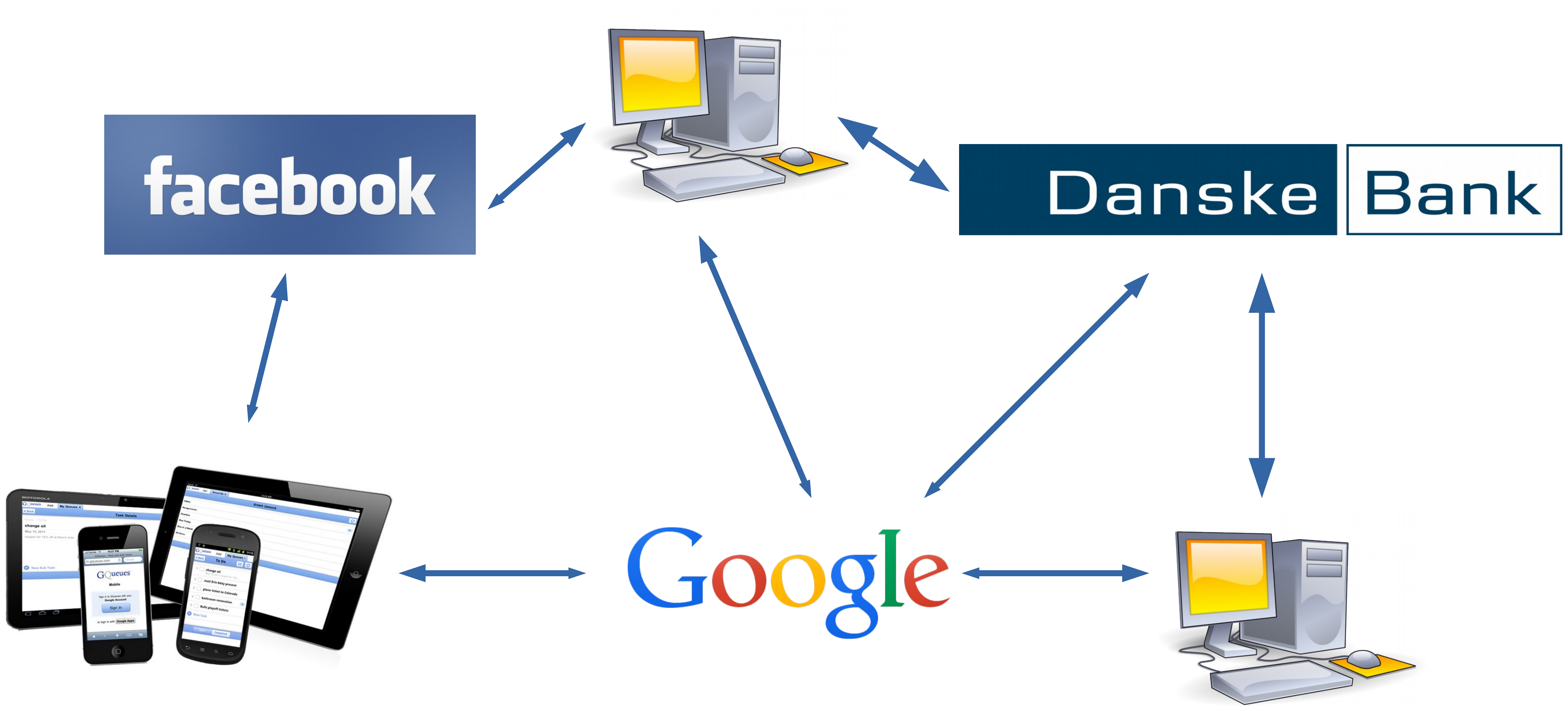
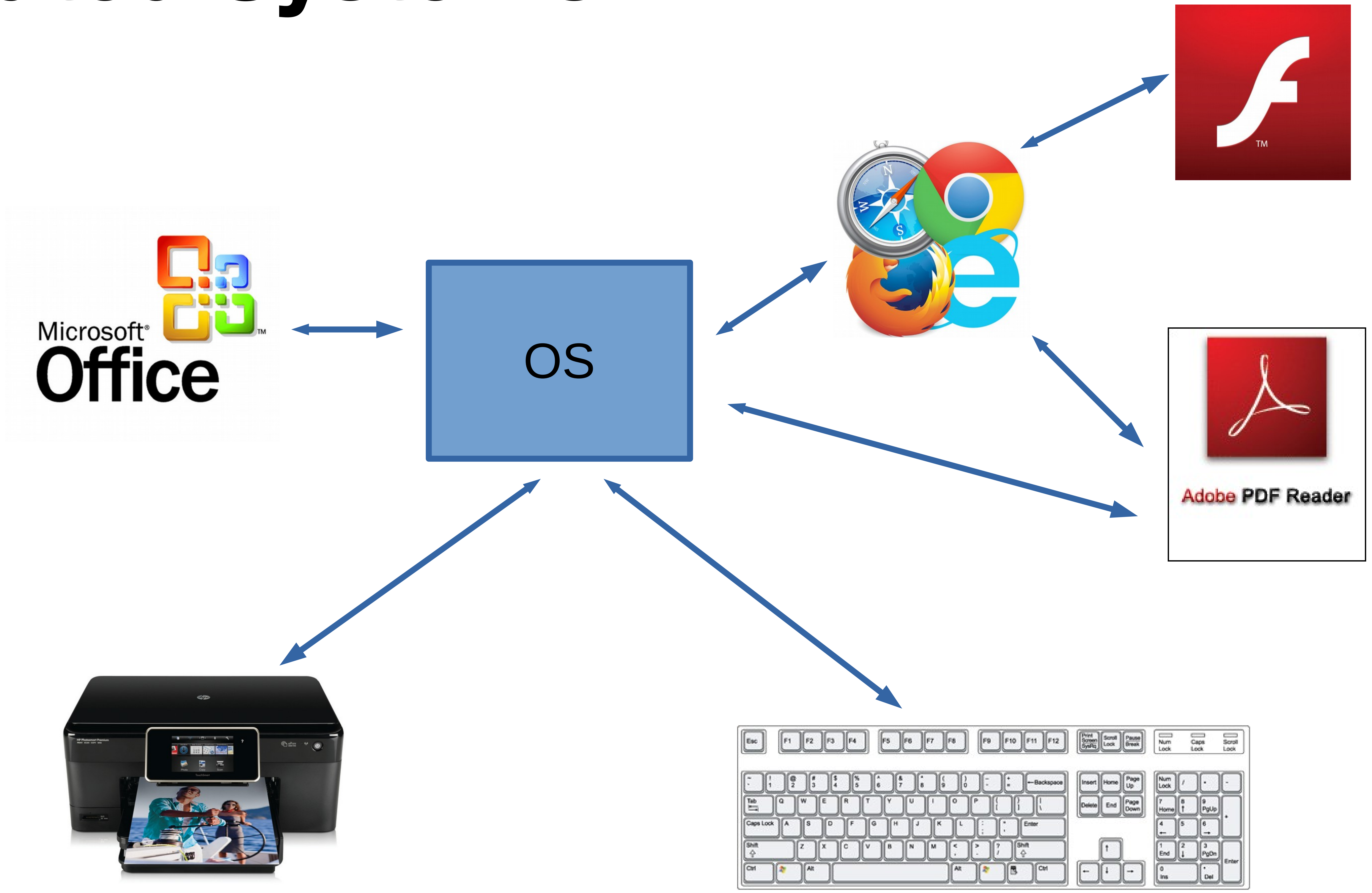
From **Service-Oriented Computing** to **Microservices and Beyond**

