# Lezione 1: WWW e HTTP

# World Wide Web (WWW)

- World Wide Web applicazione Internet più usata
  - Definita da Tim Berners-Lee nel 1990 (CERN – Ginevra)
  - Fornisce accesso a contenuto multimediale
  - Nodi (pagine Web) memorizzati su macchine in tutto il mondo e connesso alla rete Internet
  - Open service

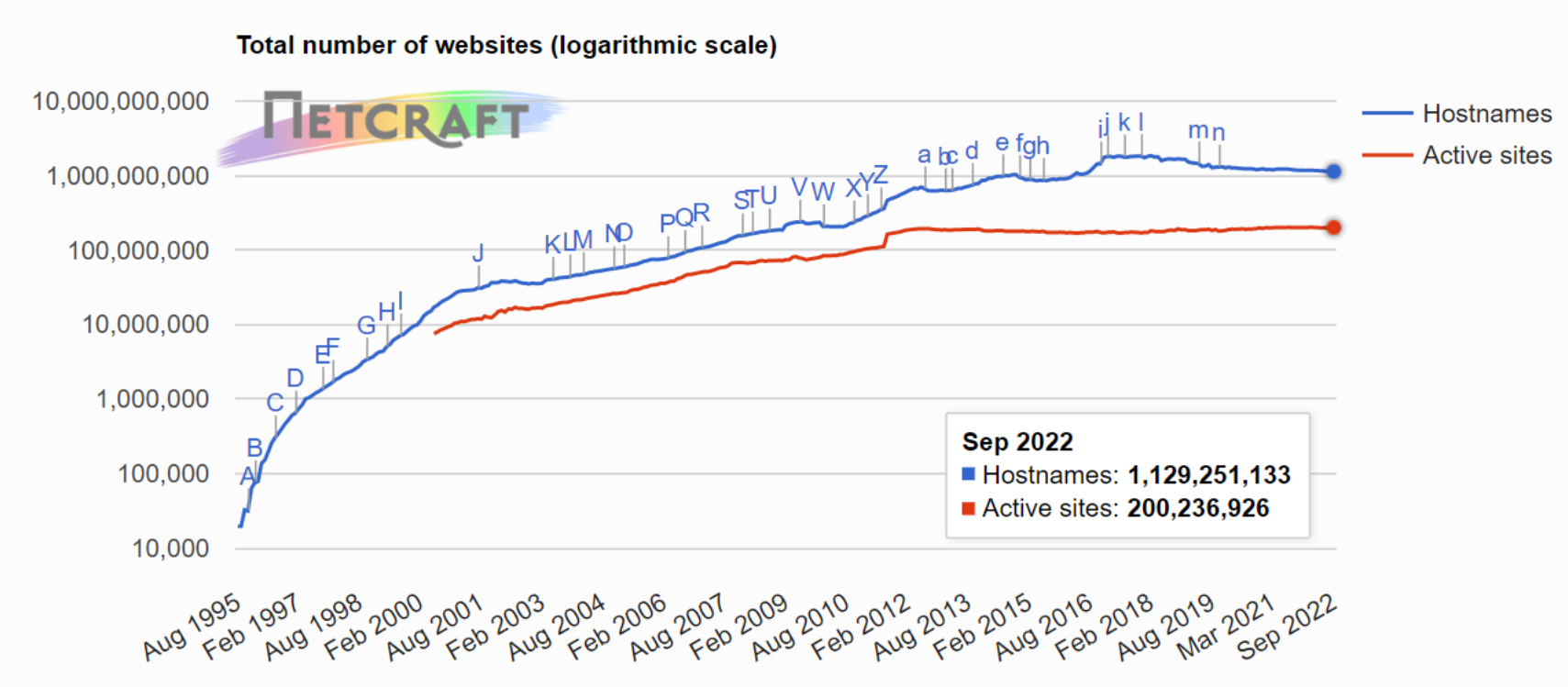
- Architettura software che fornisce accesso a documenti *linkati* distribuiti su migliaia/milioni di macchine

# Importanza del Web

- All'inizio della storia di Internet, il trasferimento di dati FTP contava un terzo di tutto il traffico
- All'inizio del 1990, viene introdotto il Web
  - Crescita esponenziale
- Nel 1995, il traffico Web supera quello FTP
  - Greatest bandwidth consumer
- Nel 2000, il traffico Web supera di gran lunga quello di altri servizi

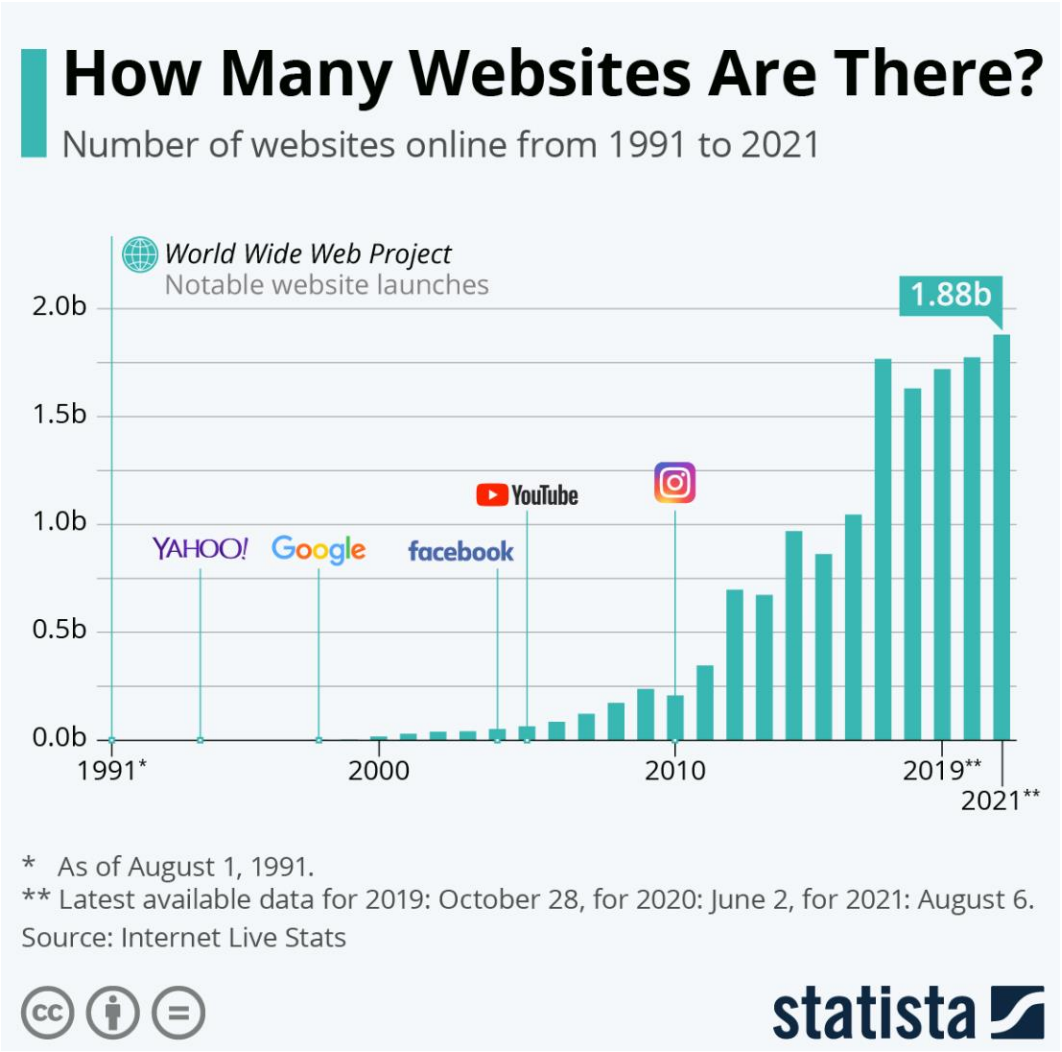
# Alcune Statistiche

- 1991-1997: Explosive growth, at a rate of 850% per year.
- 1998-2001: Rapid growth, at a rate of 150% per year.
- 2002-2006: Maturing growth, at a rate of 25% per year.
- September 2022: 1,129 billions of hostnames (200 millions active sites)



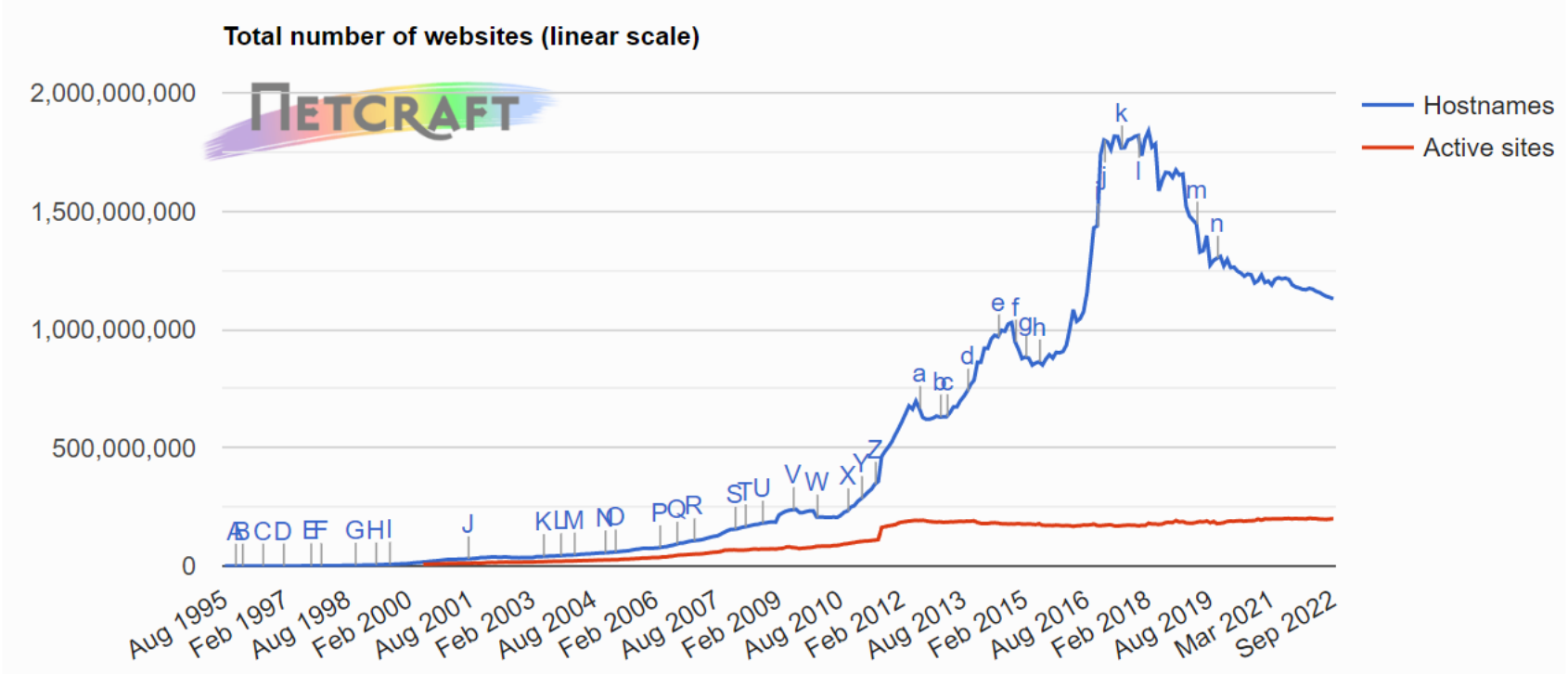
# Alcune Statistiche

https://www.statista.com/chart/19058/number-of-websites-online/



# Alcune Statistiche

- Netcraft (the Internet monitoring company) tracks the growth of the Web and reports every year the number of web sites
- August 1995: 19,705 websites
- September 2022: 1,129 billions websites with domain names and content
- A growth of nearly “make you the rate”% in 20+ years

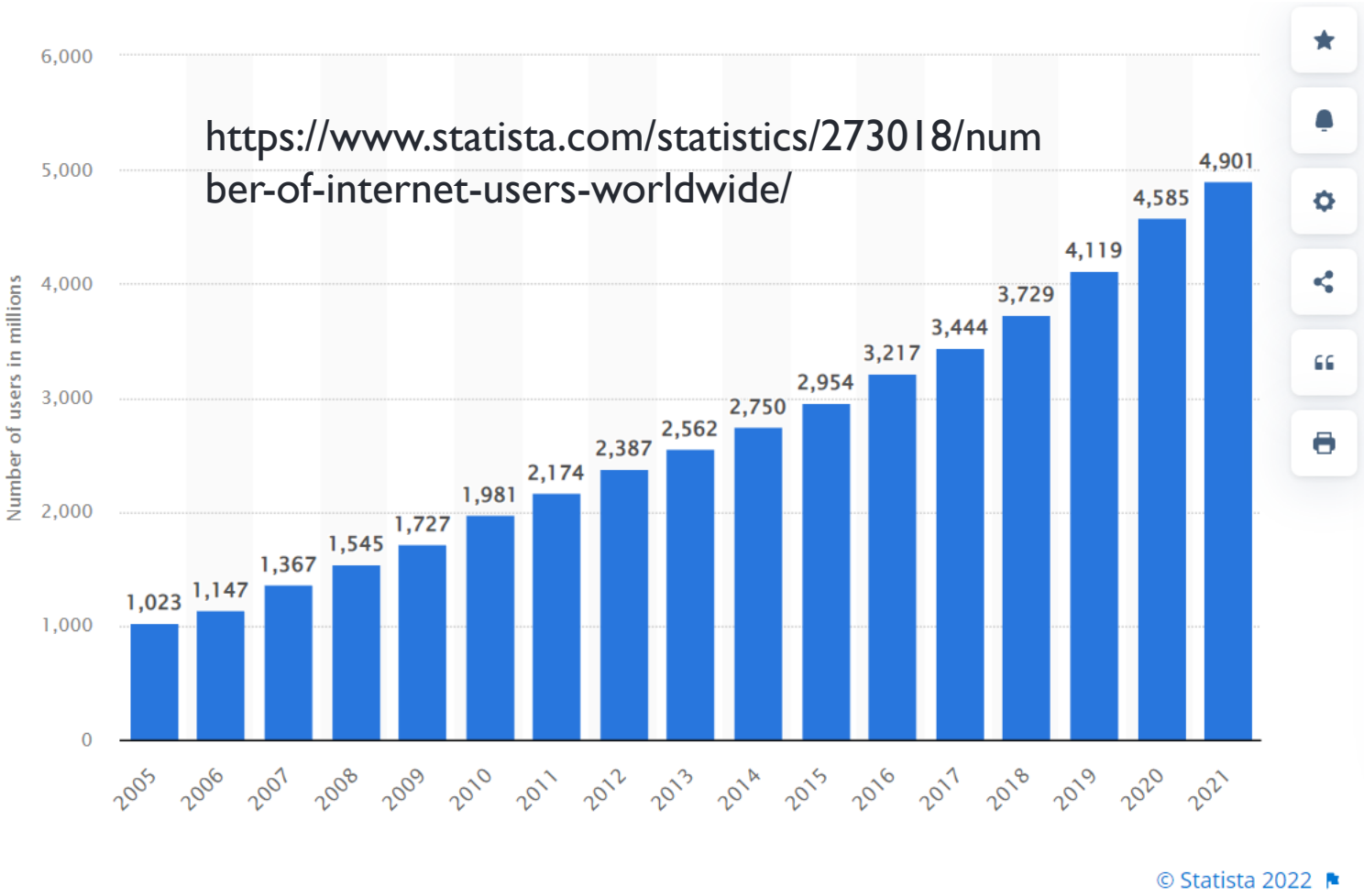


# Alcune Statistiche

- Il Web ha un impatto maggiore delle statistiche
  - Ogni persona nei paesi sviluppati accede al Web
  - WWW è un “diritto dell’uomo”
  - Internet.org tenta di raggiungere tutti

