HTTP/2 for Developers

How HTTP/2 Works and How It Changes Developer's Life?

Svetlin Nakov

Manager Training and Inspiration

Software University

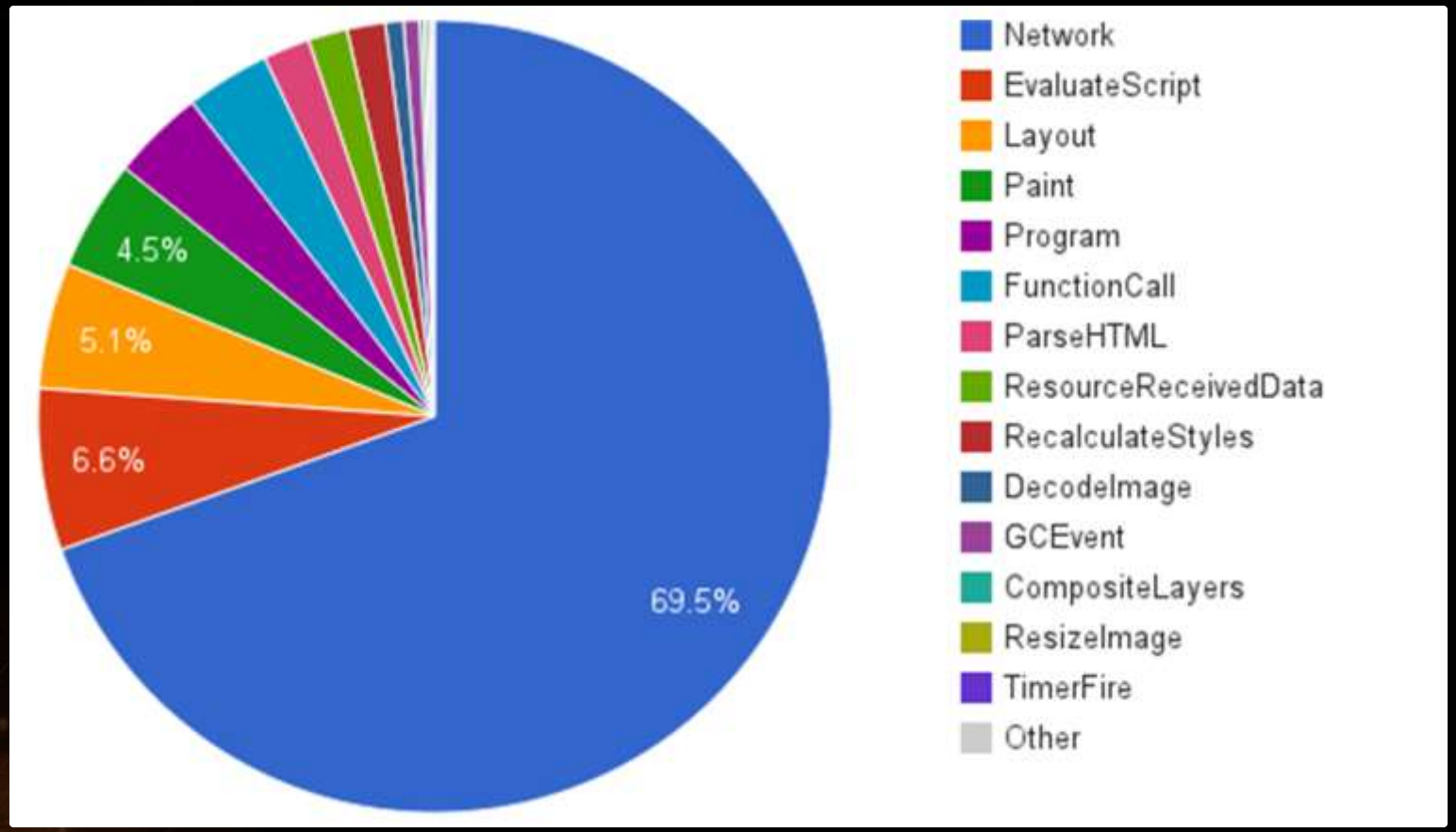
http://softuni.bg

About Me

- Co-founder of the Software University (SoftUni) – https://softuni.bg
  - Trained thousands people in programming and software development
  - Taught hundreds of software development lessons
- 20+ years in software development
  - 15 years in training programmers
- Author of 7 programming books
  - Speaker at hundreds talks, presentations, lessons, courses, seminars, conferences, and technical events
- Blog: http://www.nakov.com

Top 1M Alexa Sites (in 2013) – Telemetry

- Networking takes ~ 70% of the page load time!



- 69.5% of time blocked on network
- 6.6% of time blocked JavaScript
- 5.1% blocked on Layout
- 4.5% blocked on Paint

Source: https://goo.gl/yX8nFN

HTTP/2: The Problem

- Opening a typical site in 2014, e.g. https://softuni.bg

65 requests | 269 KB transferred | Finish: 9.43 s | DOMContentLoaded: 2.45 s | Load: 2.78 s

- HTTP 1.0 supports only one connection per request
  - 65 connections need to be open, then closed
- HTTP 1.1 supports "keep alive"
  - Open connection, download a resource, then another resource, …
  - Works in serial mode: if one resource is slow, page load is slow
- Modern Web needs "a better HTTP protocol"

What is HTTP/2?

- HTTP/2 is protocol designed for
  - Low latency transport of content over the Web
- No change to HTTP semantics
  - It's about how data travels through the wire
- Key new features in HTTP/2
  - Multiplexing: multiple streams over a single connection
  - Header compression: reuse headers from previous requests
  - Sever push: multiple parallel responses for a single request
  - Prioritization: some resources have priorities

HTTP/2 History

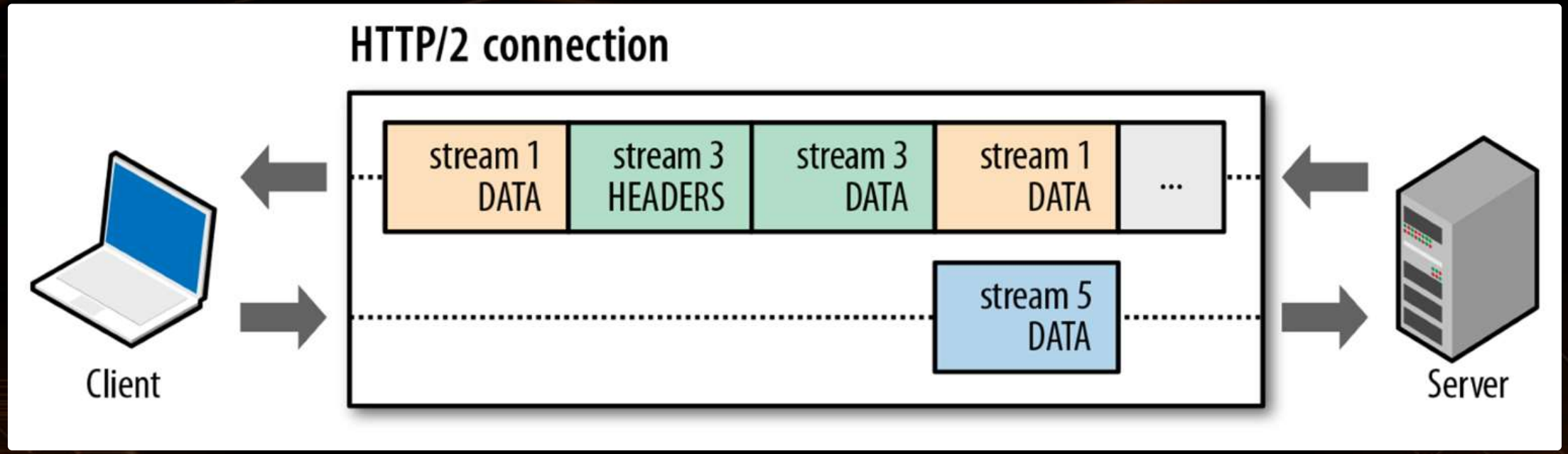
- HTTP 0.9 (1991) – supports simple GET requests
- HTTP 1.0 (1996) – single connection, POST, headers, …
- HTTP 1.1 (1999) – keep-alive connections + pipelining
- SPDY (2009-2010) – connection multiplexing + server push
  - Google developed the SPDY to replace HTTP 1.1 in Chrome and in the Google ecosystem
- HTTP/2 (2014-2015) – de facto standardized SPDY
  - An official standard for the new low-latency HTTP protocol

HTTP/2 Enabled Web Sites – Examples

- Facebook
  - Technically not HTTP/2
  - SPDY/3.1
- Google and its sites (GMail, YouTube, …)
  - Use the latest HTTP/2
  - Combines QUIC + SPDY
- Twitter, Yahoo, Wikipedia

