

First Service &



Jolie

Basic Deployment

Saverio Giallorenzo | sgiallor@cs.unibo.it

Previously on Jolie

Service-Oriented Programming

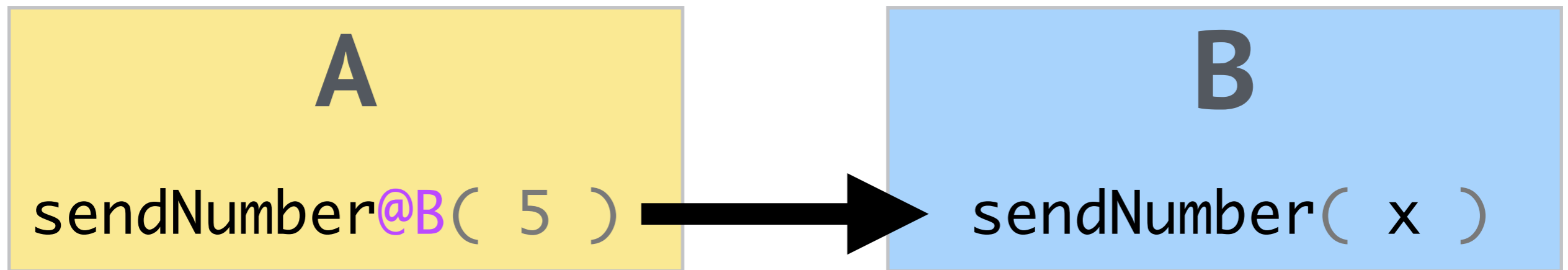
3 Commandments

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



Our first Service-Oriented Architecture

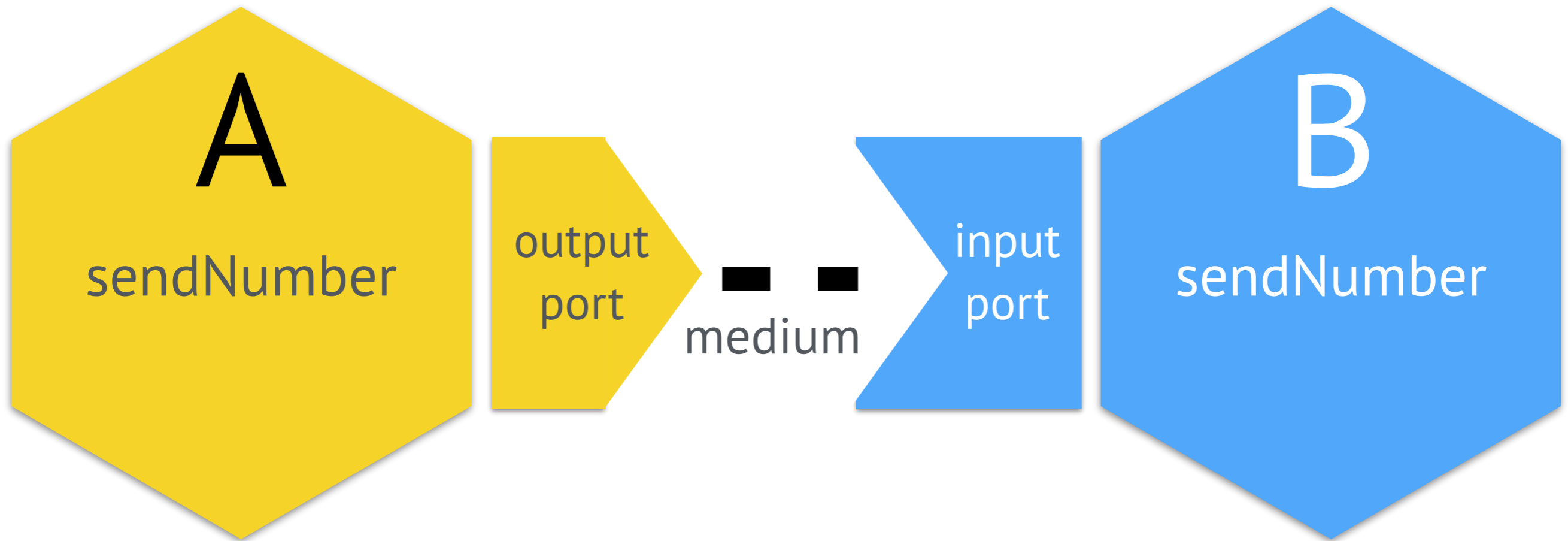
What we want



What we need to define (in general, not just in Jolie)

- **How to reach B** from A;
- **How B offers** (exposes) `sendNumber`;
- and much much more, but let us stick with these right now

How services communicate



- 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.

