Microservices Overview

What are Microservices?

Microservices Characteristics

Deconstructing a Monolith

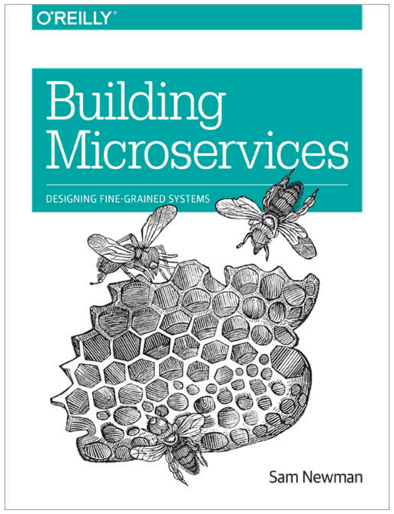
What are Microservices?

Fowler (2014):

Microservices are "an approach to developing a single application as 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."

Norman (2015):

"Microservices are small, autonomous services that work together"

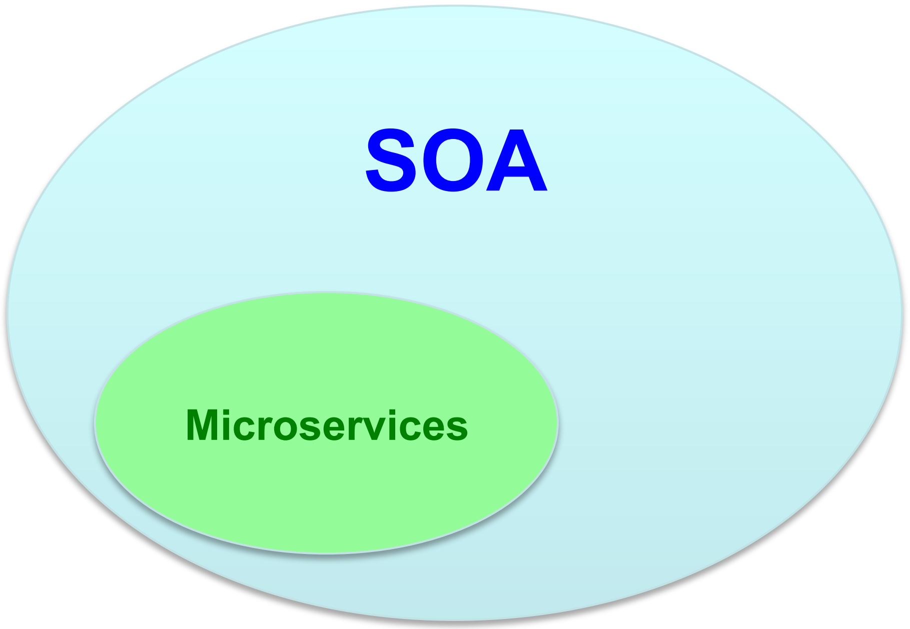


Microservices and SOA

In a 2014 talk at GOTO, Fowler says Microservices are a distinct subset of SOA.

Norman says:

"SOA at its heart is a very sensible idea. However, despite many efforts, there is a lack of good consensus on how to do SOA well. In my opinion, much of the industry has failed to look holistically enough at the problem and present a compelling alternative to the narrative set out by various vendors in this space."



