

Unit 5: Design of Web Services (2/2)

RPC API. Message API. Flexible Query API.

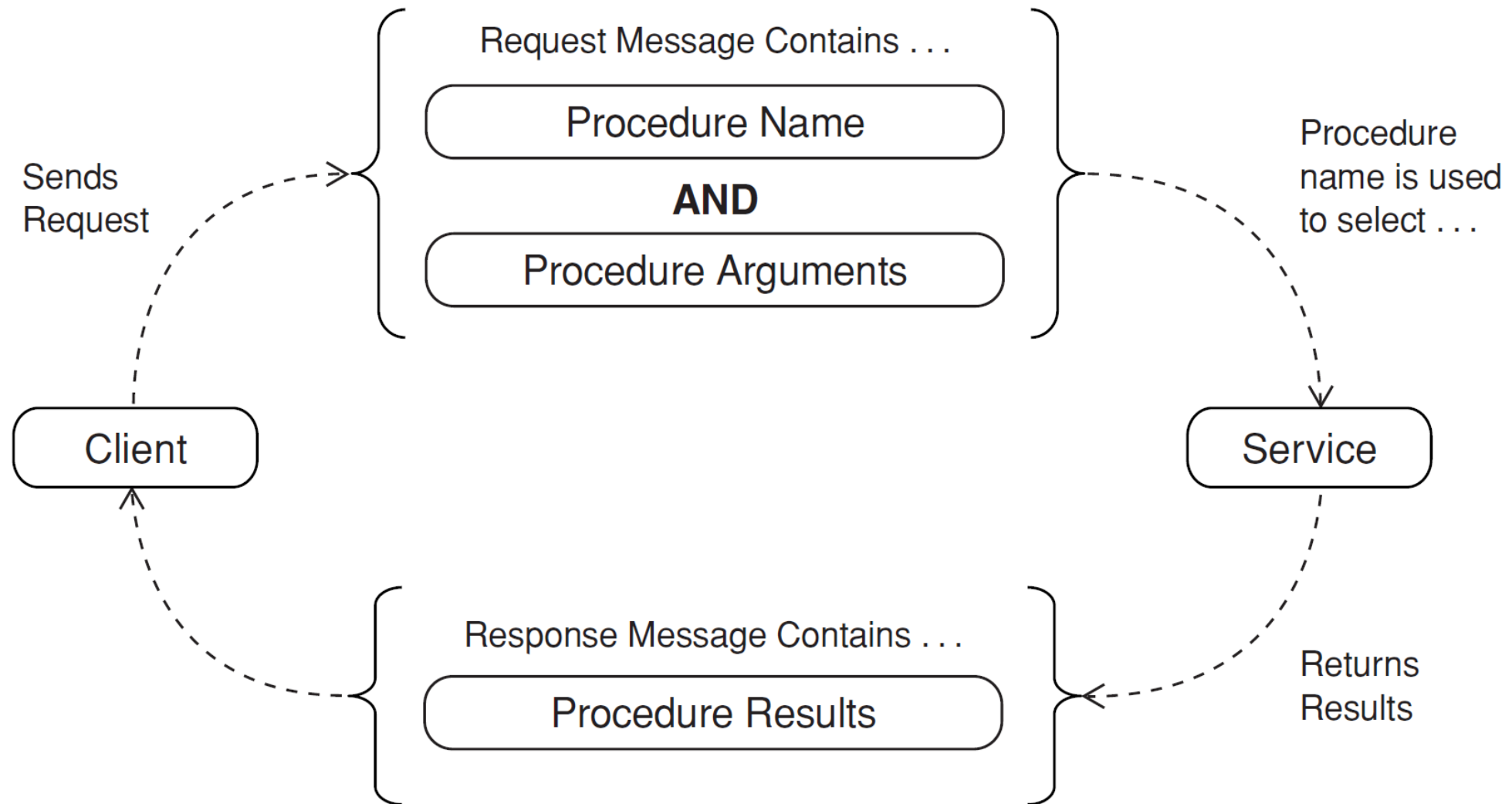
Design of Web Services

- ▶ Introduction to Web Services
- ▶ Resource API with REST: RESTful Web Services
- ▶ **RPC API**
- ▶ Message API
- ▶ Flexible Query API

