[301] Web 1

Tyler Caraza-Harter

Learning Objectives Today

Network basics

- IP addresses
- host/domain names
- client/server and request/response

HTTP basics

- URLs
- GET/POST/etc
- headers
- status codes

Requests modules

- downloading data with requests.get
- remote calls with requests.post

Learning Objectives Today

Motivation

Networking Basics

HTTP (Hypertext Transfer Protocol)

Requests Module

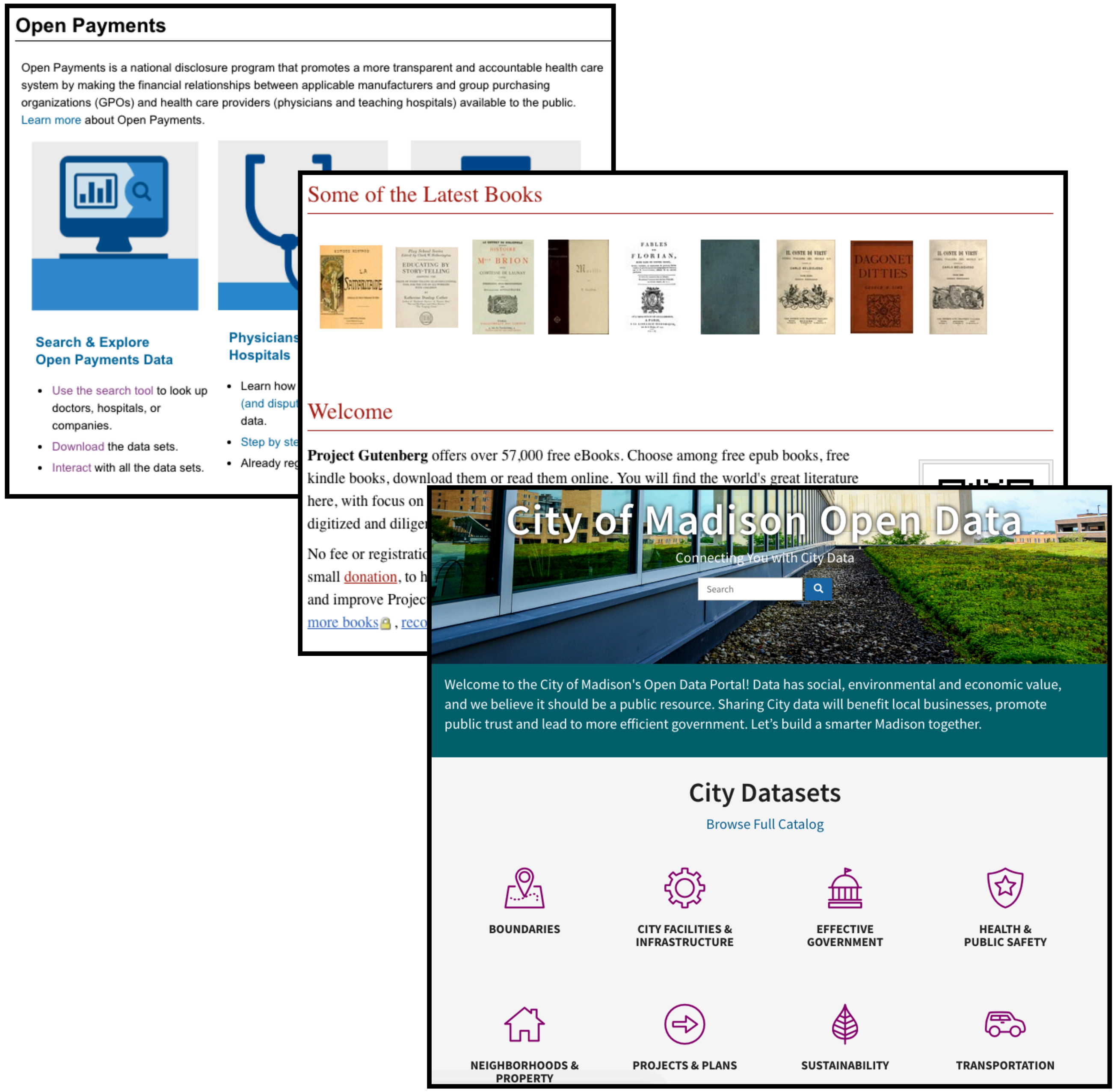
Data Science and the Internet

There are tons of online sources of data

- Examples: https://tyler.caraza-harter.com/cs301/spring19/datasets.html

Wide range of topics

- healthcare
- roads and city planning
- astronomy
- population
- business
- entertainment
- education
- etc



Open Payments

Open Payments is a national disclosure program that promotes a more transparent and accountable health care system by making the financial relationships between applicable manufacturers and group purchasing organizations (GPOs) and health care providers (physicians and teaching hospitals) available to the public. Learn more about Open Payments.

Search & Explore Open Payments Data

- Use the search tool to look up doctors, hospitals, or companies.
- Download the data sets.
- Interact with all the data sets.

Physicians Hospitals

- Learn how (and disput data.
- Step by ste
- Already reg

Some of the Latest Books

Welcome

Project Gutenberg offers over 57,000 free eBooks. Choose among free epub books, free kindle books, download them or read them online. You will find the world's great literature here, with focus on digitized and diliger

No fee or registratio small donation, to h and improve Projec more books, reco

City of Madison Open Data

Connecting You with City Data

Welcome to the City of Madison's Open Data Portal! Data has social, environmental and economic value, and we believe it should be a public resource. Sharing City data will benefit local businesses, promote public trust and lead to more efficient government. Let's build a smarter Madison together.

City Datasets

Browse Full Catalog

Data Science and the Internet

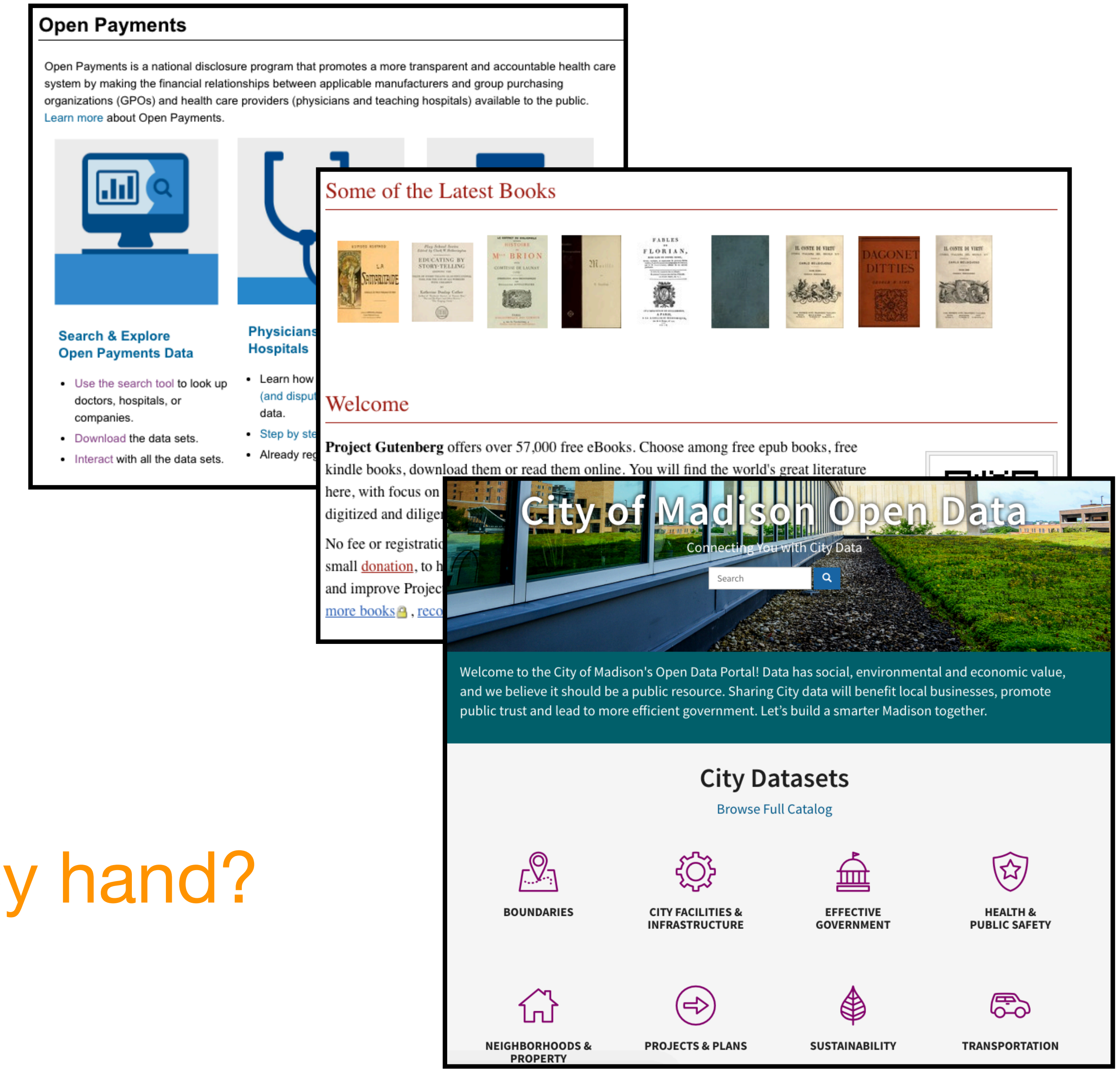
There are tons of online sources of data

- Examples: https://tyler.caraza-harter.com/cs301/spring19/datasets.html

Wide range of topics

- healthcare
- roads and city planning
- astronomy
- population
- business
- entertainment
- education
- etc

Why not just download data by hand?



Open Payments

Open Payments is a national disclosure program that promotes a more transparent and accountable health care system by making the financial relationships between applicable manufacturers and group purchasing organizations (GPOs) and health care providers (physicians and teaching hospitals) available to the public. Learn more about Open Payments.

Search & Explore Open Payments Data

- Use the search tool to look up doctors, hospitals, or companies.
- Download the data sets.
- Interact with all the data sets.

Physicians Hospitals

- Learn how (and disput data.
- Step by ste
- Already reg

Some of the Latest Books

Welcome

Project Gutenberg offers over 57,000 free eBooks. Choose among free epub books, free kindle books, download them or read them online. You will find the world's great literature here, with focus on digitized and diliger

No fee or registratio small donation, to h and improve Projec more books, reco

City of Madison Open Data

Connecting You with City Data

Welcome to the City of Madison's Open Data Portal! Data has social, environmental and economic value, and we believe it should be a public resource. Sharing City data will benefit local businesses, promote public trust and lead to more efficient government. Let’s build a smarter Madison together.

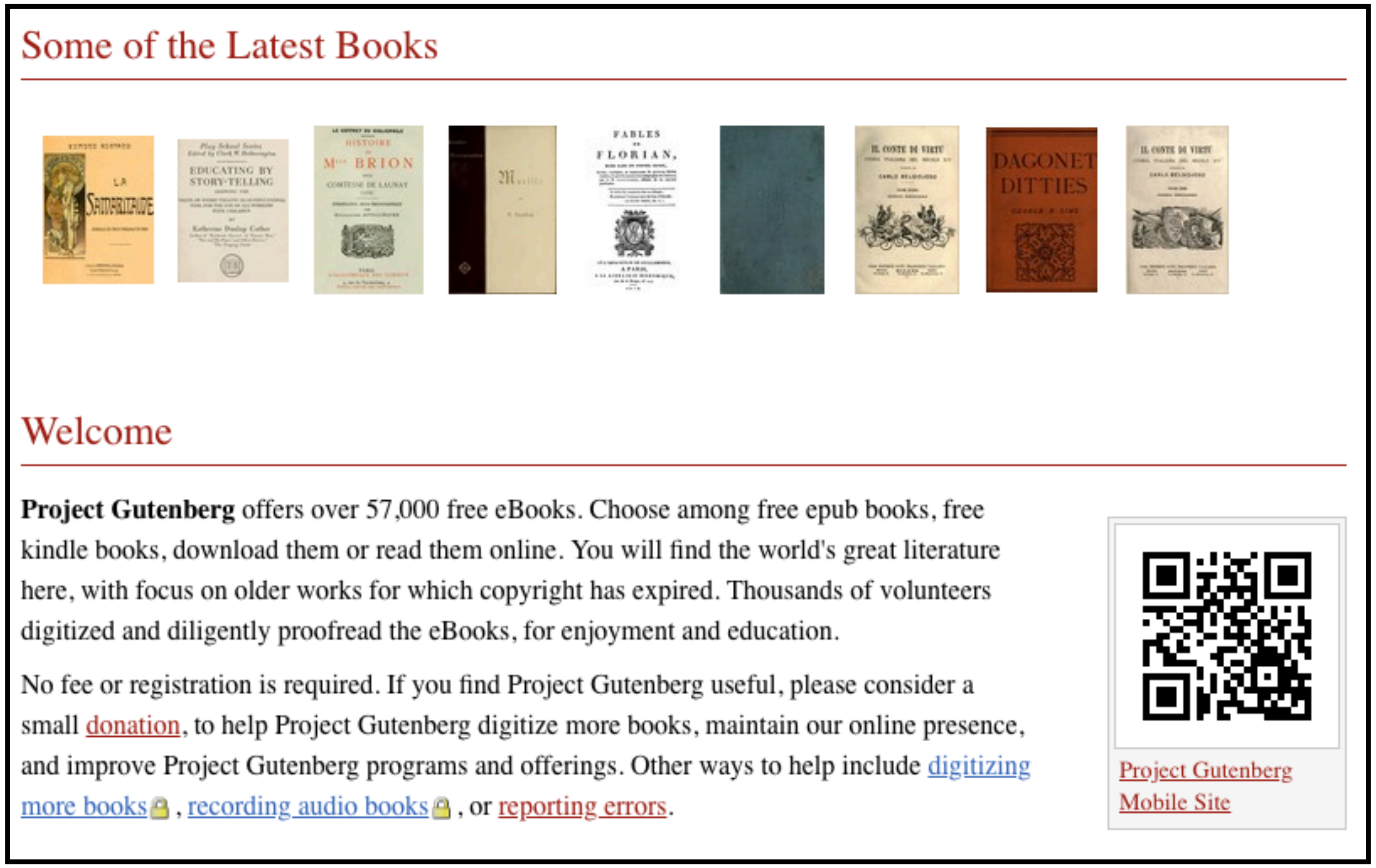
City Datasets

Browse Full Catalog

Motivation 1: too much data

What if you're analyzing language trends over time?

- Dataset: Project Gutenberg has 57K free books
- Too much work to download one by one



Some of the Latest Books

Welcome

Project Gutenberg offers over 57,000 free eBooks. Choose among free epub books, free kindle books, download them or read them online. You will find the world's great literature here, with focus on older works for which copyright has expired. Thousands of volunteers digitized and diligently proofread the eBooks, for enjoyment and education.

No fee or registration is required. If you find Project Gutenberg useful, please consider a small donation, to help Project Gutenberg digitize more books, maintain our online presence, and improve Project Gutenberg programs and offerings. Other ways to help include digitizing more books, recording audio books, or reporting errors.

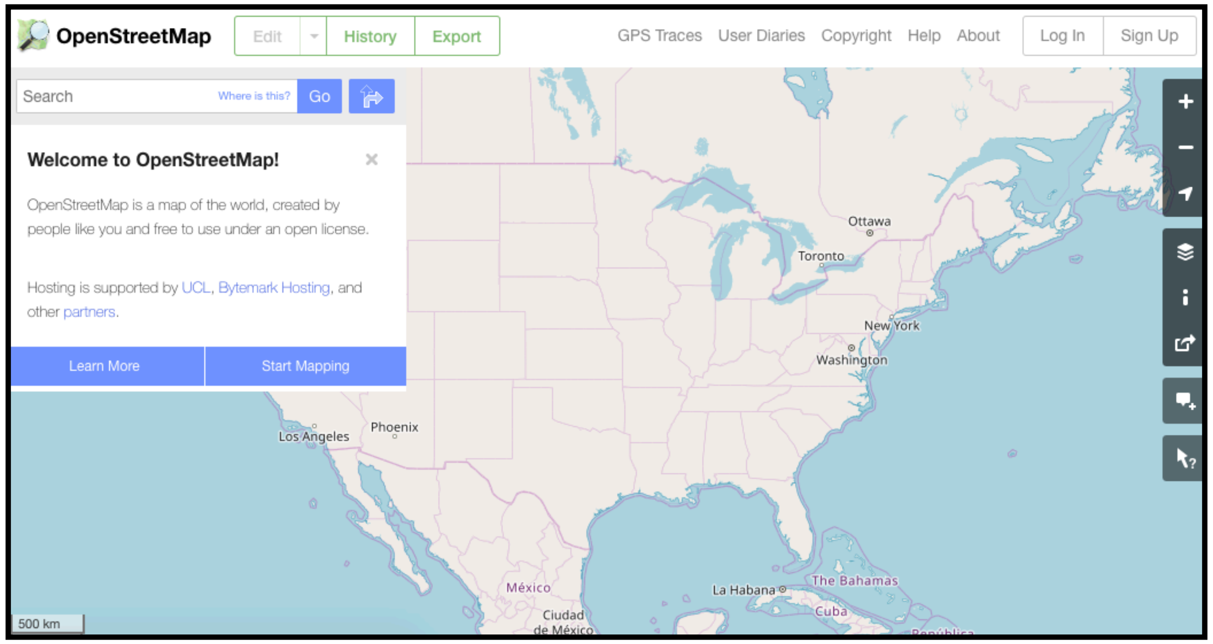
Project Gutenberg Mobile Site

Motivation 2: data doesn’t always come in files

Many datasets are difficult to download complete

Instead, you can make function calls to servers (we’ll learn how) to grab specific data

- Dataset: OpenStreetMap
- You issue calls to get specific data:
    1. specify latitude/longitude rectangle
    2. specify structures of interest (e.g., bike paths)



Welcome to OpenStreetMap!

OpenStreetMap is a map of the world, created by people like you and free to use under an open license.

Hosting is supported by UCL, Bytemark Hosting, and other partners.

Learning Objectives Today

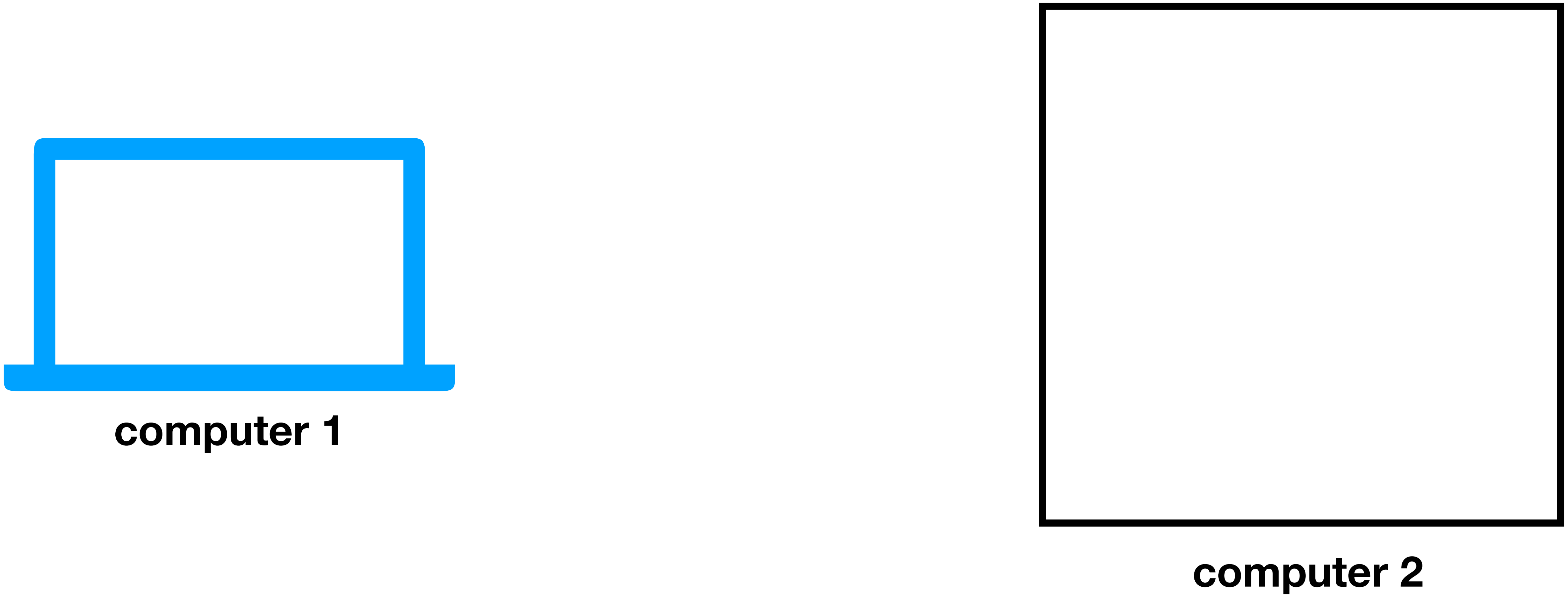
Motivation

Networking Basics

HTTP (Hypertext Transfer Protocol)

