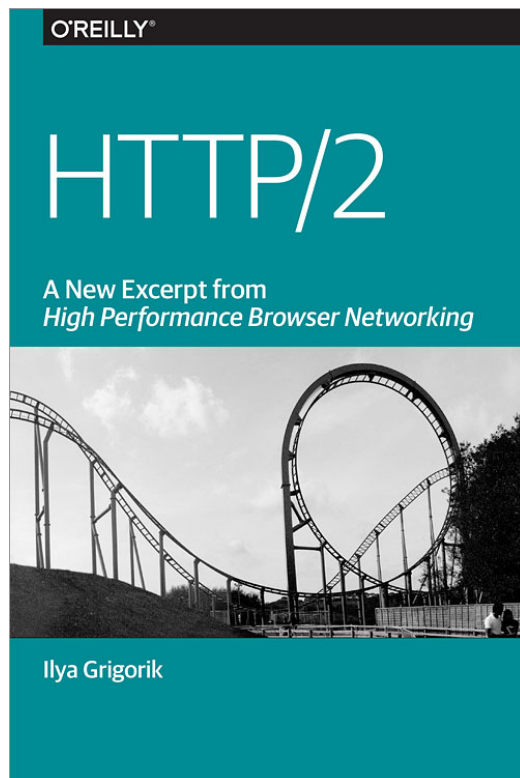
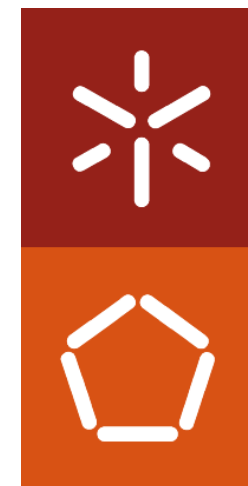


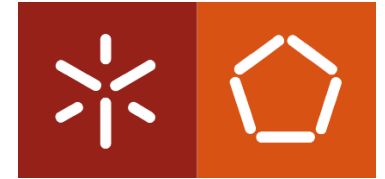
HTTP2 e QUIC



Comunicações por Computador
Mestrado Integrado em Engenharia Informática
3º ano/2º semestre
2016/2017

Disponível online (grátis): hpbn.co/http2
Slides: bit.ly/http2-opt



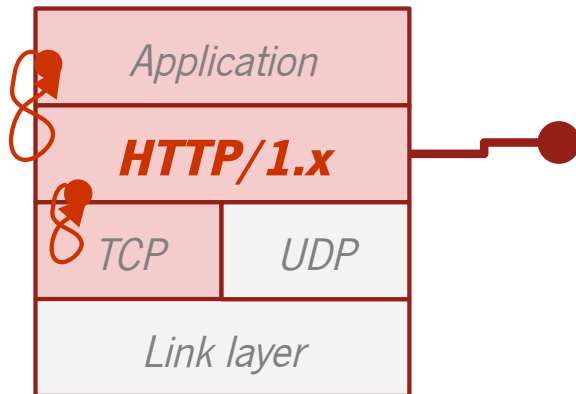
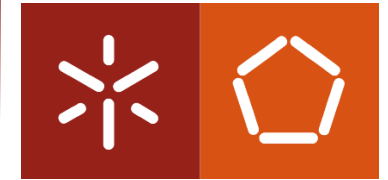


Desempenho HTTP/1.*

- **Desempenho do HTTP: Quais as melhores práticas?**

- Reduzir o número de consultas ao DNS (*DNS Lookups*)
- Reutilizar conexões TCP
- Utilizar CDNs (*Content Delivery Network*)
- Minimizar o número de redireccionamentos HTTP (*HTTP Redirects*)
- Eliminar bytes desnecessários nos pedidos HTTP (cabeçalhos)
- Comprimir os artefactos na transmissão (compressão corpo)
- Cache dos recursos do lado do cliente
- Eliminar o envio de recursos desnecessários

Problemas do HTTP/1.*



Paralelismo limitado

- *O paralelismo está limitado ao número de conexões*
- *Na prática, mais ou menos 6 conexões por origem*

Head-of-line blocking

- *Bloqueio do cabeça de fila, acumula pedidos em queue e atrasa a solicitação por parte do cliente*
- *Servidor obrigado a responder pela ordem (ordem restrita)*























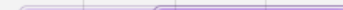




Overhead protocolar é elevado

- *Metadados do cabeçalho não são compactados*
- *Aproximadamente 800 bytes de metadados por pedido, mais os cookies*



Problemas do HTTP/1.*

● Paralelismo é limitado pelo número de conexões...

