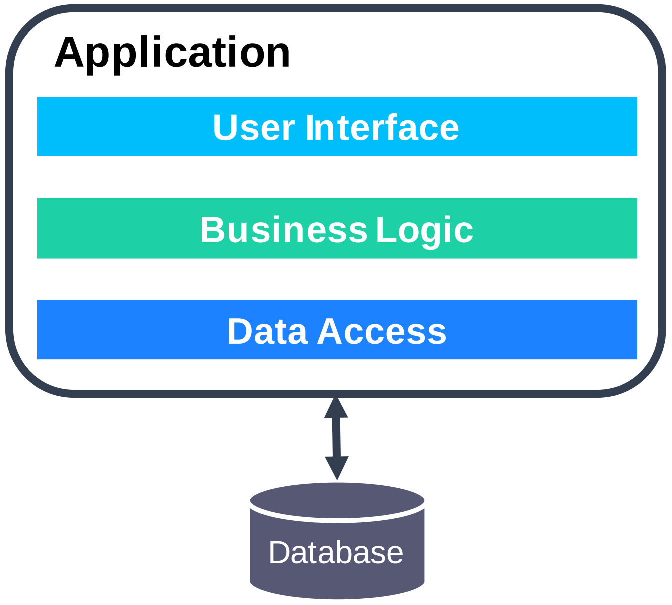
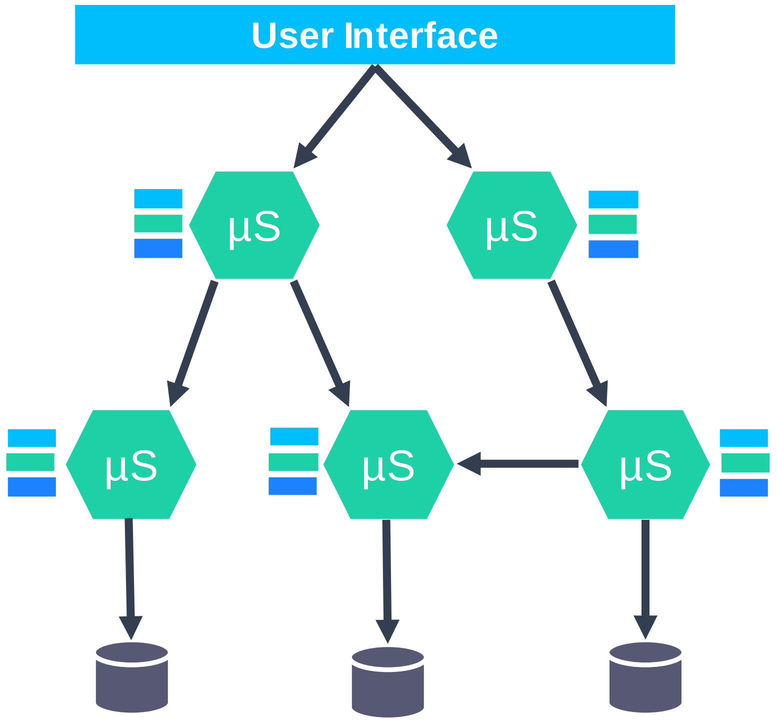
Microservice Architecture