Distributed Systems



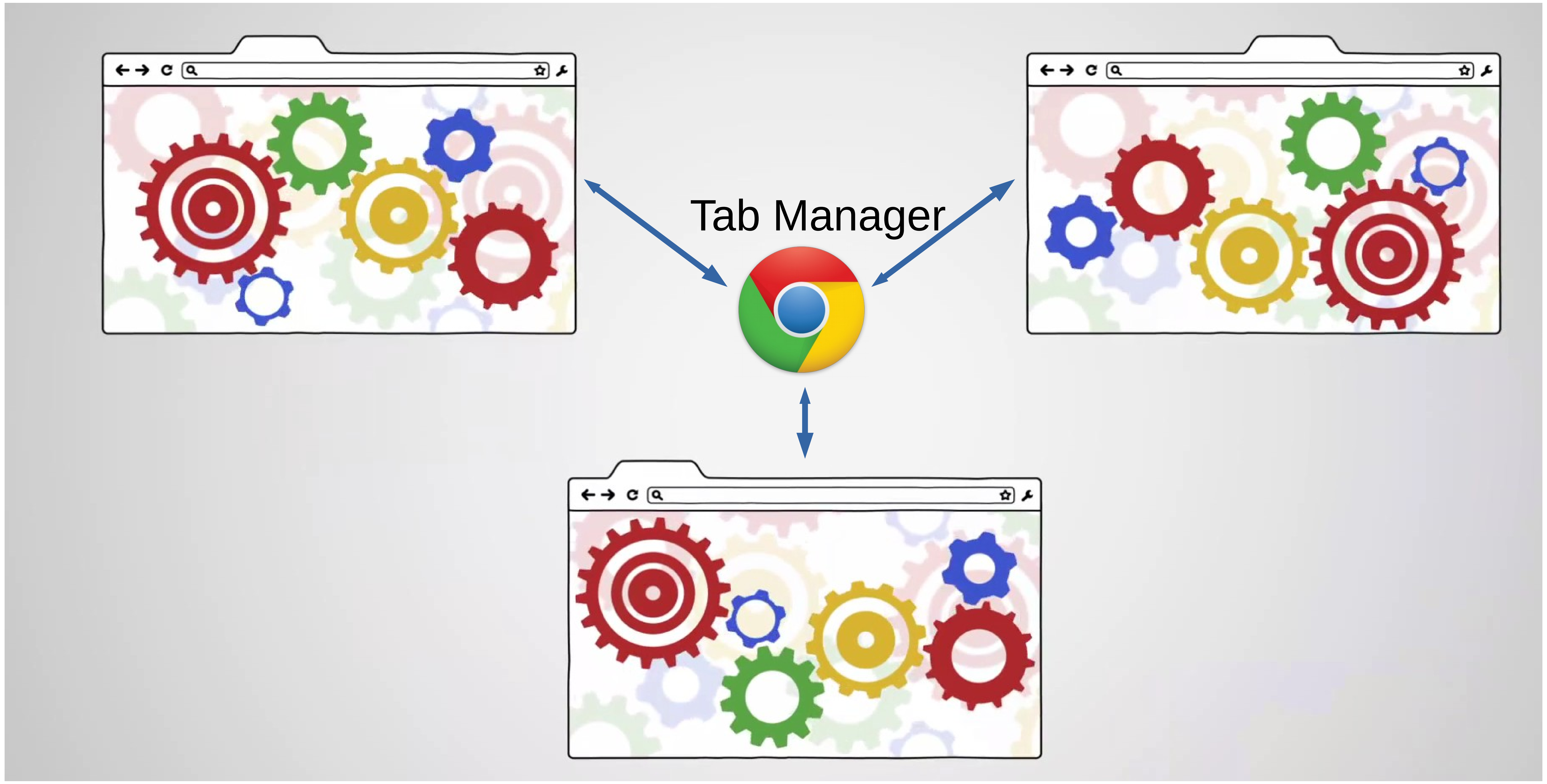
Courtesy of Fabrizio Montesi

Distributed Systems



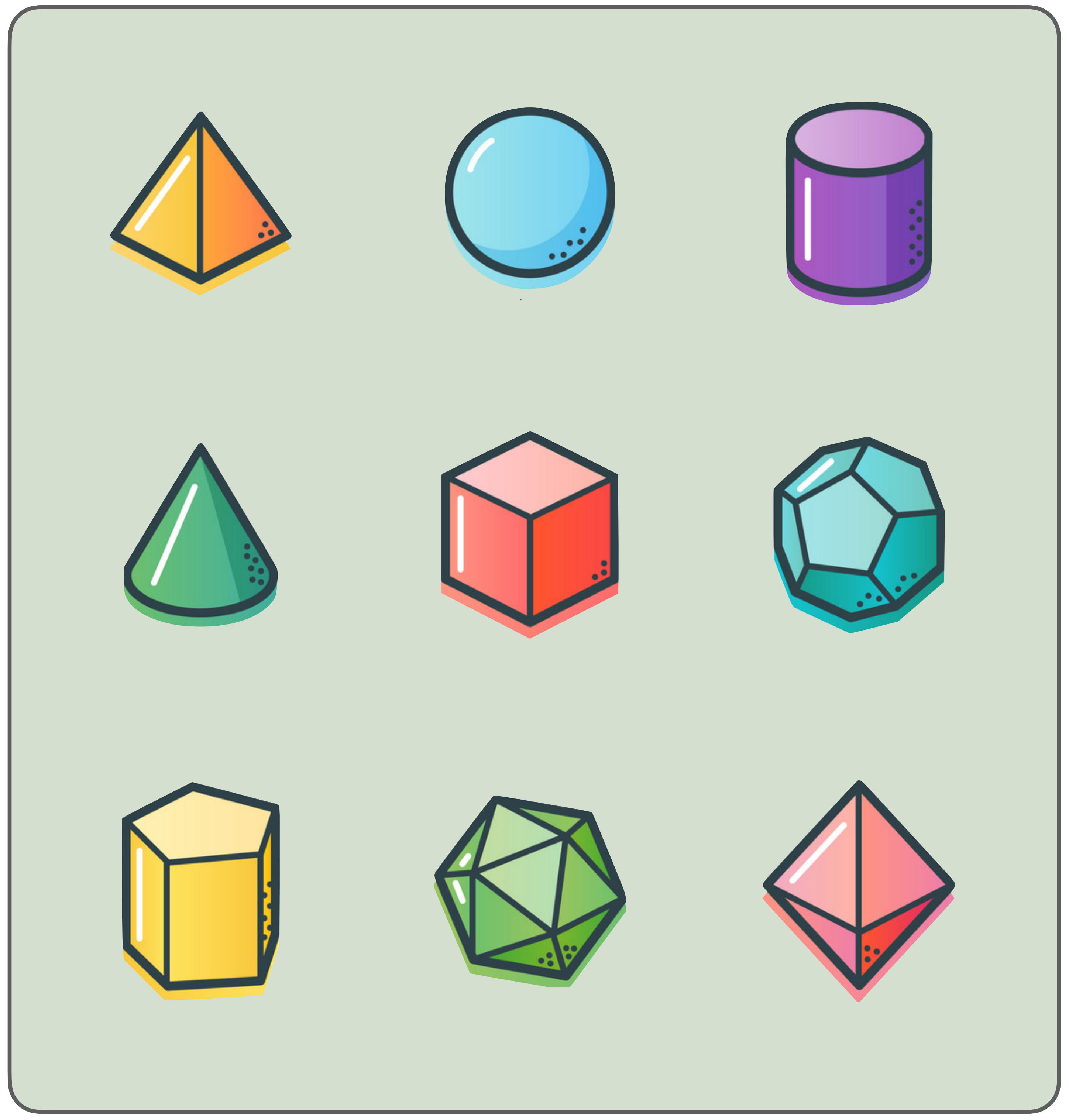
Courtesy of Fabrizio Montesi

Distributed Systems

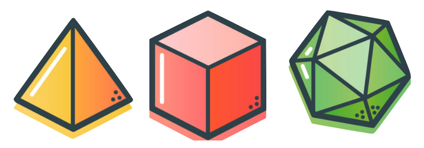


Courtesy of Fabrizio Montesi

From Monoliths



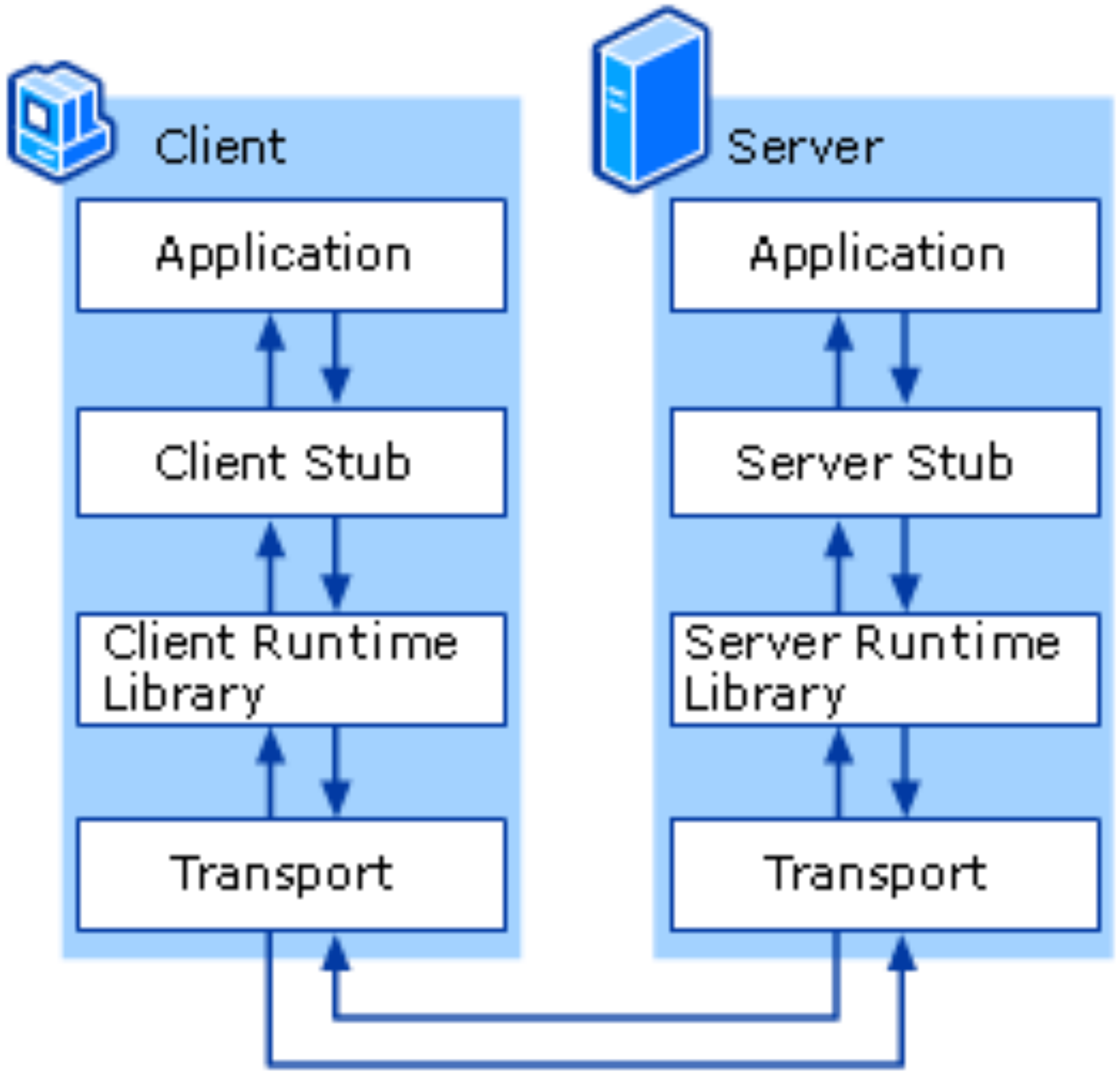
Monolith

 Function

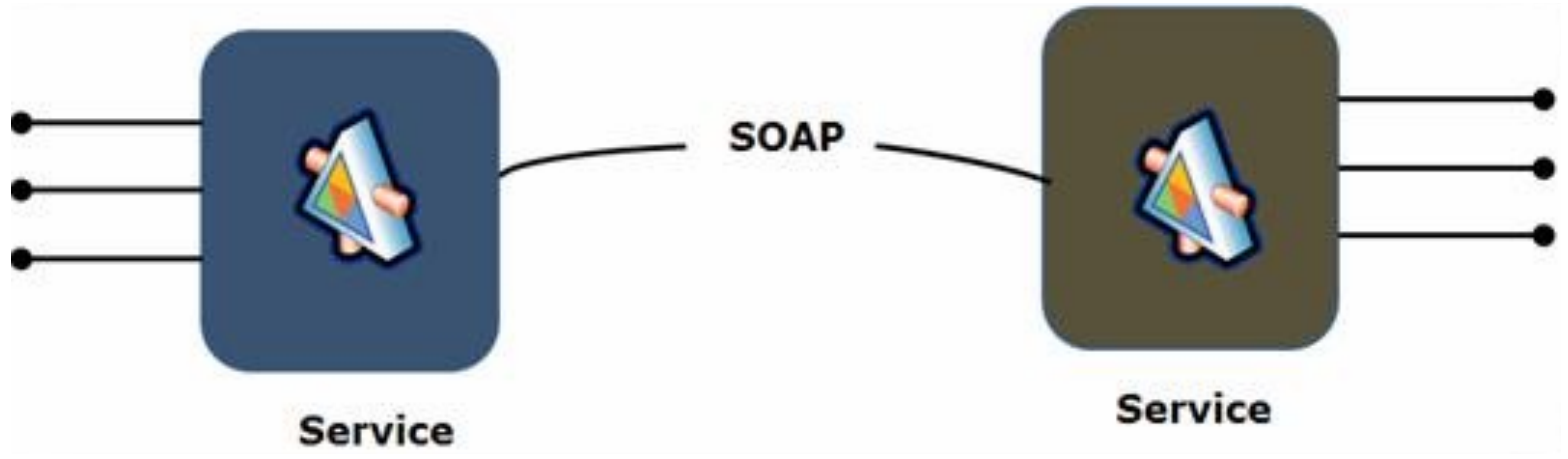
Software Unit

