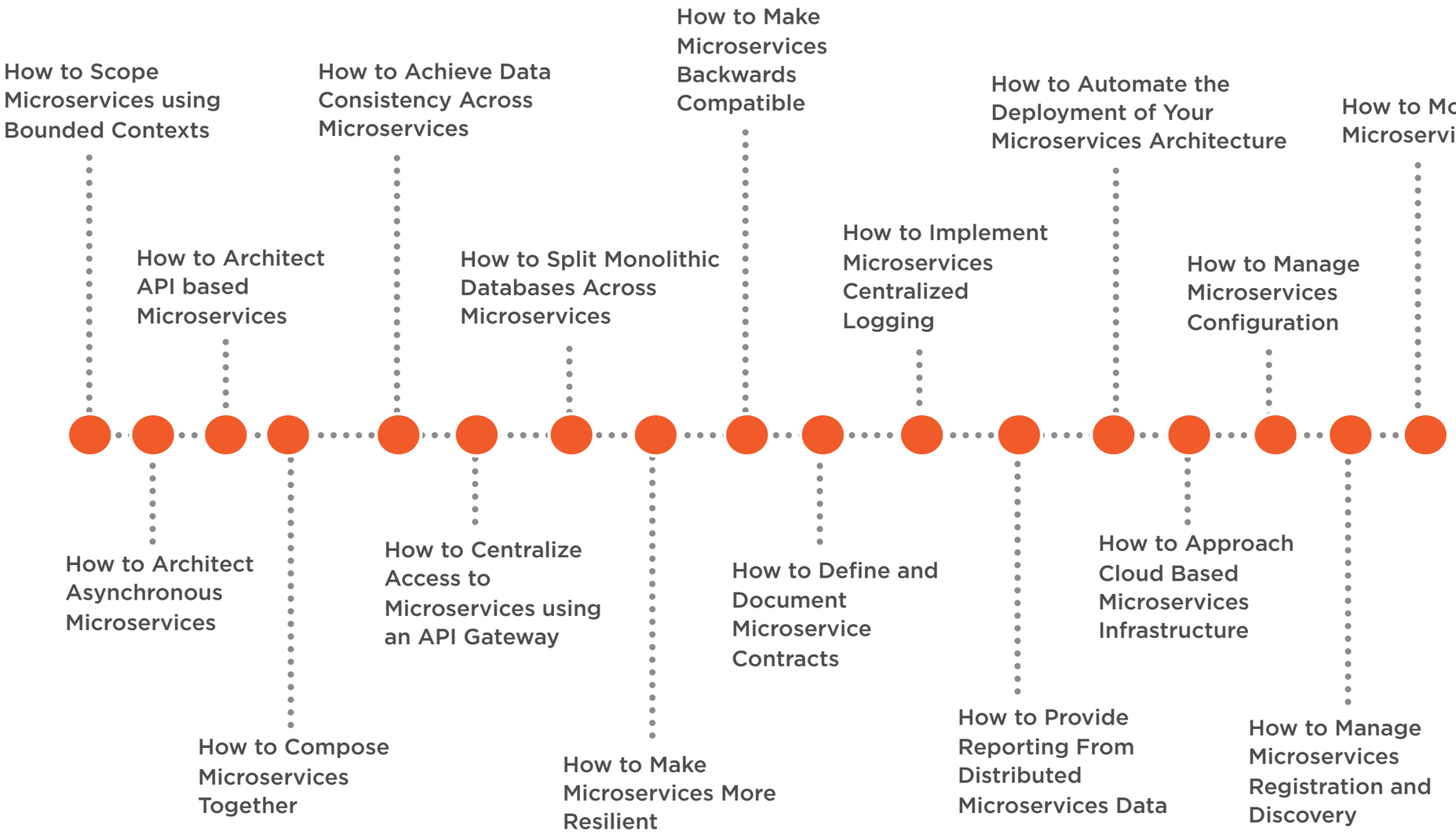
How to Make Microservices More Resilient

