Microservices

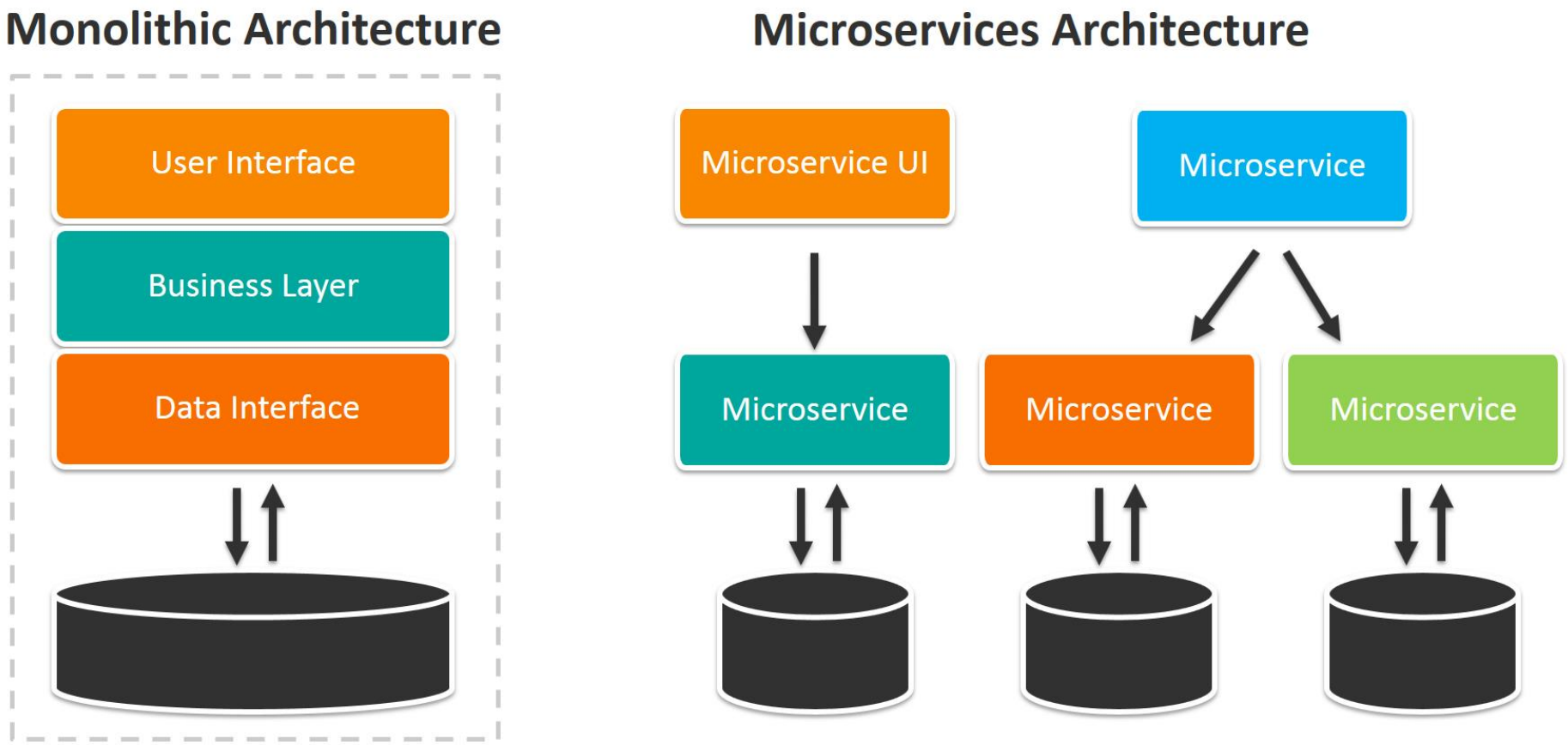
- Pratik Das

Outline

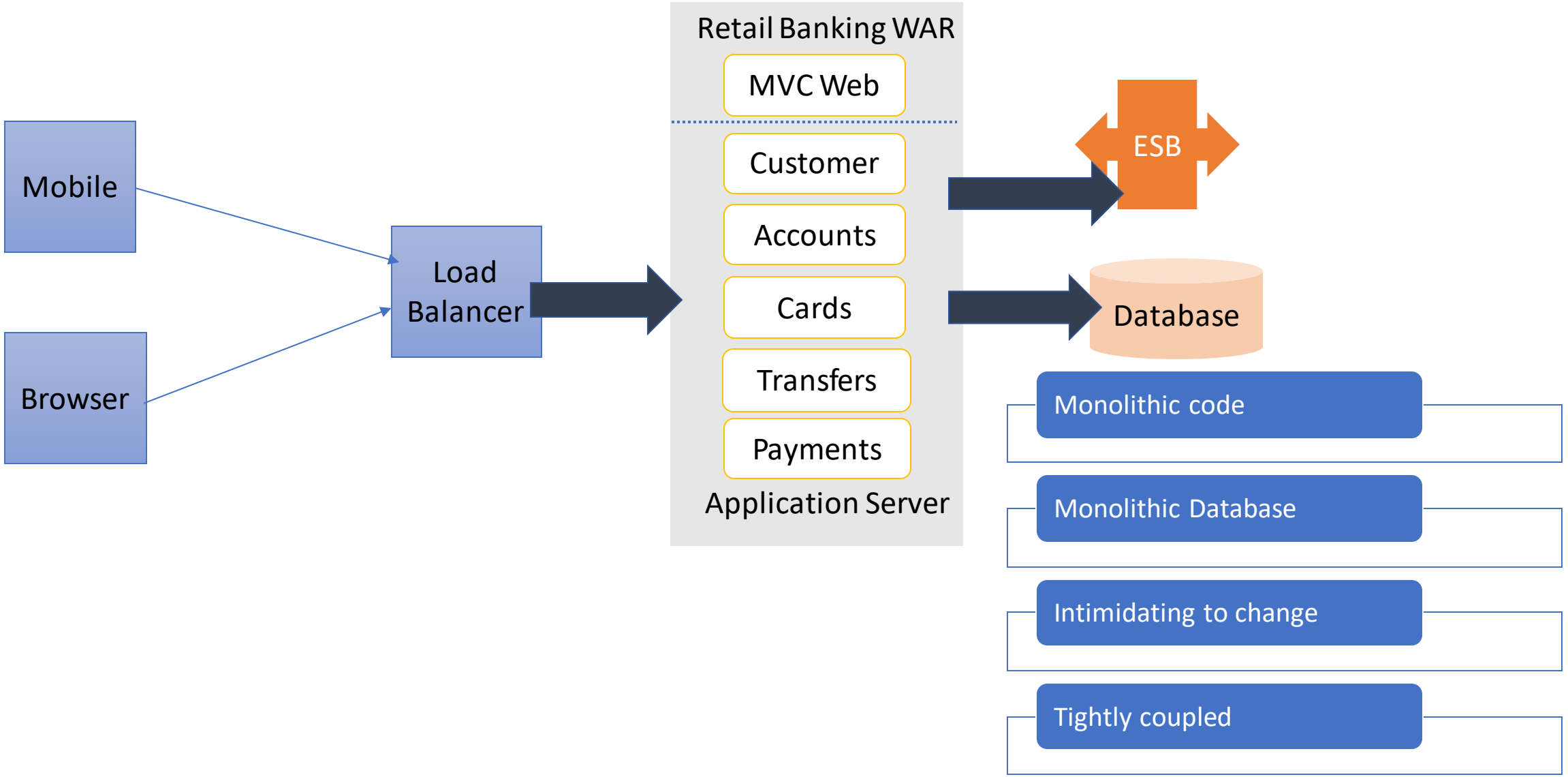
- Microservice Architecture – what, why and when
- Coding Microservices – Spring Boot
- Deploying Microservices – Containers, Docker
- Operating at scale – Container Clusters - Kubernetes

