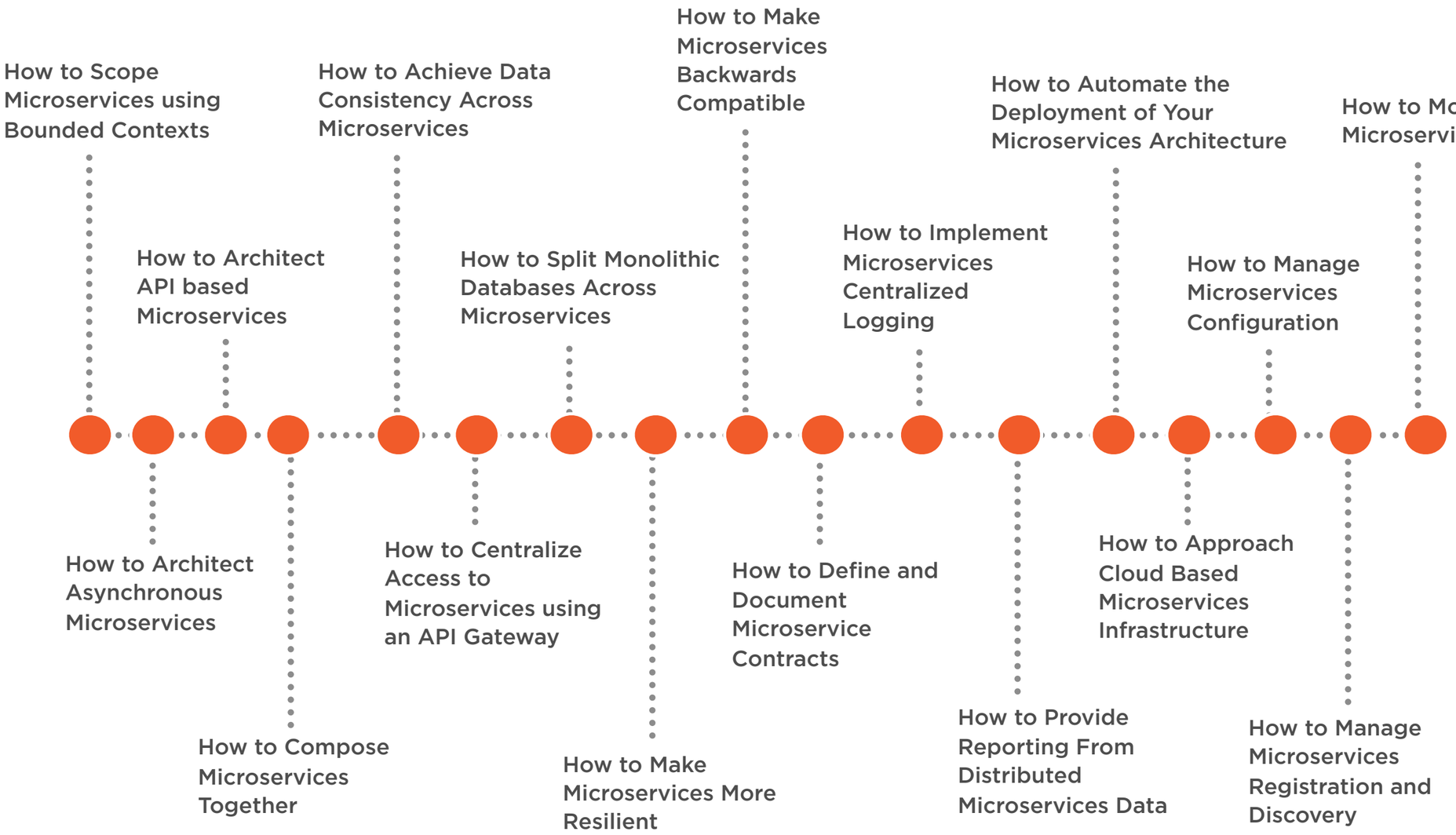
How to Architect Asynchronous Microservices

