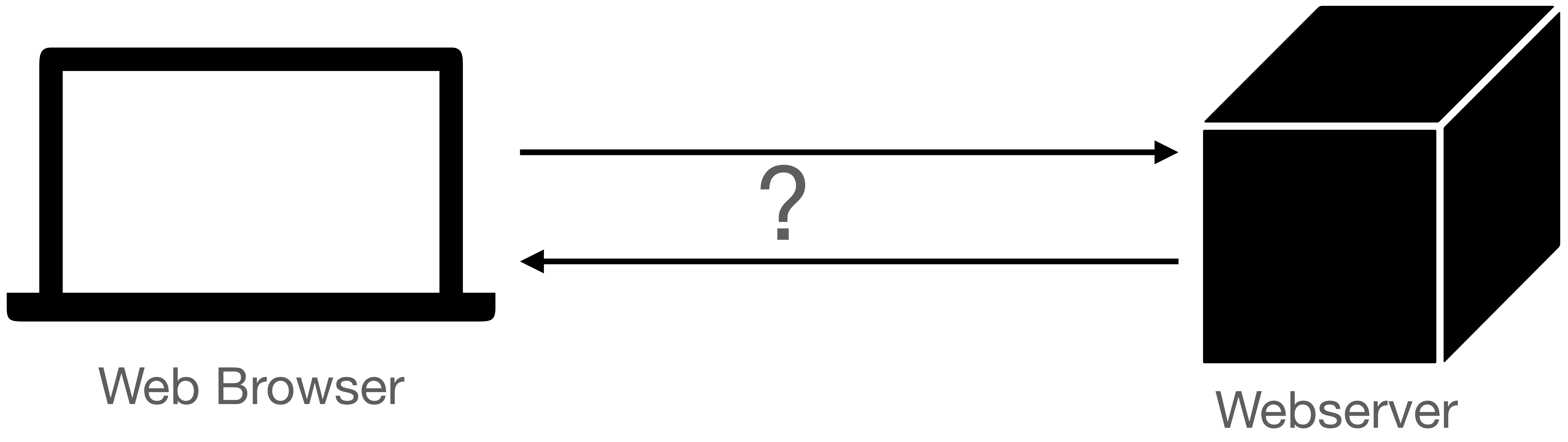
Week 15 Lab
Communicating With HTTP and REST

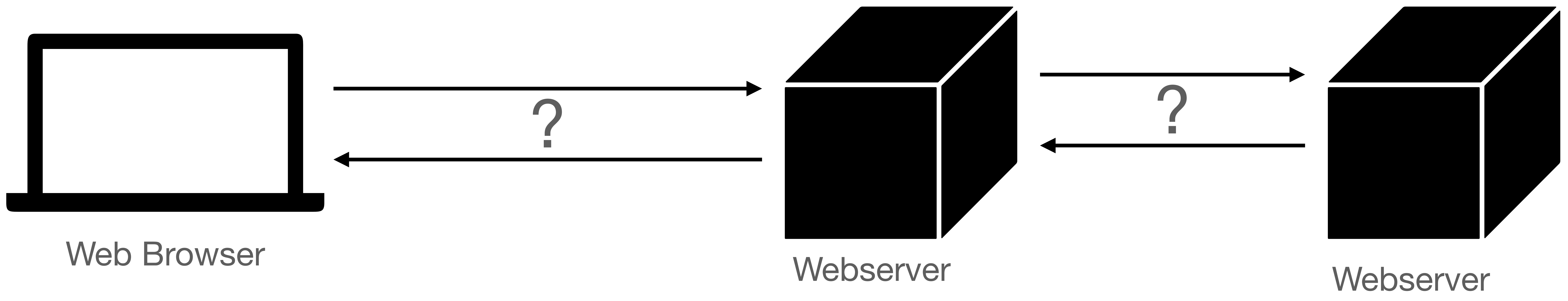
Chaufournier & Wood
CSCI 2541

We've talked about web servers and browsers

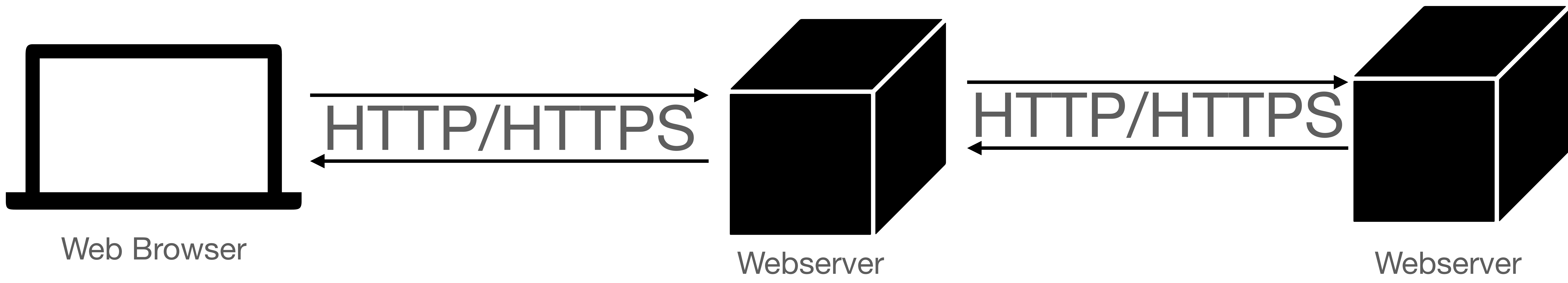
But not how they communicate with each other