Microservice Architecture – what, why and when

Monolith vs Microservices



Traditional N-tier Applications



Microservice Architecture Principles

Componentization via Services

Single Purpose- Organized around business capabilities.

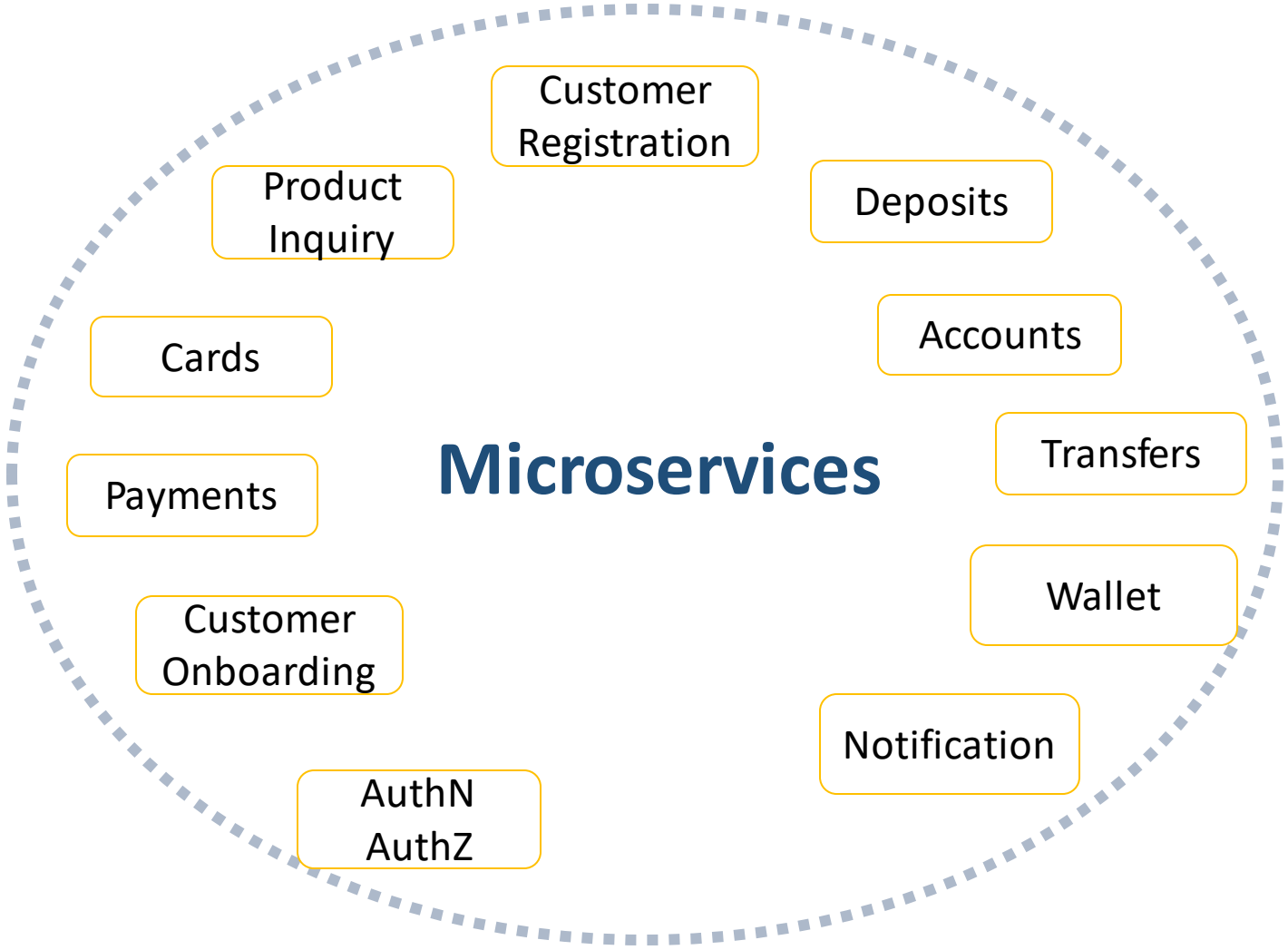
Smart endpoints dumb pipes

Independently deployable

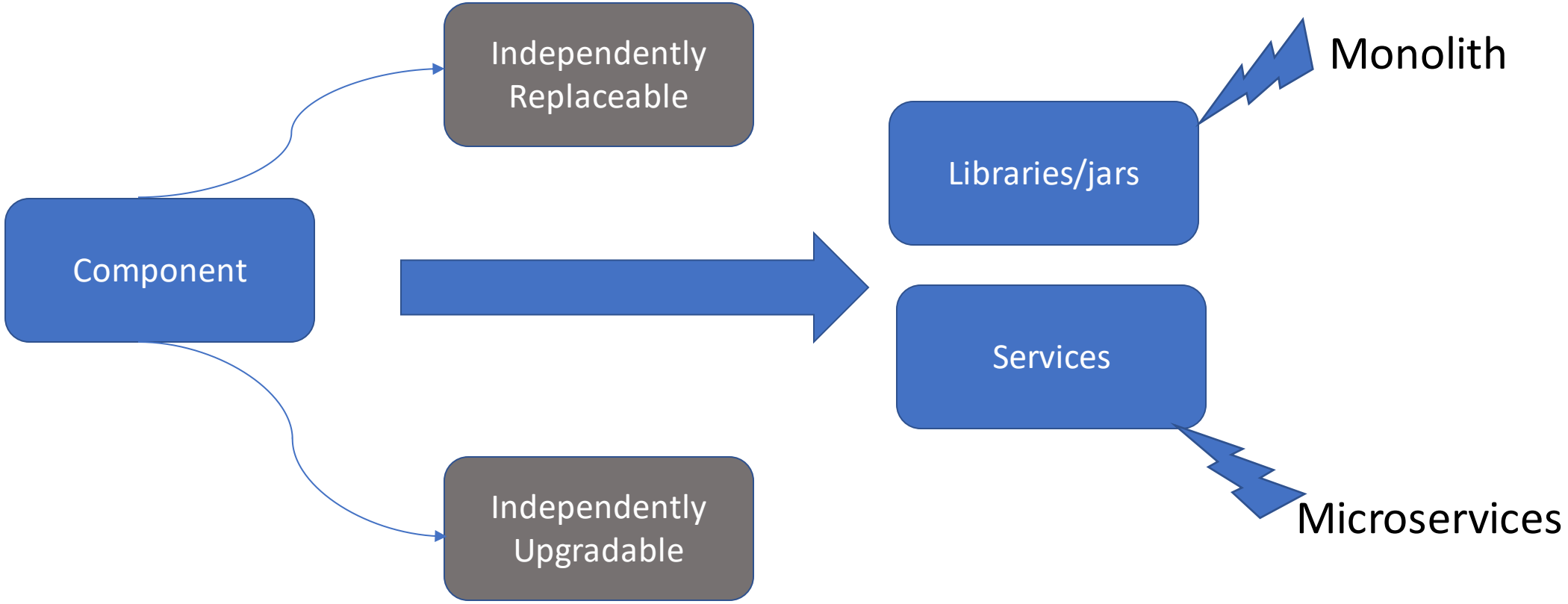
Share nothing(code,data,functionality)

