

Linguagem de Programação para Web

Protocolo HTTP

Prof. Tales Bitelo Viegas

<https://fb.com/ProfessorTalesViegas>

A decorative graphic at the bottom of the slide consisting of several overlapping, wavy, horizontal bands. The colors include light blue, dark blue, black, and a grey band with a fine diagonal line pattern.

HyperText Transfer Protocol

- ▶ Padrão definido pelo W3C (www.w3c.org)
- ▶ Criado em 1994 (RFC 2616)
- ▶ Utiliza o TCP/IP como base
- ▶ Portas padrões:
 - Não-segura: 80
 - Segura: 443
- ▶ Define **toda a arquitetura da web**

Protocolo

- ▶ É um conjunto de regras de comunicação
- ▶ HTTP é um cliente-servidor clássico



Elementos

- ▶ Cliente (User-Agent)
 - Navegador (Browser)
 - spider
 - wget
 - curl
- ▶ Servidor (Server)
- ▶ Intermediários (Via)
 - Proxy
 - Gateway
 - Tunnel

Uma requisição HTTP

```
GET /index.html HTTP/1.1  
Host: gravatai.ulbra.tche.br  
\n  
\n
```

Uma requisição HTTP

Método

Recurso

Protocolo

GET /index.html HTTP/1.1

Host: gravatai.ulbra.tche.br

Cabeçalhos

\n

Linha em branco

Uma resposta HTTP

HTTP/1.1 200 Ok

Date: Wed, 14 March...

Content-Length: 3

Content-Type: text/html

\r\n

Oi!

Uma resposta HTTP

Protocolo

Status

HTTP/1.1 200 Ok

Date: Wed, 14 March...

Content-Length: 3

Content-Type: text/html

\r\n

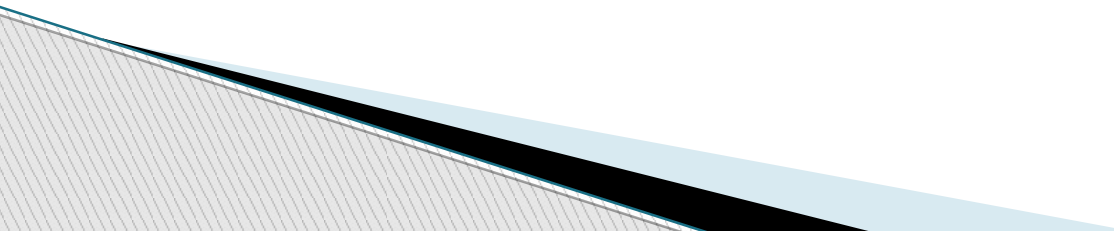
Linha em branco

Oi!

Corpo

Cabeçalhos

Métodos

- ▶ GET
 - solicita um recurso
 - ▶ POST
 - envia dados no corpo da mensagem
 - ▶ PUT
 - upload
 - ▶ HEAD
 - realiza uma requisição idêntica ao GET, mas só devolve os cabeçalhos (não o corpo da resposta)
 - ▶ DELETE
 - remove um recurso
- 

Métodos

▶ TRACE

- Devolve a mesma solicitação, para verificar se algum intermediário alterou a solicitação original

▶ OPTIONS

- Devolve os métodos suportados pelo servidor

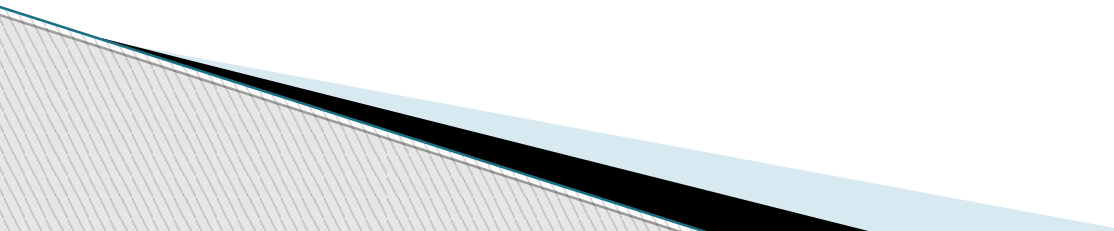
▶ CONNECT

- Utilizado para túneis TCP/IP, tipicamente para realizar conexões HTTPs através de um proxy HTTP