Web Service API Styles: RPC API

Pattern Name	Problem	Description	Technology
<i>RPC API</i>	How can clients execute remote procedures over HTTP?	Define messages that identify the remote procedures to execute and also include a fixed set of elements that map directly into the parameters of remote procedures	<ul style="list-style-type: none">• SOAP• gRPC

RPC API



* Extracted from: Robert Daigneau. Service Design Patterns. Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Addison Wesley, 2012

Designing a RPC API: The Roadmap

1. Identify the services and their operations
2. Define the type of interaction between the client and the service
3. Apply the DTO pattern when defining the operation parameters
4. Describe the service contracts

Identify services & operations

- ▶ Study the sources from which services must be derived
 - ◆ BPM, conceptual models, others
- ▶ Identify services and classify them into:
 - ◆ **Application services** represent technology and application logic (utility and wrapper services)
 - ◆ **Entity-centric services** encapsulate business entities
 - ◆ **Task-centric services** encapsulate logic specific to a task or business process
 - ◆ **Orchestration services** represent coordination of other services
- ▶ Apply service orientation design principles:
 - ◆ Definition of a service contract, granularity for reuse, low coupling, stateless services

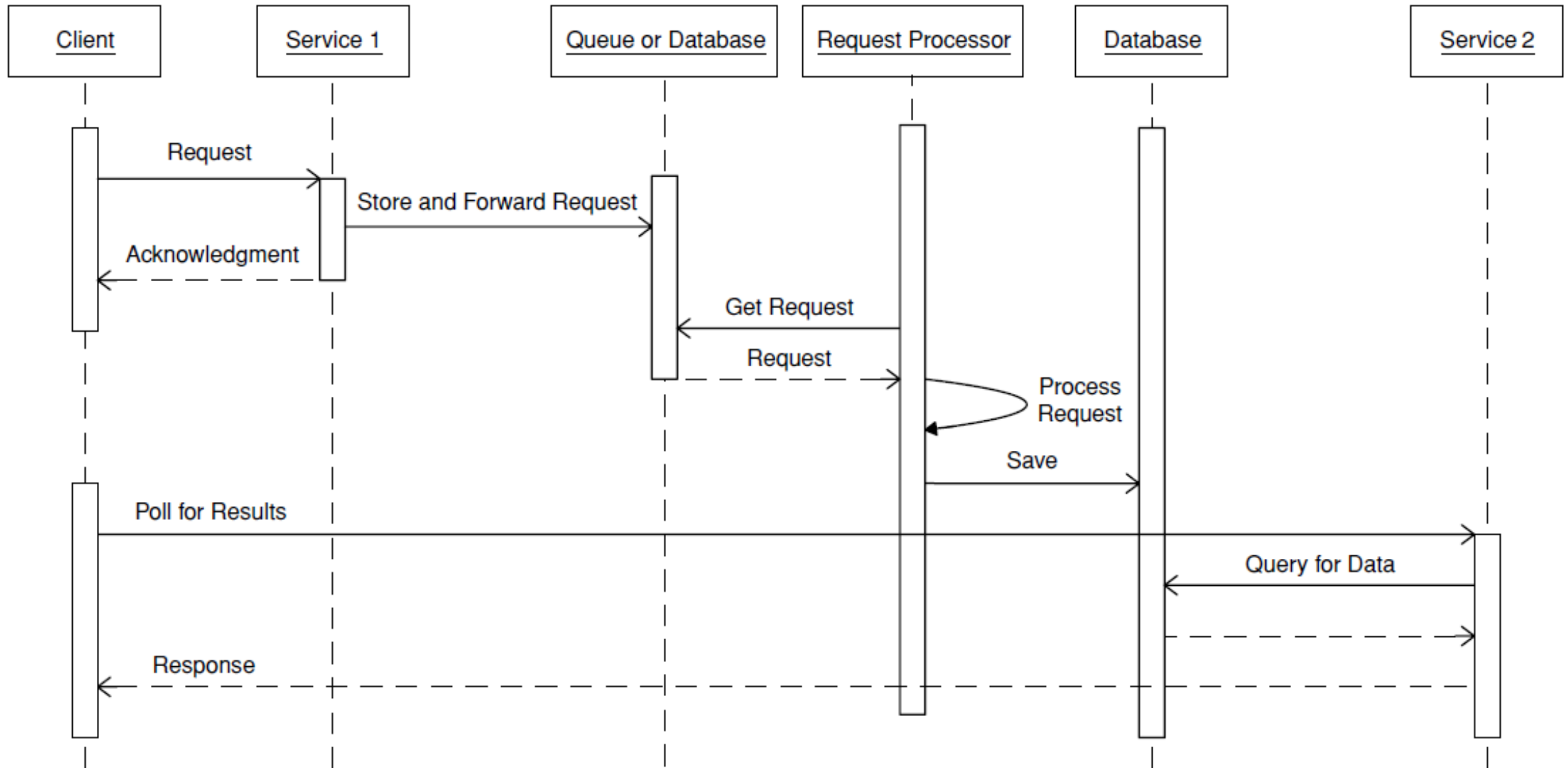
Defining the Client/Service Interaction

Pattern Name	Problem	Description
<i>Request/Response</i>	What's the simplest way for a web service to process a request and provide a result?	Process requests when they're received and return results over the same client connection.
<i>Request/Acknowledge</i>	How can a web service safeguard systems from spikes in request load and ensure that requests are processed even when the underlying systems are unavailable?	When a service receives a request, forward it to a background process, then return an acknowledgment containing a unique request identifier.

