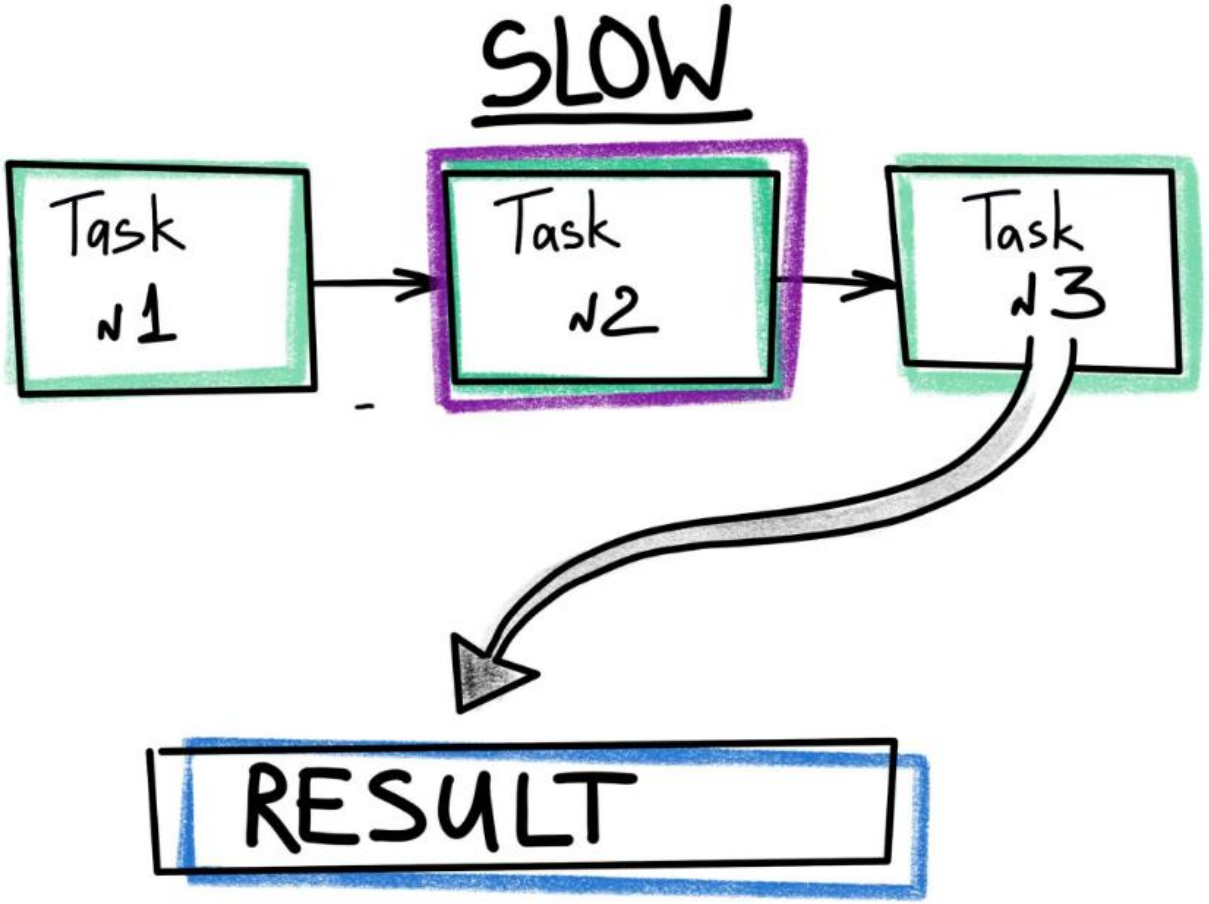
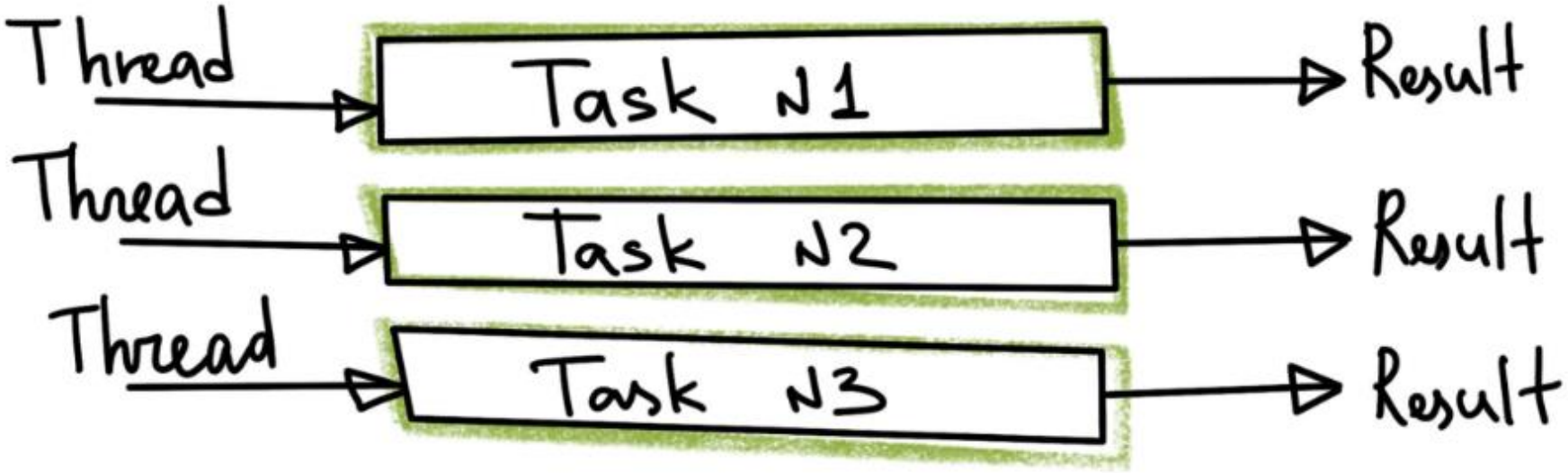
Lecture 16

