# Web?

**2022** — RFC 9114 : HTTP/3

**2015** — RFC 7540 : HTTP/2


**1999** — RFC 2616 : HTTP/1.1

**1994** — Création du W3C® → Organisme de standardisation du World Wide Web

**1990** — Tim Berners-Lee invente le Web pour le CERN →
- HTTP
- Adresse Web (URI)
- HTML

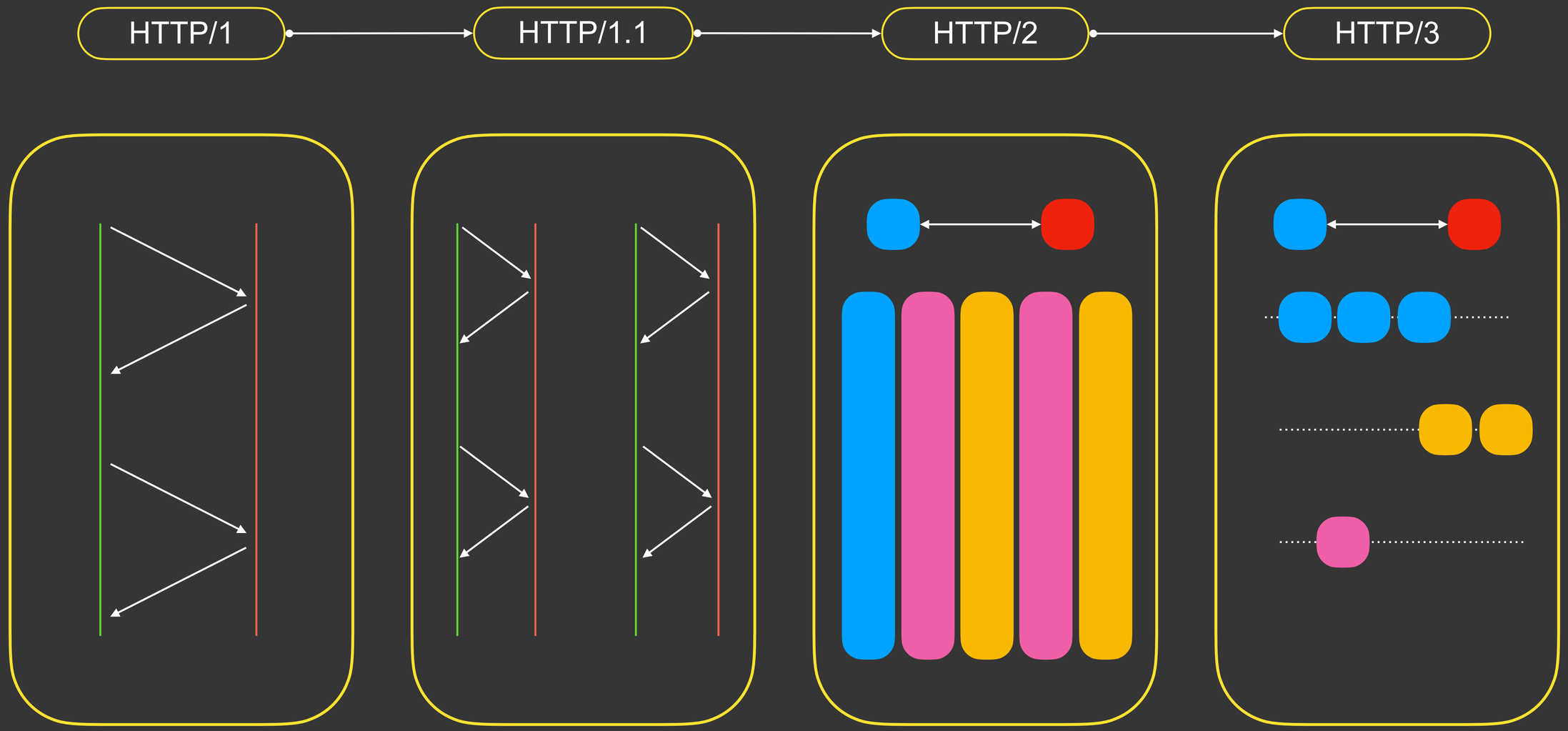
**1984** — Paul Mockapetris invente DNS → Le réseau Internet est adressage par les humains

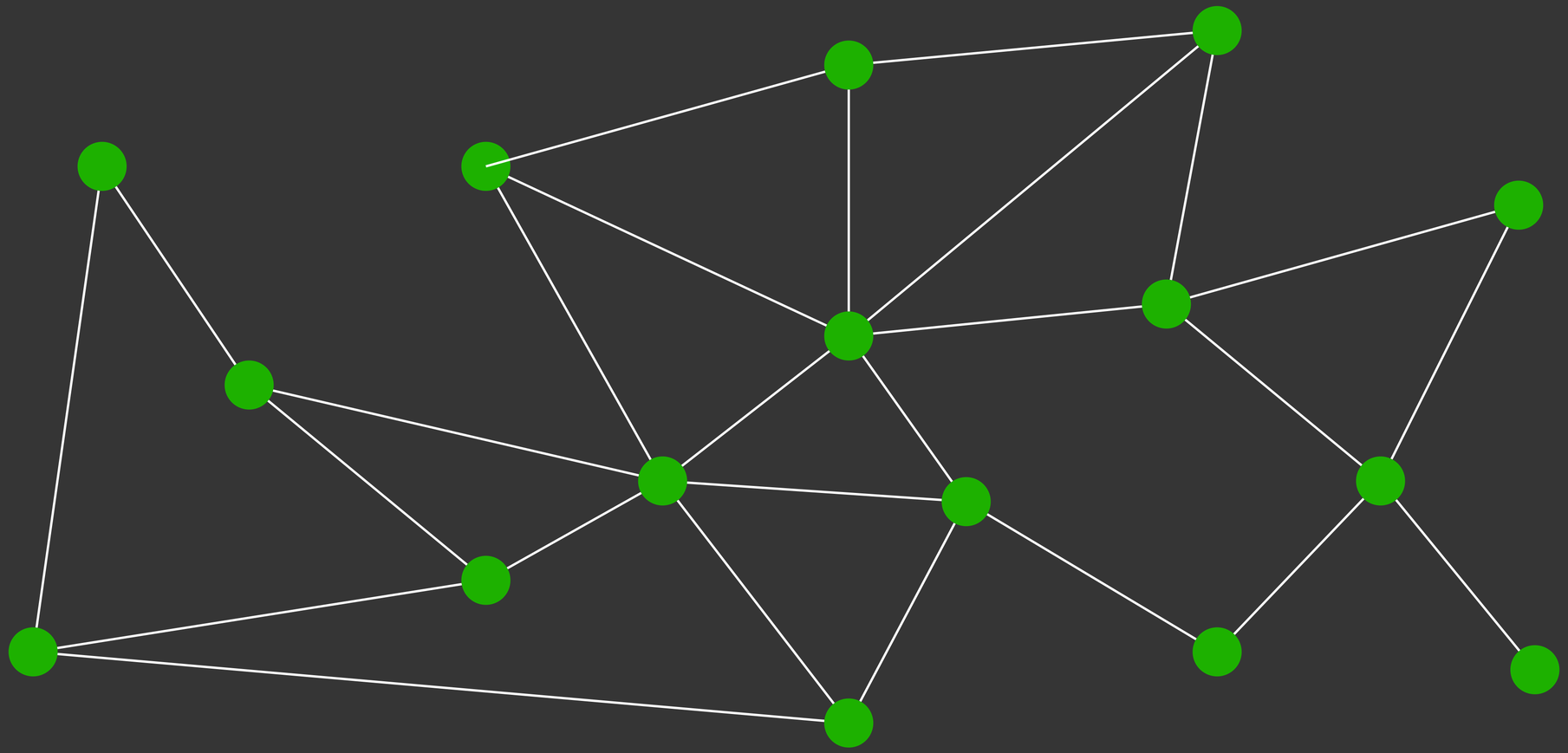
**1971** — RFC 114 : FTP → Transfert de fichiers

