Don’t Build a Distributed Monolith

How to Avoid Doing Microservices Completely Wrong

Jonathan "J." Tower

Hi, I'm J.

Jonathan "J." Tower

Principal Consultant & Partner

Trailhead Technology Partners

TRAILHEAD
TECHNOLOGY PARTNERS

trailheadtechnology.com

- Microsoft MVP in .NET
- Organizer of Beer City Code

- jtower@trailheadtechnology.com
- trailheadtechnology.com/blog
- jtowermi

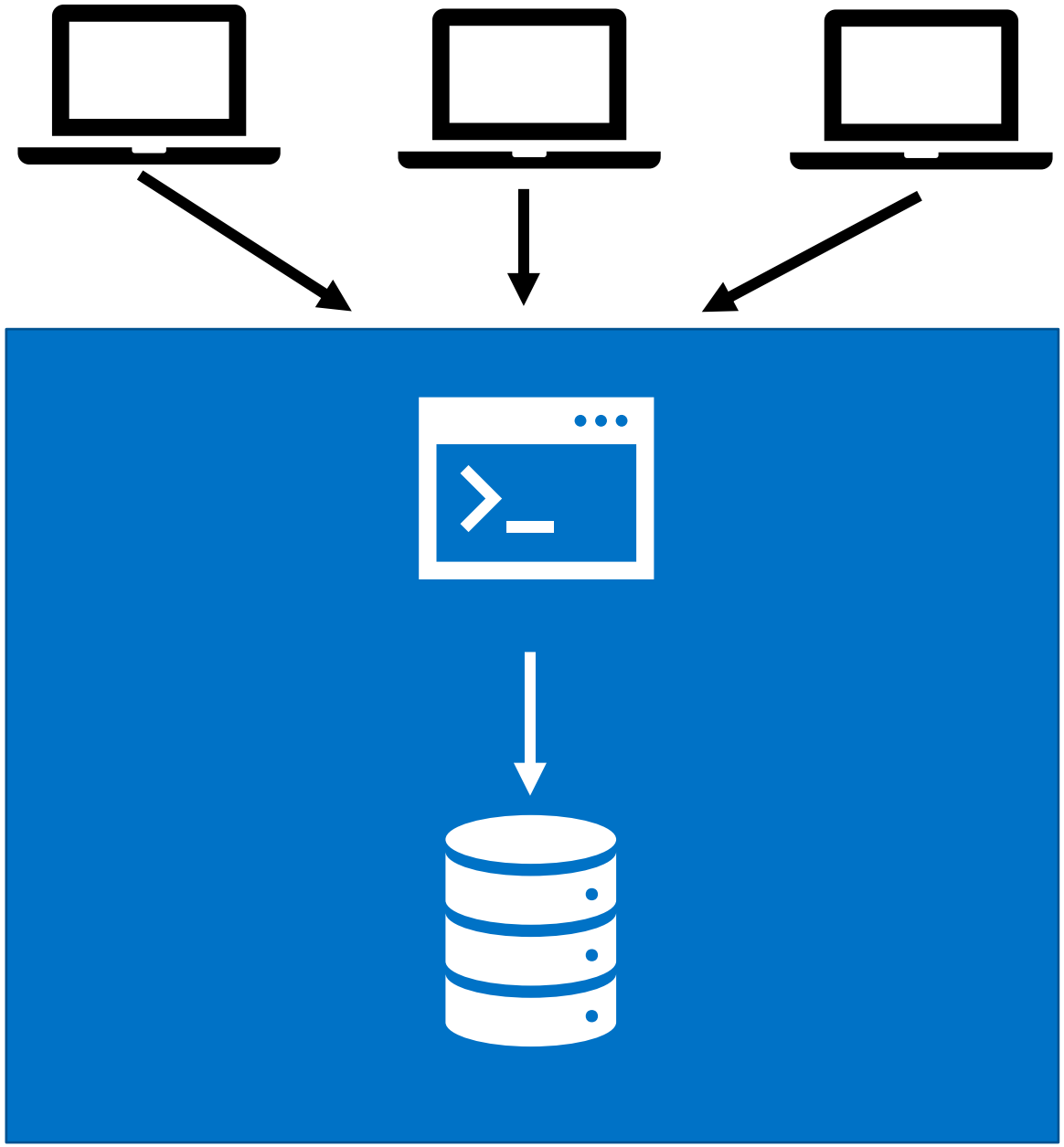
https://github.com/jonathantower/distributed-monolith

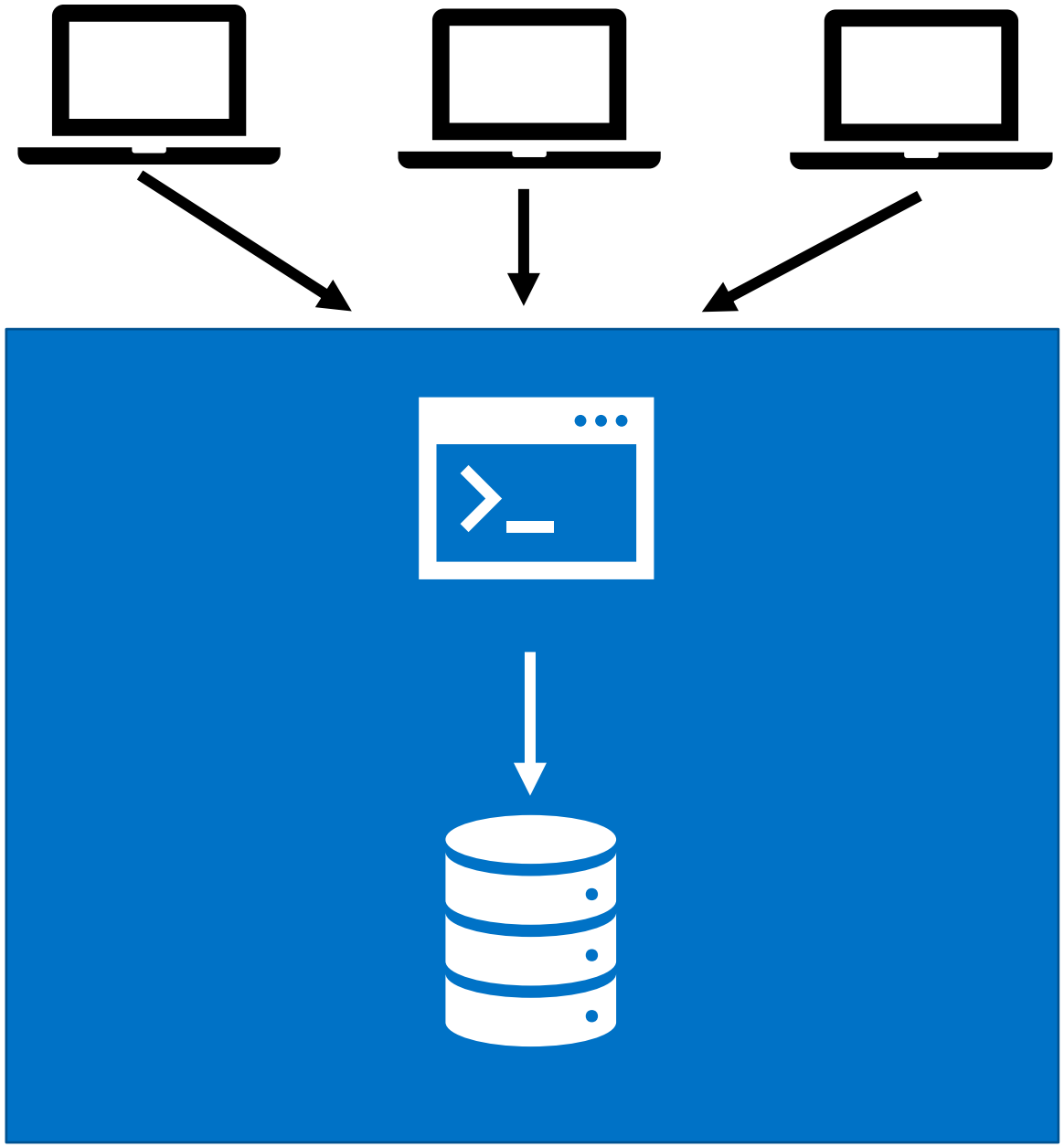
What We'll Talk About

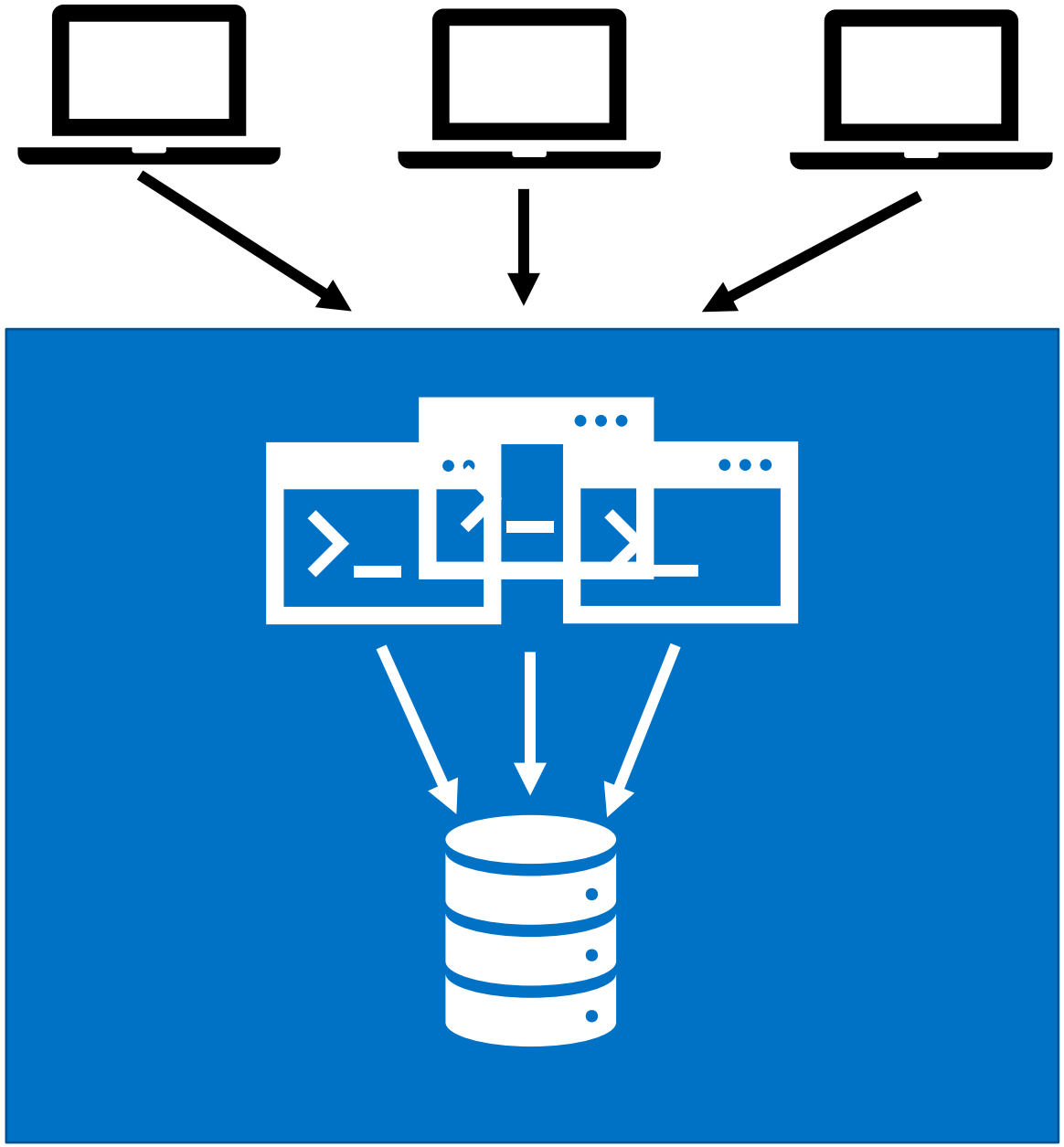
- Definitions
  - Monolith
  - Microservices
  - Distributed Monolith
- 13 Most Common Mistakes
- Further Reading
- Q&A

