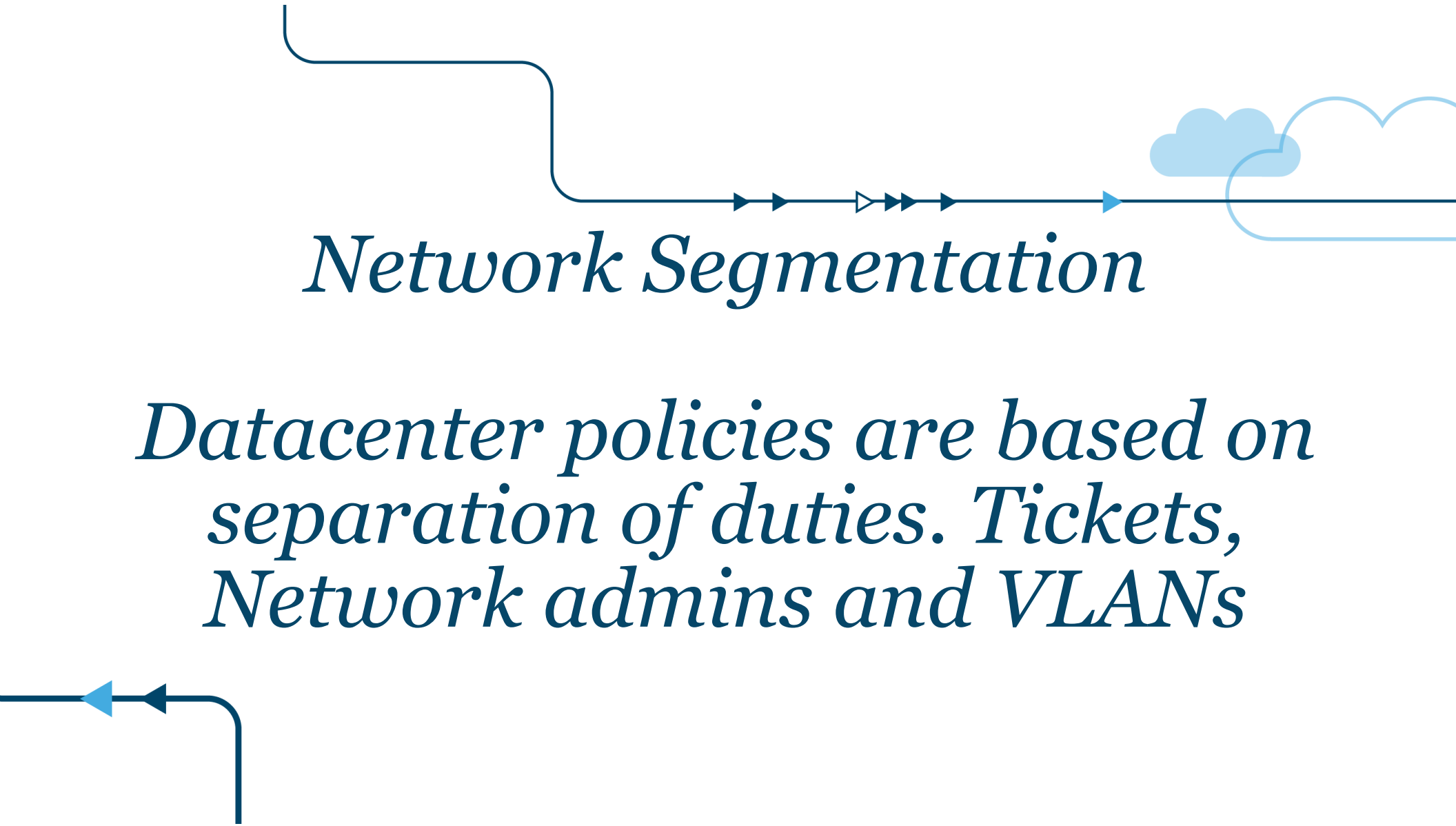
Who can set policy for what?
Needs distributed policy management



Network Segmentation

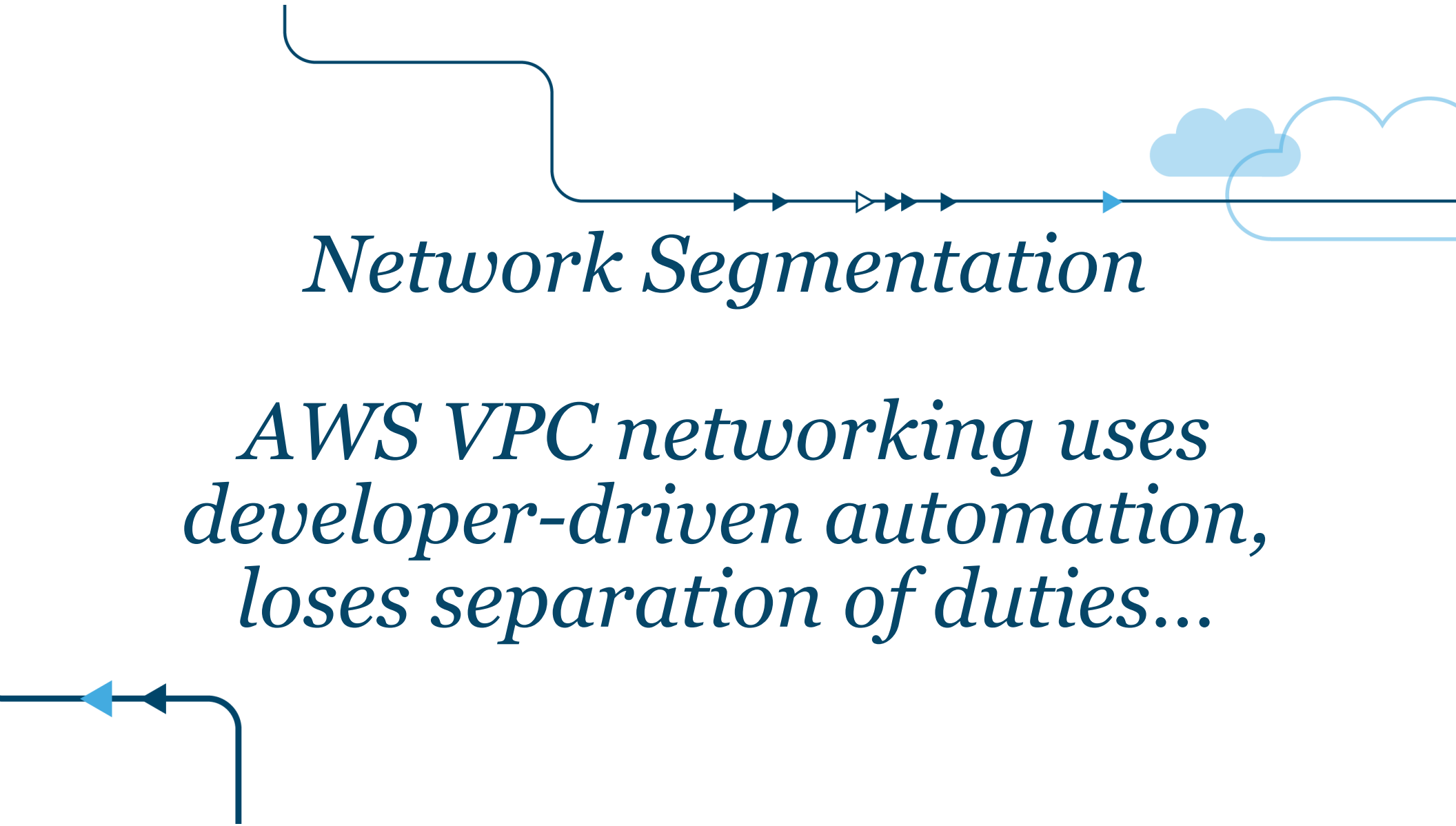
Who controls the network?





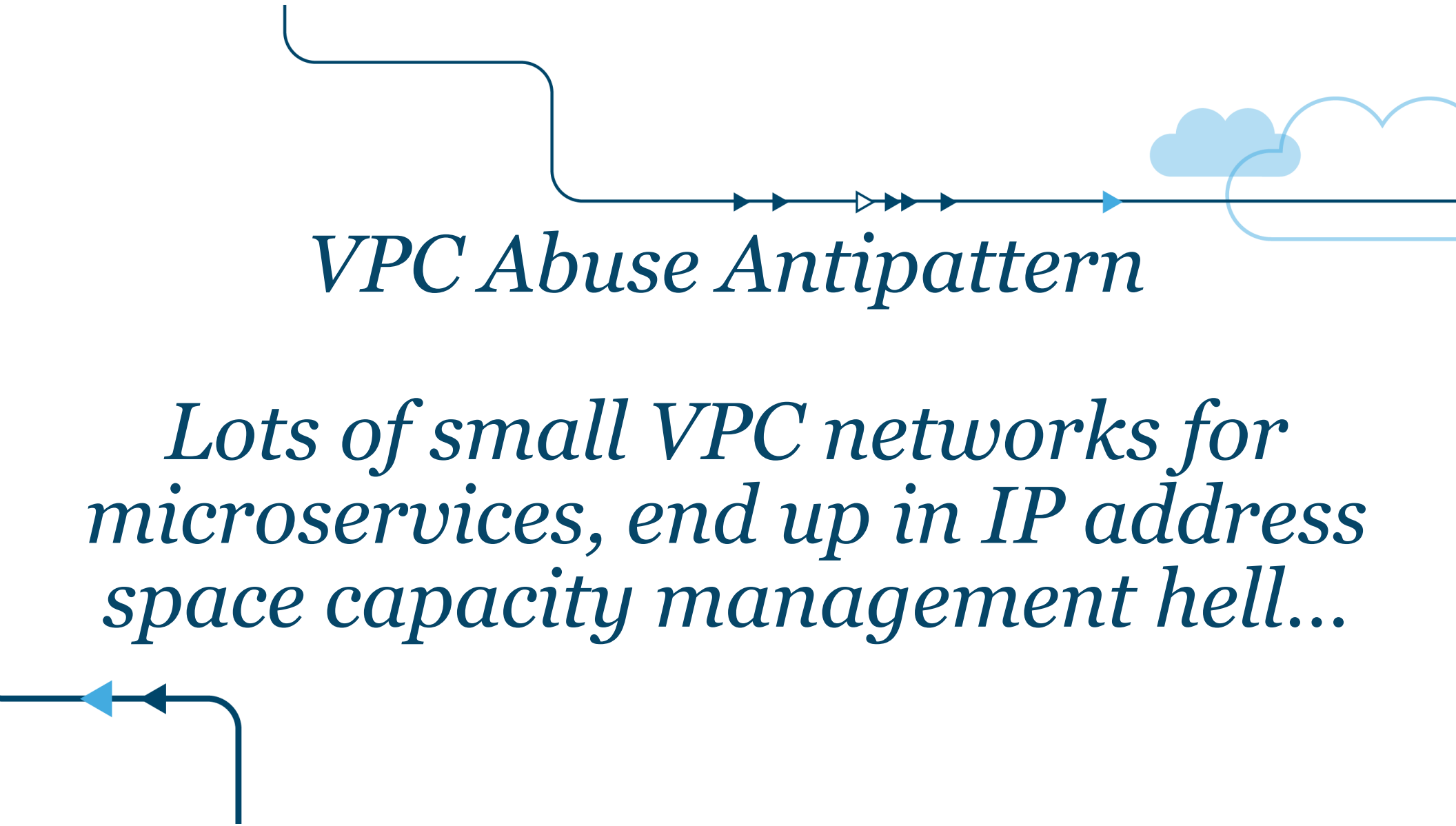
Network Segmentation

Datacenter policies are based on separation of duties. Tickets, Network admins and VLANs



Network Segmentation

AWS VPC networking uses developer-driven automation, loses separation of duties…

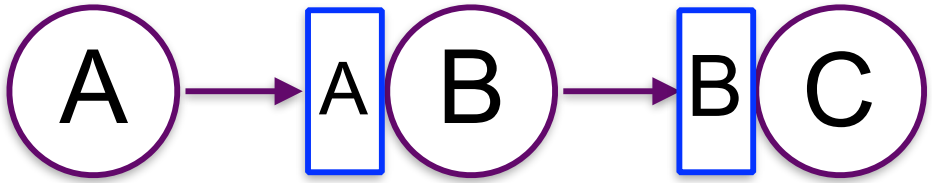


VPC Abuse Antipattern

Lots of small VPC networks for microservices, end up in IP address space capacity management hell...

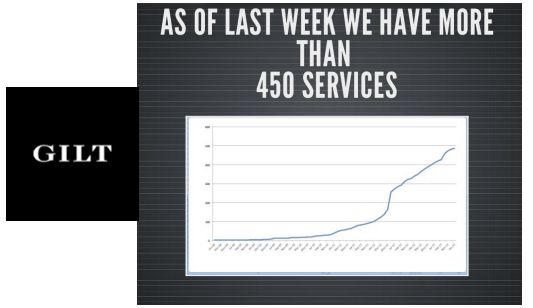
Hypervisor and Security Group Segmentation

Distributed firewall rules



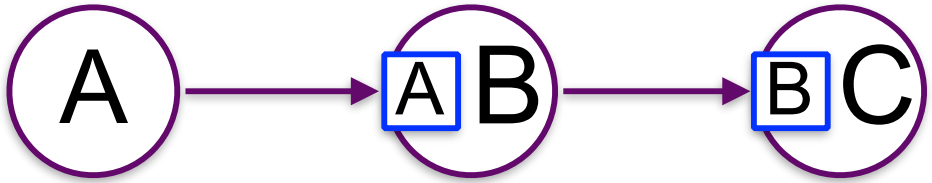
Security Group Abuse Antipattern

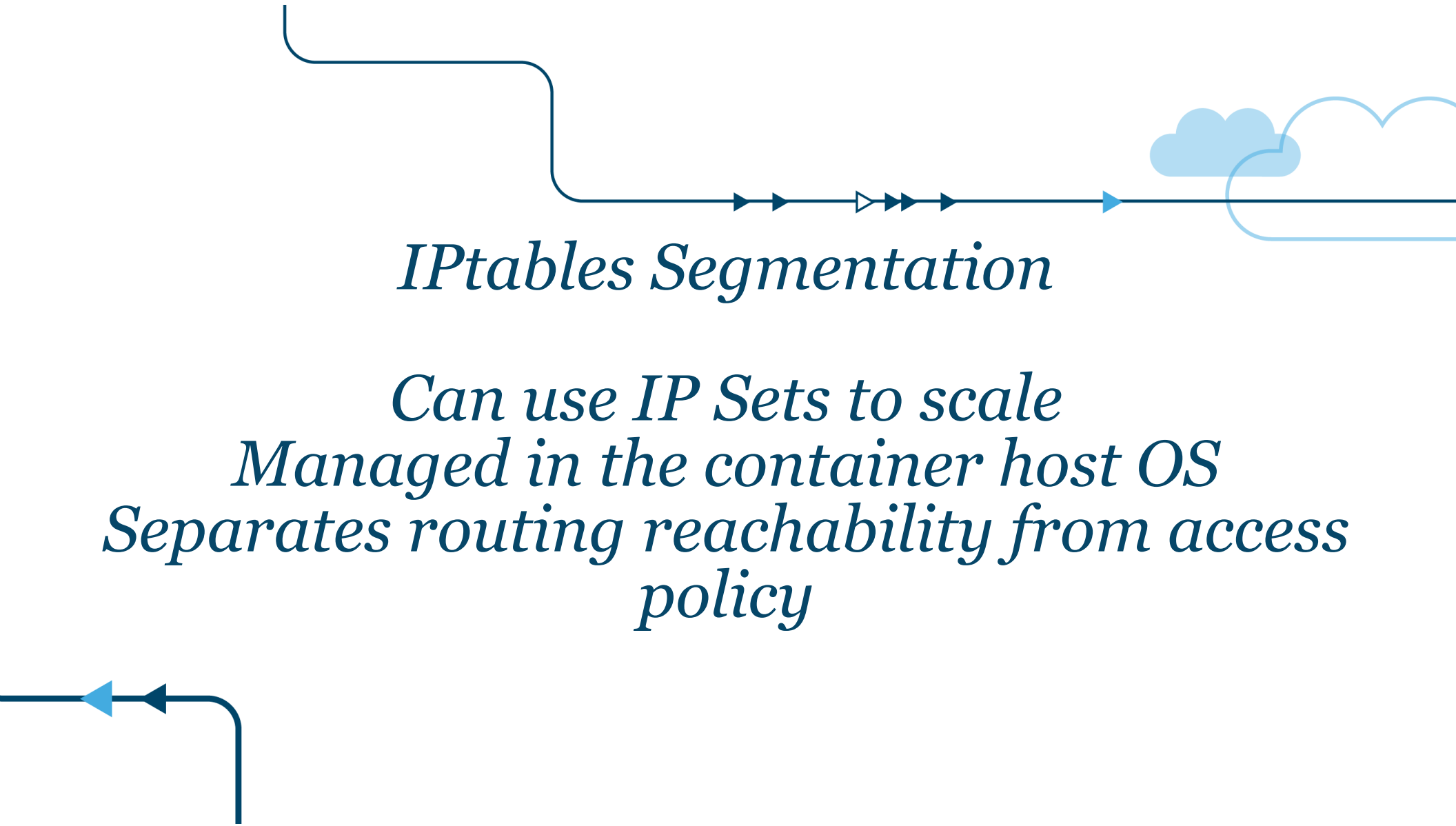
Too many microservices need to be in the same group, overloads configuration limitations



Kernel eBPF & Calico IPtables Segmentation

Distributed firewall rules



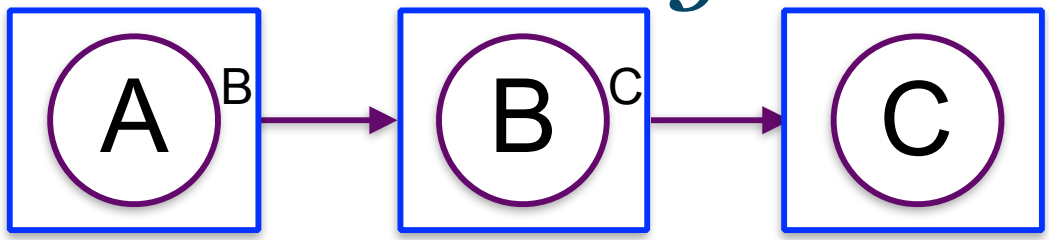


IPtables Segmentation

Can use IP Sets to scale
Managed in the container host OS
Separates routing reachability from access policy

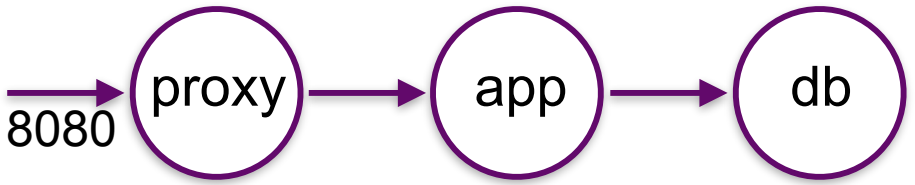
Docker & Weave Segmentation

Docker daemon manages connections



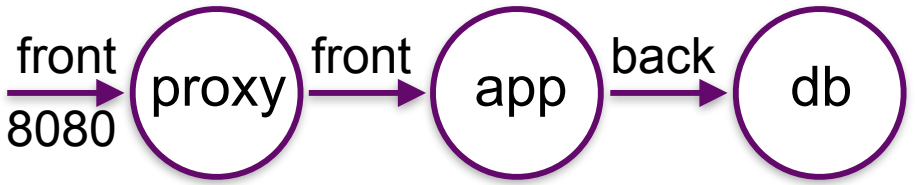
Docker Compose V1

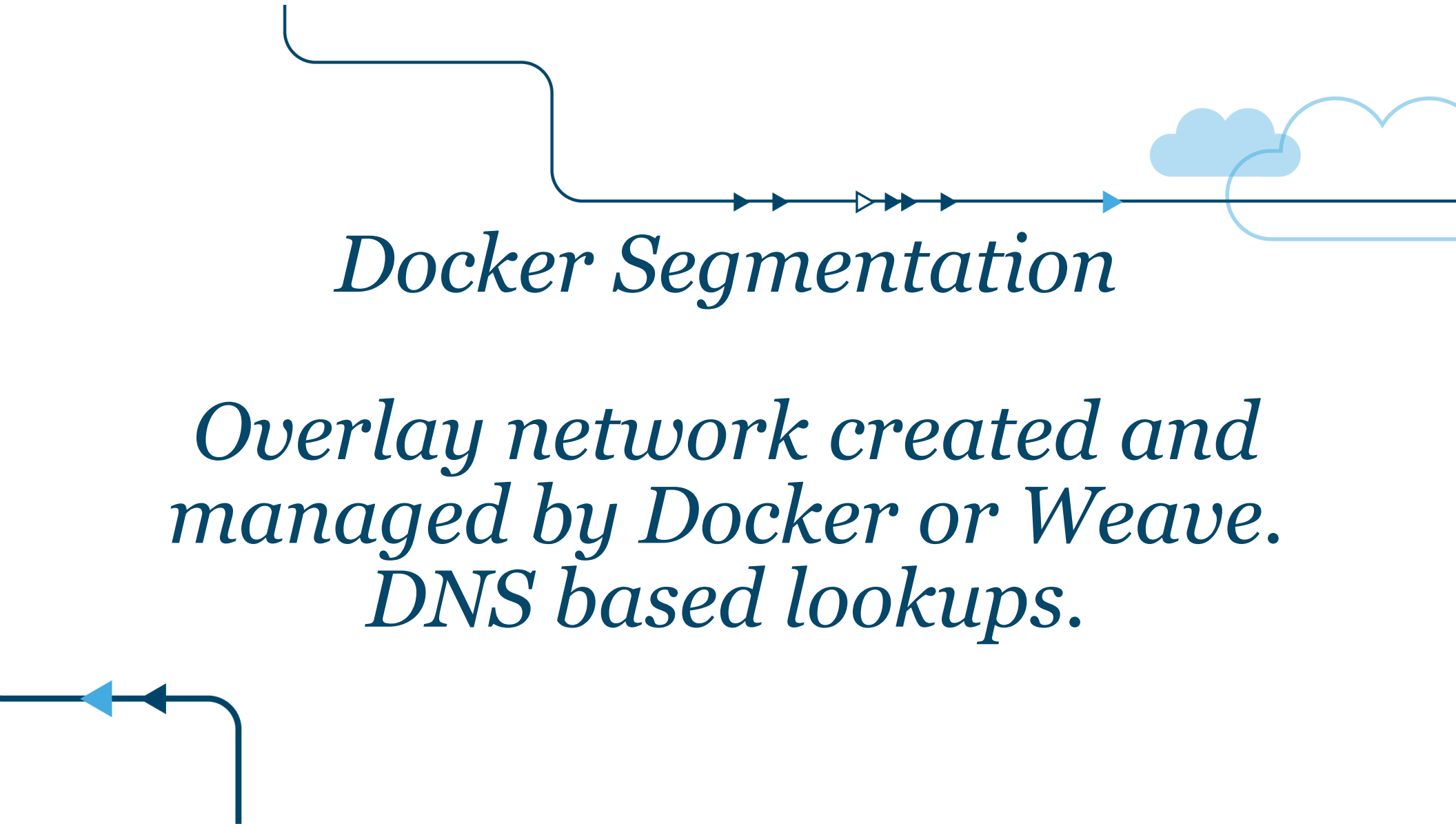
```
proxy:
  build: ./proxy
  ports:
    - "8080:8080"
  links:
    - app
app:
  build: ./app
  links:
    - db

db:
  image: postgres
```



Docker Compose V2

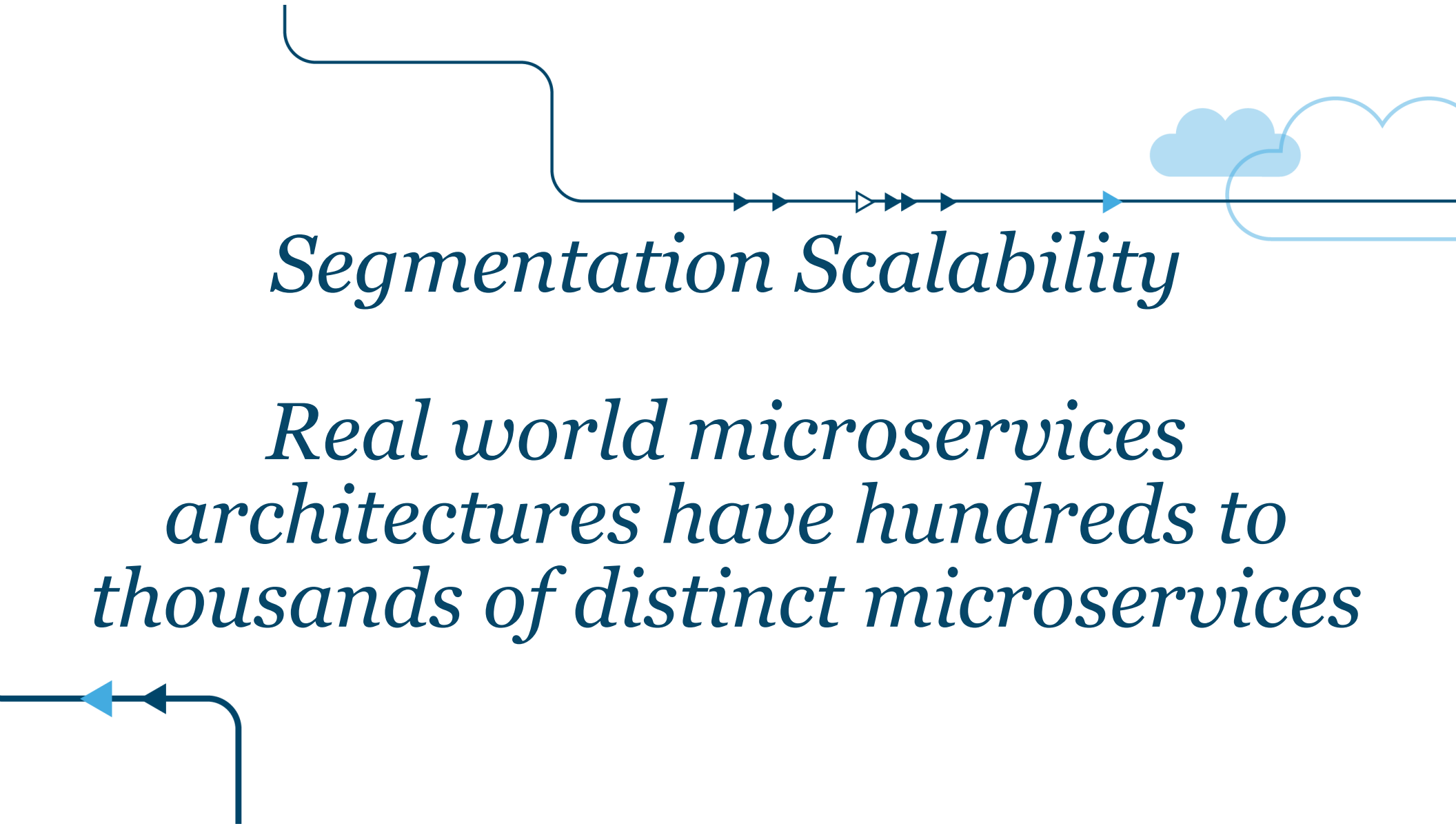
```
version: '2'

services:
  proxy:
    build: ./proxy
    ports:
      - "8080:8080"
    networks:
      - front
  app:
    build: ./app
    networks:
      - front
      - back
  db:
    image: postgres
    networks:
      - back

networks:
  front:
  back:
```





