Jolie Microservices and Choreographies for the Web

Saverio Giallorenzo | sgiallor@cs.unibo.it

Service-Oriented Programming

3 Commandments:

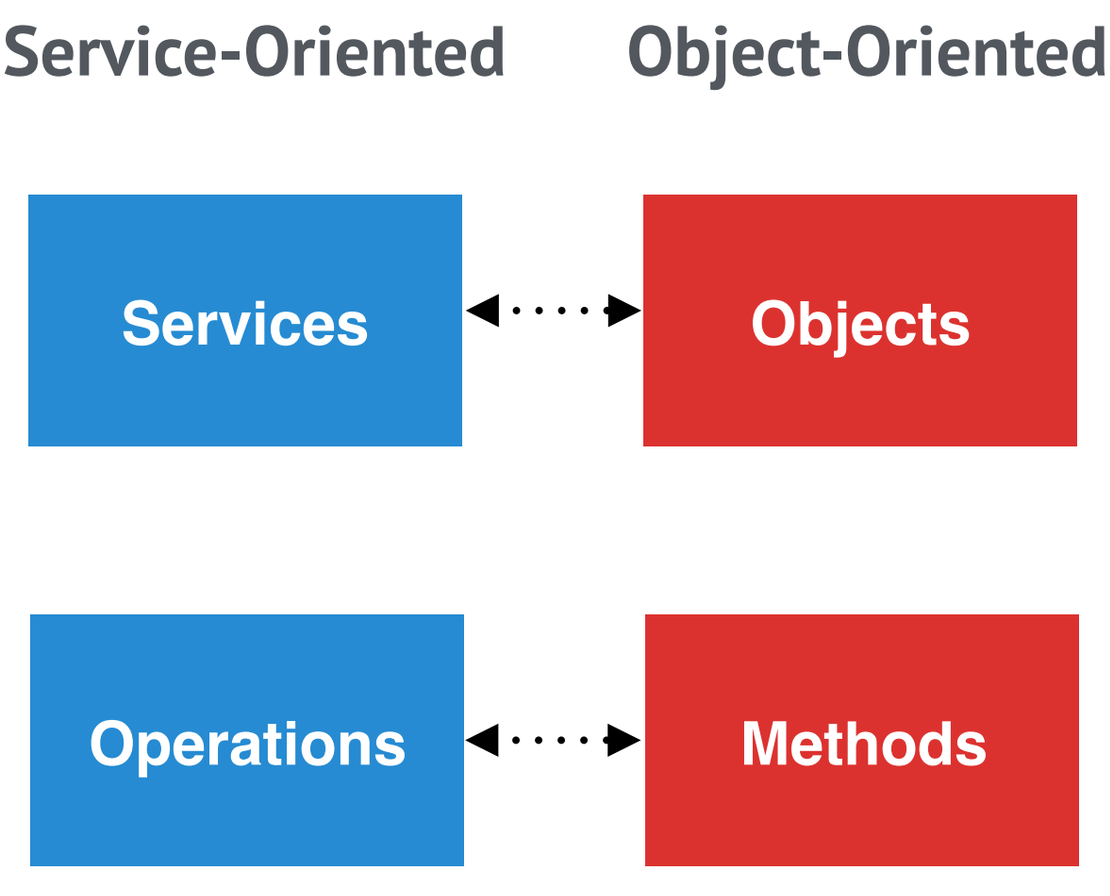
- Everything is a **service**;
- A service is an application that offers **operations**;
- A service can **invoke** another service by calling one of its operations.

Service-Oriented Programming

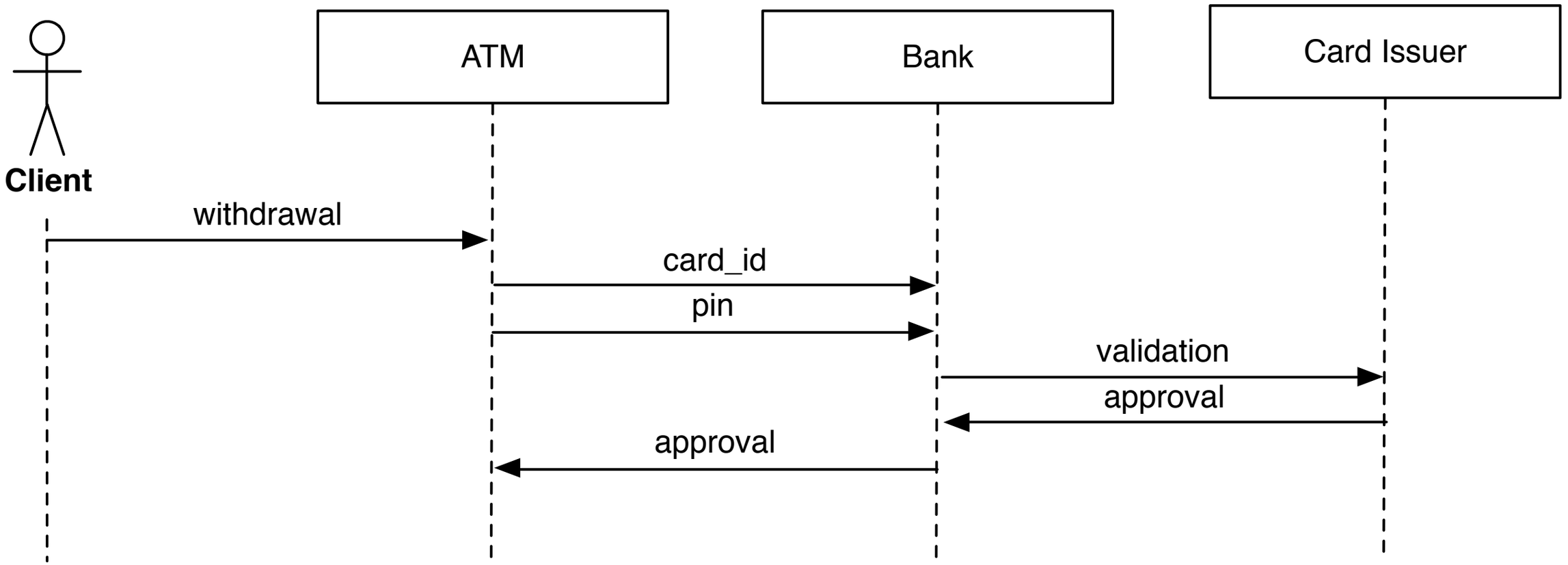
3 Commandments

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

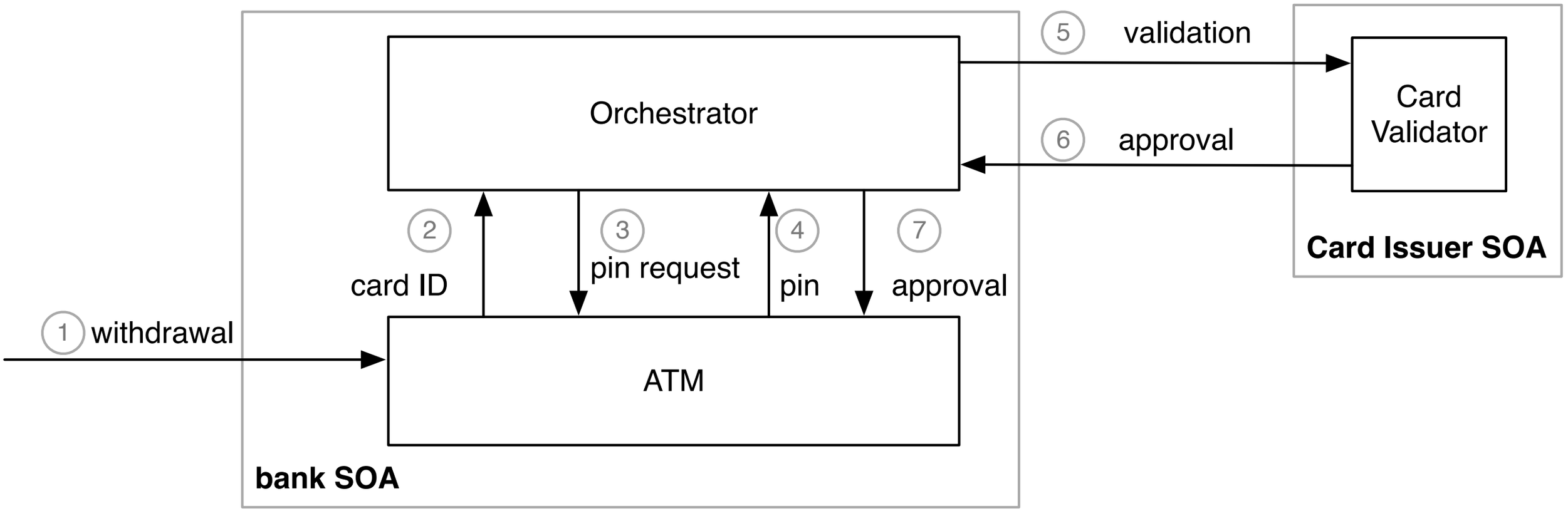
Recalling the Object-Oriented creed



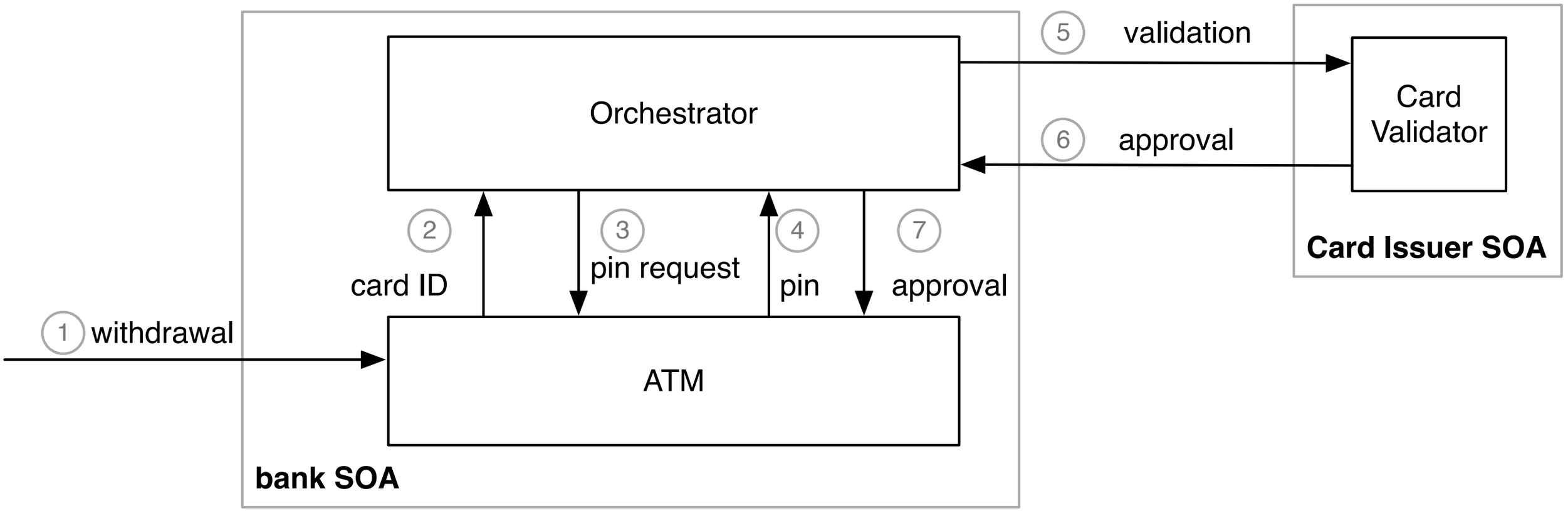
A simple distributed system



Orchestration



Orchestration



Process-Oriented

Why SOC and Jolie?

Need for languages that express complex compositions of services.

Why SOC and Jolie?

Need for languages that express complex compositions of services.

But I already know Java!
Why shall I use Jolie?

Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
  socketChannel.connect(
new InetSocketAddress("http://someurl.com", 80));
  Buffer buffer = . . .; // byte buffer
  while( buffer.hasRemaining() ) {
     channel.write( buffer );
}
```

Happy?

Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
  socketChannel.connect(
new InetSocketAddress("http://someurl.com", 80));
  Buffer buffer = . . .; // byte buffer
  while( buffer.hasRemaining() ) {
     channel.write( buffer );
}
```

Happy?

Ok, but you did not even close the channel or handled exceptions

Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
try {
  socketChannel.connect(new InetSocketAddress("http://someurl.com",
80));
  Buffer buffer = . . .; // byte buffer
  while( buffer.hasRemaining() ) {
    channel.write( buffer );
} }
catch( UnresolvedAddressException e ) { . . . }
catch( SecurityException e ) { . . . }
/* . . . many catches later . . . */
catch( IOException e ) { . . . }
finally { channel.close(); }
```

Happier now?

Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
try {
  socketChannel.connect(new InetSocketAddress("http://someurl.com",
80));
  Buffer buffer = . . .; // byte buffer
  while( buffer.hasRemaining() ) {
    channel.write( buffer );
} }
catch( UnresolvedAddressException e ) { . . . }
catch( SecurityException e ) { . . . }
/* . . . many catches later . . . */
catch( IOException e ) { . . . }
finally { channel.close(); }
```

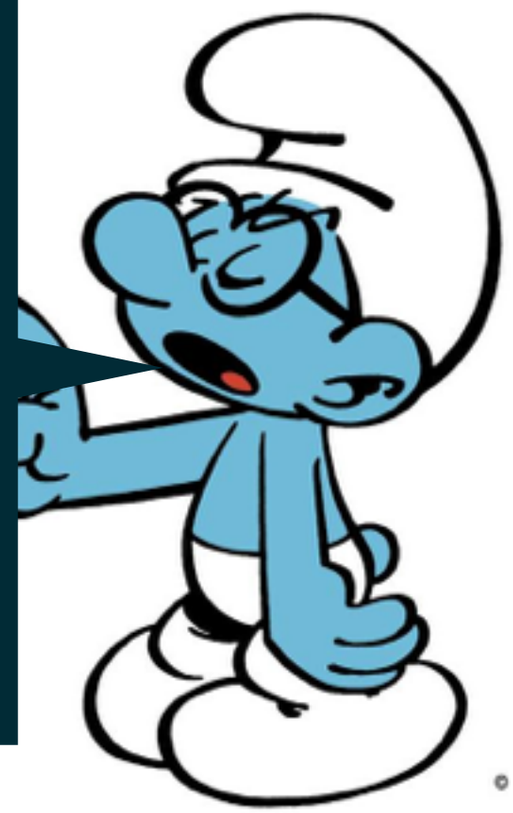
Happier now?