# Hypertext Transfer Protocol

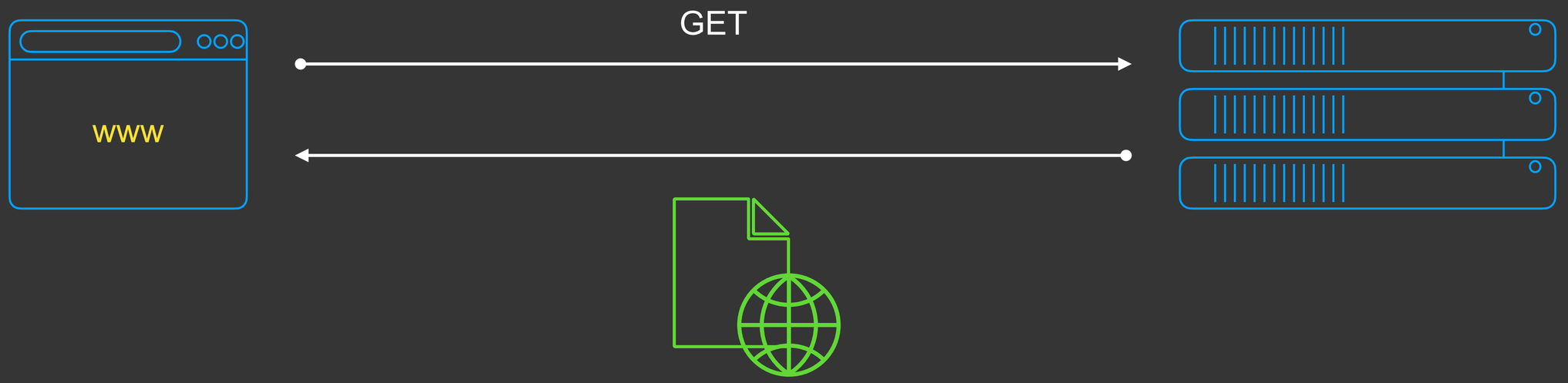
RFC 2616 / 7230 / 7540

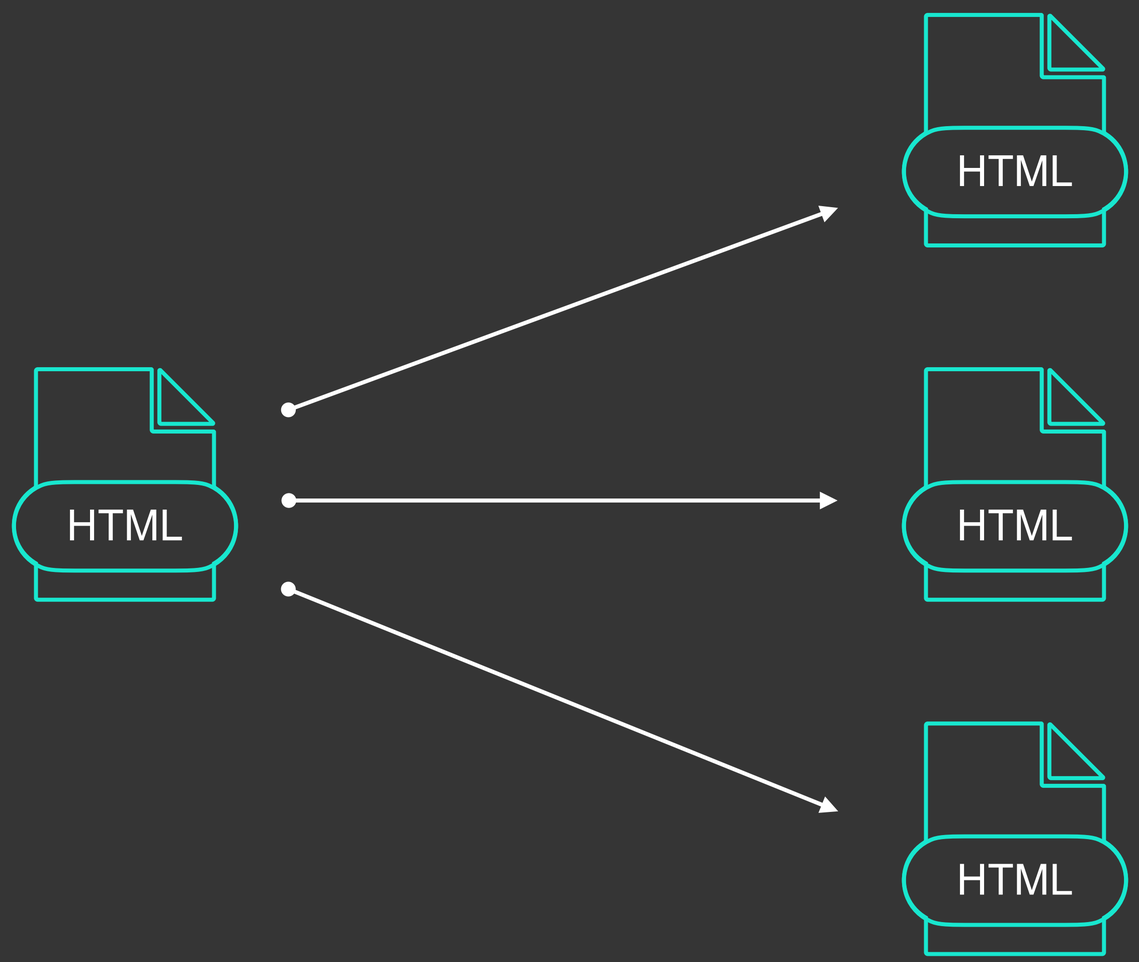
The Web protocol

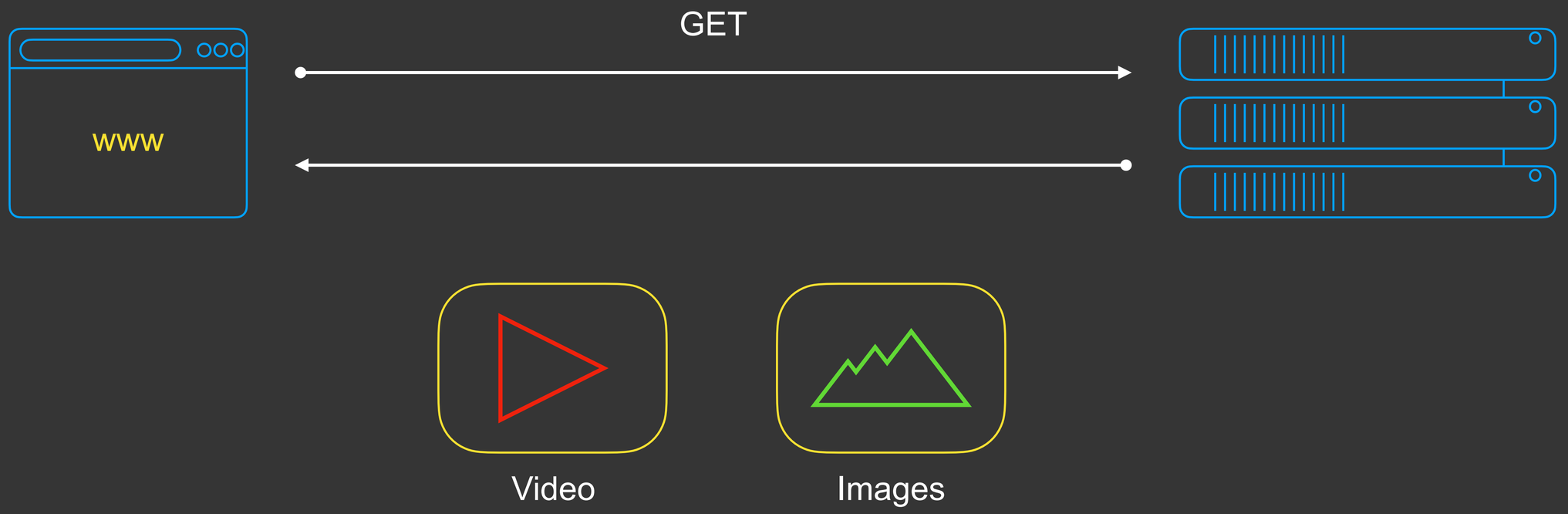

HTTP/1
HTTP/1.1
HTTP/2
HTTP/3

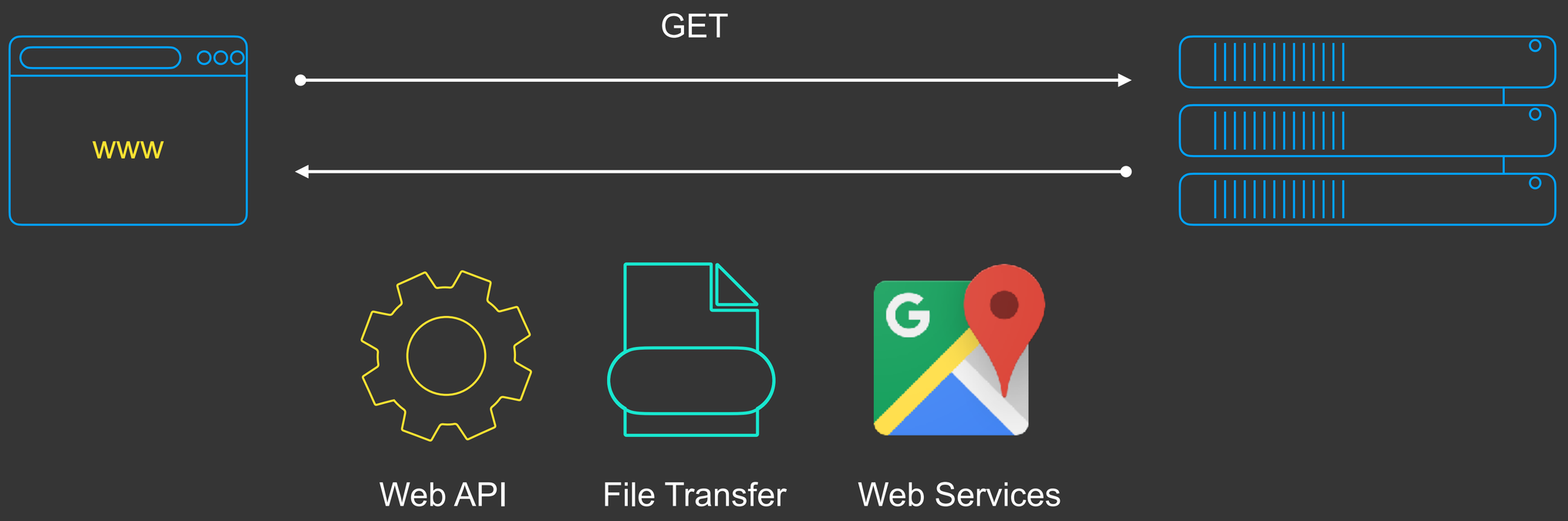


HTTP = The backbone of the Web


GET
www


HTML
HTML
HTML
HTML


GET
www
Video
Images

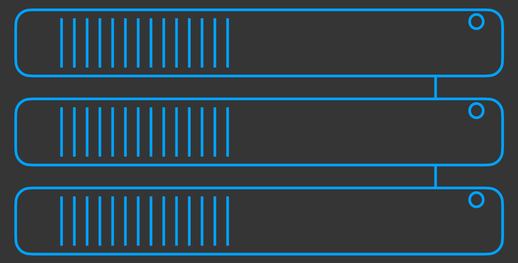

GET
www
Web API
File Transfer
Web Services

# HTTP Protocol

## Application Layer Protocol

TCP Port 80 : HTTP
TCP Port 443 : HTTPS


Client

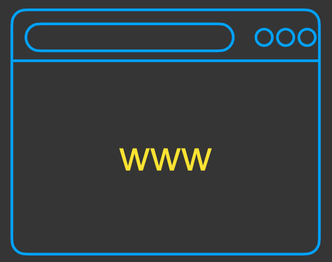


Web Server

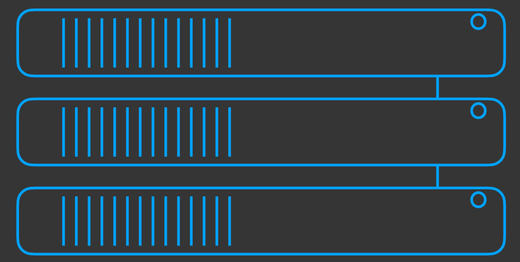
# HTTP Protocol

Request a ressource

Respond requested resource

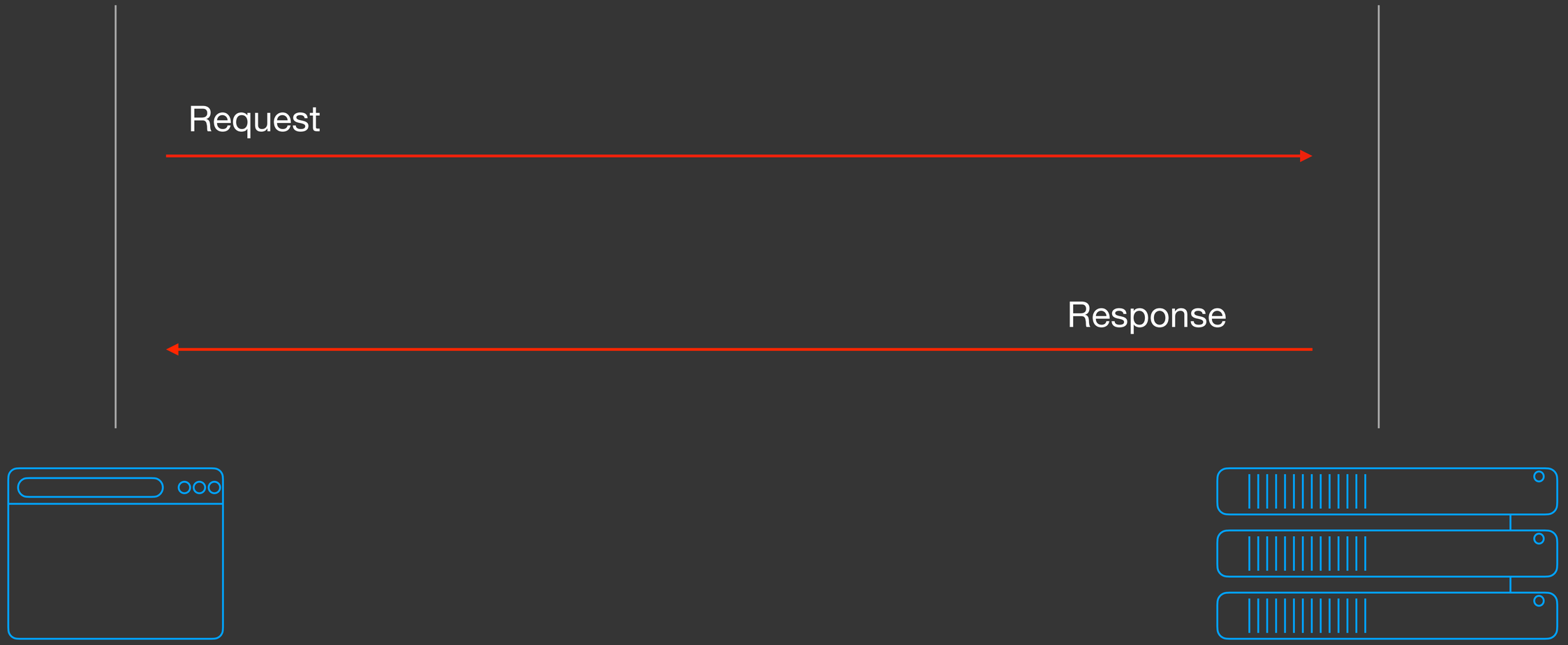


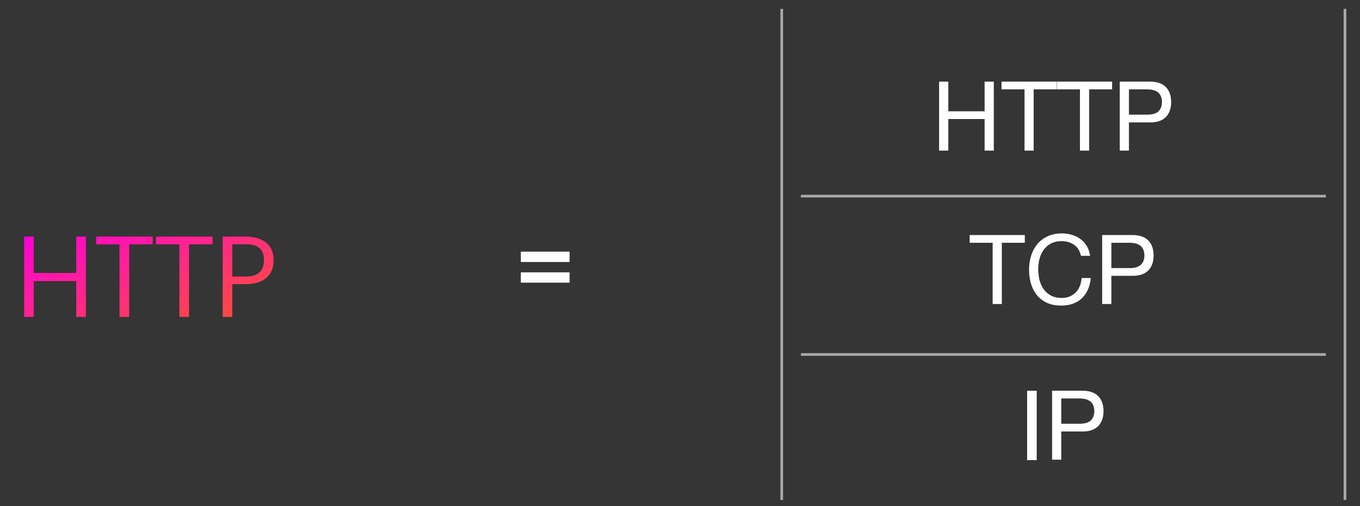

Client



Web Server

# HTTP Protocol




HTTP
=
HTTP
TCP
IP

# HTTP Workflow

UDP

www.pizza.com ?

33.33.33.33

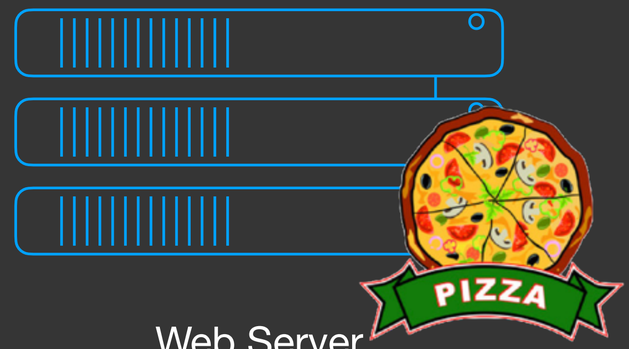
DNS

TCP

HTTP Request

HTTP Response


Client

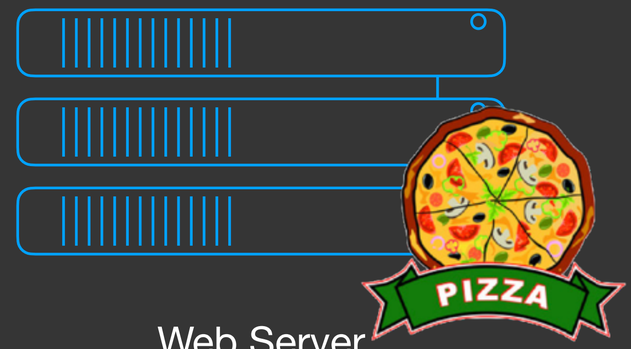



Web Server

# HTTP Request

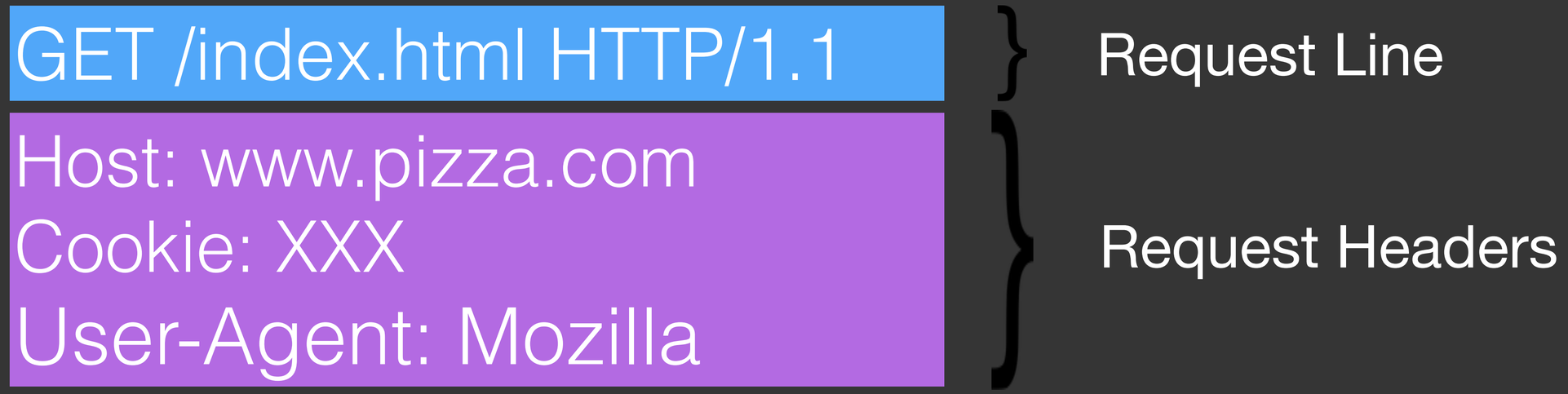
GET / HTTP/1.1
Host: www.pizza.com
Cookie: XXX
User-Agent: Mozilla


Client

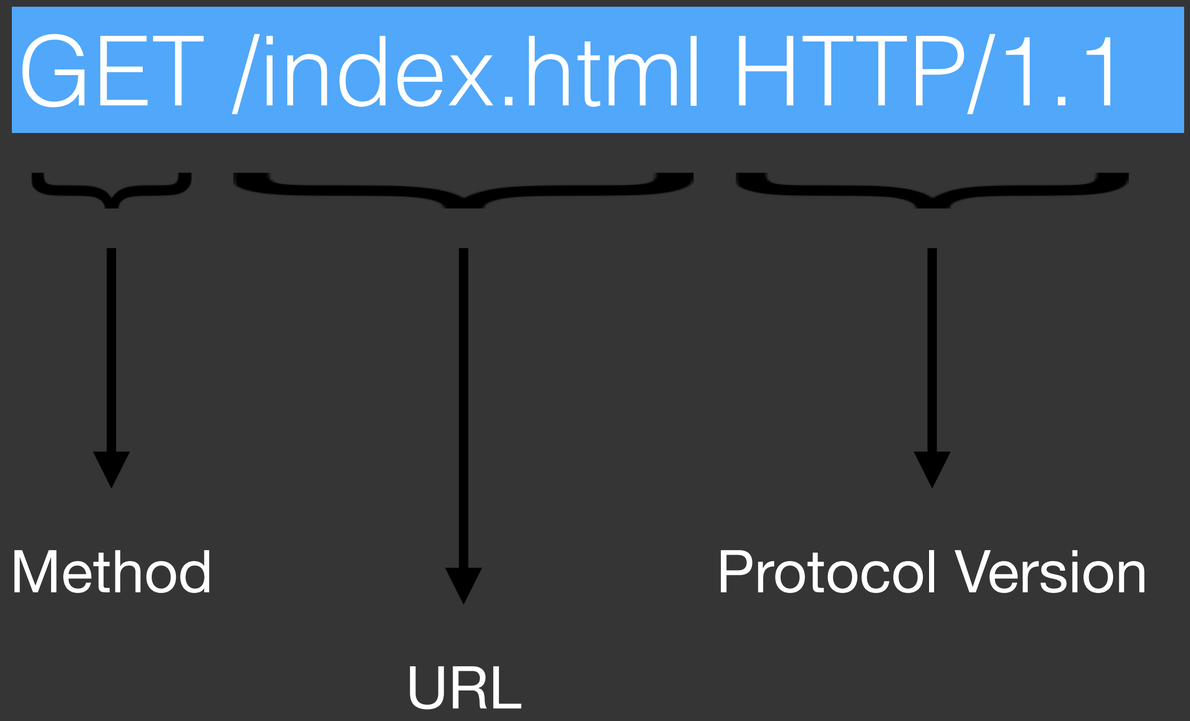



Web Server

# HTTP Request



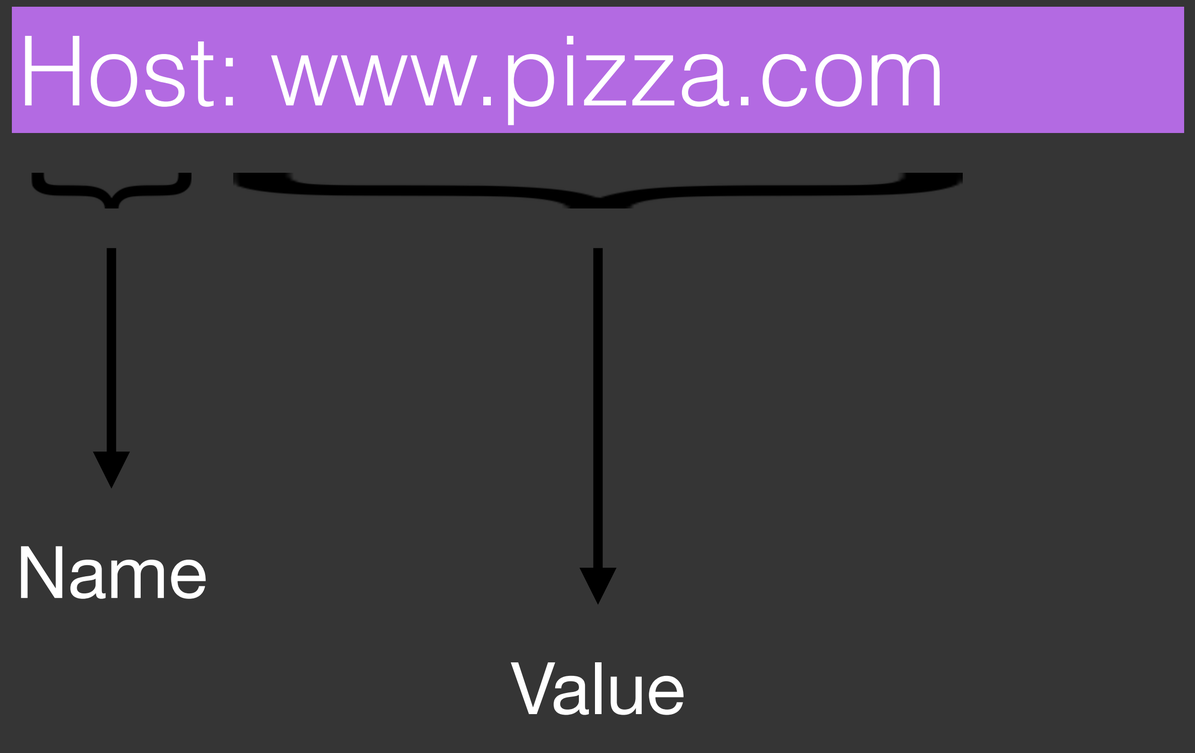
# Request Line



# Method HTTP

| | |
|---|---|
| **GET** | Requests a representation of the specified resource |
| **HEAD** | Response identical to that of a GET request, but without the response body |
| **POST** | Requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI |
| **OPTIONS** | Returns the HTTP methods that the server supports for the specified URL |
| **CONNECT** | Converts the request connection to a transparent TCP/IP tunnel |
| **TRACE** | Echoes the received request so that a client can see what (if any) changes or additions have been made by intermediate servers. |
| **PUT** | Requests that the enclosed entity be stored under the supplied URI. |
| **DELETE** | Deletes the specified resource |

# Request Headers



# Request Headers

## Exemple

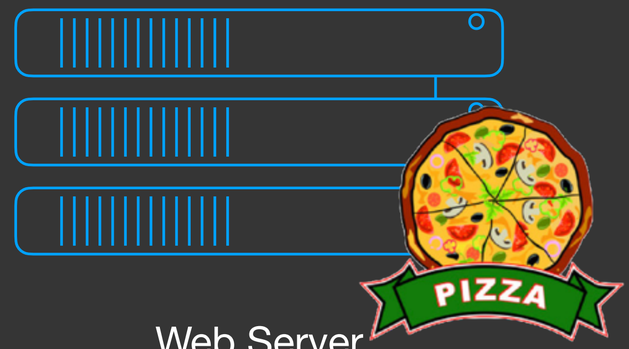
| | |
|---|---|
| Host | Used for Name based Virtual Host |
| User-Agent | Software used by the end user |
| Cookie | An HTTP cookie previously sent by the server with Set-Cookie |
| Referer | Identifies the address of the webpage |
| Date | The date and time that the message was originated |

# HTTP Response

HTTP/1.1 200 OK
Server: Apache
Content-Type: text/html

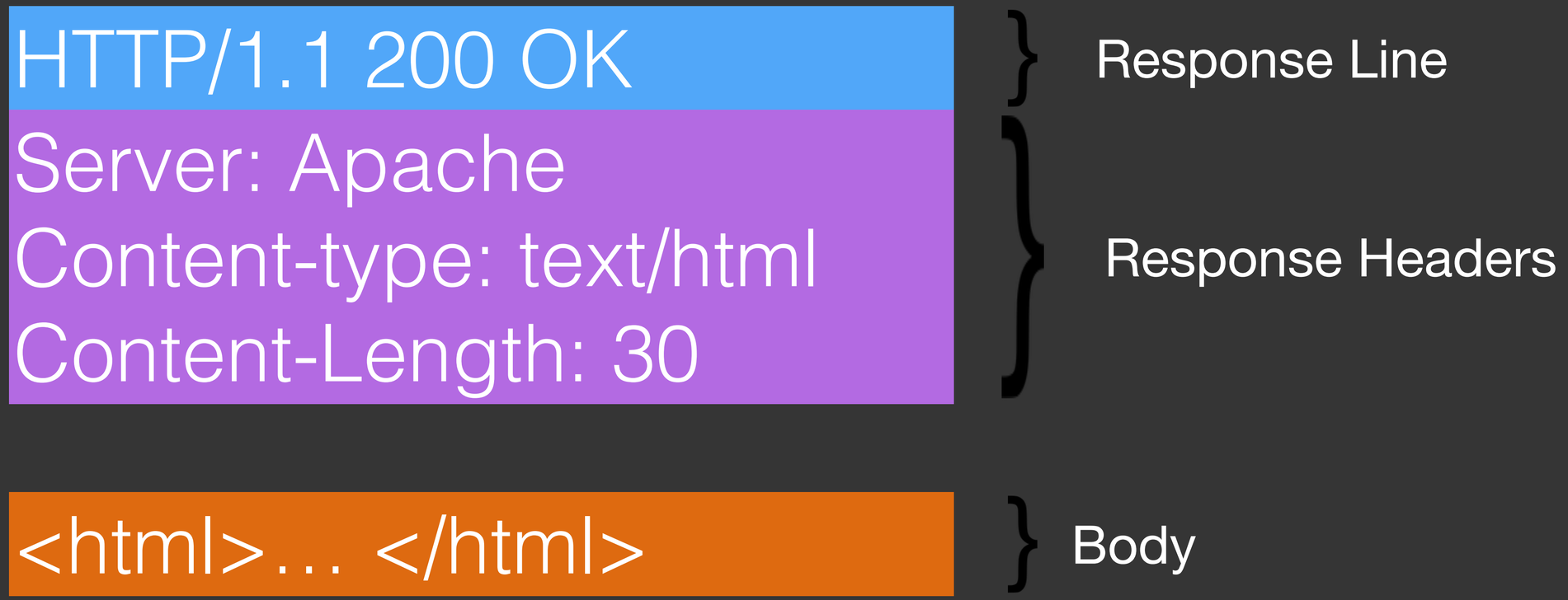
```
<html>
<body>
  Welcome Bob
</body>
</html>
```


Client

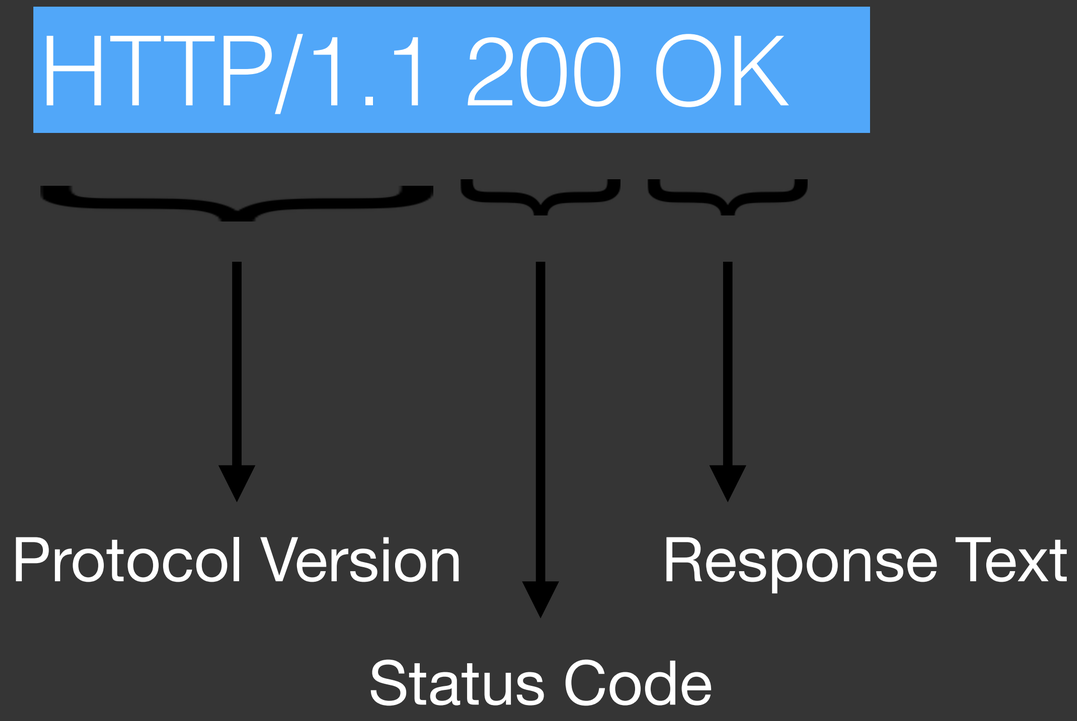



Web Server

# HTTP Response

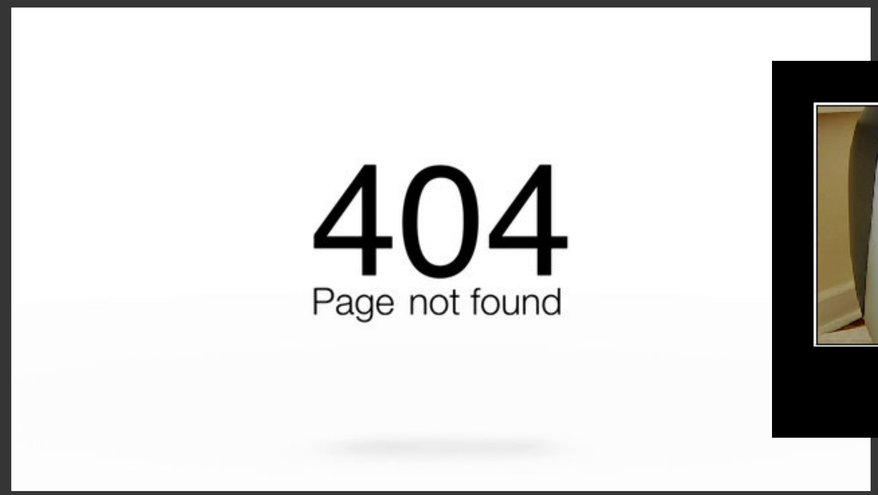


# Response Line



# Status Code

| | |
|---|---|
| 1XX | Information |
| 2XX | Succes |
| 3XX | Redirection |
| 4XX | Client Error |
| 5XX | Server Error |




