Cloud Native Introduction, and Patterns

INFORTE - Summer School on Software Evolution: From Monolithic to Cloud-Native

Davide Taibi
davide.taibi@tuni.fi
www.taibi.it

Software Architecture Evolution

1990s and earlier | **2000s** | **2010s**

Coupling

Pre-SOA (monolithic)
Tight coupling

Traditional SOA
Looser coupling

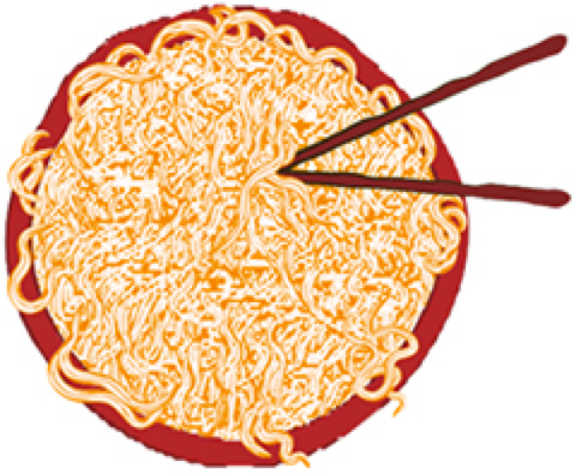
Microservices
Decoupled

D.Taibi, V.Lenarduzzi, C.Pahl, A.Janes. Microservices Architectural Styles: Agile or not Agile? XP 2017

Software Architecture Evolution

1990s and earlier

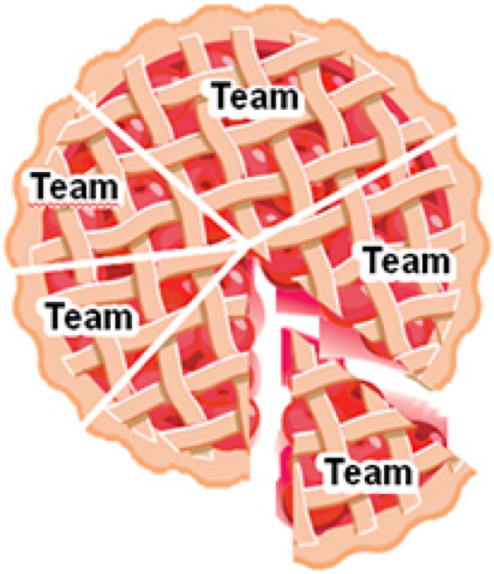
Pre-SOA (monolithic)
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

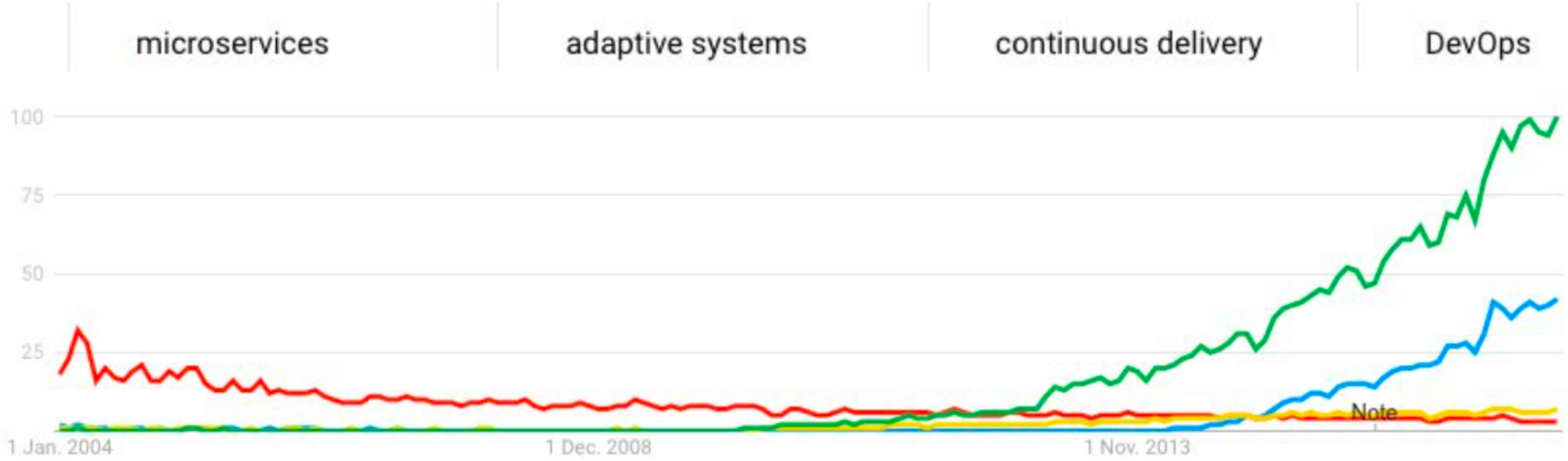
2010s

Microservices
Decoupled

Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.