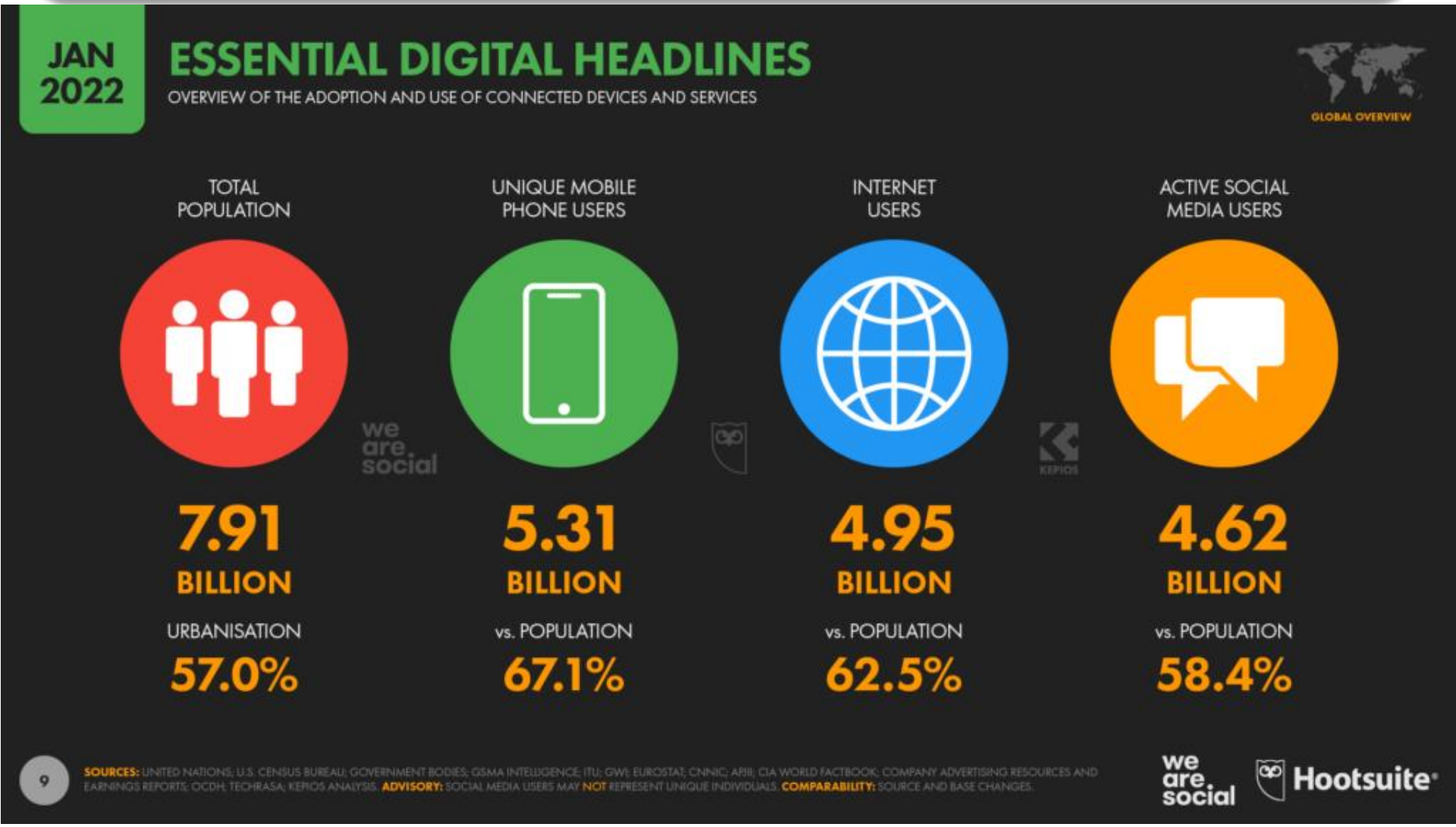
# Alcune Statistiche

- Numero di utenti internet nel mondo



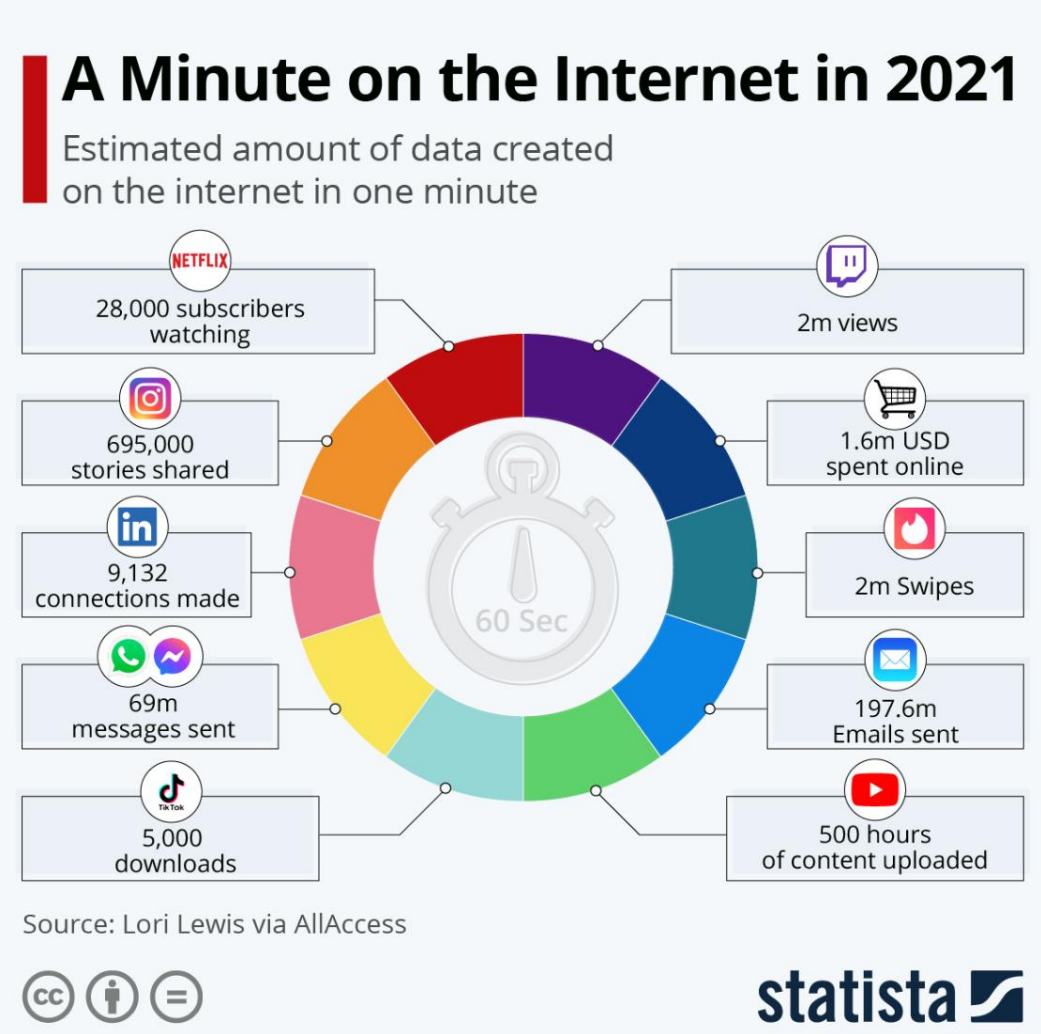
# Alcune Statistiche

https://wearesocial.com/uk/blog/2022/01/digital-2022-another-year-of-bumper-growth-2/



# Alcune Statistiche

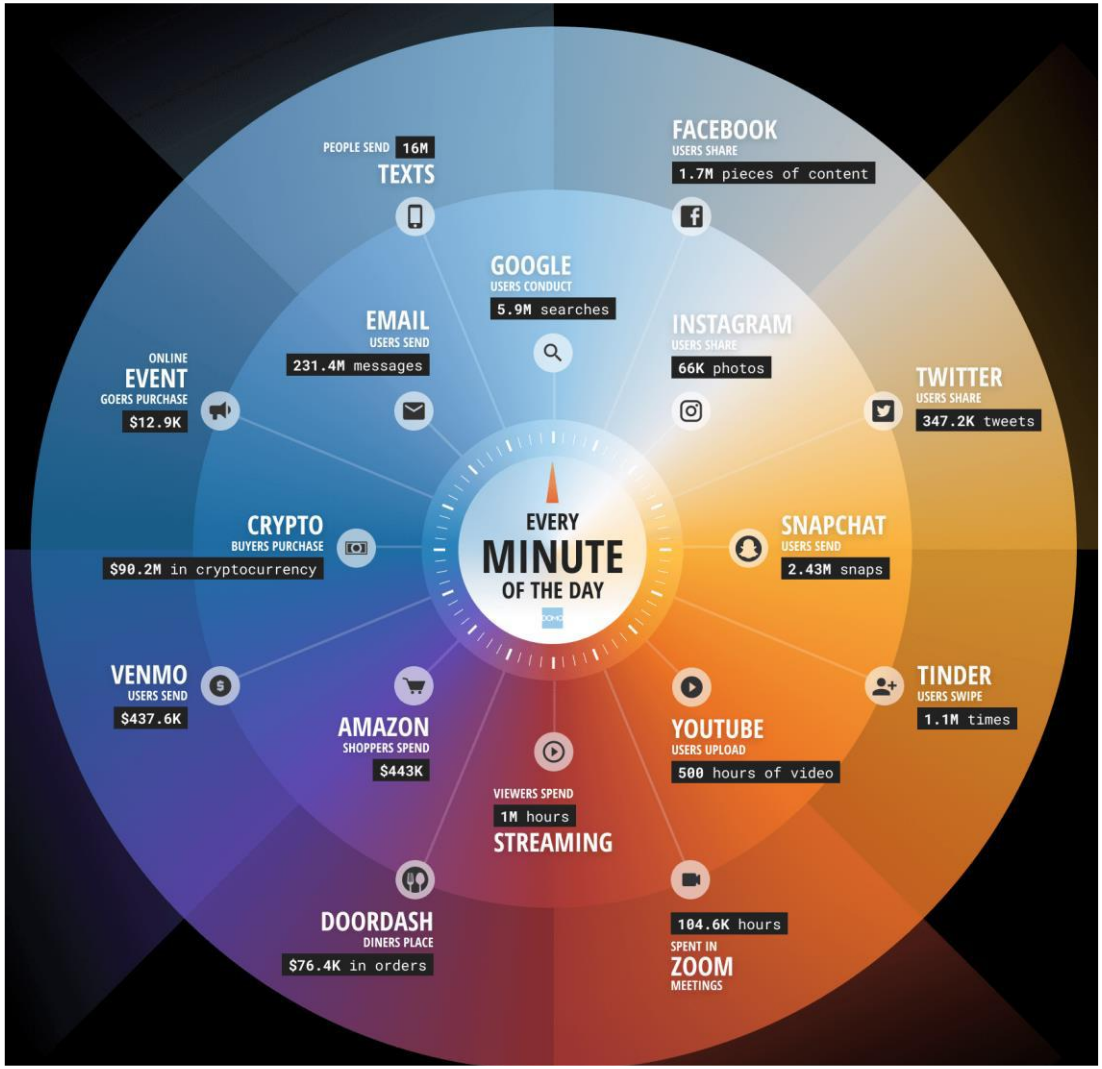
https://www.statista.com/chart/25443/estimated-amount-of-data-created-on-the-internet-in-one-minute/



# Alcune Statistiche

https://www.domo.com/data-never-sleeps#

- Aprile 2022: Internet raggiunge il 63% della popolazione (circa 5 miliardi)
- Quantità di dati creati, collezionati, copiati e consumati raggiungono i 97 zettabyte, 181 zettabyte entro il 2025
- 2.5 quintillion byte creati ogni giorno del 2018 (!)



# Alcune Statistiche

- *La tecnologia web è così importante che erroneamente Internet e web vengono spesso considerati come la stessa tecnologia!*

# World Wide Web (WWW)

# World Wide Web (WWW)

- L’applicazione internet più usata
- Può essere definita come un insieme di ipertesti multimediali distribuiti su molti nodi
- Fornisce accesso a documenti ipertestuali attraverso la rete internet
  - Distribuito e scalabile

# Ipertesto

- Documento con una struttura non sequenziale
- Composto di diverse sezioni linkate
  - Link interni ed esterni
  - Permette agli utenti di ricercare e muoversi tra argomenti collegati
  - Indipendente dall'ordine di presentazione
- C'è più di solo testo (hypermedia)
  - Immagini, musica, video, …

# Prima pagina web

## World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library )
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

# Componenti architetturali

- Paradigma a ipermedia distribuito
  - Iper: link per riferirsi ad altri documenti
  - Media: più di semplice testo
  - Consiste di pagine web (documenti ipermediali)

- Building block del web
  - Web browser: programma applicativo che permette agli utenti di accedere e visualizzare pagine web
  - Web server: entità che fornisce le pagine web al web browser

# Architettura

- Architettura client-server
  - Il **client** (Web browser) è un programma che si connette al server, sottomette una richiesta e aspetta una risposta
  - Il **server** (Web server) è un programma “in ascolto” (su una porta TCP) che fornisce un servizio
- Data una richiesta del client, il server
  - Analizza la richiesta
  - Prepara la risposta
  - Ritorna la risposta al client

# Comunicazione via Web

**Quando l'utente fa clic su un collegamento nel browser, il browser identifica il server Web e invia una richiesta per quella pagina.**

**Computer su cui è installato un browser Web**


**Server Web**

**Il server individua la pagina specificata nelle sue directory/cartelle e invia indietro la pagina al browser per visualizzarla.**

# Concetti base

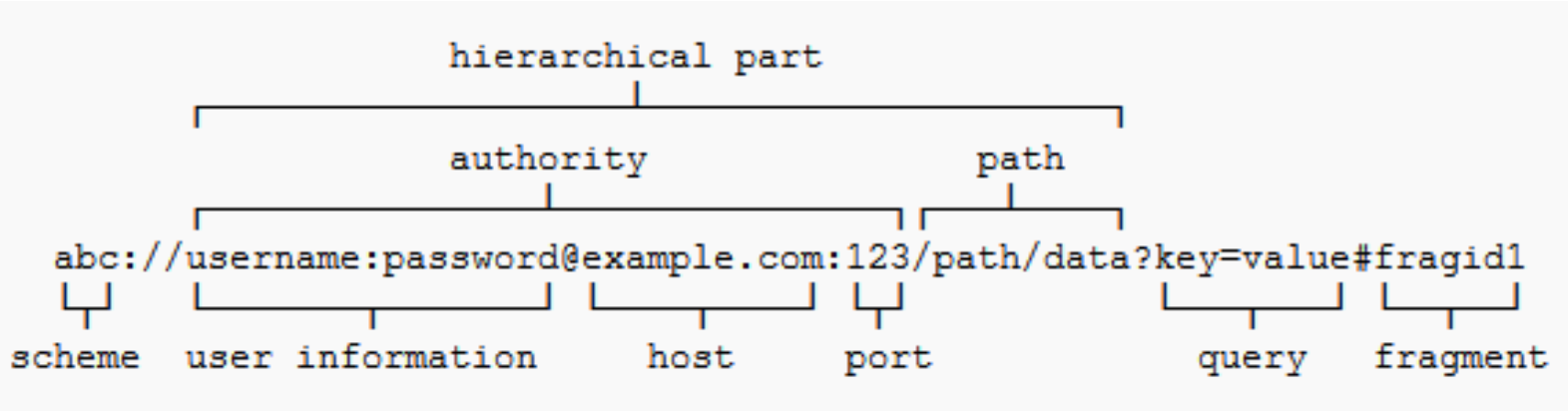
- Uniform Resource Locator (URL)
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)

# Web Page Identifier

- Miliardi di pagine accessibili attraverso il web
- Ogni pagina deve avere un nome univoco
- Uniform Resource Identifier (URI)
  - Sintassi universale con cui identificare le risorse di rete
  - Indipendente dal protocollo
  - Basato su un set di caratteri ristretto, per evitare problemi di cattiva codifica
  - Non definisce la semantica di una risorsa

# Web Page Identifier

- Sintassi URI
  - scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
    - Esempio foo://example.com:8042/over/there?name=ferret#nose



# Web Page Identifier

- Uniform Resource Locator (URL) è un'URI che in aggiunta a identificare una risorsa fornisce anche un meccanismo per localizzarla
  - Sottoinsieme delle URI
- Rispetta la sintassi URI
  - http: *// hostname [: port] / path [? query] (versione semplificata)*
    - Scheme definisce lo schema usato per la comunicazione
    - Il formato dell'URL dipende dallo schema
    - Campi tra parentesi quadre sono opzionali

# Web Page Identifier

- Encoding
  - Protocollo usato (e.g., http)
  - Nome di dominio del server (o indirizzo IP)
  - Numero di porta del protocollo (opzionale)
    - Necessaria solo quando una porta di default non è specificata per il protocollo
  - Percorso attraverso il file system del server
  - Query (opzionali)

# Web Page Identifier

- Di solito gli utenti non vedono i parametri opzionali

- Esempio: http://homes.di.unimi.it/pippo
  - Server è disponibile al computer identificato dal nome di dominio homes.di.unimi.it
  - Il documento si chiama “pippo” o meglio il documento è index.html all’interno della cartella pippo sul server

# Web Page Identifier

- URL può assumere due diversi formati
  - Assoluto
    - Usato dagli utenti del web
    - Contiene una specifica completa dell'URL
    - E.g., homes.di.unimi.it/pippo
  - Relativo
    - Significativo solo quando un server è già stato identificato
    - Omette l'indirizzo del server
    - E.g., /pippo/didattica/

# Web Standards

- Standard separati per
  - Trasferimento (HTTP)
  - Rappresentazione (HTML)

# HTTP

# Accesso al web

- Semplice e intuitivo
  - Utente inserisce un'URL o clicca su un link
  - Il browser parsa l'URL, estrae le informazioni e riceve una copia della pagina
  - Il server riceve la richiesta, seleziona la pagina e la ritorna al browser
  - Il browser parsa la pagina HTML e la mostra all'utente

# Trasferimento

- Protocollo *HyperText Transfer Protocol (HTTP)*
- Si basa su TCP
- Usato tra browser e Web server

# HyperText Transfer Protocol (HTTP)

- Protocollo di comunicazione per lo scambio di messaggi e ipertesti
  - Set-up della connessione
    - Web browser verifica la disponibilità del server
  - HTTP request
  - HTTP response
  - Connection closed

# Caratteristiche

- Livello applicativo
  - Assume un protocollo di trasporto affidabile e orientato alla connessione
  - Non fornisce affidabilità e ritrasmissione
- Paradigma richiesta/risposta
  - La comunicazione inizia con una richiesta HTTP
- Stateless
  - Il server non mantiene lo stato della connessione

# Caratteristiche

- Permette trasferimenti bi-direzionali (ad es., form)
- Negoziazioni delle capability
  - Il mittente specifica le sue capability e il ricevente le accetta tutte o in parte (ad es., negoziazione del character set)
- Supporto per caching
  - Diminuisce il tempo di risposta
  - Per richieste alla stessa pagina, il browser determina se la pagina è cambiata
- Supporta intermediari nella comunicazione
  - Proxy server

