Introduction to Microservices Using Patterns

Neueda Training – Ericsson

July 2022

Objectives

- What are Microservices?
- Microservices Patterns
- Appendix: Microservices with C++

What are Microservices?

Breakout Exercise: What Are Microservices?

What differentiates a microservice from a “small app”?

How is a microservice different from good object oriented design?

What benefits do you think microservices offer over traditional monolithic designs?

What challenges do you think you face with microservices that don’t exist in monolithic designs?

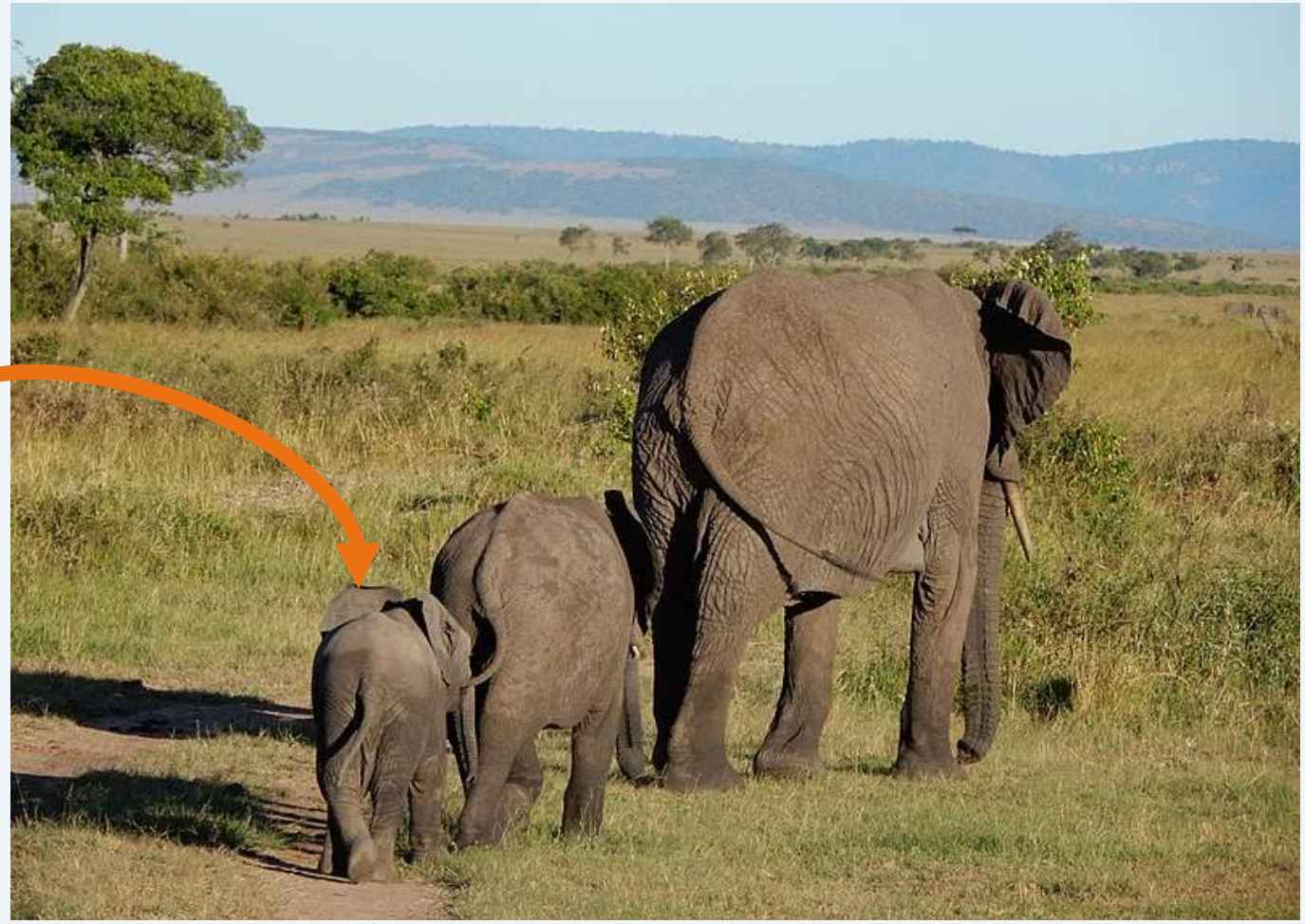
What are microservices?

… the microservice architectural style is an approach to developing a single application as a suite of small services, each running in it's own process and communicating with lightweight mechanisms, often an HTTP resource API.

- Martin Fowler

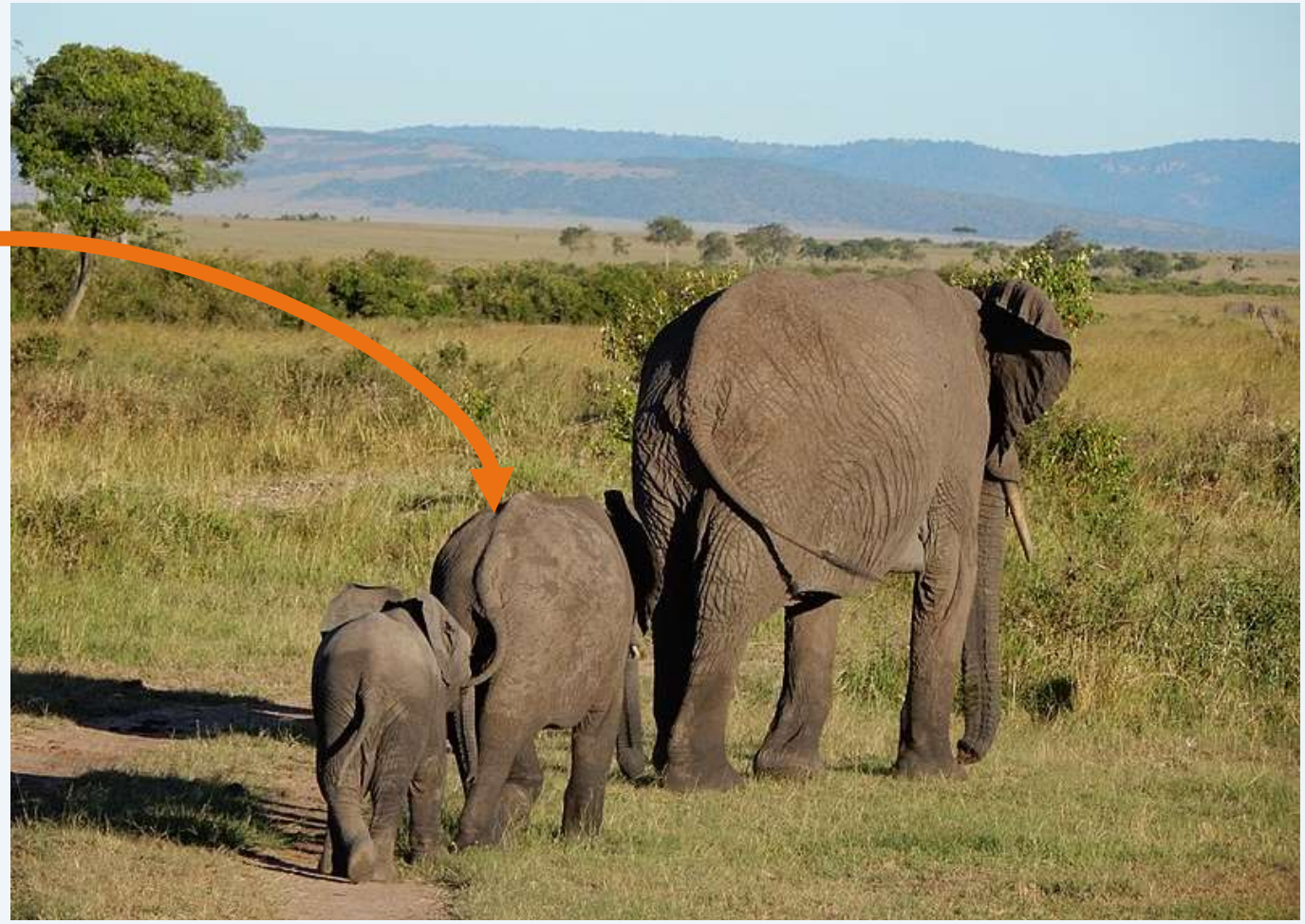
Why do we need microservices?

