HTTP/Servlet-based Distributed Systems

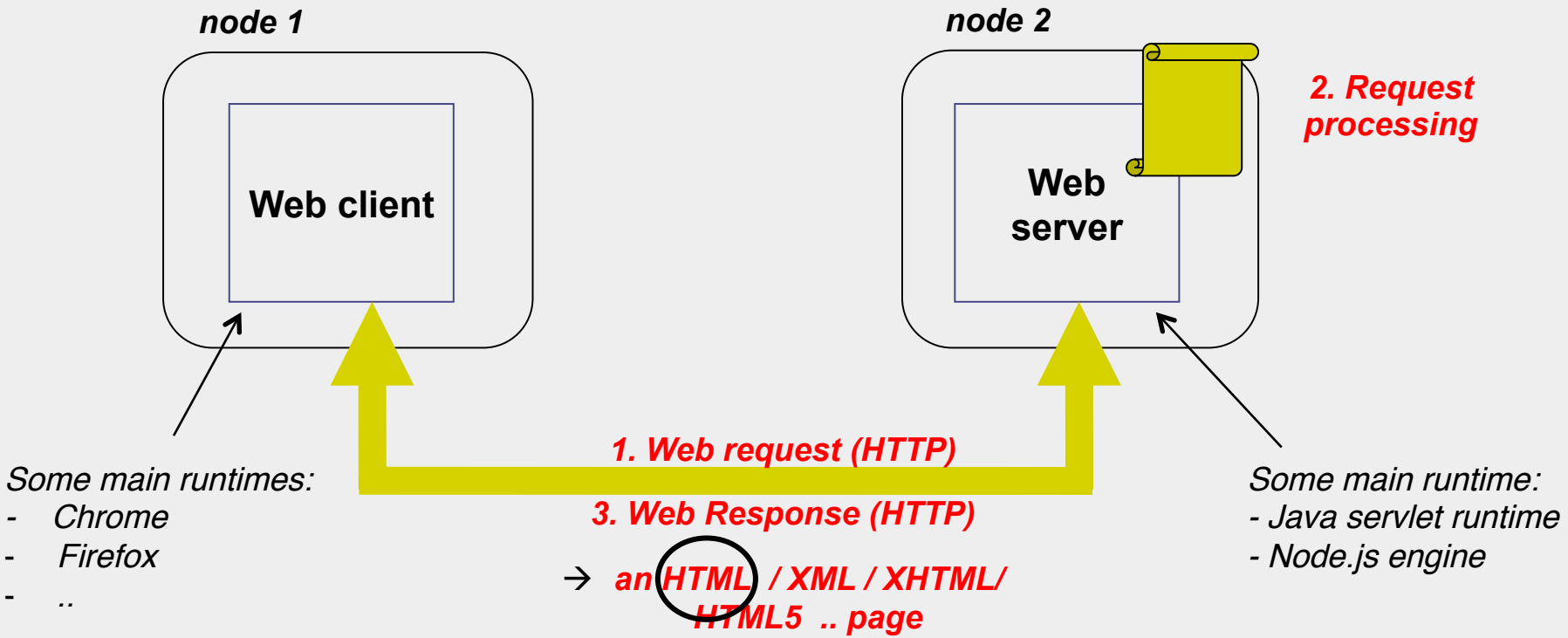
Polytech/INFO 4, 2022-2023
Fabienne Boyer, Olivier Gruber,
UFR IM2AG, LIG, Université Grenoble Alpes
Fabienne.Boyer@imag.fr

Outline

- Web architecture
- HTTP principles
- Servlets principles
- Work to do

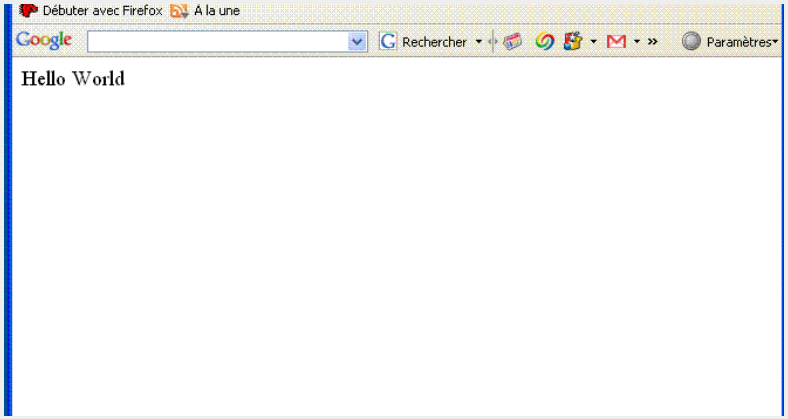
Web Architecture

Client-Server, HTTP protocol



A very simple HTML page

- A returned page may range from very simple HTML to complex pages including code processed at client side (animations, drawings, ..)




```
<HTML>
        <HEAD> <TITLE> Hello World </TITLE> </HEAD>
        <BODY> <BIG> Hello World </BIG> </BODY>
</HTML>
```