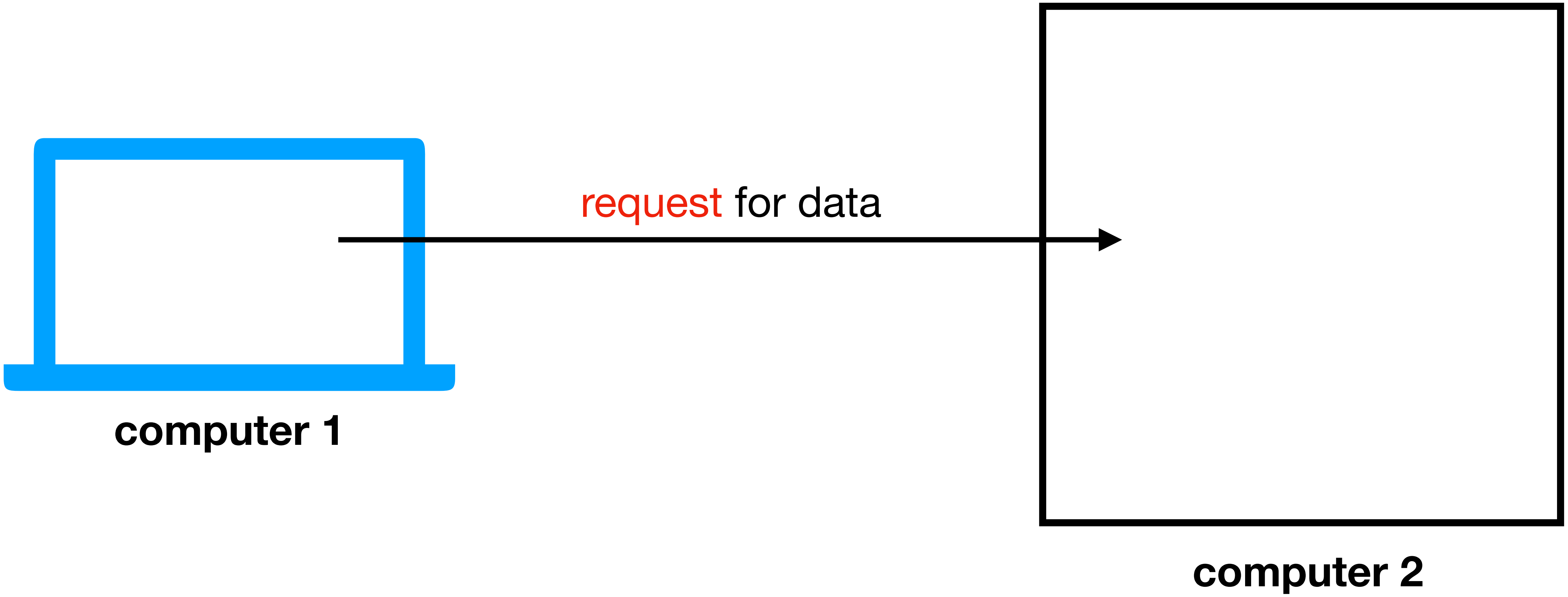
Requests Module

Networking Basics



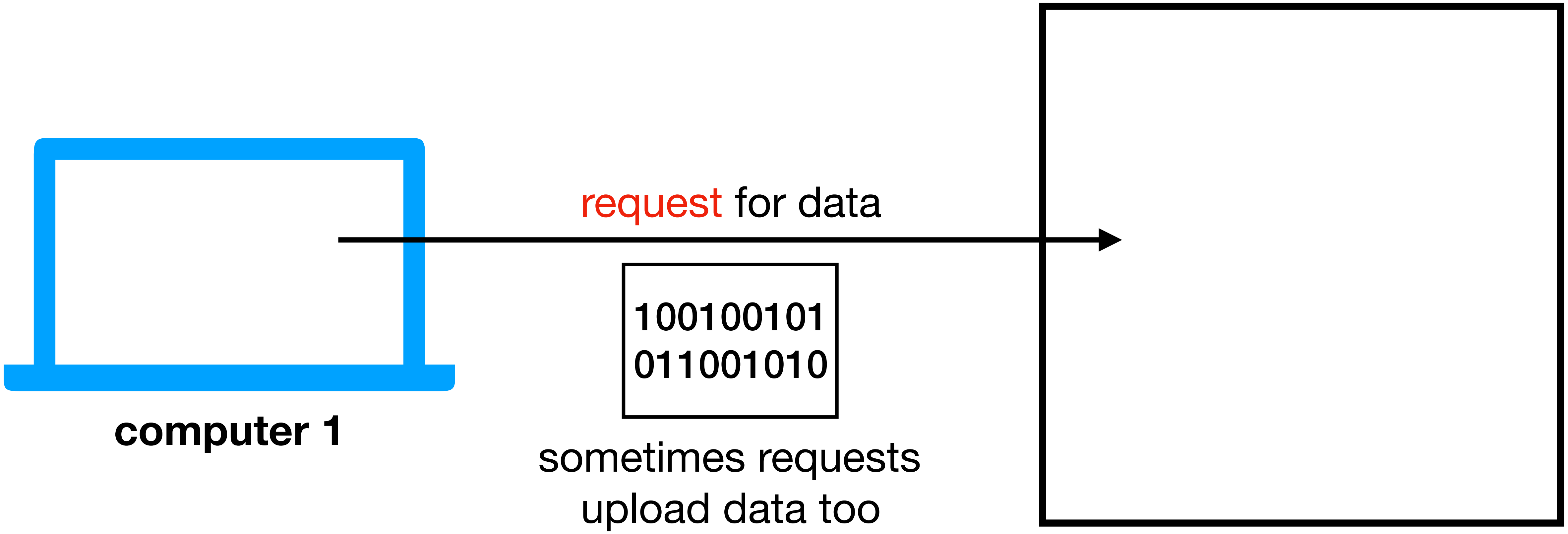
Computers communicate over a network (e.g., the Internet)
by sending messages to each other

Networking Basics



Computers communicate over a network (e.g., the Internet)
by sending messages to each other

Networking Basics



request for data

100100101
011001010

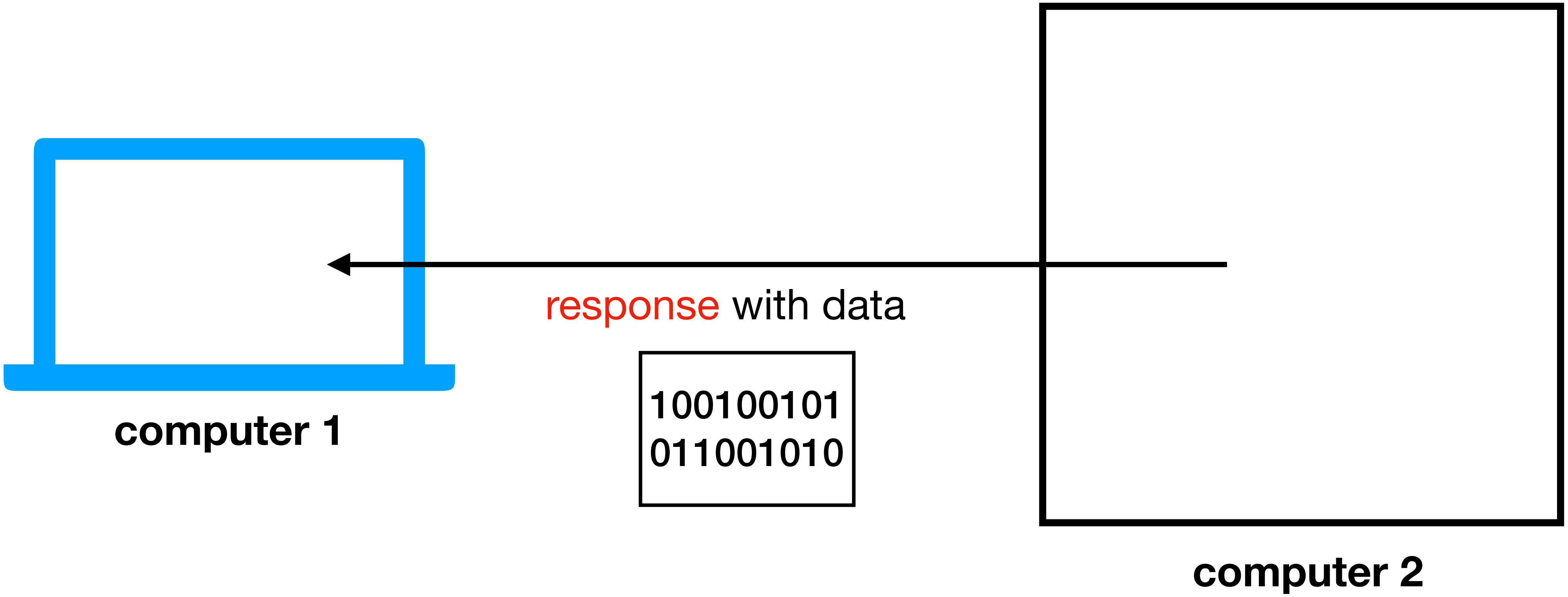
computer 1

sometimes requests upload data too

computer 2

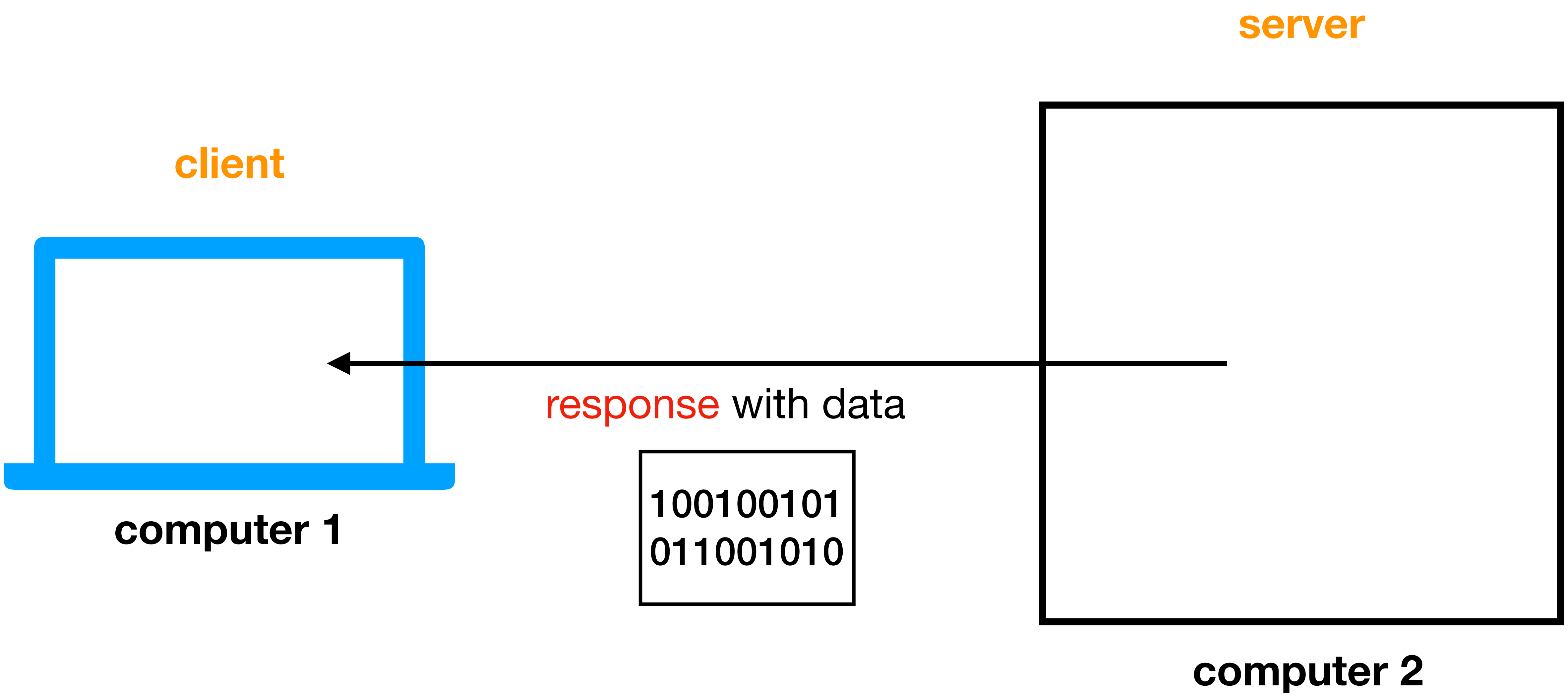
Computers communicate over a network (e.g., the Internet) by sending messages to each other

Networking Basics



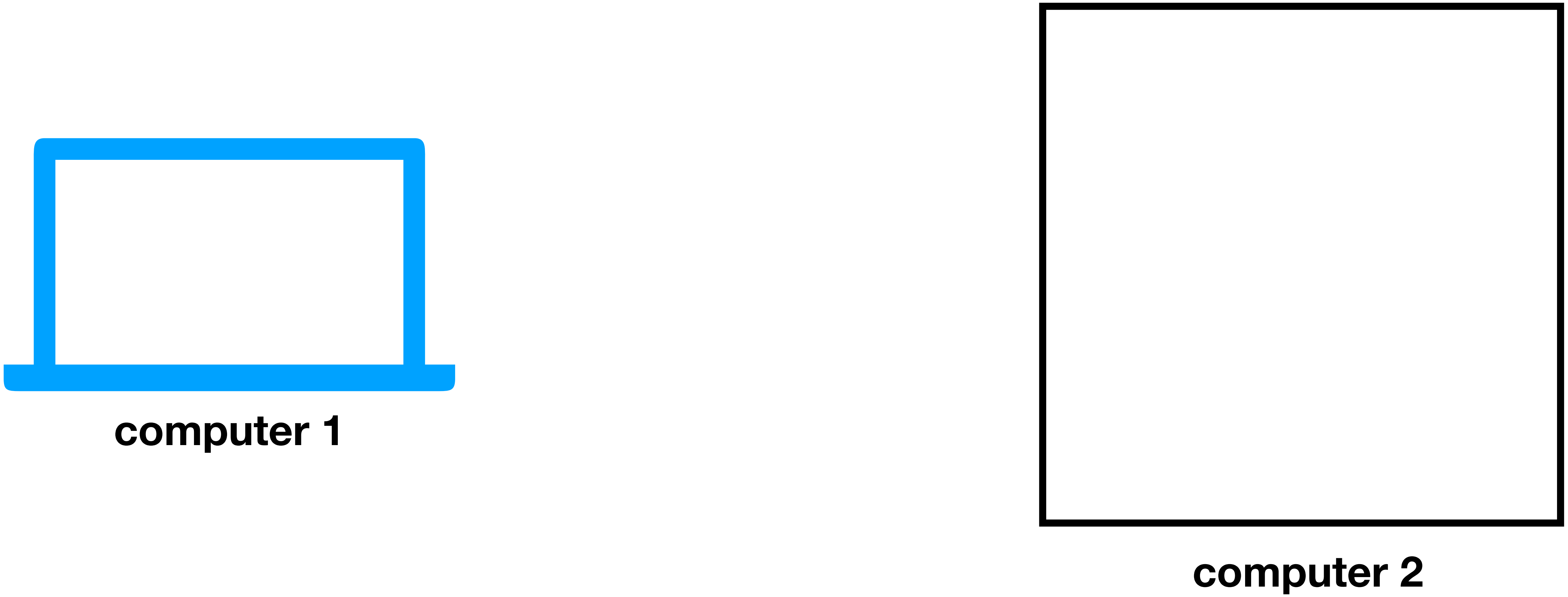
Computers communicate over a network (e.g., the Internet)
by sending messages to each other

Networking Basics



Computers communicate over a network (e.g., the Internet)
by sending messages to each other

Networking Basics

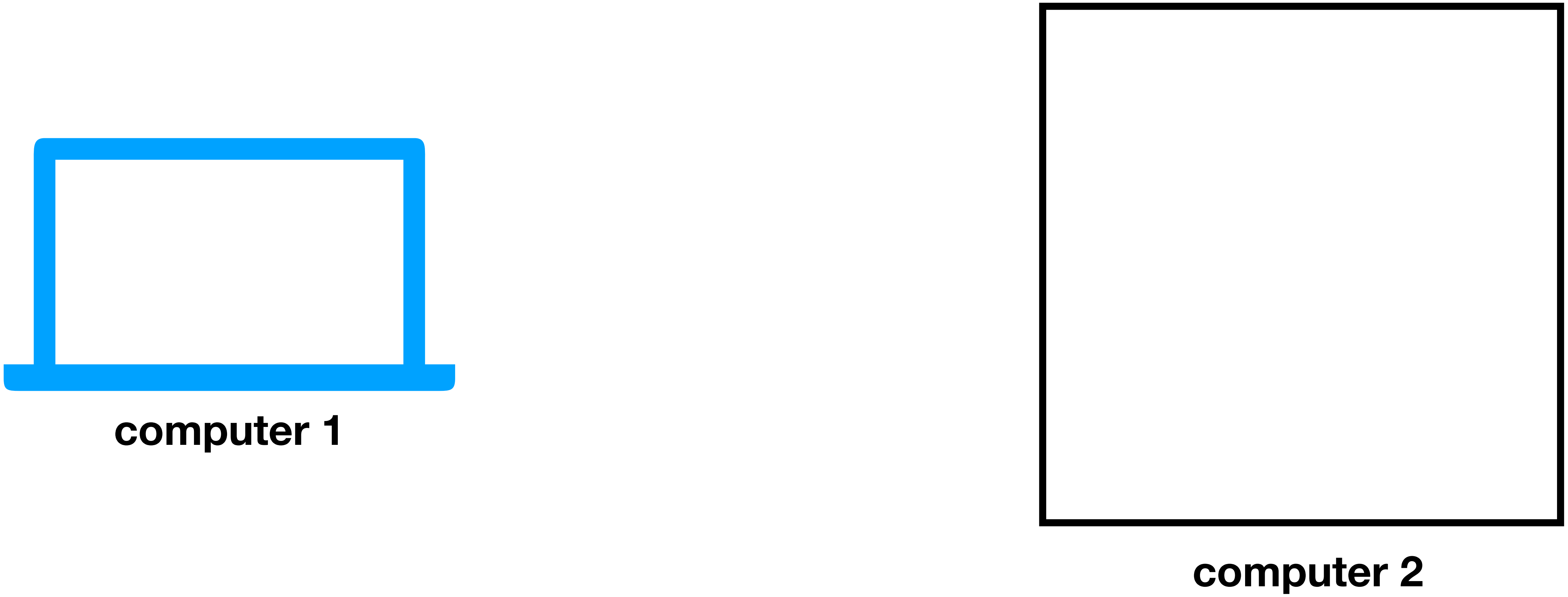


Challenge: there are millions of computers.
How do we indicate which machine should get our request?

How do we send a letter?

1. lookup friend's address in phone book
2. put address on the envelope
3. trust postal service to get letter to that address

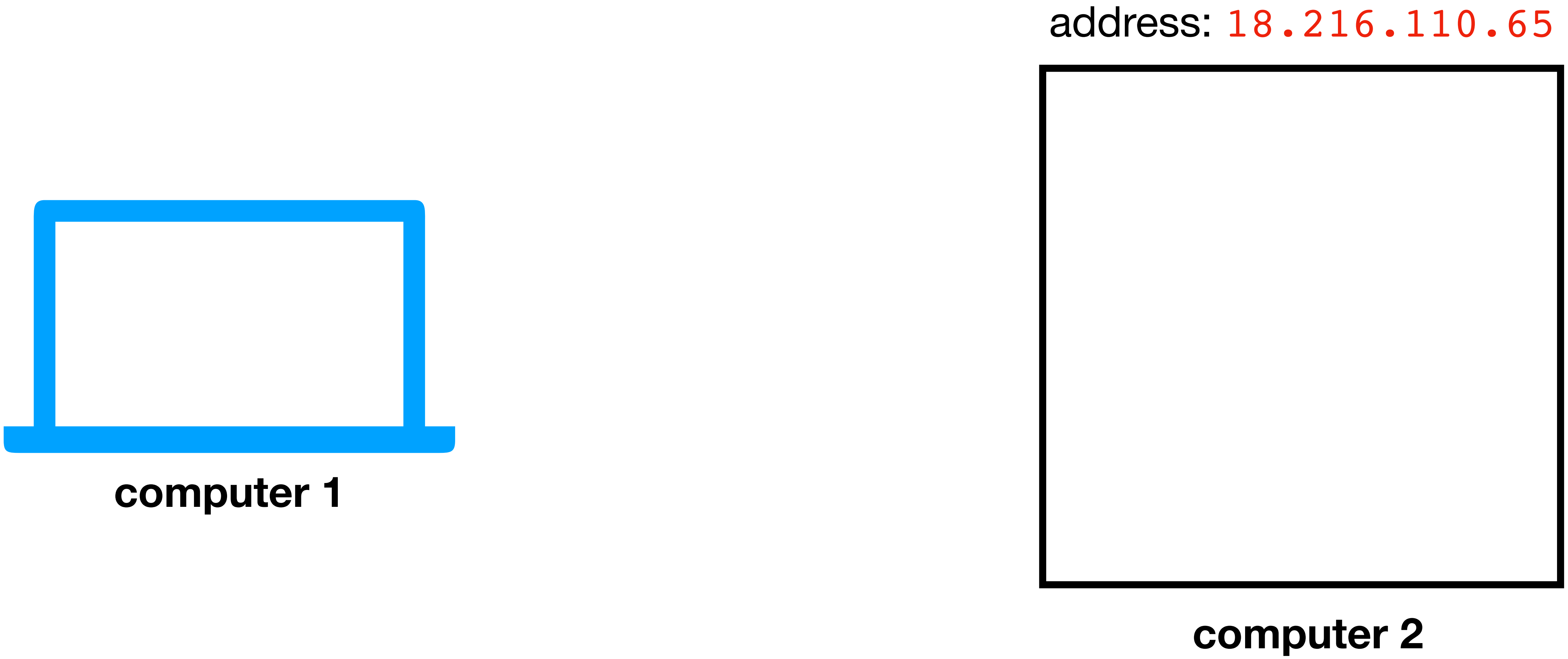
Internet Protocol



Solution: every machine* has an IP address (Internet Protocol).
Requests are sent to a specific IP address.

*some machines have more multiple addresses

Internet Protocol



address: 18.216.110.65

computer 1

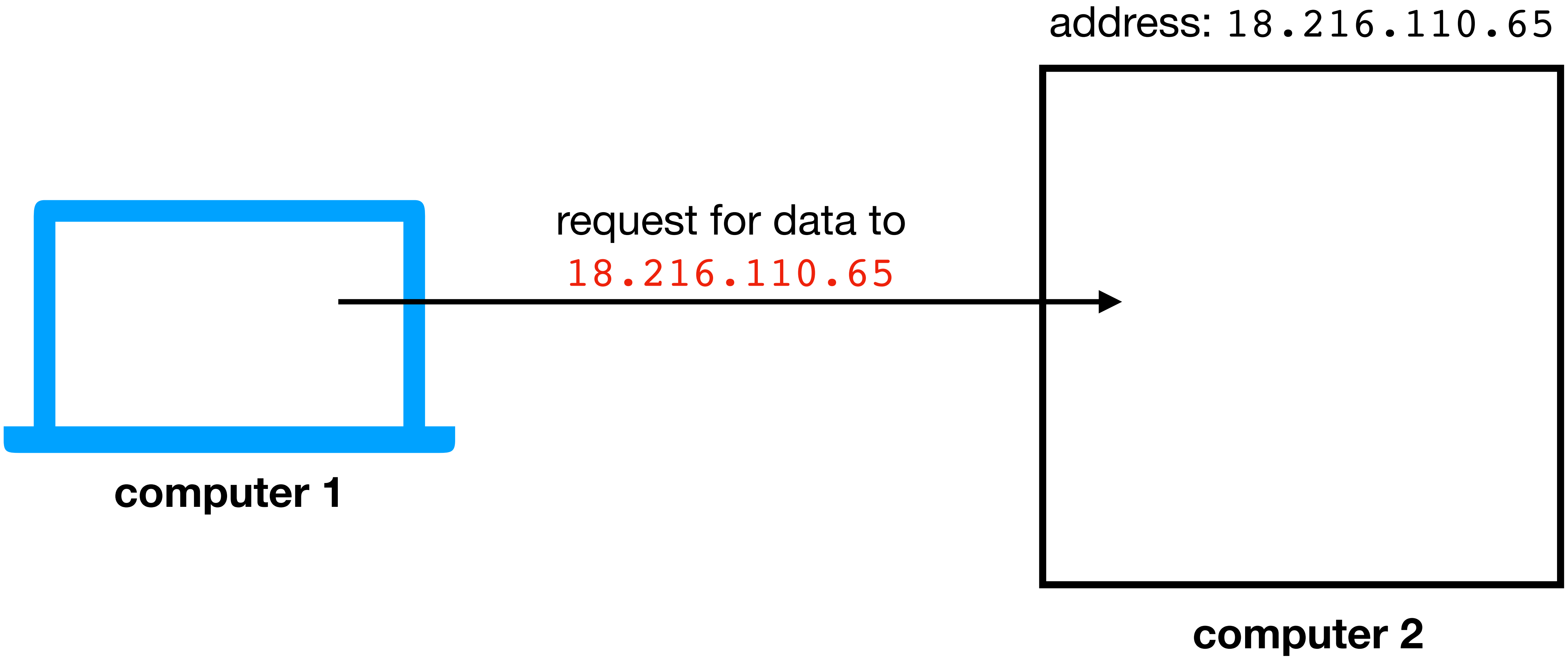
computer 2

Solution: every machine* has an IP address (Internet Protocol).