GET

- ▶ Método HTTP para **obter dados**
- ▶ Os parâmetros são parâmetros **de pesquisa**
- ▶ Formulários de busca, filtros de relatórios, etc.
- ▶ Ex:
 - <http://www.google.com.br/search?q=tales+bitelo+viegas>

POST

- ▶ Método HTTP para **enviar dados**
 - ▶ Os parâmetros são **dados a serem enviados**
 - ▶ Formulários de inserção, atualização, processamento remoto, etc.
- 

Códigos de Retorno

- ▶ Retorno da requisição, enviado pelo servidor
- ▶ Se agrupam em:
 - 1xx – Informação
 - 2xx – Sucesso
 - 3xx – Redirecionamento
 - 4xx – Erro de cliente
 - 5xx – Erro de servidor

Códigos de Sucesso (200)

- ▶ 200 Ok
 - Requisição foi realizada com sucesso
- ▶ 201 Created
 - Sua requisição gerou algo no servidor, aqui está ele
- ▶ 202 Accepted
 - Sua requisição foi aceita e o servidor está gerando algo
- ▶ 204 No Content
 - Deu tudo certo, mas não tenho nada para exibir

Códigos de Redirecionamento (300)

- ▶ 300 Multiple Choices
 - Achei várias coisas, escolha uma
- ▶ 301 Redirect
 - Isto que você procura mudou para sempre de lugar
- ▶ 302 Found
 - Isto que você procura, no momento está aqui
- ▶ 303 See Other
 - Sua requisição foi aceita, olha ela aqui
- ▶ 304 Not-Modified
 - Nada novo no servidor, nem se preocupe
- ▶ 307 Temporary Redirect
 - O que você procura está temporariamente aqui

Códigos de Erro de Cliente (400)

- ▶ 400 Bad Request
 - Não consigo entender o que você pediu
- ▶ 401 Unauthorized
 - Você não tem permissão para isso
- ▶ 403 Forbidden
 - Ninguém tem permissão para isso
- ▶ 404 Not Found
 - Não achei
- ▶ 405 Method Not-Allowed
 - Método não permitido

Códigos de Erro de Servidor (500)

- ▶ 500 Internal Server Error
 - Deu alguma m.... enquanto tratava a requisição
- ▶ 501 Not Implemented
 - Esse servidor não entende este método
- ▶ 502 Bad Gateway
 - Eu estava contando com outro servidor, mas ele pisou na bola
- ▶ 503 Service Unavailable
 - O servidor está sobrecarregado, espere um momento

Cabeçalhos

- ▶ Componentes da mensagem HTTP que definem parâmetros utilizados por quem envia e/ou recebe uma requisição
- ▶ Padronizados pela RFC 2616
- ▶ <http://www.iana.org/assignments/message-headers/perm-headers.html>

Cabeçalhos de Solicitação

- ▶ Host
 - Domínio a ser acessado
- ▶ Accept
 - Tipos de conteúdo que o cliente aceita
- ▶ Accept-Language
 - Linguagens disponíveis pelo cliente
- ▶ Accept-Charset
 - Caracteres que são aceitas no cliente
- ▶ Content-Type
 - Tipo de conteúdo do corpo da requisição
- ▶ Referer
 - URL da página que está realizando a requisição

Cabeçalhos de Resposta

- ▶ **Server**
 - Nome do servidor
- ▶ **Expires**
 - Data para até quando considerar o conteúdo válido
- ▶ **Last-Modified**
 - Última data da modificação do conteúdo
- ▶ **Location**
 - Localização do conteúdo (para redirecionamentos)
- ▶ **Set-Cookie**
 - Criação de cookies

Negociação de Conteúdos

- ▶ Mecanismo para responder recursos ou documentos diferentes, a partir de uma URL solicitada
 - Ex: tipo de imagem (gif, png), idioma, text/html ou text
- ▶ Cabeçalhos
 - Cliente. Ex: Accept-encoding: gzip, deflate
 - Servidor. Ex: Content-encoding: gzip