Focus of the lecture

- **Web Server programming**
  - Back-end
    - May be complex from a functional point of view
  - Servlet model
    - A commonly used solution, provides an elaborated sdk
  - Practical work
    - Install, launch, manage an Apache Tomcat servlet server
- **HTTP Server design & programming**
  - Understanding the main challenges
    - Session data, isolation, ..
  - Practical work
    - Implement your proper HTTP server

Web Architecture

- **Client-Server**
  - HTTP (Hyper Text Transfer **Protocol**), upon TCP/IP on port 80
  - Basically, a new connection per request
- **HTTP Requests**
  - Allow to access a resource on the server
    - a *file* or a *directory* (static requests)
    - a *script* or a *program* to run (dynamic requests)
  - The resource is identified and located using an URL
    - http://serverhost/index.html (static request)
    - http://serverhost/program?arg1=val1&arg2=val2 (dynamic request)
- **Dynamic requests are useful for a number of reasons**
  - The returned page is based on data submitted by the user
  - The returned page contains information from databases or other sources

Web Architecture

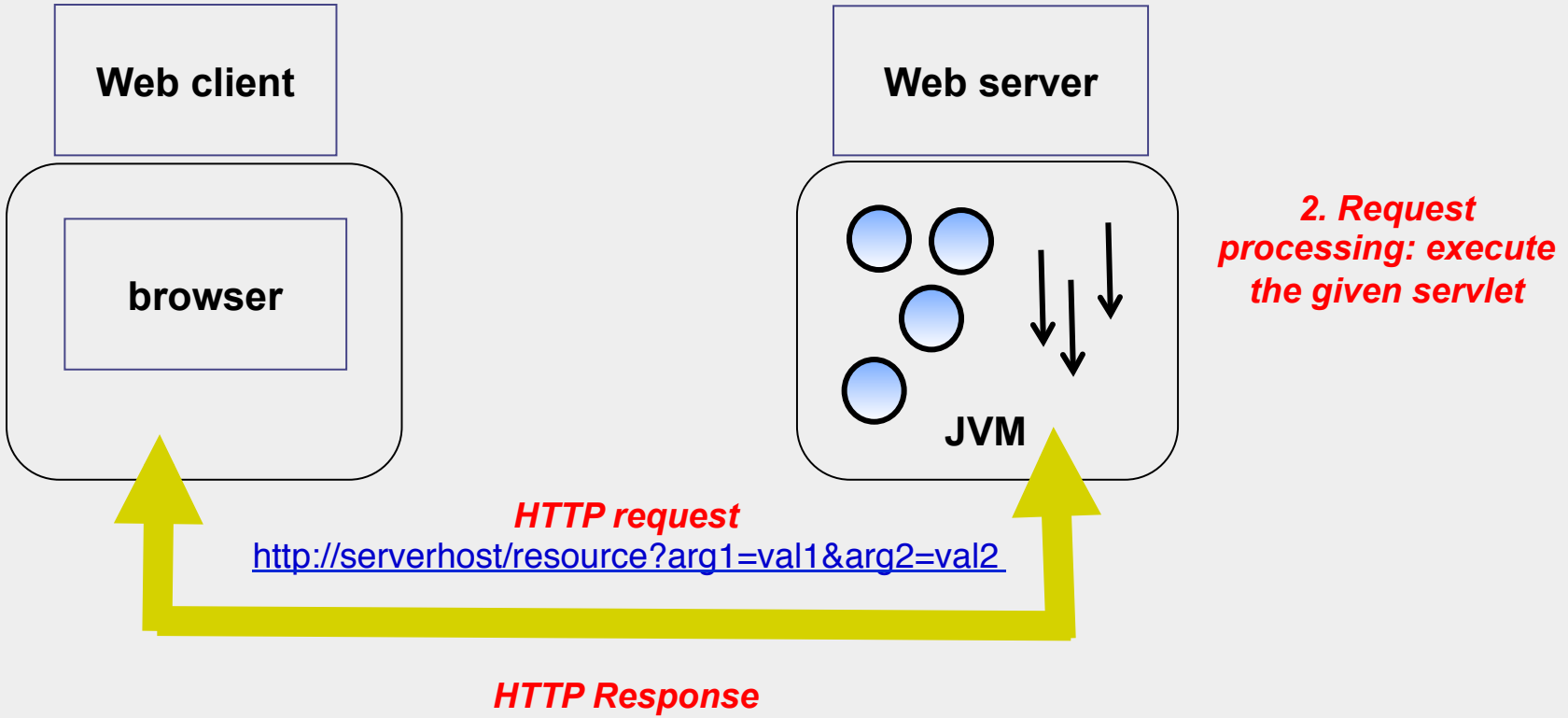
- **HTTP requests (verbs)**
  - GET, POST: to get/invoke a resource
  - PUT: to upload a resource
  - DELETE, CONNECT, OPTIONS, TRACE, ..

