Where do MicroServices come from?

An approach to building and deploying applications

that evolved from organizations
facing issues with monolithic approach

when their applications grew, became more complex, needed
to scale, and development velocity was slowing

- Jeff Bezos mandate, 2002:
  - *All teams will henceforth expose their data and functionality through service interfaces.*
  - *Teams must communicate with each other through these interfaces.*
  - *There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.*

- Werner Vogels: you build it, you run it

Netflix

Adrian Cockcroft:

MicroServices grew out of our culture, and the way that we organized work into small independent teams

Characteristics of MicroService Architectures

Break deployment dependencies

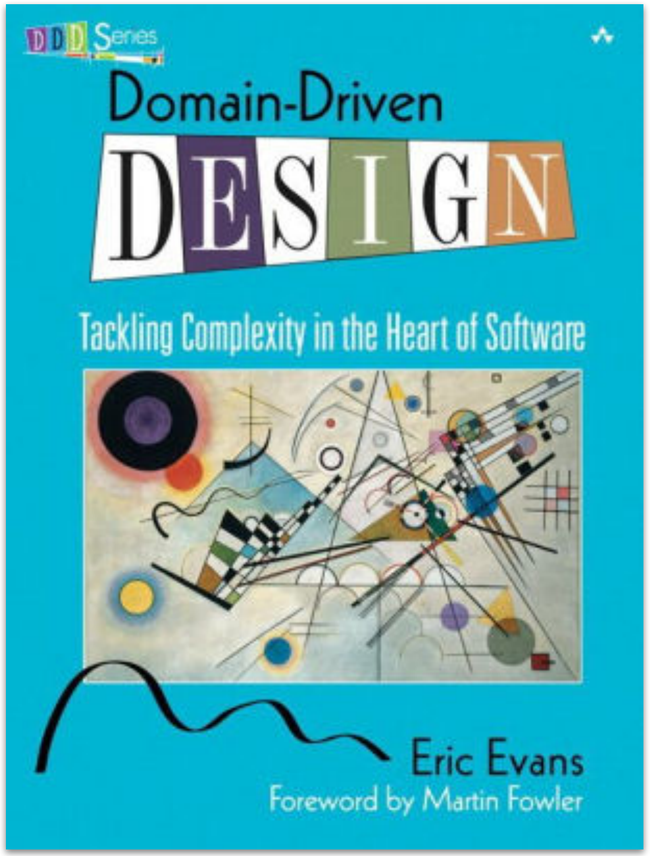
Each team works on..
a different codebase,
different version control repository,
a different app, deployed on its own schedule,
without the need to coordinate its work with other teams

