Governance

Microservices Delivery

- Microservices are a collection of *delivery patterns* to support responsive systems delivery and maintenance
- We've already seen key concepts such as:
  - Bounded contexts and business capabilities for incremental service design
  - Service composition as the basis of application development
  - Continuous deployment and Consumer-Driven Contracts
  - Monitoring
  - Security
  - Enterprise architecture and governance
- Now we'll discuss how governance of systems of services works

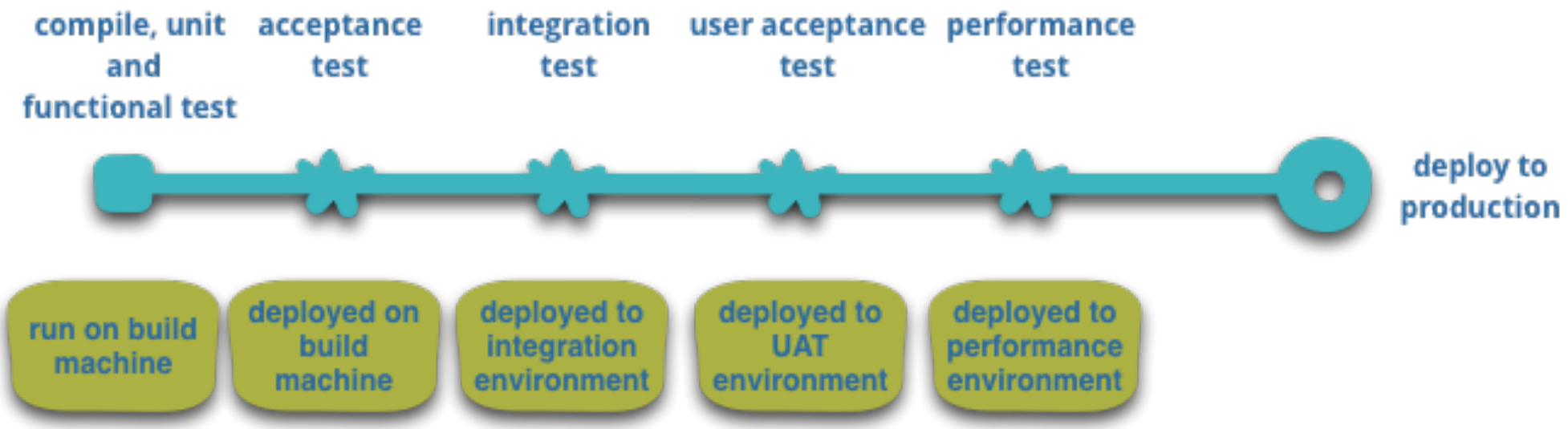
Technical and Business Governance go Hand-in-Hand

- Technical considerations:
  - Individual services are not islands
  - Individual services need to innovate and thrive over the long-haul
  - Technical expertise will differ per-service
- Business needs to see its capabilities reflected in the system-of-services:
  - Investment must be transparent and practical
  - Services live for as long as their business capabilities are required, but evolve with those capabilities

Delivery Automation

- It's the 21st century, so what we're about to talk about is not controversial
- Any software project, including microservices will need:
  - Source control
  - Testing
    - Functional: Unit, functional, integration, UAT…
    - Non-Functional: Security, performance, destructive...
  - Continuous build/continuous delivery

One Build Pipeline Per Service

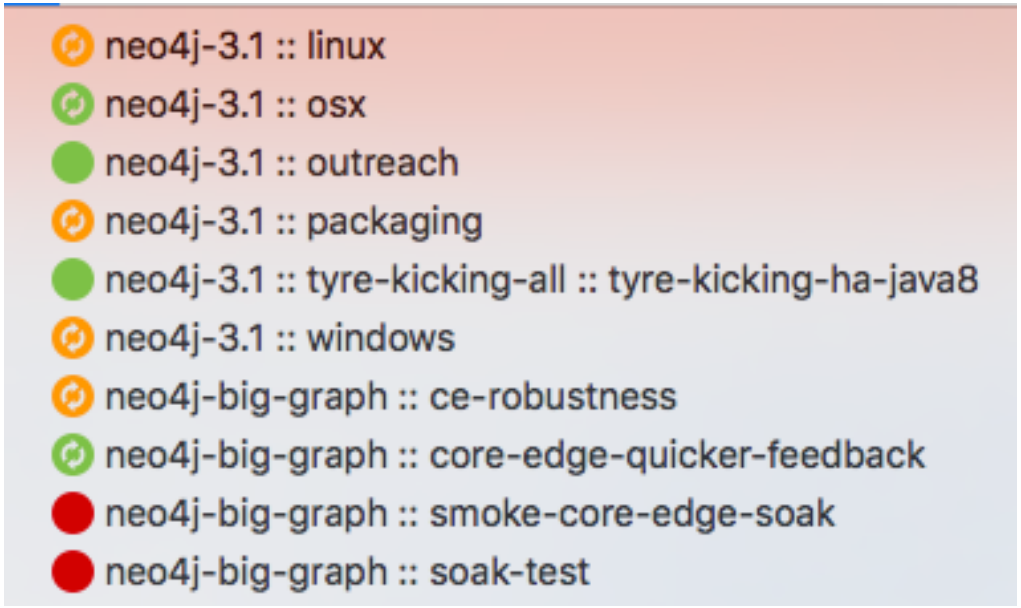


http://martinfowler.com/articles/microservices.html

- For a single service, it's just like a traditional monolith
  - Except you don't have to worry about other pesky teams getting all up in your codebase

