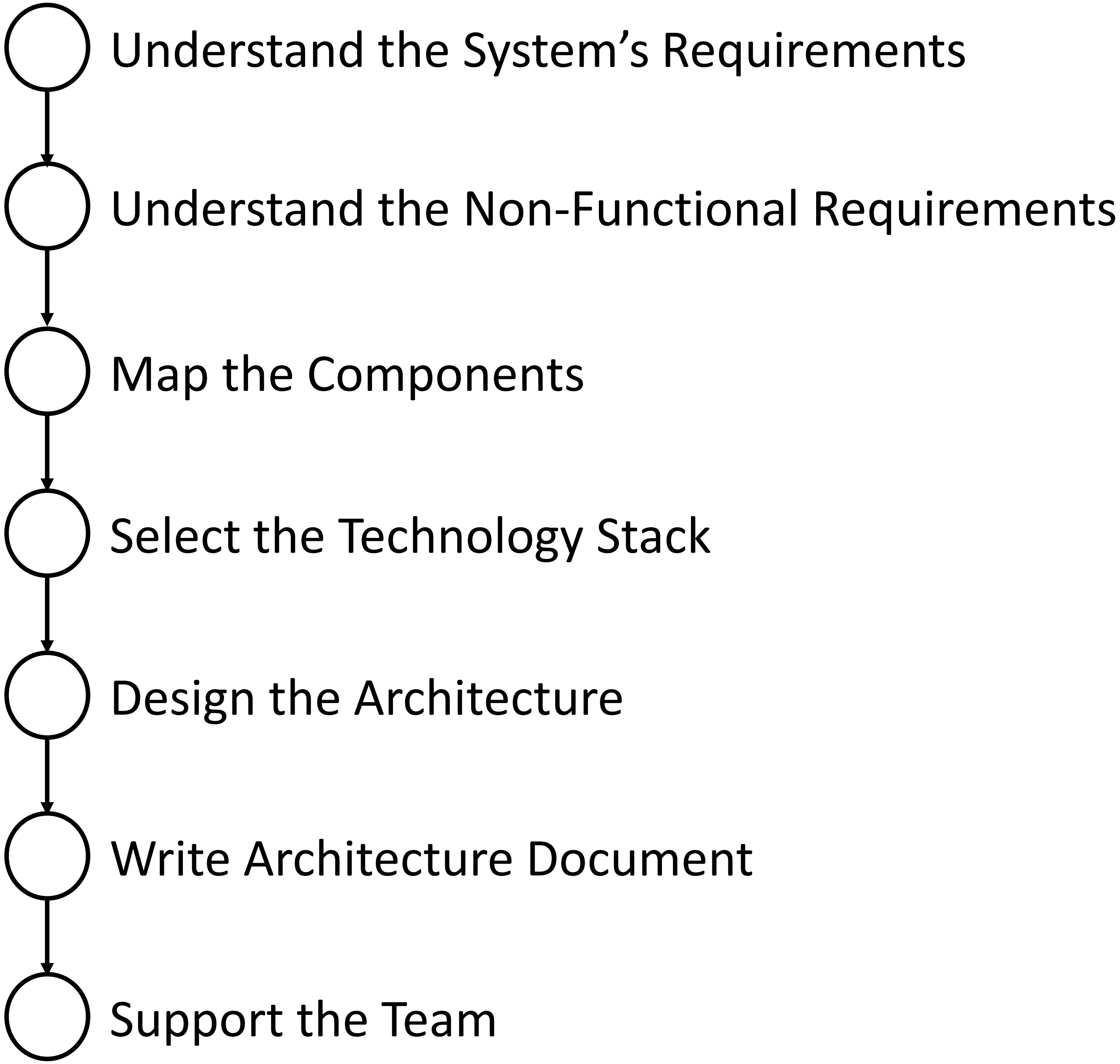
Designing Microservices Architecture

Memi Lavi
www.memilavi.com

Architecture Process

- Designing Microservices Architecture should be methodical
- Do not rush into development
- “Plan more, code less”
- Critical to the success of the system

The Architecture Process



Understand the System’s Requirements

Understand the Non-Functional Requirements

Map the Components

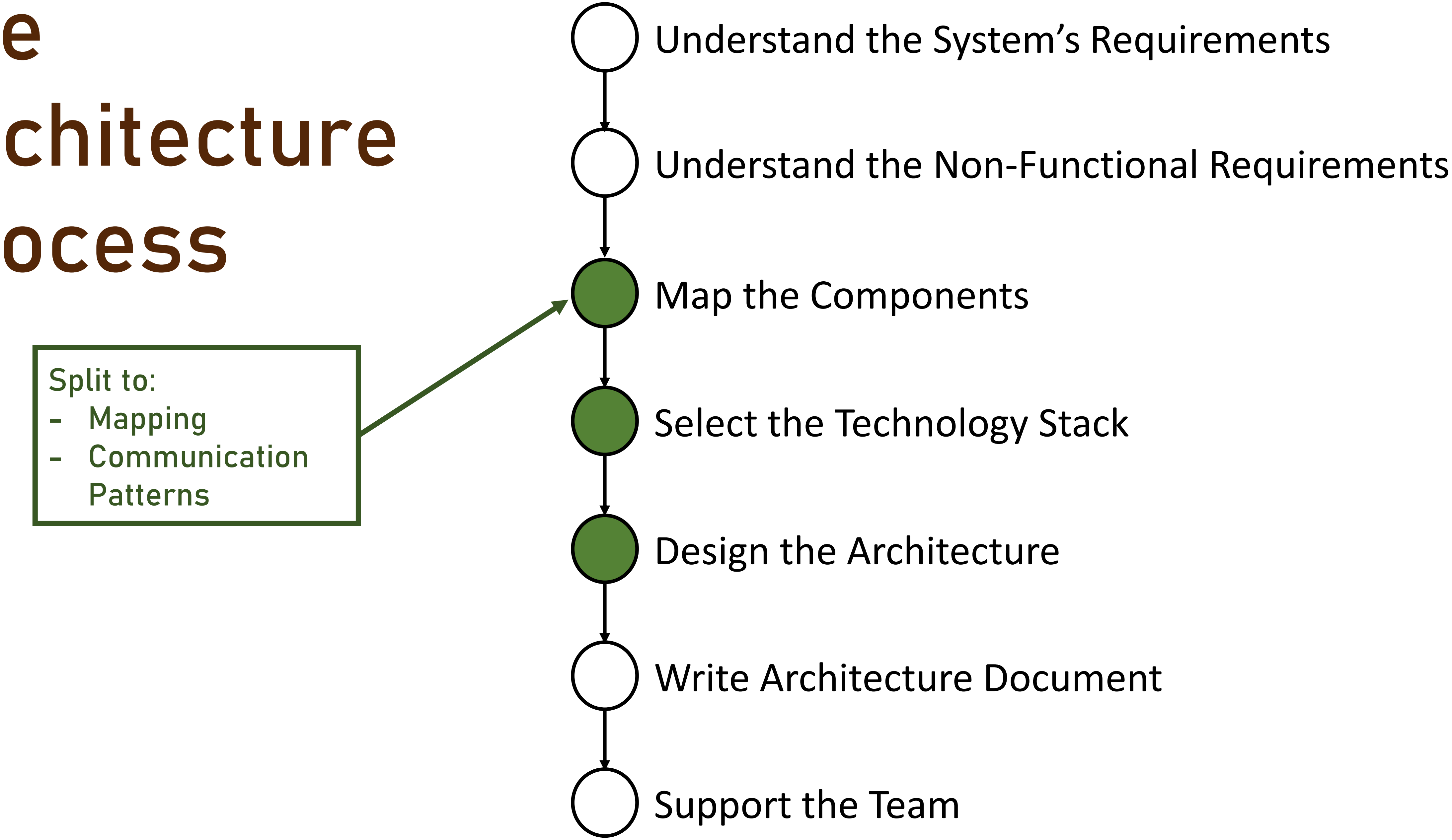
Select the Technology Stack

Design the Architecture

Write Architecture Document

Support the Team

The Architecture Process



- ○ Understand the System’s Requirements
- ○ Understand the Non-Functional Requirements
- ● Map the Components
- ● Select the Technology Stack
- ● Design the Architecture
- ○ Write Architecture Document
- ○ Support the Team