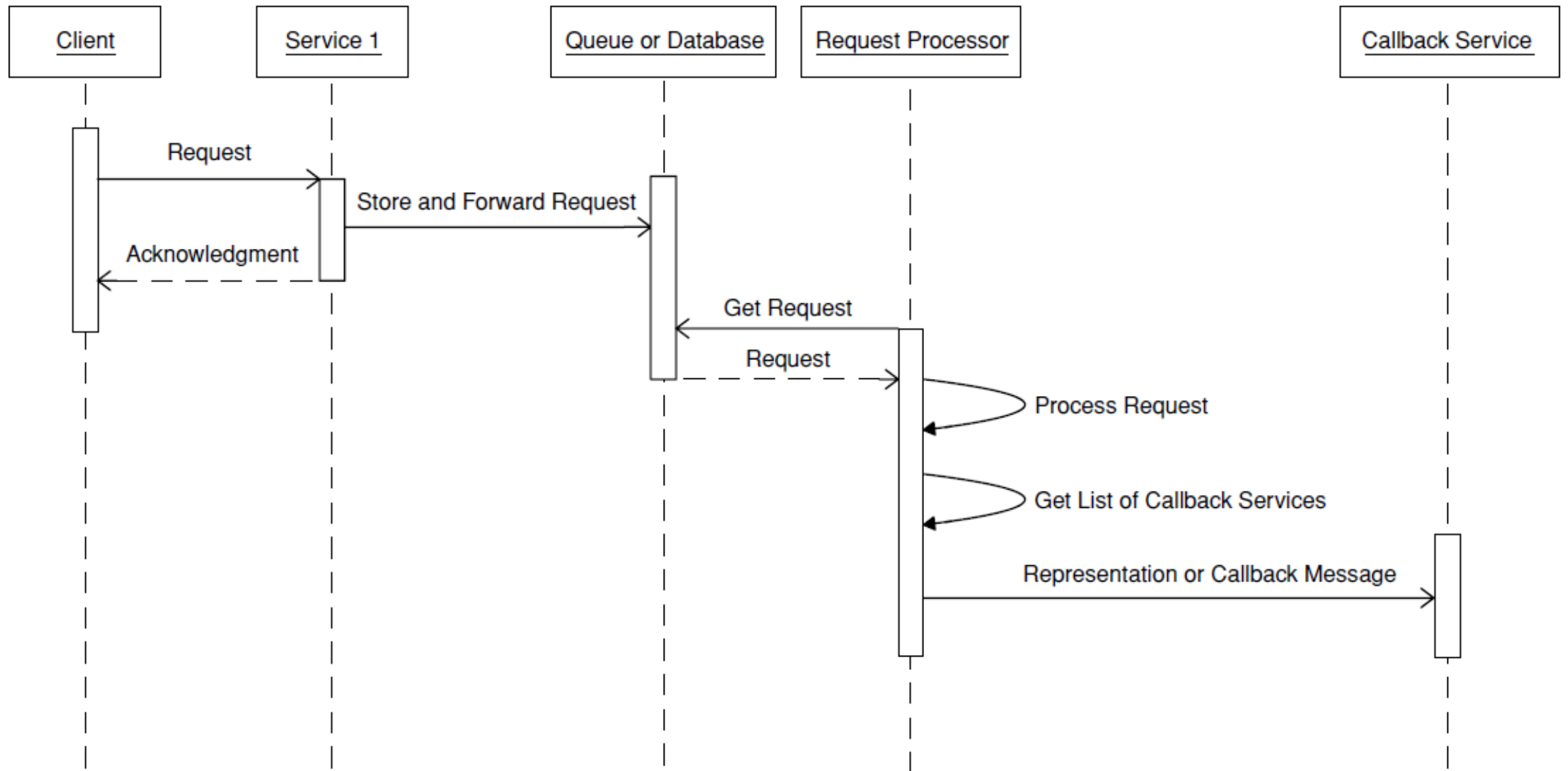
Request/Response Pattern: Issues

- ▶ High temporal coupling
 - ◆ Requests must be processed as soon as they are received
 - ◆ Workload of services. Availability and scalability issues
- ▶ Client-side blocking
 - ◆ By default, clients block and wait for responses.

Request/Acknowledge Pattern: Poll Variation



Request/Acknowledge Pattern: Callback Variation



DTO Pattern

▶ Context

- ◆ Systems need to communicate data to services in an efficient manner

▶ Problem

- ◆ To minimize the number of calls in the system, each call has to carry more information in the parameters and in the return value

