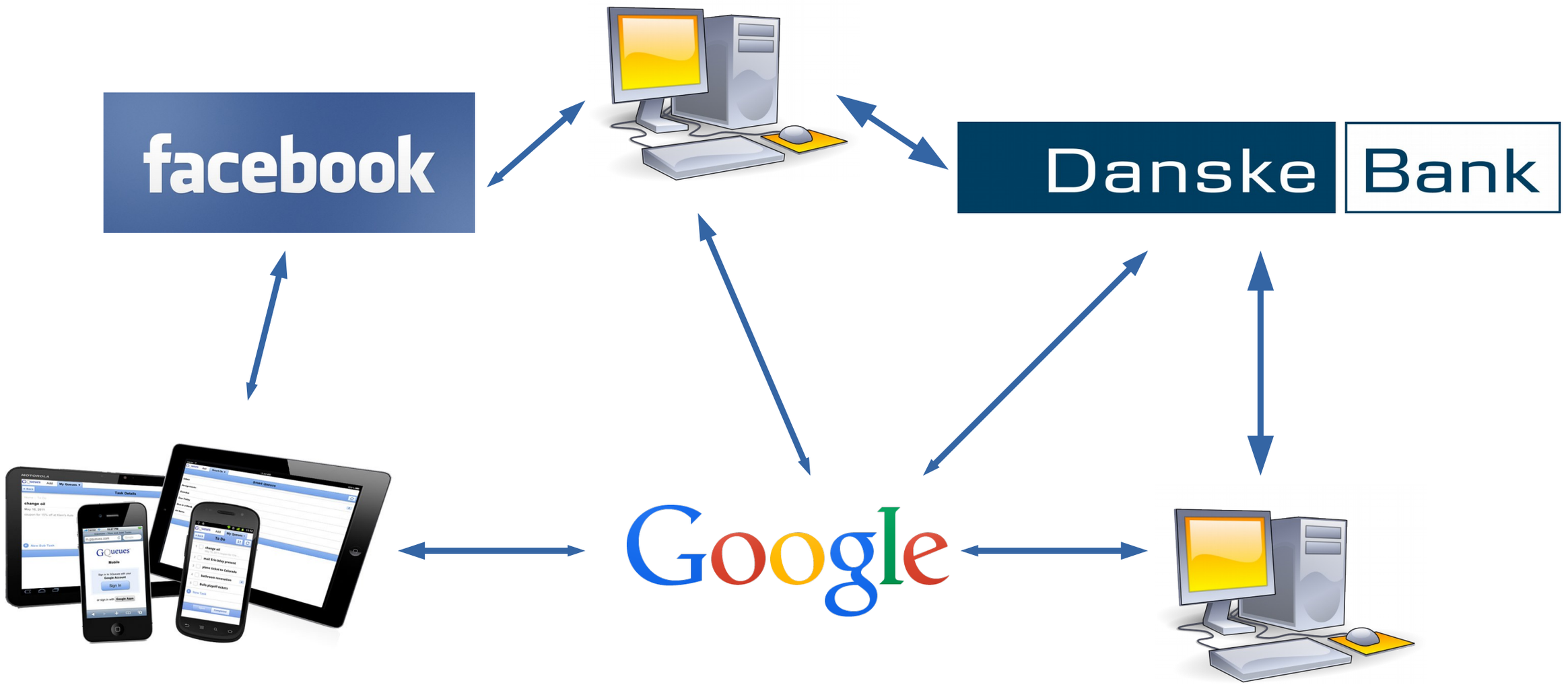
From

Service-Oriented Computing

to

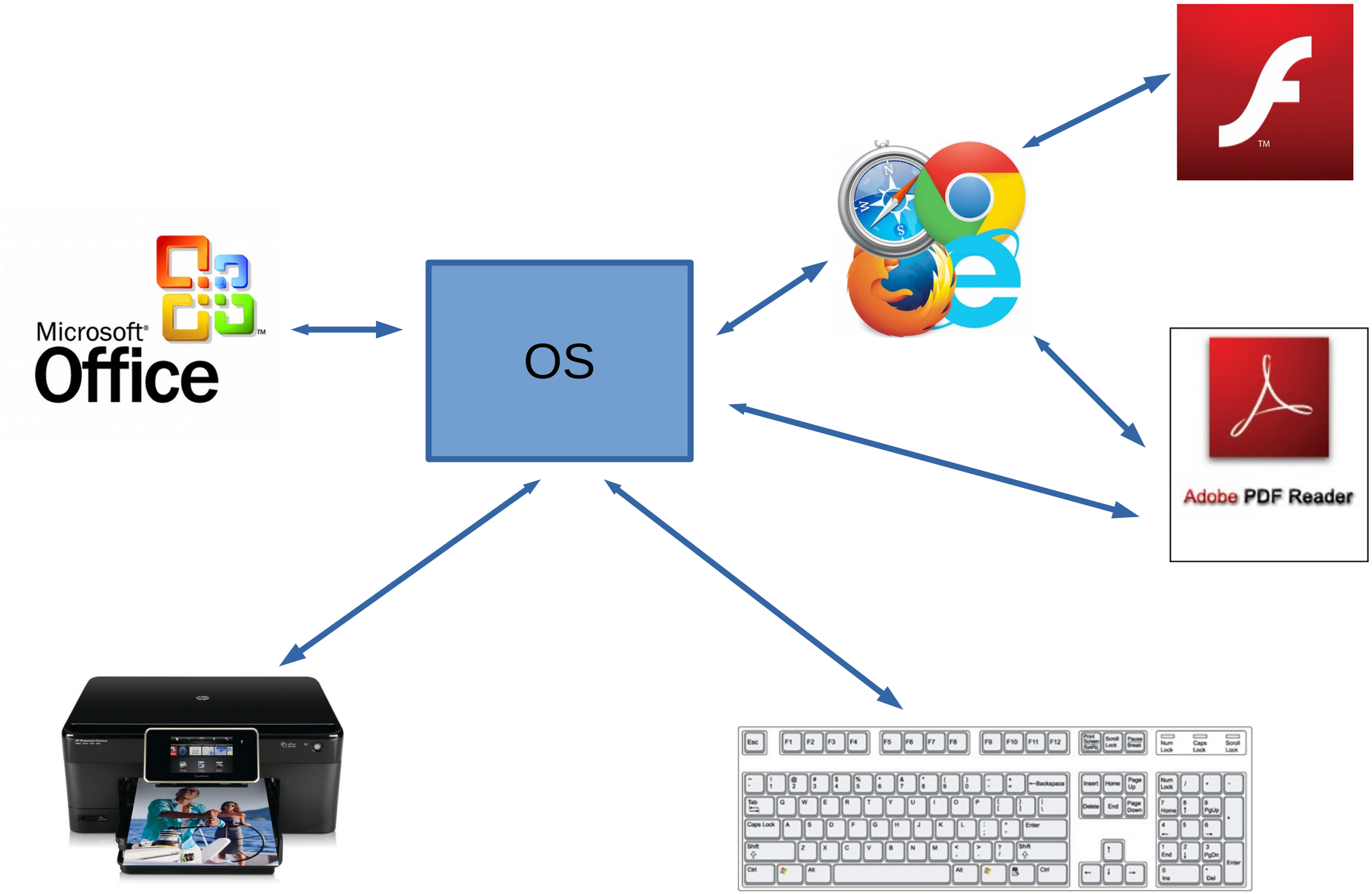
Microservices and Beyond

Distributed Systems



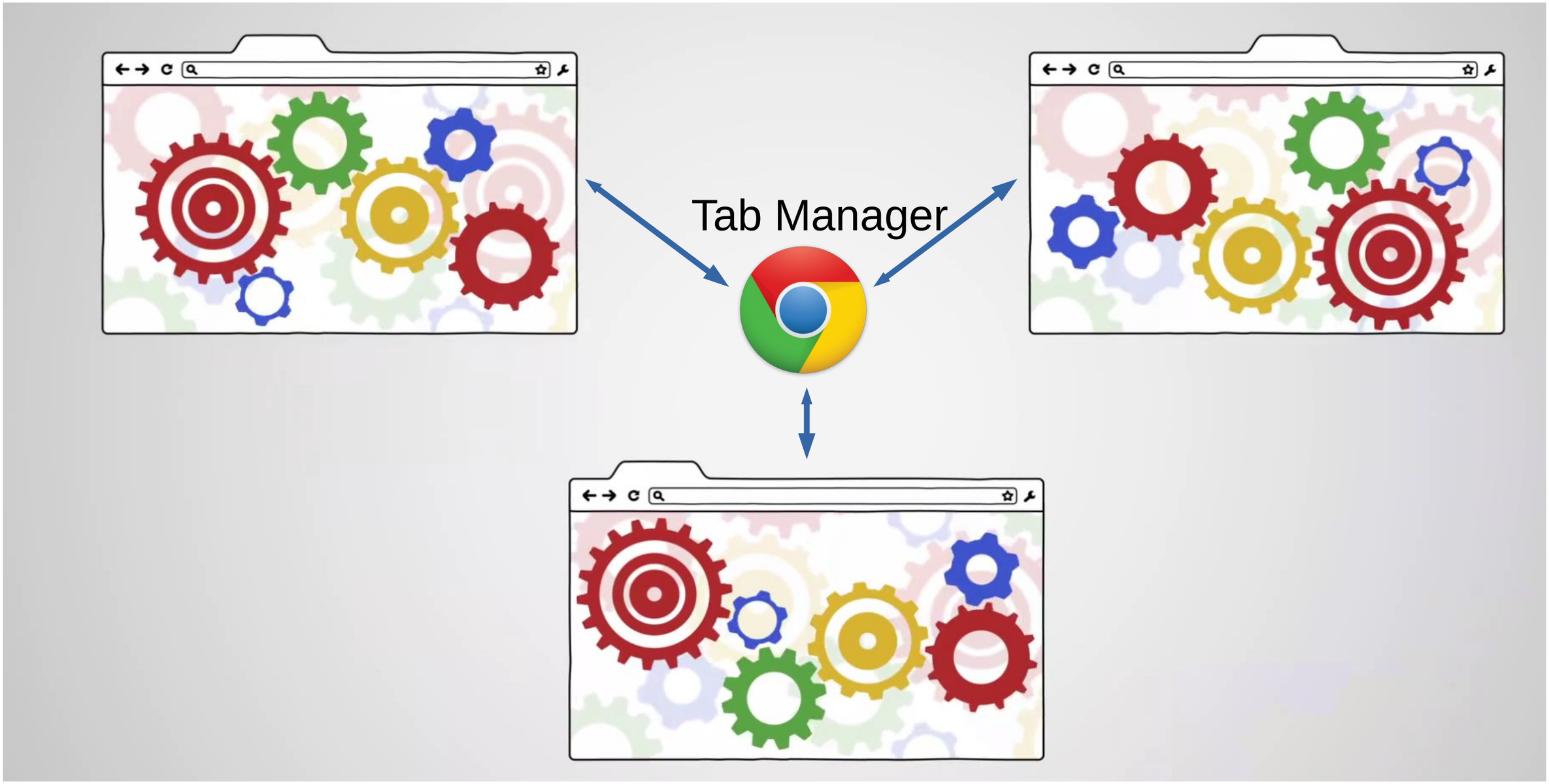
Courtesy of Fabrizio Montesi

Distributed Systems



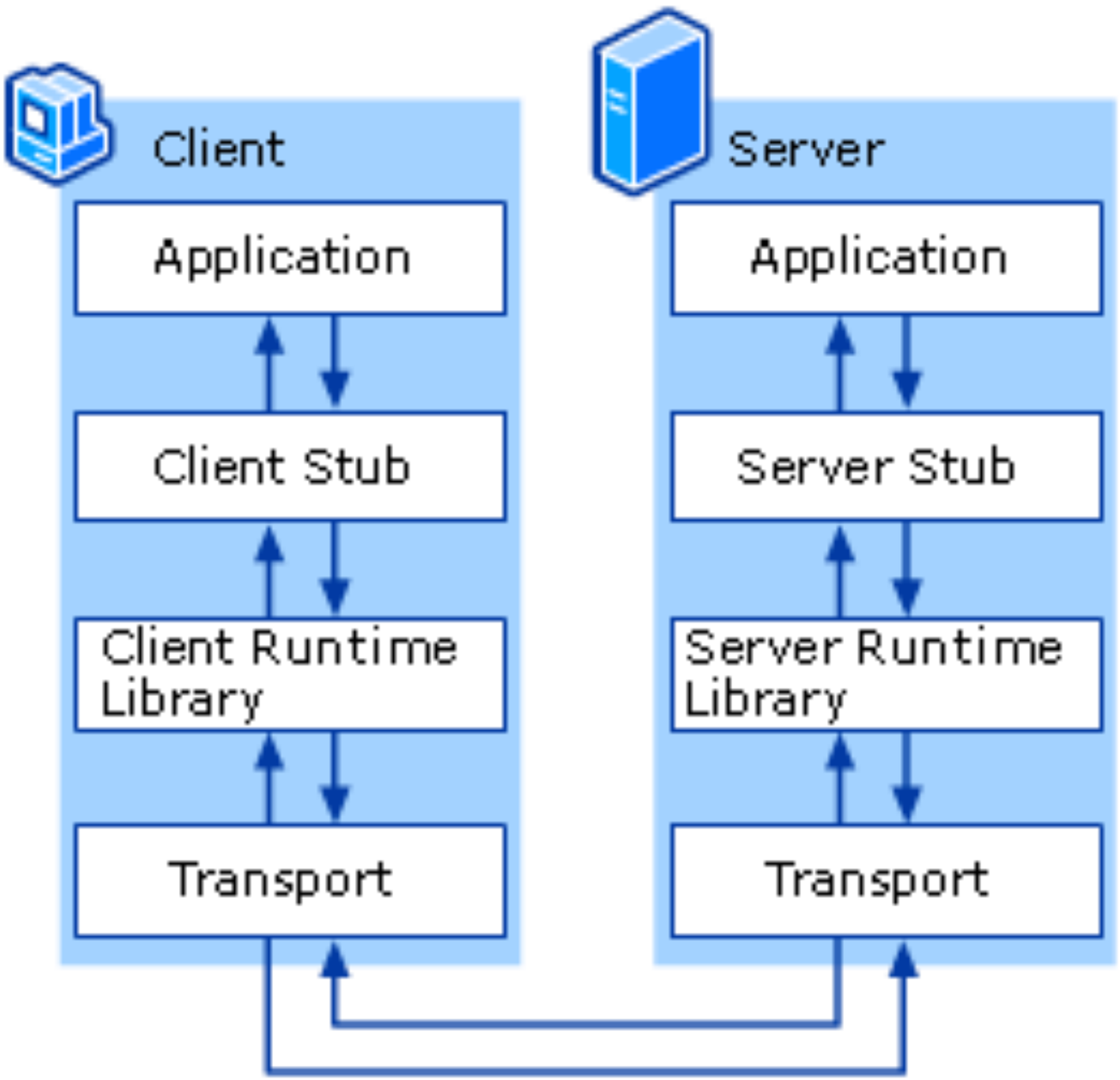
Courtesy of Fabrizio Montesi

Distributed Systems

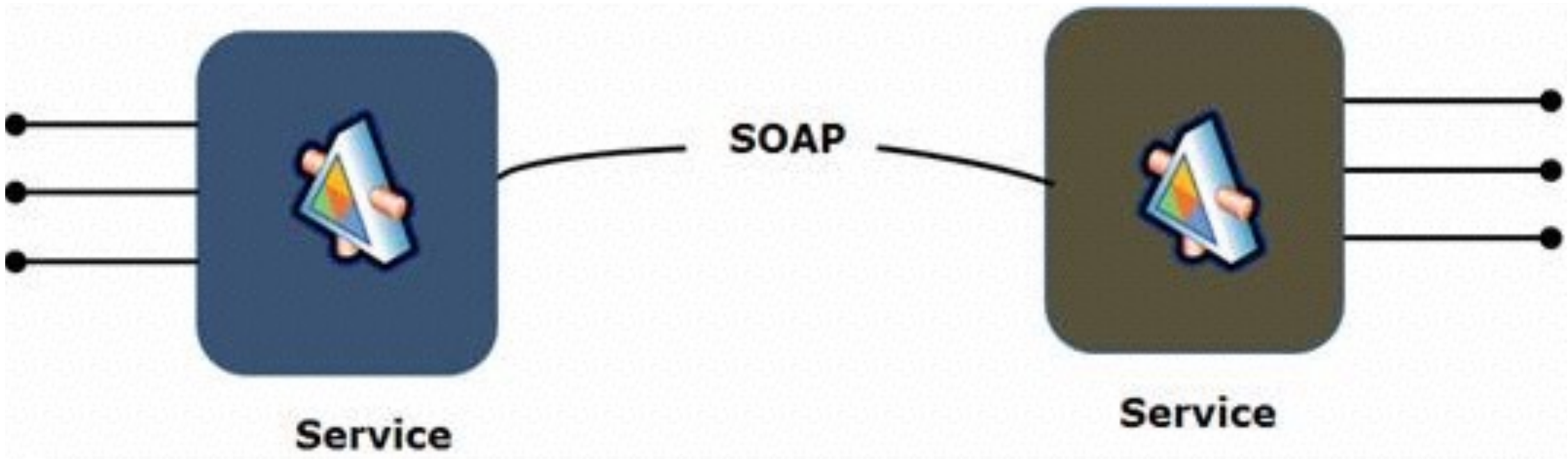