Characteristics of a Microservice Architecture

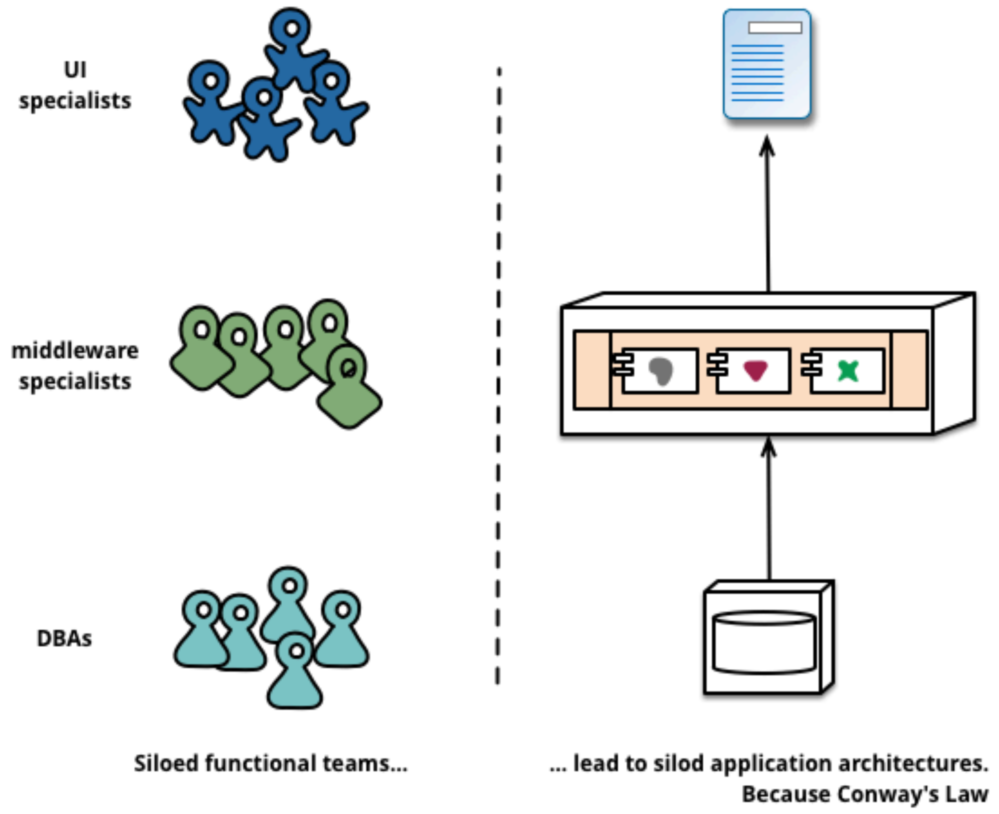
1. Componentization via Services
2. *Organized around business capabilities*
3. Products not Projects
4. Smart endpoints and dumb pipes
5. *Decentralized Governance*
6. Decentralized Data Management
7. Infrastructure Automation
8. Deign for Failure
9. Evolutionary Design
10. There is no #10

We will discuss #2 and #5 as they are the most pertinent to the ideas of splitting up a monolith

Organized around business capabilities

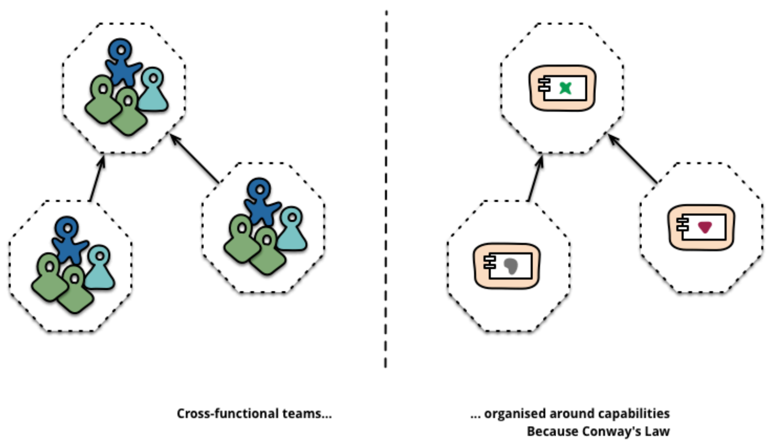
Conway's Law (1967)

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


UI specialists
middleware specialists
DBAs
Siloed functional teams...
... lead to silod application architectures. Because Conway's Law

The idea is to use cross-functional teams.

- Instead of teams per tier, create a team with all of the specialities (UX, DB, PM, etc.)