Handle Failures

Culture of automation



MSA Principles – Components as Services



MSA Principles – Organized around business capabilities

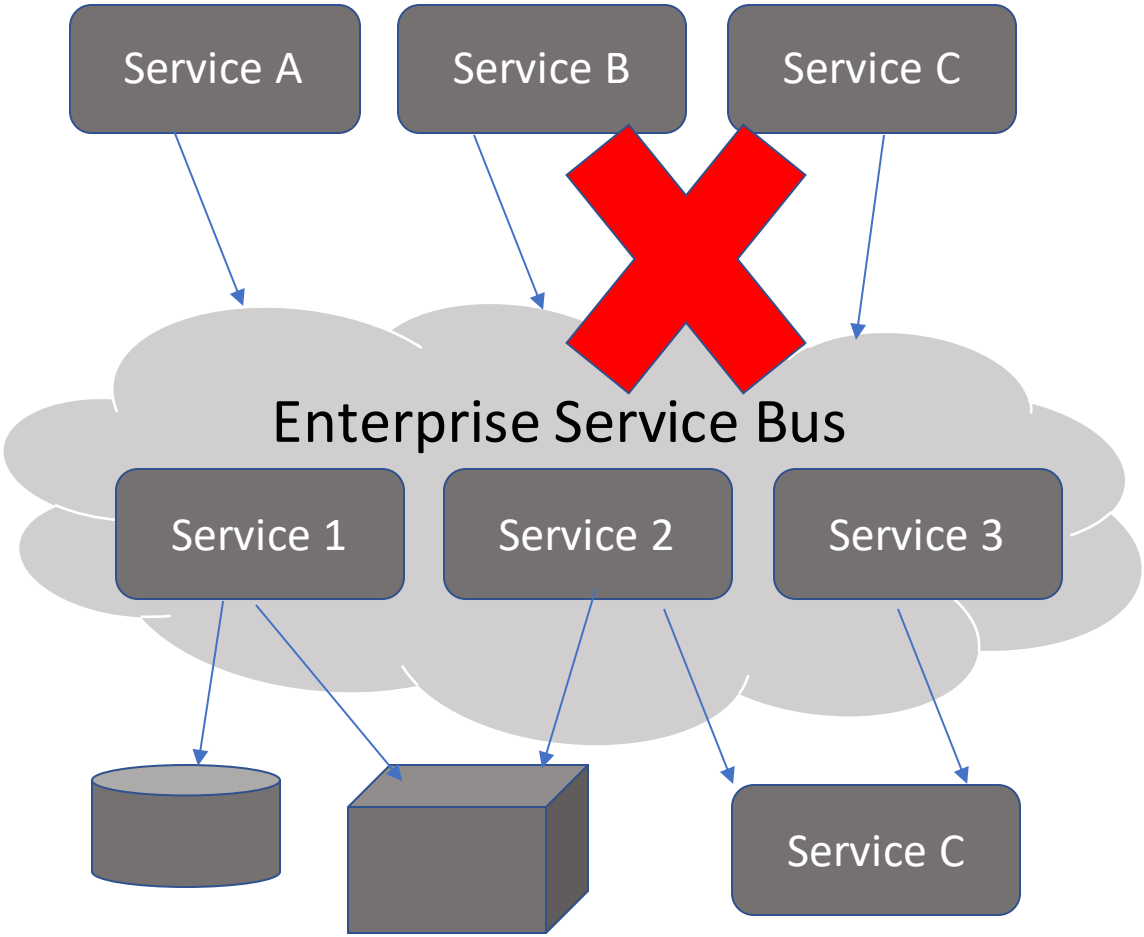
Teams organized by Technology

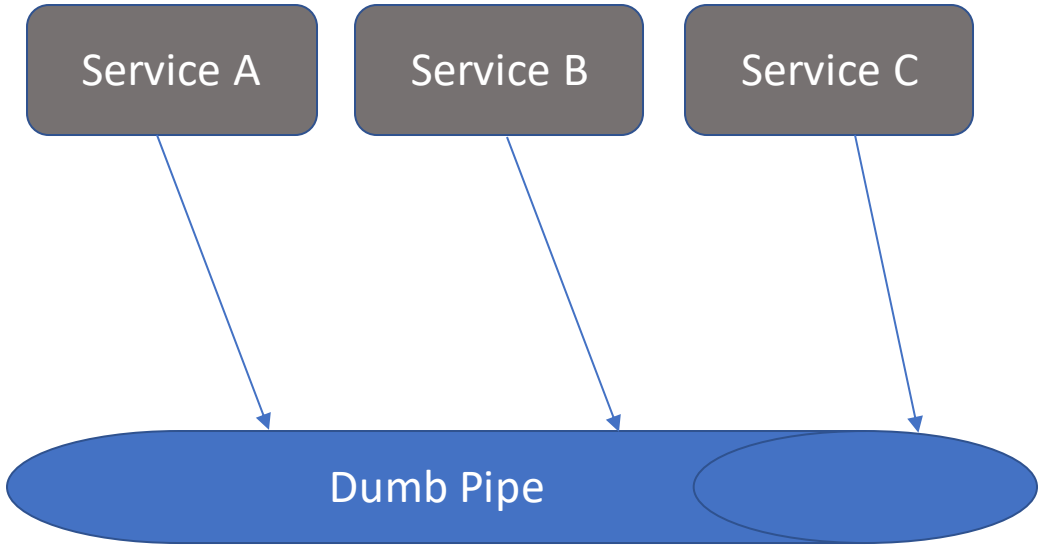
Conway's Law

*Any organization that designs a system will inevitably produce a design whose **structure is a copy of the organization's communication structure***

Teams organized by Business Capability

MSA Principles – Use Smart Endpoints and Dumb Pipes





Test for failures- Chaos Monkey

- **Chaos Monkey**
  - Randomly terminates instances in a cluster
- **Chaos Gorilla**
  - Simulate an Availability Zone becoming unavailable