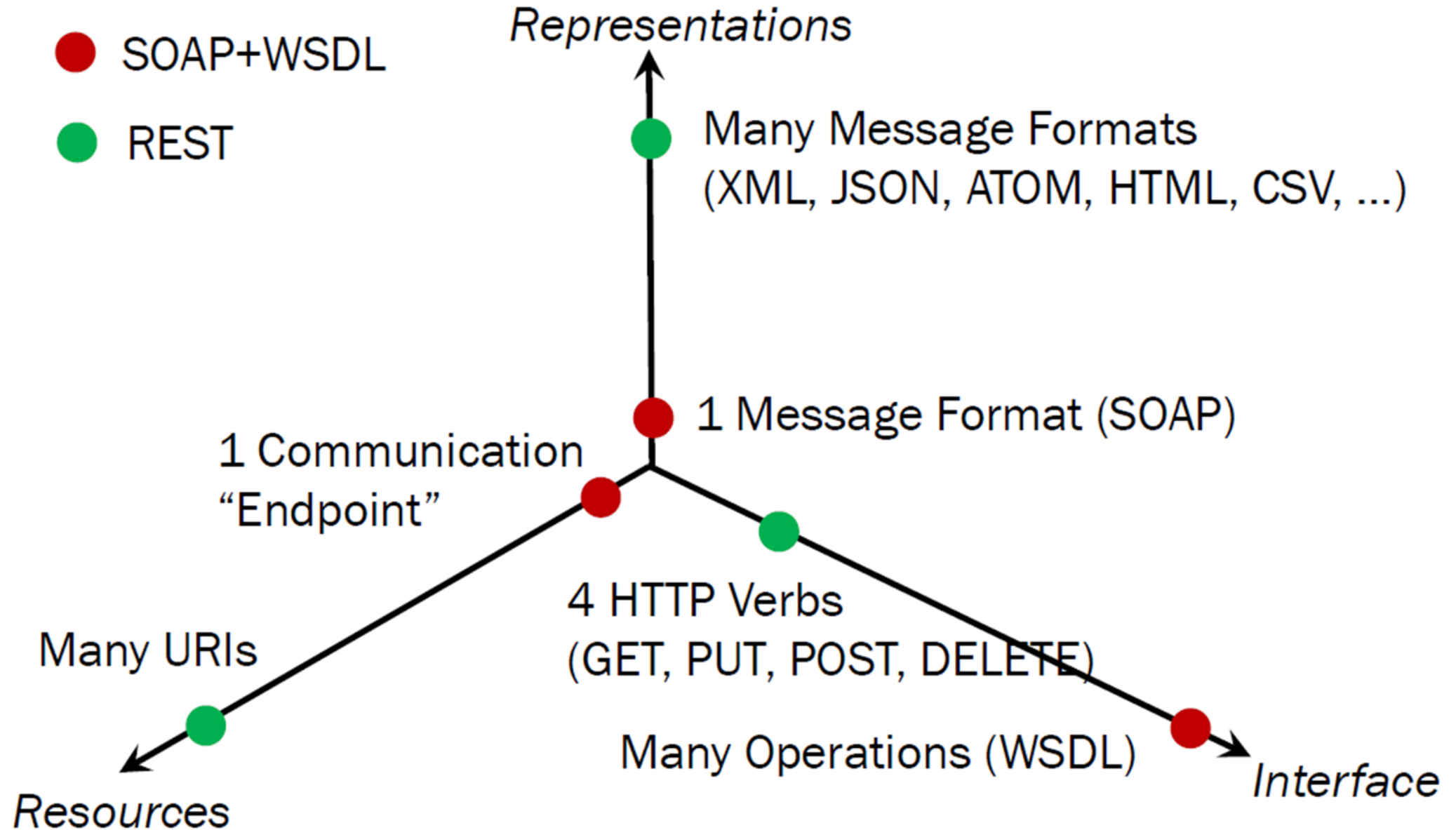
▶ Solution

- ◆ We define (use) a new data group that contains all the data that has to be passed as parameter or result. The objects that are these kind of groups are called Data Transfer Objects (DTO)

