Tutorial 18: Going for Speed

09:00	Introduction
09:15	Part 1: Efficient Frontend Design
09:40	Q&A
09:50	Coffee Break
10:00	**Part 2: High-Performance Networking**
10:40	Q&A
10:50	Coffee Break
11:00	Part 3: Scalable Backend Architectures
11:40	Q&A
11:50	Coffee Break
12:00	Part 4: Performance Tracking & Analysis
12:00	The Core Web Vitals (Google Guest Speaker!)
12:30	Measuring Web Performance
12:50	Q&A

Baqend

The 4 Challenges of Web Performance

Up next!

2 Network Delays

1 Client

3 Backend Processing

4 Tracking & Analysis

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Benjamin Wollmer

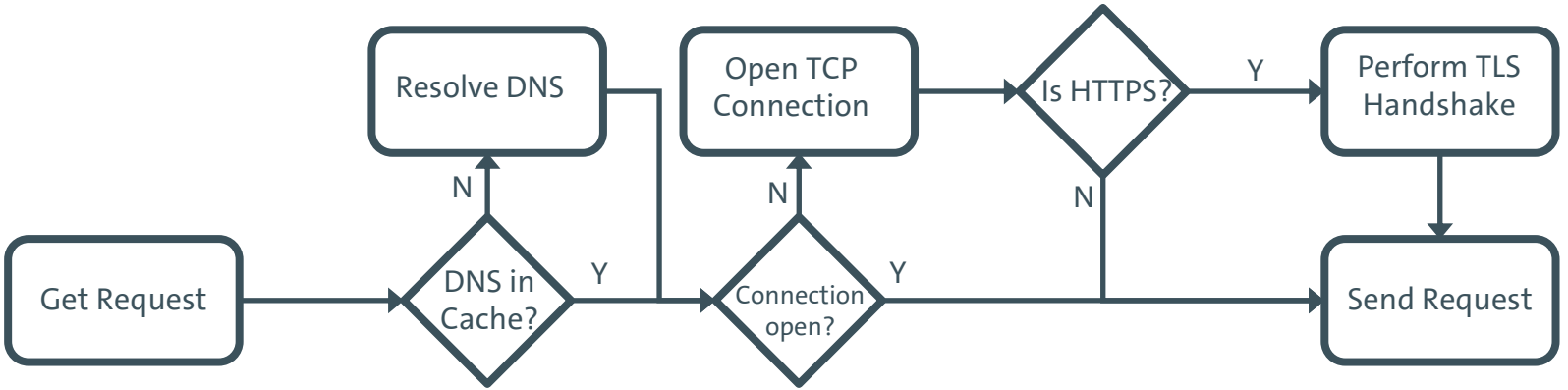
Going for Speed: Network

Invoking a Request Is Easy....

```
fetch('https://example.com/profil')
.then(response => response.json())
.catch((error) => {
  console.error('Error:', error);
});
```

```
<link rel="stylesheet" href="styles.css">
```

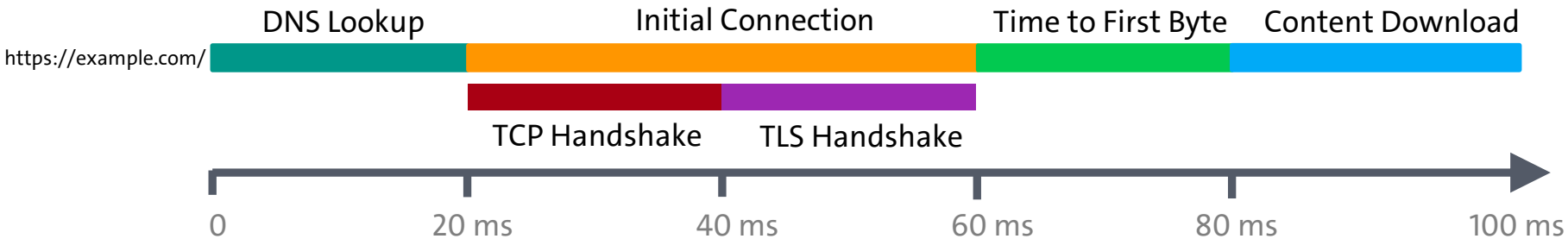
… But Handling It?



Stephen Ludin, Javier Garza
Learning HTTP/2 – A practical guide for beginners. 2017

Request Waterfall

Request Breakdown

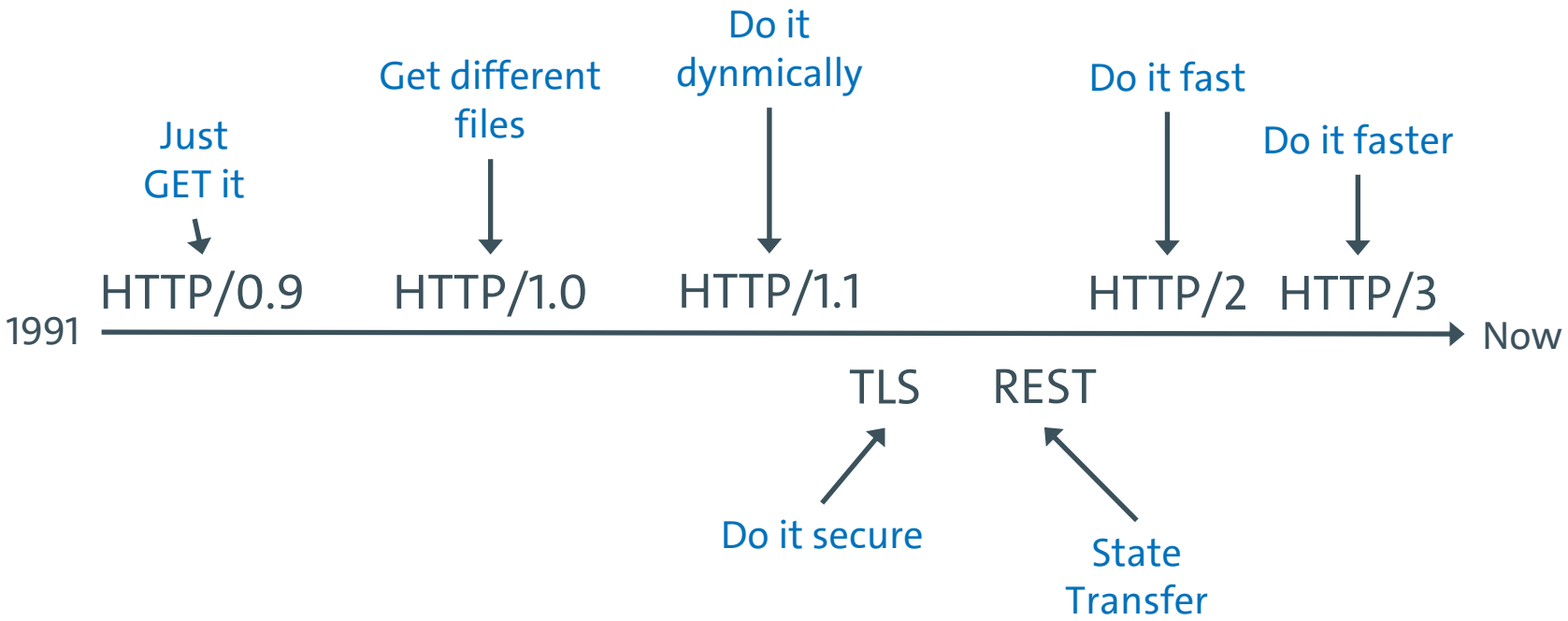


Optimization Knobs

- DNS Performance
- TCP Connection Handling
- TLS Handshakes
- HTTP/2 Usage

Hypertext Transfer Protocol

HTTP


Just
GET it
Get different
files
Do it
dynmically
Do it fast
Do it faster
HTTP/0.9
HTTP/1.0
HTTP/1.1
HTTP/2
HTTP/3
1991
Now
TLS
REST
Do it secure
State
Transfer

HTTP Requests and Responses

▼ **Request Headers**

:authority: www.baqend.com
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
cache-control: no-cache
cookie: Tawk_5939023fb3d02e11ecc68cfb=vs27.tawk.to::0; a
pragma: no-cache
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

▼ **General**

Request URL: https://www.baqend.com/
Request Method: GET
Status Code: 200
Remote Address: 151.101.13.132:443
Referrer Policy: no-referrer-when-downgrade