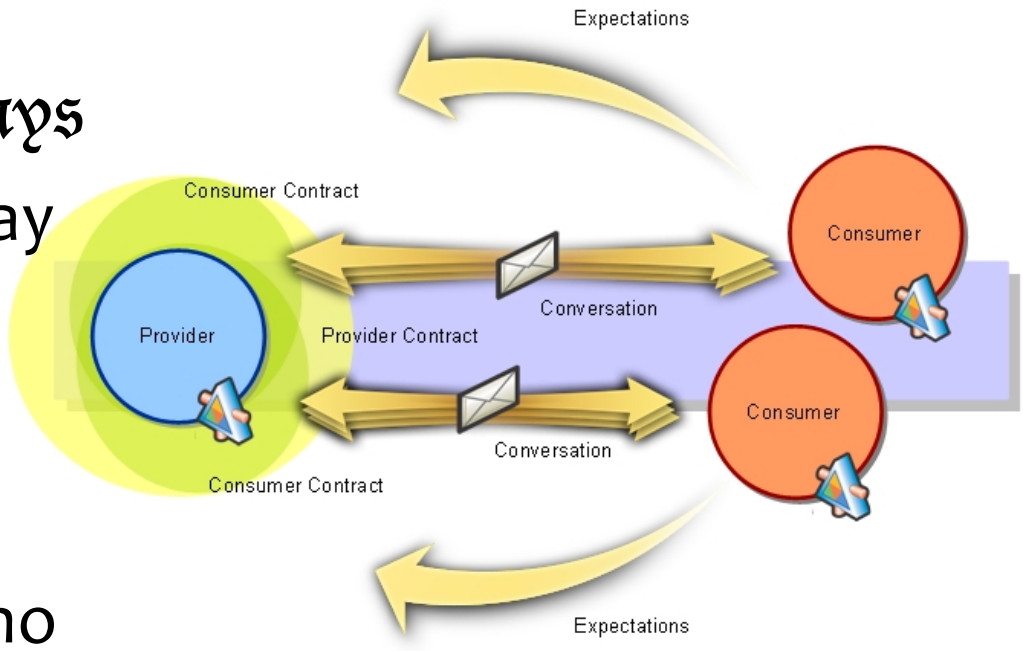
Roll it up

- It's necessary to know the state of your build
  - But insufficient
- At the very least have the top-level builds of all the services visible
  - Dashboard
  - CCTray, CCMenu etc
- Then you've got at-a-glance health check of your services
- Red dependencies aren't necessarily a cause of anger or excitement as we shall see…



Consumer-Driven Contracts

- Services have APIs
  - Used to be called "contracts" in ye olde days
- On the Web, they're one-way
  - Use it, or don't
- In the enterprise we can be more collaborative
  - APIs have consumers who expect continuity



http://martinfowler.com/articles/consumerDrivenContracts.html

Service Contracts

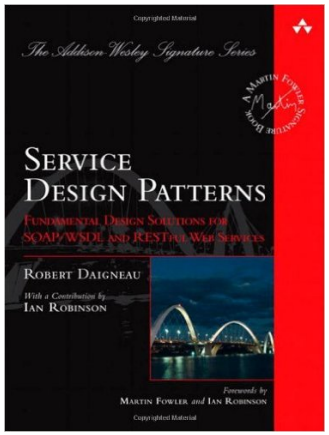
- Normally we think of a contract as the API a service offers
- It (loosely!) guarantees outputs given inputs
- When a service changes so does it's contract
  - We'll shelve versioning discussion for now!
- We've seen this before: IDL, WSDL, WADL (ha!), etc
- It hasn't been pretty because it left the clients with no say in servers' evolution
- And clients are often **important systems**

Client-Server Interactions

- Two areas of focus:
  - What a server ***offers***
  - What a client ***consumes***
- Service contracts typically focus on what a service offers
- Consumer-Driven Contracts focus on the intersection of these two areas

