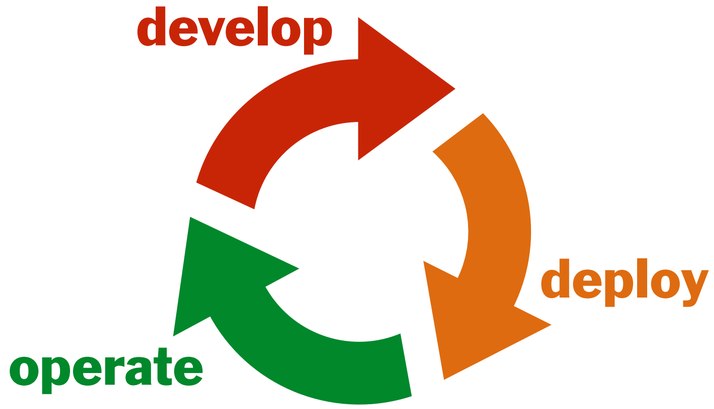
web services

&

micro services

learning objectives



- learn about web services and their underlying technologies
- learn about architectures based on micro services
- learn how those two approaches are related

what is a web service?

a service offered by an electronic device to another electronic device, communicating with each other via the world wide web

wikipedia

rmi* on http

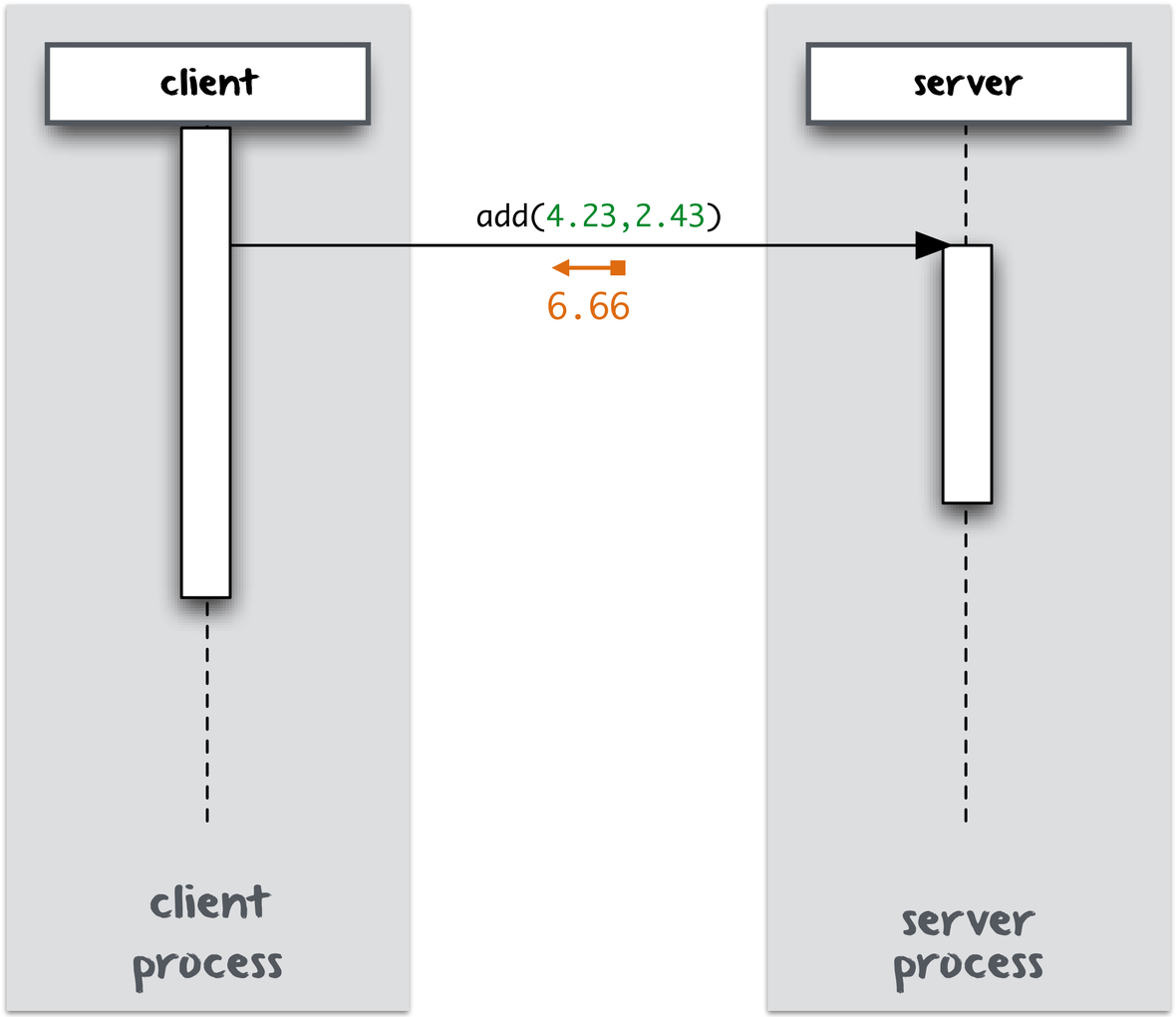
*remote method invocation

software system designed to support interoperable machine-to-machine interaction over a network

world wide web consortium (w3c)

remote method invocation

a reminder



a **remote method** is transparently invoked across the network, **as if it was local**

remote method invocation
a brief history

end 70's Bill Joy introduces the "Berkeley Enhancements", mainly interprocess communication (IPC) facilities. The modern network Unix is born (BSD)

mid 80's Sun Microsystems uses BSD Unix as operating system for their workstations. They extends it with Remote Procedure Calls (RPCs), the ancestor of RMI top of which they build the first Network File System (NFS) and Network Information System (NIS/NIS+).

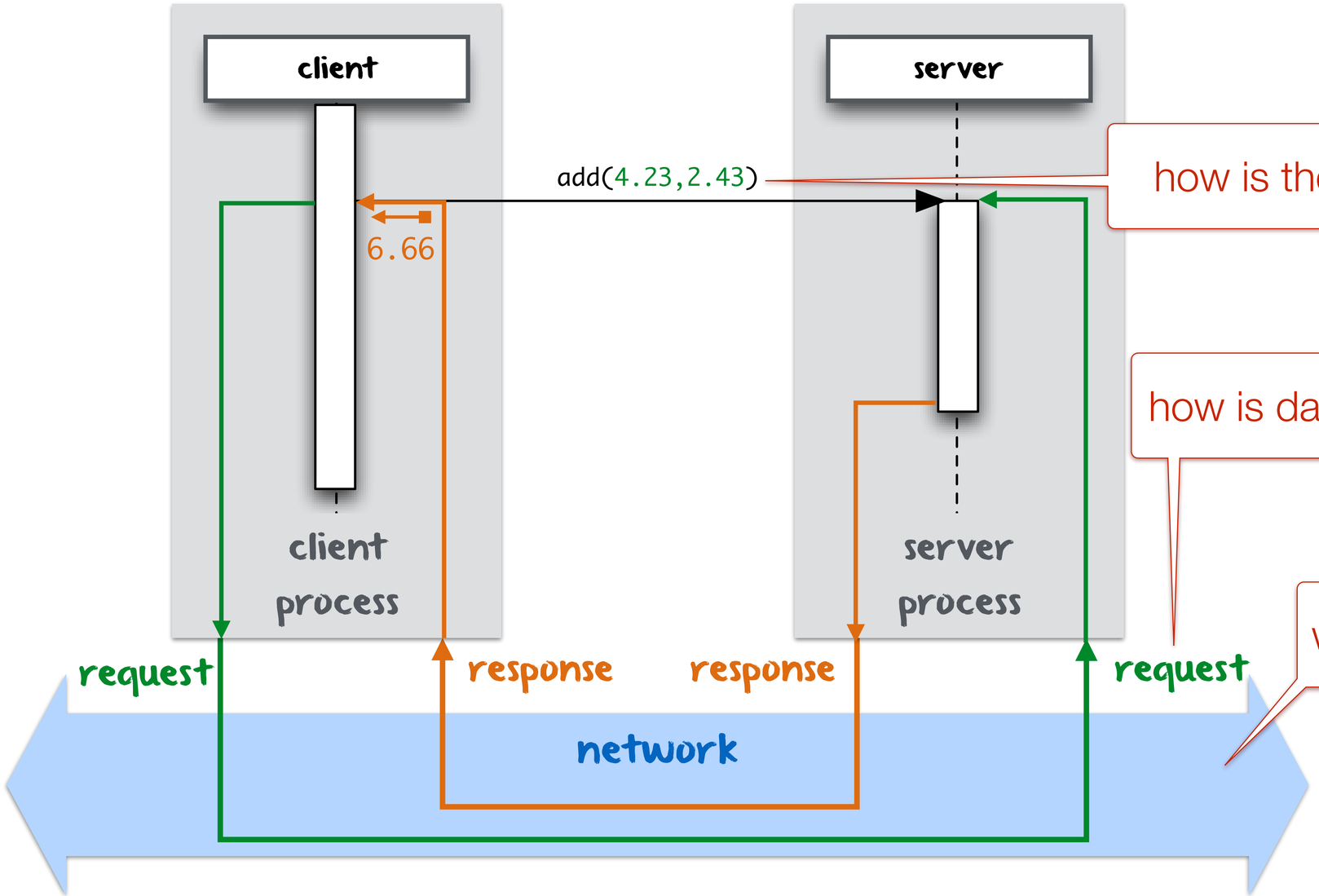
end 80's The Open Software Foundation (OSF) is formed to develop a portable open system platform, known as the Distributed Computing Environment (DCE). The latter proposes DCE RPC as basic communication mechanism

mid 90's The Object Management Group (OMG) follows the same approach to devise the Common Object Request Broker Architecture (CORBA) for object-based middleware. At the same time, Sun greatly simplifies & extends the RMI paradigm its Java & Jini platforms