Split to:

- Mapping
- Communication Patterns

Mapping the Components

- The single most important step in the whole process

- Determines how the system will look like in the long run

- Once set – not easy to change

Mapping the Components

- What is it?
  - Defining the various components of the system
  - Remember: Components = Services

Mapping the Components

- Mapping should be based on:
  - Business requirements
  - Functional autonomy
  - Data entities
  - Data autonomy

Mapping the Components

- Business requirements:
  - The collection of requirements around a specific business capability
  - For example: Orders management
    - Add, remove, update, calculate amount

Mapping the Components

- Functional Autonomy:
  - The maximum functionality that does not involve other business requirements
  - For example:
    - Retrieve the orders made in the last week ✓
    - Get all the orders made by users aged 34-45 ✗

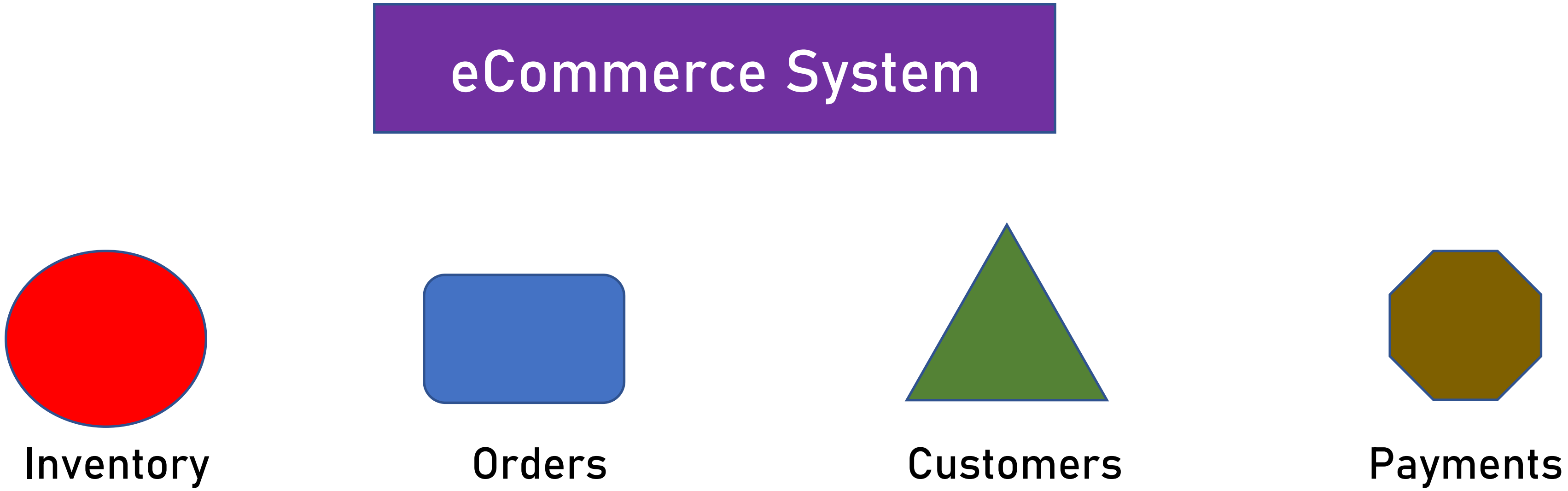
Mapping the Components

- Data Entities:
    - Service is designed around well-specified data entities
    - For example: orders, items
    - Data can be related to other entities but just by ID
        - Example: Order stores the Customer ID

Mapping the Components

- Data Autonomy:
    - Underlying data is an atomic unit
    - Service does not depend on data from other services to function properly
    - For example: Employees service that relies on Addresses service to return employee’s data

Mapping the Components – Example



eCommerce System

Inventory

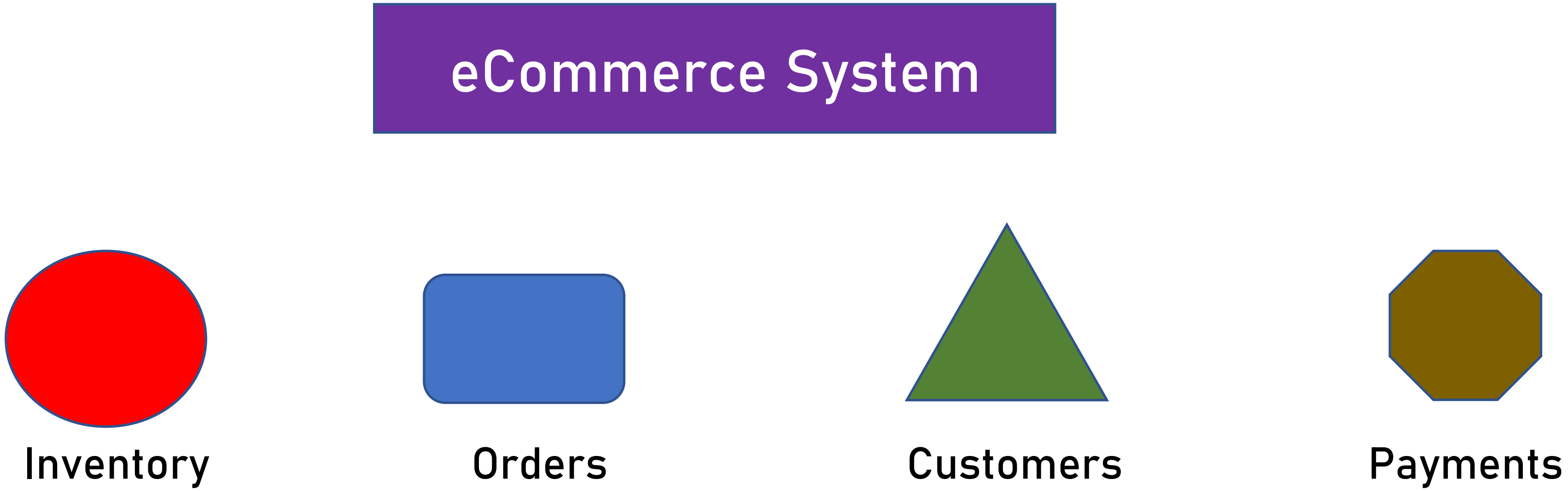
Orders

Customers

Payments

Business Requirements	Manage inventory items	Manage orders	Manage customers	Perform payments