×	Elements	Resources	Network	Sources	Timeline	Profiles	Audits	Console	PageSpeed	
Name	Method	Status	Type	...	Time	Start Time	302 ms	453 ms	604 ms	755 m
 localhost	GET	200	text/html	...	17 ms					
 01.jpeg	GET	202	image/jpeg	...	242 ms					
 02.jpeg	GET	202	image/jpeg	...	243 ms					
 03.jpeg	GET	202	image/jpeg	...	242 ms					
 04.jpeg	GET	202	image/jpeg	...	241 ms					
 05.jpeg	GET	202	image/jpeg	...	235 ms					
 06.jpeg	GET	202	image/jpeg	...	235 ms					
 07.jpeg	GET	202	image/jpeg	...	475 ms					
 08.jpeg	GET	202	image/jpeg	...	563 ms					
 09.jpeg	GET	202	image/jpeg	...	561 ms					
 10.jpeg	GET	202	image/jpeg	...	561 ms					
 11.jpeg	GET	202	image/jpeg	...	561 ms					
 12.jpeg	GET	202	image/jpeg	...	561 ms					

~6 parallel downloads per origin

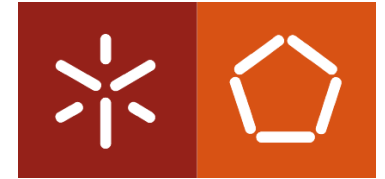
- Cada conexão implica overhead de handshake inicial
- Se for HTTPS, ainda tem mais um overhead do handshake TLS
- Cada conexão gasta recursos do lado do servidor
- As conexões competem umas com as outras



Problemas do HTTP/1.*

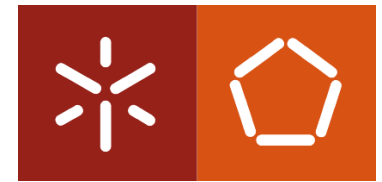
- **Porque não subdividir em N sub-domínios, em vez de um único domínio por servidor? (*domain sharding*)**
 - Aumenta o paralelismos — passamos a ter 6 conexões por subdomínio
 - Aumenta as consultas ao DNS...
 - Mais servidores, competição nas conexões, complexidade nas aplicações
- **Reduzir pedidos → concatenar objetos (*concatenated assets*)**
 - Vários CSS ou vários JS num único objeto! Resulta...
 - Atrasa o processamento no cliente, pode dificultar o uso da cache
- **Incluir recursos em linha no HTML (*inline objects*)**
 - Os mesmos objetivos do anterior: reduzir pedidos, antecipar conteúdos...
 - Os mesmos problemas: atrasa processamento no cliente, dificulta o uso da cache

HTTP2



- Em meados de **2009**, a **Google** inicia o seu projeto **SPDY!**
 - Objetivo nº1: reduzir em 50% o tempo de carregamento de página (***PLT Page Load Time***)
 - Outros objetivos:
 - Evitar que os autores Web tenham de mexer nos conteúdos
 - Minimizar o tempo de implantação e as alterações na infraestrutura
 - Desenvolver em parceria com a comunidade Open Source
 - Teste com dados reais que validem ou invalidem o protocolo
- Clientes: **Firefox, Opera e Chrome aderiram rapidamente...**
- Servidores: **Twitter, Facebook, e Google, claro!...**
- E o **IETF** (Internet Engineering Task Force)?
 - Teve de ir atrás, a reboque, e formar um grupo de trabalho **HTTP/2**

HTTP2



- Normalizado em menos de 3 anos!! Muita pressão...



Mid 2009: SPDY introduced as an experiment by google

Mar, 2012: Firefox 11 had support, turned on by default in version 13

Mar, 2012: Call for proposals for HTTP/2 – resulted in 3 proposals but SPDY was chosen as the basis for H/2

Nov, 2012: First draft of HTTP/2 (based on SPDY)

Aug, 2014: HTTP/2 draft-17 and HPACK draft-12 are published

Aug, 2014: Working Group last call for HTTP/2

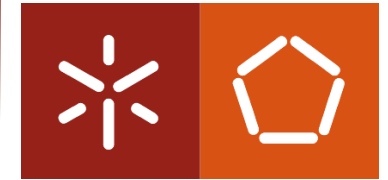
Feb, 2015: (IESG) Internet Engineering Steering Group approved HTTP/2

HTTP2



- **HTTP2 é uma extensão e não uma substituição do HTTP/1.1**
 - Não se mexe nos métodos, URLs, headers, códigos de resposta, etc.
 - **Semântica** – para a aplicação – deve é a mesma!
 - Não há alterações na **API** **aplicacional**...
- **Alvo → as limitações de desempenho das versões anteriores**
 - Primeiras versões do HTTP foram desenhadas para serem de fácil implementação!
 - Clientes HTTP/1.* obrigados a lançar várias conexões em paralelo para baixar a latência.
 - Não há compressão nem prioridades
 - **Mau uso** da conexão TCP de suporte!...