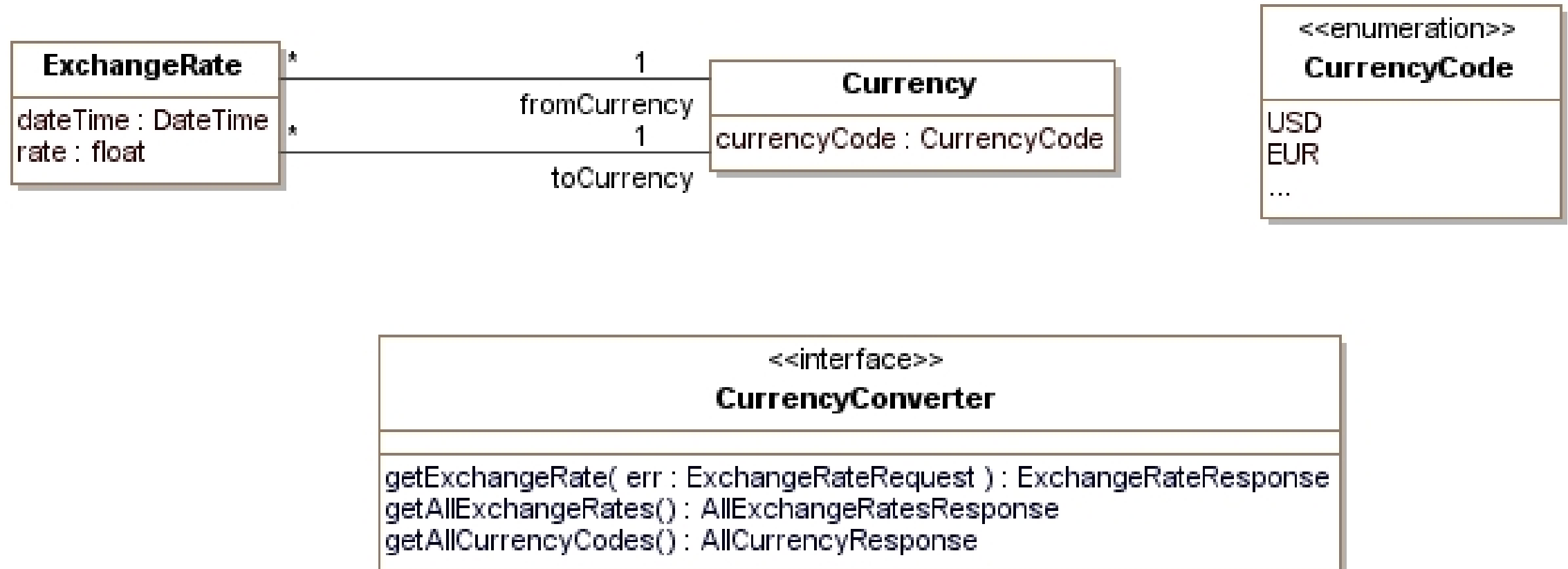
Describing The Service Contract

- ▶ Formal contracts should be provided to describe services and to define terms of information exchange
- ▶ Service contracts provide the definition of:
 - ◆ Service endpoint describes the point of contact for a service by stating the physical location of the service
 - ◆ Service operations
 - ◆ Input and output message supported by each operation
 - ◆ Rules and characteristics of the service and its operations
- ▶ In the case of SOAP services, WSDL is used to describe service contracts

SOAP+WSDL vs. RESTful



Example



RPC API + DTO + Request/Response

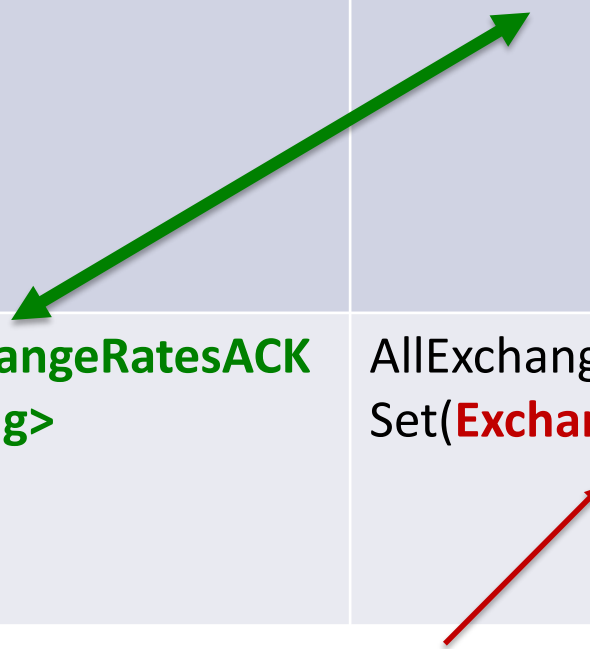
Name Operation	Input	Output	Description
getExchangeRate	ExchangeRateRequest = <from: CurrencyCode, to: CurrencyCode>	ExchangeRateResponse = <from: CurrencyCode, to: CurrencyCode, rate: Float, date: DateTime>	Returns the exchange rate between two given currencies.
getAllExchangeRates		AllExchangeRatesResponse = Set(ExchangeRateResponse)	Returns the exchange rates between every two currencies.
getAllCurrencyCodes		AllCurrencyResponse = Set (currencyCode:CurrencyCode)	Returns all the currency codes

DTOs

RPC API + DTO + Request/Acknowledge::Poll

Let us suppose that the operation **getAllExchangeRates** takes some time to be processed.

Name Operation	Input	Output	Description
getAllExchangeRates		getAllExchangeRatesACK = <id: String>	Returns the acknowledgment that the request to obtain the exchange rates between every two currencies will be processed.
getResult	getAllExchangeRatesACK = <id: String>	AllExchangeRatesResponse = Set(ExchangeRateResponse)	Returns the result of the previous request