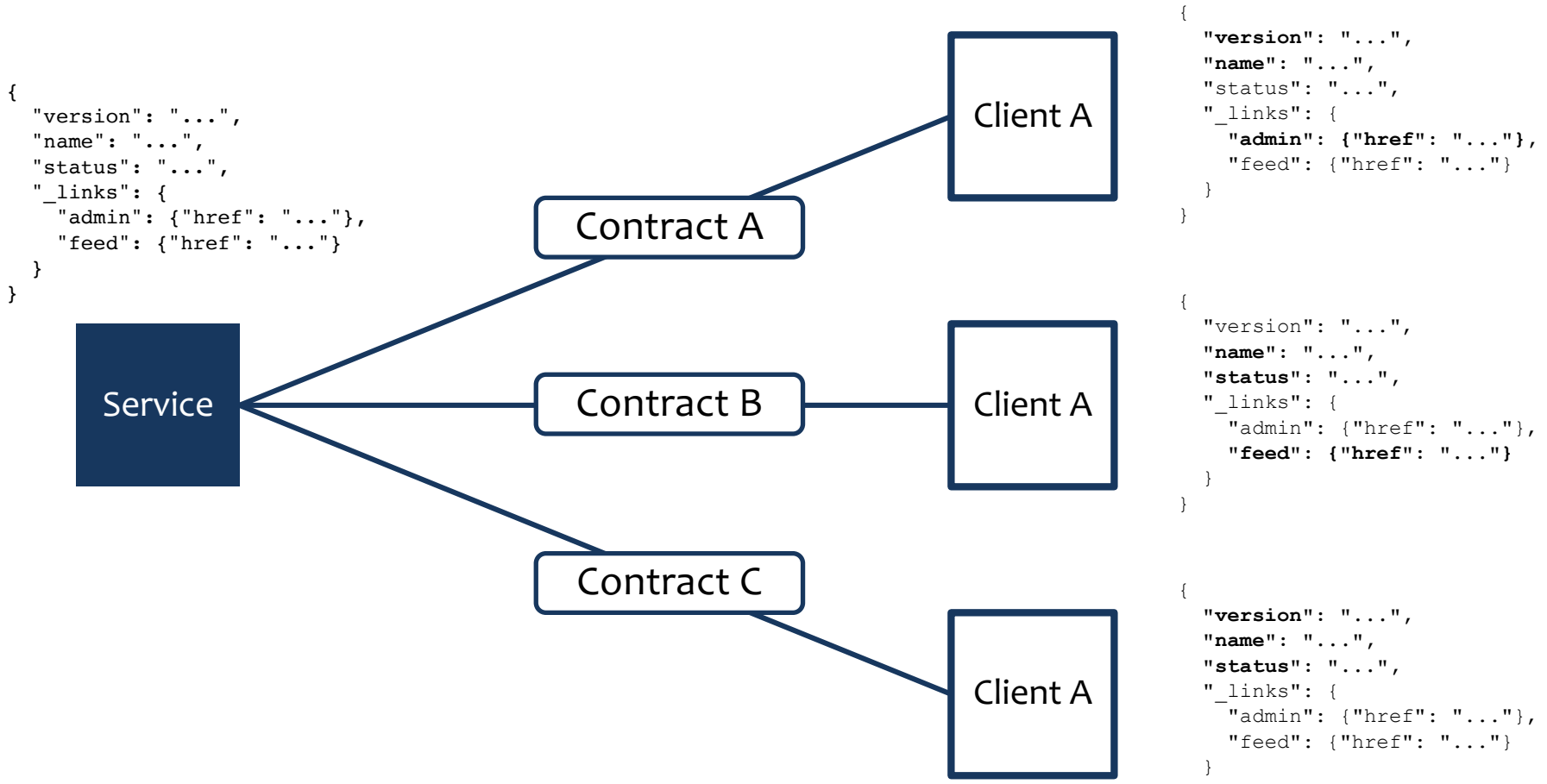
Consumer-Driven Contracts

- Clients write consumer tests
  - Describe which parts of the service API they actually use
- Servers incorporate these tests into their test suites
  - Aggregate view of how they are actually being consumed

