3-Tier REST services

Part 1

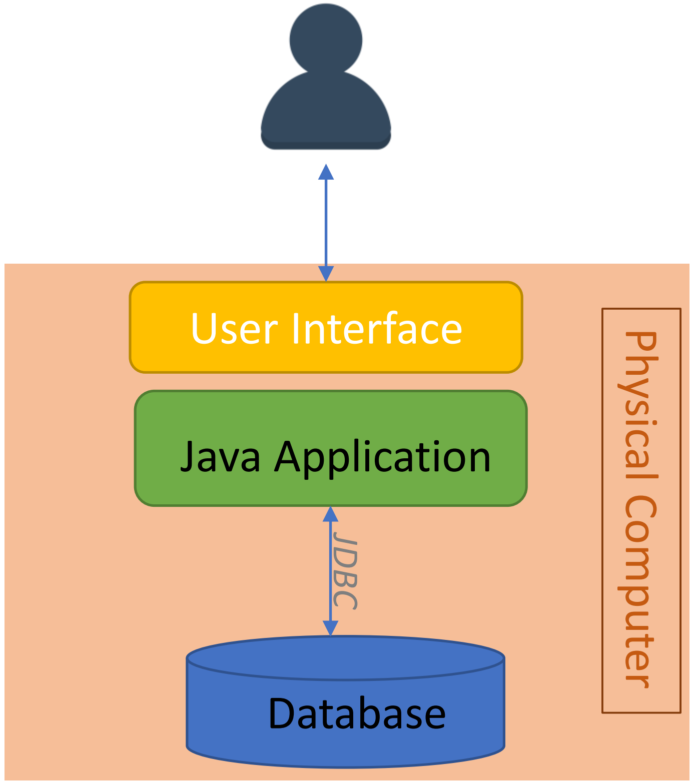
2020

Contents

- Client-Server architecture
  - TCP/IP Network Stack
  - HTTP protocol
- Handling HTTP requests
  - A basic example
  - Java Servlets
  - Technologies evolution
  - REST API definition
  - Providing State
- Recap: Theoretical Background
  - TCP/IP
    - IP and Port
    - URL and URI
  - HTTP Protocol
    - HTTP methods
    - GET vs POST
    - Providing state
- Conclusion

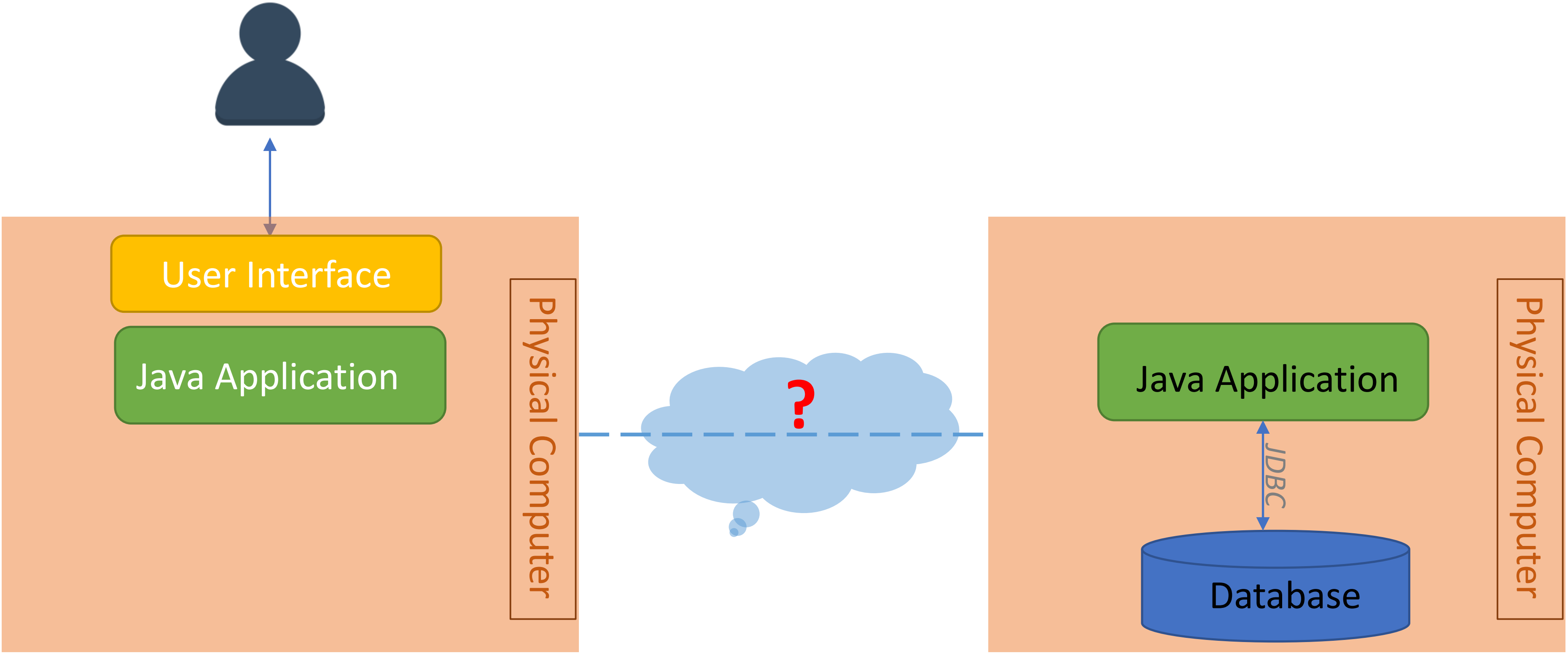
Client-Server Architecture (1)

PT 2nd year assignments…..



Client-Server Architecture (2)

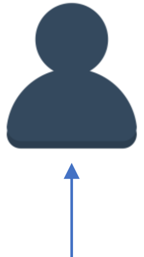
What if the client is on a different location using another computer? How can he access the same resources?



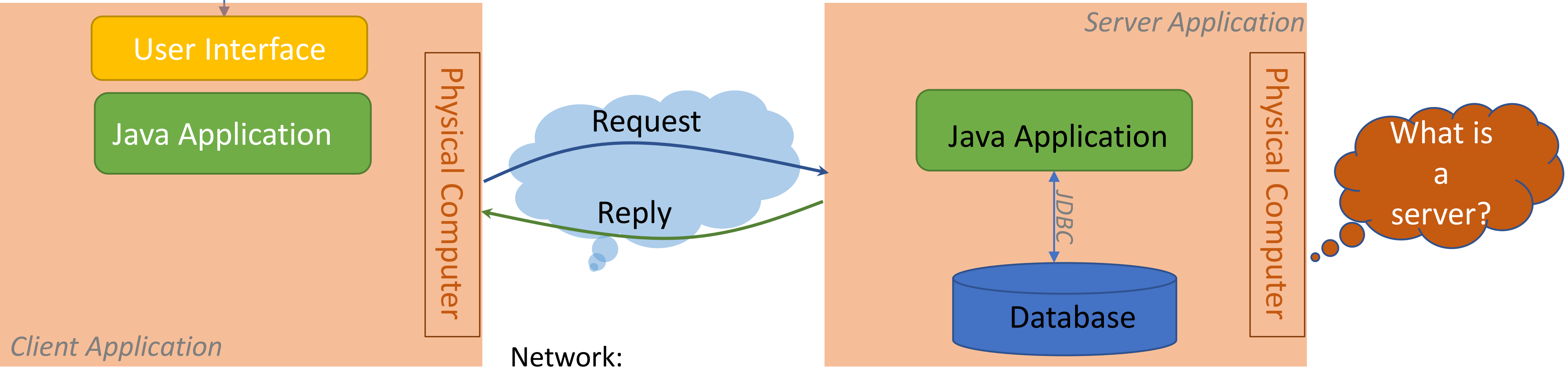
Client-Server Architecture (3)

What if the client is on a different location using another computer:

- The client application can request some resources
- The server application can response with the requested resources



Difference between **physical server** and **software server**



Client applications:

- Web browsers
- Desktop applications

(e.g. MySQL Workbench)

- Mobile applications

Network:

- Connection medium: copper (UTP cables, optical fiber, wireless, GSM network, etc.
- Software network stack: TCP/IP

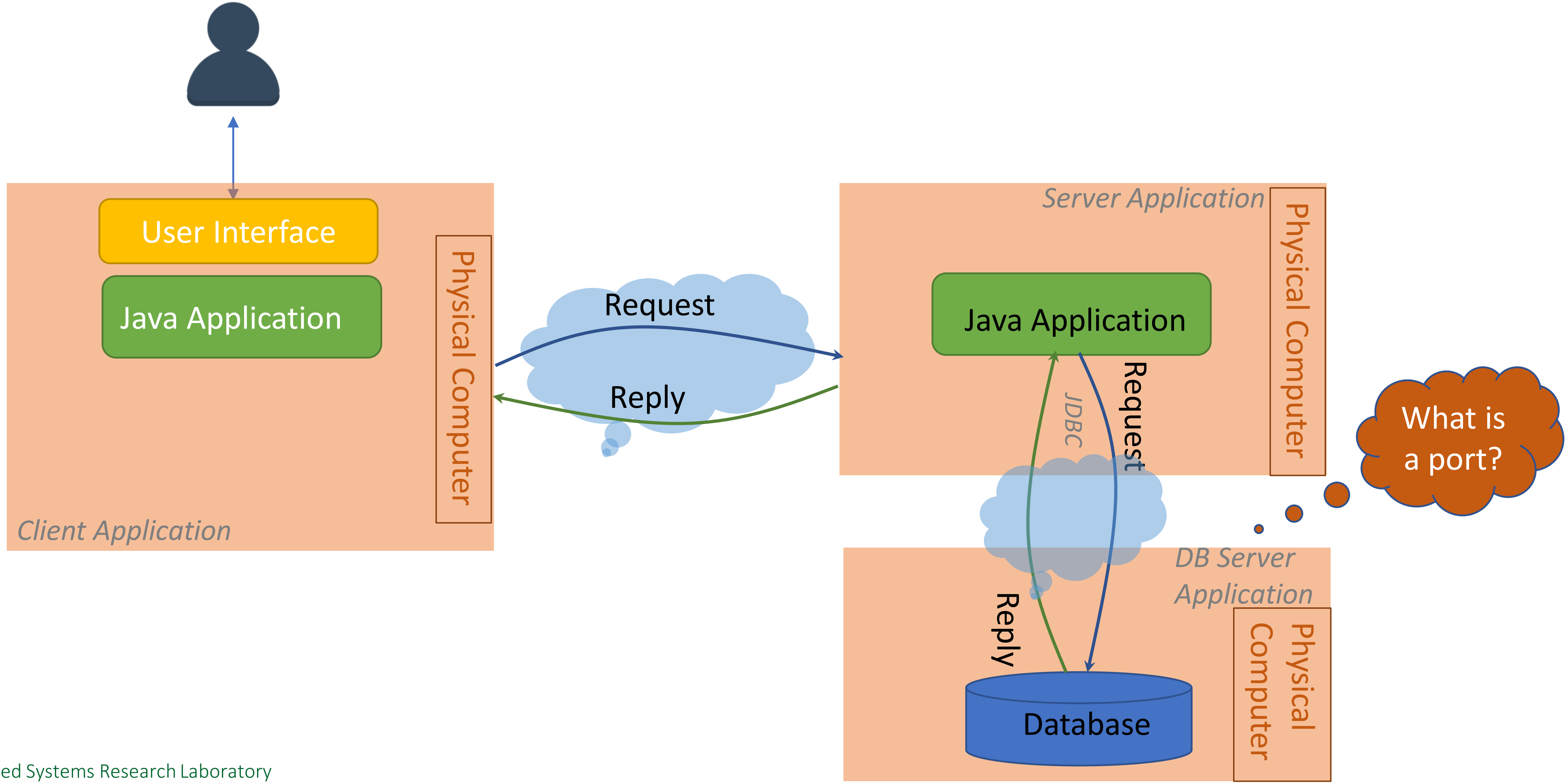
Server applications:

- Web servers (e.g. Apache Tomcat, Glassfish, IIS)
- DB Servers (e.g. MySQL, SQL Server, Oracle, etc.)

Client-Server Architecture (4)

What if the client is on a different location using another computer:

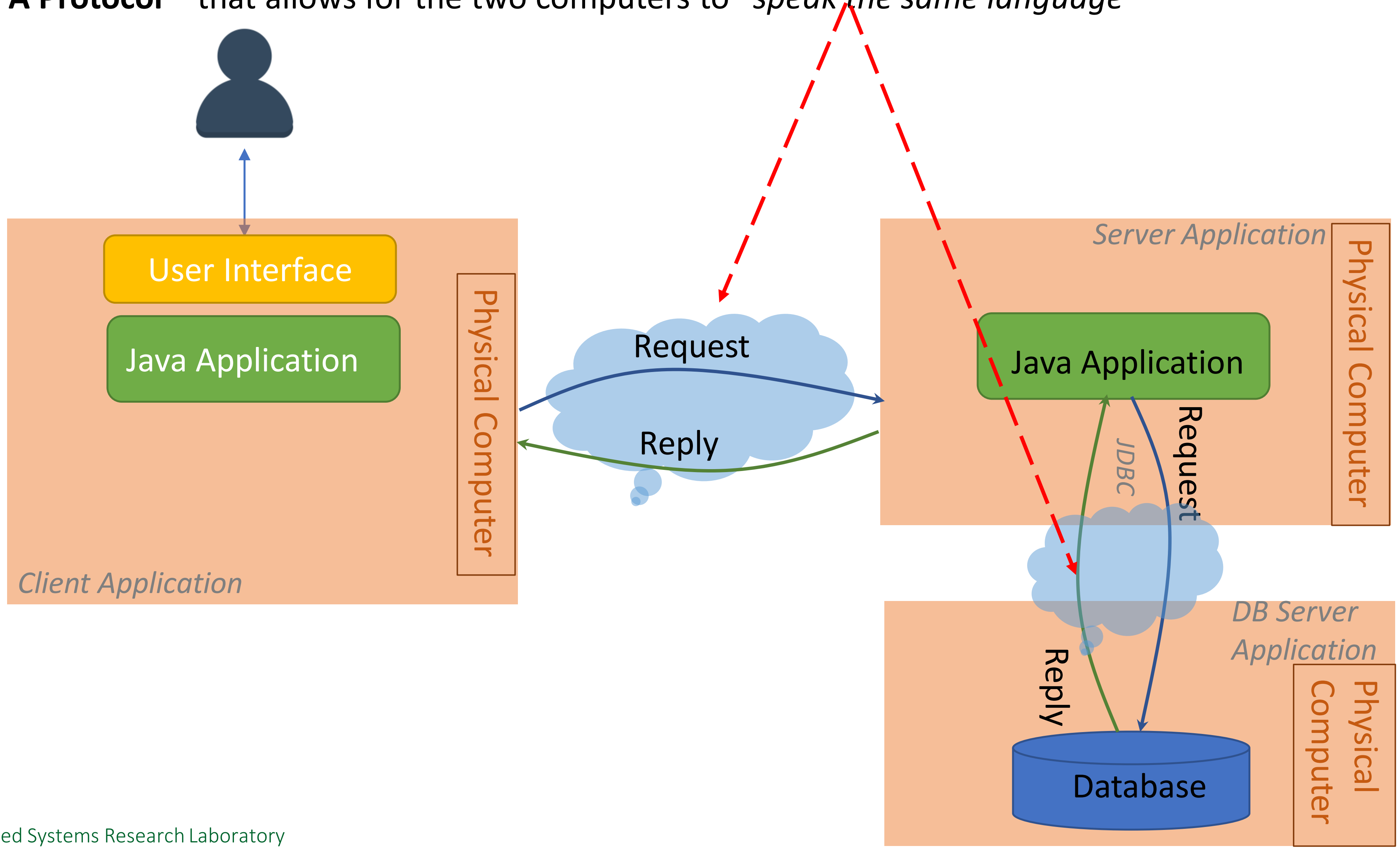
- The client application can request some resources
- The server application can response with the requested resources



Client-Server Architecture (5)

How is the connection made between the two-by-two computers?

