Bottom Up Adoption of Microservices w/ Kubernetes & Envoy (Service Oriented Development)

Rafael Schloming - @rschloming
Philip Lombardi - @TheBigLombowski

How do I break up my monolith?

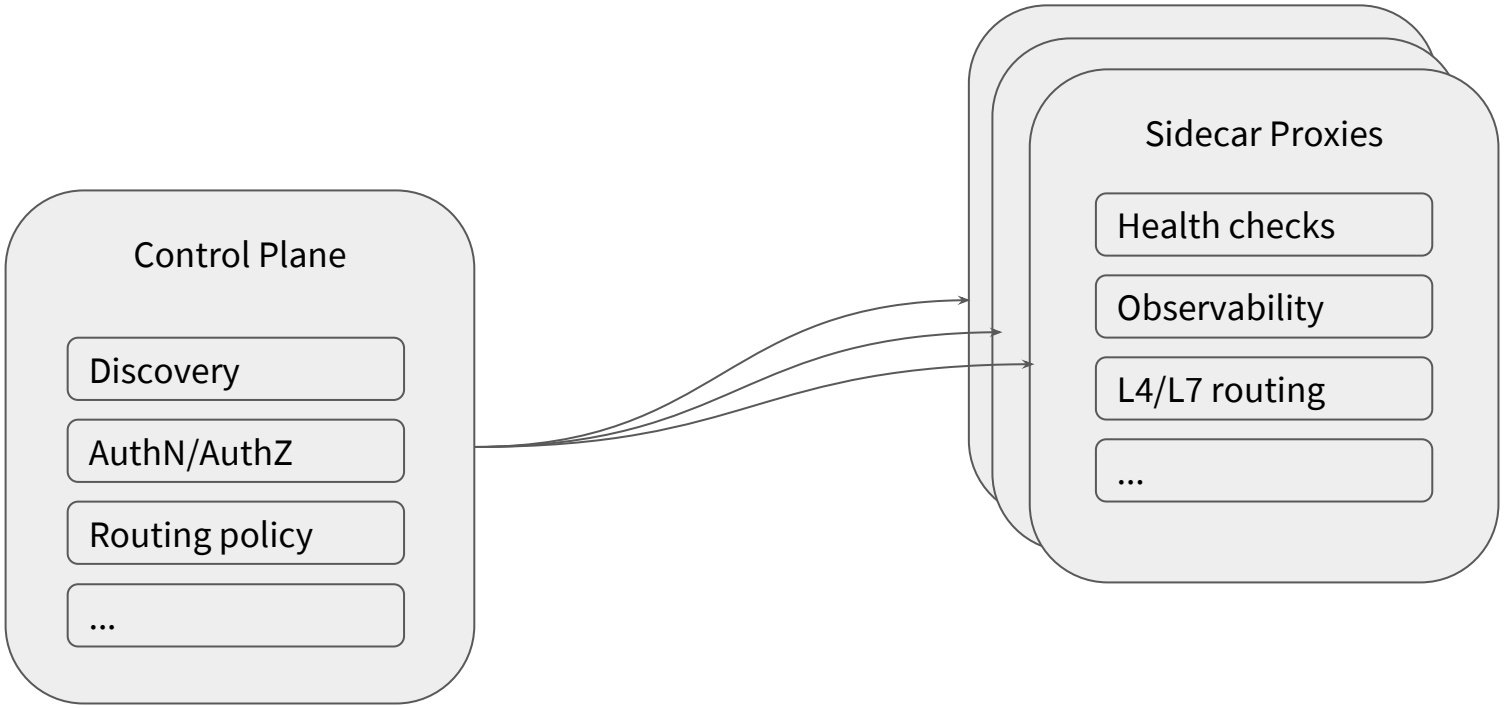
How do I architect my app with microservices?

What infrastructure do I need in place before I can benefit from microservices?

Microservices at Datawire ...

- Building a cloud application using microservices in 2015
- Distributed systems engineers
- Deeply studied microservices Architectures & Technology

Control Plane and Data Plane

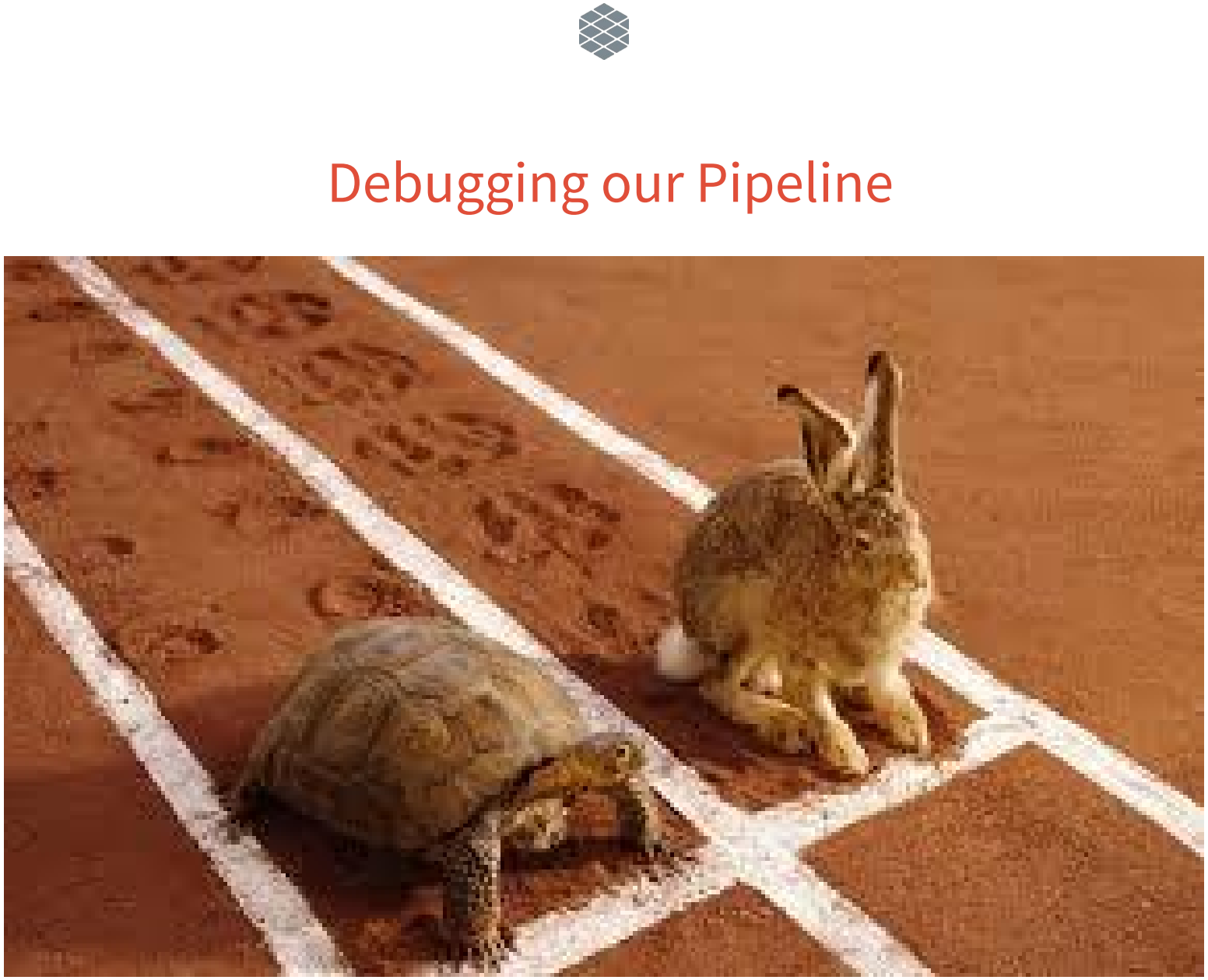


Debugging Velocity (or lack thereof)

~~Tooling~~

~~Architecture~~

Process!!!

