CS290b – Lecture 3
HTTP protocol

Scalable Internet Services and Systems, Fall 2013

Jon Walker
Department of Computer Science
University of California at Santa Barbara

For today…

- HTTP Protocol
- Announcements

HTTP 0.9

- **The Original HTTP as defined in 1991**
  - After-the-fact definition “as originally implemented by the World Wide Web initiative software in the prototype released”
- **Request**
  - Request: "GET", a space, the document address, CR-LF.
  - The document address will consist of a single word (ie no spaces)
- **Response**
  - The response is a message in HTML. This is a byte stream of ASCII characters.
  - The format of the message is HTML. It also allows for plain ASCII text to be returned following the PLAINTEXT tag .
  - The message is terminated by the closing of the connection by the server.
  - Error responses are supplied in human readable text in HTML syntax. There is no way to distinguish an error response from a satisfactory response except for the content of the text.

HTTP versions

- **HTTP/0.9**
  - Deprecated. Only supports one command, GET — which does not specify the HTTP version.
  - Does not support headers.
  - Does not support POST, the client can't pass much information to the server.
- **HTTP/1.0**
  - First protocol revision to specify its version in communications and still in wide use, especially by proxy servers.
  - Allows persistent connections when explicitly negotiated but otherwise connections are closed after each request
- **HTTP/1.1**
  - Current version; widely used; persistent connections enabled by default
  - Supports request pipelining, allowing multiple requests to be sent at the same time
- **HTTP/1.2**
  - The initial 1995 working drafts of *PEP — an Extension Mechanism for HTTP*. PEP later became subsumed by the experimental RFC 2774 — *HTTP Extension Framework*.

HTTP 2.0

- **Draft Standard**
- **April 2014 is last call for comments on HTTP 2.0**
- **November 2014 submit as proposed standard**
- **Based on Google's SPDY (original draft was a copy of SPDY)**
  - Supported by some browsers already
- **Microsoft also has HTTP Speed+Mobility**

- **Speed up HTTP**
  - Compression – of headers
  - Multiplexing – single connection for multiple resource requests
  - Prioritization or Pipelining – server pushing of content that is likely to be requested

HTTP basics

- **URL = Uniform Resource Locator**
  - Resource ≠ file
- **Requests and responses**
  - Initial request/response line
    - `GET /path/to/file/index.html HTTP/1.0`
    - `HTTP 404 Not found`
  - Zero or more header lines
  - Blank line (CRLF)
  - Optional message body
- **TCP transport**
  - Client opens TCP connection to server

HTTP's 8 methods

- **GET**
  - Requests a representation of the specified resource.
- **POST**
  - Submits data to be processed (e.g. from a HTML form) to the identified resource.
  - The data is included in the body of the request or the query string.
- **PUT**
  - Uploads a representation of the specified resource.
- **DELETE**
  - Deletes the specified resource.
- **HEAD**
  - Asks for the response identical to the one that would correspond to a GET request, but without the response body.
  - Useful for retrieving meta-information written in response headers, without having to transport the entire content.
- **TRACE**
  - Echoes back the received request, to see what intermediate servers are adding or changing in the request.
- **OPTIONS**
  - Returns the HTTP methods that the server supports.
- **CONNECT**
  - For use with a proxy that can change to being an SSL tunnel.

HTTP methods (cont.)

- **Safe methods**
  - GET and HEAD are defined as safe
    - they are intended only for information retrieval and should not change the state of the server
  - POST, PUT and DELETE may modify information
  - OPTIONS and TRACE are also defined to be safe
- **Idempotent methods**
  - GET, HEAD, PUT and DELETE are defined to be *idempotent*
    - multiple identical requests should have the same effect as a single request
- **GET is safe and idempotent?**
  - Has anyone heard of Google Web Accelerator?
    - How can you safely prefetch?

