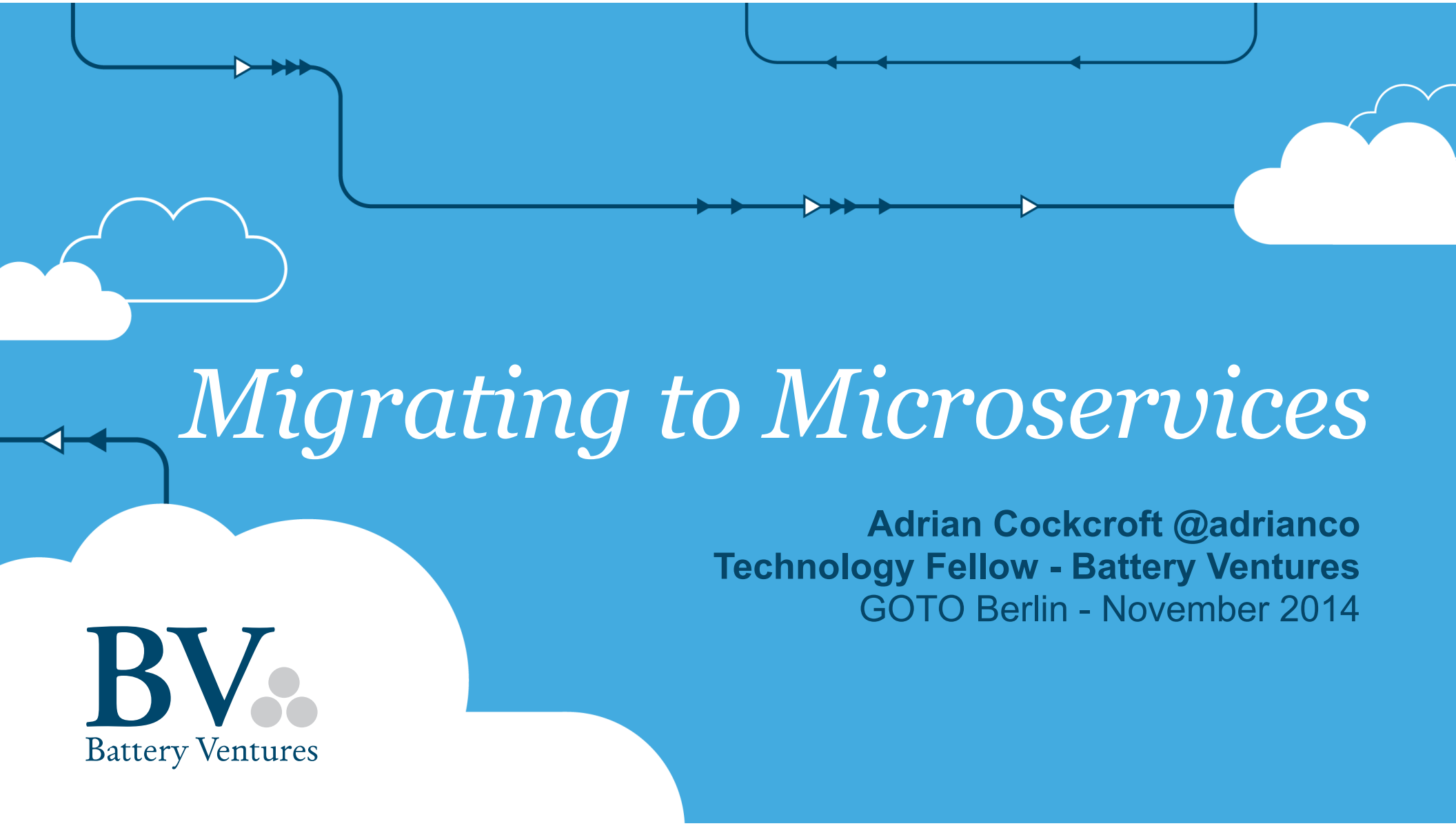


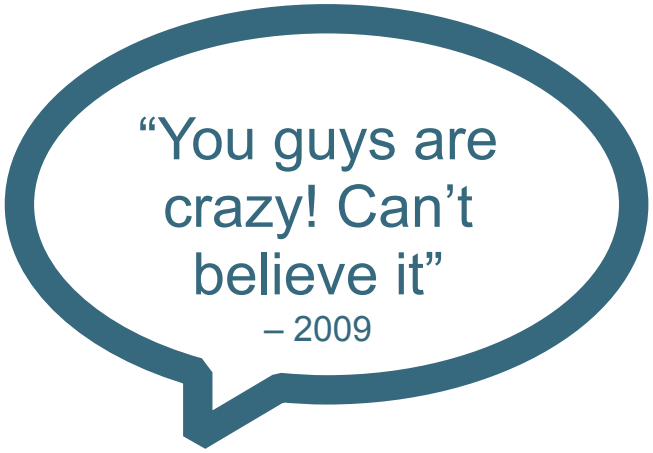
Migrating to Microservices

Adrian Cockcroft @adrianco
Technology Fellow - Battery Ventures
GOTO Berlin - November 2014

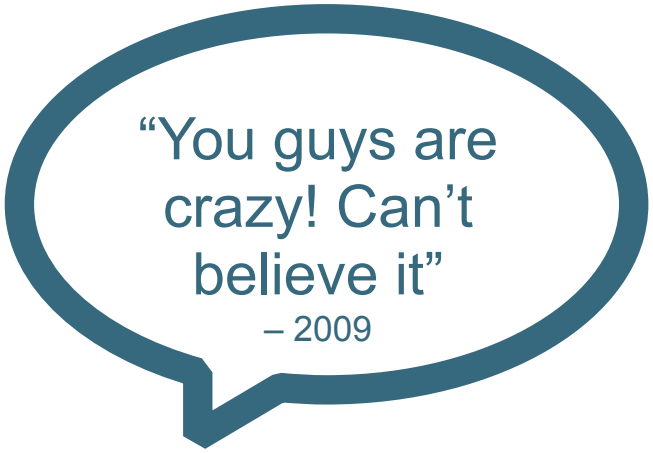
BV
Battery Ventures

Typical reactions to my Netflix talks…

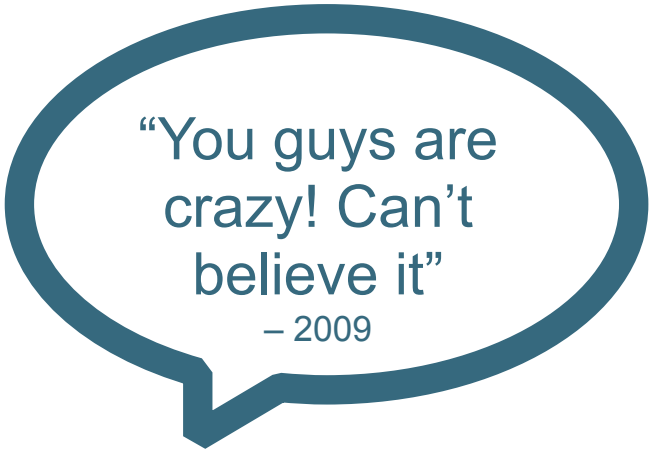
Typical reactions to my Netflix talks…



Typical reactions to my Netflix talks…



Typical reactions to my Netflix talks…



Typical reactions to my Netflix talks…

Typical reactions to my Netflix talks…

What I learned from my time at Netflix

What I learned from my time at Netflix

- *Speed wins in the marketplace*

What I learned from my time at Netflix

- *Speed wins in the marketplace*
- *Remove friction from product development*

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*

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*

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*

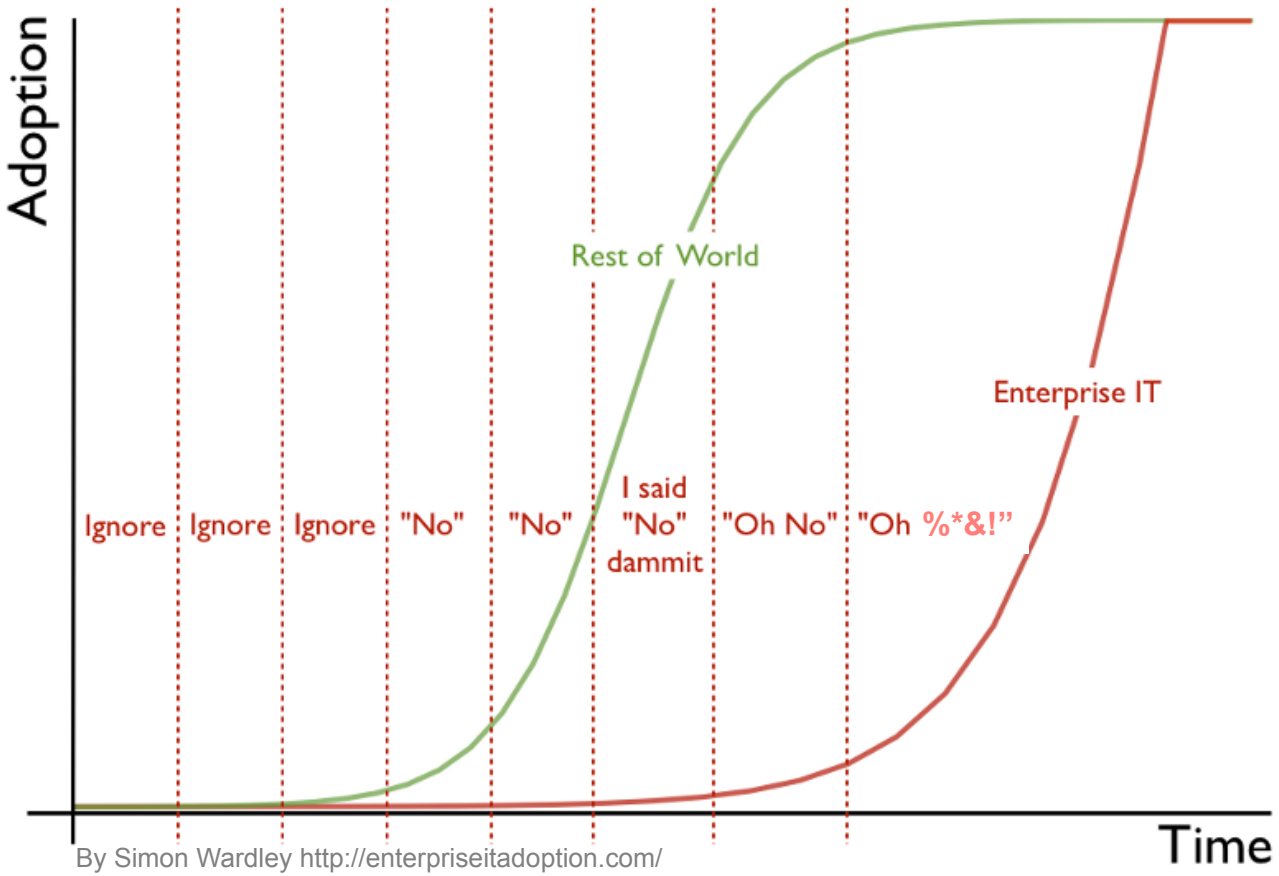
What I learned from my time at Netflix

- *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*

What I learned from my time at Netflix

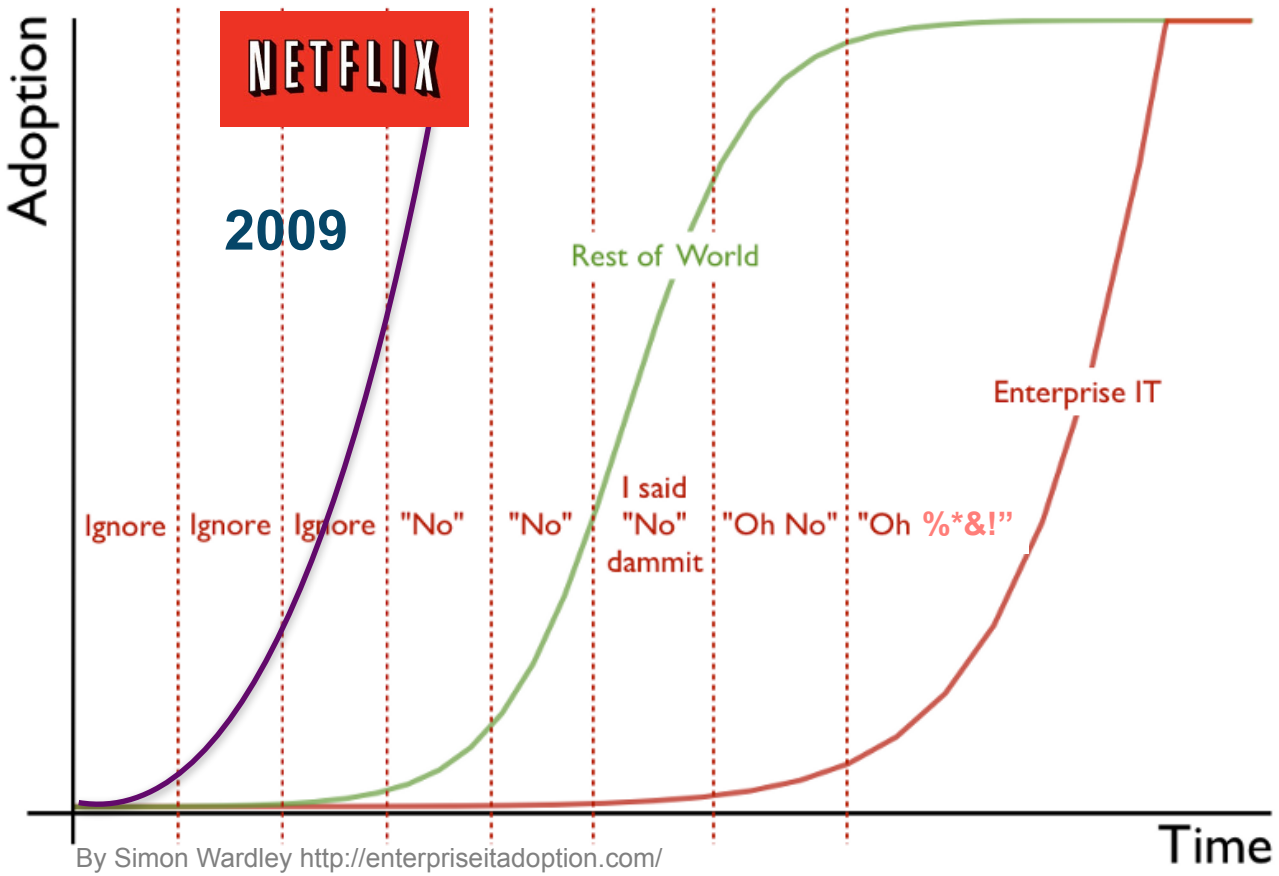
- *Speed wins in the marketplace*
- *Remove friction from product development*
- *High trust, low process, no hand-offs between teams*
- *Freedom and responsibility culture*
- *Don't do your own undifferentiated heavy lifting*
- *Use simple patterns automated by tooling*
- *Self service cloud makes impossible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