Yes, but what about the **server**?

Why SOC and Jolie?

```
Selector selector = Selector.open();
channel.configureBlocking(false);
SelectionKey key = channel.register(selector, SelectionKey.OP_READ);
while(true) {
  int readyChannels = selector.select();
  if(readyChannels == 0) continue;
  Set<SelectionKey> selectedKeys = selector.selectedKeys();
  Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
  while(keyIterator.hasNext()) {
    SelectionKey key = keyIterator.next();
    if(key.isAcceptable()) {
        // a connection was accepted by a ServerSocketChannel.
    } else if (key.isConnectable()) {
        // a connection was established with a remote server.
    } else if (key.isReadable()) {
        // a channel is ready for reading
    } else if (key.isWritable()) {
        // a channel is ready for writing
    }
    keyIterator.remove();
  }
}
```

Here you are



Why SOC and Jolie?

Well, ok, but again, you are not **handling exceptions**.
And what about if **different operations** use the **same channel**?
And if we wanted to use **RMIs** instead of **Sockets**?
In what **format** are you transmitting data? And if we need to **change** the **format** after we wrote the application? Do you **check** the **type of data** you receive/send?

Why SOC and Jolie?

Well, ok, but again, you are not **handling exceptions**.
And what about if **different operations** use the **same channel**?
And if we wanted to use **RMIs** instead of **Sockets**?
In what **format** are you transmitting data? And if we need to **change** the **format** after we wrote the application? Do you **check** the **type of data** you receive/send?

Why SOC and Jolie?

Programming distributed systems is usually harder than programming non distributed ones.

Concerns of **concurrent** programming.

Plus (not exhaustive):

- handling **communications**;
- handling **heterogeneity**;
- handling **faults**;
- handling the **evolution** of systems.


Jolie

A **Service-Oriented Orchestration** Programming Language

Resources | **Online**

- Official Website:
  - **http://www.jolie-lang.org**
- Official Docs:
  - **http://docs.jolie-lang.org**
- Official Codebase:
  - **https://github.com/jolie/jolie**

Hello World! in Jolie

"Hello World!" is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
  println@Console( "Hello, world!" )()
}
```

Hello World! in Jolie

"Hello World!" is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
  println@Console( "Hello, world!" )()
}
```

Include a service

Hello World! in Jolie

"Hello World!" is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
  println@Console( "Hello, world!" )()
}
```

Include a service

program entry point

Hello World! in Jolie

"Hello World!" is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
  println@Console( "Hello, world!" )()
}
```

Include a service

program entry point

operation

Hello World! in Jolie

"Hello World!" is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
  println@Console( "Hello, world!" )()
}
```

Include a service

program entry point

operation

service

A more interesting example

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

A

B

```
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 more interesting example

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

A

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

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

B

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

main
{
   sendNumber( x )
}
```

A more interesting example

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

A

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

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

B

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

main
{
   sendNumber( x )
}
```

A more interesting example

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

A

B

```
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 more interesting example

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

A

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

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

B

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

main
{
   sendNumber( x )
}
```

A more interesting example

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

A

B

```
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 more interesting example

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

A

B

```
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 more interesting example

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

A

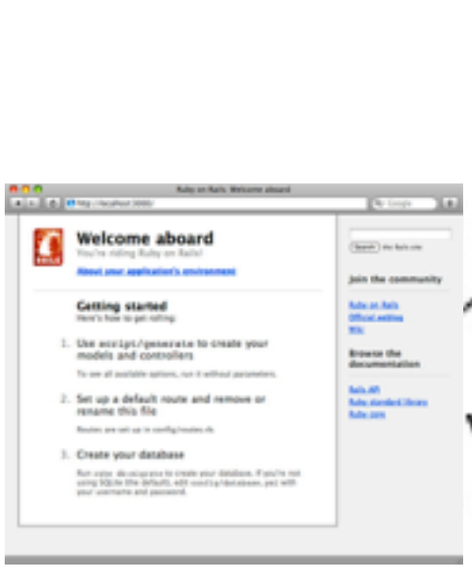
```
include "MyInterface.iol"
outputPort B {
Location:
   "socket://localhost:8000"
Protocol: sodep
Interfaces: MyInterface
}

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

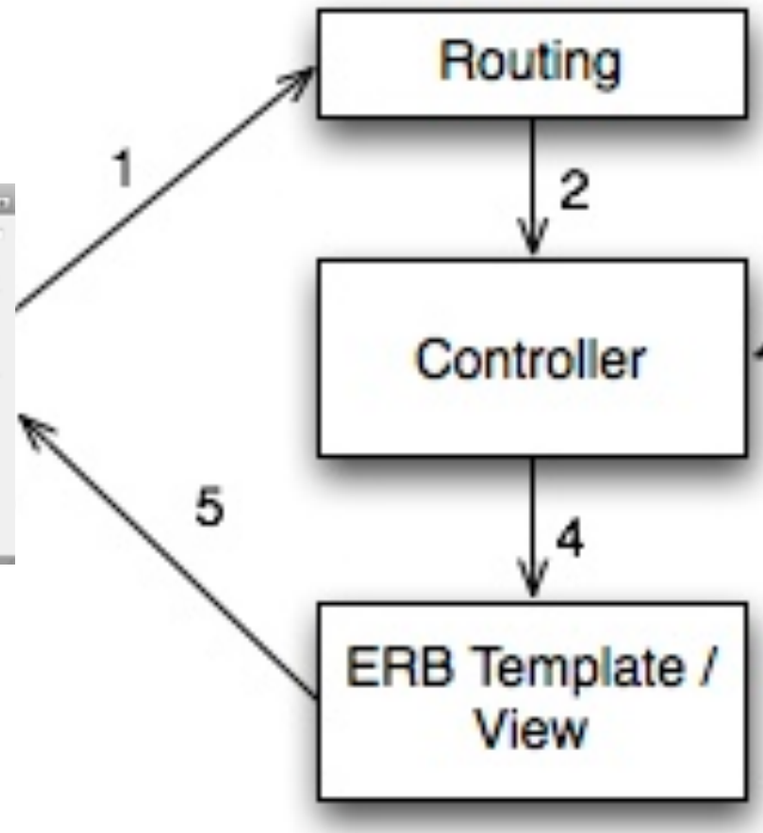
B

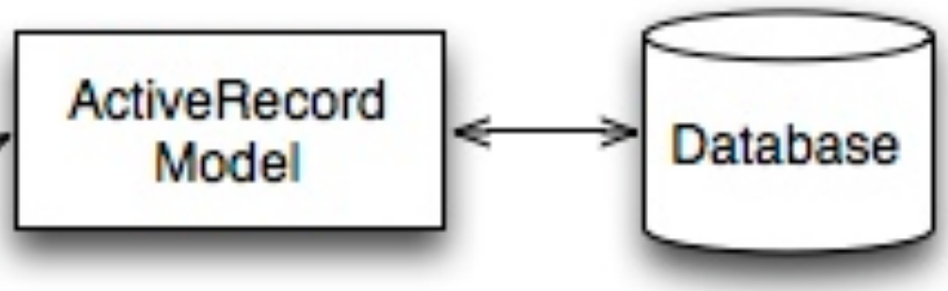
```
include "MyInterface.iol"
inputPort B {
Location:
   "socket://localhost:8000"
Protocol: sodep
Interfaces: MyInterface
}