Negociação de Conteúdos

- ▶ Tipos MIME (Multipurpose Internet Mail Extensions)
 - Forma de descrever o tipo de documento a transmitir
 - Sintaxe: tipo major/tipo minor. Ex: text/html, image/gif
 - Baseado geralmente na extensão ou em análise do arquivo

Características

▶ **Chunked transfer encoding**

- Dividir o conteúdo em fragmentos e enviar
- Vantagens:
 - ▢ Transmissão streaming ao invés de buffered.
 - ▢ Possibilidade de enviar conteúdos a partir do servidor antes de conhecer o tamanho total do arquivo
 - ▢ Compressão
- Cabeçalho de resposta
 - ▢ Transfer-encoding: chunked

▶ **Byte serving**

- Envia apenas a parte do arquivo que o cliente indicar

Características

- ▶ Conexões persistentes (HTTP Keep-Alive)
 - Possibilita fazer várias requisições HTTP em uma única conexão TCP
 - Vantagens:
 - ▢ Menor carga do sistema operacional
 - ▢ Menor congestionamento na rede e latência em conexões posteriores
 - ▢ Possibilidade de HTTP pipelining
 - Segundo a RFC2616, um cliente não deveria estabelecer mais de 2 conexões persistentes ao mesmo tempo com um mesmo servidor

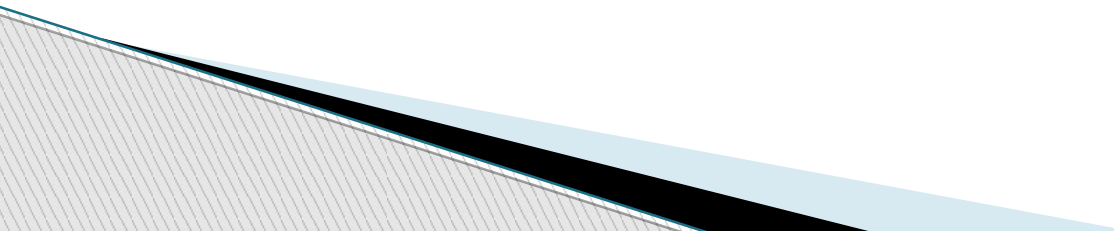
Segurança

- ▶ Métodos de autenticação próprios do HTTP:
 - Basic access: não realiza encriptação
 - Digest: baseado em MD5 com SALT
 - ▢ Considera usuário, senha e realm (domínio ou espaço)
 - ▢ Vulnerável a ataque “man-in-the-middle”
- ▶ Outro método:
 - HTTPs: HTTP sobre SSL/TLS (o melhor)


Gestão de Sessões

- ▶ Protocolo Stateless: não armazena informações sobre os clientes
- ▶ Técnica para manter histórico:
 - Session/Local Storage (HTML 5)
 - Cookies (RFC 2109)
 - URL rewriting informando ID de sessão
 - Campos ocultos em formulários

HTTP/2.0

- ▶ Formalmente aprovado pelo IETF em 17/02/2015
 - ▶ Protocolo baseado no SPDY (2009)
 - ▶ Foco em Performance
- 