# Caratteristiche (Sommario)

- Protocollo a livello applicativo per sistemi informativi ipermediali, collaborativi e distribuiti
  - Orientato agli oggetti, generico, stateless
  - Può essere usato per molte applicazioni come i sistemi distribuiti di gestione degli oggetti
  - Protocollo principale del Web
  - Uno dei pochi protocolli applicativi per cui è garantita la connettività tra qualsiasi coppia di punti su Internet (firewall traversal)

# Connessioni HTTP

- Browser invia la richiesta a cui il server risponde
  - Accede all' URL
  - Estrae l'hostname
  - Contatta il DNS per ottenere l'indirizzo IP
  - Crea una connessione TCP con il server
  - Una volta che la connessione TCP è pronta, HTTP è usato per comunicare
  - Il browser invia una query e il serve la soddisfa

# HTTP/1

# Operazioni HTTP

- HTTP/1.0
- GET
- POST
- HEAD
  - Per avere solo INFO sull'oggetto e non l'oggetto stesso (ad esempio sulla data di ultima modifica). Utile per il caching

- HTTP/1.1
- GET, POST, HEAD
- PUT
  - Upload un file
- DELETE
  - Cancella un file

# Operazioni HTTP

- GET
  - HTTP GET usata per ottenere un documento
- HEAD
  - Simile a GET, ma richiede solo le informazioni dell'header
  - Utile per controllare se un documento esiste e quanto è recente
- POST
  - Simile a GET, ma codifica gli input in modo diverso
  - Utile per proporre i contenuti dei form a un programma CGI
- PUT
  - Trasferisce un documento al server
  - Presente a partire dalla versione HTTP/1.1
- DELETE
  - Elimina un documento dal server
  - Presente a partire dalla versione HTTP/1.1

# Operazioni HTTP

- Richiesta tipica: GET per ottenere un documento
  - GET URL HTTP/version number
- Esempio
  - GET http://www.cs.purdue.edu/people/comer/ HTTP/1.0
- URL relativa è permessa
  - GET  /people/comer/  HTTP/1.1

# Header HTTP

- HTTP header hanno la stessa sintassi degli header delle mail (RFC 822 e MIME extension)
  - *Header – linea vuota – oggetto da mandare*
  - Header: linee di testo con formato “keyword: information”

| Header | Type | Contents |
|---|---|---|
| User-Agent | Request | Information about the browser and its platform |
| Accept | Request | The type of pages the client can handle |
| Accept-Charset | Request | The character sets that are acceptable to the client |
| Accept-Encoding | Request | The page encodings the client can handle |
| Accept-Language | Request | The natural languages the client can handle |
| Host | Request | The server's DNS name |
| Authorization | Request | A list of the client's credentials |
| Cookie | Request | Sends a previously set cookie back to the server |
| Date | Both | Date and time the message was sent |
| Upgrade | Both | The protocol the sender wants to switch to |
| Server | Response | Information about the server |
| Content-Encoding | Response | How the content is encoded (e.g., gzip) |
| Content-Language | Response | The natural language used in the page |
| Content-Length | Response | The page's length in bytes |
| Content-Type | Response | The page's MIME type |
| Last-Modified | Response | Time and date the page was last changed |
| Location | Response | A command to the client to send its request elsewhere |
| Accept-Ranges | Response | The server will accept byte range requests |
| Set-Cookie | Response | The server wants the client to save a cookie |

# Esempio di trasferimento HTTP

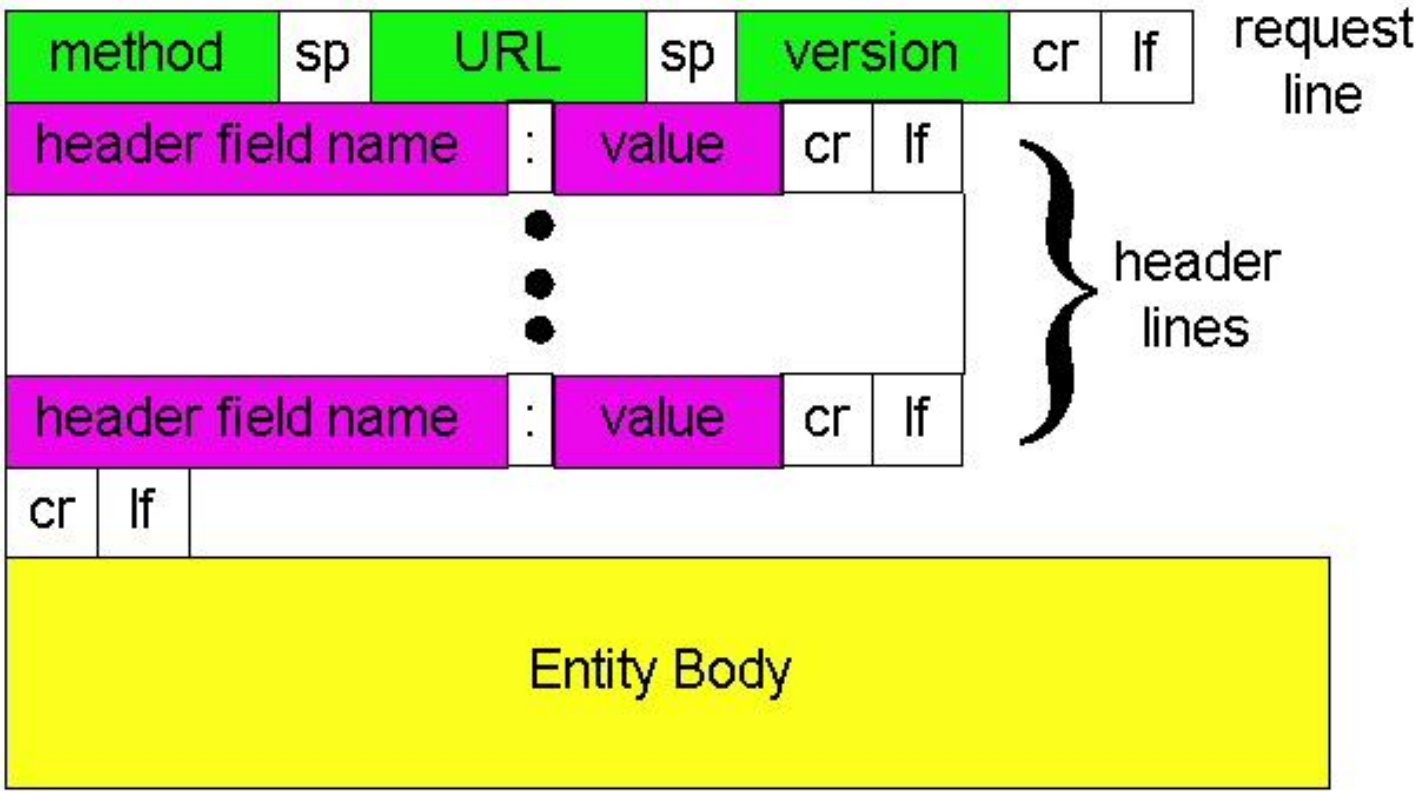
Content-Length: 34

Content-Language: english

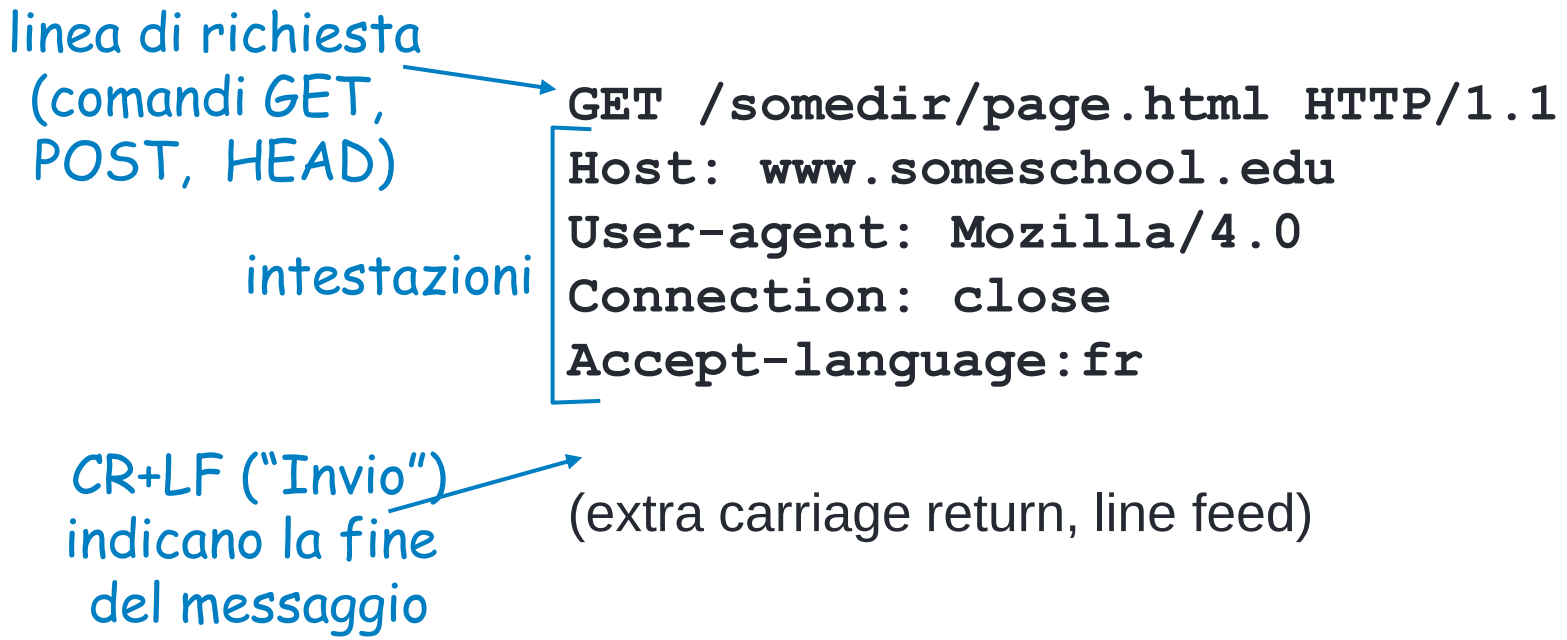
Content-Encoding: ascii

<HTML> A trivial example. </HTML>

# Formato generale richiesta



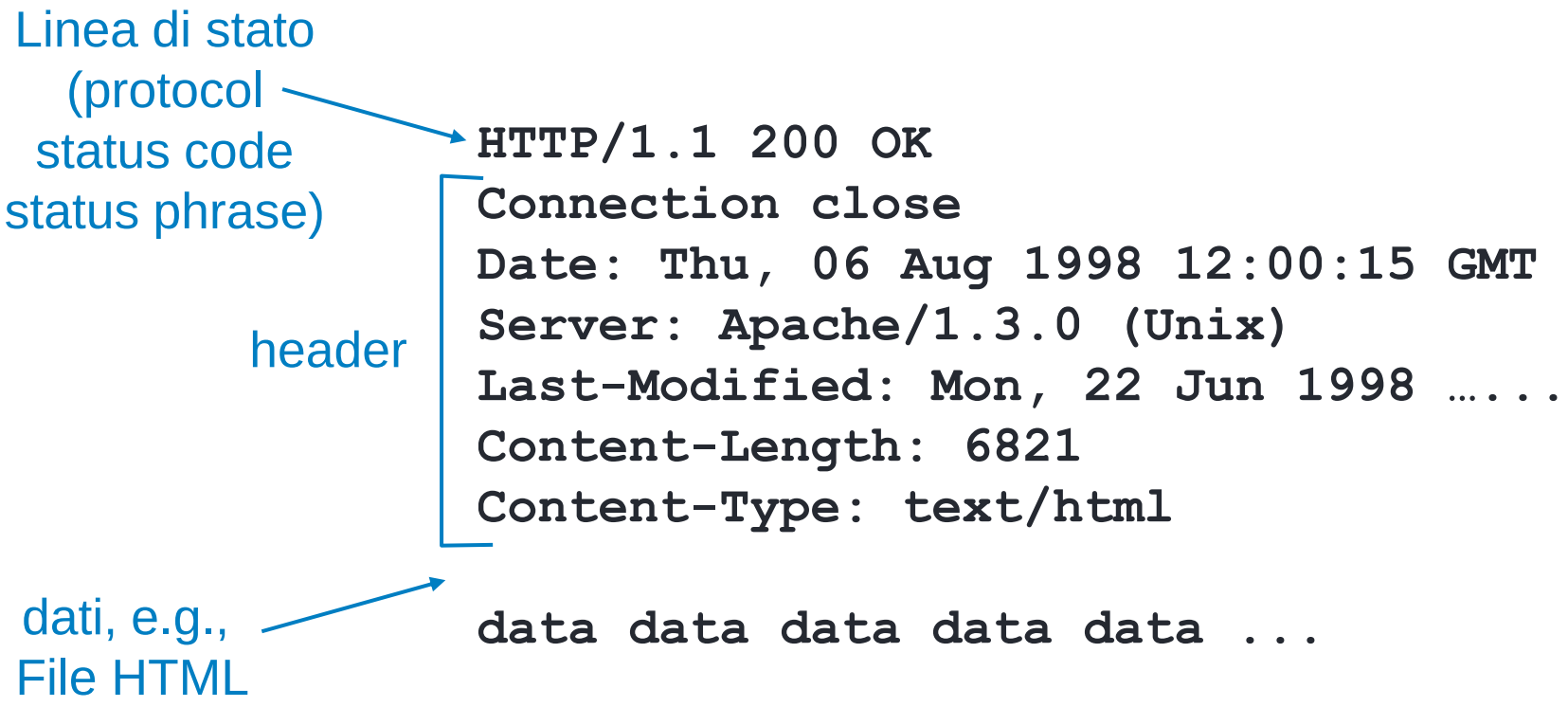
# Formato del messaggio di richiesta HTTP



# Campi dell'header di richiesta

- Il client può specificare ulteriori informazioni nella richiesta
  - User-Agent: Indica la versione del browser
  - Referer: Comunica al server da dove proviene l'utente (utile per eseguire il log e per tenere traccia dell'utente)
  - From: Contiene l'indirizzo e-mail dell'utente (generalmente non usato per motivi di privacy)
  - Authorization: può inviare il nome utente e la password (usato con documenti che richiedono l'autorizzazione)
  - If-Modified-Since: invia il documento solo se è più recente della data specificata (usato per memorizzare nella cache)

# Formato del messaggio di risposta HTTP



# Campi dell'header di risposta

- La prima riga della risposta del server contiene un codice di stato

| | |
|---|---|
| **OK 200** | La richiesta ha avuto successo |
| **Moved 301** | Il documento è stato spostato permanentemente |
| **Not modified 304** | La versione nella cache è aggiornata |
| **Bad request 400** | Errore di sintassi nella richiesta del client |
| **Forbidden 403** | Al cliente non è consentito l'accesso (cioè è protetto) |
| **Not found 404** | Il file non è stato trovato |
| **Internal error 500** | Il server ha incontrato una condizione inaspettata |
| **Service unavailable 503** | Il server è sovraccarico |

# Altri campi dell'header di risposta

- Oltre al codice di stato, la risposta del server può includere
  - Date: Tempo di risposta, ora GMT
  - Server: Informazioni d'identificazione sul server
  - Last-modified: Ora in cui è stato modificato per l'ultima volta, ora GMT
  - Content-length: Dimensione del documento in byte
  - Content-type: Formato file (ad esempio, html, gif, pdf)
  - Expires: Evita al browser di memorizzare la pagina nella cache oltre la data di scadenza

# Richiesta GET

- Il server Web riceve solo i contenuti del messaggio di richiesta GET

- Di solito il messaggio GET viene generato automaticamente dal browser quando si seleziona un URL, ma
  - Può venire da un controllore di collegamenti, da un robot di motori di ricerca, ecc.
  - Può venire direttamente da una connessione telnet che usa la porta 80
  - GET URI HTTP/1.1
    Host: hostname

# Esempio di GET

```
bash-2.05b$ telnet
www.myserver.unimi.it 80
Trying 10.0.64.233...
Connected to www.myserver.unimi.it
Escape character is '^]'.
GET /~avf/index.html HTTP/1.1
Host: www.myserver.unimi.it
```

La risposta del server riporta le informazioni dell'header, seguite dalla pagina.

```
HTTP/1.1 200 OK
Date: Tue, 06 Jan 2006 11:45:09 GMT
Server: HP Apache-based Web Server/1.3.27 (Unix) mod_perl/1.27
     PHP/4.2.2
Last-Modified: Tue, 30 Dec 2005 12:37:03 GMT
ETag: "10b3e-1000-431452ef"
Accept-Ranges: bytes
Content-Length: 4096
Content-Type: text/html


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.or g/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
     lang="en">
<head>
…
</head>
<body>


 <h1>Sample homepage</h1>


…
</body>
</html>


Connection closed by foreign host.
```

