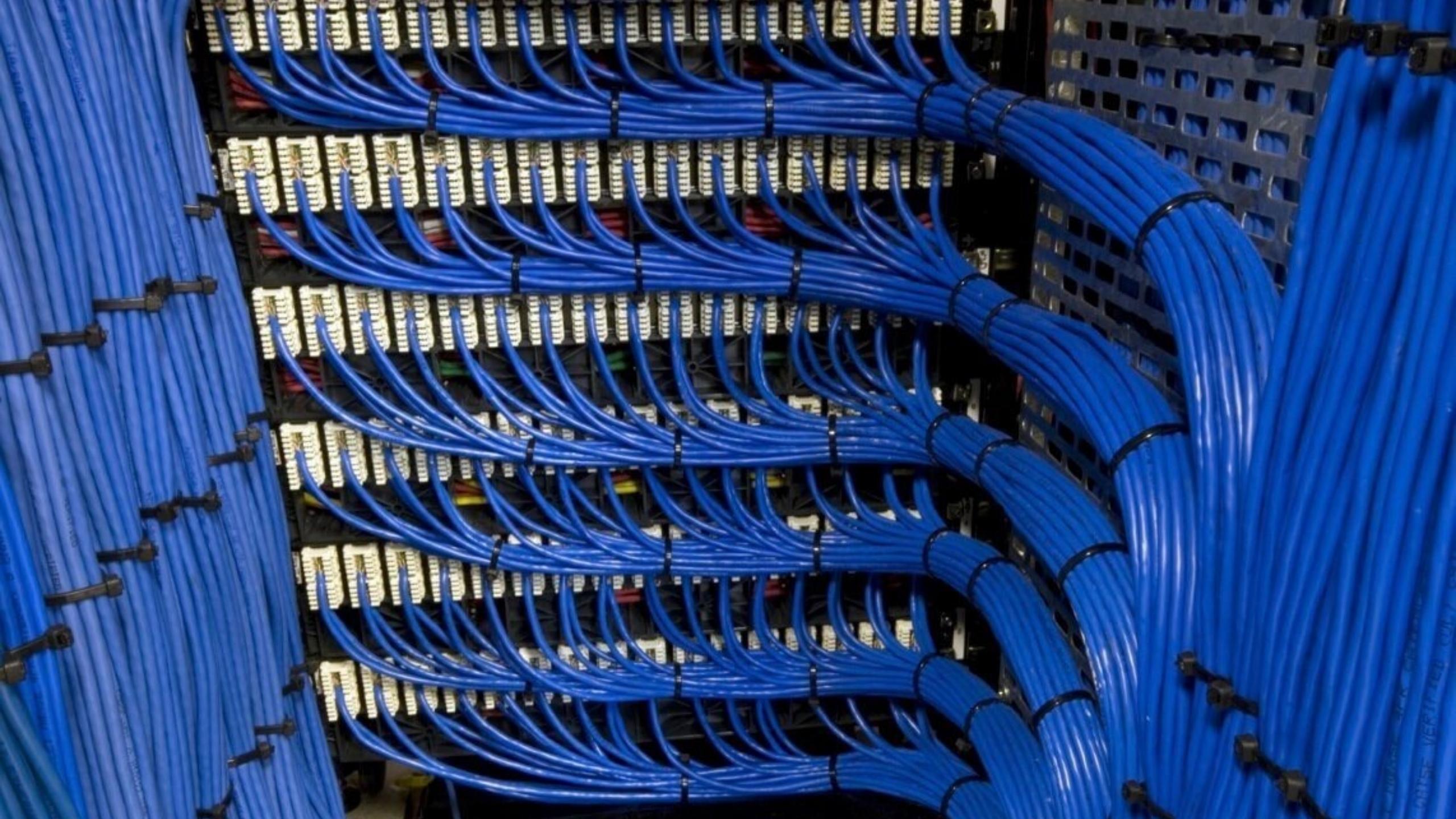
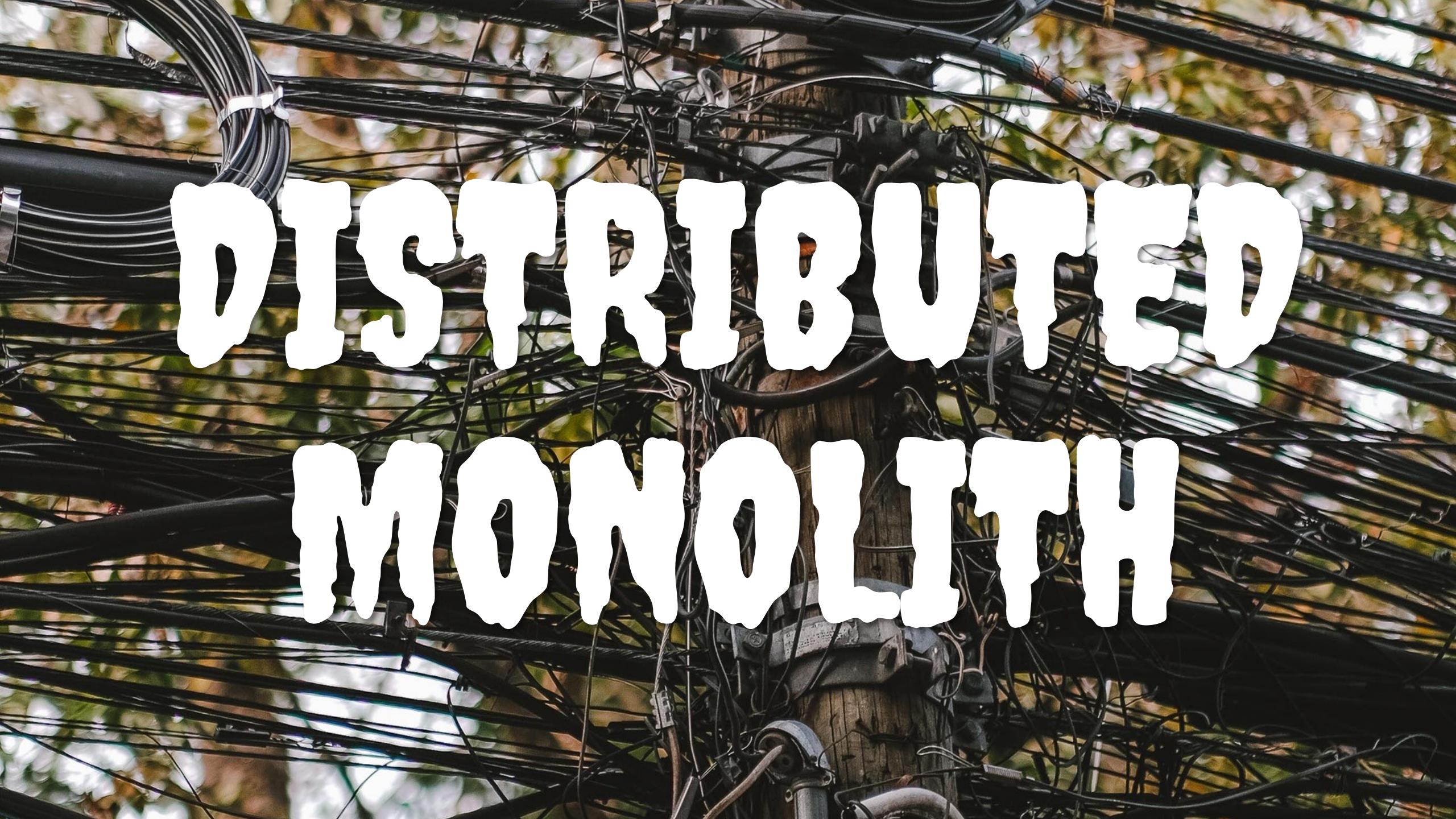


Don't Build a Distributed Monolith

How to Avoid Doing Microservices Completely Wrong

Jonathan "J." Tower



A dense web of black electrical wires against a blurred background of foliage.

DISTRIBUTED
MONOLITH

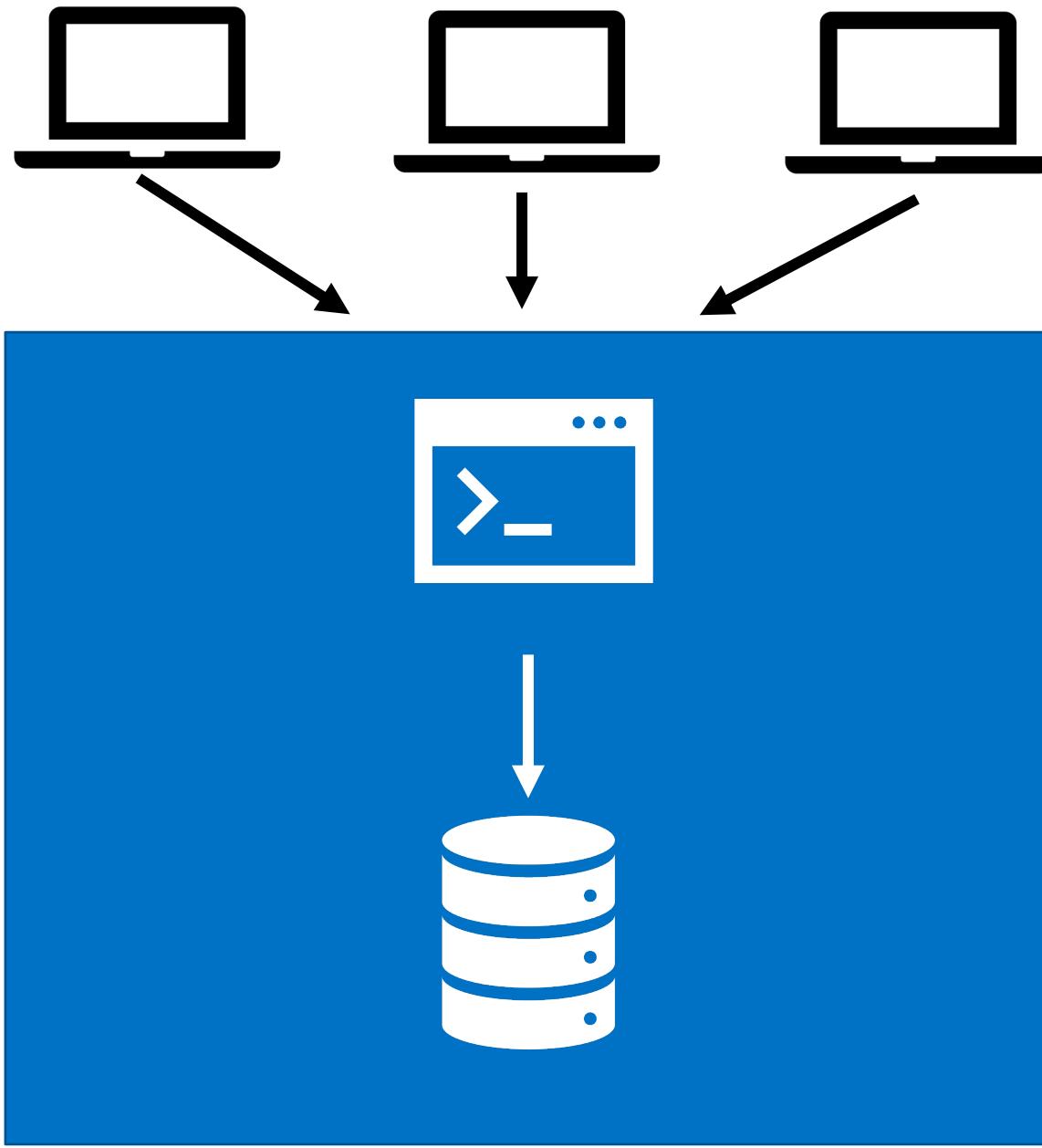
The text "DISTRIBUTED MONOLITH" is displayed in large, white, bold, sans-serif letters. The word "DISTRIBUTED" is positioned above the word "MONOLITH". Both words have a thick, three-dimensional appearance, with visible shadows and highlights that give them a metallic or plastic texture. The background is a complex, tangled mess of black electrical wires crisscrossing against a backdrop of out-of-focus green and yellow leaves, suggesting a chaotic or industrial setting.

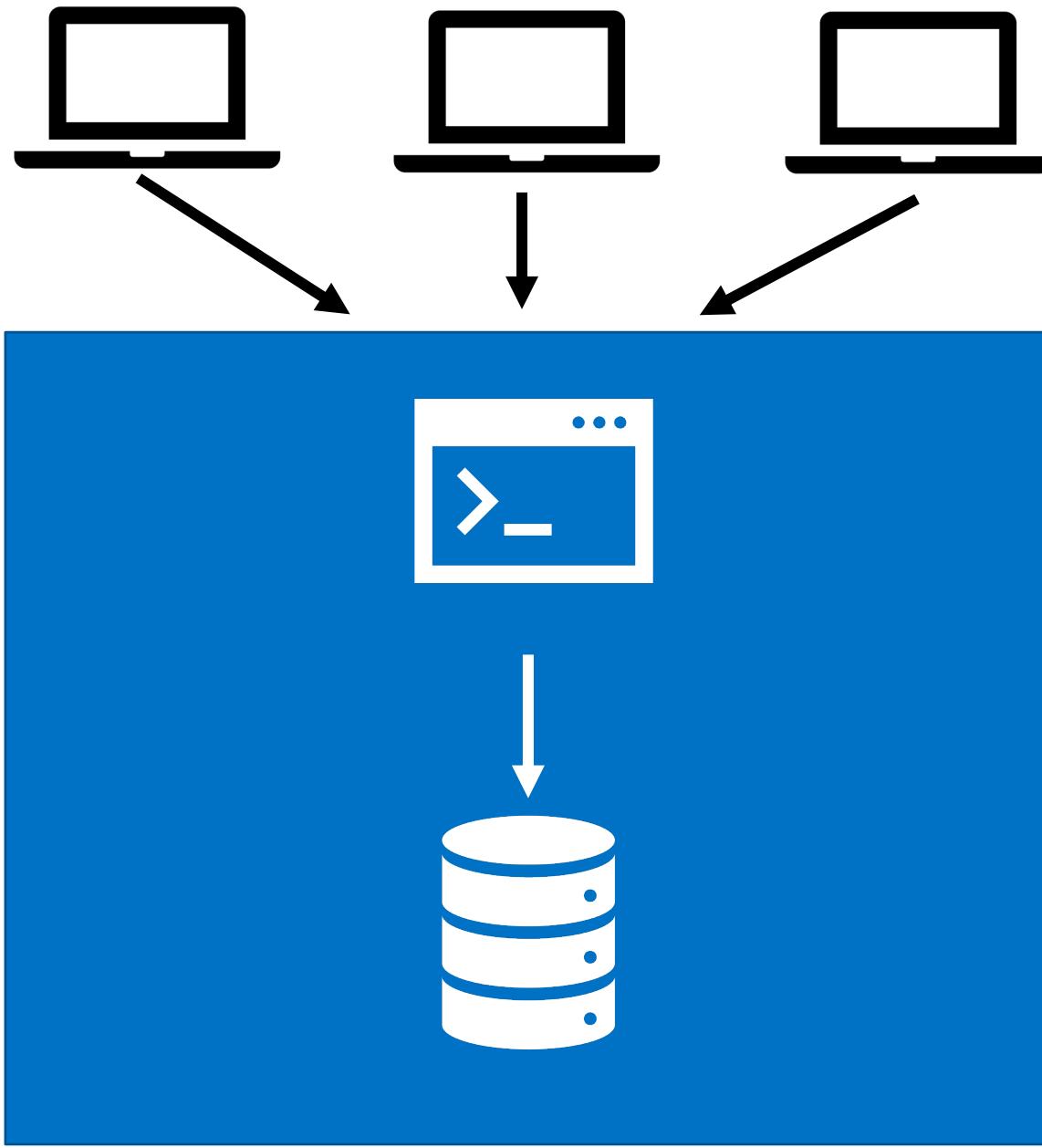
Avoid the **10 Most Common** Mistakes Made When Building Microservices