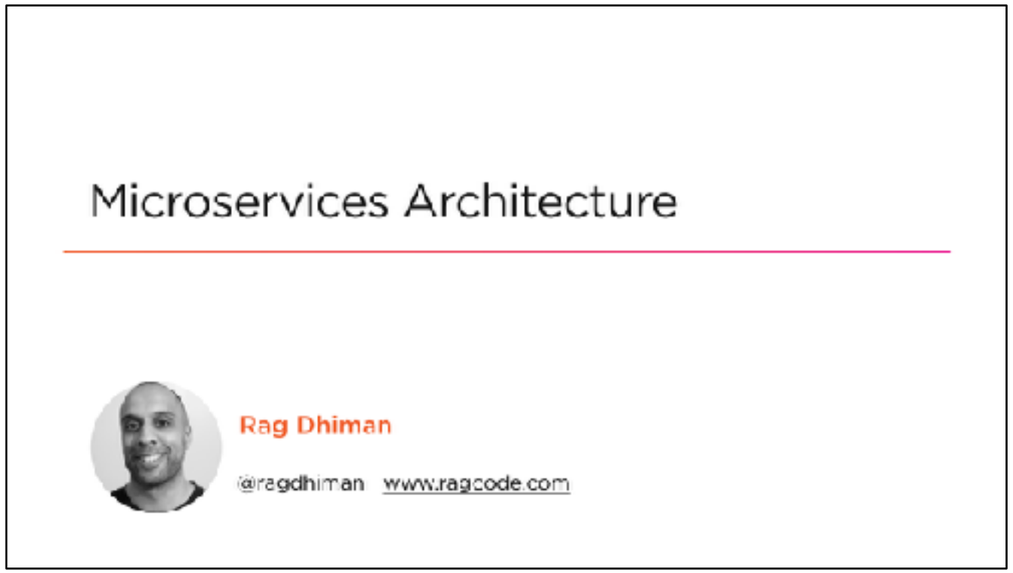
Rag Dhiman

@ragdhiman www.ragcode.com

Microservices Architectural Design Patterns Playbook



Microservices Architectural Design Patterns Playbook



Overview

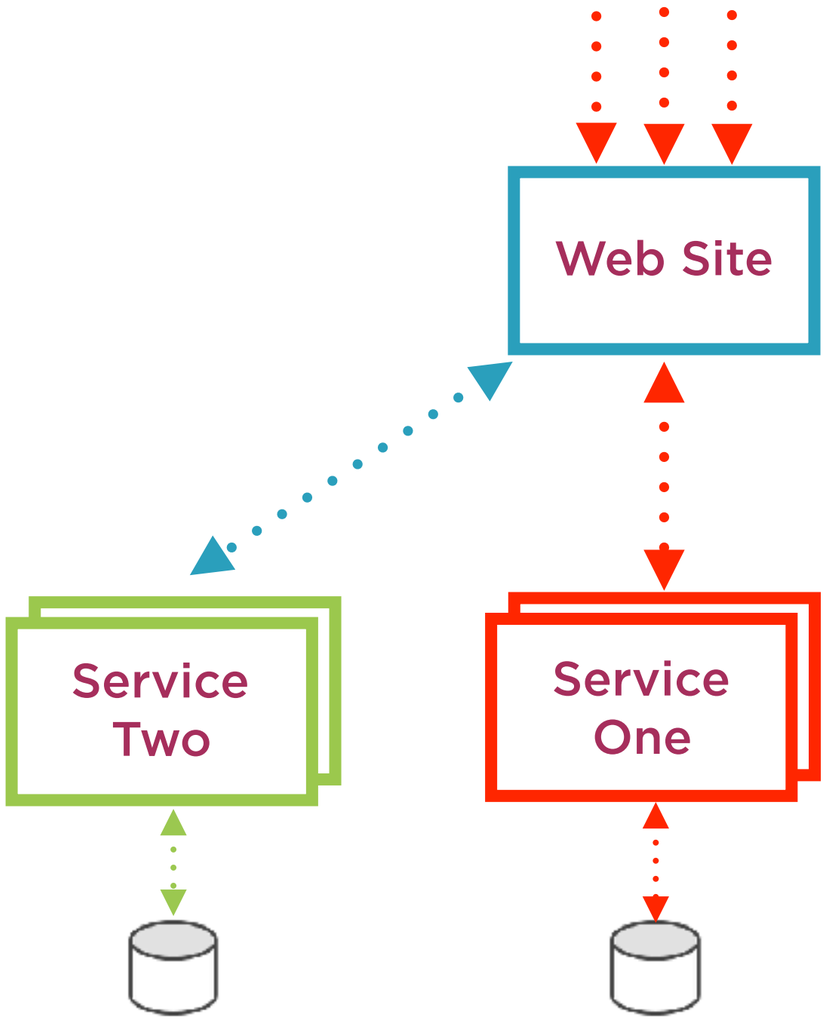
Introduction

Patterns and Approach

Design Patterns

Approach to Design

Introduction



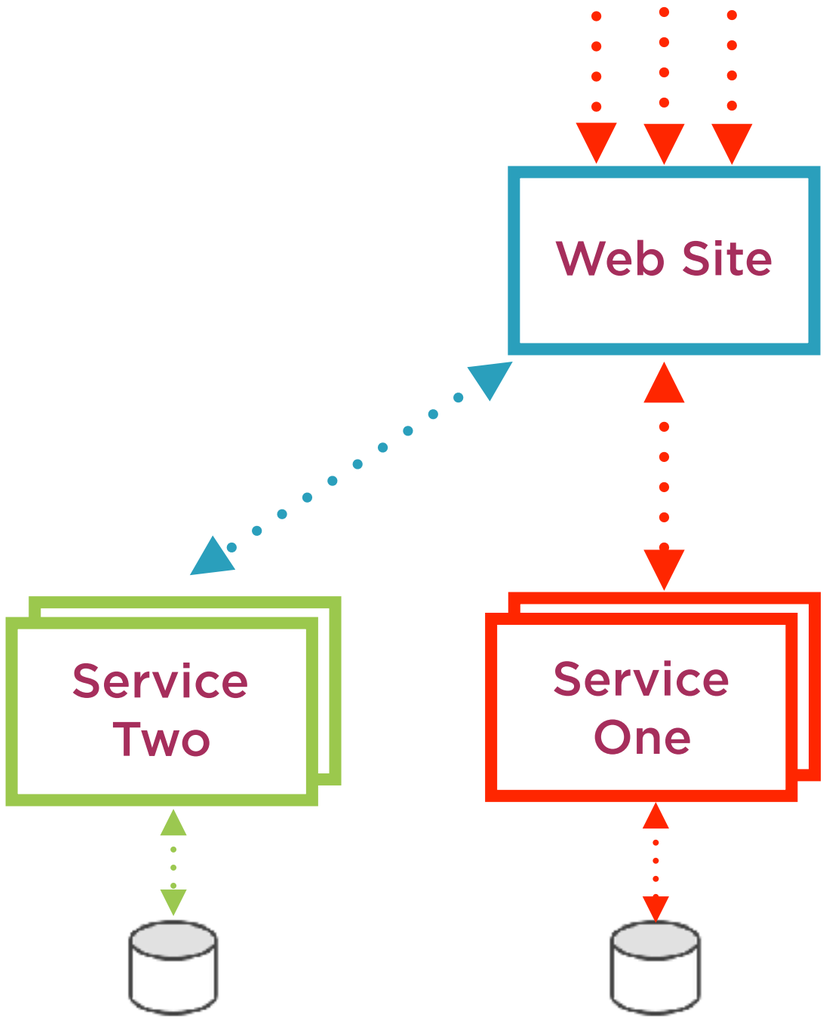
Need for resiliency

- Distributed architecture
- Communication over a network
- Fault tolerance required
- Avoid cascading failures
- Avoid exhausting resource pools

Types of failure

- Hardware
- Infrastructure
- Communication layer
- Dependencies
- Microservices

Patterns and Approach



Design patterns for resiliency

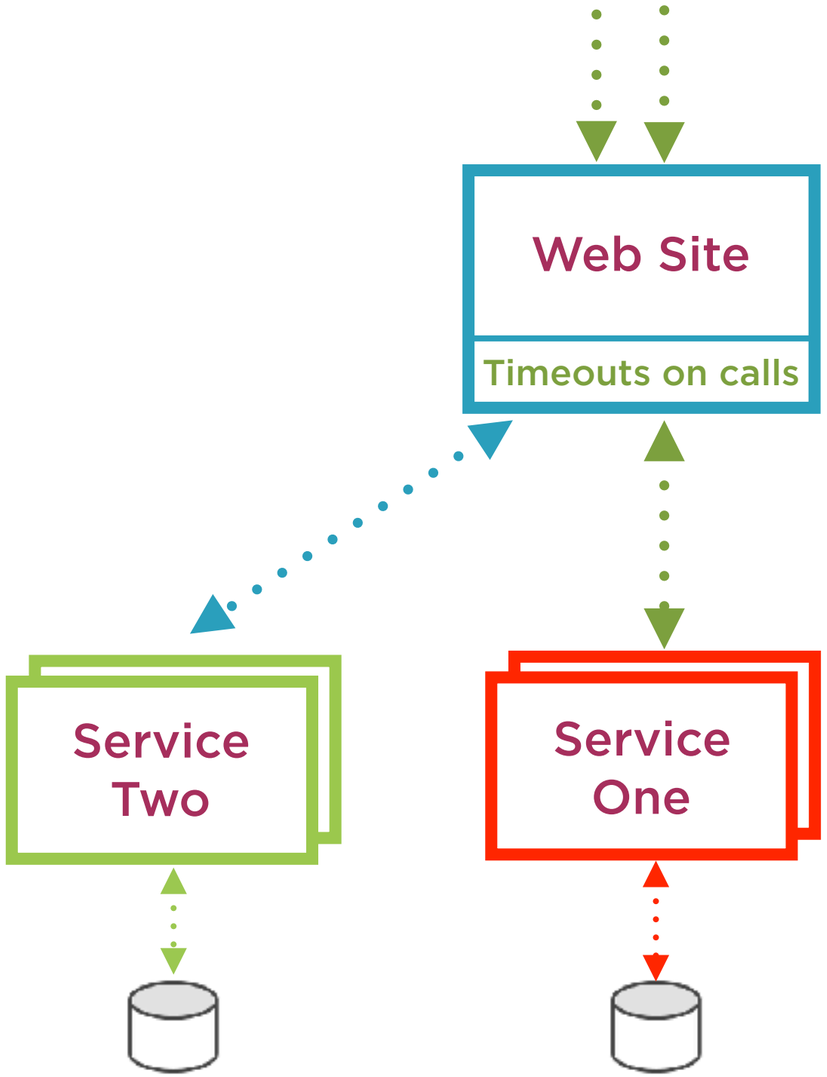
- Timeouts
- Circuit Breaker
- Retry
- Bulkhead

Approach to software development

- Design for known failures
- Embrace failures
- Fail fast
- Degrade or default functionality

Timeouts Design Pattern

Timeouts



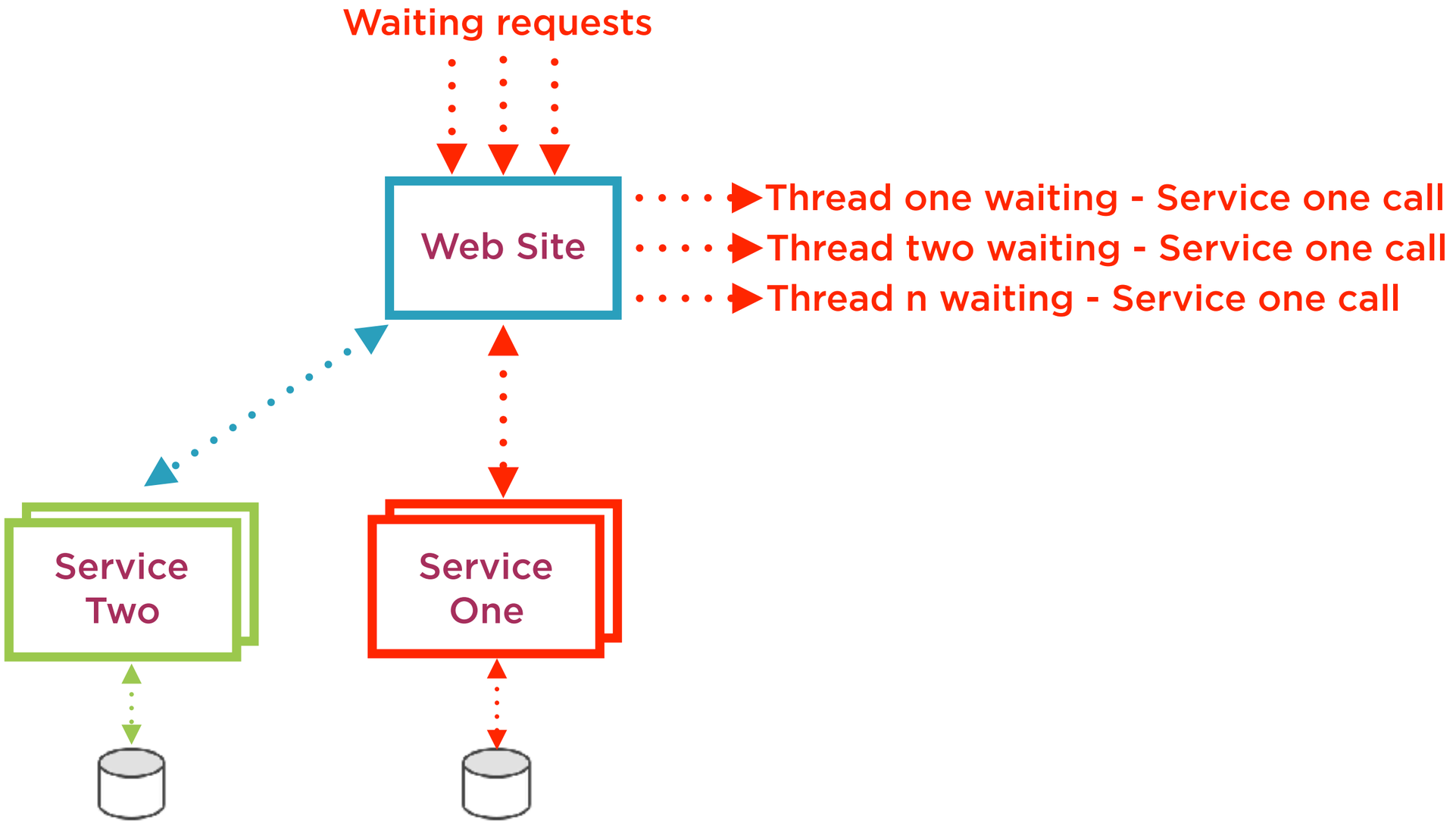
Client calling code

- Explicitly configure timeout value
- Do not rely on default timeout values
- Configurable value
- Can be used with retries