Ideally we want something that covers both how clients talk to servers and how servers talk to each other



Ideally we want something that covers both how clients talk to servers and how servers talk to each other



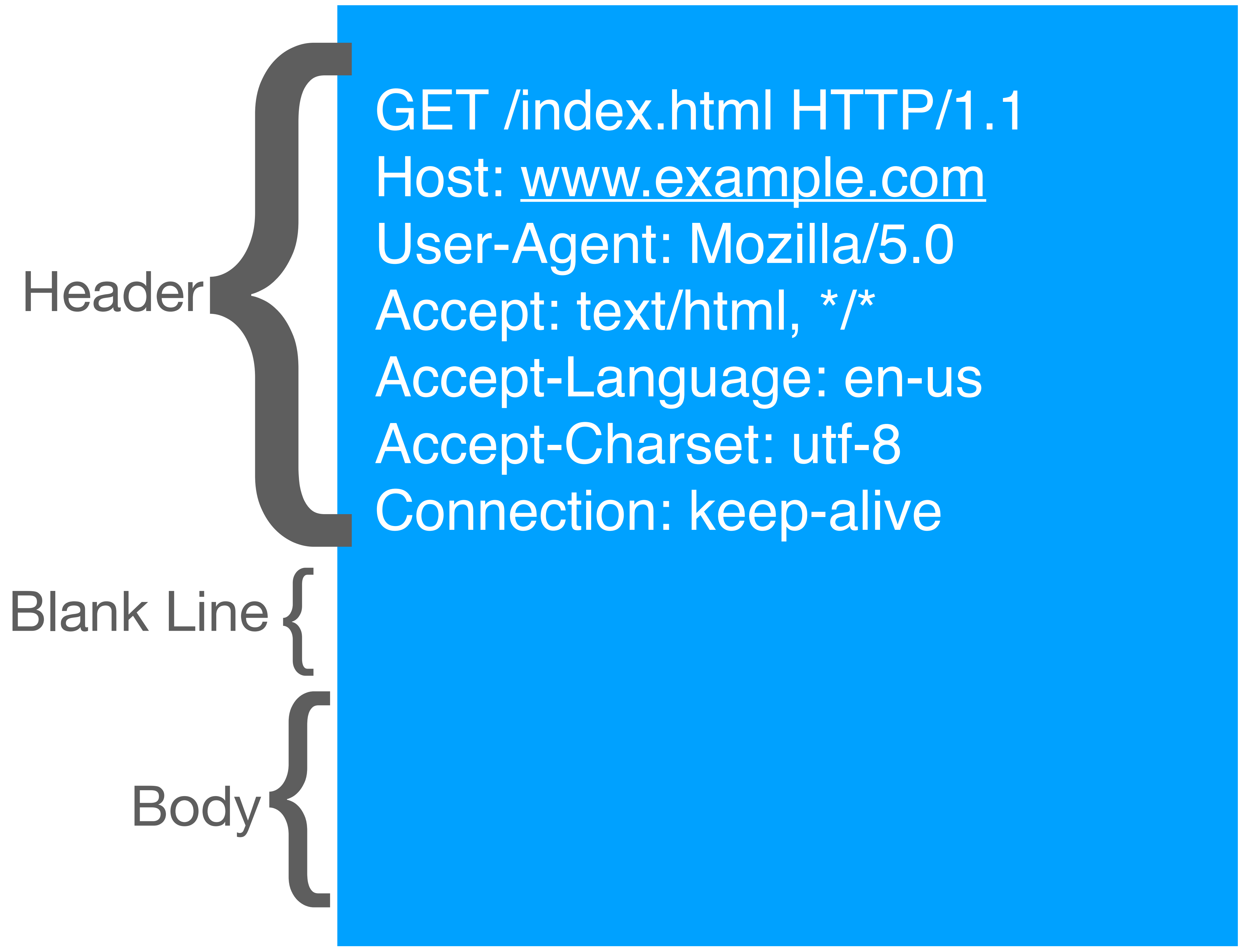
Hypertext Transfer Protocol

HTTP

- HTTP is versatile request/response communication standards
- Defined in RFC 1945 and RFC 2616, it proposed a set of methods for communicating across the web.
- Defined a standard set of request types and response codes that enables the various technologies across the web to communicate using a standard protocol.
- Defined the following methods GET, HEAD, POST, PUT, PATCH,DELETE,TRACE, CONNECT for performing different operations.

