ThoughtWorks®

MYTHS OF MICROSERVICES

Nima Montazeri *@nimamon*

TONIGHT...

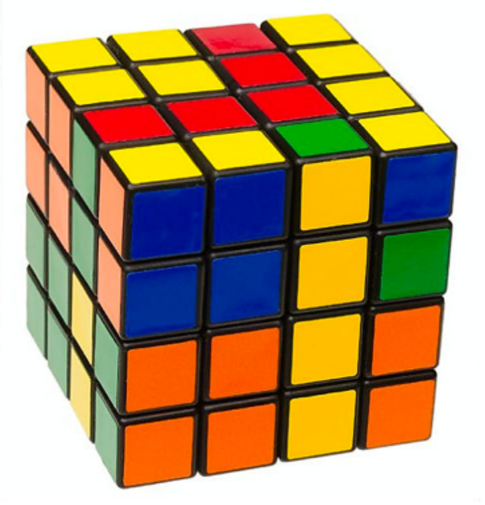
- Definitions & Characteristics
- Microservices Premium
- **Where** to use Microservices
- **When** to use Microservices
- Q&A

@nimamon

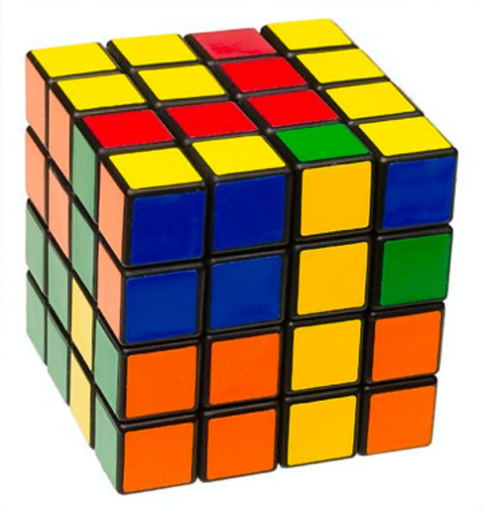
RESOURCES

http://nima.tech/micro-services/

@nimamon



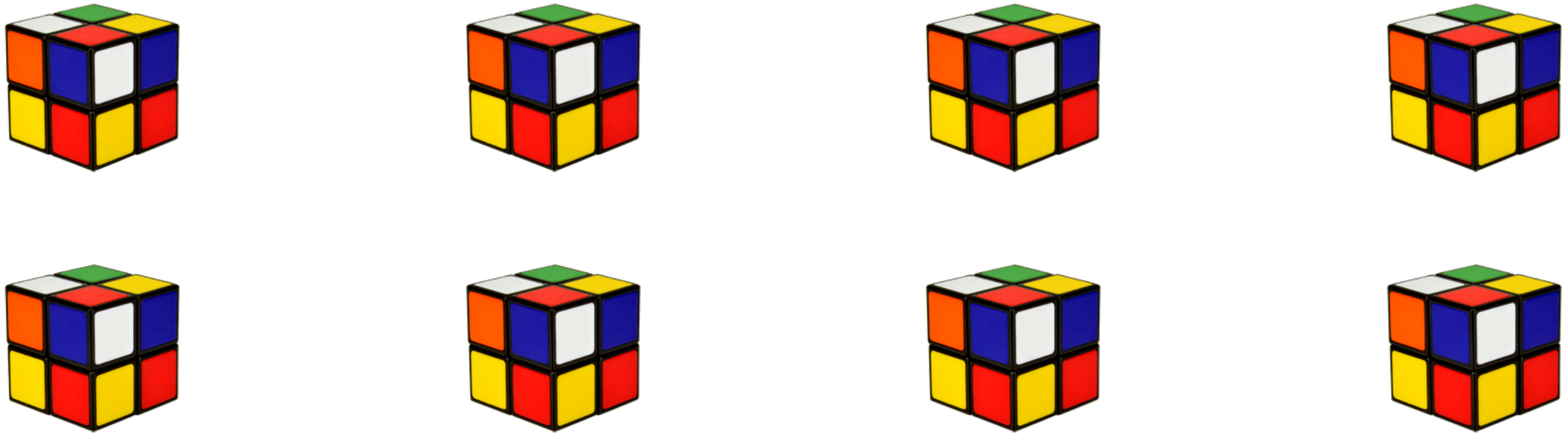
QUESTION OF COMPLEXITY



4X4 Rubik's Cube

7.40×10^{45}

7401196841564901869874093974498574336000000000



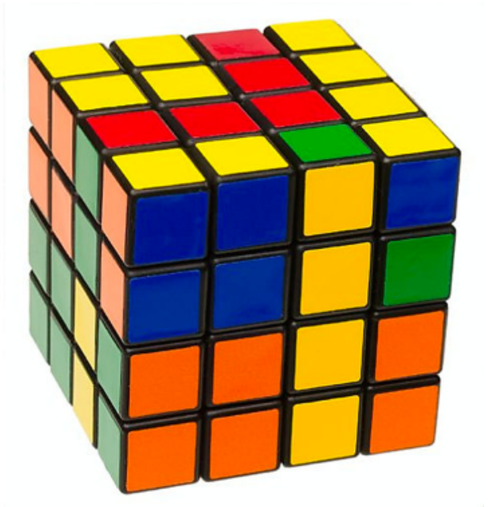