Runtime Environment

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

Service

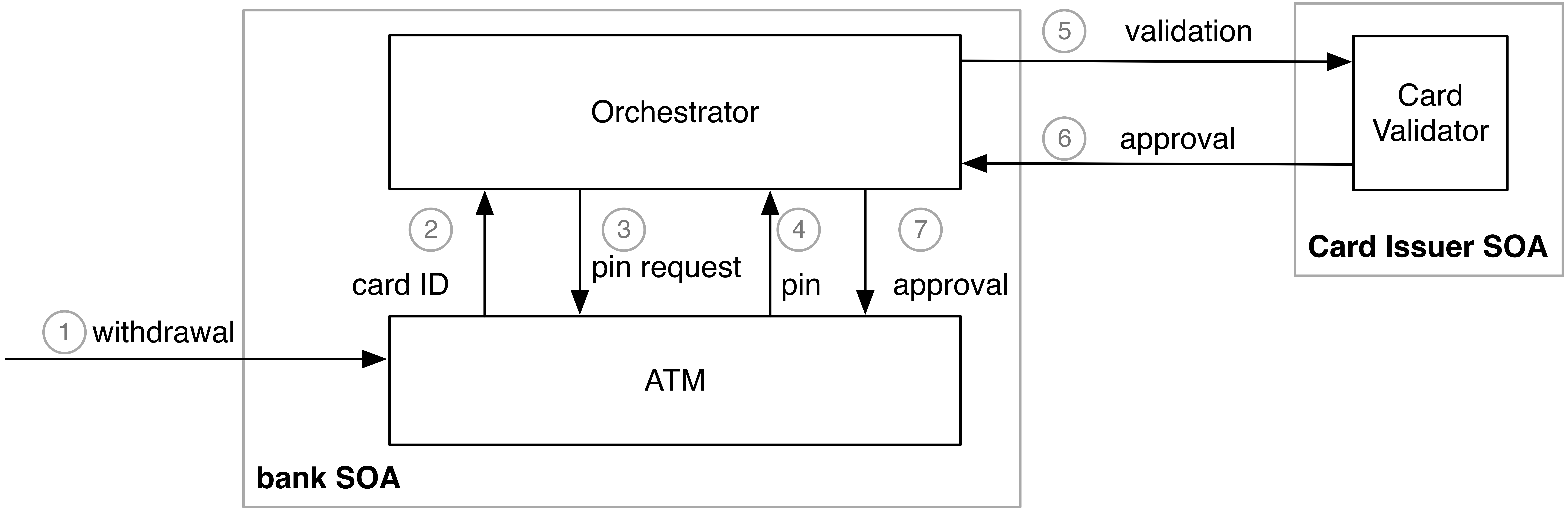
Limited knowledge of **how messages** are passed to or retrieved from it

Implements simple, granular functions

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

Distributed Systems | Service Composition

Orchestration



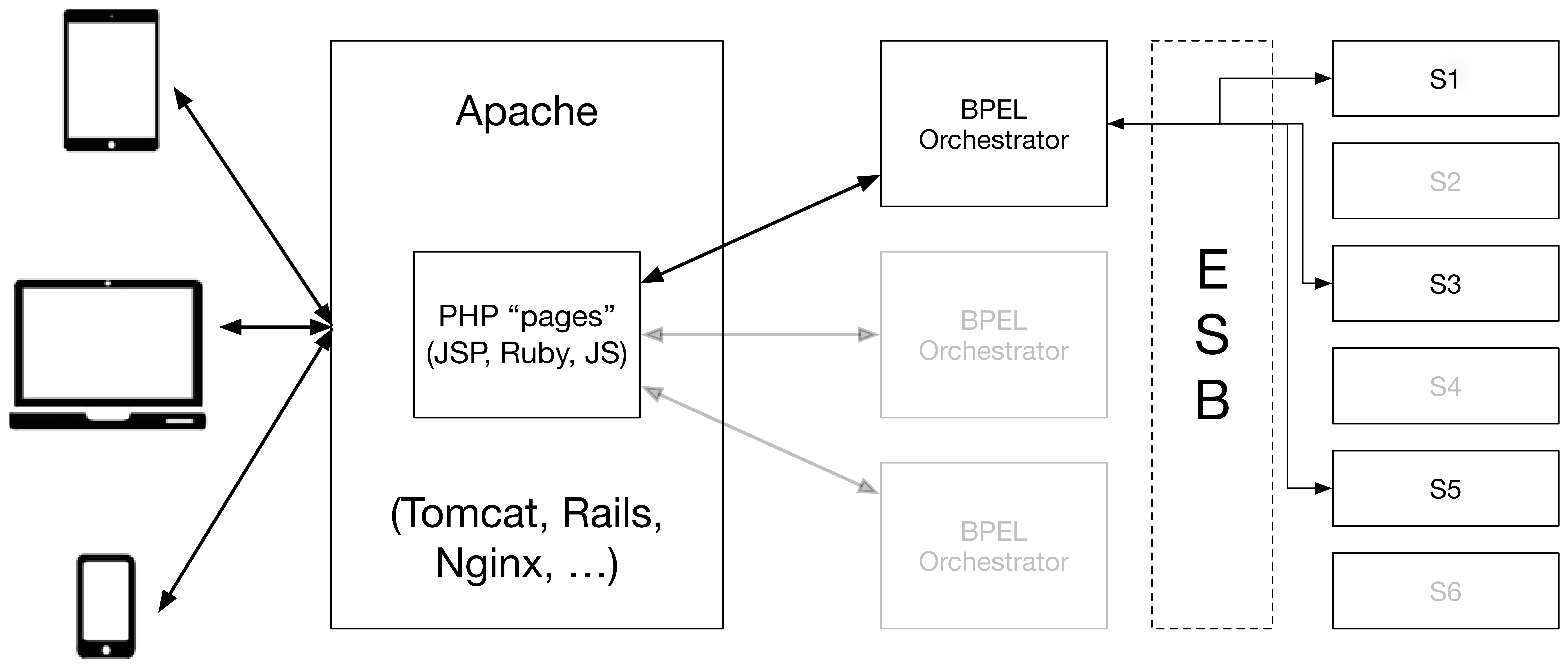
Distributed Systems | Service Composition