- Your app starts here
  - A small team of developers
  - Single code base
  - Rapid development
  - Interfaces can change and adapt quickly
  - Testing is straightforward
  - Most of your developers understand how it works end to end.



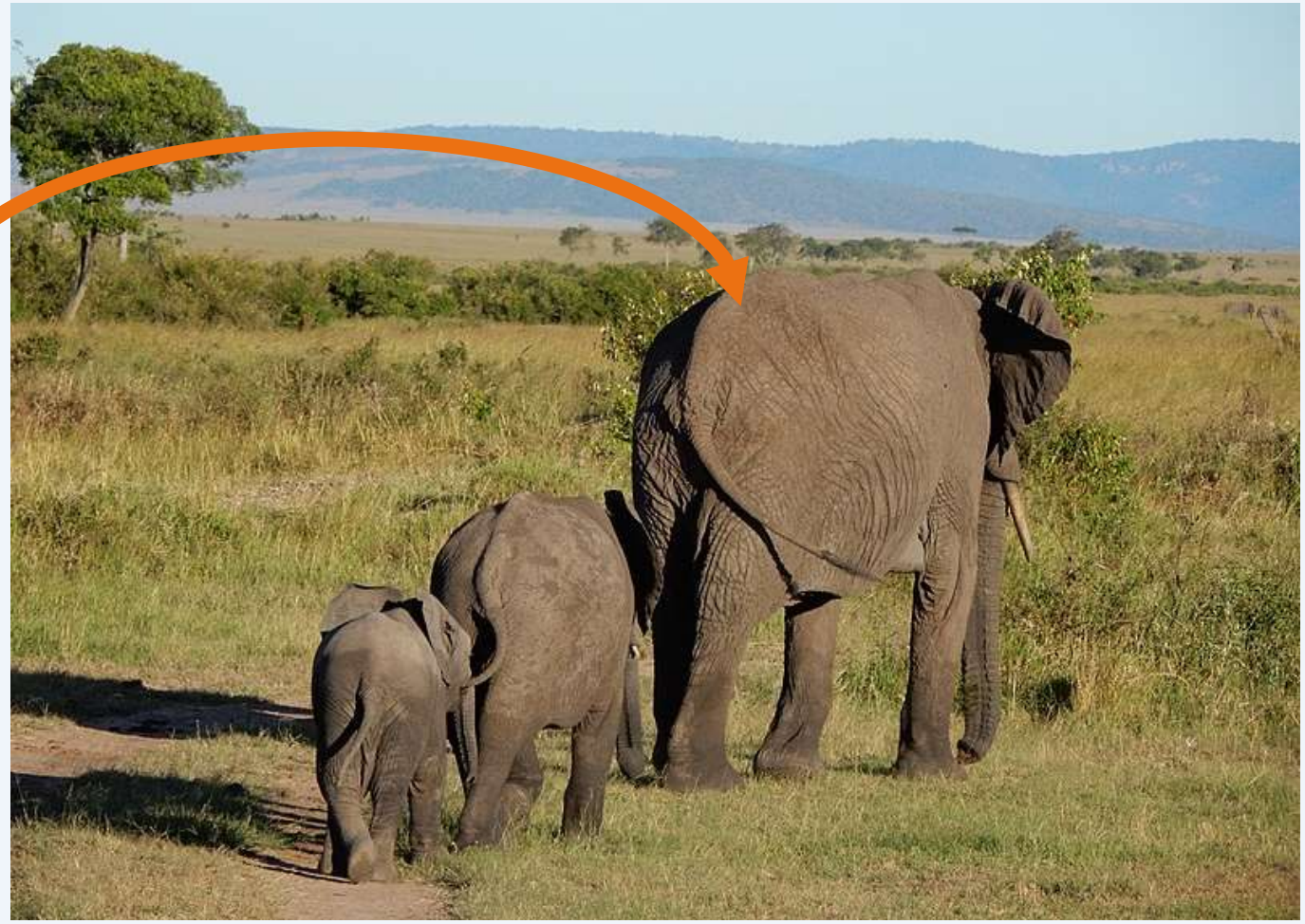
Why do we need microservices?

- You succeed and grow
  - An established user base
  - Your team of developers grows
  - More endpoints and functionality
  - Your codebase expands
  - Making interface changes touches lots of modules
  - Testing gets difficult
  - Rollouts are a major event



Why do we need microservices?

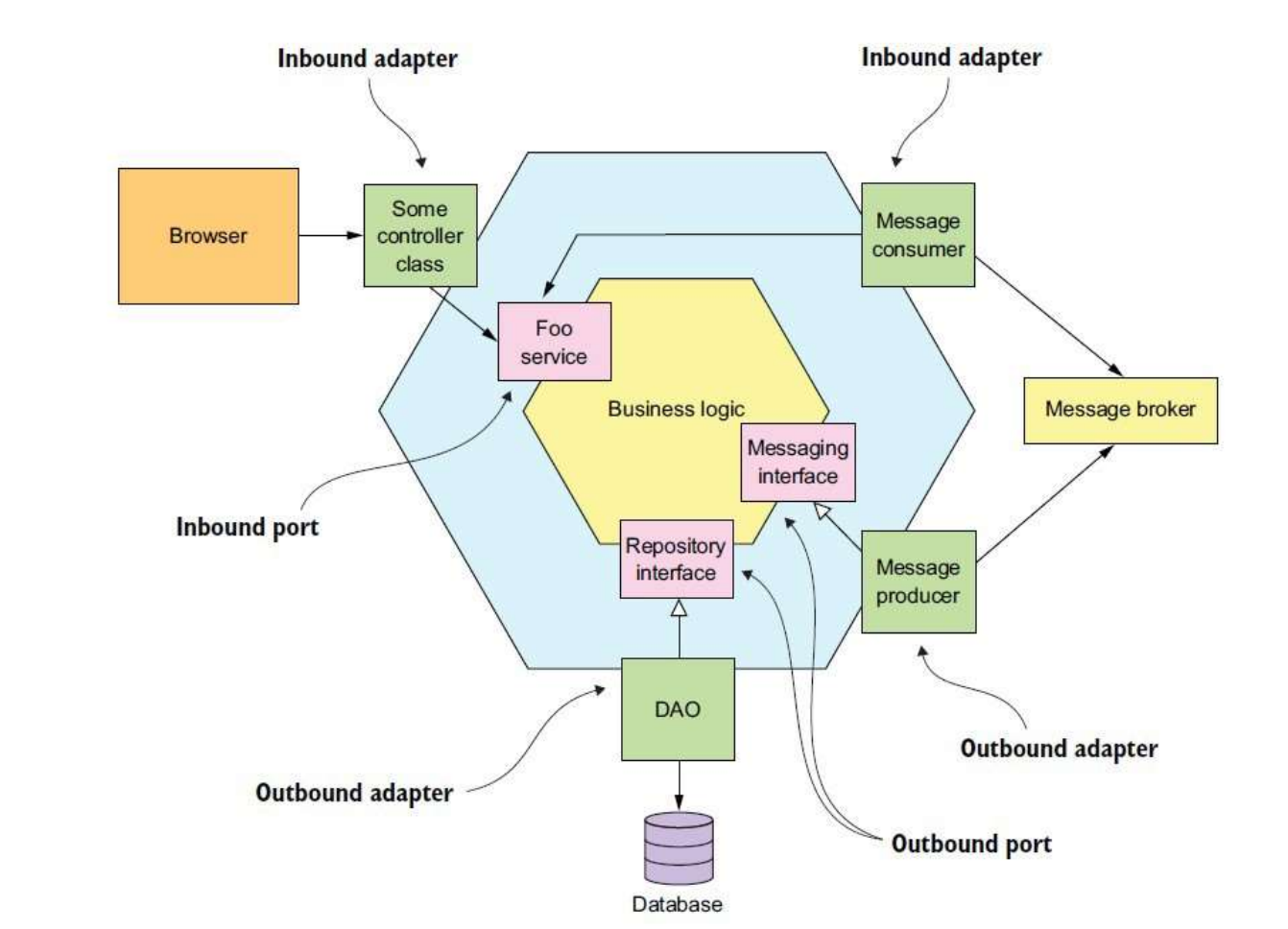
- Eventually …
  - Your codebase is massively complex
  - No-one actually understands it all
  - Changes in one part can inexplicably break other parts
  - Your releases get fewer and further between
  - You don't rollout bug fixes because you can't be sure of the overall effect



Monolithic Architecture

- Traditionally code is structured with
  - Tight Coupling between modules
  - A single database
  - External facing APIs, UIs and adapters to access users and services.

