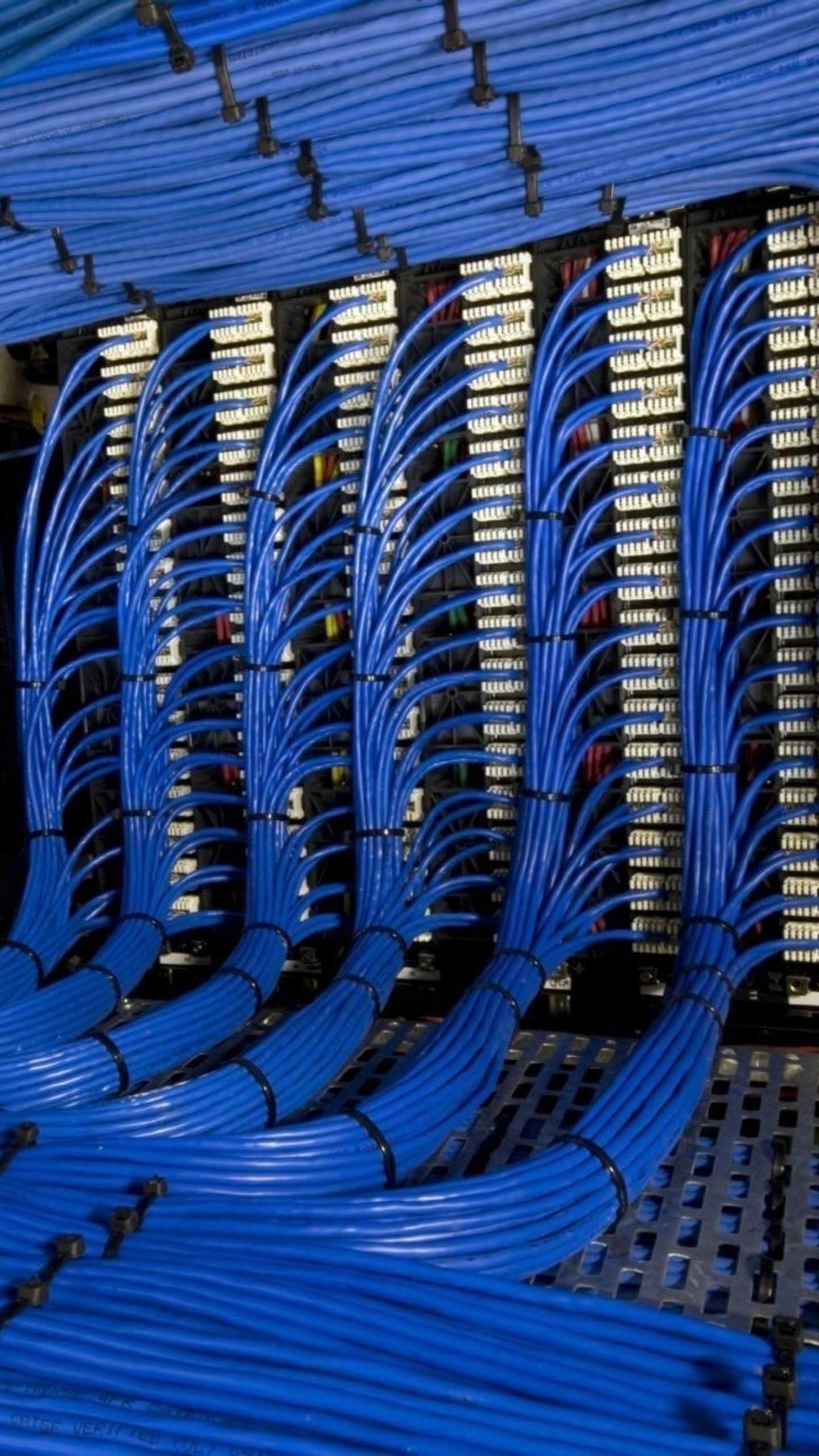
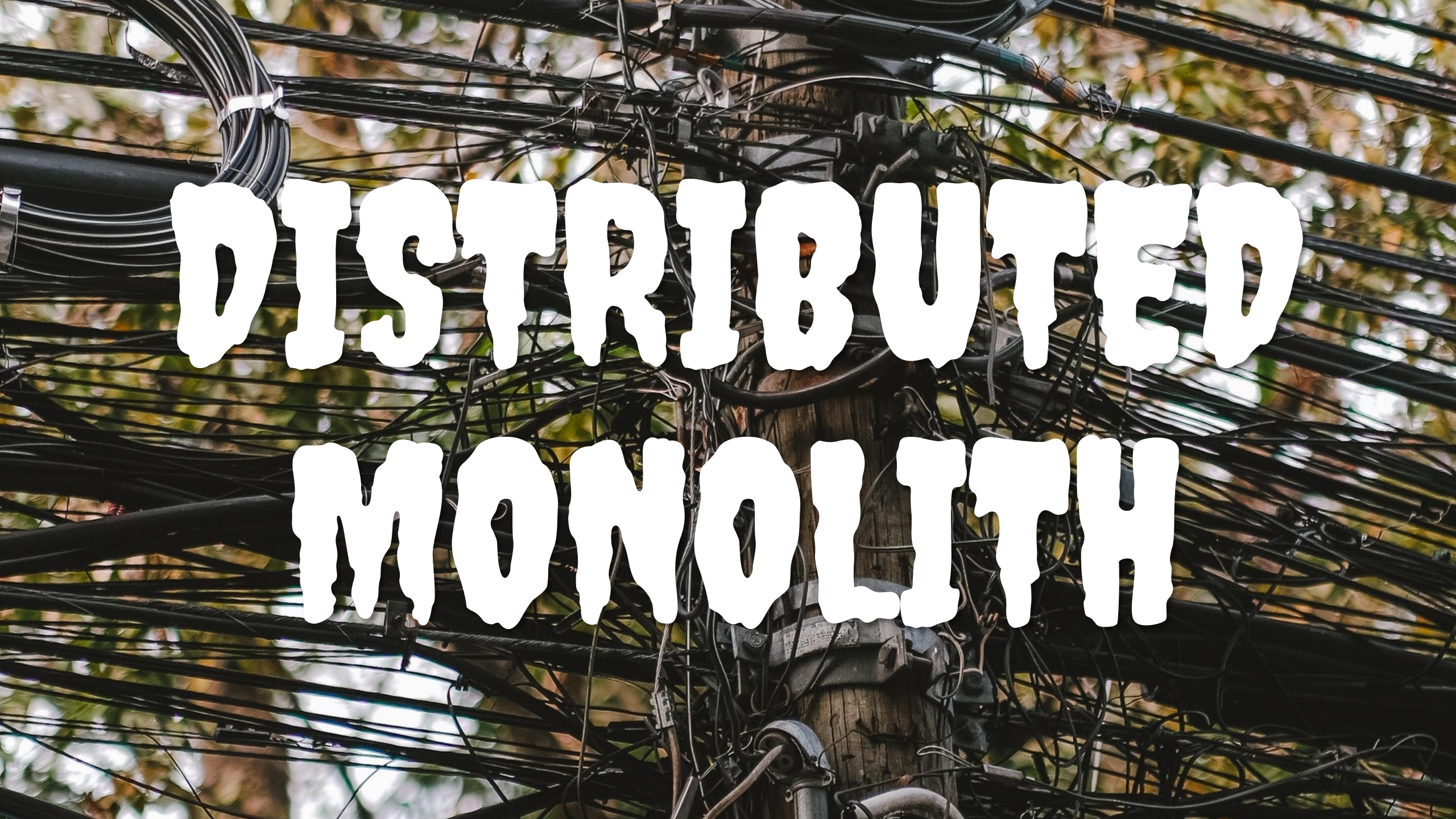
Don’t Build a Distributed Monolith

