Razvoj programske potpore za web i pokretne uređaje

- predavanja -

2021./2022.

12. Dinamički web

4/4

Creative Commons

- slobodno smijete:
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
  - **prerađivati** djelo
- pod sljedećim uvjetima:
  - **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
  - **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava. *Tekst licence preuzet je s http://creativecommons.org/*

Dinamički web

- Performance: želimo ocijeniti obilježja i parametre kvalitete usluge web-poslužitelja
  - analiziramo detaljnije kako je implementiran web-poslužitelj
  - Kako se koristi transportni sloj za implementaciju web-poslužitelja i kako to utječe na njegove performance?
  - analiziramo najvažnije parametre performanci samog poslužitelja i komunikacije između klijenta i poslužitelja
- Pogledajmo detaljnije kako poslužiti **statički** sadržaj
  - napravimo rudimentarni web poslužitelj u Javi!
  - podržat ćemo samo GET zahtjev

Http Server u Javi

- … kako poslužiti **statički** sadržaj

➢ Napravimo rudimentarni web poslužitelj u Javi!

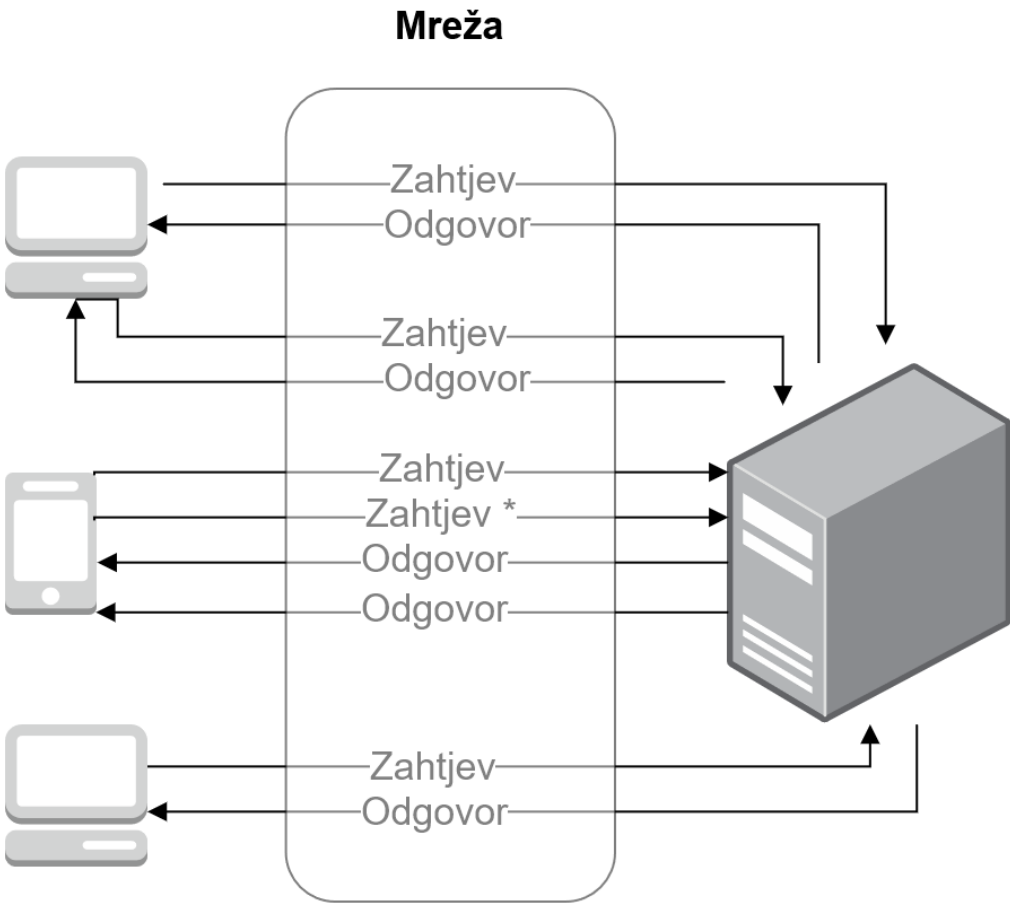
- Podržati ćemo **samo GET zahtjev**, vraćamo:
  - Ako je GET:
    - Ako se traži "/" -> vratiti defaultni `index.html`
    - Inače:
      - Ako postoji (npr. "`/slika.jpg`") - vrati traženi sadržaj (`slika.jpg`)
      - Inače: `404`
  - Inače za ostale zahtjeve (HEAD, POST, PUT, …):
    - `501 - method not allowed`
- HTTP je protokol aplikacijskog sloja koji koristi (tipično) TCP na transportnom sloju!

Podsjetimo se...

Model klijent-poslužitelj

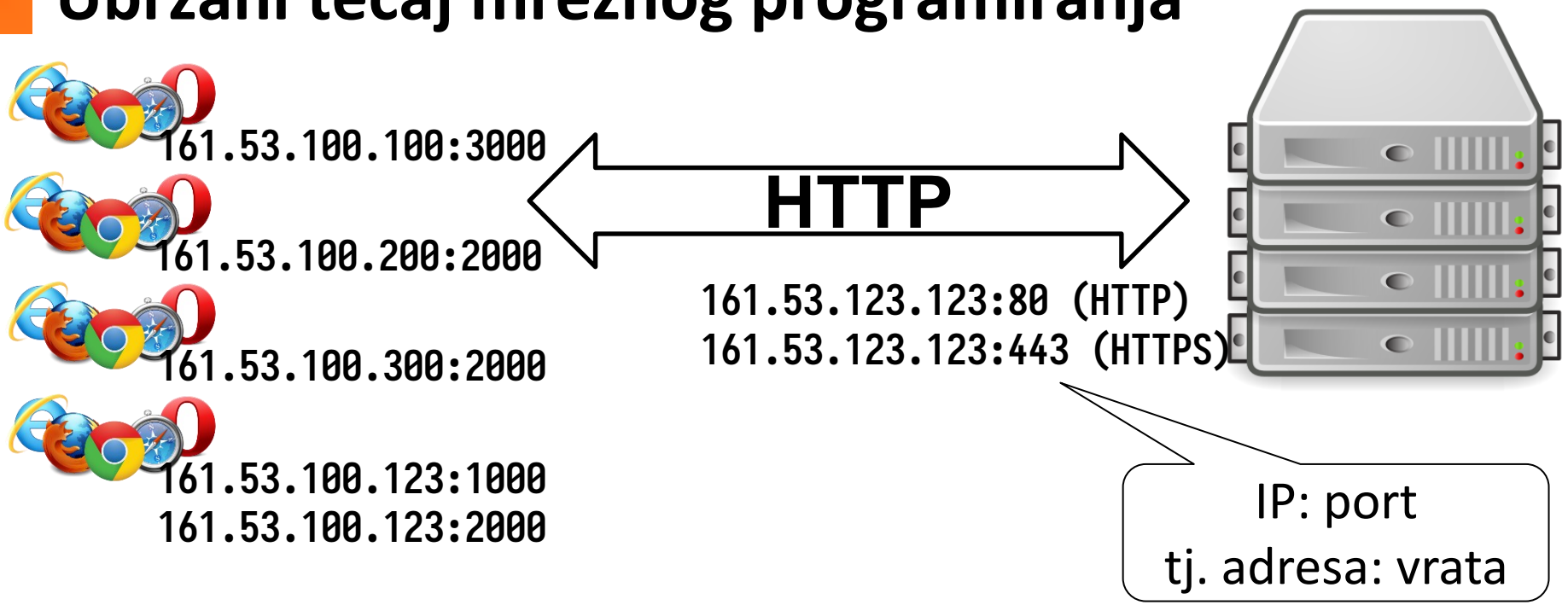
Klijent uvijek definira zahtjev, a poslužitelj odgovor:

request/response



* Verzije protokola HTTP koje se danas koriste omogućuju klijentu paralelno **slanje više zahtjeva** bez čekanja odgovora

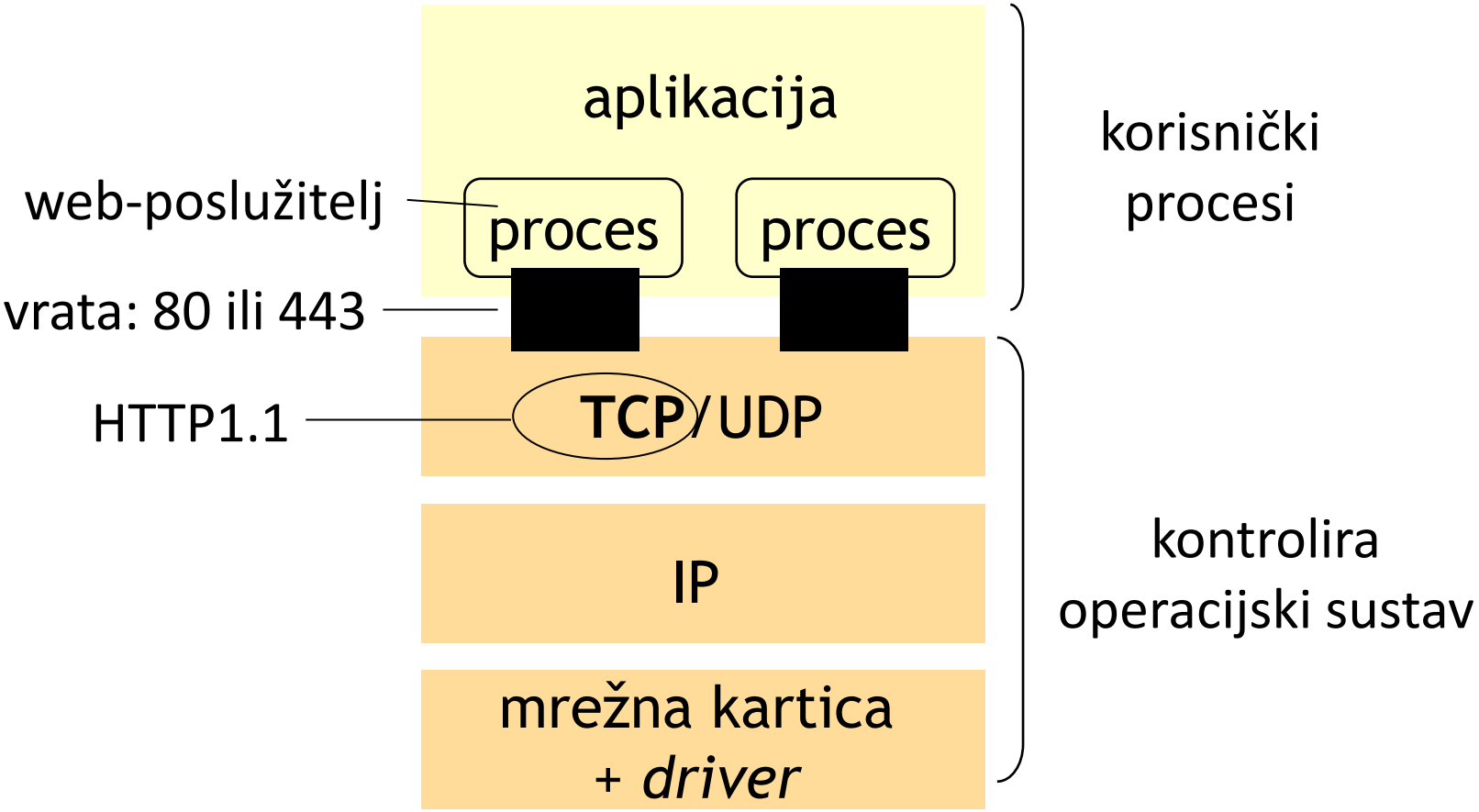
Ubrzani tečaj mrežnog programiranja



...

- Samo IP adresa nije dovoljna!
- Uvodi se pojam **transportne adrese**:
  - **Vrata** (*port*), npr. poslužitelj sluša na vratima 80
  - **Priključnica** (*socket*): IP + vrata, npr.
    - `161.53.100.123:1000 <-> 161.53.123.123:80`

Što je na strani poslužitelja?



Kako ostvariti komunikaciju klijent-poslužitelj?