Mapping the Components - Example



eCommerce System

Inventory

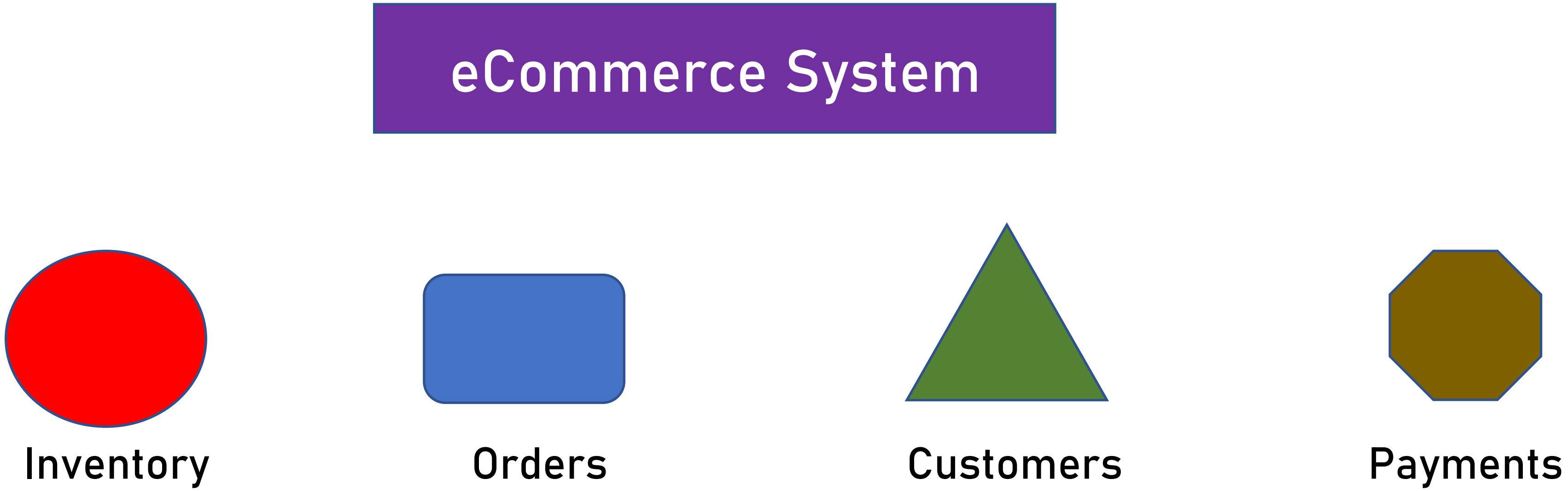
Orders

Customers

Payments

Business Requirements	Manage inventory items	Manage orders	Manage customers	Perform payments
Functional	Add, remove, update, quantity	Add, cancel, calculate sum	Add, update, remove, get account details	Perform payments

Mapping the Components – Example



eCommerce System

Inventory

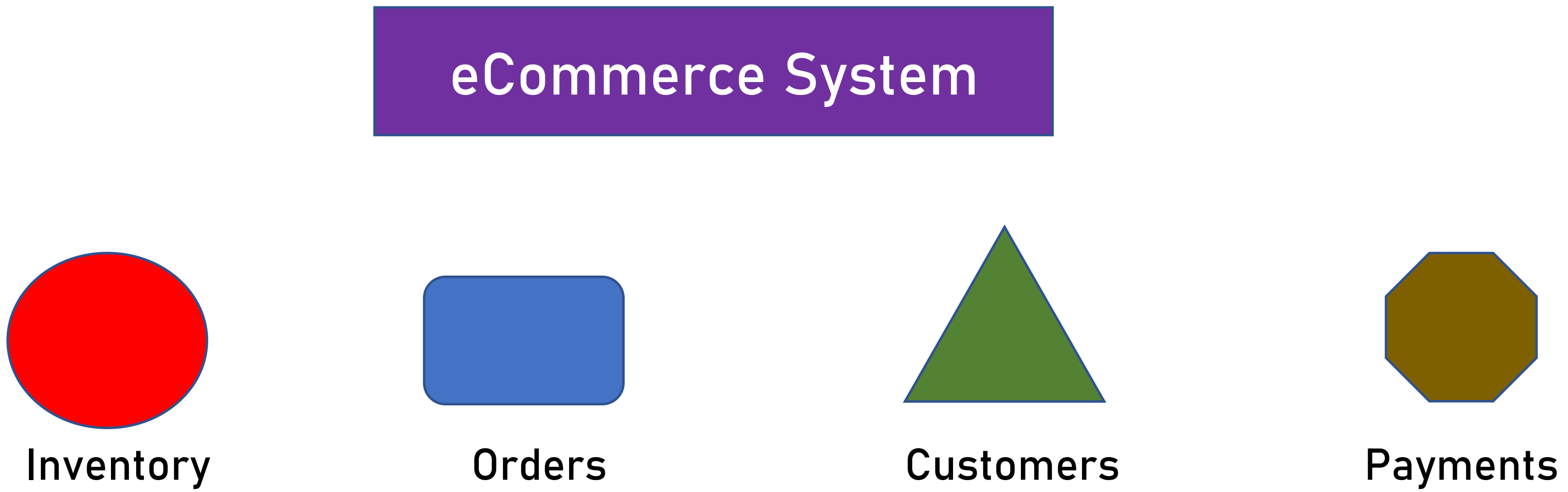
Orders

Customers

Payments

Business Requirements	Manage inventory items	Manage orders	Manage customers	Perform payments
Functional	Add, remove, update, quantity	Add, cancel, calculate sum	Add, update, remove, get account details	Perform payments
Data Entities	Items	Orders, shipping address	Customer, address, contact details	Payment history

Mapping the Components – Example



eCommerce System

Inventory

Orders

Customers

Payments

Business Requirements	Manage inventory items	Manage orders	Manage customers	Perform payments
Functional	Add, remove, update, quantity	Add, cancel, calculate sum	Add, update, remove, get account details	Perform payments
Data Entities	Items	Orders, shipping address	Customer, address, contact details	Payment history
Data Autonomy	None	Related to Items by ID Related to Customer by ID	Related to Orders by ID	None

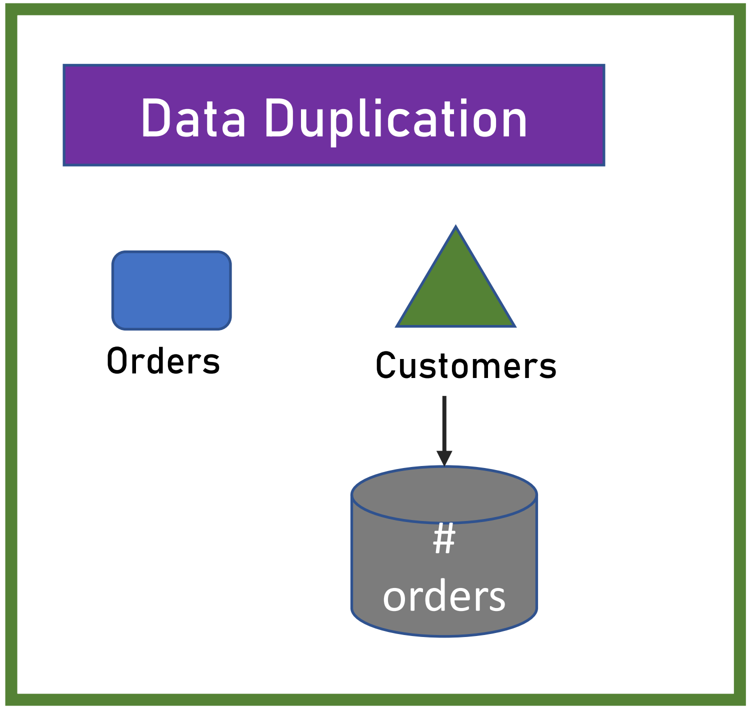
Mapping the Components

- Edge case #1:
  - Retrieve all customers from NYC with total number of orders for each customer

Customer name	No. of orders
David Smith	16
Diane Rice	23
George Murray	22

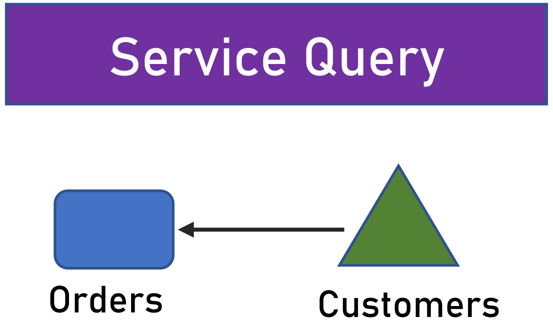
Mapping the Components

- Three approaches:

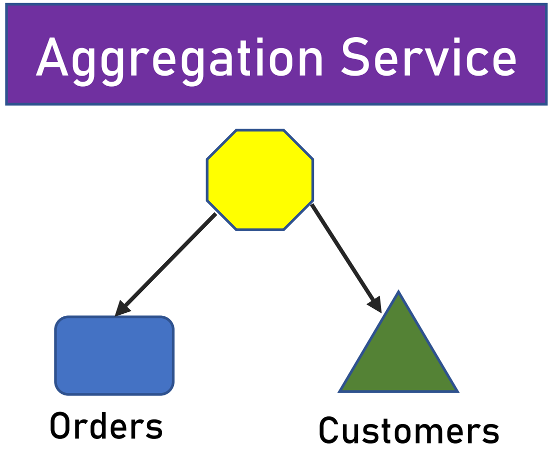


Data Duplication

- Very little data
- Read only



Service Query



Aggregation Service

Mapping the Components

- Edge case #2:
    - Retrieve list of all the orders in the system

Mapping the Components

- Services are not designed for this scenario
- Find out what's the purpose of this query
- Report engine is the preferred mechanism for this