- **Chaos Kong**
  - Simulate an entire region outages
- **Latency Monkey**
  - Introduce latency to network packets to simulate degradation of the EC2 network
- **Janitor Monkey**
  - Clean up unused resources

Breaking up a Monolith

Monolith – pros and cons

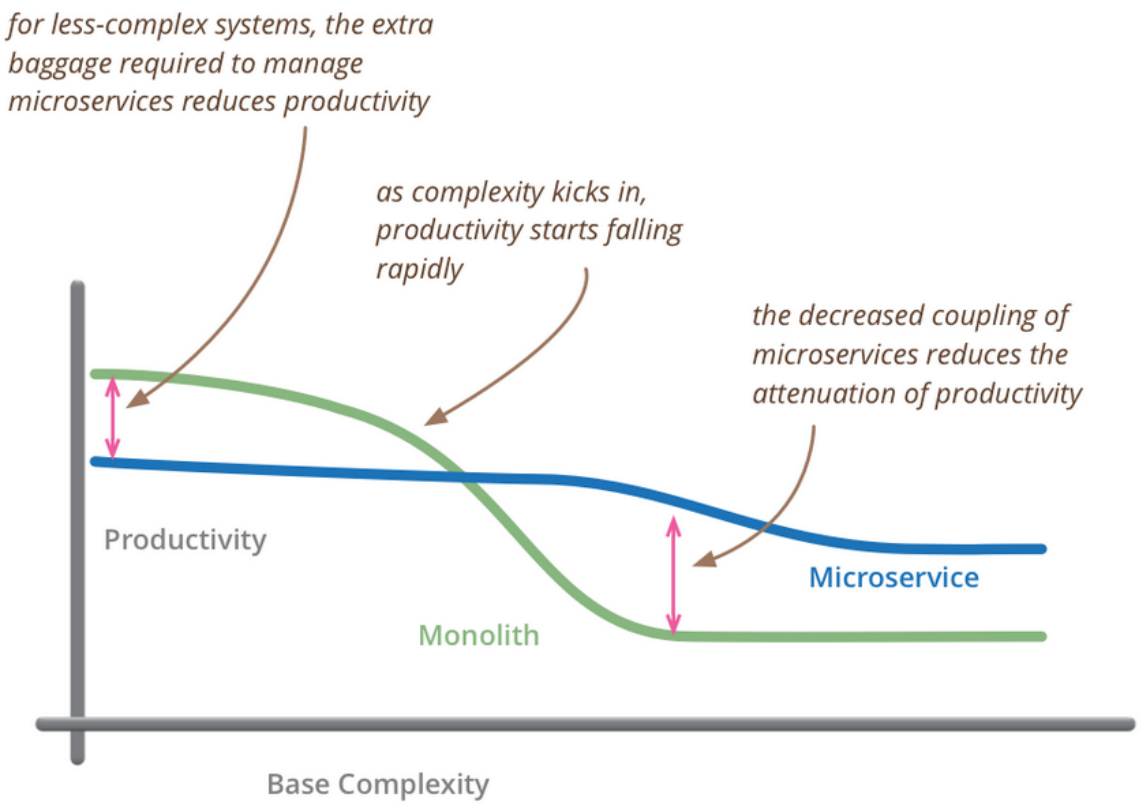
Pros

- Simple
- Easy to deploy
- Test
- Scale

Cons

- Change is Intimidating
- Discourages Frequent Deployments
- Overloads IDE
- High deployment times
- Conflicting scaling requirements of modules
- Constrains update of Technology Stack

When to break a Monolith



but remember the skill of the team will outweigh any monolith/microservice choice

