HTTP in a nutshell

INTEGRANTES:
- GALO FERNÁNDEZ ACHILLE
- SANTINO NICOLÁS ANDREATTA
- PEDRO VILLARINO

¿Que es?

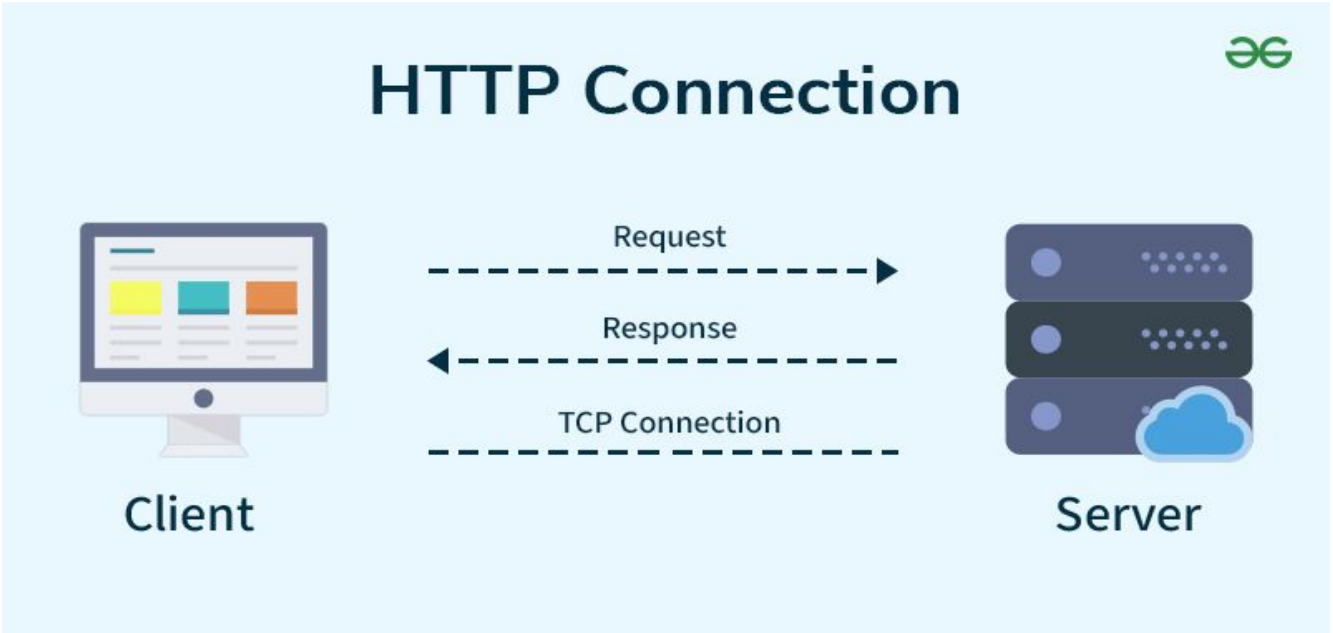
Hypertext Transfer Protocol

Protocolo de comunicación entre un **cliente** y un **servidor web**

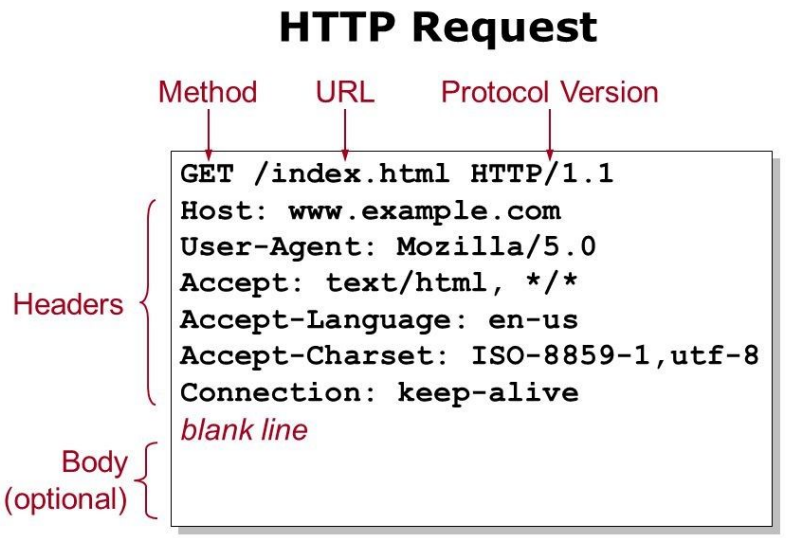
Estándar de comunicación para transferencia de datos

La base de la **World Wide Web**

ESQUEMA

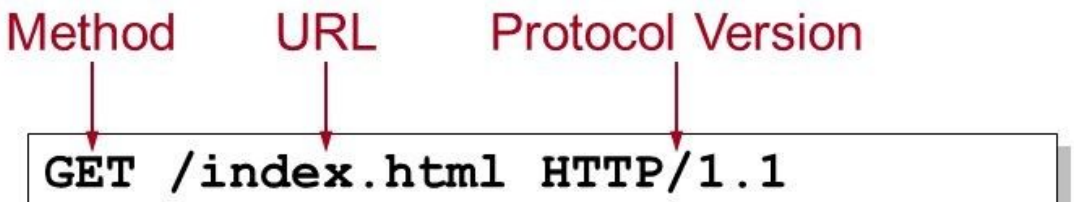

HTTP Connection
Request
Response
TCP Connection
Client
Server

REQUEST


HTTP Request
Method
URL
Protocol Version
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: ISO-8859-1,utf-8
Connection: keep-alive
blank line
Headers
Body
(optional)

