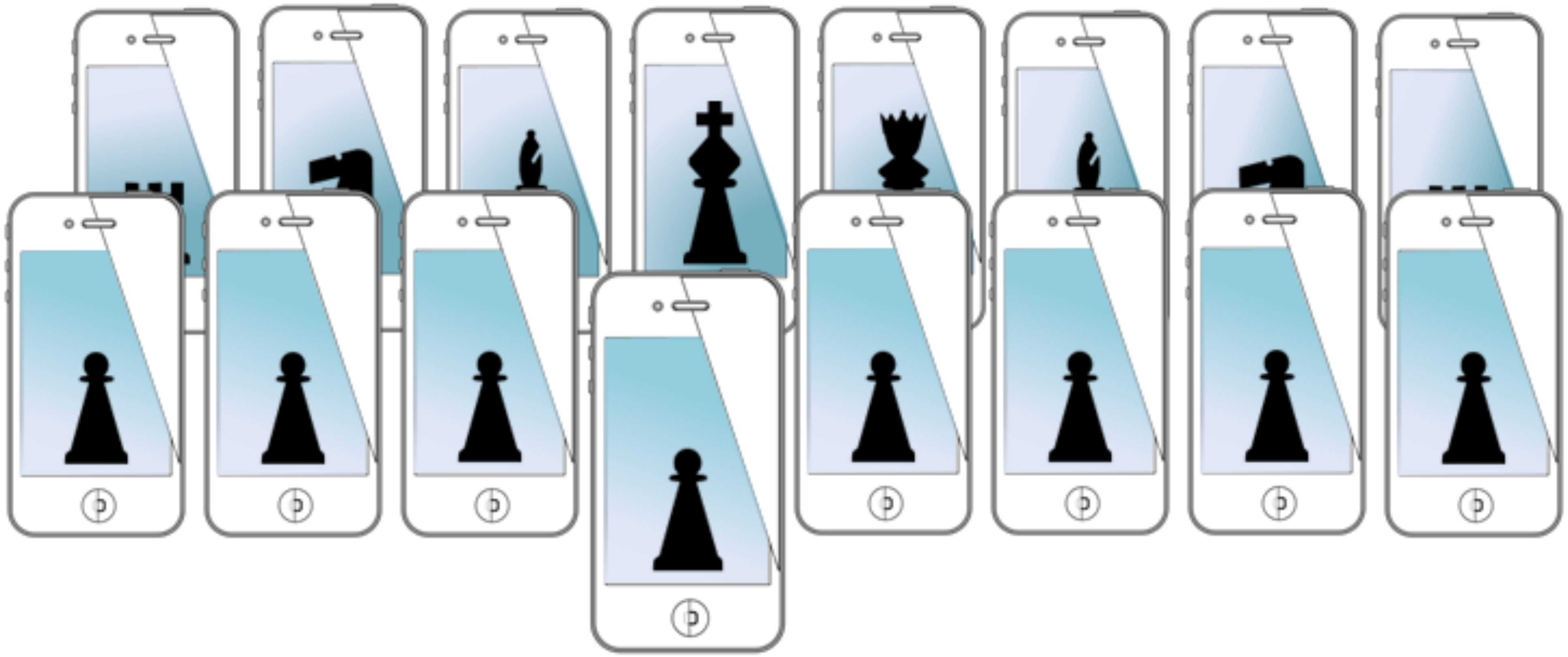
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