Cloud Adoption

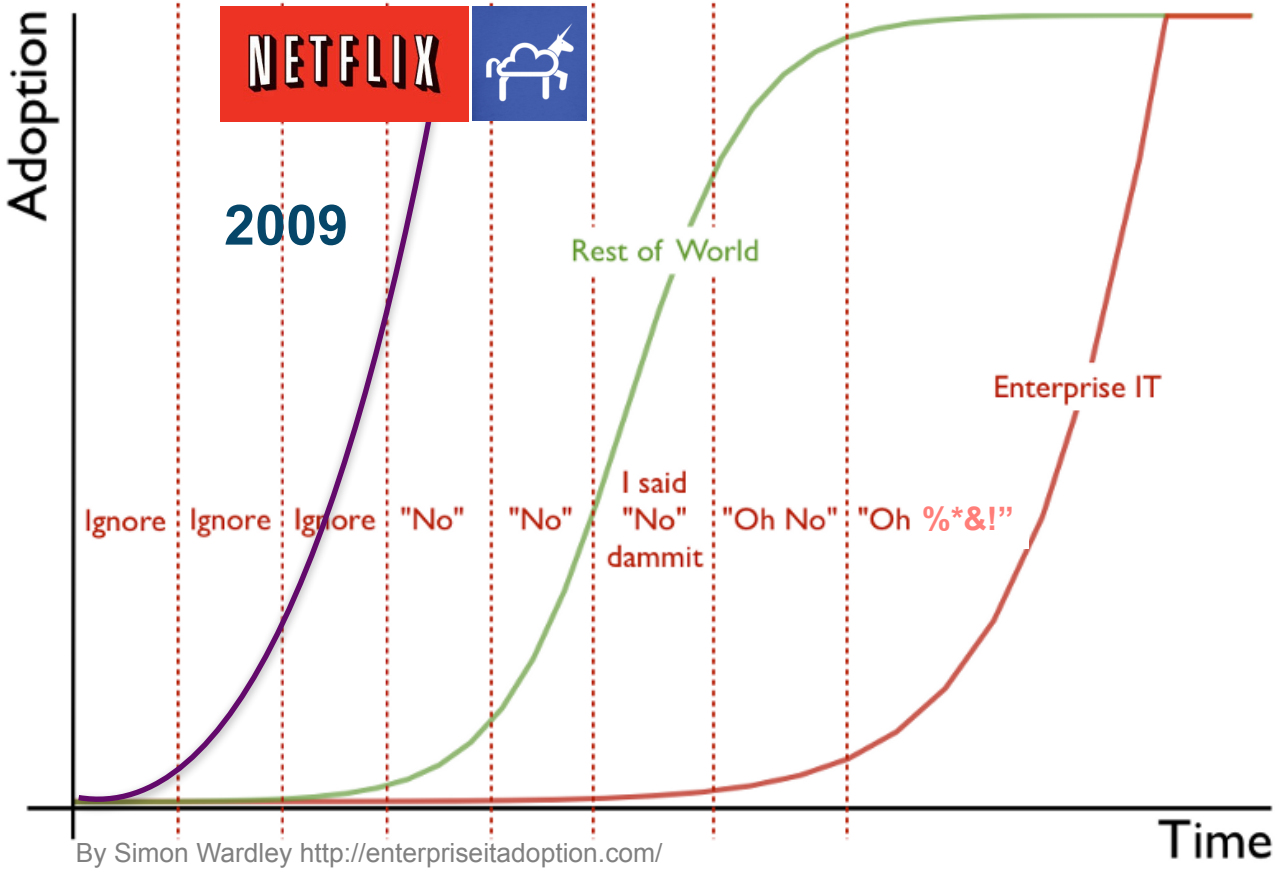


By Simon Wardley http://enterpriseitadoption.com/

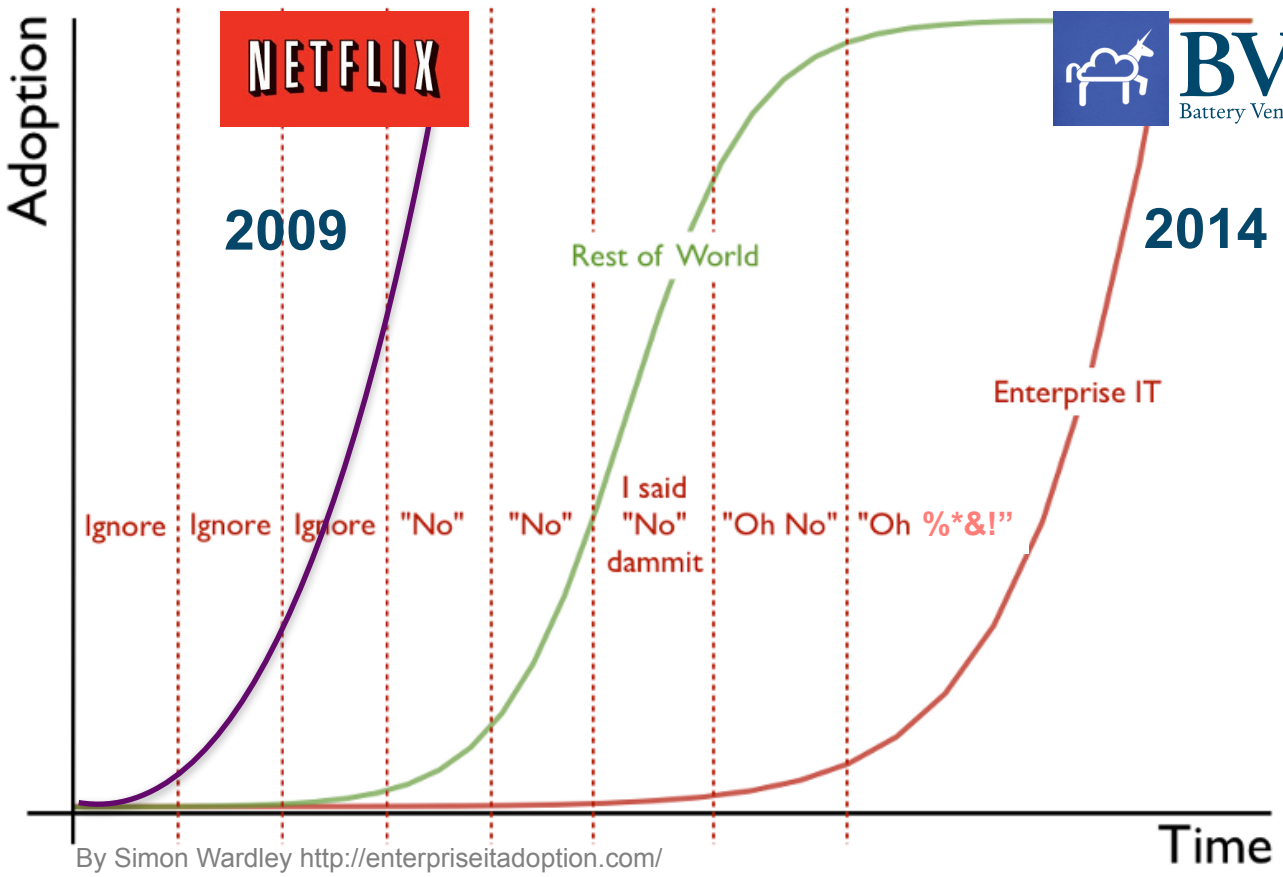
Cloud Adoption



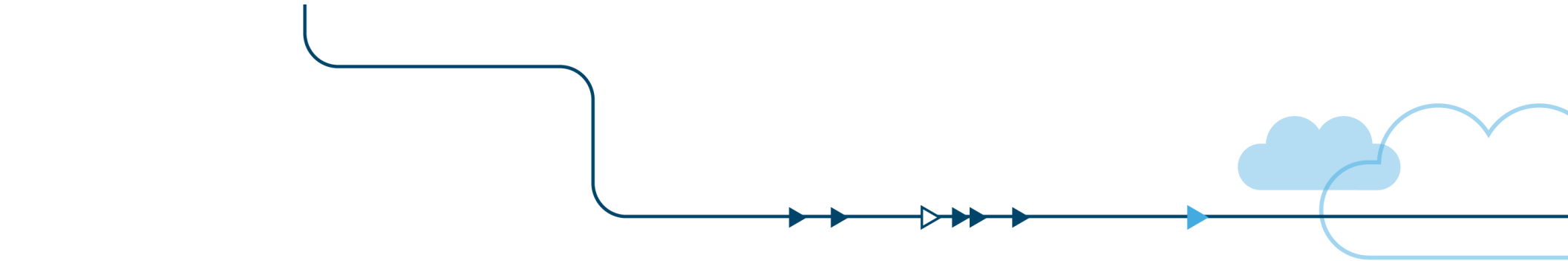
Cloud Adoption



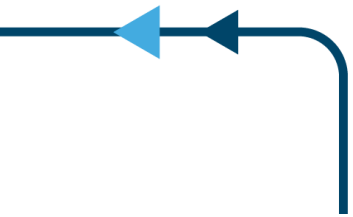
Cloud Adoption

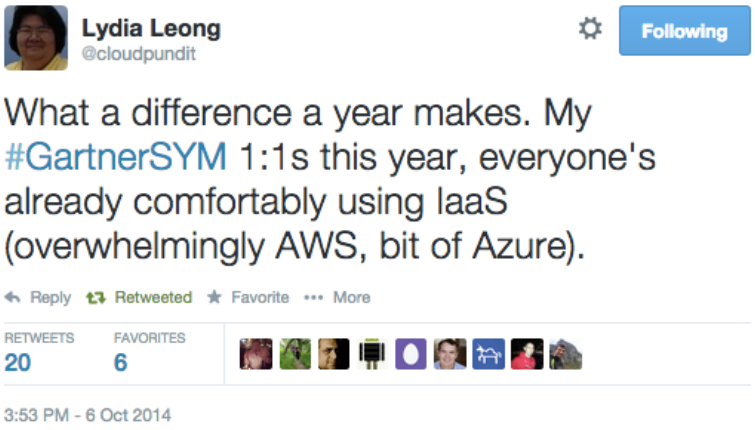


@adrianco's new job at the intersection of cloud and Enterprise IT

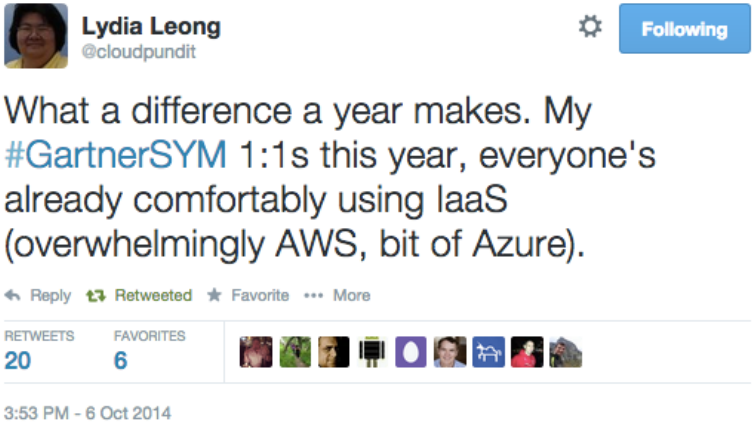


This is the year that Enterprises finally embraced cloud.

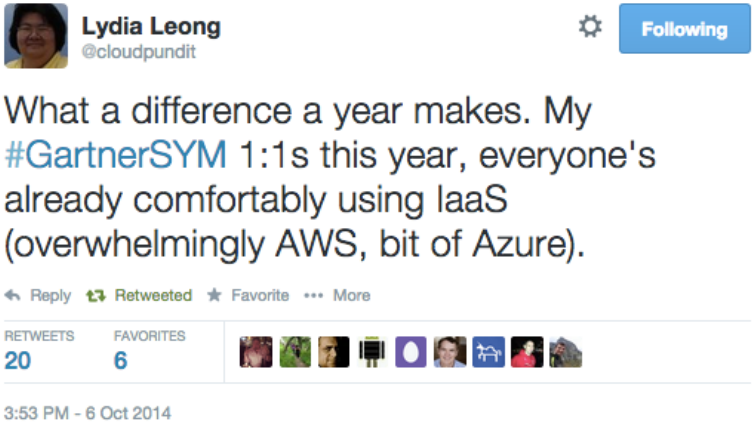




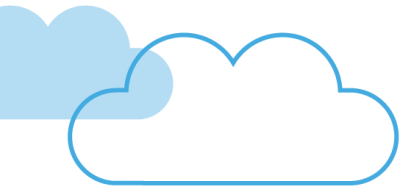
This is the year that Enterprises finally embraced cloud.



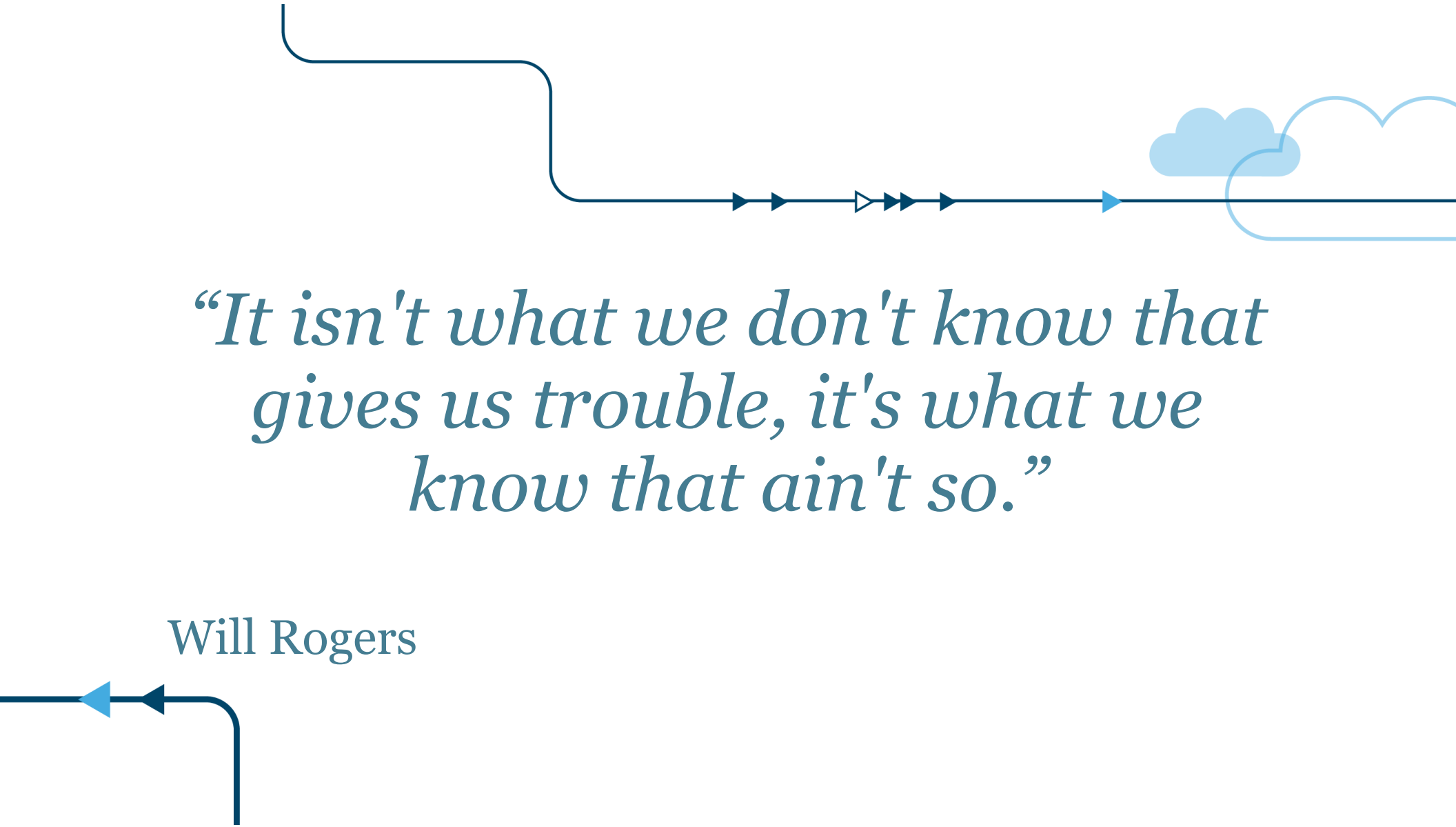
This is the year that Enterprises finally embraced cloud.



This is the year that Enterprises finally embraced cloud.

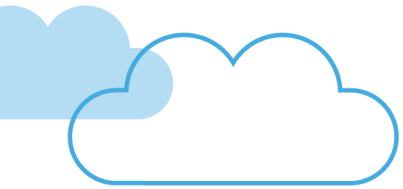



What separates incumbents from disruptors?

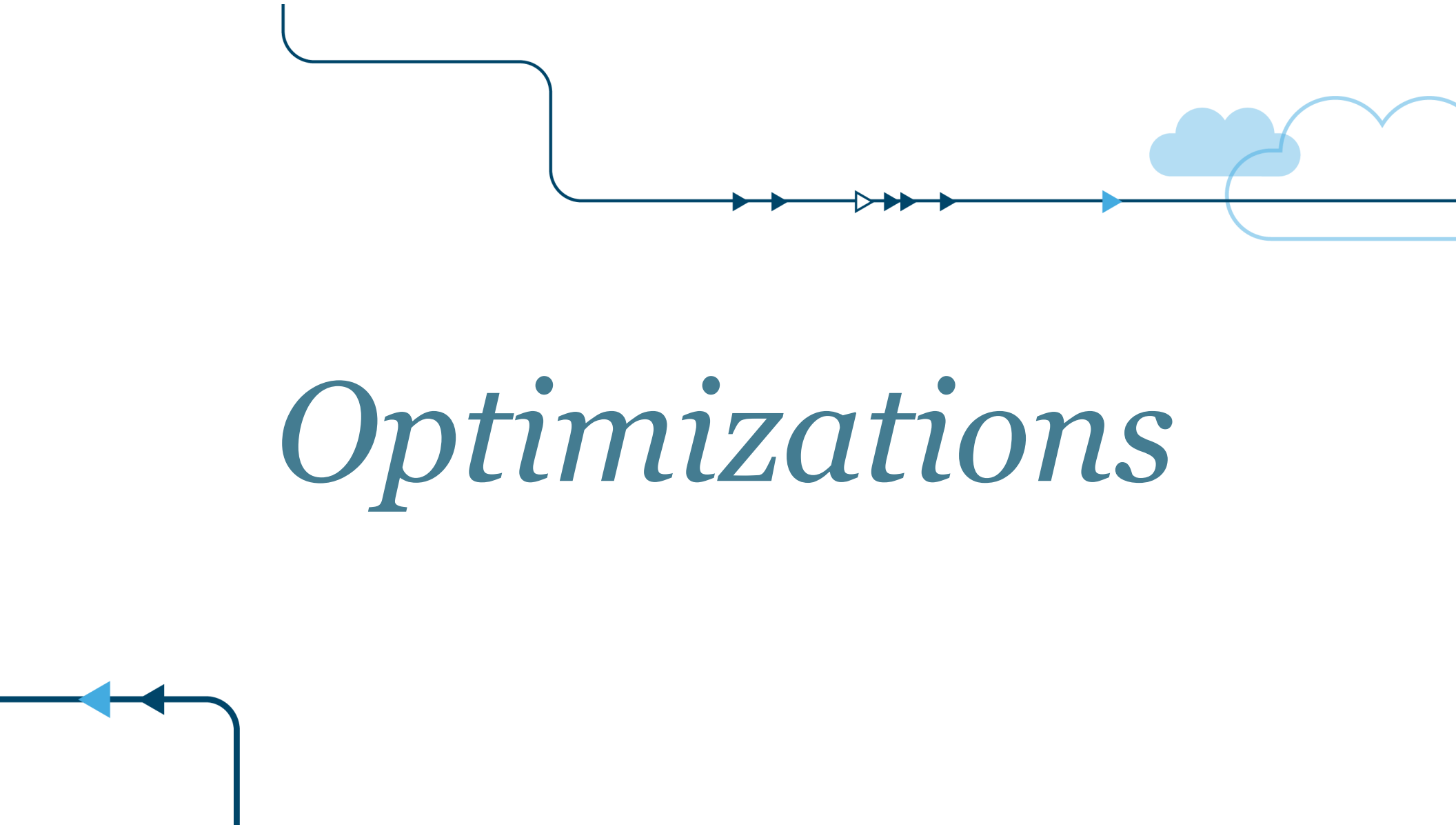


“It isn't what we don't know that gives us trouble, it's what we know that ain't so.”

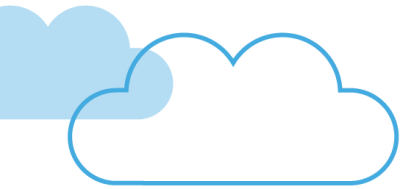
Will Rogers



Assumptions



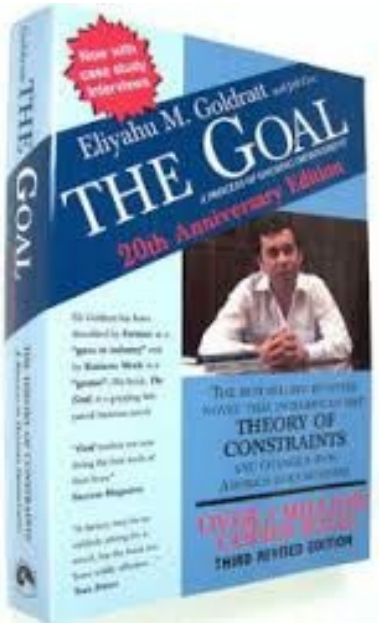
Optimizations




Assumption: Process prevents problems

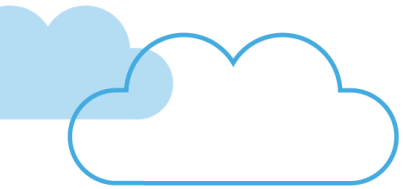
Organizations build up slow complex "Scar tissue" processes