REQUEST

- **Método**: Indica cómo se quiere acceder al recurso
- **URL**: Es el “directorio” del recurso a solicitar



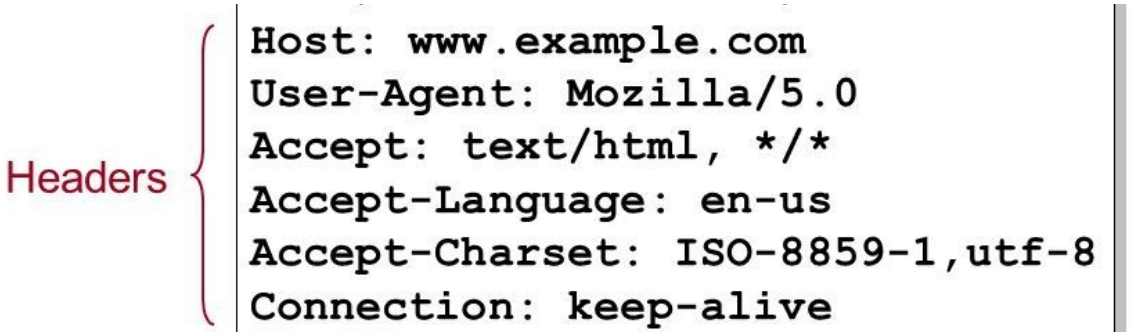
MÉTODOS

- GET
- POST
- PUT
- PATCH
- DELETE
- HEAD
- TRACE
- OPTIONS
- CONNECT

HEADERS

Sirve a modo de **registro** de la información de la comunicación.

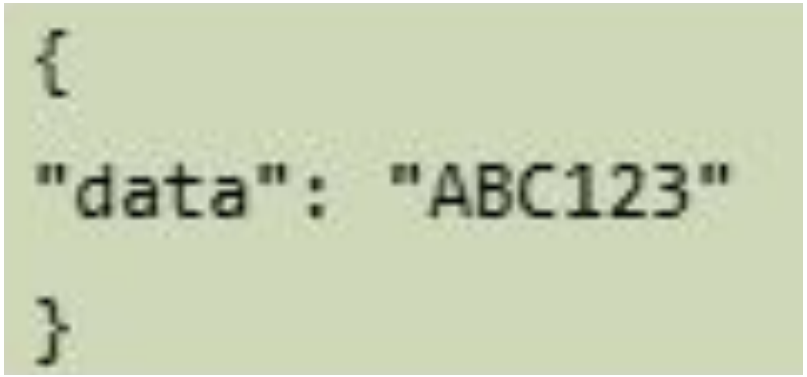
Transferencia de **metadatos**.



BODY

Transporta el **payload** de una request o respuesta.

Información para indicar cómo se accede a un recurso, o recurso que devuelve el servidor al cliente.




```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

RESPONSE

Diferencia request y response

- Código de estado

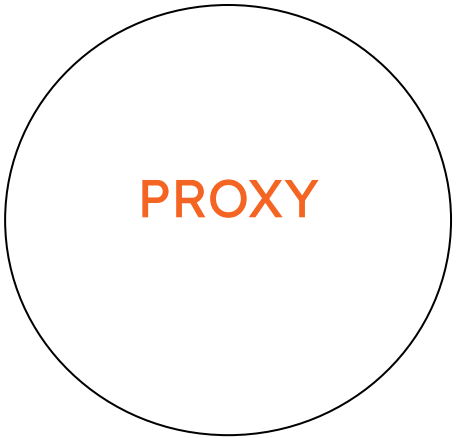
```
HTTP/1.1 403 Forbidden
```

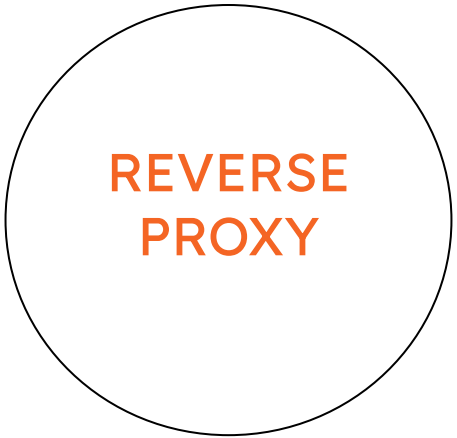
Response

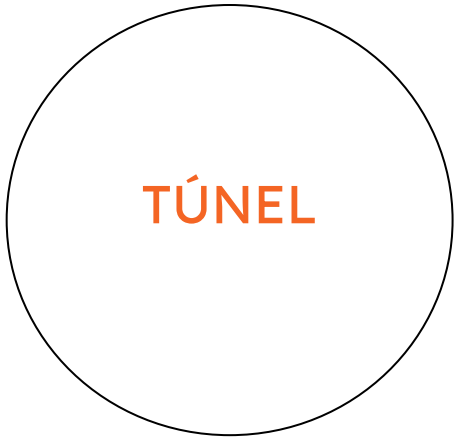
```
HTTP/1.1 403 Forbidden
Server: Apache
Date: Fri, 21 Jun 2024 12:52:39 GMT
Content-Length: 678
Content-Type: text/html
Cache-Control: no-store

