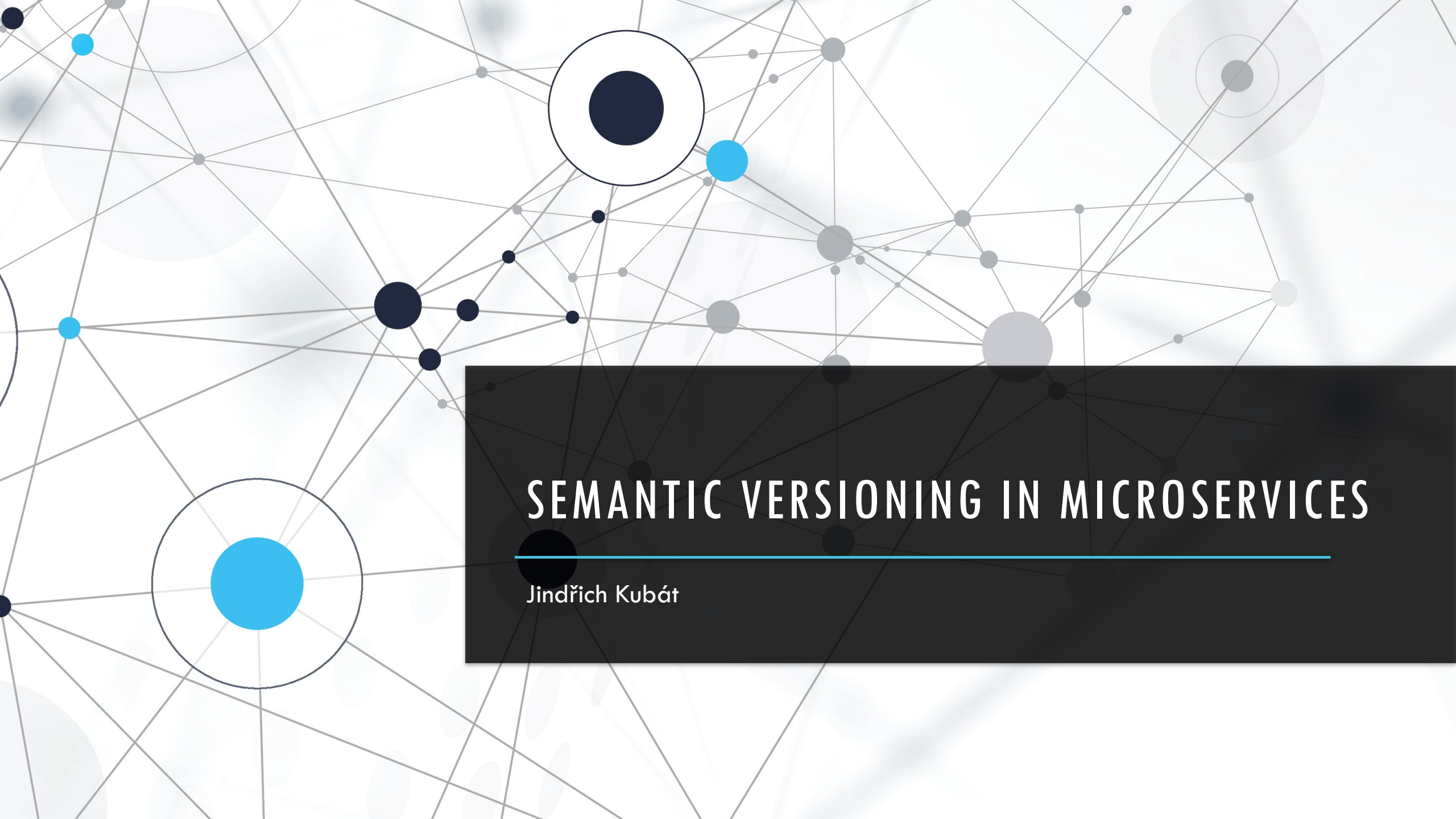


# SEMANTIC VERSIONING IN MICROSERVICES

Jindřich Kubát

# ABOUT ME

10+ years of experience as a developer

5+ years of experience as development manager

Solution architect

DevOps & Agile enthusiast

Microservices lector

Head of Development CoE at #komercka


linkedin.com/in/jindrich-kubat
@Techitouch

# AGENDA

Why we need API versioning

Semantic versioning

Types of API versioning

KB API design proposal

Demo

Conclusion

REST

SOAP

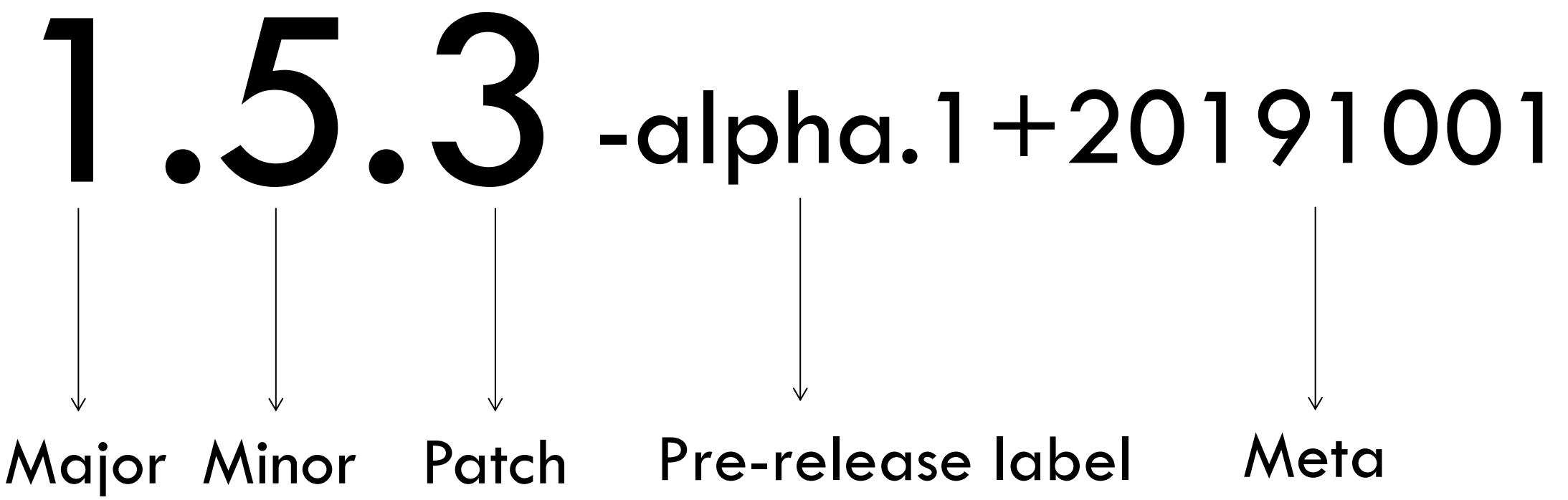
AMQP

gRPC

# TYPES OF COMMUNICATIONS

# WHY WE NEED VERSIONING?

# SEMANTIC VERSIONING 2.0




https://semver.org/

# VERSIONING IN THE URI

**/accounts/v1/{id}**

**Advantages:**

- Clear visibility of used version
- Ability to version an entire resource hierarchy or branch
- Enables automated navigation or discovery of resources

**Disadvantage:**

- Every change in the URI will change resource or location.
- Difficult to identify which version of your API is the currently supported version
- Can be hard to read for multiple hiearchy nodes /accounts/v1/address/v2/{id}