Problemas do HTTP/1.1

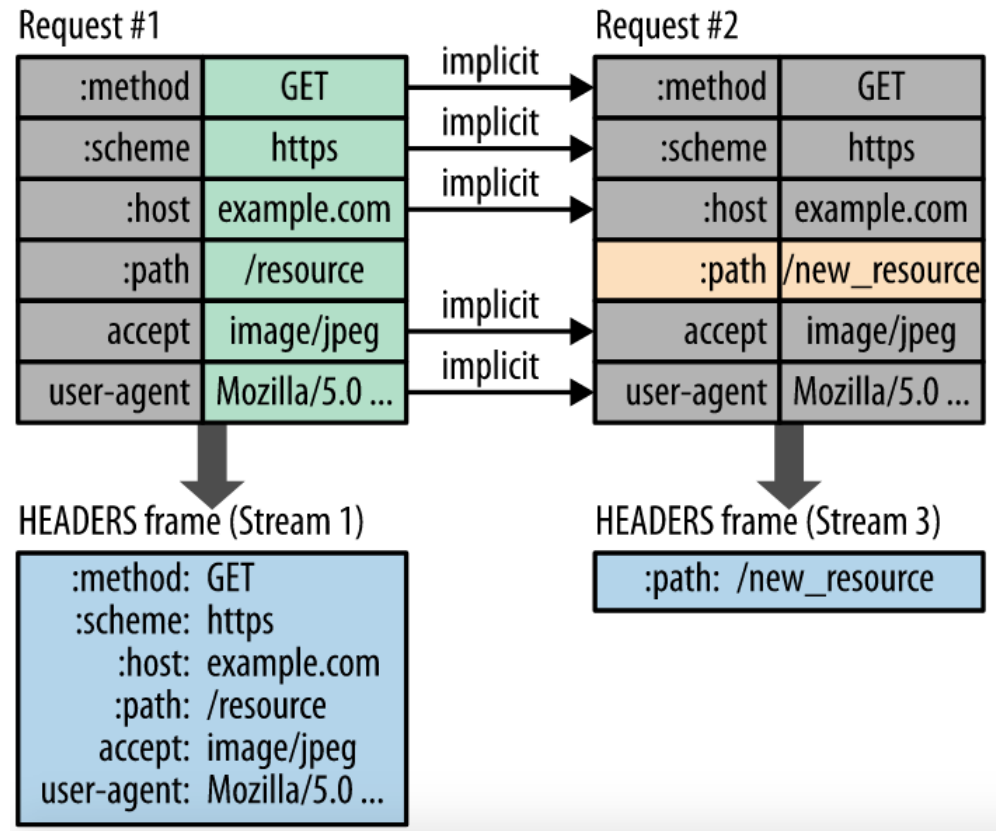
- ▶ Insegurança
 - Nem toda a comunicação é feita com SSL
 - ▶ Performance
 - Tráfego de dados “inúteis” (cabeçalhos repetidos)
 - Sem otimização de processamento de cliente ou servidor
 - ▶ Compressão e Otimização são opcionais
 - ▶ Limitação de Request Simultâneos
- 

HTTP/2.0

- ▶ Segurança
 - SSL por default
 - HPACK - Header Compression
 - Conversão binária dos dados (criptografados)
- ▶ Performance
 - GZIP por default
 - Apenas os cabeçalhos que mudam

HTTP/2.0

- ▶ Protocolo binário
- ▶ Cabeçalhos comprimidos por HPACK
 - Código que reduz o tamanho da transferência
 - Parâmetros repetidos podem ser evitados



HTTP/2.0

▶ Fluxo

- Um fluxo bidirecional de bytes dentro de uma conexão estabelecida, que pode conter uma ou mais mensagens