HTTP2 – Tudo num único slide!



1. Uma única conexão TCP!

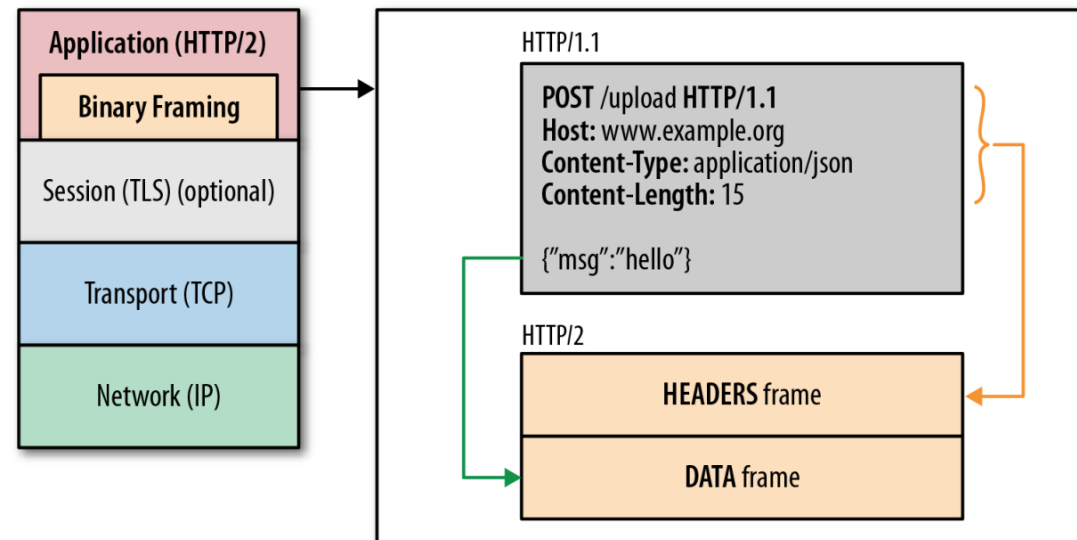
2. Request → Stream

- Streams são multiplexadas!
- Streams são priorizadas!