The world is really Asynchronous!

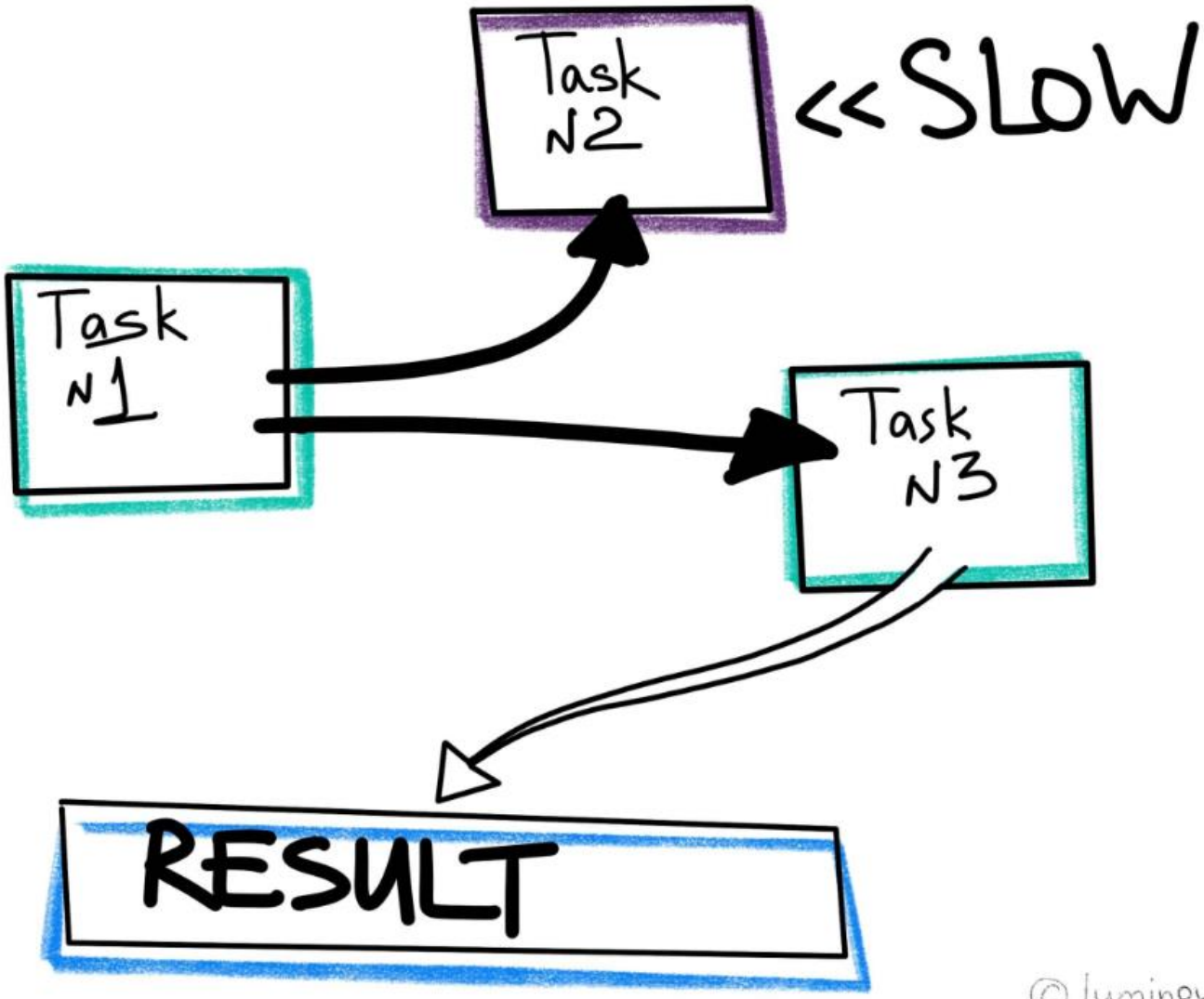
Single-threaded



Multi-threaded



Asynchrony



Asynchrony and context switching

- Asynchronous programming was actually designed for an entirely different problem: CPU context switching.
- When you have multiple threads running, each CPU core can still only run one thread at a time.
- In order to allow all threads/processes to share resources, the CPU switches context very often.
- To oversimplify things, the CPU, at a random interval, saves all the context info of a thread and switches to another thread.
- The CPU is constantly switching between your threads in non-deterministic intervals.

Asynchrony and context switching

- Asynchronous programming is essentially cooperative multitasking with user-space threading.
- The application manages the threads and context switching rather than the CPU.

In an asynchronous world, context is switched at defined switch points.

Synchronous vs Asynchronous

- The asynchronous model performs best when:

1. There are a large number of tasks.
2. The tasks perform lots of I/O, causing the synchronous program to waste lots of time.
3. The tasks are largely independent of one another so there is little need for inter-task communication.