How to Avoid Doing Microservices Completely Wrong

Jonathan "J." Tower




DISTRIBUTED
MONOLITH

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

Jonathan "J." Tower

Principal Consultant & Partner

TRAILHEAD
TECHNOLOGY PARTNERS

Microsoft MVP in .NET

jtower@trailheadtechnology.com

trailheadtechnology.com/blog

jtowermi

Free Offer

30-Minute Consultation

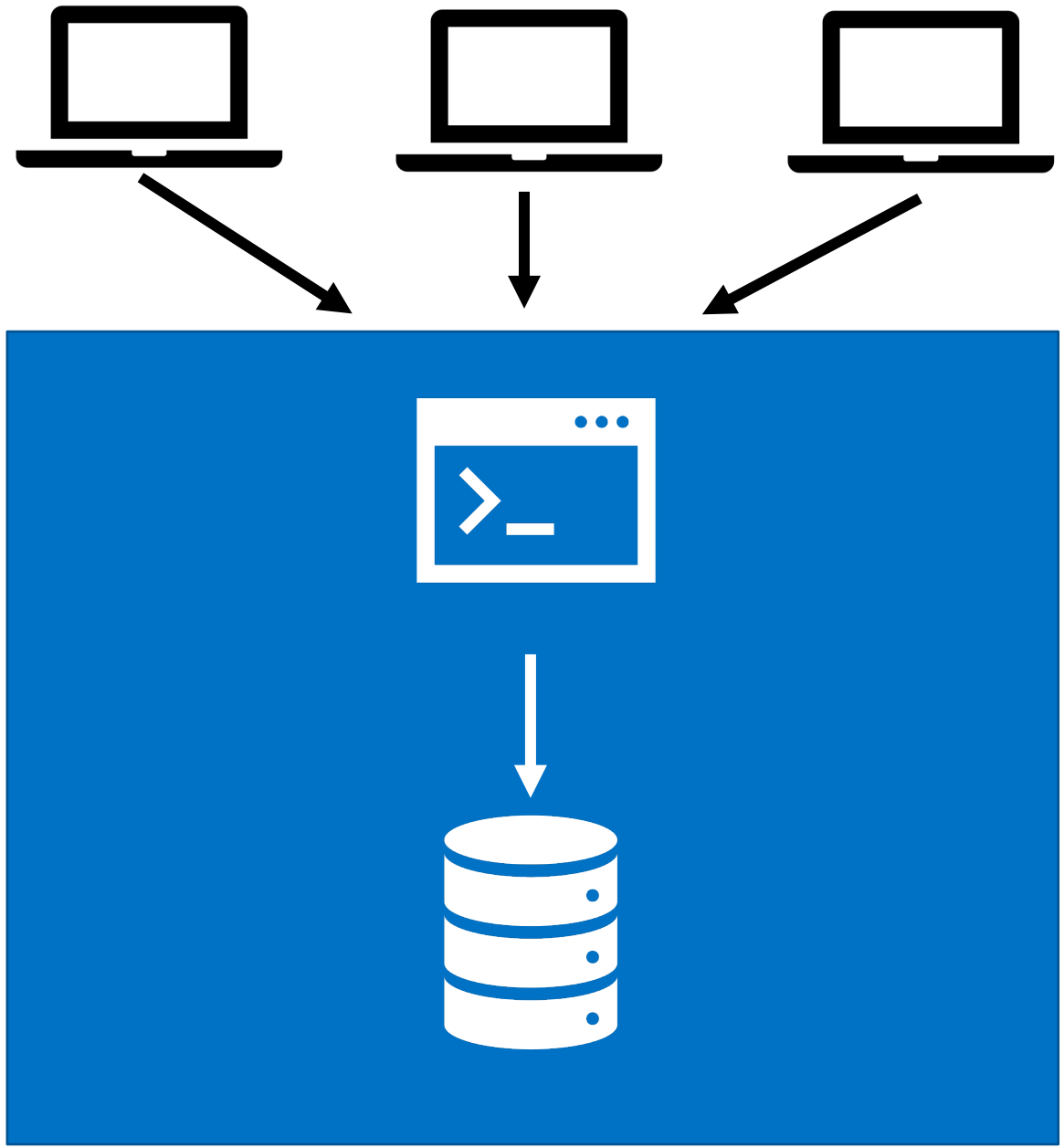
bit.ly/th-offer

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

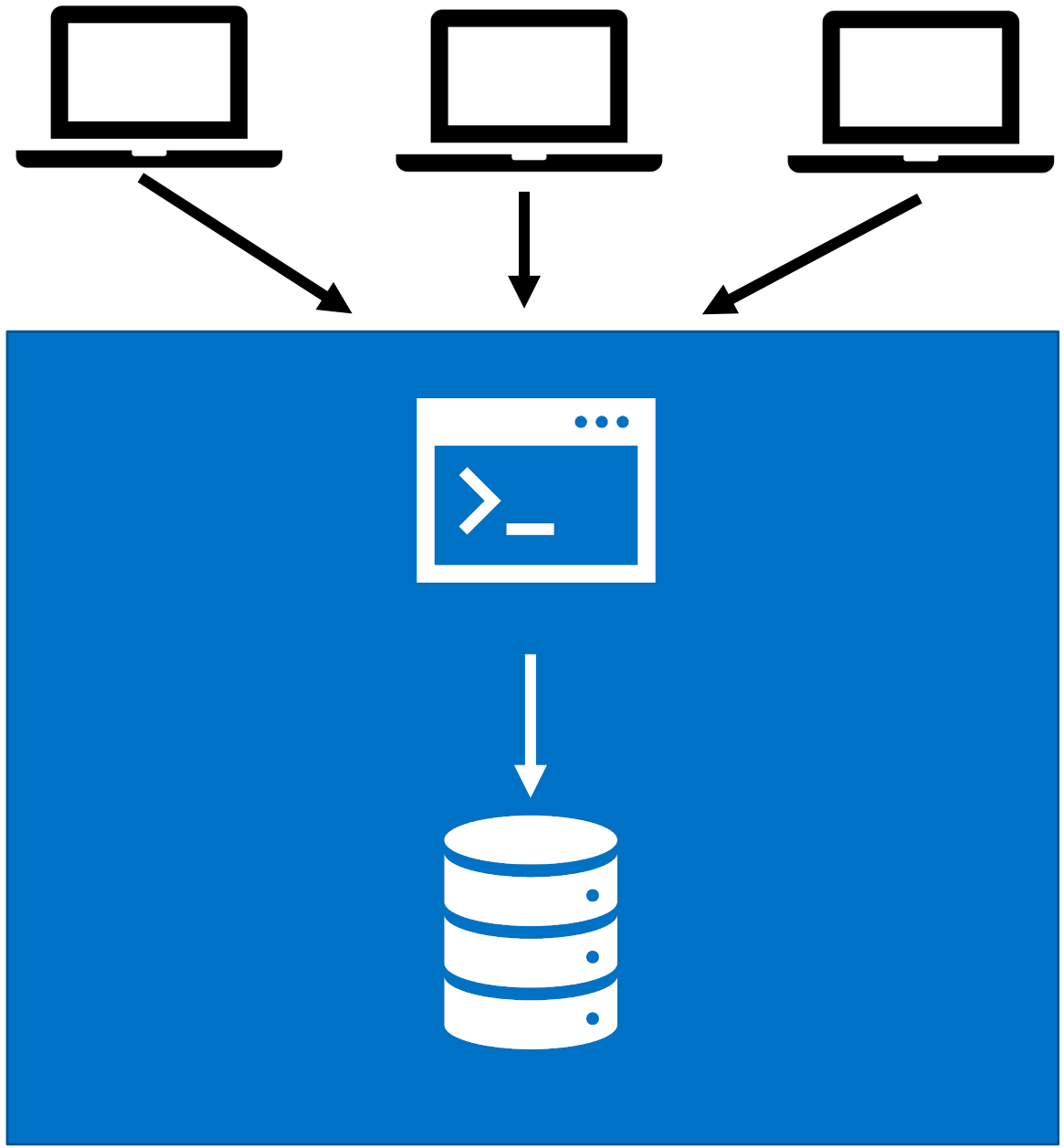
Some Definitions

TRAILHEAD
TECHNOLOGY PARTNERS

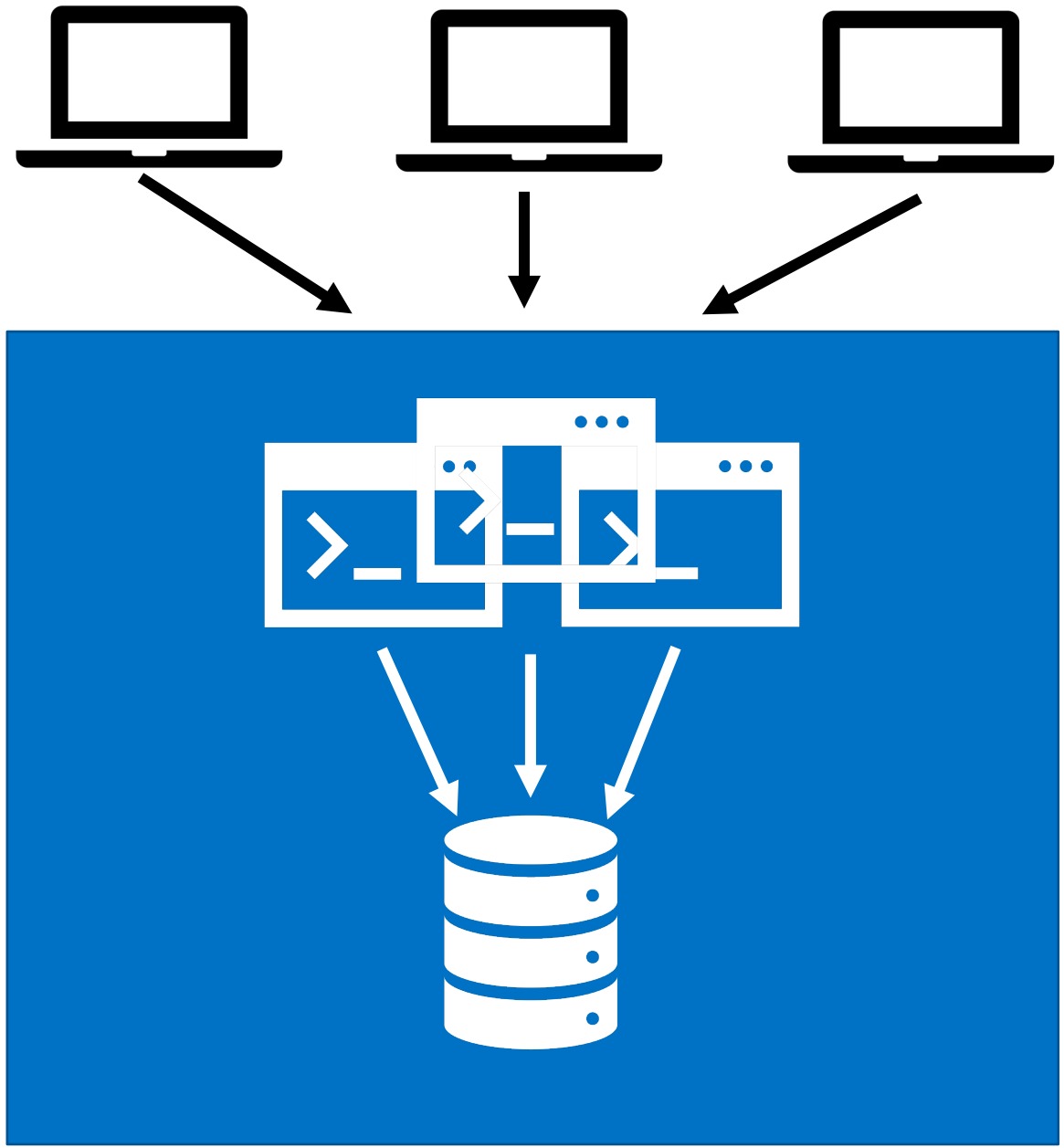

Monolith



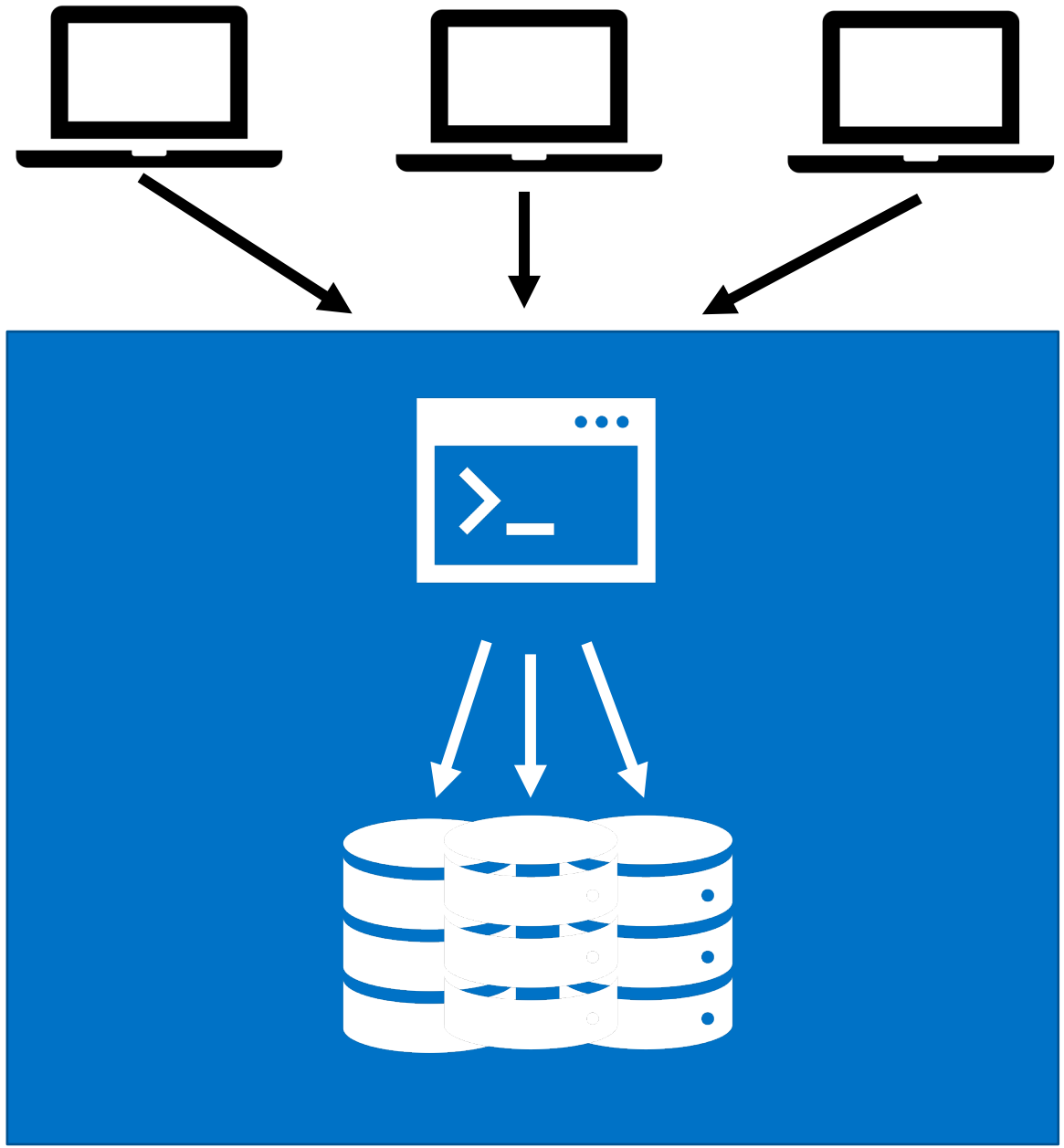
TRAILHEAD
TECHNOLOGY PARTNERS



TRAILHEAD
TECHNOLOGY PARTNERS