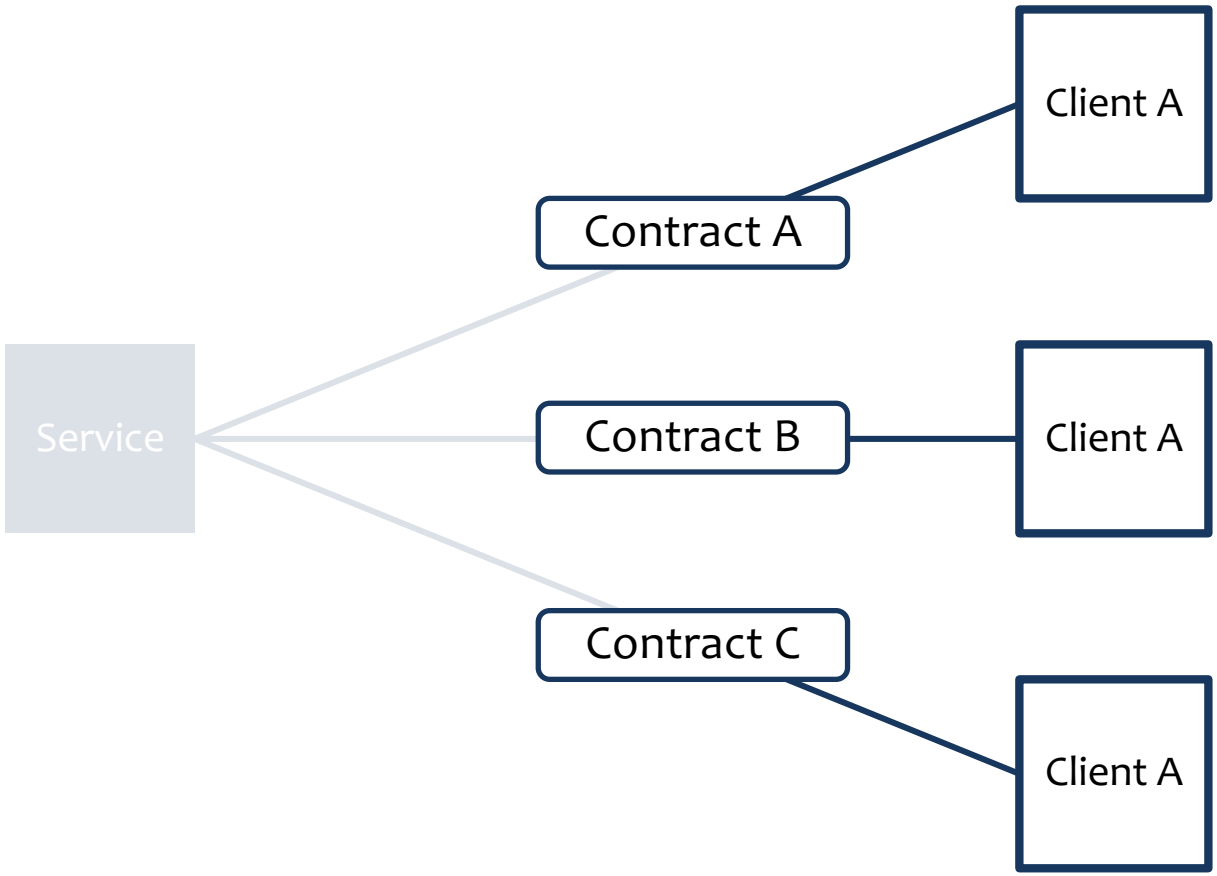


http://martinfowler.com/articles/consumerDrivenContracts.html

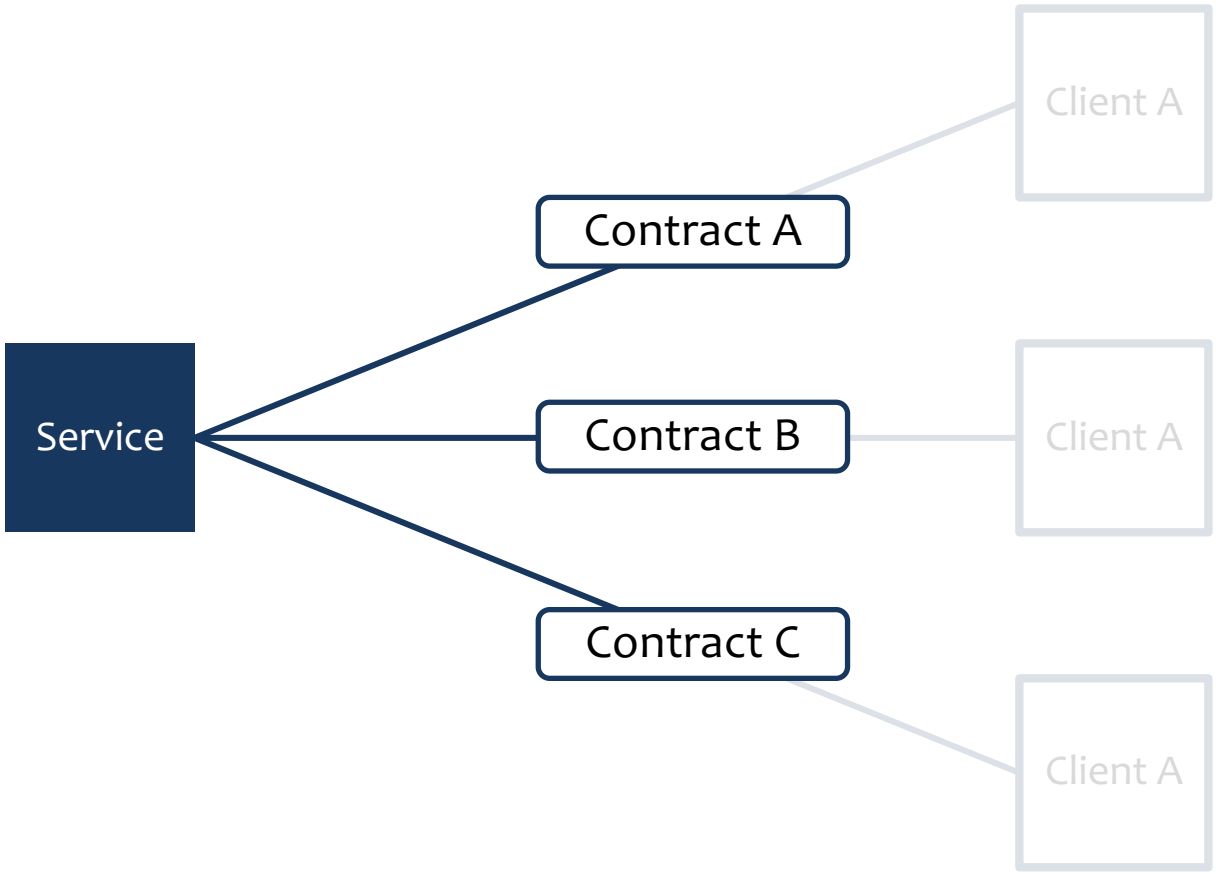
Example


{
"version": "...",
"name": "...",
"status": "...",
"_links": {
"admin": {"href": "..."},
"feed": {"href": "..."}
}
}
Service
Contract A
Contract B
Contract C
Client A
Client A
Client A
{
"version": "...",
"name": "...",
"status": "...",
"_links": {
"admin": {"href": "..."},
"feed": {"href": "..."}
}
}
{
"version": "...",
"name": "...",
"status": "...",
"_links": {
"admin": {"href": "..."},
"feed": {"href": "..."}
}
}
{
"version": "...",
"name": "...",
"status": "...",
"_links": {
"admin": {"href": "..."},
"feed": {"href": "..."}
}
}

Client Tests



Server Tests



Contract Assertions

- Schema/structure
  - Fields
  - Data types (string, number, list, etc)
  - *Specific values*
    - *Specify inputs and expected outputs*
- Protocol
  - Hypermedia controls
  - Semantic annotations (e.g. link relations)
  - HTTP idioms

Benefits of Consumer-Driven Contracts

- Service providers understand ***how*** their service is being used
  - Which parts of the interface
  - Which parts of the protocol
- Help identity breaking and non-breaking changes
  - Plan evolution of service
- Allow consumers to guide the evolution of a service
  - Encourages interactions between teams