▼ **Response Headers**

accept-ranges: bytes
age: 0
cache-control: public, max-age=30
content-encoding: gzip
content-length: 14810
content-type: text/html
date: Sat, 27 Jan 2018 17:56:45 GMT
etag: "a7b9eaa89f9fb4c6fb8d6b706b6dc62c"
last-modified: Tue, 23 Jan 2018 14:32:29 GMT
server: AmazonS3
status: 200
vary: Accept-Encoding
via: 1.1 varnish

HTTP Requests and Responses

▼ **Request Headers**

:authority: www.baqend.com
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
cache-control: no-cache
cookie: Tawk_5939023fb3d02e11ecc68cfb=vs27.tawk.to::0; a
pragma: no-cache
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

- Standarized HTTP request headers:
  - authority
  - path
  - cookies
  - user-agent

HTTP Requests and Responses

▼ **Request Headers**

:authority: www.baqend.com
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
cache-control: no-cache
cookie: Tawk_5939023fb3d02e11ecc68cfb=vs27.tawk.to::0; a
pragma: no-cache
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

- Negotiation
- Preferred Encodings/ Compression algorithms
- User's languages

HTTP Requests and Responses

▼ **Request Headers**

:authority: www.baqend.com
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
cache-control: no-cache
cookie: Tawk_5939023fb3d02e11ecc68cfb=vs27.tawk.to::0; a
pragma: no-cache
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

- Indicates the type of operation:
  - GET/HEAD: read
  - PUT: (over-)write
  - POST: update
  - DELETE: delete
  - Others: PATCH, OPTIONS
- Different semantics:
  - Safe, Idempotent, Cacheable

HTTP Requests and Responses

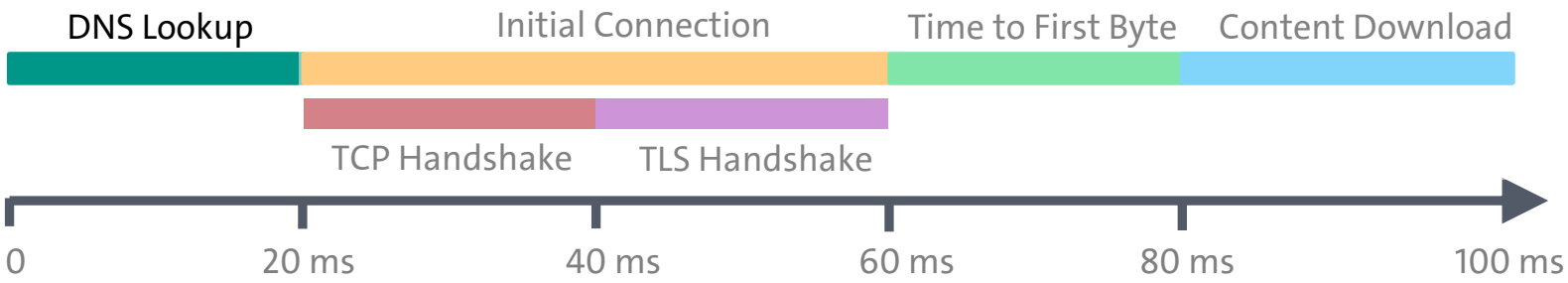
HTTP Method	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	Optional	Yes	Yes	Yes	Yes
HEAD	No	No	Yes	Yes	Yes
POST	Yes	Yes	No	No	Yes
PUT	Yes	Yes	No	Yes	No
DELETE	No	Yes	No	Yes	No
CONNECT	Yes	Yes	No	No	No
OPTIONS	Optional	Yes	Yes	Yes	No
PATCH	Yes	Yes	No	No	No

HTTP Requests and Responses

▼ **Request Headers**

:authority: www.baqend.com
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
cache-control: no-cache
cookie: Tawk_5939023fb3d02e11ecc68cfb=vs27.tawk.to::0; a
pragma: no-cache
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

- Bypass any web caches
- Other options, for Cache-Control e.g.:
  - max-age
  - max-stale
  - only-if-cached

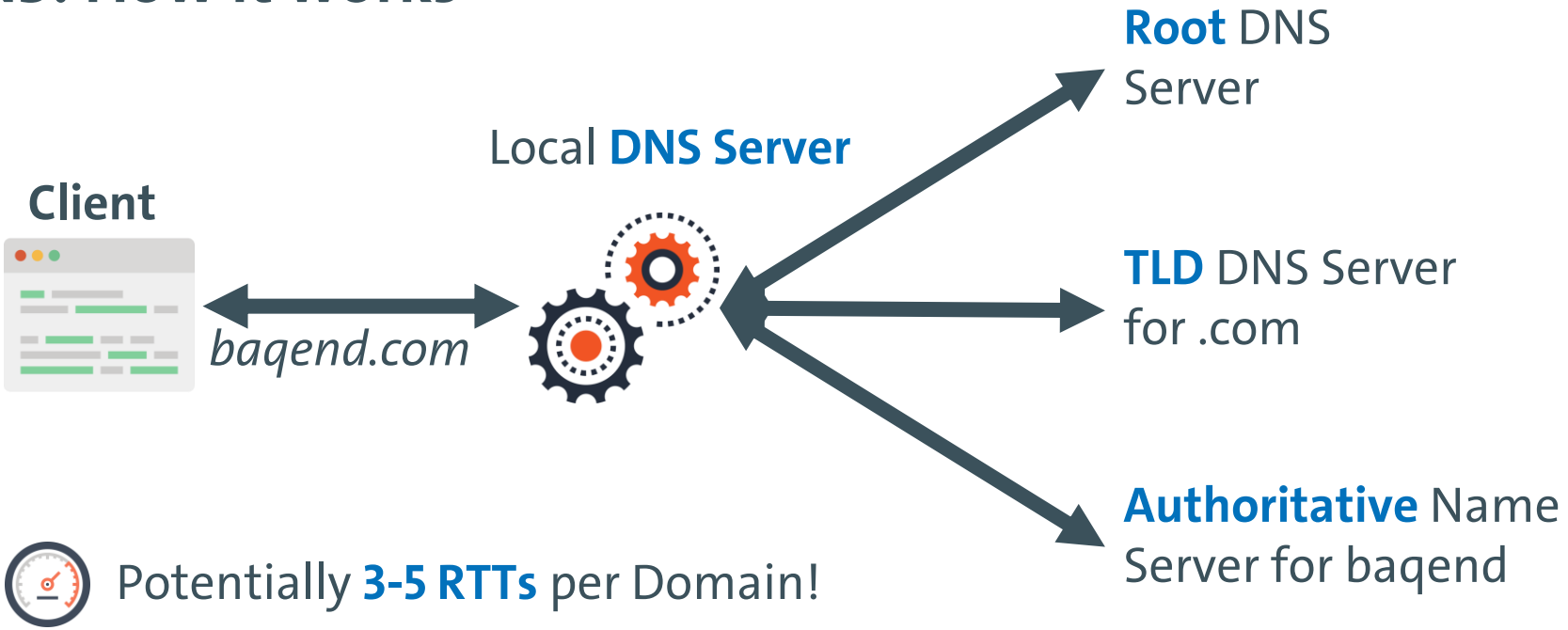


Domain Name System

Domain Name System

- Translates Domain **Names to IP** Addresses
  Based on (unreliable but fast) UDP
- Database: **<Name, Value, Type, Class, TTL>**
  TTL-based caching ensures performance
- Highly **Performance-Relevant**
  DNS lookup in the critical request path

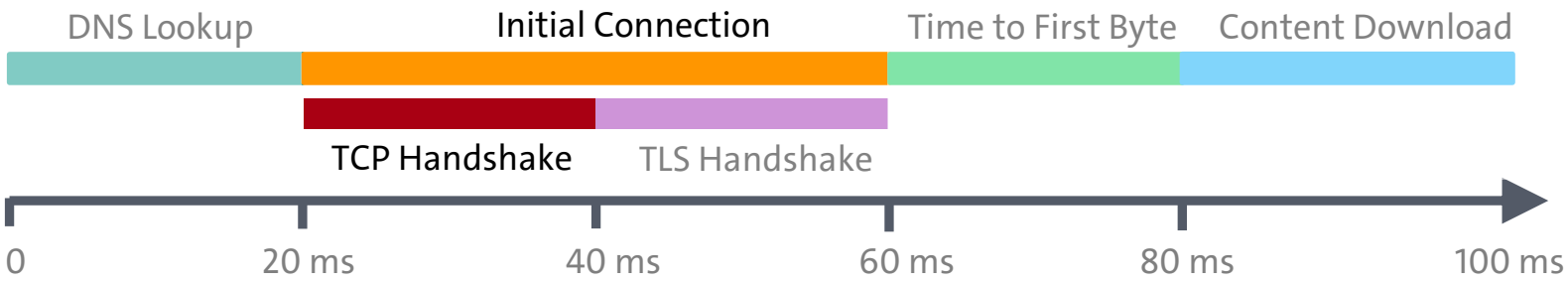
DNS: How it works



DNS: Performance Measures

- DNS Replicas + IP Anycasting
- Setting TTLs for Caching (e.g., 1 hour)
  Problem: Can lead to unavailability
- Combine multiple DNS providers
  - ISP favors the fastest server per region
  - Availability increased (cf. Dyn attack)