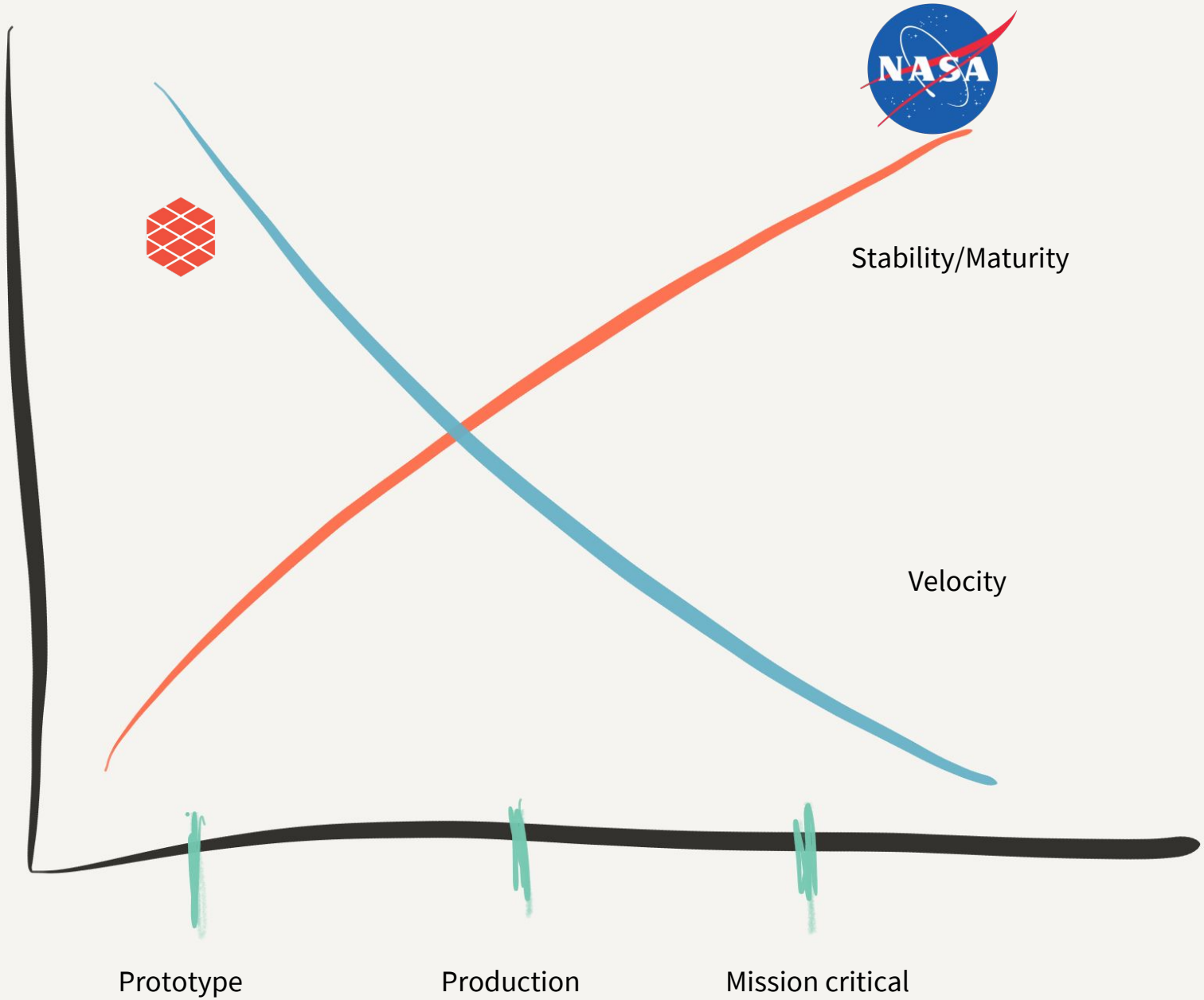


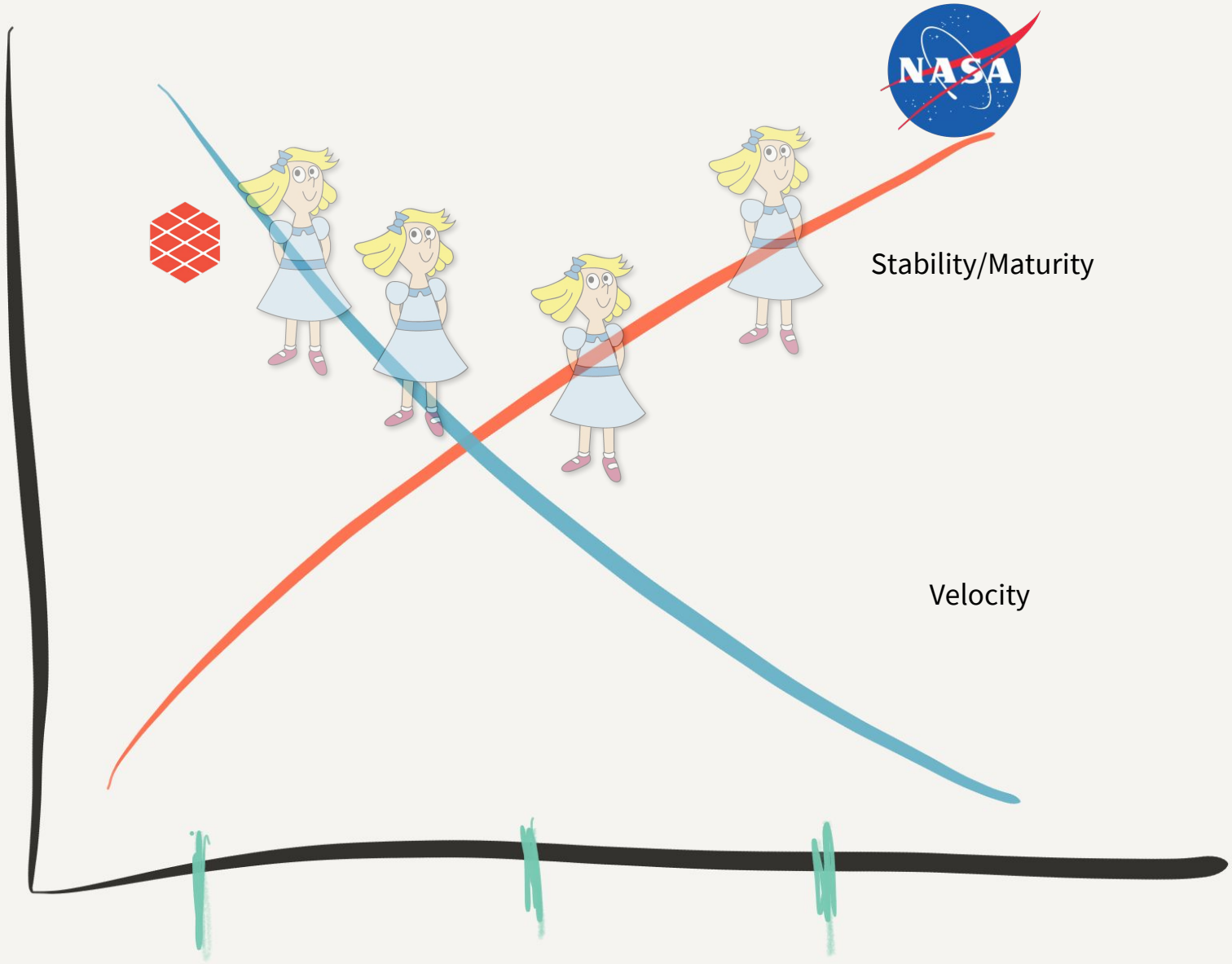
Debugging our Pipeline

Velocity comes from Process, not Architecture

~~Service Oriented Architecture~~

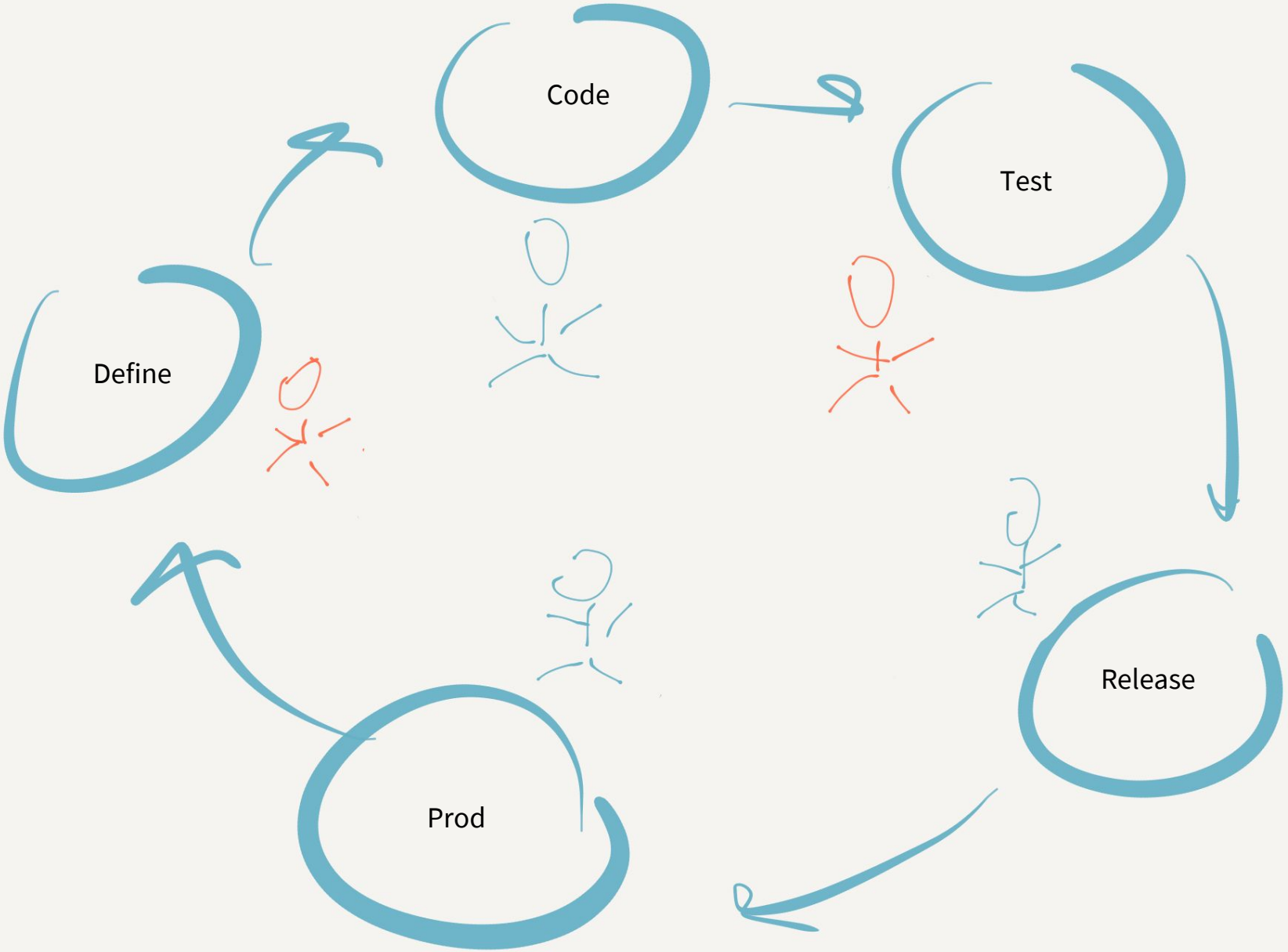
Service Oriented Development

