# Web Services

SOFTENG 325 – Software Architecture

Andrew Meads

## Last time, in SE325…

- Java RMI
  - Aims to make remote method invocation as simple as invoking a method on a local object
  - Provides object-invocation programming abstraction but cannot completely mask a distributed environment
  - Parameter passing and invocation semantics are different for remote invocations, object discovery is required in a distributed environment, and Remote interfaces are necessitated for remotely accessible objects
  - Only useful when all participants in the distributed system are written in Java

## Agenda

- Service-oriented architecture
- HTTP
- Servlets and servlet containers
- Web services
  - SOAP and REST

SOFTENG 325 Lecture 03 – Web Services

## Service-Oriented Architecture

## Service-Oriented Architecture

Service-oriented architectures (SOA) are distributed systems made up of software units (services). With SOA, consumers can discover and interact with services, without regard for the technologies used to implement individual services. SOA applications often cross organisational boundaries.

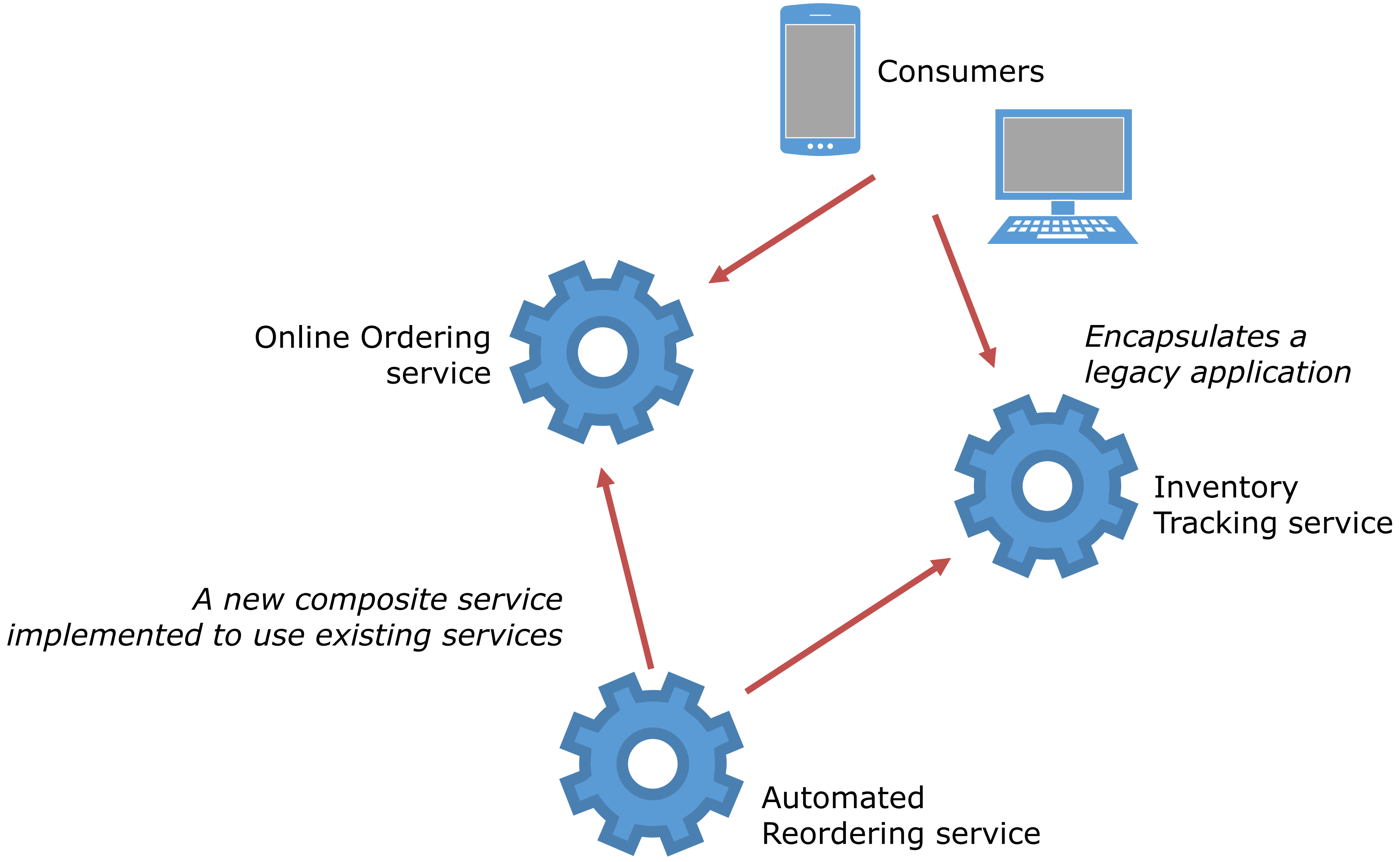
### Service characteristics

- Distributed
- Coarse-grained units of reuse
- Well-defined interfaces, hidden implementations
- Technology independent
- Loosely coupled
- Discoverable
- Composable

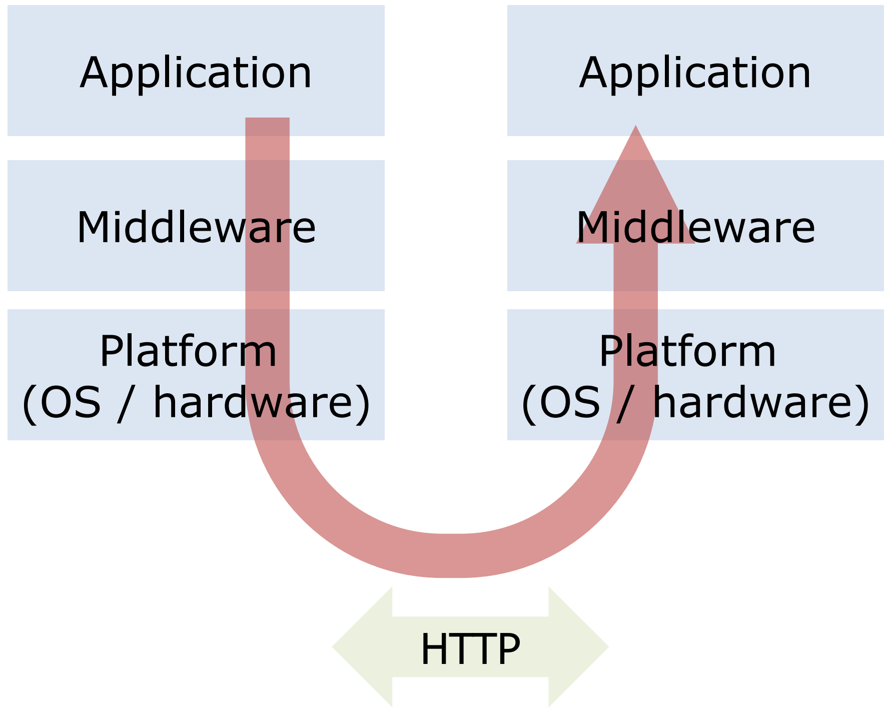
### Why use SOA?