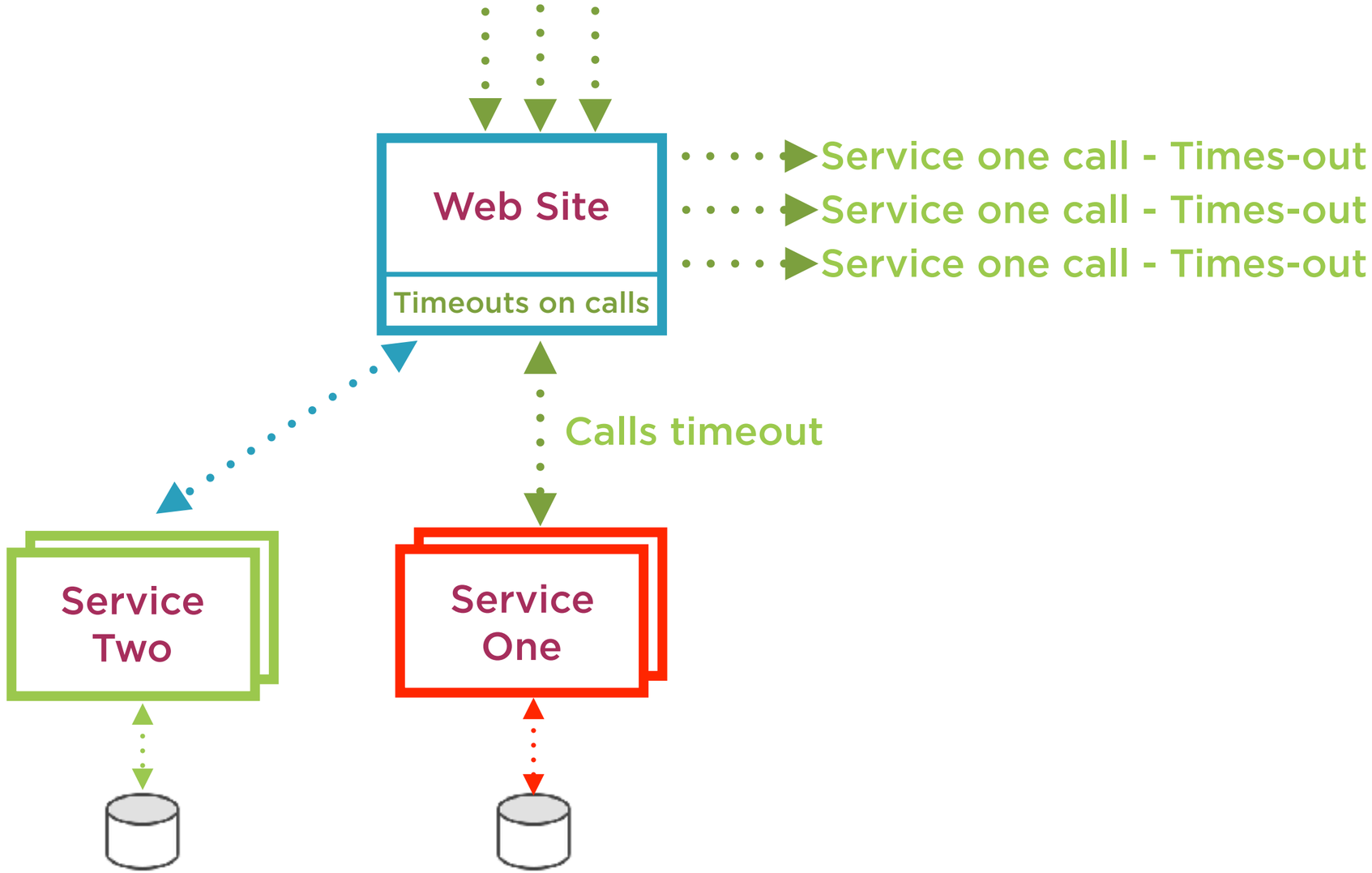
Advantage of timeouts

- Avoid waiting for a response forever
- Frees up calling thread
- Allows you to handle call failure
- Supported by nearly all client technologies