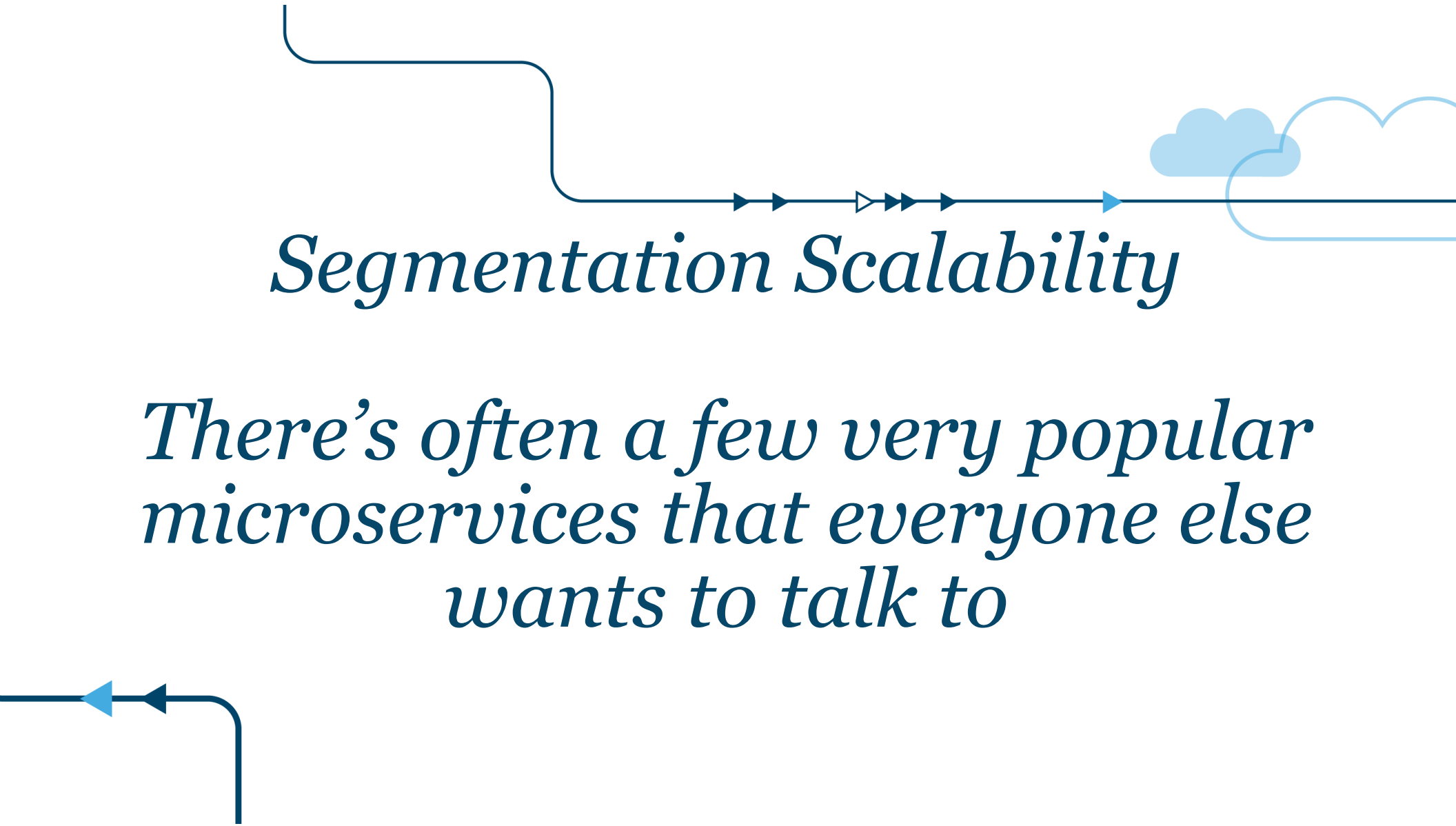
Docker Segmentation

Overlay network created and managed by Docker or Weave.
DNS based lookups.



Segmentation Scalability

Real world microservices architectures have hundreds to thousands of distinct microservices



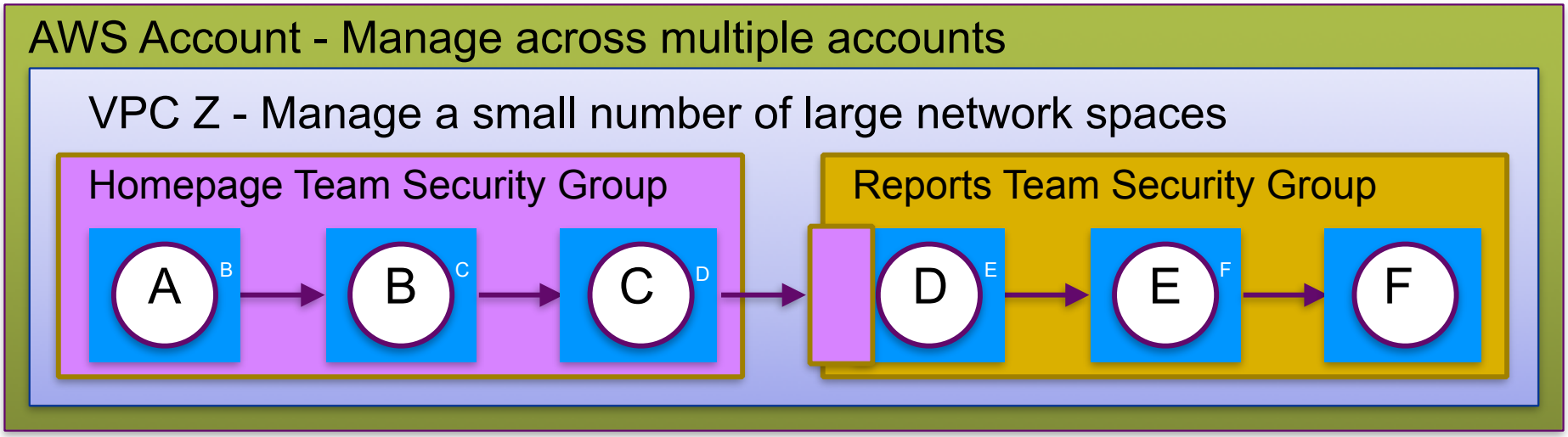
Segmentation Scalability

There's often a few very popular microservices that everyone else wants to talk to

In Search of Segmentation

Datacenters	*AWS Accounts*
AD/LDAP Roles	*IAM Roles*
VLAN Networks	*VPC*
Hypervisor	*Security Groups*
IPtables	*Calico Policy*
Docker Links	*Docker Net/Weave*

Hierarchical Segmentation



Setup all the layers with Terraform?

What's Missing?

Advanced Microservices Topics

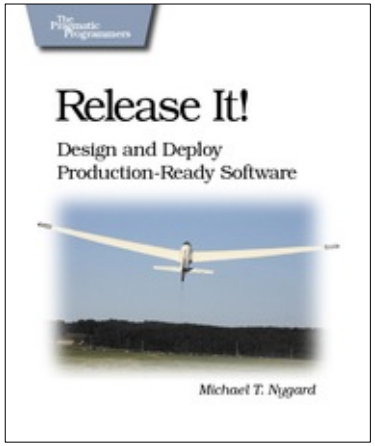
Failure injection testing
Versioning, routing
Binary protocols and interfaces
Timeouts and retries
Denormalized data models
Monitoring, tracing
Simplicity through symmetry

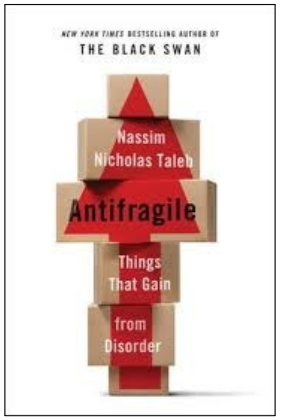
@adrianco
BV
Battery Ventures

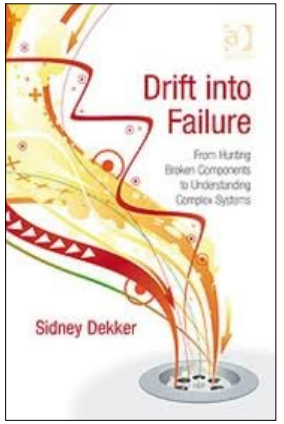
Failure Injection Testing

Netflix Chaos Monkey, Simian Army, FIT and Gremlin

http://techblog.netflix.com/2011/07/netflix-simian-army.html
http://techblog.netflix.com/2014/10/fit-failure-injection-testing.html
http://techblog.netflix.com/2016/01/automated-failure-testing.html
https://www.infoq.com/presentations/failure-test-research-netflix







@adrianco
BV
Battery Ventures

NETFLIX | OSS Trust with Verification

- *Chaos Monkey - enforcing stateless business logic*
- *Chaos Gorilla - enforcing zone isolation/replication*
- *Chaos Kong - enforcing region isolation/replication*
- *Security Monkey - watching for insecure configuration settings*
- *FIT & Gremlin - inject errors to enforce robust dependencies*
- *See over 100 NetflixOSS projects at netflix.github.com*
- *Get "Technical Indigestion" reading techblog.netflix.com*

Benefits of version aware routing

Immediately and safely introduce a new version
Canary test in production
Use DIY feature flags, 

, A|B tests with Wasabi

Route clients to a version so they can't get disrupted
Change client or dependencies but not both at once

Eventually remove old versions
Incremental or infrequent "break the build" garbage collection

Versioning, Routing

Version numbering: Interface.Feature.Bugfix

V1.2.3 to V1.2.4 - Canary test then remove old version

V1.2.x to V1.3.x - Canary test then remove or keep both
Route V1.3.x clients to new version to get new feature
Remove V1.2.x only after V1.3.x is found to work for V1.2.x clients

V1.x.x to V2.x.x - Route clients to specific versions
Remove old server version when all old clients are gone

@adrianco
BV
Battery Ventures

Protocols

Measure serialization, transmission, deserialization costs

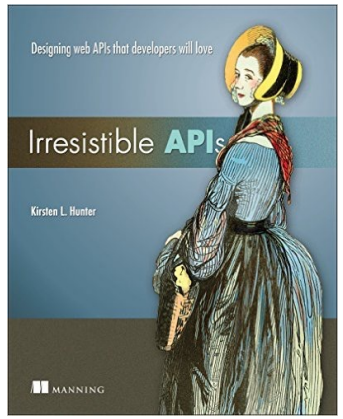
Sending a megabyte of XML between microservices will make you sad, but not as sad as 10yrs ago with SOAP

Use Thrift, Protobuf/gRPC, Avro, SBE internally
Use JSON for external/public interfaces
https://github.com/real-logic/simple-binary-encoding

Interfaces

When you build a service, build a "driver" client for it
Reference implementation error handling and serialization
Release automation stress test using client
Validate that service interface is usable!
Minimize additional dependencies

Swagger - OpenAPI Specification
Datawire Quark adds behaviors to API spec



@adrianco
BV
Battery Ventures

Interface Version Pinning

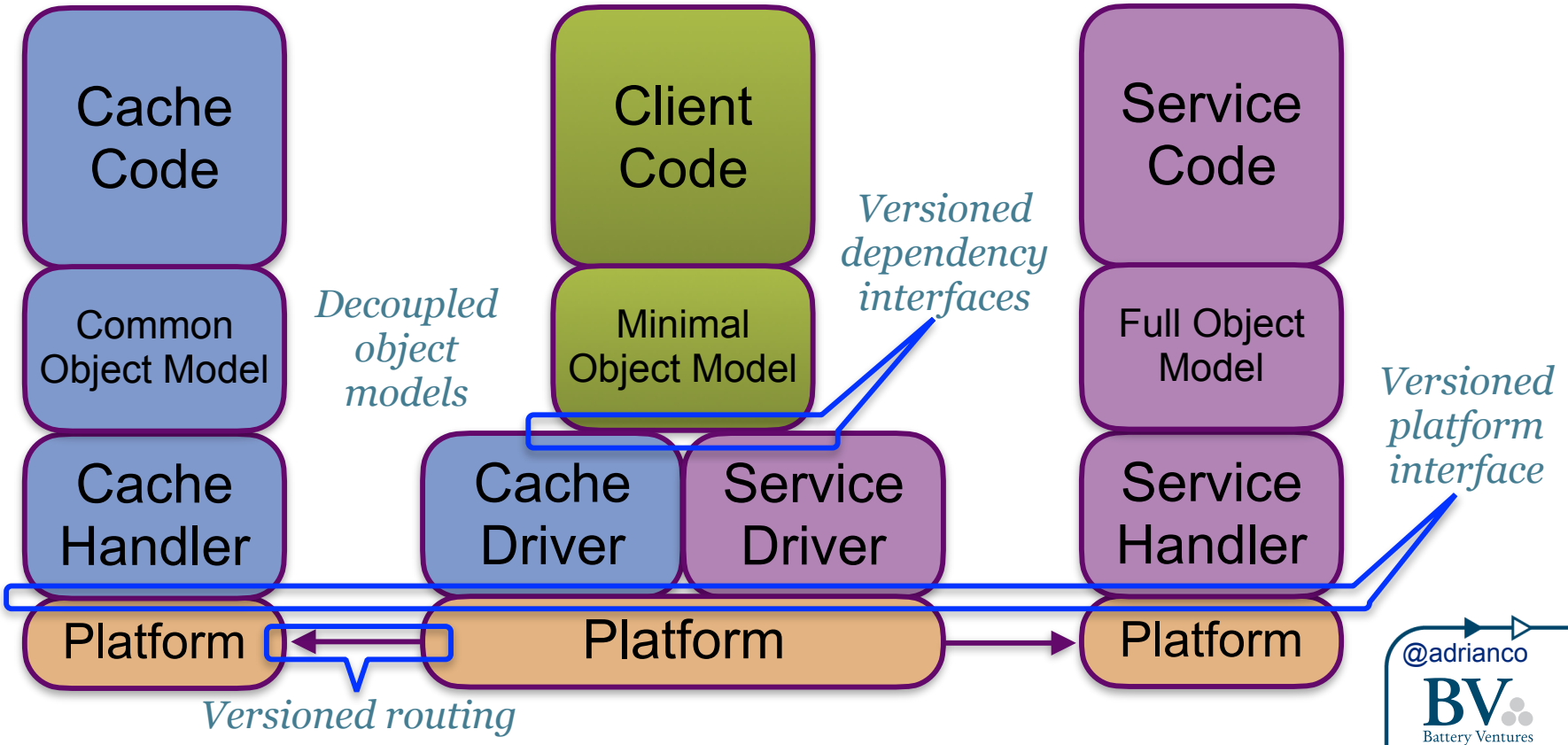
Change one thing at a time!
Pin the version of everything else

Incremental build/test/deploy pipeline

Deploy existing app code with new platform
Deploy existing app code with new dependencies
Deploy new app code with pinned platform/dependencies

@adrianco
BV
Battery Ventures

Interfaces between teams



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

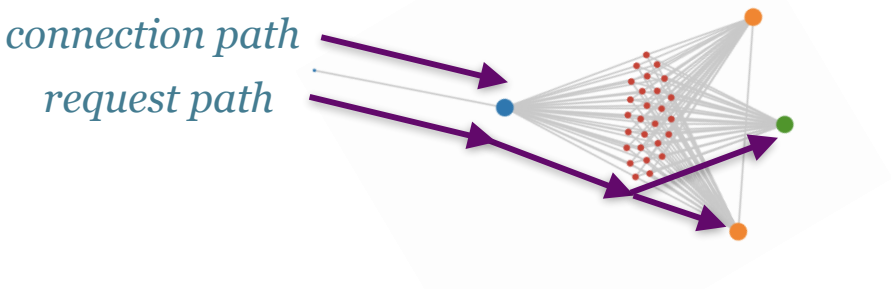
@adrianco
BV
Battery Ventures

Connections and Requests

TCP makes a connection, HTTP makes a request
HTTP hopefully reuses connections for several requests

Both have different timeout and retry needs!

TCP timeout is purely a property of one network latency hop
HTTP timeout depends on the service and its dependencies



@adrianco
BV
Battery Ventures

Timeouts and Retries

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

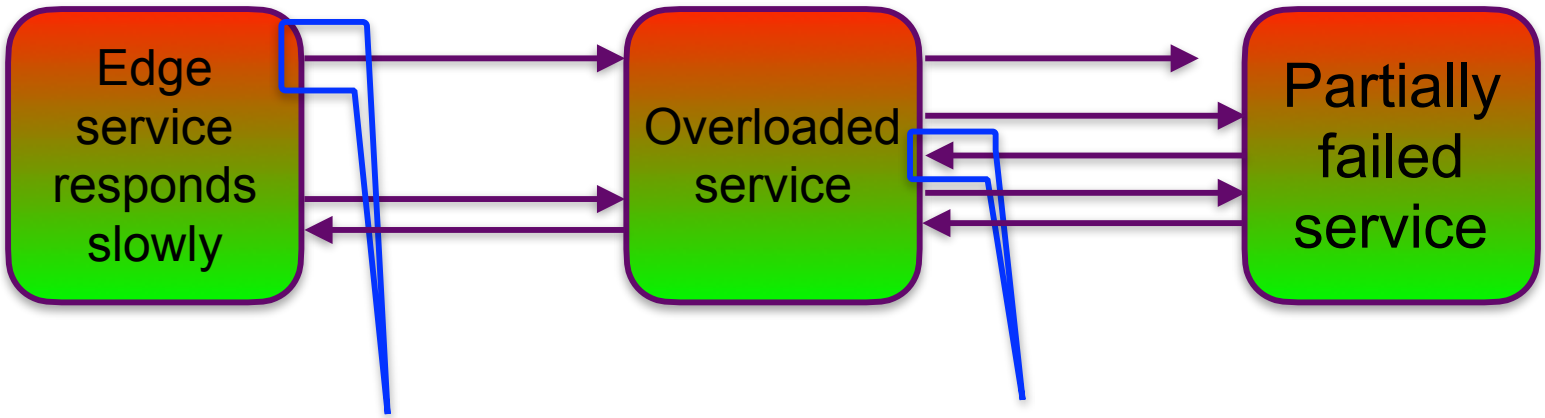
If anything breaks, everything upstream stops responding

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

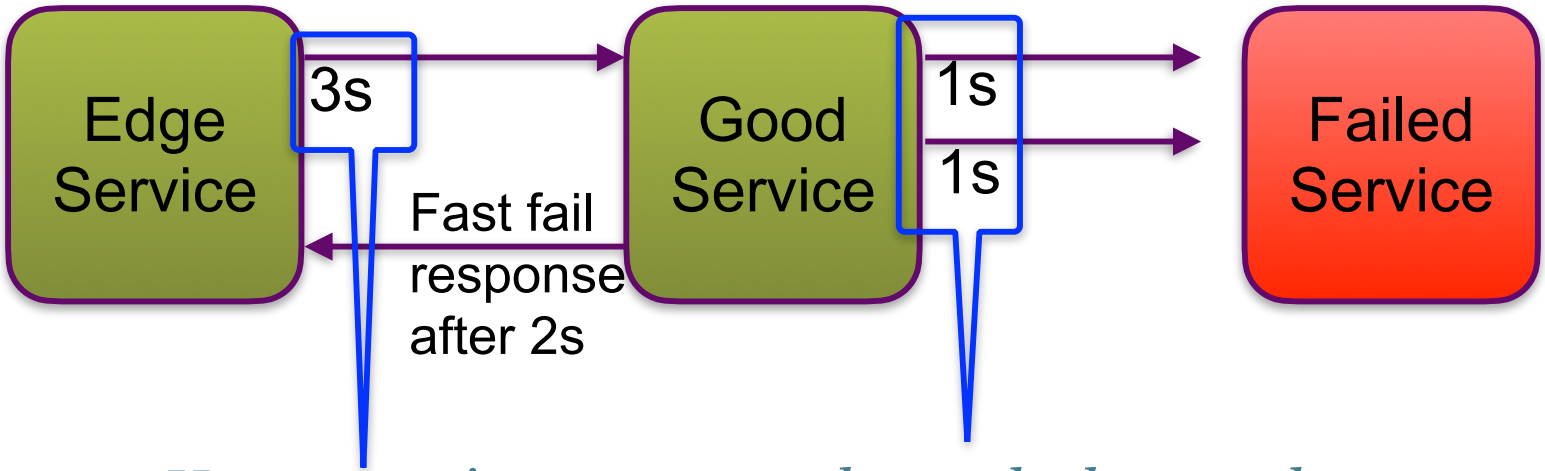
Cascading timeout budget
Static settings that decrease from the edge
or dynamic budget passed with request

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

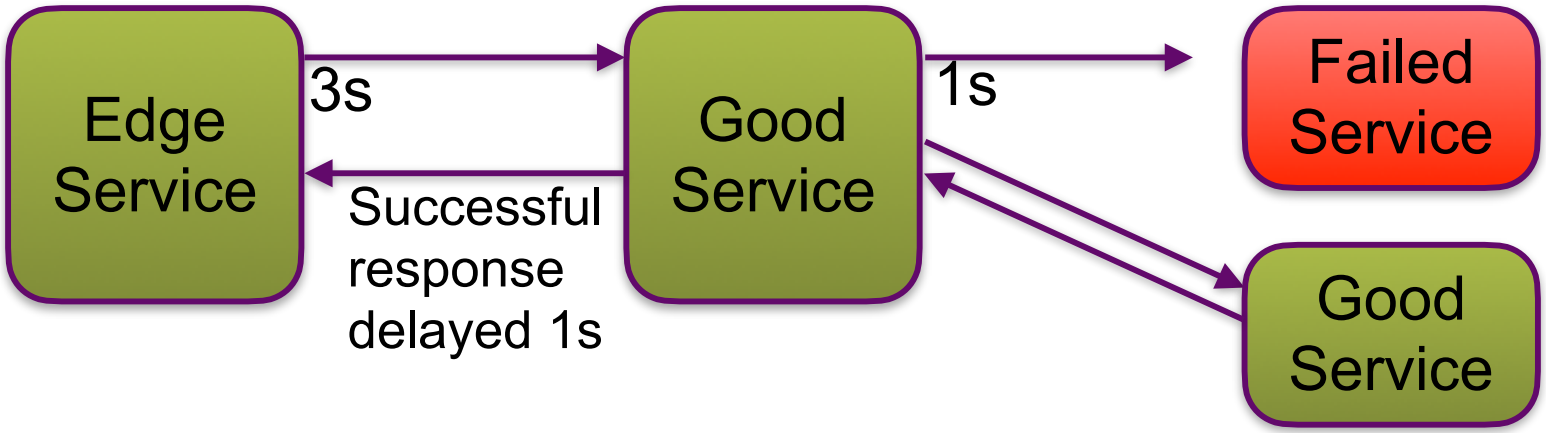


*Upstream timeout must always be longer than total downstream timeout * retries delay*

No unproductive work while fast failing

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

Manage Inconsistency

ACM Paper: "The Network is Reliable"
Distributed systems are inconsistent by nature
Clients are inconsistent with servers
Most caches are inconsistent
Versions are inconsistent
Get over it and
Deal with it

See http://queue.acm.org/detail.cfm?id=2655736

@adrianco
BV
Battery Ventures

Denormalized Data Models

Any non-trivial organization has many databases
Cross references exist, inconsistencies exist

Microservices work best with individual simple stores
Scale, operate, mutate, fail them independently

NoSQL allows flexible schema/object versions

@adrianco
BV
Battery Ventures

Denormalized Data Models

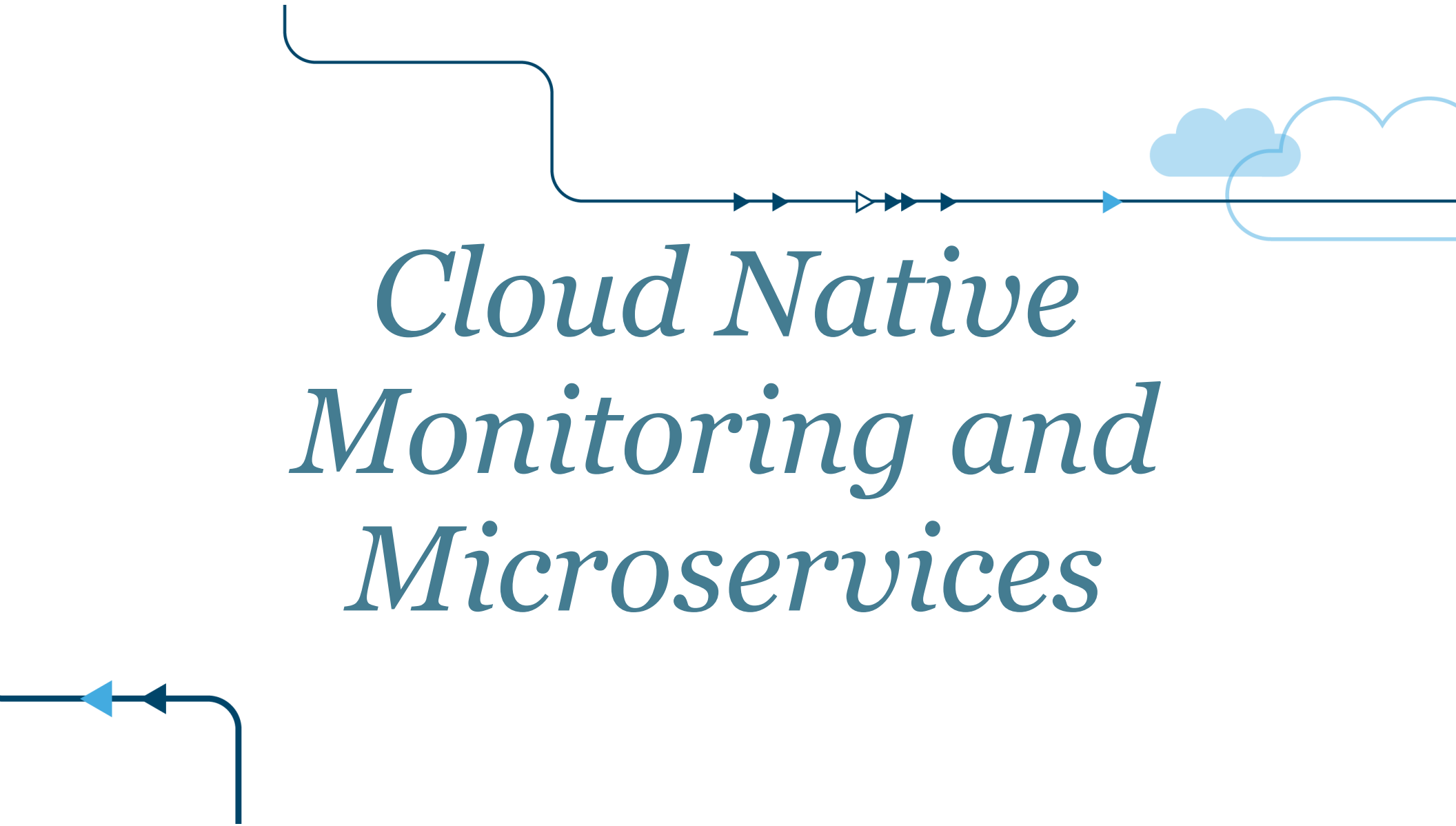
Build custom cross-datasource check/repair processes
Ensure all cross references are up to date
Read these Pat Helland papers