Questions

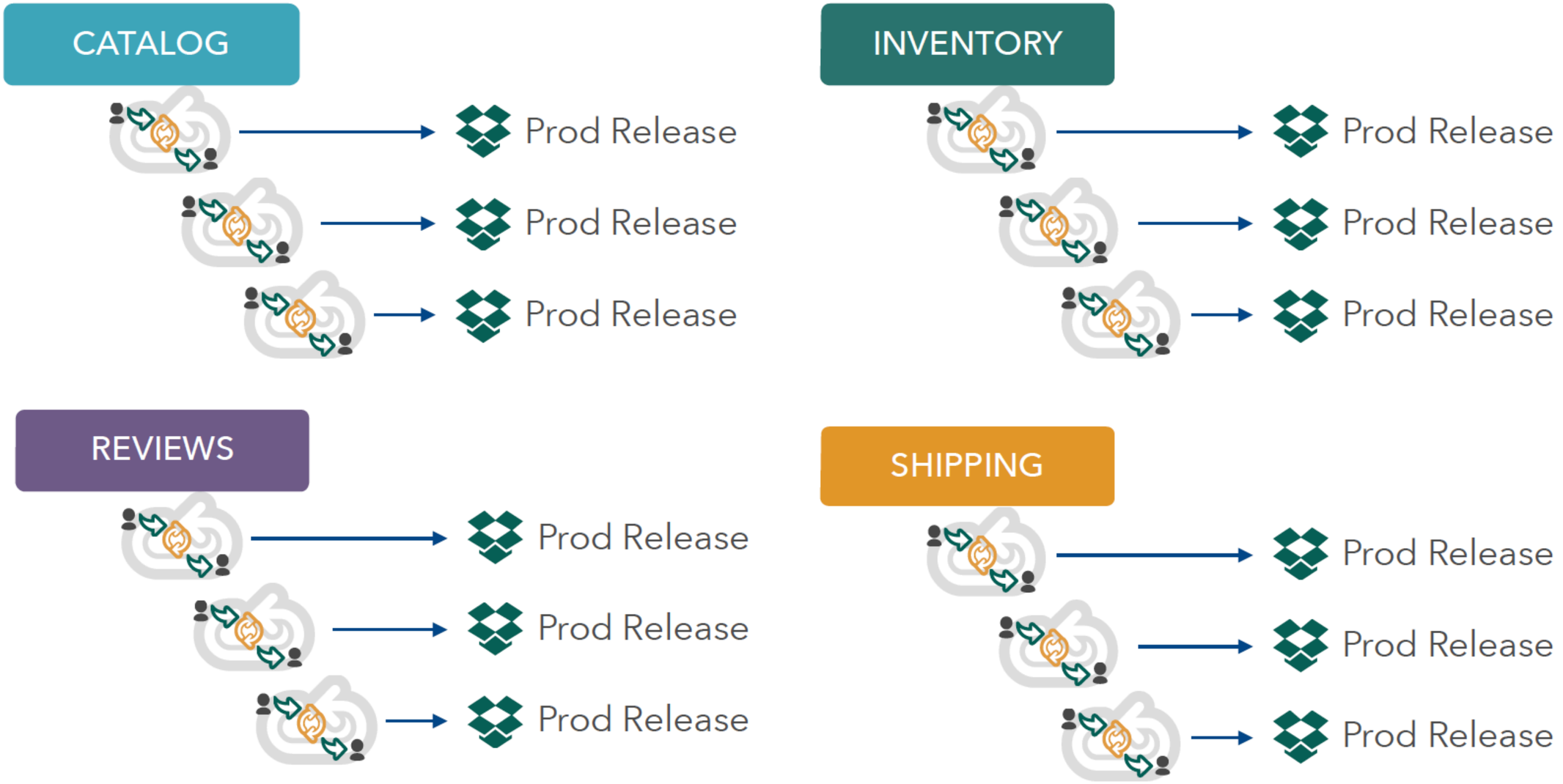
- Along what lines do we break the larger application into multiple smaller applications?
- What is the composition of the team?
- How do these smaller applications communicate with one another?
- How do I transition from my current monolithic application to this new model?

Domain-Driven Design (DDD)