<!DOCTYPE html>
<html lang="en">
(more data…)
```

INTERMEDIARIOS

HTTP permite que haya conexiones intermedias en una request. Se pueden generar “cadenas” de intermediarios.



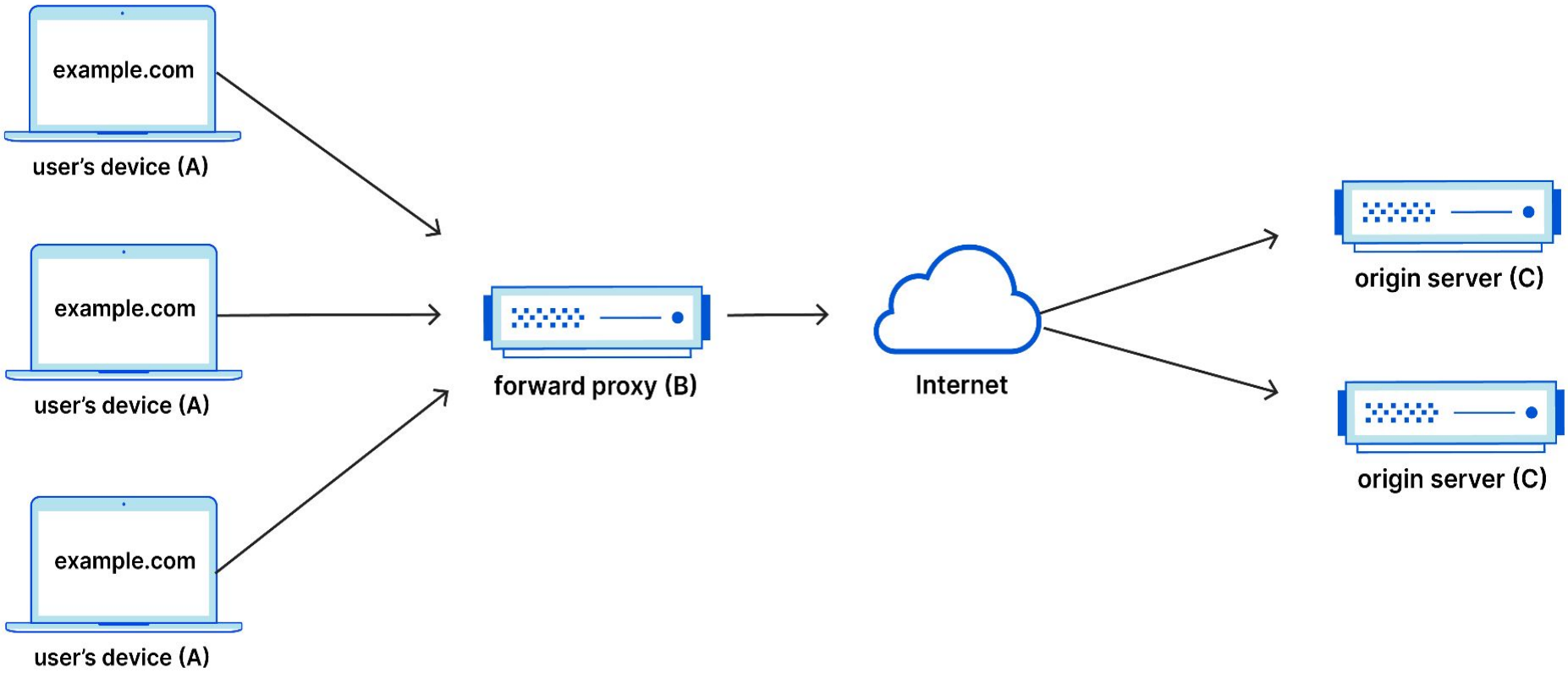




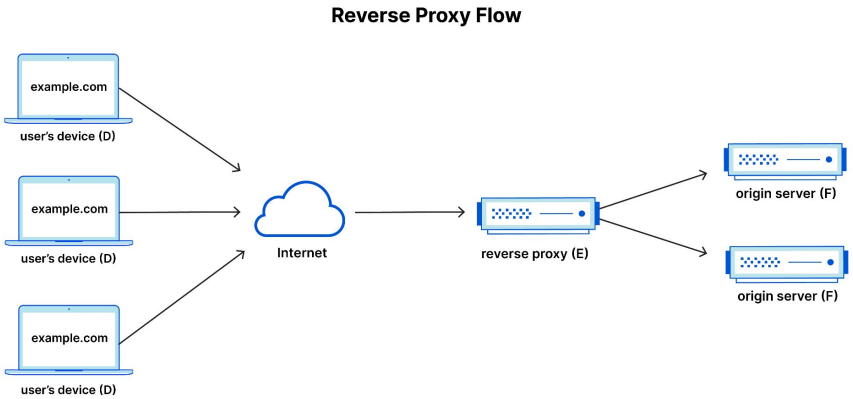
PROXY

- **Puente entre un cliente e internet.**
- **Permite:**
  - Restricciones y filtros
  - Cachear contenido
- **Casos de uso comunes:**
  - Redes públicas
  - Empresas

Forward Proxy Flow



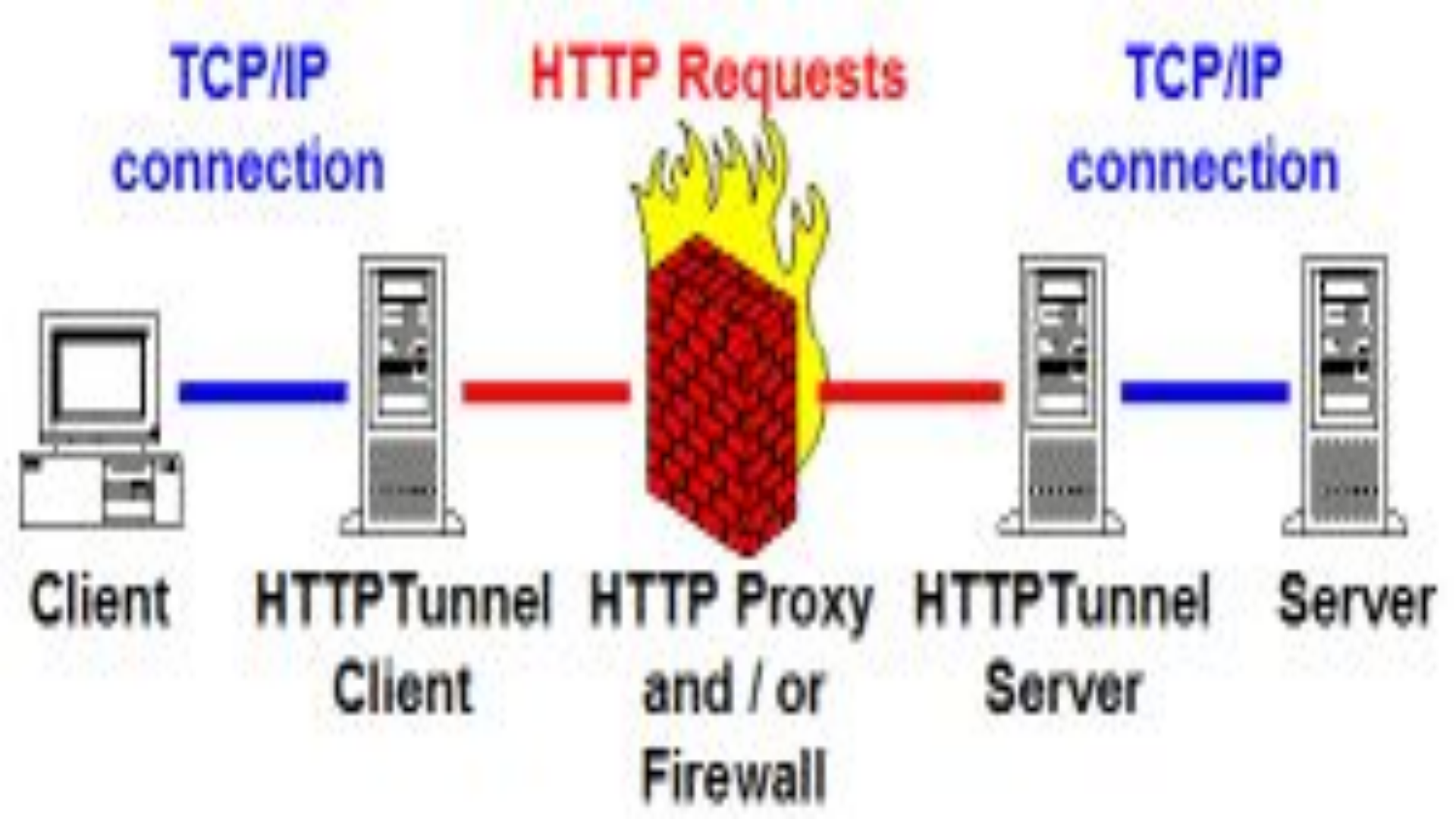
Reverse proxy



- Puente entre internet y un servidor
- **Permite:**
  - Filtrar requests de los clientes
  - Redirigir el tráfico a distintos servidores
  - Load balancer
  - Cachear respuestas

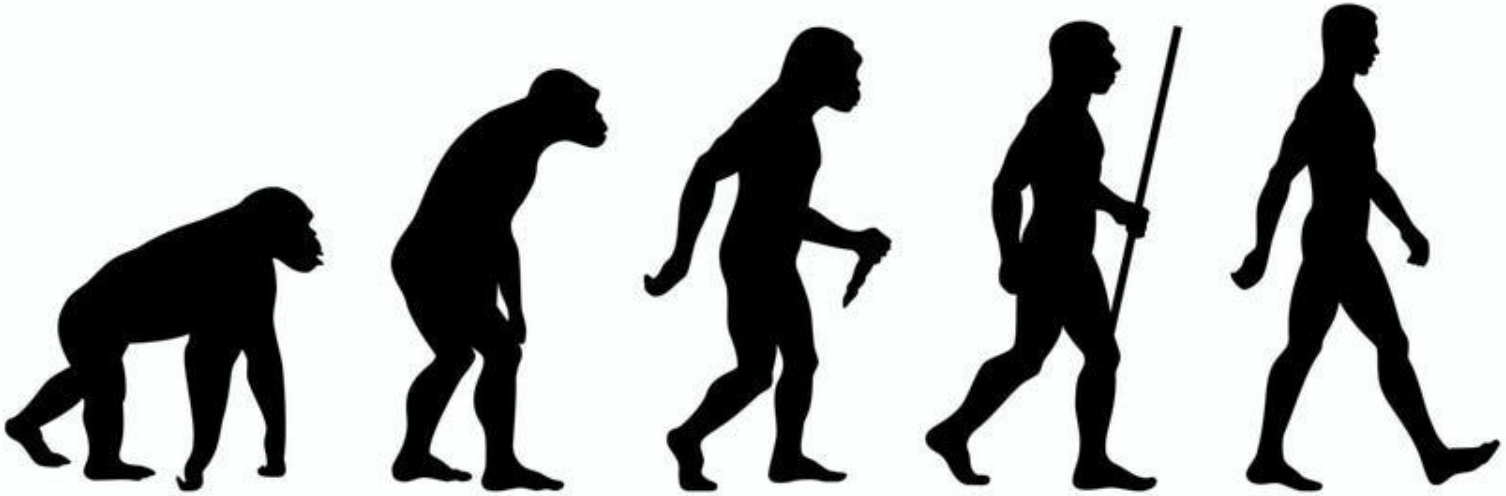
TÚNEL

