Web Technologies

Lab session 4

HTTP protocol

- Now you know how to write beautiful, responsive and user-friendly applications that run within the web browser
- **Problem:** How do you deliver the application and its resources to the user?
  - Send it over email?
  - Upload it to an FTP server?
- Use a dedicated protocol: **HTTP**
  - Hyper-Text Transfer Protocol
- https://www.tutorialspoint.com/http/http_quick_guide.htm

HTTP: Overview

- Used to deliver hypermedia (HTML files, image files, query results etc) on the World Wide Web
- Based on TCP/IP communication stack
  - By default, HTTP uses TCP sockets and port 80
  - Client-server model: clients request resources, servers serve them
  - Text-based protocol
  - *Current* version: **HTTP/1.1**, HTTP/2 slowly on the way
- HTTP specifies
  - **how clients request data**, and
  - **how servers respond with data** to these requests

HTTP: Basic features

- HTTP is **connectionless** and **stateless**
  - The client (browser) creates a request and sends it to the server
  - The server process the request and sends back a response
  - The connection is closed
  - The server and client are aware of each other only during the request-response cycle; afterwards, both of them forget about each other
    - None of them can retain information between different request across different web pages
- HTTP is **media independent**
  - Any type of data can be sent over HTTP: the client and the server specify the content type using appropriate MIME-type

HTTP/1.1: Request example

- A client (web browser) sends a request to https://fri.uni-lj.si/sl

```
GET /sl HTTP/1.1
Host: fri.uni-lj.si
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,...
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8,sl;q=0.6,de;q=0.4,la;q=0.2
Cache-Control: max-age=0
Connection: keep-alive
Cookie: has_js=1
DNT: 1
Host: fri.uni-lj.si
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 …
```

Request line

Request headers

Empty line

HTTP/1.1: Response example

- The server sends back a response

```
HTTP/1.1 200 OK
Cache-Control: no-cache, must-revalidate
Connection: keep-alive
Content-Encoding: gzip
Content-Language: sl
Content-Type: text/html; charset=utf-8
Date: Mon, 20 Mar 2017 13:07:41 GMT
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Server: nginx/1.6.2
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Powered-By: HHVM/3.18.1

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
```

Status line

Response headers

Empty line

Response body (HTML page)

HTTP: Request and Response

- What is in an **HTTP request**?
  - **Request line**: Request method, URI, and protocol version
  - **Request headers**: request modifiers, client information, and possible body content
- What is in an **HTTP response**?
  - **Status line:** including the message's protocol version and a success or error code
  - **Response headers**: server information, entity meta-information, and possible entity-body content

HTTP: Request methods

- Indicates the method to be performed on the resource (in given URI)
- Always in uppercase

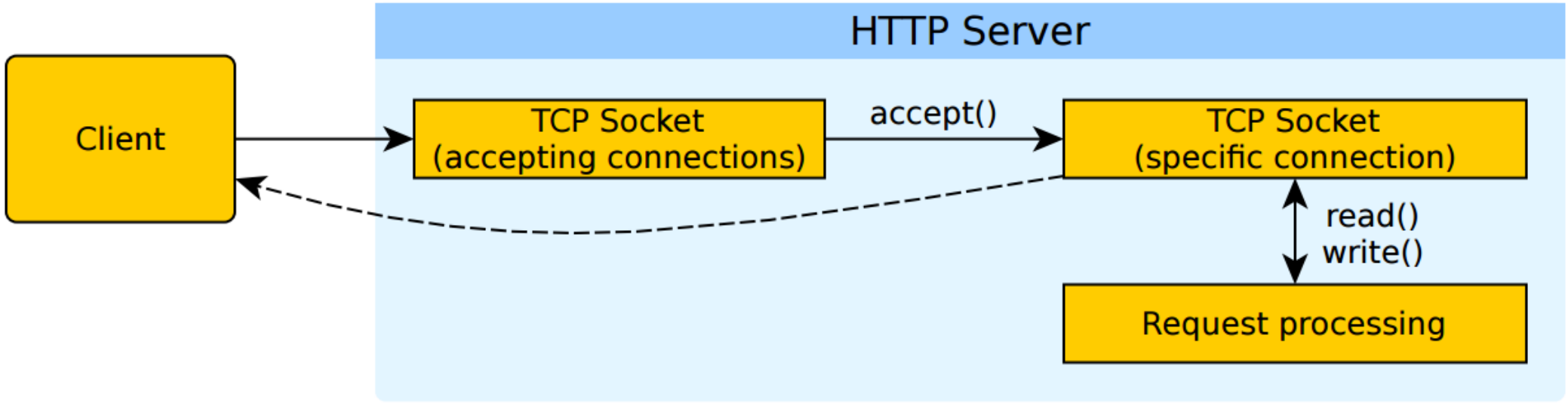
- **GET**: retrieve the resource
- **POST**: send resource to the server
- **PUT**: replace current resource on the server with provided resource
- **DELETE**: remove resource
- HEAD, CONNECT, OPTIONS, TRACE