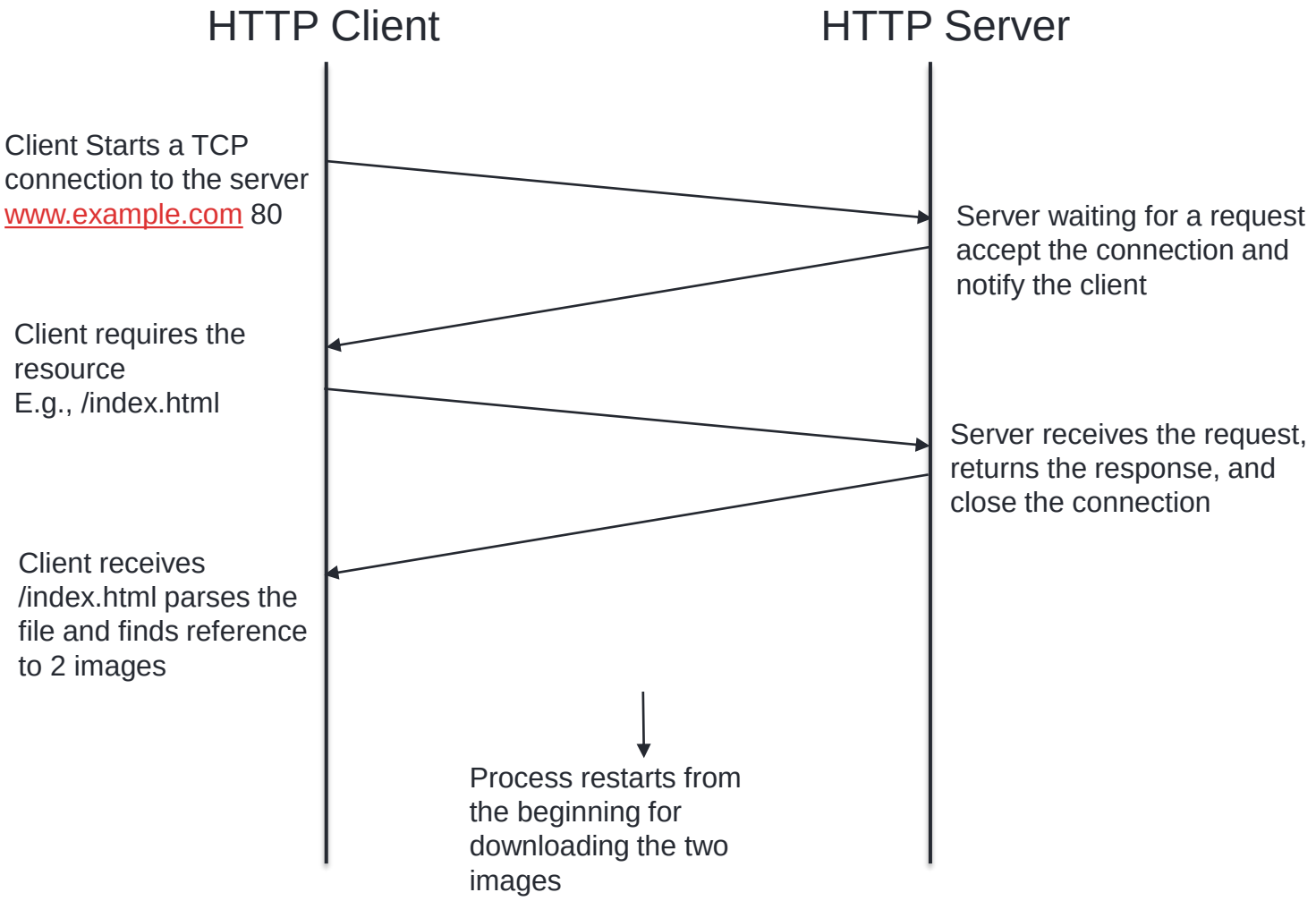
# Connessioni persistenti vs non persistenti

- Connessioni non persistenti
  - Una connessione TCP per ogni file
  - HTTP/1.0
- Connessioni persistenti
  - Più trasferimenti viaggiano su una stessa connessione TCP
  - HTTP/1.1

# Connessioni non persistenti

- HTTP versione 1.0 usa una connessione TCP per ogni trasferimento
  - Il browser crea la connessione TCP al server
  - Il browser invia una richiesta
  - Il server ritorna l'header che la descrive
  - Il server ritorna l'oggetto
  - Il server chiude la connessione
    - Client legge dati fino a quando un end of file è trovato

# Connessioni non persistenti



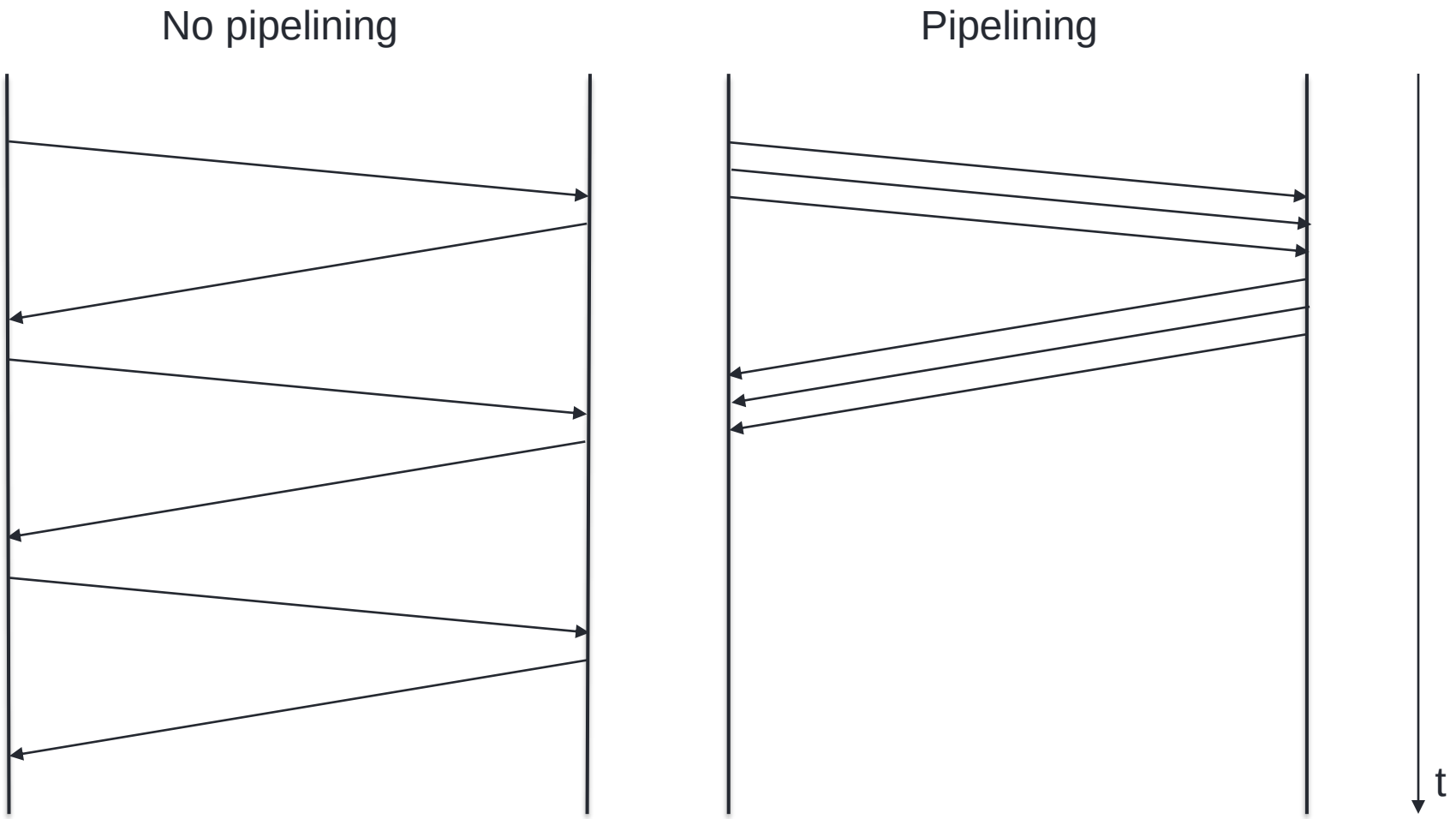
# Connessioni persistenti

- HTTP versione 1.1 (RFC Giugno 1999) introduce un'inversion di tendenza
- HTTP versione 1.1 permette alle connessioni TCP di persistere tra diverse richieste
  - Un client apre una connessione TCP con un server
  - Il client usa la connessione per richieste e risposte multiple
  - Il client o il server chiudono la connessione

# Connessioni persistenti

- Vantaggi
  - Overhead ridotto
  - Meno connessioni TCP: latenza della risposta più bassa, meno memoria per i buffer, e minor tempo di CPU usato
  - Richieste vengono messe in pipeline, le richieste vengono mandate una dopo l'altra senza aspettare la risposta
- Svantaggi
  - Bisogna identificare inizio e fine di ogni oggetto inviato sulla rete Internet
    - Opzione 1: mandare una lunghezza seguita da un oggetto
    - Opzione 2: inviare un valore sentinella che identifica la fine di un oggetto

# Connessioni persistenti



# Gestione della persistenza

- Pagine web dinamiche non permetto di calcolare la dimensione della pagina in anticipo
- Salvare i dati su un file per calcolarne la dimensione è poco pratico
  - Alto consumo di risorse e ritardo
- Server chiude la connessione dopo aver mandato i dati
  - Come alternativa si può usare l'HTTP chunking

# HTTP Chunking

- Una soluzione alla persistenza nel web dinamico
- HTTP 1.1 supporta chunked encoding
  - Aggiunto un header speciale (Transfer-Encoding) che permette di spezzare la risposta in parti diverse
- Ogni scrittura sulla connessione è misurata e un chunk di lunghezza zero rappresenta la fine della transazione

# HTTP Chunking

- Chunked encoding è utile quando grandi quantità di dati sono inviate al cliente senza conoscerne la dimensione totale
- Dimensione totale conosciuta solo quando la richiesta è stata completemente processata
- Ad esempio, generazione di una grande tabella HTML che deriva da una query a un database o invio di grandi immagini

# HTTP Chunking: Esempio

HTTP/1.1 200 OK

Date: Mon, 29 Nov 2004 01:45:36 GMT

Transfer-Encoding: chunked

Content-Type: text/html

1000

[some bytes here]

230

[some bytes here]

0

# Virtual Hosting

- Virtual hosting permette di dare accesso a diversi siti web memorizzati sulla stessa macchina server
  - Possibile perché il browser manda il nome del sito a cui connettersi come parte integrante della richiesta

- Due tipi
  - Name-based
  - IP-based

# Virtual Hosting: Name-based

- Con header HTTP/1.0 hostname non era obbligatorio
  - Alcuni browser come Internet Explorer non includevano hostname
  - Richiesta non veniva soddisfatta
- HTTP/1.1 richiede hostname come parte integrante degli header della richiesta
  - HTTP header “host:”
    - Confrontato con il nome o alias del server (slide successiva)
- Inizialmente SSL non funzionava con name-based hosting
  - Hostname non era incluso inizialmente nell’handshake TLS/SSL
  - Impossibile identificare il certificate da usare per ogni sito
  - Ora la funzione Server Name Indication usata per passare il server hostname di interesse nell’SSL handshake message

# Virtual Hosting: Name-based

```
<VirtualHost *:80>
    # This first-listed virtual host is also the default for *:80
    ServerName www.example.com
    ServerAlias example.com
    DocumentRoot "/www/domain"
</VirtualHost>

<VirtualHost *:80>
    ServerName other.example.com
    DocumentRoot "/www/otherdomain"
</VirtualHost>
```

# Virtual Hosting: IP-based

- Richiede un indirizzo IP separato per ogni hostname su web server
- IP-based hosting funziona con tutte le implementazioni SSL
- IP-based hosting funziona anche senza DNS
- Richiede un indirizzo IP per ogni sito web
  - Potrebbe non essere possibile e difficile da implementare

# Virtual Hosting: IP-based

```
<VirtualHost 172.20.30.40:80>
    ServerAdmin webmaster@www1.example.com
    DocumentRoot "/www/vhosts/www1"
    ServerName www1.example.com
    ErrorLog "/www/logs/www1/error_log"
    CustomLog "/www/logs/www1/access_log" combined
</VirtualHost>

<VirtualHost 172.20.30.50:80>
    ServerAdmin webmaster@www2.example.org
    DocumentRoot "/www/vhosts/www2"
    ServerName www2.example.org
    ErrorLog "/www/logs/www2/error_log"
    CustomLog "/www/logs/www2/access_log" combined
</VirtualHost>
```

# Virtual Hosting: Name-based vs IP-based

- IP-based virtual hosting usa indirizzo IP della connessione per determinare il virtual host a cui inoltrare la richiesta
  - Richiede un indirizzo IP per ogni host (costoso)

- Name-based virtual hosting richiede al client di specificare hostname come parte degli header HTTP
  - Diversi host condividono lo stesso indirizzo IP
  - Più semplice: necessità di configurare il DNS server per mappare hostname verso l'indirizzo IP corretto e configurare l'Apache HTTP Server per riconoscere i diversi hostname
  - Richiesta minore di indirizzi IP

# Negoziazione

- HTTP header usati per negoziare capability
- Capability riguardano
  - Connessione (ad es., autenticazione)
  - Rappresentazione (ad es., JPEG è accettabile, quale tipo di compressione)
  - Contenuto (ad es., linguaggio)
  - Controllo (e.g., validità della pagina)

# Negoziazione

- Guidata dal server
  - Negoziazione inizia con una richiesta del browser
  - La richiesta contiene URL e capability
  - Il server seleziona la capability che soddisfa le sue preferenze o la migliore (se molte si applicano)
- Guidata dal client
  - Una richiesta del browser chiede cosa è disponibile
  - Il server ritorna un set di possibilità
  - Il browser selezioni una che soddisfa le sue preferenze e ritorna la richiesta per un oggetto
  - Vantaggio: browser ha il controllo
  - Svantaggio: due interazioni

# Negoziazione

- Il browser può specificare rappresentazioni che sono accettabili con un valore di preferenza per ognuna

- Esempio: Accept header
  - Accept: text/html, text/plain; q=0.5, text/x-dvi; q=0.8
  - q è il livello di preferenza

# Oggetti per la negoziazione

- Esistono diversi *Accept*- header

- Corrispondono ai *Content*- headers
  - Accept-Encoding:
  - Accept-Charset:
  - Accept-Language:

# Proxy server

- Forniscono un’ottimizzazione con carico e latenza ridotti
- Il browser può essere configurato per contattare un proxy
- Il proxy configurator per mantenere una cache di pagine web
  - Permette il caching a livello di organizzazione

# Proxy server

- HTTP ha supporto esplicito per proxy
  - Regola come i proxy gestiscono richieste, interpretano gli header e interagiscono con browser e server
  - Diversi header HTTP sono definiti per i proxy (ad es., per autenticarsi al server)

- Il server può specificare il massimo numero di proxy lungo un percorso (incluso nessun proxy)
  - Max-Forwards: 1

# Caching di pagine web

- Essenziale per garantire efficienza
  - Traffico e latenza ridotti
  - Pagine web memorizzate su disco
  - Pagine web accedute dal disco

- PROBLEMA: stabilire la data di scadenza per la pagina

# Caching di pagine web

- Il server specifica
  - Se la pagina può essere messa in cache
  - Il tempo massimo per cui la pagina può essere tenuta in cache
  - Molto altro

- Il browser può specificare l'età massima di una pagina (ad es., forza la rivalidazione dei documenti in cache)
  - Età massima uguale a zero
  - Ogni volta la pagina viene richiesta al server

# Memorizzazione nella cache

- I browser memorizzano le pagine nella cache per risparmiare banda
  - Mantengono una memorizzazione temporanea (cache) per le pagine recenti
  - Quando è richiesta una pagina, il browser controlla per vedere se è già nella cache
  - Se non è nella cache, genera la richiesta GET
  - Quando arriva il messaggio di risposta, il browser visualizza la pagina e la memorizza nella cache (insieme ad informazioni dell'header)

# Memorizzazione nella cache

- La cache è controllata dai response header
  - Cache-control
  - Last-modified
  - Expires

# Memorizzazione nella cache

- Cache control
  - No-store: indica che il contenuto non può essere memorizzato in cache
  - No-cache: indica che il contenuto potrà essere inserito nella cache e potrà essere riusato solo dopo una rivalidazione (if-modified-since)
  - Private: il contenuto può essere memorizzato solo per l'utente che ha fatto la richiesta
  - Public: il contenuto è memorizzato sia nelle cache pubbliche che private
  - Se assente qualsiasi cache può memorizzare la pagina

# Memorizzazione nella cache

- Last modified
  - Permette di sapere l'ultima modifica della pagina che è stata acceduta
  - Mantenuto nella cache e usato per controllare se la pagina deve essere ritrasferita
- Expires
  - Data di scadenza
  - Il browser riutilizza la pagina fino alla scadenza senza contattare il server

# Memorizzazione nella cache

- Se la pagina è già presente nella cache, il browser invia la richiesta GET con l'header `If-Modified-Since` relativo ai dati della pagina
- Quando arriva il messaggio di risposta
  - se il codice di stato è 200, il browser visualizza la pagina e la memorizza nella cache
  - se il codice di stato è 304, visualizza la versione memorizzata nella cache

# Richieste condizionali

- La richiesta contiene un header con alcune condizioni
- Se le condizioni non sono soddisfatte, il server non ritorna l'oggetto richiesto

# Conditional Request

- Permette al browser di controllare la copia in cache per vedere se è valida
  - Elimina latenza inutile
  - Invia *If-Modified-Since* nell'header di richieste GET
  - La pagina Web è trasferita solo se aggiornata