- **Un intermediario que SOLO PASA DATOS.**
  - No los modifica
  - No los lee
- **Permite conectarse por HTTPS a un proxy**
- **Cifrado extremo a extremo**
  - Nadie puede ver lo que se transmite, sólo el remitente y receptor
  - Previene man in the middle
  - El servidor no puede ver el contenido de los mensajes (similar a whatsapp)


TCP/IP
connection
HTTP Requests
TCP/IP
connection
Client
HTTPTunnel
Client
HTTP Proxy
and / or
Firewall
HTTPTunnel
Server
Server

Evolution of HTTP

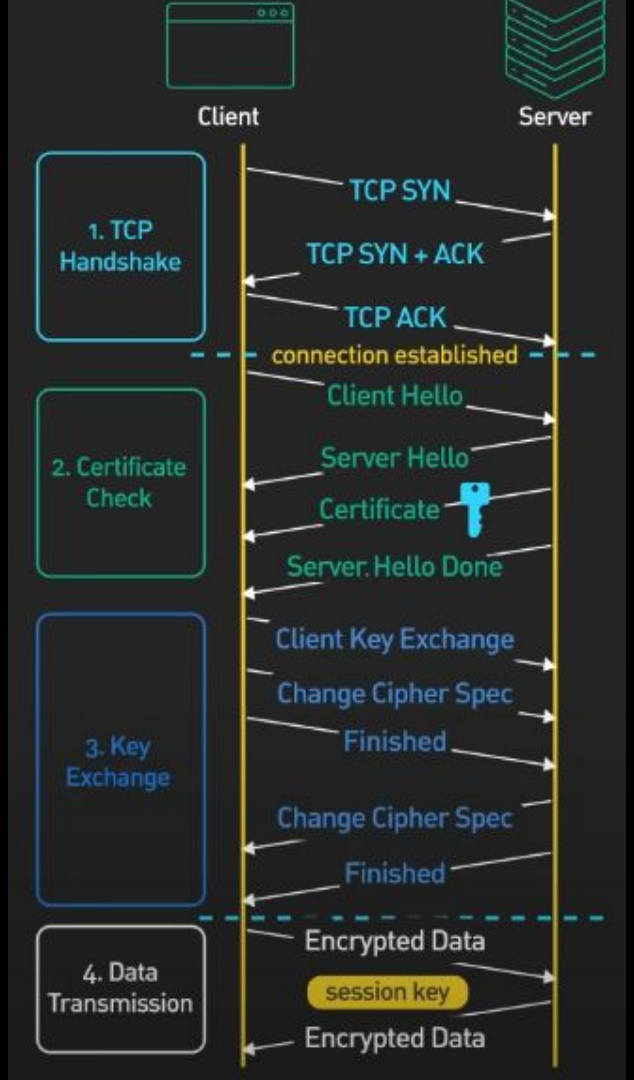
Where are we?



Version	Year introduced	Current status	Usage in August 2024	Support in August 2024
HTTP/0.9	1991	Obsolete	0	100%
HTTP/1.0	1996	Obsolete	0	100%
HTTP/1.1	1997	Standard	33.8%	100%
HTTP/2	2015	Standard	35.3%	66.2%
HTTP/3	2022	Standard	30.9%	30.9%

