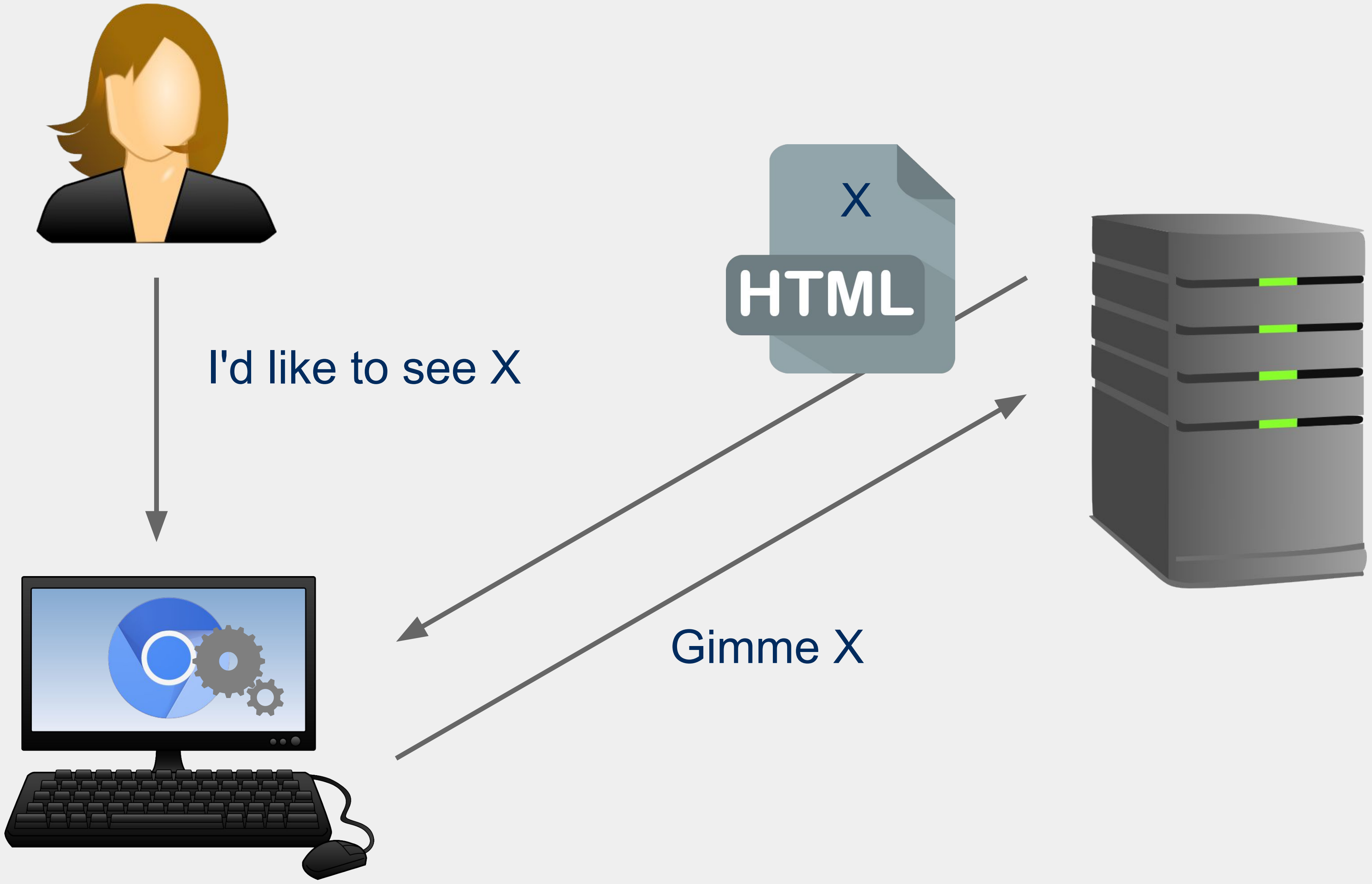
CS/COE 1520

pitt.edu/~ach54/cs1520

HTTP Overview and a Brief Introduction to Networking