Today Web Services are a widespread approach to invoke remote services on the web but they are really just a web-flavored version of the good old RPC/RMI paradigm, using HTTP & XML/JSON

remote method invocation

key questions



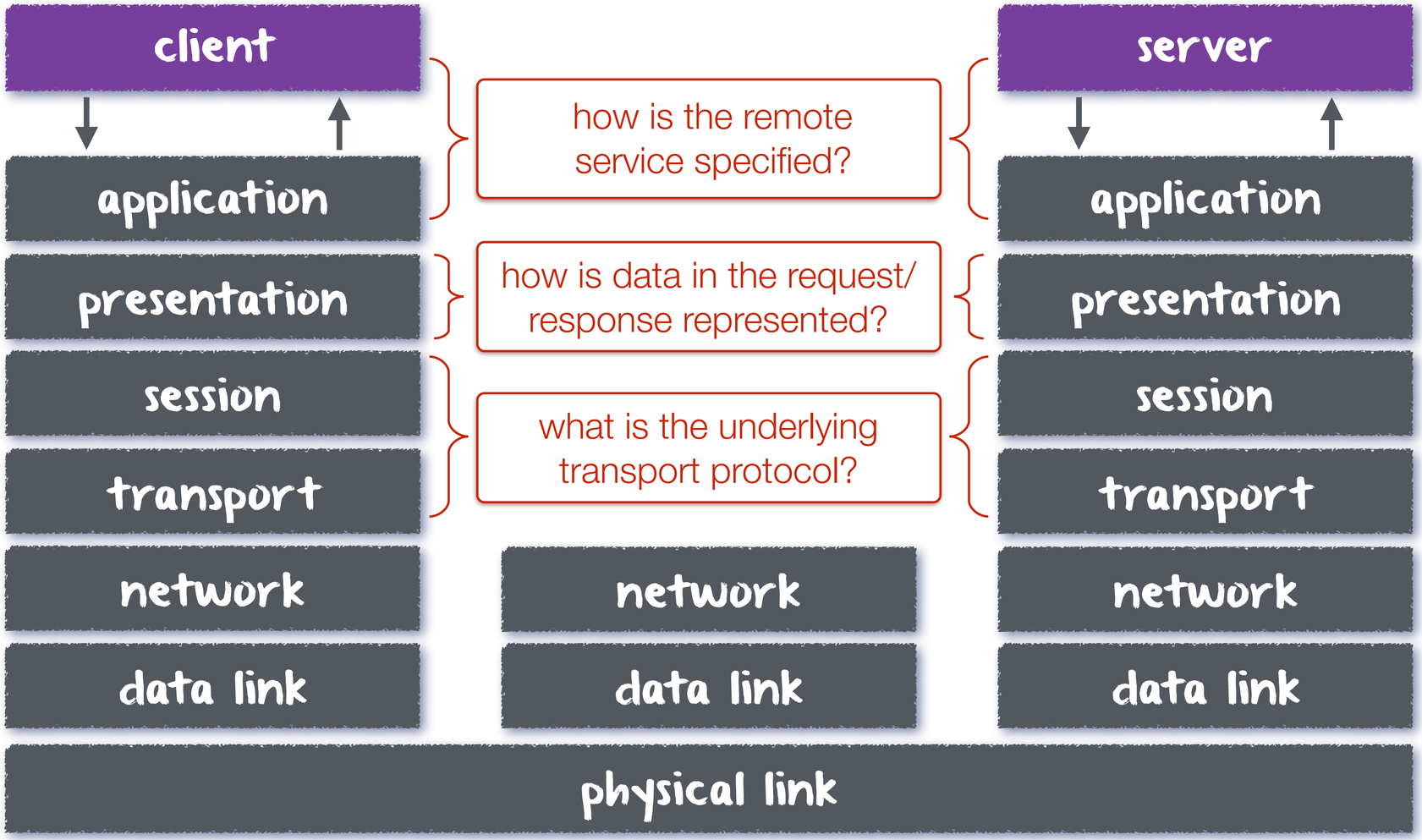
how is the remote service specified?

how is data in the request/response represented?

what is the underlying transport protocol?

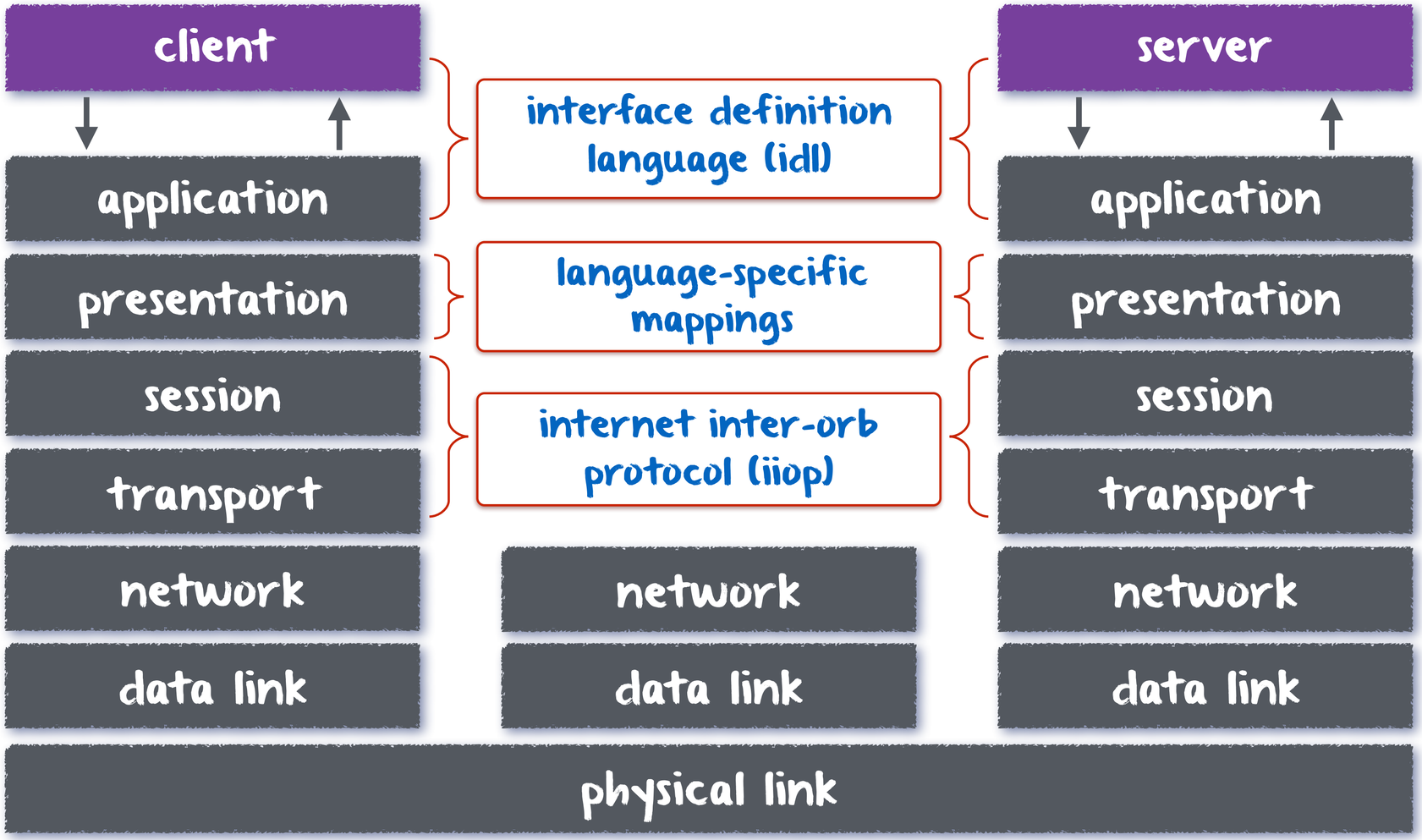
remote method invocation

key questions



remote method invocation

key questions ⇒ corba



remote method invocation

key questions ⇒ corba

```
module employee {
  struct EmployeeInfo {
    wstring name;
    long number;
    double salary;
  };

  exception SQLError {
    wstring message;
  };

  interface Employee {
    EmployeeInfo getEmployee (in wstring name) raises (SQLError);
    EmployeeInfo getEmployeeForUpdate (in wstring name) raises (SQLError);
    void updateEmployee (in EmployeeInfo name) raises (SQLError);
  };
};
```