HTTP responses

- **Initial response line:**
  - `HTTP/1.0 <code> <text>`
  - Code is error code:
    - 1xx: informational
    - 2xx: success, e.g. 200 OK
    - 3xx: redirect, e.g. 301 Moved Permanently, 302 Moved Temporarily
    - 4xx: error by client, e.g. 404 Not Found
    - 5xx: error by server, e.g. 500 Internal Server Error
- **Headers:**
  - <header-name>: <value>, <value>, …
  - Example:

```
Header1: some-value-1a, some-value-1b
HEADER1: some-value-1a,
         some-value-1b
```

Persistent Connections

- Connections are persistent by default:
  - Send request 1, receive response 1
  - Send request 2, receive response 2
  - Note: requires well-delimited requests & responses (content-length & chunked encoding)
- Termination:
  - Server: "`Connection: close`", or close connection (e.g. idle)
  - Client: close connection (or "`Connection: close`")
  - Note: client must handle aborted connections by retrying
- Pipelining:
  - Send request 2 before response 1 recv'd
  - Strictly in-order responses
  - Careful with non-idempotent requests

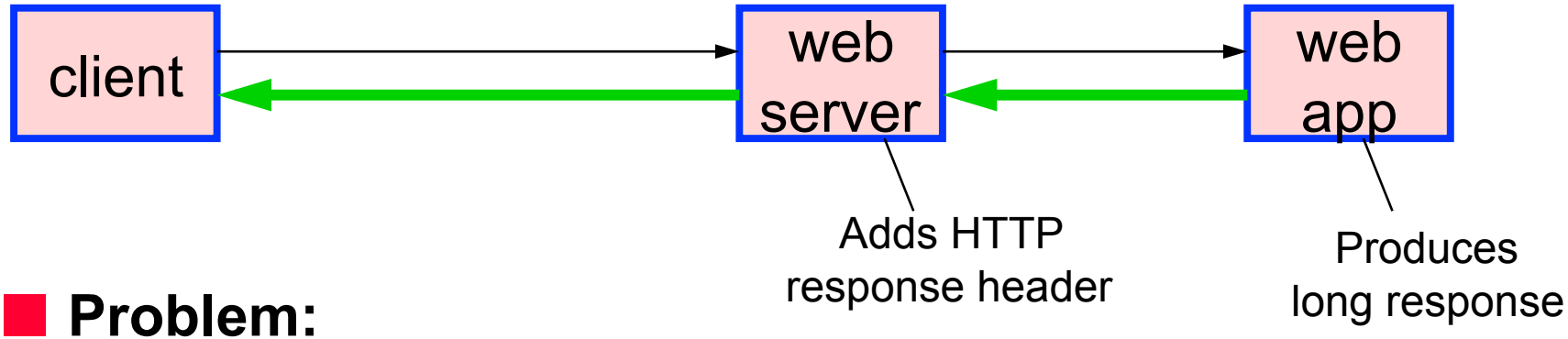
Host header

- **Problem: 1 server but 1000 domain names (www.cs290.com)**
- **Solution: required host header**
  - GET /index.html HTTP/1.1
    Host: www.cs290.com
  - As opposed to assigning 1000 IP addresses to the server

Range header

- **Problem:**
  - aborted connections
  - fetching prefixes
- **Solution: specify requested byte range**
  - GET /long_song.mpg HTTP/1.1
    Range: bytes=44580–
  - GET /bigdocument.xml HTTP/1.1
    Range: bytes=10000–20000

Chunked encoding



- **Problem:**
  - Streaming dynamically generated responses
- **Solution: chunked encoding**
  - Send response in chunks
  - Each chunk prefixed by length
  - Zero-length chunk signifies end of document
  - Chunk: $length_{16}$ CRLF data CRLF
  - (Additional headers can be sent after last chunk)

MIME types

- **Problem: what type of content can the requestor process?**
  - Accept header
    - Accept: text/plain; q=0.5, text/html, text/x-dvi; q=0.8, text/x-c
      Verbally, this would be interpreted as "text/html and text/x-c are the preferred media types, but if they do not exist, then send the text/x-dvi entity, and if that does not exist, send the text/plain entity."
  - Accept-charset
  - Accept-encoding
    - Accept-Encoding: compress, gzip
  - Accept-Language: da, en-gb;q=0.8, en;q=0.7
- **Content-Type**

MIME types (cont.)