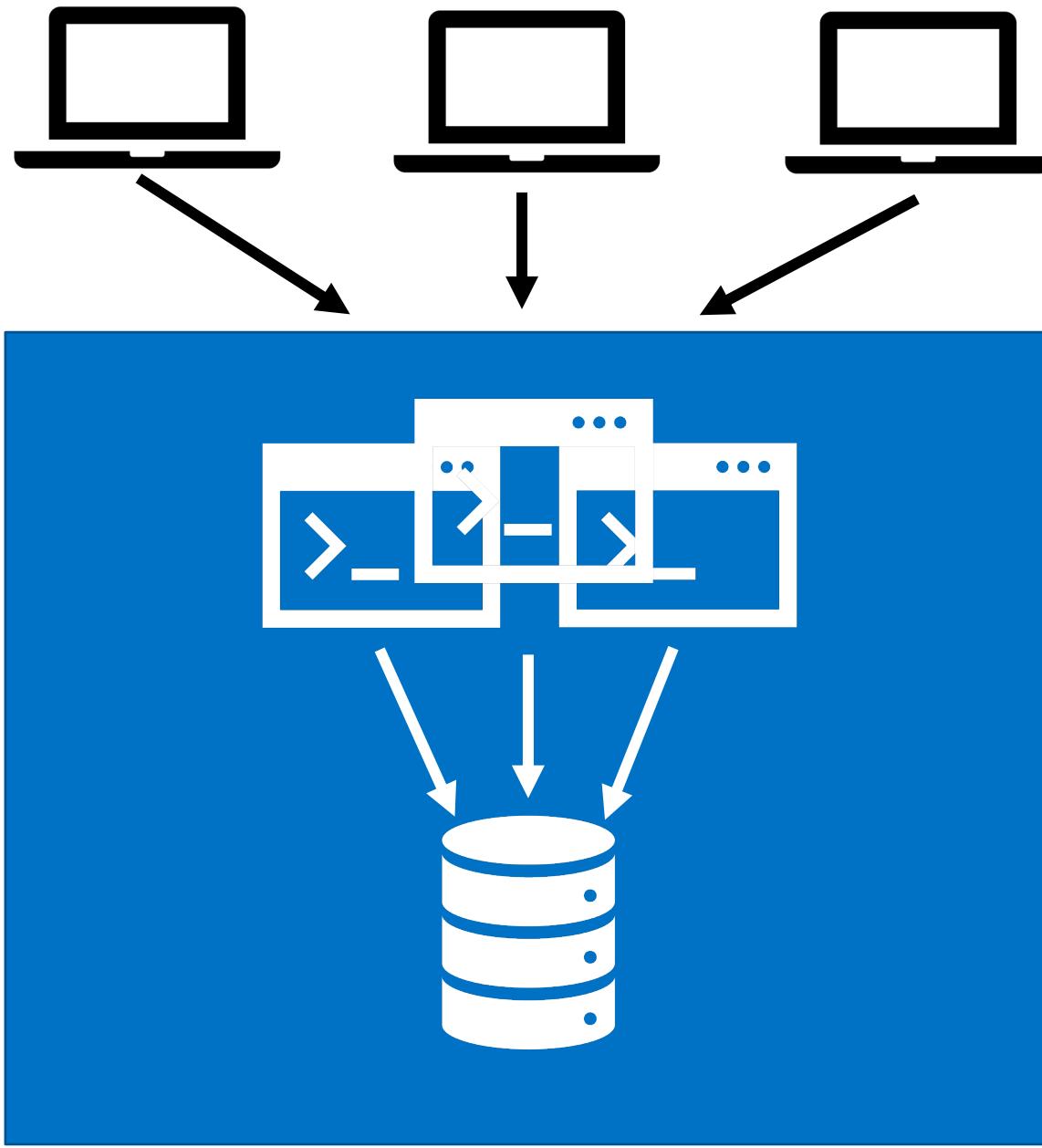
Some Definitions

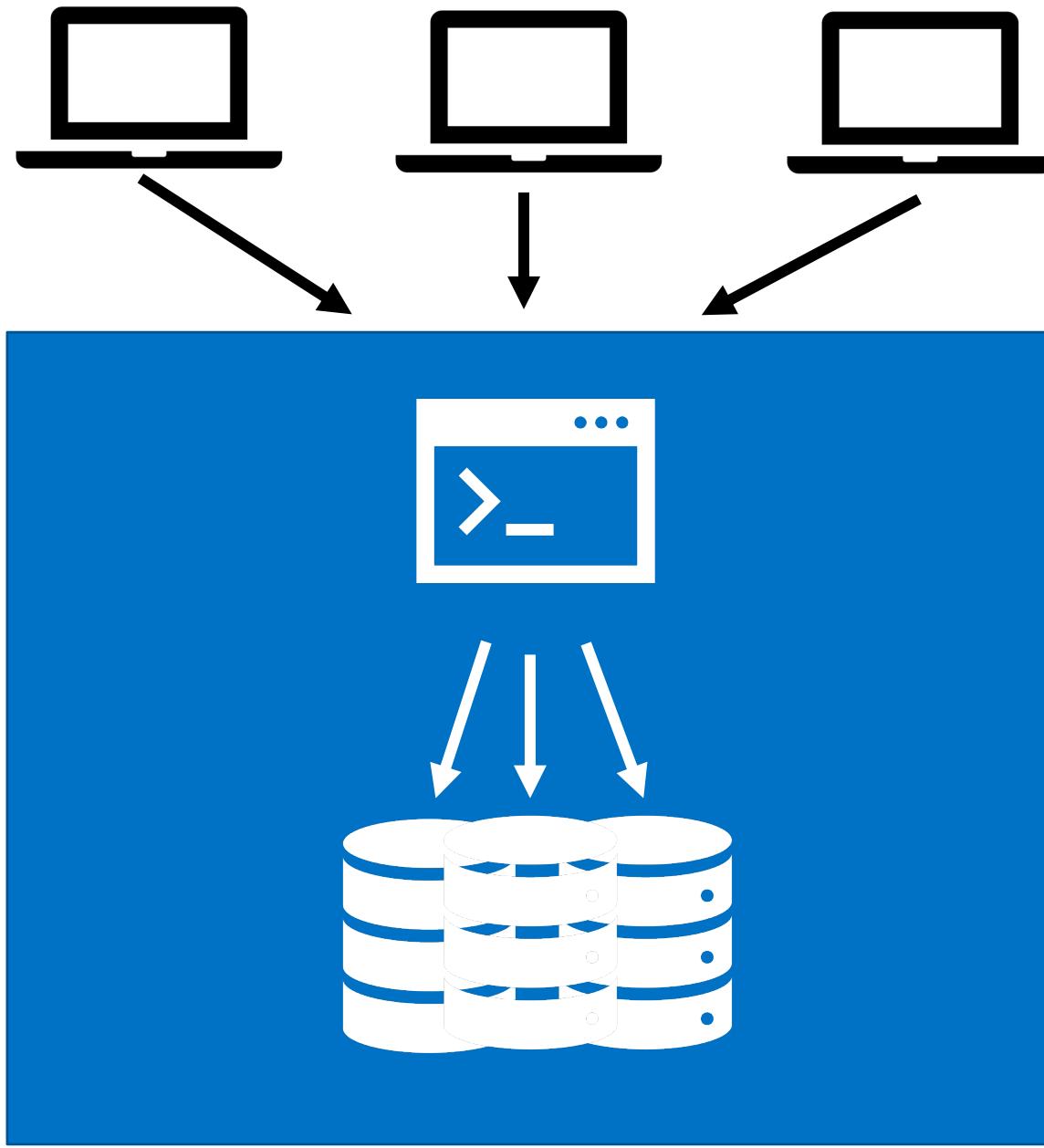
Monolith









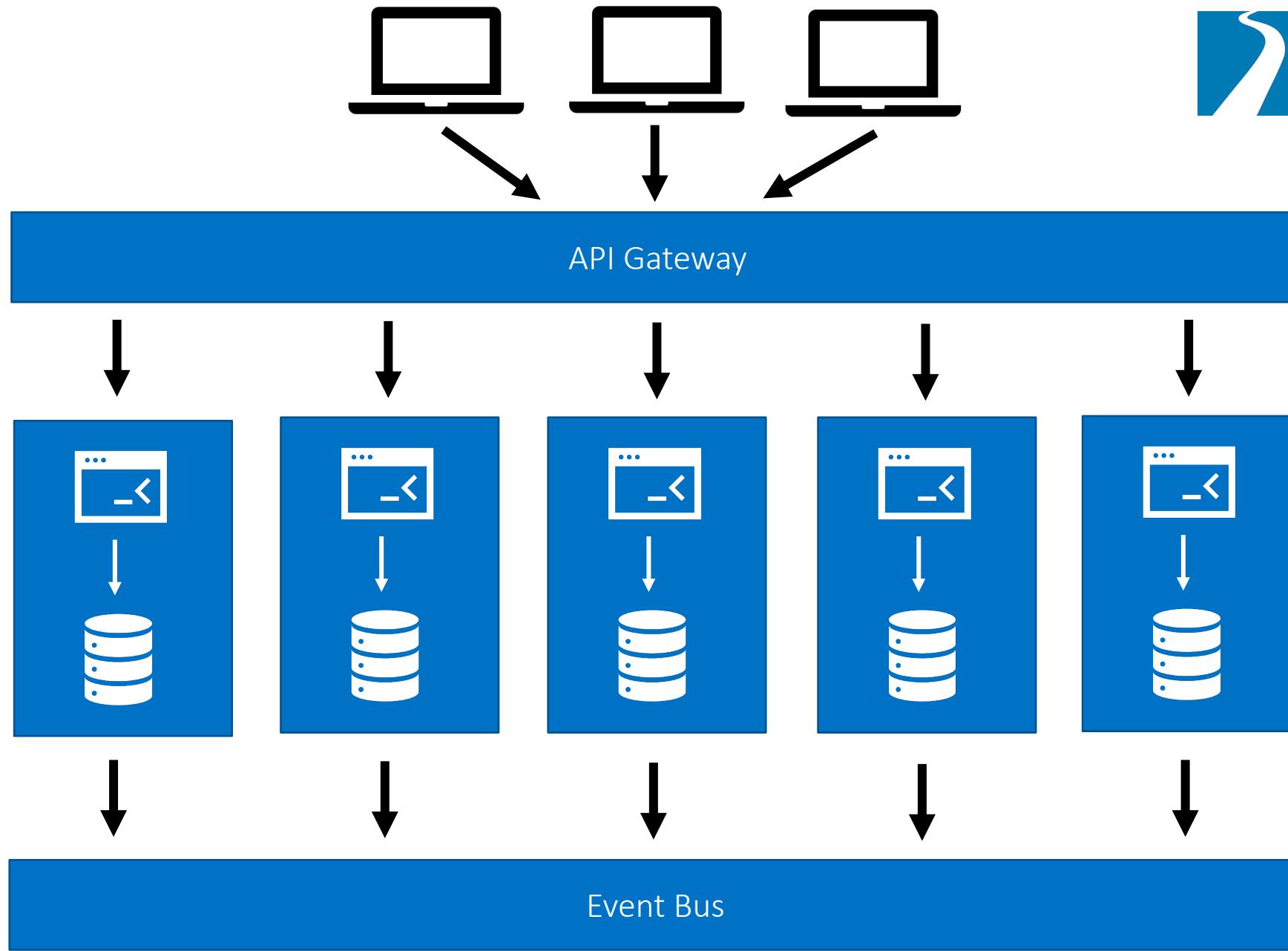


Microservices





TRAILHEAD
TECHNOLOGY PARTNERS

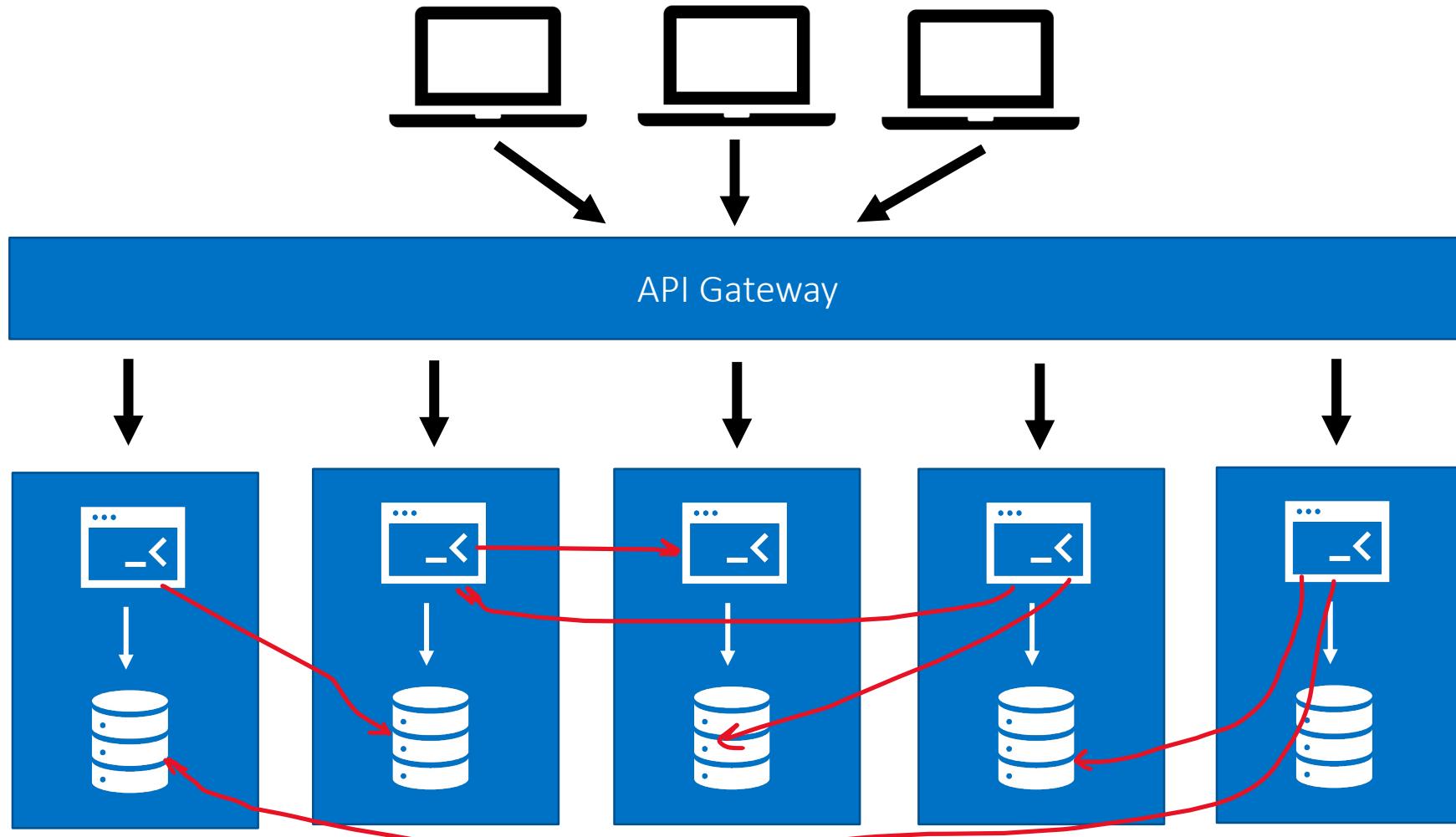


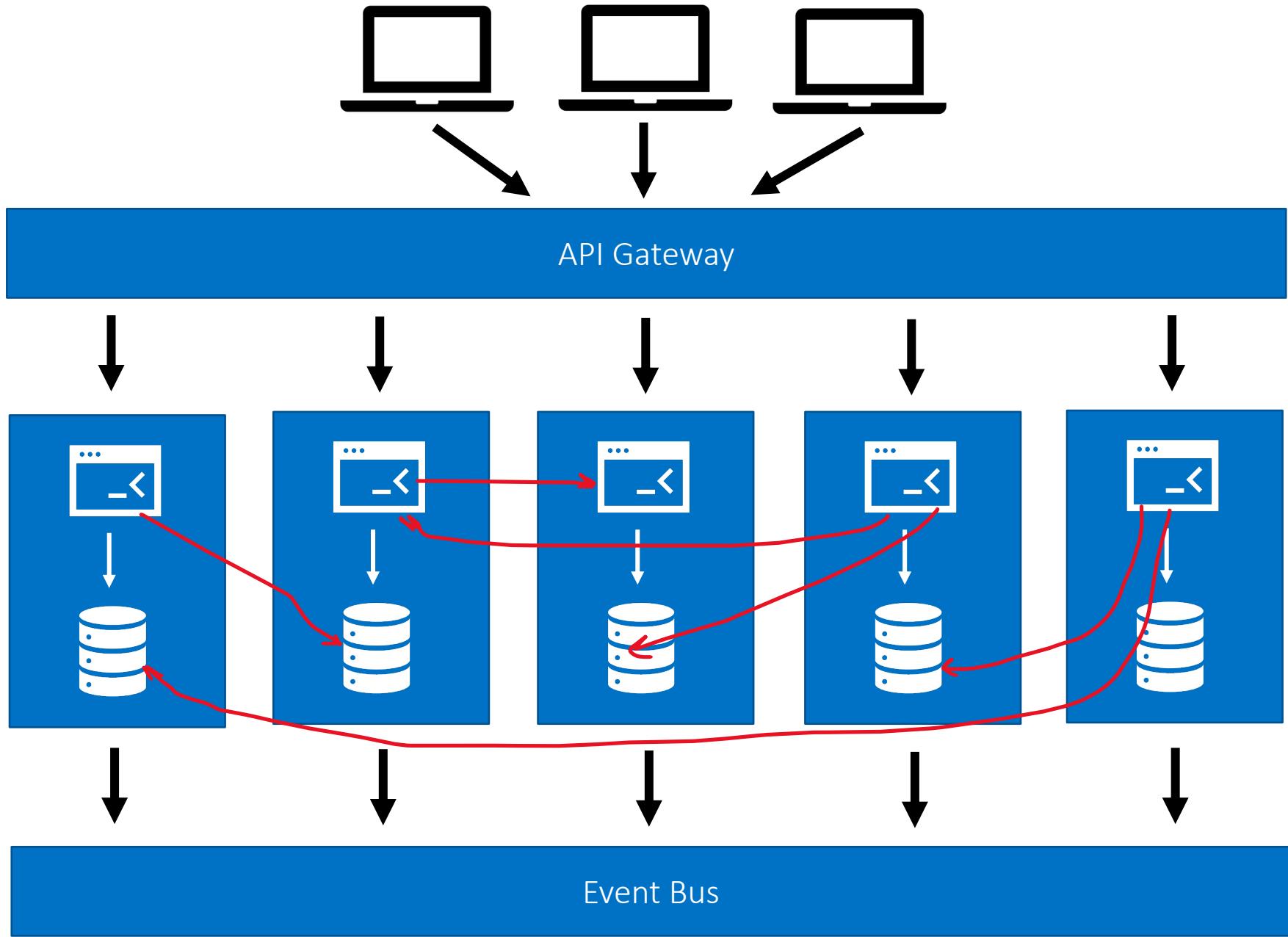
Distributed Monolith

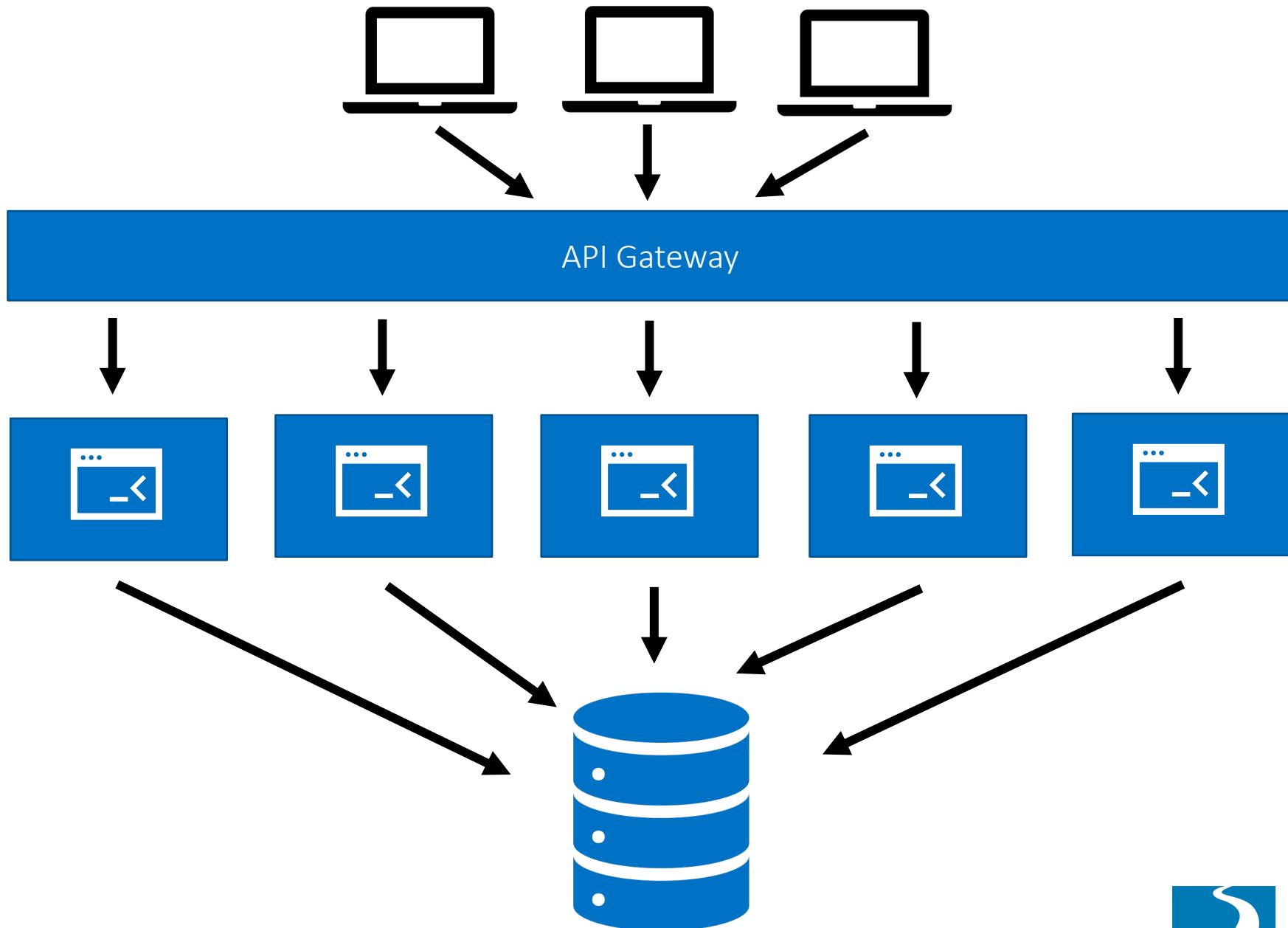


Distributed Monolith

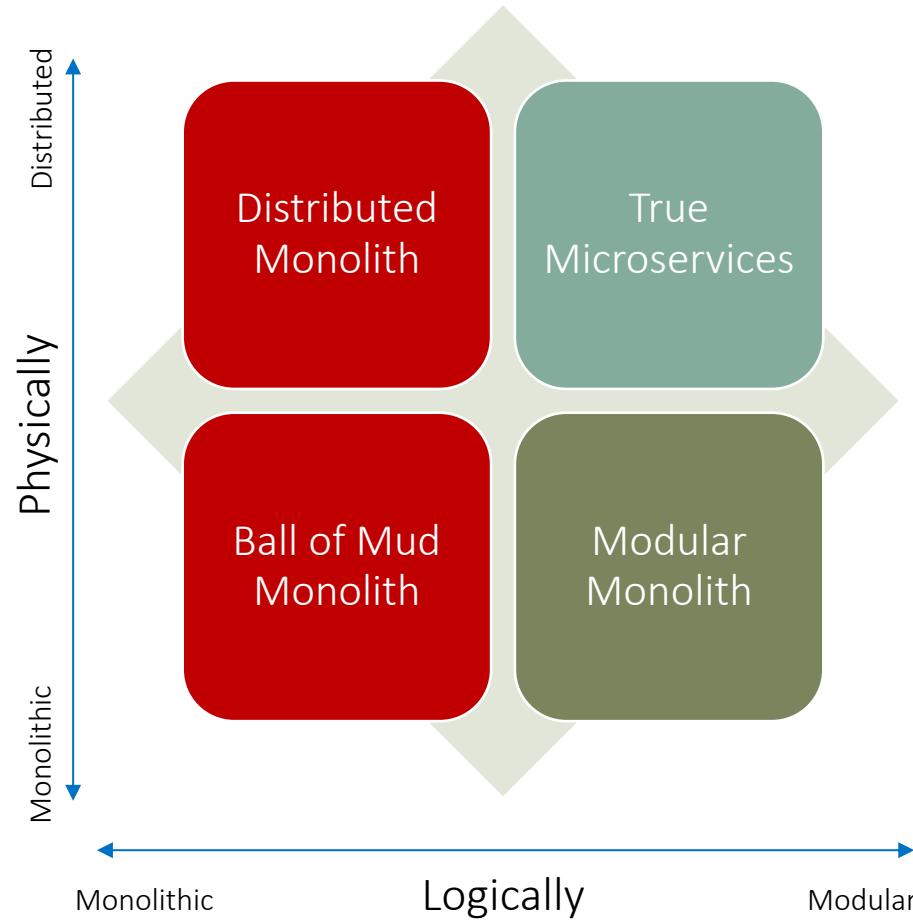








Good vs Bad Monoliths





10 Most Common Mistakes

Avoid Creating a “Monster”

Assuming Microservices are Always Better

Problem #1

First Rule of Microservices: Don't Use Microservices

Have a “Really Good Reason” – Sam Newman



Monoliths aren't inherently bad or un-scalable



Microservices are *hard* to do well



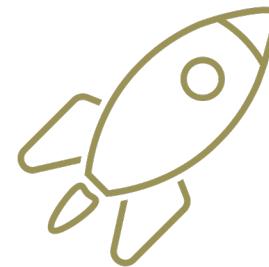
Wrong reasons create distributed monoliths

Some Good Reasons to Microservice

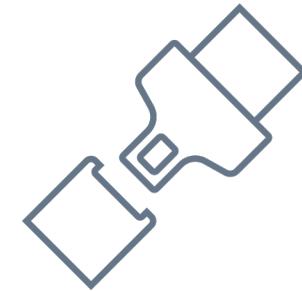
“Microservice architectures **buy you options**” – James Lewis



