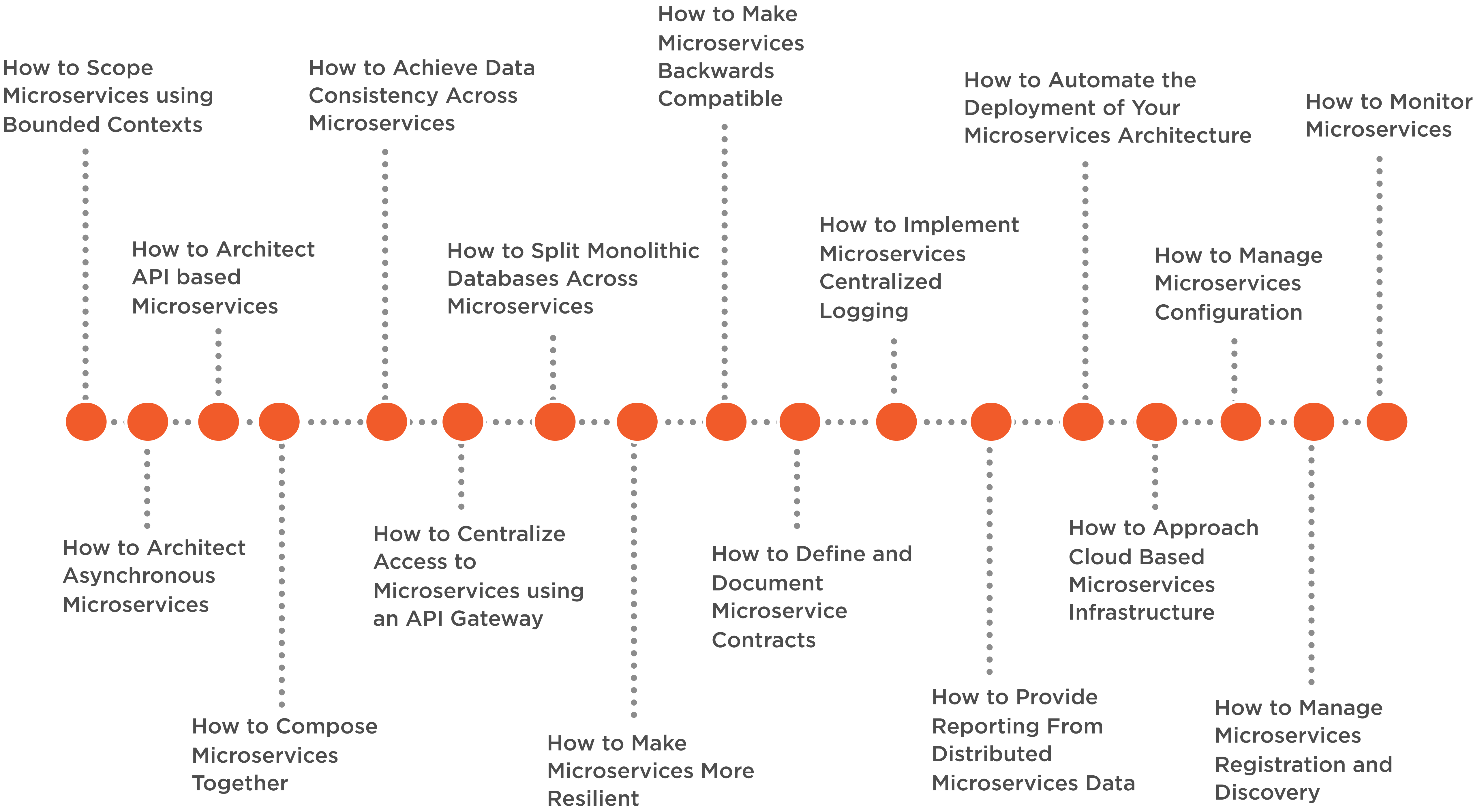
How to Split Monolithic Databases Across Microservices

Rag Dhiman

@ragdhiman www.ragcode.com

Microservices Architectural Design Patterns Playbook



- How to Scope Microservices using Bounded Contexts
- How to Architect Asynchronous Microservices
- How to Architect API based Microservices
- How to Compose Microservices Together
- How to Achieve Data Consistency Across Microservices
- How to Centralize Access to Microservices using an API Gateway
- How to Split Monolithic Databases Across Microservices
- How to Make Microservices More Resilient
- How to Make Microservices Backwards Compatible
- How to Define and Document Microservice Contracts
- How to Implement Microservices Centralized Logging
- How to Provide Reporting From Distributed Microservices Data
- How to Automate the Deployment of Your Microservices Architecture
- How to Approach Cloud Based Microservices Infrastructure
- How to Manage Microservices Configuration
- How to Manage Microservices Registration and Discovery
- How to Monitor Microservices