- **Rest (Representational State Transfer) API**
  - Allow to manipulate resources through uniform operations
  - Each request is independent
  - Each request is self-contained (contains the necessary information to be processed by any server)
  - → Make easier the management of faults / overloads

What are Servlets

- **A concept provided to build Web applications in the Java world**
  - Instead of programming a Web application as a set of programs or scripts (one per request type), servlets allows to program the application as a set of Java classes
- **Servlets are just code running server-side**
  - They can be remotely triggered by Web clients over HTTP
  - They build dynamic web pages on the fly and return them to clients

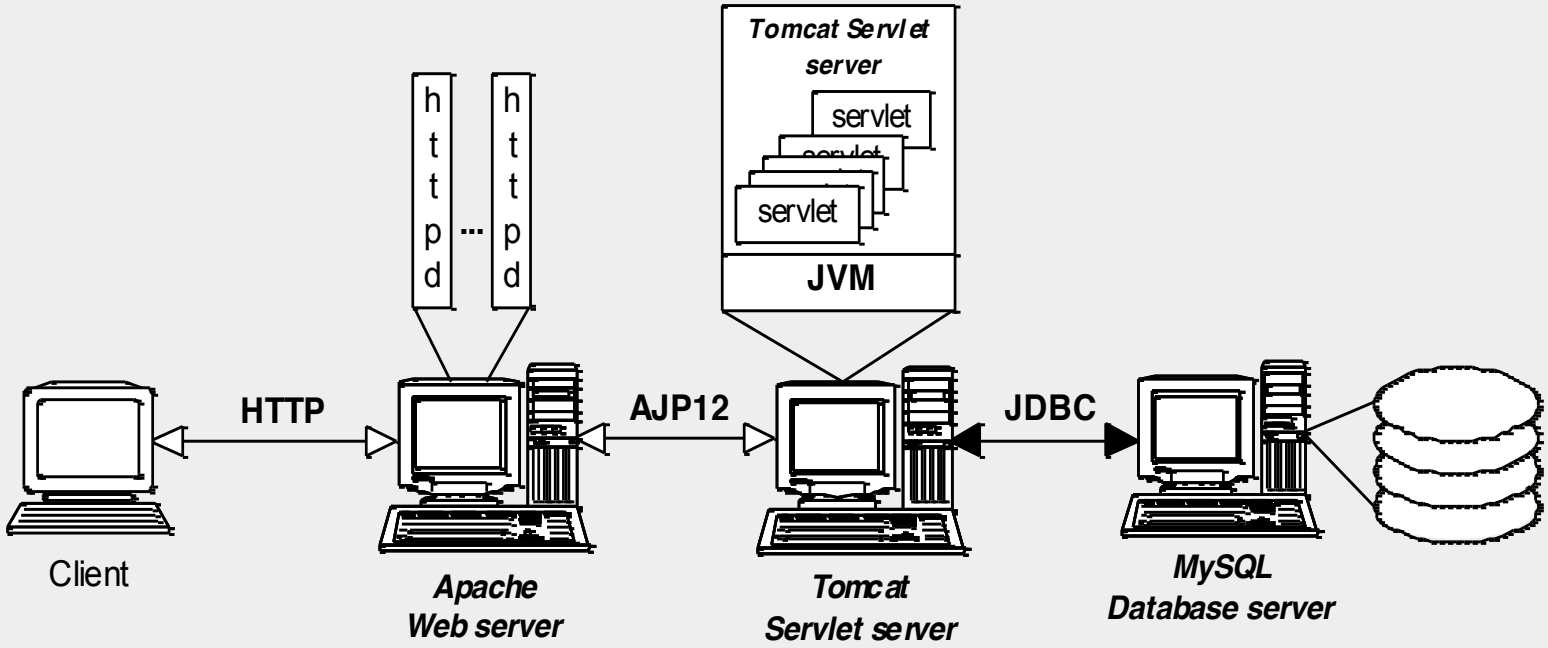
Servlet-based Web Architecture



Advantages of Servlets

- **Clarity**
  - Web application designed as an object-oriented program
  - Allow to build sophisticated backend applications
- **Efficiency**
  - With traditional CGI, a new process is started for each HTTP request
  - With servlets each request is handled by a thread
- **Portability**
  - Servlets are written in Java and follow a well-standardized API
  - So they can run virtually unchanged on any Servlet server (e.g. Apache Tomcat, IBM's WebSphere, BEA WebLogic Application Server, etc.)
- **Power**
  - Servlets provide useful facilities such as user session tracking and database connection pools

Typical Servlet-based Architecture



Outline

- Web architecture
- HTTP principles
- Servlet principles
- Tasks to do

HTTP principles

- **A simple stateless communication protocol**
  - Basically, no state kept at server-side
- **The HTTP protocol specifies the format of**
  - Requests
  - Responses
  - Headers

HTTP requests

- **Start with a request line specifying**
  - a method
  - a target resource address (URL)
  - a protocol version
- **Methods**
  - GET, POST, ..