# Response Headers

## Exemple

| | |
|---|---|
| **Server** | Name of the Server |
| **Content-Type** | The MIME type of the body of the request |
| **Content-Length** | The length of the request body in octets |
| **Set-Cookie** | Set a new cookie value |
| **Last-Modified** | The last modified date for the requested object |

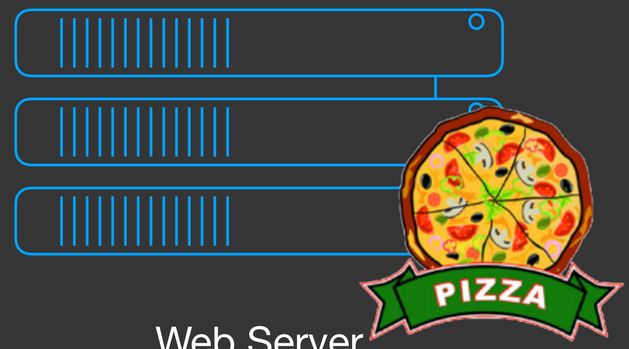
# Login

GET / HTTP/1.1
Host: www.pizza.com
User-Agent: Mozilla

HTTP/1.1 200 OK
Server: Apache
Content-Type: text/html
Content-Length: 128

```
<form action="/login.php">
  Login:<br>
  <input type="text" name="login"><br>
  Password:<br>
  <input type="text" name="password"><br><br>
  <input type="submit" value="Submit">
</form>
```


Client




Web Server

# Set-cookie

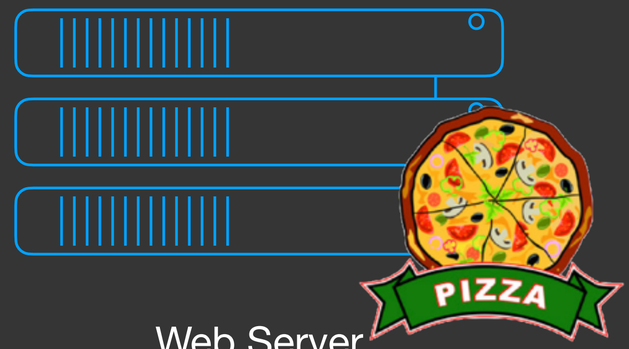
```
POST /login.php HTTP/1.1
Host: www.pizza.com
User-Agent: Mozilla
Content-Length:25
```

```
login=Bob&password=voili
```

```
HTTP/1.1 200 OK
Server: Apache
Content-Type: text/html
Set-Cookie: SESSION=GHAK89SJASHJ
```

```
<html>
<body>
<h1>Welcome Bob</h1>
<a href="commandList">List Pizza command</a>
</body>
</html>
```


Client




Web Server

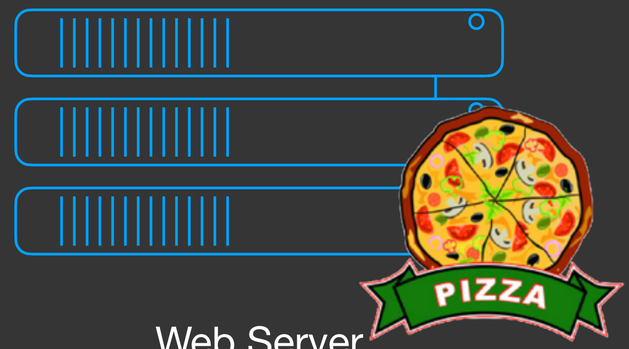
# Cookie

GET /commandList HTTP/1.1
Host: www.pizza.com
User-Agent: Mozilla
Cookie: SESSION=GHAK89SJASHJ

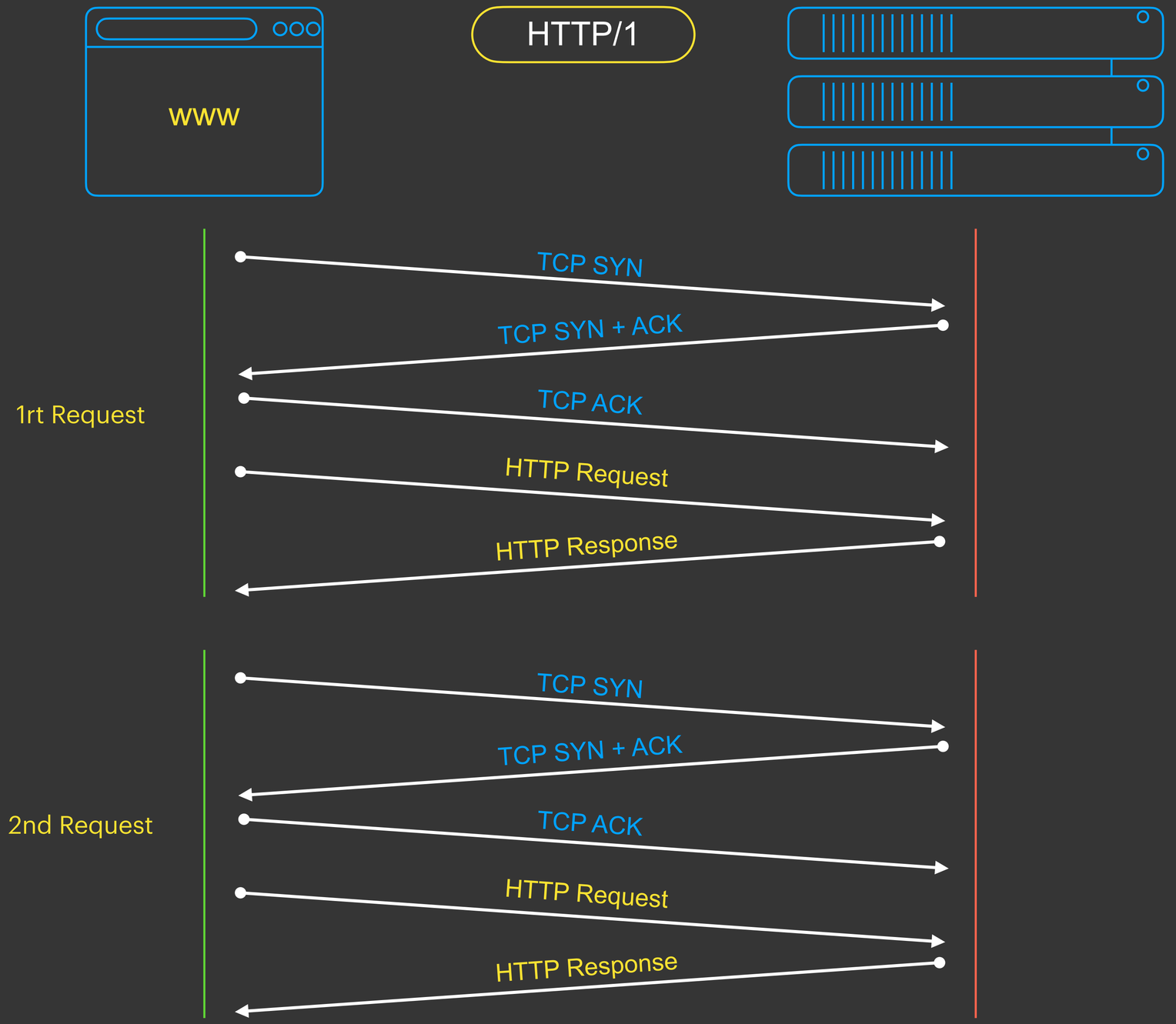
HTTP/1.1 200 OK
Server: Apache
Content-Type: text/html
Content-Length: 32

```
<html>
<body>
<ul>
  <li>Royal</li>
  <li>Figateli</li>
</ul>
</body>
</html>
```

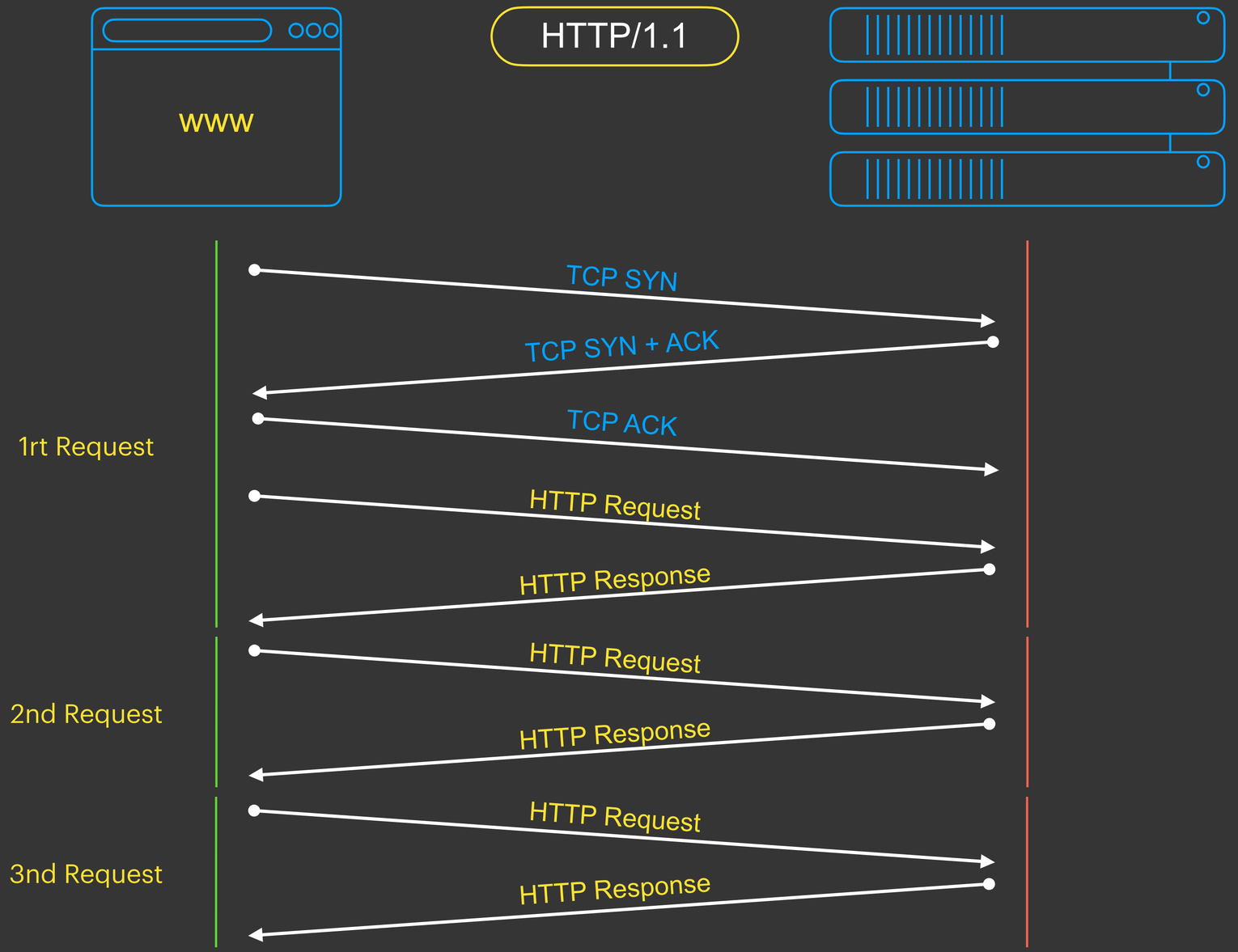

Client



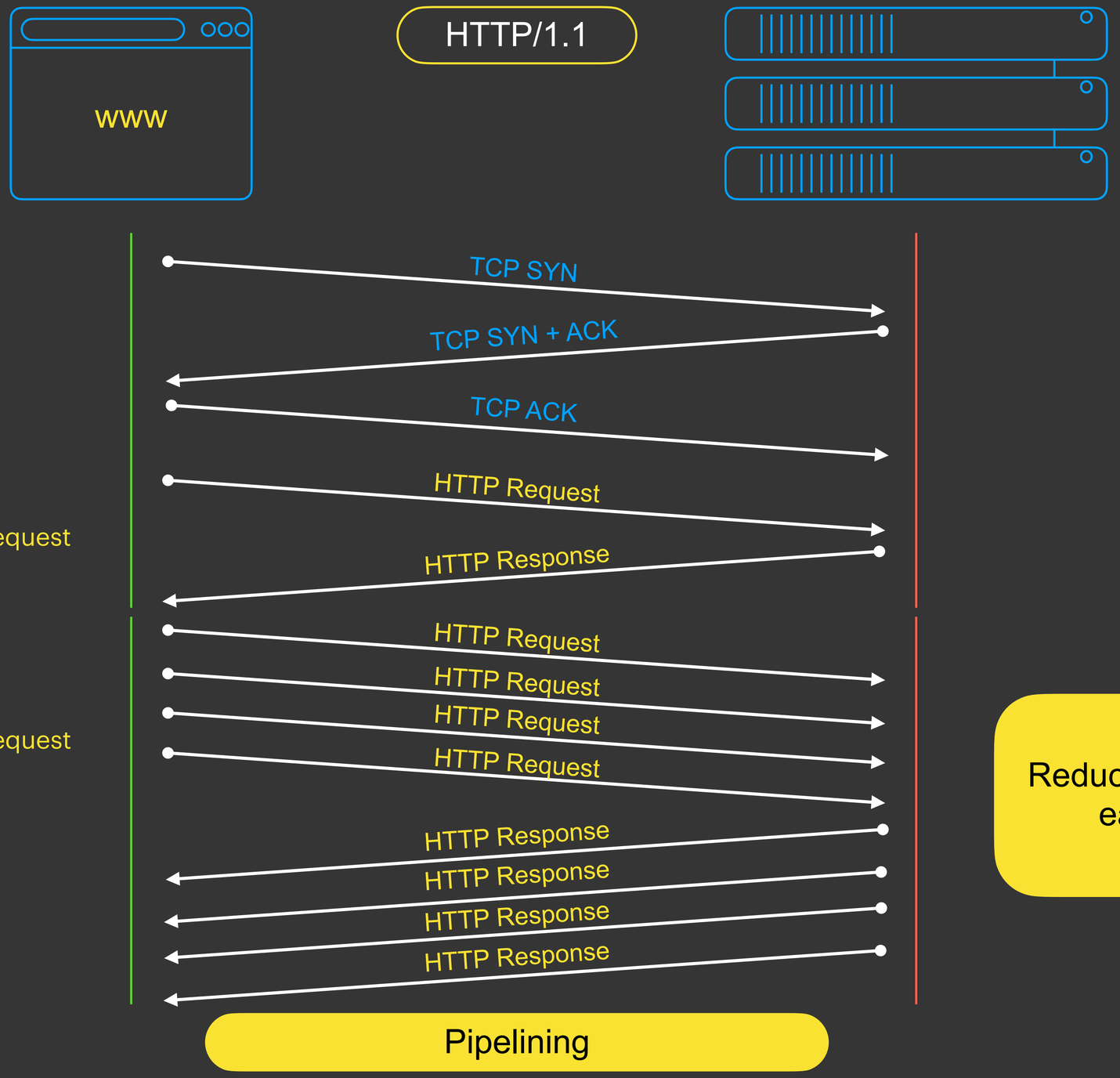

Web Server


www
HTTP/1
1rt Request
TCP SYN
TCP SYN + ACK
TCP ACK
HTTP Request
HTTP Response
2nd Request
TCP SYN
TCP SYN + ACK
TCP ACK
HTTP Request
HTTP Response

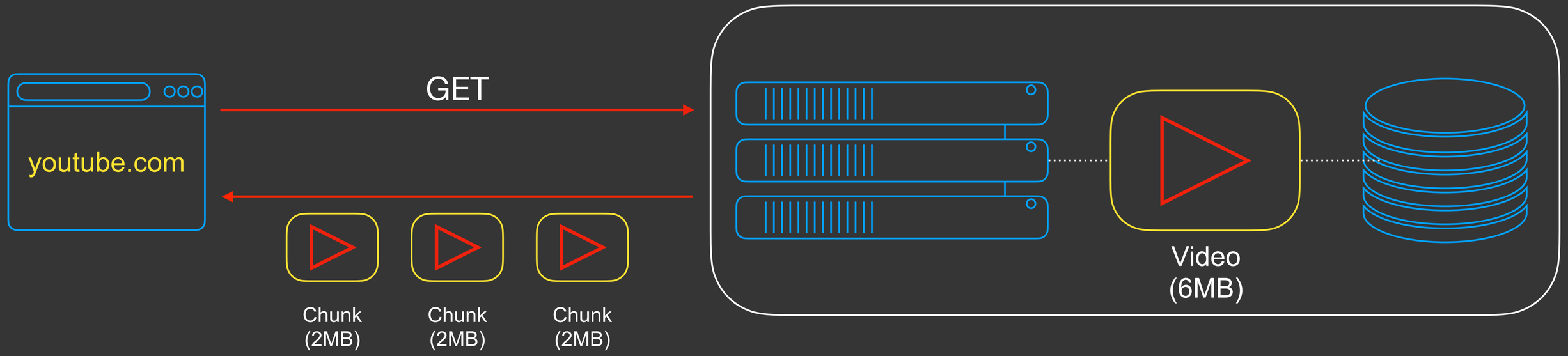
HTTP/1.1




Persistent HTTP Connections

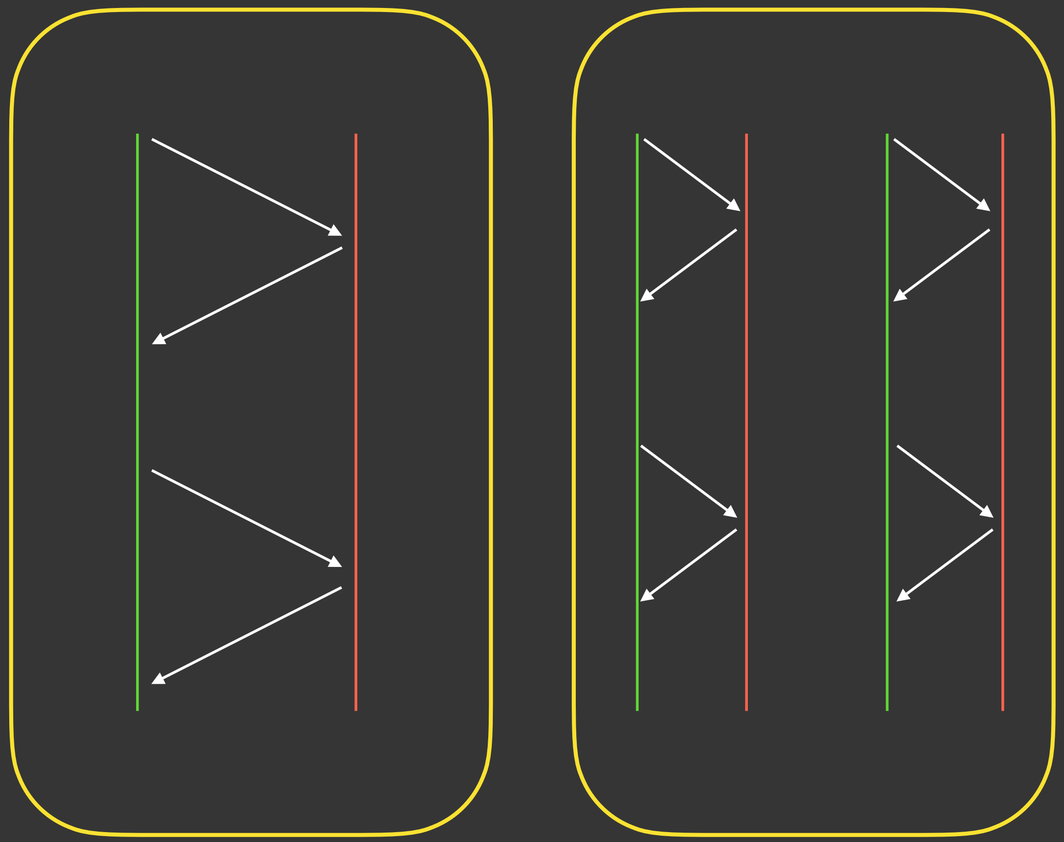

www
HTTP/1.1
TCP SYN
TCP SYN + ACK
TCP ACK
HTTP Request
HTTP Response
1rt Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
2nd, 3nd, 4nd, 5n Request
HTTP Response
HTTP Response
HTTP Response
HTTP Response
Pipelining
Reduce the wait time for each response

