

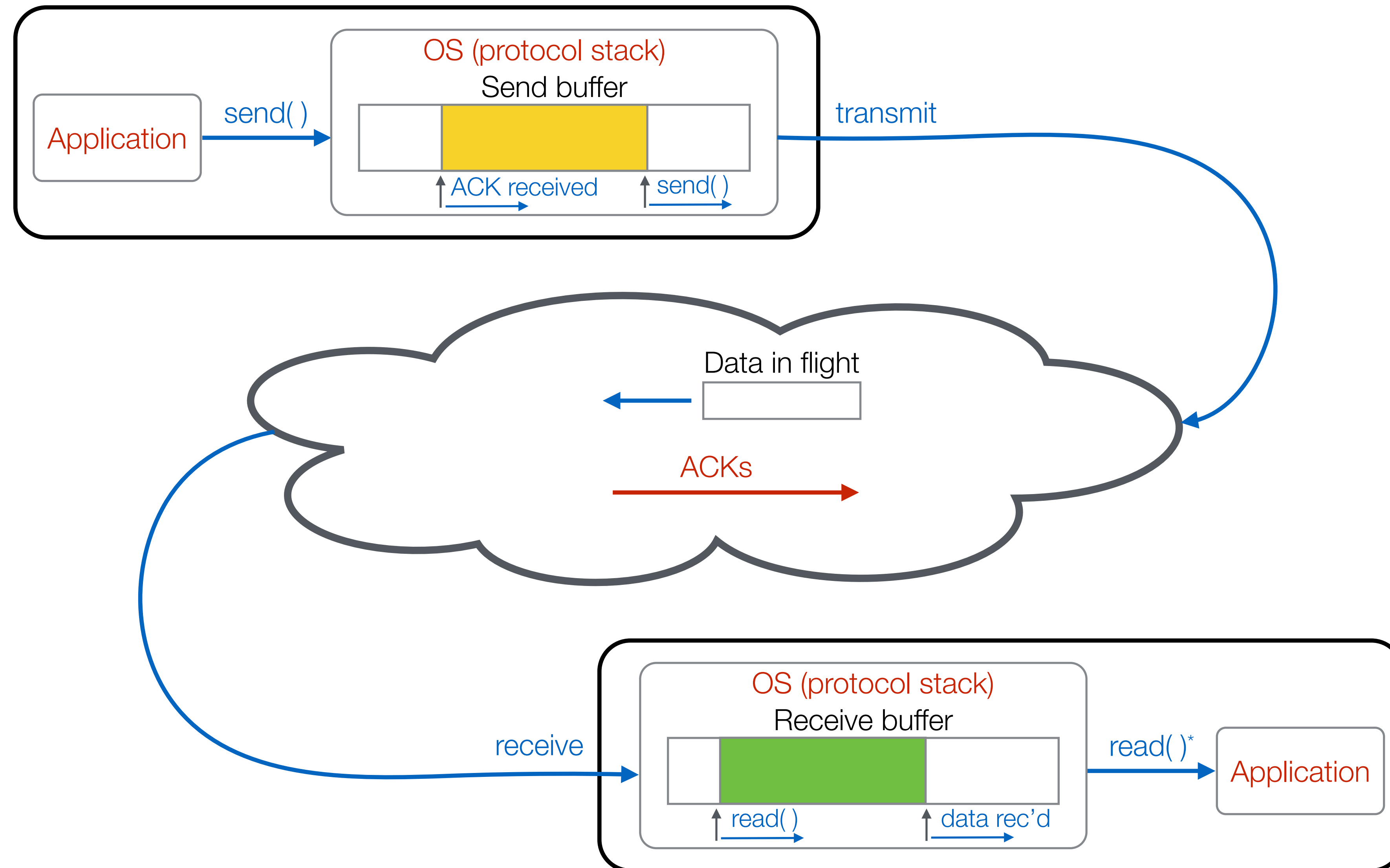
CS 725/825 & IT 725

Lecture 10

Application Layer

October 13, 2026

Flow of data



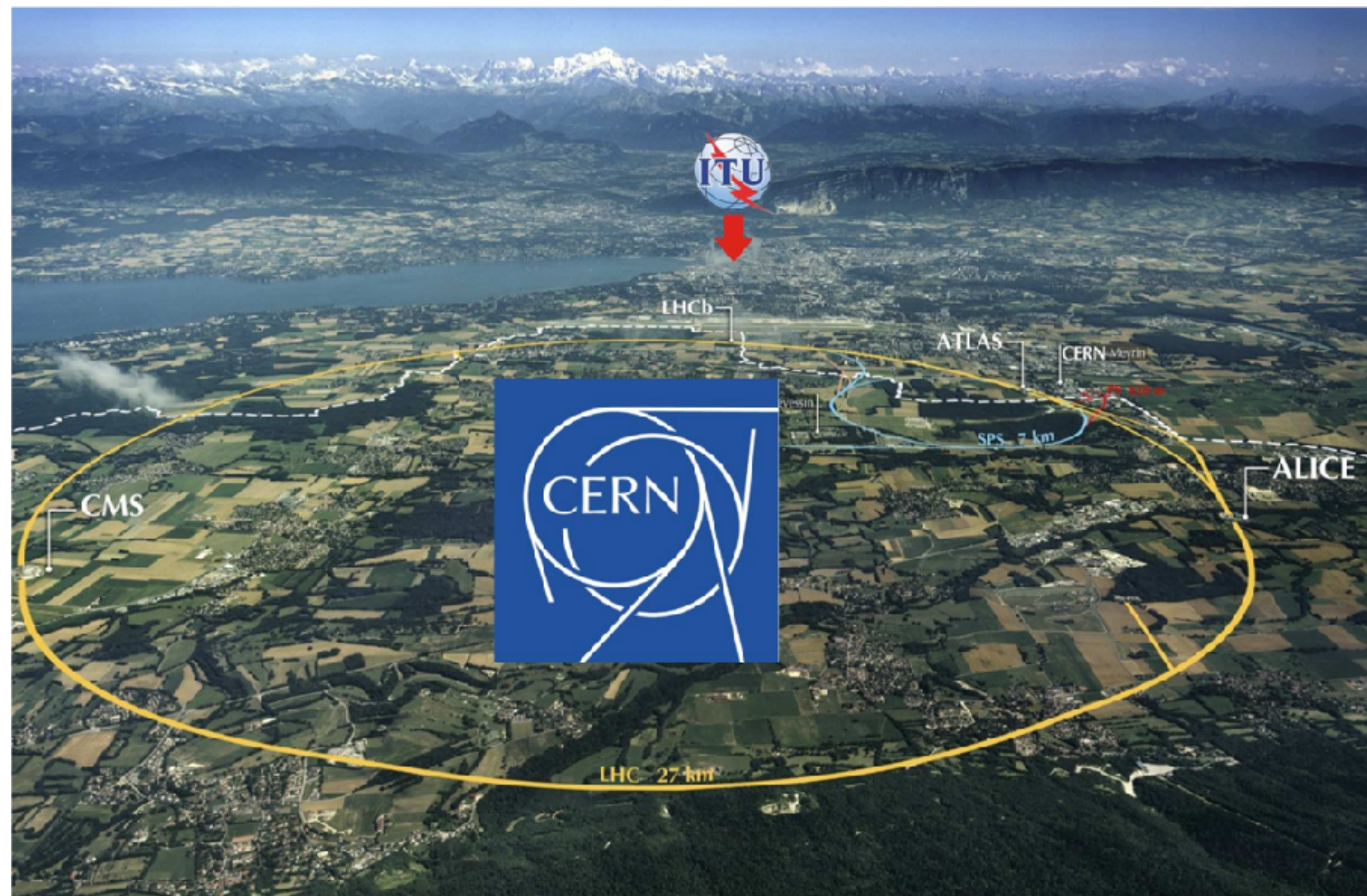
(*) many APIs call the `read()` operation "receive" (eg: `recv()`), `read` is used here to avoid confusion with receiving data on an interface