Microservices Architectural Design Patterns Playbook

Microservices Architecture

Rag Dhiman

@ragdhiman www.ragcode.com

Microservices Architectural Design Patterns Playbook

Rag Dhiman

@ragdhiman www.ragcode.com

Overview

Introduction

Approach to Database Design

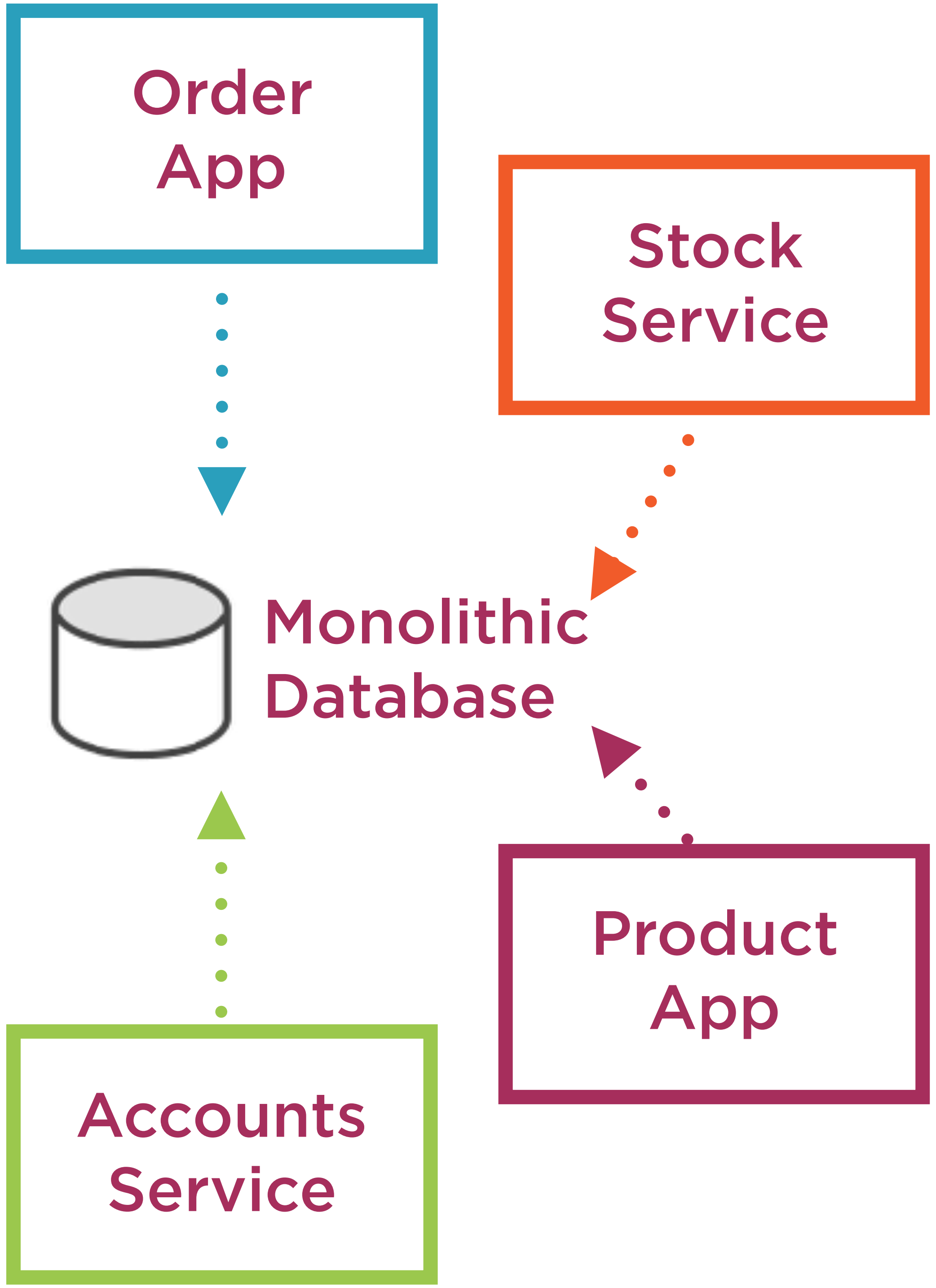
Patterns for Database Design

Greenfield Database Approach

Brownfield Migration Strategy

Introduction

Introduction



Monolithic database

- Provides ability to share data easily

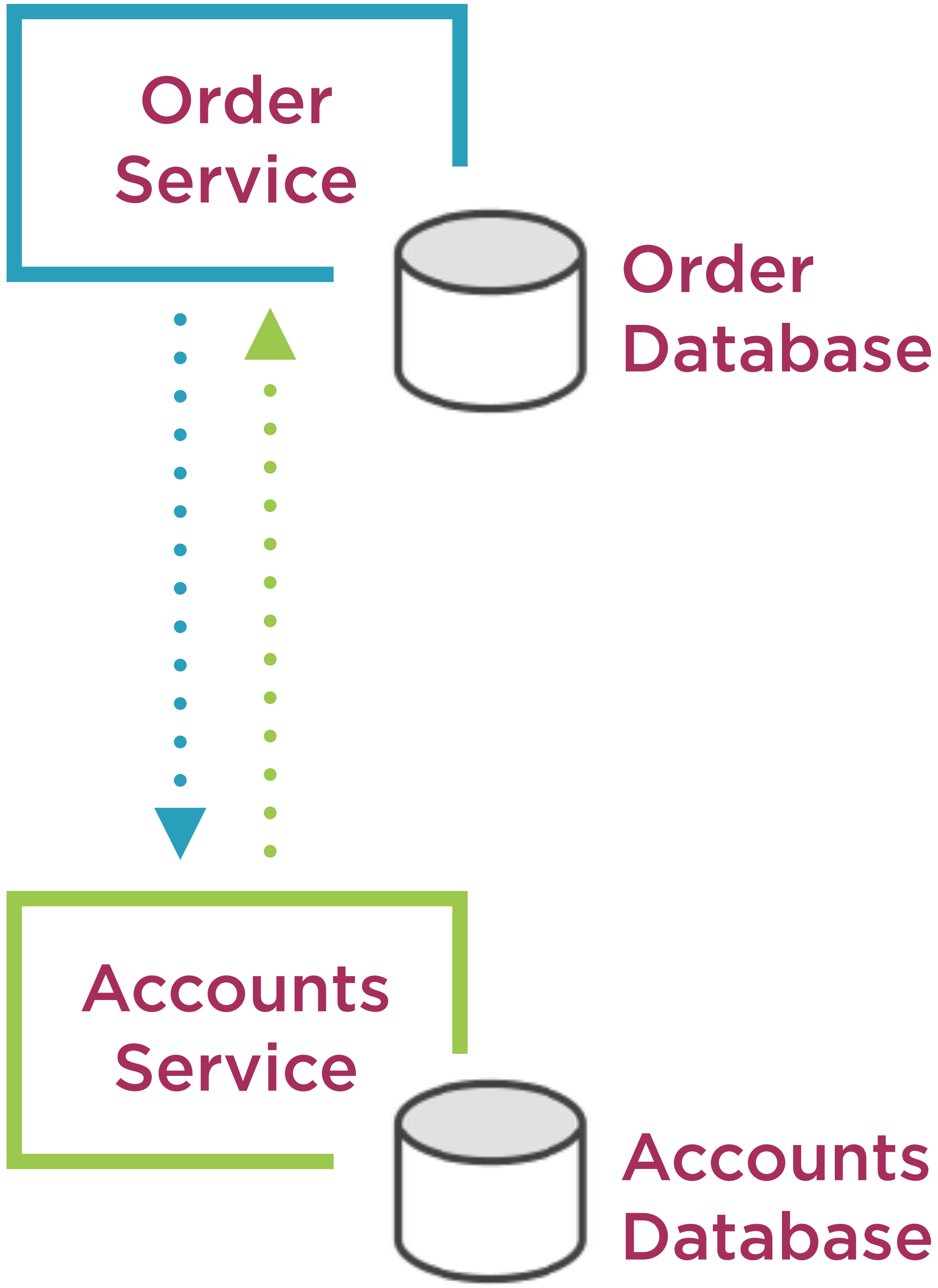
Why you should avoid

- Need to independently change
- Need to independently deploy
- Avoid tight coupling
- Harder to scale out
- Performance bottleneck

Microservices databases

- Database per service
- Microdatabase

Microdatabase How?



Approach to database design

- Avoid data first design
- Function first

Patterns for Database Design

- Sharing data using events
- Store data as events
- Separate write model from the query model

Greenfield database scenario

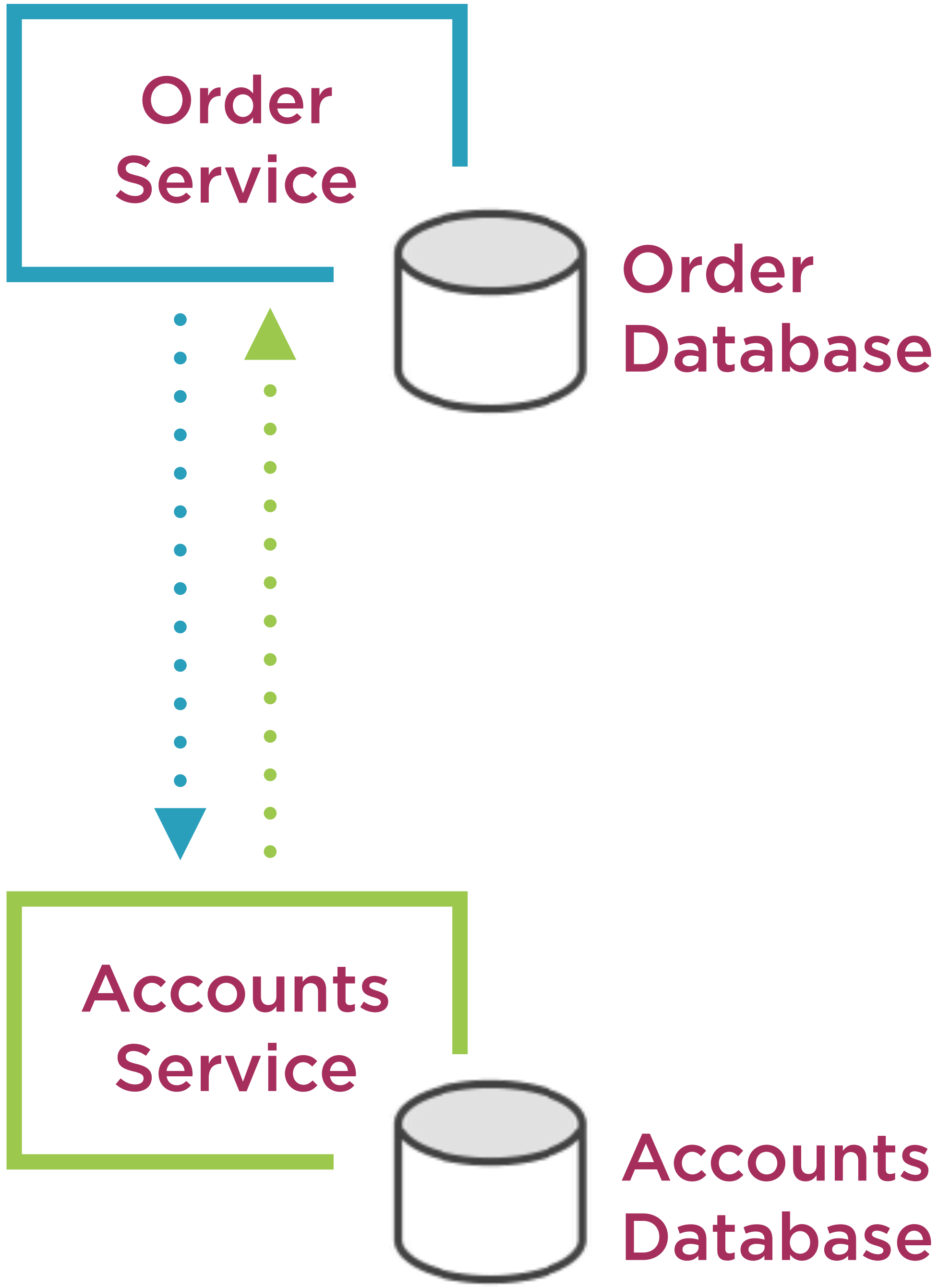
- Approach and design patterns

Brownfield database scenario

- Migration strategy

Approach to Database Design

Data First vs. Function First



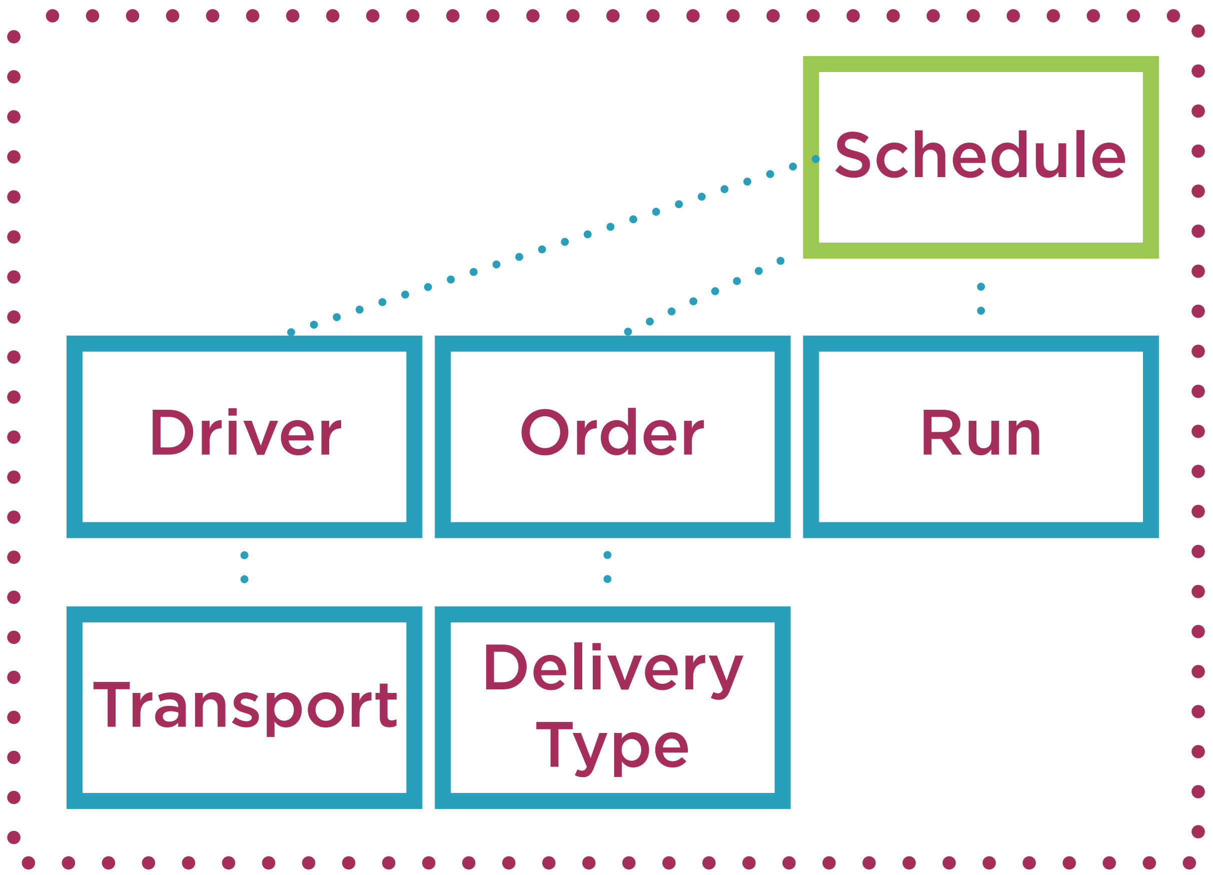
Data first design leads to tight coupling

- Traditional approach
- Microservices anti-pattern
- Anaemic CRUD based services
- Exposing internal data structures

Function first design for loose coupling

- Aim is not just to share data
- APIs that provide more than CRUD
  - Interfaces that provide function
  - Function defined by the scope of the service

Function First: How?



