Multitier Languages for Microservice Architectures

Simon Schönwälder* - Pascal Weisenburger* - Guido Salvaneschi†

* Technical University of Darmstadt, Germany

† University of St. Gallen, Switzerland

Developing Microservices is Hard!

DISTRIBUTED SERVICES

FINE-GRANULAR SERVICES

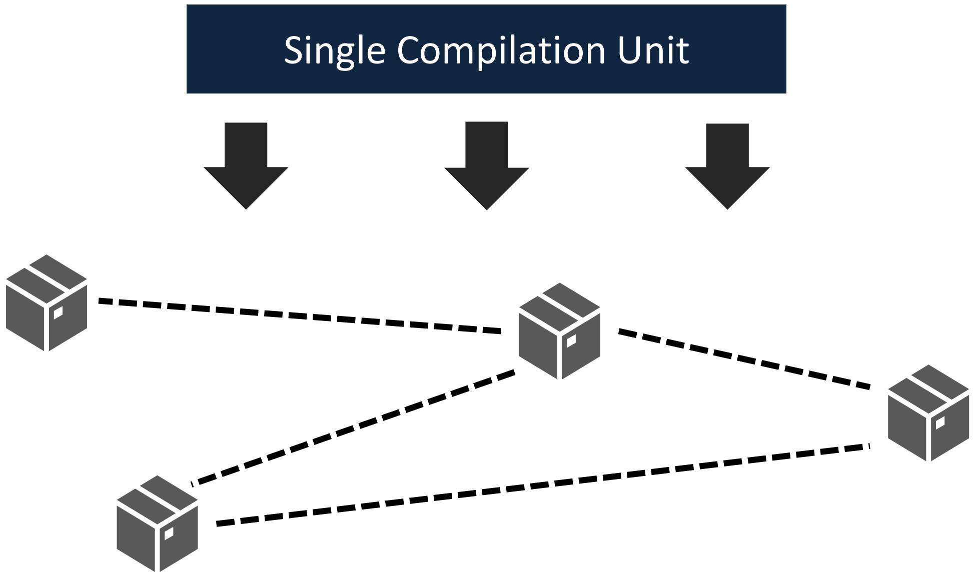
INDEPENDENT SERVICES

Communication

Coordination

Compatibility

Multitier Languages to the Rescue



Communication

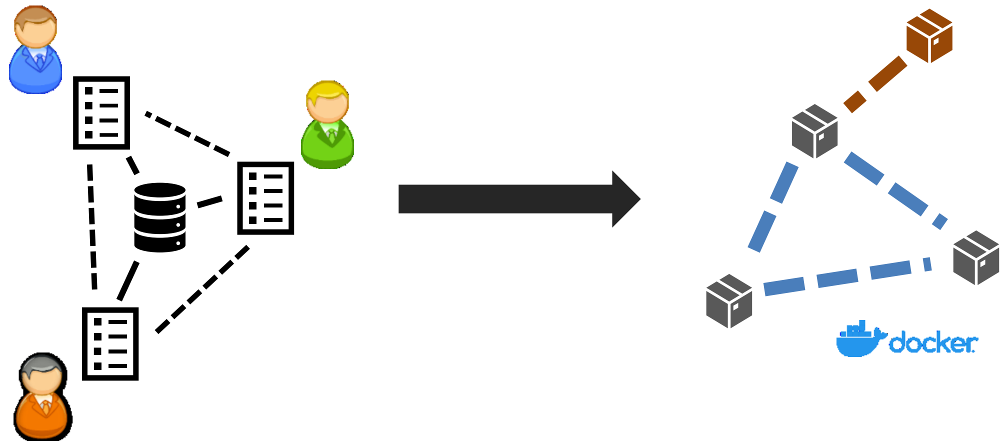
Coordination

Compatibility

Multitier Language for MSA: MSLoci

MSLoci =

ScalaLoci + *"Deploy Services as Container"*



[1] Richardson, Chris: Microservices Patterns (2019)
[2] Pascal Weisenburger, Mirko Köhler, and Guido Salvaneschi. 2018. Distributed System Development with ScalaLoci. Proceedings of the ACM on Programming Languages 2, OOPSLA, Article 129 (October 2018)