QUESTION OF COMPLEXITY

2X2 Rubik's Cube

only(!) $3.7 \times 10^{6} \times 8$

29,600,000

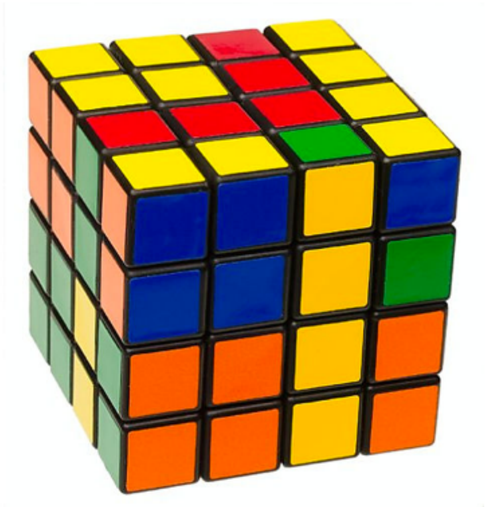
QUESTION OF COMPLEXITY



7401196841564901869874093974498574336000000000

x 8 29,600,000

QUESTION OF COMPLEXITY



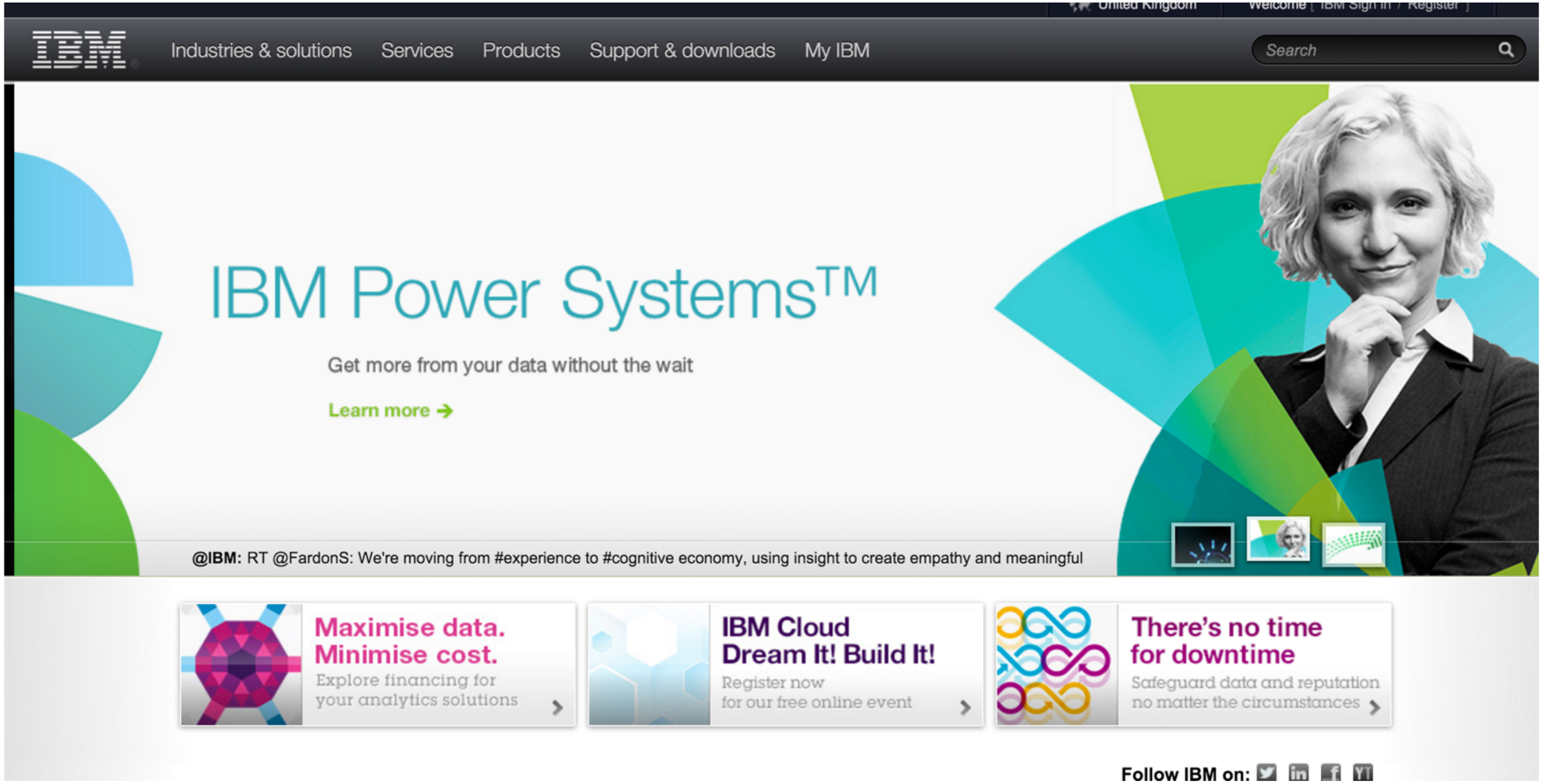
4×10^{35} times simpler

x 8

Microservices

Chart by Google Trends

http://microservices.org



Microservices vs SOA

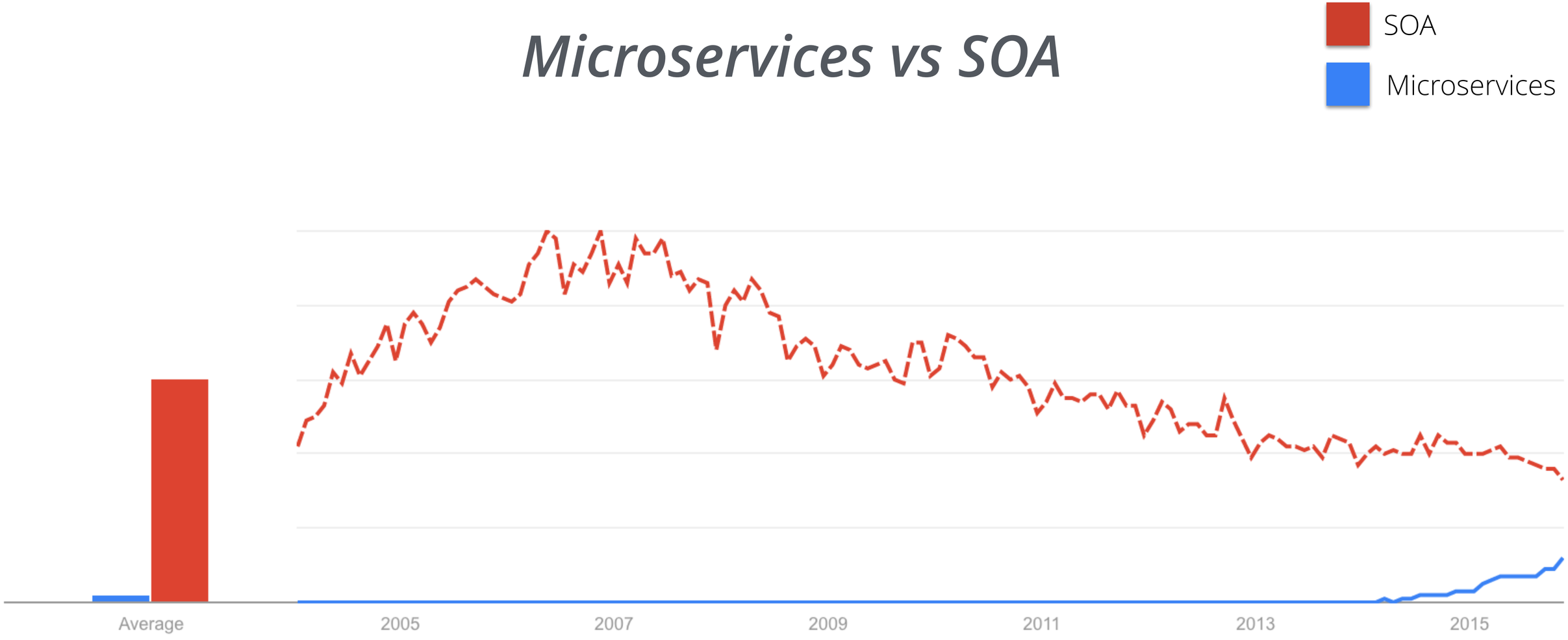


Chart by Google Trends

ARE MICROSERVICES JUST SOA?