- Reduce IT costs
- Reduce time to market
- Agility
- Leverage legacy systems

# Service-Oriented Architecture


Consumers
Online Ordering service
*Encapsulates a legacy application*
Inventory Tracking service
*A new composite service implemented to use existing services*
Automated Reordering service

## Interoperability in the presence of heterogeneity



- Interoperability necessitates use of a common communication protocol
- HTTP is an **open** and **standardised** protocol
- HTTP is a text-based protocol
  - The character content of HTTP messages can be encoded in an agreed way, e.g. UTF-8 in practice
  - Middleware converts UTF-8 encoded data to and from native formats
- Web services use HTTP

## HTTP

- HTTP defines two messages
  - Request

| *method* | *URL or pathname* | *HTTP version* | *headers* | *message body* |
|---|---|---|---|---|
| GET | //http://www.bbc.com/news | HTTP/ 1.1 | | |

  - Reply

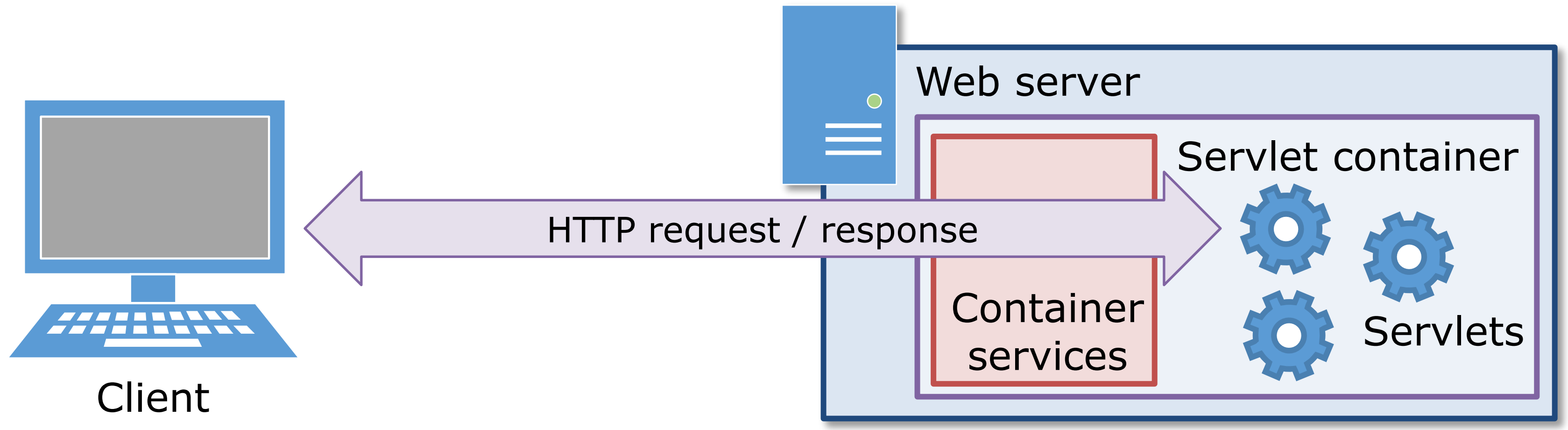
| *HTTP version* | *status* | *reason* | *headers* | *message body* |
|---|---|---|---|---|
| HTTP/1.1 | 200 | OK | | |

- URL syntax
  - http:// serverName [:port] [/pathName] [?query]

