

Introduction to Microservice Architecture

Wiwat Vatanawood

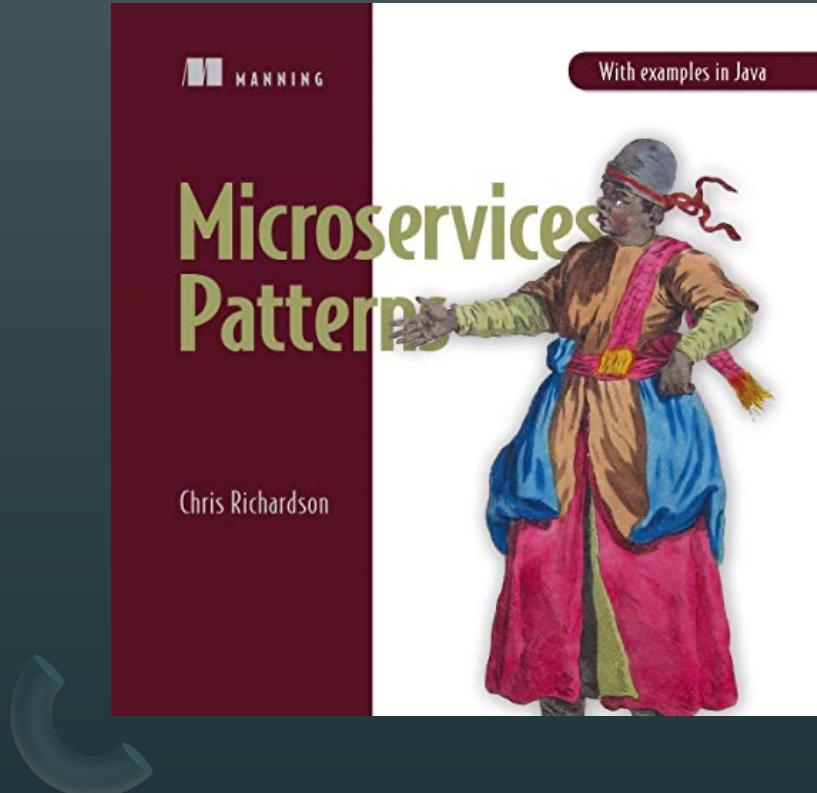
Duangdao Wichadakul

Nuengwong Tuaycharoen

Pittipol Kantavat



References



- Book
 - Chris Richardson, Microservices Patterns: With examples in Java, Manning, 2019
- Website
 - <https://microservices.io/>

The Chapter covers

- The symptoms of monolithic hell and how to escape it by adopting the microservice architecture
- The essential characteristics of the microservice architecture and its benefits and drawbacks
- How microservices enable the DevOps style of development of large, complex applications
- The microservice architecture pattern language and why you should use it

Agenda

- **A brief refresher on software architecture**
- From monolith to microservices
- Microservices != silver bullet
- Applying the microservice pattern language



What is software architecture?

“The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.”



Documenting Software Architectures, Bass et. al.

What is software architecture?

Architecture is multi-dimensional

e.g. Structural, electrical, plumbing, mechanical



Described by multiple views

View = (elements, relations, properties)



Why architecture matters?

- Enables an application to satisfy the second category of requirements: its **quality-of-service requirements**
- Non-functional requirements
- Also known as quality attributes and are the so-called –ilities



- Maintainability
- Testability
- Deployability

Development velocity

- Evolvability
- Scalability
- Security
- Reliability
- ...



https://en.wikipedia.org/wiki/Non-functional_requirement

Businesses must innovate faster



Build better software faster

- **Reducing lead time**
- **Increasing deployment frequency**

Modern software development: moving fast and not breaking things!

2017 State of DevOps Report | presented by Puppet + DORA



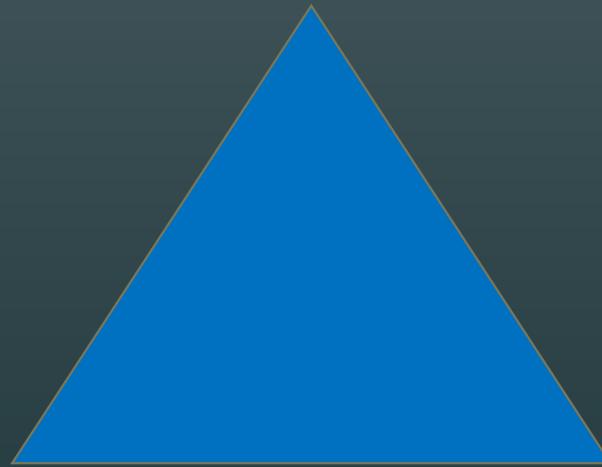
Table 2: 2017 IT performance by cluster

Survey questions	High IT performers	Medium IT performers	Low IT performers
Deployment frequency <i>For the primary application or service you work on, how often does your organization deploy code?</i>	On demand (multiple deploys per day) 46x	Between once per week and once per month	Between once per week and once per month*
Lead time for changes <i>For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?</i>	Less than one hour 440x Netflix: 16 minutes	Between one week and one month	Between one week and one month*
Mean time to recover (MTTR) <i>For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?</i>	Less than one hour 24x	Less than one day	Between one day and one week
Change failure rate <i>For the primary application or service you work on, what percentage of changes results either in degraded service or subsequently requires remediation (e.g., leads to service impairment, service outage, requires a hotfix, rollback, fix forward, patch)?</i>	0-15% 5x lower Amazon: ~0.001%	0-15%	31-45%

Modern software development

Process:

DevOps/Continuous delivery/deployment



Organization:

Small, autonomous teams



Architecture:

??

Agenda

- A brief refresher on software architecture
- **From monolith to microservices**
- Microservices != silver bullet
- Applying the microservice pattern language

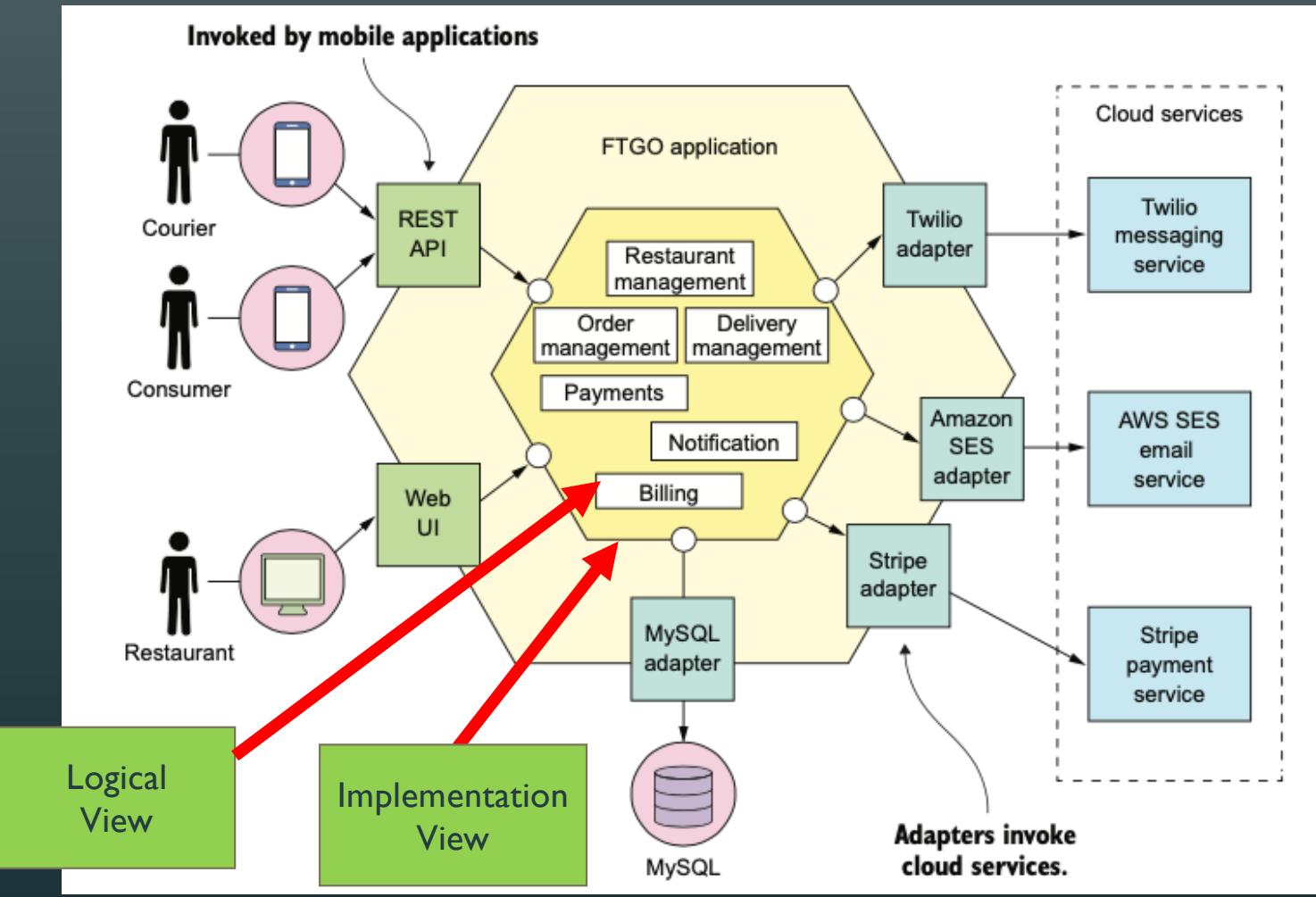


The monolithic architecture is an **architectural style** that structures the **application** as a **single executable component**



Implementation
View

Traditional: Monolithic architecture



-ilities of small monoliths

- Maintainability
- Testability
- Deployability
- ...