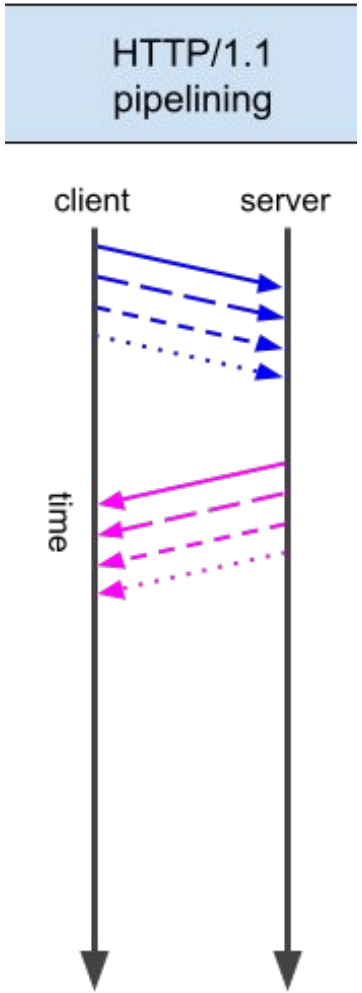
HTTP/0.9

- Introducido en 1991.
- Solo soporta el método GET.
- No tiene headers, devuelve texto HTML.
- Una sola request por conexión.
- Era muy limitado y está obsoleto.


Client
Server
1. TCP Handshake
TCP SYN
TCP SYN + ACK
TCP ACK
connection established
2. Certificate Check
Client Hello
Server Hello
Certificate
Server. Hello Done
3. Key Exchange
Client Key Exchange
Change Cipher Spec
Finished
Change Cipher Spec
Finished
4. Data Transmission
Encrypted Data
session key
Encrypted Data

HTTP/1.0

- Introducido en 1996.
- **Header y Body**
- Mismo problema que HTTP/0.9.
- Introduce **STATUS CODES.**
- Métodos POST, HEAD



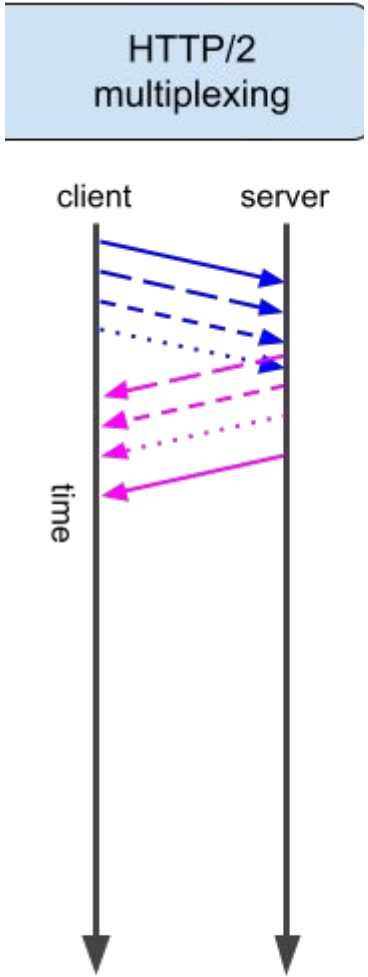
HTTP/1.1 (1er versión oficial)