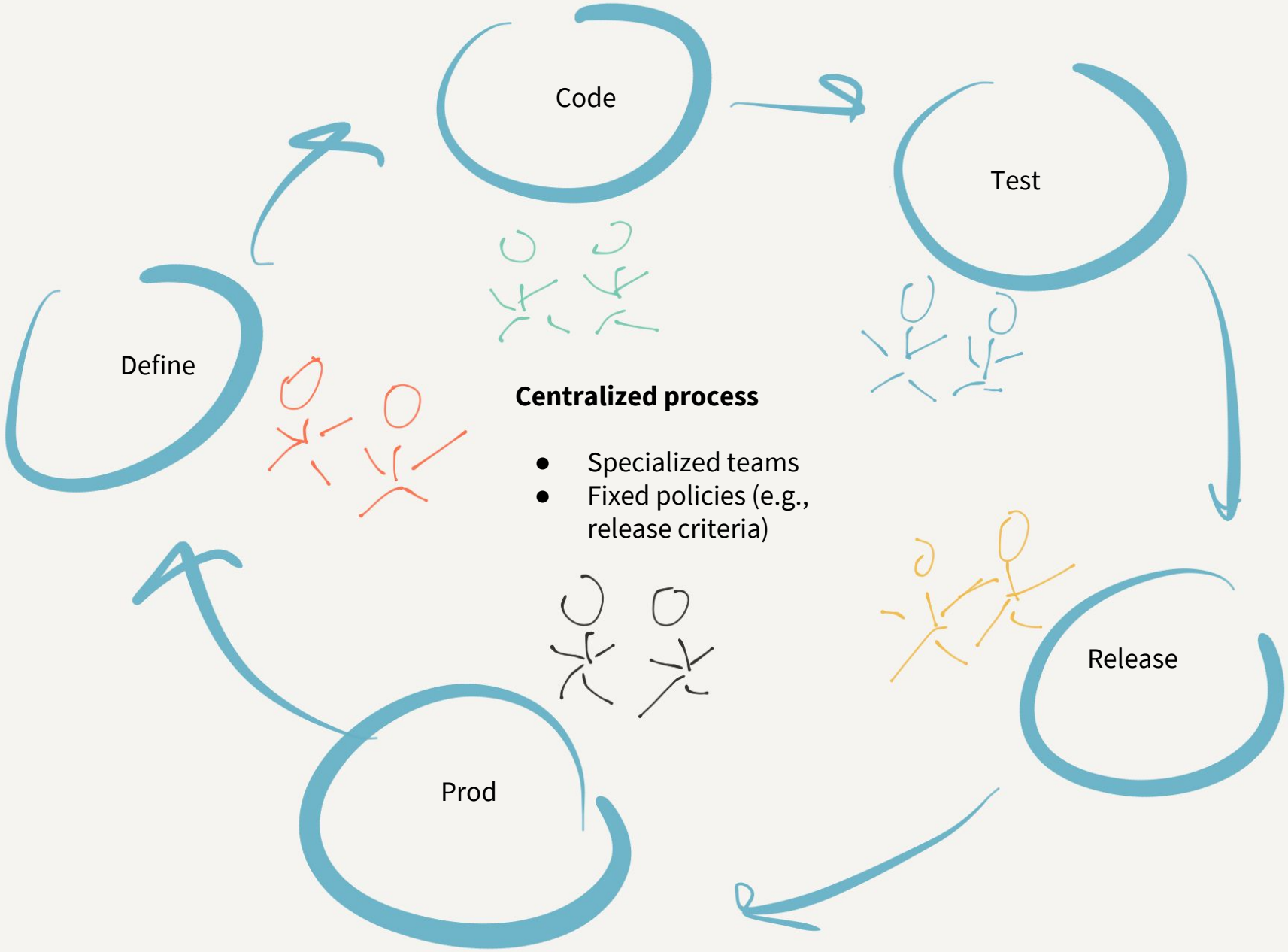

Stability/Maturity
Velocity
Prototype
Production
Mission critical


Stability/Maturity
Velocity
Prototype
Production
Mission critical

A single process is inefficient

(Forces a single Stability vs Velocity Tradeoff)