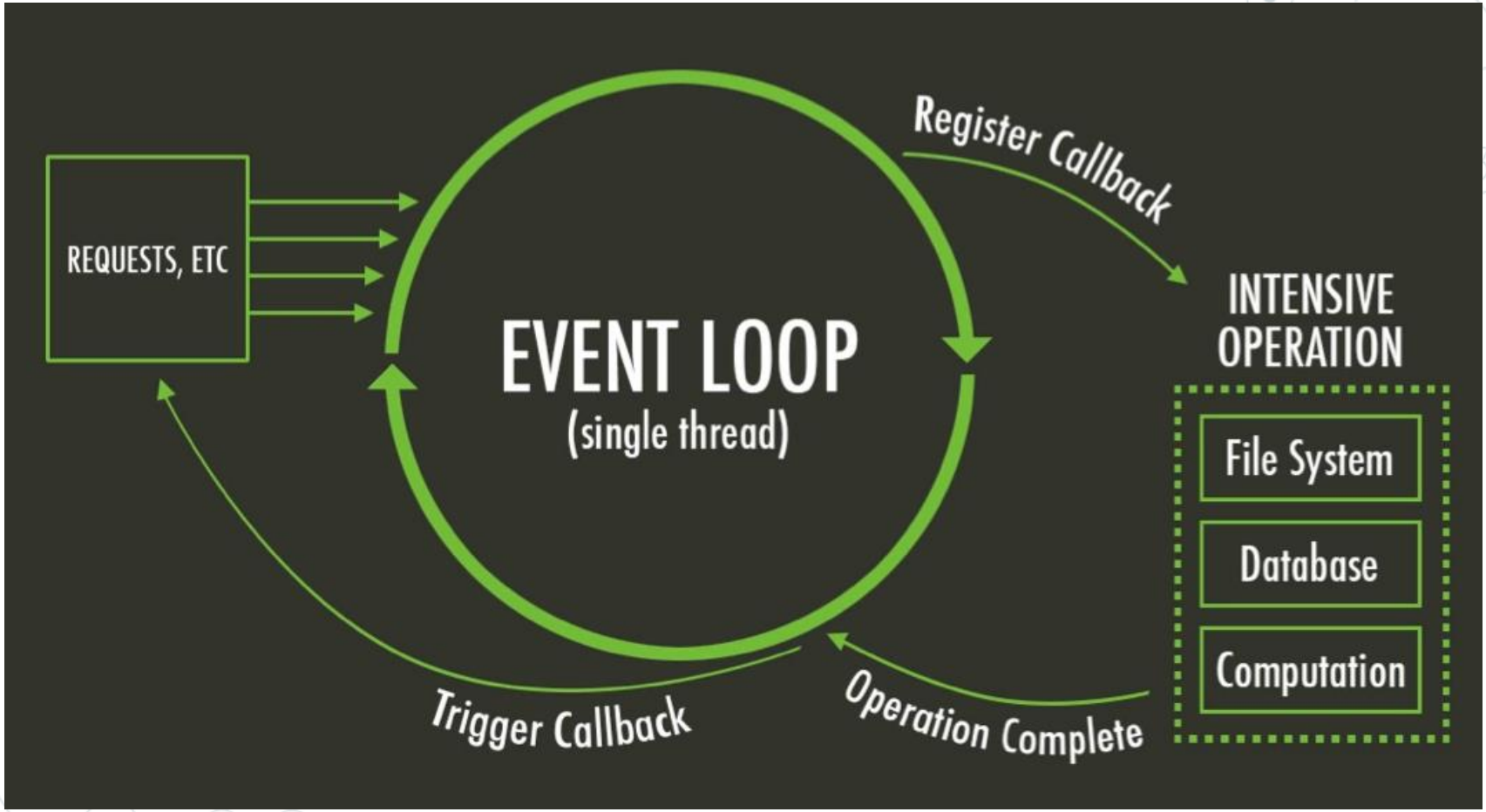
Synchronous vs Asynchronous

- These conditions almost perfectly characterize a typical busy server (like a webserver) in a client-server environment.
- Each task represents one client request with I/O in the form of receiving the request and sending the reply.