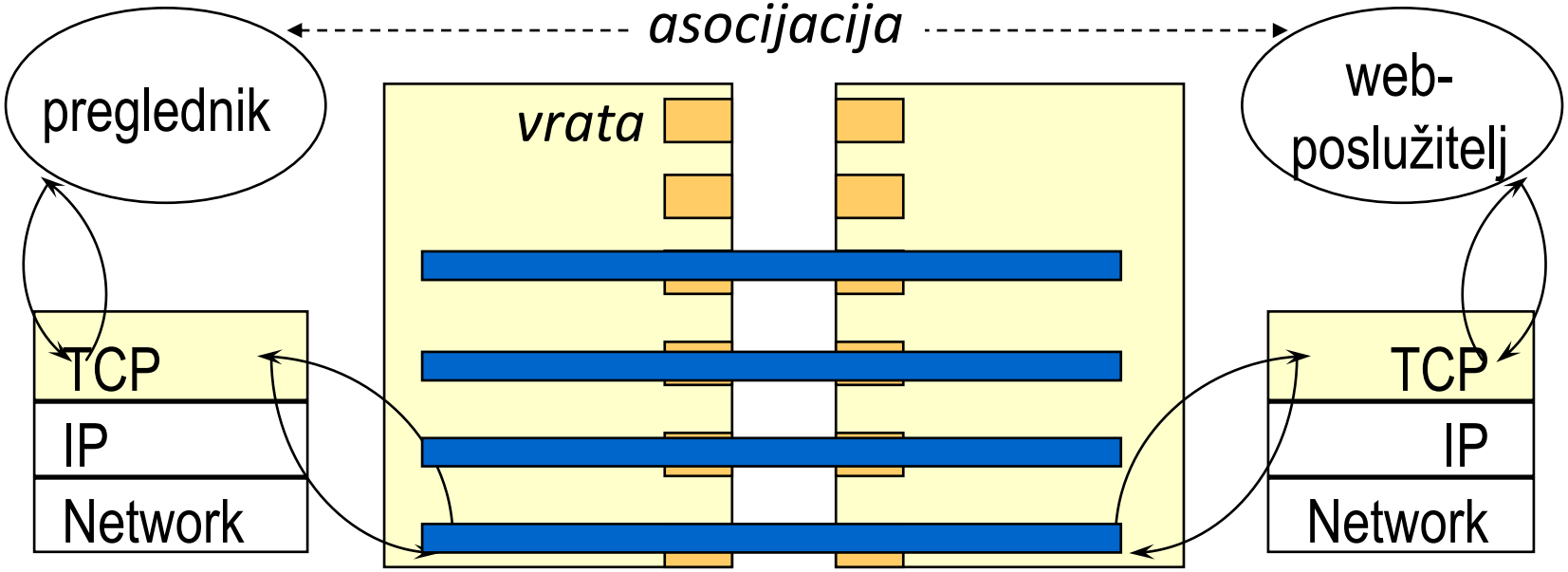
komunikacija korištenjem priključnica

- *Socket API*: programska knjižnica, koristi funkcionalnost transportnog sloja (TCP ili UDP)
  - TCP: koristi logičku vezu za pouzdan dvosmjerni prijenos podataka. Dijeli poruke u pakete i ponovno ih sastavlja u ispravnom slijedu na kraju primatelja. Brine se za ponovni prijenos izgubljenih paketa.
  - UDP – prijenos nezavisnih paketa (*datagrami*), nepouzdan prijenos
- priključnica (engl. *socket*)
  - komunikacijska točka preko koje proces šalje podatke u mrežu i iz koje čita primljene podatke
  - veže se uz (IP-adresa, transportni protokol, broj vrata), jednoznačno određuje proces kojemu su poruke namijenjene

Transportni protocol TCP

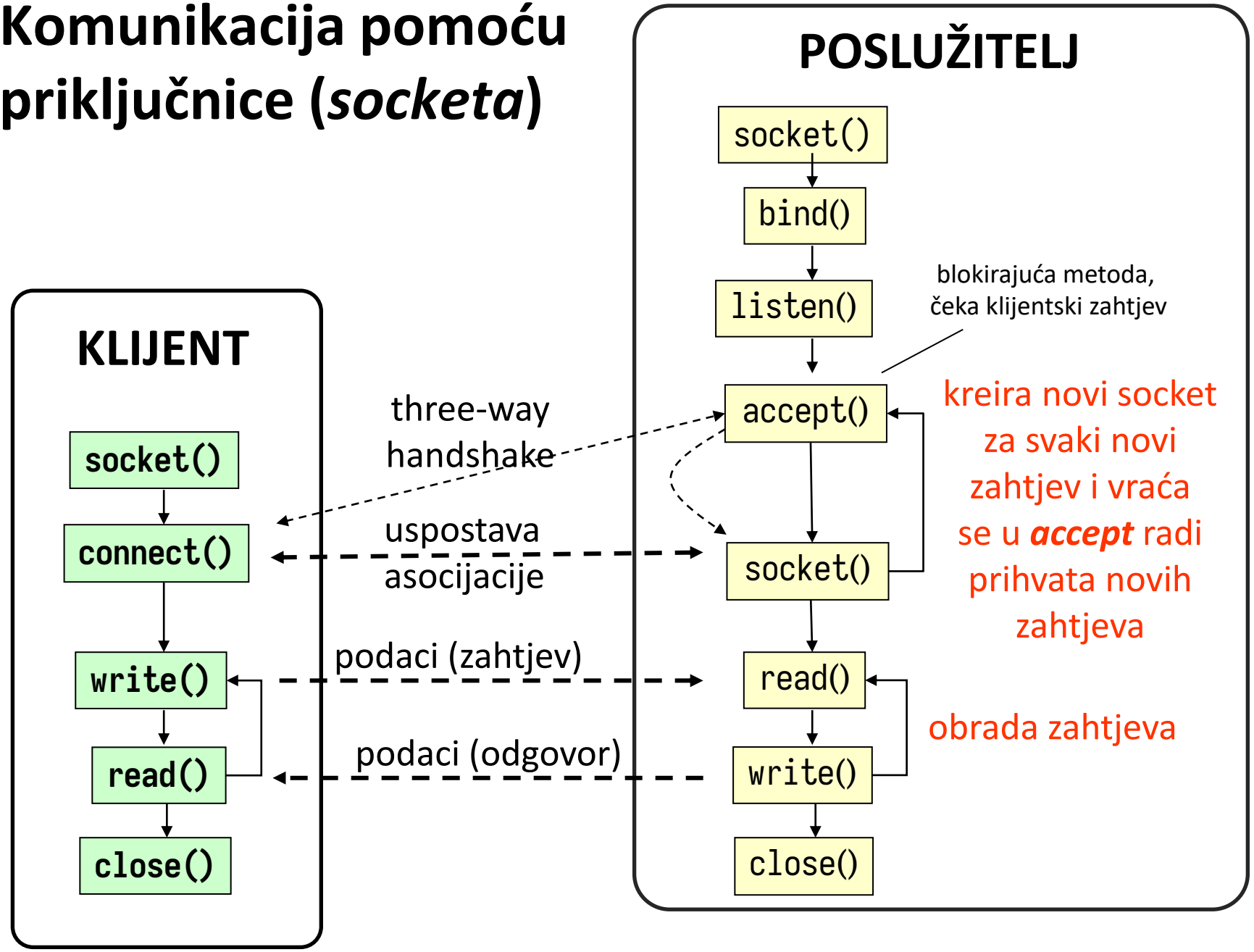
Transmission Control Protocol (TCP)

- omogućuje dvosmjernu komunikaciju između dvije krajnje točke koje se prvo moraju dogovoriti o logičke TCP-veze (*three-way handshake*)



logička TCP-veza

Komunikacija pomoću priključnice (*socketa*)



Mrežno programiranje u Javi

- Paket `java.net`
- **API specification**
  https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html
- Osnovne klase
  - Socket, ServerSocket, URL, URLConnection, (koriste TCP)
  - DatagramPacket, DatagramSocket, MulticastSocket (koriste UDP)
- Java Networking Tutorial
  http://docs.oracle.com/javase/tutorial/networking/

TCP: implementacija poslužitelja

1. Kreirati socket poslužitelja:

```
ServerSocket serverSocket;
serverSocket = new ServerSocket( PORT );
```

2. Čekati korisnički zahtjev (blokira proces do klijentskog zahtjeva!!!) i kreirati kopiju originalnog socketa:

```
Socket copySocket = serverSocket.accept();
```

3. Kreirati I/O stream za komunikaciju s klijentom

```
DataInputStream is = new DataInputStream(
    copySocket.getInputStream() );
DataOutputStream os = new DataOutputStream(
    copySocket.getOutputStream() );
```

4. Komunikacija s klijentom
5. Zatvoriti kopiju socketa:

```
copySocket.close();
```

6. Zatvoriti poslužiteljski socket:

```
serverSocket.close();
```

TCP: implementacija klijenta

1. Kreirati klijentski socket:

```
clientSocket = new Socket( address, port );
```

2. Kreirati I/O stream za komunikaciju s poslužiteljem:

```
is = new DataInputStream( clientSocket.getInputStream() );
os = new DataOutputStream( clientSocket.getOutputStream() );
```

3. Komunikacija s poslužiteljem:

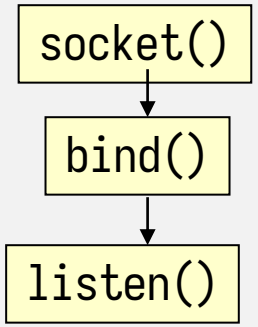
```
//Receive data from server:
            String line = is.readLine();
//Send data to server:
            os.writeBytes("Hello\n");
```

4. Zatvoriti socket:

```
clientSocket.close();
```

Primjer poslužitelja TCP (1)

```
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;

public class TCPServer {
   public static void main(String args[]) throws IOException,
        UnknownHostException  {

   int port = 10002;
   ServerSocket serverSocket = new ServerSocket(port);

   String rcvStr = null;
   String sendStr = null;

   ...
```

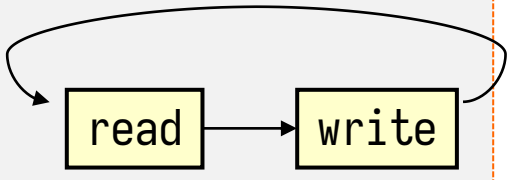


Primjer poslužitelja TCP (2)

```
...
   while( true ){
     Socket socket = serverSocket.accept();
     PrintWriter outToClient =
            new PrintWriter(socket.getOutputStream(), true);
     BufferedReader inFromClient = new BufferedReader(new

     InputStreamReader(socket.getInputStream()));
     while( (rcvStr = inFromClient.readLine()) != null ) {
            System.out.println( "Server received " + rcvStr );
            if( rcvStr.equals( "\n" ))
                break;
            outToClient.println(rcvStr.toUpperCase());
            System.out.println( "Server sends:\t" + sendStr);
        }
        outToClient.close();
        inFromClient.close();
        socket.close();
   }
 }}
```

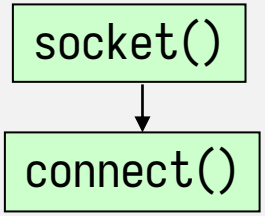
accept()

close()



Primjer klijenta TCP (1)

```
import java.io.IOException;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.Socket;

public class TCPClient {
    public static void main(String args[]) throws IOException,
   UnknownHostException  {

        String serverName = new String("localhost");
        int port = 10002;

        //create the socket connection
        Socket clientSocket = new Socket( serverName, port );
        String sendString = new String("Any string...");
   ...
```



Primjer klijenta TCP (2)

```
...
   //get the socket's output stream and open a PrintWriter on it
   PrintWriter outToServer =
        new PrintWriter( clientSocket.getOutputStream(), true);

   //get the socket's input stream and open a BufferedReader on it
   BufferedReader inFromServer = new BufferedReader(
        new InputStreamReader (clientSocket.getInputStream()));

   outToServer.println(sendString);
   String rcvString = inFromServer.readLine();
   System.out.println("FROM SERVER:" + rcvString);

   outToServer.println("\n");
   clientSocket.close();
  }
}
```

write()

read()

close()

Java HTTP server: klasa MyHttpServer (1/2)

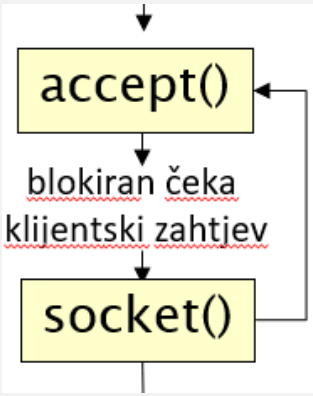
```
public class HttpServer implements Runnable {
    private Socket socket;
    public HttpServer(Socket s) {
        socket = s;
    }
    private void sendFile (ResponseCode responseCode, String fileName,
         PrintWriter txtOut, BufferedOutputStream binOut) throws IOException {}
    private byte[] readFileData(File file, int fileLength) throws IOException {}
    private String guessMimeType(String fileName) {}
    public static void main(String[] args) {}

    @Override
    public void run() {/* ovo se obavlja u svojoj posebnoj dretvi*/}
   // Razni formati ispisa na System.out, nebitno:
    private void printAndLog(PrintWriter txtOut, String msg){}
    private void logHttpIn(String msg) {}
    private void logHttpOut(String msg) {}
    private static void log(String msg) {}
}
```