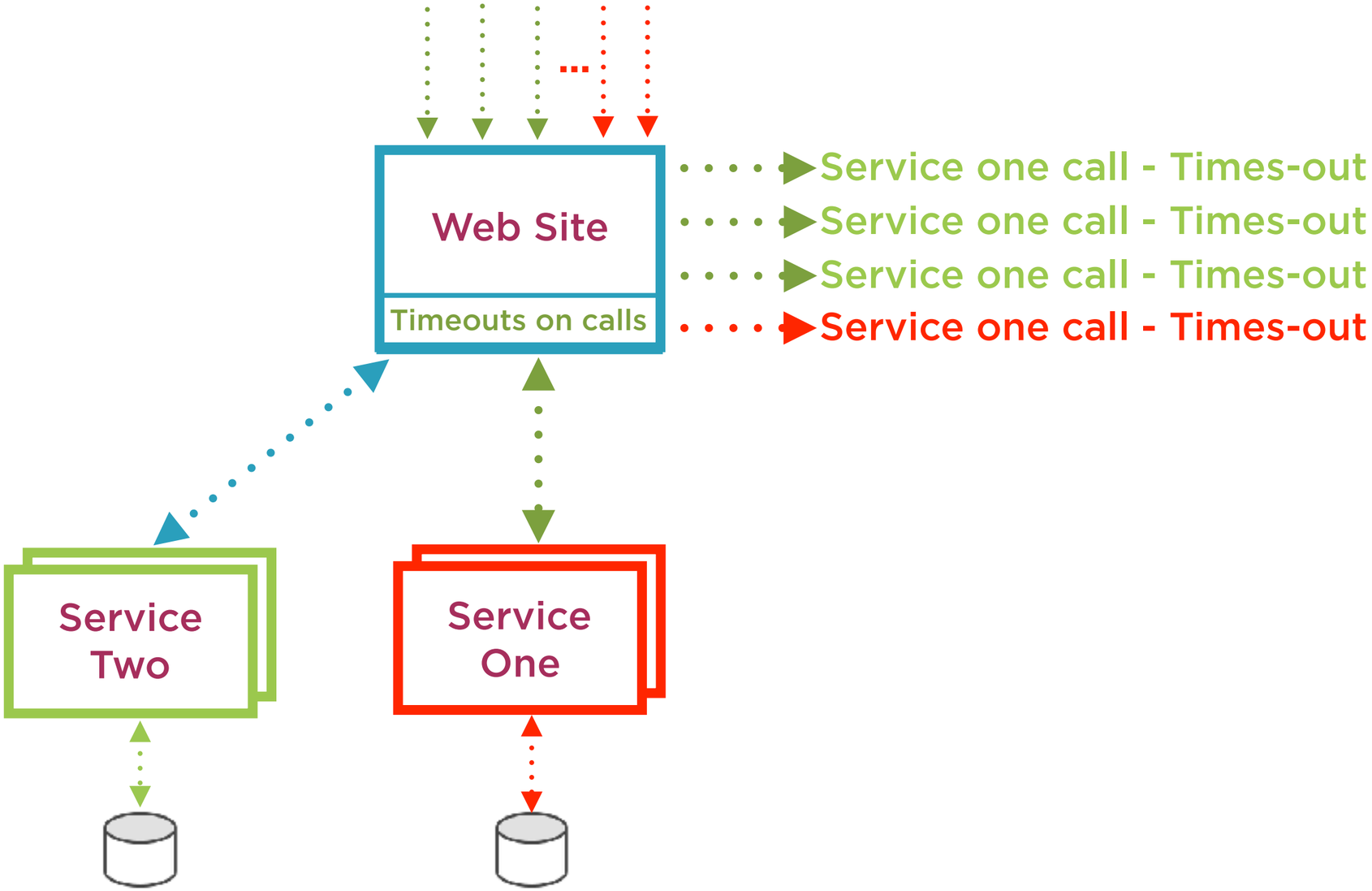
Example of Failure



Example of Timeouts Handling Failure

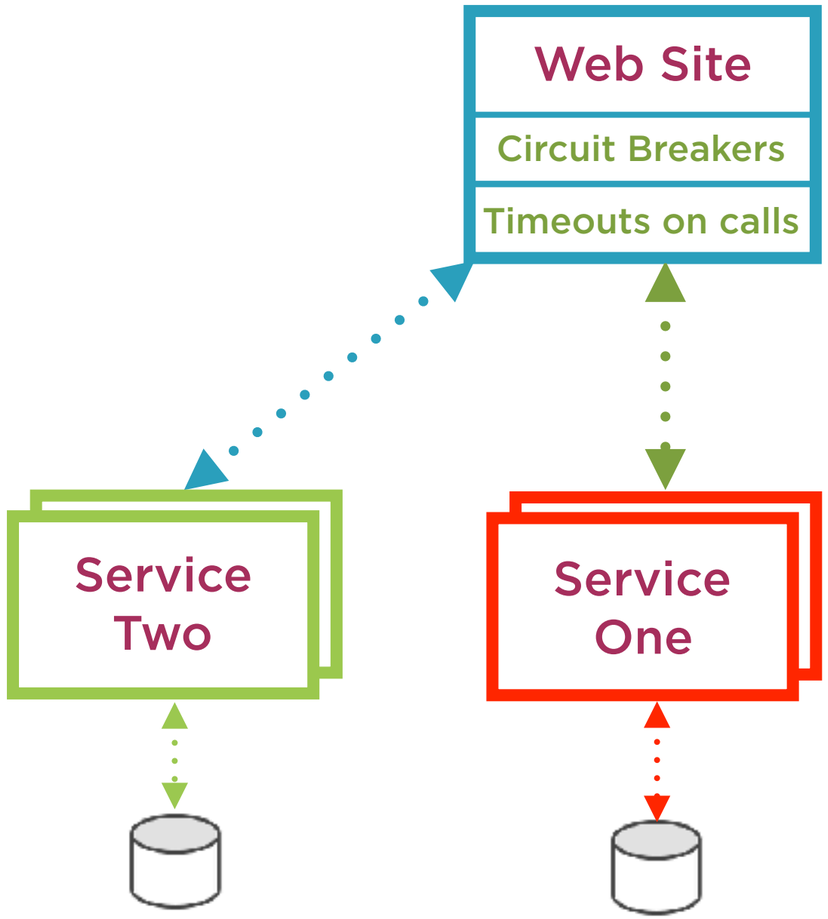


Example of Failure With Timeouts



Circuit Breaker Design Pattern

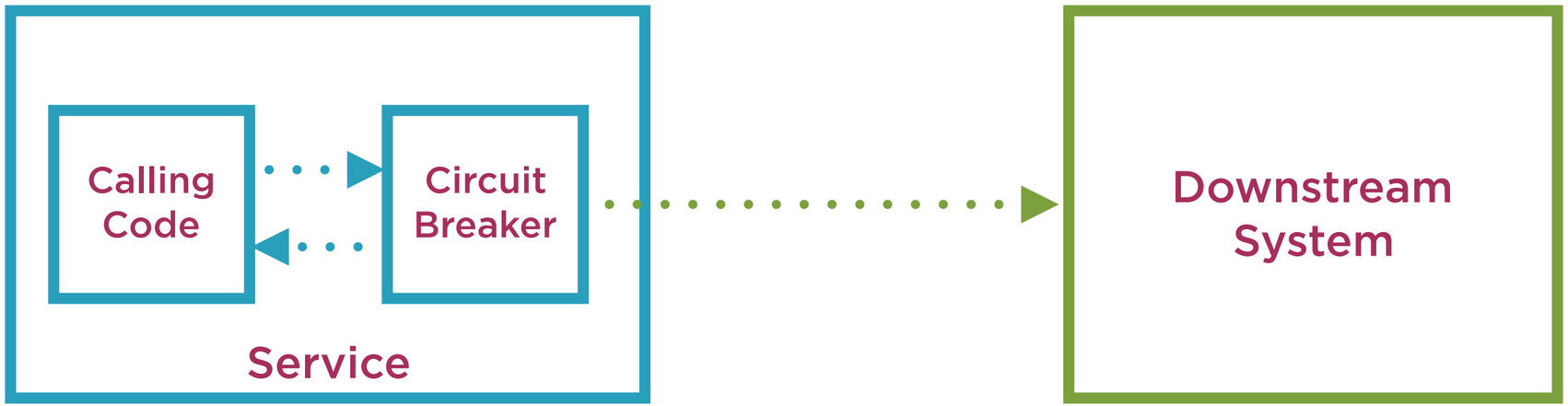
Circuit Breaker Design Pattern



Circuit breaker

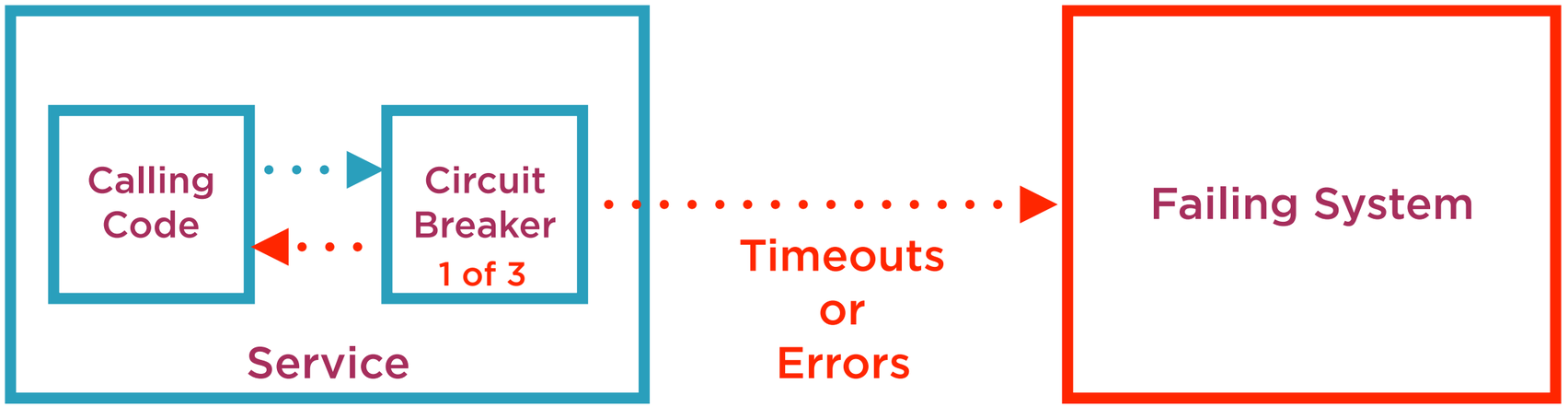
- Object which wraps calls
- Monitors for failures
- Trips after reaching threshold
  - Circuit is open
  - Returns an error without making call
- Different thresholds for different issues
- Reset after a number of tries
- Way to default of degrade functionality
- Avoids cascading failures
- Use for all synchronous calls
- Always report: monitor and log state