Render!
HTML

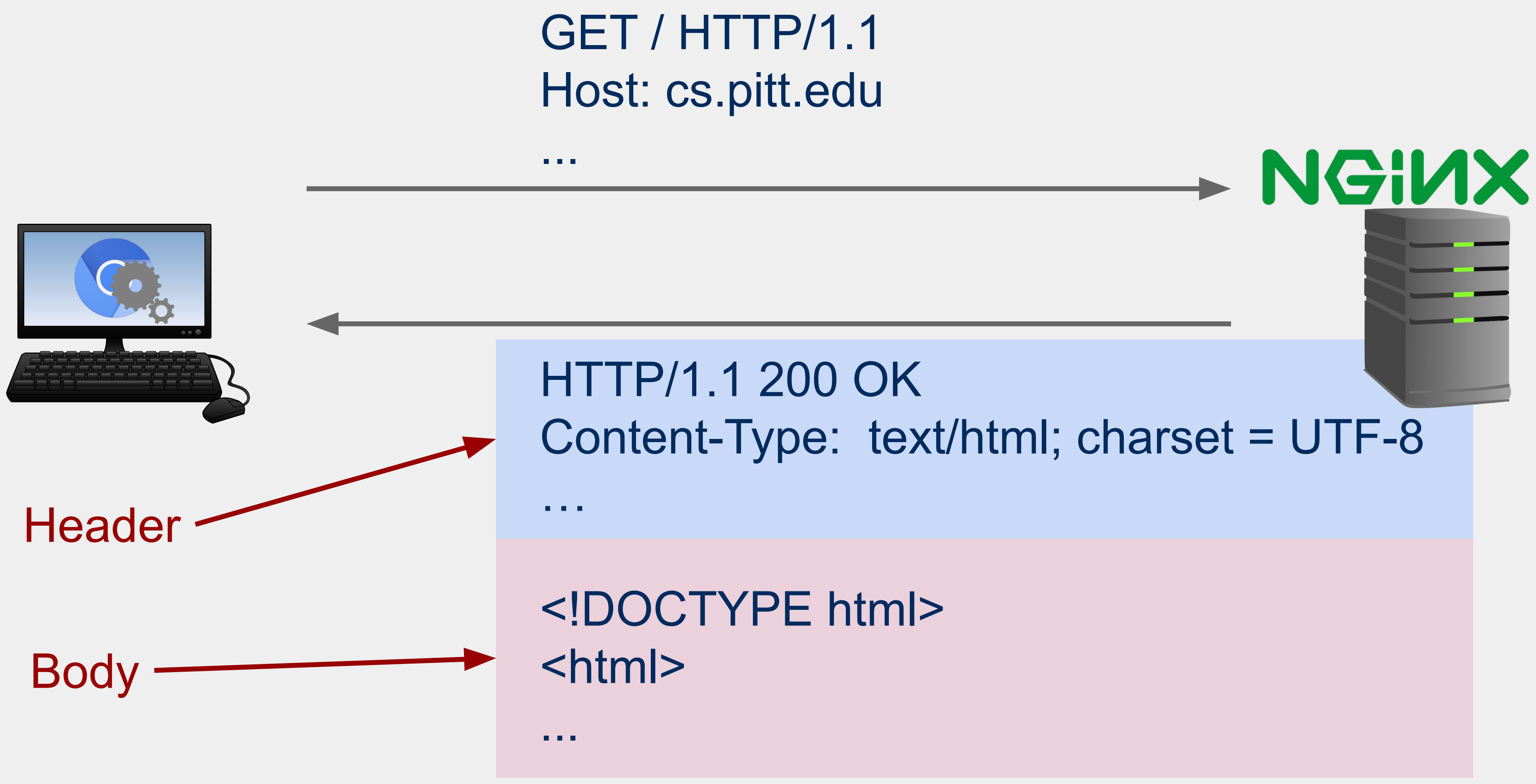

I'd like to see X
X
HTML
Gimme X