Example

GET /intro.html HTTP/4.01

HTTP request headers

- **A header may follow a request line**
- **It tells extra information such as**
  - what software the client is running
  - what content types the client understands
- **This information could be used by the server in generating its response**
- **The header is followed by an empty line** (<CR><LF>, "\r\n")

Example

```
GET /intro.html  HTTP/4.01
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)
Accept: image/gif, image/jpeg, text/*, */*
```

HTTP responses

- **Start with a response line specifying**
  - the server's protocol version
  - a status code (e.g. 200 for successful, 404 for "Not Found")
  - a description of the status code

Example

HTTP/4.01 200 OK

HTTP response headers

- **A header may follow the response line**
- **It may indicate extra information such as**
  - what software the server is running
  - what content types the server understands
- **The header is followed by an empty line** (<CR><LF>, "\r\n")
- **If the request was successful, the requested data is then sent**

Example

```
HTTP/4.01          200        OK
Date: Saturday, 20-October-2007 03:25:12 GMT
MIME-version: 1.0
Content-type: text/html
Content-length: 12
Last-modified: Thursday, 18-October-2007 12:15:35 GMT

Hello World!
```

HTTP GET method

- **Designed for getting a resource, such as**
  - an HTML/image file, a chart, the result of a database query, ..
- **Can have parameters**
  - Example: an x, y scale for a dynamically created chart
  - Parameters are passed as a sequence of characters appended to the request URL

Example

```
GET http://www.mysite.com/mypage.html?x=100&y=1000  HTTP/1.0
```

HTTP POST method

- Designed for posting data to the server
- Passes this data as part of the HTTP request body
- So the data is not visible (in opposite to the parameters of the GET method that are visible as part of the URL)