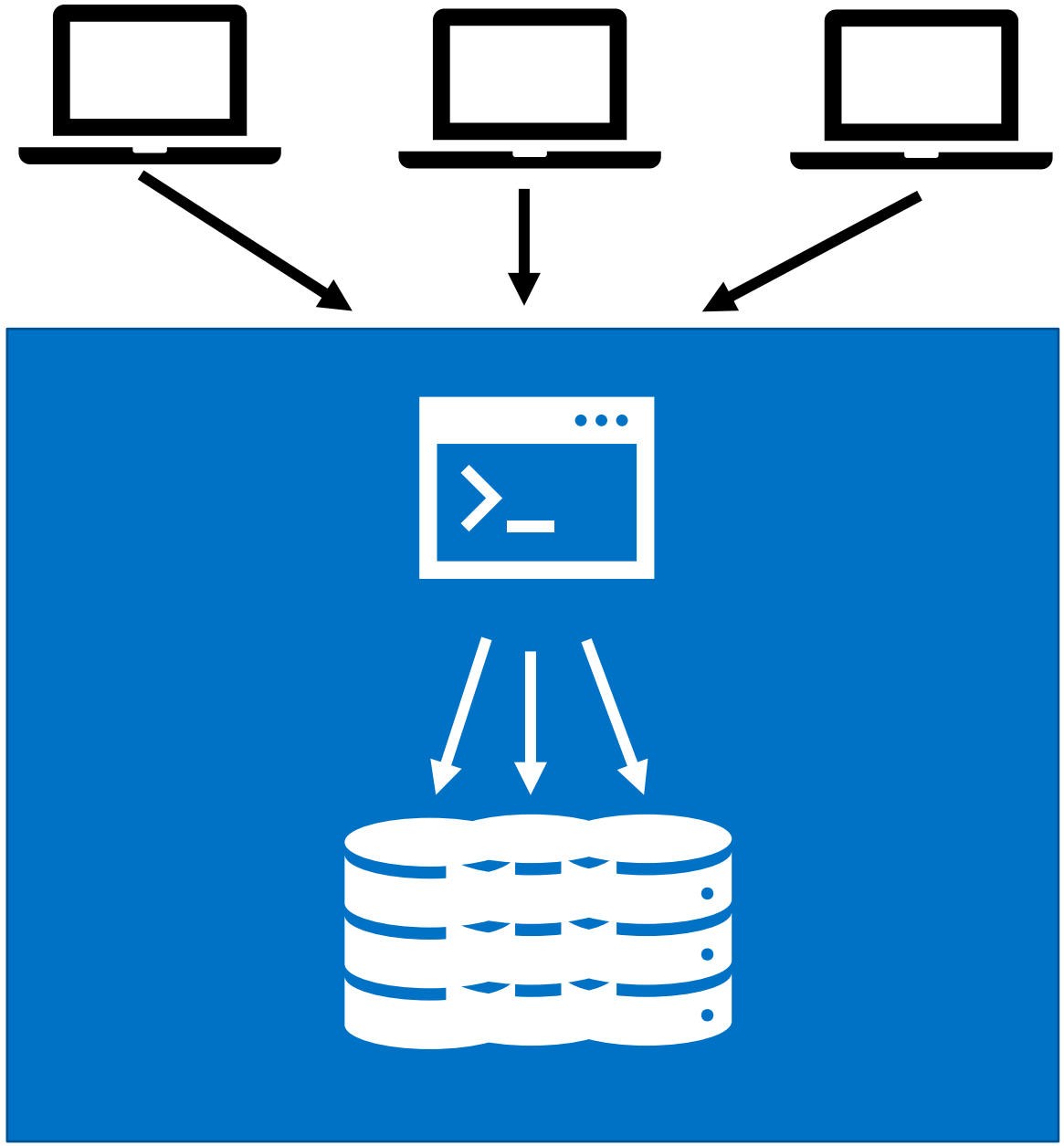
Some Definitions


Monolith

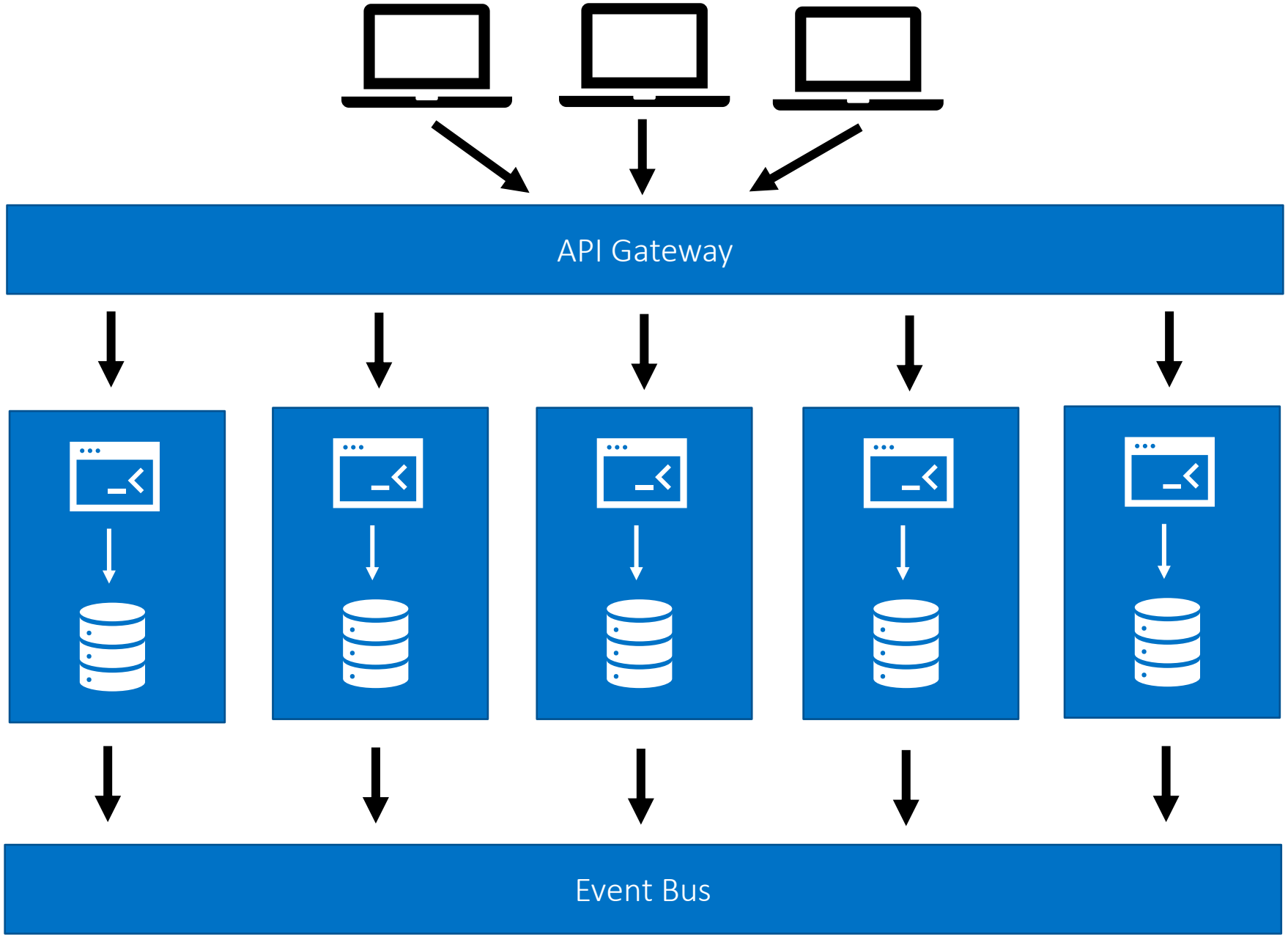






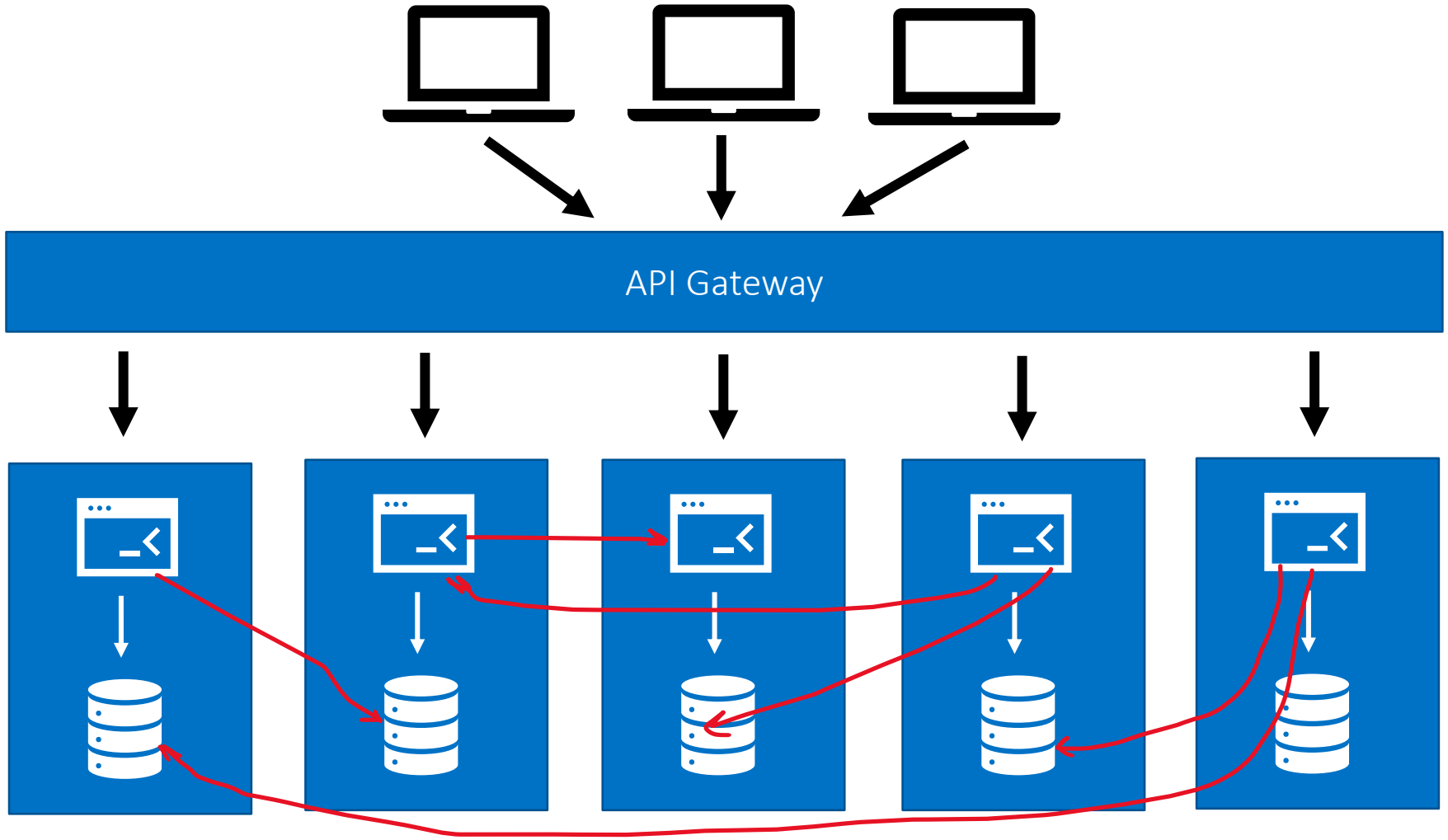


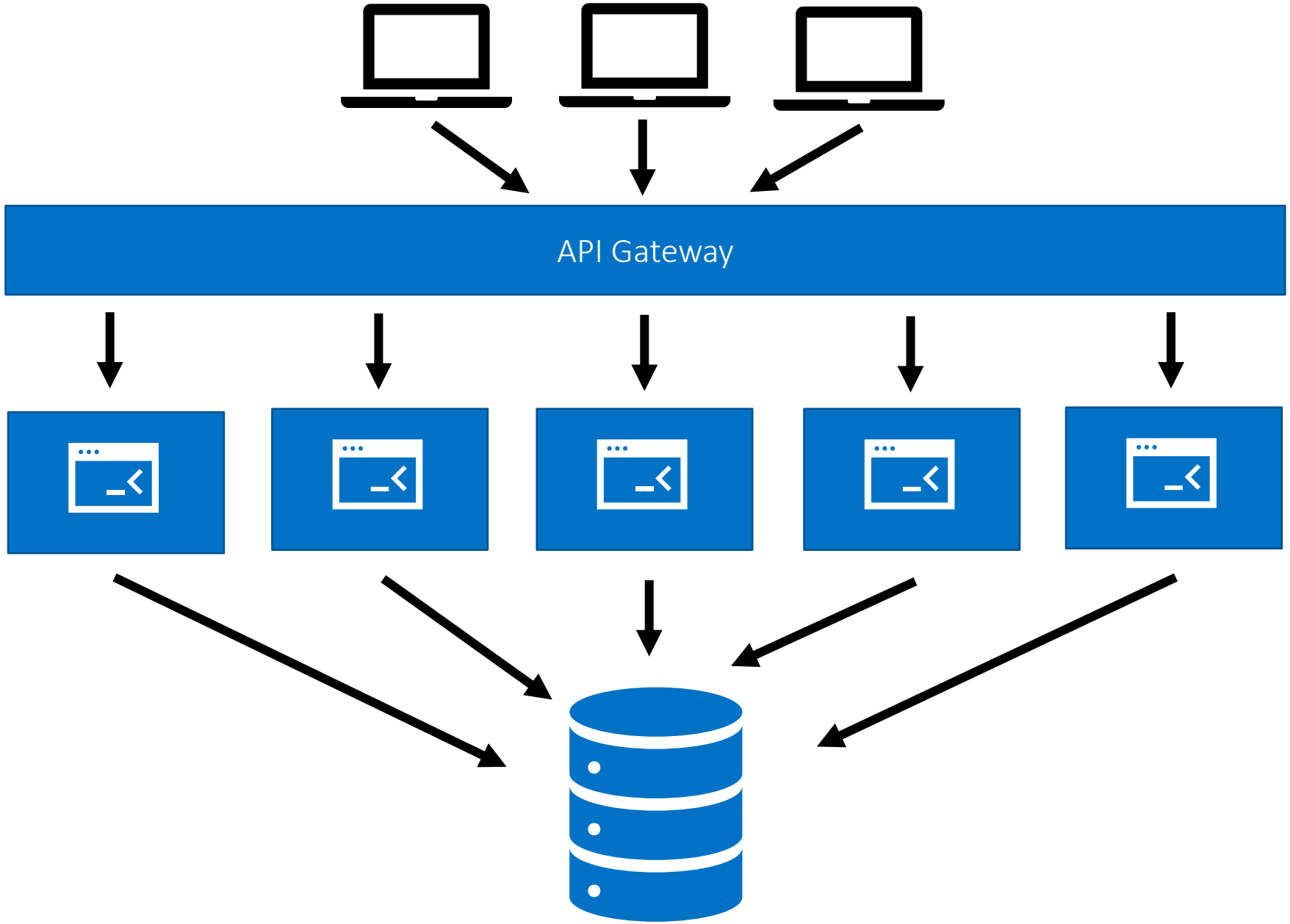
Microservices


API Gateway
Event Bus

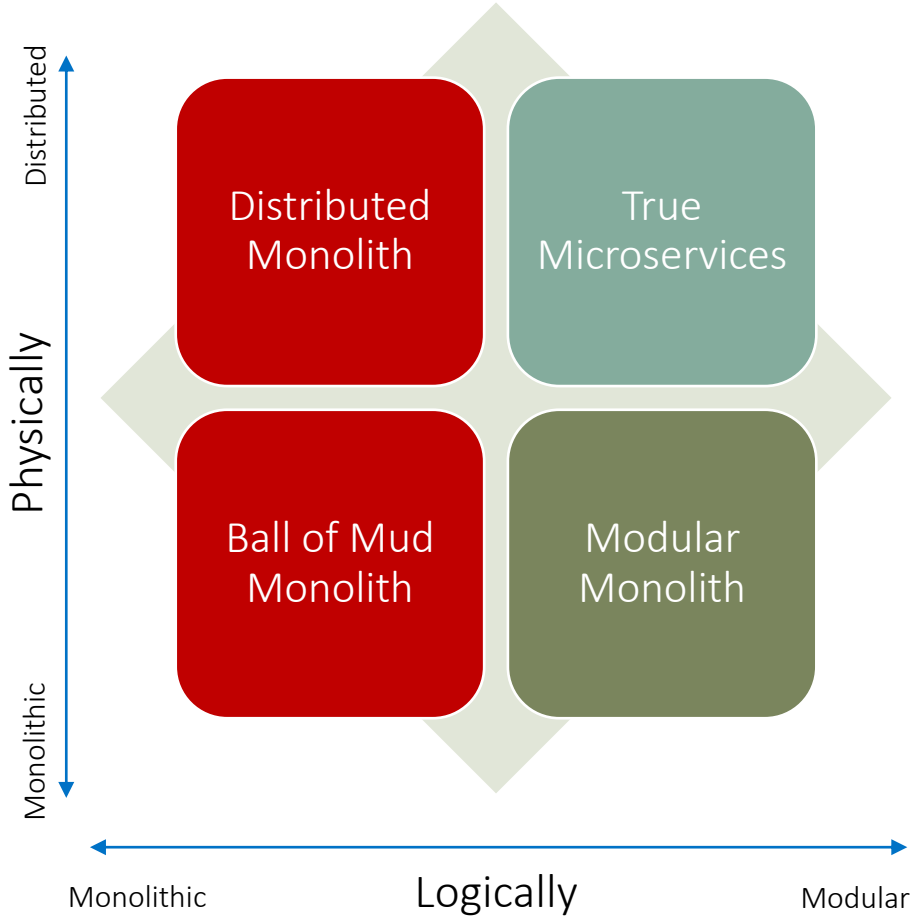
Distributed Monolith

Distributed Monolith


API Gateway


API Gateway

Good and Bad Monoliths



13 Most Common Mistakes

Avoid Creating a Distributed Monolith “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