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

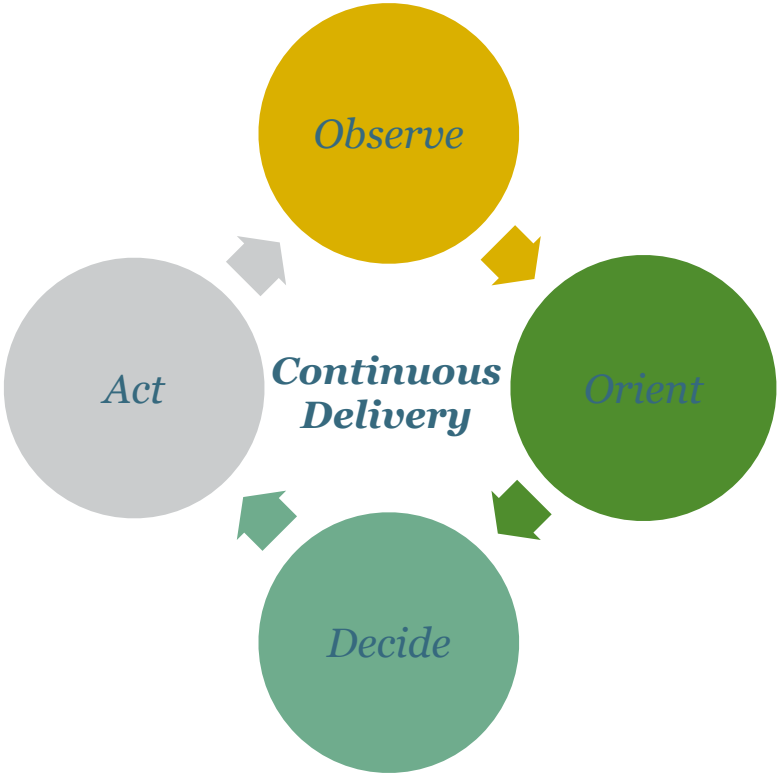


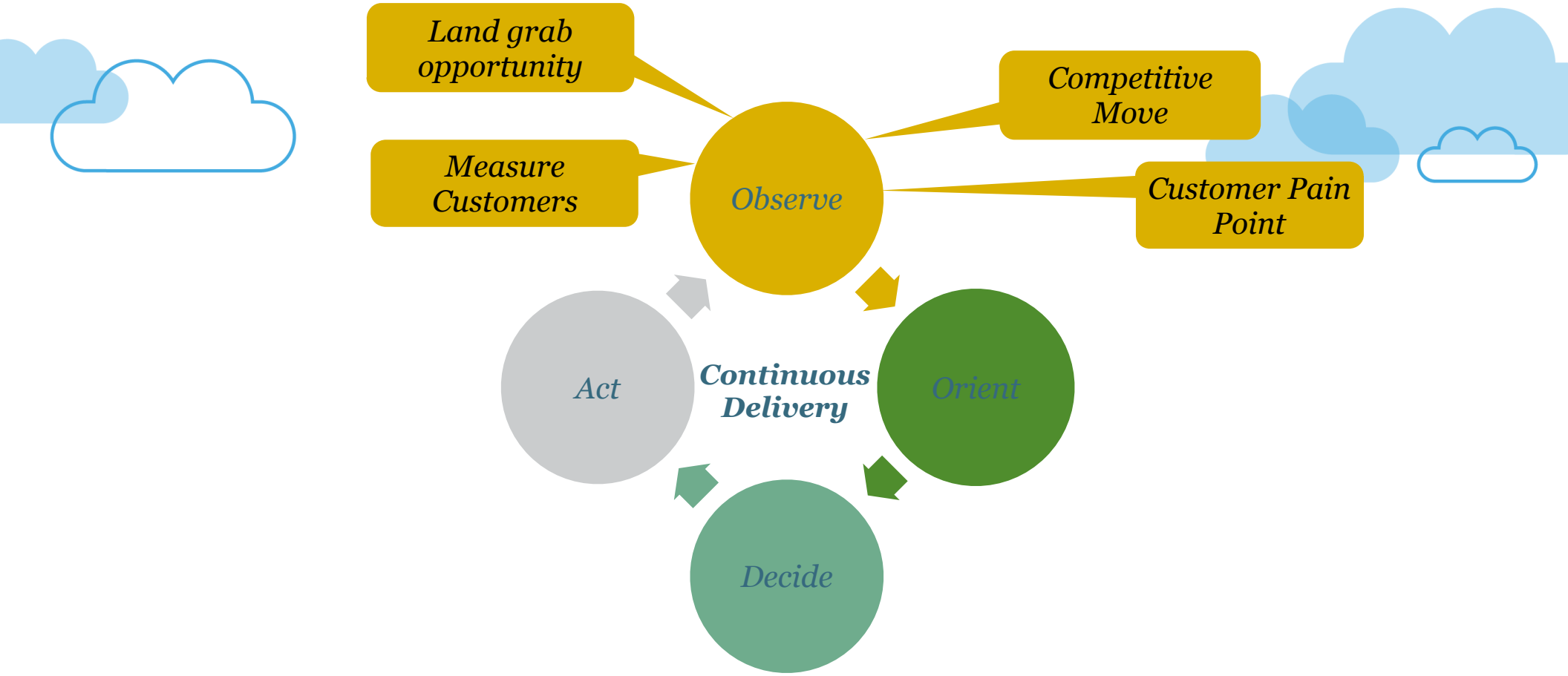
1984

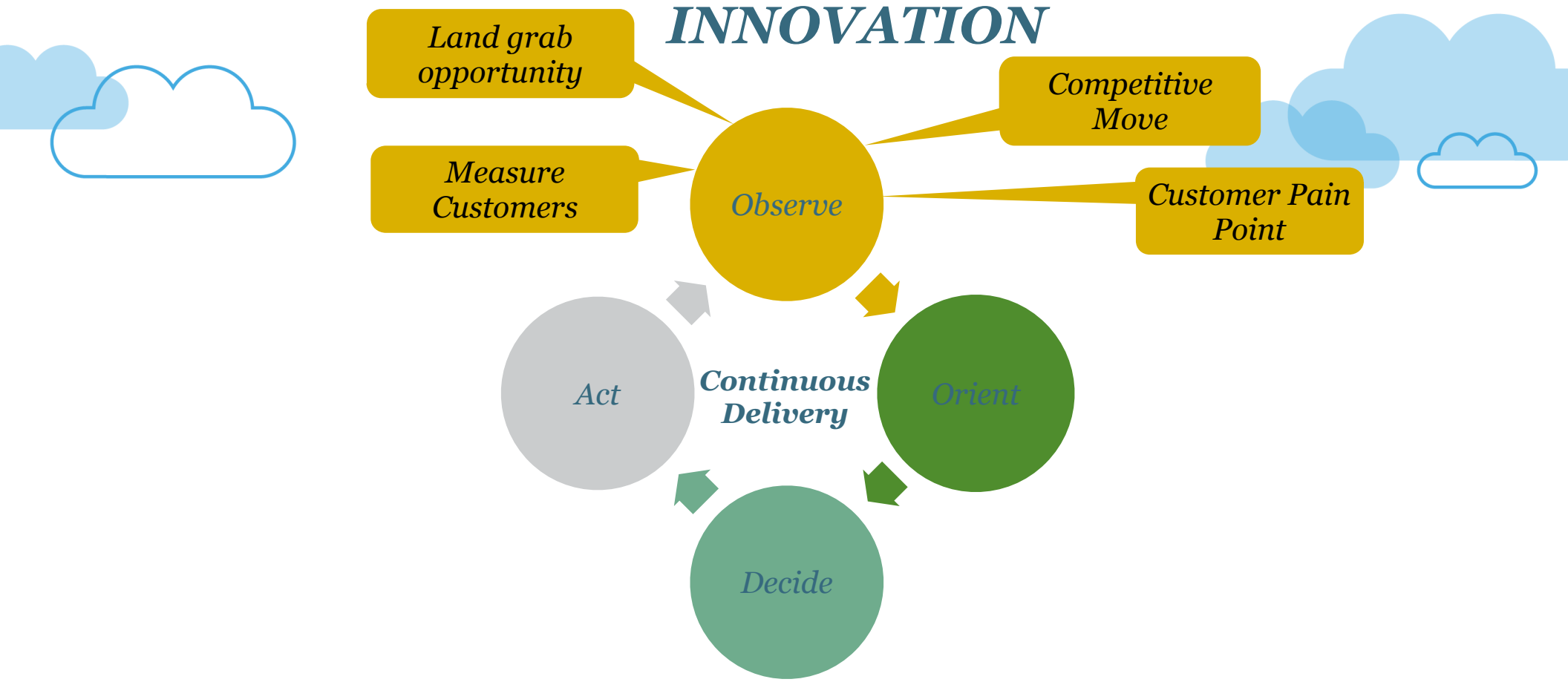
2014

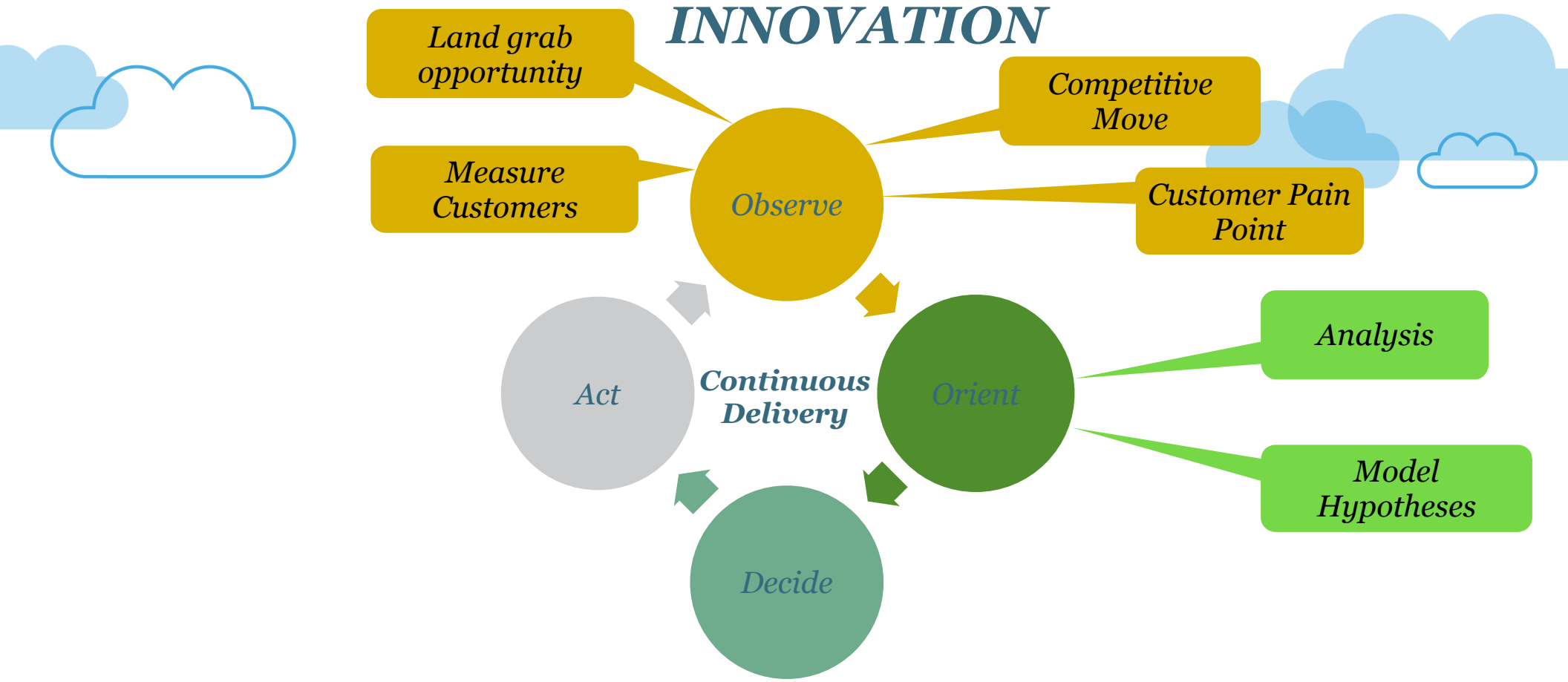


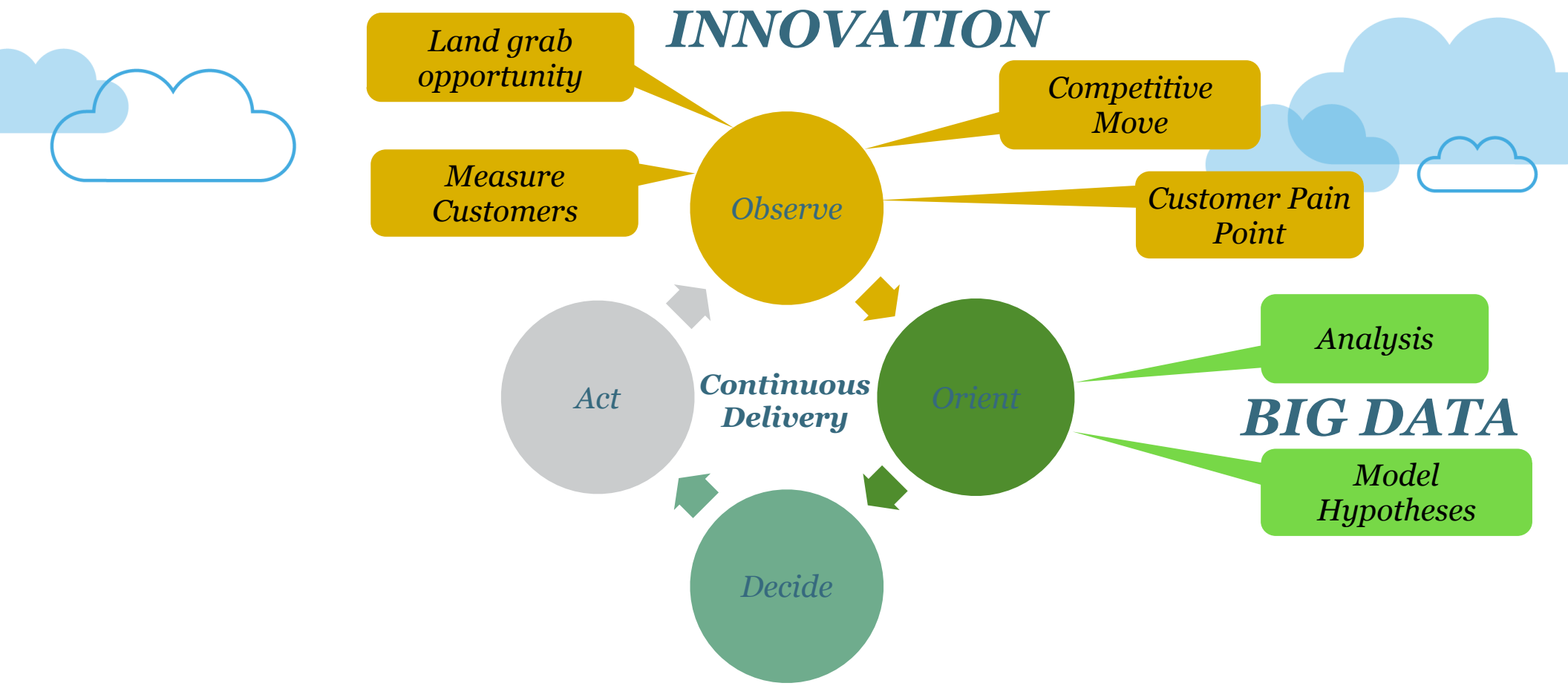
Product Development Processes

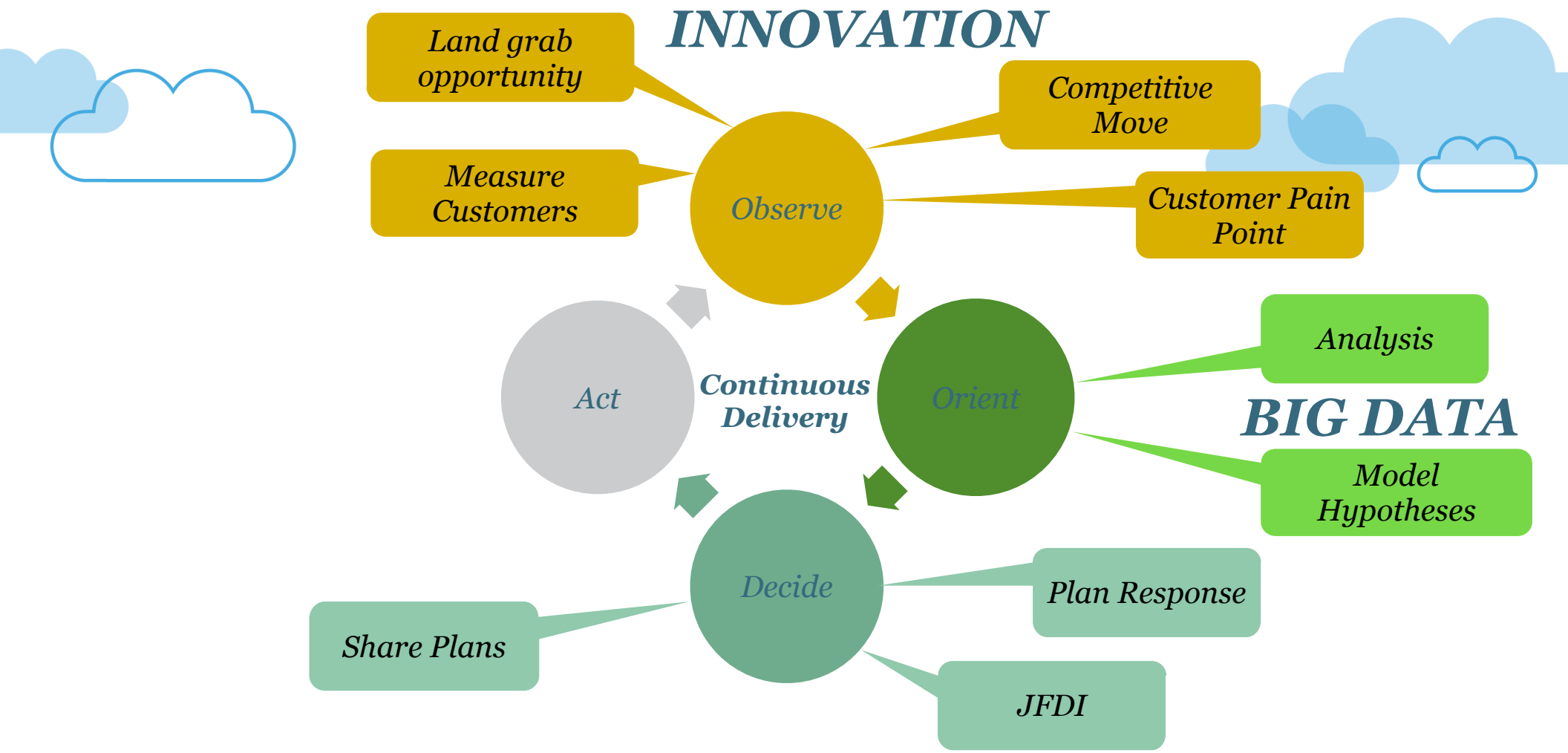

Observe
Orient
Decide
Act
Continuous Delivery

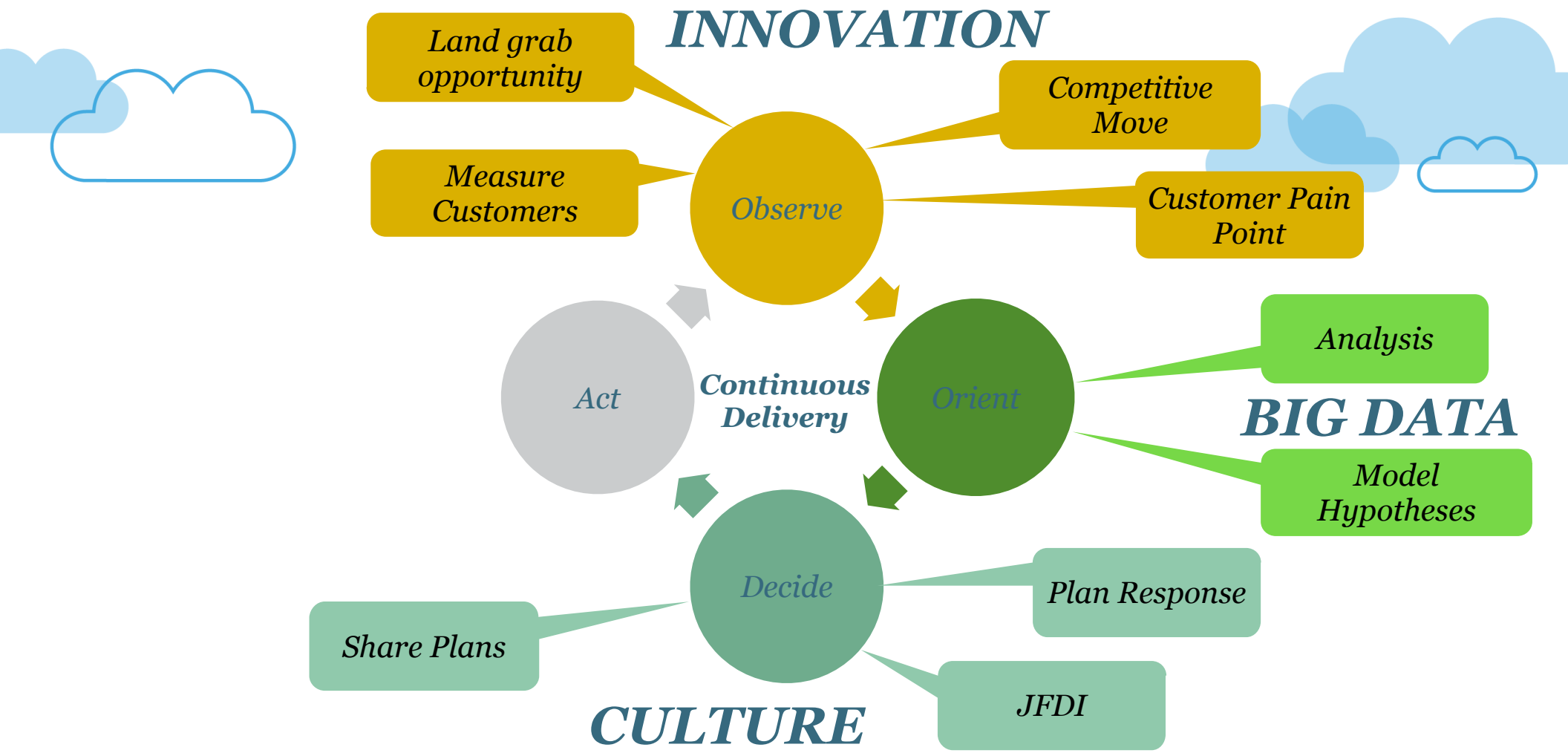

Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Act
Continuous Delivery
Orient
Decide

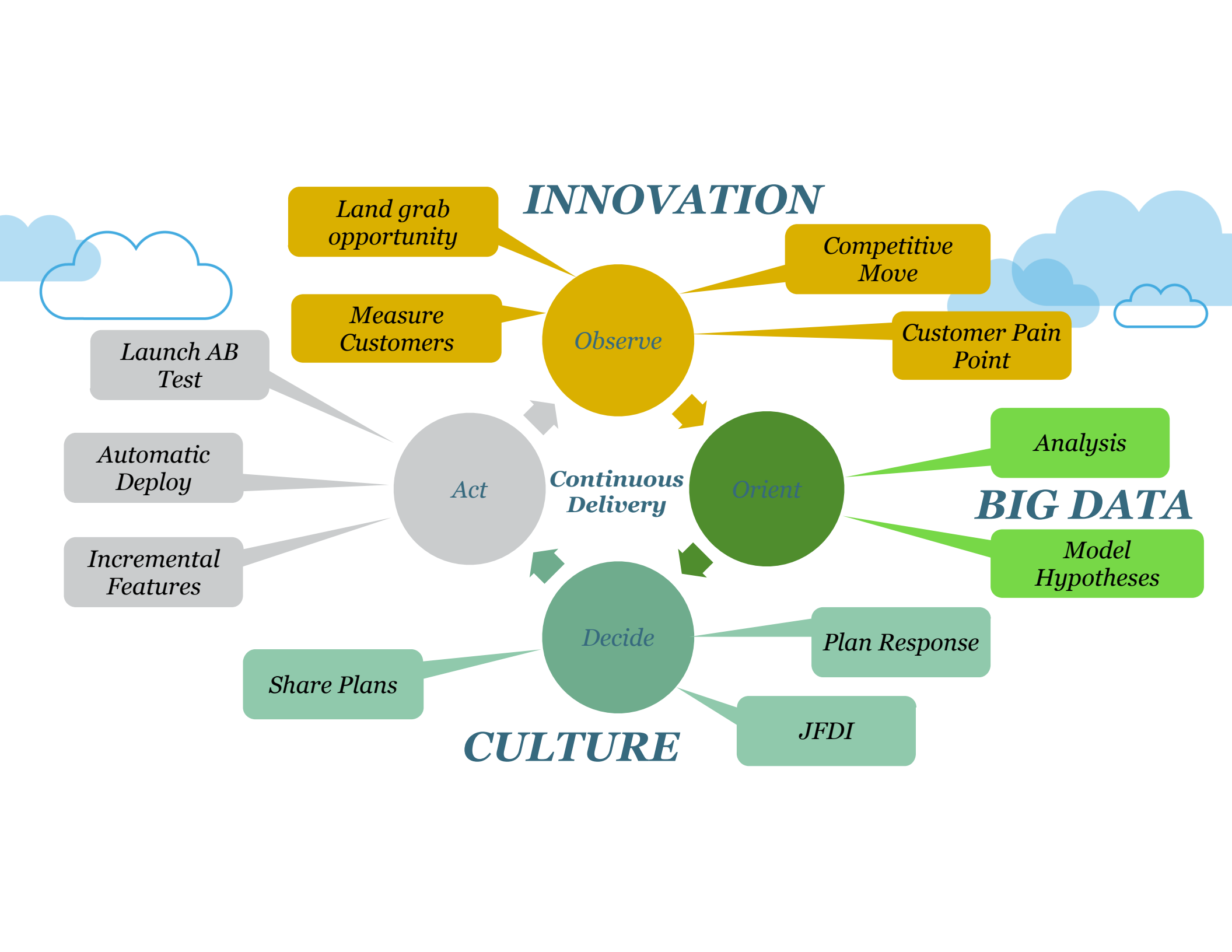

INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Orient
Decide
Act
Continuous Delivery

