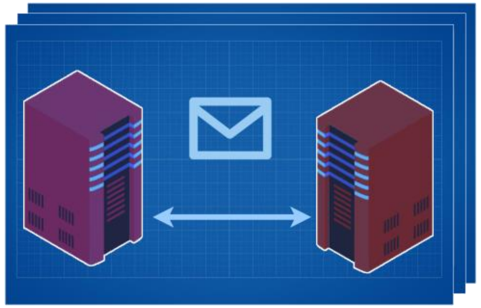
Programming Microservice Choreographies: a security use case

Saverio Giallorenzo[1], Fabrizio Montesi[1], Marco Peressotti[1], Luisa Zeppelin[2]

[1] University of Southern Denmark
[2] University of Hamburg

2020-09-09 @ Microservices 2020

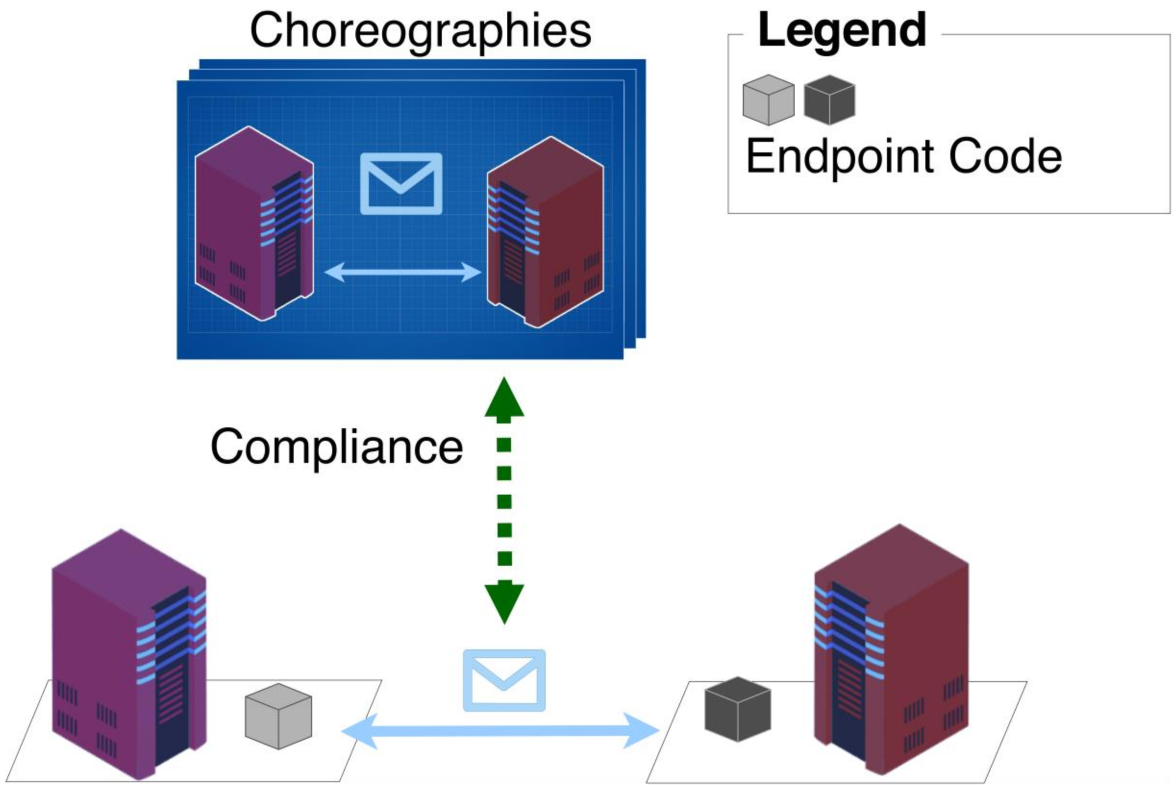


- Choreography

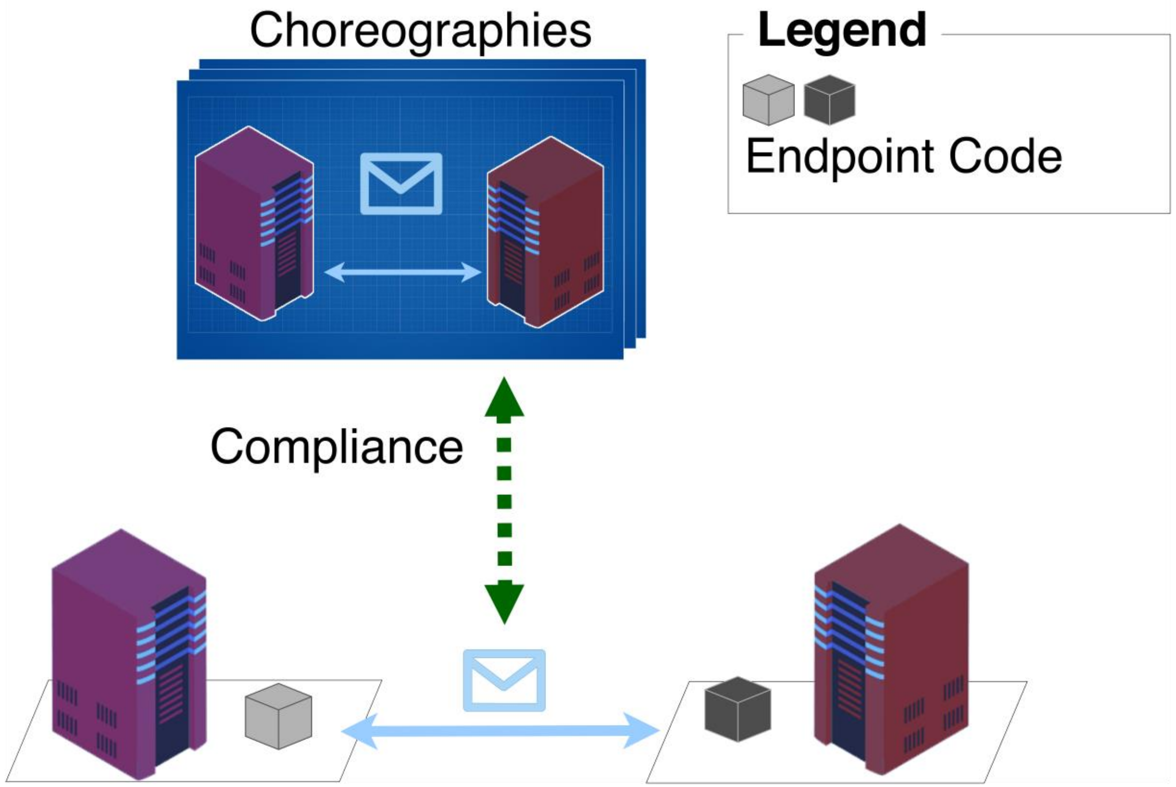
noun

a (decentralised) coordination plan for concurrent systems based on message passing