Code
Test
Release
Prod
Define



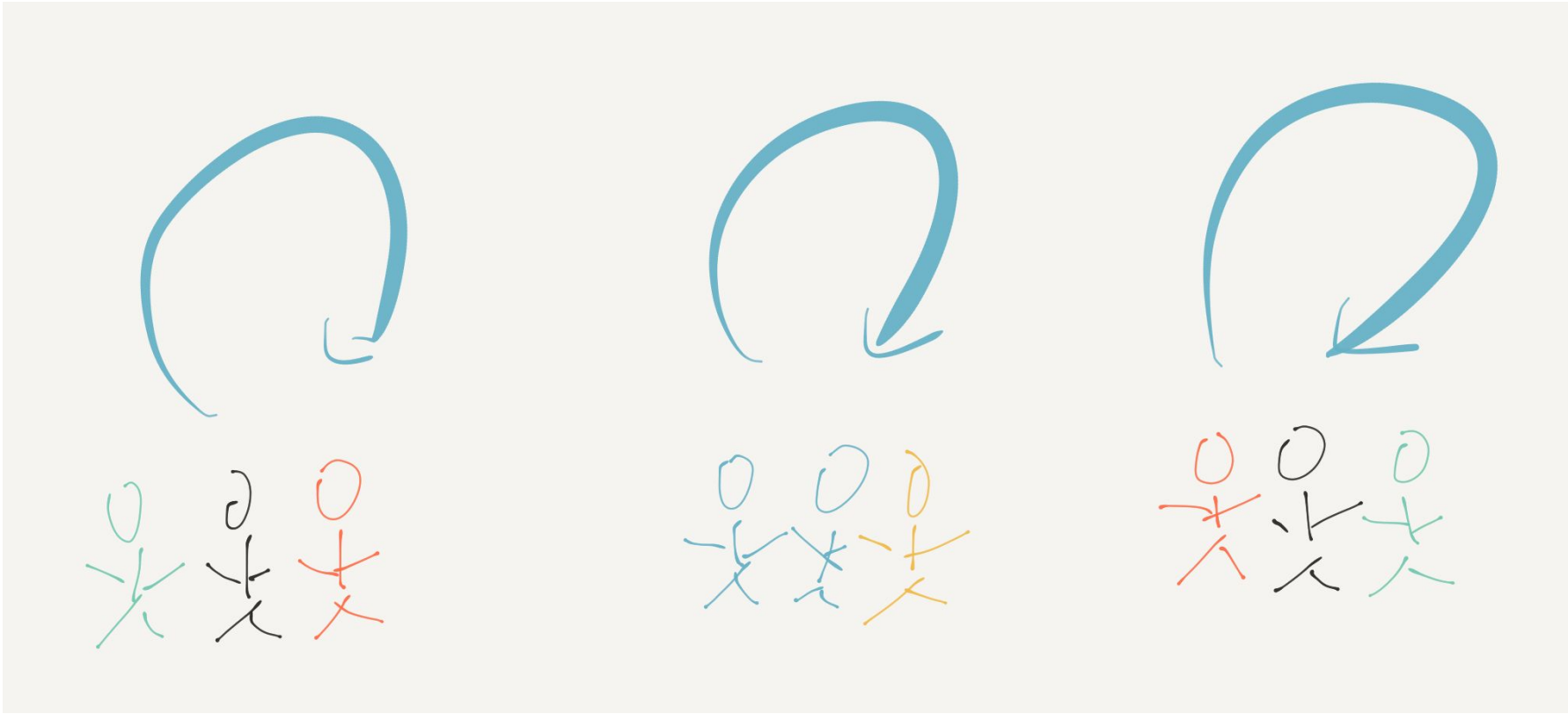
Centralized process

- Specialized teams
- Fixed policies (e.g., release criteria)

A single process doesn’t scale

~~How do I break up my monolith?~~

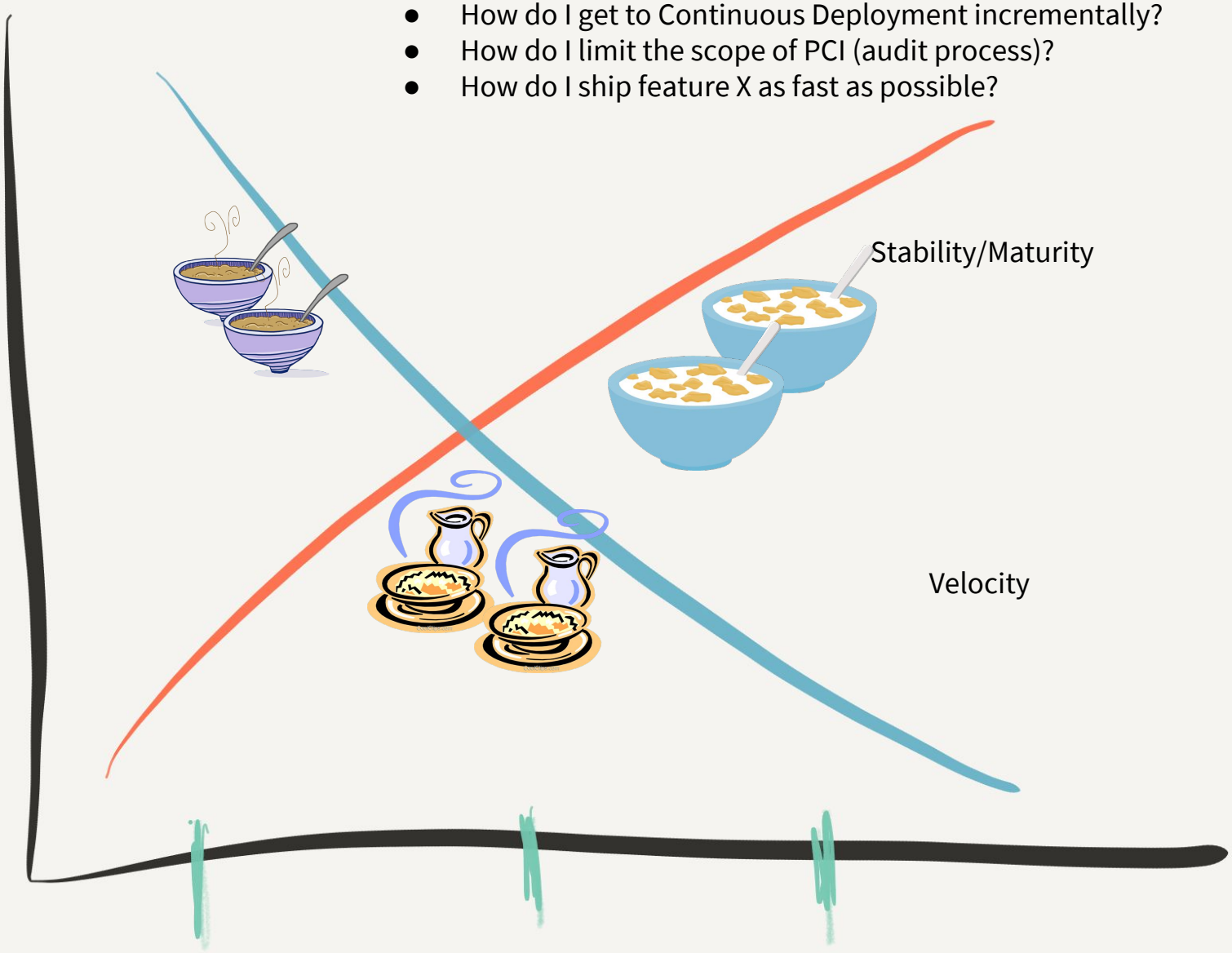
How do I break up my *process*?