Courtesy of Fabrizio Montesi

Distributed Systems | **How to program them?**



Distributed Systems | **Service-Orientation**



From **remotely invoking methods** on objects
To **passing messages** between services

Distributed Systems | **Service-Orientation**

Interacted through well-defined message exchanges

Designed for both availability and stability

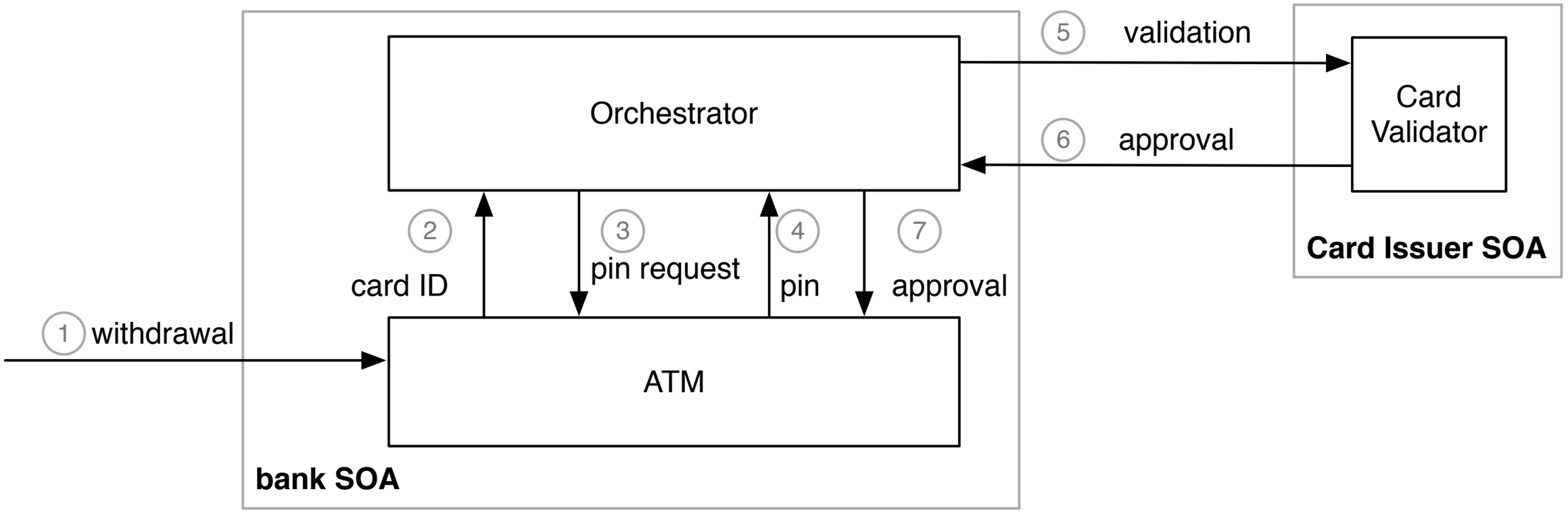
Limited knowledge of **how messages** are passed to or retrieved from it

Implements simple, granular functions

It is **service configurations** and **aggregations** that change (loosely-coupled infrastructure).

Distributed Systems | **Service Composition**

Orchestration



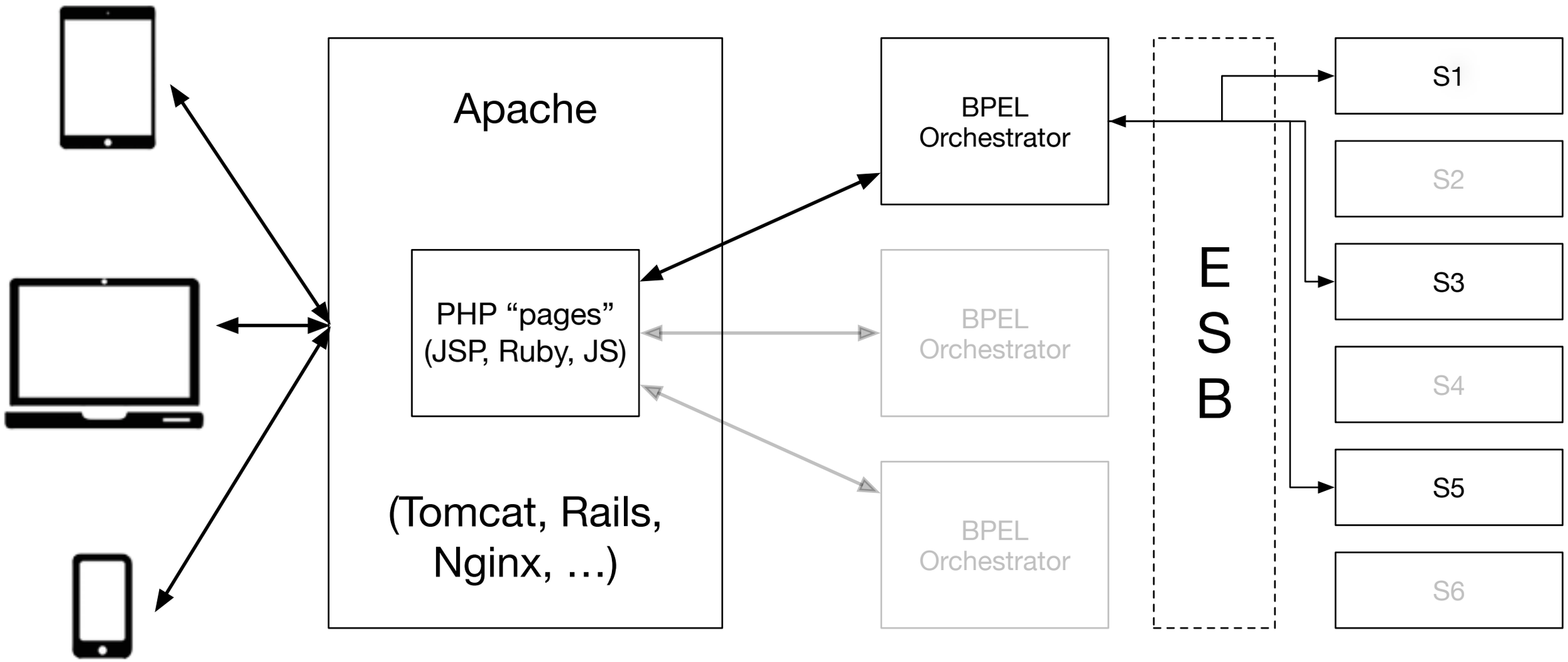
Distributed Systems | **Service Composition**