More Scalability
Options



Independent
Deployability



Isolate Surface Area
of Failure

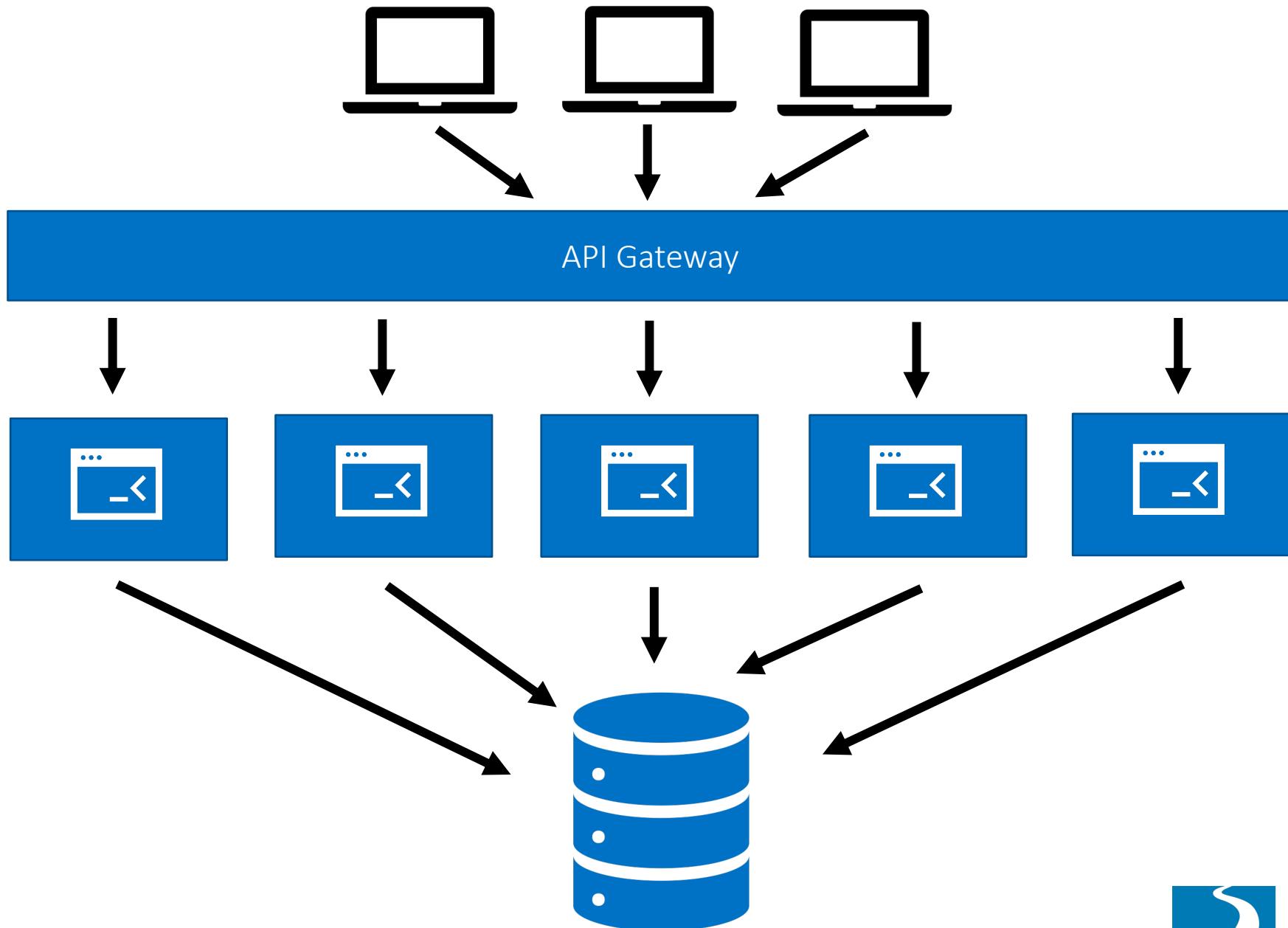
The Big Trade Off

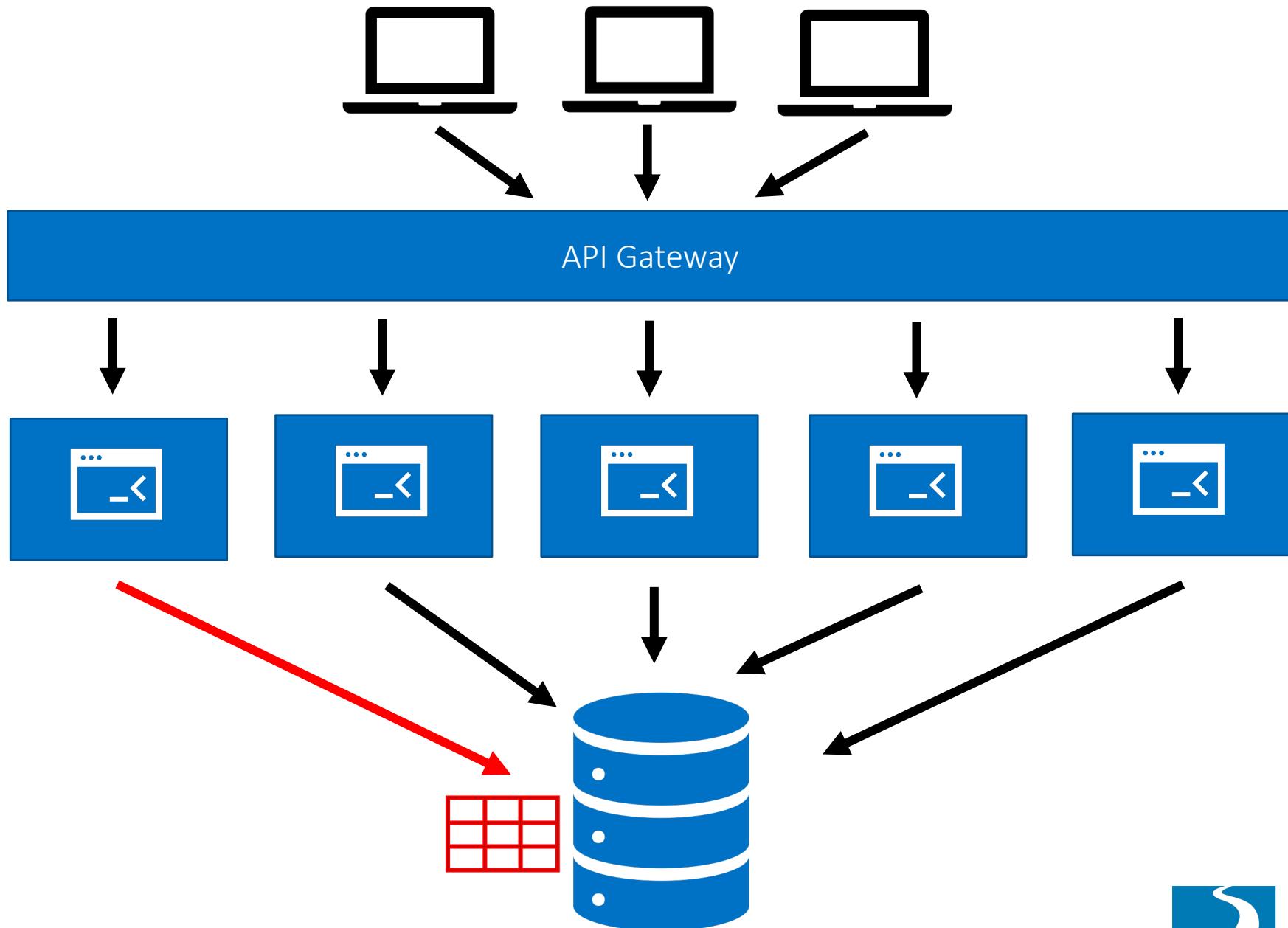


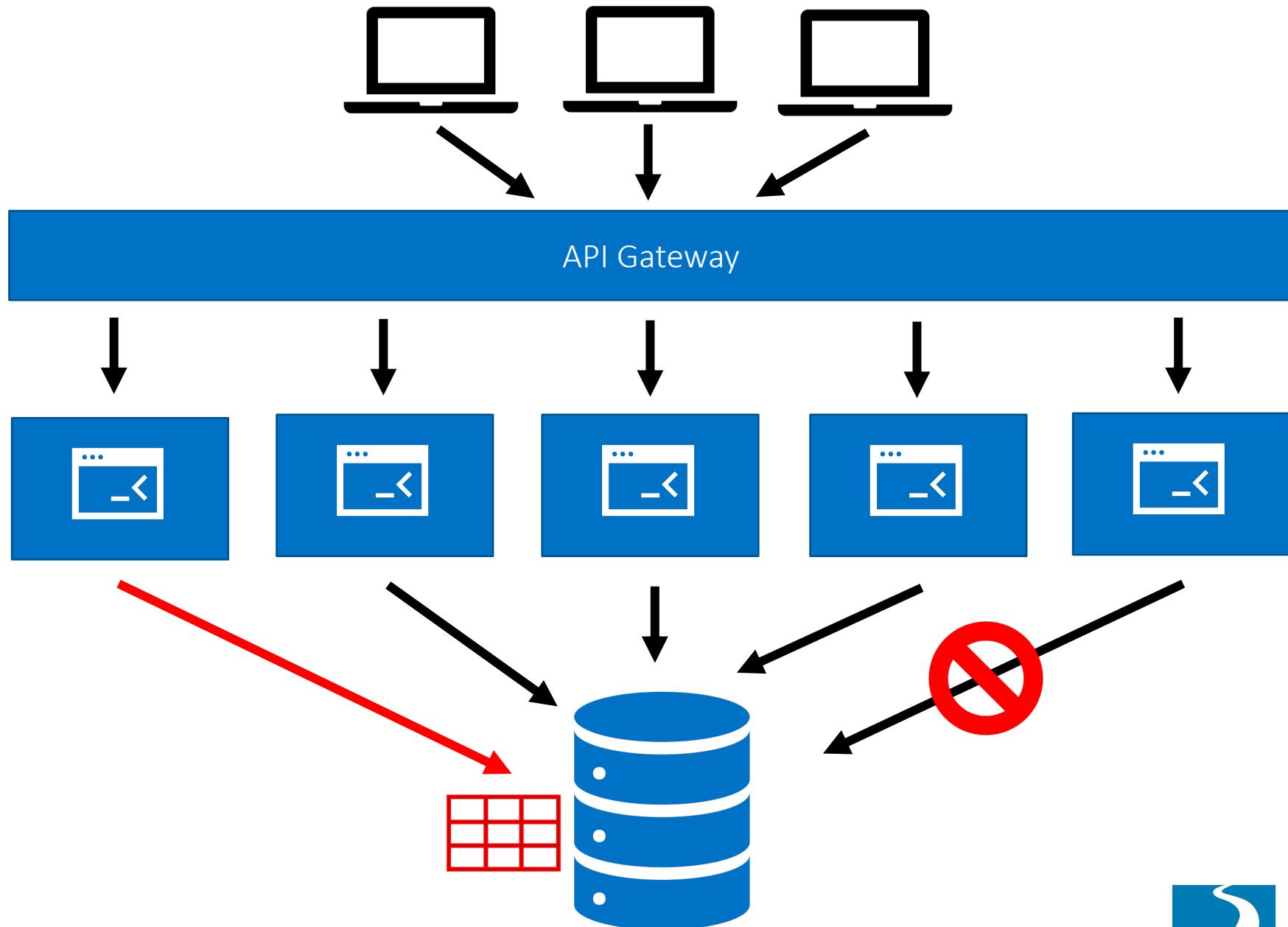
Shared Data Store or Models

Problem #2









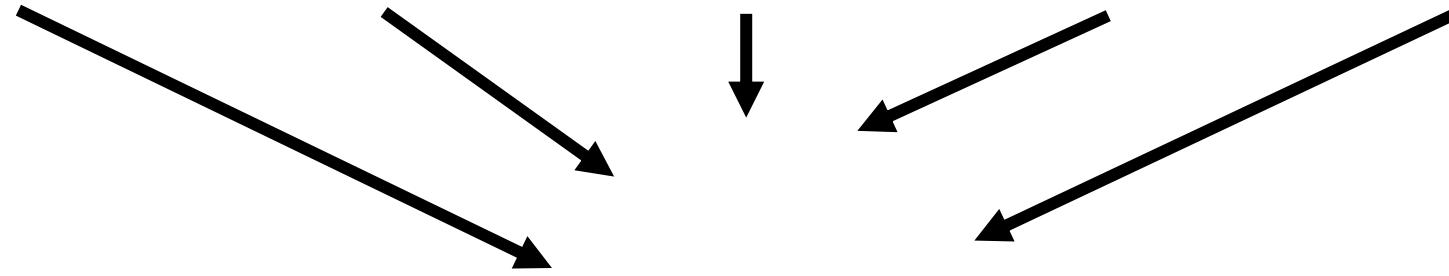
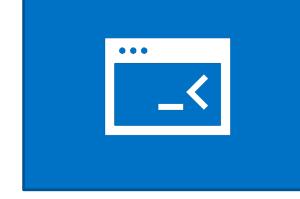
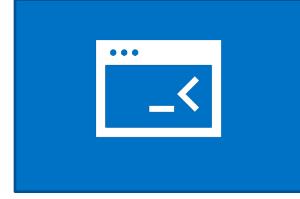
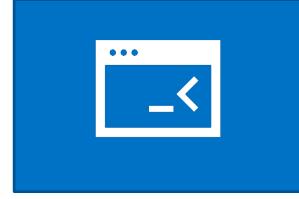
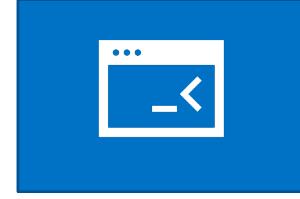
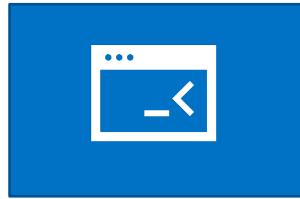
Reservations

...

Maintenance

...

Housekeeping



```
HotelRoom {  
    CurrentlyOccupied : bool  
    NumberofBeds : int  
    MinutesToClean : float  
    RepairHistory: [  
        { RepairDate: ... },  
        { RepairDate: ... },  
    ]  
}
```



TRAILHEAD
TECHNOLOGY PARTNERS

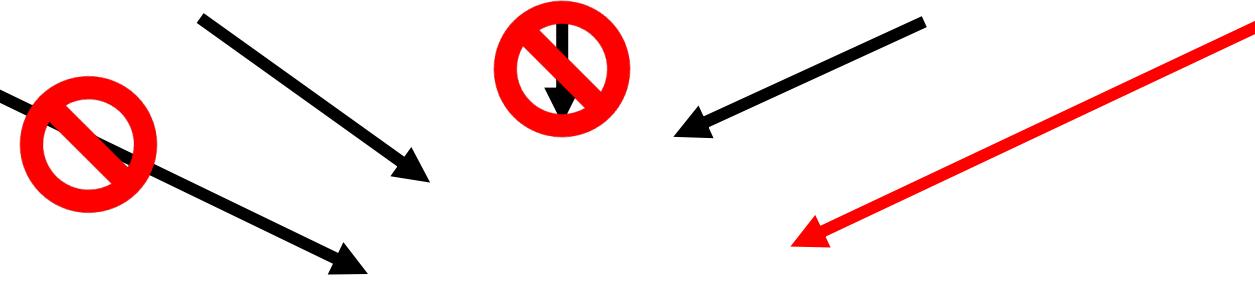
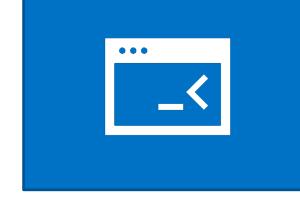
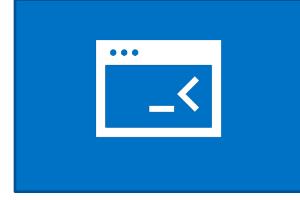
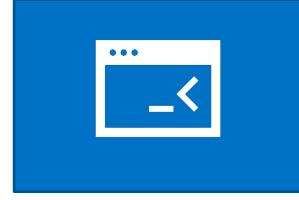
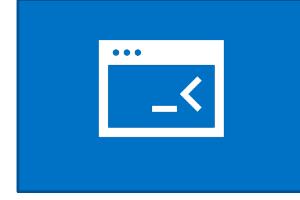
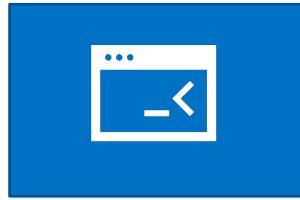
Reservations

...

Maintenance

...

Housekeeping



```
HotelRoom {  
    CurrentlyOccupied : bool  
    NumberofBeds : int  
    MinutesToClean : int float  
    RepairHistory: [  
        { RepairDate: ... },  
        { RepairDate: ... },  
    ]  
}
```



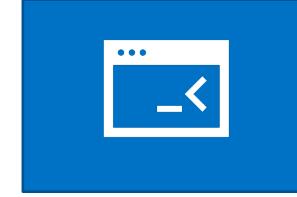
TRAILHEAD
TECHNOLOGY PARTNERS

Reservations



...

Maintenance



...