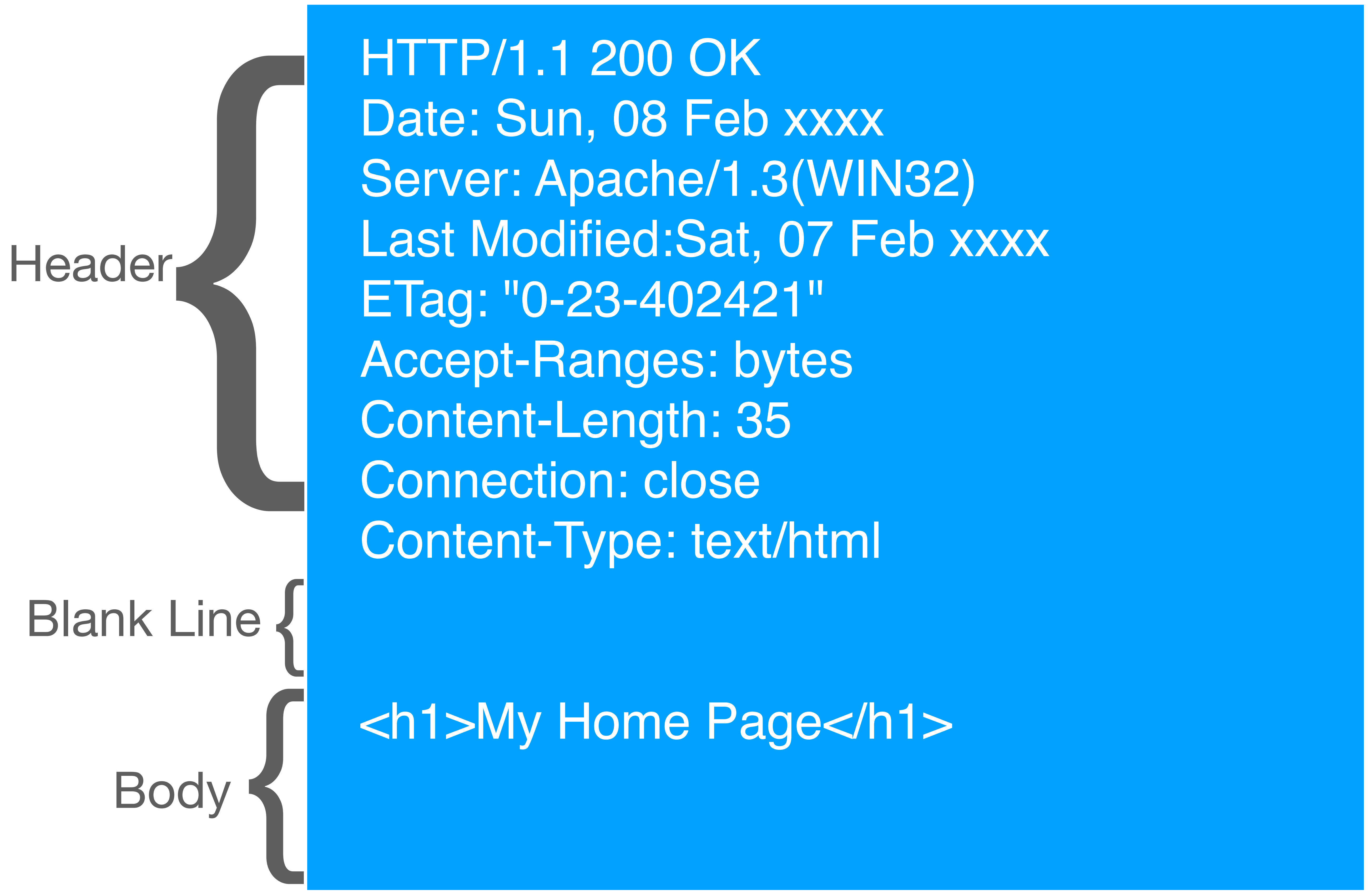
Fetching Data with GET

Format: GET URL Version



```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: utf-8
Connection: keep-alive
```

Sample Get Request

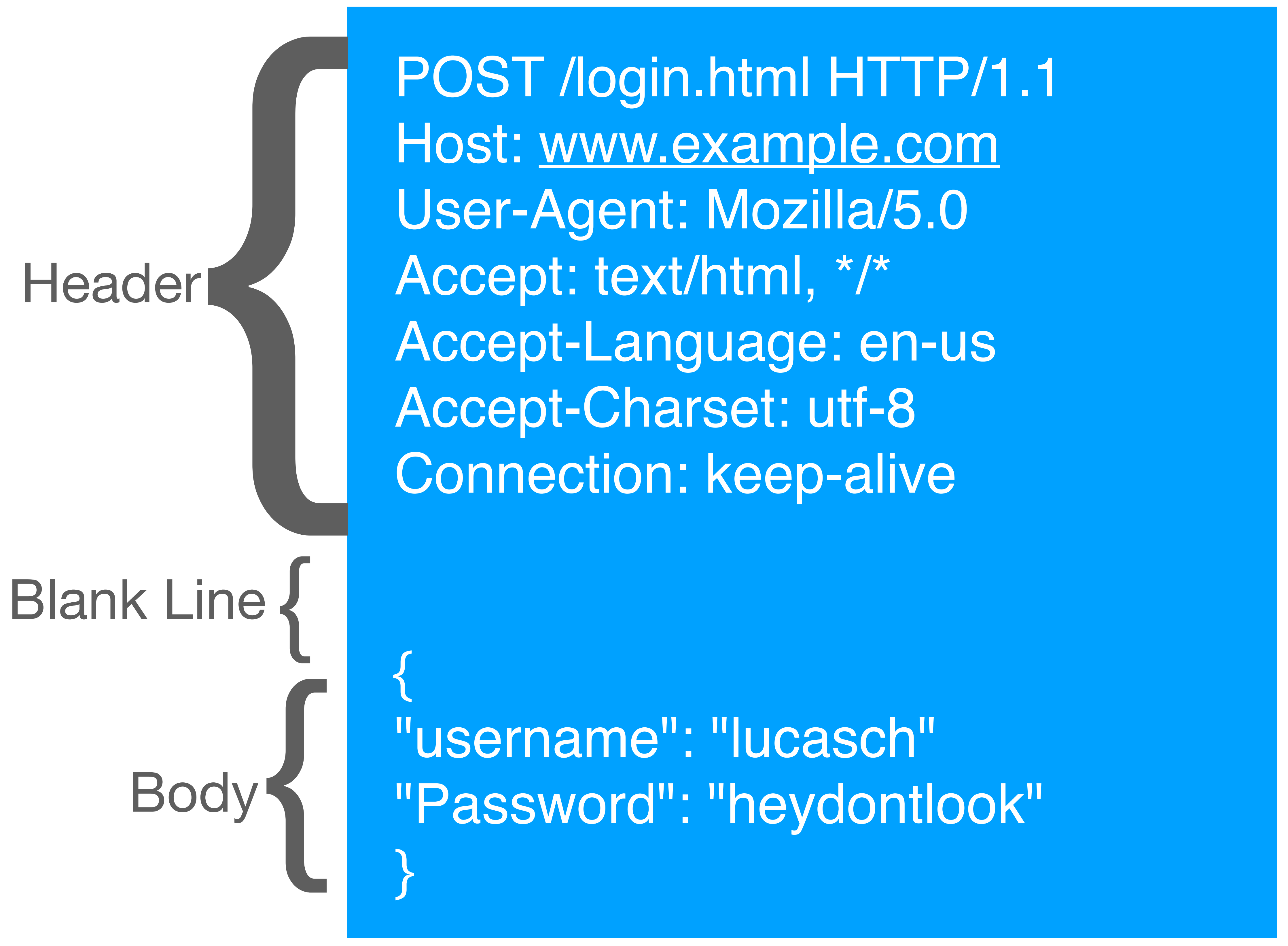


```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx
Server: Apache/1.3(WIN32)
Last Modified:Sat, 07 Feb xxxx
ETag: "0-23-402421"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home Page</h1>
```

Sample Response

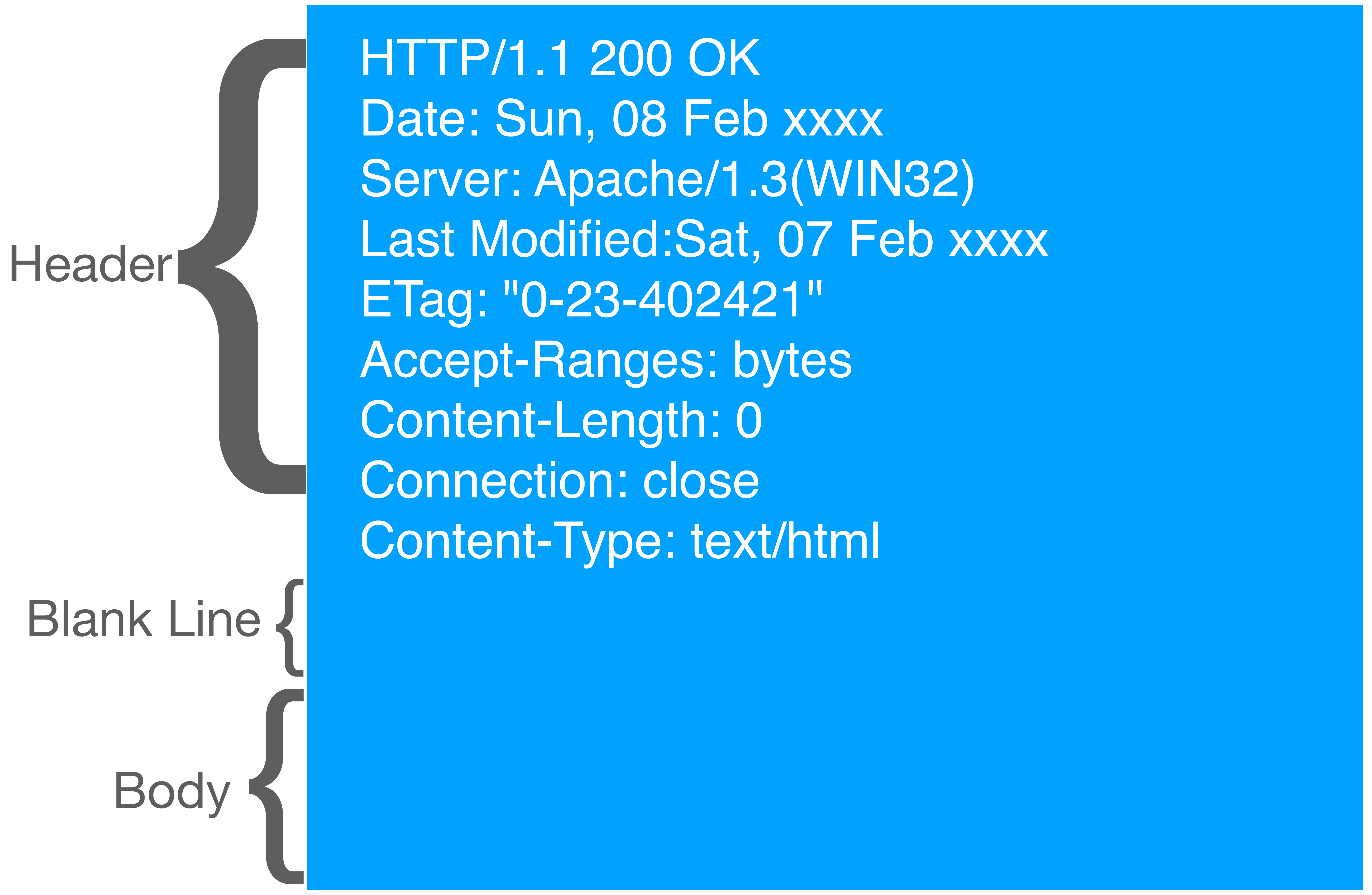
Sending Data with POST

Format: POST URL Version



```
POST /login.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: utf-8
Connection: keep-alive


{
"username": "lucasch"
"Password": "heydontlook"
}
```