Top down approach to database design

- Application and its function come first

Bounded contexts as a design tool

- Microservices that represent a function

Microservices with contracts for function

- Separate internal models from external

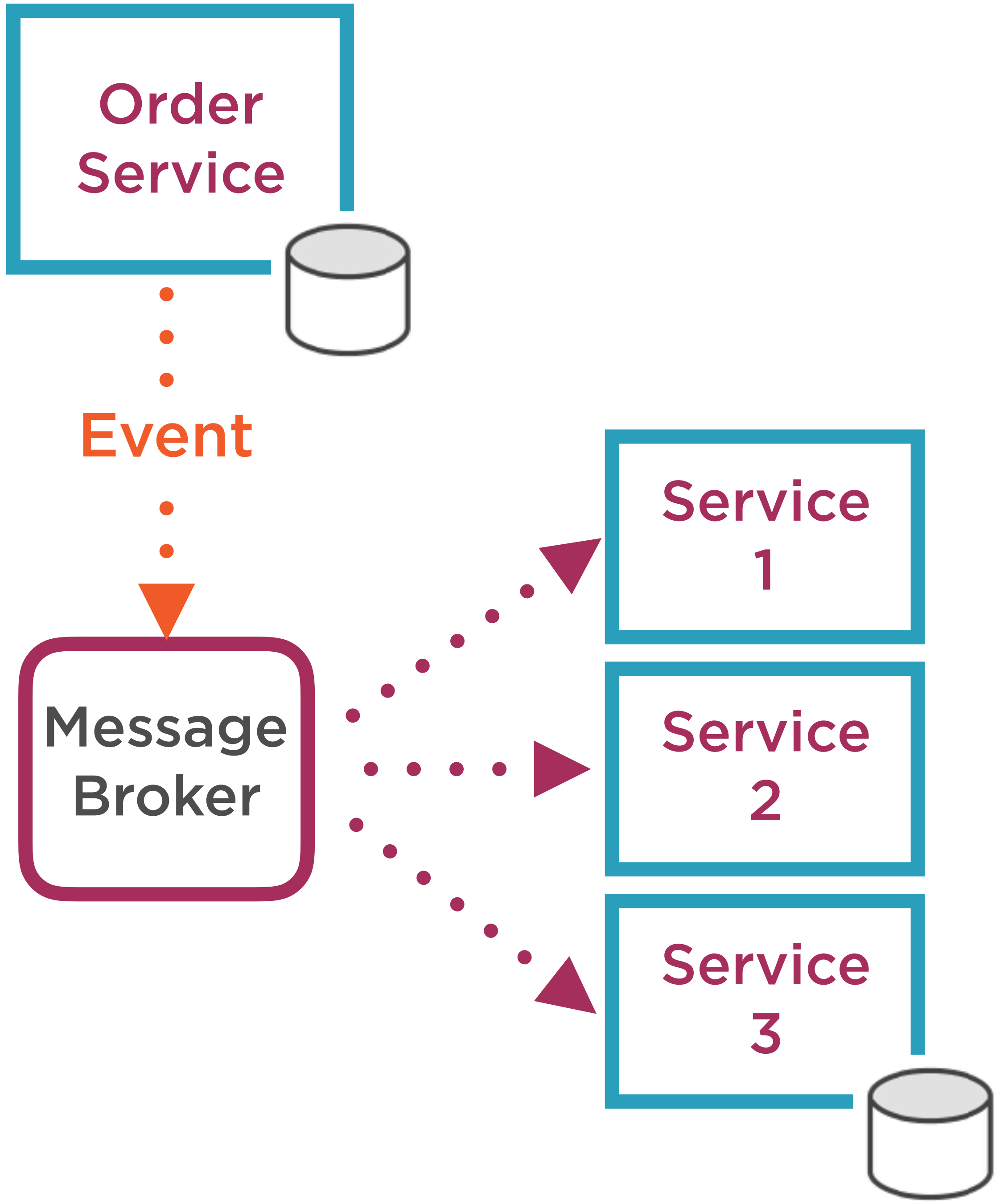
Supporting microdatabase implementation

- Code first approach to database design
- Frameworks to support code first

Function defines data store technology

Patterns for Database Design

Event Driven



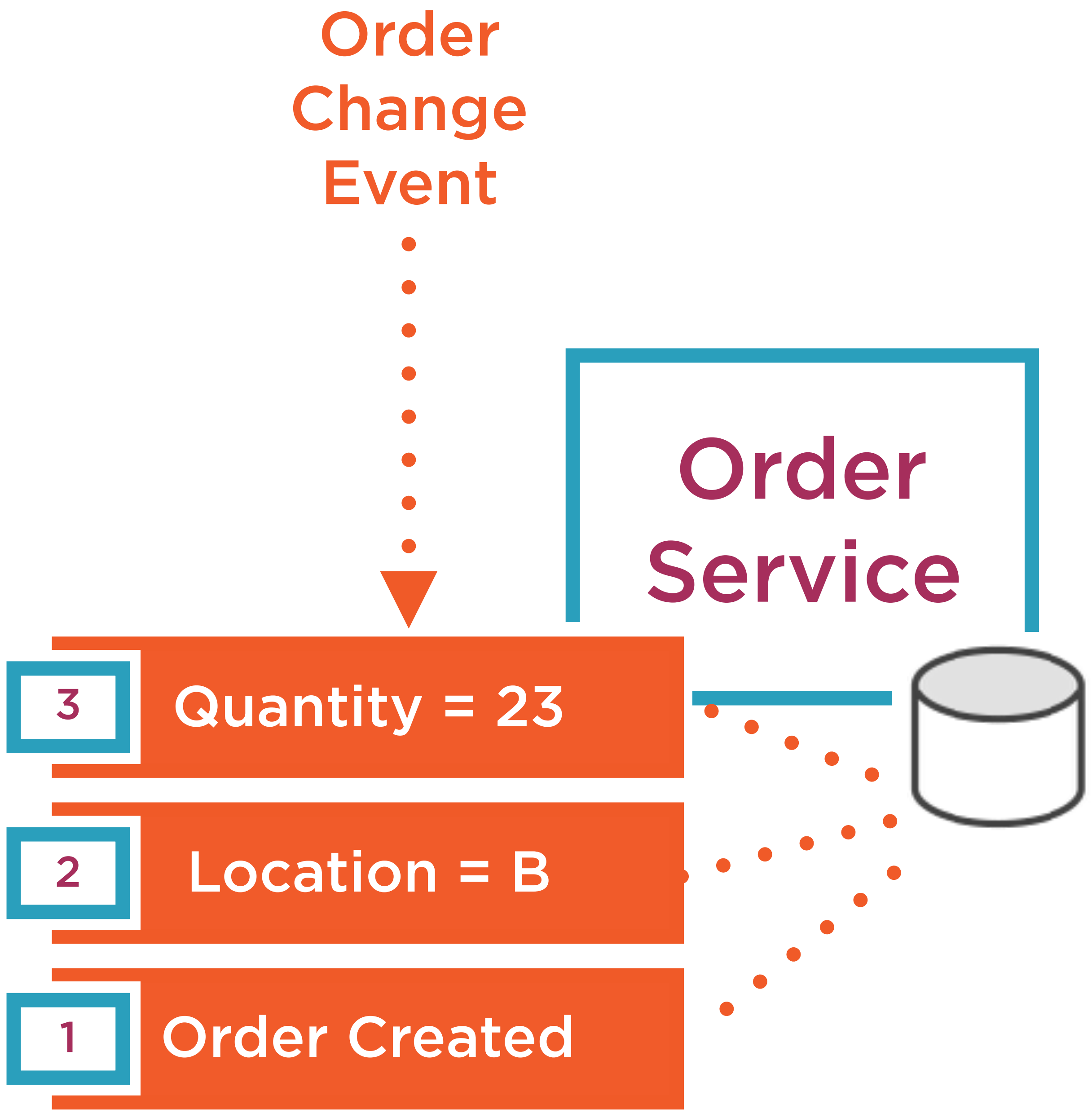
Sharing data changes using events

- Way to avoid shared databases
- Interested parties are subscribers
- Interested parties store data locally
  - Local cache for data of interest
  - Data for joins

Events are ideal for microservices

- Avoids anemic CRUD services
- Push is better than pull
- Asynchronous communication
- Use of message brokers
- Decoupled architecture

Event Sourcing



Traditional approach to storing state

- State of data recorded as a record(s)

Event sourcing is an alternative approach

- State is stored as a series of events
- An event record states what has changed
- Replay events to get current state

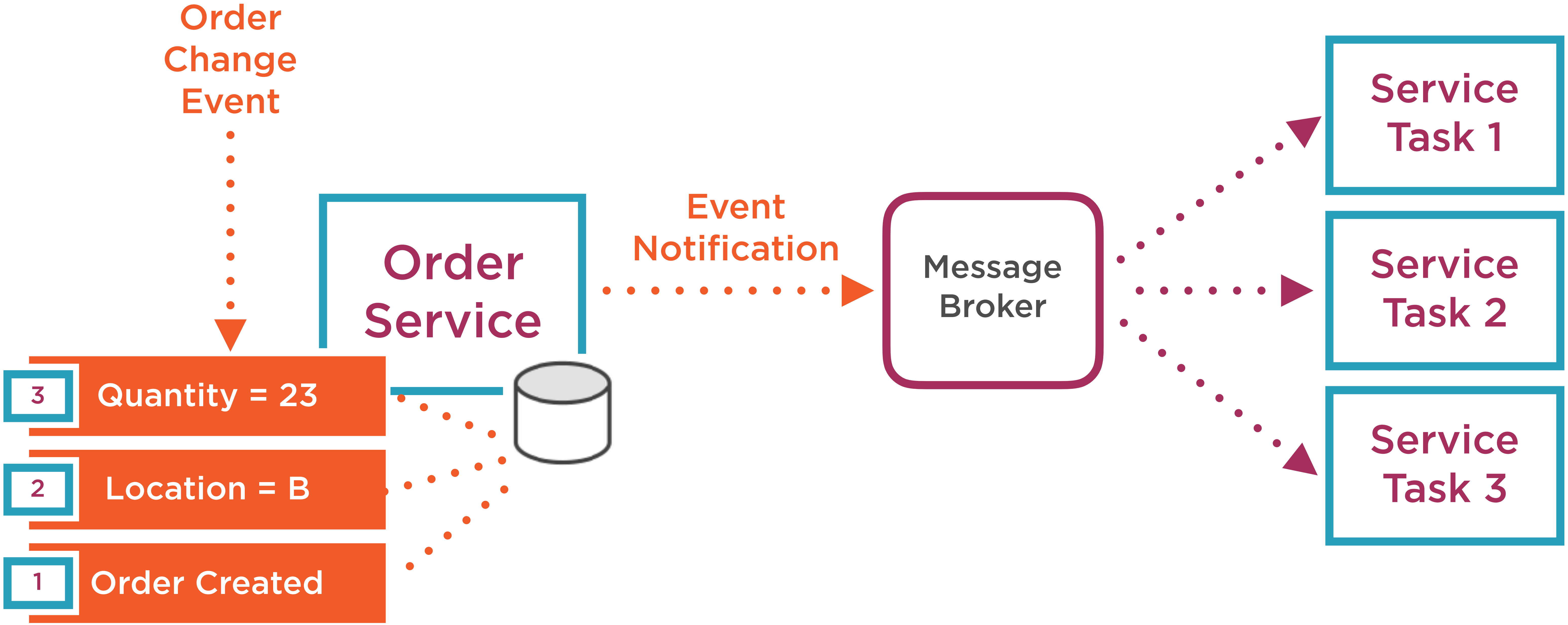
Why use event sourcing for microservices

- Data is shared using event notifications
- Join back to the correct state of the record