Hexagonal (Monolithic) Architecture



Benefits of Monolithic Architecture

- **Simple** to develop
  - Developer tools are set up for this!
- **Simple** to make sweeping changes
  - Just change your API, Schema, whatever, then rebuild and deploy.
- **Simple** to test
  - Lots of tools for testing in this case
- **Simple** to deploy
  - Just drop an executable in place
- **Simple** to scale
  - Add another instance and run with a load balancer

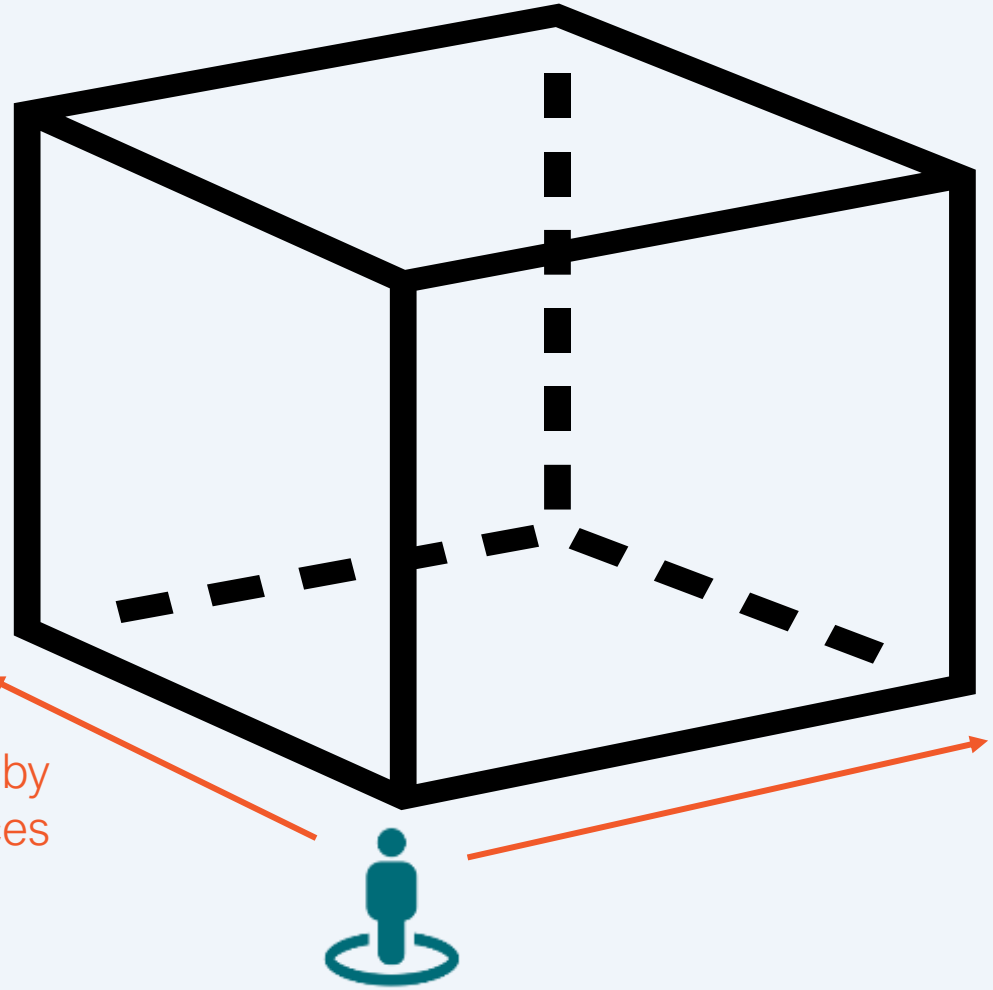
Drawbacks of Monolithic Architecture

- **Difficult** to keep developing
  - Complexity intimidates developers & development is slow
- **Difficult** to test
  - Small change requires testing across the entire monolith
- **Difficult** path from commit to deploy
  - Lots of developers, codebase is often in an unreleasable state
- **Difficult** to make reliable
  - Lacks fault isolation, one failure crashes whole system
- **Difficult** to scale
  - Conflicting requirements of different modules (CPU, memory, etc)