But successful applications keep growing



Development
Team

Application

... and growing

Development
Team A

Development
Team B

Development
Team C

Application





Eventually:
agile development
and deployment
becomes
impossible

=

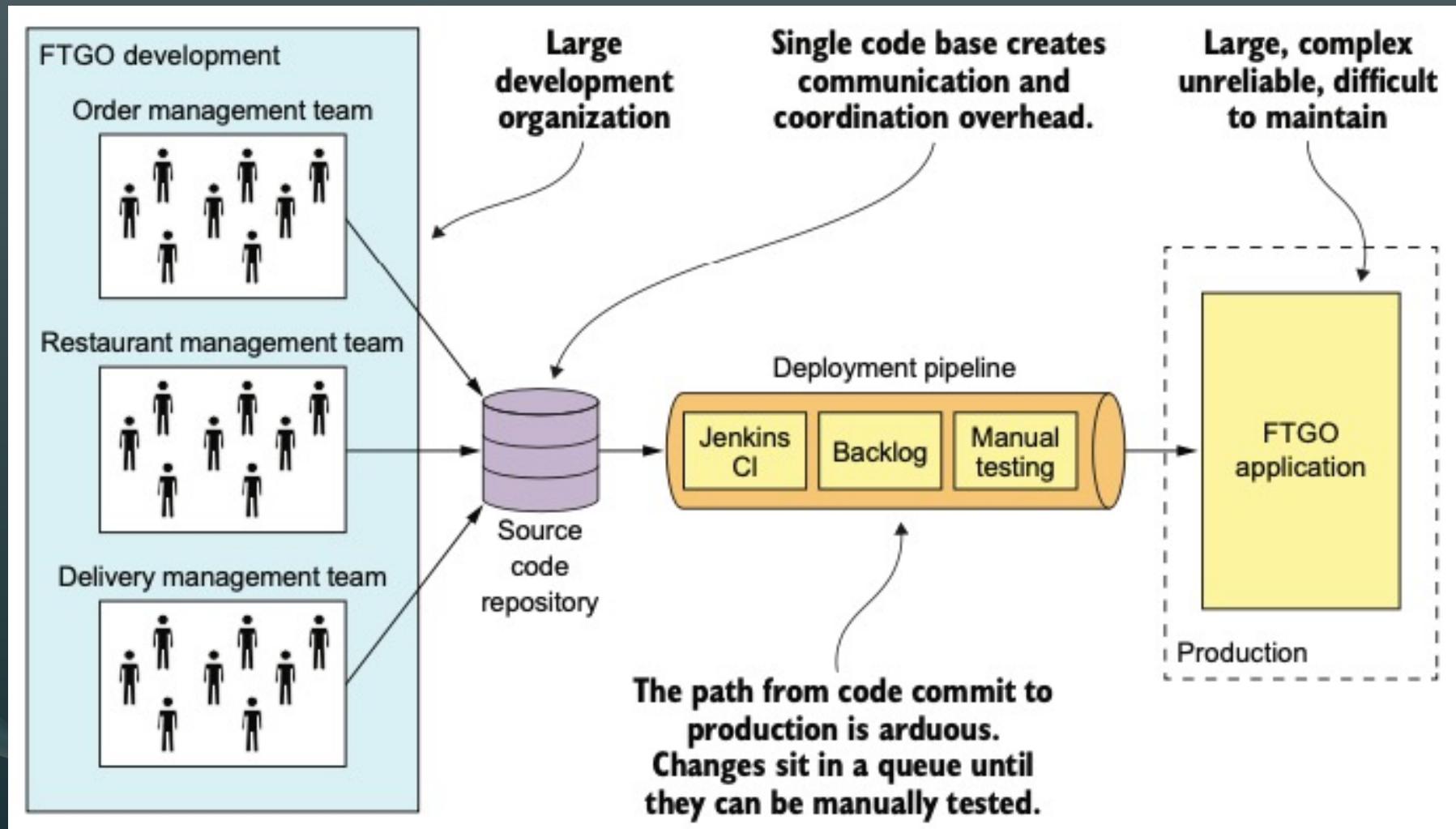
monolithic hell

-ilities of large monoliths

- Maintainability
- Testability
- Deployability
- ...



The slow march toward monolithic hell



Living in monolithic hell (1)

- Complexity intimidates developers
 - Fixing bugs and correctly implementing new features become difficult and time consuming
 - Deadlines are missed
- Development is slow
 - The large application overload and slow down developer's IDE
 - The application takes a long time to start up

Living in monolithic hell (2)

- Path from commit to deployment is long and arduous
 - The build is frequently in an unreleasable state
 - Painful merges
- Scaling is difficult
 - Data consume a lot of memory
 - CPU is intensively used



Living in monolithic hell (3)

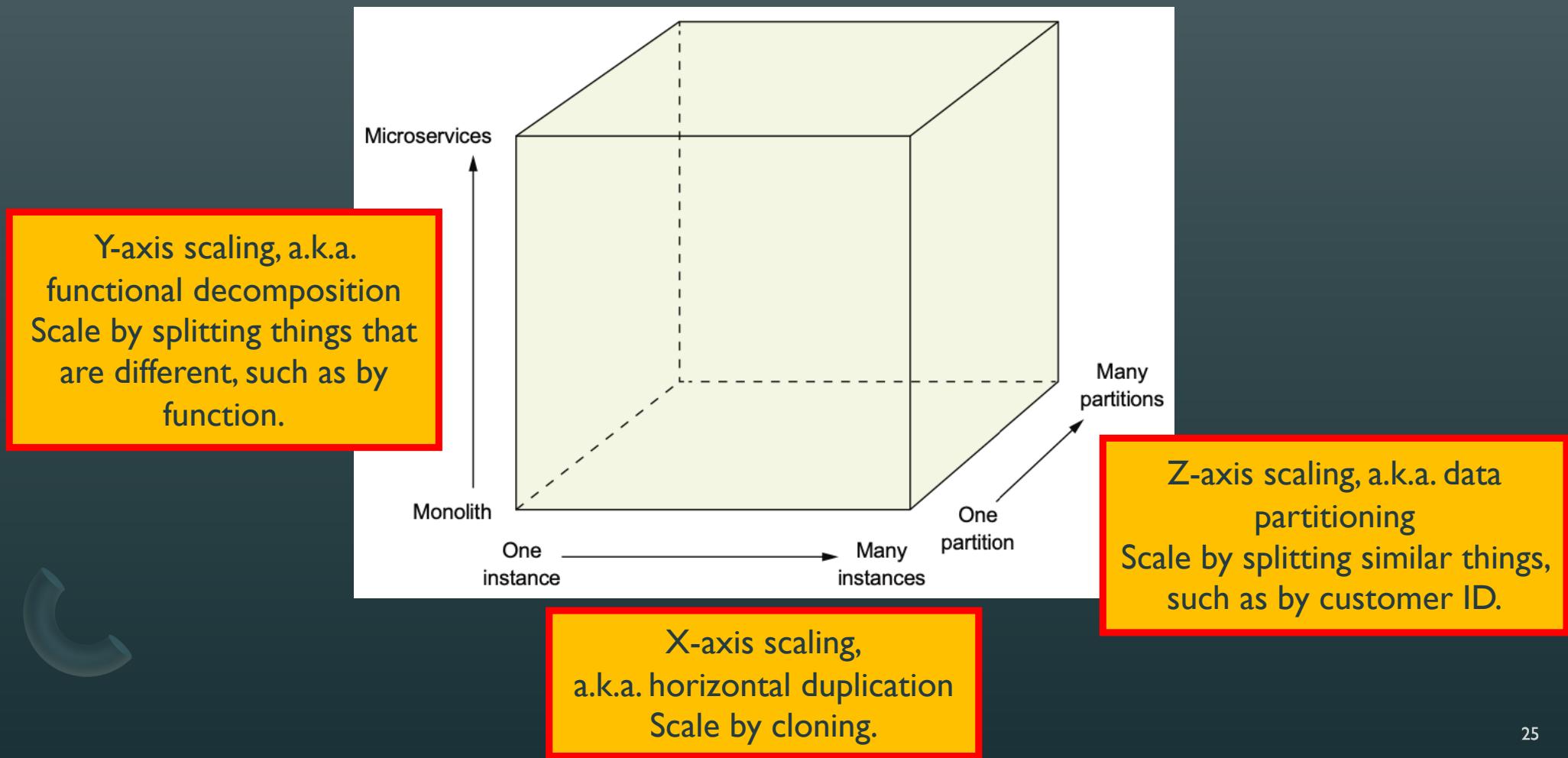
- Delivering a reliable monolith is challenging
 - There are frequent production outages
 - Due to lack of testability
- Locked into increasing obsolete technology stack
 - Difficult to adopt new frameworks and languages
 - Difficult to upgrade version for each components



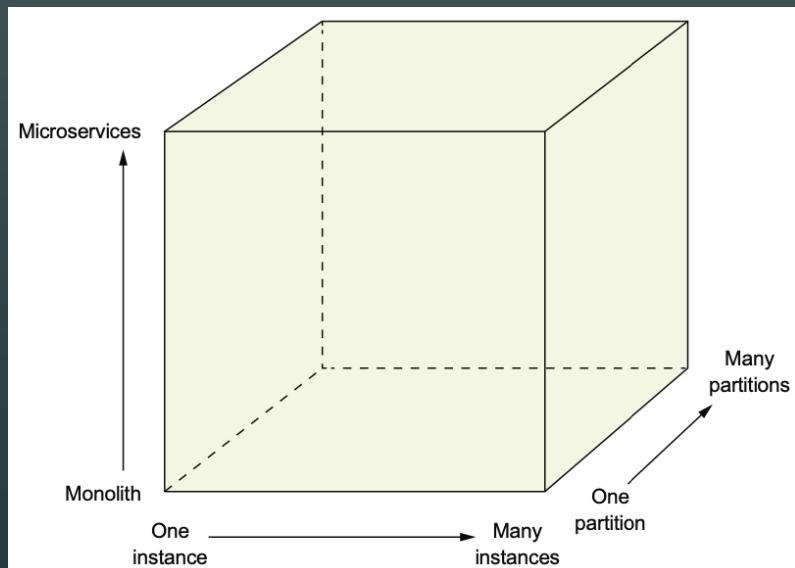
The microservice architecture is an **architectural style** that **structures** an application as a **set of loosely coupled, services** organized around **business capabilities**



Scale cube and microservices



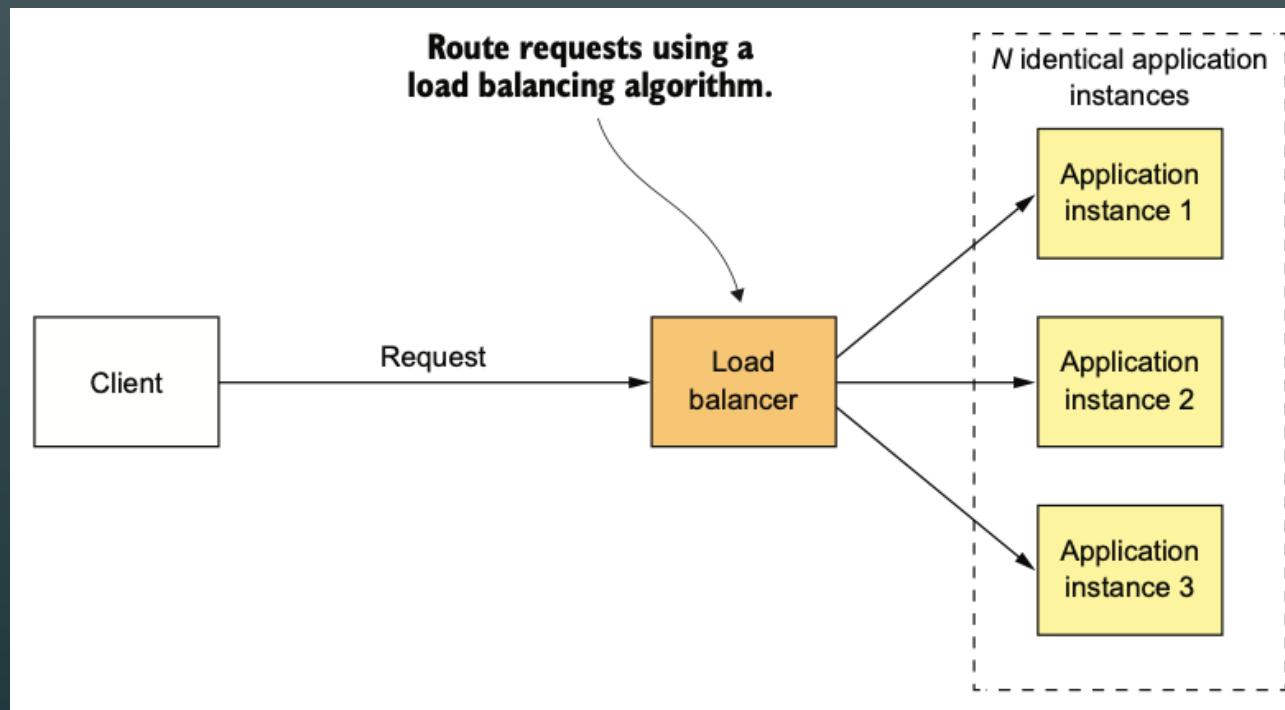
X-axis :- scaling load balances requests across multiple instances (1)



X-axis scaling,
a.k.a. horizontal duplication
Scale by cloning.

