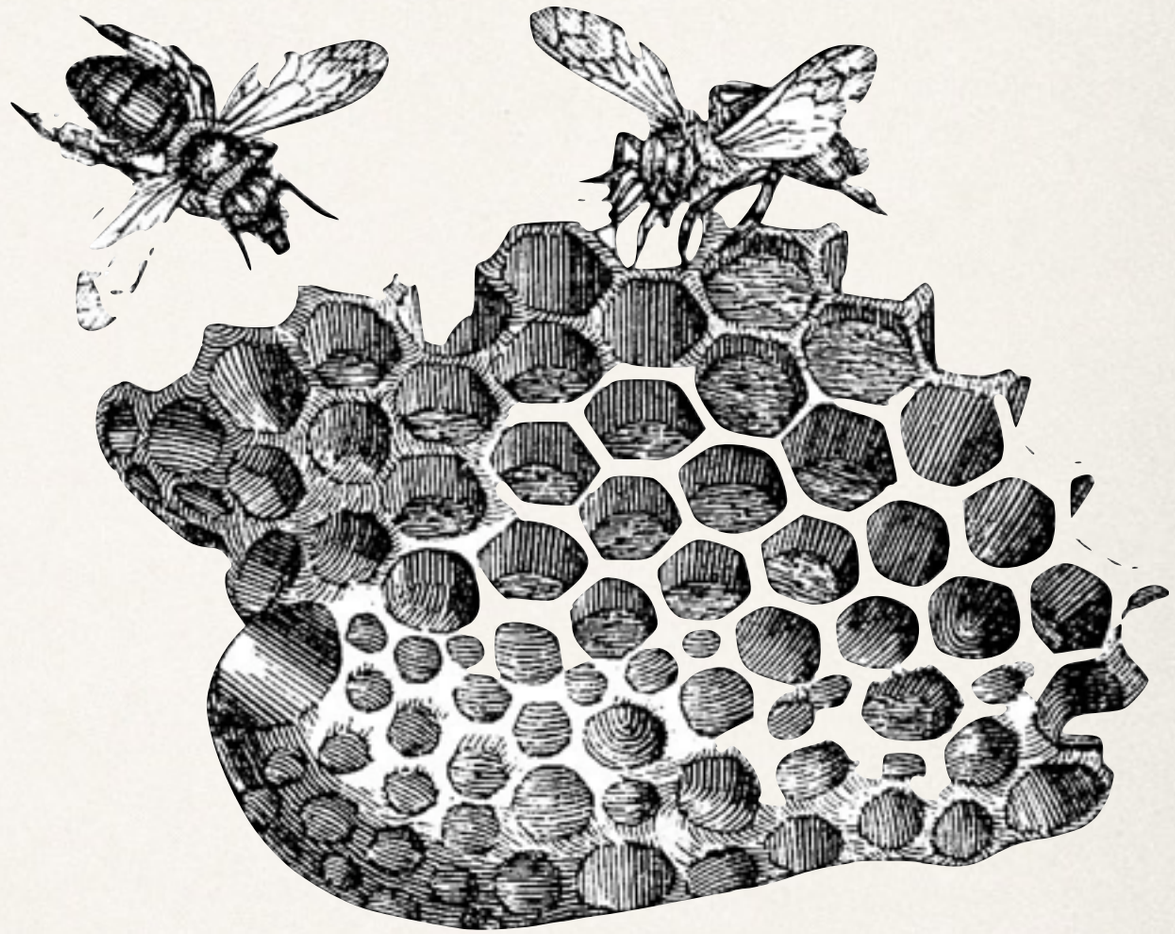
Jolie Microservices and Choreographies

Saverio Giallorenzo

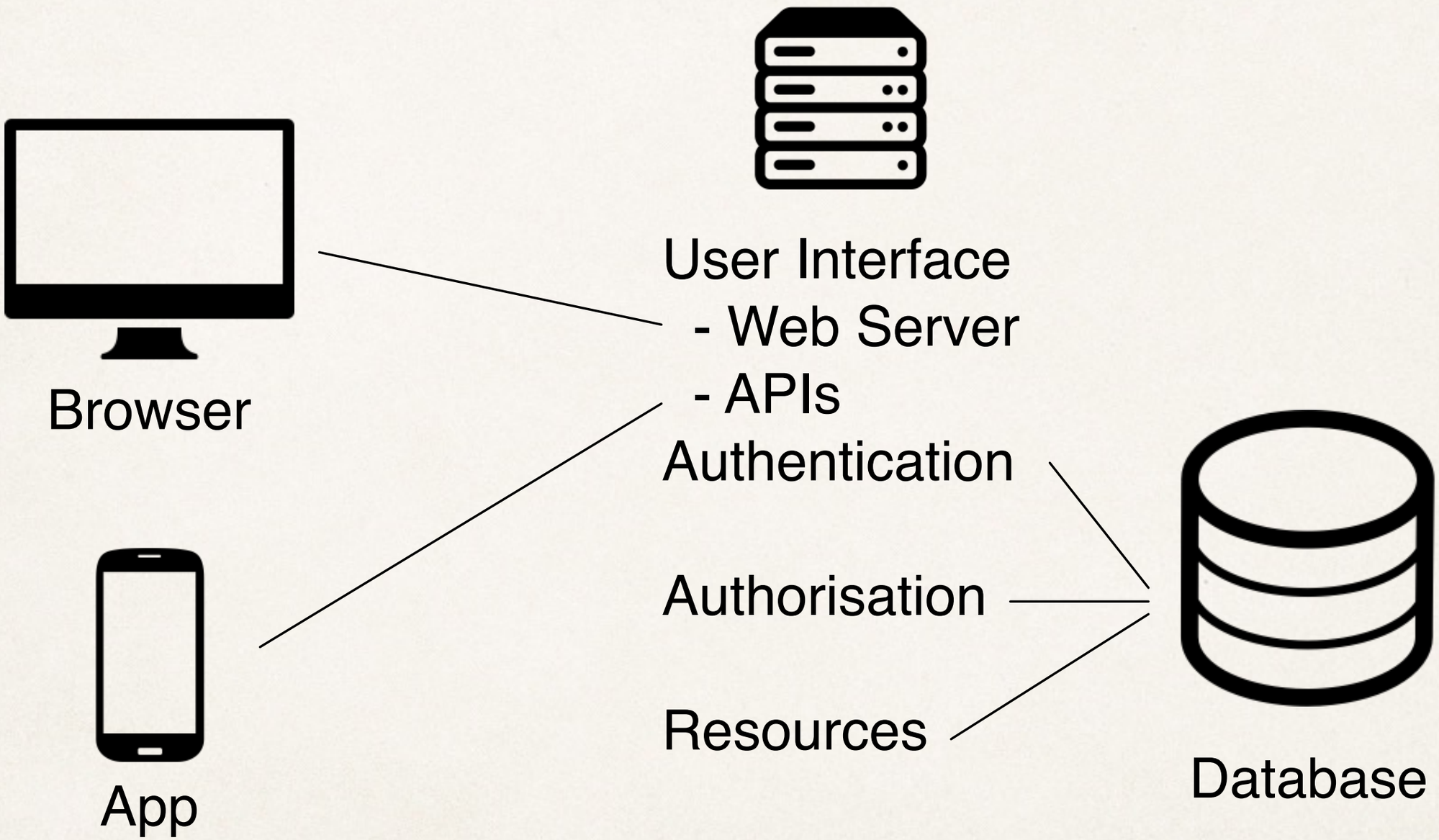
saverio.giallorenzo@gmail.com | *DISI at Unibo* | *Sophia Antipolis* | *Workshop on Programming Languages*

