

Impact of Architecture on Continuous Delivery

Russell Miller

rmiller@impulse.com

Miller.Russ@gmail.com

[@RussMiller123](#)

Podcast: ArchitectureCast.net

Context:

- Greenfield project
- Build cloud-based monitoring system
- Social/collaborative

Pilot project for:

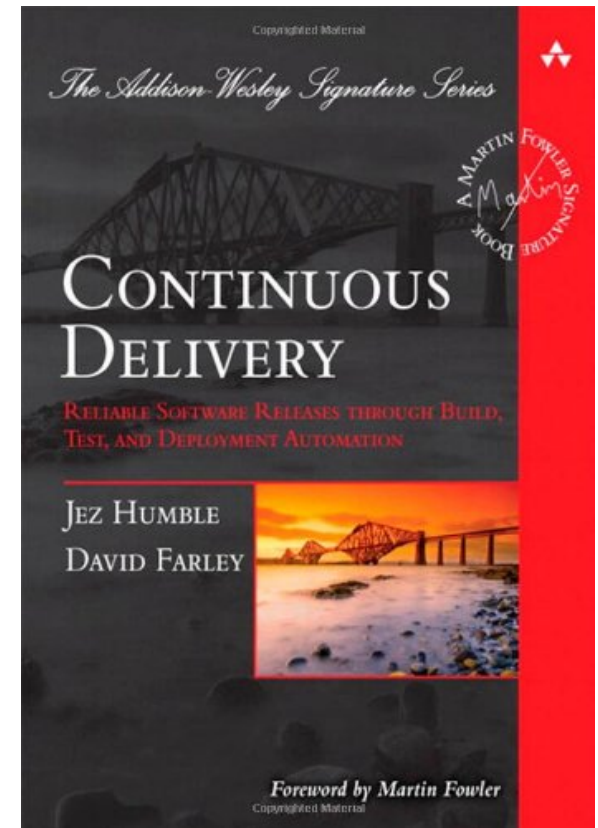
- Truly cloudy application(cloud native)
- Agile pulling in some lean principles

Definition:

Continuous Delivery: An automated approach to software delivery characterized by **frequent** and **predictable** delivery of **incremental** units of business value.

Utilizing:

- Continuous integration
- Automated testing
- Automated Deployment (at least to test)