useful subset of SOA practices

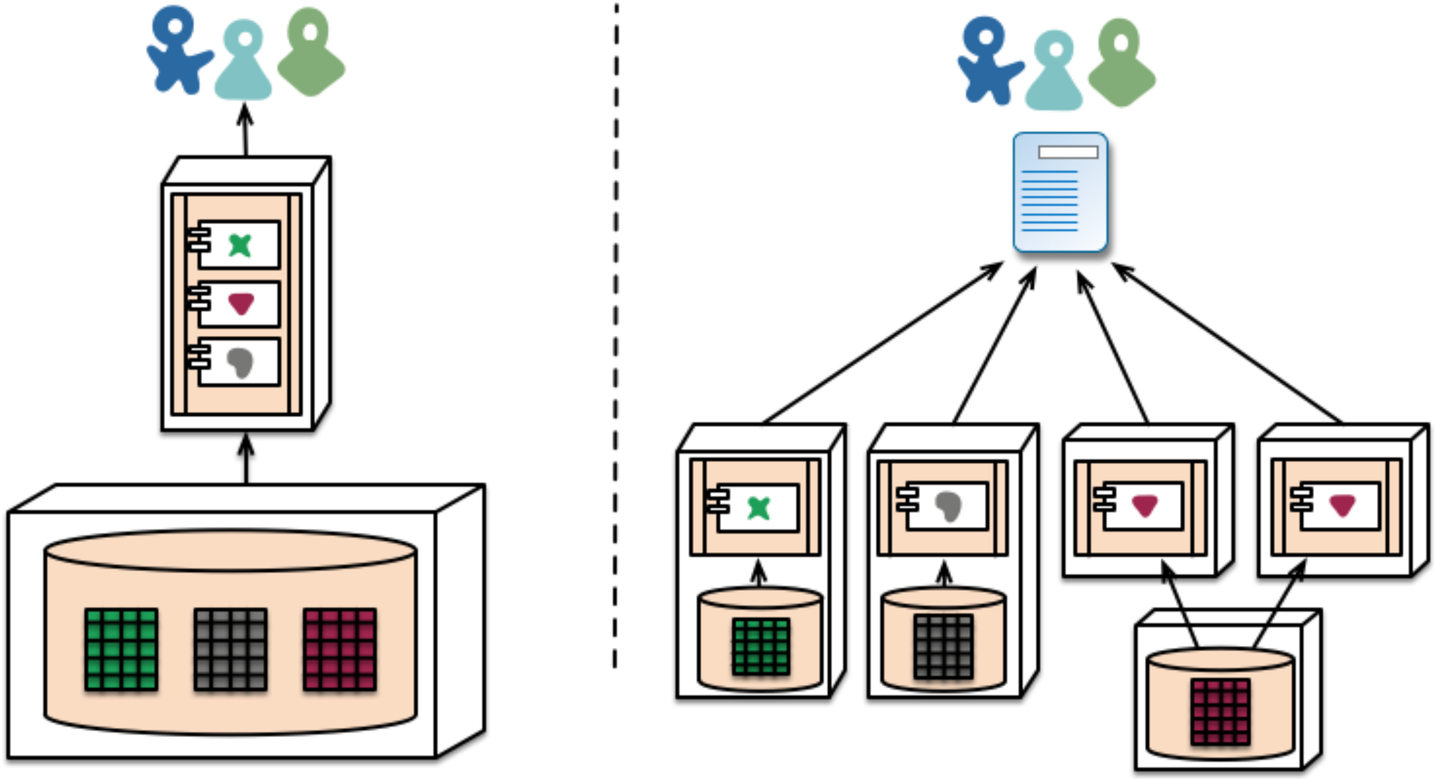
Characteristics

~~DEFINITION~~ OF MICROSERVICES

Componentization via Services

- Idea has been around for decades
- **Replace** and/or **upgrade** independently
- Services as components

Independent Data Storage



monolith - single database

microservices - application databases

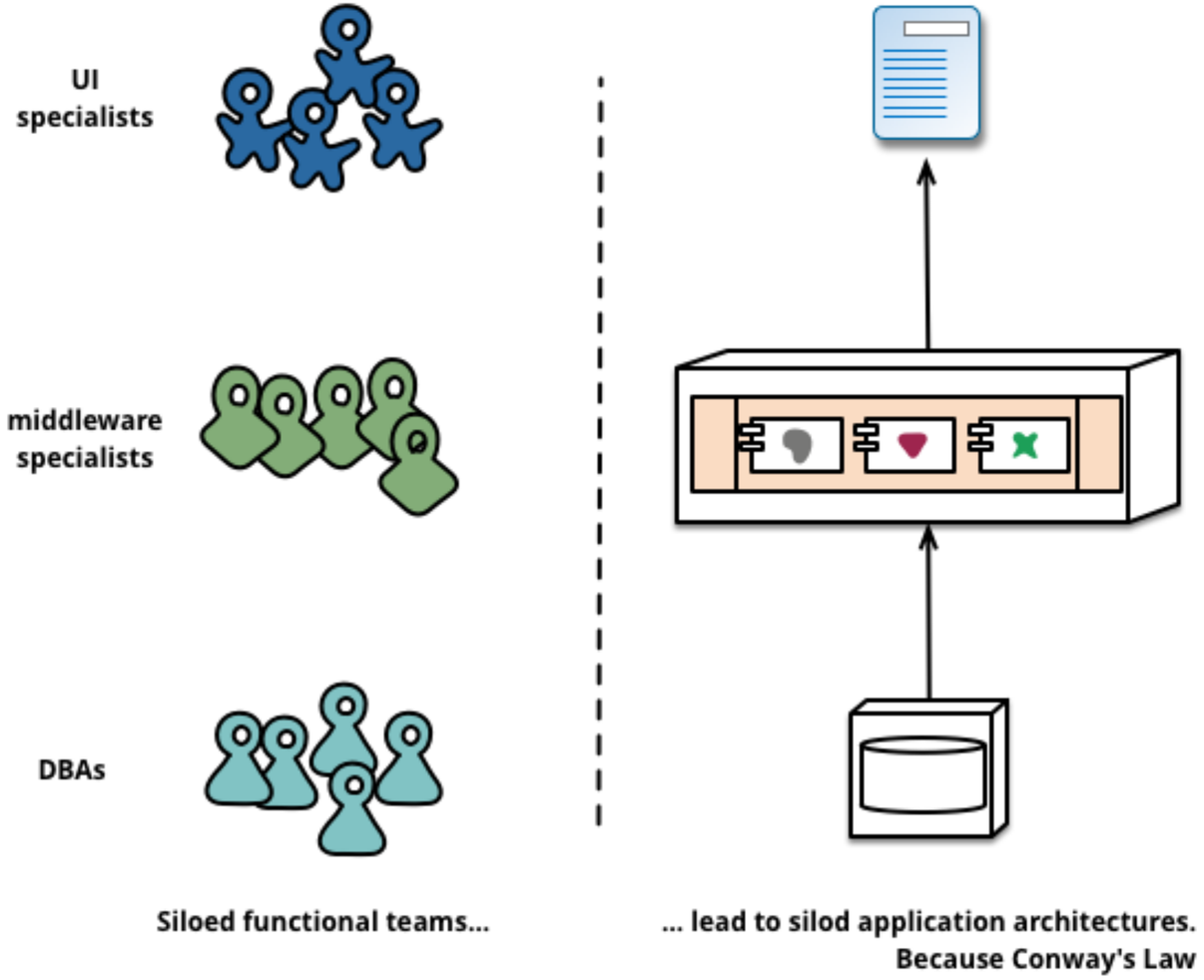
Infrastructure Automation

- Provision new machines in minutes not weeks
- Cloud Infrastructure
- Automate build, test and deployment