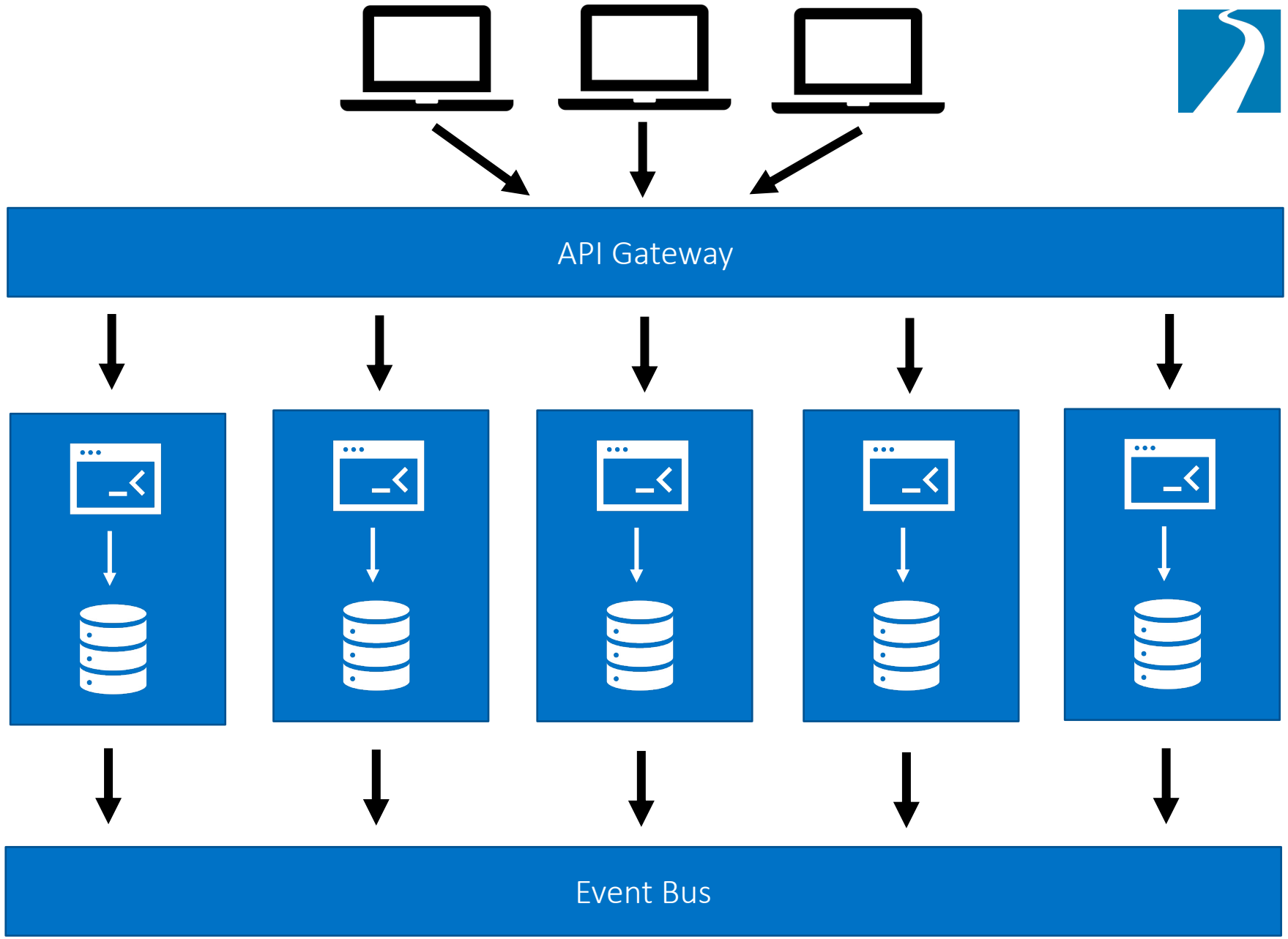


TRAILHEAD
TECHNOLOGY PARTNERS



TRAILHEAD
TECHNOLOGY PARTNERS

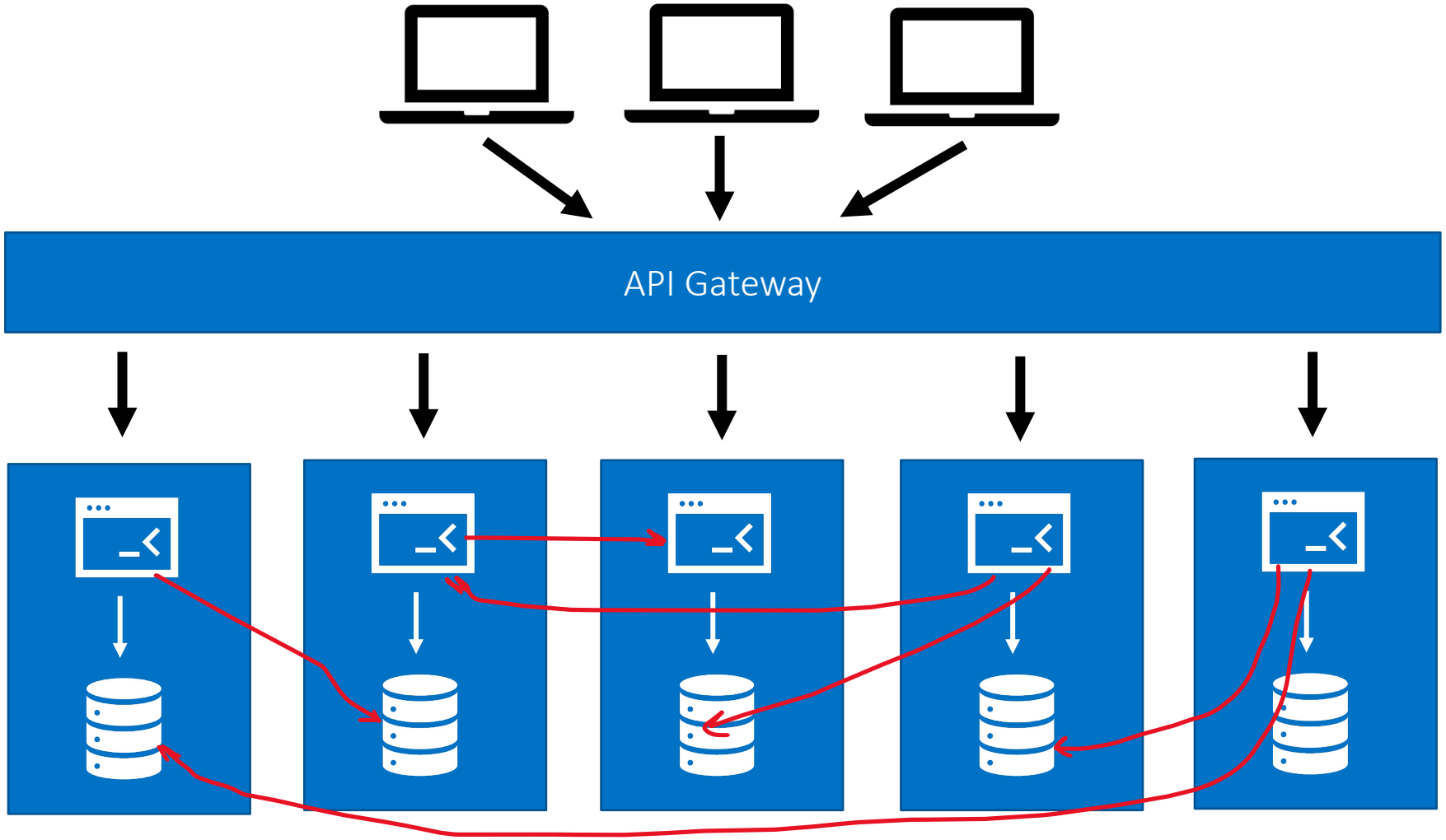
Microservices



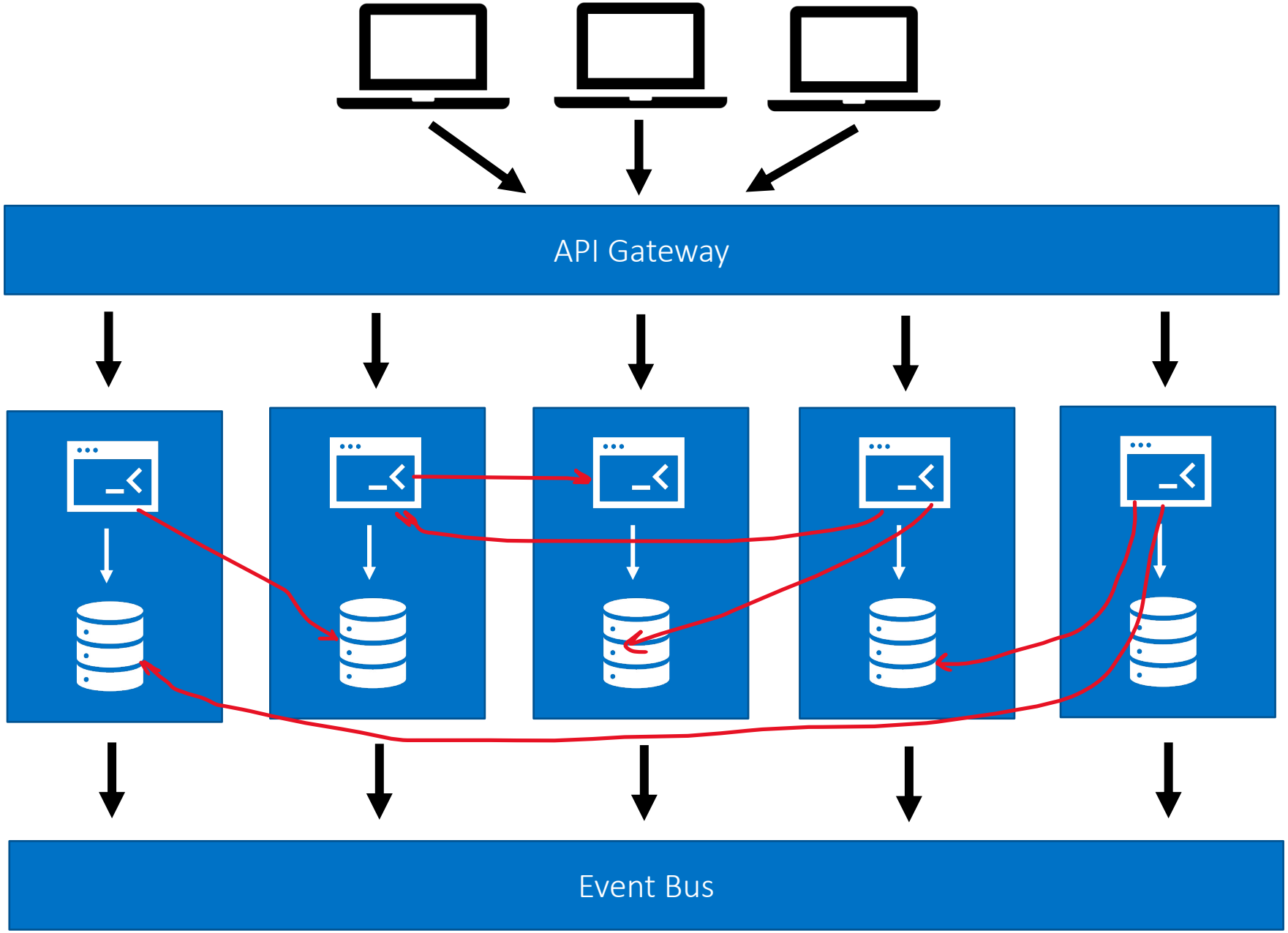
TRAILHEAD
TECHNOLOGY PARTNERS

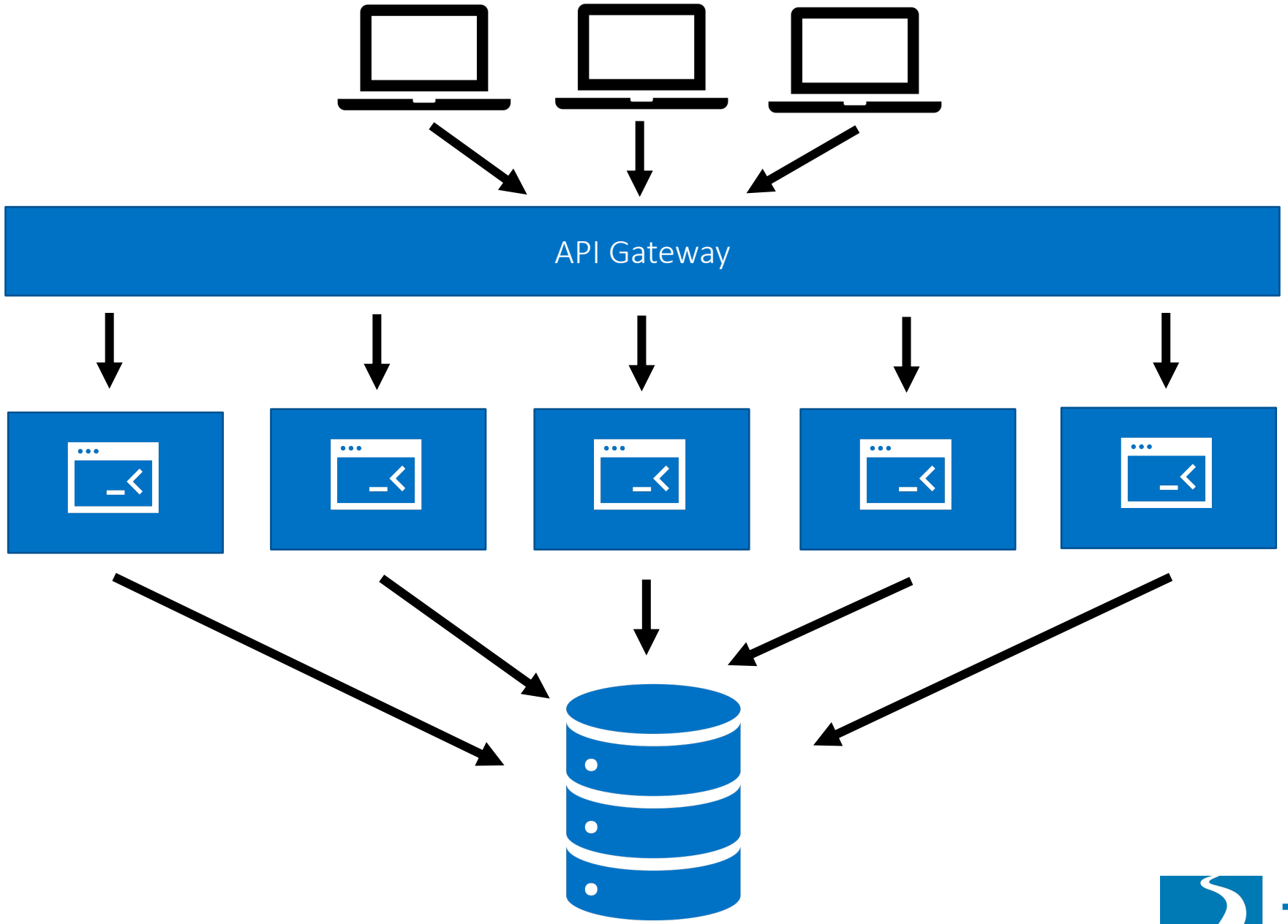
Distributed Monolith

Distributed Monolith



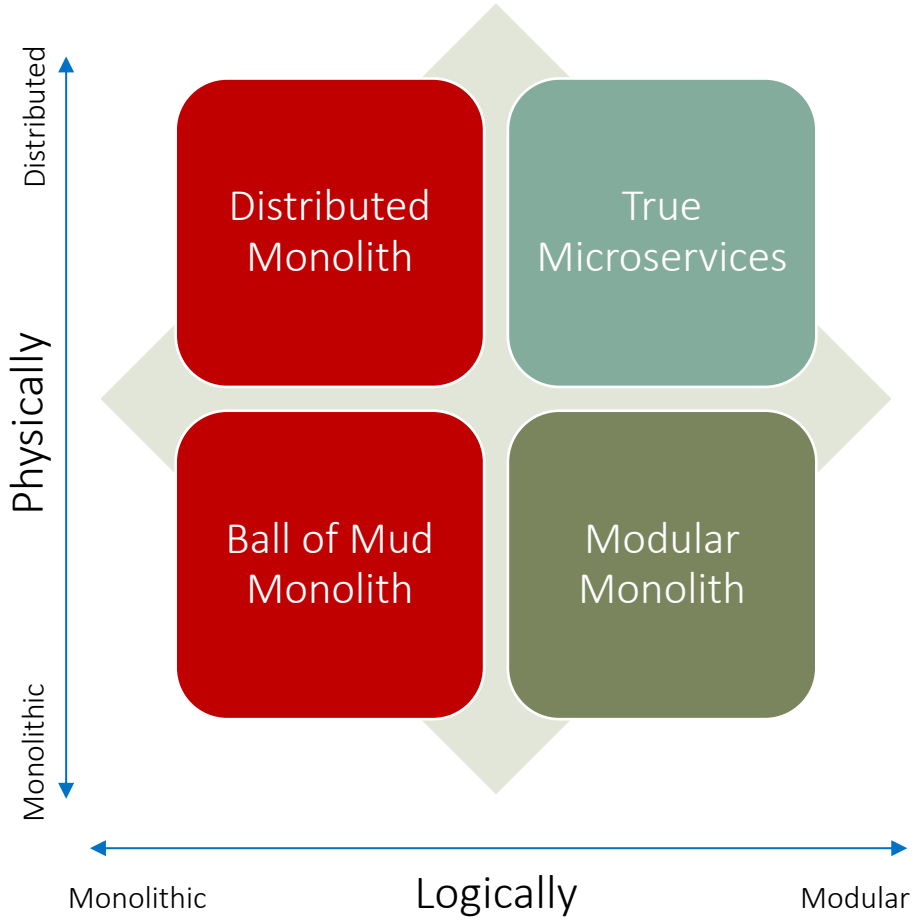
TRAILHEAD
TECHNOLOGY PARTNERS


API Gateway
Event Bus



TRAILHEAD
TECHNOLOGY PARTNERS

Good vs Bad Monoliths



TRAILHEAD
TECHNOLOGY PARTNERS

10 Most Common Mistakes

Avoid Creating a “Monster”

TRAILHEAD
TECHNOLOGY PARTNERS

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

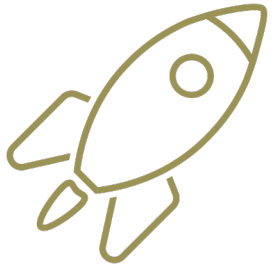
Wrong reasons create distributed monoliths