Requests are sent to a specific IP address.

*some machines have more multiple addresses

Internet Protocol

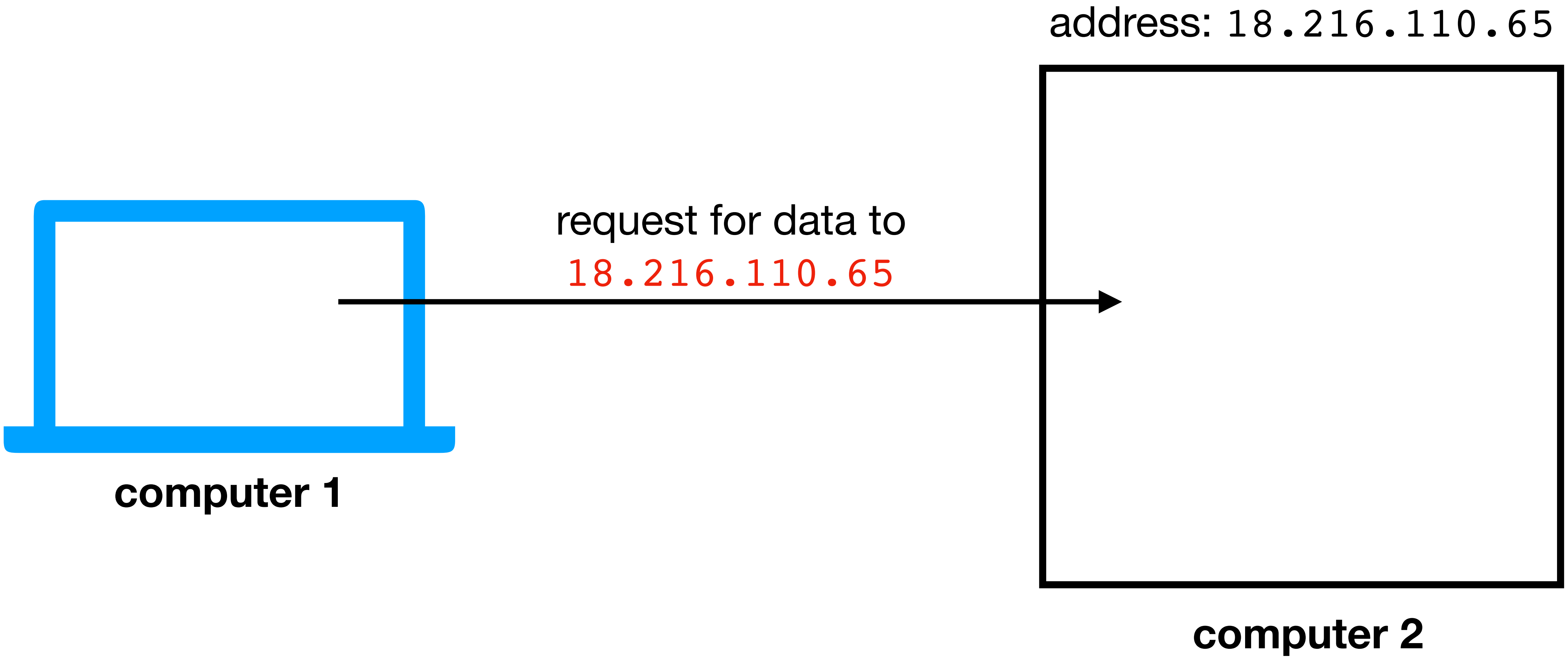


Solution: every machine* has an IP address (Internet Protocol).

Requests are sent to a specific IP address.

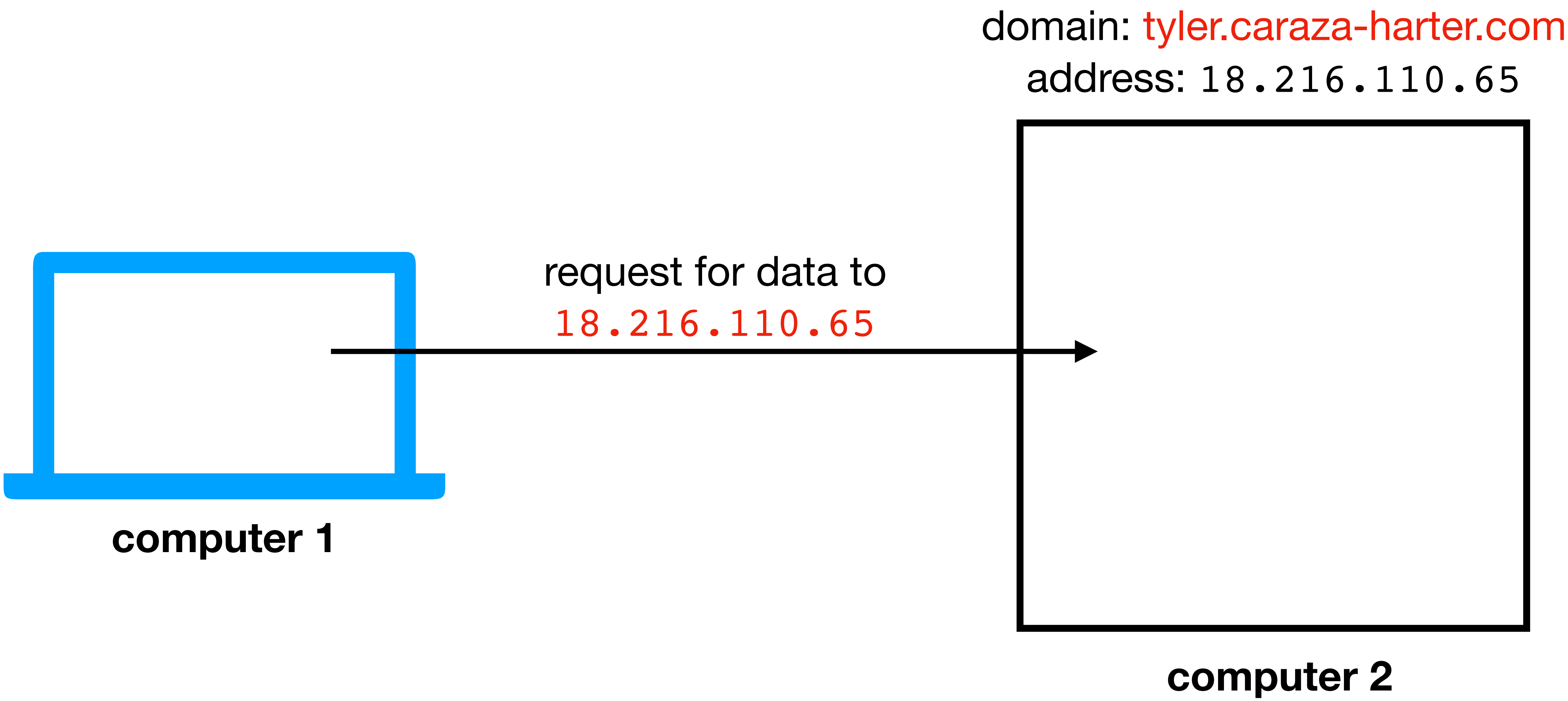
*some machines have more multiple addresses

Internet Protocol



Challenge: it's hard to remember IP addresses.
Imagine you had to type a number instead of www.google.com!

Domain Names



domain: tyler.caraza-harter.com

address: 18.216.110.65

request for data to

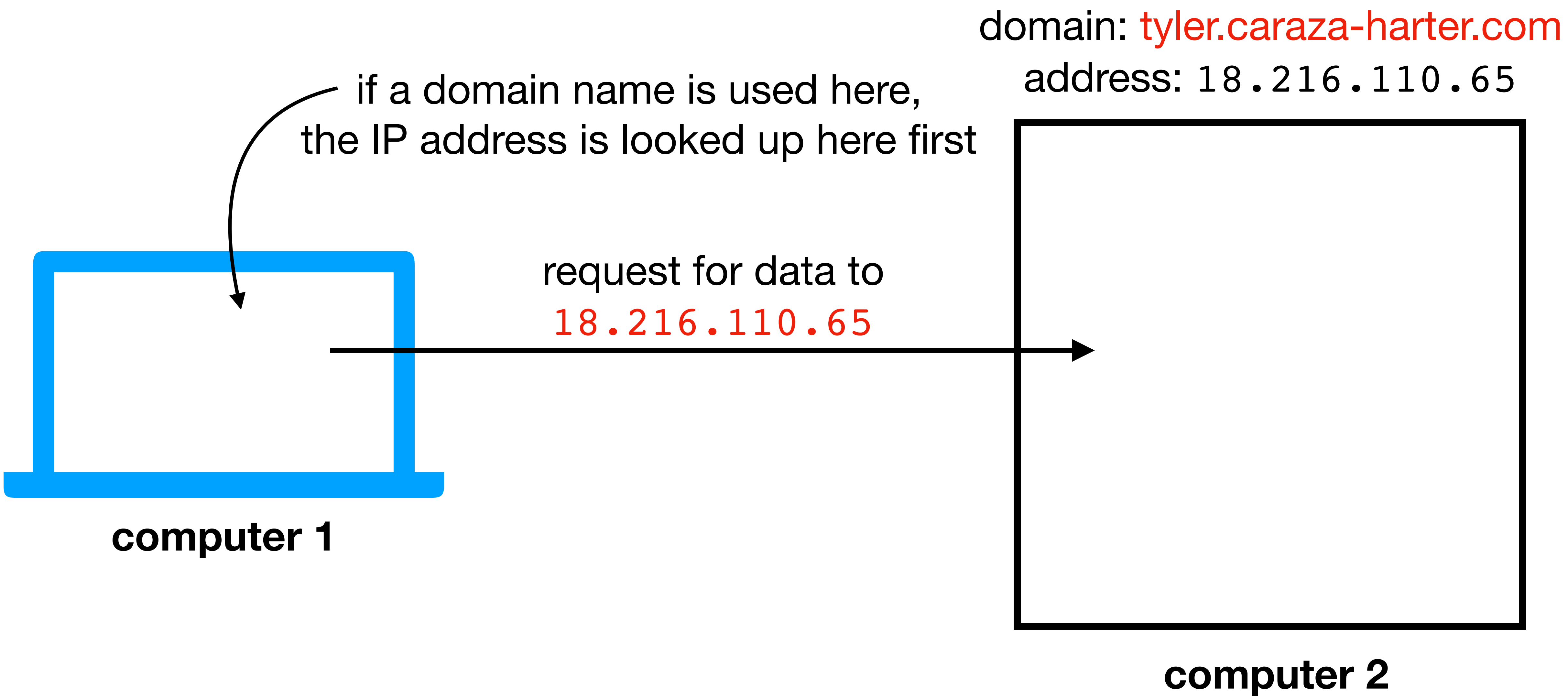
18.216.110.65

computer 1

computer 2

Solution: use "nicknames" (called domain names) for IP addresses of machines that serve data

Domain Names



domain: tyler.caraza-harter.com

address: 18.216.110.65

if a domain name is used here,
the IP address is looked up here first

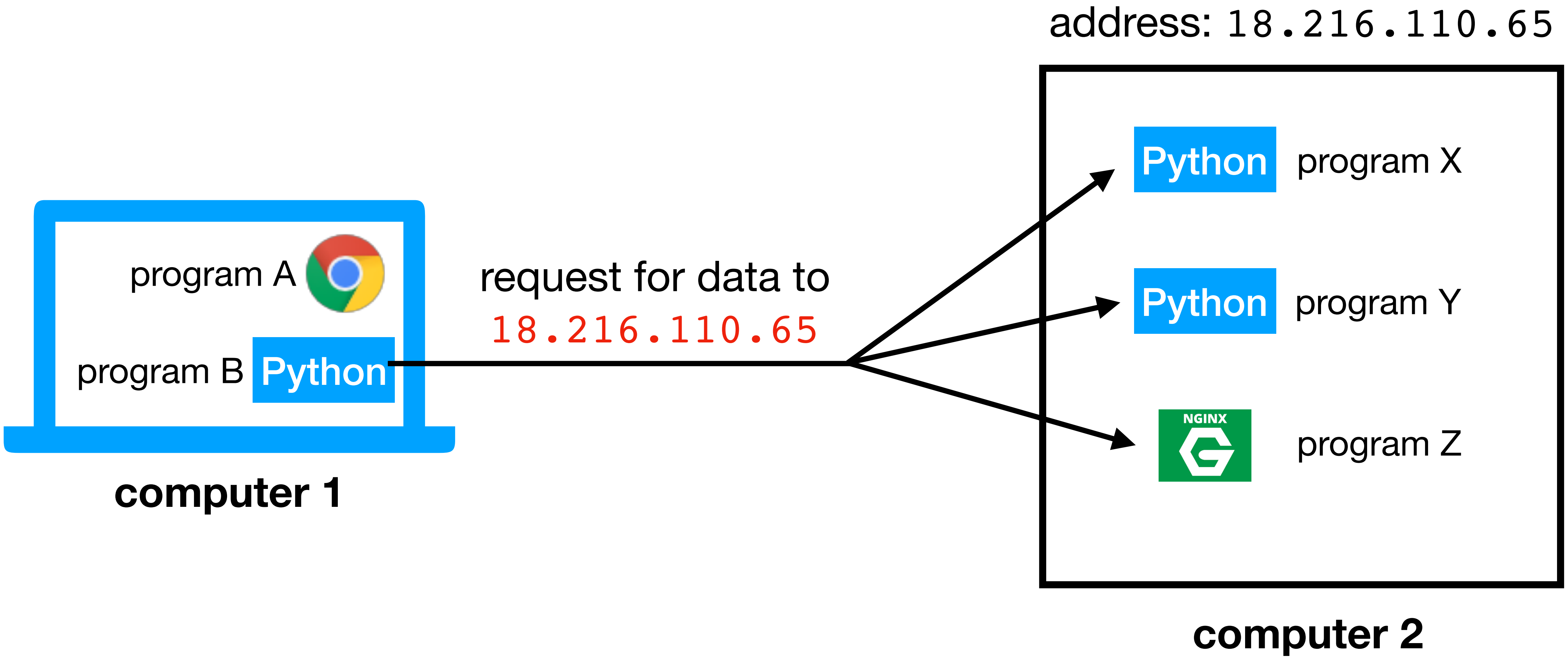
request for data to
18.216.110.65

computer 1

computer 2

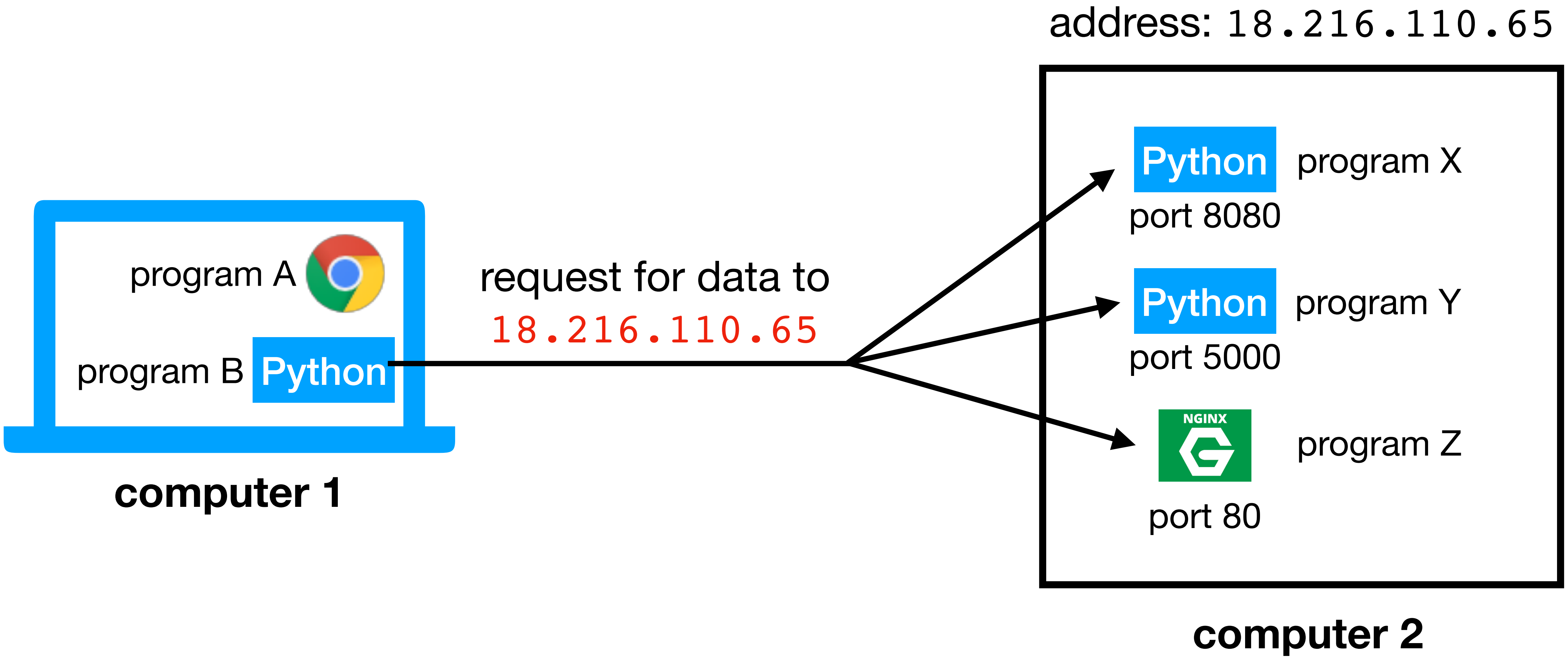
Solution: use "nicknames" (called domain names)
for IP addresses of machines that serve data

Port Numbers



Challenge: there may be multiple programs running on each computer.
How do we get the messages to the right program?

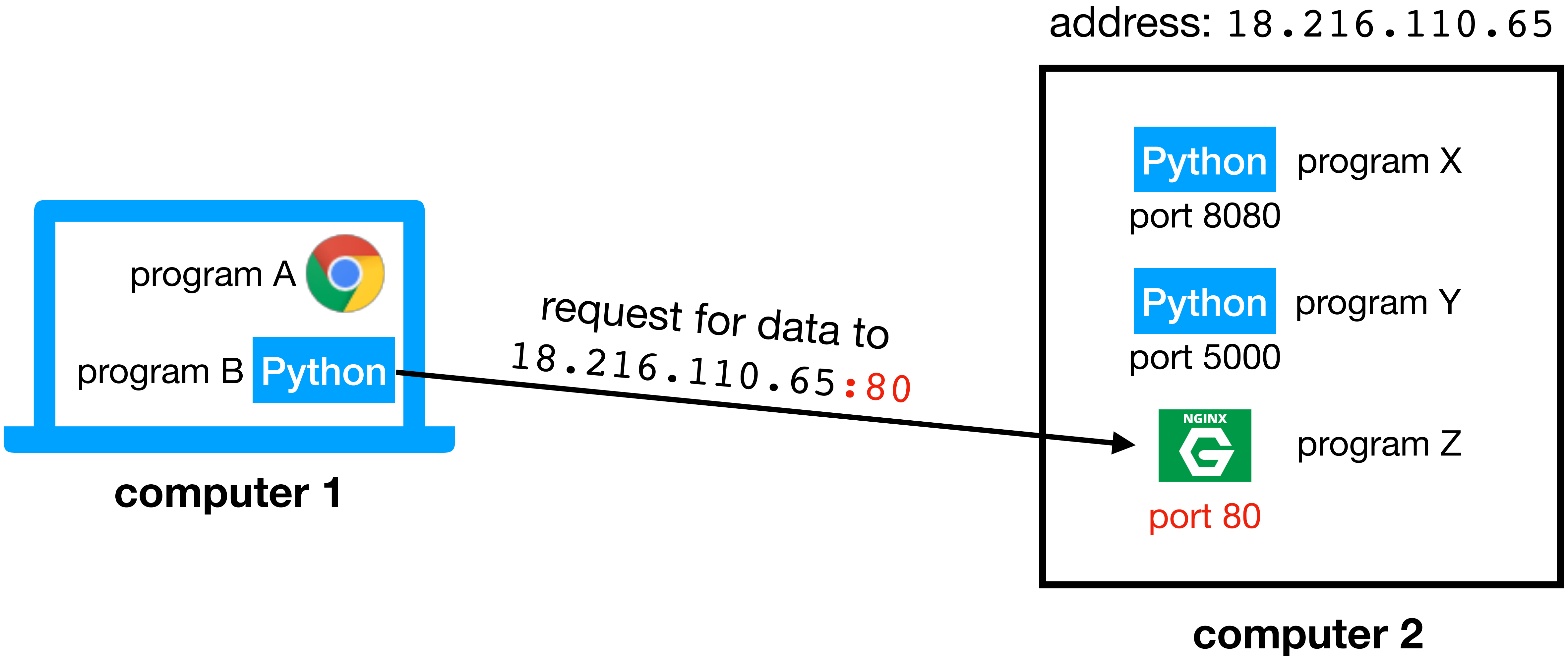
Port Numbers



Solution: give each program a unique ID (called a "port number")

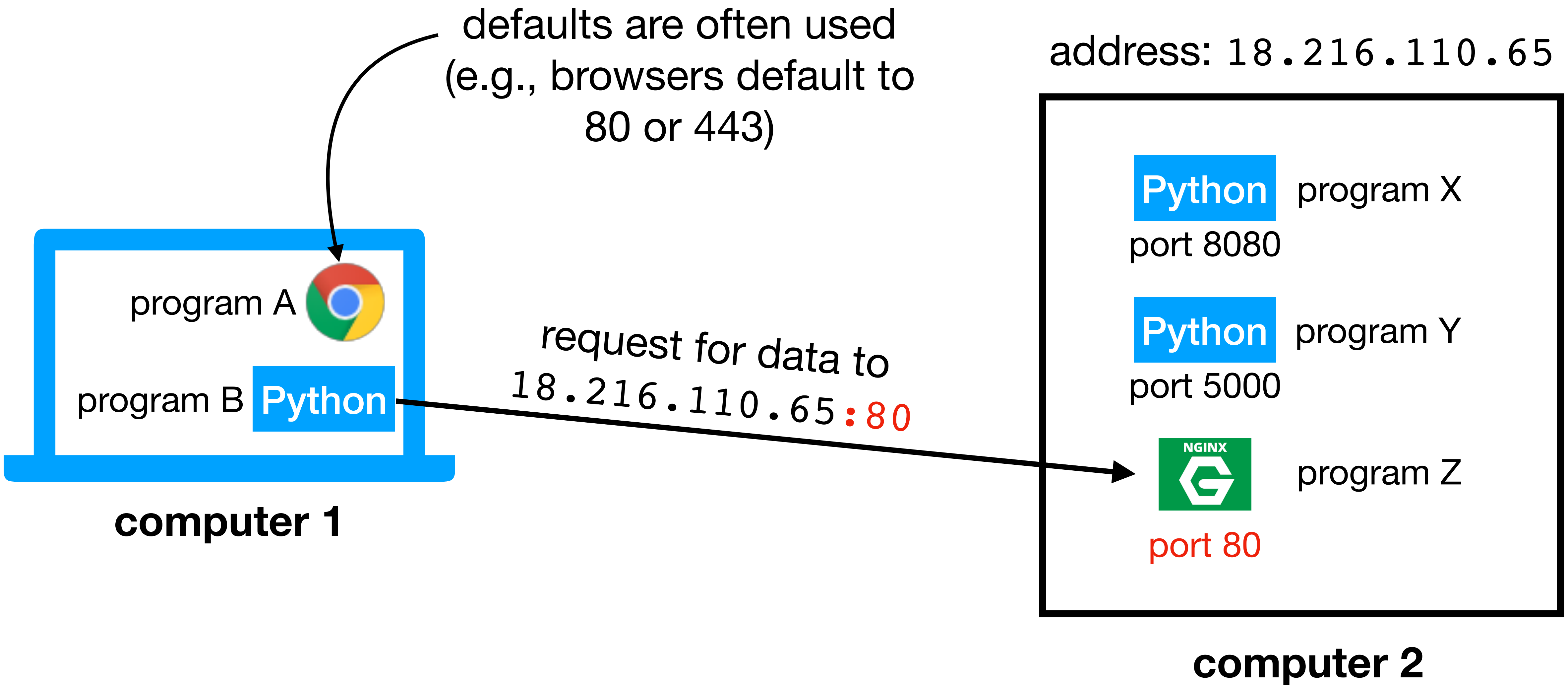
(like apartment numbers)