Service Implementation

Declare →

```
@multitier
trait OpenIDApi {
         @peer type OpenIDService
         def auth : Boolean on OpenIDService
}
```

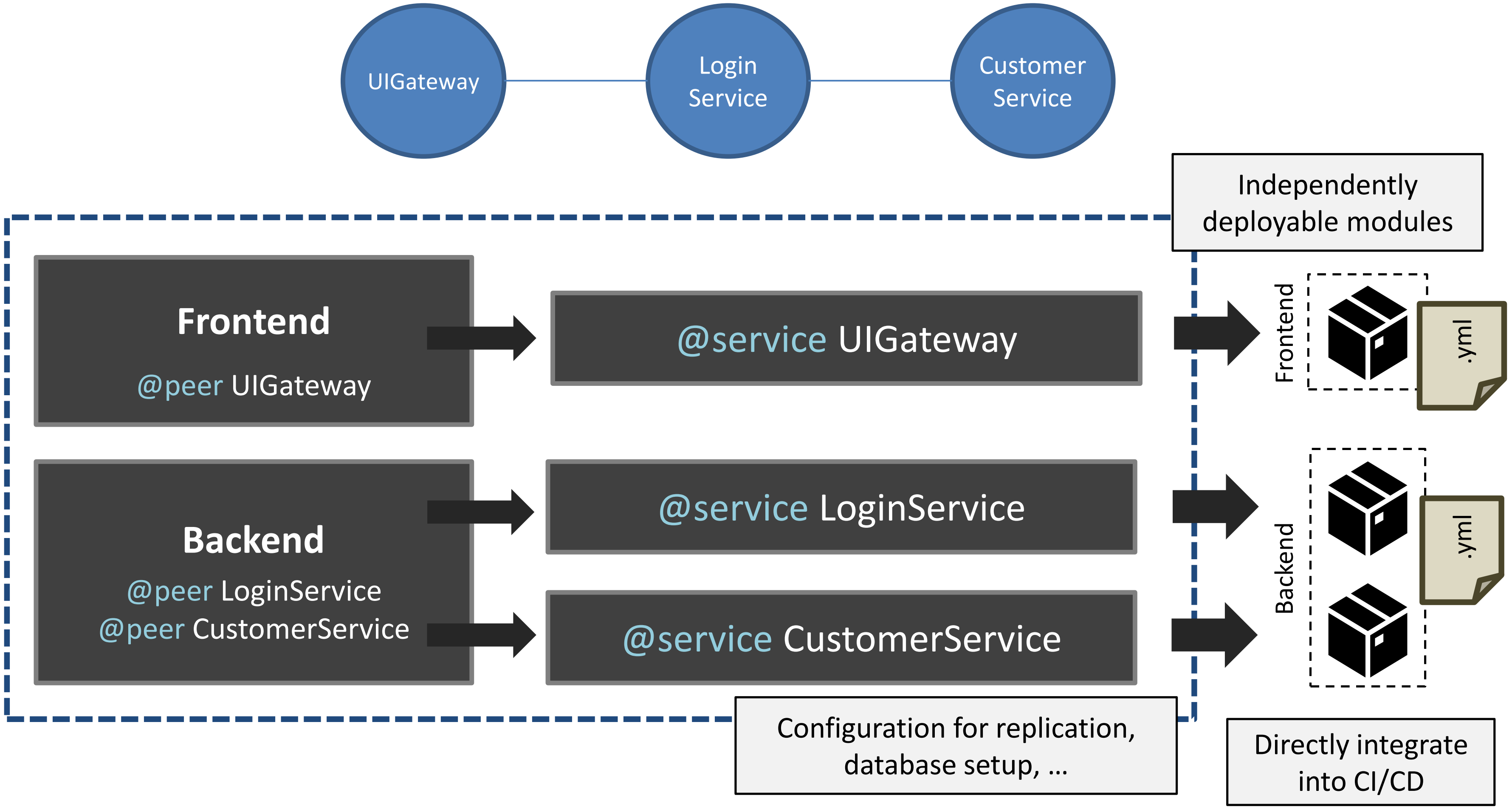
Implement →

```
@containerized @multitier
object OpenID extends OpenIDApi {
         def auth : Boolean on OpenIDService = placed{
                  … // do authentication
         }
}
```