Microservices lets you run multiple processes!

Microservices is a distributed development ~~architecture~~ workflow.

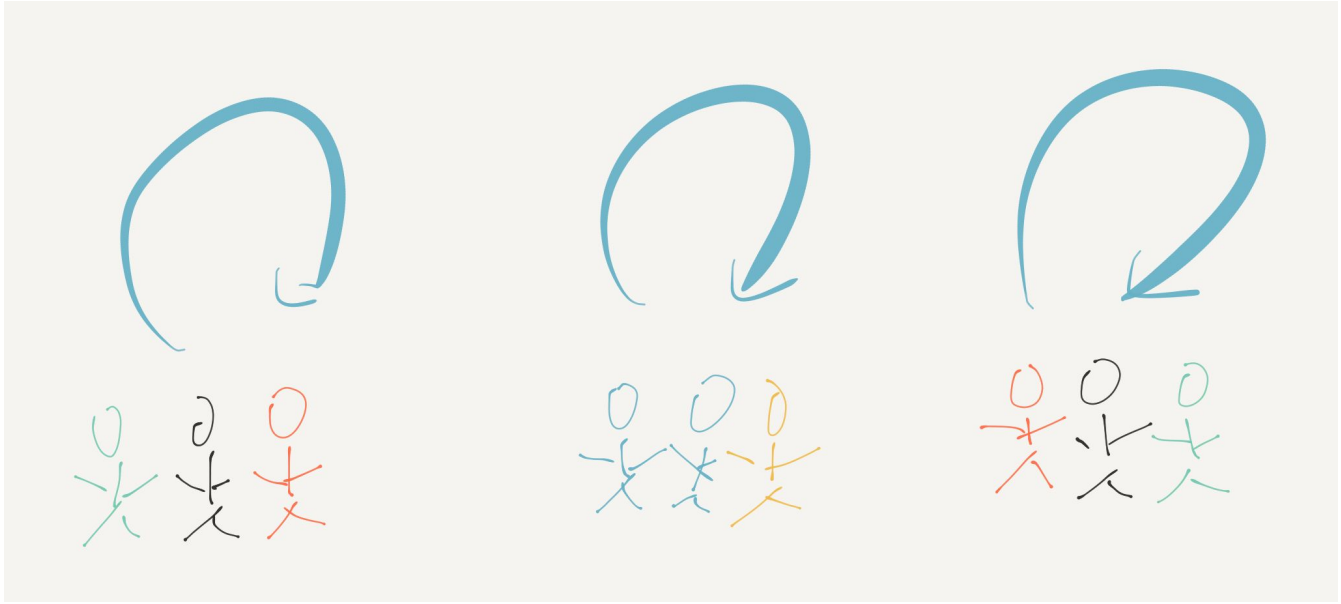
- How do I get to Continuous Deployment incrementally?
- How do I limit the scope of PCI (audit process)?
- How do I ship feature X as fast as possible?



Microservices is ...

- **Multiple** processes
  - Including your existing process!
  - Processes designed for different stability/velocity tradeoffs
- **Simultaneous** processes

Doing things this way shifts how people operate!



- Requires *both* **organizational** and **technical** changes

Organizational Implementation

You gotta give in order to get

Education

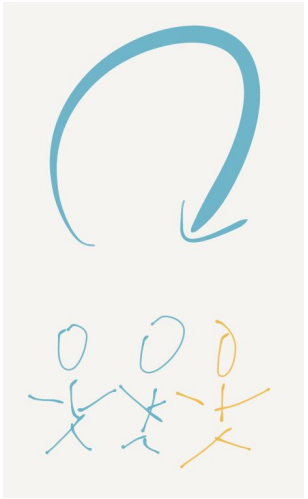
- Everyone exposed to full dev cycle

Communication

- Nobody speaks the same language

Delegation

- Small teams own big important parts



But you get a lot

Education

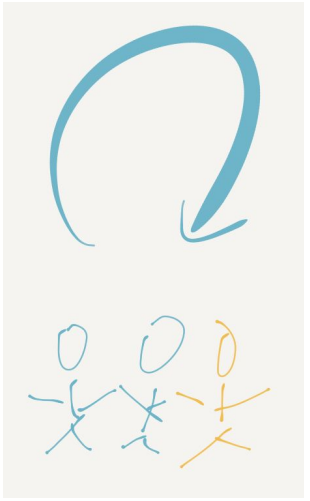
- Specialists become generalists -> Better holistic systems
- Learning, personal growth -> Job satisfaction

Communication

- Conflict -> Collaboration

Delegation

- Massive organizational scale



Create self-sufficient, autonomous software **teams**.

Why self-sufficiency and autonomy?

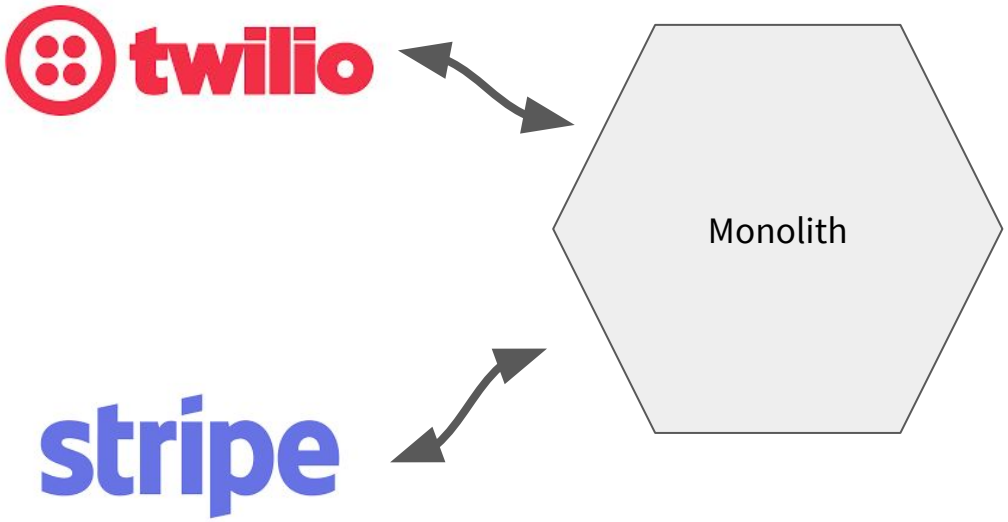
- Self-sufficient
  - Team does not need to rely on other teams to achieve its goals
- Autonomy
  - Team is able to independently make (process) decisions on how to achieve its goals