- Esempio
  - If-Modified-Since: Wed, 31 Dec 2003 05:00:01 GMT

# GET condizionale

```
bash-2.05b$ telnet www.myserver.unimi.it 80
Trying 10.0.64.233...
Connected to www.myserver.unimi.it.
Escape character is '^]'.
GET /~avf/ HTTP/1.1
Host: www.myserver.unimi.it
If-Modified-Since: Tue, 30 Aug 2005 14:00:00 GMT
```

```
HTTP/1.1 304 Not Modified
Date: Tue, 06 Jan 2006 14:08:58 GMT
Server: HP Apache-based Web Server/1.3.27
   (Unix) mod_perl/1.27 PHP/4.2.2
ETag: "10b3e-1000-431452ef"

Connection closed by foreign host.
```

Poiché il documento non è stato modificato dopo la data specificata, la pagina NON viene inviata dal server (codice di stato 304)

# HTTP/2

# HTTP 1.1: Stato

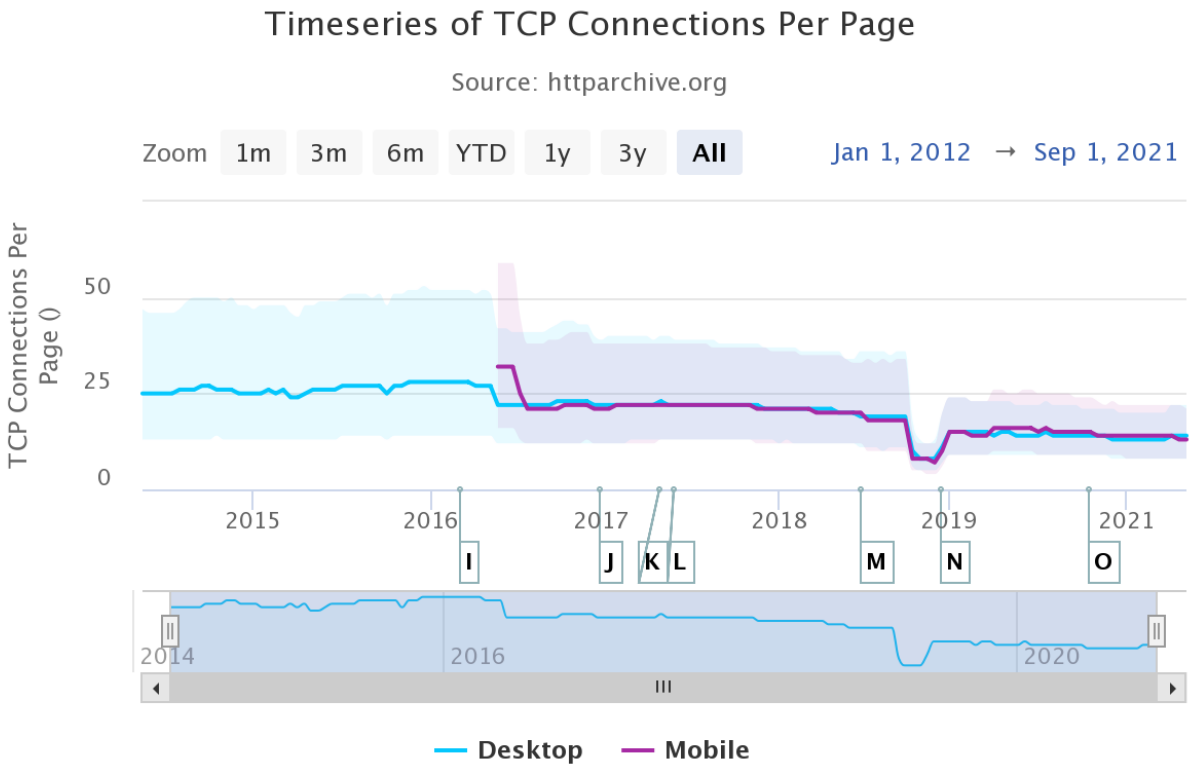
- Gigantesco
  - HTTP1.1 (RFC 2626) rilasciato nel 99 – 176 pagine
  - Diviso in 6 documenti – RFC 7230 e famiglia
- Implementazione dell'RFC
  - Ad esempio, pipeling:  molti server HTTP/1.1 non supportano correttamente il pipelining, costringendo la maggior parte dei client HTTP a non utilizzare il pipelining HTTP in pratica
- Uso di TCP non adeguato con grande latenza e aumento esponenziale del traffico
  - Da gennaio 2012 a dicembre 2016 traffico aumenta di circa +300% (httparchive.org)

# HTTP 1.1: Problemi

- Concurrent connection limit
- Head of line blocking
- TCP slow start
- Latenza – page load time

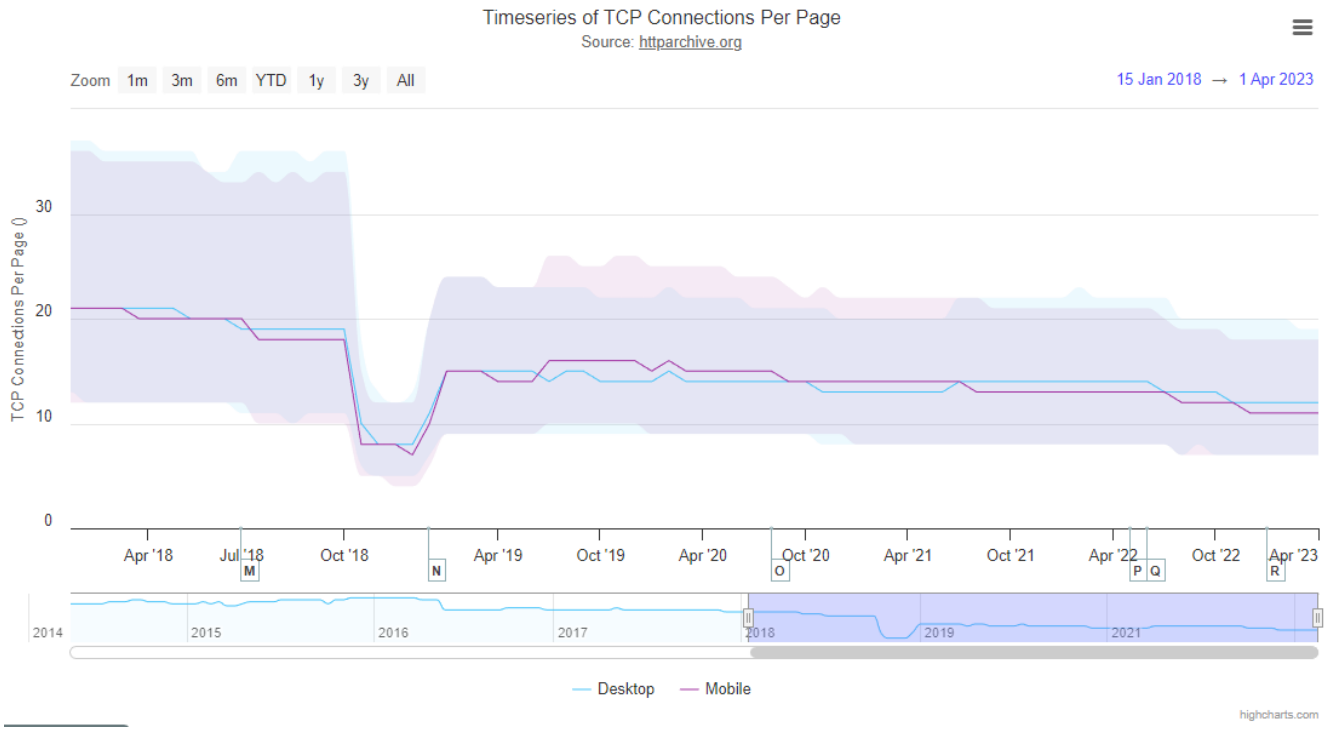
# HTTP 1.1: Concurrent connection limit

- Concurrent connection limit
  - RFC2626: «Clients that use persistent connections should limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. [..] these guidelines are intended to improve HTTP response times and avoid congestion.»
  - Alcuni browser hanno aumentato il limite a 6/8 connessioni contemporanee (Microsoft, Google, FireFox, Opera, etc)



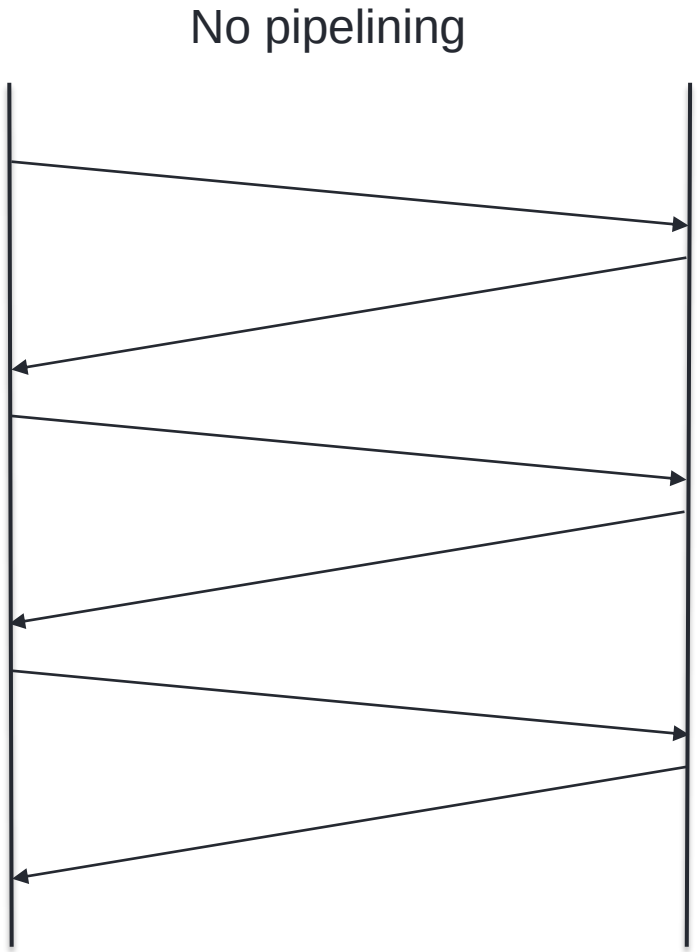
# HTTP 1.1: Concurrent connection limit

- Concurrent connection limit
  - RFC2626: «Clients that use persistent connections should limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. [..] these guidelines are intended to improve HTTP response times and avoid congestion.»
  - Alcuni browser hanno aumentato il limite a 6/8 connessioni contemporanee (Microsoft, Google, FireFox, Opera, etc)



# HTTP 1.1: Head of line blocking

- No pipelining
- Ogni connessione gestisce una richiesta alla volta
- Se la richiesta si blocca, tutte le altre aspettano
- HTTP/1.1 definisce il pipelining ma molti non lo applicano

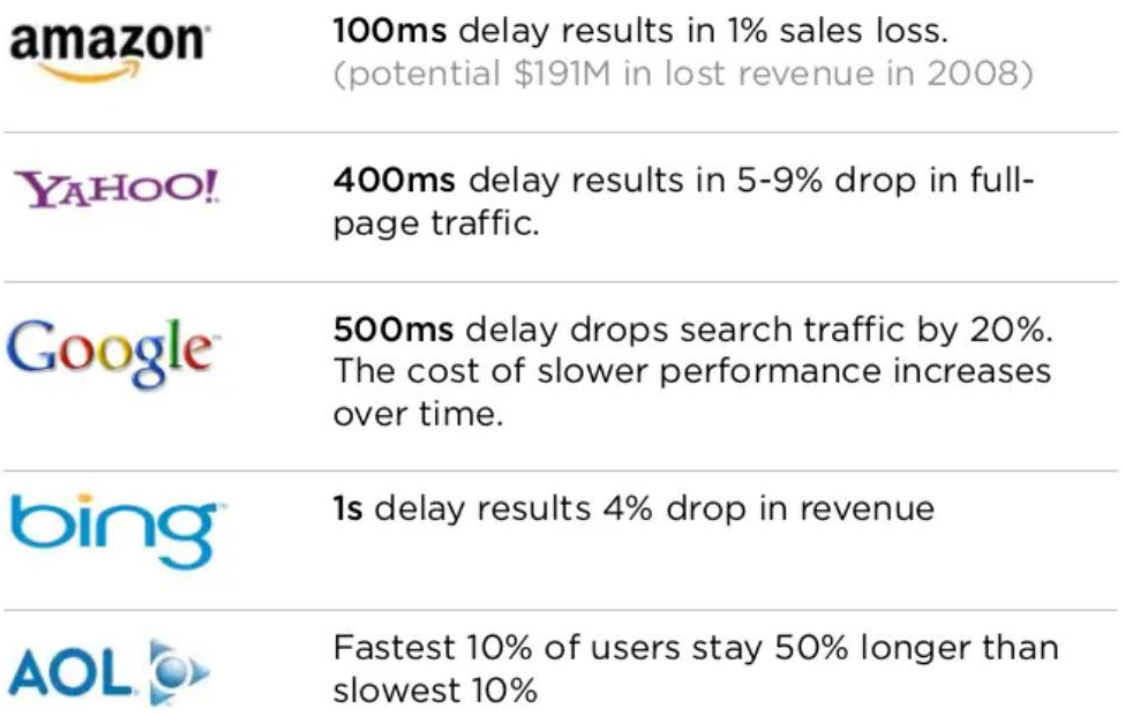


# HTTP 1.1: TCP slow start

- TCP non usa la piena capacità della banda fin da subito
- TCP chiede alla rete per trovare la capacità disponibile
- TCP Slow start è un algoritmo che bilancia la velocità di una connessione di rete
  - Aumenta gradualmente la quantità di dati trasmessi fino a trovare la capacità di trasporto massima della rete

# HTTP 1.1: Latenza – page load time

- L'impatto maggiore per il download di una pagina non è la banda ma la latenza




https://www.slideshare.net/pob1970/mobile-first-lukew/41-100ms_delay_results_in_1

# HTTP 1.1: Scorciatoie

- Riduzione del numero di richieste e della latenza
  - Concatenation e Inlining
    - Unire file di script/stile in una singola risorsa
    - Integrare JS e CSS all'interno di una stessa pagina
    - Immagini in linea in CSS e HTML usando URIs (encoding base 64)
  - Spriting: "The technique of storing multiple images in a single larger image so that they load more quickly into a webpage."
    - Anche se la dimensione è maggiore, abbiamo benefici in termini di richieste HTTP

# HTTP/2: SPDY

- Proposto da google come estensione di HTTP nel 2011
- Implementa tutte le feature principali di HTTP/2
- Testbed per testare i miglioramenti di HTTP senza la complessità di una standardizzazione
- Diventa la base di HTTP/2 nel 2012
- Chrome ferma l'utilizzo di SPDY dal 2016

# HTTP/2 in a nutshell

- Consente di utilizzare in modo più efficiente le risorse di rete e di ridurre la percezione della latenza
  - Introduce compressione del campo header e consente più scambi simultanei sulla stessa connessione
- Pipelining, “head of line blocking”, connessione singola
  - Consente di intervallare messaggi di richiesta e di risposta sulla stessa connessione e utilizza un codifica efficiente per i campi di intestazione HTTP
  - Consente inoltre di prioritizzare le richieste, permettendo di completare più rapidamente le richieste più importanti, migliorando ulteriormente le prestazioni
- Mantiene la semantica di HTTP/1.1
  - *Metodi, codici di stato, campi degli header, URI*

# HTTP/2

- Risultato
  - Meno connessioni TCP in confronto a HTTP/1.x
  - Meno concorrenza con altri flussi di dati e connessioni a lungo termine
  - Migliore utilizzazione della capacità di rete disponibile
  - Elaborazione più efficiente dei messaggi tramite l'uso di framing dei messaggi binari

# HTTP/2: Specifiche

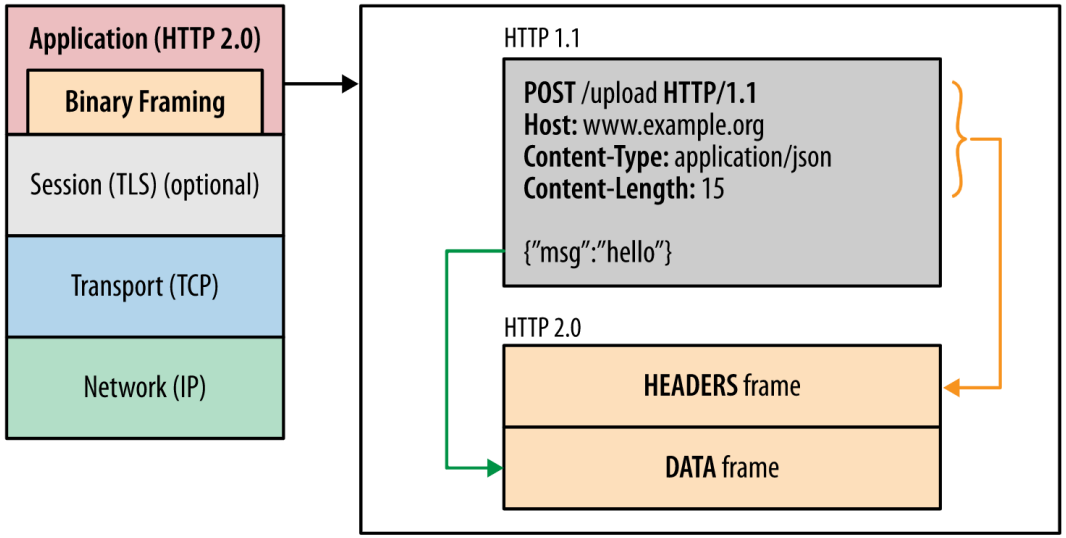
- HTTP/2 – RFC 7540
- HPACK (header compression) – RFC7541
- Implementazioni
  - HTTP/2 over TCP (h2c)
  - HTTP/2 over TLS (h2)
    - TLS è opzionale
    - Molti browser impongono TLS
    - Solo HTTPS viene usato con HTTP/2