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"/>
```

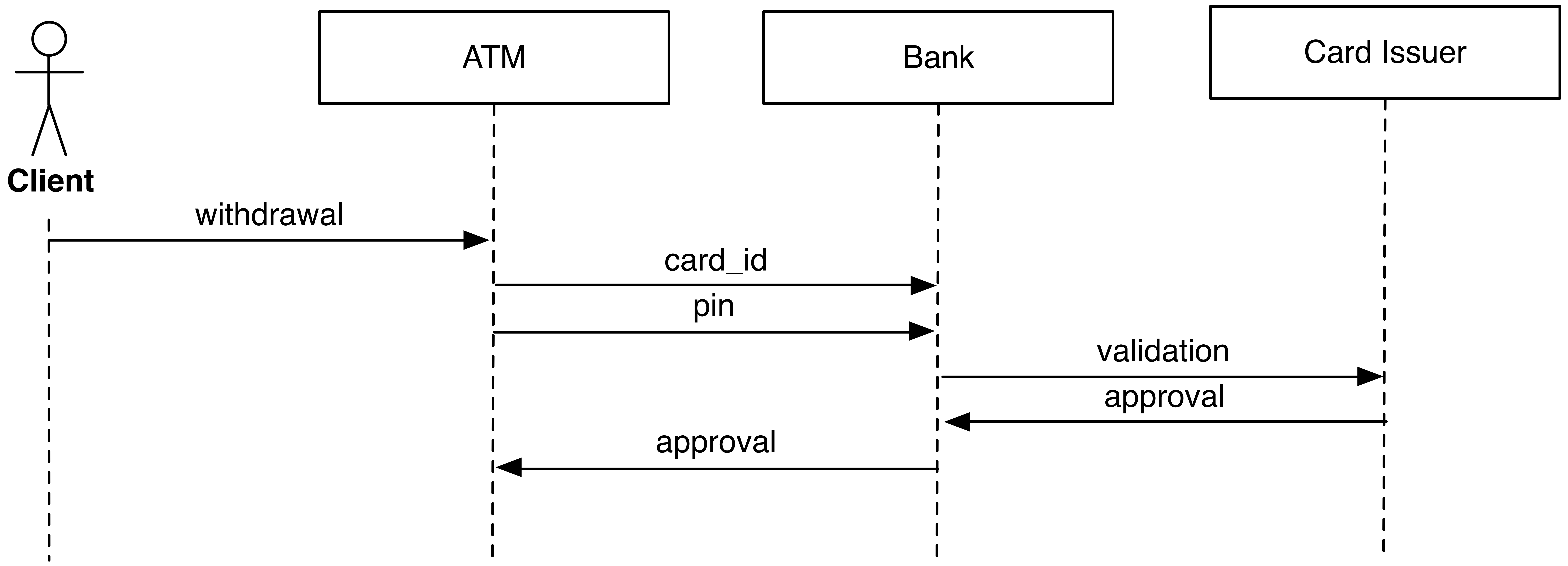
Distributed Systems | Service Composition

Orchestration • WS-BPEL



Distributed Systems | Service Composition

Choreographies



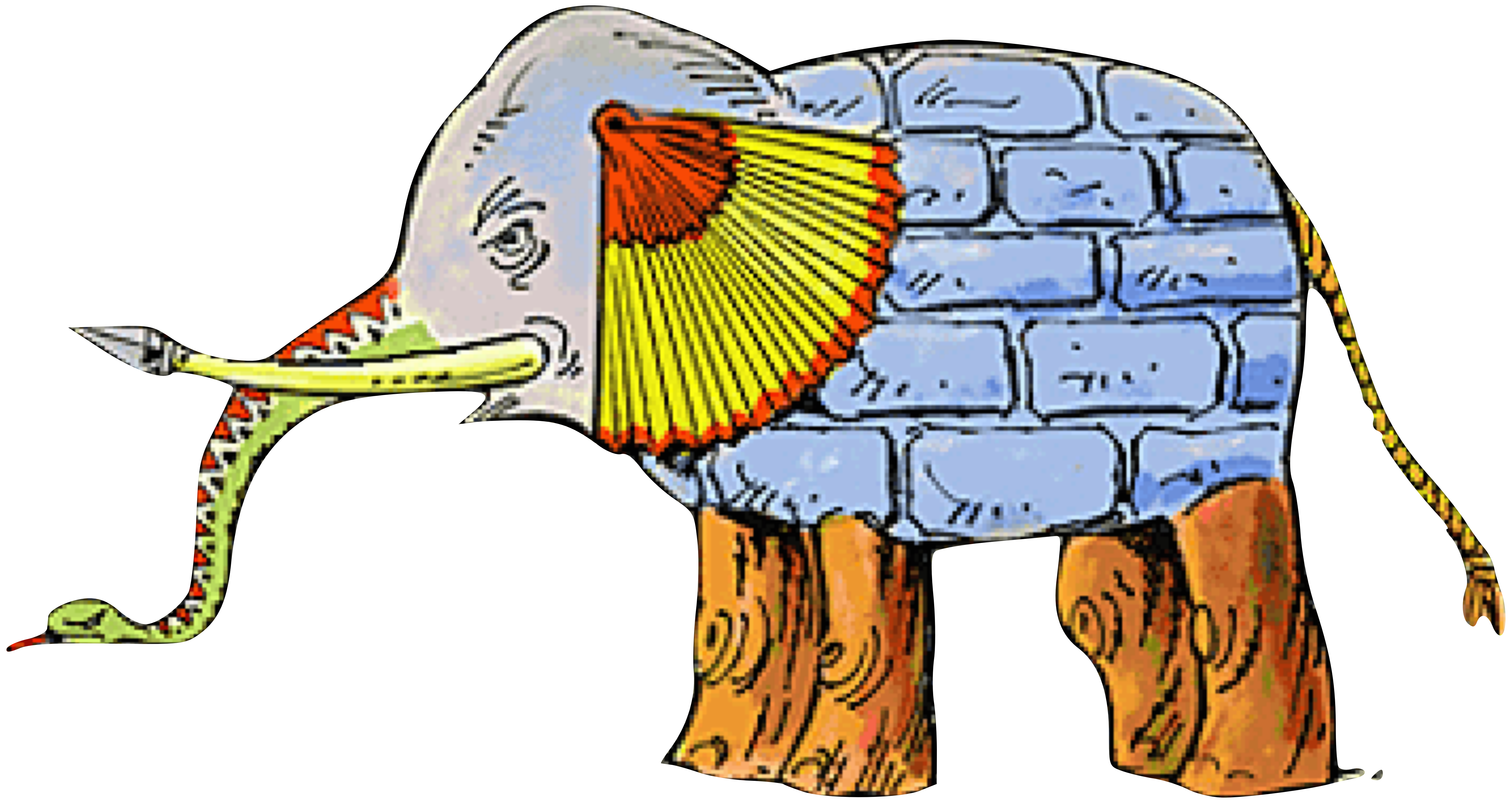
Distributed Systems | Service Composition

Choreographies • WS-CDL

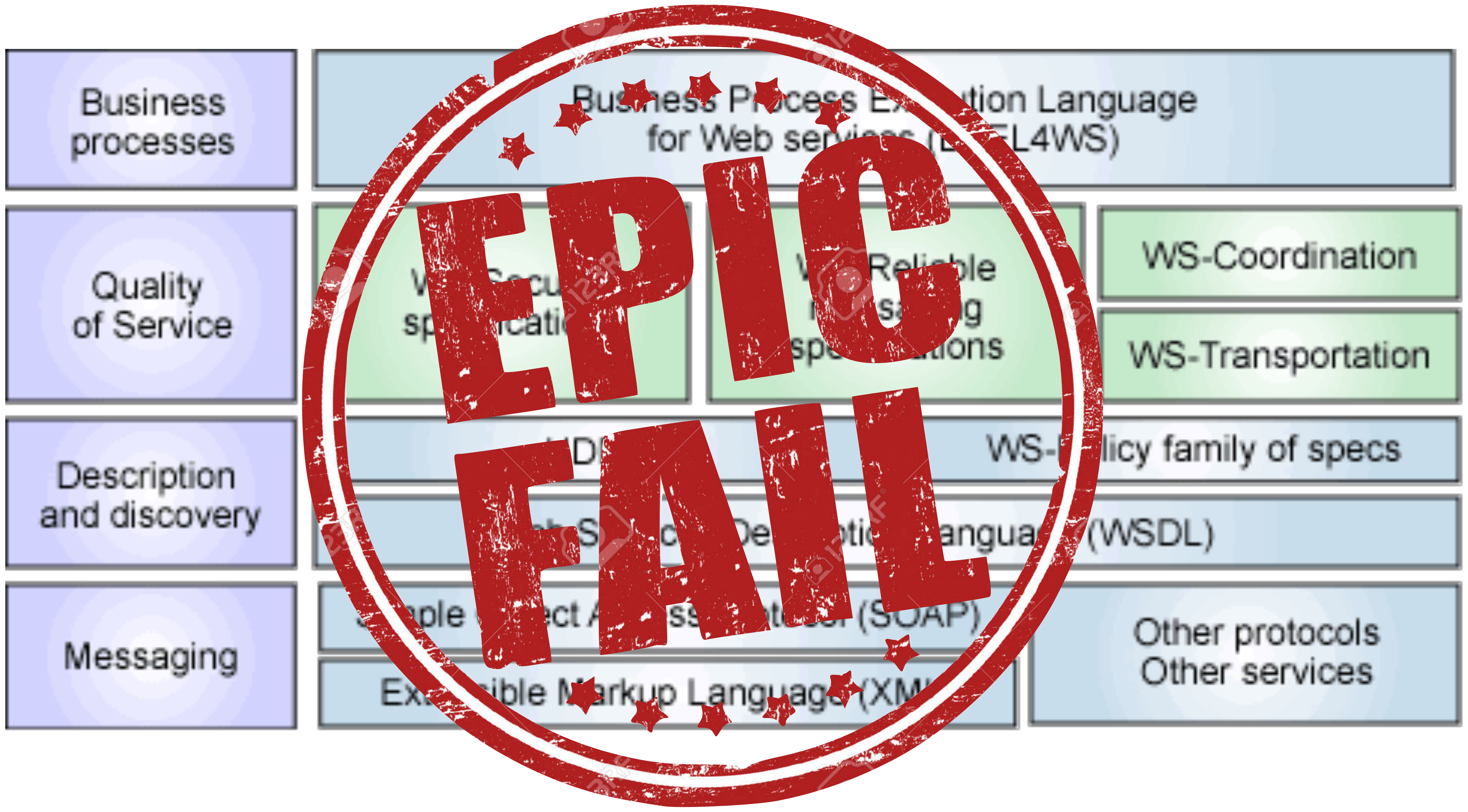
```
<choreography name="CreditAuthorization" root="false" coordination="true">
  <relationship type="tns:CreditReqCreditResp"/>
  <variableDefinitions>
    <variable name="CreditExtended" informationType="xsd:int" silent="true"
              roleTypes="tns:CreditResponder"/>
    <variable name="creditRequest"/>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

