Microsoft

Microservices Basics

1.Foundation

Hiro Tarusawa
Service Engineer
Fast Track for Azure. Microsoft

FTA Live! Microservices Basics Trilogy

- Objective:
  - Learn modern architecture style
  - Design application with microservices
- Three classes:
  - Foundation ☞ This session
    - Understand the concept of cloud native computing and microservices
  - Modeling - Logical Design
    - Understand the overview of service modeling
  - Design with Azure - Physical Design
    - Understand the holistic view of microservice-based application on Azure

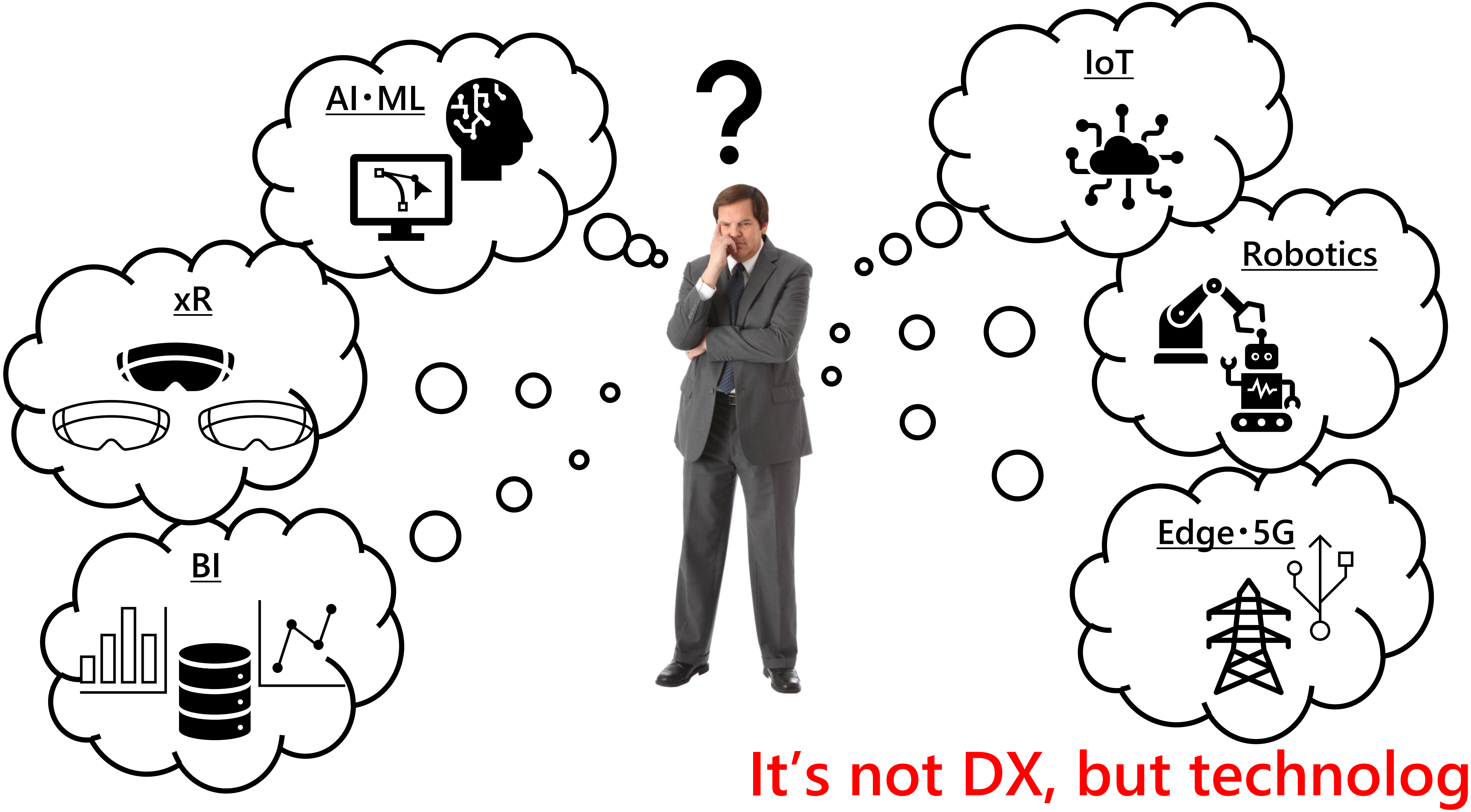
Agenda

1. Digital Transformation and Cloud Native Computing
2. Microservices Concept
3. Microservice Architecture Overview
4. Points for Success

Microsoft

1. Digital Transformation and Cloud Native Computing

What image do you have of DX



It’s not DX, but technologies ?

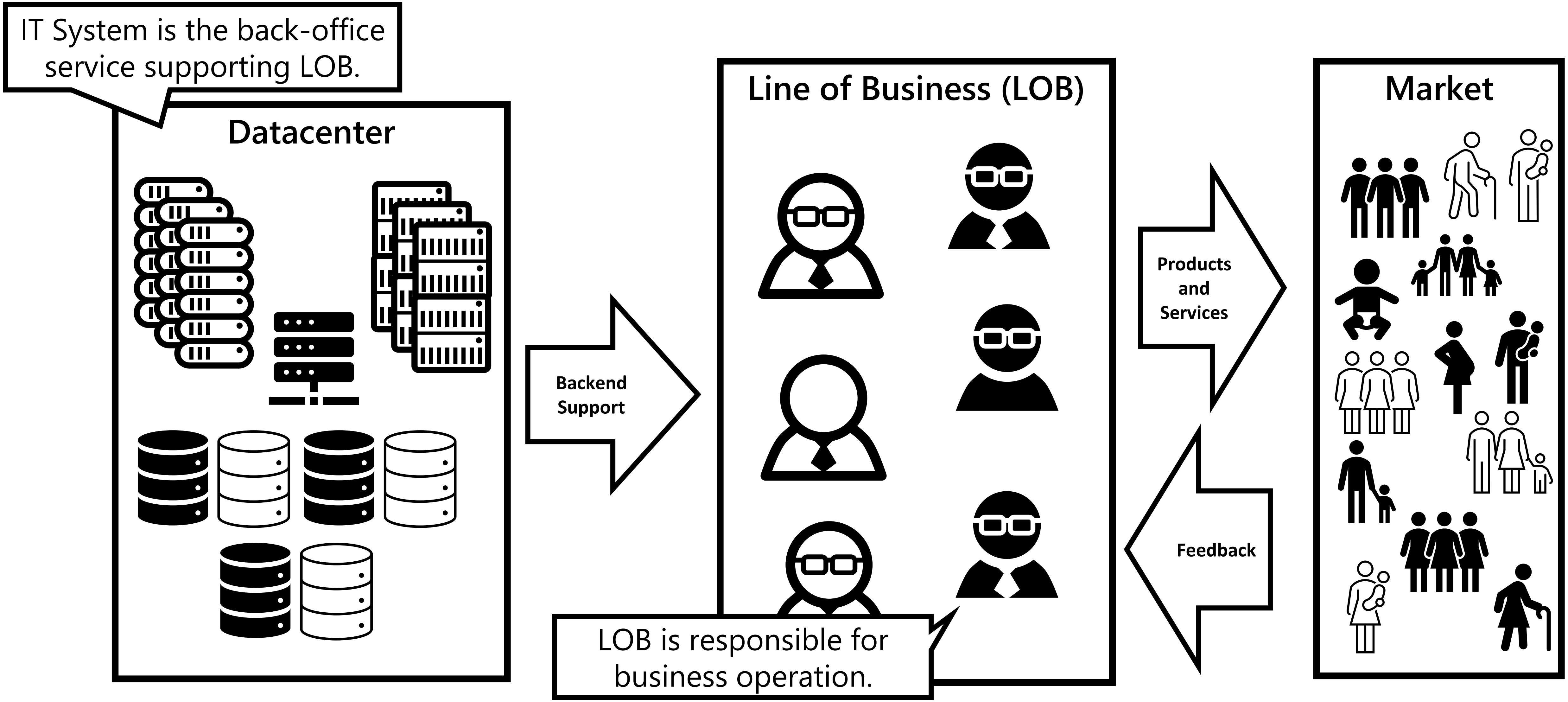
The Essence of DX

- **Adoption of digital technology**
- **Digitalization of non-digital business**

Digital transformation (DX) is the adoption of digital technology by an organization to **digitize non-digital products, services or operations**. The goal for its implementation is to **increase value through innovation, invention, customer experience or efficiency**.

Source : https://en.wikipedia.org/wiki/Digital_transformation

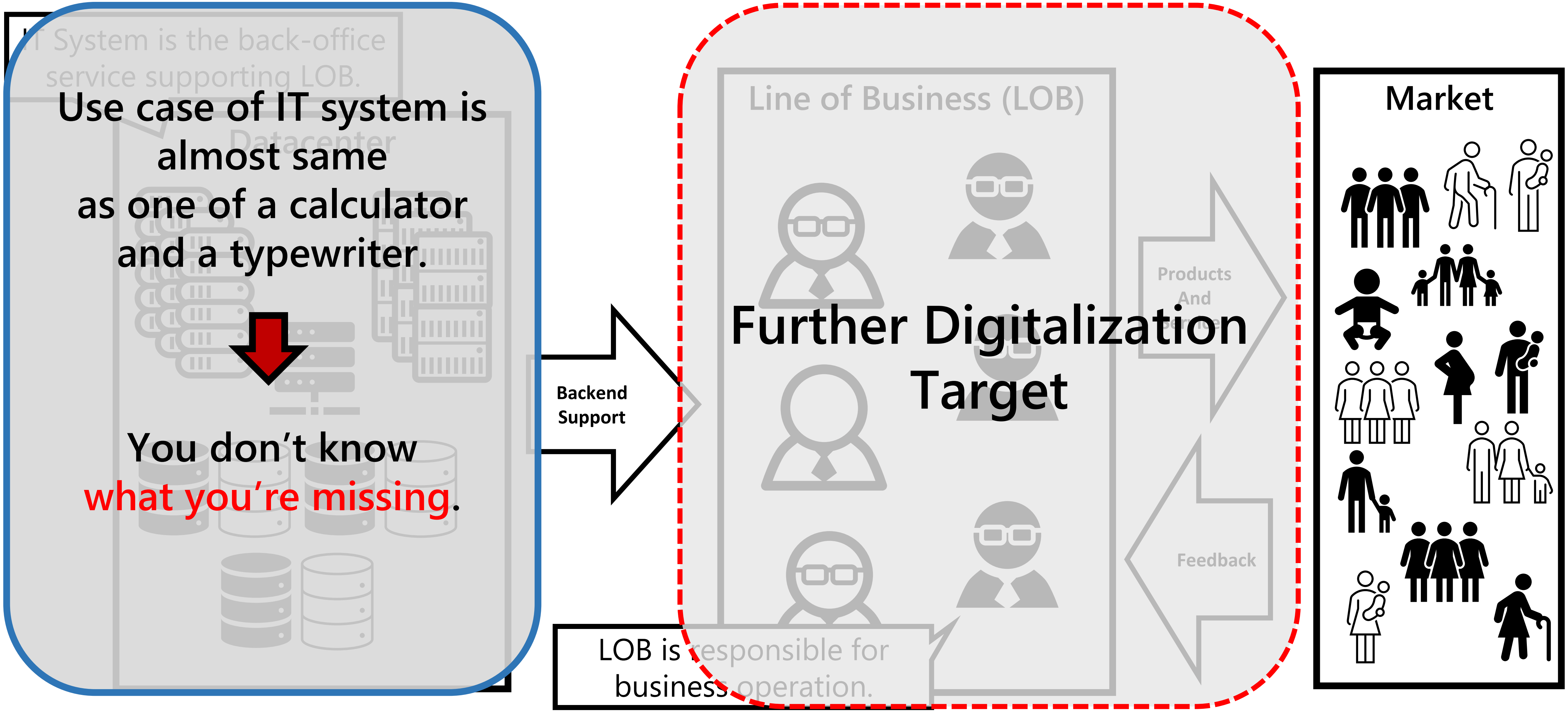
IT Adoption Model : As-is



IT System is the back-office service supporting LOB.

LOB is responsible for business operation.

IT Adoption Model : As-is



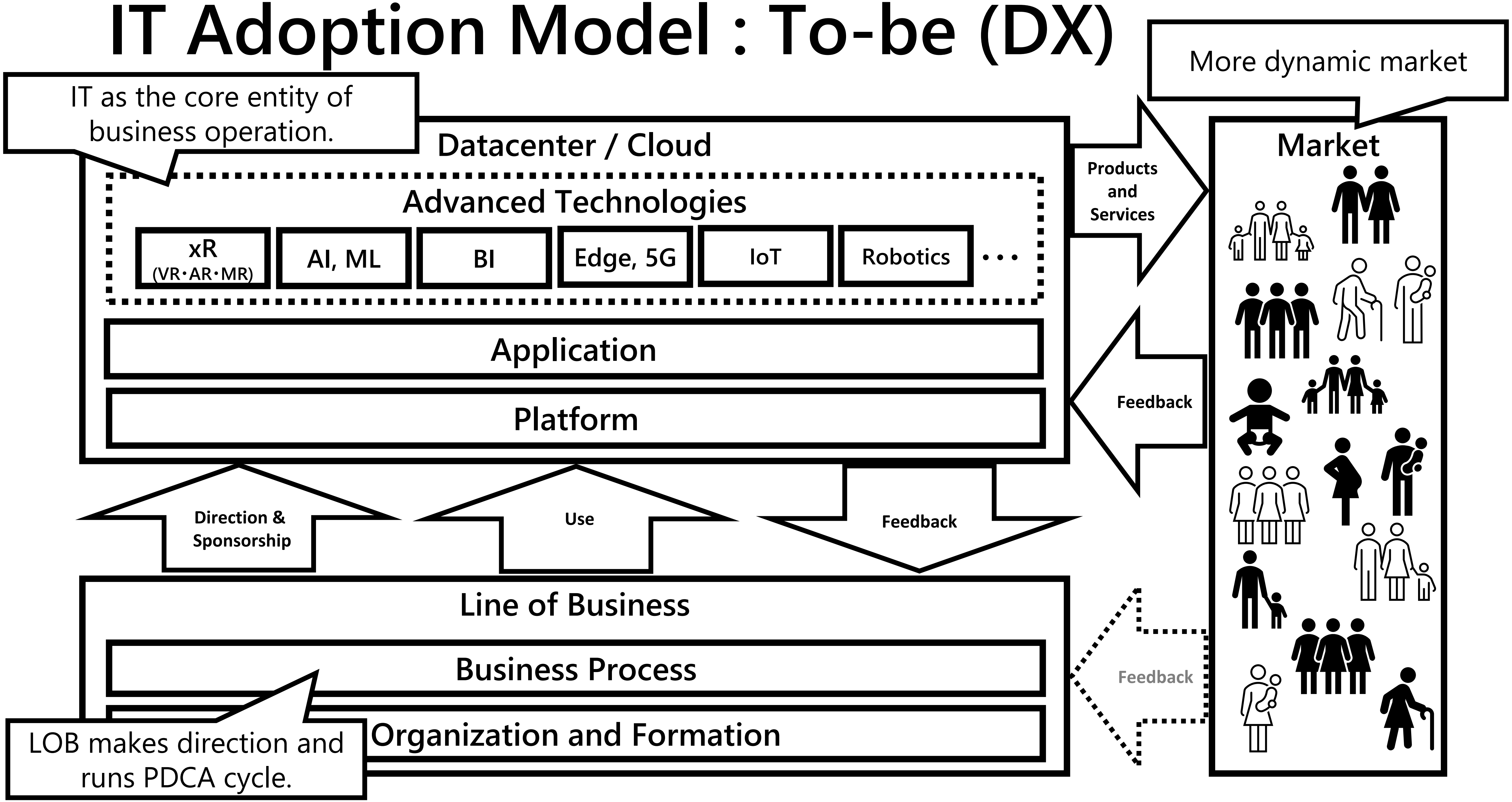
Use case of IT system is almost same as one of a calculator and a typewriter.

You don't know what you're missing.

Further Digitalization Target

IT System is the back-office service supporting LOB.
Datacenter
Backend Support
Line of Business (LOB)
Products And Service
Feedback
Market
LOB is responsible for business operation.

IT Adoption Model : To-be (DX)

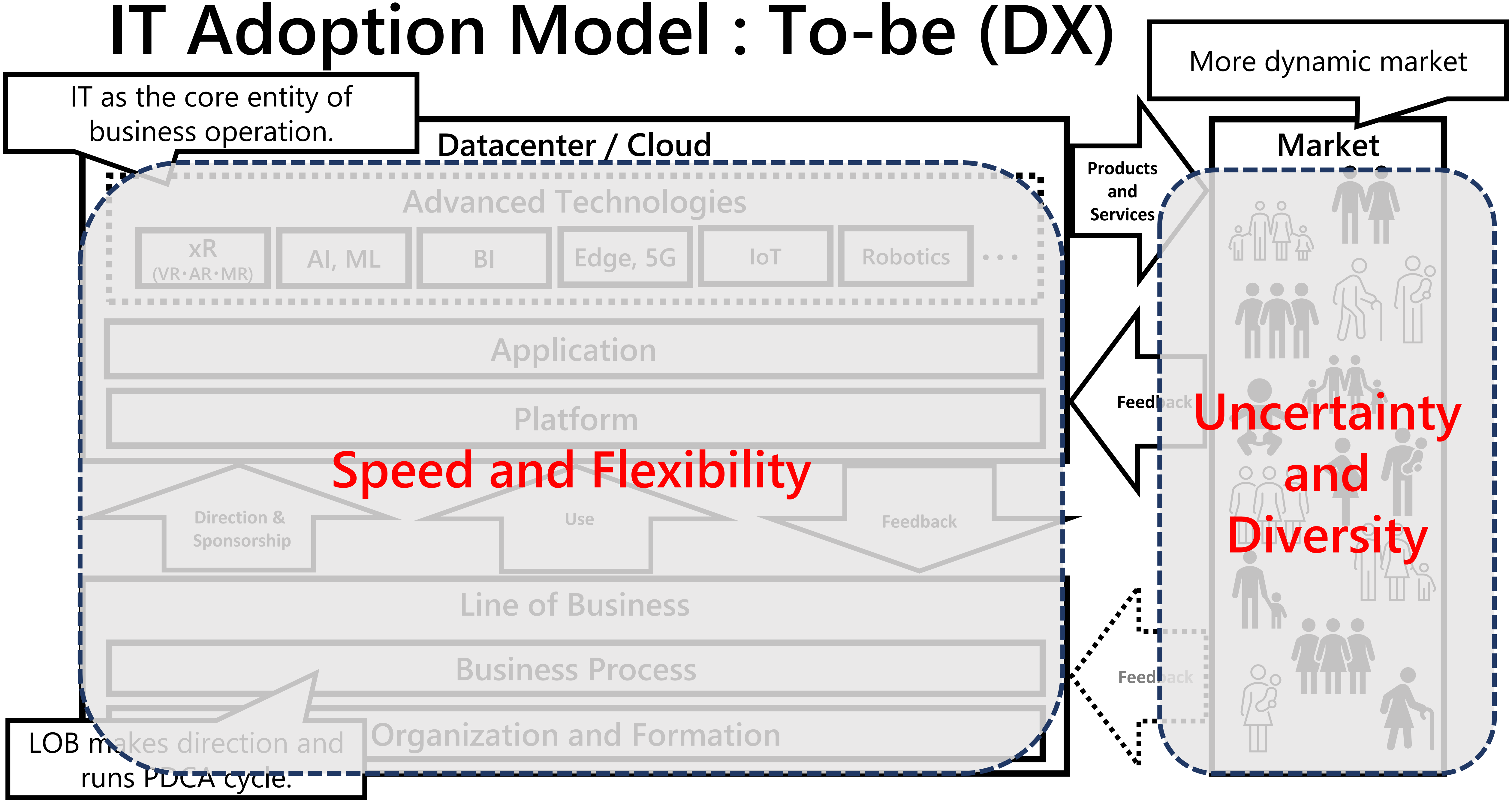


IT as the core entity of business operation.

More dynamic market

LOB makes direction and runs PDCA cycle.

IT Adoption Model : To-be (DX)



IT as the core entity of business operation.