TRAILHEAD
TECHNOLOGY PARTNERS

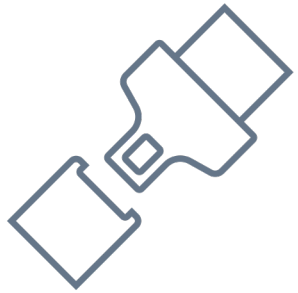
Some Good Reasons to Microservice

"Microservice architectures **buy** you **options**" – James Lewis

More Scalability Options



Independent Deployability



Isolate Surface Area of Failure

TRAILHEAD TECHNOLOGY PARTNERS

The Big Trade Off

Microservices

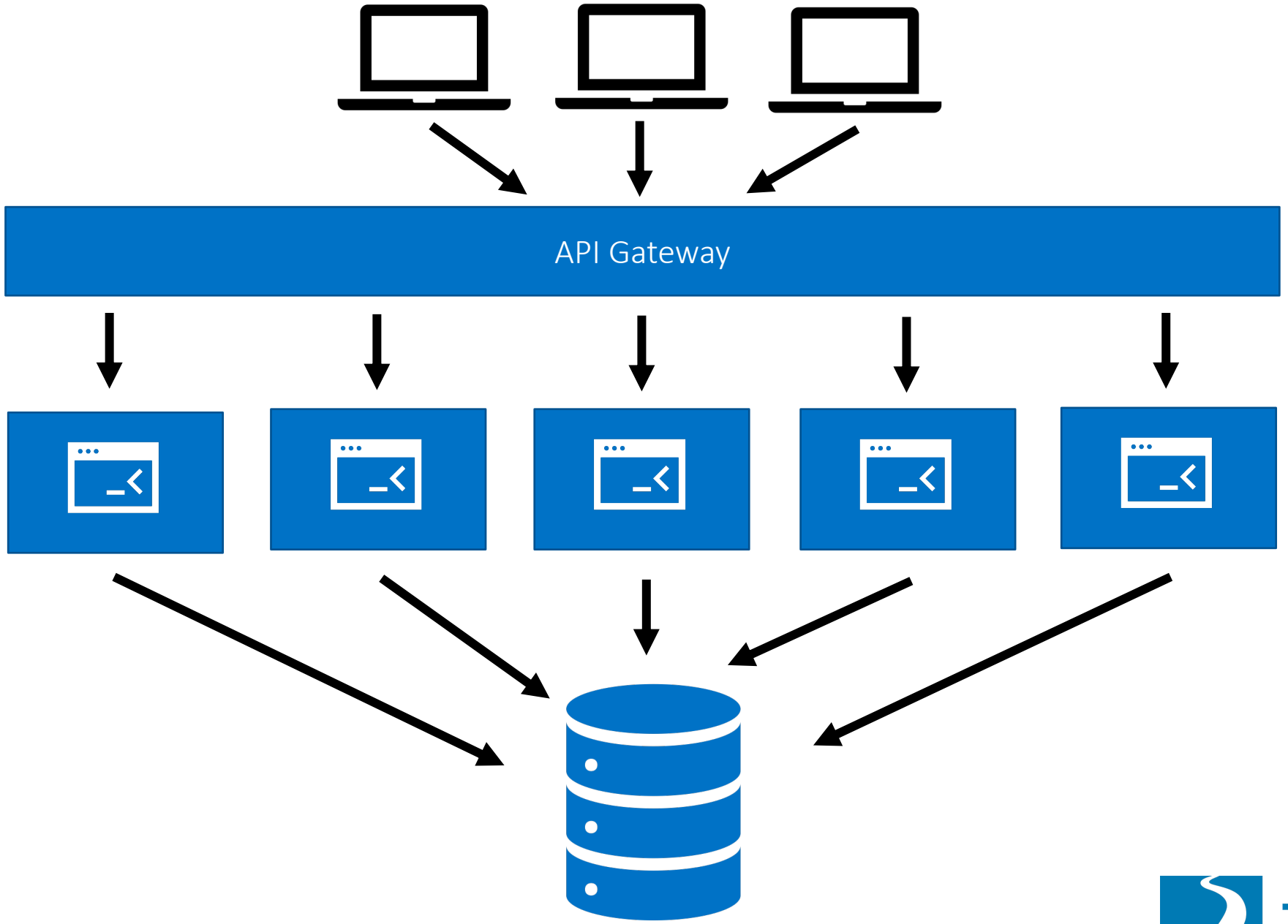
Monoliths

TRAILHEAD
TECHNOLOGY PARTNERS

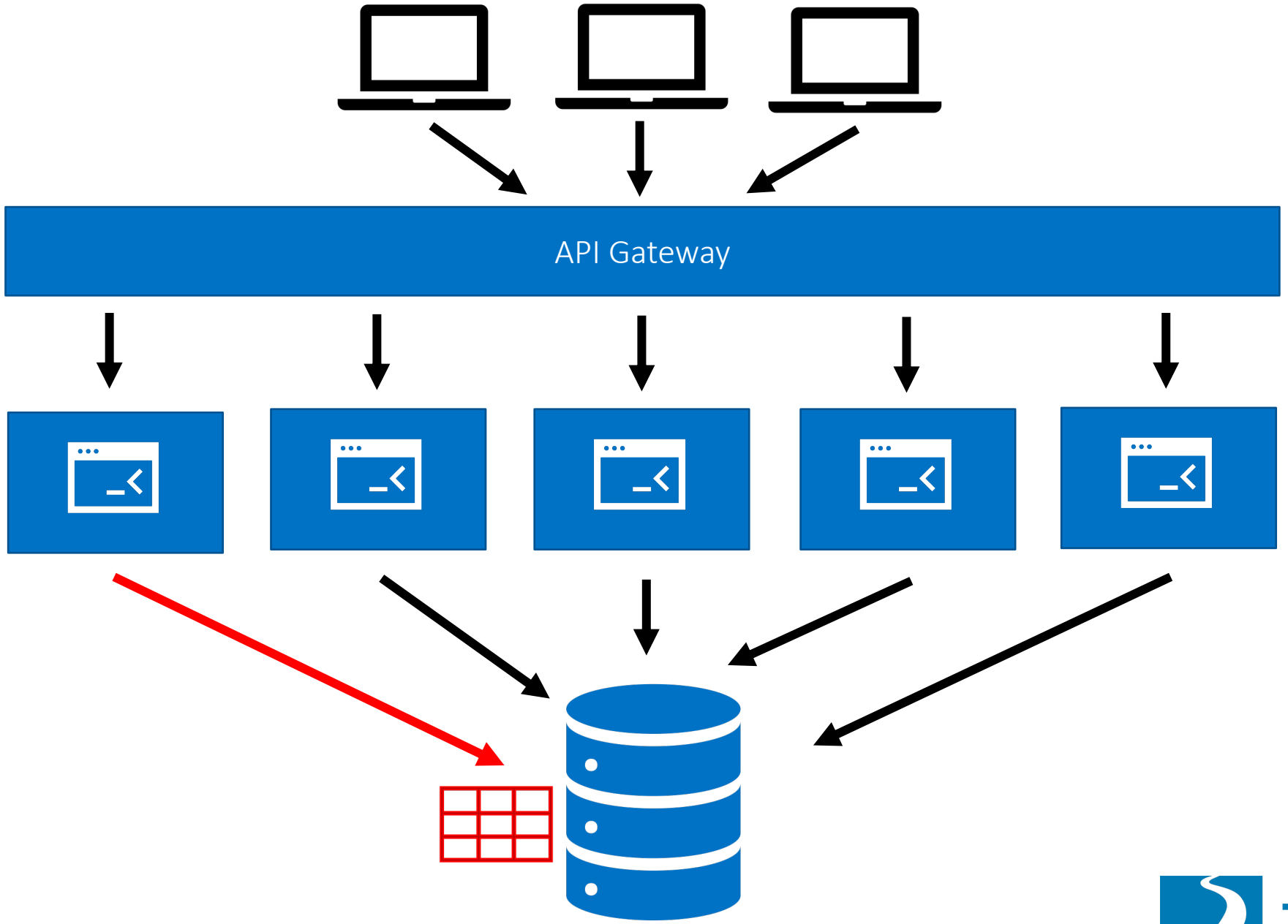
Shared Data Store or Models

Problem #2

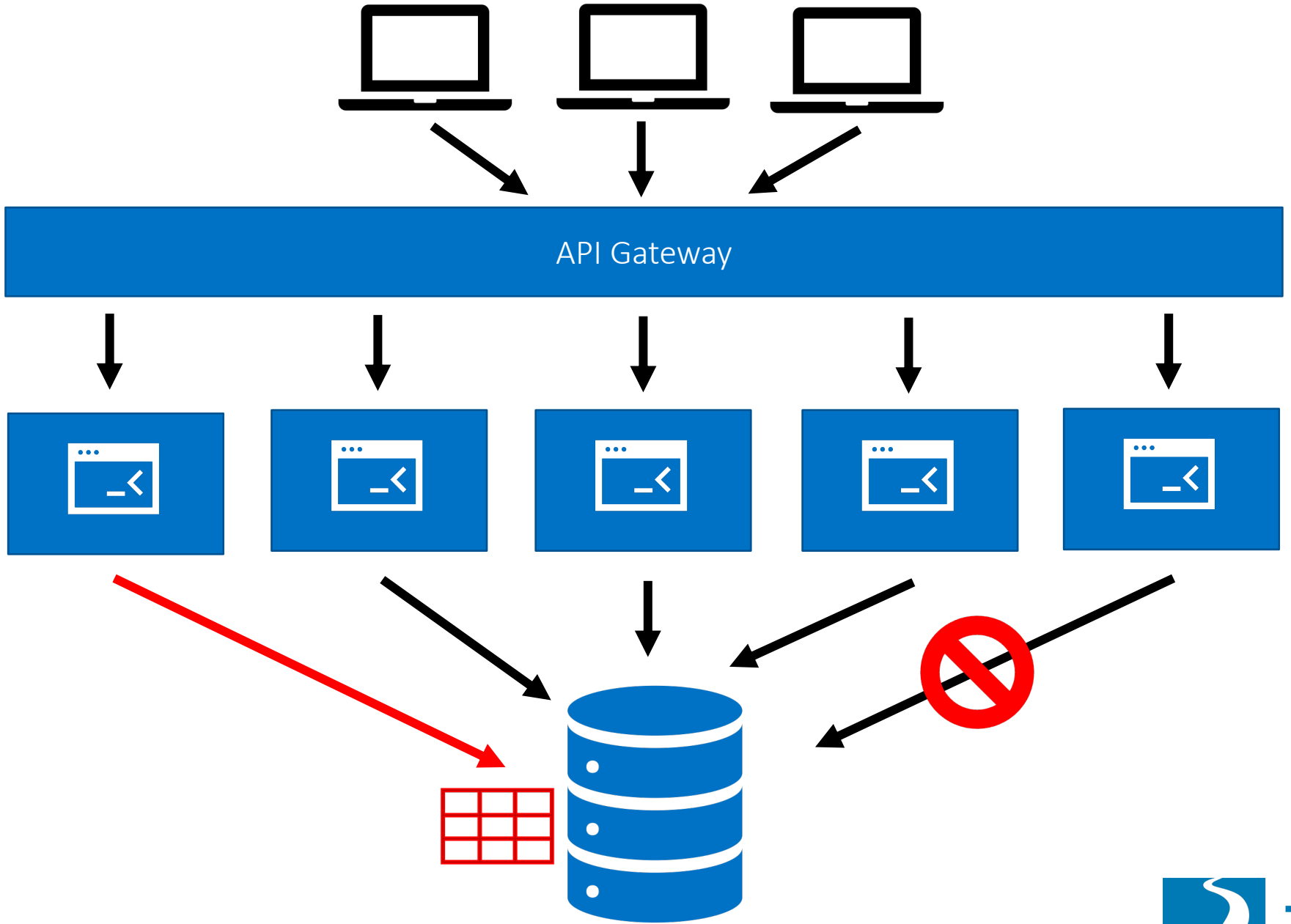
TRAILHEAD
TECHNOLOGY PARTNERS



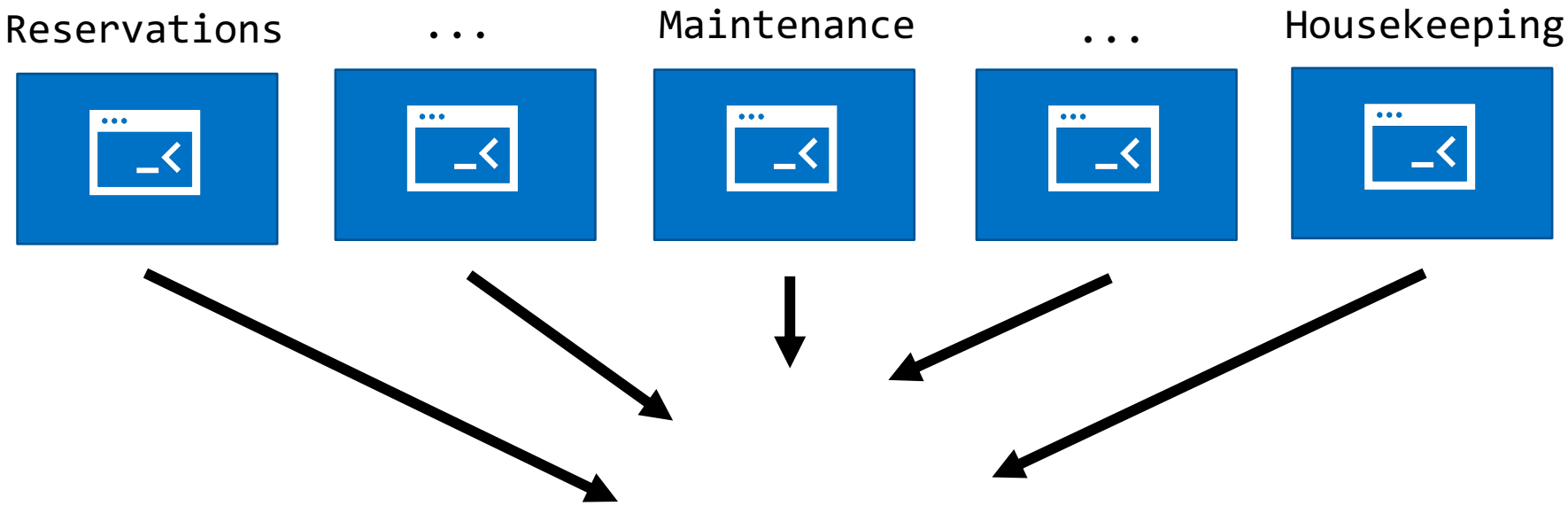
TRAILHEAD
TECHNOLOGY PARTNERS



TRAILHEAD
TECHNOLOGY PARTNERS

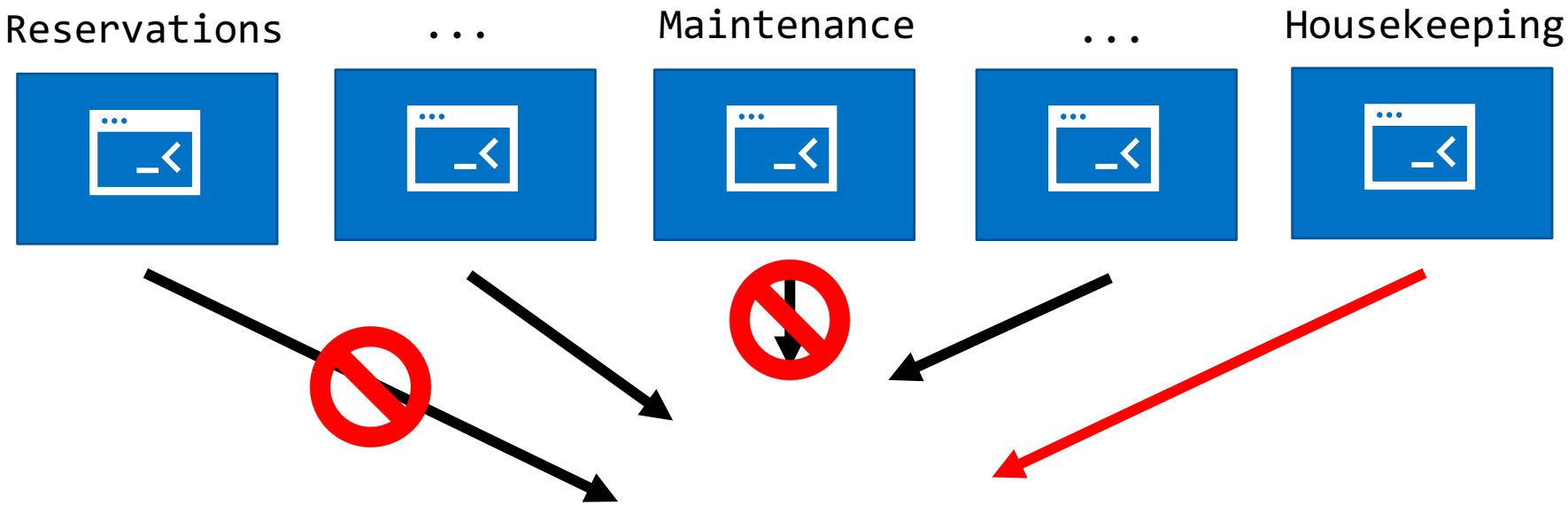


TRAILHEAD
TECHNOLOGY PARTNERS



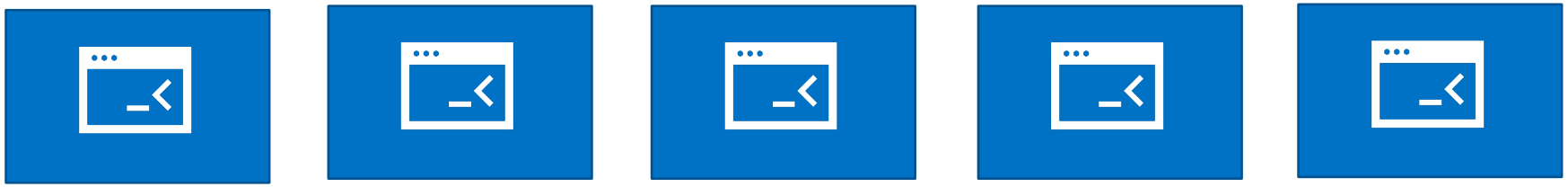

```
HotelRoom {
  CurrentlyOccupied : bool
  NumberofBeds : int
  MinutesToClean : float
  RepairHistory: [
    { RepairDate: ... },
    { RepairDate: ... },
  ]
}
```

TRAILHEAD
TECHNOLOGY PARTNERS




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

TRAILHEAD
TECHNOLOGY PARTNERS

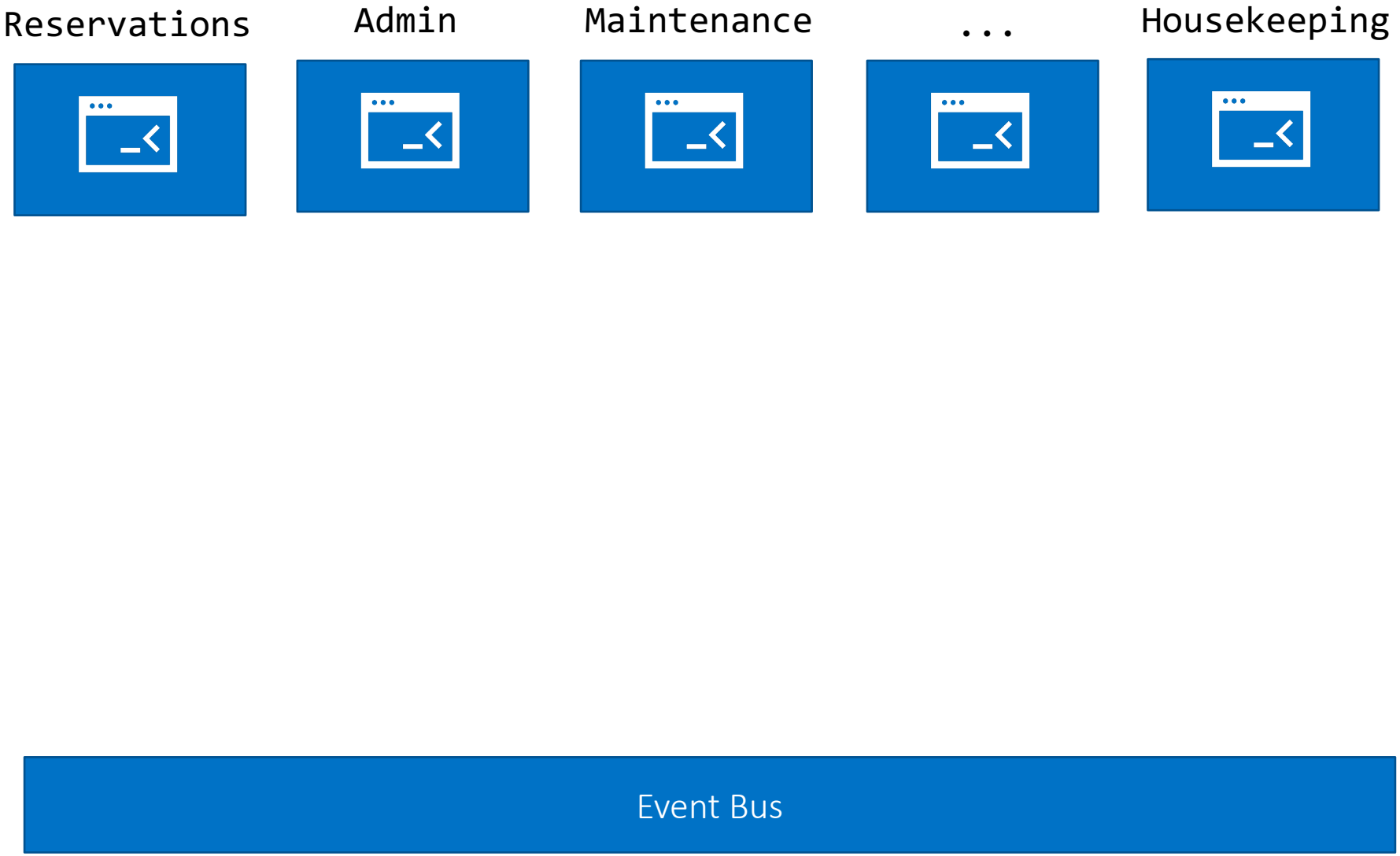



```
HotelRoom {
  CurrentlyOccupied : bool
  NumberofBeds : int
}
```

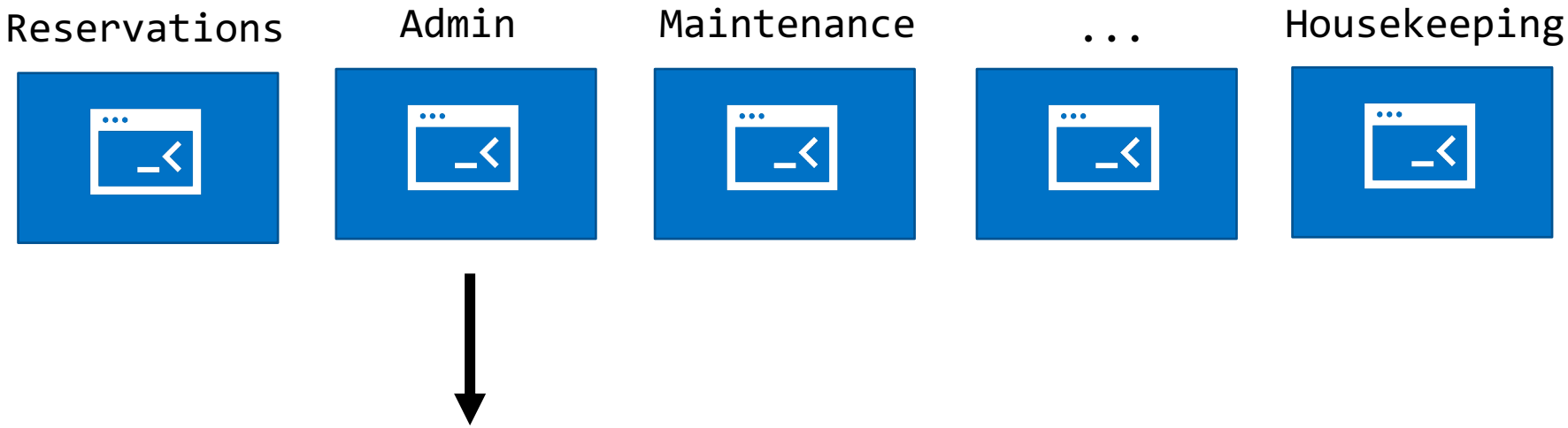
```
HotelRoom {
  RepairHistory: [
    { RepairDate: ... },
    { RepairDate: ... },
  ]
}
```

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

TRAILHEAD
TECHNOLOGY PARTNERS



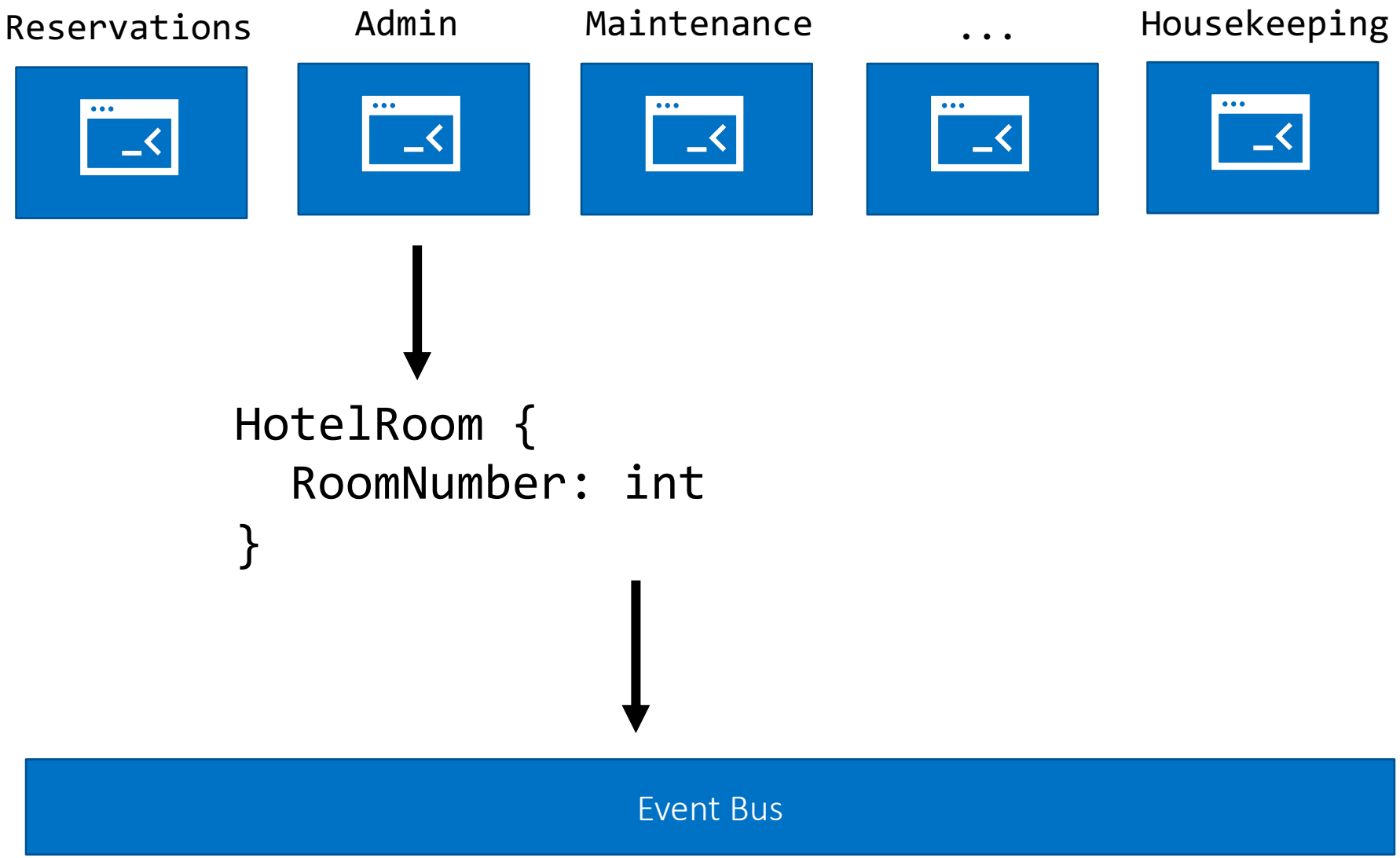
TRAILHEAD
TECHNOLOGY PARTNERS



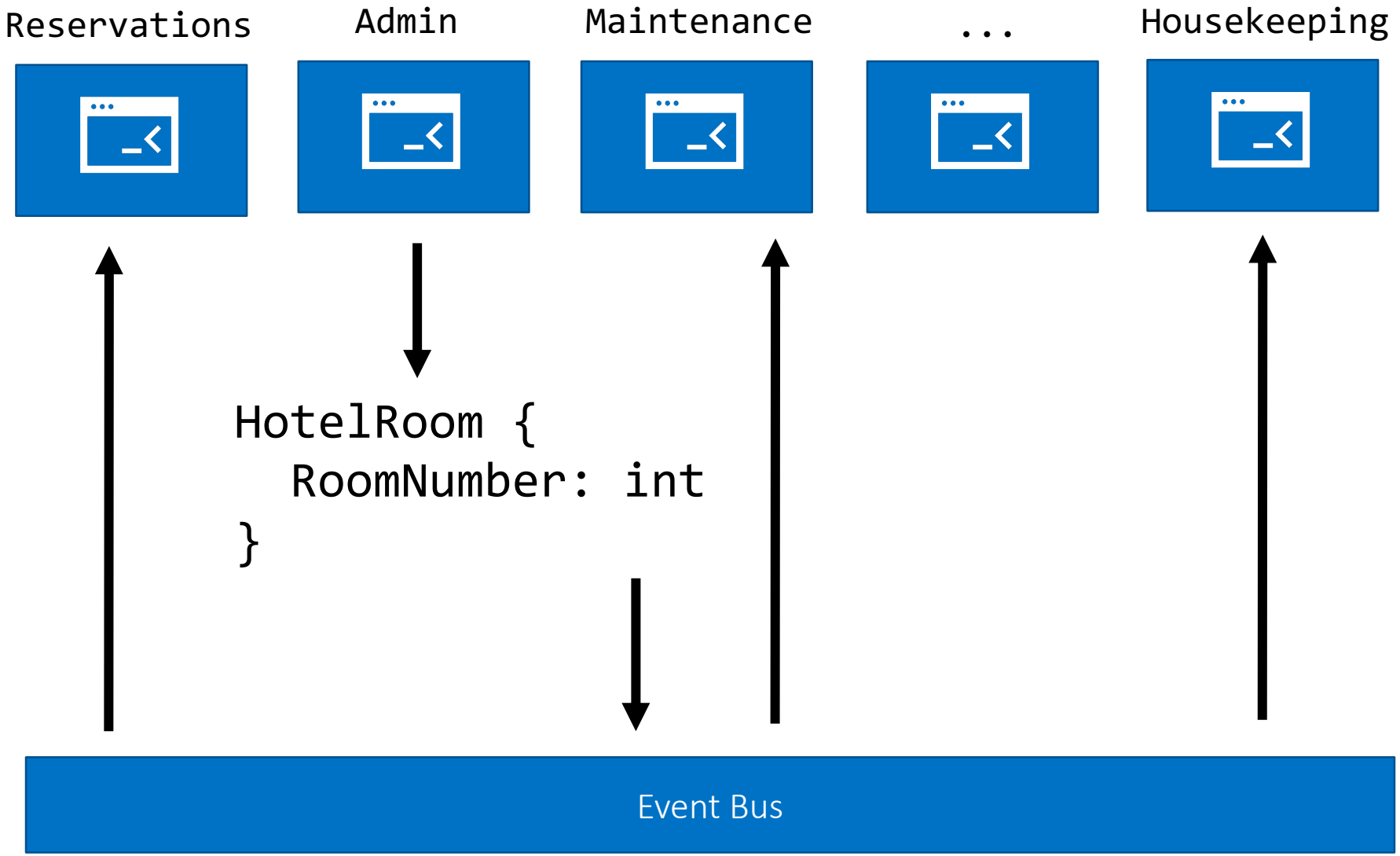

```
HotelRoom {
  RoomNumber: int
}
```