Microservices and Choreographies

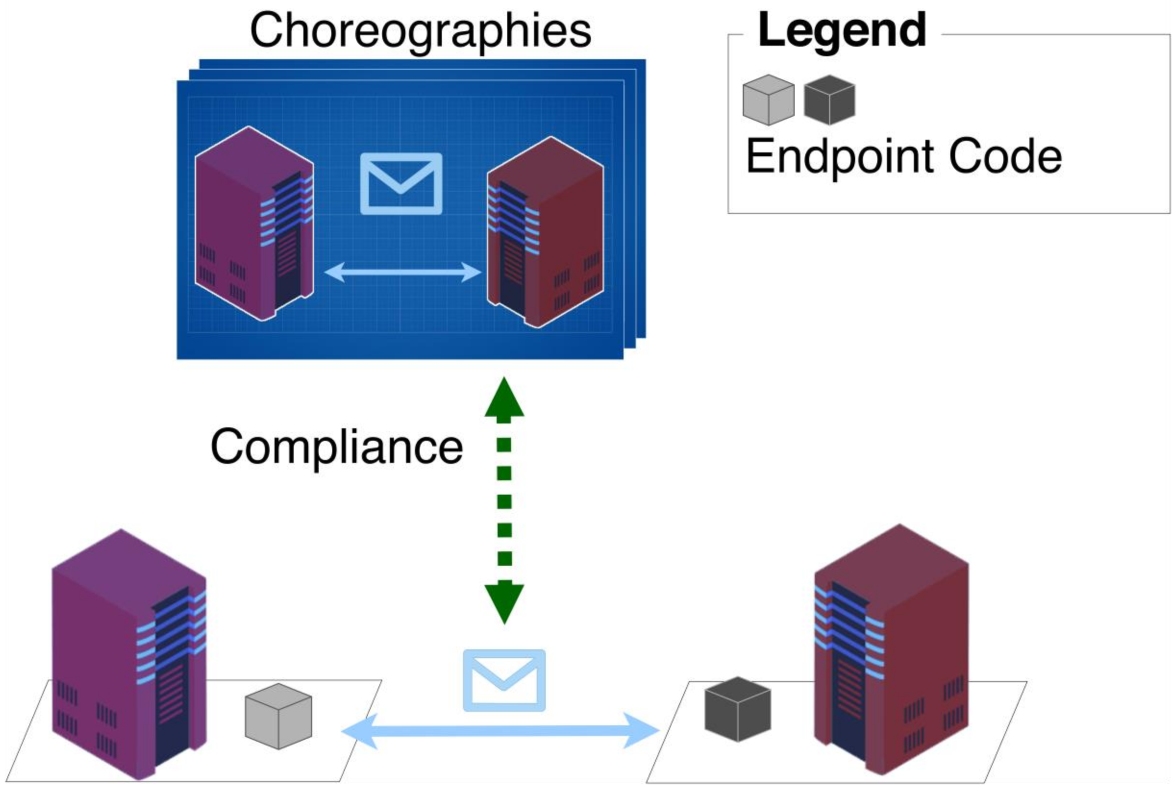


Microservices and Choreographies



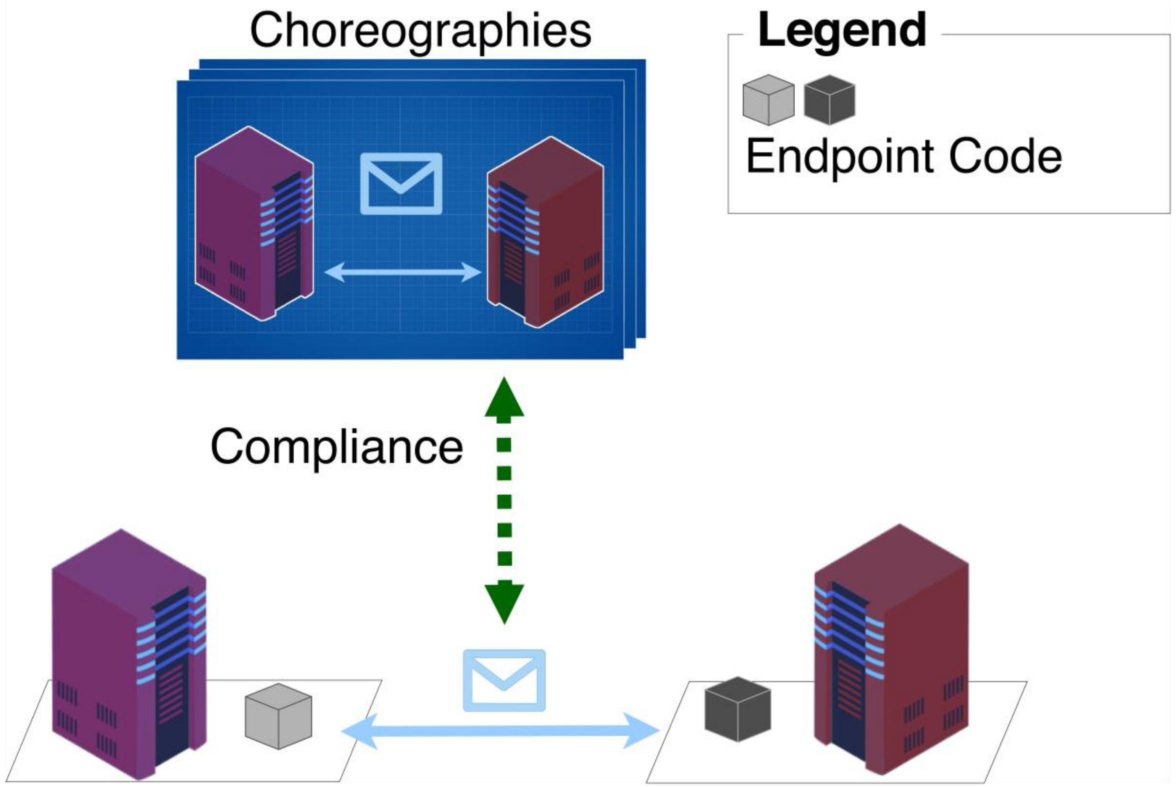
- Coordination is hard

Microservices and Choreographies



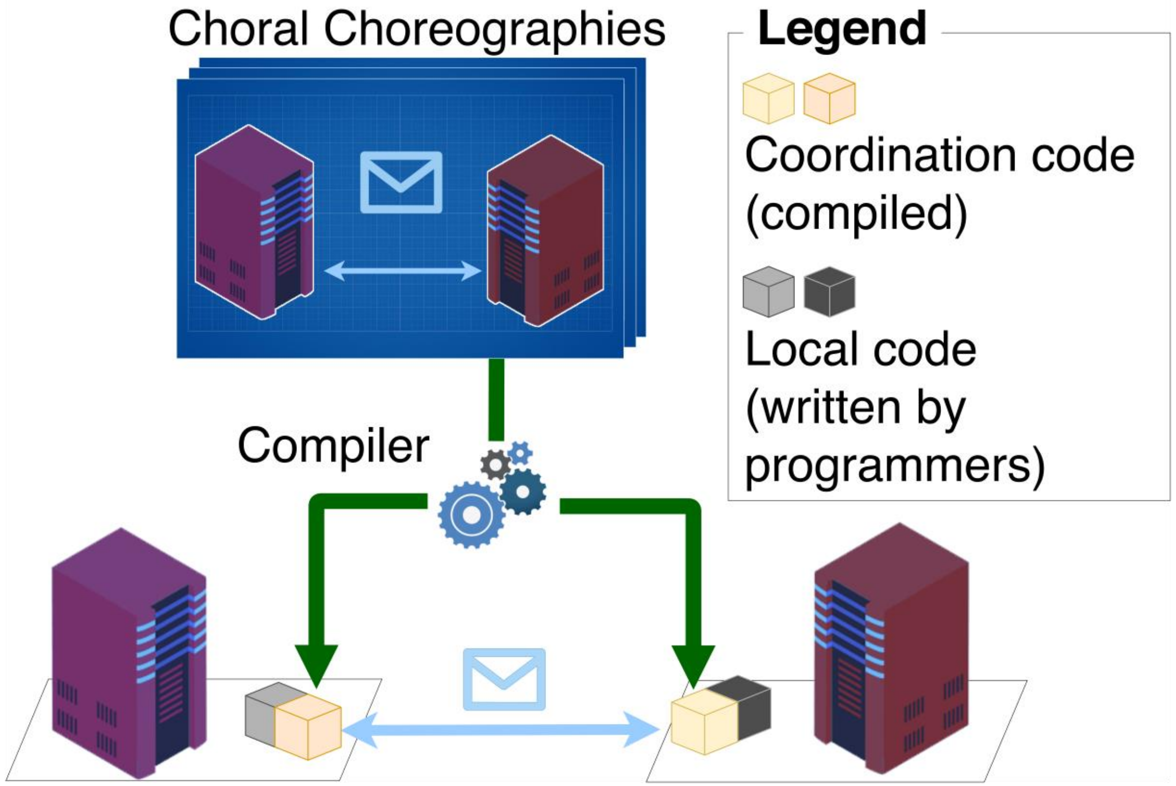
- Coordination is hard
- What do we want?
  - Global specification of choreographies