Java HTTP server: klasa MyHttpServer (2/2)

```
public class HttpServer implements Runnable {

   // KONSTANTE:
    static final int PORT = 80;
    public enum ResponseCode {
         OK200, FileNotFound404, UnknownMethod501
    }
    private static final Map<ResponseCode,String> HttpResponseCodeMap = Map.of(
        ResponseCode.OK200, "HTTP/1.1 200 OK",
        ResponseCode.FileNotFound404, "HTTP/1.1 404 File Not Found",
        ResponseCode.UnknownMethod501, "HTTP/1.1 501 Not Implemented");
    private static final Map<ResponseCode,String> HttpResponseCodeFileMap = Map.of(
        ResponseCode.OK200, "index.html",
        ResponseCode.FileNotFound404, "404.html",
        ResponseCode.UnknownMethod501, "501.html");
    static final File WWW_ROOT = new File(".");

    ...
}
```

Java HTTP server: uspostava konekcije (socketa)

```
public static void main(String[] args) {
    try {
        ServerSocket ss = new ServerSocket(PORT);
        log("Server started.\nListening for connections on port : " + PORT + " ...\n");
        while (true) {
            HttpServer myWorker = new HttpServer(ss.accept());
            log("Connectin opened.");
            // Otvoriti posebnu dretvu koja će obraditi zahtjev:
            Thread thread = new Thread(myWorker);
            thread.start();
        }
    } catch (IOException e) {
        log("Error : " + e.getMessage());
    }
}
```



kreira novi socket za svaku novu konekciju i vraća se u ***accept*** radi novih zahtjeva

Java HTTP server: obrada zahtjeva: 1/2

```
@Override    // ovo se obavlja u svojoj posebnoj dretvi
public void run() {
    BufferedReader in = null;
    PrintWriter txtOut = null;
    BufferedOutputStream binOut = null;
    String fileRequested = null;
    try {
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        txtOut = new PrintWriter(socket.getOutputStream());
        binOut = new BufferedOutputStream(socket.getOutputStream());
        String input, firstLine = null;
        while ((input = in.readLine()) != null) {
            logHttpIn(input);
            if (firstLine == null) {
                firstLine = input;
            }
            if (input.equals("")) break;
        } //... nastavak na sljedećem slajdu...
```

Java HTTP server: obrada zahtjeva: 2/2

```
...     // Parsiramo na primitivan nacin prvu liniju da vidimo je li to GET
        StringTokenizer parse = new StringTokenizer(firstLine);
        String method = parse.nextToken().toUpperCase(); // HTTP method
        fileRequested = parse.nextToken().toLowerCase(); // datoteka?
        if (method.equals("GET")) {// podrzavamo samo GET:
            sendFile(ResponseCode.OK200, fileRequested, txtOut, binOut);
        } else {
            sendFile(ResponseCode.UnknownMethod501, null, txtOut, binOut);
        }
    } catch (IOException ioe) {
        log("Server error : " + ioe);
    } finally {
        try {
            in.close();  txtOut.close();  binOut.close();  socket.close();
        } catch (Exception e) {
            log("Error: " + e.getMessage());
        }
        log("Connection closed.\n");
    }
}
```

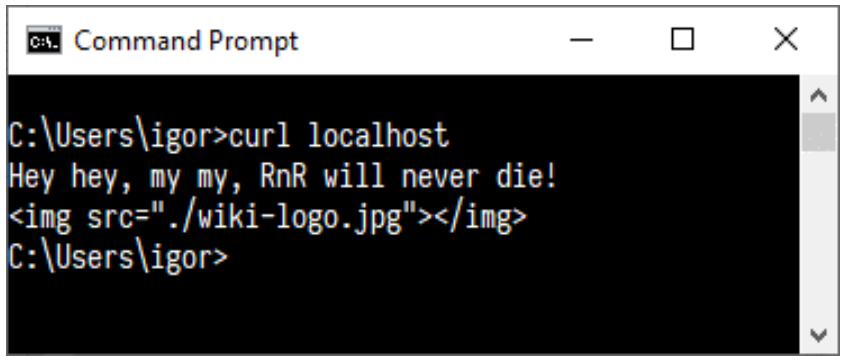
Java HTTP server: slanje odgovora

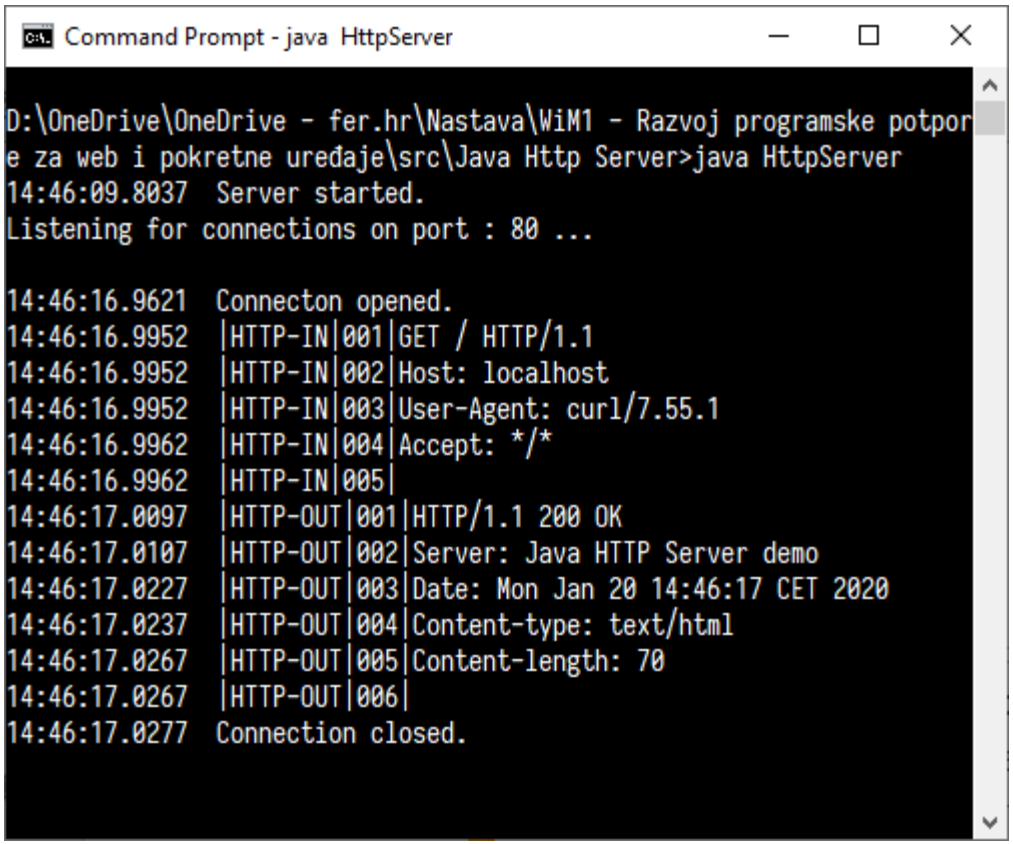
```
private void sendFile (ResponseCode responseCode, String fileName, PrintWriter txtOut, BufferedOu
tputStream binOut) throws IOException {
    if (responseCode != ResponseCode.OK200 ) {// dohvati dflt file za reponse code:
        fileName = HttpResponseCodeFileMap.get(responseCode);
    } else if (fileName.endsWith("/")) {// da dobijemo: /index.html
        fileName += HttpResponseCodeFileMap.get(responseCode);
    }
    File file = new File(WWW_ROOT, fileName);
    if (!file.exists() || file.isDirectory()) {
        sendFile(ResponseCode.FileNotFound404, null, txtOut, binOut);
    } else {
        int fileLength = (int) file.length();
        byte[] fileData = readFileData(file, fileLength);
        printAndLog(txtOut, HttpResponseCodeMap.get(responseCode));
        printAndLog(txtOut, "Server: Java HTTP Server demo");
        printAndLog(txtOut, "Date: " + new Date());
        printAndLog(txtOut, "Content-type: " + guessMimeType(fileName));
        printAndLog(txtOut, "Content-length: " + fileLength);
        printAndLog(txtOut, ""); // prazna linija koja razdvaja header i content !!
        txtOut.flush(); // flush character output stream buffer
        binOut.write(fileData, 0, fileLength); // binarni sadrzaj
        binOut.flush();
    }
}
```

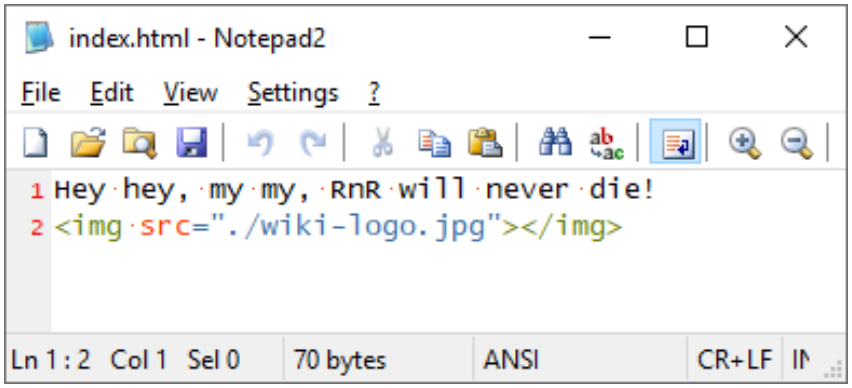
Java HTTP server: pomoćne funkcije

```
// ovo je naravno nepotpuno...
private String guessMimeType(String fileName) {
    if (fileName.toLowerCase().endsWith(".css"))       return "text/css";
    else if (fileName.toLowerCase().endsWith(".png"))  return "image/png";
    else if (fileName.toLowerCase().endsWith(".js"))   return "application/javascript";
    else return "text/html";
}

// samo pomoćna funkcija, ništa zanimljivo ni novo:
private byte[] readFileData(File file, int fileLength) throws IOException {
    FileInputStream fileIn = null;
    byte[] fileData = new byte[fileLength];

    try {
        fileIn = new FileInputStream(file);
        fileIn.read(fileData);
    } finally {
        if (fileIn != null)
            fileIn.close();
    }
    return fileData;
}
```

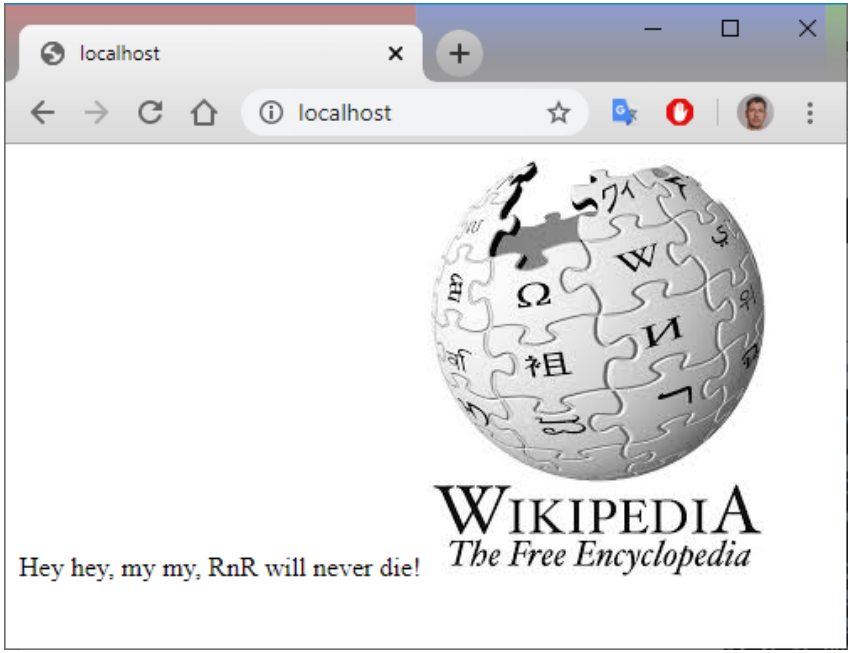
Java HTTP Server: primjer korištenja - curl