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

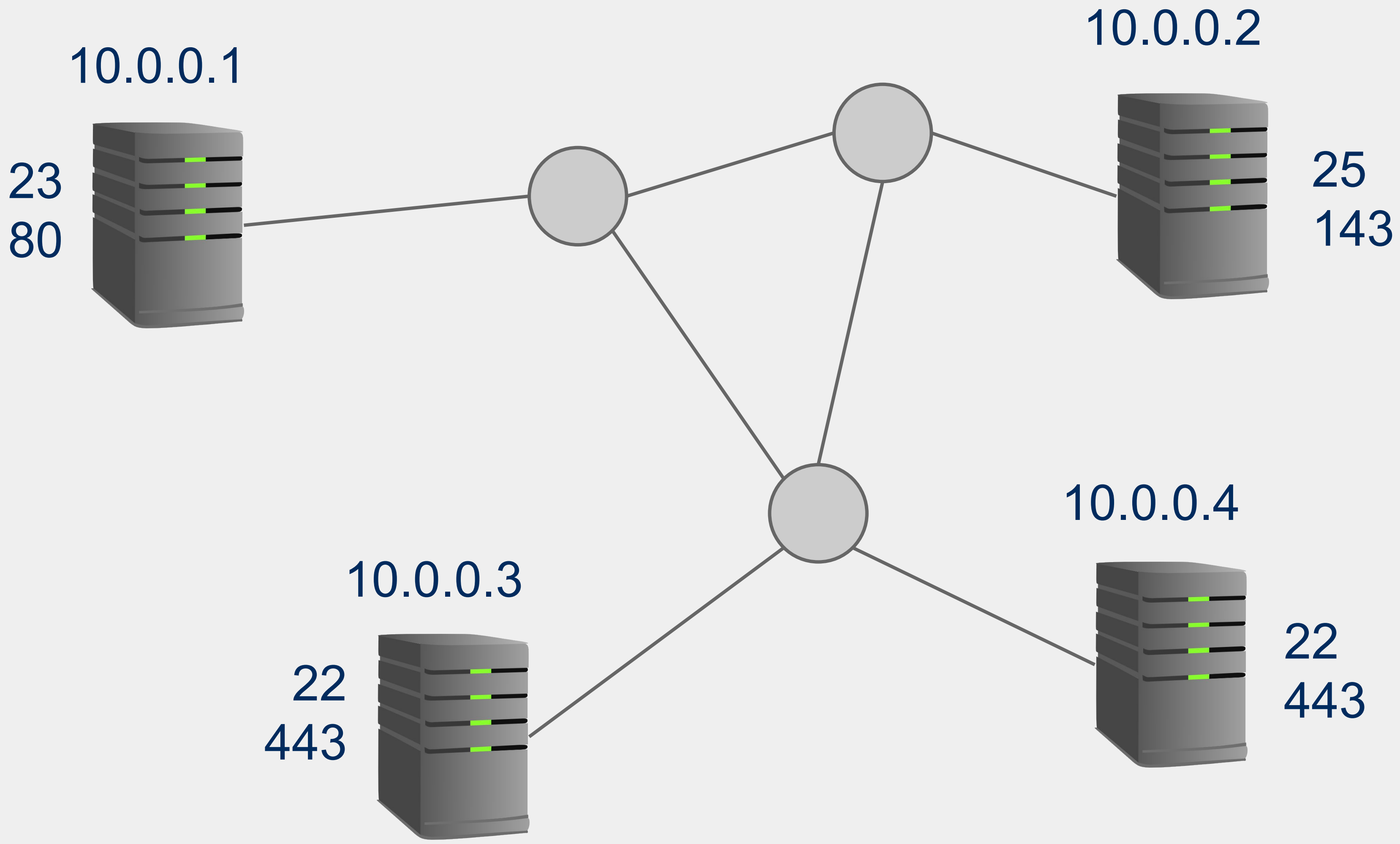

```
GET / HTTP/1.1
Host: cs.pitt.edu
...
```