Orchestration | WS-BPEL

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Asynchronous BPEL process -->

<process name="BusinessTravelProcess"
         targetNamespace="http://packtpub.com/bpel/travel/"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:trv="http://packtpub.com/bpel/travel/"
         xmlns:emp="http://packtpub.com/service/employee/"
         xmlns:aln="http://packtpub.com/service/airline/" >

   <partnerLinks>
      <partnerLink name="client"
                   partnerLinkType="trv:travelLT"
                   myRole="travelService"
                   partnerRole="travelServiceCustomer"/>

      <partnerLink name="employeeTravelStatus"
                   partnerLinkType="emp:employeeLT"
                   partnerRole="employeeTravelStatusService"/>
```

Distributed Systems | **Service Composition**

Orchestration | WS-BPEL



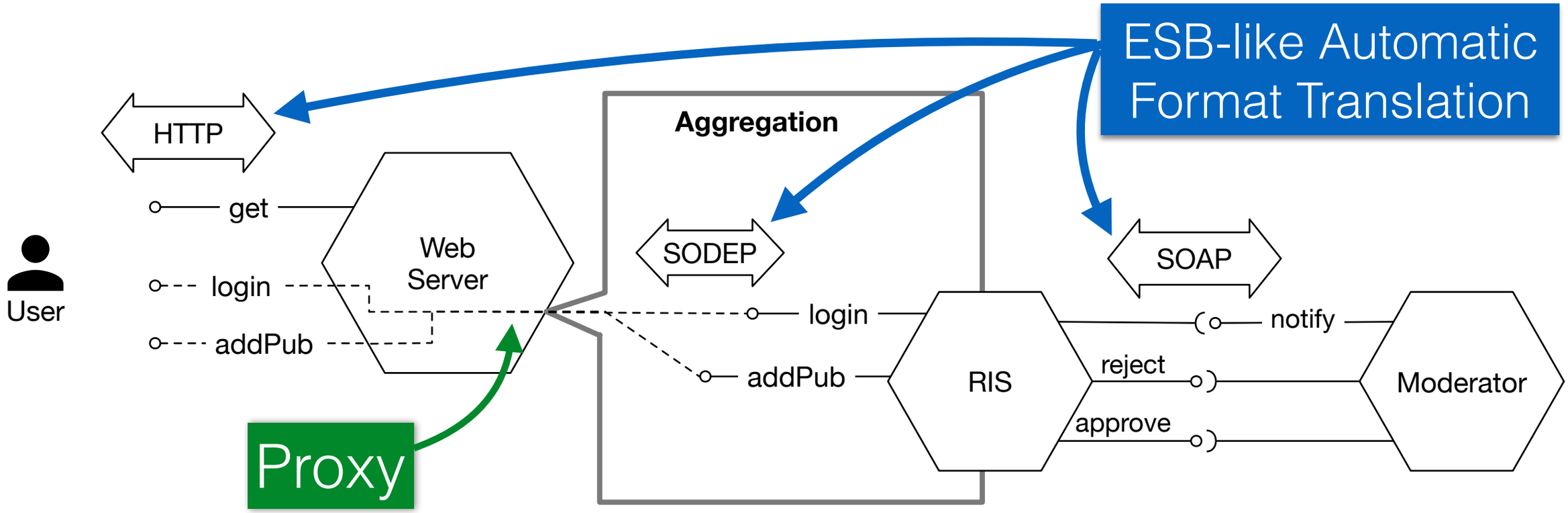
Distributed Systems | **Service Composition**

Orchestration | Jolie

```
requestID@ATM()( request.cardID );
requestPIN@ATM()( request.pinID );
validateID@CardValidator( request )( approval );
if ( approval ){
  requestOperation@ATM()( operation );
  ...
} else {
  ejectCard@ATM()
}
```

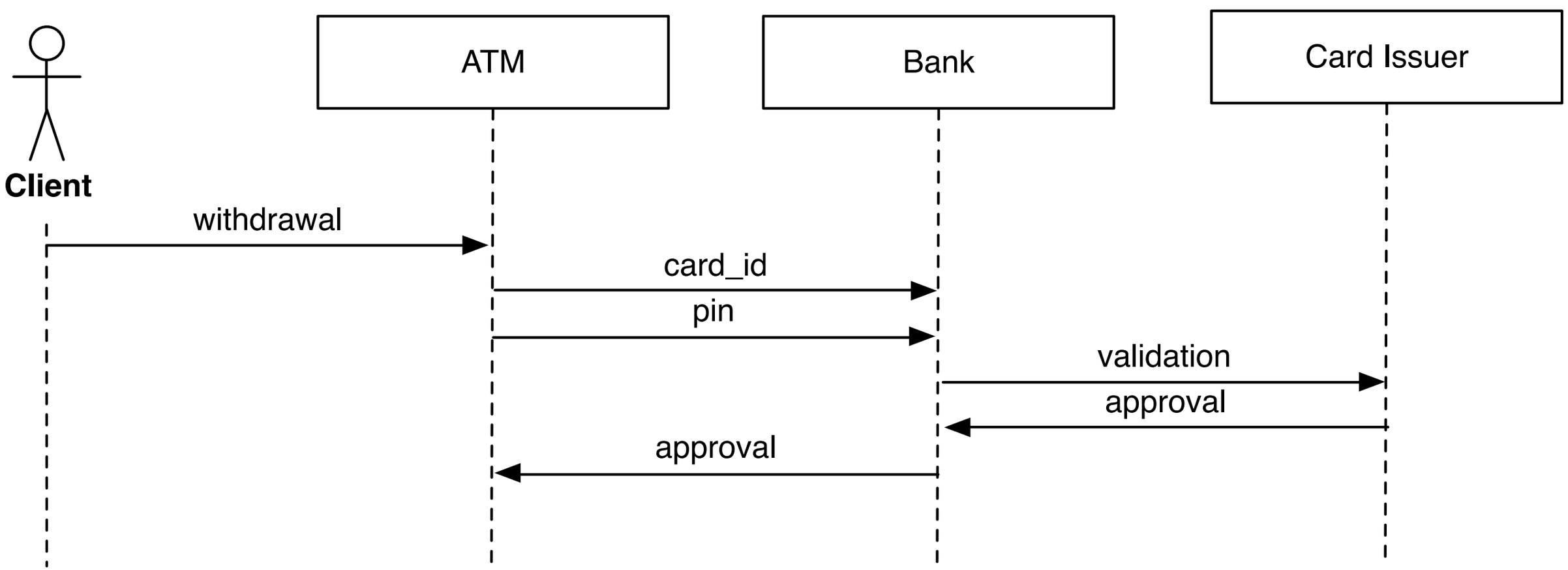
Distributed Systems | **Service Composition**

Orchestration | Jolie



Distributed Systems | **Service Composition**

Choreographies



Distributed Systems | **Service Composition**

Choreographies | WS-CDL

```
<choreography name="CreditAuthorization" root="false" coordination="true">
  <relationship type="tns:CreditReqCreditResp"/>
  <variableDefinitions>
    <variable name="CreditExtended" informationType="xsd:int" silent="true"
          roleTypes="tns:CreditResponder"/>
    <variable name="creditRequest"/>
    <variable name="creditAuthorized"/>
    <variable name="creditDenied" informationType = "tns:creditDeniedType"/>
</variableDefinitions>