Microservices are *hard*

Some Good Reasons to Microservice

A Need to Independently Deploy New Functionality with Zero Downtime

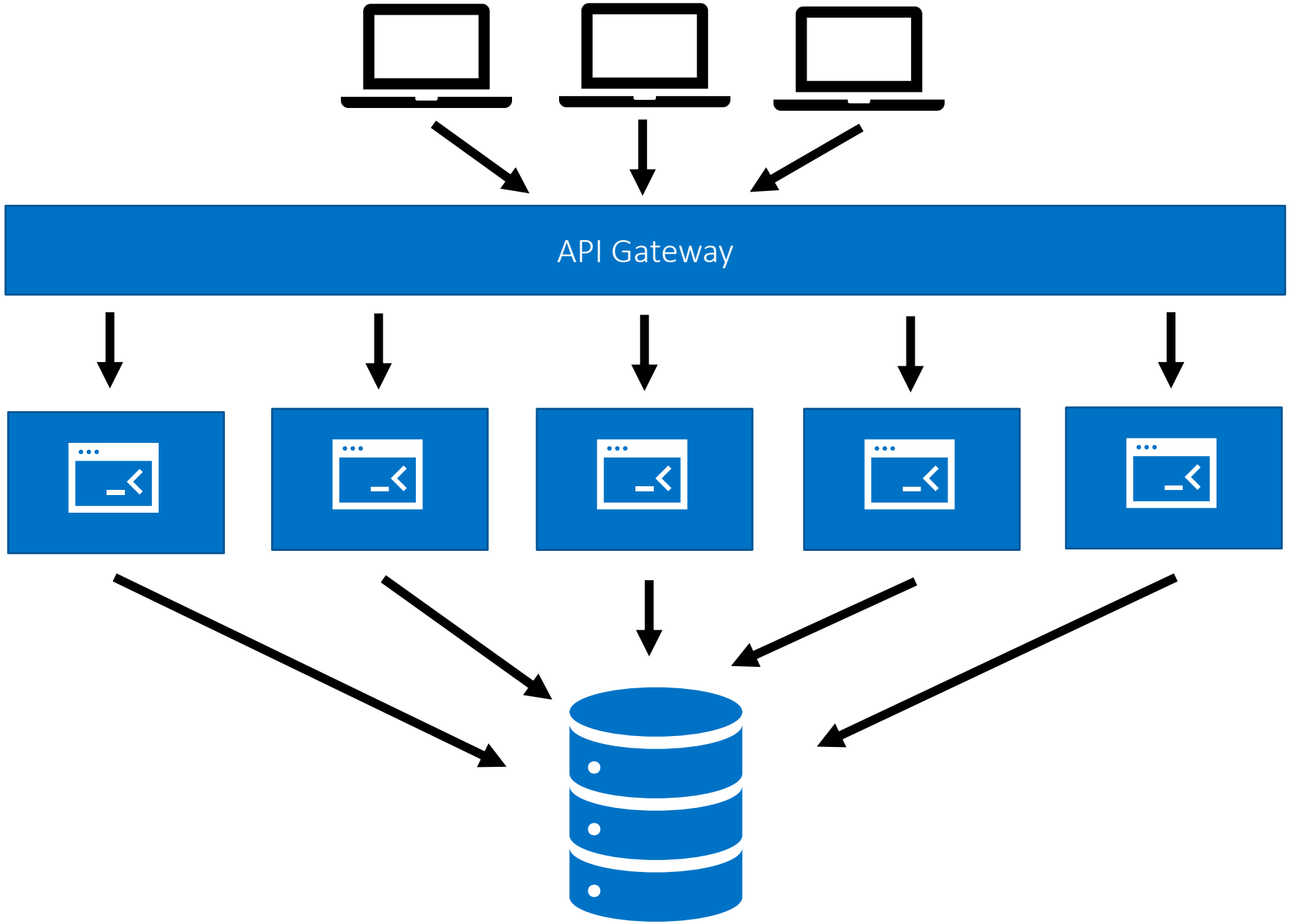
A Need to Isolate Specific Data and Data Processing Through Data Partitioning

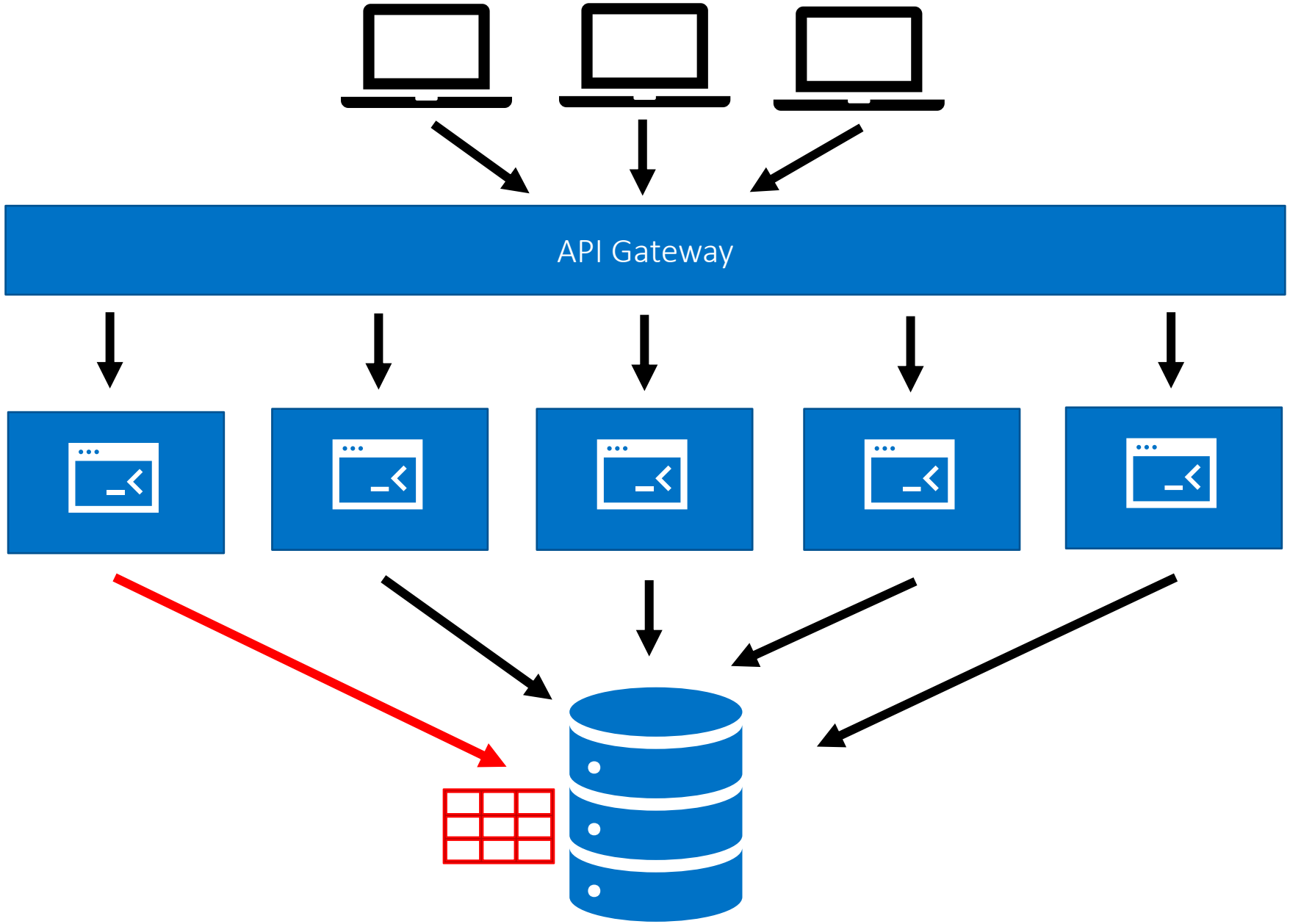
A Need to Enable a High Degree of Team Autonomy

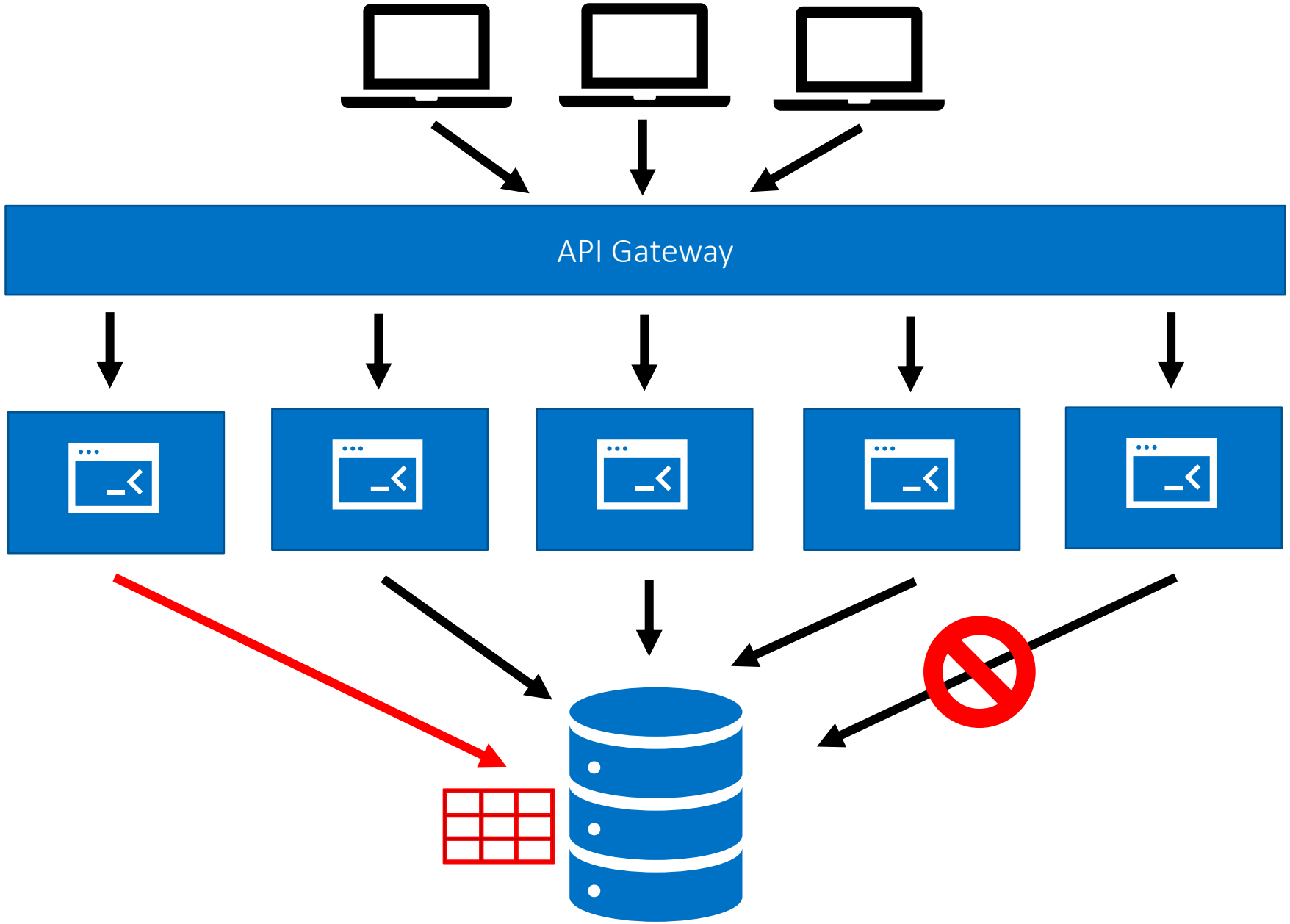
Source: https://blog.dreamfactory.com/when-to-use-microservices-sam-newman-and-martin-fowler-share-their-knowledge