- **Problem: what type of content is in the response?**
  - File name extensions are not universal…
  - Content-Type: text/plain, image/jpeg, …
  - Content-Language
  - Content-Charset
  - Content-Encoding: gzip
    - Example: content-type: text/plain content-encoding: gzip

ETags, conditional gets

- **Problem: how to validate a cached version**
- **Solutions: etag and last-modified**
- **Entity tags (ETag header): unique tag**
  - Two entities may have the same tag only if they are equal byte-by-byte
  - Often MD5 sum is used (e.g. by Amazon S3)
  - Server provides `ETag` with response
  - Client/cache queries with `If-Not-Match:` *`entity-tag`*
  - Server responds with
    - `304 Not Modified` – no response body
    - `200 OK` – new response body
- **Also:**
  - If-Match: *entity-tag* – useful for PUTs

Conditional GETs (cont.)

- **Date-based validation:**
  - Server provides `last-modified` date/time in response
  - Client/cache queries using `If-Modified-since:` *`date`*
    - should use last-modified header value, not arbitrary date/time
  - Server responds with 200 or 304
- **Also:**
  - If-Unmodified-Since

Cookies

- **Cookies not part of HTTP/1.1! See RFC2965…**
  - Headers: Set-Cookie2, Cookie
  - Fields:
    - Domain=.gotomypc.com
    - Path=/myaccount/
    - Port="80,8080"
    - Discard (when u-a terminates)
    - Max-age=3600
    - Secure
- **Uses?**
  - How to handle users who disable them?
- **Security?**

HTTP “Work Arounds”

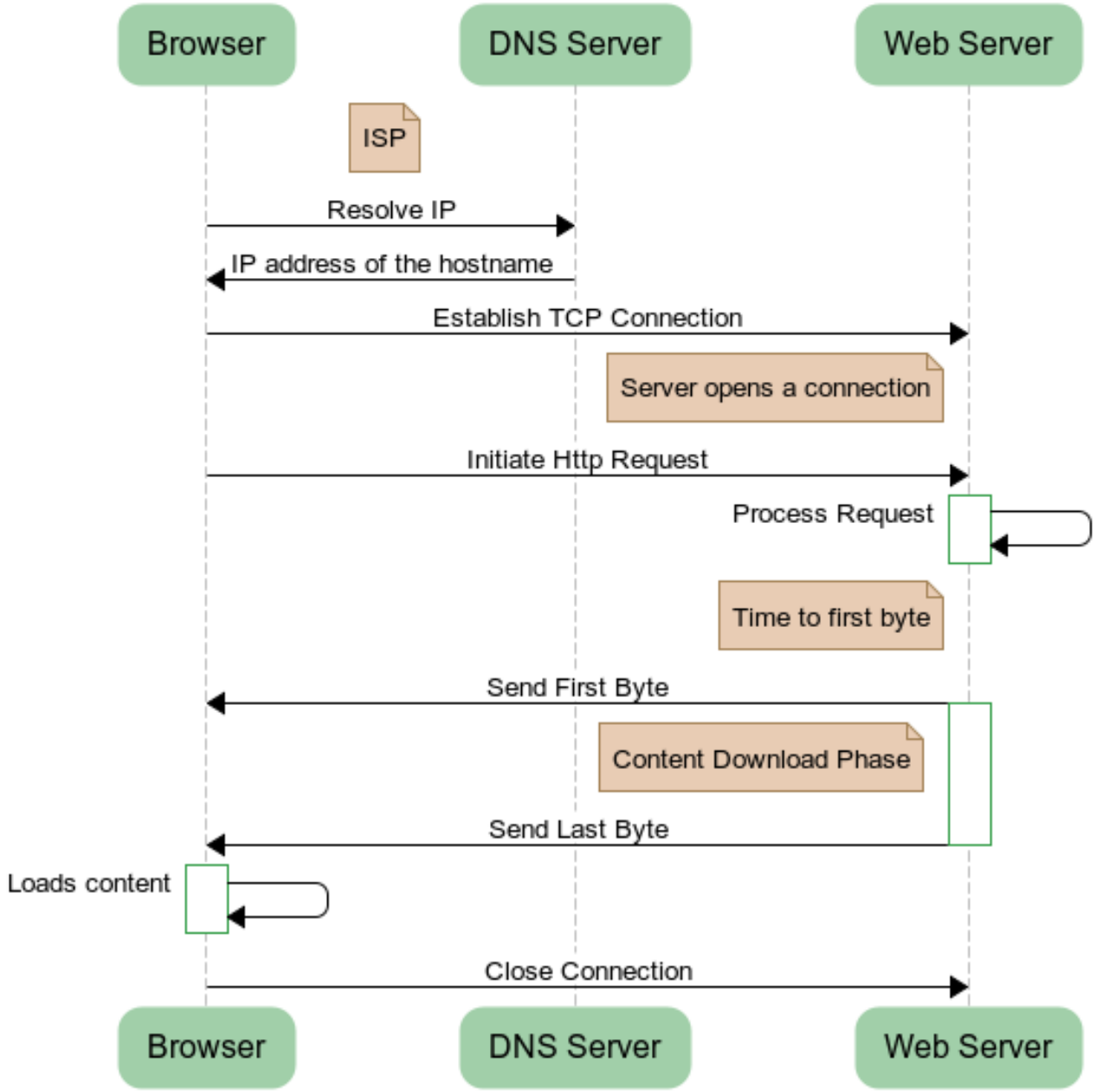
- **Instant notifications**
  - How to get notified when server state changes
  - E.g. instant message / chat message arrives?
- **Bidirectional communication**
  - How can “server” issue “requests” to client?