HTTP/1.1




Chunked transfer encoding

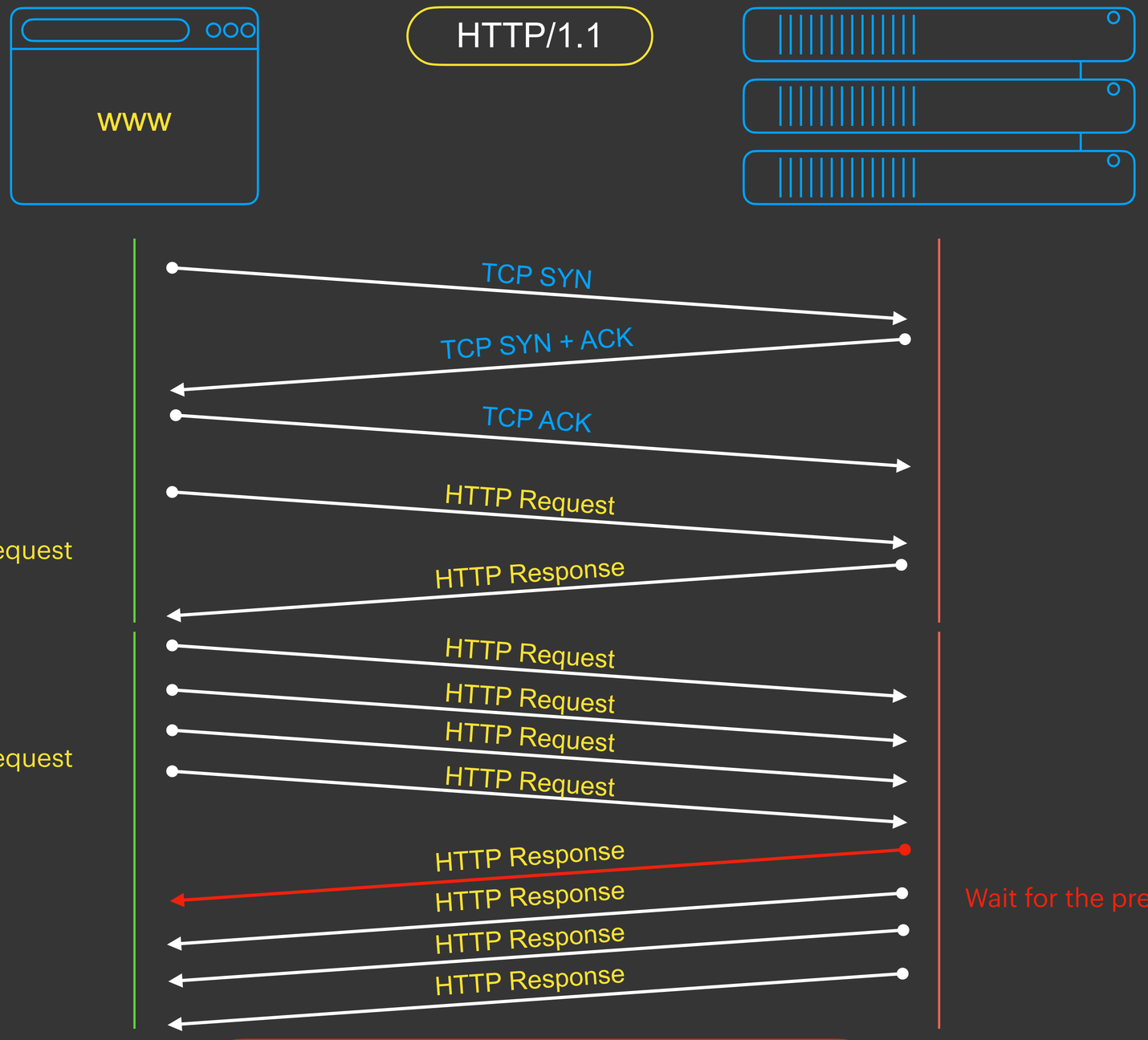
HTTP/1 → HTTP/1.1



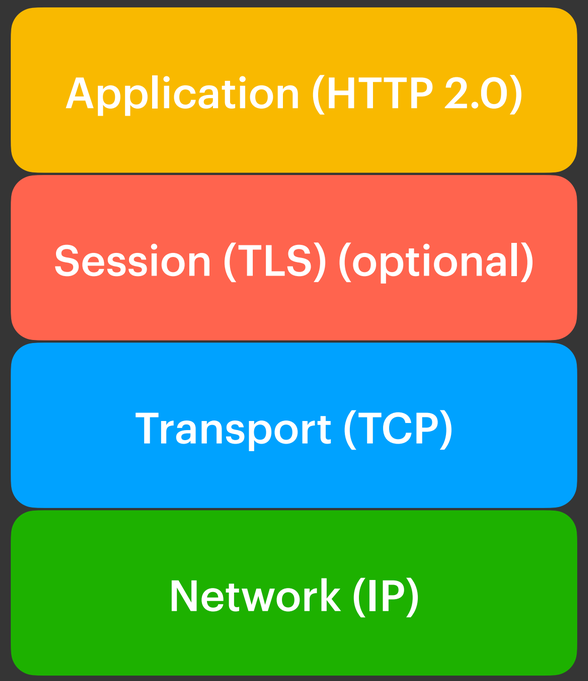
RFC 2616

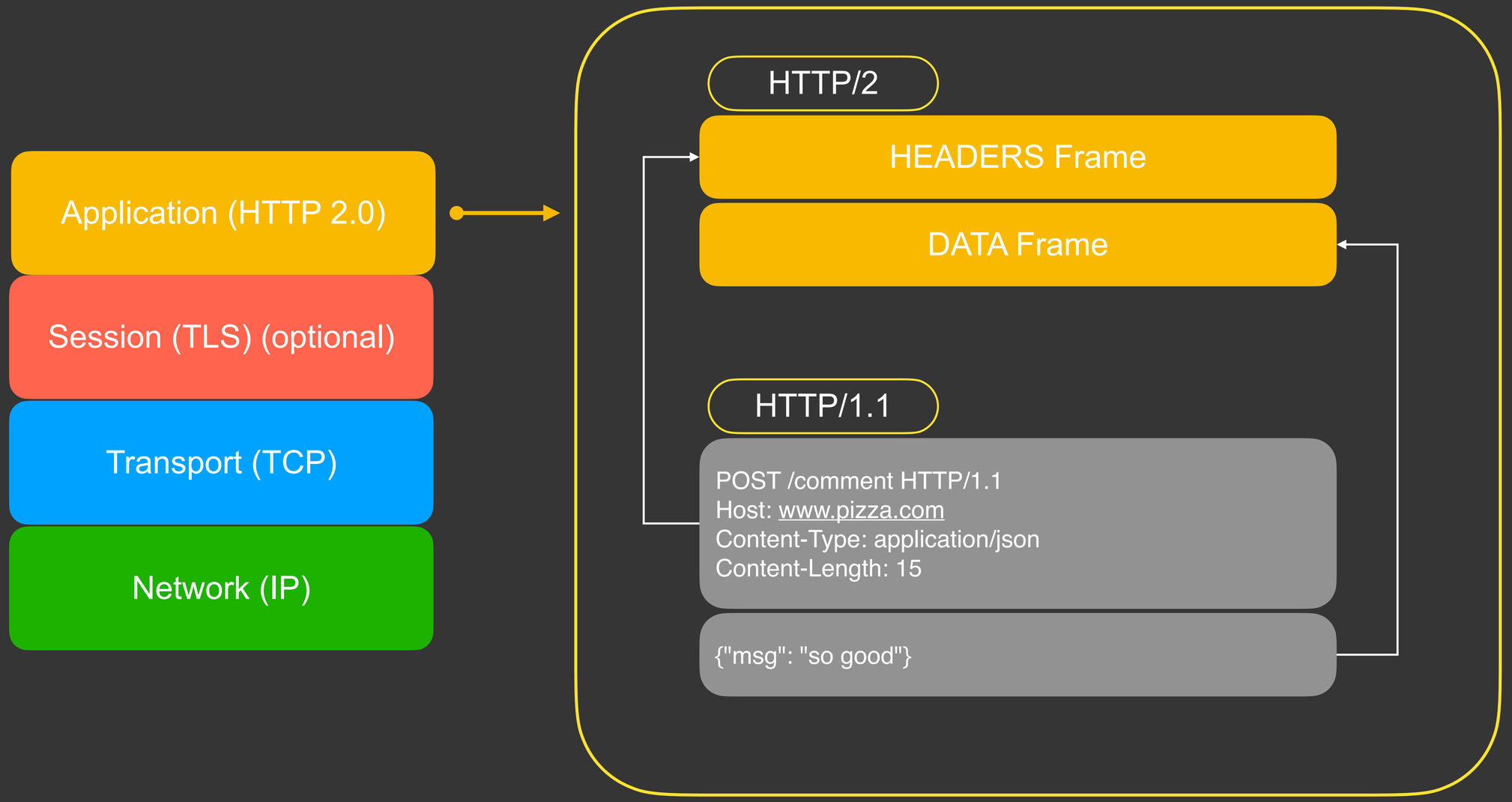
June 1999

- Virtual Host by Name
- Persistent Connections
- Pipelining
- Chunked transfer encoding


www
HTTP/1.1
TCP SYN
TCP SYN + ACK
TCP ACK
HTTP Request
HTTP Response
1rt Request
HTTP Request
HTTP Request
HTTP Request
HTTP Request
2nd, 3nd, 4nd, 5n Request
HTTP Response
HTTP Response
HTTP Response
HTTP Response
Wait for the previous request to complete
Pipelining Head of Line Blocking

# HTTP/2


Application (HTTP 2.0)
Session (TLS) (optional)
Transport (TCP)
Network (IP)


Application (HTTP 2.0)
Session (TLS) (optional)
Transport (TCP)
Network (IP)
HTTP/2
HEADERS Frame
DATA Frame
HTTP/1.1
POST /comment HTTP/1.1
Host: www.pizza.com
Content-Type: application/json
Content-Length: 15
{"msg": "so good"}
Binary Framing Layer

## HEADERS Frame (stream 1)

| | |
|---|---|
| :method: | GET / HTTP/1.1 |
| :path: | /index.html |
| :version: | HTTP/2.0 |
| :scheme: | https |
| Host: | pizza.com |
| user-agent: | firefox/132.0.2 |

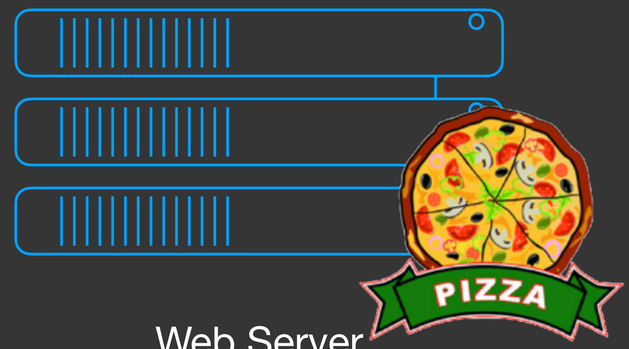
## HEADERS Frame (stream 1)

| | |
|---|---|
| :status: | 200 |
| :version: | HTTP/2.0 |
| server: | nginx/1.26 |
| set-cookie: | SESSION_ID=787889 |
| Content-type: | Text/html |

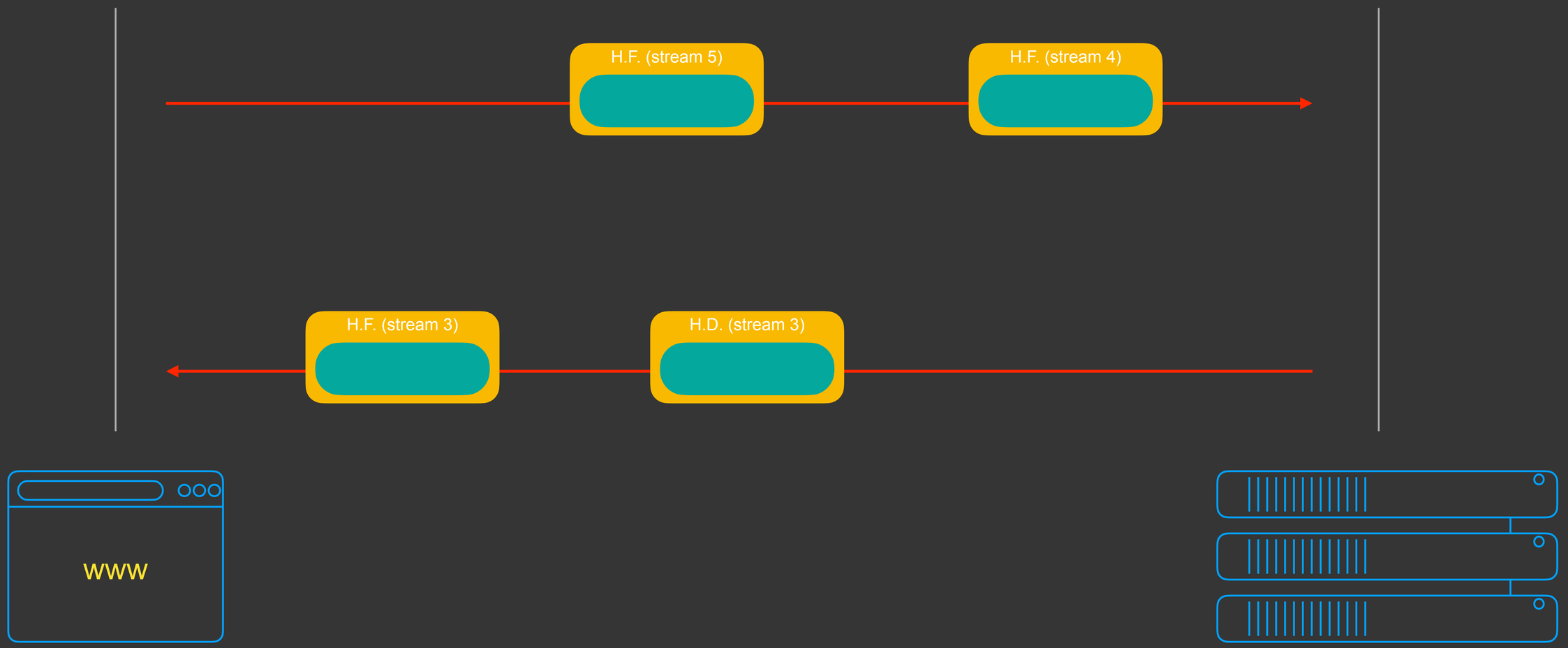
## DATA Frame (stream 1)

```
<html>
<body>
...
</body>
</html>
```


Client



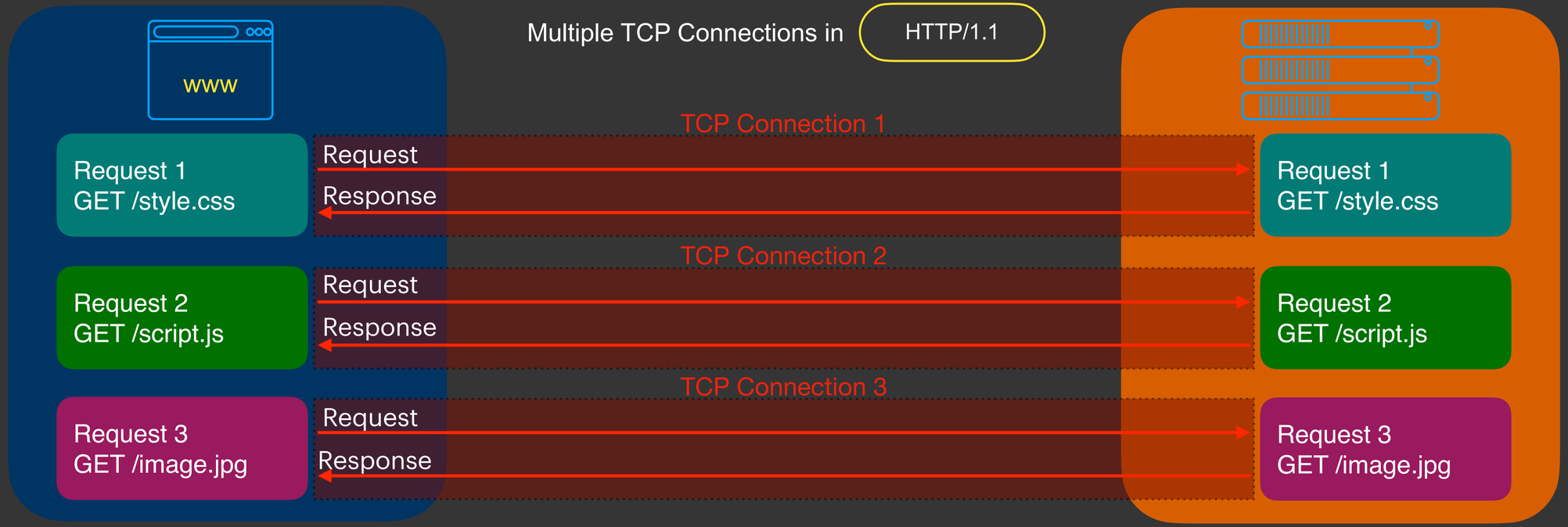

Web Server

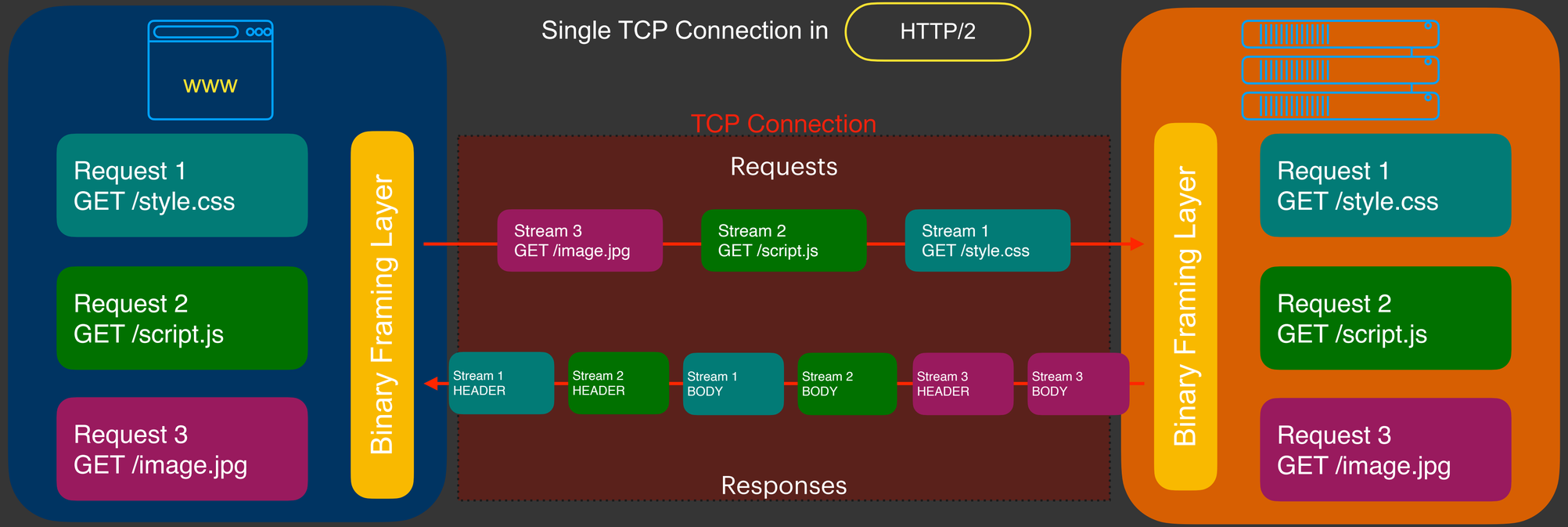

H.F. (stream 5)
H.F. (stream 4)
H.F. (stream 3)
H.D. (stream 3)
www
Client
Web Server

How to solve the **Pipelining Head of Line Blocking** Problem?