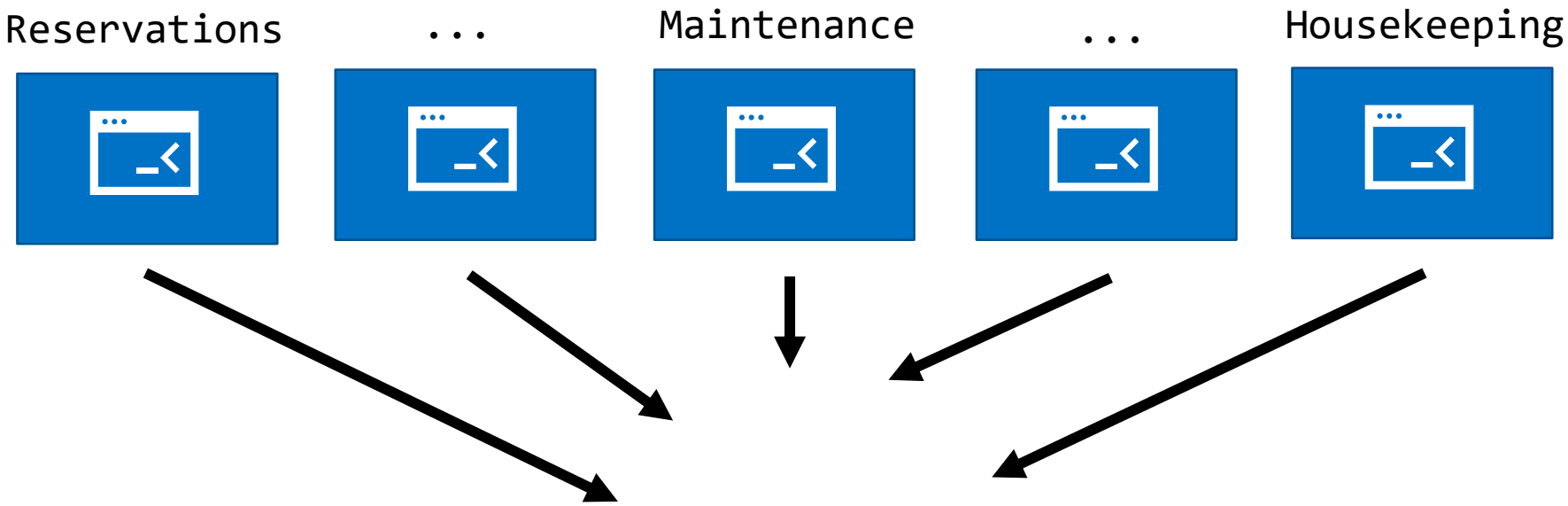
Shared Data Store or Models

Problem #2

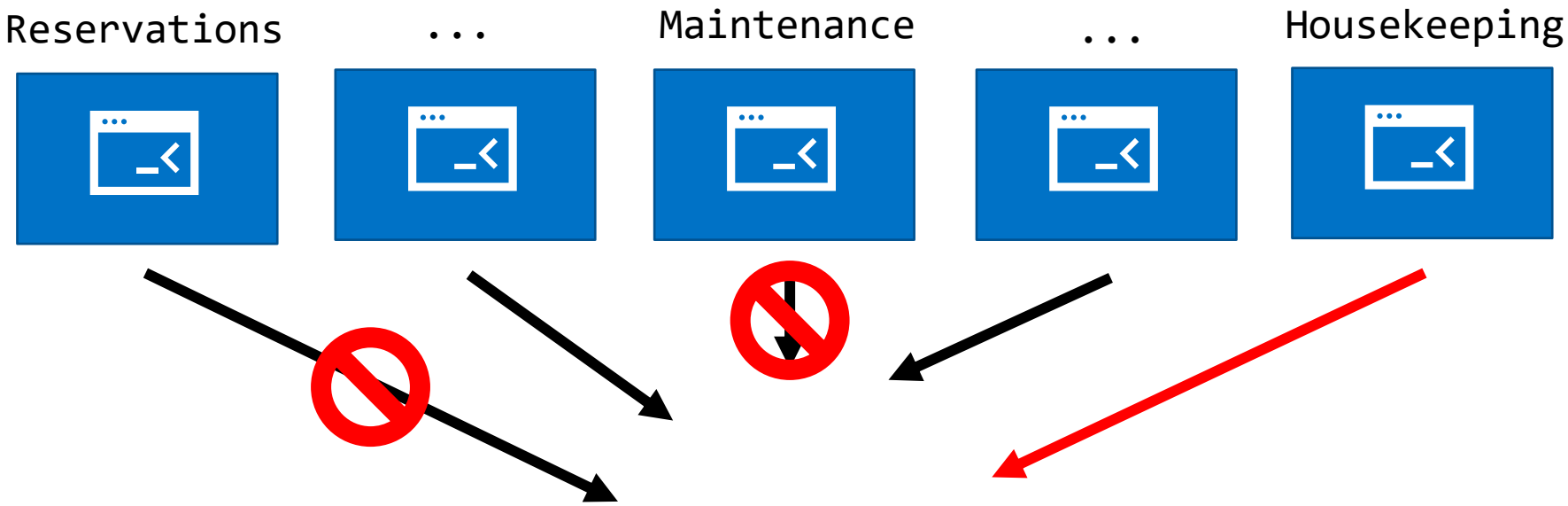

API Gateway


API Gateway

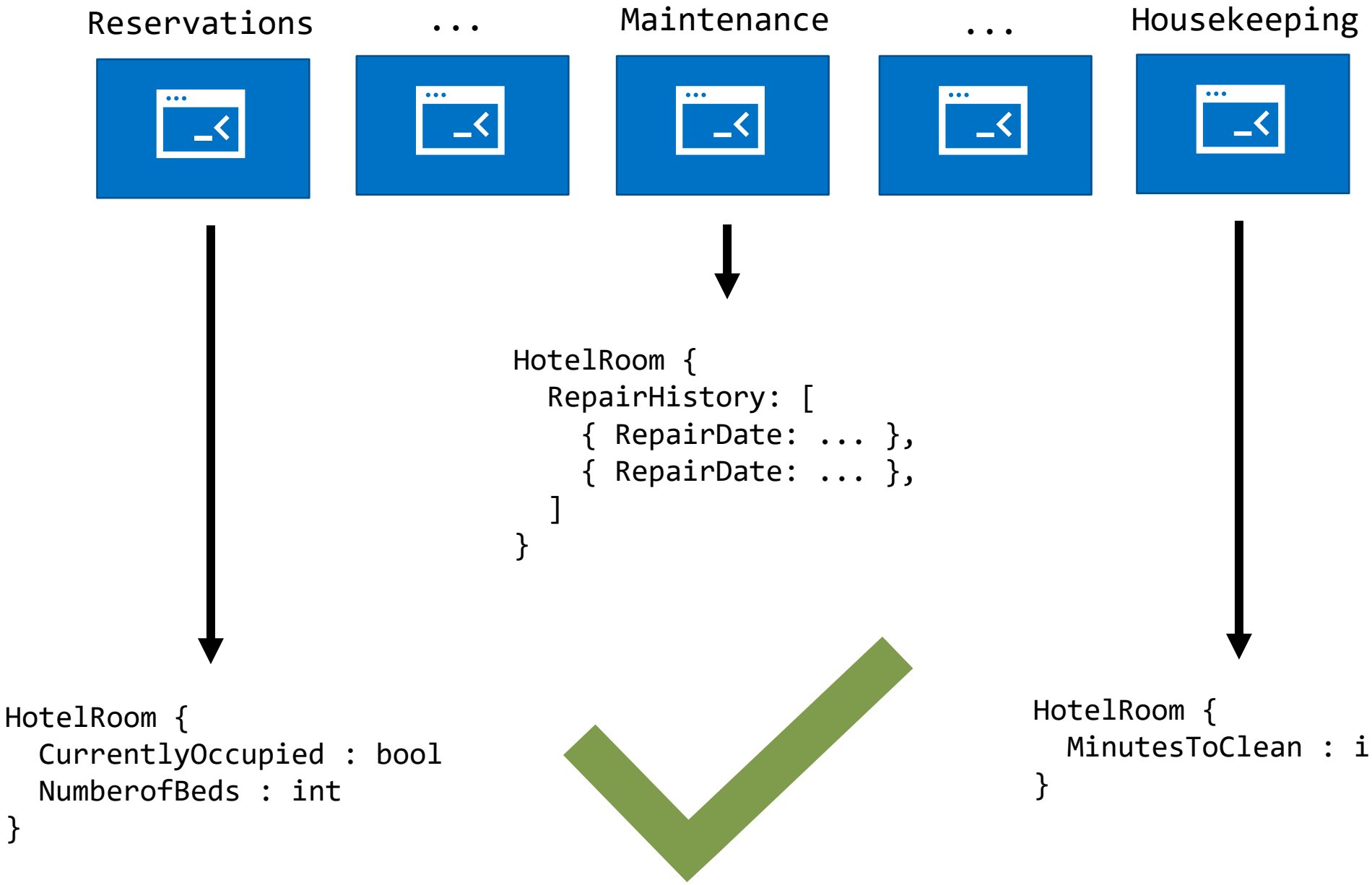

API Gateway




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




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


Reservations
...
Maintenance
...
Housekeeping
HotelRoom {
RepairHistory: [
{ RepairDate: ... },
{ RepairDate: ... },
]
}
HotelRoom {
CurrentlyOccupied : bool
NumberofBeds : int
}
HotelRoom {
MinutesToClean : int
}

Failing to Separate Sub-Domains

Problem #3

Domain Driven Development

Domain
 Subdomain
  Bounded Context

Domain Driven Development

Domain
 Subdomain
  Bounded Context

Smallest possible microservices without chatty communication between services

Starting from Scratch

Problem #4

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 Approaches: Monolith to Microservices

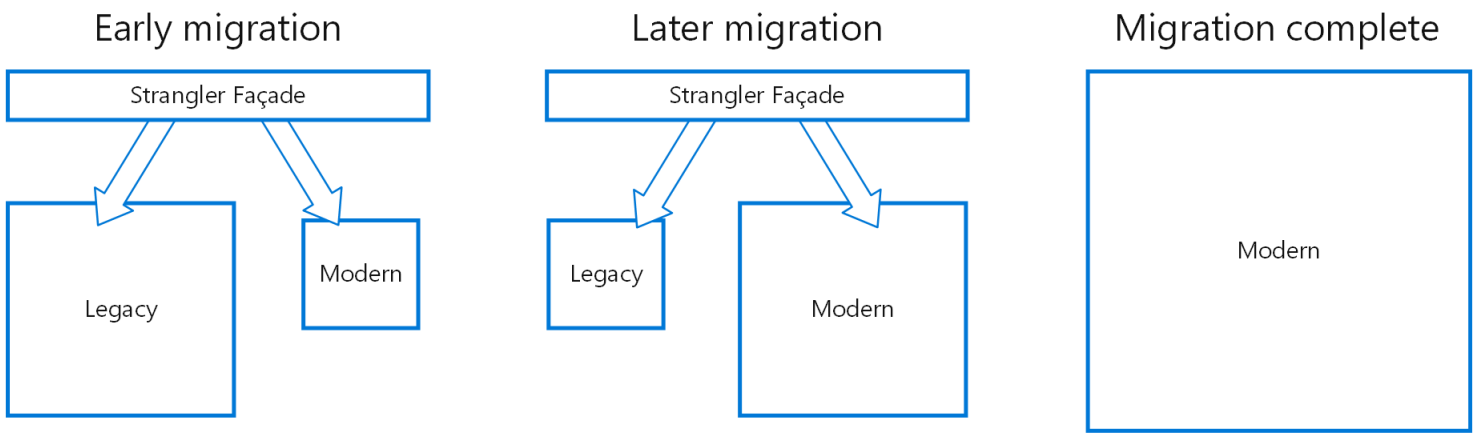
Big bang

Evolution

“Strangler fig” pattern

Source: https://blog.dreamfactory.com/when-to-use-microservices-sam-newman-and-martin-fowler-share-their-knowledge

Strangler Fig Pattern



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

Tight Coupling of Services

Problem #5

Easy To Tightly Couple Accidentally

Synchronous calls
(time coupling)

Shared message
definitions

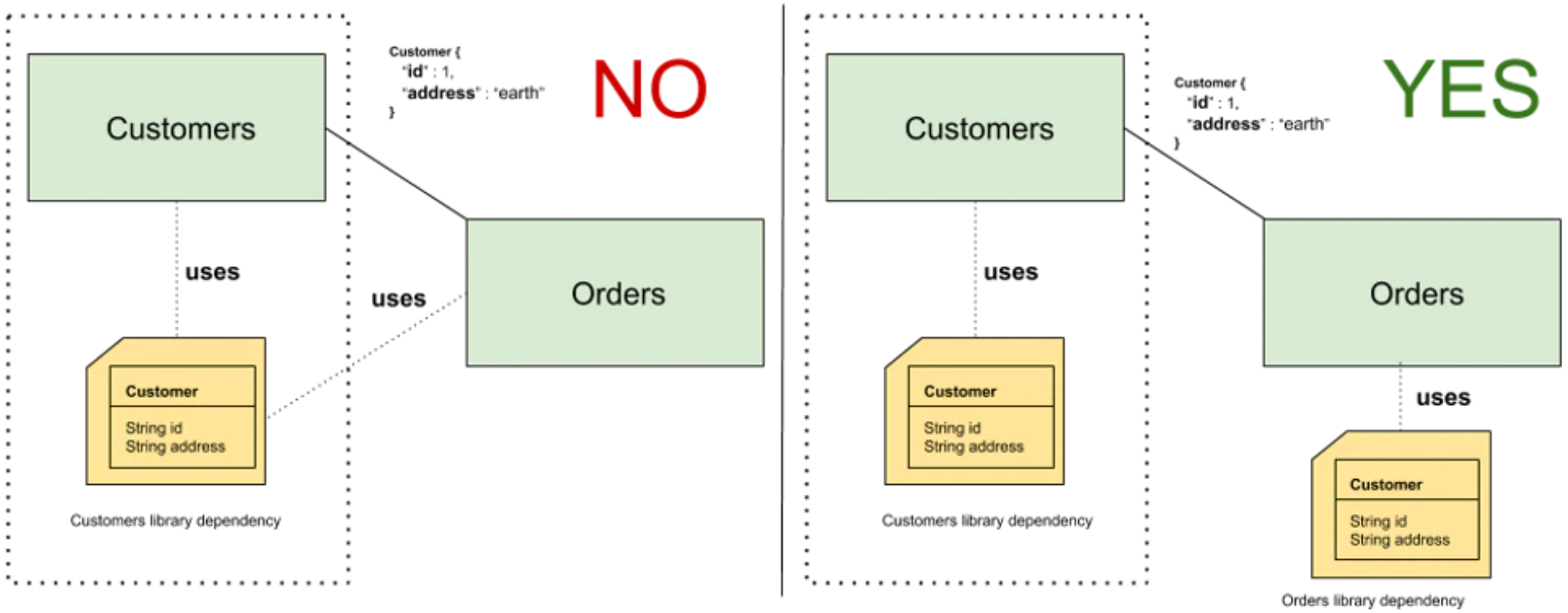
Shared object models
(DTOs, models)

Shared helper classes

Improper Code Sharing

Problem #6

Separate Copies of Dependencies

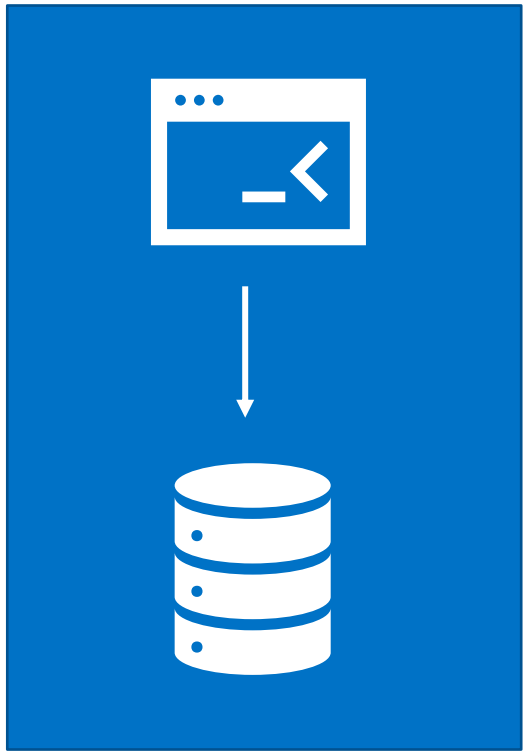


Source: https://www.capitalone.com/tech/software-engineering/how-to-avoid-loose-coupled-microservices/