Event Bus

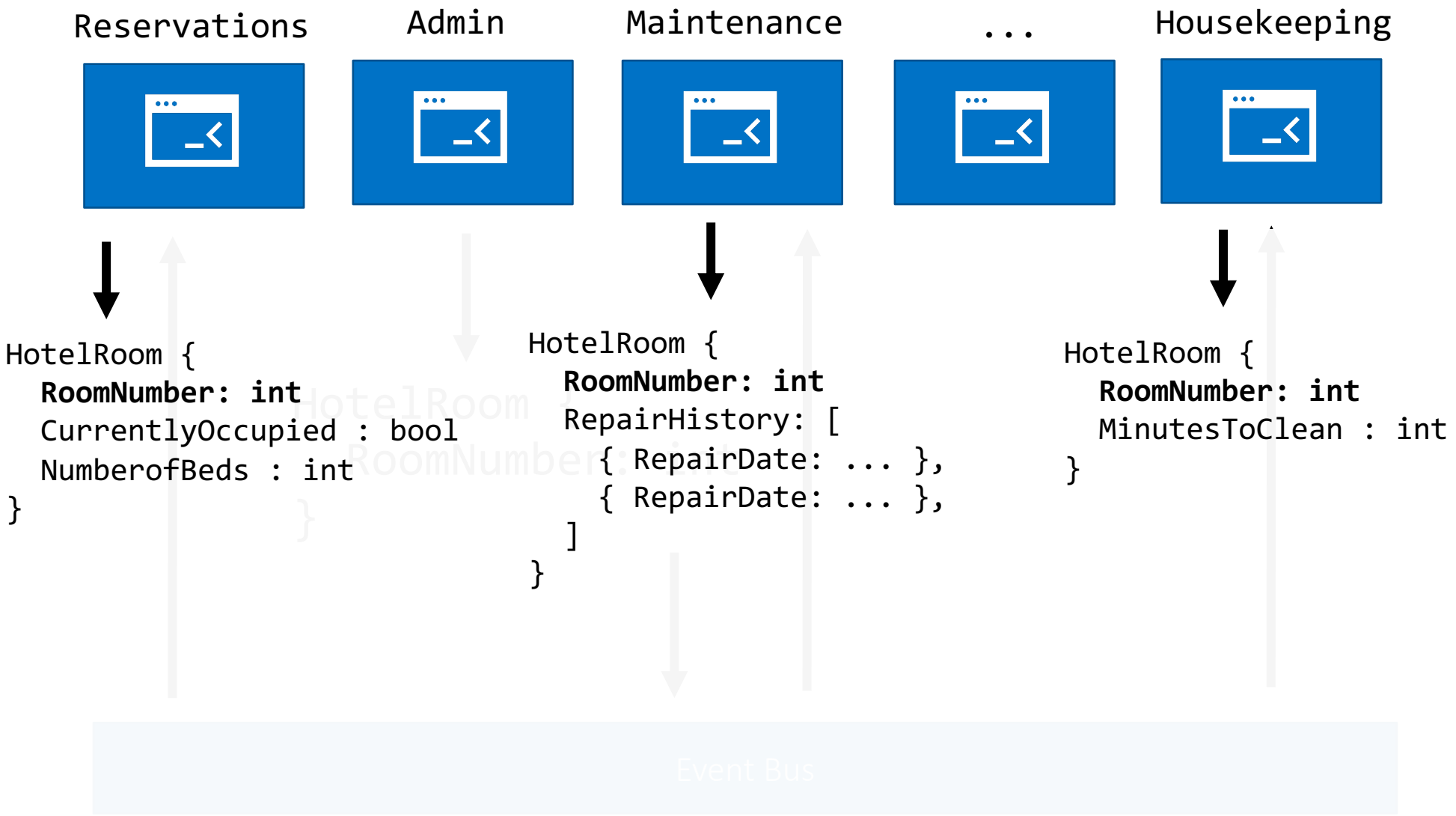
TRAILHEAD
TECHNOLOGY PARTNERS



TRAILHEAD
TECHNOLOGY PARTNERS



TRAILHEAD
TECHNOLOGY PARTNERS



TRAILHEAD
TECHNOLOGY PARTNERS

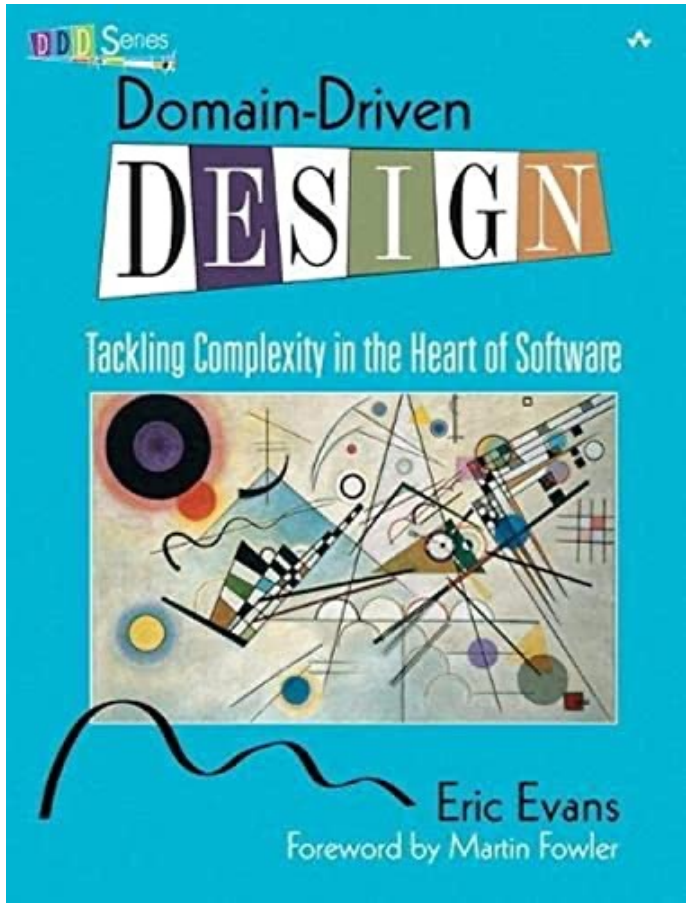
Microservices That Are Too Big

Problem #3

Domain Driven Development

Domain
 Subdomain
  Bounded Context

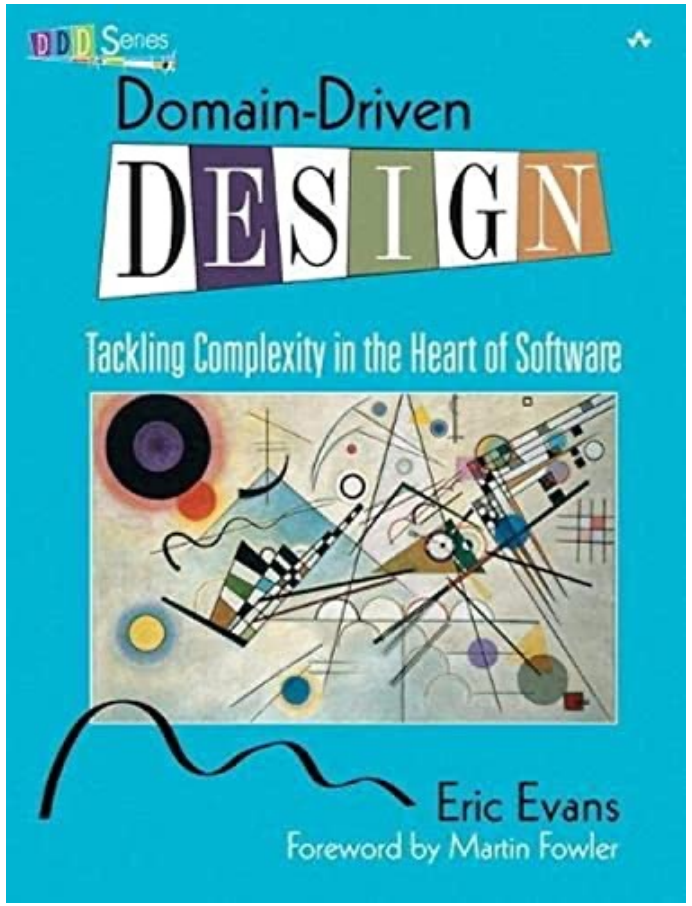
Domain Driven Development



Domain
Subdomain
Bounded Context

TRAILHEAD
TECHNOLOGY PARTNERS

Domain Driven Development



Domain
Subdomain
Bounded Context

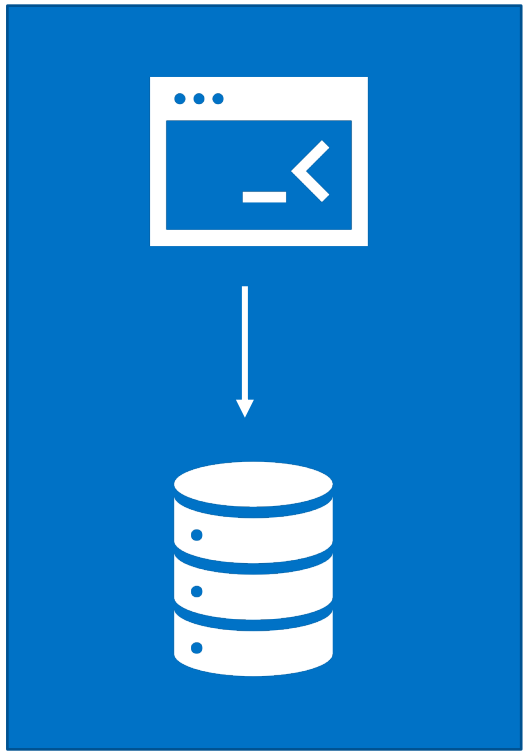
J's SIMPLE RULE: smallest possible microservices without chatty communication between services