- X-axis scaling is a common way to scale a monolithic application
- Running multiple instances of the application behind a load balancer
- The load balancer distributes requests among the N identical instances of the application

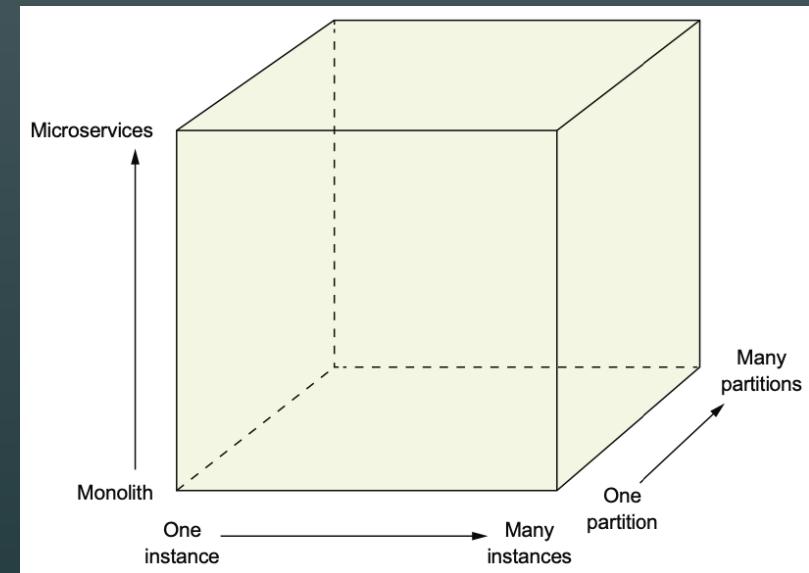
X-axis :- scaling load balances requests across multiple instances (2)



- X-axis is great way of improving the capacity and availability of an application.

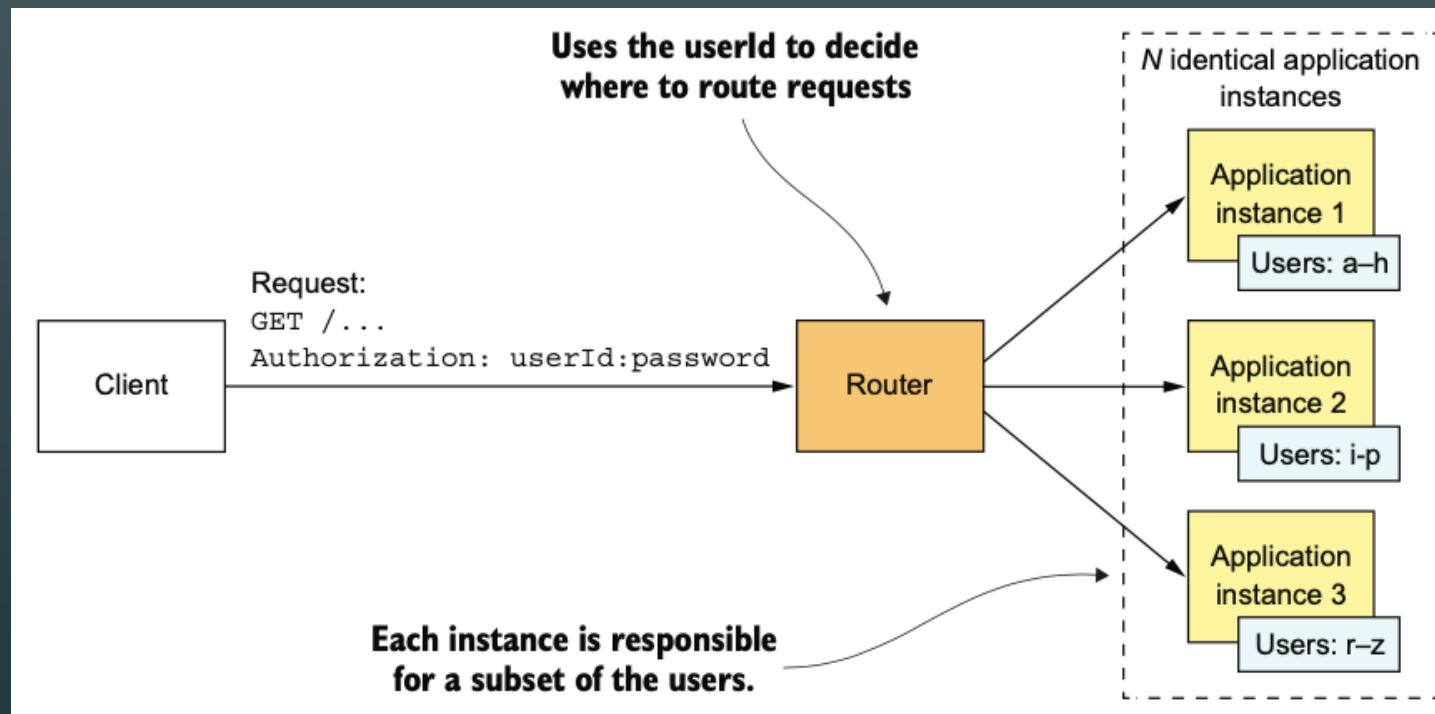
Z-axis :- scaling routes requests based on an attribute of the request (l)

- Also runs multiple instances of the monolith application
- But each instance is responsible for only a subset of the data
- The router in front of the instances uses a request attribute to route it to the appropriate instance
- for example, route requests using *userId*.



Z-axis scaling, a.k.a. data partitioning
Scale by splitting similar things, such as by customer ID.

Z-axis :- scaling routes requests based on an attribute of the request (2)



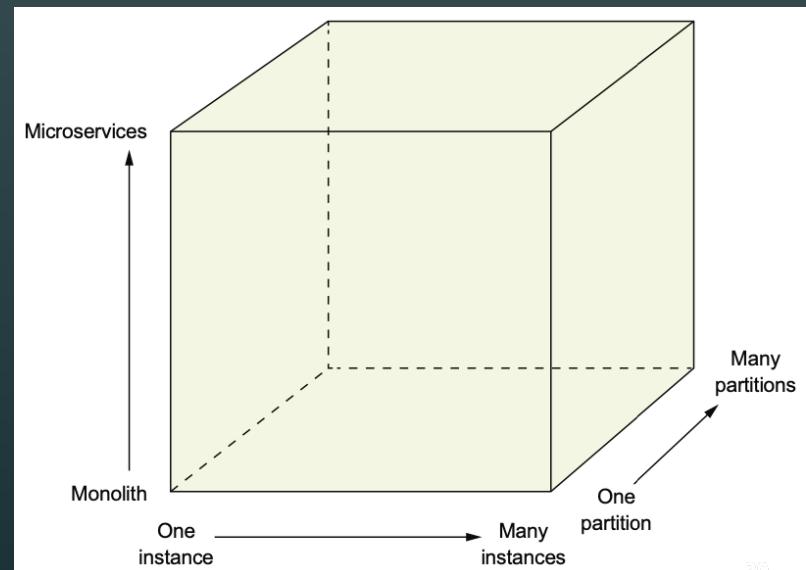
- Z-axis scaling is a great way to scale an application to handle increasing transaction and data volumes.

Y-axis :- scaling functionally decomposes an application into services (1)

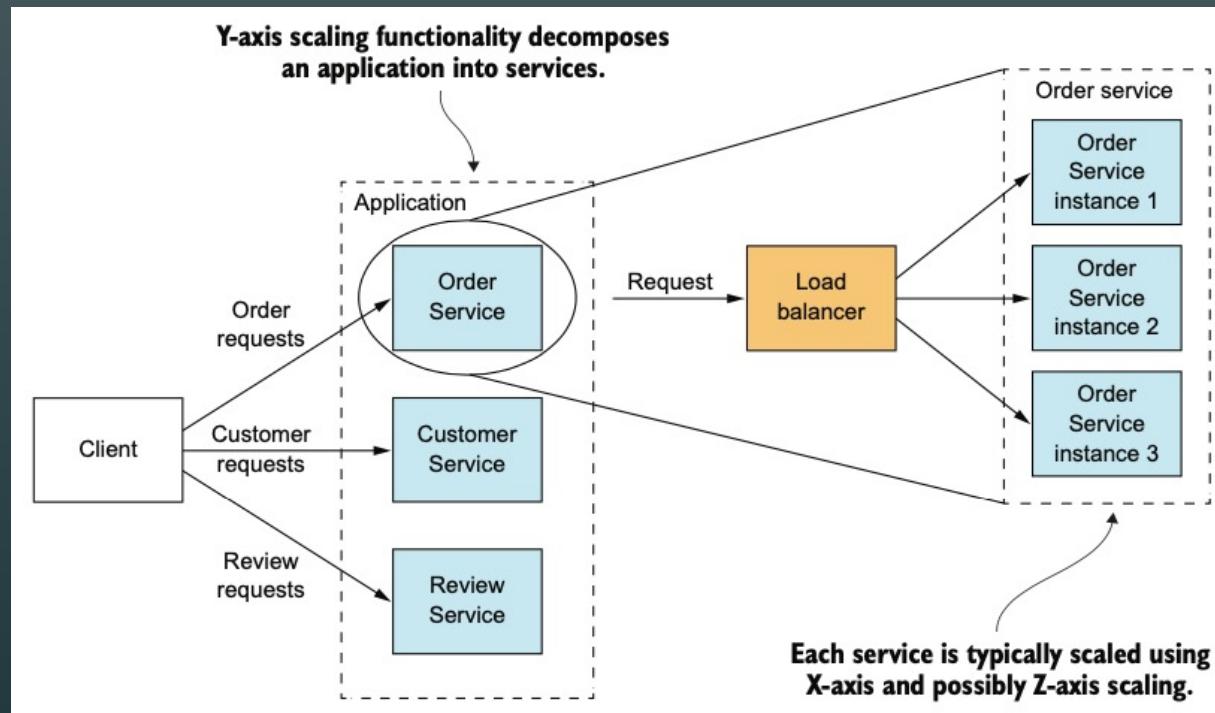
- Y-axis scaling, or functional decomposition, solves the problem of increasing development and application complexity.
- Splitting a monolithic application into a set of services
 - for examples, order management, customer management, and so on



Y-axis scaling, a.k.a.
functional decomposition
Scale by splitting things that
are different, such as by
function.