It boils down to how you scale your app…

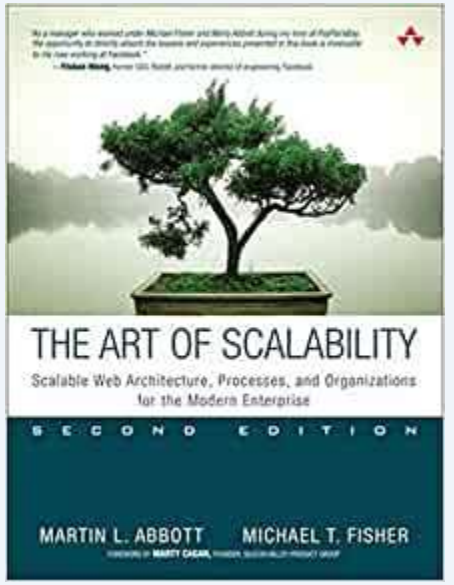
MICROSERVICES

Y-scale: functional decomposition. Scale by splitting different things

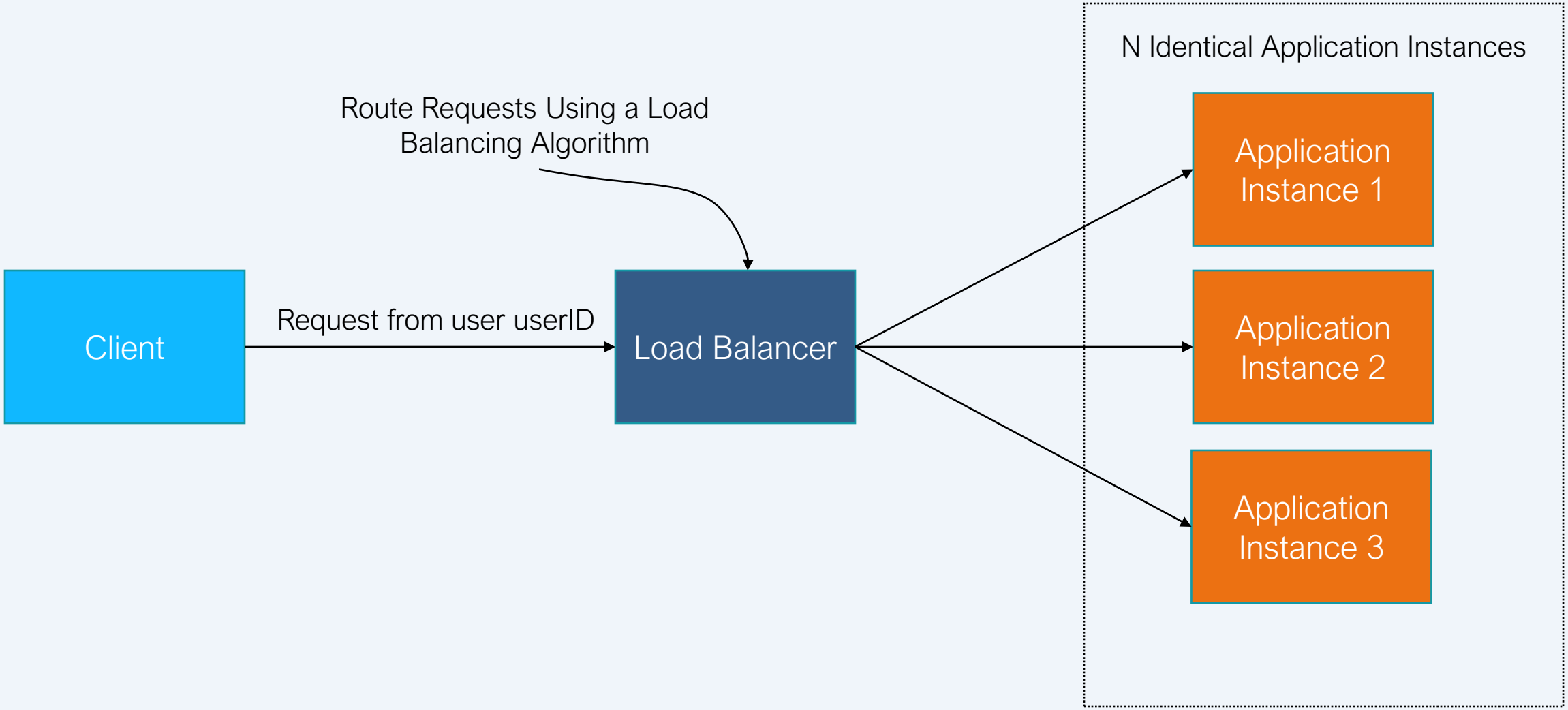


X-scale: add capacity. Scale by duplicating instances

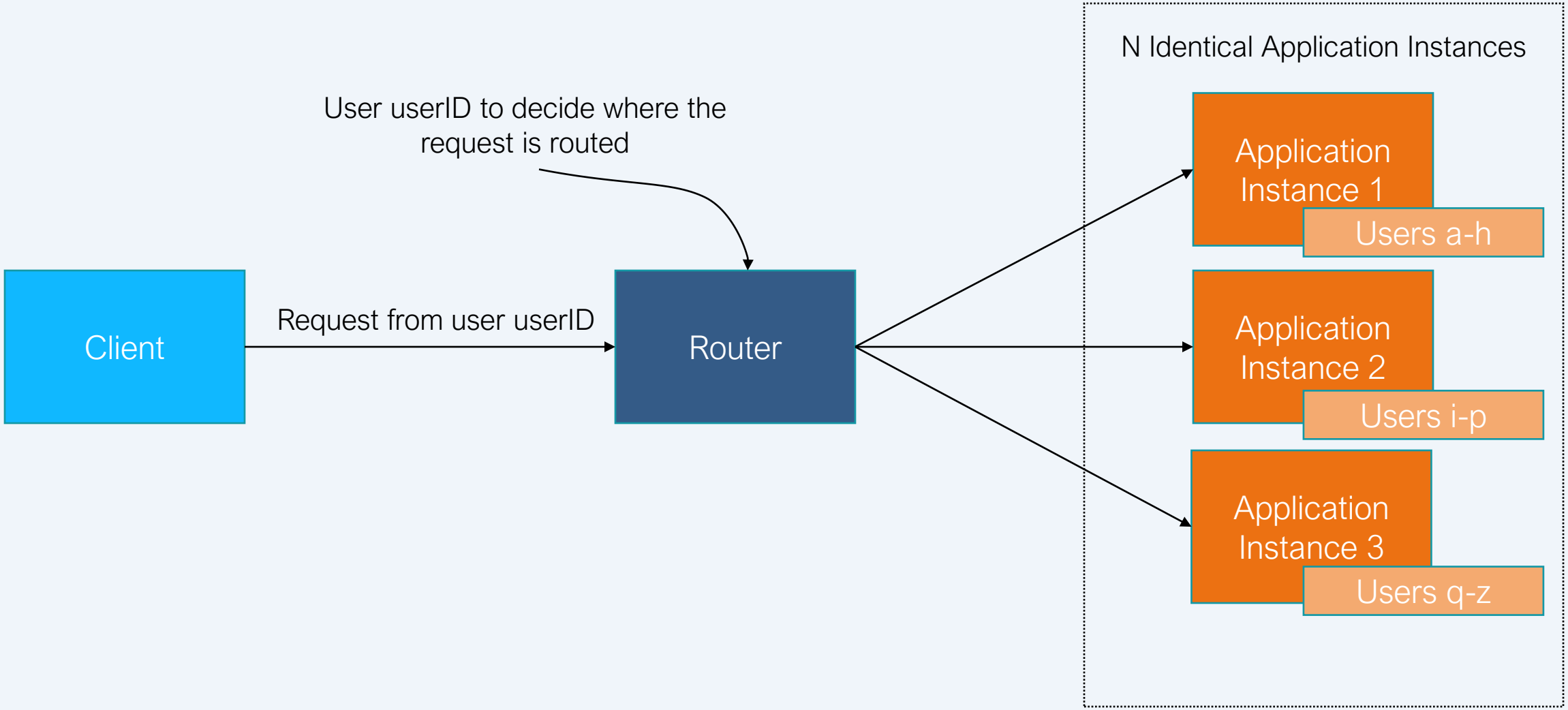
Z-scale: data partitioning. Scale by splitting similar things