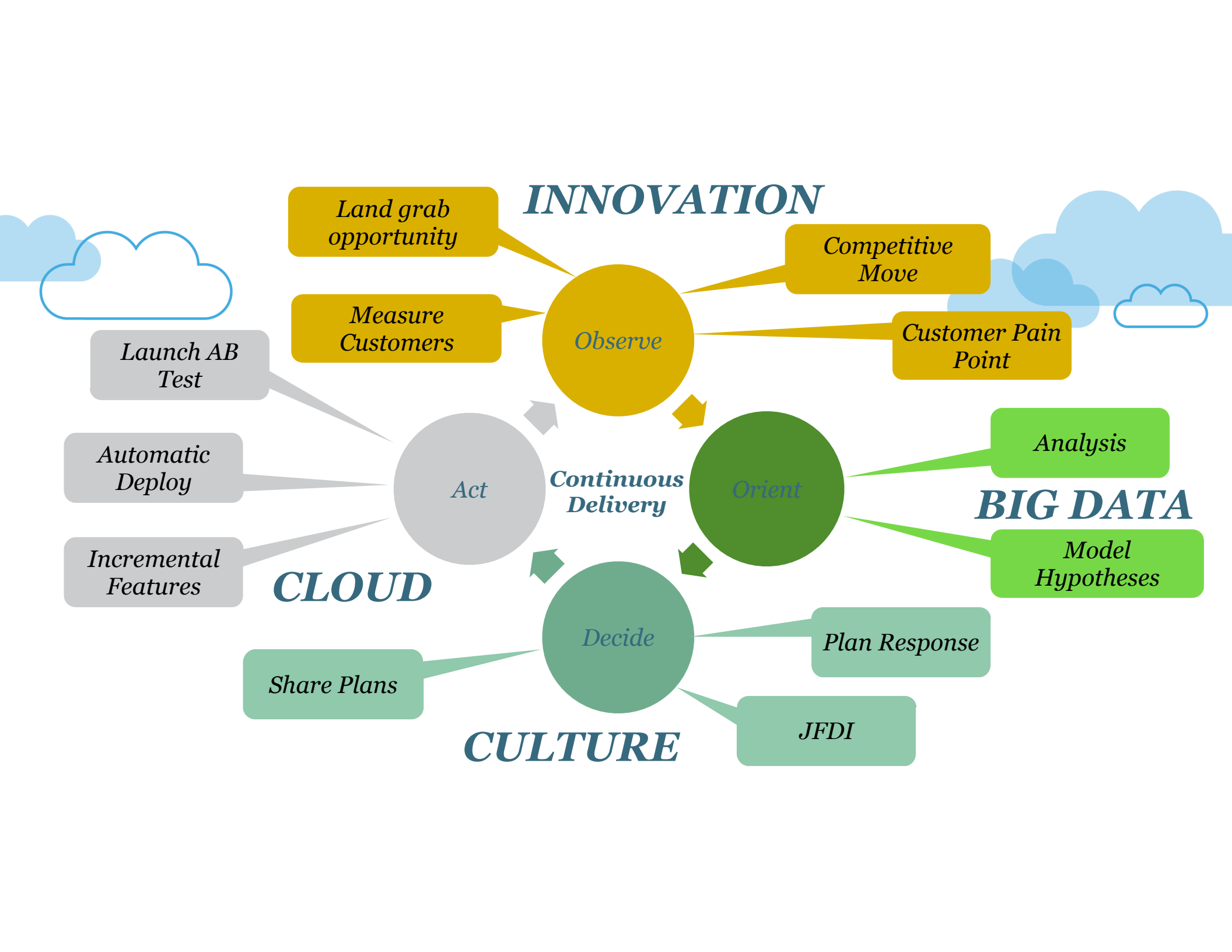

INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Orient
Analysis
Model Hypotheses
Decide
Act
Continuous Delivery

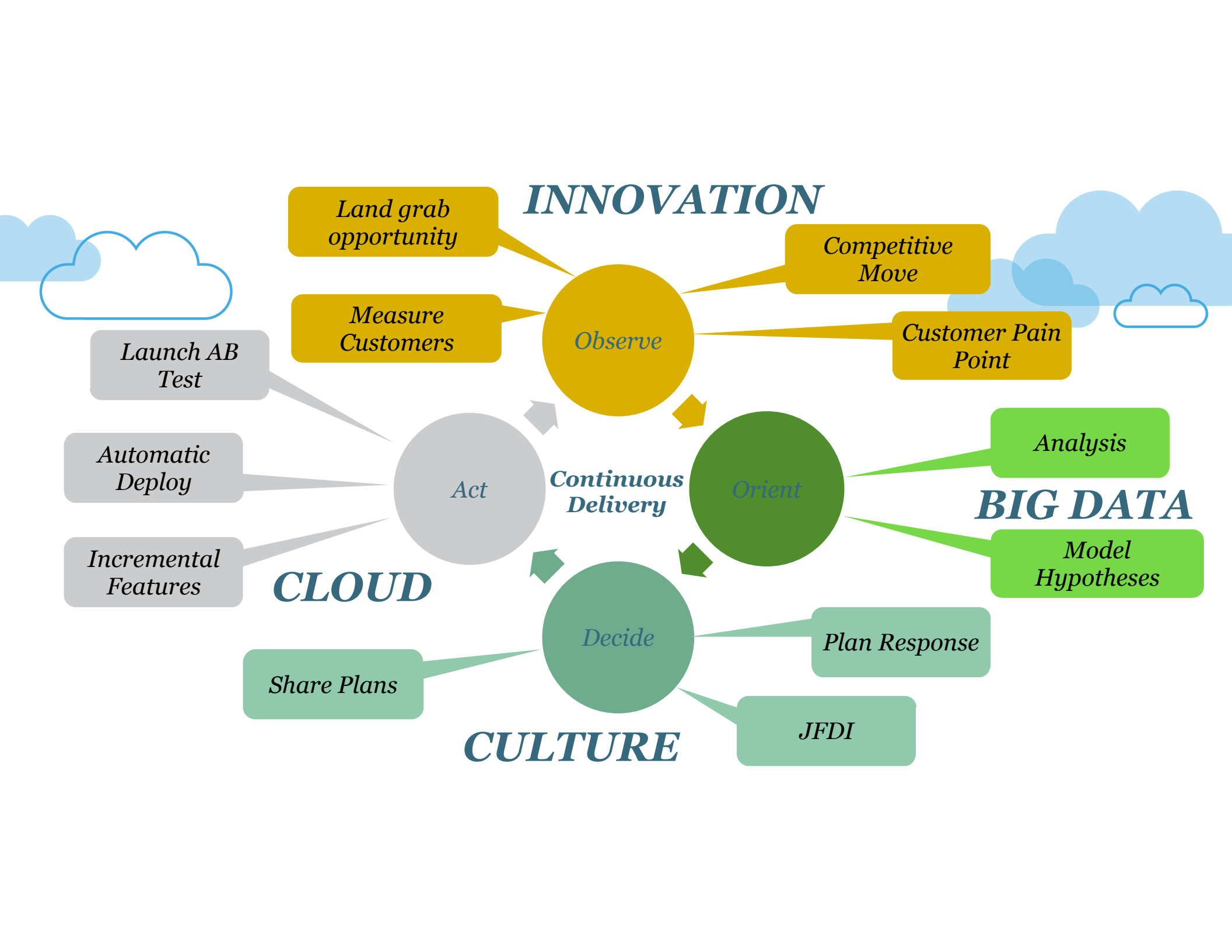

INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Orient
Analysis
BIG DATA
Model Hypotheses
Decide
Act
Continuous Delivery

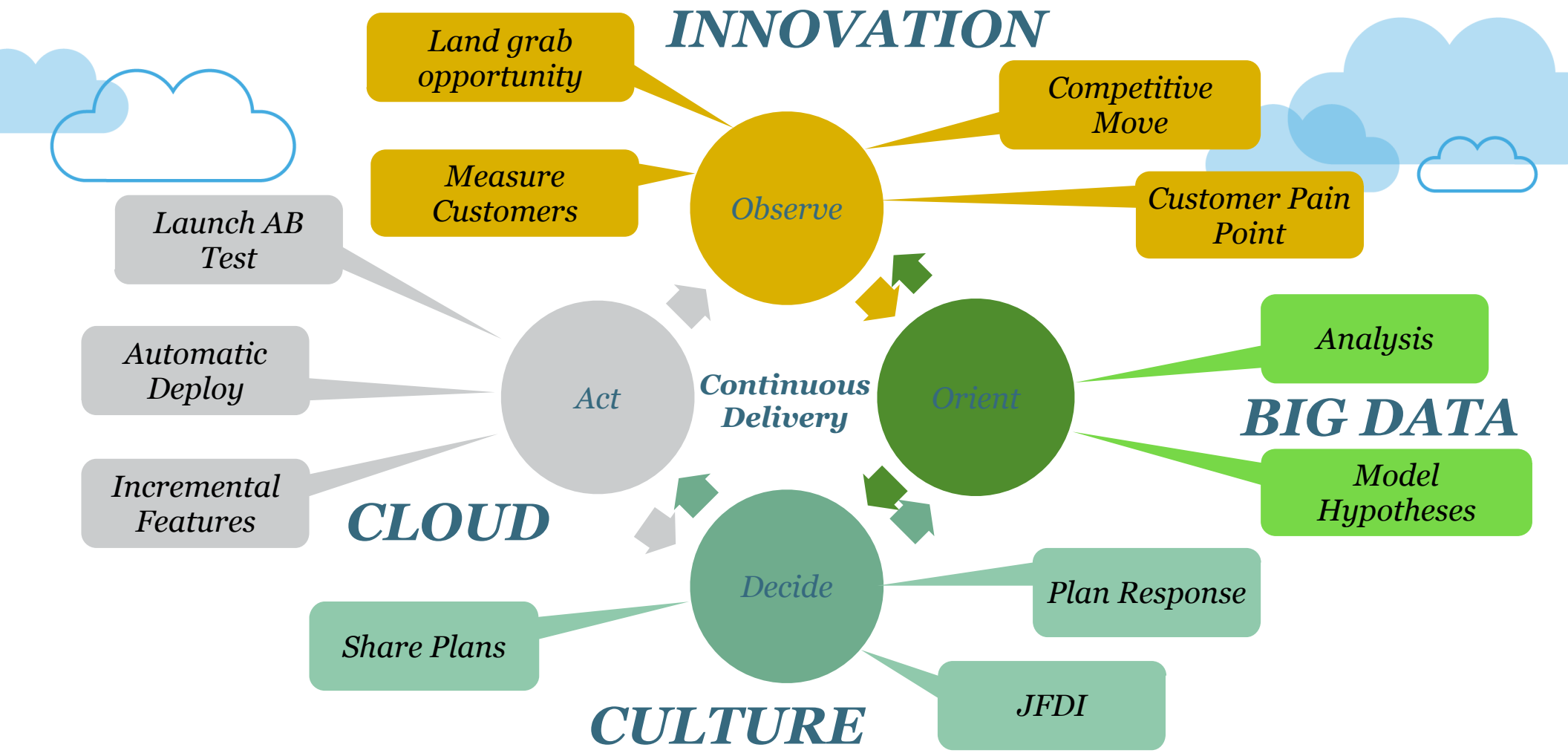

INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Act
Continuous Delivery
Orient
Analysis
BIG DATA
Model Hypotheses
Decide
Share Plans
Plan Response
JFDI


INNOVATION
Land grab opportunity
Competitive Move
Measure Customers
Observe
Customer Pain Point
Act
Continuous Delivery
Orient
Analysis
BIG DATA
Model Hypotheses
Decide
Plan Response
Share Plans
JFDI
CULTURE


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
Decide
Share Plans
Plan Response
JFDI
CULTURE

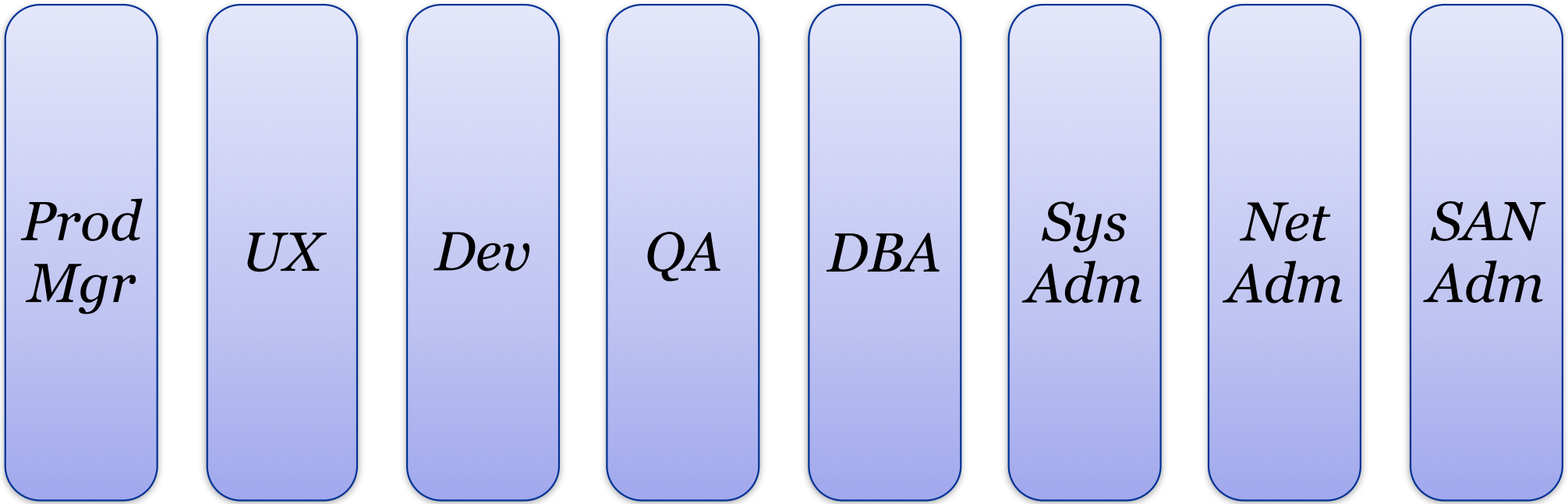

INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Launch AB Test
Automatic Deploy
Incremental Features
Act
Continuous Delivery
Orient
Analysis
BIG DATA
Model Hypotheses
CLOUD
Decide
Plan Response
Share Plans
JFDI
CULTURE


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

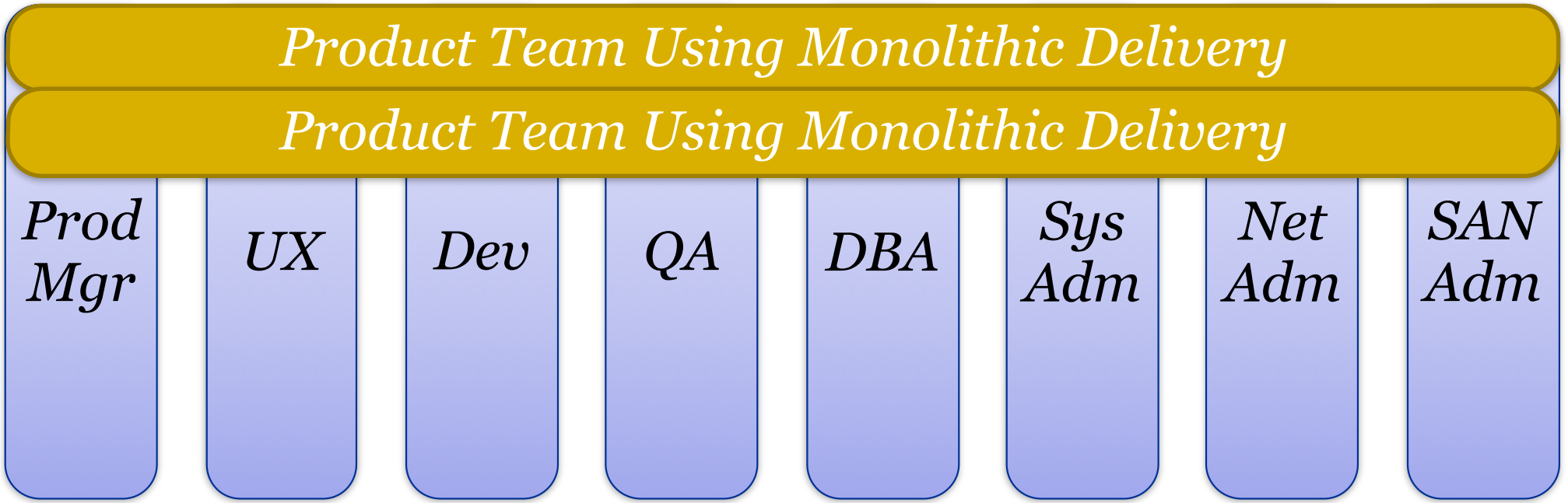

INNOVATION
Land grab opportunity
Measure Customers
Observe
Competitive Move
Customer Pain Point
Launch AB Test
Automatic Deploy
Incremental Features
Act
Continuous Delivery
Orient
Analysis
BIG DATA
Model Hypotheses
CLOUD
Decide
Plan Response
Share Plans
JFDI
CULTURE