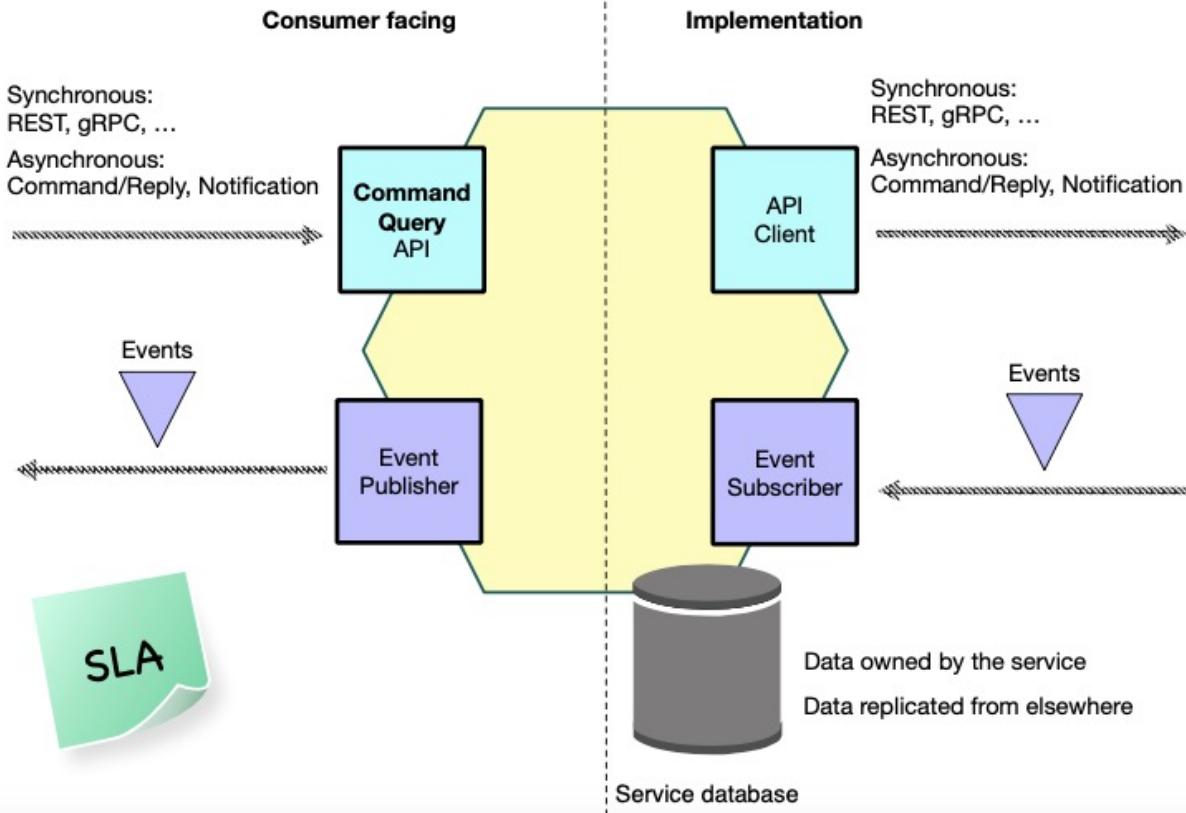


Y-axis :- scaling functionally decomposes an application into services (2)

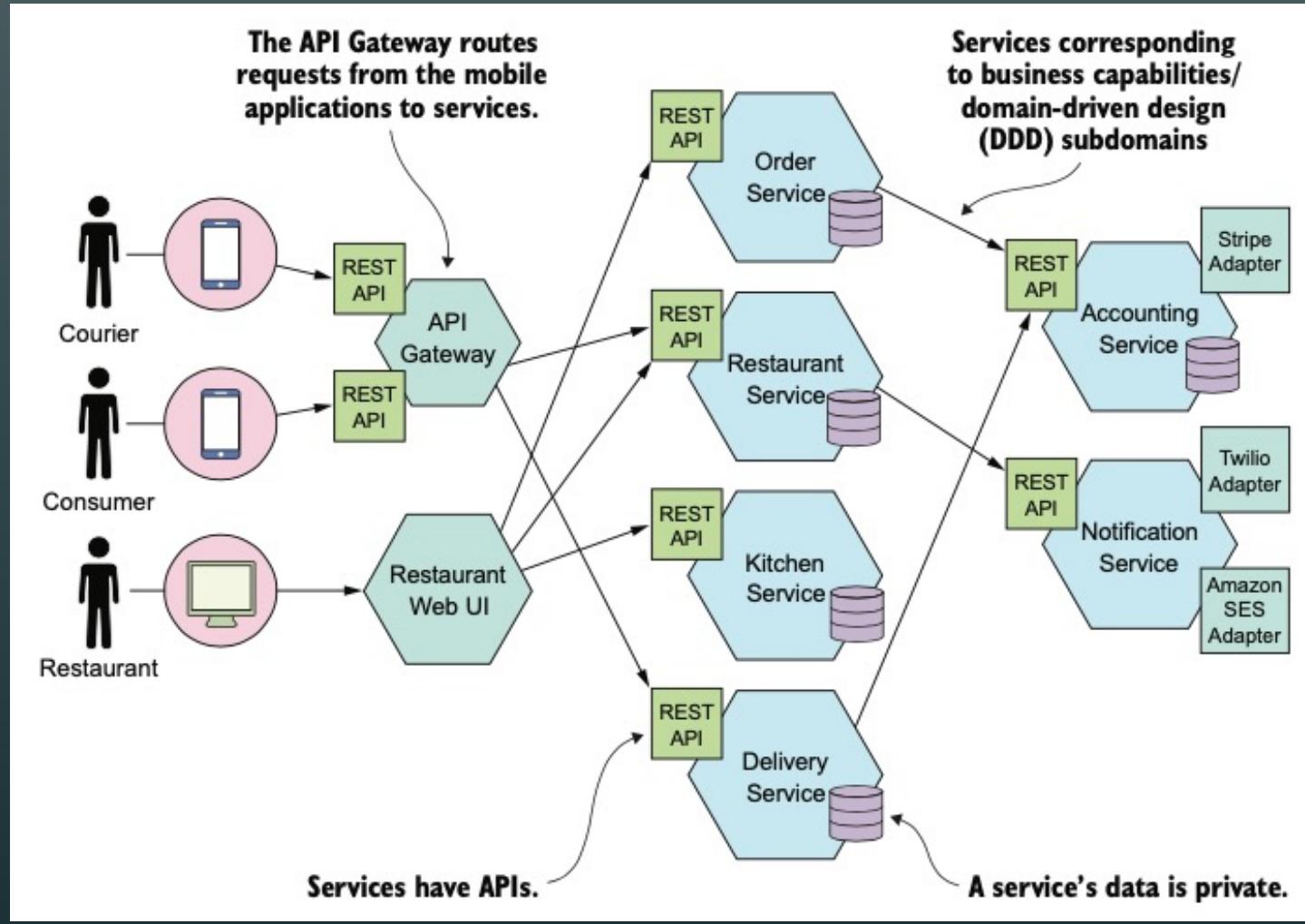


- A service can be scaled using X-axis scaling and, possibly, Z-axis scaling.

Service = independently deployable component



Solution: microservice architecture



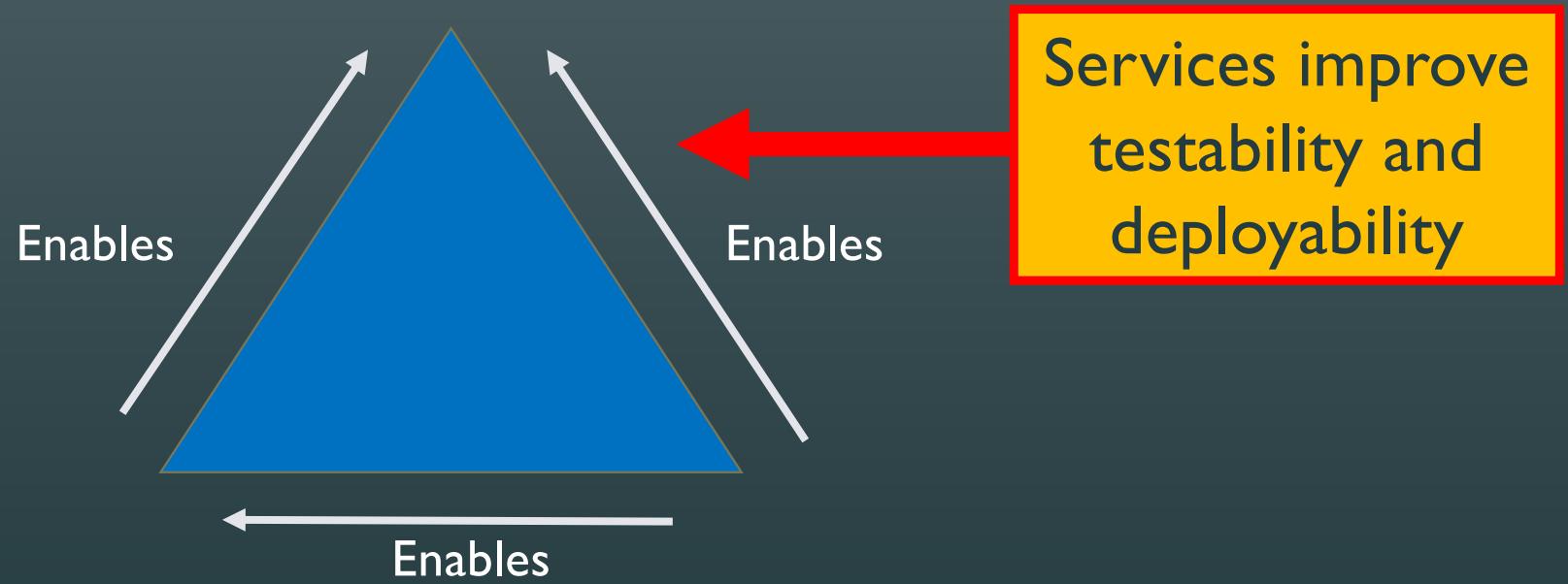
-ilities of microservice architecture

- Maintainability
- Testability
- Deployability
- ...



Process:

DevOps/Continuous delivery/deployment



Organization:

Small, autonomous teams



Architecture:

Microservice architecture

Teams own services

Microservices ⇒ Evolve the technology stack

- Pick a new technology when
 - Writing a new service
 - Make major changes to an existing service
- Let you experiment and fail safely

Agenda

- A brief refresher on software architecture
- From monolith to microservices
- **Microservices != silver bullet**
- Applying the microservice pattern language



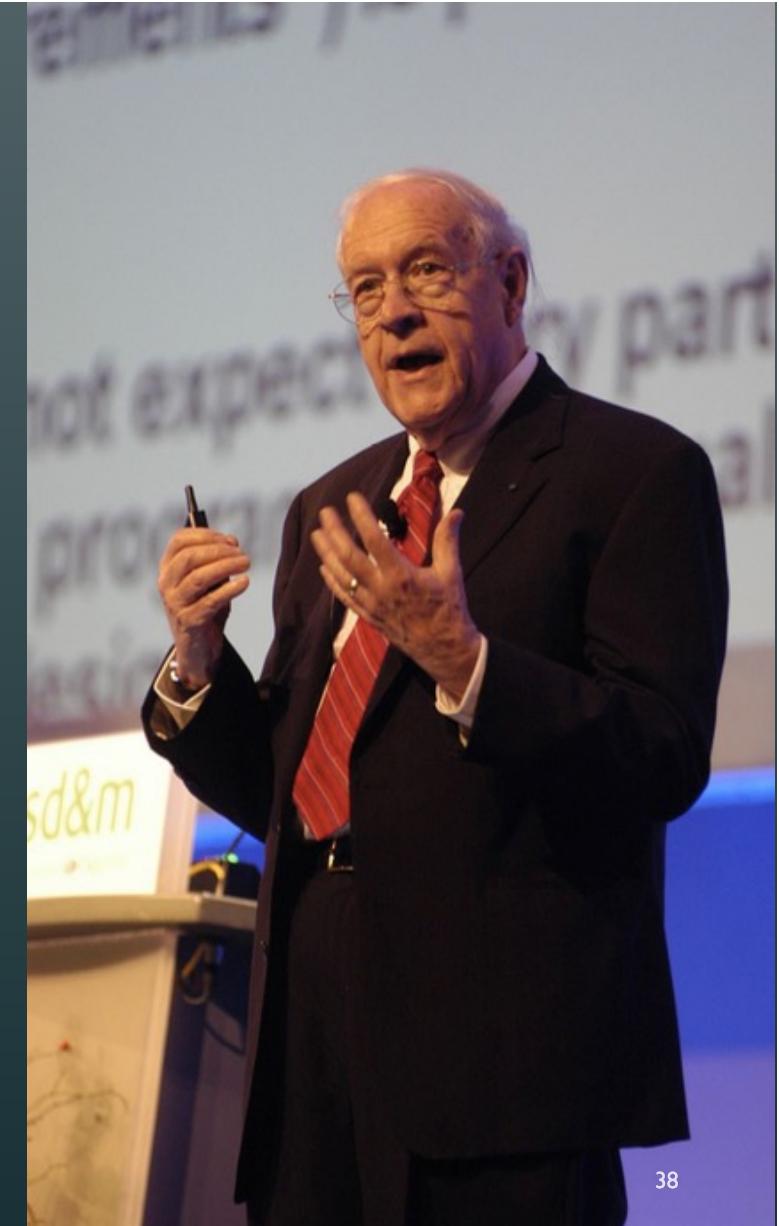
No silver bullet

No Silver Bullet —Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

https://en.wikipedia.org/wiki/Fred_Brooks



Benefits of the microservice architecture (1)