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

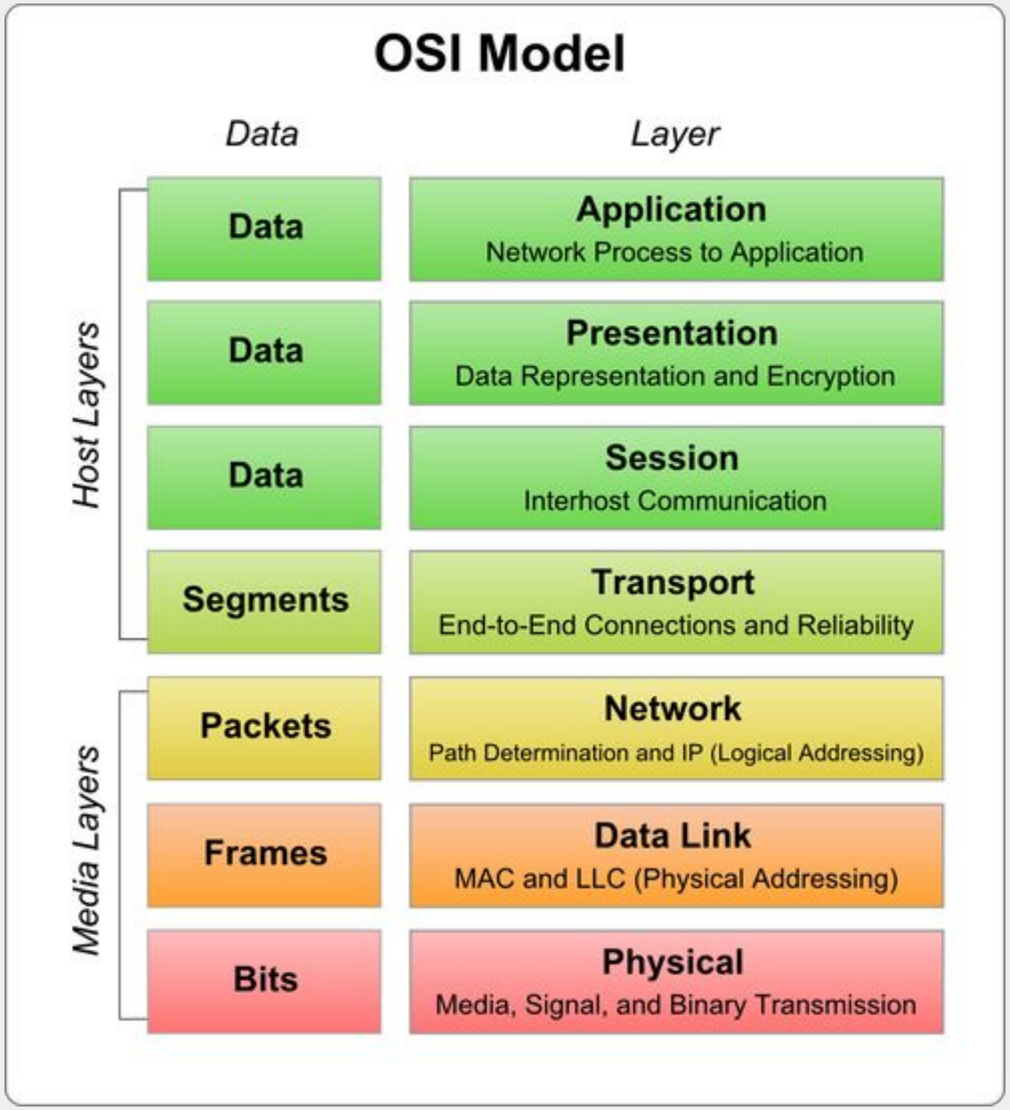
```
<!DOCTYPE html>
<html>
...
```

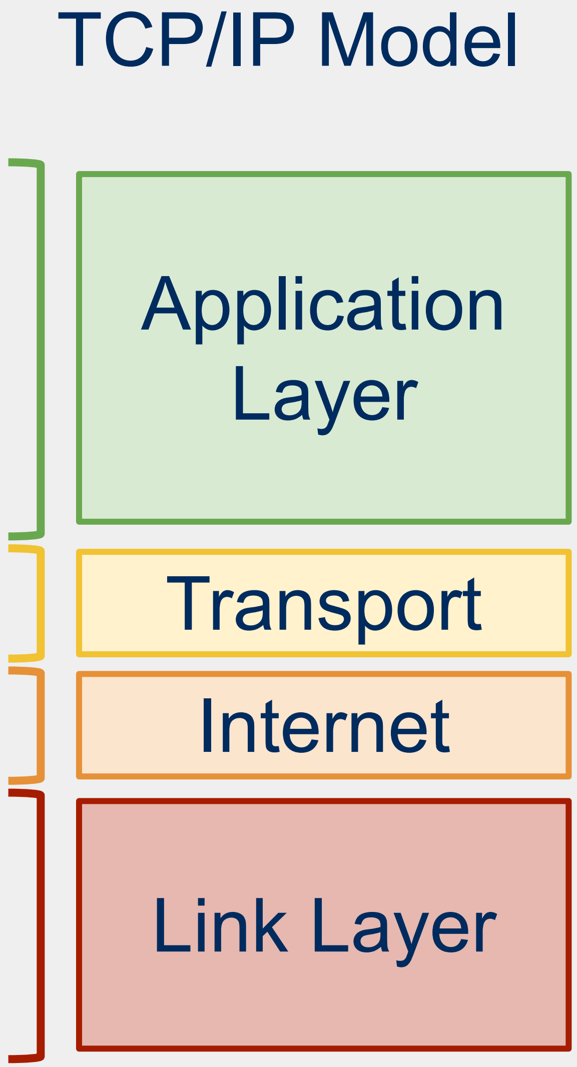
So the Host attribute of the request says where to go