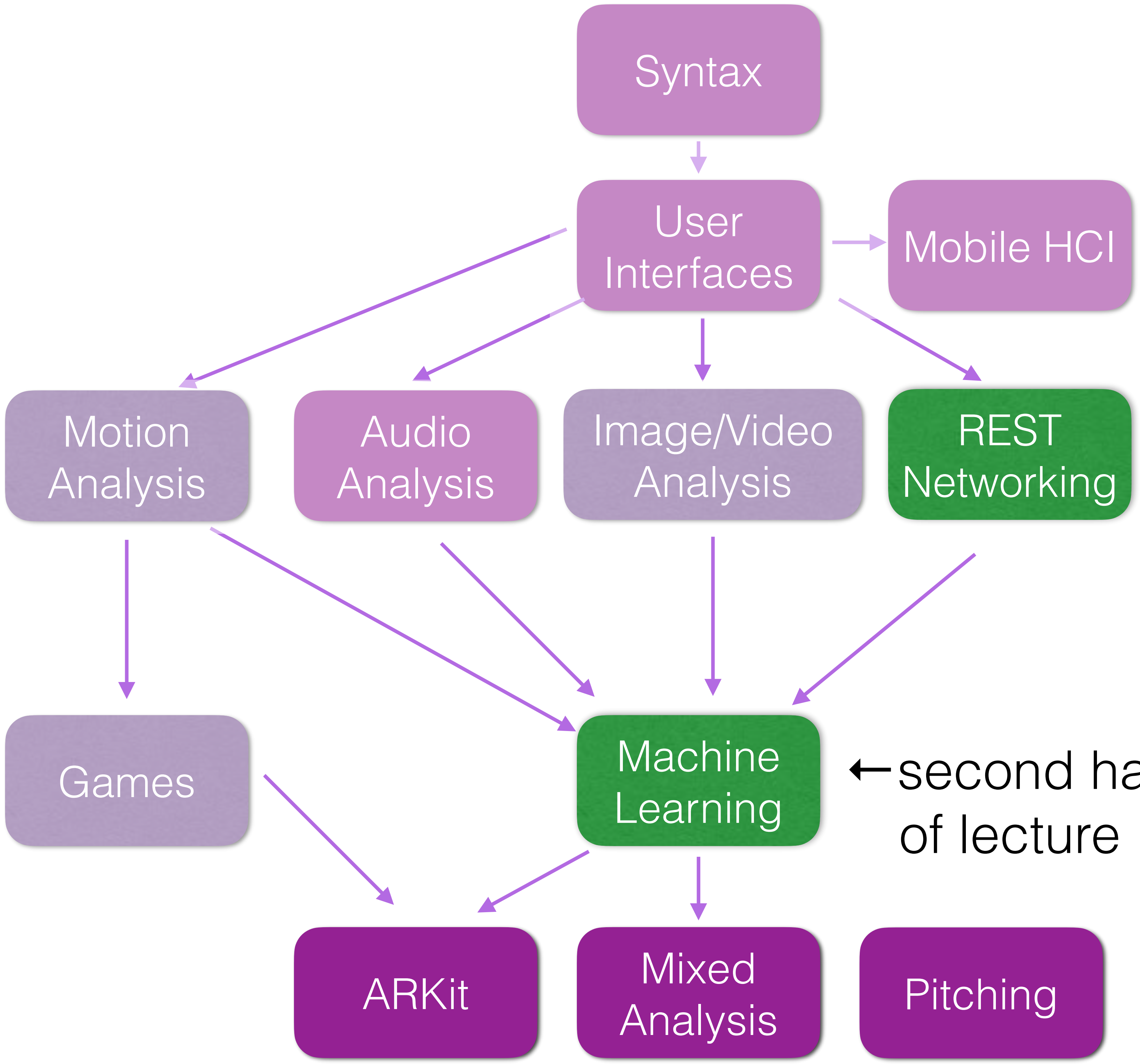
fastapi, pymongo, and http requests

Eric C. Larson, Lyle School of Engineering,
Computer Science, Southern Methodist University

agenda and logistics

- logistics
  - next week, final flipped assignment!
- agenda
  - mongodb, fastapi
  - http requests in iOS

class overview



←first half of lecture

←second half of lecture

←remainder of course

mongodb

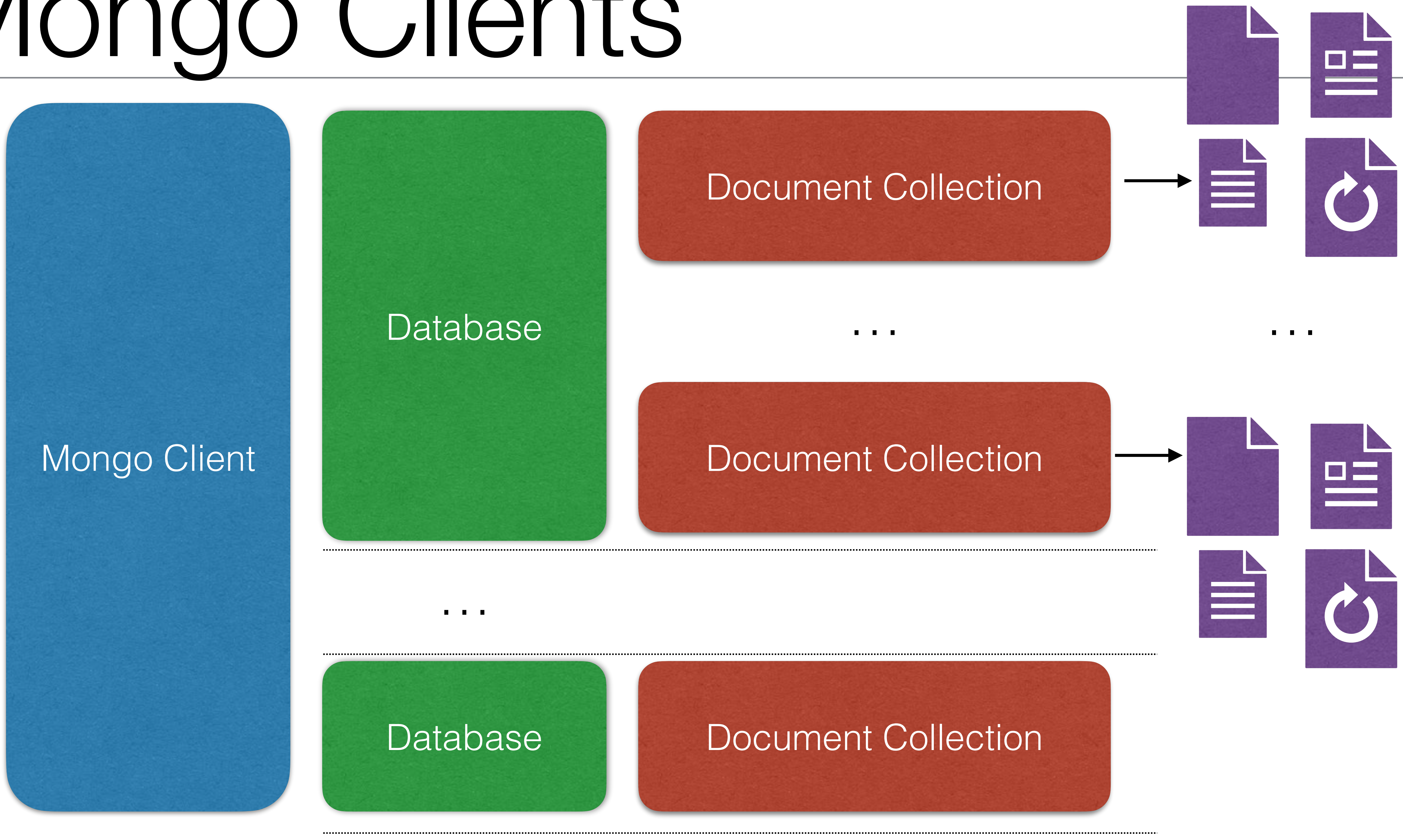
- hu**mongo**us data
- NoSQL database (vs relational database)
  - its a document database
- everything stored as a document
  - more or less json
  - key: value/array
- schema is dynamic
  - the key advantage of NoSQL

mongodb install

Instructions also in Repository
`InstallMongoDB.txt`

- install it
  - `brew tap mongodb/brew`
  - `brew update`
  - `brew install mongodb-community@6.0`
- you can also **run as a service** `(./mongo)`
  - `brew services start mongodb-community@6.0`
  - its running! localhost
  - `brew services stop mongodb-community@6.0`