Stackoverflow Blog: https://blog.serverfault.com/2017/01/09/surviving-the-next-dns-attack/



Transmission Control Protocol

Transmission Control Protocol

Ensure that sent data arrives **in order**

Tolerance against **packet loss** and network **congestion**

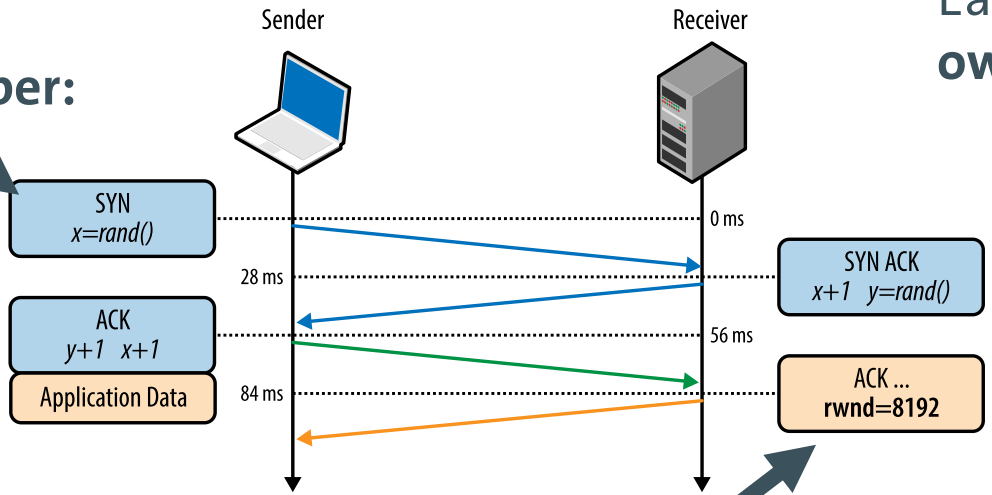
Provide data **integrity**

Fall, Kevin R., and W. Richard Stevens. TCP/IP illustrated, volume 1: The protocols. addison-Wesley, 2011.

I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

Three-Way Handshake

Sequence Number:
order of data

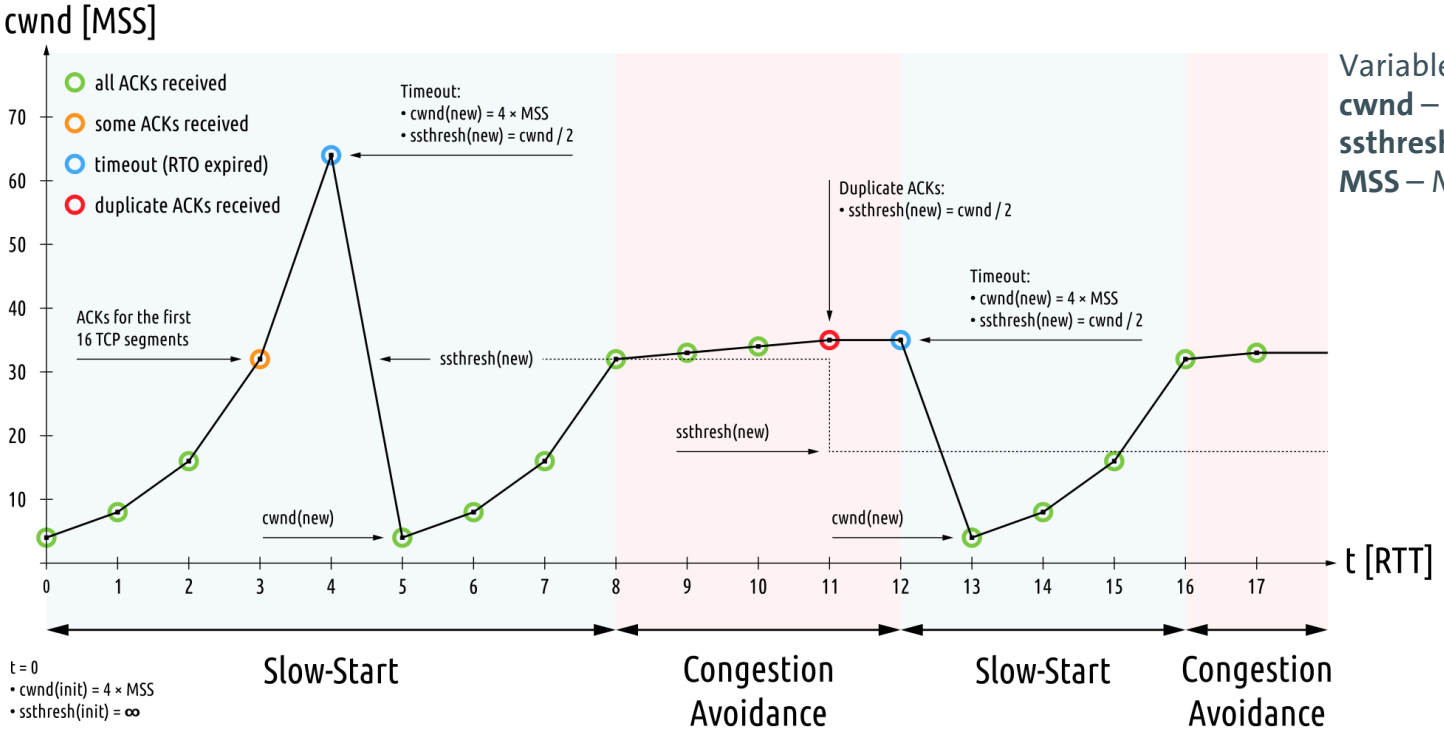


Each side has its **own** seq. numbers

Receiver Window: amount of data the client is willing to receive at once

I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

Congestion Avoidance



Variables & Constants
cwnd – Congestion Window Size
ssthresh – Slow Start Threshold
MSS – Maximum Segment Size

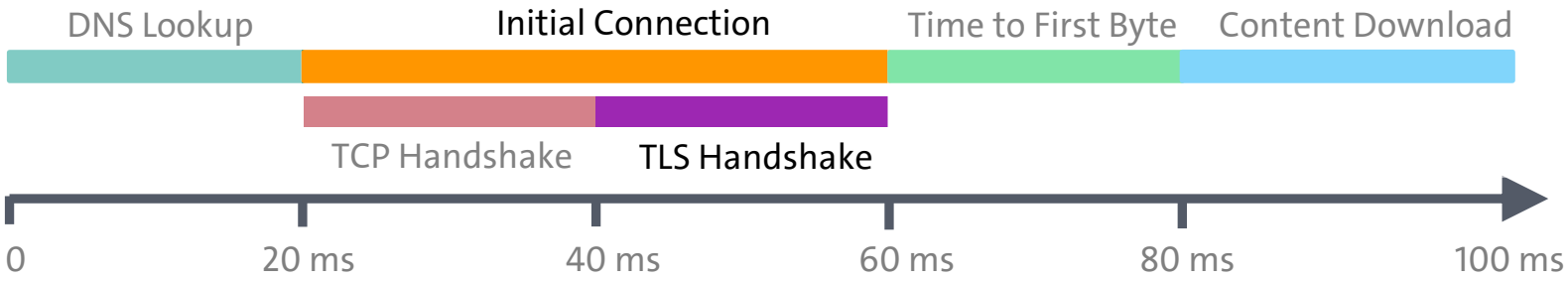
https://commons.wikimedia.org/wiki/File:TCP_Slow-Start_and_Congestion_Avoidance.svg

Basics: Transmission Control Protocol

Core Question:
How to balance between aggressive bandwidth use and packet loss?