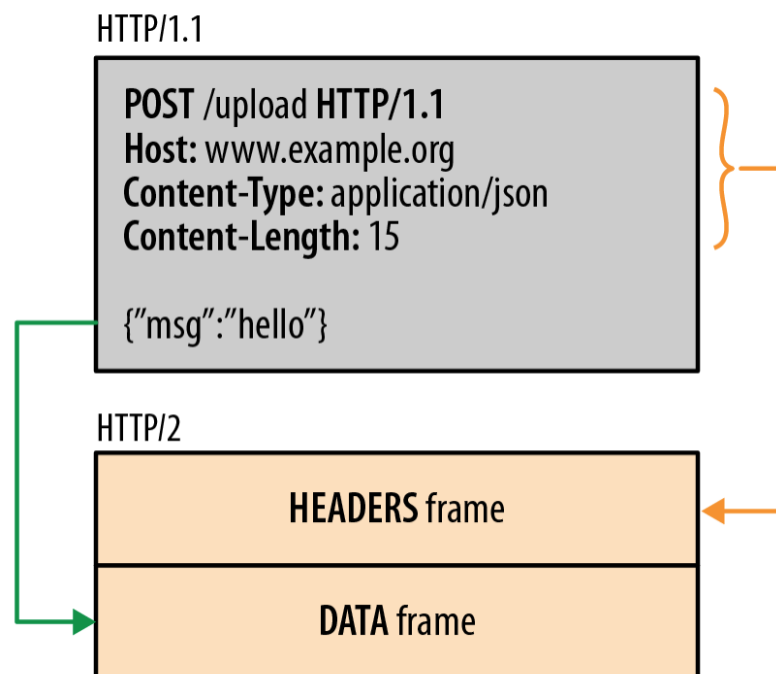
3. Camada de “*framing*” binário

- Priorização
- Controlo de Fluxo
- Server push

4. Compressão do cabeçalho (HPACK)



HTTP2 – “Framing” binário



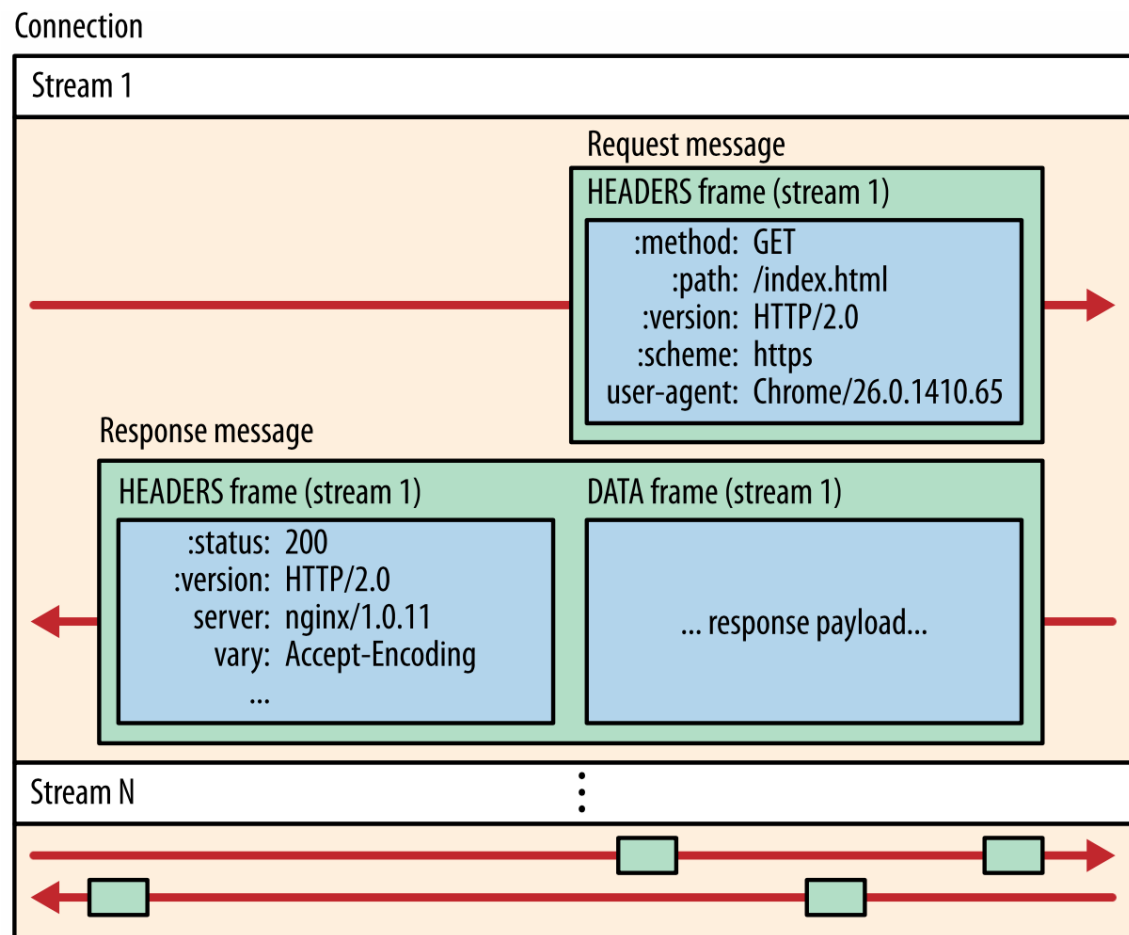
- **Mensagens HTTP são divididas em uma ou mais frames**
 - HEADERS para metadados
 - DATA para dados (*payload*)
 - RST_STREAM para cancelar
 - ...
- **Cada frame tem um cabeçalho comum**
 - 9-byte, com tamanho à cabeça
 - De *parsing* fácil e eficiente