main
{
   sendNumber( x )
}
```

Can service-orientation aid web programming?

SOC already snuck into web programming

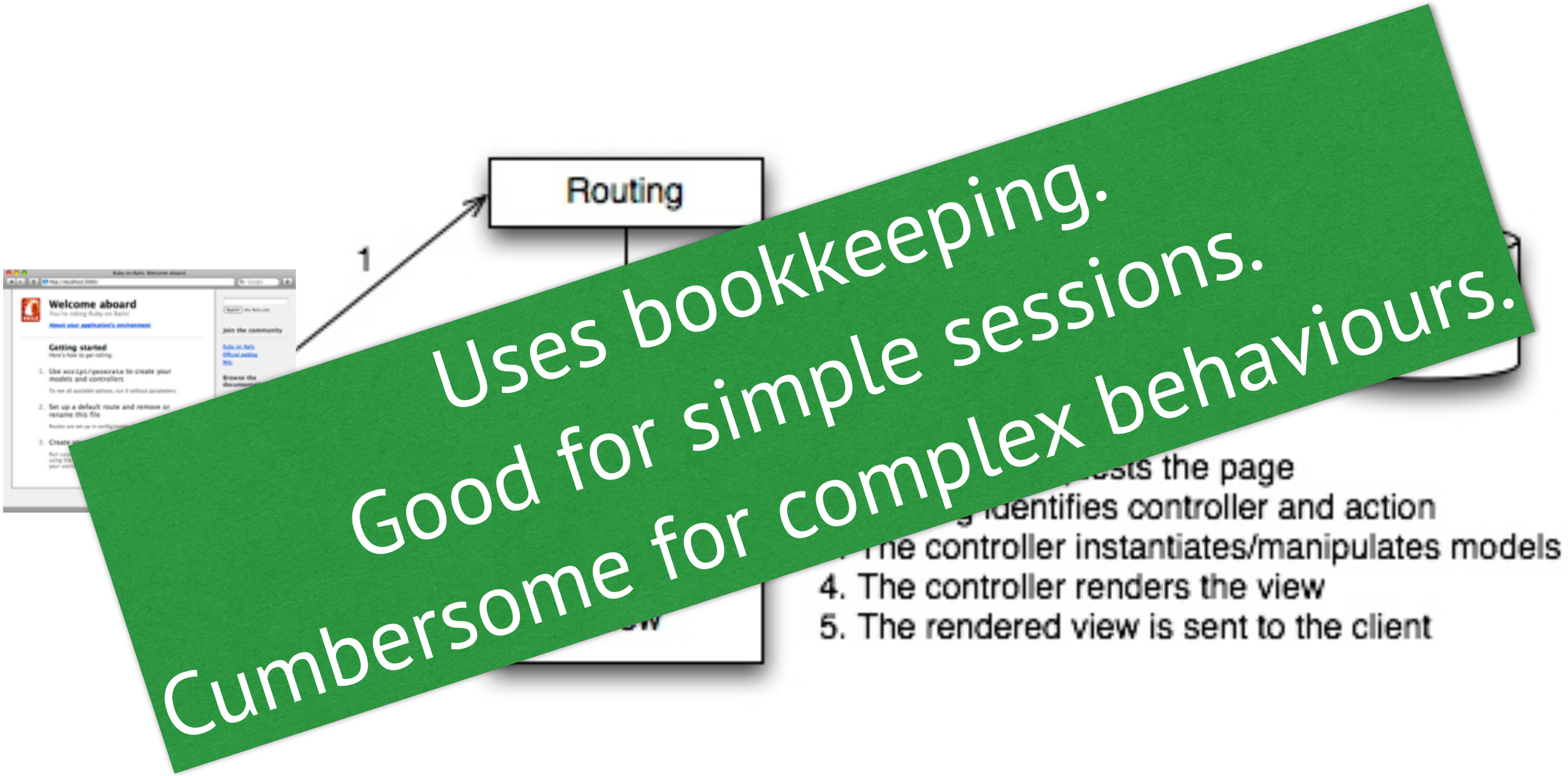




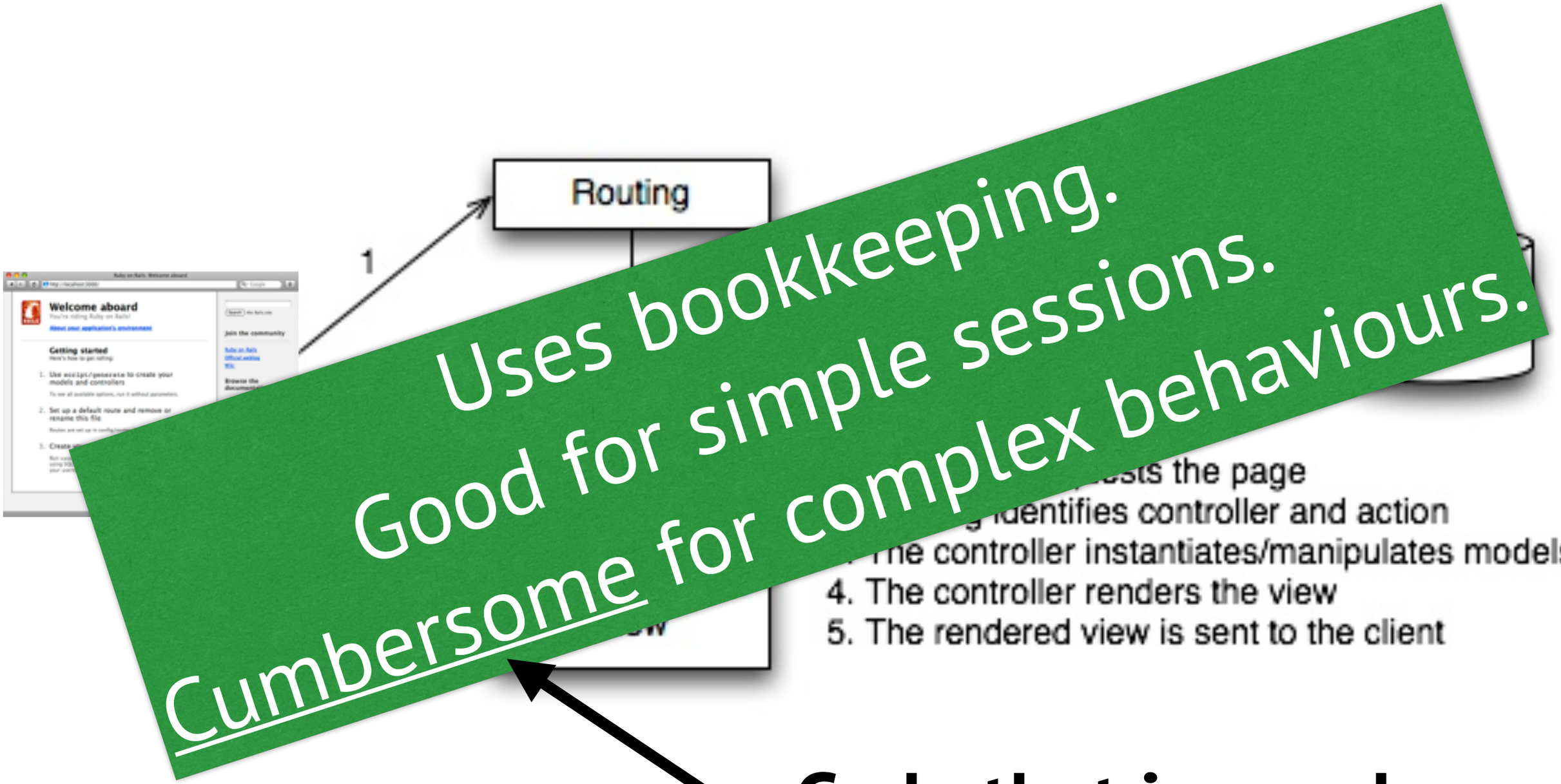


1. The Client requests the page
2. Routing identifies controller and action
3. The controller instantiates/manipulates models
4. The controller renders the view
5. The rendered view is sent to the client

SOC already snuck into web programming

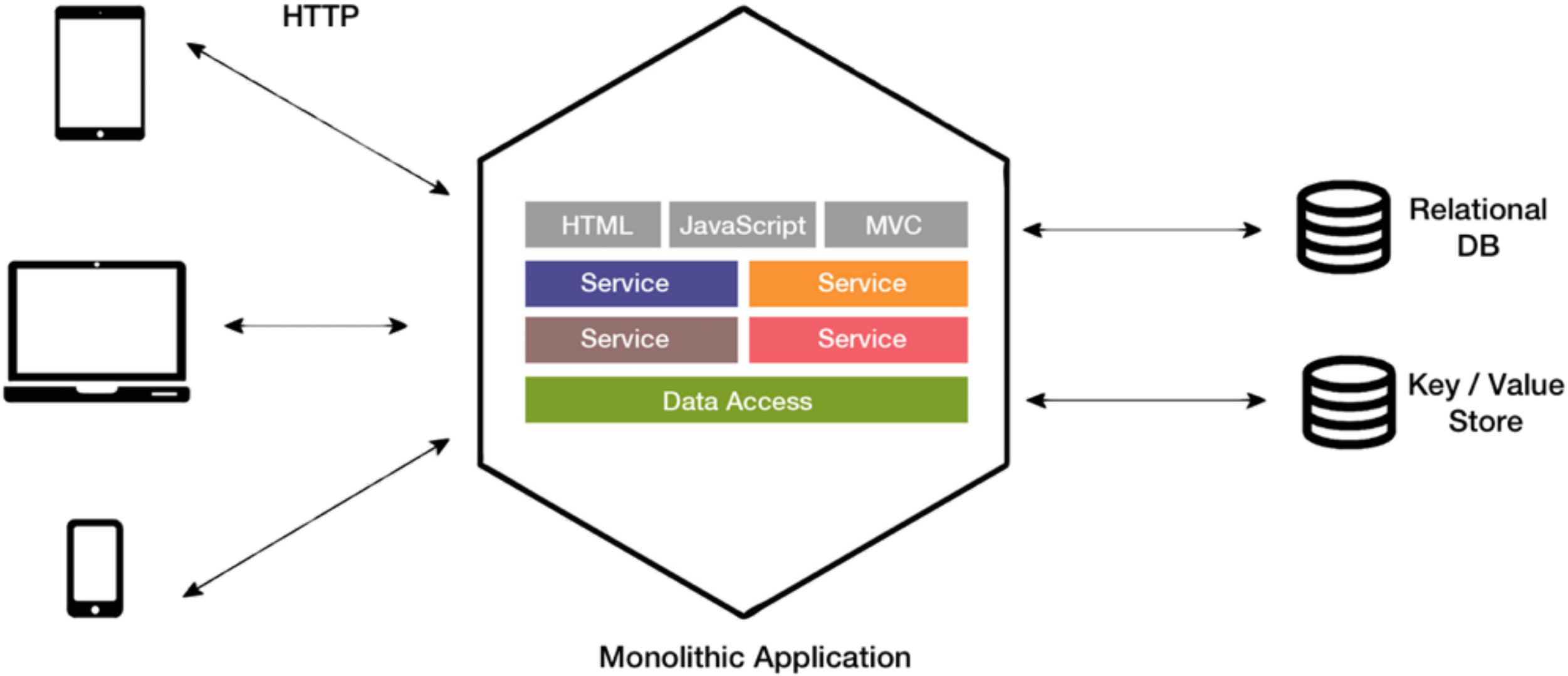


SOC already snuck into web programming



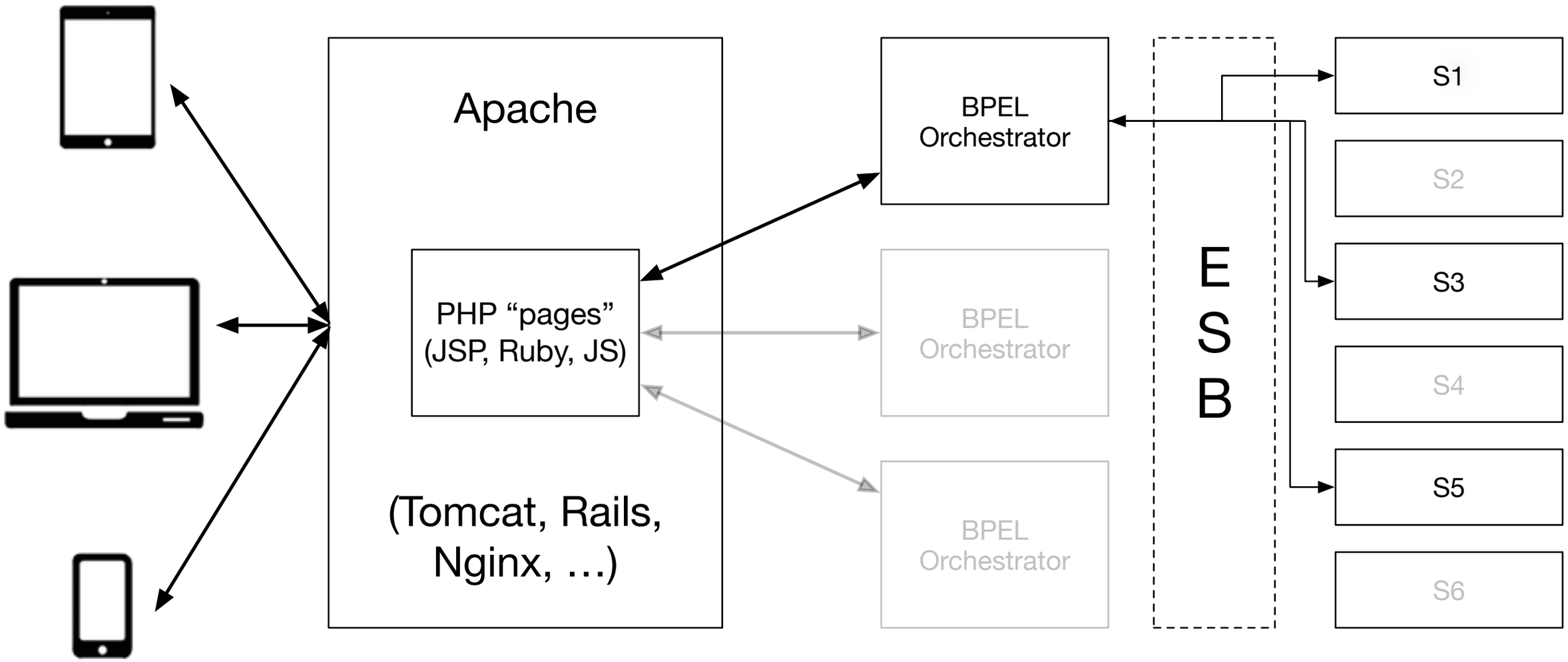
Code that is poorly readable and hard to maintain

SOC already snuck into web programming



Copyright Emiliano Mancuso | http://bits.citrusbyte.com/microservices/

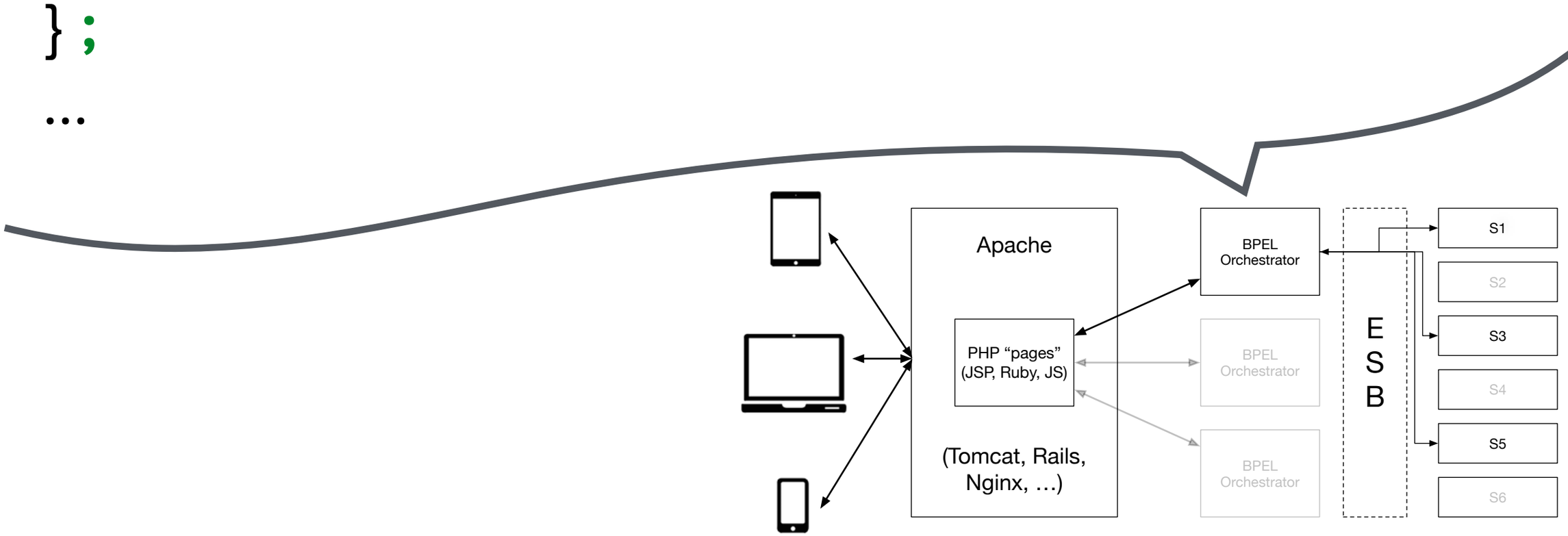
SOC already snuck into web programming



Multi-layered architecture

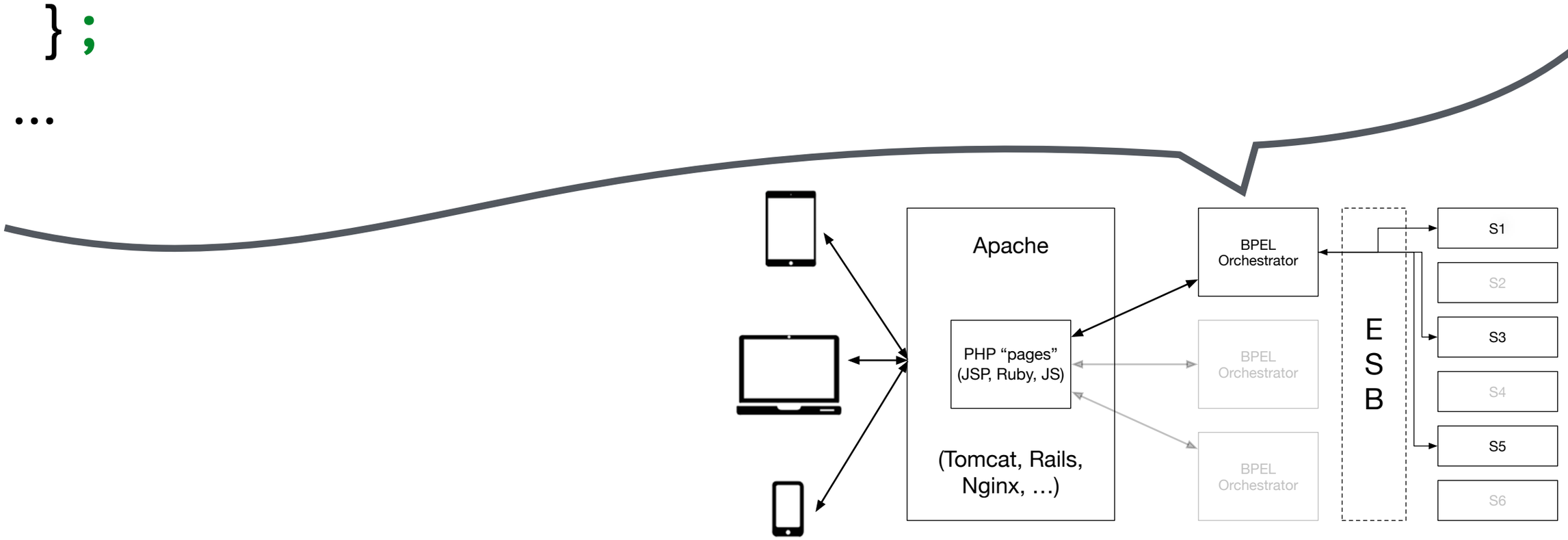
SOC already snuck into web programming