Microservices and Choreographies



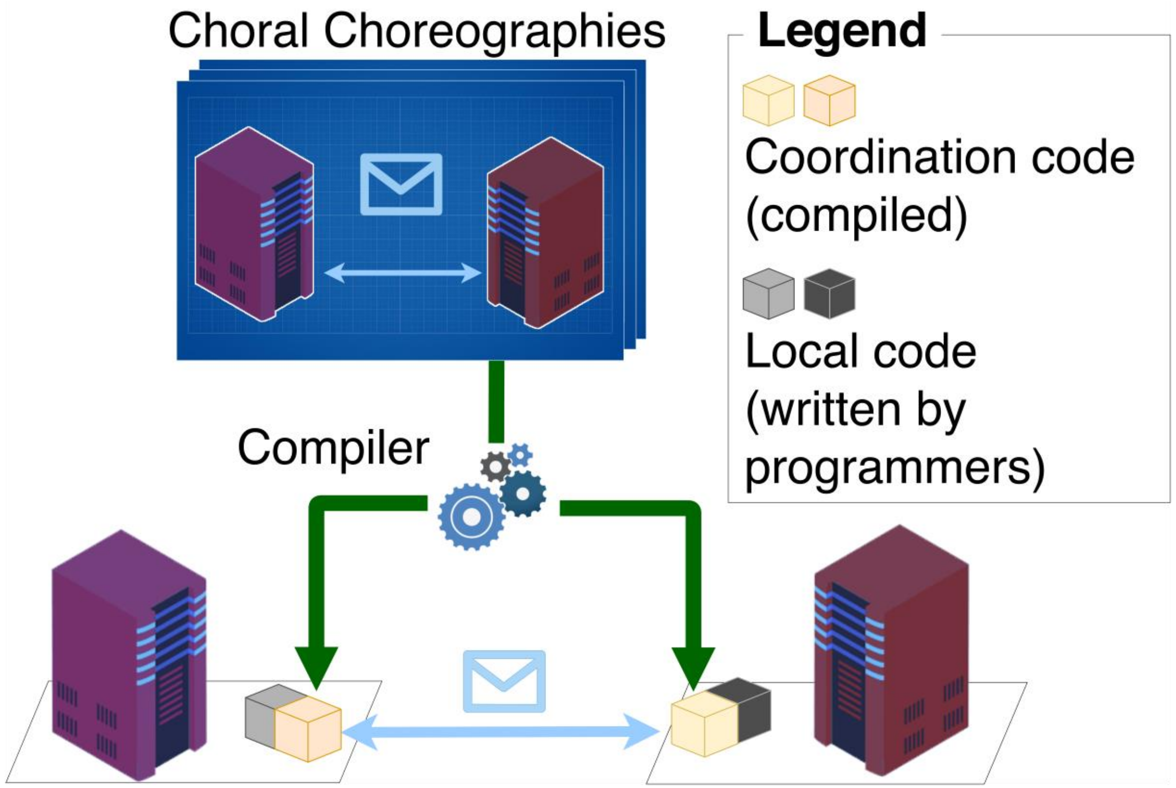
- Coordination is hard
- What do we want?
  - Global specification of choreographies
  - Automatic translation to compliant endpoint implementations

From Choreographies to CHORAL


Choral Choreographies
Legend
Coordination code (compiled)
Local code (written by programmers)
Compiler