Housekeeping



```
HotelRoom {  
    RepairHistory: [  
        { RepairDate: ... },  
        { RepairDate: ... },  
    ]  
}
```

```
HotelRoom {  
    CurrentlyOccupied : bool  
    NumberofBeds : int  
}
```



```
HotelRoom {  
    MinutesToClean : int  
}
```



TRAILHEAD
TECHNOLOGY PARTNERS

Microservices That Are Too Big

Problem #3

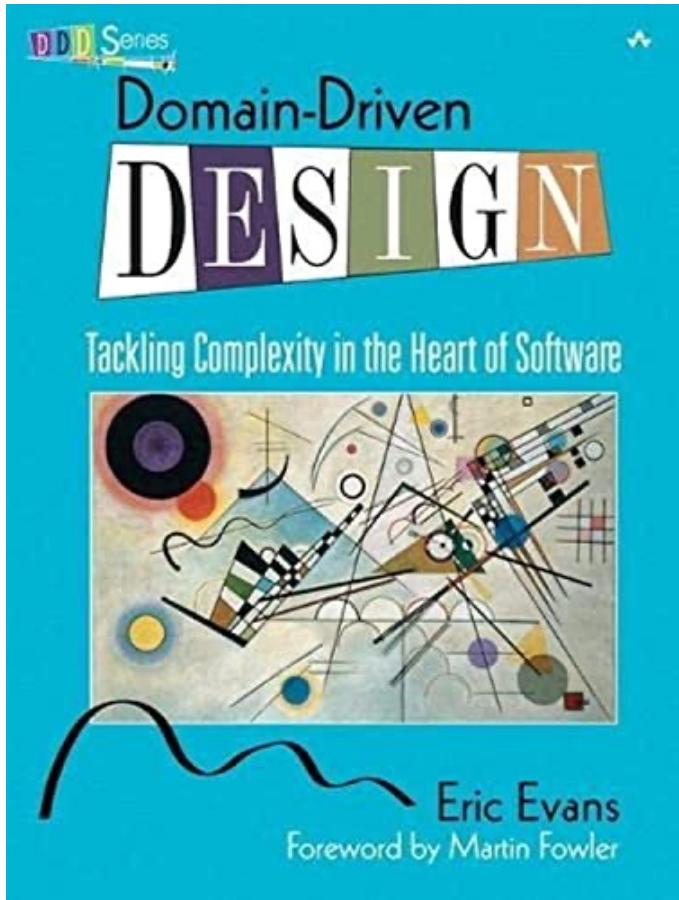


Domain Driven Development

Domain
Subdomain
Bounded Context

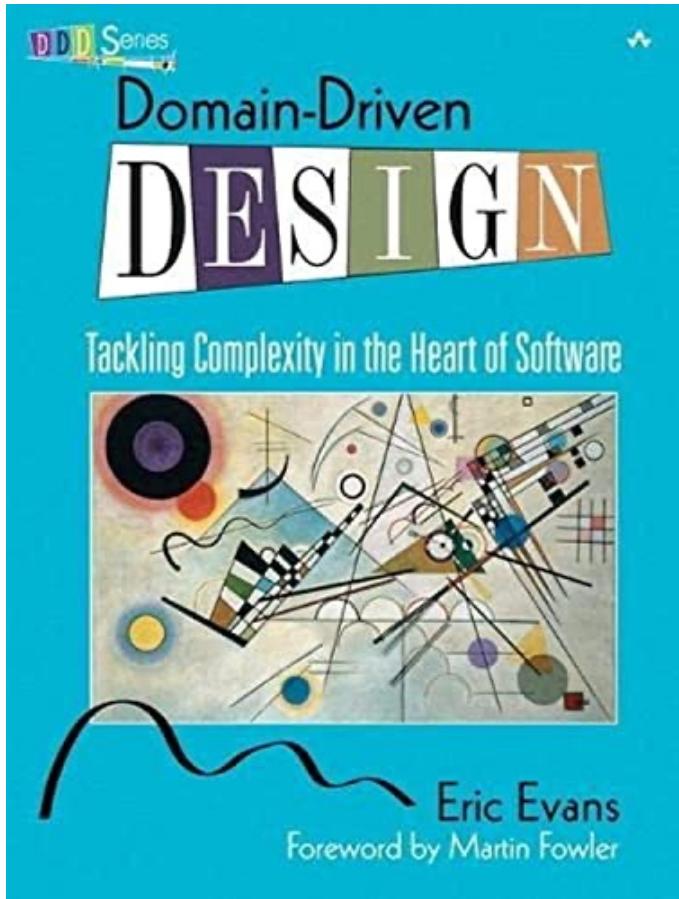


Domain Driven Development



Domain
Subdomain
Bounded Context

Domain Driven Development



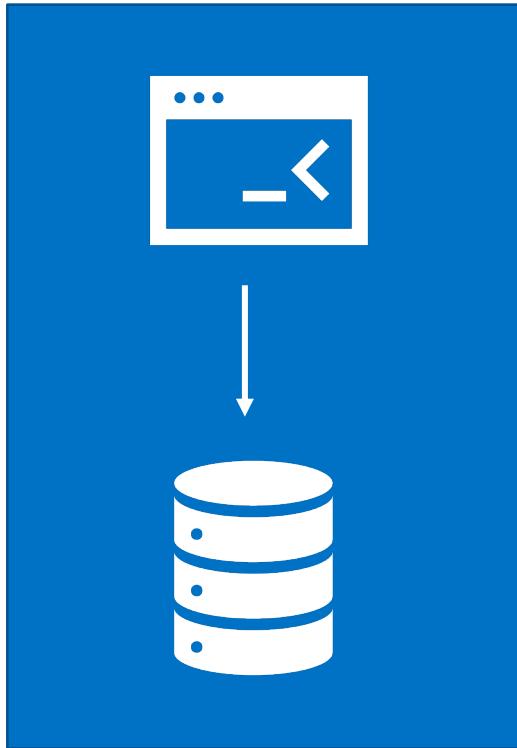
Domain
Subdomain
Bounded Context

SIMPLE RULE: smallest possible microservices without chatty communication between services

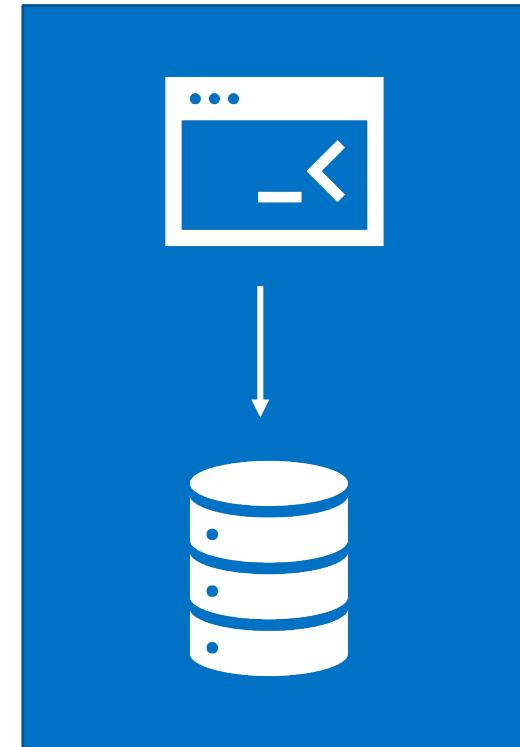
Microservices That Are Too Small

Problem #4

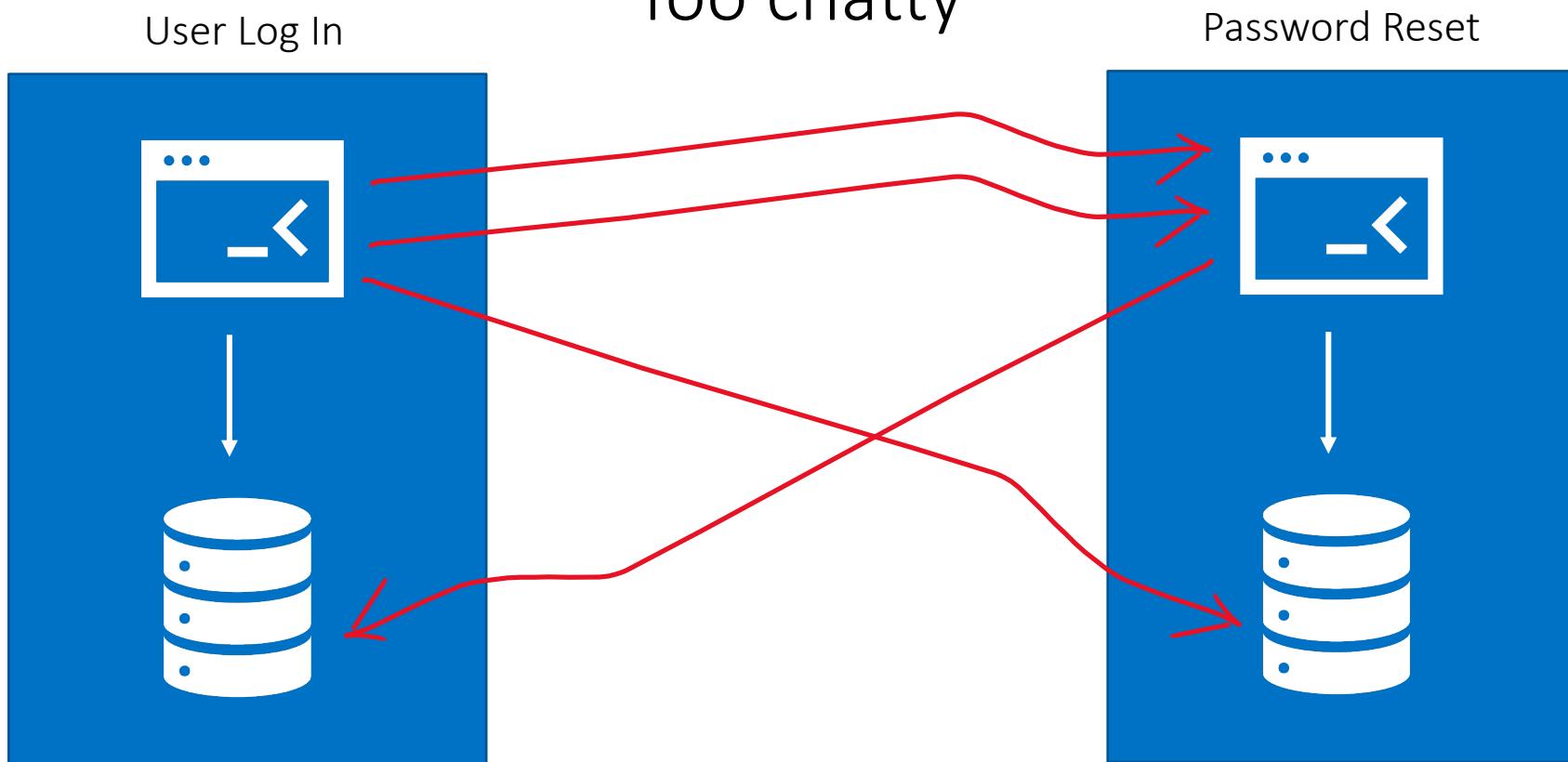
User Log In



Password Reset



Too chatty



Starting from Scratch

Problem #5



Greenfield is Actually Harder

Easier to partition an existing, "brownfield" system

Brownfield → Microservices Advantages:

1. Code and relationships to examine
2. People to talk to who know the system
3. A system that already works
4. Baseline to compare to refactoring

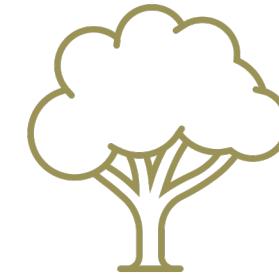
Three “Brownfield” Migration Approaches



Big bang

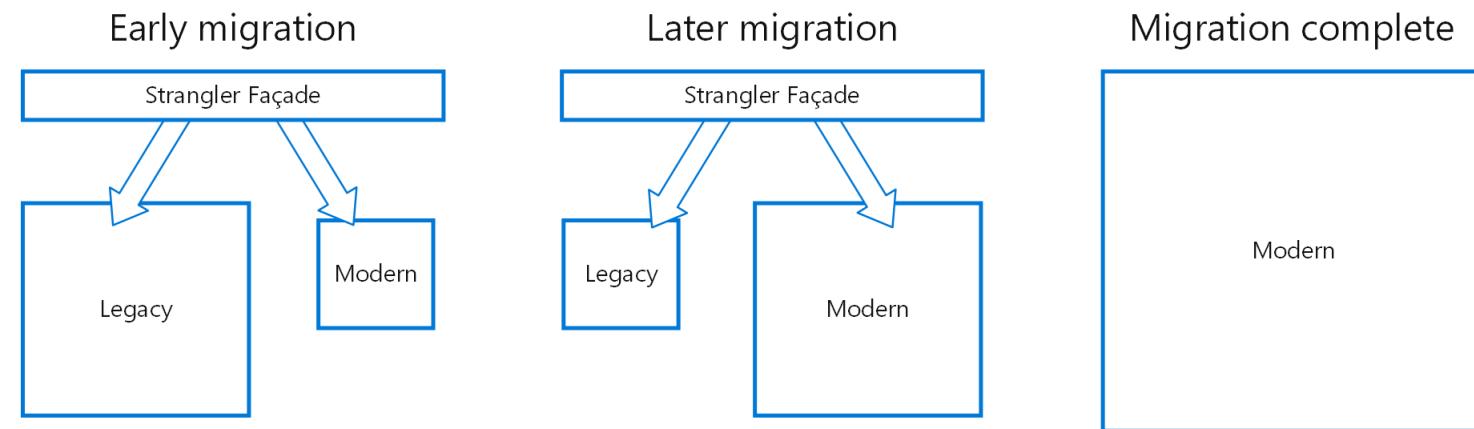


Evolution



“Strangler fig” pattern

Strangler Fig Pattern



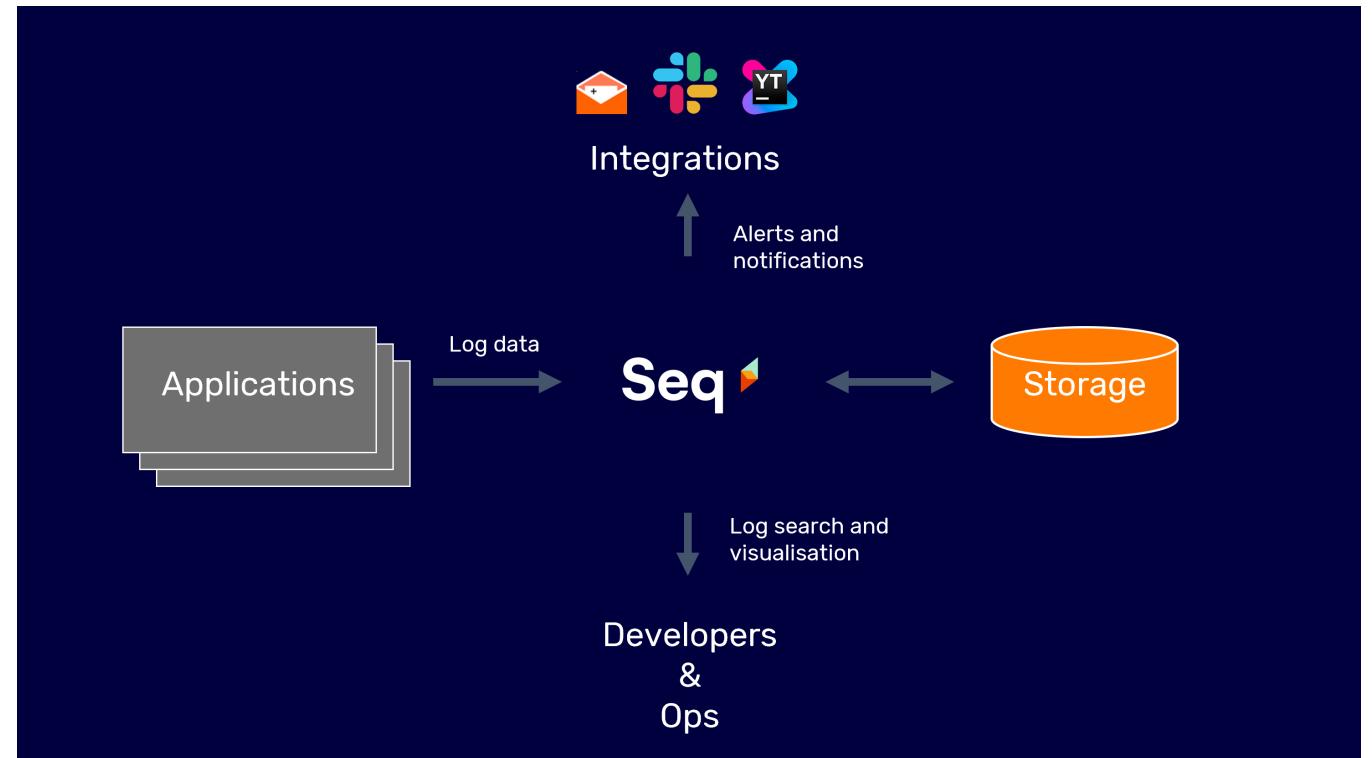
Source: <https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler-fig>