Example

POST mypage.html HTTP/4.01

Host: www.mysite.com

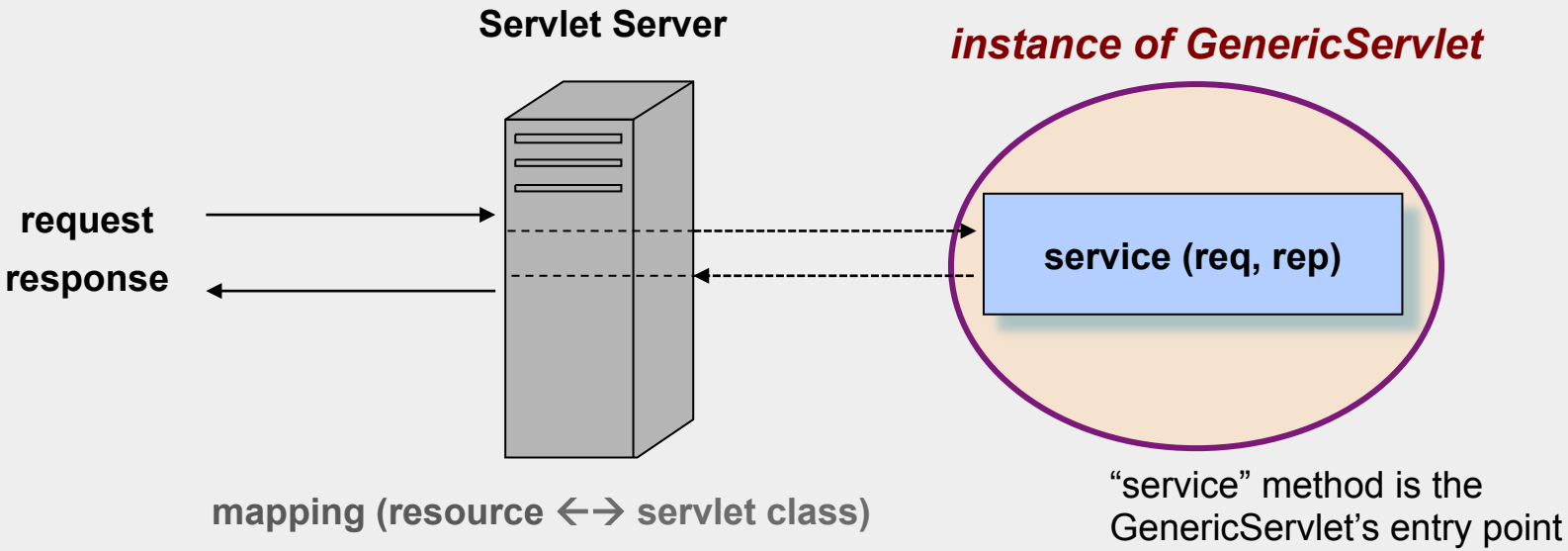
x=100&y=1000

Outline

- Introduction
- HTTP principles
- Servlets principles
- Tasks to do

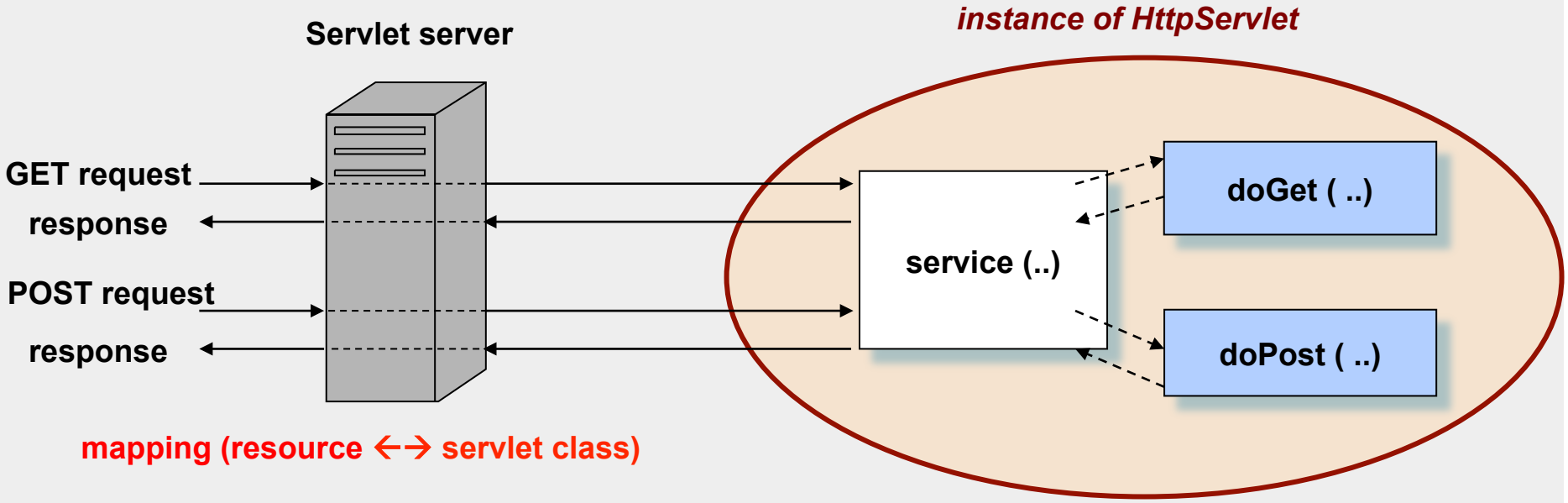
Generic Servlet

- A Java object serving requests



HTTP Servlet

- **A Java object serving HTTP requests**



"doGet" method is the entry point for GET requests

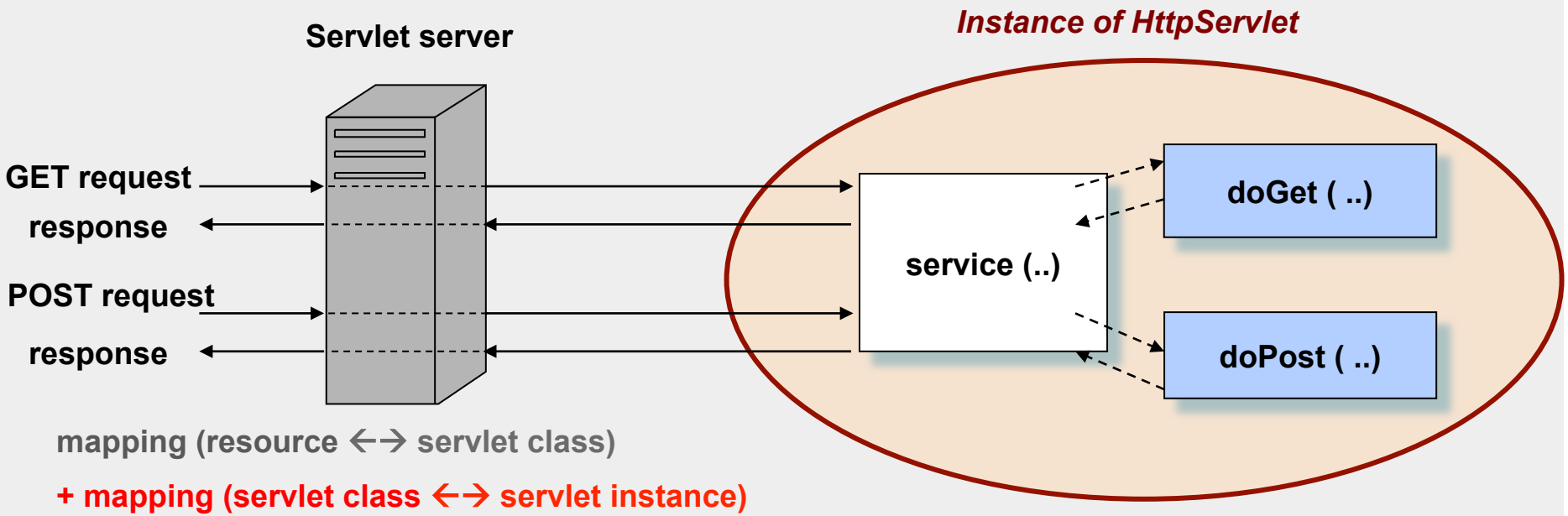
"doPost" method is the entry point for POST requests