Inter Process Communication

Overview

Problem Statement





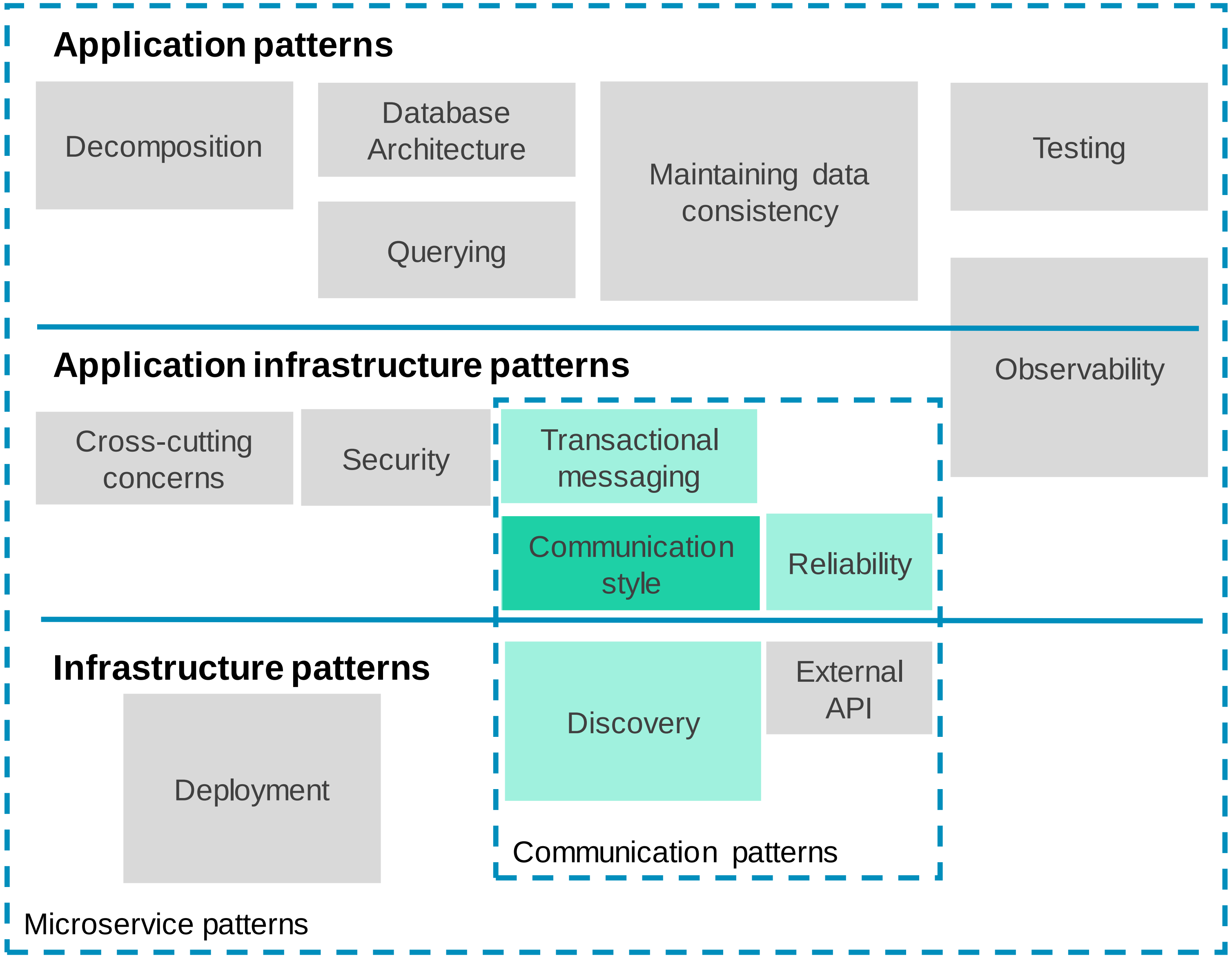
In a **monolithic** application, components invoke one another via **language-level method** or **function calls**.

A **microservices** application is a **distributed system** running on multiple machines. Each service instance is typically a **process**.

→ services must interact using an **inter-process communication (IPC)** mechanism.

Problem Statement

Problem areas to solve



By the end of this course, you will be able to

1. **Determine** how services interact.

2. **Specify** the appropriate message format

3. **Define** and **manage the evolution** of a service's API

1

Agenda

Overview of Inter Process Communication (IPC) in a microservice architecture

1. How services interact?
2. What are the possible message formats?
3. How to specify the API for each service?
4. How to manage APIs when they evolve?

Interaction styles

How services interact?

	One-to-One	One-to-Many
Synchronous	Request / Response	---
Asynchronous	Notification Request / Async Responses	Publish / Subscribe Publish / Async Responses

Interaction styles can be categorized along two dimensions:

- **One-to-one**: Each client request is processed by exactly one service instance.
- **One-to-many**: Each request is processed by multiple service instances.