Circuit Breaker Design Pattern



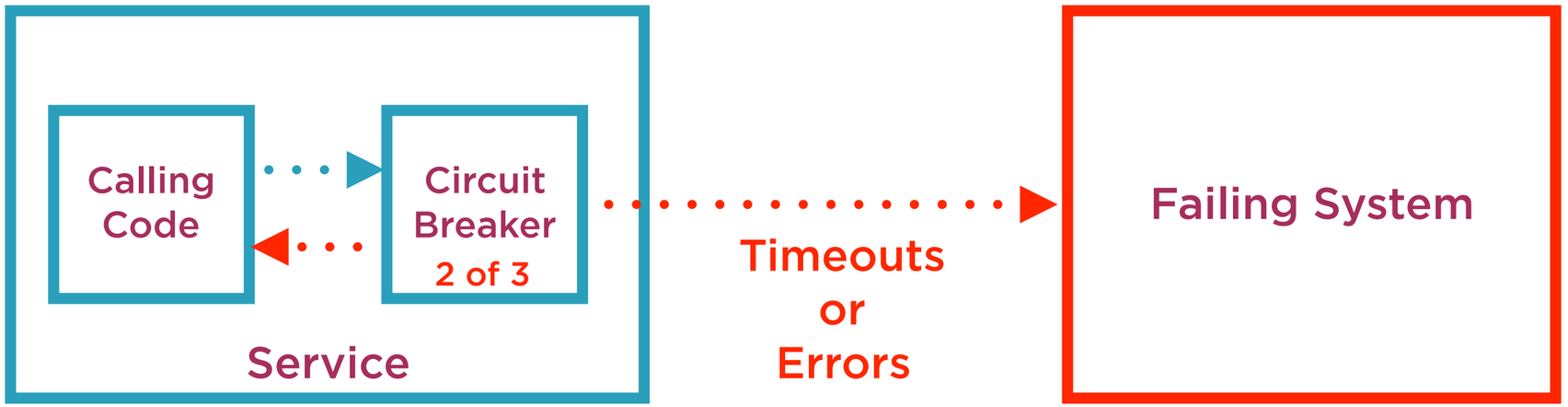
Circuit Closed State

Circuit Breaker Design Pattern



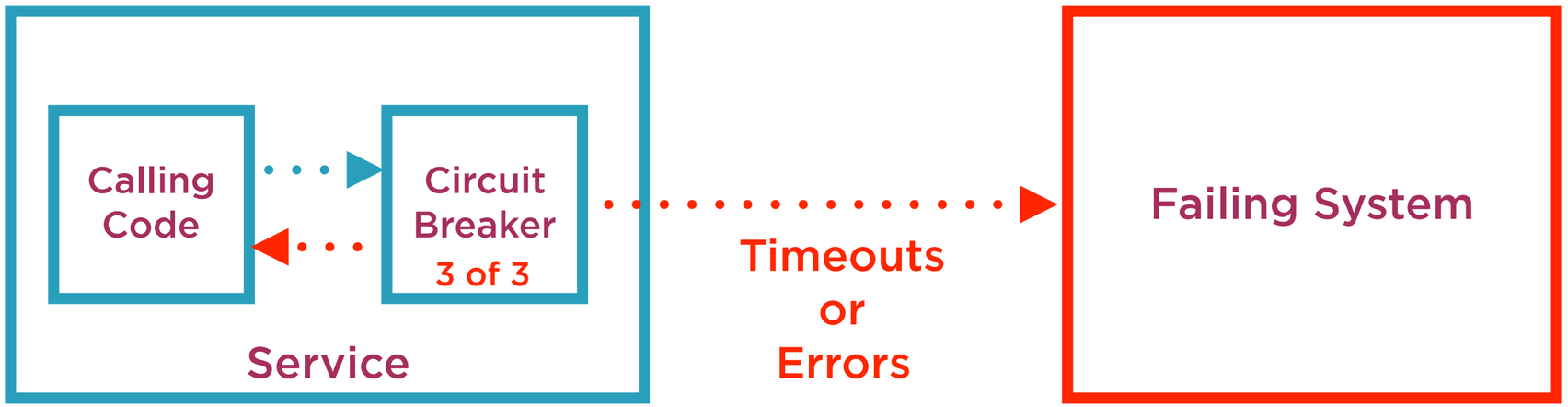
Circuit Closed State

Circuit Breaker Design Pattern



Circuit Closed State

Circuit Breaker Design Pattern



Circuit Closed State

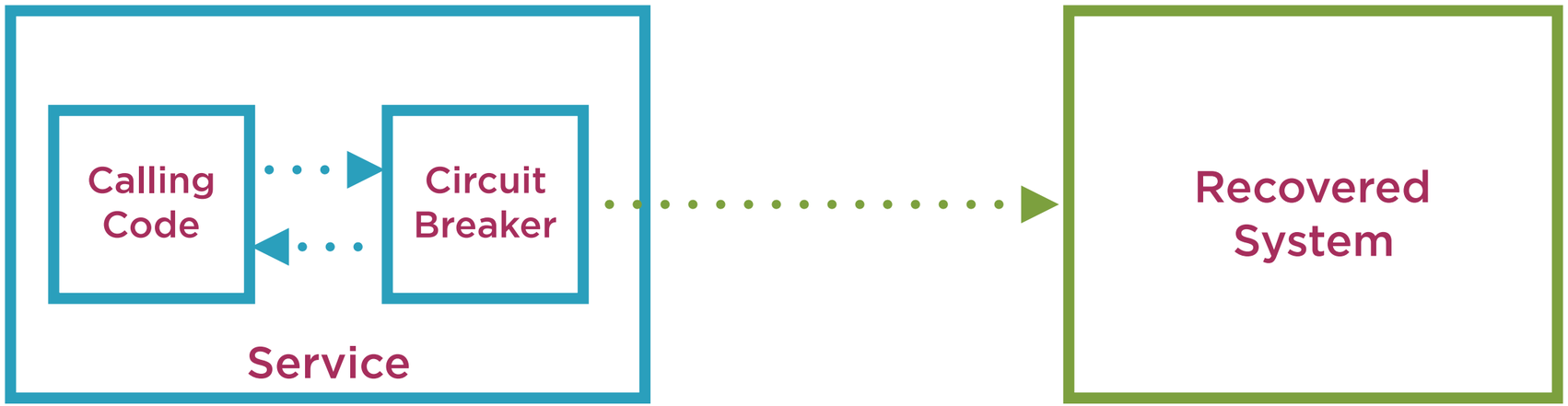
Circuit Breaker Design Pattern

Circuit Open State

Circuit Breaker Design Pattern

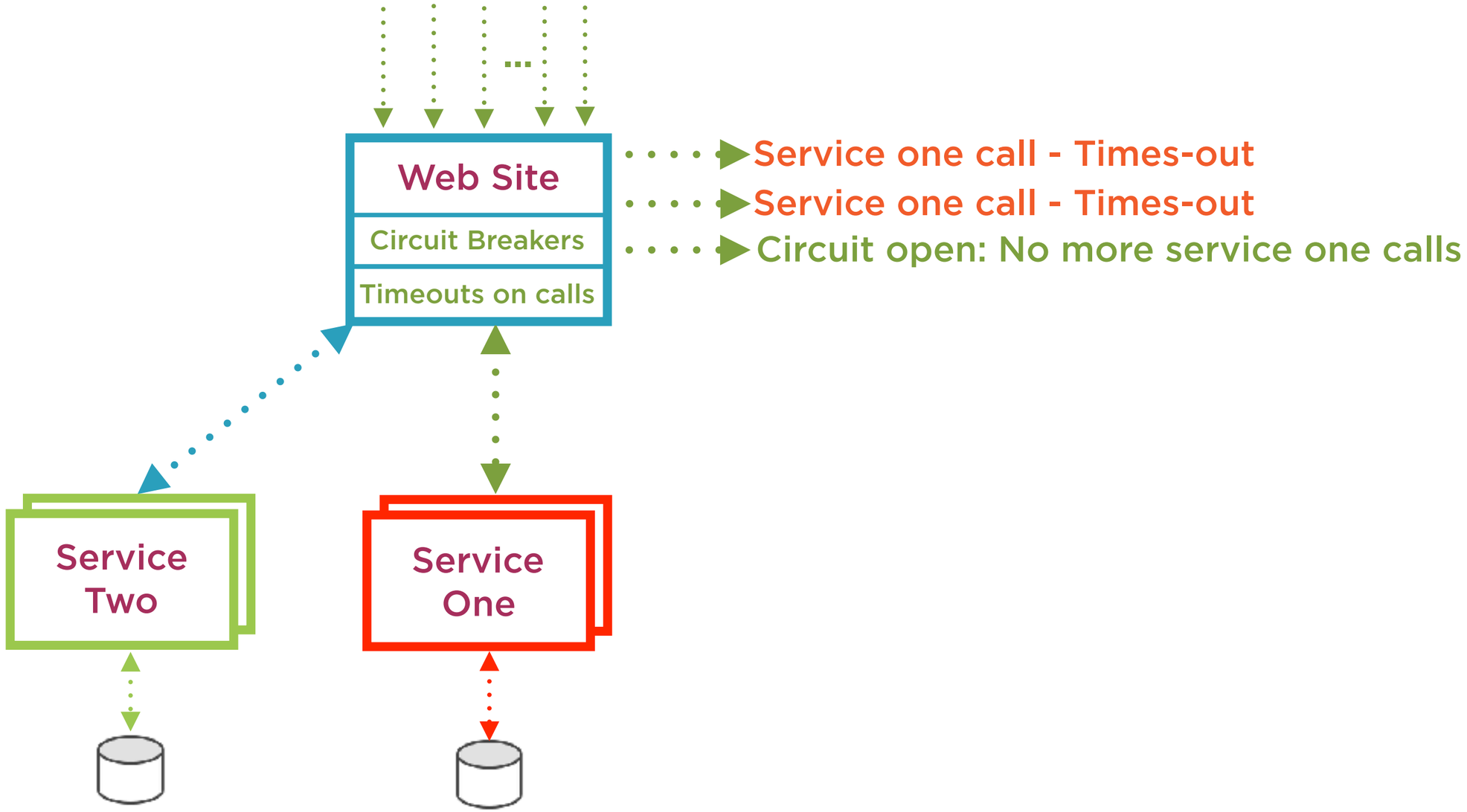
Circuit Half-Open State

Circuit Breaker Design Pattern

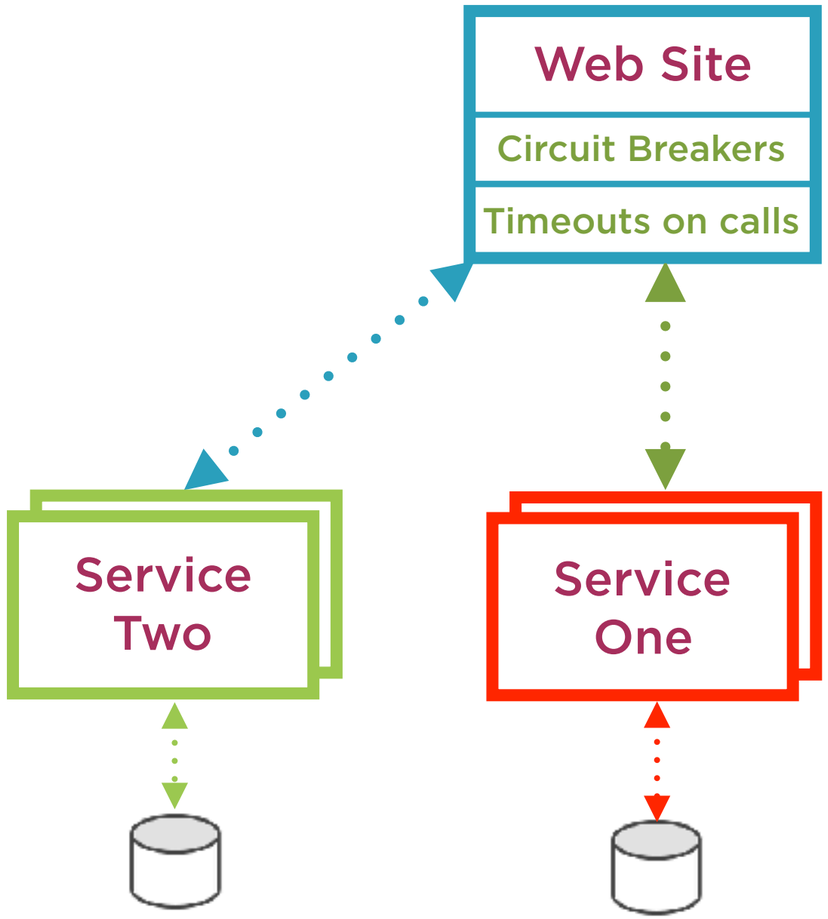


Circuit Closed State

Example of Circuit Breaker Handling Failure



Implementing Circuit Breaker



Develop your own

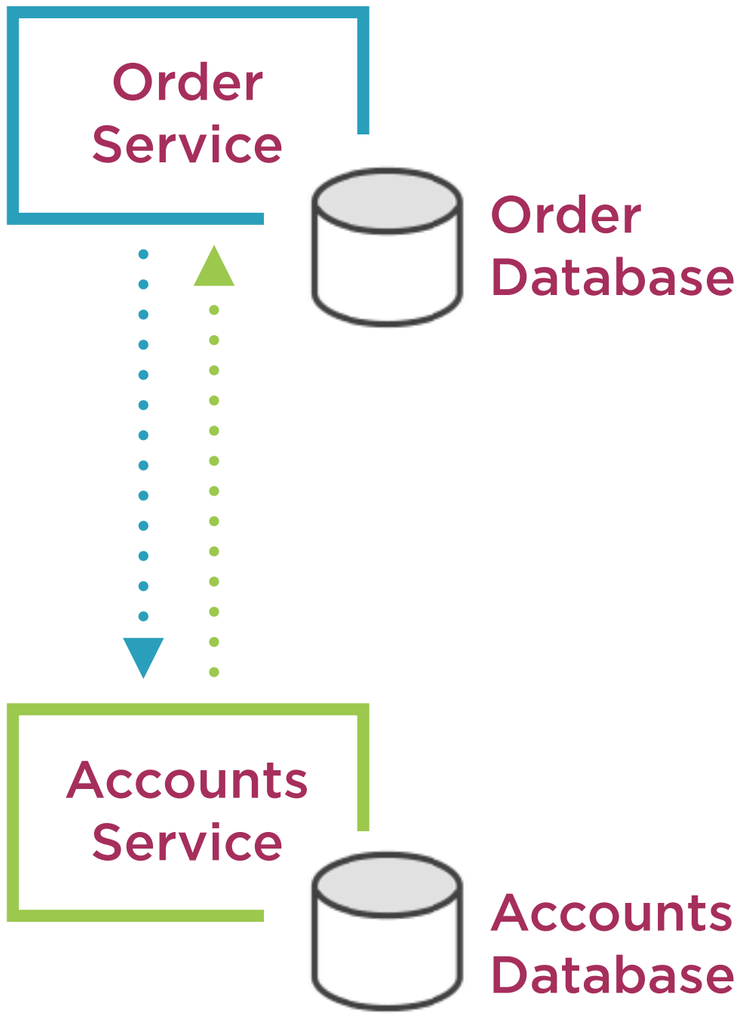
- Many open source examples
- Make thread safe
- Make configurable