Cross-functional teams...
... organised around capabilities Because Conway's Law

Decentralized Governance (or, no ESB!)

Decentralized Governance is mainly about teams (like the previous principle #2)

1. Teams are cross-functional and *own* all aspects of a service
   - Design, Development, Maintenance, Test, Deployment, Support…
   - No more "throw it over the wall"!
2. Teams are fully empowered to decide the technology stack they want to use to implement the microservice

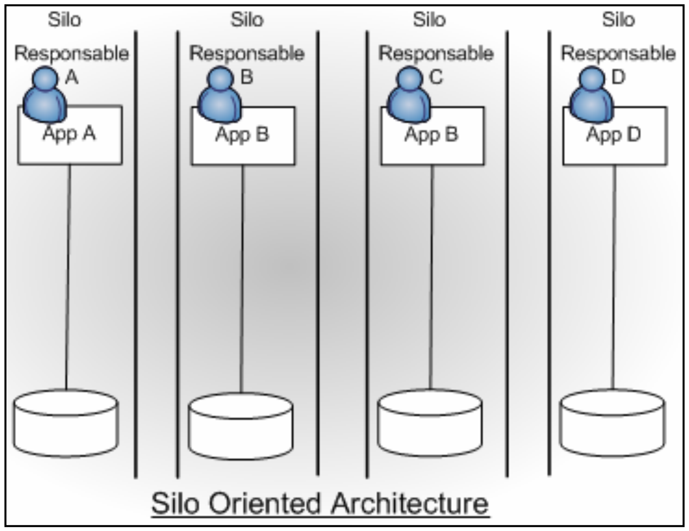
Aside: #1 is kinda new, but haven't we seen #2 before?

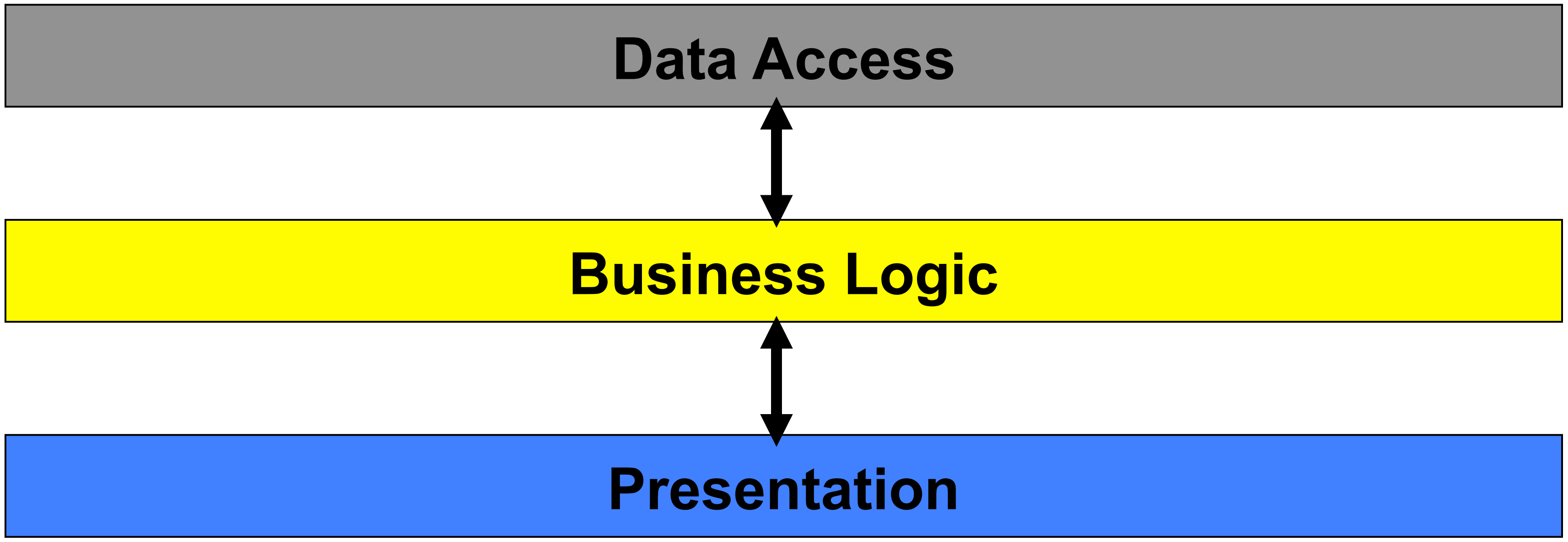
- Both Fowler and Norman acknowledge we have seen technologies and approaches for interoperability before, and they hypothesize a number of factors and changes
  - In legacy days, the use of interoperable technology stacks was driven by legacy systems, or your need to have Java talk to .NET
  - Distributed computing vendors did not like decentralization since there was nothing to sell like an "ORB" or an "ESB" or a homogeneous stack

