Psychoinformatics & Neuroinformatics

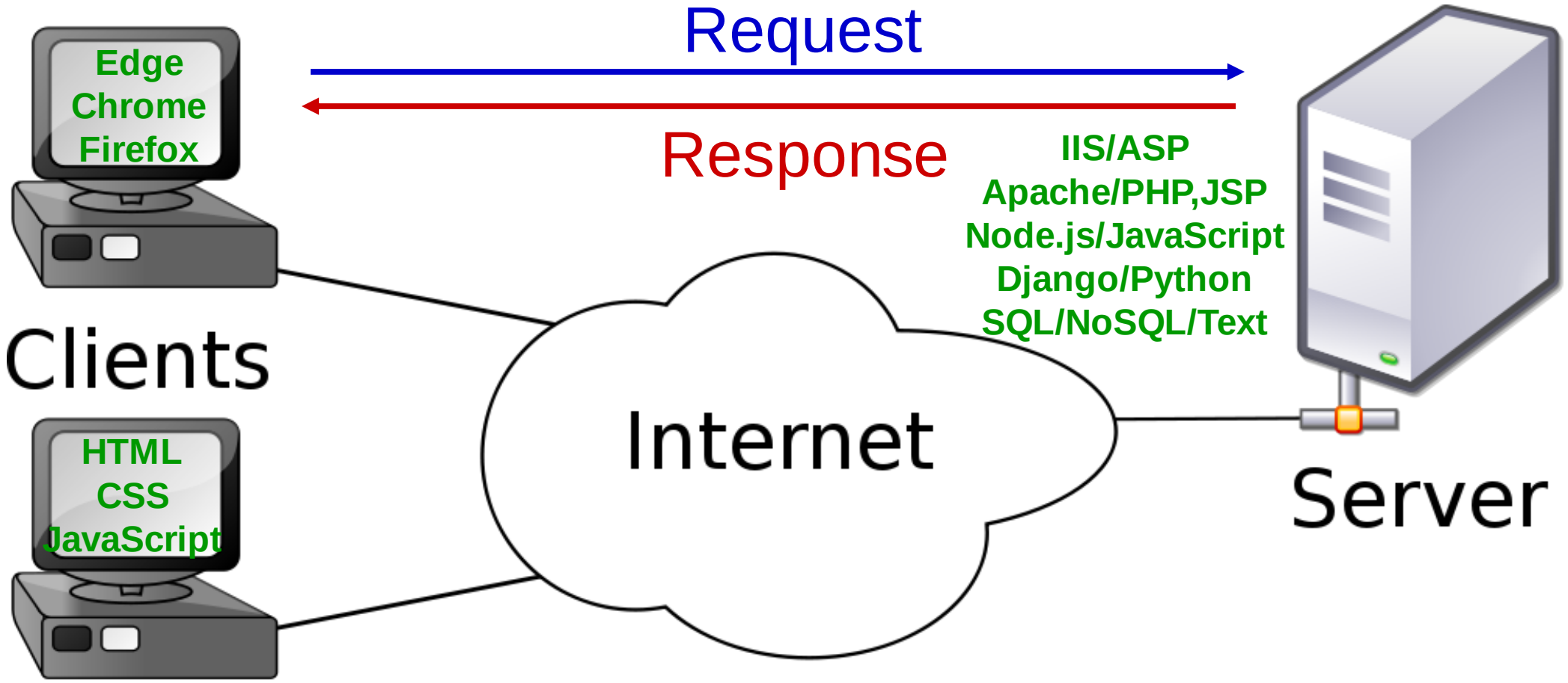
Week 7

Web Backend

by Tsung-Ren (Tren) Huang 黃從仁

Frontend vs. Backend

Client=Frontend for getting data; Server=Backend for giving data



Why bother backend development

Making APIs for other people

To provide functionality or data

Making APIs for other devices

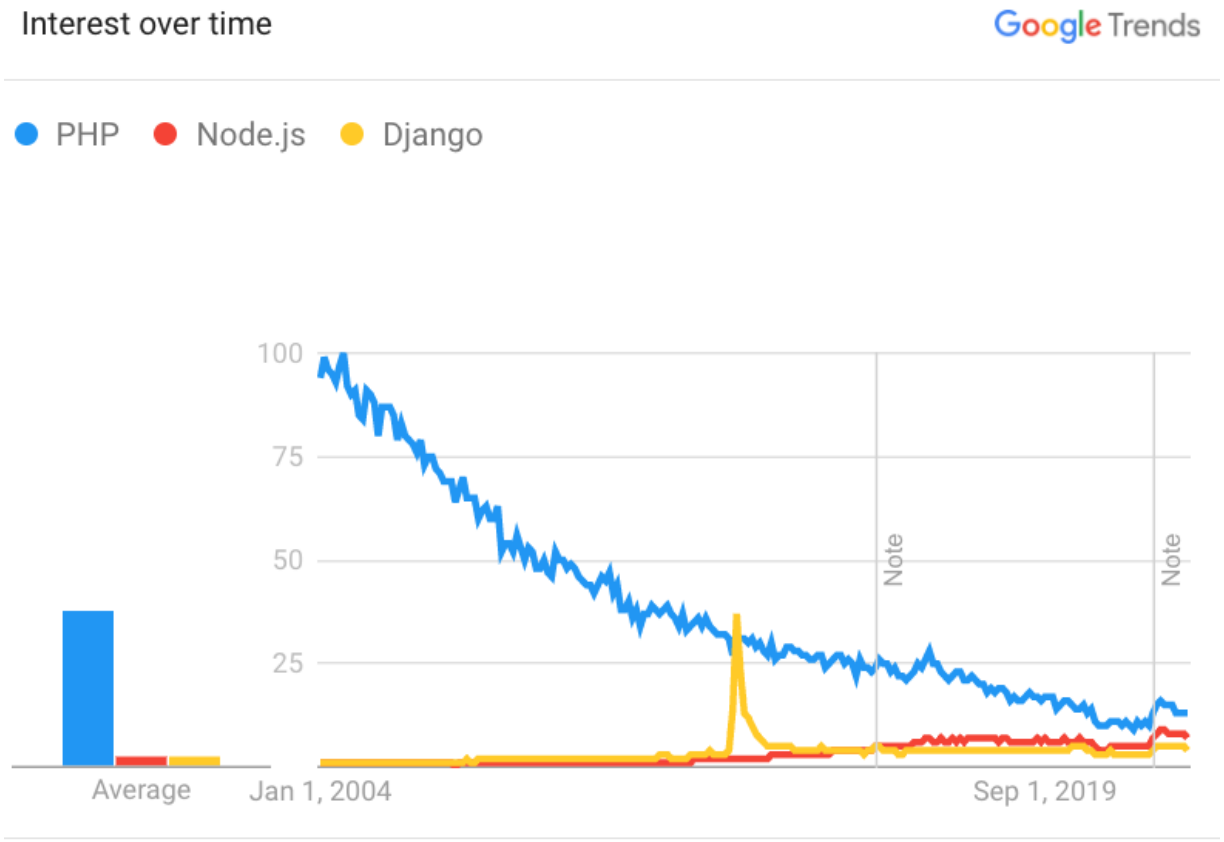
To augment computation and storage

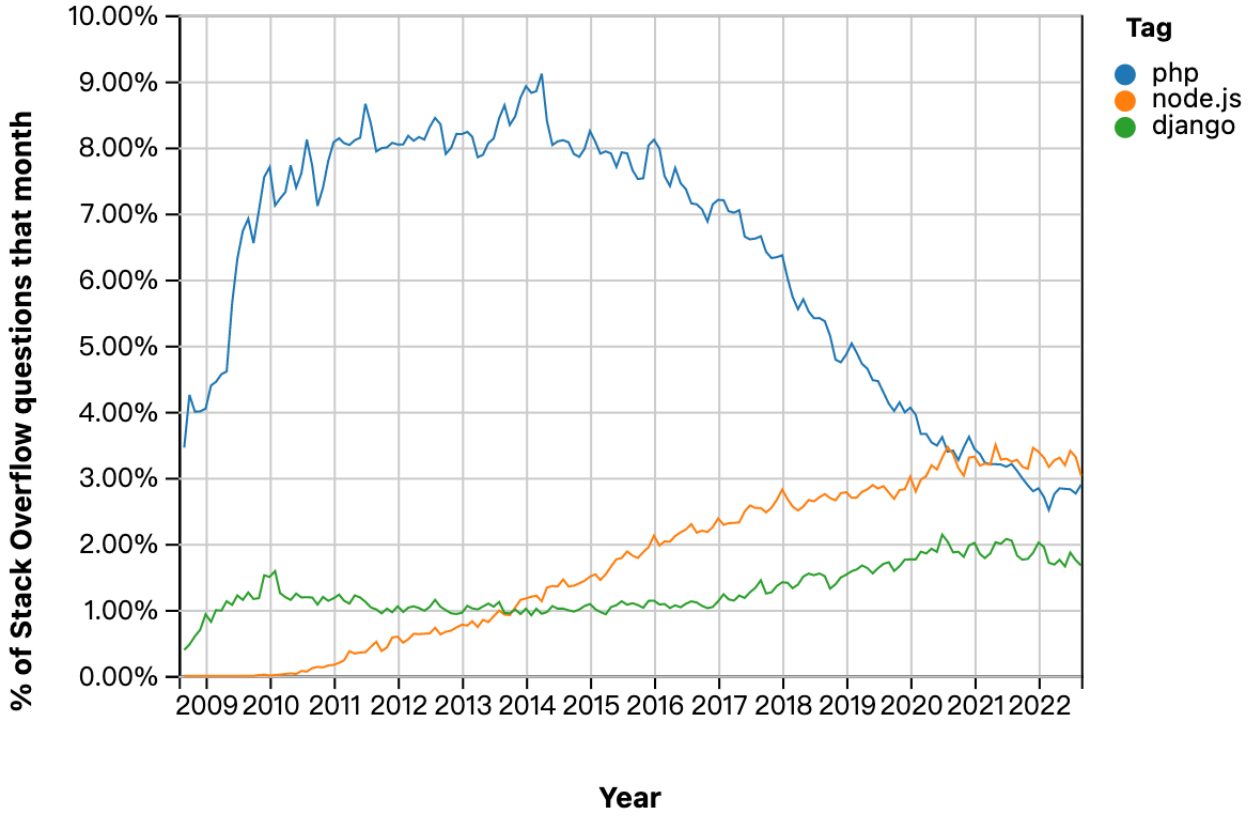
Making APIs for ourselves

To separate frontend and backend development

PHP vs. Javascript vs. Python (1/3)

PHP used to be the go-to choice



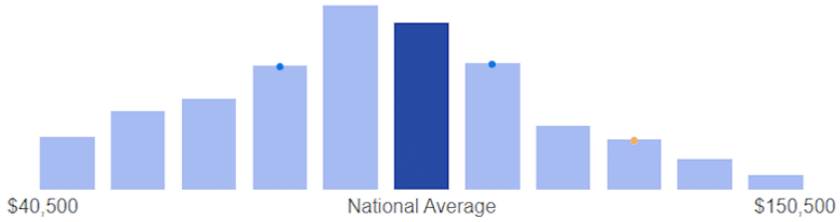


PHP vs. Javascript vs. Python (2/3)

PHP skills are less valuable nowadays

Laravel Developer Salary

Yearly Monthly Weekly Hourly Table View



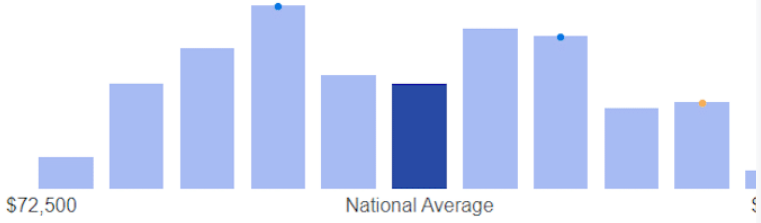
$92,285 /year
$44 /hour

Laravel Developer Salary Comparison by Location

Nationwide United States	$92,285
