Render!
I'd like to see X

Gimme X

HTML
HTTP: the HyperText Transfer Protocol

- Originally developed by Sir Tim
- HTTP v1.0 standard presented 1996
- HTTP/1.1 standard finalized in 1997
  - Via RFC 2068
    - Though improvements and updates in RFC 2616 (1999) essentially replace RFC 2068 as the definition of HTTP/1.1
- In 2009, Google produced SPDY, another protocol for the transfer of web traffic
  - Doesn't replace HTTP, provides a tunnel for HTTP traffic
- In 2015, HTTP/2.0 standard was finalized
  - Based around SPDY
  - Google has since deprecated SPDY
HTTP basics: GET

- First method implemented
  - HTTP now has several methods defined that specify the action that is requested to be performed on given resource
- Simply fetch the resource at some URL
GET / HTTP/1.1
Host: cs.pitt.edu

HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>

HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
So the Host attribute of the request says where to go

- Well… no, not really…
So how do we get HTTP requests to the webserver?
To send an HTTP request

- Link layer...
- IP
- TCP
- SPDY
- HTTP

- MAC address of next hop, etc.
- Destination IP
- Destination port
How do we get information to the server?

- **POST**
  - Attaches data with the request that should be handled by the specified resource
    - E.g.,
      - The result of a web form
      - A new entry to add to a database

- **PUT**
  - Attaches data that should be *placed* at the specified resource
    - If the resource does not currently exist, specified data should not be that resource identified by the given URL
PUT sounds dangerous...

- Safe HTTP methods
  - Should only request a resource, should not change the state of the server
    - GET is (by convention) a safe method
- POST and PUT are intended to cause side-effects (i.e., change the state of the server)
In theory, there is no difference between theory and practice ...

- In practice there is
  - Yogi Berra
  - Michael Scott

- URL format:
  scheme://[user:password@]host[:port]/[path][?query][#fragment]

- The URL *query string* can be used to affect server state

E.g.:
  - http://example.com/storefront?user=nick&newitem=laptop

  - Could be used by the example.com webstore app to have me request to buy a laptop

  - This is BAD
HTTP Methods

- GET
- HEAD
  - Like GET, but returns headers only, no body
- POST
- PUT
- DELETE
  - Delete listed resource
## Comparisons of HTTP Methods

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>RFC 7231</th>
<th>Request Has Body</th>
<th>Response Has Body</th>
<th>Safe</th>
<th>Idempotent</th>
<th>Cacheable</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>RFC 7231</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HEAD</td>
<td>RFC 7231</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>POST</td>
<td>RFC 7231</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PUT</td>
<td>RFC 7231</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DELETE</td>
<td>RFC 7231</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CONNECT</td>
<td>RFC 7231</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>RFC 7231</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TRACE</td>
<td>RFC 7231</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PATCH</td>
<td>RFC 5789</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
HTTP Status Codes

- 200
  - OK
- 301
  - Moved Permanently
- 400
  - Bad Request
- 403
  - Forbidden
- 404
  - Not Found
- 500
  - Internal Server Error
- ...
GET start.php HTTP/1.1