- **Synchronous**: The client expects a timely response from the service and might even block while it is waiting.
- **Asynchronous**: The client doesn't block while waiting for a response, and the response, if any, isn't necessarily sent immediately.

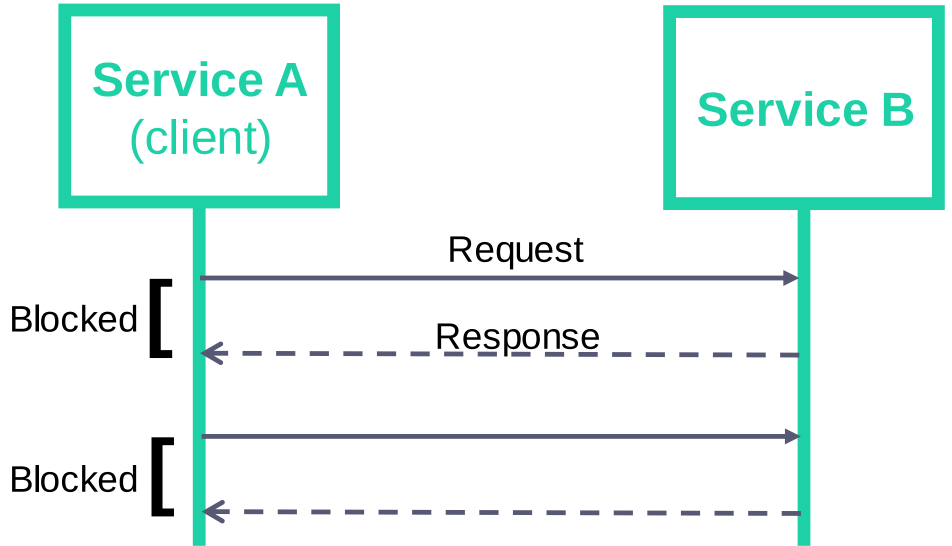
Interaction styles

How services interact?

	One-to-One	One-to-Many
Synchronous	Request / Response	---
Asynchronous	Notification Request / Async Responses	Publish / Subscribe Publish / Async Responses

One-to-One interactions

- **Request / Response:** A client makes a request to a service and waits for a response. The client expects the response to arrive in a timely fashion.



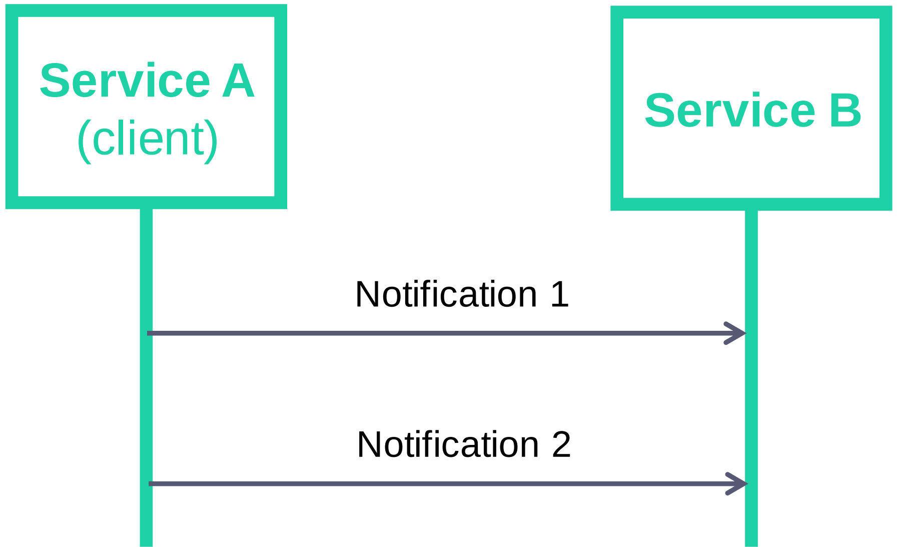
Interaction styles

How services interact?

	One-to-One	One-to-Many
Synchronous	Request / Response	---
Asynchronous	Notification Request / Async Responses	Publish / Subscribe Publish / Async Responses

One-to-One interactions

- **Notification (a.k.a. a one-way request):** A client sends a request to a service but no reply is expected or sent.