SOFTENG 325 Lecture 03 – Web Services

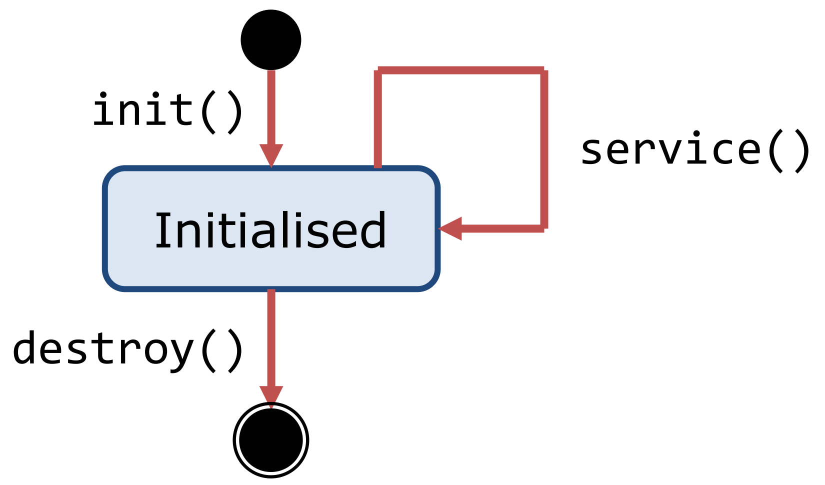
## Servlets & Servlet Containers

## Servlet containers

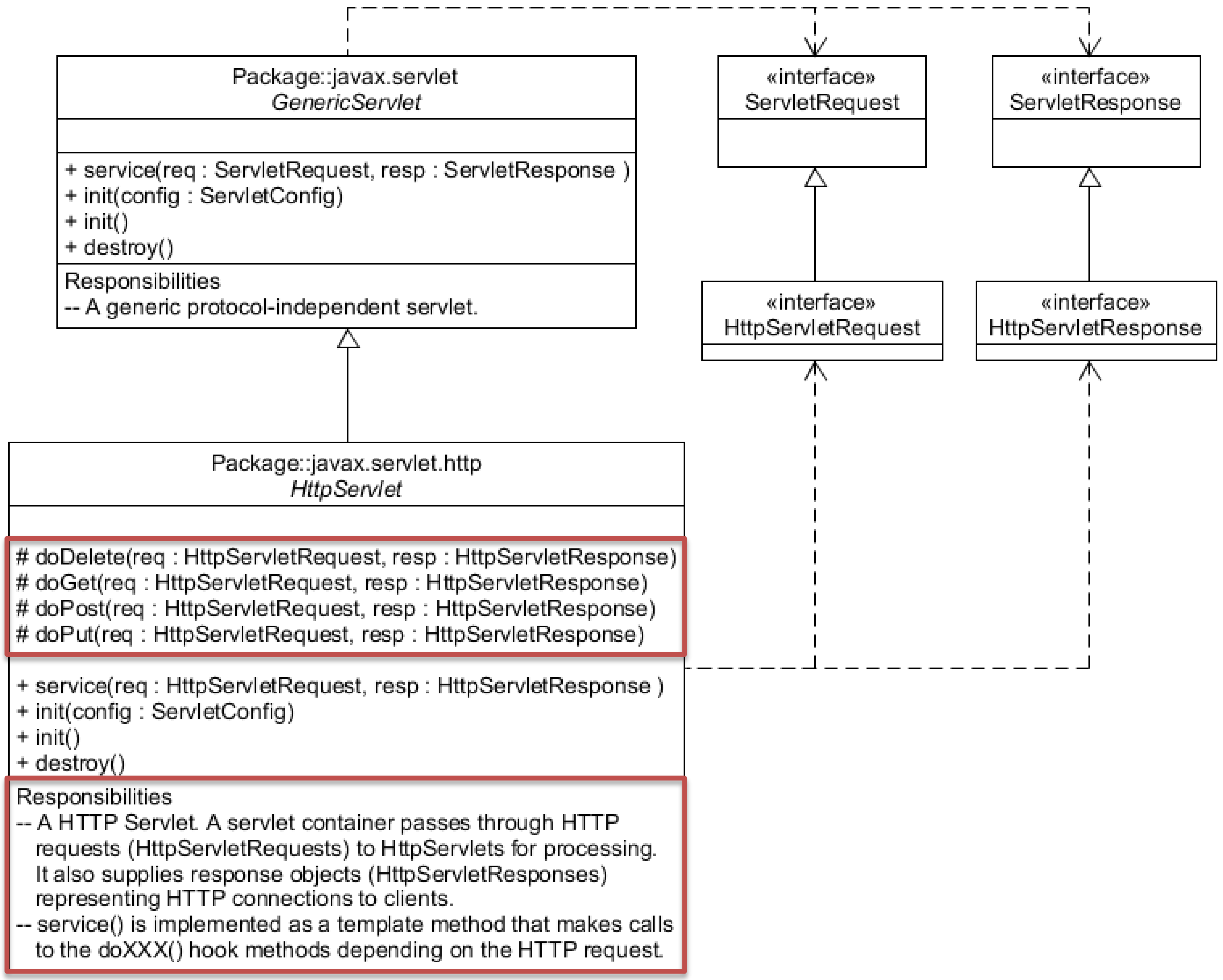


- The servlet container specification provides a managed execution environment for servlets
  - A servlet is a Java component that extends the capabilities of a server
  - A servlet has a lifecycle that is controlled by the container
- Servlet containers route requests through to particular servlets
  - Each incoming request is managed by a separate thread
  - There is at most one instance of any servlet class – servlets need to be threadsafe
- Servlet containers are an example of middleware

## Servlet lifecycle



- The servlet container calls the lifecycle methods
  - `init()`
    - Initialises a servlet instance
  - `service()`
    - Called once per incoming request for the servlet
    - Supplies request data and a connection to the client
  - `destroy()`
    - Called when the servlet container is shutting down or where resources need to be freed
    - Typically implemented to save state to persistent storage


Package::javax.servlet
*GenericServlet*
+ service(req : ServletRequest, resp : ServletResponse )
+ init(config : ServletConfig)
+ init()
+ destroy()
Responsibilities
-- A generic protocol-independent servlet.
«interface»
ServletRequest
«interface»
ServletResponse
«interface»
HttpServletRequest
«interface»
HttpServletResponse
Package::javax.servlet.http
*HttpServlet*
# doDelete(req : HttpServletRequest, resp : HttpServletResponse)
# doGet(req : HttpServletRequest, resp : HttpServletResponse)
# doPost(req : HttpServletRequest, resp : HttpServletResponse)
# doPut(req : HttpServletRequest, resp : HttpServletResponse)
+ service(req : HttpServletRequest, resp : HttpServletResponse )
+ init(config : ServletConfig)
+ init()
+ destroy()
Responsibilities
-- A HTTP Servlet. A servlet container passes through HTTP requests (HttpServletRequests) to HttpServlets for processing. It also supplies response objects (HttpServletResponses) representing HTTP connections to clients.
-- service() is implemented as a template method that makes calls to the doXXX() hook methods depending on the HTTP request.

## HttpServlet

HelloWorldServlet.java

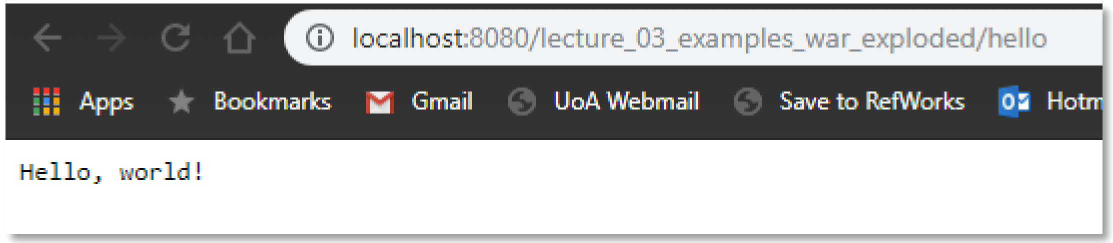
```java
public class HelloWorldServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
          throws ServletException, IOException {

        resp.setContentType("text/plain");

        PrintWriter out = resp.getWriter();
        out.println("Hello, world!");

    }
}
```

web.xml (config file)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app …>

    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>se325.lecture03.servlets.HelloWorldServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

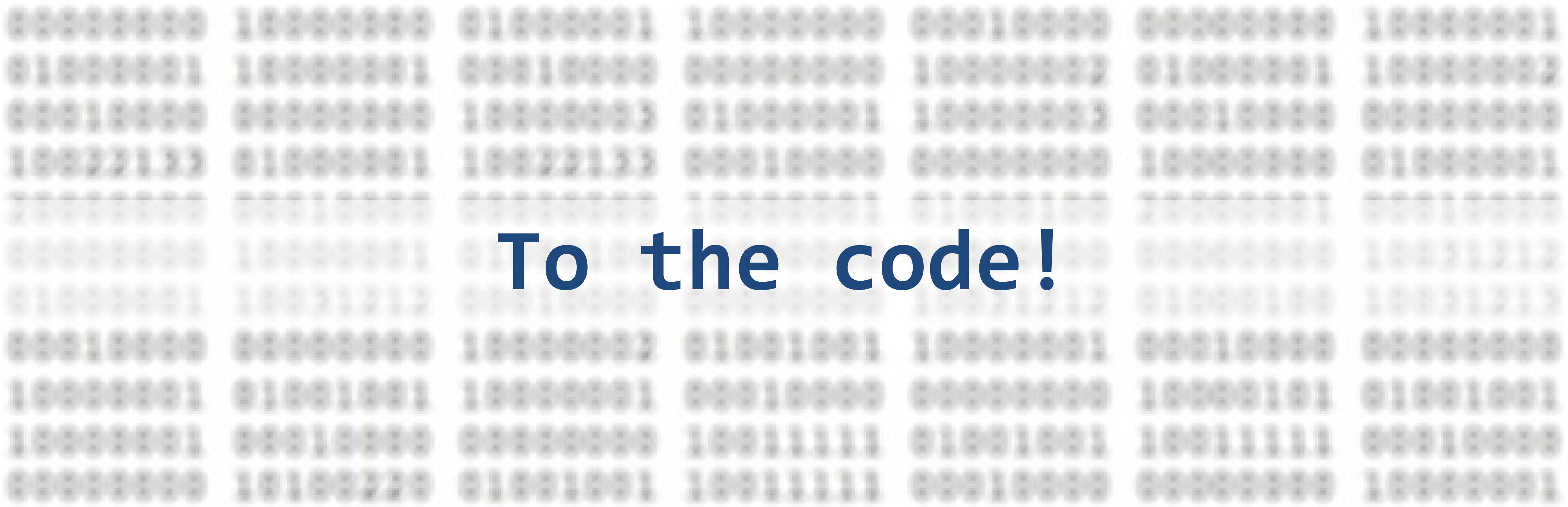
## HTTP Clients

- Any software implementing the HTTP protocol can act as a client – including your web browser!
- We can write HTTP clients in Java too, using the `Client` class – part of JAX-RS (more on this next lecture!)



```
Hello, world!
```

```
Client httpClient = ClientBuilder.newClient();
Response response = httpClient.target("http://.../hello").request().get();

System.out.println("Status: " + response.getStatus());

String message = response.readEntity(String.class);
System.out.println("Message: " + message);

httpClient.close();
```