Chicago, IL	$94,470

Node JS Developer Salary

Yearly Monthly Weekly Hourly Table View



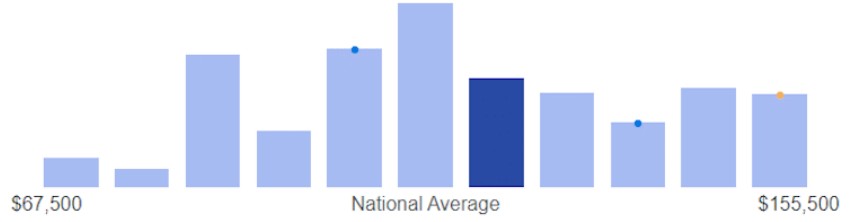
$113,791 /year
$55 /hour

Node JS Developer Salary Comparison by Location

Nationwide United States	$113,791
Chicago, IL	$116,487

Python Django Developer Salary

Yearly Monthly Weekly Hourly Table View



$117,673 /year
$57 /hour

Python Django Developer Salary Comparison by Location

Nationwide United States	$117,673
Chicago, IL	$120,460

PHP vs. Javascript vs. Python (3/3)

PHP is best for more static websites

e.g., Wikipedia & Wordpress

Javascript is best for dynamic websites

e.g., Twitter & multiplayer games

Python is best for intelligent websites

e.g., ML- or DL-powered Apps

Goals for today

Learning the Javascript way

Node.js (+Express)

Learning the Python way

FastAPI

Learning the basic SQL

SQLite

Goals for today

Learning the Javascript way

Node.js (+Express)

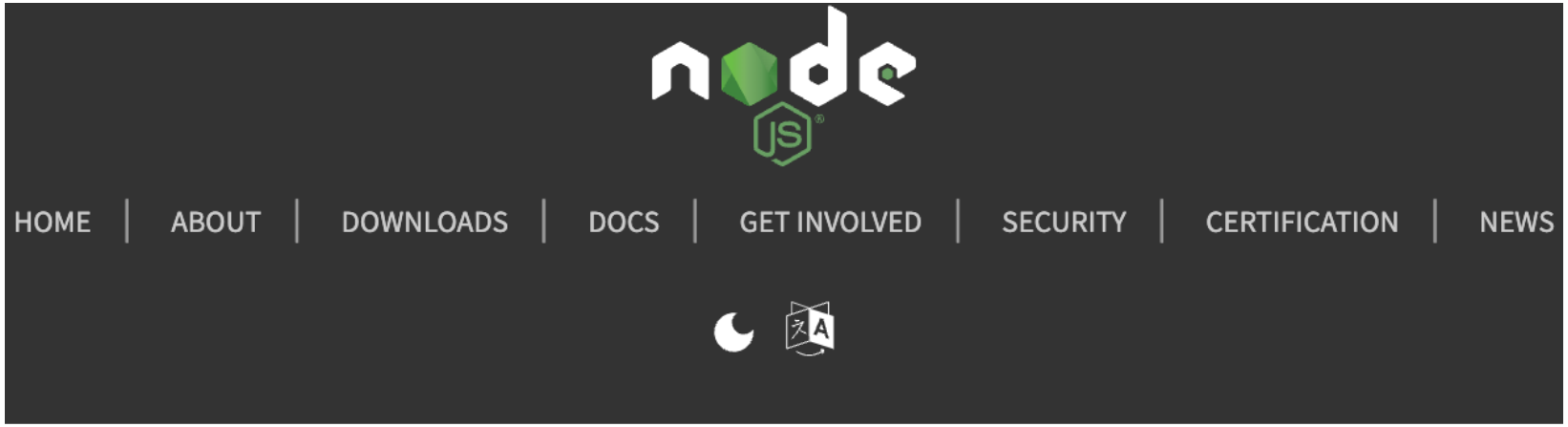
Learning the Python way

FastAPI

Learning the basic SQL

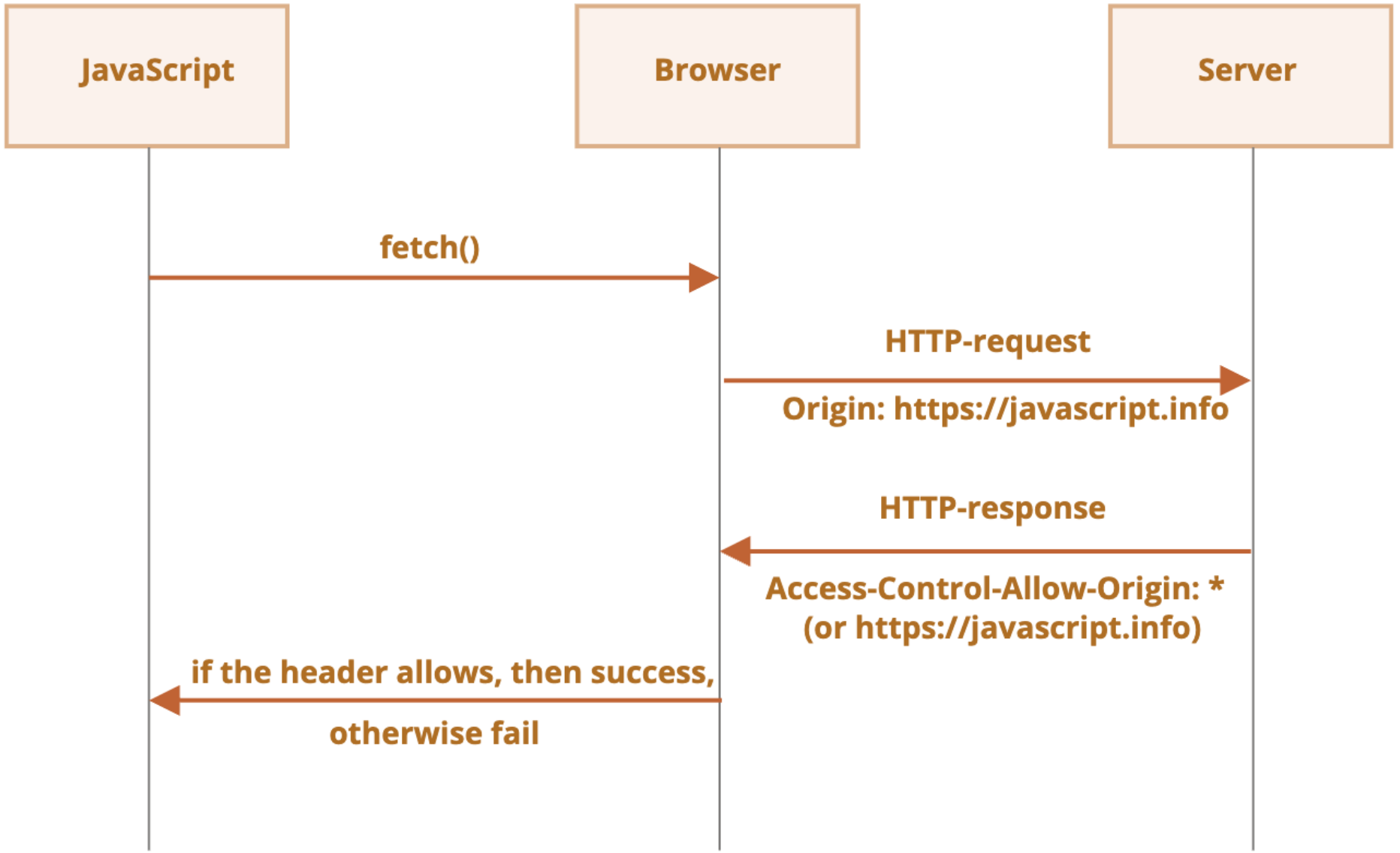
SQLite

Node.js + Express: npm i express cors



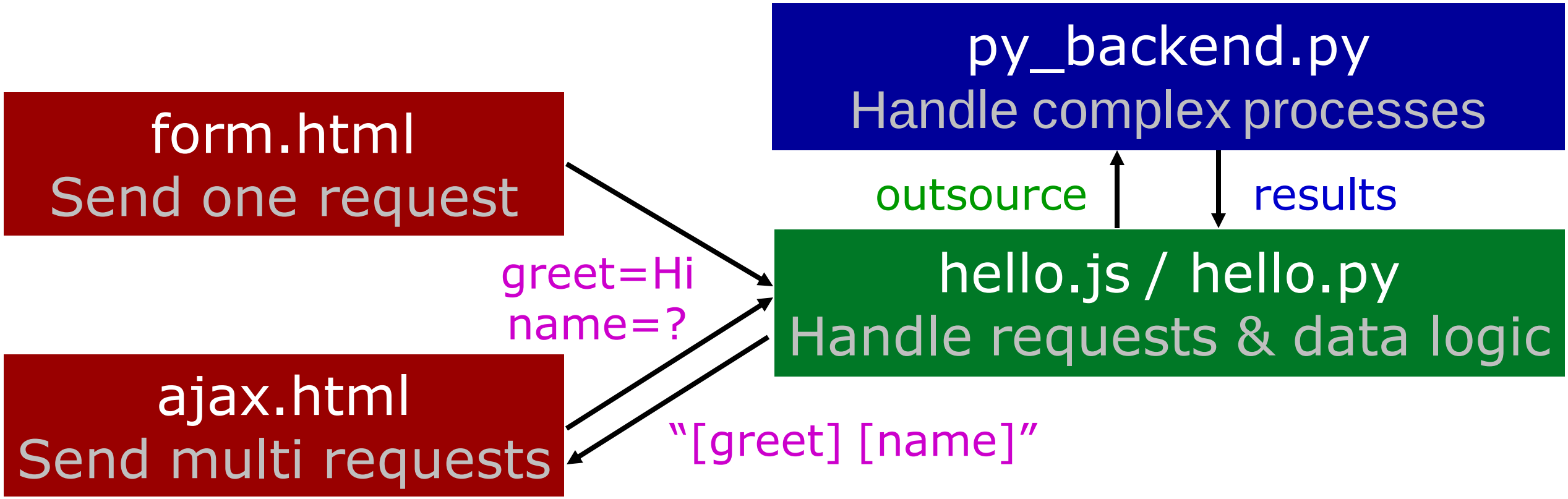
Node.js® is an open-source, cross-platform JavaScript runtime environment.

Cross-Origin Resource Sharing (CORS)



Requests & Responses (1/2)

Backends respond to the requests from frontends

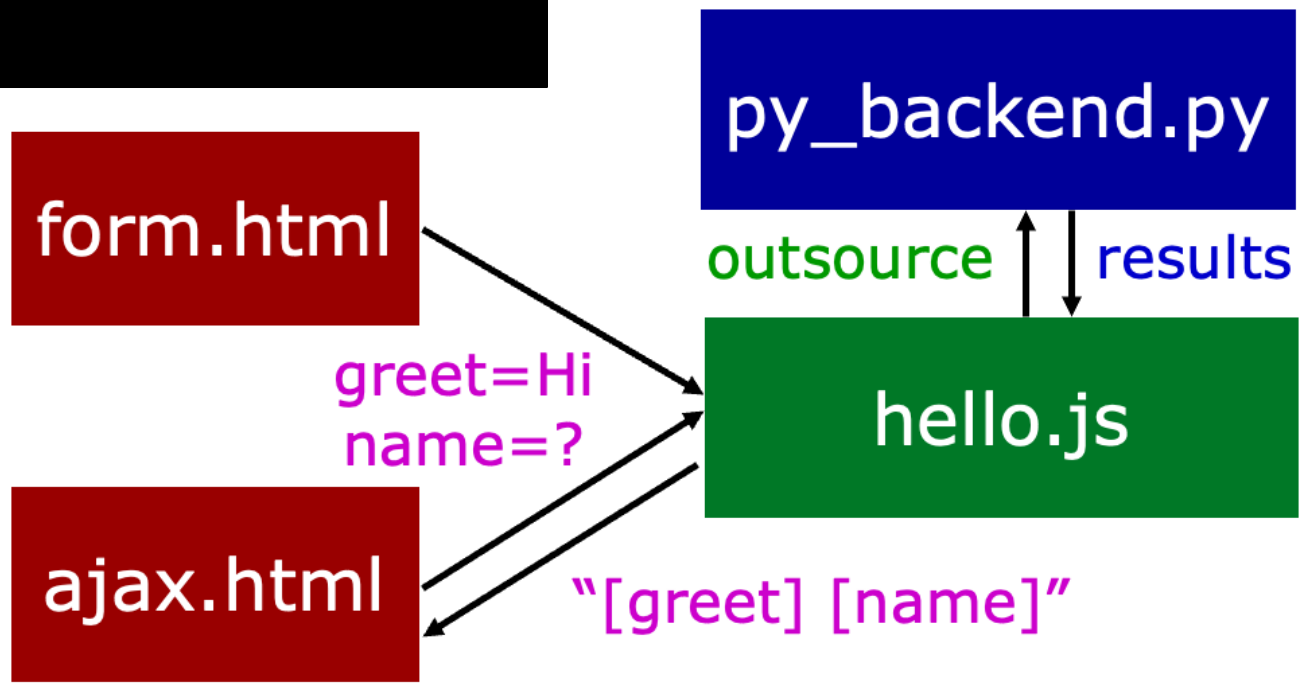


Requests & Responses (2/2)

```
<form name="input" action="hello.js method="get">
  <input type="hidden" name="greet" value="Hi">
  Username: <input type="text" name="name">
  <input type="submit" value="Submit">
</form>
```

```
Import sys
print(str(sys.argv))
```

```
<div id="msg"></div><hr>
Name:<input type="text" id="input">
<script>
  input.addEventListener('keyup',()=>.
  {...});
</script>
```



Goals for today

Learning the Javascript way

Node.js (+Express)

Learning the Python way

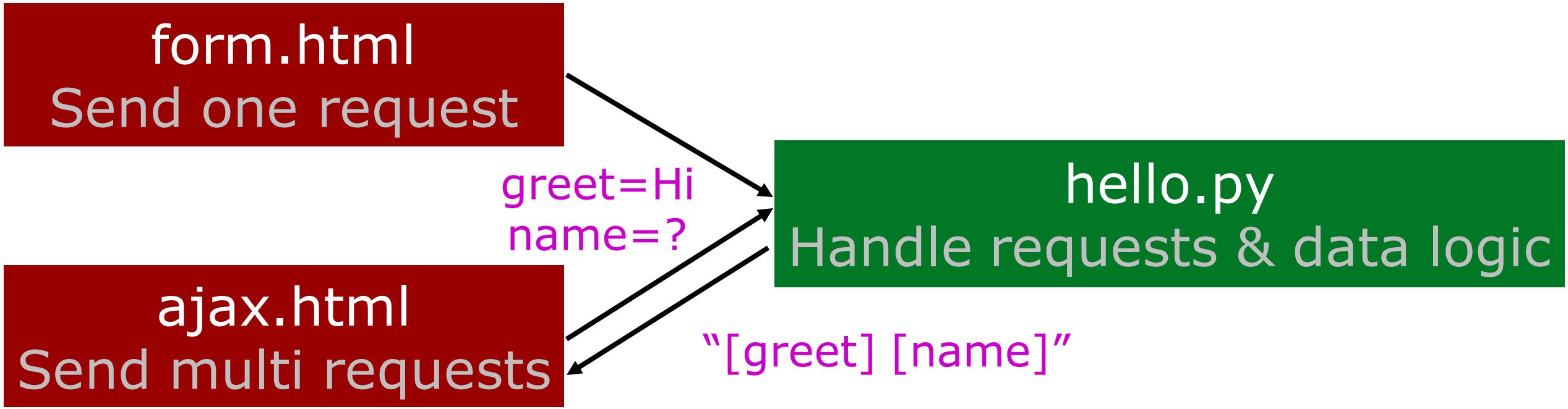
FastAPI

Learning the basic SQL

SQLite

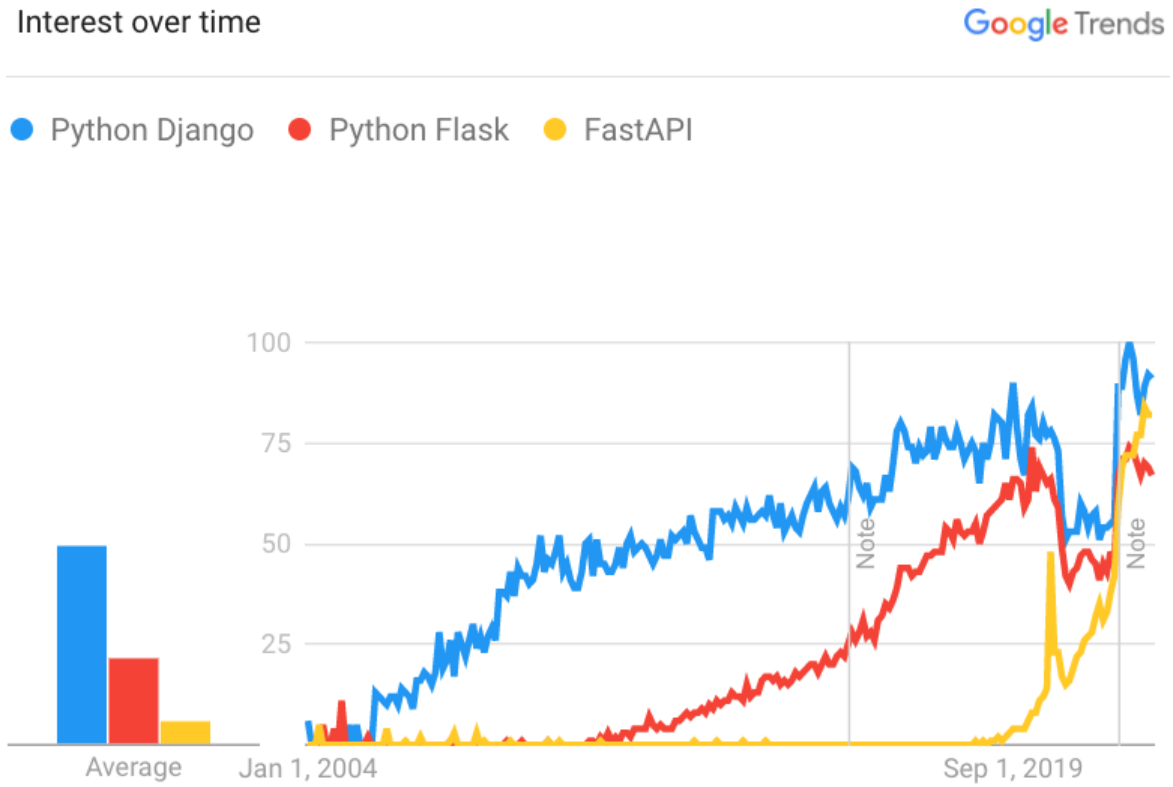
Requests & Responses: Architecture

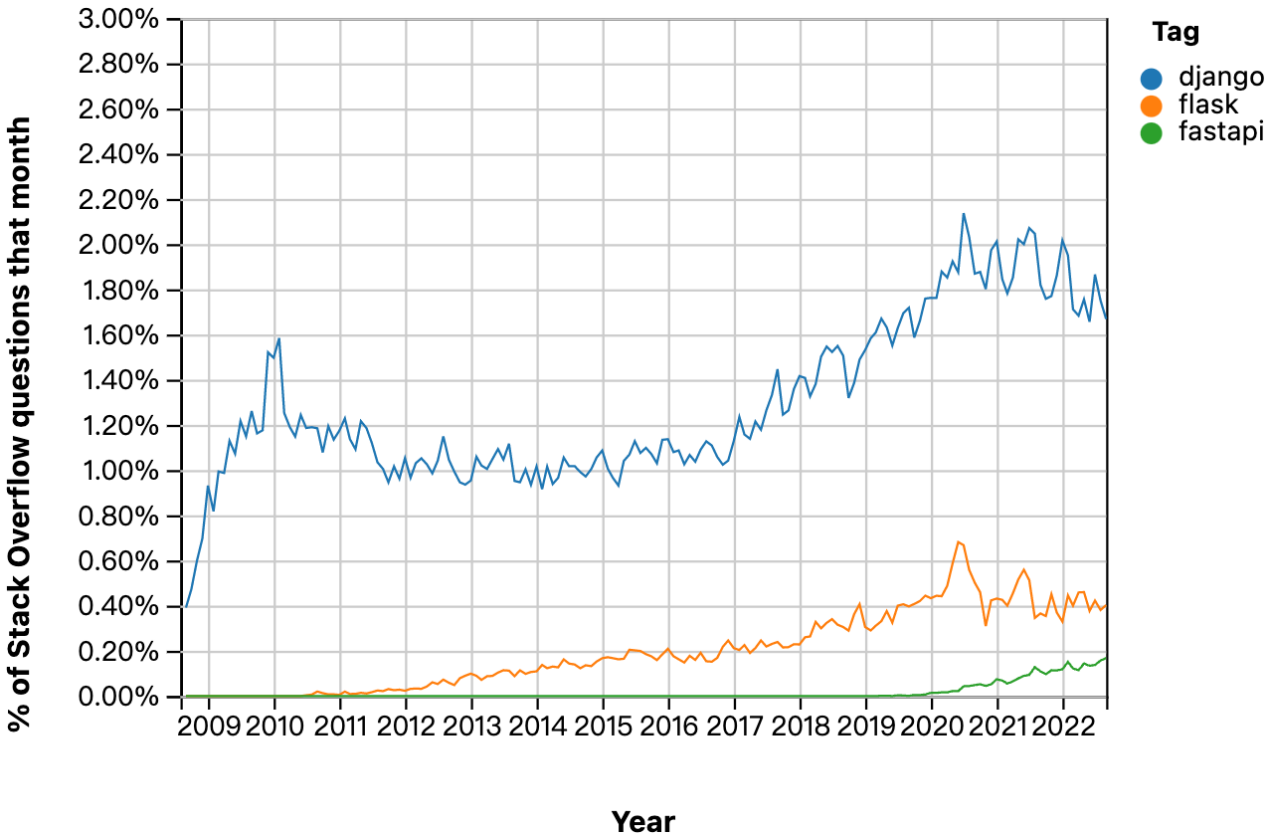
No need to outsource anymore



Django vs. Flask vs. FastAPI (1/3)

Django dominates but it’s quite heavy





Django vs. Flask vs. FastAPI (2/3)

	django	Flask	FastAPI
Performance speed	Normal	Faster than Django	The fastest out there
Async support	**YES** with restricted latency	**NO** needs Asyncio	**YES** native async support
Packages	**Plenty** for robust web apps	**Less than Django** for minimalistic apps	**The least of all** for building web apps faster
Popularity	The most popular	The second popular	Relatively new
Learning	Hard to learn	Easier to learn	The easiest to learn

Django vs. Flask vs. FastAPI (3/3)

Django is best for large-scale websites

The ones complex in features

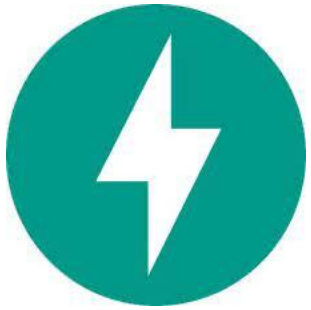
Flask is best for microservices

The ones with simple functionalities

FastAPI is best for pure API backends

Even the ones with session controls

FastAPI

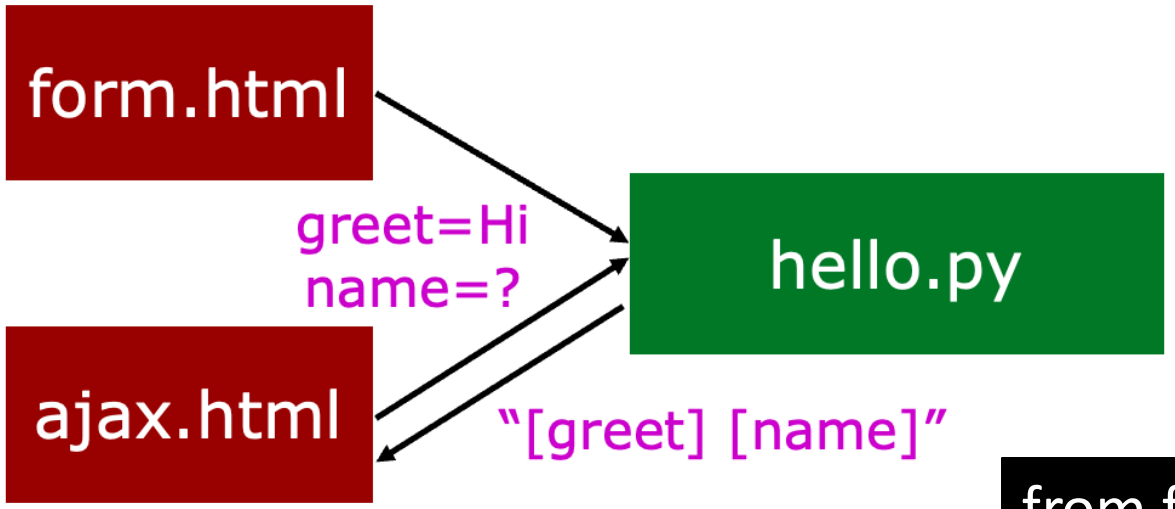


```
bash

$ pip install fastapi

100%
```

```
bash

$ pip install "uvicorn[standard]"

100%
```

```
bash

$ uvicorn main:app --reload

INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [28720]
INFO:     Started server process [28722]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Requests & Responses: Codes




```
from fastapi import FastAPI
from fastapi.responses import PlainTextResponse

@app.get("/", response_class=PlainTextResponse)
def read_root(greet:str="Hello", name:str="Tren"):
  return(f"{greet} {name}")

@app.get("/{greet}/{name")
def read_all(greet:str="Aloha", name:str="Tren"):
  return {"greet": greet, "name": name}
```

Get JSON & JSONP in Frontends

```
fetch('http://localhost:8000/hi/tren')
.then(r => r.json()) //Simplified arrow function
.then((r) => {  //Arrow function
      alert(`${r.greet} ${r.name}`);
}).catch(err => console.log(err)) //Simplified arrow function
```


```
<div id="outputDiv"></div>

<script>
function JavaScriptFetch() {
 let q='cat'; var s = document.createElement('script');
 s.src = 'https://api.flickr.com/services/feeds/photos_public.gne?format=json&tags='+ q
 document.querySelector('head').appendChild(s);
}

function jsonFlickrFeed(data){…}
</script>
```

Goals for today

Learning the Javascript way

Node.js (+Express)

Learning the Python way

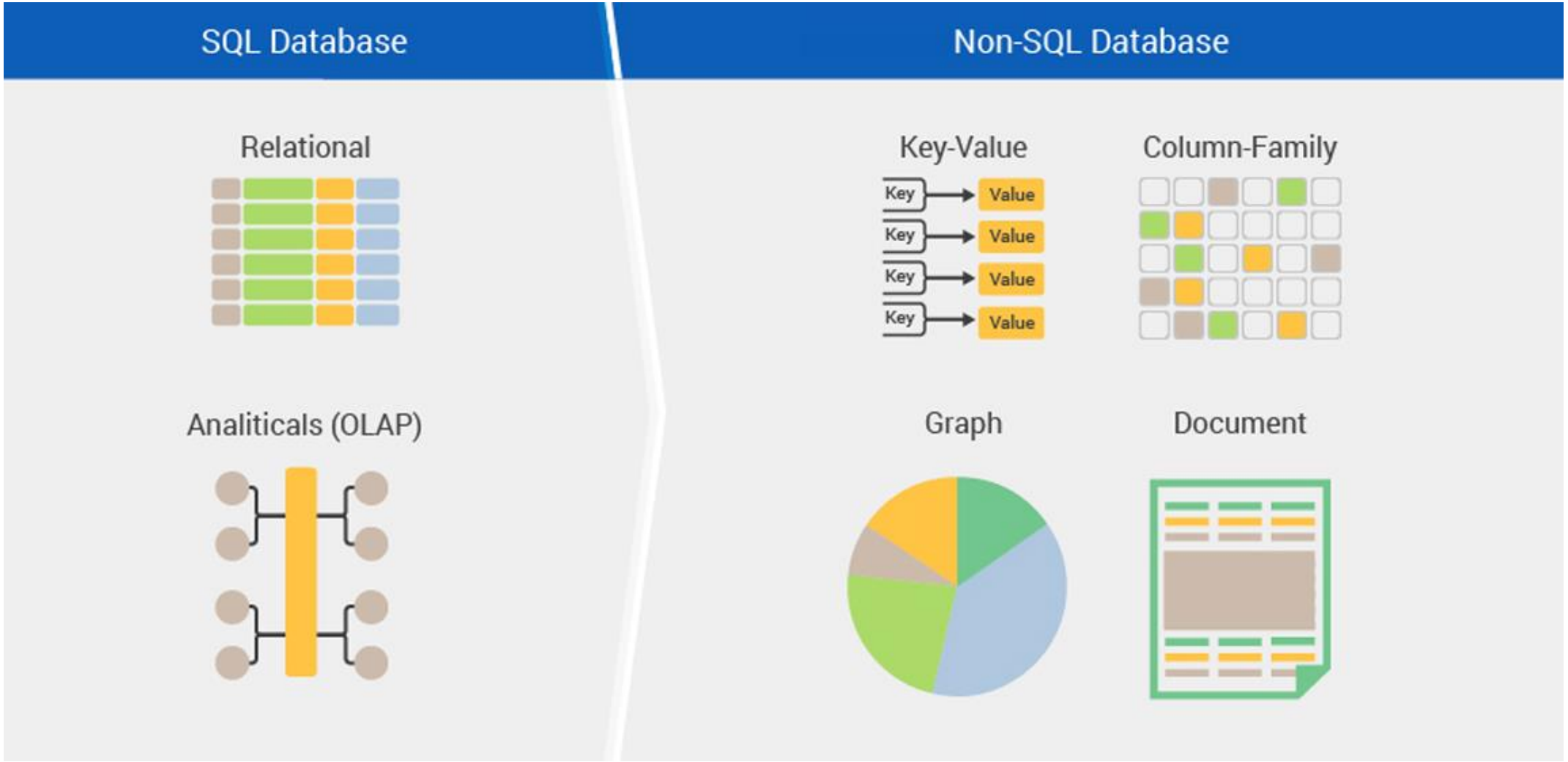
FastAPI

Learning the basic SQL

SQLite

SQL vs. NoSQL (1/3)

NoSQL better handles unstructured data (e.g., Facebook profile)



SQL vs. NoSQL (2/3)

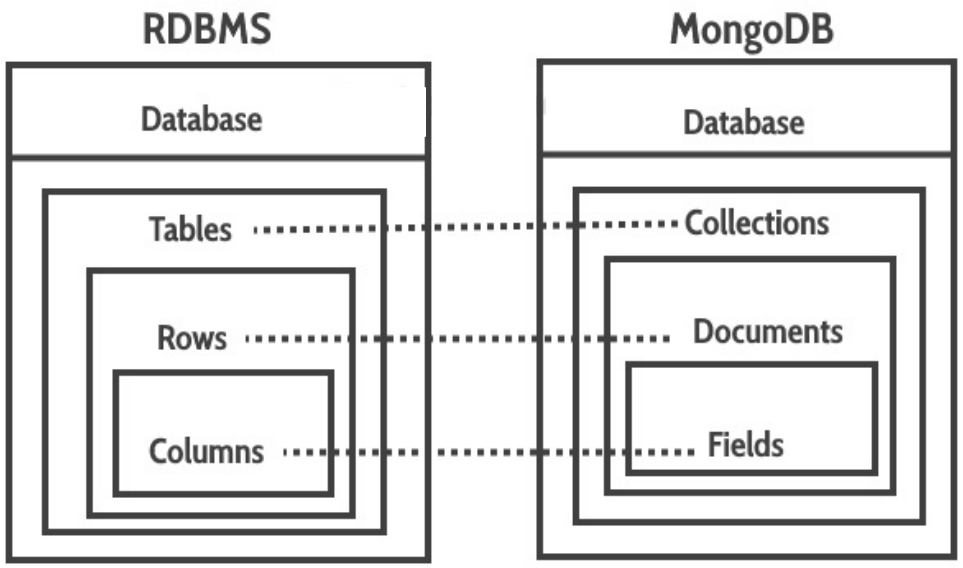
MongoDB is the most popular NoSQL

397 systems in ranking, October 2022

Rank			DBMS	Database Model	Score		
Oct 2022	Sep 2022	Oct 2021			Oct 2022	Sep 2022	Oct 2021
1.	1.	1.	Oracle	Relational, Multi-model	1236.37	-1.88	-33.98
2.	2.	2.	MySQL	Relational, Multi-model	1205.38	-7.09	-14.39
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	924.68	-1.62	-45.93
4.	4.	4.	PostgreSQL	Relational, Multi-model	622.72	+2.26	+35.75
5.	5.	5.	MongoDB	Document, Multi-model	486.23	-3.40	-7.32
6.	6.	6.	Redis	Key-value, Multi-model	183.38	+1.91	+12.03
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	151.07	-0.37	-7.19
8.	8.	↓ 7.	IBM Db2	Relational, Multi-model	149.66	-1.73	-16.30
9.	9.	↑ 11.	Microsoft Access	Relational	138.17	-1.87	+21.79
10.	10.	↓ 9.	SQLite	Relational	137.80	-1.02	+8.43
11.	11.	↓ 10.	Cassandra	Wide column	117.95	-1.17	-1.33
12.	12.	12.	MariaDB	Relational, Multi-model	109.31	-0.85	+6.71
13.	13.	↑ 18.	Snowflake	Relational	106.72	+3.22	+48.46
14.	14.	↓ 13.	Splunk	Search engine	94.66	+0.60	+4.04
15.	15.	↑ 16.	Amazon DynamoDB	Multi-model	88.35	+0.93	+11.80
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-model	84.96	+0.54	+5.24
17.	17.	↓ 14.	Hive	Relational	80.60	+2.17	-4.14
18.	18.	↓ 17.	Teradata	Relational, Multi-model	66.07	-0.51	-3.76
19.	19.	19.	Neo4j	Graph	58.68	-0.79	+0.81
20.	20.		Databricks	Multi-model	57.61	+1.99	

SQL vs. NoSQL (3/3)

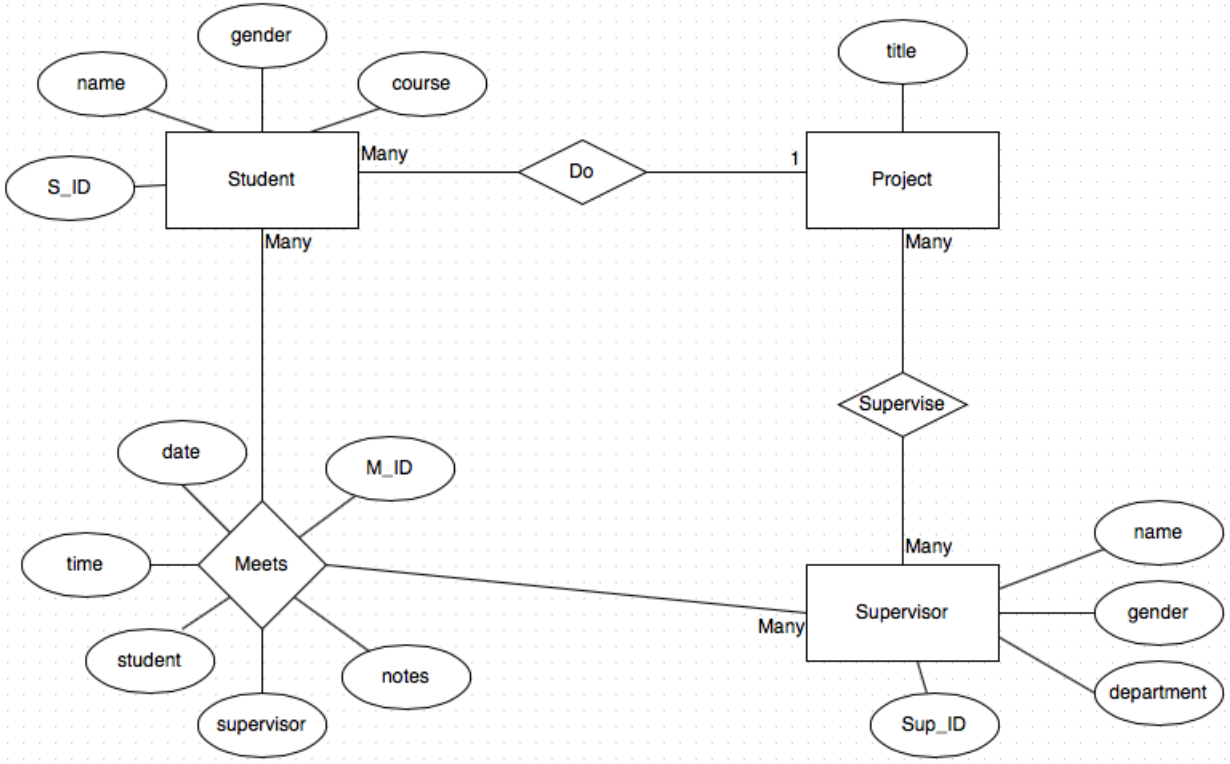
MongoDB uses SQL + Javascript concepts/syntaxes

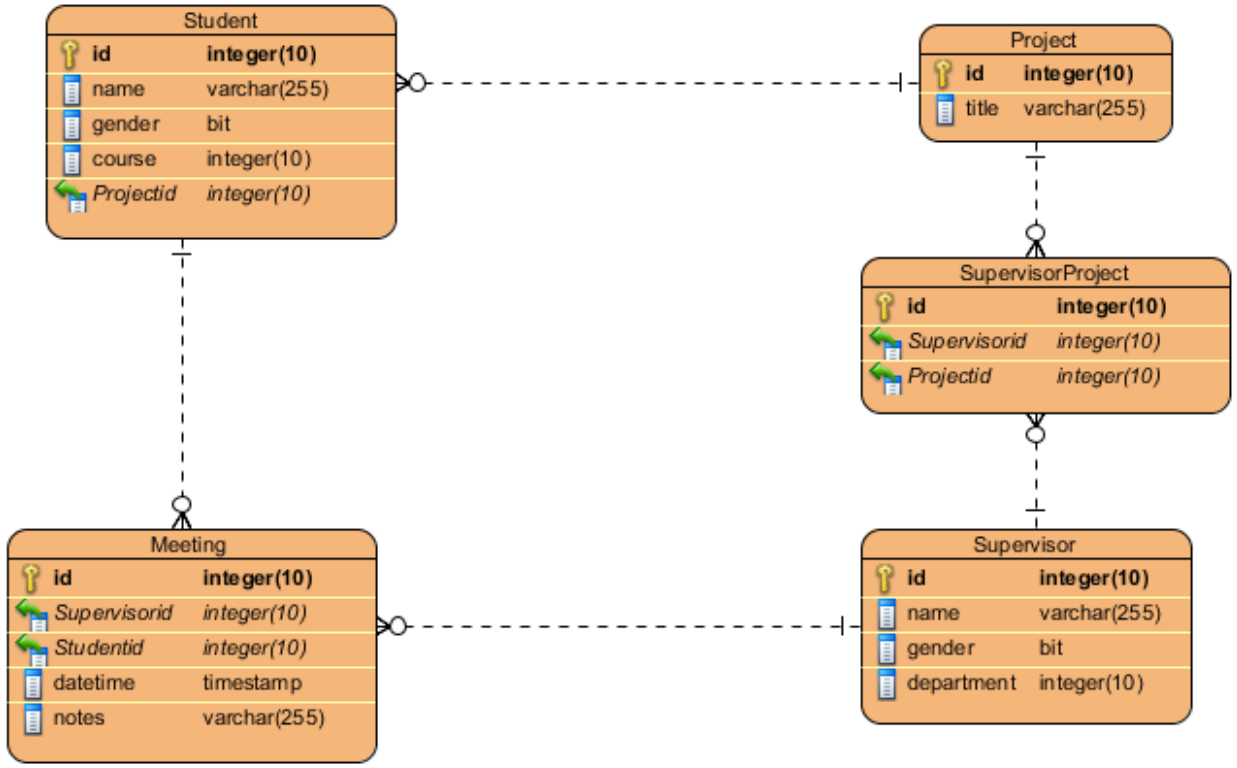



```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });
});
```

Entity-Relationship Diagram (ERD)

Helps to design data tables in SQL





SQL Syntax (1/3)

SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

SQL Syntax (2/3)

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING TABLES

```
CREATE TABLE t (
    id INT PRIMARY KEY,
    name VARCHAR NOT NULL,
    price INT DEFAULT 0
);
```
Create a new table with three columns

```
DROP TABLE t;
```
Delete the table from the database

```
ALTER TABLE t ADD column;
```
Add a new column to the table

```
ALTER TABLE t DROP COLUMN c;
```
Drop column c from the table

```
ALTER TABLE t ADD constraint;
```
Add a constraint

```
ALTER TABLE t DROP constraint;
```
Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```
Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2;
```
Rename column c1 to c2

```
TRUNCATE TABLE t;
```
Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
   c1 INT, c2 INT, c3 VARCHAR,
   PRIMARY KEY (c1,c2)
);
```
Set c1 and c2 as a primary key

```
CREATE TABLE t1(
   c1 INT PRIMARY KEY,
   c2 INT,
   FOREIGN KEY (c2) REFERENCES t2(c2)
);
```
Set c2 column as a foreign key

```
CREATE TABLE t(
   c1 INT, c1 INT,
   UNIQUE(c2,c3)
);
```
Make the values in c1 and c2 unique

```
CREATE TABLE t(
  c1 INT, c2 INT,
  CHECK(c1> 0 AND c1 >= c2)
);
```
Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 VARCHAR NOT NULL
);
```
Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
       (value_list), ....;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
    c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

SQL Syntax (3/3)

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement  -- anchor part
UNION [ALL]
select-statement;  -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name;
```
Delete a view

MANAGING INDEXES

```
CREATE INDEX idx_name
ON t(c1,c2);
```
Create an index on c1 and c2 of the table t

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
```
Create a unique index on c3, c4 of the table t

```
DROP INDEX idx_name;
```
Drop an index

SQL AGGREGATE FUNCTIONS

AVG returns the average of a list

COUNT returns the number of elements of a list

SUM returns the total of a list

MAX returns the maximum value in a list

MIN returns the minimum value in a list

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```
Create or modify a trigger

WHEN
- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

EVENT
- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

TRIGGER_TYPE
- **FOR EACH ROW**
- **FOR EACH STATEMENT**

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```
Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```
Delete a specific trigger

SQLite: A file-based SQL

Often used in small applications (e.g., smartphone apps)

Game Over