Port Numbers

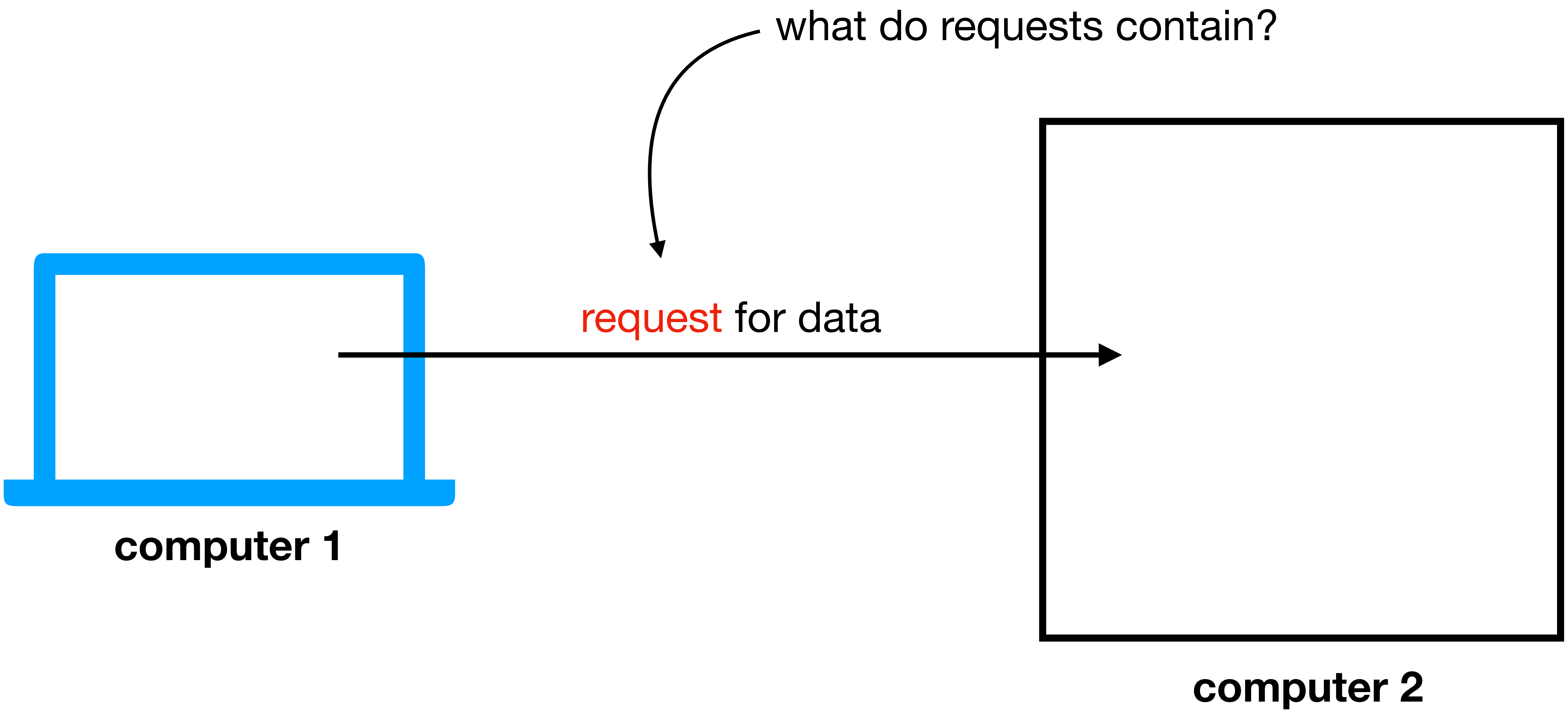


Solution: specify port number in request

Port Numbers



Solution: specify port number in request

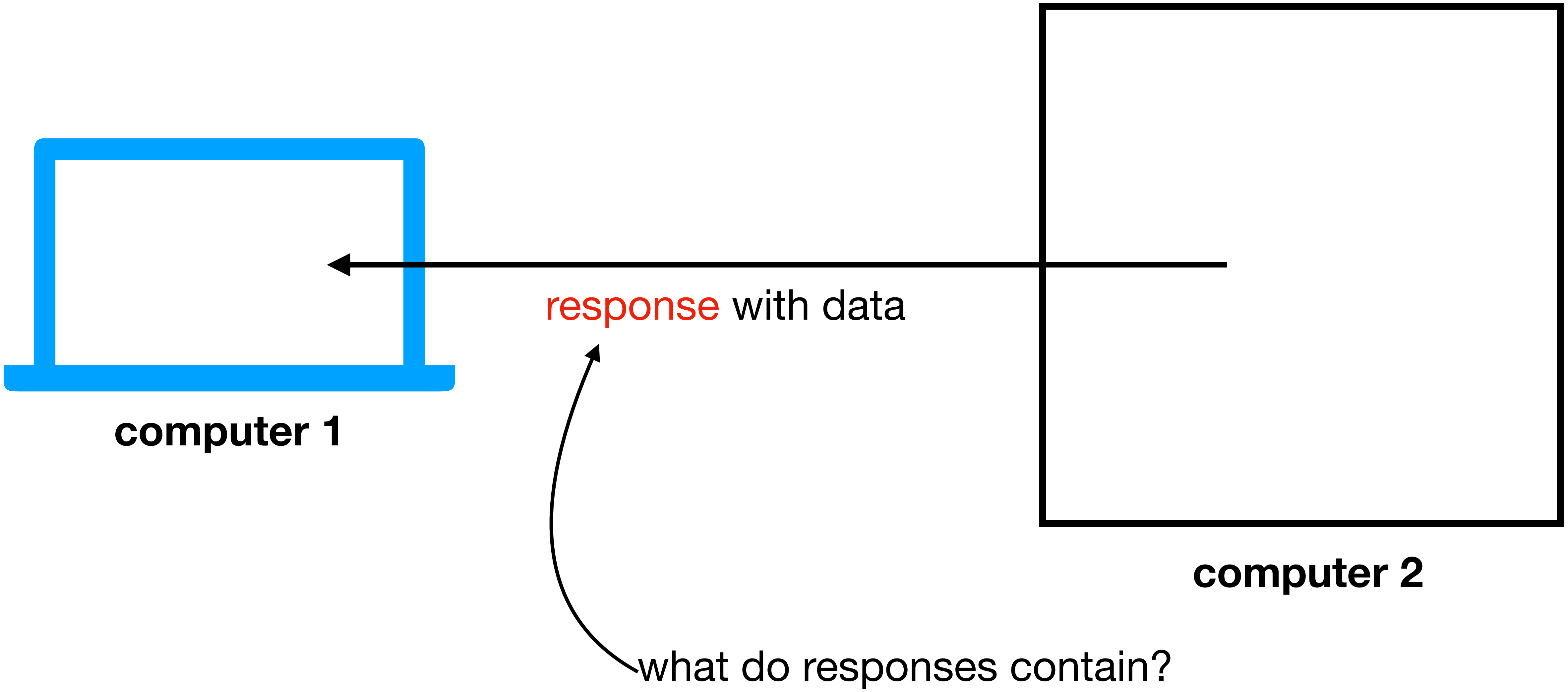


what do requests contain?

request for data

computer 1

computer 2



response with data

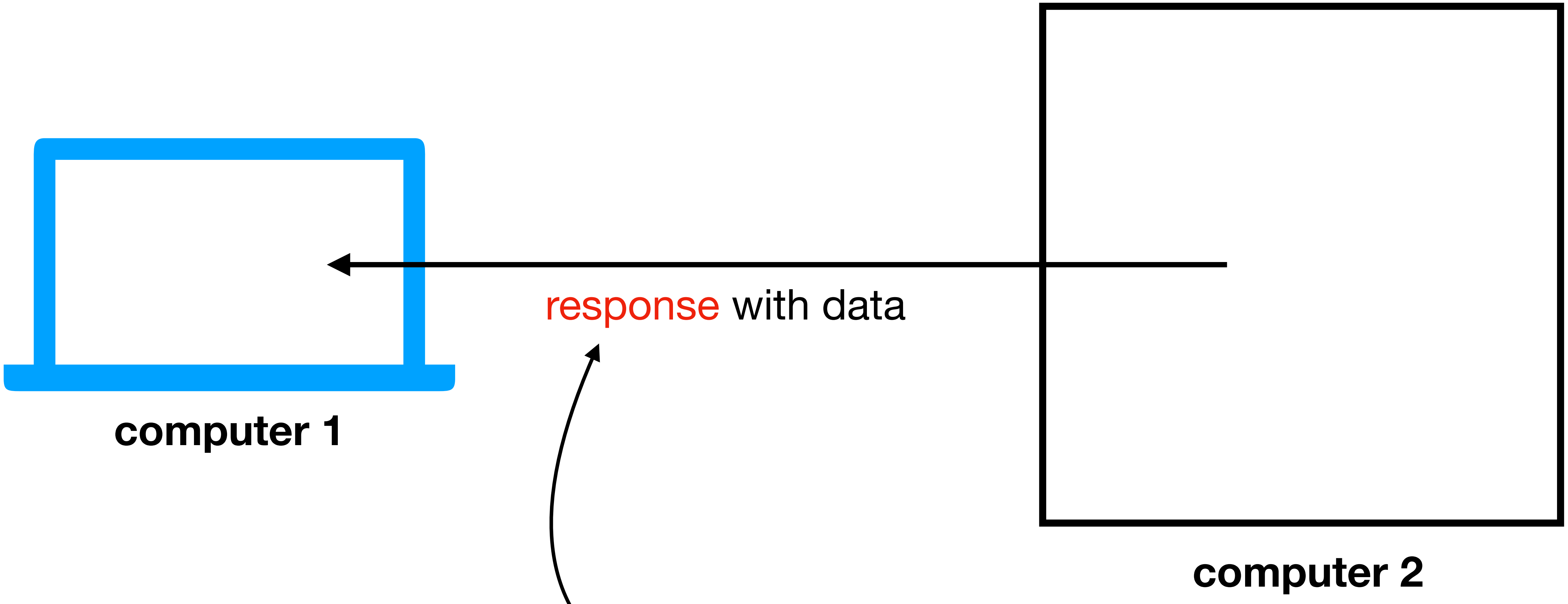
computer 1

computer 2

what do responses contain?

depends on application! (video chat, web browsing, etc)

we'll only consider web applications for this semester



what do responses contain?

Learning Objectives Today

Motivation

Networking Basics

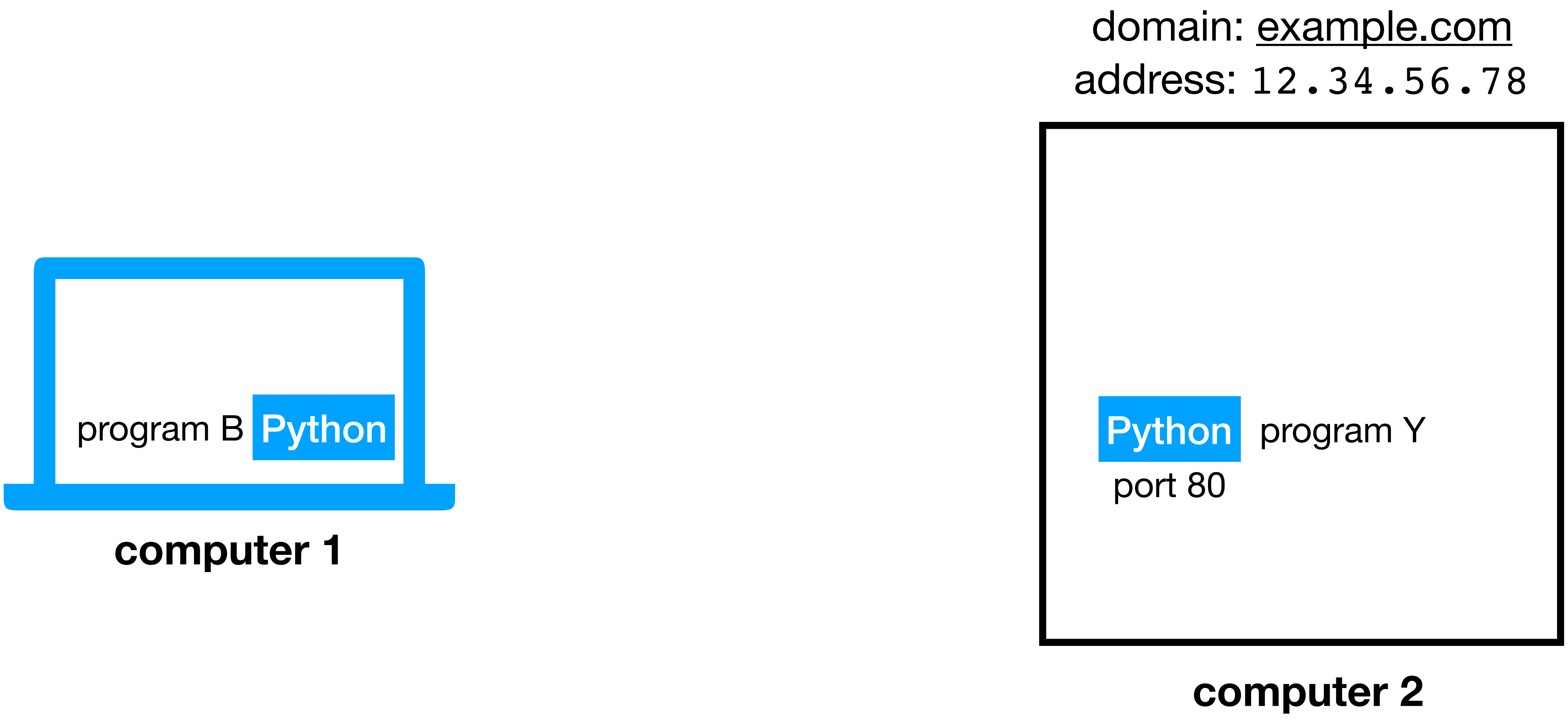
HTTP (Hypertext Transfer Protocol)

Requests Module

HTTP

Protocol for communicating web data

- downloading a specific webpage, image, etc



domain: example.com

address: `12.34.56.78`

program B Python

computer 1

Python program Y

port 80

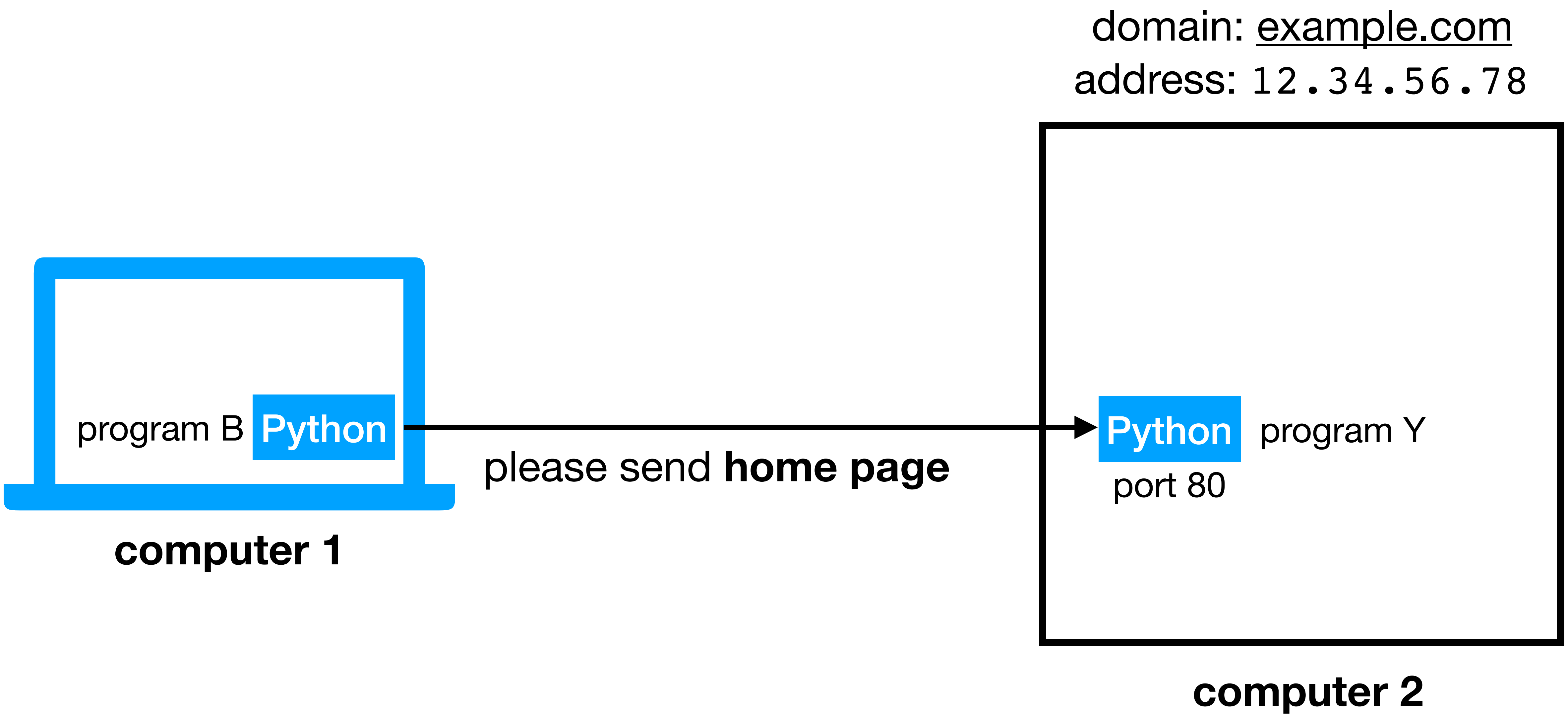
computer 2

Note: we won’t talk about HTTPS today, which is HTTP with encryption

HTTP

Protocol for communicating web data

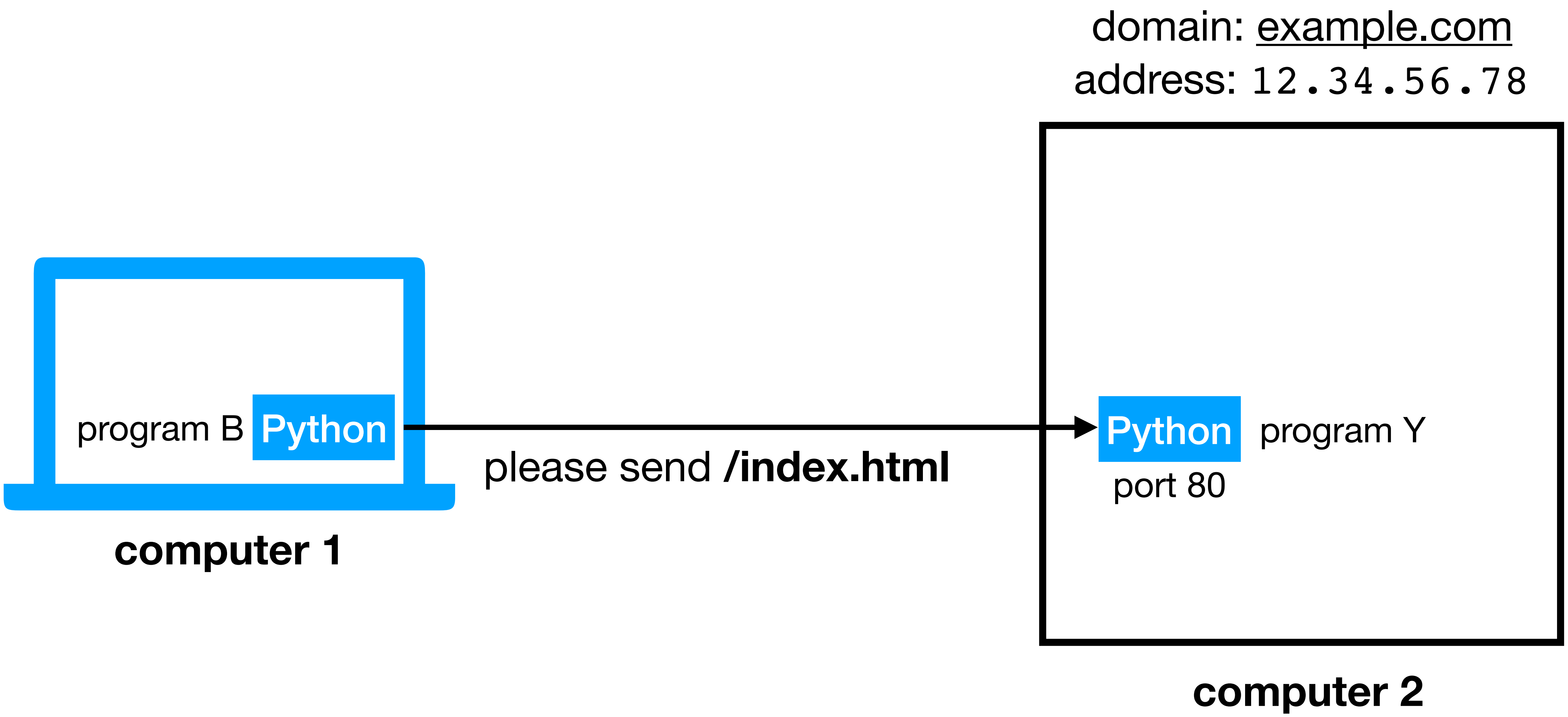
- downloading a specific webpage, image, etc