## To the code!

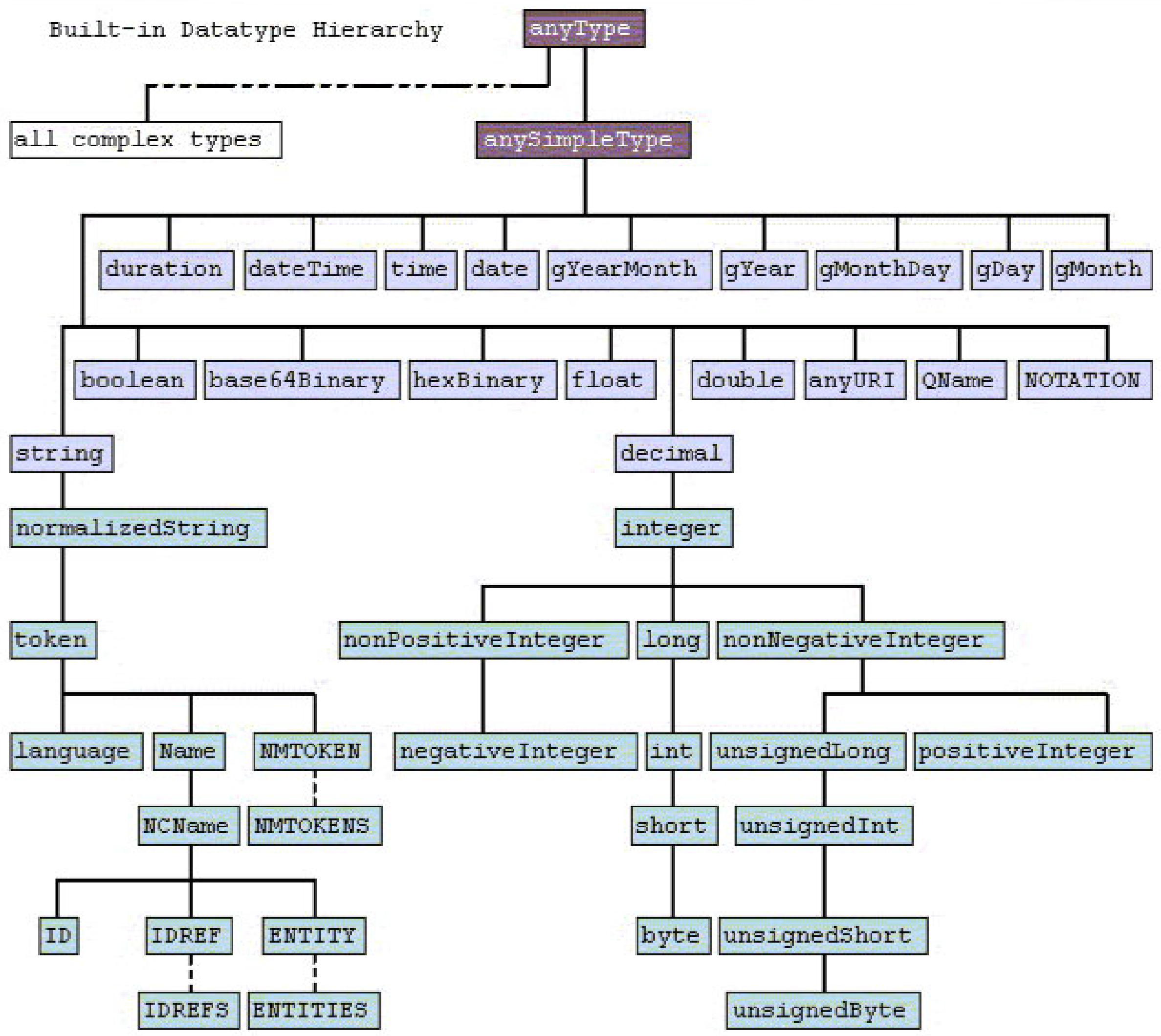
SOFTENG 325 Lecture 03 – Web Services

## SOAP

## Web services

- HTTP is useful in developing an RMI-like service-invocation mechanism that isn’t constrained to Java – but alone it’s insufficient
  - How do we represent service interfaces in a programming language-neutral way?
  - How do we describe service invocations?
  - How do we deal with different datatype systems?
- One method: SOAP over HTTP

W3C Recommendation



# Programming-language-neutral interfaces

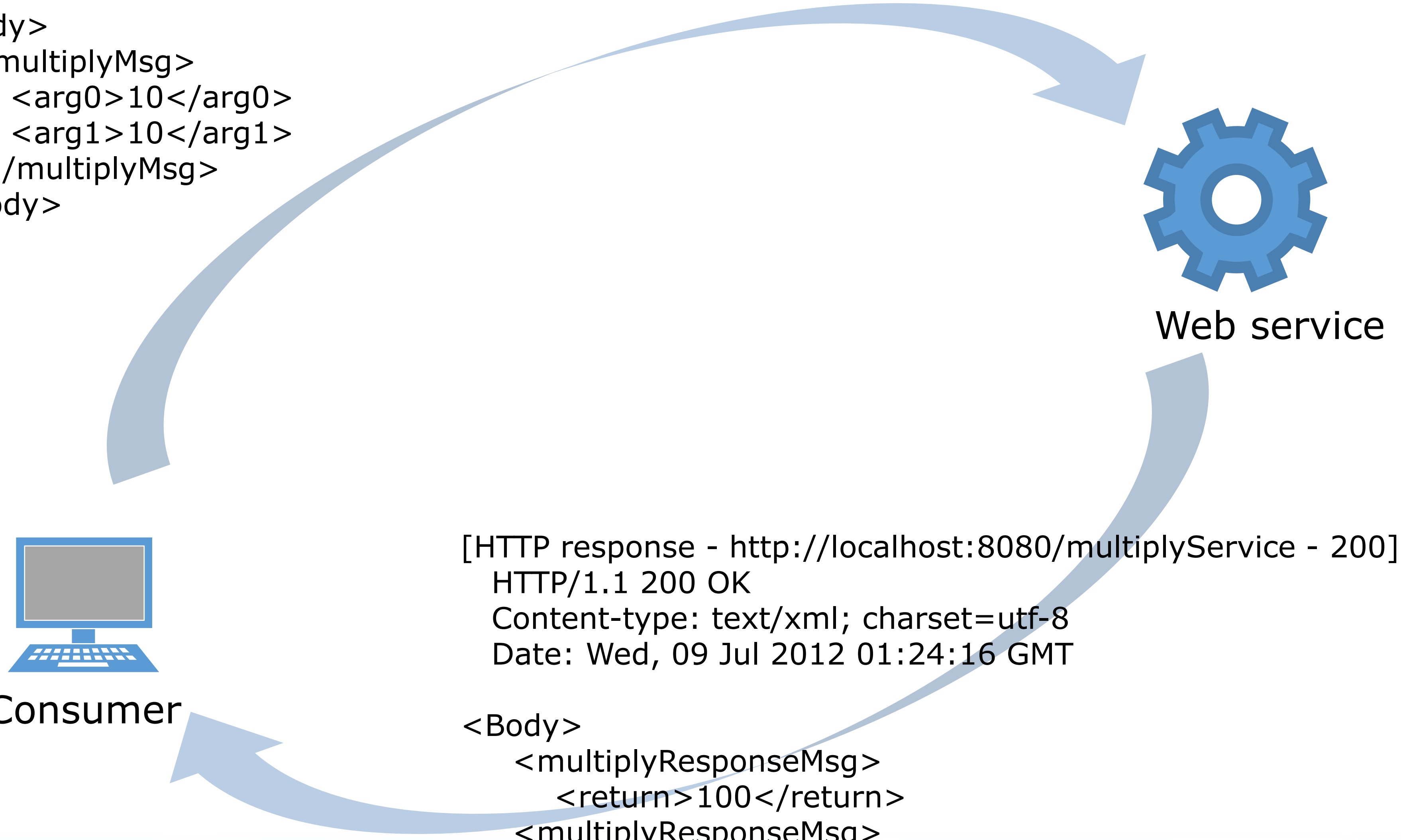
```
<portType name="MultiplyService">
    <operation name="multiply">
        <input message="multiplyMsg"/>
        <output message="multiplyResponseMsg"/>
    </operation>
</portType>

<message name="multiplyMsg"/>
<message name="multiplyResponseMsg"/>

<types>
    <schema>
        <element name="multiplyMsg" type="multiplyType"/>
        <element name="multiplyResponseMsg"
           type="multiplyResponseType"/>
        <complexType name="multiplyType">
            <sequence>
                <element name="arg0" type="int"/>
                <element name="arg1" type="int"/>
            </sequence>
        </complexType>
        <complexType name="multiplyResponseType">
            <sequence>
                <element name="return" type="int"/>
            </sequence>
        </complexType>
    </schema>
</types>
```