Hypermedia and Consumer-Driven Contracts

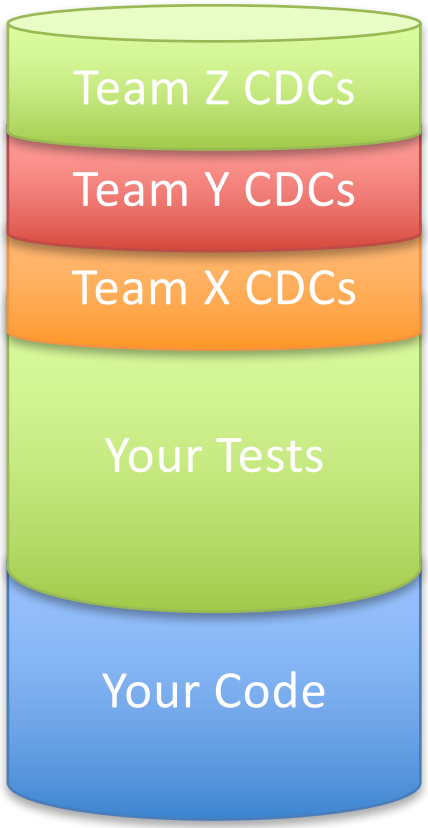
- Hypermedia representations contain 2 kinds of data:
  - Resource state
  - Permissible state transitions
- We've discussed this previously in terms of the distinction between ***interface*** and ***communication protocol***
  - Protocol is disclosed as data in a self-descriptive message
- Consumer-Driven Contracts for hypermedia-based services therefore allow us to assert the consumer's expectations regarding both interface and protocol

Implementation Options

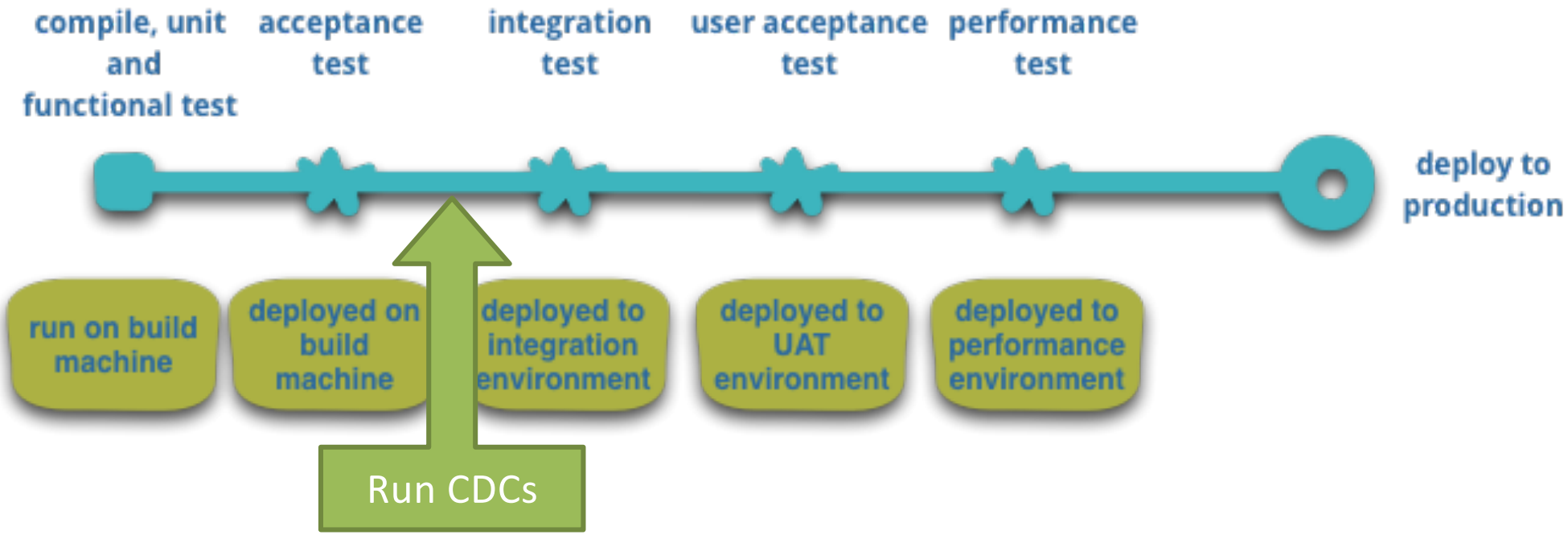
- Schematron
  - Language for making assertions about the presence or absence of patterns in XML documents
  - *http://www.schematron.com/*
- Several libraries for asserting JSON paths
  - *https://github.com/stsvilik/jPath*
- Consumer-Driven Contract tools:
  - *https://github.com/realestate-com-au/pact*
  - *https://github.com/thoughtworks/pacto*
  - *https://github.com/afacanerman/api-valter*
- Git submodules
  - For hosting and aggregating contracts

Their Contracts, Your Repo

- Other teams can add tests to your repo
- Caveat: they are only allowed to test the API
  - Format, syntax, semantics
- These tests must pass
- Failing tests indicate a failure of services to communicate in production