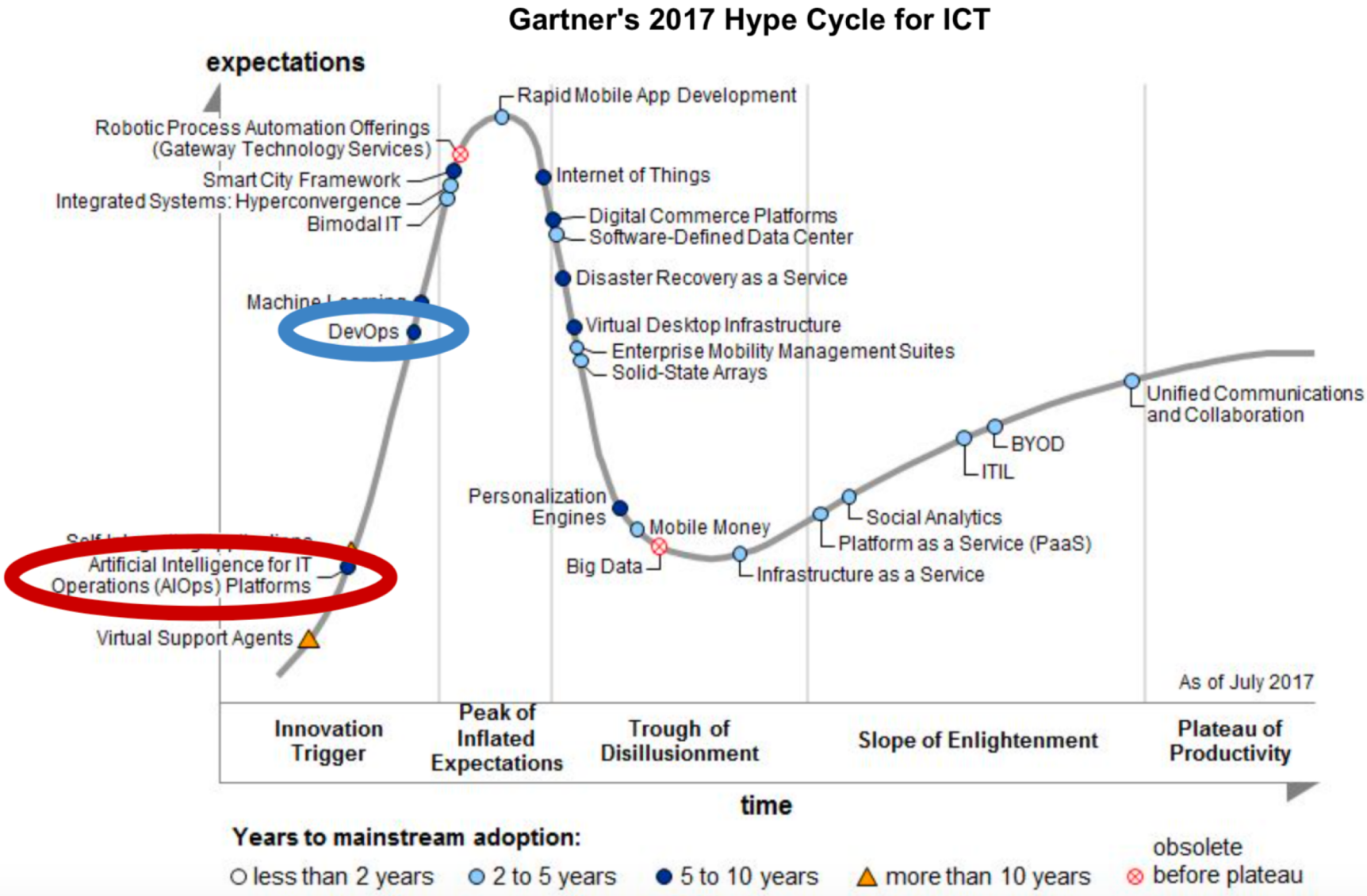
D.Taibi, V.Lenarduzzi, C.Pahl, A.Janes. Microservices Architectural Styles: Agile or not Agile? XP 2017

Why?



source: https://trends.google.co.in/trends/

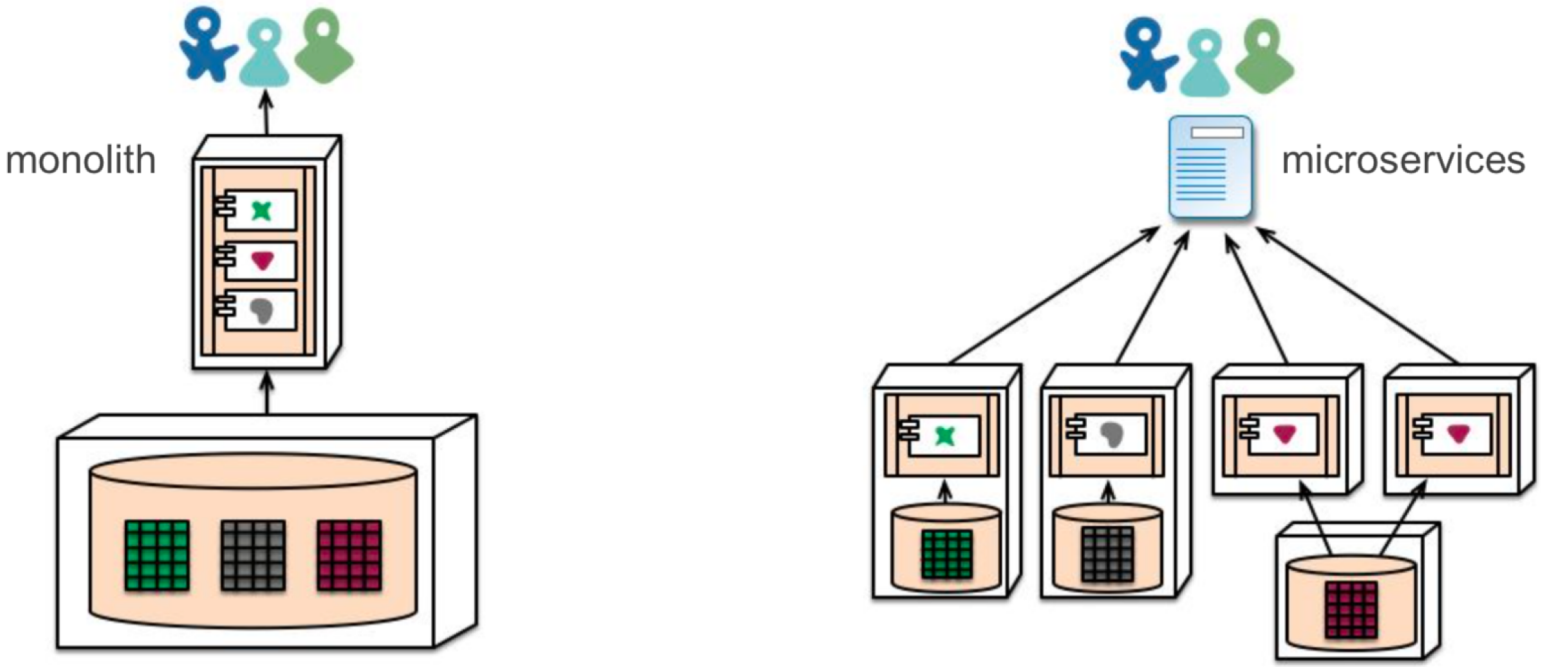
Why?