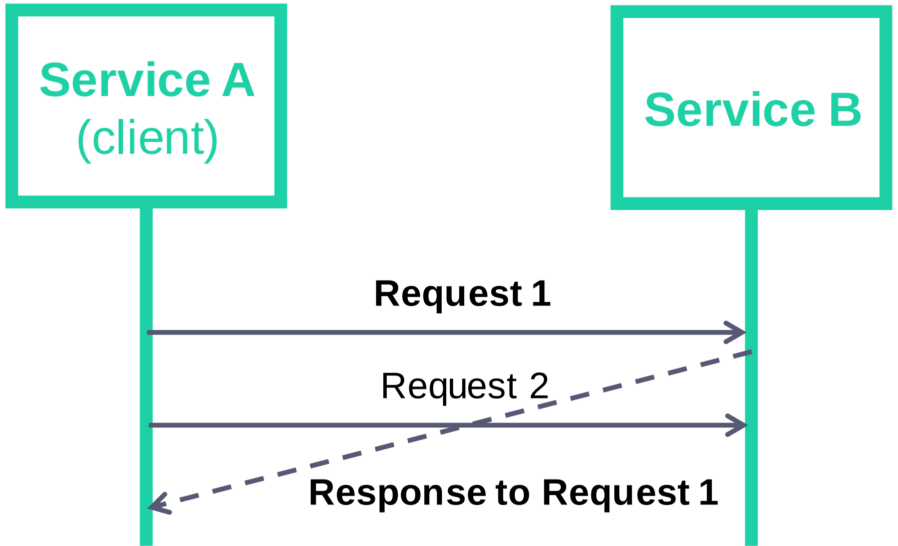
Interaction styles

How services interact?

	One-to-One	One-to-Many
Synchronous	Request / Response	---
Asynchronous	Notification Request / Async Responses	Publish / Subscribe Publish / Async Responses

One-to-One interactions

- **Request / Async Responses:** A client sends a request to a service, which replies asynchronously.