TRAILHEAD
TECHNOLOGY PARTNERS

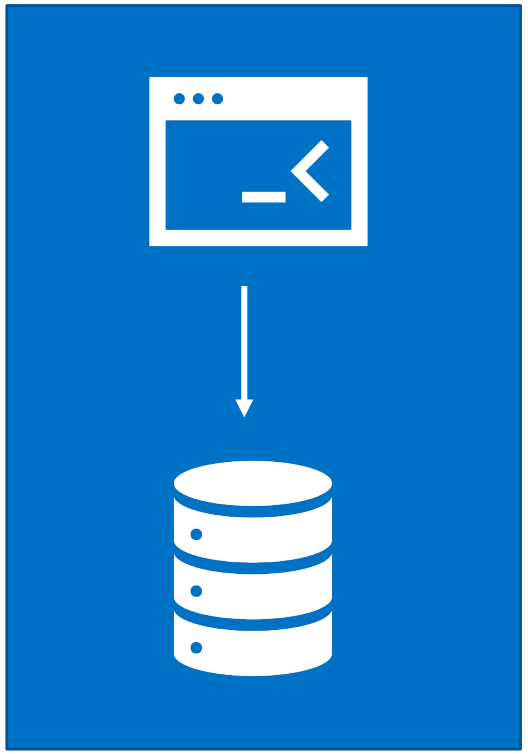
Microservices That Are Too Small

Problem #4

User Log In

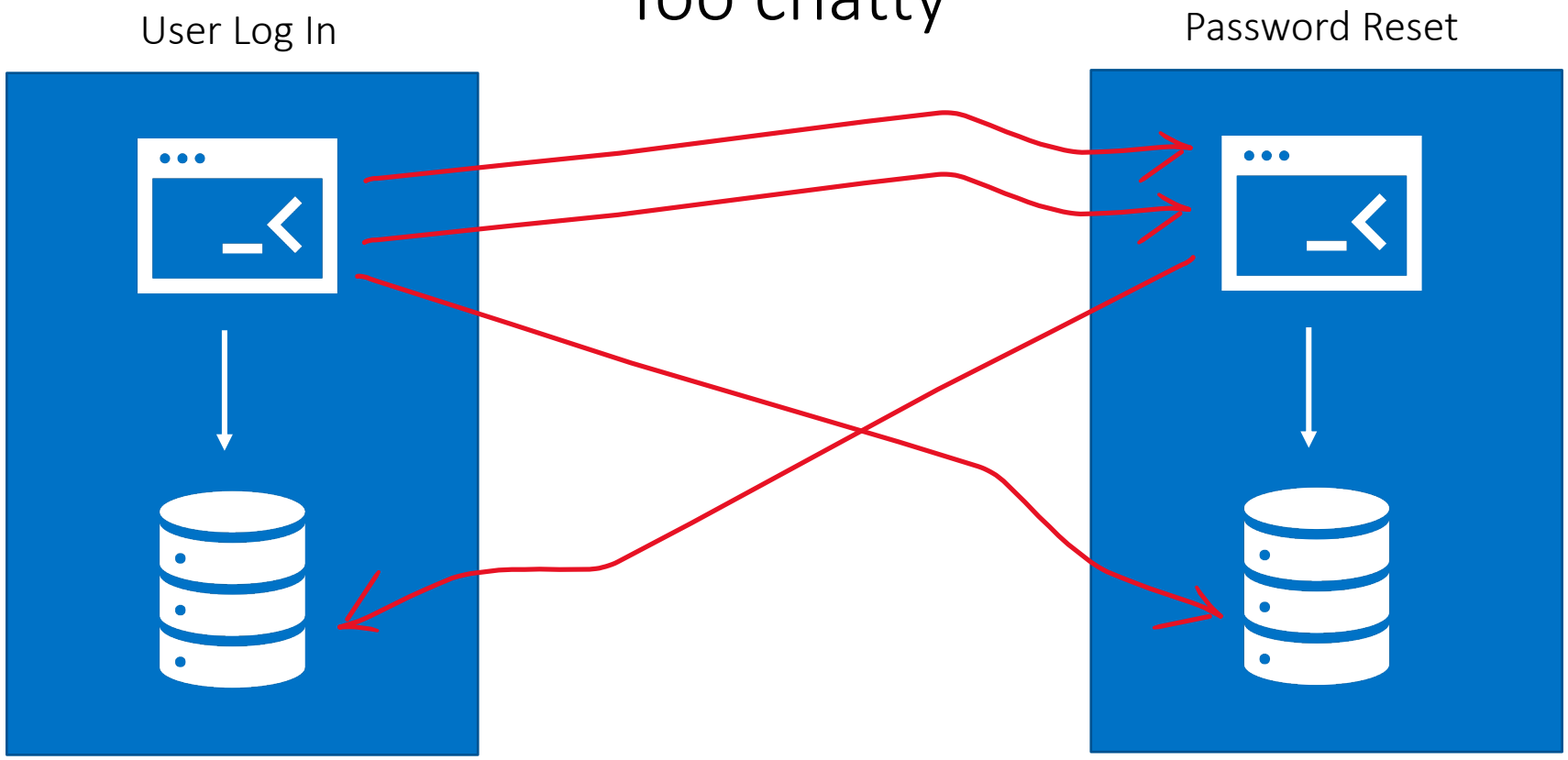


Password Reset



TRAILHEAD
TECHNOLOGY PARTNERS

Too chatty



TRAILHEAD
TECHNOLOGY PARTNERS

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

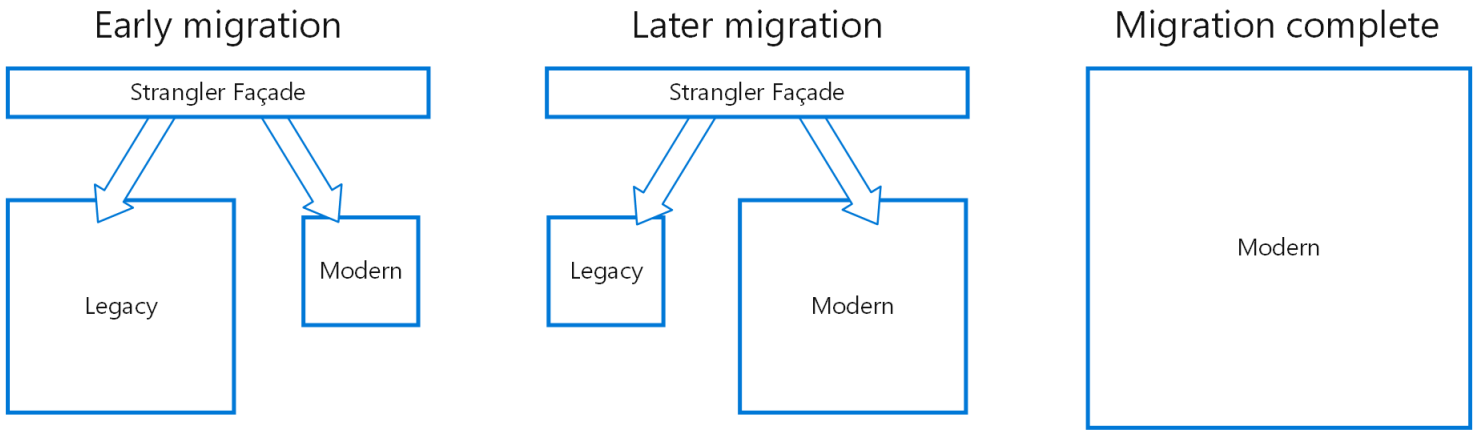
Big bang

Evolution

“Strangler fig” pattern

TRAILHEAD
TECHNOLOGY PARTNERS

Strangler Fig Pattern



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

TRAILHEAD
TECHNOLOGY PARTNERS

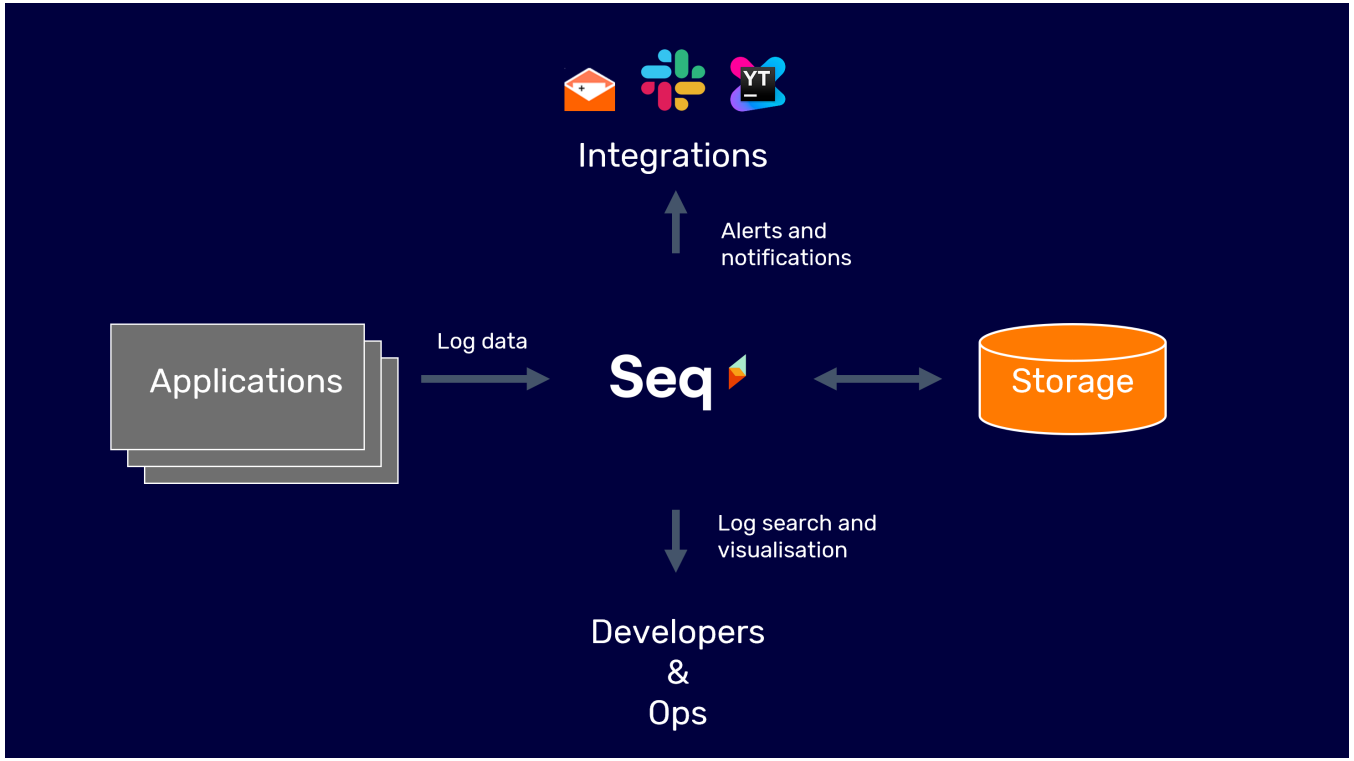
Coupling Through Cross-Cutting Concerns

Problem #6

TRAILHEAD
TECHNOLOGY PARTNERS

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

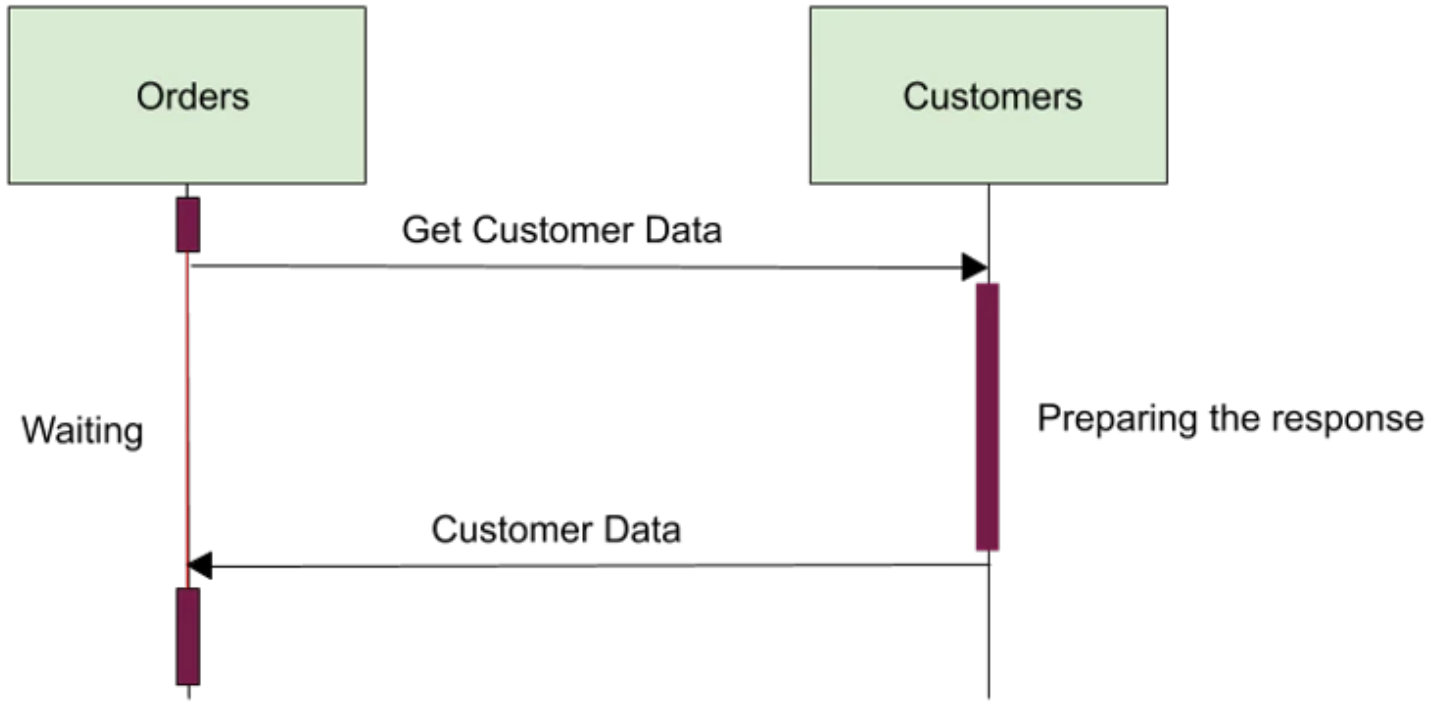


TRAILHEAD TECHNOLOGY PARTNERS

Use of Synchronous Communication

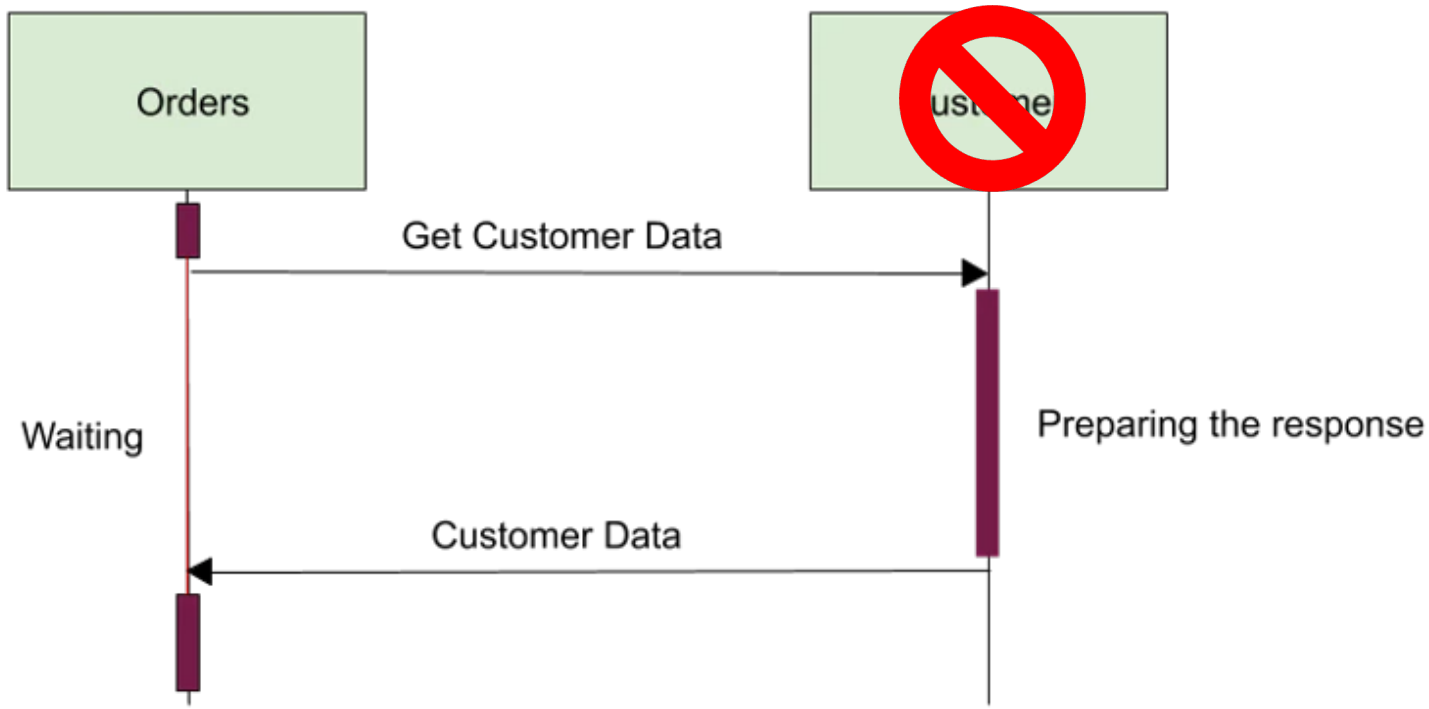
Problem #7

TRAILHEAD
TECHNOLOGY PARTNERS



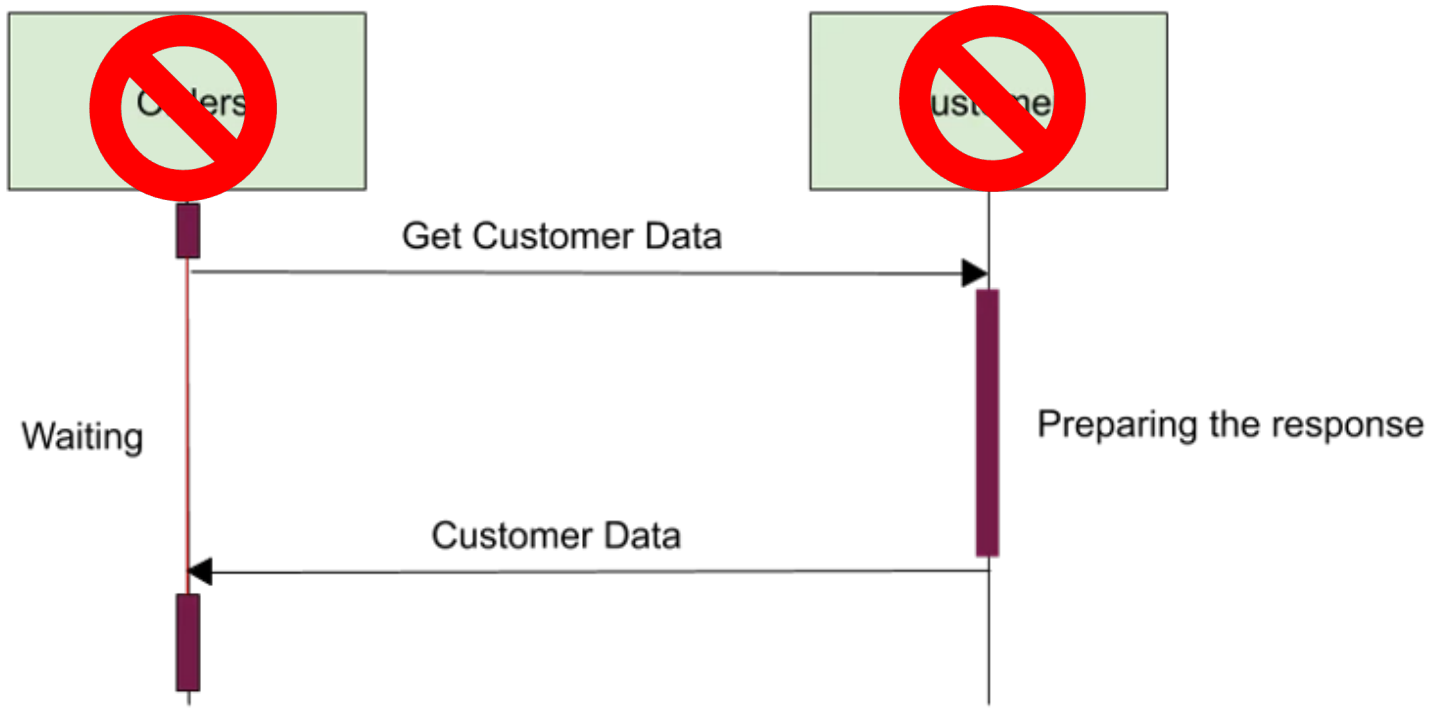
Source: https://www.capitalone.com/tech/software-engineering/how-to-avoid-loose-coupled-microservices/