Mongo Clients

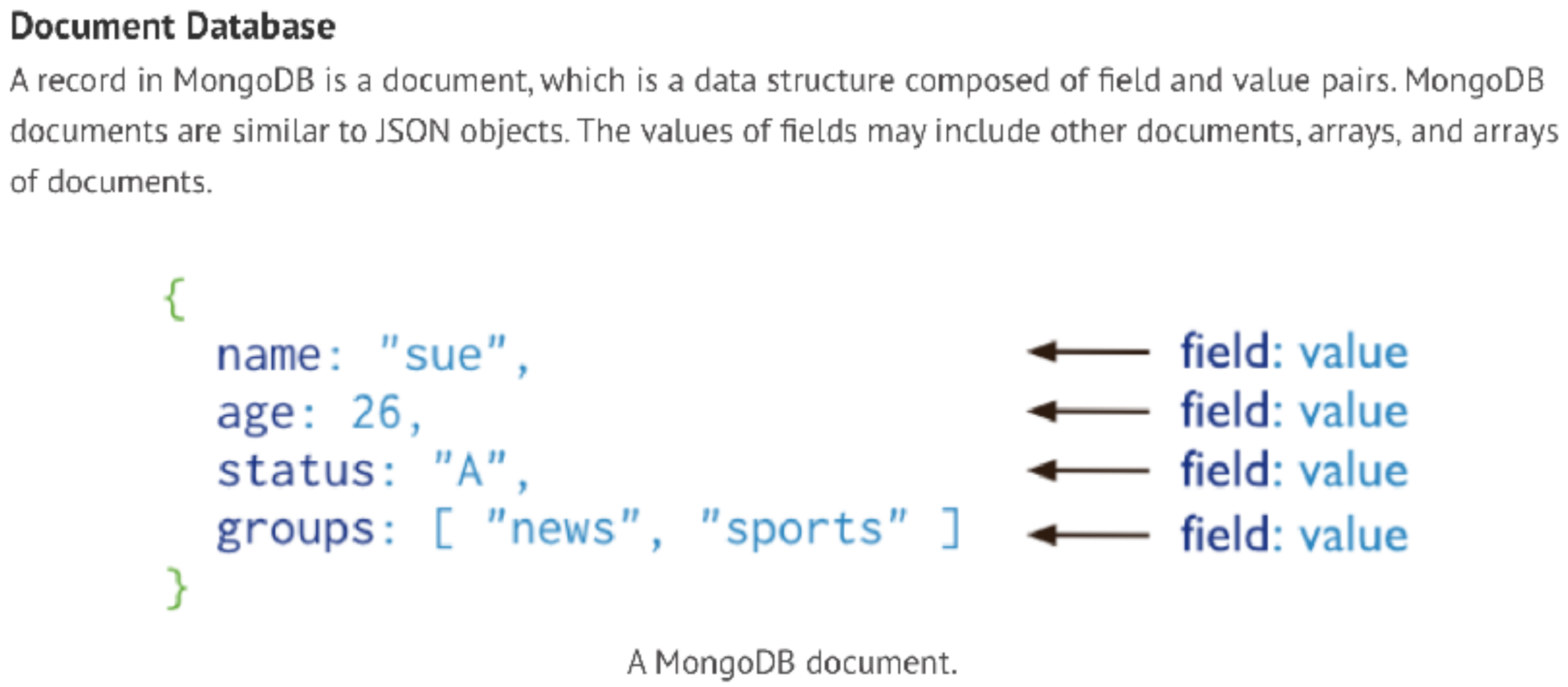


Interface to outside world
listen on large port number

Organizational Structure in MongoDB

mongodb

- a document, as stated by mongodb

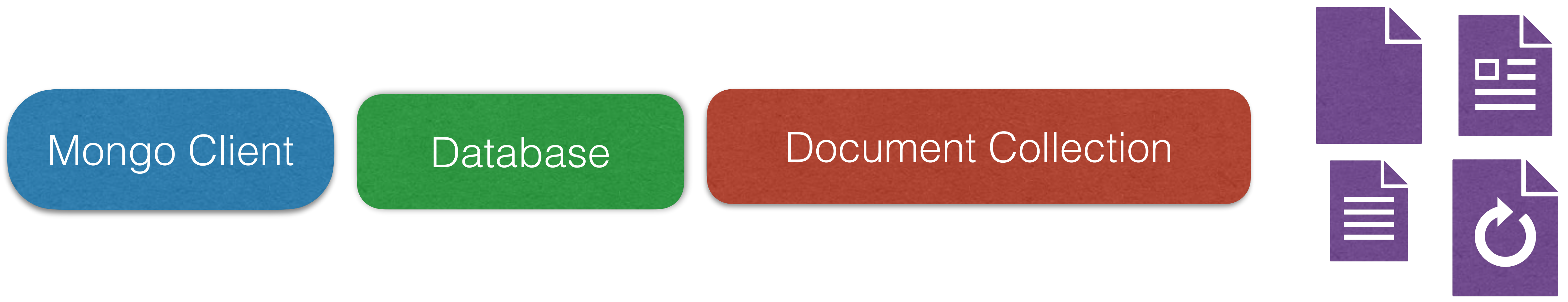


Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

```
{
   name: "sue",                              ←—— field: value
   age: 26,                                  ←—— field: value
   status: "A",                              ←—— field: value
   groups: [ "news", "sports" ]              ←—— field: value
}
```

A MongoDB document.



docs and collections

Database: MSLC_creations

default limit on size of each document: **16MB**

apps_collection

```
{
   app: "mongoApp",
   users: 100005,
}

{
   app: “StepCount",
   users: 45,
   rating: 2.6,
}
. . . .
{
   app: “Trench",
   users: 4050000,
   rating: 5,
}
```

teams_collection

```
{
   team: "mongo",
   members: [ "Eric", “Ringo”, “Paul” ],
   numApps: 21,
   website: “teammongo.org",
}
{
   team: “ran off",
   members: [ "John", “Yoko” ],
   website: “flewthecoop.org",
}
. . . .
{
   team: “21 Pilots",
   members: [ "Tyler", “Nick” ],
   numApps: 4,
   website: “RollingStone.com”,
}
```

Mongo Client

Database

Document Collection

pymongo

- python wrapper for using mongo db

```
client  = MongoClient() # localhost, default port
db = client.some_database # access database
```

create this database, if it does not exist

```
collect = client.some_database.some_collection # access a collection
```

Mongo Client   Database   Document Collection

nothing is **created** until the first insert!!!

```
db.collection_names()
[u'system.indexes', u'some_collection']
```

get collections

pymongo (add data)

- insertion

```
    dbid = db.some_collect.insert_one(
               {“key1”:values,"key2":more_values,
                 “coolkey”:with_cool_values}
               )
```

unique key, _id

doc to insert

- update

where ever this key is…

equal to this

```
      db.some_collect.update(  {“thiskey":keyValue},
             {  "$set": {"keyToSet":valueToSet}  },
             upsert=True)
```

set

this key to this value

insert if it does not exist (put/post)

pymongo (get data)