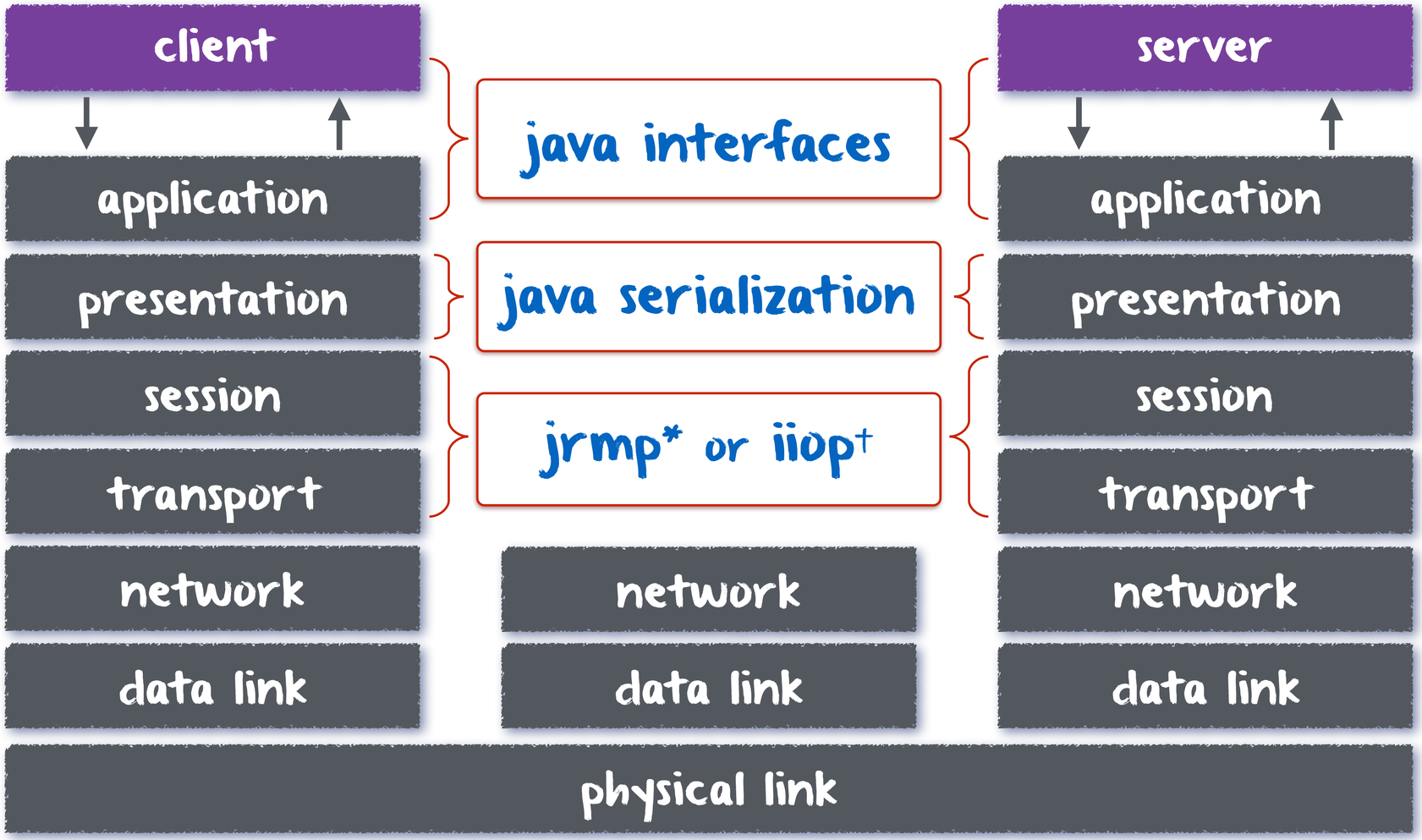
IDL Type	Java Type
module	package
boolean	boolean
char, wchar	char
octet	byte
string, wstring	java.lang.String
short, unsigned short	short
long, unsigned long	int
long long, unsigned long long	long
float	float
double	double
fixed	java.math.BigDecimal
enum, struct, union	class
sequence, array	array
...	...

```
public final class EmployeeInfo implements org.omg.CORBA.portable.IDLEntity
{
  public String name = null;
  public int number = (int)0;
  public double salary = (double)0;

  public EmployeeInfo (String _name, int _number, double _salary)
  {
    name = _name;
    number = _number;
    salary = _salary;
  }
}
```

```
public interface EmployeeOperations
{
  employee.EmployeeInfo getEmployee (String name) throws employee.SQLError;
  employee.EmployeeInfo getEmployeeForUpdate (String name) throws employee.SQLError;
  void updateEmployee (employee.EmployeeInfo name) throws employee.SQLError;
}
```

remote method invocation

key questions ⇒ java rmi



*java remote method protocol

†internet inter-orb protocol

remote method invocation

key questions ⇒ java rmi

```
@Stateful
public class BankBean implements BankRemote {
    @Resource
    SessionContext ctx;

    public void transfer( Account source, Account destination,double amount )
    throws BankingException { ... }

    public void initialize() { ... }
}
```

server

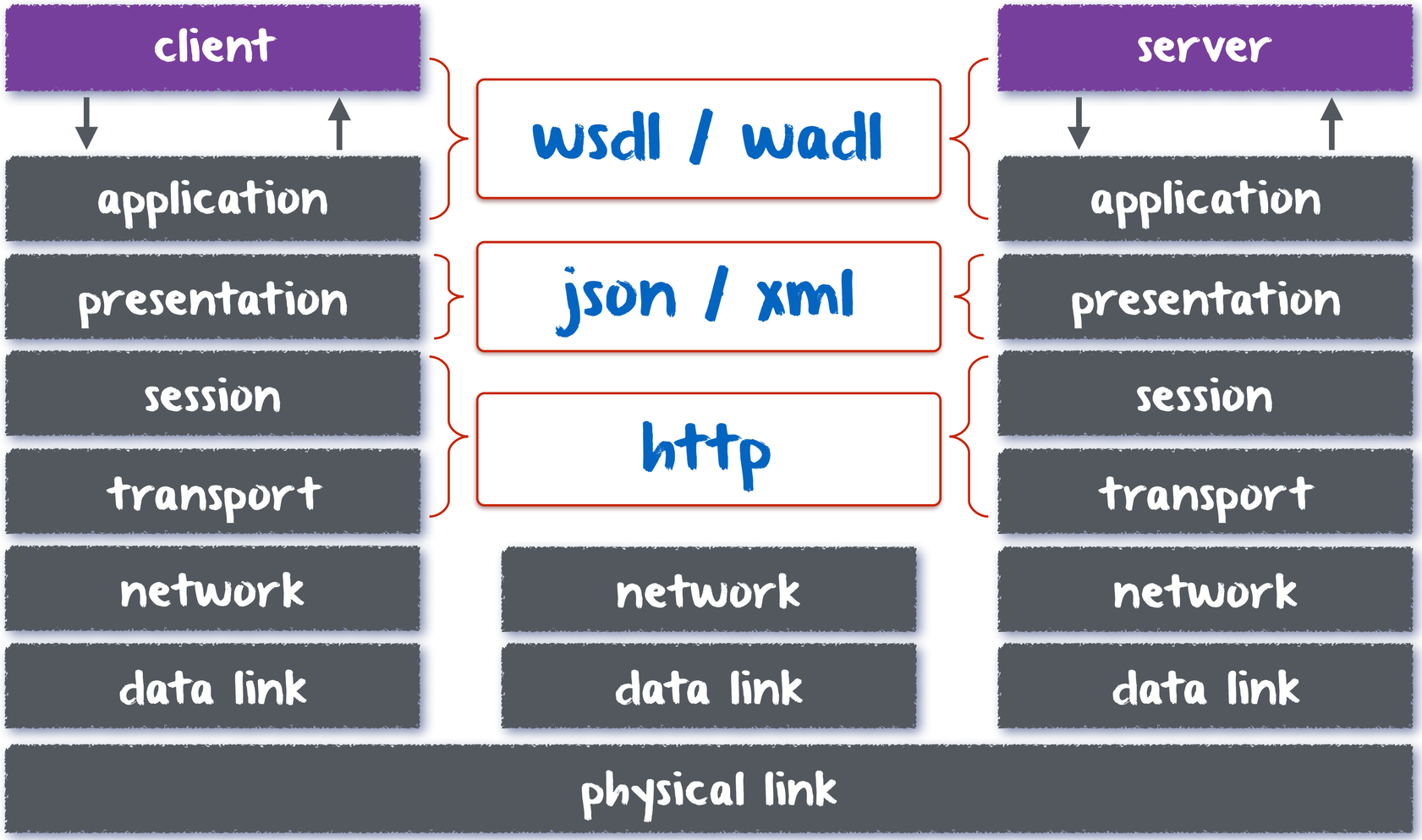
```
@Remote
public interface BankRemote {
    public void transfer( Account source, Account destination,double amount )
    throws BankingException;
    void initialize();
}
```

client

```
@EJB
private static Bank myBank;
…
myBank.initialize();
myBank.transfer(…);
```

remote method invocation

key questions ⇒ web services



web services

the exists essentially two types of web services

general web services aim at remotely invoking any operations

restful web services focus on calls related to state transfer

rest = representational state transfer

Roy Thomas Fielding. 2000. ***Architectural Styles and the Design of Network-Based Software Architectures***. Ph.D. Dissertation. University of California, Irvine.

both types of web services require a significant amount of boilerplate code* to work