More dynamic market

Datacenter / Cloud

Advanced Technologies

xR (VR・AR・MR)

AI, ML

BI

Edge, 5G

IoT

Robotics

・・・

Application

Platform

Speed and Flexibility

Direction & Sponsorship

Use

Feedback

Line of Business

Business Process

Organization and Formation

LOB makes direction and runs PDCA cycle.

Products and Services

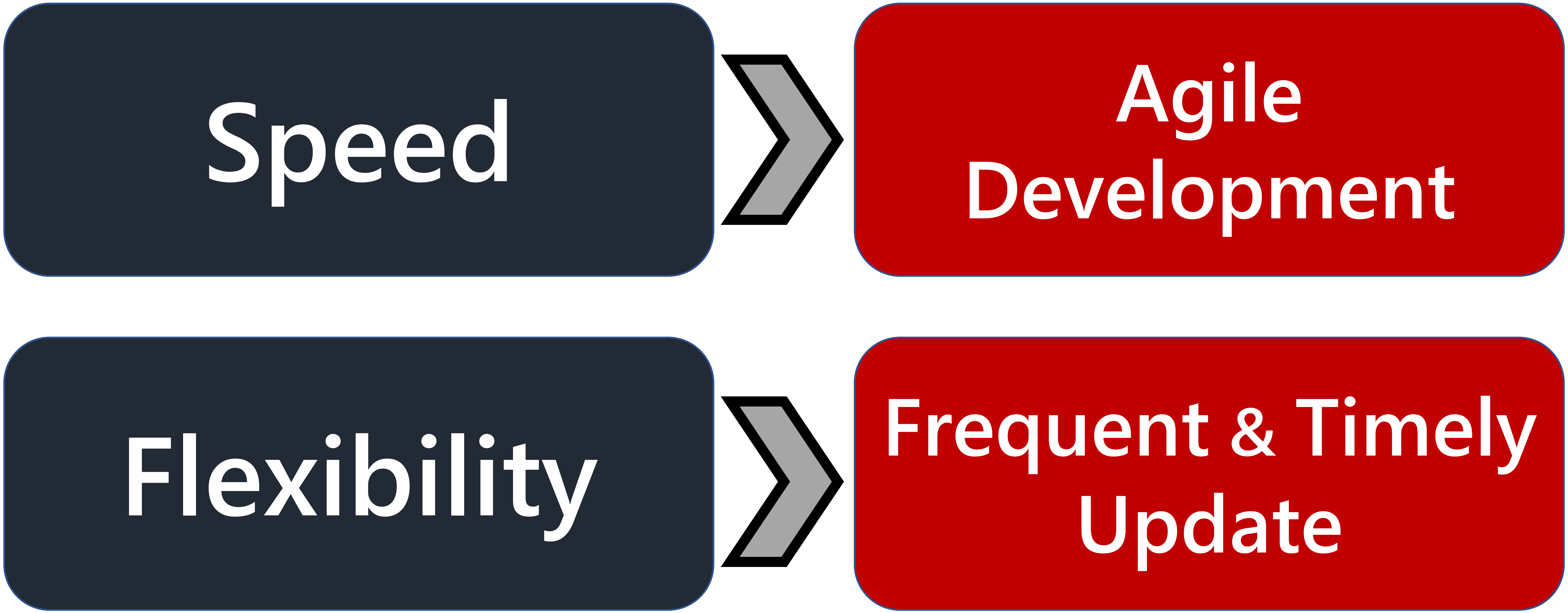
Feedback

Feedback

Market

Uncertainty and Diversity

Requirements in DX



Business Requirements

IT System Requirements

Speed

Agile Development

Flexibility

Frequent & Timely Update

Cloud Native Computing

Source : https://github.com/cncf/toc/blob/main/DEFINITION.md

What?

- ✓ Cloud native technologies empower organizations **to build and run scalable applications in modern, dynamic environments** such as public, private, and hybrid clouds.

How?

- ✓ Containers, service meshes, **microservices**, immutable infrastructure, and declarative APIs exemplify this approach.

And then?

- ✓ These techniques enable **loosely coupled systems** that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make **high-impact changes frequently and predictably with minimal toil**.

Core Technologies in Cloud Native Computing

Source : Charter in FAQ of CNCF in 2016

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

Container

- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

Orchestration

- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc.

Container

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

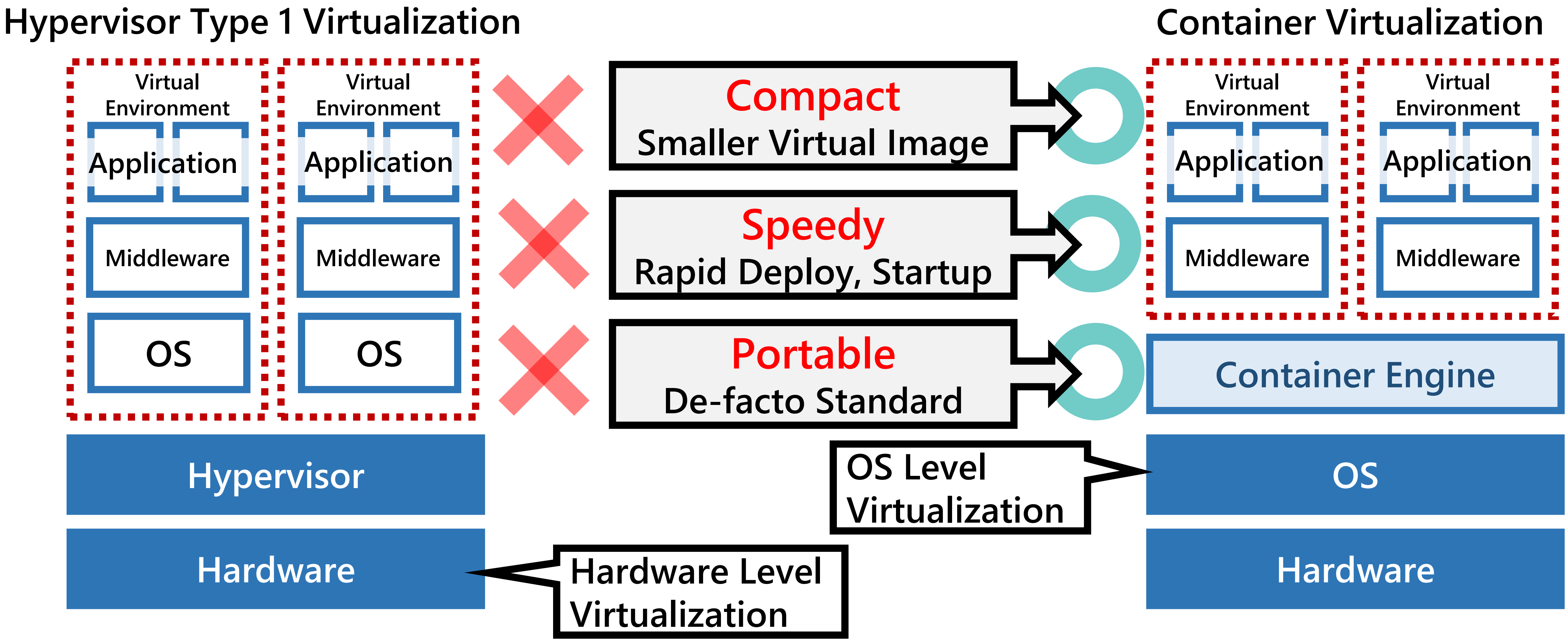
Container

- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

Orchestration

- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc.

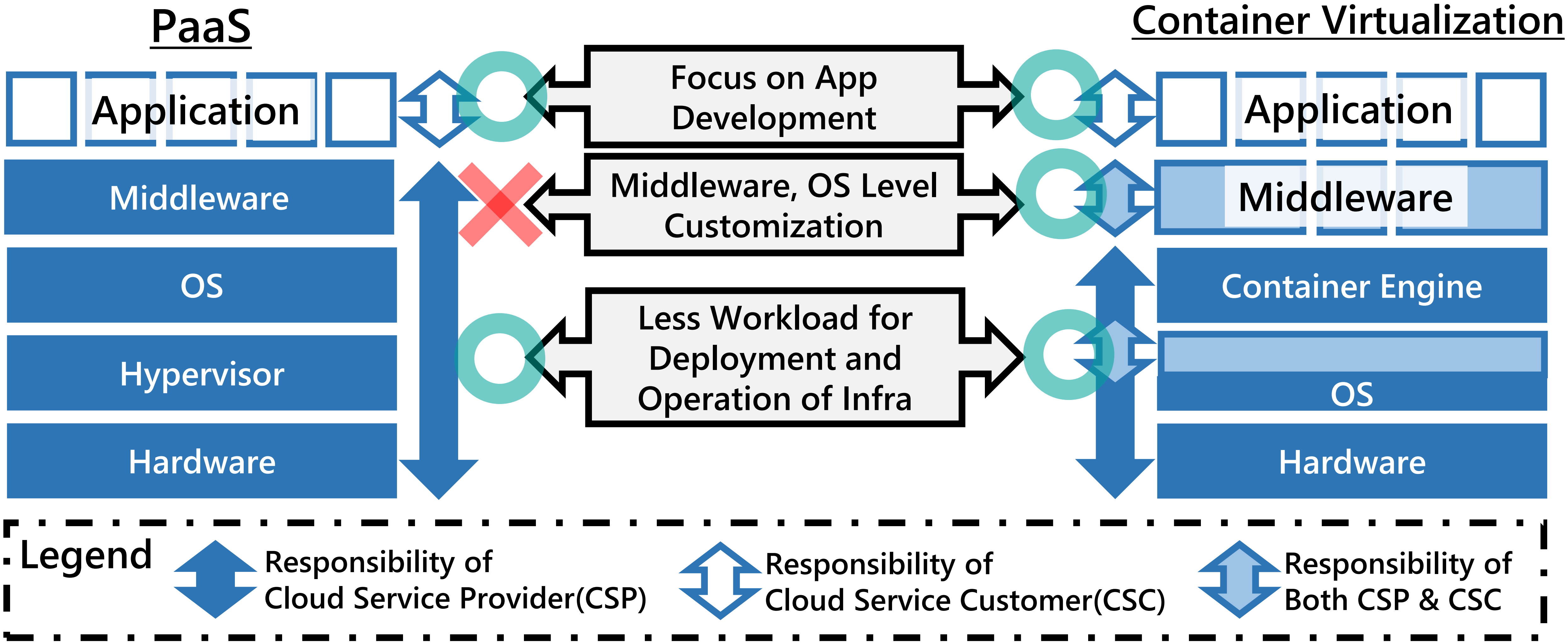
Container: Speedy, Compact, Portable



Hypervisor Type 1 Virtualization

Container Virtualization

Container: Customizability


PaaS
Container Virtualization
Application
Middleware
OS
Hypervisor
Hardware
Focus on App Development
Middleware, OS Level Customization
Less Workload for Deployment and Operation of Infra
Application
Middleware
Container Engine
OS
Hardware
Legend
Responsibility of Cloud Service Provider(CSP)
Responsibility of Cloud Service Customer(CSC)
Responsibility of Both CSP & CSC

Orchestration

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

Container

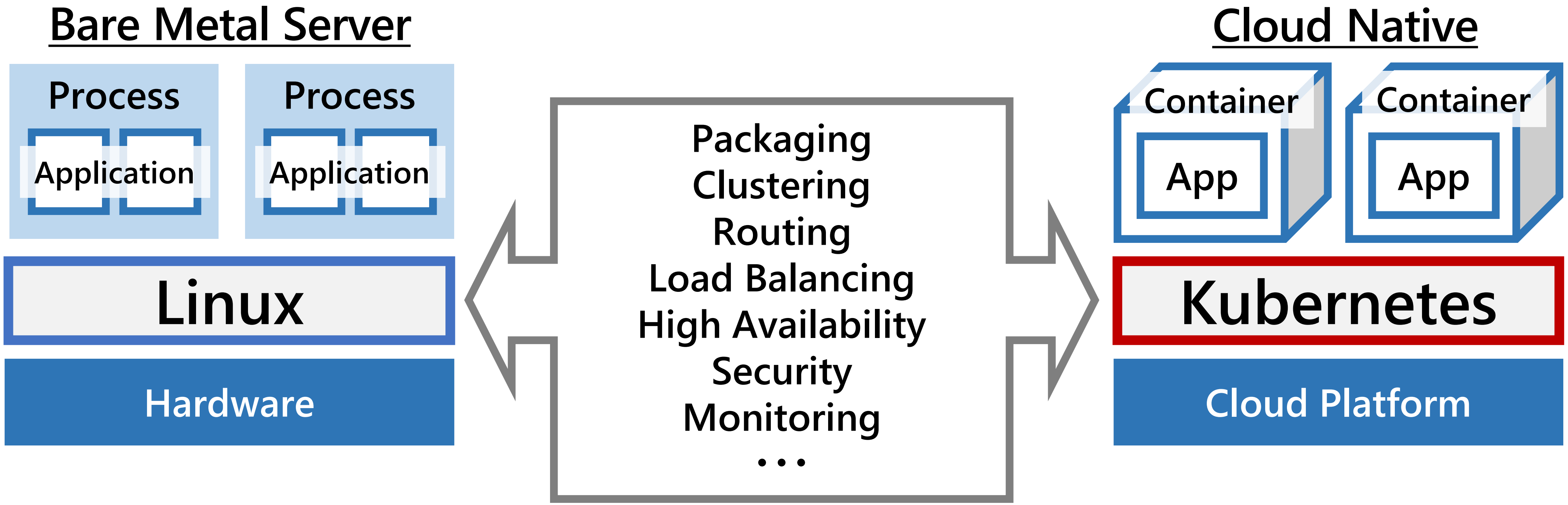
- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

Orchestration

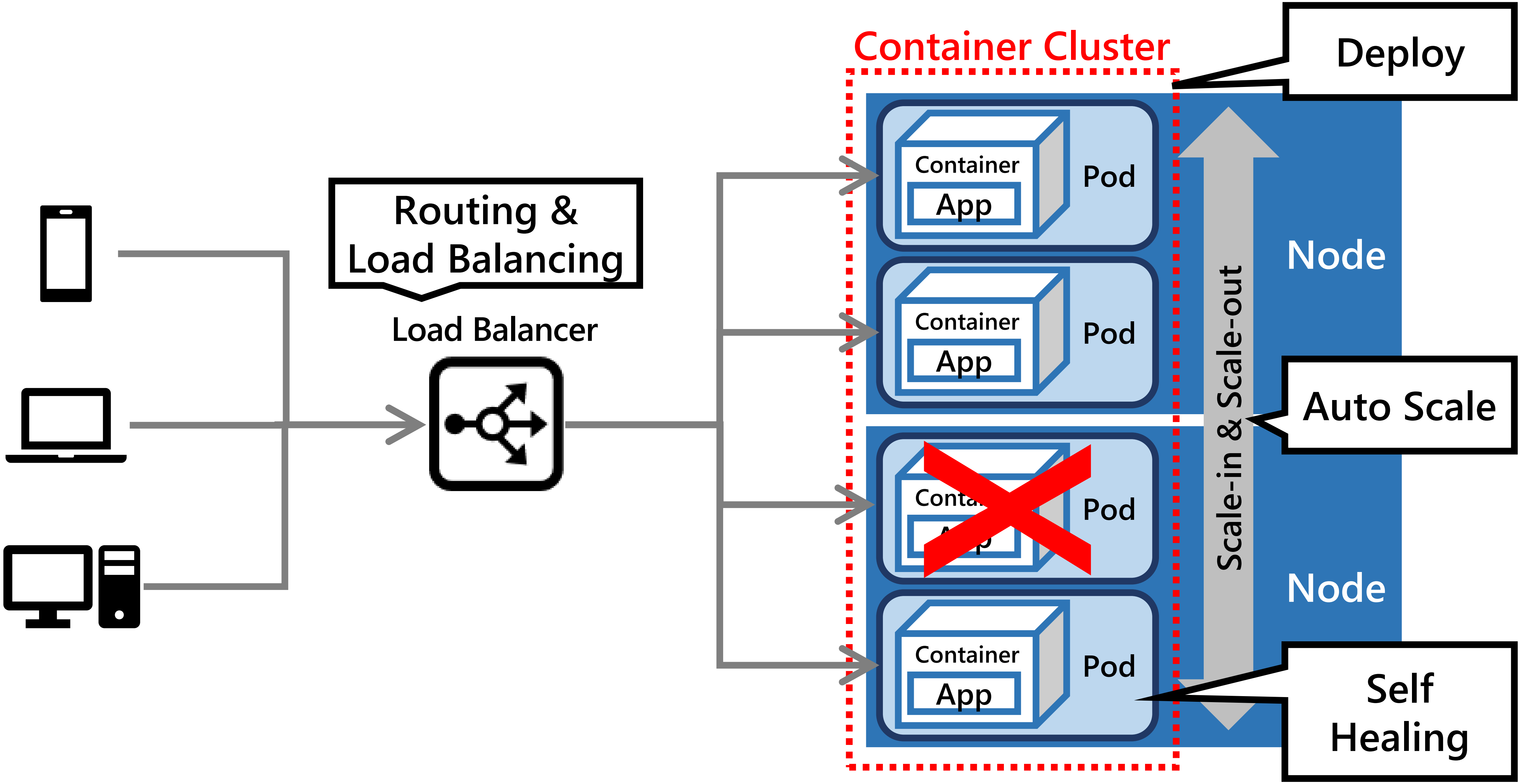
- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc.

Orchestration

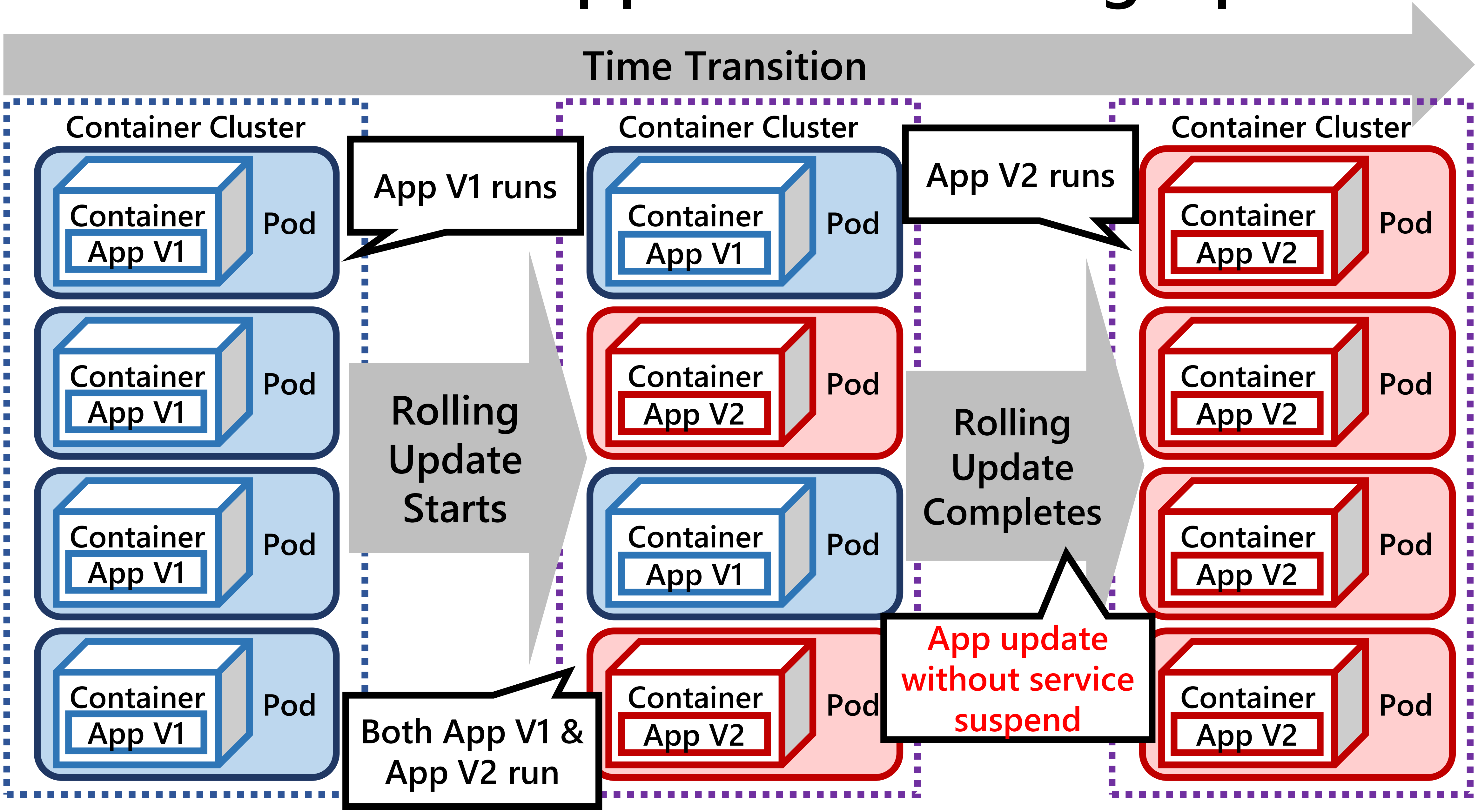
- ●Orchestration manages container clusters
- ●Kubernetes (K8s)
  - ■De-facto standard container orchestrator
- ●'Kubernetes is becoming the Linux of the cloud' tweeted by Jim Zemlin, Linux Foundation
  - ■Meaning "Kubernetes is foundation of cloud native computing"



Orchestration: Routing, Load Balancing, High Availability



Orchestration: Application Rolling Update



Microservices

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

Container

- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

Orchestration

- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc.

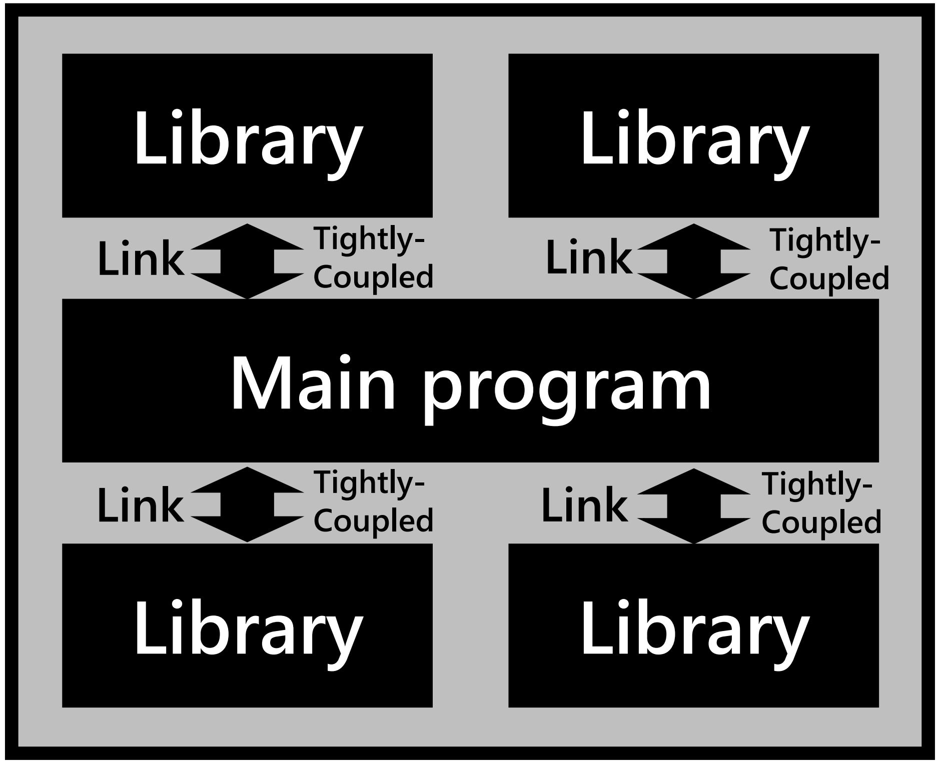
Microservices: Concept

Architecture style for modern application development and operation

- Componentization
  - Ease of maintenance and update by each application component.
- Foundation for cloud native application
  - Microservices are frequently adopted in the cloud native application development.

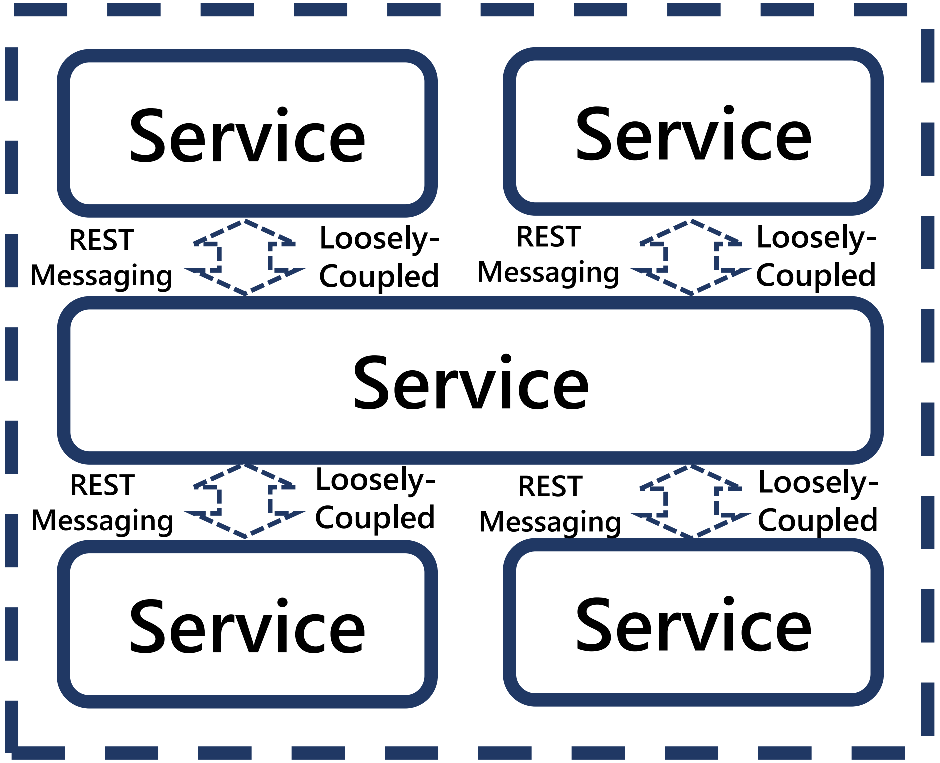
Microservices: Concept

Monolith: Library-oriented



- ✓ Each app component tightly coupled.
- ✓ Difficult to update each app component,

Microservices: Service-oriented



- ✓ Each app component loosely coupled.
- ✓ Easy to update each app component.

Beyond Container, Orchestration, and Microservices

DevOps

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

Container

- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

Orchestration

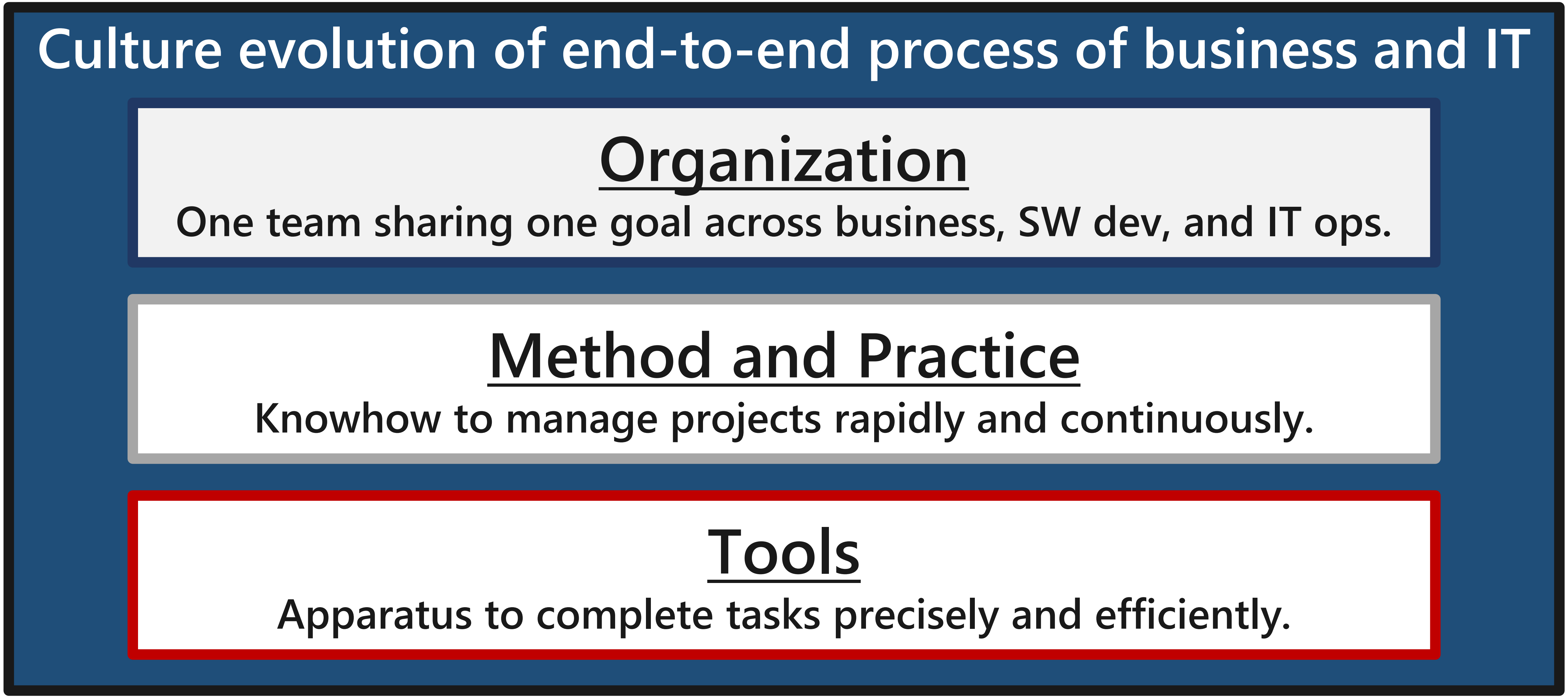
- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc

DevOps: Overview

Combination of software development and IT operations.

- Goal
  - Culture evolution of end-to-end process of business and IT.
- Benefit
  - Speed
  - Quality
  - Visibility
- How
  - Agile software development
  - Deployment Pipeline (CI/CD)

DevOps: Goal



Culture evolution of end-to-end process of business and IT

Organization

One team sharing one goal across business, SW dev, and IT ops.

Method and Practice

Knowhow to manage projects rapidly and continuously.

Tools

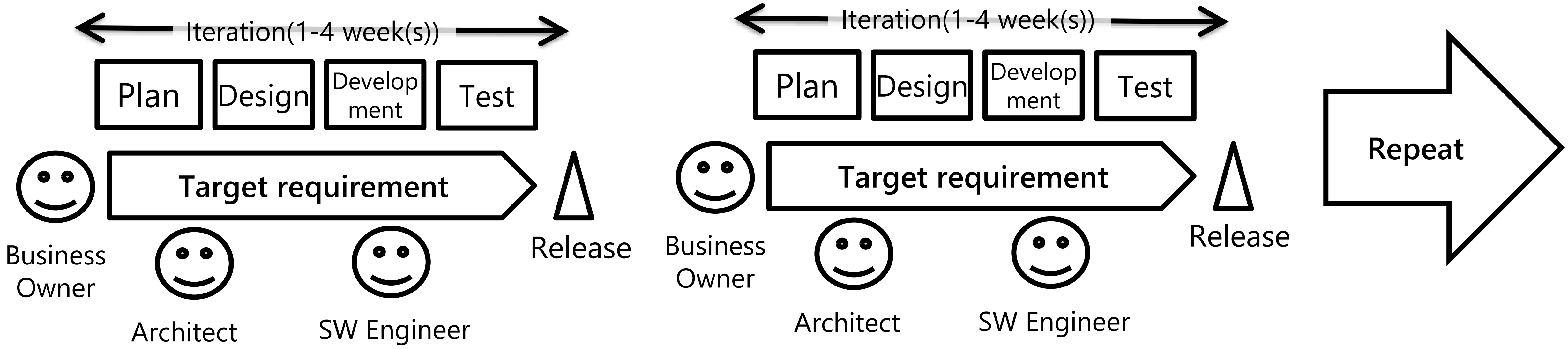
Apparatus to complete tasks precisely and efficiently.

DevOps: How -Agile Software Development-

- Software development practice
  - Collaborative effort of self-organizing and cross-functional teams
  - Early delivery
  - Continual improvement
  - Flexible responses to changes

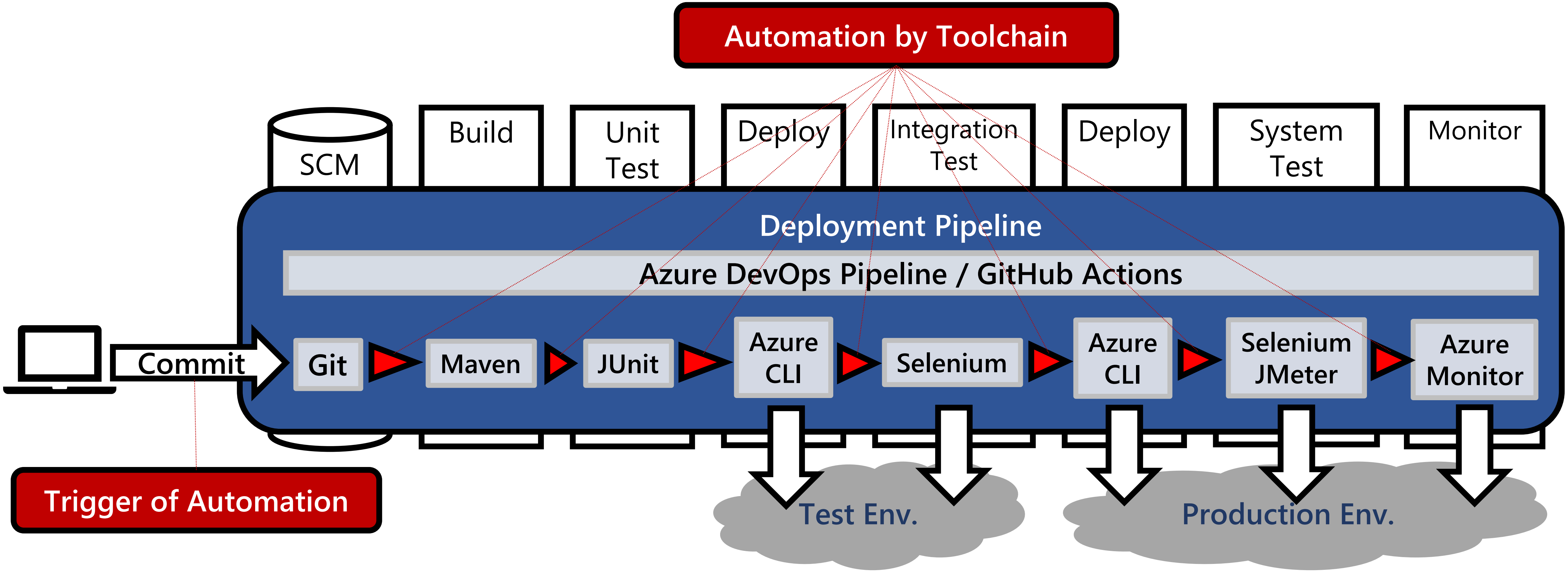
DevOps: How -Agile Software Development-

- Agile software development flow
  - Breakup of requirements.
  - Development of a small target domain by a cross functional team in a short term.
  - Repeat of short-term development cycle for enhancement.

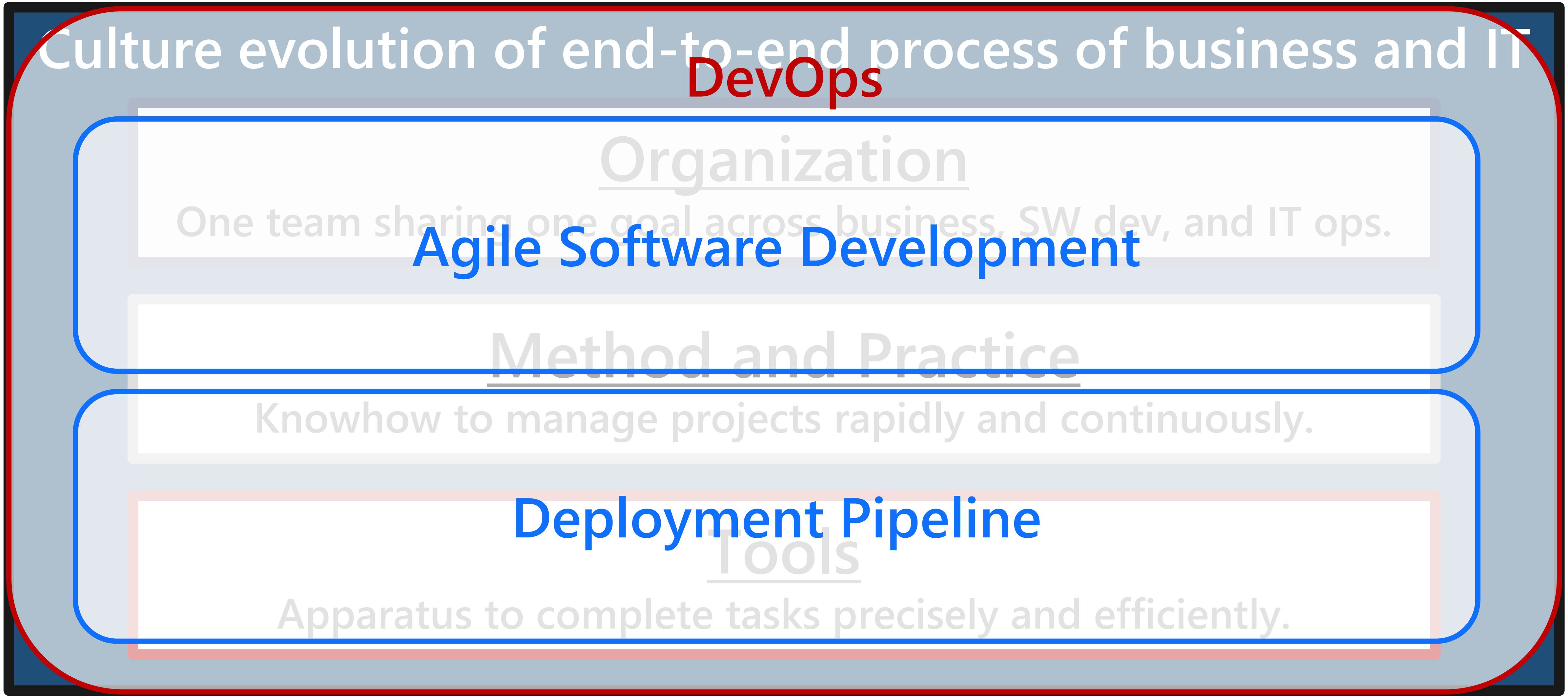


DevOps: How - Deployment Pipeline (CI/CD) -

- Fully automated process by toolchain to deploy and release any version of software promptly



DevOps: Positioning of Agile Software Development and Deployment Pipeline



Culture evolution of end-to-end process of business and IT

DevOps

Organization

One team sharing one goal across business, SW dev, and IT ops.

Agile Software Development

Method and Practice

Knowhow to manage projects rapidly and continuously.

Deployment Pipeline

Tools

Apparatus to complete tasks precisely and efficiently.

Cloud Native as Next Generation of Cloud Platform

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

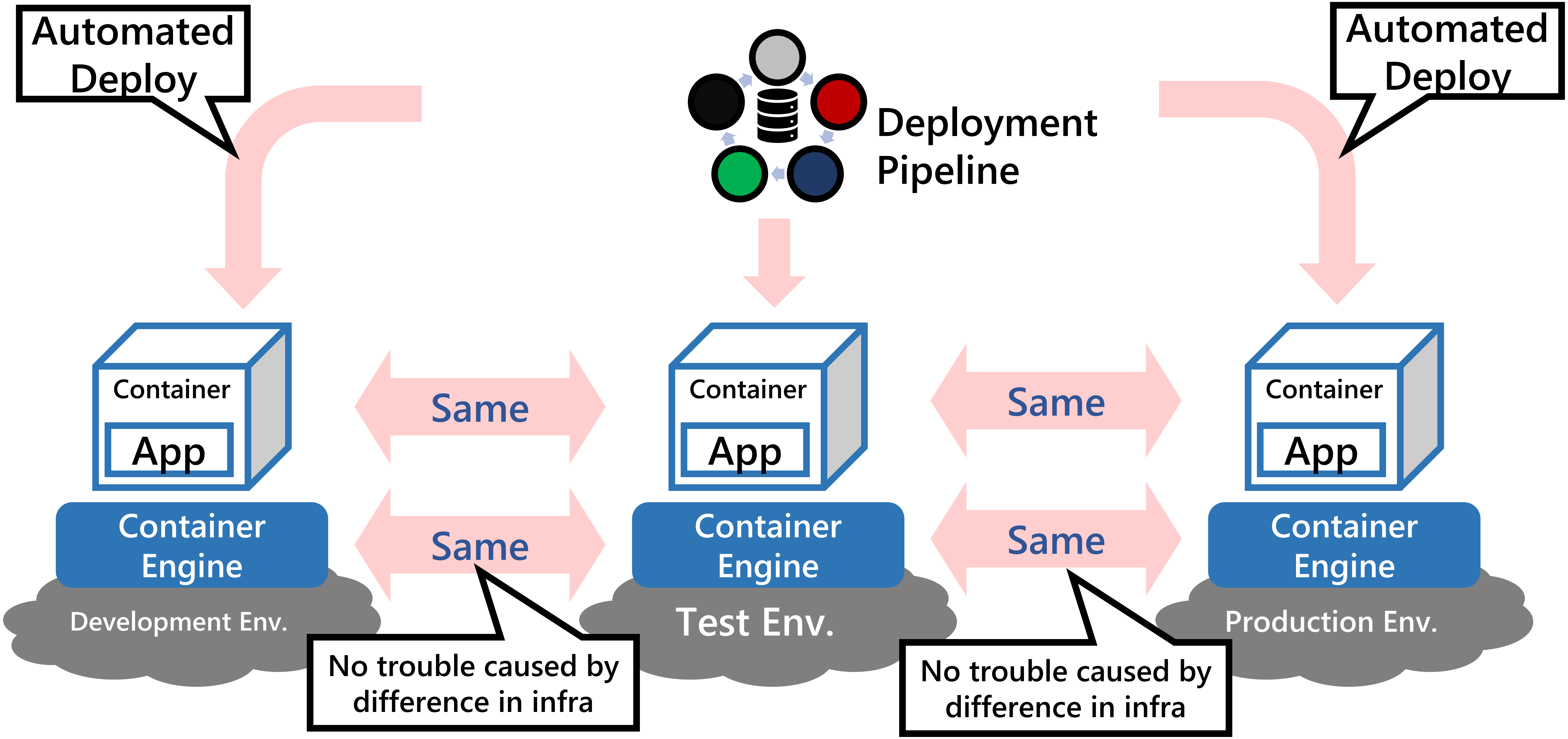
Container

- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

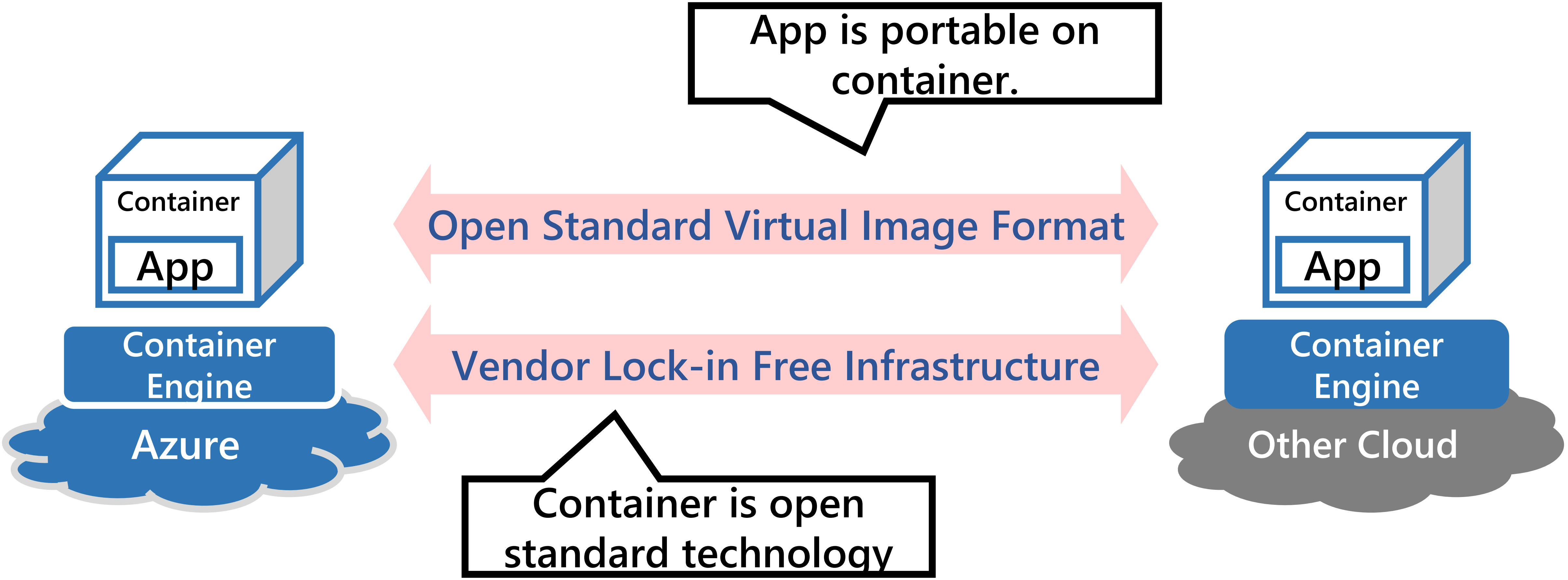
Orchestration

- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc.

Why Cloud Native #1: Speed & Quality

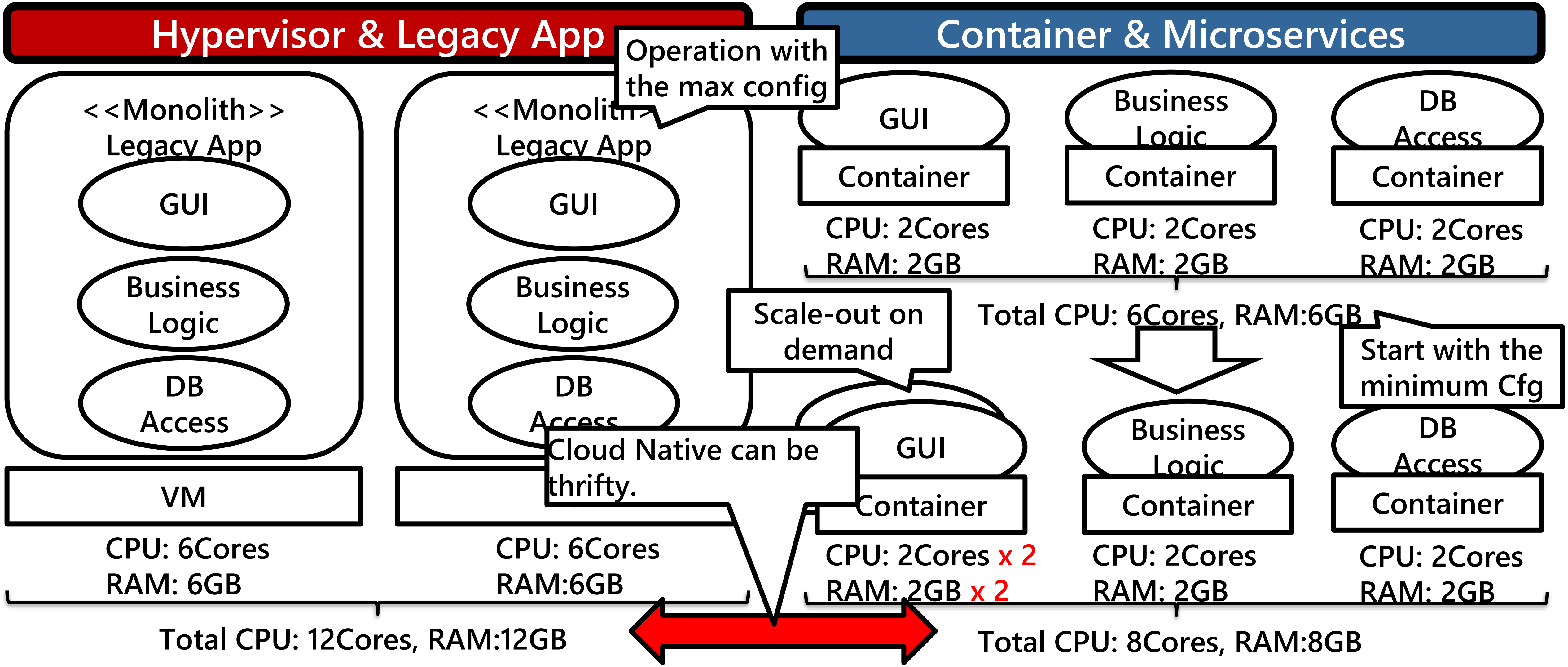


Why Cloud Native #2: Investment Protection


App is portable on container.
Container
App
Container Engine
Azure
Open Standard Virtual Image Format
Vendor Lock-in Free Infrastructure
Container
App
Container Engine
Other Cloud
Container is open standard technology

Why Cloud Native #3: Optimization of Utilization

Legacy (Monolith) needs the max config of dedicated system resources.
Microservices dynamically use system resources on demand.



Hypervisor & Legacy App

Container & Microservices

Operation with the max config

<<Monolith>>
Legacy App
GUI
Business Logic
DB Access
VM
CPU: 6Cores
RAM: 6GB

<<Monolith>>
Legacy App
GUI
Business Logic
DB Access
CPU: 6Cores
RAM:6GB

Total CPU: 12Cores, RAM:12GB

GUI
Container
CPU: 2Cores
RAM: 2GB

Business Logic
Container
CPU: 2Cores
RAM: 2GB

DB Access
Container
CPU: 2Cores
RAM: 2GB

Total CPU: 6Cores, RAM:6GB

Start with the minimum Cfg

Scale-out on demand

GUI
Container
CPU: 2Cores x 2
RAM: 2GB x 2

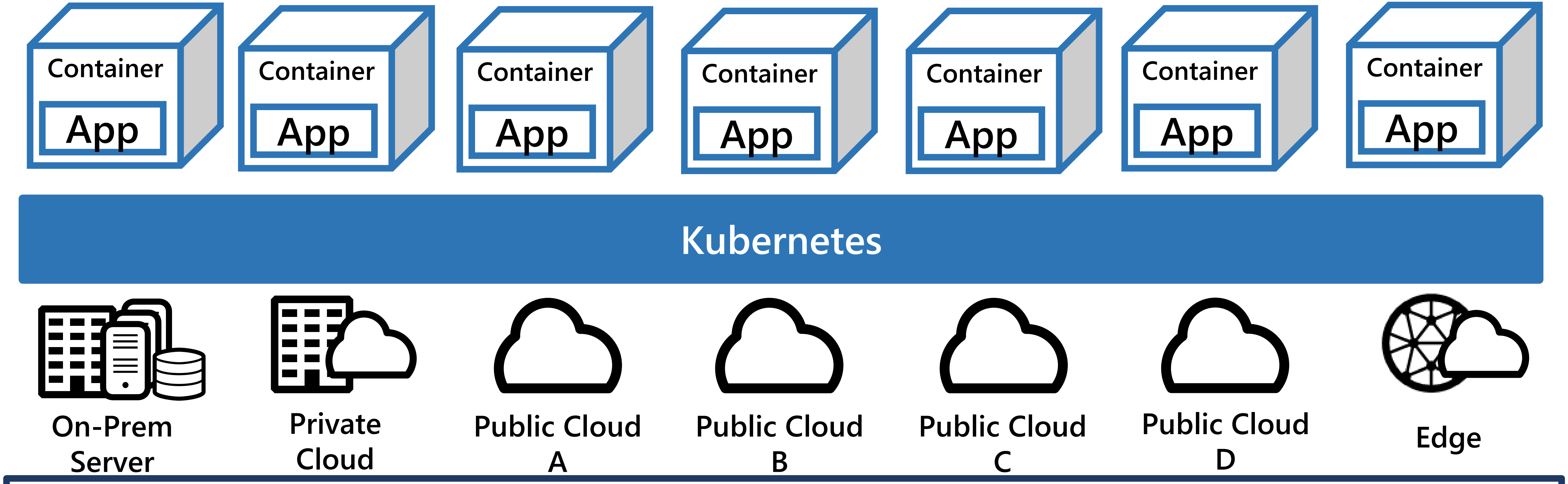
Business Logic
Container
CPU: 2Cores
RAM: 2GB

DB Access
Container
CPU: 2Cores
RAM: 2GB

Total CPU: 8Cores, RAM:8GB

Cloud Native can be thrifty.

Cloud Native: Next Generation of Cloud Platform



Container App | Container App | Container App | Container App | Container App | Container App | Container App

Kubernetes

On-Prem Server | Private Cloud | Public Cloud A | Public Cloud B | Public Cloud C | Public Cloud D | Edge

Merit of Container, Kubernetes, and Microservices

Speedy Infra Deployment

Speedy App Release

Quality of IT Systems

Scalability

Investment Protection

Microsoft

2.Microservices Concept

Core Technologies in Cloud Native Computing

Source : Charter in FAQ of CNCF in 2016

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

Container

- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

Orchestration

- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc.

Microservices

Microservices

- ✓ Modern app development and operation
  - ✓ Speedy
  - ✓ Flexible

Container

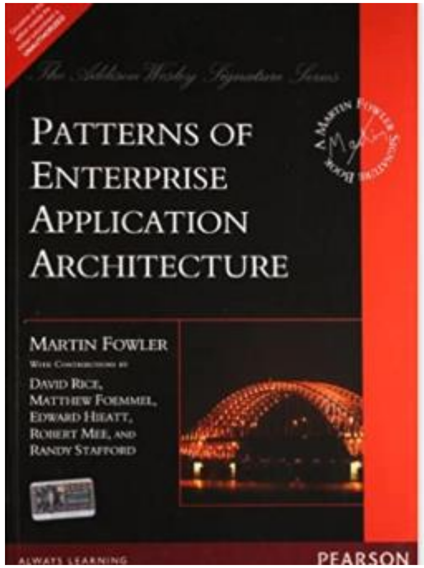
- ✓ Virtual computing environment
  - ✓ Speedy
  - ✓ Compact
  - ✓ Portable

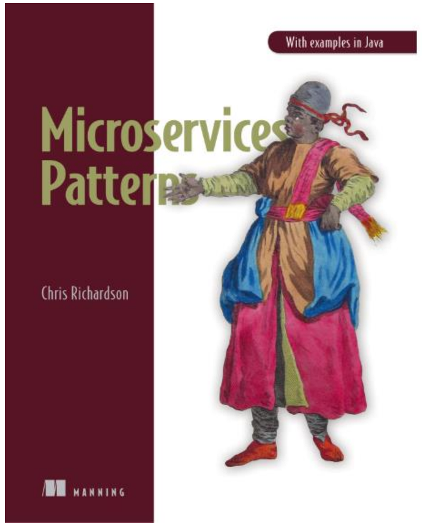
Orchestration

- ✓ Container cluster management
  - ✓ Routing
  - ✓ Load balancing
  - ✓ Scaling
  - ✓ High availability
  - ✓ etc.

Attention

- No standard definition about Microservices.
- This presentation follows thoughts of Martin Fowler and Chris Richardson.
  - Martin Fowler’s web site: https://martinfowler.com/articles/microservices.html
  - Chris Richardson’s web site: https://microservices.io/





Definition

- Microservices Guide (https://www.martinfowler.com/microservices/)
  - **In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.** These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.
- What are microservices? (https://microservices.io/)
  - **Microservices is an architecture style that structures an application as a collection of services that are**
    - Independent deployable
    - Loosely coupled
    - Organized around business capabilities
    - Owned by a small team
    - Highly maintainable and testable

Common Misconception

> - The size of a service is mostly unimportant.
> - One problem with the term microservices is that the first thing you hear is micro.
>
> Chris Richardson, Microservices Patterns, Manning, October 2018, ISBN 9781617294549

- Avoid argument about service granularity.
  - How on earth we can say this is small or large???
  - The size of a service should not always be small.
  - There could be large services.

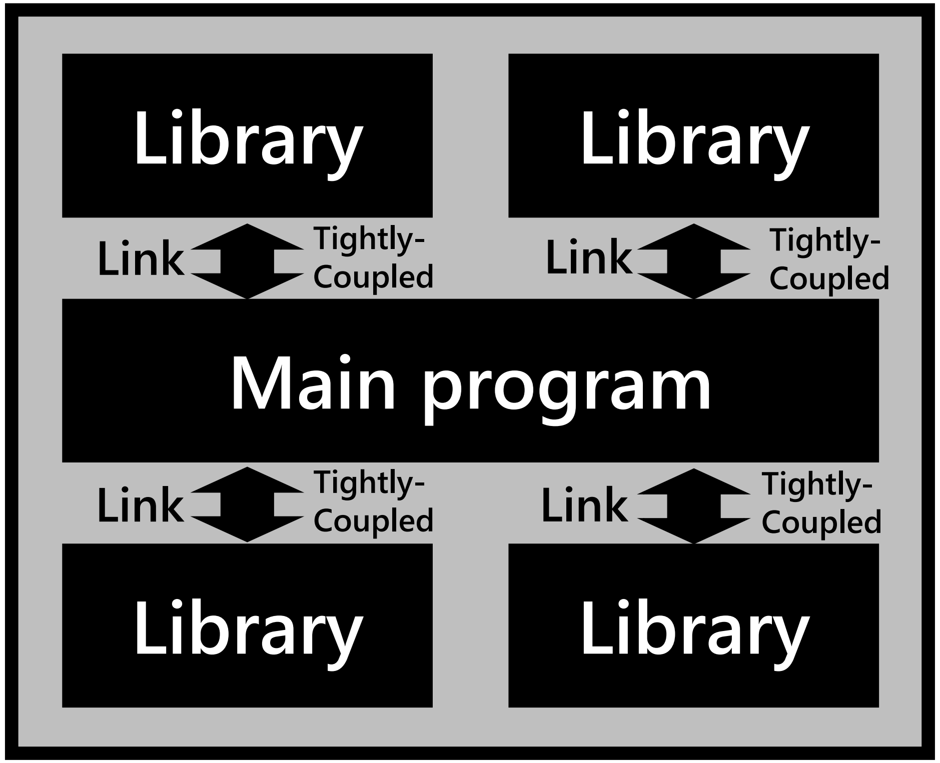
Concept

Architecture style for modern application development and operation

- Componentization
  - Ease of maintenance and update by each application component.
- Foundation for cloud native application
  - Microservices are frequently adopted in the cloud native application development.

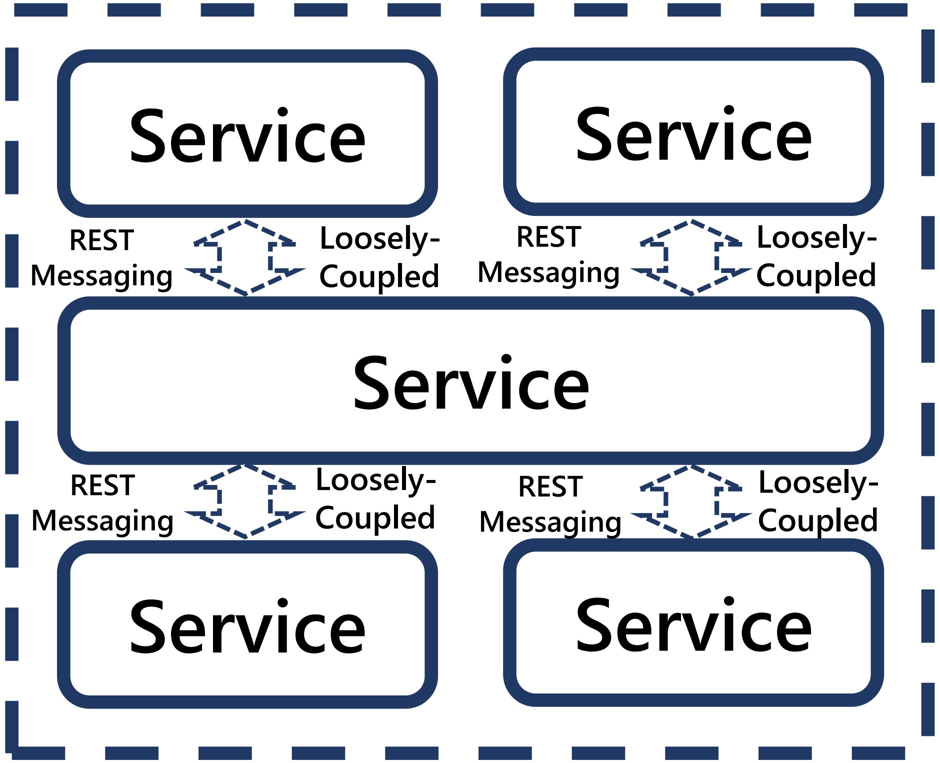
Concept

Monolith: Library-oriented



- ✓ Each app component tightly coupled.
- ✓ Difficult to update each app component,

Microservices: Service-oriented



- ✓ Each app component loosely coupled.
- ✓ Easy to update each app component.

Merits

- Fine-grained
  - App release
  - App updates and maintenance
  - Scaling
- Technology diversity
- Less impacts by failures

Reference: https://www.martinfowler.com/articles/microservice-trade-offs.html

Drawbacks

- Distributed application, then …
  - Network latency in communication among services
  - Difficulty and restriction in data consistency and synchronization
  - Operational complexity
- Learning cost for service modeling

Reference: https://www.martinfowler.com/articles/microservice-trade-offs.html

Terminology

- Domain
  - An independent business area
- Bounded context
  - A part in a domain
  - A target of systematization with IT
- Microservices
  - Architecture style that structures an application as a collection of services
    - Methods and apparatus of app design, development, and operation to accelerate application development and update.
- Microservice Architecture
  - Software structure based on a collection of services
- Service
  - A software component
    - Developed and deployed respectively and independently
    - Run on an independent app runtime
  - A basic unit of Microservice architecture

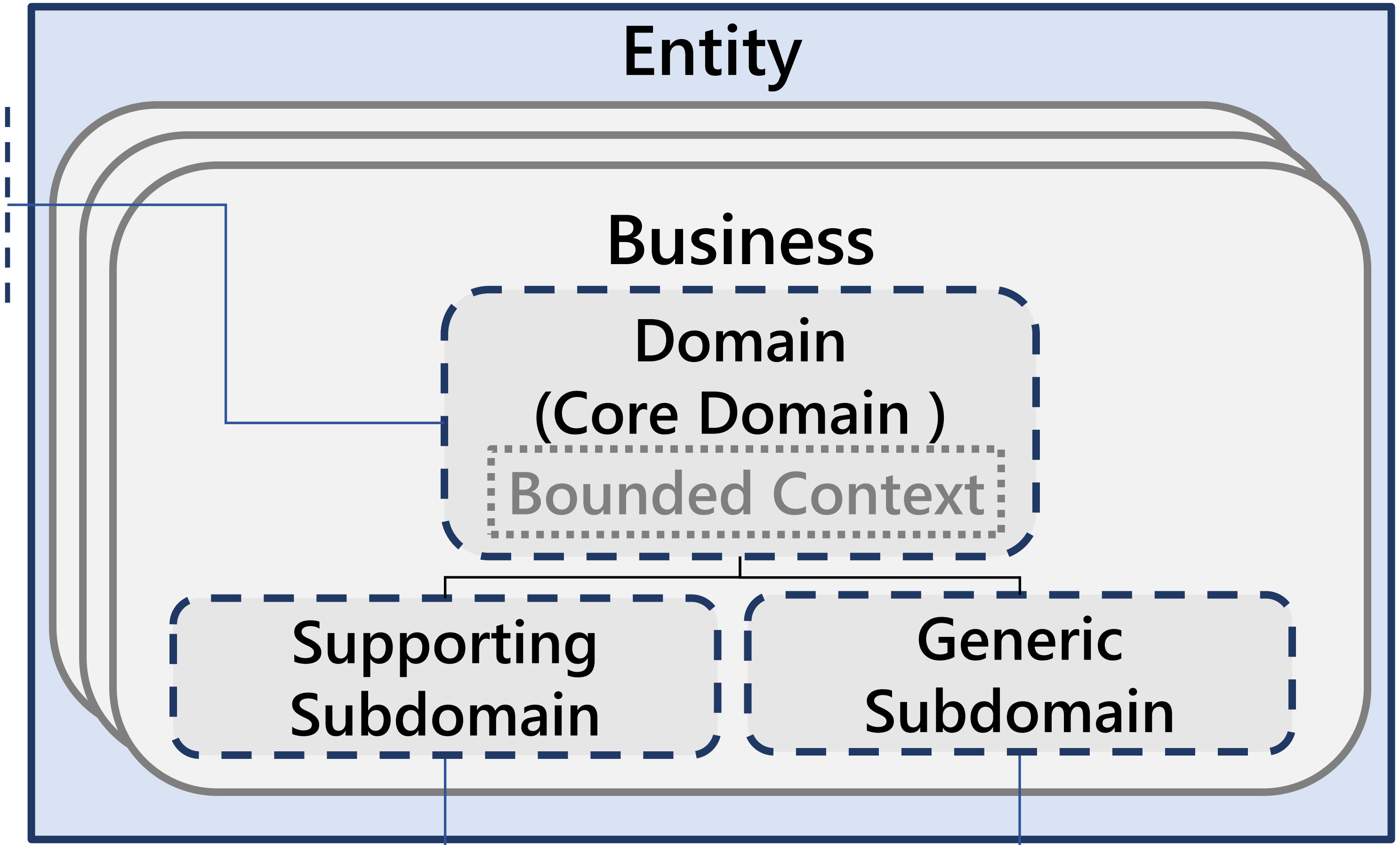
Terminology: Domain

- Domain
  - An independent business area
- Bounded context
  - A part in a domain
  - A target of systematization with IT
- Microservices
  - Architecture style that structures an application as a collection of services
    - Methods and apparatus of app design, development, and operation to accelerate application development and update.
- Microservice Architecture
  - Software structure based on a collection of services
- Service
  - A software component
    - Developed and deployed respectively and independently
    - Run on an independent app runtime
  - A basic unit of Microservice architecture

Terminology: Domain

Domain:

- Main business
- Target of analysis



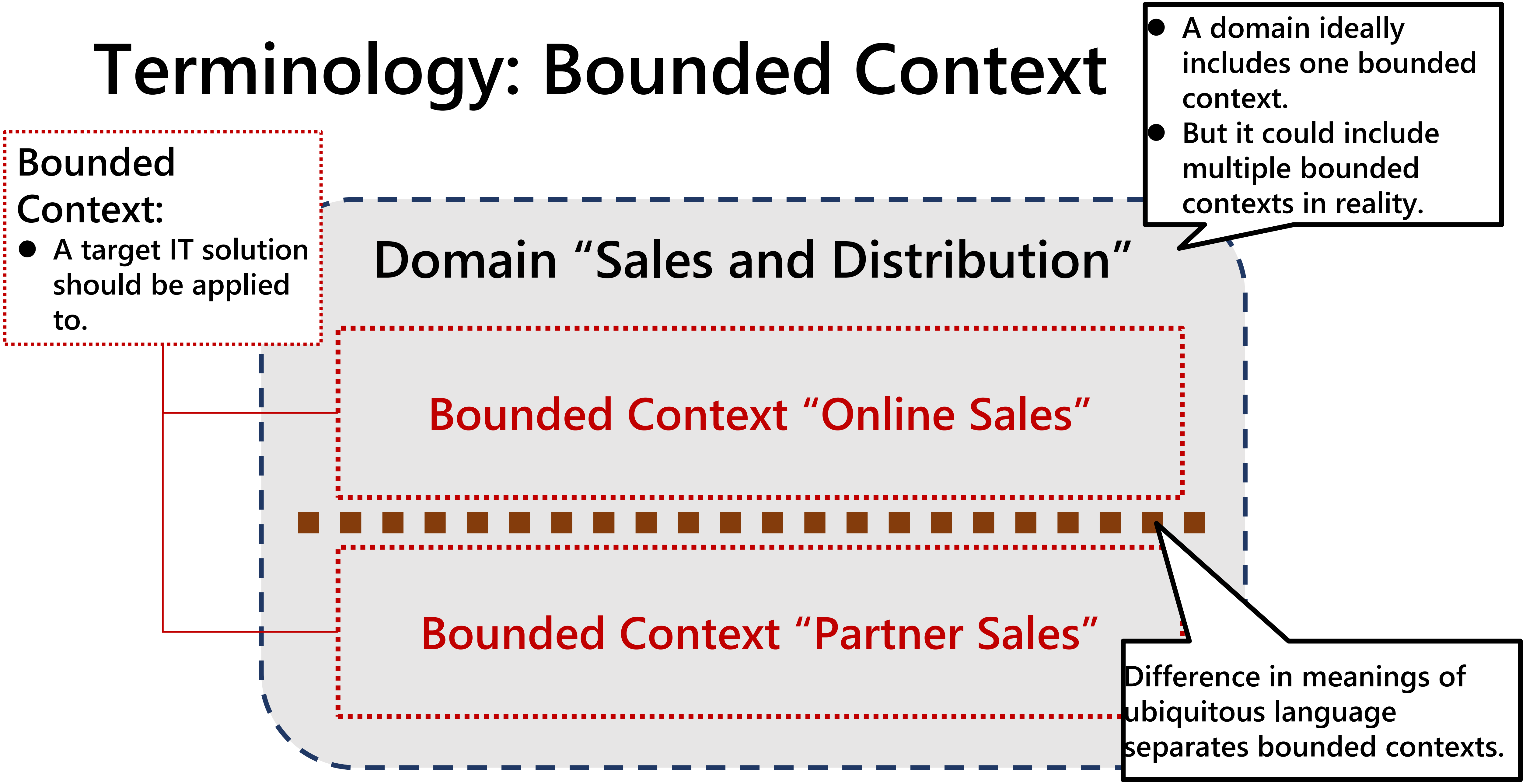
Subdomain:

- Business function used by Core domain
- Supporting Subdomain:
  - Adjunct to main business
- Generic subdomain:
  - Functions used in general like authentication

Terminology: Bounded Context

- Domain
  - An independent business area
- Bounded context
  - A part in a domain
  - A target of systematization with IT
- Microservices
  - Architecture style that structures an application as a collection of services
    - Methods and apparatus of app design, development, and operation to accelerate application development and update.
- Microservice Architecture
  - Software structure based on a collection of services
- Service
  - A software component
    - Developed and deployed respectively and independently
    - Run on an independent app runtime
  - A basic unit of Microservice architecture

Terminology: Bounded Context



Bounded Context:

- A target IT solution should be applied to.

Domain “Sales and Distribution”

Bounded Context “Online Sales”

Bounded Context “Partner Sales”

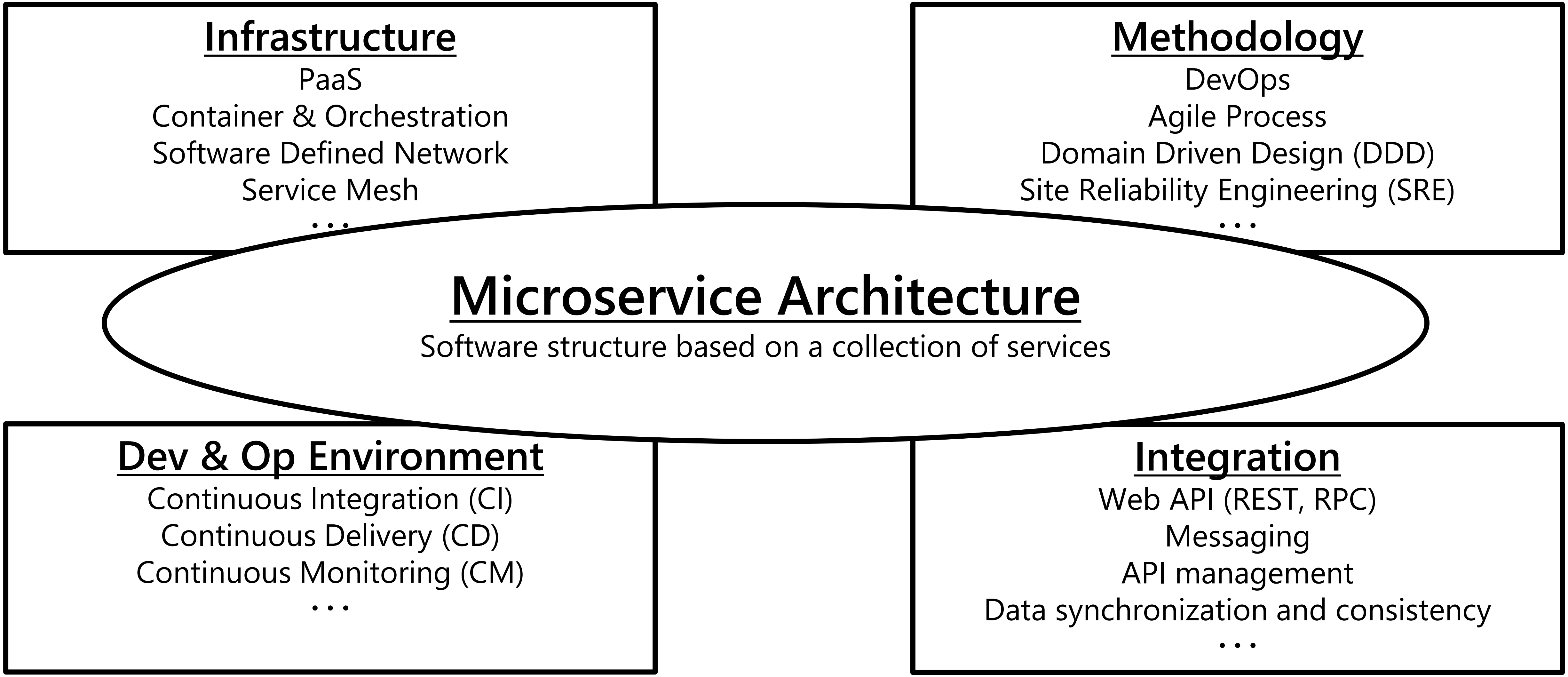
- A domain ideally includes one bounded context.
- But it could include multiple bounded contexts in reality.

Difference in meanings of ubiquitous language separates bounded contexts.

Terminology: Microservices

- Domain
  - An independent business area
- Bounded context
  - A part in a domain
  - A target of systematization with IT
- Microservices
  - Architecture style that structures an application as a collection of services
    - Methods and apparatus of app design, development, and operation to accelerate application development and update.
- Microservice Architecture
  - Software structure based on a collection of services
- Service
  - A software component
    - Developed and deployed respectively and independently
    - Run on an independent app runtime
  - A basic unit of Microservice architecture

Terminology: Microservices as “Architecture Style”



Infrastructure

PaaS
Container & Orchestration
Software Defined Network
Service Mesh
…

Methodology

DevOps
Agile Process
Domain Driven Design (DDD)
Site Reliability Engineering (SRE)
…

Microservice Architecture

Software structure based on a collection of services

Dev & Op Environment

Continuous Integration (CI)
Continuous Delivery (CD)
Continuous Monitoring (CM)
…

Integration

Web API (REST, RPC)
Messaging
API management
Data synchronization and consistency
…

Terminology: Microservice Architecture

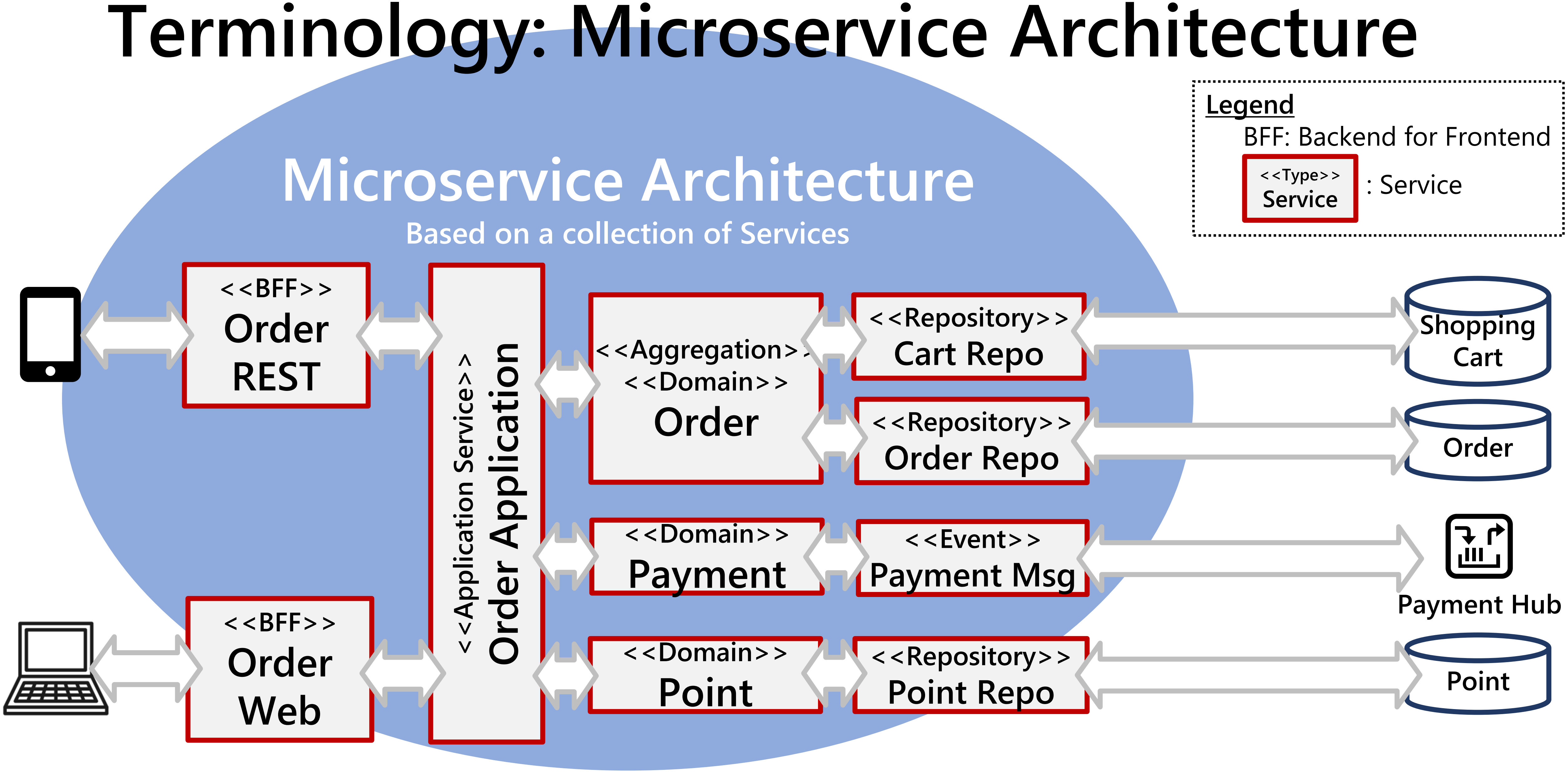
- Domain
  - An independent business area
- Bounded context
  - A part in a domain
  - A target of systematization with IT
- Microservices
  - Architecture style that structures an application as a collection of services
    - Methods and apparatus of app design, development, and operation to accelerate application development and update.
- Microservice Architecture
  - Software structure based on a collection of services
- Service
  - A software component
    - Developed and deployed respectively and independently
    - Run on an independent app runtime
  - A basic unit of Microservice architecture

Terminology: Microservice Architecture

Legend

BFF: Backend for Frontend

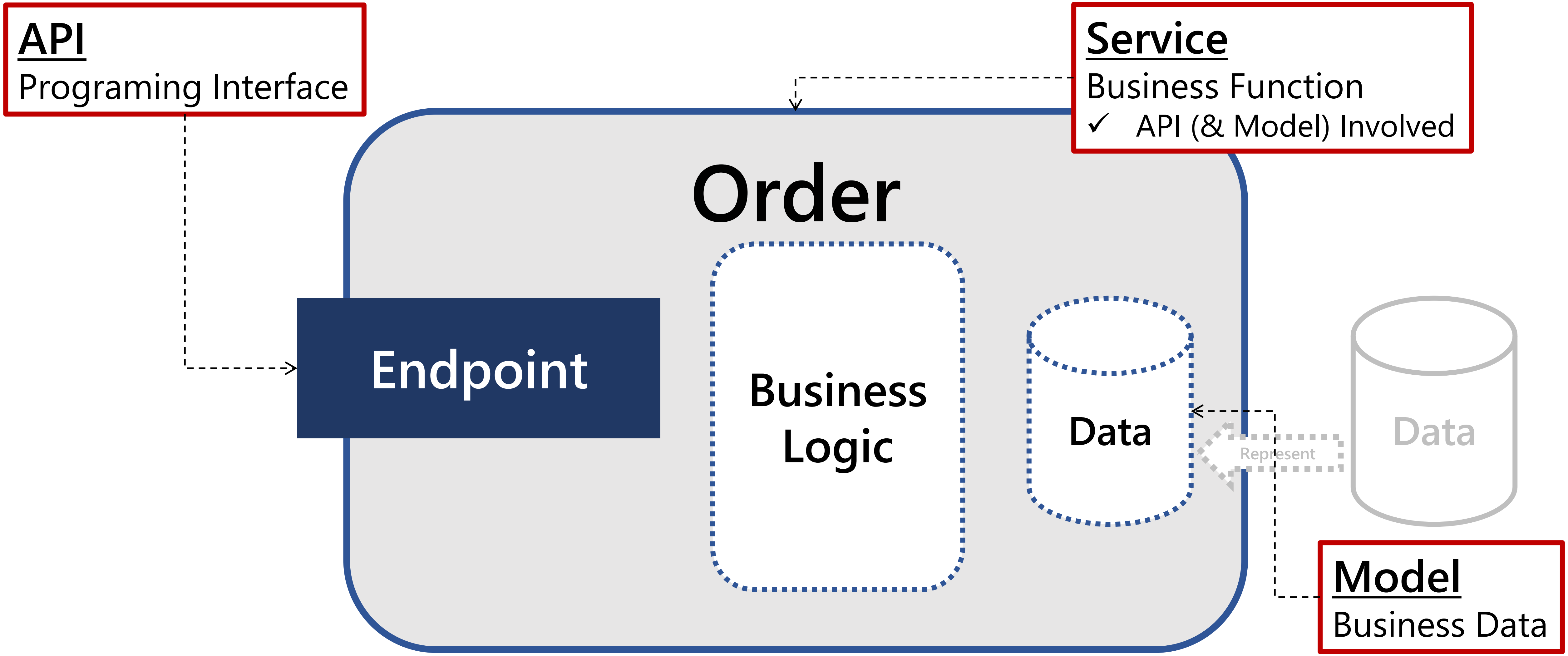
<<Type>> Service : Service


Microservice Architecture
Based on a collection of Services
<<BFF>> Order REST
<<BFF>> Order Web
<<Application Service>> Order Application
<<Aggregation>> <<Domain>> Order
<<Domain>> Payment
<<Domain>> Point
<<Repository>> Cart Repo
<<Repository>> Order Repo
<<Event>> Payment Msg
<<Repository>> Point Repo
Shopping Cart
Order
Payment Hub
Point

Terminology: Service

- Domain
  - An independent business area
- Bounded context
  - A part in a domain
  - A target of systematization with IT
- Microservices
  - Architecture style that structures an application as a collection of services
    - Methods and apparatus of app design, development, and operation to accelerate application development and update.
- Microservice Architecture
  - Software structure based on a collection of services
- Service
  - A software component
    - Developed and deployed respectively and independently
    - Run on an independent app runtime
  - A basic unit of Microservice architecture

Terminology: Service



API
Programing Interface

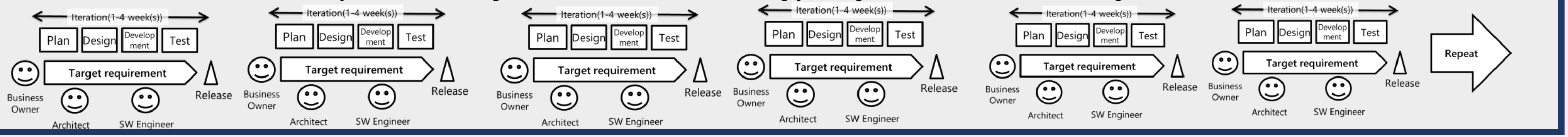
Service
Business Function
✓ API (& Model) Involved

Model
Business Data

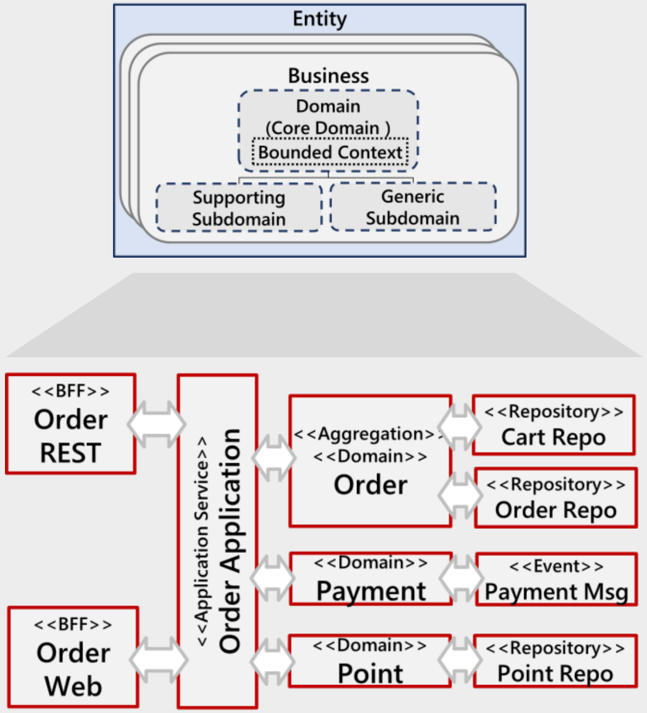
Microservices: Holistic View

Microservices

Project Management Methodology: Agile Process Management

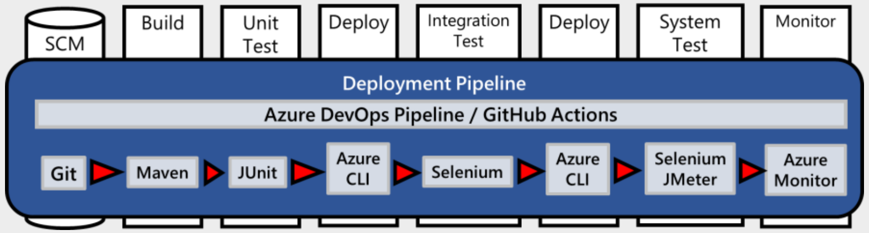


Design: Domain Drive Design



Development and Operation

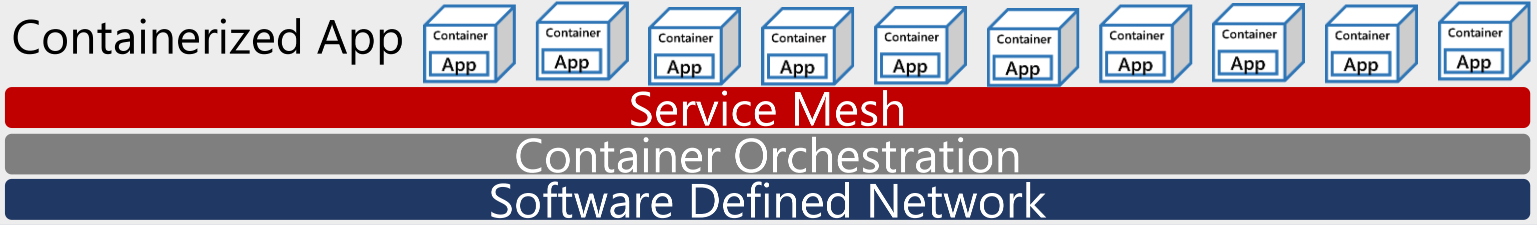
Continuous Delivery (Deployment Pipeline)



Site Reliability Engineering (SRE)

App Runtime

Containerized App

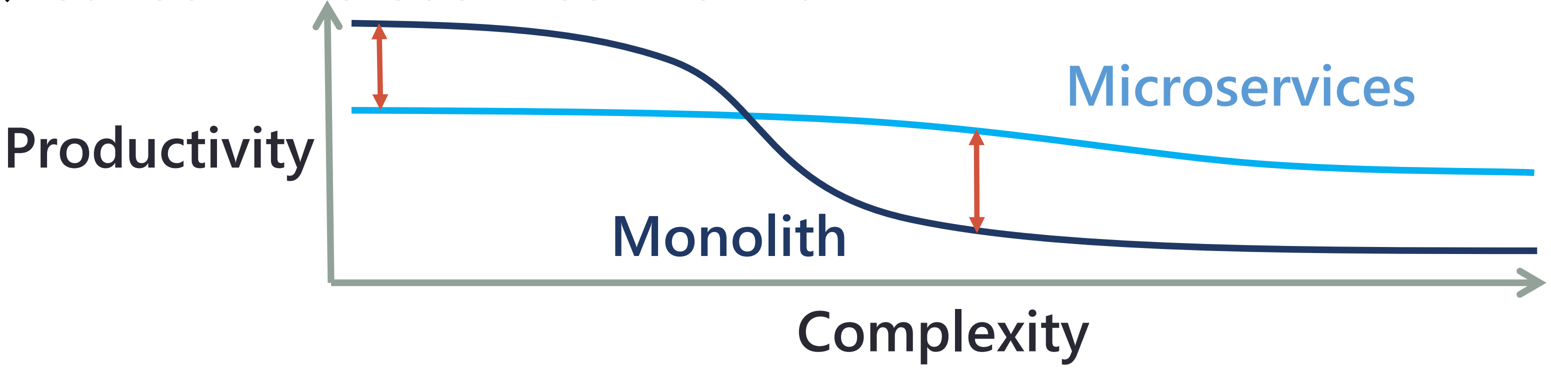


When to Use Microservices

Avoid Microservice Premium

Microservices for simple system results in high cost

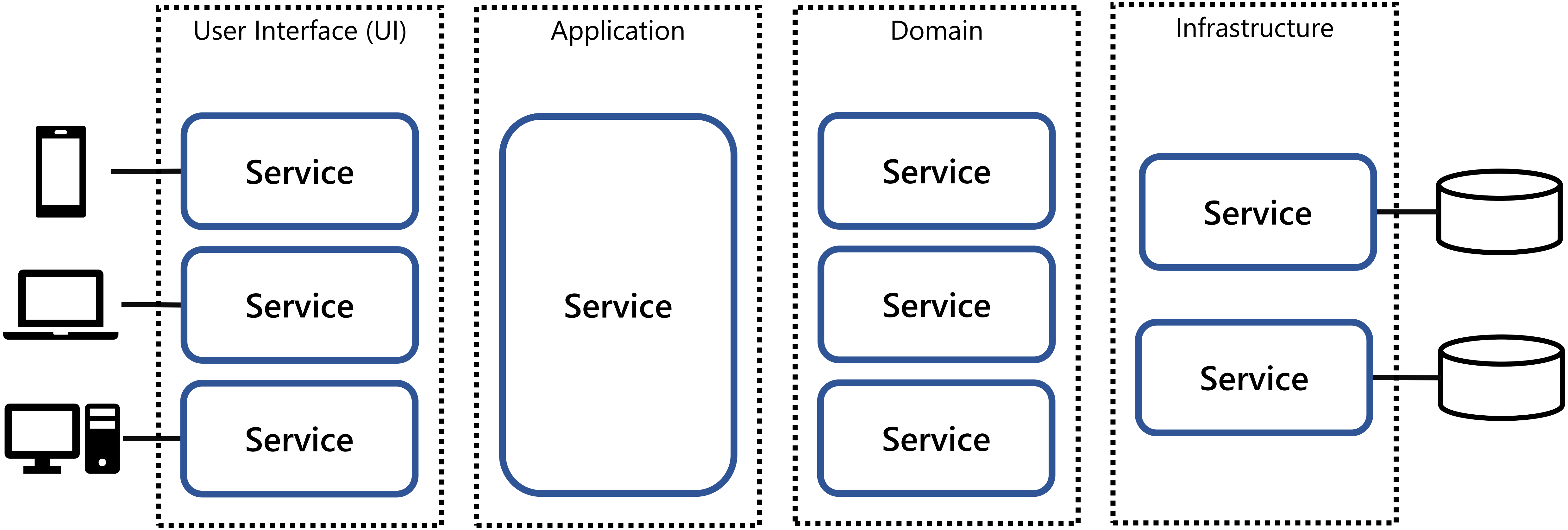
- Principle
  - Target is “COMPLEX” system: Consider Microservices
  - Target is “SIMPLE” system: Microservices may not fit
    - Called “Microservice Premium”



Reference: https://www.martinfowler.com/bliki/MicroservicePremium.html

3. Microservice Architecture Overview

Layered Architecture



- ✓Receive requests
- ✓Send responses

- ✓Transaction management
- ✓Application coordination
- ✓Access to domain services

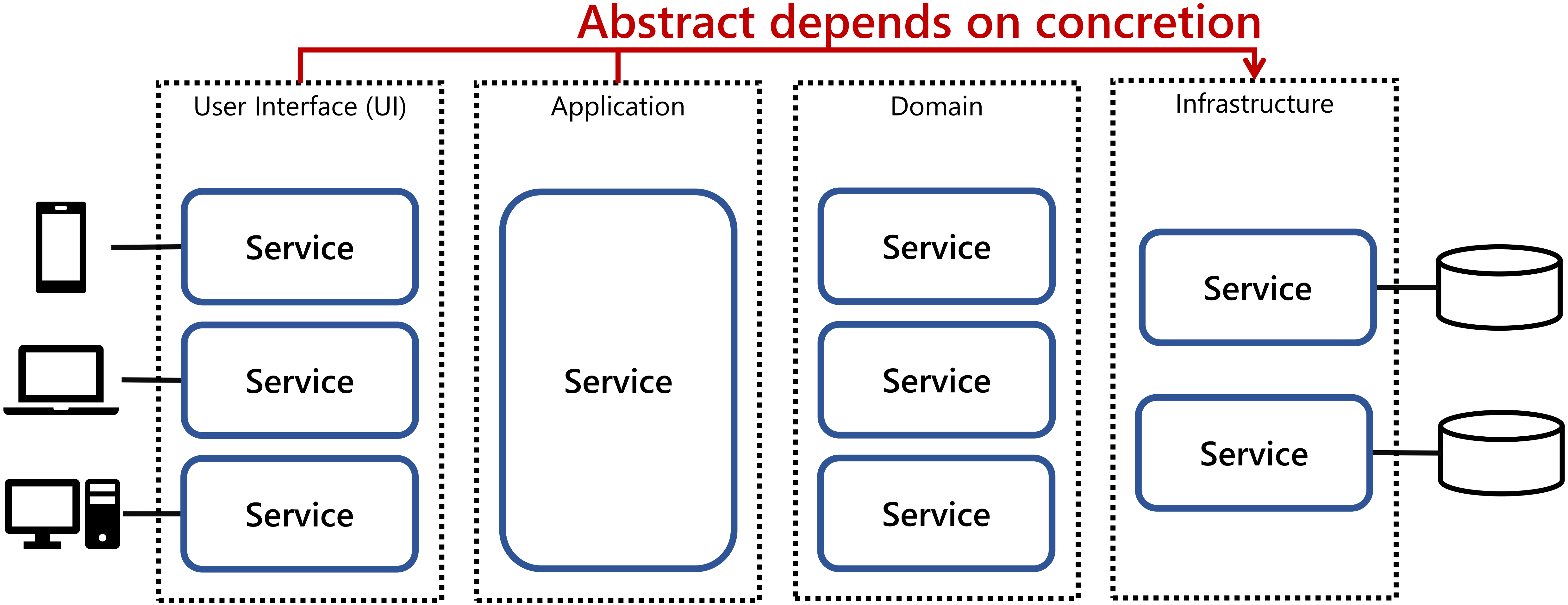
- ✓Business implementation
- ✓Status and behavior of business domain

- ✓External service support (ex. Access to data and messaging)

According to Domain Driven Design

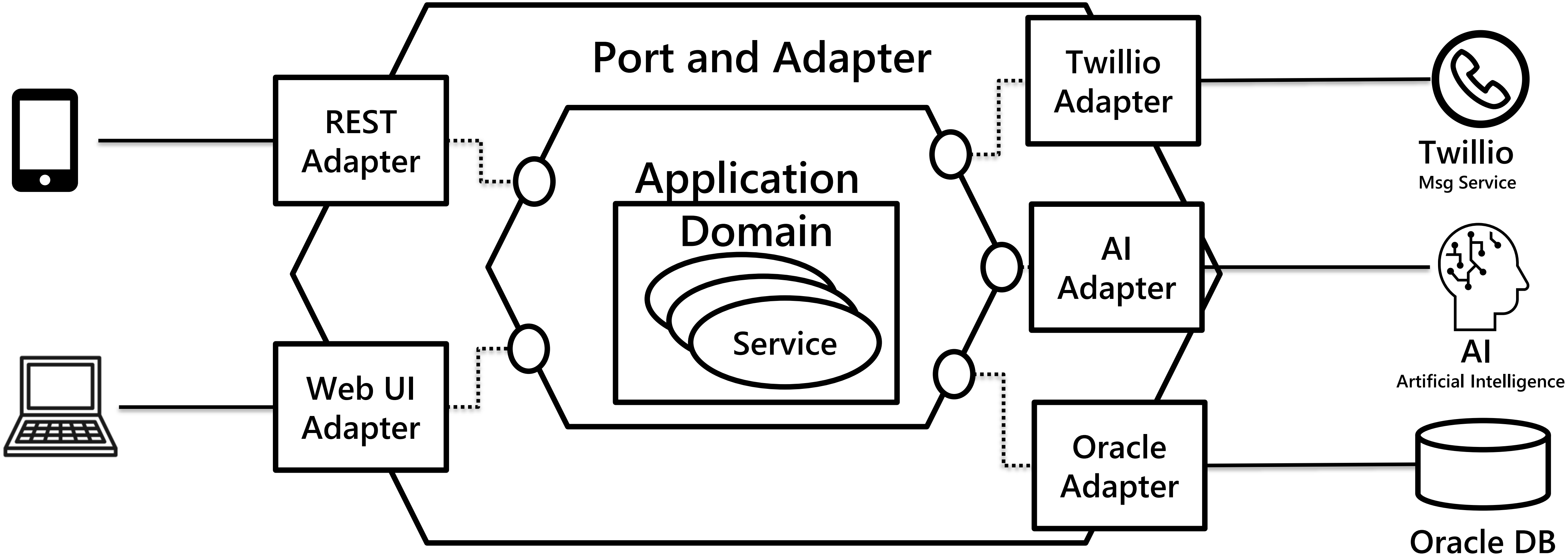
Layered Architecture: Pros & Cons

- ◆ Simple and easy to understand
- ◆ Difficult to extend: Abstraction (App) depends on concretion (Infra)



Hexagonal Architecture

- ◆ App Domain Oriented, I/O through Port & Adapter
- ◆ Ease of extension
  - ➢ Abstraction (App) separated from concretion (Infra)



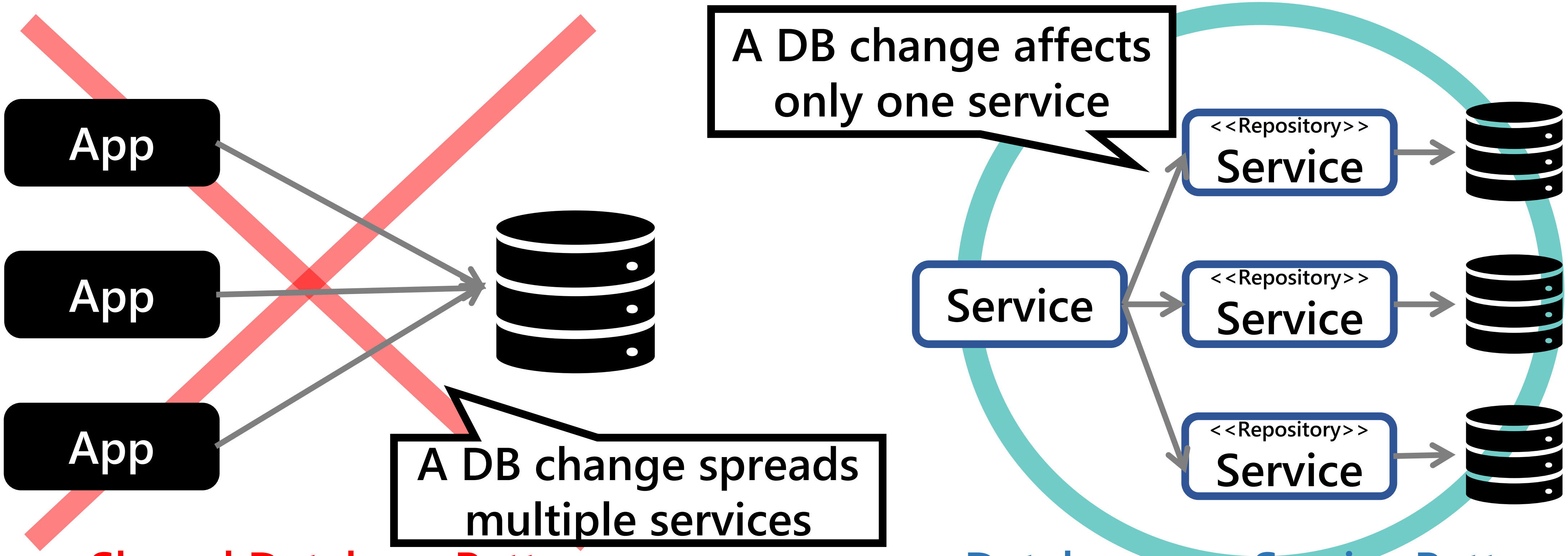
Data Access: Design Policy

Access Data through Service

◆ Minimize effect of change in DB design and implementation

DB Access in a Monolith fashion

DB Access in a Microservices fashion



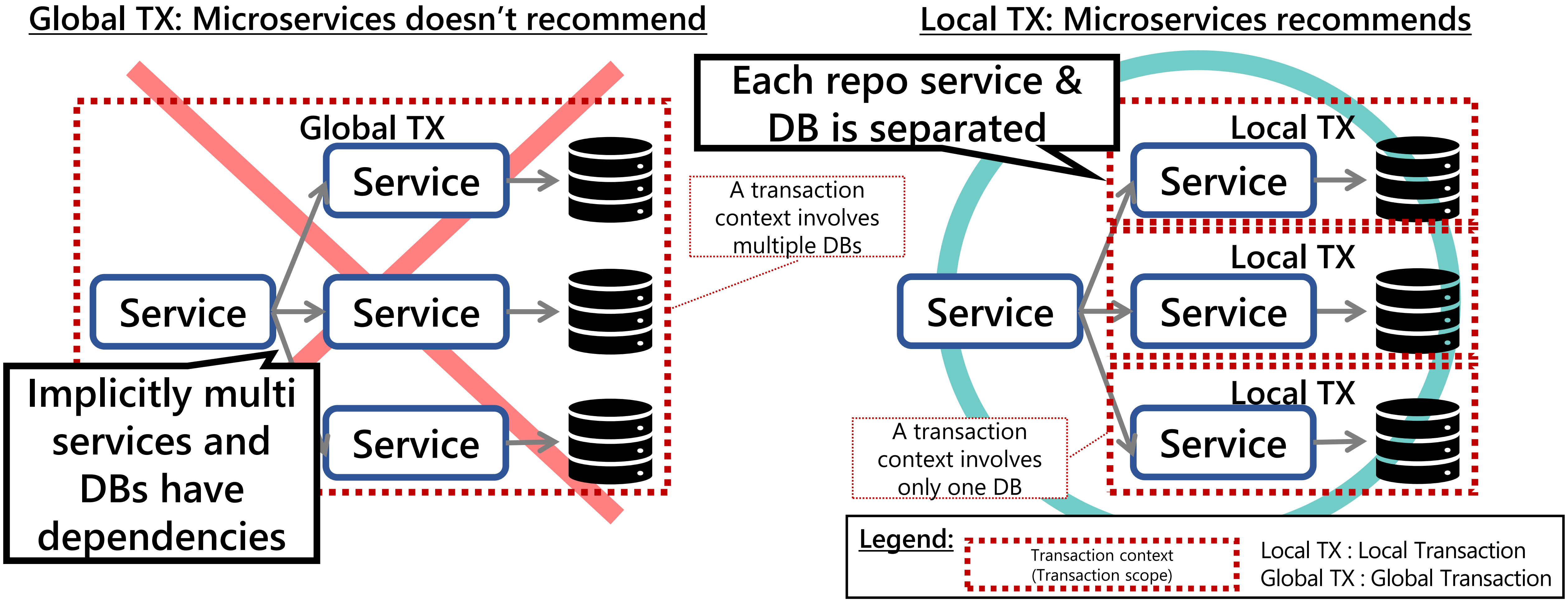
Shared Database Pattern

Database per Service Pattern

Transaction Management: Design Policy

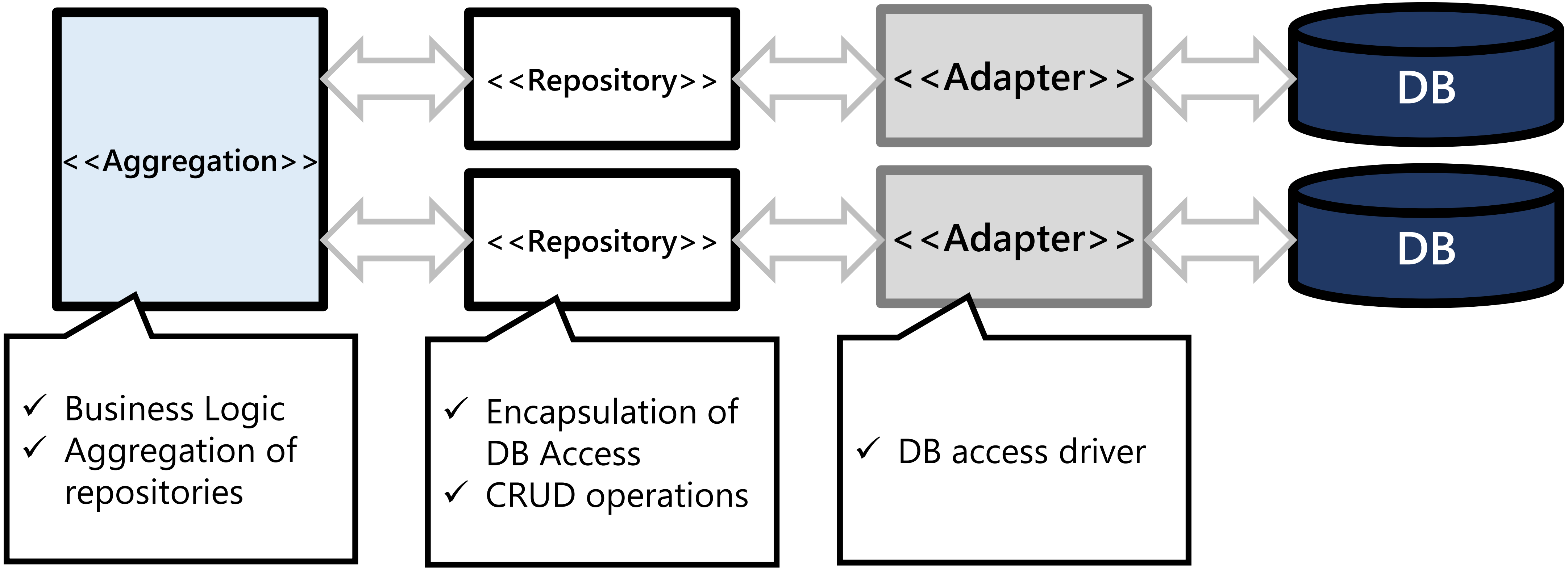
Local Transaction is recommended

◆ Keep simple structure to maximize values of loosely-coupled design



Global TX: Microservices doesn't recommend
Global TX
Service
Service
Service
Service
Implicitly multi services and DBs have dependencies
A transaction context involves multiple DBs
Local TX: Microservices recommends
Each repo service & DB is separated
Local TX
Service
Local TX
Service
Local TX
Service
Service
A transaction context involves only one DB
Legend:
Transaction context (Transaction scope)
Local TX : Local Transaction
Global TX : Global Transaction

Data Access: Basic Pattern



<<Aggregation>>

<<Repository>>

<<Repository>>

<<Adapter>>

<<Adapter>>

DB

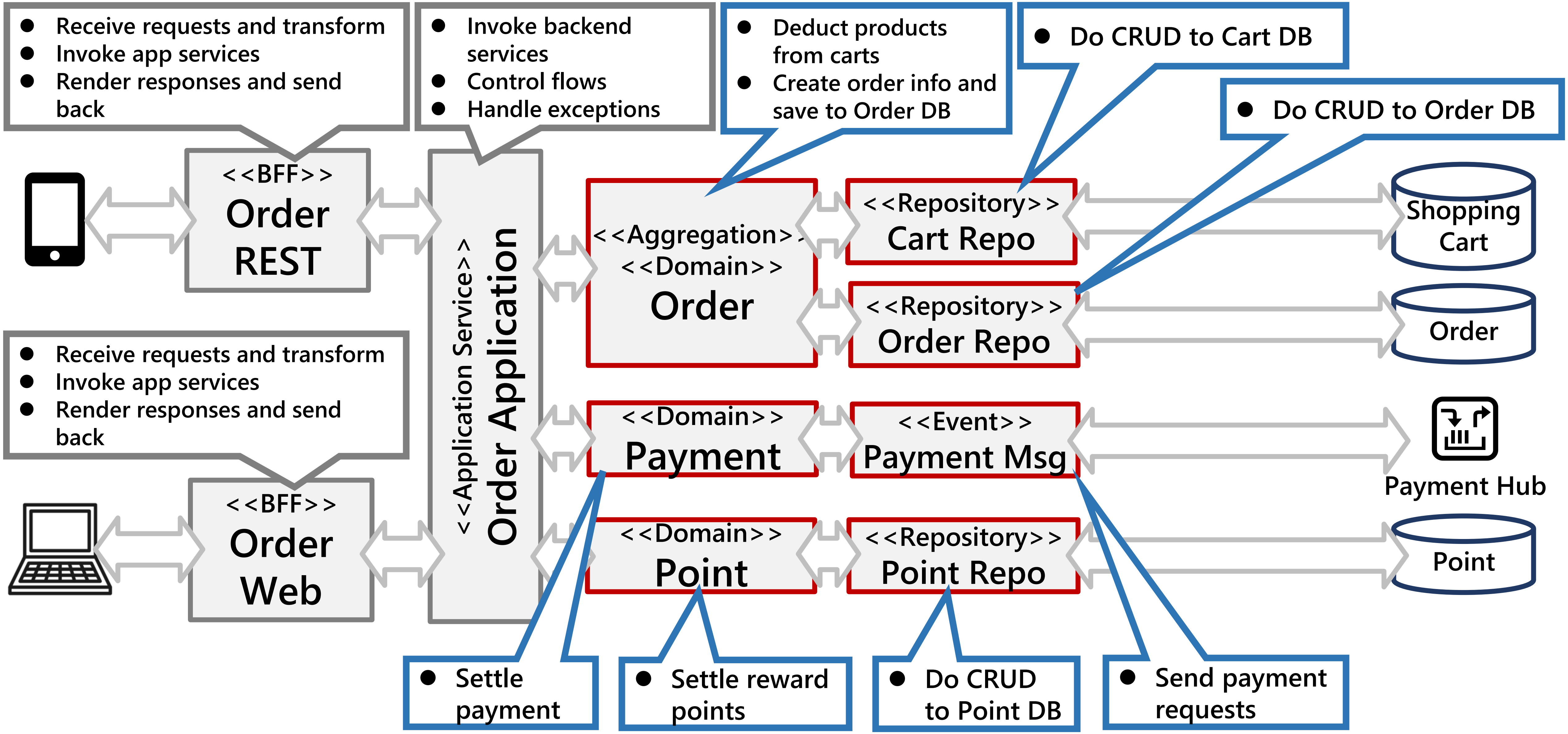
DB

- ✓ Business Logic
- ✓ Aggregation of repositories

- ✓ Encapsulation of DB Access
- ✓ CRUD operations

- ✓ DB access driver

Data Access: Design Example



Microsoft

4.Points for Success

How we can accelerate app modernization with Microservices

- This chapter discusses points for success from four aspects:
  - **Domain (Bounded Context)**
    - How to find targets for microservices?
  - **Formation**
    - How to organize a team?
  - **Methodology**
    - How to proceed?
  - **Design**
    - How to shape it?

Domain (Bounded Context)

Find targets from below points of view

- Business Priorities
  - Invest in "high priority" business
- Frequency of change and update
  - Microservices enables frequent change and update of application in a fine-grained manner
- Edge
  - User interface (API) layer, authn and authz, etc.
  - Easy to pick up targets for innovation

Reference:
https://learn.microsoft.com/azure/architecture/guide/technology-choices/microservices-assessment#understand-business-priorities
https://learn.microsoft.com/azure/architecture/guide/technology-choices/microservices-assessment#identify-business-areas-that-change-frequently
https://martinfowler.com/articles/break-monolith-into-microservices.html#DecoupleWhatIsImportantToTheBusinessAndChangesFrequently
https://martinfowler.com/articles/break-monolith-into-microservices.html#WarmUpWithASimpleAndFairlyDecoupledCapability

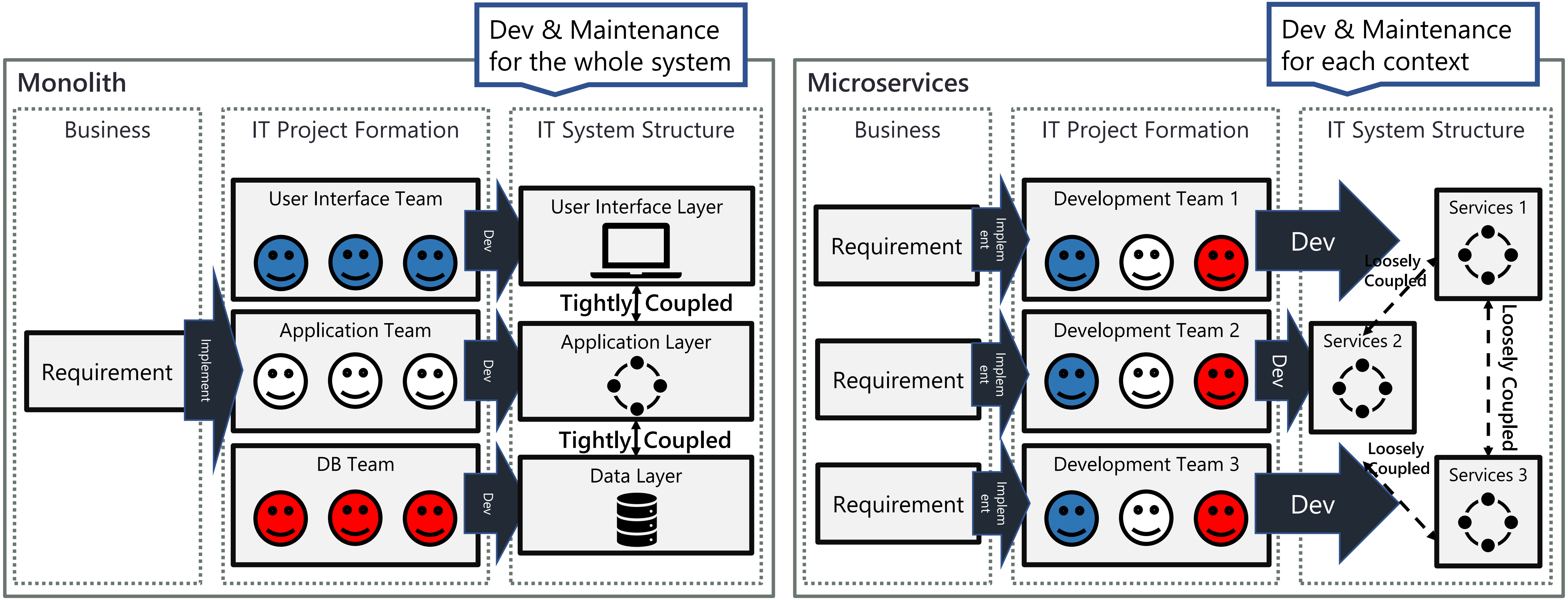
Formation

- Focused and cross-functional team
- Each team formed by a bounded context
- 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 Conway*

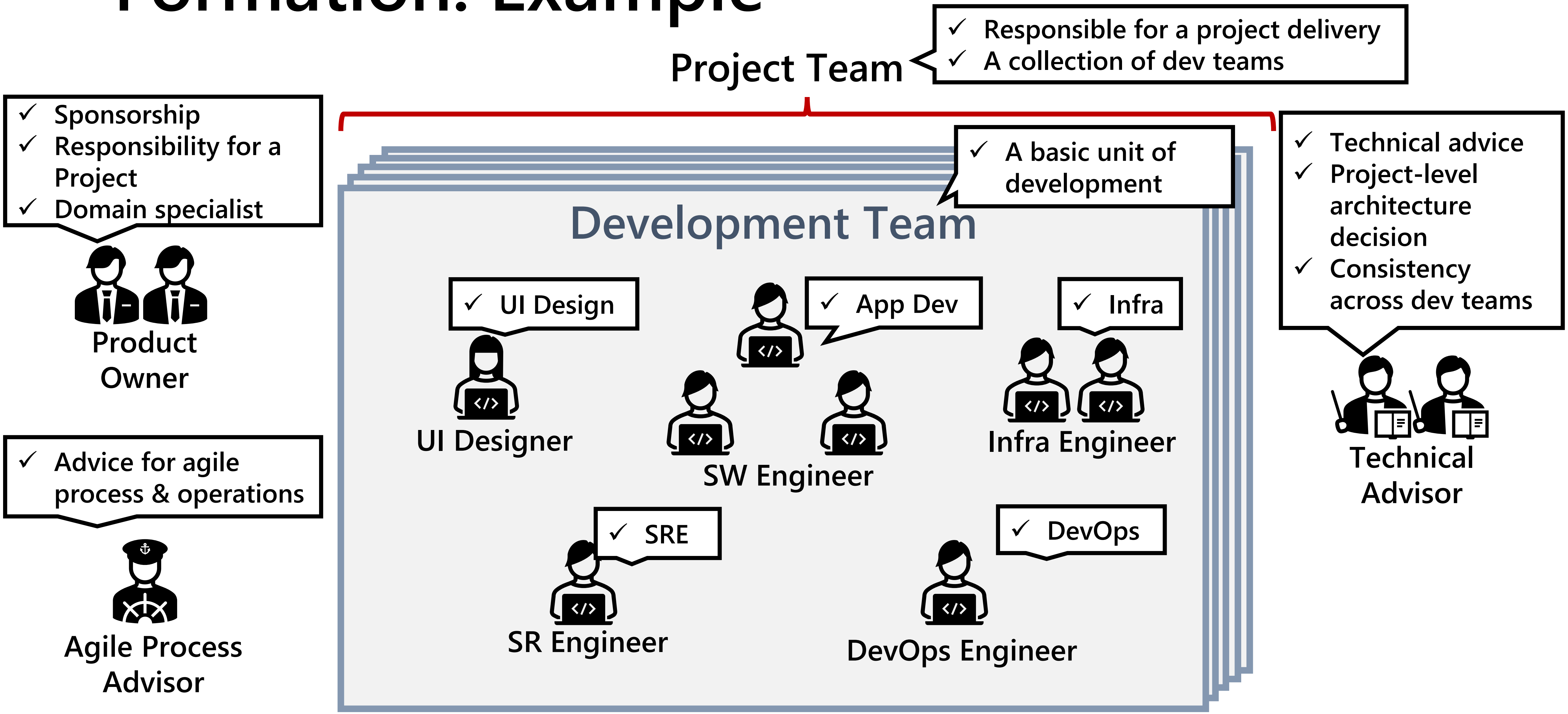
Reference:
https://learn.microsoft.com/en-US/azure/architecture/guide/technology-choices/microservices-assessment#assess-team-composition
https://martinfowler.com/bliki/ConwaysLaw.html

Formation

- Focused and cross-functional team
- Each team formed by a bounded context



Formation: Example



Project Team

- ✓ Responsible for a project delivery
- ✓ A collection of dev teams

Product Owner

- ✓ Sponsorship
- ✓ Responsibility for a Project
- ✓ Domain specialist

Agile Process Advisor

- ✓ Advice for agile process & operations

Development Team

- ✓ A basic unit of development

UI Designer

- ✓ UI Design

SW Engineer

- ✓ App Dev

Infra Engineer

- ✓ Infra

SR Engineer

- ✓ SRE

DevOps Engineer

- ✓ DevOps

Technical Advisor

- ✓ Technical advice
- ✓ Project-level architecture decision
- ✓ Consistency across dev teams

Methodology: The Twelve-Factor App

Guidelines for building maintainable and scalable applications

- ●Main Features
  - ■Application and Environment Consistency
    - ◆Keep one codebase in the revision control
    - ◆Keep development, staging, and production as similar as possible
  - ■Application Portability
    - ◆Declare dependencies outside of application implementation
    - ◆Store config in the environment
    - ◆Bind backend services and network interface with config
  - ■Scalability
    - ◆Scale out via the process model horizontally

Reference:
https://12factor.net/
https://learn.microsoft.com/en-US/azure/architecture/guide/technology-choices/microservices-assessment#use-the-twelve-factor-methodology
https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition#the-twelve-factor-application

Methodology: The Twelve-Factor App

I	Codebase	One codebase tracked in revision control and deployed to many environments.
II	Dependencies	Explicitly declare dependencies of software. Isolate dependency declaration from application implementation.
III	Config	Store config in the environment.
IV	Backing services	Treat backing services as attached resources.
V	Build, release, run	Strictly separate build and run stages.
VI	Process	Execute the app as one or more stateless processes.
VII	Port binding	Export services via port binding.
VIII	Concurrency	Scale out via the process model.
IX	Disposability	Maximize robustness with fast startup and graceful shutdown.
X	Dev/prod parity	Keep development, staging, and production as similar as possible.
XI	Logs	Treat logs as event streams.
XII	Admin processes	Run admin/management tasks as one-off processes.

Reference:
https://12factor.net/

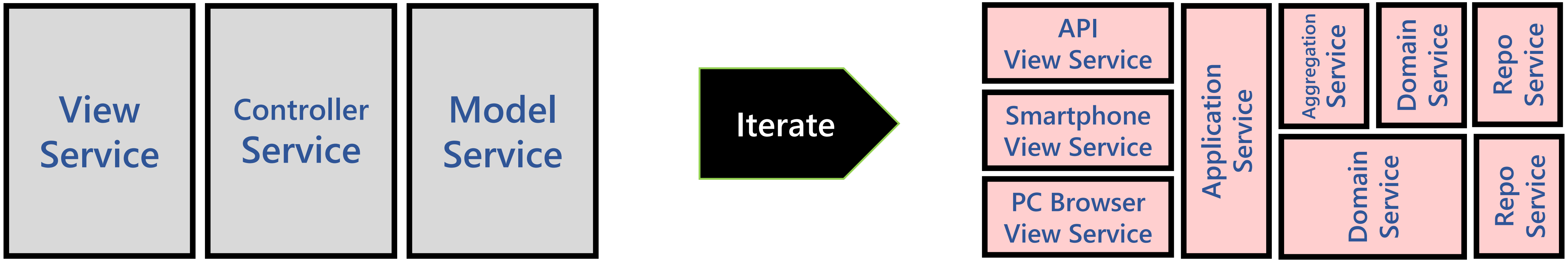
Methodology: Go Macro First, then Micro

Make services fine-grained on demand

- One of methods to proceed toward microservice architecture
- Direction
  - First development phase: Coarse-grained service
  - Iteration phase: Fine-grained service if required
- Motivation
  - No clue to find appropriate granularity for services
  - Feedback from business front-line

Implement macro services, first

Breakdown into micro, “if required”



View Service

Controller Service

Model Service

Iterate

API View Service

Smartphone View Service

PC Browser View Service

Application Service

Aggregation Service

Domain Service

Repo Service

Domain Service

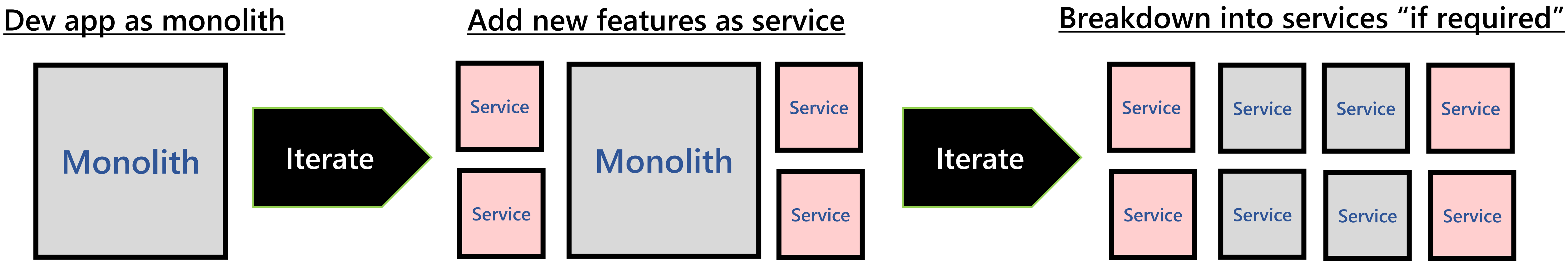
Repo Service

Reference:
https://martinfowler.com/articles/break-monolith-into-microservices.html#GoMacroFirstThenMicro

Methodology: Monolith First

Release app as monolith first, then break down into services

- One of methods to proceed toward microservice architecture
- Direction
  - First development phase: Monolith
  - Iteration phase: Service oriented
- Motivation
  - No clue to find services
  - Feedback from business front-line
- Arguments
  - Don’t start with a monolith (https://www.martinfowler.com/articles/dont-start-monolith.html)



Reference:
http://martinfowler.com/bliki/MonolithFirst.html
https://www.martinfowler.com/articles/dont-start-monolith.html

Methodology: DevOps

Access DevOps readiness

- Do people in your organization know the fundamental practices and principles of DevOps?
- Do teams understand source control tools and their integration with CI/CD pipelines?
- Do you implement DevOps practices properly?
- Do you follow agile practices?
  - Is continuous integration implemented?
  - Is continuous delivery implemented?
  - Is continuous deployment implemented?
  - Is continuous monitoring implemented?
  - Is Infrastructure as Code (IaC) in place?
- Do you use the right tools to support CI/CD?
- How is configuration of staging and production environments managed for the application?
- Does the tool chain have community support and a support model and provide proper channels and documentation?

Reference:
https://learn.microsoft.com/en-US/azure/architecture/guide/technology-choices/microservices-assessment#assess-devops-readiness

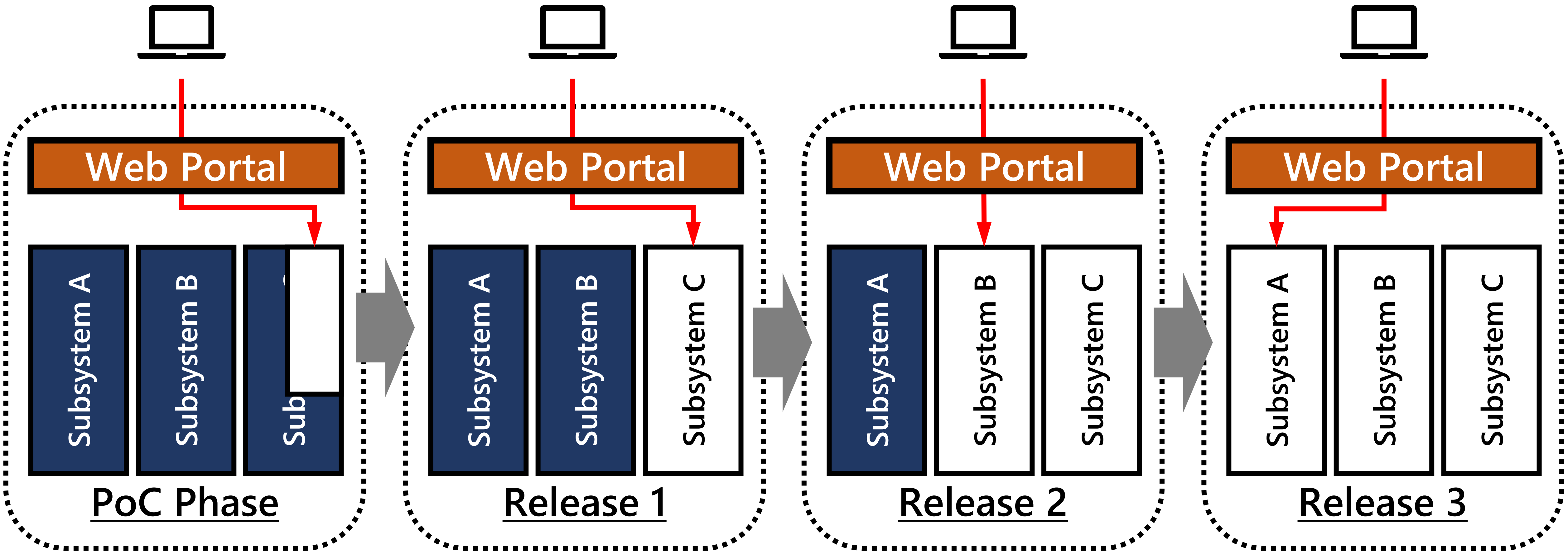
Methodology: Migration with Strangler Fig

Release app timely just after development process completed

- **Strangler Fig**
  - **Pattern to timely release and/or migrate application**
  - **Strangler Facade**
    - Web portal app
    - Hosts application menu screen
    - Rewrite URL to an app after the app development process completed
      - Then a request is routed to the new released app

Reference:
https://martinfowler.com/bliki/StranglerFigApplication.html
https://learn.microsoft.com/en-US/azure/architecture/patterns/strangler-fig

Methodology: Migration with Strangler Fig



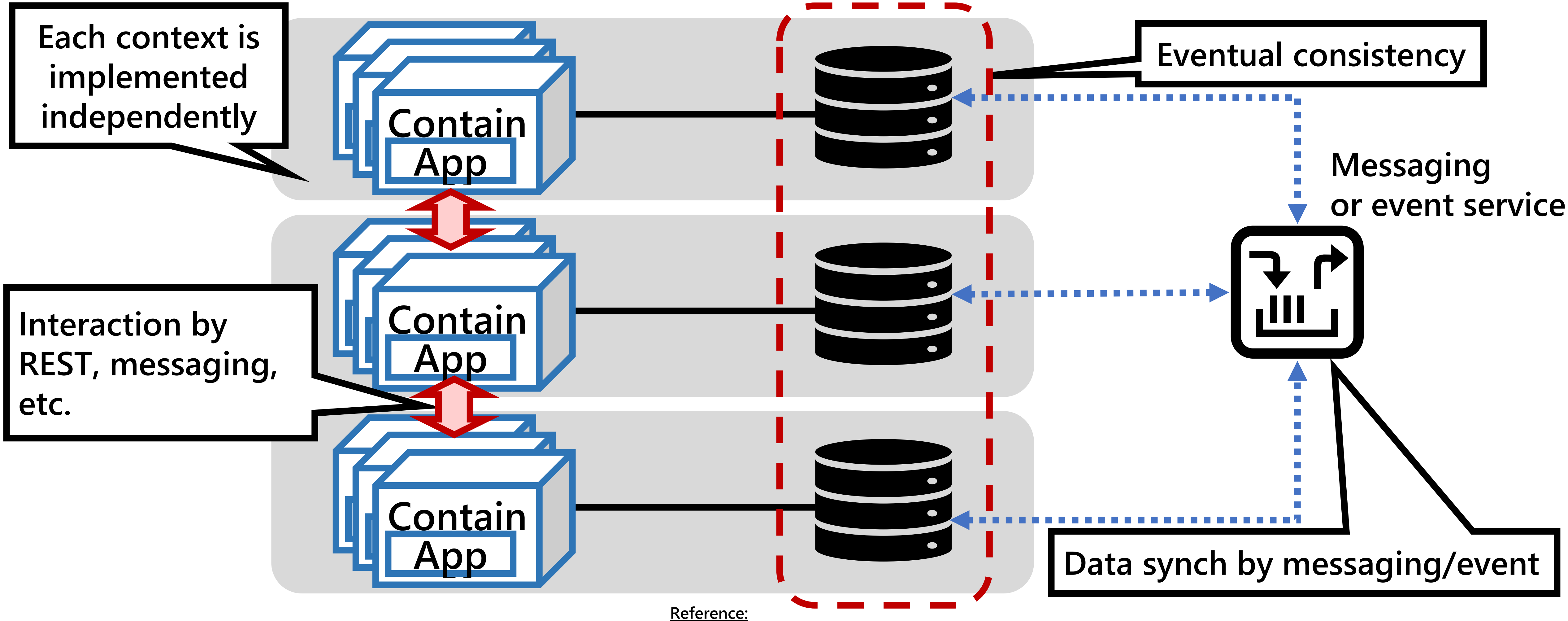
- ✓ Modernization of a part of subsystem C completes
- ✓ Modernization of Subsystem C completes
- ✓ Modernization of Subsystem B completes
- ✓ Modernization of Subsystem A completes

Legend: Monolithic Subsystem | Cloud Native Subsystem

Design: Decouple Core Capability with its Data

A bounded context in a domain is the target of app dev

- Pay attention to data synchronization and consistency



Each context is implemented independently

Interaction by REST, messaging, etc.

Eventual consistency

Messaging or event service

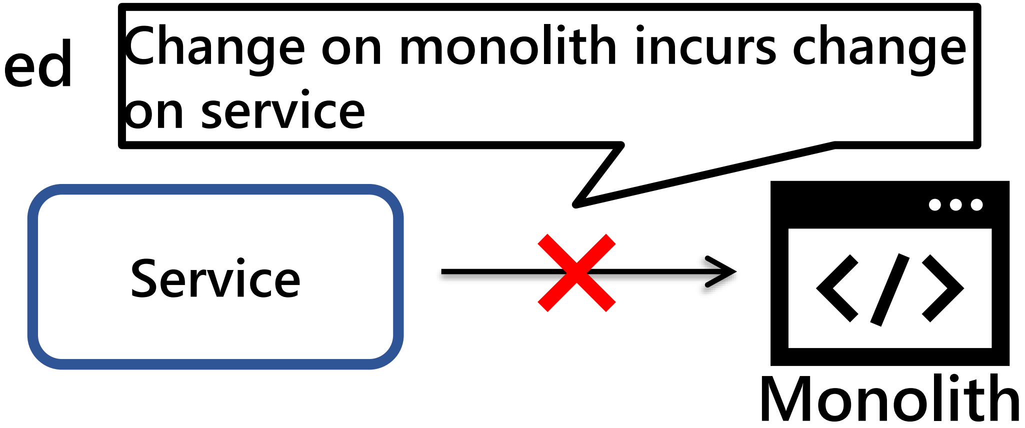
Data synch by messaging/event

Reference:
https://martinfowler.com/articles/break-monolith-into-microservices.html#DecoupleVerticallyAndReleaseTheDataEarly

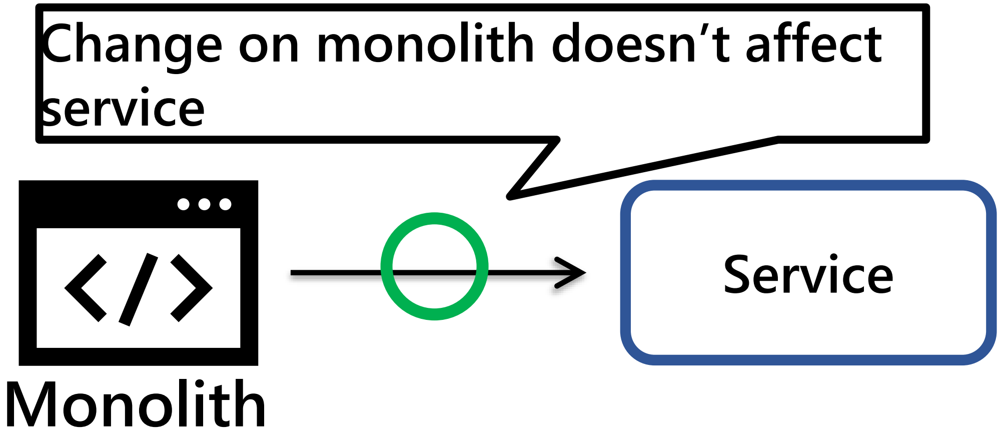
Design: Dependency

Dependency should be from monolith to service

- A domain consists of monolith and modern service in reality
- Design principle about dependency
  - ■Service depends on monolith: Not recommended
    - ◆Monolith may be replaced in the future
    - ◆Future modernization impacts on service, too


Change on monolith incurs change on service

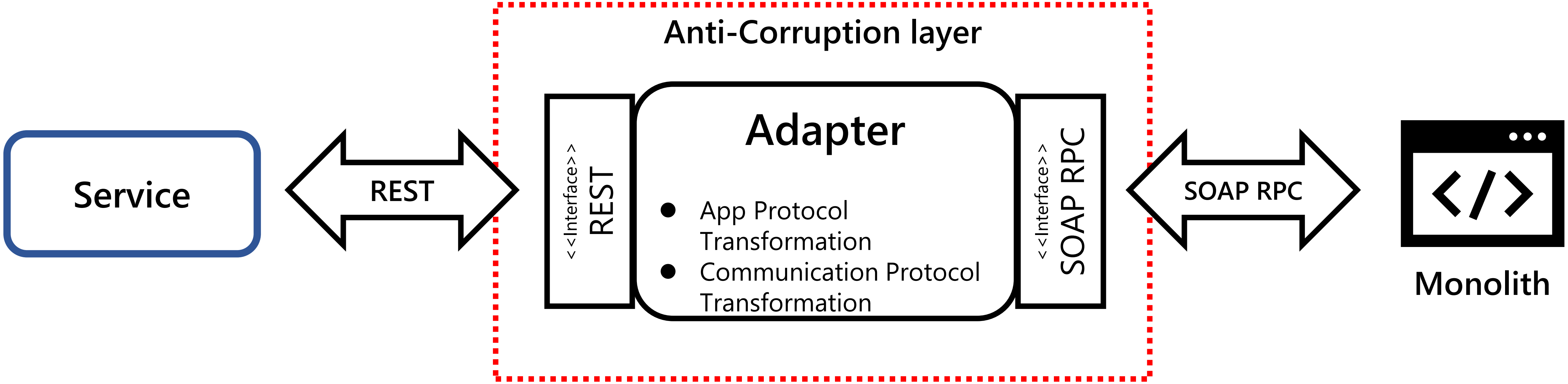
  - ■Monolith depends on service: OK
    - ◆ Change on monolith doesn’t impact on service


Change on monolith doesn’t affect service

Design: Dependency

When service depends on monolith, Consider Anti-Corruption layer

- Anti-Corruption layer
  - Responsible for protocol transformation between monolith and service
  - Implemented as a service



Reference:
https://learn.microsoft.com/en-US/azure/architecture/patterns/anti-corruption-layer

Microsoft

Summary

Summary

- Microservices is the very important technical element of Cloud Native Computing.
- Microservices is the architecture style that structures an application as a collection of services.
- In order to proceed with Microservices, it is recommended to adopt the below
    - DevOps
        - Agile process management
        - Automation by deployment pipeline
    - Domain Driven Design
    - Site Reliability Engineering
    - Containerization
- There're recommended practices for success