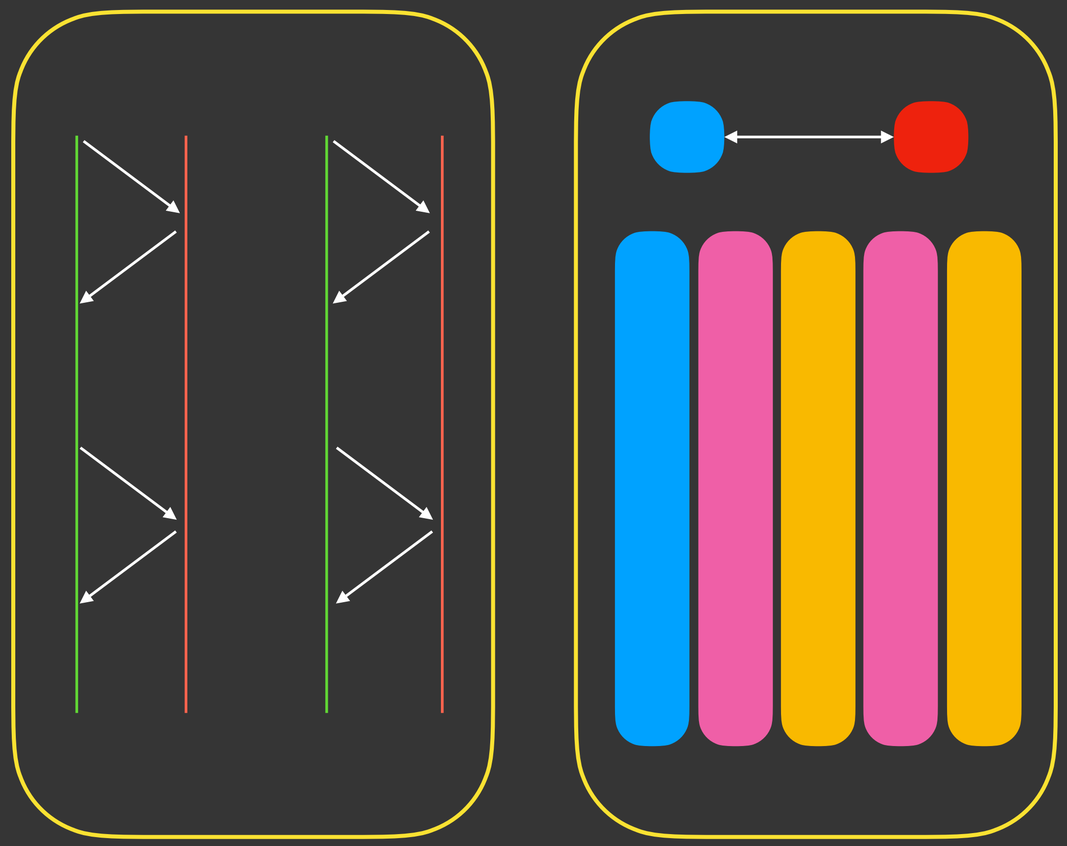
HTTP/1.1

HTTP/2


Multiple TCP Connections in
HTTP/1.1
www
Request 1
GET /style.css
Request 2
GET /script.js
Request 3
GET /image.jpg
TCP Connection 1
Request
Response
TCP Connection 2
Request
Response
TCP Connection 3
Request
Response
Request 1
GET /style.css
Request 2
GET /script.js
Request 3
GET /image.jpg


Single TCP Connection in
HTTP/2
www
Request 1
GET /style.css
Request 2
GET /script.js
Request 3
GET /image.jpg
Binary Framing Layer
TCP Connection
Requests
Stream 3
GET /image.jpg
Stream 2
GET /script.js
Stream 1
GET /style.css
Stream 1
HEADER
Stream 2
HEADER
Stream 1
BODY
Stream 2
BODY
Stream 3
HEADER
Stream 3
BODY
Responses
Binary Framing Layer
Request 1
GET /style.css
Request 2
GET /script.js
Request 3
GET /image.jpg

HTTP/1.1 → HTTP/2



RFC 7540 — May 2015

- Binary framing
- Multiplexed
- HPack Headers compression
- Push

# Web technologies

## Browser Side

HTML
CSS
Javascript
SVG
PNG
WebGL
DOM
MathML
WASM
...

***All W3C Standards***

## Server Side

Apache
Tomcat
Tomee
Microsoft IIS
Nodejs
Nginx

Python
PHP
Perl
Ruby
Java
Golang
Rust

***Open source or commercial***

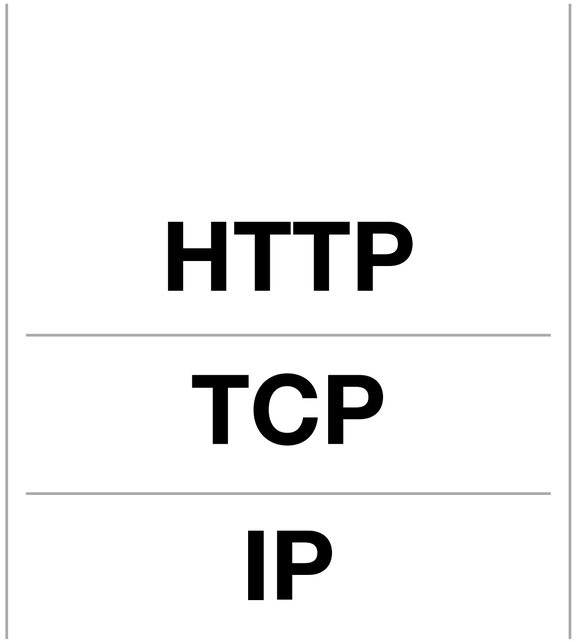
# Transport Layer Security

Ex-Secure Socket Layer

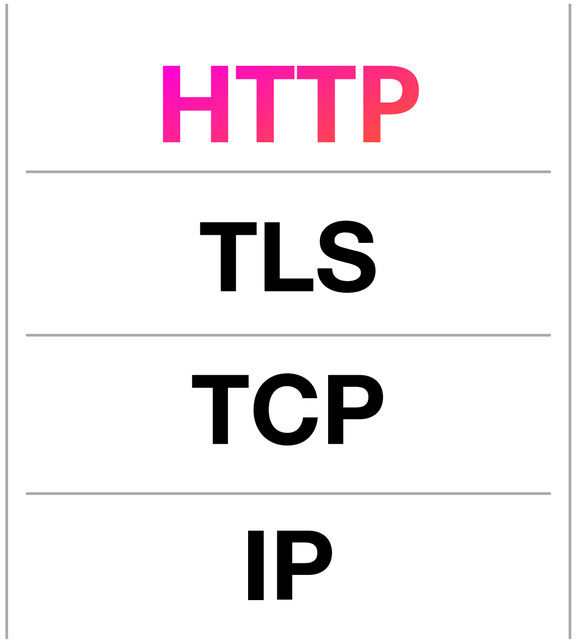
**RFC 2246 / 4346 / 5246**

# TLS : Security Properties

- **Confidentiality** of the communication between the client and the server
- **Integrity** of the communication between the client and the server
- **Authentication** of the server
  and optionally of the client

**HTTP** = 

**HTTPS** =

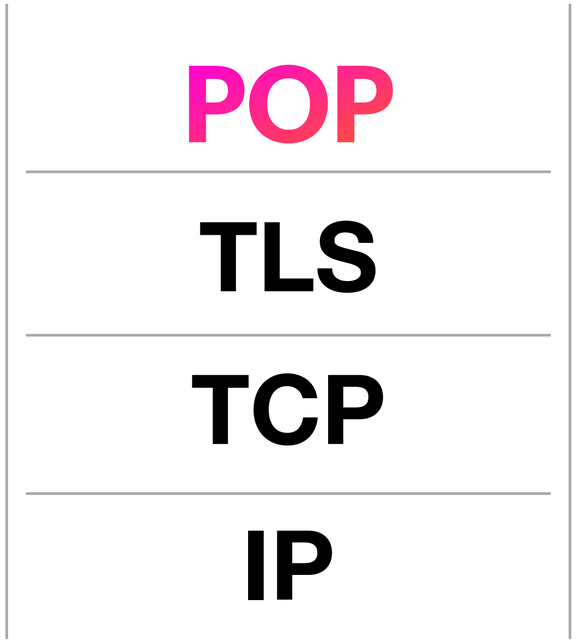


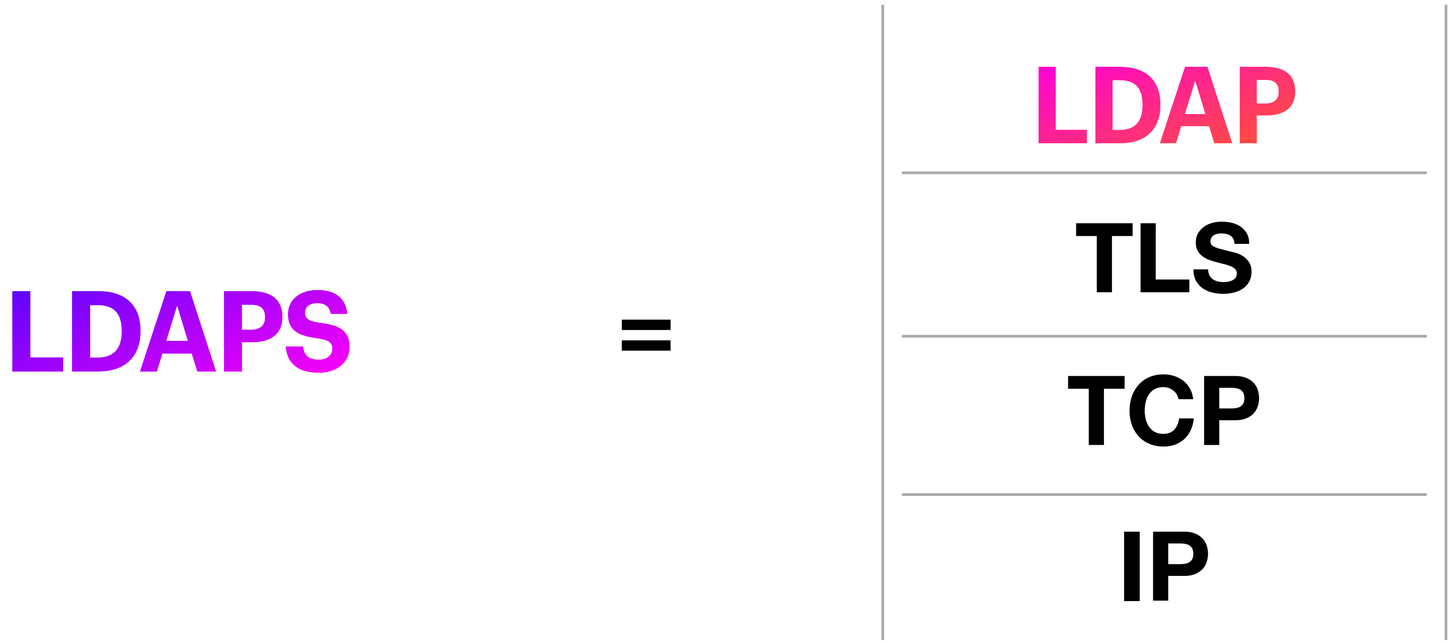
**SMTPS** =

| SMTP |
| --- |
| TLS |
| TCP |
| IP |

**IMAPS** =

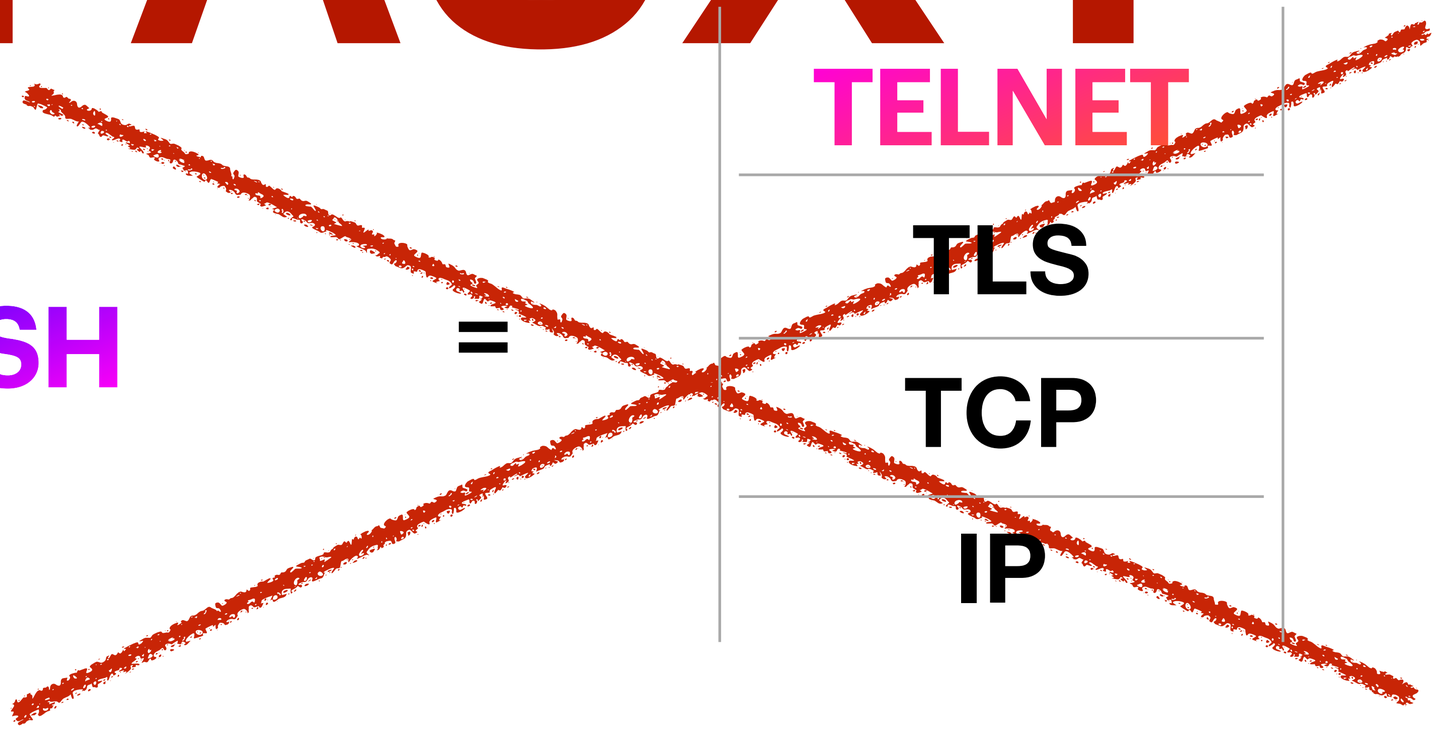
| IMAP |
| --- |
| TLS |
| TCP |
| IP |

**POPS** = 


LDAPS
=
LDAP
TLS
TCP
IP

# FAUX !

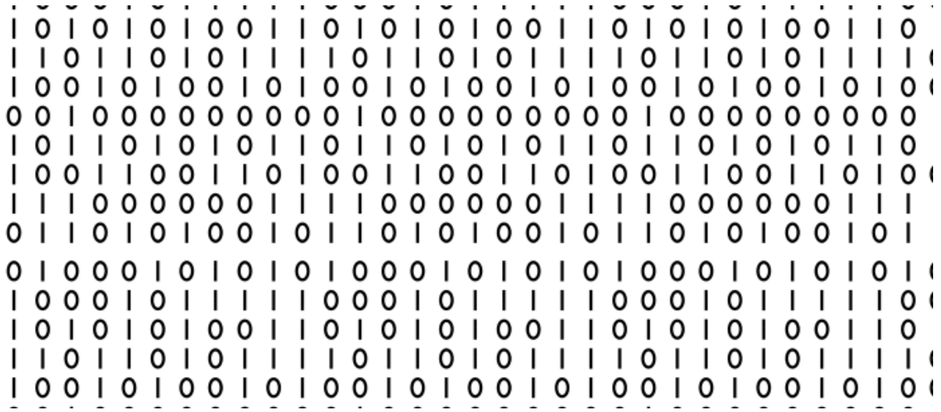
SSH = TELNET



**TLS** = {

- **Asymmetric cryptography**
- **Symmetric cryptography**
- **Cryptographic Hash functions**

# Cryptographic Hash functions



```
4c2383f5c88e9110642953b5dd7c88a1
```

**Undefined length**

**Fixed length**

# Cryptographic Hash functions

## Length

| | |
|---|---|
| MD5 | 16 |
| SHA1 | 20 |
| SHA256 | 32 |
| SHA512 | 64 |

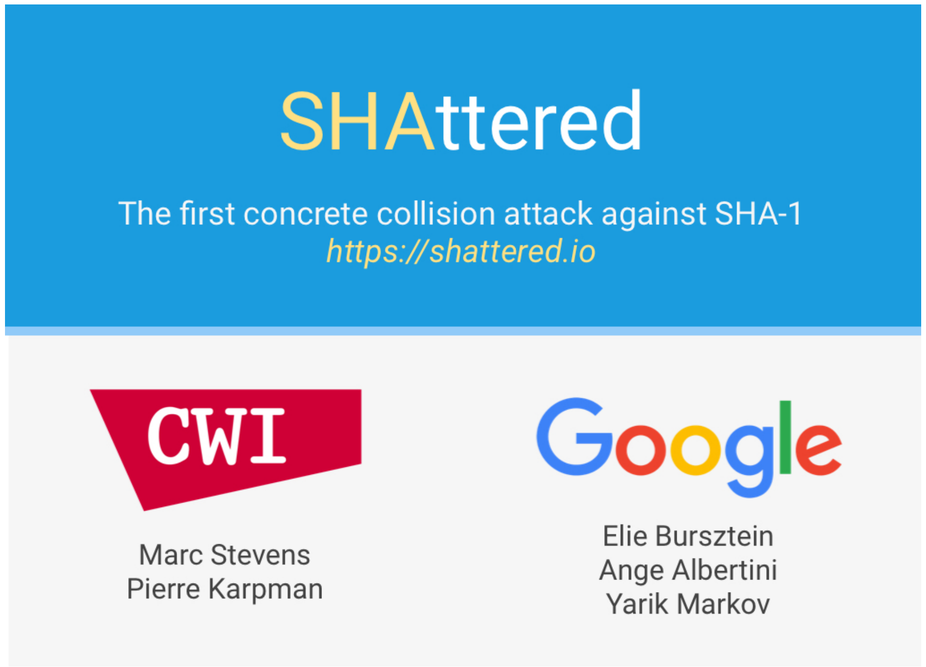
# Cryptographic Hash functions

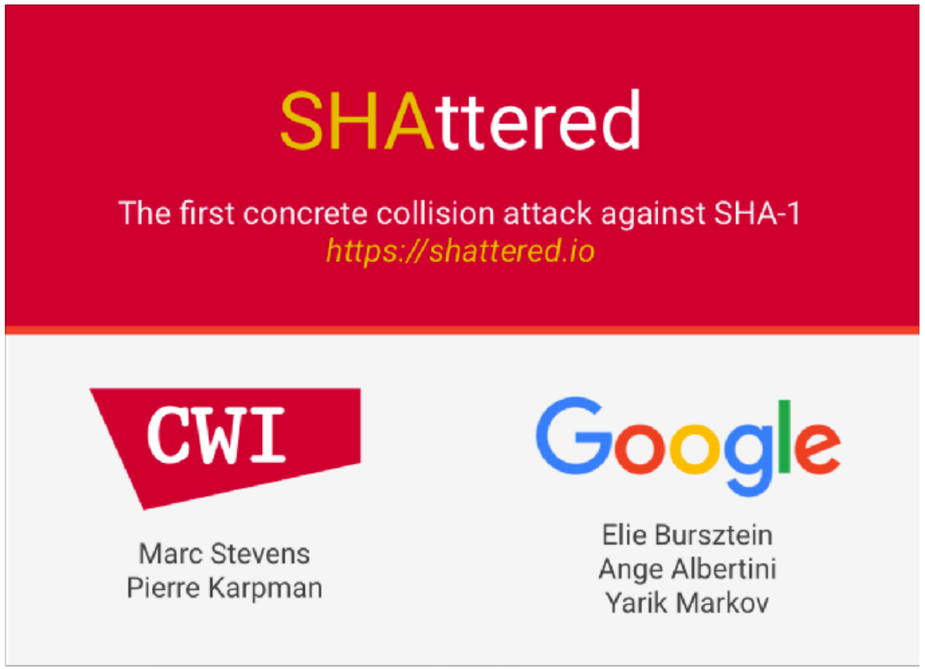
## Length

| | |
|---|---|
| ~~MD5~~ | ~~16~~ |
| ~~SHA1~~ | ~~20~~ |
| SHA256 | 32 |
| SHA512 | 64 |

Deprecated

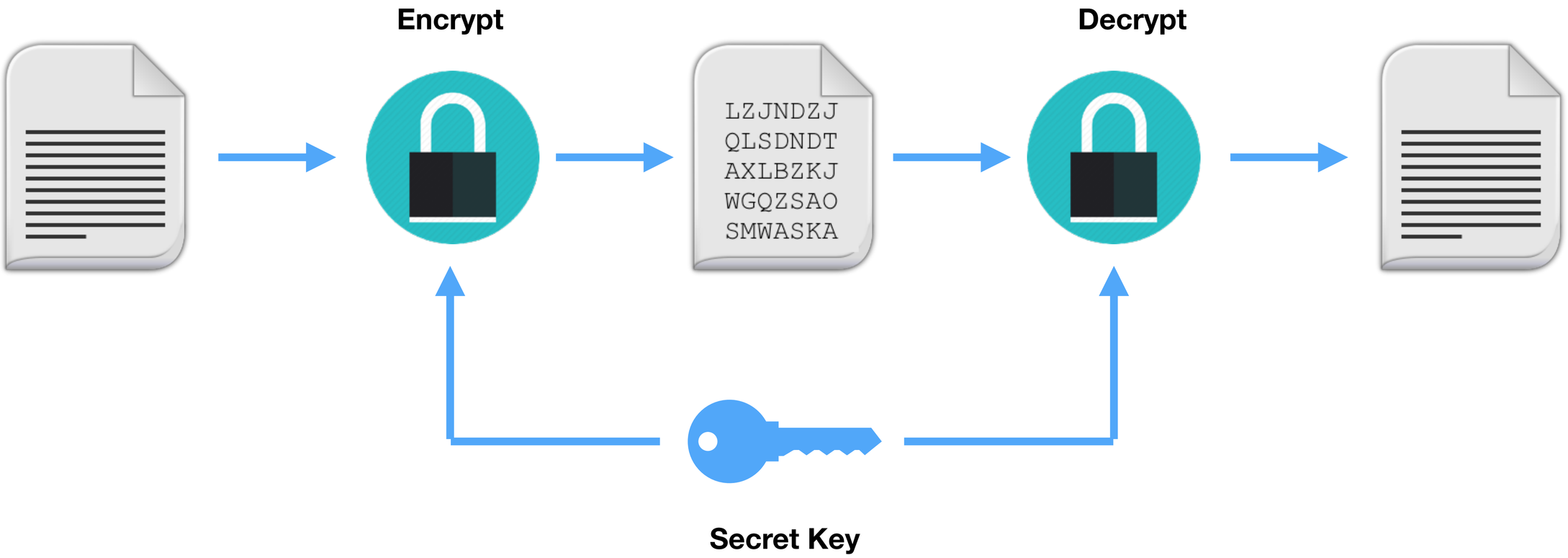
openssl sha1 shattered-1.pdf
SHA1(shattered-1.pdf)= 38762cf7f55934b34d179ae6a4c80cadccbb7f0a




openssl sha1 shattered-2.pdf
SHA1(shattered-2.pdf)= 38762cf7f55934b34d179ae6a4c80cadccbb7f0a