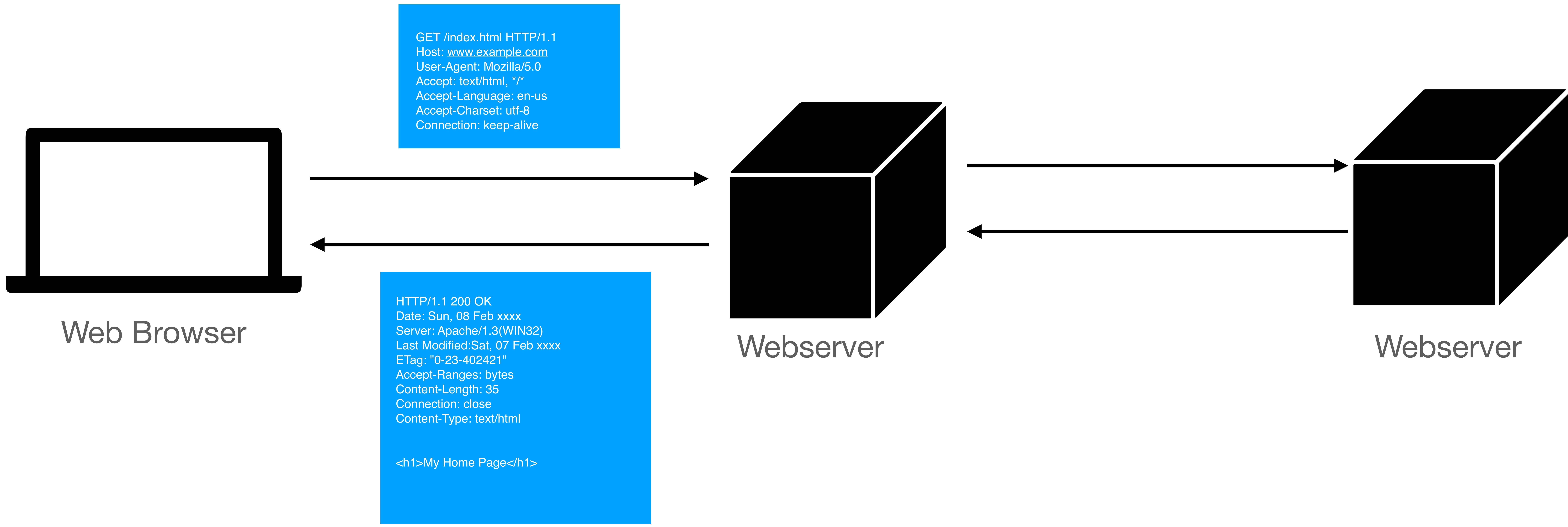
Sample POST Request



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx
Server: Apache/1.3(WIN32)
Last Modified:Sat, 07 Feb xxxx
ETag: "0-23-402421"
Accept-Ranges: bytes
Content-Length: 0
Connection: close
Content-Type: text/html
```

Response

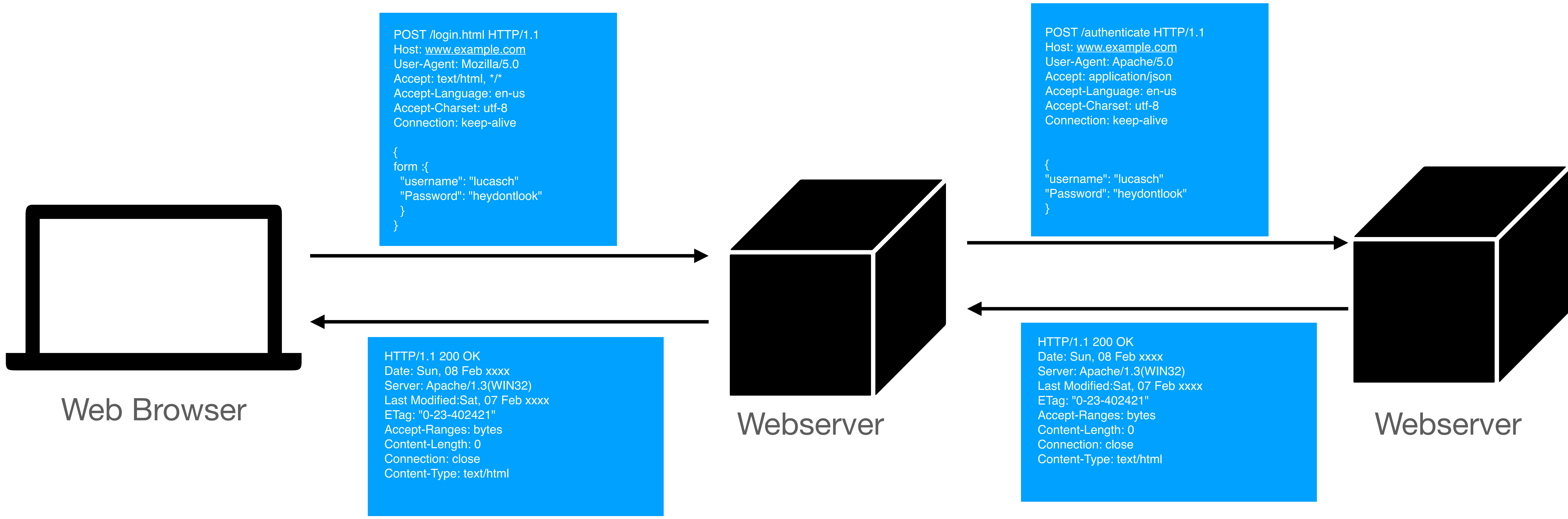
The web browser sends a request asking for the home page.