- Well… no, not really…

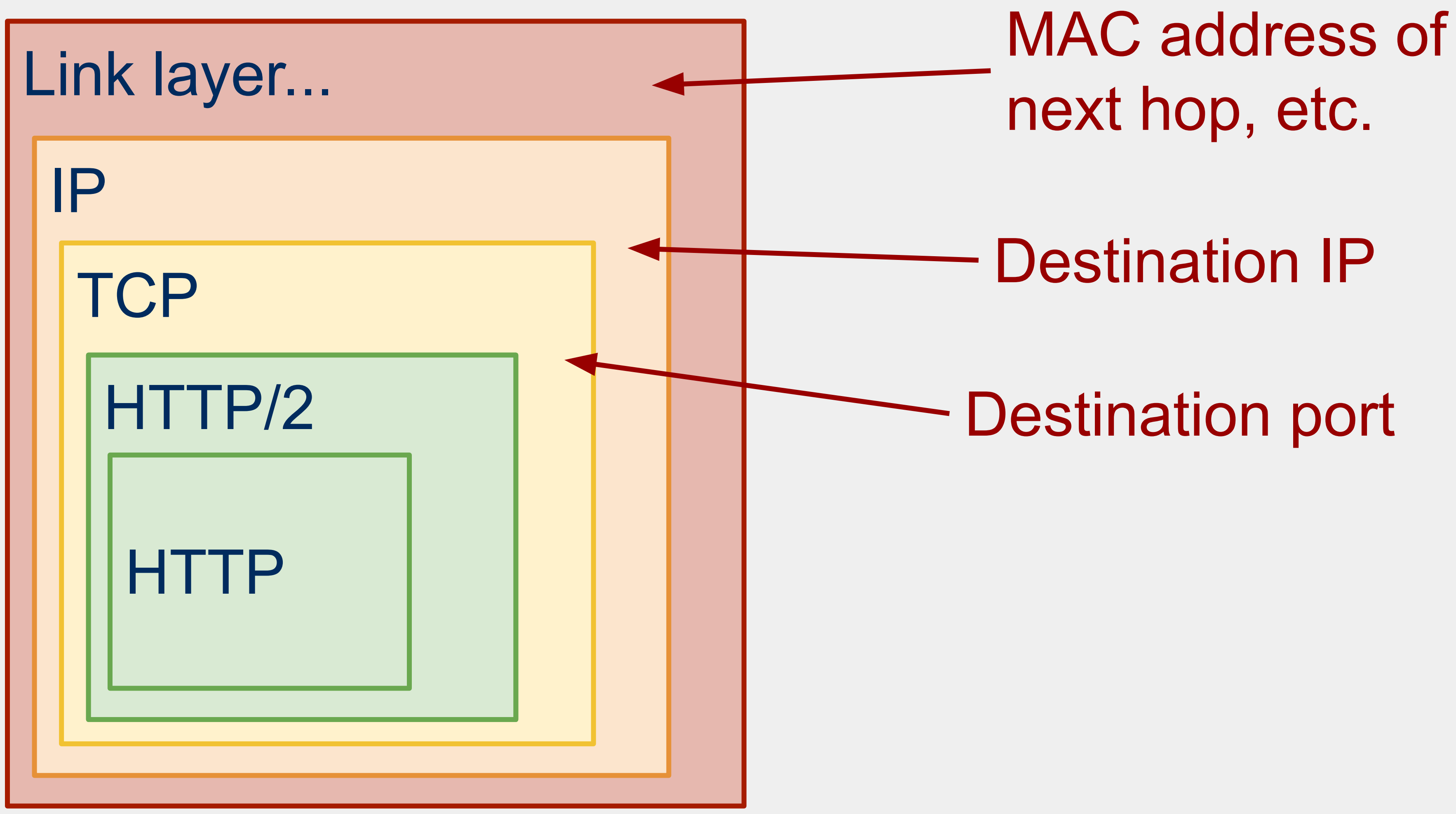

10.0.0.1
23
80
10.0.0.2
25
143
10.0.0.3
22
443
10.0.0.4
22
443

So how do we get HTTP requests to the webserver?


OSI Model
Data
Layer
Host Layers
Media Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections and Reliability
Packets
Network
Path Determination and IP (Logical Addressing)
Frames
Data Link
MAC and LLC (Physical Addressing)
Bits
Physical
Media, Signal, and Binary Transmission



To send an HTTP request



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 now 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
- URL format:

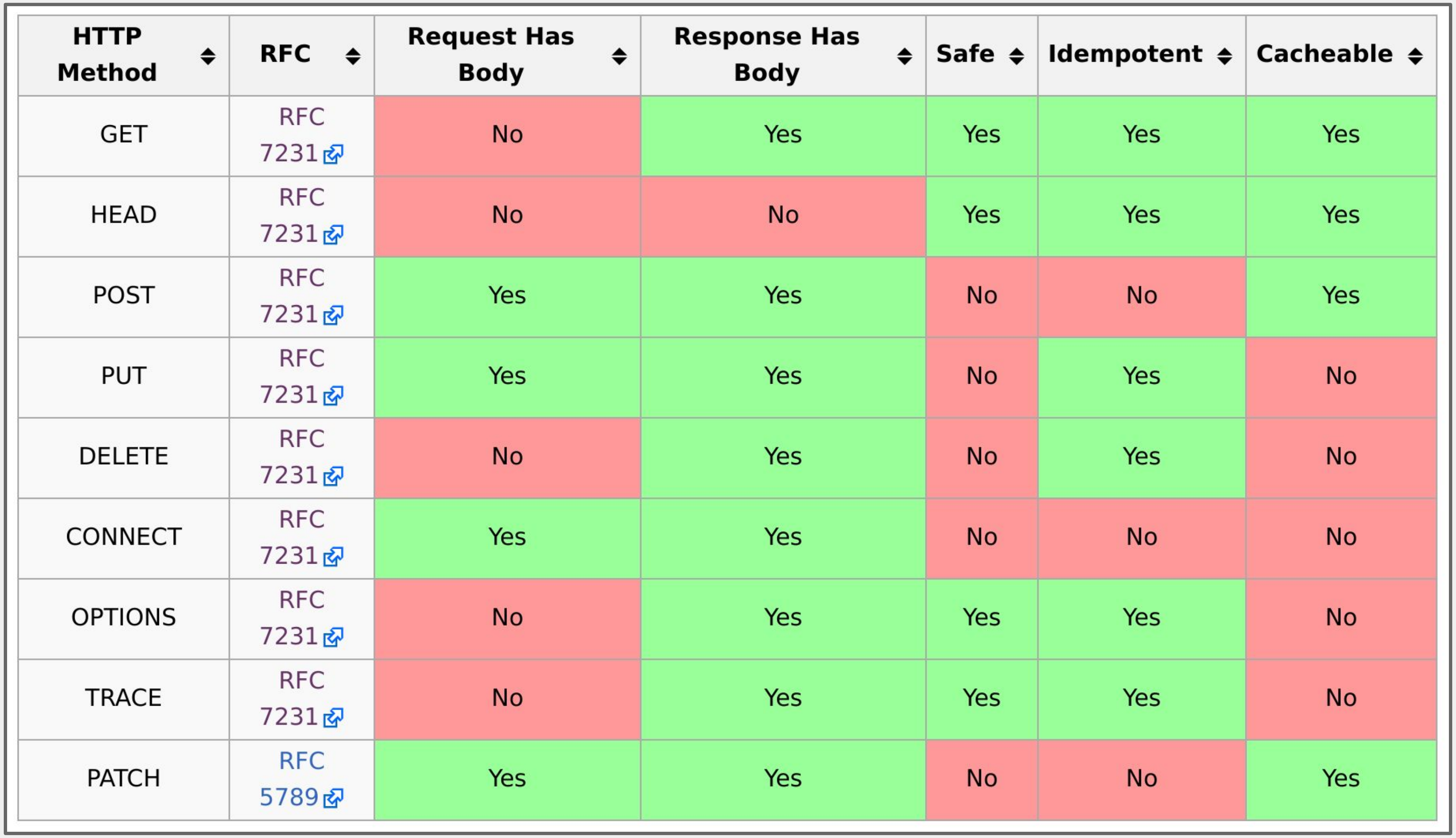
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

- The URL *query string* can be used to affect server state
- E.g.:
    - http://example.com/storefront?user=adam&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