Gartner's 2017 Hype Cycle for ICT
expectations
Rapid Mobile App Development
Robotic Process Automation Offerings (Gateway Technology Services)
Smart City Framework
Integrated Systems: Hyperconvergence
Bimodal IT
Internet of Things
Digital Commerce Platforms
Software-Defined Data Center
Disaster Recovery as a Service
Machine Learning
DevOps
Virtual Desktop Infrastructure
Enterprise Mobility Management Suites
Solid-State Arrays
Unified Communications and Collaboration
BYOD
ITIL
Personalization Engines
Mobile Money
Social Analytics
Platform as a Service (PaaS)
Big Data
Infrastructure as a Service
Artificial Intelligence for IT Operations (AIOps) Platforms
Virtual Support Agents
As of July 2017
Innovation Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
time
Years to mainstream adoption:
less than 2 years
2 to 5 years
5 to 10 years
more than 10 years
obsolete before plateau

"A **suite of small services**, each **running in its own process** and **communicating with lightweight mechanisms**, often an HTTP resource API."

"These services are **built around business capabilities** and **independently deployable** by **fully automated** deployment machinery."

"There is a **bare minimum of centralized management** of these services, which may be written in **different programming languages** and use **different data storage technologies**."
J. Lewis & M. Fowler, ThoughtWorks



source: https://martinfowler.com/articles/microservices.html

"Small **autonomous** services that **work together**, modelled around a **business domain**."

S. Newman, ThoughtWorks, author of "Building Microservices"

"**Loosely coupled** service-oriented architecture with **bounded contexts**."

"Monolithic apps have **invisible internal complexity**. Microservices expose that [complexity] as **explicit micro service dependencies**."

Adrian Cockcroft, AWS (formerly at Netflix)

"We need to move to **managed complexity**. Microservices are about **negotiated interfaces**, **strict boundaries**, **shared nothing**!"

J. Higginbotham, LaunchAny

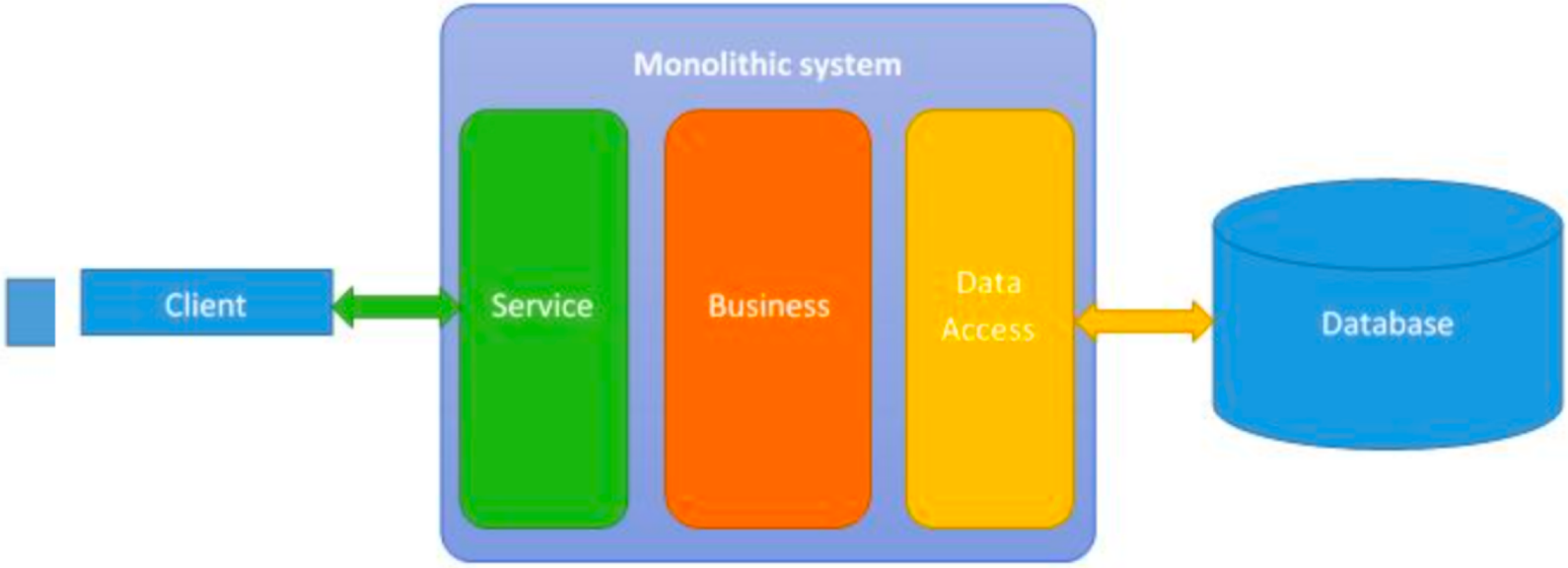
Amazon's now famous migration from the Obidos monolithic application to a service-oriented architecture with encapsulated databases and small, "two-pizza" teams

Amazon's design principles:

- Design for flexibility
- Design for on demand
- Design for automation
- Design for failure
- Be elastic
- Design for utility pricing
- Break transparency
- Decompose to its simplest form
- Design with security in mind
- Don't do It alone
- Focus on what doesn't change
- Let your customers benefit
- Continuously innovate

"For us service orientation means encapsulating the data with the business logic that operates on the data, with the only access through a published service interface. No direct database access is allowed from outside the service, and there's no data sharing among the services."