# VERSIONING IN THE HEADER

**X-Version:1.1**

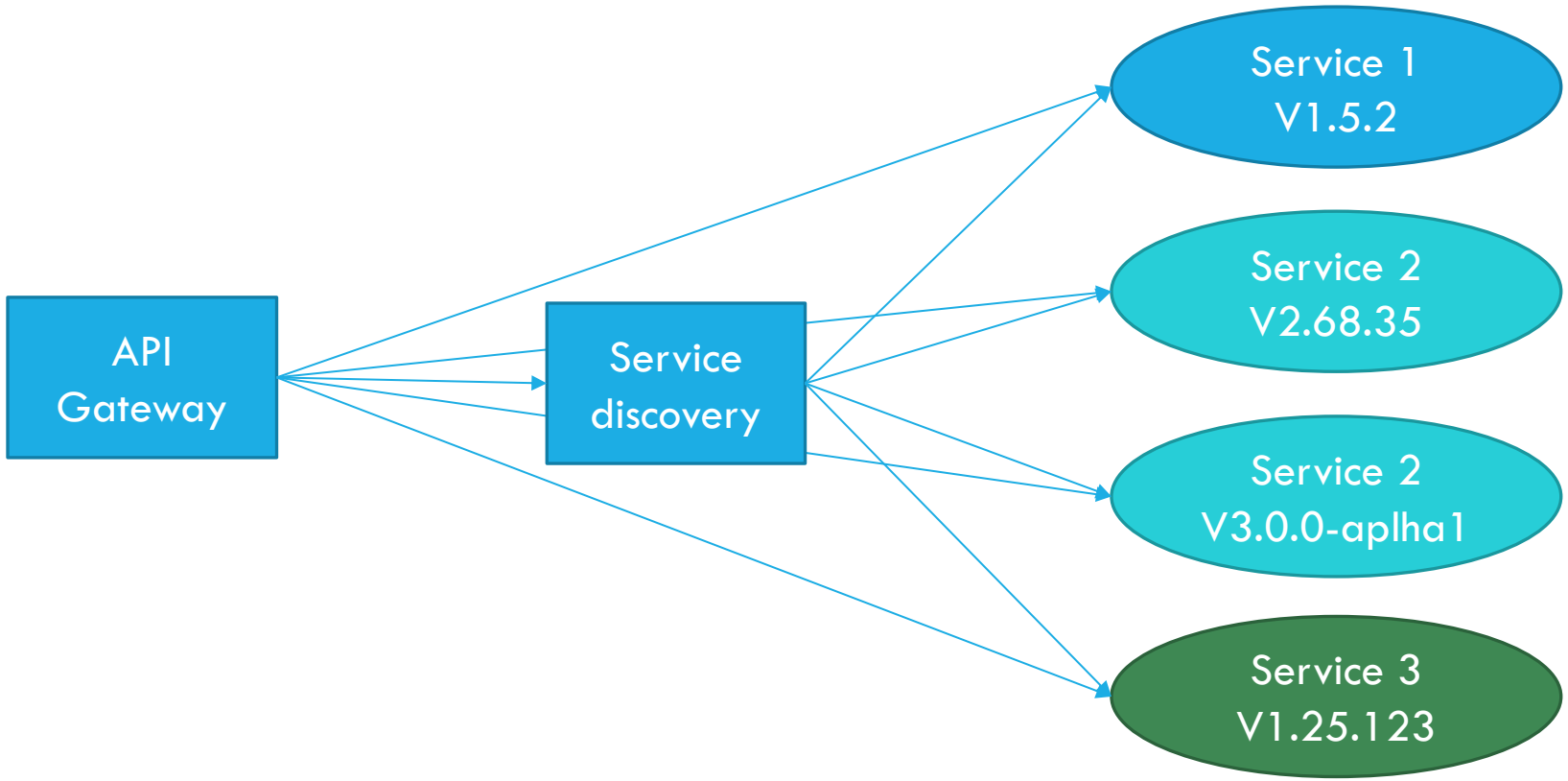
**Advantages:**

- In some cases, it may be much easier, if URI doesn't change.
- Parsing version in the header may be easier for integration platforms like ESB

**Disadvantage:**

- Hard to see resource version in logs
- Works only for clients that know how to parse version from header

# TYPICAL USAGE



# CONTENT NEGOTIATION

Based on Content negotation principles

The client can request specific resource according to „accept“ in the header, eg.

```
accept: application/accounts+json.v1.1.0
```

The server sends a response that is compatible with the requested resource and media format

```
{
  ver: 1.1.0
  name: „accounts“
  data: [...]
}
```

*Some kind of /info endpoint with all supported versions is a must*

# HYPERMEDIA AS THE ENGINE OF APPLICATION STATE - **HATEOAS**

**Request:**

**GET** /accounts/12345/ HTTP/1.1 Host: **api.kb.cz**
Accept: application/vnd.acme.account+json.v1.0.0

**Response:**

**HTTP**/1.1 200 **OK** Content-Type: application/vnd.acme.account+json.v1.0.0 Content-Length: ...

```
{
  "account":{
    "account_number":12345,
    "balance":{
      "currency":"czk",
      "value ": 1000.00
     },
     "links": {
          "deposit": “/accounts/12345/deposit",
          "withdraw": "/accounts/12345/withdraw",
          "transfer": "/accounts/12345/transfer",
          "close": "/accounts/12345/close“
      }
  }
}
```

# HYPERMEDIA AS THE ENGINE OF APPLICATION STATE - **HATEOAS**

Let's withdraw some money, f.e 1000Kč ;)