Splitting the monolith

"In the old days" a monolith was a collection of *stovepipes*, where each business unit had their own systems

Then we came up with partitions based on technical responsibilities, or *concerns*





Splitting the monolith

Separation of Concerns (SoC) motivated our MVC pattern.

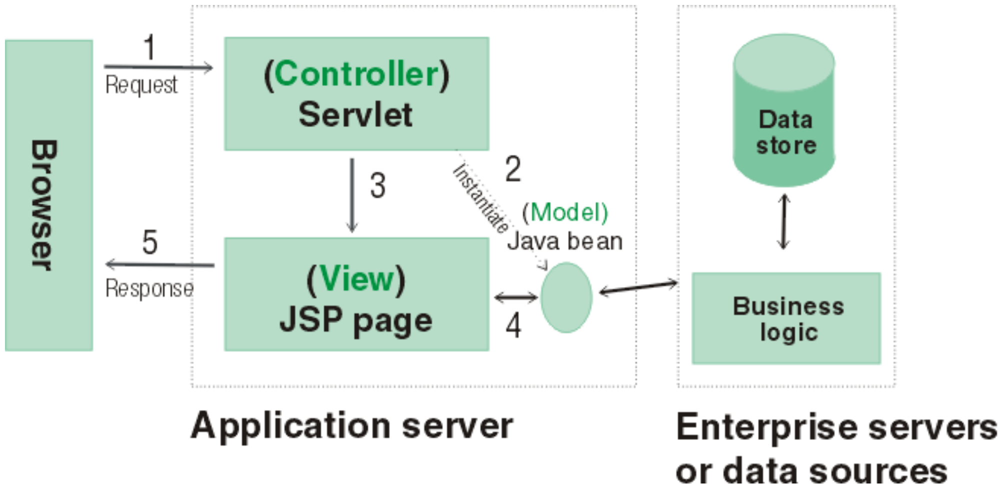
Susan Fowler (Production Microservices): Code for Client/Server/Database Systems combined in 3 ways:

1. Front-end and Backend in the same codebase (repo) and run as one process (server-side web applications)
2. Separate front-end and backend into separate executables (the rise of client-side web applications)
3. Combine all 3, especially if no external DB