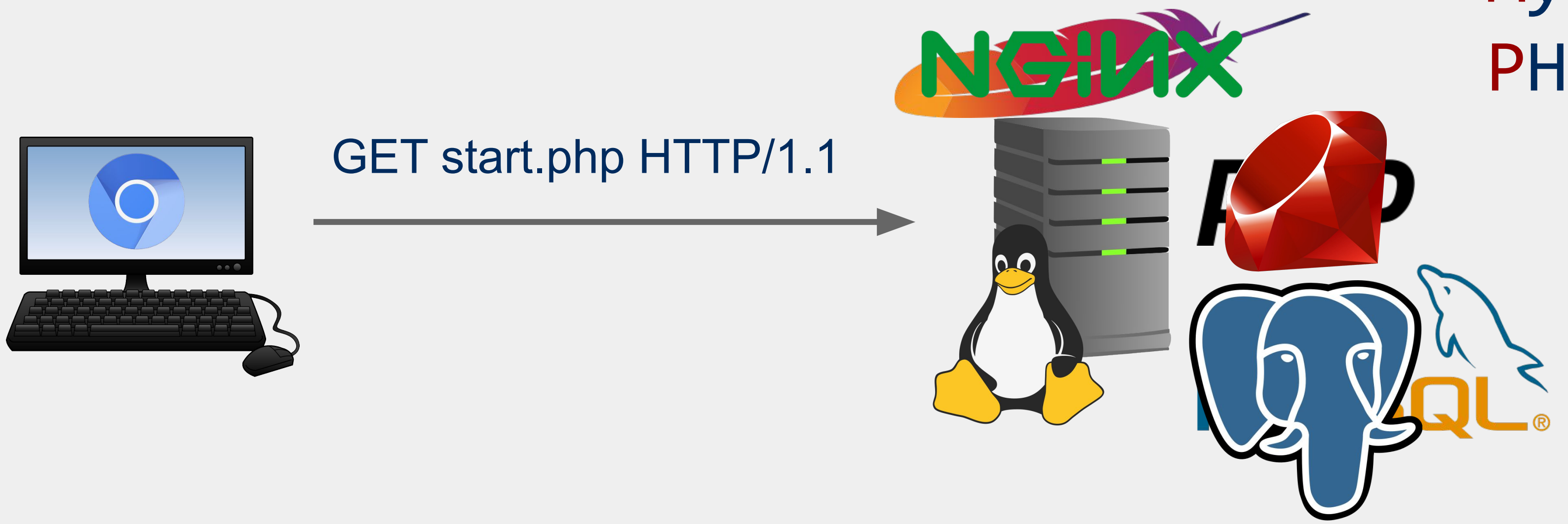
Comparisons of HTTP Methods



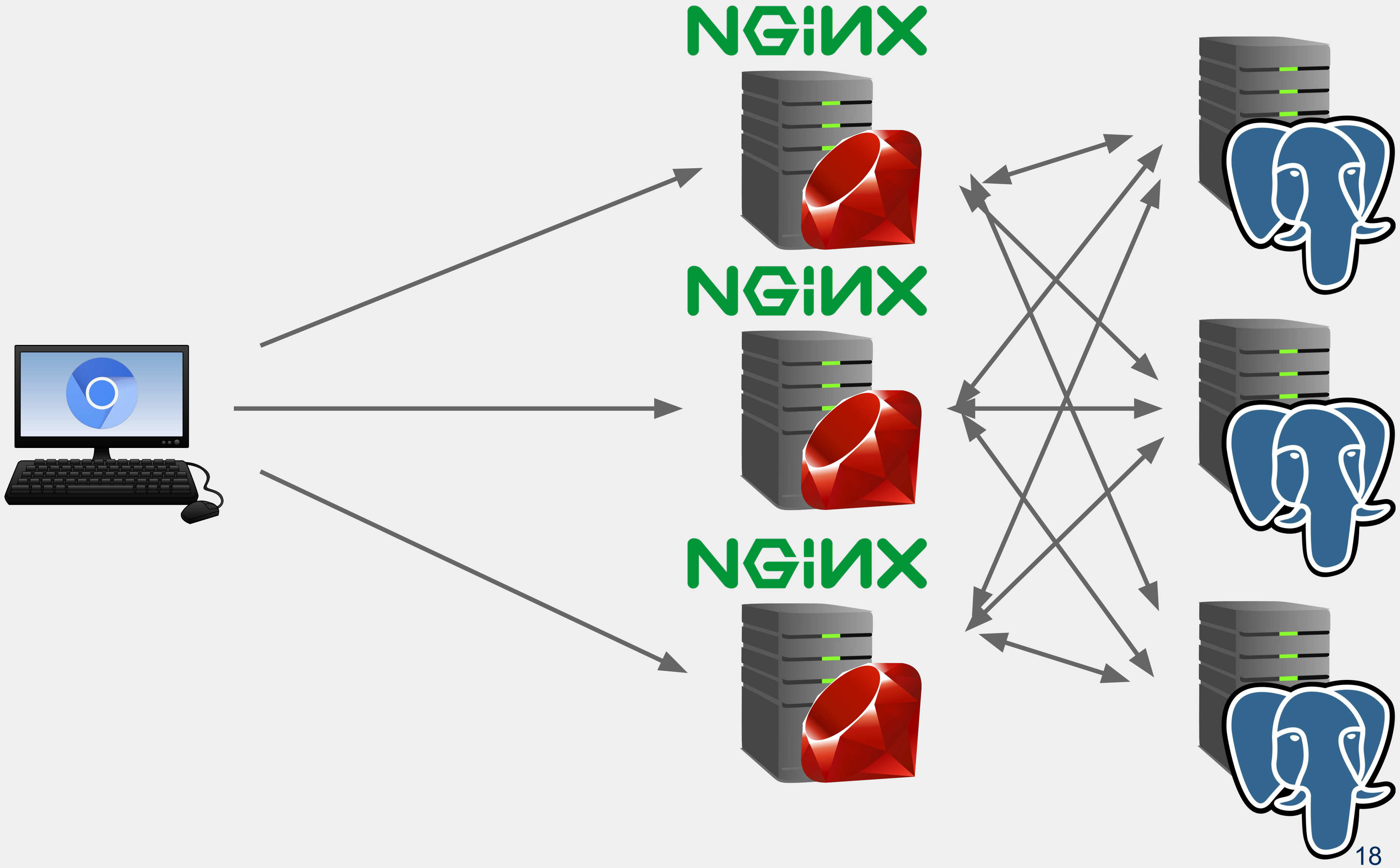
HTTP Method	RFC	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	No	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	No	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	Yes