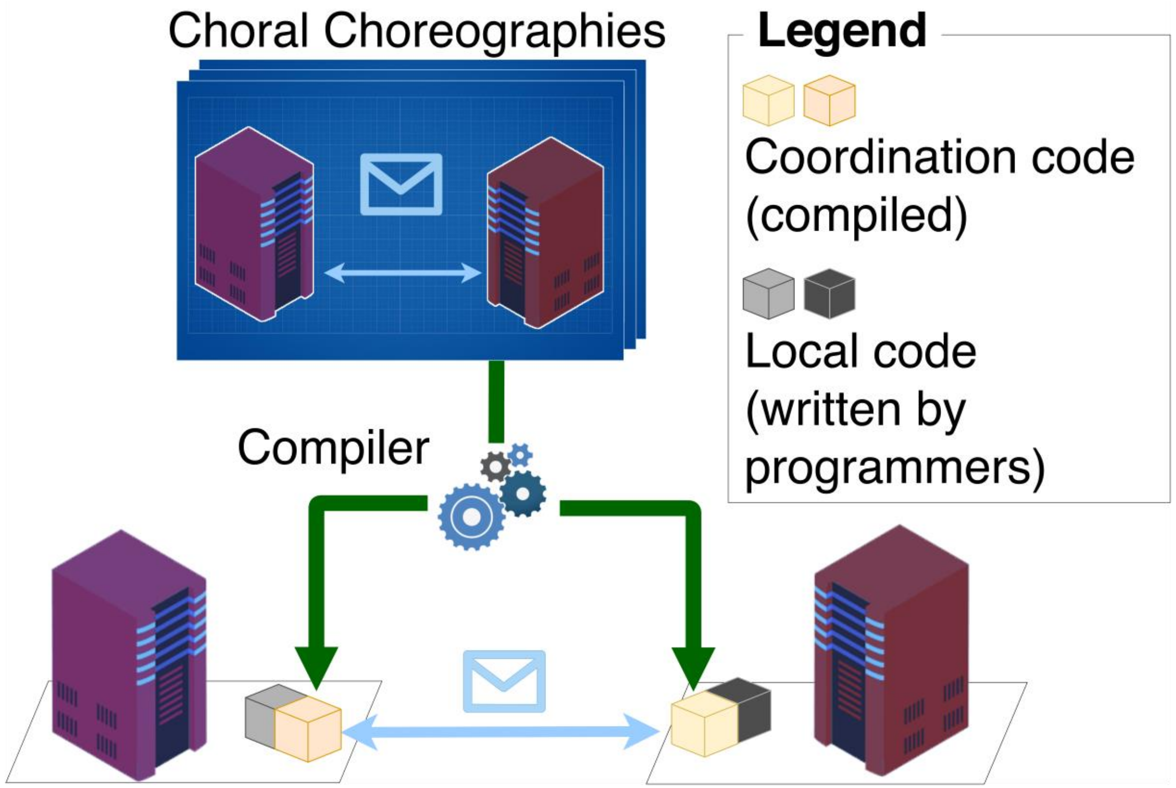
- Automatic generation of Java library that implements each role

From Choreographies to CHORAL


Choral Choreographies
Legend
Coordination code (compiled)
Local code (written by programmers)
Compiler

- Automatic generation of Java library that implements each role
- Use for a microservice system

From Choreographies to CHORAL


Choral Choreographies
Legend
Coordination code (compiled)
Local code (written by programmers)
Compiler

- Automatic generation of Java library that implements each role
- Use for a microservice system
- Deadlock-free!

How does it work?

Hello Roles

Example

- Demo time!

```
class HelloRoles@( A, B ) {
    public void sayHello() {
        String@A a = "Hello from A"@A;
        String@B b = "Hello from B"@B;
        System@A.out.println( a );
        System@B.out.println( b );
    }
}
```

```
class HelloRoles_A {
    public void sayHello() {
        String a;
        a = "Hello from A";
        System.out.println( a );
    }
}
```

Hello Roles

- Demo time!

```
class HelloRoles@( A, B ) {
    public void sayHello() {
        String@A a = "Hello from A"@A;
        String@B b = "Hello from B"@B;
        System@A.out.println( a );
        System@B.out.println( b );
    }
}
```

```
class HelloRoles_A {
    public void sayHello() {
        String a;
        a = "Hello from A";
        System.out.println( a );
    }
}
```

- Computation @ different roles (see Hybrid Logic)

Hello Roles

Example

- Demo time!

```
class HelloRoles@( A, B ) {
    public void sayHello() {
        String@A a = "Hello from A"@A;
        String@B b = "Hello from B"@B;
        System@A.out.println( a );
        System@B.out.println( b );
    }
}
```

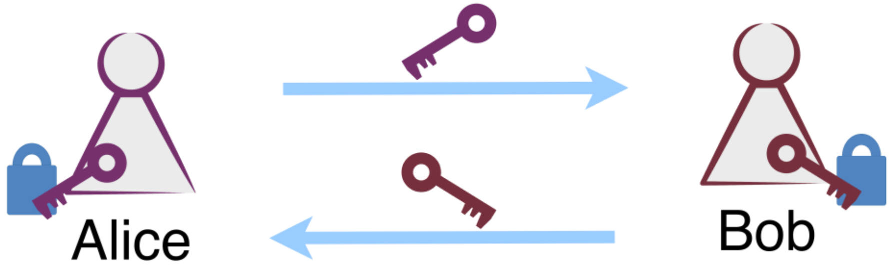
```
class HelloRoles_A {
    public void sayHello() {
        String a;
        a = "Hello from A";
        System.out.println( a );
    }
}
```

- Computation @ different roles (see Hybrid Logic)
- Compliance!

What about interactions?

Diffie-Hellman Key Exchange

Example



- Communication!

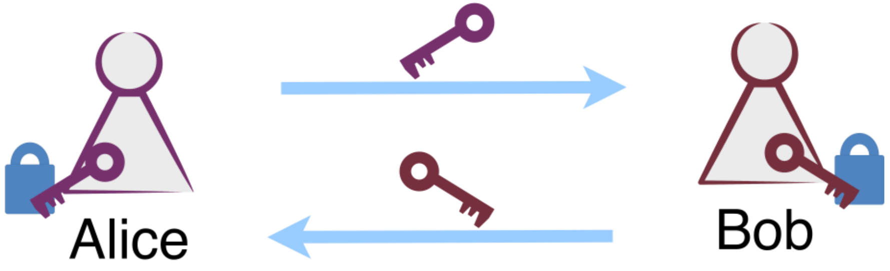
```
class DiffieHellman@( Alice, Bob ){
  public static void run(){
    Key@Alice aPK = exp( aKpair.gen, aKpair.secret );
    Key@Bob bPK = exp( bKpair.gen, bKpair.secret );




  }
}
```

Diffie-Hellman Key Exchange

Example



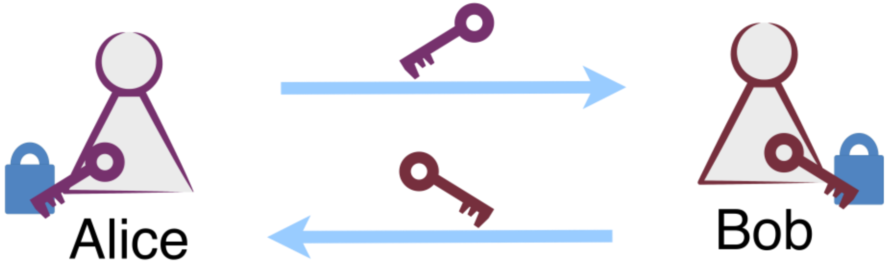
- Communication!
- Method com

```
class DiffieHellman@( Alice, Bob ){
  public static void run(){
    Key@Alice aPK = exp( aKpair.gen, aKpair.secret );
    Key@Bob bPK = exp( bKpair.gen, bKpair.secret );
    Key@Bob aliceKey = < Key >com( aPK@Alice );
    Key@Alice bobKey = < Key >com( bPK@Bob );


  }
}
```