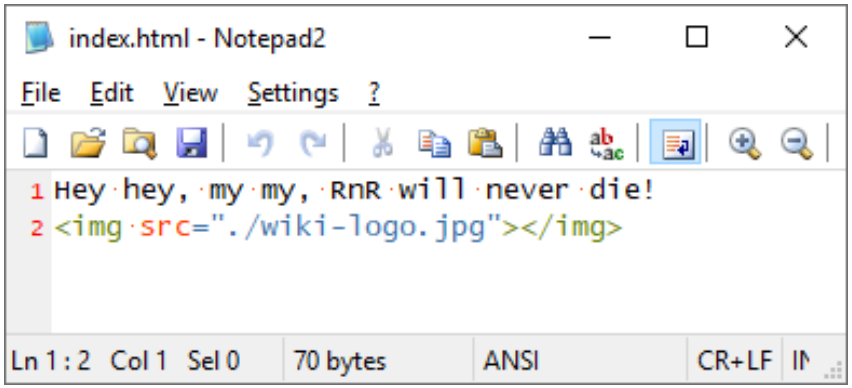






Java HTTP Server: primjer korištenja - Chrome

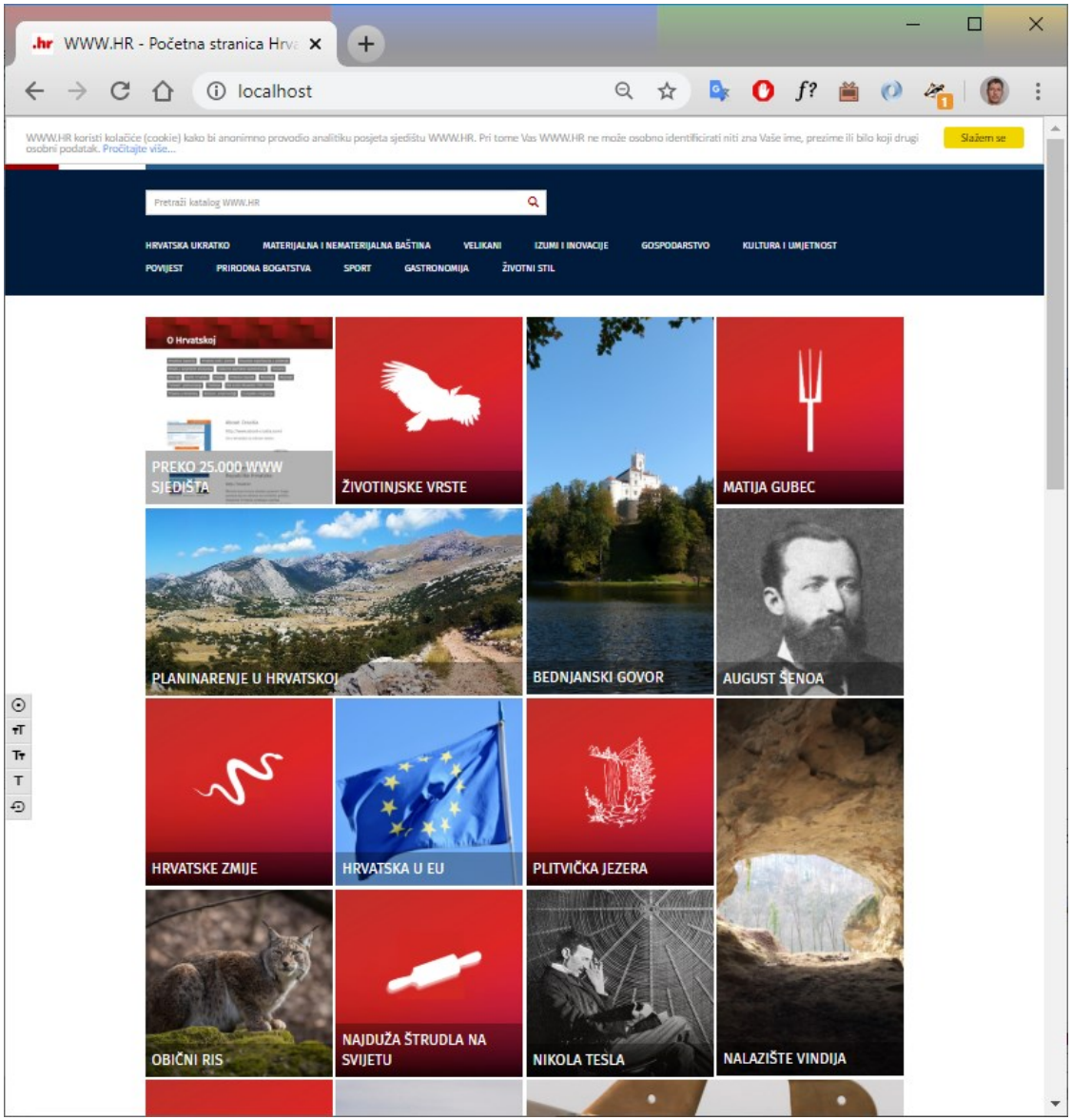


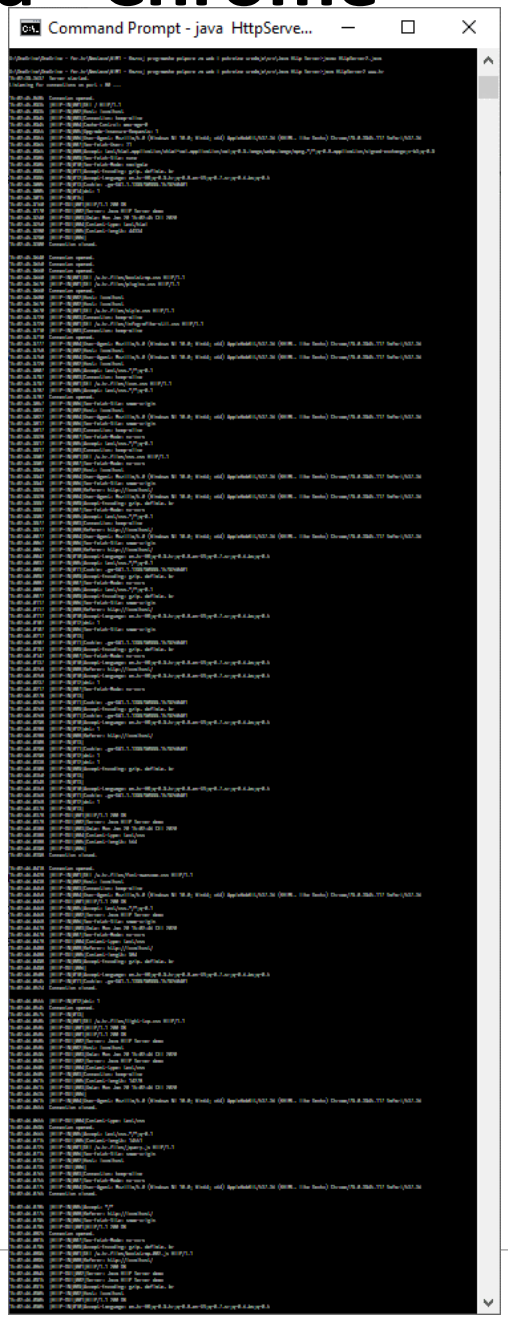


Command Prompt - java HttpServer

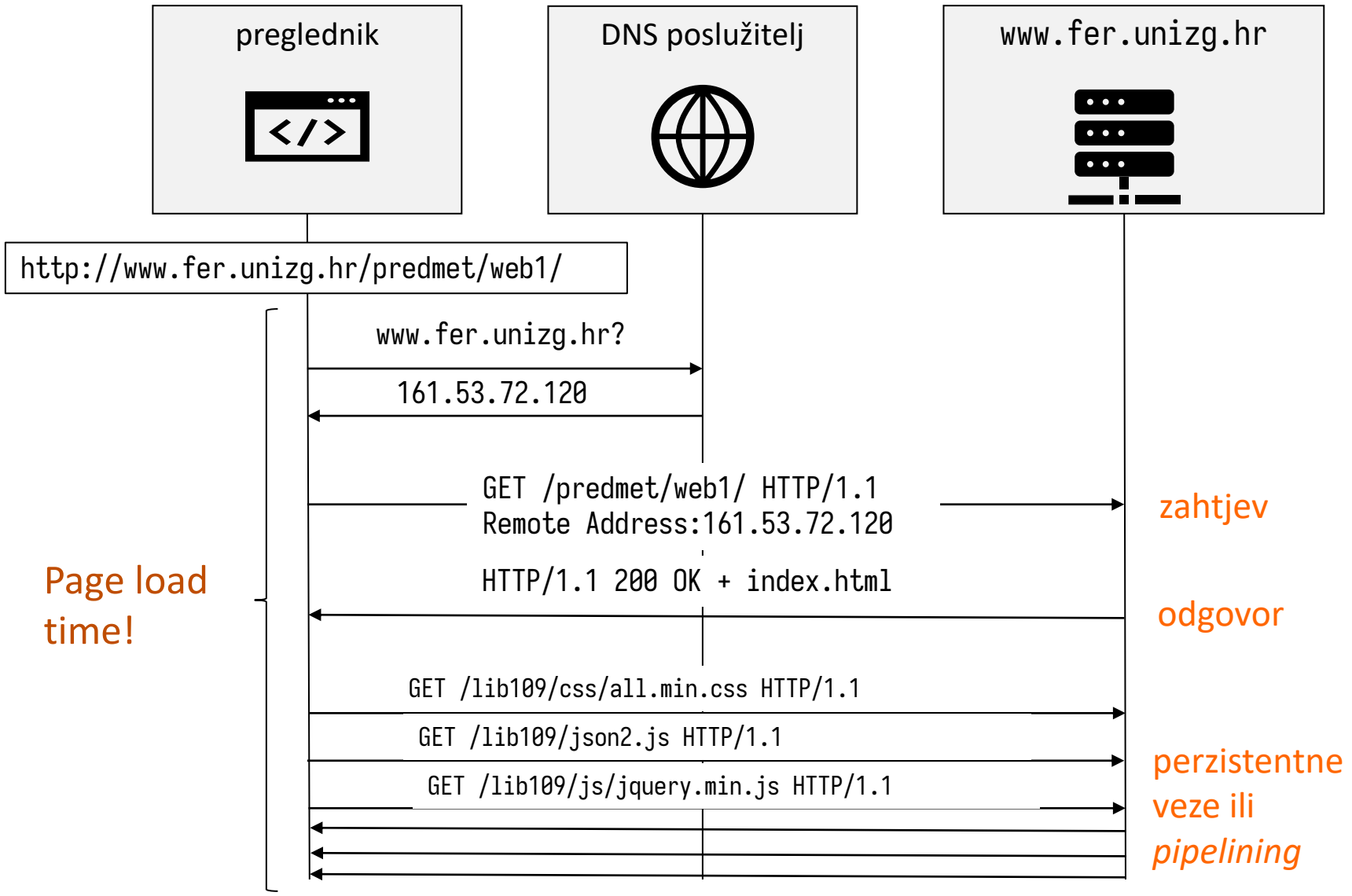
```
Listening for connections on port : 80 ...

14:48:38.0832  Connecton opened.
14:48:38.1152  |HTTP-IN|001|GET / HTTP/1.1
14:48:38.1162  |HTTP-IN|002|Host: localhost
14:48:38.1162  |HTTP-IN|003|Connection: keep-alive
14:48:38.1162  |HTTP-IN|004|Upgrade-Insecure-Requests: 1
14:48:38.1172  |HTTP-IN|005|User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/79.0.3945.117 Safari/537.36
14:48:38.1175  |HTTP-IN|006|Sec-Fetch-User: ?1
14:48:38.1175  |HTTP-IN|007|Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14:48:38.1182  |HTTP-IN|008|Sec-Fetch-Site: none
14:48:38.1182  |HTTP-IN|009|Sec-Fetch-Mode: navigate
14:48:38.1182  |HTTP-IN|010|Accept-Encoding: gzip, deflate, br
14:48:38.1182  |HTTP-IN|011|Accept-Language: en,hr-HR;q=0.9,hr;q=0.8,en-US;q=0.7,sr;q=0.6,bs;q=0.5
14:48:38.1202  |HTTP-IN|012|Cookie: _ga=GA1.1.1399790999.1579260401
14:48:38.1202  |HTTP-IN|013|dnt: 1
14:48:38.1212  |HTTP-IN|014|
14:48:38.1352  |HTTP-OUT|001|HTTP/1.1 200 OK
14:48:38.1352  |HTTP-OUT|002|Server: Java HTTP Server demo
14:48:38.1452  |HTTP-OUT|003|Date: Mon Jan 20 14:48:38 CET 2020
14:48:38.1452  |HTTP-OUT|004|Content-type: text/html
14:48:38.1492  |HTTP-OUT|005|Content-length: 70
14:48:38.1492  |HTTP-OUT|006|
14:48:38.1502  Connection closed.

14:48:38.1922  Connecton opened.
14:48:38.1932  |HTTP-IN|001|GET /wiki-logo.jpg HTTP/1.1
14:48:38.1942  |HTTP-IN|002|Host: localhost
14:48:38.1942  |HTTP-IN|003|Connection: keep-alive
14:48:38.1952  |HTTP-IN|004|User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/79.0.3945.117 Safari/537.36
14:48:38.1962  |HTTP-IN|005|Accept: image/webp,image/apng,image/*,*/*;q=0.8
14:48:38.1962  |HTTP-IN|006|Sec-Fetch-Site: same-origin
14:48:38.1972  |HTTP-IN|007|Sec-Fetch-Mode: no-cors
14:48:38.1972  |HTTP-IN|008|Referer: http://localhost/
14:48:38.1972  |HTTP-IN|009|Accept-Encoding: gzip, deflate, br
14:48:38.1982  |HTTP-IN|010|Accept-Language: en,hr-HR;q=0.9,hr;q=0.8,en-US;q=0.7,sr;q=0.6,bs;q=0.5
14:48:38.1982  |HTTP-IN|011|Cookie: _ga=GA1.1.1399790999.1579260401
14:48:38.1992  |HTTP-IN|012|dnt: 1
14:48:38.1992  |HTTP-IN|013|
14:48:38.2122  |HTTP-OUT|001|HTTP/1.1 200 OK
14:48:38.2132  |HTTP-OUT|002|Server: Java HTTP Server demo
14:48:38.2132  |HTTP-OUT|003|Date: Mon Jan 20 14:48:38 CET 2020
14:48:38.2132  |HTTP-OUT|004|Content-type: text/html
14:48:38.2142  |HTTP-OUT|005|Content-length: 12322
14:48:38.2142  |HTTP-OUT|006|
```