* often generated by support tools

general web services

they provide support to remotely call any kind of operations

they rely on the web services description language (wsdl)

they rely on the simple object access protocol (soap), an xml standard defining a message architecture and format

in java, jax-was is the technology that encapsulates (part of) the complexity of defining and using general web services

general web services

server side

```java
@WebService(serviceName = "Calculator")
public class Calculator {

    @WebMethod(operationName = "add")
     public double add(@WebParam(name = "d1") double d1, @WebParam(name = "d2") double d2) {
         double result = d1 + d2;
         System.out.println("-> Calculator:   " + d1 + " + " + d2 + " = " + result);
         return result;
     }

    @WebMethod(operationName = "multiply")
     public double multiply(@WebParam(name = "d1") double d1, @WebParam(name = "d2") double d2) {
         double result = d1 * d2;
         System.out.println("-> Calculator:   " + d1 + " * " + d2 + " = " + result);
         return result;
     }
}
```

soap request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope … />
    <SOAP-ENV:Header/>
    <S:Body xmlns:ns2="http://doplab.unil.ch/">
        <ns2:add>
            <d1>4.23</d1>
            <d2>2.43</d2>
        </ns2:add>
    </S:Body>
</S:Envelope>
```

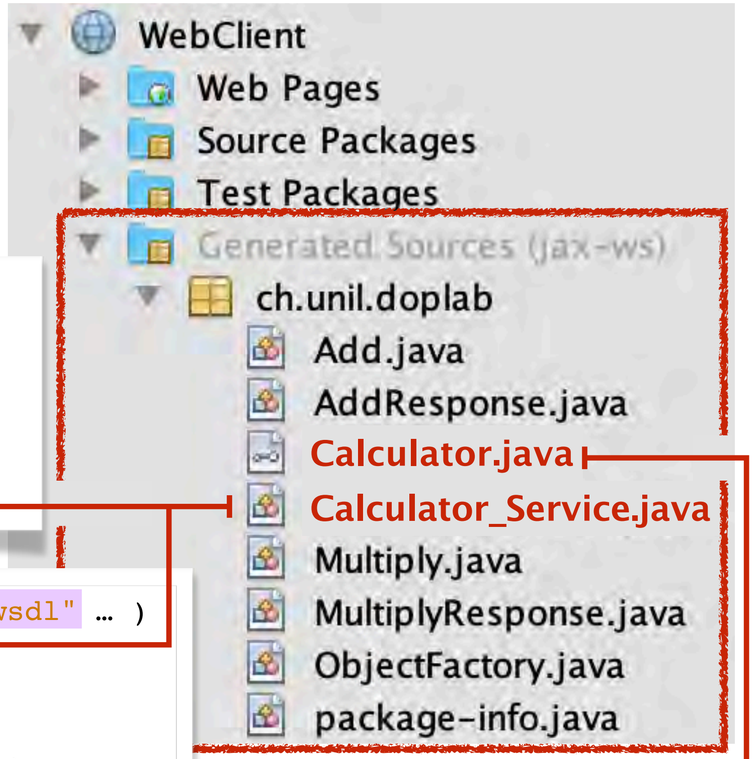
soap response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope … />
    <SOAP-ENV:Header/>
    <S:Body xmlns:ns2="http://doplab.unil.ch/">
        <ns2:addResponse>
            <return>6.66</return>
        </ns2:addResponse>
    </S:Body>
</S:Envelope>
```

general web services

client side

```
@WebServiceRef(wsdlLocation = "http://localhost:8080/WebService/Calculator?wsdl")
private Calculator_Service service;
⋮
Calculator port = service.getCalculatorPort();
double result = port.multiply(d1, d2);;
```




```
@WebServiceClient(name = "Calculator", wsdlLocation = "http://localhost:8080/WebService/Calculator?wsdl" … )
public class Calculator_Service extends Service {

    @WebEndpoint(name = "CalculatorPort")
    public Calculator getCalculatorPort() {
        return super.getPort(new QName("http://doplab.unil.ch/", "CalculatorPort"), Calculator.class);
    }
    ⋮
}
```

```
@WebService(name = "Calculator", targetNamespace = "http://doplab.unil.ch/")
@XmlSeeAlso({ ObjectFactory.class })
public interface Calculator {

  @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "add", targetNamespace = "http://doplab.unil.ch/", className = "ch.unil.doplab.Add")
    @ResponseWrapper(localName = "addResponse", targetNamespace = "http://doplab.unil.ch/", className = "ch.unil.doplab.AddResponse")
    @Action(input = "http://doplab.unil.ch/Calculator/addRequest", output = "http://doplab.unil.ch/Calculator/addResponse")
    public double add(
        @WebParam(name = "d1", targetNamespace = "")
        double d1,
        @WebParam(name = "d2", targetNamespace = "")
        double d2);
    ⋮
}
```

general web services

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasi
    <types>
        <xsd:schema>
            <xsd:import schemaLocation="http://localhost:8080/WebSe
        </xsd:schema>
    </types>
    <message name="add">
        <part name="parameters" element="tns:add"/>
    </message>
    <message name="addResponse">
        <part name="parameters" element="tns:addResponse"/>
    </message>
    <message name="multiply">
        <part name="parameters" element="tns:multiply"/>
    </message>
    <message name="multiplyResponse">
        <part name="parameters" element="tns:multiplyResponse"/>
    </message>
    <portType name="Calculator">
        <operation name="add">
            <input wsam:Action="http://doplab.unil.ch/Calculator/addRequest" message="tns:add"/>
            <output wsam:Action="http://doplab.unil.ch/Calculator/addResponse" message="tns:addResponse"/>
        </operation>
        <operation name="multiply">
            <input wsam:Action="http://doplab.unil.ch/Calculator/multiplyRequest" message="tns:multiply"/>
            <output wsam:Action="http://doplab.unil.ch/Calculator/multiplyResponse" message="tns:multiplyResponse"/>
        </operation>
    </portType>
    ⋮
```

```
    ⋮
    <binding name="CalculatorPortBinding" type="tns:Calculator">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/
        <operation name="add">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="multiply">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="Calculator">
        <port name="CalculatorPort" binding="tns:CalculatorPortBinding">
            <soap:address location="http://localhost:8080/WebService/Calculator"/>
        </port>
    </service>
</definitions>
```

wsdl

restful web services

they focus on state transfer, usually from/to some persistent storage, e.g., a relational database

they manipulate state as resources accessed using uniform resource identifiers (uri) and four http verbs as crud operations

HTTP	CRUD
POST	CREATE
GET	READ
PUT	UPDATE
DELETE	DELETE

Uniform Resource Identifier (URI)	GET	PUT	POST	DELETE
Collection, such as: `http://myservice.com/items`	**get** the URIs and possibly some details of all items	**replace** the entire collection with a new one	**create** a new item in the collection and return its automatically assigned URI	**delete** the entire collection
Item, such as: `http://myservice.com/items/{id}`	**get** the details of item identified by `id`	**replace** the item identified by `id` or **create** it if it does not exist and `id` is valid	**treat** the item identified by `id` as a collection and **create** a element in it (**not often used**)	**delete** the item identified by `id`

restful web services

they require no specific format to represent the data, although json and xml are usually used

they require no specific interface definition language, although it is now common practice to use the web application definition language (wadl) in java

in java, jax-rs is a specification that hides (part of) the complexity of defining and using restful web services and jersey is the reference implementation of jax-rs

restful web services

server side

```
@Entity
@NamedQueries({
        @NamedQuery(name = "findByAccountID",  query = "SELECT a FROM Account a WHERE a.id = :id"),
        @NamedQuery(name = "findByBalance",    query = "SELECT a FROM Account a WHERE a.balance = :balance")})
@XmlRootElement
public class Account implements Serializable {
    @Id
    @Column(name = "ID", nullable = false)
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id = 0L;

    @Column(name = "LASTNAME")
    private String lastName;

    @Column(name = "FIRSTNAME")
    private String firstName;

    @Column(name = "BALANCE")
    private double balance = 0.0;

    public Account(String lastName, String firstName) {
        this.lastName = lastName;
        this.firstName = firstName;
    }
   public double deposit(double amount) {
        balance = balance + amount;
        return balance;
   }
   public double withdraw(double amount) {
        balance = balance - amount;
        return balance;
   }
  public double getBalance() { return balance; }
   public void setBalance(double balance) {
        this.balance = balance;
   }
   ⋮
}
```

```
@Stateless
@Path("ch.unil.doplab.bank.account")
public class AccountFacadeREST extends AbstractFacade<Account> {

    @PersistenceContext(unitName = "BankPU")
    private EntityManager em;

    public AccountFacadeREST() {
        super(Account.class);
    }

    @POST
    @Override
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void create(Account entity) {
        super.create(entity);
    }
    @PUT
    @Path("{id}")
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void edit(@PathParam("id") Long id, Account entity) {
        super.edit(entity);
    }
    @GET
    @Path("{id}")
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Account find(@PathParam("id") Long id) {
        return super.find(id);
    }
    @GET
    @Override
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Account> findAll() {
        return super.findAll();
    }
   ⋮
}
```

restful web services

client side

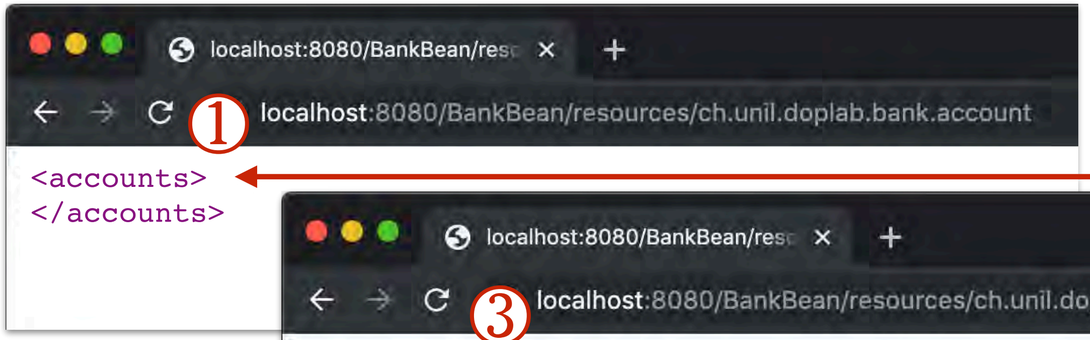
```
@Inject
@RestClient
private AccountPersistenceClient accountPersistenceClient;