<!-- the normal work - receive the request and decide whether to approve -->
<interaction name="creditAuthorization" channelVariable="tns:CreditRequestor"
          operation="authorize">
  <participate relationshipType="SuperiorInferior"
            fromRoleTypeRef="tns:Superior"
            toRoleTypeRef="tns:Inferior"/>
  <exchange name="creditRequest" informationType="creditRequest"
          action="request">
    <send variable="getVariable('tns:creditRequest','')"/>
    <receive variable="getVariable('tns:creditRequest','')"/>
  </exchange>
  <exchange name="creditAuthorized" informationType="creditAuthorizedType"
```

Distributed Systems | **Service Composition**

Choreographies | Chor/AIOCJ

Choreography —*EPP*→ Endpoint Projection

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

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

EPP

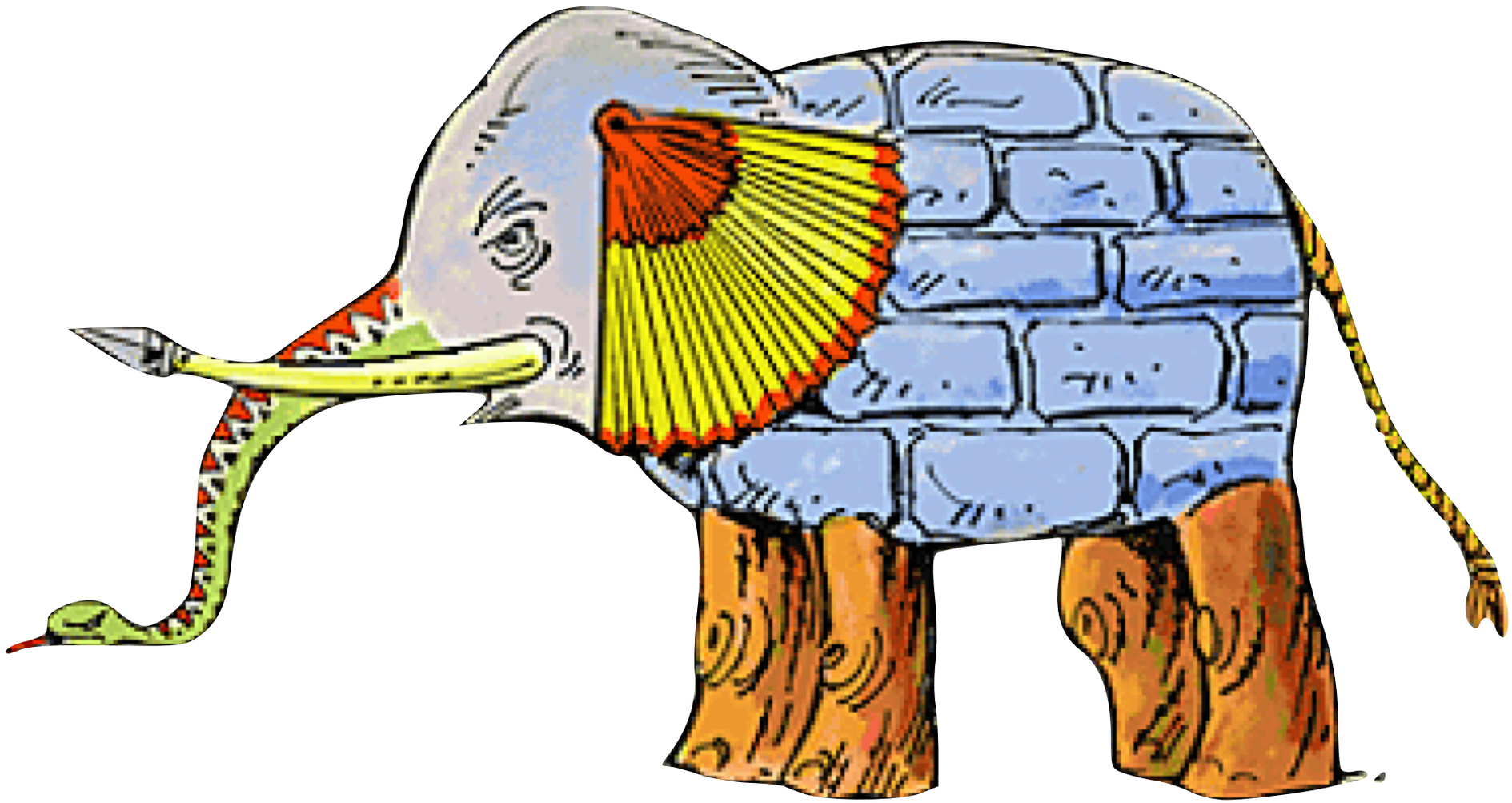
ATM process
to Bank : *card_id*;
to Bank : *pin*;
from Bank : *approval*

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*

Distributed Systems | **Service-Orientation**

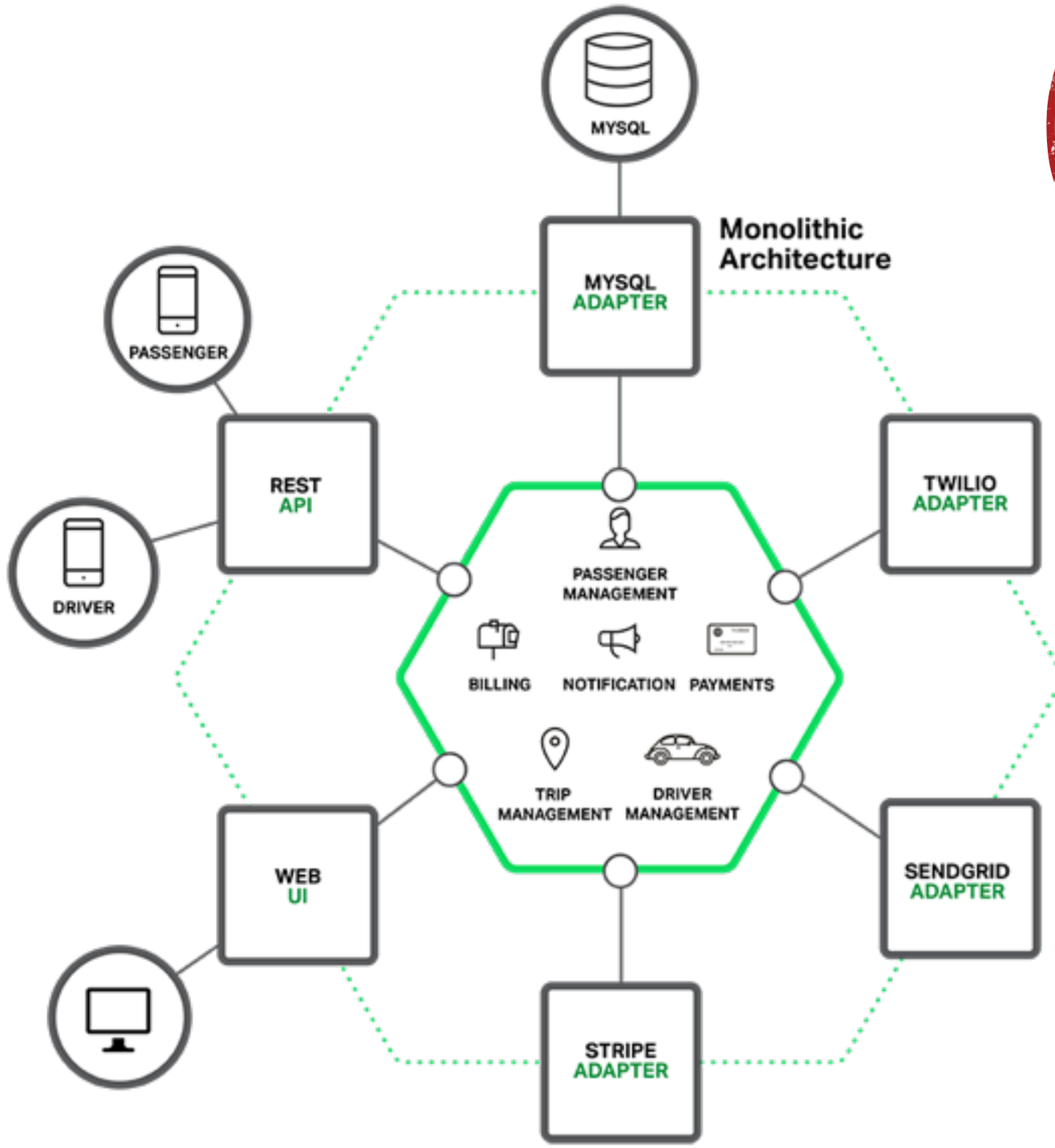
Saxe's Elephant



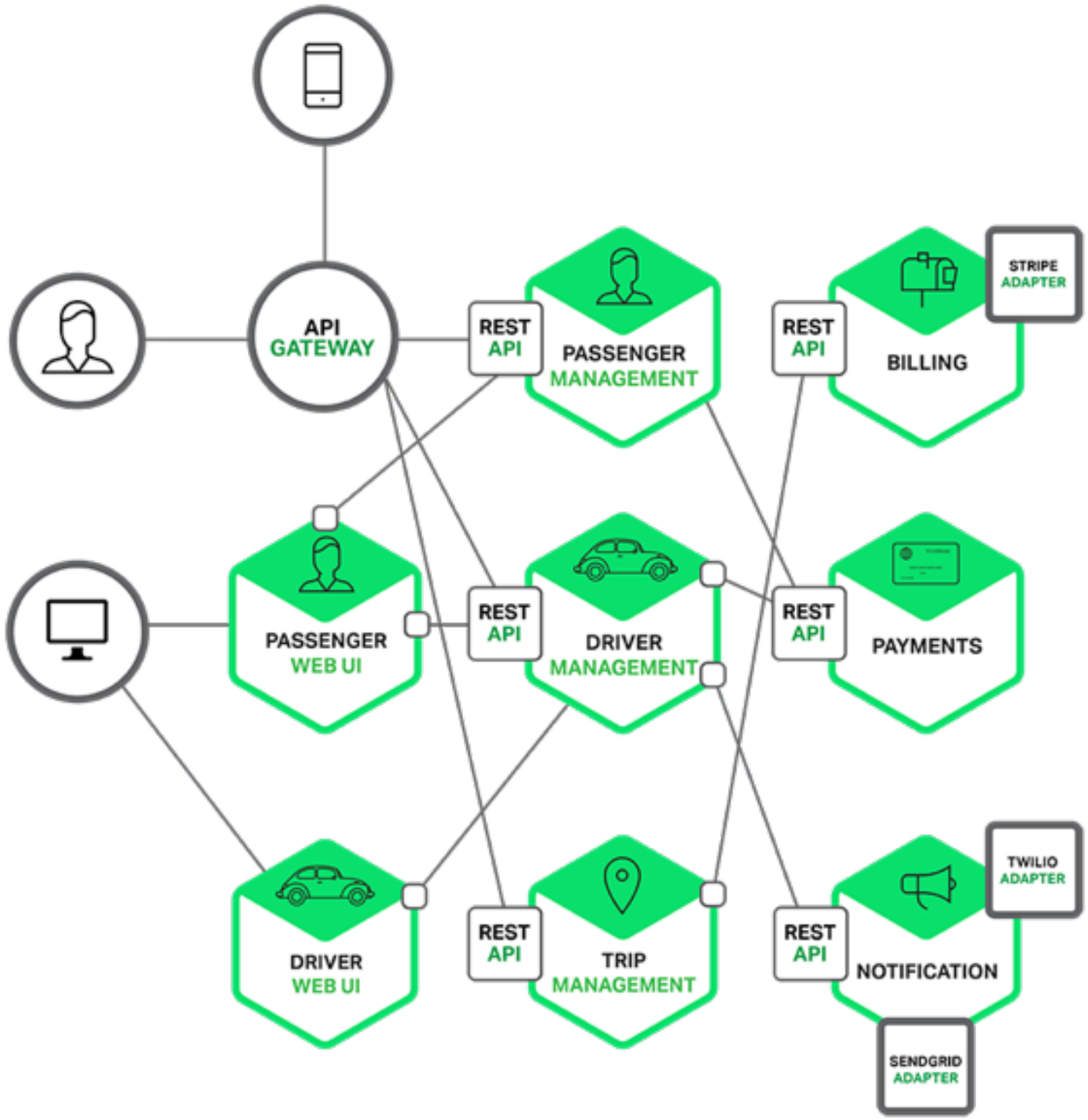
Distributed Systems | **Service-Orientation**

Zeitgeist

Distributed Systems | **Service-Orientation**

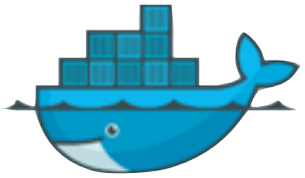


Distributed Systems | **Microservices**



Distributed Systems | **Microservices**

Lends to containerisation



Interacted through well-defined message exchanges

Designed for both availability and stability

RESTful "interfaces"

Microservice

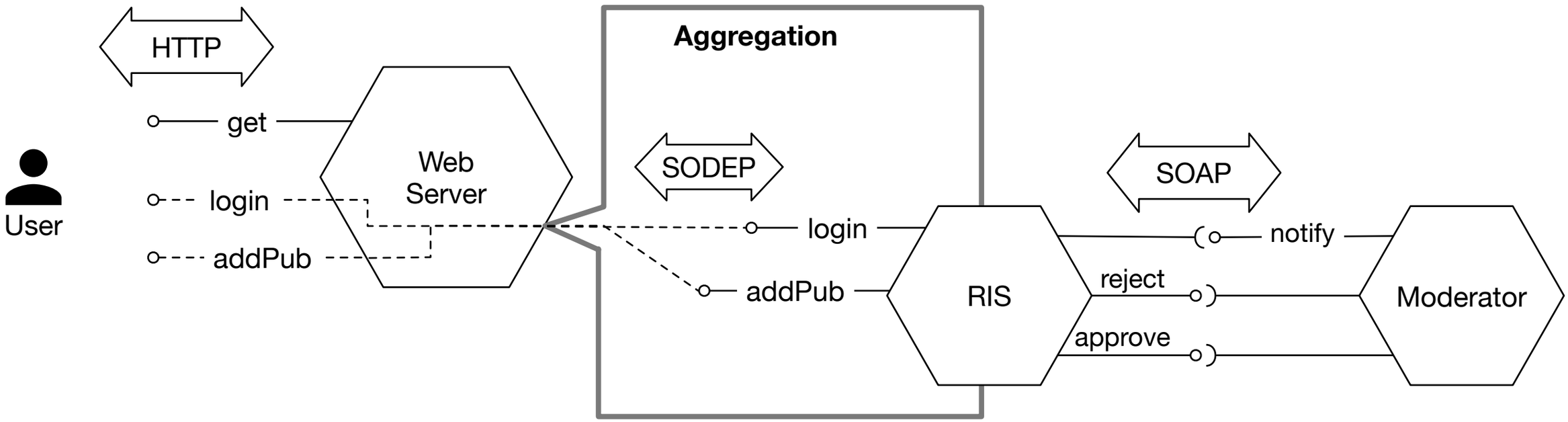
Limited knowledge of **how messages** are passed to or retrieved from it

Implements simple, granular functions