Diffie-Hellman Key Exchange

Example



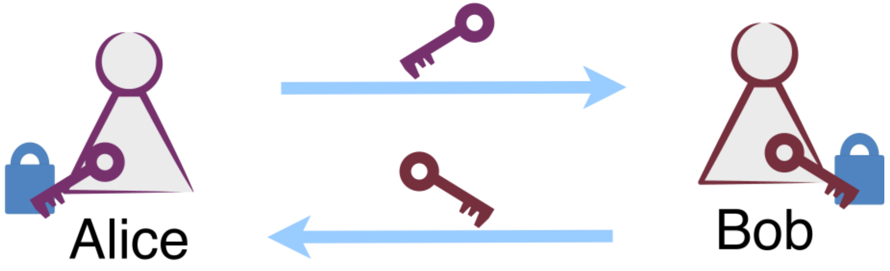
- Communication!
- Method com

```
class DiffieHellman@( Alice, Bob ){
  public static void run(){
    Key@Alice aPK = exp( aKpair.gen, aKpair.secret );
    Key@Bob bPK = exp( bKpair.gen, bKpair.secret );
    Key@Bob aliceKey = < Key >com( aPK@Alice );
    Key@Alice bobKey = < Key >com( bPK@Bob );


  }
}
```

Diffie-Hellman Key Exchange

Example



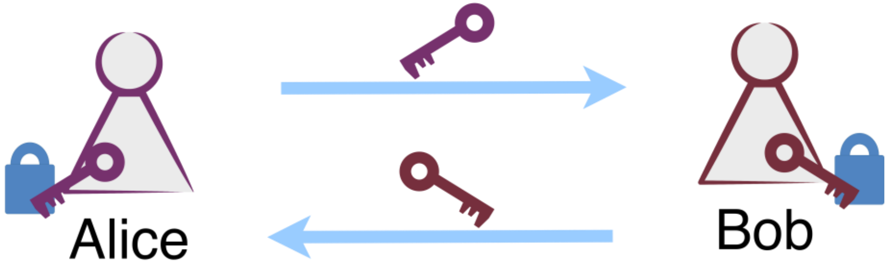
- Communication over channels

```
class DiffieHellman@( Alice, Bob ){
  public static void run(SymChannel@( Alice, Bob )< Key > channel){
    Key@Alice aPK = exp( aKpair.gen, aKpair.secret );
    Key@Bob bPK = exp( bKpair.gen, bKpair.secret );
    Key@Bob aliceKey = channel.< Key >com( aPK@Alice );
    Key@Alice bobKey = channel.< Key >com( bPK@Bob );


  }
}
```

Diffie-Hellman Key Exchange

Example



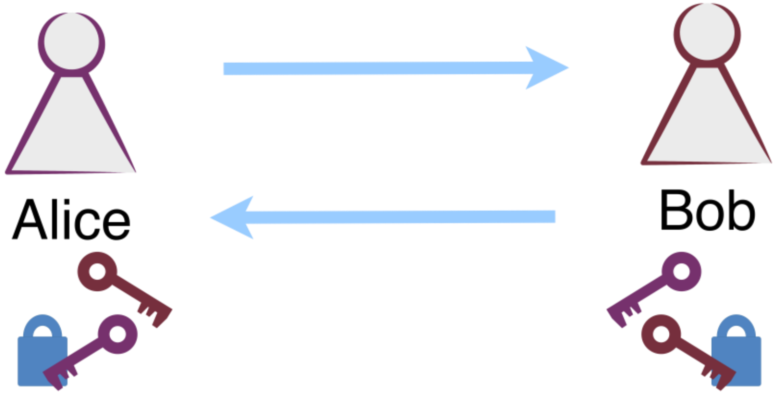
- Communication over channels

```
class DiffieHellman@( Alice, Bob ){
  public static void run(SymChannel@( Alice, Bob )< Key > channel){
    Key@Alice aPK = exp( aKpair.gen, aKpair.secret );
    Key@Bob bPK = exp( bKpair.gen, bKpair.secret );
    Key@Bob aliceKey = channel.< Key >com( aPK@Alice );
    Key@Alice bobKey = channel.< Key >com( bPK@Bob );


  }
}
```

Diffie-Hellman Key Exchange

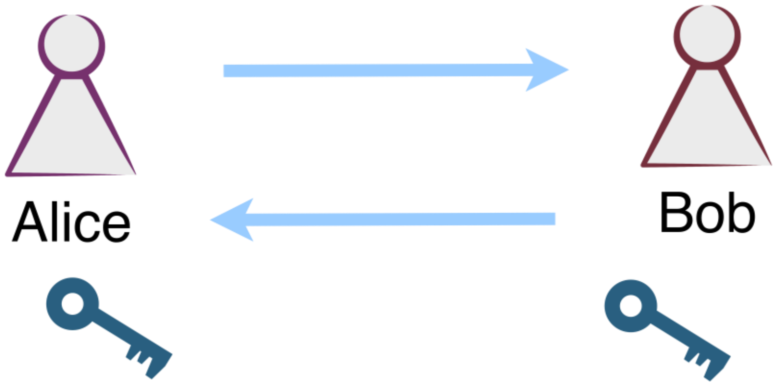
Example



```
class DiffieHellman@( Alice, Bob ){
  public static void run(SymChannel@( Alice, Bob )< Key > channel){
    Key@Alice aPK = exp( aKpair.gen, aKpair.secret );
    Key@Bob bPK = exp( bKpair.gen, bKpair.secret );
    Key@Bob aliceKey = channel.< Key >com( aPK@Alice );
    Key@Alice bobKey = channel.< Key >com( bPK@Bob );
    Key@Alice sharedKey = exp( bobKey, aKpair.secret );
    Key@Bob sharedKey = exp( aliceKey, bKpair.secret );
  }
}
```