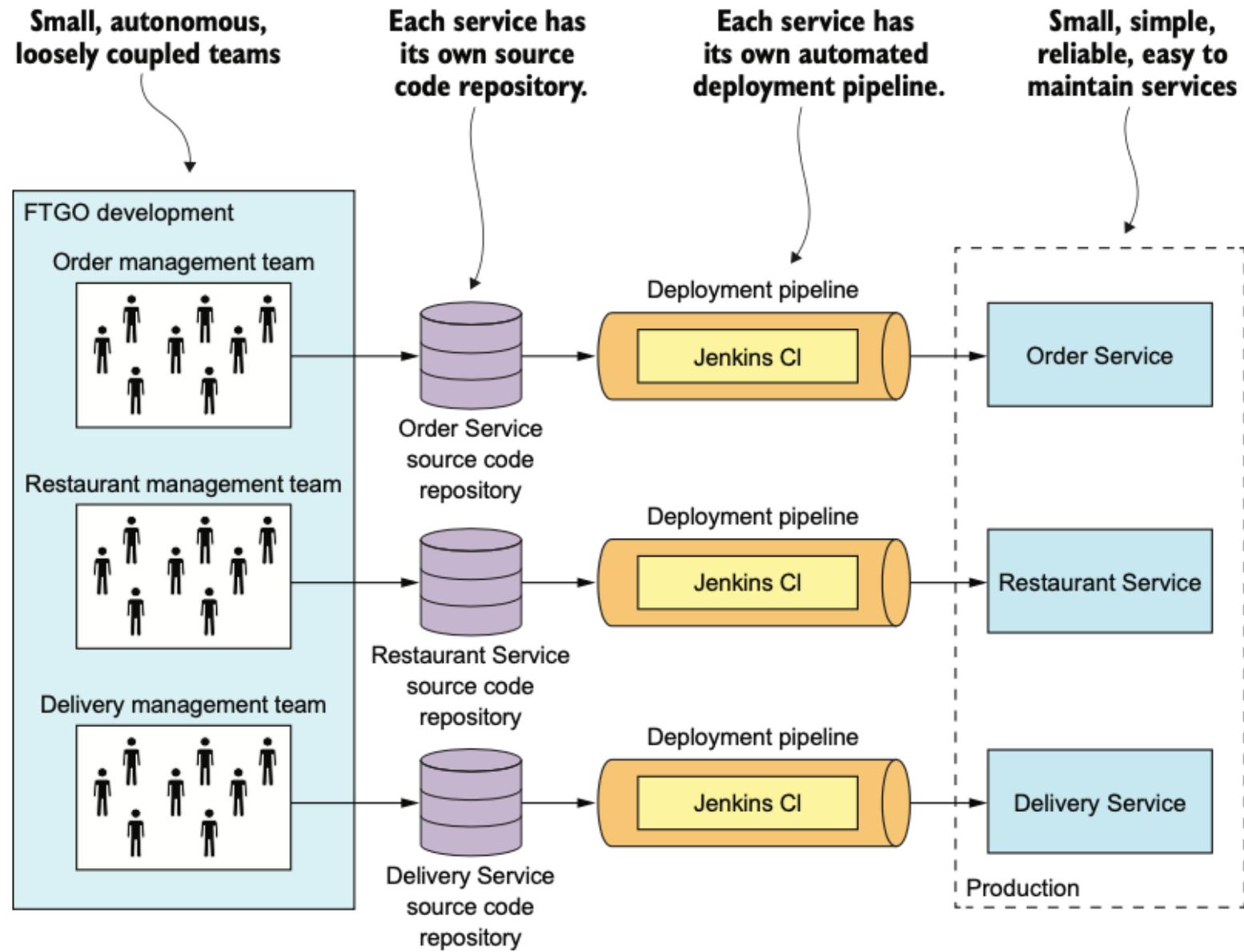
- It enables the continuous delivery and deployment of large, complex applications
 - continuous delivery/deployment is part of DevOps
(a set of practices for the rapid, frequent, and reliable delivery of software)
 - 3 ways that the microservice architecture enables CD
 - It has the testability required by continuous delivery/deployment
 - It has the deployability required by continuous delivery/deployment
 - It enables development teams to be autonomous and loosely coupled



Benefits of the microservice architecture (2)

- Services are small and easily maintained
 - The code is easier for a developer to understand
 - The small code base doesn't slow down the IDE, making developers more productive
 - Each service typically starts a lot faster than a large monolith does





Benefits of the microservice architecture (3)

- Services are independently deployable
 - Each service in a microservice architecture can be scaled independently
 - CPU-intensive vs. memory-intensive no need to be deployed together
- It enables teams to be autonomous
- It has better fault isolation
- It allows easy experimenting and adoption of new technologies



Drawback of the microservice architecture (1)

- Finding the right set of services is challenging
 - If you decompose a system incorrectly, you'll build a distributed monolith
 - It has the drawbacks of both the monolithic and the microservice architecture.



Drawback of the microservice architecture (2)

- Distributed systems are complex
 - Makes development, testing, and deployment difficult
 - Requires the use of unfamiliar techniques
 - Must use sagas to maintain data consistency across services
 - Must implement queries using either API composition or CQRS views



Drawback of the microservice architecture (3)

- Deploying features that span multiple services requires careful coordination
 - Need to create a rollout plan that orders service deployments based on the dependencies between services



Drawback of the microservice architecture (4)

- Deciding when to adopt the microservice architecture is difficult
 - A startup should begin with a monolithic application
 - When the problem is how to handle complexity
→ It makes sense to decompose the application into microservices



Agenda

- A brief refresher on software architecture
- From monolith to microservices
- Microservices != silver bullet
- **Applying the microservice pattern language**

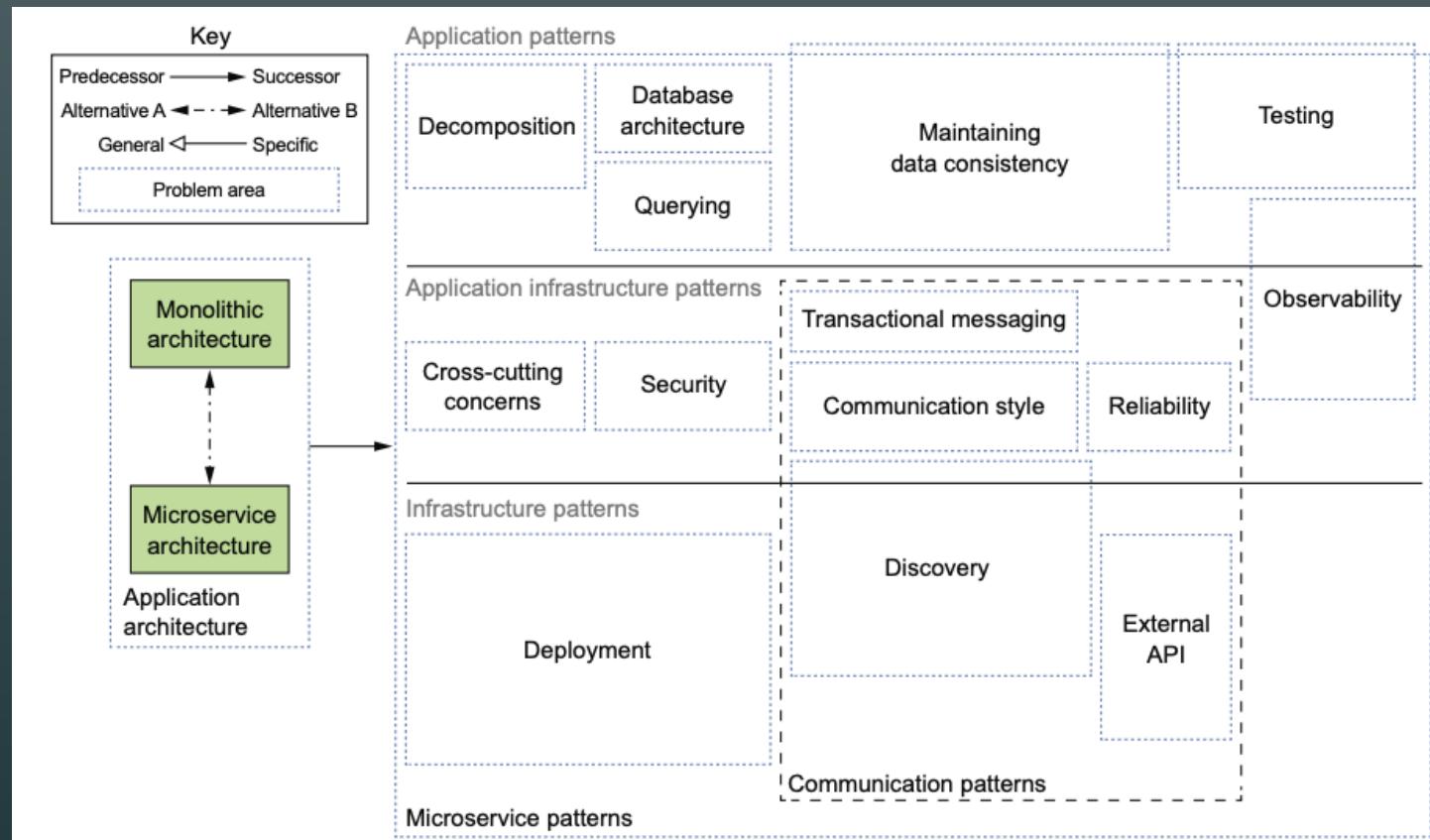


Patterns and Pattern Language

- A pattern is a reusable solution to a problem that occurs in a particular context.
- The pattern language guides you when developing an architecture.
- What architectural decisions you must make for each decision:
 - Available options
 - Trade-offs of each option