Immutability Changes Everything
http://highscalability.com/blog/2015/1/26/paper-immutability-changes-everything-by-pat-helland.html

Memories, Guesses and Apologies
https://blogs.msdn.microsoft.com/pathelland/2007/05/15/memories-guesses-and-apologies/

Standing on the Distributed Shoulders of Giants
http://queue.acm.org/detail.cfm?id=2953944

@adrianco
BV
Battery Ventures

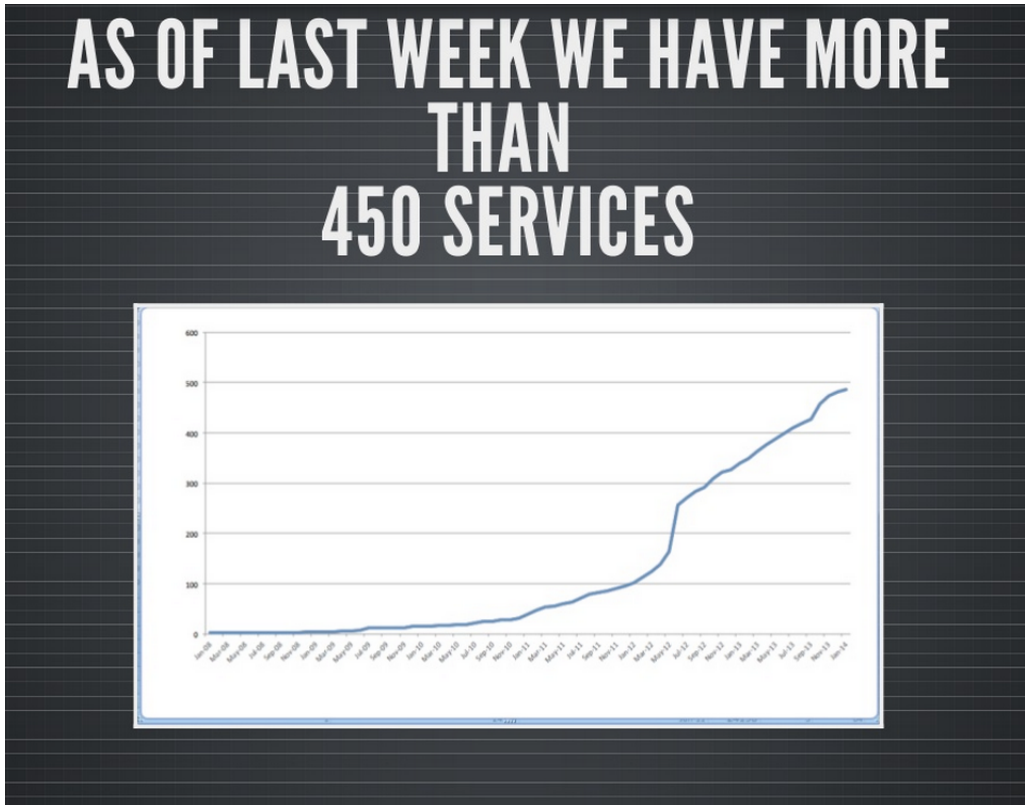


Cloud Native Monitoring and Microservices

Cloud Native Microservices

- *High rate of change*
  - *Code pushes can cause floods of new instances and metrics*
  - *Short baseline for alert threshold analysis – everything looks unusual*
- *Ephemeral Configurations*
  - *Short lifetimes make it hard to aggregate historical views*
  - *Hand tweaked monitoring tools take too much work to keep running*
- *Microservices with complex calling patterns*
  - *End-to-end request flow measurements are very important*
  - *Request flow visualizations get overwhelmed*

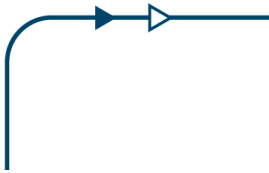
Microservice Based Architectures

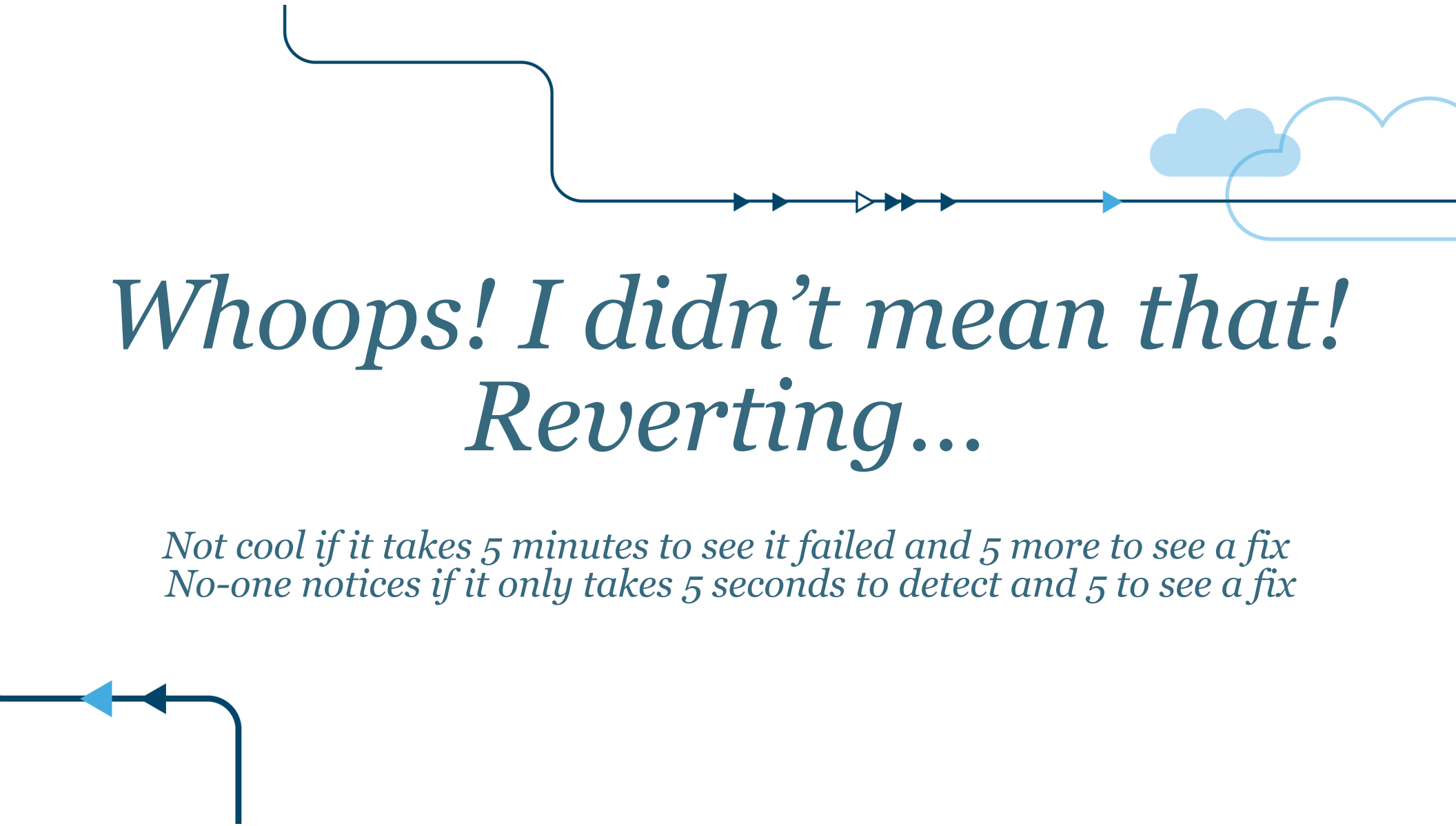


See http://www.slideshare.net/LappleApple/gilt-from-monolith-ruby-app-to-micro-service-scala-service-architecture

Continuous Delivery and DevOps

- *Changes are smaller but more frequent*
- *Individual changes are more likely to be broken*
- *Changes are normally deployed by developers*
- *Feature flags are used to enable new code*
- *Instant detection and rollback matters much more*

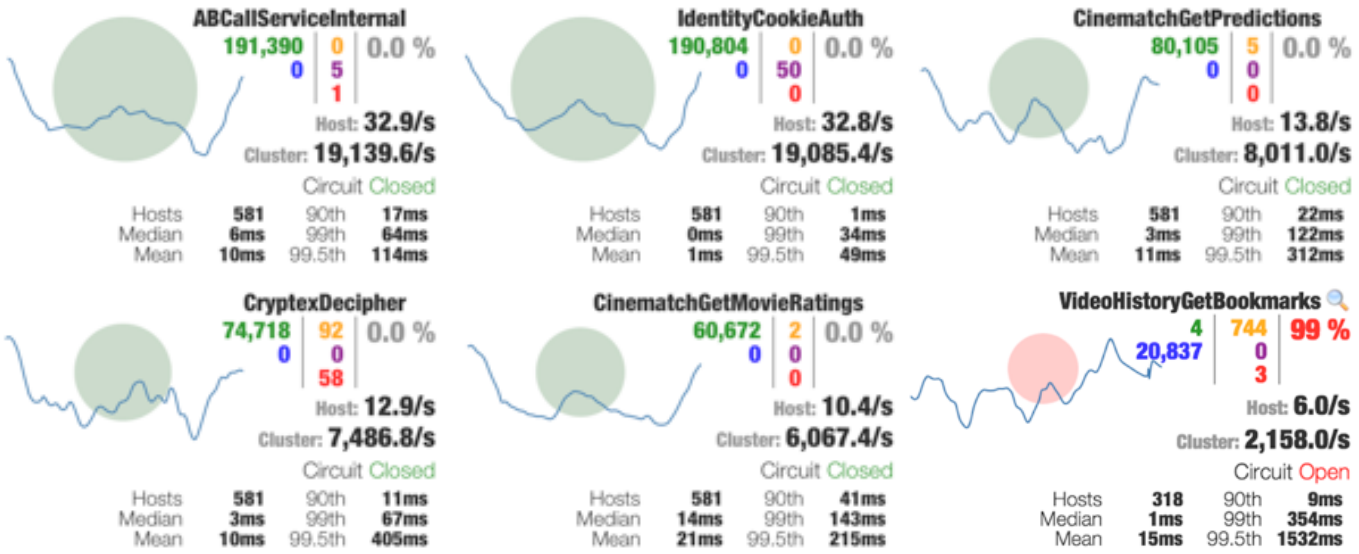




Whoops! I didn't mean that! Reverting…

Not cool if it takes 5 minutes to see it failed and 5 more to see a fix
No-one notices if it only takes 5 seconds to detect and 5 to see a fix

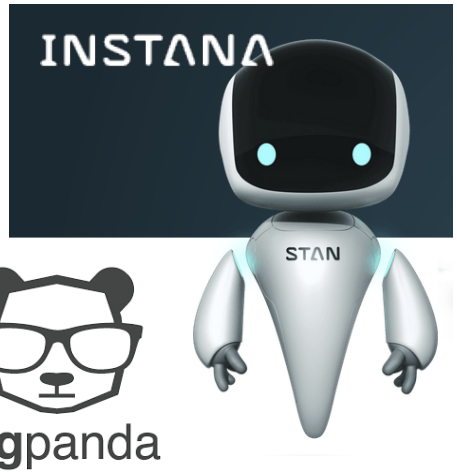
NetflixOSS Hystrix/Turbine Circuit Breaker



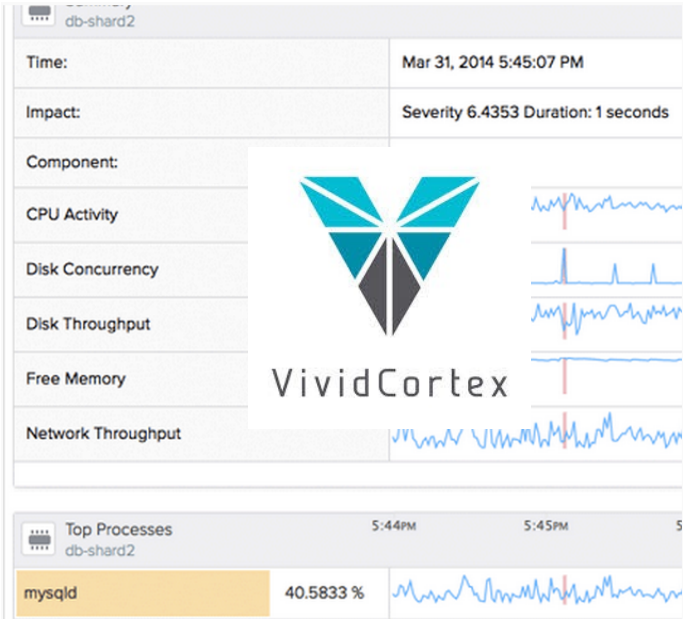
http://techblog.netflix.com/2012/12/hystrix-dashboard-and-turbine.html

Low Latency SaaS Based Monitors

DATADOG



bigpanda



signal fx

wavefront

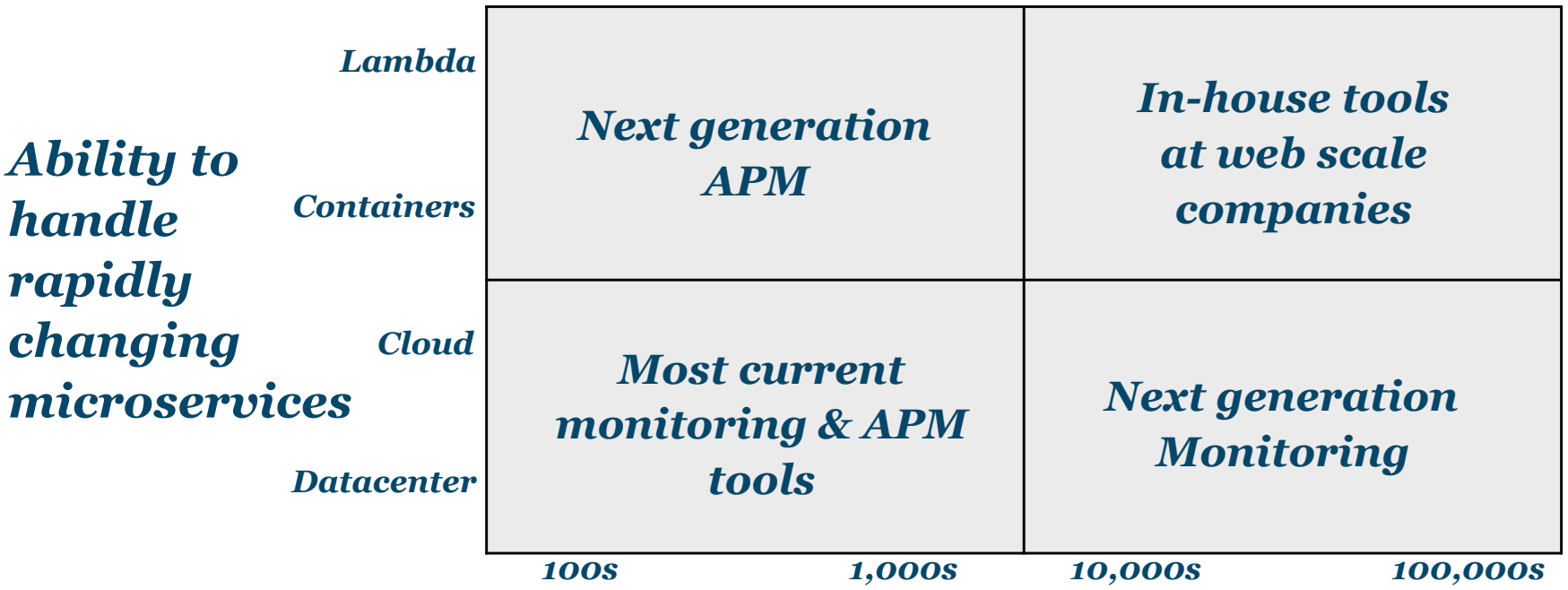
sysdig

Dataloop.IO

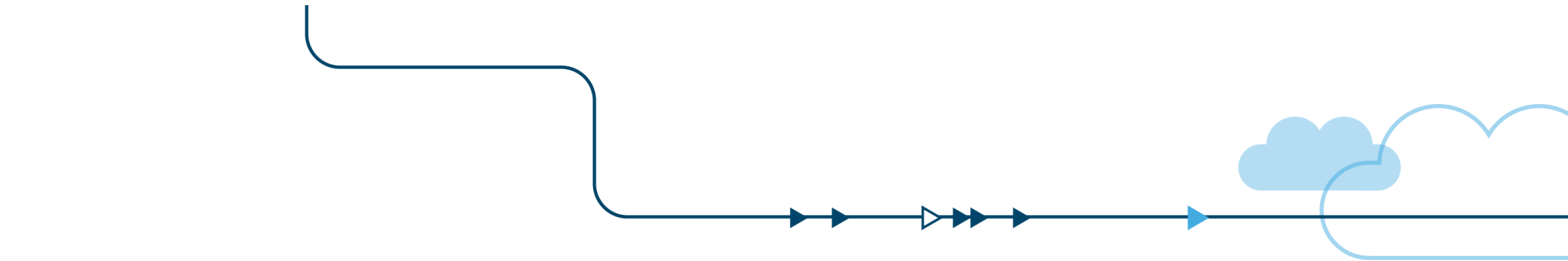
https://www.datadoghq.com/ http://www.instana.com/ www.bigpanda.io www.vividcortex.com signalfx.com wavefront.com sysdig.com dataloop.io

See www.battery.com for a list of portfolio investments

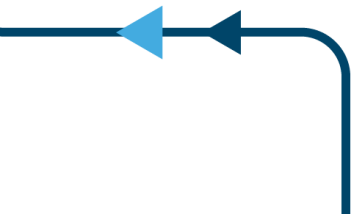
A Tragic Quadrant



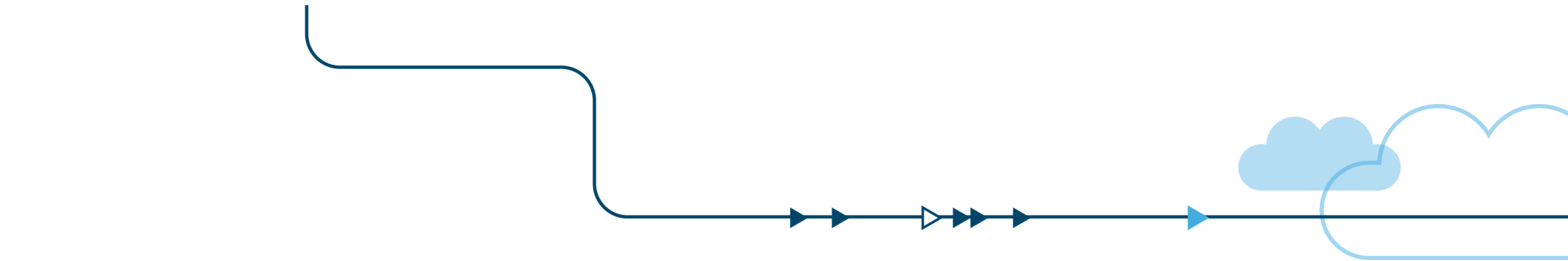
YMMV: Opinionated approximate positioning only



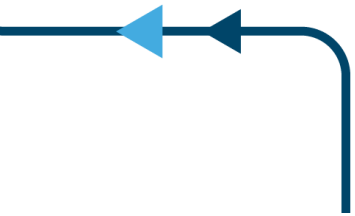
Metric to display latency needs to be less than human attention span (~10s)

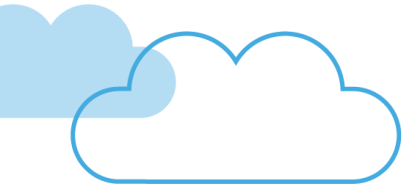



Challenges for
Microservice
Platforms



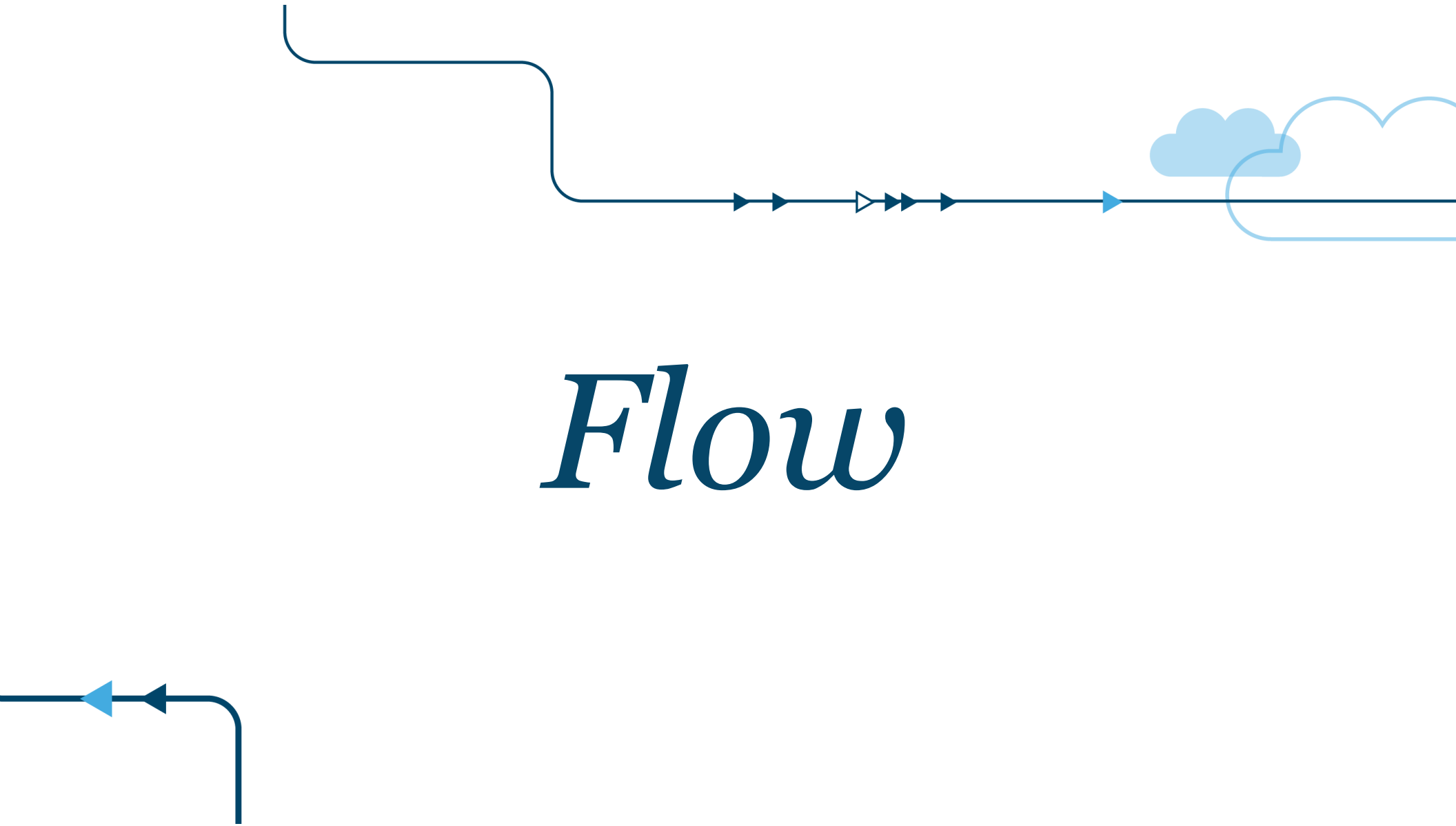
Managing Scale





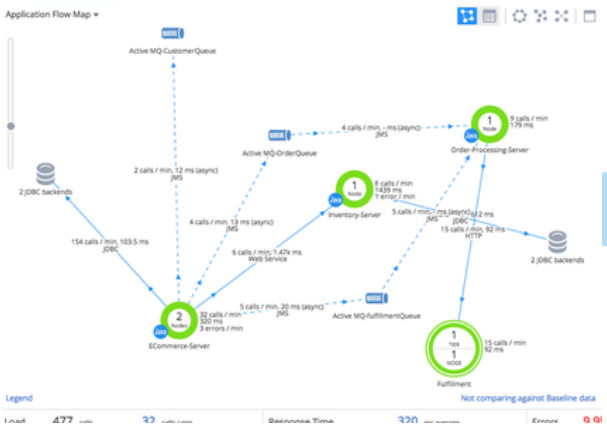
It's much more challenging than just a large number of machines

A Possible Hierarchy	***How Many?***
Continents	*3 to 5*
Regions	*2-4 per Continent*
Zones	*1-5 per Region*
Services	*100's per Zone*
Versions	*Many per Service*
Containers	*1000's per Version*
Instances	*10,000's*

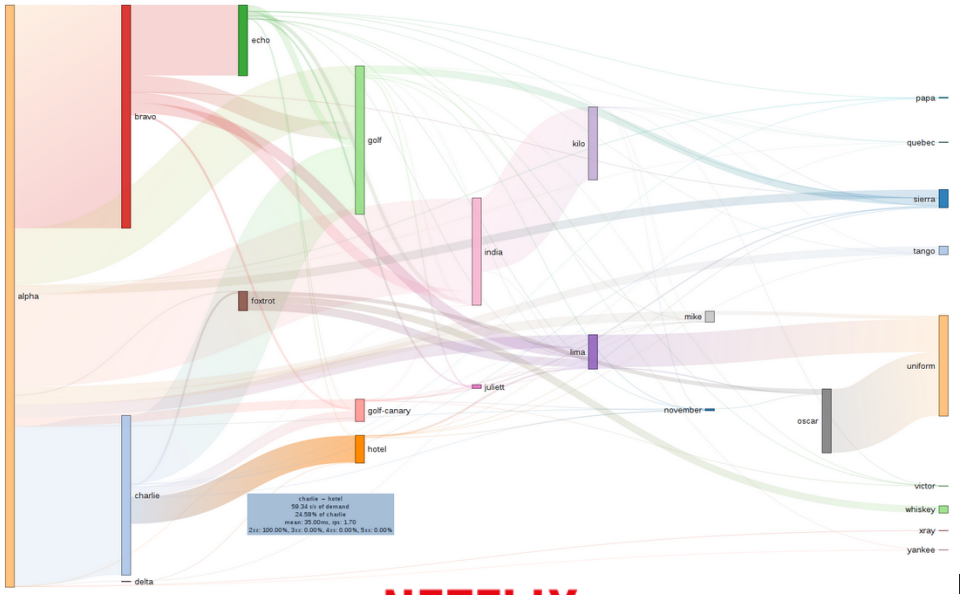


Flow

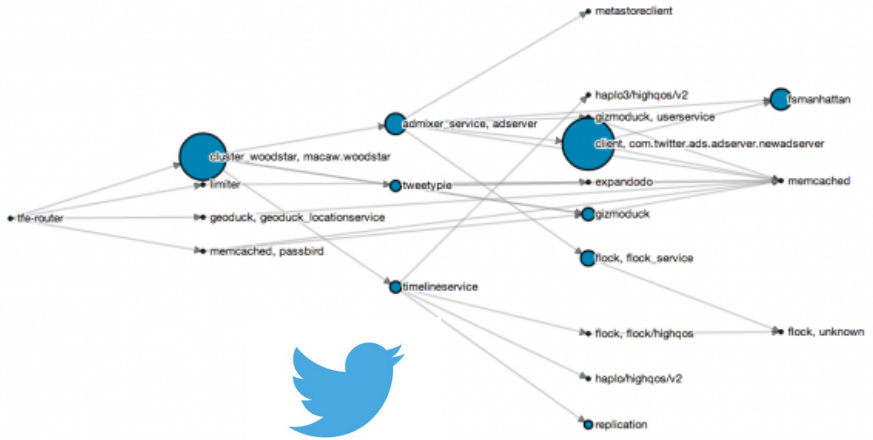
Some tools can show the request flow across a few services