Many Implementations:

- Original: TCP Tahoe and Reno (original implementations)
- Newer ones: TCP Vegas, TCP New Reno, TCP BIC, TCP CUBIC (Linux), or Compound TCP (Windows)



Transport Layer Security

TLS: Cryptographically Secure Channels Over TCP

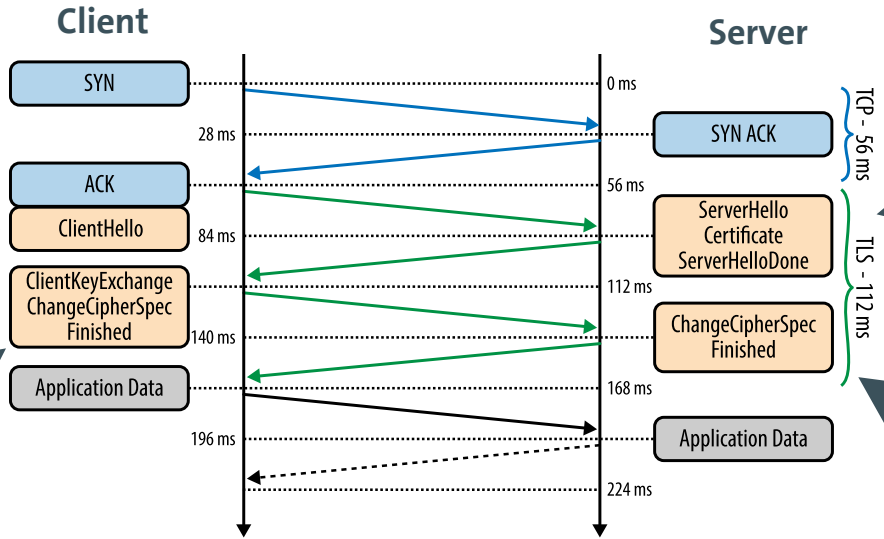
Encryption
to protect against eavesdropping

Authentication
To verify the counterpart‘s identity

Integrity
to protect against modifications

TLS Handshake

TLS Options and supported **Ciphersuites**



Pick TLS version & cipher, send **certificate**

Initiate **key exchange** for symmetric crypto

Acks **session key**

I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

TLS False Start: Sending data in the 2nd handshake step

Idea: session key known after **1st handshake**

- Client already sends application **data in 2nd round**
- Does not change TLS protocol, only **timing**

Requirements for deployment:

- Ciphersuite with **Forward Secrecy (Diffie-Hellmann)** required by browsers
- Server must support **ALPN**

Application Layer Protocol Negotiation: Upgrading Protocols

Problem: which protocol will be used over TLS?

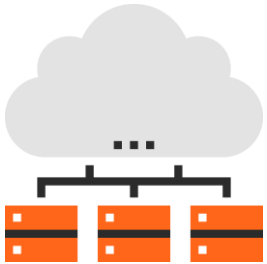
Client attaches **supported protocols** to ClientHello message

Server **selects** a protocol and sends it back in ServerHello message

Session Resumption: Skipping key negotiation the 2nd time

32-Byte **Session ID** identifies parameters

1. Client sends session ID to server
2. Server looks up ID and reuses ciphersuite and session key



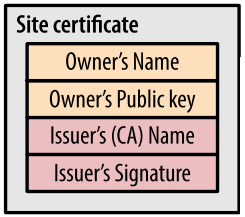
Problem: **shared state** across servers

→ Encrypt stateless **Session Ticket** (RFC 5077) based on a secret server key

Optimizing TLS Certificates: Performance Best Practices

Minimize length of the trust chain

- Missing certs will be fetched
  → DNS, TCP, HTTP overhead
- Include intermediate certs
- But not CA cert



I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

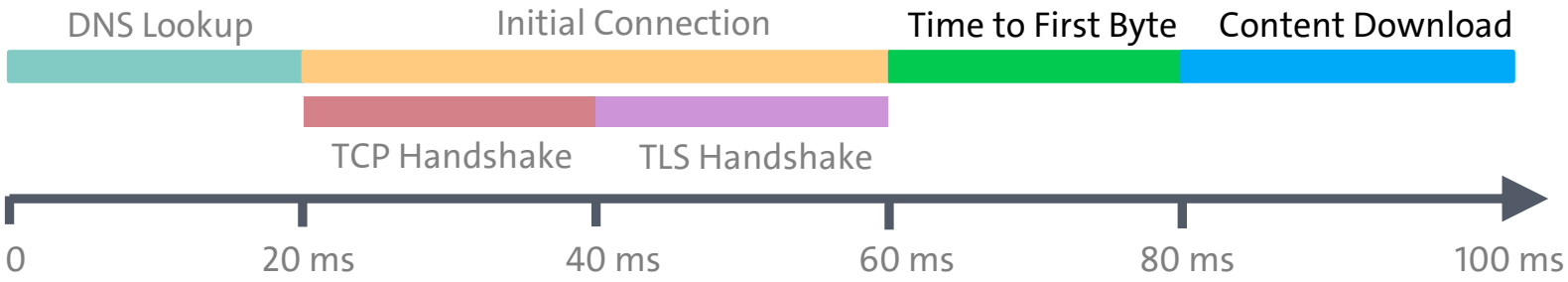
Optimizing Certificate Revocation

Problem:

- Clients have to check if certificates were revoked
- Certificate Revocation Lists (CRLs): CA maintains list of revoked serial numbers → large & slow
- Online Certificate Status Protocol (OCSP): client requests CA database for each serial number → additional RTT

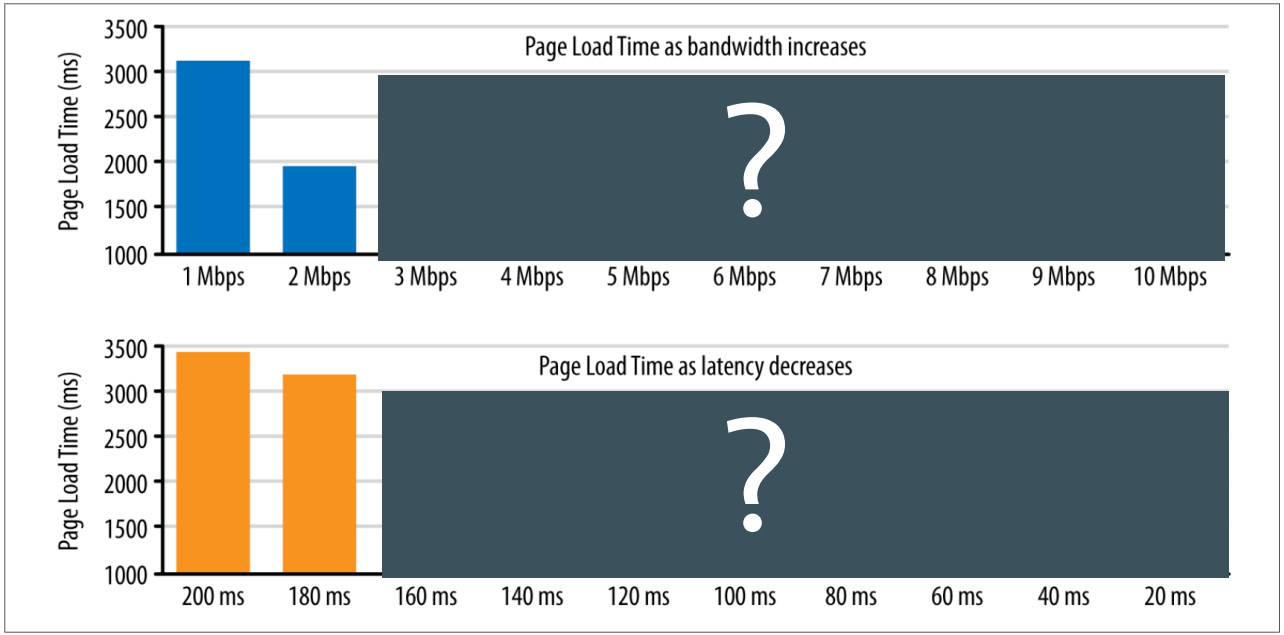
Improvement:

- OCSP Stapling
- Server staples pre-fetched, timestamped OCSP responses to certificate reply