Account account = new Account("Simpson","Marge");
Response response = accountPersistenceClient.create(account);
⋮
if (response() == 201) {
  System.out.println("Account was successfully created");

} else {
  System.out.println("Problem creating new account");
}
⋮
```

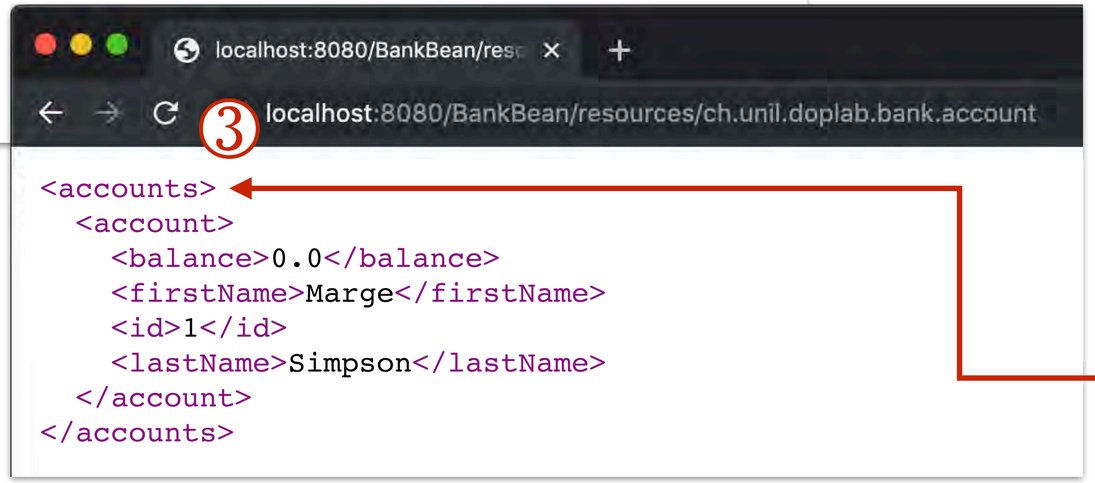
Request

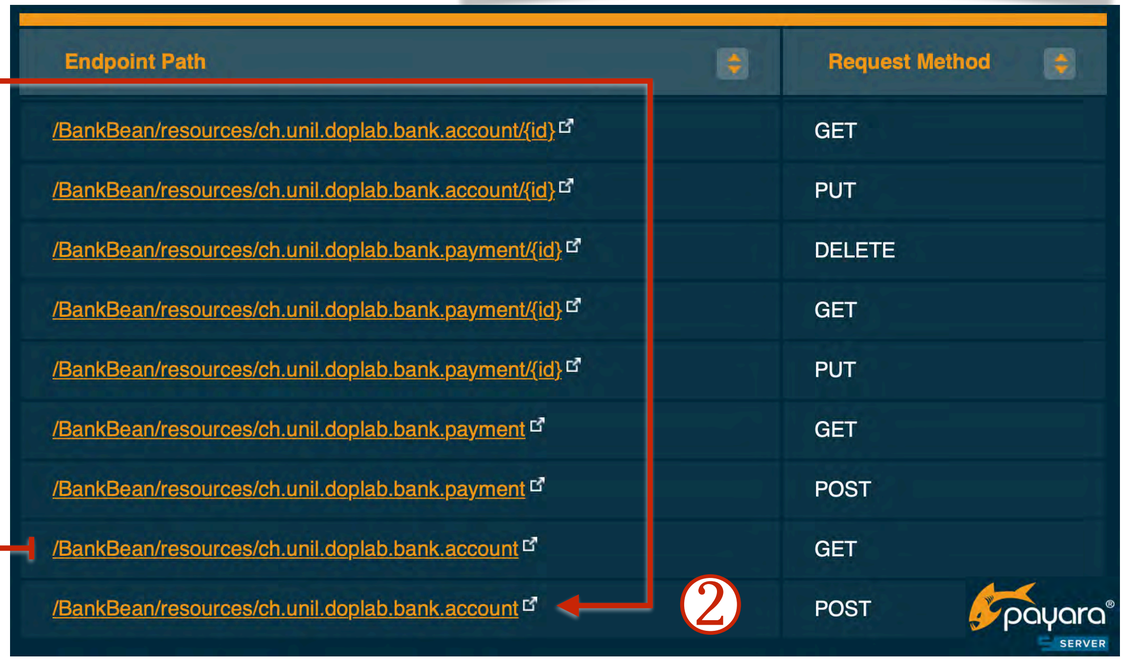
```
GET /BankBean/resources/ch.unil.doplab.bank.account HTTP/1.1
Cookie: JSESSIONID=699c411a2c693029aae28f8b1855
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Host: wallace-palace.local:8080
User-Agent: Mozilla/5.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Content-Length: 183
Content-Type: application/xml
X-Frame-Options: SAMEORIGIN
X-Powered-By: Servlet/4.0 JSP/2.3
Server: Payara Server 5.194 #badassfish
```







restful web services

```
<application xmlns="http://wadl.dev.java.net/2009/02">

    <grammars>
        <include href="application.wadl/xsd0.xsd">
            <doc title="Generated" xml:lang="en"/>
        </include>
    </grammars>
    <resources base="http://wallace-palace.local:8080/BankBean/resources/">
        <resource path="ch.unil.doplab.bank.account">
            <method id="create" name="POST">
                <request>
                    <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="account" mediaType="application/xml"/>
                    <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="account" mediaType="application/json"/>
                </request>
            </method>
            <method id="findAll" name="GET">
                <response>
                    <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="account" mediaType="application/xml"/>
                    <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="account" mediaType="application/json"/>
                </response>
            </method>
            <resource path="{id}">
                <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="id" style="template" type="xs:long"/>
                <method id="edit" name="PUT">
                    <request>
                        <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="account" mediaType="application/xml"/>
                        <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="account" mediaType="application/json"/>
                    </request>
                </method>
            </resource>

        </resource>
    </resources>
</application>
```

wadl

restful web services

data formats

```
@GET
@Override
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<Account> findAll() {
  return super.findAll();
}
```

the xml format
(extensible markup language)

```
<accounts>
  <account>
    <balance>1000000.0</balance>
    <firstName>Marge</firstName>
    <id>1</id>
    <lastName>Simpson</lastName>
  </account>
  <account>
    <balance>100.0</balance>
    <firstName>Bart</firstName>
    <id>2</id>
    <lastName>Simpson</lastName>
  </account>
</accounts>
```

the json format
(javascript object notation)