▶ Mensagem

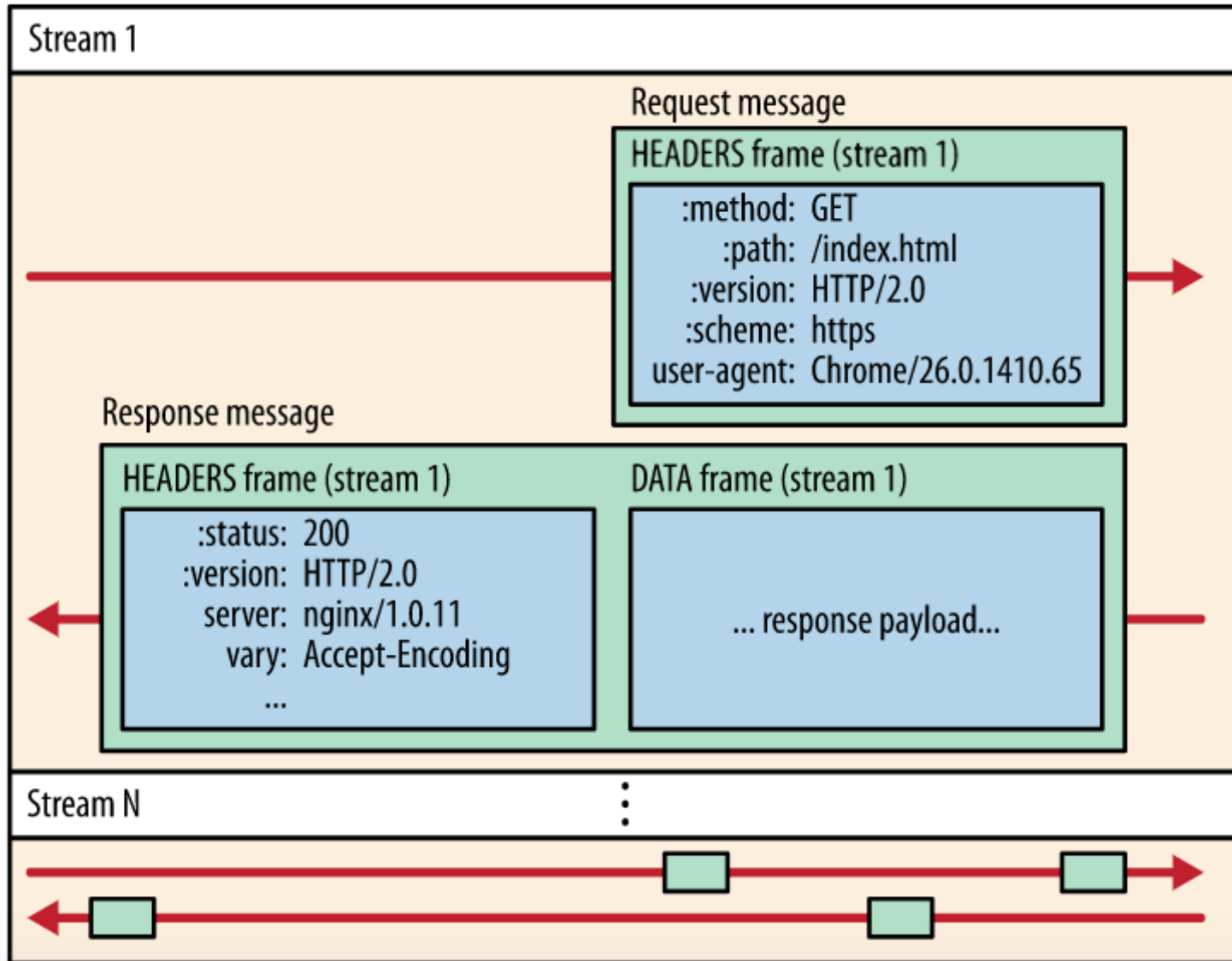
- Uma sequência completa de quadros vinculada a uma solicitação ou resposta