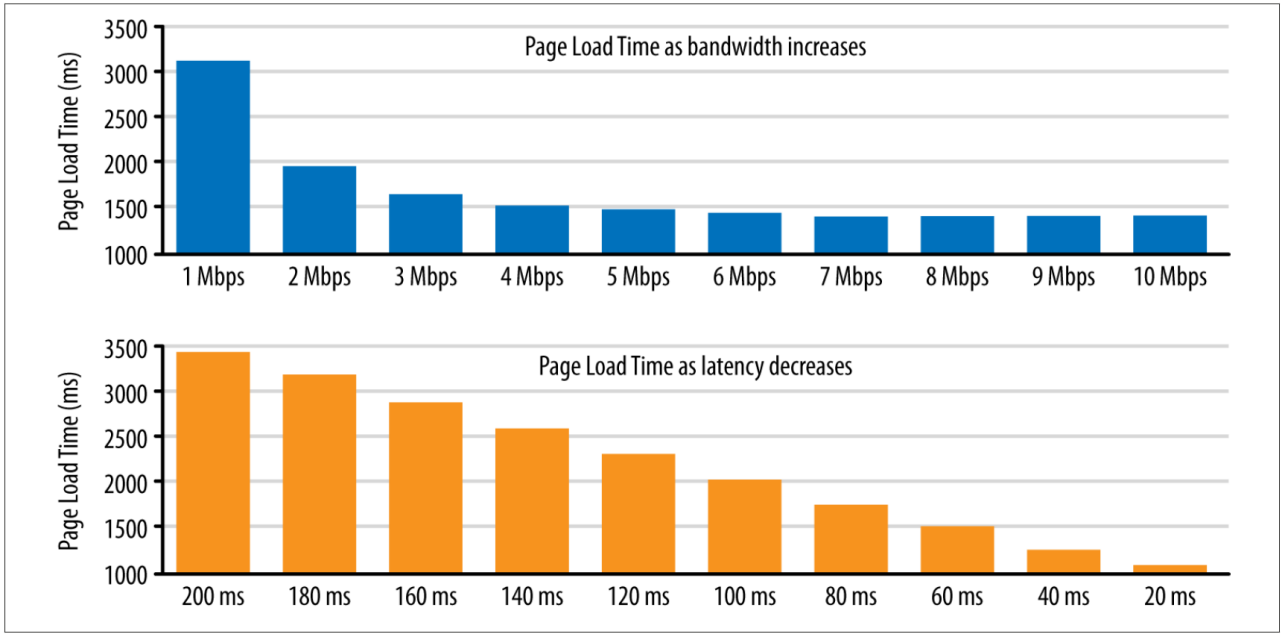
Content Delivery Network

Latency vs Bandwidth



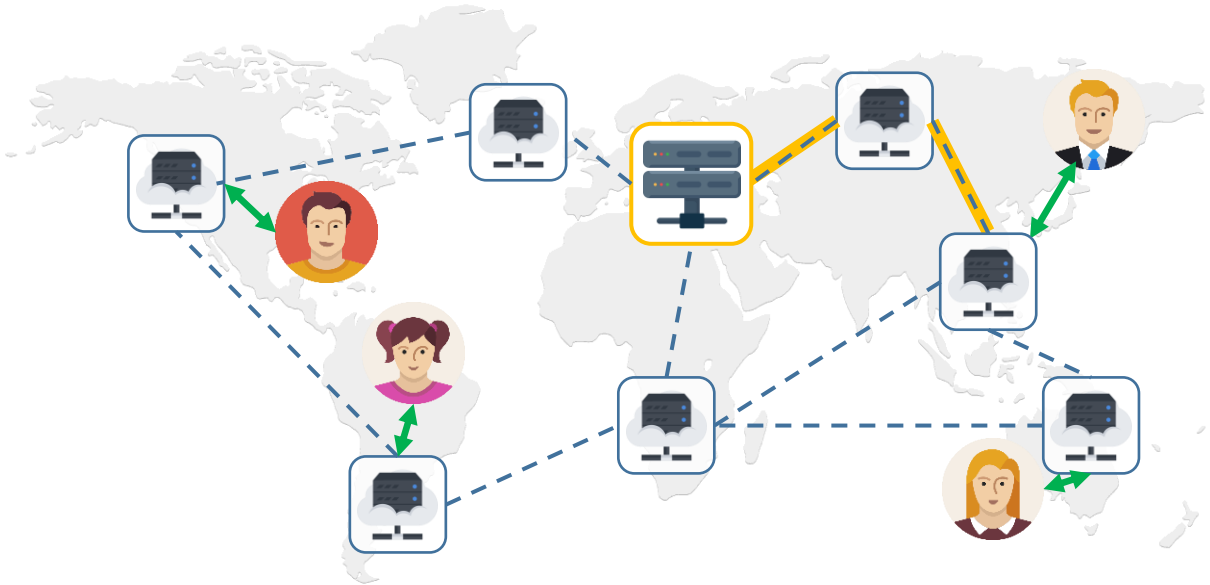
I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

Latency vs Bandwidth

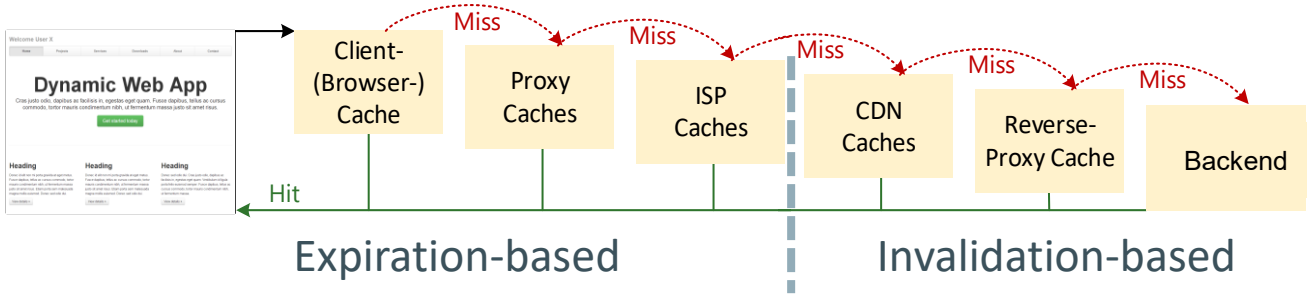


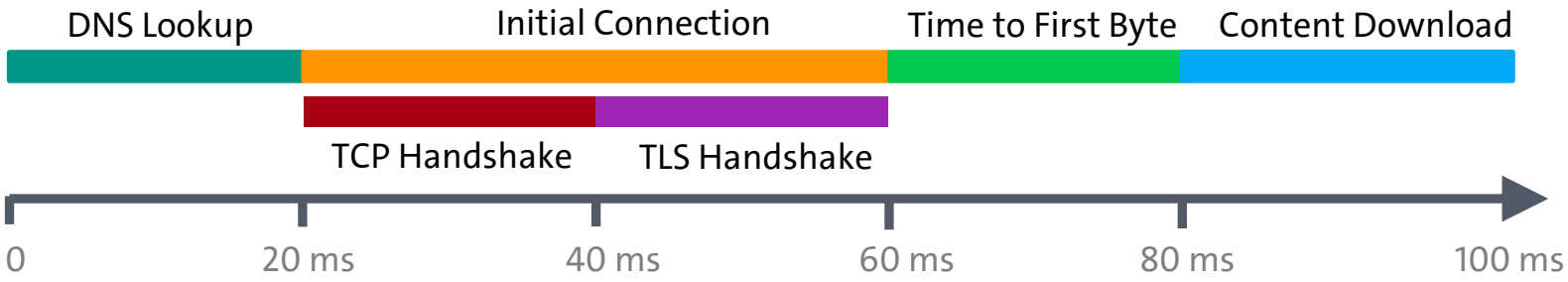
I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

Adding a Content Delivery Network (CDN)



Expiration vs Invalidation




DNS Lookup
Initial Connection
Time to First Byte
Content Download
TCP Handshake
TLS Handshake
0
20 ms
40 ms
60 ms
80 ms
100 ms

HTTP/2

HTTP/2 – What is it about

Fix **design flaws** of HTTP1.x in terms of performance

Give new knobs to tune **network performance**

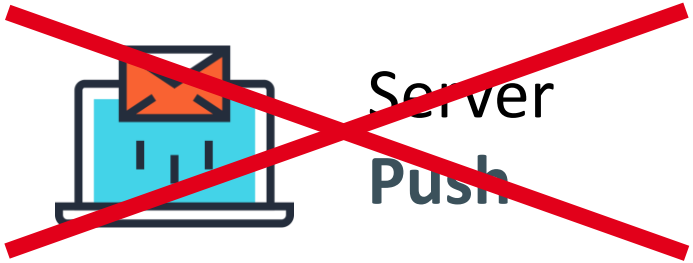
Keep HTTP **semantics** intact

HTTP/2 – Features

Multiplexing
(1 Connection)