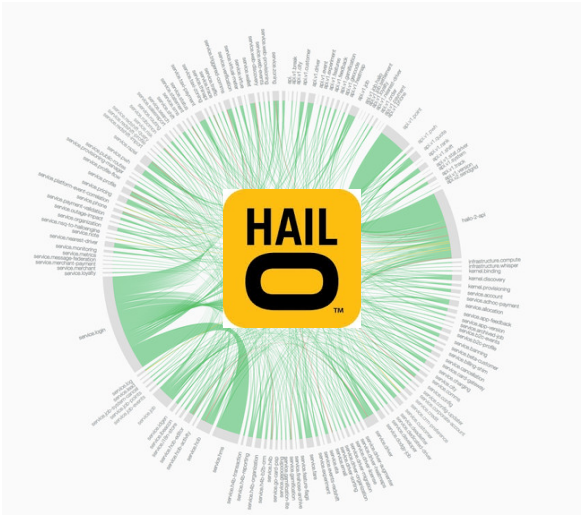
APPDYNAMICS

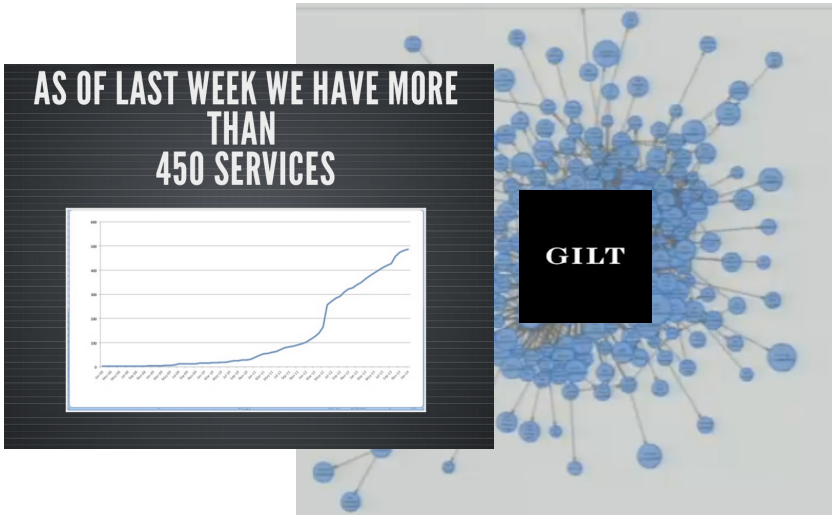


NETFLIX



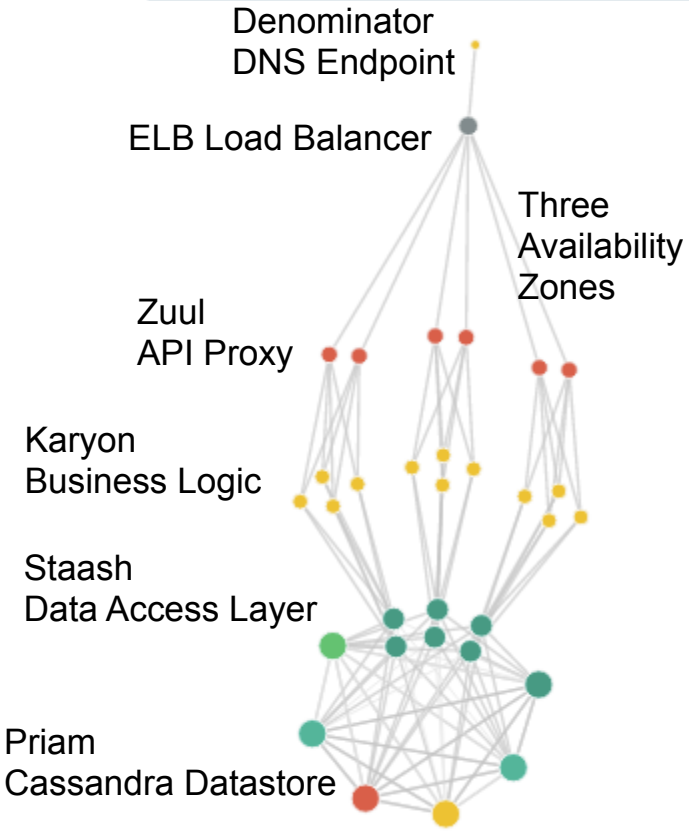
Interesting architectures have a lot of microservices! Flow visualization is a big challenge.





See http://www.slideshare.net/LappleApple/gilt-from-monolith-ruby-app-to-micro-service-scala-service-architecture

Simulated Microservices



Model and visualize microservices
Simulate interesting architectures
Generate large scale configurations
Eventually stress test real tools

Code: github.com/adrianco/spigo
Simulate Protocol Interactions in Go
Visualize with D3
See for yourself: http://simianviz.surge.sh
Follow @simianviz for updates

Spigo Nanoservice Structure

```
func Start(listener chan gotocol.Message) {
        ...
        for {
                select {
                case msg := <-listener:
                        flow.Instrument(msg, name, hist)
                        switch msg.Imposition {
                        case gotocol.Hello:      // get named by parent
                                ...
                        case gotocol.NameDrop:   // someone new to talk to
                                ...
                        case gotocol.Put: // upstream request handler
                                ...
                                outmsg := gotocol.Message{gotocol.Replicate, listener, time.Now(),
                                        msg.Ctx.NewParent(), msg.Intention}
                                flow.AnnotateSend(outmsg, name)
                                outmsg.GoSend(replicas)
                        }
                case <-eurekaTicker.C:           // poll the service re
                        ...
                }
        }
}
```

Instrument incoming requests

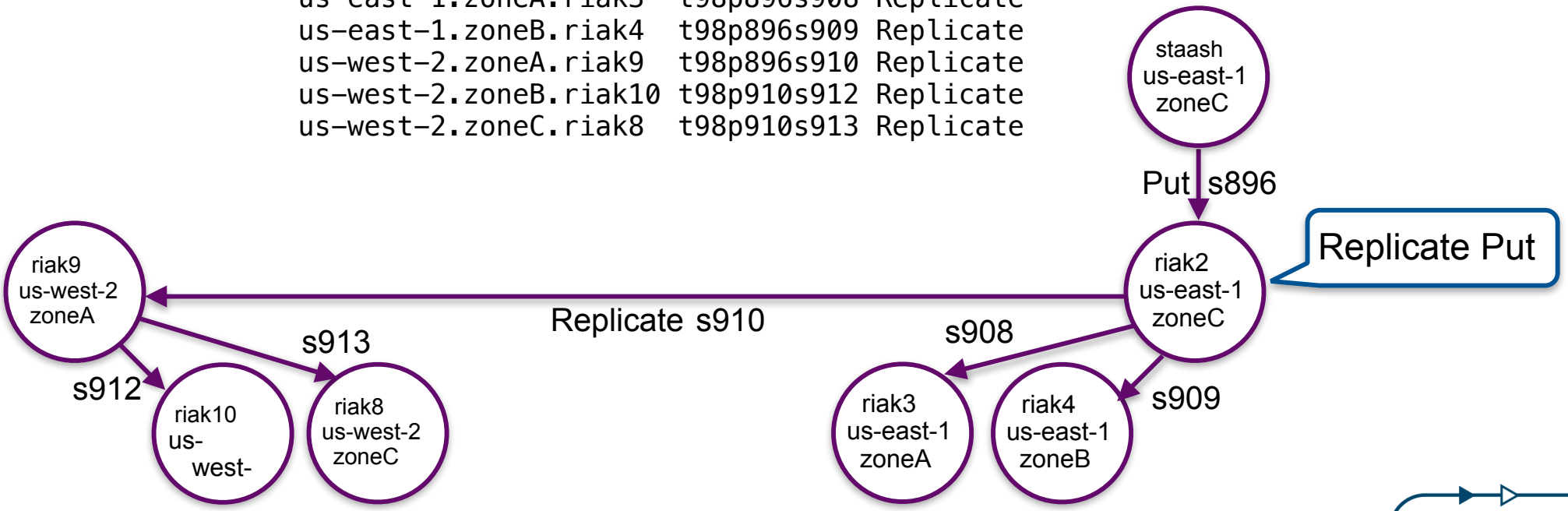
update trace context

Instrument outgoing requests

Skeleton code for replicating a Put message

Flow Trace Records

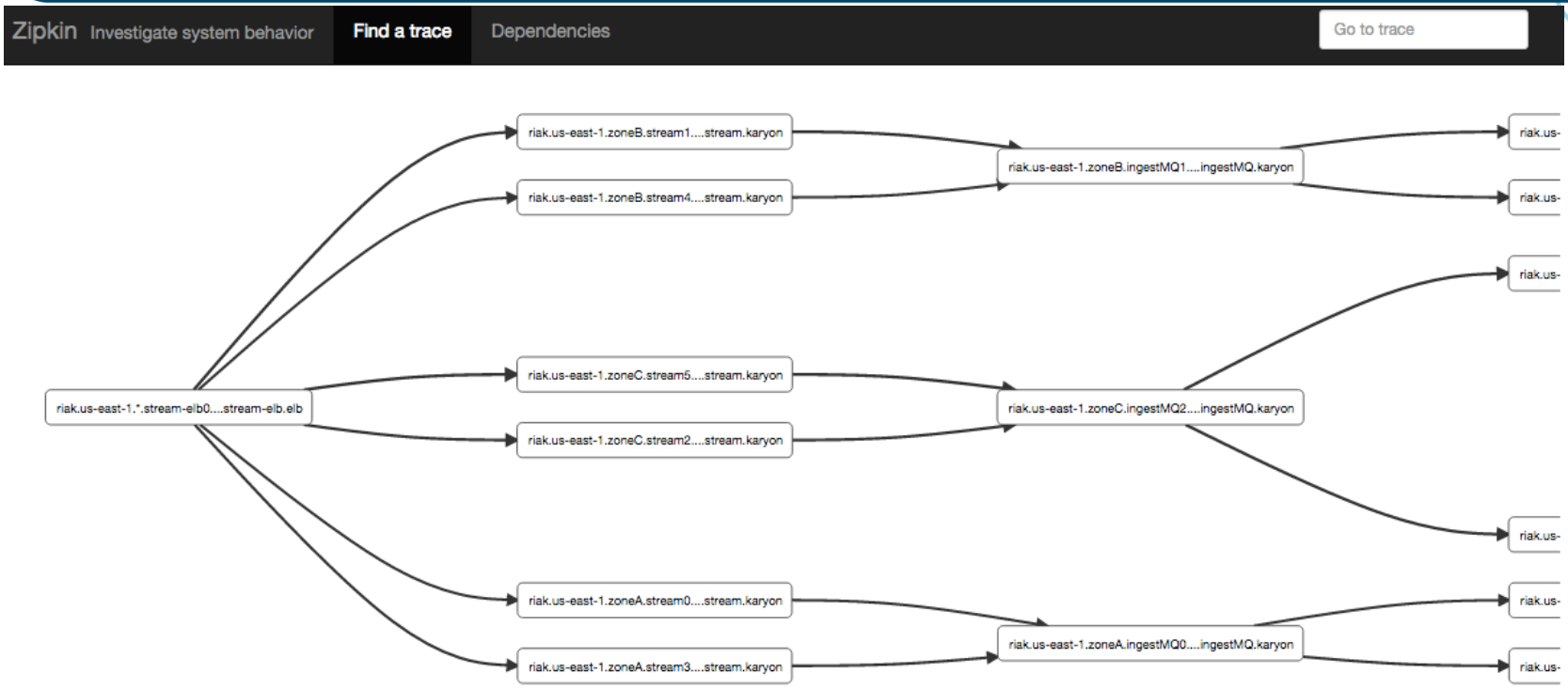
```
us-east-1.zoneC.riak2  t98p895s896 Put
us-east-1.zoneA.riak3  t98p896s908 Replicate
us-east-1.zoneB.riak4  t98p896s909 Replicate
us-west-2.zoneA.riak9  t98p896s910 Replicate
us-west-2.zoneB.riak10 t98p910s912 Replicate
us-west-2.zoneC.riak8  t98p910s913 Replicate
```



Open Zipkin

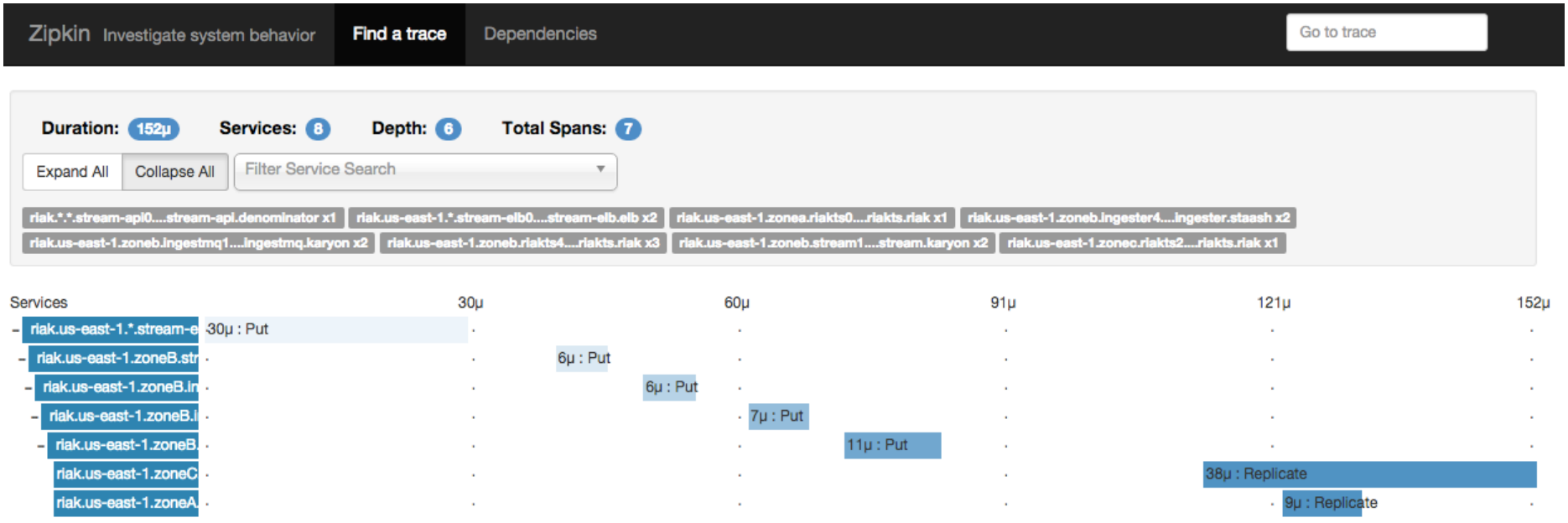
A common format for trace annotations
A Java tool for visualizing traces
Standardization effort to fold in other formats
Driven by Adrian Cole (currently at Pivotal)
Extended to load Spigo generated trace files

Zipkin Trace Dependencies

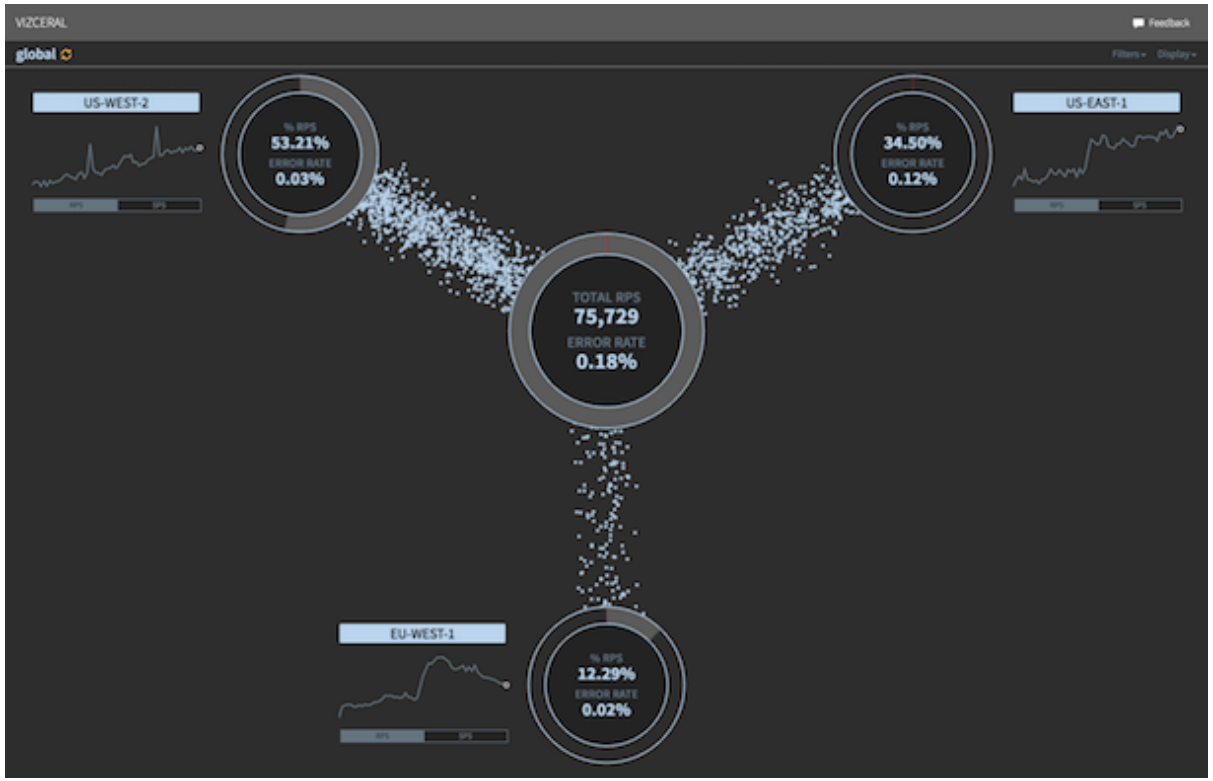


Zipkin Trace Dependencies

Trace for one Spigo Flow



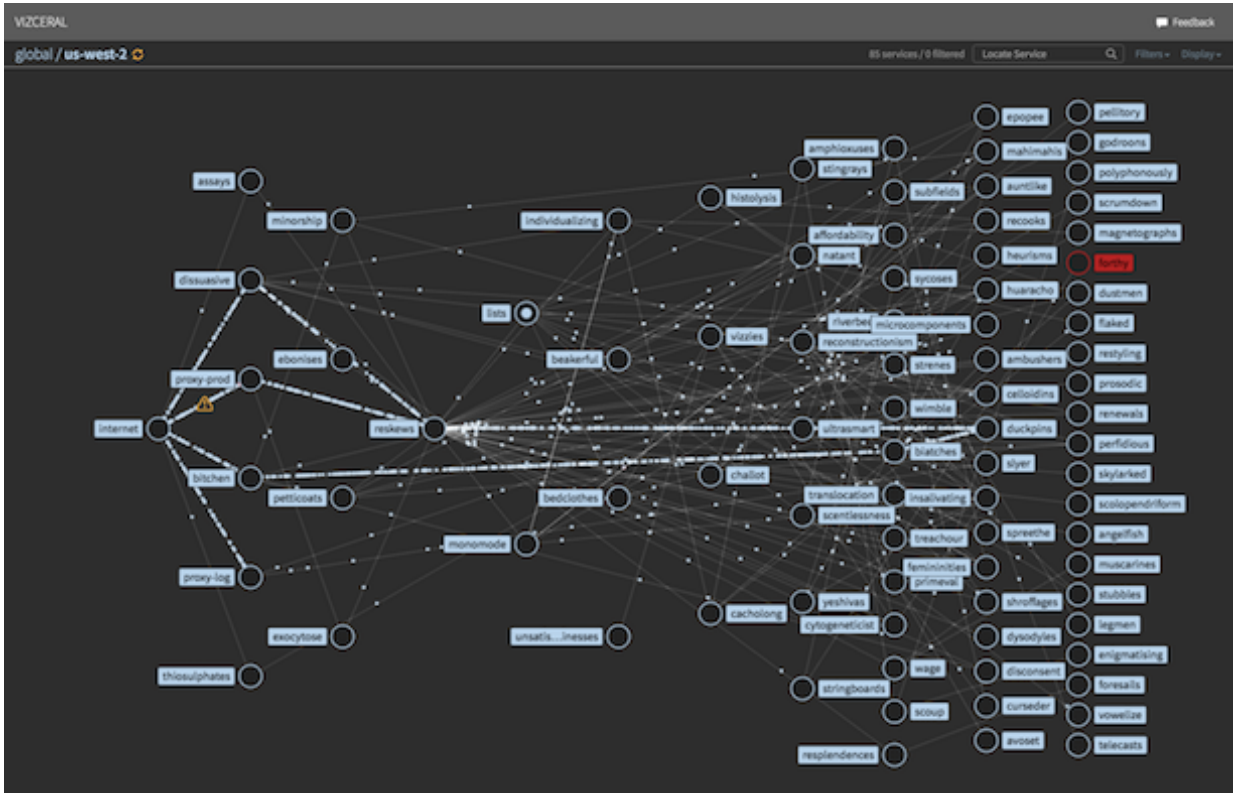
Global Visualization



NetflixOSS
Vizceral

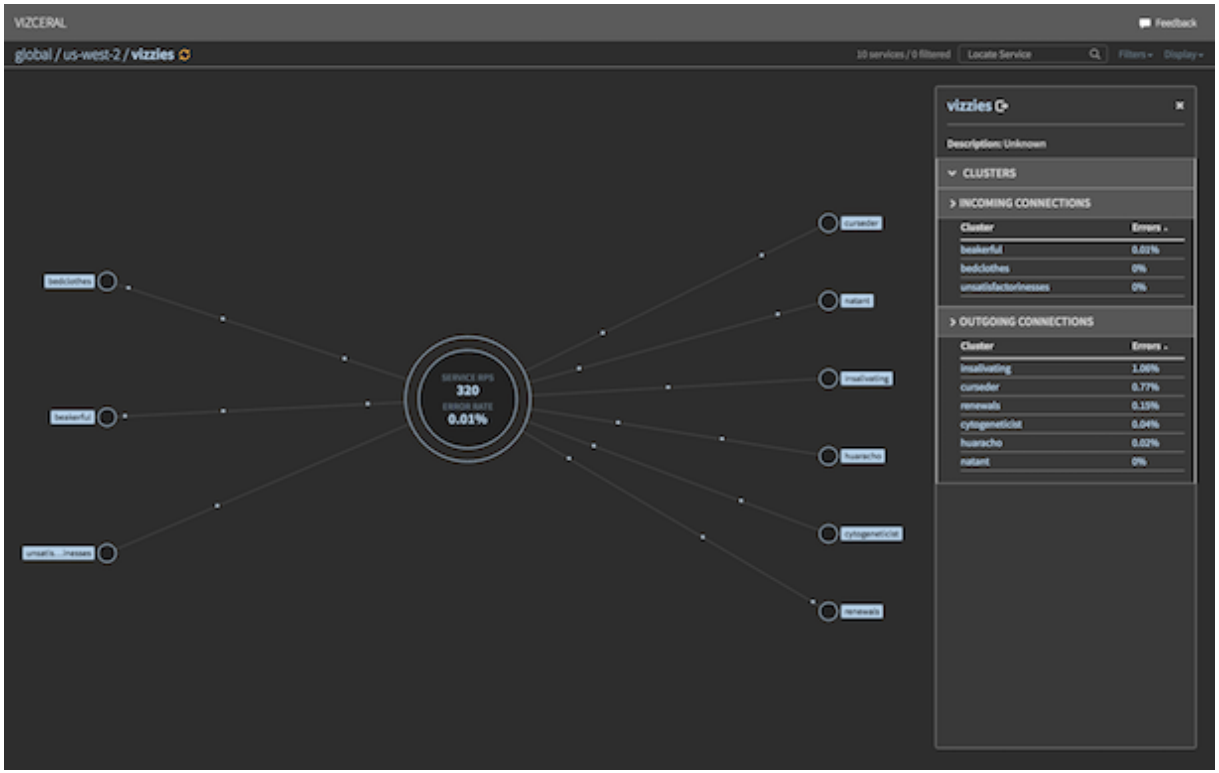
https://github.com/Netflix/vizceral
https://github.com/adrianco/go-vizceral

Regional Visualization



NetflixOSS
Vizceral

Service Context Visualization



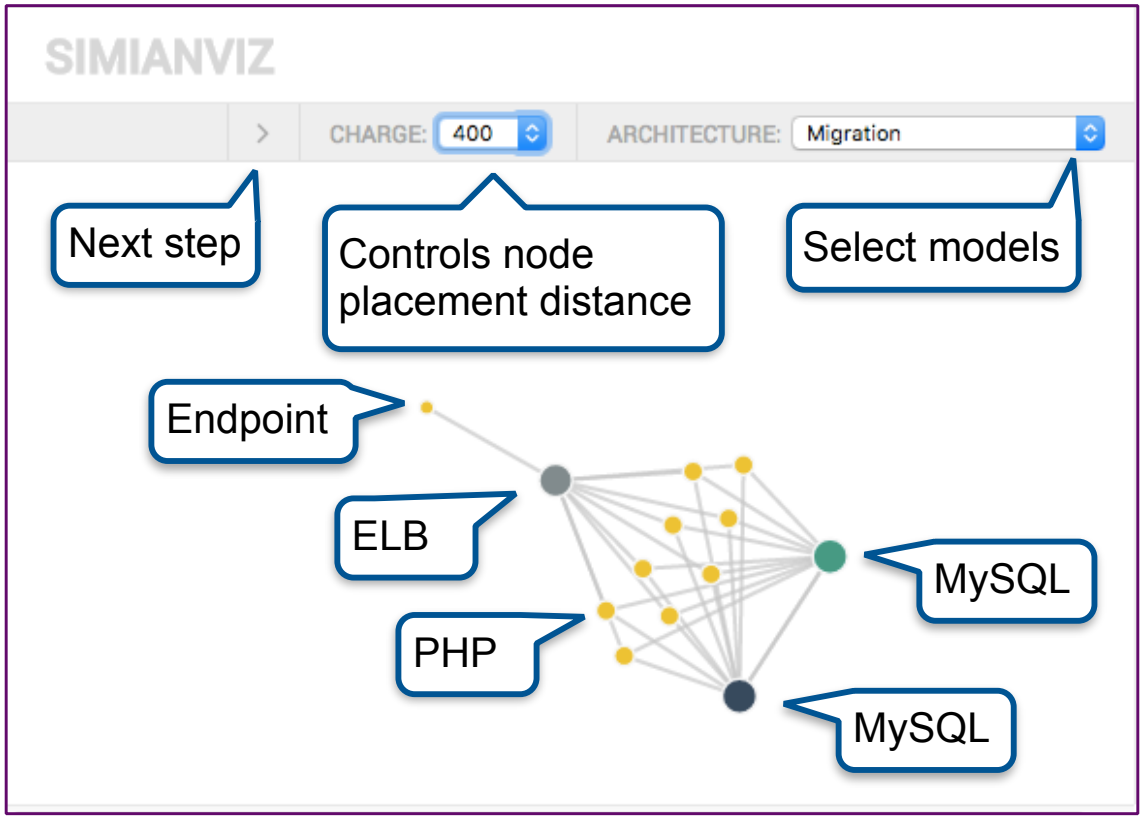
NetflixOSS
Vizceral

Simple Architecture Principles

Symmetry
Invariants
Stable assertions
No special cases

Migrating to Microservices

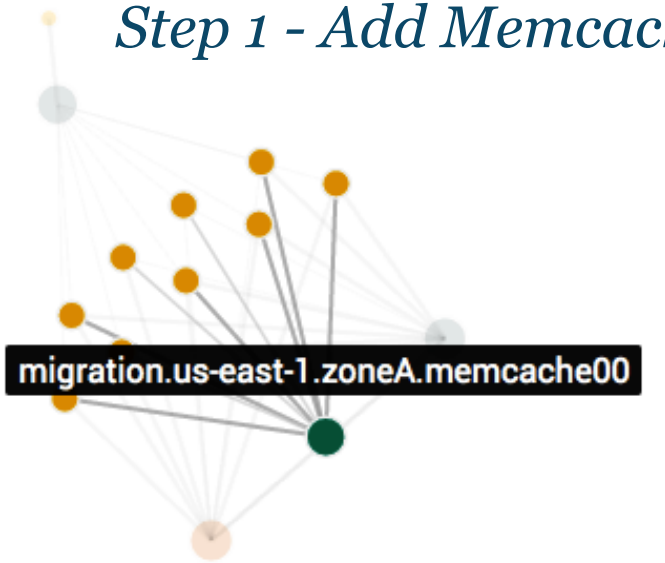
See for yourself: http://simianviz.surge.sh/migration



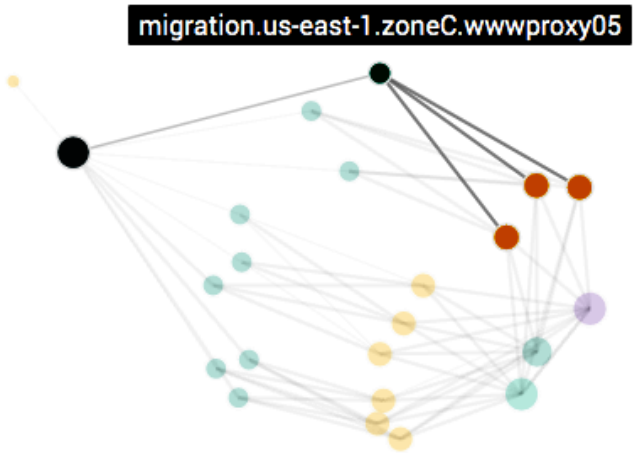
Migrating to Microservices

See for yourself: http://simianviz.surge.sh/migration

Step 1 - Add Memcache



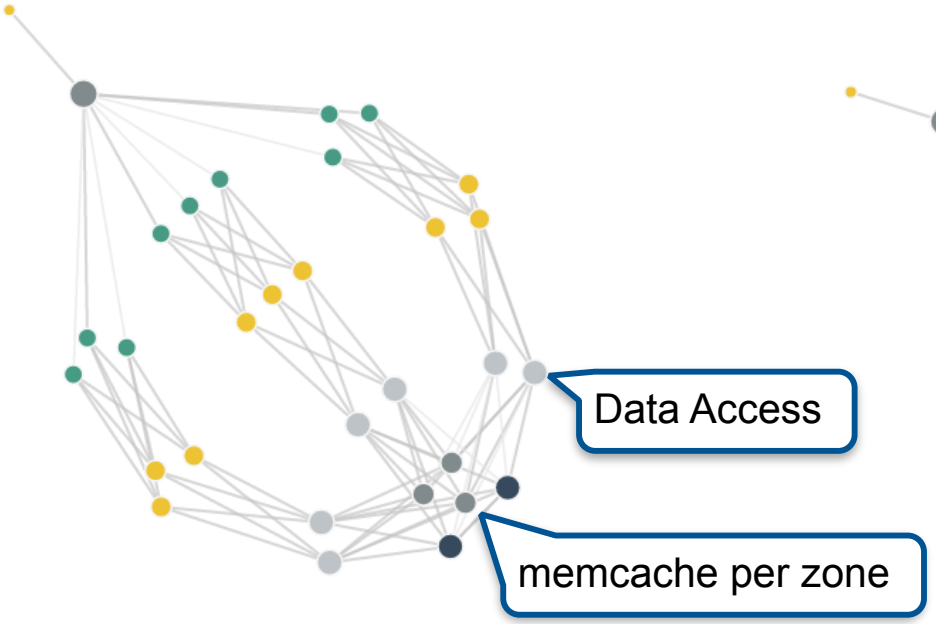
Step 2 - Add Web Proxy Service



Migrating to Microservices

See for yourself: http://simianviz.surge.sh/migration

Step 3 - Add Data Access Layer

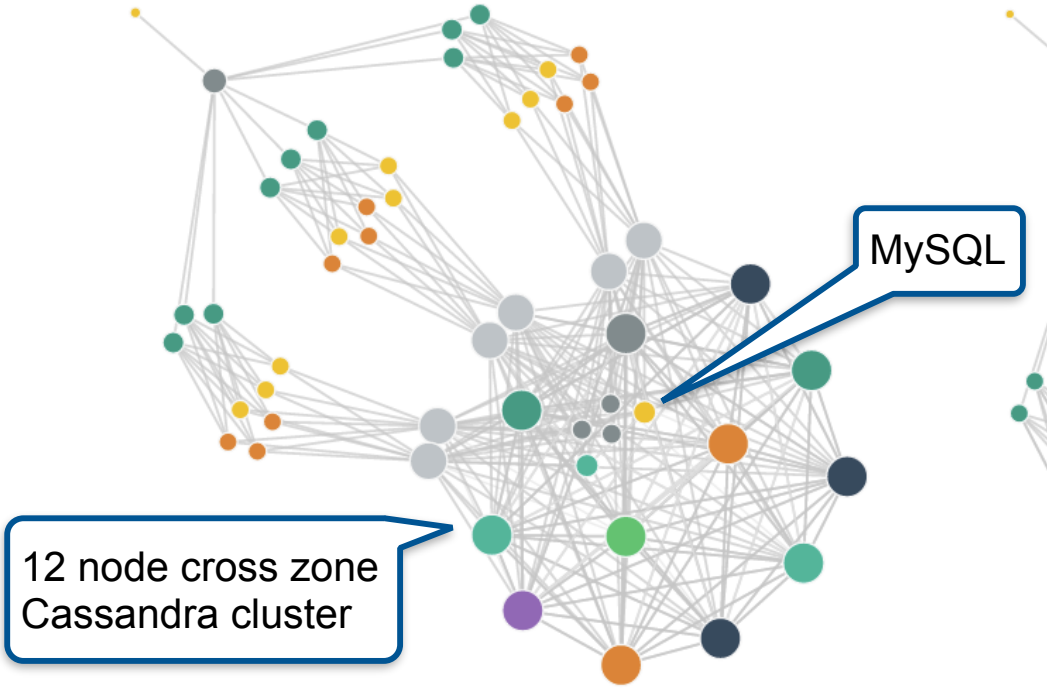


Step 4 - Add Microservices

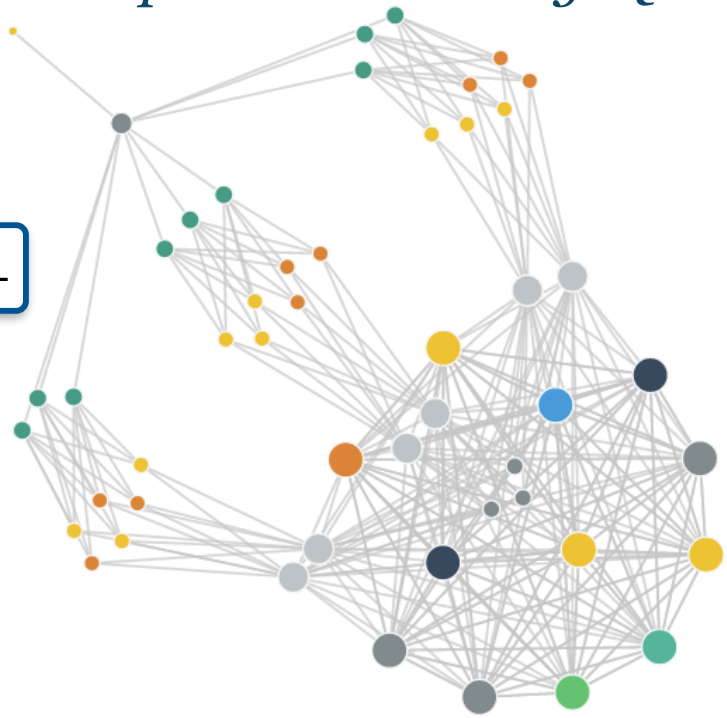
Migrating to Microservices

See for yourself: http://simianviz.surge.sh/migration

Step 5 - Add Cassandra



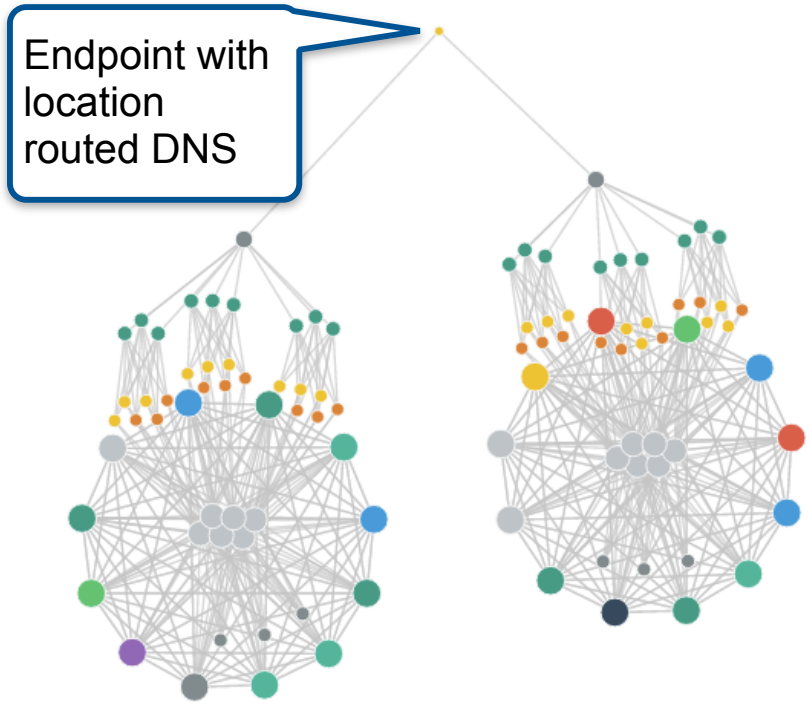
Step 6 - Remove MySQL



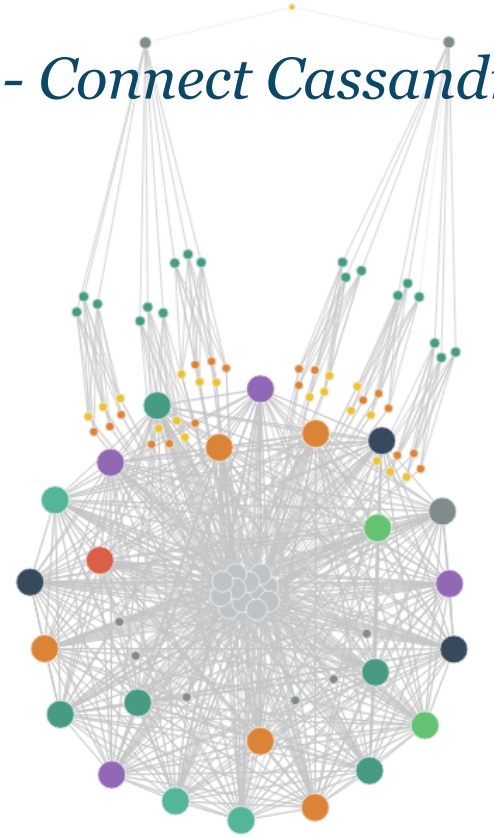
Migrating to Microservices

See for yourself: http://simianviz.surge.sh/migration

Step 7 - Add Second Region



Step 8 - Connect Cassandra Regions

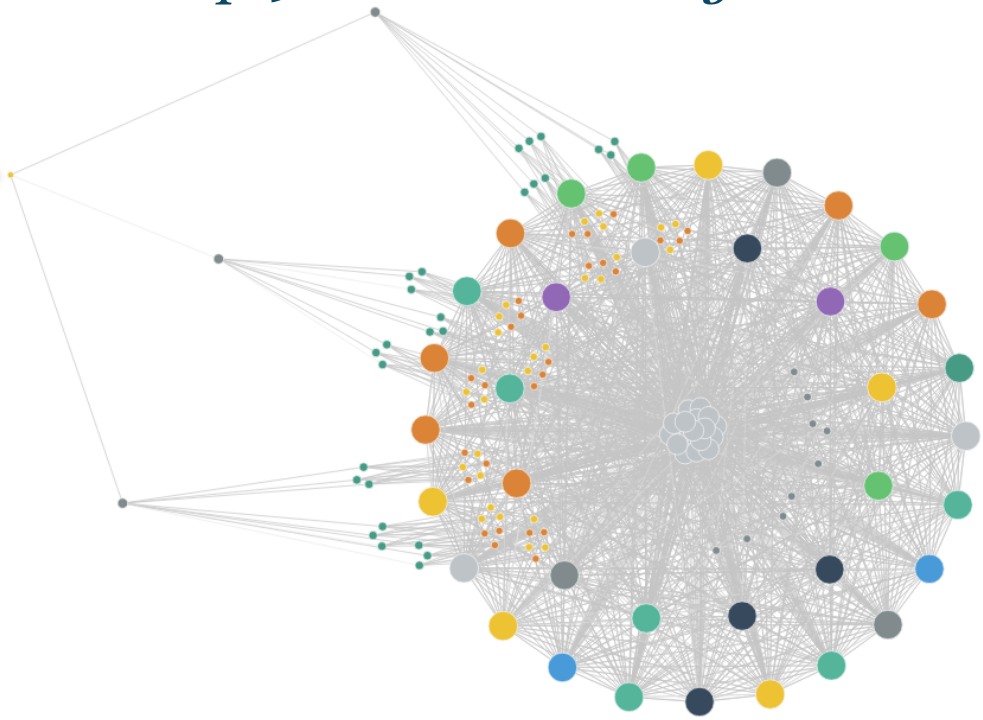


Migrating to Microservices

See for yourself: http://simianviz.surge.sh/migration

Step 9 - Add Third Region

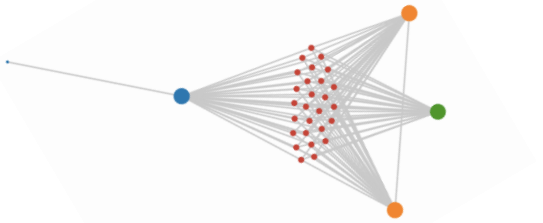
Endpoint with location routed DNS



Definition of an architecture

Header includes chaos monkey victim

See for yourself: http://simianviz.surge.sh/lamp



```
{
    "arch": "lamp",
    "description":"Simple LAMP stack",
    "version": "arch-0.0",
    "victim": "webserver",
    "services": [
        { "name": "rds-mysql",     "package": "store",       "count": 2,  "regions": 1, "dependencies": [] },
        { "name": "memcache",      "package": "store",       "count": 1,  "regions": 1, "dependencies": [] },
        { "name": "webserver",     "package": "monolith",    "count": 18, "regions": 1, "dependencies": ["memcache", "rds-mysql"] },
        { "name": "webserver-elb", "package": "elb",         "count": 0,  "regions": 1, "dependencies": ["webserver"] },
        { "name": "www",           "package": "denominator", "count": 0,  "regions": 0, "dependencies": ["webserver-elb"] }
    ]
}
```

New tier name

Tier package

Node count

0 = non Regional

List of tier dependencies

Running Spigo

-a architecture lamp
-j graph json/lamp.json
-d run for 2 seconds

```
$ ./spigo -a lamp -j -d 2
2016/01/26 23:04:05 Loading architecture from json_arch/lamp_arch.json
2016/01/26 23:04:05 lamp.edda: starting
2016/01/26 23:04:05 Architecture: lamp Simple LAMP stack
2016/01/26 23:04:05 architecture: scaling to 100%
2016/01/26 23:04:05 lamp.us-east-1.zoneB.eureka01....eureka.eureka: starting
2016/01/26 23:04:05 lamp.us-east-1.zoneA.eureka00....eureka.eureka: starting
2016/01/26 23:04:05 lamp.us-east-1.zoneC.eureka02....eureka.eureka: starting
2016/01/26 23:04:05 Starting: {rds-mysql     store 1 2 []}
2016/01/26 23:04:05 Starting: {memcache     store 1 1 []}
2016/01/26 23:04:05 Starting: {webserver     monolith 1 18 [memcache rds-mysql]}
2016/01/26 23:04:05 Starting: {webserver-elb     elb 1 0 [webserver]}
2016/01/26 23:04:05 Starting: {www     denominator 0 0 [webserver-elb]}
2016/01/26 23:04:05 lamp.*.*.www00....www.denominator activity rate  10ms
2016/01/26 23:04:06 chaosmonkey delete: lamp.us-east-1.zoneC.webserver02....webserver.monolith
2016/01/26 23:04:07 asgard: Shutdown
2016/01/26 23:04:07 lamp.us-east-1.zoneB.eureka01....eureka.eureka: closing
2016/01/26 23:04:07 lamp.us-east-1.zoneA.eureka00....eureka.eureka: closing
2016/01/26 23:04:07 lamp.us-east-1.zoneC.eureka02....eureka.eureka: closing
2016/01/26 23:04:07 spigo: complete
2016/01/26 23:04:07 lamp.edda: closing
```

Riak IoT Architecture

```
{
    "arch": "riak",
    "description":"Riak IoT ingestion example for the RICON 2015 presentation",
    "version": "arch-0.0",
    "victim": "",
    "services": [
        { "name": "riakTS",      "package": "riak",          "count": 6, "regions": 1, "dependencies": ["riakTS", "eureka"]},
        { "name": "ingester",    "package": "staash",        "count": 6, "regions": 1, "dependencies": ["riakTS"]},
        { "name": "ingestMQ",    "package": "karyon",        "count": 3, "regions": 1, "dependencies": ["ingester"]},
        { "name": "riakKV",      "package": "riak",          "count": 3, "regions": 1, "dependencies": ["riakKV"]},
        { "name": "enricher",    "package": "staash",        "count": 6, "regions": 1, "dependencies": ["riakKV", "ingestMQ"]},
        { "name": "enrichMQ",    "package": "karyon",        "count": 3, "regions": 1, "dependencies": ["enricher"]},
        { "name": "analytics",   "package": "karyon",        "count": 6, "regions": 1, "dependencies": ["ingester"]},
        { "name": "analytics-elb", "package": "elb",         "count": 0, "regions": 1, "dependencies": ["analytics"]},
        { "name": "analytics-api", "package": "denominator", "count": 0, "regions": 0, "dependencies": ["analytics-elb"]},
        { "name": "normalization", "package": "karyon",      "count": 6, "regions": 1, "dependencies": ["enrichMQ"]},
        { "name": "iot-elb",     "package": "elb",           "count": 0, "regions": 1, "dependencies": ["normalization"]},
        { "name": "iot-api",     "package": "denominator",   "count": 0, "regions": 0, "dependencies": ["iot-elb"]},
        { "name": "stream",      "package": "karyon",        "count": 6, "regions": 1, "dependencies": ["ingestMQ"]},
        { "name": "stream-elb", "package": "elb",            "count": 0, "regions": 1, "dependencies": ["stream"]},
        { "name": "stream-api", "package": "denominator",    "count": 0, "regions": 0, "dependencies": ["stream-elb"]}
    ]
}
```

New tier name

Tier package

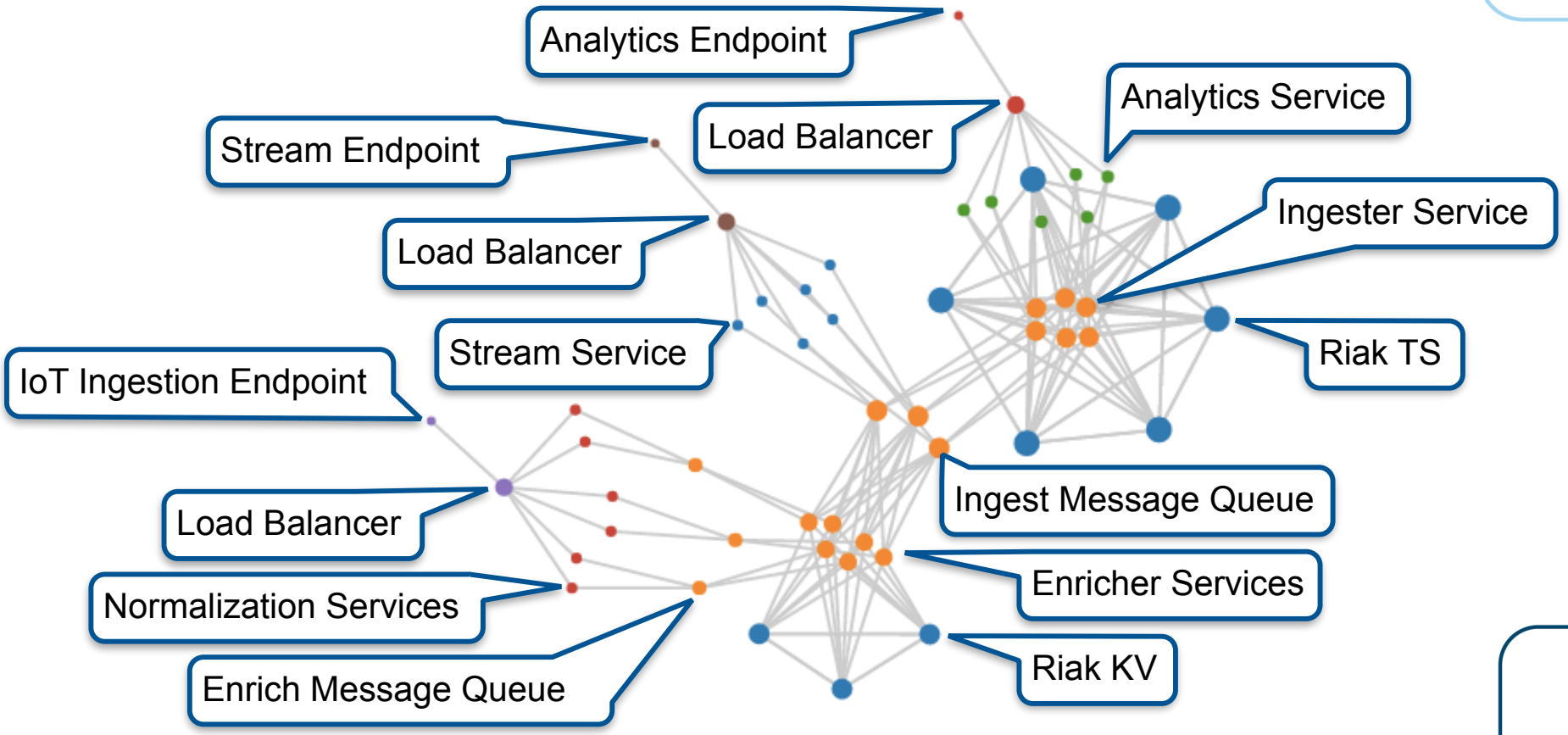
Node count

0 = non Regional

List of tier dependencies

Single Region Riak IoT

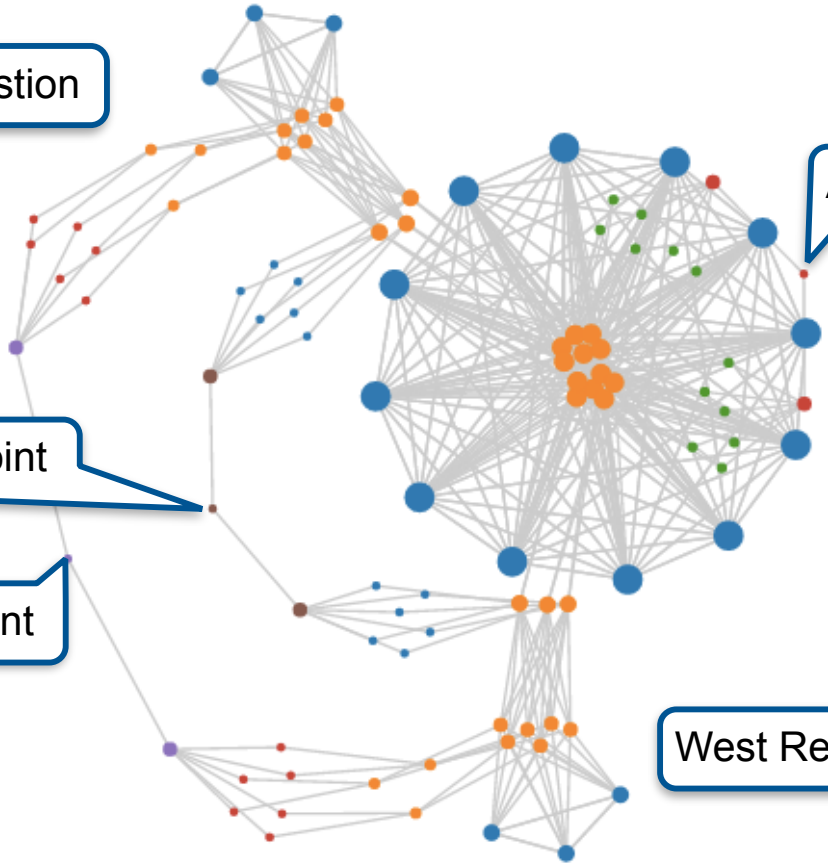
See for yourself: http://simianviz.surge.sh/riak



Two Region Riak IoT

See for yourself: http://simianviz.surge.sh/riak

What's the response time of the stream endpoint?



East Region Ingestion

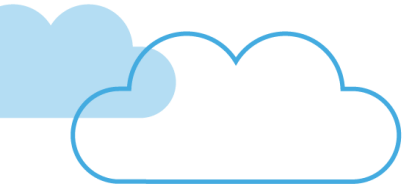
Analytics Endpoint

Multi Region TS Analytics

Stream Endpoint

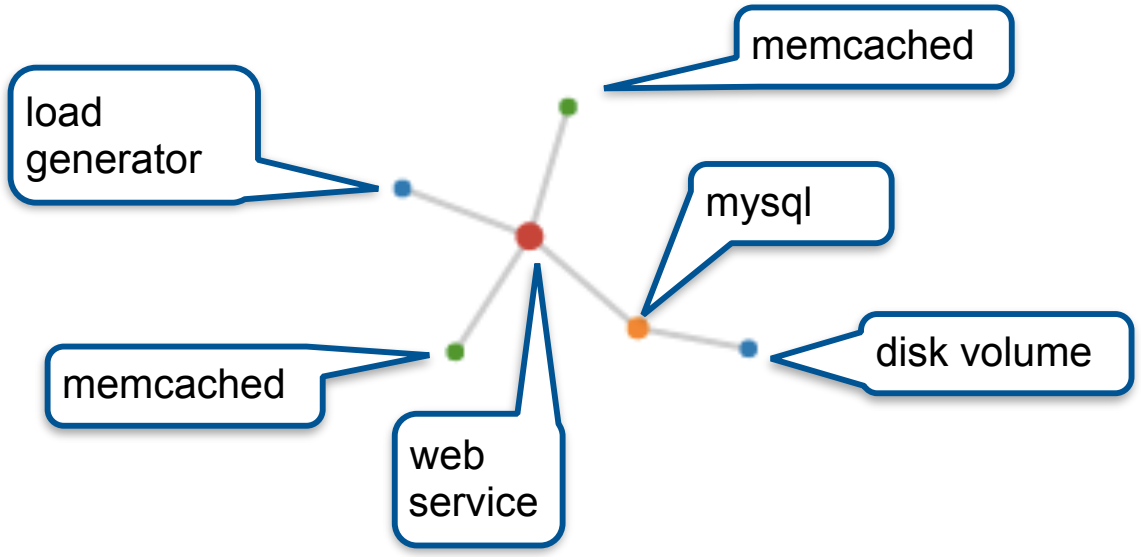
IoT Ingestion Endpoint

West Region Ingestion

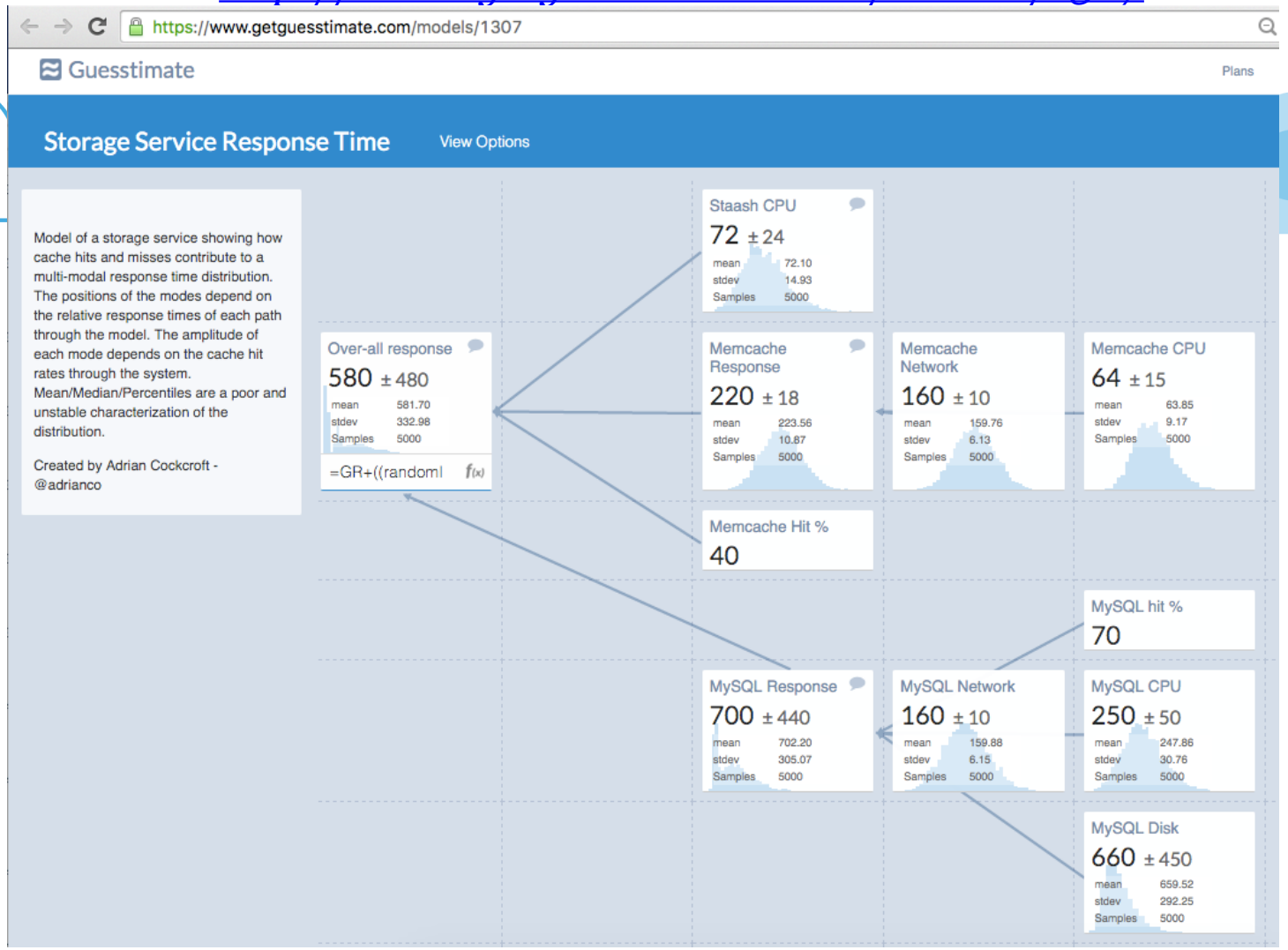



Response Times

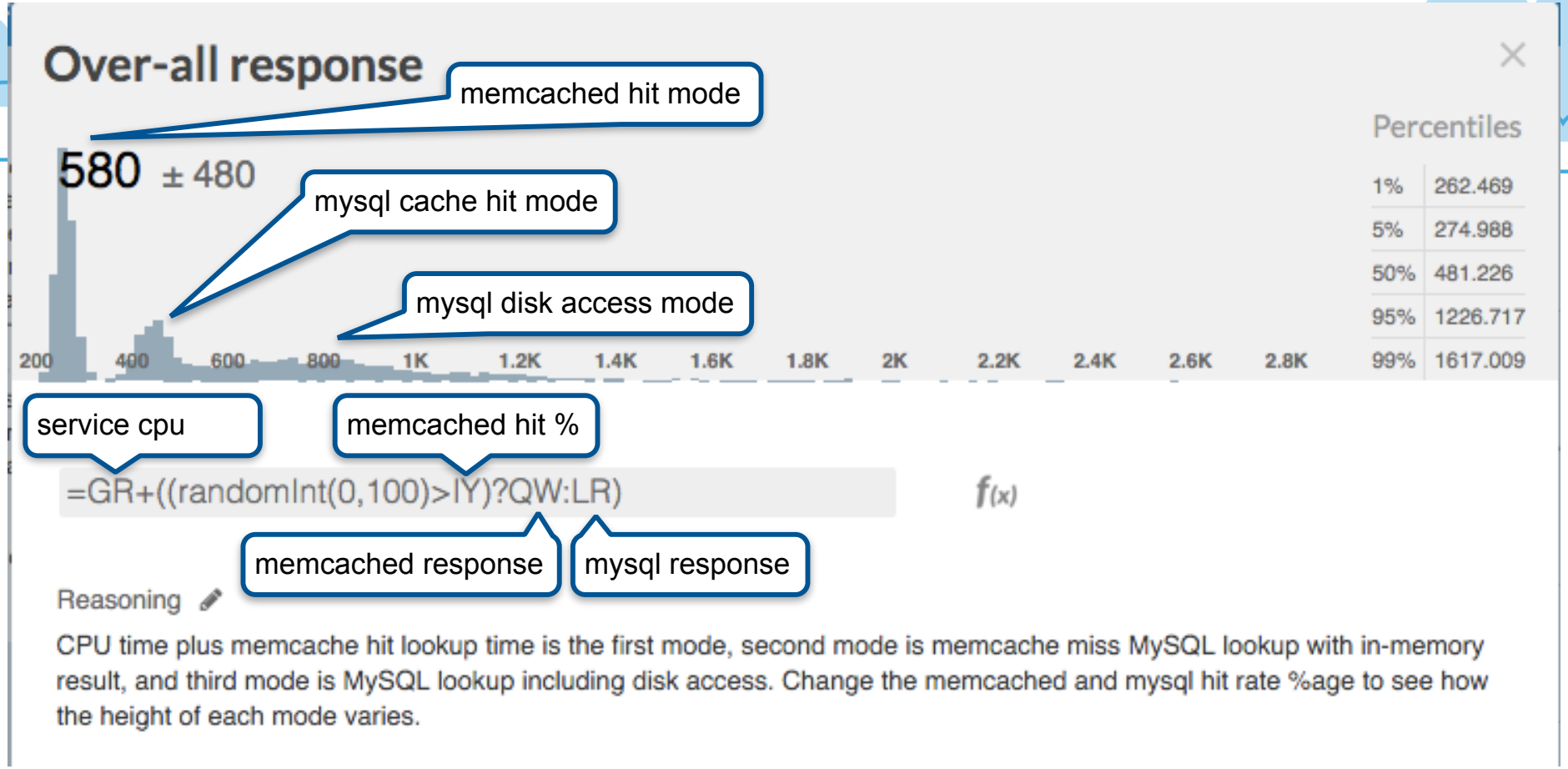
What's the response time distribution of a very simple storage backed web service?



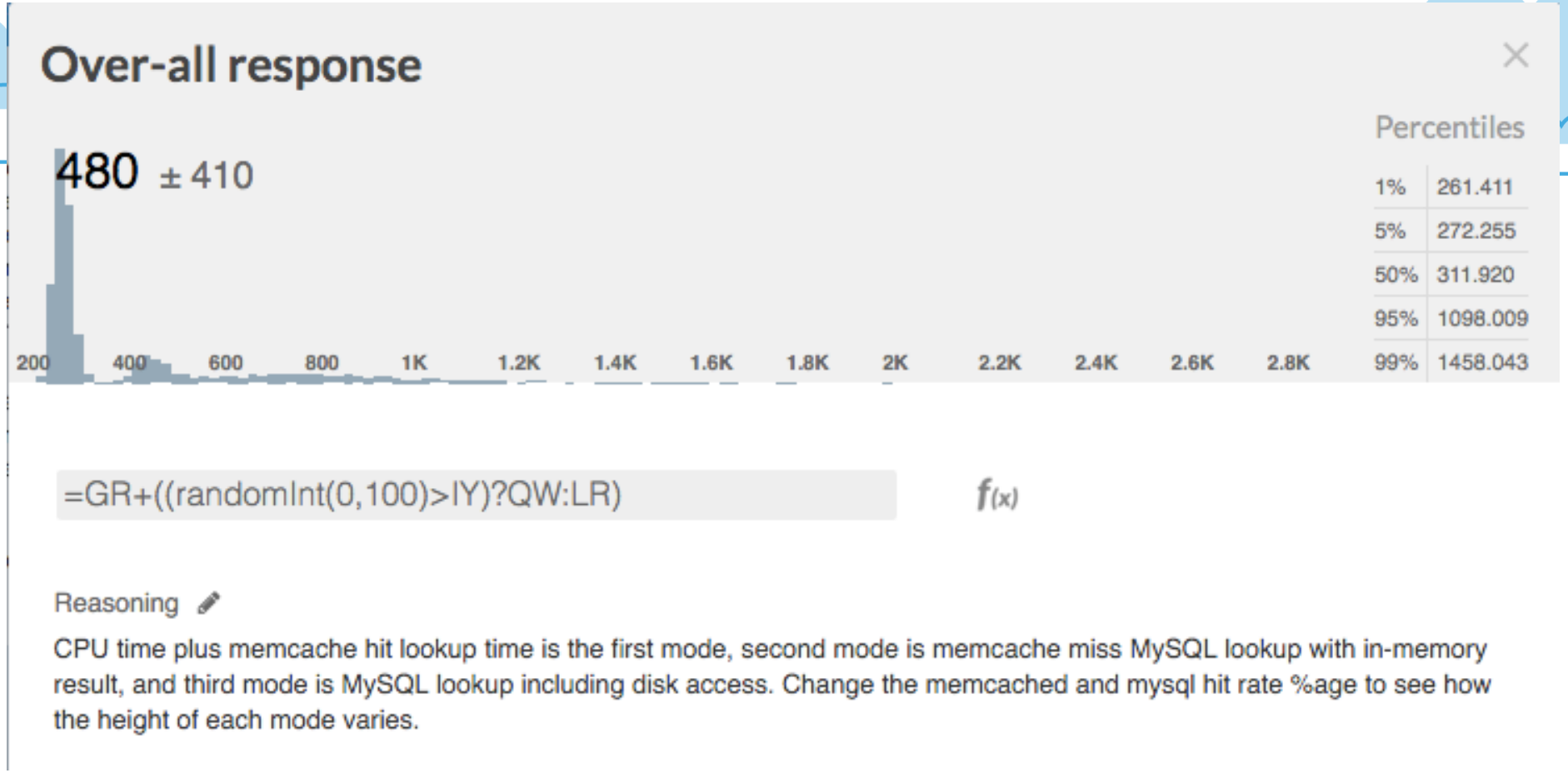
See http://www.getguesstimate.com/models/1307



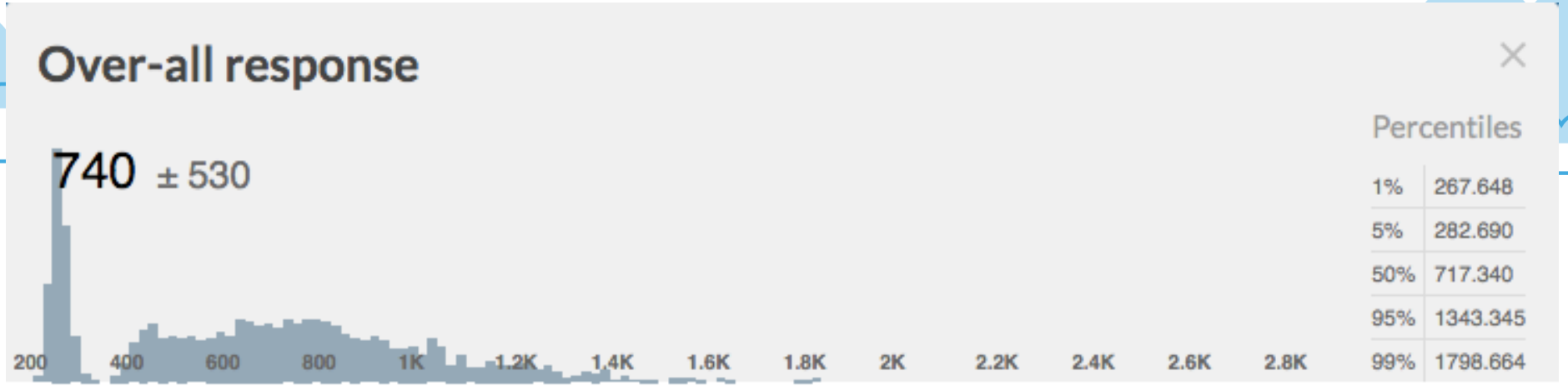
Hit rates: memcached 40% mysql 70%



Hit rates: memcached 60% mysql 70%



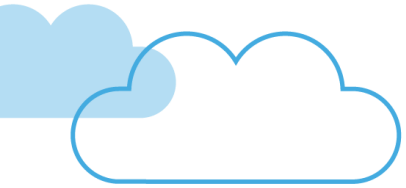
Hit rates: memcached 20% mysql 90%



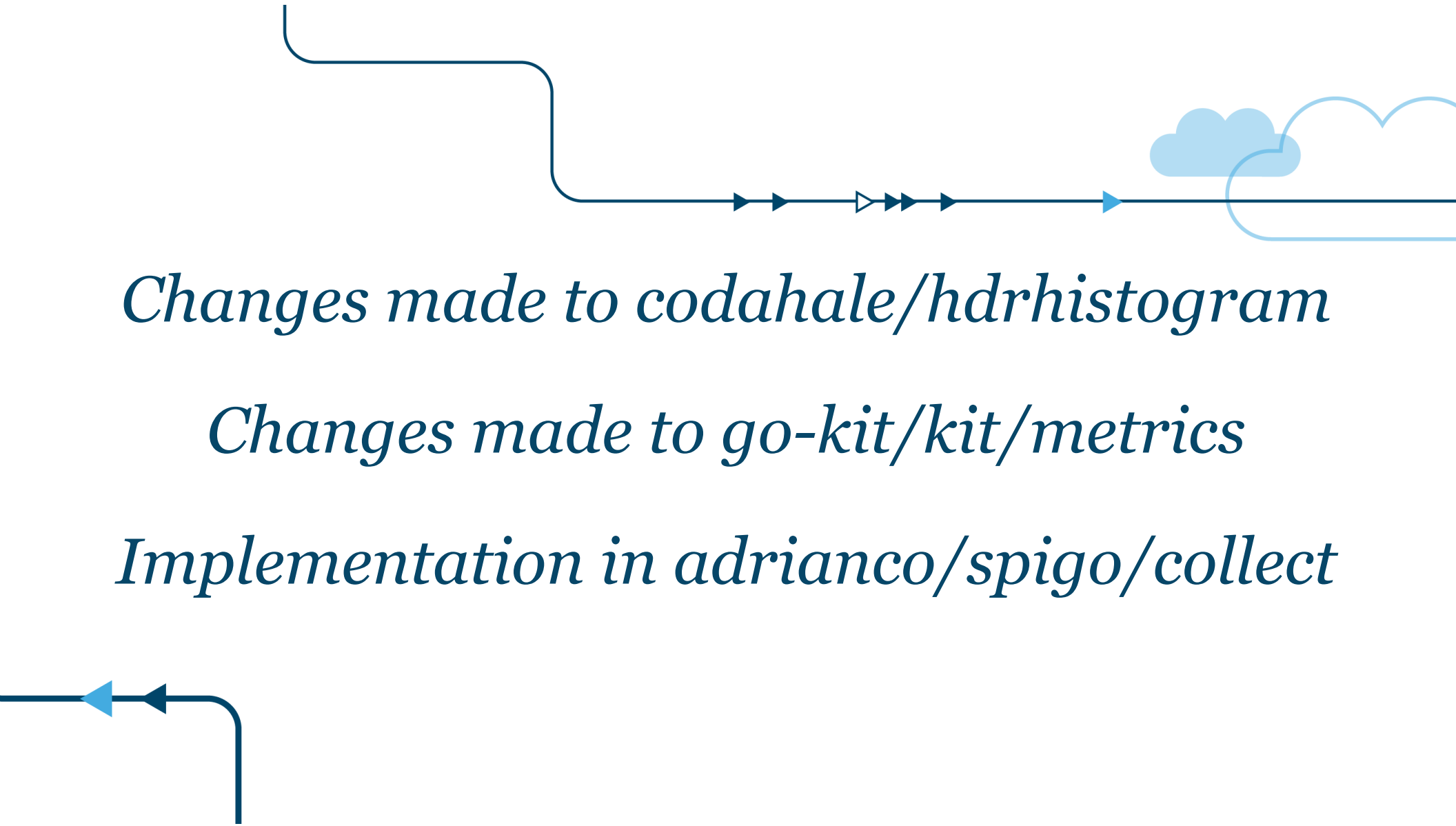
=GR+((randomInt(0,100)>IY)?QW:LR)

Reasoning

CPU time plus memcache hit lookup time is the first mode, second mode is memcache miss MySQL lookup with in-memory result, and third mode is MySQL lookup including disk access. Change the memcached and mysql hit rate %age to see how the height of each mode varies.




Measuring Response Time With Histograms

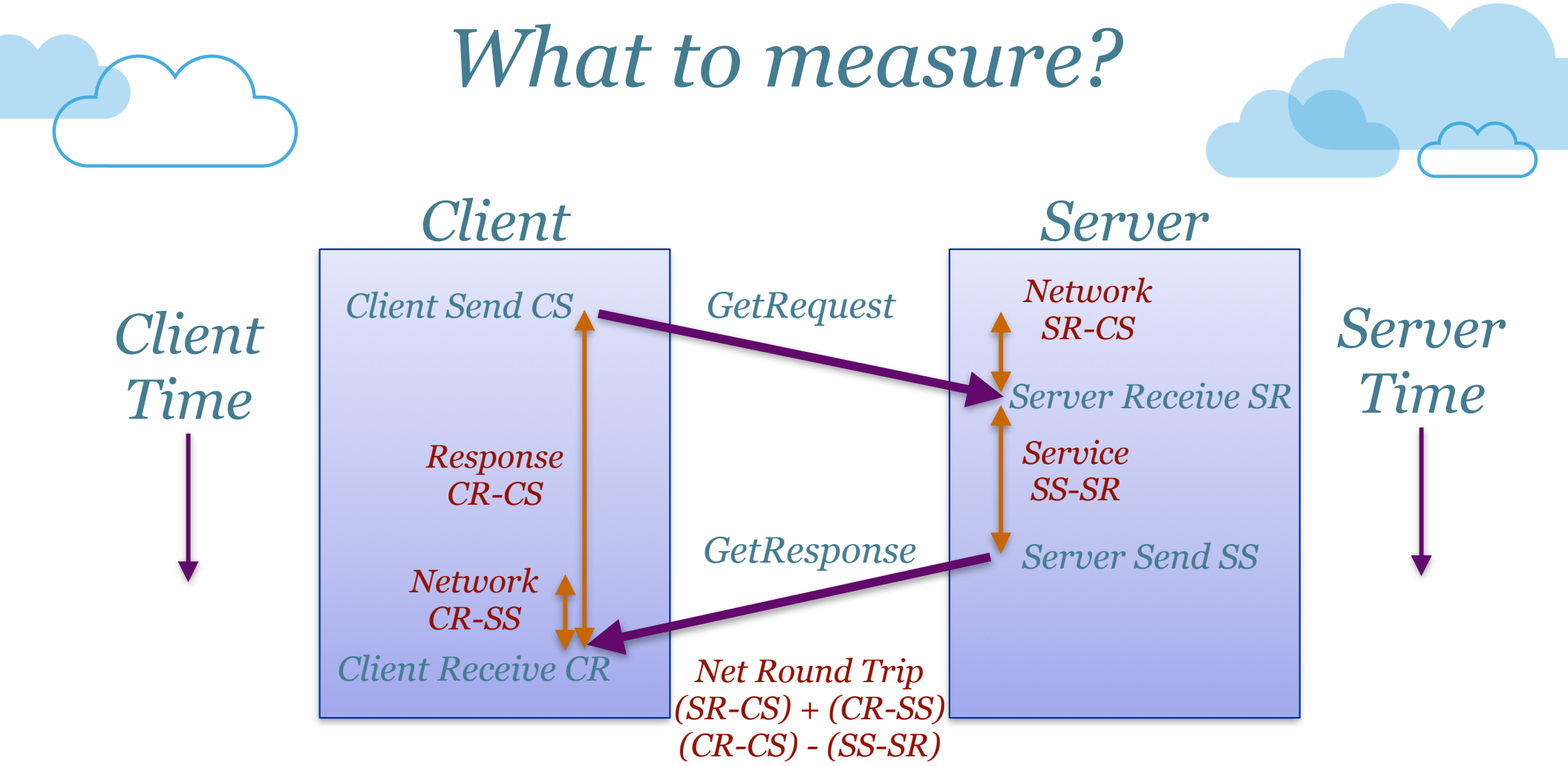


Changes made to codahale/hdrhistogram

Changes made to go-kit/kit/metrics

Implementation in adrianco/spigo/collect

What to measure?



Spigo Histogram Results

Collected with: `% spigo -d 60 -j -a storage -c`

```
name: storage.*.*..load00...load.denominator_serv
quantiles: [{50 47103} {99 139263}]
From    To       Count  Prob    Bar
20480   21503    2      0.0007  :
21504   22527    2      0.0007  |
23552   24575    1      0.0003  :
24576   25599    5      0.0017  |
25600   26623    5      0.0017  |
26624   27647    1      0.0003  |
27648   28671    3      0.0010  |
28672   29695    5      0.0017  |
29696   30719    127    0.0421  |####
30720   31743    126    0.0418  |####
31744   32767    74     0.0246  |##
32768   34815    281    0.0932  |#########
34816   36863    201    0.0667  |######
36864   38911    156    0.0518  |#####
38912   40959    185    0.0614  |######
40960   43007    147    0.0488  |####
43008   45055    161    0.0534  |#####
45056   47103    125    0.0415  |####
47104   49151    135    0.0448  |####
49152   51199    99     0.0328  |###
51200   53247    82     0.0272  |##
53248   55295    77     0.0255  |##
55296   57343    66     0.0219  |##
57344   59391    54     0.0179  |#
59392   61439    37     0.0123  |#
61440   63487    45     0.0149  |#
63488   65535    33     0.0109  |#
65536   69631    63     0.0209  |##
69632   73727    98     0.0325  |###
73728   77823    92     0.0305  |###
77824   81919    112    0.0372  |###
81920   86015    88     0.0292  |##
86016   90111    55     0.0182  |#
90112   94207    38     0.0126  |#
94208   98303    51     0.0169  |#
98304   102399   32     0.0106  |#
102400  106495   35     0.0116  |#
106496  110591   17     0.0056  |
110592  114687   19     0.0063  |
114688  118783   18     0.0060  |
118784  122879   6      0.0020  |
122880  126975   8      0.0027  |
```

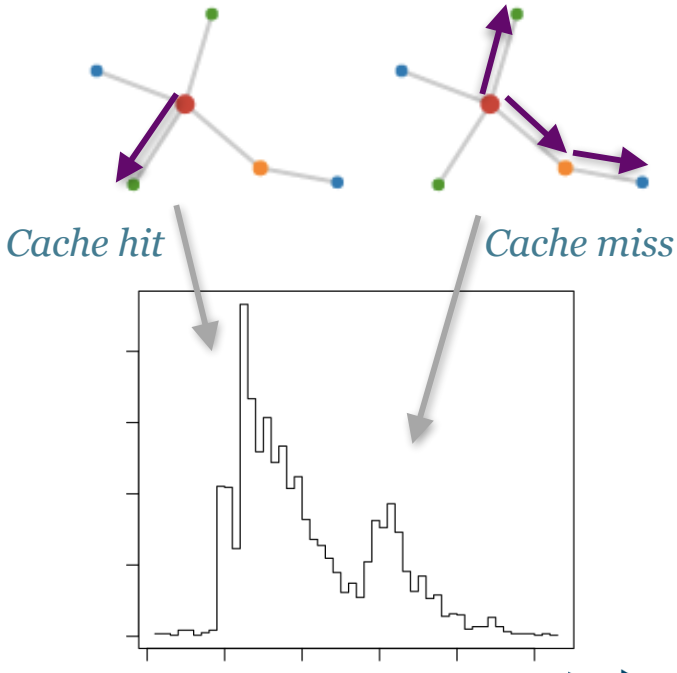
service time for load generator

Median and 99th percentile values

Response time distribution measured in nanoseconds using High Dynamic Range Histogram

Normalized probability

:# Zero counts skipped
|# Contiguous buckets



Go-Kit Histogram Example

```
const (
	maxHistObservable = 1000000 // one millisecond
	sampleCount       = 1000    // data points will be sampled 5000 times to build a distribution by guesstimate
)

var sampleMap map[metrics.Histogram][]int64
var sampleLock sync.Mutex

func NewHist(name string) metrics.Histogram {
	var h metrics.Histogram
	if name != "" && archaius.Conf.Collect {
		h = expvar.NewHistogram(name, 1000, maxHistObservable, 1, []int{50, 99}...)
		sampleLock.Lock()
		if sampleMap == nil {
			sampleMap = make(map[metrics.Histogram][]int64)
		}
		sampleMap[h] = make([]int64, 0, sampleCount)
		sampleLock.Unlock()
		return h
	}
	return nil
}

func Measure(h metrics.Histogram, d time.Duration) {
	if h != nil && archaius.Conf.Collect {
		if d > maxHistObservable {
			h.Observe(int64(maxHistObservable))
		} else {
			h.Observe(int64(d))
		}
		sampleLock.Lock()
		s := sampleMap[h]
		if s != nil && len(s) < sampleCount {
			sampleMap[h] = append(s, int64(d))
		sampleLock.Unlock()
		}
	}
}
```

Nanoseconds resolution!