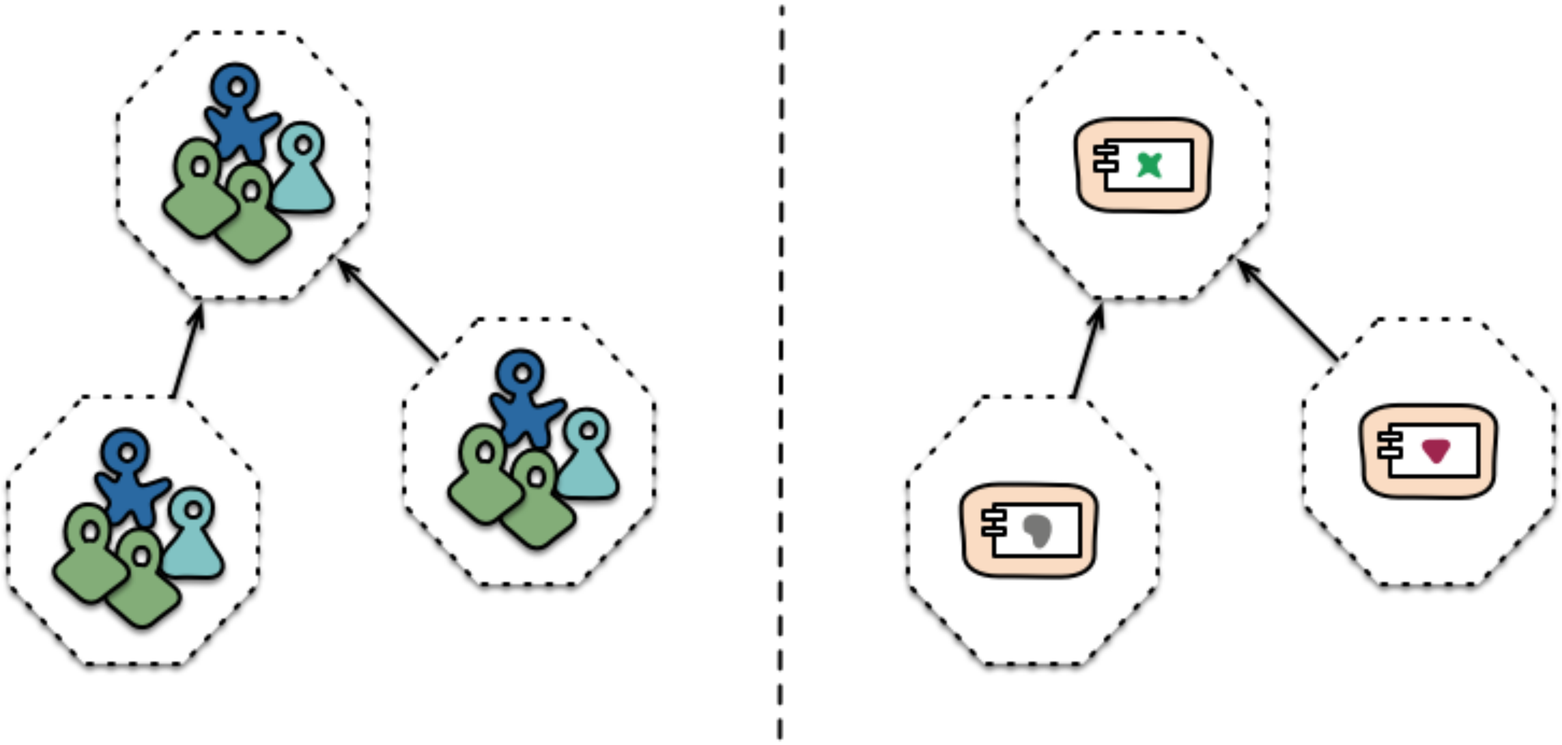
Design for Failure

- Services are going to fail
- Resiliency over availability
- Monitoring and troubleshooting

Organised around Business Capabilities



Organised around Business Capabilities



Cross-functional teams...

... organised around capabilities
Because Conway's Law

Organised around Business Capabilities

THE RADAR

TECHNIQUES

ADOPT
1 Capturing client-side JavaScript errors
2 Continuous delivery for mobile devices
3 Mobile testing on mobile networks
4 Segregated DOM plus node for JS Testing
5 Windows infrastructure automation

TRIAL
6 Capture domain events explicitly
7 Client and server rendering with same code
8 HTML5 storage instead of cookies
9 Instrument all the things
10 Masterless Chef/Puppet
11 Micro-services
12 Perimeterless enterprise
13 Provisioning testing
14 Structured Logging

ASSESS
15 Bridging physical and digital worlds with simple hardware
16 Collaborative analytics and data science
17 Datensparsamkeit
18 Development environments in the cloud
19 Focus on mean time to recovery
20 Machine image as a build artifact
21 Tangible interaction

HOLD
22 Cloud lift and shift
23 Ignoring OWASP Top 10
24 Siloed metrics
25 Velocity as productivity