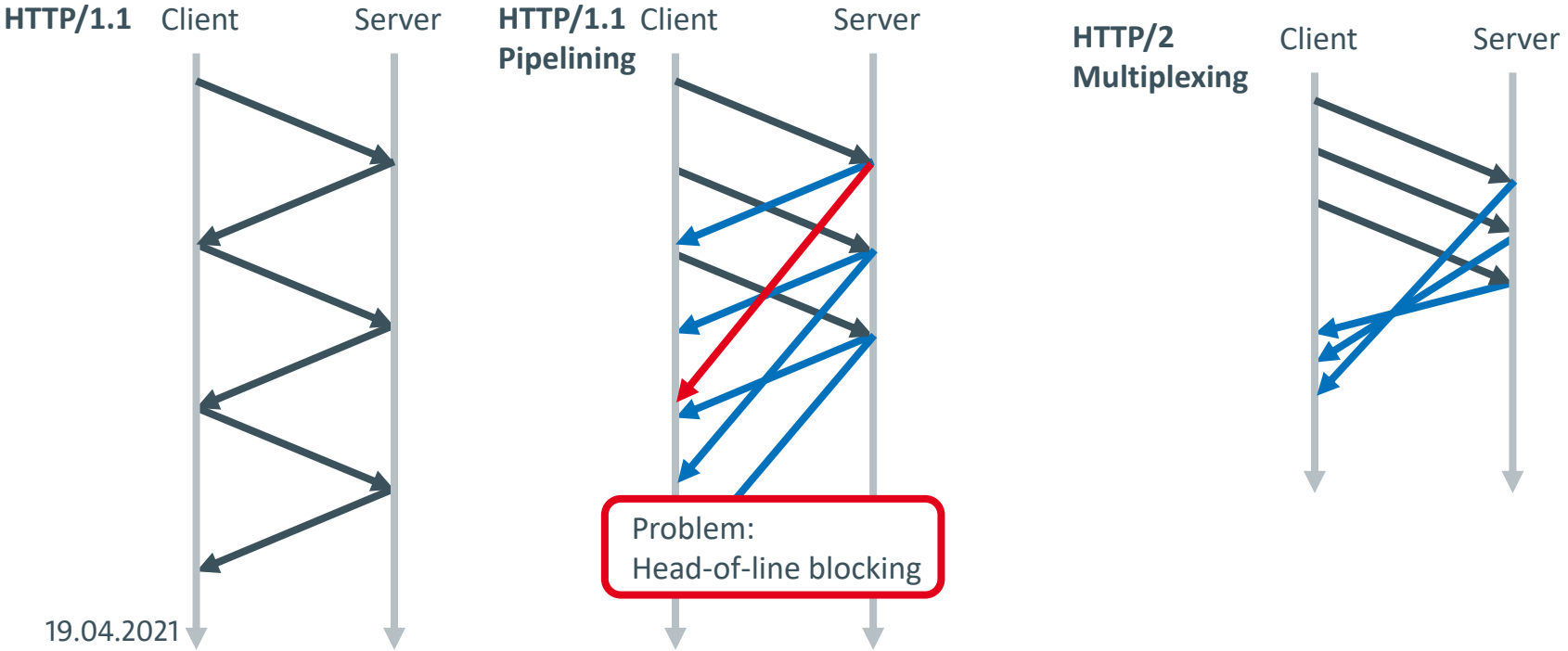
Resource
Prioritization

Header
Compression

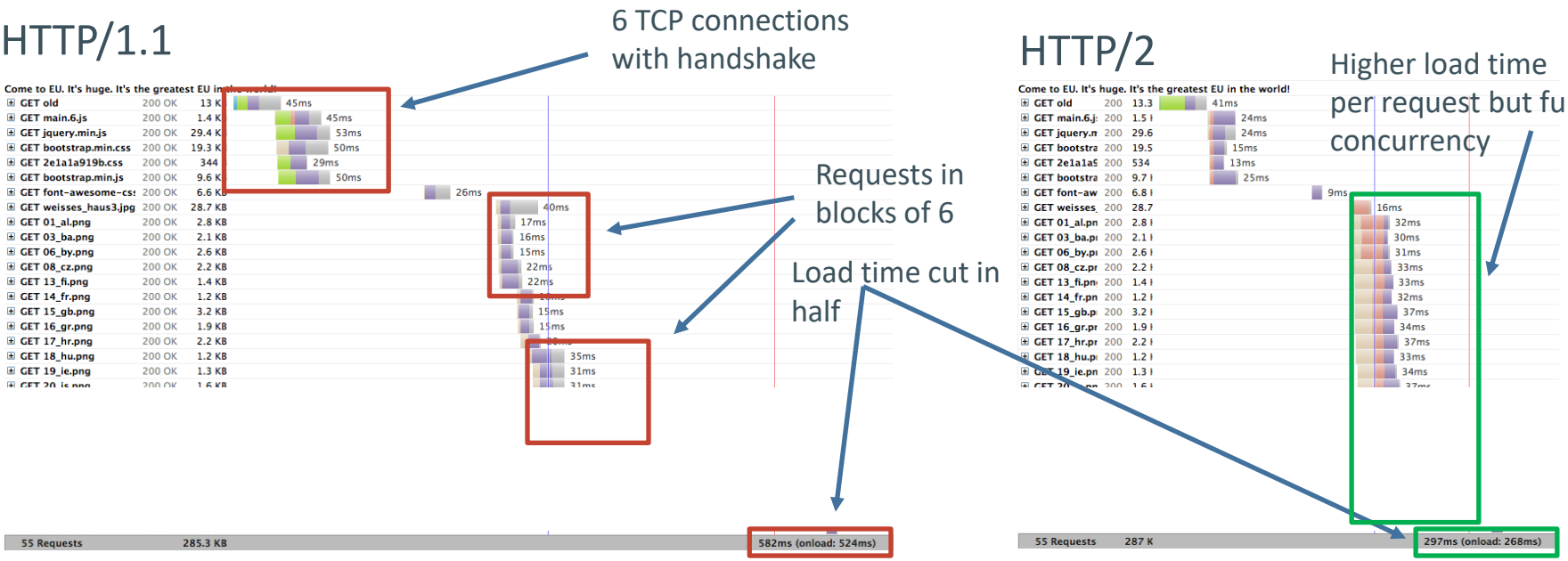


Server
Push

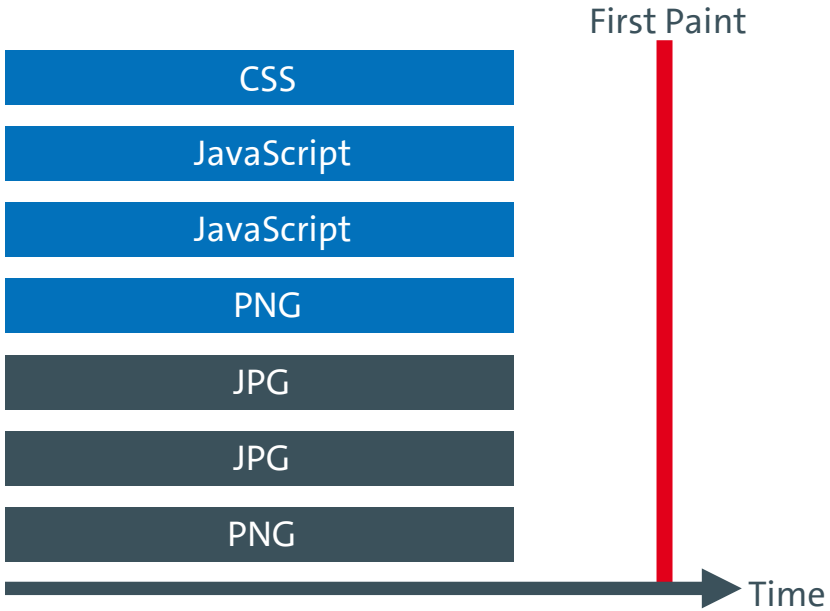
HTTP/1.1 Pipelining vs. HTTP/2 Multiplexing