TRAILHEAD
TECHNOLOGY PARTNERS



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

TRAILHEAD
TECHNOLOGY PARTNERS



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

TRAILHEAD
TECHNOLOGY PARTNERS

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

TRAILHEAD
TECHNOLOGY PARTNERS

Unencapsulated Services

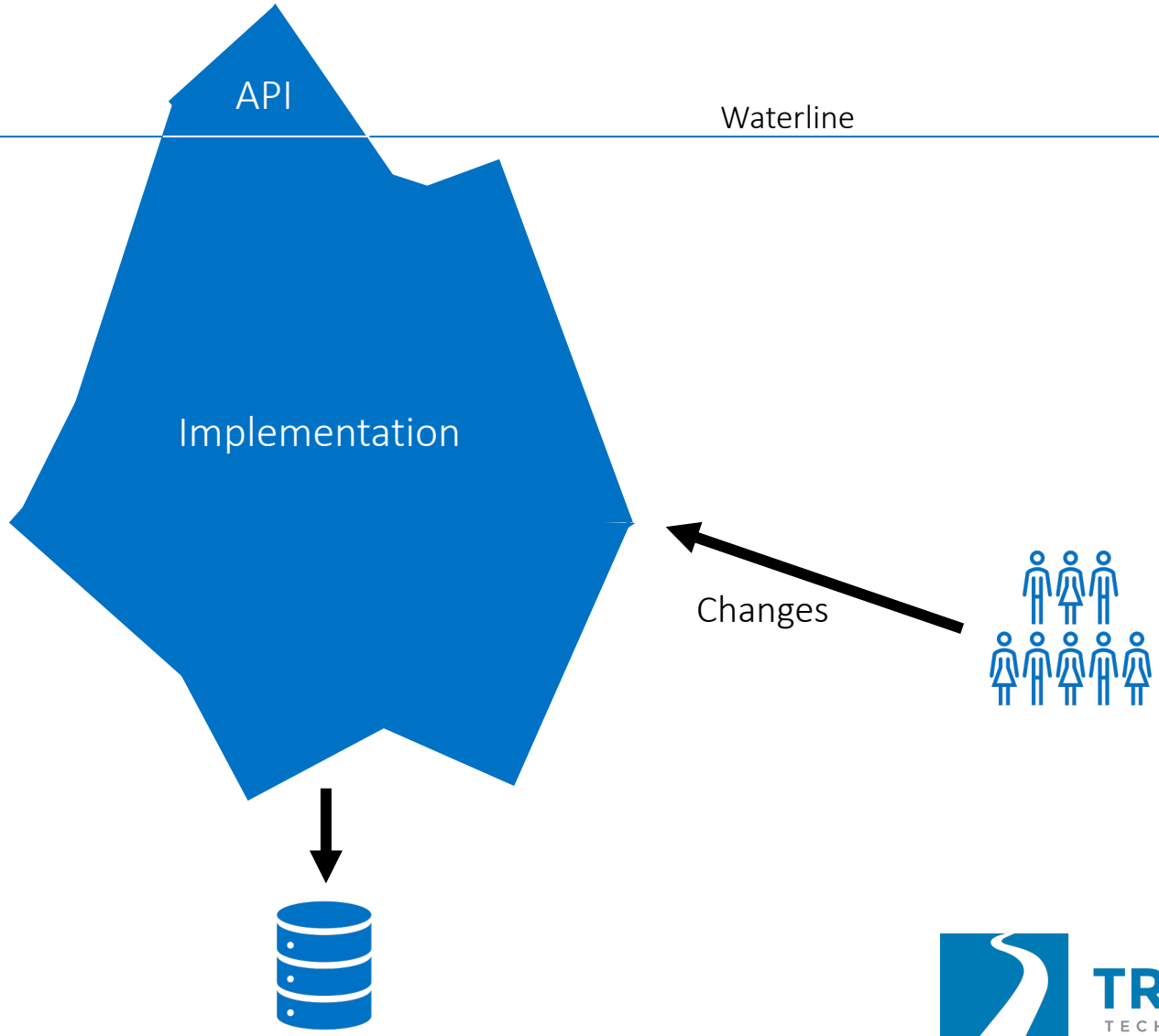
Problem #10

Iceberg Services

Services ENCAPSULATE significant business logic

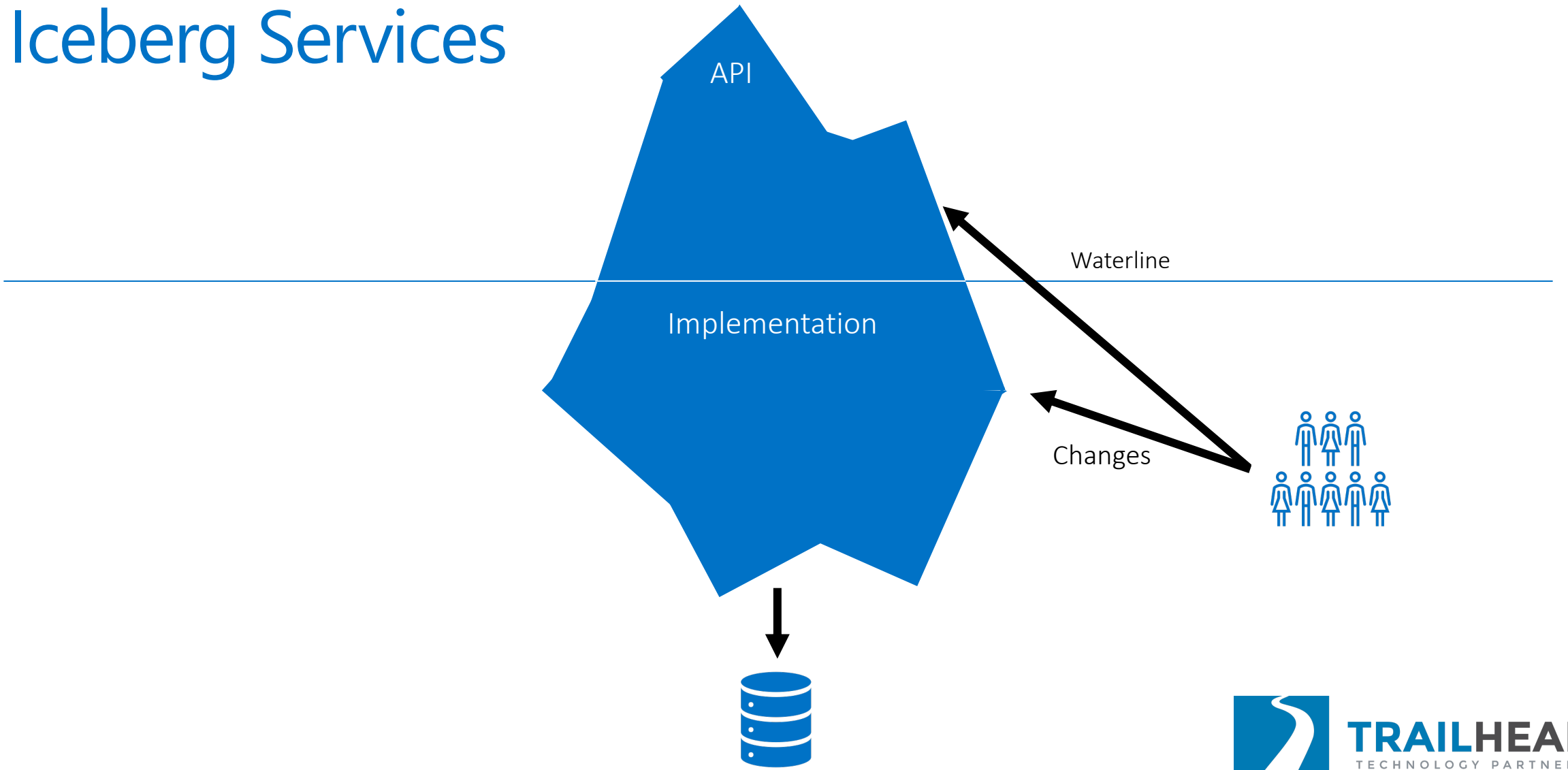
Small, stable API

Large implementation



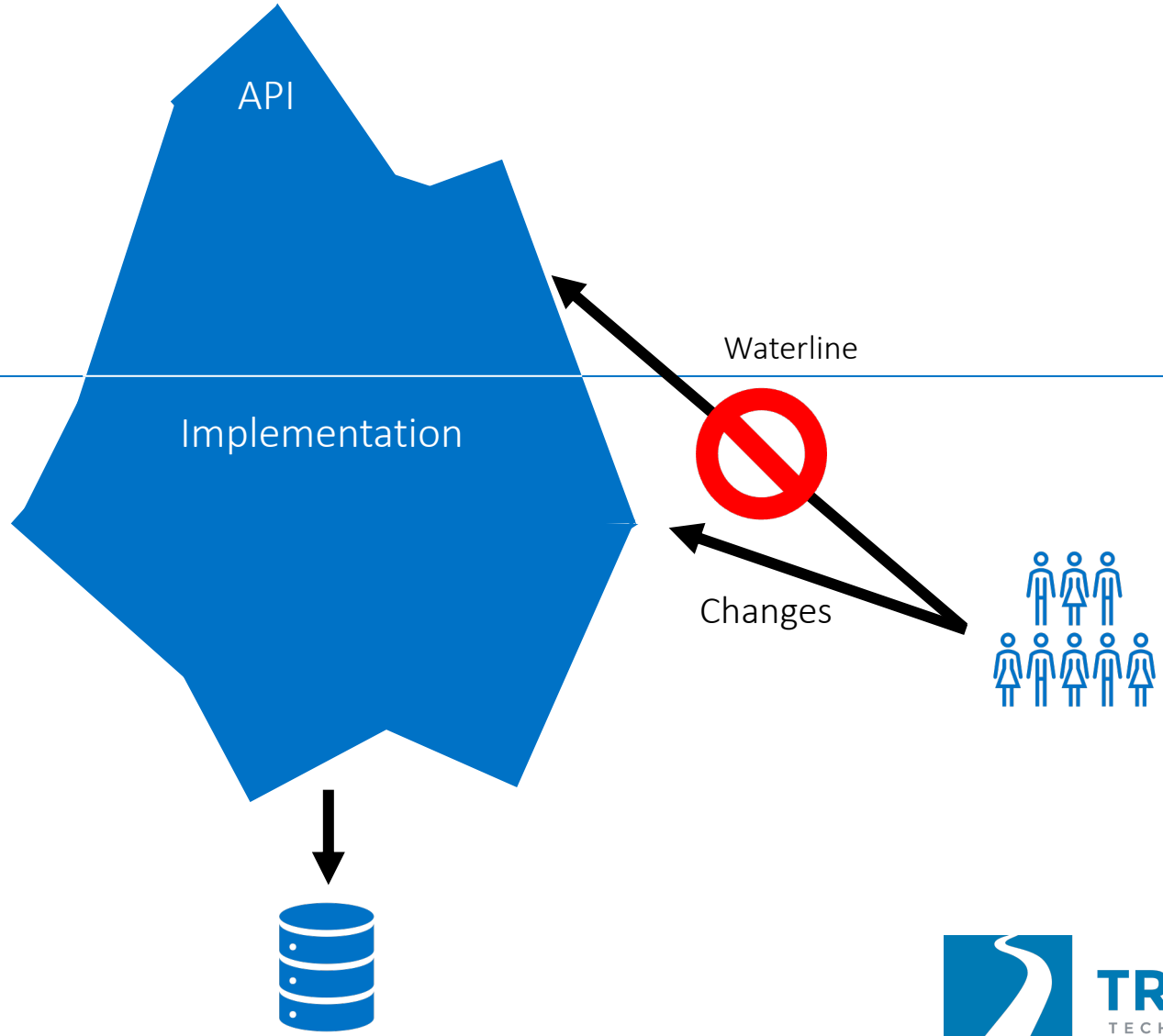
TRAILHEAD
TECHNOLOGY PARTNERS

Iceberg Services



TRAILHEAD
TECHNOLOGY PARTNERS

Iceberg Services



TRAILHEAD
TECHNOLOGY PARTNERS

Mismatched Teams

BONUS PROBLEM

Conway's Law

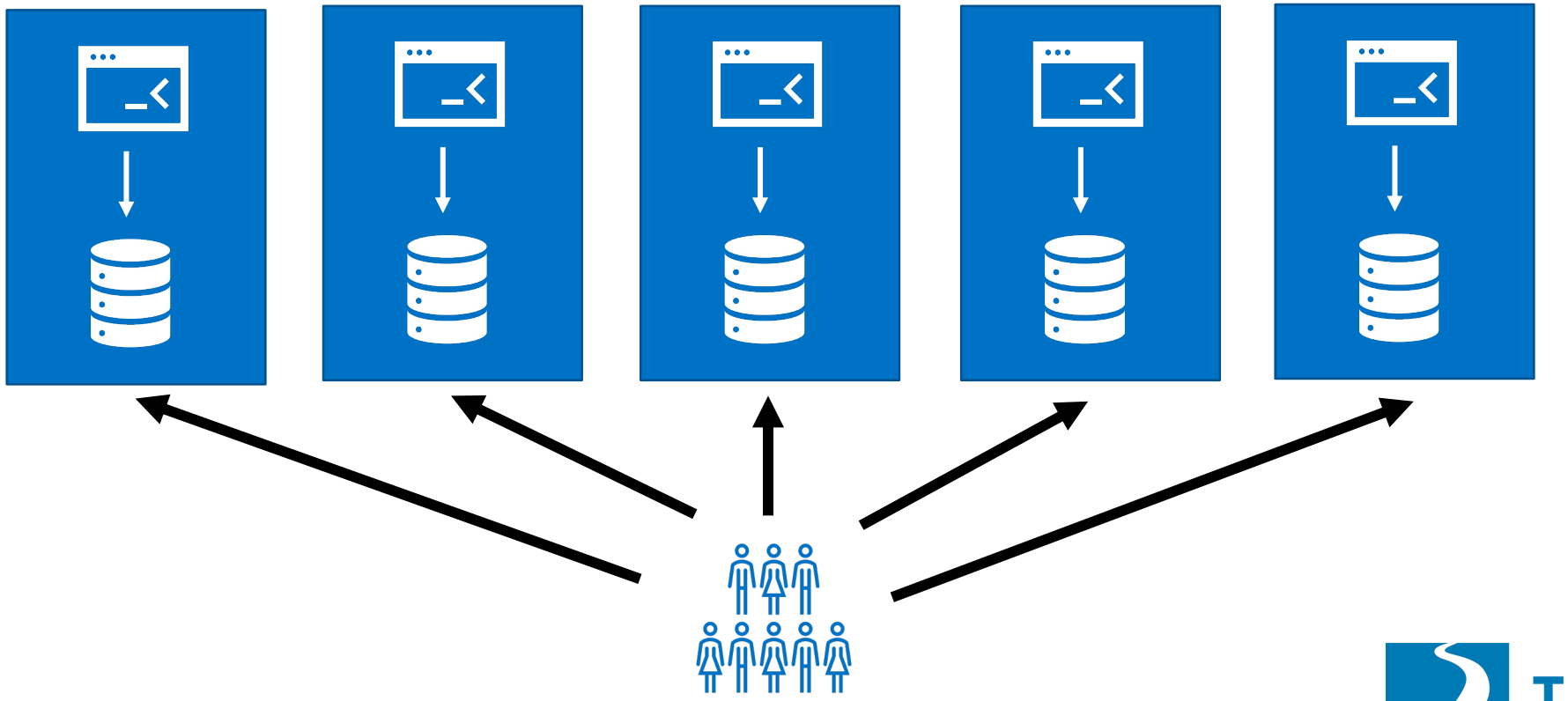
"Any organization that designs a system (defined broadly) will produce a design whose **structure** is a **copy** of the organization's **communication structure**."