*(this is a snippet – not the whole WSDL!)*

- WSDL (Web Service Description Language) is an XML dialect used to describe Web service interfaces
- A WSDL document include several elements:
  - PortType
    - A set of named operations (like an interface)
    - Each operation is described by an input and output message
  - Message
    - Typed messages
  - Types
    - Datatype definitions
  - Binding/Service
    - Communication endpoints identifying the location of a service

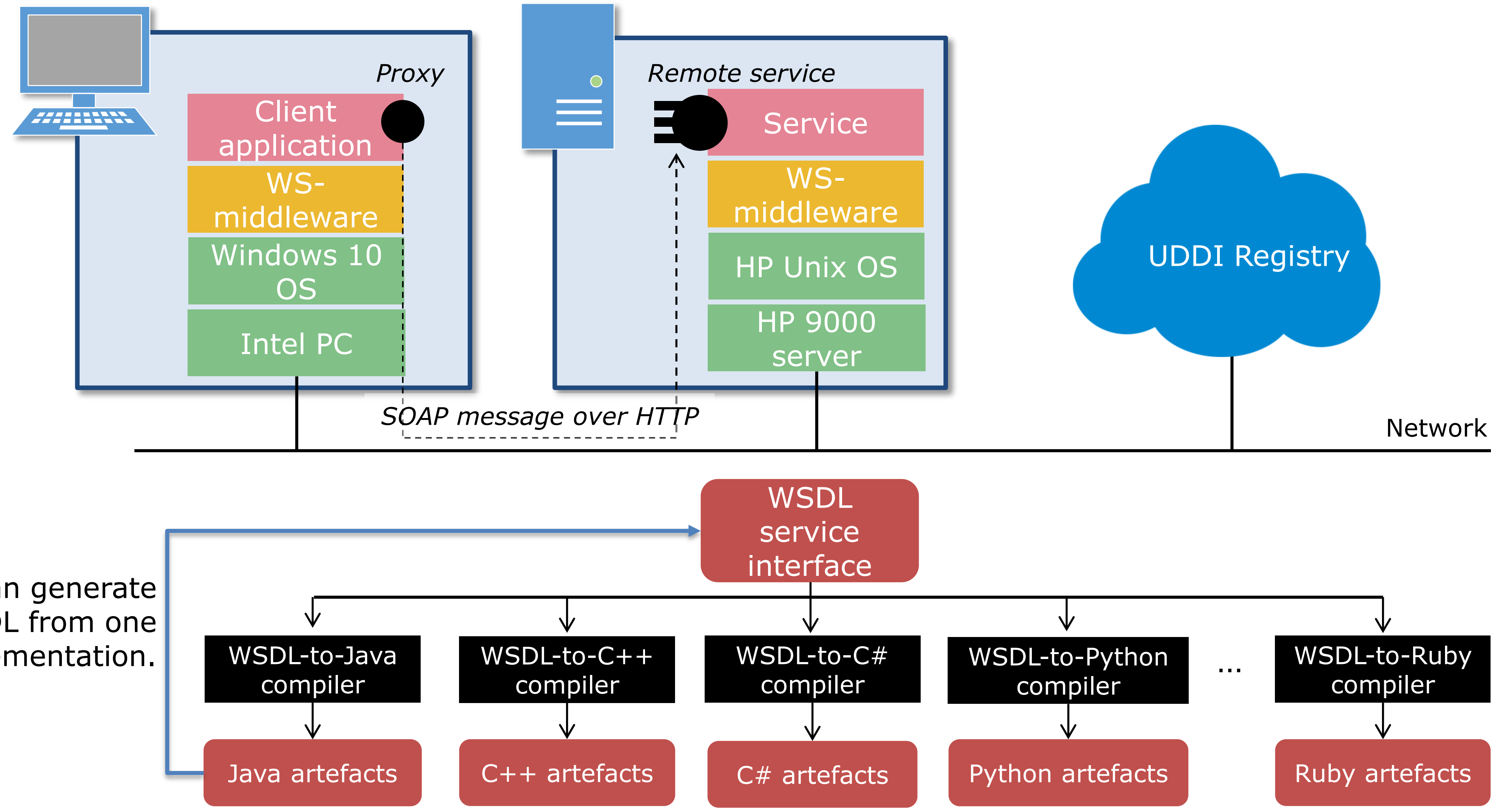
# SOAP over HTTP

```
[HTTP request - http://localhost:8080/multiplyService]
   Accept: text/xml
   Content-Type: text/xml; charset=utf-8
   SOAPAction: “multiplyRequest"

<Body>
    <multiplyMsg>
        <arg0>10</arg0>
        <arg1>10</arg1>
     </multiplyMsg>
</Body>
```



```
[HTTP response - http://localhost:8080/multiplyService - 200]
   HTTP/1.1 200 OK
   Content-type: text/xml; charset=utf-8
   Date: Wed, 09 Jul 2012 01:24:16 GMT

<Body>
     <multiplyResponseMsg>
         <return>100</return>
     <multiplyResponseMsg>
</Body>
```

# SOAP web services



Usually, can generate the WSDL from one implementation.