Diffie-Hellman Key Exchange

Example

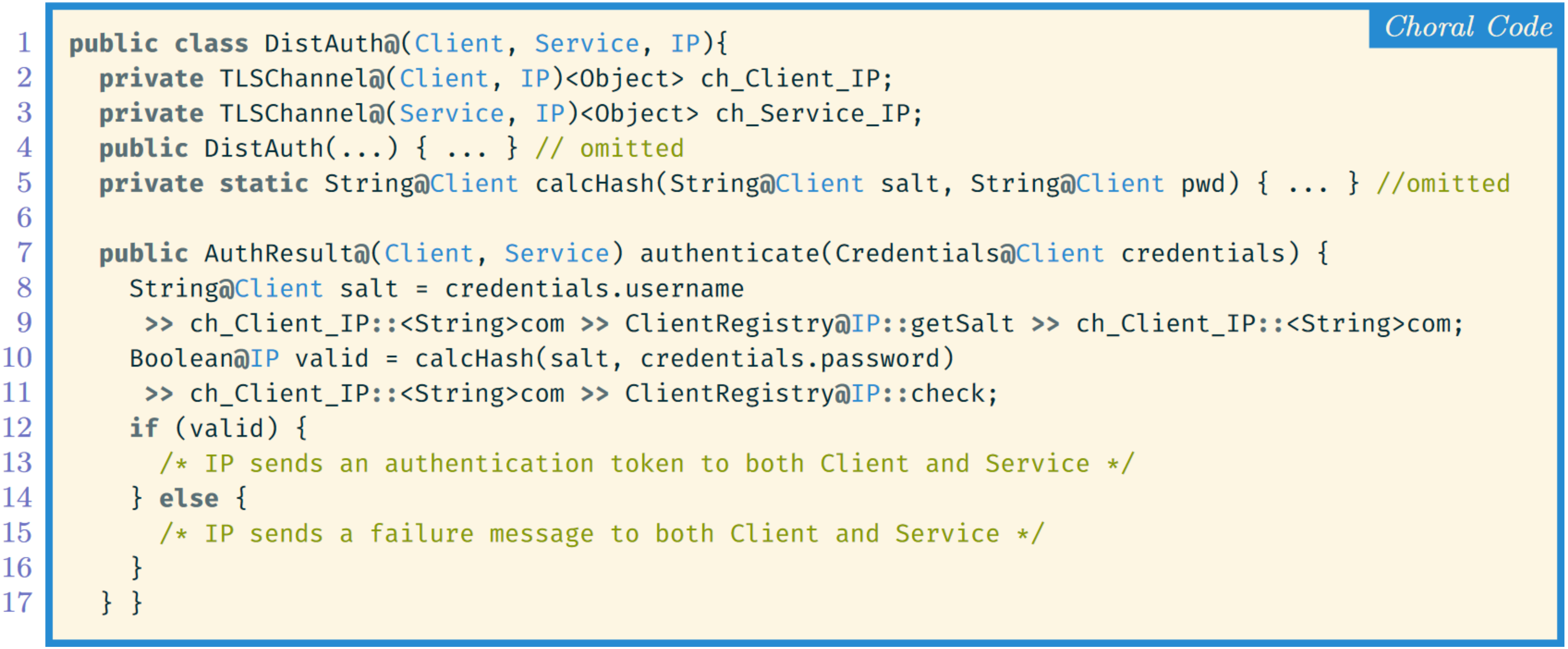


```
class DiffieHellman@( Alice, Bob ){
  public static void run(SymChannel@( Alice, Bob )< Key > channel){
    Key@Alice aPK = exp( aKpair.gen, aKpair.secret );
    Key@Bob bPK = exp( bKpair.gen, bKpair.secret );
    Key@Bob aliceKey = channel.< Key >com( aPK@Alice );
    Key@Alice bobKey = channel.< Key >com( bPK@Bob );
    Key@Alice sharedKey = exp( bobKey, aKpair.secret );
    Key@Bob sharedKey = exp( aliceKey, bKpair.secret );
  }
}
```

What about conditionals?

Distributed Authentication

Example



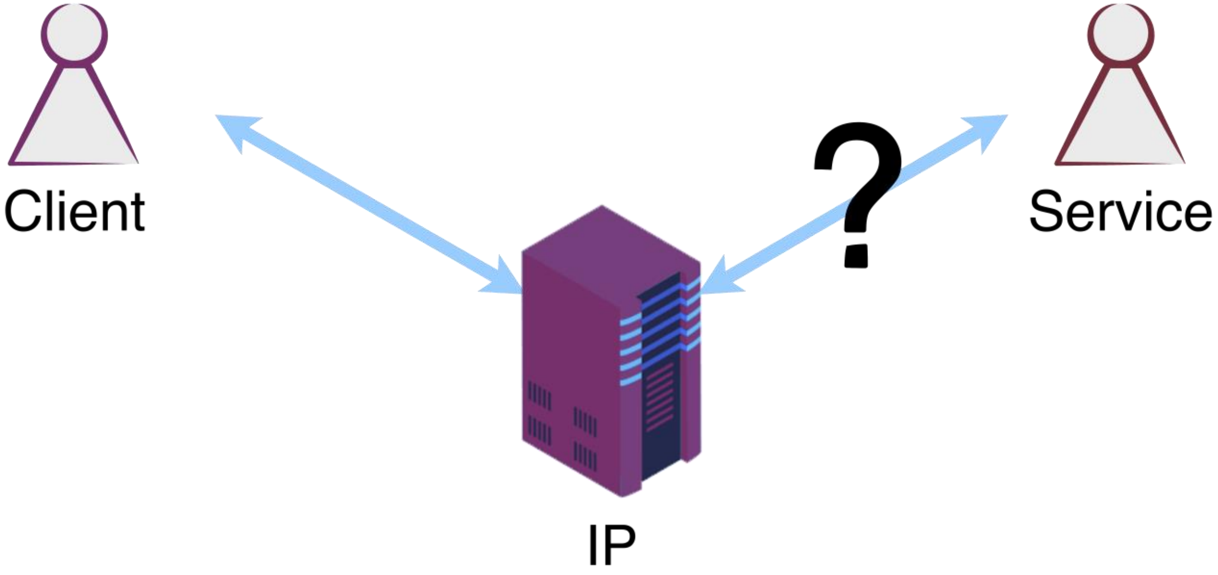
Choral Code

```
public class DistAuth@(Client, Service, IP){
  private TLSChannel@(Client, IP)<Object> ch_Client_IP;
  private TLSChannel@(Service, IP)<Object> ch_Service_IP;
  public DistAuth(...) { ... } // omitted
  private static String@Client calcHash(String@Client salt, String@Client pwd) { ... } //omitted

  public AuthResult@(Client, Service) authenticate(Credentials@Client credentials) {
    String@Client salt = credentials.username
     >> ch_Client_IP::<String>com >> ClientRegistry@IP::getSalt >> ch_Client_IP::<String>com;
    Boolean@IP valid = calcHash(salt, credentials.password)
     >> ch_Client_IP::<String>com >> ClientRegistry@IP::check;
    if (valid) {
      /* IP sends an authentication token to both Client and Service */
    } else {
      /* IP sends a failure message to both Client and Service */
    }
  } }
```

Distributed Authentication

Example



Distributed Authentication

Example

```
Boolean@IP valid = calcHash( salt, credentials.password )
    >> ch_Client_IP::< String >com
    >> ClientRegistry@IP::check;
if( valid ){


    AuthToken@IP t = AuthToken@IP.create();
    return new AuthResult@( Client, Service )(
       ch_Client_IP.< AuthToken >com( t ),
       ch_Service_IP.< AuthToken >com( t )
    );
} else {


    return new AuthResult@( Client, Service )();
    }
}
```