HTTP server

- What is the task of an HTTP server? To run an endless loop in which it:
  - Waits for client requests: by calling `accept()`
  - *Serve* the requests
    - **Reads** request data from the **socket**
    - Creates a response content, and
    - **Writes** it to the **socket**
  - Closes the connection: by calling `close()` on the socket
- Several server architectures possible
  - Serial processing: **one request at a time**
  - Parallel requests: **pre-fork**
  - Parallel requests: **event-based**

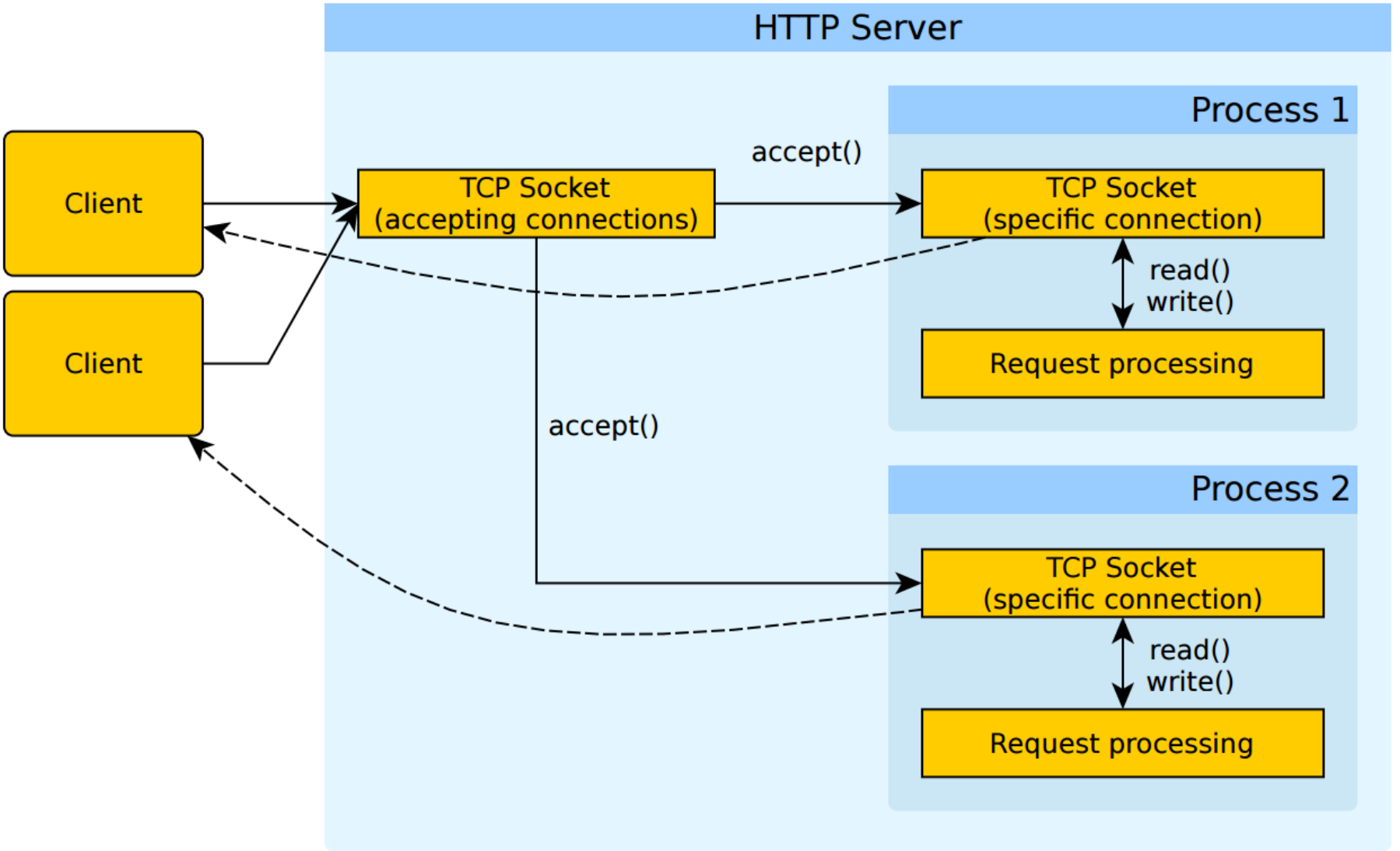
Serial processing

- Process one request at a time
- Slow performance
- Good for learning



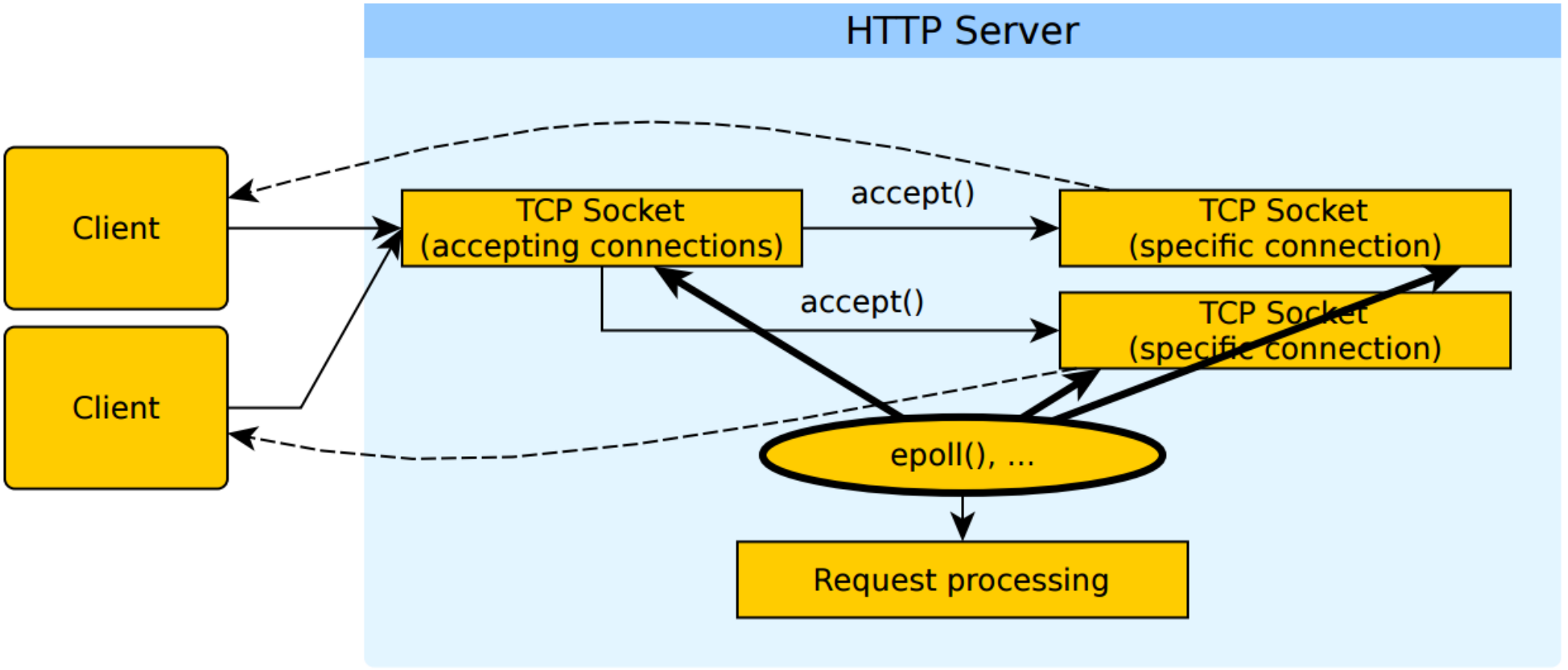
Parallel requests: pre-fork

- A pool of processes or threads



Parallel requests: event-based

- Multiplexing requests inside the same process



Assignment

- Write a simple HTTP server in Python
  - It should serve files inside some directory
  - It does not need to support parallel connections
- You have to implement
  - Accepting incoming connections (done)
  - Parsing of HTTP requests
  - Generation of HTTP responses
- The server should return either
  - The **contents of the resource** when the resource **exists**, or
  - A **404 message** when the **resource is not found**

Assignment

- Testing your solution
  - Use Telnet/Netcat/Putty to manually send HTTP requests
  - Use dedicated programs like curl
    - https://curl.haxx.se/download.html
  - Use your web browser