HTTP2 – “Framing” binário



● Terminologia HTTP2

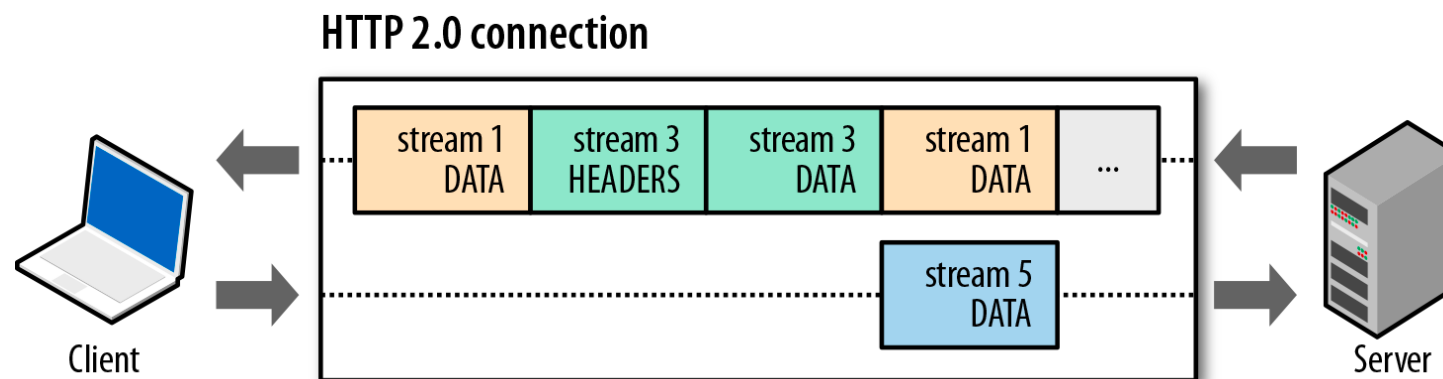
- **Stream** – um fluxo bidirecional de dados, dentro de uma conexão, que pode carregar uma ou mais mensagens
- **Mensagem** - Uma sequência completa de *frames* que mapeiam num pedido ou numa resposta HTTP
- **Frame** – A unidade de comunicação mais pequena no HTTP2, contendo um cabeçalho que no mínimo identifica a *Stream* a que pertence



HTTP2 – fluxo de dados



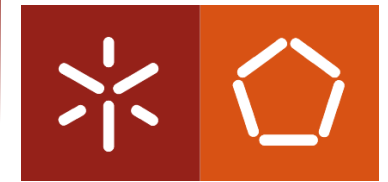
● Fluxo de dados numa conexão HTTP2



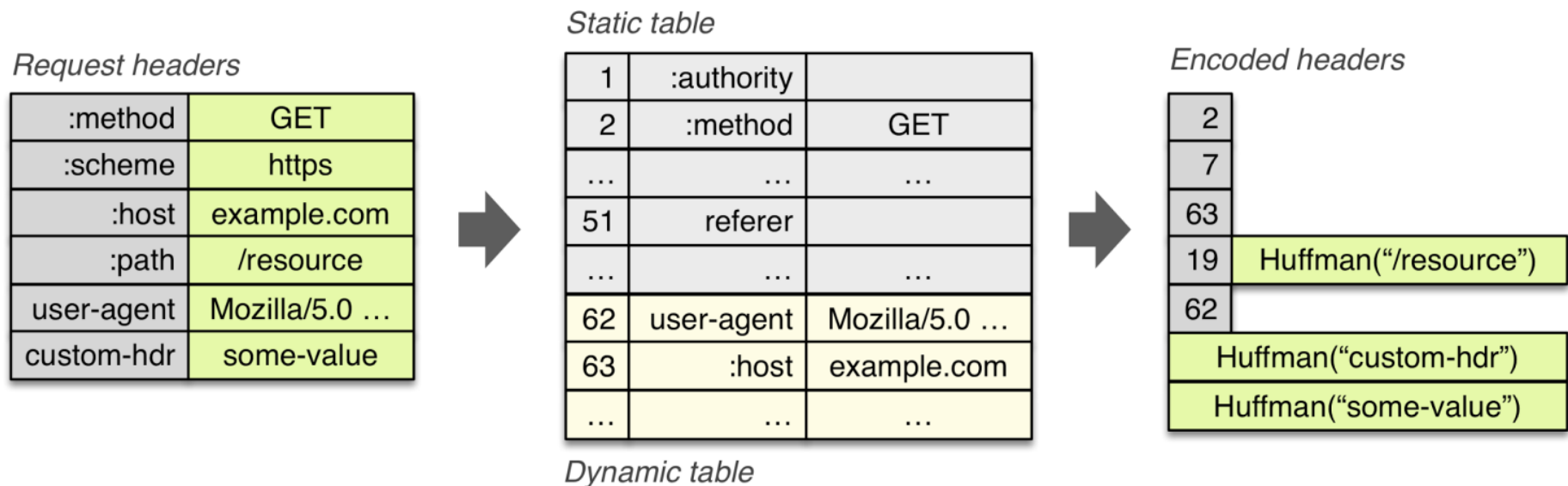
As streams são multiplexadas porque as frames pode ser intercaladas umas com as outras!