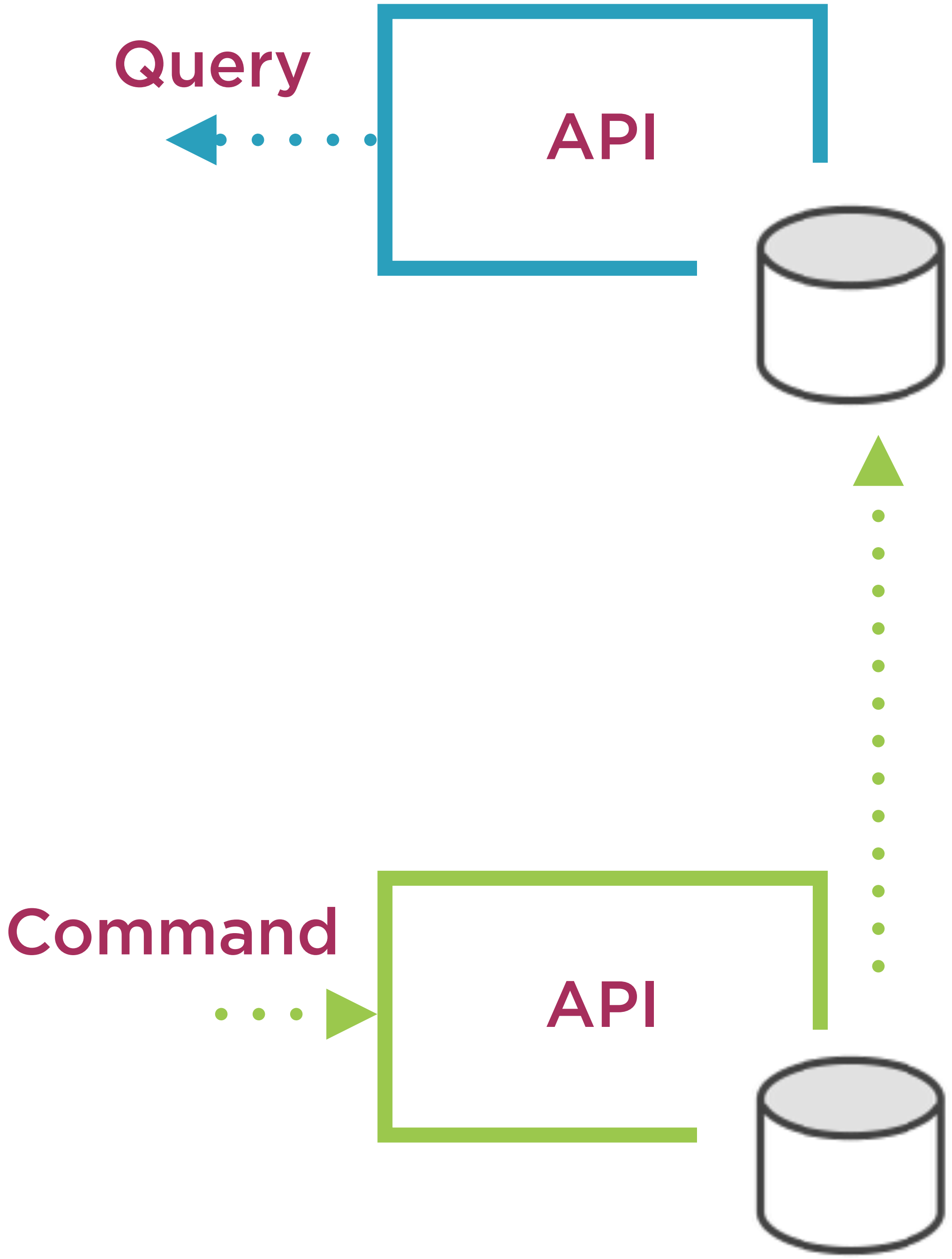
Challenges to resolve

- Regular snapshots to increase performance
- Regular snapshots to decrease storage

Event Sourcing



Command Query Response Segregation



CQRS

- Command models and or services
- Query models and or services

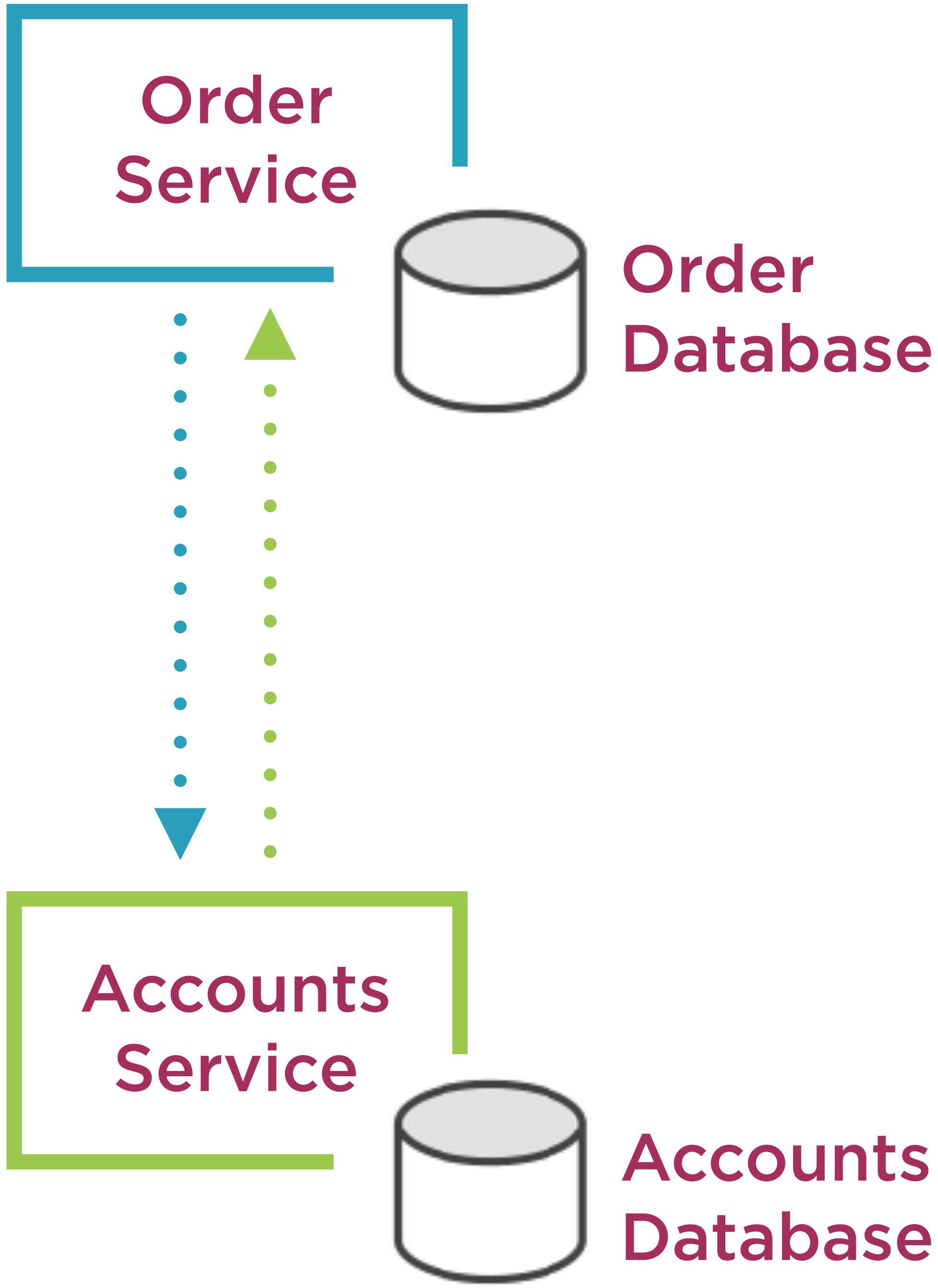
Why

- Separation of responsibilities
  - Event notifications handled by command
  - Reporting/functions handled by query
- Separation of technologies
  - Service and storage

Challenges

- Command and query database syncing

CQRS How



Command microservice that receives events

- Subscribes to a queue
- Multiple subscribing microservices
- Using command models store event