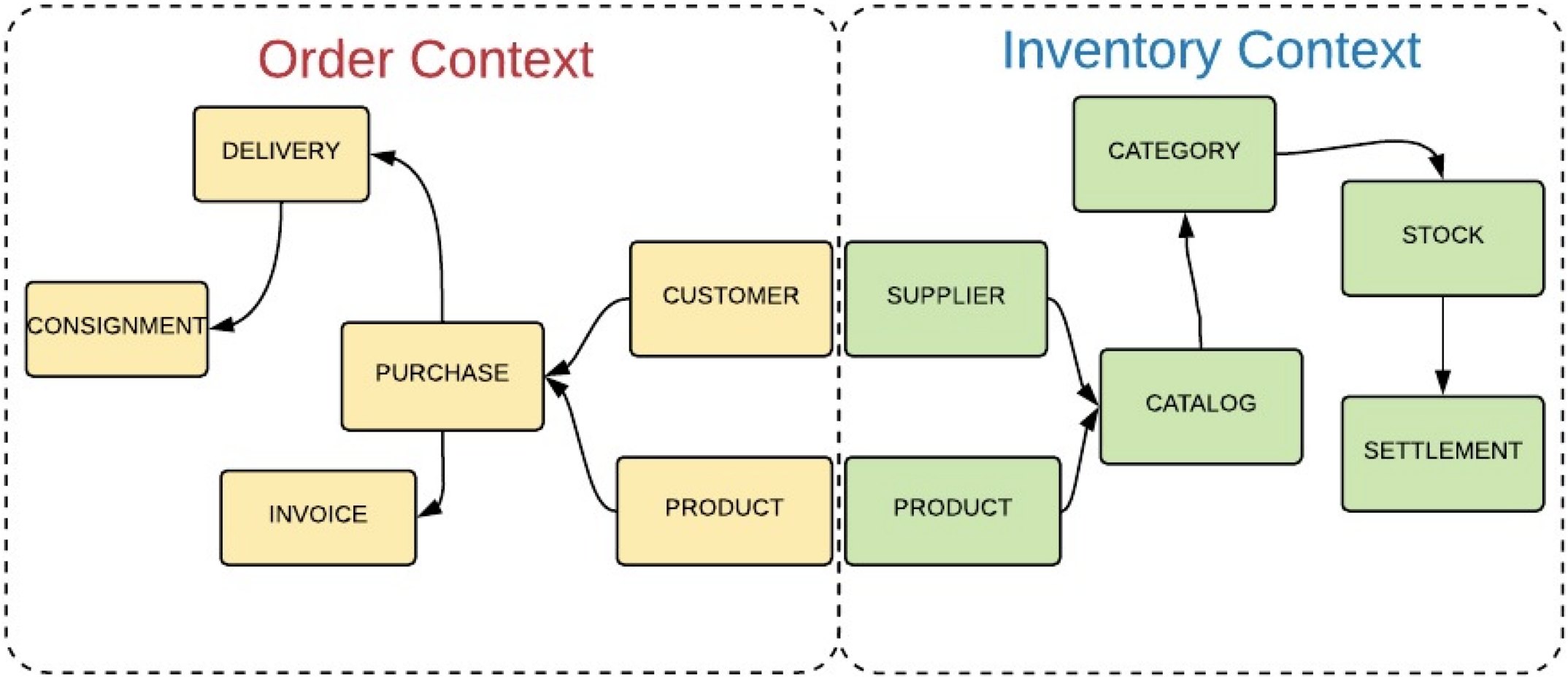
Decomposition Techniques

- Single Responsibility Principle
- Bounded Context-DDD
- Ensure autonomous development and deployment
- Size is linked to autonomous delivery of business capability
- Should have cohesive operations/functionalities
- Start with broad service boundaries. Decompose to smaller ones based on business requirements.
- Apply Strangler pattern for breaking a monolith
- Strive to achieve DRY but be liberal to embrace WET

Clues to Identify Service Boundaries

- Identify service boundaries(namespace, packages) and refactor
- Feature based
- Team location
- Security/transactions vs Inquiry
- Technology fitment

Functional Decomposition using DDD - Bounded Context



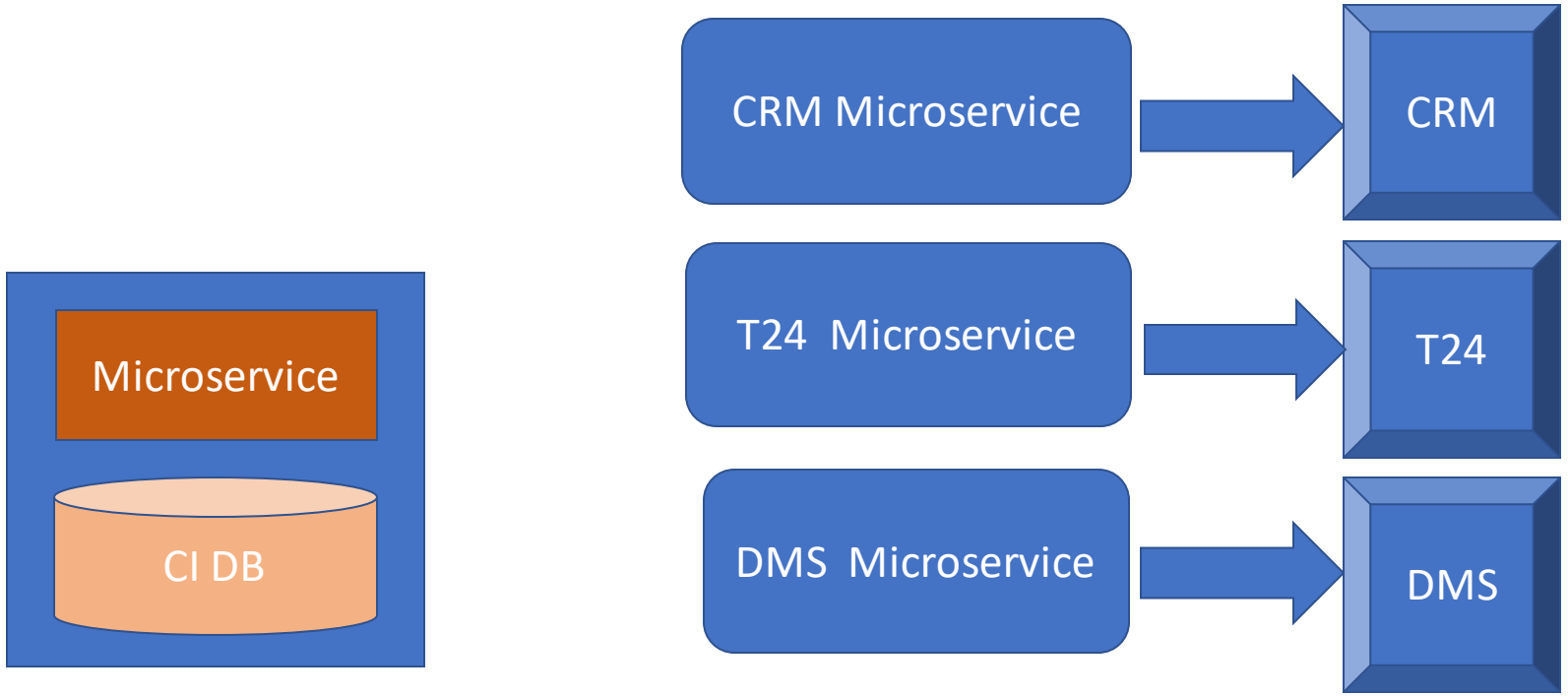
Refactoring databases

- Splitting the database
- Break down Foreign key relationships
- Handle Shared static data (Config files, separate service, duplicate table)
- Extract Shared data modelled in database in separate domain
- Handle Shared table
- Manage transactions
- How to handle Reporting