Synchronous vs Asynchronous

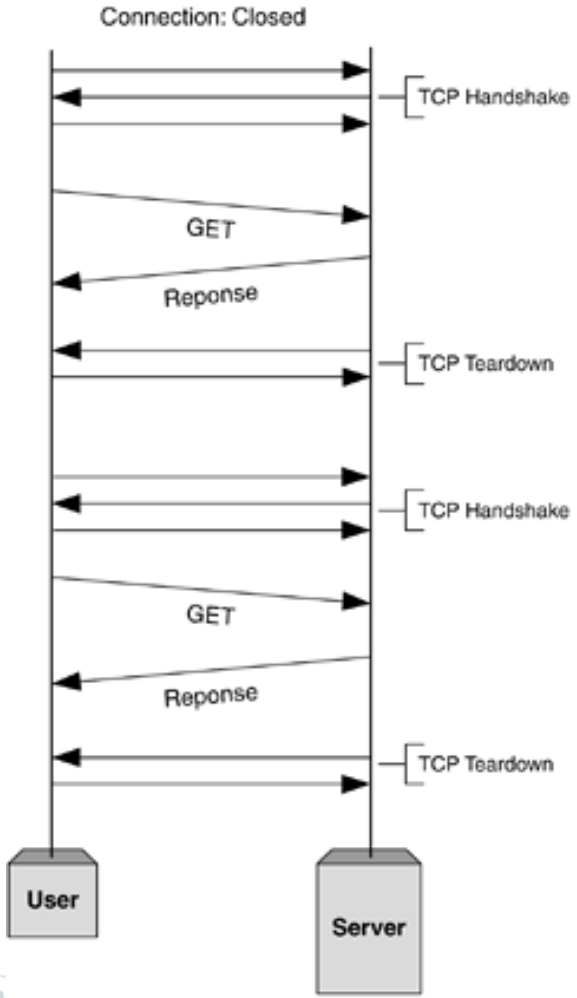
- However, as the number of threads increases, your server may start to experience performance problems.
- With each new thread, there is some memory overhead associated with the creation and maintenance of the thread.
- Another performance gain from the asynchronous model is that it avoids context switching.
- Every time the OS transfers control over from one thread to another it has to save all the relevant registers, memory map, stack pointers, CPU context etc.
- The overhead of doing this can be quite significant.

Event Loop (Remember GUI)