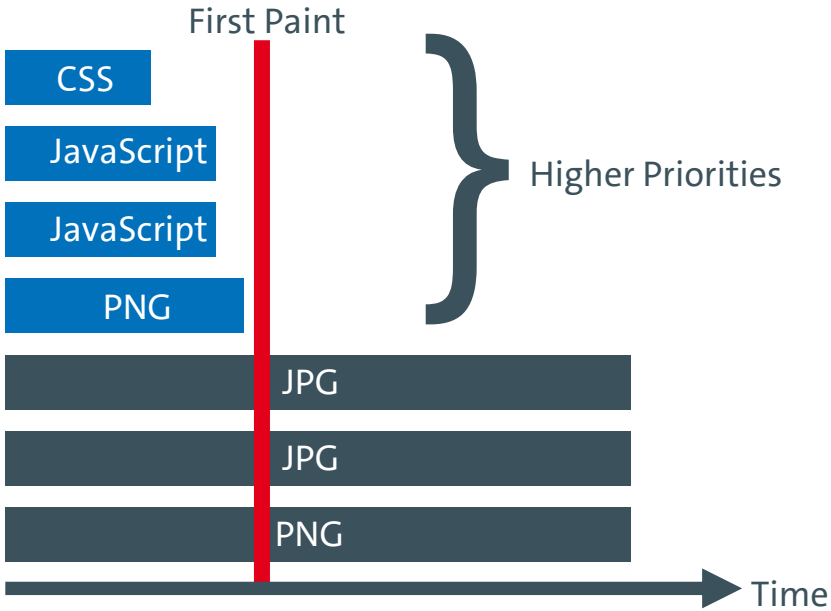


Features: Multiplexing (vs. Pipelining)



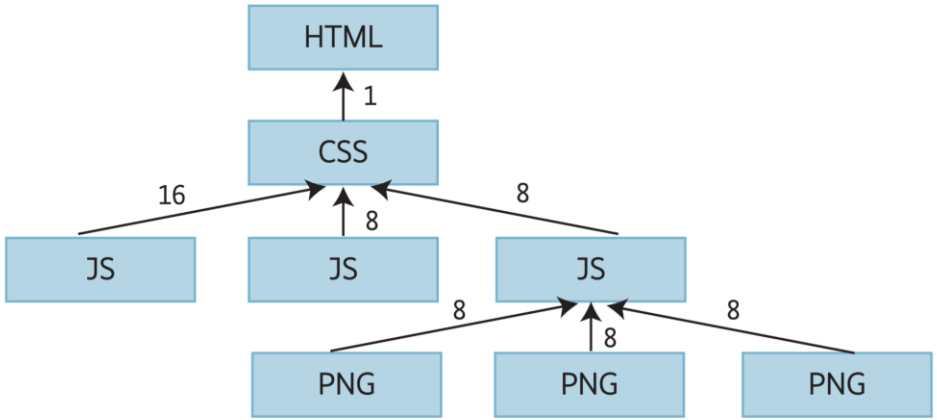
Multiplexing vs. Priorities





Resource Prioritization

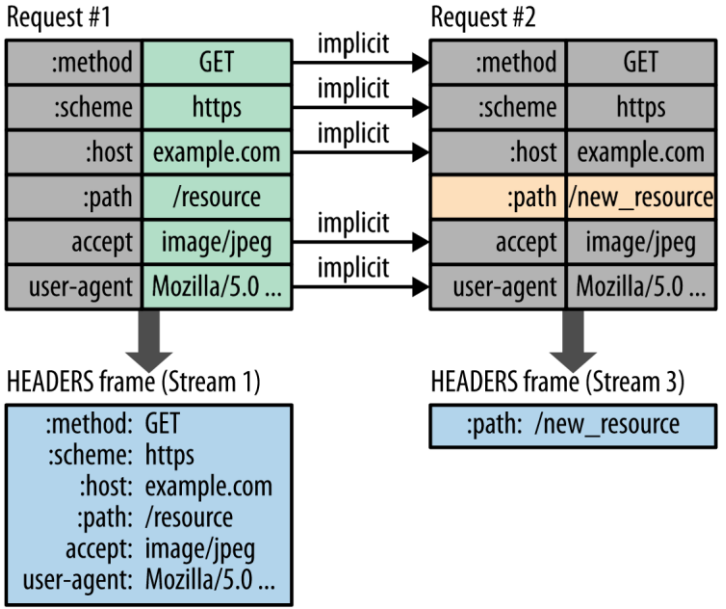
- Dependencies: A resource is transferred ahead of its dependencies
- Weights: Resources with same parent are send in proportion to their weight
- Browser sets priority automatically



Features: Header Compression

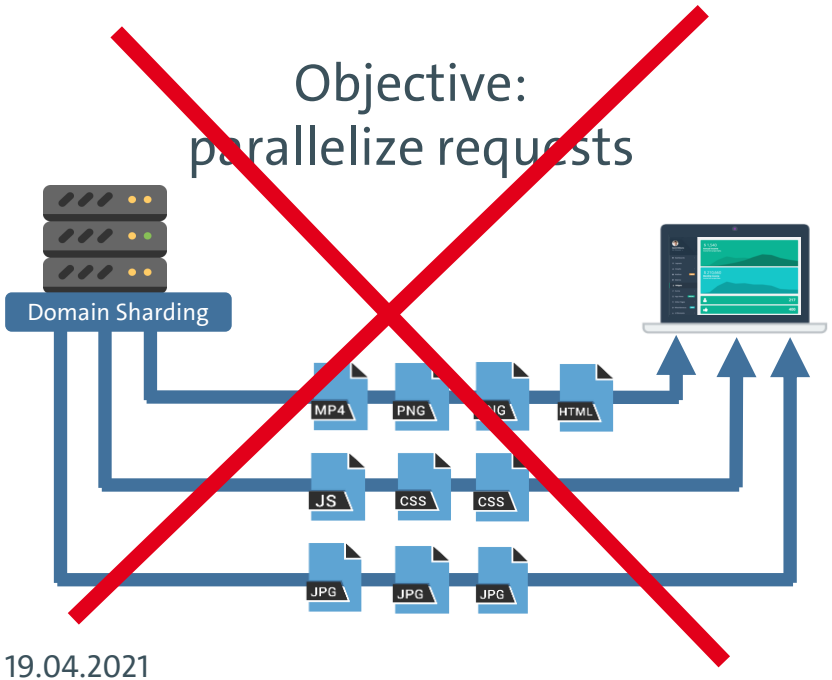
HPACK Compression:

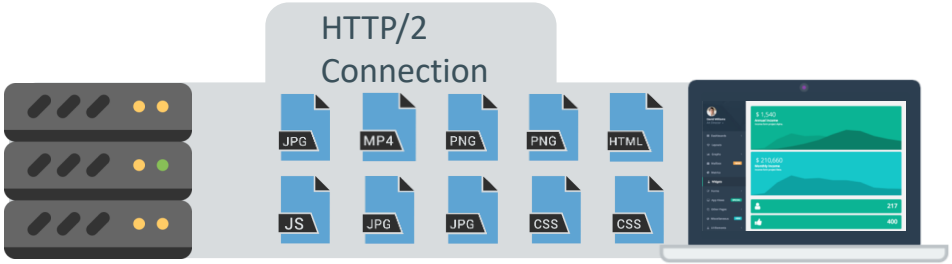
- Dictionaries on client and server for sent headers
  - Static dictionary for common headers
  - Dynamic dictionary for other headers
- Huffman encoding for header fields



Ilya Grigorik - High Performance Browser Networking – HTTP/2
https://hpbn.co/http2/

Optimization: Domain Sharding

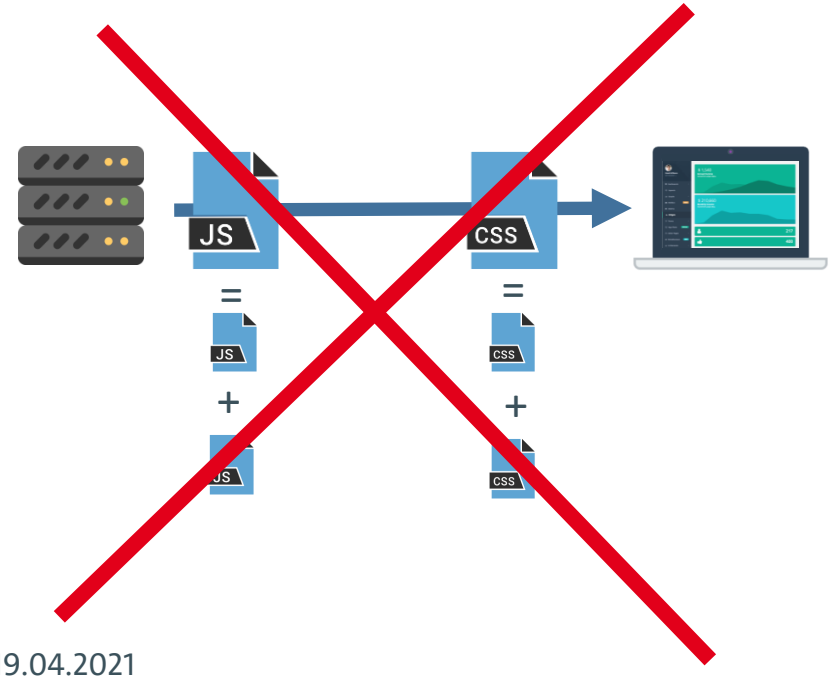




Advantages

- Unlimited parallelism on one connection
- Less protocol overhead (1 vs 6 TLS handshakes)
- Better resource prioritization
- Better congestion control