HTTP

Protocol for communicating web data

- downloading a specific webpage, image, etc



domain: example.com
address: `12.34.56.78`

program B Python

please send **/index.html**

Python program Y
port 80

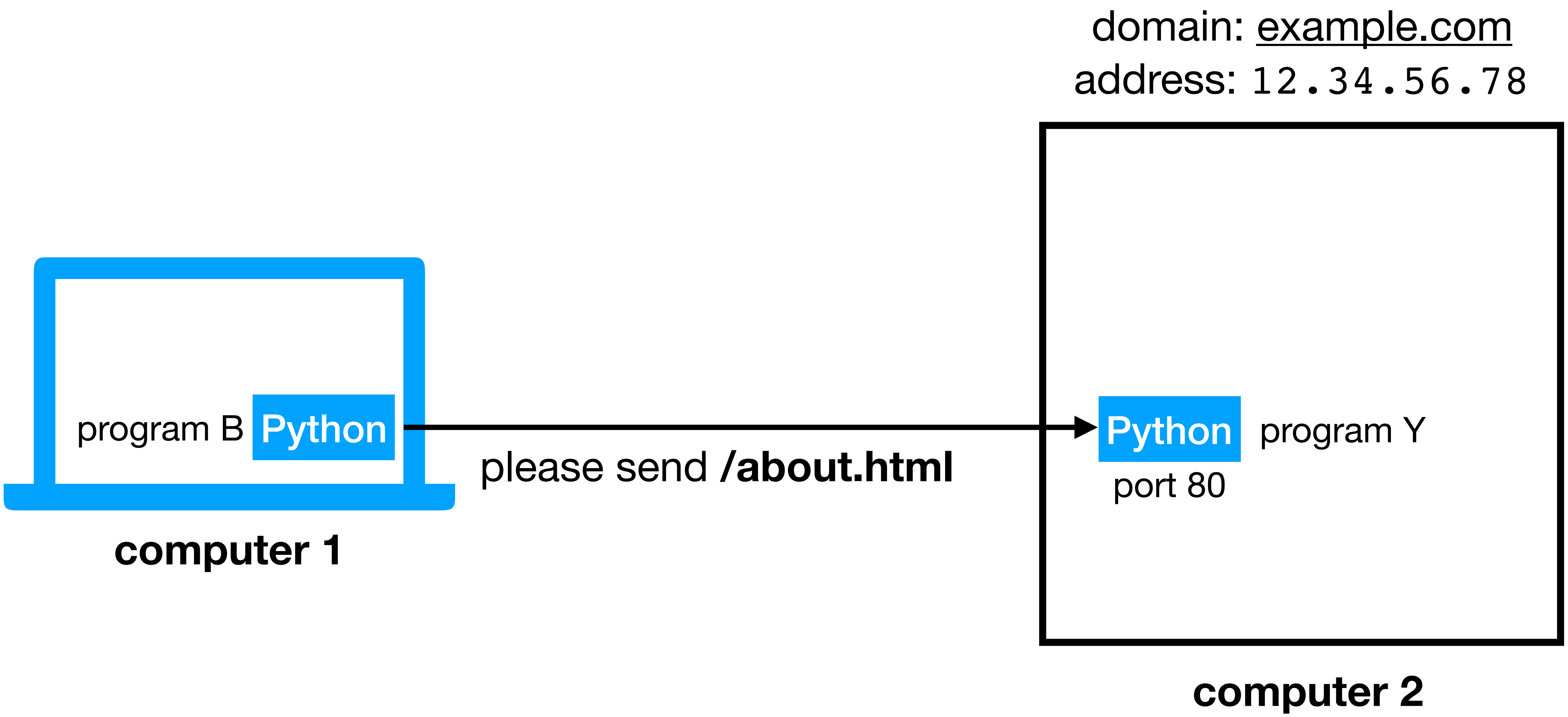
computer 1

computer 2

HTTP

Protocol for communicating web data

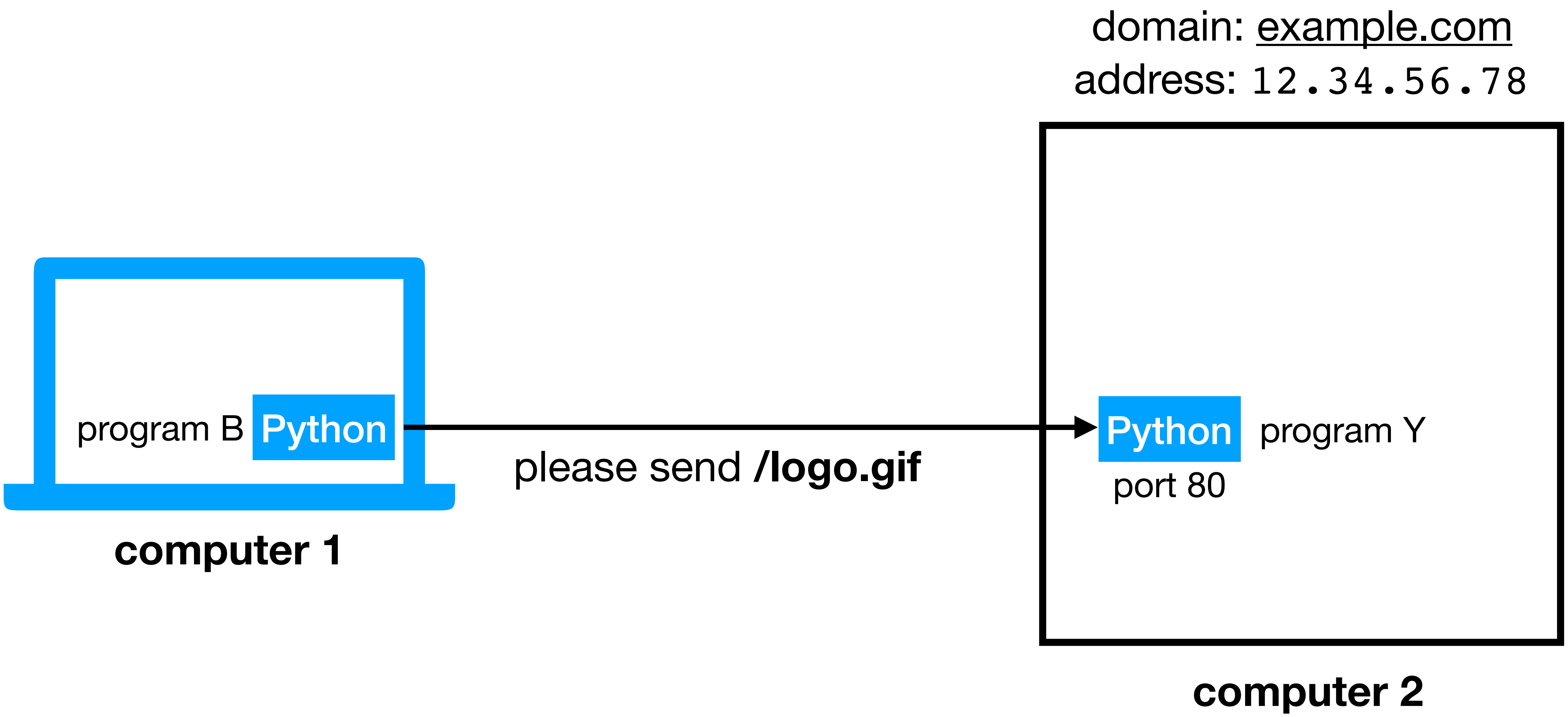
- downloading a specific webpage, image, etc



HTTP

Protocol for communicating web data

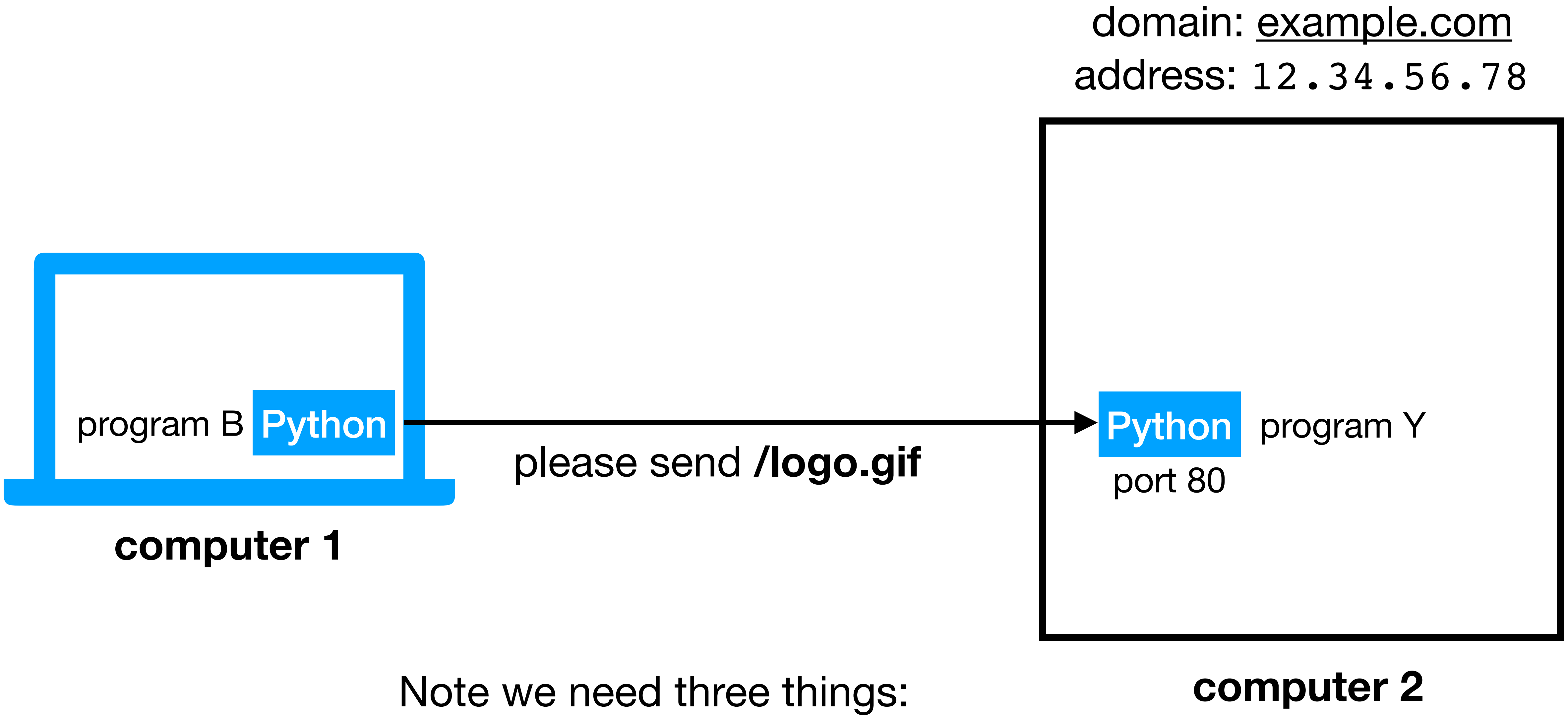
- downloading a specific webpage, image, etc



HTTP

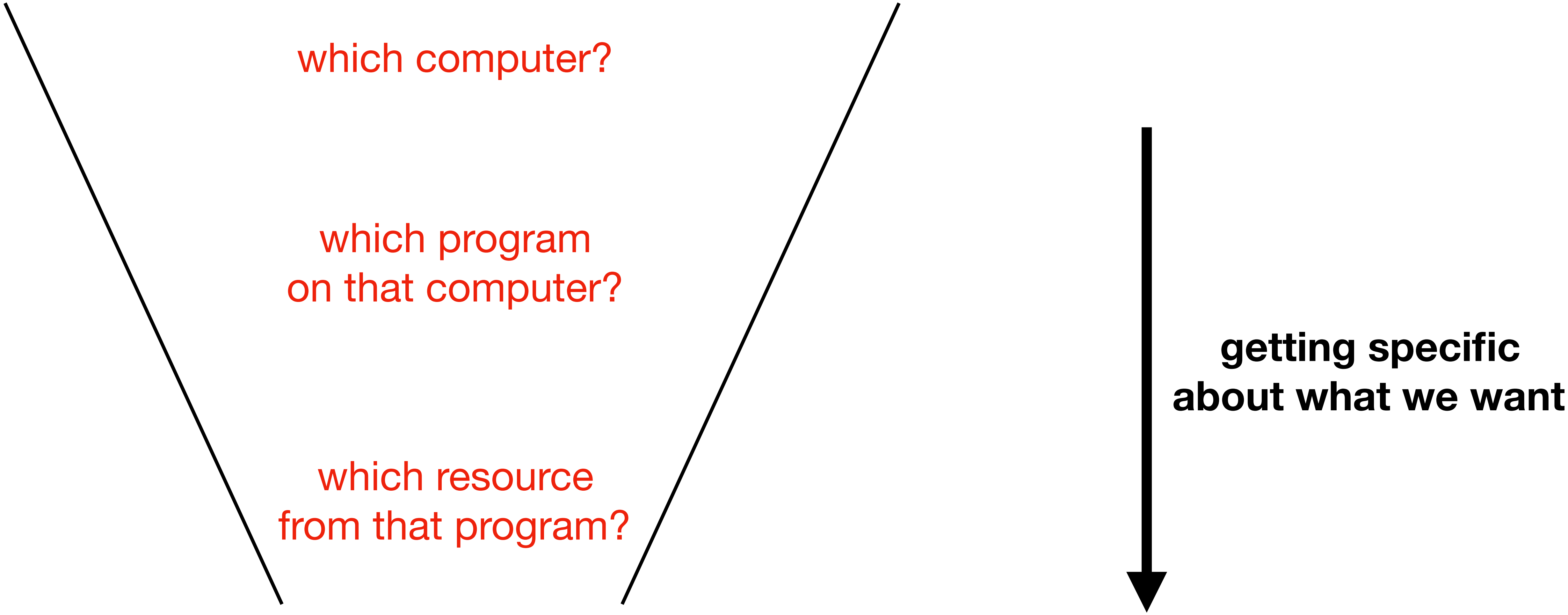
Protocol for communicating web data

- downloading a specific webpage, image, etc



Note we need three things:

1. domain name
2. port number
3. resource (file name)



which computer?

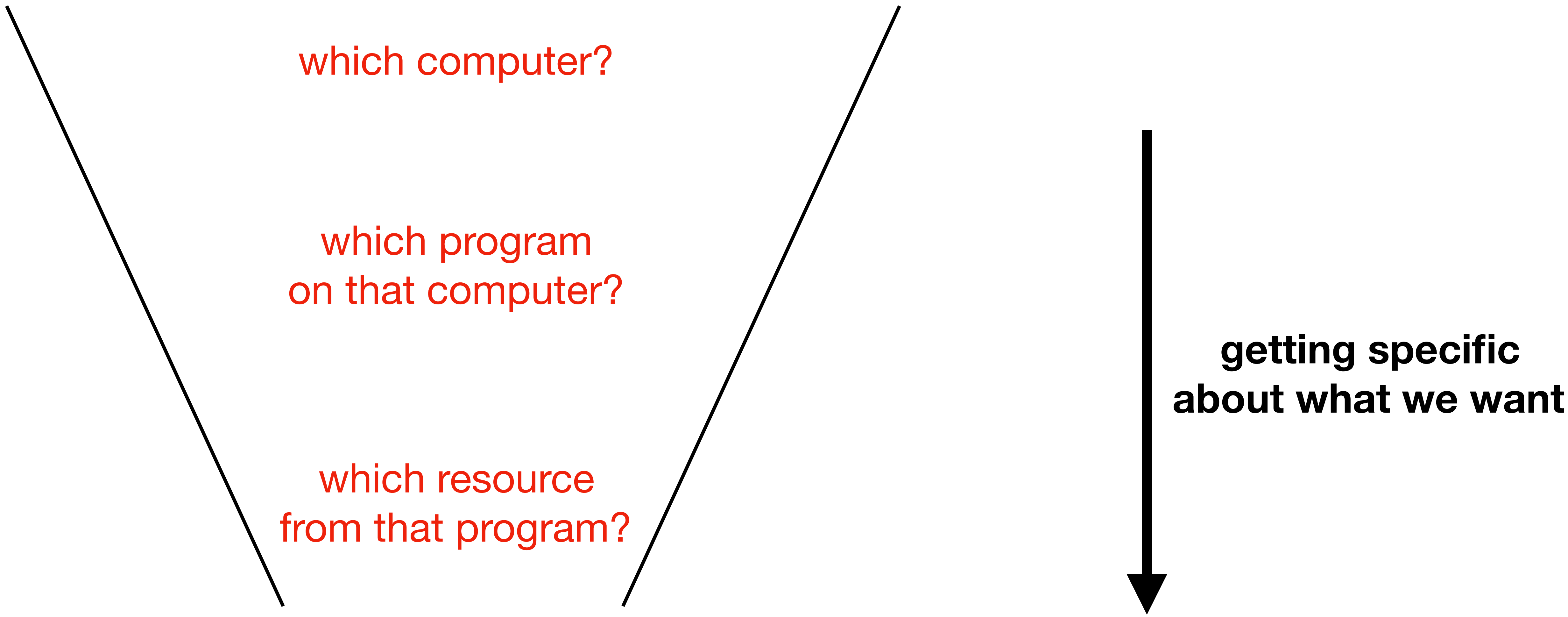
which program
on that computer?

which resource
from that program?

**getting specific
about what we want**

Note we need three things:

1. domain name
2. port number
3. resource (file name)



which computer?

which program on that computer?

which resource from that program?

getting specific about what we want

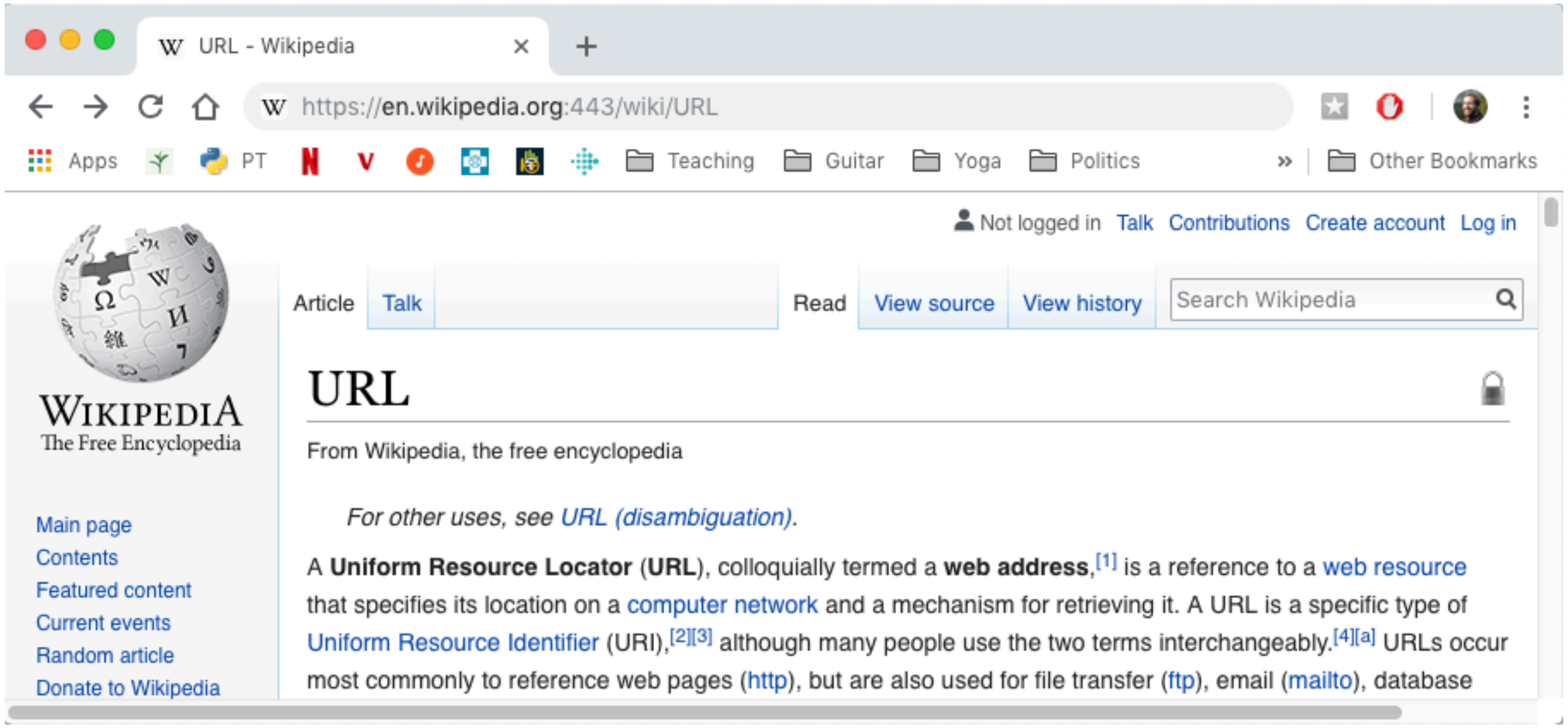
URL

Note we need three things:

1. domain name
2. port number
3. resource (file name)

URLs

`https://en.wikipedia.org:443/wiki/URL`



URL

From Wikipedia, the free encyclopedia

For other uses, see URL (disambiguation).

A **Uniform Resource Locator** (**URL**), colloquially termed a **web address**,[1] is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI),[2][3] although many people use the two terms interchangeably.[4][a] URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database

URL

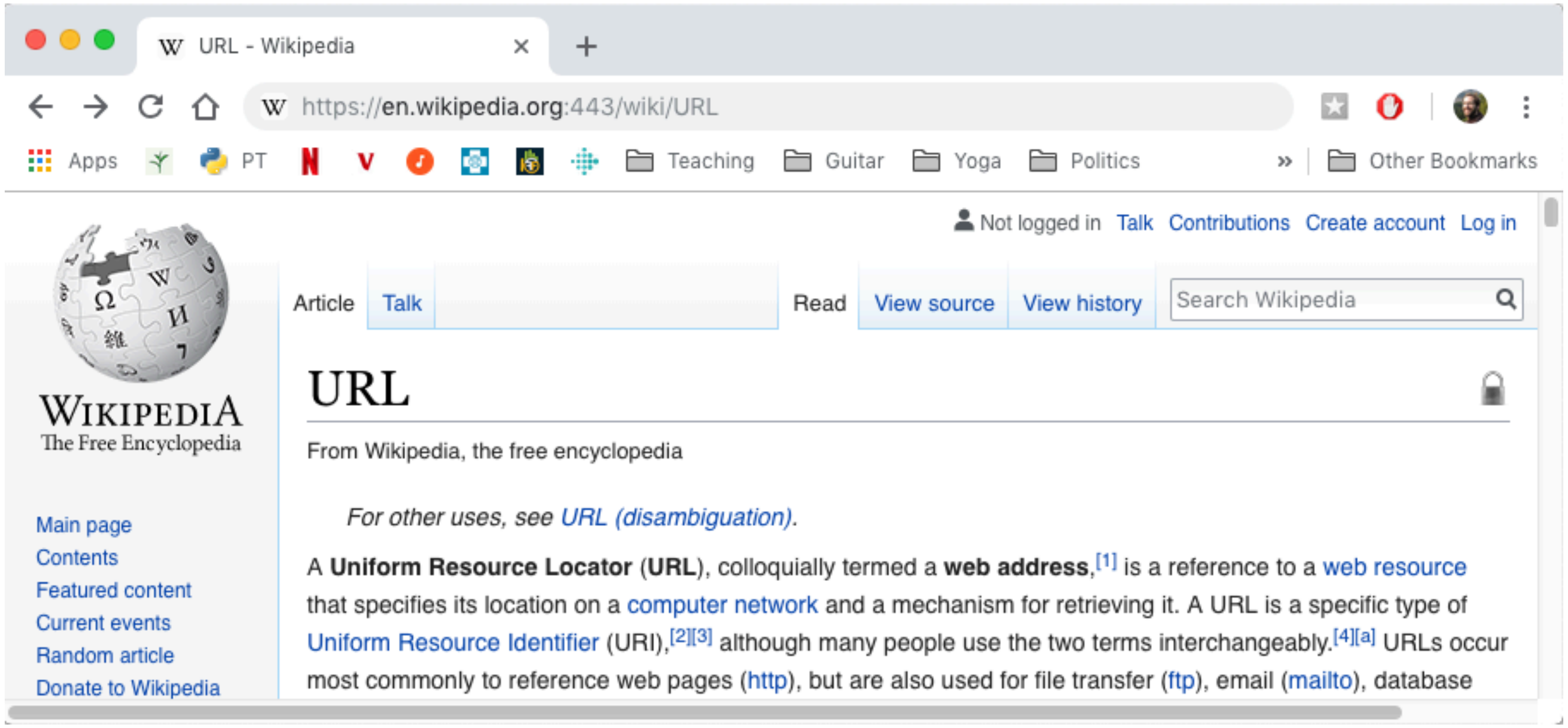
Note we need three things:

1. domain name
2. port number
3. resource (file name)

URLs

domain name

`https://en.wikipedia.org:443/wiki/URL`



URL

From Wikipedia, the free encyclopedia

For other uses, see URL (disambiguation).