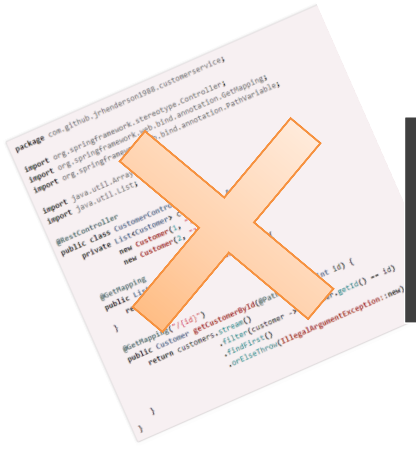
Deploy →

```
@service({ ’localdb’:’mongo’ })
object OpenIDService extends App {
multitier start new Instance[OpenID.OpenIDService](…)
}
```

Automatic Deployment Process



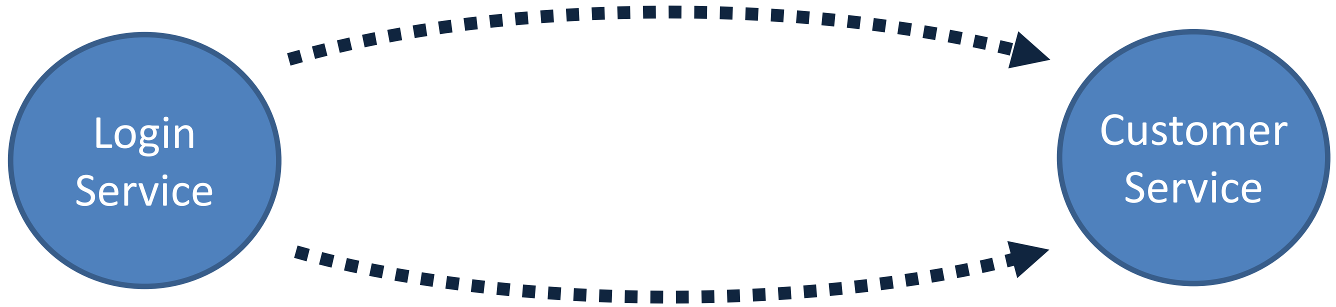
Communication made easy



```
package com.github.jrhenderson1988.customerservice;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframew[illegible].bind.annotation.PathVariable;

import java.util.Array[illegible]
import java.util.List;

@RestController
public class CustomerContro[illegible]
    private List<Customer> c[illegible]
        new Customer(1, [illegible]
        new Customer(2, [illegible]

    @GetMapping
    public Lis[illegible]
        re[illegible]
    }
    @GetMapping("/{id}")
    public Customer getCustomerById(@Pat[illegible]int id) {
        return customers.stream()
                [illegible]
                .filter(customer -> [illegible]r.getId() == id)
                .findFirst()
                .orElseThrow(IllegalArgumentException::new)


    }
}
```

```
@multitier trait Api {
        val loggedIn : Event[Customer] on LoginService
        def persist  : Unit            on CustomerService
}
```

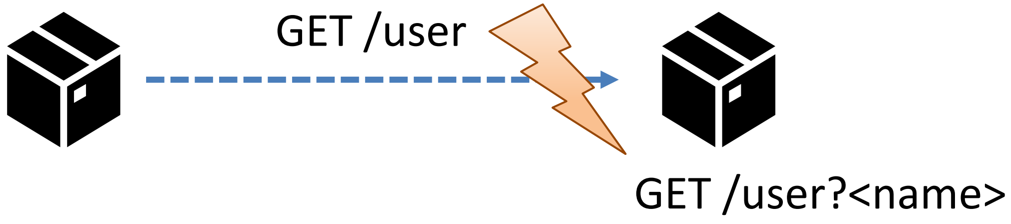
```
loggedIn.asLocal.observe(persist)
```