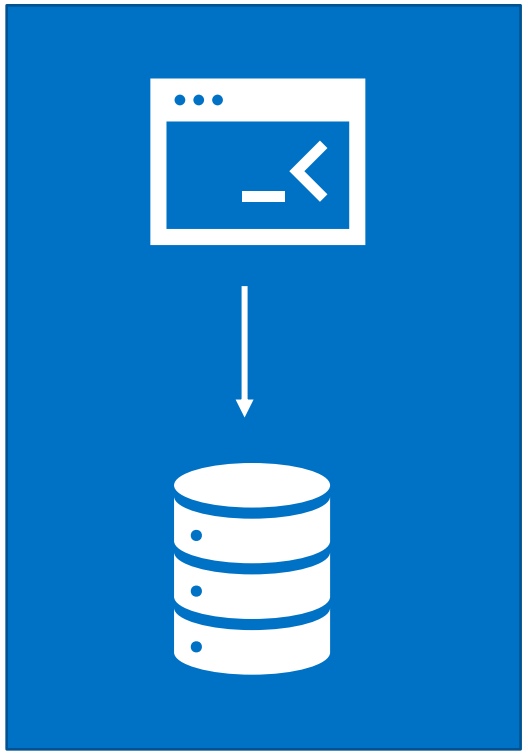
Microservices That Are Too Small

Problem #7

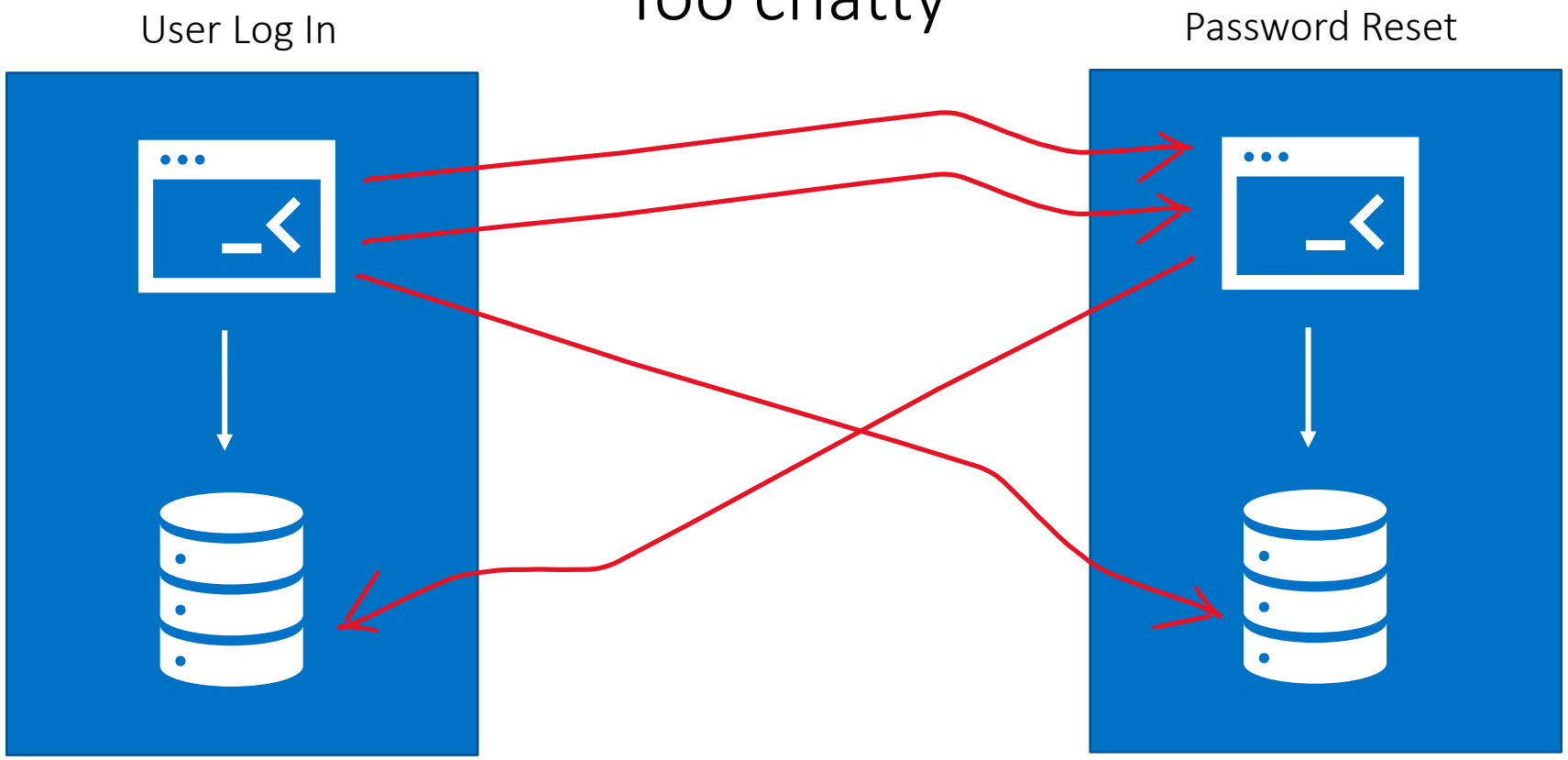
User Log In



Password Reset



Too chatty

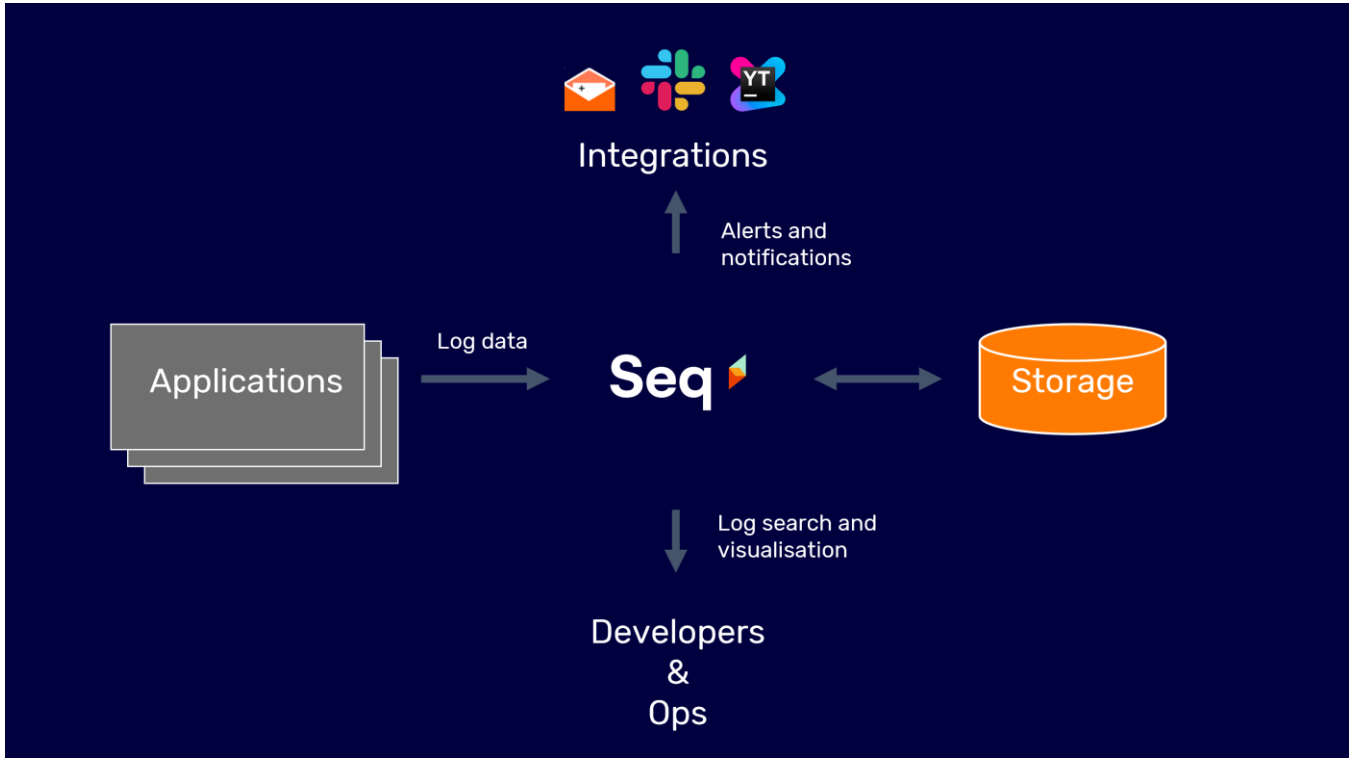


Decentralized Logging

Problem #8

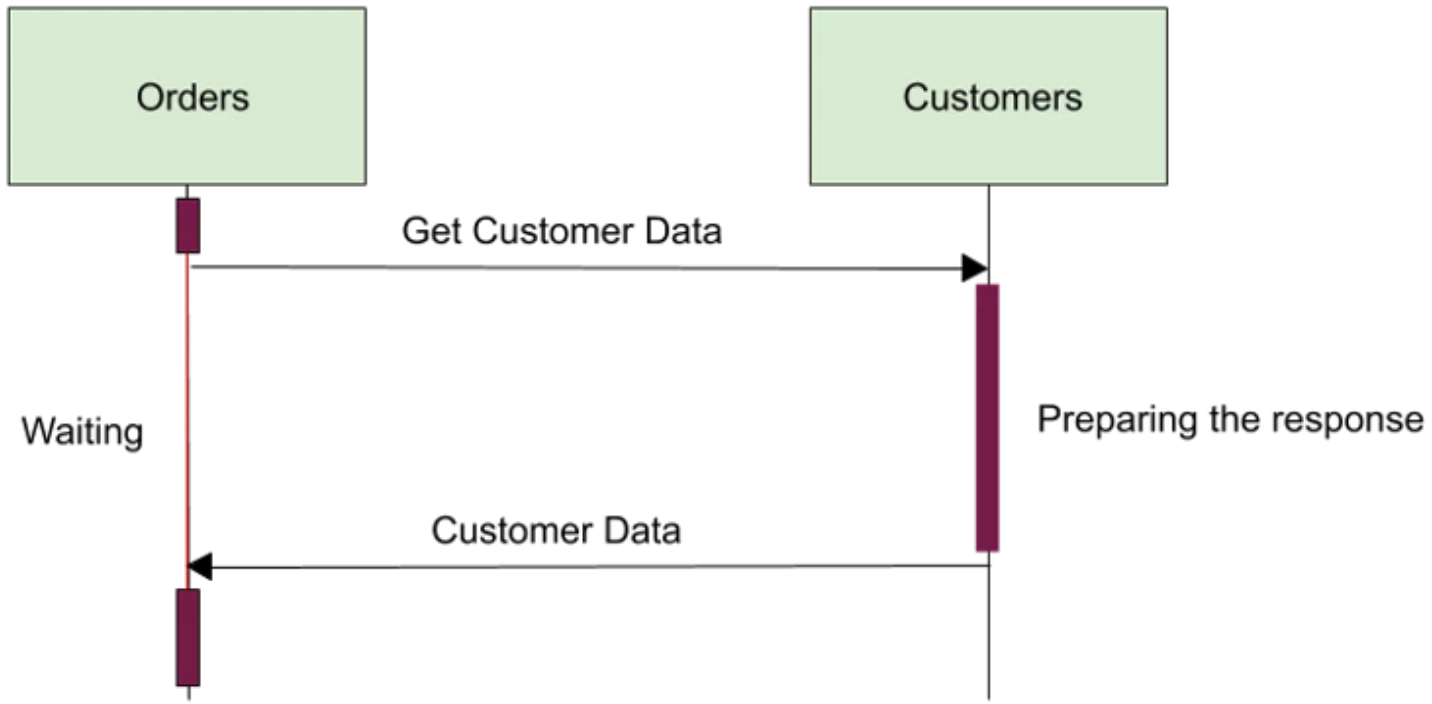
Debugging Distributed Systems Is Hard

- 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

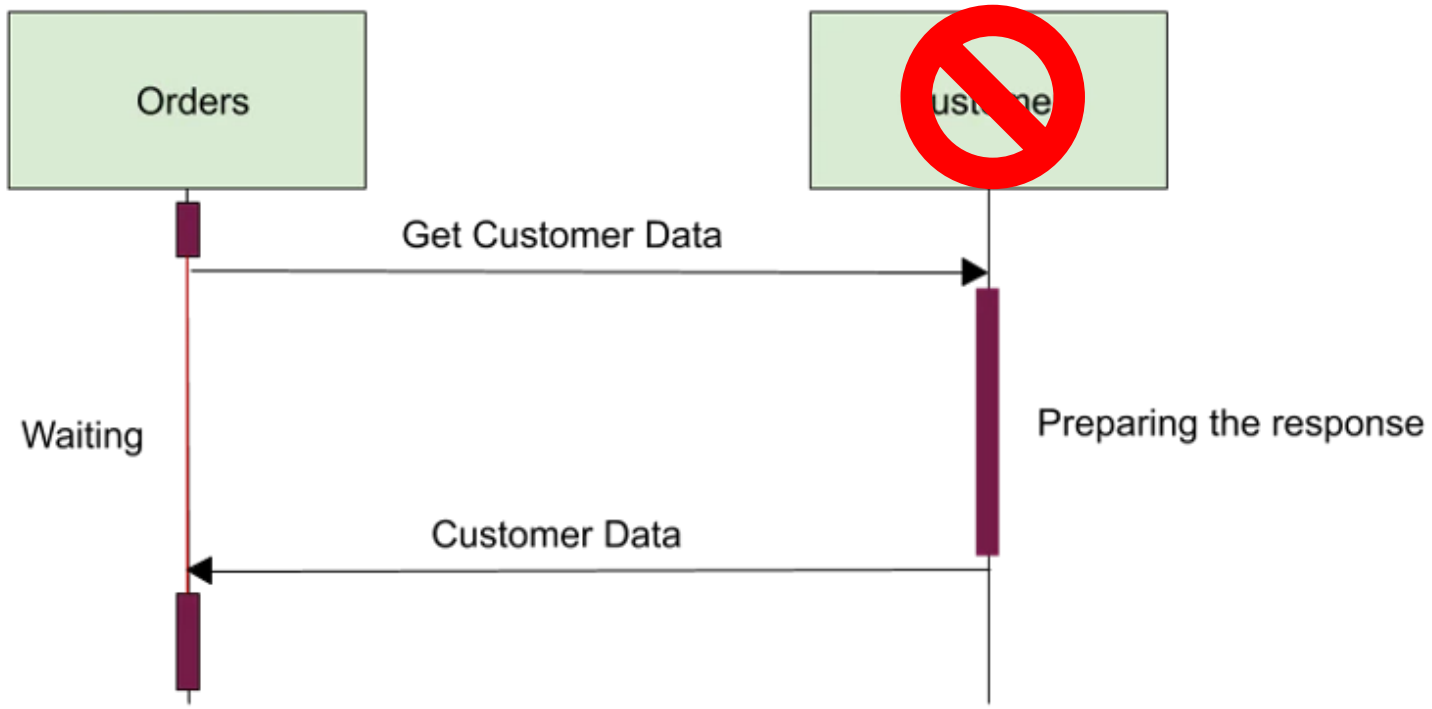


Synchronous Communication

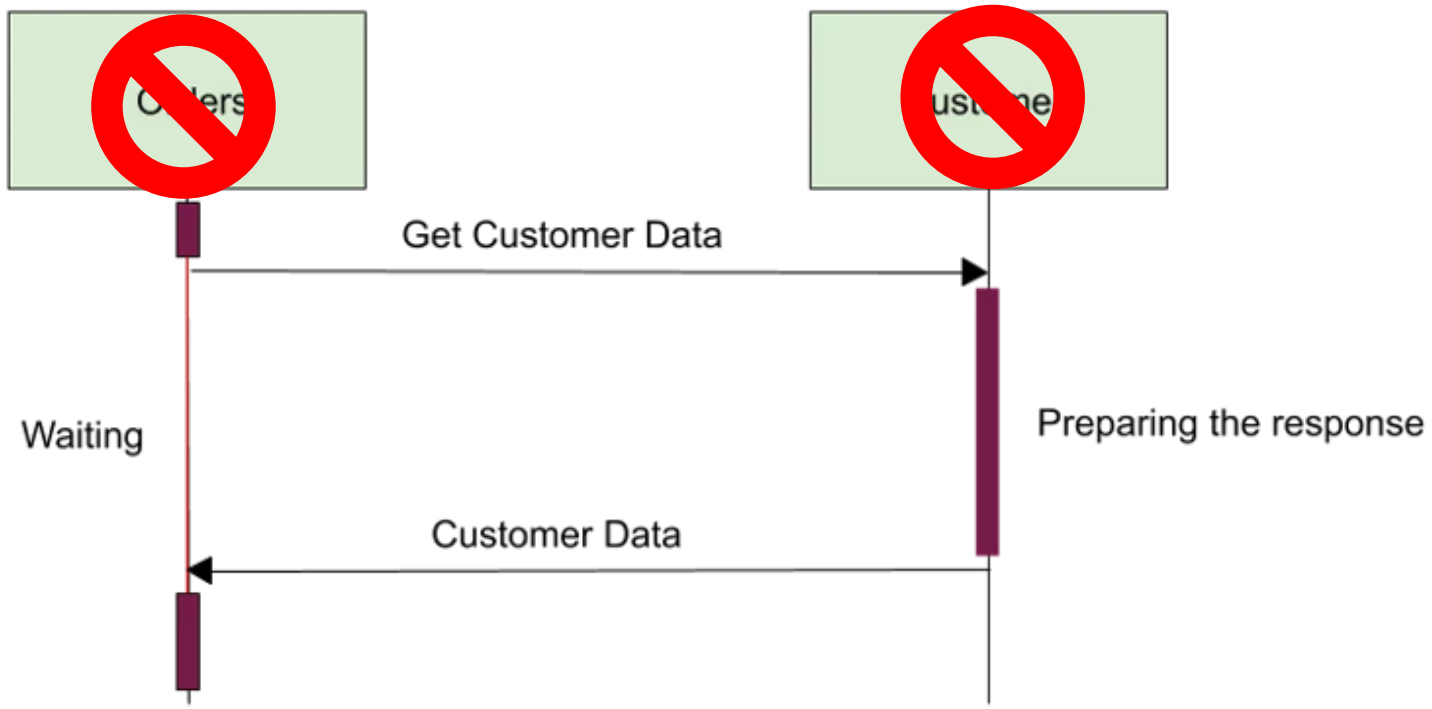
Problem #9



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/

The Big Trade Off

Microservices