A high-level view of the Microservice architecture pattern language



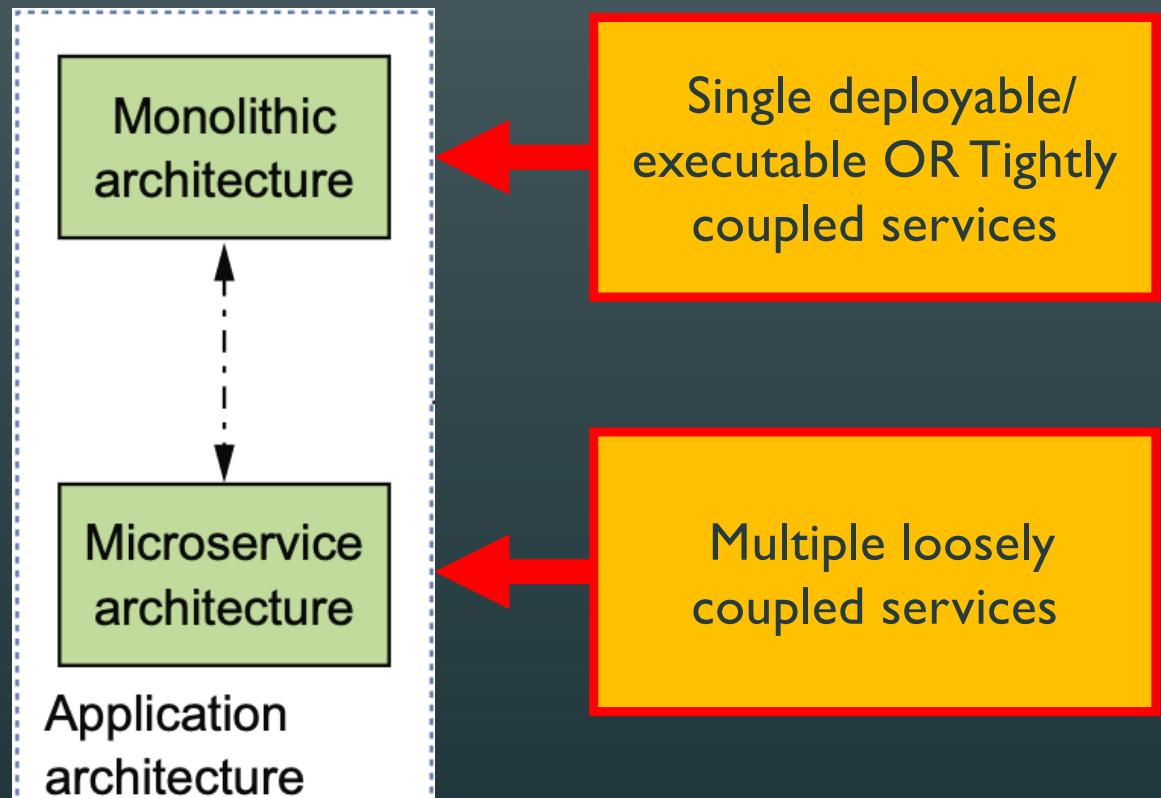
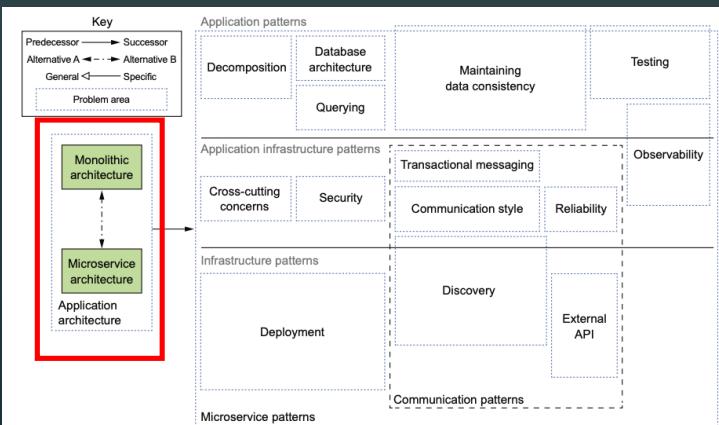
Application patterns —These solve problems faced by developers.

Application infrastructure —These are for infrastructure issues that also impact development.

Infrastructure patterns —These solve problems that are mostly infrastructure issues outside of development.

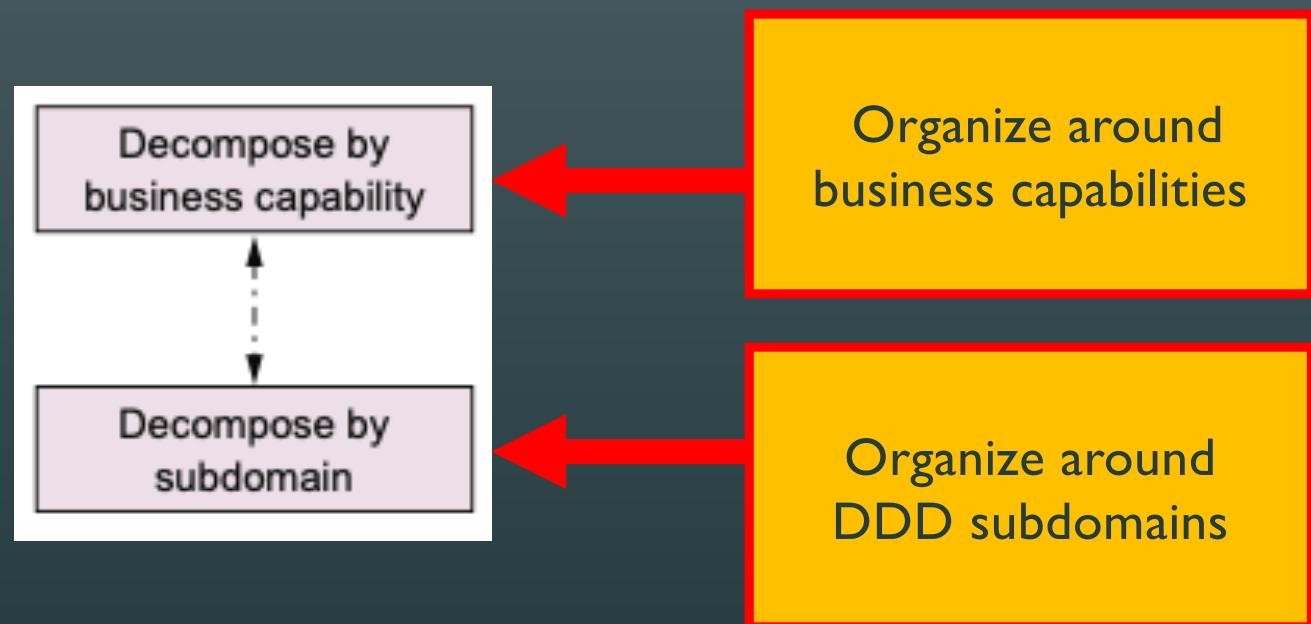
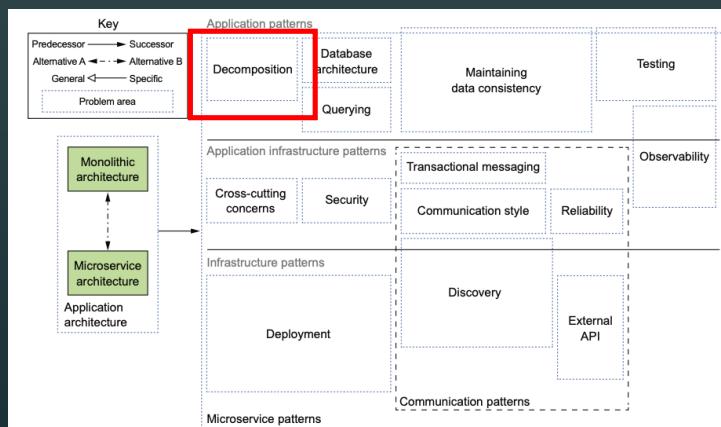
Issue: What's the deployment architecture?

- Force :-
 - Maintainability
 - Testability
 - Deployability

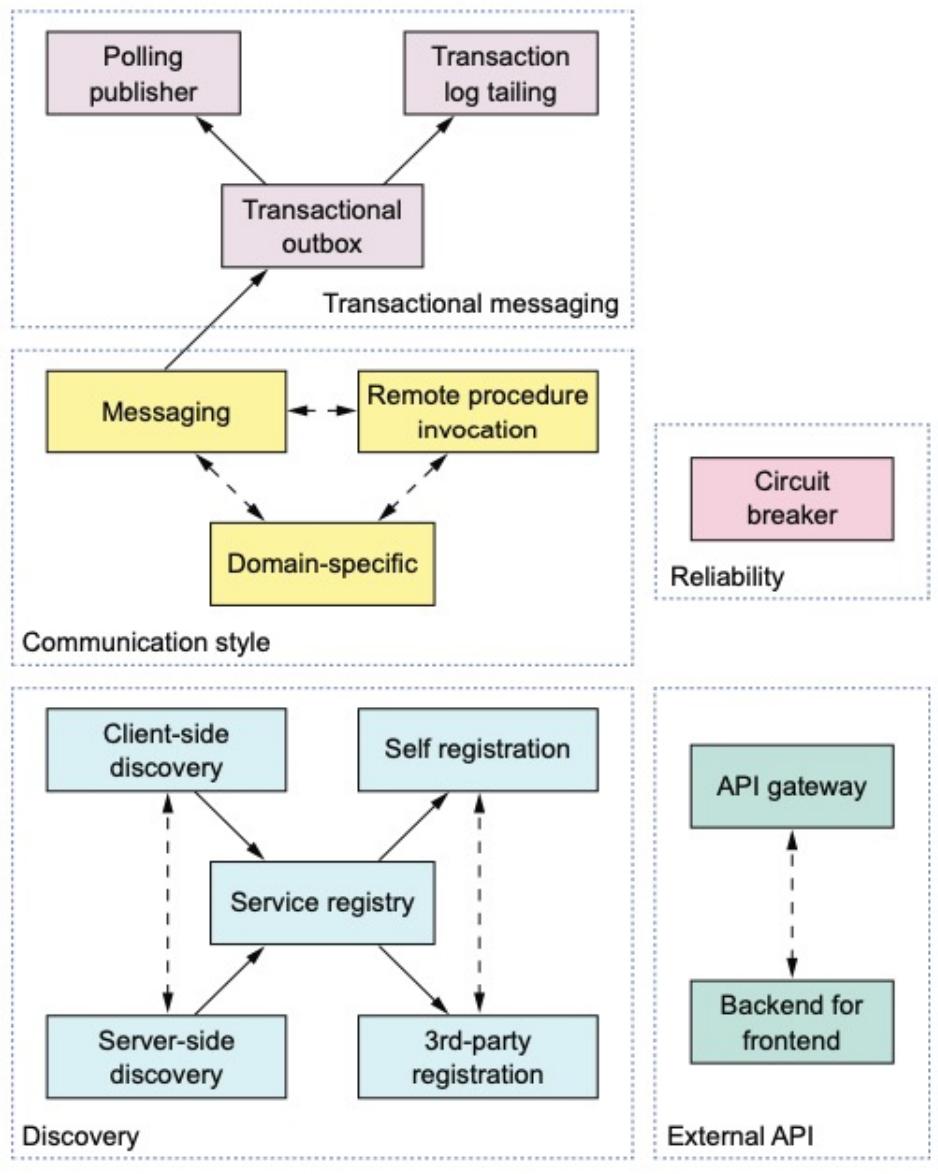
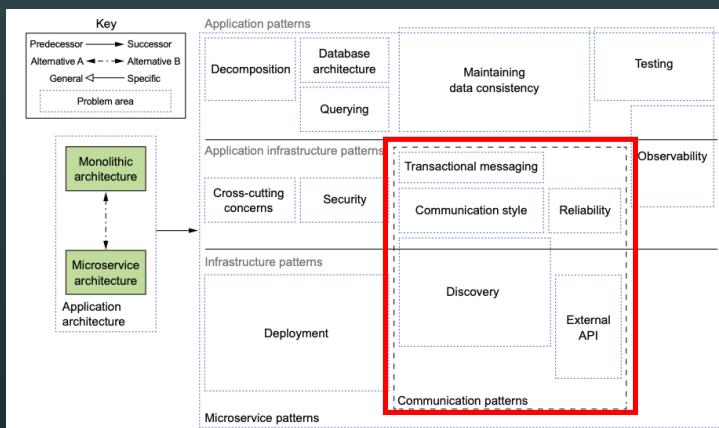
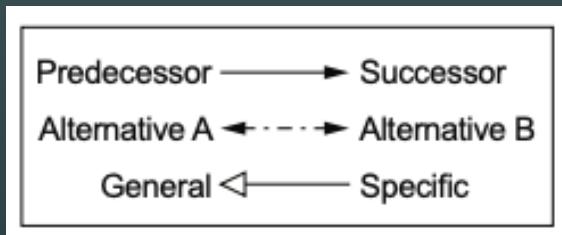


Issue: How to decompose an application into services?