ENGINEERING

SOFTENG 325 Lecture 03 – Web Services

## REST

## REST web services

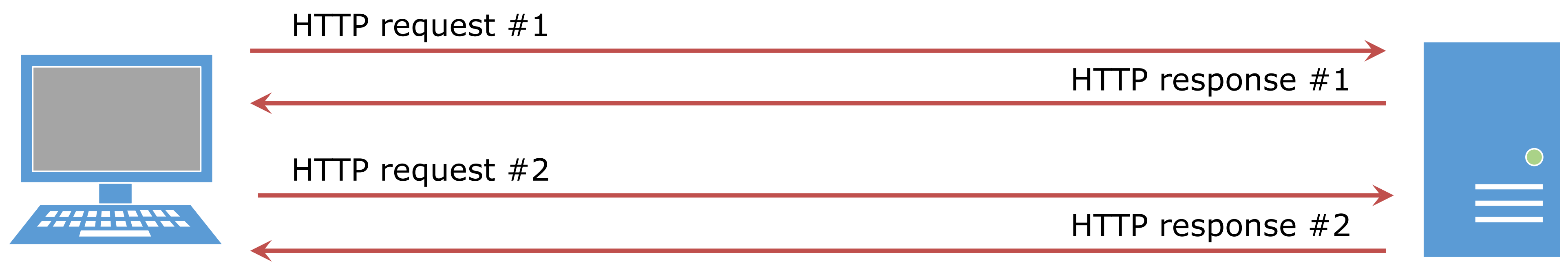
- REST (REpresentational State Transfer) more fully leverages the capabilities of the HTTP protocol
- In addition to being an open request/reply protocol, HTTP:
  - Is a stateless protocol
  - Defines methods for request messages and typed responses
  - Supports negotiable content

| HTTP Method | Purpose |
|---|---|
| GET | • Requests specified resource<br>• Can be made conditional on resource’s last modification time |
| HEAD | • Similar to GET, only returns metadata (e.g. modification time, size, type) |
| POST | • Requests that the named resource processes data<br>• Typically used to process form data |
| PUT | • Requests that the named resource is replaced with data contained in the message body |
| DELETE | • Requests that a resource be deleted |
| OPTIONS | • Requests the methods that are applicable to the named resource |
| TRACE | • Requests that the server simply sends back the request message |

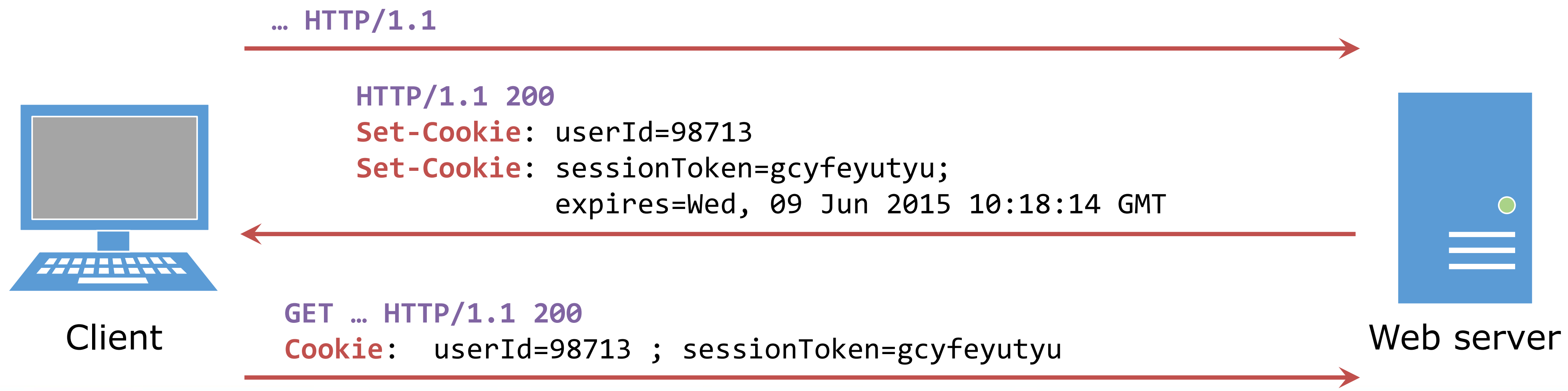
| Response status code | Meaning |
|---|---|
| 1xx | Informational |
| 2xx | Success – the server received the request and successfully carried it out |
| 3xx | Redirection |
| 4xx | Client error. Typically used to represent:<br>• A malformed URL supplied by the client; or<br>• An attempt to access a resource which isn’t held by the server |
| 5xx | Server error |

# Stateless protocol

As a stateless protocol, the server maintains no "session" state between requests.



It's often useful for a server to track which client it is processing requests for.
A cookie allows clients to store session state and send this with each request.



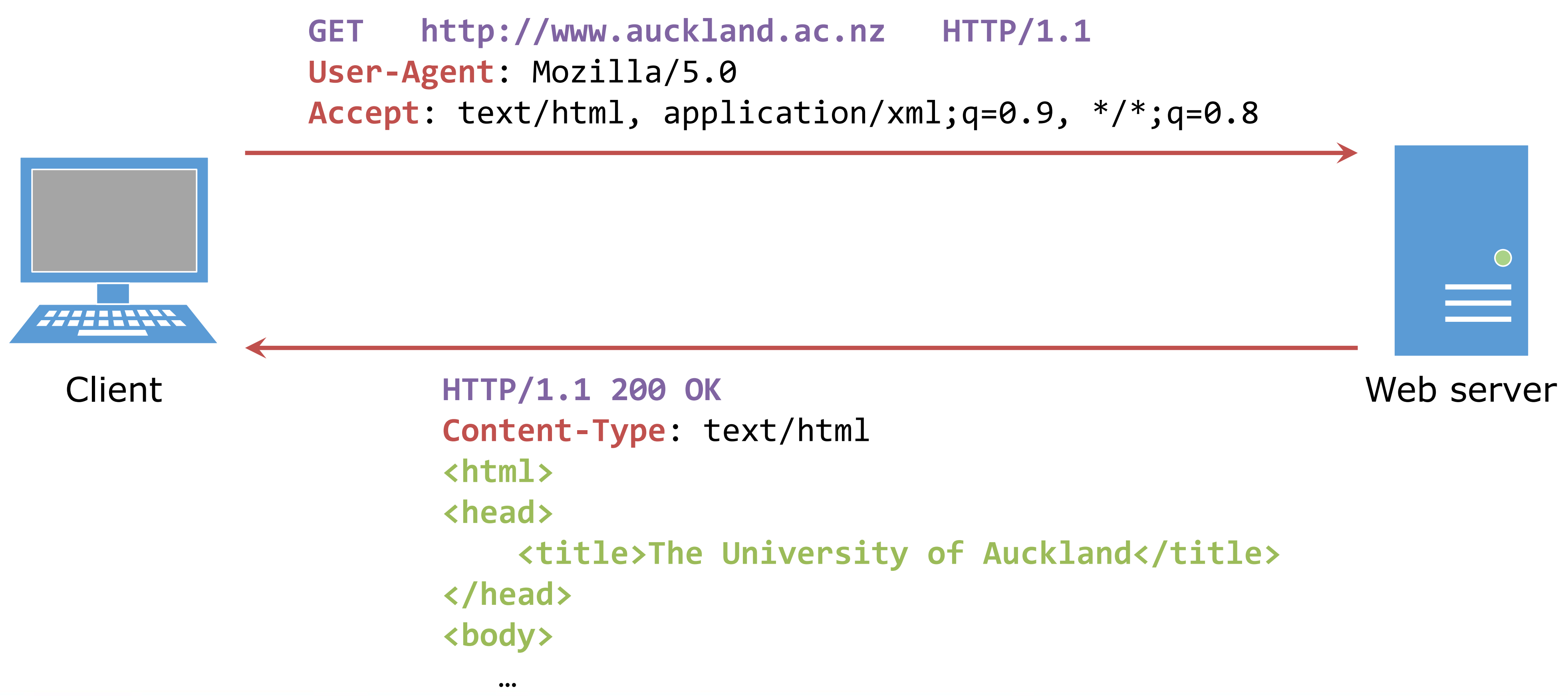
```
… HTTP/1.1
```

```
HTTP/1.1 200
Set-Cookie: userId=98713
Set-Cookie: sessionToken=gcyfeyutyu;
            expires=Wed, 09 Jun 2015 10:18:14 GMT
```

```
GET … HTTP/1.1 200
Cookie:  userId=98713 ; sessionToken=gcyfeyutyu
```