-- Werner Vogel, Amazon's CTO, 2006



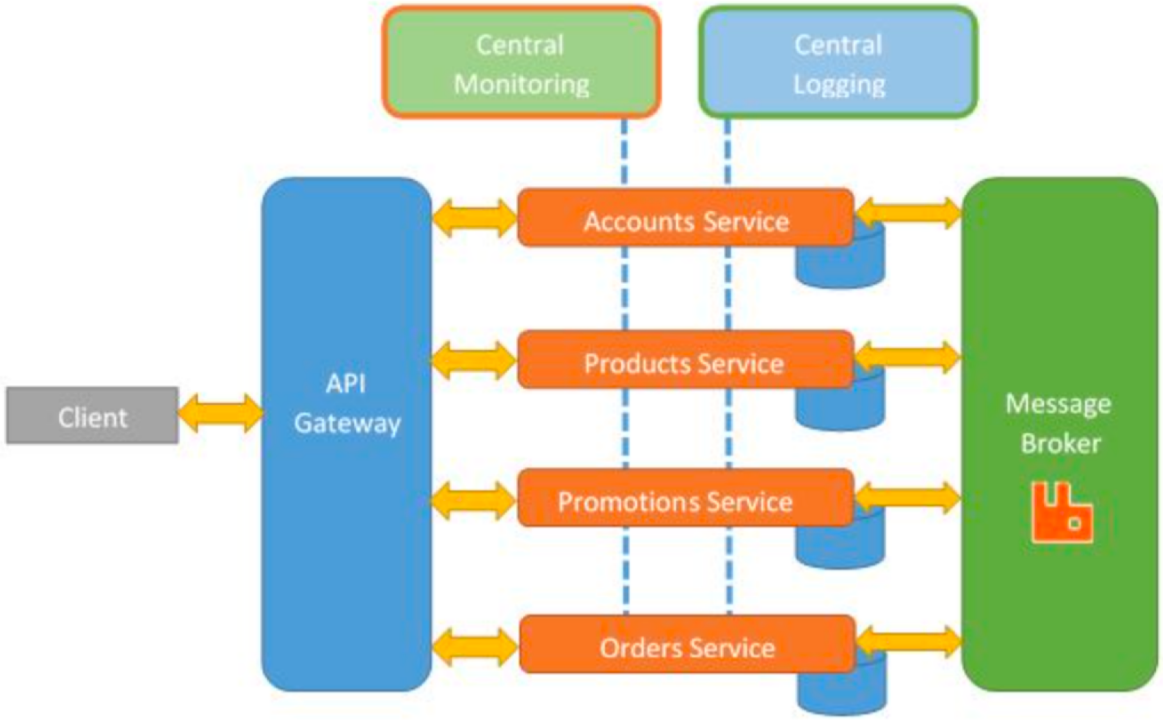
source: http://www.acarlstein.com/

"There's no reason why you can't make a single monolith with well defined module boundaries. At least there's no reason *in theory*. In practice, it seems too easy for module boundaries to be breached and monoliths to get tangled as well as large."

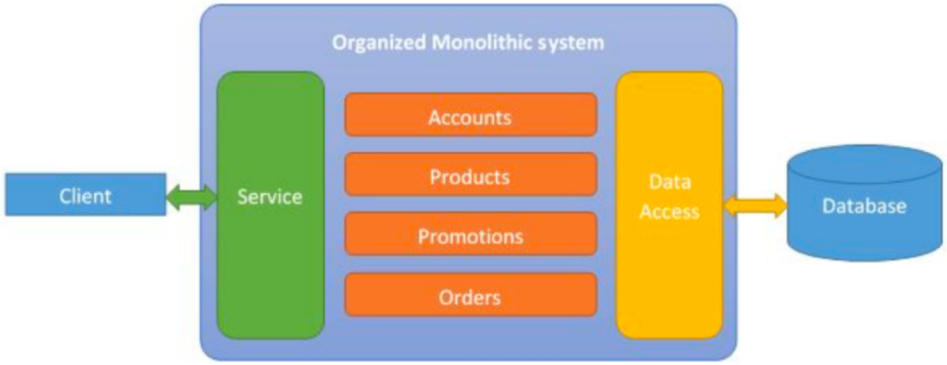
-- Martin Fowler

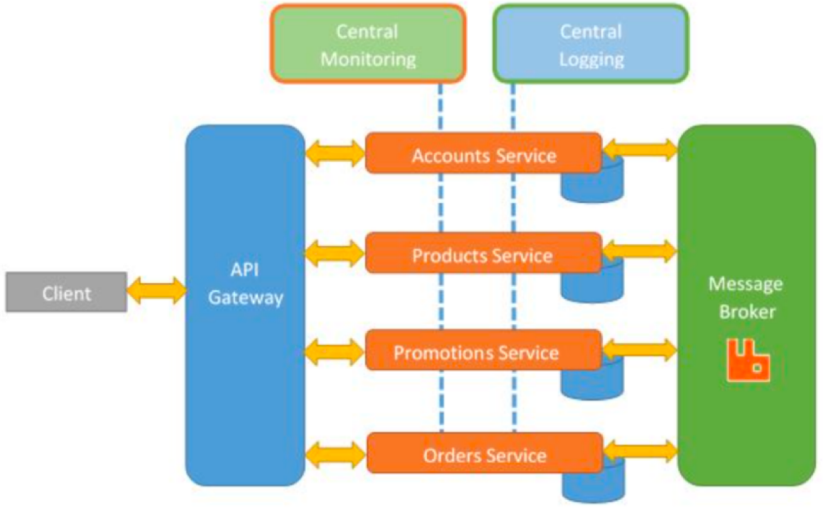
The microservices style is an approach to design systems whose parts are easy to change and replace *at runtime*