It is **service configurations** and **aggregations** that change (loosely-coupled infrastructure).

Distributed Systems | **Microservices**

Microservices | Jolie

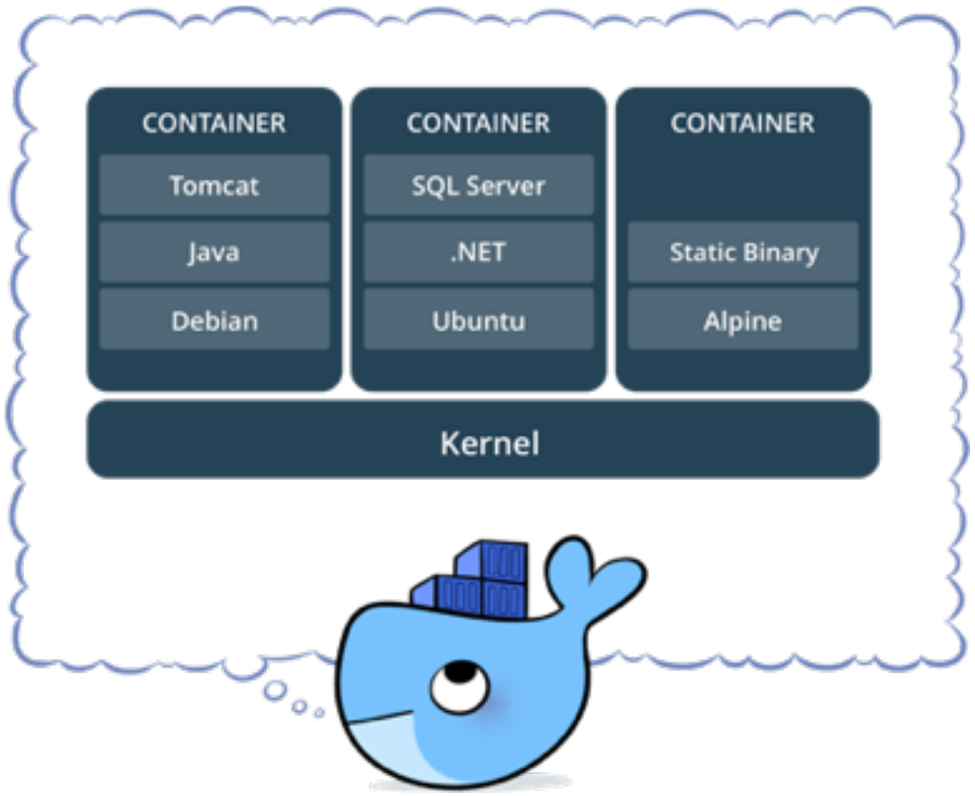


Distributed Systems | **Microservices**

Deployment vs Programming

System Deployment

Independent applications enclosed within **containers**.

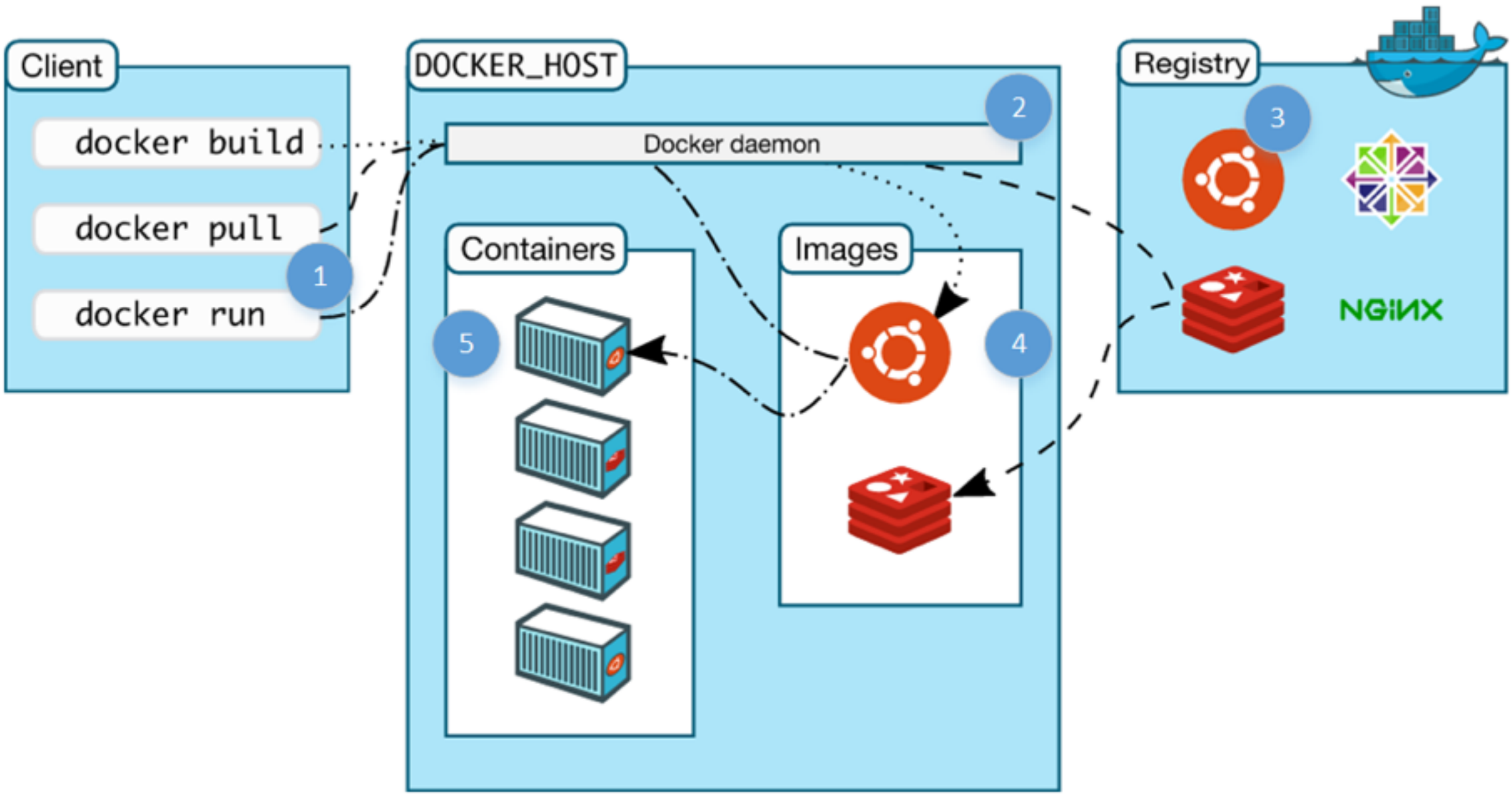


System Programming

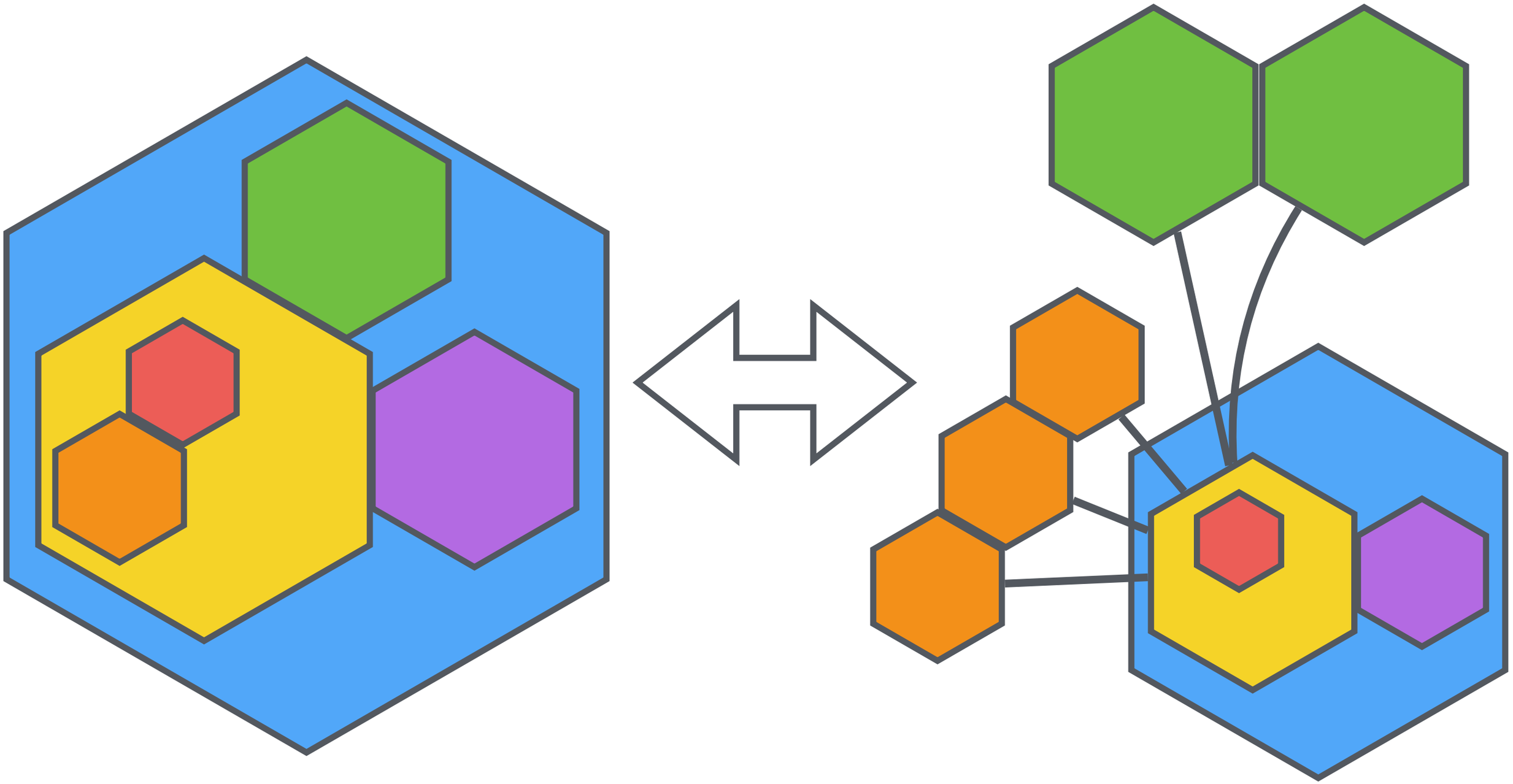
Independent **microservices**, possibly enclosed within containers.

Distributed Systems | **Microservices**

System Deployment



Distributed Systems | **Microservices** System Programming



Distributed Systems | **Serverless**

Run without provisioning or managing servers

Executes only when needed and scales automatically

Lambda

run on a compute fleet that automatically handles memory, CPU, network, and other resources