Cross-Cutting Services

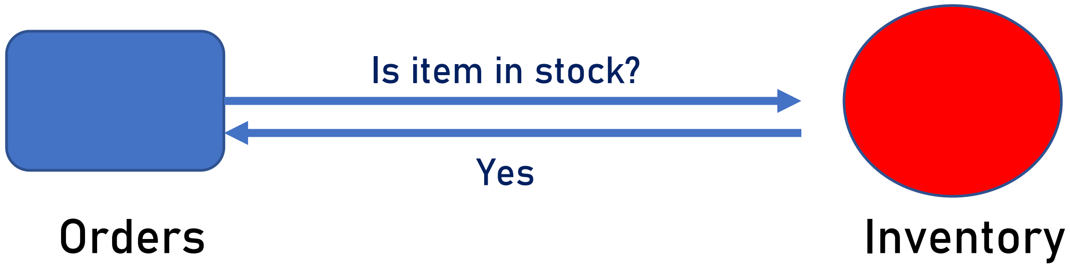
- Services that provide system-wide utilities
- Common examples:
  - Logging
  - Caching
  - User management
- MUST be part of the mapping

Defining Communication Patterns

- Efficient communication between services is crucial
- It’s important to choose the correct communication pattern
- Main patterns:
  - 1-to-1 Sync
  - 1-to-1 Async
  - Pub-Sub / Event Driven

1-to-1 Sync

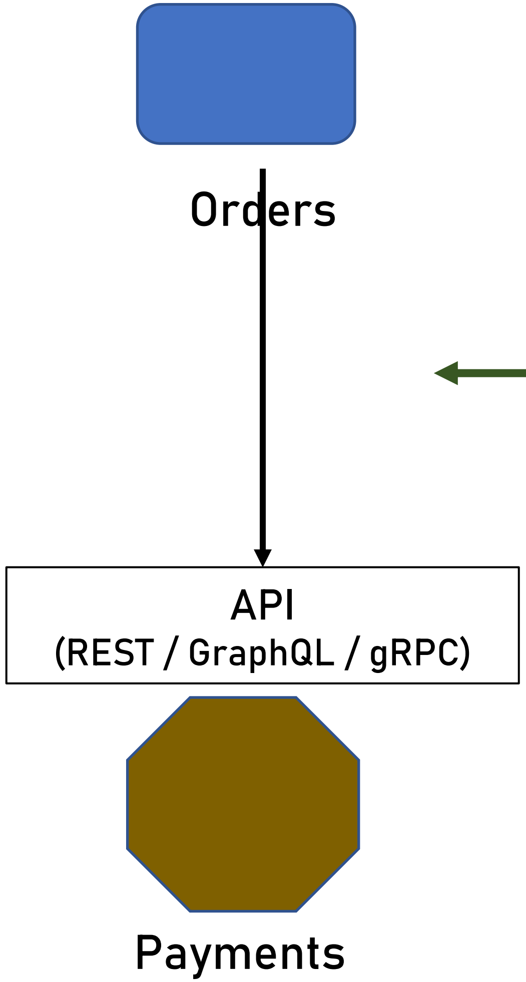
- A service calls another service and waits for the response
- Used mainly when the first service needs the response to continue processing



1-to-1 Sync

Pros	Cons
- Immediate response	- Performance
- Error handling	
- Easy to implement	