  - Failure/timeout threshold
  - Open state duration

Third-party libraries

- Thepollyproject.org
- Hystrix
- and many more...

Retry Design Pattern

Retry Design Pattern



Ideal for transient faults

- Momentary loss of connectivity
- Temporary unavailability of service
- Timeouts when service is busy
- Service throttles accepted requests
- Faults self correct

Strategies

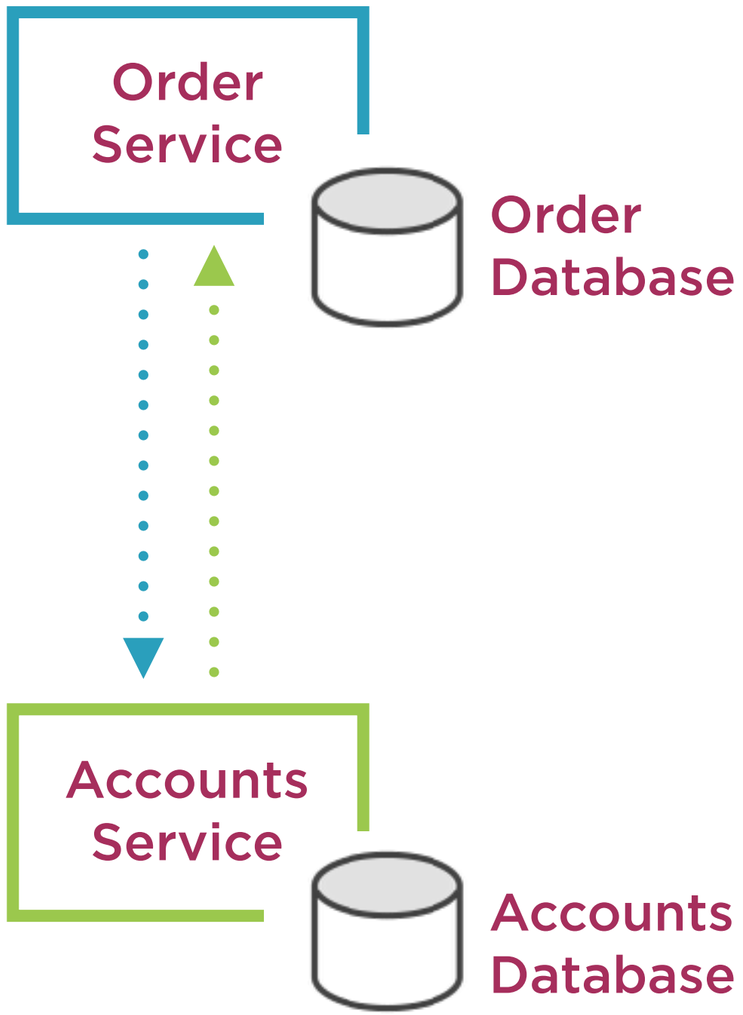
- Retry
- Retry after delay
- Cancel

Log and monitor occurrences

Use in conjunction with circuit breakers

Bulkheads Design Pattern

Bulkheads Design Pattern



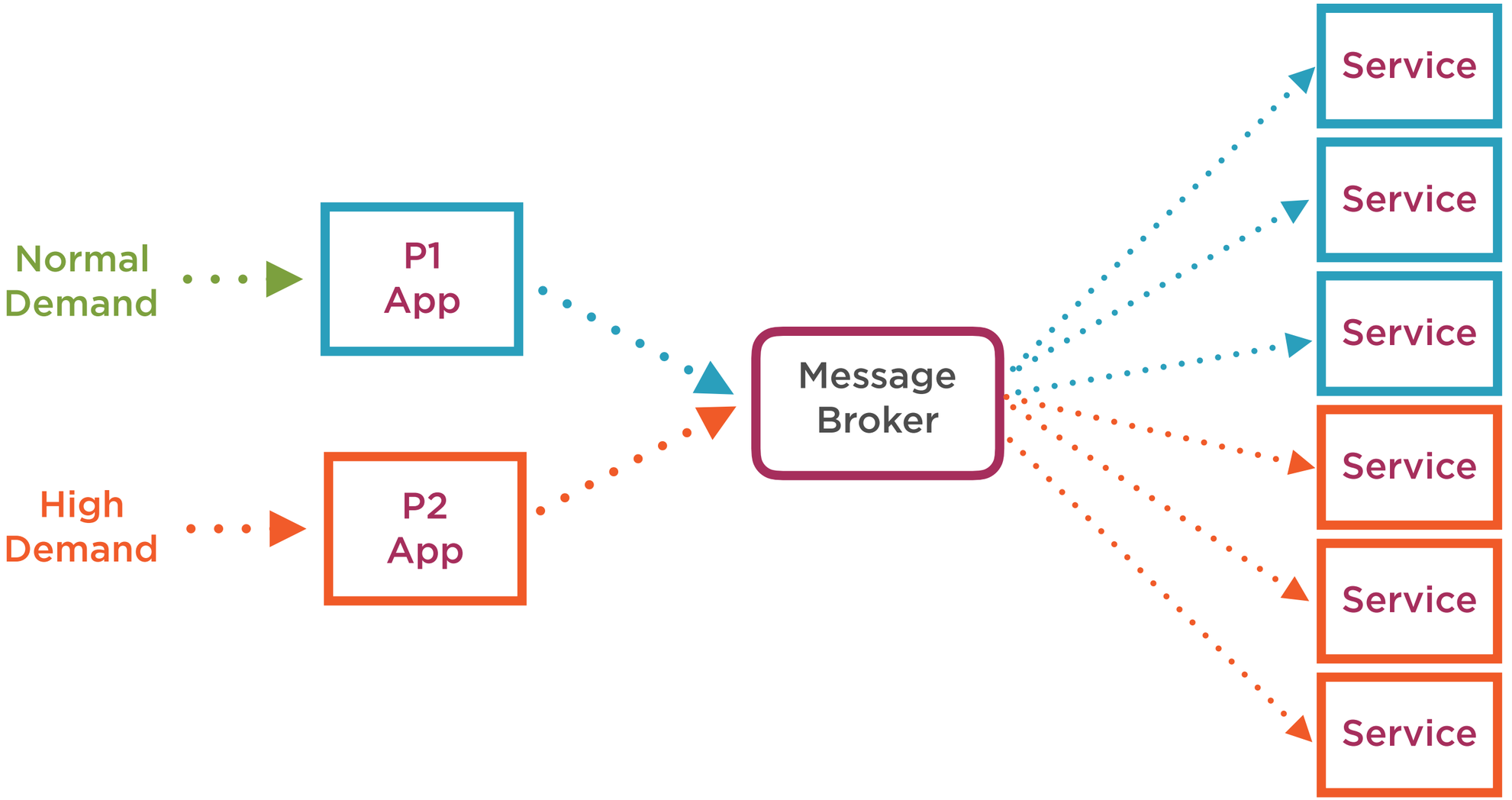
High-level design pattern

- Way of sealing off failure
- Shipping bulkhead analogy
- Avoid cascading failures
- Separate from failed component
- Ensure degraded functionality