- Todas as frames (ex: **HEADERS**, **DATA**, etc.) são enviadas numa única conexão TCP
- A frames são entregues por prioridades, tendo em conta os pesos das streams e as dependências entre elas!
- As frames **DATA** estão sujeitas a um controlo de fluxo por stream e por conexão

HTTP2 – Compressão do cabeçalho

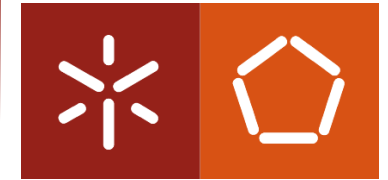


● HPACK



- Valores literais (texto) são codificados com **código de Huffman** estático
- Tabela **indexação estática** → por ex: "2" corresponde a "method: GET"
- Tabela **indexação dinâmica** → Valores enviados anteriormente pode ser indexados!

HTTP2 – Server “push”



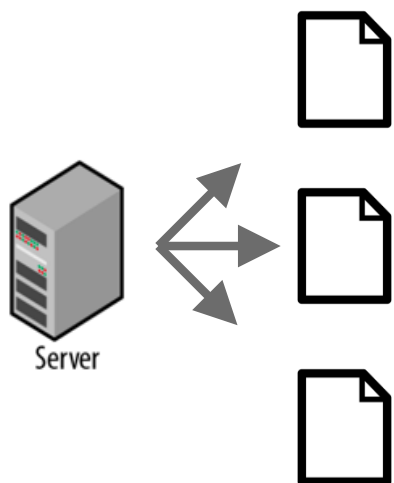
Server: “You asked for `/product/123`, but you’ll need `app.js`, `product-photo-1.jpg`, as well... I promise to deliver these to you. That is, unless you decline or cancel.”

- Maior granularidade no envio de recursos
 - Evita o inlining e permite caching eficiente dos recursos
 - Permite multiplexar e definir prioridades no envio dos recursos
 - Precisa de controlo de fluxo, para o cliente dizer basta/quero mais

HTTP2 – Server “push”



- Há espaço para estratégias de “Server push” inteligente
- Ex: implementação Jetty



1. Servidor observa o tráfego de entrada

- Constrói um modelo de dependências baseado no campo **Referer** do cabeçalho (ou outras):
 - e.g. `index.html` → `{style.css, app.js}`

2. Servidor inicia um push inteligente de acordo com as dependências que aprendeu

- `client` → `GET index.html`
- `server` → `push style.css, app.js, index.html`

HTTP2 – Controlo de fluxo



→ **Client:** “I want first 20KB of photo.jpg”

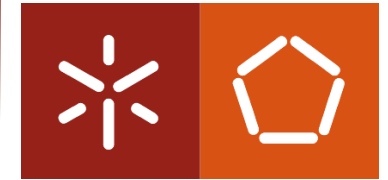
→ **Server:** “Ok, 20KB... pausing stream until you tell me to send more.”

→ **Client:** “Send me the rest now.”

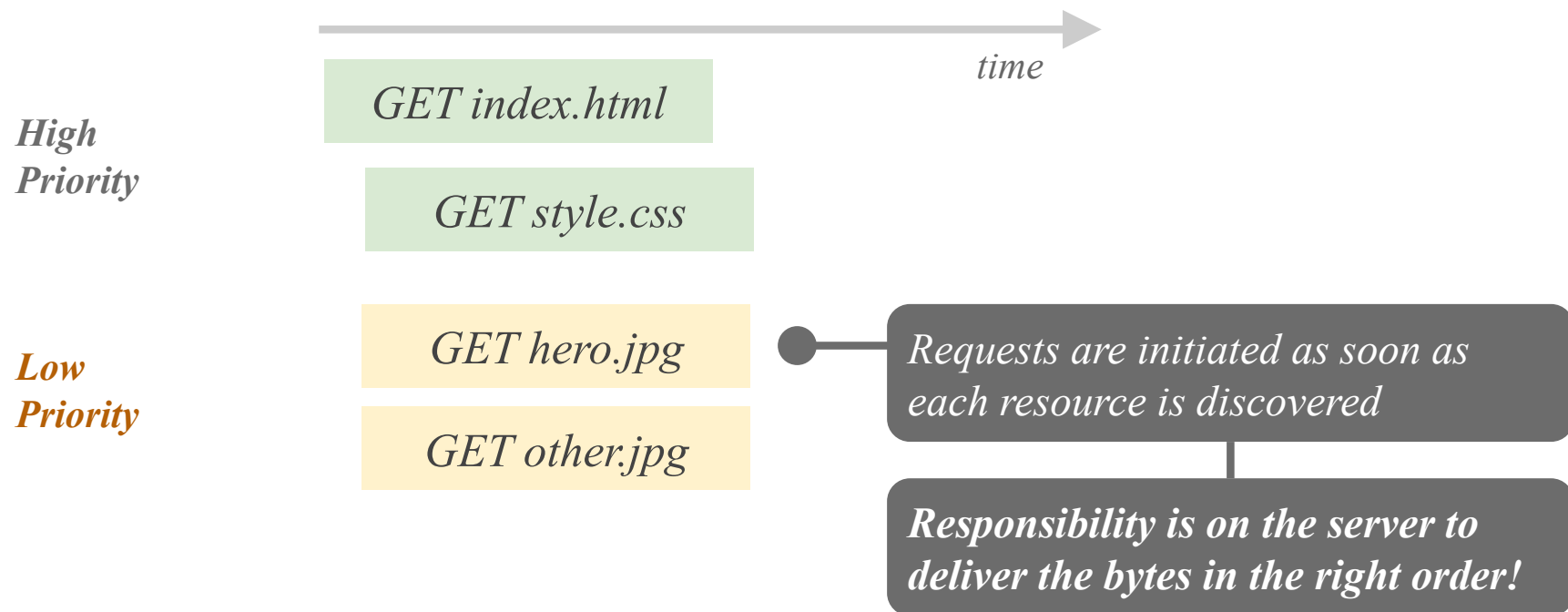
I want image geometry and preview, and I'll fetch the rest later...