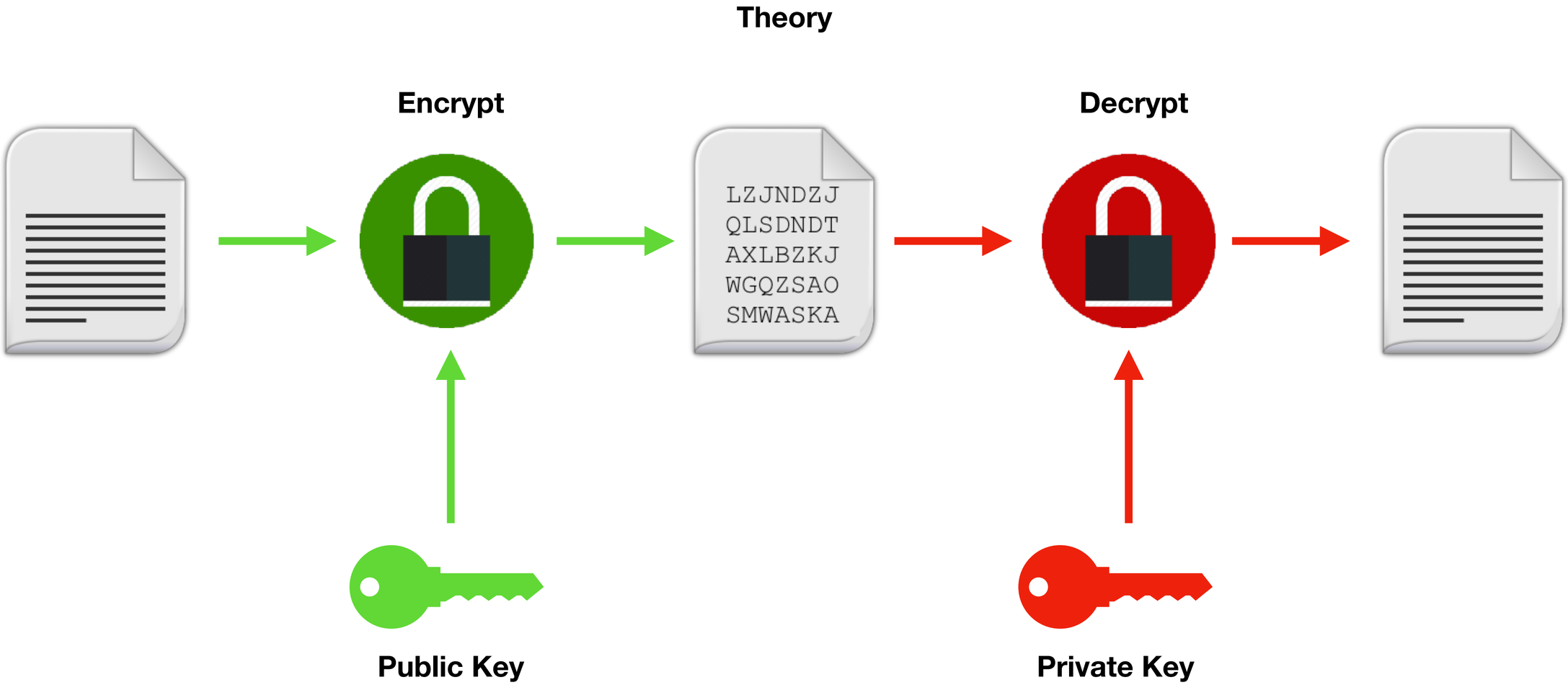
# Symmetric cryptography

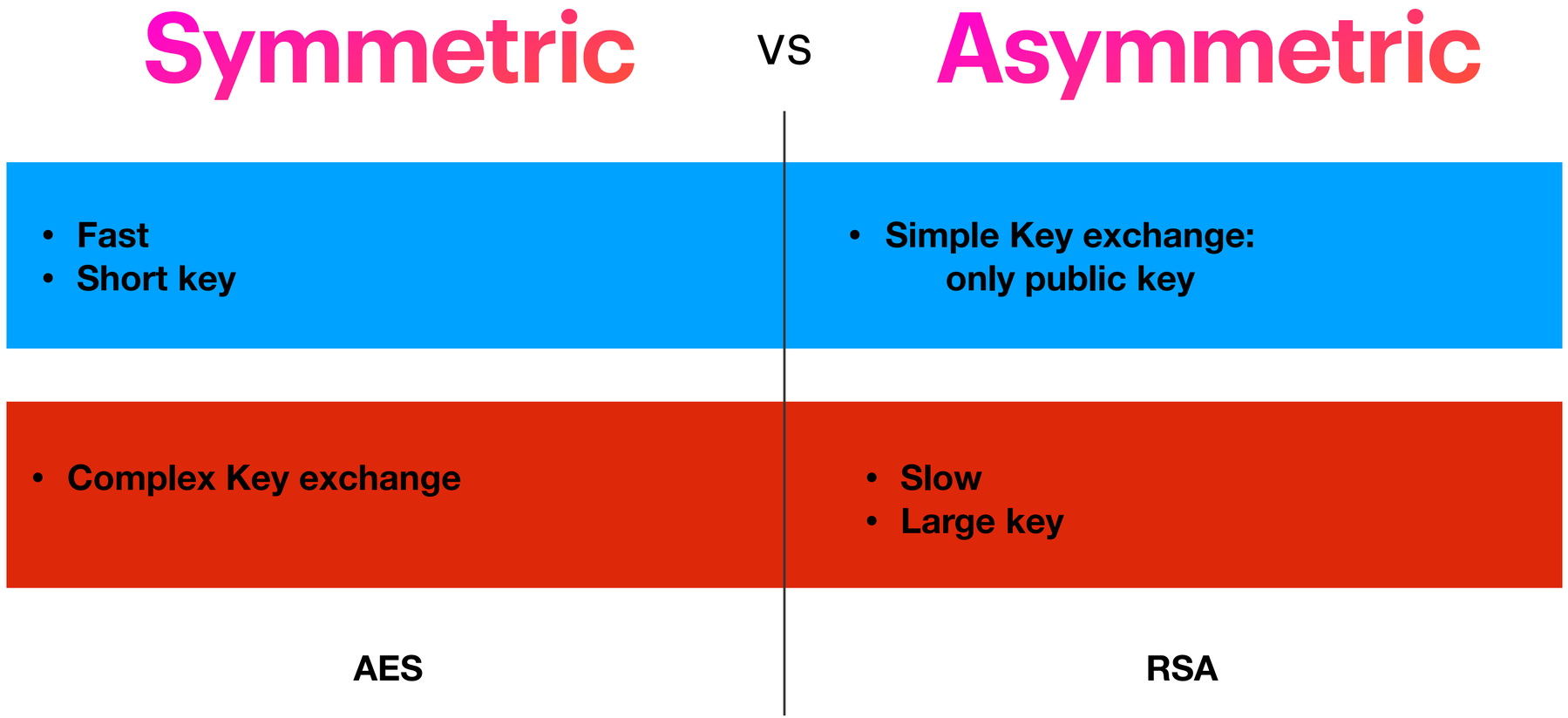


# Symmetric cryptography

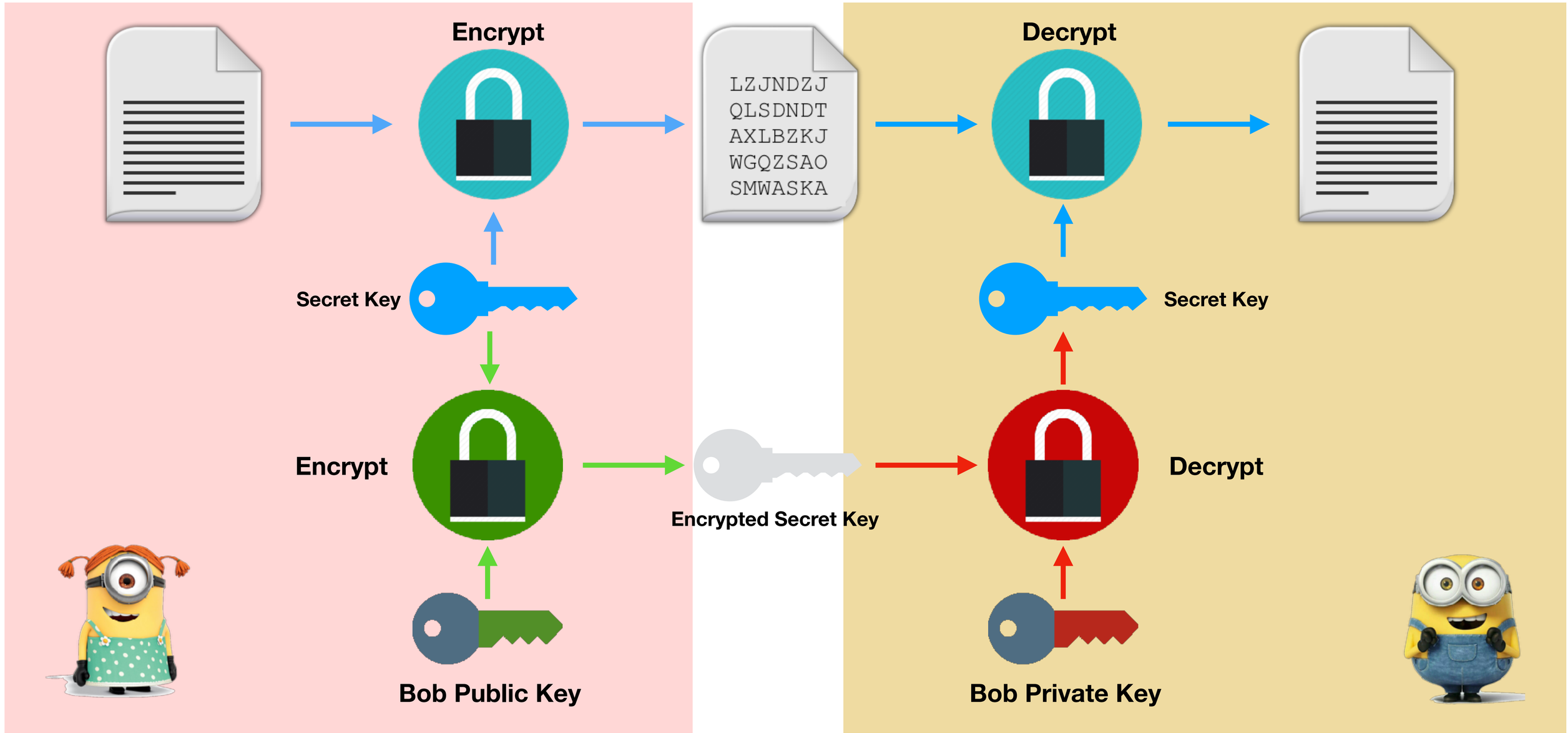
| |
|---|
| **AES** |
| 3DES |
| Twofish |
| Blowfish |
| RC4 |
| Serpent |
| Salsa20 |
| **ChaCha** |

# Asymmetric encryption

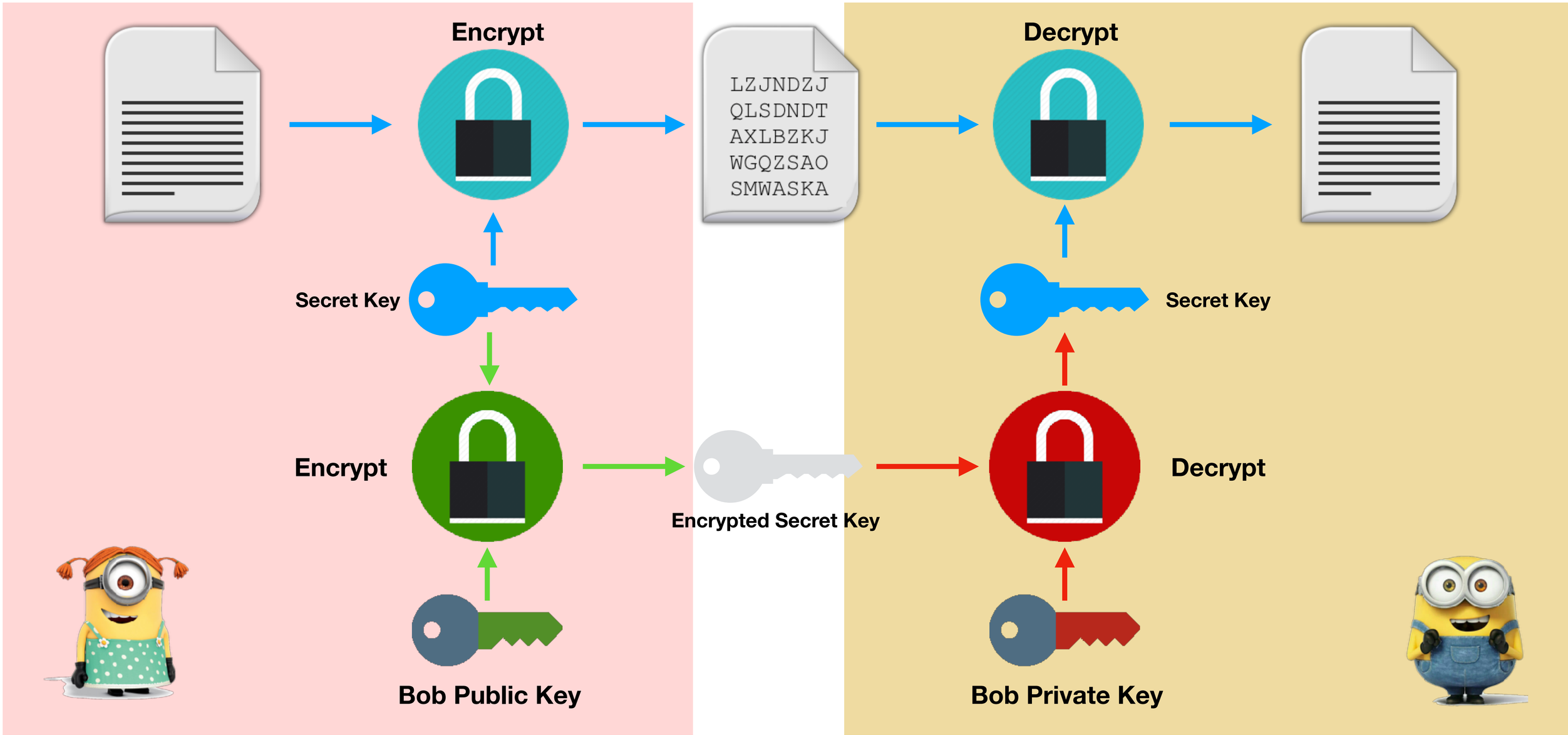



Symmetric
VS
Asymmetric
• Fast
• Short key
• Simple Key exchange:
only public key
• Complex Key exchange
• Slow
• Large key
AES
RSA

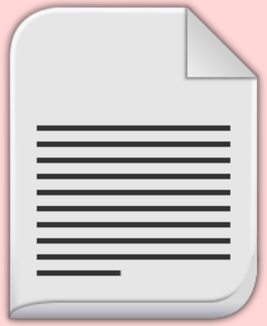
# Asymmetric encryption on data



# Asymmetric encryption on data



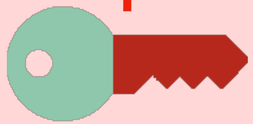
# Digital signature



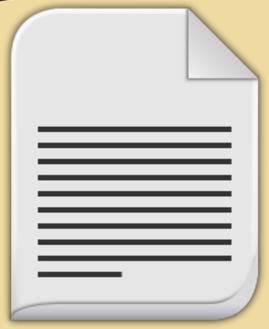
4c2383f5c88e9110
642953b5dd7c88a1

Encrypt

6d9c252e4359ba82
e889b06ce62eb1af



Alice Private Key



4c2383f5c88e9110
642953b5dd7c88a1

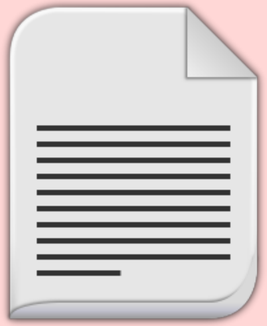
= ?