▶ Quadro

- A menor unidade de comunicação do HTTP/2.0, com cada uma contendo um cabeçalho

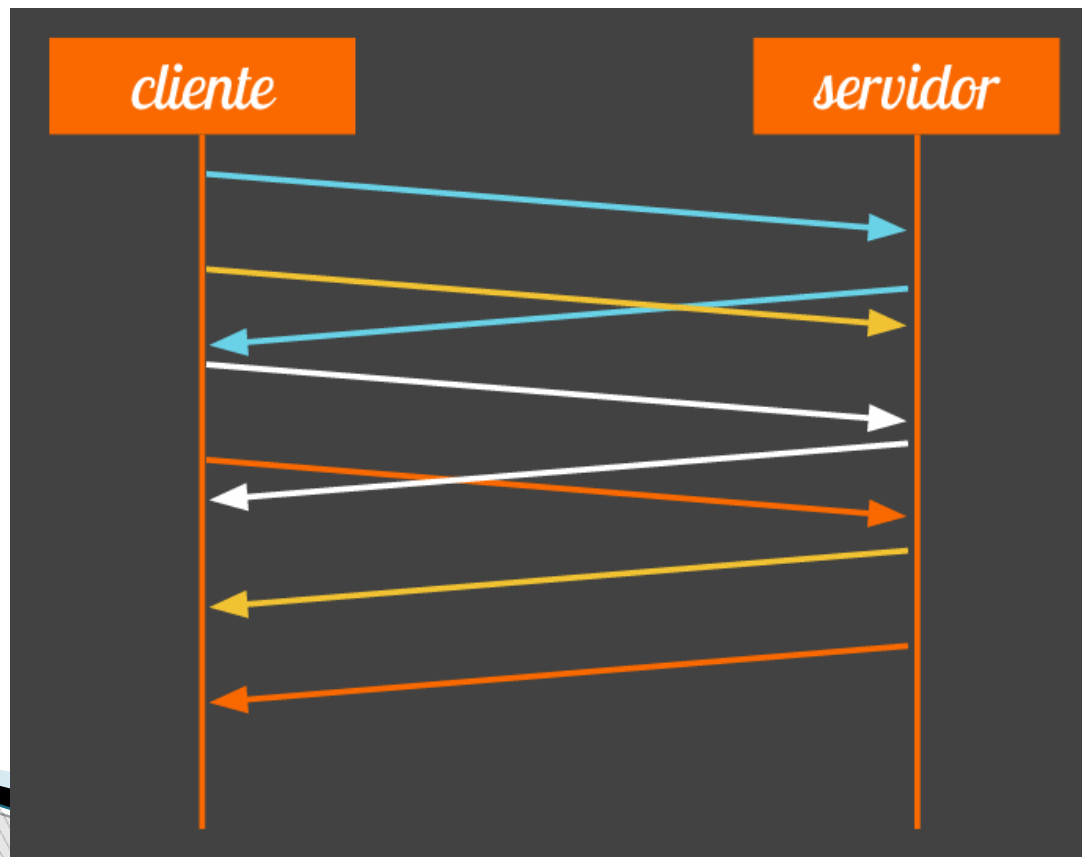
HTTP/2.0

Connection

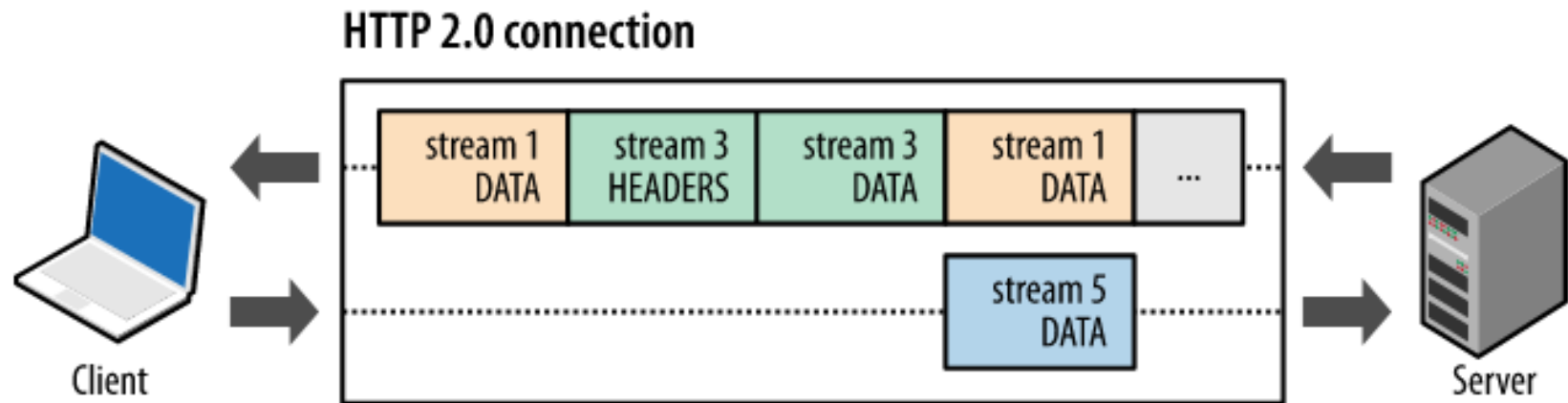


HTTP/2.0

- ▶ Multiplexing
 - Conexão TCP assíncrona