It pushes information hiding to new heights by enforcing strict module boundaries and by promoting information isolation



source: http://www.acarlstein.com/



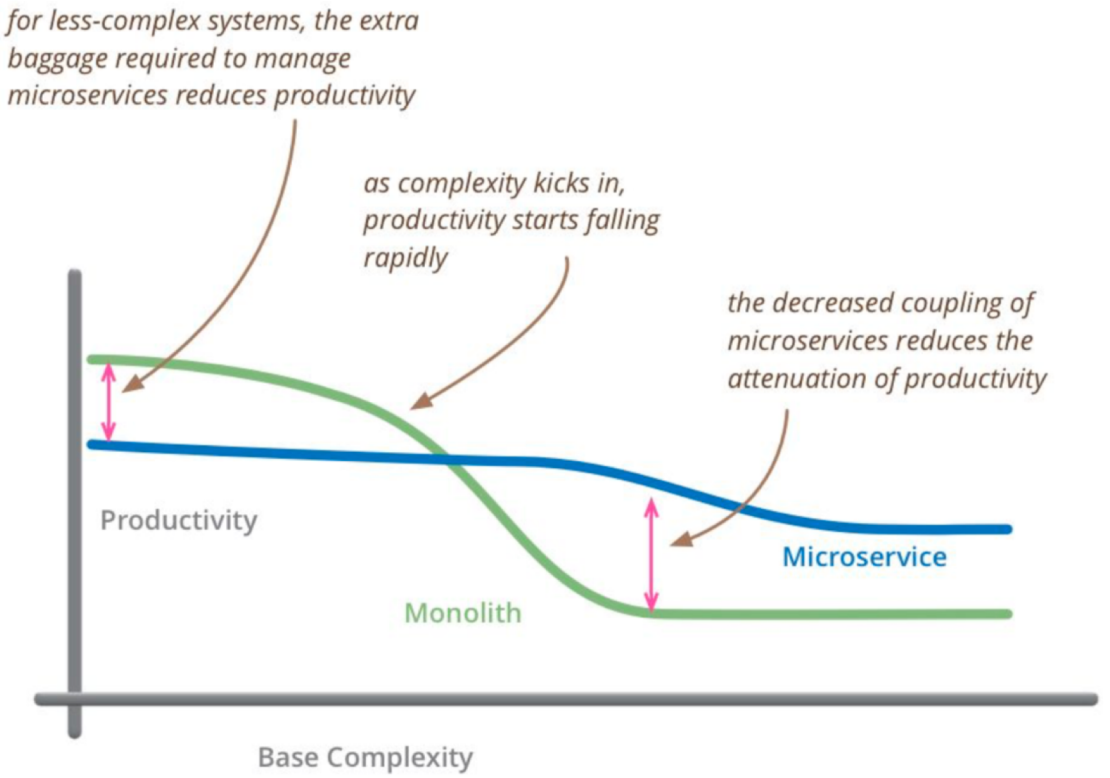


- One large code base
- Long release cycles
- Usually stuck with one dominant technology
- More susceptible to system-wide failures
- Can only monitor system-level properties (difficult to identify which modules are causing problems)
- Scaling requires updating or replicating the whole system stack

- Multiple "small" code bases
- Short release cycles
- Freedom to explore different (i.e., competing) technologies
- Failure in one module may not necessarily affect the other modules
- Modules can be individually monitored
- Modules can be independently scaled

The microservices style introduces its own set of (distributed systems related) complexities:

- automated deployment
- continuously monitoring
- dealing with failure
- eventual consistency
- security