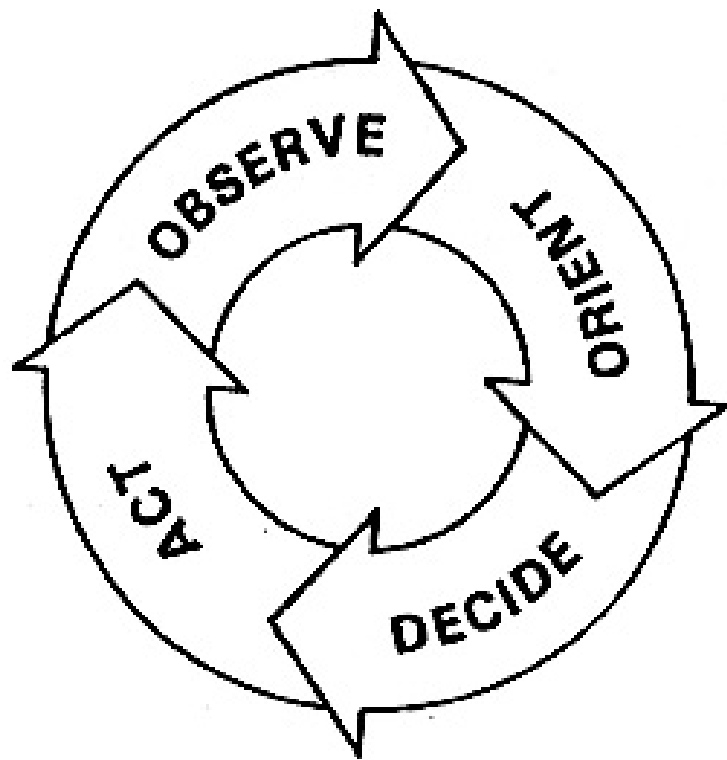




Frequent, Small, Predictable



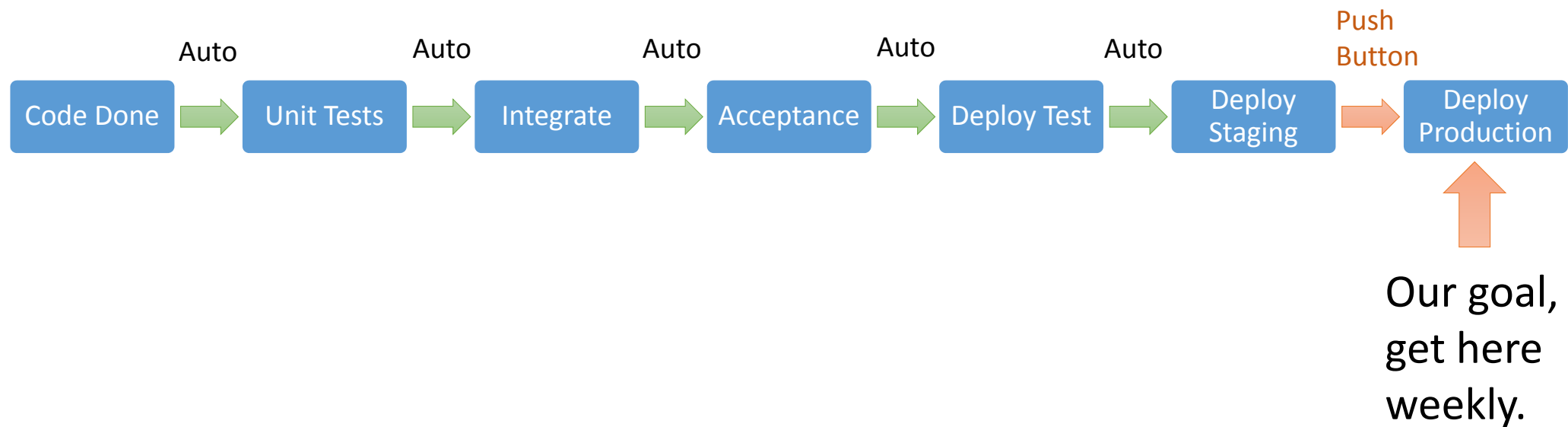
- Continuously usable
- Incrementally better



Continuously Learn
and Innovate



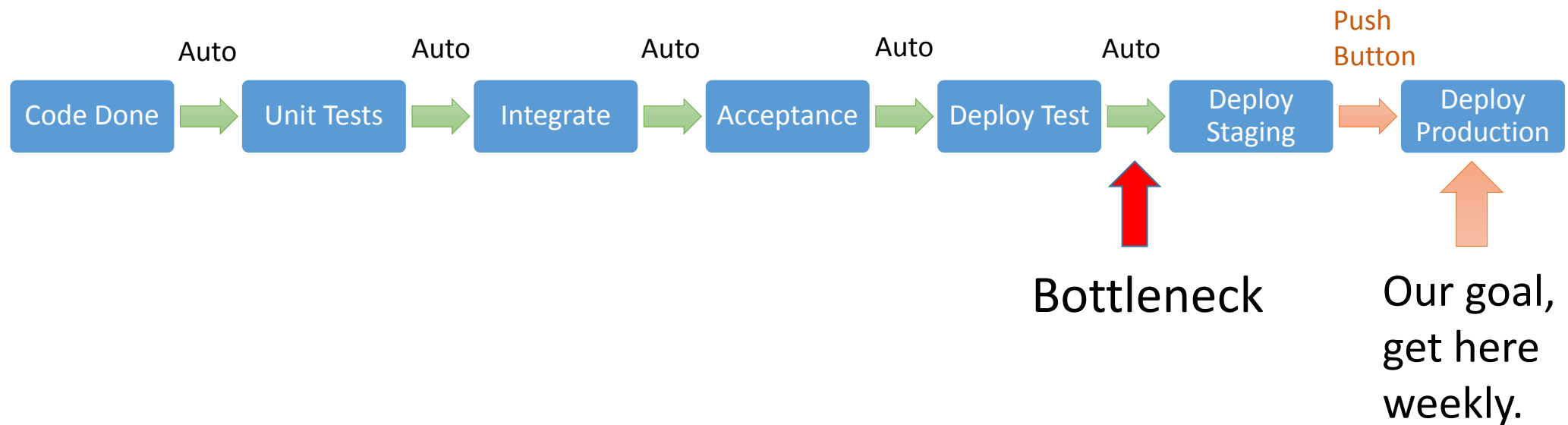
No Drones, instead we have a build pipeline:





Our dream

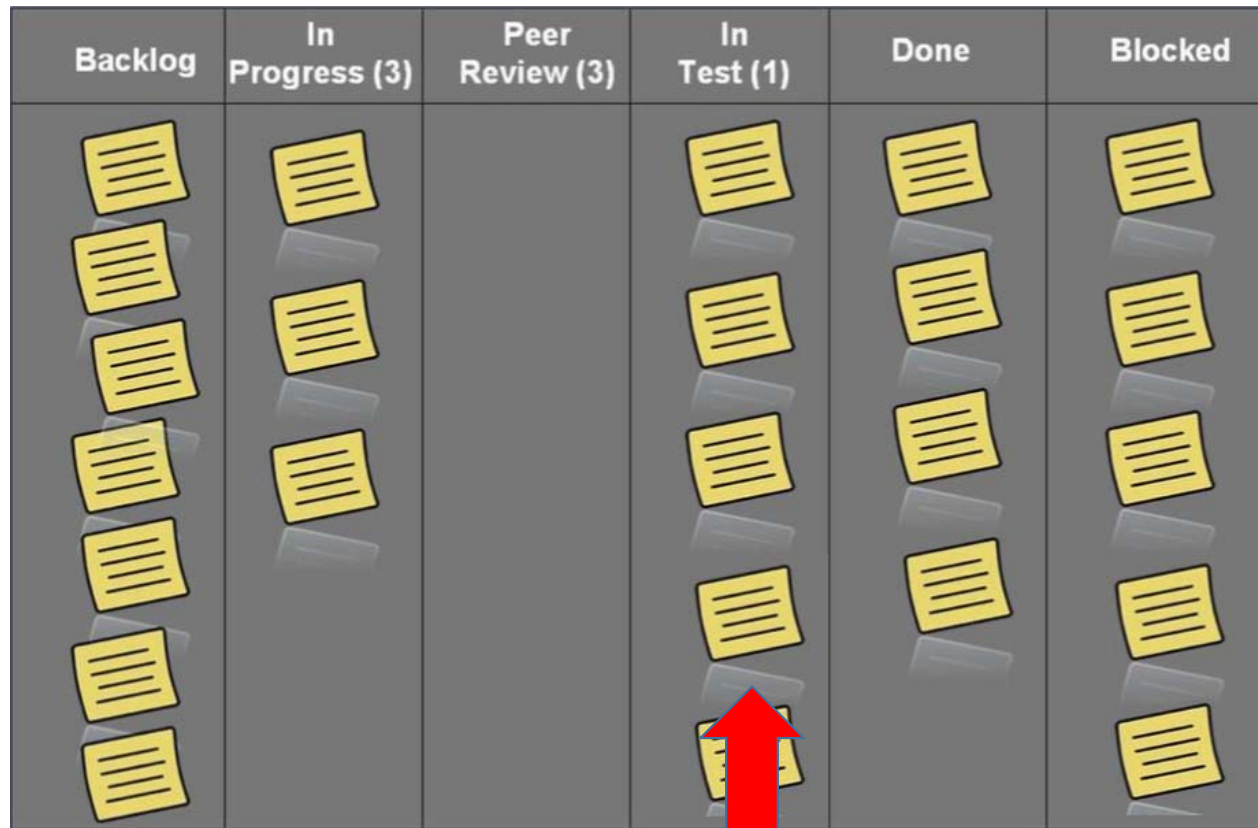
No Drones, instead we have a build pipeline:



As if, some drones had to wait on others to land!