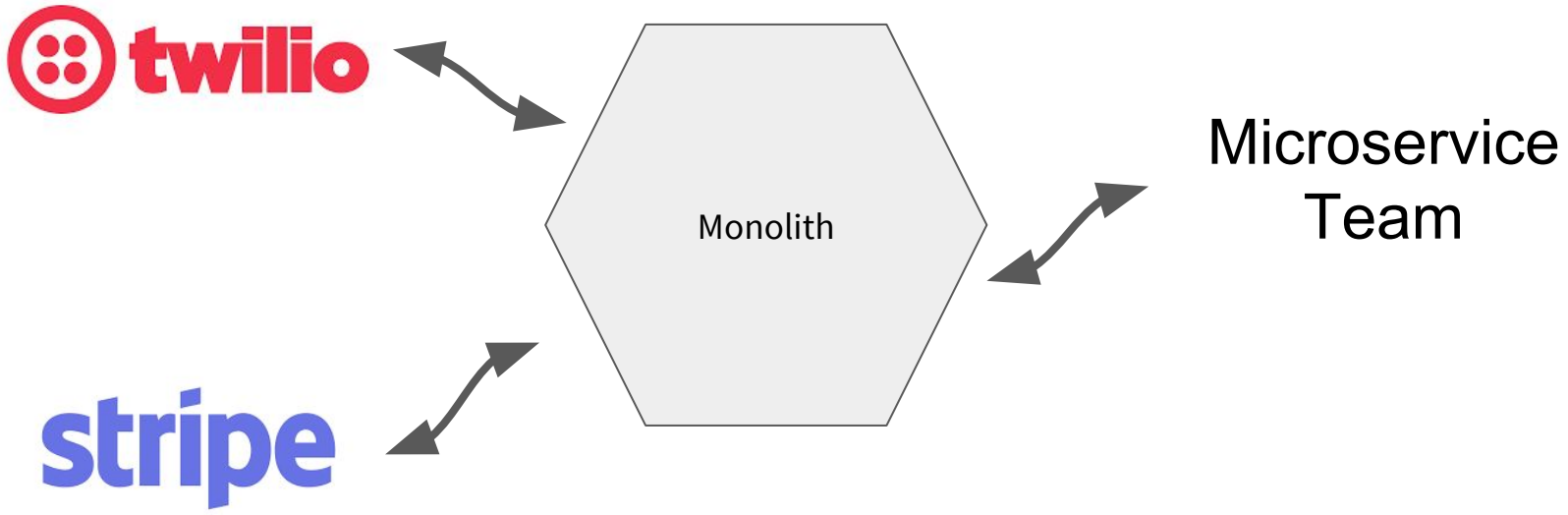
Eliminate centralized specialist functions

Centralized architecture

Centralized infrastructure / ops*
(You might need a platform team)