Implemented by subclass of HttpServlet

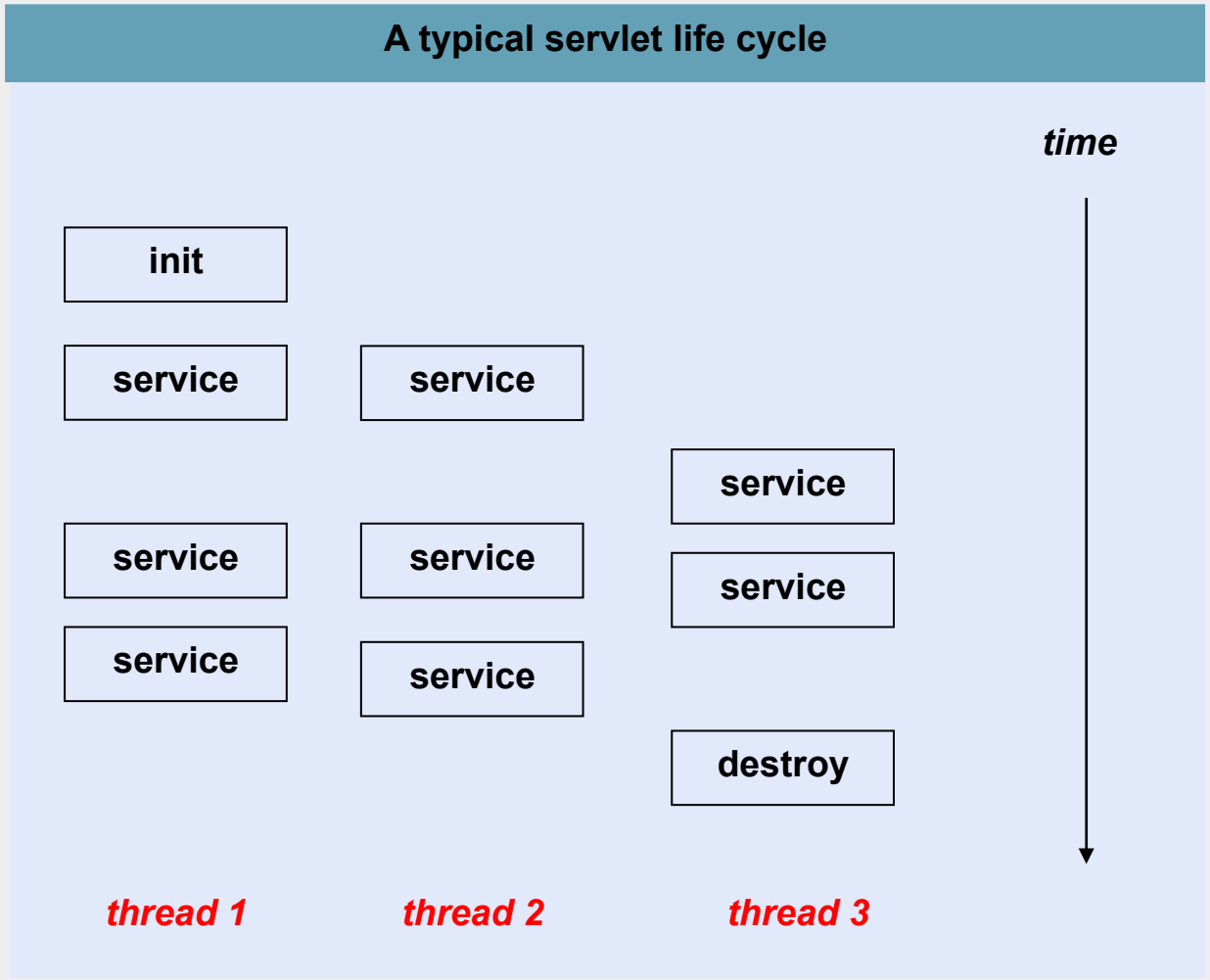
Servlet lifecycle

- **Does the servlet runtime create a servlet instance per request? NO**
- **The servlet runtime**
  - Manage servlets as singletons
  - When a request arrives, it creates the corresponding servlet if not already created
  - Then it invokes the service(..) method on the servlet
- **Creating a servlet goes though**
  - Loading the Servlet class
  - Creating an instance through the no-args constructor
  - Initializing the servlet (init(..) method)