Java HTTP Server: primjer korištenja - Chrome

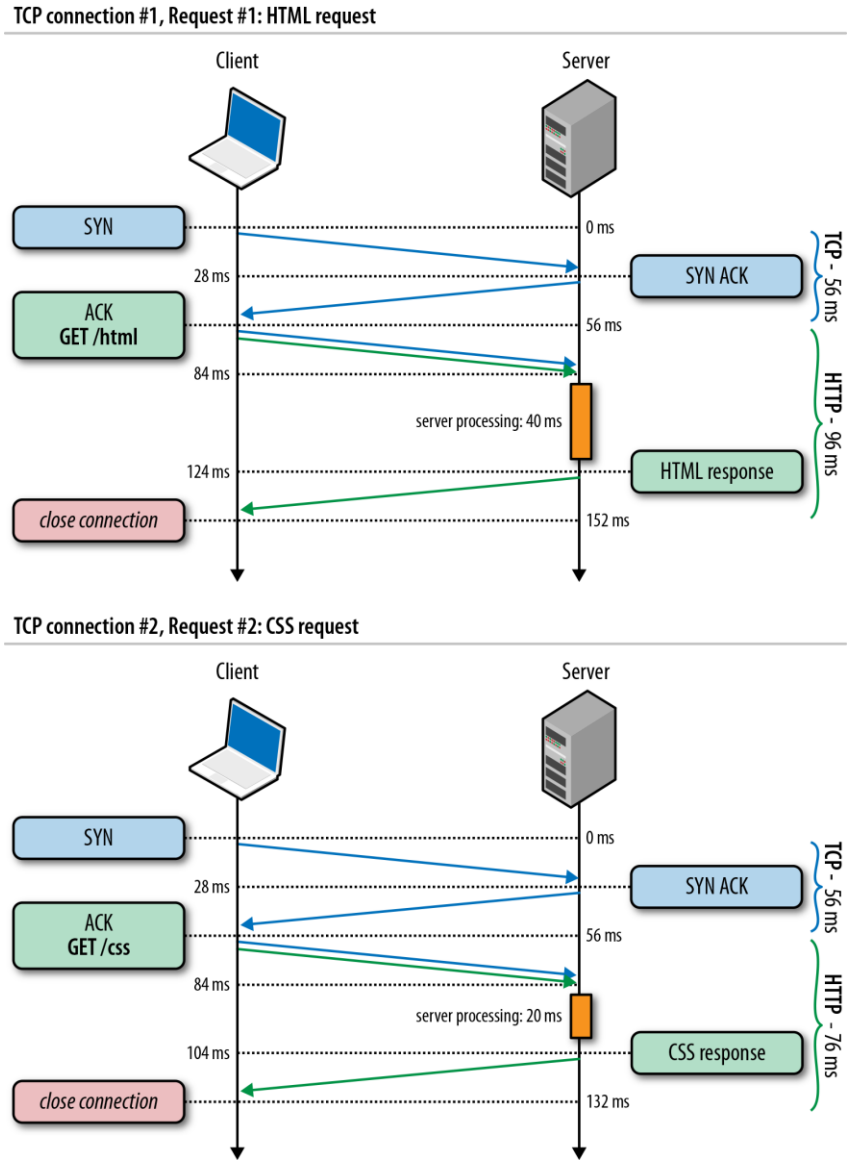




Prisjetimo se: Koliko zahtjeva šalje preglednik?



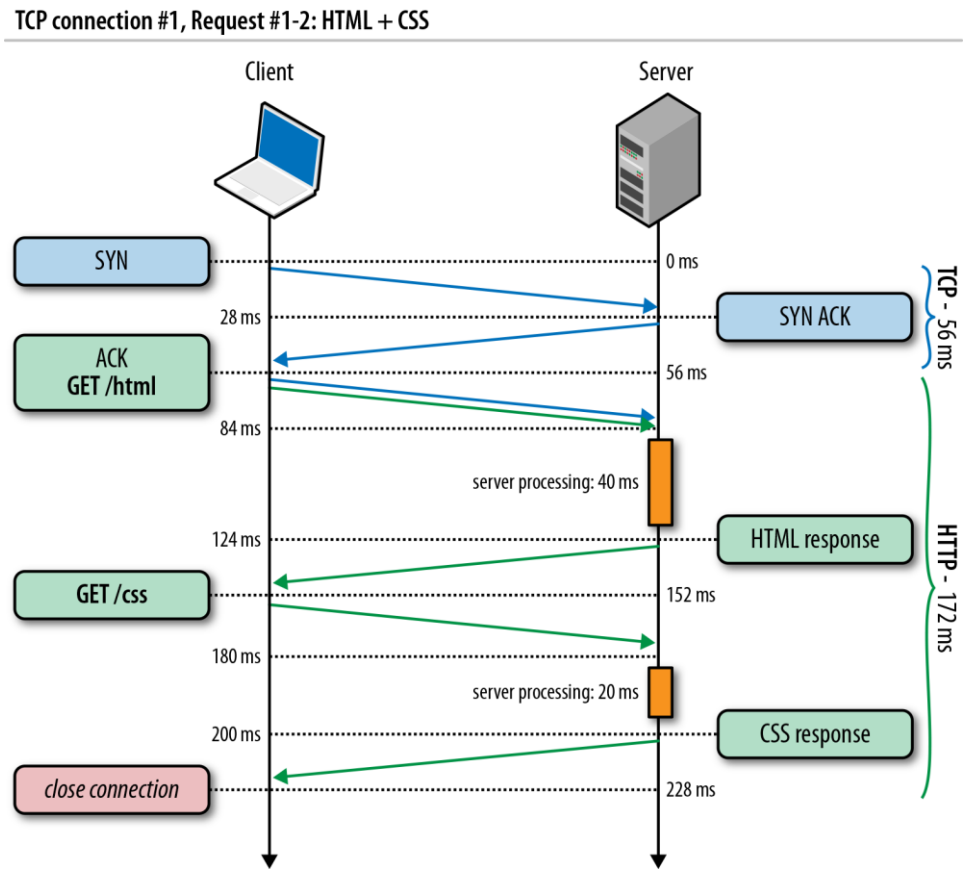
HTTP/1.0: nova veza za svaki zahtjev



Otvara se nova TCP-veza za svaki HTTP-zahtjev
Neučinkovito rješenje!
WWW prozvan “World Wide Wait”

Izvor: Iliya Grigorik, HTTP protocols, O’Reilly, 2017.

HTTP/1.1: postojana (perzistentna) veza



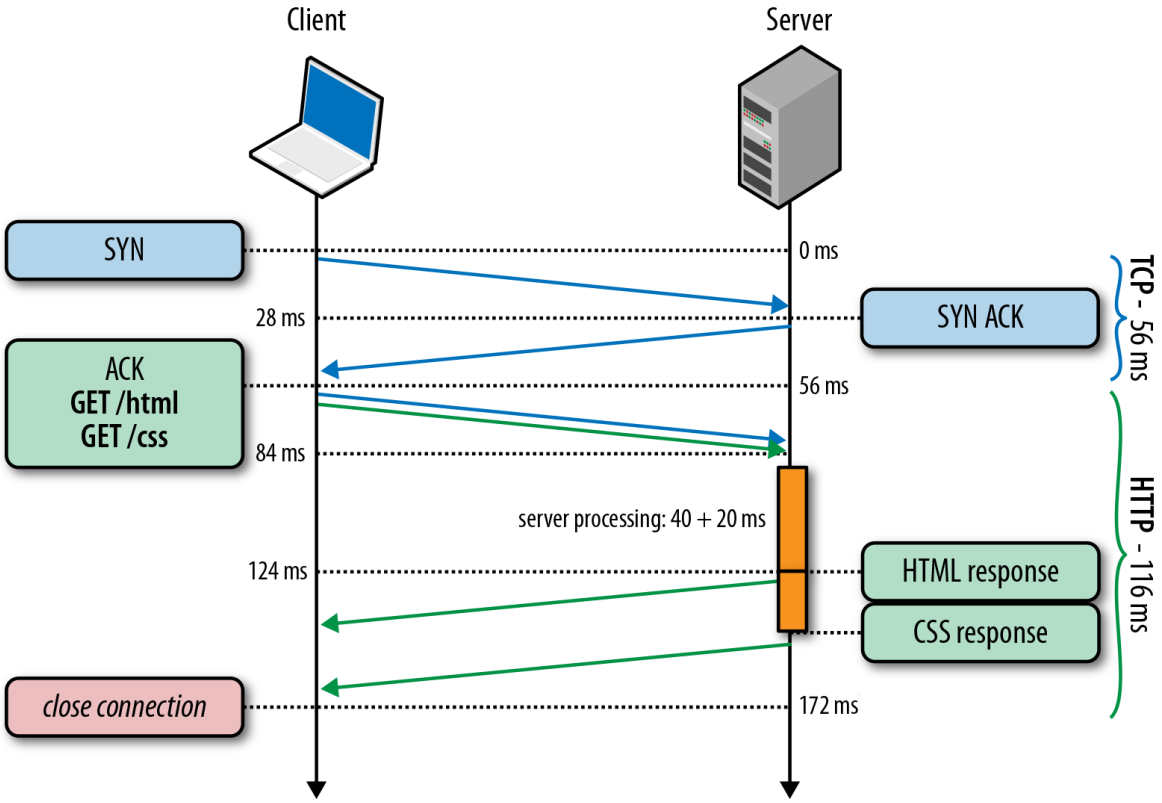
Zaglavlje odgovora: Keep-alive
Koristi se postojeća TCP-veza za sve zahtjeve od preglednika i poslužitelja (naravno do istog poslužitelja, za zahtjeve prema nekom drugom poslužitelju potrebno je otvoriti novu vezu)

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Thu, 11 Aug 2016 15:23:13 GMT
Keep-Alive: timeout=5, max=1000
Last-Modified: Mon, 25 Jul 2016 04:32:39 GMT
Server: Apache