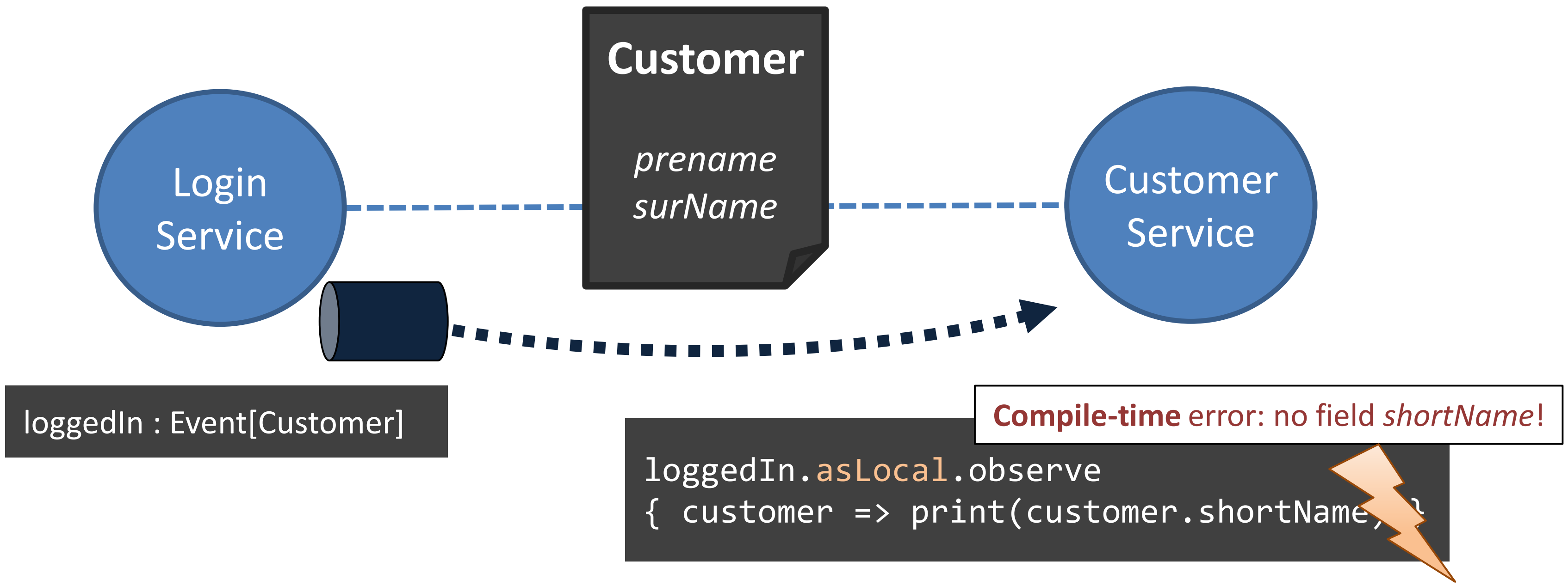


```
loggedIn observe (remote call persist)
```

Image source: *https://blog.scottlogic.com/2019/10/31/building-microservices-with-spring-boot.html*