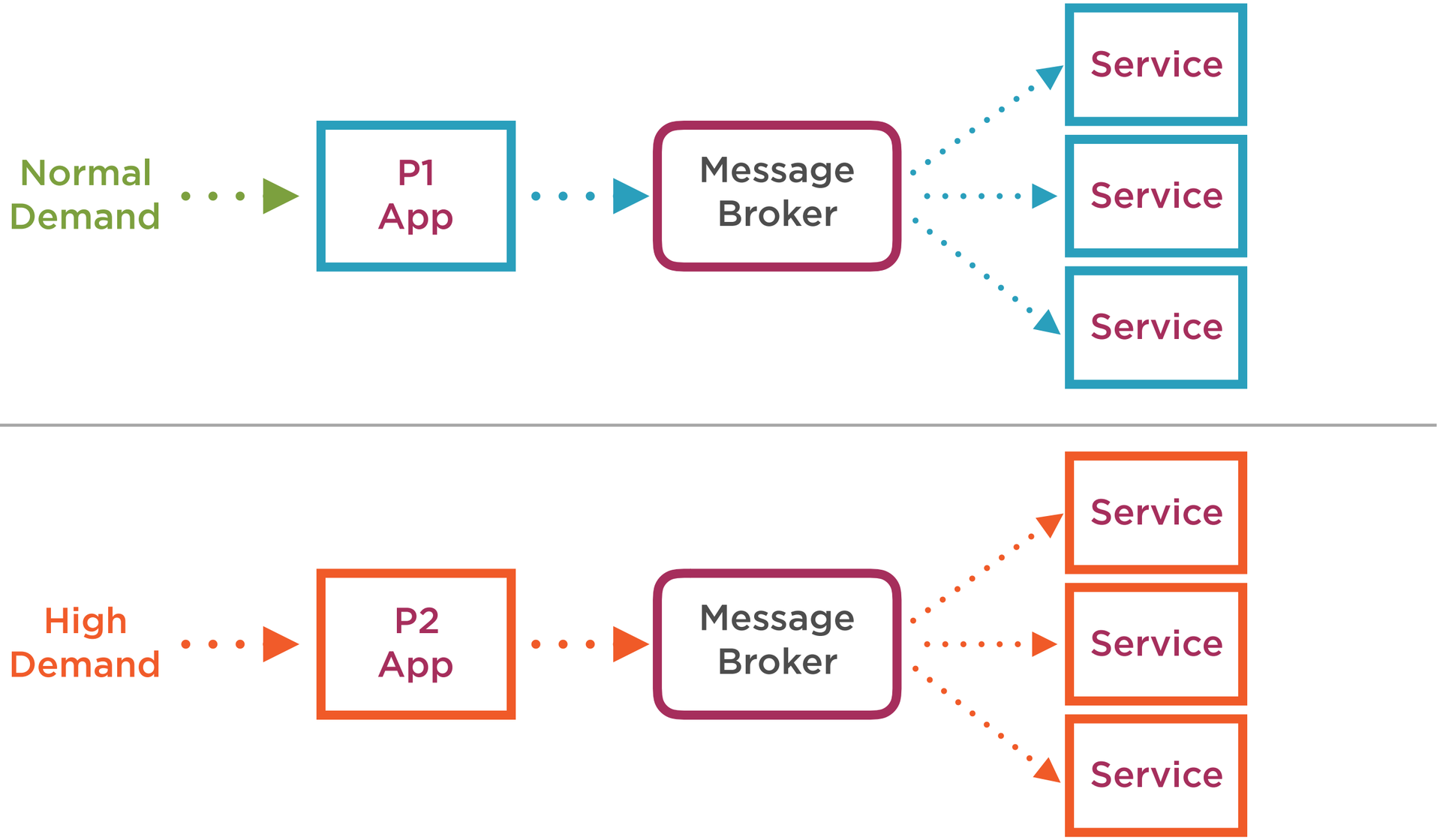
How?

- Separation by criticality
- Isolated microservices
- Redundancy
- Circuit breakers for all synchronous calls
- Rejecting calls using Load Shedding

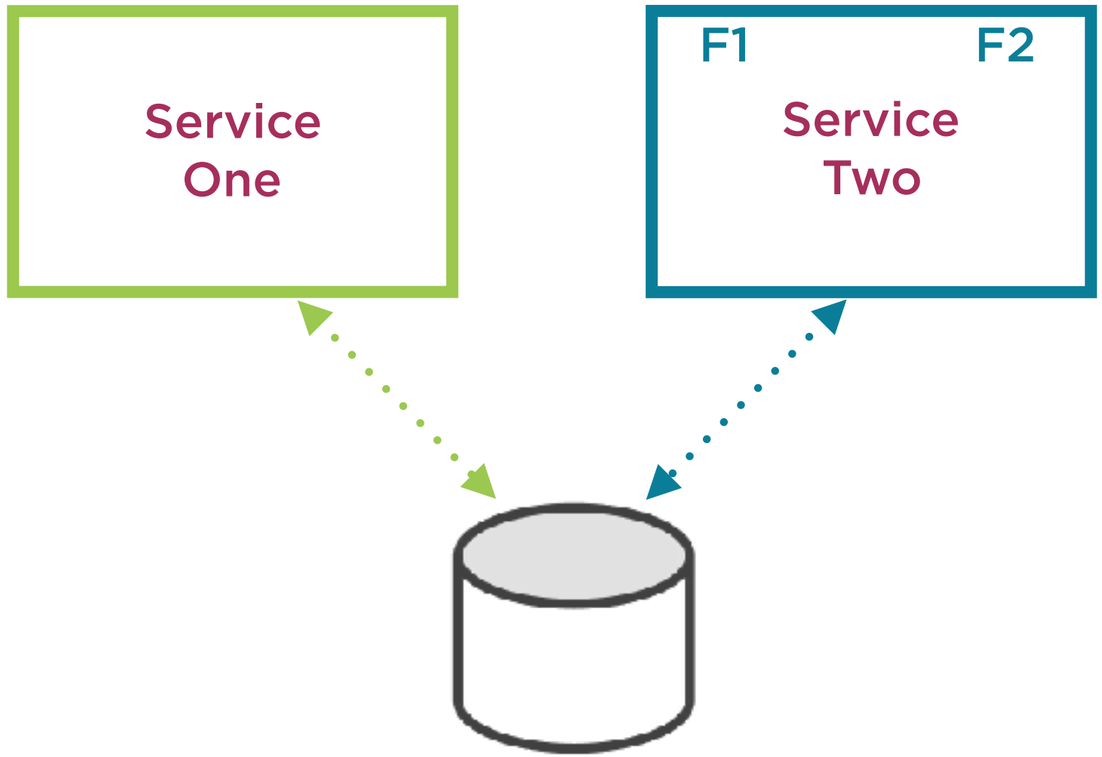
Bulkhead Pattern: Separation by Criticality