Getting to what the business considers done...



Bottleneck

Weren't able to drive the **last mile** over the finish line!



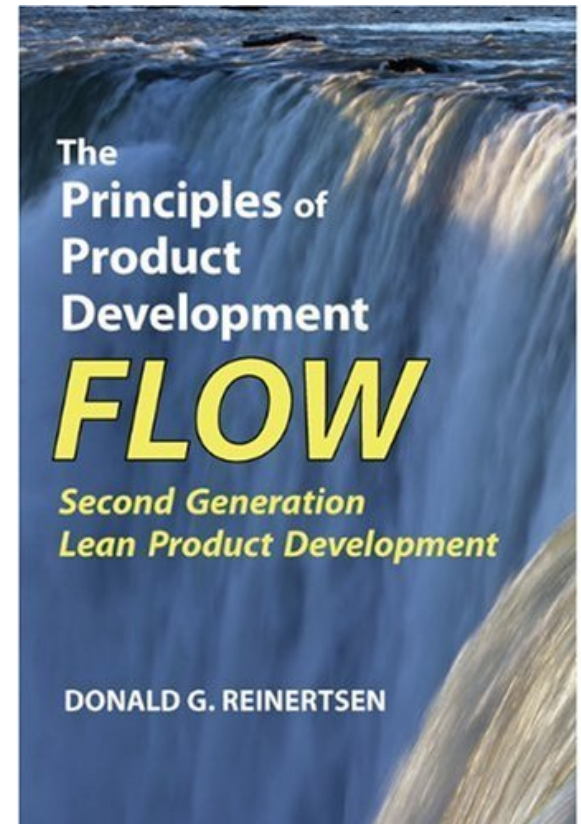
“because the information [developers are] producing is invisible to them, the batch sizes are, too”,
HBR, “Six Myths of Product Development” by Thomke and Reinertsen

Main lesson we learned:

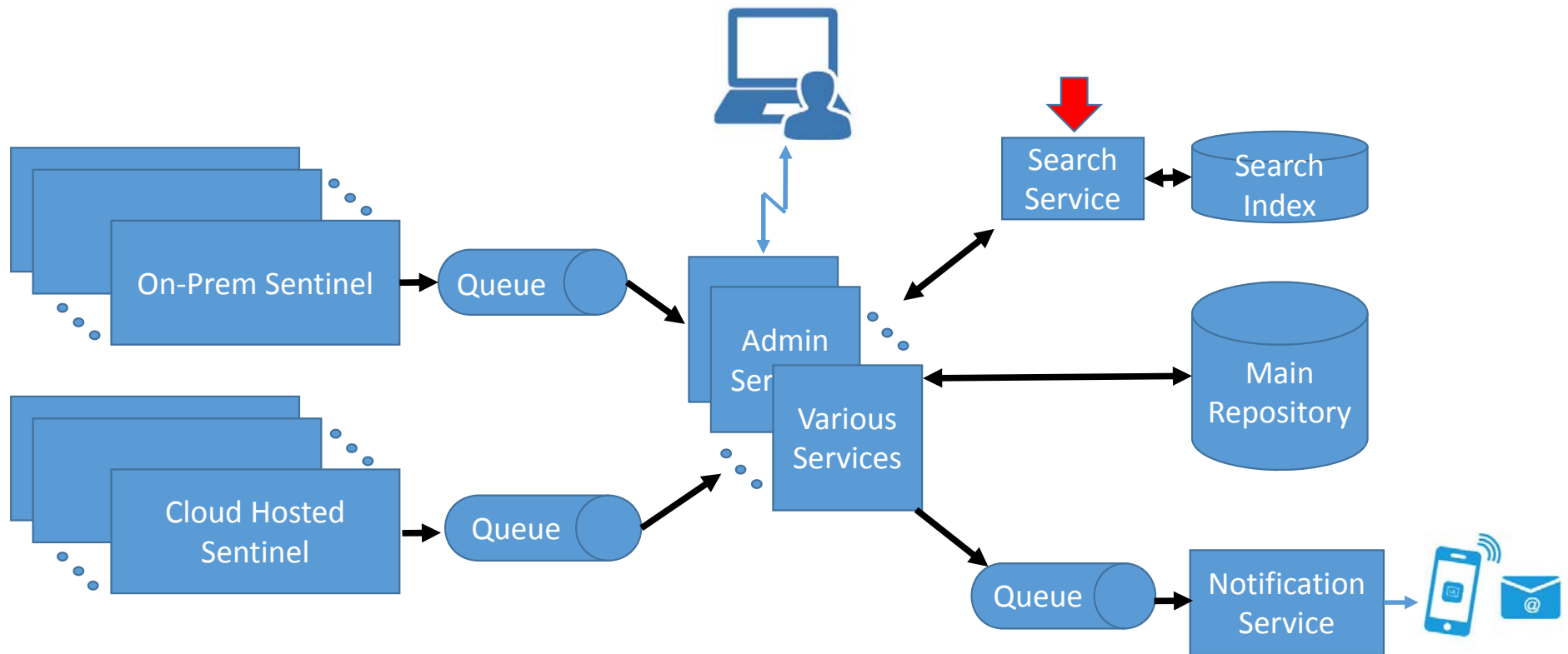
The Fluidity Principle: Loose coupling between product subsystems enables small batches. (p.126)

"Once a product developer realizes that small batches are desirable, they start adopting product architectures that permit work to flow in small, decoupled batches."