cannot log in to compute instances, or customise the operating system or language runtime

Distributed Systems | **Serverless**

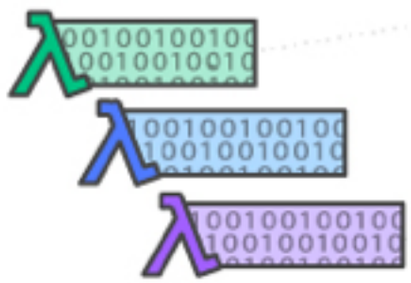
1

Upload your bot code to AWS Lambda

2

Set up your code to trigger when user commands are sent to your bot. The commands are API requests routed through API Gateway to Lambda

3

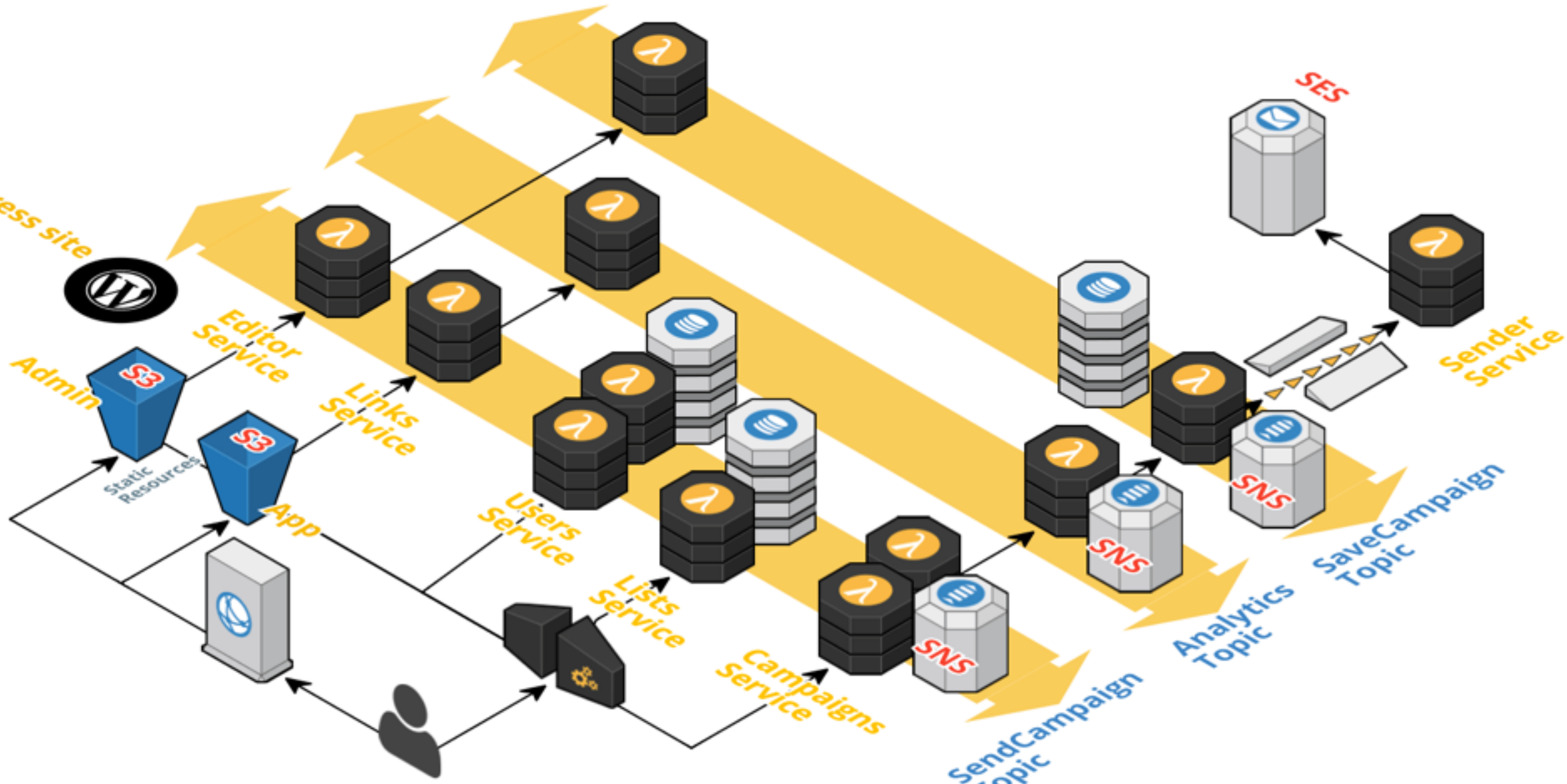


Lambda runs your bot code only when triggered by commands, using only the compute resources needed

4

Pay just for the compute time you use

Distributed Systems | **Serverless**



Distributed Systems | **Micro-Serverless?**

Microservice

```
inputPort TwiceService {
  Location: "socket://localhost:8000"
  Protocol: sodep
  RequestResponse: twice( int )( int )
}
```

Lambda

```
main
{
  twice( number )( result ) {
    result = number * 2
  }
}
```