```
startSession( req )( sid ) { login@S1( req )( sid ) };
 {
    userApproval( u )( ru ){   userApproval@S3( u )( ru )   }
    |
    adminApproval( a )( ra ){   adminApproval@S5( a )( ra )   }
 };
...
```



SOC already snuck into web programming

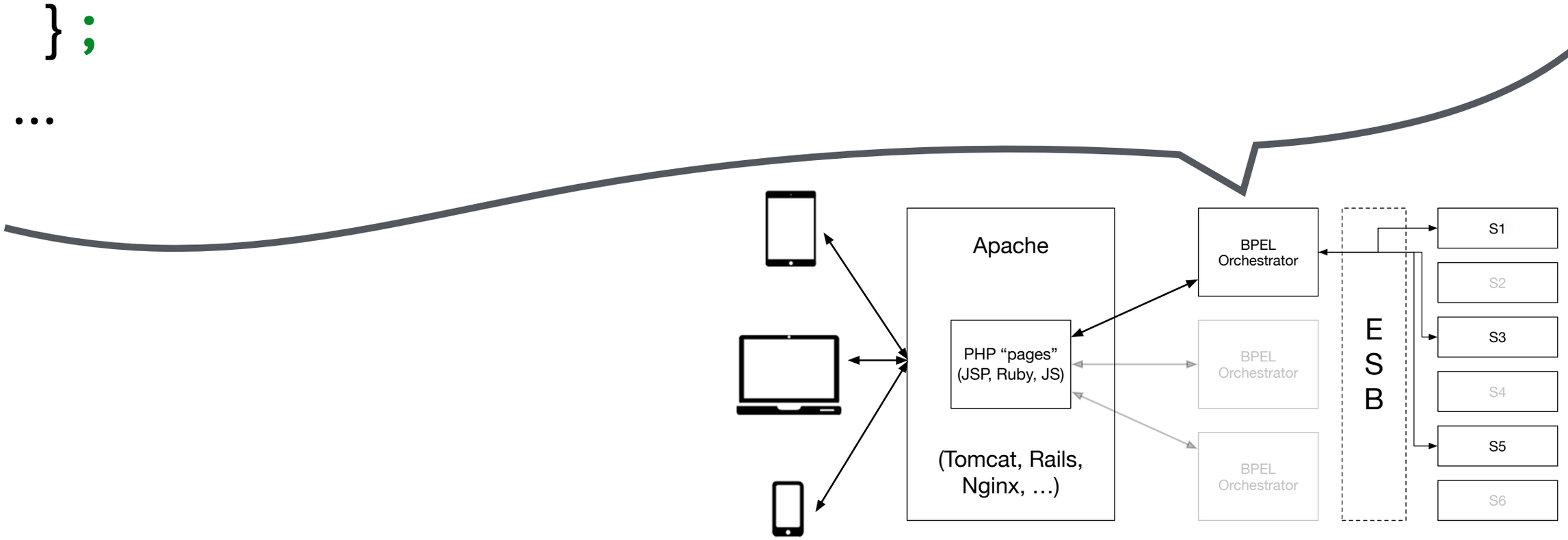
```
startSession( req )( sid ) { login@S1( req )( sid ) };
 {
    userApproval( u )( ru ){   userApproval@S3( u )( ru )   }
    |
    adminApproval( a )( ra ){   adminApproval@S5( a )( ra )   }
 };
...
```



SOC already snuck into web programming

```
startSession( req )( sid ) { login@S1( req )( sid ) } ;
 {
    userApproval( u )( ru ){   userApproval@S3( u )( ru )   }
    |
    adminApproval( a )( ra ){   adminApproval@S5( a )( ra )   }
 };
...
```

Sequence

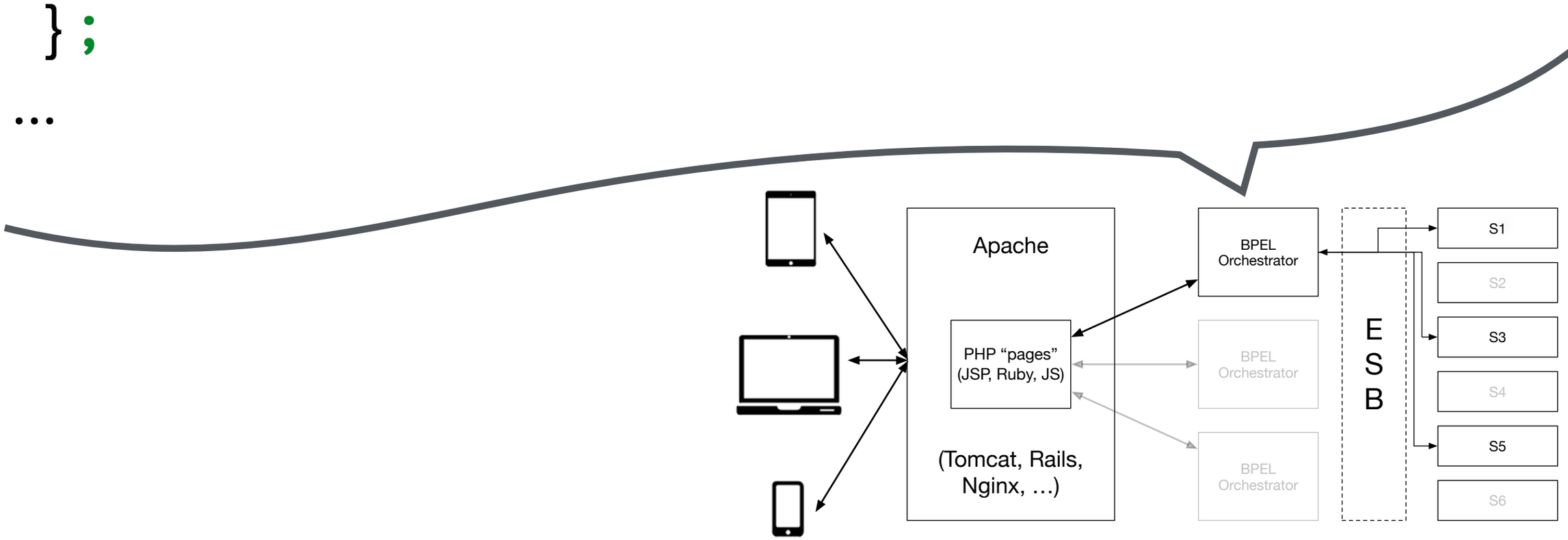


SOC already snuck into web programming

```
startSession( req )( sid ) { login@S1( req )( sid ) } ;
 {
    userApproval( u )( ru ){   userApproval@S3( u )( ru )   }
    |
    adminApproval( a )( ra ){   adminApproval@S5( a )( ra )   }
 };
...
```

Sequence →

| ← Parallel



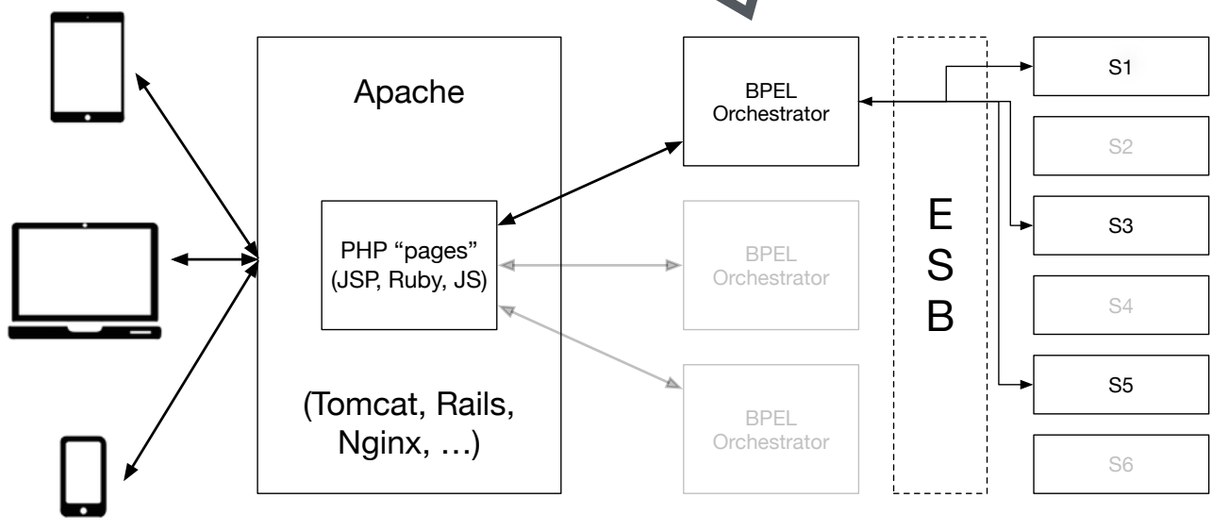
SOC already snuck into web programming