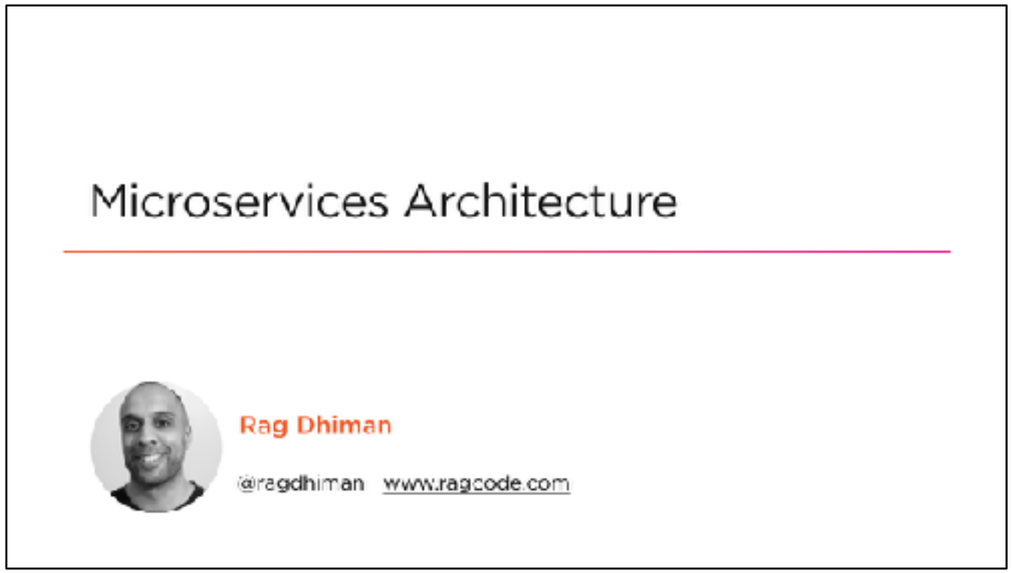
Rag Dhiman

@ragdhiman www.ragcode.com

Microservices Architectural Design Patterns Playbook



Microservices Architectural Design Patterns Playbook



Overview

Introduction

Event Based

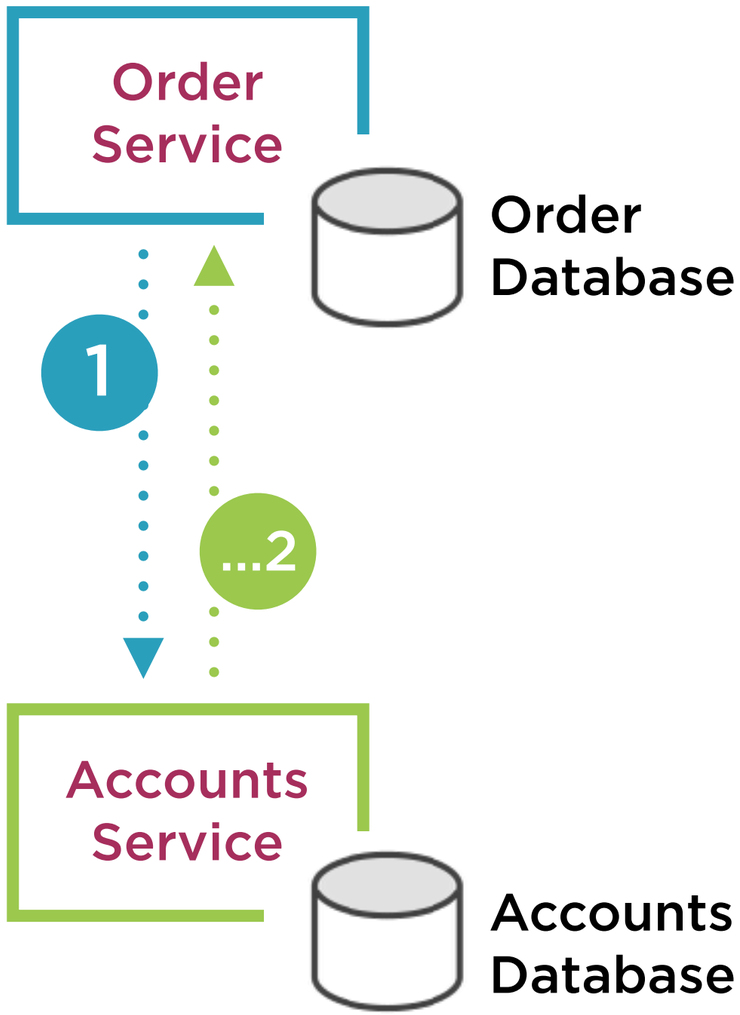
- Competing Workers Pattern
- Fanout Pattern