Let us reason a bit before ...



... coding everything down

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

```
main
{
  sendNumber( x )
}
```

... coding everything down

```
outputPort B {
  Location:
    "socket://localhost:8000"
  Protocol: sodep
  Interfaces: MyInterface
}

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

```
inputPort B {
  Location:
    "socket://localhost:8000"
  Protocol: sodep
  Interfaces: MyInterface
}

main
{
  sendNumber( x )
}
```


... coding everything down

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

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

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

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

main
{
    sendNumber( x )
}
```

... coding everything down

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

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

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

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

main
{
  sendNumber( x )
}
```

A closer look on ports

- A port specifies:
- the **location** on which the communication can take place;
- the **protocol** to use for encoding/decoding data;
- the **interfaces** it exposes.

```
outputPort B {  
  Location:  
    "socket://localhost:8000"  
  Protocol: sodep  
  Interfaces: MyInterface  
}
```

```
inputPort B {  
  Location:  
    "socket://localhost:8000"  
  Protocol: sodep  
  Interfaces: MyInterface  
}
```

There is no limit to how many ports a service can use.

A closer look on ports - Locations

A location describes:

- the **communication medium**;
- the **parameters** to set the communication up.

In Jolie a **location** is a **Uniform Resource Identifier (URI)**
with form: **medium[:parameters]**



	Medium	Parameters
TCP/IP	socket://	www.google.it:80
Bluetooth	bt2cap://	localhost: 3B9FA89520078C303355AAA694238F07;name=Vision;encrypt= false;authenticate=false
Local	localsocket:	/tmp/mysocket.socket
Java RMI	rmi://	myRmiUrl.com/MyService
In-Memory	local	

A closer look on ports - Protocols

A protocol defines the format the data is sent (**encoded**) and received (**encoded**)

In Jolie protocols are names and possibly additional parameters:

```
json/rpc
```

```
sodep
```

```
https
```

```
soap
```

```
http { .debug = true }
```

A closer look on ports - a http request

```
test@Out( { .params = "Hello all!", .id = 99 } )
```

- ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
- ▶ Transmission Control Protocol, Src Port: 51371 (51371), Dst Port: 8000 (8000), Seq: 1, Ack: 1, Len: 204
- ▼ Hypertext Transfer Protocol
 - ▶ POST /test HTTP/1.1\r\n
 - Host: localhost\r\n
 - Accept-Encoding: gzip, deflate\r\n
 - X-Jolie-MessageID: 1\r\n
 - Content-Type: text/xml; charset=utf-8\r\n
 - ▶ Content-Length: 51\r\n
 - \r\n
 - [\[Full request URI: http://localhost/test\]](http://localhost/test)
 - [HTTP request 1/1]
 - [\[Response in frame: 13\]](#)
- ▼ eXtensible Markup Language
 - ▼ <test>
 - ▼ <id>
 - 99
 - </id>
 - ▼ <params>
 - Hello all!
 - </params>
 - </test>

A closer look on ports - Interfaces

```
inputPort id {  
  Location: URI  
  Protocol: p  
  Interfaces: iface_1,  
             ...,  
             iface_n  
}
```

```
outputPort id {  
  Location: URI  
  Protocol: p  
  Interfaces: iface_1,  
             ...,  
             iface_n  
}
```

A closer look on ports - Interfaces

```
interface InterfaceName {  
    OneWay:  
        ow_name1( MyType1 ),  
        ow_name2( basicType ),  
        //...,  
        ow_nameN( MyTypeN )  
    RequestResponse:  
        rr_name1( MyType1 )( MyType2 ),  
        rr_name2( basicType )( MyType3 ),  
        //...  
        rr_nameN( basicType )( MyTypeN )  
}
```


A closer look on ports - Basic Types

- `bool`: booleans;
- `int`: integers;
- `long`: long integers (with “L” or “l” suffix);
- `double`: double-precision float (decimal literals);
- `string`: strings;
- `raw`: byte arrays;
- `void`: the empty type.
- `undefined`

```
type T: basicType
```

e.g.,

```
type MyType: string
```

A closer look on ports - Custom Types

```
type T: basic_type {  
  .subnode1[R]: (basic_type OR Ti)  
  ...  
  .subnodeN[R]: (basic_type OR Ti)  
}
```

```
type myType: string {  
  .x[ 1, * ]: mySubType  
  .y[ 1, 3 ]: void {  
    .value*: double  
    .comment: string  
  }  
  .z?: void { ? }  
}
```

```
type mySubType: void {  
  .value: double  
  .comment: string  
}
```