Microservices

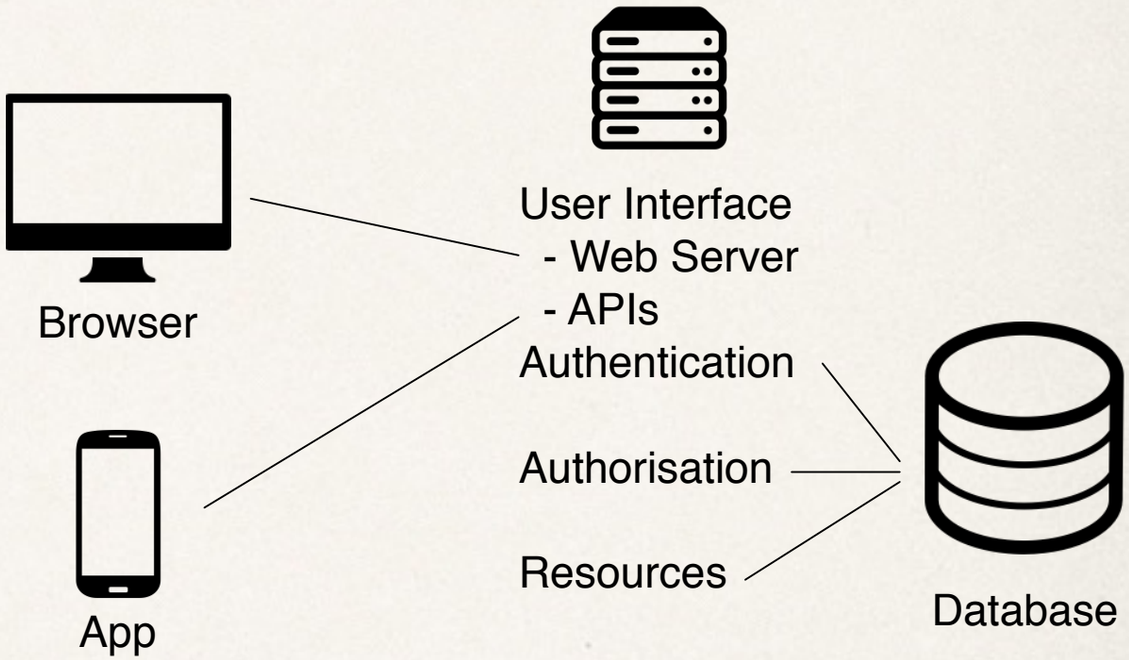
Architectures



The Monolith

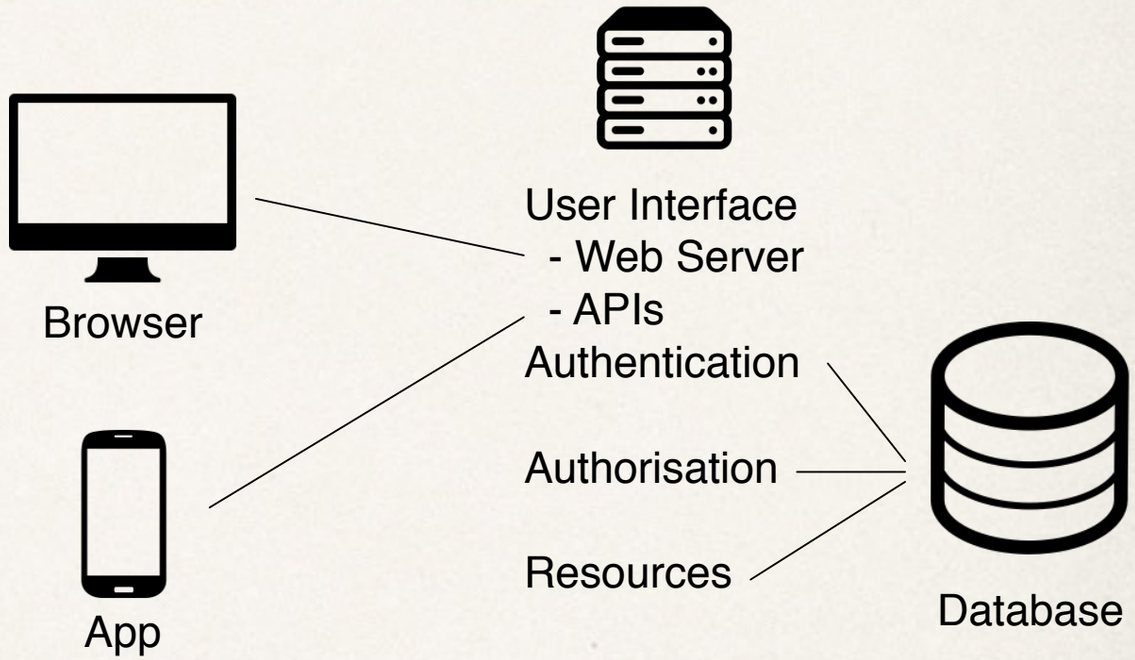


The Monolith



The Monolith

- Difficult to scale;
- Each change requires rebooting of the whole system and possibly also to re-deploy it;
- A crash of a component can bring down the whole system;
- Technology lock-in;

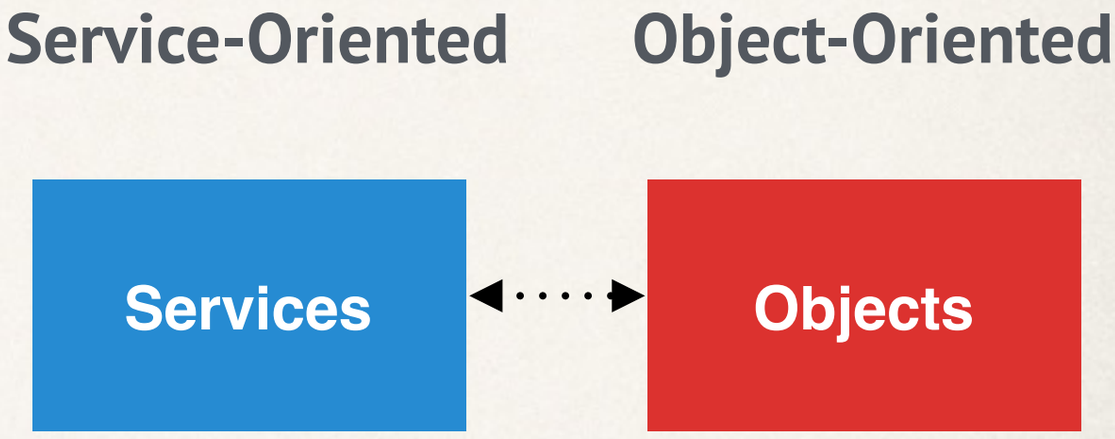


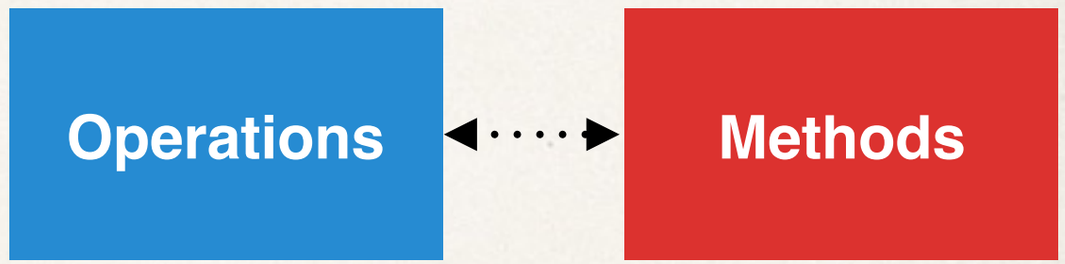
Service-Orientation

- Everything is a **service**;
- A service is an application that offers functionalities through **operations**;
- A service can **invoke** another service by calling one of its operations.

Service-Orientation

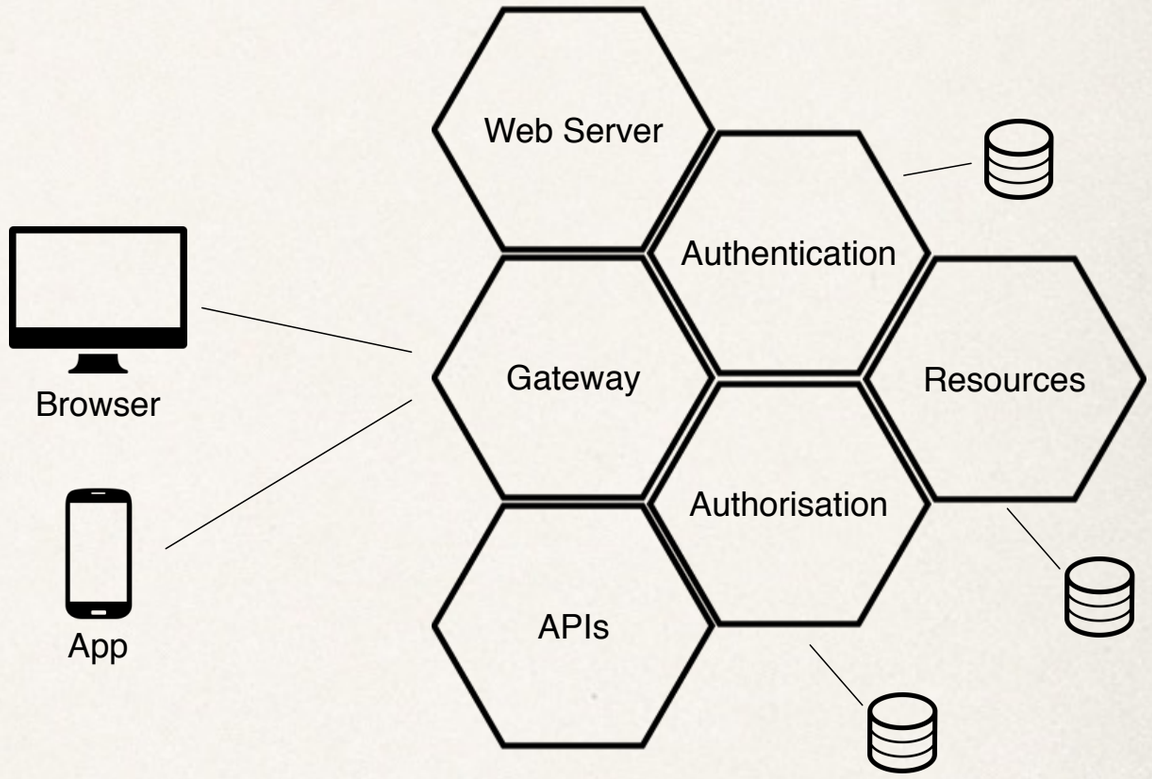
- Everything is a **service**;
- A service is an application that offers functionalities through **operations**;
- A service can **invoke** another service by calling one of its operations.