Breaking Down the SILOs

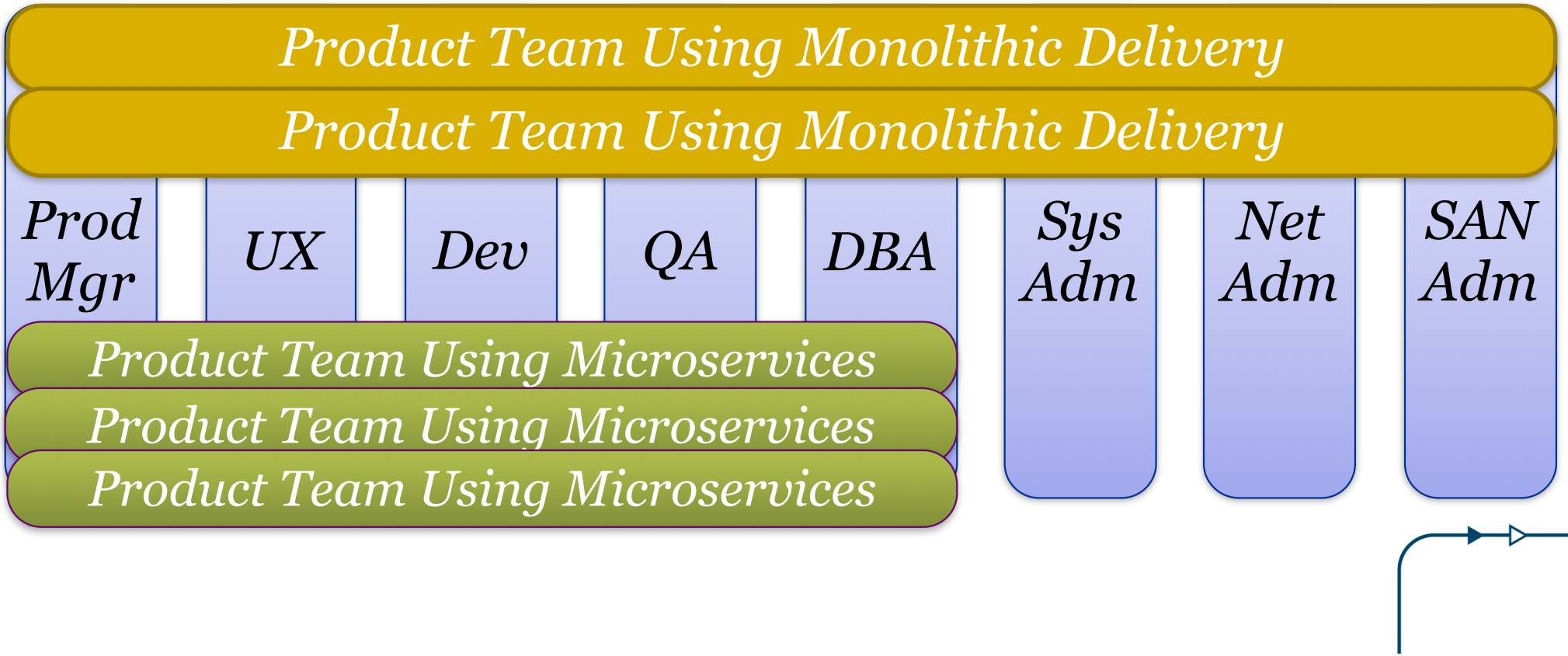
Breaking Down the SILOs



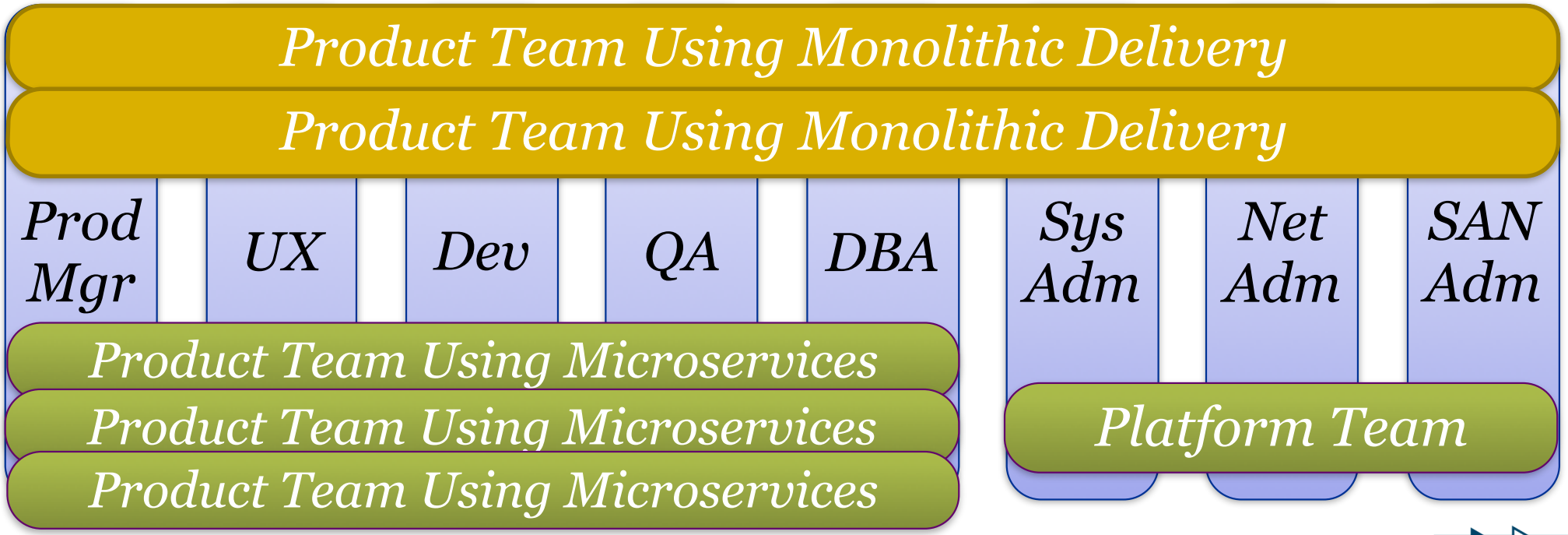
Breaking Down the SILOs



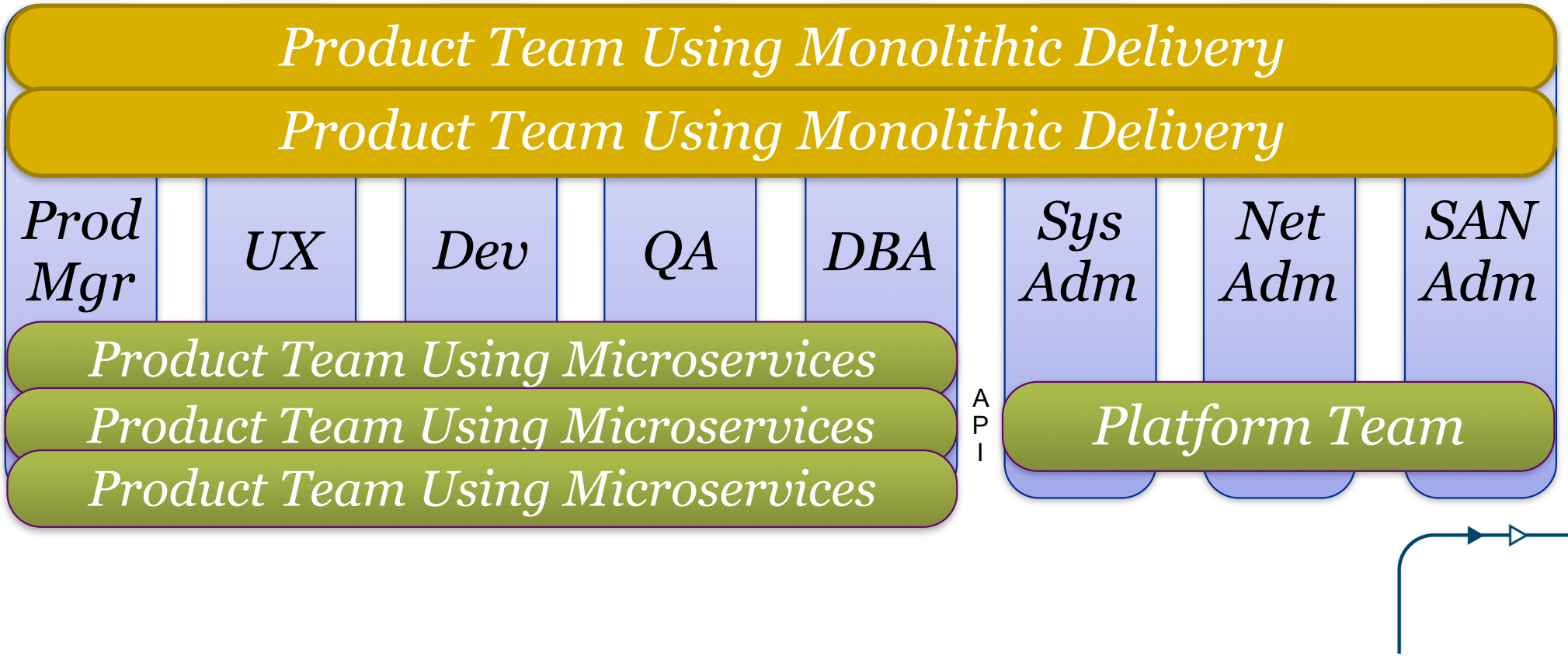
Breaking Down the SILOs



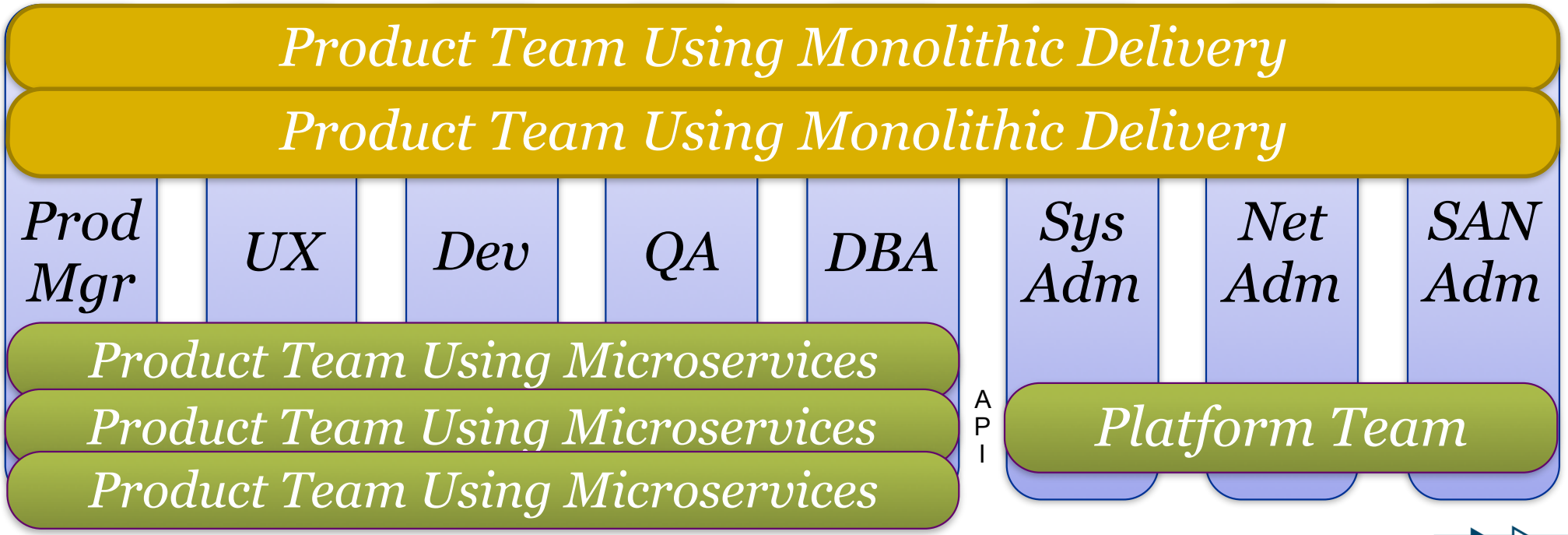
Breaking Down the SILOs



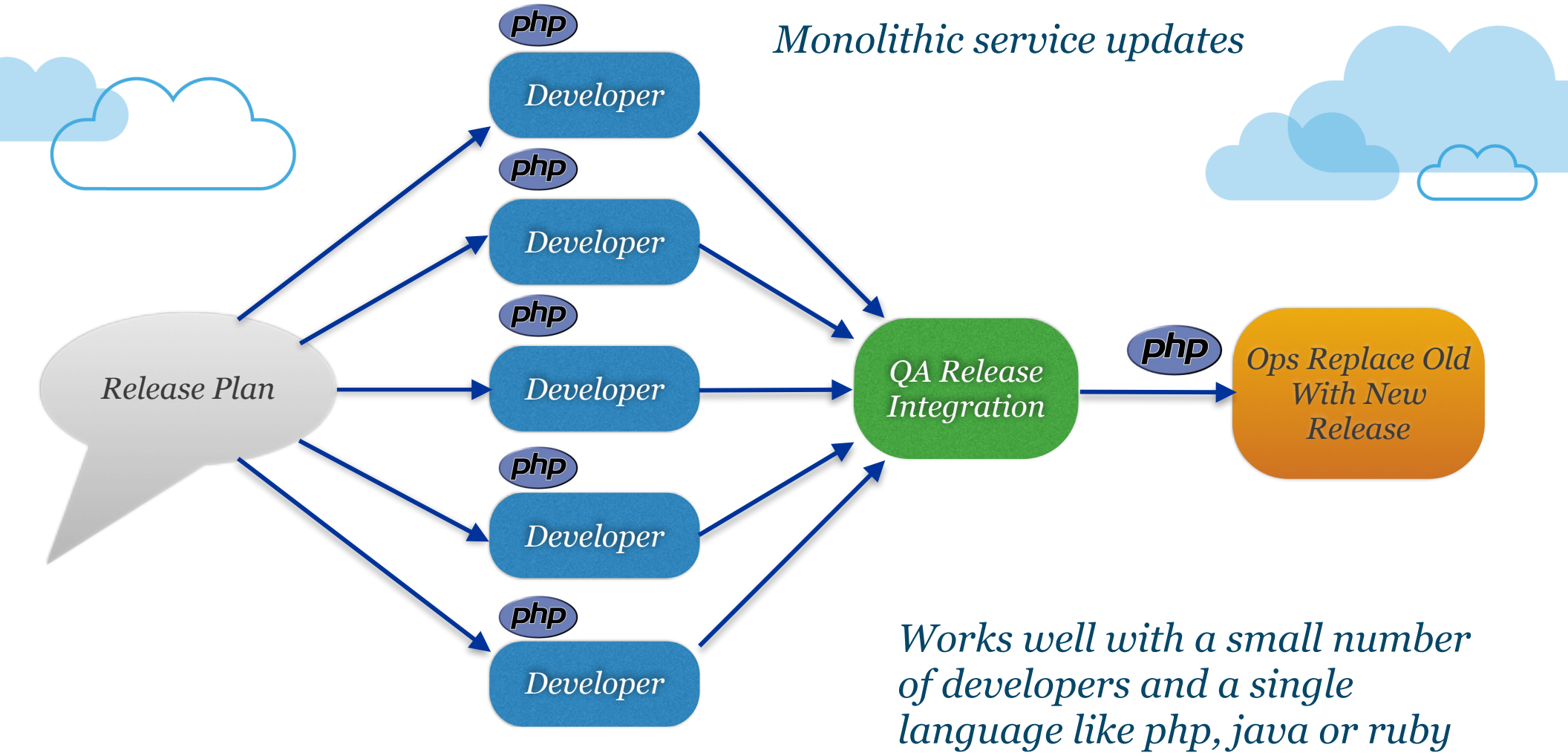
Breaking Down the SILOs

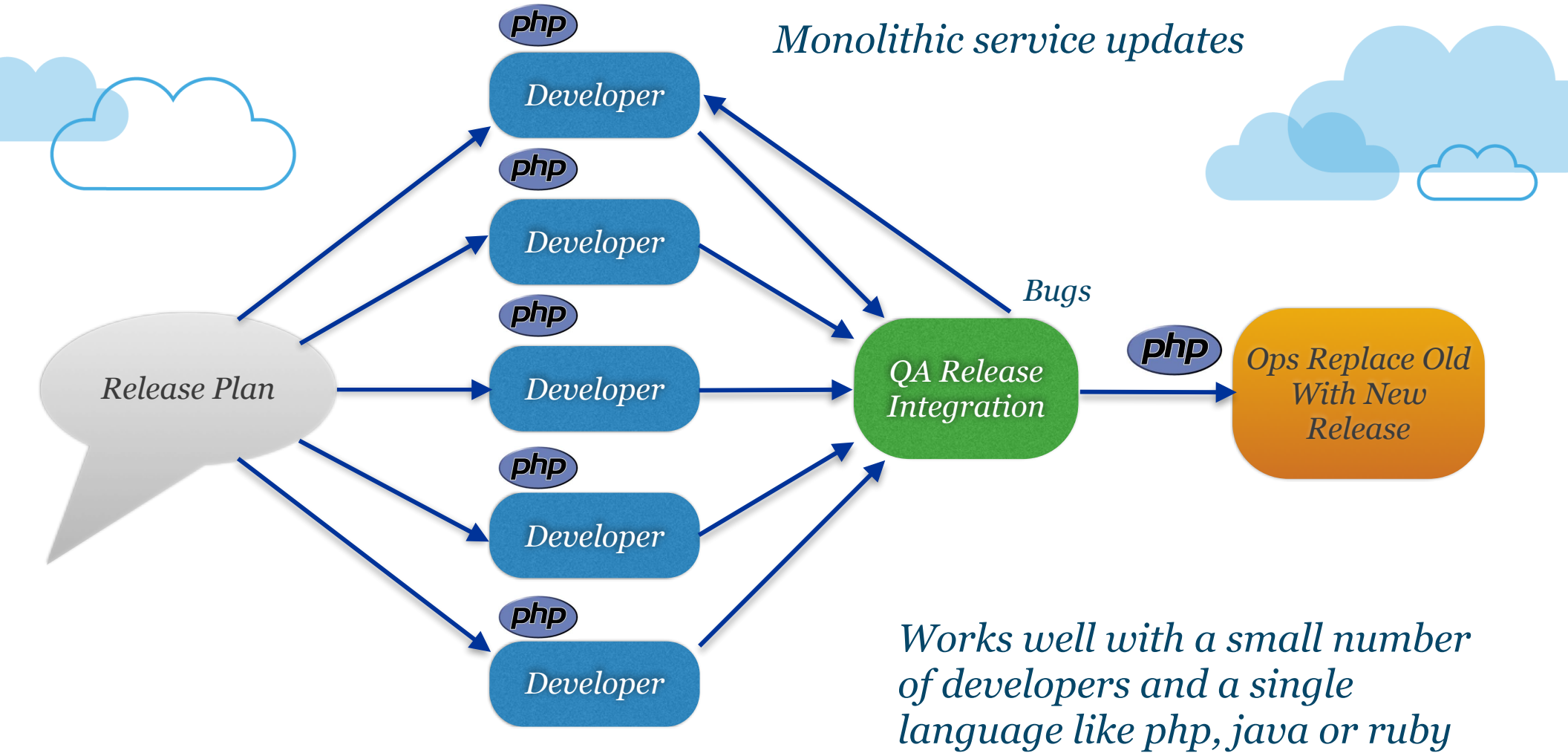


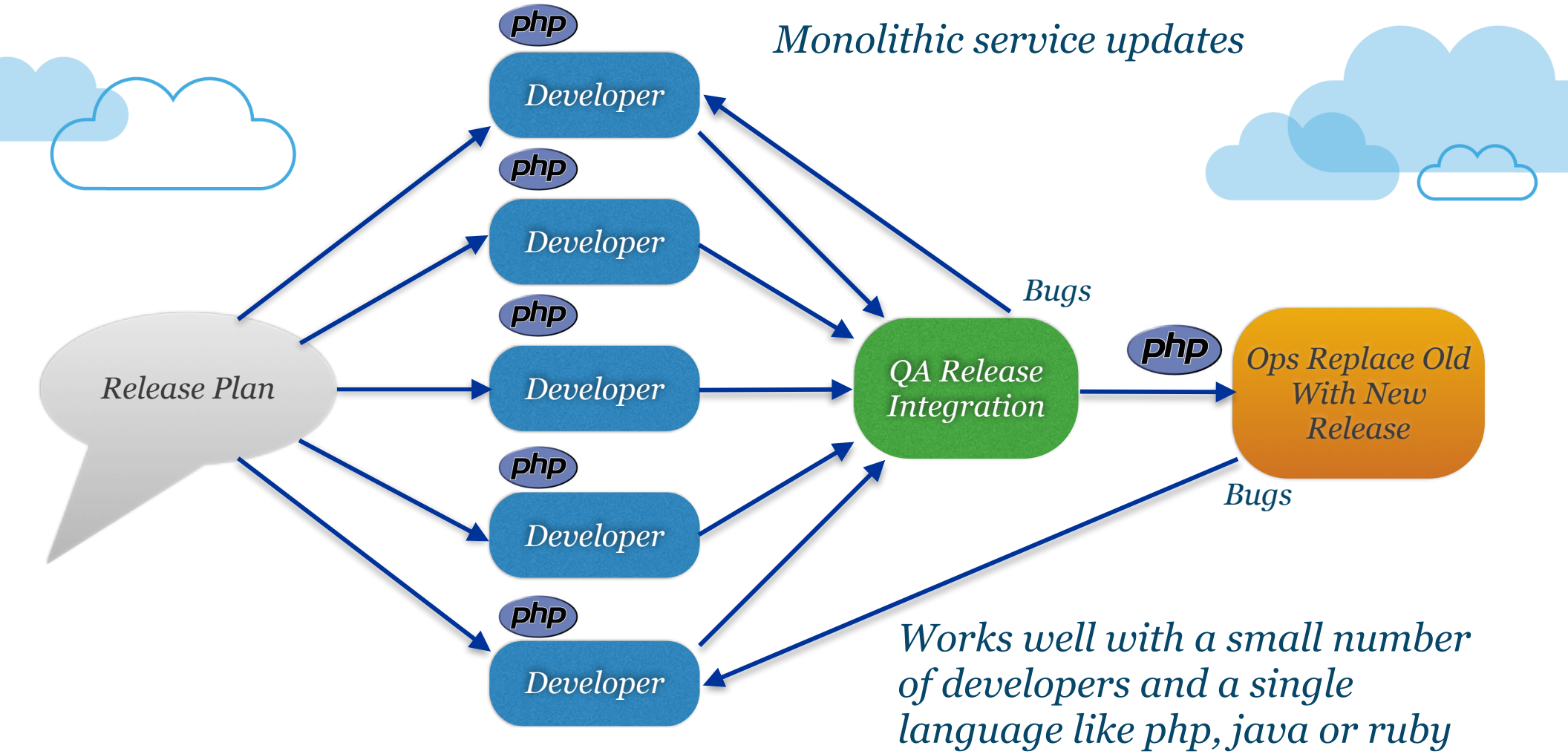
Breaking Down the SILOs

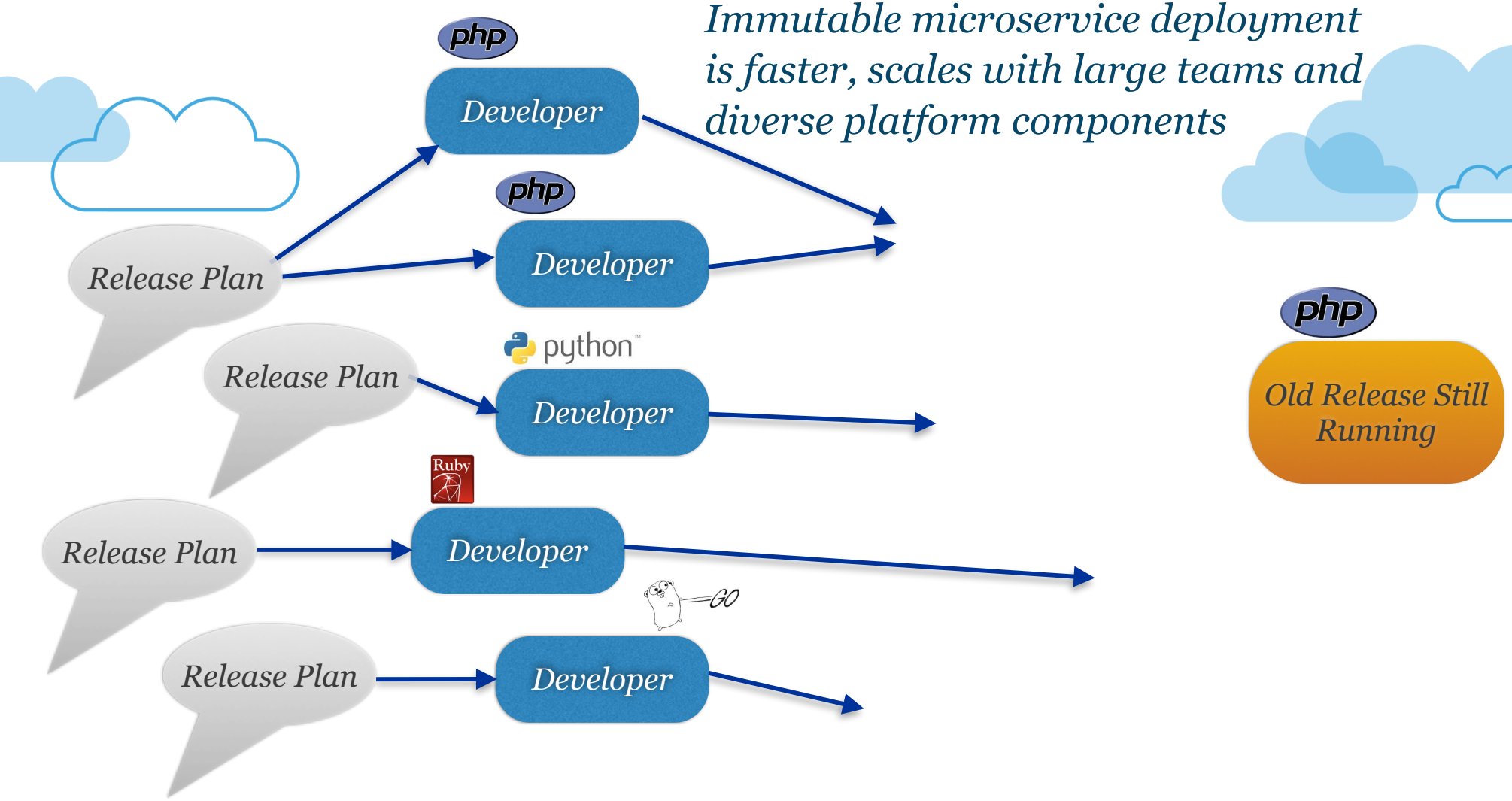


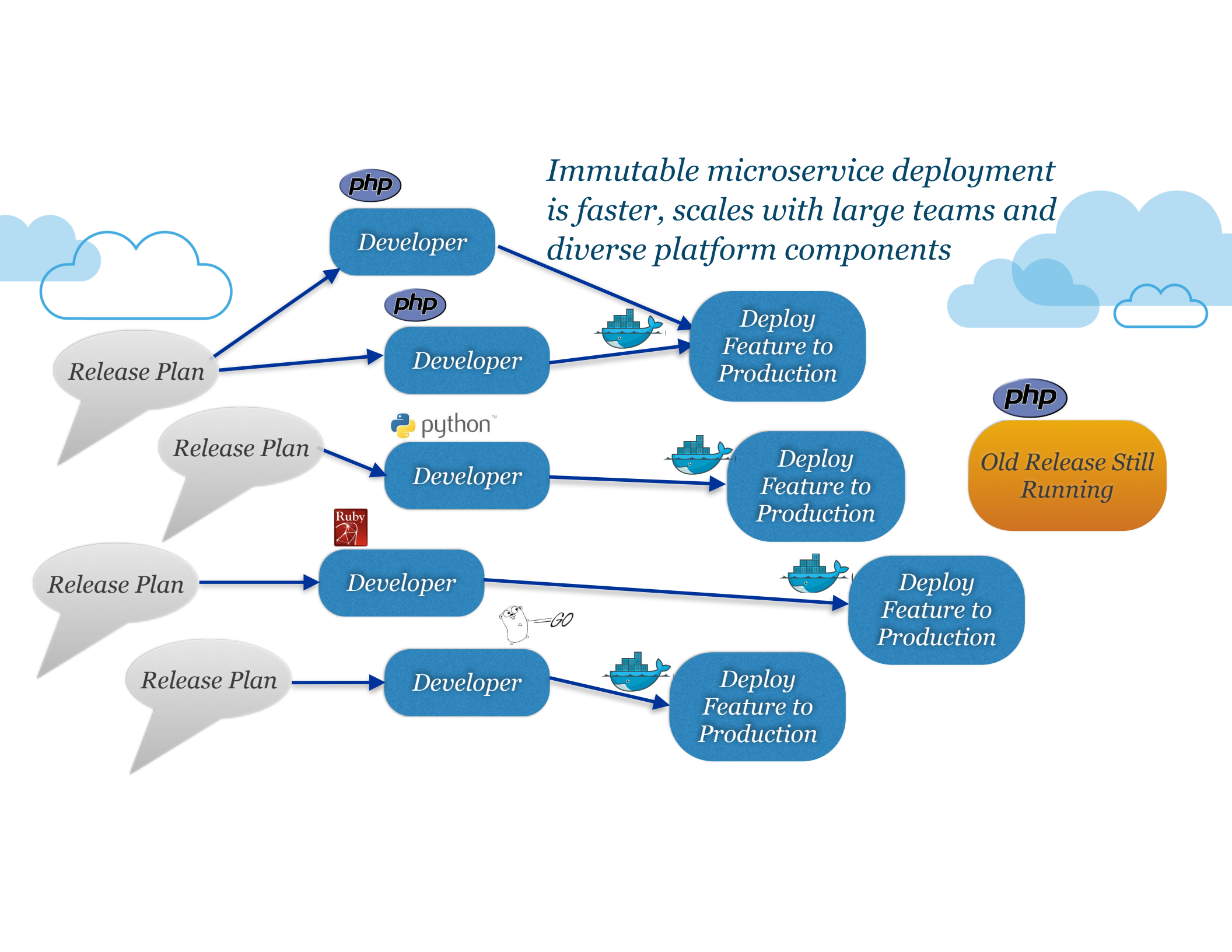
DevOps is a Re-Org

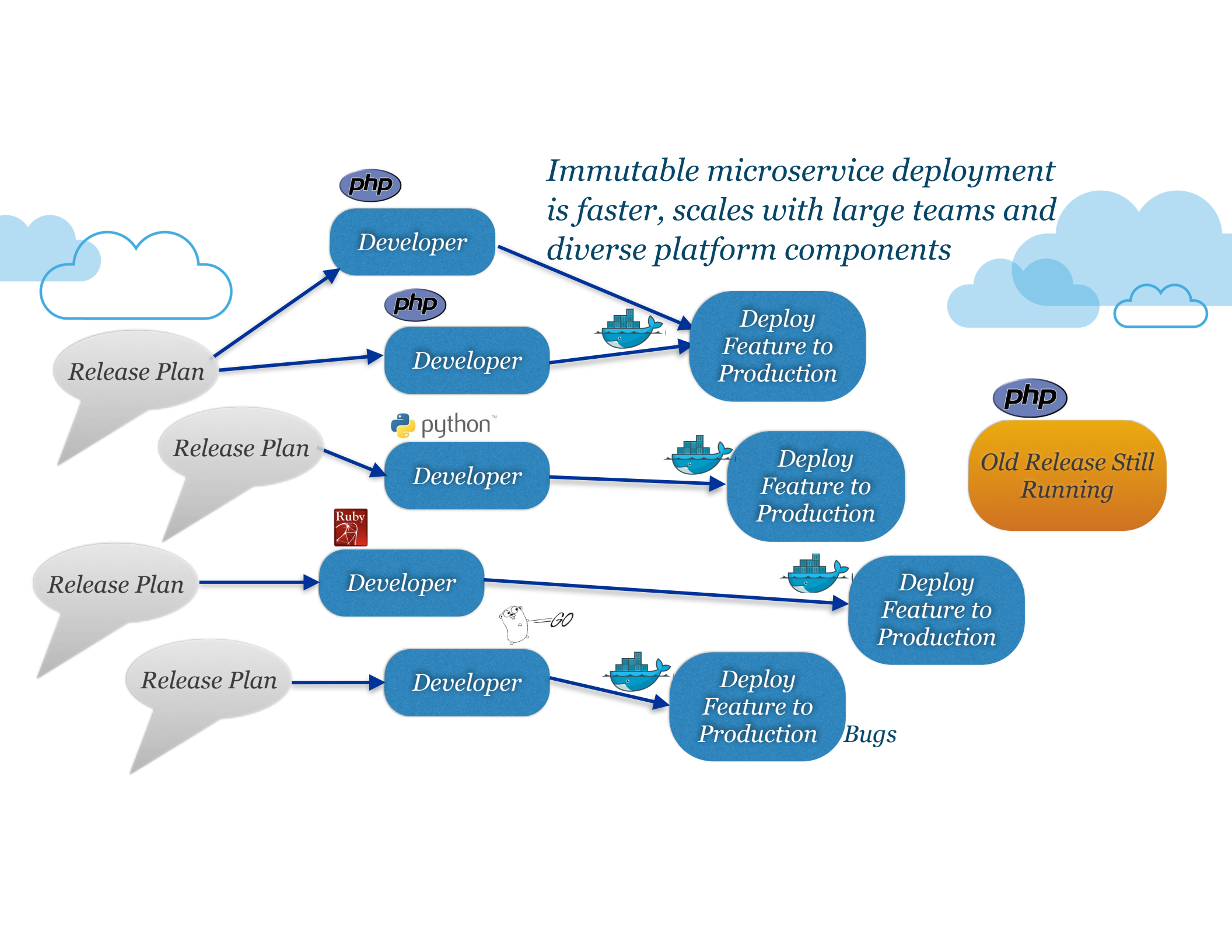

Monolithic service updates
php
Developer
php
Developer
php
Developer
php
Developer
php
Developer
Release Plan
QA Release Integration
php
Ops Replace Old With New Release
Works well with a small number of developers and a single language like php, java or ruby

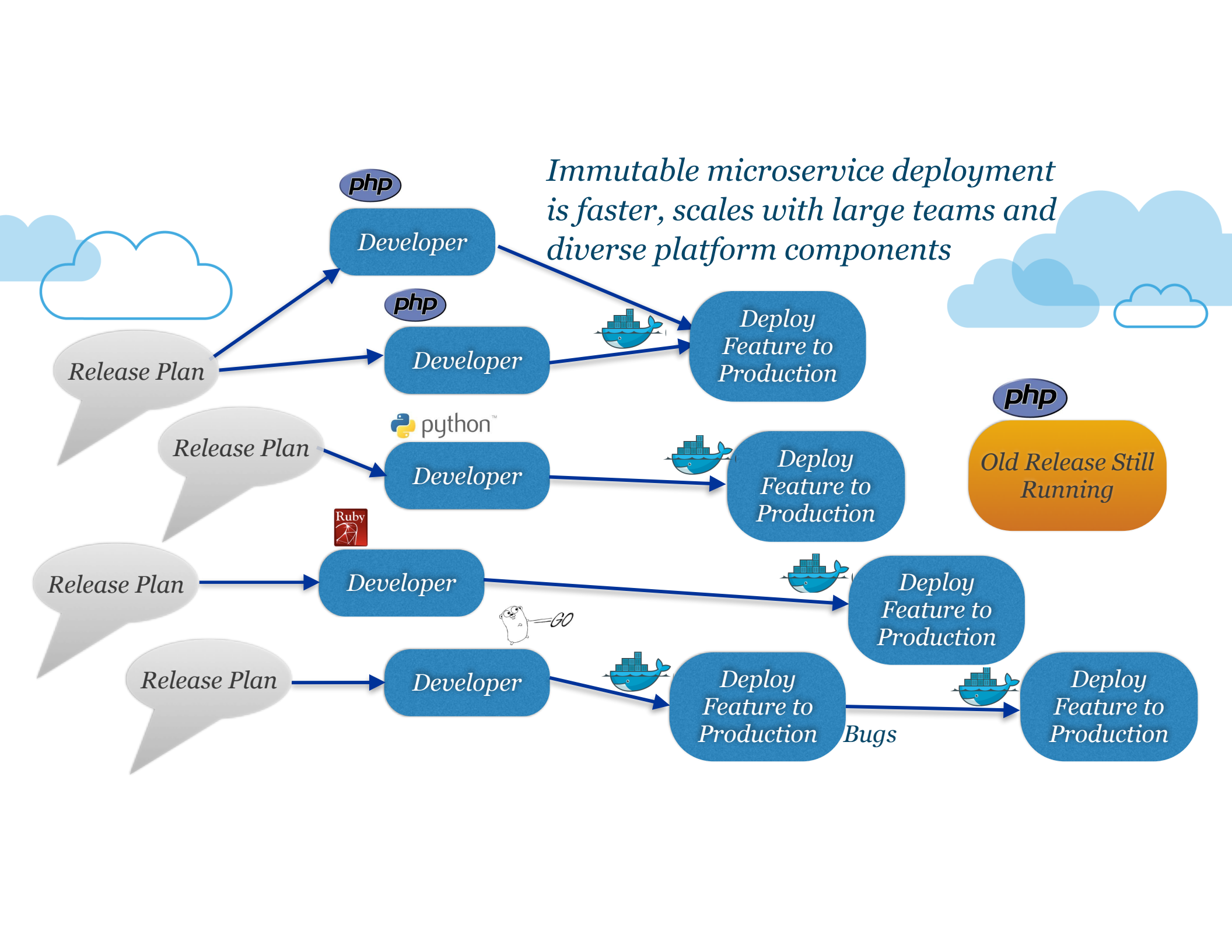

Monolithic service updates
Release Plan
Developer
Developer
Developer
Developer
Developer
Bugs
QA Release Integration
Ops Replace Old With New Release
Works well with a small number of developers and a single language like php, java or ruby


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


Immutable microservice deployment is faster, scales with large teams and diverse platform components
php
Developer
Release Plan
php
Developer
Release Plan
python
Developer
Ruby
Release Plan
Developer
GO
Release Plan
Developer
php
Old Release Still Running

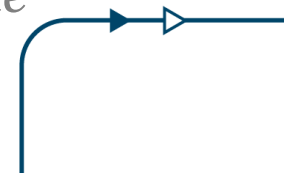