- find one datum in database

```
a = db.some_collect.find_one(sort=[("sortOnThisKey", -1)])
newData = float( a['sortOnThisKey'] );
```

could be list of keys!

access the result

sort with this key

return last element

- iterate through many results

return iterator to loop over

```
f=[];
for doc in db.some_collect.find({"keyIWant":valueOfKeyIWant}):
   doc['key1'] # entire document, is availab
   f.append( str(doc['keyToGrabDataWith']) )
```

each iteration gives one document

- lots of advanced queries are possible

https://api.mongodb.org/python/current/

teams example

```
>>> from pymongo import MongoClient
>>> client = MongoClient()

>>> db = client.some_database
>>> collect1 = db.some_collection
>>> collect1.insert_one({“team”:"TeamFit","members":["Matt","Mark","Rita","Gavin"]})
ObjectId('53396a80291ebb9a796a8af1')

>>> db.collection_names()
[u'system.indexes', u'some_collection']

>>> db.some_collection.find_one()
{u'_id': ObjectId('53396a80291ebb9a796a8af1'), u'members': [u'Matt', u'Mark', u'Rita', u'Gavin'],
u'team': u'TeamFit'}

>>> collect1.insert_one({“team":"Underscore","members":["Carly","Lauryn","Cameron"]})
ObjectId('53396c80291ebb9a796a8af2')

>>> db.some_collection.find_one()
{u'_id': ObjectId('53396a80291ebb9a796a8af1'), u'members': [u'Matt', u'Mark', u'Rita', u'Gavin'],
u'team': u'TeamFit'}

>>> db.some_collection.find_one({"team":"Underscore"})
{u'_id': ObjectId('53396c80291ebb9a796a8af2'), u'members': [u'Carly', u'Lauryn', u'Cameron'],
u'team': u'Underscore'}
```

bulk operations

```
from pymongo import MongoClient

client = MongoClient()
db=client.some_database
collect1 = db.some_collection

insert_list = [{"team":"MCVW","members":["Matt","Rowdy","Jason"]},
               {“team":"CHC", "members":["Hunter","Chelsea","Conner"]}]

obj_ids=collect1.insert_many(insert_list)
```

anything iterable

```
for document in collect1.find({"members":"Matt"}):
    print(document)
```

```
{u'_id': ObjectId('53396a80291ebb9a796a8af1'), u'members': [u'Matt', u'Mark', u'Rita', u'Gavin'], u'team': u'TeamFit'}
{u'_id': ObjectId('53397331291ebb9afdd3cd2f'), u'members': [u'Matt', u'Rowdy', u'Jason'], u'team': u'MCVW'}
```

```
document = collect1.find_one({"members":"Matt","team":"MCVW"})
print (document)
```

```
{u'_id': ObjectId('53397331291ebb9afdd3cd2f'), u'members': [u'Matt', u'Rowdy', u'Jason'], u'team': u'MCVW'}
```

async mongodb (+ tornado)

- we will use pymongo and fastapi
  - mongodb runs localhost, fastapi mediates access
  - and asynchronous calls (decorators or async/await)

```
# Motor imports
from bson import ObjectId
import motor.motor_asyncio

app.mongo_client = motor.motor_asyncio.AsyncIOMotorClient()

# remember the db collection, named “database” and “mycollection"
app.collection = app.mongo_client.mydatabase.get_collection(“mycollection”)
```

```
new_label = await app.collection.insert_one(
         datapoint
     )
```

https://motor.readthedocs.io/en/stable/tutorial-tornado.html

what we have

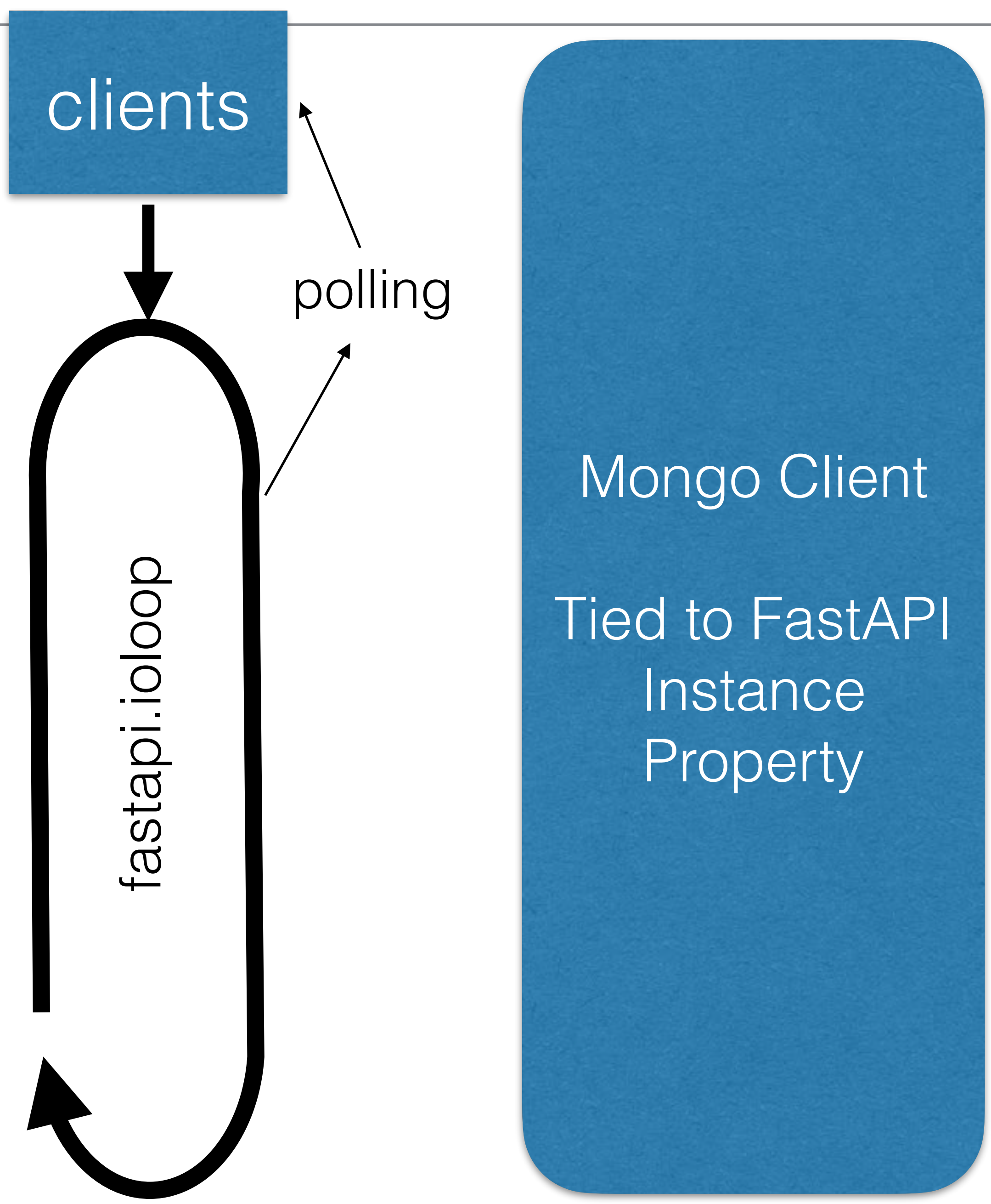
- good working knowledge of python syntax
- an asynchronous database for storing info