```
startSession( req )( sid ) { login@S1( req )( sid ) } ;
 {
    userApproval( u )( ru ){   userApproval@S3( u )( ru )   }
    |
    adminApproval( a )( ra ){   adminApproval@S5( a )( ra )   }
 } ;
...
```

Sequence

Parallel

Same instance (session) subsequent requests

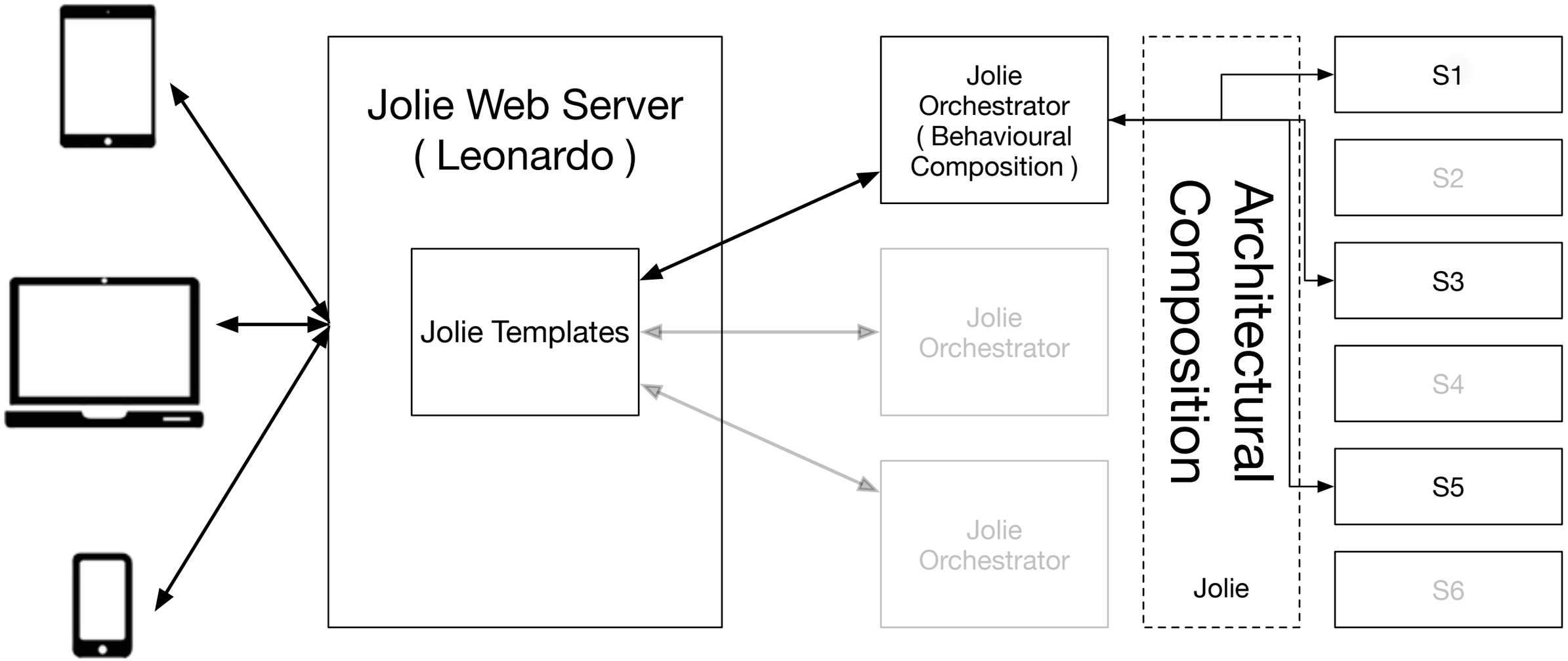


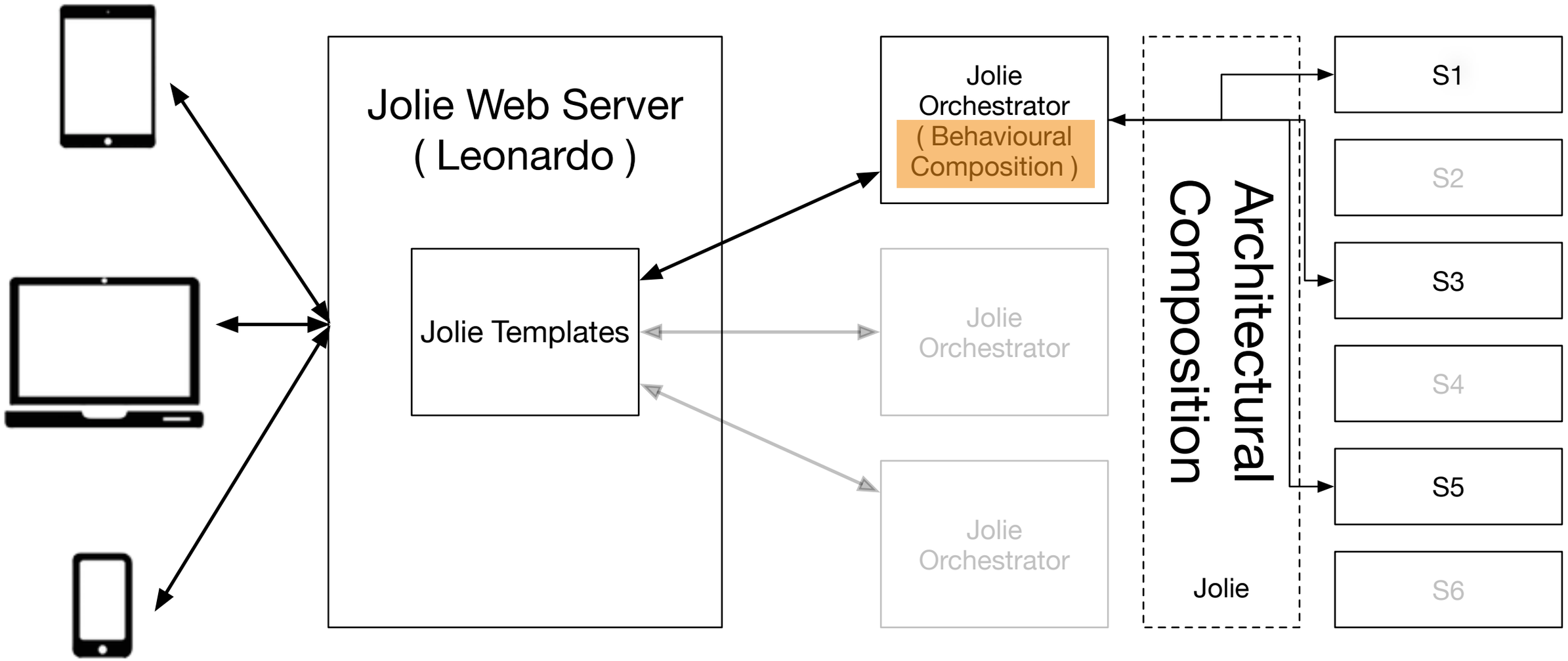
SOC already snuck into web programming

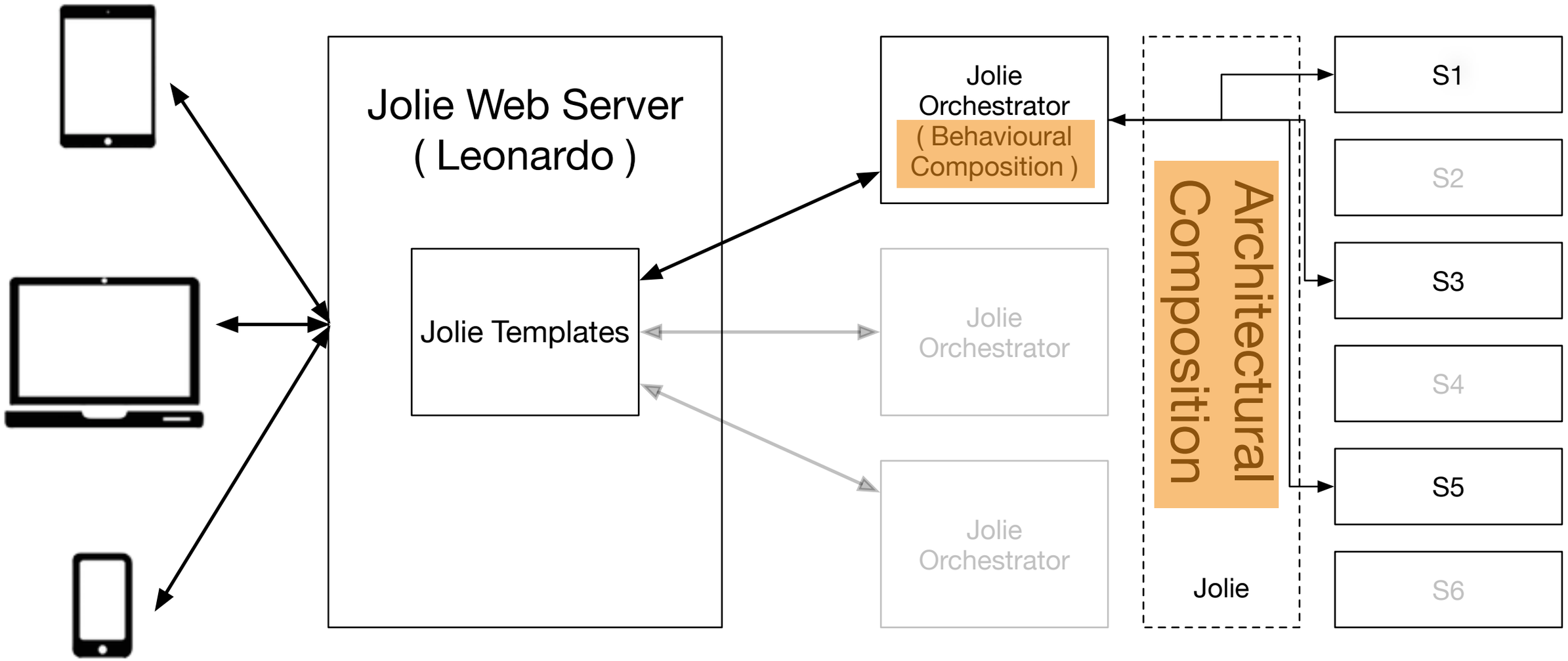
However:

- many “moving parts”;
- heterogeneous (specific know-how);
- hard to maintain;
- prone to breakage with modifications (upgrades, patches, etc).

Can Jolie aid web programming?


Jolie Web Server
(Leonardo)
Jolie Templates
Jolie
Orchestrator
(Behavioural
Composition)
Jolie
Orchestrator
Jolie
Orchestrator
Architectural
Composition
Jolie
S1
S2
S3
S4
S5
S6


Jolie Web Server
(Leonardo)
Jolie Templates
Jolie
Orchestrator
(Behavioural
Composition)
Jolie
Orchestrator
Jolie
Orchestrator
Architectural
Composition
Jolie
S1
S2
S3
S4
S5
S6


Jolie Web Server
(Leonardo)
Jolie Templates
Jolie
Orchestrator
(Behavioural
Composition)
Jolie
Orchestrator
Jolie
Orchestrator
Architectural
Composition
Jolie
S1
S2
S3
S4
S5
S6

Behavioural Composition

B	::=	η	*(input)*
	|	$\overline{\eta}$	*(output)*
	|	`[` η_1 `]` `{` B_1 `}` ... `[` η_n `]` `{` B_n `}`	*(input choice)*
	|	**`if`**`(` e `)` B_1 **`else`** B_2	*(cond)*
	|	**`while`**`(` e `)` B	*(while)*
	|	B **`;`** B'	*(seq)*
	|	B **`|`** B'	*(par)*
	|	**`throw`**`(f)`	*(throw)*
	|	`x =` e	*(assign)*
	|	`x -> y`	*(alias)*
	|	**`nullProcess`**	*(inact)*

η	::=	`o(x)`	*(one-way)*
	|	`o(x)` `(`e`)` `{` B `}`	*(request-response)*
$\overline{\eta}$	::=	`o@OP(`e`)`	*(notification)*
	|	`o@OP(`e`)(y)`	*(solicit-response)*

Architectural Composition (Deployment)

IP ::= **inputPort** $Port$ OP ::= **outputPort** $Port$

$Port$ ::= id {
Location: Loc
Protocol: $Proto$
Interfaces: $\text{iface}_1, \ldots, \text{iface}_n$
}

+ **Aggregation**
+ **Redirection**

Jolie HTTP Protocol

Jolie can support web applications with the **http** protocol.

Jolie HTTP Protocol

```
interface SumIface { RequestResponse: sum(SumT)(int) }

inputPort SumInput {
Location: "socket://localhost:8000"
Protocol: soap
Interfaces: SumIface
}

main
{
   sum( req )( resp ) {
      resp = req.x + req.y
   }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

```
interface SumIface { RequestResponse: sum(SumT)(int) }

inputPort SumInput {
Location: "socket://localhost:8000"
Protocol: soap
Interfaces: SumIface
}
```

just change “soap” to “http”

```
main
{
   sum( req )( resp ) {
      resp = req.x + req.y
   }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

```
interface SumIface { RequestResponse: sum(SumT)(int) }

inputPort SumInput {
Location: "socket://localhost:8000"
Protocol: soap
Interfaces: SumIface
}

main
{
   sum( req )( resp ) {
      resp = req.x + req.y
   }
}
```

just change “soap” to “http”

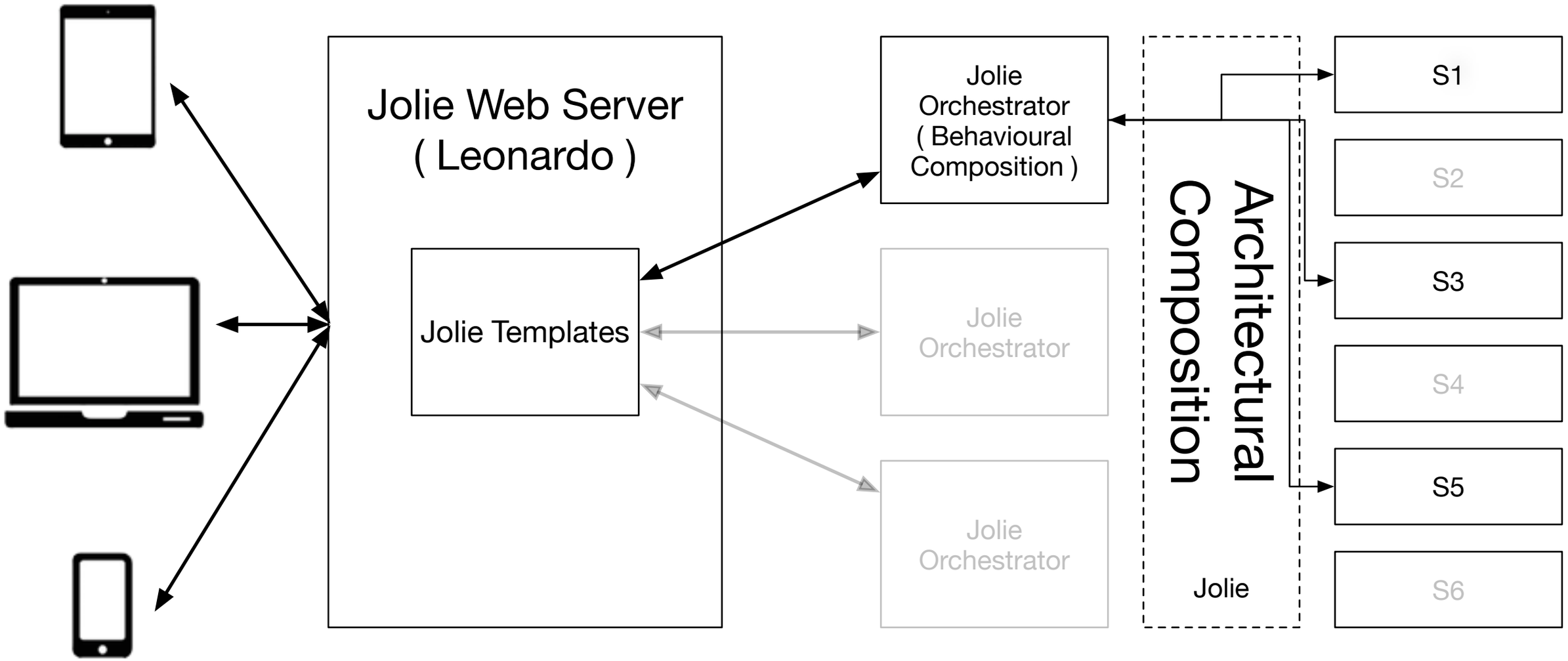
types are automatically casted (otherwise TypeMismatch with 403)

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

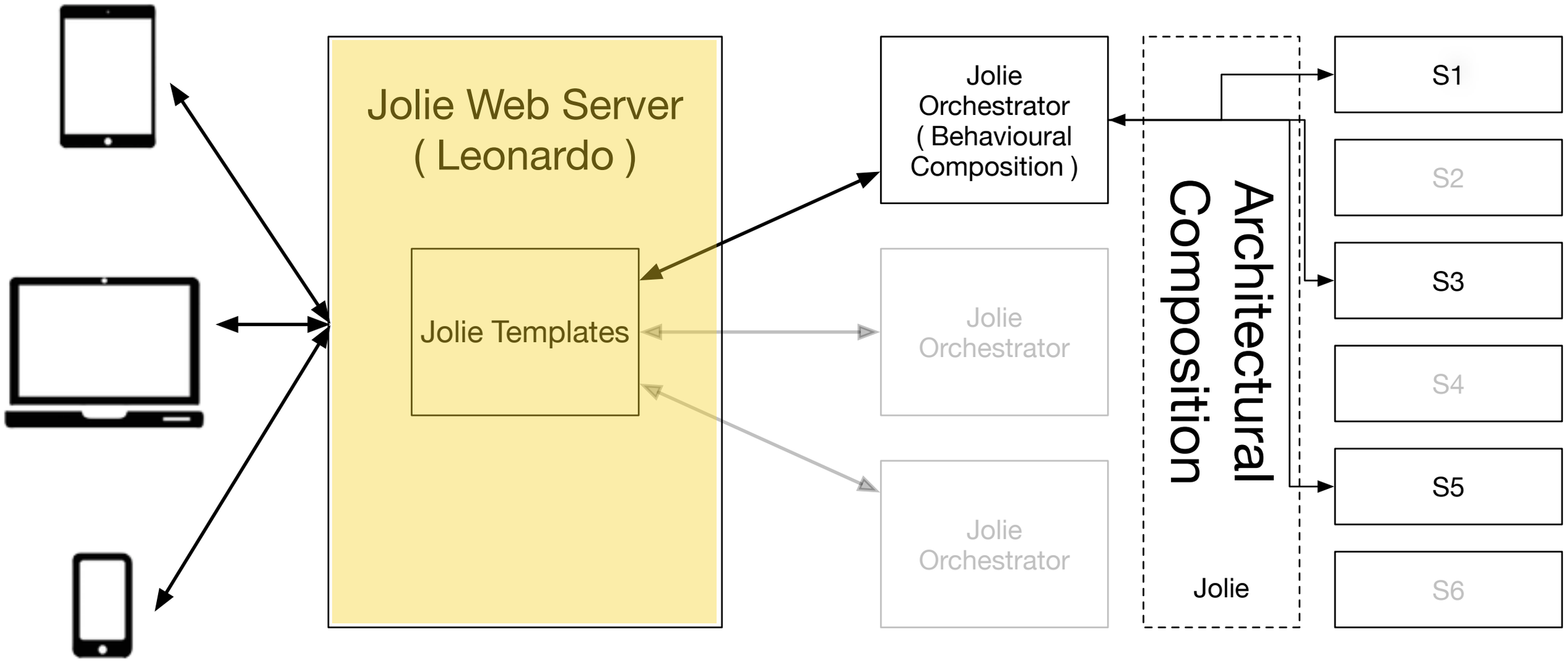
Jolie HTTP Protocol

http://localhost:8000/sum?x=2&y=3

Jolie HTTP Protocol



Jolie HTTP Protocol



Jolie HTTP Protocol