HTTP “Work Arounds”

- **Instant notifications**
  - How to get notified when server state changes
  - E.g. instant message / chat message arrives?
- **Bidirectional communication**
  - How can “server” issue “requests” to client?
- **Techniques**
  - Comet – long-held HTTP request
    - Long polling, Flash XML Socket, XMPP, Script tag polling, etc.
  - Server-Sent Events (HTML 5)
  - WebSockets (HTML 5)

Anatomy of a Web Request

- What does a web request look like?


Browser
DNS Server
Web Server
ISP
Resolve IP
IP address of the hostname
Establish TCP Connection
Server opens a connection
Initiate Http Request
Process Request
Time to first byte
Send First Byte
Content Download Phase
Send Last Byte
Loads content
Close Connection
Browser
DNS Server
Web Server

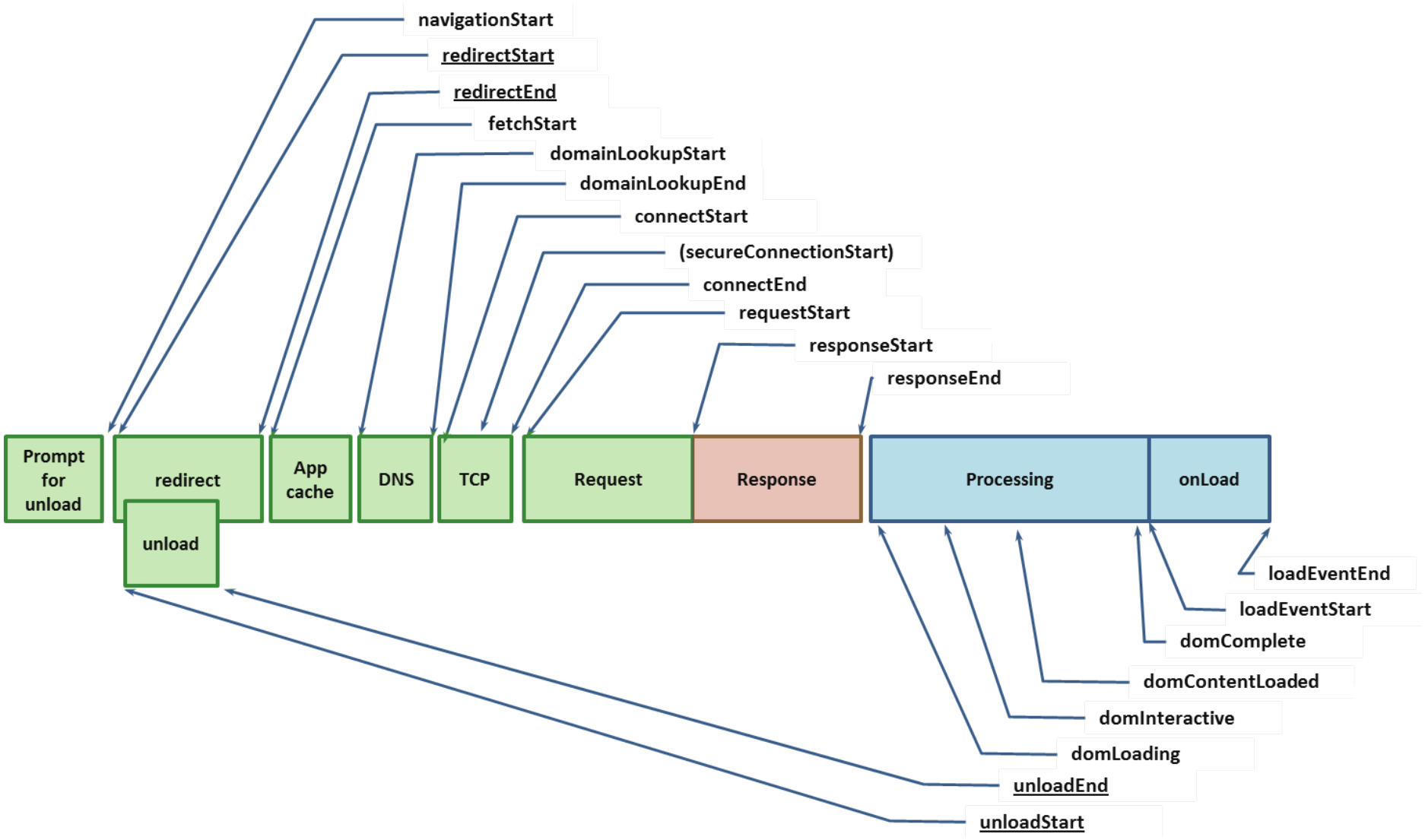
Navigation Timing

W3C Draft Standard

- Javascript API to provide complete client side latency measurements
- http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/NavigationTiming/Overview.html
- Supported by Chrome, Firefox, IE 9+
- Collected by Google Analytics

In Chrome

- View -> Developer -> Javascript console
- `Performance`
- Great talk by Ilya Grigorik – Google