- Permite ao cliente fazer uma pausa na *stream* e retomar o envio mais tarde
- Controlo de fluxo baseado num sistema de créditos (janela):
 - Cada frame do tipo **DATA** decrementa o valor
 - Cada frame do tipo **WINDOW_UPDATE** atualiza o valor

HTTP2 – priorização



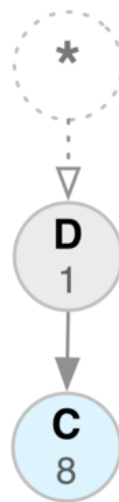
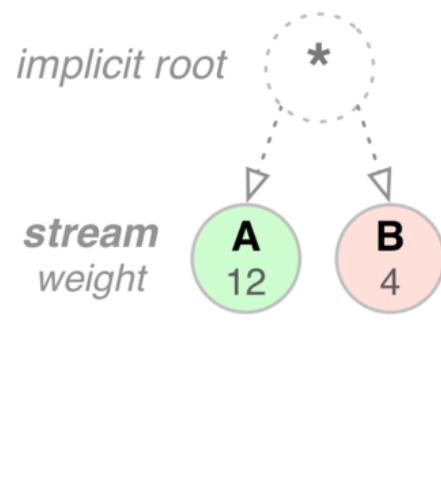
- Priorização é fundamental para um *rendering* eficiente!
- Com HTTP2, o cliente define as prioridades e faz logo os pedidos; cabe ao servidor entregar os conteúdos com a prioridade certa



HTTP2 – pesos e dependências



- Exemplo: stream A deve ter 12/16 e a B 4/16 dos recursos totais
- Exemplo: stream D deve ser entregue antes da stream C



- Cada **stream** pode ter um peso
 - [1-256] integer value
- Cada stream pode ter uma dependência
 - ... uma outra stream ID

HTTP2 – Negociação protocolar



- **http://** pode ser servido tanto em HTTP/1.* como em HTTP2
- **Mecanismo de Upgrade:**

```
GET /page HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c ①
HTTP2-Settings: (SETTINGS payload) ②
```

```
HTTP/1.1 200 OK ③
Content-length: 243
Content-type: text/html

(... HTTP/1.1 response ...)
```

(or)

```
HTTP/1.1 101 Switching Protocols ④
Connection: Upgrade
Upgrade: h2c

(... HTTP/2 response ...)
```