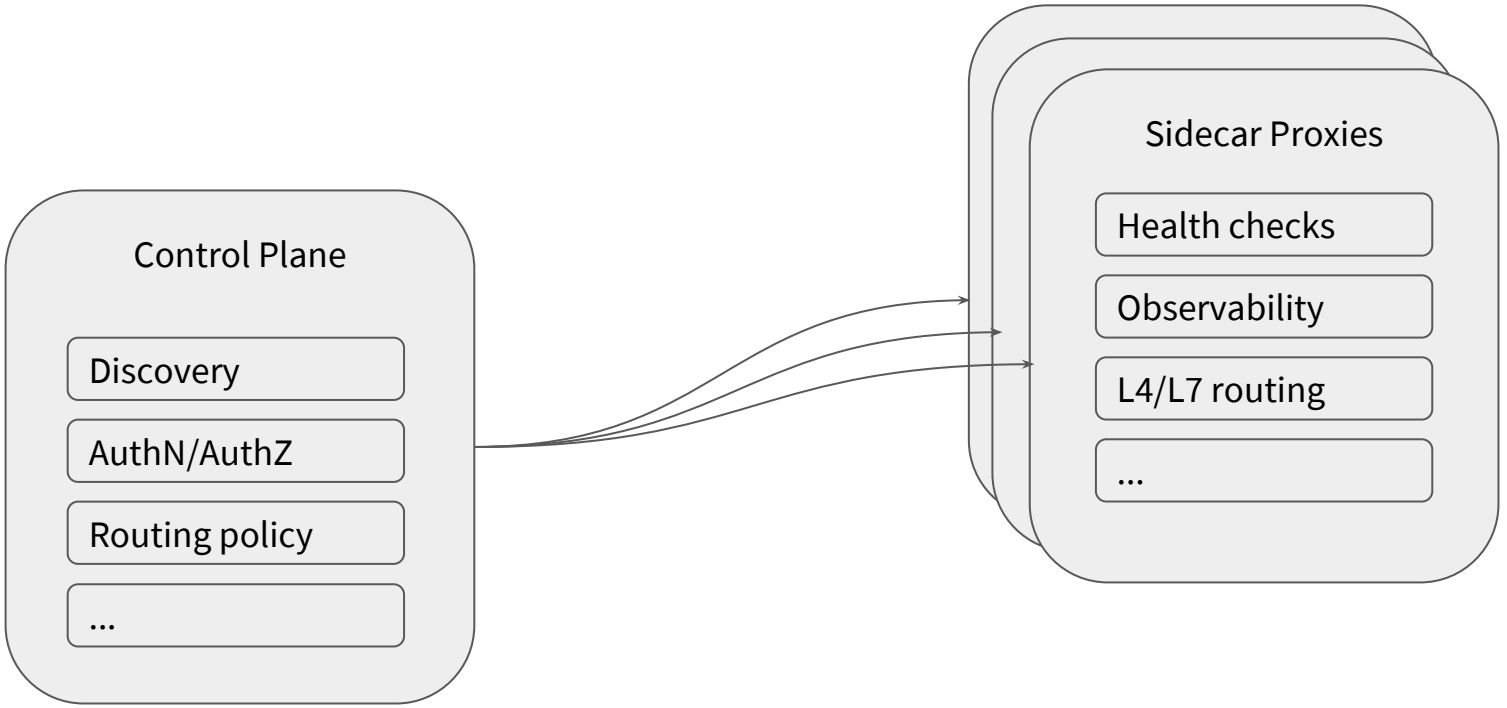
Think Spinoff


twilio
Monolith
stripe

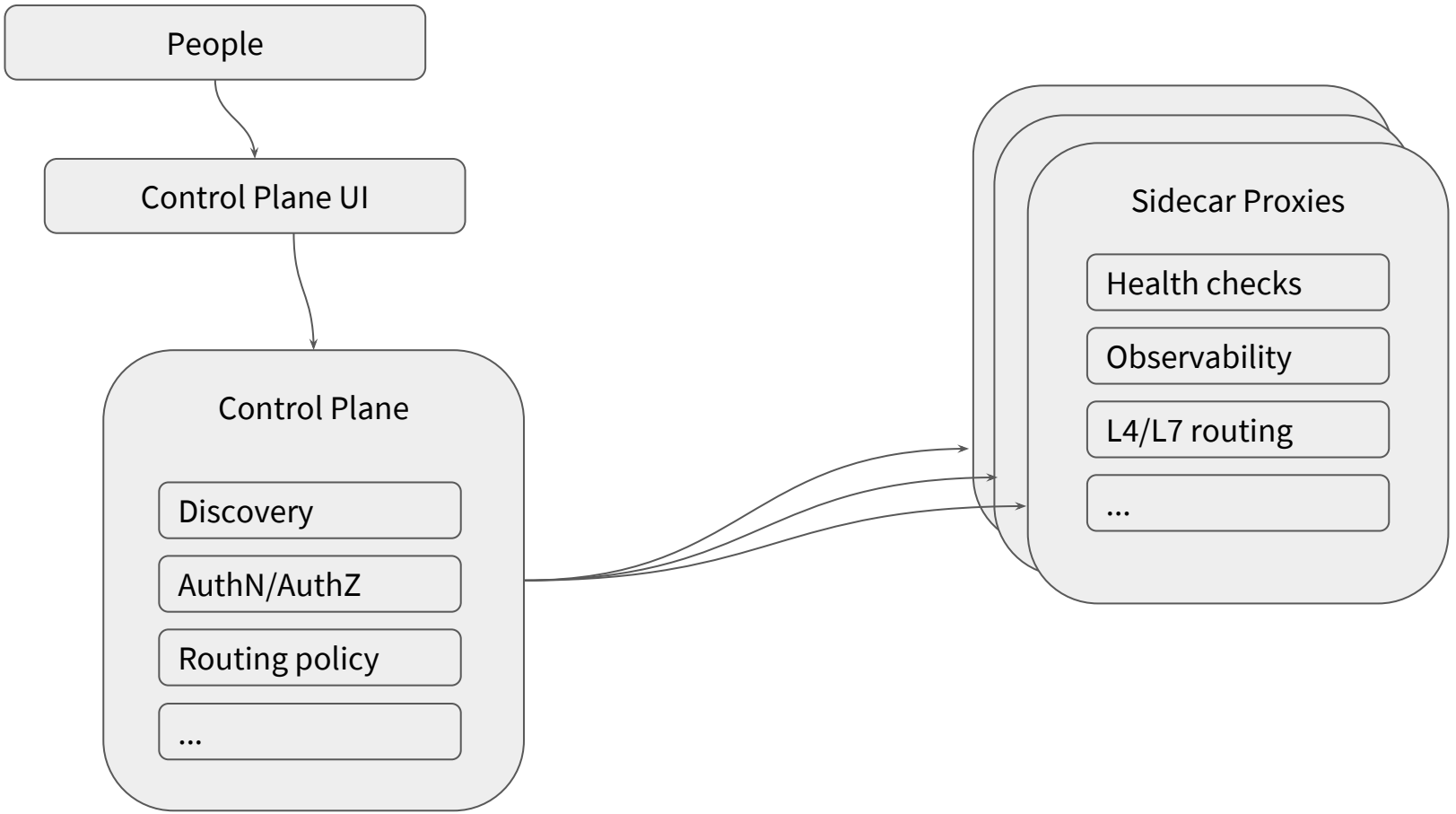

twilio
Monolith
Microservice
Team
stripe

Technical Implementation

Control Plane and Data Plane



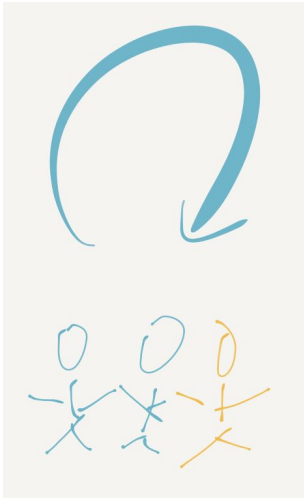
People operate the control plane



Guiding goal: make our team productive

Tools for Generalists

- Generalist UX != Specialist UX
- Specialist: All the things
- Generalist: Simple, Complete, Discoverable



Tools for Education

- Build on familiar concepts
- Safe defaults
- Great feedback

Tools for Autonomous Developers

- Fit multiple processes

The Datawire Story...

- We have been building cloud apps as microservices since 2015

- Way too small to have an actual dedicated ops / platform engineer

The Datawire Story...

- **But...!** Good news I did that in a previous life.

- **Problem...!** I’m supposed to be writing product code.

The Datawire Story...

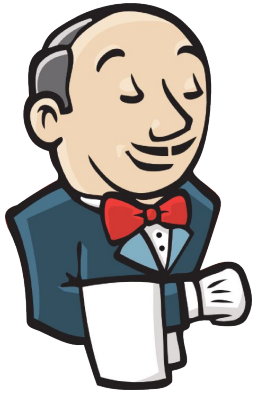
I wanted to do self-service so any developer could scratch their own itch without bugging me.

Some design considerations...

- Polyglot programming shop
- Lots of different toolchains
- Distributed-ish engineering organization
- Not everyone has a strong background in microservices and infrastructure
- Several mission critical services

The Datawire Story...

Also I wanted to do it “right”... so I architected the shit out the problem with tools!



Expected Reaction… :D

… Actual Reaction D:

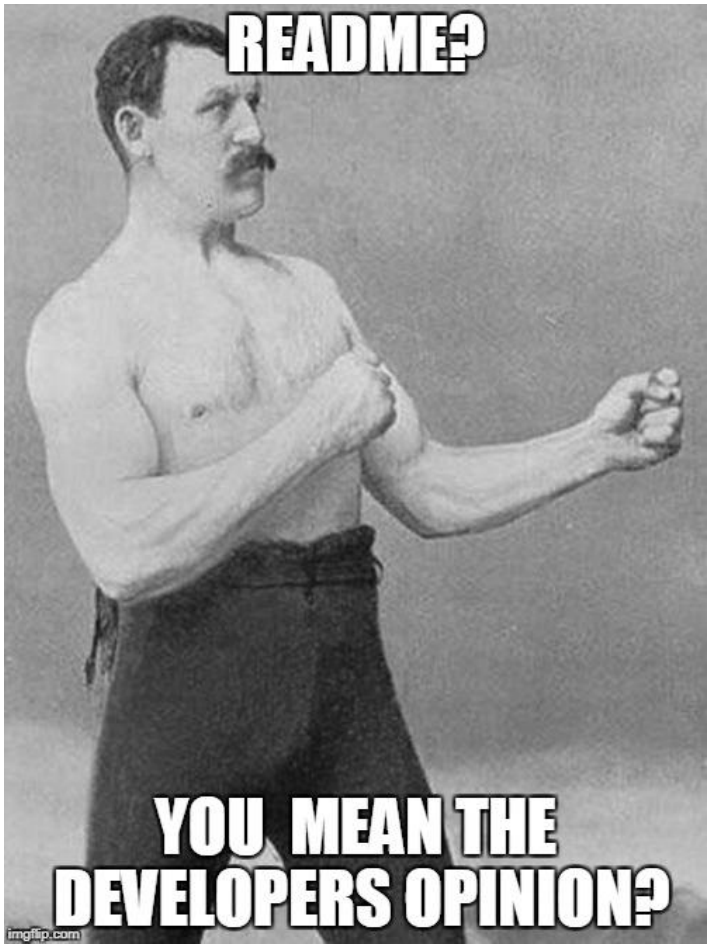
Late 2015… 2016...

Change was slow and painful before adopting better starting principles...

It’s about the people and process, dummy

- Turns out it is not a tooling problem…
- It is a people and process problem…
- Engineers hate adopting new tools when the reason is not compelling… especially if the new tools make their life harder.
- All the tooling in the world does not stop people from continuing their existing behaviors…
- Every tool has a cost because every tool can and will be used slightly differently by different people.

“Did you read the README?” ~ “No, but...”



- A README isn’t enough.
- Developers are bad at RTFM and WTFM.
- Lots of arguments over people doing things differently from how the README stated…
- Lots of arguments also over people not knowing how to do things because the README did not state anything about <X>.

Back to the drawing board...

- We wanted a common build/deployment mechanism that would work for everyone and anything (including CI, multiple languages, etc.)

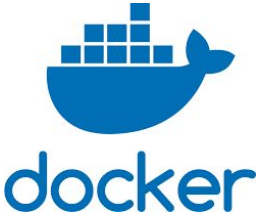
- Started with Docker as it gave a common way to package and ship code.

- Kubernetes had been on my radar since 2014 but seemed finally ready for mass adoption at this point so we picked that to run containers.

Our Kubernetes Journey...

- We built tools based on two things:
  a. Pain
  b. Common patterns we discovered across all our repositories

Step 1: Provide fast repeatable builds for everyone

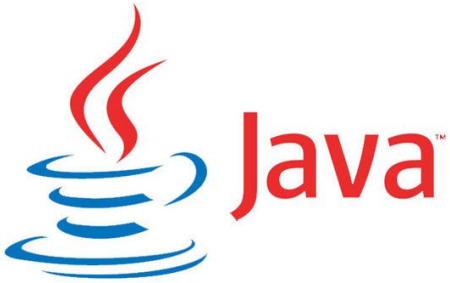


1. Code a change on a branch (e.g. `dev/awesome-feature`)

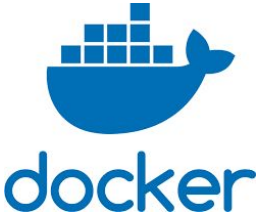
2. Package the build toolchain into the Docker image itself!