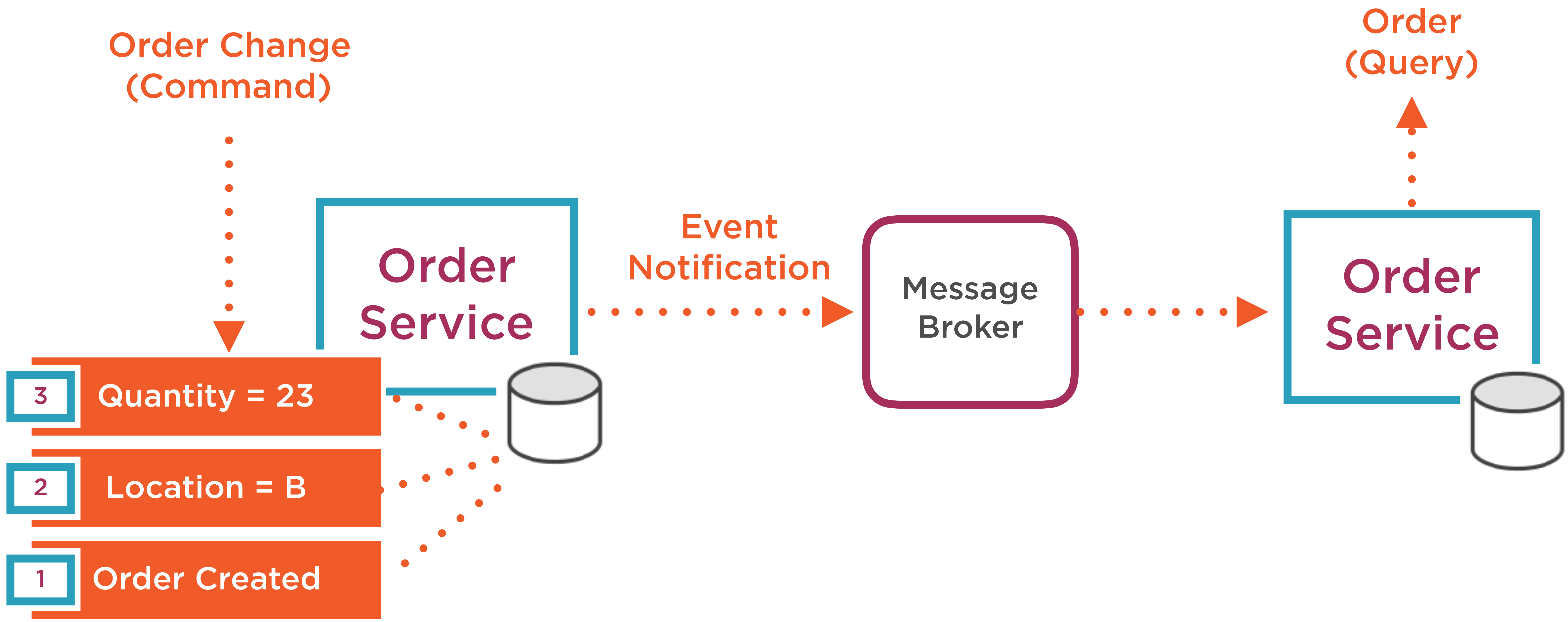
Query microservices for data

- In the form of a function
- In the form of data retrieval

Process to sync command store to query store

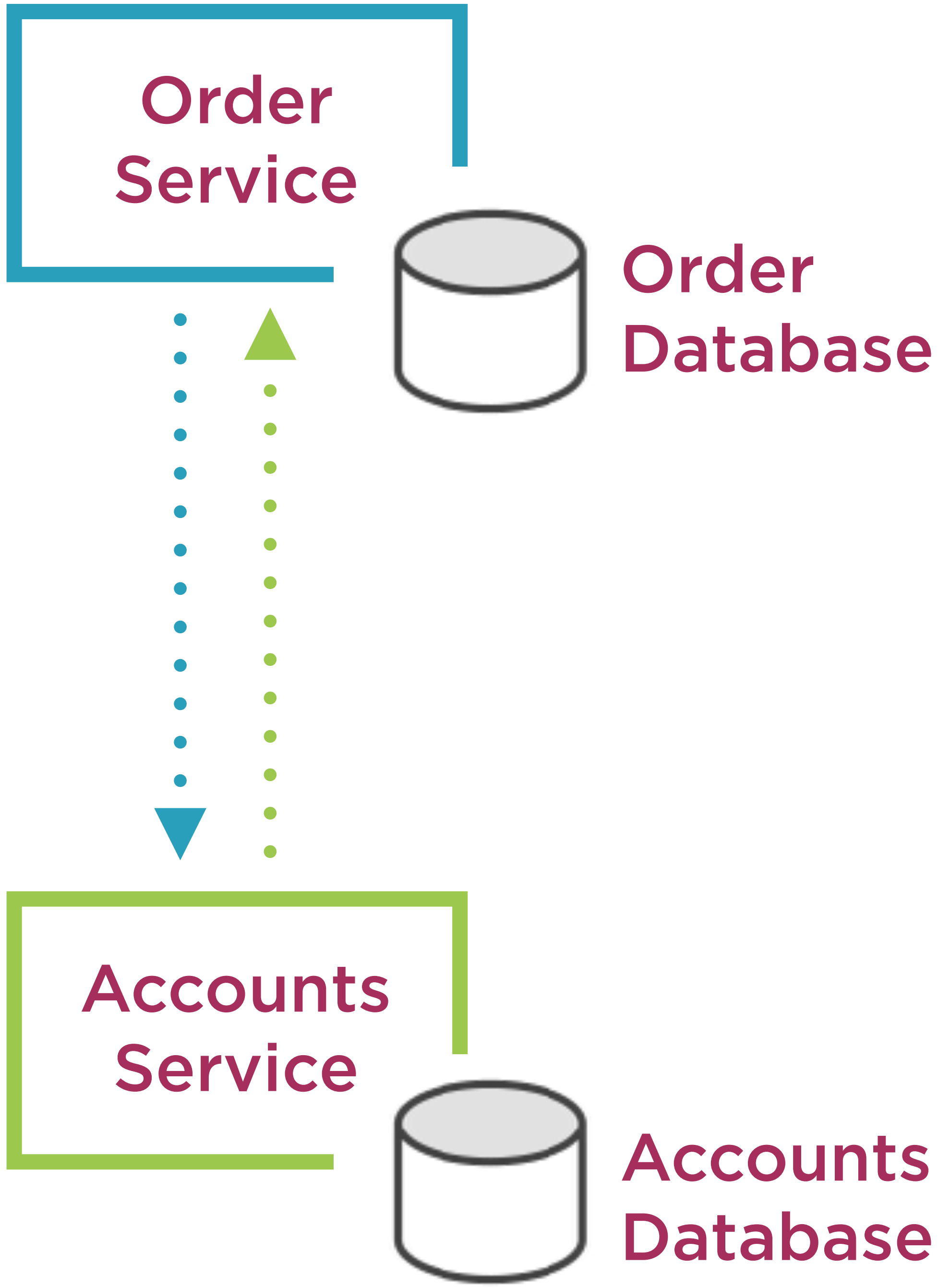
- Could receive same event notification
- Could be done at database tech level

Event Sourcing and CQRS



Greenfield Database Approach

Greenfield Database Approach



Function first design

- Bounded context and code first tools
- Avoiding modeling data first

Avoid anaemic CRUD microservices

- Functions and contracts over internal models

Avoid internal data and database sharing

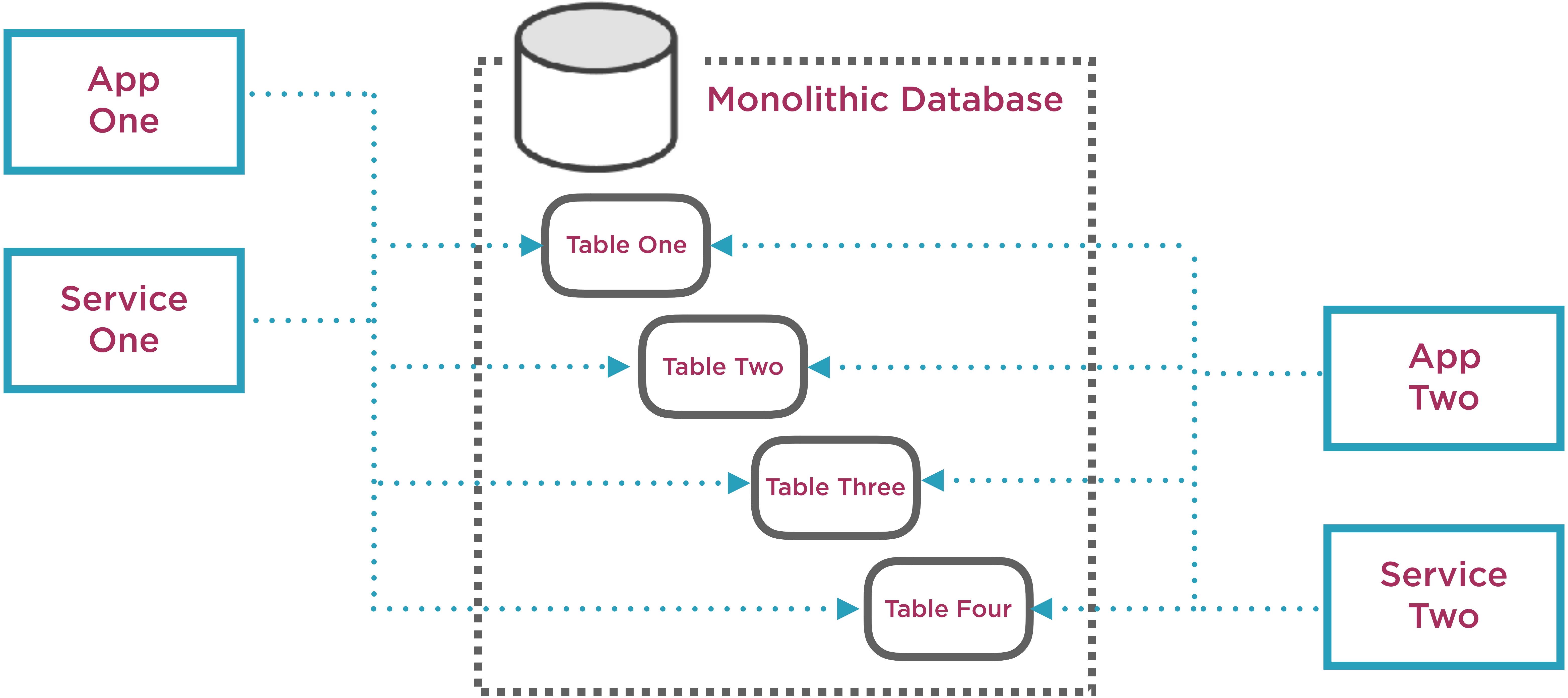
- Event notifications to share data
- Event sourcing

CQRS to further split service and its database

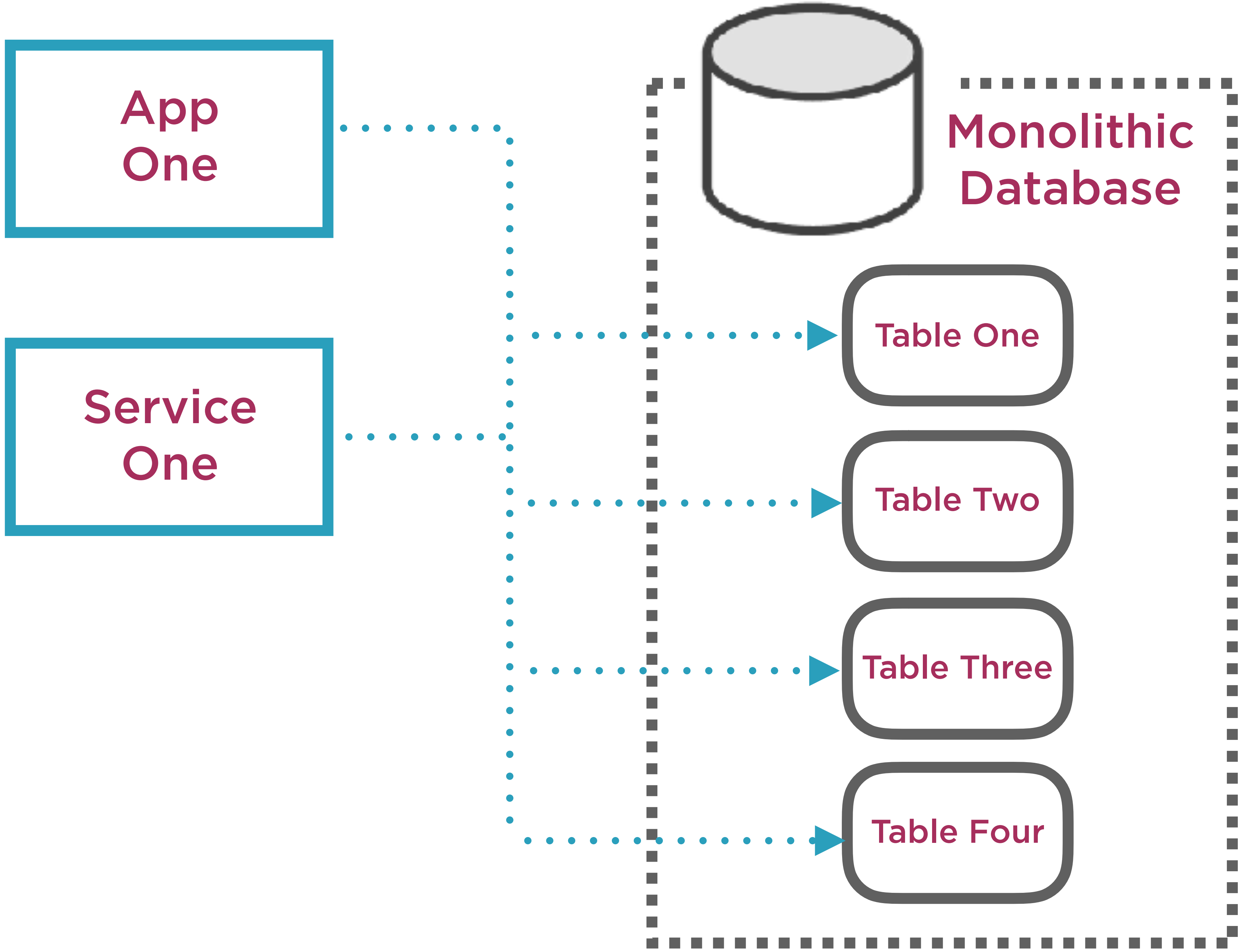
- Microservice for events (Command)
- Microservice for data (Query)