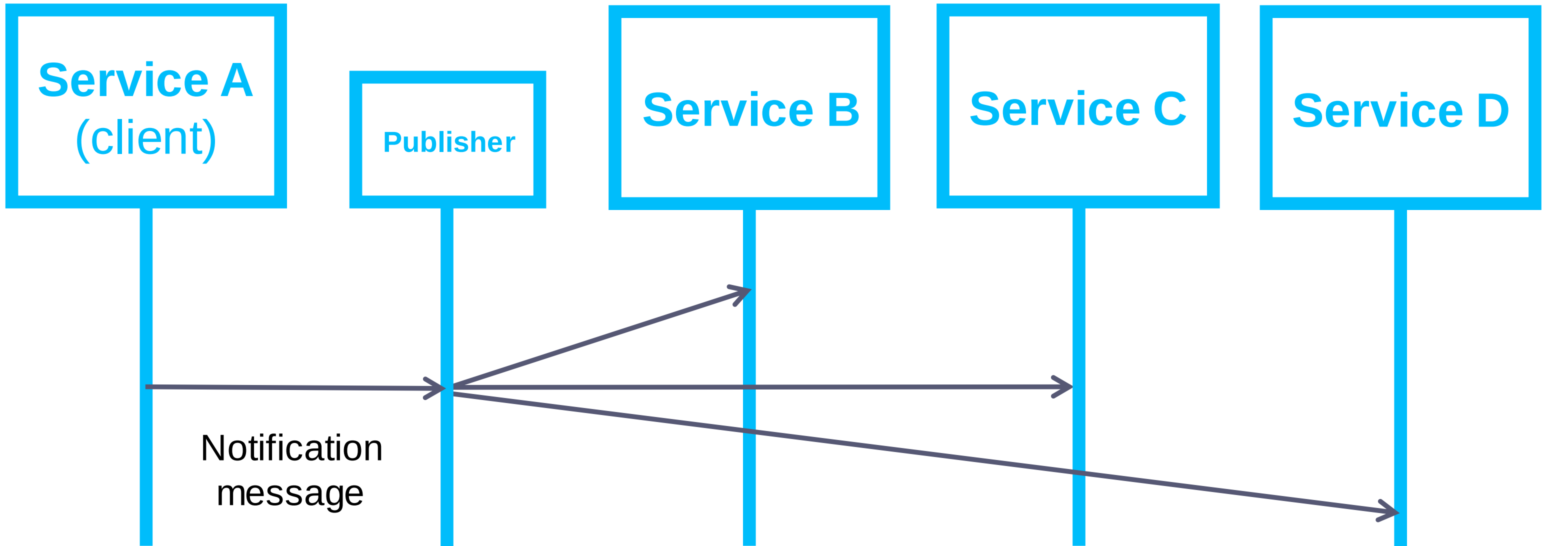
Interaction styles

How services interact?

	One-to-One	One-to-Many
Synchronous	Request / Response	---
Asynchronous	Notification Request / Async Responses	Publish / Subscribe Publish / Async Responses

One-to-Many interactions

- **Publish / Subscribe:** A client publishes a notification message, which is consumed by zero or more interested services.



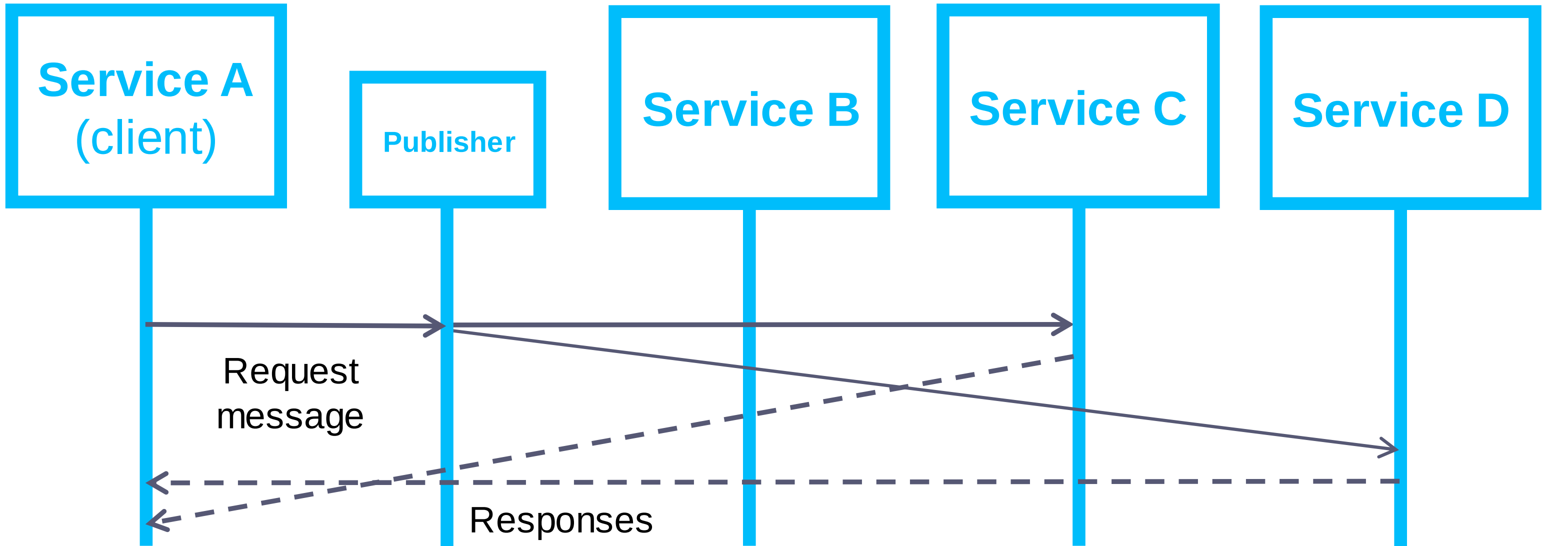
Interaction styles

How services interact?