Immutable microservice deployment is faster, scales with large teams and diverse platform components
php
Developer
Release Plan
php
Developer
Deploy Feature to Production
python
Release Plan
Developer
Deploy Feature to Production
php
Old Release Still Running
Ruby
Release Plan
Developer
Deploy Feature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


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


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

Non-Destructive Production Updates

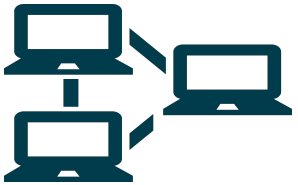
- *"Immutable Code" Service Pattern*
  - *Existing services are unchanged, old code remains in service*
  - *New code deploys as a new service group*
  - *No impact to production until traffic routing changes*
- *A|B Tests, Feature Flags and Version Routing control traffic*
  - *First users in the test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



Developing at the Speed of Docker

Developers
- Compile/Build
- Seconds

Extend container
- Package dependencies
- Seconds



PaaS deploy Container
- Docker startup
- Seconds

ERICSSON
apcera

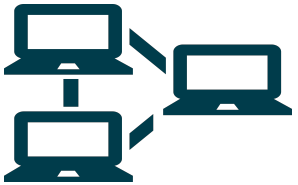
etc…

Developing at the Speed of Docker

Developers
- Compile/Build
- Seconds

docker

Extend container
- Package dependencies
- Seconds



PaaS deploy Container
- Docker startup
- Seconds

CLOUD FOUNDRY

Google App Engine

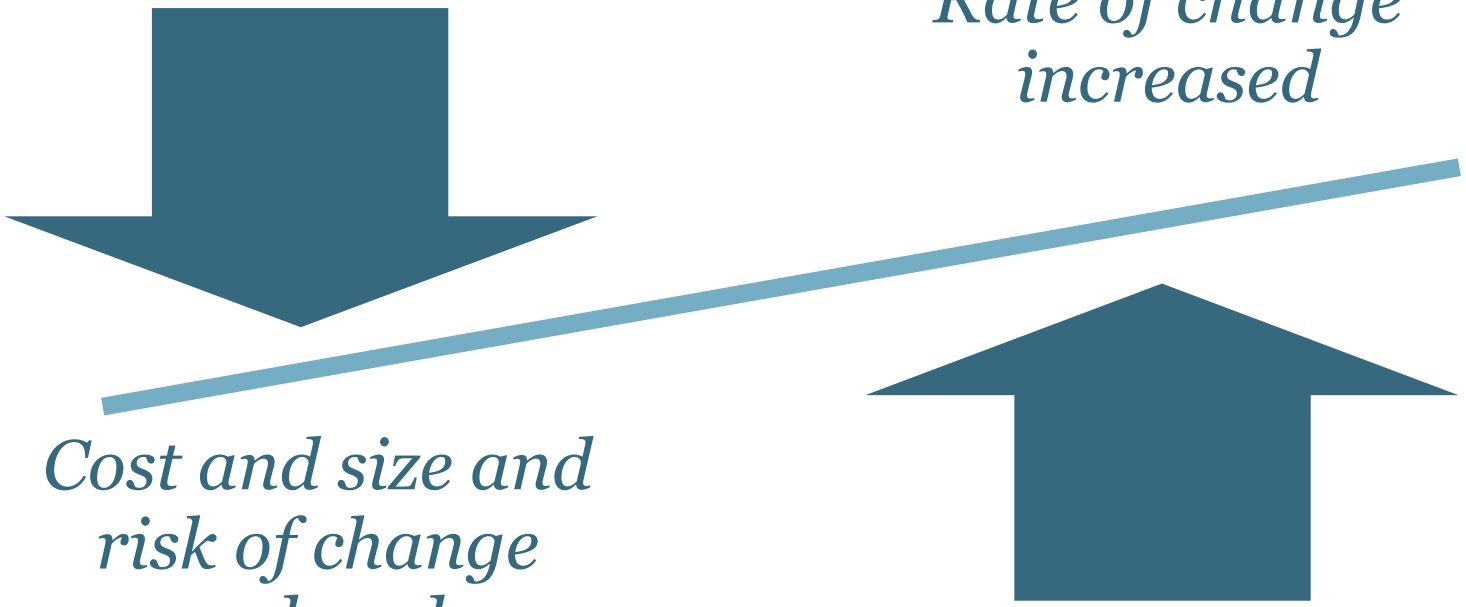
ERICSSON apcera

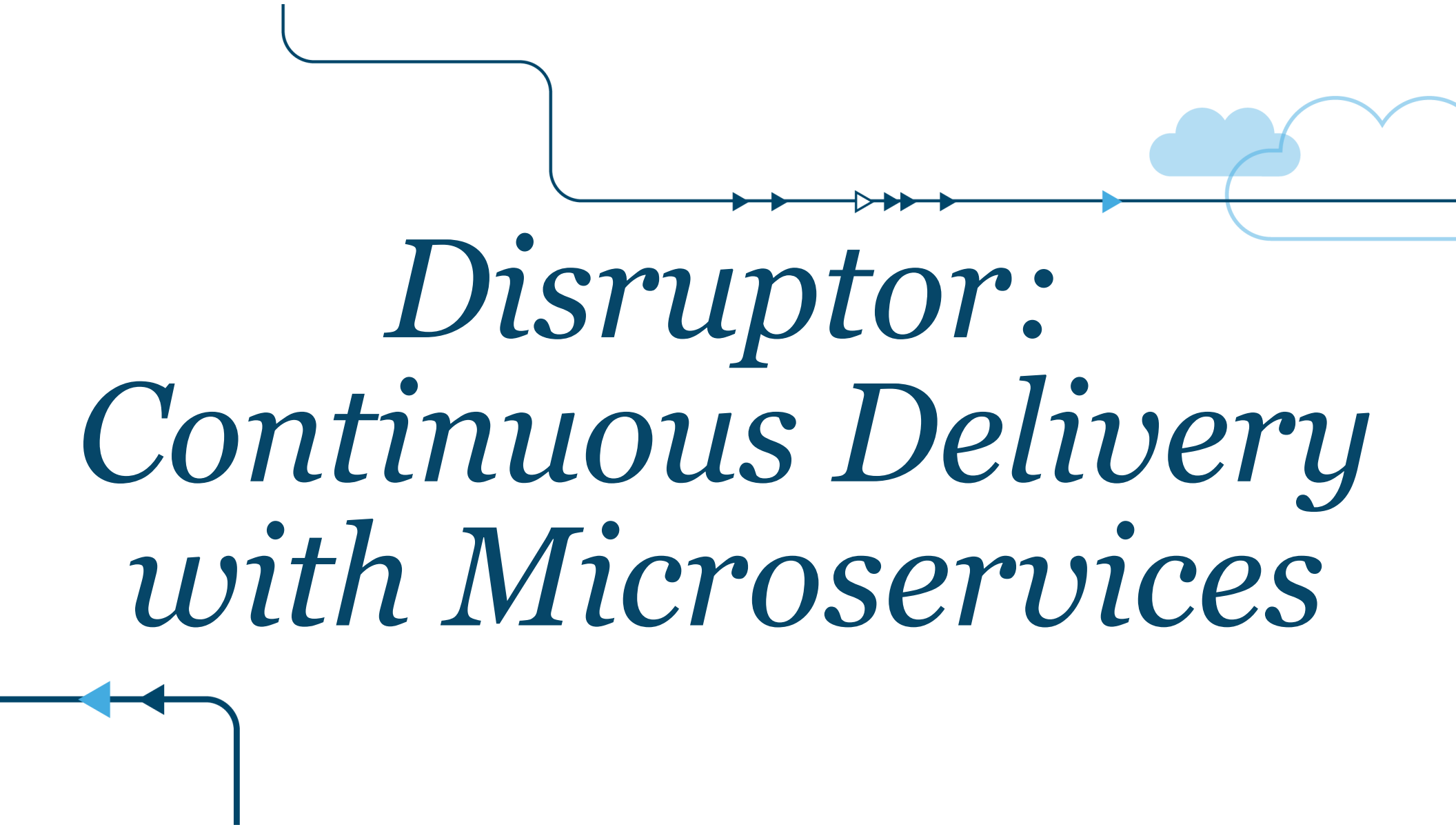
MESOS

etc…

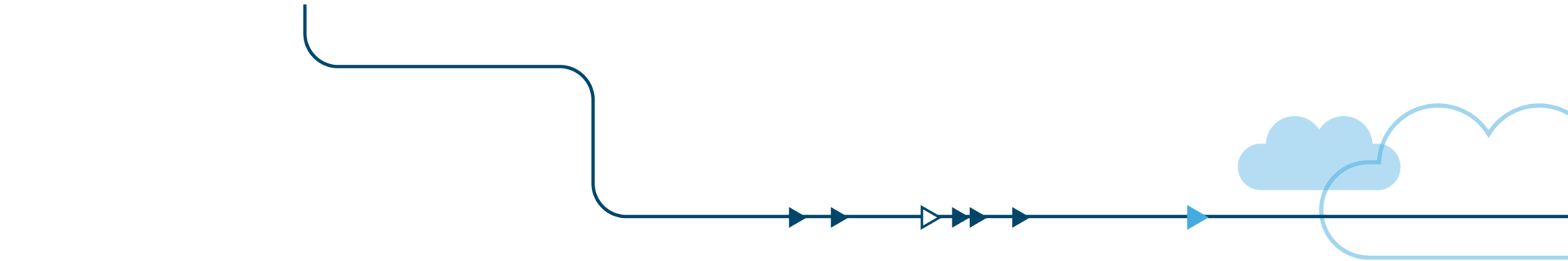
Emerging market for Docker runtime orchestration options

What Happened?