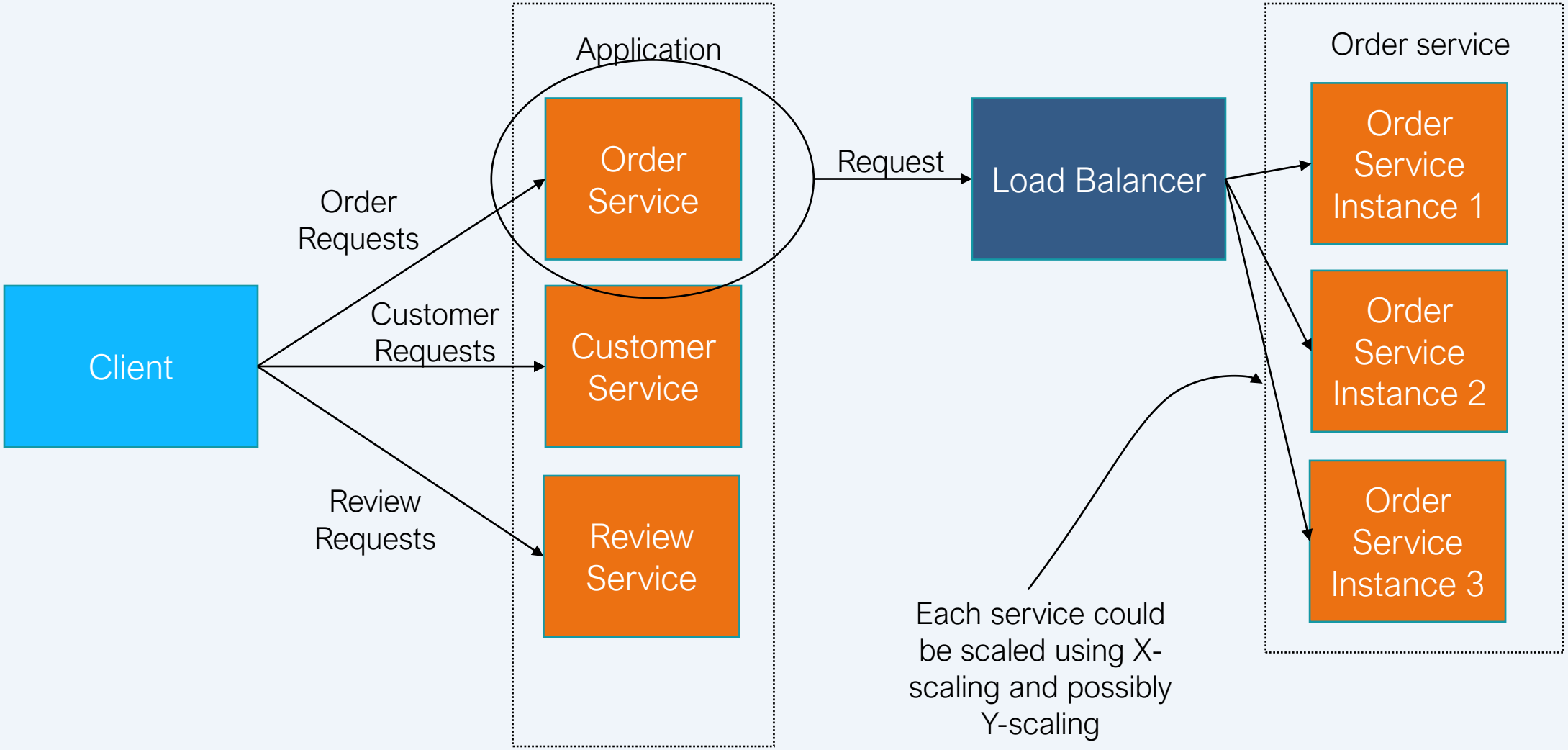
X-scaling



Z-scaling



Y-scaling



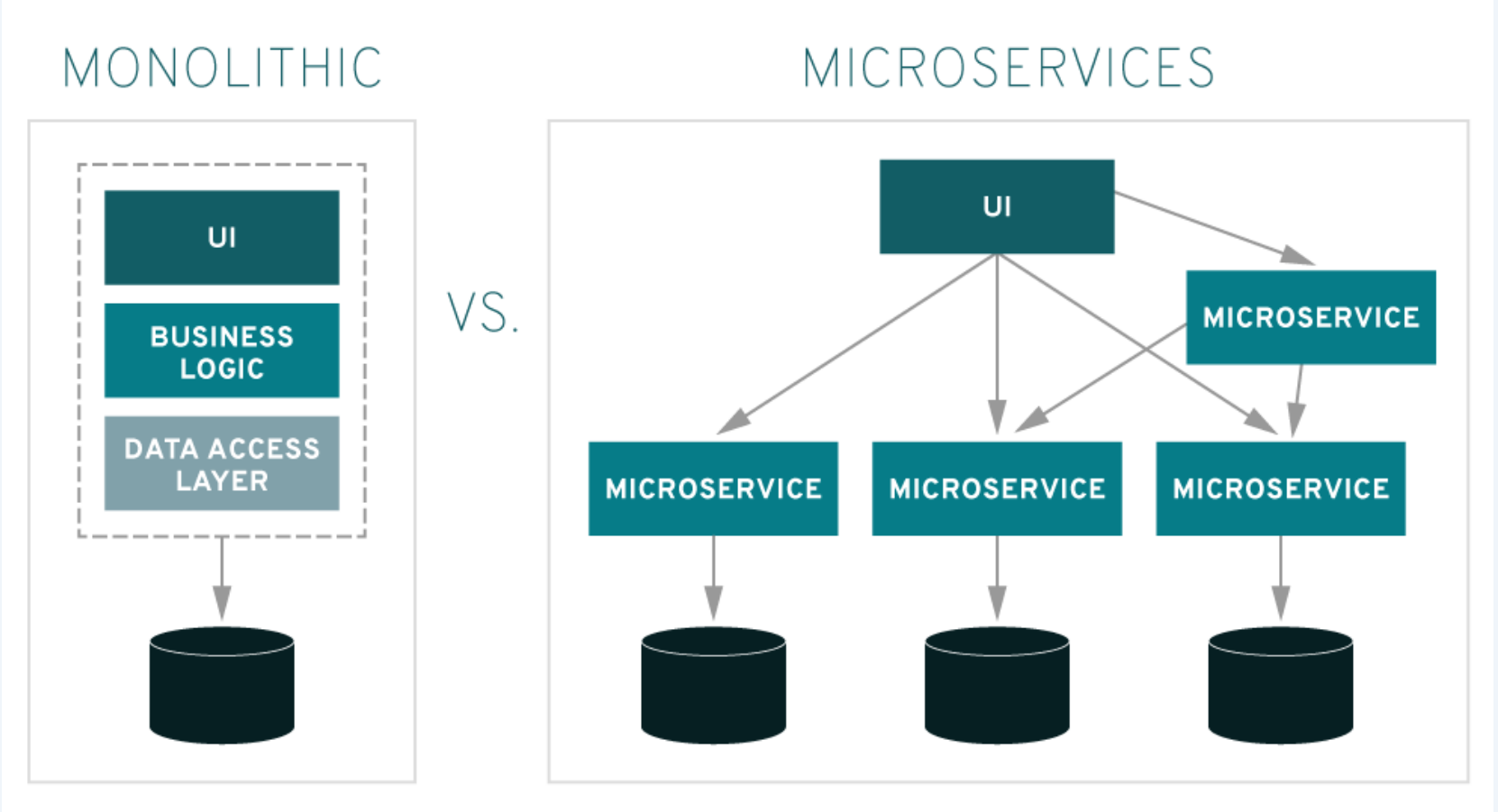
The takeaway

- You should probably start monolithic … but you probably won't finish there!
  - You will hit development, testing and deployment problems
  - Also you may wind up stuck with code on a legacy platform no-one supports
    - Anyone remember Adobe Flash? Cobol? PDP11?

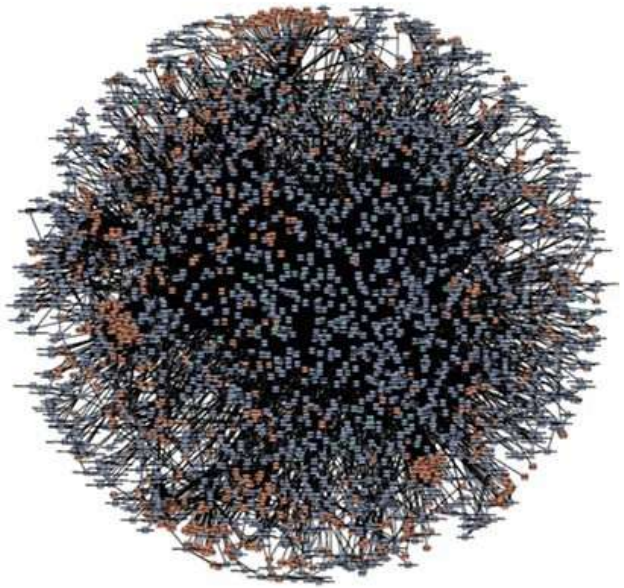
- It's worth keeping this in mind when you start.

So … what are microservices?

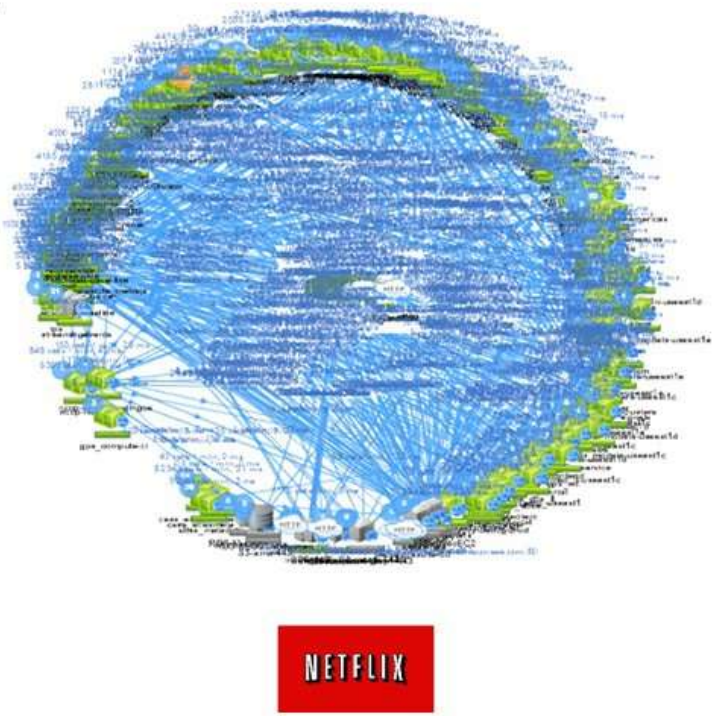
- Microservices should be:
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team



Microservices Can Scale

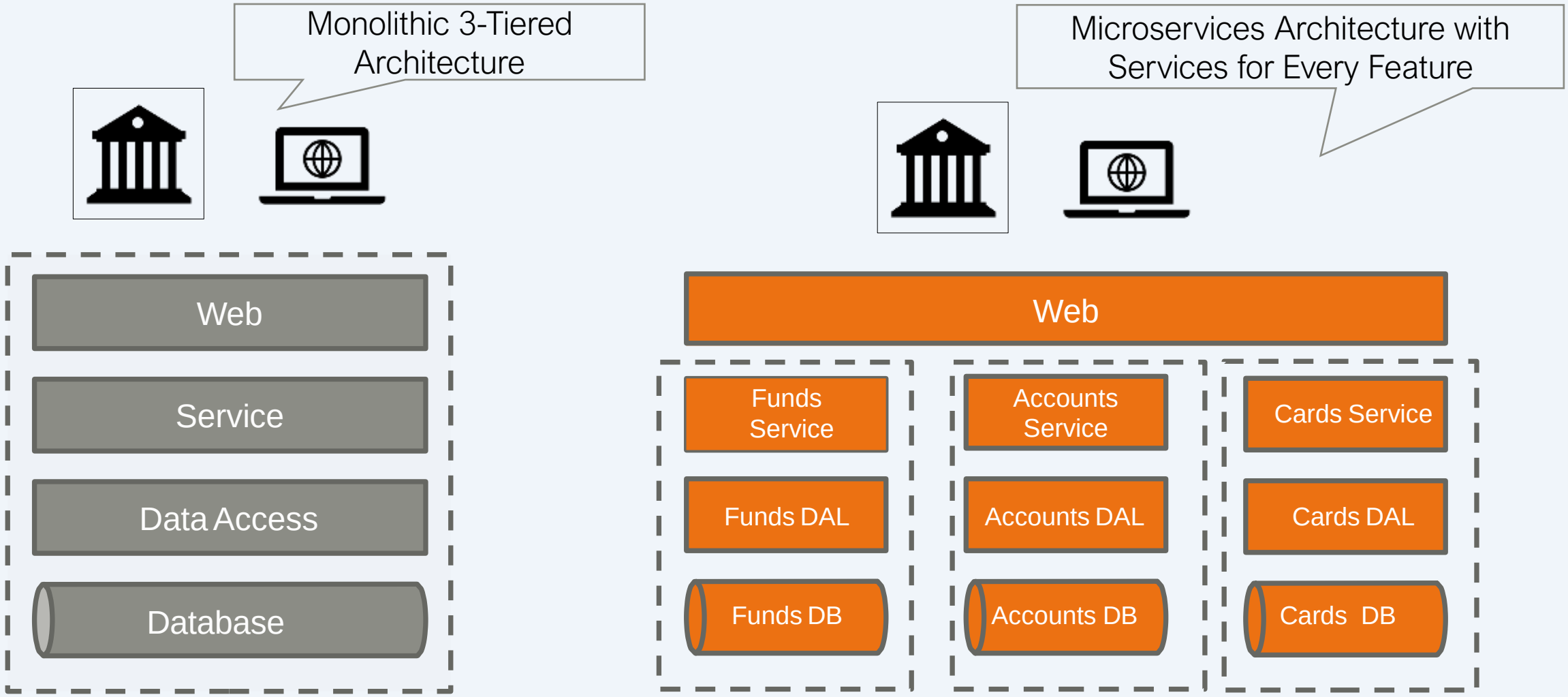


amazon.com



NETFLIX

Example: Banking Microservices



Group Exercise

In groups, make three lists:

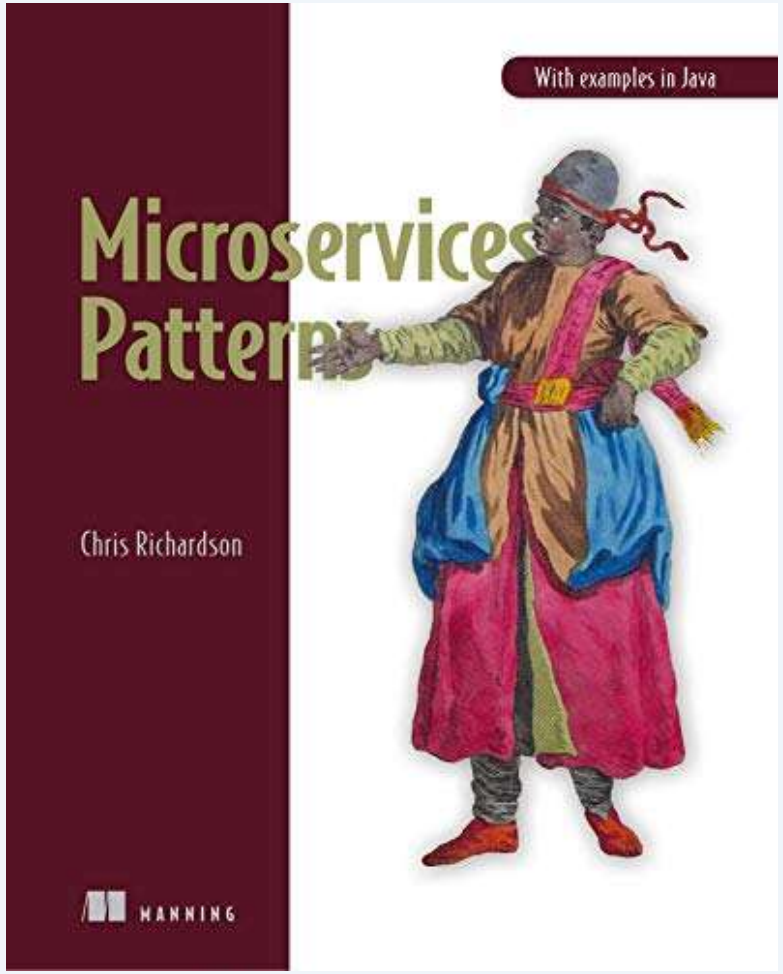
1. What would a microservices architecture make easier ?
2. What would a microservices architecture make harder?
3. What problems would you encounter in microservices that don't even exist in a monolithic architecture?

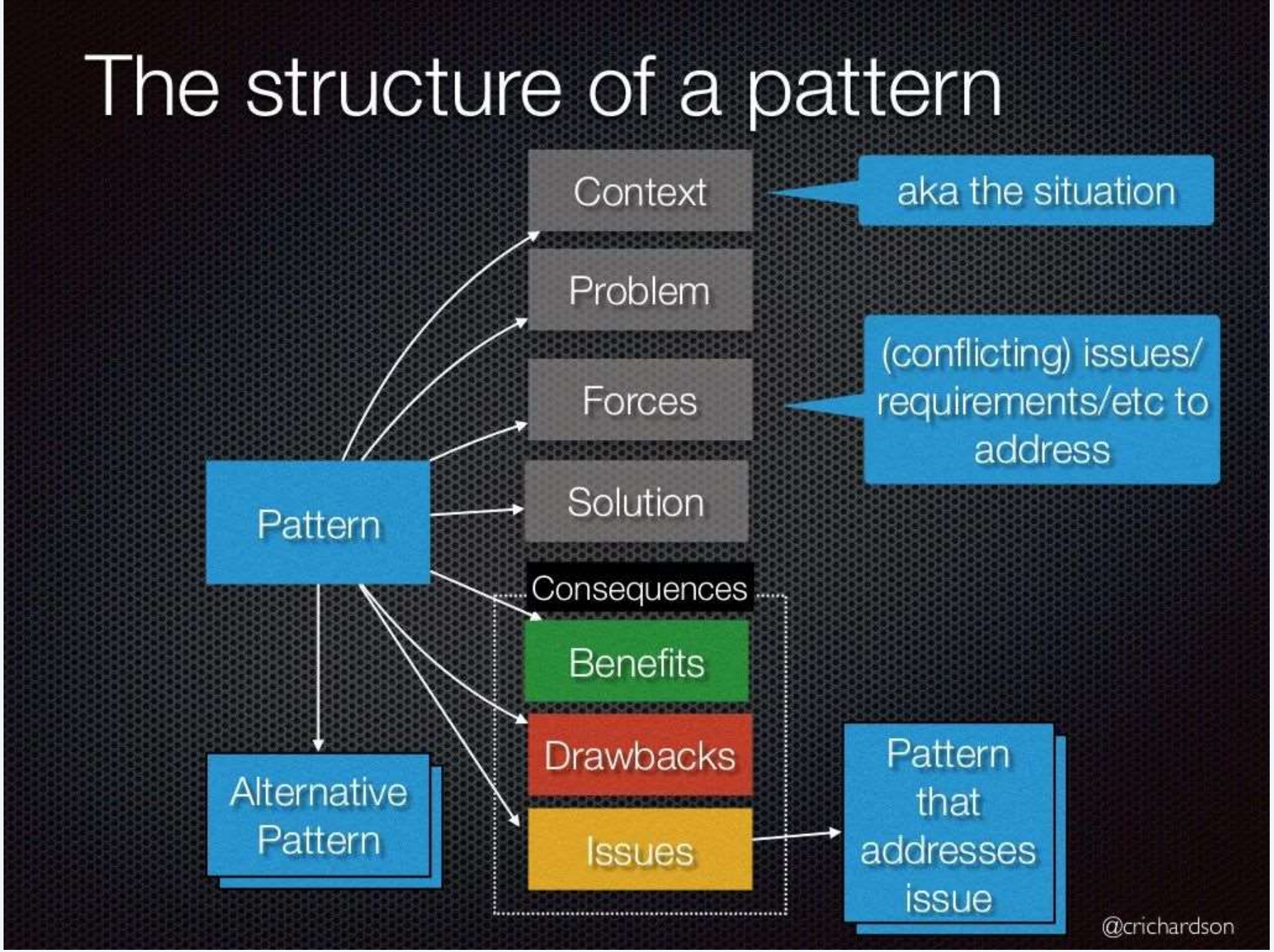
Microservices Patterns

Microservices Patterns

- Much of this section (and course) will be based on this!
- There is also a lot of useful info on Chris Richardson's website:
- https://microservices.io/index.html
  - Descriptions of all the patterns
  - Presentations and links to examples

- Derived from:
  - "A Pattern Language: Towns, Buildings, Construction" (OUP 1977, Christopher Alexander)
  - Design Patterns: Elements of Reusable Object-Oriented Software (Addison Wesley 1994, Gamma, Helm et al.





"A pattern language for microservices", Chris Richardson, June 2021

Pattern Language

- A *Pattern* describes
  - A context (a market opportunity for selling chicken feed)
  - A problem (how do we design an application for selling chicken feed)
  - The forces associated with that problem
    - We must bring our application to market quickly
    - We must be able to roll out updates continually
    - Our application must be secure
    - The codebase should be easily understandable
    - We want to be able to quickly adopt new technologies
    - …
  - The solution proposed by the pattern (microservices architecture)
  - The benefits of the pattern solution
  - The drawbacks of the pattern solution
  - The issues arising from those drawbacks and patterns that can address those issues.

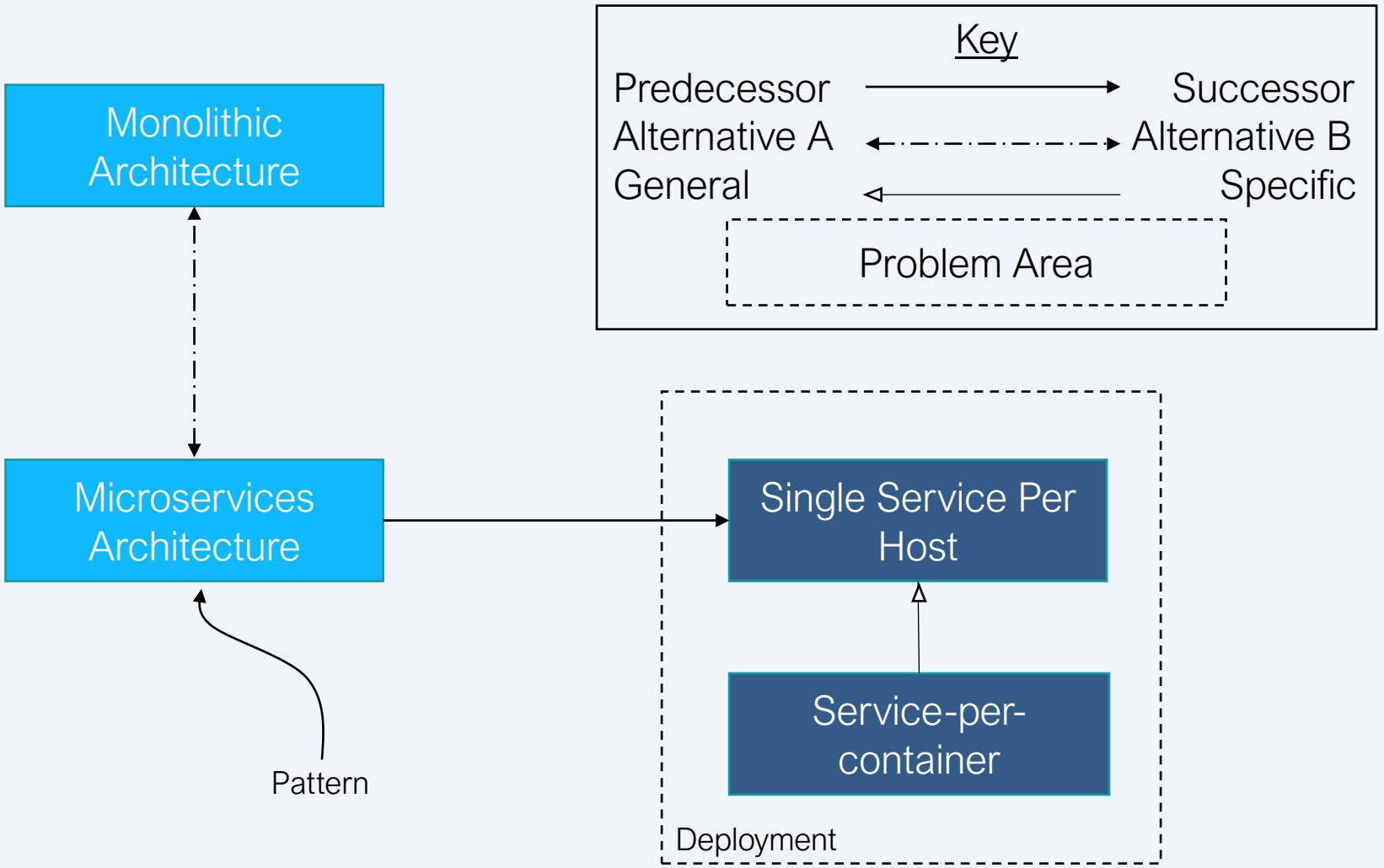
Related Patterns

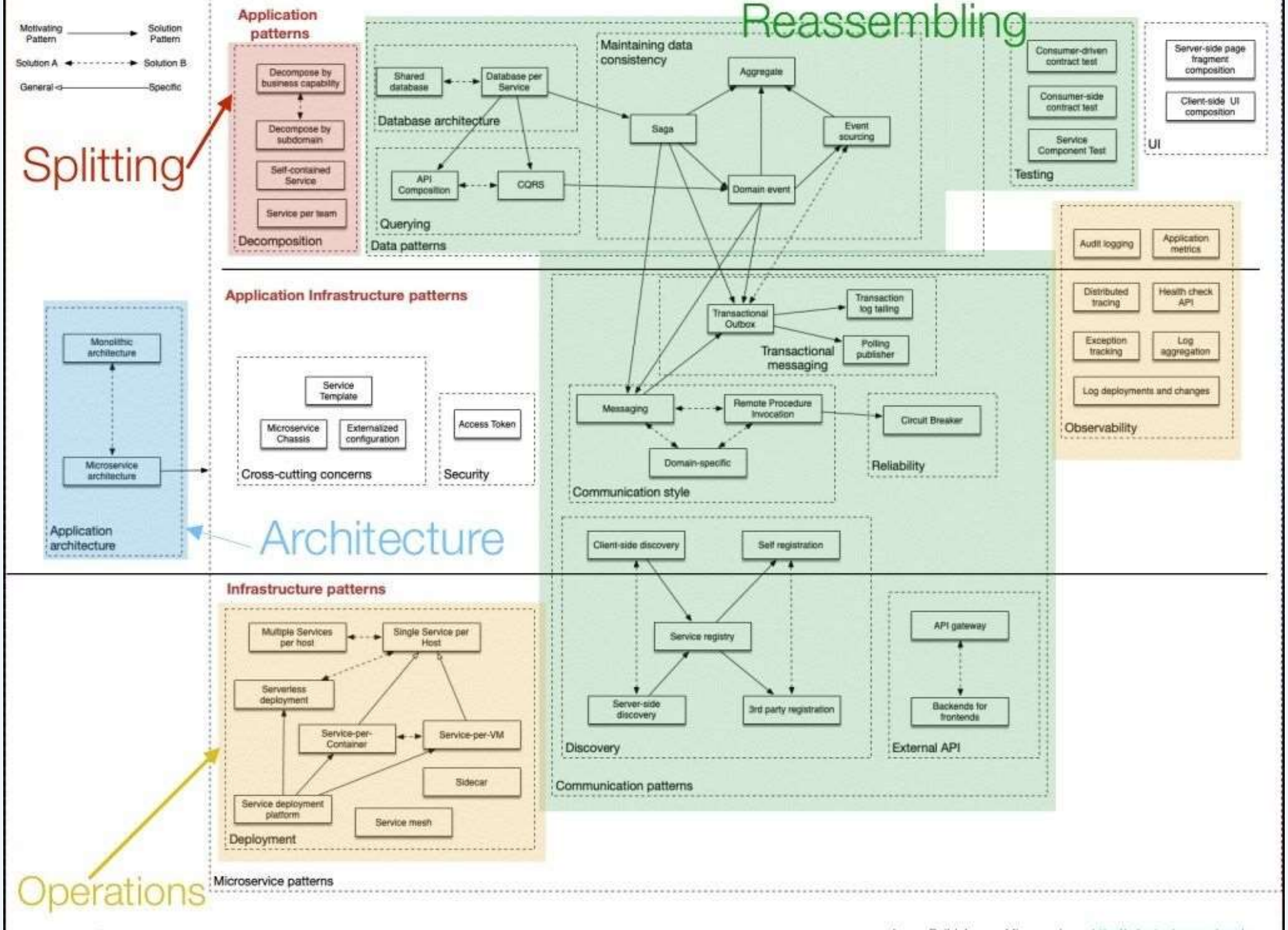
- Every pattern also gives rise to related patterns
  - Predecessor pattern: a pattern that motivates the need for this pattern
    - E.g. the microservices architecture pattern is the predecessor to all the other patterns in the pattern language except the monolithic architecture pattern
  - Successor pattern: a pattern that solves an issue in the resultant context of this pattern.
    - E.g. the database per service pattern raises the issue of cross-service queries, which has the CQRS pattern and the API composition pattern as successors.
  - Alternative patterns: a pattern giving an alternative solution to this problem
    - E.g. the monolithic architecture pattern is an alternative solution to the microservices architecture pattern.

Related Patterns

- Every pattern also gives rise to related patterns
  - Generalization pattern: a pattern that is a general solution to a problem
    - E.g. the single service per host pattern is a general solution to the service decoupling issue
  - Specialization pattern: a pattern that is a specialized form of a particular pattern
    - E.g. the deploy a service as a container pattern is a particular form of the single server per host pattern.

Pattern Language Graphical Representation




Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Application patterns
Splitting
Decompose by business capability
Decompose by subdomain
Self-contained Service
Service per team
Decomposition
Shared database
Database per Service
Database architecture
Maintaining data consistency
Aggregate
Saga
Event sourcing
API Composition
CQRS
Domain event
Querying
Data patterns
Reassembling
Consumer-driven contract test
Consumer-side contract test
Service Component Test
Testing
Server-side page fragment composition
Client-side UI composition
UI
Audit logging
Application metrics
Distributed tracing
Health check API
Exception tracking
Log aggregation
Log deployments and changes
Observability
Application Infrastructure patterns
Monolithic architecture
Microservice architecture
Application architecture
Architecture
Service Template
Microservice Chassis
Externalized configuration
Cross-cutting concerns
Access Token
Security
Transactional Outbox
Transaction log tailing
Polling publisher
Transactional messaging
Messaging
Remote Procedure Invocation
Domain-specific
Communication style
Circuit Breaker
Reliability
Client-side discovery
Self registration
Service registry
Server-side discovery
3rd party registration
Discovery
API gateway
Backends for frontends
External API
Communication patterns
Infrastructure patterns
Multiple Services per host
Single Service per Host
Serverless deployment
Service-per-Container
Service-per-VM
Sidecar
Service deployment platform
Service mesh
Deployment
Operations
Microservice patterns

What Does a Pattern Language Do?

- A way to look at our architecture objectively
- A way to catalogue and address issues that arise
- A way to think through what we are really trying to achieve

There are no silver bullets – any pattern that has successors solving its issues in a reasonable way is a solution.
We need to weigh the positives and negatives and decide!

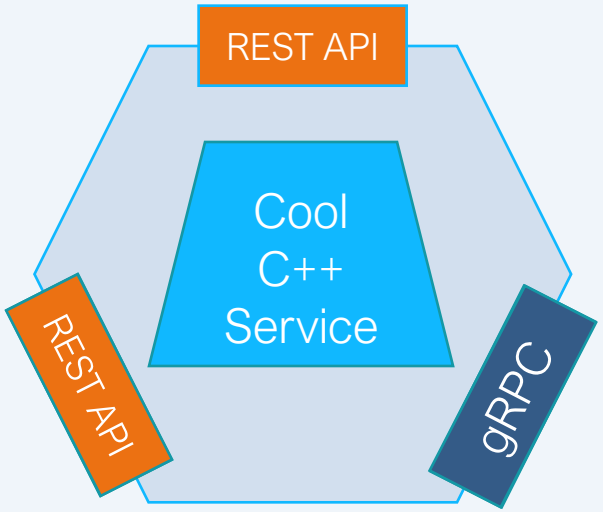
Summary

- What are Microservices?
- Microservices Patterns

Questions or Comments?

Appendix: Microservices with C++

- The microservices architecture pattern doesn't care which language the individual service is written in.
  - It could be Java, Go, Rust, Python … or C++
- What is important is that your services emit and receive API calls, procedure calls and/or events in a way that is intelligible to your infrastructure.



Some initial comments

- C++ is not a major player in the microservices architecture world
  - Lack of frameworks and support that languages like Java and Go have.
- Gluing together microservices is not a task that plays to the strengths of C++
- A choice to go with a C++ only implementation may mean you need to re-implement and maintain code that is not on your critical business path.

What are your constraints?

- Legacy code?
  - If it is that complicated to re-implement it probably breaks the microservices architecture pattern anyway!
  - Does it actually interface in the ways I need?
  - What sort of wrapper would I need to put around it to function in a microservices architecture?
- Management directives?
  - This is always tricky – you might need to be prepared to make a case for a change, and willing to lose your case!
- Time pressures?
  - Always!
  - But be sure that reimplementing is really as hard as you think.

Why do I want to use C++?

- Personal preference?
  - Very strong point – it's hard to get proficient in a new language!
- It fits the unique requirements of my system
  - Then you have to use it!

Remember – microservices are loosely coupled

- Sometimes C++ will be the best (or only) available choice for a service.
- That doesn't dictate the language for all your services
  - Polyglot architectures are quite acceptable in the microservices world.
- Pay close attention to the messaging infrastructure posited
  - REST is very common, but it is a synchronous technology that can couple your services more tightly than you'd like.

Some Existing C++ libraries

- A lot of the messaging infrastructures are built to integrate with C++ code bases
  - REST: Microsoft “supports” and opensource C++ library for REST and other microservices features at https://github.com/Microsoft/cpprestsdk, however they do not recommend its use for new projects.
  - gRPC: C++ is actively and currently supported. https://grpc.io/
  - Apache Thrift has a similar IDL approach to gRPC, and automatically generates code in C++ (a good blog post and tutorial Polyglot Microservices: Application Integration With Apache Thrift - DZone Cloud)
- A list of CPP libraries (not sure how up to date)
  - A list of open source C++ libraries - cppreference.com

- **BE CAREFUL** – things like gRPC and Apache Thrift are well-supported, but smaller projects may not have the support needed for commercial applications.