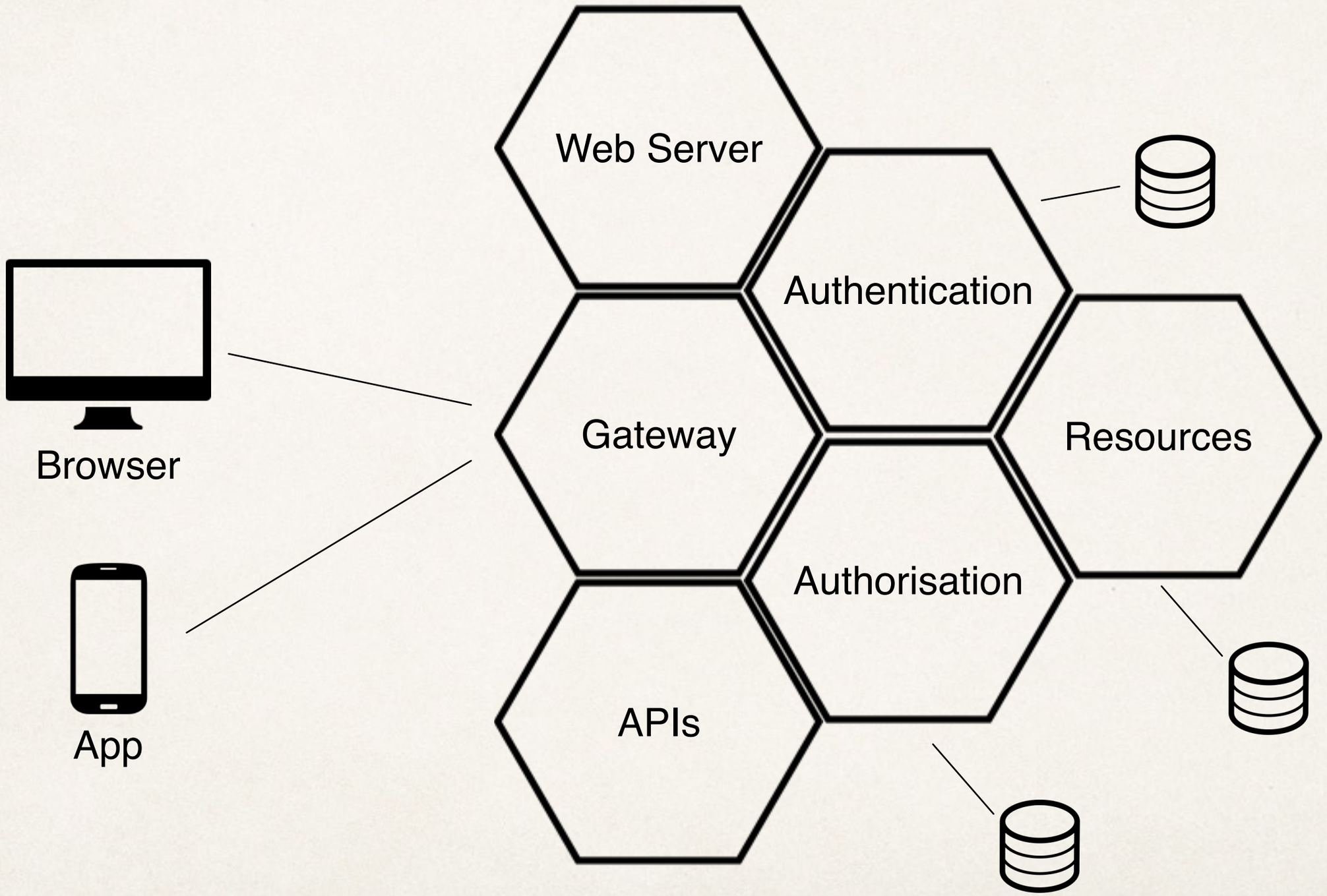


Microservices

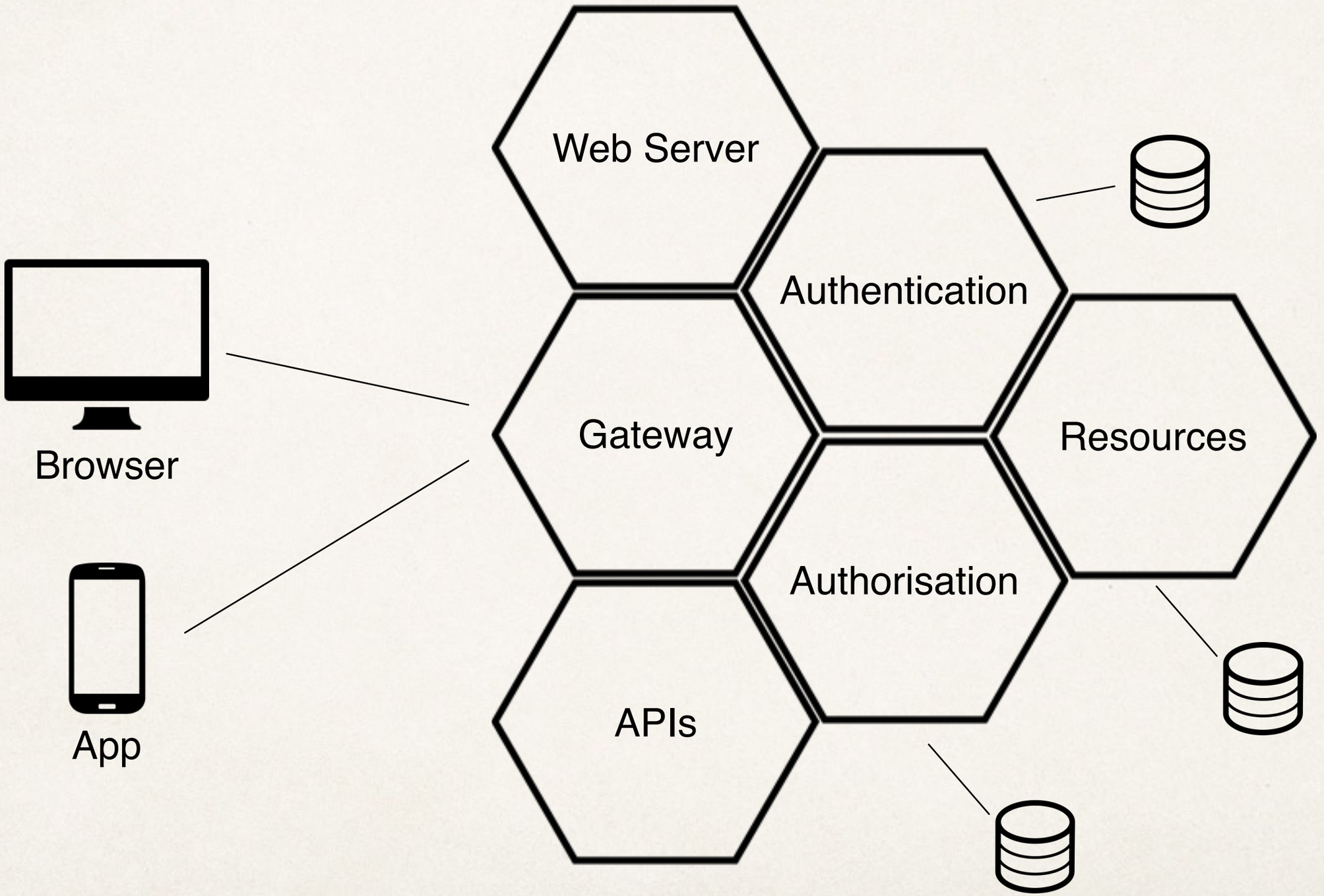
- API design is paramount;
- Partition of work and parallel development;
- Breakdown of complexity into “simple” and specialised services;
- Minimalistic evolution of “bloated” WS-* Service-Oriented Architectures:
  - Integrate ESB-like functionalities
  - Lightweight and human-oriented protocols (REST, JSON, etc.)