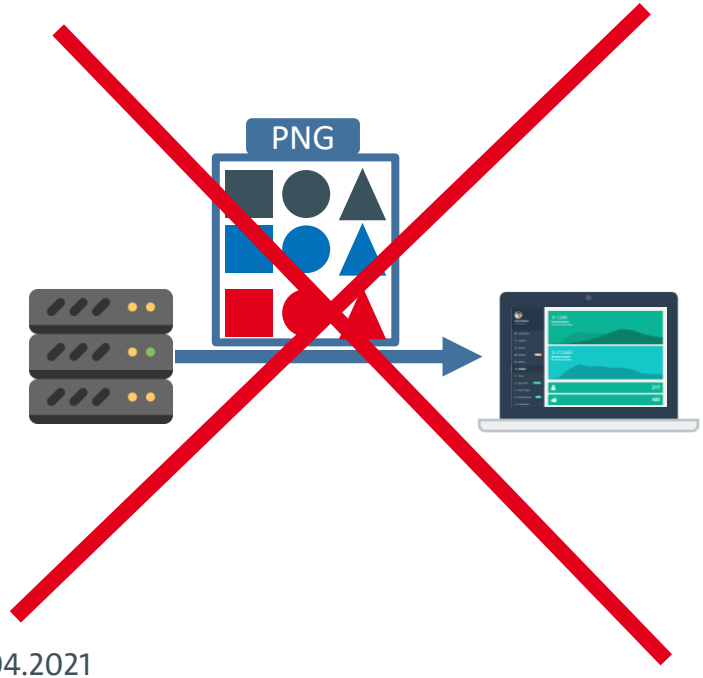
Optimization: Concatenation

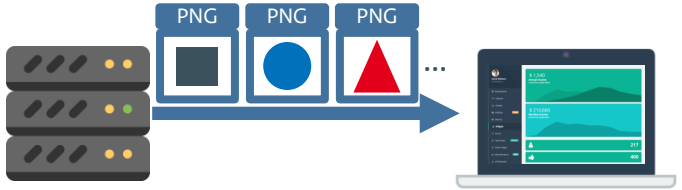


Advantages

- Better prioritization
- Individually cachable

Optimization: Spriting

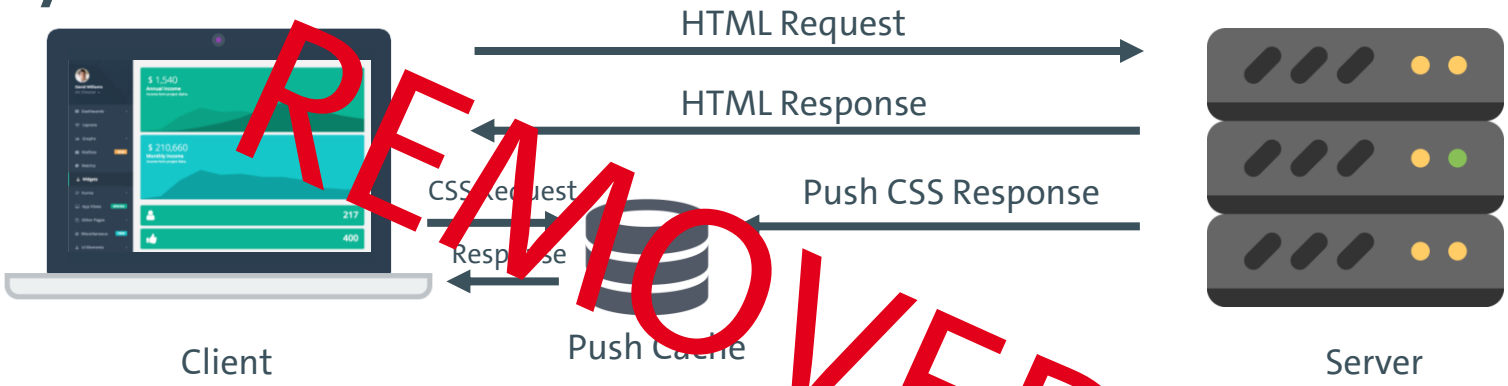




Advantages

- Better prioritization
- Individually cachable

HTTP/2: Push



REMOVED

- Server pushes needed resources proactively to the client
- Browser request is answered immediately from push cache

Saved:

- one roundtrip for the request
- several roundtrips for download

HTTP/3

Problem: HTTP/2 Head-Of-Line Blocking

- Now on TCP Level
- 1 Connection = Everything is blocked
- Performance may be worse then HTTP/1.1 (Bad Connection)

HTTP/3: TCP → UDP (QUIC)

- Semantics Stay
- Browser Support
- (Only) Enabled in Chrome
- Additional Computing Overhead

- Quic
  - Secure by Default
  - Independed Requests
  - Fast Network Switch

Compression

HTTP Text Compression

Content Encoding	Desktop	Mobile	Combined
No text compression	60.06%	59.31%	59.67%
Gzip	30.82%	31.56%	31.21%
Brotli	9.10%	9.11%	9.11%
Other	0.02%	0.02%	0.02%

Web Almanac 2020, Chapter 19: Compression
https://almanac.httparchive.org/en/2020/compression

Deflate vs. Brotli

- Encoded Dictionary
- Decompression:
  - ~443 MB/s - ~484 MB/s
- Compression:
  - ~146 MB/s - ~32 MB/s

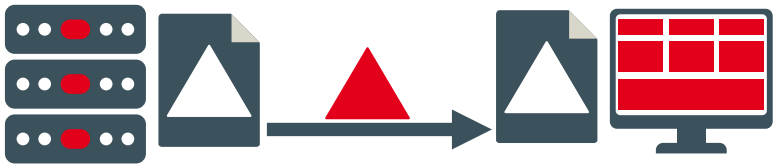
- Static Dictionary
- Decompression:
  - ~441 MB/s - ~508 MB/s
- Compression:
  - ~145 MB/s - ~0.6 MB/s

J. Alakuijala, E. Kliuchnikov, Z. Szabadka, L. Vandevenne: Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms, 2015

The Forgotten Ones

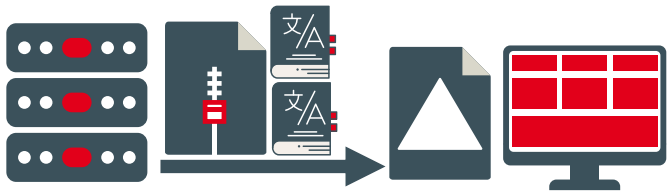
Delta Encoding

- Reuse Stale Content
- Only Updates

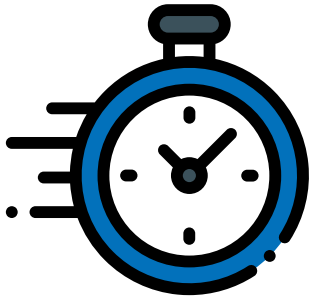


SDCH

- Predecessor of Brotli
- Dynamic Dictionaries

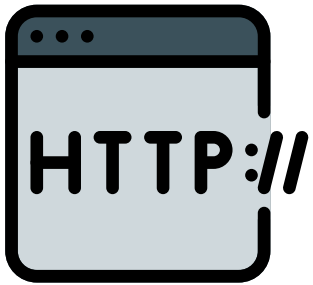


Summary



Reduce **Latency** by **Caching**

Optimize **TCP, DNS** and **TLS**



Make use of new **HTTP** Features

Tutorial 18: Going for Speed

09:00	Introduction
09:15	Part 1: Efficient Frontend Design
09:40	Q&A
09:50	Coffee Break
10:00	Part 2: High-Performance Networking
10:40	Q&A
10:50	Coffee Break
11:00	**Part 3: Scalable Backend Architectures**
11:40	Q&A
11:50	Coffee Break
12:00	Part 4: Performance Tracking & Analysis
12:00	The Core Web Vitals (Google Guest Speaker!)
12:30	Measuring Web Performance
12:50	Q&A

Baqend