Saxe’s Elephant

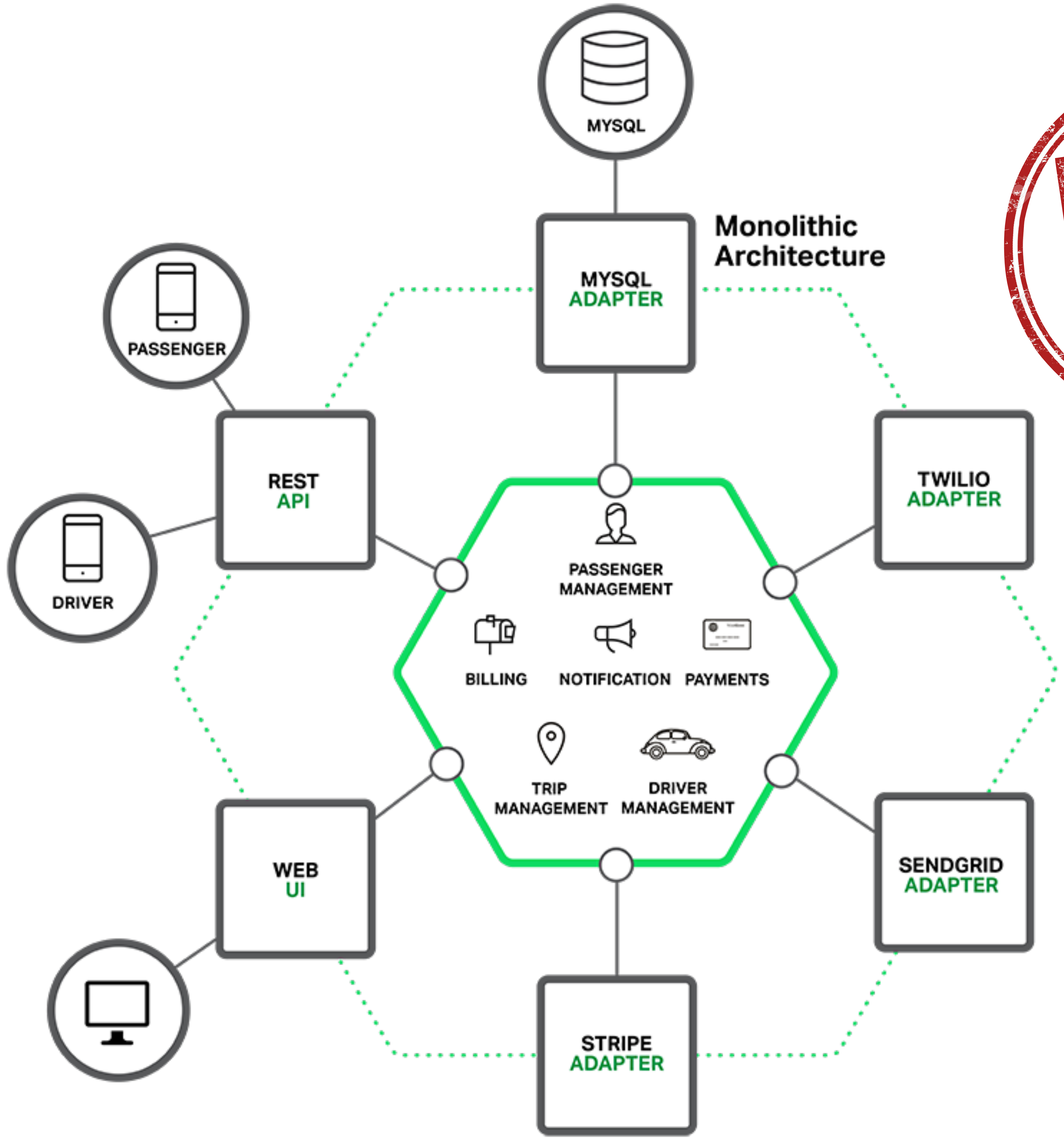


Distributed Systems | Service-Orientation

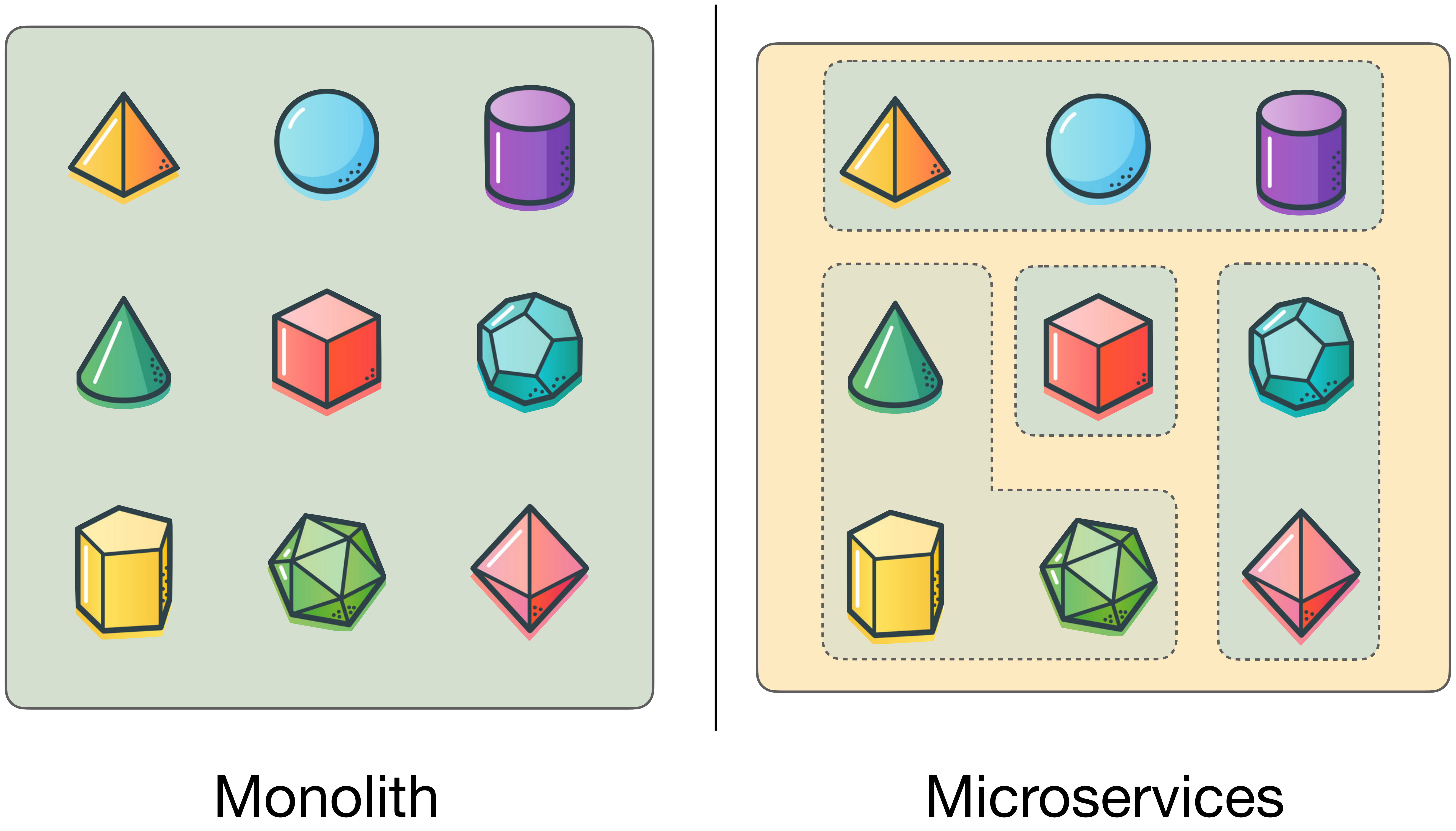

Business processes
Business Process Execution Language for Web services (BPEL4WS)
Quality of Service
WS-Security specifications
WS-Reliable Messaging specifications
WS-Coordination
WS-Transportation
Description and discovery
UDDI
WS-Policy family of specs
Web Services Description Language (WSDL)
Messaging
Simple Object Access Protocol (SOAP)
Extensible Markup Language (XML)
Other protocols
Other services
EPIC FAIL

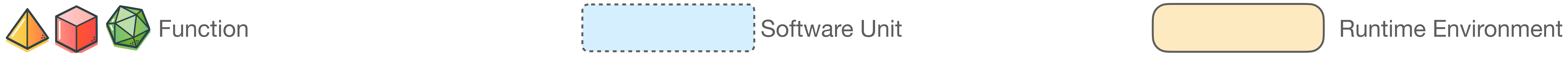
Zeitgeist

Distributed Systems | Service-Orientation

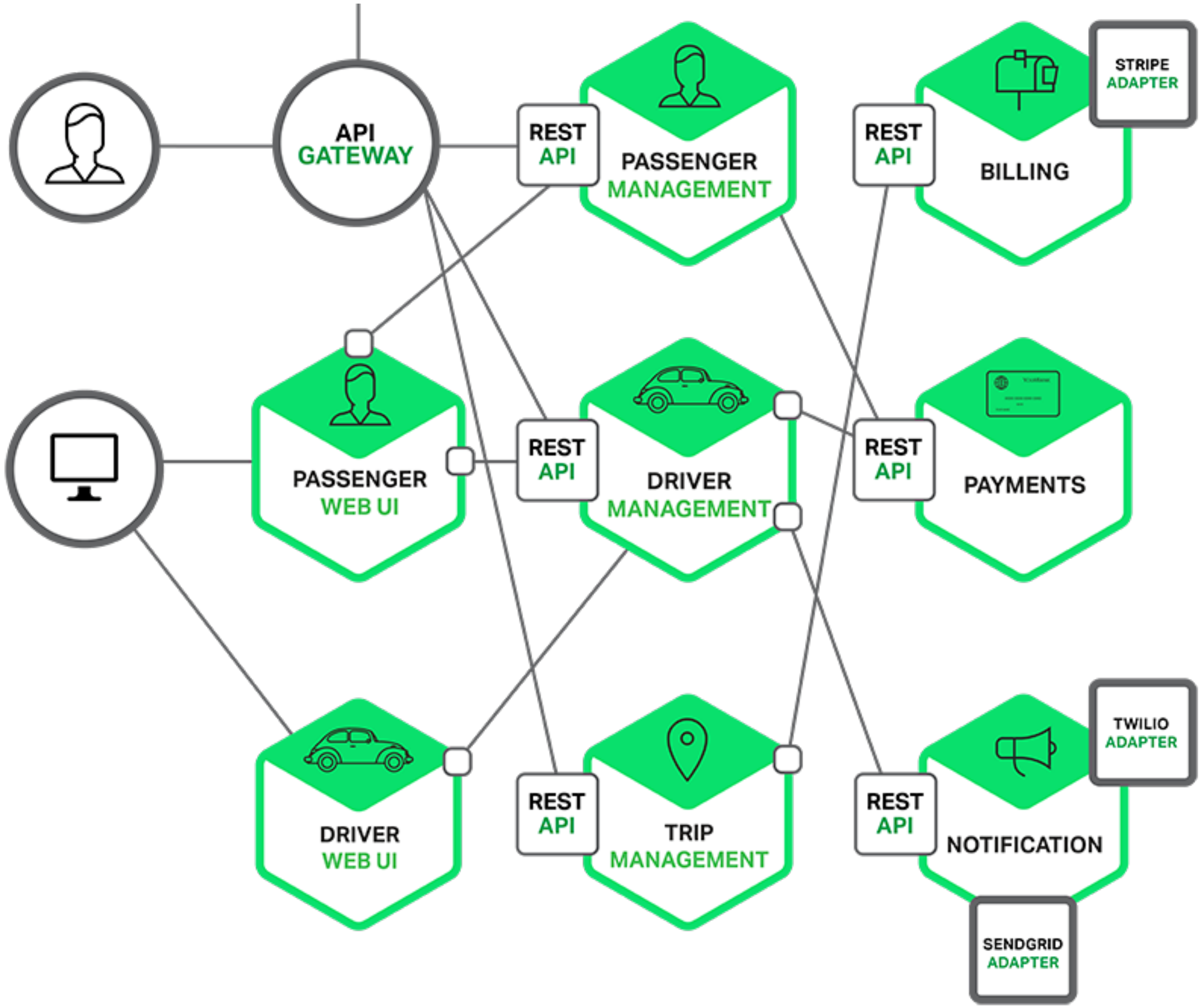


From Monoliths to Microservices





Distributed Systems | Microservices



Distributed Systems | Microservices

Lends themselves to containerisation

~~Interacted through well-defined message exchanges~~

RESTful “interfaces”