Messing Up Cross-Cutting Concerns

Problem #6

Example: Distributed Logging Is Hard

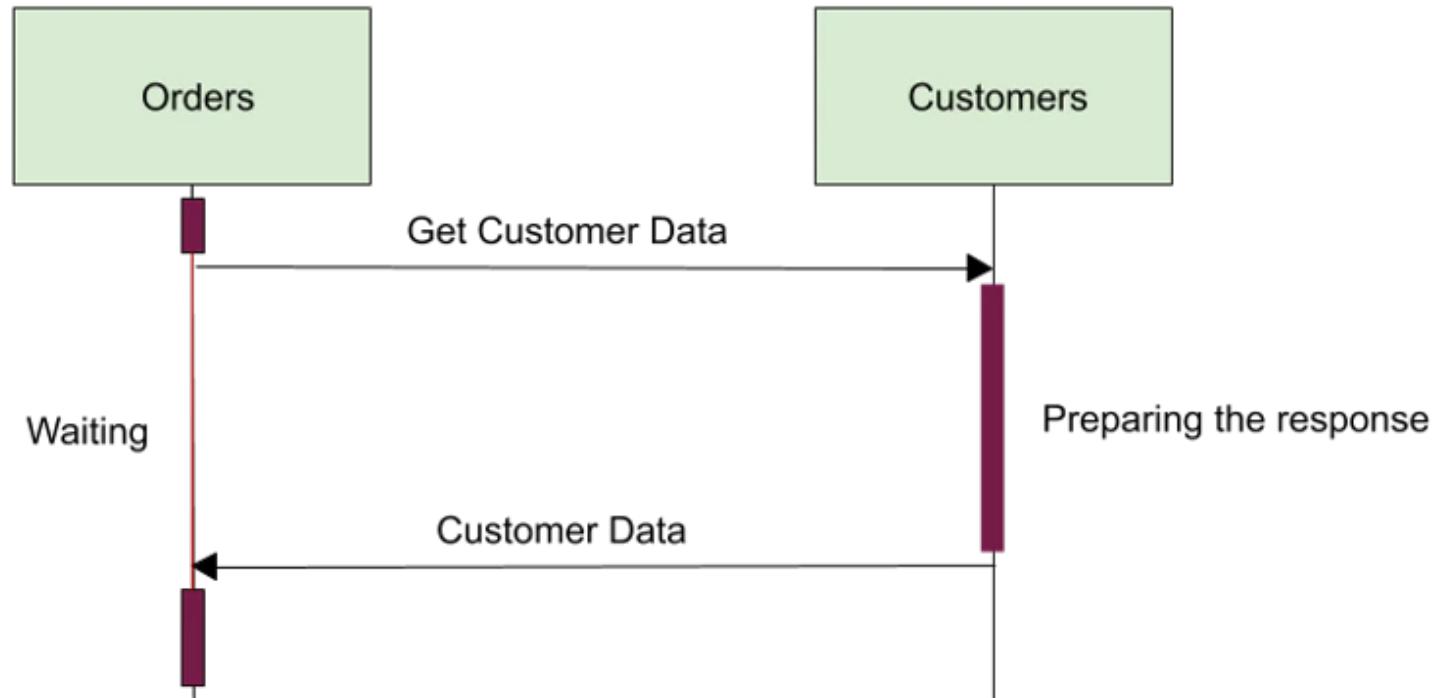
- Centralized without coupling
- Third party solutions like Seq
 - Seq: “Intelligent search, analysis, and alerting server built specifically for modern structured log data”
 - Supports .NET, Java, NodeJS, Ruby, Go, Python, more.
 - Inherently fault tolerant, embraces eventual consistency



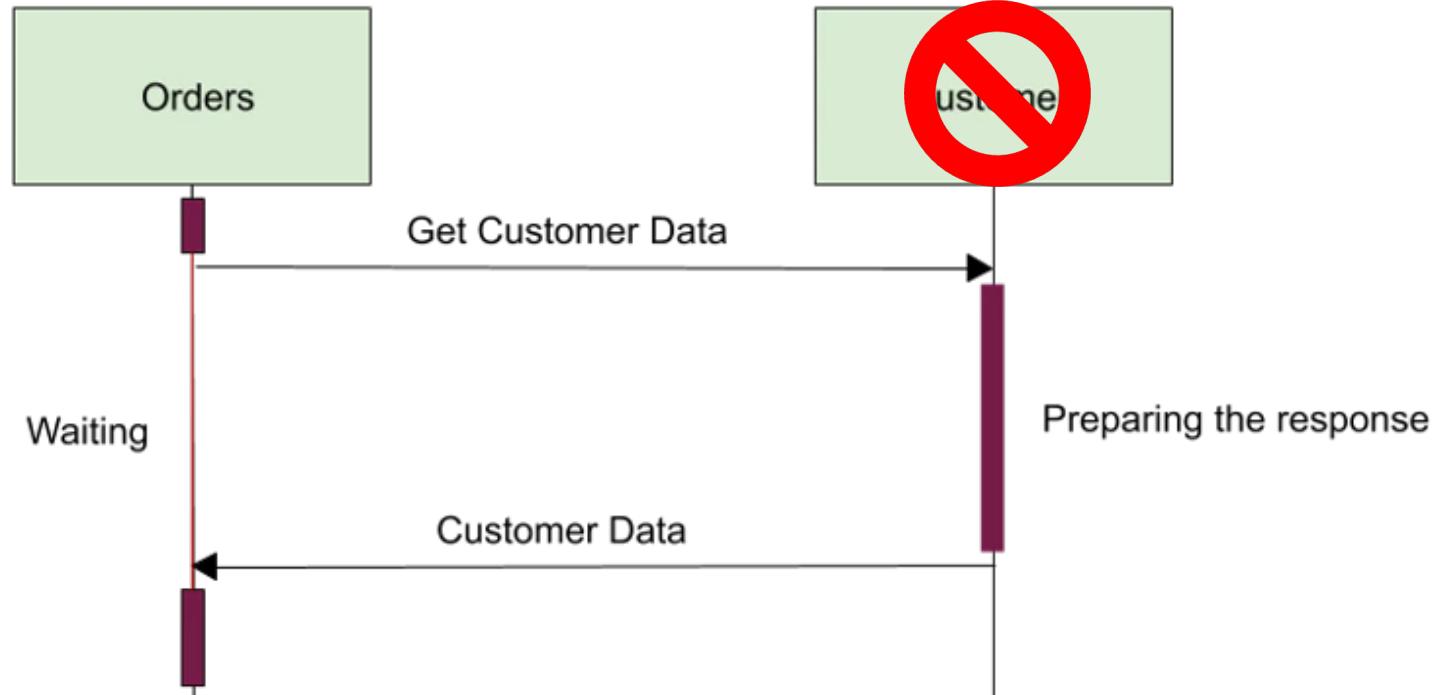
Over-Use of Synchronous Communication

Problem #7

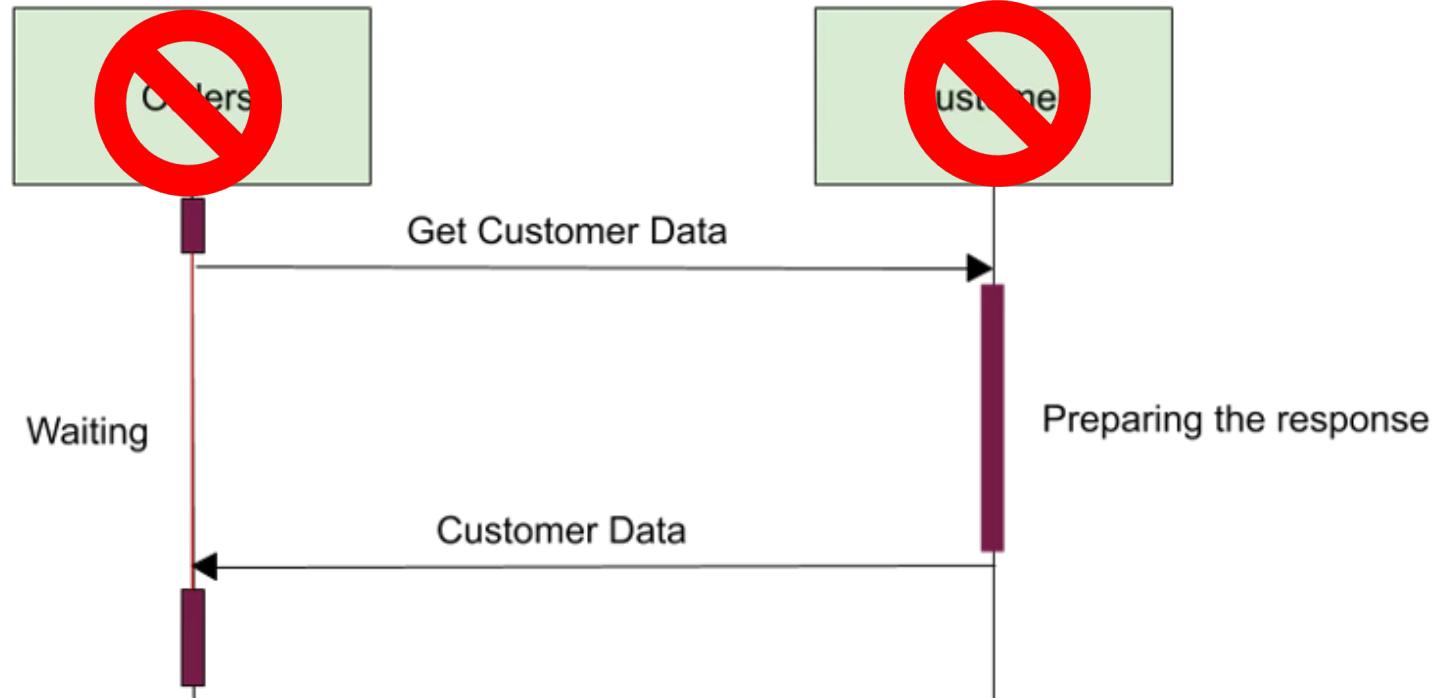




Source: <https://www.capitalone.com/tech/software-engineering/how-to-avoid-loose-coupled-microservices/>



Source: <https://www.capitalone.com/tech/software-engineering/how-to-avoid-loose-coupled-microservices/>



Source: <https://www.capitalone.com/tech/software-engineering/how-to-avoid-loose-coupled-microservices/>

Breaking Changes to Event Contracts

Problem #8



Rules for Event Changes

01

No new **required** fields, only optional fields (with documented default values).

02

Unrecognized fields are **ignored** (but forwarded)