  - MTWF (Make The Web Fast)
  - http://www.confreaks.com/videos/886-railsconf2012-let-s-make-the-web-faster-tips-from-trenches-google


navigationStart
redirectStart
redirectEnd
fetchStart
domainLookupStart
domainLookupEnd
connectStart
(secureConnectionStart)
connectEnd
requestStart
responseStart
responseEnd
Prompt for unload
redirect
unload
App cache
DNS
TCP
Request
Response
Processing
onLoad
loadEventEnd
loadEventStart
domComplete
domContentLoaded
domInteractive
domLoading
unloadEnd
unloadStart

Summary

- **HTTP Protocol**
  - More complex than it seems
  - It has evolved to do more
  - Still “work arounds” to solve problems (changing with HTTP 2.0 and HTML 5)
- **Request processing is the core function of a Scalable Web Service**
  - Understand entire request life cycle
  - There is active work to make this easier

Questions about HTTP?

Announcements

- Assignments – email to grading@cs290.com
  - AWDWR through Chapter 8 due Wed.10/9 at midnight
  - AWDWR through Chapter 15 due by Monday 10/14 at midnight
- We will kick off class projects and form teams next week
- Guest Lecture Wed.
  - Andrew Mutz, Dir. Of Engineering, AppFolio
- Lab tonight!
  - 5pm-7pm
  - Phelps 1401