# HTTP/2: Caratteristiche

- Usa una connessione singola e multiplex
  - Numero massimo di connessioni per dominio può essere ignorato
- Comprime i dati nello header e li invia in un formato binario compatto
  - Migliora il formato basato su testo usato in precedenza
- Non necessita degli accorgimenti usati per migliorare le performance di HTTP/1.1

# HTTP/2: Caratteristiche

- Una connessione TCP
- Richiesta -> Stream
  - Stream sono multiplexati (stream multipli combinati in un singolo canale)
  - Stream sono prioritizzati
- Binary framing layer
  - Prioritization
  - Flow control
  - Server push
- Compressione dell'header (HPACK)




https://developers.google.com/web/fundamentals/performance/http2

# HTTP/2: Binary Framing Layer

- Responsabile della modalità di incapsulamento e trasferimento dei messaggi HTTP tra il client e il server

- Cambia la codifica del dato in transito
  - Tutte le comunicazioni HTTP/2 sono suddivise in messaggi e frame più piccoli, ognuno dei quali codificato in formato binario

# HTTP/2: Binary Framing Layer

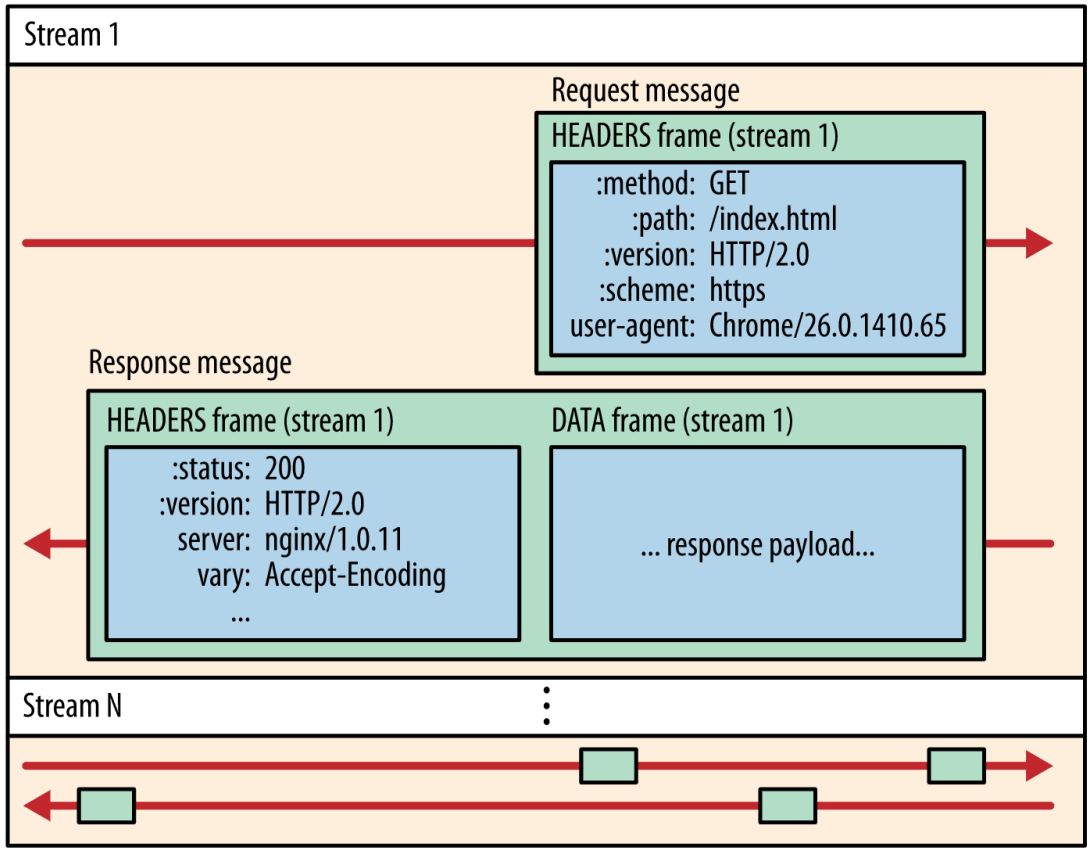
- Componenti
  - Stream: flusso bidirezionale di byte all'interno di una connessione stabilita
    - Può trasportare uno o più messaggi
  - Messaggio: una sequenza completa di frame che mappano a una richiesta logica o un messaggio di risposta
  - Frame: l'unità di comunicazione
    - Ogni frame contiene una intestazione di frame, che come minimo identifica il flusso a cui appartiene il fotogramma

# HTTP/2: Binary Framing Layer

- Ogni singola connessione TCP contiene tutte le comunicazioni con un numero di stream qualsiasi
  - Un identificativo univoco più informazioni di priorità opzionali descrivono ogni stream
- Un messaggio è un messaggio HTTP logico, cioè una richiesta o una risposta
- Il frame è la più piccola unità di comunicazione che porta un tipo specifico di dati, ad esempio, intestazioni HTTP, payload del messaggio e così via
  - Frame di diversi flussi possono essere multiplexati e successivamente assemblati tramite l'identificatore di stream

# HTTP/2: Binary Framing Layer

- Tutti gli stream sono  multiplexati in una connessione TCP



# HTTP/2: Multiplexing

- Ogni stream di richiesta e risposta ha un ID
- Ogni stream contiene diversi frame (header, data…)
- Connessioni TCP possono avere stream multipli
- Frame possono essere intrecciati in un singolo canale TCP
  - In HTTP/1 può essere consegnata una risposta per volta (head of line blocking, uso inefficiente della connessione TCP)
- Dipendenze tra stream controllano la prioritizzazione dei frame

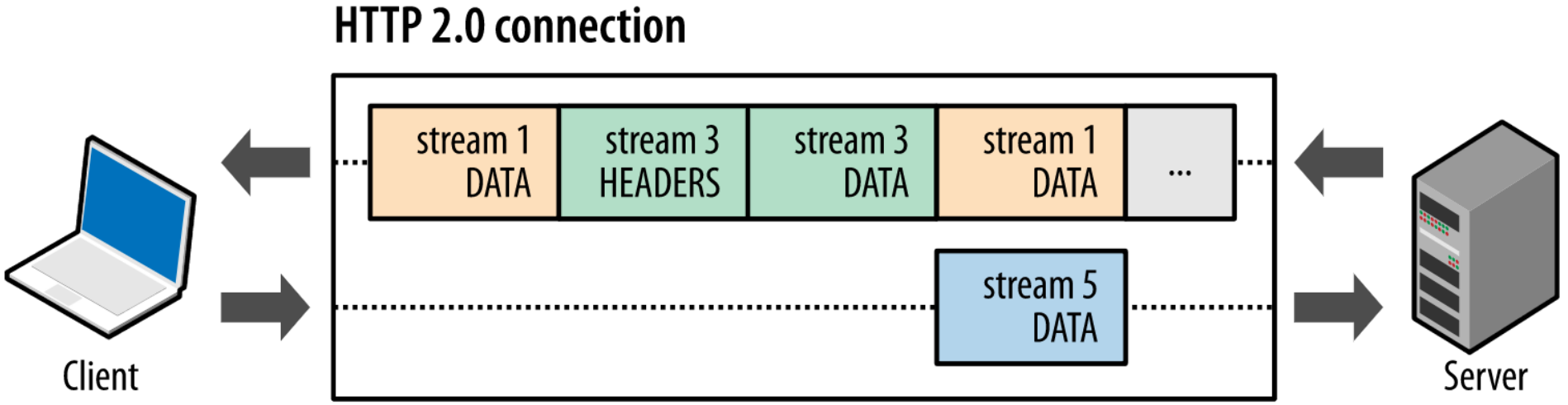
# HTTP/2: Multiplexing

- Tipi di frame

| Frame type | Description |
|---|---|
| DATA | HTTP body |
| HEADERS | Header fields |
| PRIORITY | Sender-advised priority of stream |
| RST_STREAM | Signal termination of stream |
| SETTINGS | Configuration parameters for the connection |
| PUSH_PROMISE | Signal a promise (push) of referenced sources |
| PING | Measure roundtrip time and "liveness" |
| GOAWAY | Inform peer to stop creating streams for current connection |
| WINDOW_UPDATE | Connection flow control |
| CONTINUATION | Continue a segment of header block fragments |

# HTTP/2: Multiplexing

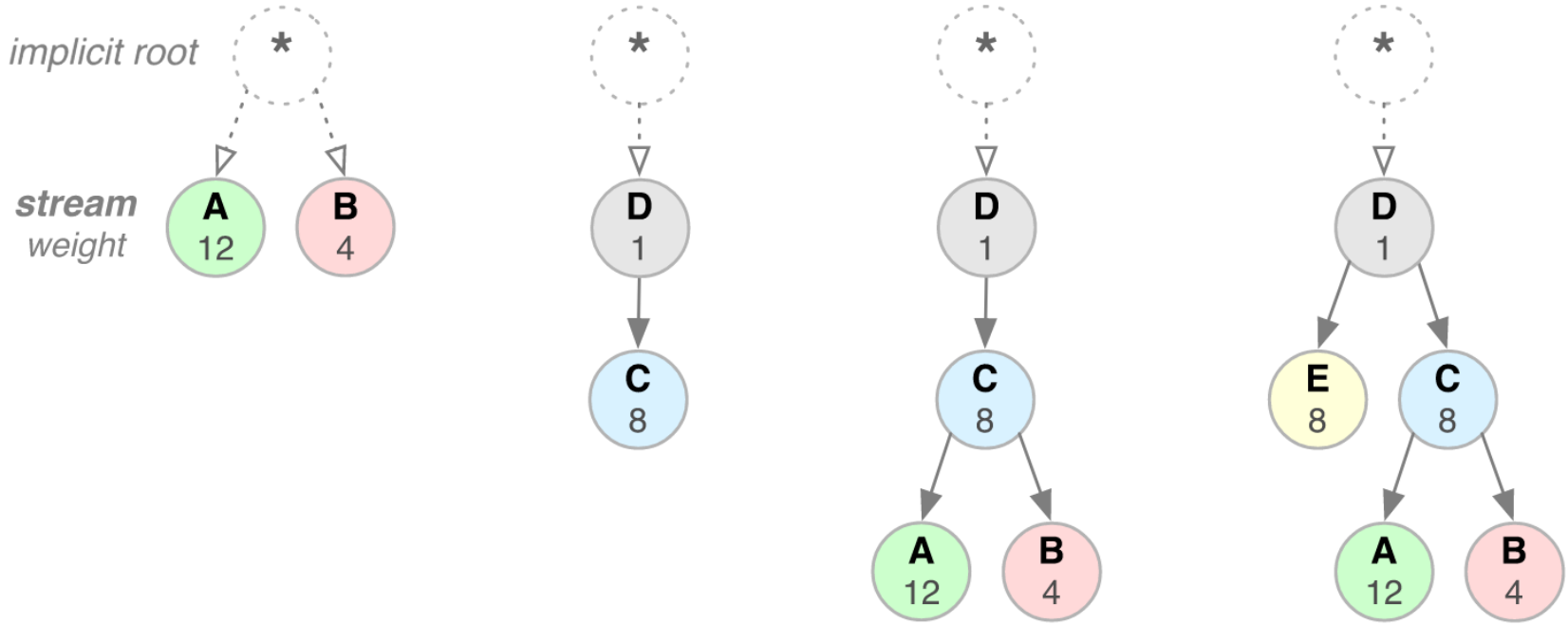
- Binary framing layer supporta il multiplexing di richiesta e risposta tramite frame indipendenti
- Vantaggi
  - Richieste in parallelo senza bloccare nessuno
  - Risposte in parallelo senza bloccare nessuno
  - Singola connessione per più richieste e risposte in parallelo
  - Eliminare le scorciatoie HTTP/1
  - Ridurre i tempi di caricamento della pagina eliminando la latenza e migliorando l'utilizzo della capacità di rete disponibile




https://developers.google.com/web/fundamentals/performance/http2

# HTTP/2: Priorità dello stream

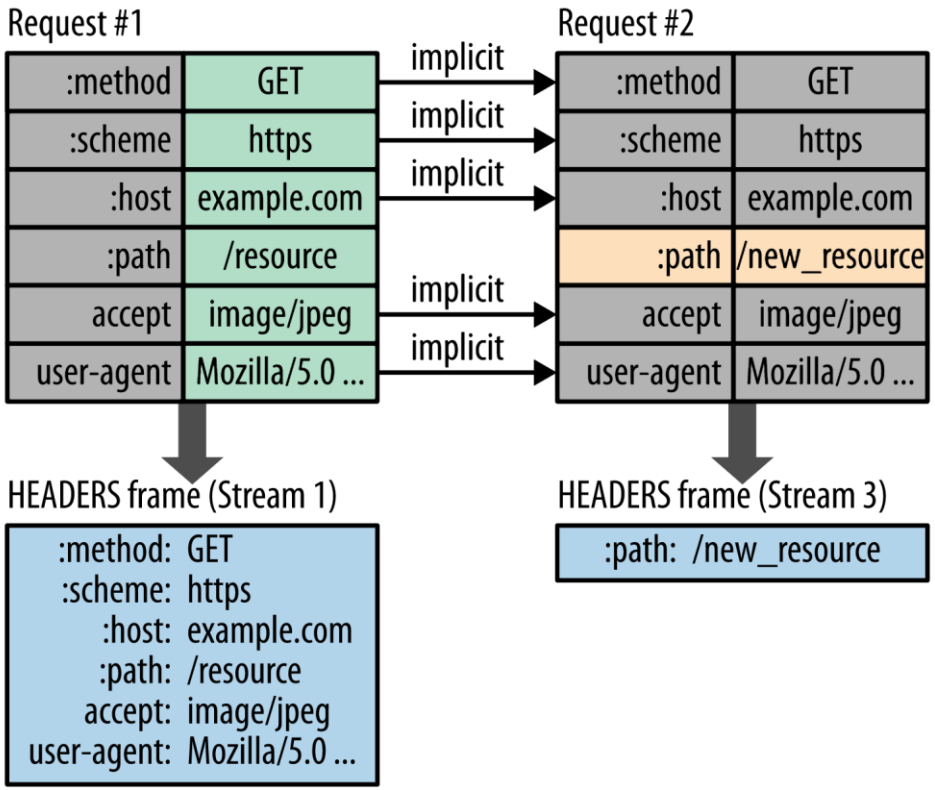
- Ogni stream ha un peso e una dipendenza
  - Peso è un intero compreso tra 1 e 256
  - Dipendenza esplicita su un altro flusso




https://developers.google.com/web/fundamentals/performance/http2

# HTTP/2: Compressione Header

- Header HTTP/1 introducono 500-800 byte di overhead per trasferimento che raggiunge più kilobyte con i cookie HTTP
- HTTP/2 introduce il formato di compressione HPACK
  - Codifica dei campi header tramite static Huffman encoding
    - Compressione dei singoli valori
  - Client e server condividono un elenco indicizzato di header già scambiati
    - Indici usati per codificare tali header a un nuovo invio
    - Come ulteriore ottimizzazione si usa una tabella statica e una dinamica



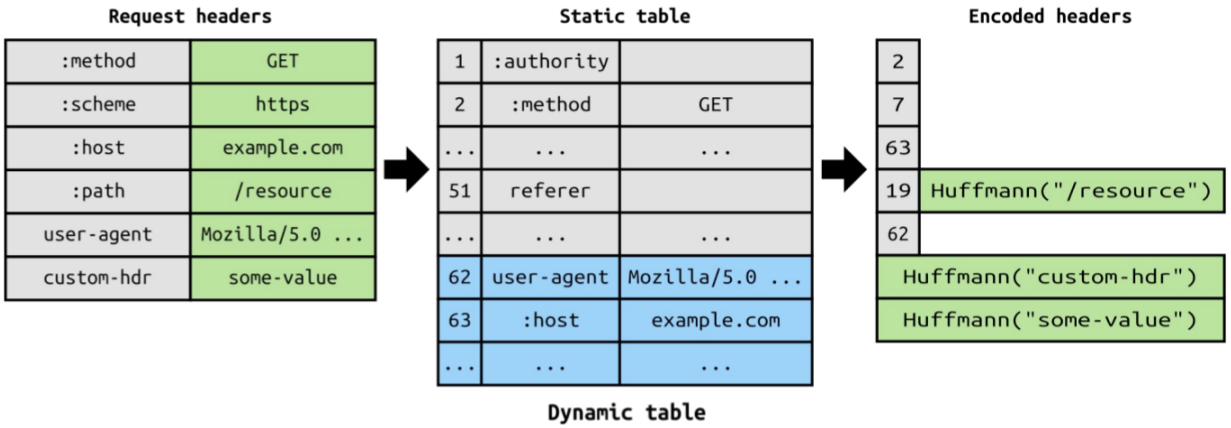

https://developers.google.com/web/fundamentals/performance/http2

# HTTP/2: Compressione Header

- Tabella statica: elenco di header HTTP comune che tutte le connessioni possono utilizzare (RFC7541)

| Index | Header Name | Header Value |
|---|---|---|
| 1 | :authority | |
| 2 | :method | GET |
| 3 | :method | POST |
| 4 | :path | / |
| 5 | :path | /index.html |