4c2383f5c88e9110
642953b5dd7c88a1

Decrypt

Alice Public Key

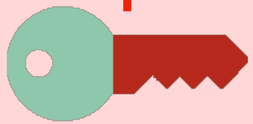
# Digital signature



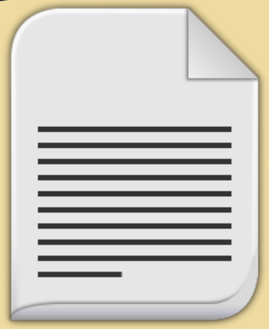
4c2383f5c88e9110
642953b5dd7c88a1

Encrypt

6d9c252e4359ba82
e889b06ce62eb1af



Alice Private Key



4c2383f5c88e9110
642953b5dd7c88a1

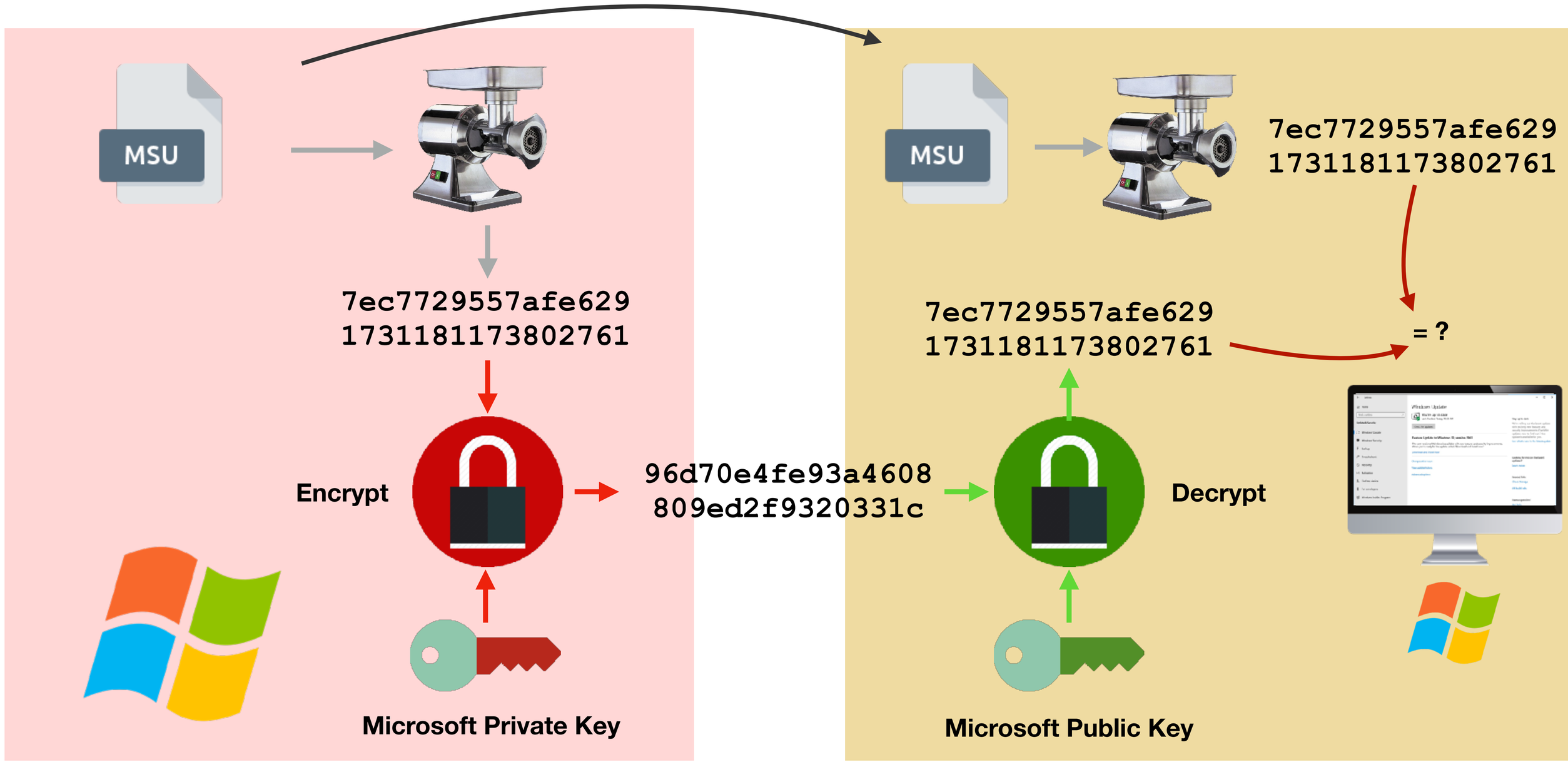
= ?

4c2383f5c88e9110
642953b5dd7c88a1

Decrypt

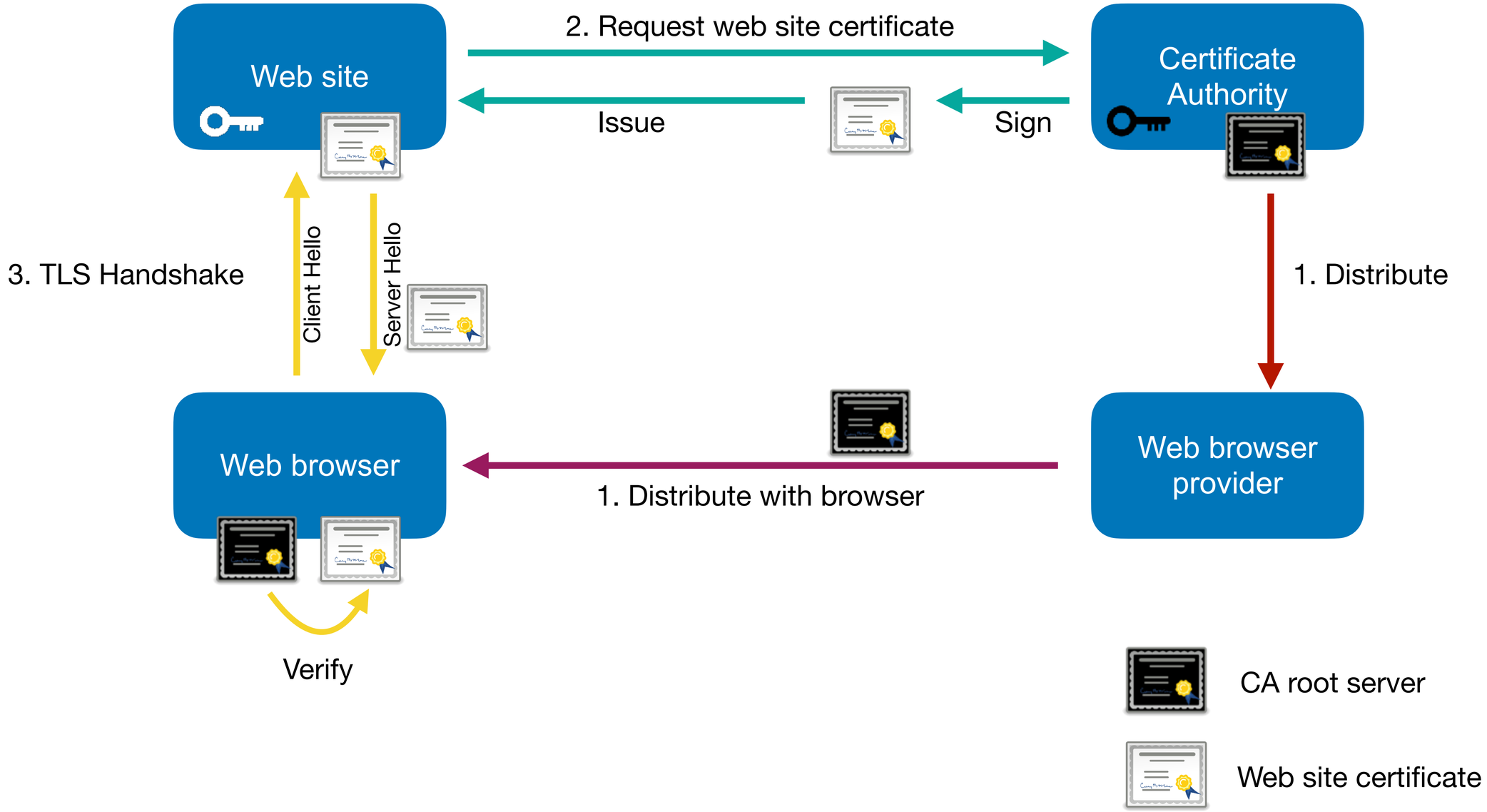
Alice Public Key

# Digital signature update example



# X509 Certificate

- Serial Number
- Issuer Name
- Validity Period
- Subject Name
- Subject Public Key
- Extensions
- Certificate Signature


Web site
2. Request web site certificate
Certificate Authority
Issue
Sign
3. TLS Handshake
Client Hello
Server Hello
1. Distribute
Web browser
1. Distribute with browser
Web browser provider
Verify
CA root server
Web site certificate

# X509 Certificate

```
echo -n | openssl s_client
-connect www.twitter.com:443 |
sed -ne '/-BEGIN
CERTIFICATE-/,/-END
CERTIFICATE-/p' | openssl x509
-noout -text
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            6d:67:4a:06:8c:68:2a:c8:ef:2b:f0:c8:36:54:61:06
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Symantec Corporation, OU=Symantec Trust Network, CN=Symantec Class 3 Secure Server
CA - G4
        Validity
            Not Before: Aug 10 00:00:00 2015 GMT
            Not After : Aug 10 23:59:59 2017 GMT
        Subject: C=US, ST=California, L=San Francisco, O=Twitter, Inc., OU=FRA2 Point of Presence,
CN=twitter.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:c9:b3:1e:d0:74:6b:f2:f2:a2:6d:0f:a0:82:64:
                    f3:40:24:2d:22:dd:90:d1:d2:3c:94:2e:4d:a8:cc:
                    42:1d:e7:20:37:52:56:6e:4c:92:c0:d3:22:5f:fa:
                    16:91:1f:be:68:ae:55:8d:d3:67:42:3e:f5:42:bb:
                    69:b0:77:d7:14:51:69:85:2f:1b:cb:ea:ee:a9:2b:
                    83:b0:60:57:80:8f:d4:c9:4f:82:39:09:04:31:eb:
                    f5:ed:8a:40:85:ae:42:38:c0:67:19:e1:25:3a:d6:
                    bc:48:ae:64:62:8b:6a:52:69:35:71:8b:7d:3e:f6:
                    e4:b9:b1:47:cc:95:b6:01:3f:22:20:7b:98:a4:bc:
                    bd:d7:02:49:2a:35:5c:e1:f4:84:56:37:f6:c9:70:
                    03:c6:2b:34:ad:1c:83:7d:08:03:d2:d2:20:16:06:
                    f8:eb:4d:97:93:46:7b:53:ba:c7:64:63:4f:4b:22:
                    ab:8d:2a:6f:a6:e9:37:be:74:65:e5:ed:38:1e:fb:
                    ad:62:09:ab:8d:ec:40:34:08:f8:d8:76:a6:dc:eb:
                    44:77:d9:10:56:16:60:86:e0:df:2f:e4:7a:c8:ce:
                    24:70:8e:0f:c7:68:1c:ae:a7:25:e5:d8:f6:77:e6:
                    17:8c:4a:74:8e:24:f1:37:4e:92:29:8b:4a:b7:c9:
                    49:7d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:twitter.com, DNS:www.twitter.com
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Certificate Policies:
                Policy: 2.23.140.1.2.2
                  CPS: https://d.symcb.com/cps
                  User Notice:
                    Explicit Text: https://d.symcb.com/rpa

            X509v3 Authority Key Identifier:
                keyid:5F:60:CF:61:90:55:DF:84:43:14:8A:60:2A:B2:F5:7A:F4:43:18:EF

            X509v3 CRL Distribution Points:
                URI:http://ss.symcb.com/ss.crl

            Authority Information Access:
                OCSP - URI:http://ss.symcd.com
                CA Issuers - URI:http://ss.symcb.com/ss.crt

    Signature Algorithm: sha256WithRSAEncryption
        7c:f8:02:62:7c:4b:ab:77:74:99:1c:8a:a1:9c:7f:bd:e0:a2:
        48:28:44:8a:05:ce:37:18:8e:9c:dc:d7:df:2c:43:6a:35:94:
        be:52:4c:3f:1b:6c:c0:78:28:06:ef:92:e7:ae:2a:c0:cf:73:
        69:c2:bb:b6:63:93:51:1a:33:a2:fb:85:0a:26:21:08:cc:53:
        82:d3:ac:99:7a:ba:f8:b7:5c:c8:15:30:29:00:71:ab:8e:52:
        0a:fa:cd:a2:77:d0:74:87:25:47:c2:21:ef:02:24:b6:c7:74:
        45:ee:c0:82:d4:ef:37:ed:17:40:27:e8:8d:15:51:4c:6f:c4:
        fb:8d:58:14:fb:13:89:a8:4d:1d:9a:9e:0c:57:68:ed:f0:ca:
        40:18:23:f6:ef:24:b1:4e:67:97:18:db:52:57:1e:3f:61:39:
        b5:0f:6b:b5:46:52:3b:e9:24:4d:f3:de:16:71:a8:cb:33:71:
        53:db:5f:05:4f:05:c8:3e:1c:23:14:1a:88:5f:8c:5a:3f:8e:
        e1:c3:47:a9:9f:90:72:2c:a1:33:3c:74:41:b7:0d:7e:cf:cc:
        ac:02:19:82:ba:f6:1a:20:2e:a9:14:52:f4:1a:16:a4:78:a3:
        ee:e9:a3:ed:49:a2:ff:1b:8e:6e:62:11:85:29:cd:b3:b2:d4:
        99:d6:97:6e
```

# Certificate Authorities are the root of trust of TLS

**So the root of trust failure…**

COMODO
CERTIFICATE AUTHORITY

DigiNotar
Internet Trust Services

trustico®

Symantec™

# How to fix it ?

Certificate pinning
HTTP Public Key Pinning
Certificate transparency
DNS Certification Authority Authorization

# TLS Handshake (RSA)

C Client random

S Server random
Public key certificate

P pre-master secret

Client Hello: client random + supported cipher suites

Server Hello: server random + cipher suite + server certificate

Encrypt pre-master secret with certificat public key

Client random C

Server random
Public key certificate S

Encrypted pre-master secret P

Private Key

Decrypt

pre-master secret P

session key

Both the client and the server generate the TLS session key from the client random, the server random and the pre-master secret

session key

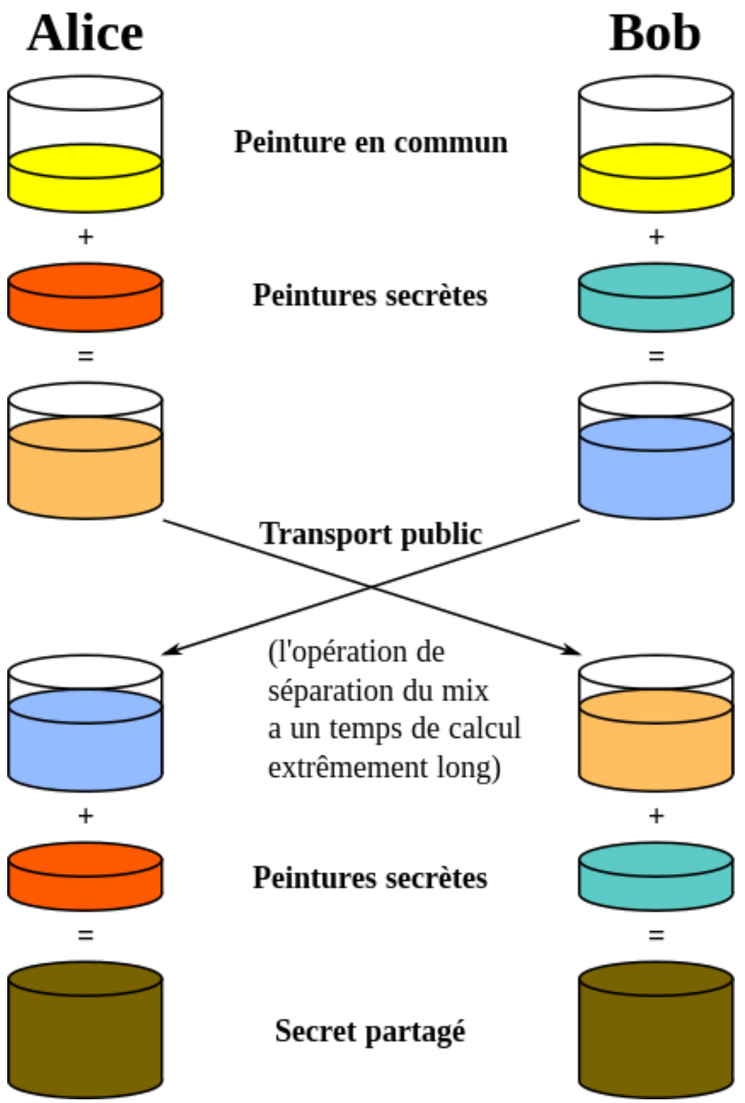
| Inbound IV | Outbound IV | Inbound Cipher key | Outbound Cipher key | Inbound MAC key | Outbound MAC key |
|---|---|---|---|---|---|

| Inbound IV | Outbound IV | Inbound Cipher key | Outbound Cipher key | Inbound MAC key | Outbound MAC key |
|---|---|---|---|---|---|

# « What is the problem ? »

No Perfect Backward Secrecy
No Perfect Forward Secrecy

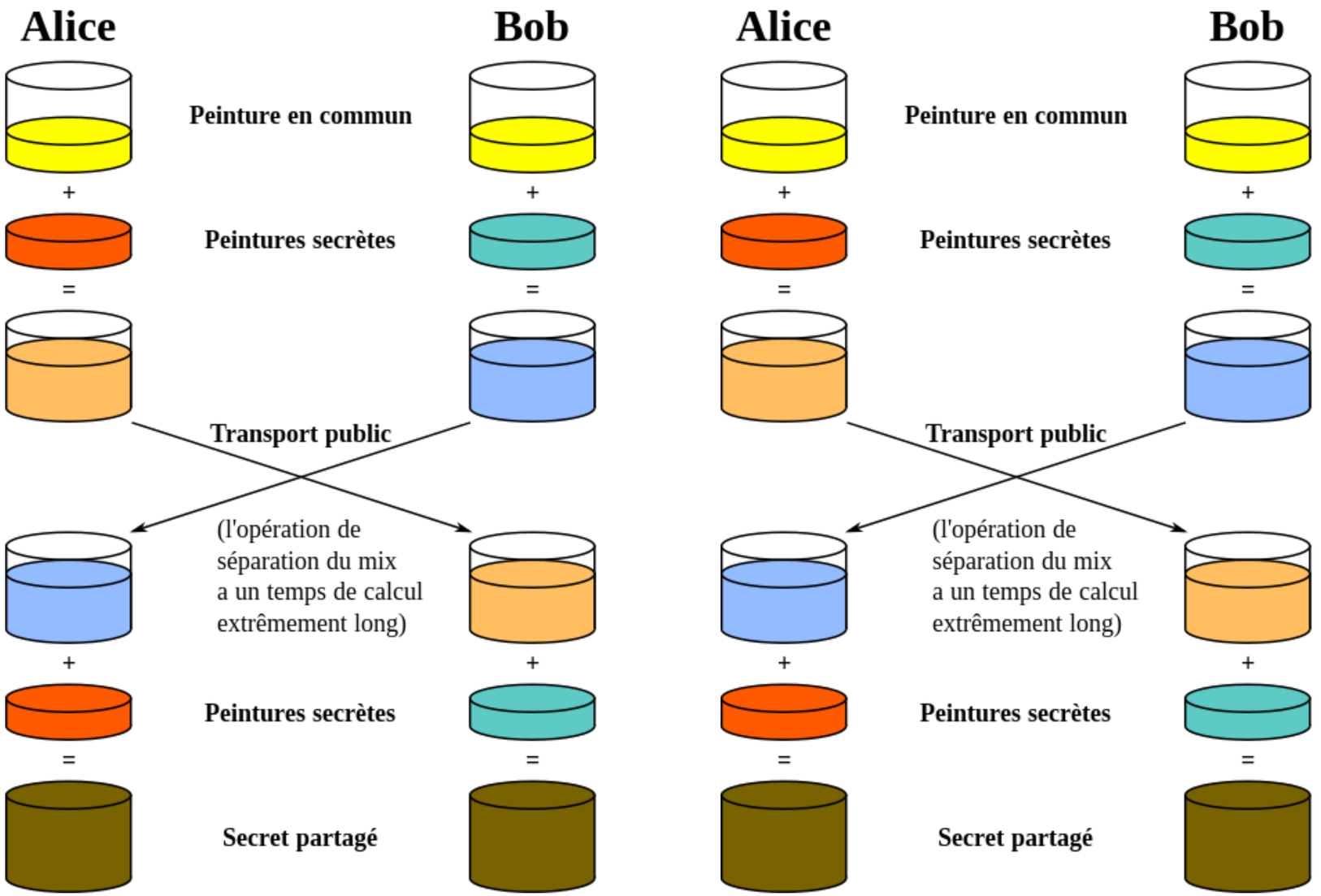
# Diffie-Hellman





https://www.youtube.com/watch?v=1Yv8m398Fv0&pp=ygUfc2NpZW5jZSBldG9ubmFudGUgZGlmZmlllGhlbG1hbg%3D%3D