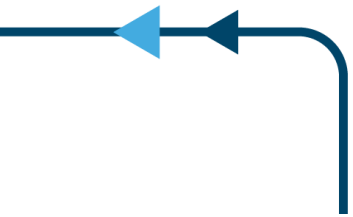


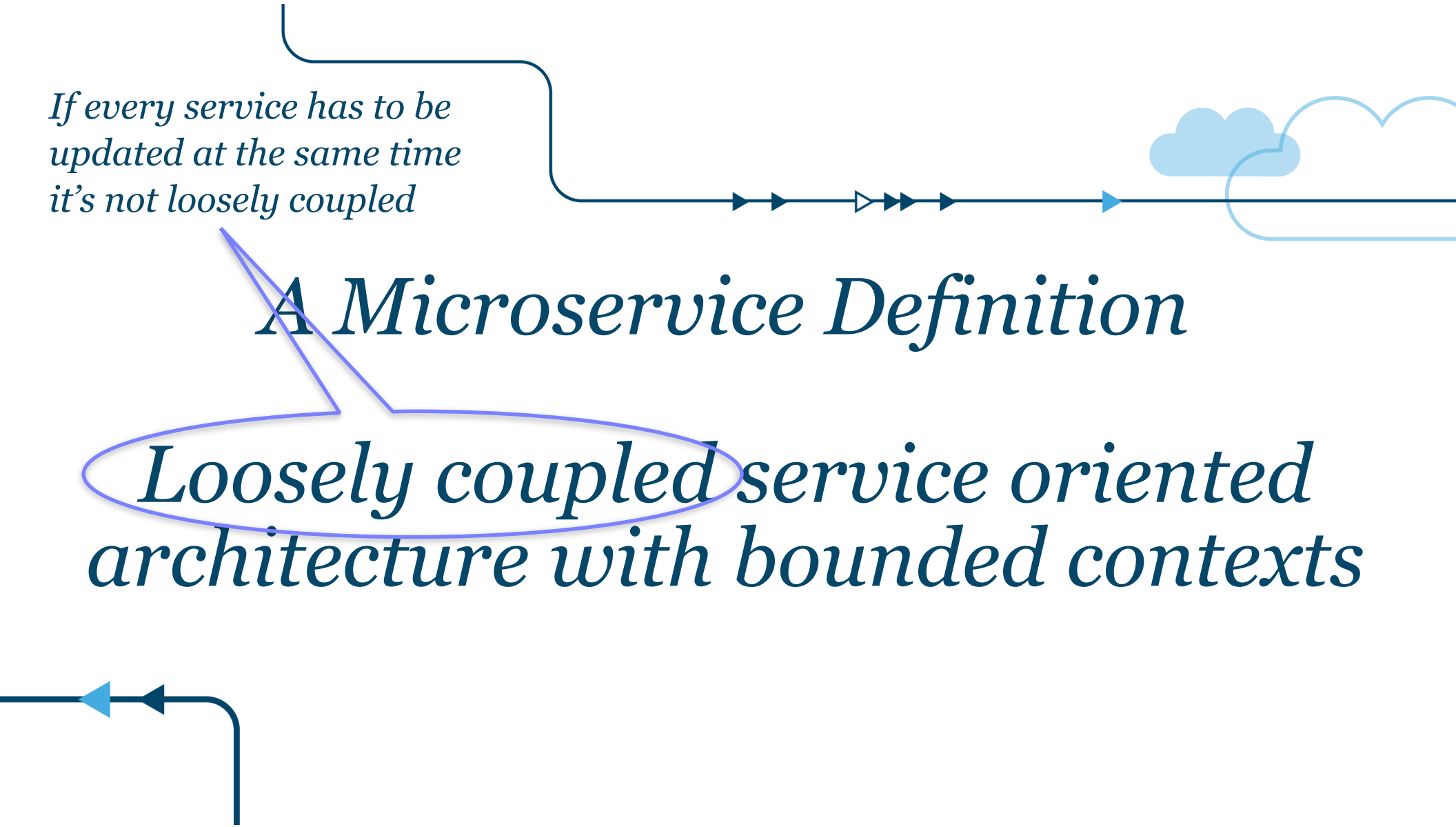
Disruptor: Continuous Delivery with Microservices

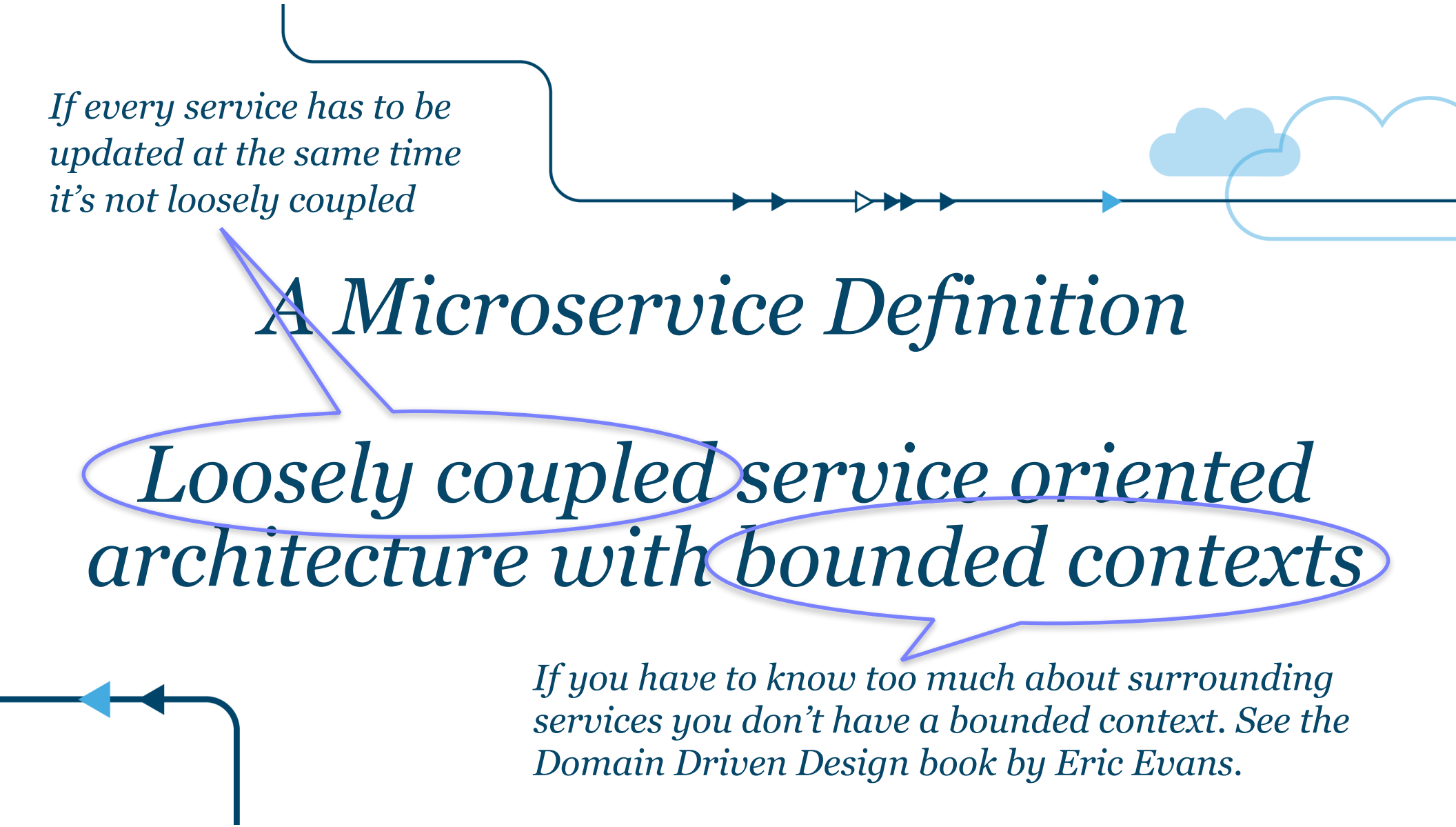


A Microservice Definition

Loosely coupled service oriented architecture with bounded contexts




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

Separate Concerns with Microservices

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

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

NETFLIX | OSS High Availability Patterns

- *Business logic isolation in stateless micro-services*
- *Immutable code with instant rollback*
- *Auto-scaled capacity and deployment updates*
- *Distributed across availability zones and regions*
- *De-normalized single function NoSQL data stores*
- *See over 40 NetflixOSS projects at netflix.github.com*
- *Get "Technical Indigestion" trying to keep up with techblog.netflix.com*

US Bandwidth April 2014

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	24.53%	Netflix	34.21%	Netflix	31.09%
2	HTTP	14.27%	YouTube	13.19%	YouTube	12.28%
3	SSL	6.54%	HTTP	11.65%	HTTP	11.84%
4	Netflix	6.44%	iTunes	3.64%	BitTorrent	5.96%
5	YouTube	5.52%	SSL	3.42%	SSL	3.80%
6	Skype	2.23%	BitTorrent	3.40%	iTunes	3.33%
7	Facebook	2.17%	MPEG	2.85%	MPEG	2.62%
8	FaceTime	1.50%	Facebook	1.99%	Facebook	1.83%
9	Dropbox	1.20%	Amazon Video	1.90%	Amazon Video	1.82%
10	iTunes	1.15%	Hulu	1.74%	Hulu	1.58%
		64.40%		76.24%		74.58%

sandvine

Table 2 - Top 10 Peak Period Applications - North America, Fixed Access

US Bandwidth April 2014

amazon web services™
ELB

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	24.53%	Netflix	34.21%	Netflix	31.09%
2	HTTP	14.27%	YouTube	13.19%	YouTube	12.28%
3	SSL	6.54%	HTTP	11.65%	HTTP	11.84%
4	Netflix	6.44%	iTunes	3.64%	BitTorrent	5.96%
5	YouTube	5.52%	SSL	3.42%	SSL	3.80%
6	Skype	2.23%	BitTorrent	3.40%	iTunes	3.33%
7	Facebook	2.17%	MPEG	2.85%	MPEG	2.62%
8	FaceTime	1.50%	Facebook	1.99%	Facebook	1.83%
9	Dropbox	1.20%	Amazon Video	1.90%	Amazon Video	1.82%
10	iTunes	1.15%	Hulu	1.74%	Hulu	1.58%
		64.40%		76.24%		74.58%

sandvine

Table 2 - Top 10 Peak Period Applications - North America, Fixed Access

US Bandwidth April 2014

NGINX OpenConnect

amazon web services ELB

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	24.53%	Netflix	34.21%	Netflix	31.09%
2	HTTP	14.27%	YouTube	13.19%	YouTube	12.28%
3	SSL	6.54%	HTTP	11.65%	HTTP	11.84%
4	Netflix	6.44%	iTunes	3.64%	BitTorrent	5.96%
5	YouTube	5.52%	SSL	3.42%	SSL	3.80%
6	Skype	2.23%	BitTorrent	3.40%	iTunes	3.33%
7	Facebook	2.17%	MPEG	2.85%	MPEG	2.62%
8	FaceTime	1.50%	Facebook	1.99%	Facebook	1.83%
9	Dropbox	1.20%	Amazon Video	1.90%	Amazon Video	1.82%
10	iTunes	1.15%	Hulu	1.74%	Hulu	1.58%
		64.40%		76.24%		74.58%

sandvine

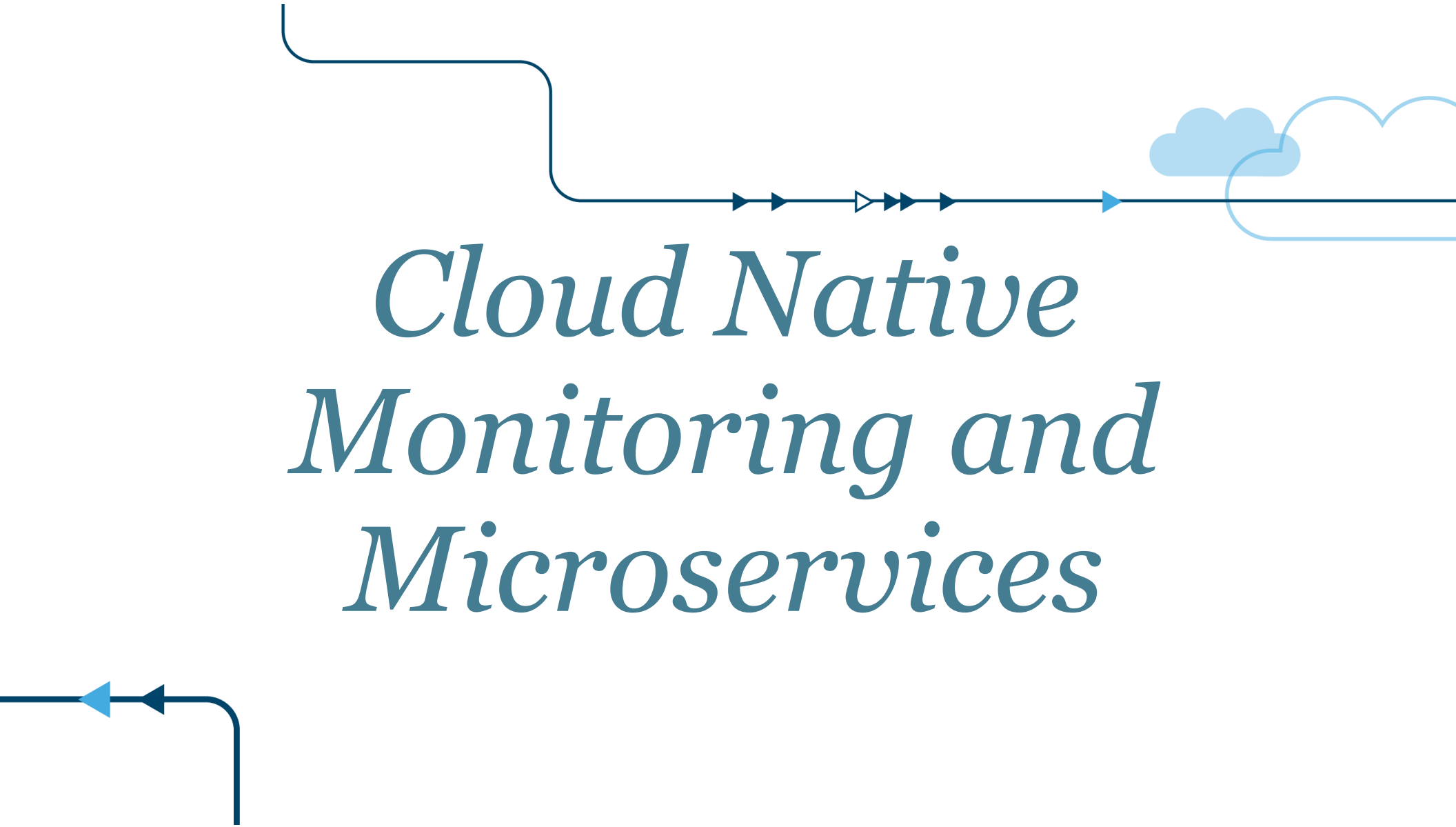
Table 2 - Top 10 Peak Period Applications - North America, Fixed Access

Microservices Development

- *Client libraries*
  - *Even if you start with a raw protocol, a client side driver is the end-state*
  - *Best strategy is to own your own client libraries from the start*
- *Multithreading and Non-blocking Calls*
  - *Reactive model RxJava uses Observable to hide concurrency cleanly*
  - *Netty can be used to get non-blocking I/O speedup over Tomcat container*
- *Circuit Breakers – See Fluxcapacitor.com for code*
  - *NetflixOSS Hystrix, Turbine, Latency Monkey, Ribbon/Karyon*
  - *Also look at Finagle/Zipkin from Twitter*

Microservice Datastores

- *Book: Refactoring Databases*
  - *SchemaSpy to examine schema structure*
  - *Denormalization into one datasource per table or materialized view*
- *Polyglot Persistence*
  - *Use a mixture of database technologies, behind REST data access layers*
  - *See NetflixOSS Storage Tier as a Service HTTP (staash.com) for MySQL and C**
- *CAP – Consistent or Available when Partitioned*
  - *Look at Jepsen torture tests for common systems aphyr.com/tags/jepsen*
  - *There is no such thing as a consistent distributed system, get over it…*

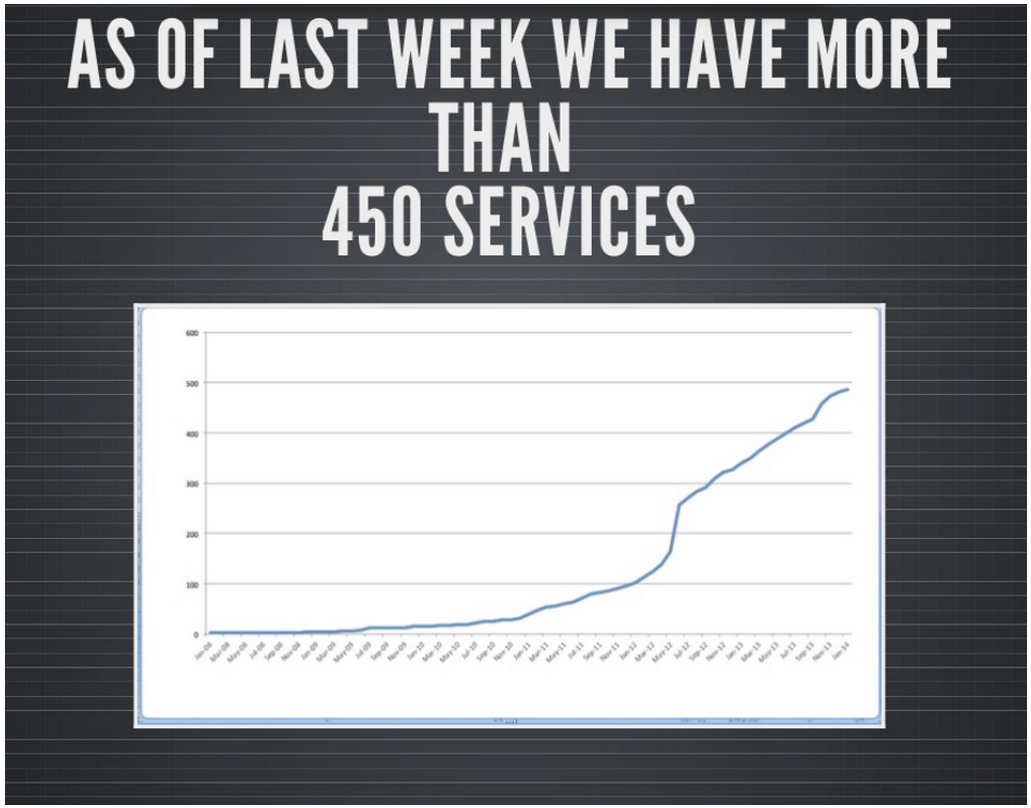


Cloud Native Monitoring and Microservices

Cloud Native

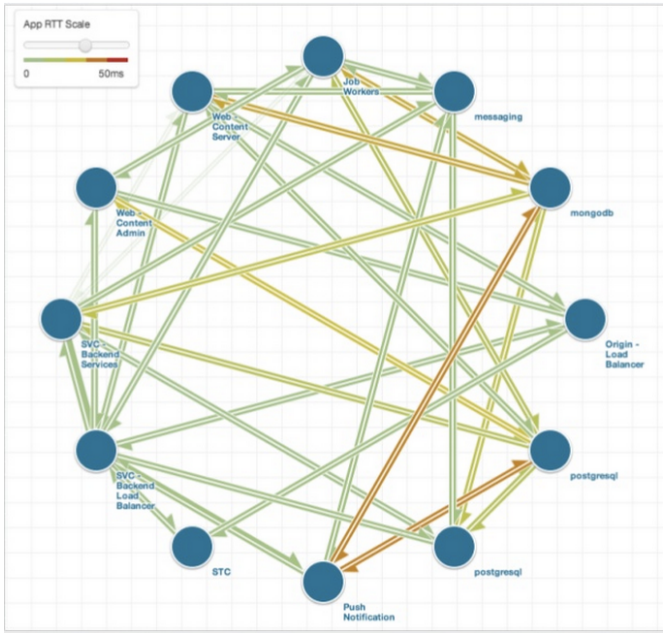
- *High rate of change*
  - *Code pushes can cause floods of new instances and metrics*
  - *Short baseline for alert threshold analysis – everything looks unusual*
- *Ephemeral Configurations*
  - *Short lifetimes make it hard to aggregate historical views*
  - *Hand tweaked monitoring tools take too 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