```
type FetchBib:void { .dblpKey:string }

interface DBLPIface {
RequestResponse: fetchBib( FetchBib )( string )
}

outputPort DBLP {
Location: "socket://dblp.uni-trier.de:80/"
Protocol: http {
  .osc.fetchBib.alias = "rec/bib2/%!{dblpKey}.bib";
  .format = "html" }
Interfaces: DBLPIface
}

main
{
  r.dblpKey = args[0];
  fetchBib@DBLP( r )( bibtex );
  println@Console( bibtex )()
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

```
type FetchBib:void { .dblpKey:string }

interface DBLPIface {
RequestResponse: fetchBib( FetchBib )( string )
}

outputPort DBLP {
Location: "socket://dblp.uni-trier.de:80/"
Protocol: http {
  .osc.fetchBib.alias = "rec/bib2/%!{dblpKey}.bib";
  .format = "html" }
Interfaces: DBLPIface
}

main
{
  r.dblpKey = args[0];
  fetchBib@DBLP( r )( bibtex );
  println@Console( bibtex )()
}
```

Fine-grained configuration parameters

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

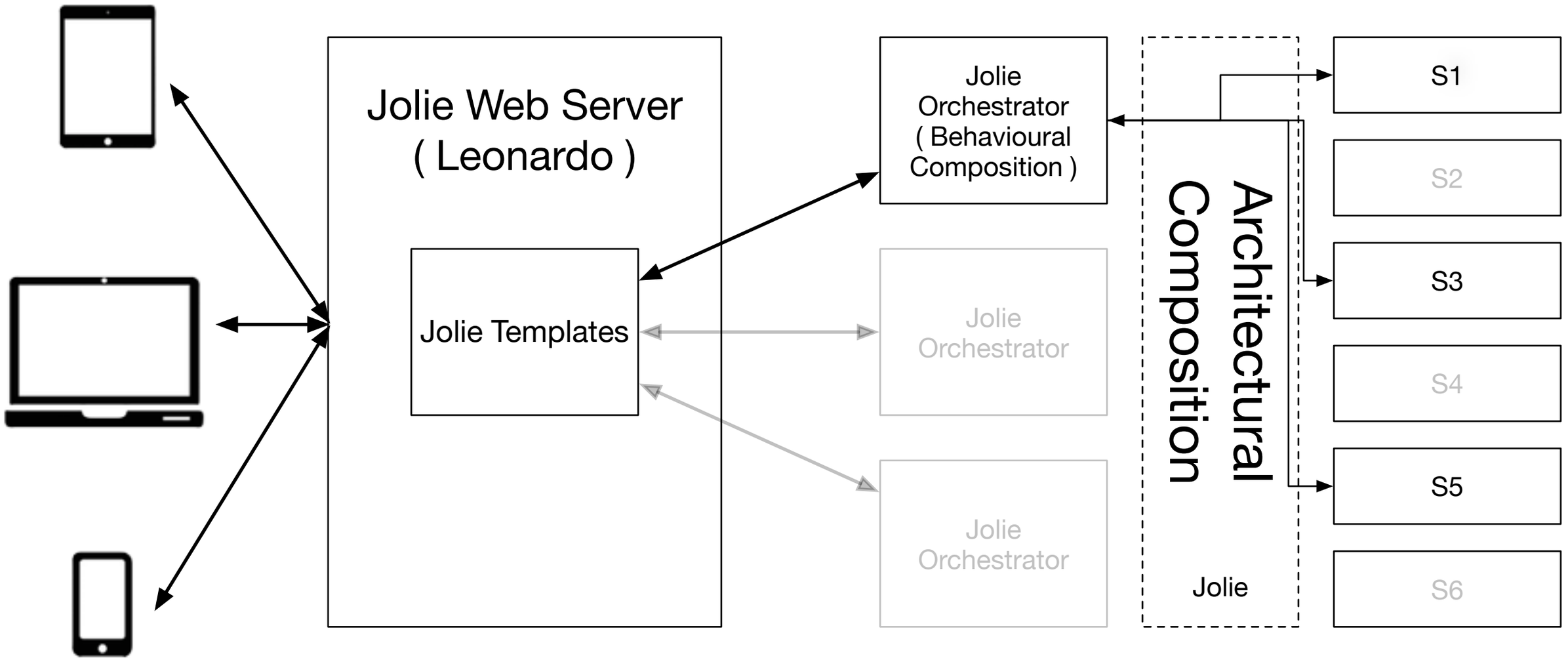
```
type FetchBib:void { .dblpKey:string }

interface DBLPIface {
RequestResponse: fetchBib( FetchBib )( string )
}

outputPort DBLP {
Location: "socket://dblp.uni-trier.de:80/"
Protocol: http {
   .osc.fetchBib.alias = "rec/bib2/%!{dblpKey}.bib";
   .format = "html" }
Interfaces: DBLPIface
}

main
{
   r.dblpKey = args[0];
   fetchBib@DBLP( r )( bibtex );
   println@Console( bibtex )()
}
```

Fine-grained configuration parameters

Major Novelty:
Operation-Specific Configurations (osc) bridge the gap between RESTful and Service-Oriented Architectures

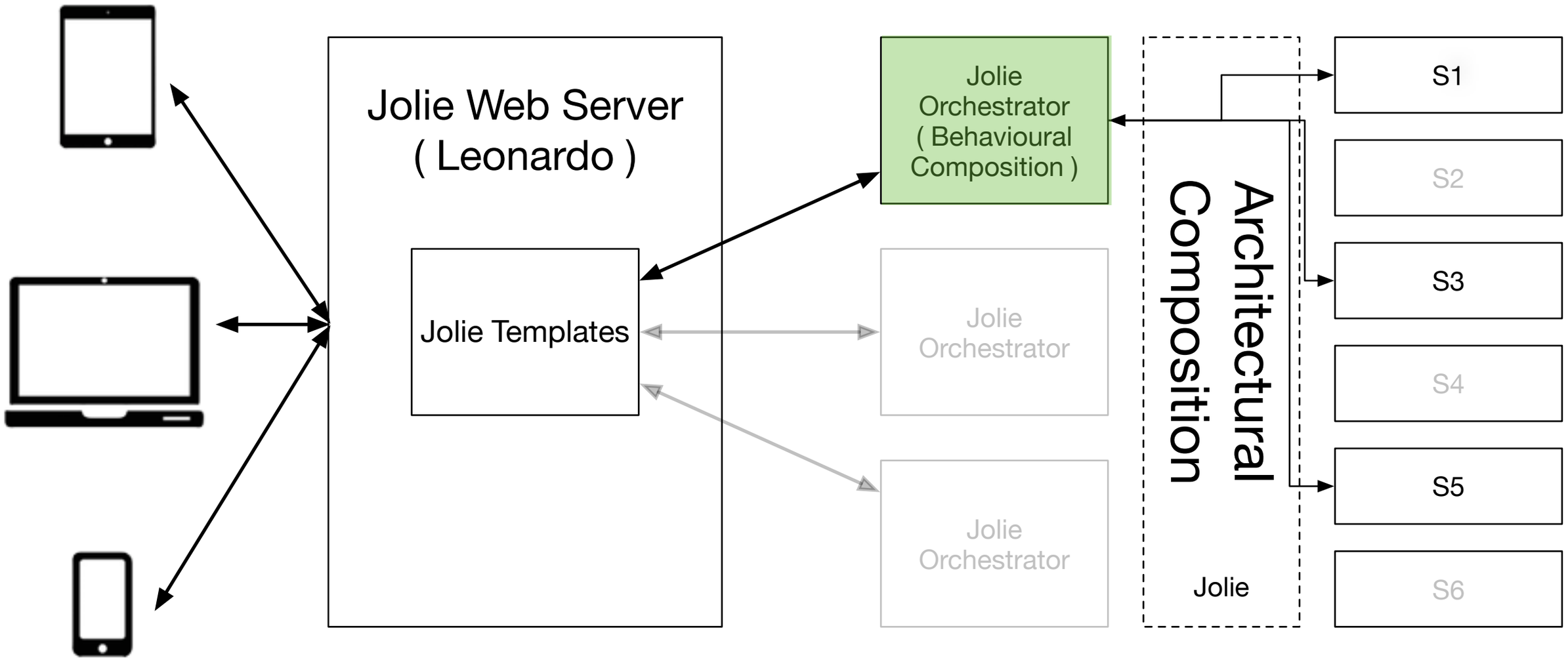
Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol



Jolie HTTP Protocol

Same instance (session) subsequent requests



Jolie HTTP Protocol

```
inputPort RISInput {
/* ... */
Protocol: http { .cookies.userKeyCookie = "userKey" }
}

outputPort Logger { /* ... */ }
outputPort Moderator { /* ... */ }

cset { userKey: addPub.userKey }
cset { modKey: approve.modKey reject.modKey }

define checkCredentials { /* ... */ }
define updateDB { /* ... */ }
```

```
main
{
  login( cred )( r ) {
    checkCredentials;
    r.userKey = csets.userKey = new
  };
  addPub( pub );
  noti.bibtex = pub.bibtex;
  noti.modKey = csets.modKey = new;
  { log@Logger( pub.bibtex )
    | notify@Moderator( noti ) };

  [ approve() ] {
    log@Logger( "Accepted " + pub.bibtex );
    updateDB
  }
  [ reject() ] {
    log@Logger( "Rejected " + pub.bibtex )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

```
inputPort RISInput {
/* ... */
Protocol: http { .cookies.userKeyCookie = "userKey" }
}

outputPort Logger { /* ... */ }
outputPort Moderator { /* ... */ }

cset { userKey: addPub.userKey }
cset { modKey: approve.modKey reject.modKey }

define checkCredentials { /* ... */ }
define updateDB { /* ... */ }
```

integration with cookies

```
main
{
  login( cred )( r ) {
    checkCredentials;
    r.userKey = csets.userKey = new
  };
  addPub( pub );
  noti.bibtex = pub.bibtex;
  noti.modKey = csets.modKey = new;
  { log@Logger( pub.bibtex )
    | notify@Moderator( noti ) };

  [ approve() ] {
    log@Logger( "Accepted " + pub.bibtex );
    updateDB
  }
  [ reject() ] {
    log@Logger( "Rejected " + pub.bibtex )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

```
inputPort RISInput {
/* ... */
Protocol: http { .cookies.userKeyCookie = "userKey" }
}

outputPort Logger { /* ... */ }
outputPort Moderator { /* ... */ }

cset { userKey: addPub.userKey }
cset { modKey: approve.modKey reject.modKey }

define checkCredentials { /* ... */ }
define updateDB { /* ... */ }
```

integration with cookies

Multiparty Session

```
main
{
  login( cred )( r ) {
    checkCredentials;
    r.userKey = csets.userKey = new
  };
  addPub( pub );
  noti.bibtex = pub.bibtex;
  noti.modKey = csets.modKey = new;
  { log@Logger( pub.bibtex )
  | notify@Moderator( noti ) };

  [ approve() ] {
    log@Logger( "Accepted " + pub.bibtex );
    updateDB
  }
  [ reject() ] {
    log@Logger( "Rejected " + pub.bibtex )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