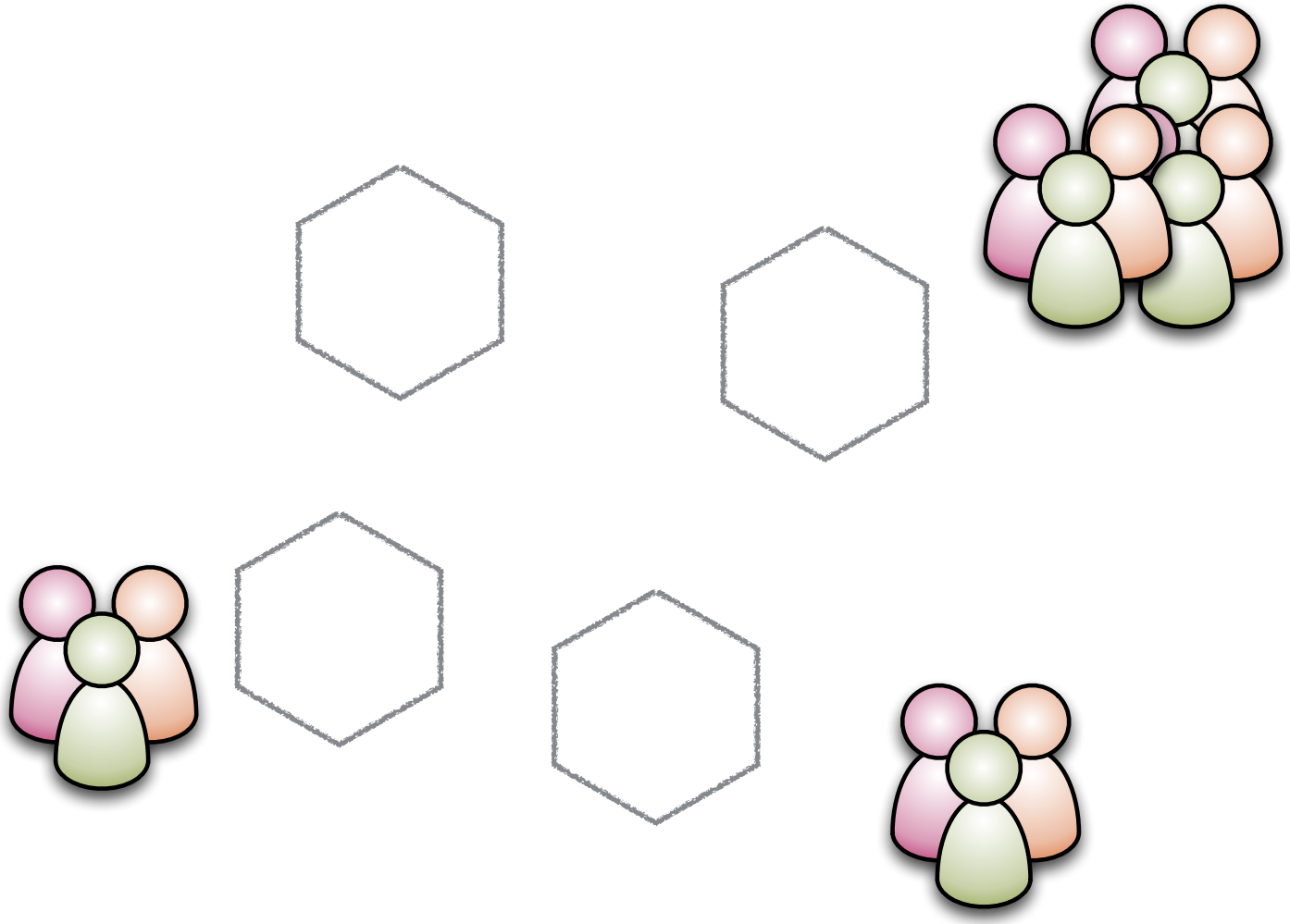


source: https://martinfowler.com/bliki/MicroservicePremium.html

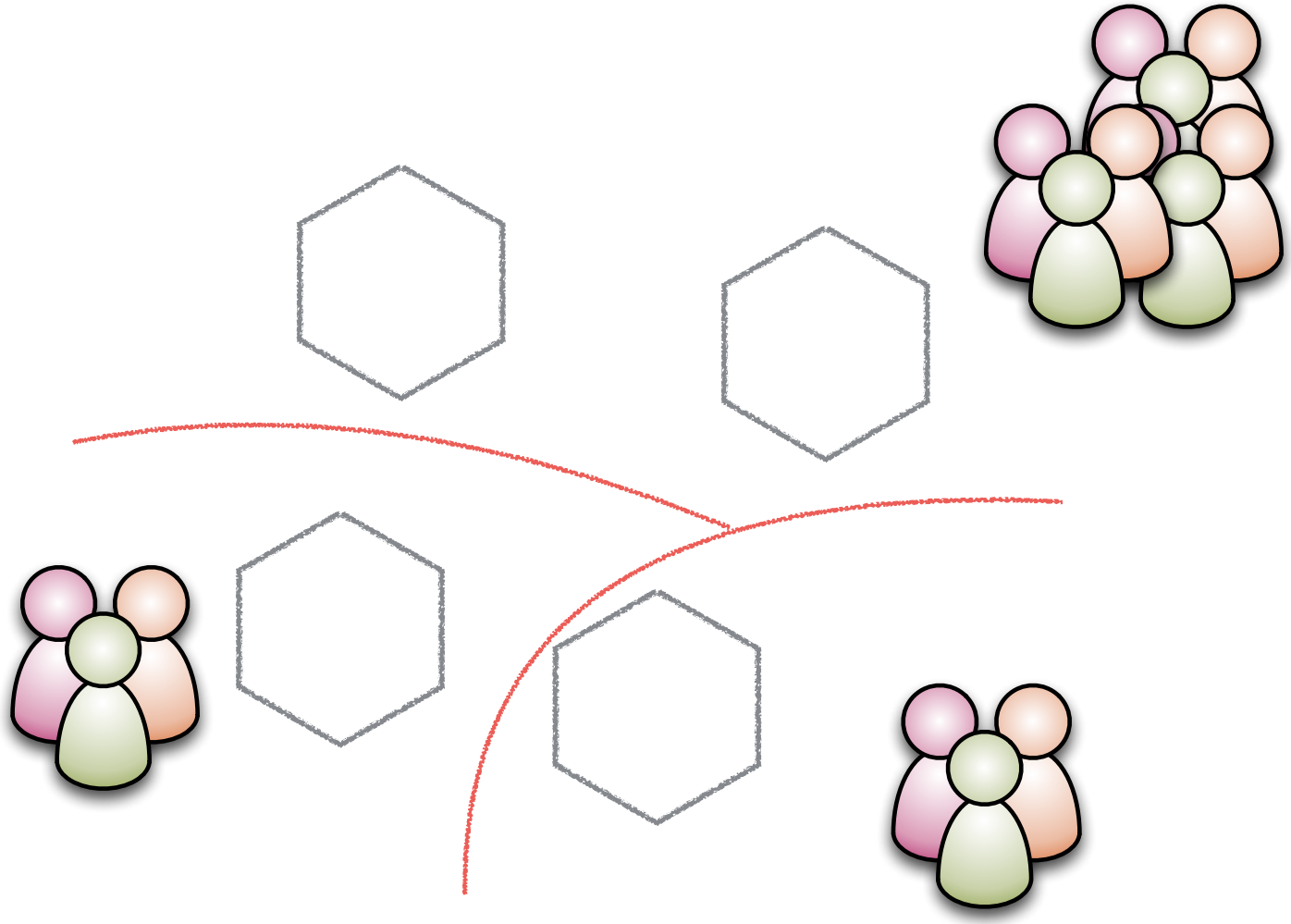
"The majority of software systems should be built as a single monolithic application. Do pay attention to good modularity within that monolith. Don't even consider microservices unless you have a system that's too complex to manage as a monolith."