Integration with legacy and vendor systems

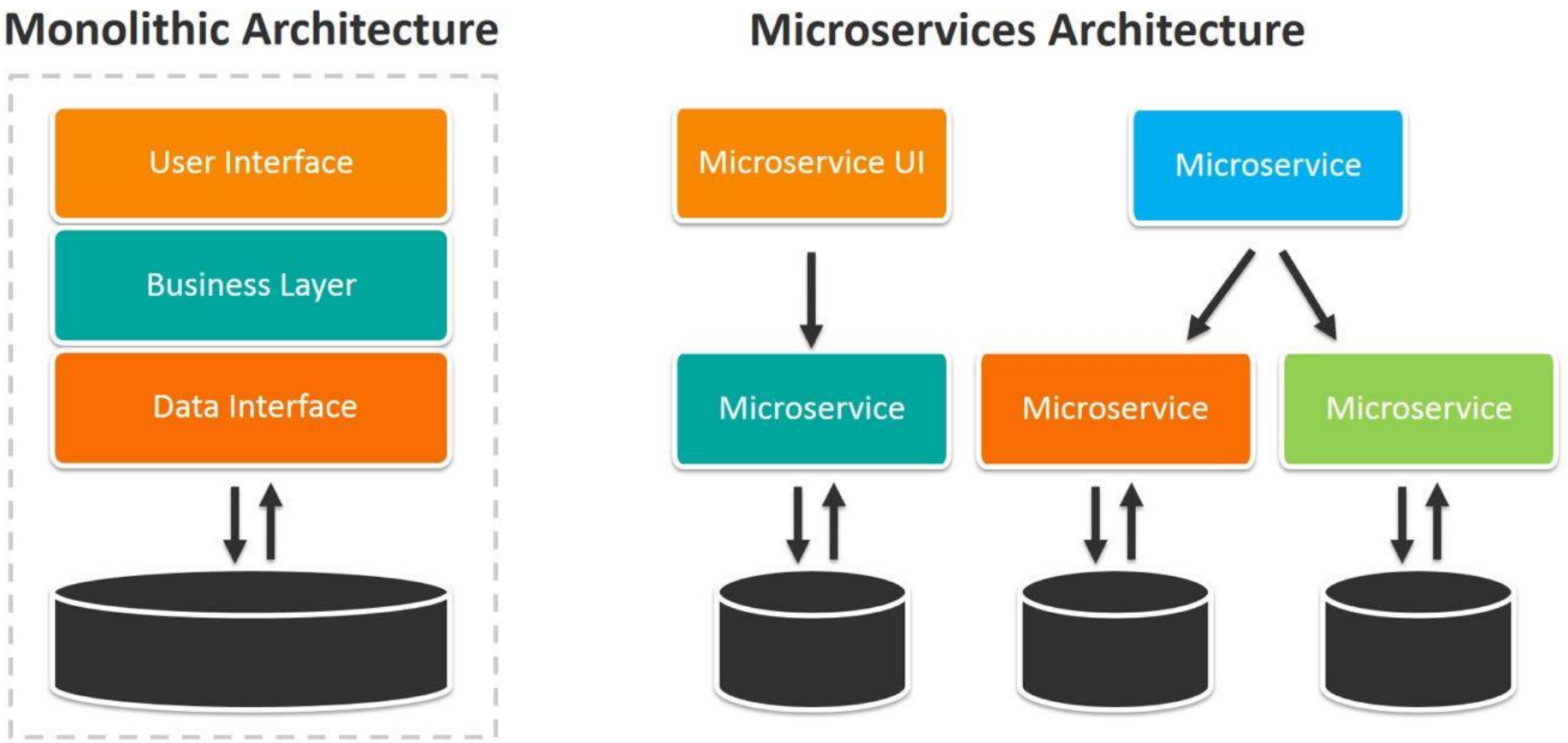
Integration with legacy and vendor systems

- Modelling legacy Application and Datastore as microservice
- Modelling Vendor Systems as Microservice



Data Management

Alternate Data Management to monoliths



Decentralized/Database per service

- Loose coupling == encapsulated data
- Each microservice will have its own private database to persist the data.
- A given microservice can only access the dedicated private database but not the databases of other microservices.
- Updating Database of other microservices if required should be done through API
- Extract Shared database to a separate Microservice

Shared Database

Problem: Monolithic database makes it easy to leverage shared data. How do we use this in Microservices?