Name	Method	Status	Protocol	Scheme
www.facebook.com	GET	307	http/1.1	http
www.facebook.com	GET	200	h2-14	https
mksngwPB2Xb.css	GET	200	spdy/3.1	https
I5kTXq1bSJZ.css	GET	200	spdy/3.1	https
hxhYuhaB85C.css	GET	200	spdy/3.1	https
MLi5OON7_Yz.css	GET	200	spdy/3.1	https

gcosuc	200	h2	xhr
gen_204?v=3&s=webhp&atyp=csi&imc=3&imn=3&imp=...	204	h2	text/ht...
frame?sourceid=1&hl=en&origin=https%3A%2F%2Fwww....	200	h2	docu...
rs=AGLTcCPfZQTHgHAJ6IAyALxG-i-tYiDctA	200	h2	styles...
rs=AltRSTOKextu3QAZfV0s_SKuW3vYp-SZuA	200	h2	script
spinner_32_794cfa28f324131c58c8934183d55d25.gif	200	h2	gif



Multiplexing

Multiple Streams over Single Connection

HTTP 1.0: Single Connection

Open a TCP connection

HTTP request + HTTP response

Close the TCP connection

- HTTP 1.0 without keep-alive
  - Uses one request per TCP connection
  - Opening / closing a TCP connection is slow
    - Especially in high-latency connections (mobile networks)
- Typical latencies:

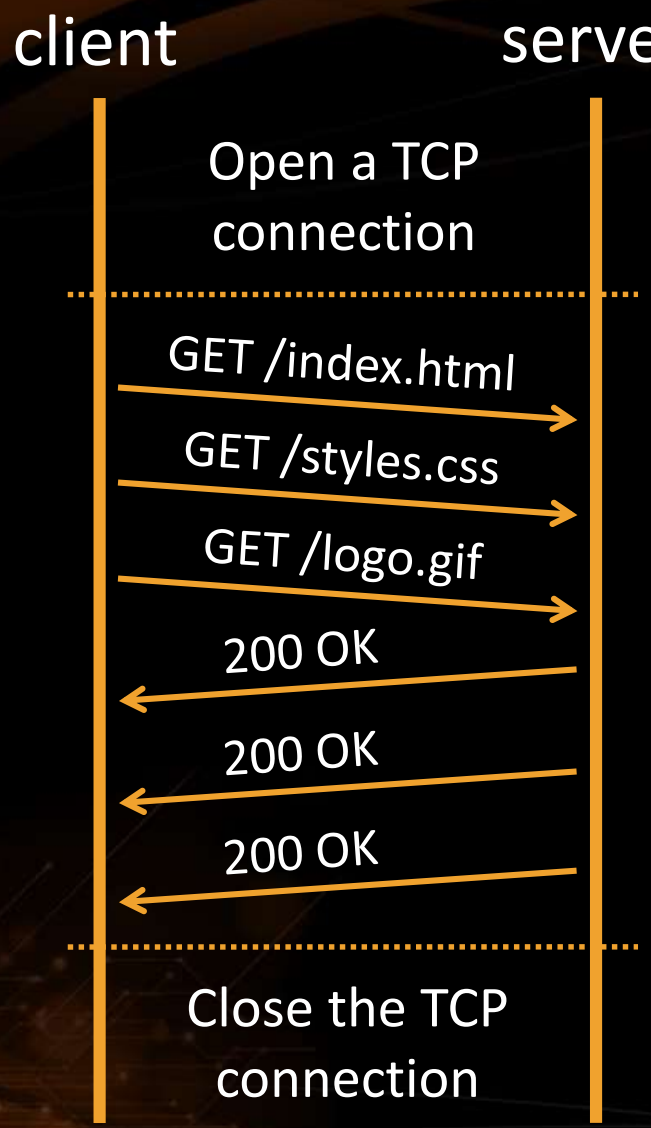
	LTE	HSPA+	HSPA	EDGE	GPRS
Latency	40-50 ms	100-200 ms	150-400 ms	600-750 ms	600-750 ms

HTTP 1.1: Keep Alive

Sequence of HTTP request + HTTP response

- HTTP 1.1 with keep-alive
  - Open a TCP connection
  - Multiple times perform:
    - Send a HTTP request
    - Read a HTTP response
  - Close the TCP connection
- Reuse TCP connections
  - Avoids multiple TCP connection open / close