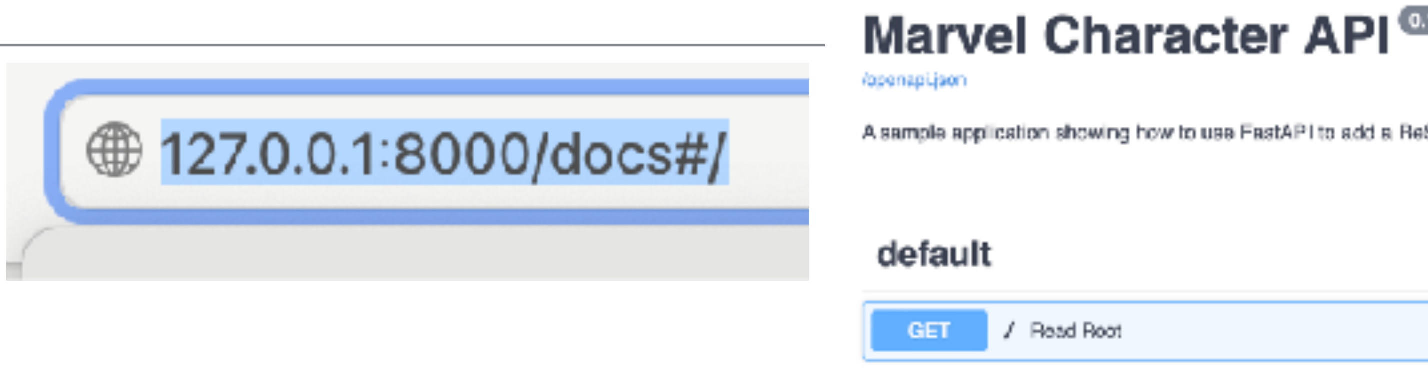
what we need

- we just need a server interface!

```
fastapi run fastapi_example.py
```

```
1. brew services start mongodb-community@6.0
```




127.0.0.1:8000/docs#/

Marvel Character API

/openapi.json

A sample application showing how to use FastAPI to add a Re

default

GET / Read Root

3. test local queries
in a browser with docs
(if you want)

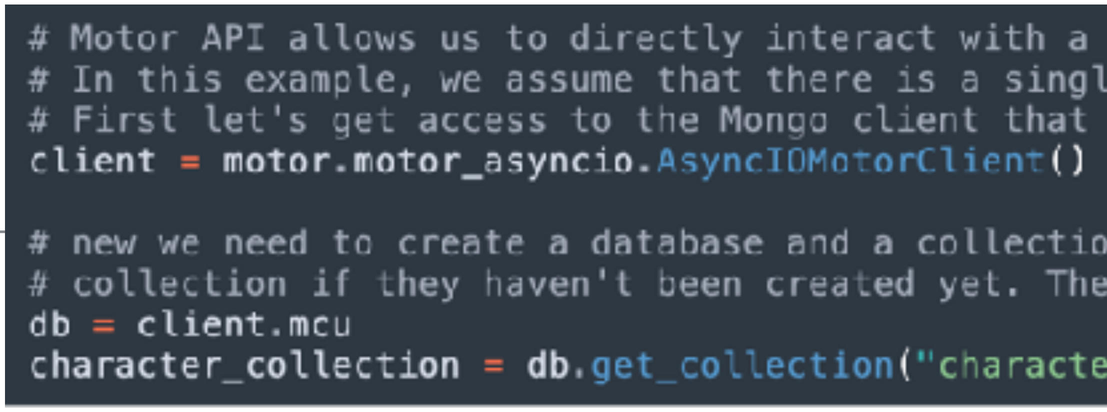
```
ifconfig | grep "inet "
```

4. make queries
from external sources
via external facing IP

2. run
fastapi.app()