```
[
    {
        "balance": "1000000.0",
        "firstName": "Marge",
        "id": "1",
        "lastName": "Simpson"
    },
    {
        "balance": "100.0",
        "firstName": "Bart",
        "id": "2",
        "lastName": "Simpson"
    }
]
```

what are microservices?

web services are the cornerstone of the service-oriented architecture (soa), where services are accessed via the network

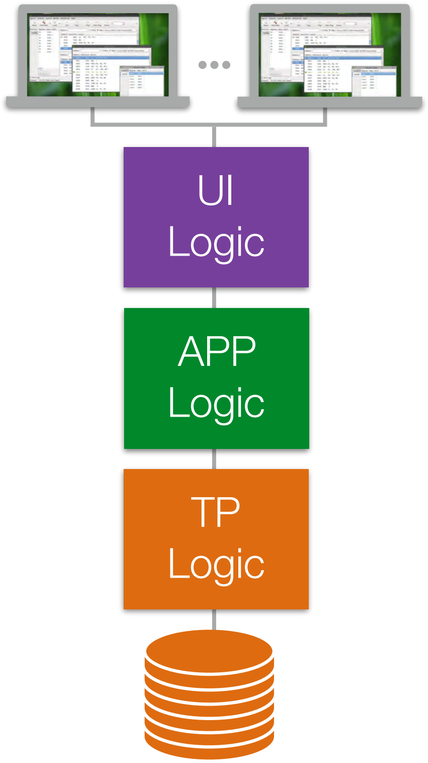
the microservice architecture a variant of soa

in the microservice architecture, an application consists in a collection of loosely coupled and fined-grained services

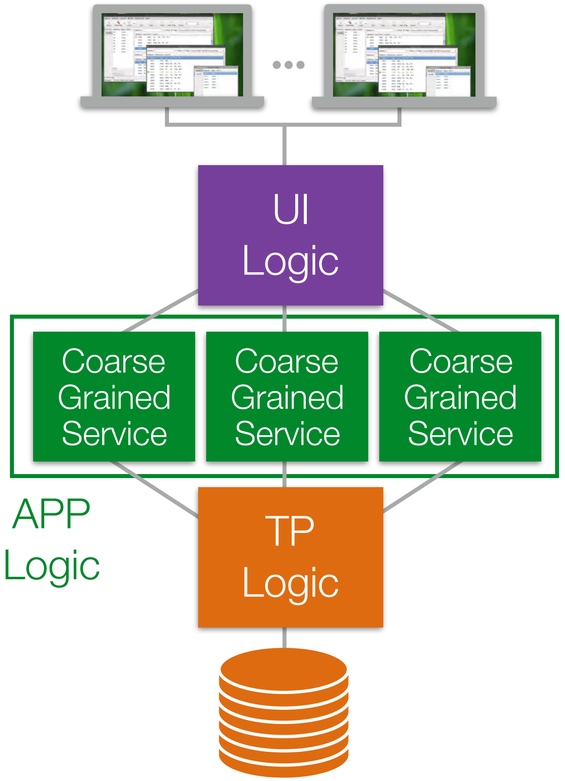
the microservice architecture is usually opposed to the "traditional" monolithic architecture

what are microservices?

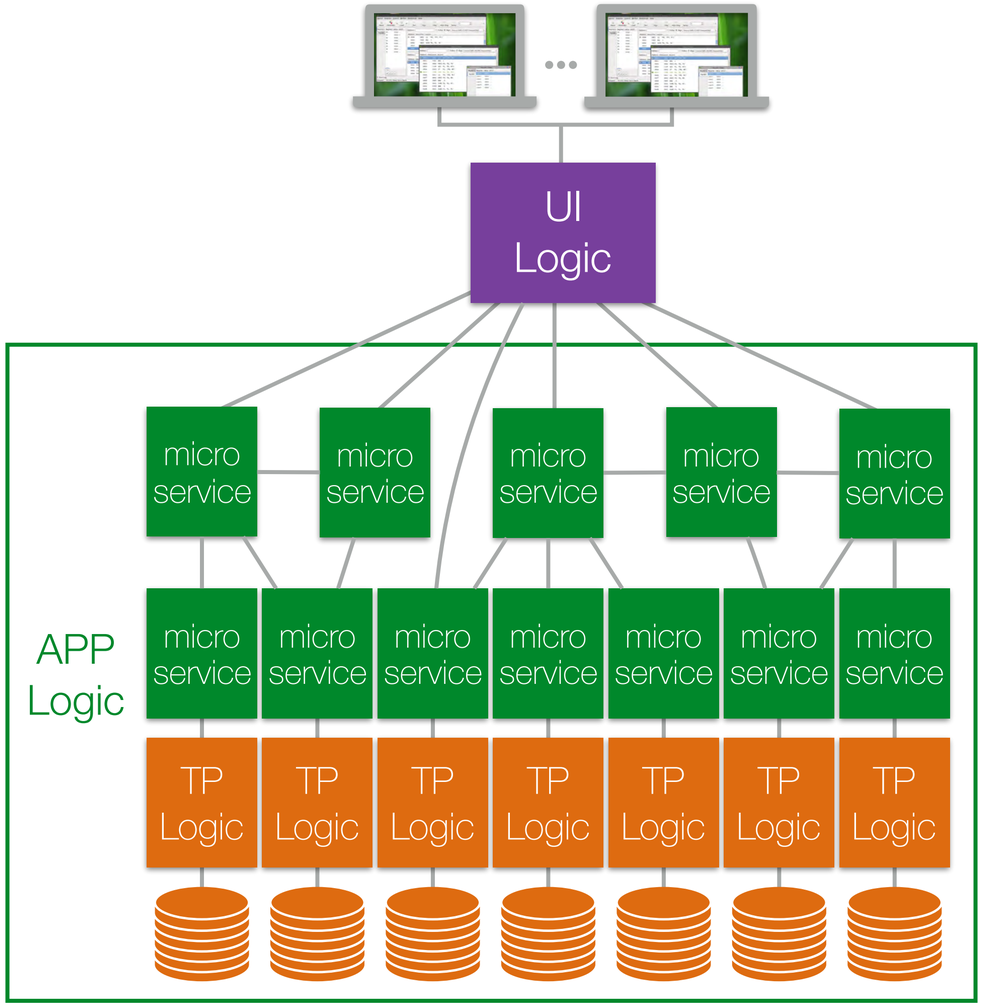
monolithic



server-oriented



microservices

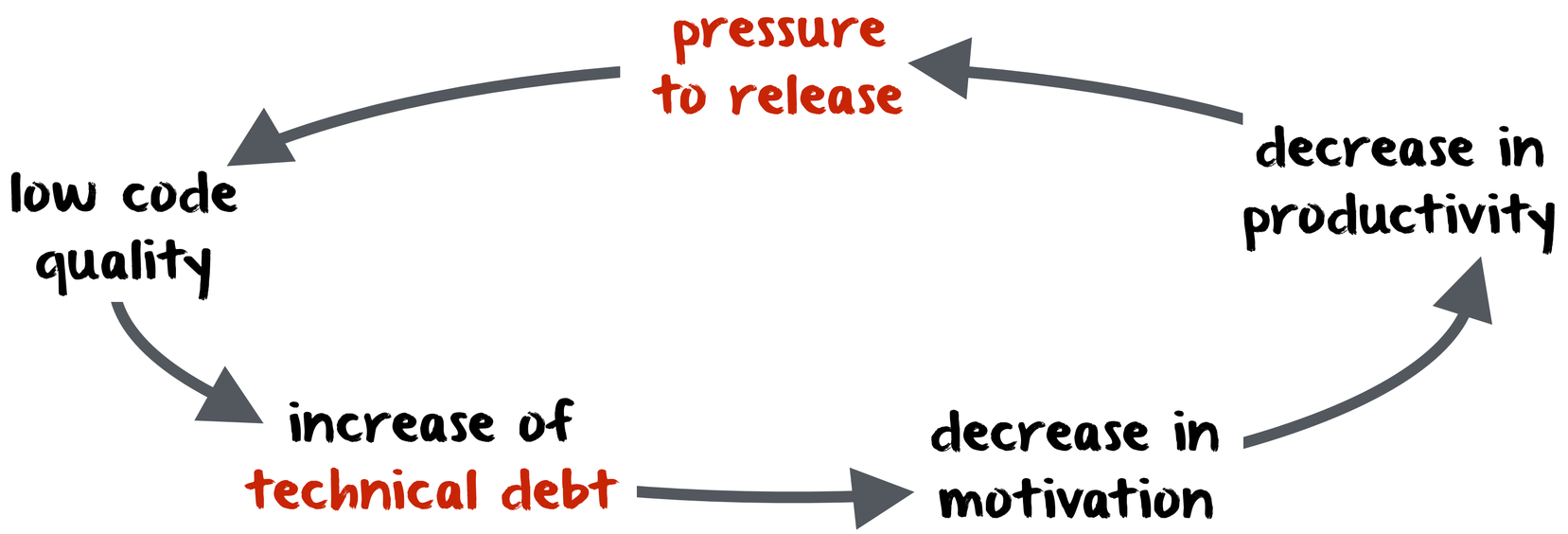


UI = User Interface APP = Application TP = Transaction Processing

why microservices?

two main forces drove the advent of microservices

1. the pressure to release software more often
2. the need to avoid accumulating a technical debt



what is a technical debt ?

in software development, a technical debt captures the cost of additional rework caused by choosing a quick but limited design solution instead of a better one that would take longer

technical debt ⟷ work not done

a technical debt is similar to a monetary debt
if you do not repay it, it accumulates interests, i.e.,
it becomes increasingly harder to implement changes

this can lead to technical bankruptcy

typical forms of a technical debt

recurring bugs

duplicated code

high code complexity

highly coupled code

low code cohesion

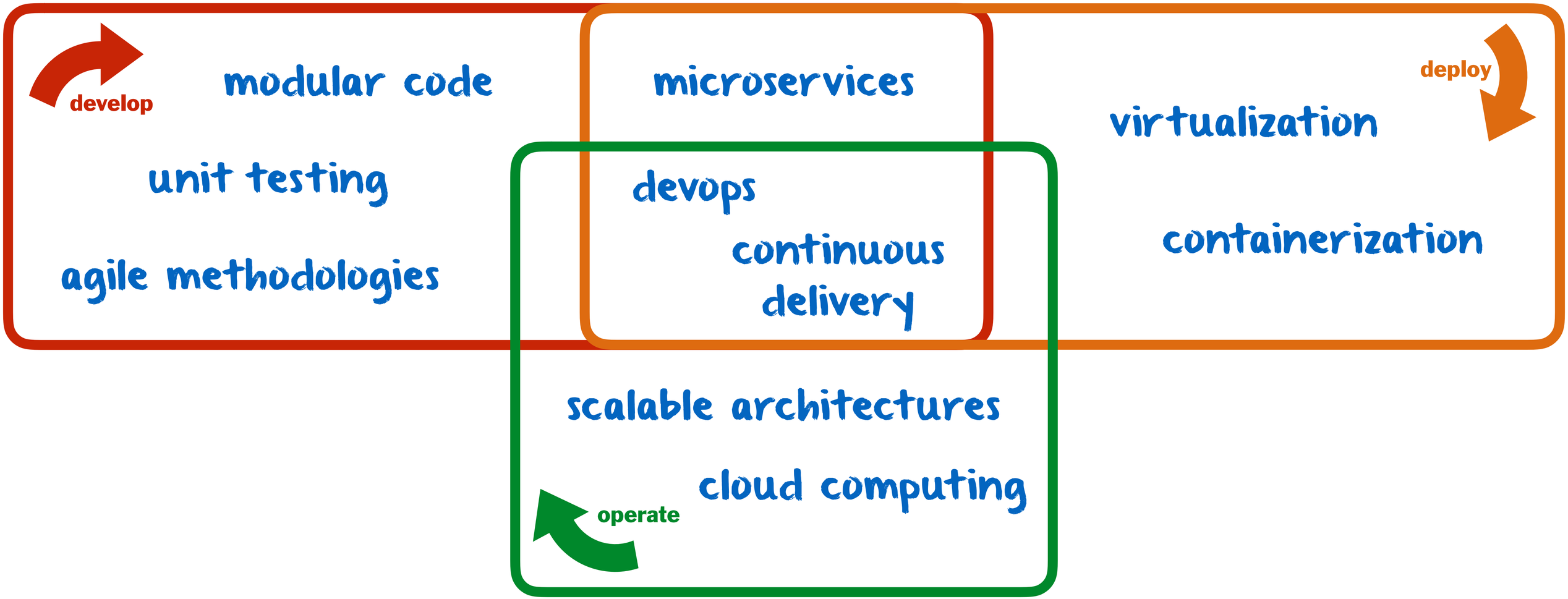
lack of code modularity

monolithic architecture

lack of unit tests

how to avoid a technical debt ?

mitigating measures



modular code vs. microservices*

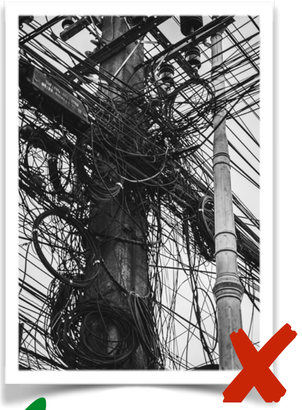
*another way to say very modular services

what they have in common

loose coupling

↳ coupling measures the degree of dependencies between program units

- units that cannot be modified without impacting each other are tightly coupled
- units that can be modified independently of each other are loosely coupled
- we aimed at loose coupling because it makes maintainability easier
- loosely coupled units can evolve independently



high cohesion

↳ cohesion measures the diversity of tasks a program unit is responsible for

- a unit that is responsible of different sets of unrelated tasks shows low cohesion
- a unit that is responsible of one well-defined set of tasks shows high cohesion
- highly cohesive units are easier to name, to understand and to reuse,
  e.g., a class should represent only one well-defined business entity and
  a method should represent one and only one well-defined task

modular code vs. microservices*

*another way to say very modular services

how they differ

program units are classes and objects	program units are networked* services *usually web services
reuse at the code level	reuse at the service level
usually only one programming language	different programming languages possible
local interaction	distributed interaction
vertical scalability	horizontal scalability

these two types of modularity are not incompatible

advantages of microservices

they reduce development complexity by encapsulating different families of application features into distinct services with well-defined interfaces

they help to focus on delivering features to users, rather than on projects

they allow to work with different programming languages

they allow for decentralized governance of the code

they facilitate continuous deployment and delivery

they allow for horizontal scalability of individual services

they force to design for failures and hence increase resilience

drawbacks of microservices

they increase deployment and operation complexity

they make debugging and log analysis very difficult

their performance might suffer from chains of remote interactions

they might be subject to complex semantical interdependencies in case of failures

they are a challenge to global data consistency

they might require complex distributed transactions

drawbacks of microservices

the root of all evil

a distributed system is one that stops you from getting any work done when a machine you've never even heard of crashes.

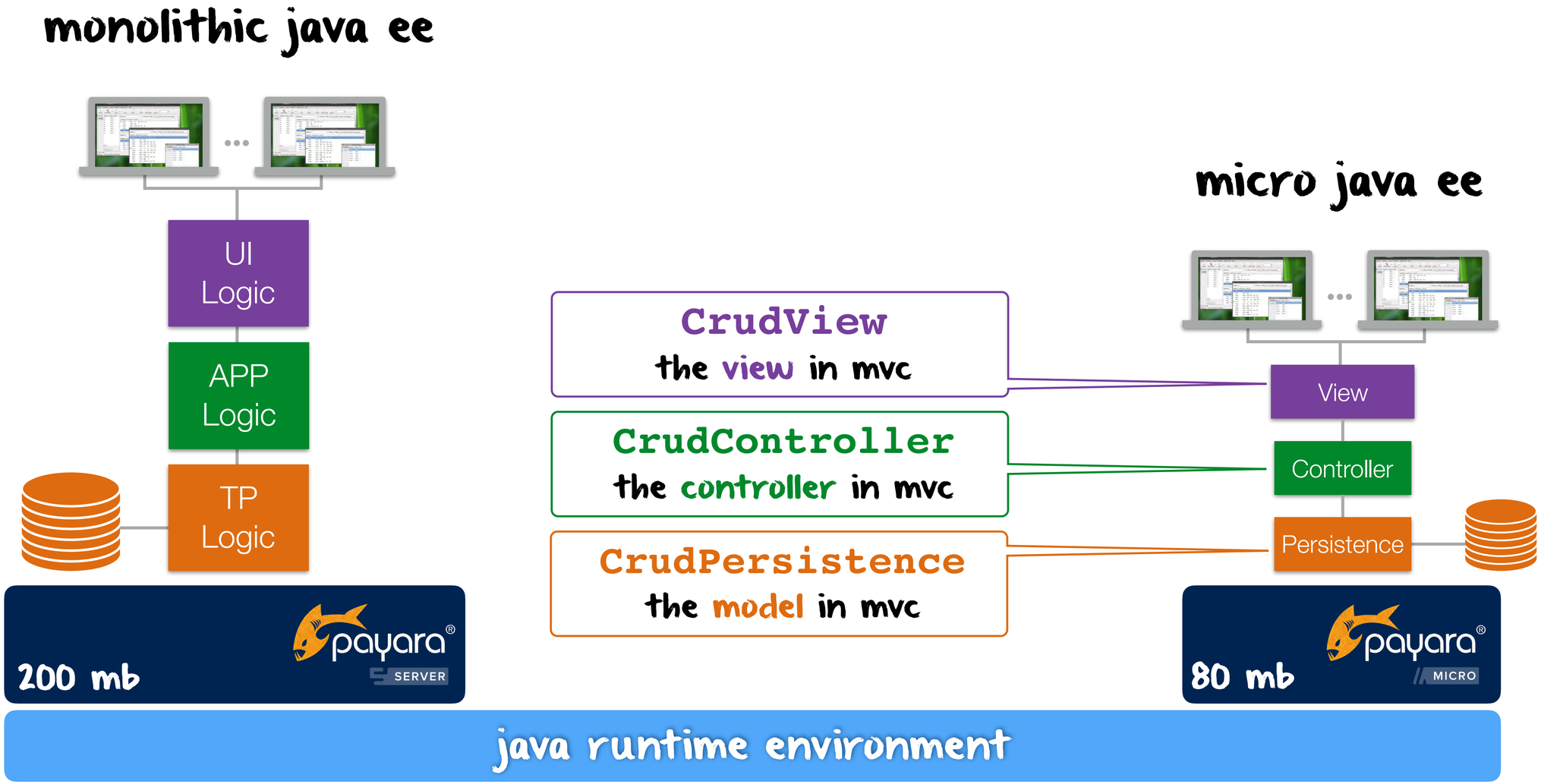
L. Lamport, quoted by S. Müllender in Distributed Systems. 2nd edition. Addison-Wesley, 1993.

they are subject to several distributed computing fallacies:

- the network is reliable and secure
- latency is zero and bandwidth is infinite
- the network is homogeneous and the topology is immutable

microservices with java ee

simple crud example

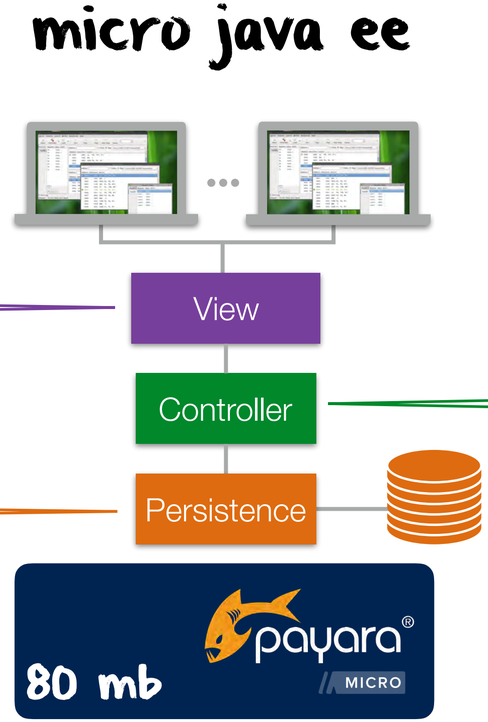


microservices with java ee

simple crud example

1