Credit for this slide to: John Esser and Russell Barnett



Service Architecture

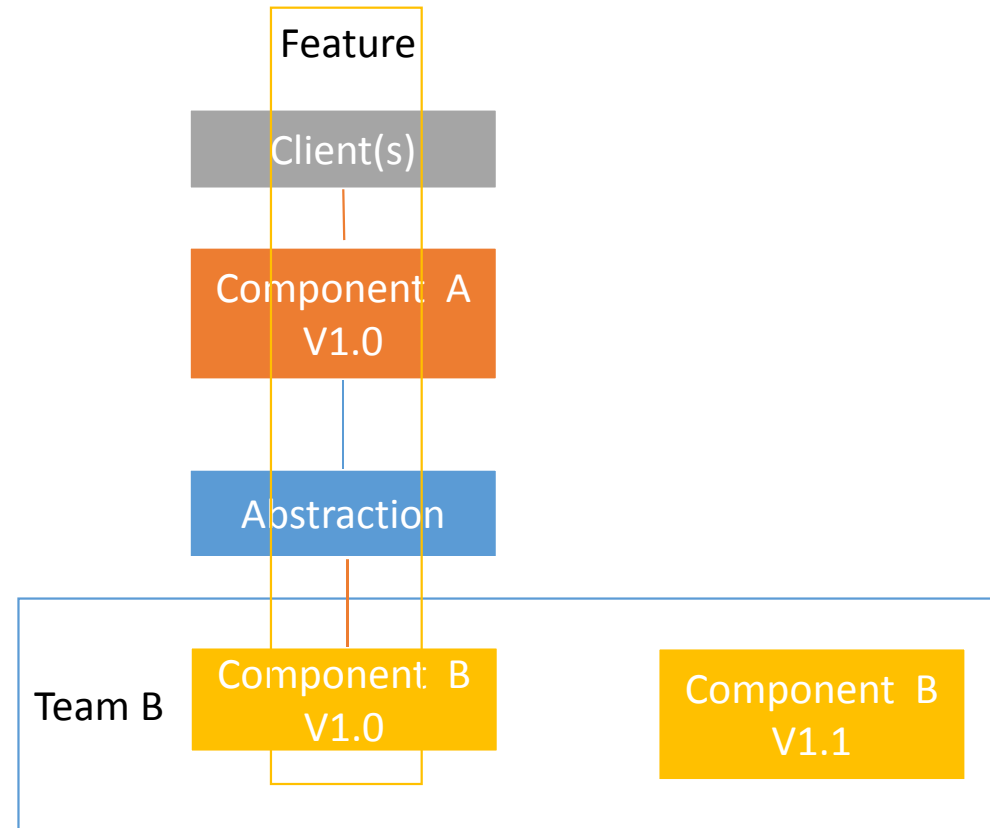


Micro Services

- Separately deployable and separately deployed! (leverage the cloud!)
- SOLID Principle (at the component level)
- Dependencies create larger batches!
- Other advantage:
 - Is it working? (Testing pieces in production)
 - Still Used?
 - Hot Swappable

Problem solved?

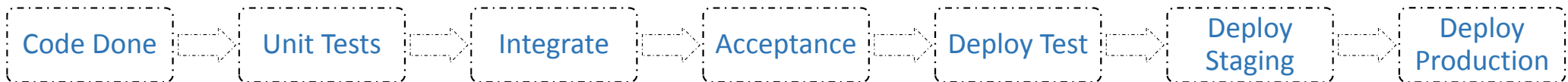
- But, many new features cut across components/services