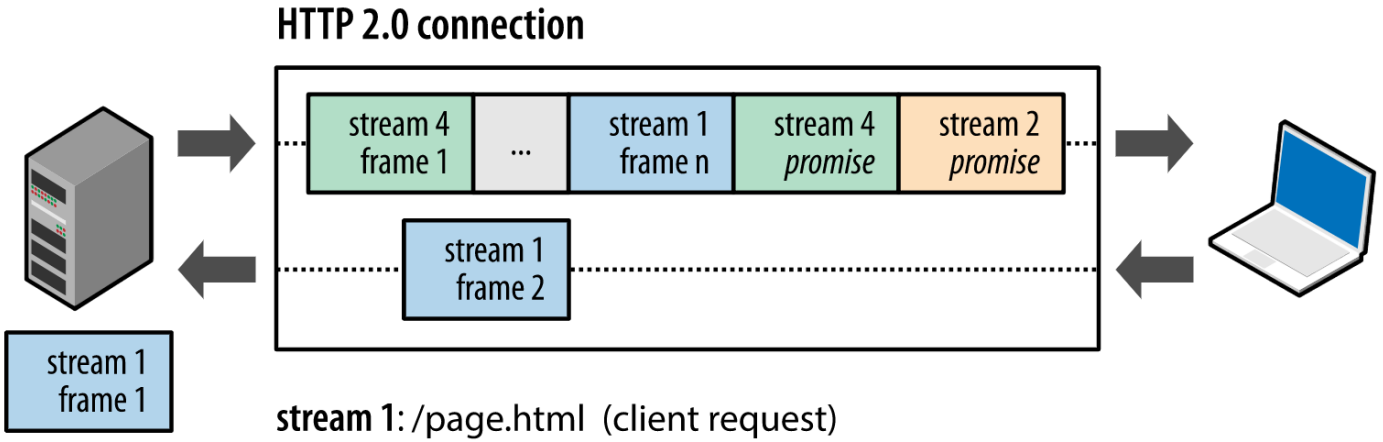
- Tabella dinamica: inizialmente vuota e aggiornata in base ai valori scambiati all'interno di una connessione




https://berlin2017.codemotionworld.com/wp-content/uploads/2017/11/HTTP2_codemotion_v3.pptx.pdf

# HTTP/2: Server Push

- Tipicamente, dopo aver inviato l'HTML, il server aspetta richieste per altre risorse
- In HTTP/2, il server fa push delle risorse comuni
  - JS, CSS, Images, pagine HTML per navigazioni future
  - PUSH_PROMISE indicano l'intenzione del server di eseguire push delle risorse descritte al client e devono essere consegnate prima dei dati
    - Invio solo degli header HTTP, in modo che il client possa rifiutare un invio (nel caso ad esempio l'elemento sia già in cache)

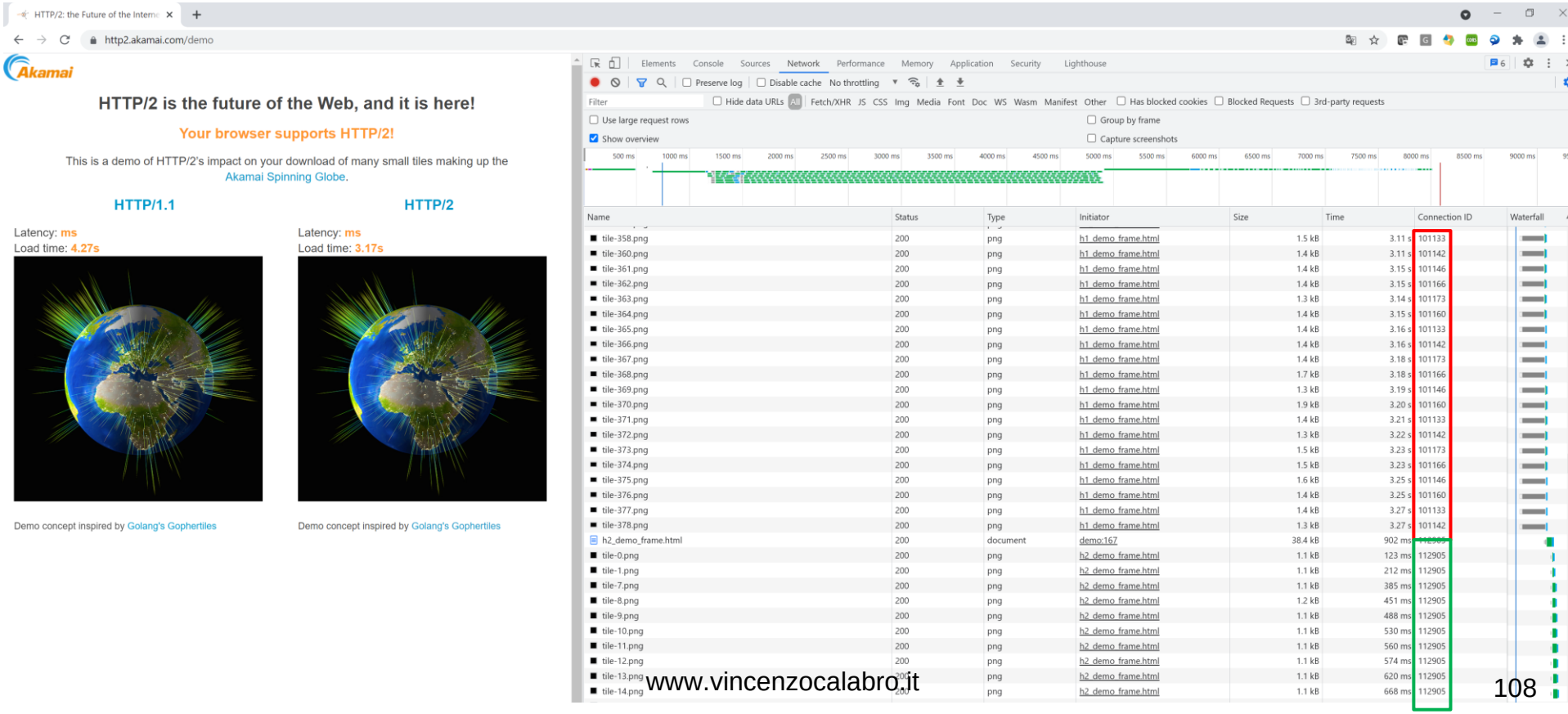



**stream 1**: /page.html (client request)
**stream 2**: /script.js (push promise)
**stream 4**: /style.css (push promise)

https://developers.google.com/web/fundamentals/performance/http2

# HTTP/1 vs HTTP/2

- Confronto tra protocollo HTTP/1 e HTTP/2
  - HTTP/1 usa connessioni multiple (6 in parallelo)
  - HTTP/2 usa una connessione

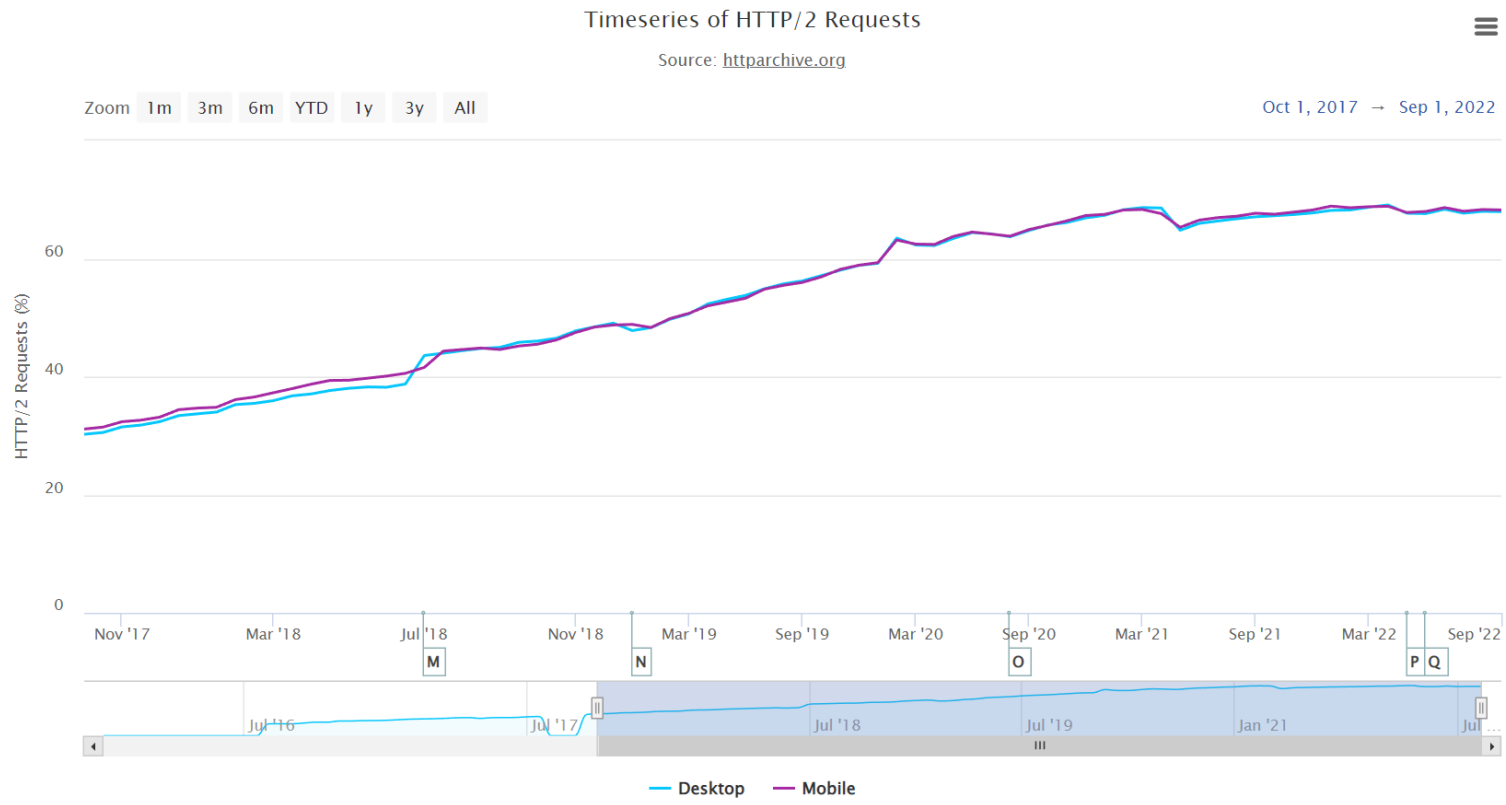


# HTTP/2: Statistics

## HTTP/2 Requests

The percent of all requests in the crawl using HTTP/2. Note that servers supporting HTTP/2 and HTTP/3 may use HTTP/2 initially due to the way the HTTP Archive starts with a fresh Chrome instance each time, but may use HTTP/3 on subsequent requests.

| DESKTOP | MOBILE |
|---|---|
| 68.0% | 68.2% |
| ▲125.2% | ▲119.3% |

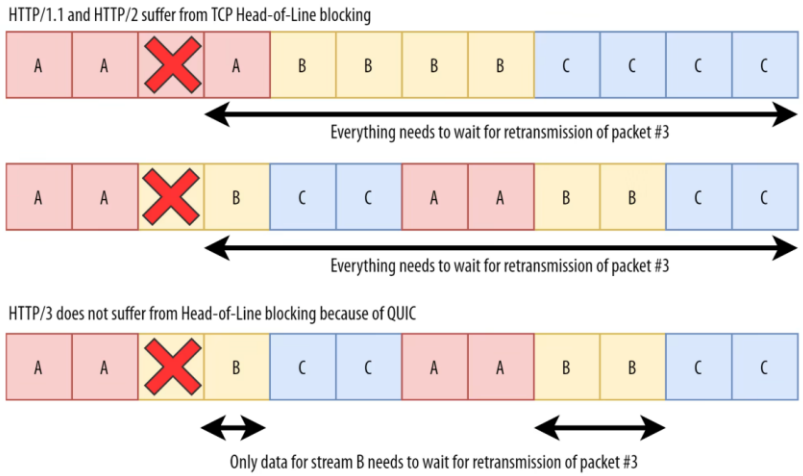


# HTTP/3

*Slide realizzate con il supporto di Simone Ognibene*

# Siamo già ad HTTP/3

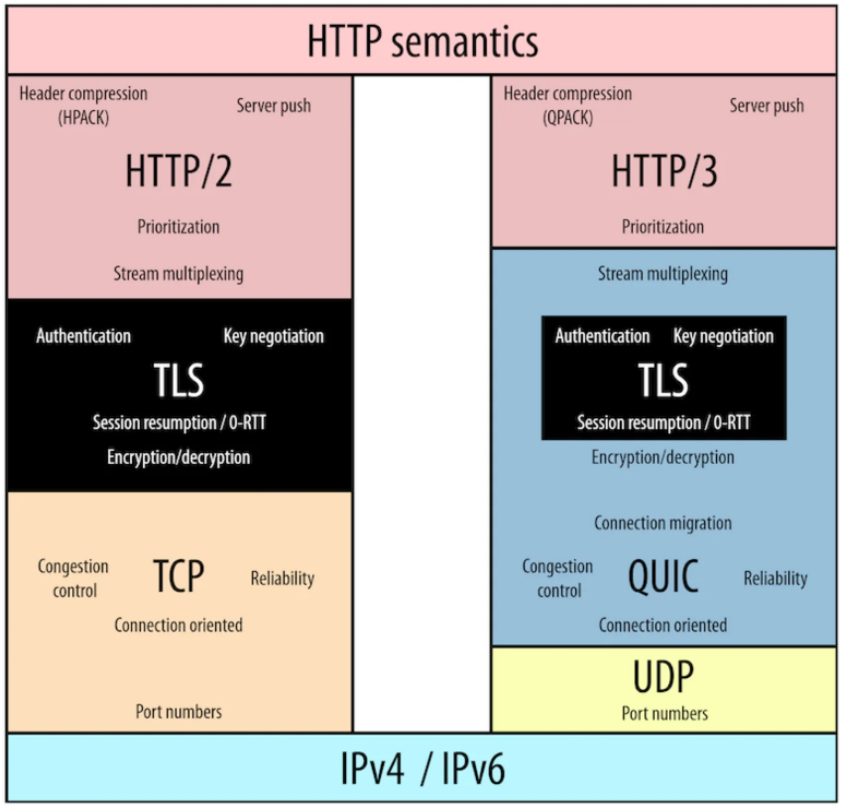
- Internet Draft ma supportato da quasi la totalità dei browser
- Mantiene la semantica di HTTP/1.1
  - *Metodi, codici di stato, campi degli header, URI*
- Modifica il mapping tra la semantica e il layer di trasporto
  - HTTP/1.1 e HTTP/2 usano TCP
  - HTTP/3 usa QUIC, un protocollo a livello di "trasporto" per migliorare il multiplexing (native in QUIC) riducendo l'impatto dei pacchetti persi ai soli stream coinvolti
    - Il protocollo di trasporto di basso livello è UDP




https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/

# Siamo già ad HTTP/3

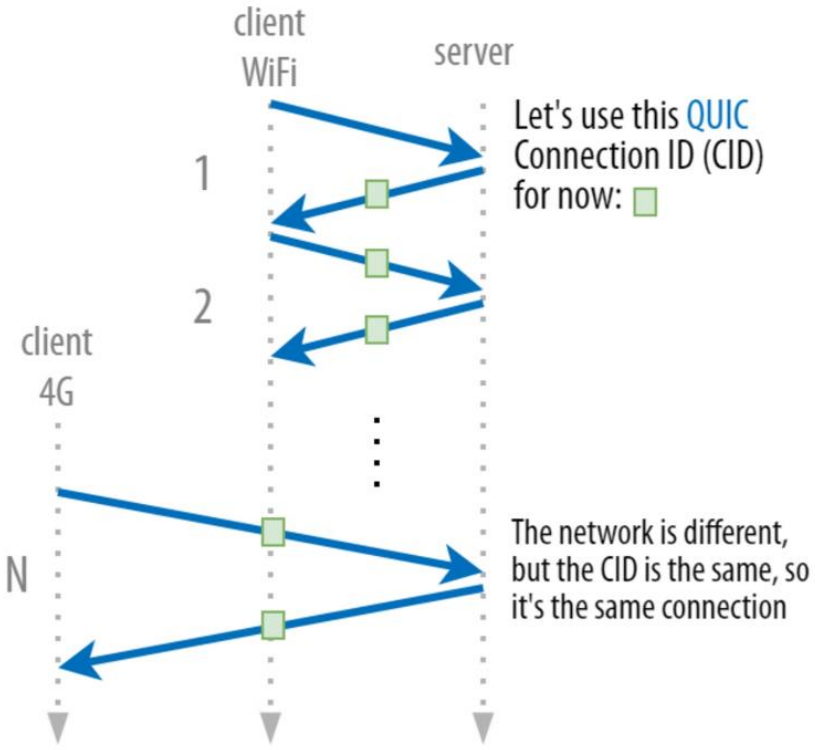
- Perché abbiamo già bisogno di una nuova versione HTTP?
  - Non è veramente necessaria una nuova versione di HTTP
  - Abbiamo bisogno di un nuovo sostituto di TCP
    - QUIC su UDP usato principalmente per gestire performance attraverso una miglior gestione di utenti mobili che passano da una rete all'altra e **miglior integrazione con TLS**




https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/

# Siamo già ad HTTP/3

- Perché abbiamo già bisogno di una nuova versione HTTP?
  - Non è veramente necessaria una nuova versione di HTTP
  - Abbiamo bisogno di un nuovo sostituto di TCP
    - QUIC su UDP usato principalmente per gestire performance attraverso una **miglior gestione di utenti mobili che passano da una rete all'altra** e miglior integrazione con TLS