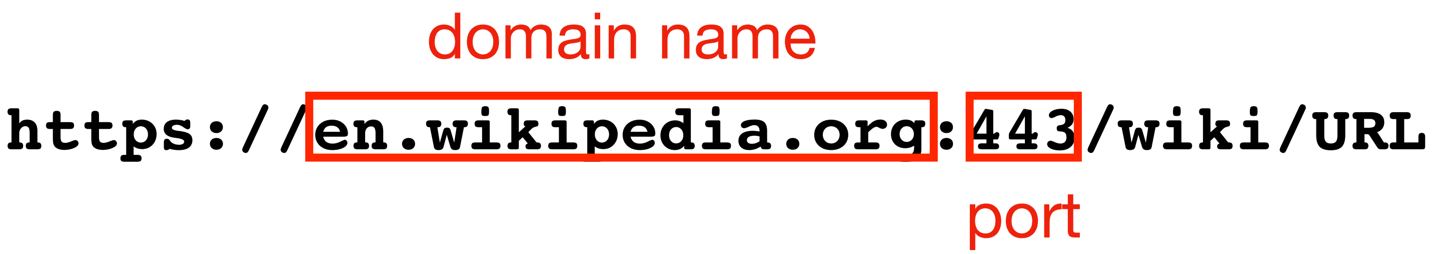
A **Uniform Resource Locator** (**URL**), colloquially termed a **web address**,[1] is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI),[2][3] although many people use the two terms interchangeably.[4][a] URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database

URL

Note we need three things:

1. domain name
2. port number
3. resource (file name)

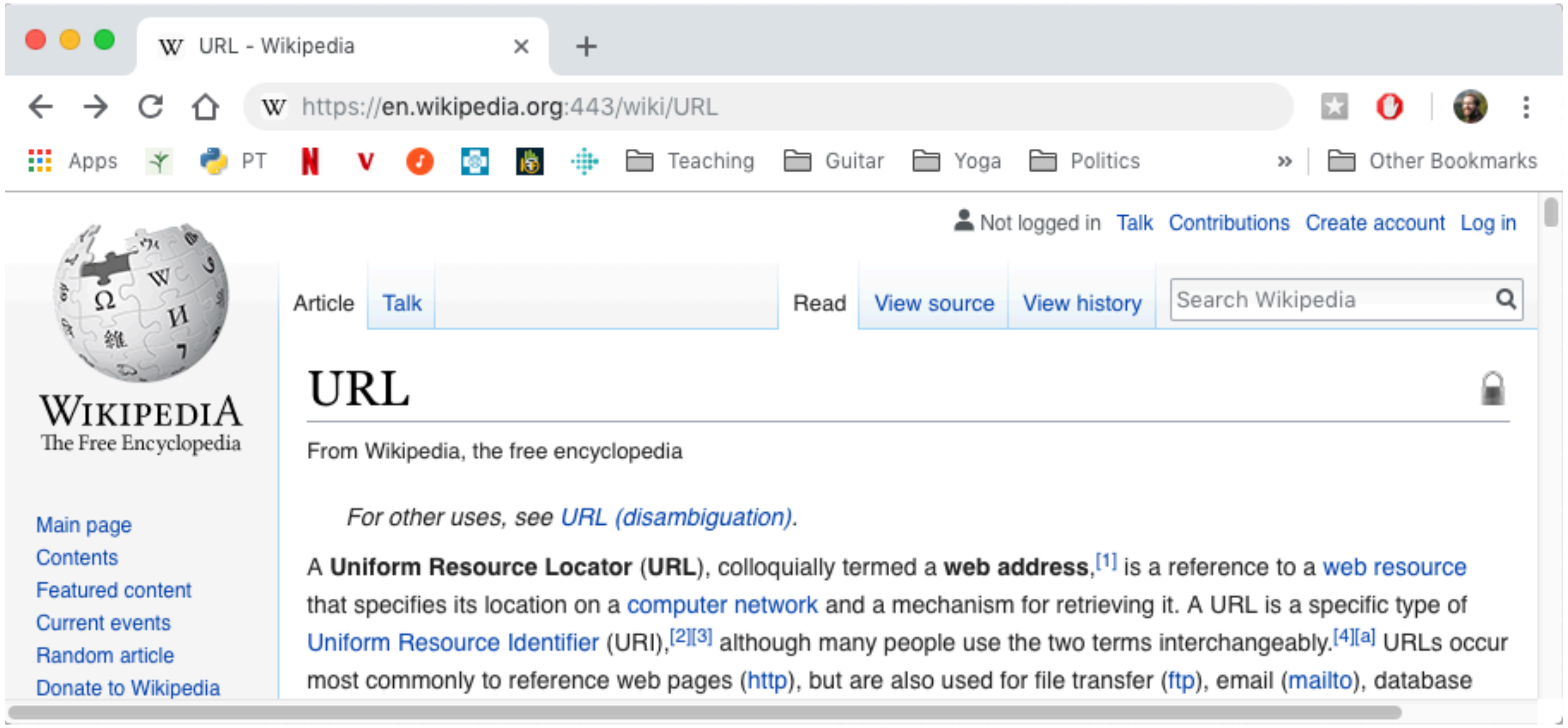
URLs



domain name

`https://en.wikipedia.org:443/wiki/URL`

port



URL

From Wikipedia, the free encyclopedia

For other uses, see URL (disambiguation).

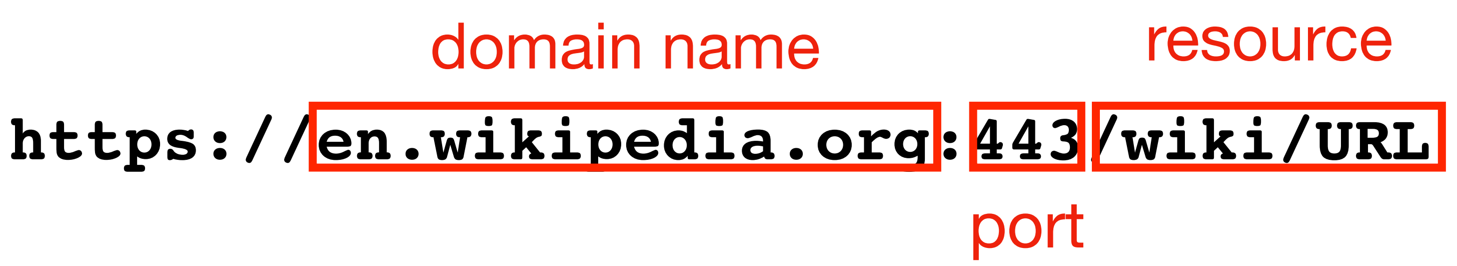
A **Uniform Resource Locator** (**URL**), colloquially termed a **web address**,[1] is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI),[2][3] although many people use the two terms interchangeably.[4][a] URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database

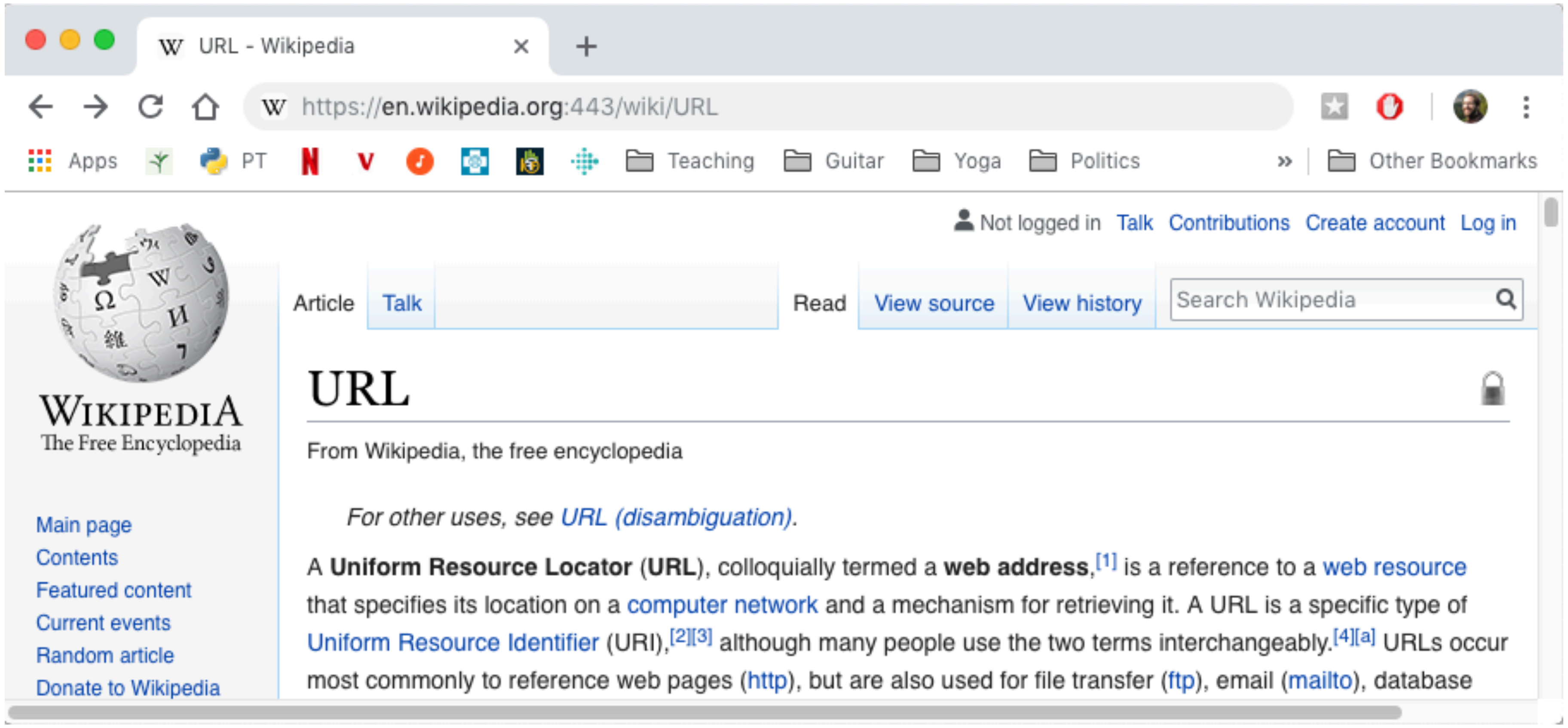
URL

Note we need three things:

1. domain name
2. port number
3. resource (file name)

URLs





URL

From Wikipedia, the free encyclopedia

For other uses, see URL (disambiguation).

A **Uniform Resource Locator** (**URL**), colloquially termed a **web address**,[1] is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI),[2][3] although many people use the two terms interchangeably.[4][a] URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database

URL

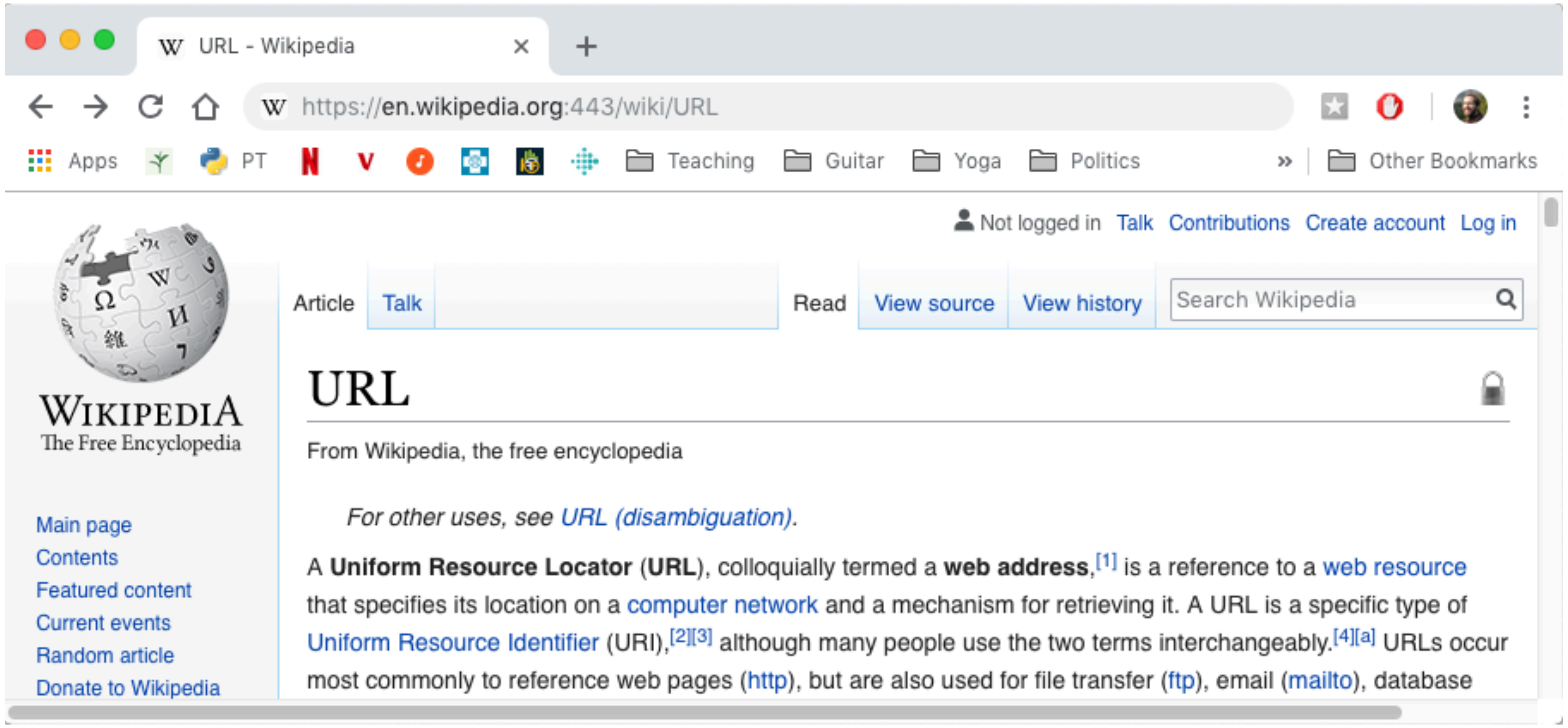
Note we need three things:
1. domain name
2. port number
3. resource (file name)

URLs

domain name resource

`https://en.wikipedia.org/wiki/URL`

port would have defaulted to 443 if not specified



URL

From Wikipedia, the free encyclopedia

For other uses, see URL (disambiguation).

A **Uniform Resource Locator** (**URL**), colloquially termed a **web address**,[1] is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI),[2][3] although many people use the two terms interchangeably.[4][a] URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database

URL

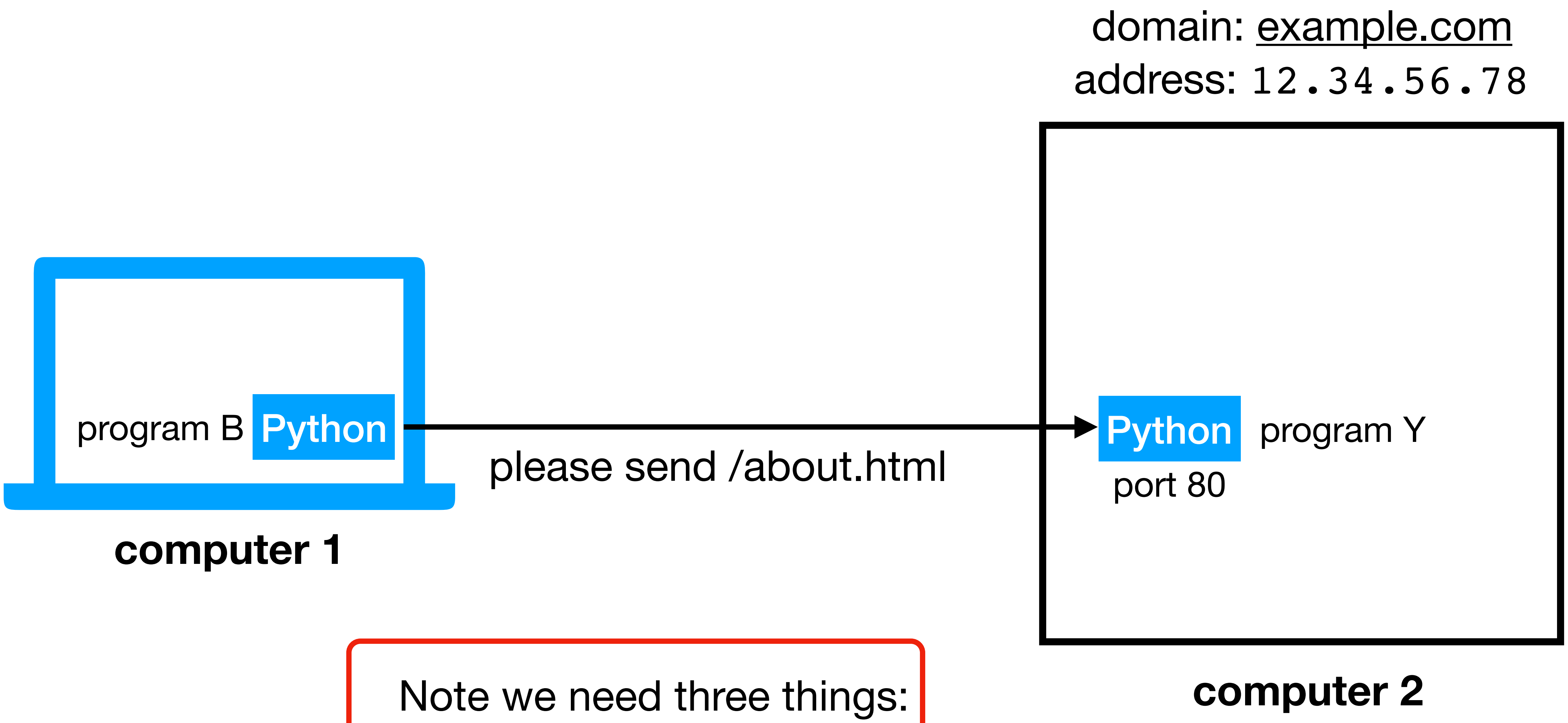
Note we need three things:

1. domain name
2. port number
3. resource (file name)

HTTP

Protocol for communicating web data

- downloading a specific webpage, image, etc



URL

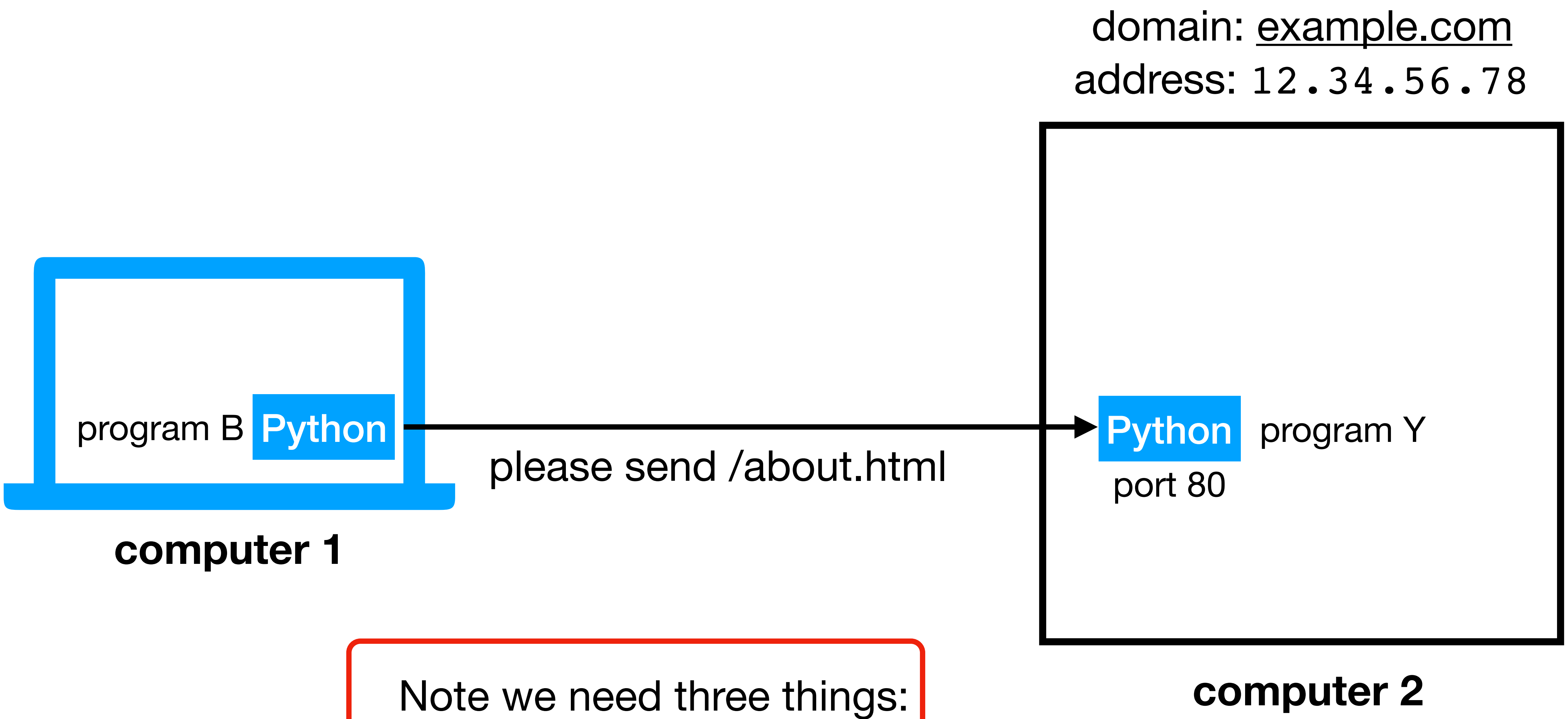
Note we need three things:

1. domain name
2. port number
3. resource (file name)

HTTP

Protocol for communicating web data

- downloading a specific webpage, image, etc



domain: example.com
address: `12.34.56.78`

program B Python

computer 1

please send /about.html

Python program Y
port 80

computer 2

URL

Note we need three things:

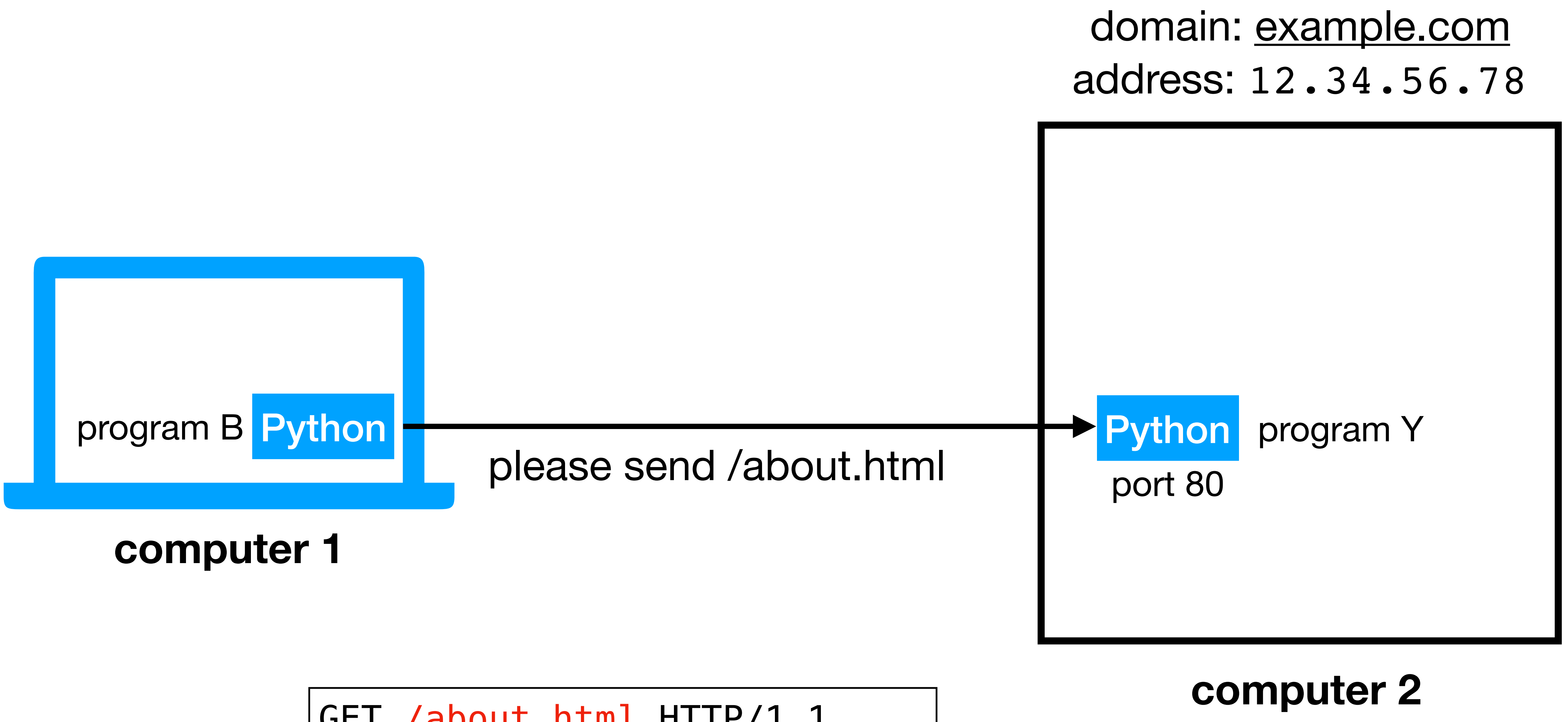
1. domain name
2. port number
3. resource (file name)

how do we specify this?