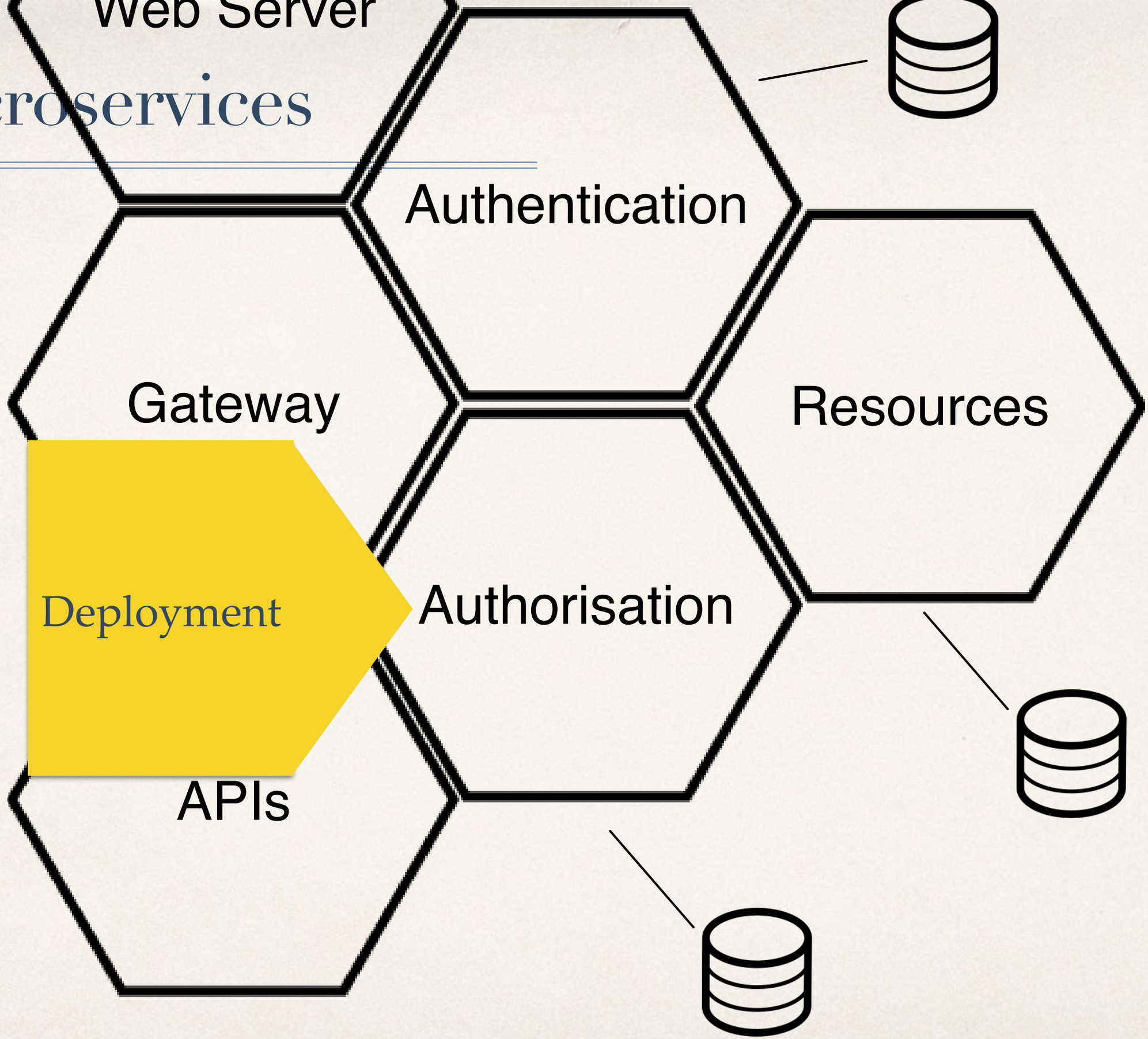
Microservices



Microservices

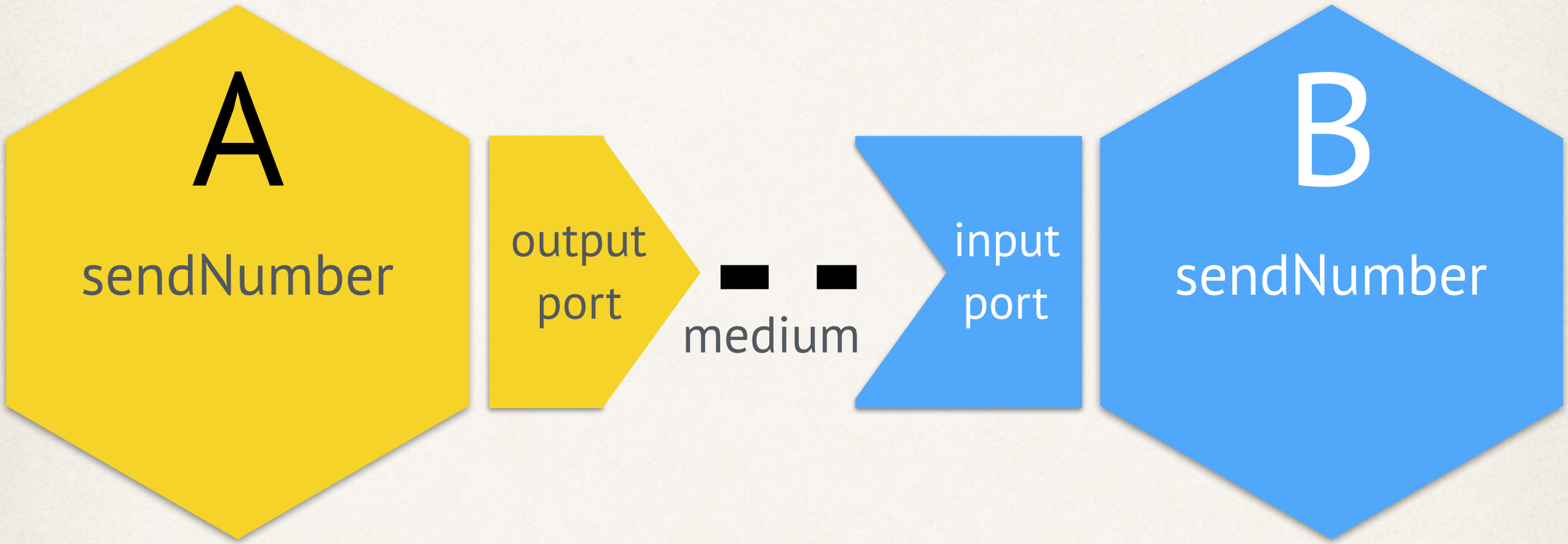


Microservices



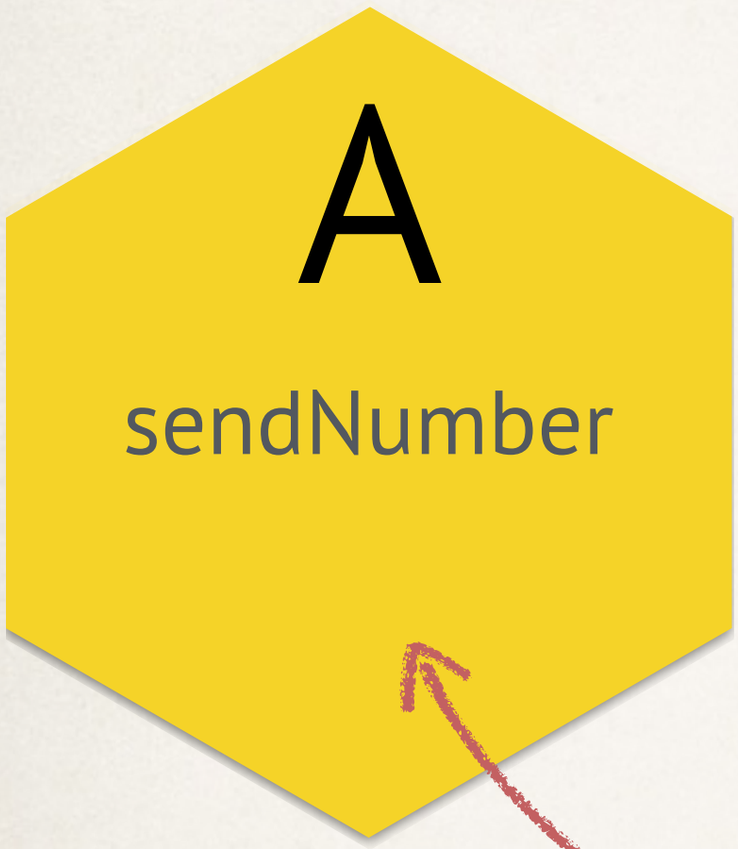
Introducing Jolie

The Jolie Way



- Services communicate through **ports**.
- **Ports** give access to an **interface**.
- An interface is a set of **operations**.
- An **output port** is used to invoke interfaces exposed by other services.
- An **input port** is used to expose an interface.