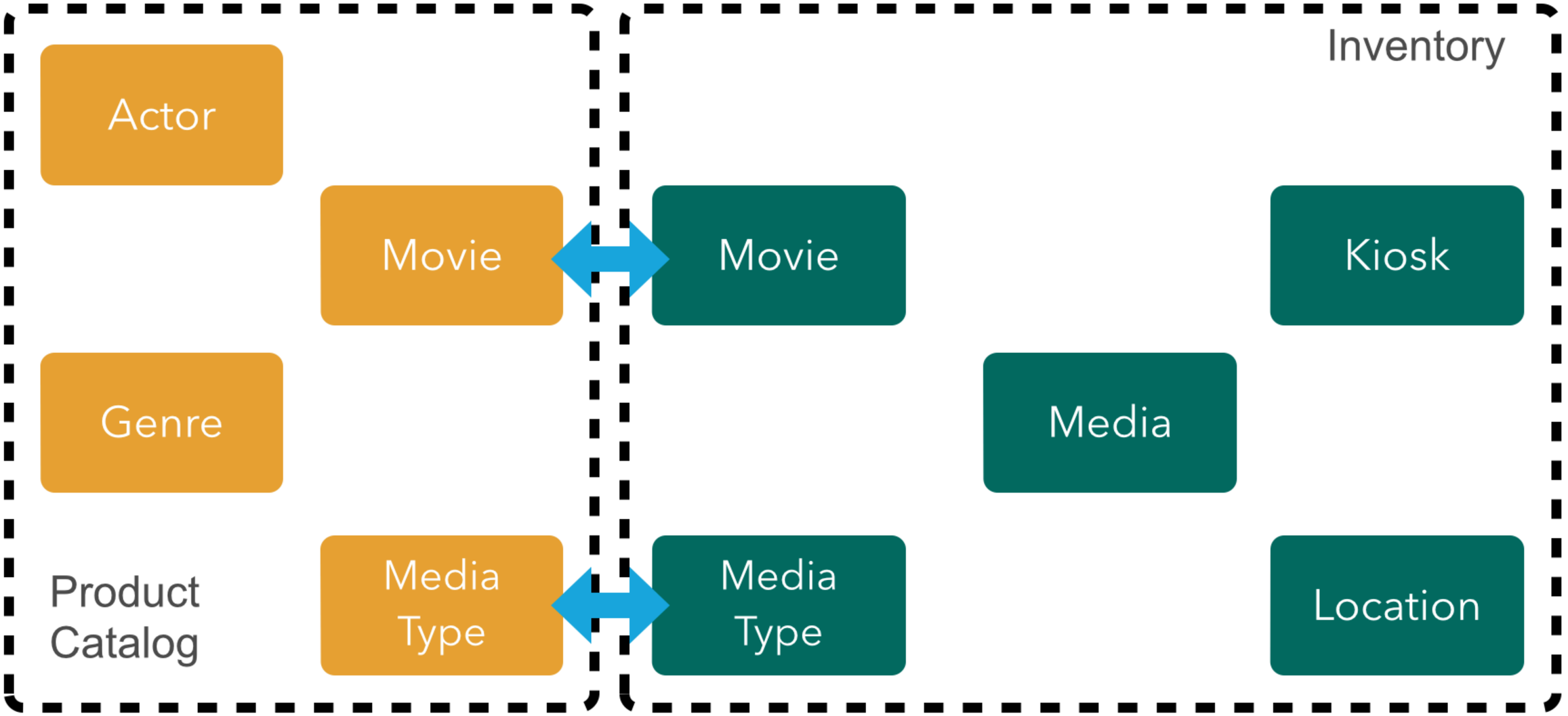
- Book by Eric Evans, 2003
- Modeling the Domain, aggregate roots, bounded contexts, anticorruption layers, and more..



Business Capability Teams



Bounded Contexts

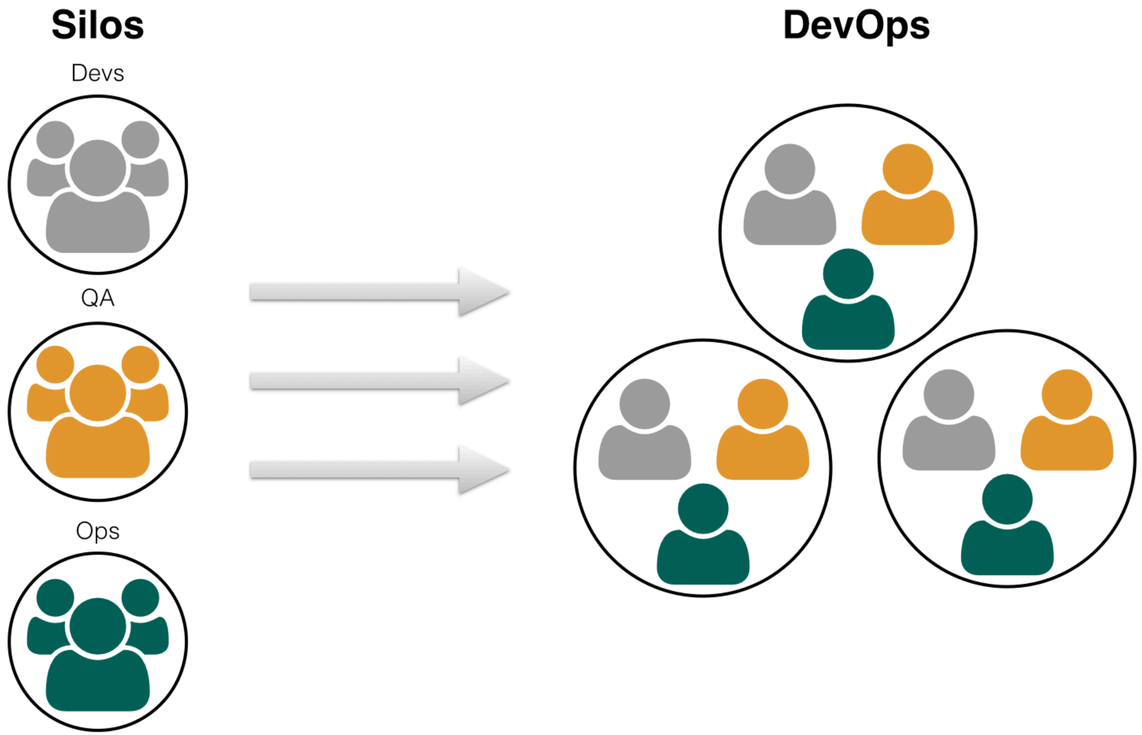


Traditional IT Teams

- Grouped by speciality
- Different vocabulary, tools, management, incentive structures
- Different views on the role of IT
- Heavyweight processes to bridge the gap
- Opposed to moving fast

Silos to DevOps

- Goal: Deliver value rapidly and safely
- Shared vocabulary, tools, and incentive structures
- Bureaucratic processes replaced with trust and accountability
- Common leadership



Cross-functional teams

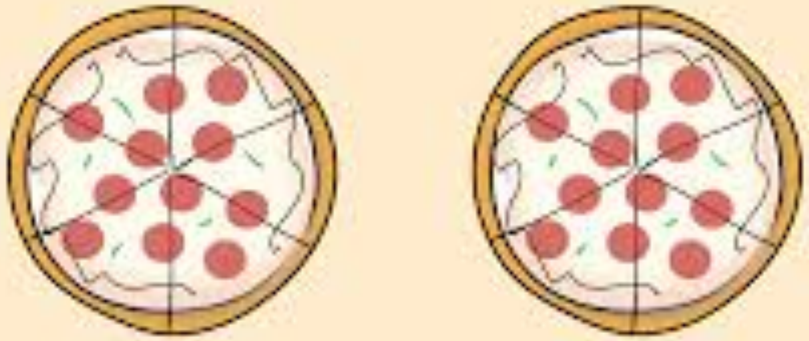
- Break the silos
- Teams consist of talent across multiple disciplines: development, business analysis, QA, ops, etc..
- Goal is for the team to be self-sufficient, independent

Conway’s law

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

Small Teams

Two Pizza Teams == Effective Communication



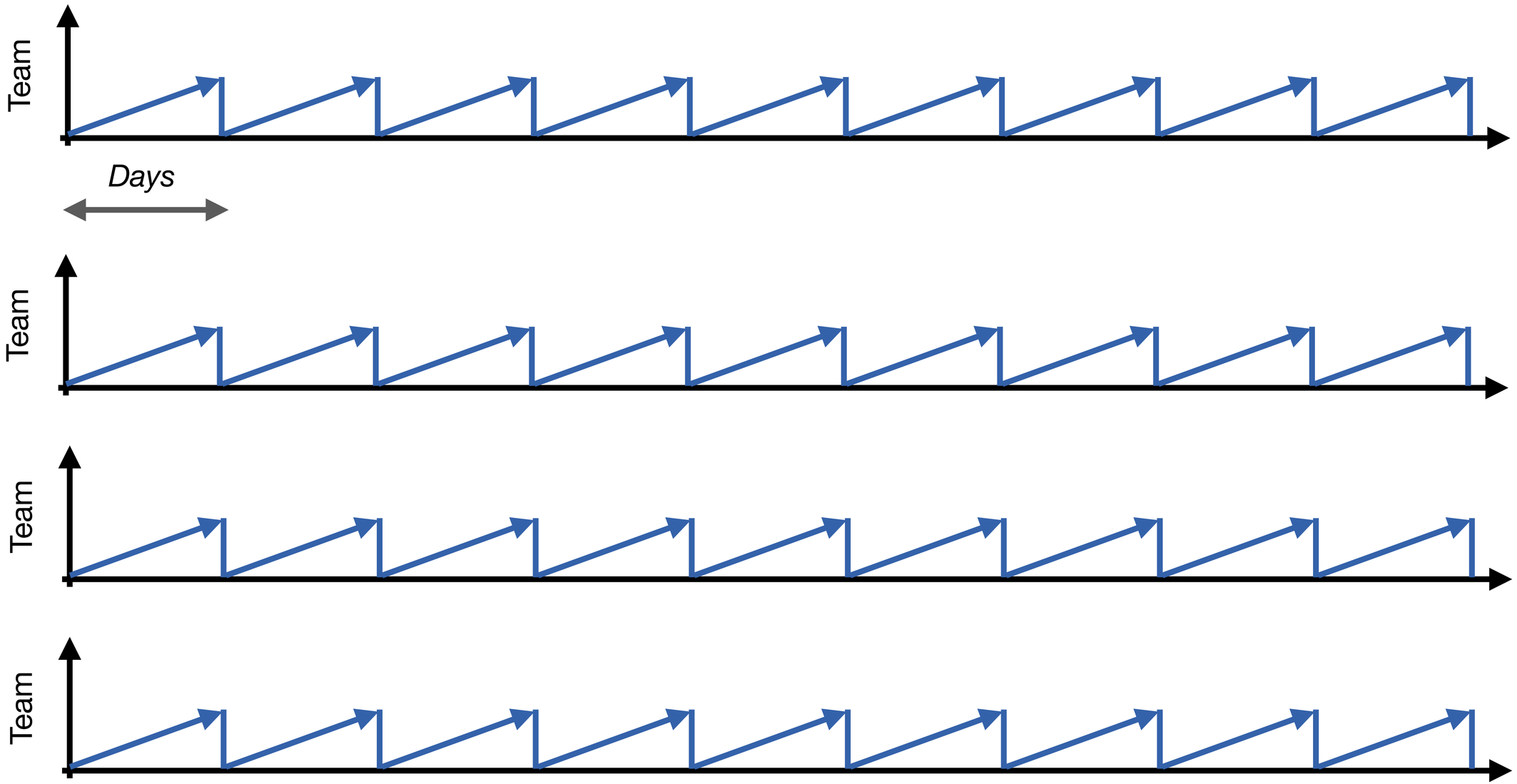
Velocity

- Multiple independent teams working in parallel
- Deploying frequently
- At their own cadence

Pre-requisites to frequent deployments

- Small, simple application
- Low complexity
- Small, independent team
- Low cost of continuous integration within team
- Team must be largely independent of other teams

Parallel work streams combined with frequent deployments

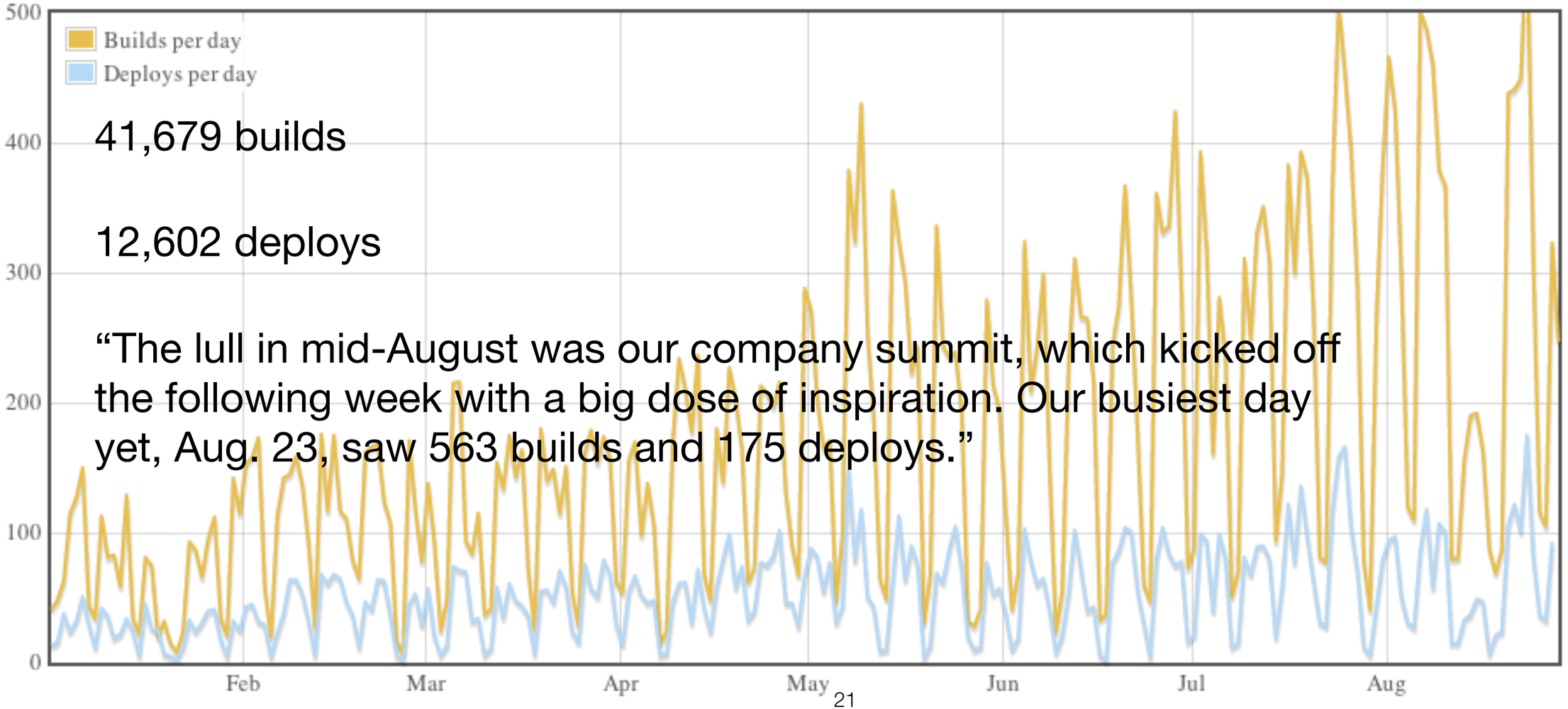


A simple calculation..

- Assume 100 teams, each deploying on average weekly
- Translates to 100 deployments per week, or on average 14 deployments per day
- So velocity can sky rocket, all the while keeping complexity low
- At any *one* point in time, *some* team is deploying to production

GitHub Deployment Stats

https://github.com/blog/1241-deploying-at-github



41,679 builds

12,602 deploys

“The lull in mid-August was our company summit, which kicked off the following week with a big dose of inspiration. Our busiest day yet, Aug. 23, saw 563 builds and 175 deploys.”

Amazon deploys new software to production every 11.6 seconds.

Risk

- Because deployments are frequent and represent changes to a simple application, the risk inherent in a deployment is significantly lower
- Because deployments are performed so often, they tend to get automated, which also lowers the risk of something going wrong
- No checklists, repeatable process
- Deployments to production become non-events

Runtime

- If one MicroService goes down, most of the system is still running — inherently lower risk (contrast with monoliths)
  - Example: *If the Amazon recommendation service is down, are you prevented from checking out?*
- Scale out: replace large machines with container technology, run multiple load-balanced instances of services

Deployment Automation

- Build pipelines and Continuous Delivery
- Techniques: blue-green deployments and canary deployments allow zero-downtime pushes to production
- Reliance on automated testing