DTO

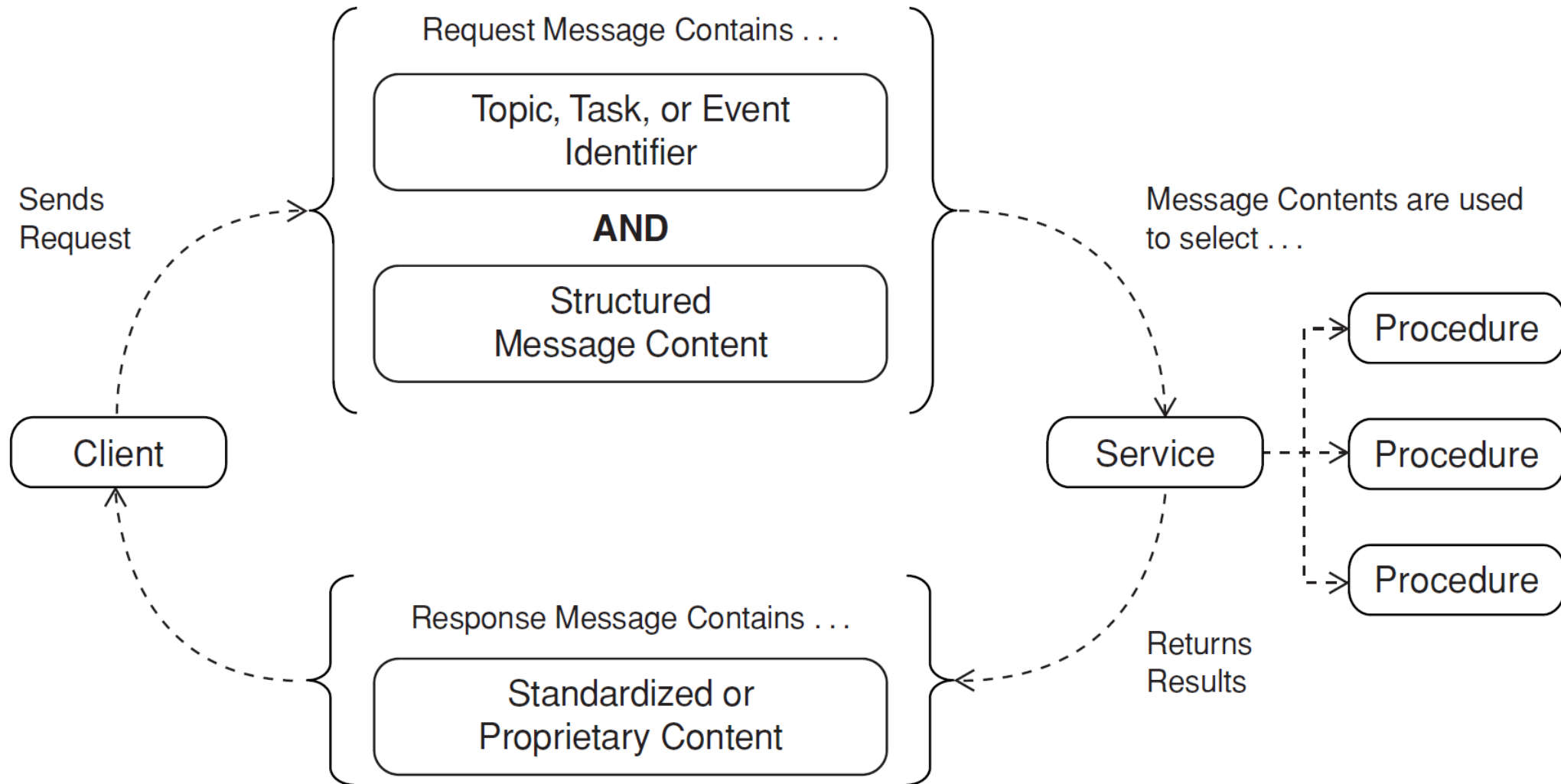
Design of Web Services

- ▶ Introduction to Web Services
- ▶ Resource API with REST: RESTful Web Services
- ▶ RPC API
- ▶ Message API
- ▶ Flexible Query API

Web Service API Styles: Message API

Pattern Name	Problem	Description	Technology
<i>Message API</i>	How can clients send commands, notifications, or other information to remote systems over HTTP while avoiding direct coupling to remote procedures?	Define messages that are not derived from the signatures of remote procedures. These messages may carry information on specific topics, tasks to execute, and events	<ul style="list-style-type: none">• AMQP• MQTT• ...