HTTP/HTML History

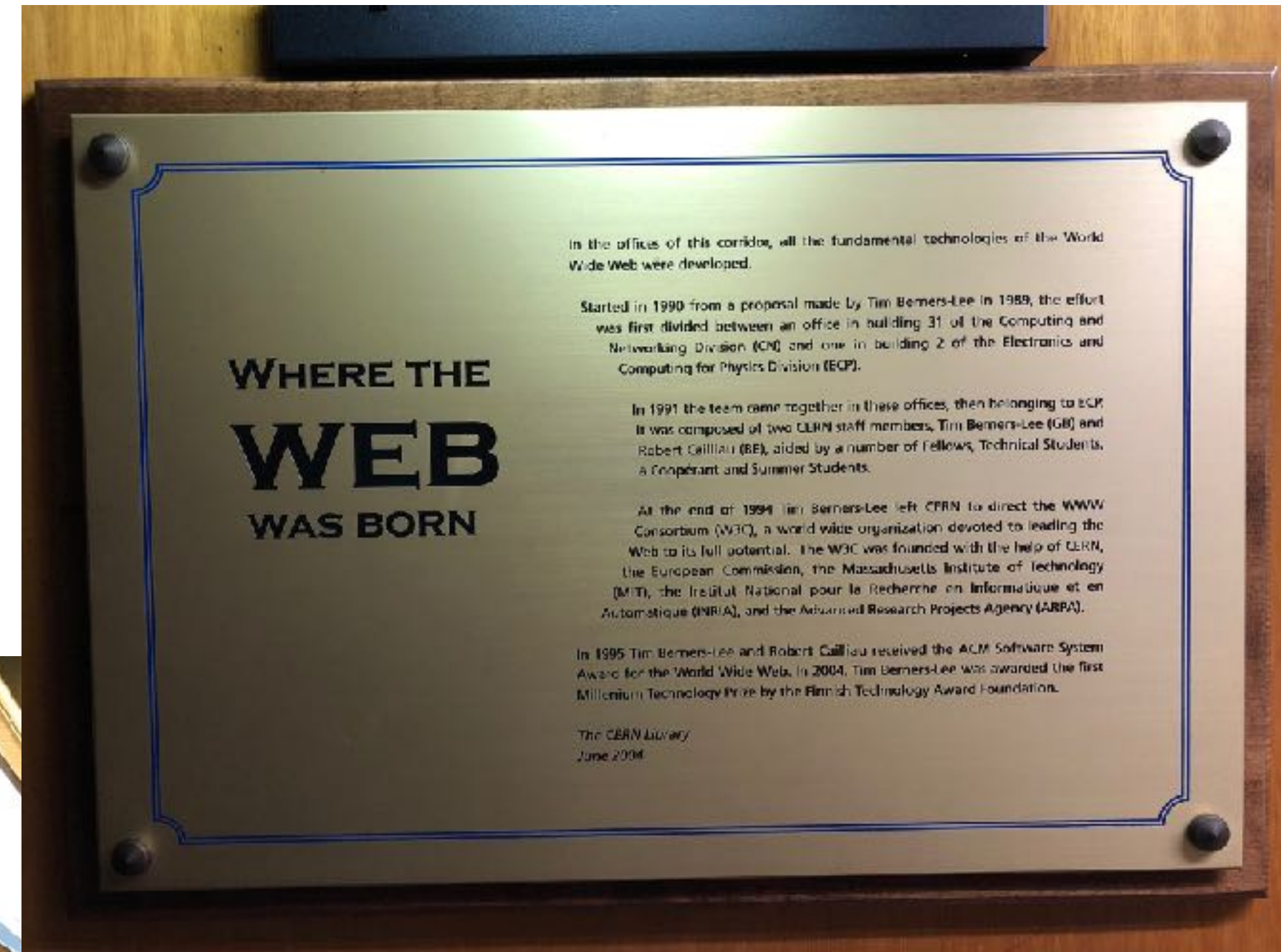


Cité des Sciences et de l'Industrie, Paris

HTTP/HTML History

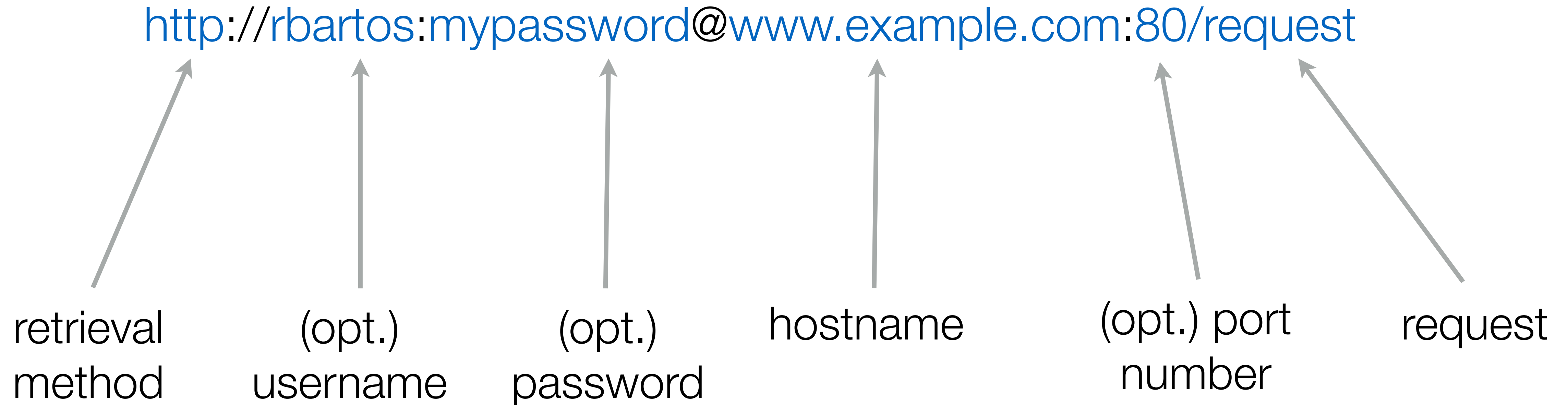


CERN, Geneva



HTTP

- ▶ HyperText Transfer Protocol (HTTP)
- ▶ URL (Universal Resource Locator)



HTTP

- ▶ HyperText Transfer Protocol (HTTP)
- ▶ “work in progress”: HTTP/1.1 → HTTP/2 → HTTP/3
- ▶ Runs on top of (reliable, transparent, connection oriented) TCP (or UDP + QUIC)
- ▶ A stateless...
- ▶ ... request/response protocol.
- ▶ protocol and payload not secured by default in HTTP/1.1
- ▶ We will focus on HTTP/1.1 initially (it is the simplest one of them and the main concepts were carried over to the newer versions)