Distributed Authentication

Example

```
Boolean@IP valid = calcHash( salt, credentials.password )
    >> ch_Client_IP::< String >com
    >> ClientRegistry@IP::check;
if( valid ){


    AuthToken@IP t = AuthToken@IP.create();
    return new AuthResult@( Client, Service )(
       ch_Client_IP.< AuthToken >com( t ),
       ch_Service_IP.< AuthToken >com( t )
    );
} else {


    return new AuthResult@( Client, Service )();
    }
}
```

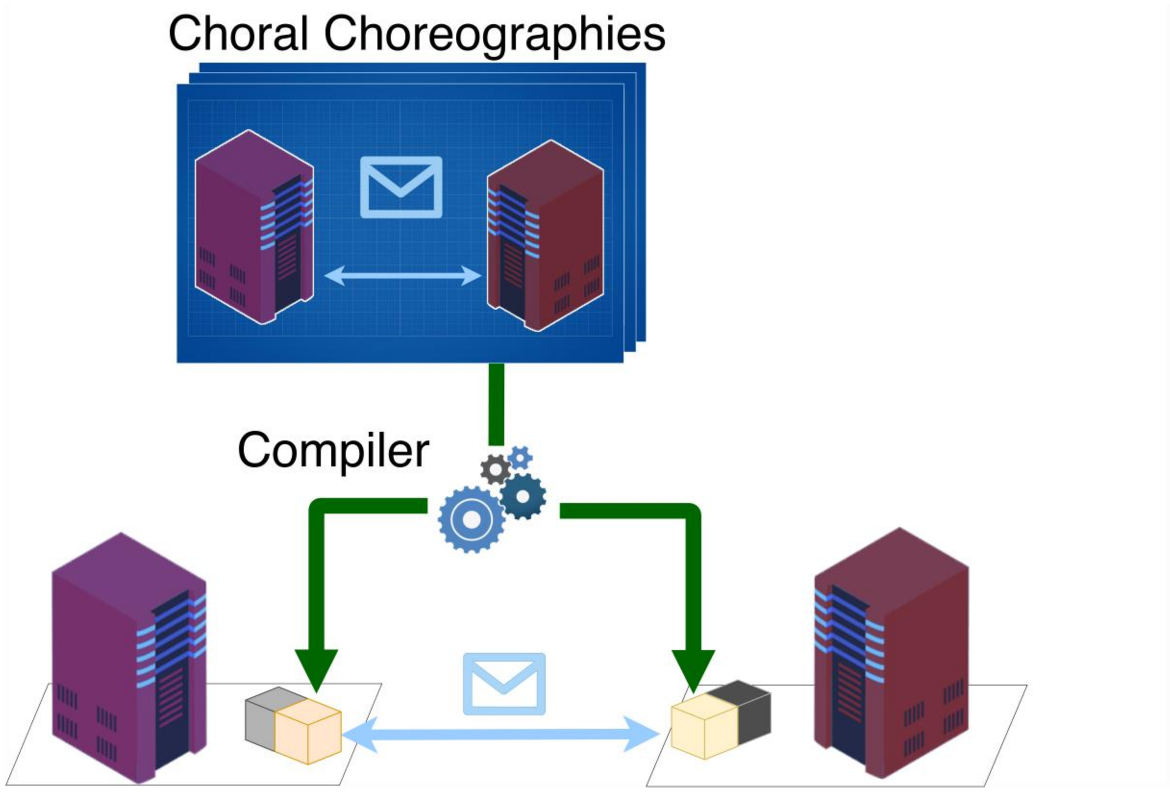
Distributed Authentication

Example

```
Boolean@IP valid = calcHash( salt, credentials.password )
    >> ch_Client_IP::< String >com
    >> ClientRegistry@IP::check;
if( valid ){
    ch_Client_IP.< EnumBoolean >select( EnumBoolean@IP.True );
    ch_Service_IP.< EnumBoolean >select( EnumBoolean@IP.True );
    AuthToken@IP t = AuthToken@IP.create();
    return new AuthResult@( Client, Service )(
       ch_Client_IP.< AuthToken >com( t ),
       ch_Service_IP.< AuthToken >com( t )
    );
} else {
    ch_Client_IP.< EnumBoolean >select( EnumBoolean@IP.False );
    ch_Service_IP.< EnumBoolean >select( EnumBoolean@IP.False );
    return new AuthResult@( Client, Service )();
    }
}
```

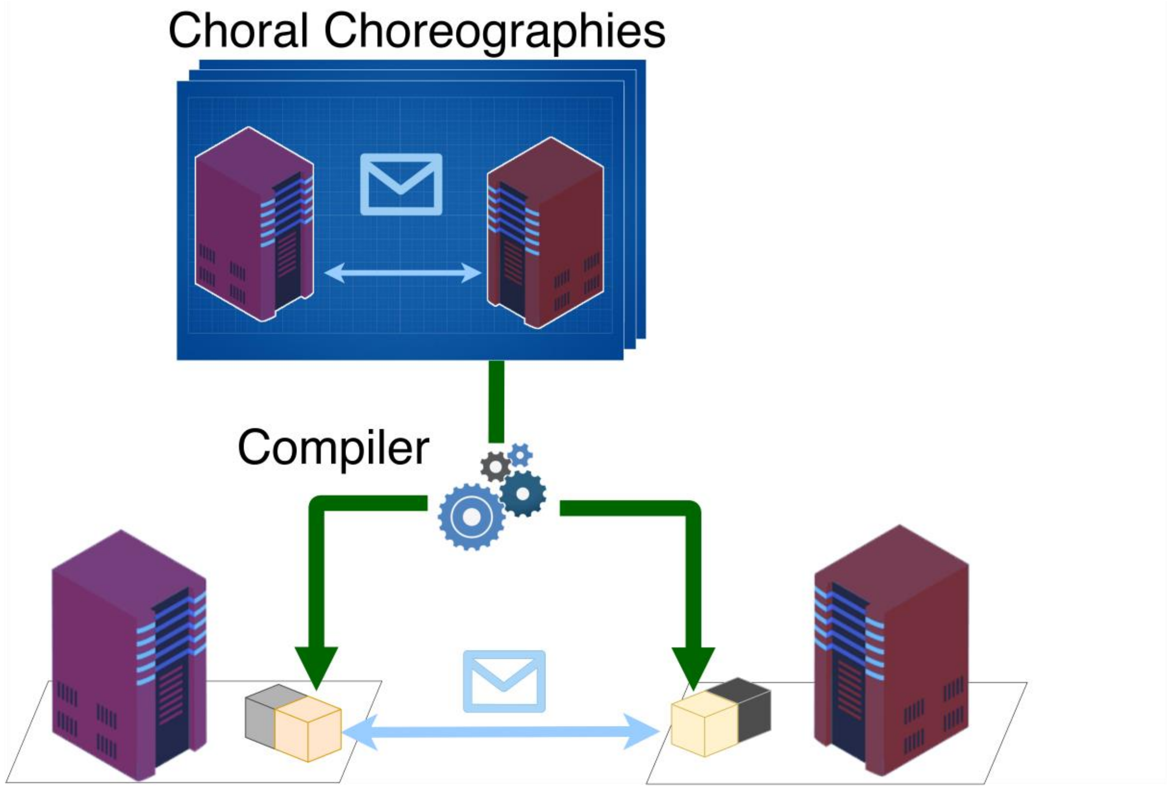
- Knowledge of Choice

Wrap things up!