Brownfield Migration Strategy

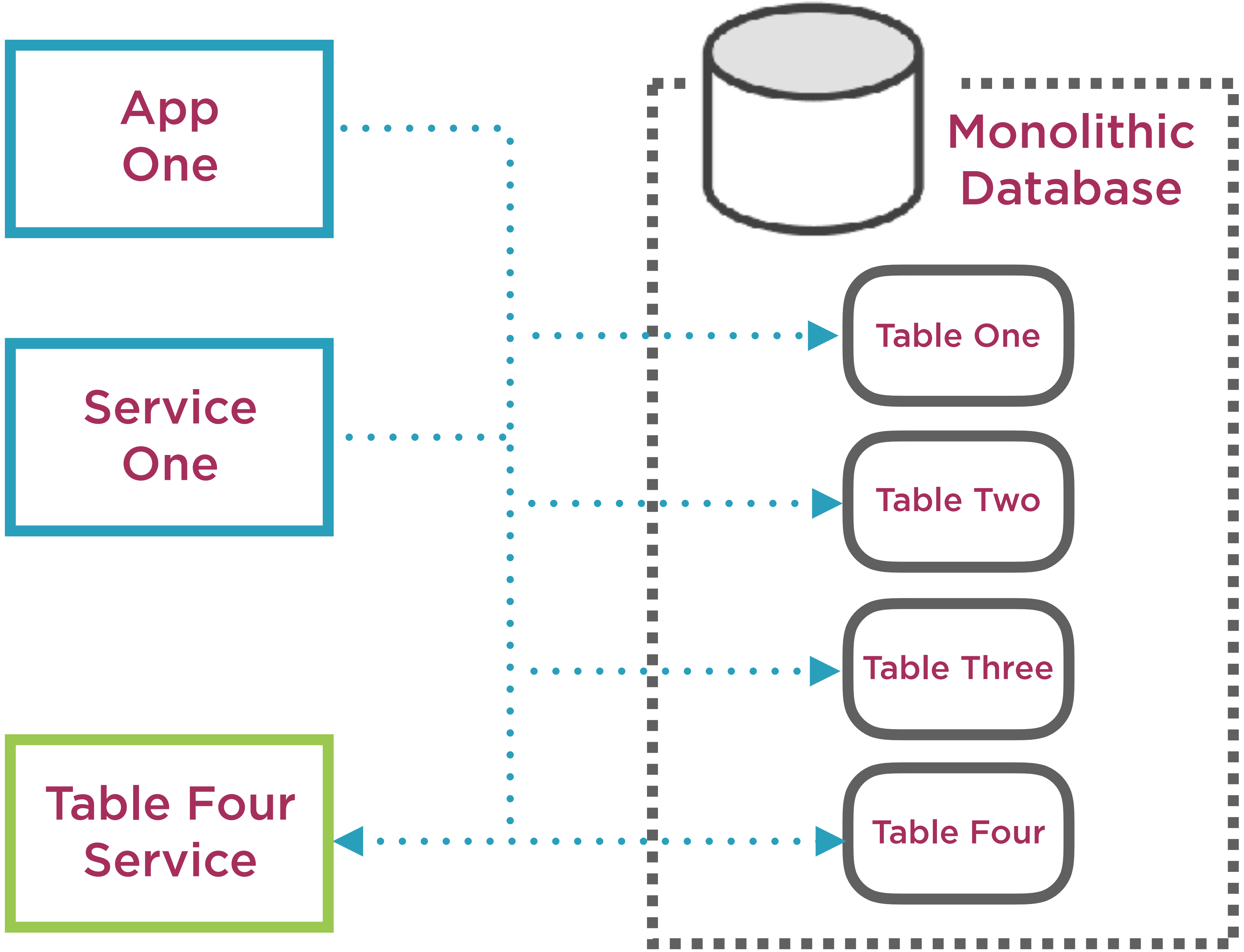
Brownfield Scenario



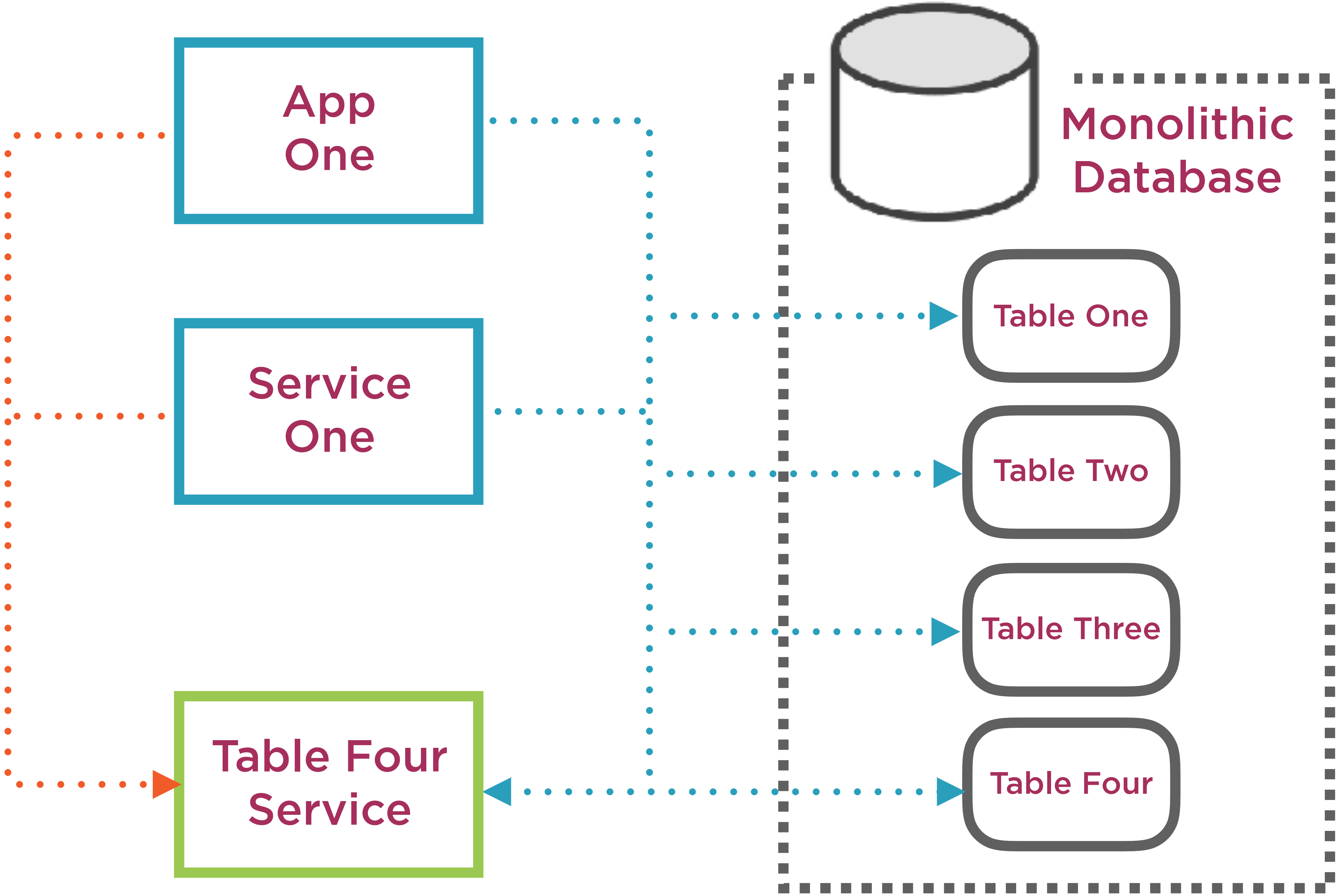
Brownfield Scenario: Step 1



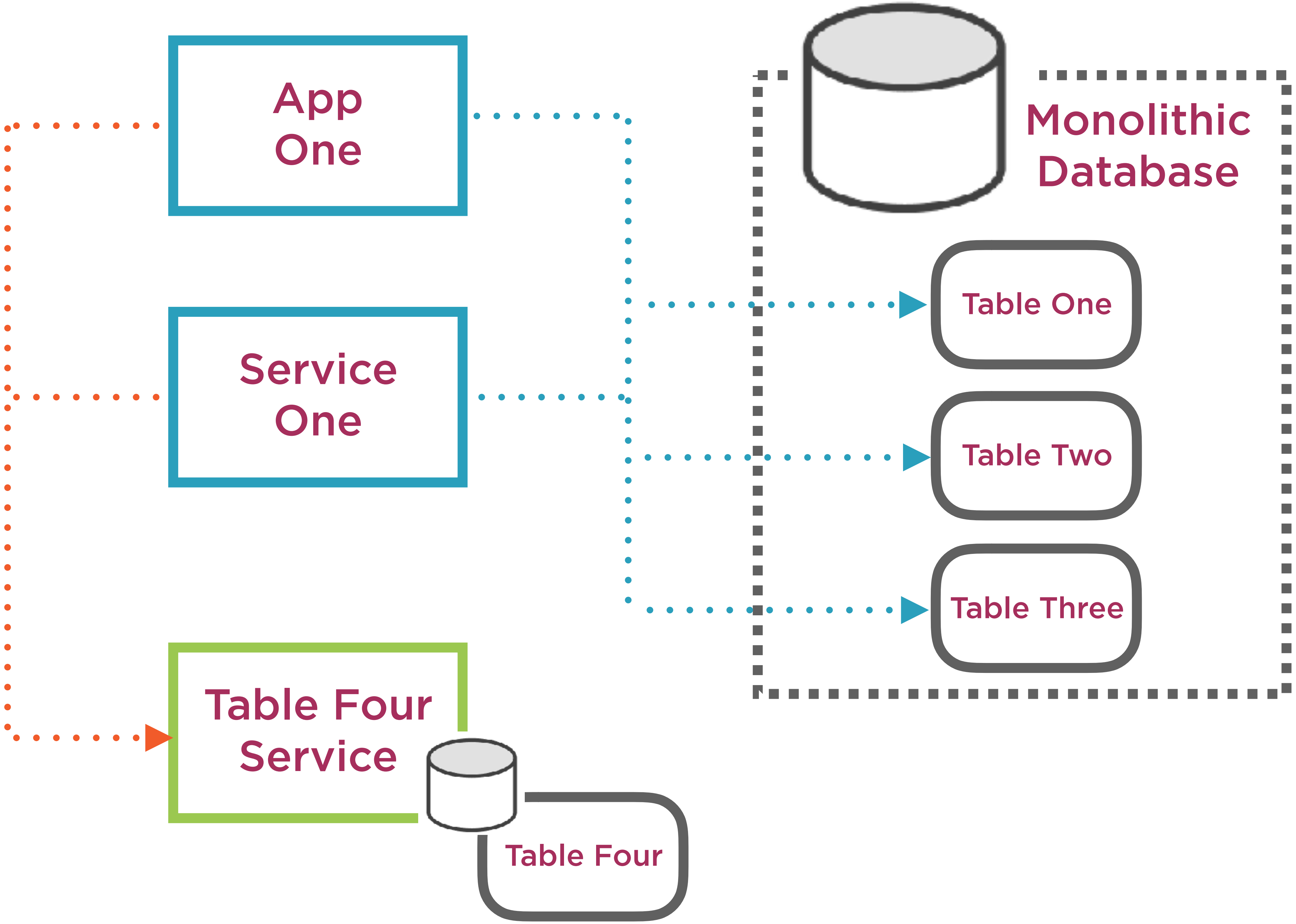
Brownfield Scenario: Step 2



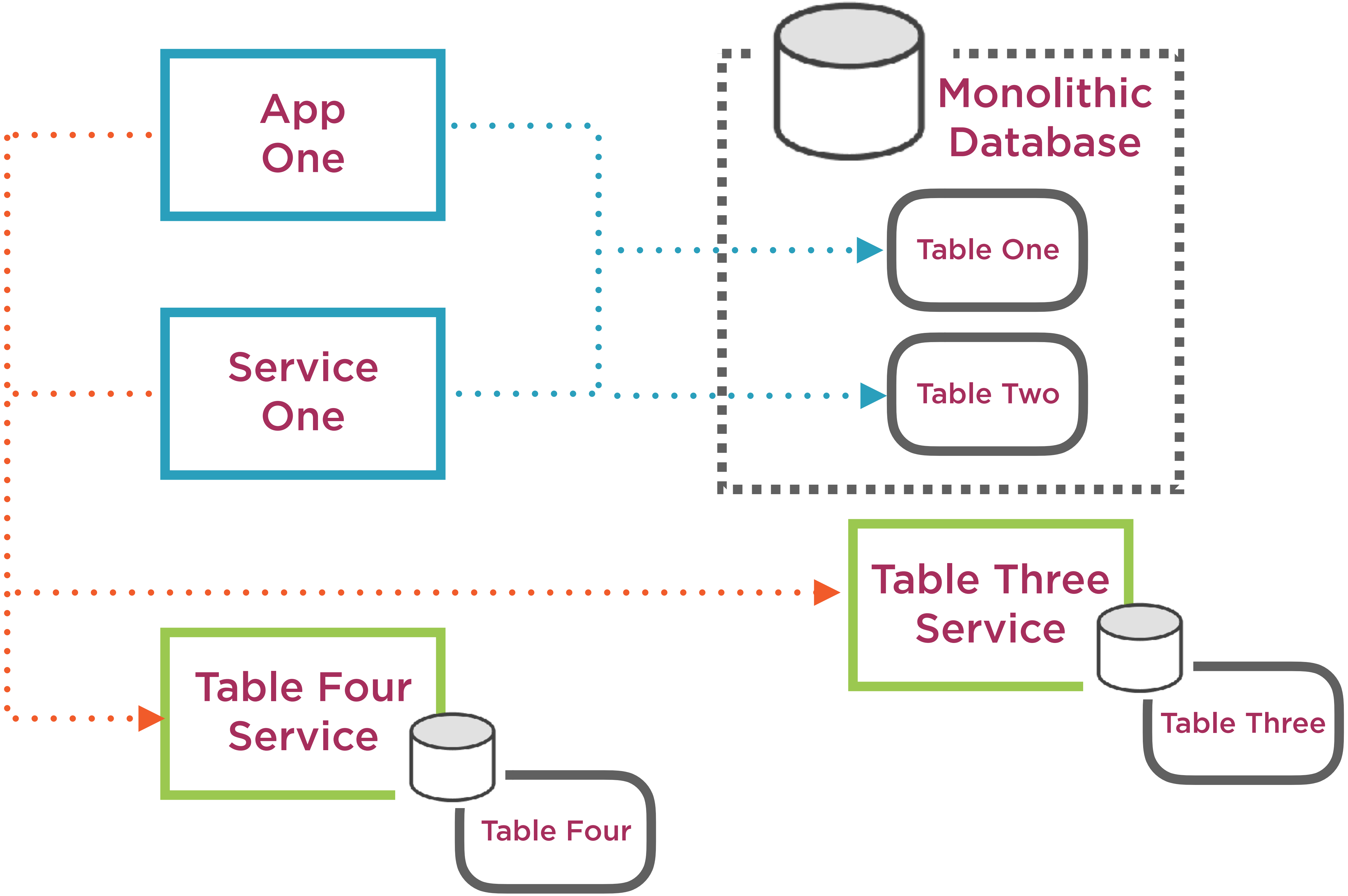
Brownfield Scenario: Step 3



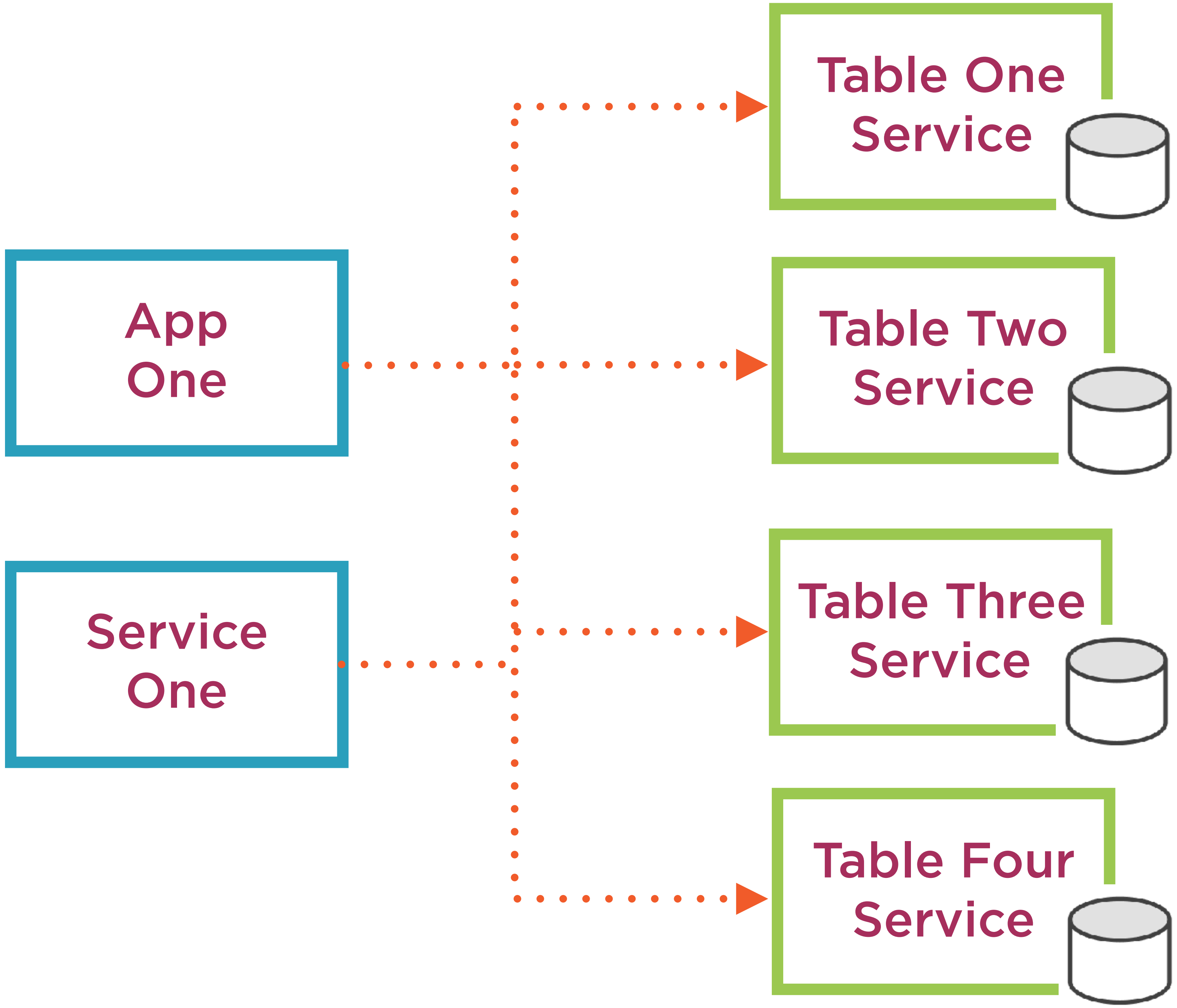
Brownfield Scenario: Step 4



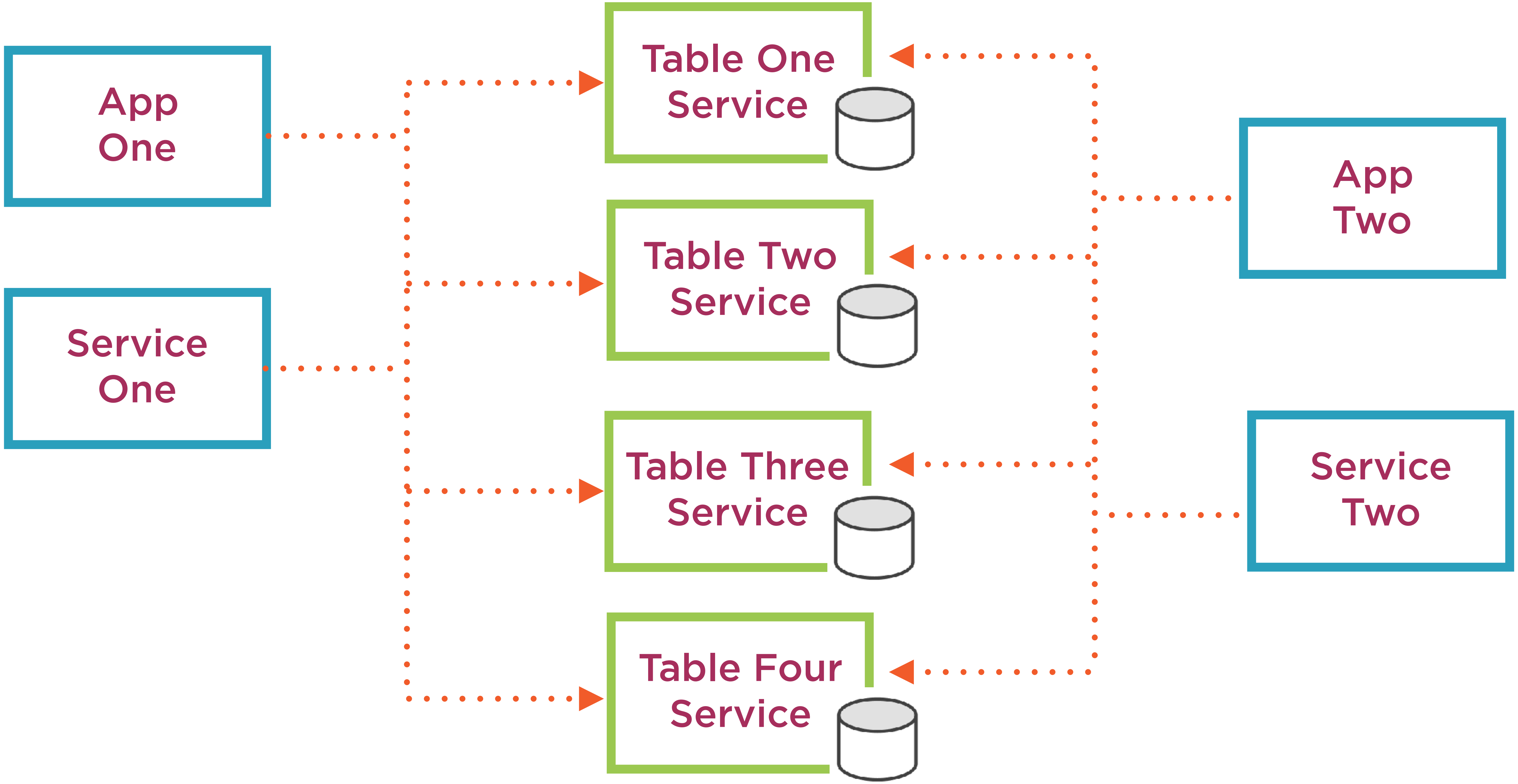
Brownfield Scenario: Step 5



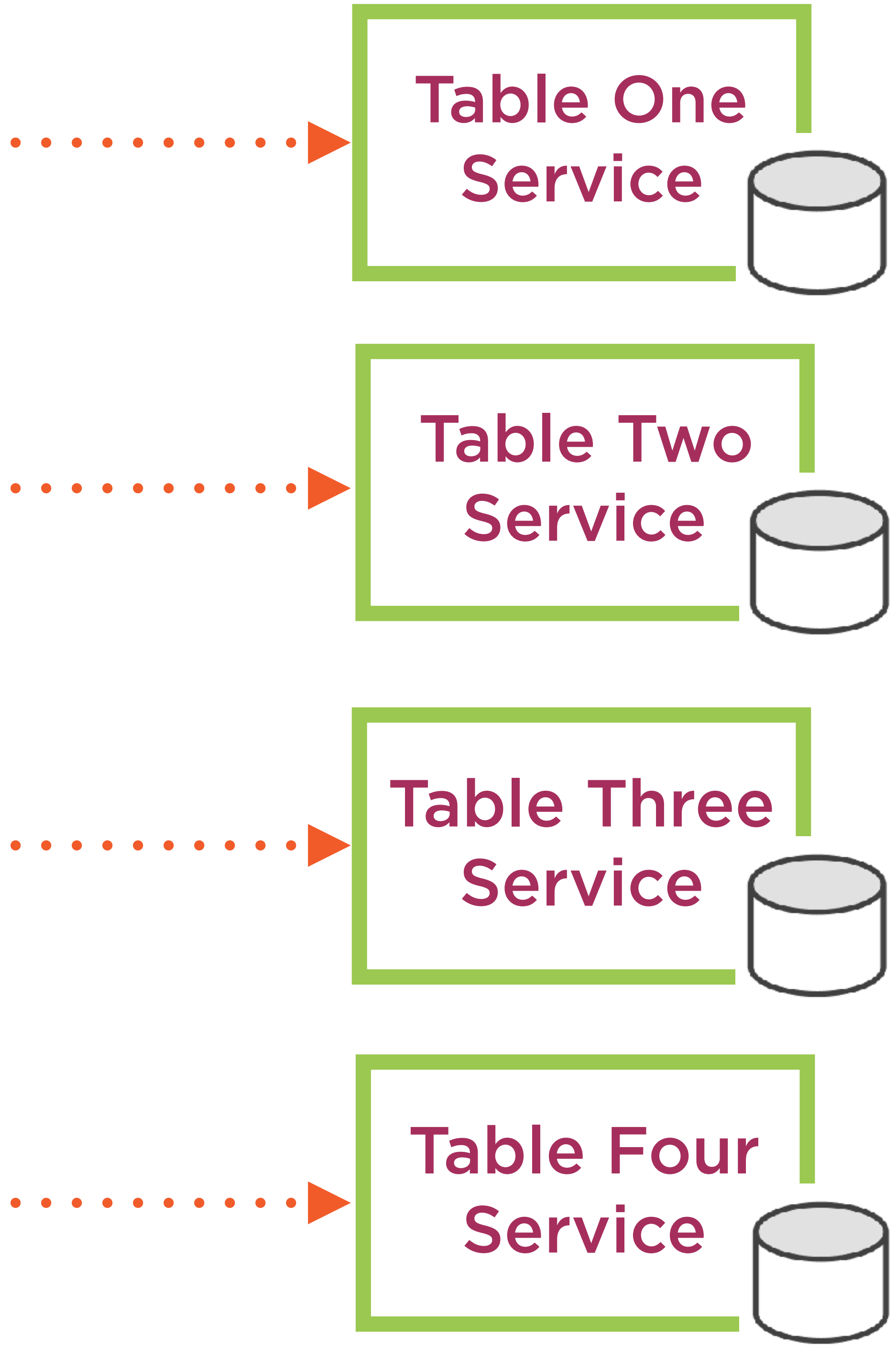
Brownfield Scenario



Brownfield Scenario



Brownfield Migration Strategy Challenges



Very difficult to retrofit greenfield patterns

- Event Sourcing and CQRS

Slow, strategic planned approach