Designed for both availability and stability

Microservice

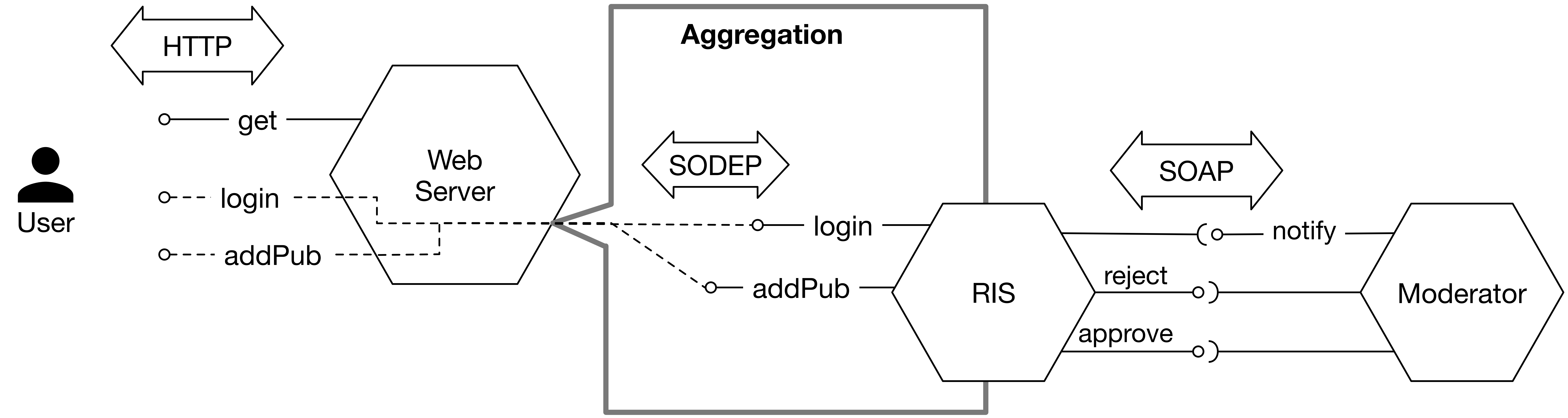
Limited knowledge of **how messages** are passed to or retrieved from it

Implements simple, granular functions

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

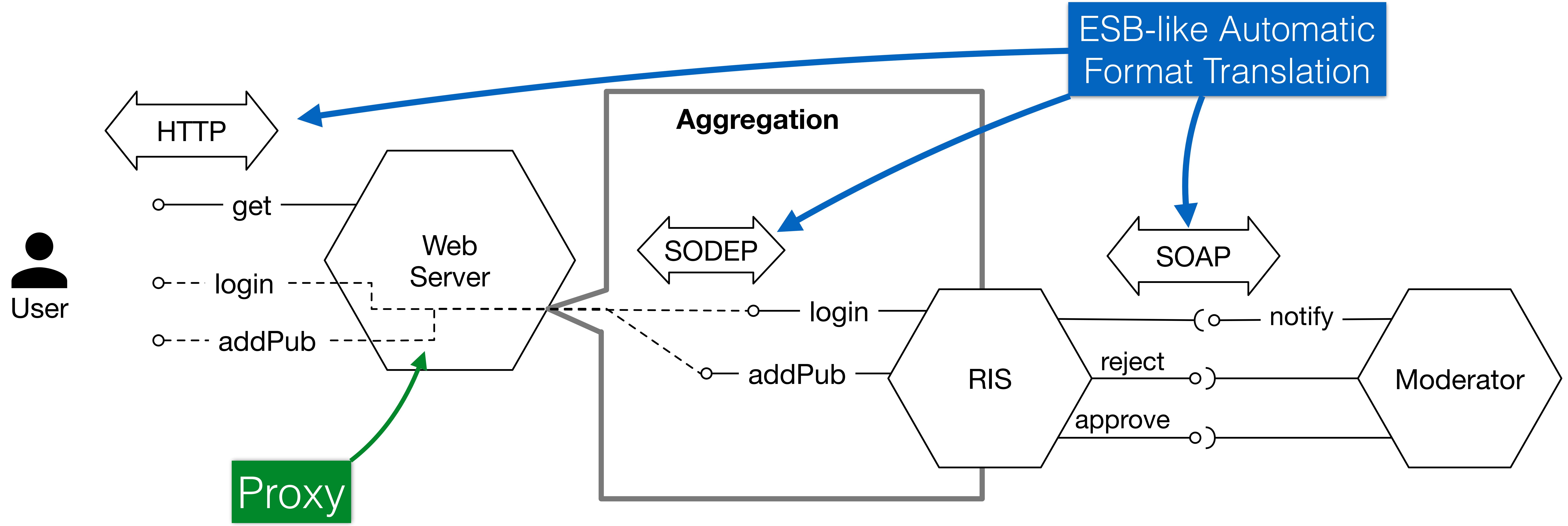
Distributed Systems | Service Composition

Microservices • Jolie



Distributed Systems | Service Composition

Microservices • Jolie



Distributed Systems | Service Composition

Microservices • Jolie

Deployment

```
outputPort CardValidator {
  Location: "socket://localhost:8000"
  Protocol: http
  Interfaces: CardValInterface
}
```

```
type ValidateRequest: {
 cardID: int
 pin: int
}

interface CardValInterface {
RequestResponse:
 validateID( ValidateRequest )( bool )
}
```

API

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

Behaviour

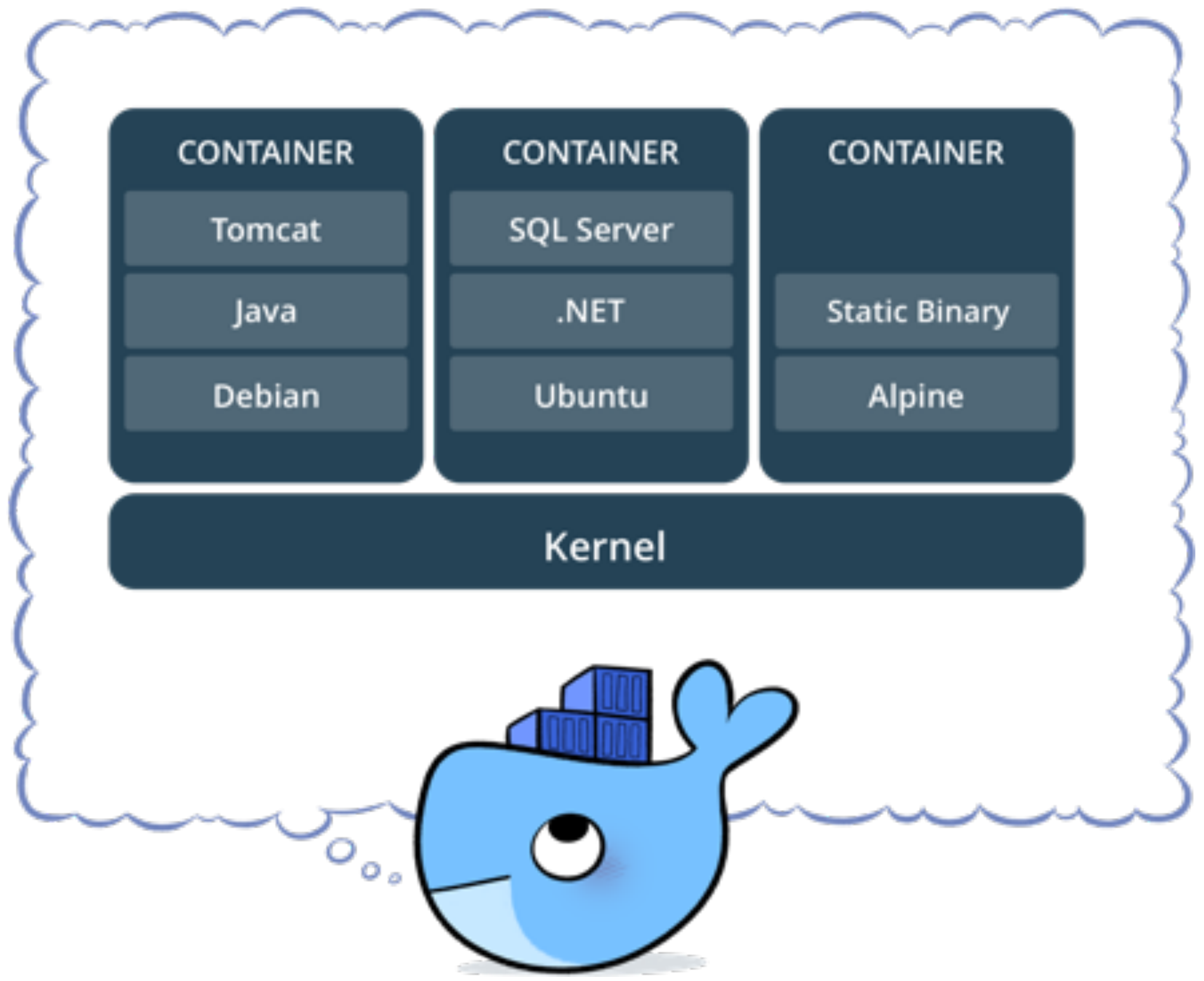
Behaviour

Distributed Systems | Microservices

Deployment vs Programming

System Deployment

Independent applications enclosed within **containers**.