Async API Calls

- Request/Acknowledge with Callback

Introduction

Introduction



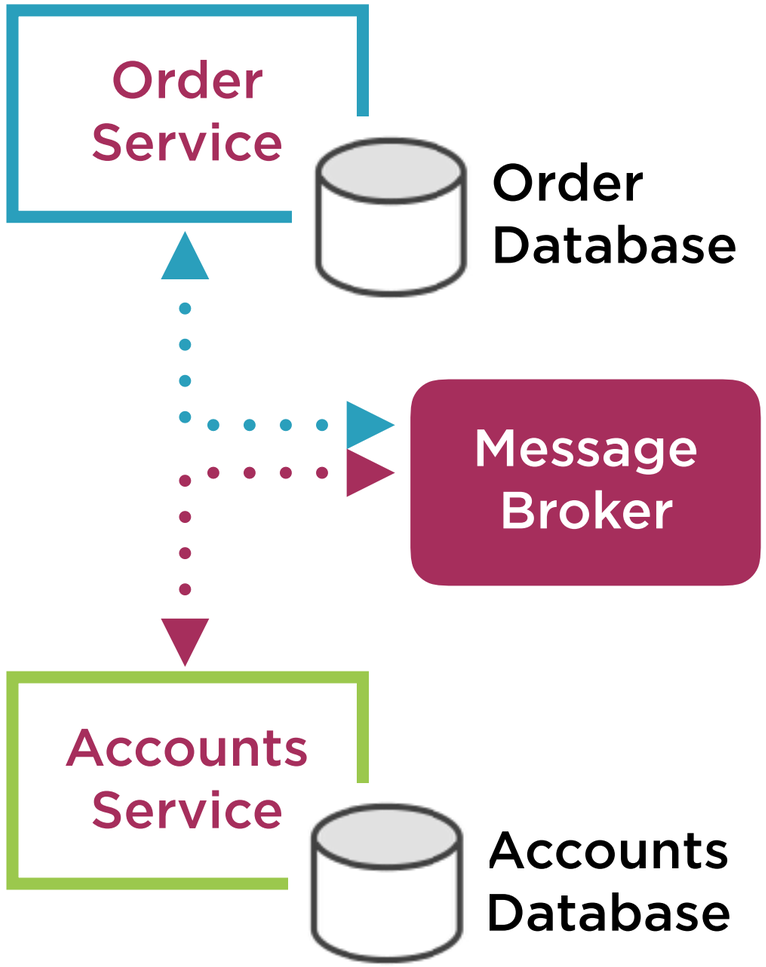
Why asynchronous microservices

- Fire and forget interactions
- Long running jobs
- Decoupled client and service
- Better user experience

How

- Event based
- Asynchronous API calls

Options



Event based

- Transaction/action as an event
- Messages using message brokers
- Decouples client and service
- Queuing Pattern