The Jolie Way



Deployment:
Location: socket://myaddress.org:80
Protocol: json/rpc (xml/rpc, …)
Interface: sendNumber(int)

Behaviour:

```
main {
 sendNumber@B( 5 )
}
```

The Jolie Way

```
interface MyInterface {
  OneWay: sendNumber( int )
}
```

```
include "MyInterface.iol"
outputPort  B  {
Location:
   "socket://localhost:8000"
Protocol: json/rpc
Interfaces: MyInterface
}

main
{
   sendNumber @ B ( 5 )
}
```

```
include "MyInterface.iol"
inputPort B {
Location:
   "socket://localhost:8000"
Protocol: json/rpc
Interfaces: MyInterface
}

main
{
   sendNumber( x )
}
```

The Jolie Way

```
interface MyInterface {
  OneWay: sendNumber( int )
}
```

```
include "MyInterface.iol"
outputPort B {
Location:
   "socket://localhost:8000"
Protocol: json/rpc
Interfaces: MyInterface
}

main
{
   sendNumber @ B ( 5 )
}
```

```
include "MyInterface.iol"
inputPort B {
Location:
   "socket://localhost:8000"
Protocol: json/rpc
Interfaces: MyInterface
}

main
{
   sendNumber( x )
}
```

Architectural Composition

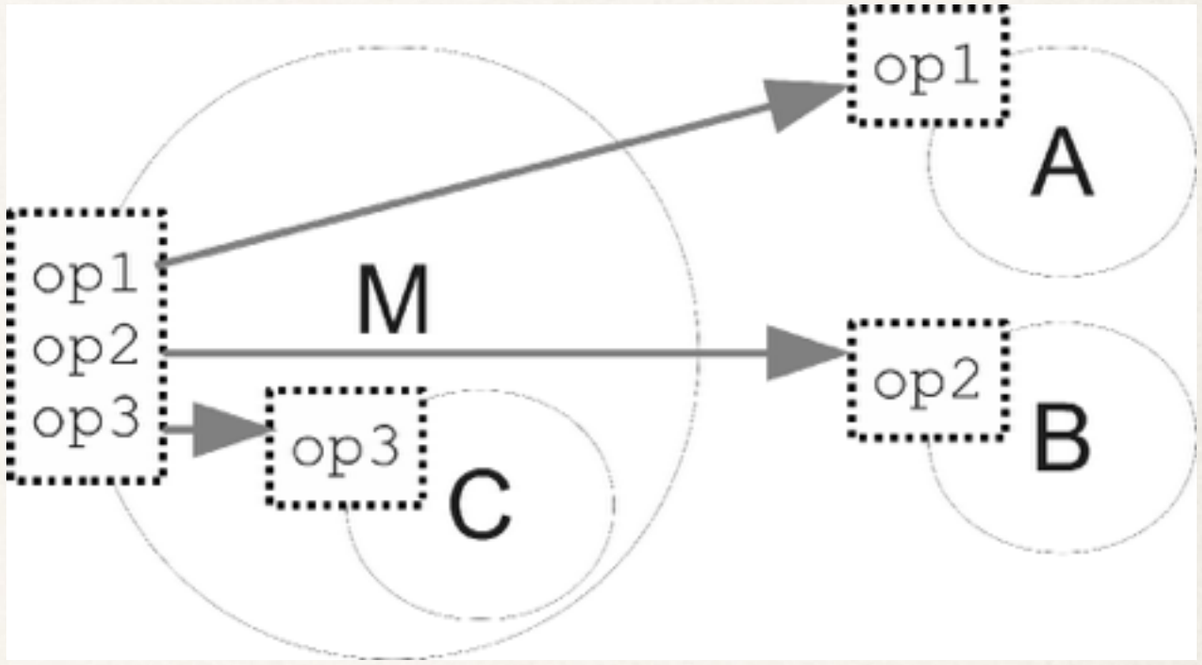
```
outputPort A {
    Location: "socket://someurlA.com:80/"
    Protocol: soap
    Interfaces: InterfaceA
}

outputPort B {
    Location: "socket://someurlB.com:80/"
    Protocol: xmlrpc
    Interfaces: InterfaceB
}

outputPort C {
    Interfaces: InterfaceC
}

embedded {
    Java: "example.serviceC" in C
}

inputPort M {
    Location: "socket://urlM.com:8000/"
    Protocol: sodep
    Aggregates: A, B, C
}
```

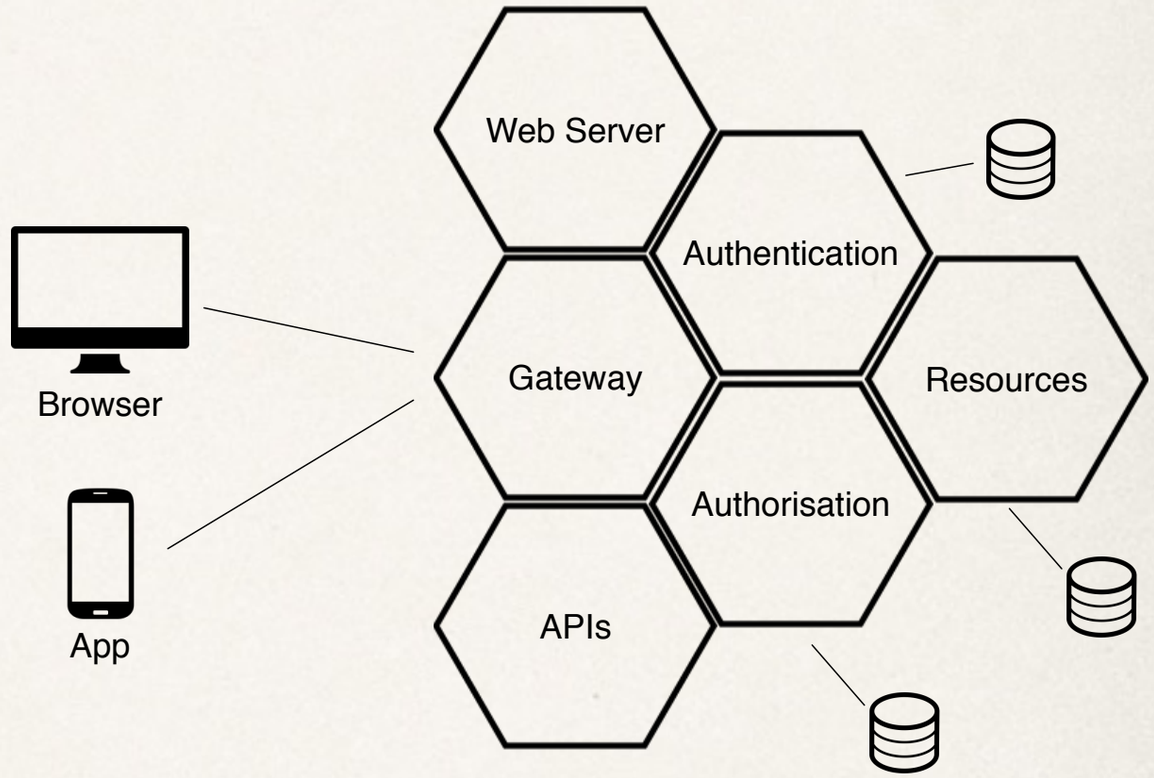


Others:

- Redirection
- Embedding

Recap on Microservices

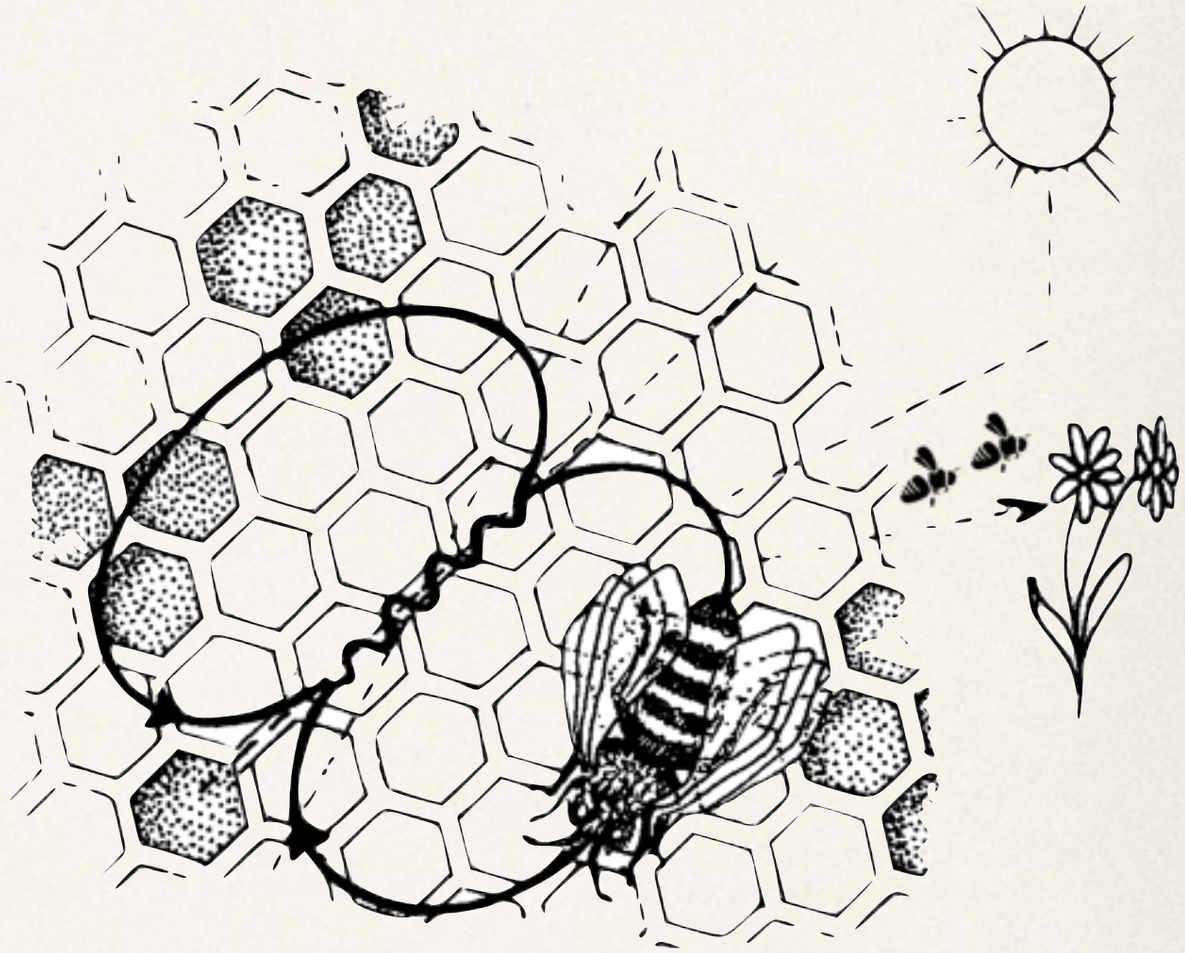
- API design is paramount;
- Partition of work and parallel development;
- Integrate ESB-like functionalities
- Lightweight and human-oriented protocols (REST, JSON, etc.)