```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: utf-8
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx
Server: Apache/1.3(WIN32)
Last Modified:Sat, 07 Feb xxxx
ETag: "0-23-402421"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html


<h1>My Home Page</h1>
```

The Web Browser sends a request to login



```
POST /login.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: utf-8
Connection: keep-alive

{
form :{
  "username": "lucasch"
  "Password": "heydontlook"
  }
}
```

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx
Server: Apache/1.3(WIN32)
Last Modified:Sat, 07 Feb xxxx
ETag: "0-23-402421"
Accept-Ranges: bytes
Content-Length: 0
Connection: close
Content-Type: text/html
```

```
POST /authenticate HTTP/1.1
Host: www.example.com
User-Agent: Apache/5.0
Accept: application/json
Accept-Language: en-us
Accept-Charset: utf-8
Connection: keep-alive


{
"username": "lucasch"
"Password": "heydontlook"
}
```

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx
Server: Apache/1.3(WIN32)
Last Modified:Sat, 07 Feb xxxx
ETag: "0-23-402421"
Accept-Ranges: bytes
Content-Length: 0
Connection: close
Content-Type: text/html
```

HTTP Methods

There are several other methods that may be useful

- GET -- Used for fetching data. Not usually sent with a body.
- HEAD -- Used to fetch the header for a get request. Useful for figuring out how much information would be returned or information about the server.
- POST -- Used for sending data to a server or creating a new resource. Sent with a body that will be used to create the resource.
- PUT -- Used for updating data on a server by replacing what exists. Sent with a body that is used for the update.
- PATCH -- Used for updating a part of a resource without replacing the whole thing. Sent with a body containing just the piece to be updated.
- DELETE -- Used for deleting data on a server.

HTTP Response Codes

HTTP response codes provide a lot of information

- 200-299: Success codes
- 300-399: Redirects. A way for servers to tell you where you should make the next request.
- 400-499: Client Errors. The client did something wrong. Most common are 400 - Client Error, 401- Unauthorized, 404- Not Found.
- 500-599: Server Errors. The client request was ok but the server itself is broken. Most common are 500- Internal Server Error, 503- Bad gateway, 504 - Gateway Timeout

What does it mean to be RESTful?

RESTful Services

Representational **s**tate **t**ransfer (REST)

- A set of architectures and guiding principles for designing web scale resources.
- RESTful services typically use HTTP for communication.
- Promotes idea of statelessness
  - No data about clients are stored per request. Each request is treated as independent.
    - This allows for servers to be brought up and down without data loss or disruption.
  - As a result, each request needs to provide the full information about what it needs including authentication.
    - Requests can be sent over and over without needing to continue where you left off.
  - Applications will still have state such as databases but requests will be treated independently.

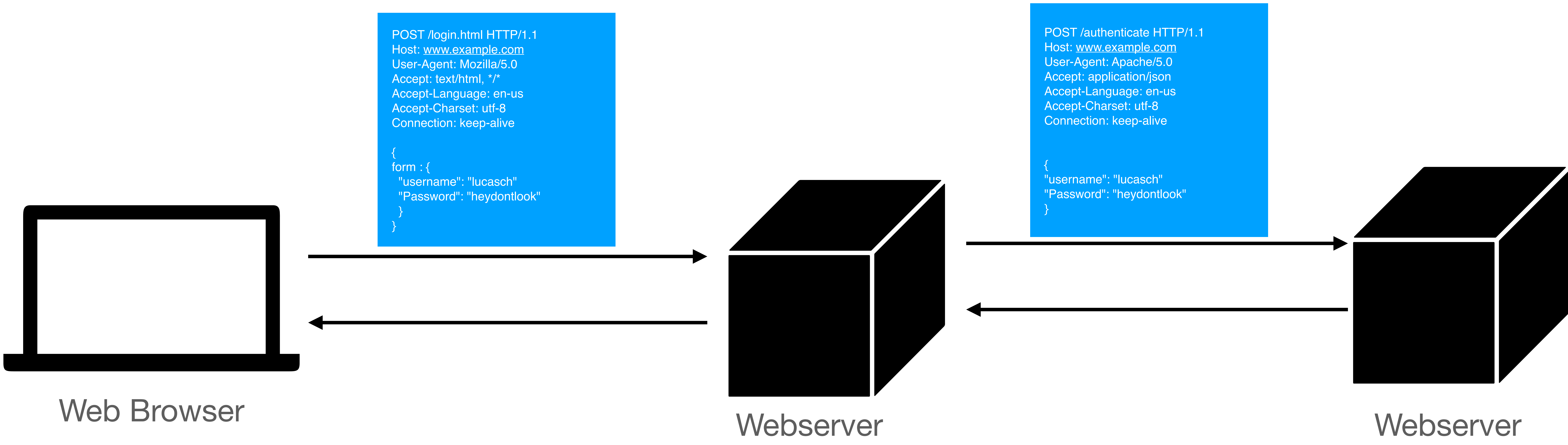
RESTful Services

Representational **s**tate **t**ransfer (REST)

- A set of architectures and guiding principles for designing web scale resources.
- RESTful services typically use HTTP for communication.
- Promotes idea of statelessness
    - No data about clients are stored per request. Each request is treated as independent.

Basic idea: use web technologies (e.g., HTTP, JSON) to make application interfaces (as opposed to ones directly viewed by clients on web browsers)

The Web Browser sends a request to login