Async API calls

- Request/Acknowledge using callbacks

Other options

- Hangfire

Event Based

Event as a Message

Event as a Message

Demo

Message Broker

- RabbitMQ

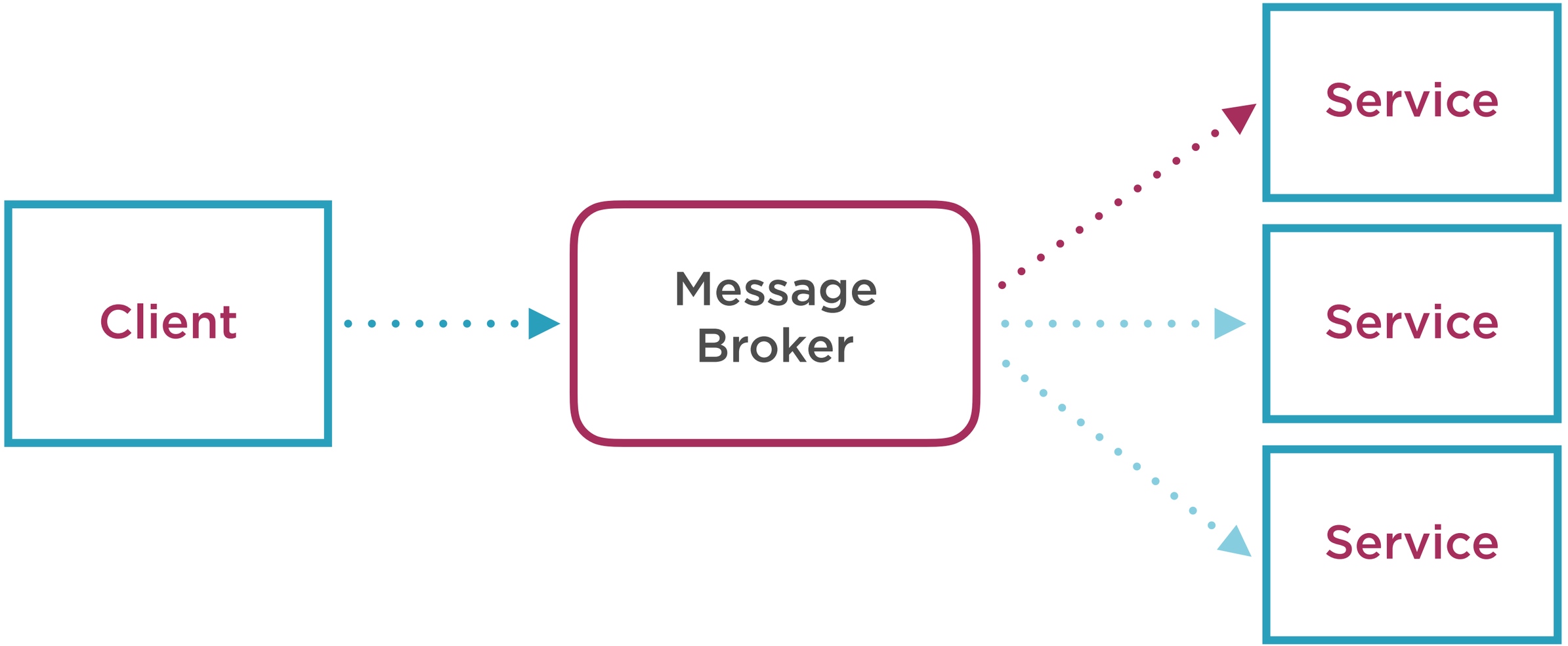
Client Application

- Message sender

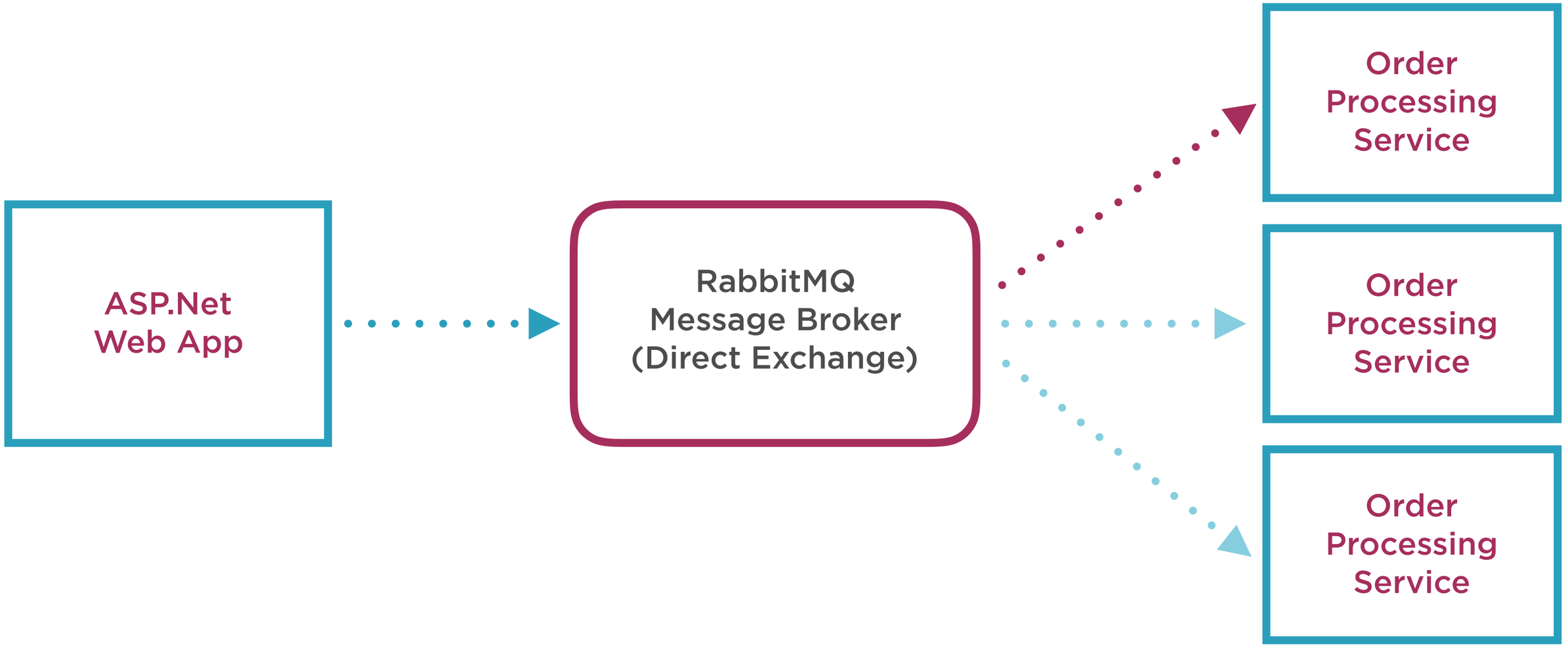
Server Application

- Microservice