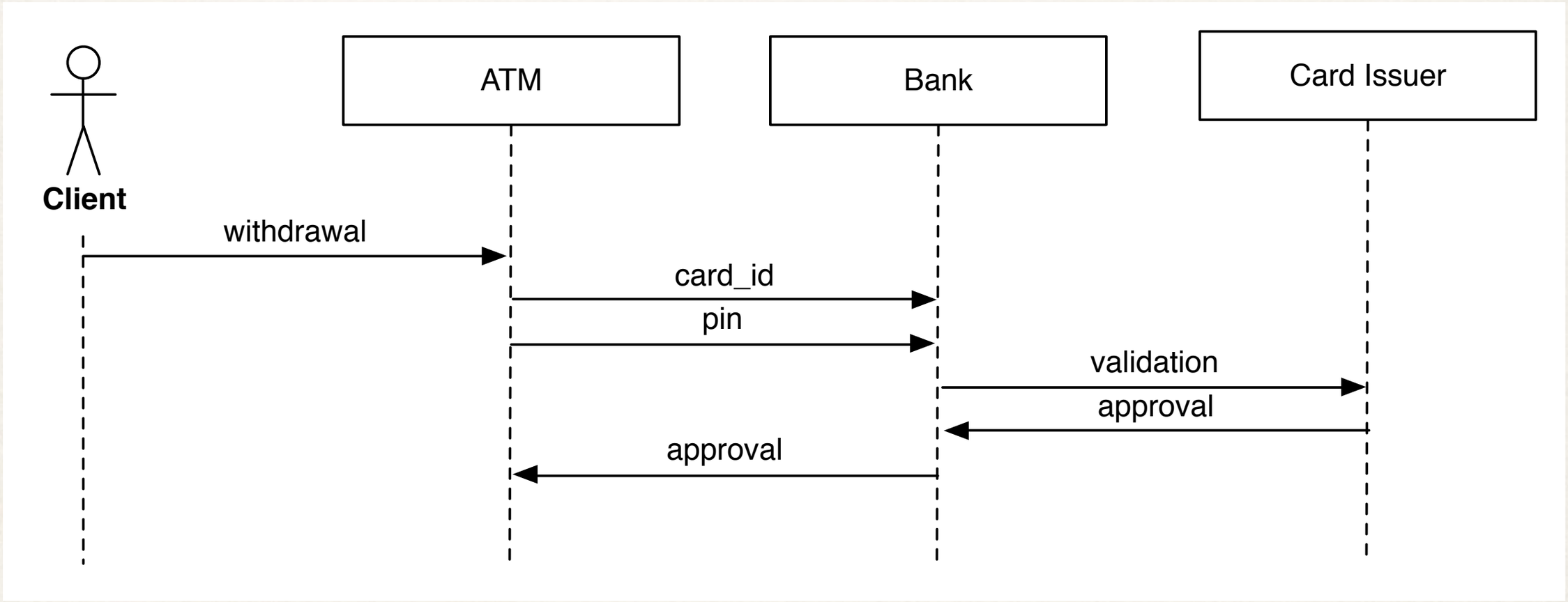
Choreographies

Protocols,

Correct implementations



Distributed programming



Distributed programming

ATM process

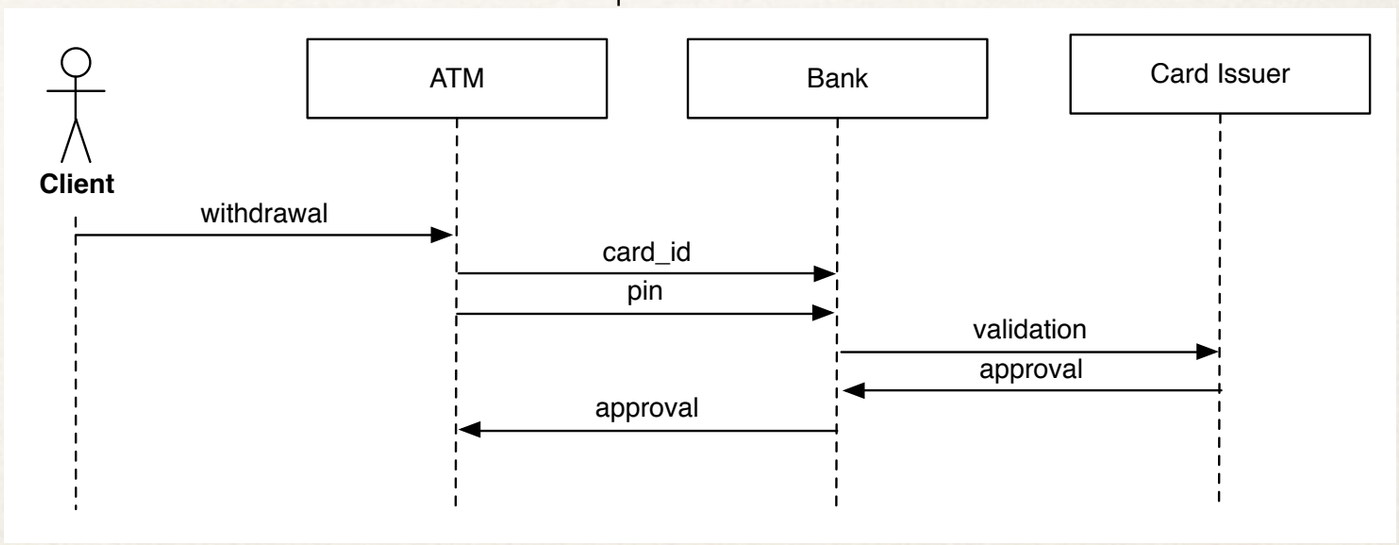
from ATM : *withdrawal*;
to Bank : *card_id*;
to Bank : *pin*;
from Bank : *approval*

Bank process

from ATM : *card_id*;
from ATM : *pin*;
to Card Issuer : *validation*;
from Card Issuer : *approval*;
to ATM : *approval*

Card Issuer process

from Card Issuer : *validation*;
to Bank : *approval*



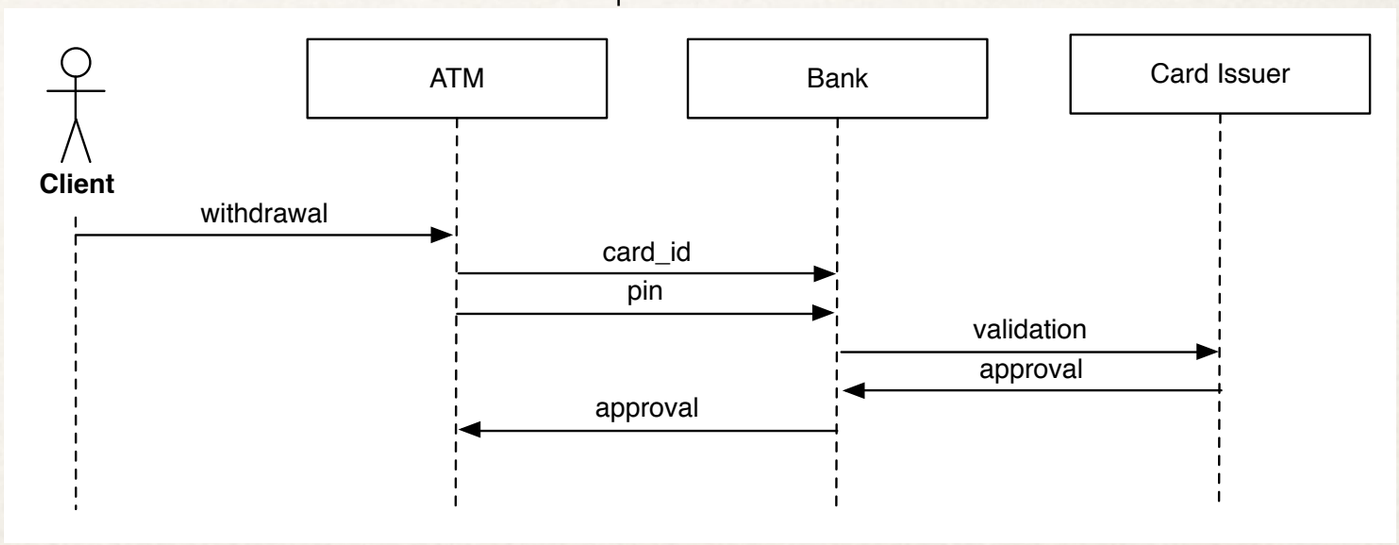
Distributed programming

Deadlocks

ATM process
from Client : *withdrawal*;
to Bank : *card_id*;
to Bank : *pin*;
from Bank : *approval*