3. Run a single command and a new Docker image is built… no need to setup tools on developers machines. Build process is codified into `Dockerfile`.

Step 1: I said everyone and I meant everyone...



Step 1: Provide fast repeatable builds for everyone



1. Tools often forget about making compiled languages compile quickly…

2. Forge assists Docker and ensures things like build caches and incremental compilation work across multiple runs :)

Step 2: Provide fast self-service deploy

- Commit (or don't commit) a change on a branch (e.g. `dev/awesome-feature`)
- Run command and presto…! Updated code is running on the cluster without disturbing any other deployment.
- Support any number of parallel deployments of the same codebase

Step 1 + 2 Recap

- Docker and Kubernetes provide the backbone of build and deployment.

- We found ourselves mostly doing the same thing (`docker build ...`, `kubectl [apply|create] ...`) so we codified it into tools that work fast and have a small learning surface area for devs.

- Nothing “magical” about what the tools do and they can be easily bypassed if necessary.

Step 3: Make it easy to reach the changed code

- Add a little metadata to a Kubernetes service manifest
- Ambassador listens and picks up change and configures Envoy
- Makes it super easy to do stuff like expose a branch (`dev/awesome-feature`) as https://dev-awesome-feature.datawire.io

Perfect?

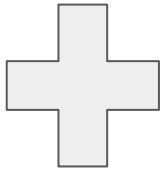
Hell no.

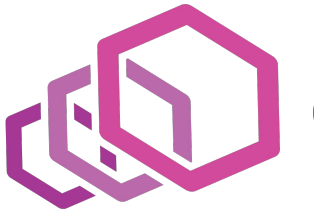
(But it’s soooooo much better than before)

What's left?

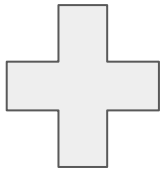
- Self-service bootstrapping
- Self-service stateful infrastructure
- Self-service external to Kubernetes infrastructure (e.g. Amazon RDS)
- Monitoring And Logging

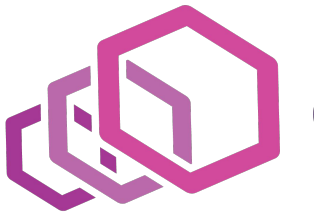
All the Things

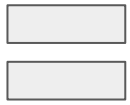




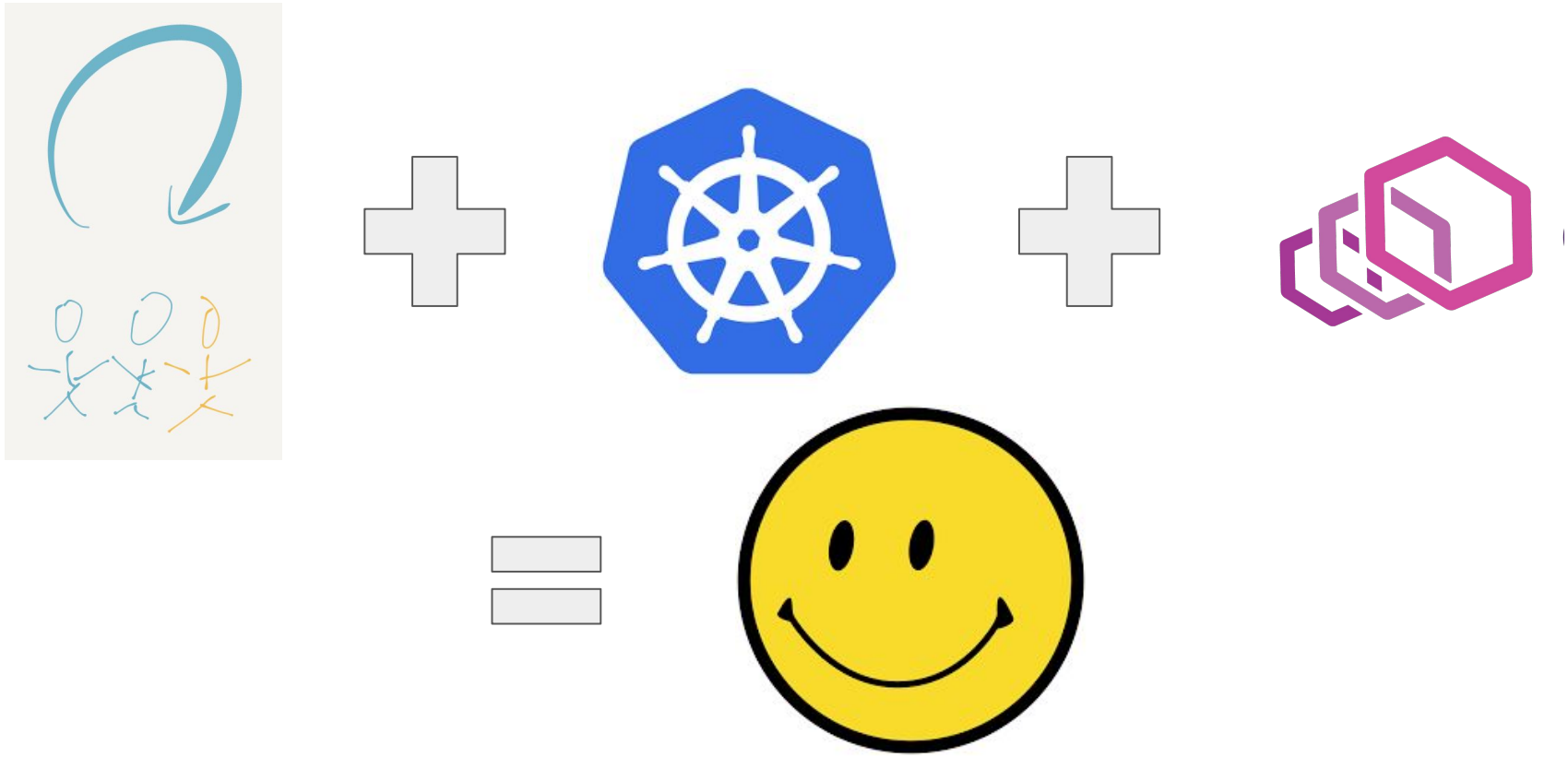
Capabilities aren’t Enough, UX Matters







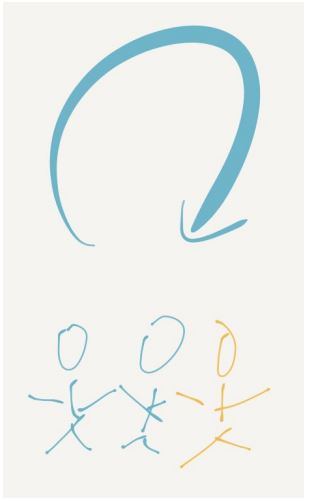
Process and People factors drive the UX



Build the tools your teams need **now**

Tools for Generalists

- Generalist UX != Specialist UX
- Generalist: Simple, Complete, Discoverable



Tools for Education

- Build on familiar concepts
- Safe defaults
- Great feedback

Tools for Autonomous Developers

- Fit multiple processes

Thank you!

Visit us at booth S58

- Cool swag
- Chat about development UX and workflow

Build your own dev workflow

https://www.datawire.io/faster

Tweet Us!

@datawireio
@thebiglombowski
@rschloming

Email Us!

rhs@datawire.io
plombardi@datawire.io