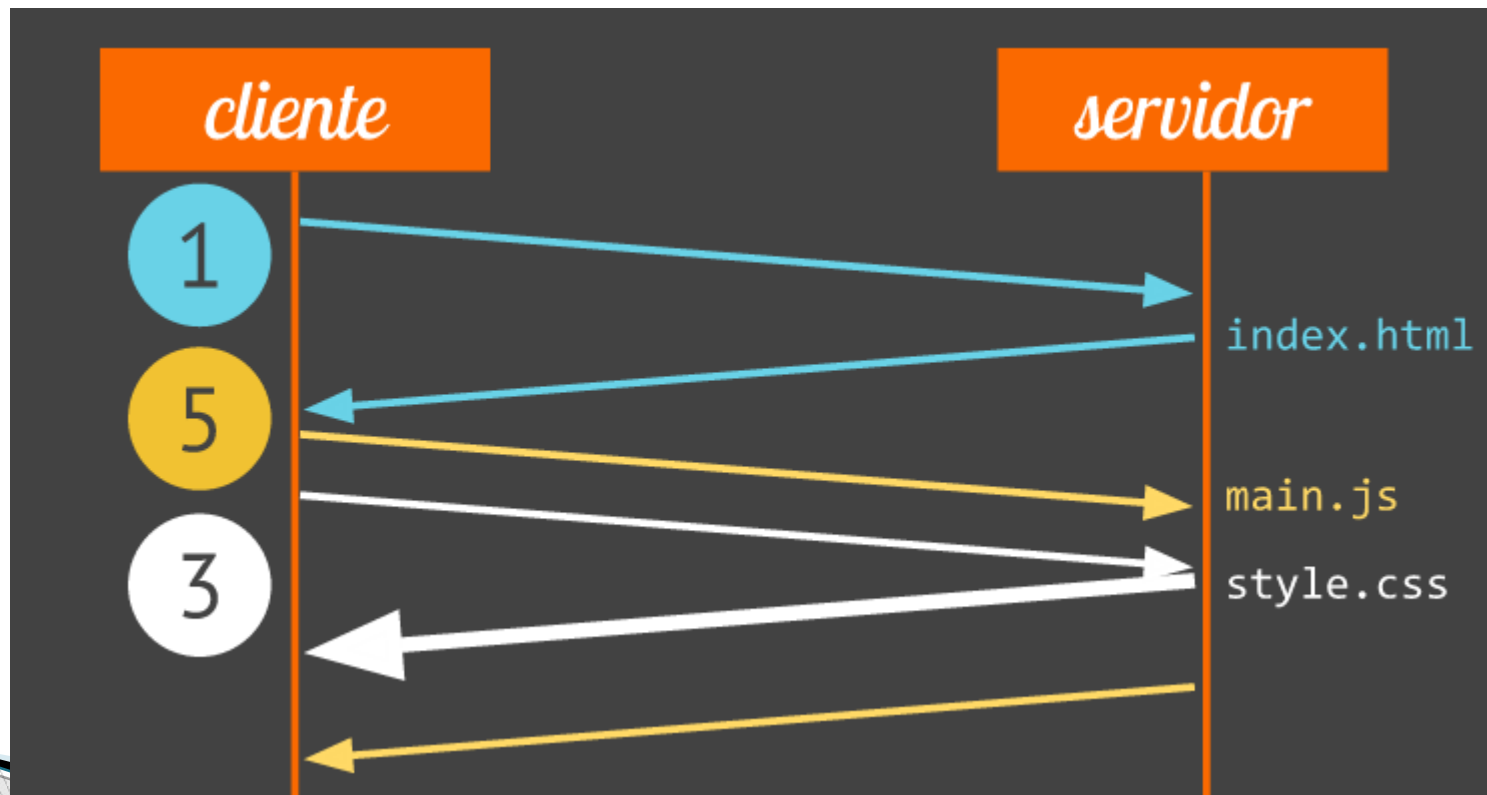


HTTP/2.0



HTTP/2.0

- Priorização de Requests



HTTP/2.0

- ▶ Navegadores definem as prioridades dos conteúdos
- ▶ Podem ser alteradas através da tag HTML link para definir:
 - Preload de conteúdo
 - Preconnect em servidores