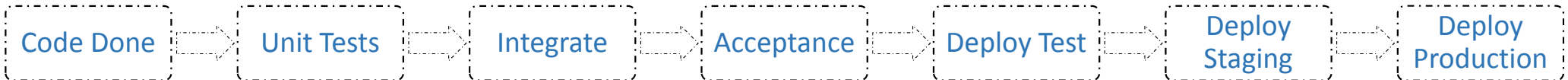
Parallel Landings



Feature Switch A



Feature Switch B



Feature Switch = Virtual Pipeline

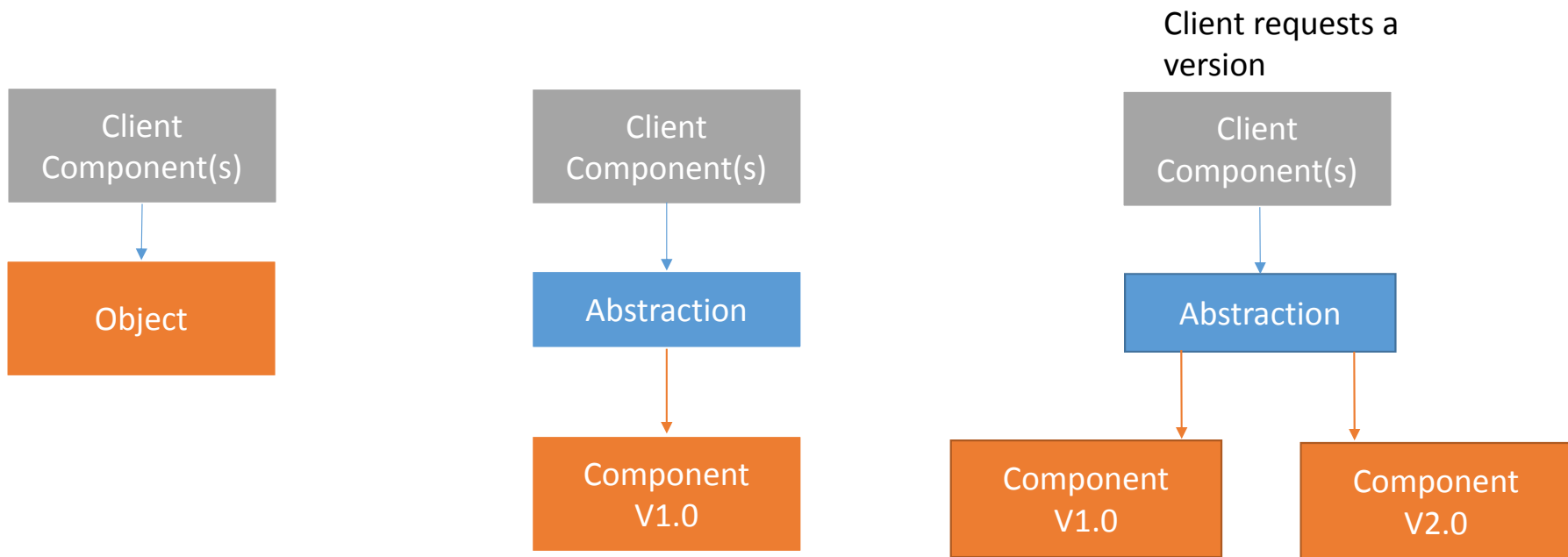


Ultimately, need “Feature Switches” (e.g. Twitter Decider framework)

- Runtime Configuration
- Built on top of license flag (tiers)
- Smart switching based on other criteria
- Pays off more than just for enabling small batches

Next lesson: Branch by Abstraction

- Layers already there if you are Mocking
- Client virtually merges with new branch





Did we realize our dream?

In Conclusion...

Lessons learned:

- Architecture impacts batch size
- Decouple deployment, also (Micro Services)
- Virtualize your pipeline:
 - Branch by abstraction
 - Feature switches
 - Piece meal dark deployment

Architecture Matters, of course!

Resources

- Branch by Abstraction: <http://continuousdelivery.com/2011/05/make-large-scale-changes-incrementally-with-branch-by-abstraction/>
- Continuous Delivery at Ancestry.com (John Esser and Russell Barnett) <http://www.infoq.com/presentations/ancestry-SOA-continuous-delivery>
- Adjusting Your Architecture for Continuous Delivery <http://www.infoq.com/interviews/laycock-continuous-delivery>
- Batch size not clear to developers: <http://hbr.org/2012/05/six-myths-of-product-development/ar/1>
- Forrester Continuous Delivery [http://info.thoughtworks.com/rs/thoughtworks2/images/Continuous%20Delivery%20 %20A%20Maturity%20Assessment%20ModelFINAL.pdf](http://info.thoughtworks.com/rs/thoughtworks2/images/Continuous%20Delivery%20%20A%20Maturity%20Assessment%20ModelFINAL.pdf)

Forrester Continuous Delivery Maturity Model

Table 1
Continuous Delivery Maturity Model

Level	Delivery focus	Characteristics	Result
5: optimizing	Hypothesis-driven delivery	Teams focus on optimizing cycle time to learn from customers. <ul style="list-style-type: none">All new requirements describe how the value of the feature will be measured.Product teams are responsible for implementing metrics to gather this data through techniques such as A/B testing.Systems are architected with continuous deployment in mind, supporting patterns such as dark launching to decouple deployment from release.Database changes are decoupled from application deployments.	Continuous deployment capability enables business innovation/experimentation
4: quantitatively managed	Release on demand	Delivery teams prioritize keeping code trunk deployable over doing new work. <ul style="list-style-type: none">Deployment pipeline automatically rejects bad changes from version control.Cross-functional end-to-end product-centric teams manage products throughout life cycle.Comprehensive automated test suites are created through TDD/ATDD and maintained by developers and testers working together.Teams monitor and manage work in process and deliver work in small batches.	Release on demand: Software is always in a releasable state. Release time box is well defined and equal to, or less than, business need.

5. Optimizing
4. Quantitatively Managed
3. Defined
2. Managed
1. Initial

“Systems are **architected** with continuous deployment in mind...”

What does that mean?

3rd Party Service considerations

Examples we used:

- Twillio
- ElasticSearch
- MongoDB
- Payment Service

What to consider:

- Auto Configurable?
- Used in a testable way?
- Monitorable?