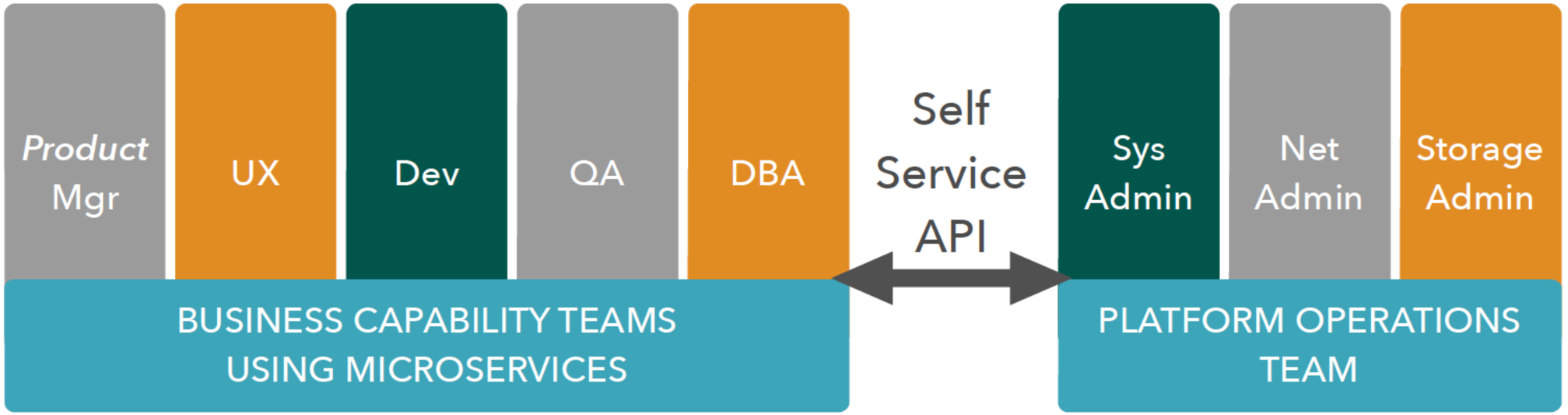
More on automation

- No checklists
- Manual == error-prone
- Want repeatable processes
- Removes bottlenecks, delays
- Frees developers to work on higher-value tasks
- Recreate environments frequently

The Self-Service API

- In most situations, teams don’t have to physically communicate with other teams to get things done: removes bottlenecks!
- Means each team is more independent, not waiting on another team to deliver something
- Lookup published API

Make the platform a self-service API too!



Adapted from: http://www.slideshare.net/adriancockcroft/goto-berlin

As a codebase grows larger,

microServices offer a mechanism to keep complexity low

by splitting the codebase into sets of smaller, largely independent applications

What’s easier?

- Each team works independently
- Developer velocity is high
- Deployments become non-events
- More flexibility vis-a-vis choice of programming language, persistence technology
- Simpler code to maintain

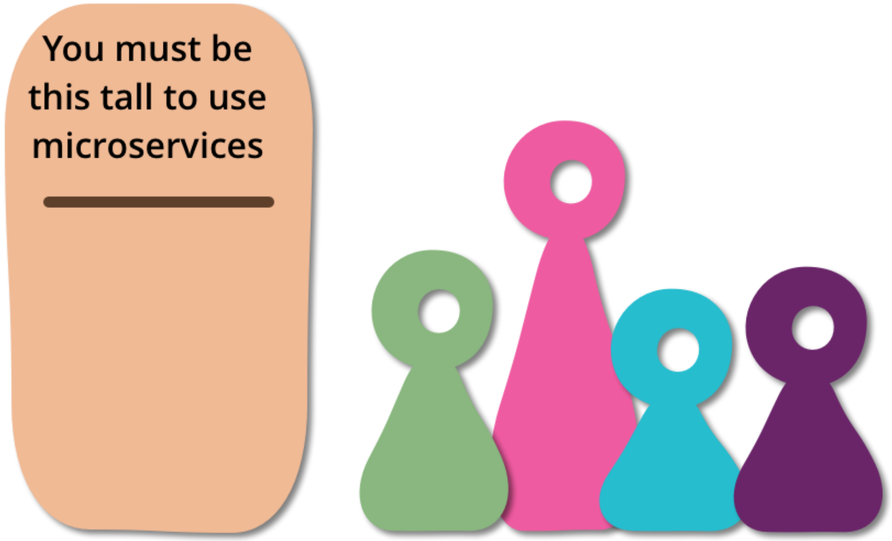
What’s harder?

- Must maintain and deploy more codebases, more applications. Without automation, this can become a real problem
- Service calls are no longer a method call away
- Application becomes a distributed system: distributed computing is hard
- More integration points: what if a contract with another service changes? Need contract testing

MicroService Pre-requisites

Maturity Model, "*you must be this tall to use MicroServices*"

- Rapid provisioning
- Basic monitoring
- Rapid application deployment



https://martinfowler.com/bliki/MicroservicePrerequisites.html

Platforms

A Platform as a Service, or *PaaS,* plays a significant role in tipping the scales in favor of MicroServices.

A PaaS..

- Makes it trivial to deploy and scale applications
- Provides a consistent deployment API
- Leverages container technology, makes efficient utilization of resources
- Makes it easy to create environments on demand
- Automatically restarts services