	One-to-One	One-to-Many
Synchronous	Request / Response	---
Asynchronous	Notification Request / Async Responses	Publish / Subscribe Publish / Async Responses

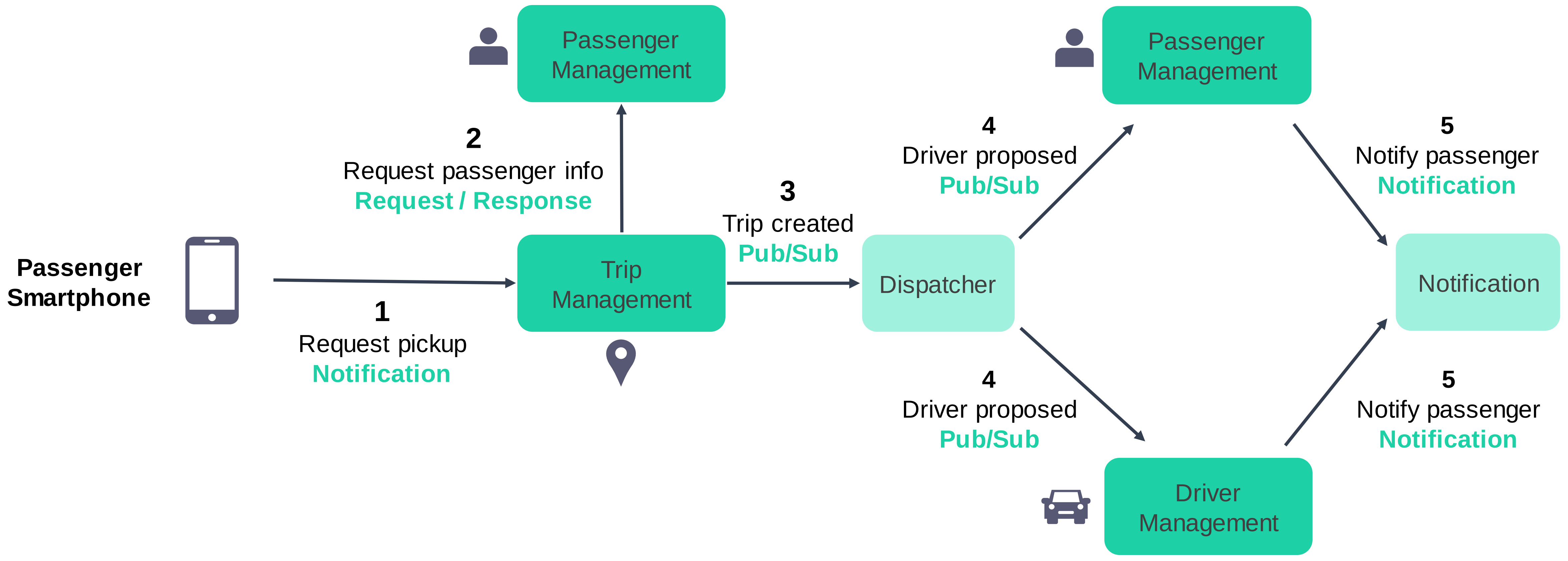
One-to-Many interactions

- **Publish / Async Responses:** A client publishes a request message, and then waits a certain amount of time for responses from interested services.



Interaction styles

Example: Taxi-hailing application



Each service typically uses a combination of these interaction styles.

Agenda

Overview of Inter Process Communication (IPC) in a microservice architecture

1. How services interact?
2. What are the possible message formats?
3. How to specify the API for each service?
4. How to manage APIs when they evolve?

Message formats

What are the possible message formats?

Text

XML **JSON**

⊕
- Human-readable
- Self-describing

⊖
- Verbose messages (especially XML)
- The overhead of parsing text

Binary

Protocol Buffers
Binary Thrift
Apache Avro

⊕
- More efficient than the text format
- Provide a typed IDL for defining the structure of the messages

3

Agenda

Overview of Inter Process Communication (IPC) in a microservice architecture

1. How services interact?
2. What are the possible message formats?
3. How to specify the API for each service?
4. How to manage APIs when they evolve?

Defining APIs

How to specify the API for each service?

A **service's API** is a **contract** between the **service** and its **clients**.