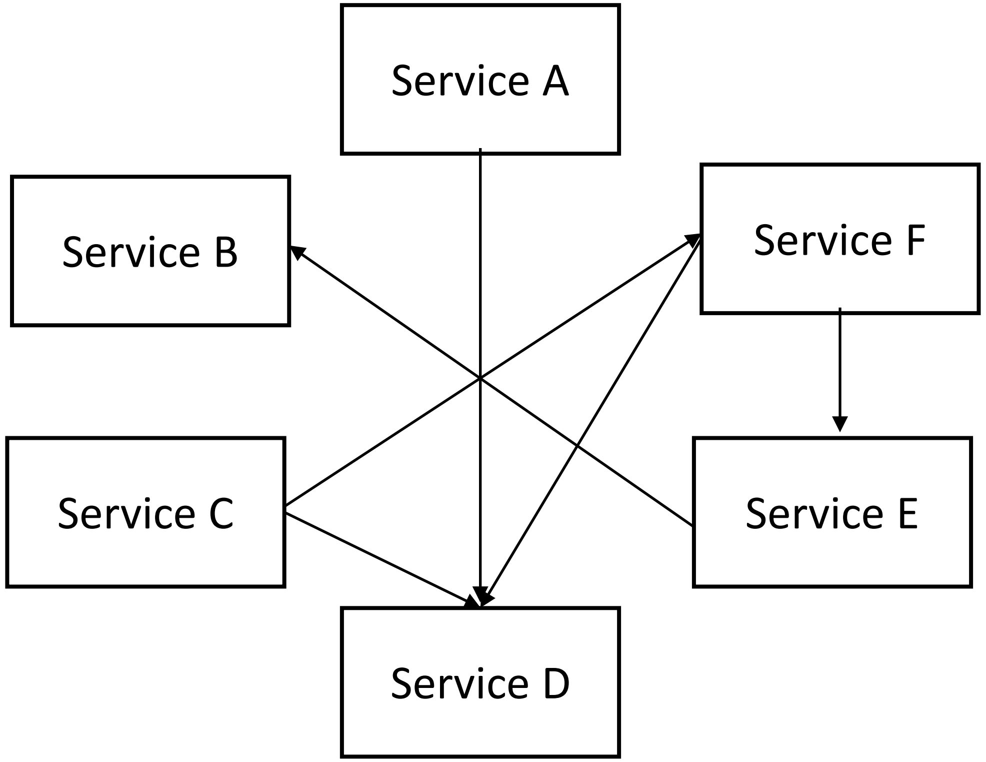
1-to-1 Sync Implementation



Direct Connection

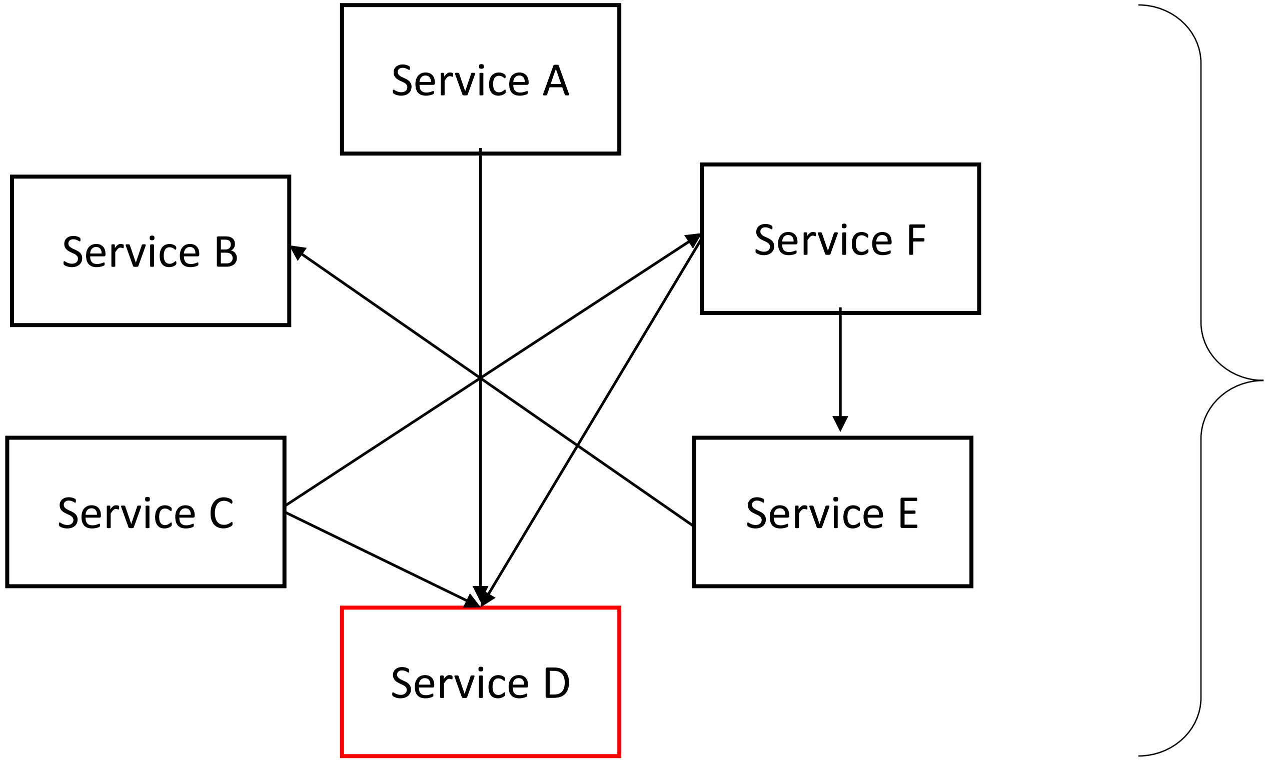
Usually not recommended

Direct Connection



The Spiderweb

Direct Connection



Service A

Service B

Service F

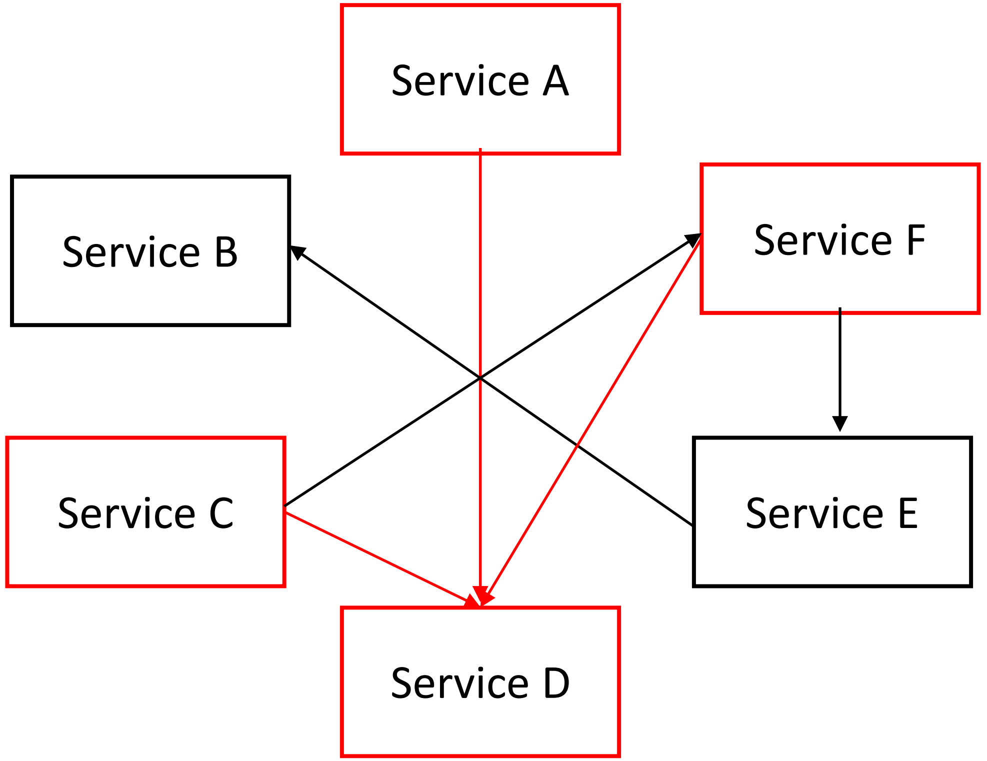
Service C

Service E

Service D

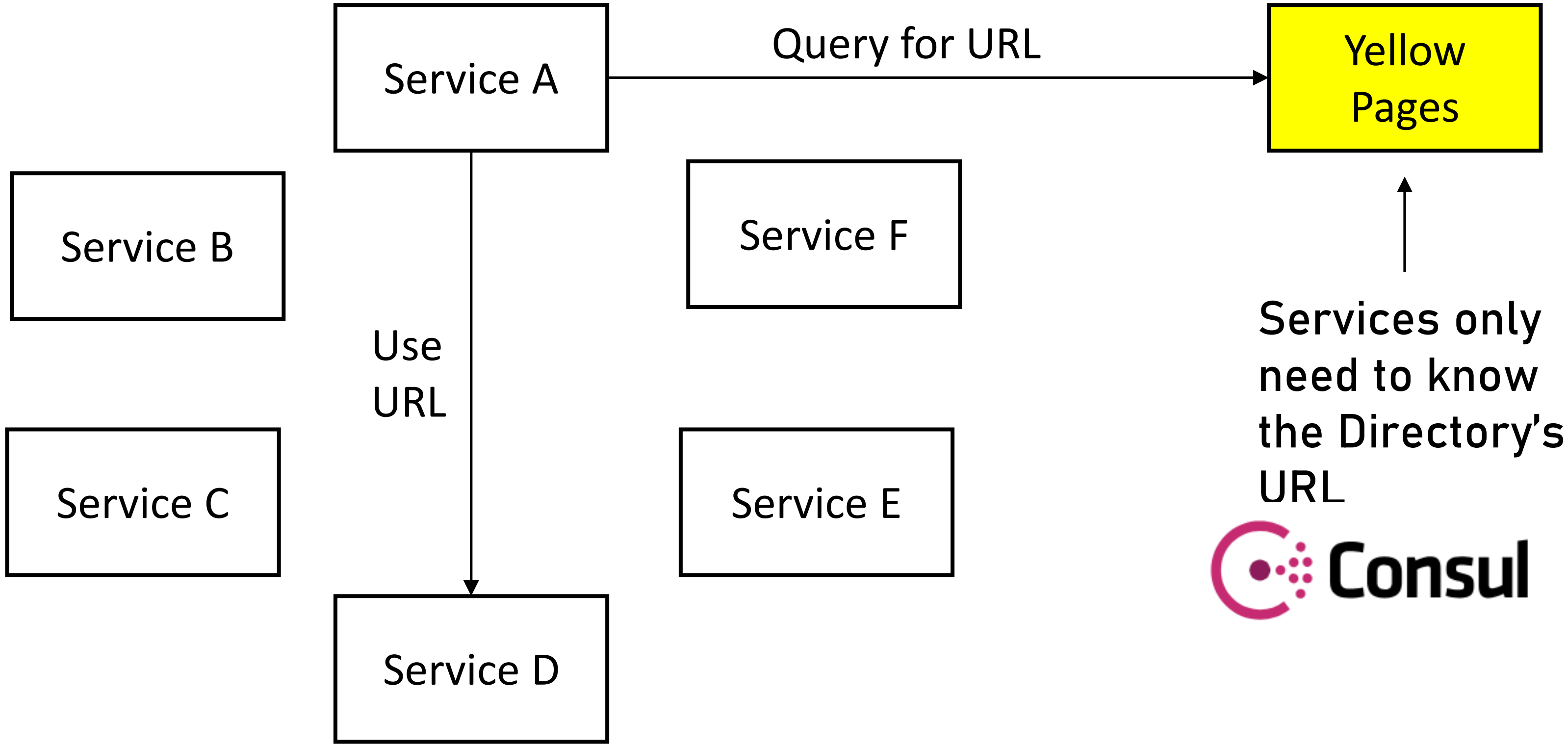
The Spiderweb

Direct Connection



The Spiderweb

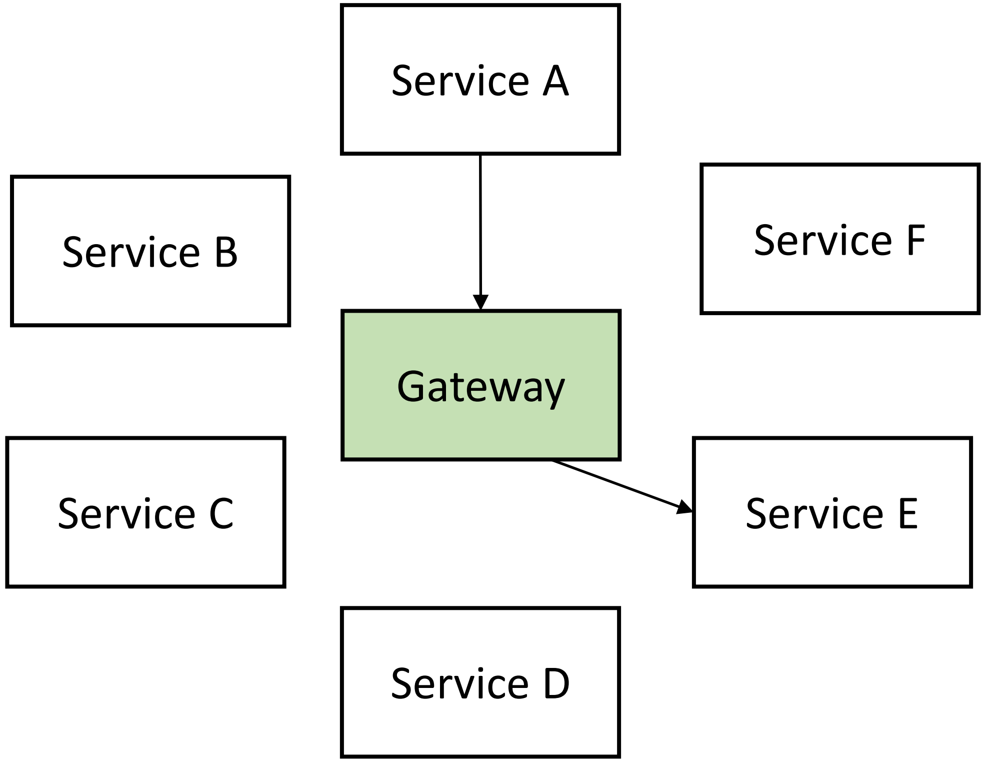