- Force :-
 - Stability
 - Cohesive
 - Loosely couples
 - Not too large

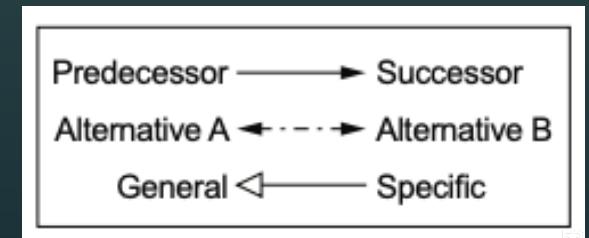
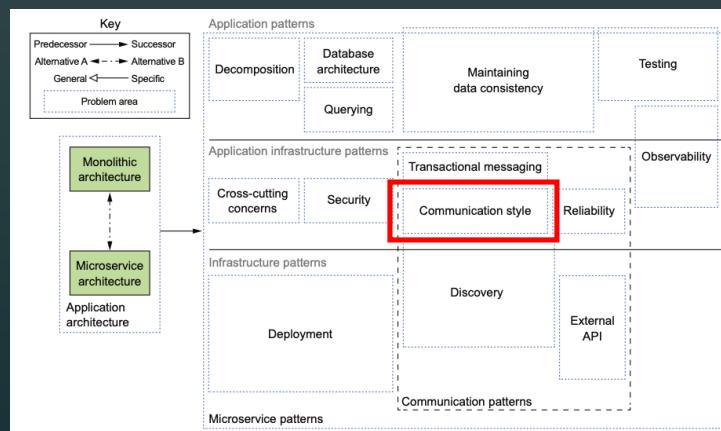
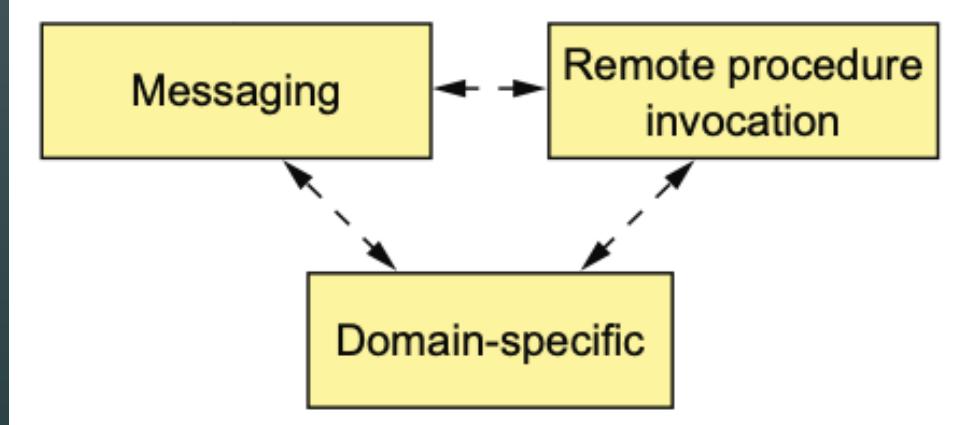


The five groups of communication patterns



Issue: How do services communicate?

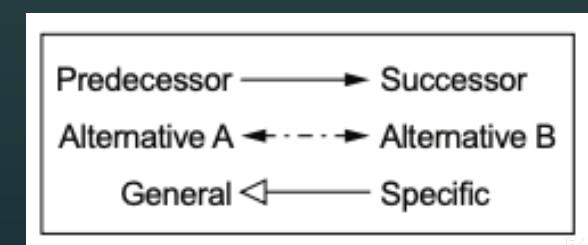
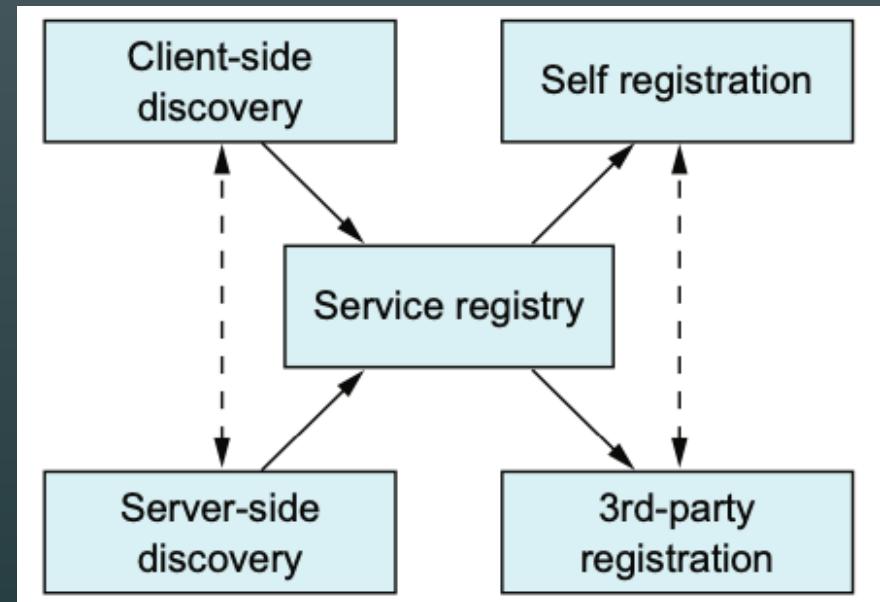
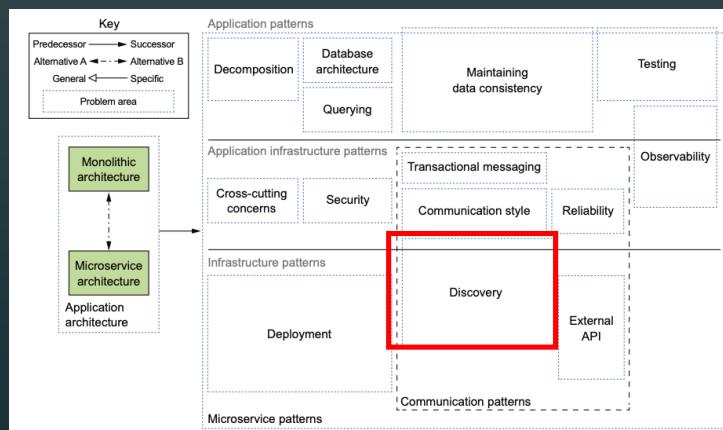
- Force :-
 - Services must communicate
 - Usually processes on different machines



Issue: How to discover a service instance's network location?

- Force :-

 - Client needs IP address of service instance
 - Dynamic IP addresses
 - Dynamically provisioned instances



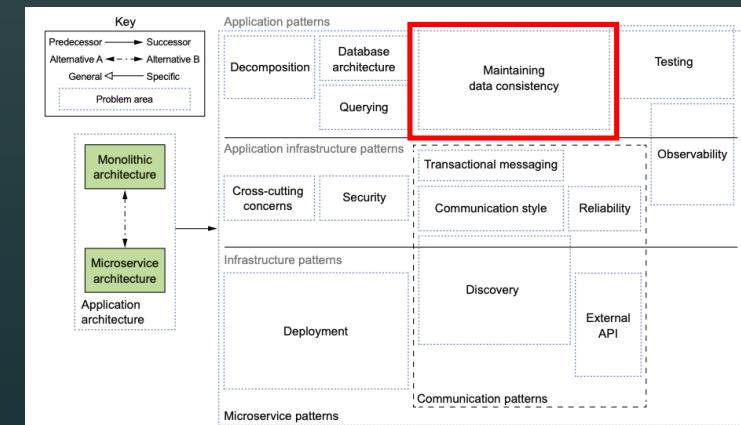
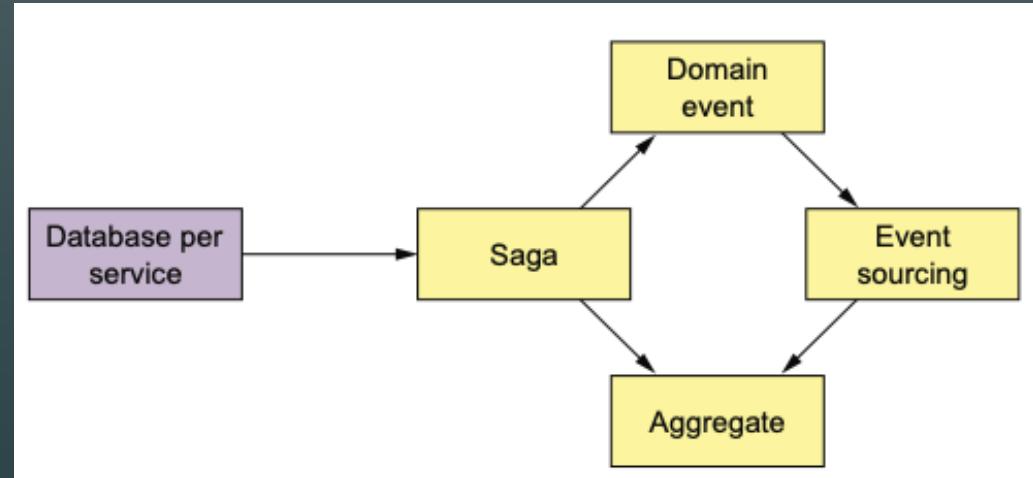
Issue: how to maintain data consistency?

- Context :-

- Each service has its own database
- Data is private to a service

- Forces :-

- Transactional data consistency must be maintained across multiple services
- 2PC (two-phase commit) is not an option



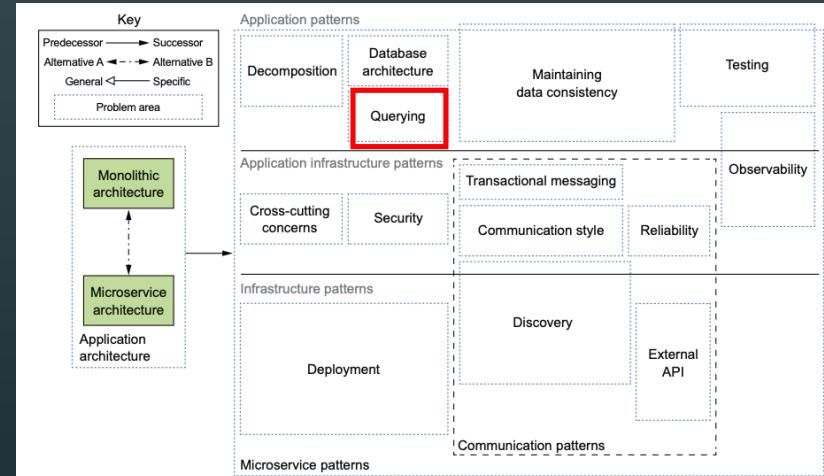
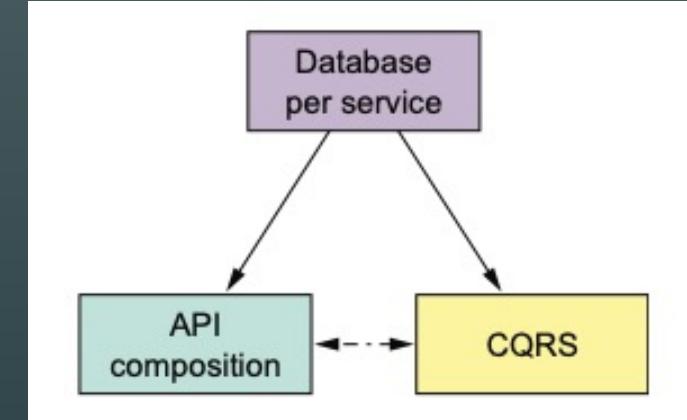
Issue: How to perform queries?