Principle: Single System of Record

- Every piece of data is used by 1 service
- Every other data is read-only copy

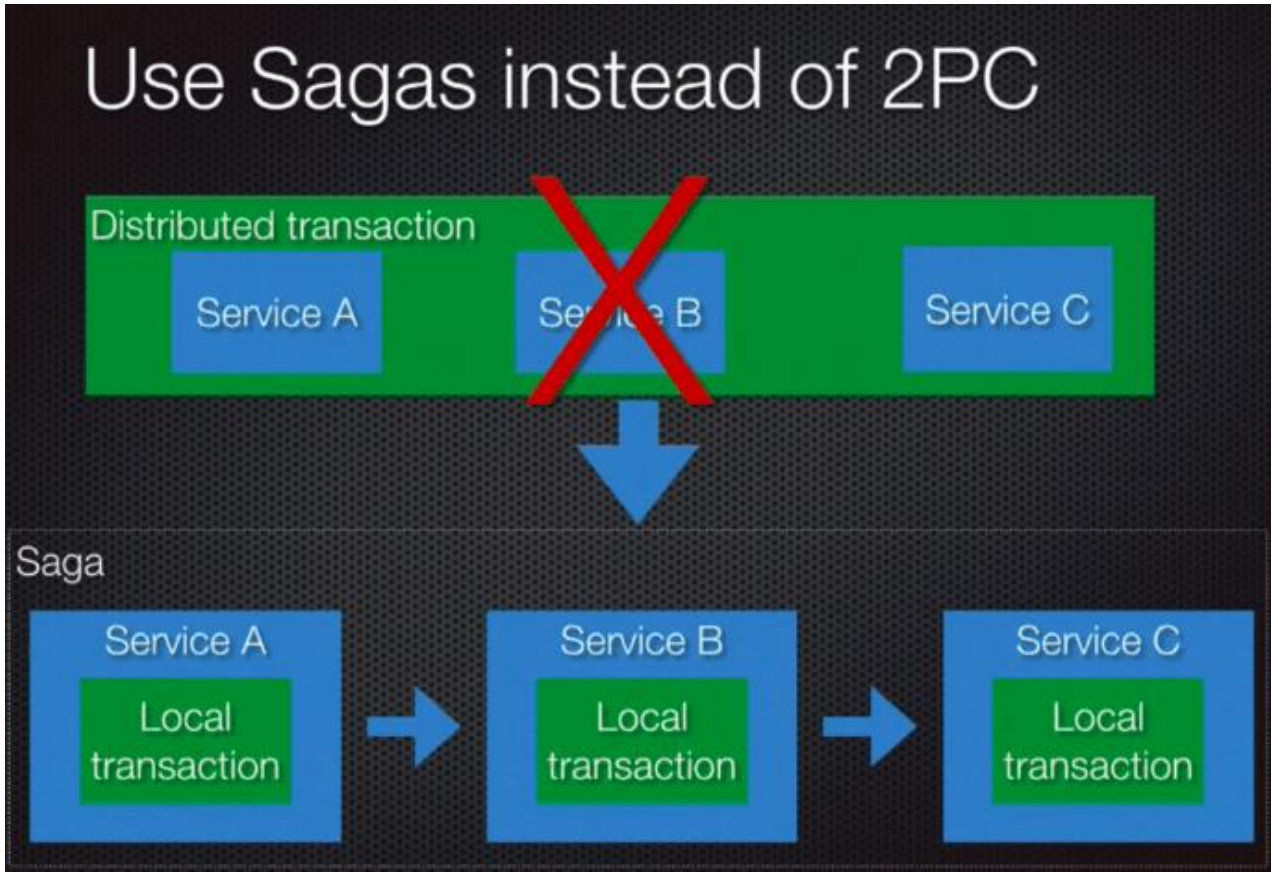
Manage Transaction- SAGAS

- 2 PC is not an option
- CAP theorem – impacts availability
- Reduced throughput due to locks
- Single point of failure
- Use sagas to maintain data consistency across services
- Use transactional messaging to make sagas reliable

Saga

Successful Saga

- Start Saga
- Start T1
- End T1
- Start T2
- End T2
- Start T3
- End T3
- End Saga

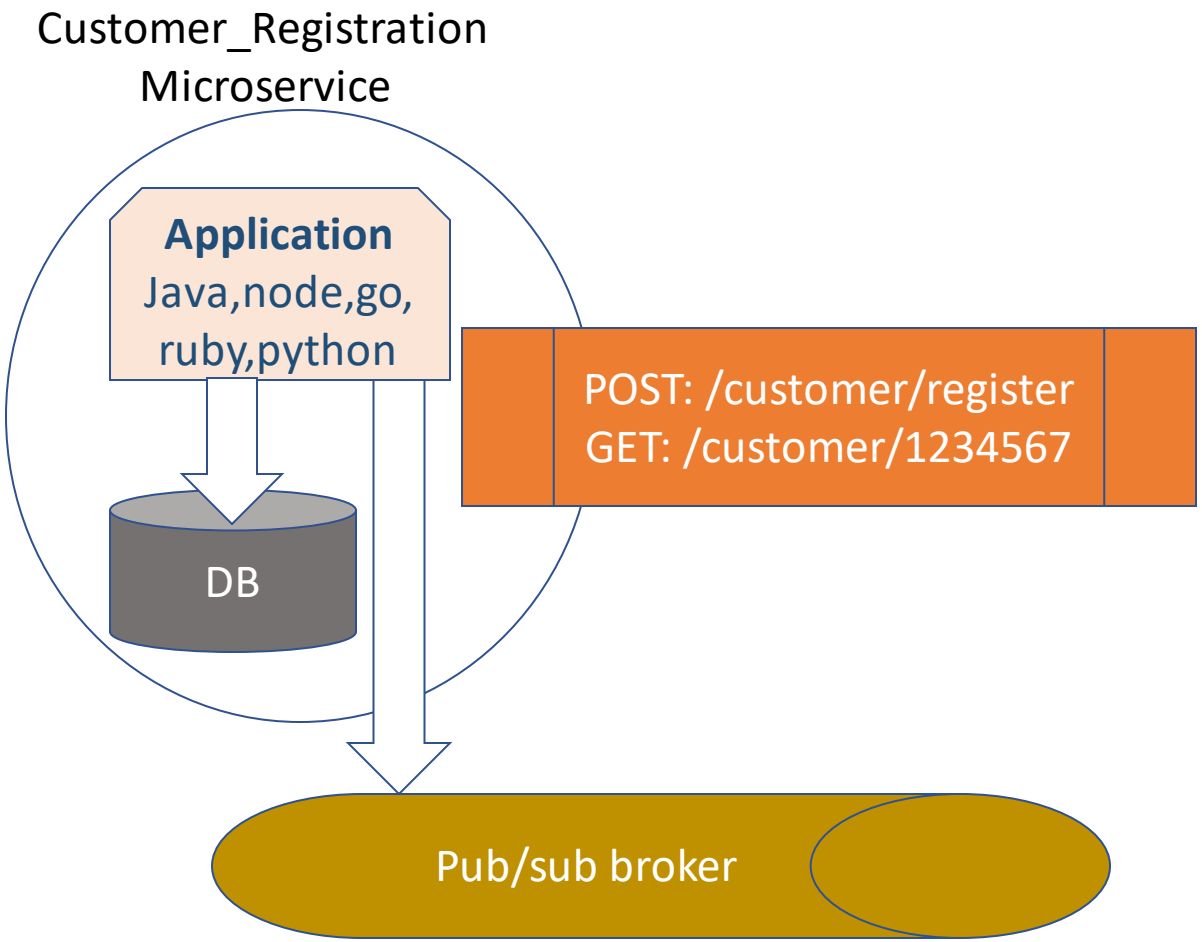


Failed Saga

- Begin Saga
- Start T1
- End T1
- Start T2
- Abort Saga
- Start C2
- End C2
- Start C1
- End C1
- End Saga