Bank process
from ATM : *card_id*;
from ATM : *pin*;
to Card Issuer : *validation*;
from Card Issuer : *approval*;
to ATM : *approval*

Card Issuer process
from Card Issuer : *validation*;
to Bank : *approval*



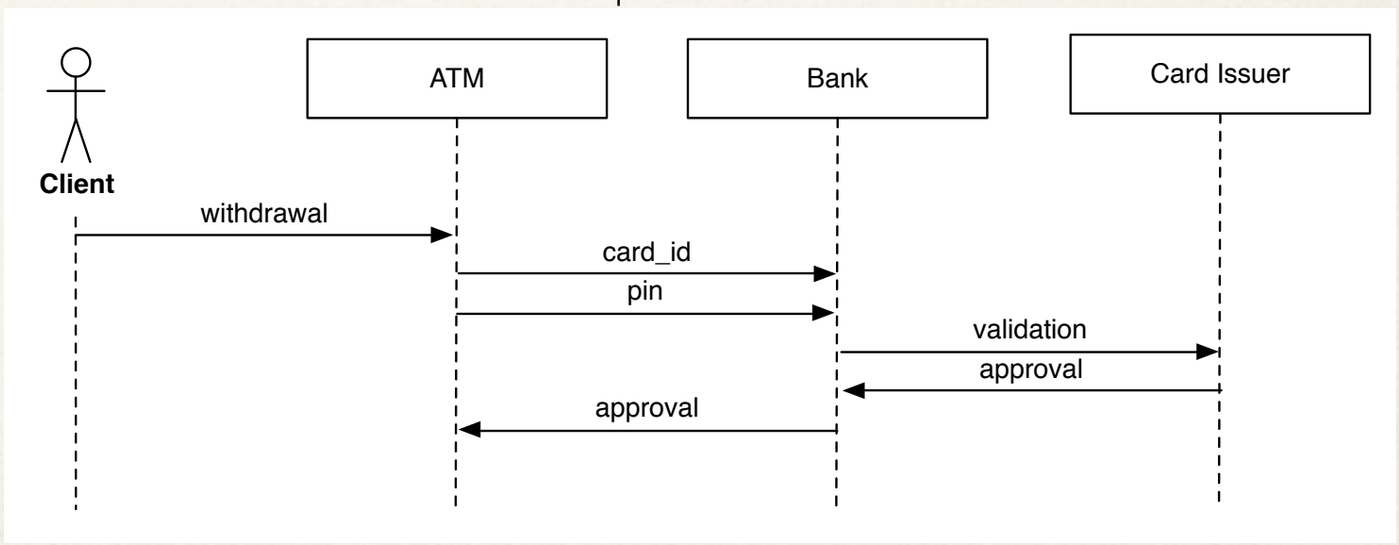
Distributed programming

Deadlocks Race Conditions

ATM process
from Client : *withdrawal*;
to Bank : *card_id*;
to Bank : *pin*;
from Bank : *approval*

Bank process
from ATM : *card_id*;
from ATM : *pin*;
to Card Issuer : *validation*;
from Card Issuer : *approval*;
to ATM : *approval*

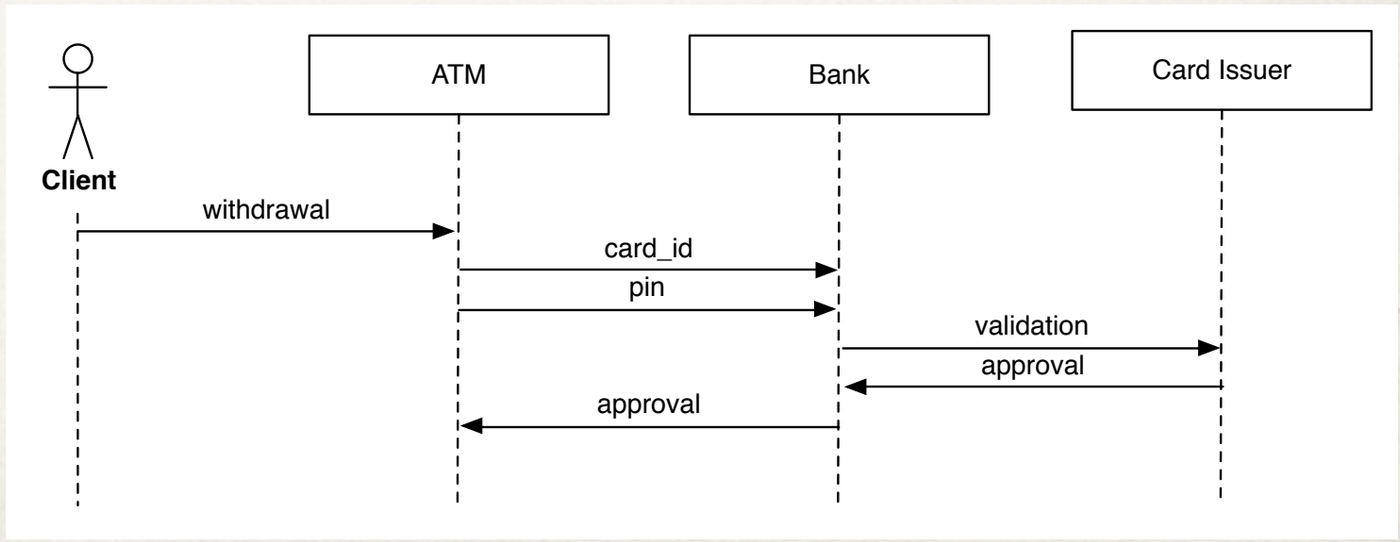
Card Issuer process
from Card Issuer : *validation*;
to Bank : *approval*



Choreography

$$\texttt{Client} \rightarrow \texttt{ATM} : withdrawal;$$
$$\texttt{ATM} \rightarrow \texttt{Bank} : card\_id;$$
$$\texttt{ATM} \rightarrow \texttt{Bank} : pin;$$
$$\texttt{Bank} \rightarrow \texttt{Card Issuer} : validation;$$
$$\texttt{Card Issuer} \rightarrow \texttt{Bank} : approval;$$
$$\texttt{Bank} \rightarrow \texttt{ATM} : approval$$

Simple + **free from deadlocks and races** by design



Correctness by design and by construction

Choreography —*EPP*→ Endpoint Projection

(Correct by design) *(Correct by construction)*

$\texttt{ATM} \rightarrow \texttt{Bank} : \mathit{card\_id};$
$\texttt{ATM} \rightarrow \texttt{Bank} : \mathit{pin};$
$\texttt{Bank} \rightarrow \texttt{Card Issuer} : \mathit{validation};$
$\texttt{Card Issuer} \rightarrow \texttt{Bank} : \mathit{approval};$
$\texttt{Bank} \rightarrow \texttt{ATM} : \mathit{approval}$

EPP

ATM process
to Bank : $card\_id$;
to Bank : pin;
from Bank : $approval$

Bank process
from ATM : $card\_id$;
from ATM : pin;
to Card Issuer : $validation$;
from Card Issuer : $approval$;
to ATM : $approval$

Card Issuer process
from Card Issuer : $validation$;
to Bank : $approval$

Choreographies

```
User.getInput(usr) → AuthCtrl.username(usr);
User.getInput(pswd) → AuthCtrl.password(pswd);
AuthCtrl.validLogin = validate@Validator(usr, pswd);
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

```
User.getInput(usr) → AuthCtrl.username(usr);
User.getInput(pswd) → AuthCtrl.password(pswd);
AuthCtrl.validLogin = validate@Validator(usr, pswd);
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

```
scope {
  User.getInput(usr) → AuthCtrl.username(usr);
  User.getInput(pswd) → AuthCtrl.password(pswd);
  AuthCtrl.validLogin = validate@Validator(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

```
scope {
  User.getInput(usr) → AuthCtrl.username(usr);
  User.getInput(pswd) → AuthCtrl.password(pswd);
  AuthCtrl.validLogin = validate@Validator(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

```
scope {
  User.getInput(usr) → AuthCtrl.username(usr);
  User.getInput(pswd) → AuthCtrl.password(pswd);
  AuthCtrl.validLogin = validate@Validator(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

```
scope {
  User.getInput(usr) → AuthCtrl.username(usr);
  User.getInput(pswd) → AuthCtrl.password(pswd);
  AuthCtrl.validLogin = validate@Validator(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

```
scope {
  User.getInput(usr) → AuthCtrl.username(usr);
  User.getInput(pswd) → AuthCtrl.password(pswd);
  AuthCtrl.validLogin = validate@Validator(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