Main HTTP Methods

▶ GET

- client requests data from the server
- server sends data

▶ POST

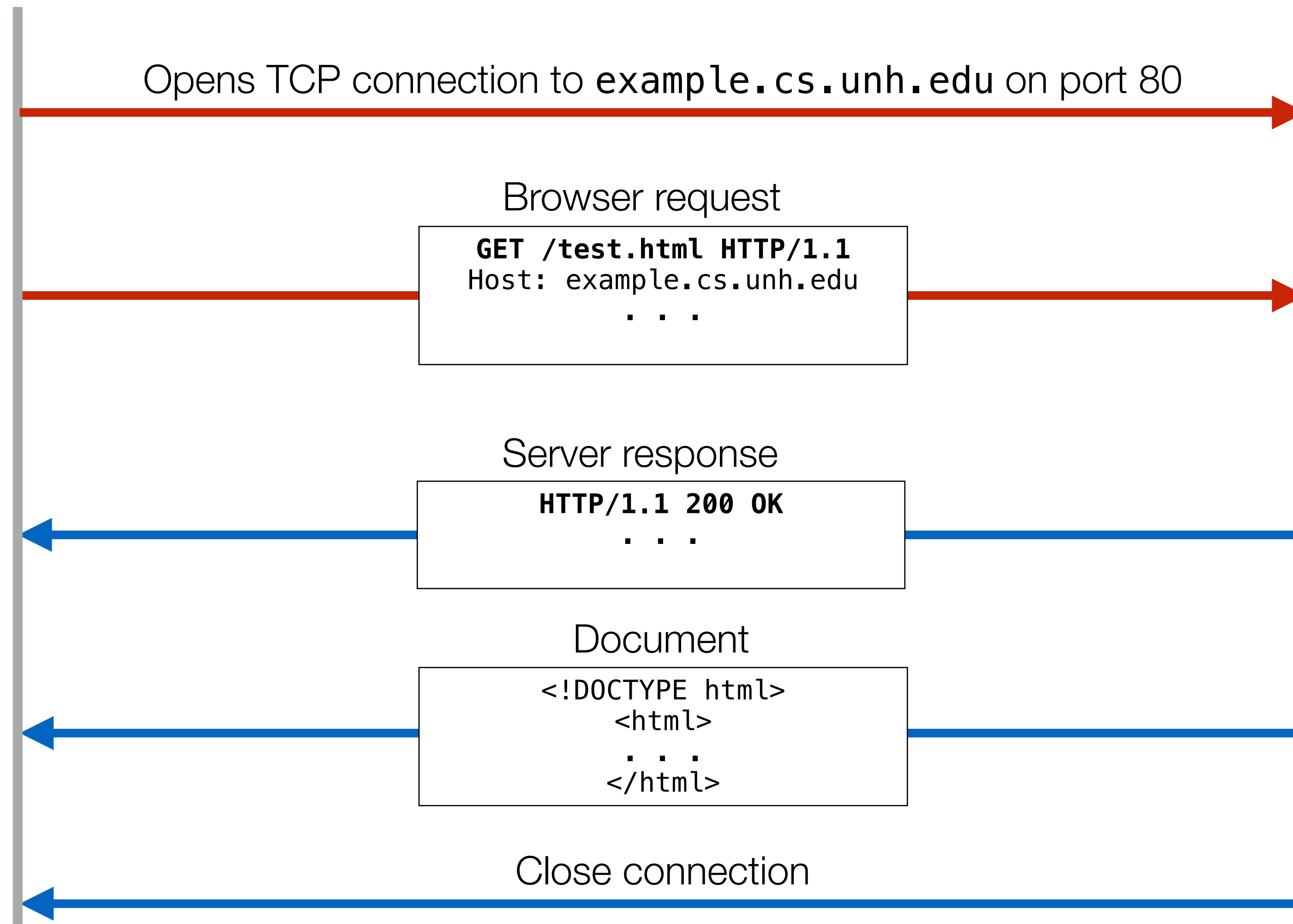
- client informs server about data to be delivered
- client sends data
- server sends data (response) back

▶ also: HEAD, PUT, DELETE, TRACE, CONNECT, ...