Deploy Abstractly

- Truly cloudy – components should not care about
- Which machine (cattle)
- IP addresses
- DNS names should be auto-set
- Certificates

Forrester Continuous Delivery Maturity Model

Table 1
Continuous Delivery Maturity Model

Level	Delivery focus	Characteristics	Result
5: optimizing	Hypothesis-driven delivery	Teams focus on optimizing cycle time to learn from customers. <ul style="list-style-type: none">All new requirements describe how the value of the feature will be measured.Product teams are responsible for implementing metrics to gather this data through techniques such as A/B testing.Systems are architected with continuous deployment in mind, supporting patterns such as dark launching to decouple deployment from release.Database changes are decoupled from application deployments.	Continuous deployment capability enables business innovation/experimentation
4: quantitatively managed	Release on demand	Delivery teams prioritize keeping code trunk deployable over doing new work. <ul style="list-style-type: none">Deployment pipeline automatically rejects bad changes from version control.Cross-functional end-to-end product-centric teams manage products throughout life cycle.Comprehensive automated test suites are created through TDD/ATDD and maintained by developers and testers working together.Teams monitor and manage work in process and deliver work in small batches.	Release on demand: Software is always in a releasable state. Release time box is well defined and equal to, or less than, business need.

5. Optimizing

4. Quantitatively Managed

3. Defined

2. Managed

1. Initial

Continuous deployment capability enables business innovation/experimentation.

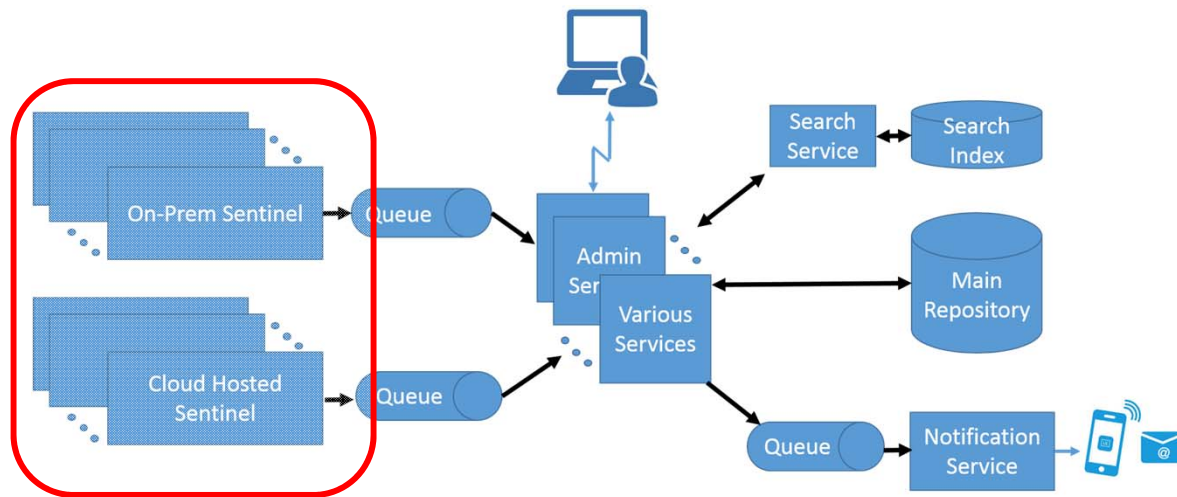
What we wanted!!!



Our Reality!
Log jam of features waiting to be integrated.

Problem solved?

- Added ability to deploy subset of Services
- Separate Andon cords to pull
- Able to run multiple versions (be careful)



Bigger batches than we realized:

- Uncovered a couple of areas in the architecture that could be decoupled (e.g. Notification)
- Issue was all-or-nothing deployment
- Development in small batches, but deploying big batches



Other techniques used tactically

- Branch by abstraction
- Prepare for rollback, but prefer rolling forward
- Deep stack monitoring (designed in)
- Test in production (TiP)
- Not trusting the upstream component