Collaborating with Consumer-Driven Contracts



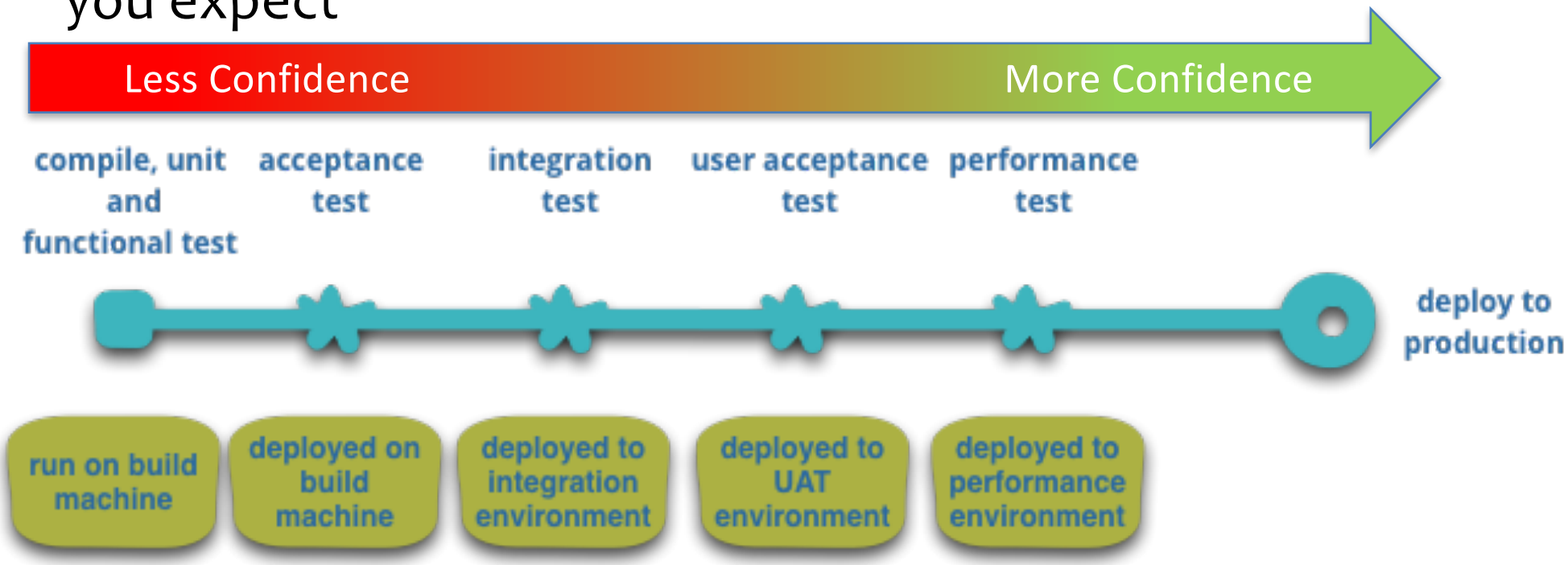
- Run everyone else's CDCs as part of your build
- They exercise your API holistically
- If you break a test, it triggers a conversation with its owner

When “Their” Tests Break “Your” Build

- When a CDC breaks, it means you have evolved the API in a way that your consumers weren’t expecting
  - Don’t mute them
  - Don’t delete them
  - For the love of `$DEITY` don’t replace their assertions with `assertTrue(true)`
    - They’re not your tests!
- Time to talk
  - They might change their CDCs and their code
  - They might ask for a finely-grained versioned API
  - You might convince them to upgrade
  - You might throw yourselves on the mercy of a higher authority…
- But only if you talk

Getting more from your CDCs

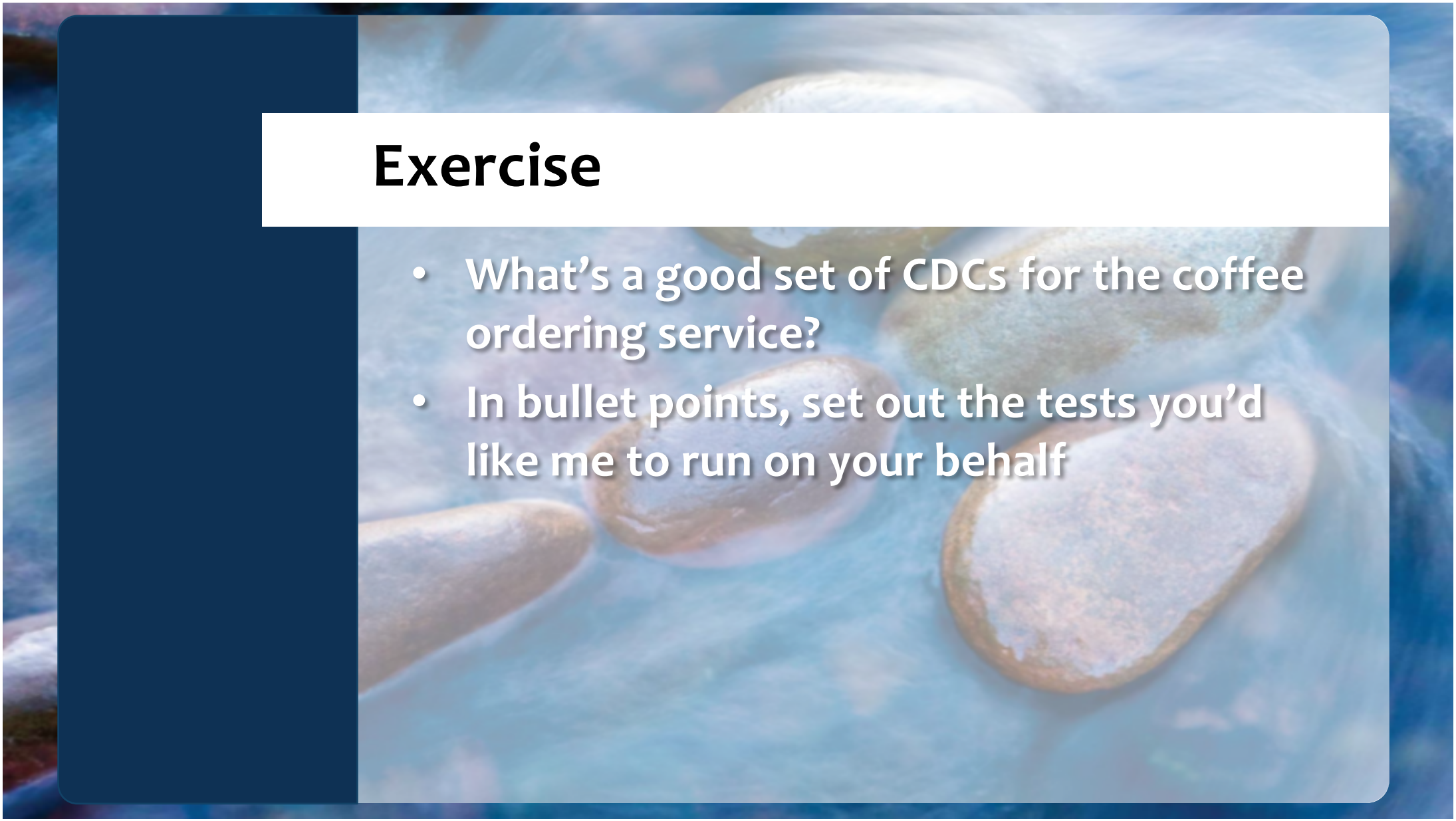
- Your build pipeline isn't just about behavioural correctness
- It's about building confidence that a service can operate at you expect



What Else Do Consumers Need Confidence In?