# Negotiable content

- Using HTTP, clients can specify preferences for content



```
GET   http://www.auckland.ac.nz   HTTP/1.1
User-Agent: Mozilla/5.0
Accept: text/html, application/xml;q=0.9, */*;q=0.8
```

```
HTTP/1.1 200 OK
Content-Type: text/html
<html>
<head>
    <title>The University of Auckland</title>
</head>
<body>
   …
```

## REST (Representational State Transfer)

- REST originated in Roy Fielding’s PhD thesis, “Architectural Styles and the Design of Network-based Software Architecture”
- Fielding proposed a set of architectural principles known as REST
  1. Addressable resources
  2. A uniform, constrained interface
  3. Representation-oriented
  4. Communicate statelessly
  5. Hypermedia As The Engine of Application State (HATEOAS)

## REST principle #1: Addressable resources

- Every resource is reachable through a unique identifier
- REST uses URIs to identify resources

**E.g.** `http://online-store.com/orders?id=111` might return the following:

```
{
  "id": 111,
  "customer": "http://online-store.com/customers/32133",
  "entries": [
    {
      "quantity": 5,
      "product": "http://online-store.com/products/111"
    },
    …
```

*(JSON notation – we'll learn about that next week!)*

## REST Principle #2: Uniform, constrained interface

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND | ENGINEERING

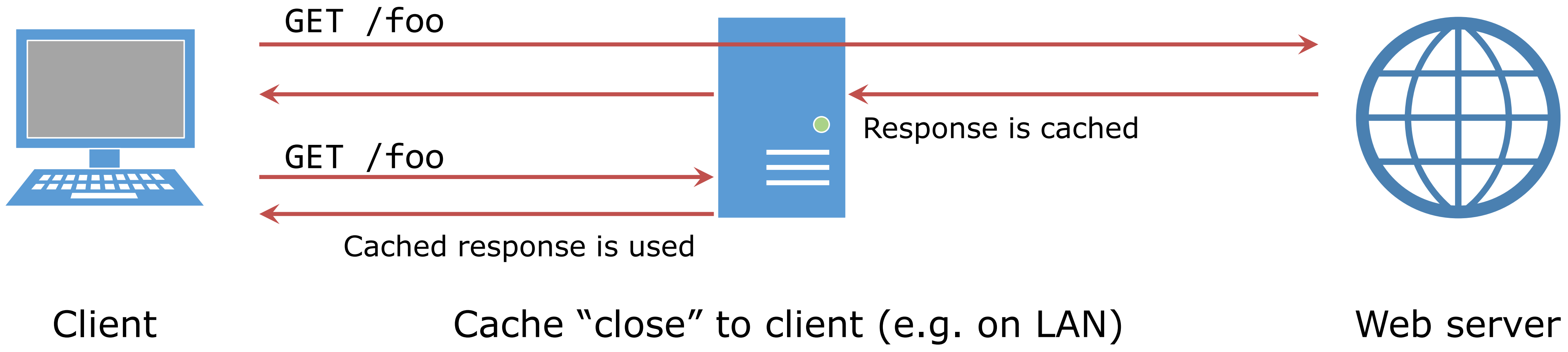
- Use the HTTP methods, as intended, in implementing the service

| HTTP method | Description | Idempotent? | Safe? | CRUD operation |
|---|---|---|---|---|
| GET | A read-only operation that queries a server for specific information | Yes | Yes | Retrieve |
| PUT | Requests that the server stores the request's message body under the location specified in the HTTP request | Yes | No | Update |
| DELETE | Removes a specified resource | Yes | No | Delete |
| POST | Changes the state of a service based in some way, e.g. creating a new resource | No | No | Create |

See https://restfulapi.net/http-methods/ for further info

## REST Principle #2: Uniform, constrained interface

- Use of HTTP methods as intended can allow us to perform optimizations
  - For example, where GET's semantics are respected, GET responses can be cached, contributing to scalability