-- Martin Fowler

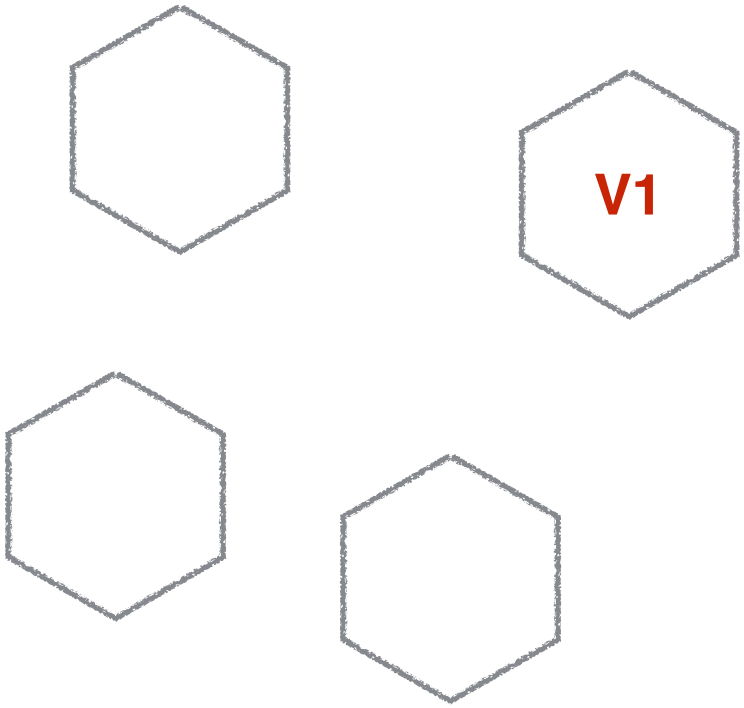
We can organise services along organisational boundaries



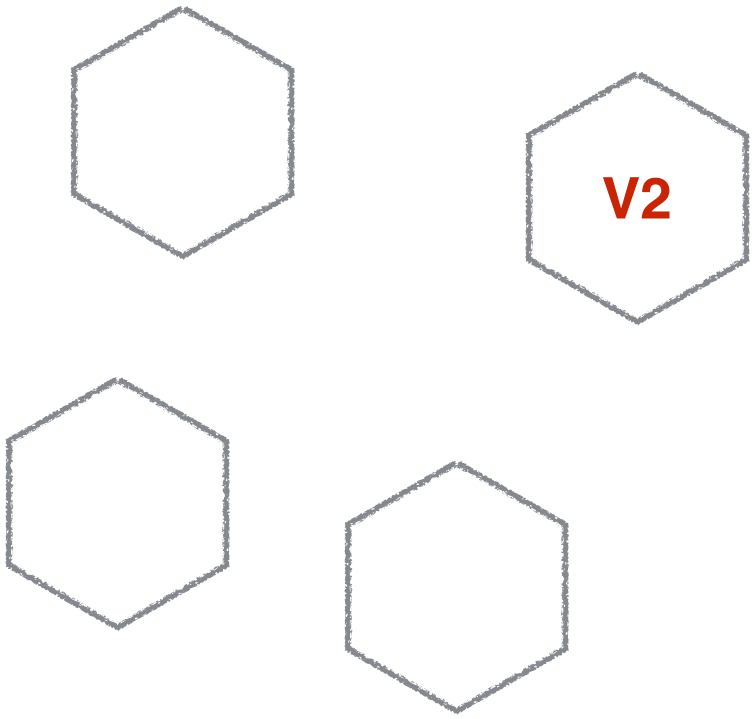
We can organise services along organisational boundaries



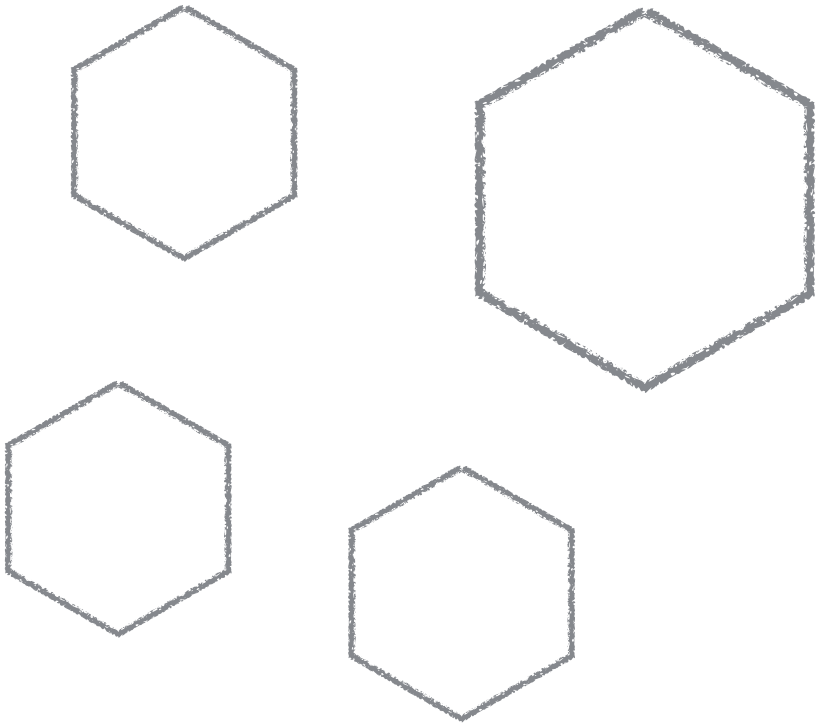
By sub-divinding our systems, we can speed the release of new features