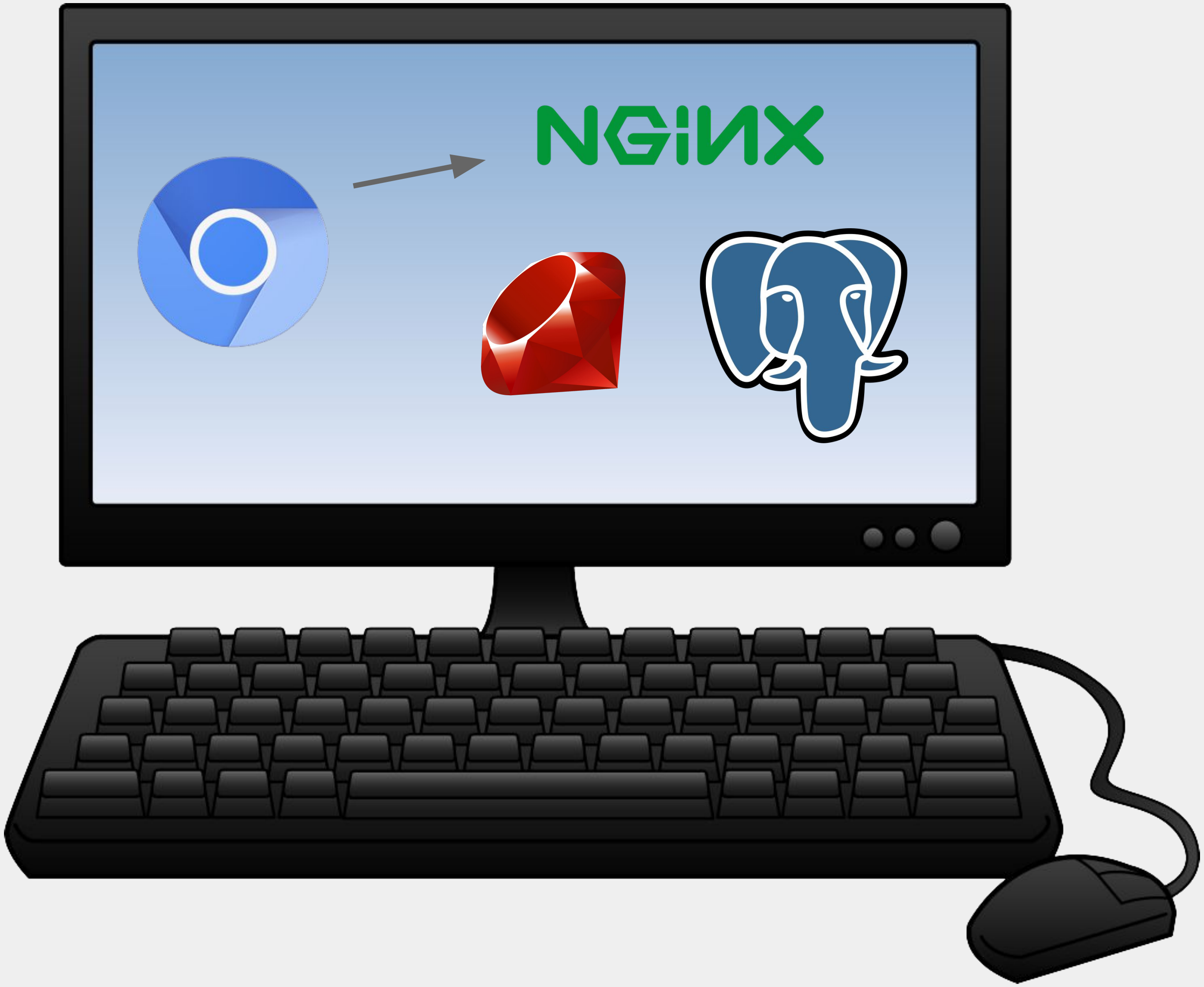
HTTP Status Codes

- 200
  - OK
- 301
  - Moved Permanently
- 400
  - Bad Request
- 403
  - Forbidden
- 404
  - Not Found
- 500
  - Internal Server Error
- …



Linux
Apache
MySQL
PHP


NGINX
NGINX
NGINX


NGINX