```
inputPort RISInput {
/*  ...  */
Protocol: http { .cookies.userKeyCookie = "userKey" }
}

outputPort Logger { /*  ...  */ }
outputPort Moderator { /*  ...  */ }

cset { userKey: addPub.userKey }
cset { modKey: approve.modKey reject.modKey }

define checkCredentials { /*  ...  */ }
define updateDB { /*  ...  */ }
```

integration with cookies

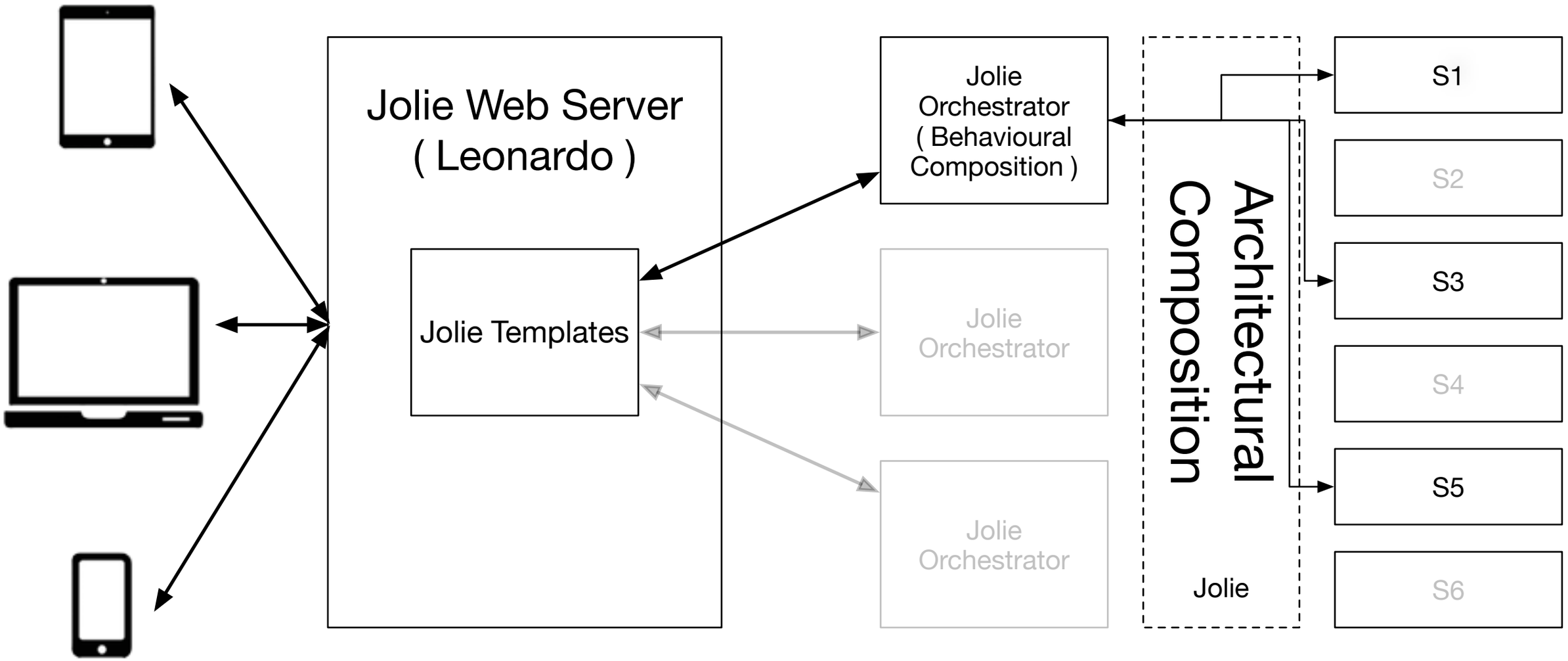
Multiparty Session

Correlation Sets defined on types (of operations).

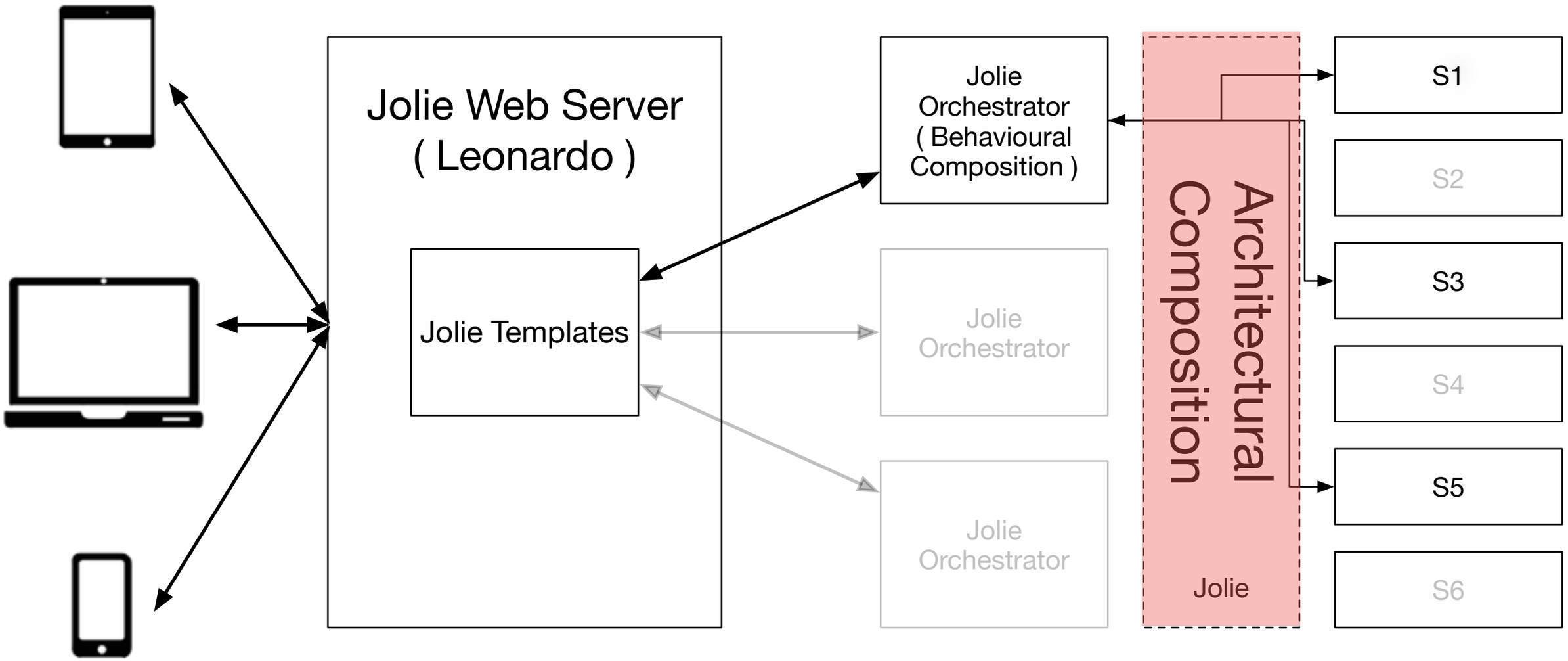
```
main
{
  login( cred )( r ) {
    checkCredentials;
    r.userKey = csets.userKey = new
  };
  addPub( pub );
  noti.bibtex = pub.bibtex;
  noti.modKey = csets.modKey = new;
  { log@Logger( pub.bibtex )
    | notify@Moderator( noti ) };

  [ approve() ] {
    log@Logger( "Accepted " + pub.bibtex );
    updateDB
  }
  [ reject() ] {
    log@Logger( "Rejected " + pub.bibtex )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

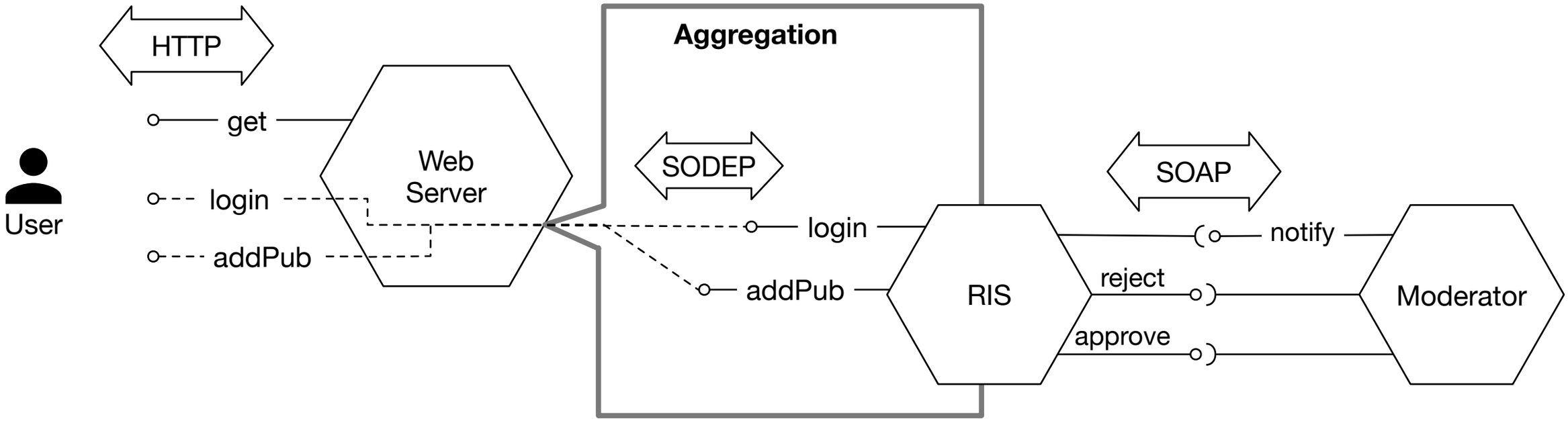
Jolie HTTP Protocol



Jolie HTTP Protocol



Jolie HTTP Protocol

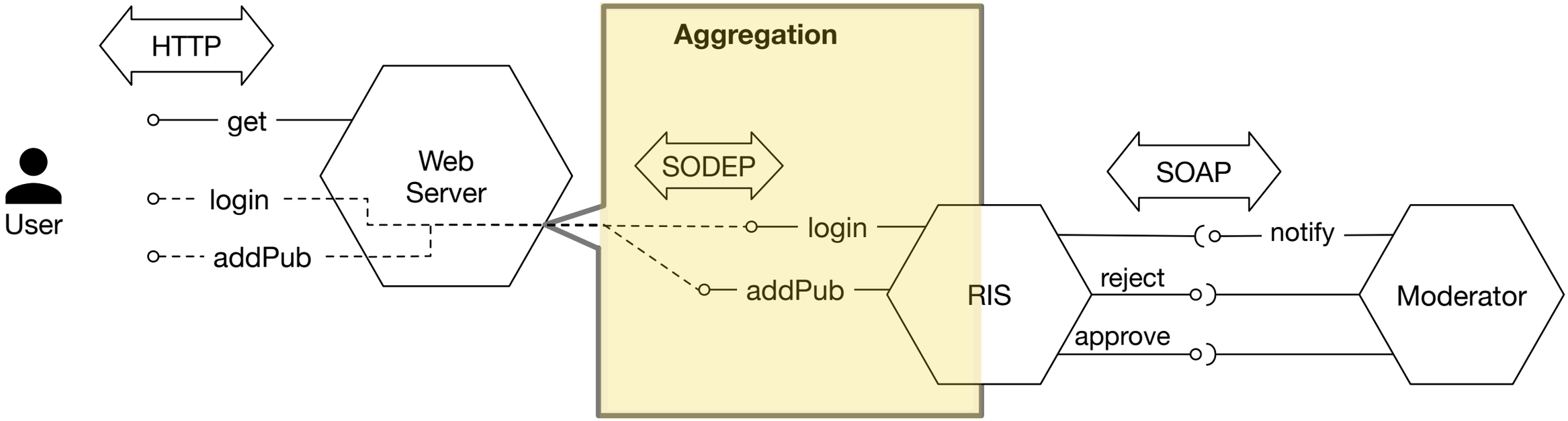



```
outputPort RIS {
Location: "socket://www.ris-example.com:8090/"
Protocol: sodep
Interfaces: RISIface
}

inputPort WebServerInput {
Location: "socket://www.webserver-example.com:80/"
Protocol: http {
  .default.get = "get";
  .cookies.userKeyCookie = "userKey"
  /* ... */
}
Interfaces: GetIface
Aggregates: RIS
}
```

```
main {
  get( req )( resp ) {
    /* ... */
    readFile@File( req.requestUri )( resp )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

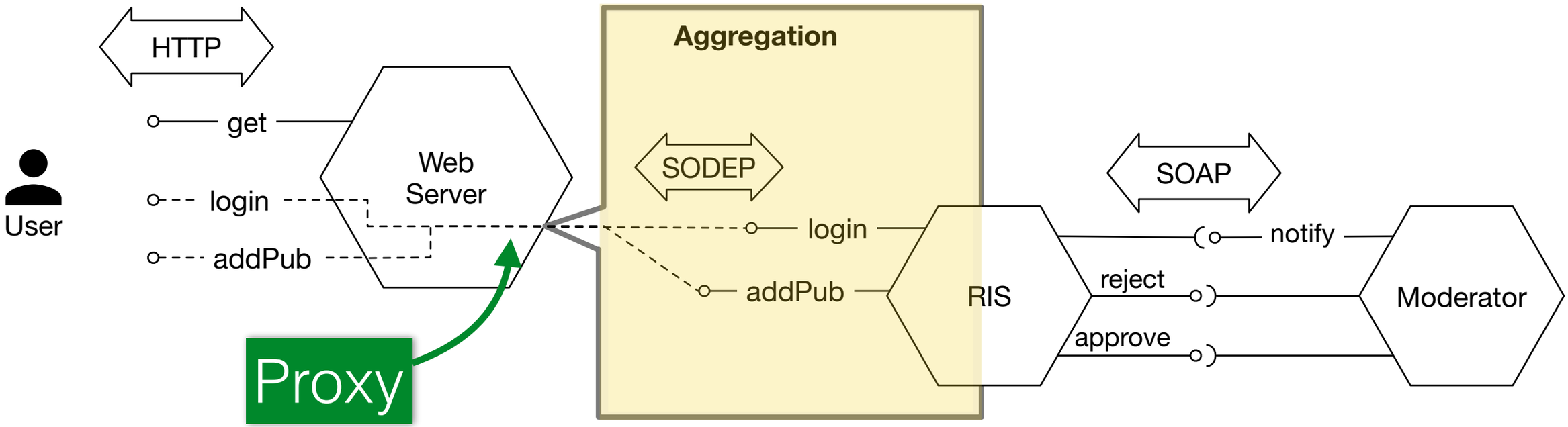