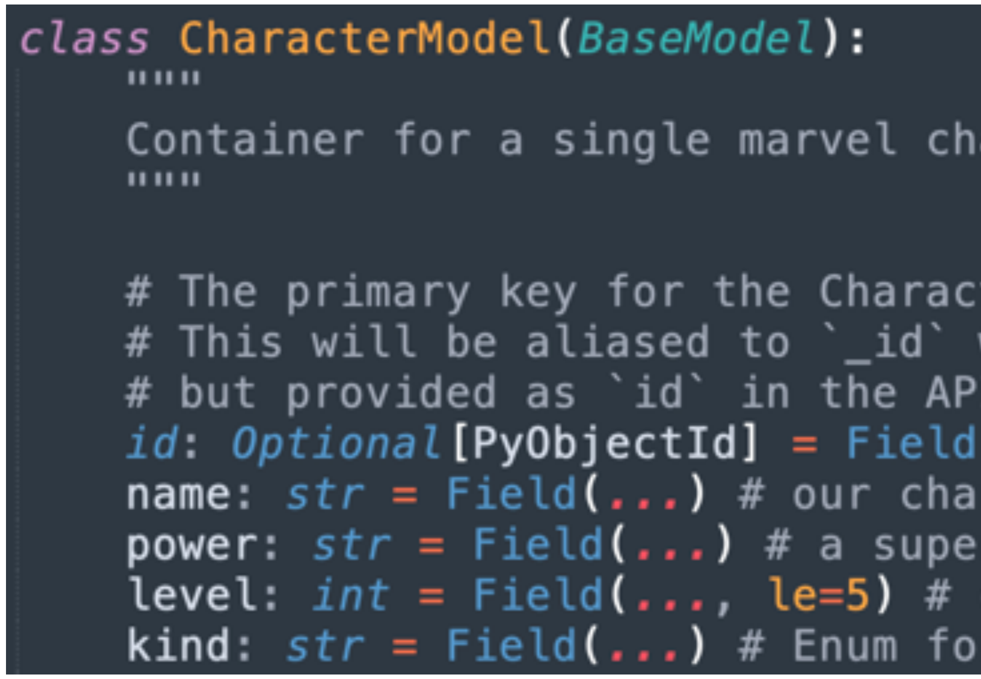
```
fastapi run fastapi_example.py
```

fastapi workflow

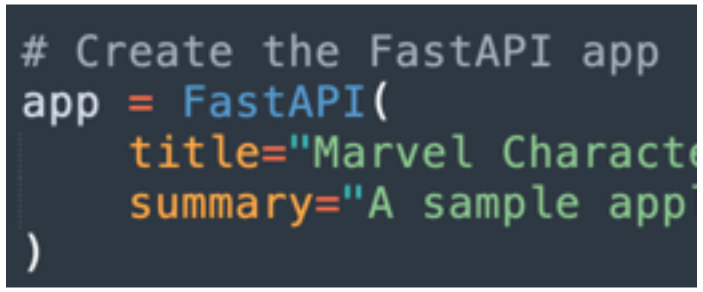


```
# Motor API allows us to directly interact with a
# In this example, we assume that there is a singl
# First let's get access to the Mongo client that
client = motor.motor_asyncio.AsyncIOMotorClient()

# new we need to create a database and a collectio
# collection if they haven't been created yet. The
db = client.mcu
character_collection = db.get_collection("characte
```

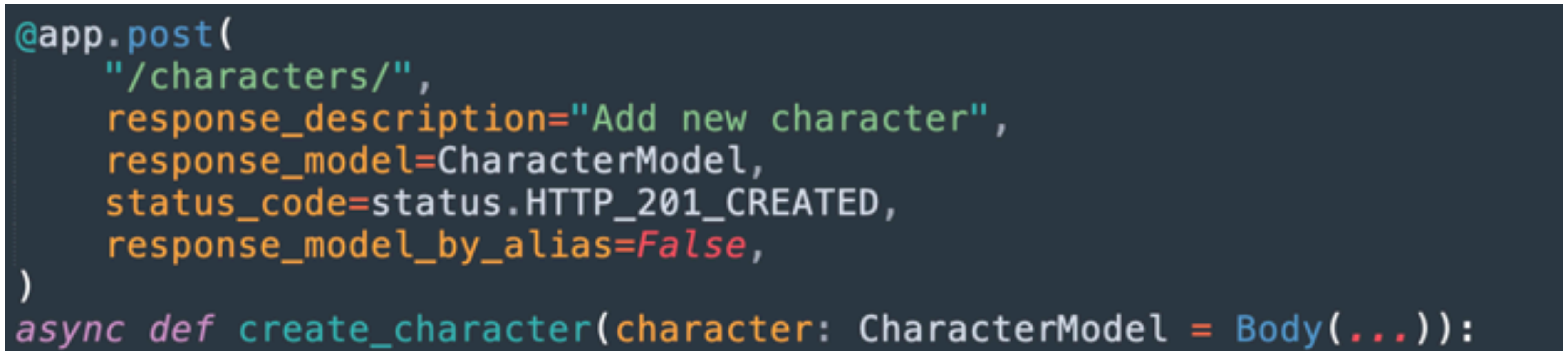
- setup “app”
- setup database client and store in app (or global…)
- setup pydantic data stores (and sync id with database format)
- setup get, post, put, delete requests
    - url for particular request
    - what response model to use (from pydantic)
    - status code to send
    - input format to function



```
class CharacterModel(BaseModel):
    """
    Container for a single marvel ch
    """

    # The primary key for the Charac
    # This will be aliased to `_id`
    # but provided as `id` in the AP
    id: Optional[PyObjectId] = Field
    name: str = Field(...) # our cha
    power: str = Field(...) # a supe
    level: int = Field(..., le=5) #
    kind: str = Field(...) # Enum fo
```

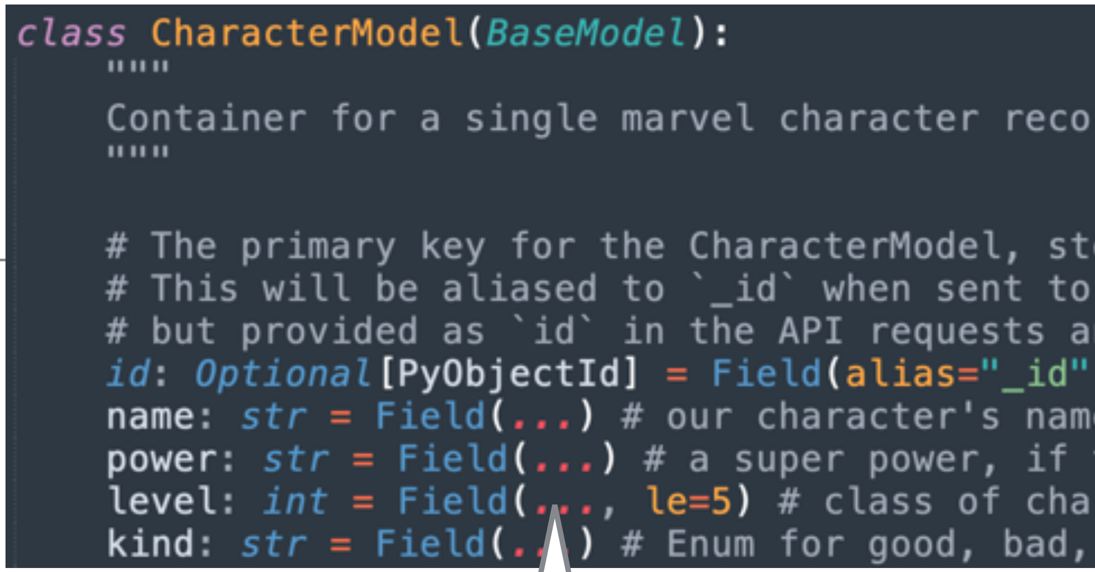


```
# Create the FastAPI app
app = FastAPI(
    title="Marvel Characte
    summary="A sample app
)
```