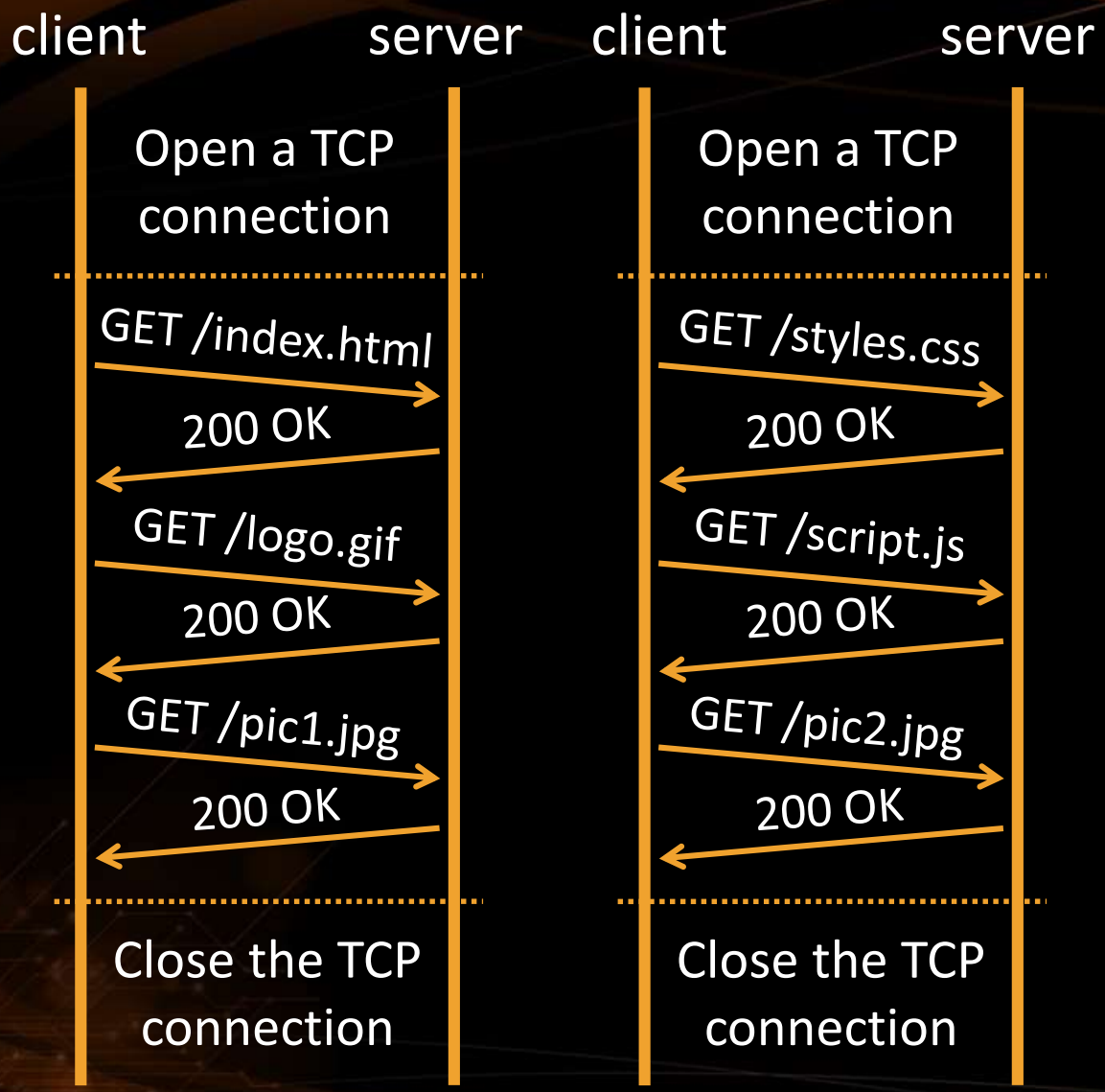
HTTP 1.1: Pipelining



Multiple HTTP requests, followed by their HTTP responses

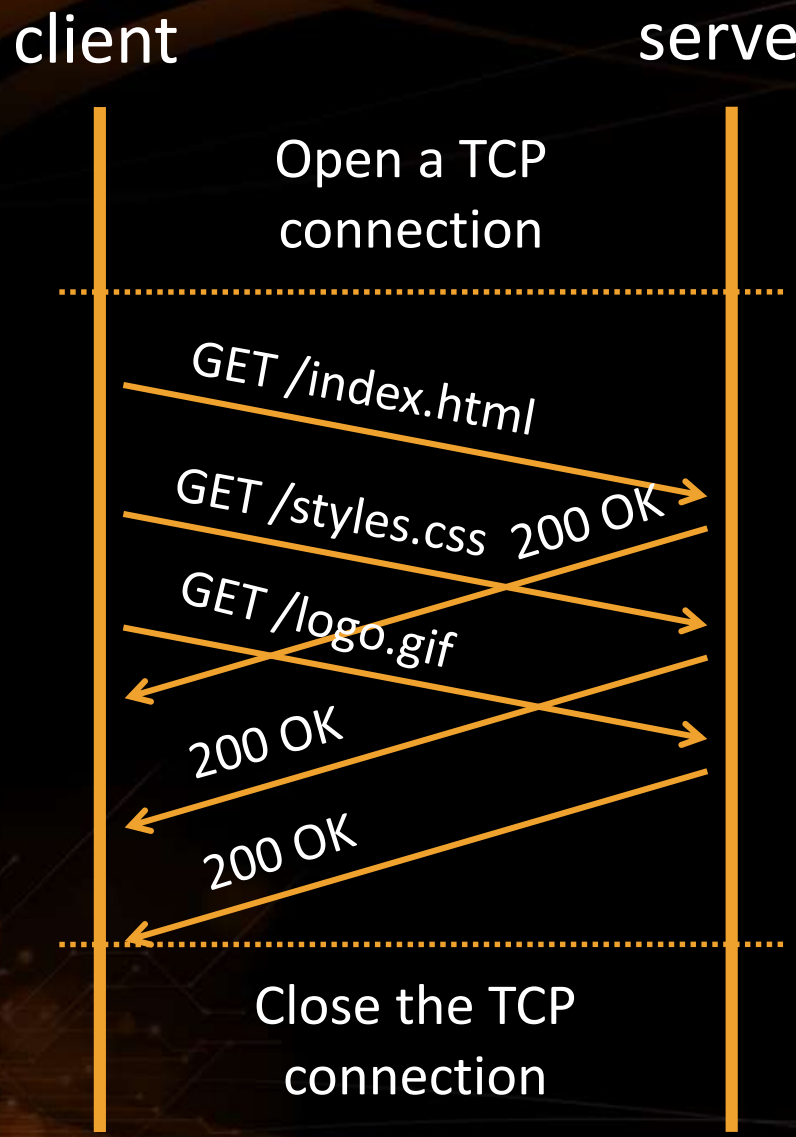
- HTTP 1.1 with pipelining
  - Opens a TCP connection
  - Send multiple HTTP requests (without waiting responses)
  - Get the HTTP responses at once in FIFO order
  - Close the TCP connection
- Head-of-line blocking (HOL) slows down the entire pipeline

HTTP 1.1: Multiple Connections



- Browsers open 4-8 parallel TCP connections to each server
  - Multiple request / response streams run in parallel
  - The initial opening of these connections still has a cost
- The number of connections is limited per domain
  - Domain sharding may improve load time for heavy sites

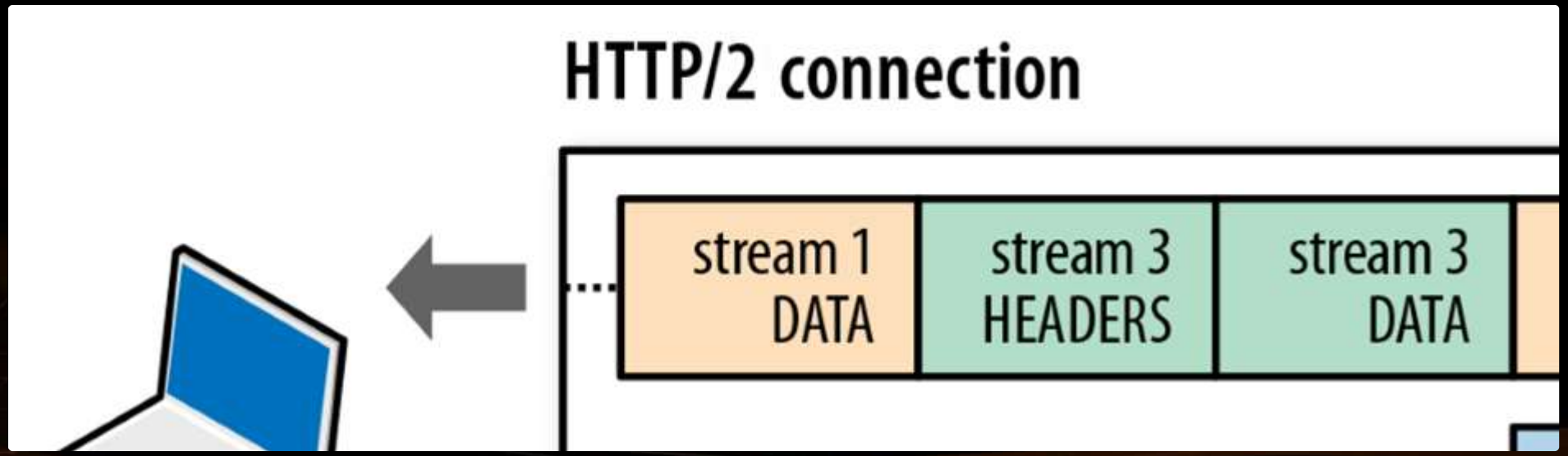
HTTP/2: Multiplexing

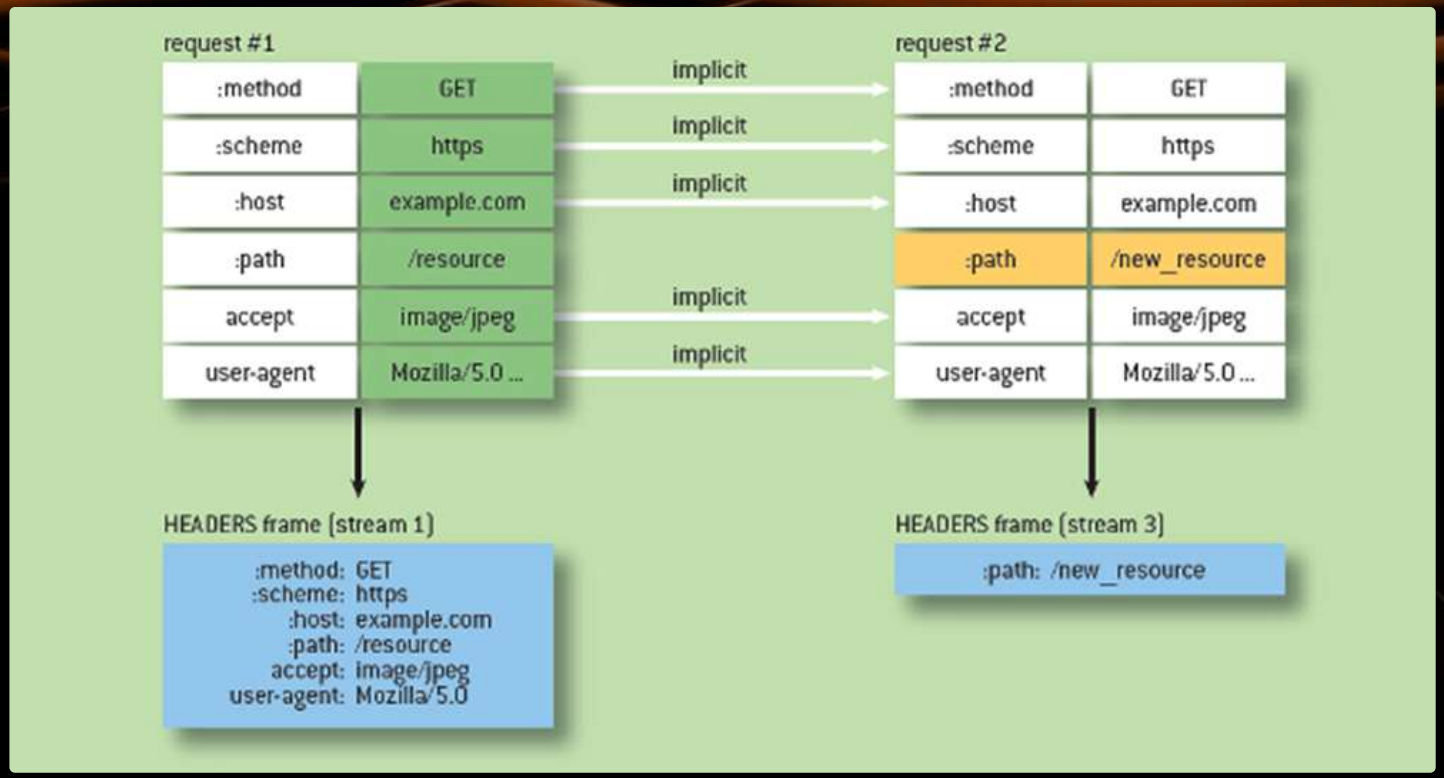


- HTTP/2 uses multiplexing
  - Single TCP connection
  - Multiple parallel streams
  - Prioritization: clients specify each resource importance
  - Out of order responses
- TCP connection is open once
- Resources come though the same connection

How Multiplexing Works?