```
{
   "Instance Configuration": {
      "Host": "myservice.com",
      "Http Port(s)": "8080",
      "Https Port(s)": "",
      "Instance Name": "payara-micro",
      "Instance Group": "no-cluster",
      "Deployed": [
         {
            "Name": "CrudView",
            "Type": "war",
            "Context Root": "/CrudView"
         }
      ]
   }
}
```



2

```
{
   "Instance Configuration": {
      "Host": "myservice.com",
      "Http Port(s)": "8180",
      "Https Port(s)": "",
      "Instance Name": "payara-micro",
      "Instance Group": "no-cluster",
      "Deployed": [
         {
            "Name": "CrudController",
            "Type": "war",
            "Context Root": "/CrudController"
         }
      ]
   }
}
```

3

```
{
   "Instance Configuration": {
      "Host": "myservice.com",
      "Http Port(s)": "8280",
      "Https Port(s)": "",
      "Instance Name": "payara-micro",
      "Instance Group": "no-cluster",
      "Deployed": [
         {
            "Name": "CrudPersistence",
            "Type": "war",
            "Context Root": "/CrudPersistence"
         }
      ]
   }
}
```

1
```
java -jar payara-micro-5.jar --noCluster --port 8080 CrudView.war
```

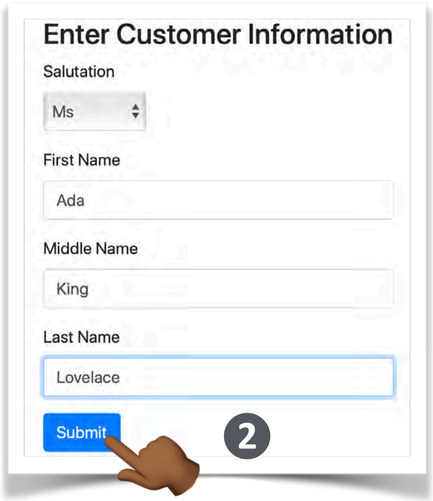
2
```
java -jar payara-micro-5.jar --noCluster --port 8180 CrudController.war
```

3
```
asadmin start-database
```