(body)
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Keep-Alive

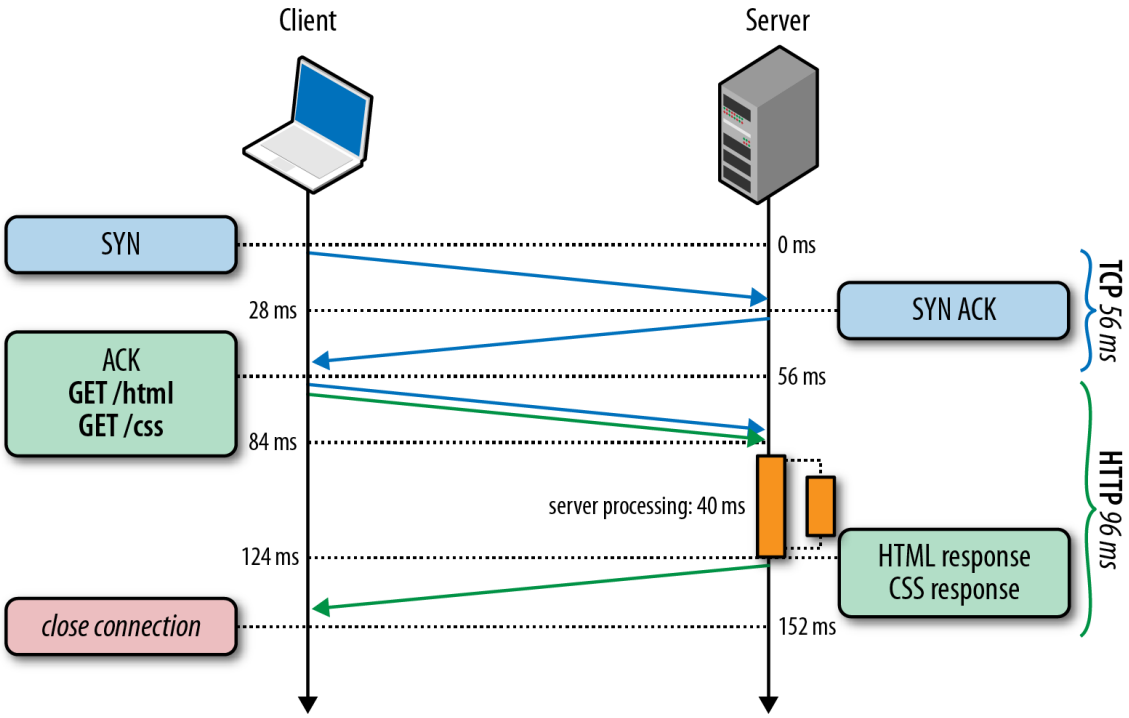
HTTP Pipelining: optimizacija

Prethodni primjer zahtijeva sekvencijalni način rada na strani preglednika: preglednik šalje novi zahtjev tek kada primi odgovor poslužitelja na prethodni zahtjev



HTTP pipelining omogućuje paralelno slanje više zahtjeva od preglednika na poslužitelj, poslužitelj ih u ovom primjeru obrađuje sekvencijalno po principu FIFO

HTTP Pipelining: paralelna obrada na poslužitelju

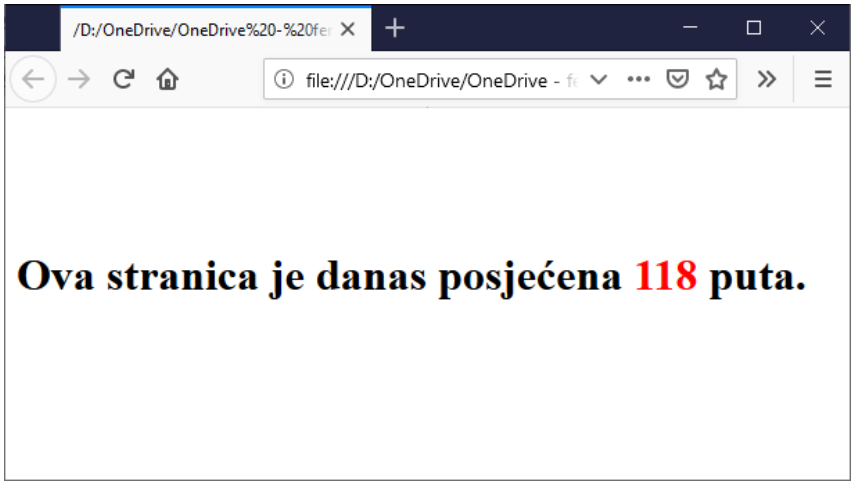


Protokol HTTP/1.x zahtijeva strogu serijalizaciju odgovora

Dva GET zahtjeva za datoteke HTML i CSS stižu paralelno na poslužitelj, ali zahtjev za HTML dokumentom je stigao prvi na poslužitelj.
Poslužitelj počinje obradu oba zahtjeva paralelno.
Zahtjev za CSS je obrađen prvi, ali se odgovor mora spremiti u međuspremnik dok se ne pošalje odgovor s HTML datotekom.
Iako je klijent izdao oba zahtjeva paralelno, a CSS resurs je prvi dostupan, poslužitelj mora pričekati da prvo dostavi HTML dokument prije nego što može nastaviti s isporukom CSS dokumenta.

Vratimo se na motivacijski primjer…

- Znamo poslužiti statički sadržaj, ali kako:

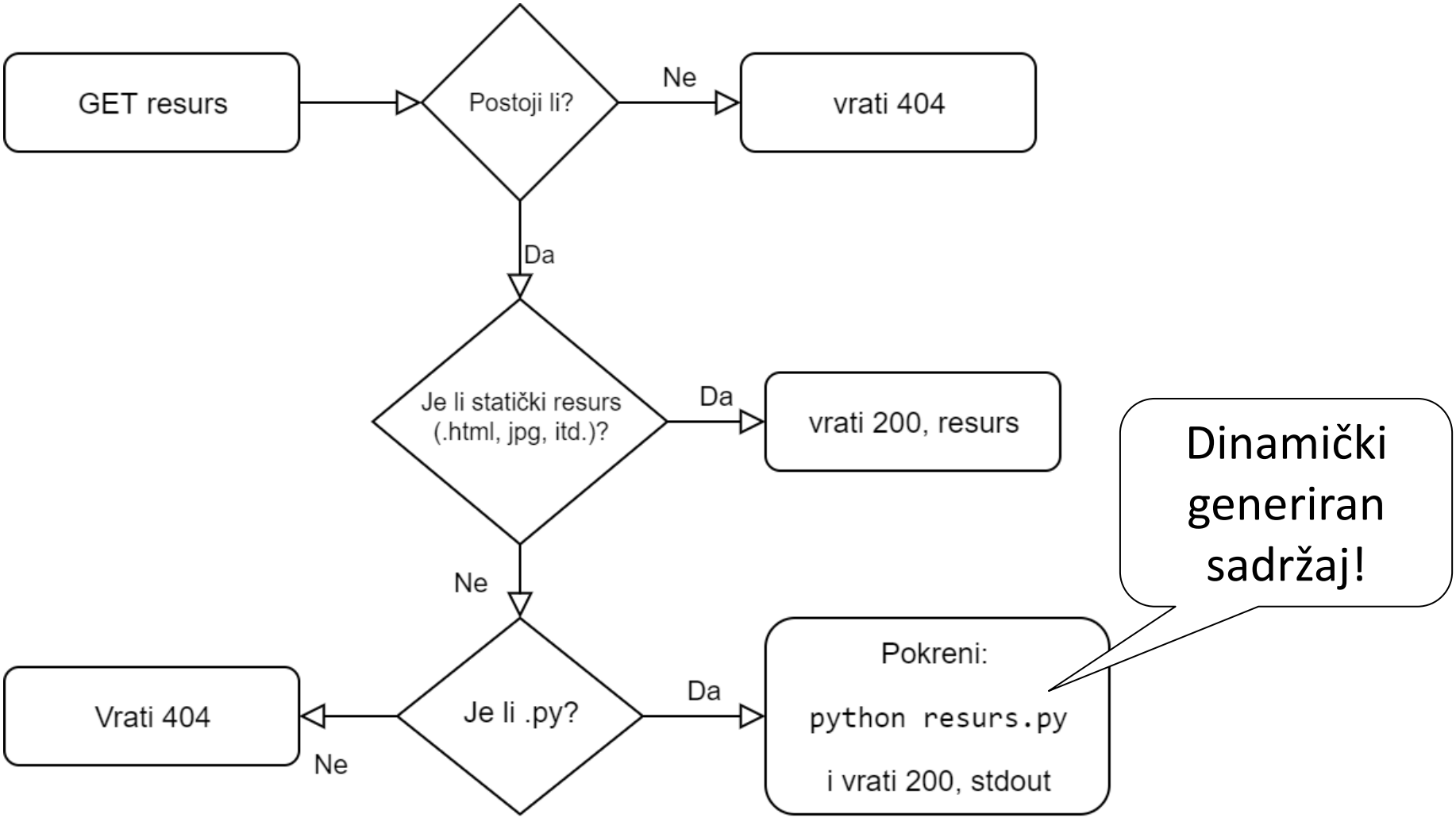


Napravimo mali program index.py koji broji pozive:

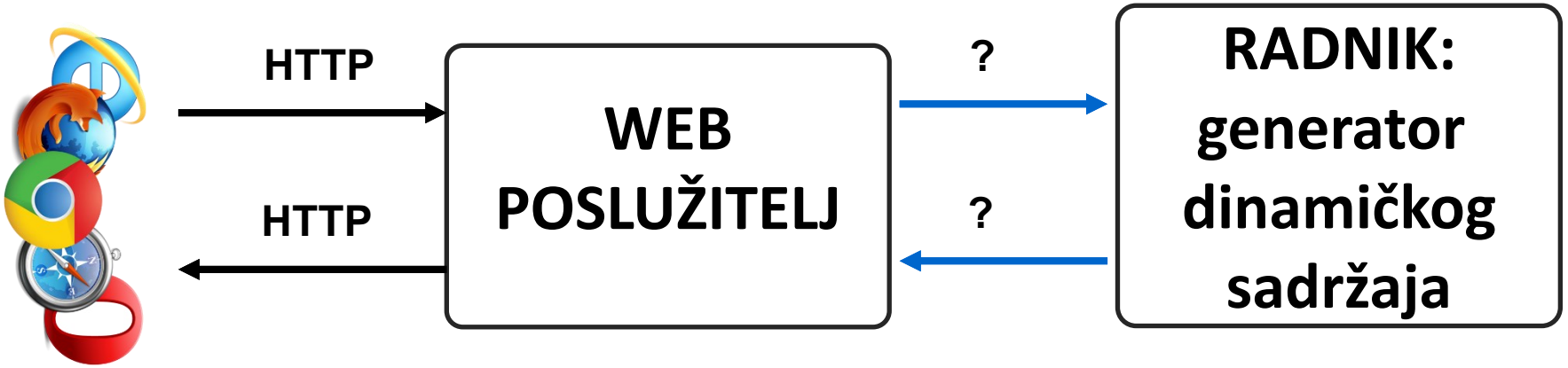
```
from datetime import date
today = date.today()
fc = today.strftime('%Y%m%d') + ".counter"
num = 0
try:
    f = open(fc, "rt")
    num = int(f.readline())
    f.close()
except IOError:
    pass
except:
    print("Other error")
num = num + 1
f = open(fc, "wt")
f.write(str(num) + "\n")
f.close()
print ("Ova stranica je posjecena " + str(num) + " puta.")
```

```
...>python index.py
Ova stranica je posjecena 1
puta.

...>python index.py
Ova stranica je posjecena 2
puta.

...>python index.py
Ova stranica je posjecena 3
puta.
```

Konačno, mogli bi modificirati naš poslužitelj:



Načelno, tako se generira dinamički sadržaj:



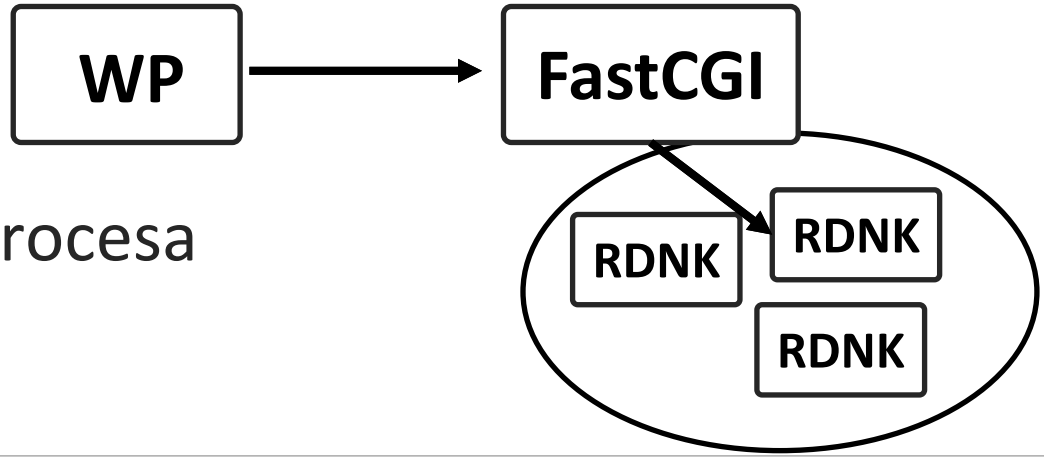
- Kako WP komunicira s radnikom (*worker*)?
- Procesni modeli: što je, kako nastaje i kakav je životni ciklus radnika?
- Kako nastaje dinamički sadržaj?
  - Interpretiranjem skripti (PHP, Perl, Ruby, Python)
  - Unaprijed prevedenim programima (JSP, Asp.Net, itd.)