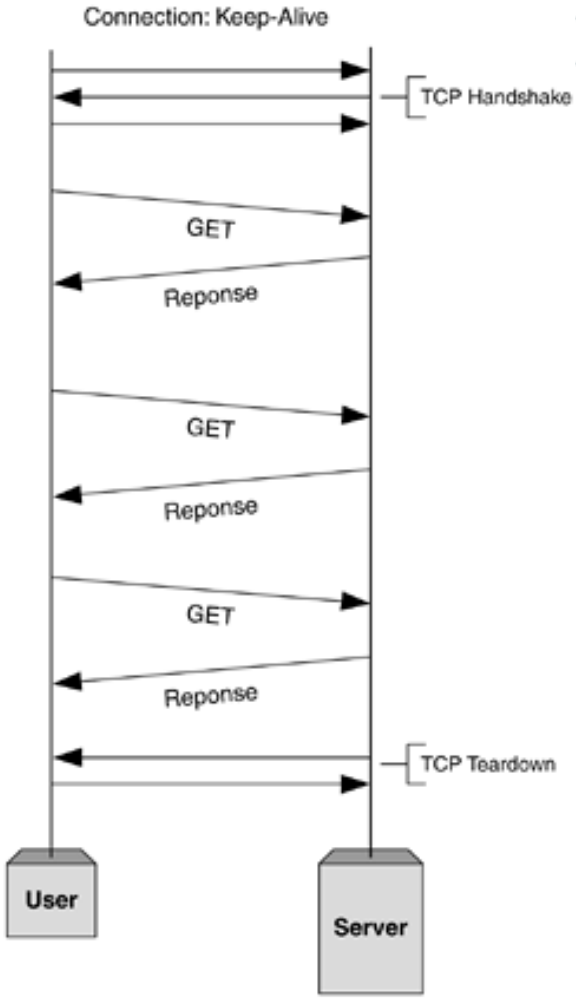


Networking, HTTP

HTTP 1 and 1.1

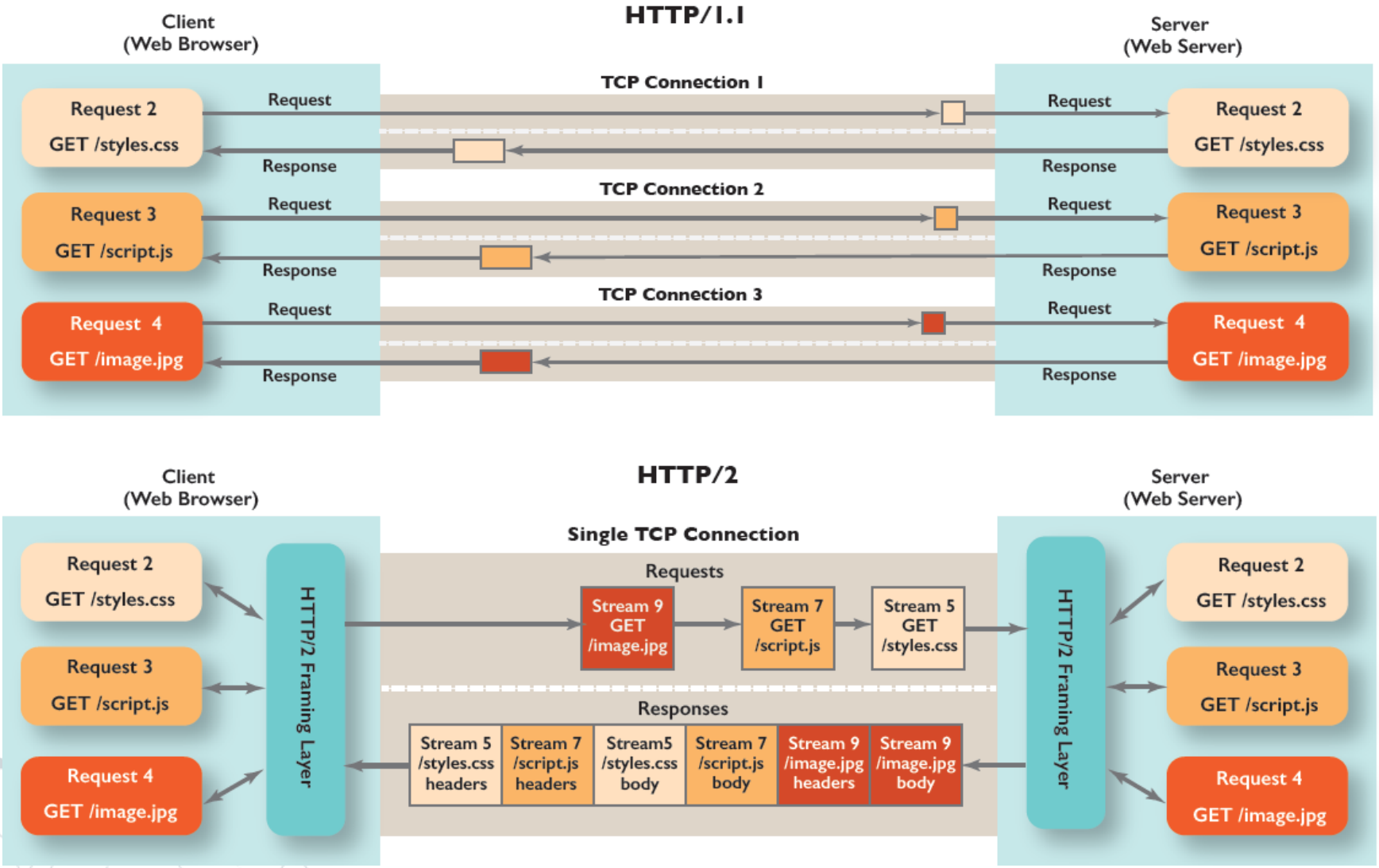


In a default HTTP/1.0 session, the TCP connection will be torn down and re-established between each HTTP GET request.



In a default HTTP/1.1 session, a single TCP connection will be held, open and multiple GET requests will be passed across.

HTTP 1.1 and 2.0



TLS 1.2 and 1.3


TLS 1.2 (Full Handshake)
Client
Server
0ms
1
Client Hello
50ms
2
Server Hello, Certificate, Server Hello Done
100ms
3
Client Key Exchange, Change Cipher Spec, Finished
150ms
4
Change Cipher Spec, Finished
200ms
5
GET HTTP/1.1
250ms
6
HTTP Response
300ms
7
CLOUDFLARE
TLS 1.3 (Full Handshake)
Client
Server
0ms
1
Client Hello, Key Share
50ms
2
Server Hello, Key Share, Certificate, Certificate Verify, Finished
100ms
3
Finished
GET HTTP/1.1
150ms
4
HTTP Response
200ms
5
250ms
300ms

HTTPS 2.0 and 3.0