Microservice Anatomy

Web Application- read from db, publish event



Coding a Microservice

Dev frameworks-traits

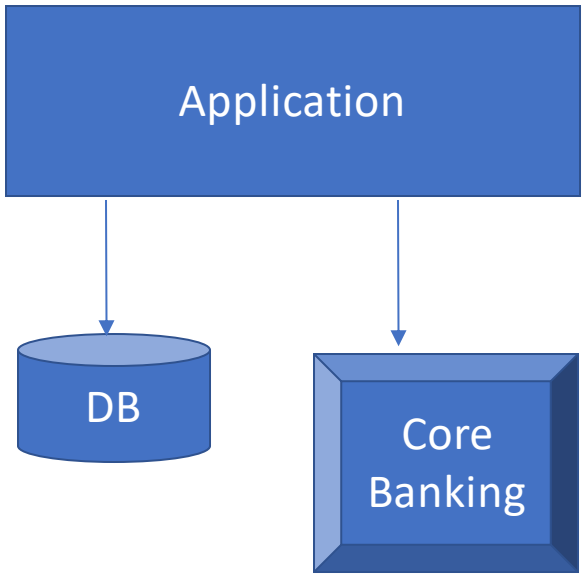
- Service Discovery
- Service registry
- Load Balancer
- Synchronous communication
- Asynchronous communication
- Resilience-circuit breakers, timeout, retry, fallback

How to design Microservice

Example – Retail Banking

Application features

- Customer Registration
- Product Inquiry
- Transfer Funds



What is a Good Service

- Loose coupling – Do not share internal details
- High Cohesion- Related behaviour sits together

Decomposition – using DDD bounded context

- Data is kept in silos, or Bounded Contexts
- All bounded context are treated as 1 big Application
- Only 1 Bounded Context is responsible for updating 1 datastore
- Handle Stale data by subscribing to changes in data of interest
- Embrace Eventual Consistency
- Cache only data of interest

Languages and Frameworks

- Java: Jakarta EE, Spring boot, Spring Cloud, Microprofile, Micronaut
- NodeJS: Express, Moleculer
- Golang: Go Micro, Go Kit, Gizmo, Kite
- Ruby: Sinatra
- Python: Django, Flask
- .Net: Service Fabric

Spring Boot

- Opinionated Framework
- Convention over configuration
- Starter packs- Automatic config with defaults
- Basic REST App components: Resource, Service, Repository

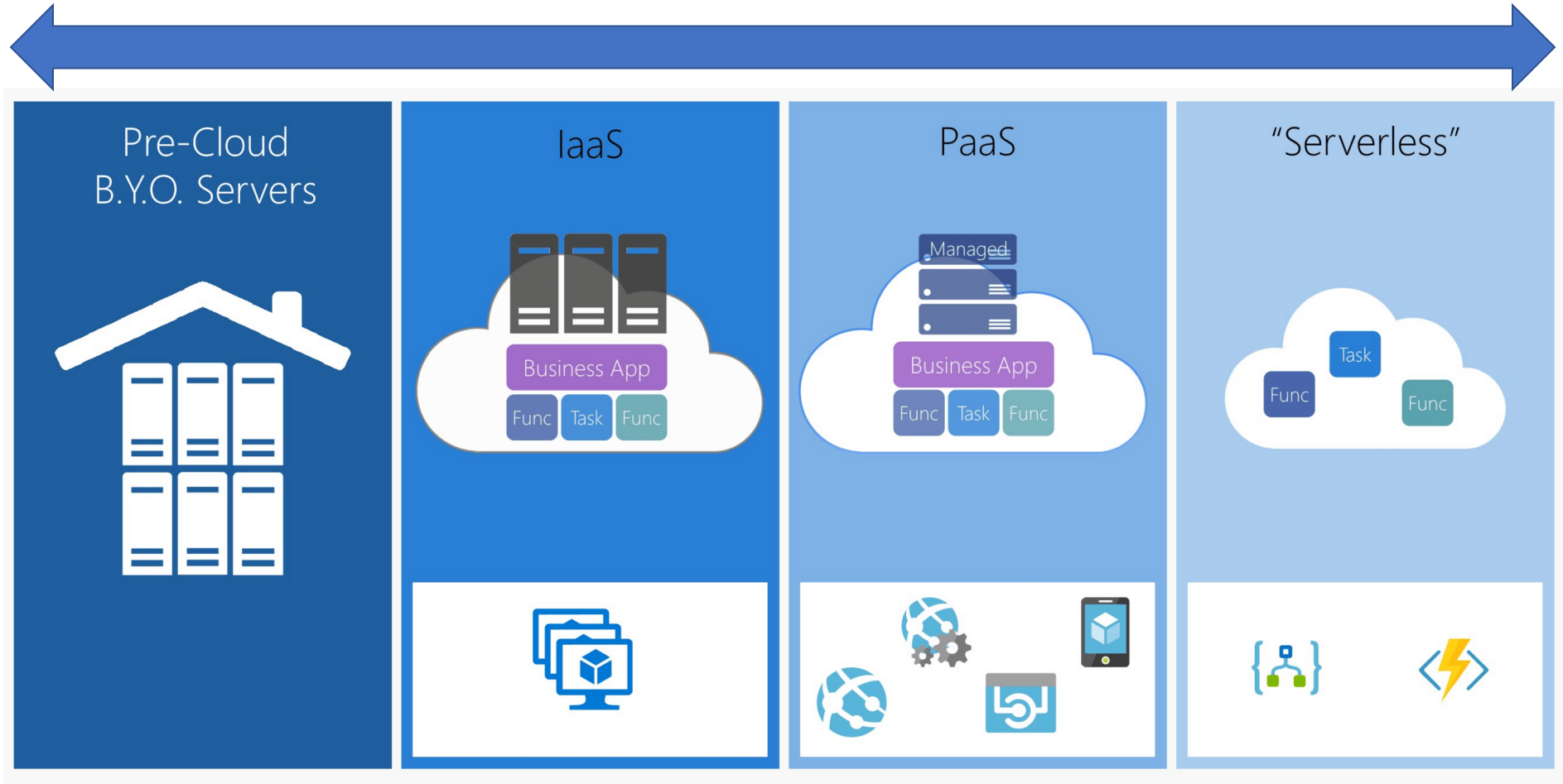
Spring boot example

- CustomerInquiry Microservice

➢ Components:

- ❖ REST endpoint : /customer/{id}
- ❖ CustomerResource
- ❖ CustomerInquiryService
- ❖ CustomerRepository
- ❖ Spring Application

B.Y.O, IAAS, PAAS, FAAS



Takeaways

Takeaways

- Monolith is not an antipattern and good for small applications
- Implementing Microservices is complex but has many benefits
- Essentially about functional decomposition
- Use DDD bounded context to identify Microservices
- Database and config are part of the service
- Microservice Anatomy- RESTful apps or FAAS/serverless
- Multiple options for languages and frameworks