Service Discovery



Services only need to know the Directory's URL

Consul

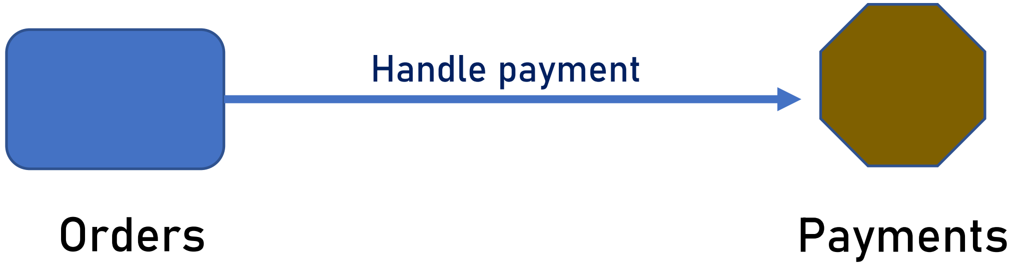
Gateway



Services only need to know the Gateway's URL

1-to-1 Async

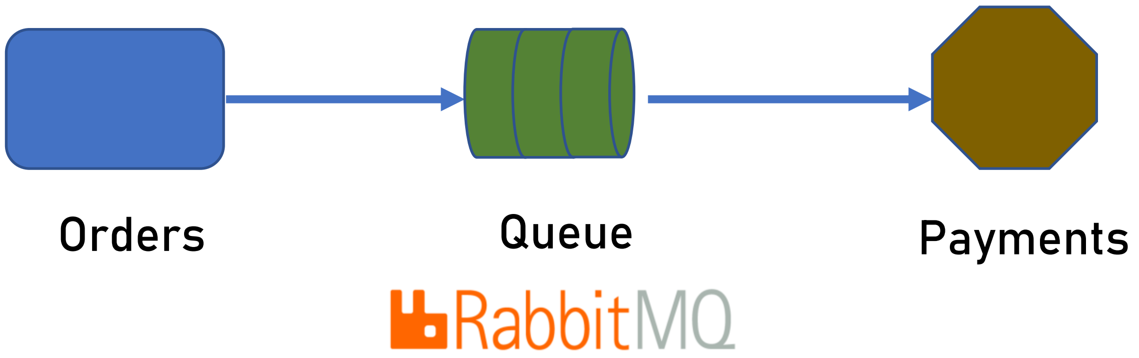
- A service calls another service and continues working
- Doesn't wait for response – Fire and Forget
- Used mainly when the first service wants to pass a message to the other service