- Message receiver

Competing Workers Pattern



Competing Workers Pattern

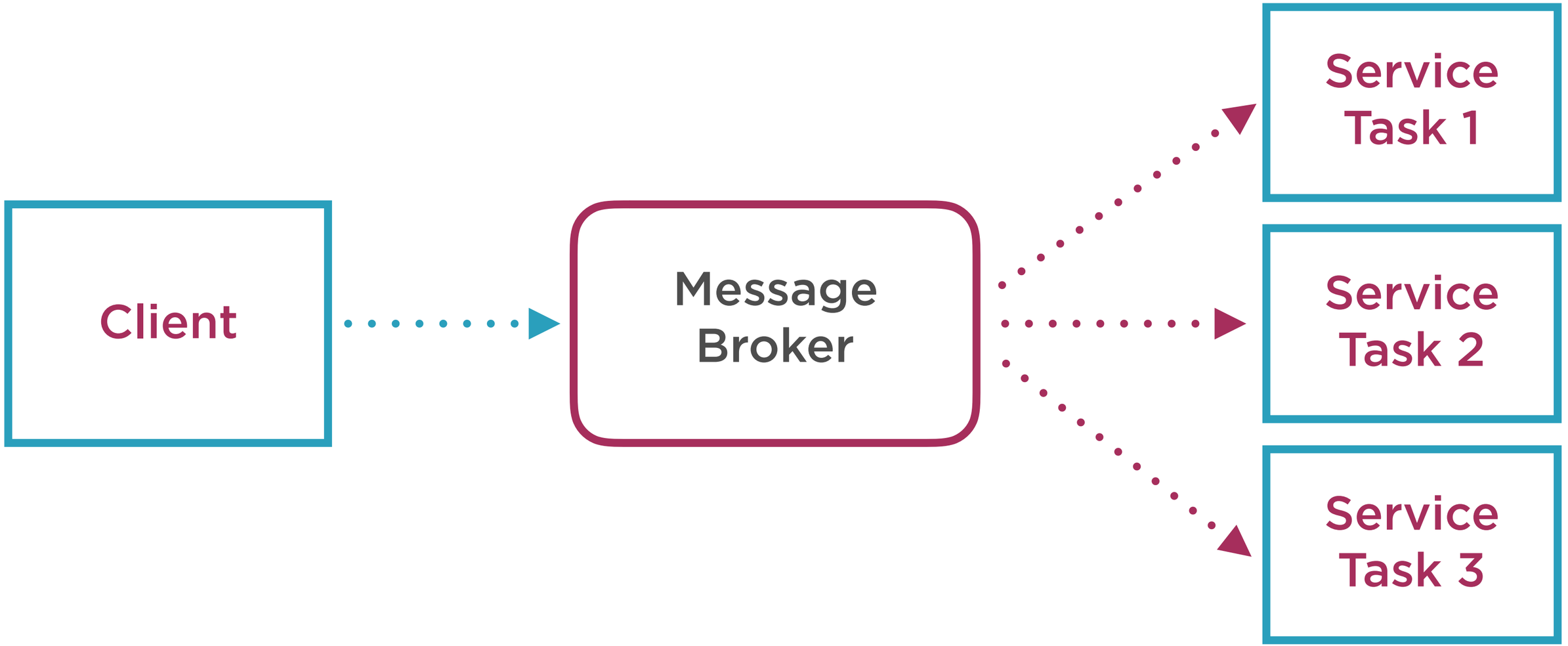


Demo

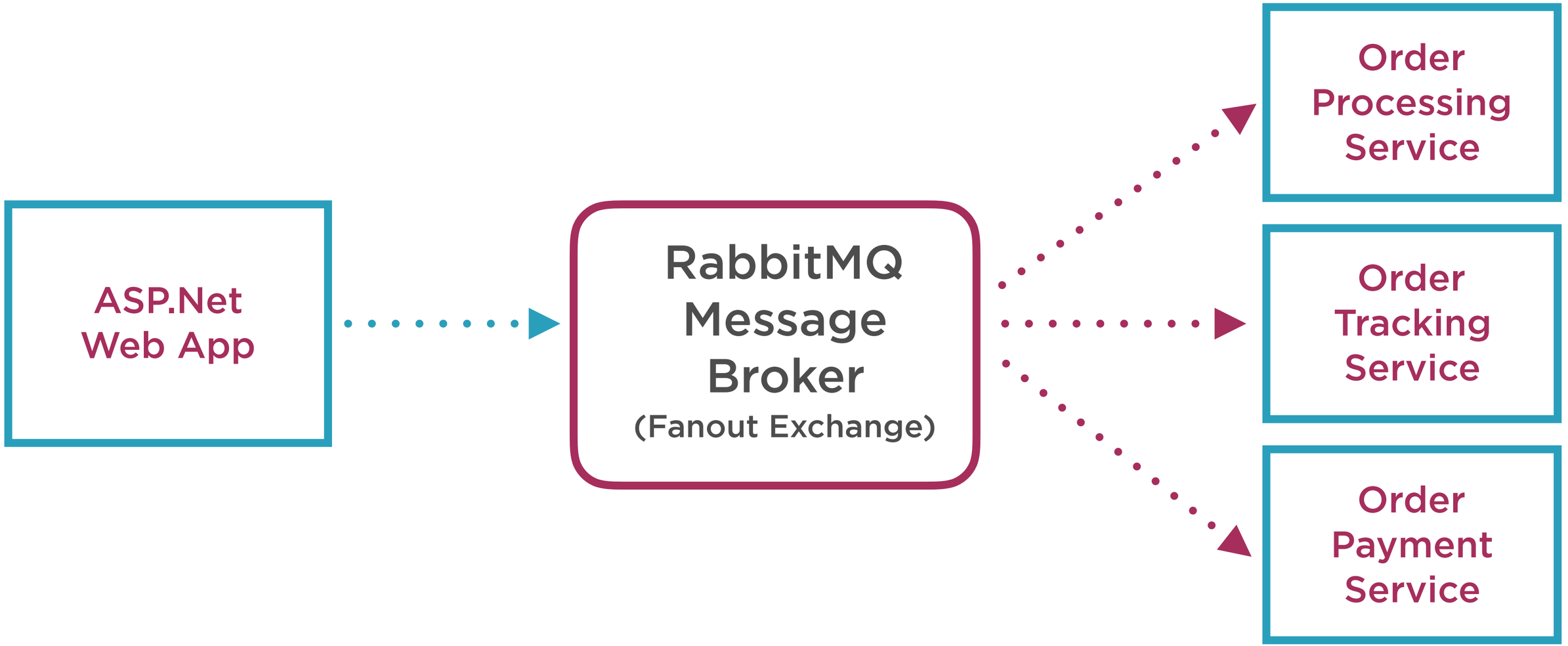
Competing Worker Pattern

- Client application
- Multiple competing microservices

Fanout Pattern



Fanout Pattern



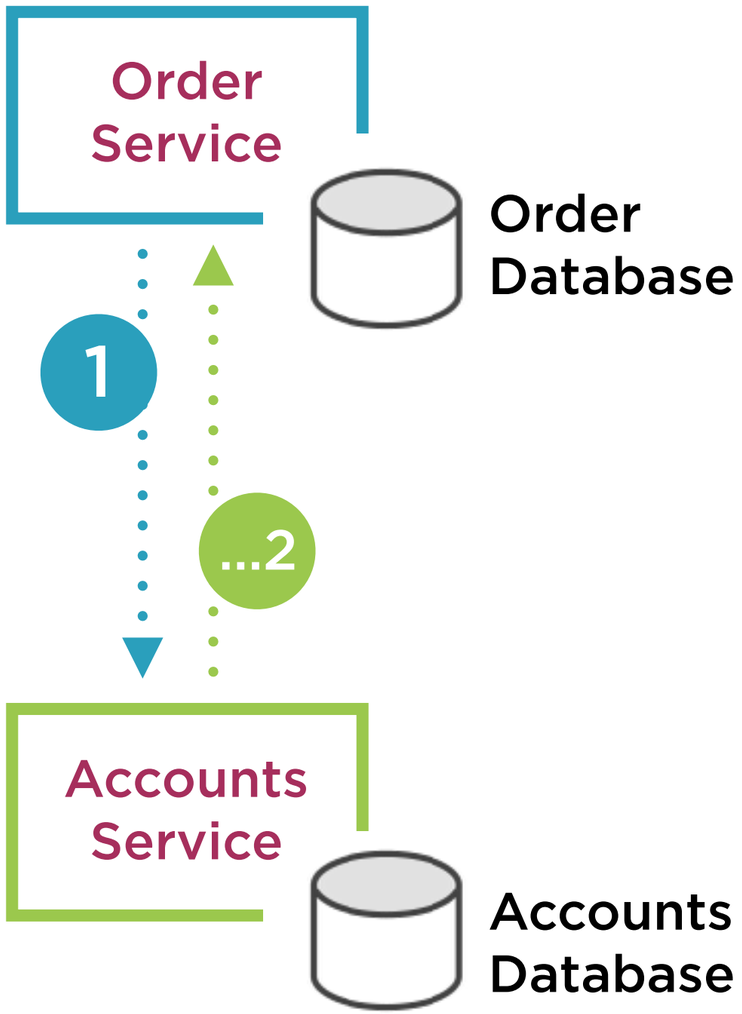