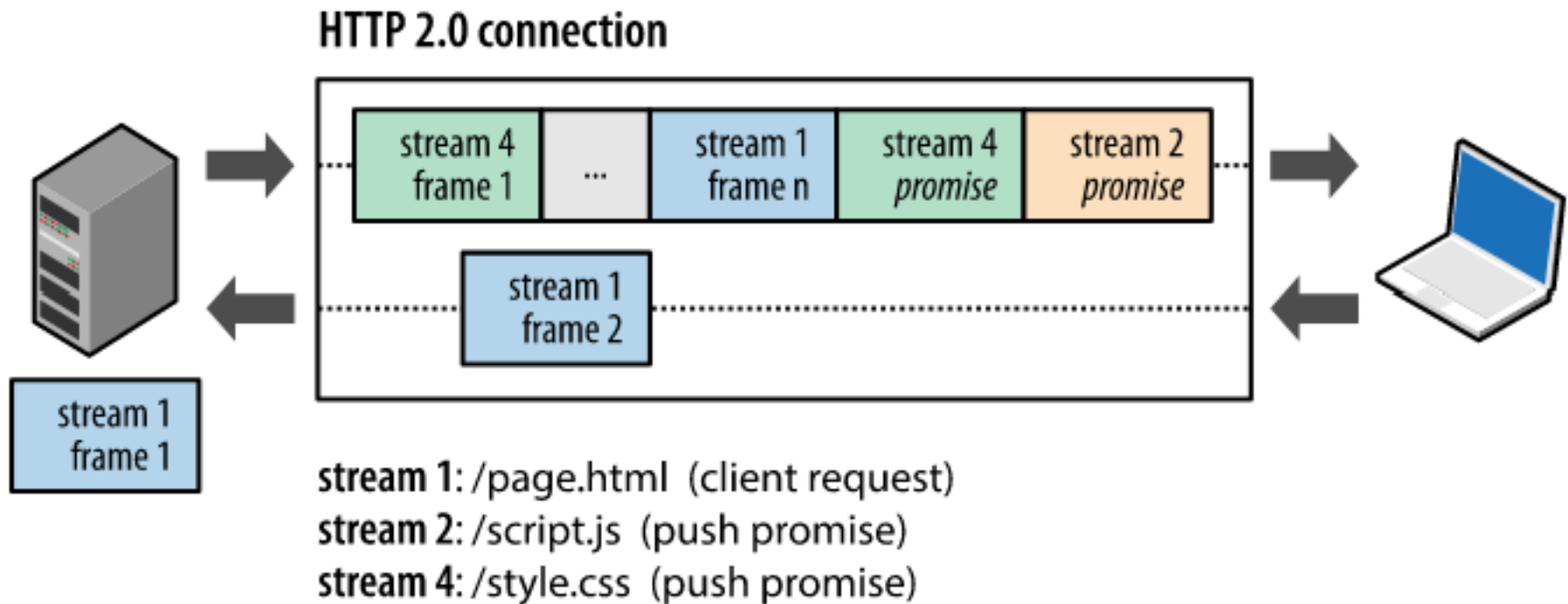
<https://developers.google.com/web/fundamentals/performance/resource-prioritization>



HTTP/2.0

► Server-Push

- Envio de recursos mesmo sem o cliente ter solicitado ainda



HTTP/2.0

- ▶ Clientes que suportam
 - IE > 11 (Windows 10)
 - Firefox
 - Chrome
- ▶ Servidores que suportam
 - Nginx
 - IIS (Windows > 10)
 - Apache >= 2.4.17
 - OpenLite Speed
 - Módulos para quase todos os servidores

<https://github.com/http2/http2-spec/wiki/Implementations>



HTTP/2.0

- ▶ Resolve problemas do HTTP/1.1
- ▶ Não é necessário alterar as aplicações para o novo protocolo continuar funcionando
- ▶ Principais navegadores já utilizam o protocolo
- ▶ Algumas melhorias para desenvolvedores
 - Minificar CSS/JS se torna desnecessário
 - CSS Sprite se torna desnecessário
- ▶ 24,2% dos websites já usam HTTP/2.0
<https://w3techs.com/technologies/details/ce-http2/all/all>