Procesni modeli i protokoli (1/2)

- *In-process*
  - Opasno - radnik utječe na WP
  - Npr. ISAPI, Apache Server API, slabo korišteno
  - Protokol binarni, nestandardizirani

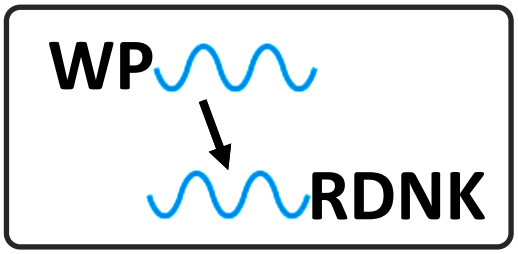
- Poseban proces
  - Sporo - stvaranje procesa je skupo
  - Komunciraju putem CGI (*Common Gateway Interface)* protokola, danas se slabo koristi
  - *npr. Perl*

- Poseban proces s bazenom (*pool*) drugih procesa
  - *Fast* CGI
  - Npr. PHP

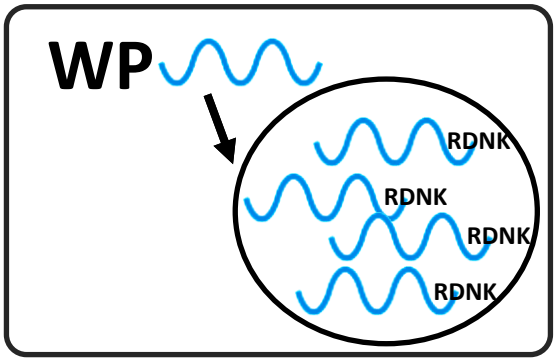


Procesni modeli i protokoli (2/2)

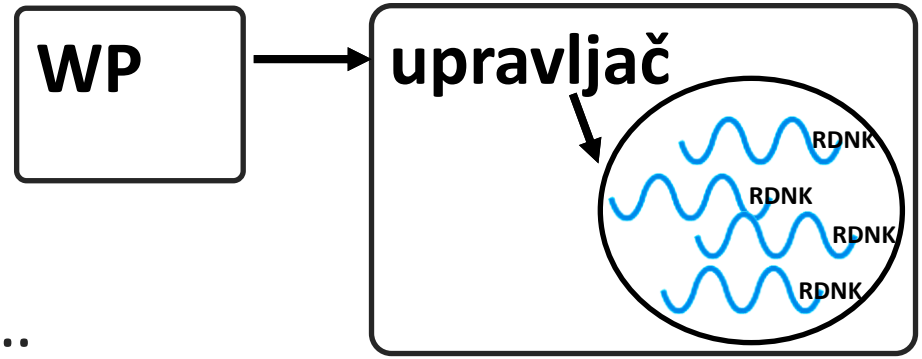
- Proces s dvije dretve (*thread*)
  - WP stvara dretvu radnika kad treba obraditi zahtjev



- Proces s bazenom dretvi
  - WP održava bazen dretvi kako bi uštedio kod stvaranja radnika



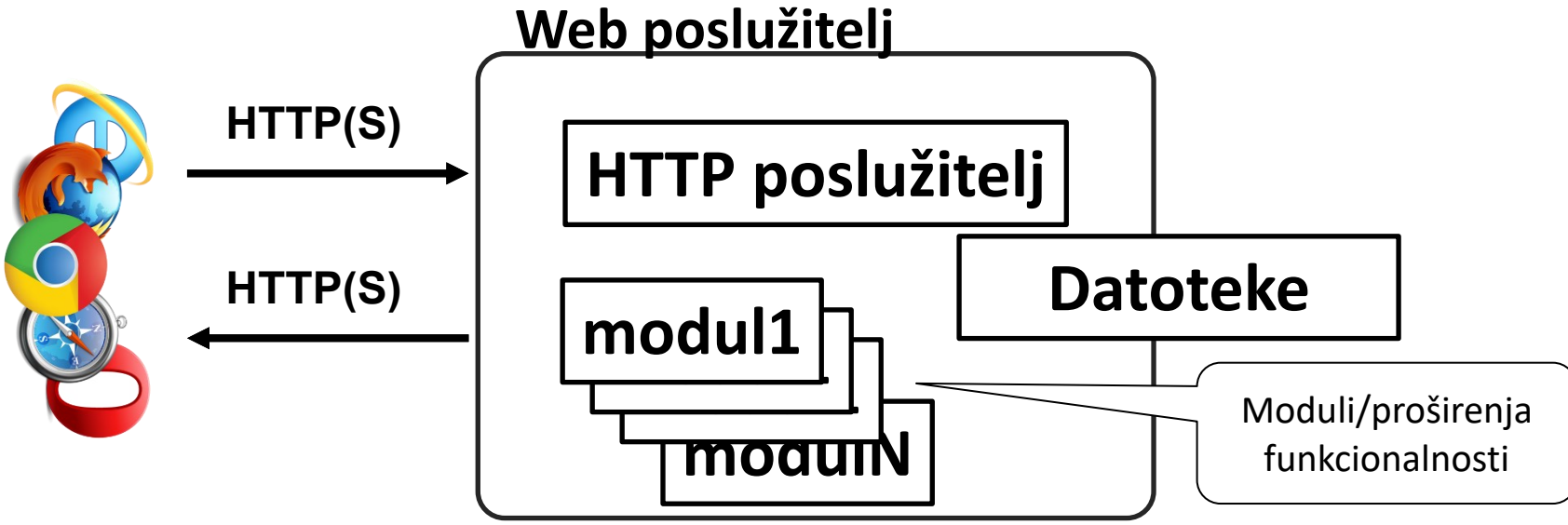
- Vanjski proces s bazenom (*pool*) dretvi radnika
  - Veći stupanj separacije
  - Vanjski proces se brine o upravljanju dretvama



- Itd. razne hibridne opcije…

Web poslužitelj

- Pojam se može shvatiti:
  - hardverski: računalo na kojem je pohranjen softver i podatci (HTML, JS, CSS, itd.) i koje je povezano na mrežu
  - **softverski**: tipično modularni softver koji kontrolira kako korisnici pristupaju raspoloživim resursima. Sadrži minimalno HTTP modul, tj. razumije HTTP i URL-ove te mu se tako može pristupati.



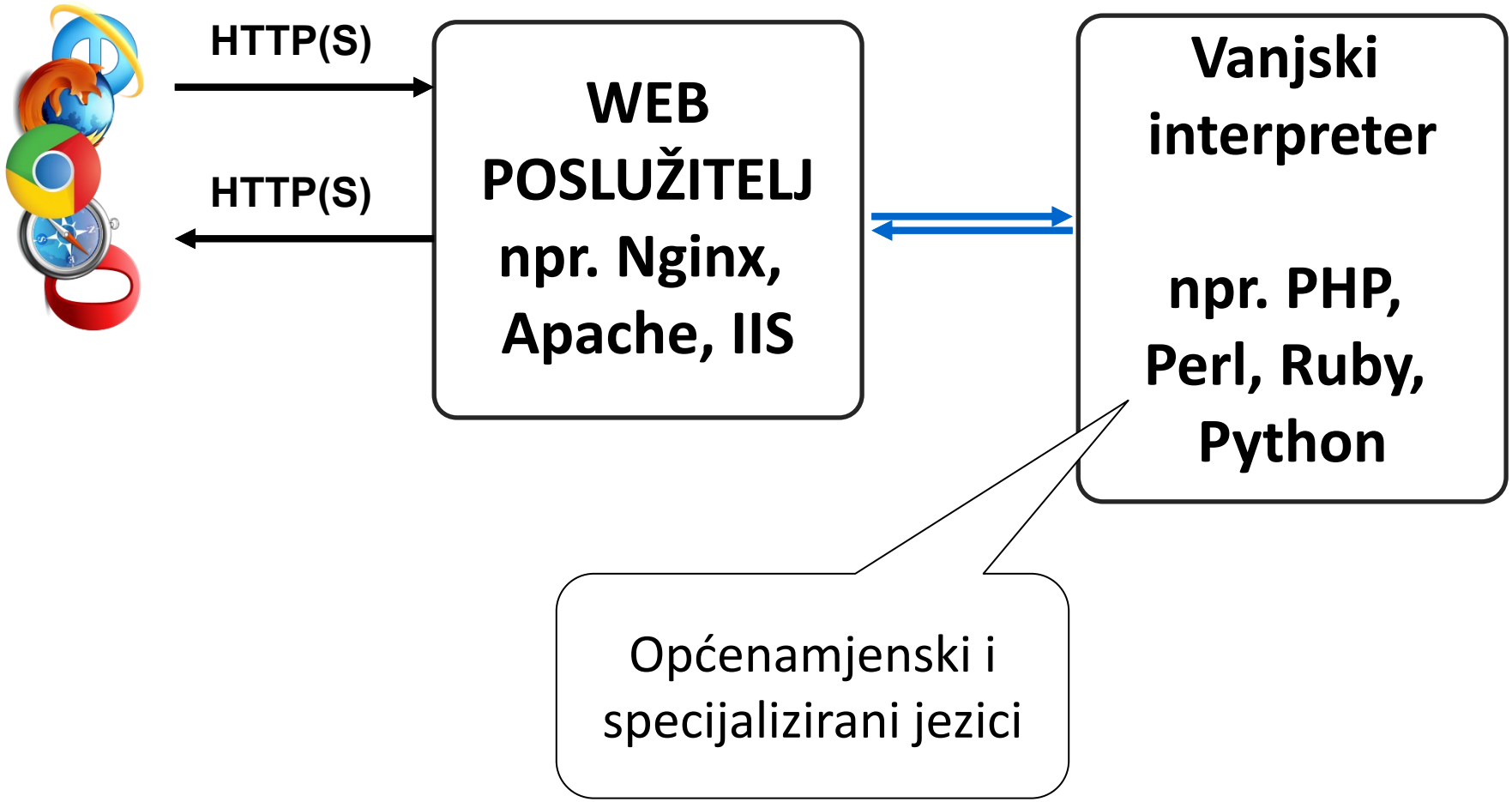
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

Statički i dinamčki web poslužitelji

1. Statički web poslužitelj
   - Osnovni oblik s HTTP poslužiteljem koji isporučuje "statičke" datoteke klijentima.
     - Logički gledano, upravo ono što smo (nedovršeno) napravili u Java primjeru.
     - Prikaz tzv. *web-stranice*
2. Dinamički web poslužitelj
   - Statički poslužitelj obogaćen mogućnostima dinamičkog generiranja sadržaja
   - Prikaz tzv. *interaktivne web-aplikacije*: uključuje HTML koji definira sadržaj, CSS za izgled, a skripte izgrađuju rezultirajuću interaktivnu aplikaciju i odgovaraju na korisnički unos, potencijalno mijenjajući i stilove i sami sadržaj.