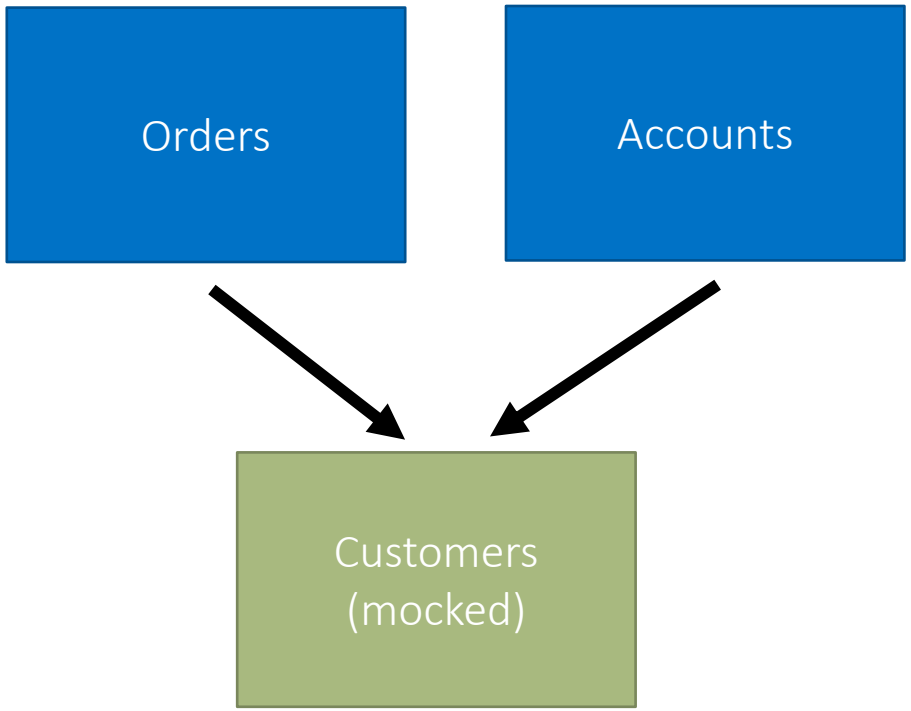
Monoliths

Shared Test Environment

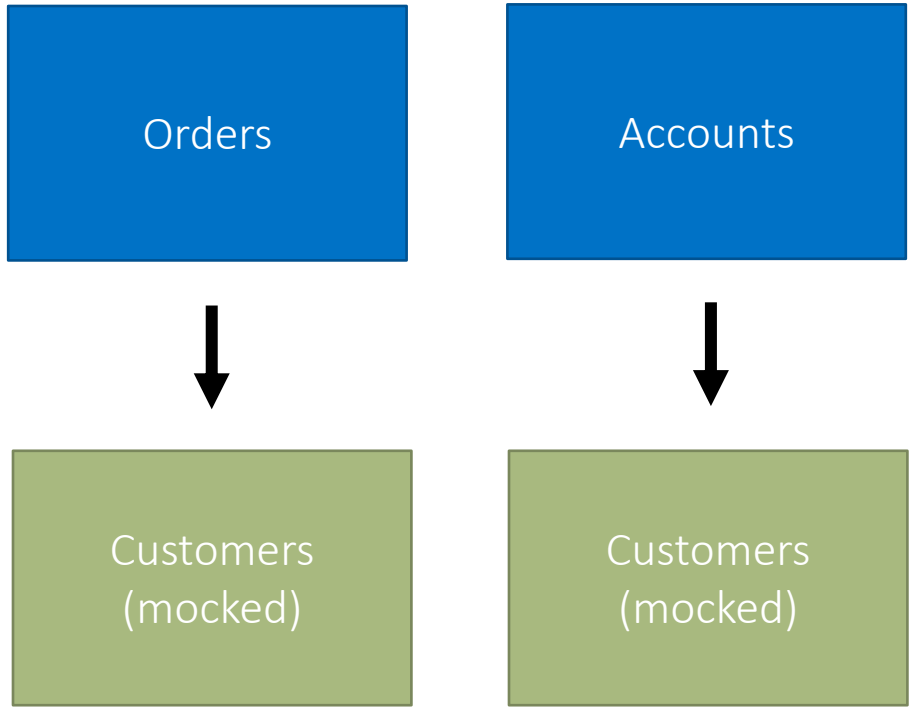
Problem #10

Dependency Coupling

BAD



GOOD



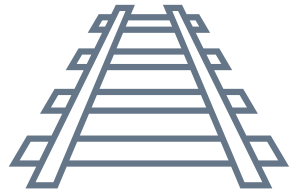
Not Automating Versioning and Release

Problem #11

Version and Release

Time consuming

Prone to human error



May need to support many concurrent verions

Mismatched Team Organization

Problem #12

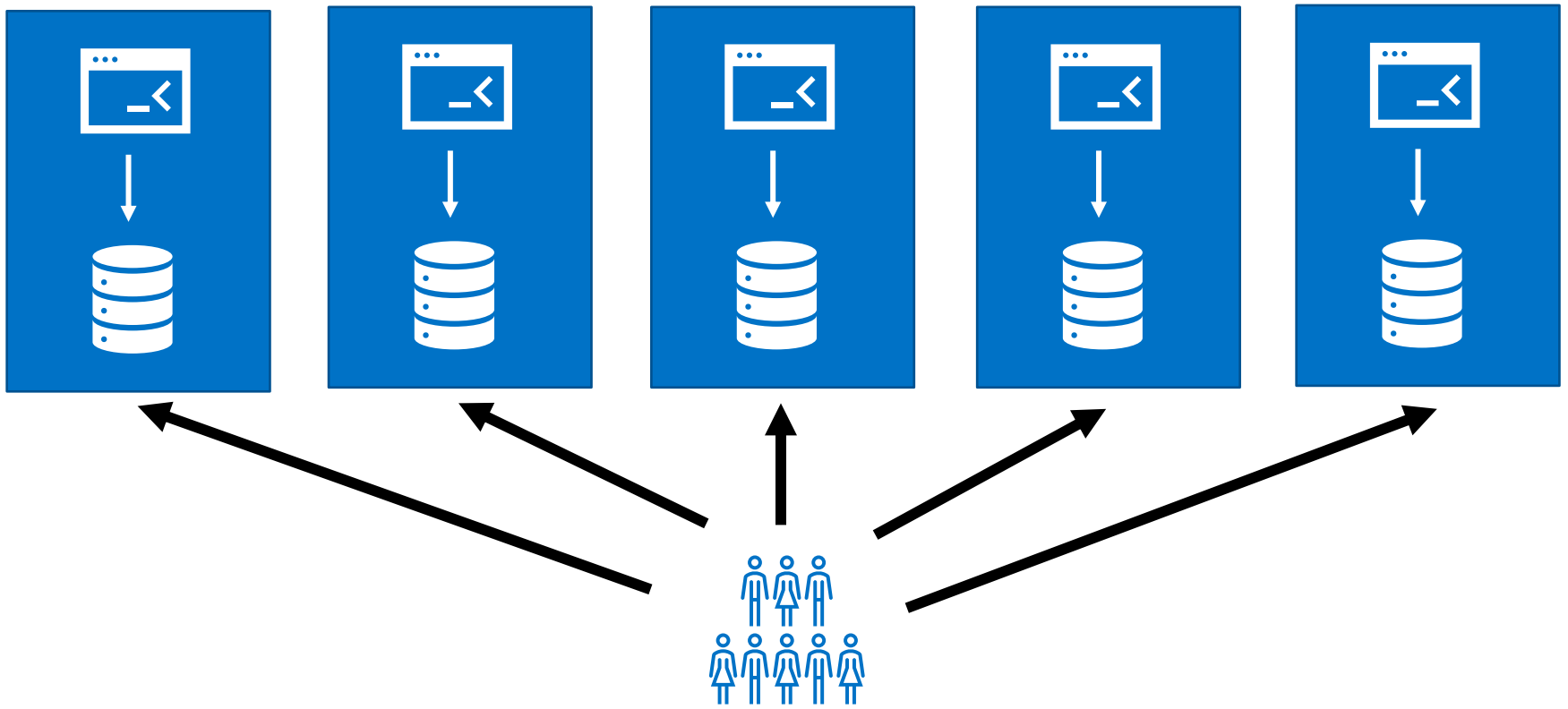
Conway's Law

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

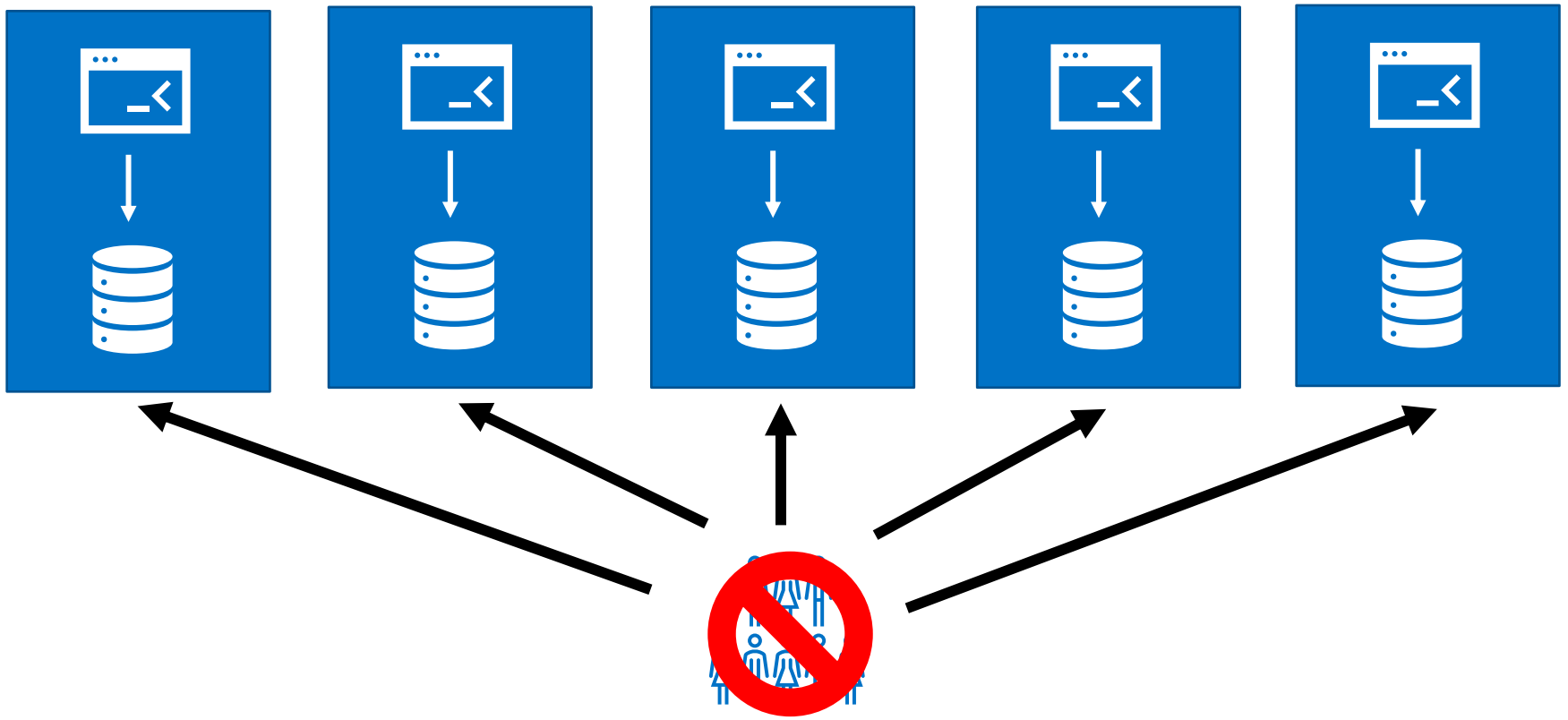
- Melvin E. Conway