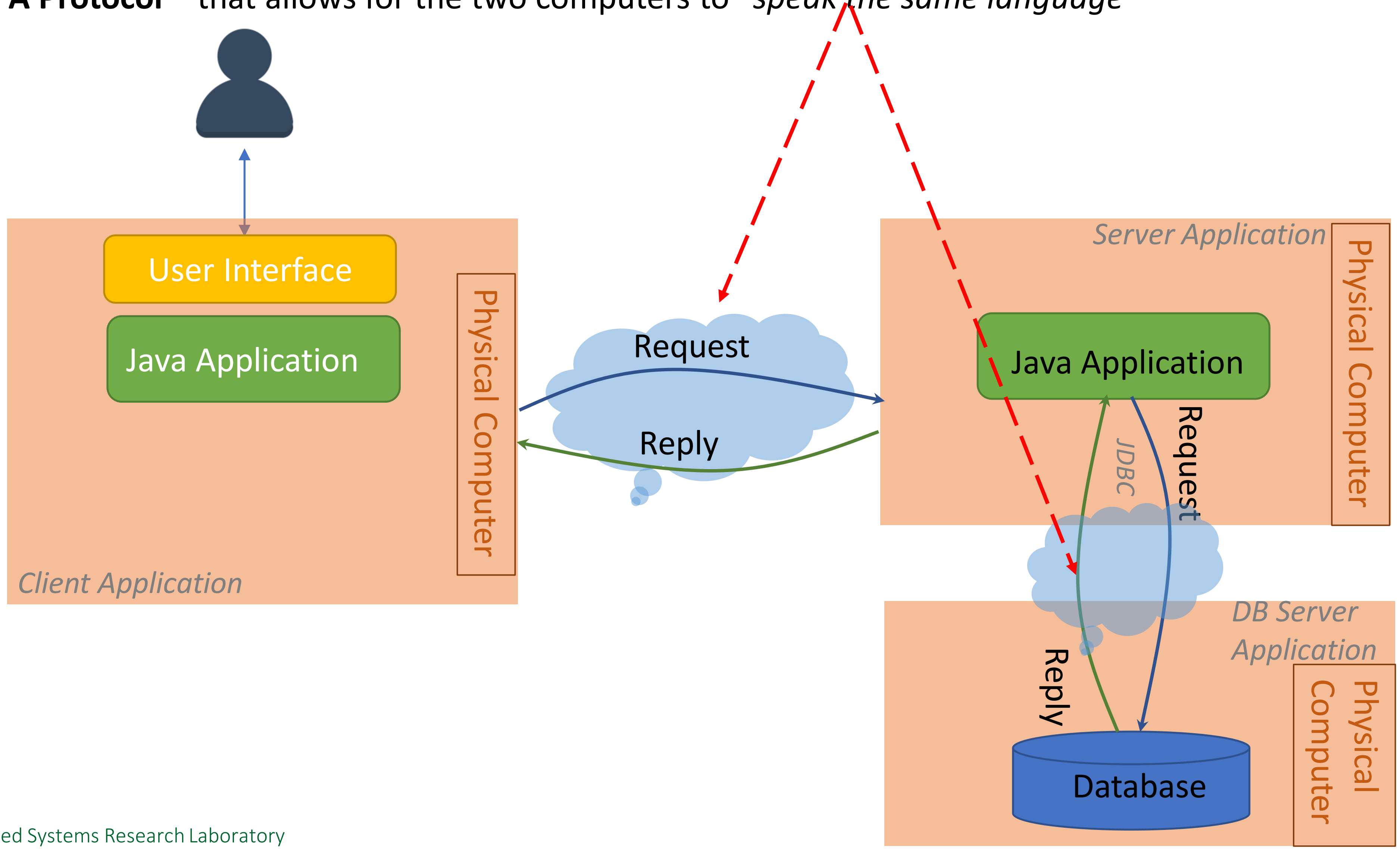
1. **Network Connection TCP/IP** – that connects the two computers through the network
2. **A Protocol** – that allows for the two computers to "*speak the same language*"



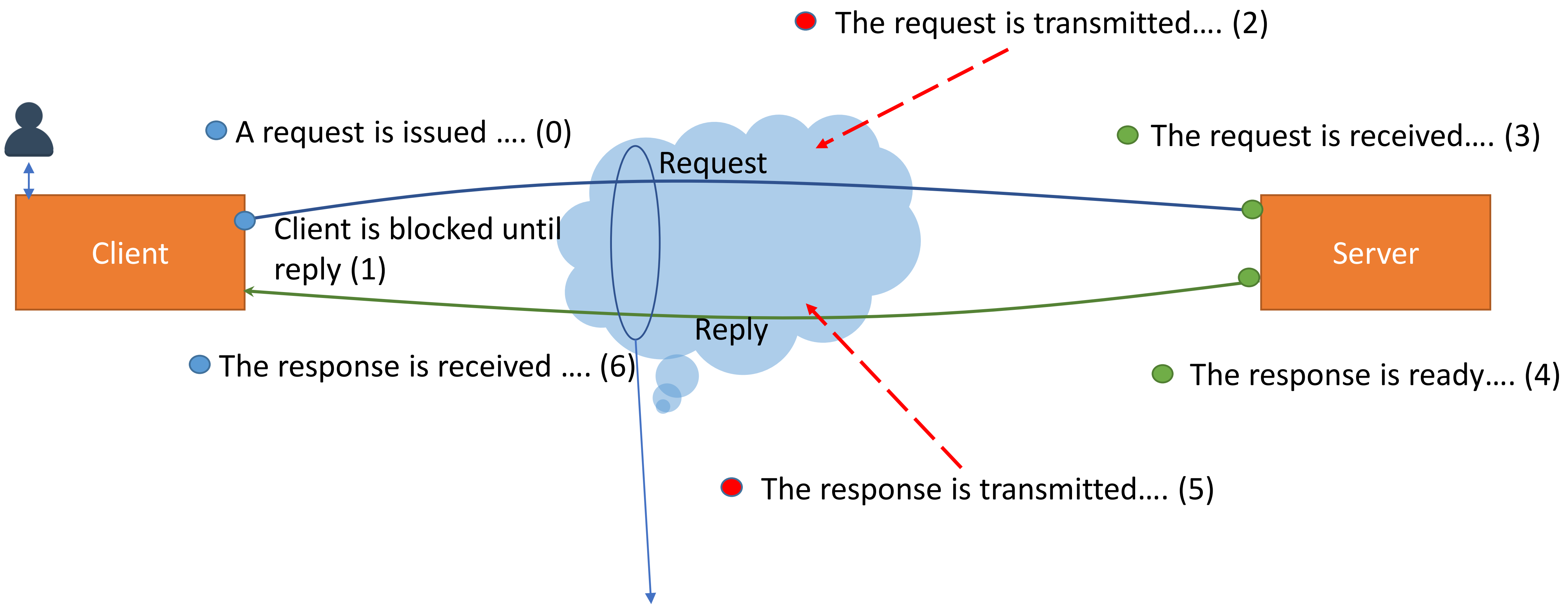
Client-Server Architecture (6)

How is the connection made between the two-by-two computers?

1. **Network Connection TCP/IP** – that connects the two computers through the network
2. **A Protocol** – that allows for the two computers to "*speak the same language*"



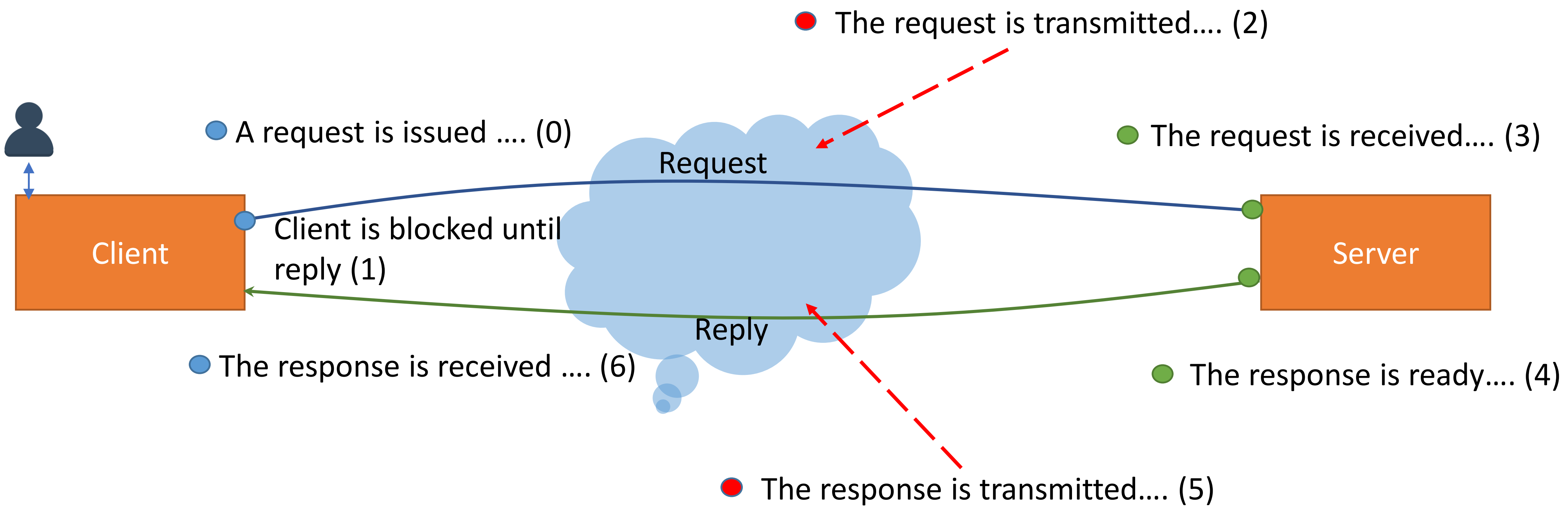
Client-Server Architecture- Network Connection TCP/IP (1)



● The request is transmitted…. (2)
● A request is issued …. (0)
● The request is received…. (3)
Request
Client
Client is blocked until reply (1)
Server
Reply
● The response is received …. (6)
● The response is ready…. (4)
● The response is transmitted…. (5)

Synchronous Communication

Client-Server Architecture- Network Connection TCP/IP (2)


The request is transmitted…. (2)
A request is issued …. (0)
Request
The request is received…. (3)
Client
Client is blocked until reply (1)
Server
Reply
The response is received …. (6)
The response is ready…. (4)
The response is transmitted…. (5)

How are the messages transmitted?

Client-Server Architecture- Network Connection TCP/IP (3)

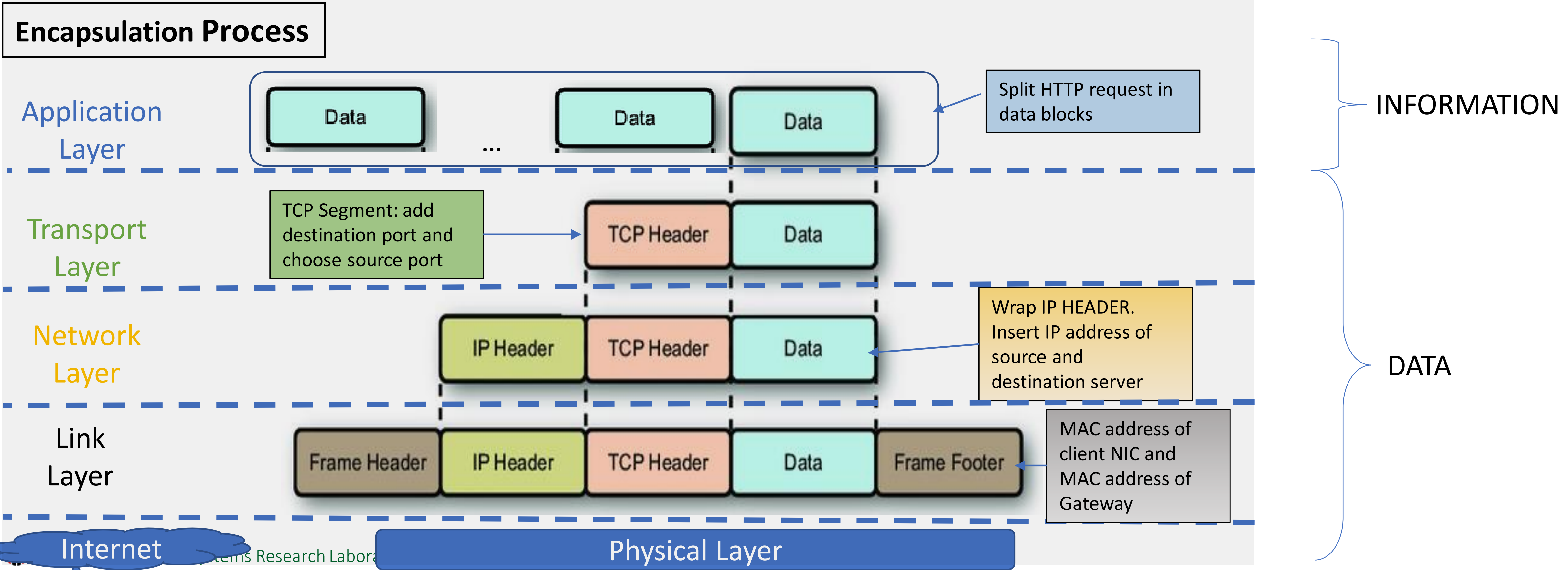
Client

Server

A request is issued …. (1)

For the request to be successfully transmitted:

- Encapsulate the data
- Incorporate relevant network related information



Encapsulation Process

Application Layer

Transport Layer

Network Layer

Link Layer

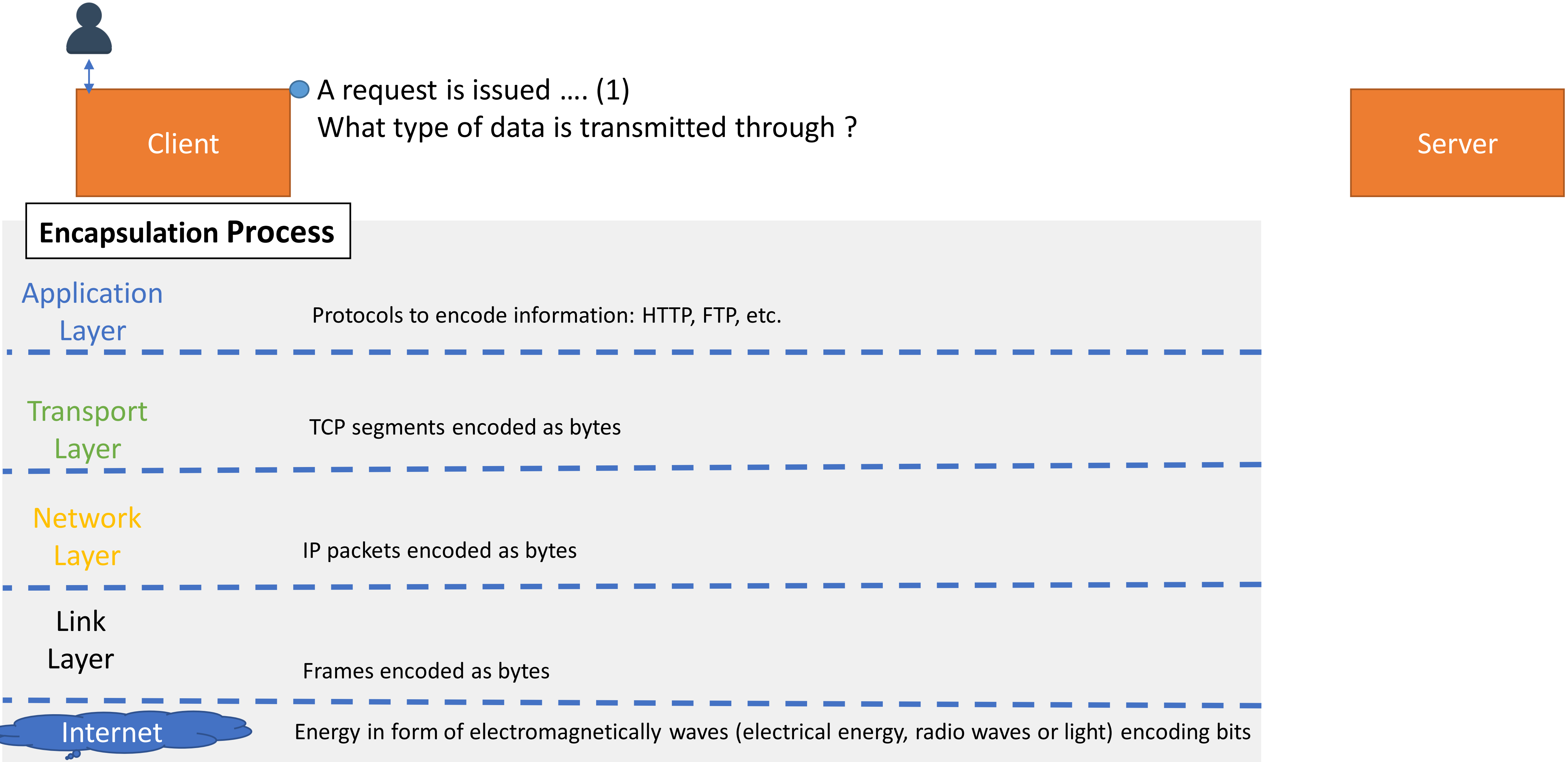
Split HTTP request in data blocks

TCP Segment: add destination port and choose source port

Wrap IP HEADER. Insert IP address of source and destination server

MAC address of client NIC and MAC address of Gateway

Client-Server Architecture- Network Connection TCP/IP (4)



Client

Server

A request is issued …. (1)

What type of data is transmitted through ?

Encapsulation Process

Application Layer — Protocols to encode information: HTTP, FTP, etc.

Transport Layer — TCP segments encoded as bytes

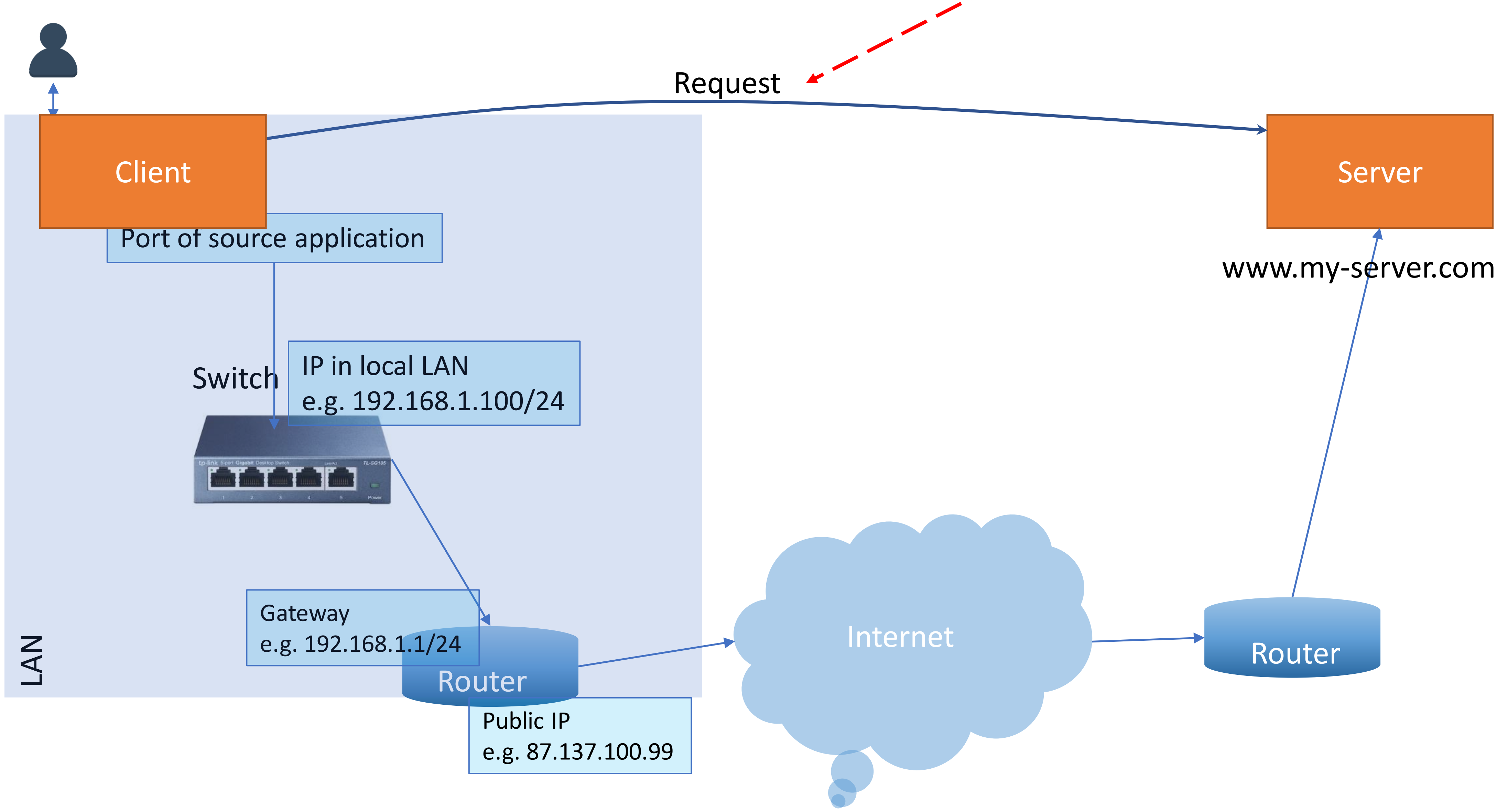
Network Layer — IP packets encoded as bytes

Link Layer — Frames encoded as bytes

Internet — Energy in form of electromagnetically waves (electrical energy, radio waves or light) encoding bits

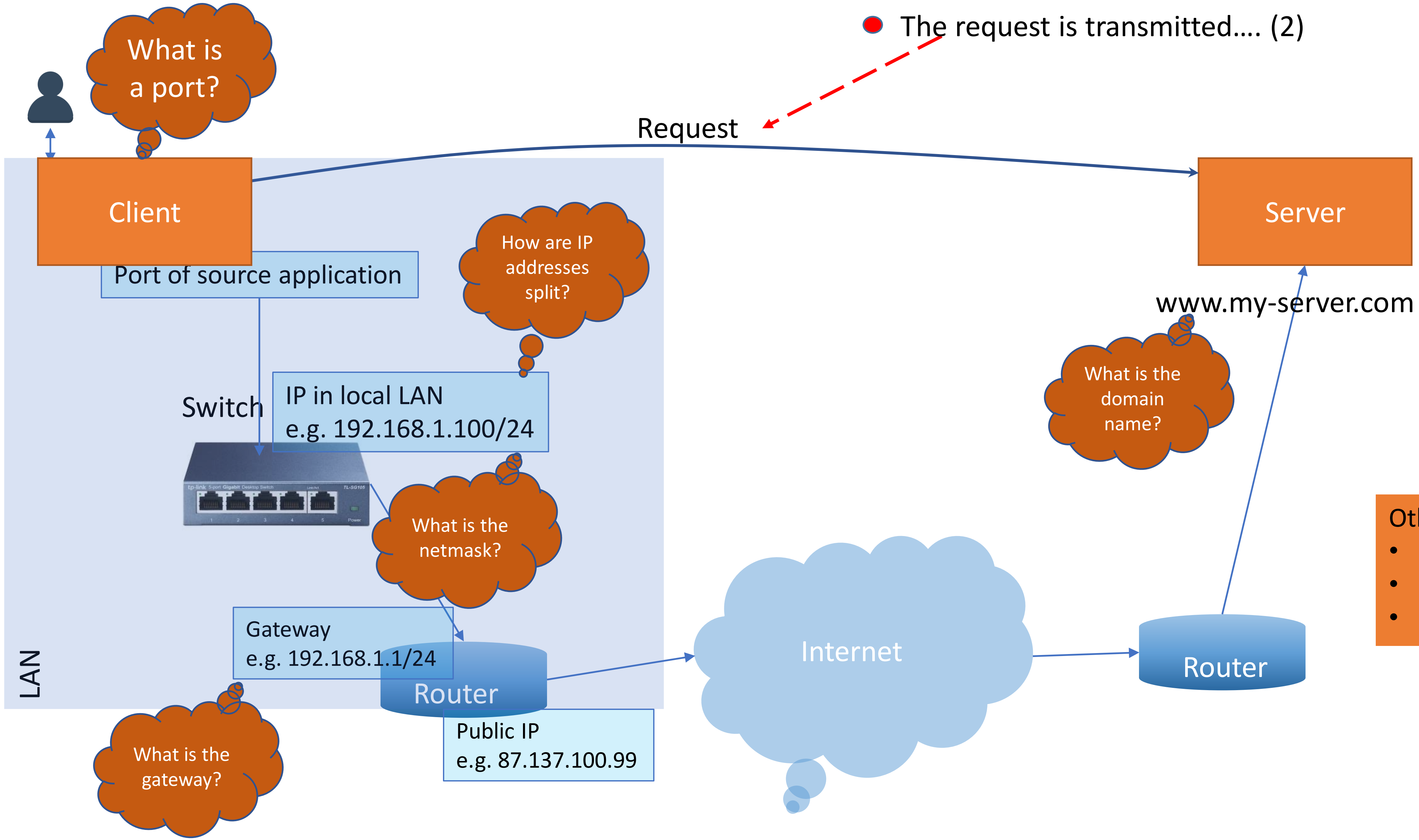
Client-Server Architecture- Network Connection TCP/IP (5)

- The request is transmitted…. (2)



Client-Server Architecture- Network Connection TCP/IP (5)

- The request is transmitted…. (2)



Request

What is a port?

Client

Port of source application

How are IP addresses split?

Switch

IP in local LAN
e.g. 192.168.1.100/24

What is the netmask?

Gateway
e.g. 192.168.1.1/24

Router

Public IP
e.g. 87.137.100.99

What is the gateway?

LAN

Internet

Router

Server

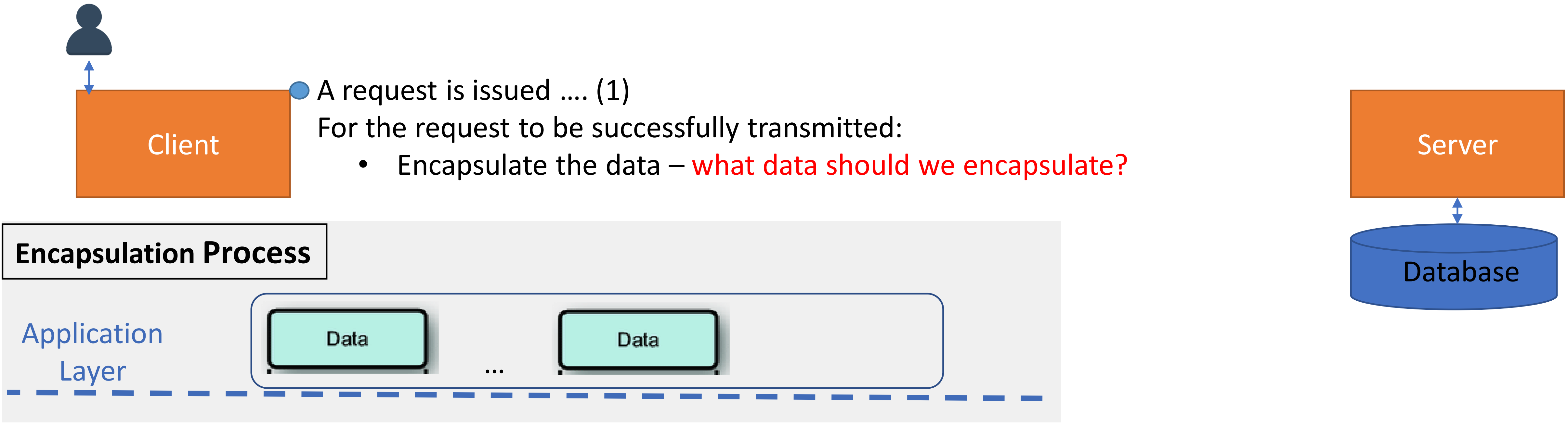
www.my-server.com

What is the domain name?

Other concepts:

- DNS
- URL
- URI

Client-Server Architecture- The Protocol (1)



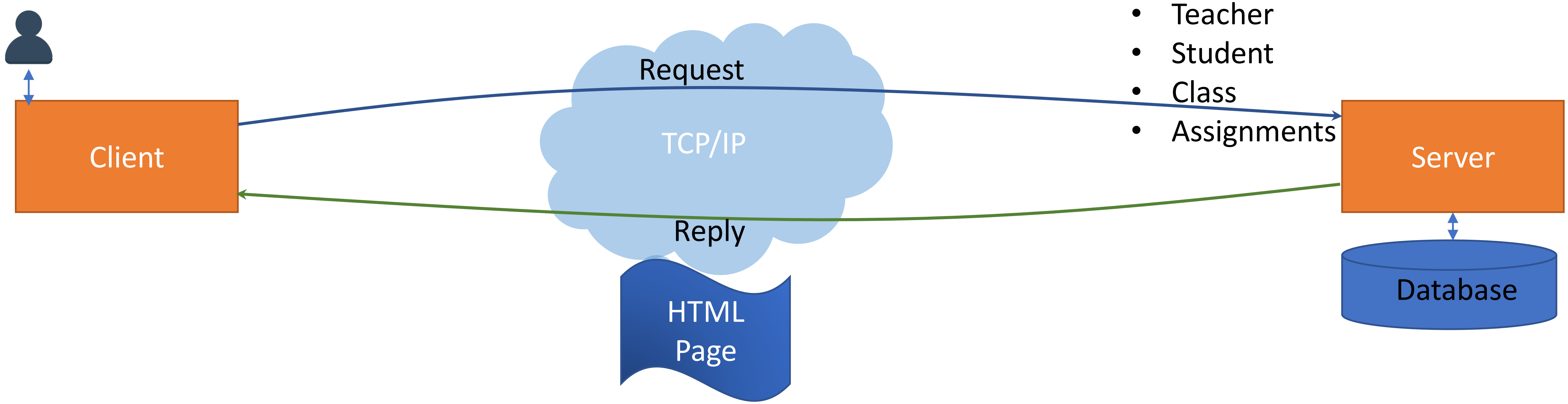
A request is issued .... (1)

For the request to be successfully transmitted:

- Encapsulate the data – what data should we encapsulate?

Encapsulation Process

Client-Server Architecture- The Protocol (2)



- Resources available:
  - Teacher
  - Student
  - Class
  - Assignments

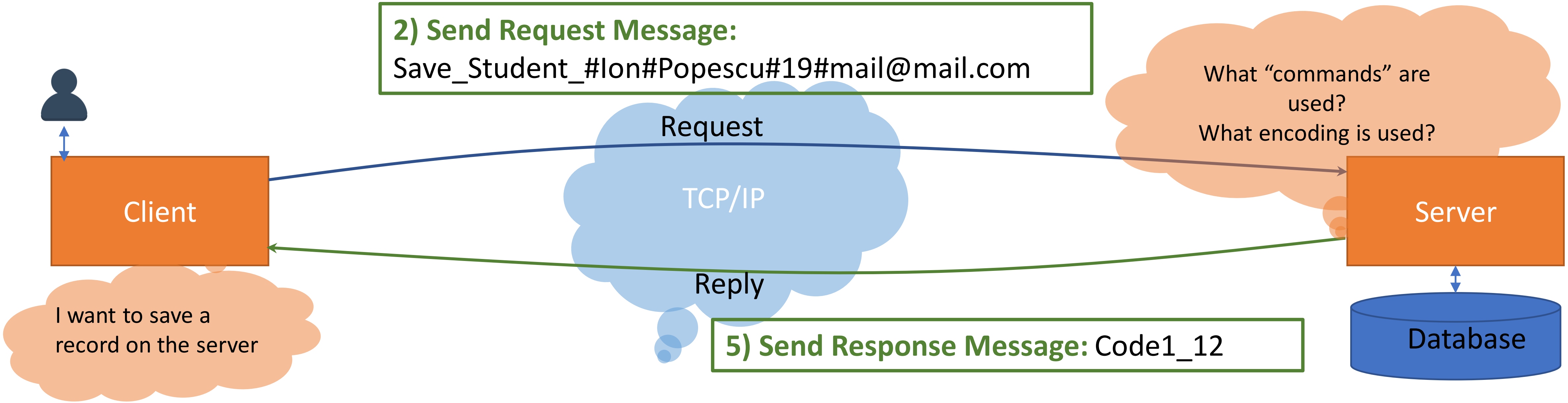
The data/information sent by the client needs to be understood by the server!!
=> We need a Protocol!

- Generate a message for server
- Send message

Send Request Message: ….

- Read message
- Decode commands from message
- Execute commands
- Return operation result

Client-Server Architecture- The Protocol (3)



2) Send Request Message:
Save_Student_#Ion#Popescu#19#mail@mail.com

5) Send Response Message: Code1_12

Client
I want to save a record on the server
Request
TCP/IP
Reply
Server
What “commands” are used?
What encoding is used?
Database

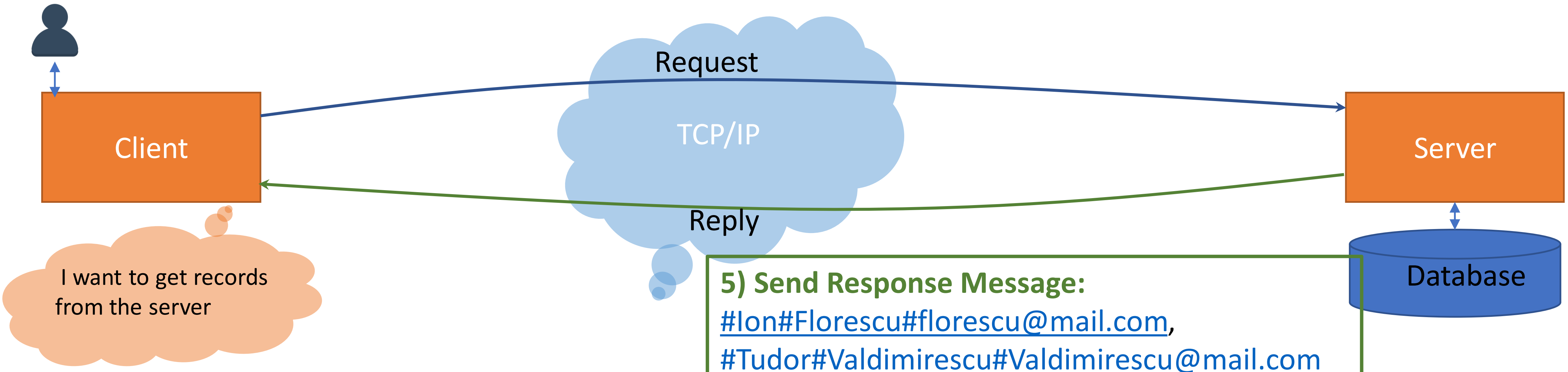
1) Encode

- What operation do I want to make?
  - Save a resource
- What resource I want to save?
  - Student
- What are the details about the student that should be saved?
  - First Name : Ion
  - Last Name: Popescu
  - Age: 19
  - Email: mail@mail.com

- 3) Receive & Decode Message:
  - First Element: Method Required
    - Save
  - Second Element: Resource
    - Student
  - Third Element: Details
    - #Ion#Popescu#19#mail@mail.com
- 4) Encode response:
  - Was save successful?
    - Yes => Code 1
  - What is the id?
    - 12

Client-Server Architecture- The Protocol (4)

2) Send Request Message: Find_Teacher



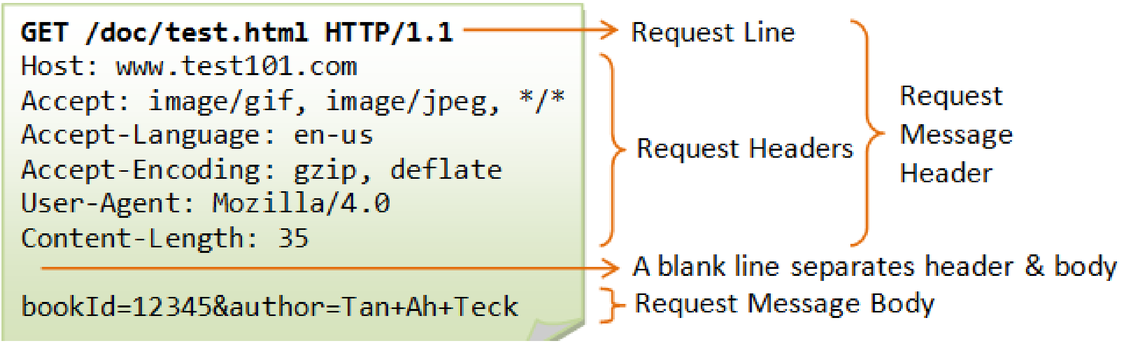
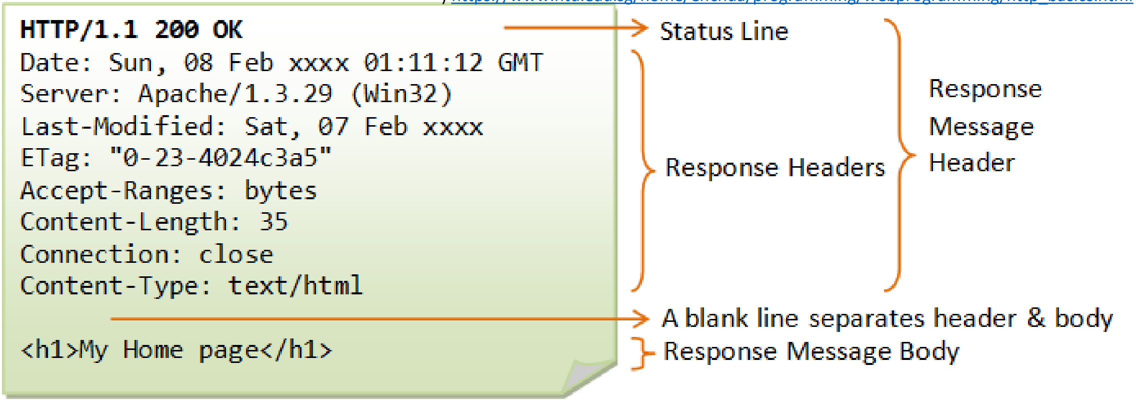
5) Send Response Message:
#Ion#Florescu#florescu@mail.com,
#Tudor#Valdimirescu#Valdimirescu@mail.com

1) Encode

- What operation do I want to make?
  - Find a resource
- What resource I want to save?
  - Teachers

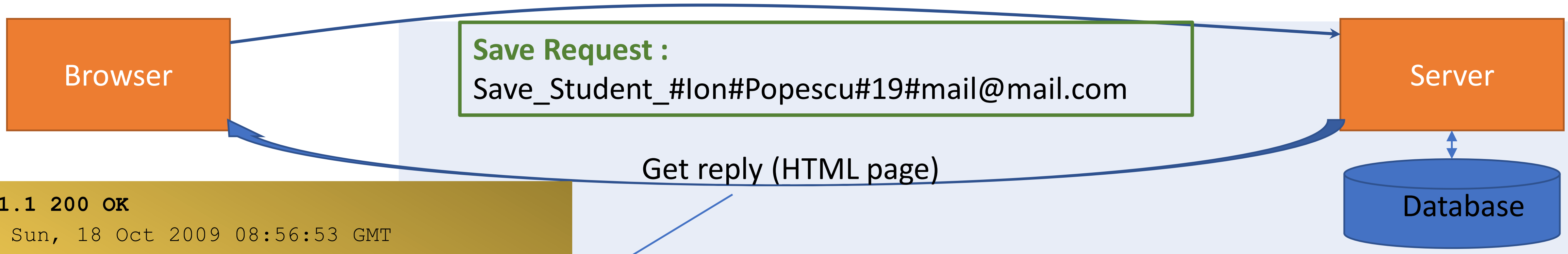
- 3) Receive & Decode Message:
  - First Element: Method Required
    - Find
  - Second Element: Resource
    - Teacher
- 4) Encode response:
  Teacher1 details, Teacher2 details

Client-Server Architecture- The Protocol (5)

Elements	Our Protocol	HTTP
Example Request	Find_Teacher	 GET /doc/test.html HTTP/1.1 Host: www.test101.com Accept: image/gif, image/jpeg, */* Accept-Language: en-us Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 Content-Length: 35 bookId=12345&author=Tan+Ah+Teck Request Line Request Headers Request Message Header A blank line separates header & body Request Message Body *) https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html
Example Response	#Ion#Florescu#florescu@mail.com, #Tudor#Valdimirescu#Valdimirescu@mail.com	 HTTP/1.1 200 OK Date: Sun, 08 Feb xxxx 01:11:12 GMT Server: Apache/1.3.29 (Win32) Last-Modified: Sat, 07 Feb xxxx ETag: "0-23-4024c3a5" Accept-Ranges: bytes Content-Length: 35 Connection: close Content-Type: text/html <h1>My Home page</h1> Status Line Response Headers Response Message Header A blank line separates header & body Response Message Body *) https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html
Method	Save/ Find	POST, PUT, DELETE, UPDATE, etc.
Resource	Resource Name (student)	Specified in URL
Body	Elements separated by # #Ion#Popescu#19#mail@mail.com	HTTP Body
Status Code	Code1, Code0	HTTP Status : 200, 404, 500, etc.
Response	Elements separated by # #Tudor#Valdimirescu#Valdimirescu@mail.com	HTTP Response

Handling HTTP requests – A basic Example

How could our protocol be implemented in the application?



Save Request :
Save_Student_#Ion#Popescu#19#mail@mail.com

Get reply (HTML page)

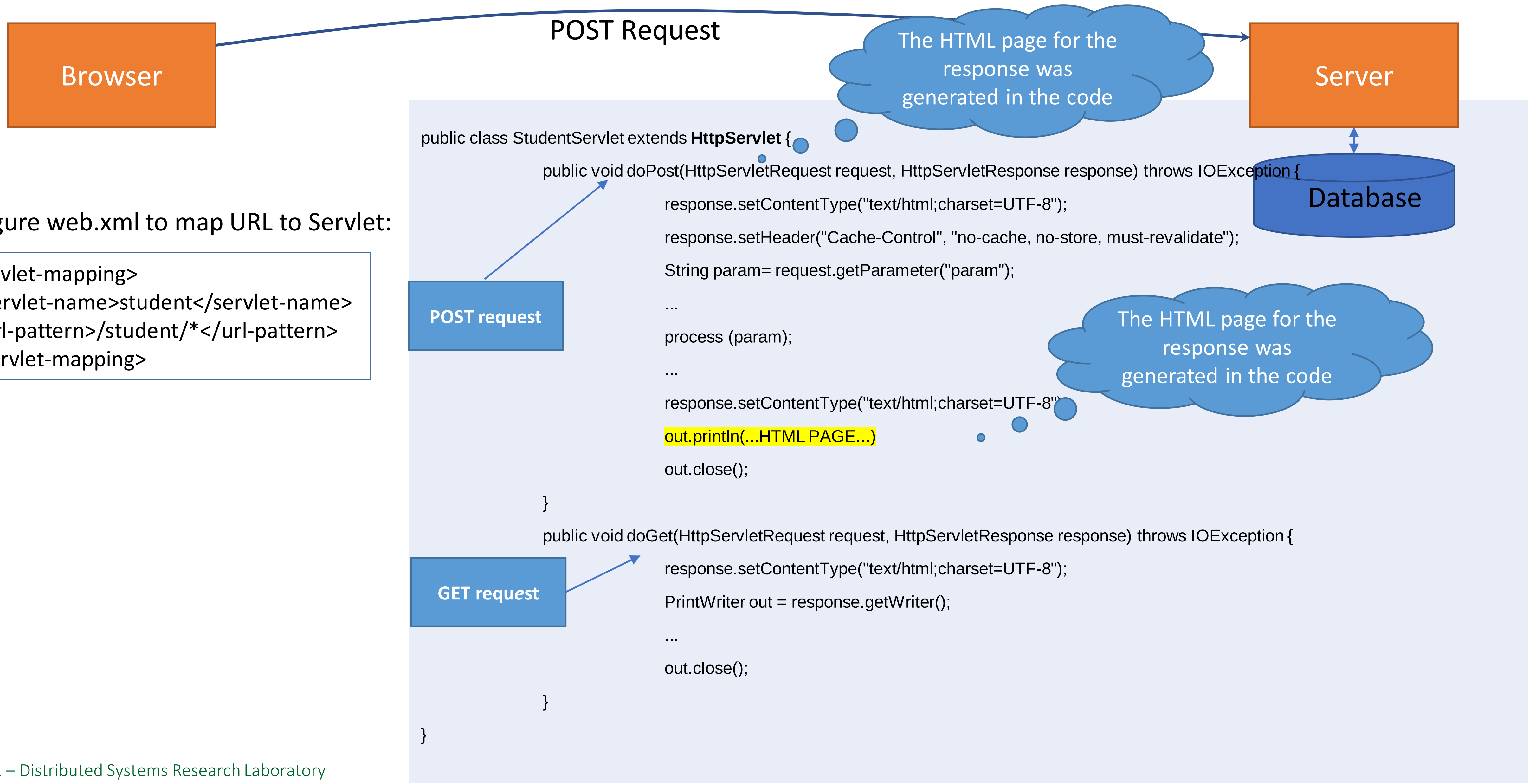
```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

```
public abstract class AbstractServlet {
    private static final Log LOGGER = LogFactory.getLog(AbstractServlet.class);

    public String doRequest(RequestMessage message) {
        try {
            switch (message.getMethod()) {
                case Find:
                    return find(message);
                case Save:
                    return save(message);
                default:
                    return ResponseMessageEncoder.encode(StatusCode.BAD_REQUEST,null);
            }
        } catch (UnsupportedOperationException e) {
            LOGGER.error("", e);
            return ResponseMessageEncoder.encode(StatusCode.NOT_ALLOWED,null);
        }
    }
    public abstract String doPost(RequestMessage message);
    public abstract String doGet(RequestMessage message);
}
```

After processing requests return a String that contains the HTML page

Handling HTTP requests – Java Servlets

How is the HTTP protocol implemented in an application?


POST Request
Browser
Server
Database
The HTML page for the response was generated in the code
POST request
The HTML page for the response was generated in the code
GET request

```
public class StudentServlet extends HttpServlet {
        public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
                response.setContentType("text/html;charset=UTF-8");
                response.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
                String param= request.getParameter("param");
                ...
                process (param);
                ...
                response.setContentType("text/html;charset=UTF-8");
                out.println(...HTML PAGE...)
                out.close();
        }
        public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
                response.setContentType("text/html;charset=UTF-8");
                PrintWriter out = response.getWriter();
                ...
                out.close();
        }
}
```

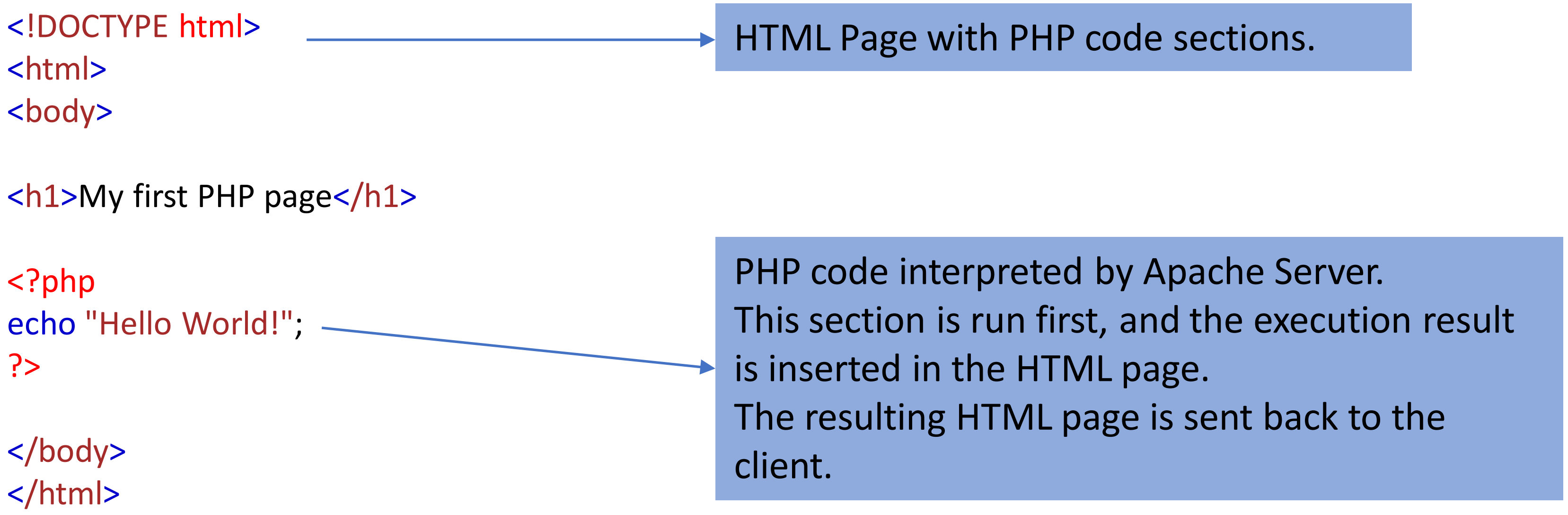
Configure web.xml to map URL to Servlet:

```
<servlet-mapping>
  <servlet-name>student</servlet-name>
  <url-pattern>/student/*</url-pattern>
</servlet-mapping>
```

Handling HTTP requests – Technology Evolution

- Java Servlet Technology is cumbersome since HTML code for rendering the response page needed to be generated within the code
- Next generation technologies allowed to write code inside HTML pages – the server initially executed the code and inserted the execution result in the HTML page
- Examples: PHP+HTML, Java JSP, ASP .NET, etc.

PHP+HTML



```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

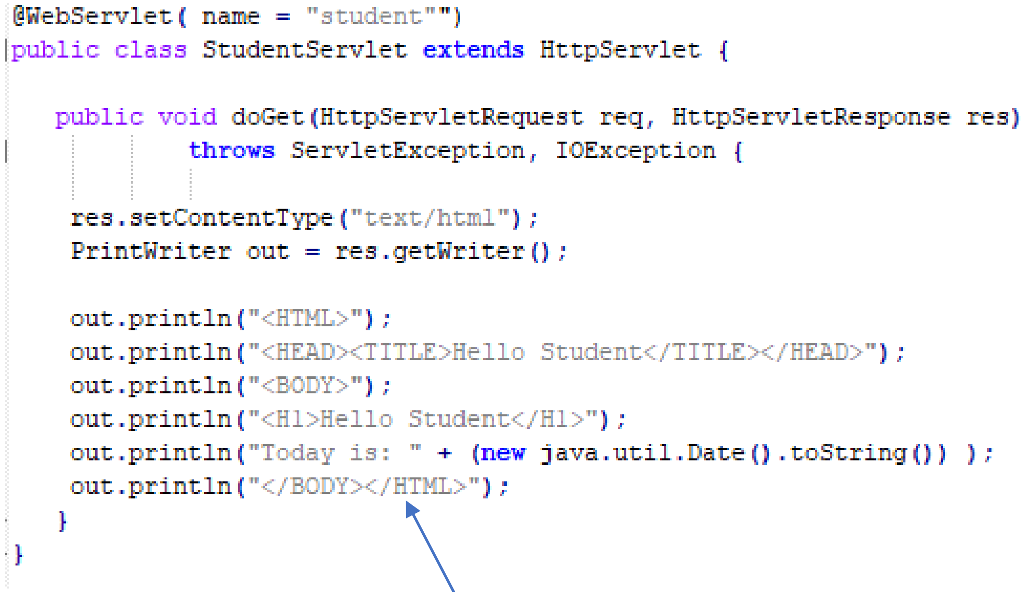
HTML Page with PHP code sections.

PHP code interpreted by Apache Server.
This section is run first, and the execution result is inserted in the HTML page.
The resulting HTML page is sent back to the client.

Handling HTTP requests – Technology Evolution

JAVA Servlet vs Java Server Pages (JSP)

- Servlet

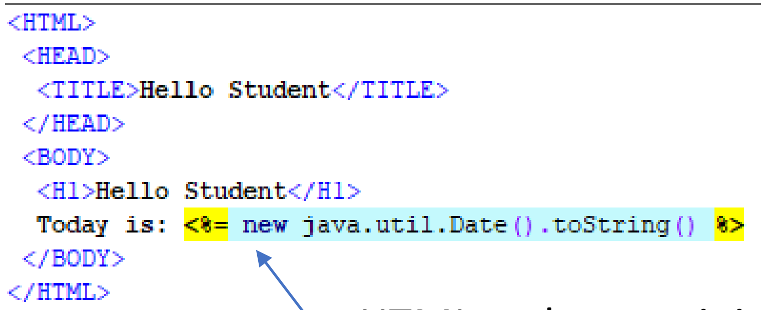


```
@WebServlet( name = "student"")
public class StudentServlet extends HttpServlet {

   public void doGet(HttpServletRequest req, HttpServletResponse res)
             throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<HTML>");
    out.println("<HEAD><TITLE>Hello Student</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<H1>Hello Student</H1>");
    out.println("Today is: " + (new java.util.Date().toString()) );
    out.println("</BODY></HTML>");
   }
}
```

Java code containing HTML insertion

Does not provide HTML compilation

- JSP



```
<HTML>
 <HEAD>
  <TITLE>Hello Student</TITLE>
 </HEAD>
 <BODY>
  <H1>Hello Student</H1>
  Today is: <%= new java.util.Date().toString() %>
 </BODY>
</HTML>
```

HTML code containing Java insertion

Cumbersome to use

Handling HTTP requests – Technology Evolution

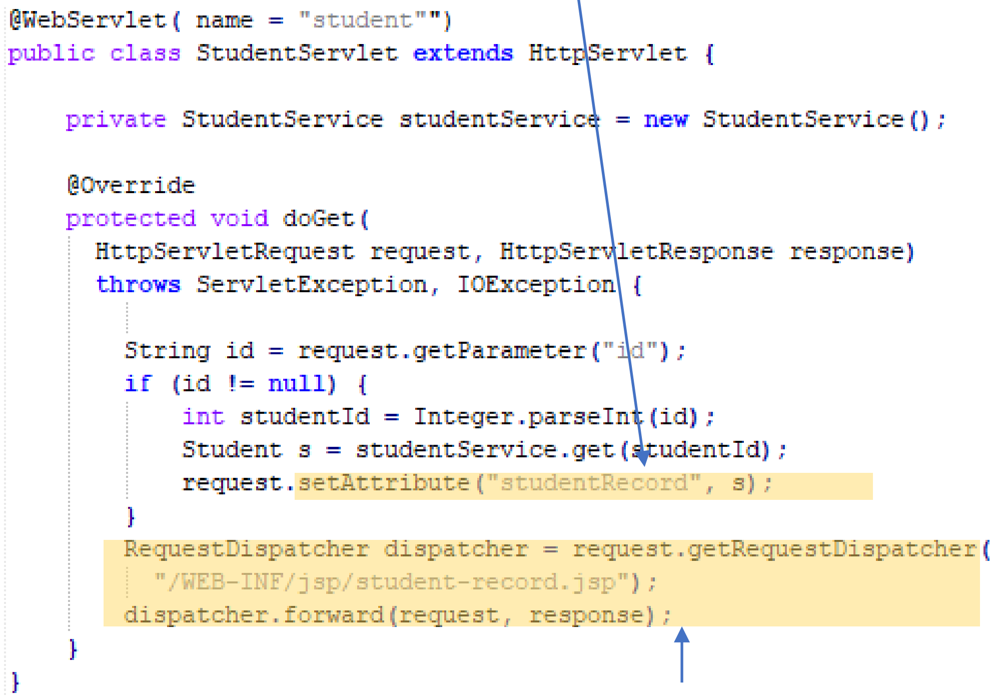
- This led to unstructured code, since frontend definition and operations where in the same code.
- Thus, next generation frameworks split the frontend definition from the operations, mostly using MVC architectural patterns
- Examples: C# MVC, Java Spring MVC, etc.

Handling HTTP requests – Technology Evolution

Servlet and JSP MVC architecture

- We can have the benefits of both Servlet & JSP by using MVC pattern

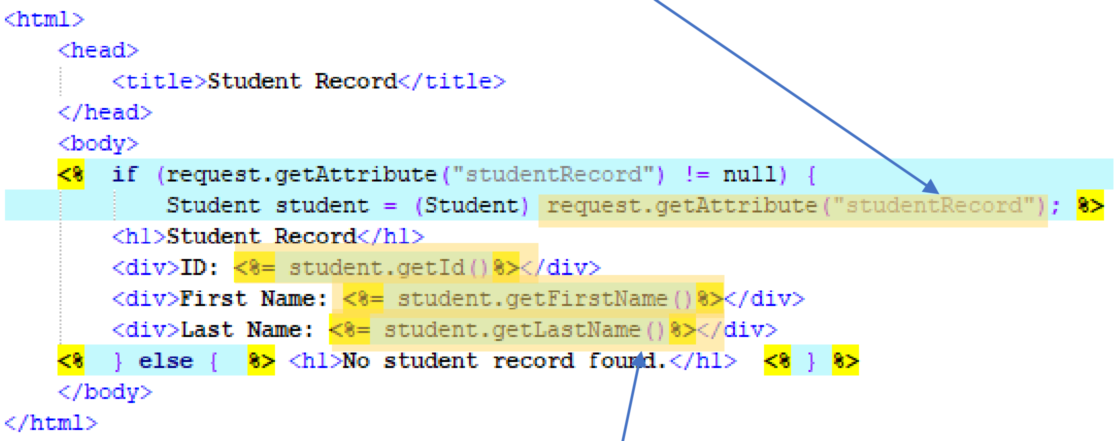
1) Set an attribute (key-value pair) on the request object in the servlet code



```
@WebServlet( name = "student"")
public class StudentServlet extends HttpServlet {

    private StudentService studentService = new StudentService();

    @Override
    protected void doGet(
      HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

        String id = request.getParameter("id");
        if (id != null) {
            int studentId = Integer.parseInt(id);
            Student s = studentService.get(studentId);
            request.setAttribute("studentRecord", s);
        }
        RequestDispatcher dispatcher = request.getRequestDispatcher(
          "/WEB-INF/jsp/student-record.jsp");
        dispatcher.forward(request, response);
    }
}
```

2) Forward the request to the JSP page

3) Get the element by key in the JSP file



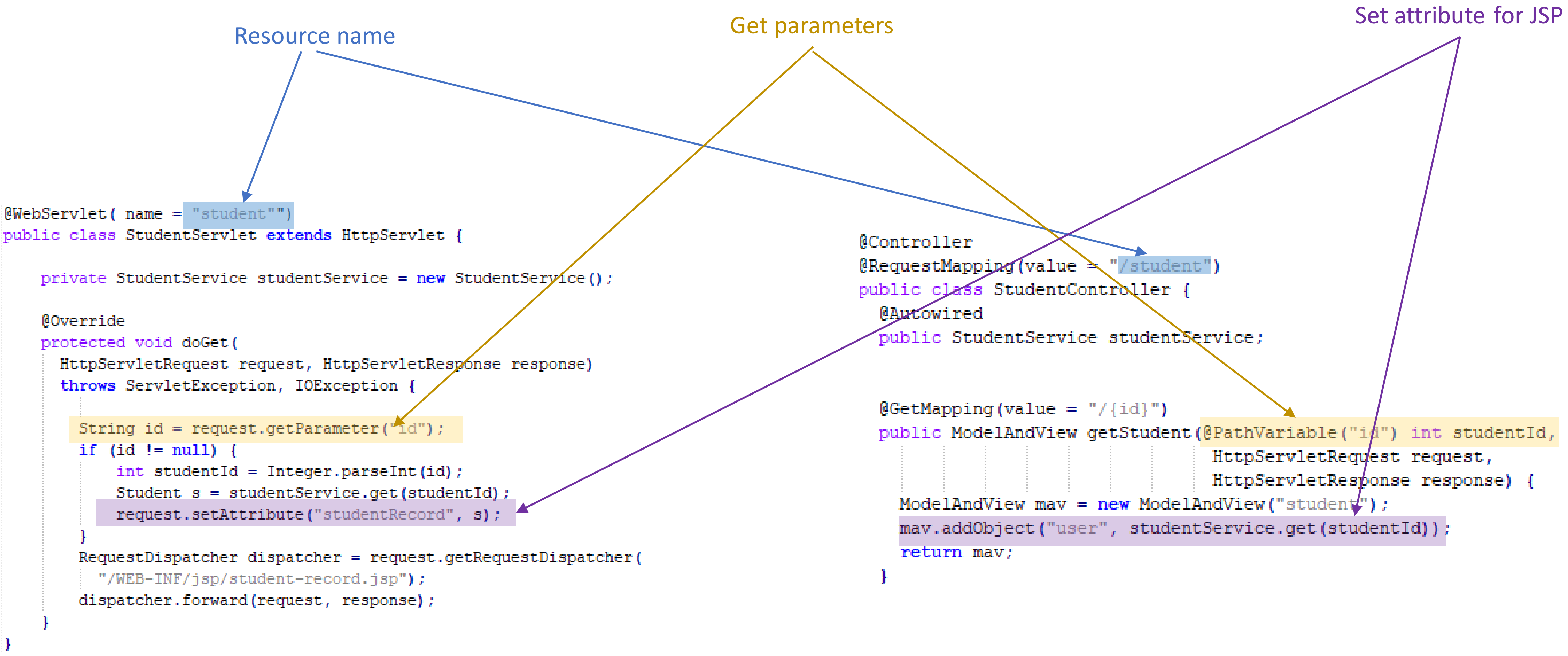
```
<html>
    <head>
        <title>Student Record</title>
    </head>
    <body>
    <%  if (request.getAttribute("studentRecord") != null) {
            Student student = (Student) request.getAttribute("studentRecord"); %>
        <h1>Student Record</h1>
        <div>ID: <%= student.getId()%></div>
        <div>First Name: <%= student.getFirstName()%></div>
        <div>Last Name: <%= student.getLastName()%></div>
    <%  } else {  %> <h1>No student record found.</h1>  <% } %>
    </body>
</html>
```

4) Populate the HTML with the obtain object information

Handling HTTP requests – Technology Evolution

Servlet vs Spring MVC

- Progress from Servlets + JSP to Spring MVC (with JSP)



Resource name

Get parameters

Set attribute for JSP

```
@WebServlet( name = "student"")
public class StudentServlet extends HttpServlet {

    private StudentService studentService = new StudentService();

    @Override
    protected void doGet(
      HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

        String id = request.getParameter("id");
        if (id != null) {
            int studentId = Integer.parseInt(id);
            Student s = studentService.get(studentId);
            request.setAttribute("studentRecord", s);
        }
        RequestDispatcher dispatcher = request.getRequestDispatcher(
          "/WEB-INF/jsp/student-record.jsp");
        dispatcher.forward(request, response);
    }
}
```

Servlets

```
@Controller
@RequestMapping(value = "/student")
public class StudentController {
  @Autowired
  public StudentService studentService;


  @GetMapping(value = "/{id}")
  public ModelAndView getStudent(@PathVariable("id") int studentId,
                                 HttpServletRequest request,
                                 HttpServletResponse response) {
    ModelAndView mav = new ModelAndView("student");
    mav.addObject("user", studentService.get(studentId));
    return mav;
  }
```

Spring MVC

Handling HTTP requests – Technology Evolution

- All the previous technologies rendered the page on **the server side!**
- Heavy network traffic, lots of media content and responsive pages lead to the need of **client-side processing**
- Modern applications have a server-side backend and a client-side application, either mobile or JavaScript-based for browsers (JavaScript – enables applications to be run inside the browsers)
- Server-Side backend is exposed as Web Services or RPC
- Examples:

Backend	Frontend
Java Spring REST	Angular
C# WEB API	ReactJS
NodeJS	Android
…	…

Handling HTTP requests – REST API

- **Representational state transfer** (**REST**)
- Software architecture
- The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation
- Access resources based on URLs/URIs
- Based on HTTP – use different HTTP methods

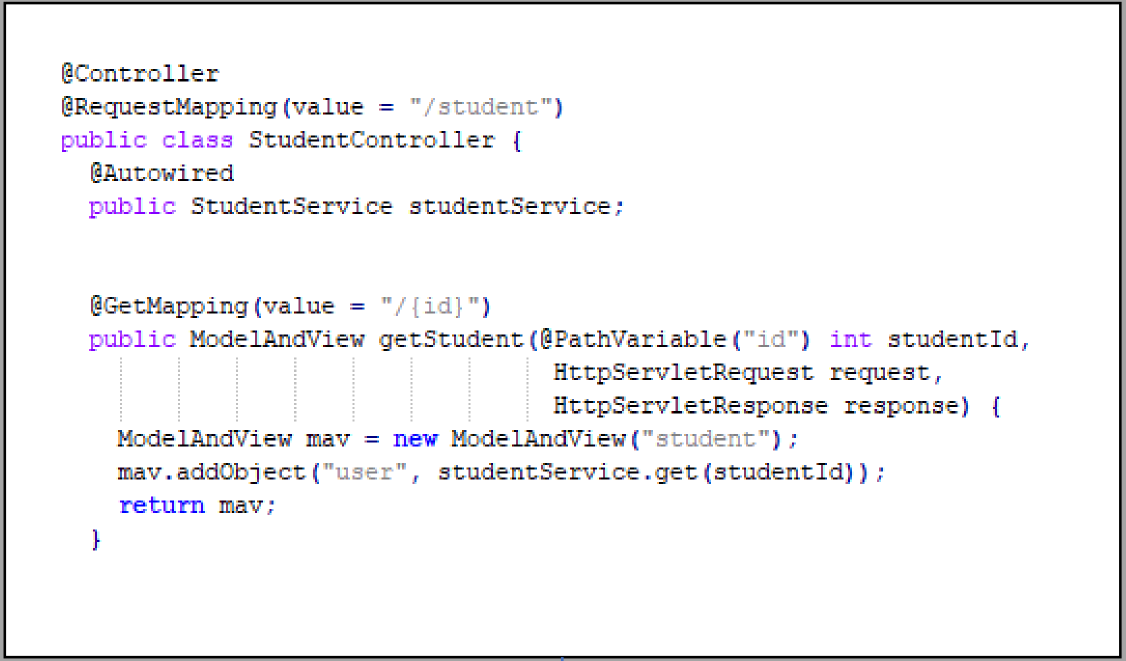
@GET

http://IP:port/applicationName/resourceName/resourceMethod/{parameters}

Handling HTTP requests – REST API

Spring MVC vs Spring REST

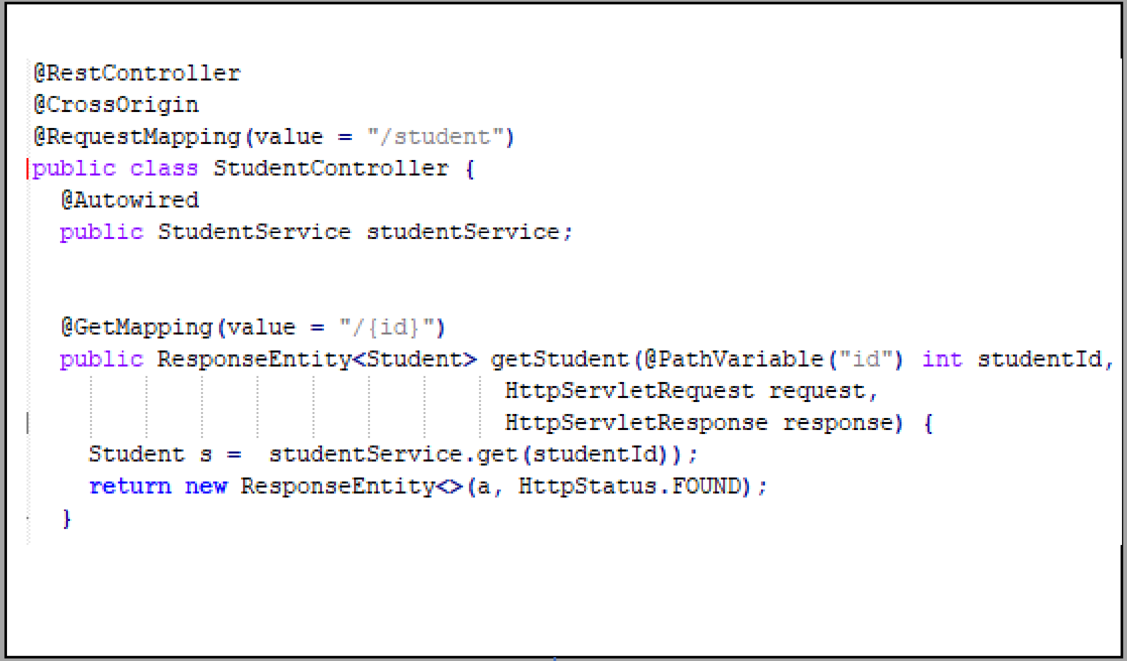
Spring MVC Controller



```
@Controller
@RequestMapping(value = "/student")
public class StudentController {
  @Autowired
  public StudentService studentService;


  @GetMapping(value = "/{id}")
  public ModelAndView getStudent(@PathVariable("id") int studentId,
                                 HttpServletRequest request,
                                 HttpServletResponse response) {
    ModelAndView mav = new ModelAndView("student");
    mav.addObject("user", studentService.get(studentId));
    return mav;
  }
```

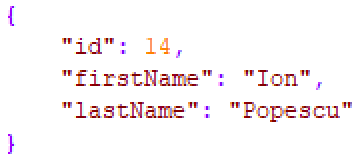
Redirects to JSP and the server returns a HTML

Spring Rest Controller



```
@RestController
@CrossOrigin
@RequestMapping(value = "/student")
public class StudentController {
  @Autowired
  public StudentService studentService;


  @GetMapping(value = "/{id}")
  public ResponseEntity<Student> getStudent(@PathVariable("id") int studentId,
                                            HttpServletRequest request,
                                            HttpServletResponse response) {
    Student s =  studentService.get(studentId);
    return new ResponseEntity<>(a, HttpStatus.FOUND);
  }
```

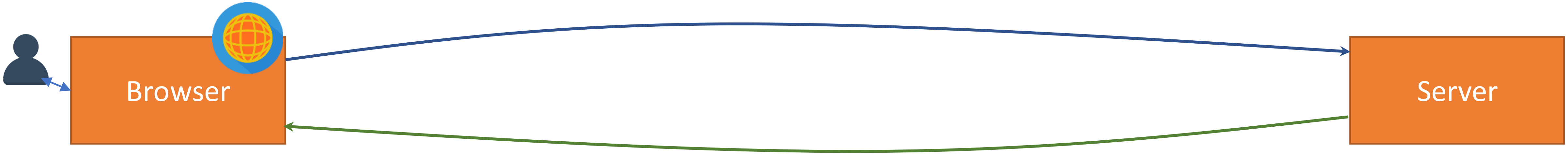
The server returns a JSON to be used by any Client App



```
{
    "id": 14,
    "firstName": "Ion",
    "lastName": "Popescu"
}
```

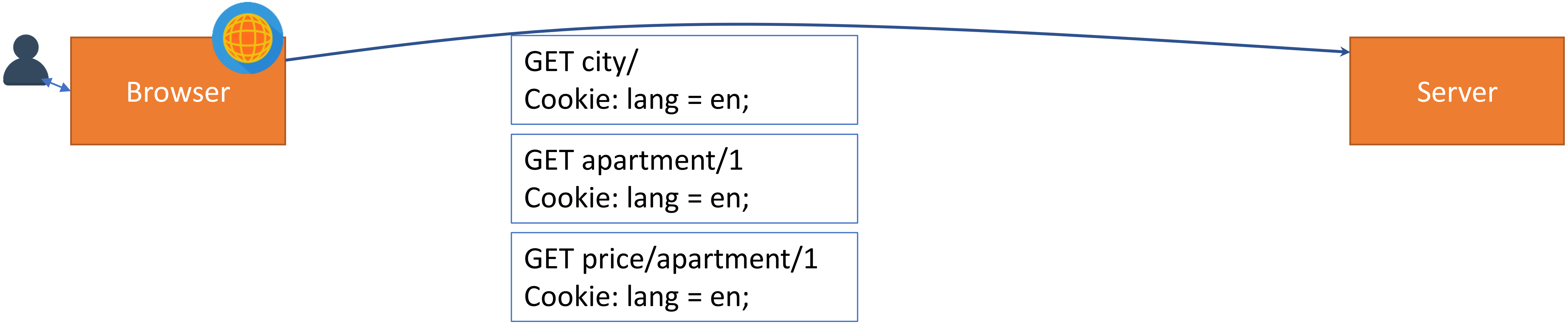
Handling HTTP requests > Providing State

- Tom wants uses a Book-Apartments application online
- When he first registers, he selects the English language
  - Q: How can the server know to respond each time in English for the apartments' description?



Handling HTTP requests > Providing State

- Tom wants uses a Book-Apartments application online
- When he first registers, he selects the English language
  - Q: How can the server know to respond each time in English for the apartments' description?
  - A: Specify the language for each request.



GET city/
Cookie: lang = en;

GET apartment/1
Cookie: lang = en;

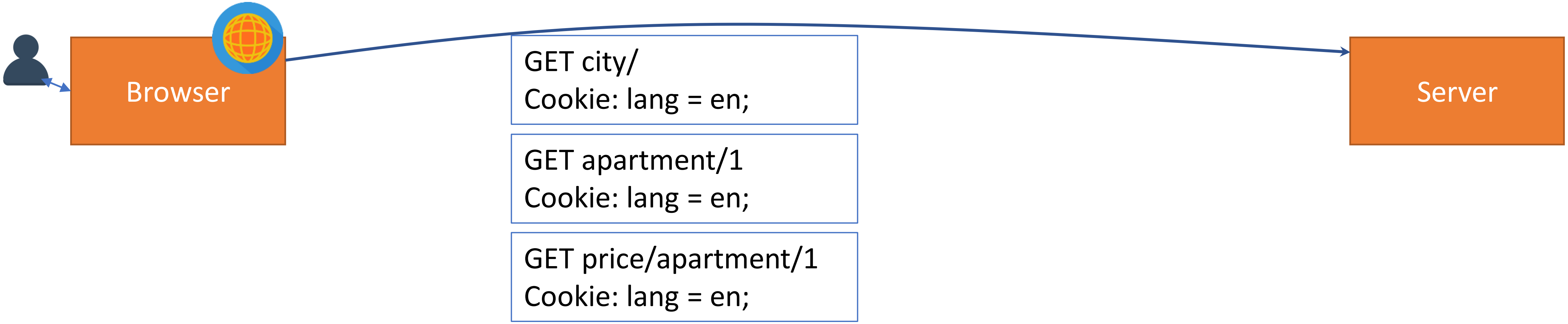
GET price/apartment/1
Cookie: lang = en;

- **HTTP cookie** is a small piece of data stored on the user's computer by the web browser

*https://en.wikipedia.org/wiki/HTTP_cookie

Handling HTTP requests > Providing State

- Q: What happens if Tom accidently closes the browser? Should he select all the preferences again?
- A: No. The cookies are stored in a file on the user's computer. The browser will automatically load the preferences whenever accessing the same host server.



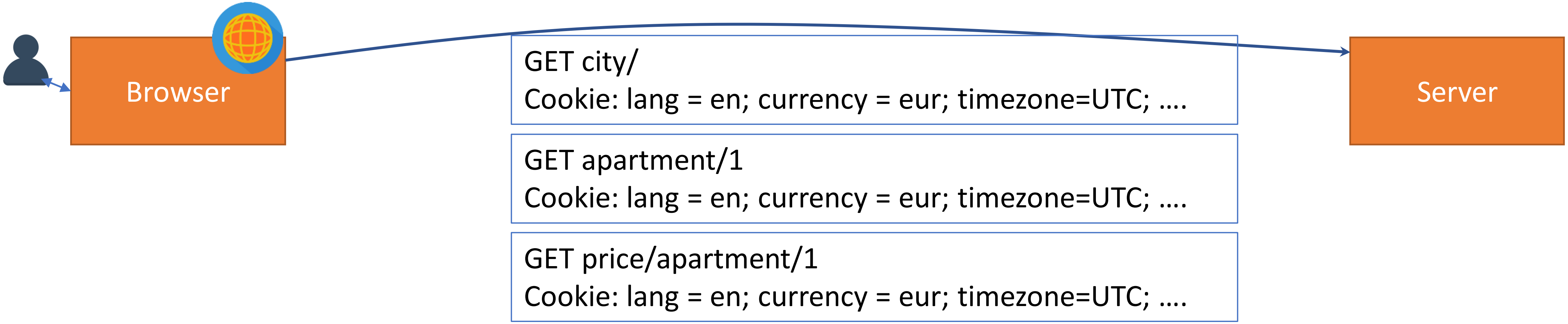
GET city/
Cookie: lang = en;

GET apartment/1
Cookie: lang = en;

GET price/apartment/1
Cookie: lang = en;

Handling HTTP requests > Providing State

- Q: What if Tom has more preferences ?
  - Currency
  - Language
  - Etc.
- A: You can add several cookies.



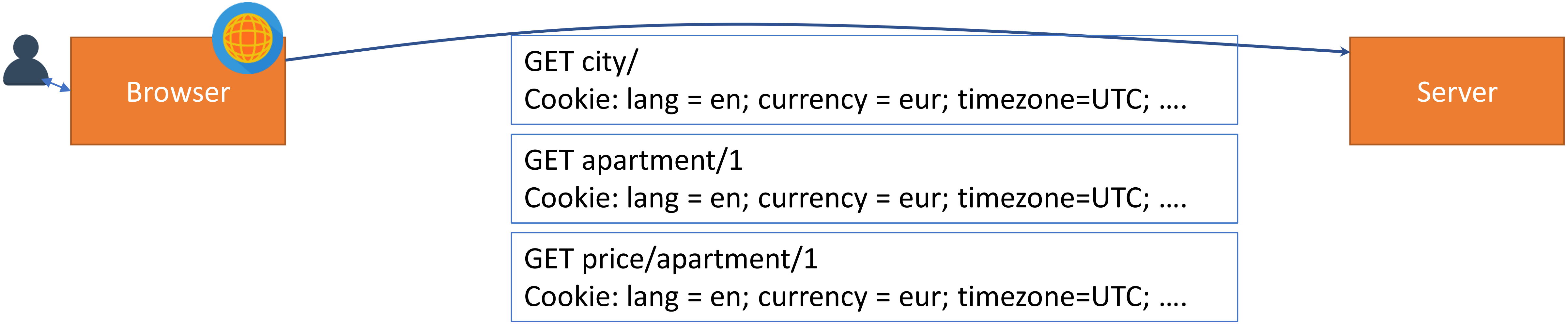
```
GET city/
Cookie: lang = en; currency = eur; timezone=UTC; ….
```

```
GET apartment/1
Cookie: lang = en; currency = eur; timezone=UTC; ….
```

```
GET price/apartment/1
Cookie: lang = en; currency = eur; timezone=UTC; ….
```

Handling HTTP requests > Providing State

- Q: How many preferences can be sent over the request ? Is there a limit?
- A: There is a limit regarding the cookies, depending on the browser it is recommended to use maximum **50 cookies per domain**, and **4093 bytes per domain**



GET city/
Cookie: lang = en; currency = eur; timezone=UTC; ….

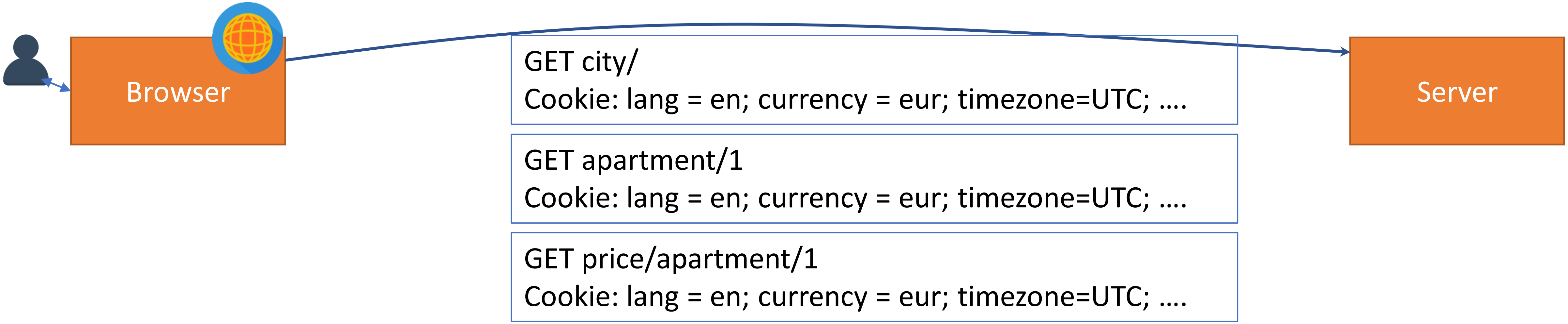
GET apartment/1
Cookie: lang = en; currency = eur; timezone=UTC; ….

GET price/apartment/1
Cookie: lang = en; currency = eur; timezone=UTC; ….

- Q: What can we do if we have to store more information? (e.g. shopping cart info)
- A: Move to HTTP Session.

Handling HTTP requests > Providing State

- Q: Are there security risks regarding the elements stored in cookies?
- A: The cookies can be easily modified from the user side, thus for security reasons sever-side storage can be considered using HTTP Sessions.



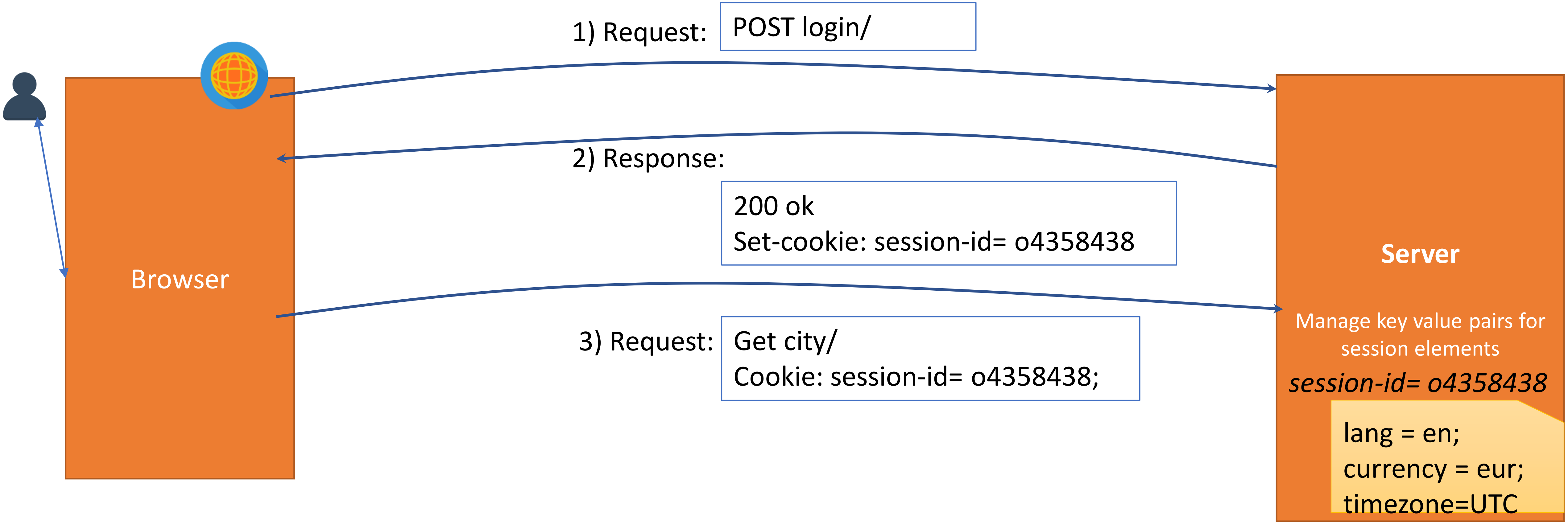
GET city/
Cookie: lang = en; currency = eur; timezone=UTC; ….

GET apartment/1
Cookie: lang = en; currency = eur; timezone=UTC; ….

GET price/apartment/1
Cookie: lang = en; currency = eur; timezone=UTC; ….

Handling HTTP requests > Providing State

- **HTTP Session –** data stored by the server application regarding the client interactions on the server computer ; identified by session ids.



1) Request:

POST login/

2) Response:

200 ok
Set-cookie: session-id= o4358438

3) Request:

Get city/
Cookie: session-id= o4358438;

session-id= o4358438

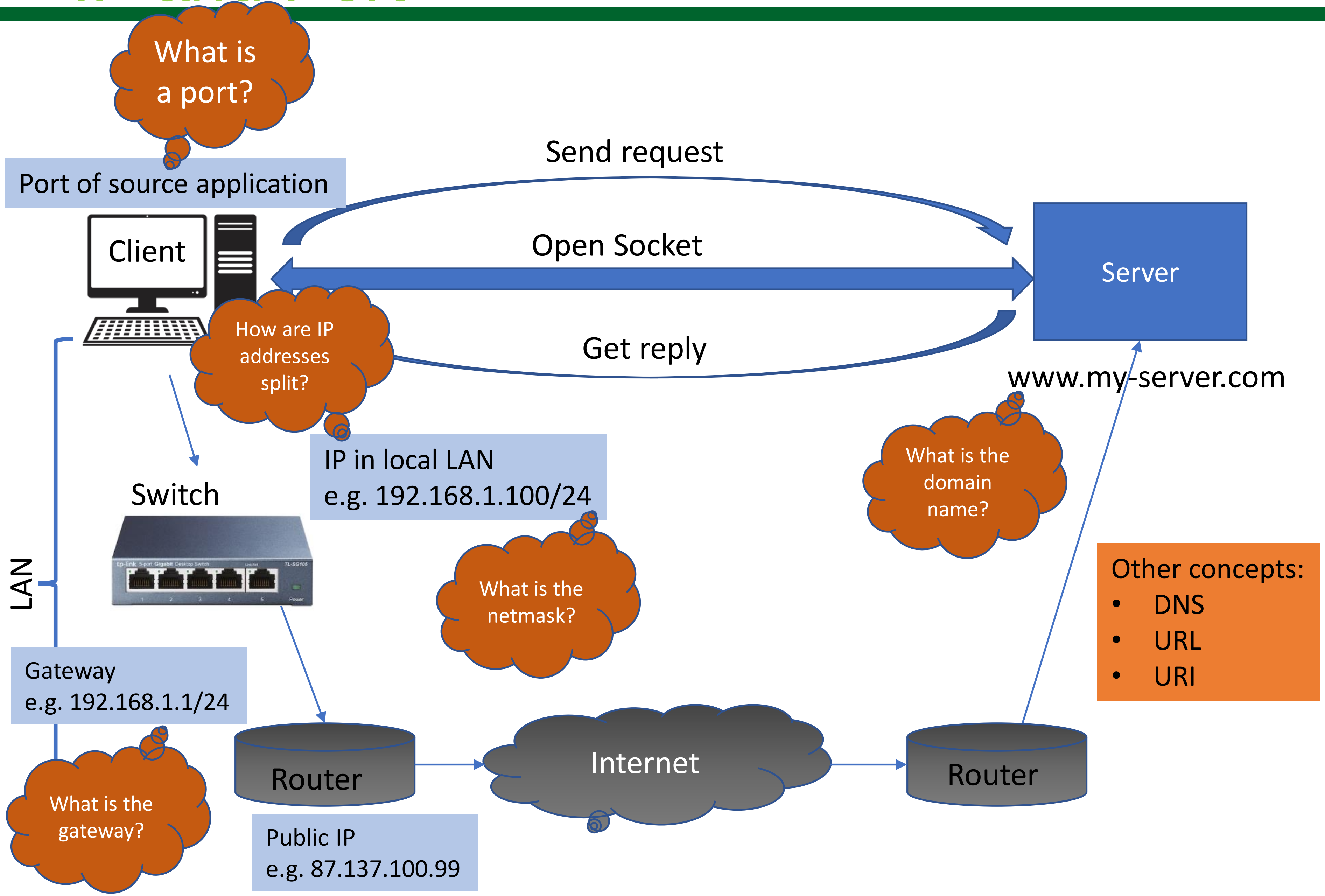
lang = en;
currency = eur;
timezone=UTC

Conclusion

Features	Method Call		HTTP	Web Service	
	Local	Remote		REST	SOA
Addressing	Name	IP:port/name	URL (port 80)	URL:port	URL:port
Parameters	Value or reference	Value	Value in URL or HTTP body	Value in URL (GET) or HTTP body (POST)	Value in HTTP body (POST method)
Signature	Interface	Interface	8 HTTP methods	REST API endpoints	WSDL
Garbage collector	Local	Distributed	Server responsible	Server responsible	Server responsible

Theoretical Background

TCP/IP > *IP and Port*



What is a port?

Port of source application

Client

Send request

Open Socket

Get reply

Server

www.my-server.com

How are IP addresses split?

IP in local LAN
e.g. 192.168.1.100/24

Switch

LAN

What is the domain name?

What is the netmask?

Gateway
e.g. 192.168.1.1/24

What is the gateway?

Router

Internet

Router

Public IP
e.g. 87.137.100.99

Other concepts:

- DNS
- URL
- URI

TCP/IP > *URI vs URL*

Uniform Resource Identifier (URI): sequence of characters allowing the complete identification of any abstract/physical resource

Uniform Resource Locator (URL): subset of URI that, in addition to identifying where a resource is available, describes the primary mechanism to access it

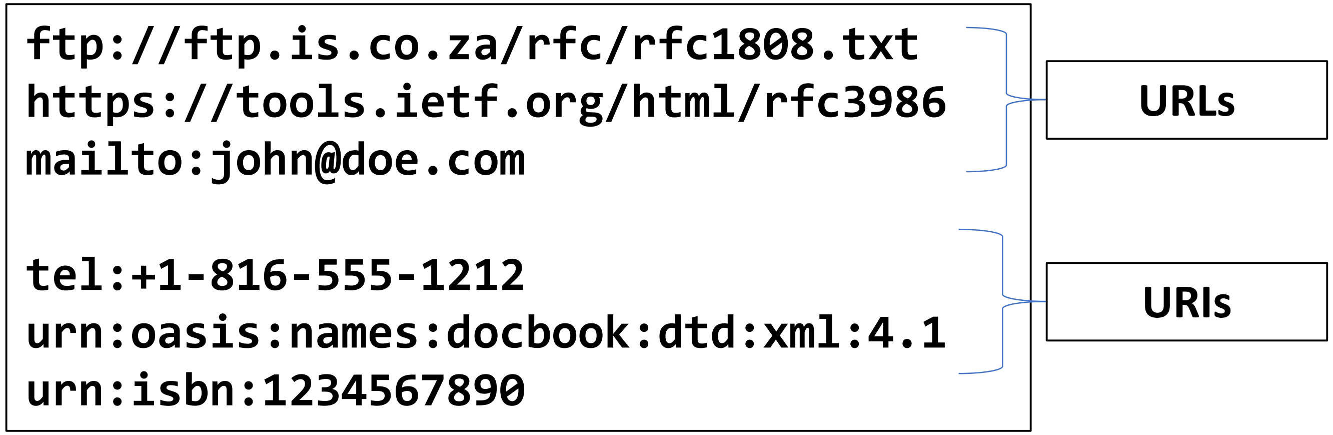
- **URI Syntax**

```
scheme:[//authority][/path][?query][#fragment]
```

- **scheme** – for URLs is the name of the protocol used to access the resource (e.g. http for Web sites)
- **authority** - an optional part comprised of user authentication information, a host and an optional port
- **path** − it serves to identify a resource within the scope of its scheme and authority
- **query** – additional data that, along with the path, serves to identify a resource (e.g. for URLs, this is the query string)
- **fragment** − an optional identifier to a specific part of the resource

TCP/IP > *URI vs URL*

- **How to differentiate between URI and URL**
  - **Check the scheme!**
  - Every URL has to start with any of these schemes: *ftp, http, https, gopher, mailto, news, nntp, telnet, wais, file, or prospero*
- **Examples**

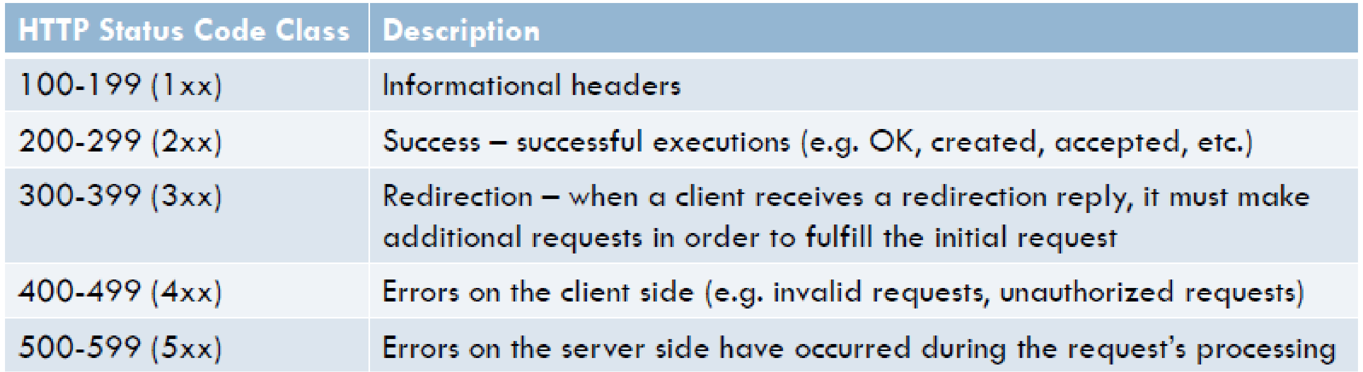


```
ftp://ftp.is.co.za/rfc/rfc1808.txt
https://tools.ietf.org/html/rfc3986
mailto:john@doe.com

tel:+1-816-555-1212
urn:oasis:names:docbook:dtd:xml:4.1
urn:isbn:1234567890
```

HTTP protocol > Message Structure

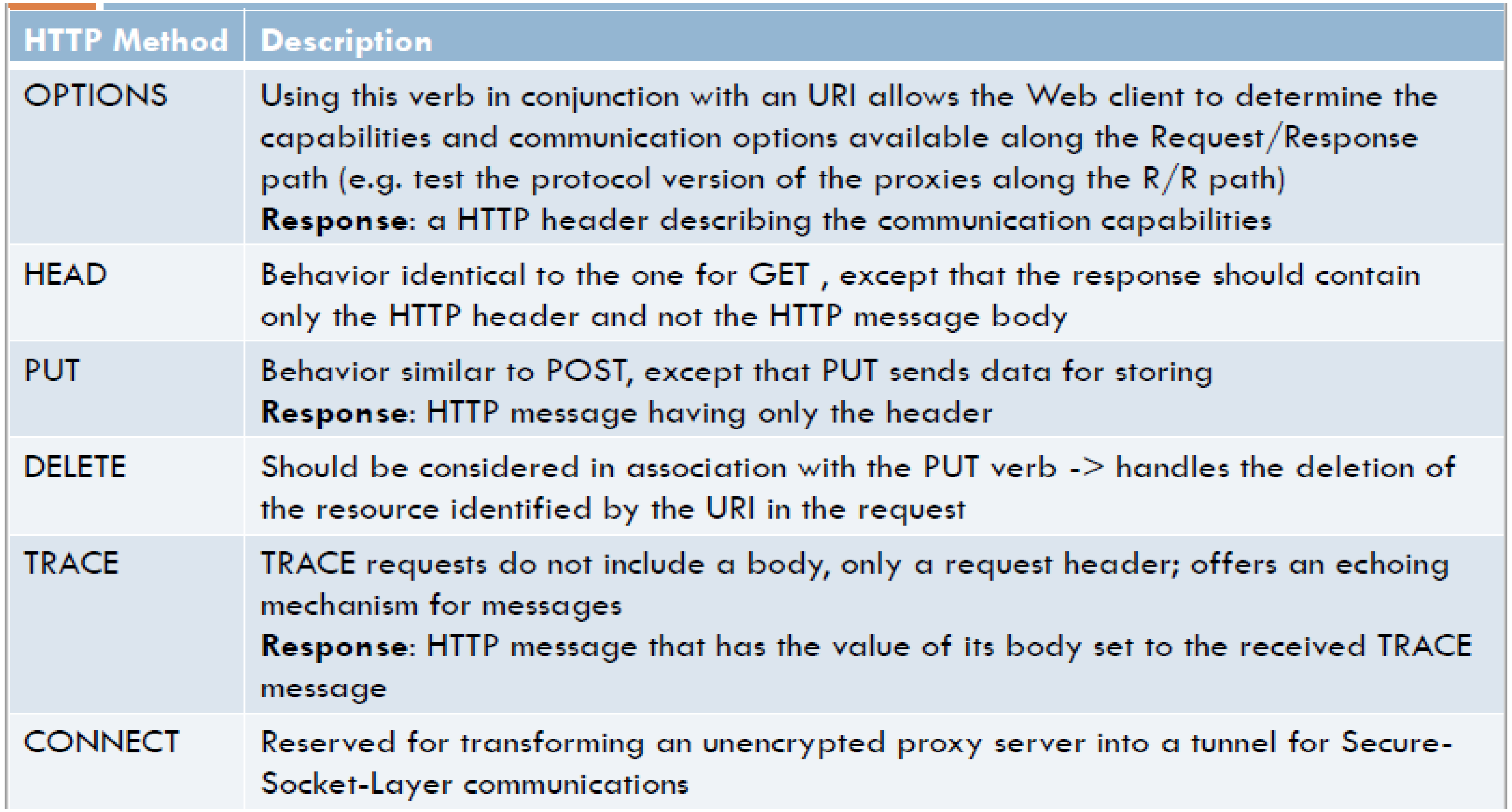
HTTP Protocol:

- Stateless Protocol
- Define a section of <Header> for the messages
- Define a section of <Body> for the messages
- Define 8 method types (key-words): GET, POST, OPTIONS, HEAD, PUT, DELETE, TRACE, CONNECT
- Each method is interpreted in a different manner on server side, as an **agreement/contract** between the programmers who implement the client applications and the server application
- A set of codes is defined to include in the server responses



HTTP Status Code Class	Description
100-199 (1xx)	Informational headers
200-299 (2xx)	Success – successful executions (e.g. OK, created, accepted, etc.)
300-399 (3xx)	Redirection – when a client receives a redirection reply, it must make additional requests in order to fulfill the initial request
400-499 (4xx)	Errors on the client side (e.g. invalid requests, unauthorized requests)
500-599 (5xx)	Errors on the server side have occurred during the request's processing

HTTP protocol > HTTP Methods



HTTP Method	Description
OPTIONS	Using this verb in conjunction with an URI allows the Web client to determine the capabilities and communication options available along the Request/Response path (e.g. test the protocol version of the proxies along the R/R path) **Response**: a HTTP header describing the communication capabilities
HEAD	Behavior identical to the one for GET , except that the response should contain only the HTTP header and not the HTTP message body
PUT	Behavior similar to POST, except that PUT sends data for storing **Response**: HTTP message having only the header
DELETE	Should be considered in association with the PUT verb -> handles the deletion of the resource identified by the URI in the request
TRACE	TRACE requests do not include a body, only a request header; offers an echoing mechanism for messages **Response**: HTTP message that has the value of its body set to the received TRACE message
CONNECT	Reserved for transforming an unencrypted proxy server into a tunnel for Secure-Socket-Layer communications

HTTP protocol > GET vs POST

- Current frameworks implement the following standards as responses for GET or POST methods
- It is just an **agreement** between the programmers, both GET and POST are **just Keywords**

GET

- Request for information located at a specified URI on the server
- The entity-body portion of a GET request is always empty
- The GET method is also used to send input - the input data is appended to the URL in the GET line of the request (query strings)

POST

- Allows data to be sent to the server in a client request - the data is directed to a data-handling program that the server has access
- The data sent to the server is in the entity-body section of the client's request

HTTP protocol > Providing state

- **Cookies**
  - Key-value pairs associated with a URI
  - The server attaches a cookie to the reply it sends to the client; the client stores the cookie locally and then re-attaches the cookie on all subsequent queries to the URI for which the cookie was associated
- **Hidden variables**
  - Types of HTML Form elements without graphical representation
  - Are present in the HTML documents, they are not displayed but they are sent to the Web server
  - Similarly, a Web server can put data in the HTML hidden variables
- **Query strings**
  - Data can be sent between pages as part of the URL encoding
  - Start with “?” and are followed by a series of key and value pairs delimited by “&”
  - E.g.: http://www.google.com/search?sitesearch=www.w3schools.com&as_q=query+string
- **Server-side sessions**
  - Shared resources that are available for reading and writing to all the server processes that handle requests originating from the same client (browser) during a specified time-span

- Checkout Cookies vs Sessions : https://www.guru99.com/difference-between-cookie-session.html

HTTP protocol > Java Servlets (1)

- **Definition**
  - Small components that are registered on the Web Server to process HTTP requests and that send a response to those requests, typically in HTML or XML format
  - Java objects that inherit from the **GenericServlet** class defined in the javax.servlet package
- Their most common use is in connection with HTTP => the **HTTPServlet** class is used as a template for creating HTTP based Servlets

HTTP protocol > Java Servlets (2)

- The HTTPServlet class defines a set of methods
  - **init**()
  - **destroy**()

    Can be overwritten to manage the resources held by the HttpServlet during its lifetime; the initialization and destruction processes occur only once for each Servlet
  - **service**() – implements the Service Dispatcher logic
    - Looks at the request method specified in the HTTP request's header and then invokes one of the following methods
      - **doPost**()
      - **doGet**()
      - **doPut**()

        Most used
      - **doDelete**()
      - **doOptions**()
      - **doHead**()
      - **doTrace**()

HTTP protocol > Java Servlets (3)

- Types of parameters for the handlers of the HTTP methods
  - **HttpServletRequest**
  - **HttpServletResponse**
    - Used by the Servlet to “print” the HTML or XML code that will be sent to the client (browser)
- Servlet configuration
  - Can be specified at the Servlet level or at the Application level
  - The Servlet specific information is held by a **ServletConfig** object
  - The configuration for all the Servlets of an application is held by a **ServletContext** object
  - All **ServletConfig** objects hold a reference to the **ServletContext** of the application to which the Servlet belongs

HTTP protocol > Java Servlets (4)

- The running environment (Servlet Container) - responsible for
  - Initializing the Servlet
  - Passing the HTTP requests to the Servlet
  - Receiving the HTTP responses from the Servlet and forwarding them to the client
- Advantages
  - Powerfull technology that allows developers to handle resource requests on the server side using the full potential of the Java platform
- Disadvantages
  - Require many “print” statements to generate the HTML /XML response
  - High development time due to lack of compile-time verification of the HTML/XML generated code