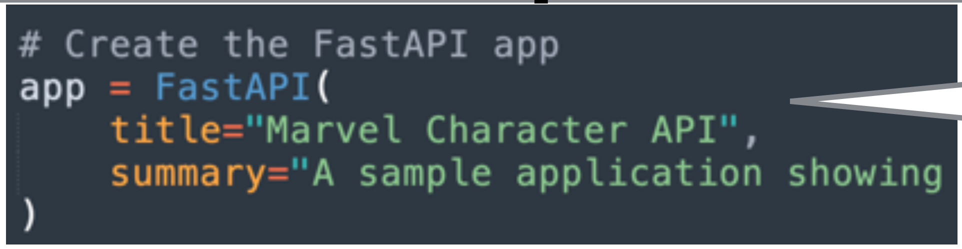


```
@app.post(
    "/characters/",
    response_description="Add new character",
    response_model=CharacterModel,
    status_code=status.HTTP_201_CREATED,
    response_model_by_alias=False,
)
async def create_character(character: CharacterModel = Body(...)):
```

fastapi workflow

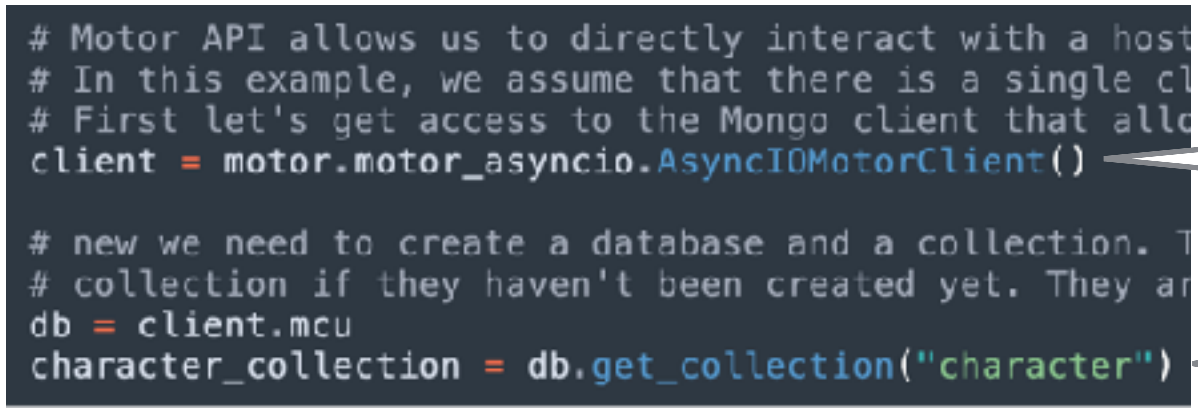


```
class CharacterModel(BaseModel):
    """
    Container for a single marvel character reco
    """

    # The primary key for the CharacterModel, st
    # This will be aliased to `_id` when sent to
    # but provided as `id` in the API requests a
    id: Optional[PyObjectId] = Field(alias="_id"
    name: str = Field(...) # our character's nam
    power: str = Field(...) # a super power, if
    level: int = Field(..., le=5) # class of cha
    kind: str = Field(...) # Enum for good, bad,
```



```
# Create the FastAPI app
app = FastAPI(
    title="Marvel Character API",
    summary="A sample application showing
)
```

fastapi object

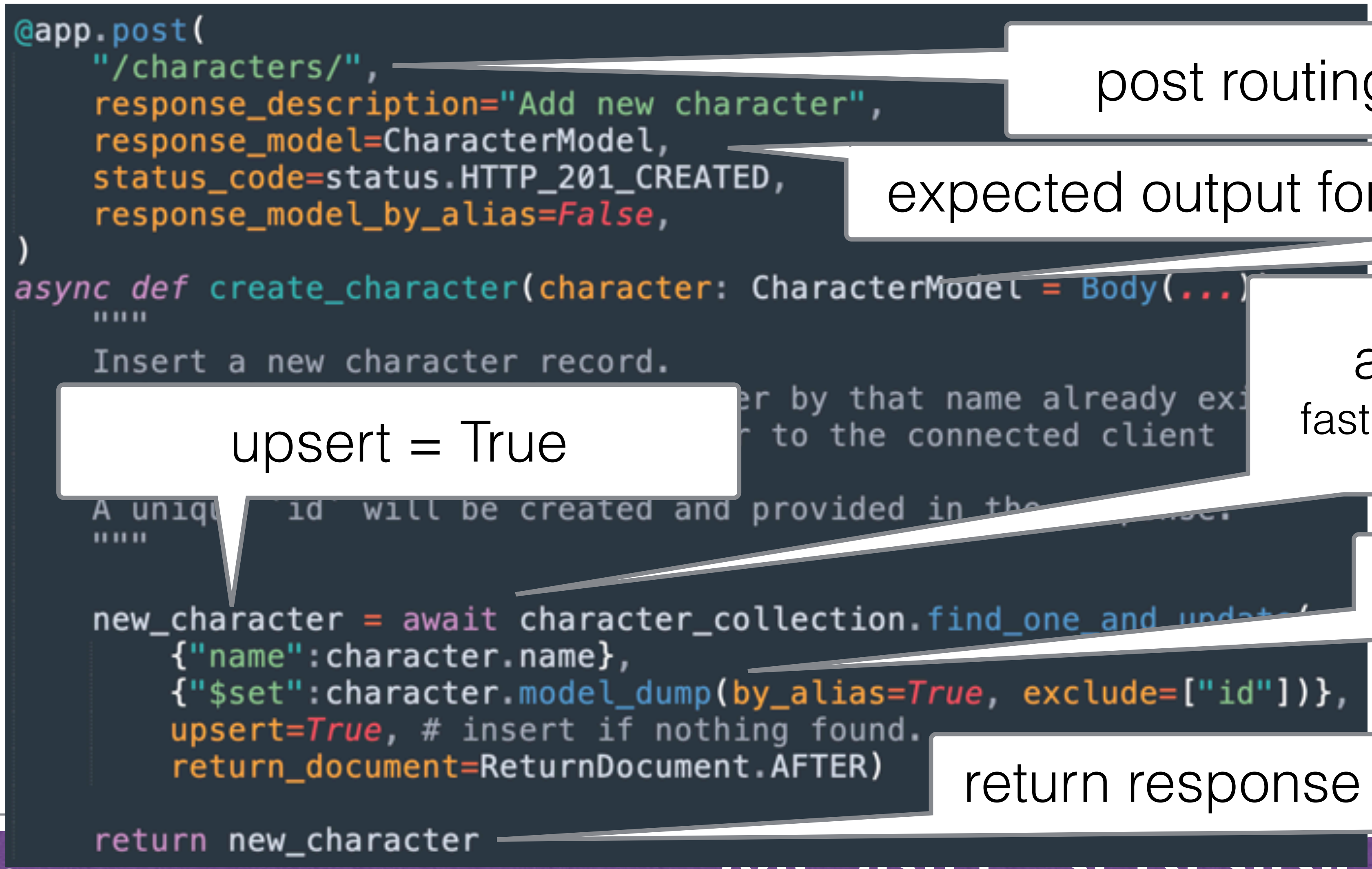


```
# Motor API allows us to directly interact with a host
# In this example, we assume that there is a single cl
# First let's get access to the Mongo client that allc
client = motor.motor_asyncio.AsyncIOMotorClient()

# new we need to create a database and a collection. T
# collection if they haven't been created yet. They ar
db = client.mcu
character_collection = db.get_collection("character")
```