1-to-1 Async

Pros	Cons
- Performance	- Needs more setup
	- Difficult error handling

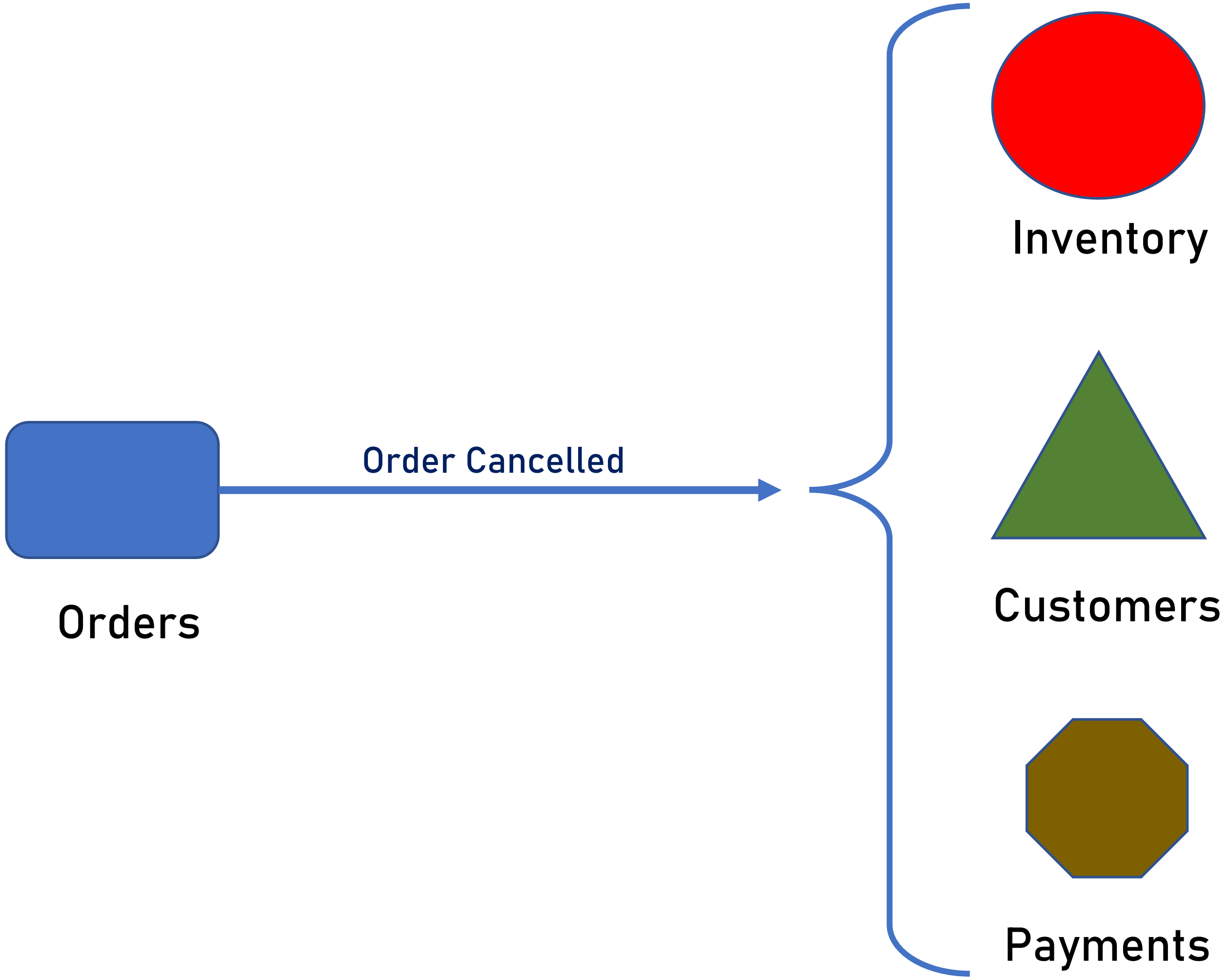
1-to-1 Async Implementation



Pub-Sub / Event Driven

- A service wants to notify other services about something
- The service has no idea how many services listen
- Doesn’t wait for response – Fire and Forget
- Used mainly when the first service wants to notify about an important event in the system

Pub-Sub / Event Driven



Pub-Sub / Event Driven

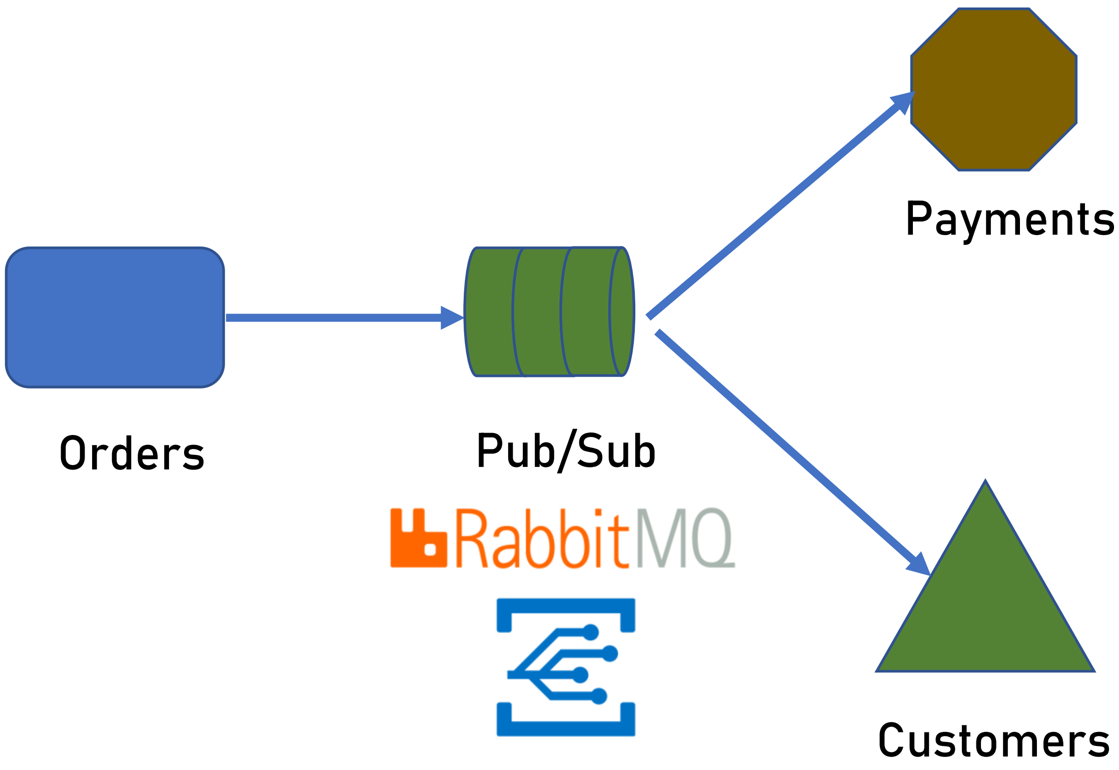
Pros

- Performance
- Notify multiple services at once

Cons

- Needs more setup
- Difficult error handling
- Might cause load

Pub-Sub / Event Driven Implementation



Communication Patterns Summary

- Choosing the correct communication pattern is crucial
- Affects:
  - Performance
  - Error Handling
  - Flow
- Almost impossible to reverse

Selecting Technology Stack

- The Decentralized Governance allows selecting different technology stack for each service.
- We’ll focus on Backend platform and Storage platforms
- There’s no objective “Right” or “Wrong”
- Make it a concrete decision based on hard evidence

Development Platform

	App Types	Type System	Cross Platform	Community	Performance	Learning Curve
.NET	All	Static	No	Large	OK	Long
.NET Core	Web Apps, Web API, Console, Service	Static	Yes	Medium and growing rapidly	Great	Long
Java	All	Static	Yes	Huge	OK	Long
node.js	Web Apps, Web API	Dynamic	Yes	Large	Great	Medium
PHP	Web Apps, Web API	Dynamic	Yes	Large	OK -	Medium
Python	All	Dynamic	Yes	Huge	OK -	Short

Data Store

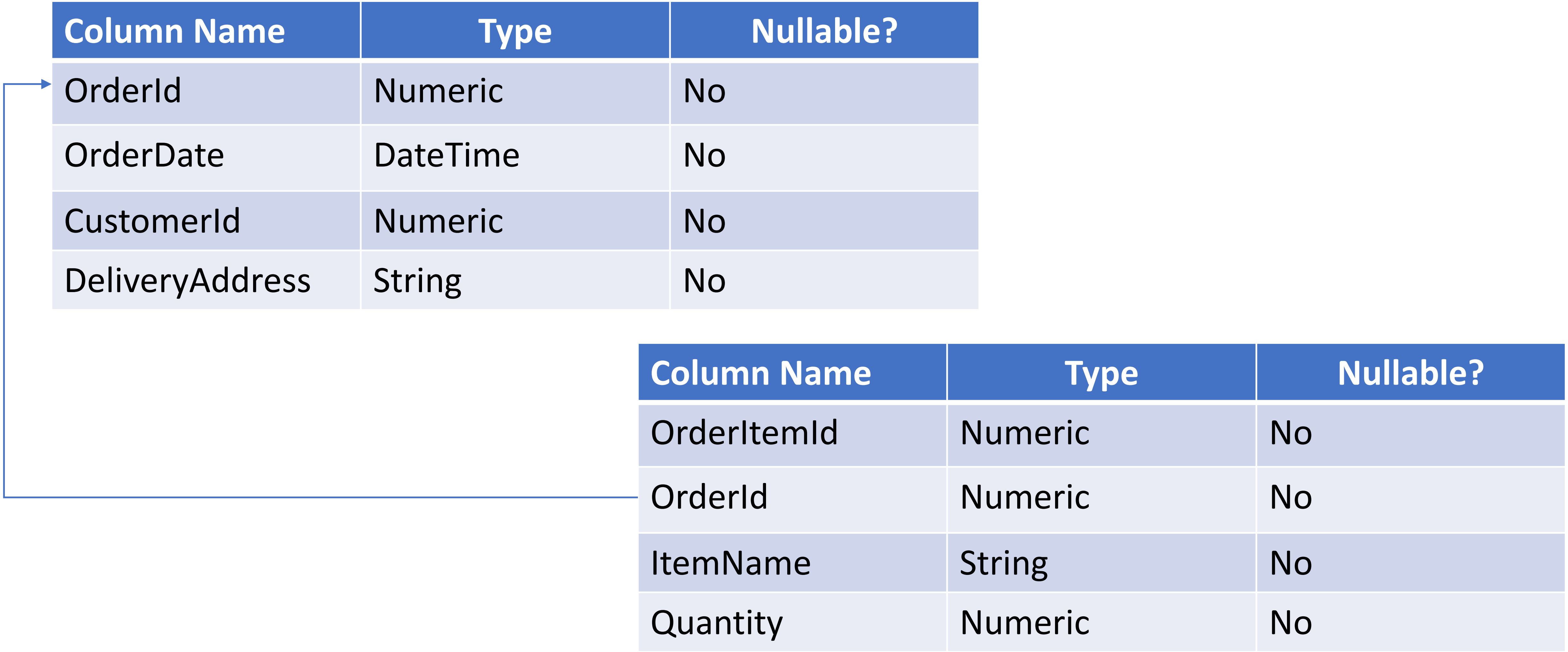
- 4 types of data store:
  - Relational Database
  - NoSQL Database
  - Cache
  - Object Store