Dvije uobičajene arhitekture (1/2)

1. Općenamjenski web poslužitelj s vanjskim interpreterom



Dvije uobičajene arhitekture (2/2)

1. Namjenski aplikacijski poslužitelj s opcionalnim **posrednikom na strani poslužitelja** (*reverse-proxy)* putem općenamjenskog web poslužitelja:

HTTP(S)

HTTP(S)

Web poslužitelj kao **posrednik** npr. Nginx, Apache, IIS

Tipično: HTTP

APLIKACIJSKI POSLUŽITELJ

npr. Tomcat, Node.js/express, IIS…

Opcionalno, npr. kod IIS-a manje uobičajeno

"SSL offloading"

Po nekom procesnom modelu…

RDNK

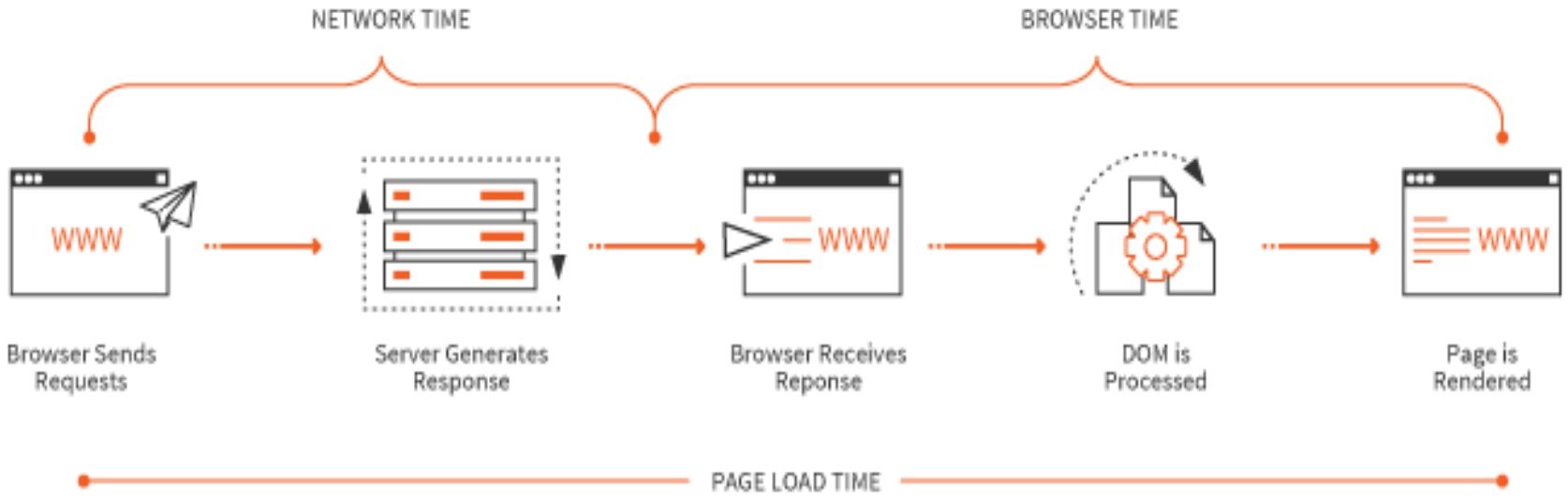
RDNK

Performance web-poslužitelja

- Osnovni parametri kvalitete
  - Vrijeme odziva (engl. *response time*): vremenski period od slanja klijentskog zahtjeva do primitka odgovora. Često se koristi termin *responsiveness* kao mjera kvalitete web poslužitelja.
  - Raspoloživost (engl. *availability*): vjerojatnost da je web poslužitelj dostupan u trenutku *t* te da generira odgovor na korisnički zahtjev.
  - Propusnost (engl. *throughput*): mjeri promet na web poslužitelju ili posredničkom poslužitelju, a izražava se brojem korisničkih zahtjeva u sekundi ili byte/s.

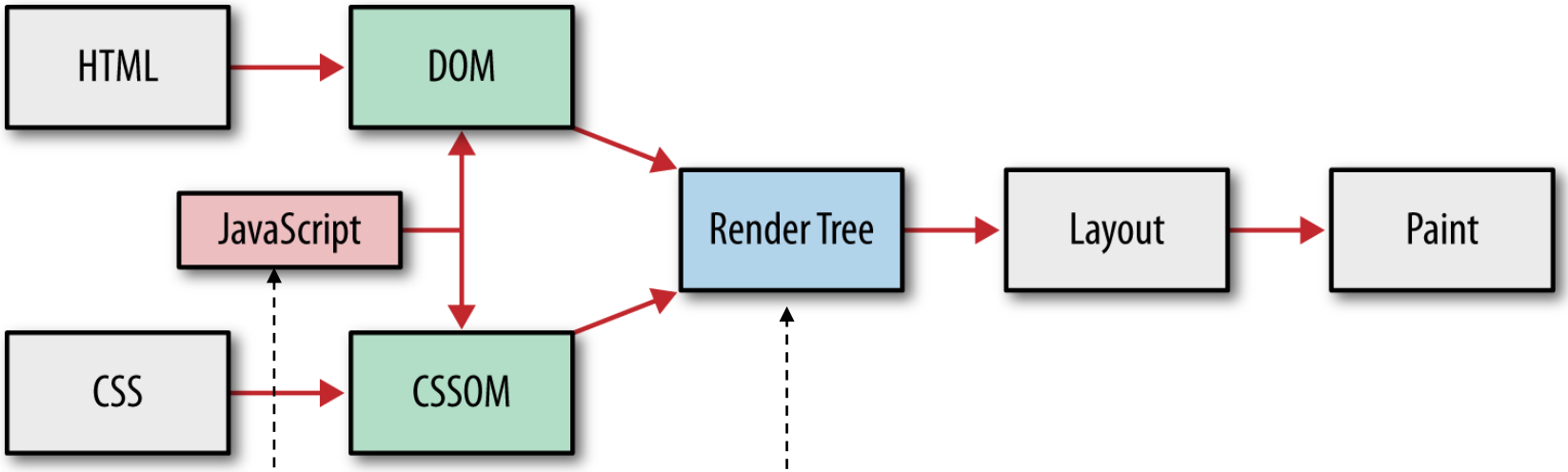
Performance web-poslužitelja

- vrijeme učitavanja stranice (*Page Load Time*, PLT)
  - vrijeme za dohvat svih resursa s poslužitelja i prikaz rezultirajućeg sadržaja u pregledniku
  - uključuje mrežno vrijeme (pogledati slajdove 29 do 31) i vrijeme potrebno pregledniku da obradi i prikaže sav primljeni sadržaj (JavaScript, slike itd.)



Kako preglednik obrađuje sadržaj?

best practice: “styles at the top, scripts at the bottom”

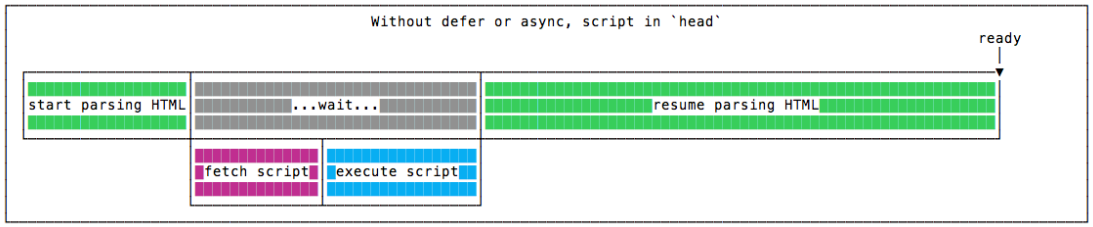


Skripte mogu blokirati obradu DOM-a i CSS-a

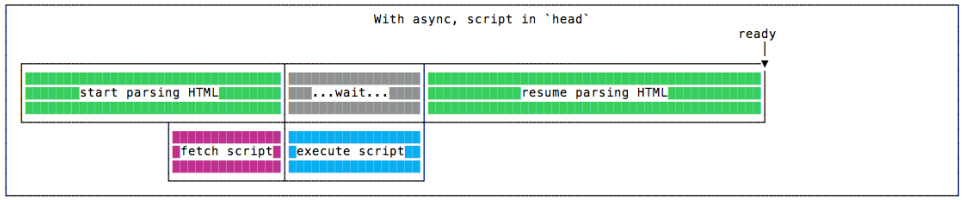
U ovom trenutku preglednik ima dovoljno informacija da posloži sadržaj i prikaže ga u pregledniku

Tri načina učitavanja js skripti

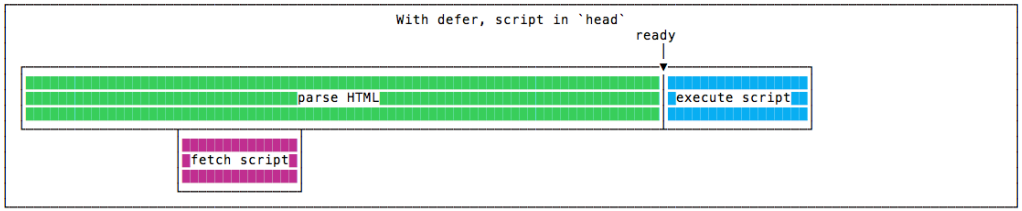
1. **inline/klasično** (podržano u svim preglednicima):
   - `<script src="script.js"></script>`
   - HEAD (najgora opcija – blokira iscrtavanje stranice)
   - BODY (bolje)



2. **async**
   - `<script async src="script.js"></script>`
   - Ne zaustavlja iscrtavanje dok se ne učita, zatim blokira dok se obavlja
   - Redoslijed obavljanja nije pouzdan

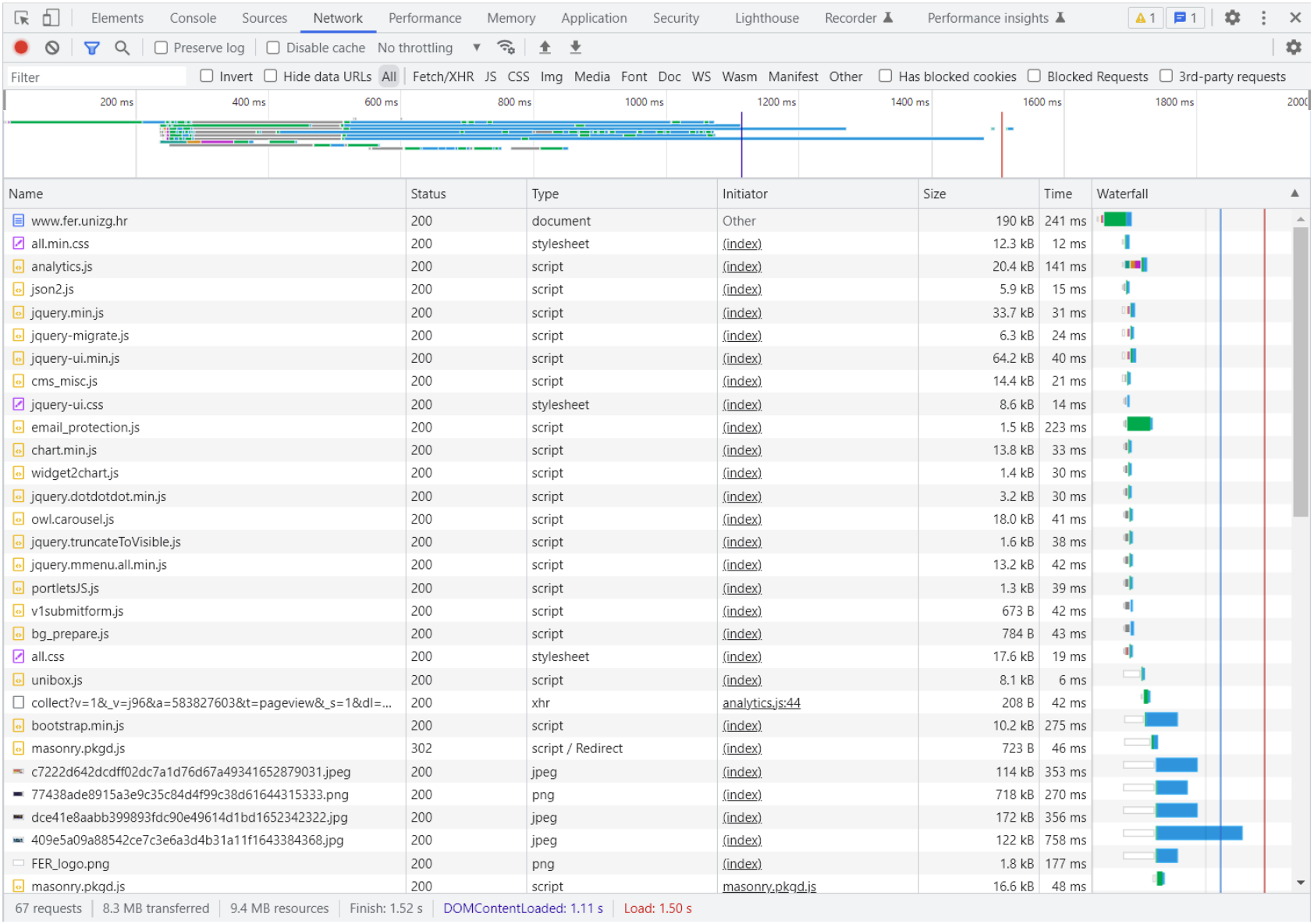


3. **defer**
   - `<script defer src="script.js"></script>`
   - Ne zaustavlja iscrtavanje, obavlja se na kraju redoslijedom kako su navedeni u kodu



Slike preuzete s: https://flaviocopes.com/javascript-async-defer/

Waterfall Chart



Za više informacija o performancama

- https://developer.mozilla.org/en-US/docs/Learn/Performance/What_is_web_performance
- https://learning.oreilly.com/library/view/web-performance-basics/9781492029243/ch01.html

Vježba (video upute)

Instalirajmo nginx+PHP

„hello world”

Par napomena, uz nginx-PHP video

- Nginx možete zaustaviti s: `nginx -s stop`
- Php-cgi možete zaustaviti s Ctrl+C
- Izmjene u nginx.conf:

```
location / {
    root   html;
    index  index.php index.html index.htm ;
}

location ~ \.php$ {
    root           html;
    fastcgi_pass   127.0.0.1:9000;
    fastcgi_index  index.php;
    fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
    include        fastcgi_params;
}
```