**Request:**

**GET** /accounts/12345/ HTTP/1.1 Host: **api.kb.cz**
Accept: application/vnd.acme.account+json.v1.0.0

**Response:**

**HTTP**/1.1 200 **OK** Content-Type: application/vnd.acme.account+json.v1.0.0 Content-Length: ...

```
{
   "account":{
      "account_number":12345,
      "balance":{
         "currency":"czk",
         "value ": 0.00
       },
       "links": {
             "deposit": "/accounts/12345/deposit",
             "close": "/accounts/12345/close"
       }
   }
}
```

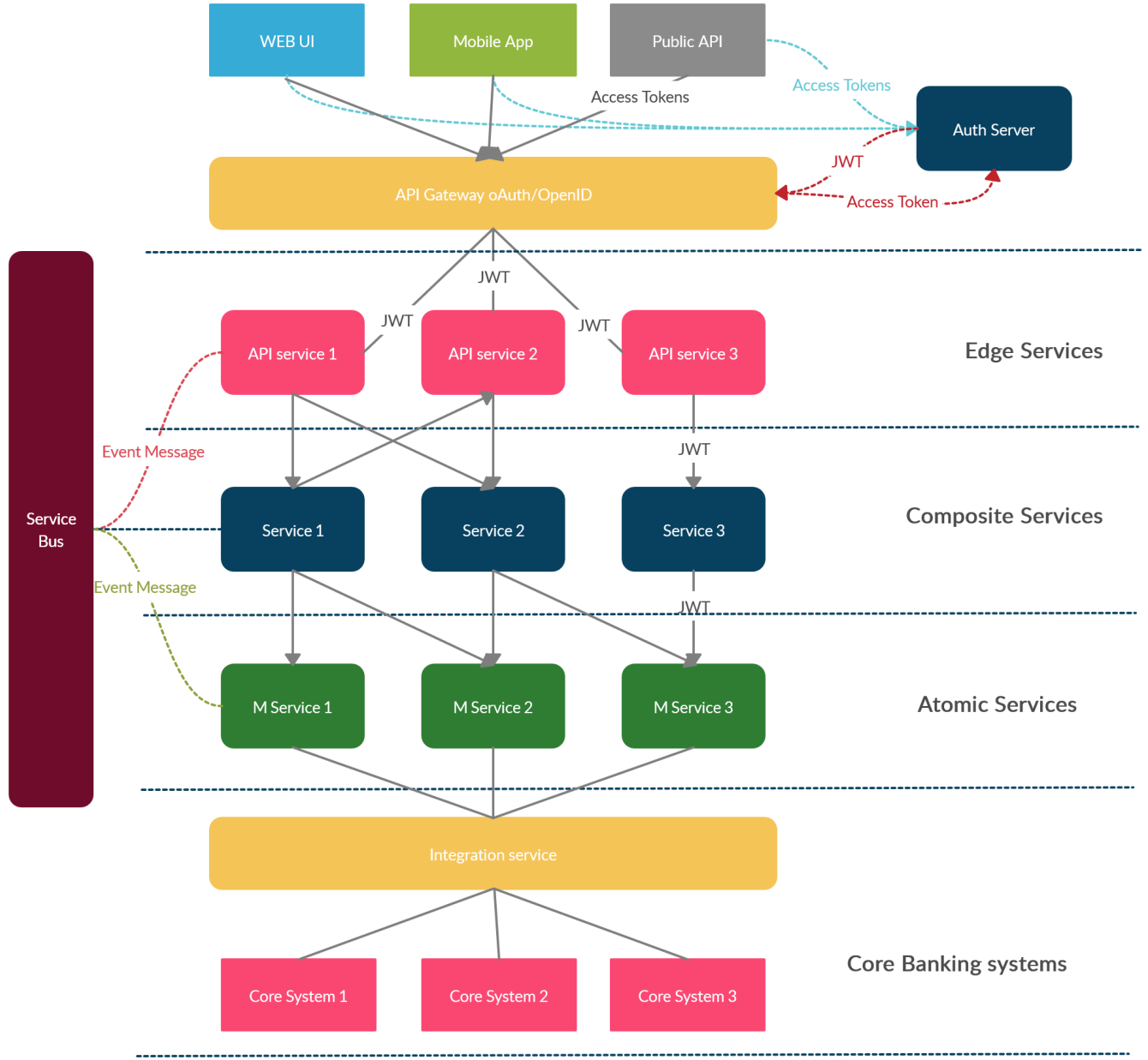
# GRAPHQL

- GraphQL can be used as a main protocol among microservices
- Like versioning in the header, change of the version won't change location
- Can be easier for some frontend applications
- Consider if you change your api frequently

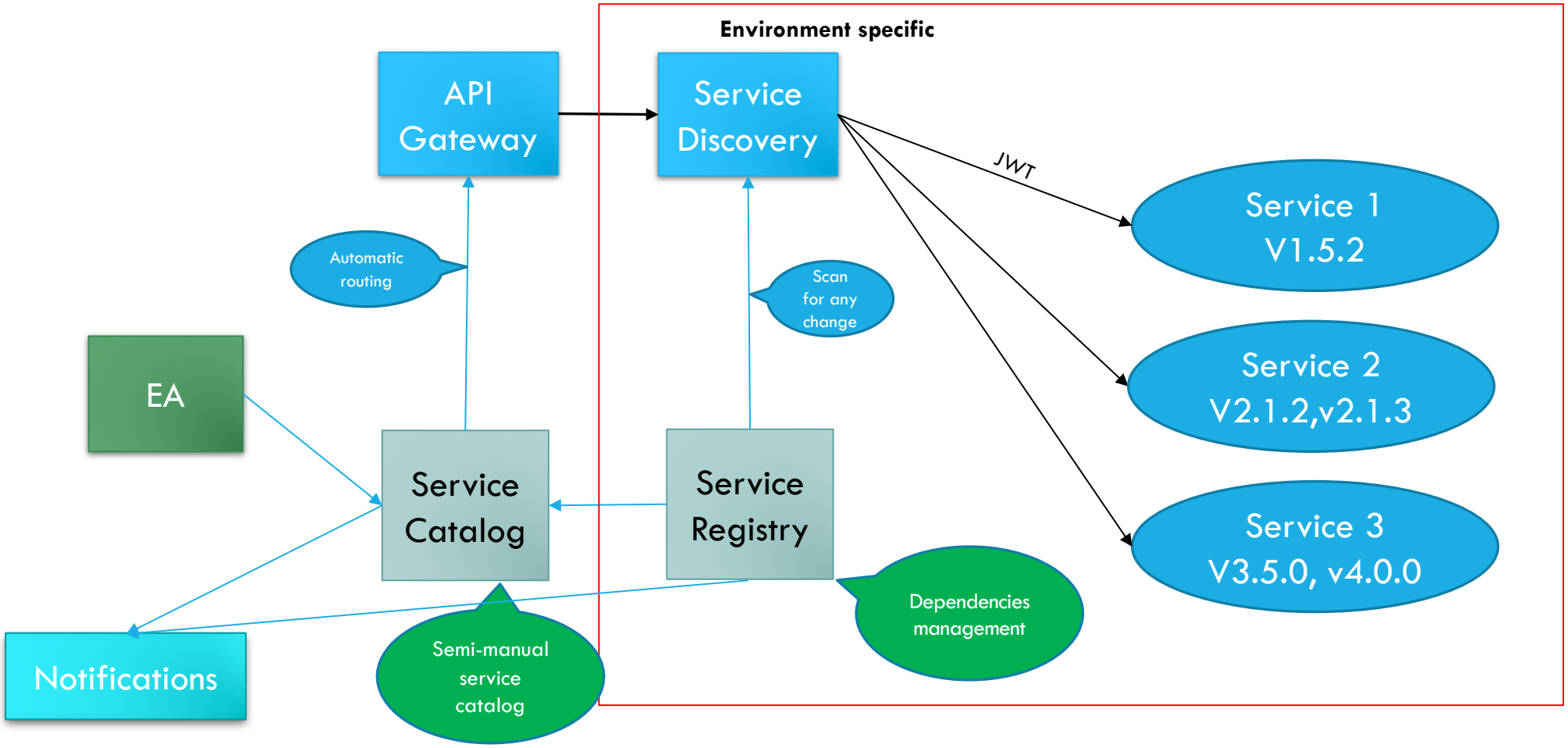
**Disadvantages:**

- It's GraphQL
- Difficult error handling
- No caching options
- Bigger complexity

# API PLATFORM DESIGN



# API WORKFLOW



# DEMO

# Demo time

# CONCLUSION

1. Package version may not correspond to API version
2. Consider how to deal with multi versions
   - New package image with different port
   - One package with multi version support
3. API Platform needs good automatic routing mechanism – API Gateway + Service discovery
4. Service catalog and Service Registry is your friend
5. Build /info endpoint with list of all supported versions