- HTTP/2 defines **streams** (bidirectional sequence of data)
  - One TCP connection serves multiple parallel streams
- The structure inside HTTP/2 is called a **frame**
  - Frame types: HEADERS, DATA, SETTINGS, PUSH_PROMISE, …





HTTP Header Compression

Reuse Headers from Previous Requests

Typical HTTP Request Holds Long Headers


```
GET https://softuni.bg/ HTTP/1.1
Host: softuni.bg
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml,image/webp,*/*
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/46.0.2490.86 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,bg;q=0.6
Cookie: .AspNet.SoftUniAuth=foQI2JhdsV0jE5PKhLAflcMlB9bl_rN3Rmo3hdR…;
_ga=GA1.2.925942971.1440514837
```

992 bytes

Next Request Holds Nearly the Same Headers


```
GET https://softuni.bg/apply HTTP/1.1
Host: softuni.bg
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml,image/webp,*/*
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/46.0.2490.86 Safari/537.36
Referer: https://softuni.bg/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,bg;q=0.6
Cookie: __RequestVerificationToken=5Eo-1AV9…1; language=bg;
_ga=GA1.2.925942971.1440514837; _gat=1;
.AspNet.SoftUniAuth=foQI2JhdsV0jE5PKhLAflcMlB9bl_rN3Rmo3hdR…
```

1173 bytes (181 bytes new)

Next Request Holds Even More Similar Headers


```
GET https://softuni.bg/contacts HTTP/1.1
Host: softuni.bg
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml,image/webp,*/*
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/46.0.2490.86 Safari/537.36
Referer: https://softuni.bg/
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,bg;q=0.6
Cookie: __RequestVerificationToken=5Eo-1AV9…1; language=bg;
_ga=GA1.2.925942971.1440514837; _gat=1;
.AspNet.SoftUniAuth=foQI2JhdsV0jE5PKhLAflcMlB9bl_rN3Rmo3hdR…
```

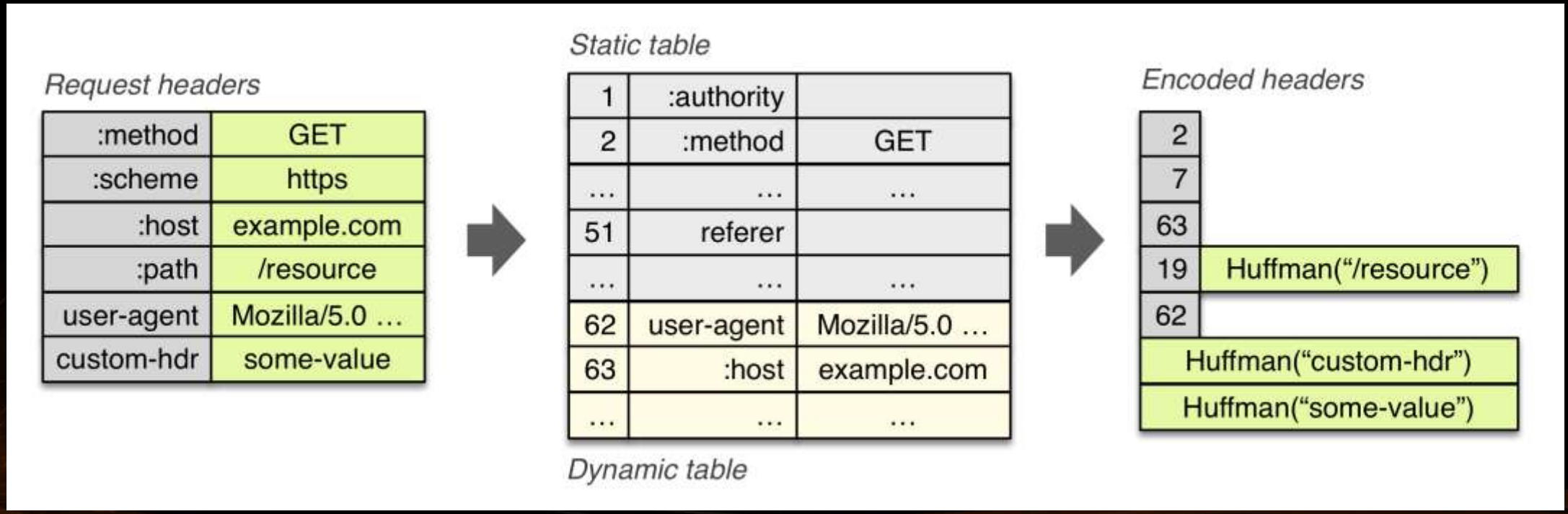
1176 bytes (8 bytes new, headers 100% the same)

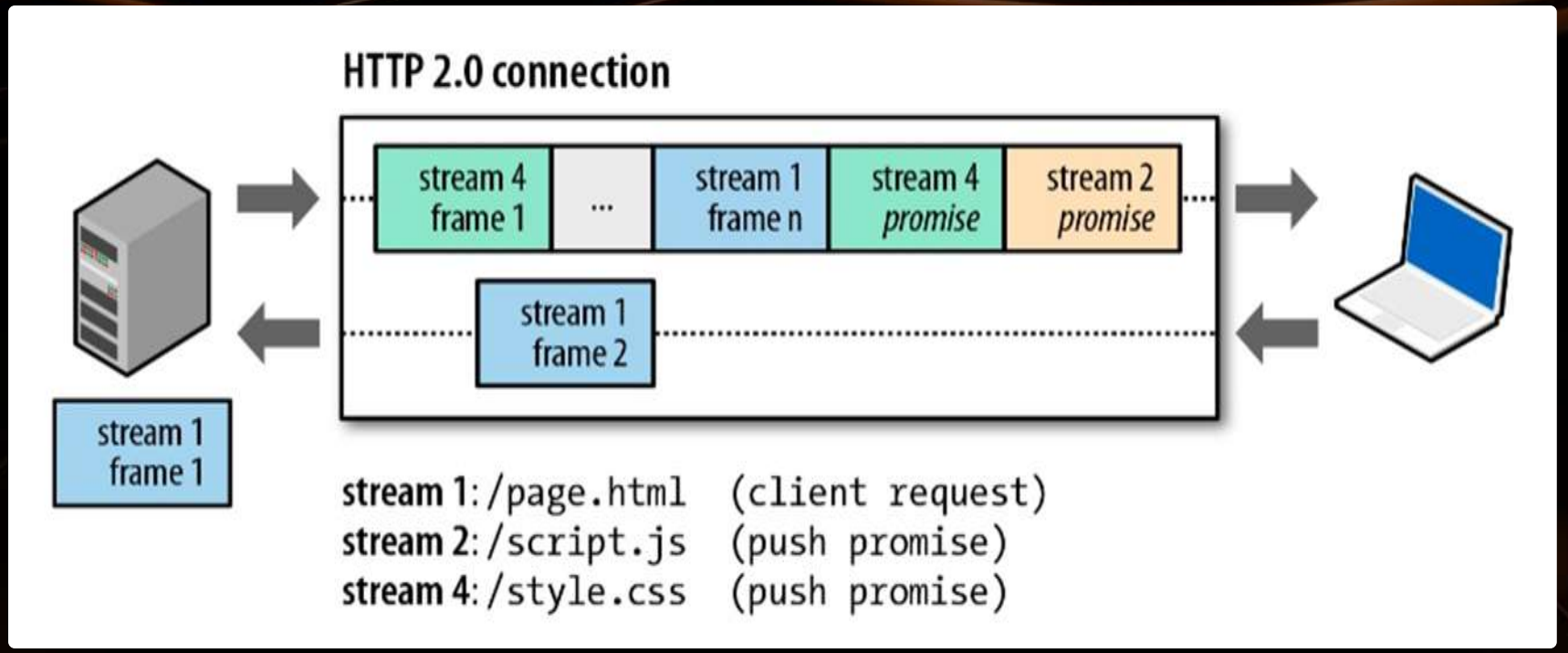
HPACK for HTTP/2 Headers Compression

- HTTP/2 uses a compression standard called **HPACK** (RFC 7541)
  - Compresses the request and response HTTP headers
    - No overhead if headers do not change (e.g. polling requests)
  - Client and server maintain a table of all previous headers
    - Used to efficiently encode previously transmitted values
    - Header tables persist for the entire HTTP 2.0 connection
    - Incrementally updated by both the client and the server
  - Static (predefined) Huffman code used to compress values

HPACK for HTTP/2 Headers Compression

- Request line is split to pseudo headers like **`:method`**, **`:path`**, …
- Static table defines common HTTP headers
- Dynamic table defines HTTP headers in the current HTTP session