```
java -jar payara-micro-5.jar --noCluster --port 8280 CrudPersistence.war
```

microservices with java ee

simple crud example



1 initial http request

2 json http request

4 json http request

7 sql request

View

Controller

Persistence

HTML

1

7

CrudView + browser

```
<html lang="en">
    <head>
        <title>Customer Info</title>
  </head>
    <body>
        <div class="container" style="width: 370px !important;">
            <div id="msg">
            </div>
            <h3>Enter Customer Information</h3>
            <form id="customerForm">
                <div class="form-group">
                    <label for="salutation">Salutation</label><br/>
                    <select id="salutation" name="salutation"
                        <option value=""> </option>
                        <option value="Mr">Mr</option>
                        <option value="Mrs">Mrs</option>
                        <option value="Miss">Miss</option>
                        <option value="Ms">Ms</option>
                        <option value="Dr">Dr</option>
                    </select>
                </div>
                <div class="form-group">
                    <label for="firstName">First Name</label>
                    <input type="text" maxlength="10"
                </div>
                <div class="form-group">
                    <label for="middleName">Middle Name</label>
                    <input type="text" maxlength="10"
                </div>
                <div class="form-group">
                    <label for="lastName">Last Name</label>
                    <input type="text" maxlength="20"
                </div>
                <div class="form-group">
                    <button type="button" id="submitBtn"
                </div>
            </form>
        </div>
    </body>
</html>
```

index.html

javascript

2

CrudController

CustomerControllerService

3

CustomerPersistenceClient

4

CrudPersistence

EntityManager

6

CrudDao

5

CustomerPersistenceService

microservices with java ee

simple crud example

1. `java -jar payara-micro-5.jar --noCluster --port 8080 CrudView.war`

```
<html lang="en">
    <head>
      <title>Customer Info</title>
  </head>
    <body>
        <div class="container" style="width: 370px !important;">
            <div id="msg">
            </div>
            <h3>Enter Customer Information</h3>
            <form id="customerForm">
                <div class="form-group">
                    <label for="salutation">Salutation</label><br/>
                    <select id="salutation" name="salutation" class="form-control" style="width: 100px !important;">
                        <option value=""> </option>
                        <option value="Mr">Mr</option>
                        <option value="Mrs">Mrs</option>
                        <option value="Miss">Miss</option>
                        <option value="Ms">Ms</option>
                        <option value="Dr">Dr</option>
                    </select>
                </div>
                <div class="form-group">
                    <label for="firstName">First Name</label>
                    <input type="text" maxlength="10" class="form-control" id="firstName" name="firstName"  placeholder="First Name">
                </div>
                <div class="form-group">
                    <label for="middleName">Middle Name</label>
                    <input type="text" maxlength="10" class="form-control" id="middleName" name="middleName" placeholder="Middle Name">
                </div>
                <div class="form-group">
                    <label for="lastName">Last Name</label>
                    <input type="text" maxlength="20" class="form-control" id="lastName" name="lastName" placeholder="Last Name">
                </div>
                <div class="form-group">
                    <button type="button" id="submitBtn" class="btn btn-primary">Submit</button>
                </div>
            </form>
        </div>
        ⋮
    </body>
</html>
```

java script for communicating with the controller (CustomerControllerService) - see next slide

microservices with java ee

simple crud example

1. `java -jar payara-micro-5.jar --noCluster --port 8080 CrudView.war`

```
<script>
    $(document).ready(function () {
        // click on button submit
        $("#submitBtn").on('click', function () {
            var customerData = $("#customerForm").serializeArray();
            var customerDataJsonObj = objectifyForm(customerData);

            console.log("customerDataJsonObj = " + JSON.stringify(customerDataJsonObj));
            $.ajax({
                headers: {
                    'Content-Type': 'application/json'
                },
                crossDomain: true,
                dataType: "json",
                type: "POST",
                url: "http://localhost:8180/CrudController/webresources/customercontroller/",
                data: JSON.stringify(customerDataJsonObj)
            }).done(function (data, textStatus, jqXHR) {
                if (jqXHR.status === 200) {
                    $("#msg").removeClass();
                    $("#msg").toggleClass("alert alert-success");
                    $("#msg").html("Customer saved successfully.");
                } else {
                    $("#msg").removeClass();
                    $("#msg").toggleClass("alert alert-danger");
                    $("#msg").html("There was an error saving customer data.");
                }
            }).fail(function (data, textStatus, jqXHR) {
                console.log("ajax call failed");
                console.log("data = " + JSON.stringify(data));
                console.log("textStatus = " + textStatus);
                console.log("jqXHR = " + jqXHR);
                console.log("jqXHR.status = " + jqXHR.status);
            });
        });
    });
    function objectifyForm(formArray) {//serialize data function
        var returnJsonObj = {};
        for (var i = 0; i < formArray.length; i++) {
            returnJsonObj[formArray[i]['name']] = formArray[i]['value'];
        }
        return returnJsonObj;
    }
</script>
```

java script for communicating with the controller (CustomerControllerService)

microservices with java ee

❷ `java -jar payara-micro-5.jar --noCluster --port 8180 CrudController.war`

simple crud example

this is not an ejb but a simple java object that exists only during the request

```
@Path("/customercontroller")
@RequestScoped
public class CustomerControllerService {
    @Inject
    @RestClient
    private CustomerPersistenceClient customerPersistenceClient;
    private static final Logger LOG = Logger.getLogger(CustomerControllerService.class.getName());

    @OPTIONS
    public Response options() {
        LOG.log(Level.INFO, "CustomerControllerService.options() invoked");
        return Response.ok("")
                .header("Access-Control-Allow-Origin", "http://localhost:8080")
                .header("Access-Control-Allow-Headers", "origin, content-type, accept, authorization")
                .header("Access-Control-Allow-Credentials", "true")
                .header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE, OPTIONS, HEAD")
                .header("Access-Control-Max-Age", "1209600")
                .build();
    }
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Operation(summary = "Add a customer to the database", description = "Add a customer to the database based on the JSON representation of the customer")
    public Response addCustomer(Customer customer) throws URISyntaxException {
        Response response = null;
        Response persistenceServiceResponse;
        try {
            persistenceServiceResponse = customerPersistenceClient.create(customer);
            if (persistenceServiceResponse.getStatus() == 201) {
                response = Response.ok("{}").header("Access-Control-Allow-Origin", "http://localhost:8080").build();
            } else {
                response = Response.serverError().header("Access-Control-Allow-Origin", "http://localhost:8080").build();
            }
        } catch (Exception e) {
            LOG.log(Level.SEVERE, "Exception while processing request", e);
            response = Response.serverError().header("Access-Control-Allow-Origin", "http://localhost:8080").build();
        }
       return response;
    }
}
```

microservices with java ee

simple crud example

❷
```
java -jar payara-micro-5.jar --noCluster --port 8180 CrudController.war
```

```
public class Customer {
    private String salutation;
    private String firstName;
    private String middleName;
    private String lastName;

    public String getSalutation() {
        return salutation;
    }
    public void setSalutation(String salutation) {
        this.salutation = salutation;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getMiddleName() {
        return middleName;
    }
    public void setMiddleName(String middleName) {
        this.middleName = middleName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

data transfer object (dto) automatically transformed into a json object when invoking the persistence service

```
@Path("/webresources/customerpersistence")
@RegisterRestClient
public interface CustomerPersistenceClient {

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public Response create(Customer customer);

}
```

microservices with java ee

❸ `java -jar payara-micro-5.jar --noCluster --port 8280 CrudPersistence.war`

simple crud example

```java
@Entity
public class Customer implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String salutation;

    @Column(name = "FIRST_NAME")
    private String firstName;

    @Column(name = "MIDDLE_NAME")
    private String middleName;

    @Column(name = "LAST_NAME")
    private String lastName;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getSalutation() { return salutation; }
    public void setSalutation(String salutation) { this.salutation = salutation; }
    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }
    public String getMiddleName() { return middleName; }
    public void setMiddleName(String middleName) { this.middleName = middleName; }
    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }
   public boolean equals(Object object) {
       if (!(object instanceof Customer)) {
            return false;
        }
        Customer other = (Customer) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
}
```

```java
@ApplicationScoped
@Path("customerpersistence")
public class CustomerPersistenceService {
    @Context
    private UriInfo uriInfo;

    @Inject
    private CrudDao customerDao;

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public Response create(Customer customer) {
        try {
            customerDao.create(customer);
        } catch (Exception e) {
            return Response.serverError().build();
        }
        return Response.created(uriInfo.getAbsolutePath()).build();
    }
}
```

```java
@ApplicationScoped
@Transactional
public class CrudDao {
    @PersistenceContext(unitName = "CustomerPersistenceUnit")
    private EntityManager em;

    public void create(Customer customer) {
        em.persist(customer);
    }
    public boolean checkDatabaseConnection() {
        boolean result;
        try {
            Connection connection = em.unwrap(Connection.class);
            result = connection.isValid(0);
        } catch (Throwable e) {
            result = false;
        }
        return result;
    }
}
```