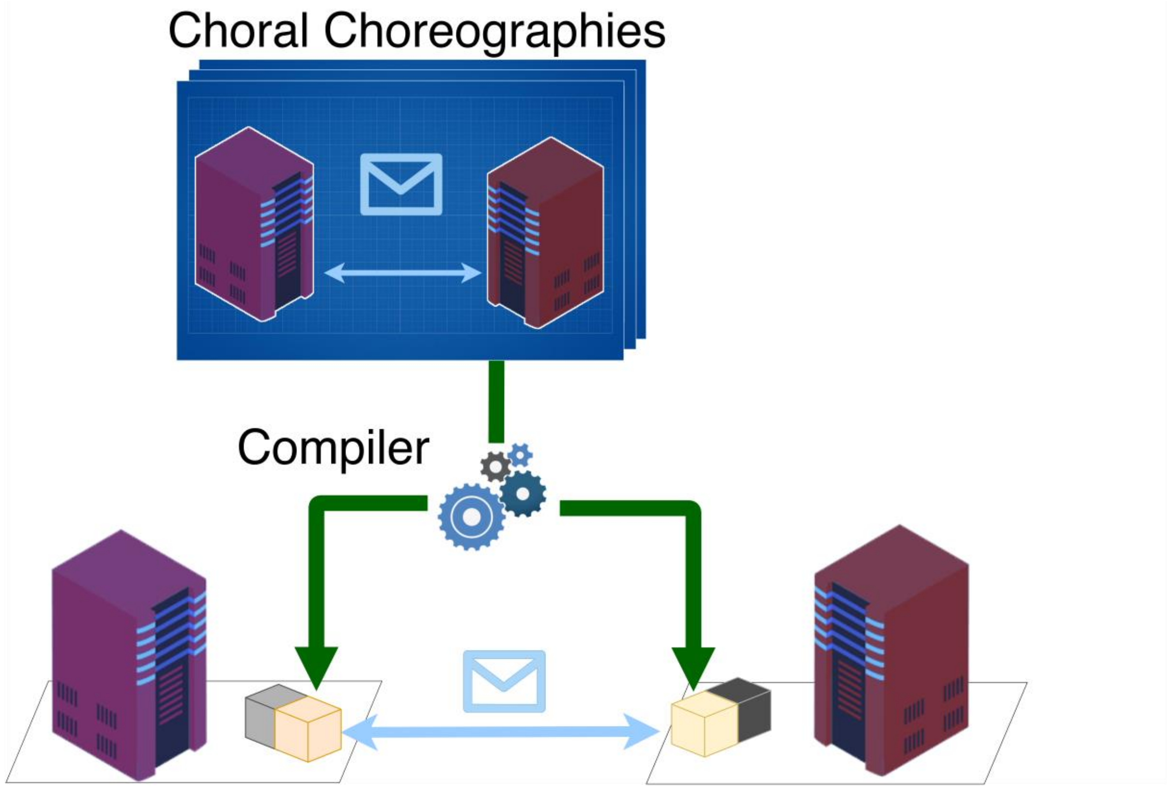
- Choreographies are objects

Wrap things up!



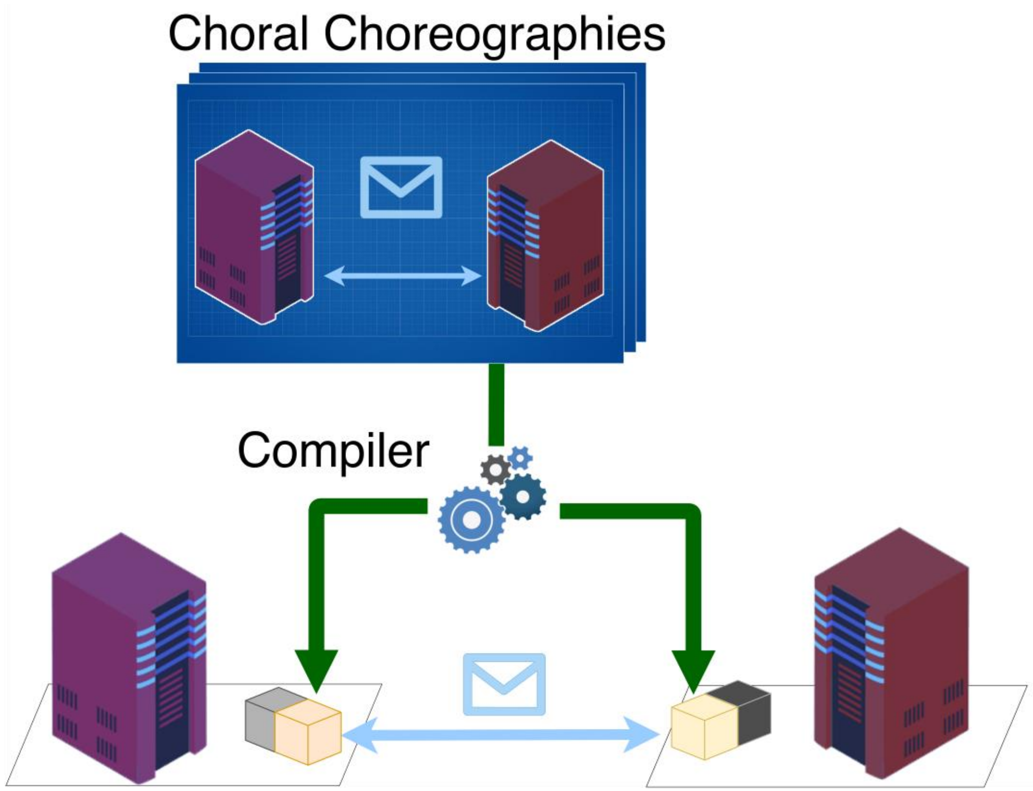
- Choreographies are objects
- Mainstream software development

Wrap things up!



- Choreographies are objects
- Mainstream software development
- Supports modularity

Wrap things up!



- Choreographies are objects
- Mainstream software development
- Supports modularity
- Possible use of multiple choreographies

Thank you for listening!

More at https://choral-lang.org