- Introducido en 1997, sigue siendo popular.
- **Conexiones persistentes** por defecto (keep-alive).
- Pipelining disponible, pero poco usado por problemas de bloqueo en línea (head-of-line blocking).
- Estandarización de más métodos y headers.
- Estandarización de status codes (1xx, 2xx, 3xx, 4xx, 5xx)

CÓDIGOS DE ESTADO

- 1xx: Información (Poco común)
- **2xx: Éxito**
- 3xx: Redirección (Poco común)
- **4xx: Error en la request**
- **5xx: Error del servidor**

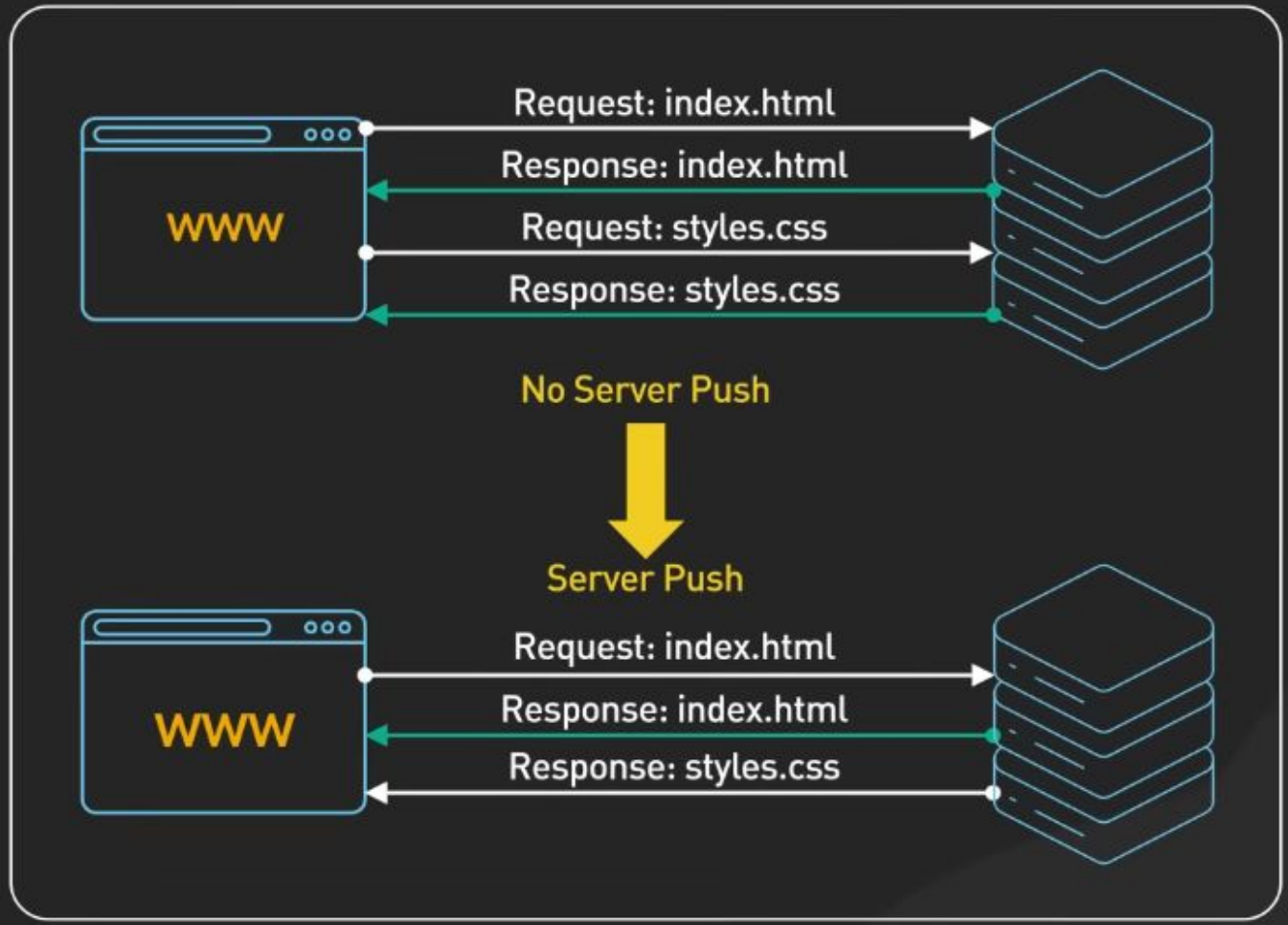
]



HTTP/2

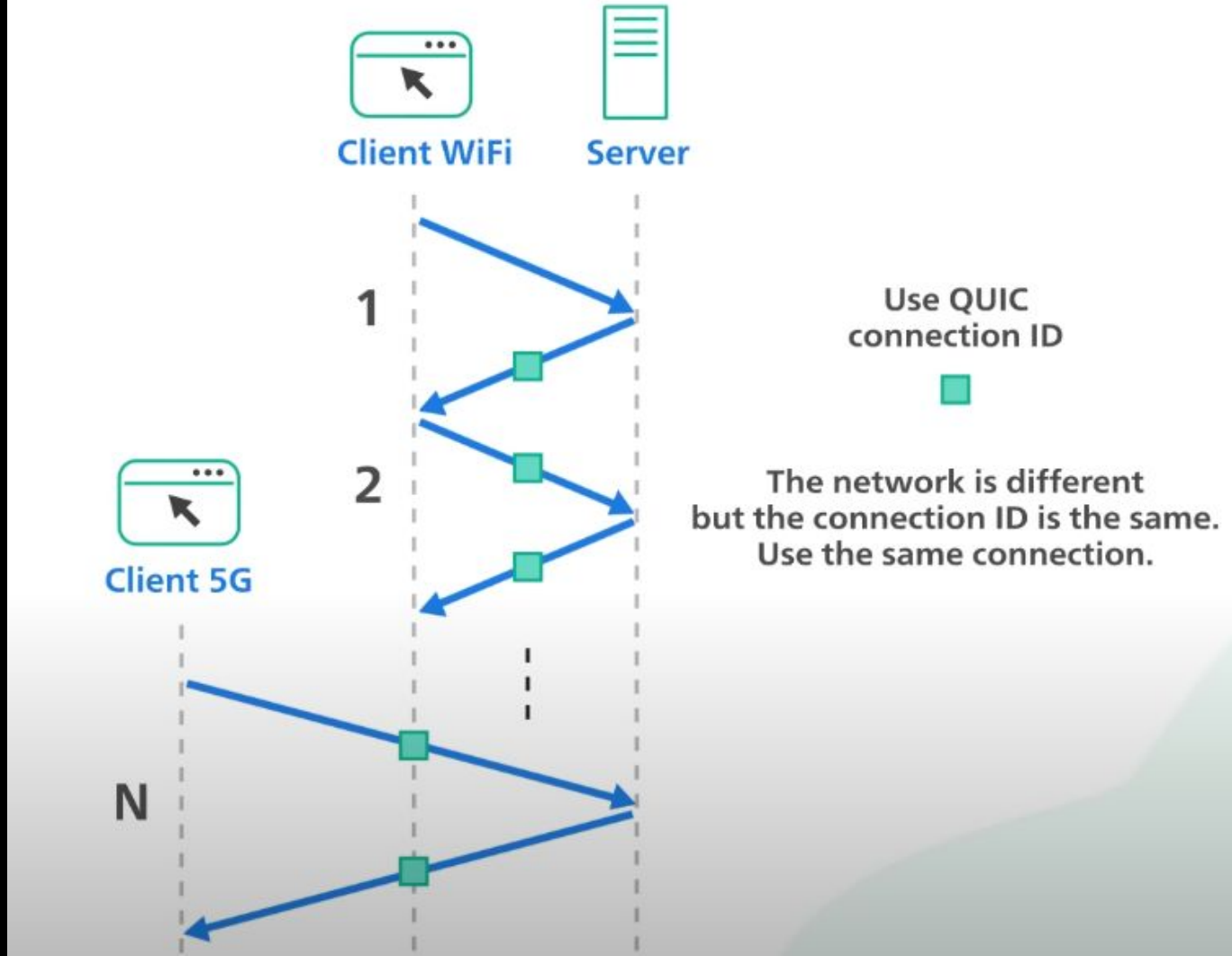
- Introducido en 2015, facil de portear desde HTTP/1.1
- Backwards compatibility
- Server push: Múltiples recursos con una sola request
- Multiplexing: Muchas solicitudes en una sola conexión.

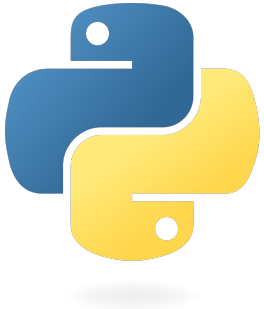
HTTP2 Server Push



HTTP/3

- Introducido en 2019 (depende de donde busques)
- Basado en QUIC: Reemplaza TCP (mayor velocidad)
- Requiere TLS 1.3 (siempre está cifrado)
- Conexión rápida y permite cambio de red sin que se corte.


Client WiFi
Server
1
2
Client 5G
N
Use QUIC
connection ID
The network is different
but the connection ID is the same.
Use the same connection.



APÉNDICE A: IMPLEMENTACIÓN SERVER HTTP

- **Rutas**
  - GET **getfile route**
  - POST write file route
  - GET user agent
  - GET echo
  - GET index
- **Funcionalidades**
  - Concurrencia
  - Subir archivos
  - Obtener archivos

- Usamos la librería **sockets** para las conexiones TCP.
- Usamos concurrent.futures para la concurrencia.


LIVE
DEMO

LOGICA PRINCIPAl