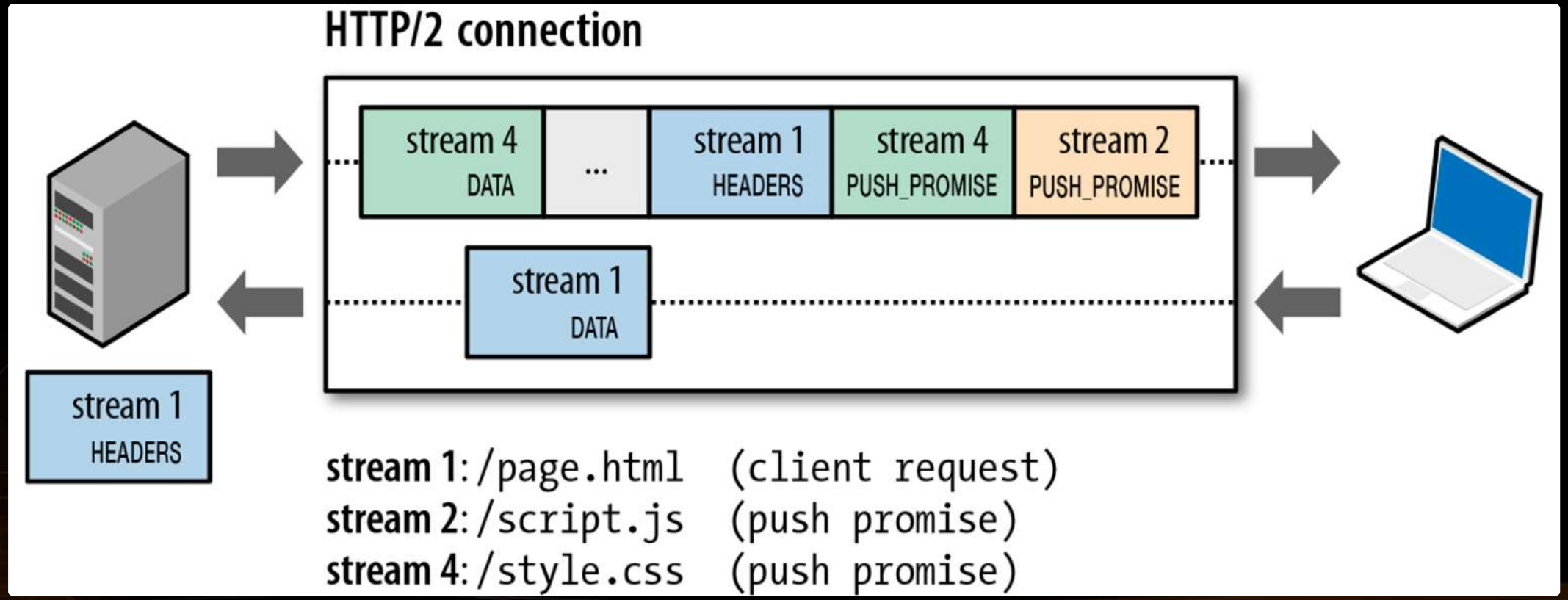


HTTP/2 Server Push

Multiple Parallel Responses for a Single Request

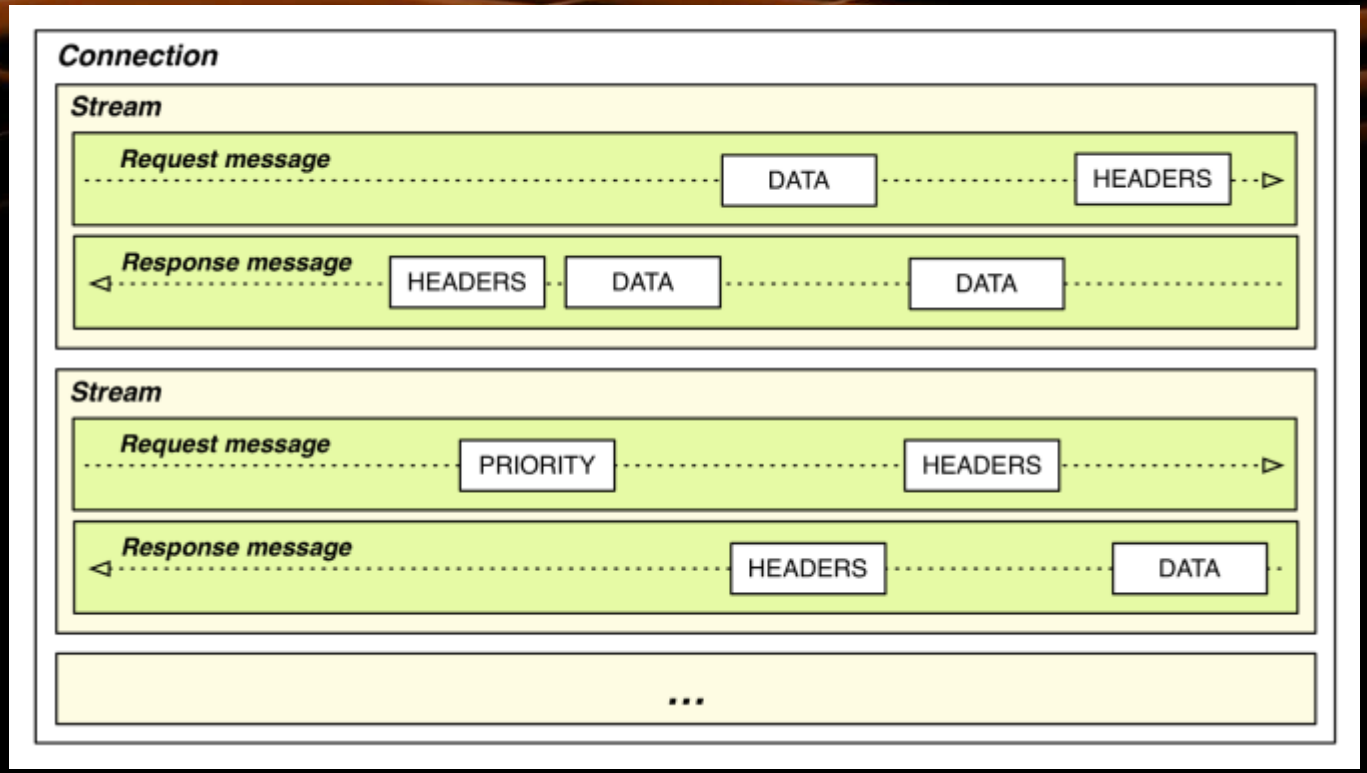
Server Push in HTTP/2

- HTTP/2 servers can send multiple responses for a single client request (push additional resources with the requested resource)



Push Promises

- **PUSH_PROMISE** frames in HTTP/2 allows the server to send additional resources
  - First headers are sent, so clients can decide what to do
  - Clients can decline pushed resources (via a **RST_STREAM** frame)
    - E.g. when the pushed resource is already in the cache
- Pushed resources can be cached and reused later
- Pushed resources can be multiplexed alongside other resources
- Pushed resources can be prioritized by the server

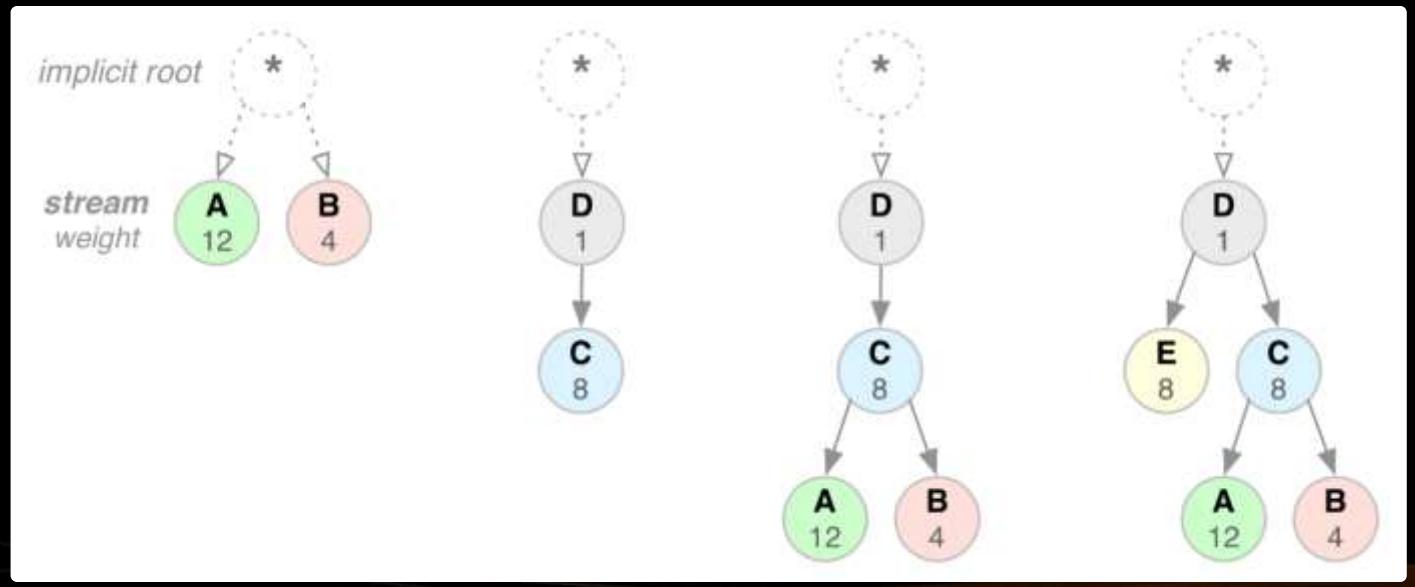


Prioritization and Flow Control

Weights, Dependencies, Flow Control

Stream Weight and Dependency

- HTTP/2 defines stream prioritization by weight + dependency
  - Each stream may be assigned a weight between 1 and 256
  - Each stream may be given a dependency on another stream
- The "prioritization tree" defines the preferred order of resources