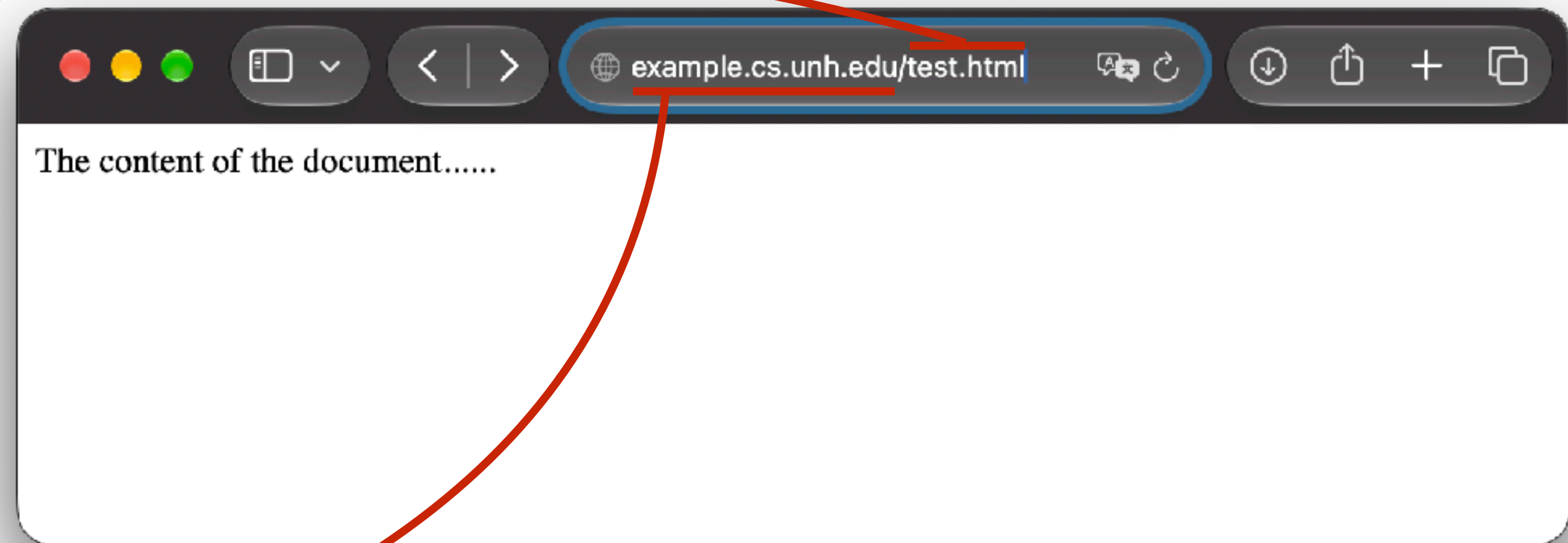
HTTP GET Request

Client - a browser connecting to
`http://example.cs.unh.edu/test.html`

Web
server



GET Request Details



GET /test.html HTTP/1.1

Host: example.cs.unh.edu

Sec-Fetch-Dest: document

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/27.0.1 Safari/605.1.15

Upgrade-Insecure-Requests: 1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Sec-Fetch-Site: none

Sec-Fetch-Mode: navigate

Accept-Language: en-US,en;q=0.9

Priority: u=0, i

Accept-Encoding: gzip, deflate

Connection: keep-alive

Browser request

Terminated by a blank line

GET Response Details

HTTP/1.1 200 OK

Date: Mon, 12 Oct 2026 21:16:54 GMT

Server: Apache/2.4.67 (Unix)

Last-Modified: Mon, 12 Oct 2026 21:12:45 GMT

ETag: "8d-65dab28c42d40"

Accept-Ranges: bytes

Content-Length: 141

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html

Server response

<!DOCTYPE html>

<html>

<head>

 <title>Title of the document</title>

</head>

<body>

 The content of the document.....