HTTP

Protocol for communicating web data

- downloading a specific webpage, image, etc



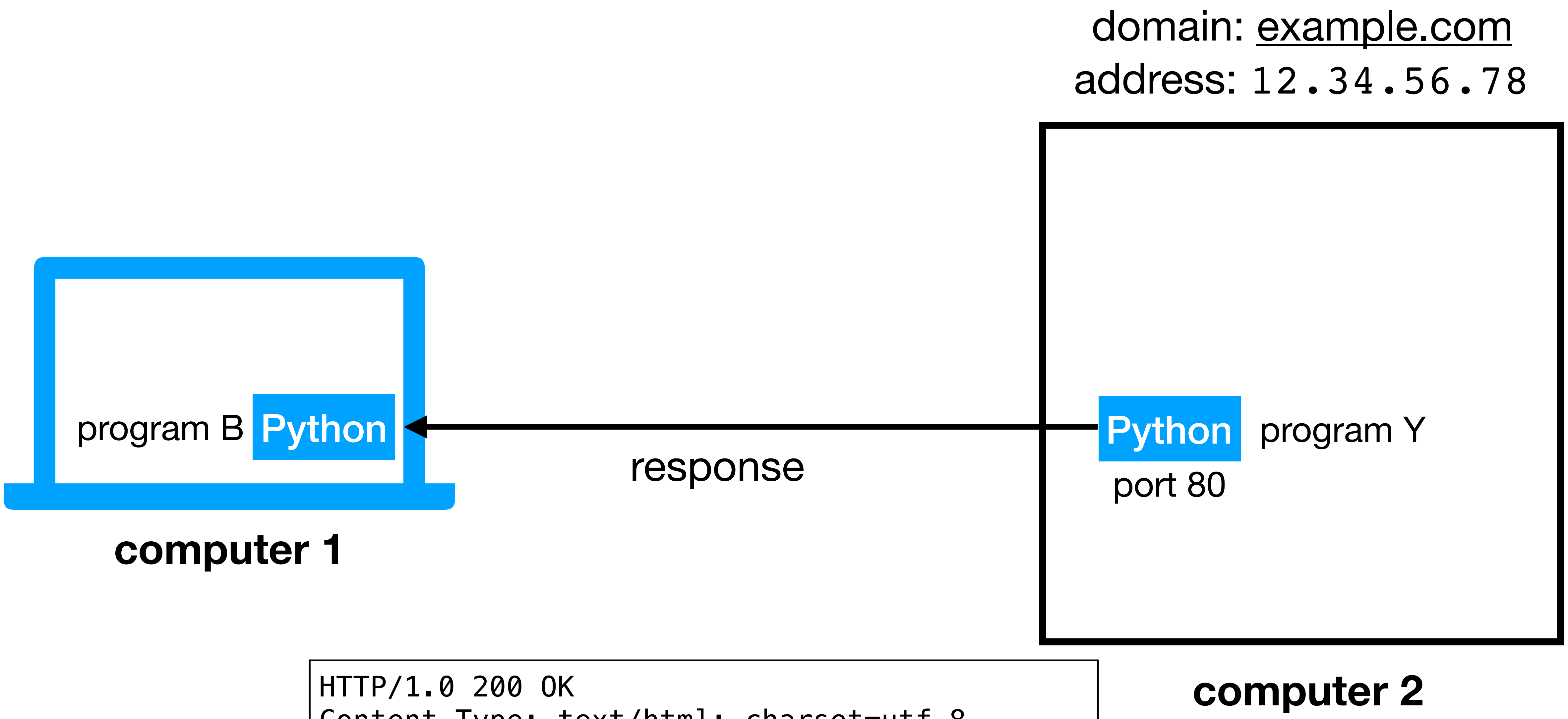
HTTP Request:

```
GET /about.html HTTP/1.1
Host: example.com
User-Agent: ...
Accept: */*
```

HTTP

Protocol for communicating web data

- downloading a specific webpage, image, etc



HTTP Response:

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
Server: Werkzeug/0.14.1 Python/3.6.6
Date: Sun, 11 Nov 2018 17:00:29 GMT

all the contents
```

Request and Response Headers

HTTP Request:

```
GET /about.html HTTP/1.1
Host: example.com
User-Agent: ...
Accept: */*
```

HTTP Response:

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
Server: Werkzeug/0.14.1 Python/3.6.6
Date: Sun, 11 Nov 2018 17:00:29 GMT

all the contents
```

There are **LOTS** of details here we don’t care about right now

Request and Response Headers

we want the about.html page

HTTP Request:

```
GET /about.html HTTP/1.1
Host: example.com
User-Agent: ...
Accept: */*
```

HTTP Response:

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
Server: Werkzeug/0.14.1 Python/3.6.6
Date: Sun, 11 Nov 2018 17:00:29 GMT

all the contents
```

data in about.html

There are **LOTS** of details here we don’t care about right now

Request and Response Headers

we want the about.html page

HTTP Request:

```
GET /about.html HTTP/1.1
Host: example.com
User-Agent: ...
Accept: */*
```

status code. 200 is good. 404, 500, others are various errors or other more complicated states

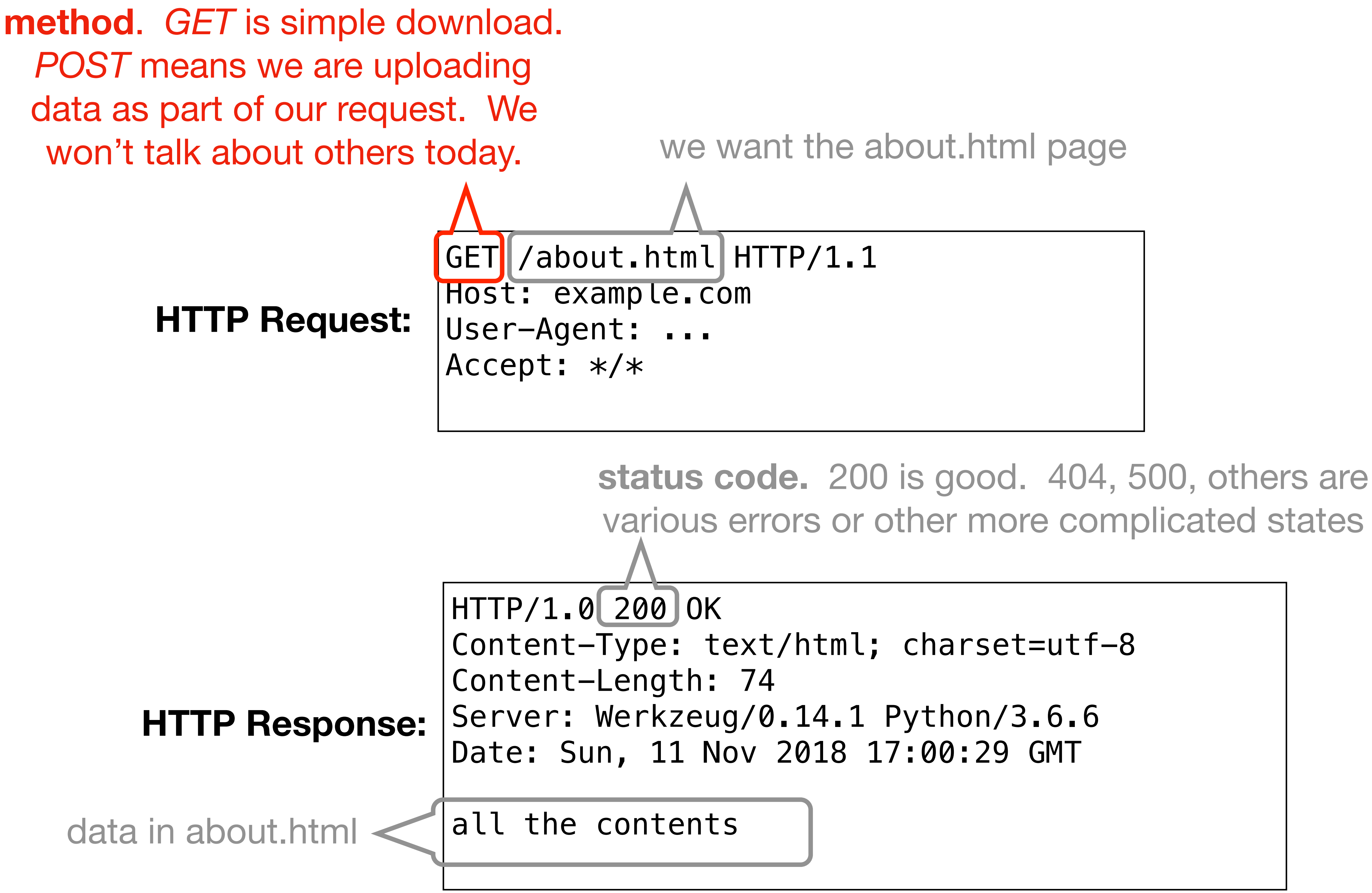
HTTP Response:

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
Server: Werkzeug/0.14.1 Python/3.6.6
Date: Sun, 11 Nov 2018 17:00:29 GMT

all the contents
```

data in about.html

There are **LOTS** of details here we don't care about right now



method. *GET* is simple download. *POST* means we are uploading data as part of our request. We won't talk about others today.

we want the about.html page

HTTP Request:

```
GET /about.html HTTP/1.1
Host: example.com
User-Agent: ...
Accept: */*
```

status code. 200 is good. 404, 500, others are various errors or other more complicated states

HTTP Response:

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 74
Server: Werkzeug/0.14.1 Python/3.6.6
Date: Sun, 11 Nov 2018 17:00:29 GMT

all the contents
```

data in about.html

There are **LOTS** of details here we don't care about right now

Learning Objectives Today

Motivation

Networking Basics

HTTP (Hypertext Transfer Protocol)

Requests Module

Requests module

Purpose

- easily send requests to a server and parse the response
- *"HTTP for Humans™"*

Installation

- install:

  ```
  pip install requests
  ```

Using it

- just import:

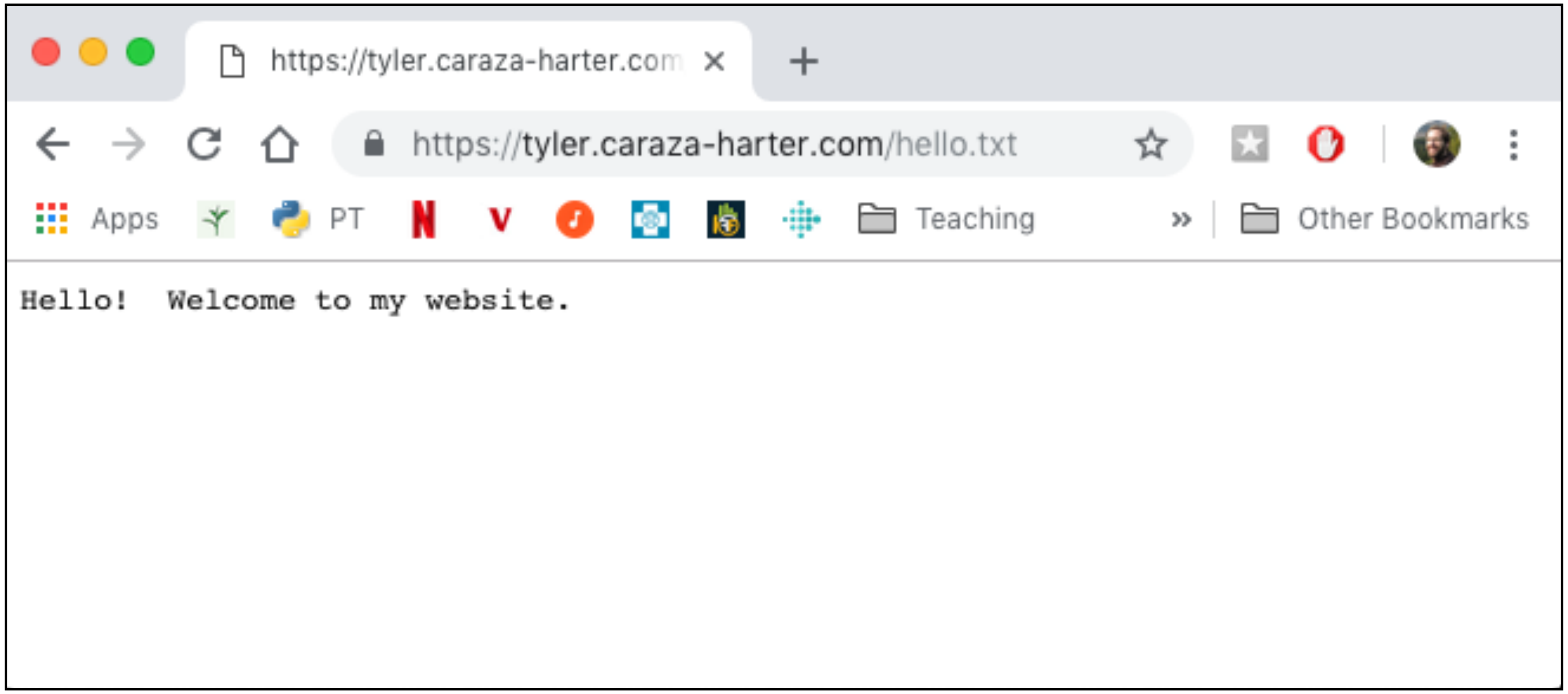
  ```
  import requests
  ```

GET Request

```
import requests

url = "https://tyler.caraza-harter.com/hello.txt"

requests.get(url)
```



https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/hello.txt
Apps
PT
Teaching
Other Bookmarks

```
Hello!  Welcome to my website.
```

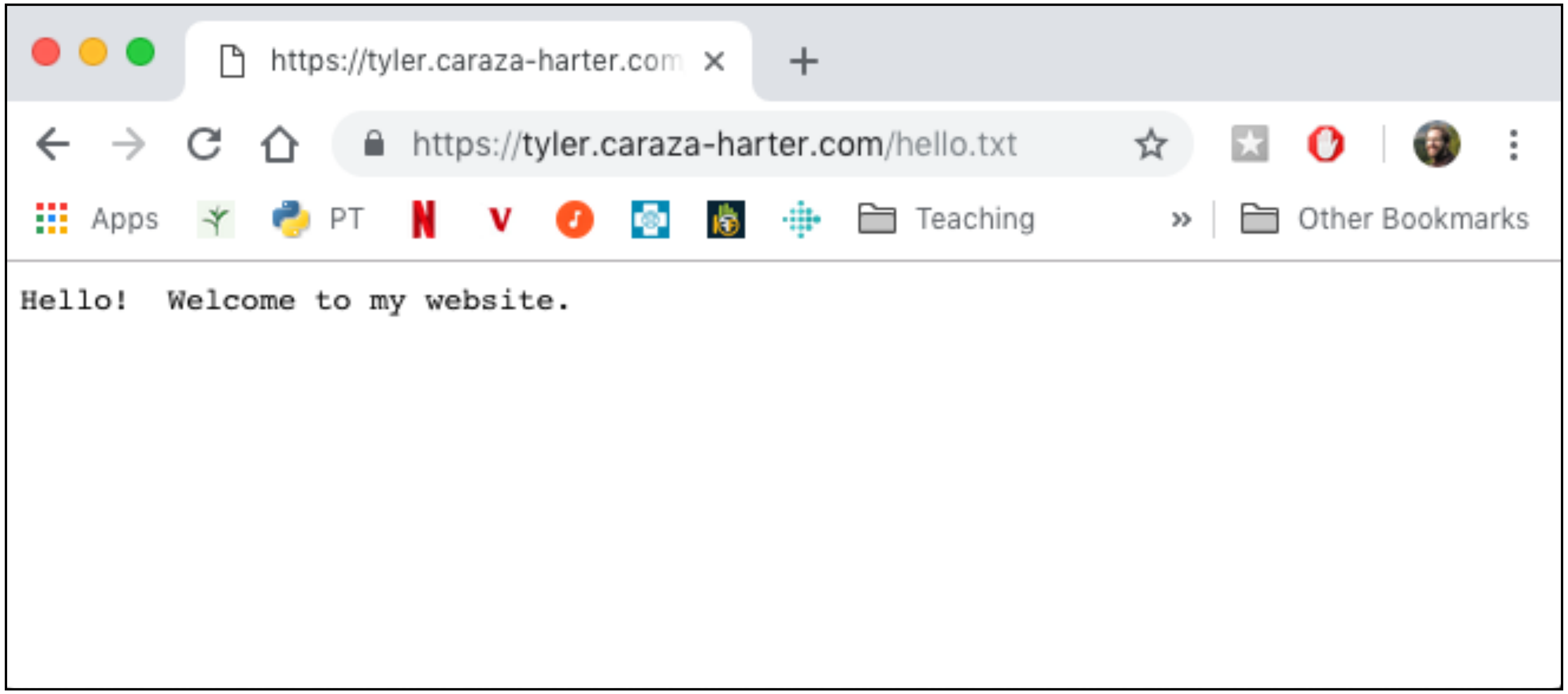
GET Request

```
import requests

url = "https://tyler.caraza-harter.com/hello.txt"

requests.get(url)
```

sends a **GET** request to tyler.caraza-harter.com,
asking for the contents of the **/hello.txt** page


https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/hello.txt
Apps
PT
Teaching
Other Bookmarks

```
Hello!  Welcome to my website.
```

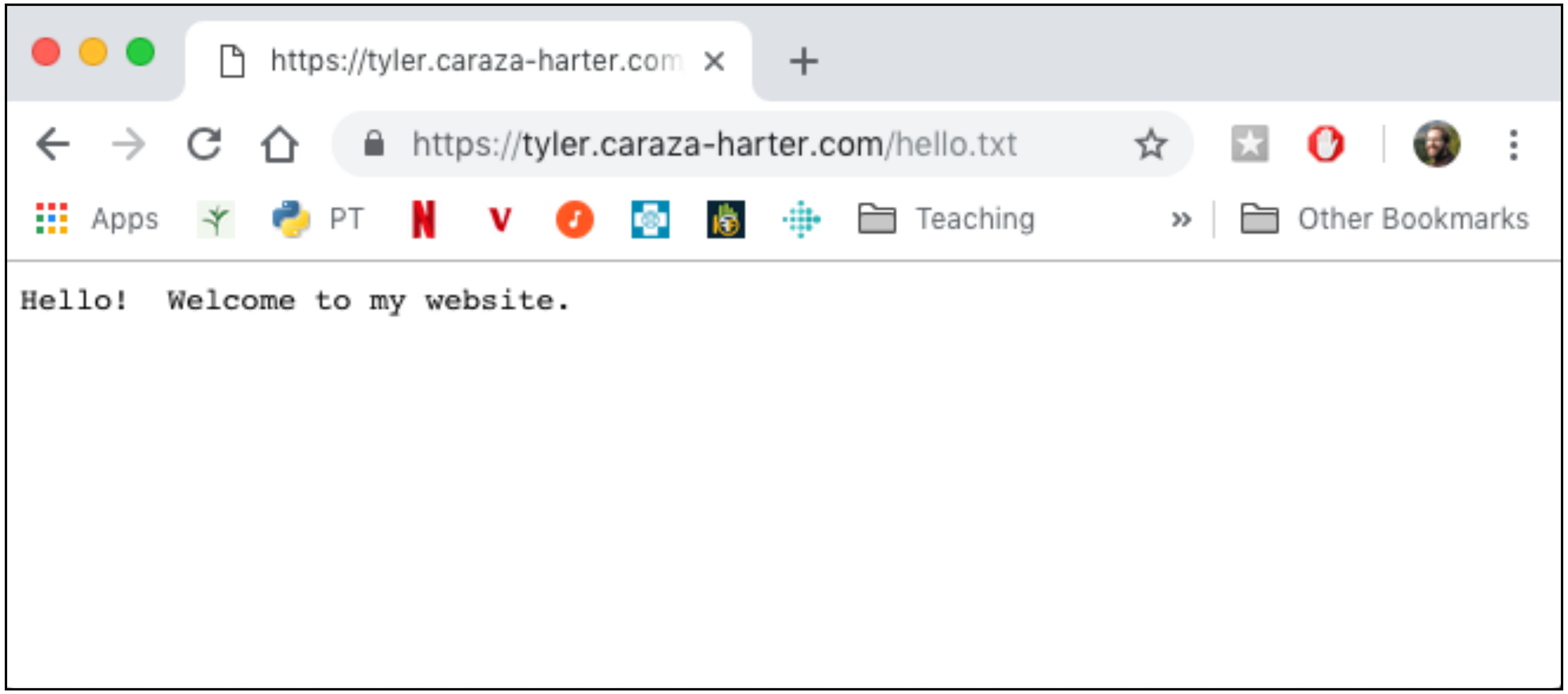
GET Request