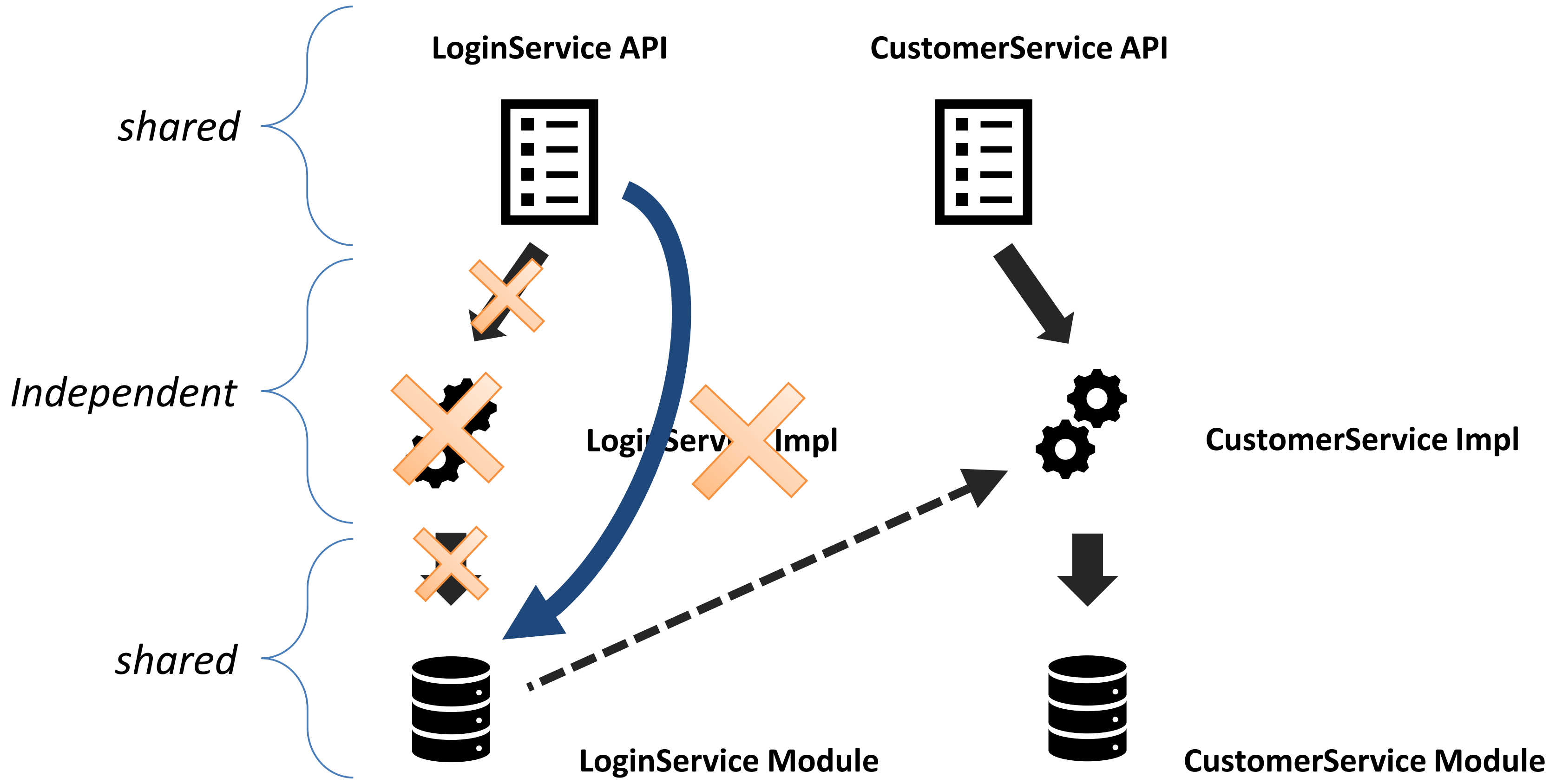
Enjoy static Guarantees



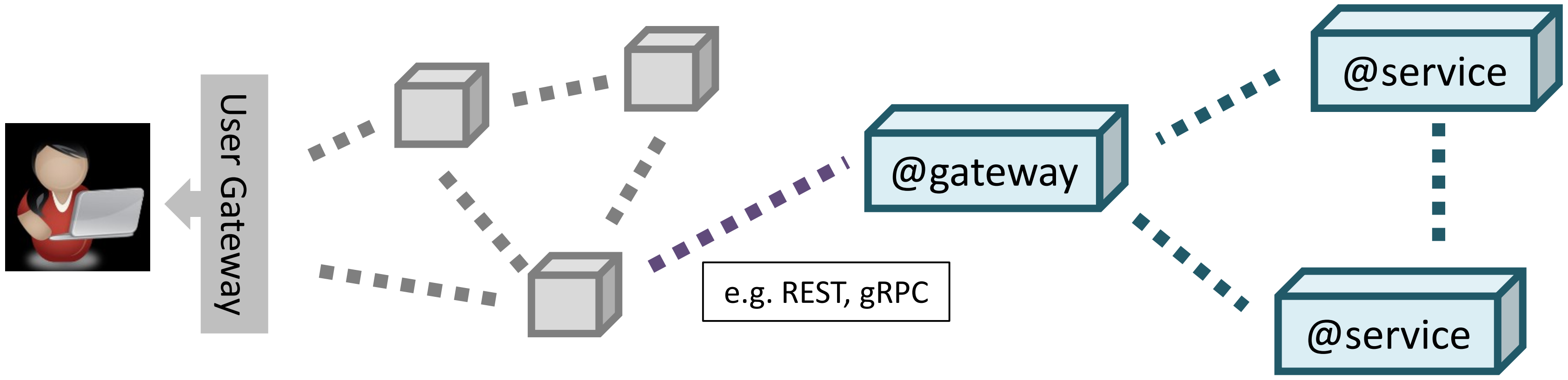

Customer
prename
surName
Login Service
Customer Service
loggedIn : Event[Customer]
Compile-time error: no field *shortName*!

```
loggedIn.asLocal.observe
{ customer => print(customer.shortName) }
```

Independent Implementation and Compilation