Message API

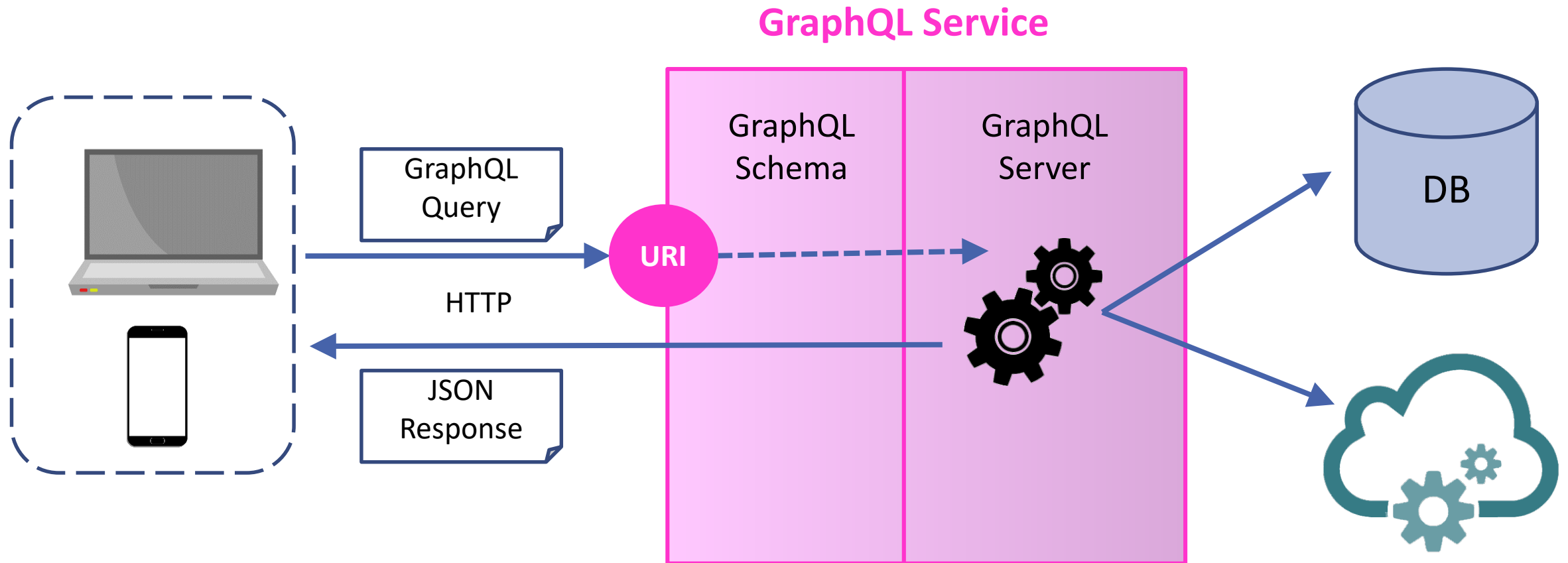


* Extracted from: Robert Daigneau. Service Design Patterns. Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Addison Wesley, 2012

Web Service API Styles: Flexible Query API

Pattern Name	Problem	Description	Technology
<i>Flexible Query API</i>	How can clients query or manipulate data with high flexibility while minimizing over-fetching and under-fetching of data?	Allow clients to specify the structure of the response data by defining queries in a strongly typed schema. The server resolves these queries based on its schema definition and returns data in a standardized format.	<ul style="list-style-type: none">• GraphQL

GraphQL



GraphQL: Example

GraphQL Query

```
getfilm (id: ".../film/100")
{
  title
  date
  director {
    director_name
  }
  genre {
    film_genre_name
  }
}
```

GraphQL Schema

```
type film {
  genre : [film_genre]
  director : director
  title : String
  actor : [actor]
  date : String
  country : country
  ... }

type director {
  director_name : String
  ... }

type film_genre {
  film_genre_name : String
  ... }

type Query {
  allfilms: [film]
  getfilm(id: String!): film }
```

JSON Response

```
{
  "data": {
    "getfilm": {
      "title": "Disraeli",
      "date": "1929",
      "director": {
        "director_name": "Alfred
          Green"
      },
      "genre": [
        {
          "film_genre_name": "Indie"
        },
        {
          "film_genre_name": "Biopic"
        },
        {
          "film_genre_name": "Drama"
        }
      ]
    }
  }
}
```

Jeff Bezzos' API Mandate (circa 2002)

- ▶ All teams will henceforth expose their data and functionality through service interfaces.
- ▶ Teams must communicate with each other through these interfaces.
- ▶ There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- ▶ It doesn't matter what technology you use.
- ▶ All service interfaces, without exception, must be designed from the ground up to be externalize-able. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- ▶ The mandate closed with:

Anyone who doesn't do this will be fired. Thank you; have a nice day!

References

- ▶ Robert Daigneau. **Service Design Patterns. Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services.** Addison Wesley, 2012
- ▶ https://www.w3schools.com/xml/xml_services.asp