```
outputPort RIS {
Location: "socket://www.ris-example.com:8090/"
Protocol: sodep
Interfaces: RISIface
}

inputPort WebServerInput {
Location: "socket://www.webserver-example.com:80/"
Protocol: http {
  .default.get = "get";
  .cookies.userKeyCookie = "userKey"
  /* ... */
}
Interfaces: GetIface
Aggregates: RIS
}
```

```
main {
  get( req )( resp ) {
    /* ... */
    readFile@File( req.requestUri )( resp )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol

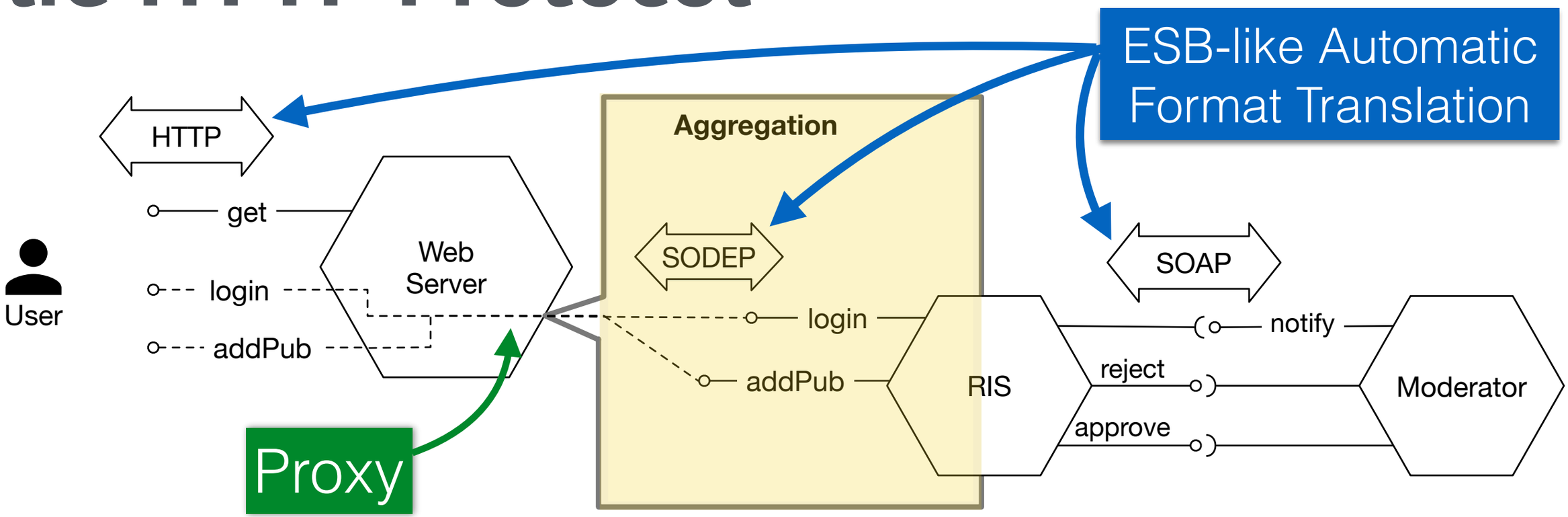



```
outputPort RIS {
Location: "socket://www.ris-example.com:8090/"
Protocol: sodep
Interfaces: RISIface
}

inputPort WebServerInput {
Location: "socket://www.webserver-example.com:80/"
Protocol: http {
  .default.get = "get";
  .cookies.userKeyCookie = "userKey"
  /* ... */
}
Interfaces: GetIface
Aggregates: RIS
}
```

```
main {
  get( req )( resp ) {
    /* ... */
    readFile@File( req.requestUri )( resp )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

Jolie HTTP Protocol




```
outputPort RIS {
Location: "socket://www.ris-example.com:8090/"
Protocol: sodep
Interfaces: RISIface
}

inputPort WebServerInput {
Location: "socket://www.webserver-example.com:80/"
Protocol: http {
  .default.get = "get";
  .cookies.userKeyCookie = "userKey"
  /* ... */
}
Interfaces: GetIface
Aggregates: RIS
}
```

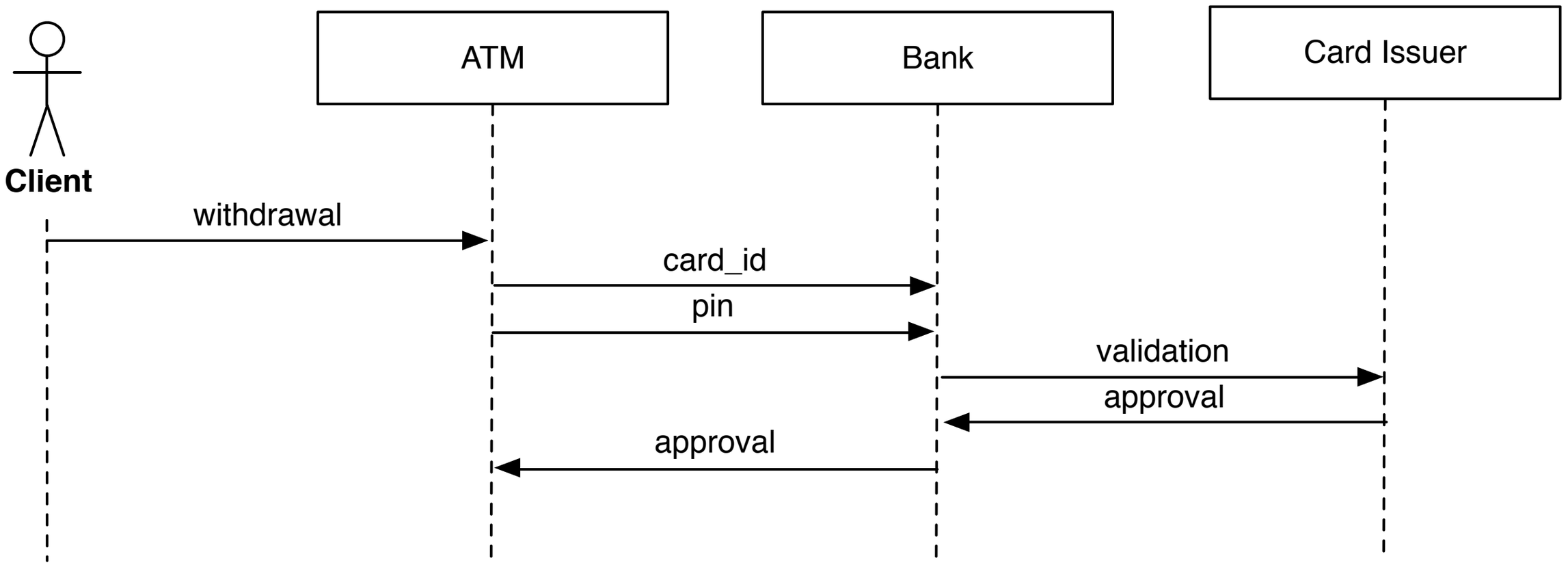
```
main {
  get( req )( resp ) {
    /* ... */
    readFile@File( req.requestUri )( resp )
  }
}
```

Example taken from Fabrizio Montesi. Process-aware Web Programming with Jolie. Submitted for publication, 2015

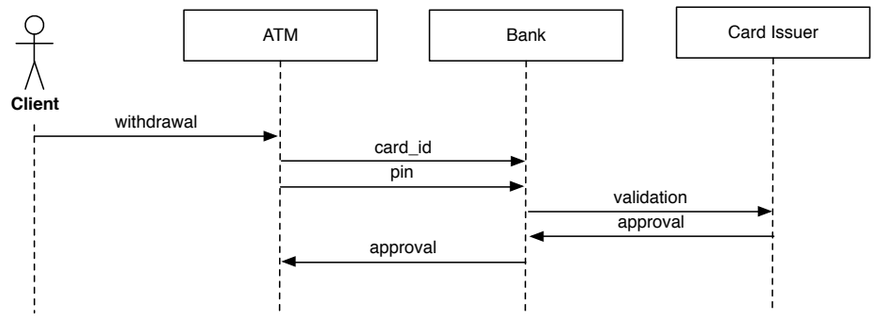
Introduction to

Choreographies

Introduction to **Choreographies**



Introduction to **Choreographies**



Introduction to **Choreographies**

$$\texttt{Client} \rightarrow \texttt{ATM}.\mathit{withdraw}(\left\{ \begin{array}{l} .\mathbf{card\_id} : \mathbf{String}, \\ .\mathbf{pin} : \mathbf{String} \end{array} \right\});$$
$$\texttt{ATM} \rightarrow \texttt{Bank}.\mathit{card\_id}(\mathbf{String});$$
$$\texttt{ATM} \rightarrow \texttt{Bank}.\mathit{pin}(\mathbf{String});$$
$$\texttt{Bank} \rightarrow \texttt{ATM}\{$$
$$\quad .\mathit{approve}(\mathbf{nil});$$
$$\quad .\mathit{reject}(\mathbf{nil});$$
$$\}$$

$$\texttt{Bank} \rightarrow \texttt{Card Issuer}.\mathit{validation}(\left\{ \begin{array}{l} .\mathbf{card} : \mathbf{String}, \\ .\mathbf{pswd} : \mathbf{String} \end{array} \right\});$$
$$\texttt{Card Issuer} \rightarrow \texttt{Bank}\{$$
$$\quad .\mathit{approve}(\mathbf{nil});$$
$$\quad .\mathit{reject}(\mathbf{nil});$$
$$\}$$

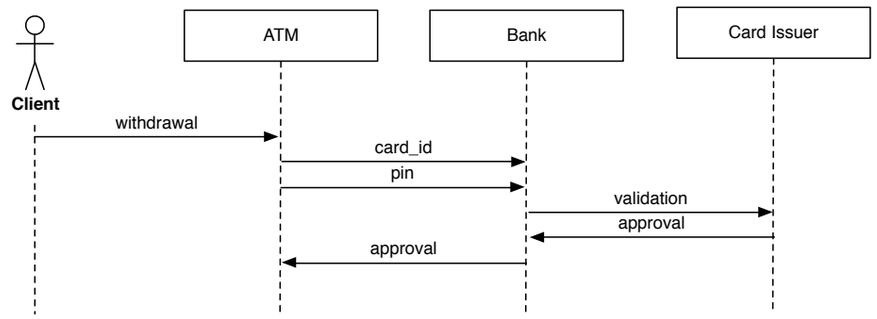
Global Types

Choreographies

$$\mathbf{req}\ k:\ \mathtt{c}[\texttt{Client}] \Leftrightarrow l_{\mathtt{A}}.\texttt{ATM}, l_{\mathtt{B}}.\texttt{Bank};$$
$$k:\ \mathtt{c}[\texttt{Client}].\underline{\mathbf{data}} \rightarrow \texttt{ATM}.\mathit{withdraw}(\underline{\mathbf{data}})$$

$$\mathbf{acc}\ k:\ l_{\mathtt{A}}.\mathtt{p}[\texttt{ATM}],\ l_{\mathtt{B}}.\mathtt{q}[\texttt{Bank}];$$
$$k:\ \texttt{Client} \rightarrow \mathtt{p}[\texttt{ATM}].\mathit{withdraw}(\underline{\mathbf{data}});$$
$$k:\ \mathtt{p}[\texttt{ATM}].\underline{\mathbf{data.card\_id}} \rightarrow \mathtt{q}[\texttt{Bank}].\mathit{card\_id}(\underline{\mathbf{validate.card}});$$
$$k:\ \mathtt{p}[\texttt{ATM}].\underline{\mathbf{data.pin}} \rightarrow \mathtt{q}[\texttt{Bank}].\mathit{pin}(\underline{\mathbf{validate.pswd}});$$
$$\mathtt{req}\ k':\mathtt{q}[\texttt{Bank}] \Leftrightarrow l_{\mathtt{C}}.\texttt{Card Issuer};$$
$$k':\ \mathtt{q}[\texttt{Bank}].\underline{\mathbf{validate}} \rightarrow \texttt{Card Issuer}.\mathit{validation};$$
$$k':\ \texttt{Card Issuer} \rightarrow \mathtt{q}[\texttt{Bank}]\{$$
$$\quad .\mathit{approve}();$$
$$\quad\quad k:\ \mathtt{q}[\texttt{Bank}] \rightarrow \mathtt{p}[\texttt{ATM}].\mathit{approve}();$$
$$\quad .\mathit{reject}();$$
$$\quad\quad k:\ \mathtt{q}[\texttt{Bank}] \rightarrow \mathtt{p}[\texttt{ATM}].\mathit{reject}();$$
$$\}$$

$$\mathbf{acc}\ k:\ l_{\mathtt{C}}.\mathtt{r}[\texttt{Card Issuer}];$$
$$k:\ \texttt{Bank} \rightarrow \texttt{Card Issuer}.\mathit{validation}(\underline{\mathbf{data}});$$
$$\mathtt{if}\ \mathtt{r}.\mathit{match}(\underline{\mathbf{data}})\{$$
$$\quad k:\ \mathtt{r}[\texttt{Card Issuer}] \rightarrow \texttt{Bank}.\mathit{approve}();$$
$$\}\ \mathtt{else}\ \{$$
$$\quad k:\ \mathtt{r}[\texttt{Card Issuer}] \rightarrow \texttt{Bank}.\mathit{reject}();$$
$$\}$$



Introduction to **Choreographies**

Main features:

- Global View;
- Modular, models asynchrony;
- Projects to correct processes;
- Deadlock- and Race-free by construction.

Choreographies for the Web?

Open for discussion!