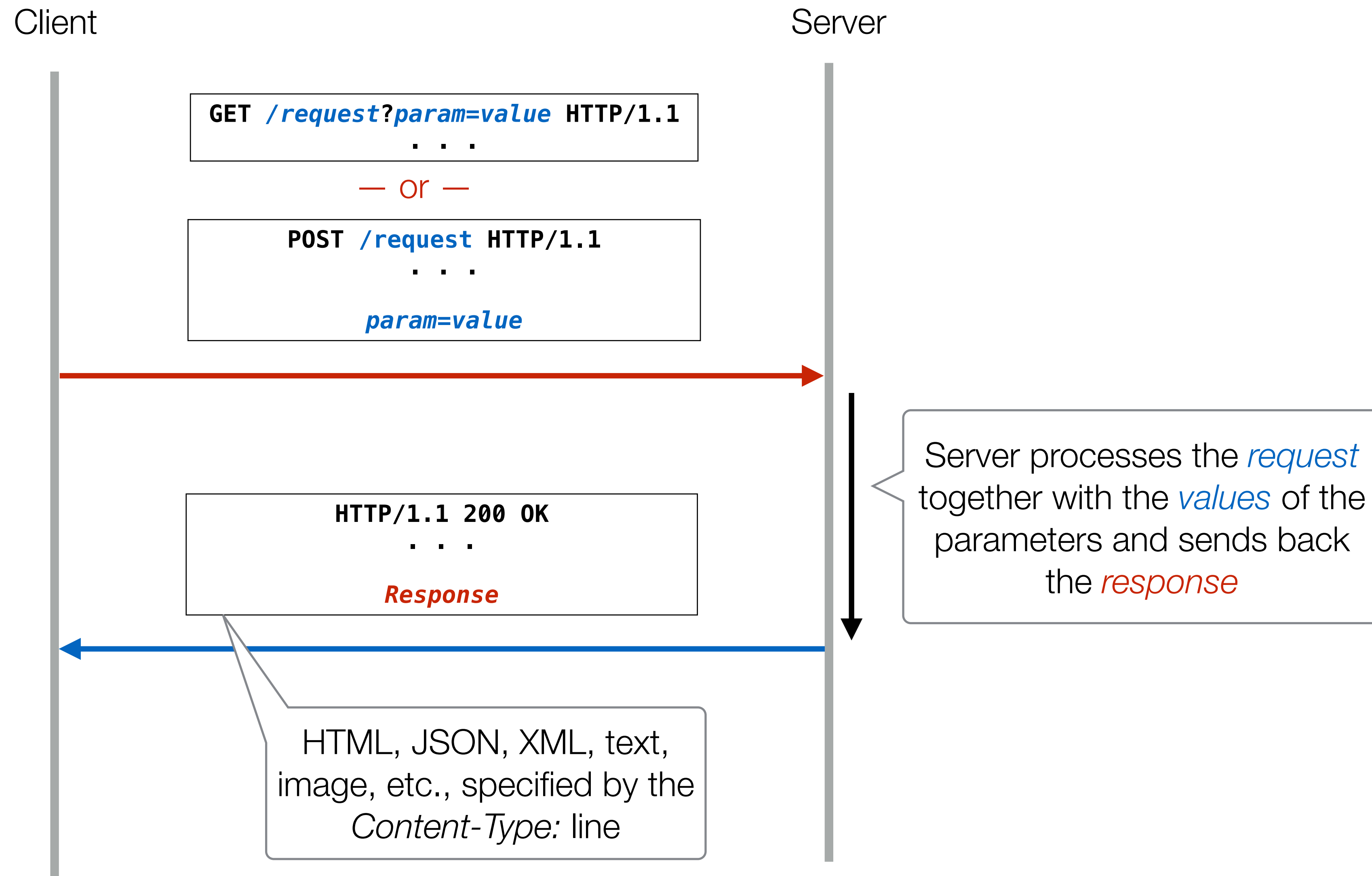
</body>

</html>

← blank line

Document

HTTP transaction in general



Server Architecture

- ▶ Infinite loop

- `while True do`

- Accept Connection

- Process the Connection

- `done`

- ▶ Concurrent request processing

- `while True do`

- Accept Connection,

- have someone else to* Process the Connection

- `done`

Concurrent Requests

- ▶ Full-blown **processes**
 - costly in terms of resources
 - independent
- ▶ **Threads**
 - more lightweight
 - shared address space between requests
- ▶ **Event-driven** approach
 - do a quick bit of processing and schedule a callback when (typically) I/O is done
- ▶ **Virtual threads**

Issues...

- ▶ Typical HTTP transaction is too short for TCP
 - (as you will see later in the course) TCP connection reaches full throughput after several RTTs
 - many HTTP request/responses are just a few packets
 - solution: Persistent Connections
- ▶ Shared fate: Head of Line (HoL) blocking
 - (as you will see later in the course) packet loss leads to reduced throughput
 - solution: Parallel Connections
- ▶ These two approaches are in conflict with each other...

HTTP/2 design goals

- ▶ Improve utilization
- ▶ Reduce latency
- ▶ Improve security
- ▶ Enable fine-grained control over resource delivery

HTTP/2 approach (1)

▶ Multiplexed connections

- limits Head Of Line (HOL) blocking and eliminates the need for concurrent TCP connections

▶ Resource push

- reduces latency of waiting for page rendering and subsequent resource requests
- (met with resistance from service providers)

HTTP/2 approach (2)

- ▶ Support for **low-latency secure connection establishment**
 - utilizes low-latency methods to open secure connections
 - while secure connection is not mandated, many current implementations do not support insecure communication
- ▶ Explicit **bandwidth allocation** for streams within a connection
 - information received concurrently on all streams with bandwidth shared according to the set ratios
 - (still needs some work, see *RFC 9218 Extensible Prioritization Scheme for HTTP* from June 2022)

HTTP/2 steps

- ▶ Secure connection is established
- ▶ Individual streams are set up
- ▶ Requests dispatched
- ▶ Information received concurrently on all streams with bandwidth shared according to the set ratios