HTTP Servlet



Servlet lifecycle: concurrent model



Servlet lifecycle: destruction

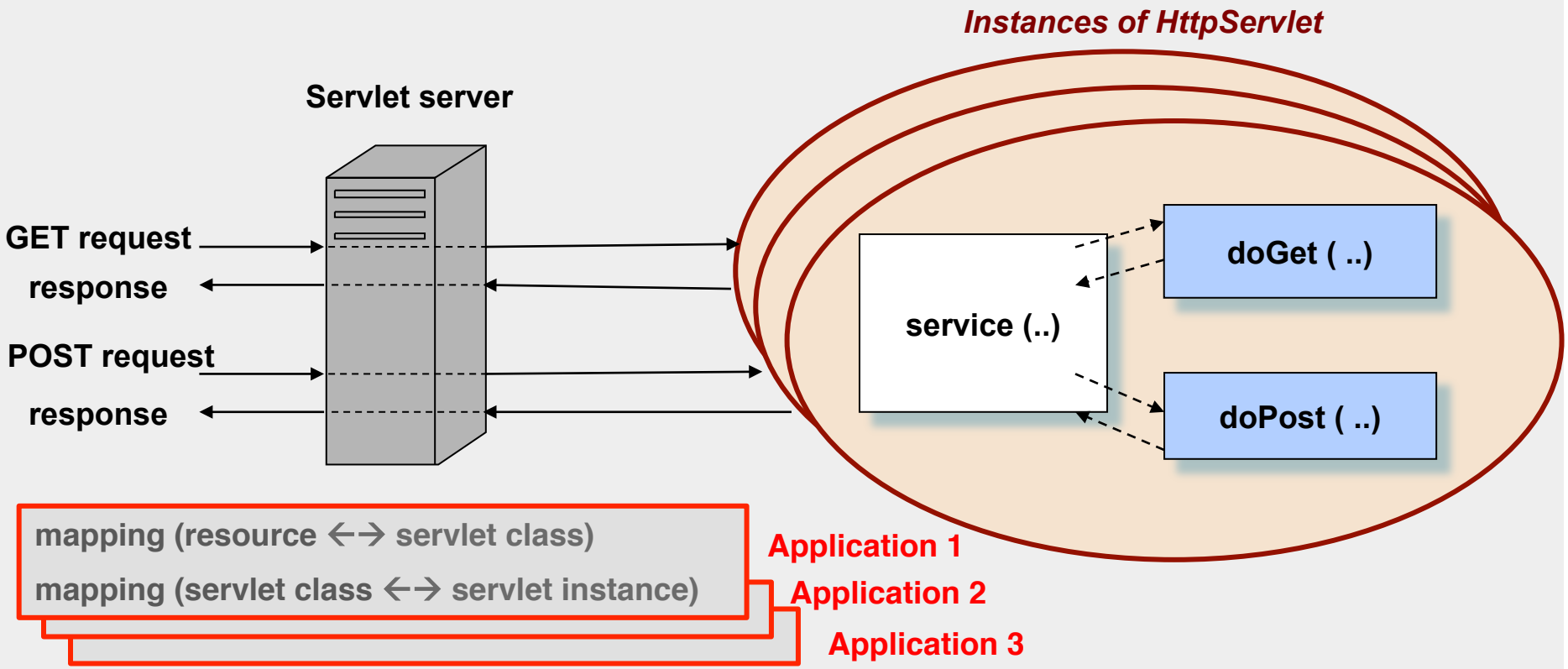
- **destroy() method**
  - When a server is shutdown, or when it needs memory place, the servlet runtime may unload (destroy) arbitrary servlets
  - When a servlet is about to be destroyed, its destroy() method is called
    - This method may save information on disk for future requests
  - There may still be threads that execute the service() method when destroy is called, so destroy() has to be thread-safe
  - The destroy() method is guaranteed to be called only once during the servlet's lifecycle

Servlet Protection Model

- **What execution model does a Servlet runtime adopt?**
  - One servlet runtime serves a single application? no
  - One servlet runtime serves several applications ? YES
    - More performant
    - But applications need their own servlet spaces
  - How to manage "isolated" servlet spaces?
    - Allocate a class loader to any application

      A class loader manages a mapping (class name, class object)
    - So the singleton property is ensured at the scope of an application

Servlet Protection Model



Servlet Packaging

- **One directory per application**
  - Web pages and other static resources (html, jpg, ..)
  - *WEB-INF* sub-directory
    - *classes* directory: servlet classes
    - *lib* directory: jar files
    - *web.xml* file: servlet descriptions

```
<web-app>

    <servlet>
    <servlet-name>myBank</servlet-name>
    <servlet-class>BankSvlet</servlet-class>
    </servlet>

    <servlet-mapping>
    <servlet-name>myBank</servlet-name>
    <url-pattern>/Bank</url-pattern>
    </servlet-mapping>

    ..
```

web.xml (example)

Servlet Programming:
a very simple HTTP Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<HTML>");
        out.println("<HEAD> <TITLE> Hello World example </TITLE> </HEAD>");
        out.println("<BODY> <BIG> Hello World </BIG> </BODY>");
    out.println("</HTML>");
        out.close();

    }
}
```

Getting information from requests

- **All requests implement the *ServletRequest* interface**
- **This interface gives information such as:**
  - String getParameter(String name):
    - returns the value of a request parameter as a String, or null if the parameter does not exist
  - String getProtocol():
    - returns the name and version of the protocol the request uses
  - String getRemoteAddr():
    - returns the Internet Protocol (IP) address of the client that sent the request

Getting information from requests

- **Example**
  - **A customer wishes to get information about a book**
  - **His browser invokes *BookInfoServlet,* specifying the book's id**
  - → http://host:port/servlets/BookInfoServlet?bookId=1234

```
public class BookInfoServlet extends HttpServlet {

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

        ...
        String bookId = req.getParameter("bookId");
        if (bookId != null) {
                // Retrieve information about that book
                ...
        }
  ...
```

Servlet Programming Pattern

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        // Use "req" to read incoming HTTP headers and HTML form data
        // (e.g. data the user entered and submitted)
        …
        // Perform any internal processing for generating dynamic results
        …
        // Use "resp" to specify the HTTP response line and headers
        // (e.g. specifying the content type)
        …
        // Use "out" to send result to browser
        PrintWriter out = resp.getWriter();
        …
```

Session data

- **Sometime an application needs to keep information between consecutive requests made from a client**
  - For instance, we may wish to keep a caddie content for an e-commerce application
  - Or when the result of a request depends on the previous requests
- **There are 2 ways to manage session data**
  - Server-side session data (HttpSession)
  - Client-side session data (Cookies)