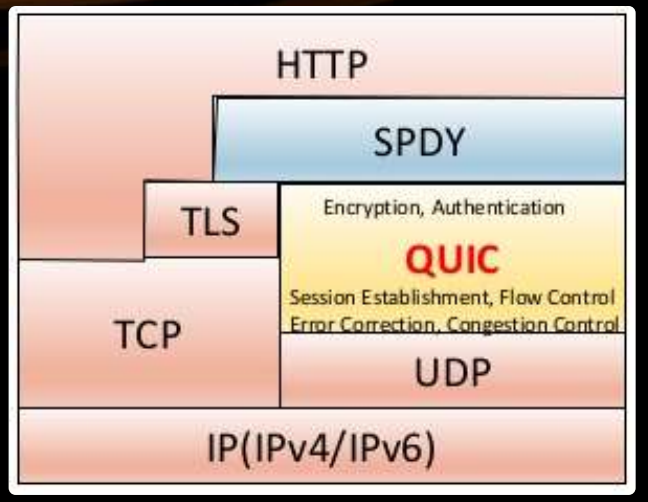


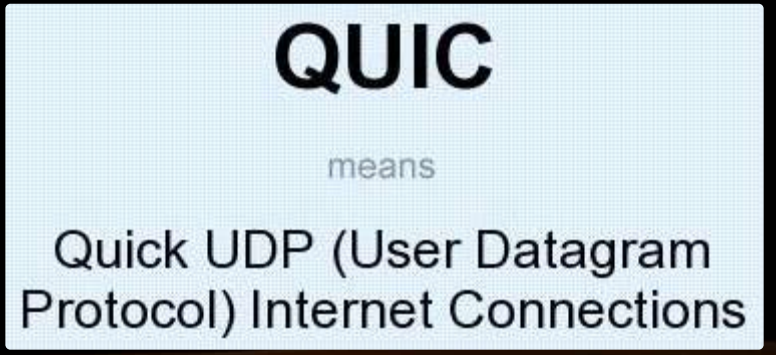
Flow Control in HTTP/2

- What is flow control?
  - Managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver
- Example:
  - User watches a video and presses the "pause" button
- For each HTTP/2 stream and for the entire connection
  - Receivers can specify how many **DATA** bytes is ready to receive
- **`WINDOW_UPDATE`** frames configure the window size

The QUIC Protocol

- Some browsers use the QUIC protocol
  - QUIC == Quick UDP Internet Connections
  - Low-latency UDP-based transport layer protocol
  - Designed to replace TCP for better performance
  - Build-in multiplexing and TLS-equivalent security
  - Still experimental, supported in Chromium
- Learn more at:
  - https://www.chromium.org/quic
  - https://en.wikipedia.org/wiki/QUIC





Using HTTP/2

How to Switch to HTTP/2?

How to Establish HTTP/2 Connection?

- There is **no** such thing as:
  - **http2://mysite.com**
- HTTP/2 uses a standard https:// connection
  - With TLS ALPN (Application Layer Protocol Negotiation)
  - ALPN allows the applications to negotiate which protocol should be used over a secure connection without round trips
  - Most browsers do not support HTTP/2 without SSL
- Browsers check for HTTP/2 support during the SSL handshake

HTTP/2 Connection Upgrade

- HTTP 1.1 connection could be upgraded to HTTP/2:

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url-encoded HTTP/2 SETTINGS>
```

- TLS is not mandatory for HTTP/2 (as defined in RFC 7540)
  - But most Web browsers refuse to implement clear text HTTP/2
- Non-browser client apps can still use clear-text HTTP/2

SOFTWARE UNIVERSITY FOUNDATION

Fiddler
HTTP Debugging Proxy

WIRESHARK

HTTP/2 Tools

How It Changes Developer's Life?

HTTP/2 is a Binary Protocol

- HTTP 1.1 was simple to test / debug / trace
  - It is text-based protocol, human-readable
- HTTP/2 will require tools
  - It is binary protocol, not human-readable
  - It runs over TLS tunnel, so sniffing is complicated

```
Command Prompt - ssh root@svn.softuni.org
root@svn:~# telnet softuni.org 80
Trying 98.124.199.71...
Connected to softuni.org.
Escape character is '^]'.
GET / HTTP/1.1
Host: softuni.org

HTTP/1.1 302 Found
Date: Sat, 21 Nov 2
Content-Type: text/
```

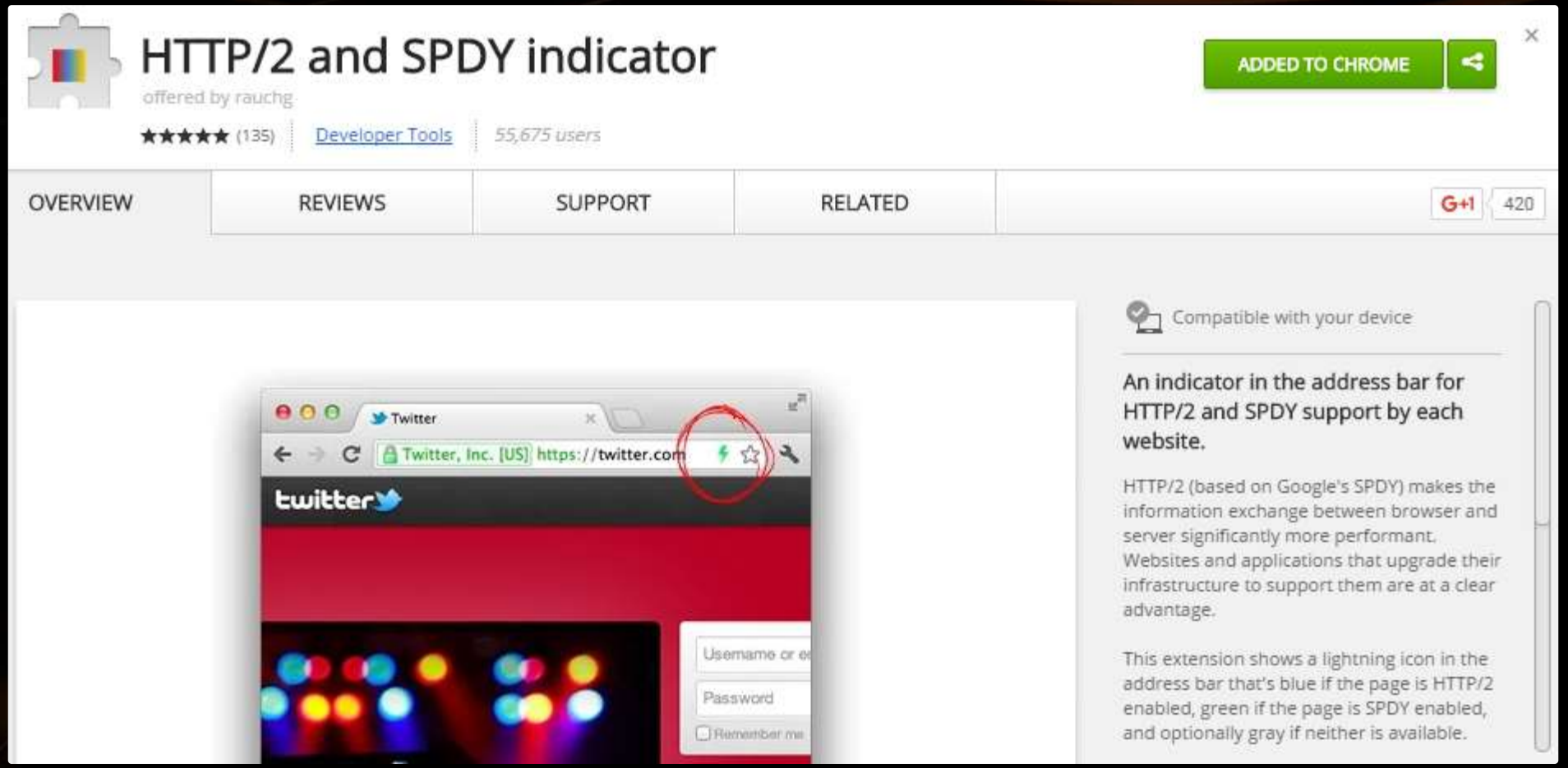
h2i