```
scope {
  User.getInput(usr) → AuthCtrl.username(usr);
  User.getInput(pswd) → AuthCtrl.password(pswd);
  AuthCtrl.validLogin = validate@Validator(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

```
// UPDATE
User.getInput(usr) → AuthCtrl.username(usr);
AuthCtrl.usr → OpenID.username(usr);
OpenID(usr) → User.forward(usr);
User.getInput(pswd) → OpenID.password(pswd);
OpenID.validLogin = validate@Validator(usr, pswd);
OpenID.validLogin → AuthCtrl.result(validLogin)
```

Dynamic Choreographies

```
{
  User.ge…  …AuthCtrl.username(usr);
  User.getInput(pswd) …assword(pswd);
  AuthCtrl.validLogin = validate@Valid…(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

```
// UPDATE
User.getInput(usr) → AuthCtrl.username(usr);
AuthCtrl.usr → OpenID.username(usr);
OpenID(usr) → User.forward(usr);
User.getInput(pswd) → OpenID.password(pswd);
OpenID.validLogin = validate@Validator(usr, pswd);
OpenID.validLogin → AuthCtrl.result(validLogin)
```

Dynamic Choreographies

```
{
  User.ge…  …AuthCtrl.username(usr);
  User.getInput(pswd) …assword(pswd);
  AuthCtrl.validLogin = validate@Valid…(usr, pswd)
};
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

```
// UPDATE
User.getInput(usr) → AuthCtrl.username(usr);
AuthCtrl.usr → OpenID.username(usr);
OpenID(usr) → User.forward(usr);
User.getInput(pswd) → OpenID.password(pswd);
OpenID.validLogin = validate@Validator(usr, pswd);
OpenID.validLogin → AuthCtrl.result(validLogin)
```

Dynamic Choreographies

```
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

```
// UPDATE
User.getInput(usr) → AuthCtrl.username(usr);
AuthCtrl.usr → OpenID.username(usr);
OpenID(usr) → User.forward(usr);
User.getInput(pswd) → OpenID.password(pswd);
OpenID.validLogin = validate@Validator(usr, pswd);
OpenID.validLogin → AuthCtrl.result(validLogin)
```

Dynamic Choreographies

```
// UPDATE
User.getInput(usr) → AuthCtrl.username(usr);
AuthCtrl.usr → OpenID.username(usr);
OpenID(usr) → User.forward(usr);
User.getInput(pswd) → OpenID.password(pswd);
OpenID.validLogin = validate@Validator(usr, pswd);
OpenID.validLogin → AuthCtrl.result(validLogin)
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

Dynamic Choreographies

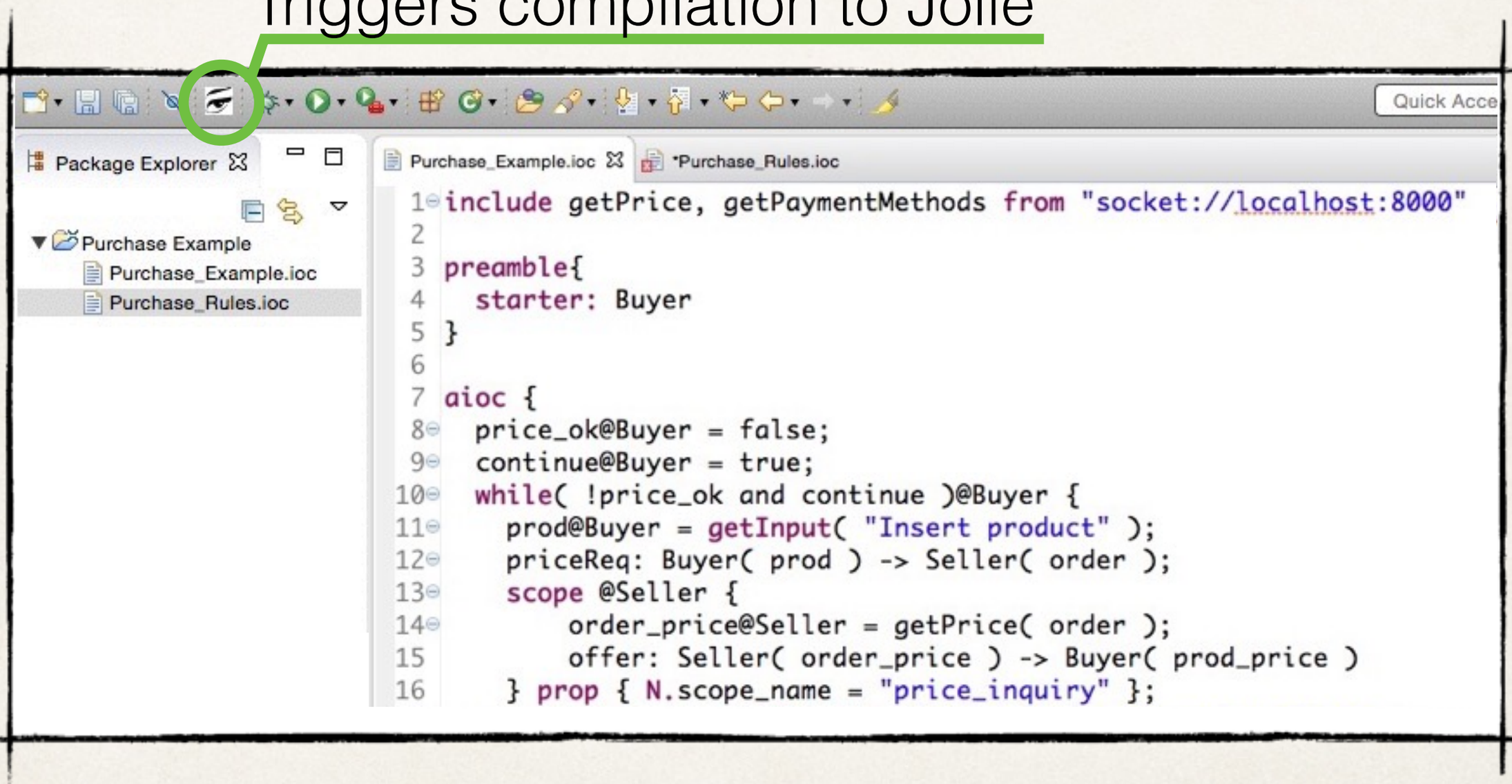
Choreography —*EPP*→ Endpoint Projection

(Correct by design) *(Correct by construction)*

```
// UPDATE
User.getInput(usr) → AuthCtrl.username(usr);
AuthCtrl.usr → OpenID.username(usr);
OpenID(usr) → User.forward(usr);
User.getInput(pswd) → OpenID.password(pswd);
OpenID.validLogin = validate@Validator(usr, pswd);
OpenID.validLogin → AuthCtrl.result(validLogin)
if AuthCtrl.validLogin {
  AuthCtrl.tkn = generateToken@TM(usr,"contacts");
  AuthCtrl.tkn → User.getToken(tkn);
  AuthCtrl.tkn → ResMng.getToken(authTkn);
  User.tkn → ResMng.reqContacts(reqTkn);
  if ResMng.(authTkn == reqTkn) {
    ResMng.contacts = retrieve@ContactsBook(usr);
    ResMng.contacts → User.getContacts(contacts)
  } else { ResMng → User.unauthorised() }
} else { AuthCtrl → User.invalidLogin() }
```

AIOCJ

Triggers compilation to Jolie



Time for discussion!

Appendix

Jolie Composition Operators

```
include "console.iol"
include "time.iol"

timeout = 250;
timeout.operation = "timeout";
txt = "Beutiful";
{
 spellCheck@BingSpell({ .text = txt, .location = myLoc })
 |
 setNextTimeout@Time( timeout )
};
[ spellCheckResponse( text ) ]{ println@Console( text )() }
[ timeout() ]{ throw( TimeoutException ) }
```

```
// USER PROCESS
// inputs
username@AuthCtrl(usr);
password@AuthCtrl(pswd);
[ getToken(tkn) from AuthCtrl ]{
   [ getContacts(contacts) from ResMng ]
   [ unauthorised() from ResMng ]
}
[ invalidLogin() from ResMng ]
```

```
// ACCESS MANAGER PROCESS
username(usr) from User;
password(usr) from User;
validate@Validator(usr, pswd)(validLogin);
if validLogin {
   generateToken@TM(usr,"contacts")(tkn);
   genToken@User(tkn);
   genToken@ResMng(tkn)
} else { invalidLogin@User() }
```

```
// RESOURCE MANAGER PROCESS
getToken(authTkn) from AuthCtrl;
reqContacts(reqTkn) from User;
if (authTkn == reqTkn) {
   retrieve@ContactsBook(usr)(contacts);
   getContacts@User(contacts)
} else { unauthorised@User() }
```

AIOCJ Choreographic Framework
(Advanced) Hello World example

AIOCJ Choreographic Framework
(Advanced) Hello World example