Median and 99%ile

Slice for first 500 values as samples for export to Guesstimate

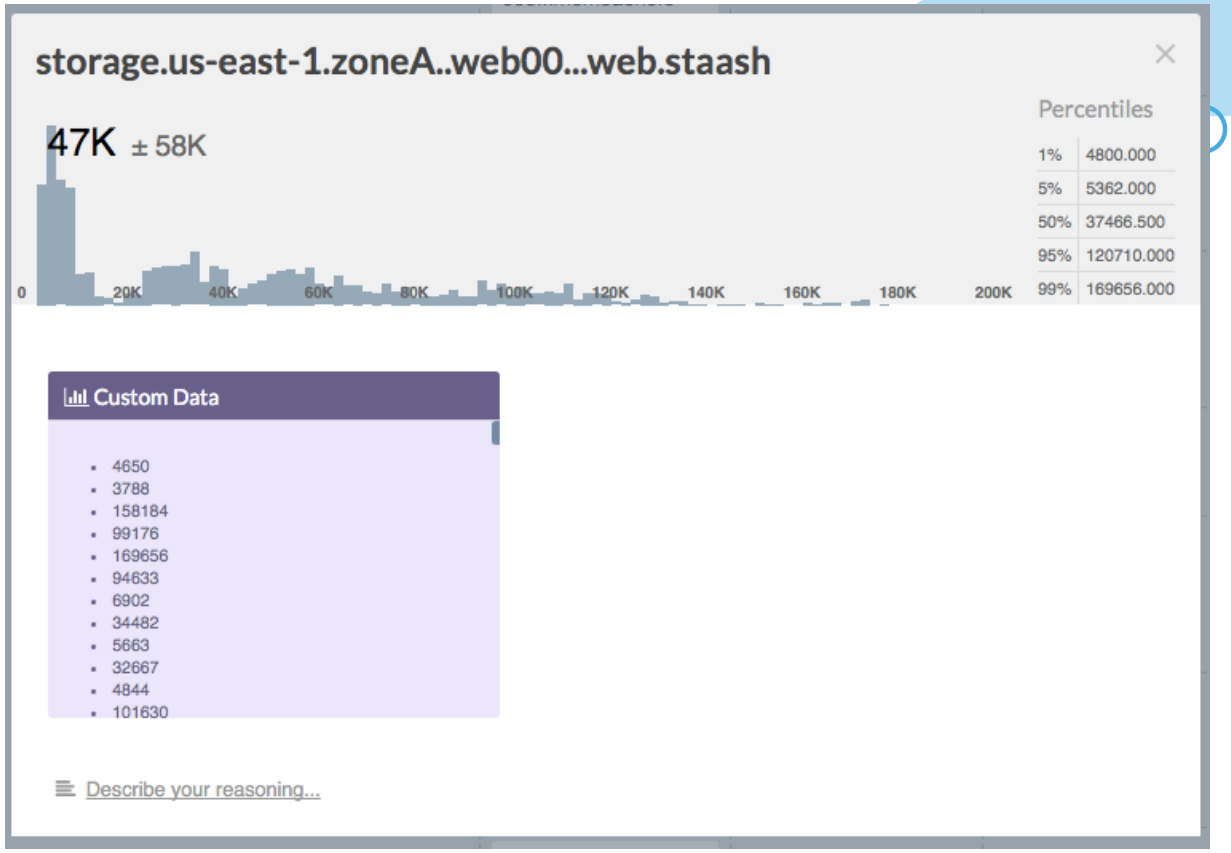
Golang Guesstimate Interface

https://github.com/adrianco/goguesstimate


```
{
      "space": {
      "name": "gotest",
      "description": "Testing",
      "is_private": "true",
      "graph":  {
          "metrics": [
              {"id": "AB", "readableId": "AB", "name": "memcached", "location": {"row": 2, "column":4}},
              {"id": "AC", "readableId": "AC", "name": "memcached percent", "location": {"row": 2,
"column":3}},
              {"id": "AD", "readableId": "AD", "name": "staash cpu", "location": {"row": 3, "column":3}},
              {"id": "AE", "readableId": "AE", "name": "staash", "location": {"row": 3, "column":2}}
              ],
          "guesstimates": [
           {"metric": "AB", "input": null, "guesstimateType": "DATA", "data":
[119958,6066,13914,9595,6773,5867,2347,1333,9900,9404,13518,9021,7915,3733,10244,5461,12243,7931,9044,11706,
5706,22861,9022,48661,15158,28995,16885,9564,17915,6610,7080,7065,12992,35431,11910,11465,14455,25790,8339,9
991]},
              {"metric": "AC", "input": "40", "guesstimateType": "POINT"},
              {"metric": "AD", "input": "[1000,4000]", "guesstimateType": "LOGNORMAL"},
              {"metric": "AE", "input": "=100+((randomInt(0,100)>AC)?AB:AD)", "guesstimateType": "FUNCTION"}
          ]
      }
  }
}
```

See http://www.getguesstimate.com

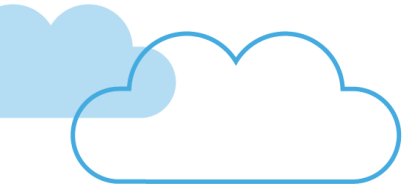
```
% cd json_metrics; sh guesstimate.sh storage
```



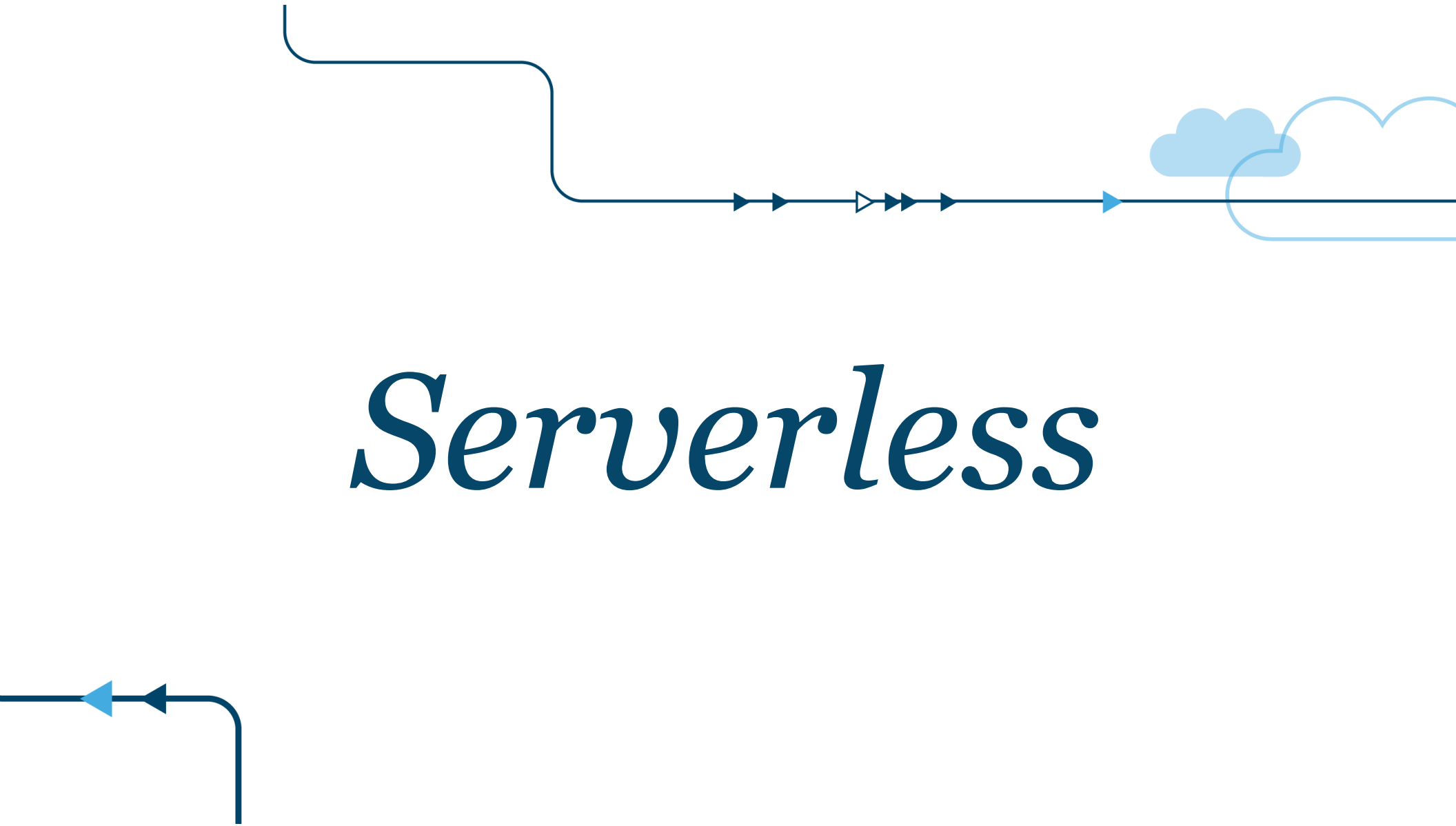
Simplicity through symmetry

Symmetry
Invariants
Stable assertions
No special cases
Single purpose components

@adrianco
BV
Battery Ventures



What's Next?



Serverless

Serverless Architectures

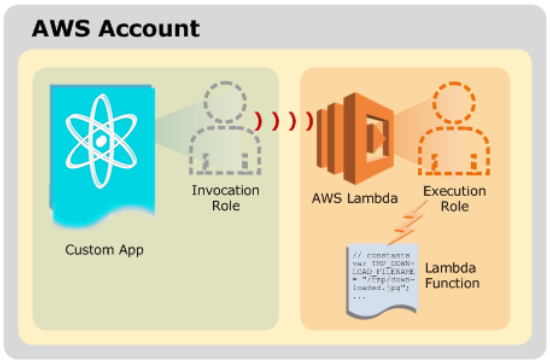
AWS Lambda starting to take off

Google Cloud Functions, Azure Functions on the way…

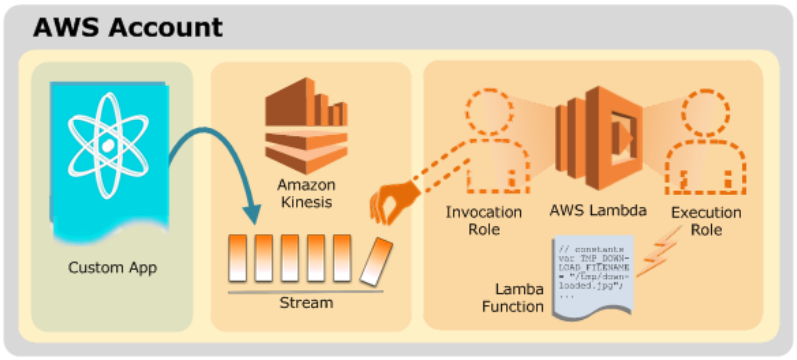
Open Source: IBM OpenWhisk, open-lambda.org, apex.run

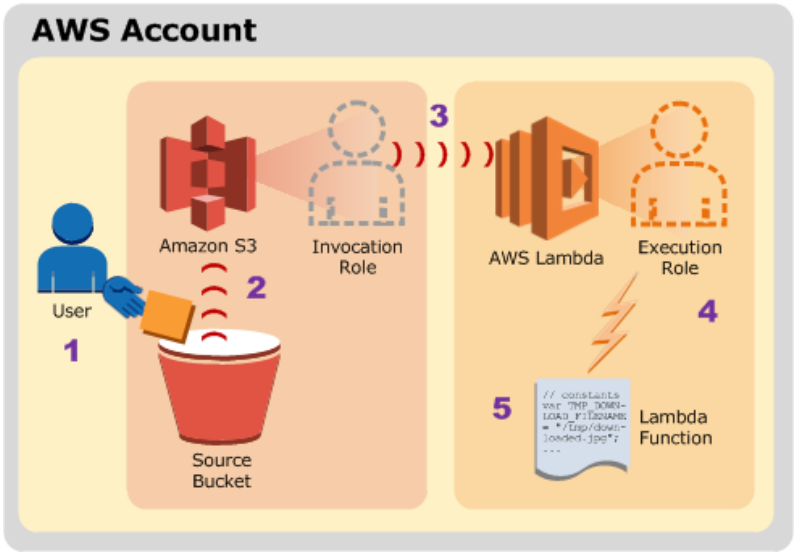
Startup activity: iron.io , serverless.com, nano-lambda.com

With AWS Lambda
compute resources are charged
by the 100ms, not the hour

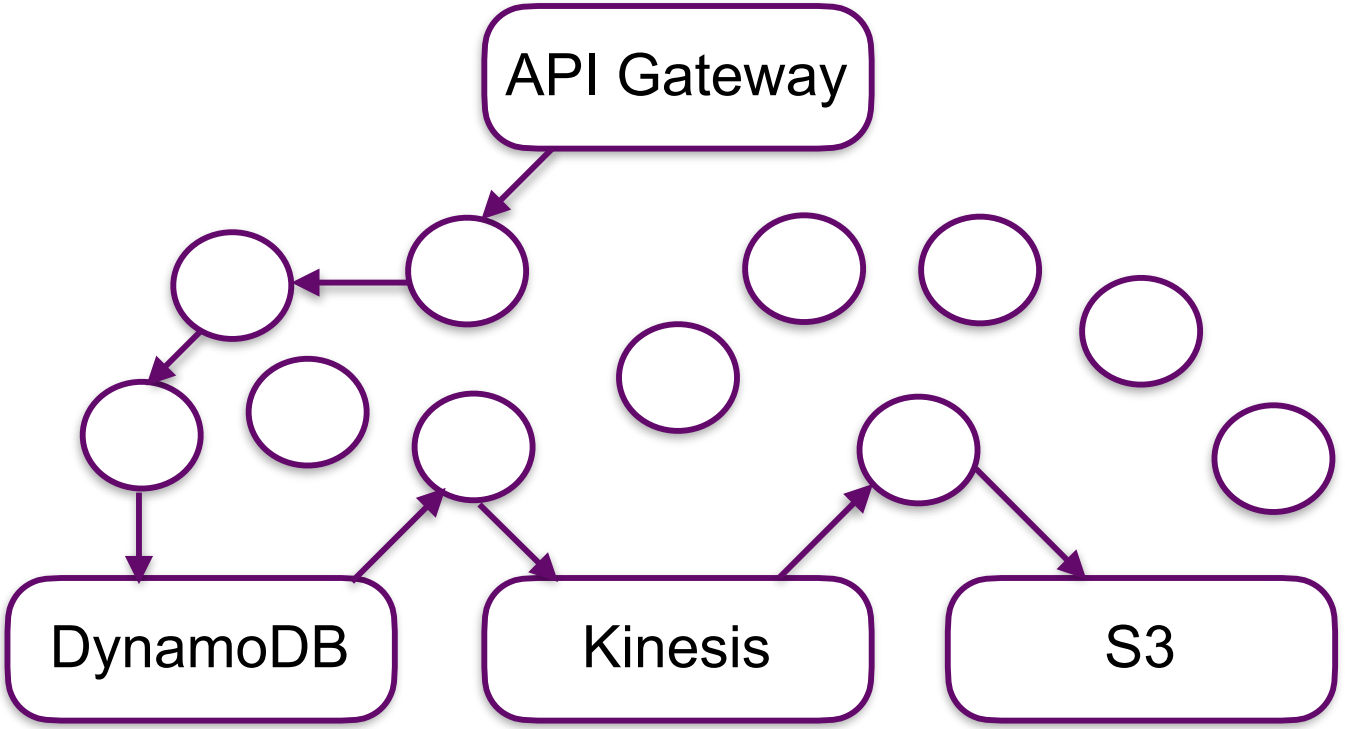


First 1M node.js executions/month are free





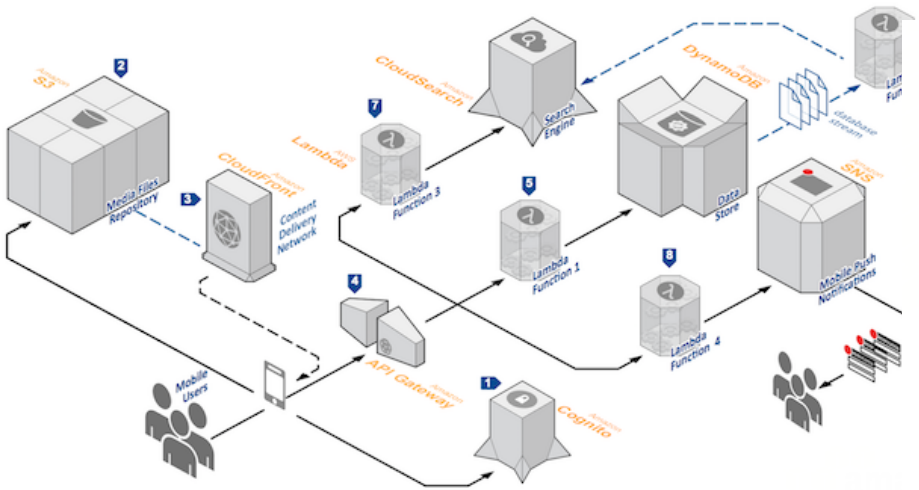
Monitorless Architecture

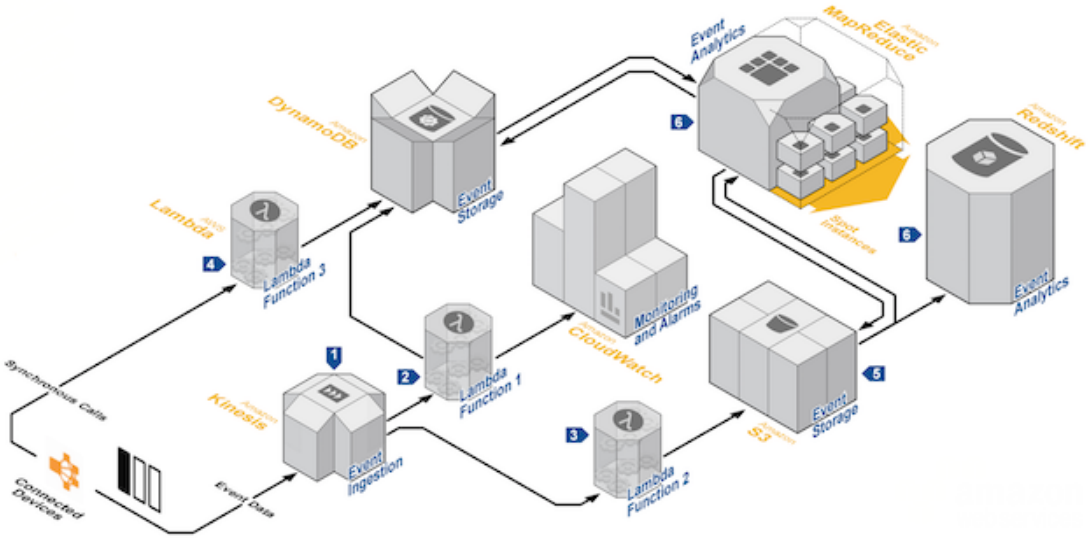


@adrianco
BV
Battery Ventures

AWS Lambda Reference Arch

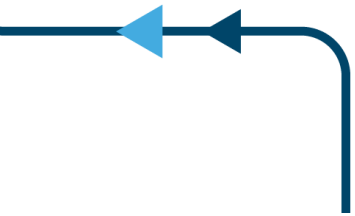
http://www.allthingsdistributed.com/2016/05/aws-lambda-serverless-reference-architectures.html

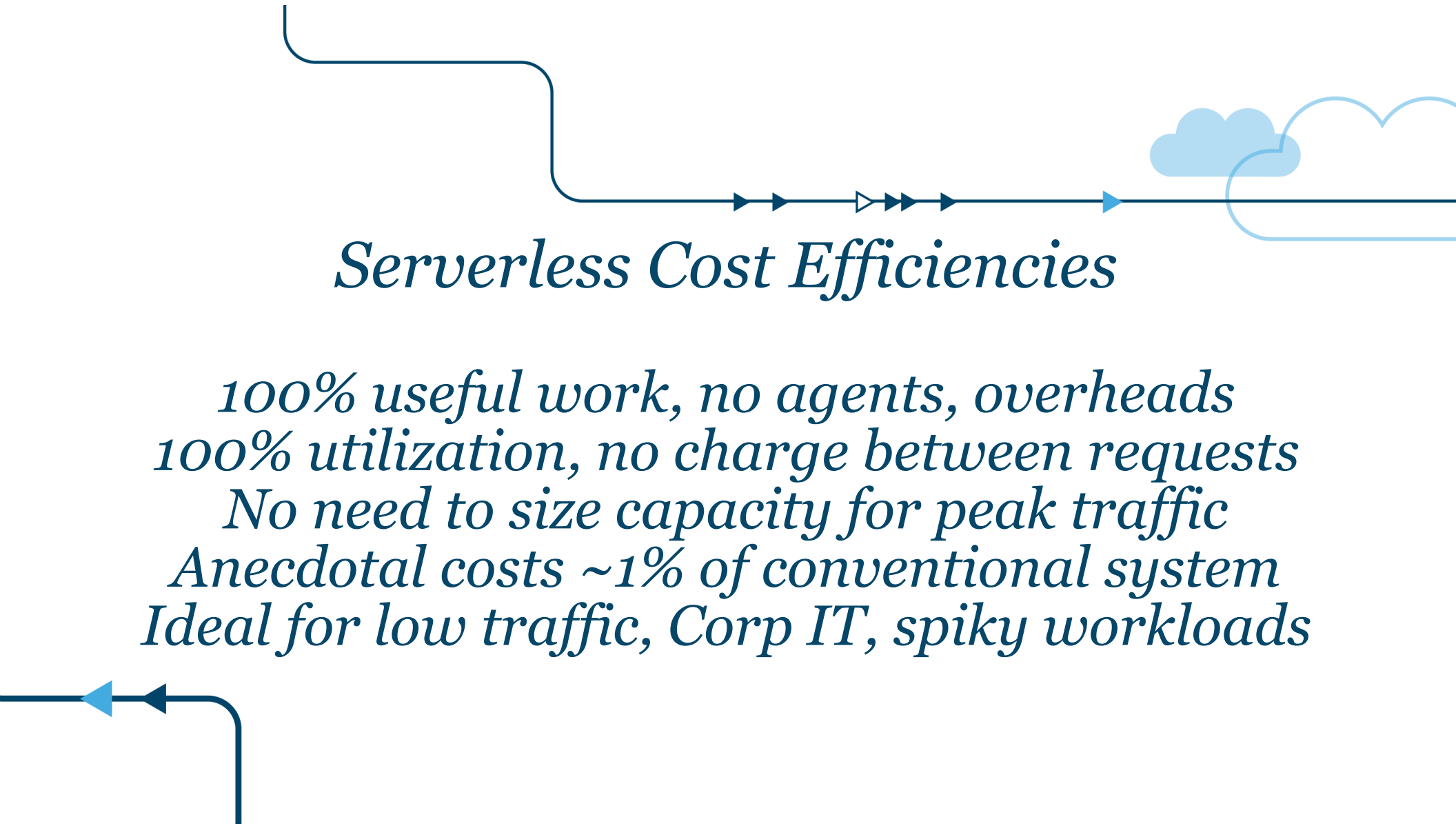




Serverless Programming Model

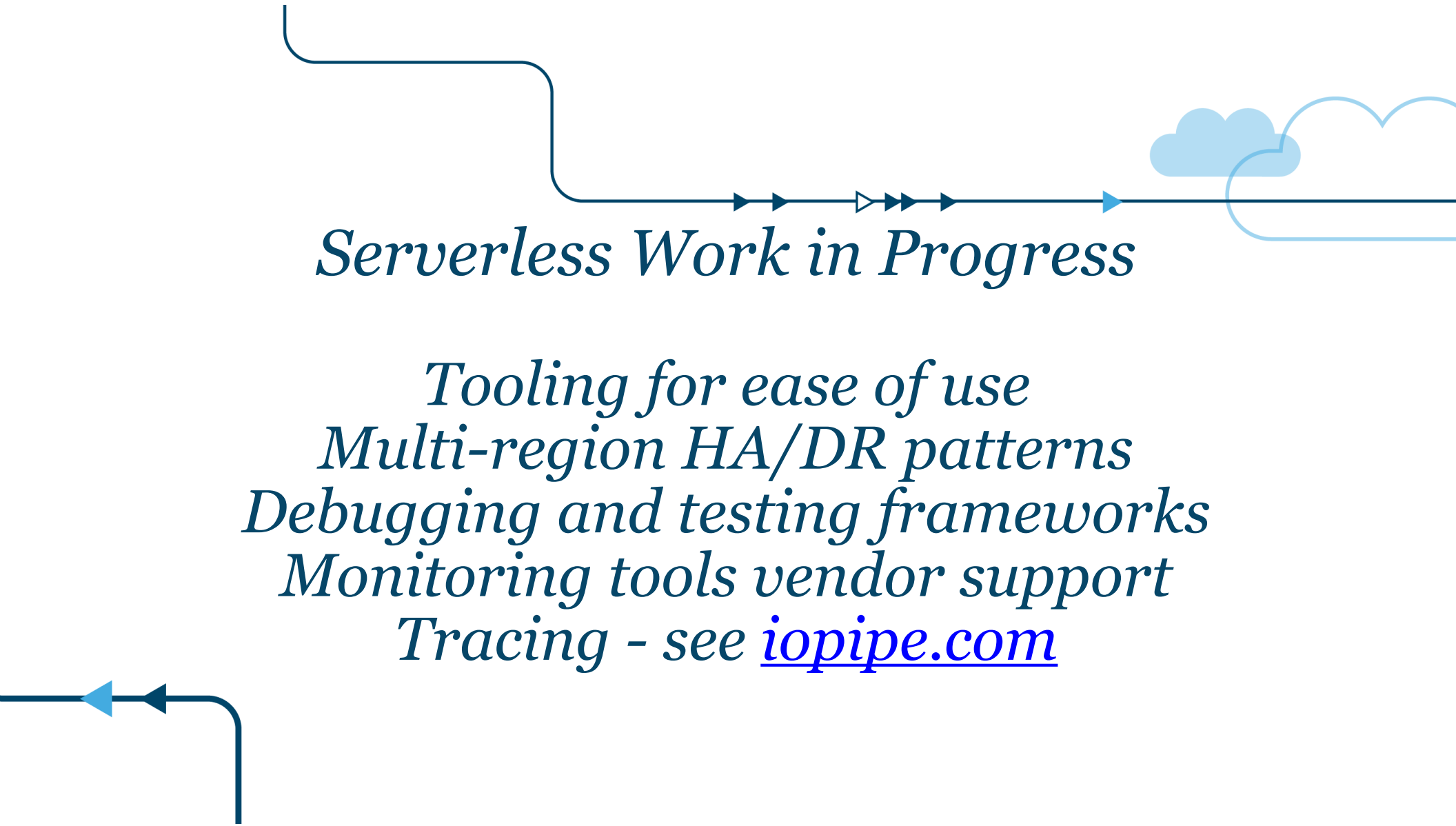
Event driven functions
Role based permissions
Whitelisted API based security
Good for simple single threaded code