```python
def main():
    print("Server escuchando en el puerto: ", PORT)
    server_socket = socket.create_server(("::", PORT), family=socket.AF_INET6, reuse_port=True)

    with ThreadPoolExecutor(max_workers=50) as executor:
        while True:
            client_socket, addr = server_socket.accept()
            executor.submit(handle_client, client_socket)

def handle_client(client_socket):
    print("Client being handled !!!!")
    with client_socket:
        try:
            data = client_socket.recv(1024)
            request = data.decode()

            request_line = request.split("\r\n")[0]
            http_method , uri, http_protocol = request_line.split(" ")

            uri:str

            headers = get_headers(request)
```

USER AGENT


$ curl -i http://localhost:4221/user-agent
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 10
curl/8.5.0 (porky@porky-i5)-[~/darthpedro/p

GET FILE

```
└─$ curl -i http://localhost:4221/files/index.html
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 218

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Index</title>
</head>
<body>
    <h1>H.</h1>

</body>
```

WRITE FILE

```
└─$ curl -v --data "print('hola profe')" -H "Content-Type: application/octet-stream" http://localhost:4221/files/holaprofe.py
* Host localhost:4221 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:4221...
* Connected to localhost (::1) port 4221
> POST /files/holaprofe.py HTTP/1.1
> Host: localhost:4221
> User-Agent: curl/8.5.0
> Accept: */*
> Content-Type: application/octet-stream
> Content-Length: 19
>
< HTTP/1.1 201 Created
< Content-Length: 0
<
* Connection #0 to host localhost left intact
```

CONCURRENCIA

```bash
#!/bin/bash

TOTAL=500
echo "Mandando $TOTAL requests concurrentes a http://localhost:4221/ ..."

SECONDS=0

for i in $(seq 1 $TOTAL); do
    curl -s http://localhost:4221/ &
done

wait

echo "Tiempo total: $SECONDS segundos"
```

APÉNDICE B: Proxy reverso NGINX

Duck DNS

free dynamic DNS hosted on Amazon VPC

certbot

¿Para qué sirve?

- Abrir solo 2 puertos en el router
- Redirigir el tráfico del puerto 80 al puerto 443

Frontend

```
server {
    server_name aprendiendoconpersonajes.duckdns.org;

    # Proxy al frontend de Vite (localhost:5173)
    location / {
        proxy_pass http://localhost:5173;  # Conecta al frontend en localhost:5173
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
```

Backend

```
server {

include /etc/nginx/snippets/letsencrypt.conf;
server_name apiaprendiendoconpersonajes.duckdns.org;

# API en /api
location /api/ {
    rewrite ^/api/(.*)$ /$1 break;
    proxy_pass http://localhost:7080;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

Redirección automática a HTTPS

```
    server {
    if ($host = aprendiendoconpersonajes.duckdns.org) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80;
        server_name aprendiendoconpersonajes.duckdns.org;
    return 404; # managed by Certbot


}
```