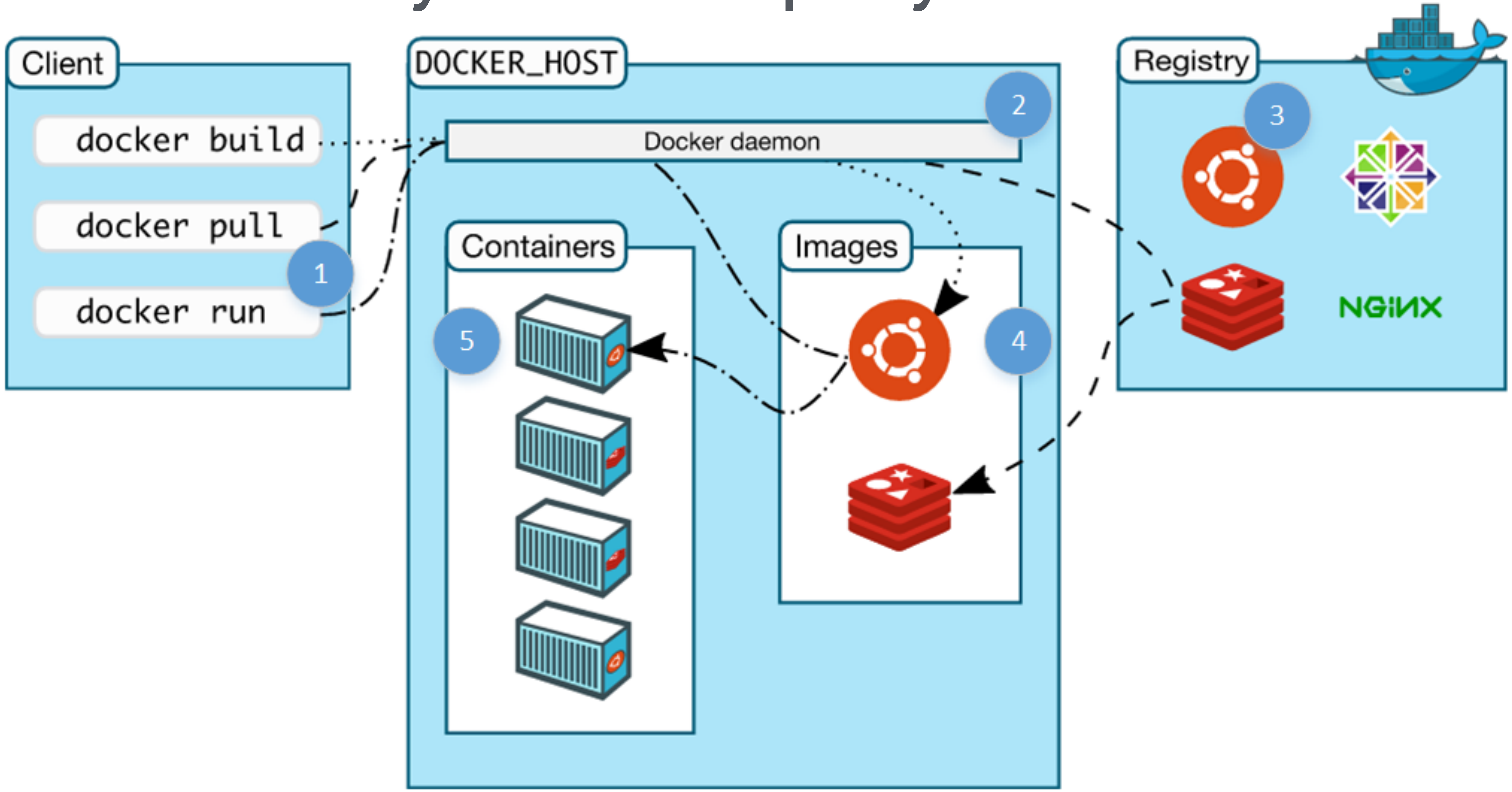


System Programming

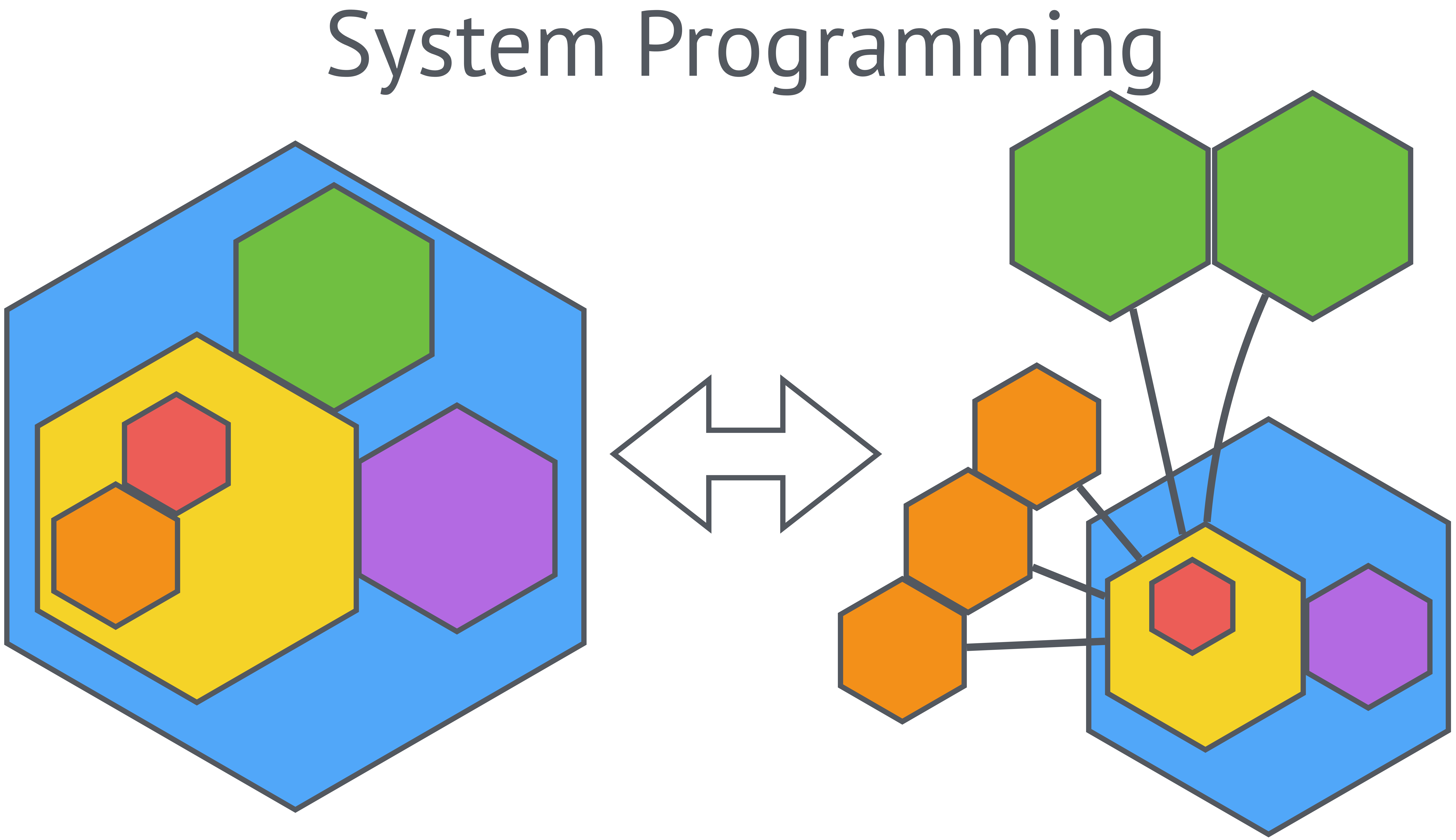
Independent **microservices**, possibly enclosed within containers.