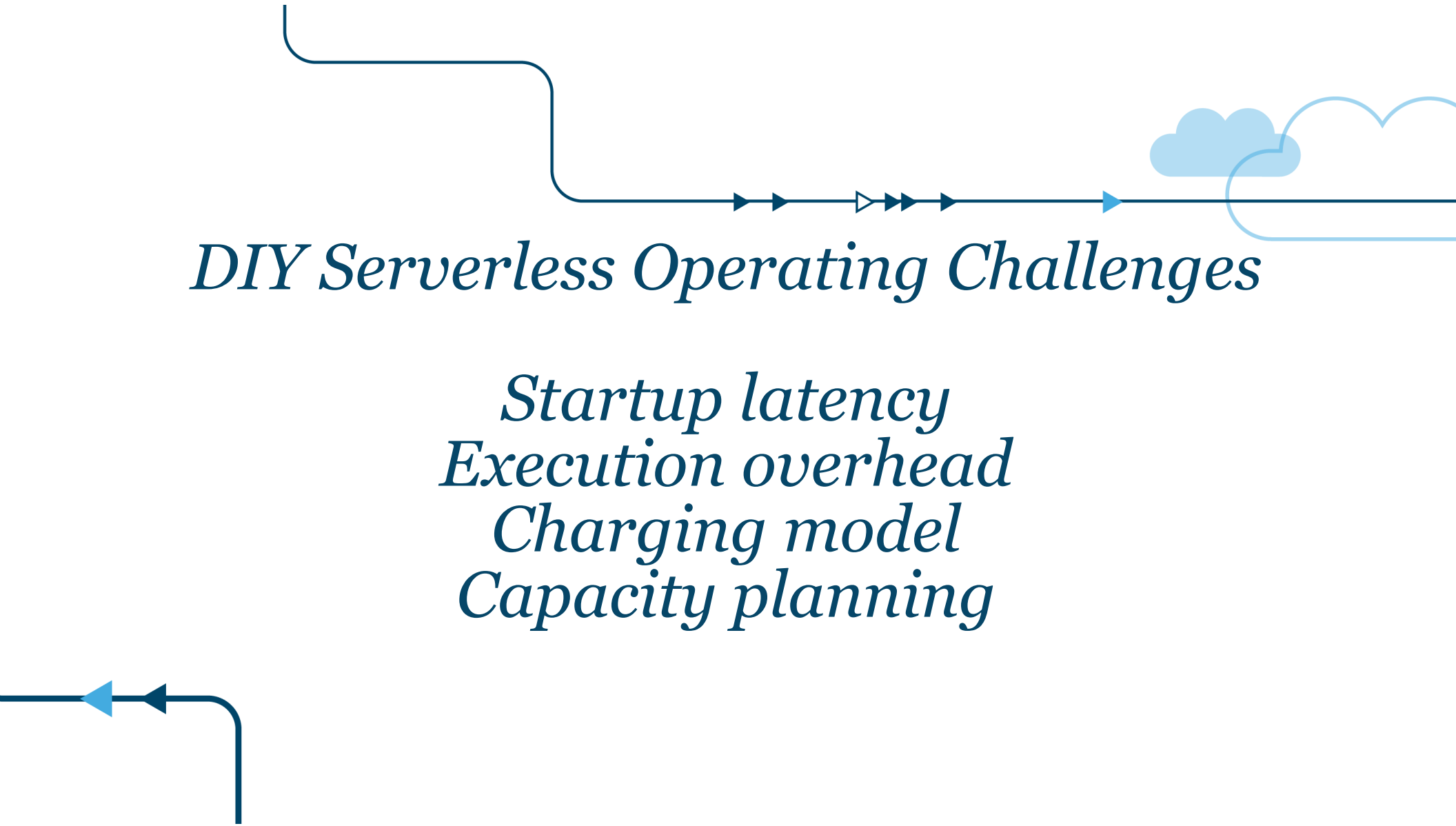
Serverless Cost Efficiencies

100% useful work, no agents, overheads
100% utilization, no charge between requests
No need to size capacity for peak traffic
Anecdotal costs ~1% of conventional system
Ideal for low traffic, Corp IT, spiky workloads



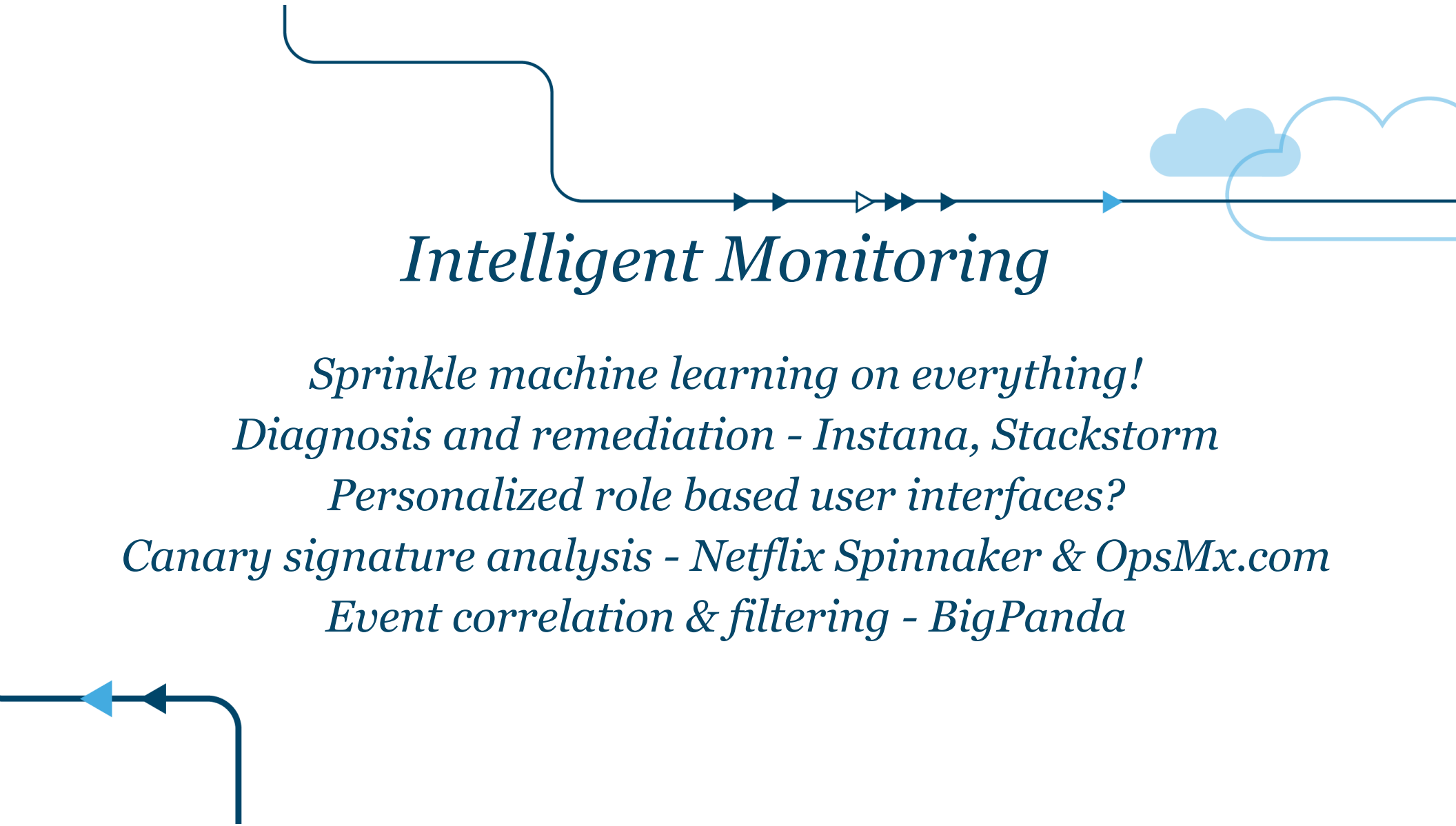
Serverless Work in Progress

Tooling for ease of use
Multi-region HA/DR patterns
Debugging and testing frameworks
Monitoring tools vendor support
Tracing - see iopipe.com



DIY Serverless Operating Challenges

Startup latency
Execution overhead
Charging model
Capacity planning



Intelligent Monitoring

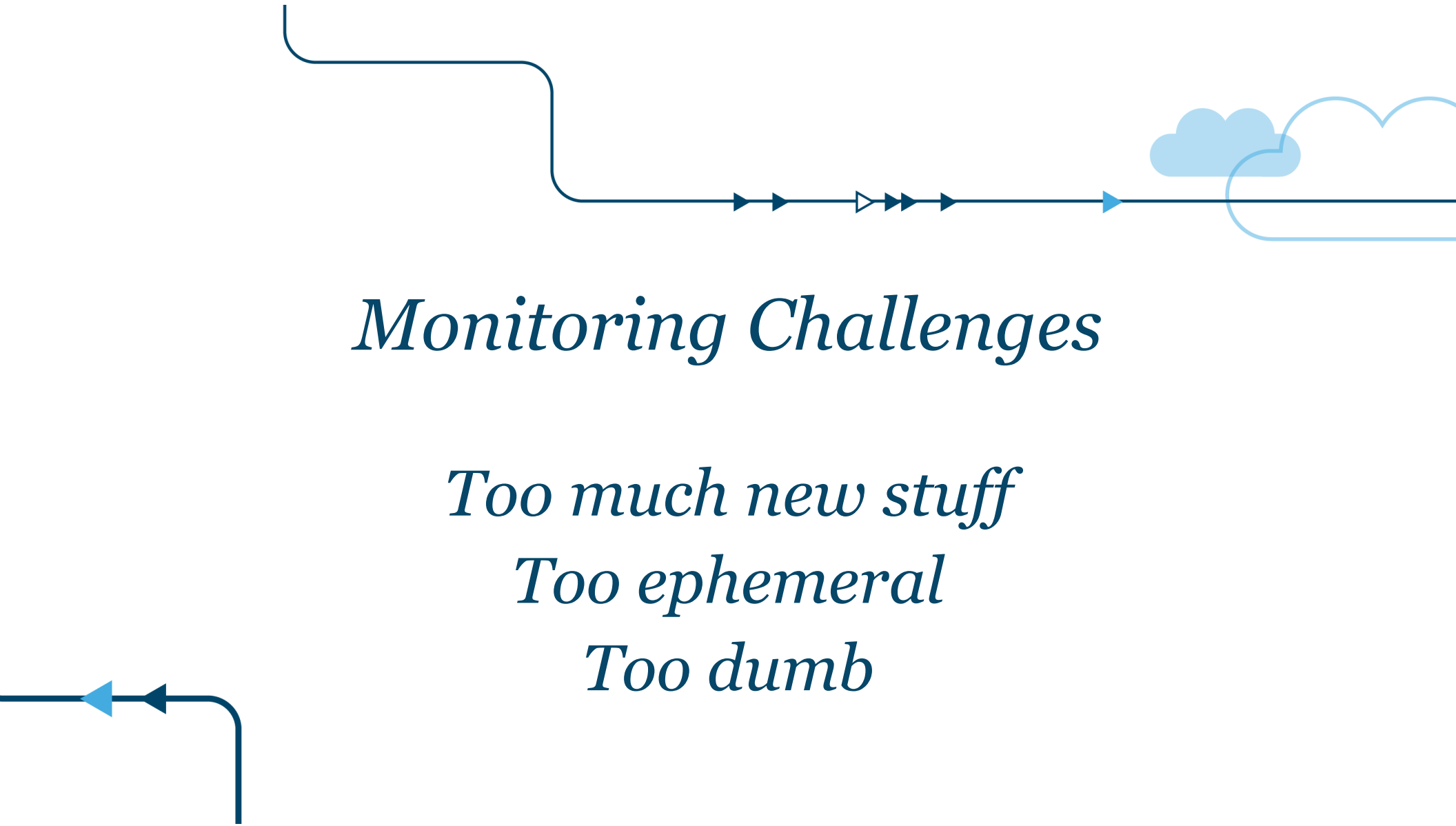
Sprinkle machine learning on everything!

Diagnosis and remediation - Instana, Stackstorm

Personalized role based user interfaces?

Canary signature analysis - Netflix Spinnaker & OpsMx.com

Event correlation & filtering - BigPanda

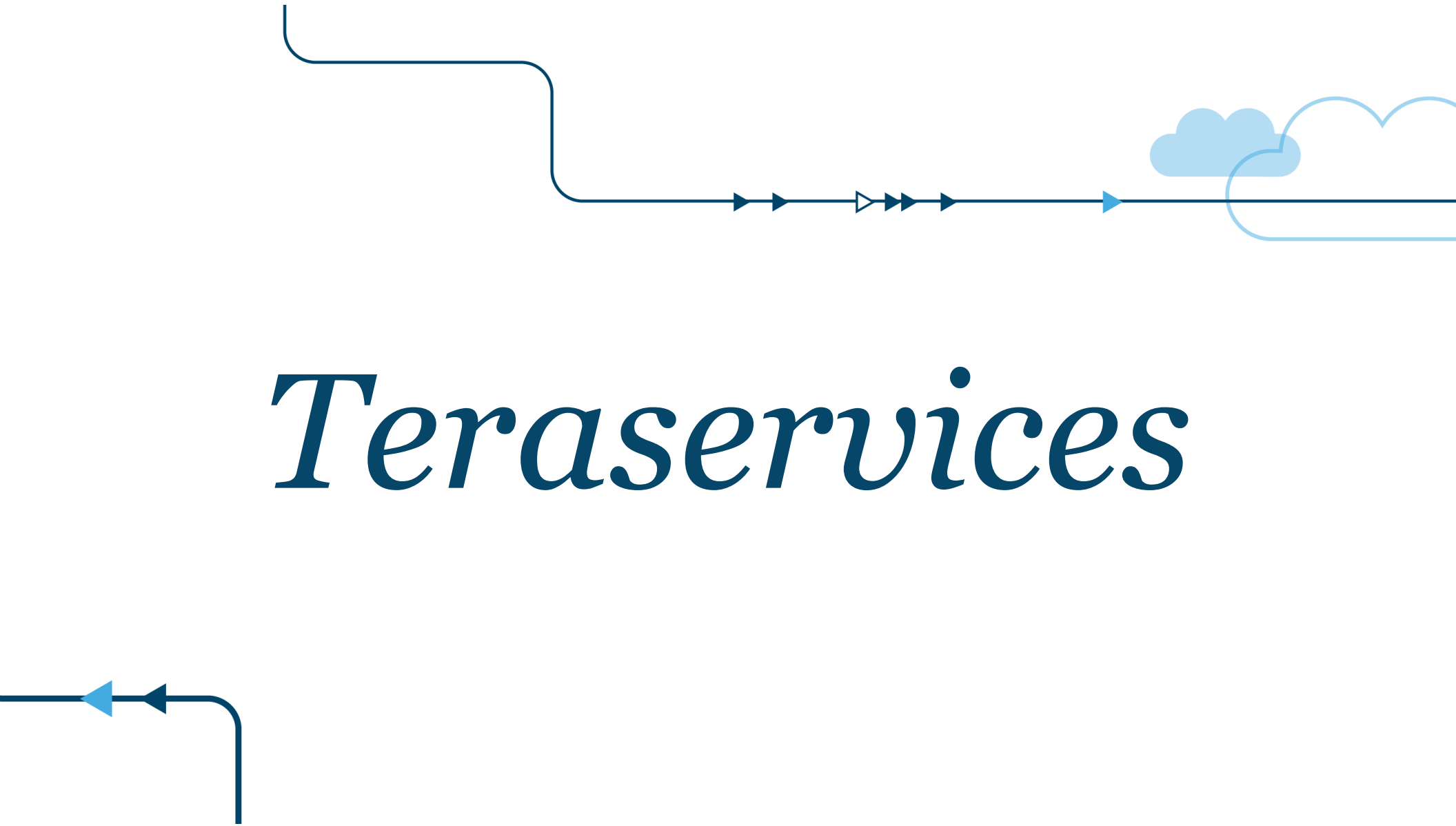


Monitoring Challenges

Too much new stuff

Too ephemeral

Too dumb



Teraservices

Terabyte Memory Directions

Engulf dataset in memory for analytics

Balanced config for memory intensive workloads

Replace high end systems at commodity cost point

Explore non-volatile memory implications

Terabyte Memory Options

Flash based Diablo 64/128/256GB DDR4 DIMM

Shipping now as volatile memory, future non-volatile

Intel 3D XPoint - new non-volatile technology on the way

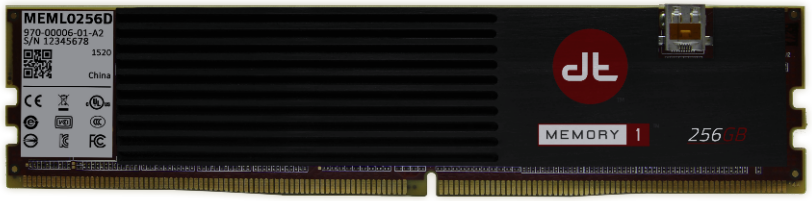
Announced May 2016

AWS X1 Instance Type - over 2TB RAM for $14/hr

Easy availability should drive innovation

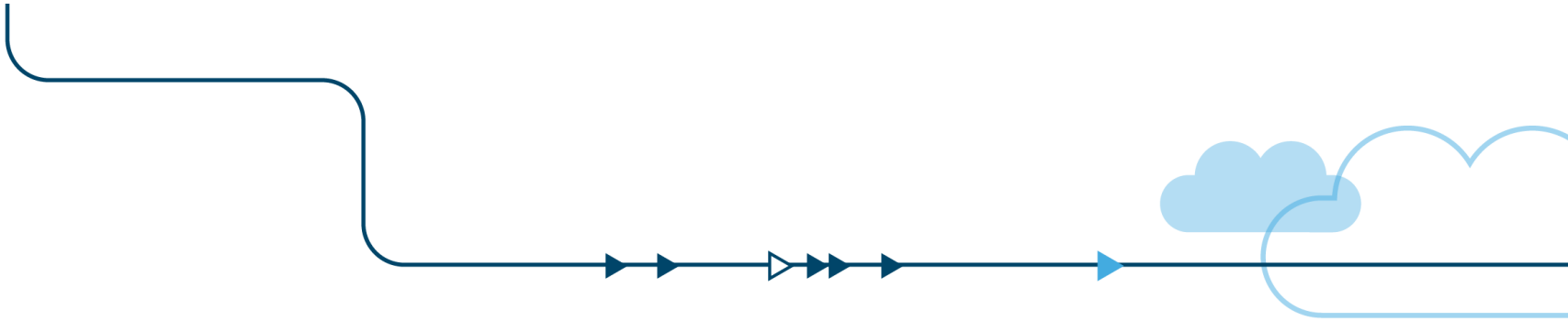
Diablo Memory1: Flash DIMM

MEMORY 1™



- ✓ **UP TO 256GB DDR4 MEMORY PER MODULE**
- ✓ **UP TO 4TB MEMORY IN 2 SOCKET SYSTEM**

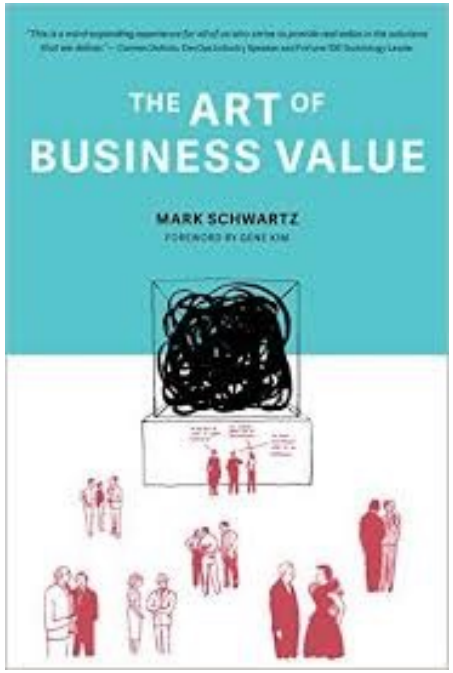
- **NO CHANGES** to CPU or Server
- **NO CHANGES** to Operating System
- **NO CHANGES** to Applications

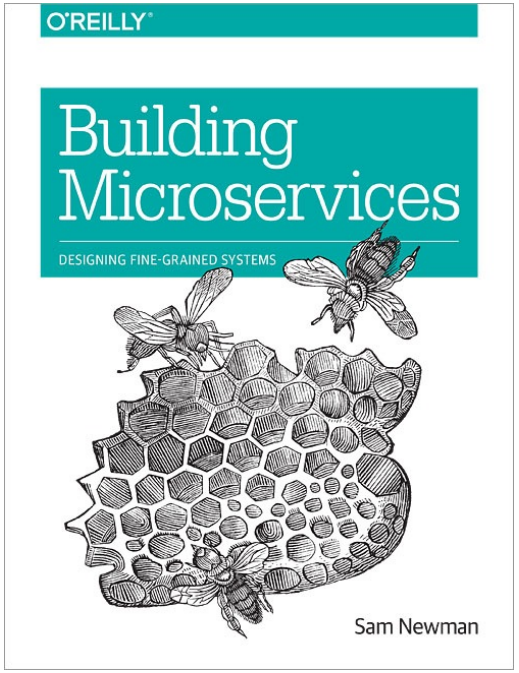


"We see the world as increasingly more complex and chaotic because we use inadequate concepts to explain it. When we understand something, we no longer see it as chaotic or complex."

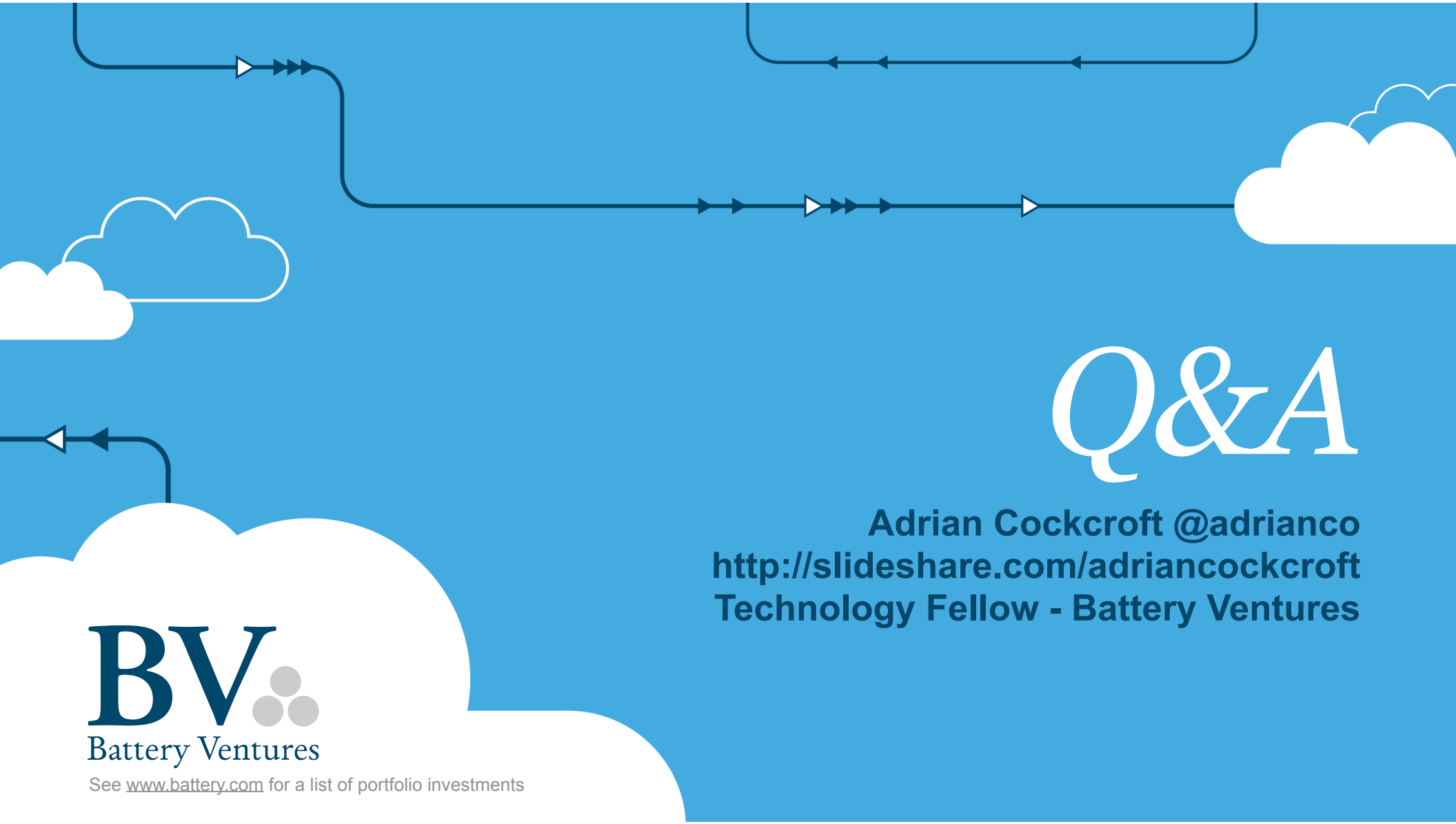
Jamshid Gharajedaghi - 2011
Systems Thinking: Managing Chaos and Complexity: A Platform for Designing Business Architecture

Learn More…


THE ART OF
BUSINESS VALUE
MARK SCHWARTZ


O'REILLY
Building
Microservices
DESIGNING FINE-GRAINED SYSTEMS
Sam Newman


AWS Lambda
A Guide to Serverless Microservices
By: Matthew Fuller



Q&A

Adrian Cockcroft @adrianco
http://slideshare.com/adriancockcroft
Technology Fellow - Battery Ventures

BV
Battery Ventures

See www.battery.com for a list of portfolio investments

BV
Battery Ventures

Enterprise IT

Visit http://www.battery.com/our-companies/ for a full list of all portfolio companies in which all Battery Funds have invested.