Relational Database

- Stores data in tables
- Tables have concrete set of columns

Column Name	Type	Nullable?
OrderId	Numeric	No
OrderDate	DateTime	No
CustomerId	Numeric	No
DeliveryAddress	String	No

Relational Database



Column Name	Type	Nullable?
OrderId	Numeric	No
OrderDate	DateTime	No
CustomerId	Numeric	No
DeliveryAddress	String	No

Column Name	Type	Nullable?
OrderItemId	Numeric	No
OrderId	Numeric	No
ItemName	String	No
Quantity	Numeric	No

Relational Database

- Popular databases:

NoSQL Database

- Emphasis on scale and performance
- Schema-less
- Data usually stored in JSON format

NoSQL Database

- Popular databases:

mongoDB®

amazon DynamoDB

Couchbase

Azure Cosmos DB

Cache

- Stores in-memory data for fast access
- Distributes data across nodes
- Uses proprietary protocol
- Stores serializable objects

Cache

- Popular cache:

Object Store

- Stores un-structured, large data
  - Documents
  - Photos
  - Files

Object Store

- Popular stores:

Microsoft Azure
Blob Storage

amazon
S3

MINIO

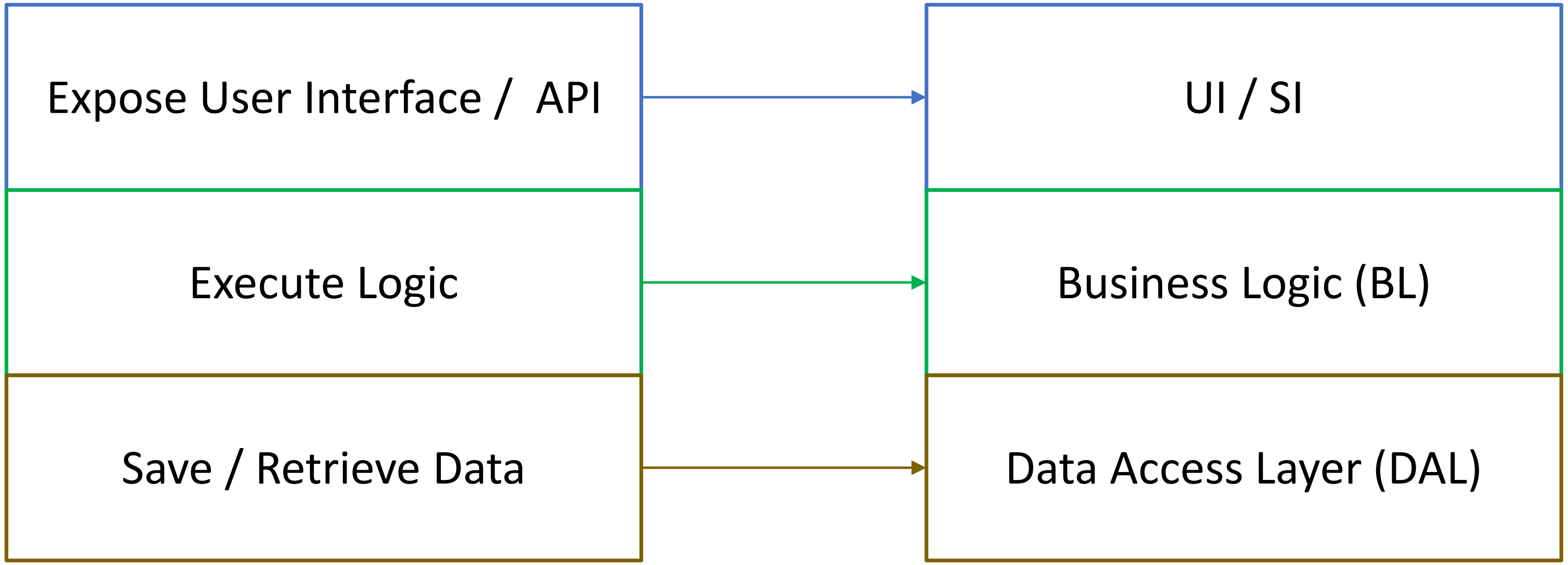
Design the Architecture

- Service's architecture is no different from regular software

- Based on the layers paradigm

Layers

- Represent horizontal functionality



Expose User Interface / API → UI / SI

Execute Logic → Business Logic (BL)

Save / Retrieve Data → Data Access Layer (DAL)

- Expose API
- JSON Handling
- Auth

- Validation
- Enrichment
- Computations

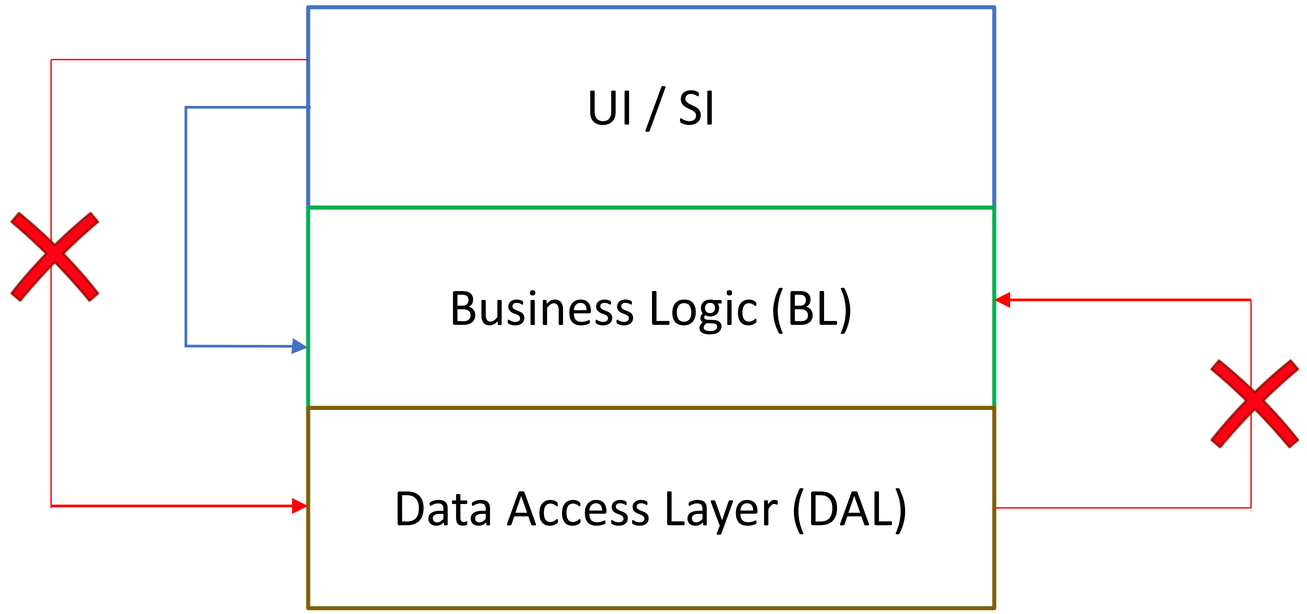
- Connection Handling
- Querying / Saving Data
- Transaction Handling

Purpose of Layers

- Forces well formed and focused code
- Modular

Concepts of Layers

- Code Flow



UI / SI

Business Logic (BL)

Data Access Layer (DAL)