- **Context :-**

- Each service has its own database

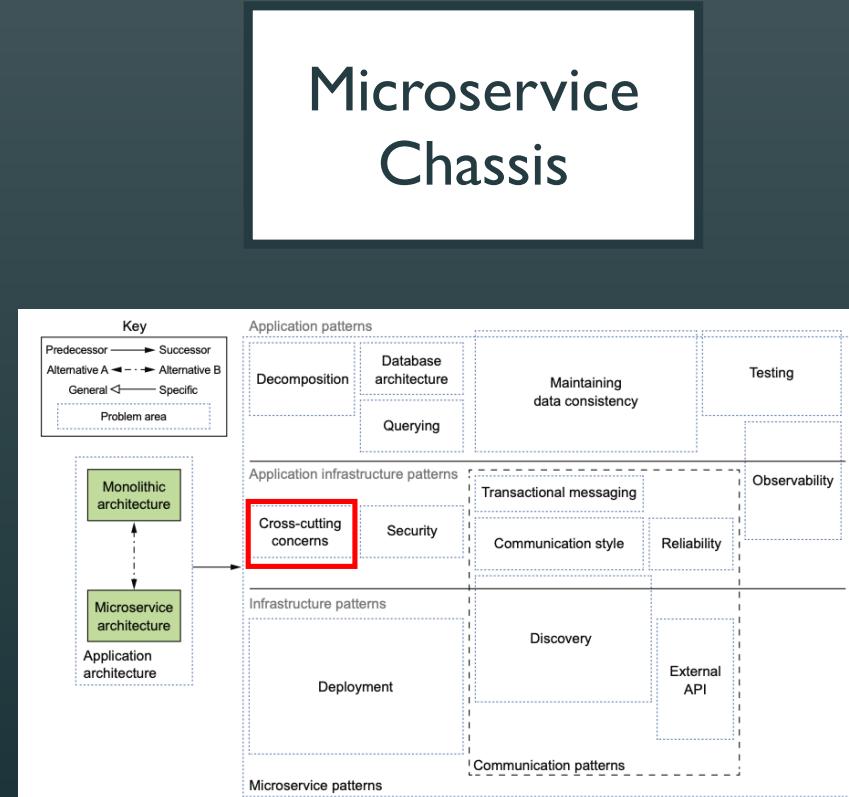
- **Force :-**

- Queries must join data from multiple services
- Data is private to a service



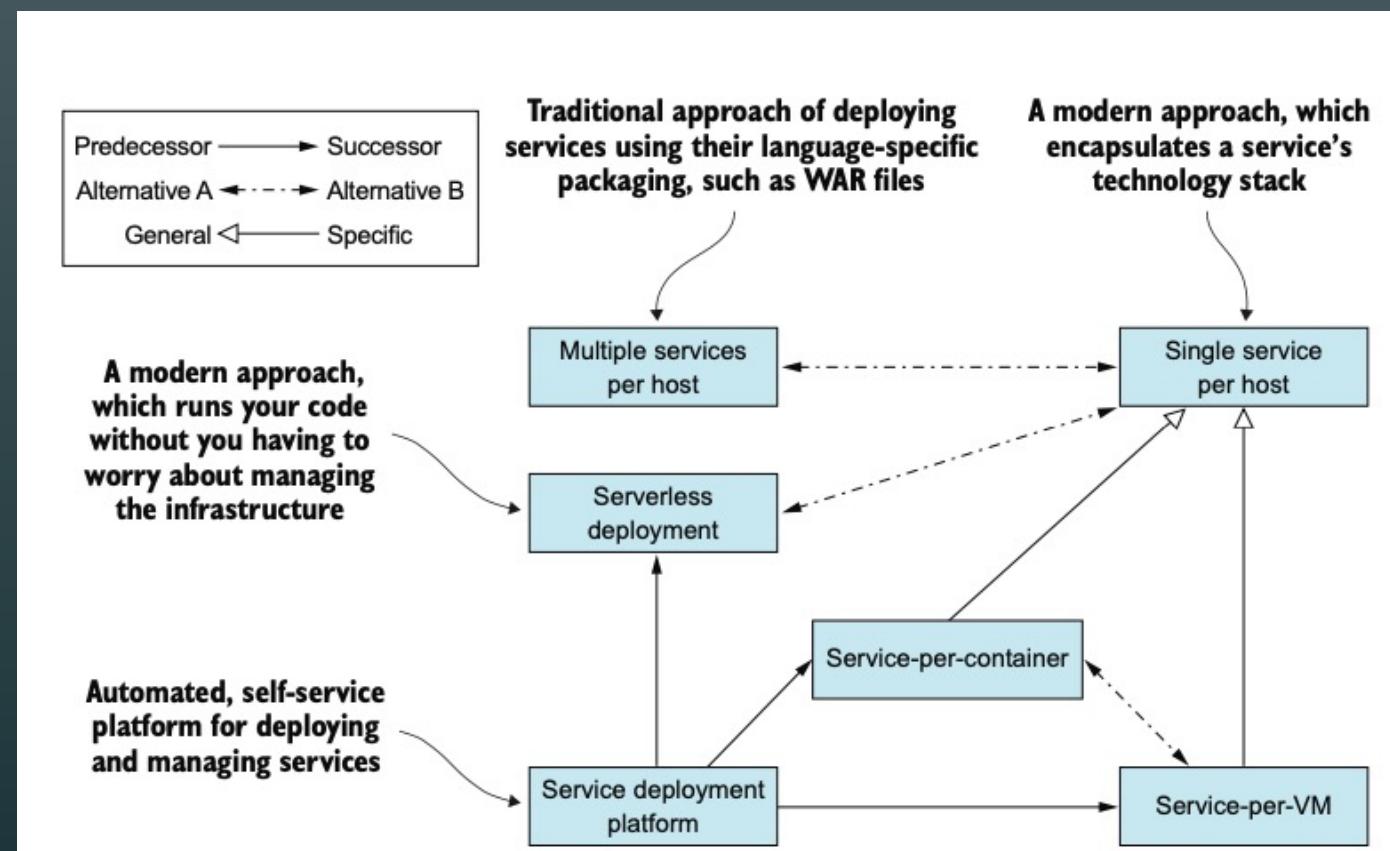
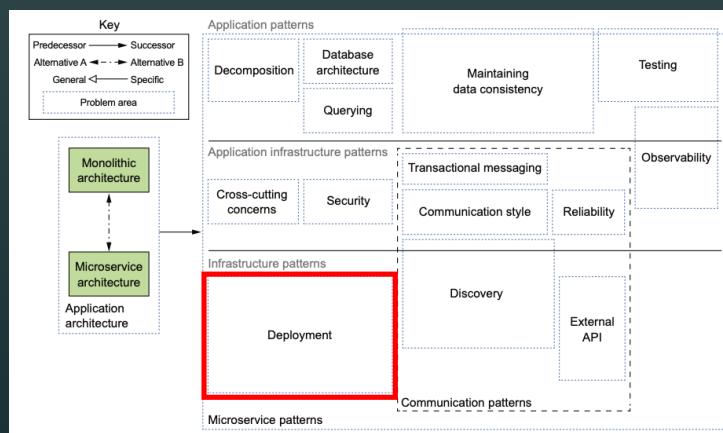
Issue: How to handle cross cutting concerns?

- Force :-
 - Every service must implement logging; externalize configuration; health check endpoint; metrics; ...

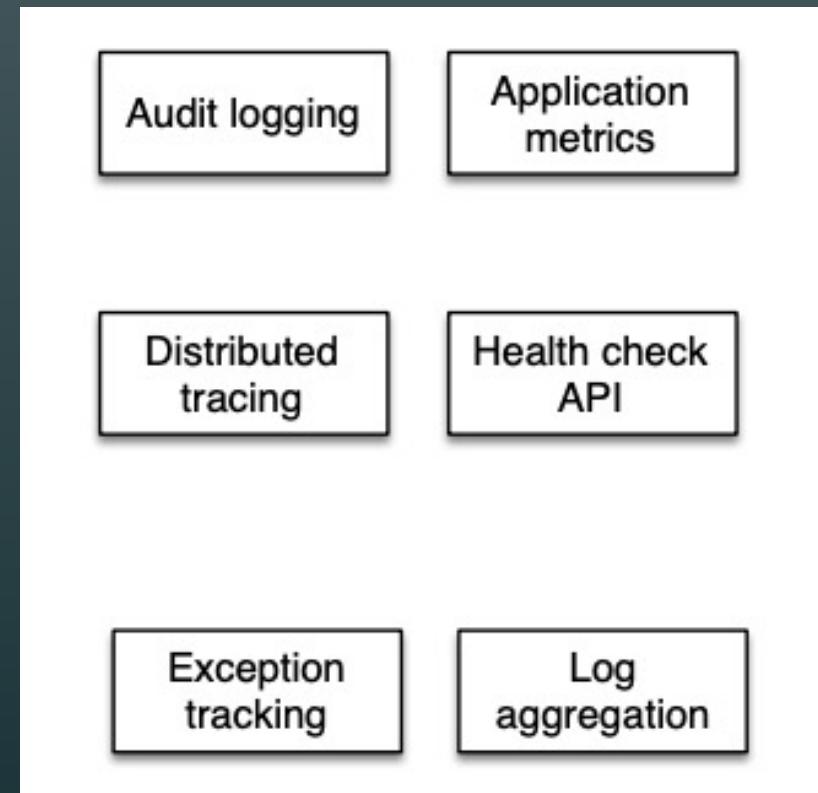
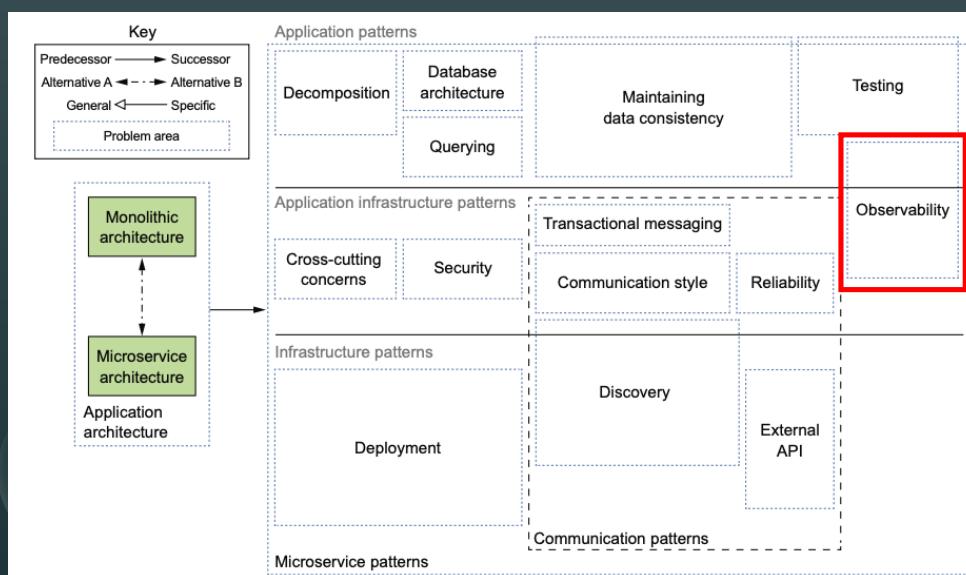


Issue: How to deploy an application's services?

- Force :-
- Multiple languages
- Isolated
- Constrained
- Monitor-able
- Reliable
- Efficient



Issue: how to monitor the behavior of your application?



Summary

- The goal of architecture is to satisfy non-functional requirements
- For continuous delivery/deployment use the appropriate architectural style
 - Small applications ⇒ Monolithic architecture
 - Complex applications ⇒ Microservice architecture
- Use the pattern language to guide your decision making