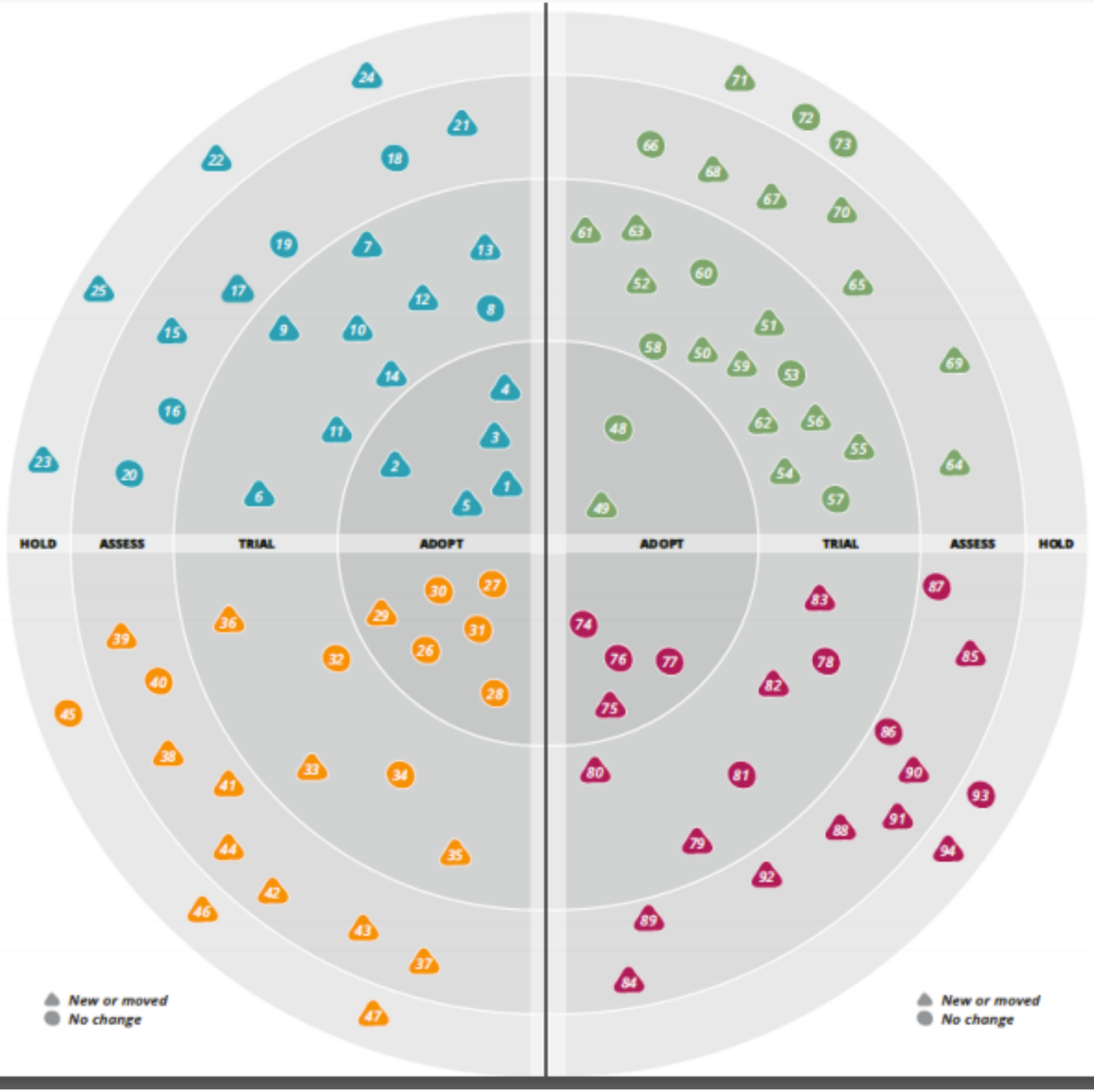
PLATFORMS

ADOPT
26 Elastic Search
27 MongoDB
28 Neo4J
29 Node.js
30 Redis
31 SMS and USSD as a UI

TRIAL
32 Hadoop 2.0
33 Hadoop as a service
34 OpenStack
35 PostgreSQL for NoSQL
36 Vumi

ASSESS
37 Akka
38 Backend as a service
39 Low-cost robotics
40 PhoneGap/Apache Cordova
41 Private Clouds
42 SPDY
43 Storm
44 Web Components standard

HOLD
45 Big enterprise solutions
46 CMS as a platform
47 Enterprise Data Warehouse



THE RADAR

TOOLS

ADOPT
48 D3
49 Dependency management for JavaScript

TRIAL
50 Ansible
51 Calabash
52 Chaos Monkey
53 Gatling
54 Grunt.js
55 Hystrix
56 Icon fonts
57 Librarian-puppet and Librarian-Chef
58 Logstash & Graylog2
59 Moco
60 PhantomJS
61 Prototype On Paper
62 SnapCI
63 Snowplow Analytics & Piwik

ASSESS
64 Cloud-Init
65 Docker
66 Octopus
67 Sensu
68 Travis for OSX/IOS
69 Visual regression testing tools
70 Xamarin

HOLD
71 Ant
72 Heavyweight test tools
73 TFS

LANGUAGES & FRAMEWORKS

ADOPT
74 Clojure
75 Dropwizard
76 Scala, the good parts
77 Sinatra

TRIAL
78 CoffeeScript
79 Go language
80 Hive
81 Play Framework 2
82 Reactive Extensions across languages
83 Web API

ASSESS
84 Elixir
85 Julia
86 Nancy
87 OWIN
88 Pester
89 Pointer Events
90 Python 3
91 TypeScript
92 Yeoman

HOLD
93 Handwritten CSS
94 JSF

Techniques

Microservice envy

HOLD

We remain convinced that microservices can offer significant advantages to organizations, in terms of improving team autonomy and faster frequency of change. The additional complexity that comes from distributed systems requires an additional level of maturity and investment. We are concerned that some teams are rushing in to adopting microservices without understanding the changes to development, test, and operations that are required to do them well. Our general advice remains simple. Avoid **microservice envy** and start with one or two services before rushing headlong into developing more, to allow your teams time to adjust and understand the right level of granularity.

Infrastructure Complexity

- Operational Complexity is the biggest problem
- Lots of Manual Testing, massive UI testing suit
- Beware of Deployment Dependency trap
- UI Composition

Service Boundaries

- Having strong module bounders is a big benefit of Microservices
- Deciding on these boundaries this can turn into a massive nightmare

Higher Cost of Refactoring

- Refactoring a concept that spans across multiple smaller services is hard

CHARACTERISTICS & PREMIUM

Componentization via Services

Independent Data Storage

Infrastructure Automation

Design for Failure

Organised around Business Capabilities

Infrastructure Complexity

Higher Cost of Refactoring

Service Boundaries

WHERE TO USE MICROSERVICES?

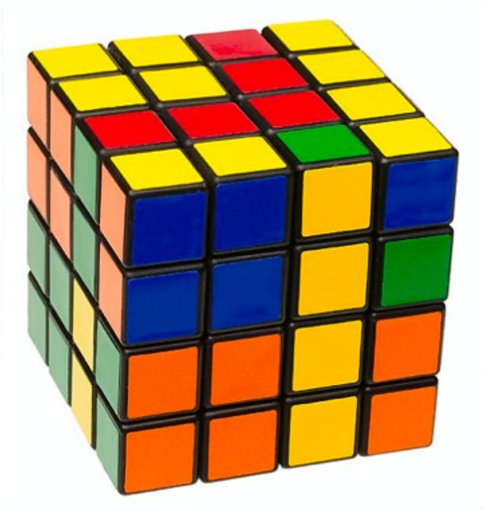
- Microservices are suitable for very complex systems

Don't even consider microservices unless you have a system that's too complex to manage as a monolith.