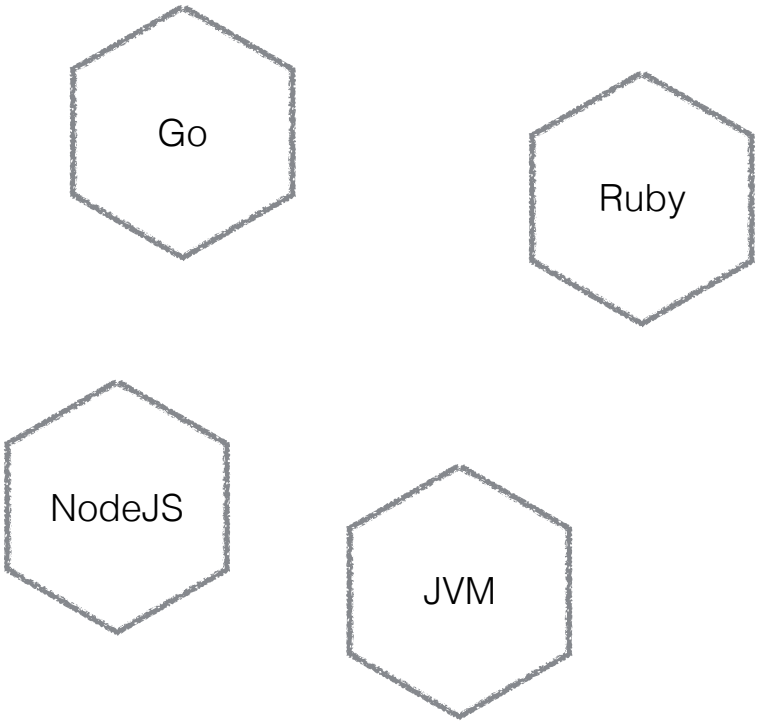
By sub-divinding our systems, we can speed the release of new features



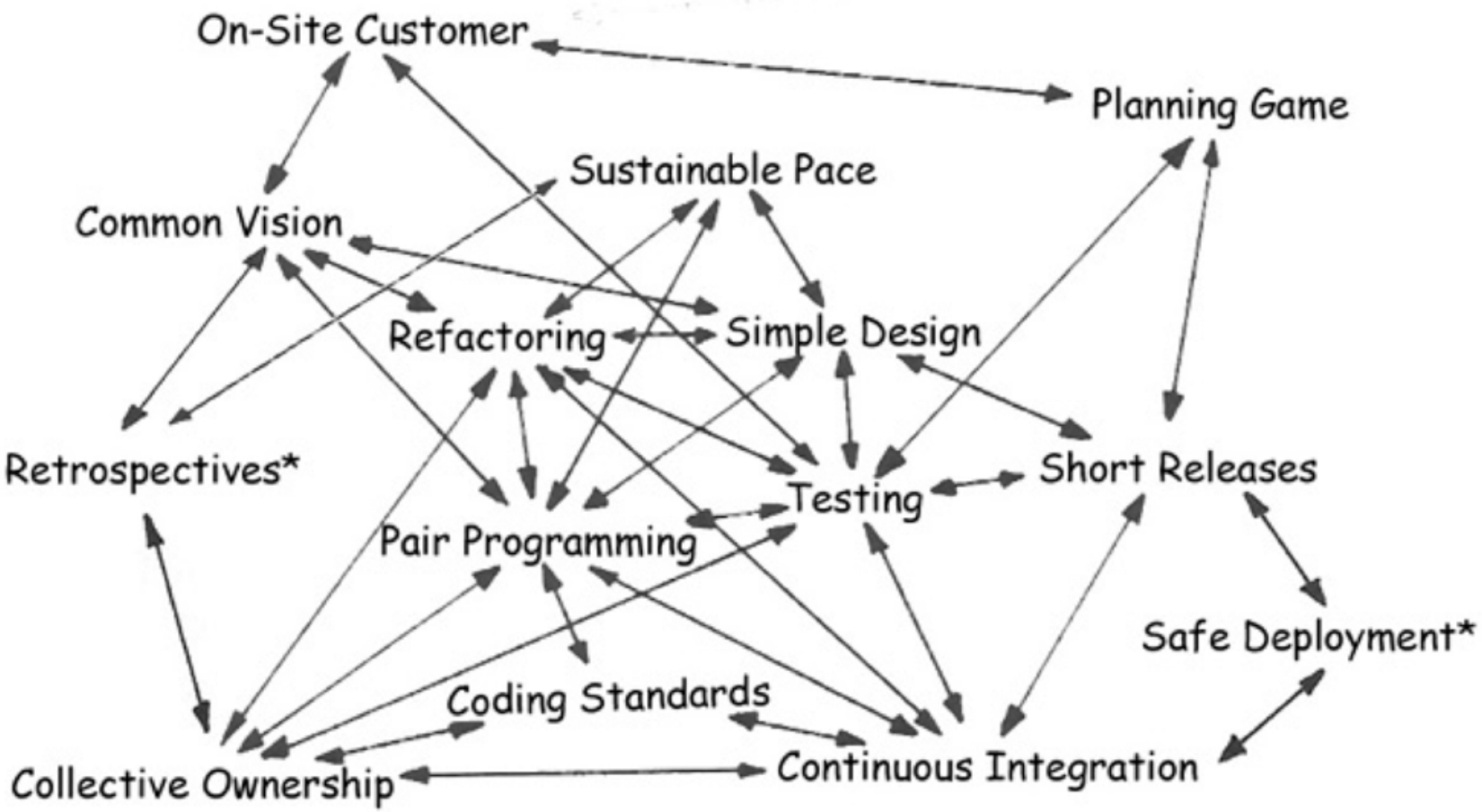
It allows us different options in terms of scaling



and we can use different tools and tech



Why now?


On-Site Customer
Planning Game
Sustainable Pace
Common Vision
Refactoring
Simple Design
Retrospectives*
Testing
Short Releases
Pair Programming
Safe Deployment*
Coding Standards
Collective Ownership
Continuous Integration


standing on the
shoulders of giants


The Pragmatic Programmers
Release It!
Design and Deploy
Production-Ready Software
Michael T. N
The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler
Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler
REST in Practice

Summary

We understand more about building reliable distributed systems

cloud compute and programmable infrastructure has matured

organisations need to adapt and change quickly to survive

we spend **too much money** on building monoliths