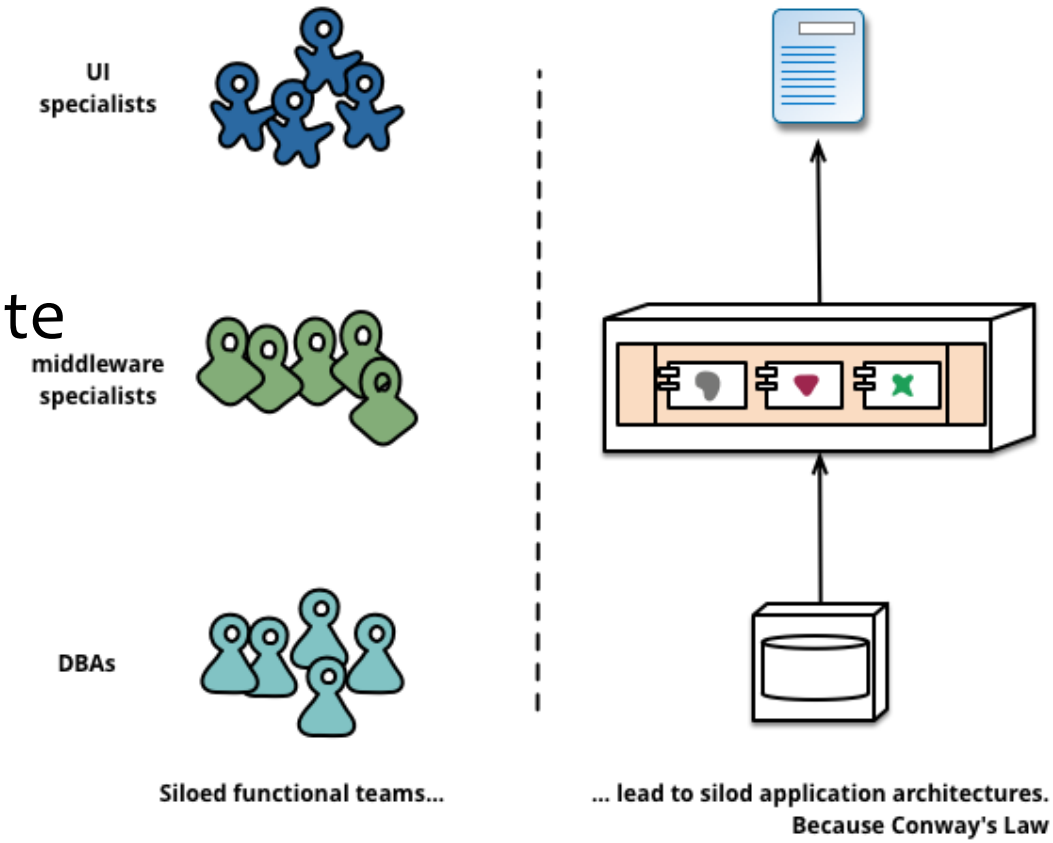
- We already know consumers expect stable:
  - Syntax
  - Formats
  - Representations
  - Hypermedia controls
  - Etc
- But what else is needed to have confidence for dependable service composition?
  - Latency
    - E.g. 95% of requests are satisfied in 30ms
  - Uptime
    - E.g. 1 hour of downtime per year
  - Volume
    - E.g. service can handle 10,000 req/sec at 30ms latency
- These are part of the service contract too!

https://mikehooklaw.files.wordpress.com/2014/07/man-reading-a-contract-with-magnifying-glass-clipart-gif?w=469


Exercise
What's a good set of CDCs for the coffee ordering service?
In bullet points, set out the tests you'd like me to run on your behalf