- Melvin E. Conway

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

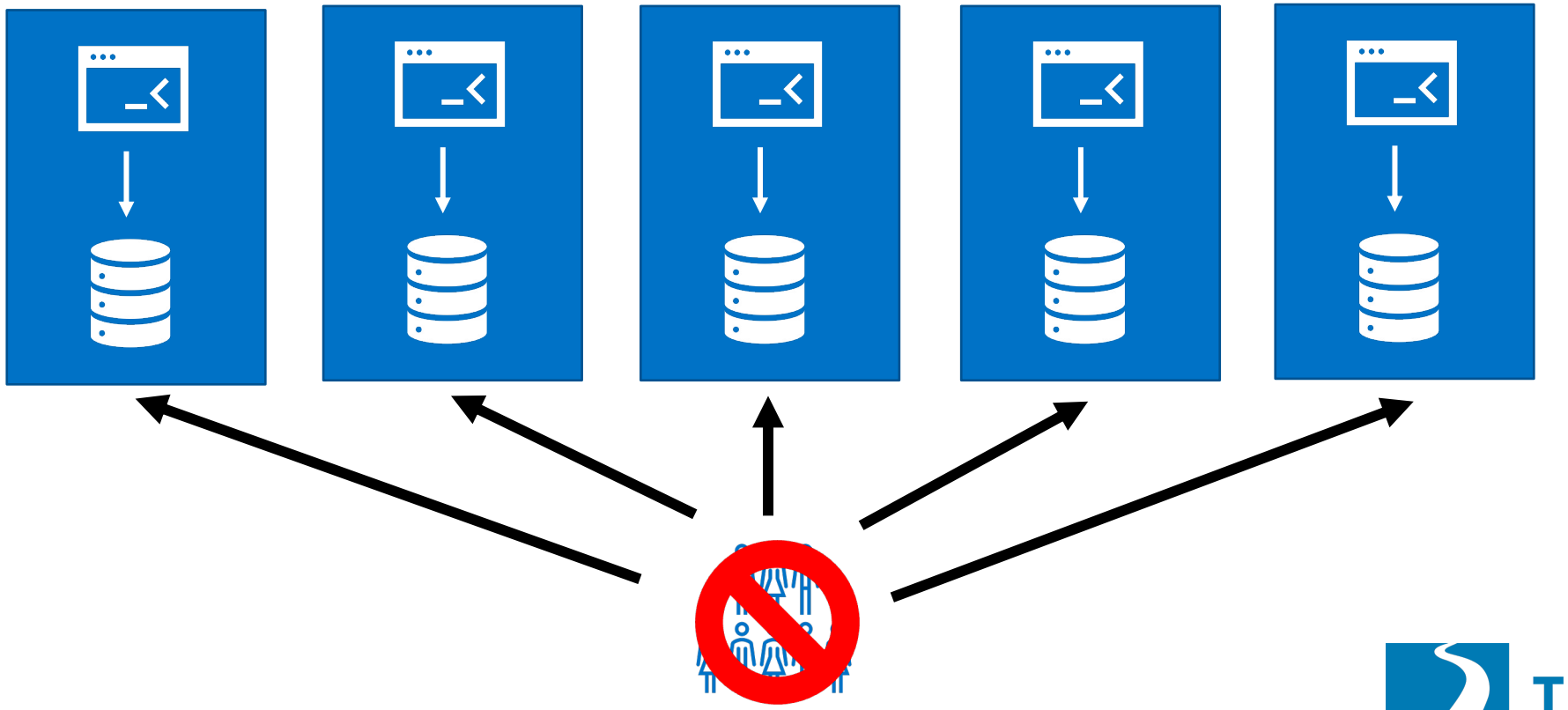
TRAILHEAD
TECHNOLOGY PARTNERS

Single Team



TRAILHEAD
TECHNOLOGY PARTNERS

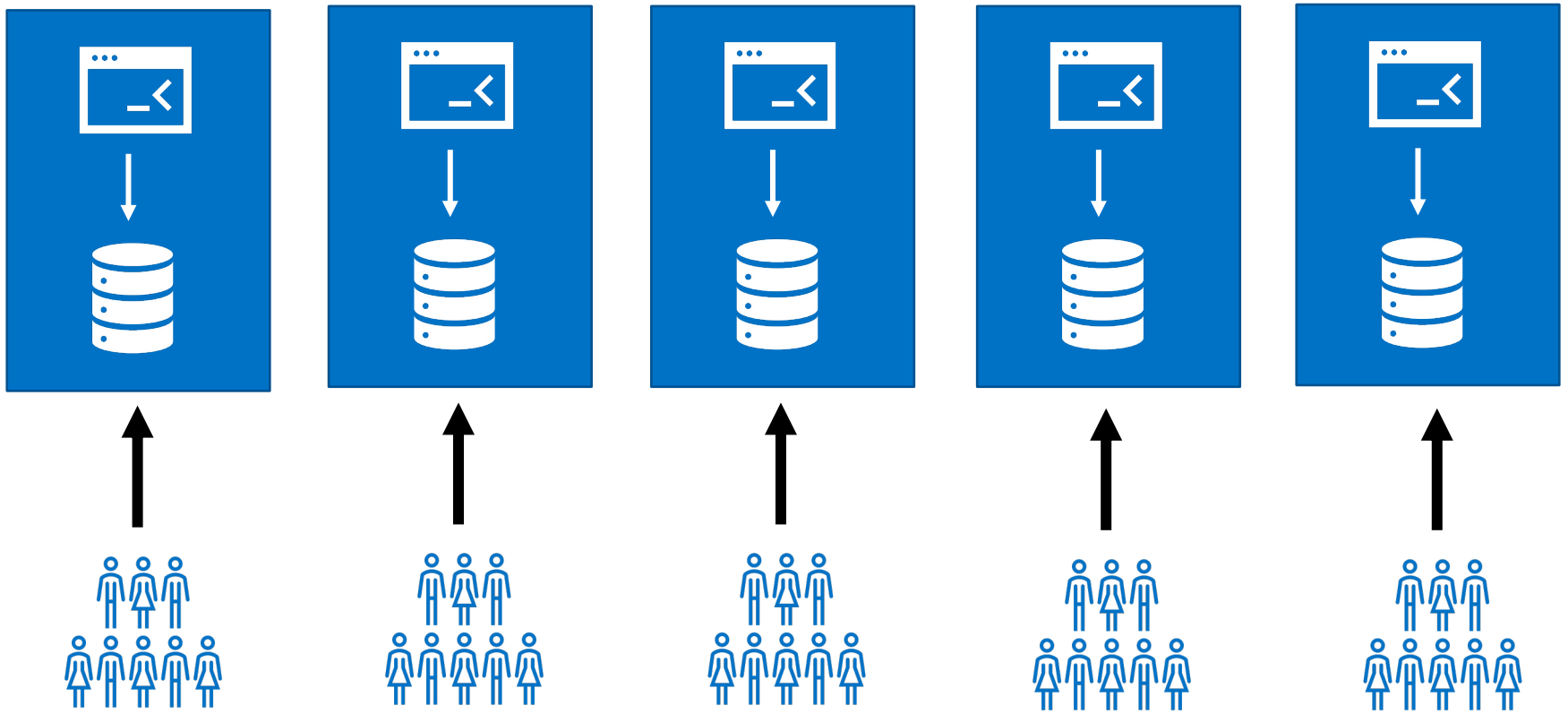
Single Team



TRAILHEAD
TECHNOLOGY PARTNERS

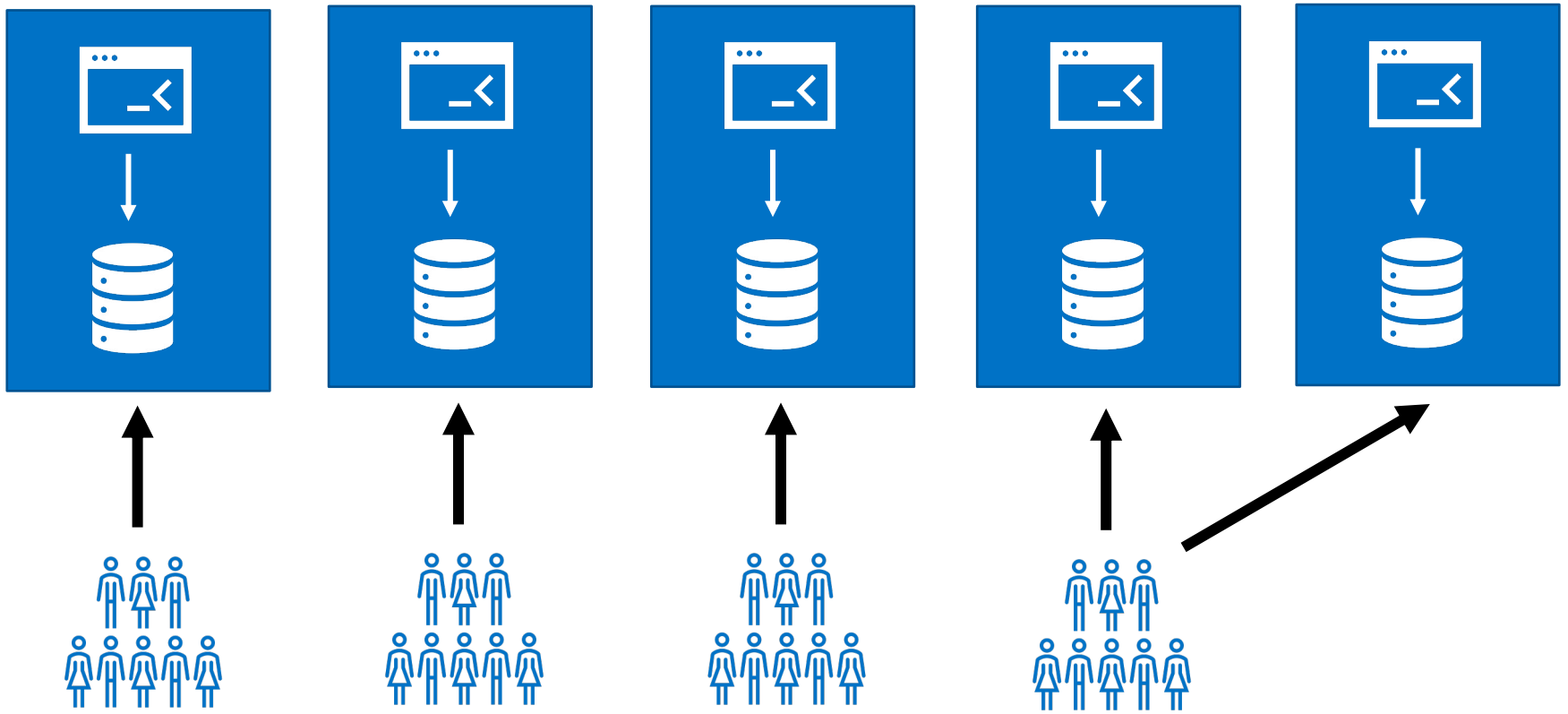
TRAILHEAD
TECHNOLOGY PARTNERS

Team Per Service



TRAILHEAD
TECHNOLOGY PARTNERS

Team Per Service



Summing Up

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

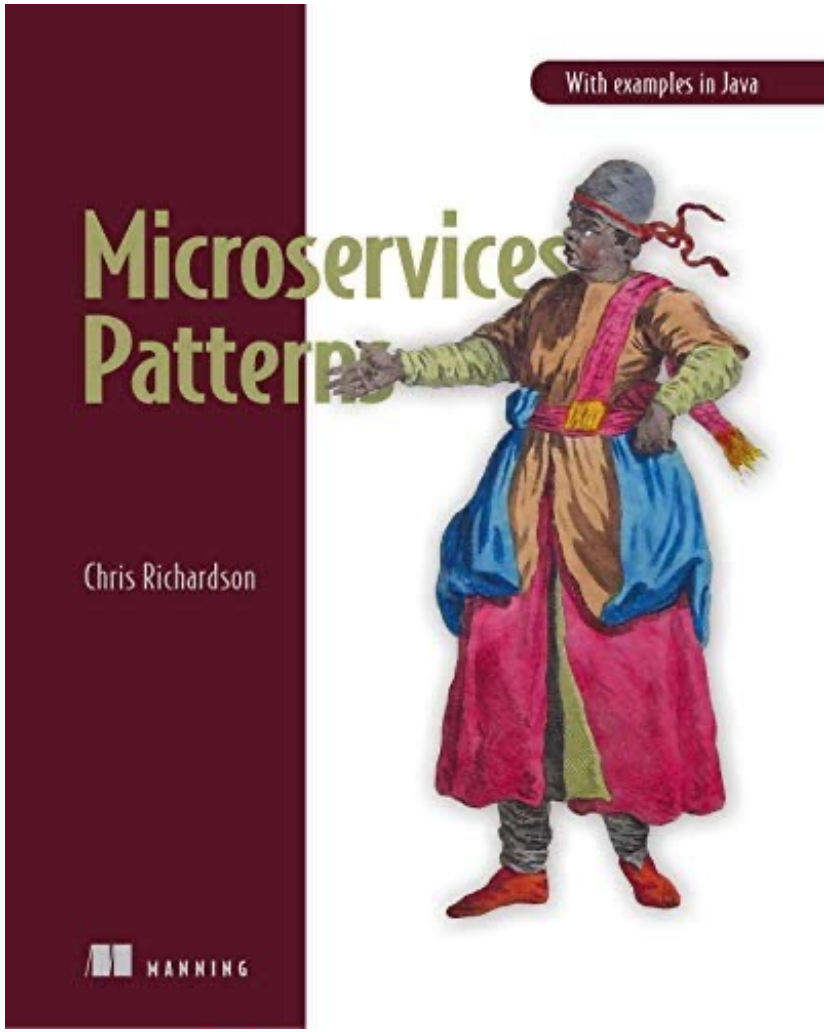
2. For a complex, evolving application with clear domains and separate teams, microservices will be best

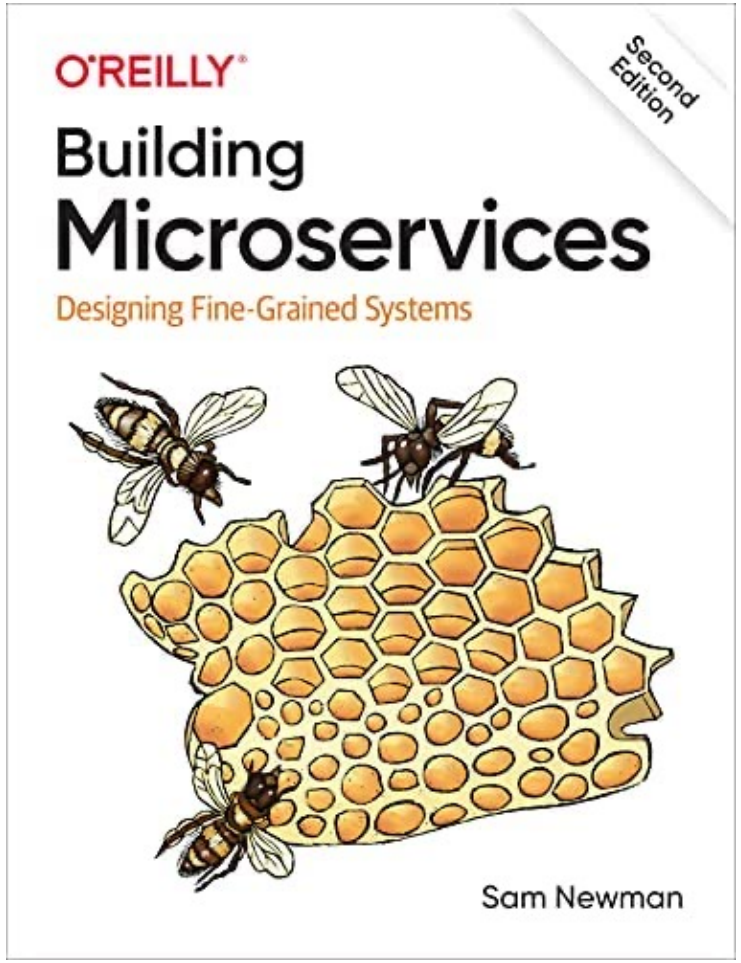
3. Don’t try microservices without “a really good reason”. Monoliths can be good!

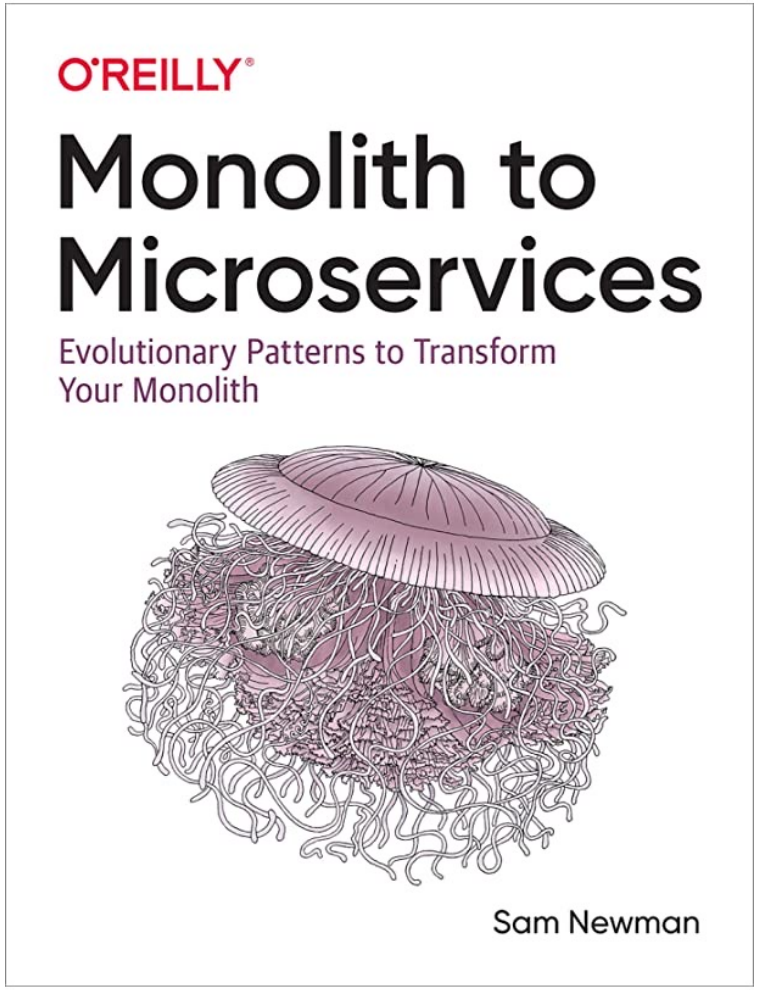
4. Avoid pitfalls of the distributed monolith

TRAILHEAD
TECHNOLOGY PARTNERS

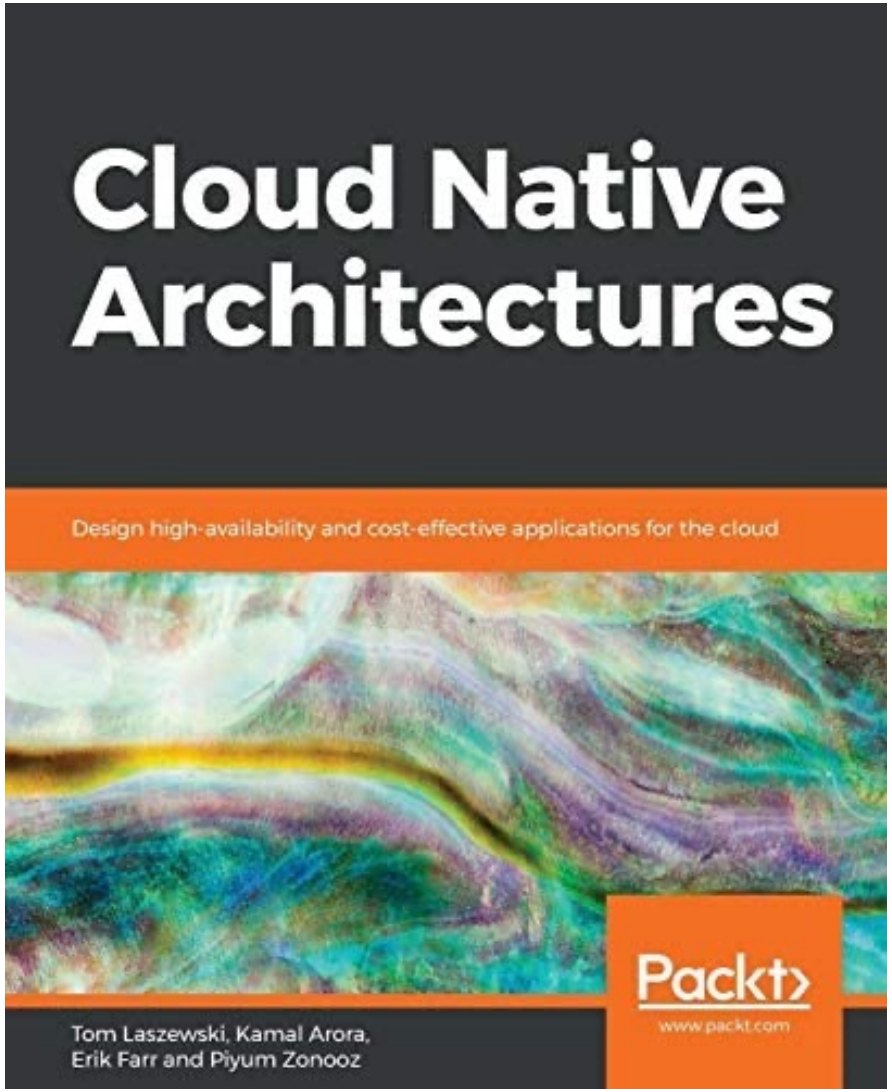
Further Reading

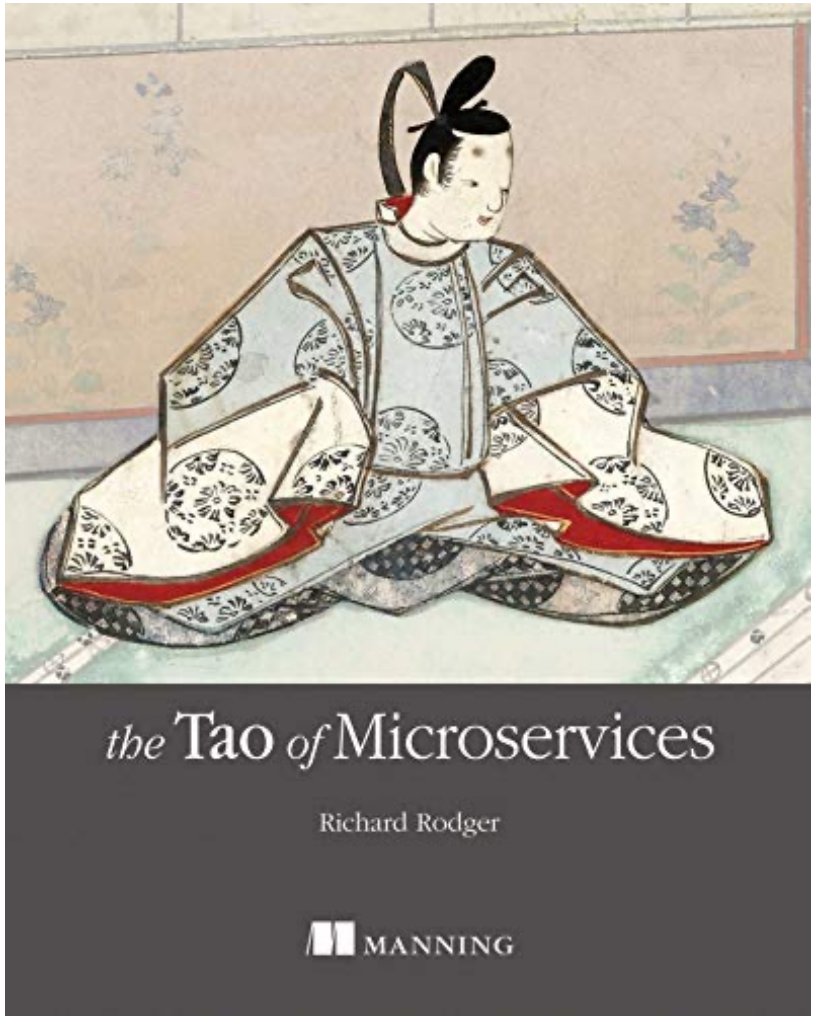






TRAILHEAD
TECHNOLOGY PARTNERS





TRAILHEAD
TECHNOLOGY PARTNERS

Recap

- Definitions
  - Monolith
  - Microservices
  - Distributed Monolith
- 10 Most Common Mistakes
- 1 Bonus Mistake
- Further Reading
- Q&A

Thanks! Questions?

Jonathan "J." Tower

jtower@trailheadtechnology.com

trailheadtechnology.com/blog

jtowermi

Free Offer

30-Minute Consultation

bit.ly/th-offer

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