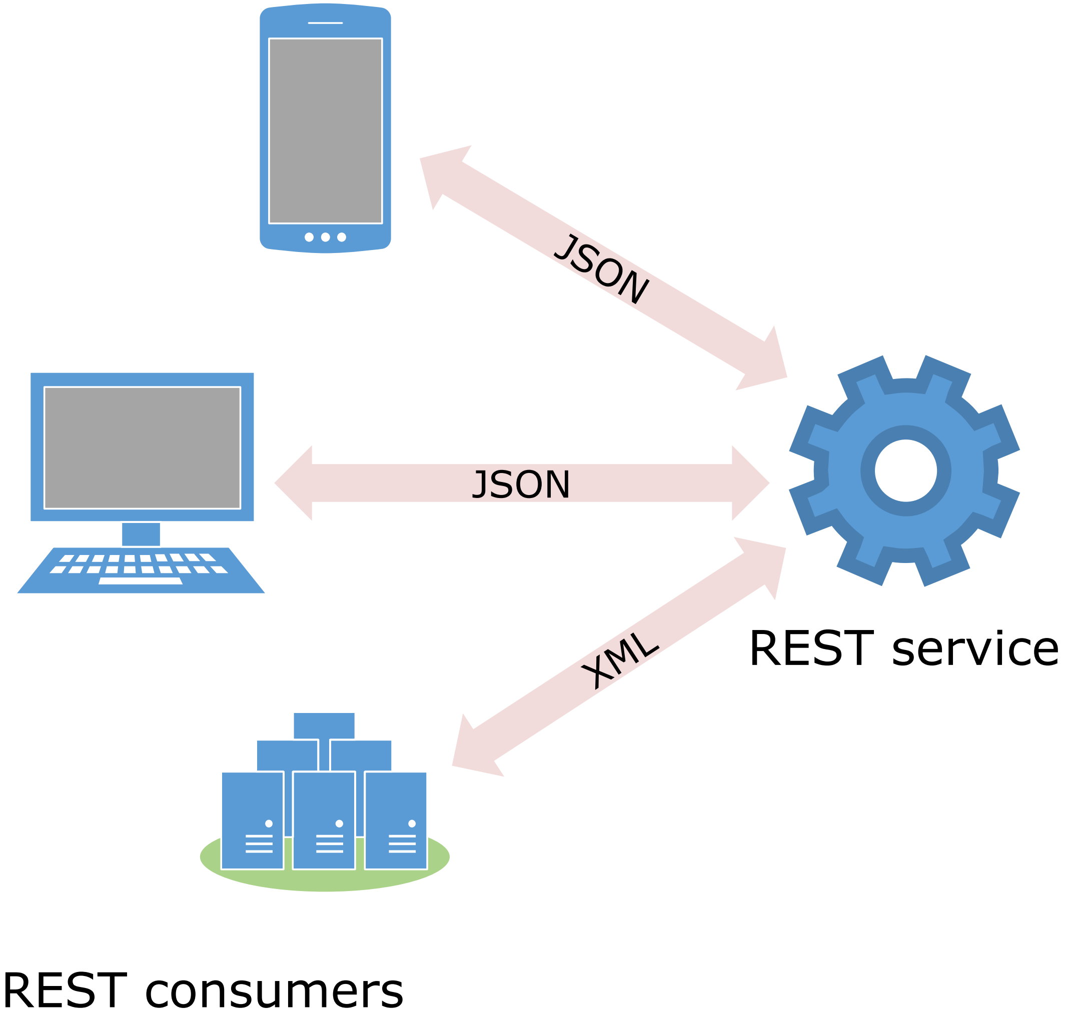


## REST Principle #3: Representation-oriented

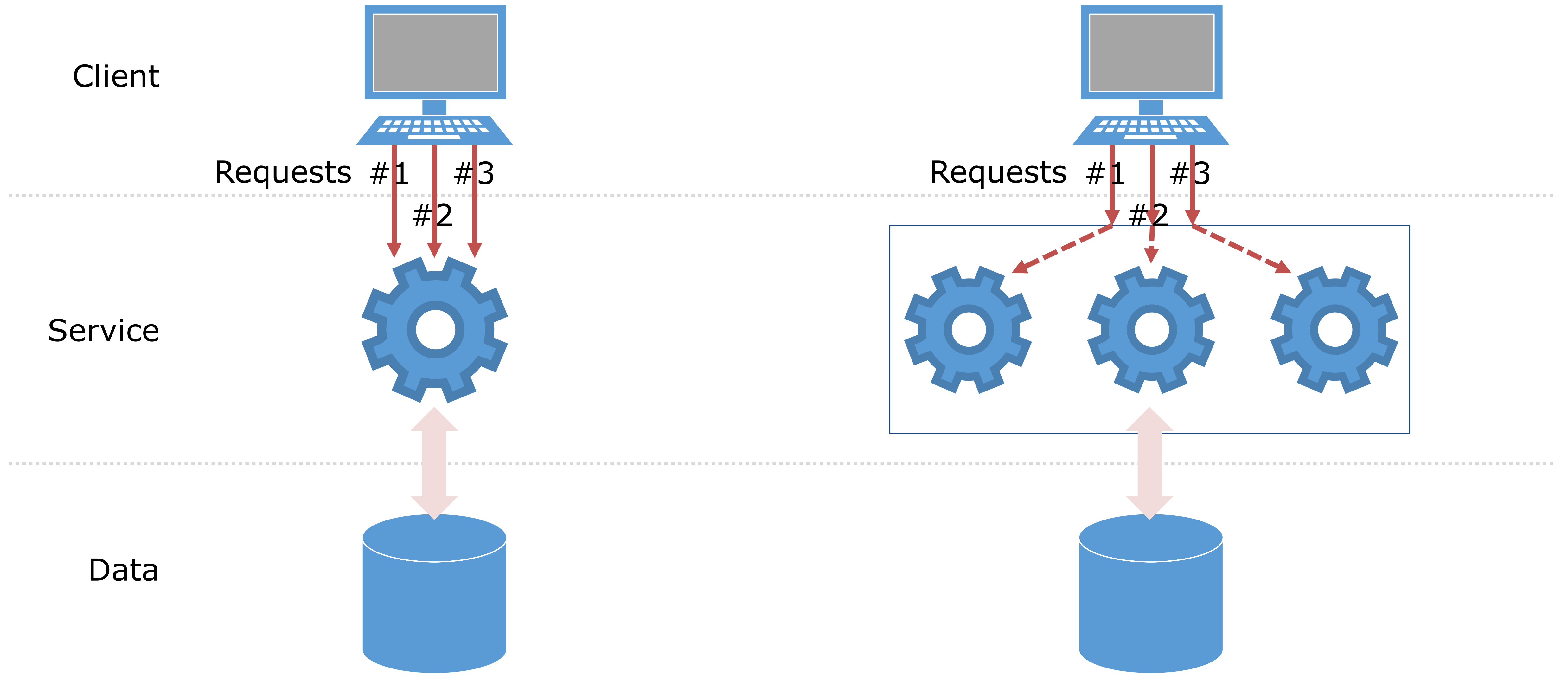
- Consumers and services exchange representations of resources
  - Representations can take many forms, as specified by MIME types

```
type/subtype

text/plain
text/html
application/xml
application/json
```

- Consumers can negotiate a representation with a service
  - Using HTTP request `Accept` header and `Content-Type` response header



## REST Principle #4: Stateless communication



## REST Principle #5: HATEOAS

- Hypermedia As The Engine Of Application State
  - Responses contain links telling the client “where they can go next”. For example:

First request: `GET /products`

Response: first five entries

```
HTTP/1.1 200 OK
Content-Type: application/json
Link: /products?startIndex=5;rel="next"
[
  {
    "id": 0,
    "name": "headphones",
    "price": "$16.99"
  },
  {
    "id": 1,
  …
]
```

Second request: `GET /products?startIndex=5`

Response: next five entries

```
HTTP/1.1 200 OK
Content-Type: application/json
Link: /products?startIndex=0;rel="previous"
Link: /products?startIndex=10;rel="next"
[
  {
    "id": 5,
    "name": "meaning of life",
    "price": "$42.00"
  },
  {
    "id": 6,
  …
]
```

## SOAP vs REST

### SOAP

- HTTP is used as nothing more than a transport protocol
- SOAP is based on many standards, e.g. SOAP and WSDL, and requires associated tools
- SOAP services have formally defined contracts that specify service interfaces

### REST

- HTTP and its features are leveraged; HTTP provides a service API
- REST relies only on HTTP; there’s no need for other standards and tools
- REST is ad-hoc; service contracts are not well defined (REST interfaces don’t specify the type of data to be exchanged)

## What have we learned today?

- SOA involves distributed systems made up of services that can be accessed and consumed without regard to their underlying implementation technology
- Servlet containers are a form of middleware that simplify development of Java Web applications
  - Servlet containers provide an abstraction that handles networking and resource management
  - Servlet containers host servlets, which implement application-logic in response to HTTP requests
- Web services are widely used to build SOAs; Web services come in two forms:
  - SOAP offers an RMI-like mechanism that uses HTTP as the transport protocol and which introduces XML-based technologies for defining Web service interfaces (WSDL), representing data types (XMLSchema types), describing service calls (SOAP), and registering service interfaces (UDDI)
  - REST is an architectural style , comprising 5 key principles , for developing Web services by leveraging HTTP and without the need for additional specifications and tool support; REST has become prevalent in industry