Server-side: HttpSession objects

- **HttpSession object**
  - One per client per application
  - Created by a servlet (HttpSession HttpServletRequest.getSession(boolean create))
  - Maintain a map (String key, Object value)
  - Forgotten when *maxInactiveTime* expires
  - The session's identifier is added by the servlet layer to any response sent to the browser, and it will be added by the browser to any future request sent to the server

```
HttpSession
   Object getAttribute(String name)
   void setAttribute(String name, Object value)
   void removeAttribute(String name)
   String getId()
   void setMaxInactiveInterval(int interval)
   ..
```

Client-side: Cookies

- **Cookie**
  - A pair (String name, String value) kept at client side
  - A cookie is associated to a server
  - Cookies are added (by the browser) to any request sent to the server
- **A servlet can**
  - Get the cookies associated to a request (Cookie[ ] HttpServletRequest.getCookies())
  - Create a cookie (Cookie(String name, String value))
  - Add a cookie to a response (void HttpServletResponse.addCookie(Cookie cookie))

```
Cookie[ ] cookies = req.getCookies(); // example of servlet code
if (cookies != null) {
        for (int i = 0; i < cookies.length; i++) {
                String name = cookies[i].getName();
                String value = cookies[i].getValue();
        }
}..
```

Practical Work

- **Today – Installing & playing with an HTTP server on your laptop**
  - Install an Apache Web server on your laptop
  - Deploy and run simple examples of Web application
  - Define simple servlets and invoke them through your favorite browser
  - Manipulate cookies and session data
- **Next week & after - Build your own (HTTP-compliant) Web server**
  - A multithreaded server running on top of TCP/IP
  - Receiving HTTP requests and processing them
  - Providing your proper servlet-library

To finish, a few more words about the HTTP ecosystem

- **Web services**
  - A term to refer to any service that can be invoked as an HTTP request
  - A web service can be implemented through Java servlets, PHP programs, Python scripts, ..
- **WebSocket protocol**
  - HTTP (<1.1) triggered a new connection for any request
  - HTTP (>1.1) keeps connections between requests, but is verbose (headers) and does not support any "server push" mode
  - WebSocket is a protocol adressing the two previous limitations
  - WebSocket is setup through an HTTP request/response cycle
- **Node.js**
  - Server-side JavaScript runtime environment. Open-source
  - Event-based, basically mono-threaded, relies on Chrome V8 engine

Node.js vs Java Servlets (for back-end programming)

- **An epic battle..**

InfoWorld 2019, https://www.infoworld.com/article/2883328/nodejs-vs-java-an-epic-battle-for-developer-mindshare.html

  - Java strengths: robust platform, sophisticated IDEs, typed language, packaging features
  - Node.js strengths: ubiquity (code may run both at client and server side), easy queries to NO-SQL DB, JSON integration

→ Multi-threading (Java) seems better for CPU-intensive applications (streaming, image, ..)

→ Mono-threading (Node.js) seems better for IO-intensive applications (frequent disk access, back ups, ..)

Java Servlets 3.1 NIO

- **With traditional Servlets**
  - If the data coming into the server is blocking or streamed slower than the server can read, then the server thread that is trying to read this data has to wait for that data
  - If the response data from the server written to ServletOutputStream is slow, the server thread has to wait

  (https://stackoverflow.com/questions/39802643/java-async-in-servlet-3-0-vs-nio-in-servlet-3-1)

- **NIO Servlets avoid such blocking time**
  - Rely on ReadListener and WriteListener interfaces (callbacks for managing incoming / outgoing data)

Annex: Servlet API

- **Package javax.servlet**
  - Contains classes to support generic, protocol-independent servlets
  - Some elements of the package:
    - Servlet interface: methods that all servlets must implement
    - GenericServlet abstract class: generic, protocol-independent servlet
    - ServletRequest interface: methods to manipulate a servlet request
    - ServletResponse interface: methods to manipulate a servlet response
    - ServletConfig interface: defines the information used by a servlet container to pass to a servlet during initialization
    - ServletContext interface: methods used by a servlet to communicate with its servlet container, (e.g. write to a log file)

Annex: Servlet API

- **Package javax.servlet.http**
  - Contains classes to support HTTP-based servlets
  - Some elements of the package
    - HttpServlet abstract class: subclass of GenericServlet, provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site
    - HttpServletRequest interface: extends the ServletRequest interface to provide methods manipulating HTTP requests
    - HttpServletResponse interface: extends the ServletResponse interface to methods manipulating HTTP responses

HTML features

Feature	HTML 1.2	HTML 4.01	HTML5	Purpose
Heading	Yes	Yes	Yes	Organize page content by adding
Paragraph	Yes	Yes	Yes	Identify paragraphs of text
Address	Yes	Yes	Yes	Identify a block of text that contains contact information
Anchor	Yes	Yes	Yes	Link to other web content
List	Yes	Yes	Yes	Organize items into a list
Image	Yes	Yes	Yes	Embed a photograph or drawing into a web page
Table	No	Yes	Yes	Organize data into rows and columns
Style	No	Yes	Yes	Add CSS to control how objects are presented
Script	No	Yes	Yes	Add Javascript to make pages respond to user actions
Audio	No	No	Yes	Add audio to a web page with a single tag
Video	No	No	Yes	Add video to a web page with a single tag
Canvas	No	No	Yes	Add drawings (animations, games,..) using Javascript