# Diffie-Hellman



# TLS Handshake (RSA-Diffie Hellman)

C Client random

Client Hello: client random + supported cipher suites

Client random C

S Server random
Public key certificate

Server Hello: server random + cipher suite + server certificate

Server random
Public key certificate S

DH Server DH parameter

Server DH parameter DH

Signature from Key Server

Server DH parameter + signature

Sign Private Key

DH Client DH parameter

Client DH parameter

Client DH parameter DH

P pre-master secret

Both the client and the server derives identical pre-master secret from client DH and server DH parameter

pre-master secret P

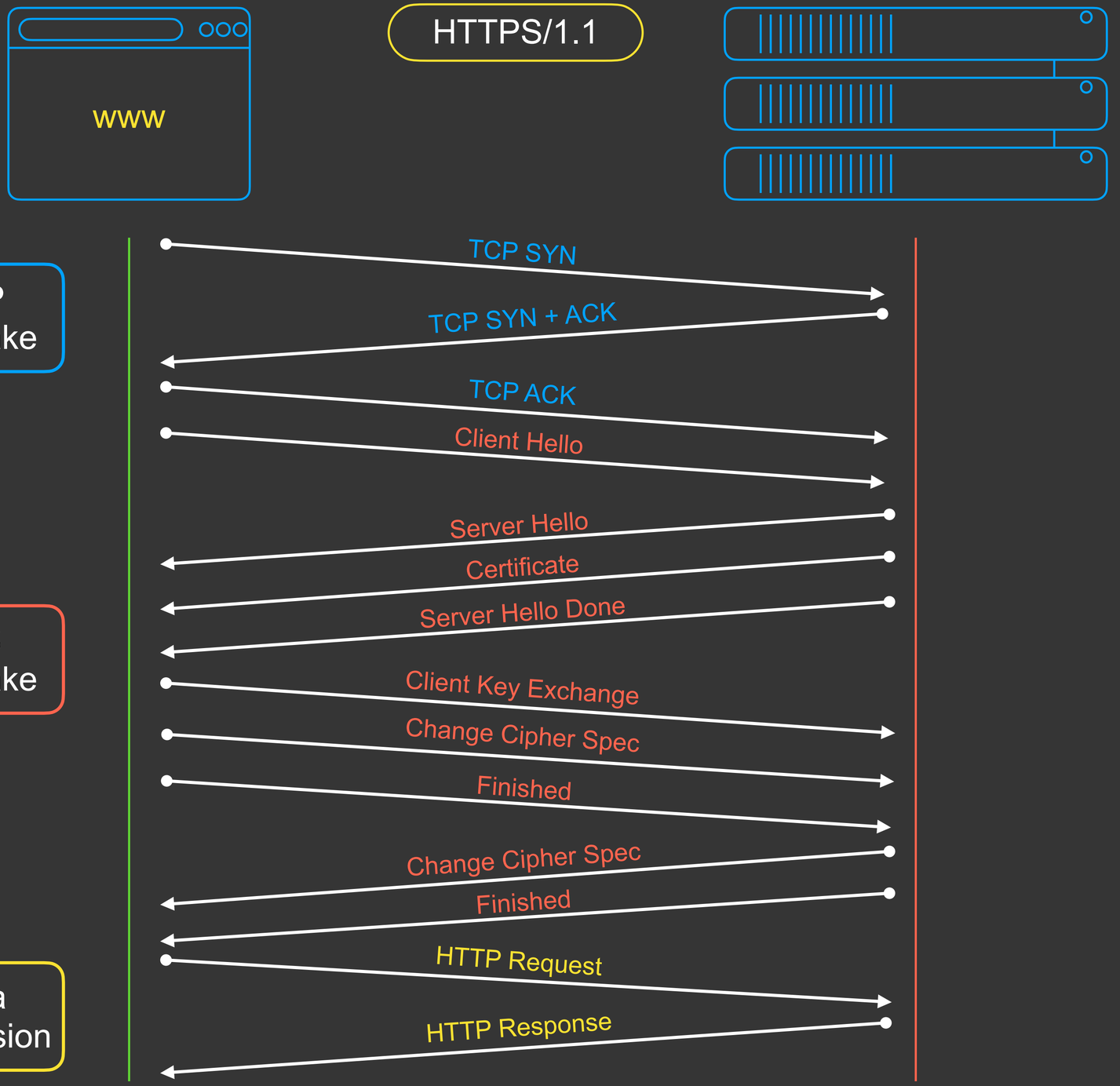
session key

Both the client and the server derives identical TLS session key from the client random, the server random and the pre-master secret

session key

IV IV Cipher key Cipher key MAC key MAC key

IV IV Cipher key Cipher key MAC key MAC key

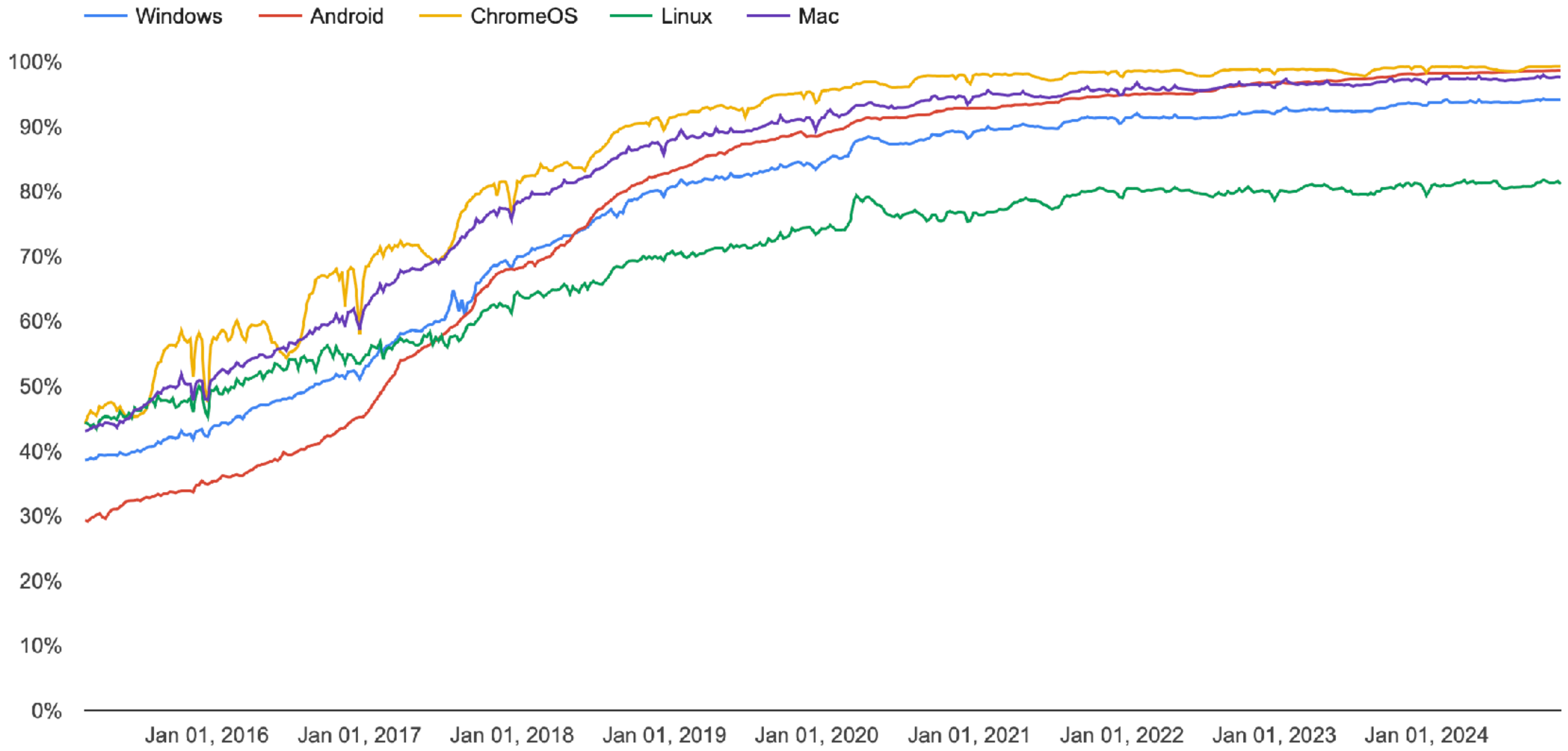

www
HTTPS/1.1
1. TCP Handshake
2. TLS Handshake
3. Data Transmission
TCP SYN
TCP SYN + ACK
TCP ACK
Client Hello
Server Hello
Certificate
Server Hello Done
Client Key Exchange
Change Cipher Spec
Finished
Change Cipher Spec
Finished
HTTP Request
HTTP Response

# TLS Hardening

- TLS protocol support : TLS 1.2
- TLS Cipher Suite authorized by the server

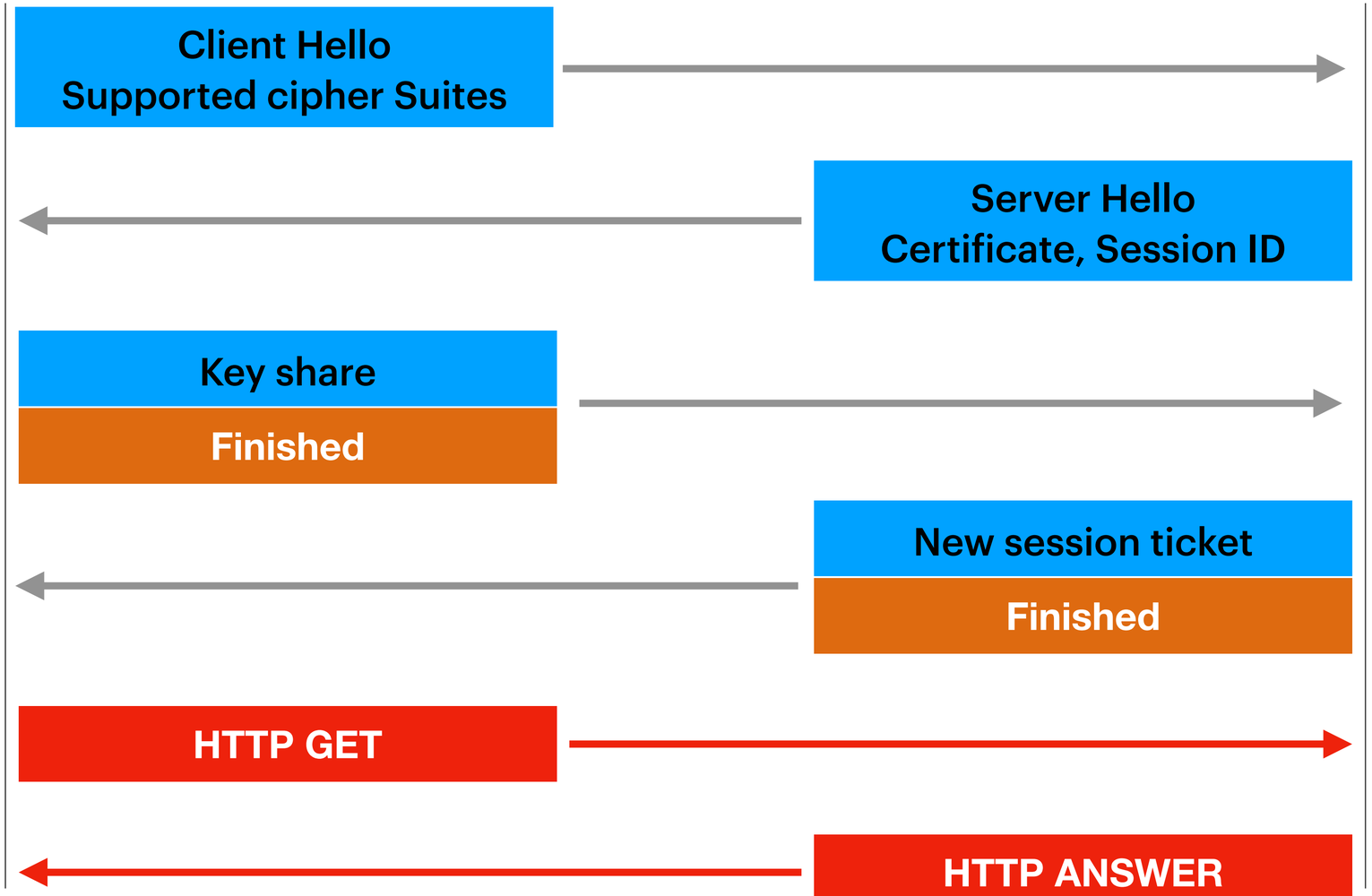
```
openssl ciphers -v 'EDH+aRSA+AESGCM:EDH+aRSA+AES:EECDH+aRSA+AESGCM:EECDH+aRSA+AES:-
SHA:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-
SHA:RSA+AESGCM:RSA+AES+SHA256:RSA+AES+SHA:DES-CBC3-SHA:DHE-RSA-AES256-SHA:DHE-RSA-
AES128-SHA:!aNULL:!eNULL:!LOW:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!ECDSA:!ADH:!IDEA'
```

# Percentage of pages loaded over HTTPS in Chrome by platform

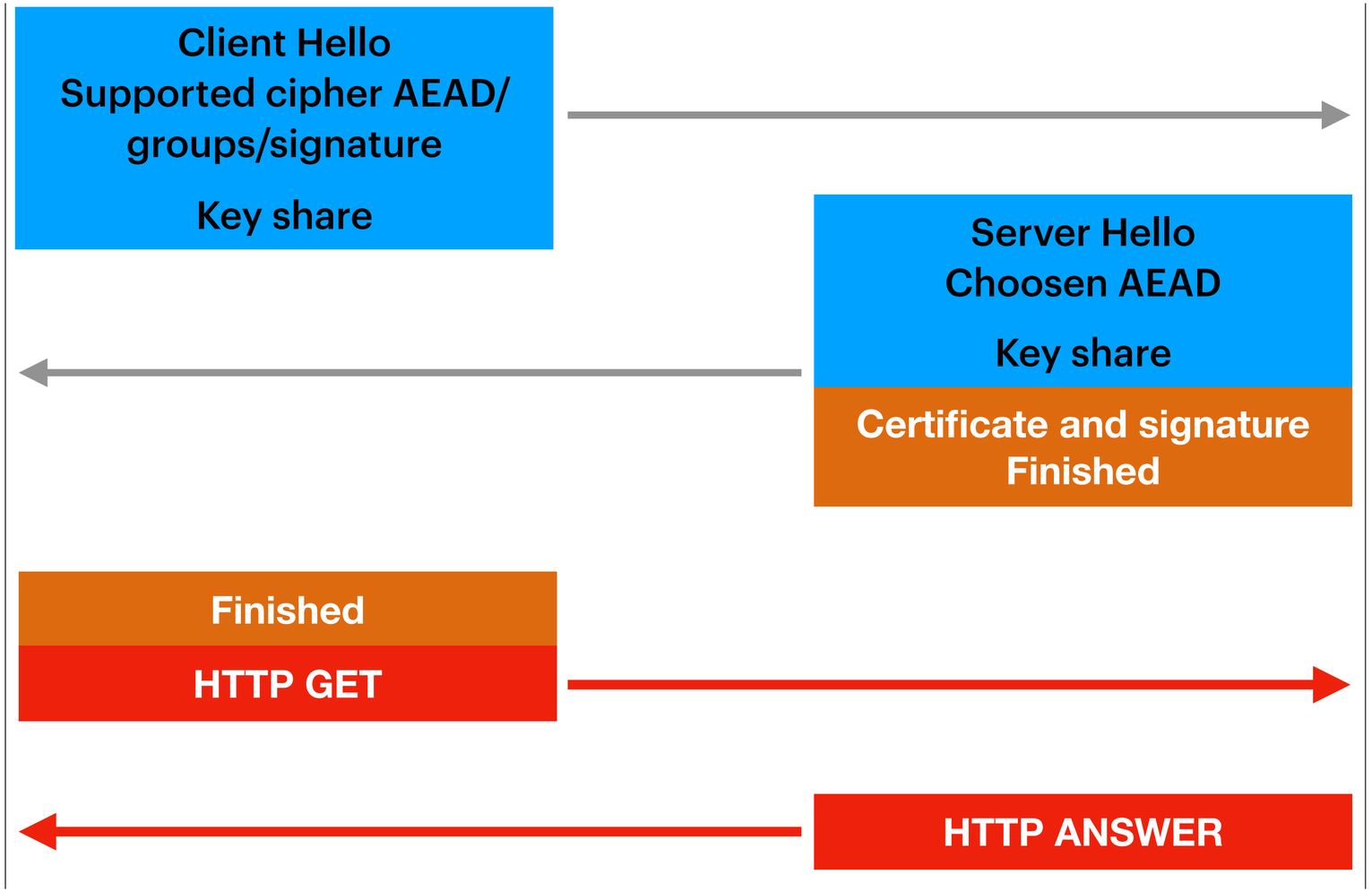


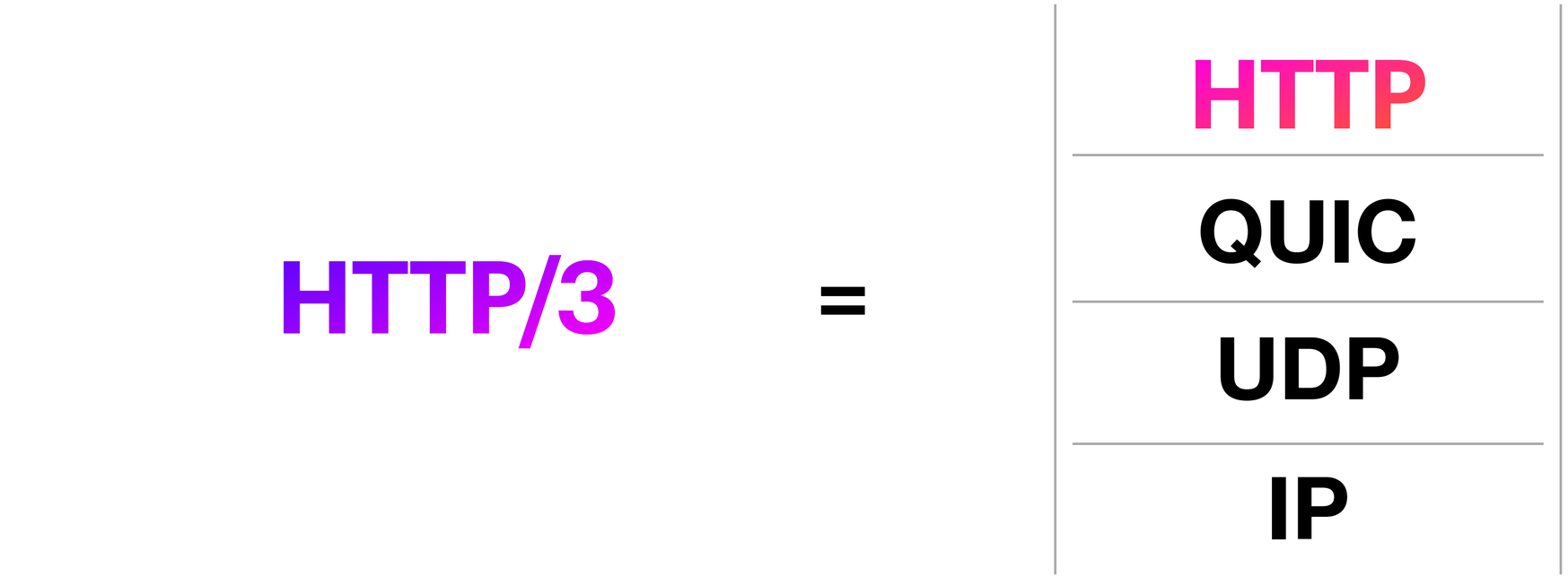
# TLS 1.3 is a big jump

# TLS 1.2 (ECDHE)

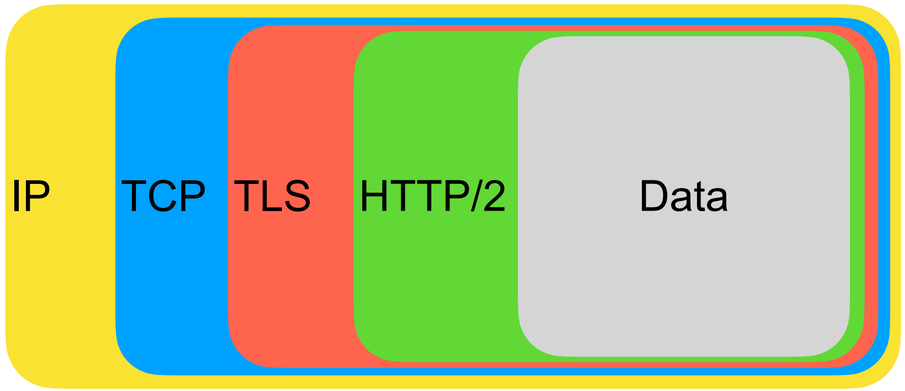

Client Hello
Supported cipher Suites
Server Hello
Certificate, Session ID
Key share
Finished
New session ticket
Finished
HTTP GET
HTTP ANSWER

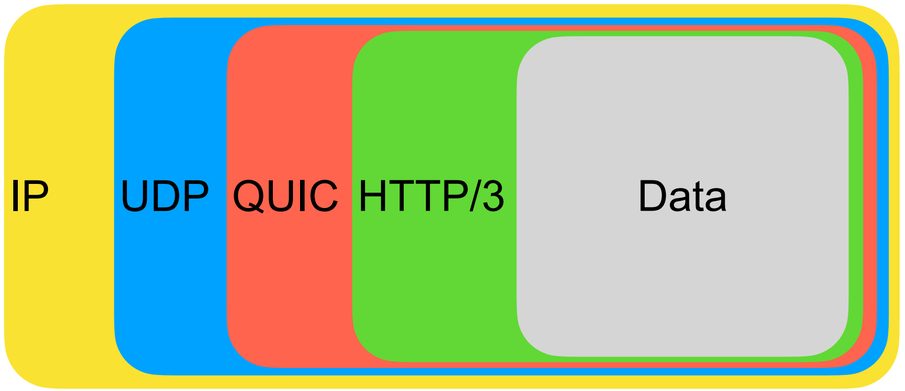
# TLS 1.3


Client Hello
Supported cipher AEAD/ groups/signature
Key share
Server Hello
Choosen AEAD
Key share
Certificate and signature
Finished
Finished
HTTP GET
HTTP ANSWER


HTTP/3
=
HTTP
QUIC
UDP
IP

# HTTP Protocol stacks



## Secure Element


## Hardware Security Module


# TLS Hardening

https://cryptosense.com

https://www.ssllabs.com/ssltest/

https://bettercrypto.org/static/applied-crypto-hardening.pdf

https://testssl.sh

https://www.owasp.org/index.php/Testing_for_SSL-TLS_(OWASP-CM-001)