Refactoring existing applications and services

- Not a simple refactor (table access to service calls)
- New service contracts
- Use patterns to shield table migration
  - Proxy class to fetch data from service
- Replacing table joins
  - Multiple service calls
  - Local cache for events of interest to a join

Static data

- Move to shared libraries

Summary

Introduction

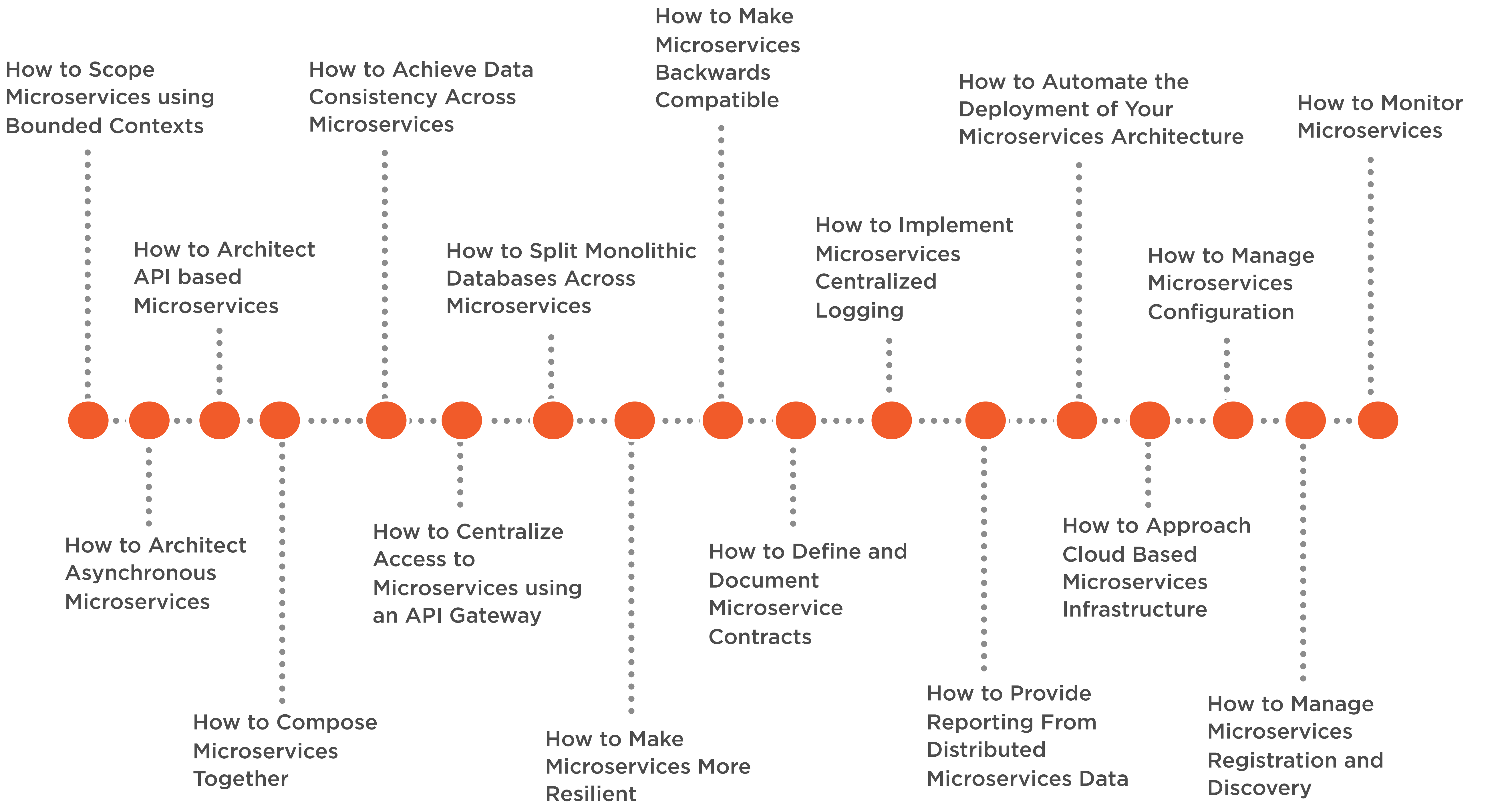
Approach to Database Design

Patterns for Database Design

Greenfield Database Approach

Brownfield Migration Strategy

Microservices Architectural Design Patterns Playbook



How to Scope Microservices using Bounded Contexts

How to Architect Asynchronous Microservices

How to Architect API based Microservices

How to Compose Microservices Together

How to Achieve Data Consistency Across Microservices

How to Centralize Access to Microservices using an API Gateway

How to Split Monolithic Databases Across Microservices

How to Make Microservices More Resilient

How to Make Microservices Backwards Compatible

How to Define and Document Microservice Contracts

How to Implement Microservices Centralized Logging

How to Provide Reporting From Distributed Microservices Data

How to Automate the Deployment of Your Microservices Architecture

How to Approach Cloud Based Microservices Infrastructure

How to Manage Microservices Configuration

How to Manage Microservices Registration and Discovery

How to Monitor Microservices