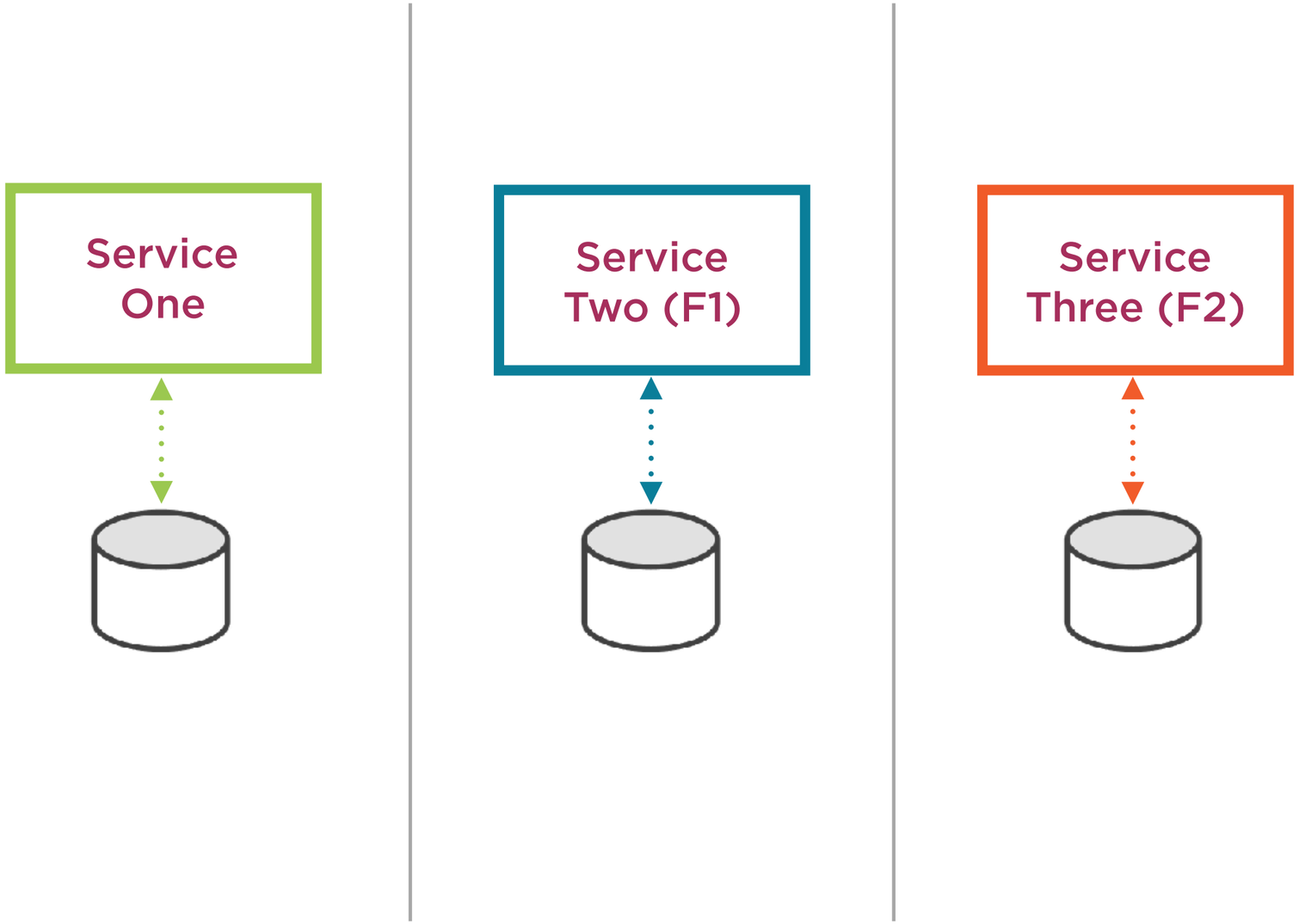
Bulkhead Pattern: Separation using Criticality



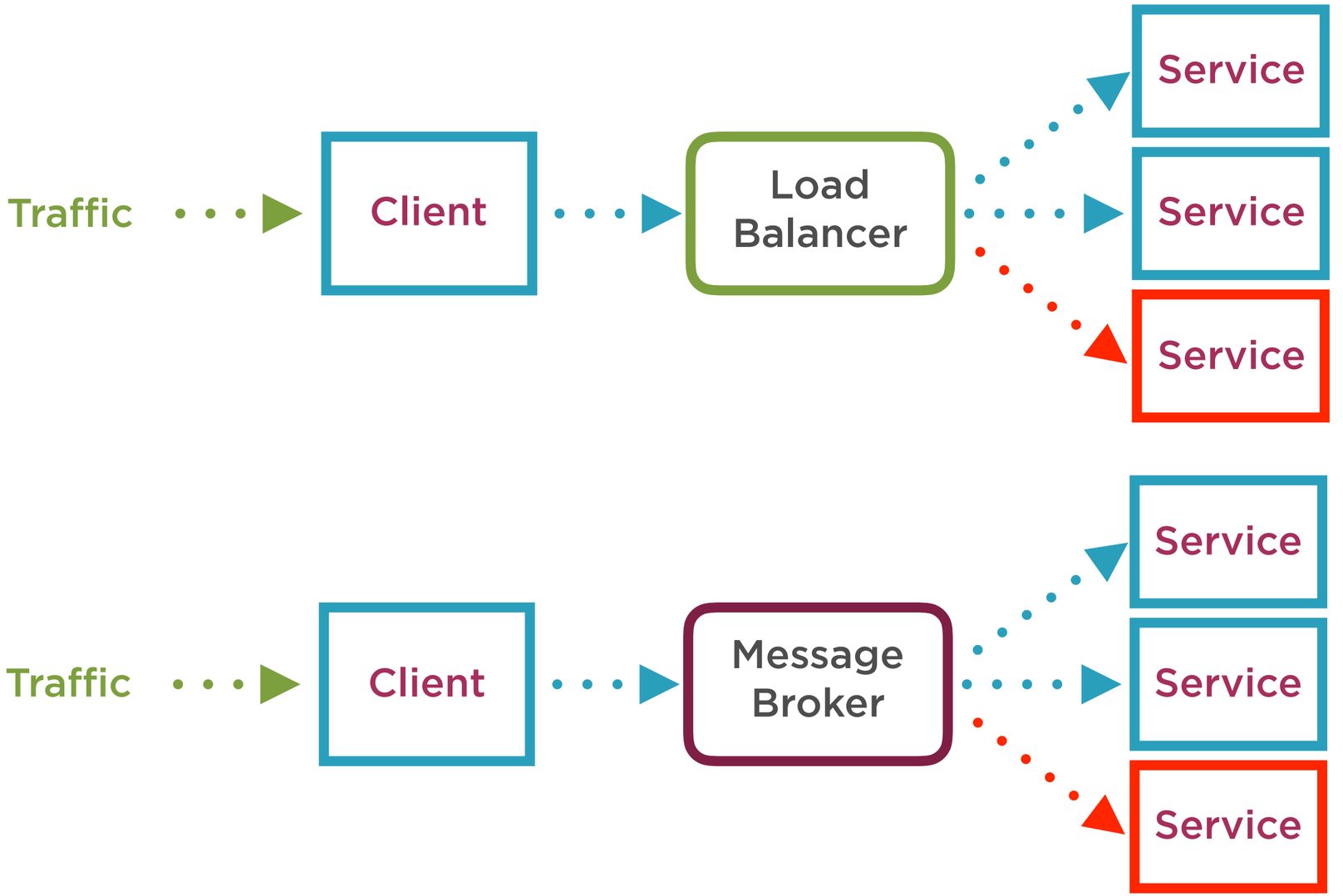
Bulkhead Pattern: Isolating Microservices



Bulkhead Pattern: Isolating Microservices

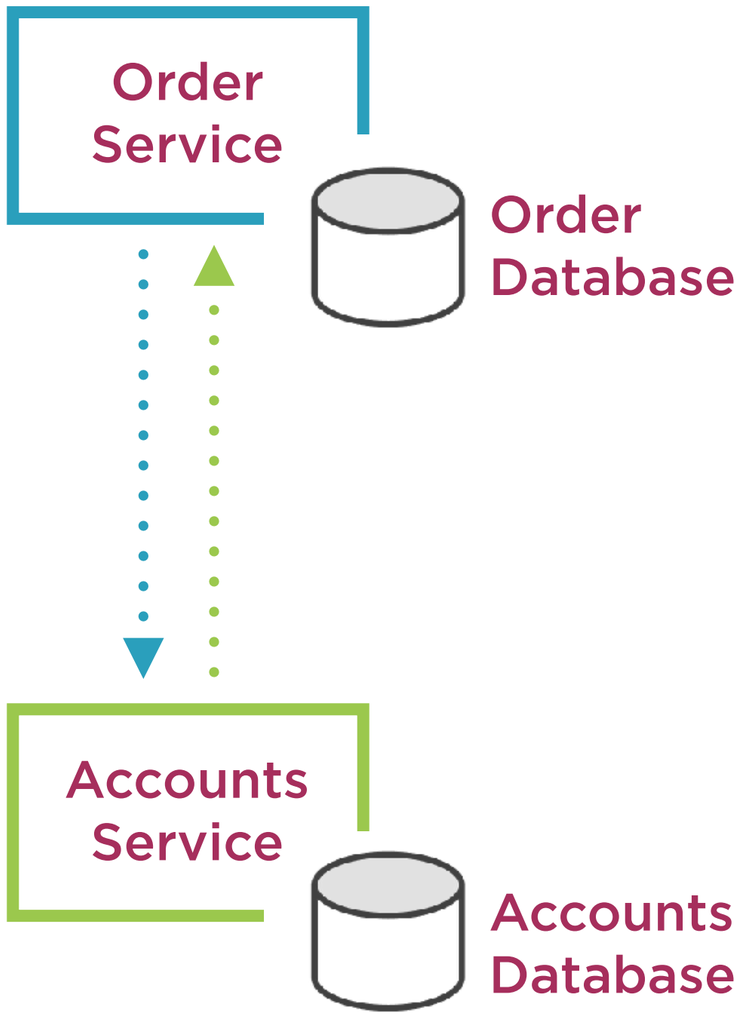


Bulkhead Pattern: Redundancy



Making a Resilient Microservices Architecture

Resiliency: How?



Design for known failures

- Look at previous failure causes
- Architect away single point of failures
- Use bulkhead pattern

Embrace failure

- Circuit breakers/retries with monitoring

Fail fast

- Using resiliency design patterns
- Fail fast for incoming request
  - Using timeouts on outbound calls

Degrade or default functionality

- Circuit breakers
- Use caches

Summary

Introduction

Patterns and Approach

Design Patterns

Approach to Design

Microservices Architectural Design Patterns Playbook