It's important to precisely define a service's API using an **interface definition language (IDL)**.

It's recommended to use an "***API-first approach***" to define services

- You begin the development of a service by writing the interface definition and reviewing it with the client developers.
- It is only after iterating on the API definition that you implement the service.

The nature of the API definition depends on the used IPC mechanism

- If you are using **messaging**, the API consists of the message **channels** the message **types**, and the message **formats**.
- If you are using **HTTP**, the API consists of the **URLs,** the HTTP **verbs** and the **request and response formats**.

4

Agenda

Overview of Inter Process Communication (IPC) in a microservice architecture

1. How services interact?
2. What are the possible message formats?
3. How to specify the API for each service?
4. How to manage APIs when they evolve?

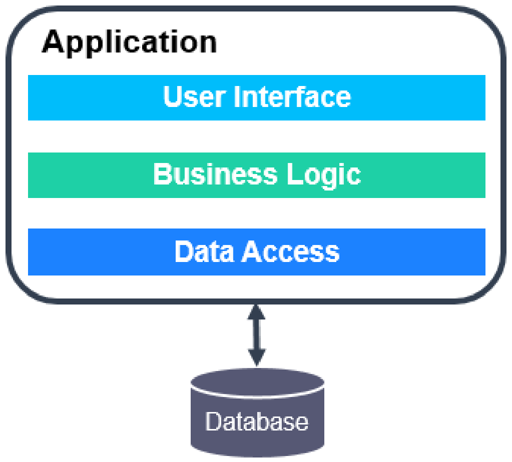
Evolving APIs

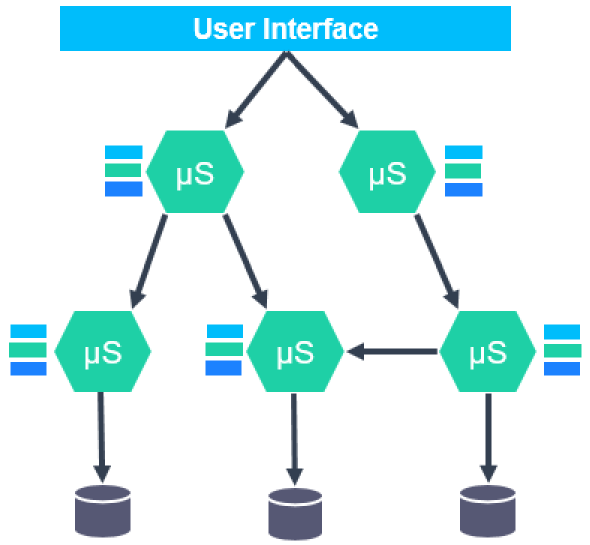
How to manage APIs when they evolve?

A service's API invariably changes over time

In a **monolithic** application it is usually straightforward to change the API and update all the callers.

In a **microservices**-based application it is a lot **more difficult**
(You usually cannot force all clients to upgrade in lock step with the service)





Evolving APIs

How to manage APIs when they evolve?

How to handle an API change depends on the size of the change

Minor changes

Are backward compatible with the previous version

Example : adding attributes to requests or responses

It makes sense to design clients and services so that they respect the "**robustness principle**"

Major changes

Incompatible changes to an API

Since you can't force clients to upgrade immediately, a service must support older versions of the API for some period.

Solution 1: embed the version number in the URL (if using REST)
Solution 2: deploy different instances that each handle a particular version

IPC Overview
Key Takeaways

- The microservice architecture is a **distributed architecture**, so **inter process communication** plays a key role.
- All the possible **interaction styles** between services can be categorized along two dimensions: :

  “**one-to-one**” or “**one-to-many**” interaction

  and

  “**synchronous”** or “**asynchronous”** communication
- He resulting interaction styles are then:
  - **Request / Response**
  - **Notification**
  - **Request / Asynchronous Responses**
  - **Publish / Subscribe**
  - **Publish / Asynchronous Responses**
- It’s important to consider a **cross-language message format** between **text format** or **binary format**.
- It’s essential to carefully **define** and **manage the evolution** of a service’s **API**.

Questions are welcome