03

Consumers of optional fields use **default** values when missing

04

When 1-3 cannot be satisfied, it's a **new event type**

Not Automating Build and Release

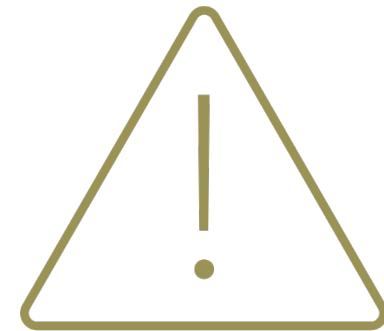
Problem #9



Prerequisite: Automated Build and Release



Time consuming



Prone to human error

Unencapsulated Services

Problem #10

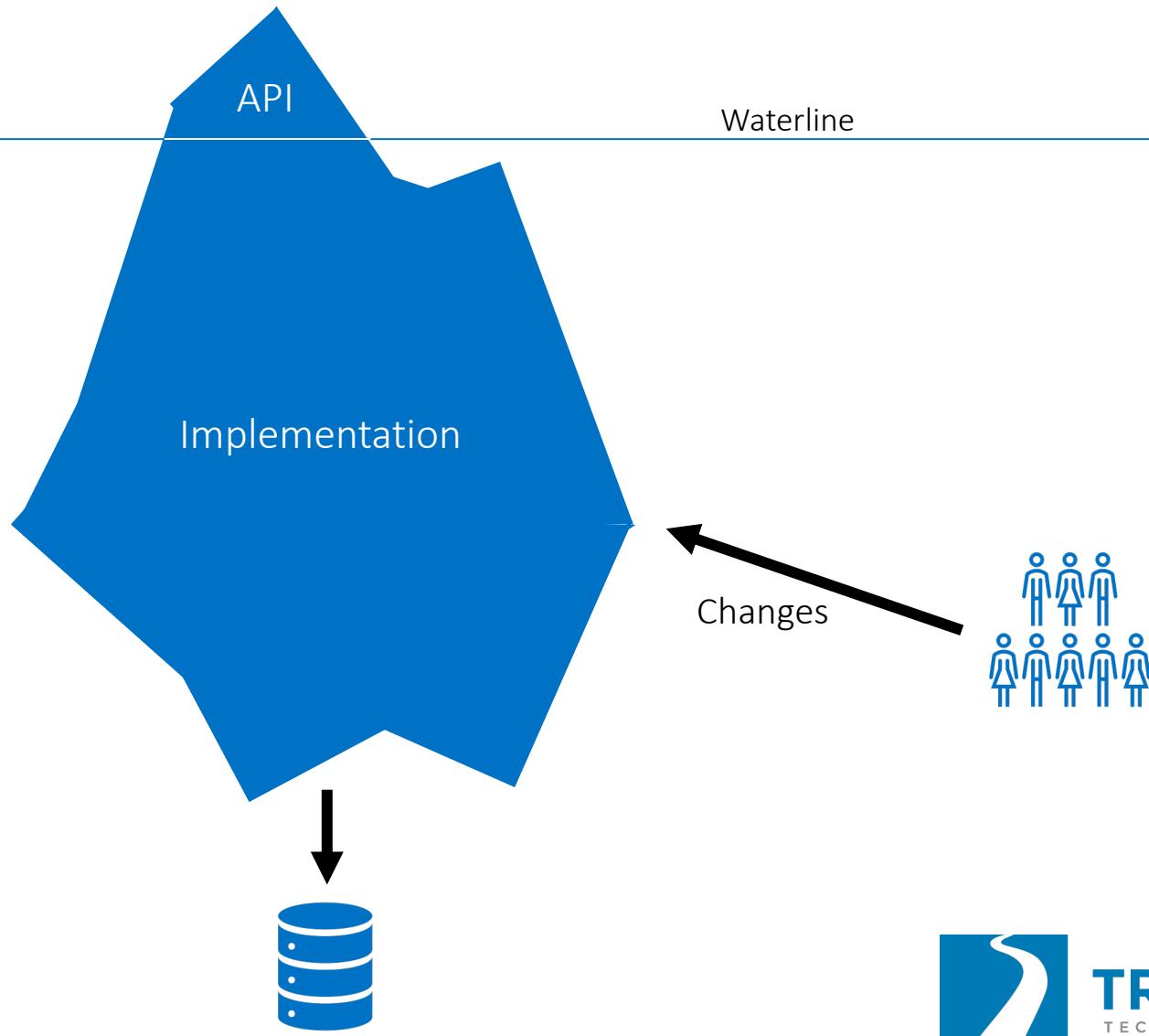


Iceberg Services

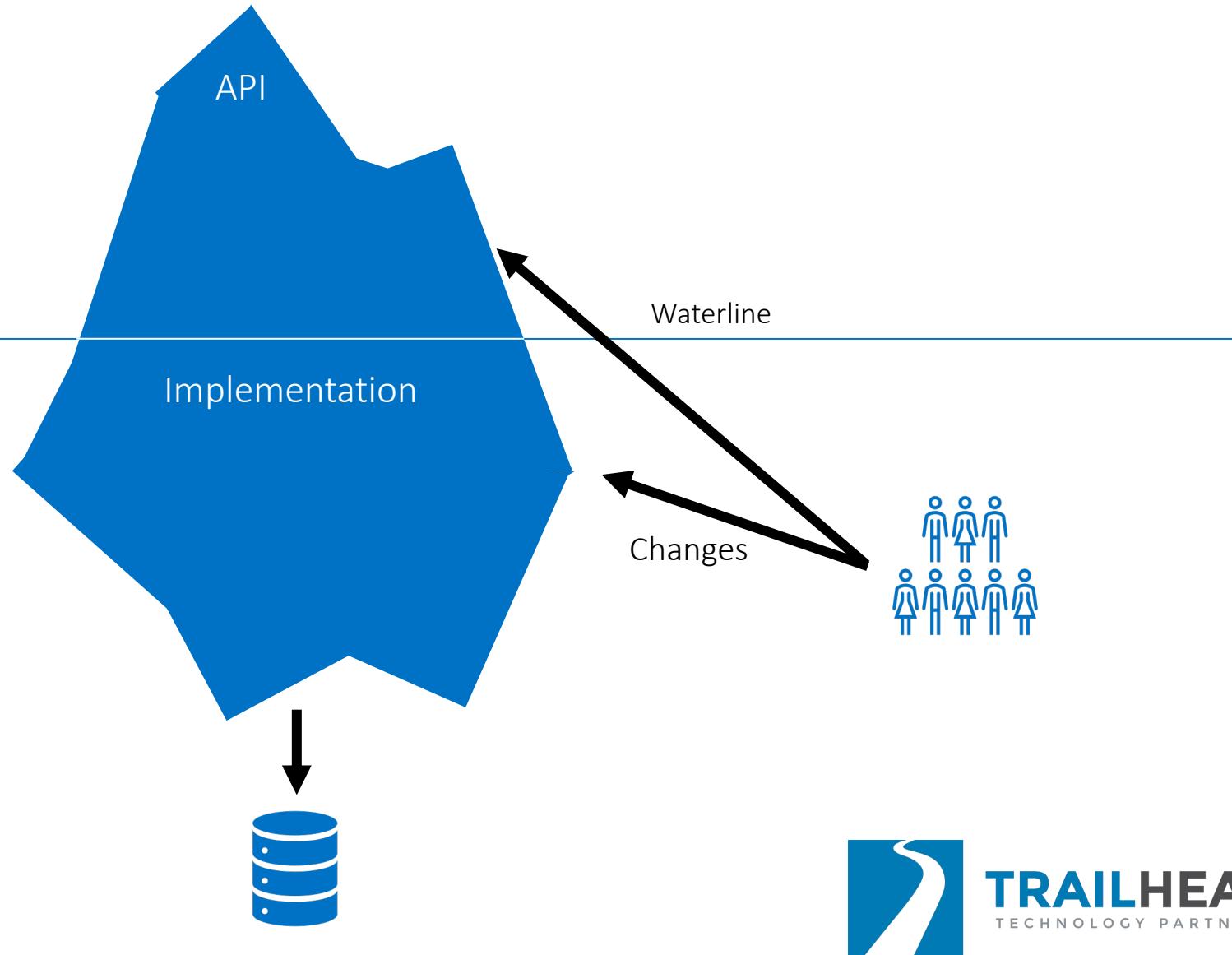
Services **ENCAPSULATE**
significant business logic

Small, stable API

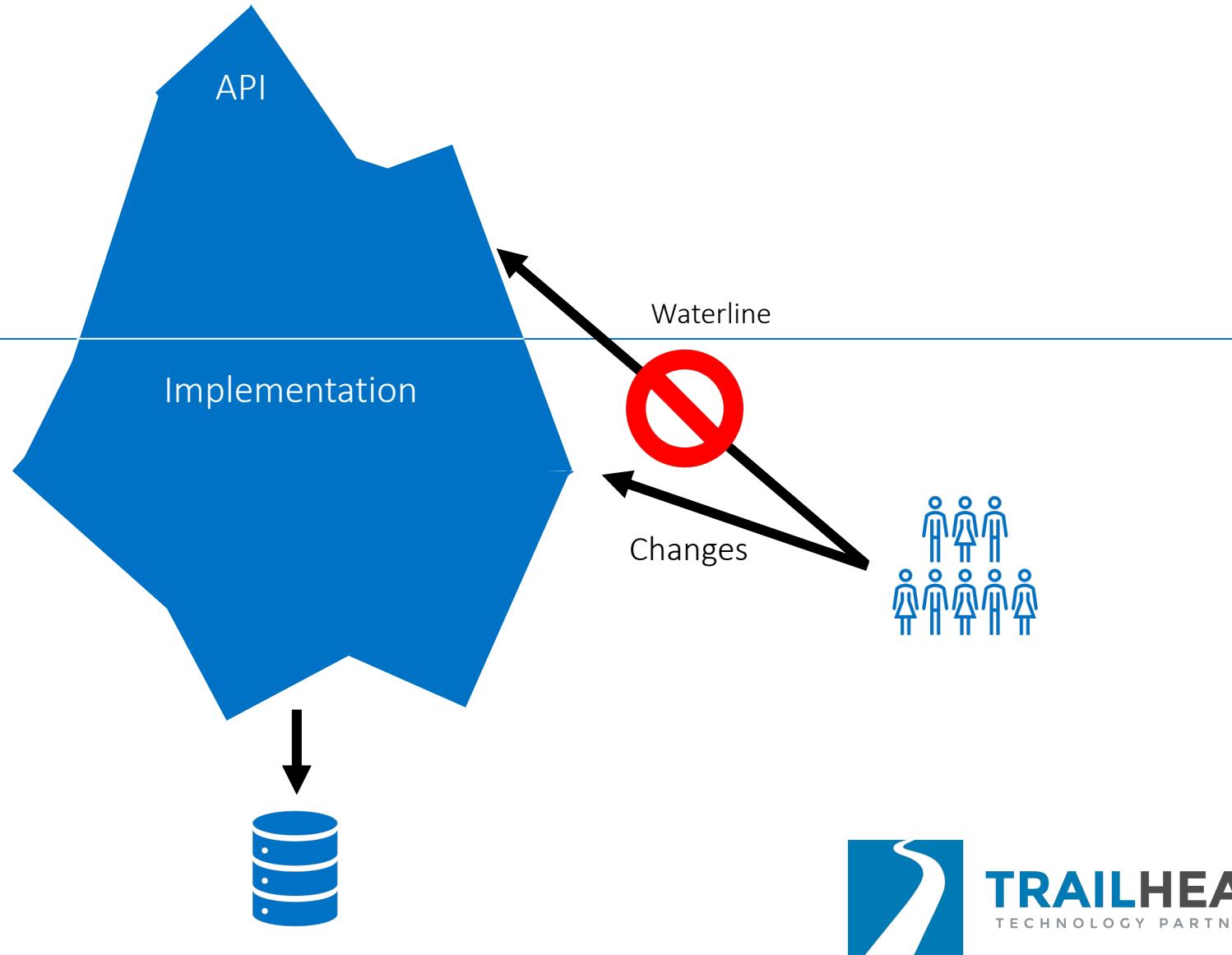
Large implementation



Iceberg Services



Iceberg Services



Mismatched Teams

BONUS PROBLEM



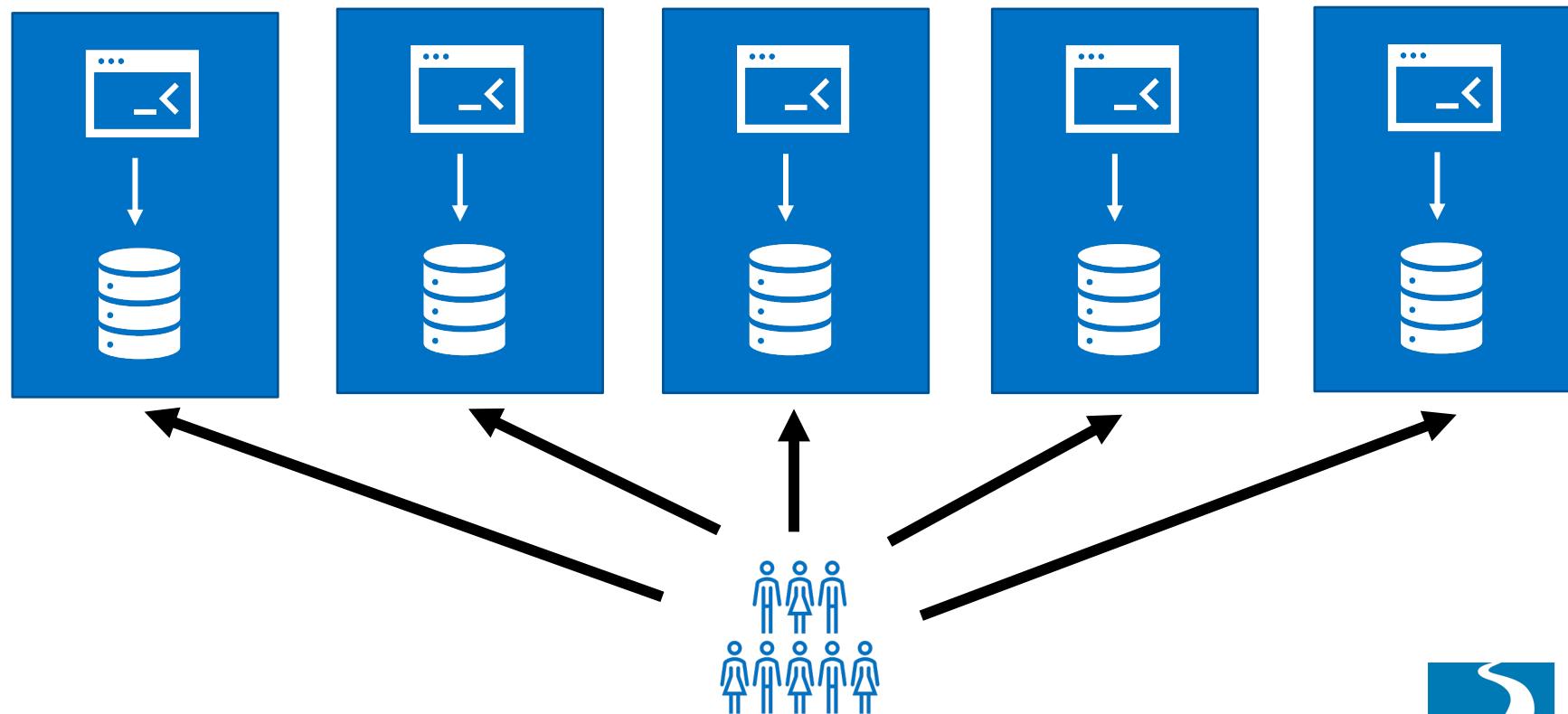
Conway's Law

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

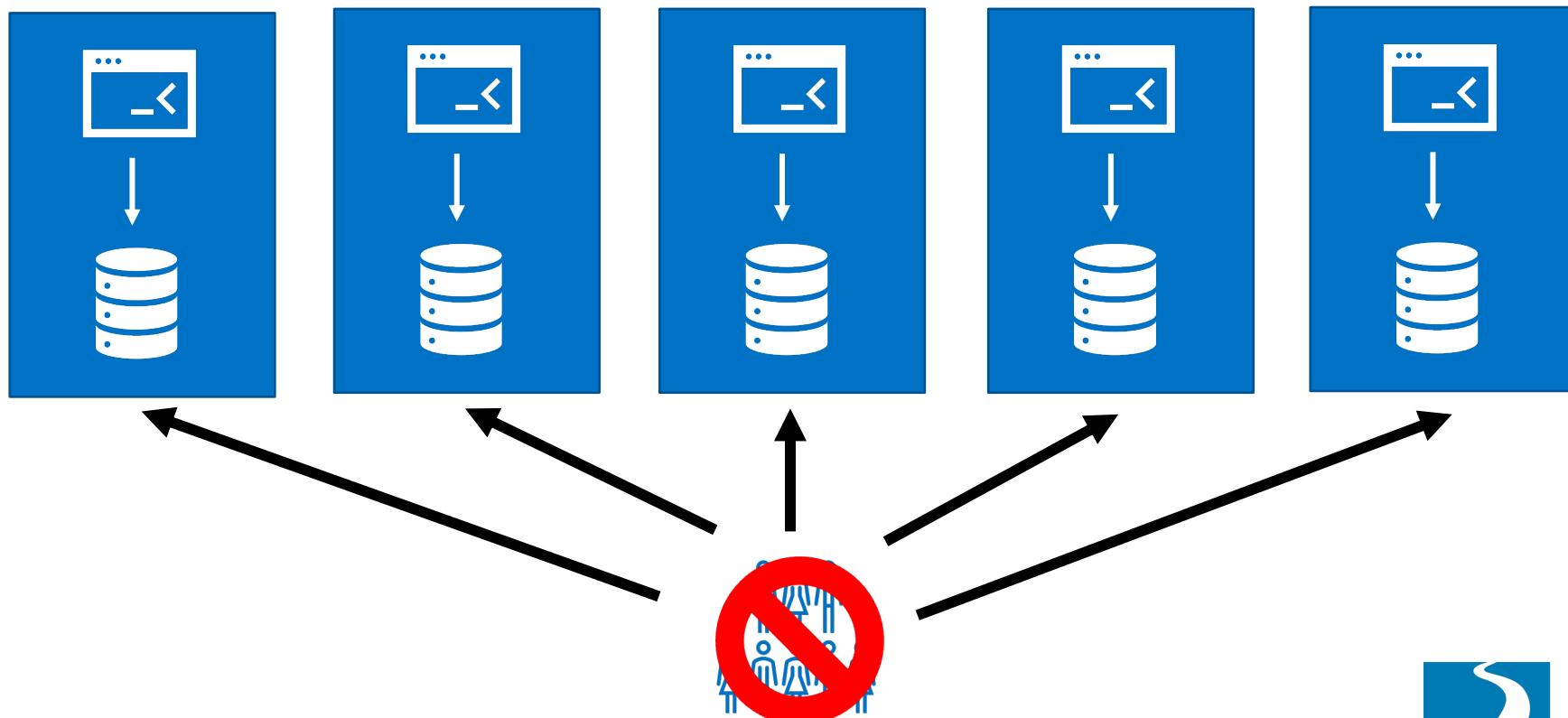
- Melvin E. Conway

IOW: if you have four groups working on a compiler, you'll get a 4-pass compiler.