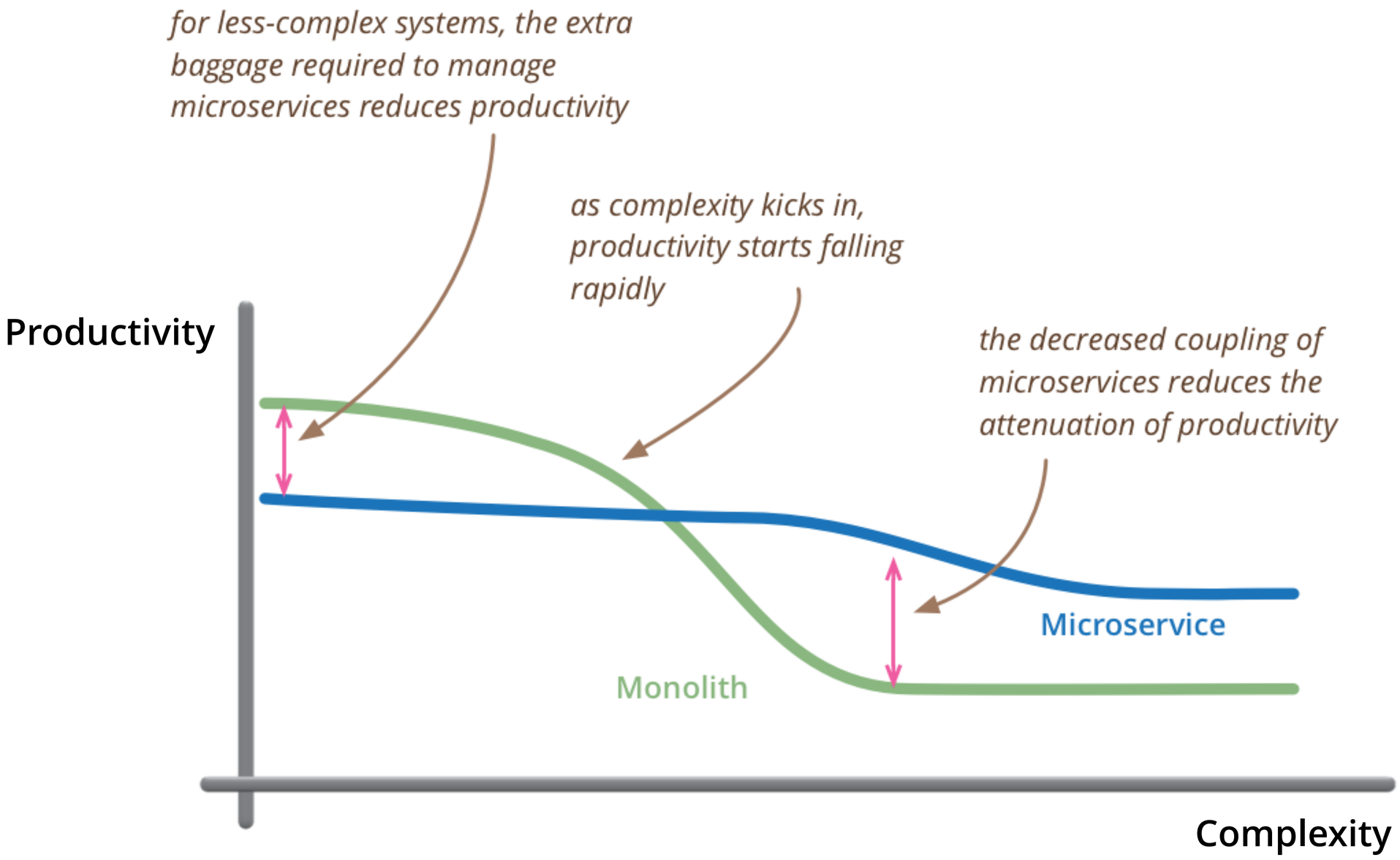
WHERE TO USE MICROSERVICES?

You shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile!

MONOLITH FIRST STRATEGY



WHERE TO USE MICROSERVICES?

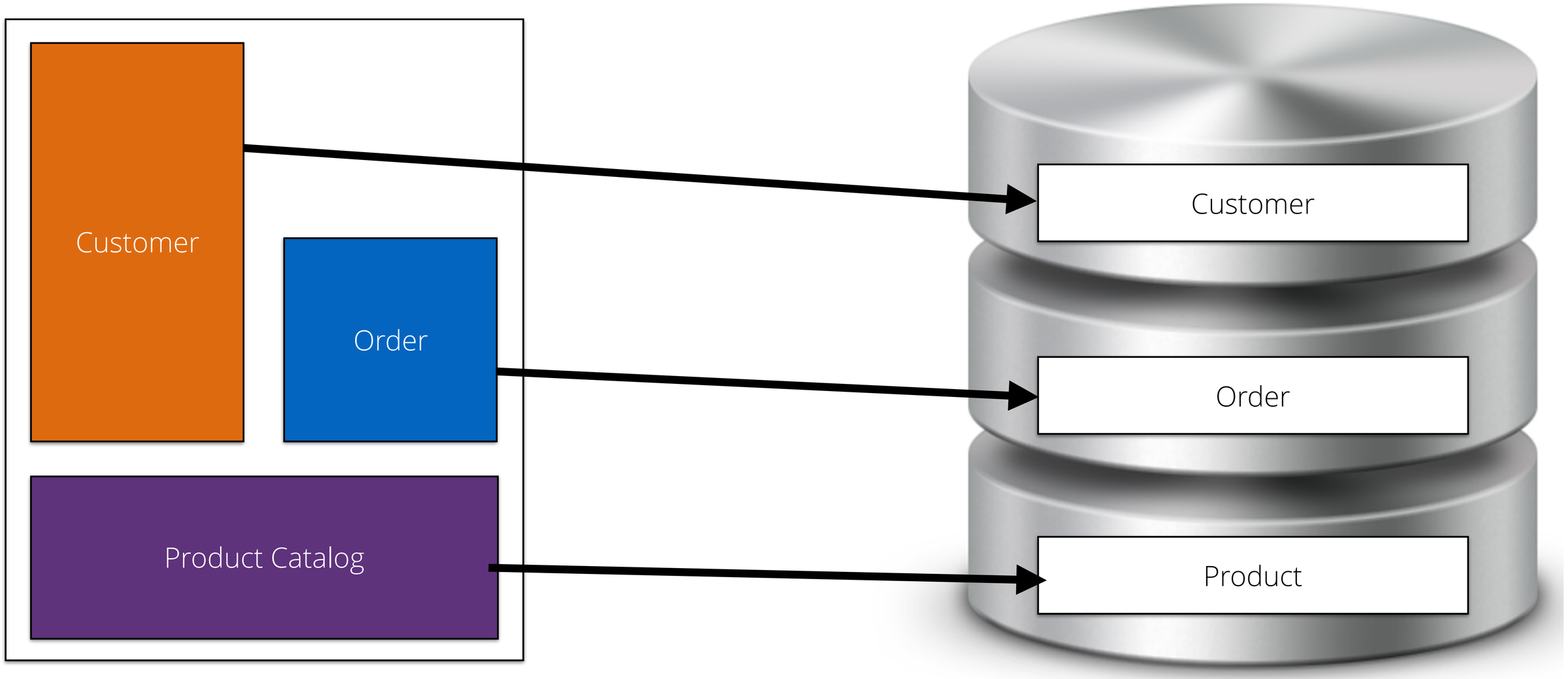


MONOLITH-FIRST STRATEGY

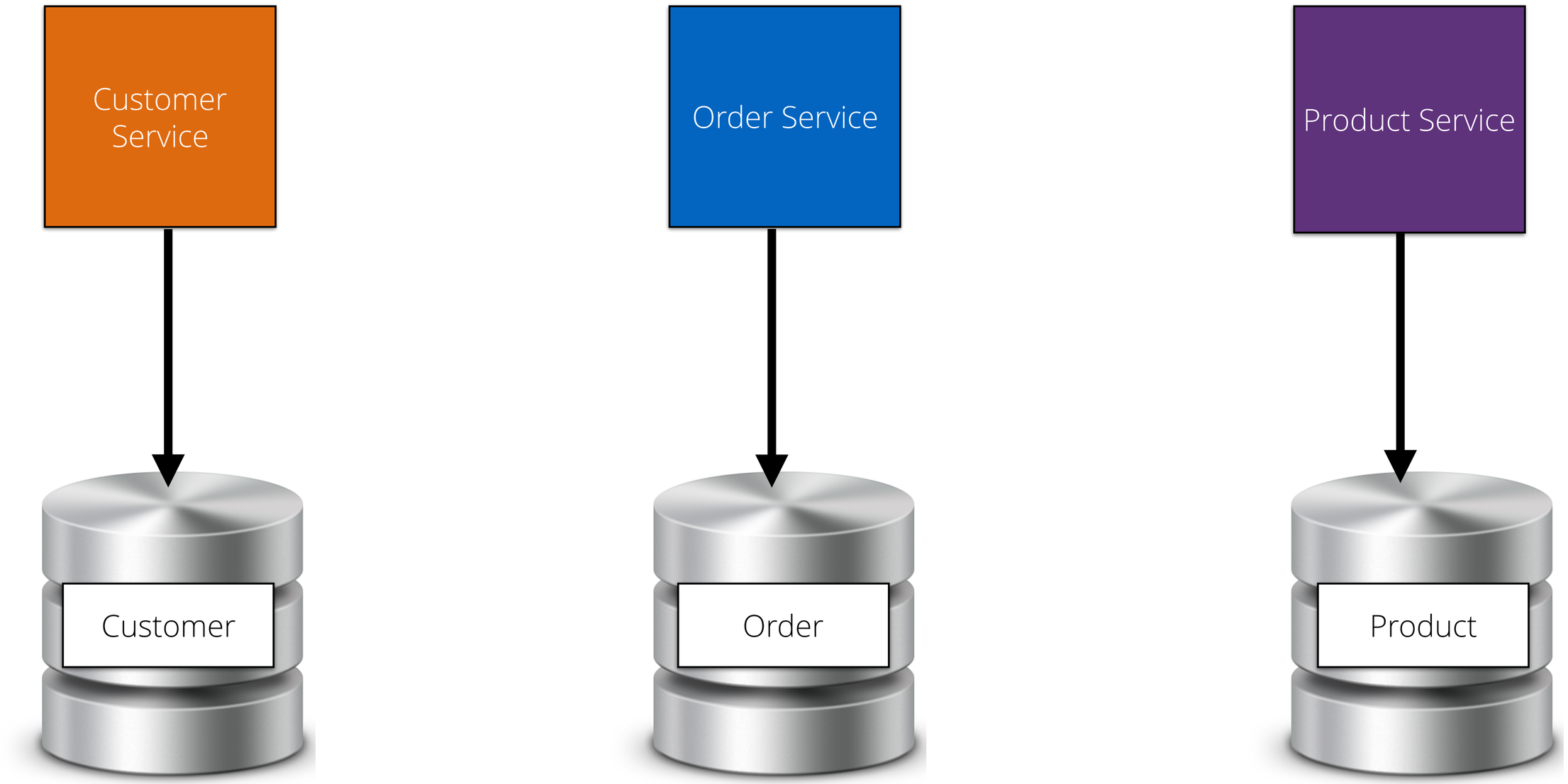
- Paying attention to modularity within the software, both at the API boundaries and how the data is stored

- Do this well, and it's a relatively simple matter to make the shift to microservices.

MONOLITH-FIRST STRATEGY

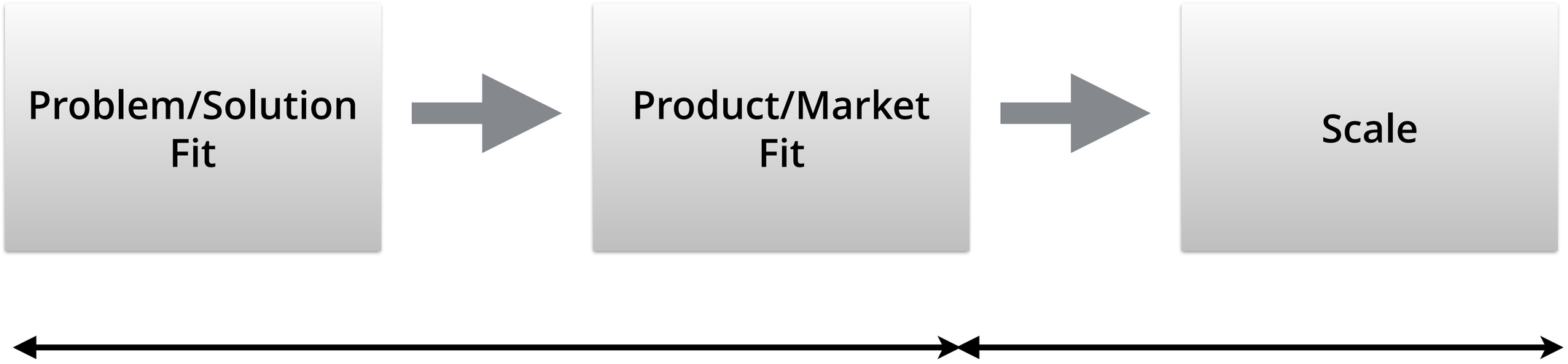


MONOLITH-FIRST STRATEGY



WHEN?

Three stages of a project



Focus: Learning and Validation

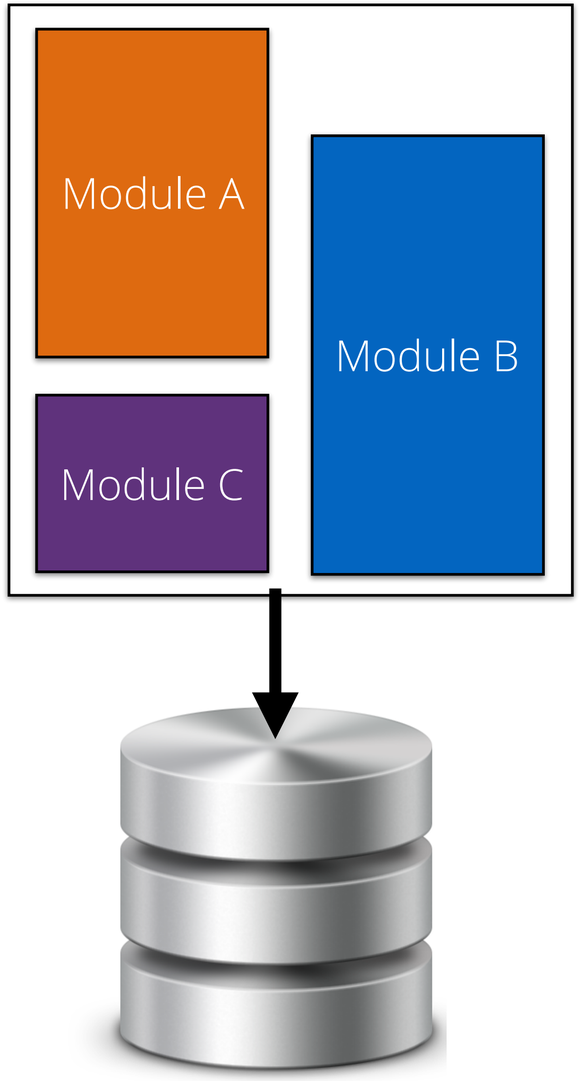
Focus: Growth and Optimisation

WHEN?

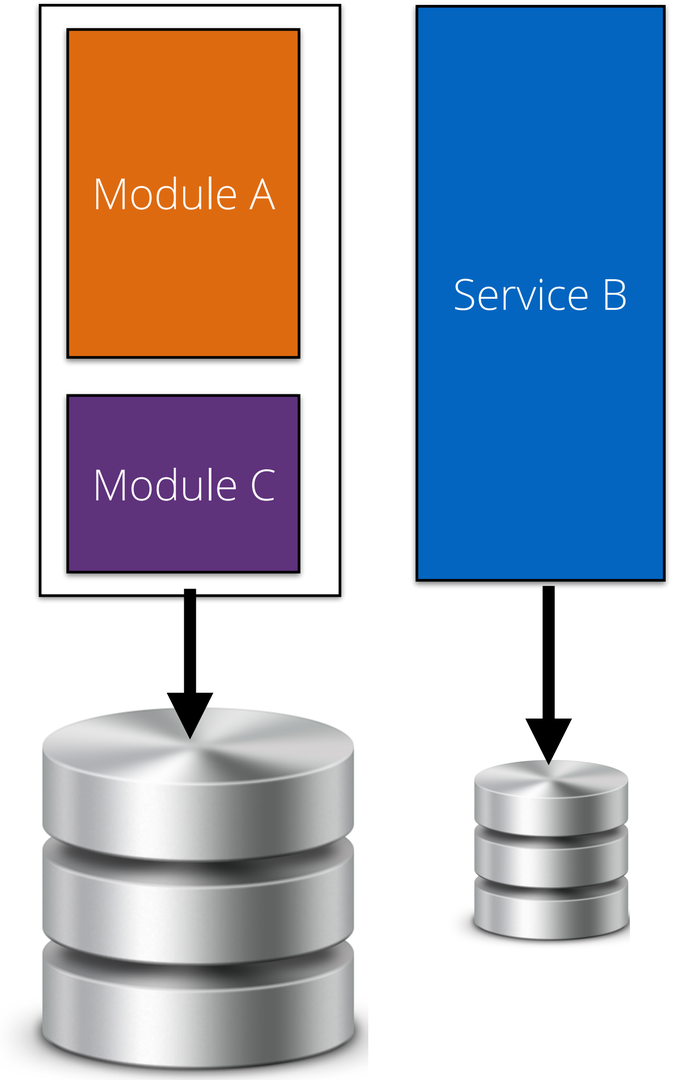
It depends...

- **Technical**: Are we mature in our CD practices?
- **Organisational**: Can we organise our teams around business capabilities?
- **Product**: Do we have product/market fit?

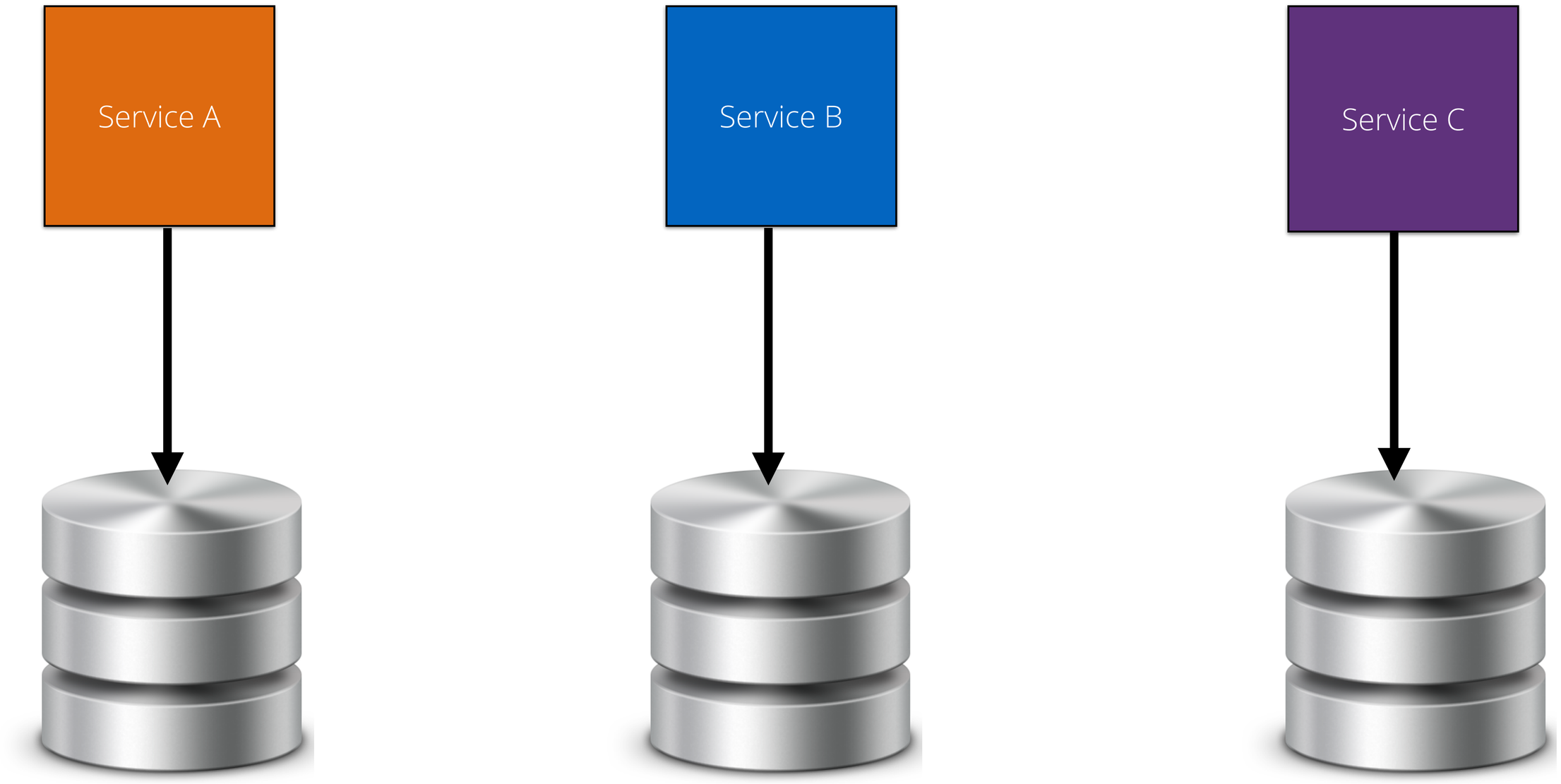
BROWN FIELD



BROWN FIELD



BROWN FIELD



This is a journey

- Expect to change team structure more than once
- Expect to continue to refine architecture.
- Be comfortable with continuous change.

ThoughtWorks®

THANK YOU!

http://nima.tech/micro-services

@nimamon