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

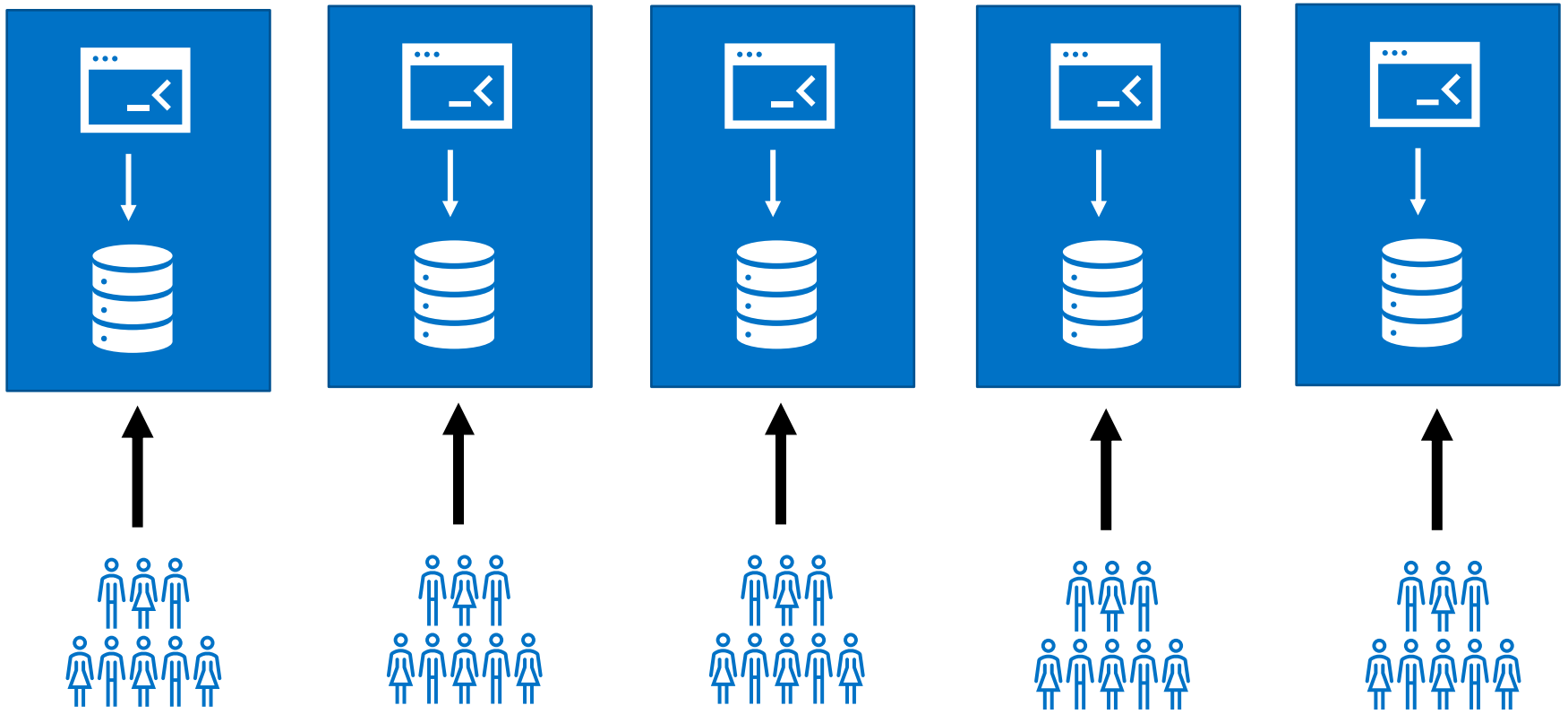
Single Team



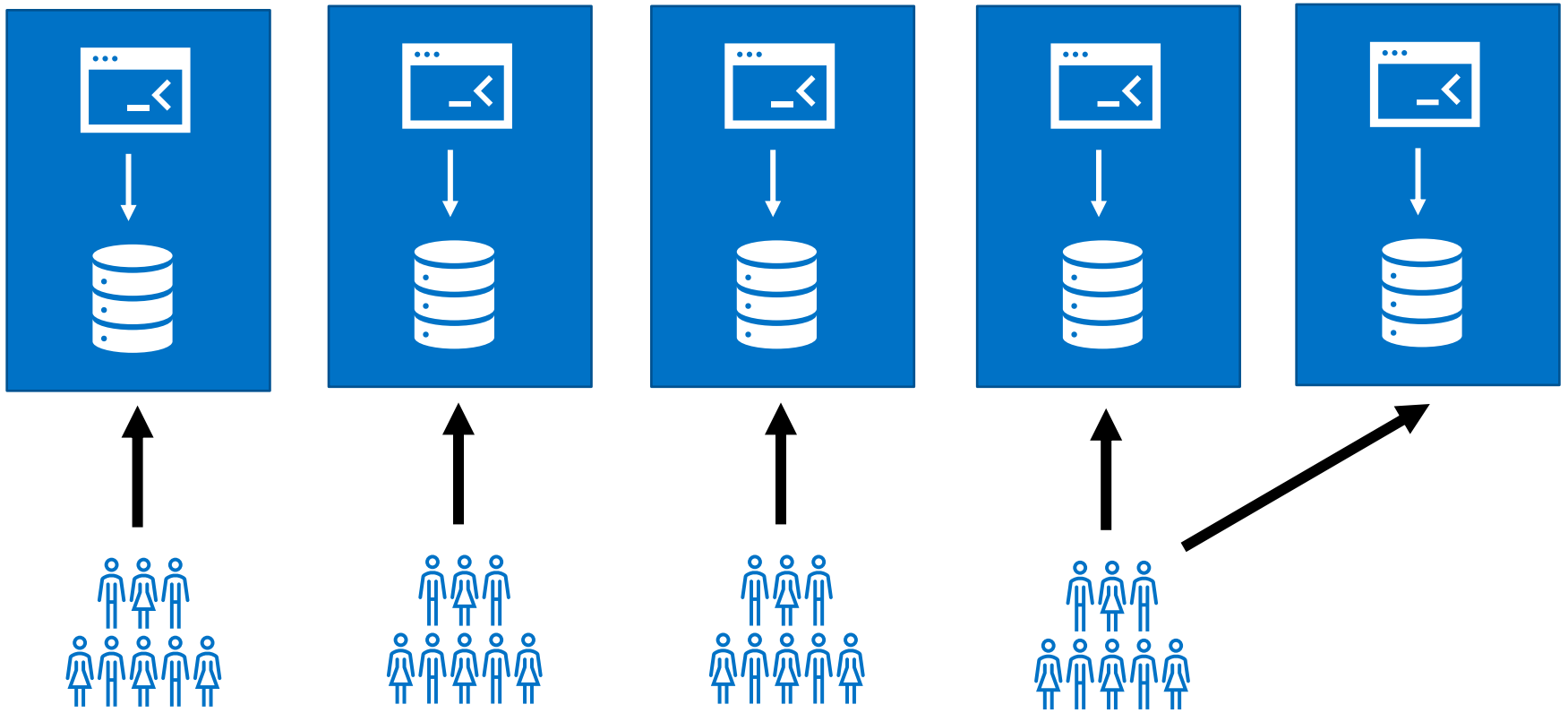
Single Team



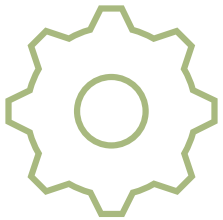
Team Per Service



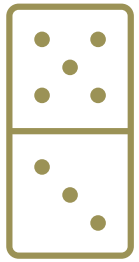
Team Per Service



Pipeline Per Service



Shared deployment or compilation



Can’t change one without impacting others

Goal: Totally independent releases

Unencapsulated Services

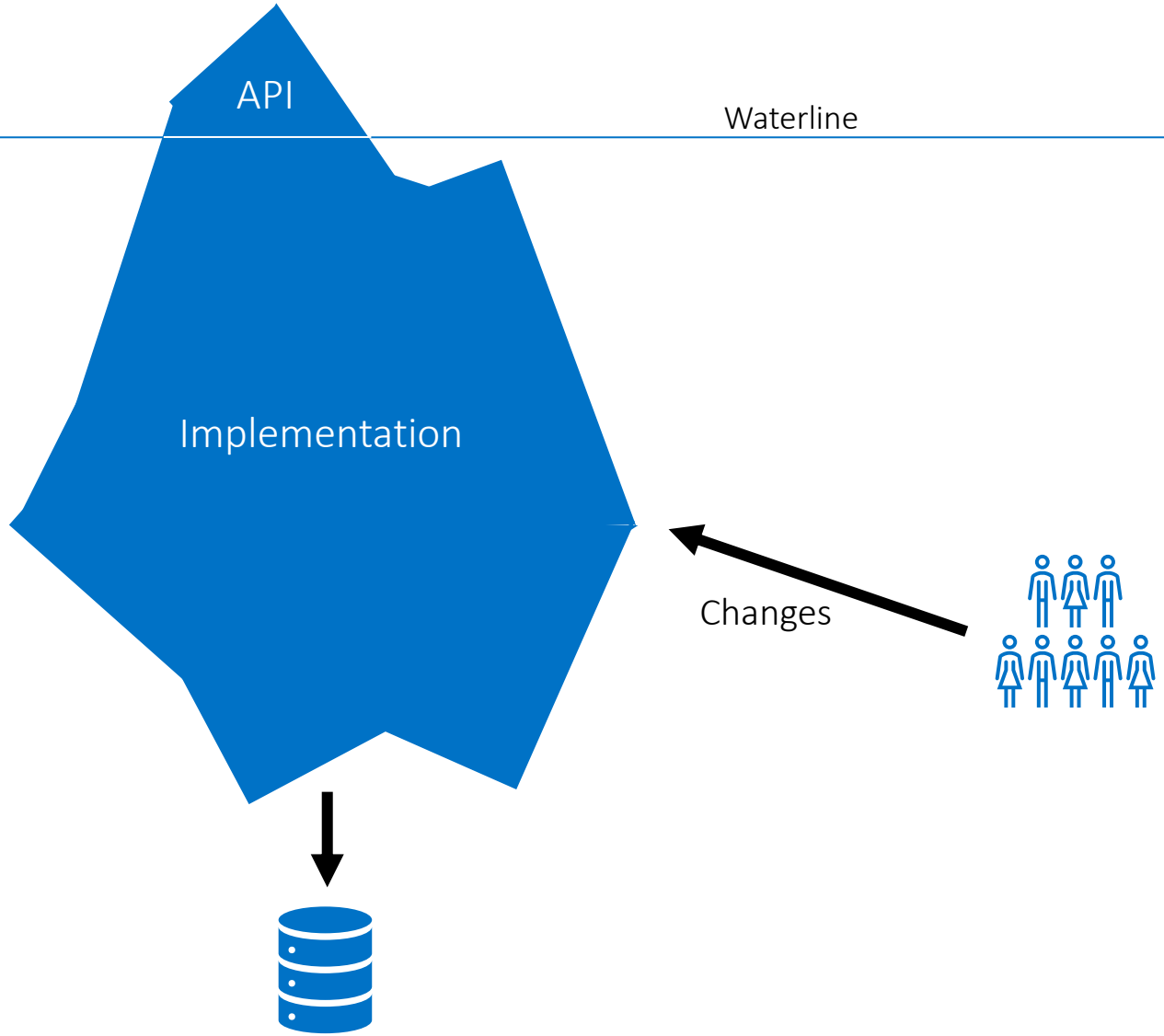
Problem #13

Iceberg Services

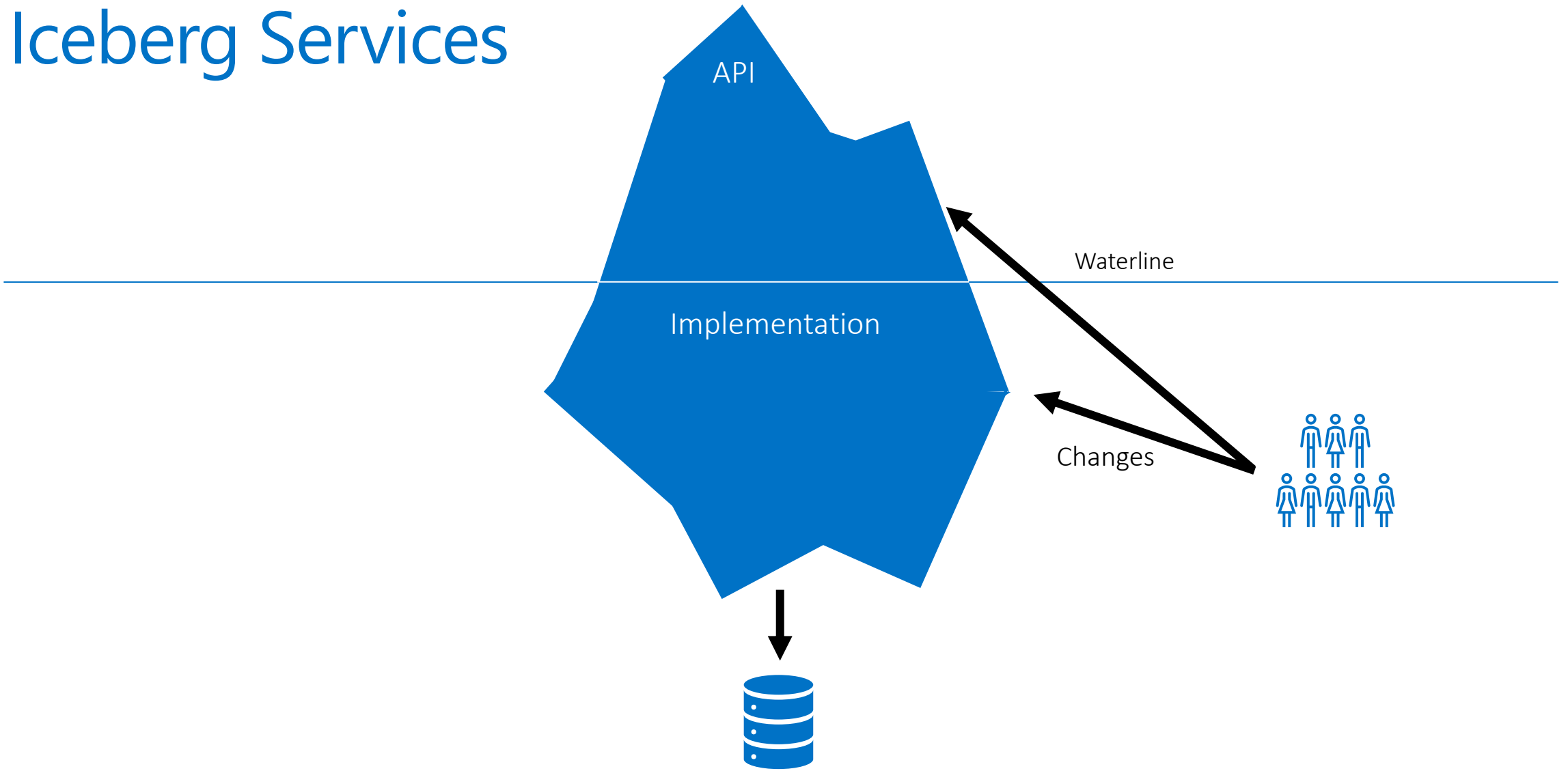
Services ENCAPSULATE significant business logic