```
$ h2i google.com
Connecting to google.com:443 ...
Connected to 74.125.224.41:443
Negotiated protocol "h2-14"
[FrameHeader SETTINGS len=18]
  [MAX_CONCURRENT_STREAMS = 100]
  [INITIAL_WINDOW_SIZE = 1048576]
  [MAX_FRAME_SIZE = 16384]
[FrameHeader WINDOW_UPDATE len=4]
  Window-Increment = 983041
```

Tools Not Working with HTTP/2

- **Wireshark** can sniff HTTP/2
  - But cannot decrypt TLS traffic
    - No out-of-the-box man-in-the-middle proxy
    - Needs the SSL private key for the remote site
  - In fact Wireshark is not of help for HTTP/2 Web developers
- **Fiddler** does not support HTTP/2
  - Wait for Microsoft to implement TLS ALPN in .NET Framework
  - It decrypts HTTPS through elegant man-in-the-middle proxy

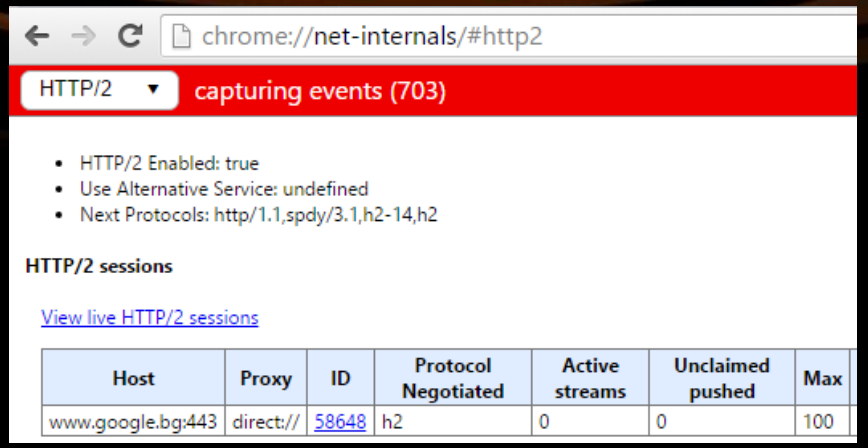
HTTP/2 and SPDY Indicator for Chrome

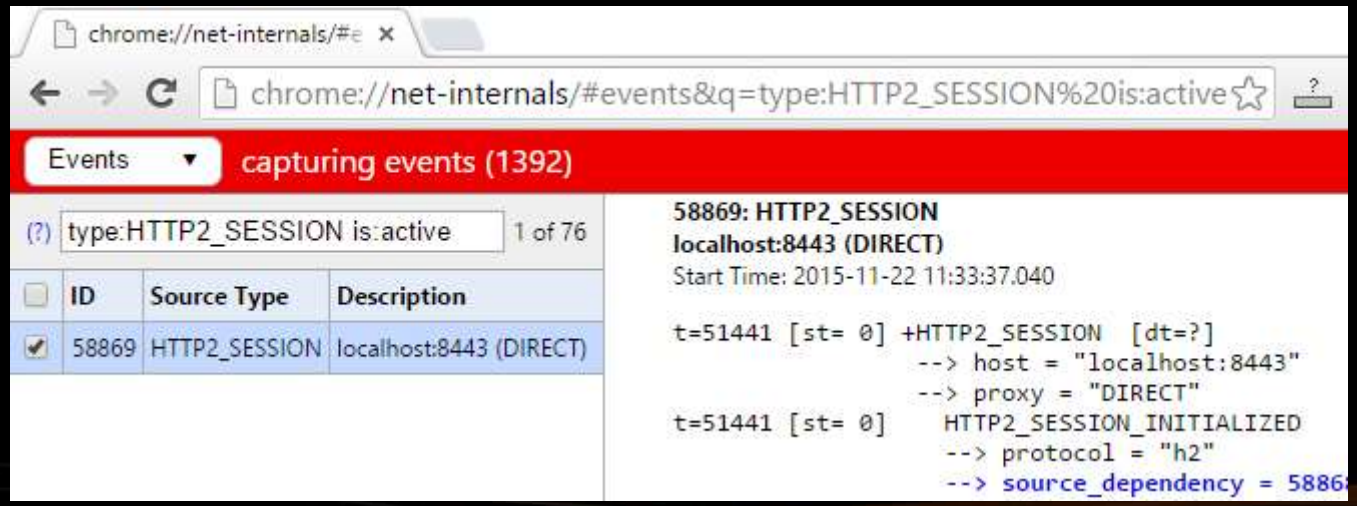


- Indicates HTTP/2, SPDY, QUIC, versions, etc.

Chrome HTTP/2 Tools

- Chrome HTTP/2 info page
  - chrome://net-internals/#http2
- View all HTTP/2 sessions
  - chrome://net-internals/#events&q=type:HTTP2_SESSION

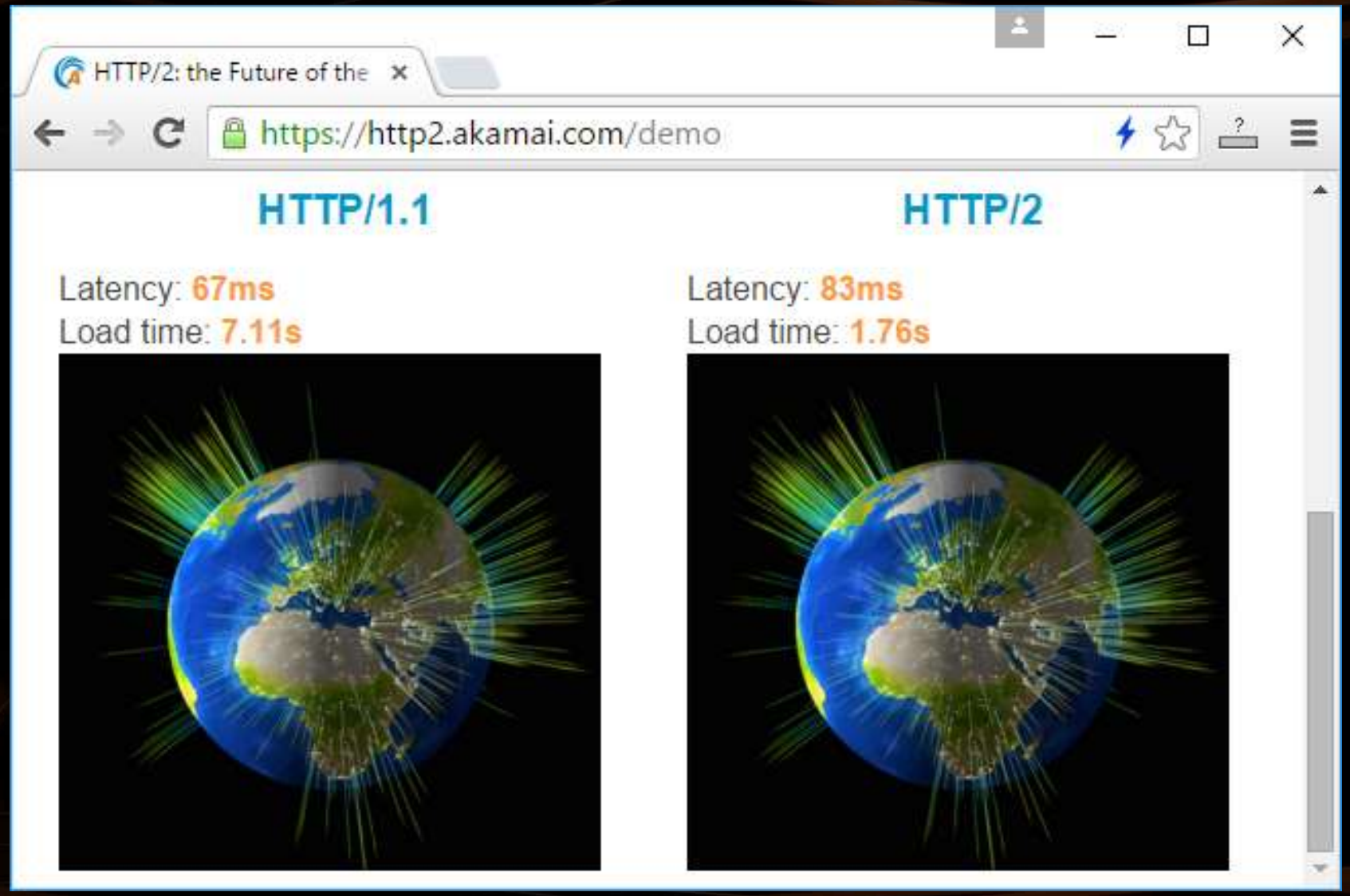




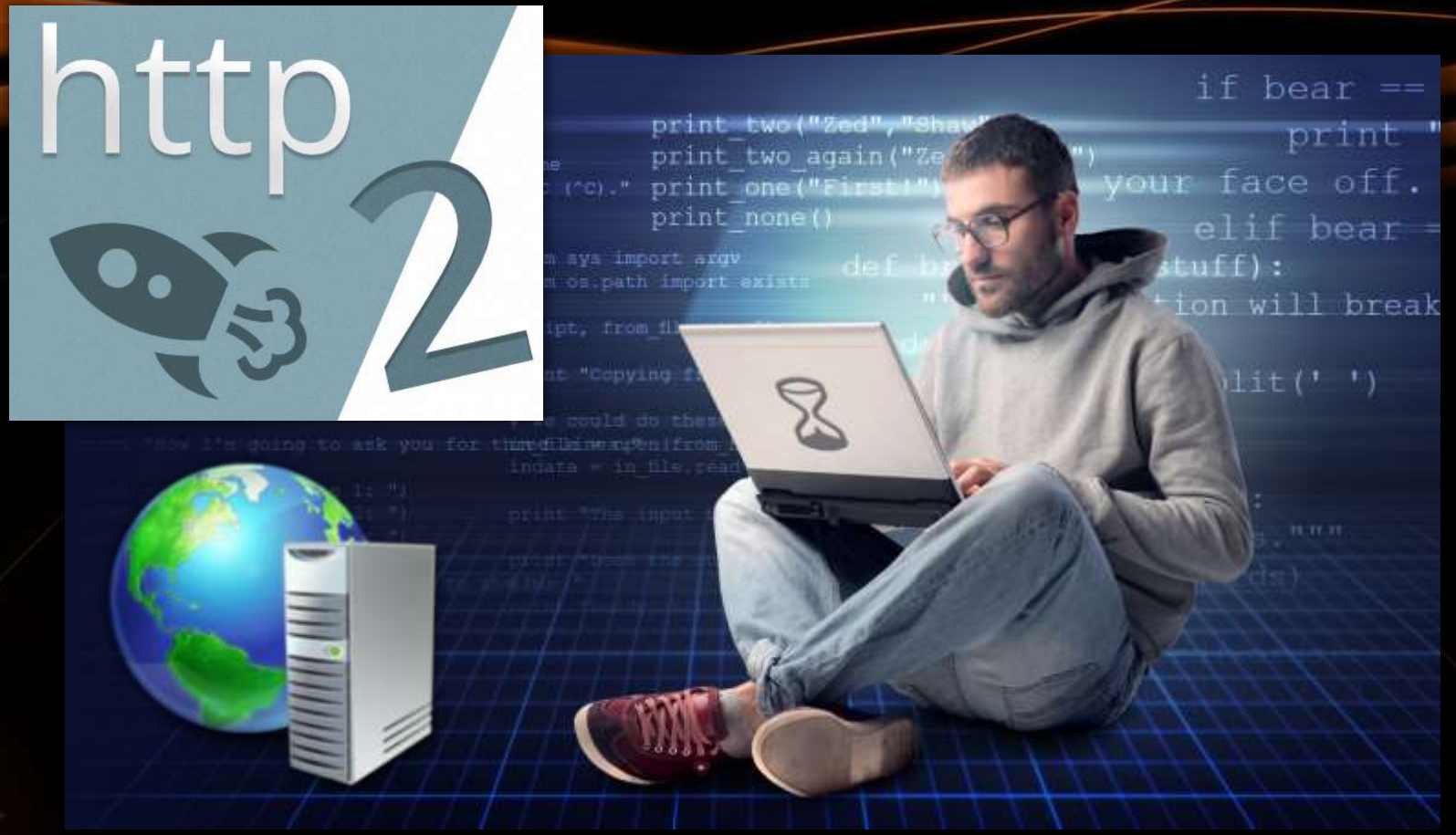
H2I

- **H2I** is an interactive HTTP/2 ("h2") console debugger
  - Something like **`telnet`** to HTTP/2 server
- https://github.com/bradfitz/http2/tree/master/h2i
  - Written in GO
- Send / receive raw HTTP/2 frames
  - Send **`PING`**, **`SETTINGS`**, **`HEADERS`** frames
  - Receive any type of frame

Akamai HTTP/2 Performance Test



SOFTWARE UNIVERSITY FOUNDATION



HTTP/2 from Dev Perspective

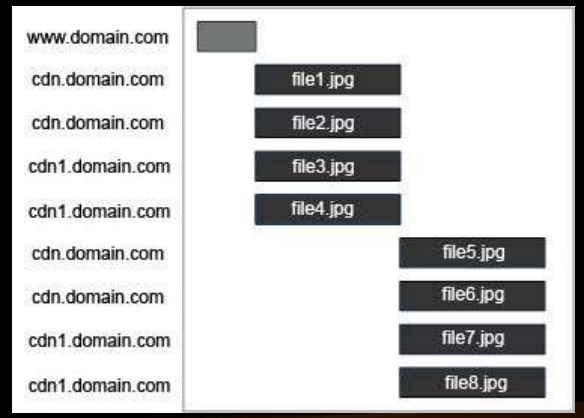
How It Changes Developer's Life?

Say "Goodbye" To Many Hacks

- No more image sprites
  - Use separate images
- No more CSS / JavaScript combining
  - Use separate CSS / JS files + server push
- No more domain sharding
  - Still might boost performance by distributing the server-side load
- HTTP APIs can be finer-grained without sacrificing performance

YUI Compressor

JS Compress



Lots of Tweaking for Developers / Admins

- Prioritization
  - Developers may specify resource prioritization for faster page load
  - Web servers / browsers may automatically set priorities
- When to push?
  - Developers may push resources for faster page load
  - Web servers may push resources transparently of developers
- How to optimize the flow control ?
  - Developers may change the flow-control of data streams

Jetty and HTTP/2

HTTP/2 for Java Developers

Jetty and HTTP/s

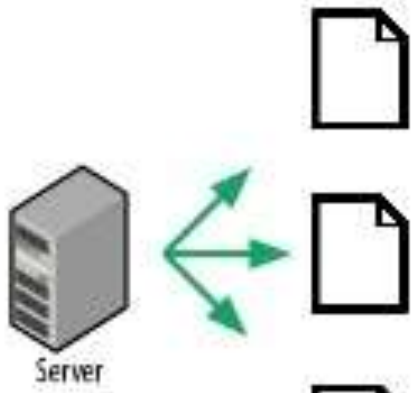
- Jetty is a Java HTTP (Web) server and Servlet engine
  - Developed under the Eclipse open-source project
  - Supports HTTP/2, WebSocket, OSGi, JMX, JNDI, JAAS
- Enable HTTP/2 in Jetty:

```
$ java -jar $JETTY_HOME/start.jar --add-to-startd=http2
```

```
$ java -jar $JETTY_HOME/start.jar
…
2015-06-17 14:16:12.549:INFO:oejs.ServerConnector:main: Started
ServerConnector@34c9c77f{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
2015-06-17 14:16:12.782:INFO:oejs.ServerConnector:main: Started
ServerConnector@711f39f9{SSL,[ssl, alpn, h2, h2-17, http/1.1]}{0.0.0.0:8443}
…
```

Jetty Smart Push

- Jetty allows configuring a server push strategy
- **`ReferrerPushStrategy`** auto-learns dependencies from referrer



1. **Server observes incoming traffic**
   a. *Build a dependency model based on Referer*
   b. *e.g. index.html → {style.css, app.js}*
2. **Server initiates push for learned dependencies**
   a. *new client → GET index.html*
   b. *server → Push style.css, app.js*

Lots of room for experimentation + innovation!

Jetty HTTP/2 Client

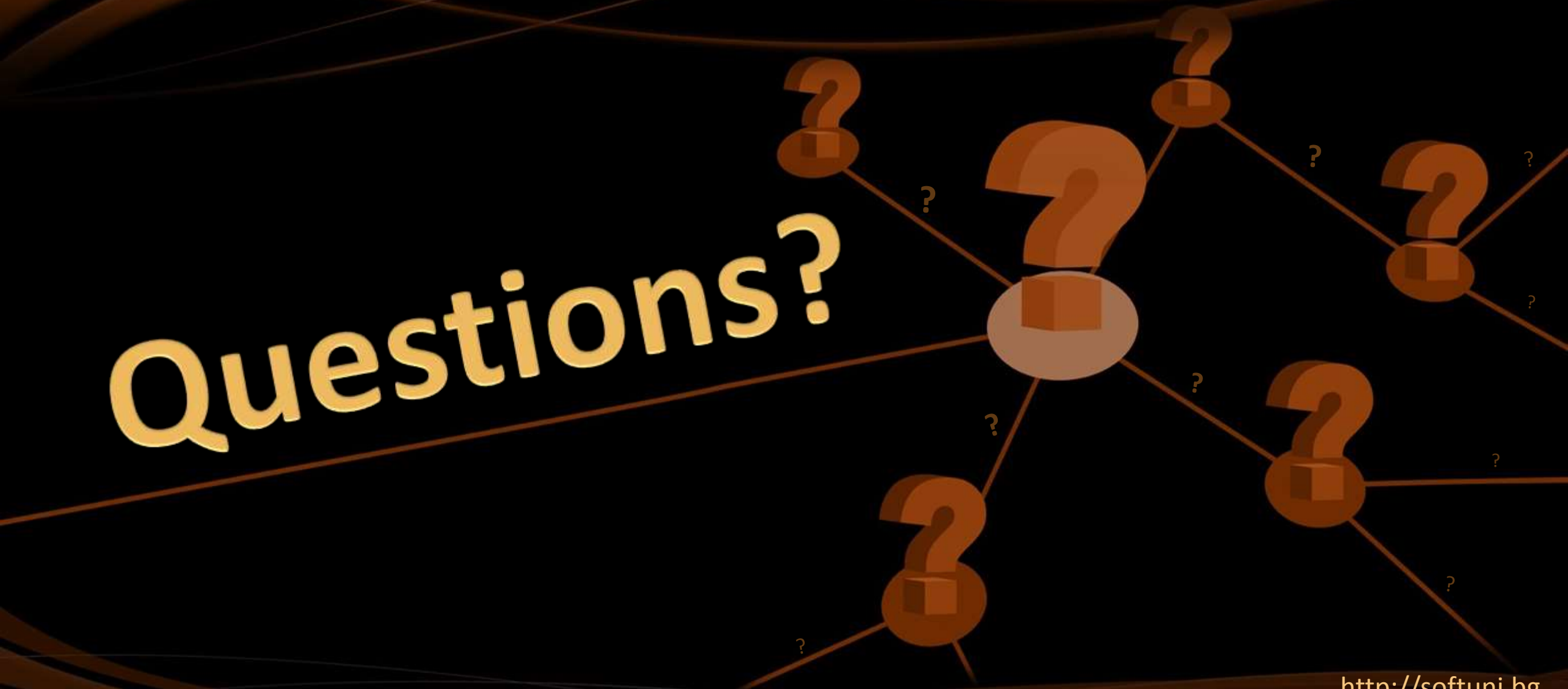
- Jetty provides **`http2-client`**
  - Implementation of HTTP/2 client with a low level HTTP/2 API, dealing with HTTP/2 streams, frames, etc.

```
SslContextFactory sslContextFactory = new SslContextFactory();
HttpClient httpClient = new HttpClient(sslContextFactory);
httpClient.start();
ContentResponse response =
  httpClient.GET("https://http2.akamai.com");
…
```

  - Learn more at http://www.eclipse.org/jetty/documentation/current/http-client-api.html

HTTP/2 for Developers

SOFTWARE UNIVERSITY FOUNDATION



License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg