Project Part 1

Internet, Databases

Software Tools for Operations Research

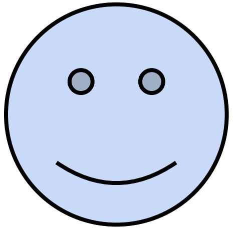
2014/01/28

Project Overview

- Over the next two classes will be working towards completing a project by teaching the skills necessary to make its various components.
- The project is “TSP-as-a-service”: creating an **internet service** that solves the **travelling salesman problem**

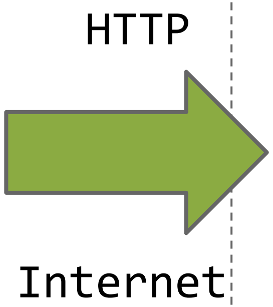
Project Overview

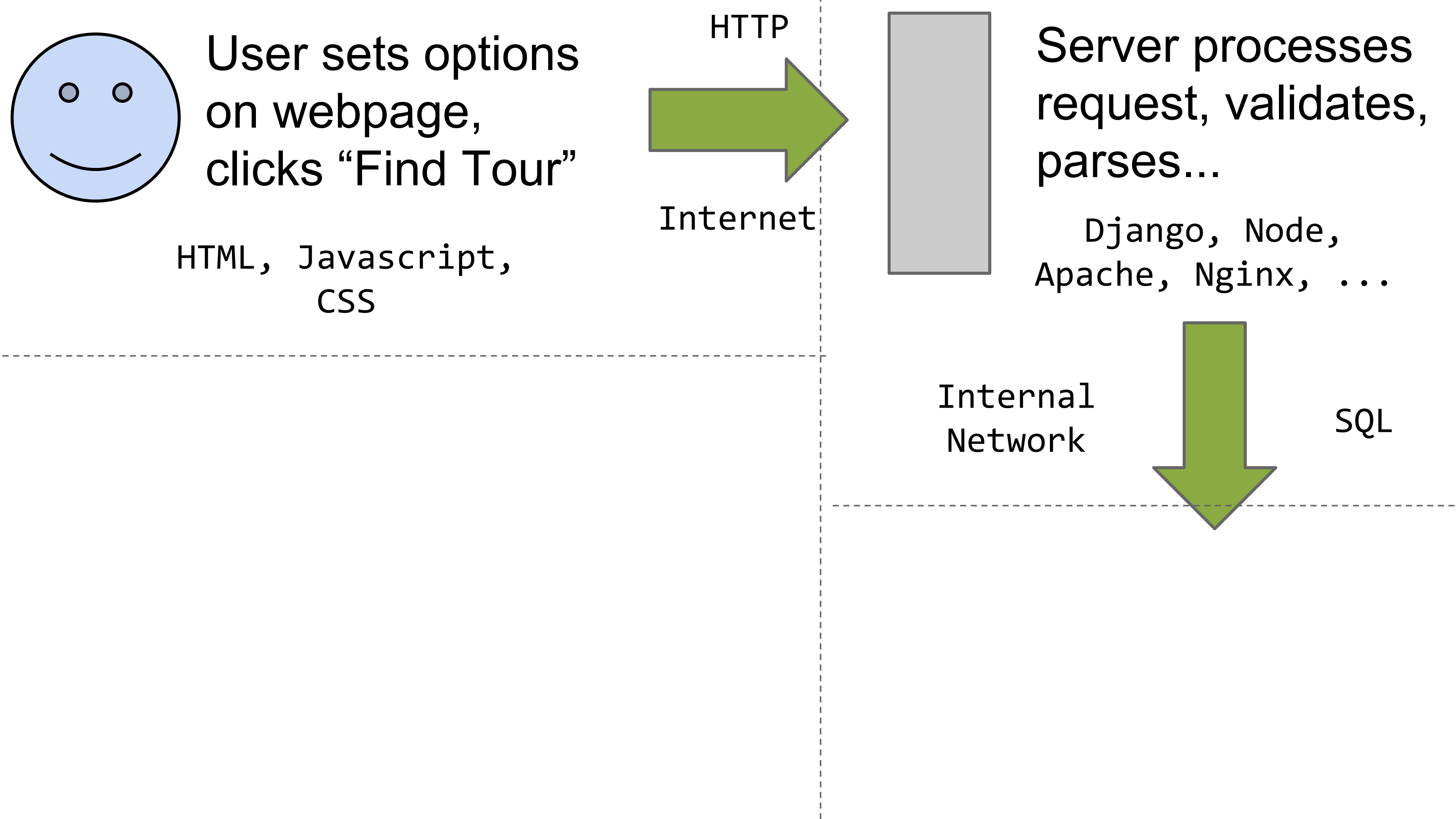
- The user will specify a set of Hubway stations, and the service will return the optimal tour over those stations.
- Diagram
- Demo
- Part 1: Server + Databases
- Part 2: MILP + Distribution



User sets options on webpage, clicks “Find Tour”

HTML, Javascript, CSS





User sets options on webpage, clicks “Find Tour”

HTML, Javascript, CSS

HTTP

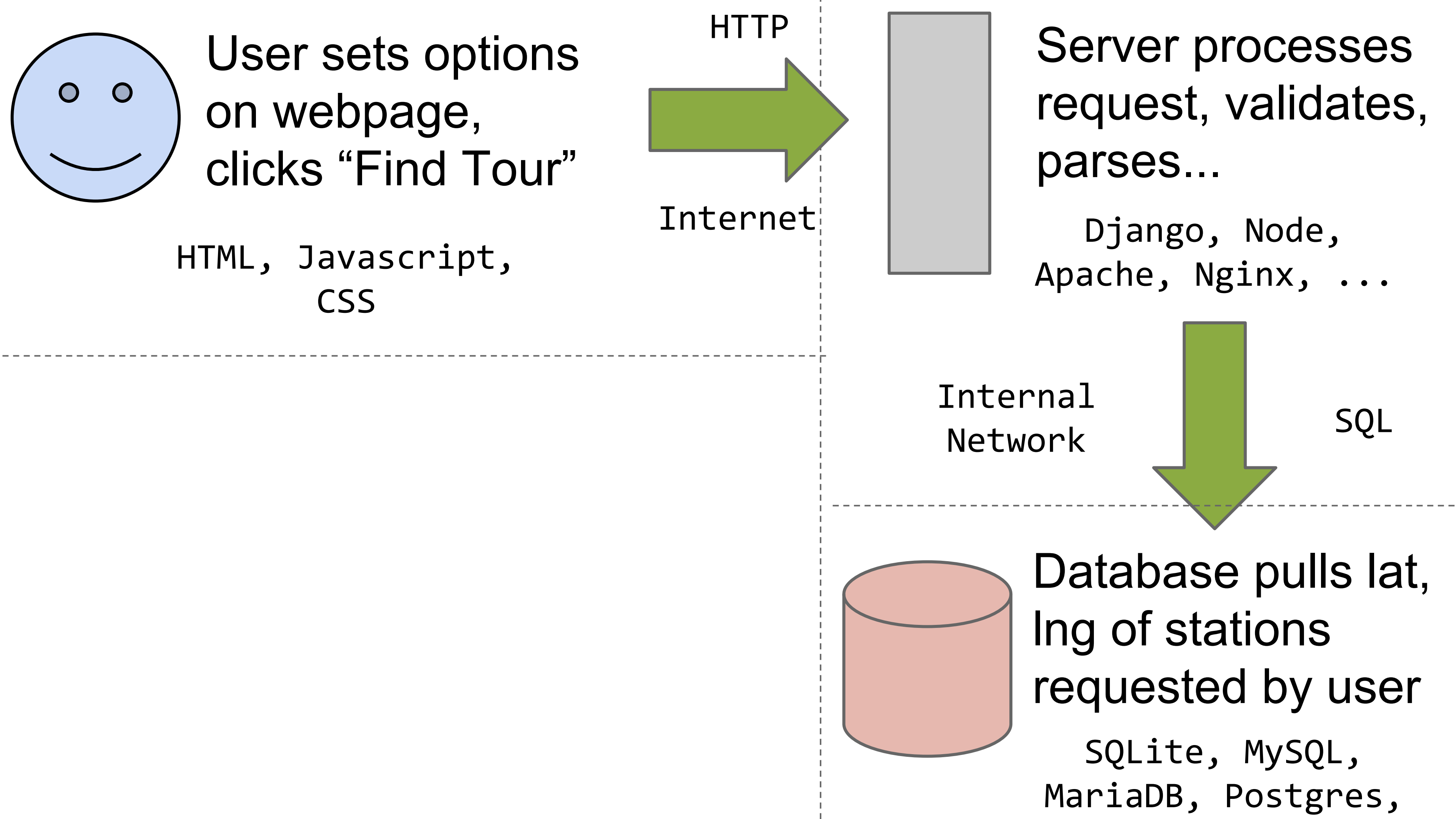
Internet

Server processes request, validates, parses...

Django, Node, Apache, Nginx, ...

Internal Network

SQL



User sets options on webpage, clicks “Find Tour”

HTML, Javascript, CSS

HTTP

Internet

Server processes request, validates, parses...

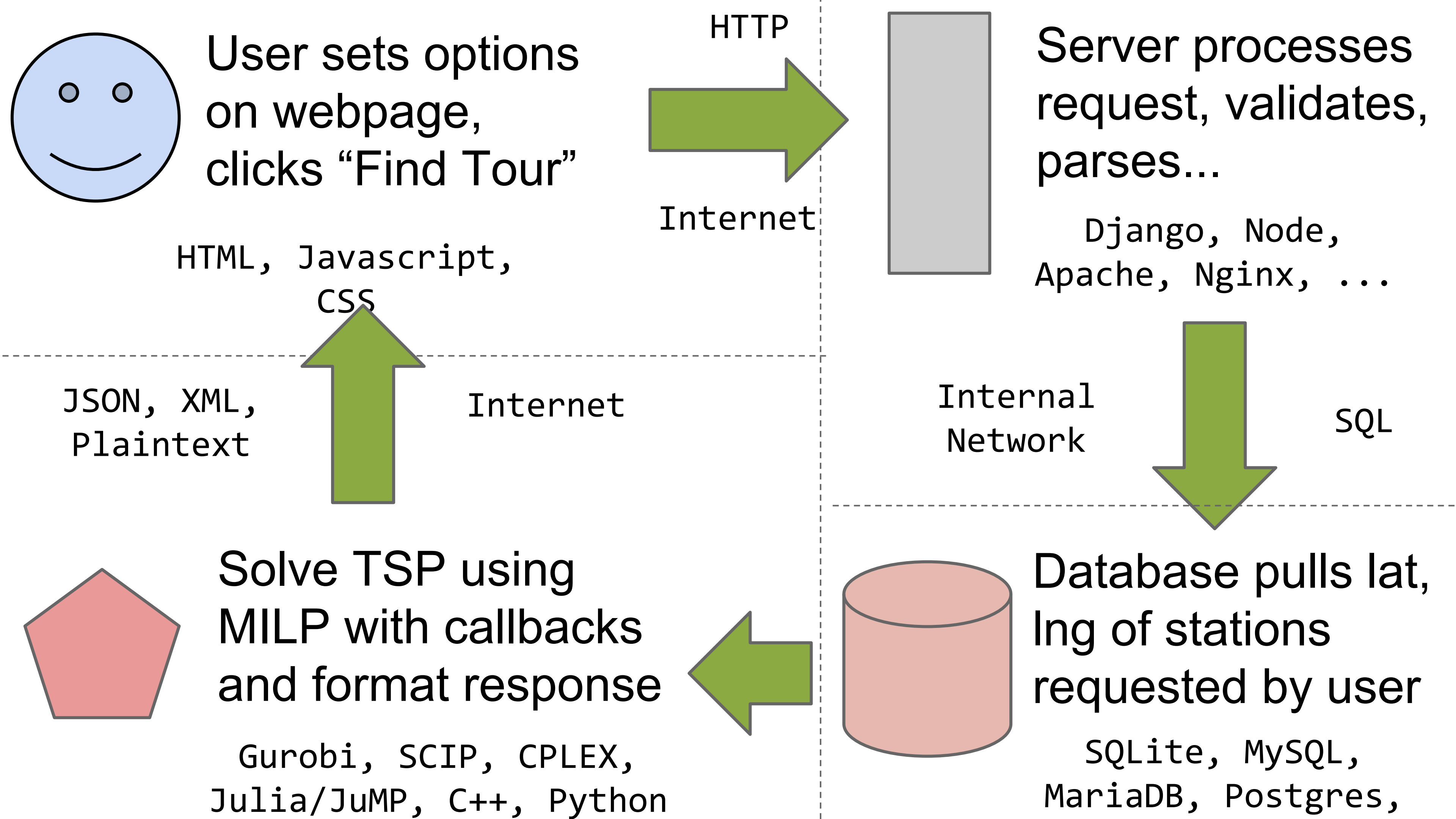
Django, Node, Apache, Nginx, ...

Internal Network

SQL

Database pulls lat, lng of stations requested by user

SQLite, MySQL, MariaDB, Postgres,



User sets options on webpage, clicks “Find Tour”

HTML, Javascript, CSS

HTTP

Internet

Server processes request, validates, parses...

Django, Node, Apache, Nginx, ...

Internal Network

SQL

Database pulls lat, lng of stations requested by user

SQLite, MySQL, MariaDB, Postgres,

Solve TSP using MILP with callbacks and format response

Gurobi, SCIP, CPLEX, Julia/JuMP, C++, Python

JSON, XML, Plaintext

Internet

Demo

```
julia projpart1_solution.jl
```

Wait, Why Are We Doing This?

- The days of distributing software to people's individual computers are numbered
  - Mobile computing
  - Cloud computing
  - Licensing
  - Installation/maintenance
- If you are deploying custom OR software, it may be an advantage to have it in the cloud

Wait, Why Are We Doing This?

- Making your software available online is **not as hard as you might think**, and might be easier than setting up your research-based code on many computers - just do it once on **your** server
- **Databases**: if you work in/with industry, you will often find critical data inside databases

Wait, Why Are We Doing This?

- **MILP Callbacks**: you may know intellectually that you can solve many hard problems using MILP, but our classes do nothing to tell you how to actually solve them.
- **TSP** is perfectly scoped to be hard to solve without them, and not-so-bad to solve with them. Also a critical part of many bigger

Computing and Networking

- In the beginning, computers were big things at universities and military sites, and were connected via long-distance phone lines.
- Computing began to be used widely by big companies, but computers were still giant machines that took up whole rooms (IBM, “big iron”)

Computing and Networking

- People used “thin clients” to connect to the “mainframes” - that is, all storage and processing was done on central large computers, the user simply had a screen and keyboard.
- Networks joined other networks, the Internet evolved, local area networks/ethernet,...

Enter: the Personal Computer

- Computers got small, mainframes fell out of fashion (sort of) - most computing was happening on personal computers.
- Internet becomes pervasive
- As we move into 2000s, computers continue to shrink and connectivity is fantastic

“Everything old is new again”

- Mainframes -> servers, the cloud
- Key difference is no longer monolithic computers, instead clusters of **commodity hardware wrapped/abstracted to be a general pool of computing power**.
- Internet/web browser provides a **common platform** to distribute software that runs in the cloud - the **browser** is the new thin client

Client-server Architecture

- **Clients**: individuals, e.g your phone, your laptop, your webbrowser, an ATM, …
- **Server**: shared resource, e.g. the machines in the ORC, the machines at Google that run searches, the machines that process financial transactions at your bank, ...

The Internet v.s. World Wide Web

- The **Internet** refers to the whole system of computers connected together worldwide using a common protocol.
- The **World Wide Web** runs on top of the internet: it is the collection of interconnected web pages we look at through our browsers
- **Email** and **BitTorrent** are separate services running on top of the internet.

The Internet

“They want to deliver vast amounts of information over the Internet. And again, the Internet is not something that you just dump something on. It's not a big truck. It's a series of tubes. And if you don't understand, those tubes can be filled and if they are filled, when you put your message in, it gets in line and it's going to be delayed by anyone that puts into that tube enormous amounts of material, enormous amounts of material.”

Senator Ted Stevens of Alaska

http://en.wikipedia.org/wiki/Series_of_tubes

The Internet - Addresses

- Computers have unique addresses, called “IP addresses”, e.g. `10.1.2.51`
- Each of the numbers ranges between 0 and 255 (the storage of 1 byte!)
- **MIT** has a whole block just for itself (1/256 of all address): `18.*.*.*`
- Thats 256^3 = 16.8 million addresses

The Internet - Domain Names

- We don’t typically use IP addresses in our day to day life - instead we use domain names
- DNS - Domain Name System
- e.g. `www.google.com` is looked up in global registry and converted to an IP address
- **Special address**: `localhost`, or `127.0.0.1` points to the computer you are on right now.

The Internet - Ports

- **Ports** are like mailboxes at an address. e.g. **port 80** is for **WWW**, and when you **ssh** into the ORC machines, you connect at **port 22**.
- You can have multiple servers/services running at one address, each one on a different port
- Most software will hide the port, e.g. browser automatically connect to port 80

Hypertext Transfer Protocol: HTTP

- **http**://www.google.com
- The protocol used to communicate on the world wide web.
- Don’t need to really understand the specifics!
- Clients make a **request** (e.g. GET, POST) for/to a **resource** (e.g. webpage, form submit)
- Servers respond with **content** and/or a **response** code

Ever Seen These?

`404 NOT FOUND`

`500 INTERNAL SERVER ERROR`

`503 SERVICE UNAVAILABLE`

But what you usually get is

`200 OK` (but you don’t see this)

People wrap this functionality in libraries so you don’t have to worry about dealing with it!

Technology Roundup

HTML
Describes structure of content on pages

CSS
Describes formatting of page

Javascript
Create programs that run in browser

HTTP
The protocol of the world wide web

Apache
A **webserver -** handles multiple concurrent requests, sends back pages, or passes requests to your software to do more work

Nginx
Another popular webserver

PHP, Node.js (Javascript), ASP (C#), Django (Python), ...
Popular server-side languages for making dynamic web applications. Takes requests, generates pages **dynamically** or returns a special response other than webpage...

XML, JSON, YAML, ...
Standard ways to encode structured information for communication

MySQL, MariaDB, Postgres, Mongo, ...
Databases commonly used in web applications.

Case Study: Healthcare.gov

- Connects with a lot of different **databases** administered by many government departments - this **service** talks to many other **services**.
- User enters information, service pulls information from variety of databases, **runs calculations** to determine eligibility
- Poor design led to inability to scale

Case Study: Healthcare.gov

- Scaling up complex systems and interacting with many databases is hard!
- Real OR software has to deal this these problems:
  - Inventory management software
  - Air traffic control software
  - Logistic network software
- “Reduce, reuse, recycle” helps!

Julia

- Julia, like most languages, has a package that makes making a webserver fairly easy
- HttpServer.jl: https://github.com/hackerschool/HttpServer.jl
- Basic idea is that you write a function that will return a response for each kind of request you want to handle.

Making our first service

- I’ll live code, then you will extend it.
- We’ll make a simple internet service.
- An complicated example:
  `https://www.google.com/#hl=en&q=cats`
- Our service will simply return a message:
  `http://localhost:4000/nameserve/Iain`

Live Coding Session with Julia

- You can follow along by reading `nameservice.jl`
- Or just watch me!

Assignment for the first half (Must Do)

1. Make nameservice return a different message each time you call it (e.g. randomly pick from 3 different options)
2. Add another service to our webserver that takes a request like `localhost:4000/addservice/5/3`
   and returns 8

Assignment for the first half (Optional)

1. Extend `addservice` to take an arbitrary number of numbers separated by slashes
2. Make a new service `localhost:4000/diceservice/n` that returns the average of n dice rolls
3. Anything you want!

Break

Any questions?

Storing Data

- In a research setting it is common to go out and collect data yourself, or maybe to generate data from your own programs
- You might store these in a **comma separated** file, or maybe a **white-space separated** file, or maybe in some other format you invented
- These are known as **“flat files”**

What is Wrong With That?

- Its not the worst thing in the world, but what do you do if...
- You want to share the file with someone?
- You want to find something quickly/efficiently?
- You want to remove/add rows?
- You do if you want to remove/add columns?
- The file is too big for your harddrive?

Enter: Databases

- “A database is an organized collection of data”
- We will discuss two types of databases
  - **Relational databases** - this is the most common style of database you’ll find, the focus for us today.
  - **Document stores/key-value stores** - a performance-sensitive type of database you may have heard about if you follow tech news.
- First, lets relate what we know (CSV) to DBs

Relational Databases

- MySQL, Postgres, Oracle, SQLite, MariaDB, MS SQL Server, even MS Access!
- All data is stored in **tables** inside a **database**
- Let's look at stations.csv from Hubway
- Header row: `id`, `name`, `lat`, `lng`
- `id` is an **integer**, and is **unique**
- `name` is a **string**
- `lat` and `lng` are **floats**

Relational Databases

- **`stations`** would be a **table** in our **database**
- The description of the columns, including their names and types, is called the **schema**
- The **`id`** column is the **primary key** of the table, it uniquely identifies each row
- We can use it to relate these rows to rows in another table, the `trips.csv` data, so it is also a **foreign key**

Relational Databases

- We can dump the schema of table to quickly understand it, like `str()` in R for dataframes
- We don’t concern ourselves with the details of how or even where the data is stored. e.g. the database may smartly copy some data into memory if we access it frequently, or may split it across disks transparently

Relational Databases

- The database software can build an **index** of the data, allowing efficient access for whatever data we want
- The schema can be a way of validating that data we store in the database is valid.
- We can do all the things we want: join two tables, add rows, add columns, etc. “efficiently”... but how?

SQL - Structured Query Language

- Pronounced Ess-Q-Ell or Sequel
- Language designed for working with relational databases
- Fairly intuitive, and can be very powerful:

```
SELECT *
FROM trips INNER JOIN stations
ON trips.start_station = stations.id
WHERE stations.lng >= -71.05
```

SQL Primer

- Many resources to help you with SQL online
- Basics commands
    - **SELECT** - extract data http://en.wikipedia.org/wiki/Select_(SQL)
    - **INSERT** - add data
    - **FROM** - the source of the data
    - **WHERE** - conditions the rows must meet http://en.wikipedia.org/wiki/Where_(SQL)

SELECT examples

```
SELECT columns       e.g. duration
FROM tablename       e.g. trips
WHERE condition      e.g. gender = ‘Male’
```

```
SELECT name, lat, lng
FROM stations
WHERE name LIKE ‘%MIT%’
```

JOIN

- Recall in “Intermediate R” we had two dataframes, stations and trips, and want to make a new dataframe where the rows from trips were augmented by the station data?

```
merged = merge(trips, stations,
            by.x="start_station", by.y="id")
final  = merge(merged, stations,
            by.x="end_station",   by.y="id")
```

JOIN

- We can perform similar operations with SQL
- There are multiple types of joins, e.g.
  - CROSS JOIN - Cartesian product i.e. all possibilities
  - INNER JOIN - Subset of Cartesian product that matches our condition, usually and equality test. This is what we did in R
  - LEFT OUTER JOIN - always has all records from “left” table, and will match from “right” where possible, NULL otherwise

Equivalent of R statements

```
SELECT duration,
       A.name AS start_name,
       B.name AS end_name
FROM trips
LEFT JOIN stations A ON trips.start = A.id
LEFT JOIN stations B ON trips.end   = B.id;
```

Aside: SQL for Dataframes?

- R package `sqldf` (http://code.google.com/p/sqldf/)
- Can be surprisingly fast (they claim)

```
> library(sqldf)
> sqldf("select * from iris order by Sepal_Length desc limit 3")

  Sepal_Length Sepal_Width Petal_Length Petal_Width   Species
1          7.9         3.8          6.4         2.0 virginica
2          7.7         3.8          6.7         2.2 virginica
3          7.7         2.6          6.9         2.3 virginica
```

Connecting to Databases

- Databases act a bit like servers themselves
- All languages will have ways to connect to databases
- Usually have some sort of “connection” object, through which queries can be submitted
- Details vary by implementation

Popular simple DB: SQLite

- **SQLite** is very light-weight, runs as part of program instead of separate server.
- Database stored in a simple file you can move around easily.
- Runs in memory - pretty fast.
- Everywhere: inside major browsers (e.g. Firefox bookmarks), Skype, Android, iOS, …
- Julia package: SQLite.jl

Live Coding

Exercise

- Combine the SQL queries you have seen / look at the Wikipedia pages to write code that will extract all **trips** with **duration longer than 2000** that **start at a station with Harvard in the name**
- Can you figure out how to just get the 5 longest trips? Can you find the average duration?

Document Stores/NoSQL

- Not all databases are relational databases
- A popular style lately is to store data as simple key-value pairs
    - ‘Associate map’, ‘Dictionary’, etc.
- The documents may follow a schema, but it isn’t enforced, could be versioned, etc.
- Examples include MongoDB, Cassandra, Neo4J, … http://en.wikipedia.org/wiki/NoSQL

Key-Value Example - stations

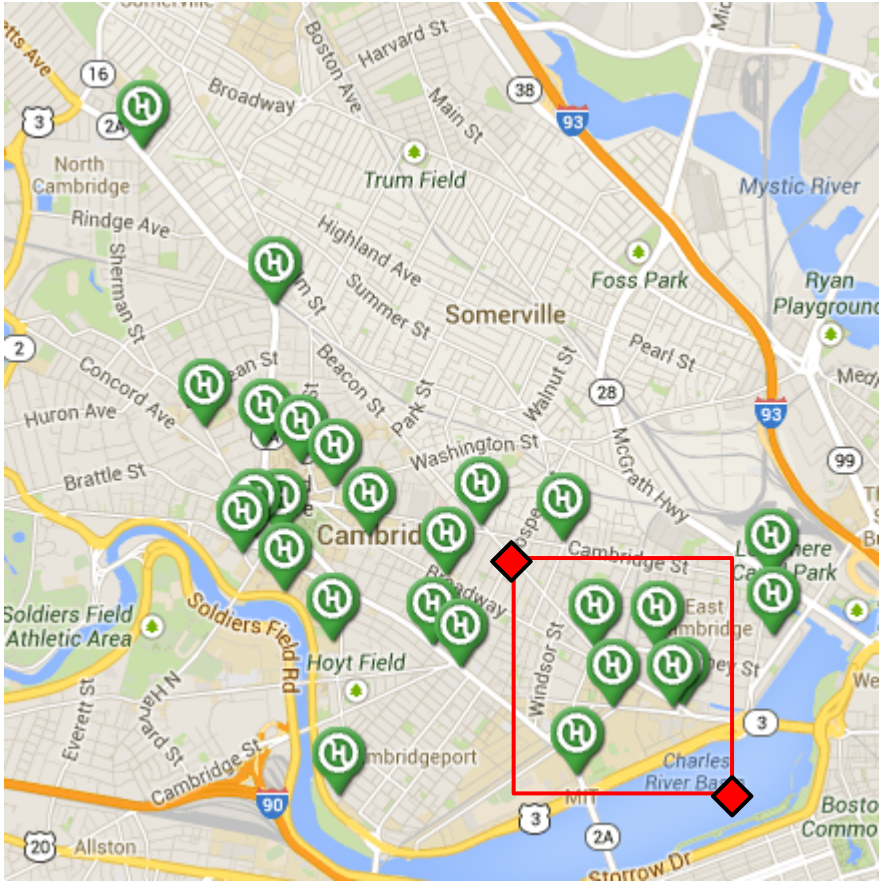
```
3 : { name: “Colleges of the Fenway”,
       lat:  42.340021,
       lng: -71.069616 }
7 : { name: “Fan Pier”,
       lat:  42.363412,
       lng: -71.044624}
```

Why Might You Want This?

- Want to extract rows at random as quick as possible.
- Rows vary a lot, i.e. highly variable amount of information in each.
- Data is inherently graph-based
- Distributed file-systems
- Probably best to stick to relational DBs unless know you need it

Project Part 1

- Our service will take in two (lat,lng) pairs
- All stations in this box will be part of our tour
- To solve the TSP, we need the Euclidean distance between all the pairs



Project Part 1

- We wish to build a webservice that will take four numbers (the two lat/lng pairs)
- Create a SQL query to extract the stations inside that box.
- Calculate the pairwise distance between the stations.
- We would then normally use the pairwise distance matrix for solving the TSP - next time!

Project Part 1

1. Create a webservice that accepts four numbers and checks they are valid. Return "yes" if OK, an error if not. (`addservice`)
2. Write a function that, given four numbers, will create an SQL query to extract stations inside the box defined by the numbers.

continued...

Project Part 1

3. Write a function that, given a matrix with N rows and two columns, returns an N by N symmetric matrix of distances (Great Circle or otherwise)

4. Combine these parts together to make a webserver that returns the distance matrix when given a box.

Useful Tips

http://www.w3schools.com/sql/sql_where.asp

http://www.w3schools.com/sql/sql_between.asp

```
string(“x >= “, 5)  # “x >= 5”
```

http://en.wikipedia.org/wiki/Great_circle_distance

Try to do something and gracefully handle it not working:

http://docs.julialang.org/en/latest/manual/control-flow/#the-try-catch-statement