```
import requests

url = "https://tyler.caraza-harter.com/hello.txt"

resp = requests.get(url)
```

put response from tyler.caraza-harter.com in the resp variable


https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/hello.txt
Apps
PT
Teaching
Other Bookmarks

```
Hello!  Welcome to my website.
```

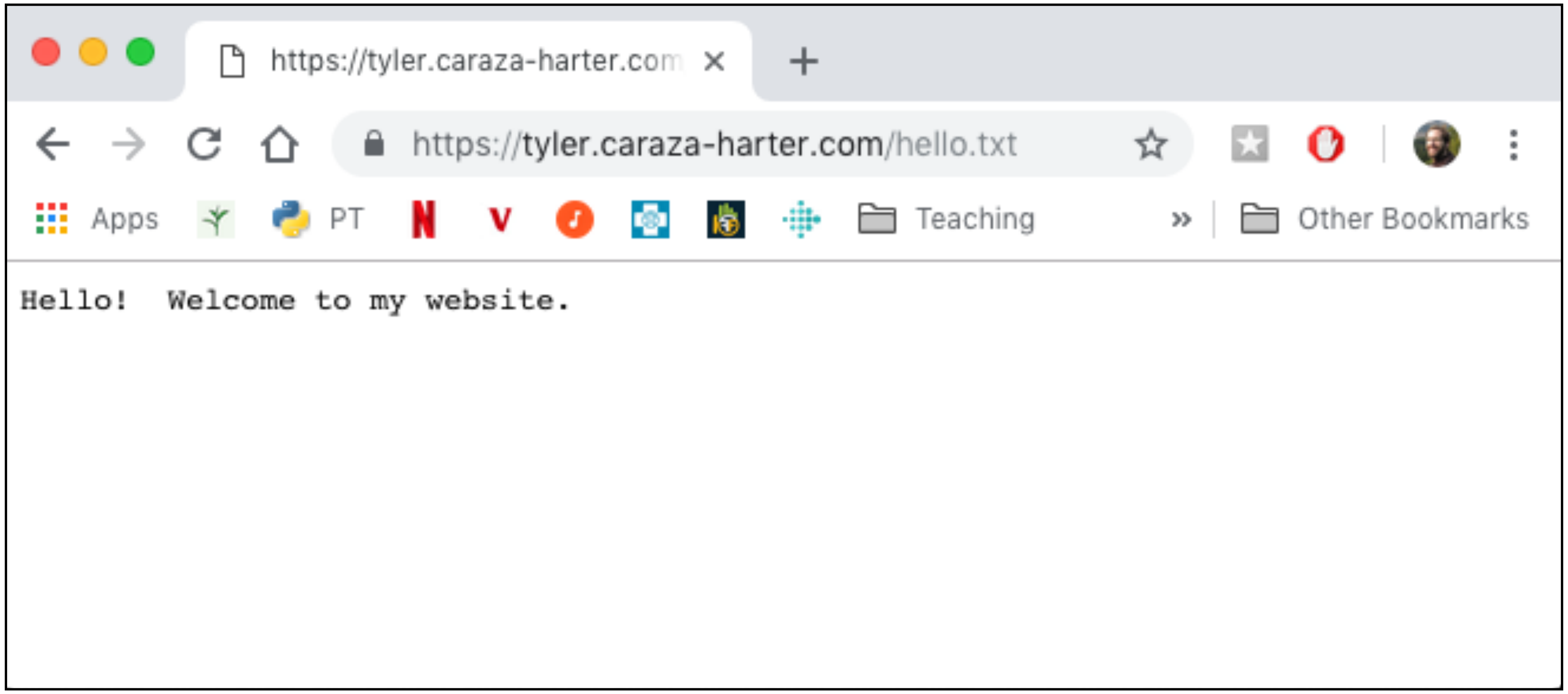
GET Request

```
import requests

url = "https://tyler.caraza-harter.com/hello.txt"

resp = requests.get(url)

# make sure we got 200 (success) back
assert(resp.status_code == 200)
```



https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/hello.txt
Apps
PT
Teaching
Other Bookmarks

```
Hello!  Welcome to my website.
```

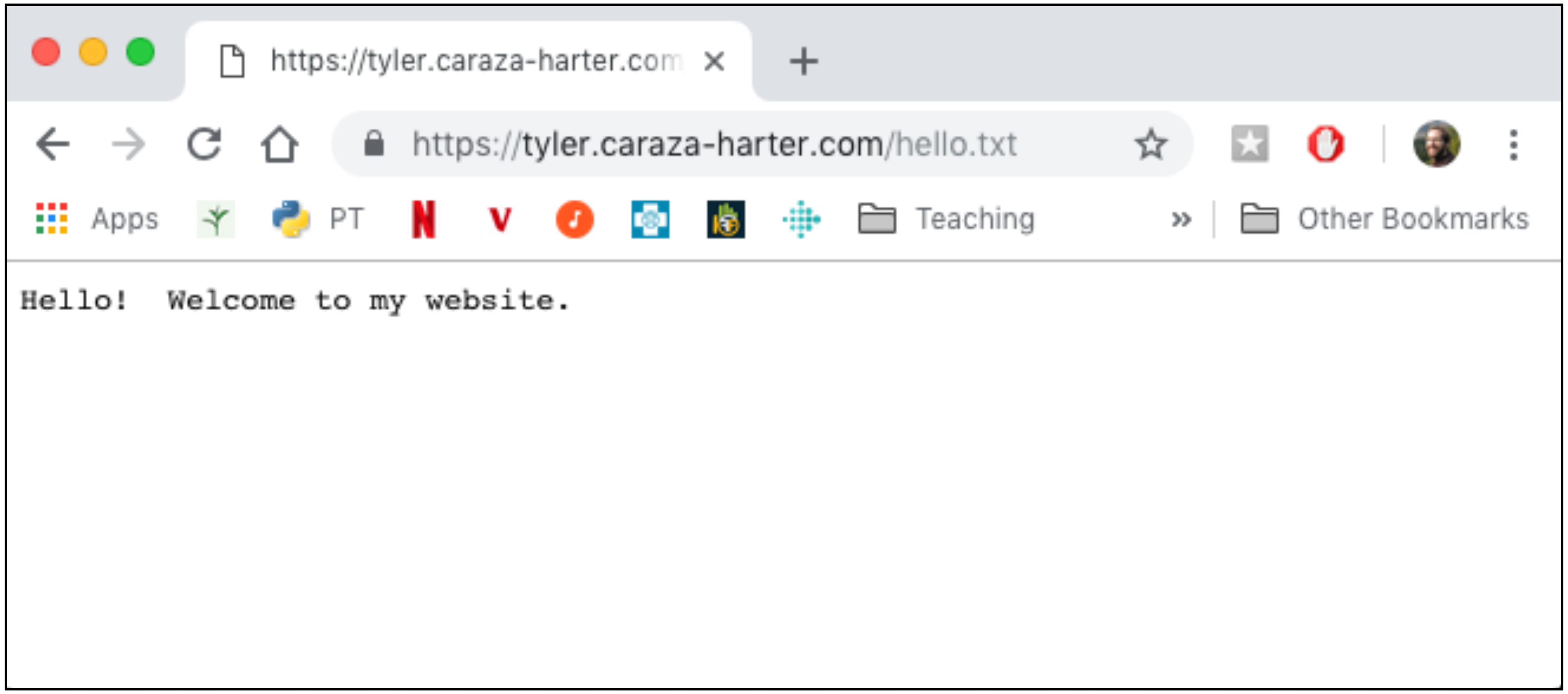
GET Request

```
import requests

url = "https://tyler.caraza-harter.com/hello.txt"

resp = requests.get(url)

resp.raise_for_status() # shortcut
```


https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/hello.txt
Apps
PT
Teaching
Other Bookmarks

```
Hello!  Welcome to my website.
```

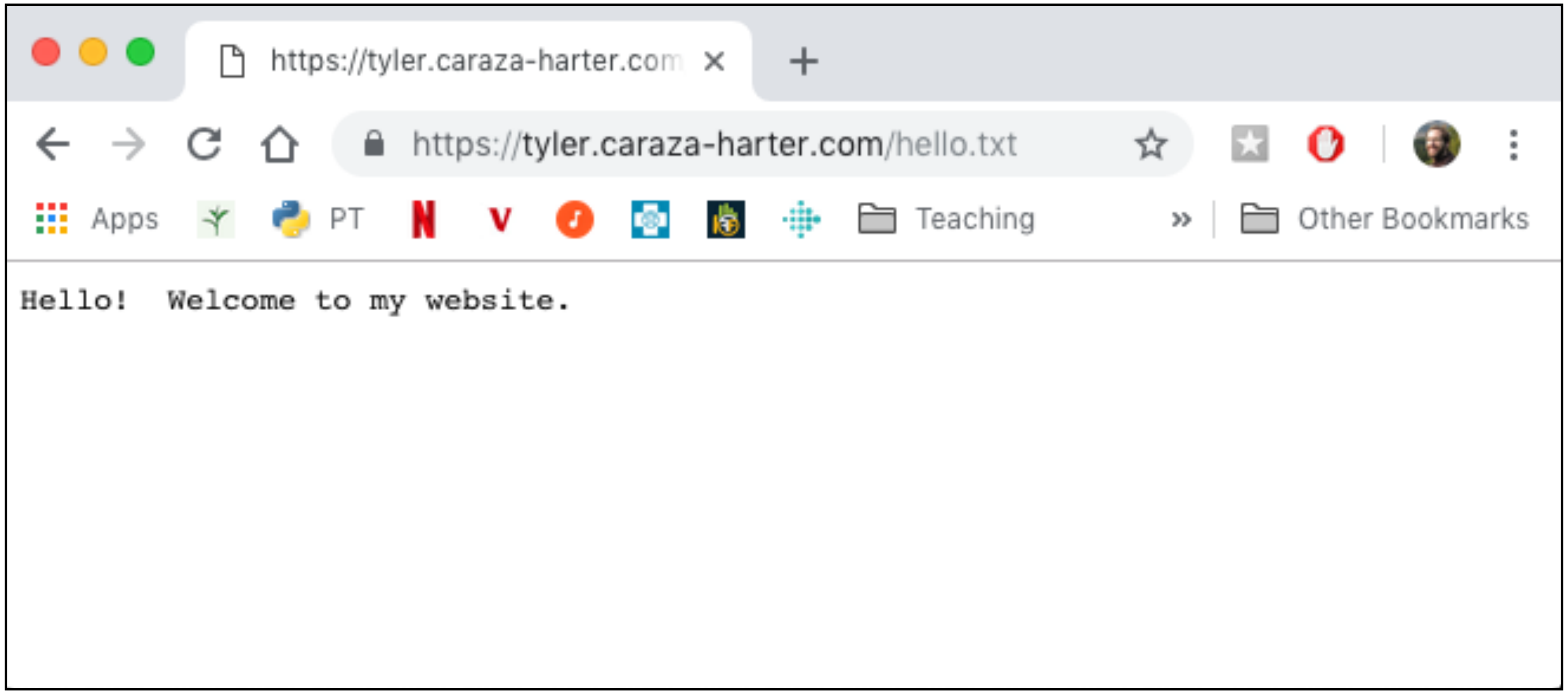
GET Request

```
import requests

url = "https://tyler.caraza-harter.com/hello.txt"

resp = requests.get(url)

resp.raise_for_status() # shortcut
print(resp.text) # "Hello! Welcome to my website."
```



https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/hello.txt
Apps
PT
Teaching
Other Bookmarks

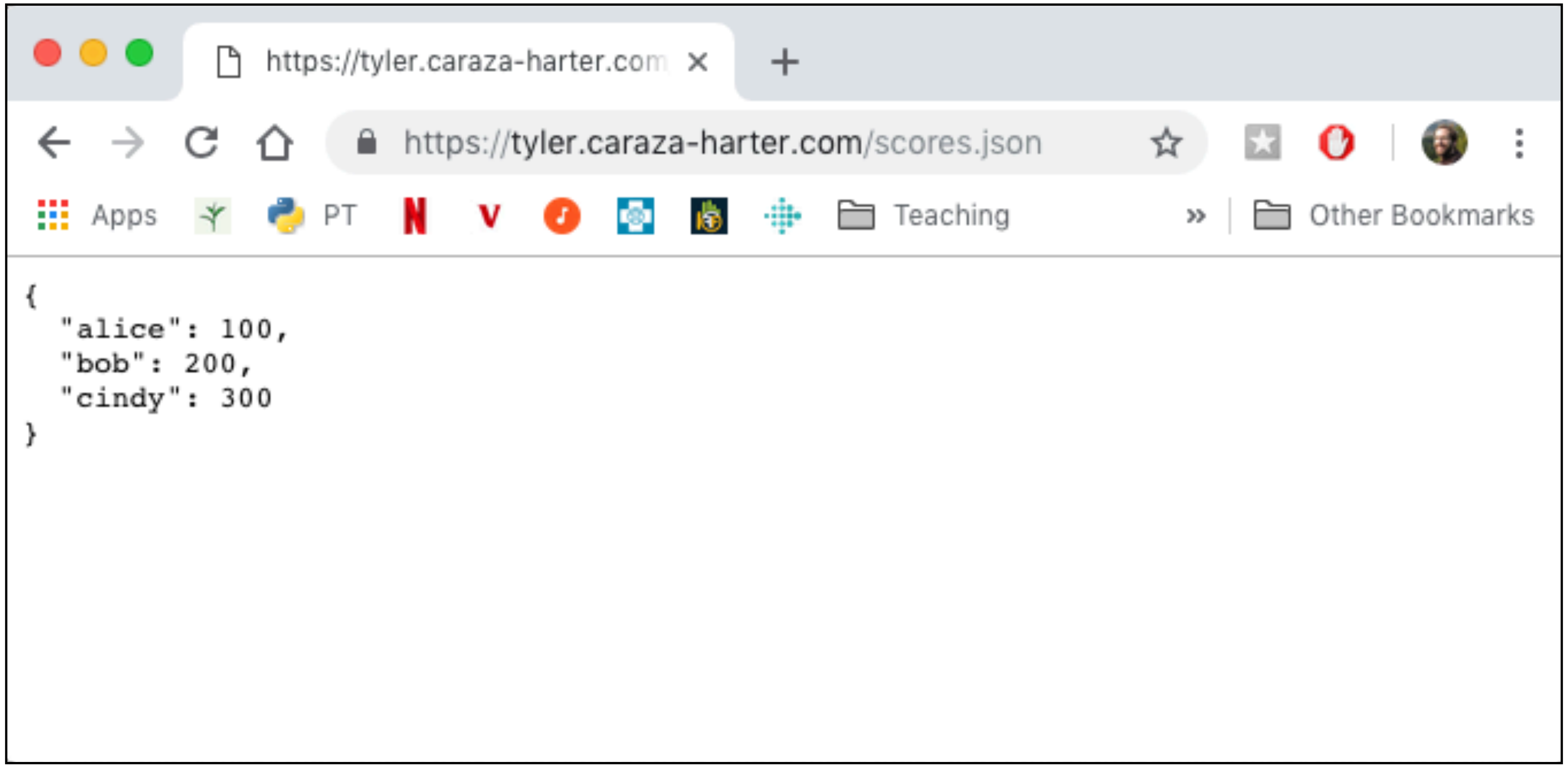
```
Hello!  Welcome to my website.
```

JSON Responses

```
import requests, json

url = "https://tyler.caraza-harter.com/scores.json"
resp = requests.get(url)

scores = json.loads(resp.text)
```



https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/scores.json
Apps
PT
Teaching
Other Bookmarks

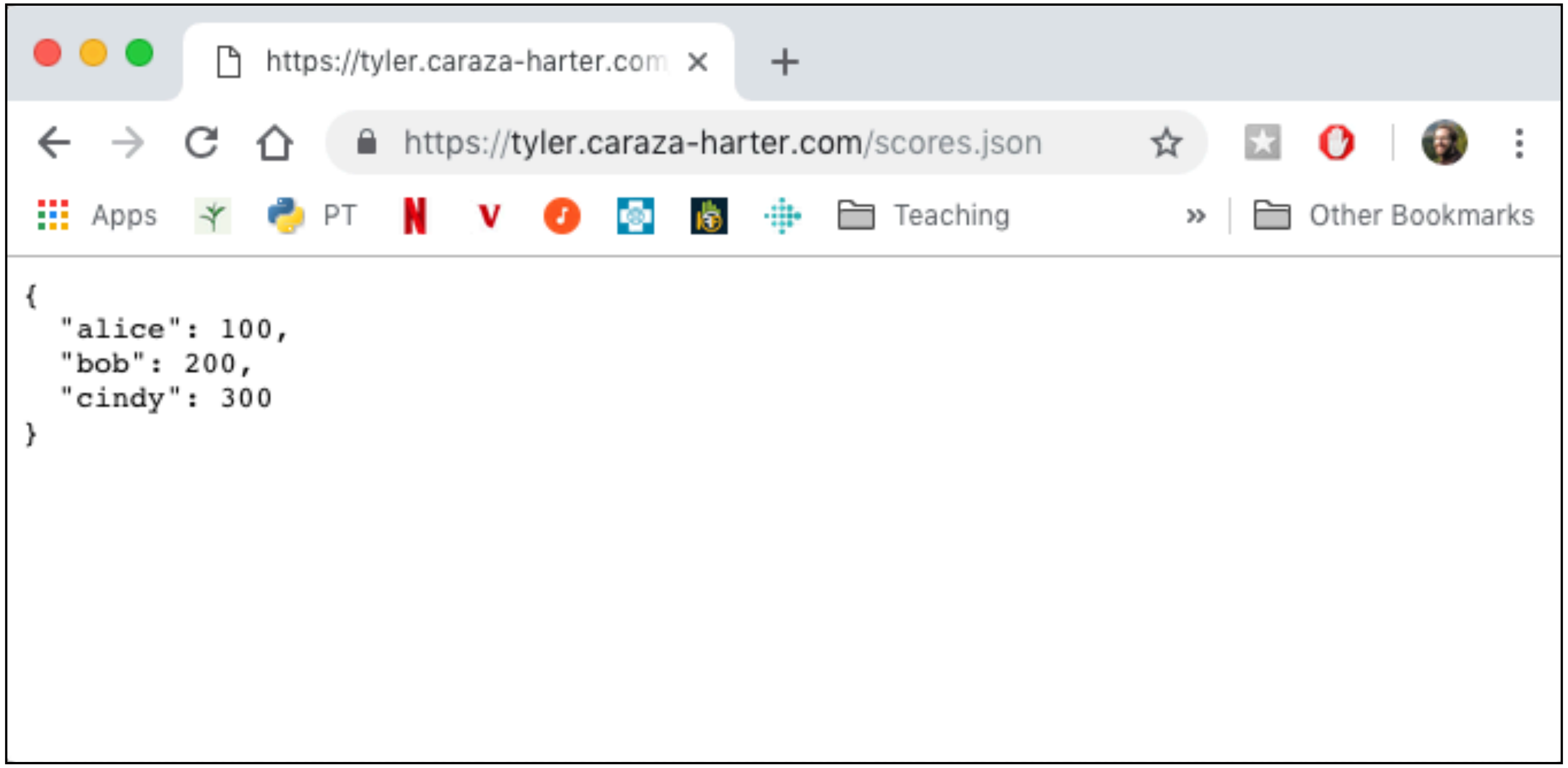
```
{
  "alice": 100,
  "bob": 200,
  "cindy": 300
}
```

JSON Responses

```
import requests~~, json~~

url = "https://tyler.caraza-harter.com/scores.json"
resp = requests.get(url)

~~scores = json.loads(resp.text)~~
scores = resp.json() # shortcut
```



https://tyler.caraza-harter.com
https://tyler.caraza-harter.com/scores.json
Apps
PT
Teaching
Other Bookmarks

```
{
  "alice": 100,
  "bob": 200,
  "cindy": 300
}
```

Demo 1: State Populations

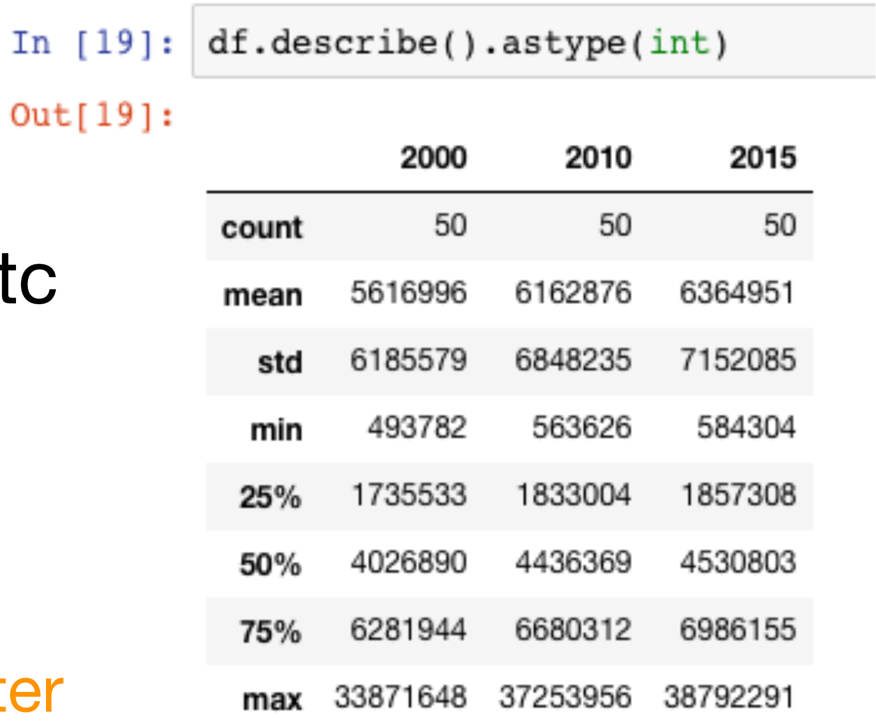
Goal: fetch population data for all states and provide summary stats

Input:

- List of state files: https://tyler.caraza-harter.com/cs301/spring19/materials/code/lec-30/data/state_files.txt
- The 50 JSON files

Output:

- Stats about population: mean, max, min, etc



In [19]:
```
df.describe().astype(int)
```

Out[19]:

	2000	2010	2015
count	50	50	50
mean	5616996	6162876	6364951
std	6185579	6848235	7152085
min	493782	563626	584304
25%	1735533	1833004	1857308
50%	4026890	4436369	4530803
75%	6281944	6680312	6986155
max	33871648	37253956	38792291

Bonus! "cache" results to make reruns of notebook faster

POST Request

```
import requests

url = "…"

requests.post(url, data)
```

POST Request

```
import requests

url = "…"

requests.post(url, data)
```

data to upload

Demo 2: Score Keeper

Goal: use POSTs and GETs to keep track of scores

Server Setup:

- `pip install flask`
- download https://raw.githubusercontent.com/tylerharter/caraza-harter-com/master/tyler/cs301/spring19/materials/code/lec-30/scores.py
- run this: `python scores.py`
- open `http://127.0.0.1:8080/` in web browser
- see code examples

to view scores:

`GET` `IP:PORT/scores`

to record score:

`POST` a player name to `IP:PORT/scores`