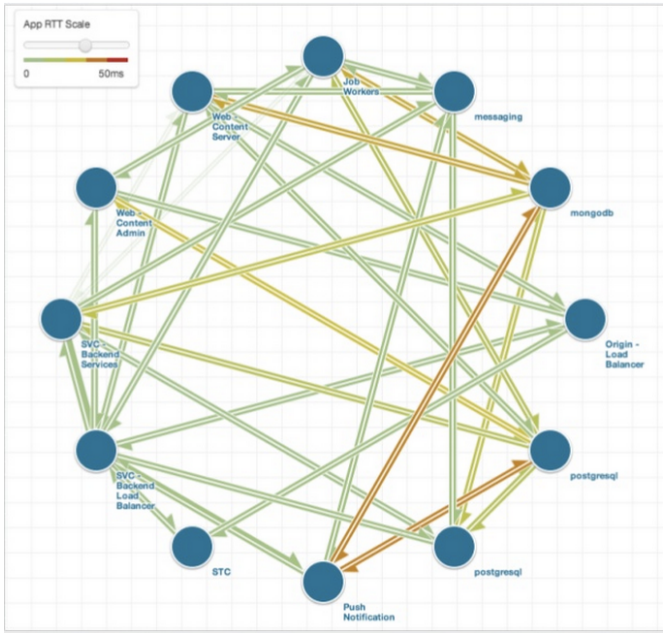
“Death Star” Architecture Diagrams



As visualized by Appdynamics, Boundary.com and Twitter internal tools

“Death Star” Architecture Diagrams

Netflix



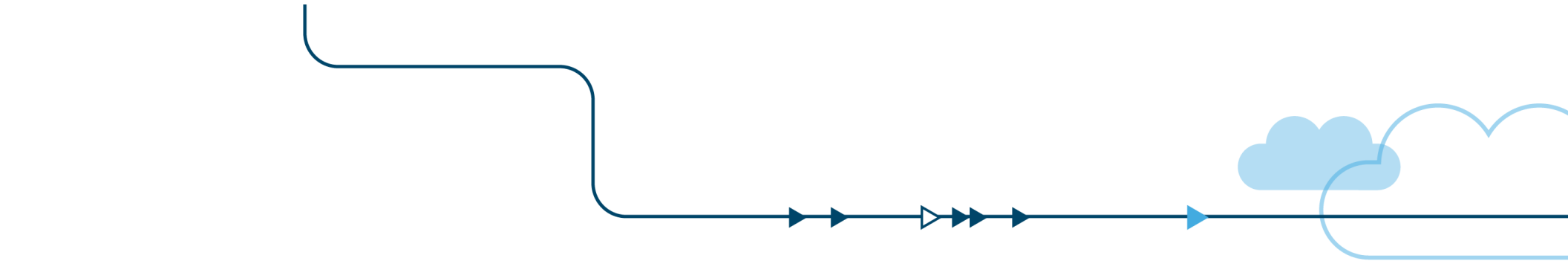
Gilt Groupe (12 of 450)

Twitter

As visualized by Appdynamics, Boundary.com and Twitter internal tools

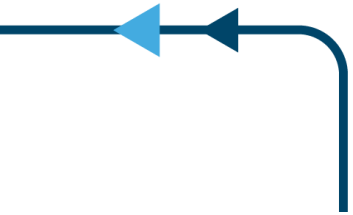
Continuous Delivery and DevOps

- *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*

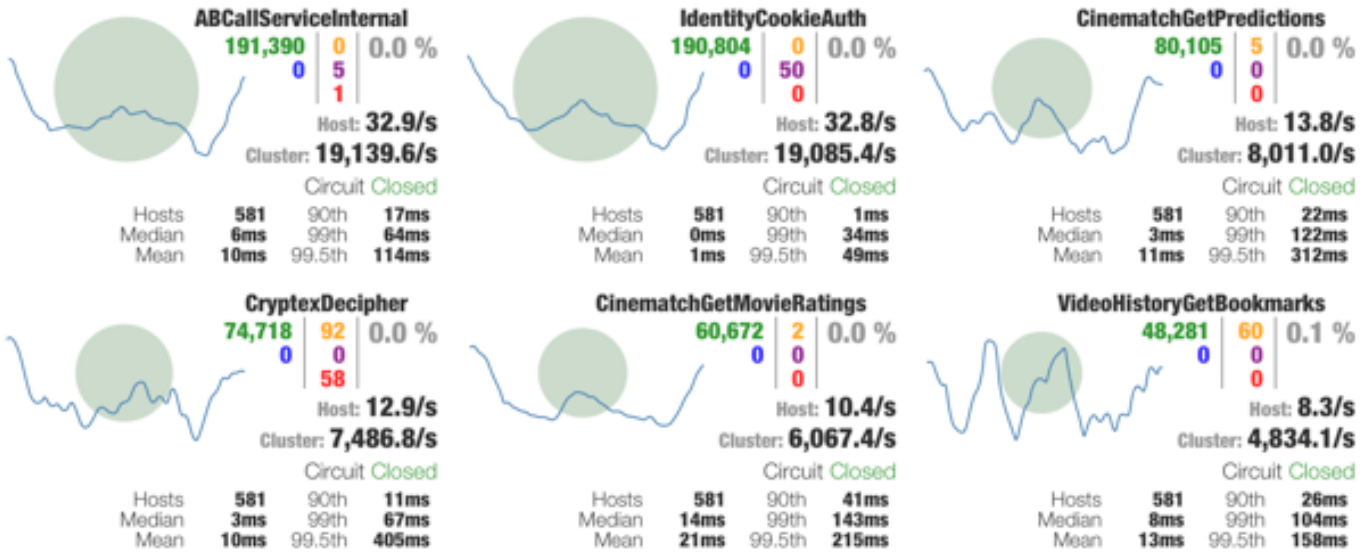


Whoops! I didn't mean that! Reverting…

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

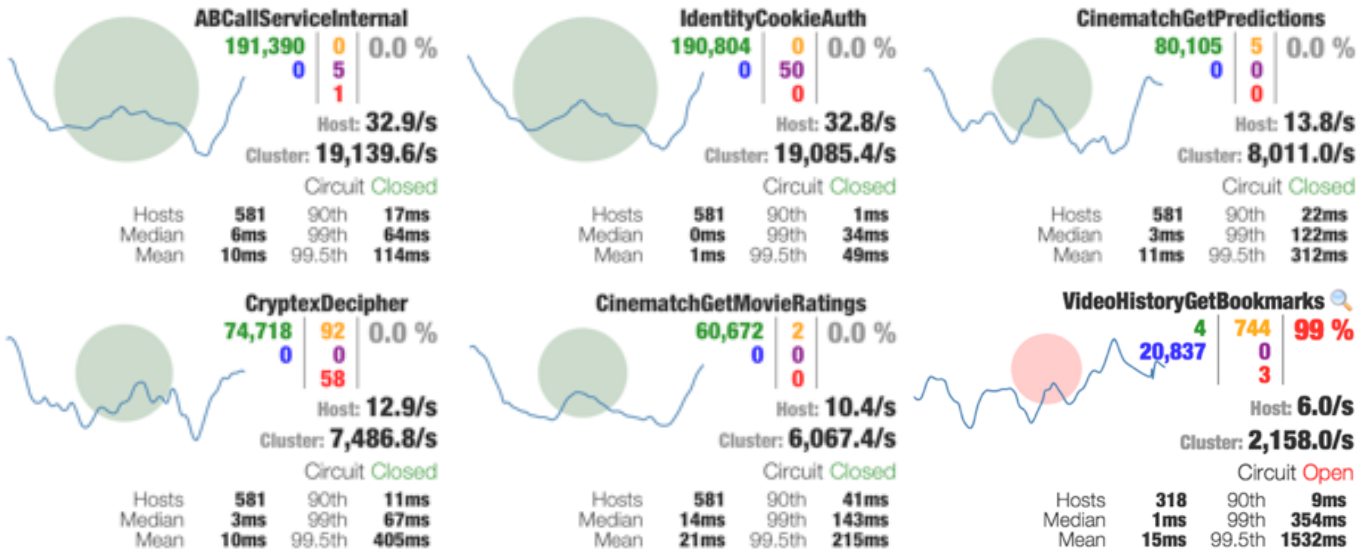


NetflixOSS Hystrix/Turbine Circuit Breaker



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

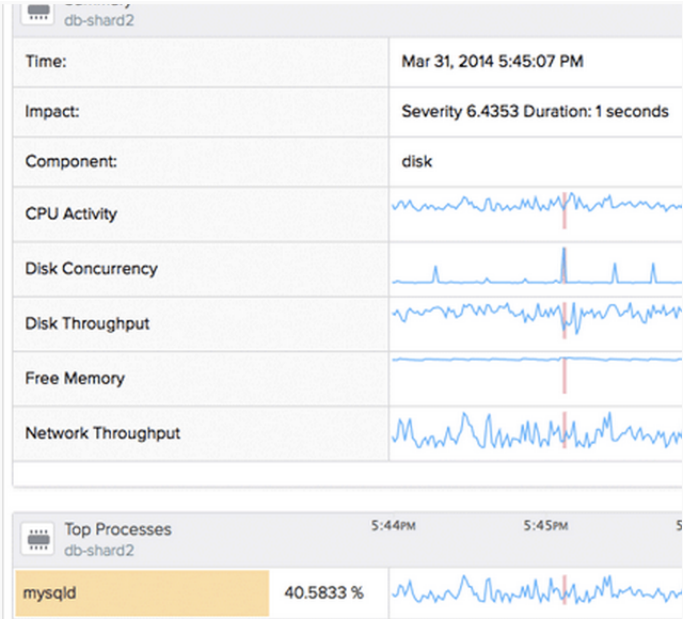
NetflixOSS Hystrix/Turbine Circuit Breaker



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

Low Latency SaaS Based Monitors

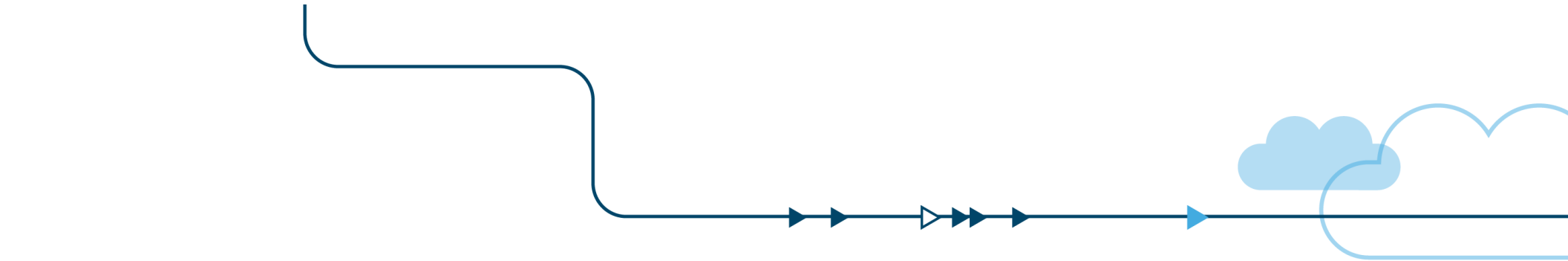
VividCortex



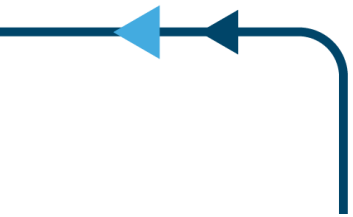
boundary

1-second data collection and real-time streaming processing on all components of the application stack

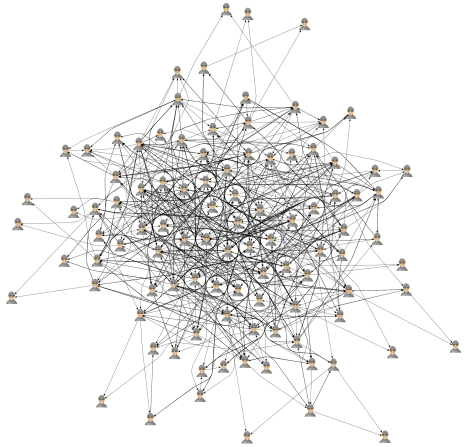
www.vividcortex.com and www.boundary.com

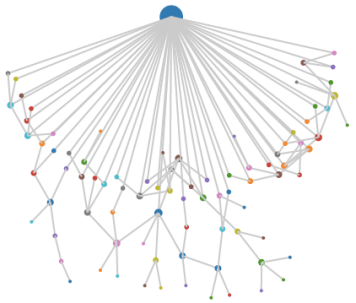


Metric to display latency needs to be less than human attention span (~10s)



Prototyping Ideas

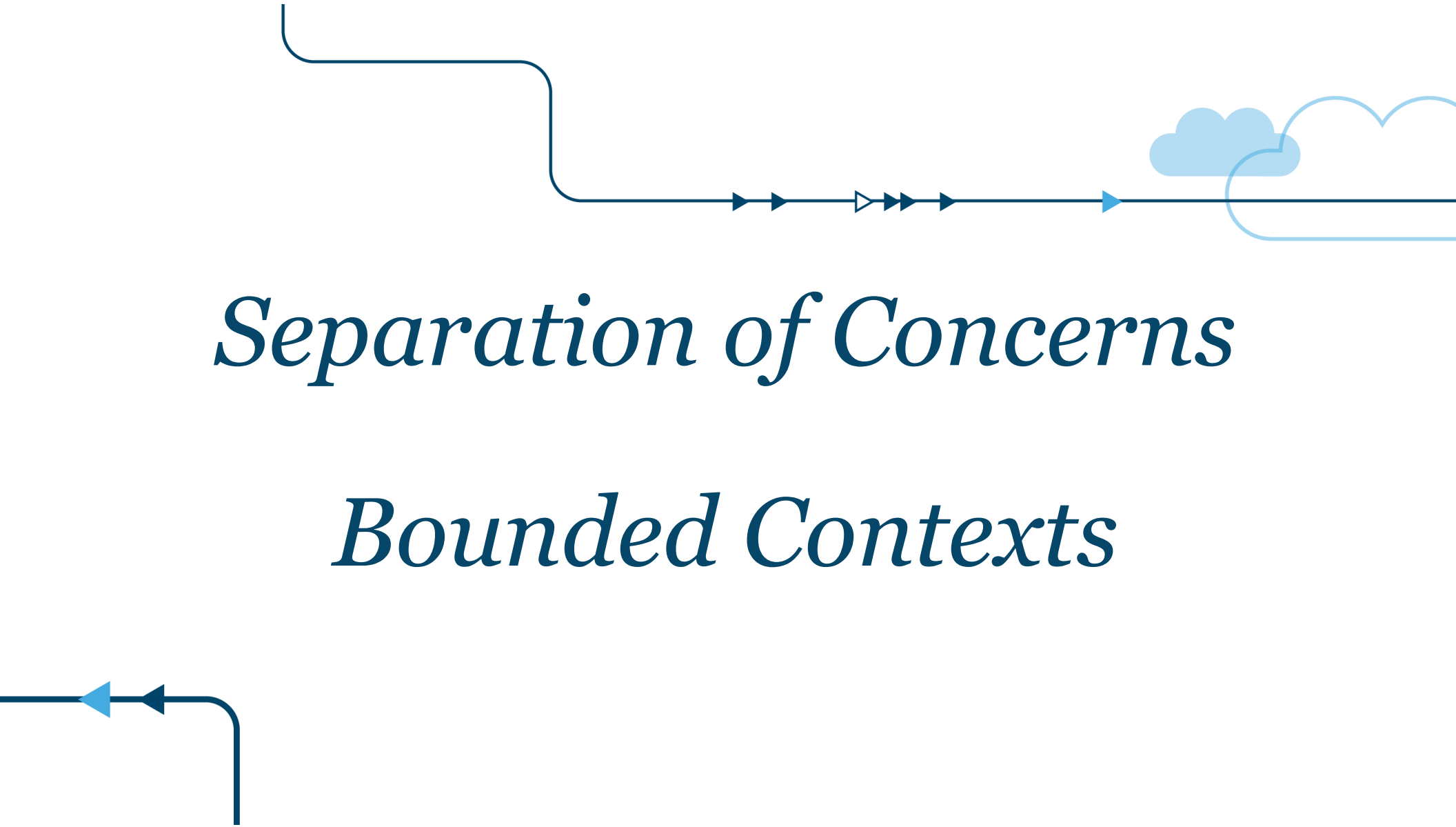




Model and visualize microservices

See github.com/adrianco/spigo
Simulate Protocol Interactions in Go

See github.com/adrianco/d3grow
Dynamic visualization concept

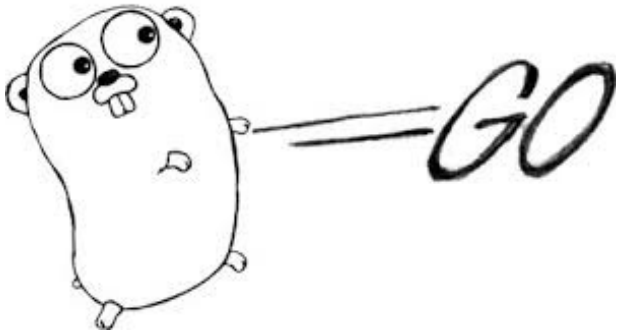


Separation of Concerns

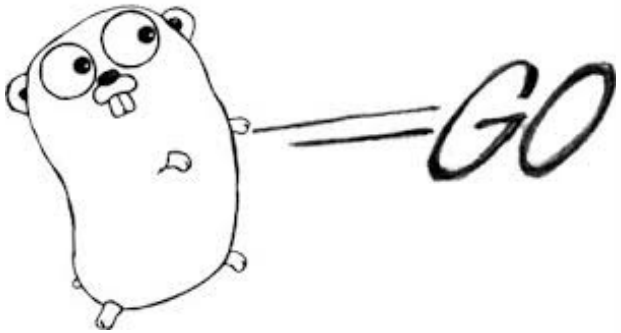
Bounded Contexts

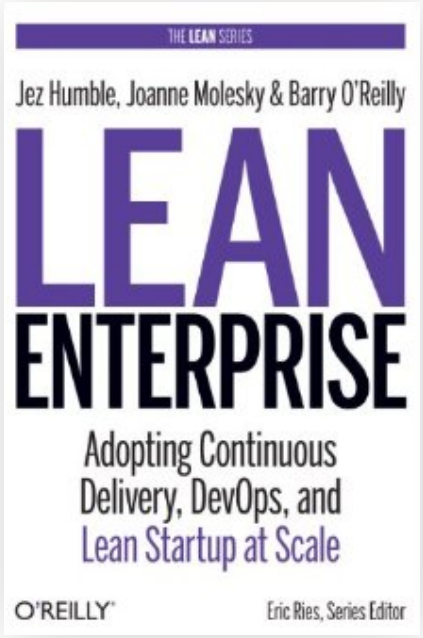
Forward Thinking

Forward Thinking

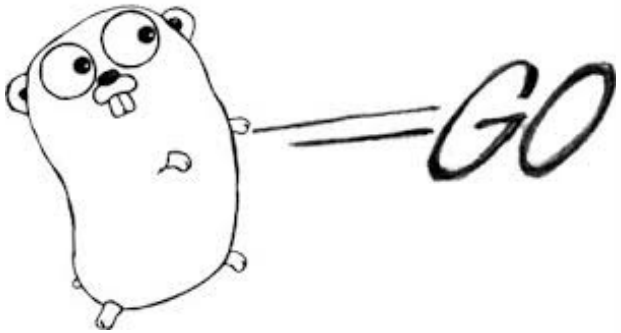


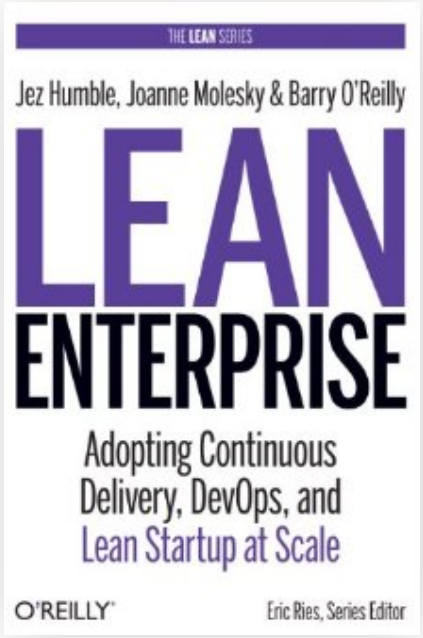
Forward Thinking

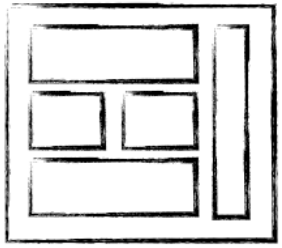




Forward Thinking







MONOLITHIC/LAYERED

MICRO SERVICES

http://eugenedvorkin.com/seven-micro-services-architecture-advantages/

Any Questions?

- *Battery Ventures http://www.battery.com*
- *Adrian's Blog http://perfcap.blogspot.com*
- *Slideshare http://slideshare.com/adriancockcroft*

- *Monitorama Opening Keynote Portland OR - May 7th, 2014 - Video available*
- *GOTO Chicago Opening Keynote May 20th, 2014 - Video available*
- *Qcon New York – Speed and Scale - June 11th, 2014 - Video available*
- *Structure - Cloud Trends - San Francisco - June 19th, 2014 - Video available*
- *GOTO Copenhagen/Aarhus – Denmark – Sept 25th, 2014*
- *DevOps Enterprise Summit - San Francisco - Oct 21-23rd, 2014 #DOES14 - Videos available*
- *GOTO Berlin - Germany - Nov 6th, 2014*
- *AWS Re:Invent - Cloud Native Cost Optimization - Las Vegas - November 14th, 2014*
- *Dockercon Europe - Amsterdam - December 4th, 2014*

Disclosure: some of the companies mentioned are Battery Ventures Portfolio Companies
See www.battery.com for a list of portfolio investments