1. Cliente começa em HTTP1.1 e pede upgrade para HTTP2

2. Settings codificados em BASE64

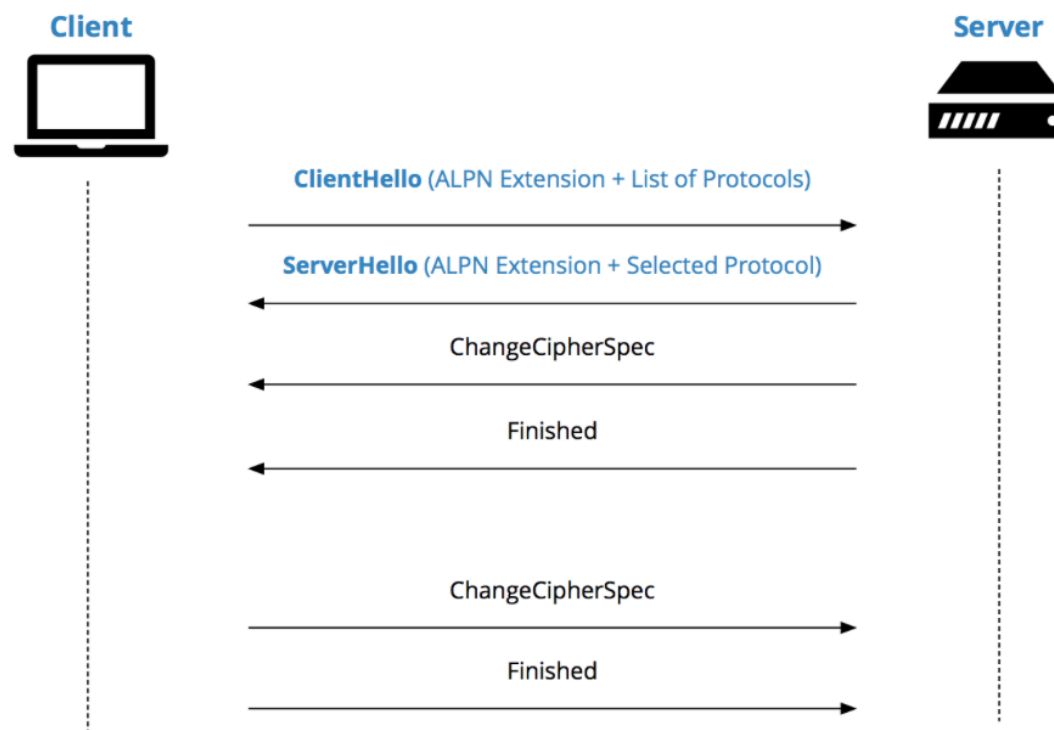
3. Servidor declina pedido, respondendo em HTTP/1.1

4. Servidor aceita pedido para HTTP2 e começa Framing binário

HTTP2 – Negociação protocolar



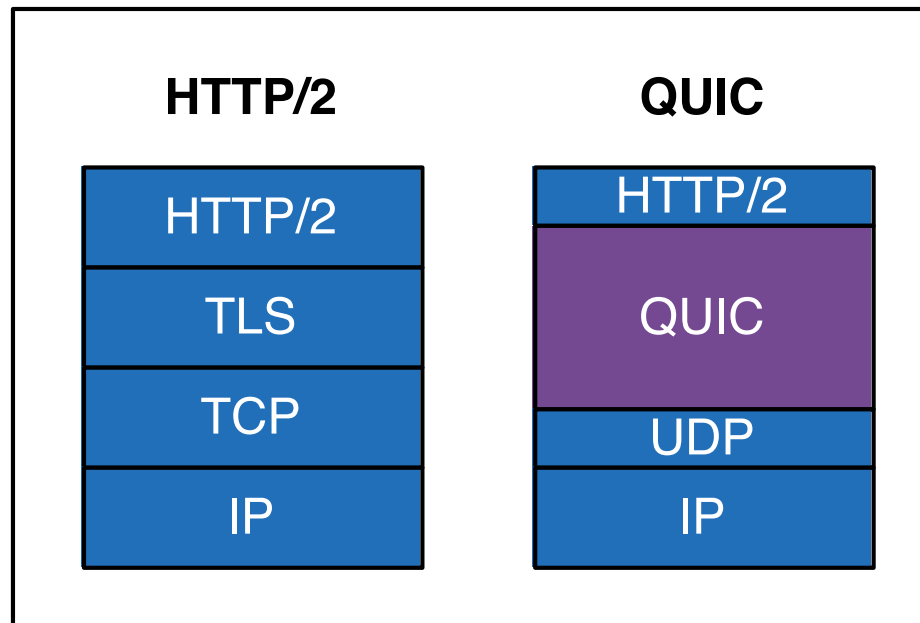
- **https://** pode ser servido quer em HTTP/1.* ou HTTP2
- Com HTTPS, negocia-se o protocolo na fase de Handshake do TLS, ao mesmo tempo que se migra para conexão segura:



QUIC



- **Controlo de congestão, cifragem e parte do HTTP2 muda-se para o QUIC que vai correr sobre UDP...**



HTTP2 e QUIC



● Links úteis:

- Software com suporte HTTP2:
 - <https://github.com/http2/http2-spec/wiki/Implementations>
- Browsers e suporte HTTP2:
 - <http://caniuse.com/#feat=http2>
- Demo da AKAMAI:
 - <https://http2.akamai.com/demo>
- Estatísticas de adoção - quem está a usar o quê?:
 - <https://blog.shodan.io/tracking-http2-0-adoption/>
- Desenvolvimento (Java):
 - <https://webtide.com/introduction-to-http2-in-jetty/>
 - <http://unrestful.io/2015/10/10/http2-java-client-examples.html>