Integrate into existing MSAs



Let’s Recap

We propose a Multitier Language for Microservice Architectures: **MSLoci**

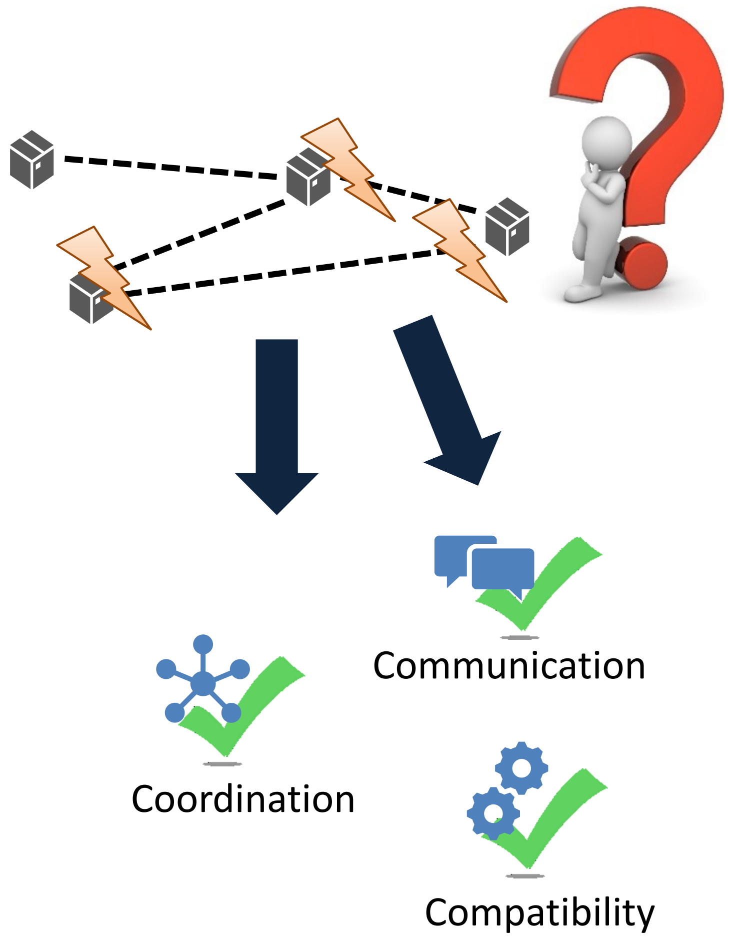
- Handles distribution & deployment complexity

→ Reduce communication boilerplate

→ Enjoy static guarantees

Thanks for your attention

Questions?


Communication
Coordination
Compatibility