```
POST /login.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: utf-8
Connection: keep-alive

{
form : {
  "username": "lucasch"
  "Password": "heydontlook"
  }
}
```

```
POST /authenticate HTTP/1.1
Host: www.example.com
User-Agent: Apache/5.0
Accept: application/json
Accept-Language: en-us
Accept-Charset: utf-8
Connection: keep-alive


{
"username": "lucasch"
"Password": "heydontlook"
}
```

RESTful says we need to use HTTP requests between servers. Data sent between servers should be JSON. Servers shouldn't remember anything about me.

How do we use RESTful concepts in python?

Communicating with other servers in Python

Issuing a GET Request

```
import requests

x = requests.get('https://mysamplepage.com/index.html')


if x.status_code > 300:
  print("Error making request")
print(x.text)
print(x.json())
```

Issue a GET request. Check the response code is under 200. Print JSON response.

Communicating with other servers in Python

Issuing a POST Request

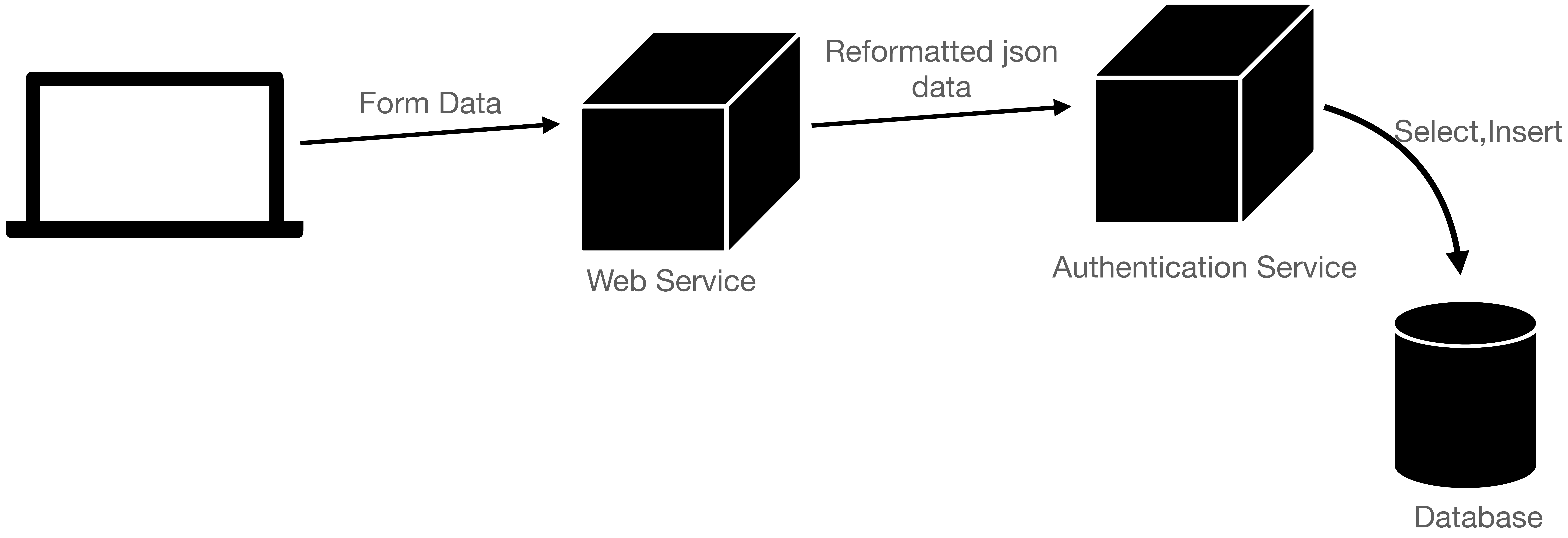
```
import requests
import json

myRequestBody = {'FirstName':"Lucas","LastName":"Chaufournier"}
headers = {"Content-Type": "application/json"}
r = requests.post('https://example.com/register', data = json.dumps(myRequestBody),
headers=headers)

if r.status_code > 300:
  print("ERROR in response")
else:
  print(r.json())
```

Issue a POST request with a python dict. Check the response code is under 200. Print JSON response.

RESTful Activity

- Clone these two repl.its:
  - https://replit.com/@thelimeburner/Authentication-Template
  - https://replit.com/@thelimeburner/AuthenticationService
- Implement a login restful service. You will have one service that takes requests from the form and one that verifies the data with the database.
- You will need to implement two post requests that support logging in and verifying a user as well as registering a user.
- Read the spec in AuthenticationService to know how to format your requests and parse the responses.

Login Architecture


Form Data
Reformatted json data
Select,Insert
Web Service
Authentication Service
Database