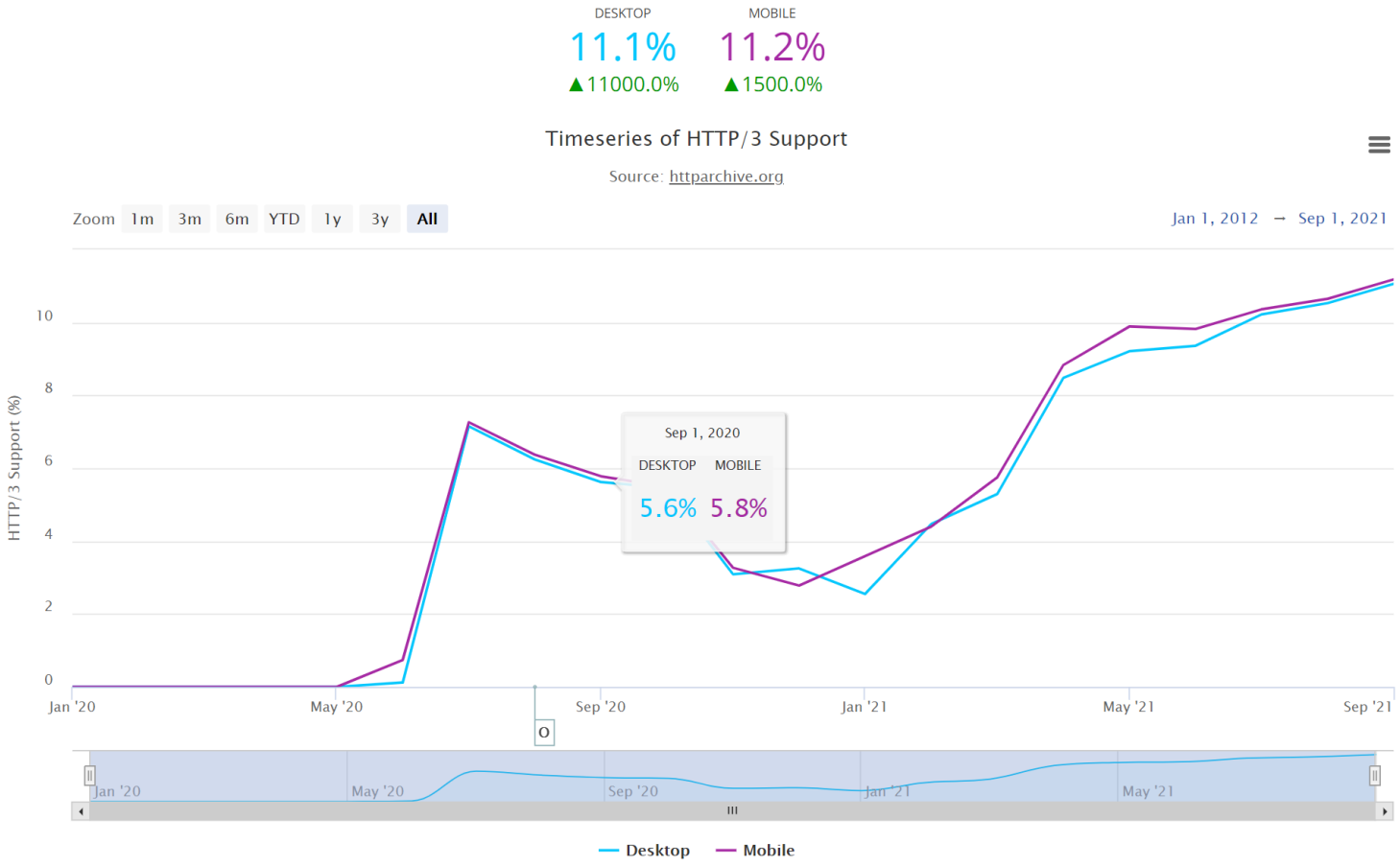



https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/

# HTTP/3: Statistics

## HTTP/3 Support

The percent of all requests in the crawl which support HTTP/3. Note that, due to the way the HTTP Archive starts with a fresh Chrome instance each time, HTTP/3 will often not be used for initial connections until the browser is aware the server supports HTTP/3, which is why we measure support, rather than actual usage like we do for HTTP/2.

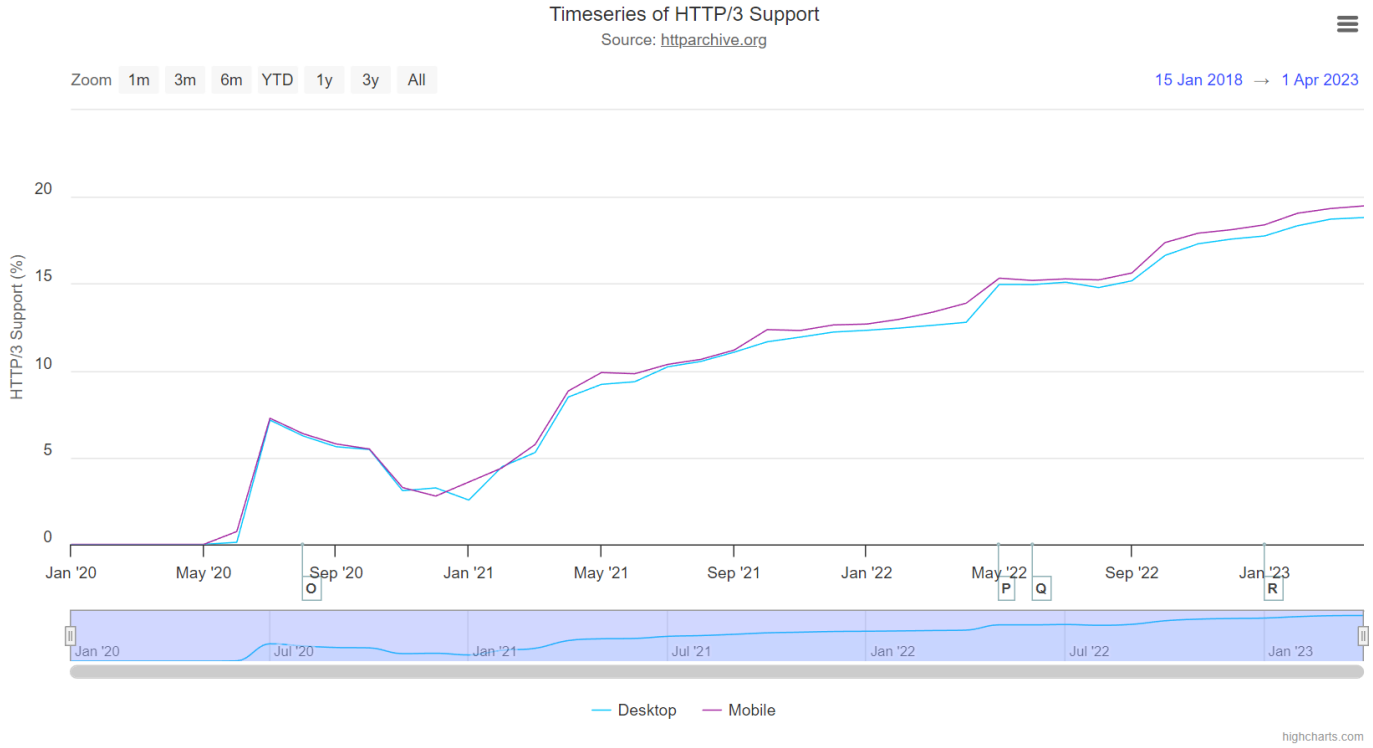


# HTTP/3: Statistics

## HTTP/3 Support

The percent of all requests in the crawl which support HTTP/3. Note that, due to the way the HTTP Archive starts with a fresh Chrome instance each time, HTTP/3 will often not be used for initial connections until the browser is aware the server supports HTTP/3, which is why we measure support, rather than actual usage like we do for HTTP/2.

| DESKTOP | MOBILE |
|---|---|
| 18.8% | 19.5% |
| ▲18700.0% | ▲2685.7% |



# HTTP/3: QUIC

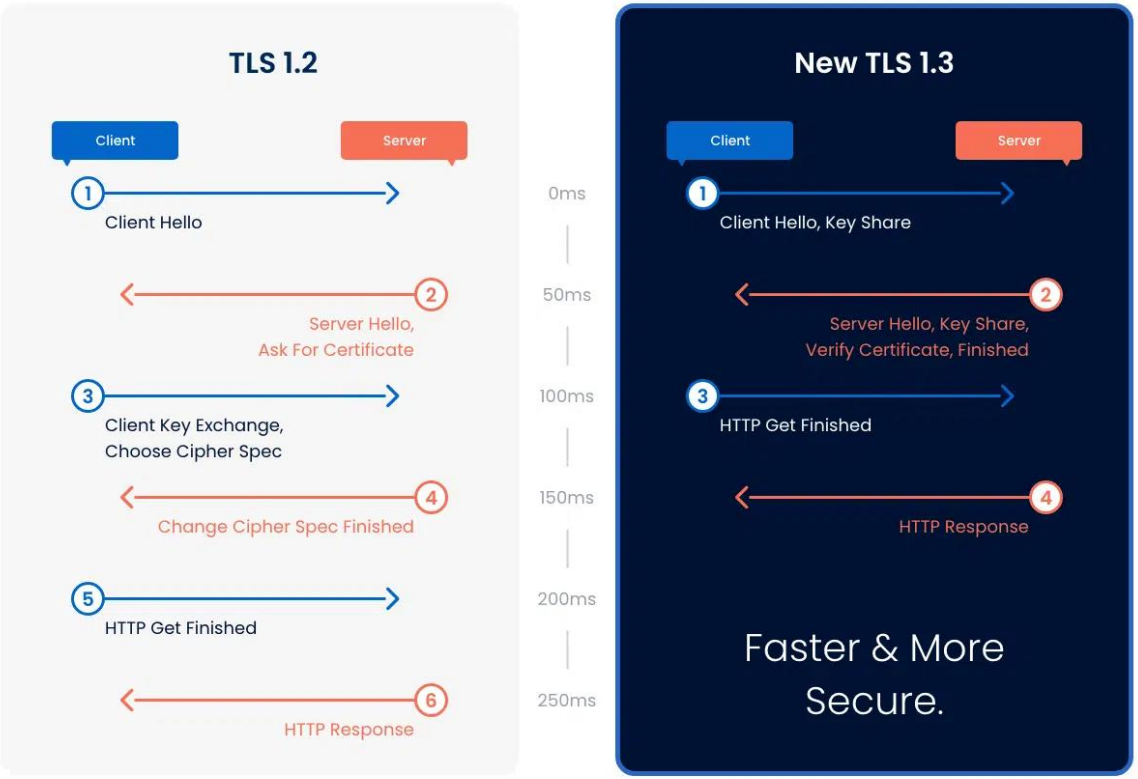
- Protocollo a livello di trasporto basato su UDP
  - Progettato e implementato da Google nel 2012

- Caratteristiche di QUIC
  - **Connectionless**: non necessita di un three-way-handshake
  - **Error handler**: la politica implementata assomiglia alla *send and wait* presente nel *continuous RQ*, ovvero l'invio incondizionato dei pacchetti e la relativa attesa da parte del destinatario di un **ACK**
    - Nel caso di **NACK** (Negative ACK) oppure se il **timeout** scade, il protocollo rinvia i pacchetti riferendosi al numero progressivo a loro assegnato

# HTTP/3: TLS

- HTTP/3 introduce il supporto a **TLS v. 1.3** (Transport Layer Security)
  - Definita nella RFC 8446 (2018)
- Permette di avere un handshake ridotto senza trascurare la sicurezza del protocollo
  - Overhead ridotto con connessioni HTTP/3
- Il client comunica I diversi parametri HTTPS e TLS all'interno di unico pacchetto

# HTTP/3: TLS 1.2 vs TLS 1.3 (Handshake)



# HTTP/3: Headers

- HTTP/3 utilizza un algoritmo di compressione degli headers differente da HTTP/2 (*HPACK*)

- HTTP/3 utilizza **QPACK** che permette una gestione dinamica della tabella contenente gli header
  - Permette di referenziarsi a campi della tabelle precedentemente inviati
  - La compressione avviene in maniera bidirezionale ovvero sia client che server posso comprimere gli header
  - Riduce la ridondanza nello scambio degli header (richiesta/risposta)
  - Anche QPACK come HPACK utilizza due tabelle, una statica e una dinamica, con un'unica differenza che consiste nella separazione dei puntatori ai campi delle tabelle (*field*)

# HTTP/3: Alt-svc

- La RFC 9114 (34° bozza) definisce un header opzionale che permette di notificare al client che il server supporta HTTP/3

- Ecco qui sotto un esempio

**Alt-Svc: h3=":443"; h2=":443"**

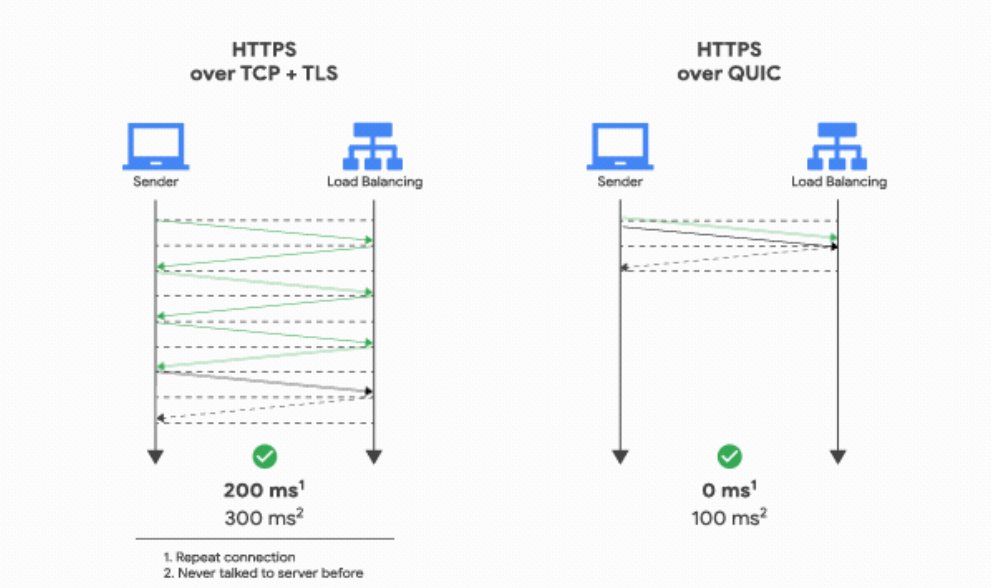
  - **H3** indica la porta del server con supporto HTTP3
  - **H2** indica la porta del server con la quale ci si può connettere con HTTP/2

HTTP/3 supporta nella configurazione del server l'opzione **reuseport** che permette l'utilizzo di una stessa porta, nel caso di HTTP/3 con UDP, nel caso di HTTP/2 con TCP

Non è obbligatorio che le due porte coincidano, posso essere due porte diverse

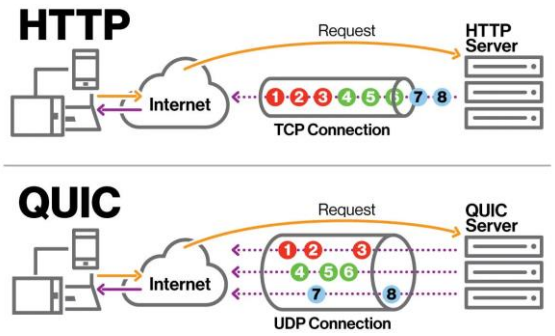
# HTTP/3: Multiplexing

- Quando ci si connette ad un server HTTP/3, QUIC crea differenti stream all'interno di una connessione
  - Quest'ultimi permettono l'invio e la ricezione in maniera concorrente ed efficiente di pacchetti inviati dal server/client
- Questo sistema di gestione delle connessioni permette un'ottimizzazione notevole dell'utilizzo della bandwidth
  - Riduce di gran lunga il numero di connessioni necessarie per l'invio dei dati richiesti, ottenendo così una ridotta congestione di rete



# Ulteriori informazioni

- HTTP/3 non supporta **HTTP Upgrade**, ovvero un header specifico che permetteva a una connessione HTTP/1 di fare l'upgrade al protocollo HTTP/2
- Ogni connessione QUIC, essendo un protocollo non orientato alla connessione, prevede un valore identificativo per ognuna di esse (connection ID), in modo da referenziarsi alla connessione indicata
- QUIC richiede un round-trip time (RTT) pari a 0 per stabilire una connessione, il protocollo può negoziare i parametri di comunicazione in modo concorrente alla trasmissione di dati
- Il multiplexing ne permette un aumento dello spazio di banda utilizzato ed un'ottimizzazione delle prestazioni di scambio attivo/passivo dei dati



# Conclusioni

- Web è l'applicazione internet più usata
  - URL identificano ogni risorsa in maniera univoca
- Standard per il trasferimento è HTTP
  - HTTP/1 vs HTTP/2 vs HTTP/3
- Standard per la rappresentazione è HTML (prossima lezione)

# Riferimenti

- https://developers.google.com/web/fundamentals/performance/http2
- https://http2.akamai.com/demo
- https://www.slideshare.net/idof/introduction-to-http2-72124509
- https://berlin2017.codemotionworld.com/wp-content/uploads/2017/11/HTTP2_codemotion_v3.pptx.pdf
- https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/
- https://www.smashingmagazine.com/2021/08/http3-performance-improvements-part2/
- https://www.smashingmagazine.com/2021/09/http3-practical-deployment-options-part3/