Demo

Fanout Pattern

- Client application
- Multiple receiving microservices

Async API Calls

Introduction



Request / Response

- Client calls service
- Service carries out task
- Client receives response

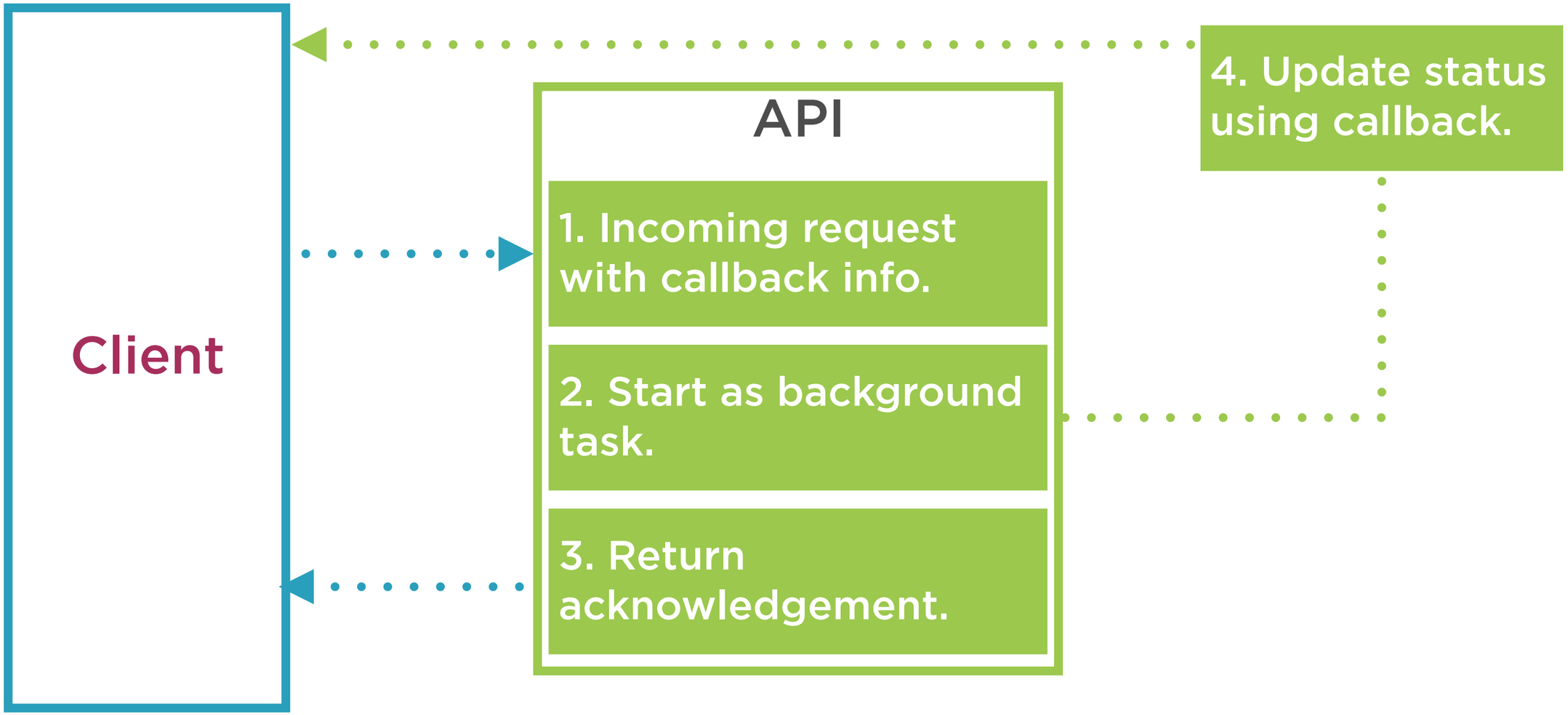
Synchronous

- Traditional for request/response
- Call waits for response (blocks)

Asynchronous

- Possible for request/response
- Registers a callback and client unblocks
- Response arrives on callback

Request/Acknowledge with Callback



```
public HttpResponseMessage
Post(Request request)
{

    ValidateRequest(request);


    QueueForProcessing(request);


    return HttpStatusCode.Accepted;
}
```

- Request with client callback info
- Validate request
- Process the request using a background mechanism
- Return accepted Http status

Summary

Introduction

Event Based

- Competing Workers Pattern
- Fanout Pattern

Async API Calls

- Request/Acknowledge with Callback

Microservices Architectural Design Patterns Playbook