Distributed Systems | Microservices

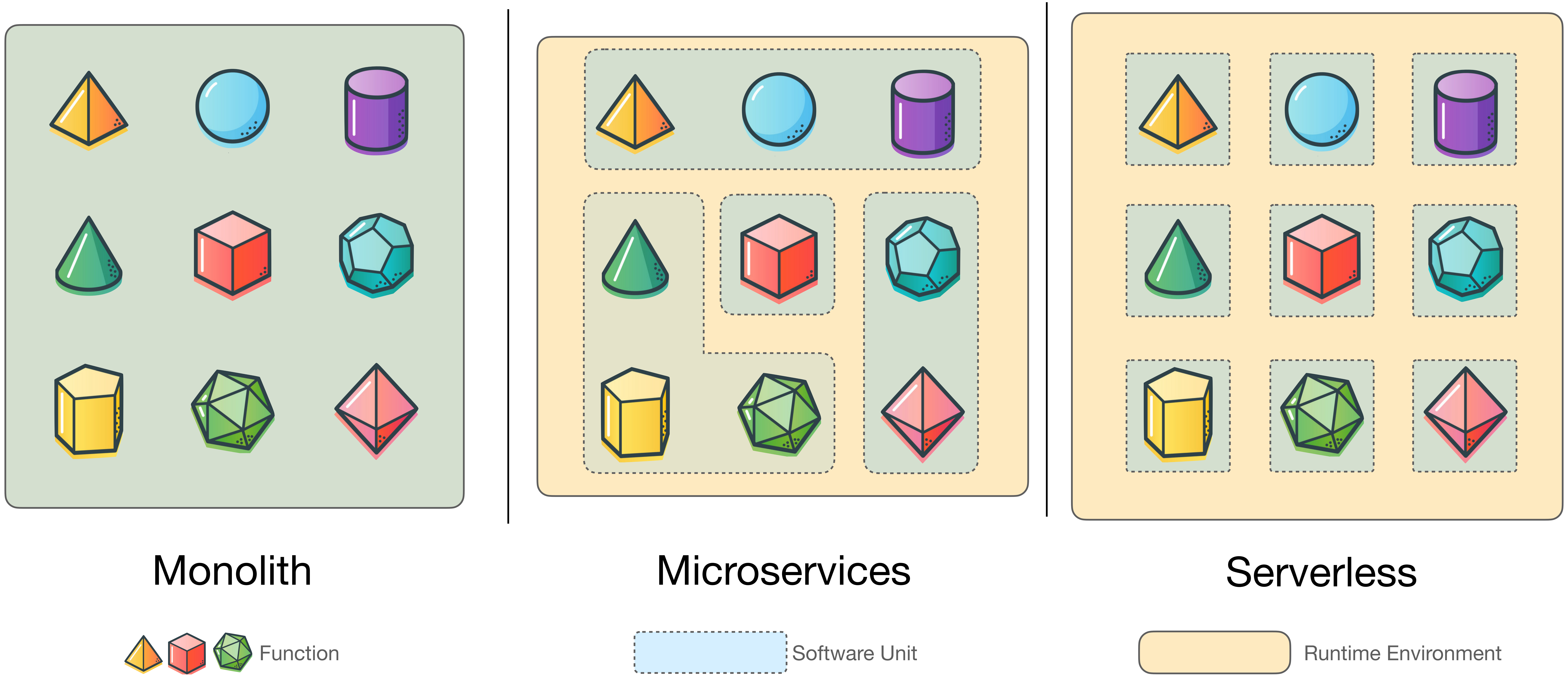
System Deployment



Distributed Systems | Microservices



From Monoliths to Microservices and Beyond



Distributed Systems | Serverless

Serverless

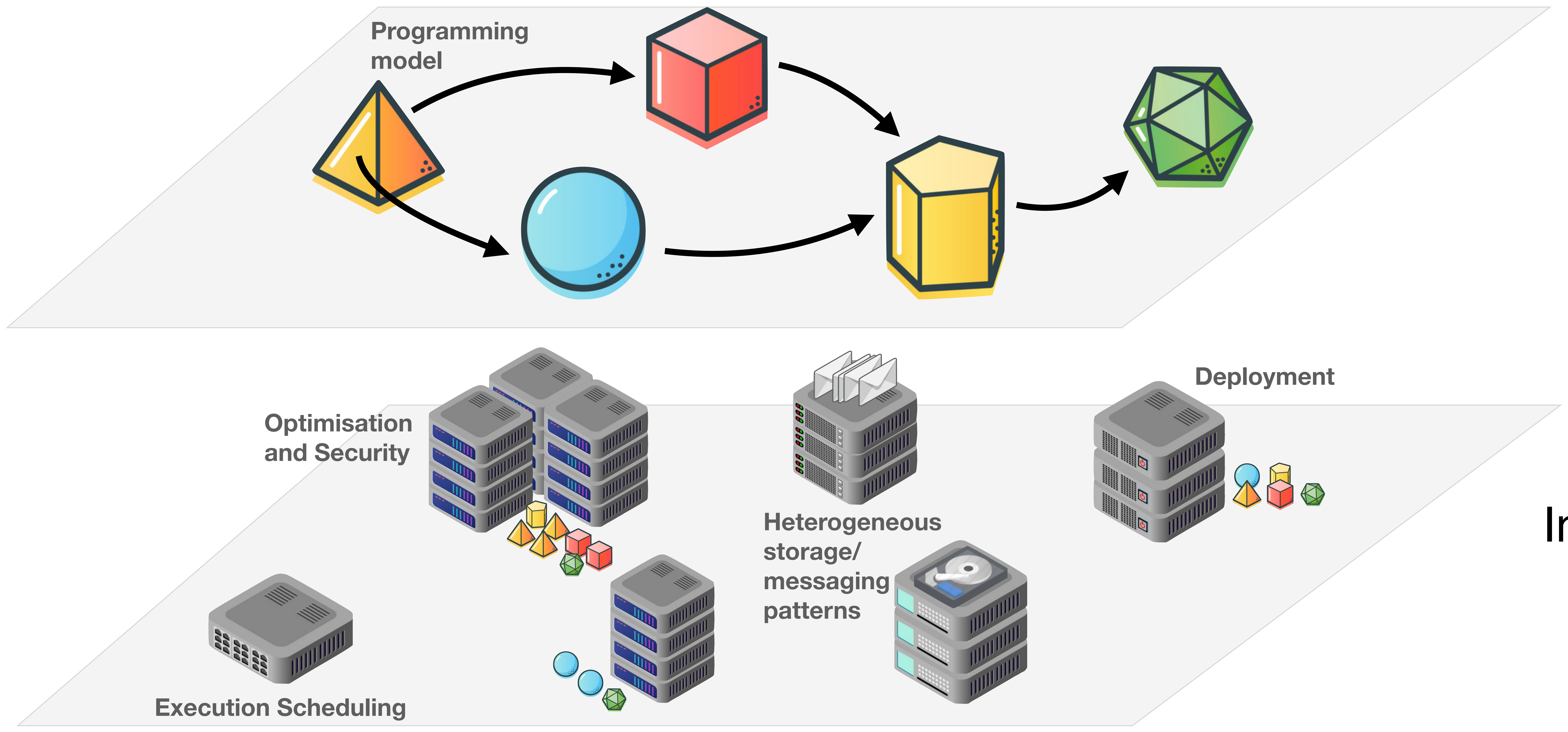
Run without provisioning or managing servers

Executes only when needed and scales automatically

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


Programming model
Optimisation and Security
Heterogeneous storage/ messaging patterns
Deployment
Execution Scheduling

Programming Layer

Implementation Layer

Distributed Systems | Micro-Serverless?

Microservice

```
inputPort TwiceService {
  Location: "socket://localhost:8000"
  Protocol: sodep
  Interface: TwiceInterface
}
```

Function

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

Distributed Systems | Service Composition

Choreographies • Chor/AIOCJ

$$\boxed{\text{Choreography}} \xrightarrow{\textit{EPP}} \boxed{\text{Endpoint Projection}}$$

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

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

$\xrightarrow{\textit{EPP}}$

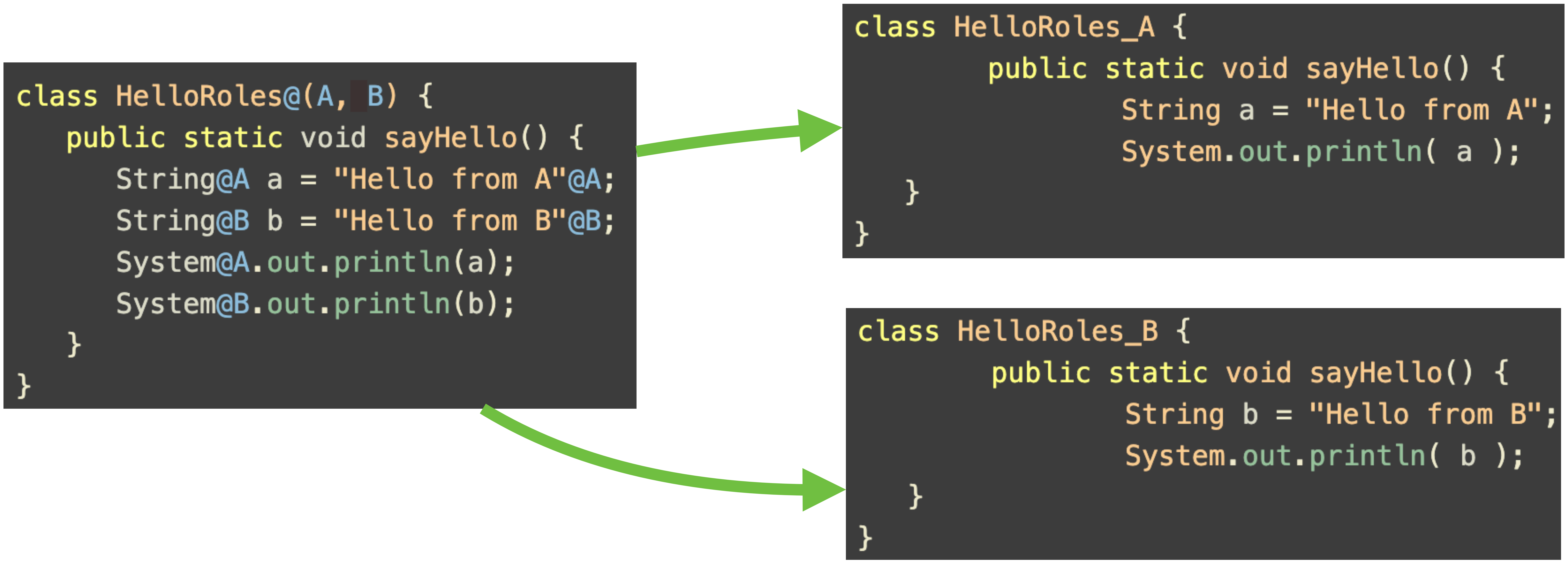
$\texttt{ATM}$ **process**
$\texttt{to Bank} : \textit{card\_id};$
$\texttt{to Bank} : \textit{pin};$
$\texttt{from Bank} : \textit{approval}$

$\texttt{Bank}$ **process**
$\texttt{from ATM} : \textit{card\_id};$
$\texttt{from ATM} : \textit{pin};$
$\texttt{to Card Issuer} : \textit{validation};$
$\texttt{from Card Issuer} : \textit{approval};$
$\texttt{to ATM} : \textit{approval}$

$\texttt{Card Issuer}$ **process**
$\texttt{from Card Issuer} : \textit{validation};$
$\texttt{to Bank} : \textit{approval}$

Distributed Systems | Service Composition

Choreographies • Choral



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

```
class HelloRoles_B {
        public static void sayHello() {
                String b = "Hello from B";
                System.out.println( b );
   }
}
```

Distributed Systems | Service Composition

Choreographies • Choral

```
consumeItems(
  DiChannel@( A, B )< Item@X > ch,
  Iterator@A< Item > it,
  Consumer@B< Item > consumer ) {
  if (it.hasNext()) {
    ch.< Choice >select( Choice@A.GO );
    it.next() >> ch::< Item >com >> consumer::accept;
    consumeItems( ch, it, consumer );
  } else {
    ch.< Choice >select( Choice@A.STOP );
  }
}
```