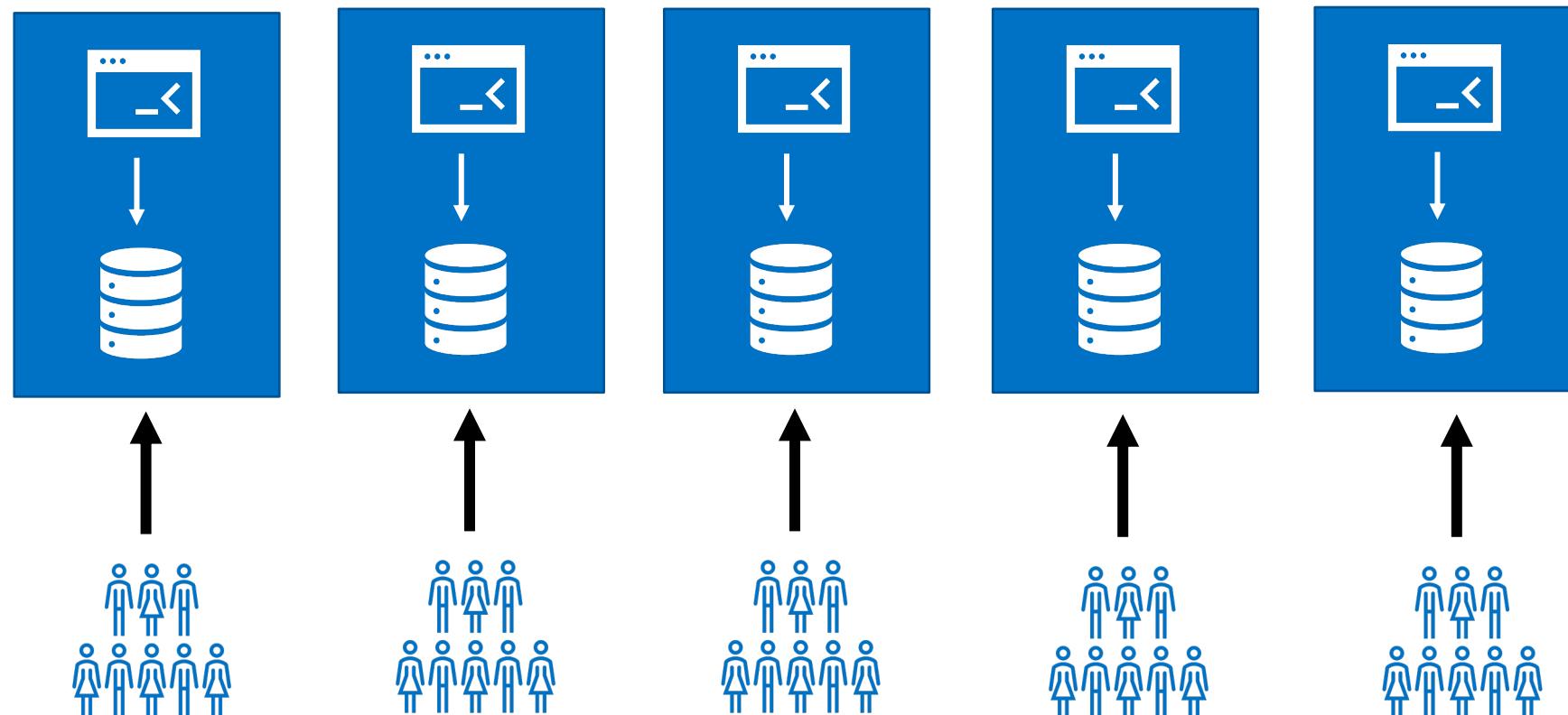
Single Team



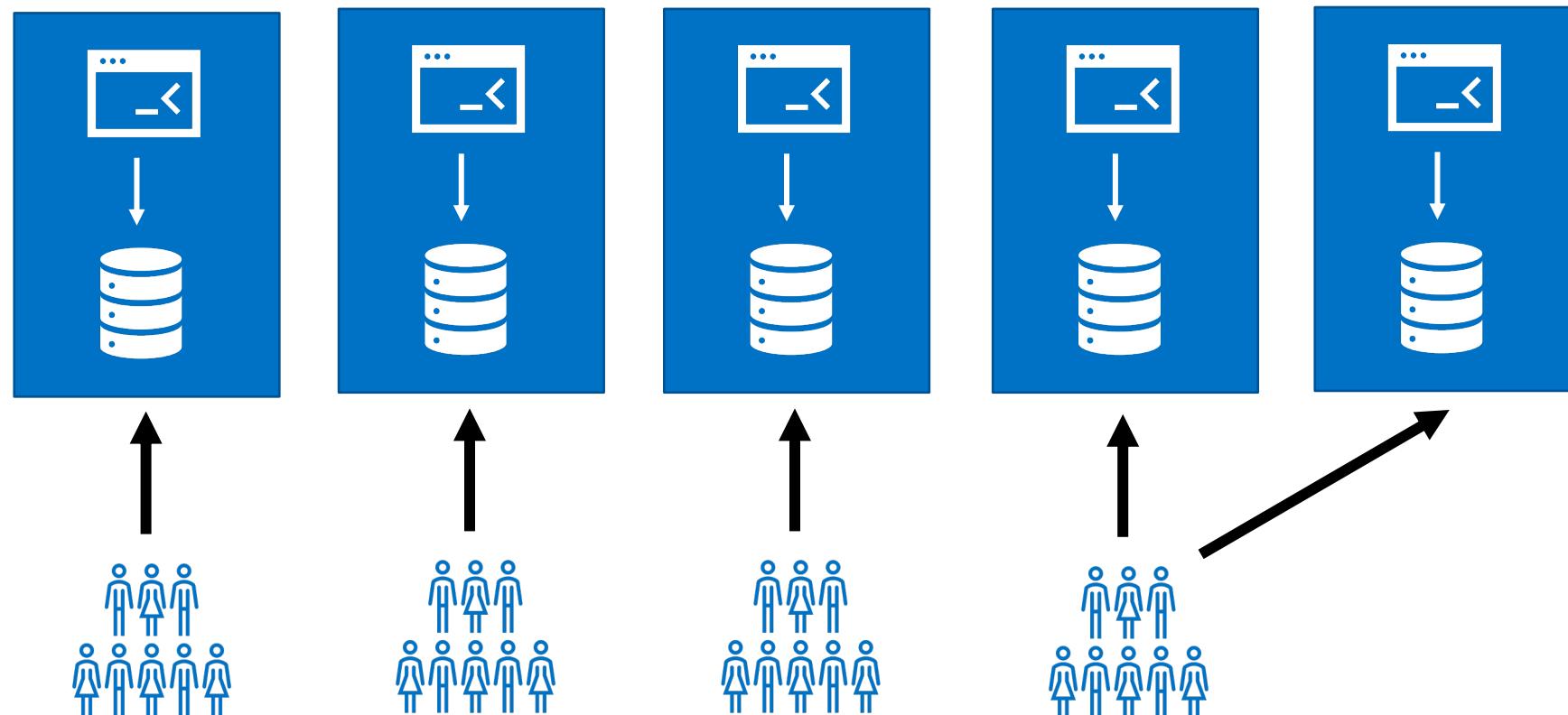
Single Team



Team Per Service



Team Per Service



Summing Up

1. For a lightweight application or single team, a monolithic system often suits better
2. For a complex, evolving application with clear domains and separate teams, microservices will be best
3. Don't try microservices without "a really good reason". Monoliths can be good!
4. Avoid pitfalls of the distributed monolith



Further Reading



Microservices Patterns

Chris Richardson



With examples in Java

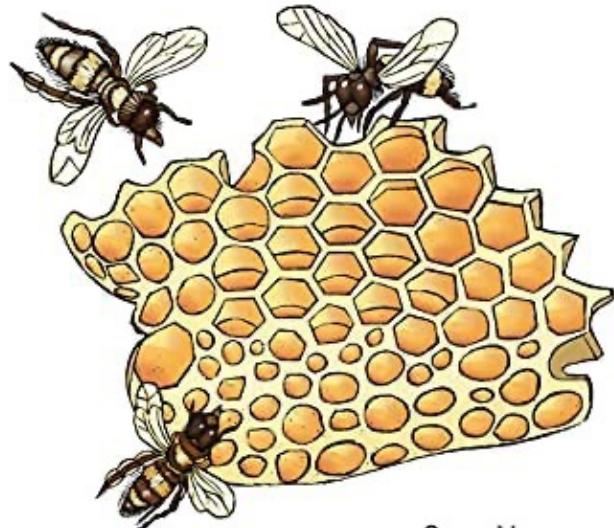


O'REILLY®

Second
Edition

Building Microservices

Designing Fine-Grained Systems

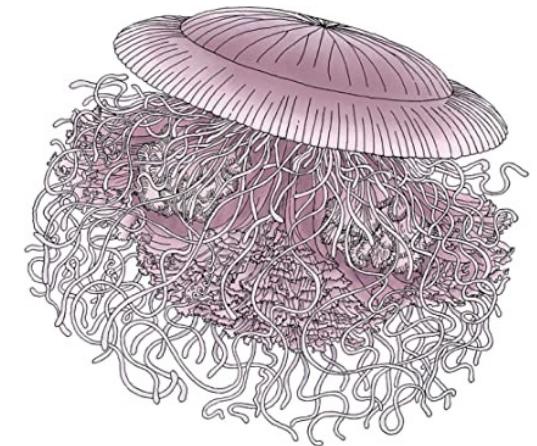


Sam Newman

O'REILLY®

Monolith to Microservices

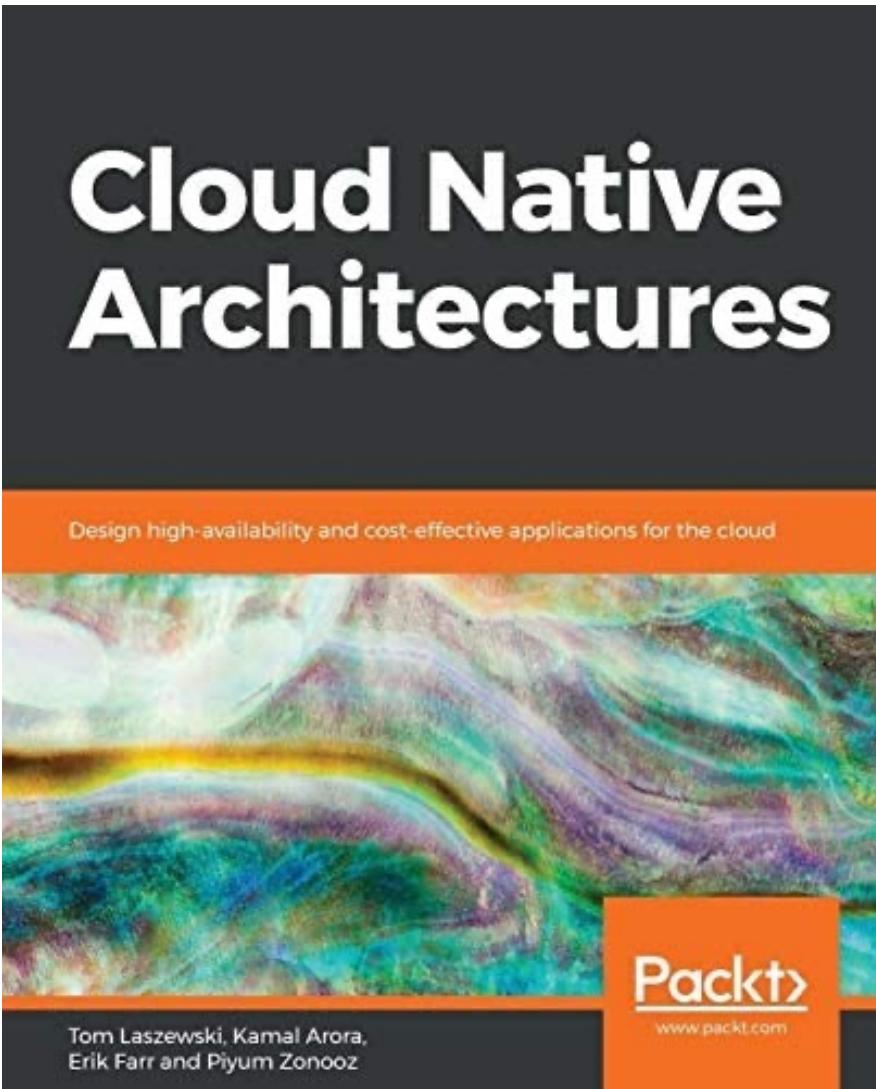
Evolutionary Patterns to Transform
Your Monolith



Sam Newman



TRAILHEAD
TECHNOLOGY PARTNERS



the Tao of Microservices

Richard Rodger

MANNING



Recap

Definitions

- Monolith
- Microservices
- Distributed Monolith

10 Most Common Mistakes

1 Bonus Mistake

Further Reading

Q&A