Small, stable API

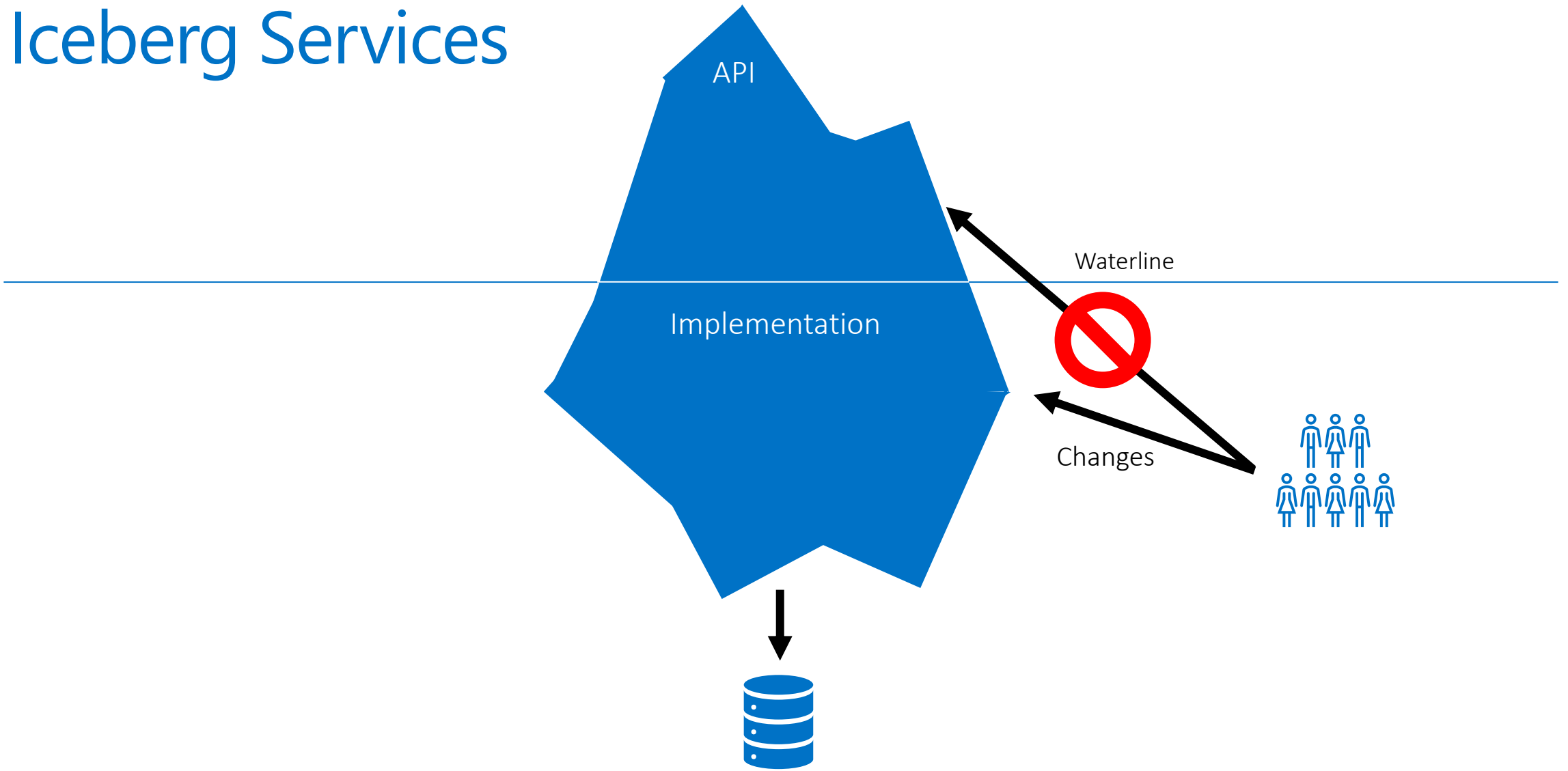
Large implementation



Iceberg Services



Iceberg Services



Summing Up

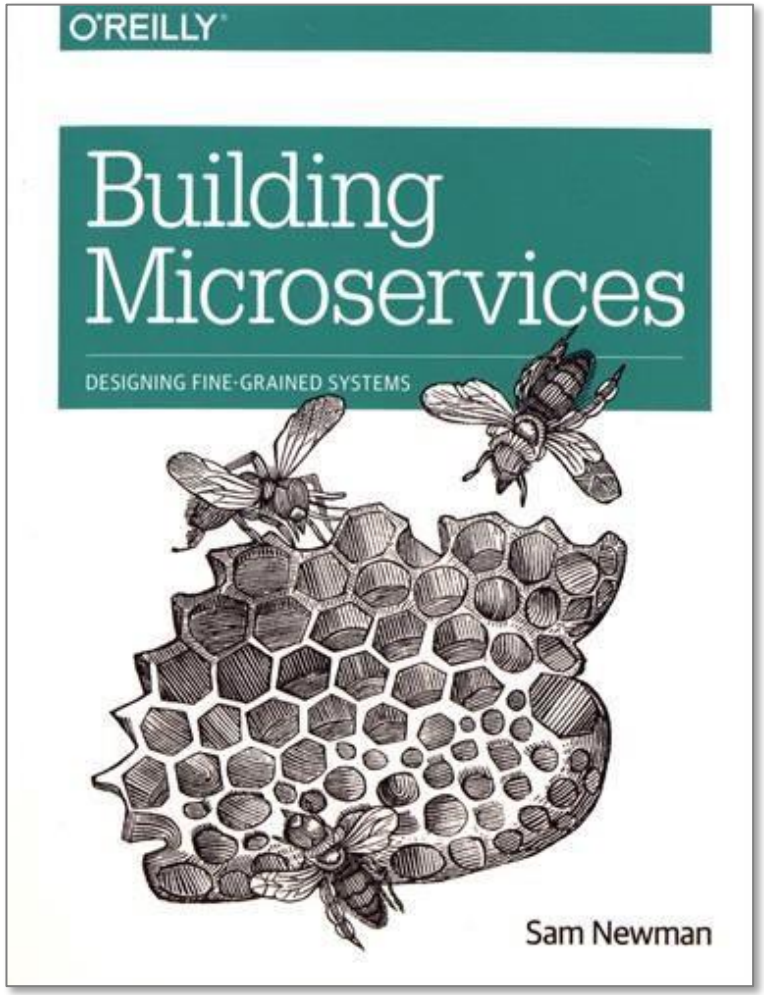
1. For a lightweight application or single team, a monolithic system often suits better

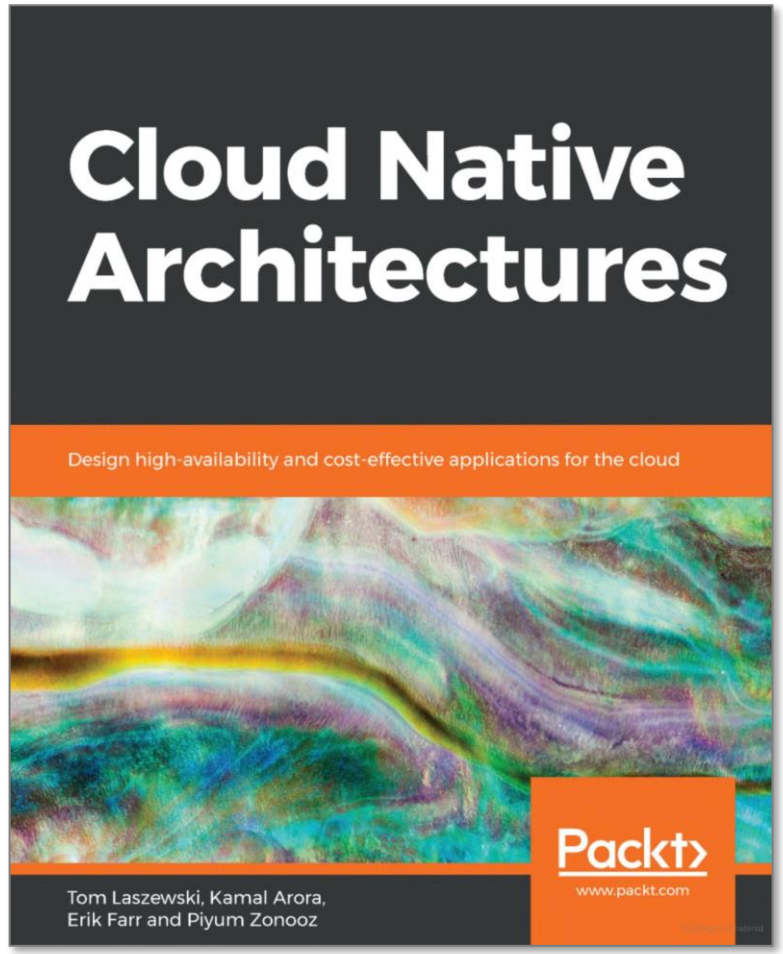
2. For a complex, evolving application with clear domains and separate teams, microservices will be best

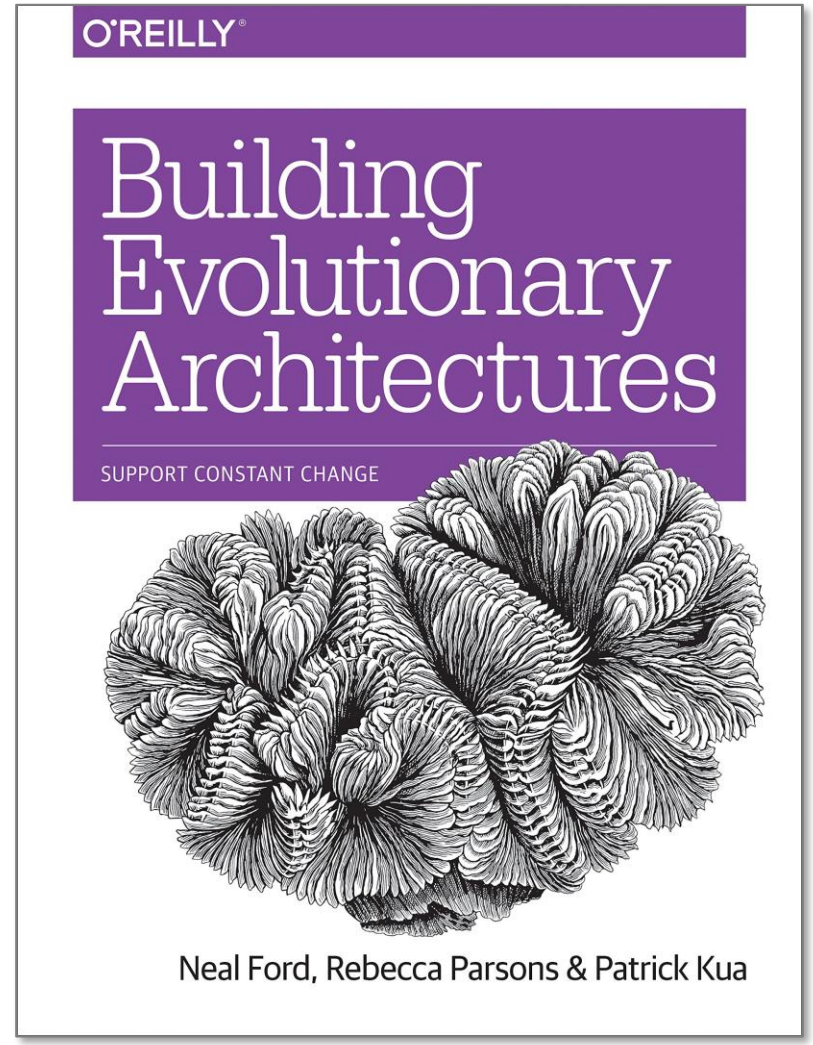
3. Don’t try microservices without “a really good reason”. Monoliths can be good!

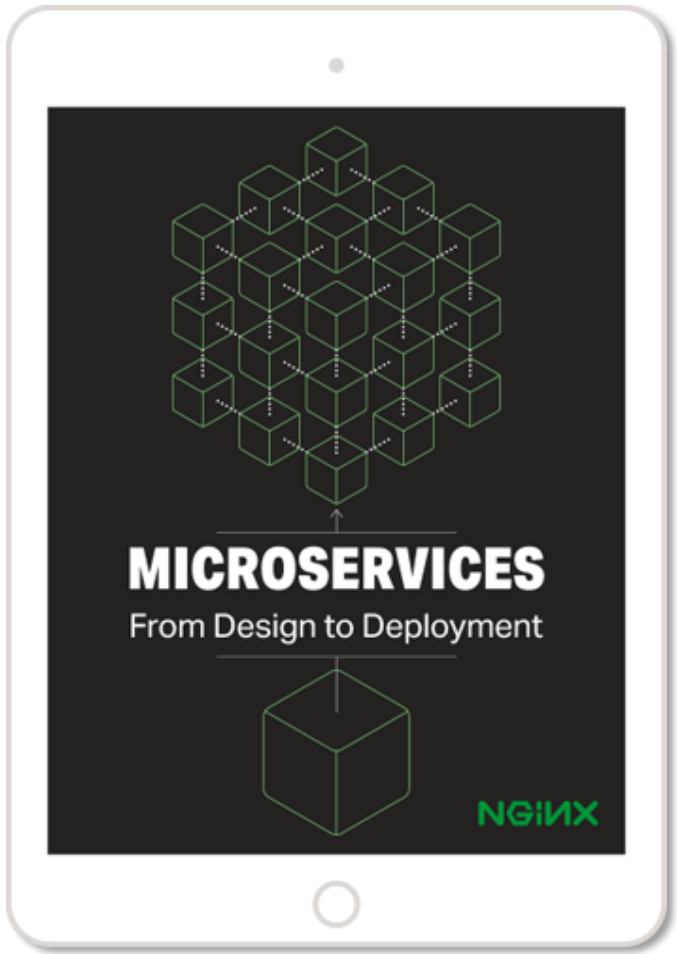
4. Avoid pitfalls of the distributed monolith

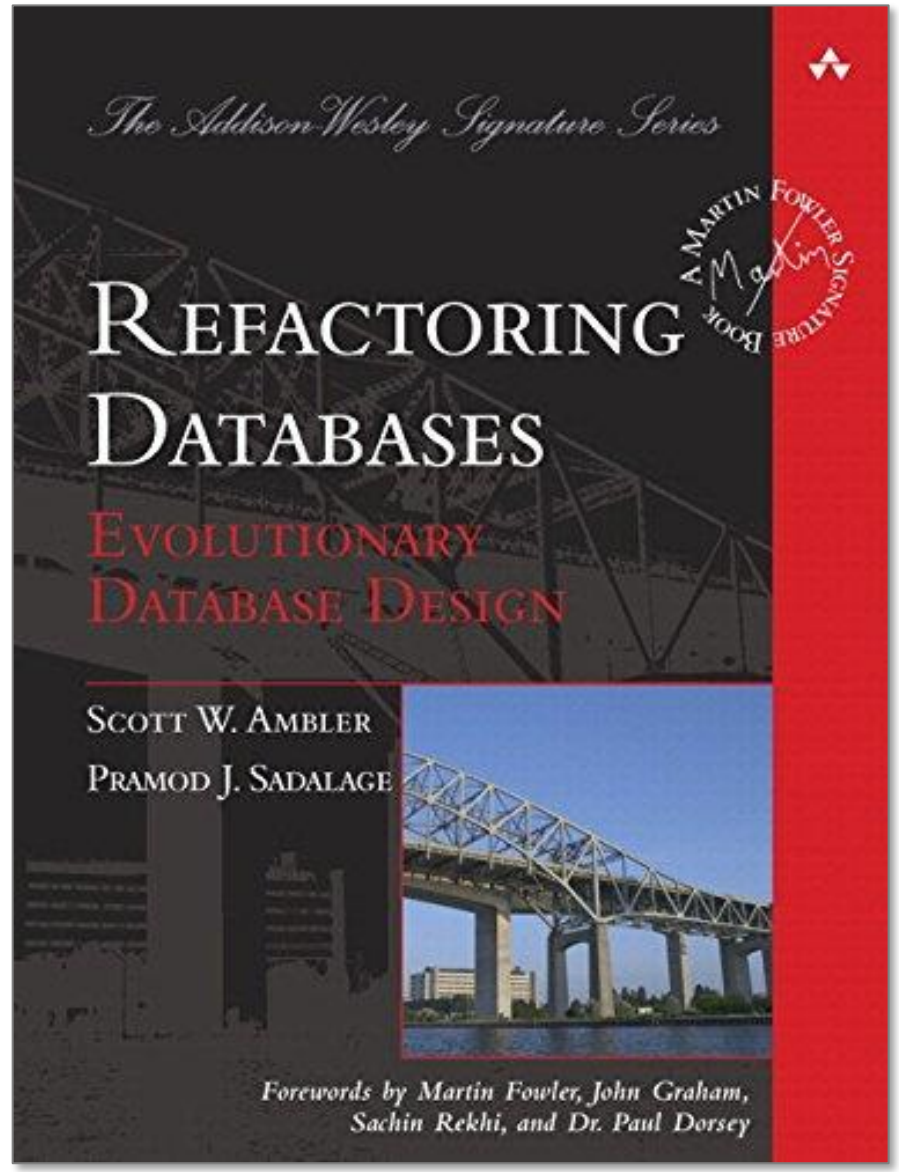
Further Reading


O'REILLY
Building Microservices
DESIGNING FINE-GRAINED SYSTEMS
Sam Newman


Cloud Native Architectures
Design high-availability and cost-effective applications for the cloud
Packt
www.packt.com
Tom Laszewski, Kamal Arora,
Erik Farr and Piyum Zonooz


O'REILLY
Building Evolutionary Architectures
SUPPORT CONSTANT CHANGE
Neal Ford, Rebecca Parsons & Patrick Kua


MICROSERVICES
From Design to Deployment
NGINX


The Addison Wesley Signature Series
A Martin Fowler Signature Book
Martin
REFACTORING DATABASES
EVOLUTIONARY DATABASE DESIGN
SCOTT W. AMBLER
PRAMOD J. SADALAGE
Forewords by Martin Fowler, John Graham, Sachin Rekhi, and Dr. Paul Dorsey

Online Resources

https://martinfowler.com/microservices

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

https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/introduce-eshoponcontainers-reference-app

https://docs.microsoft.com/en-us/dotnet/architecture/microservices/

Recap

- Definitions
  - Monolith
  - Microservices
  - Distributed Monolith
- 13 Most Common Mistakes
- Further Reading
- Q&A

Thank You! Questions?

Jonathan "J." Tower

Principal Consultant & Partner

Trailhead Technology Partners

TRAILHEAD
TECHNOLOGY PARTNERS

trailheadtechnology.com

- Microsoft MVP in .NET
- Organizer of Beer City Code

- jtower@trailheadtechnology.com
- trailheadtechnology.com/blog
- jtowermi

https://github.com/jonathantower/distributed-monolith