Horizontal Delivery Teams: Anti-pattern

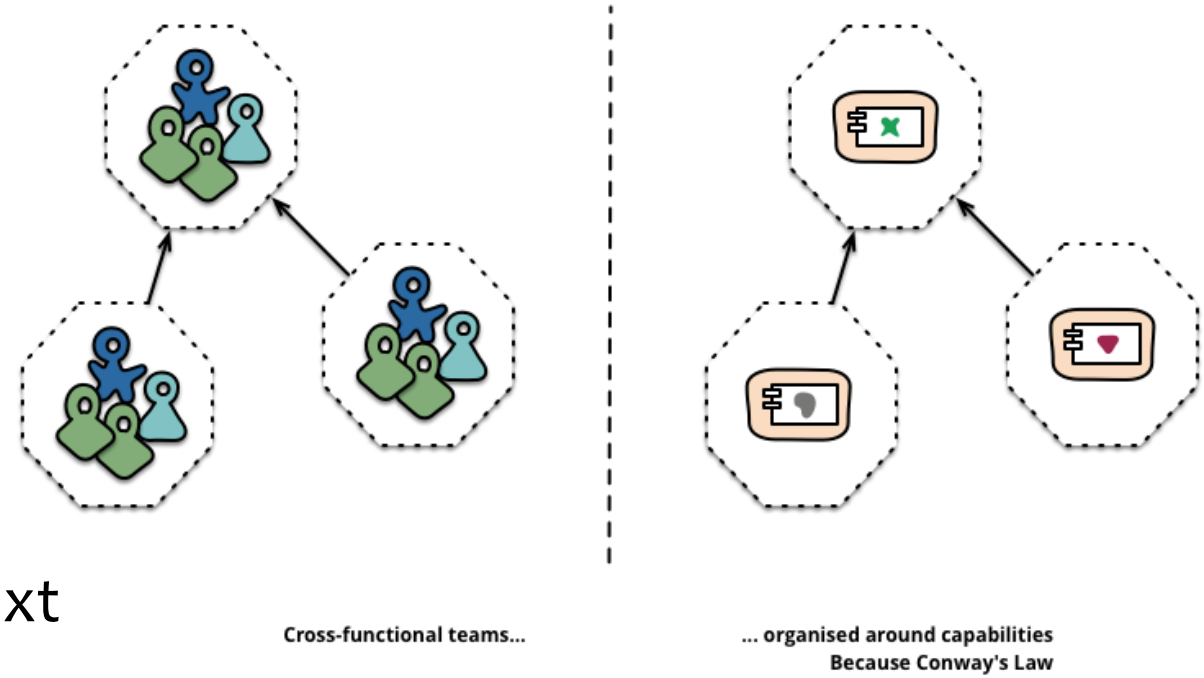
- Exclusively horizontal teams are sub-optimal
- Create silos
- Unable to own and operate a service independently
- Fertile ground for finger-pointing and acrimony



http://martinfowler.com/articles/microservices.html

Organise Along Business Capabilities

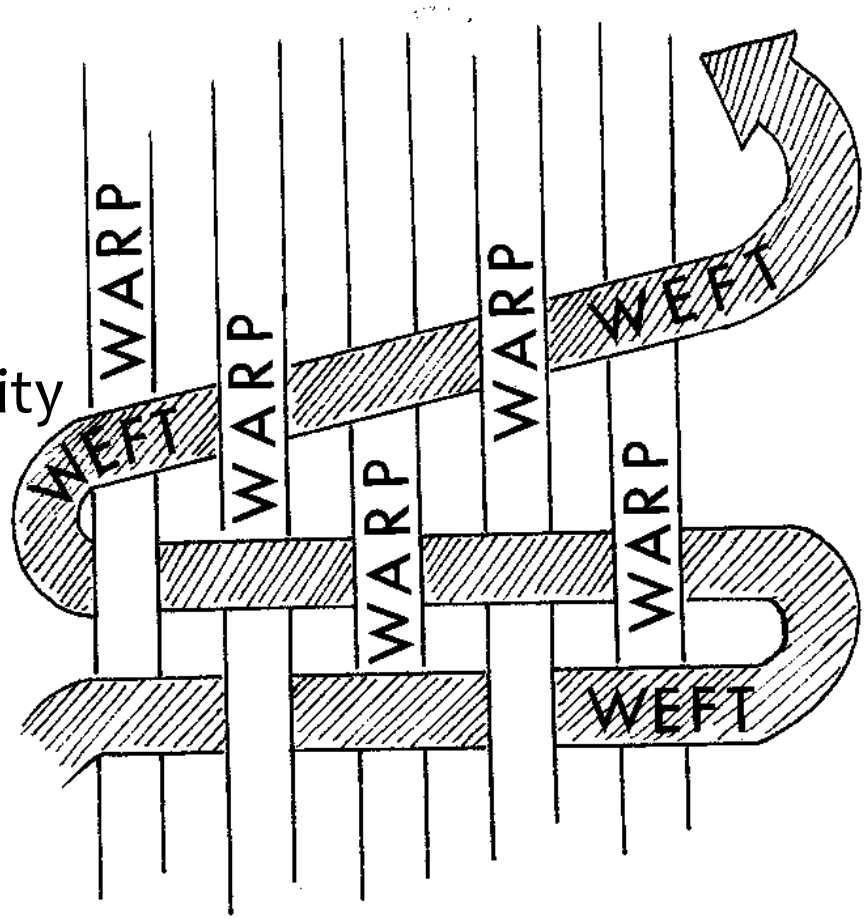
- Each service needs specialists
- Cross-functional teams are normal
  - PMs, Devs, Testers, BAs, Architects, etc
- Expertise can still be injected
  - See warp-n-weft next
- Teams organise along business capabilities



http://martinfowler.com/articles/microservices.html

Warp-n-Weft

- Services are our *warp*
  - Long lived
  - Have guardians
  - Have business owners
  - Encapsulate a business capability
- Projects are our *weft*
  - Short lived
  - Have business sponsors
  - Deliver investment to services
  - Interact with service guardians



https://jmarks001.files.wordpress.com/2015/04/warp-and-weft.gif

Are you a warpie or a weftie?

- Warpies own services
  - Deep, detailed knowledge of a narrow domain and technical implementation
  - Enjoy living with and operating a service for the long haul
- Wefties deliver projects
  - Broad business domain and technical knowledge
  - Enjoy traditional “projects” with discrete handoff and variety

Warpies and Wefties

- You need both warpies and wefties to deliver and run an ongoing system of services
- Allow for folks to switch between warp and weft roles
  - At a modest pace
- Allow warpies to move between services
  - At a modest pace
- Benefits:
  - Wefties will be warpies. Nobody wants a 4am wake up.
  - Warpies will be wefties. Nobody wants bullshit politics.

Summary

- Microservices are foremost a delivery pattern
- Built automation is critical
- Consumer-Driven Contracts provide agility, confidence
- Warp-n-weft pattern for service owners and investment via projects