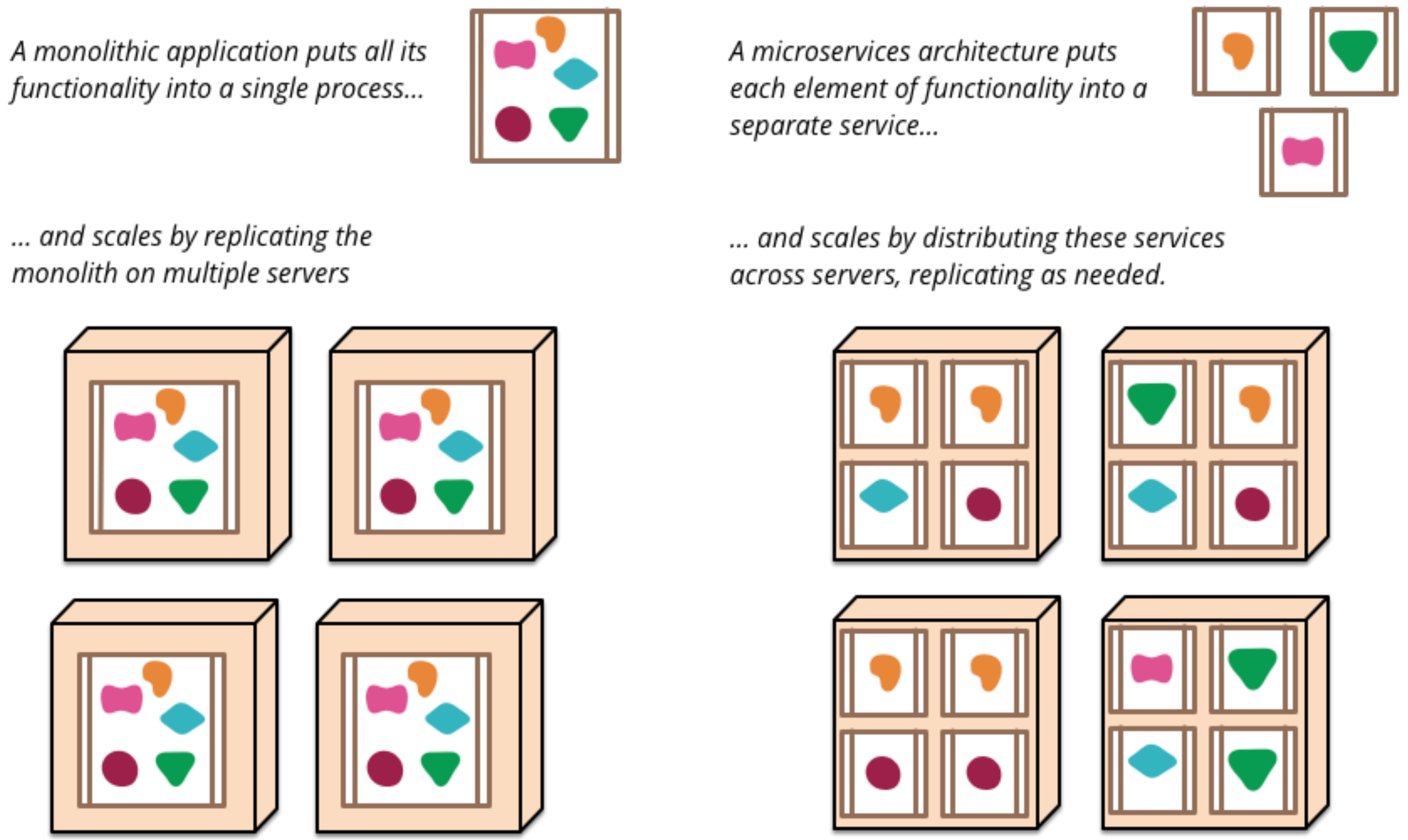
We run all of our requests through the same set of layers!

We have discussed how the MVC pattern is a guiding principle for organizing our code, runtimes, and even our people

(Is this bad?)



Splitting the monolith



A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.

Summary

- Fowler in a separate 2014 article says his gut feel is that monolith is better because distribution adds complexity (we'll get to this)
- He says in a different article (paraphrase) "all successful microservice implementations I have seen started as monoliths, and all failed one started with microservices"
  - He means it is hard to design directly for microservices (some debate on this in the community)
  - He also says that you need to be sure that you need microservices, and this is pretty generally agreed upon.
- Sam Norman focuses on the concept of a *seam* from Feathers which is a way of reverse engineering legacy code by identifying *bounded contexts* (stay tuned)
  - He also talks a lot about what to do with that big database, which we will also get to…