async MongoDB

data format

document collection



```
@app.post(
    "/characters/",
    response_description="Add new character",
    response_model=CharacterModel,
    status_code=status.HTTP_201_CREATED,
    response_model_by_alias=False,
)
async def create_character(character: CharacterModel = Body(...)):
    """
    Insert a new character record.

                                  er by that name already exi
                                  r to the connected client

    A uniq   `id` will be created and provided in the
    """

    new_character = await character_collection.find_one_and_update(
        {"name":character.name},
        {"$set":character.model_dump(by_alias=True, exclude=["id"])},
        upsert=True, # insert if nothing found.
        return_document=ReturnDocument.AFTER)

    return new_character
```

post routing from "/"

expected output format

expected input format

async find and update
fastapi poll while mongo is running

upsert = True

serialize pydantic as dictionary

return response as json, using pydantic model

motor + fastapi

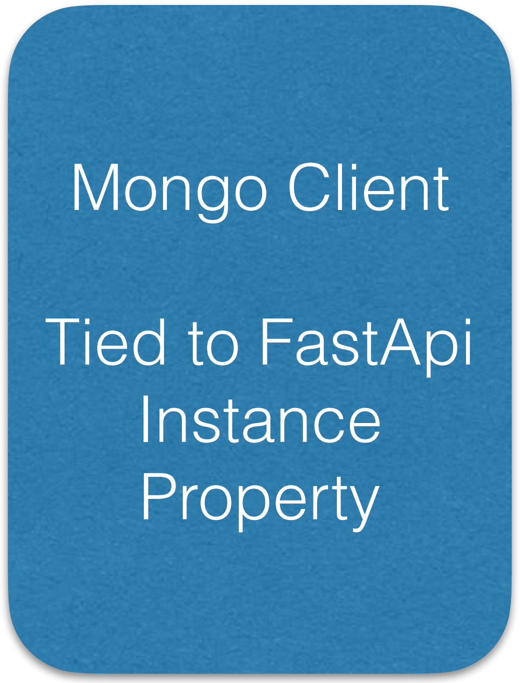
ifconfig | grep "inet "

- demo:
    - store data inside mongodb for different characters

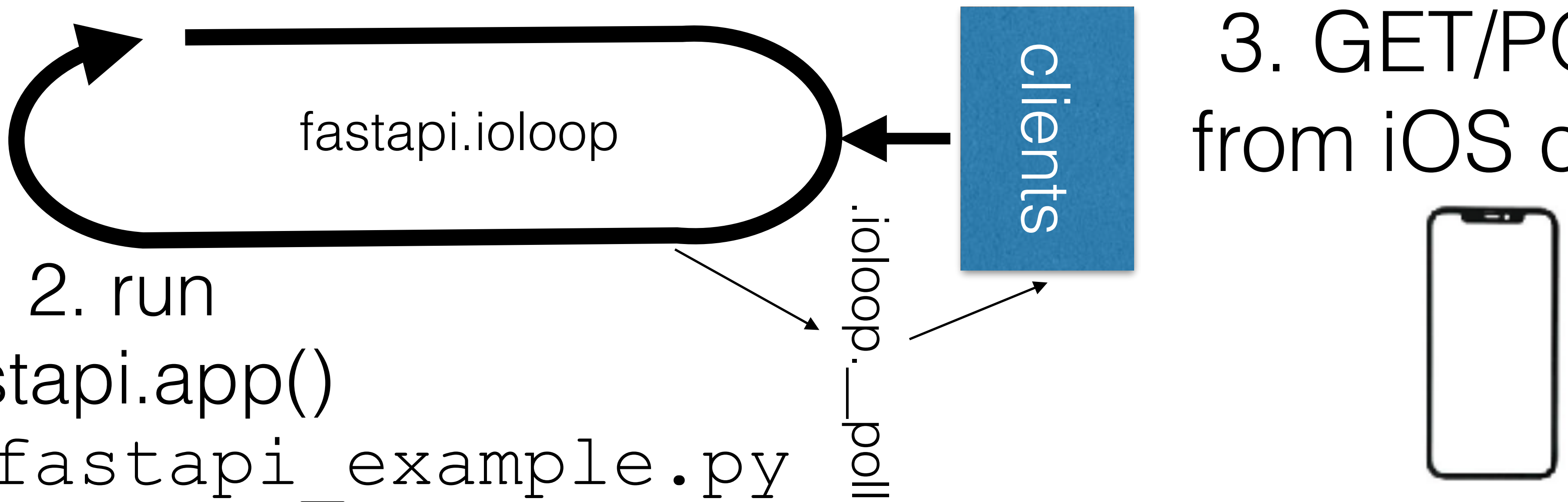
CAL POLY
SAN LUIS OBISPO
Drexel
UNIVERSITY
ingésup
PLYMOUTH UNIVERSITY
RMIT UNIVERSITY
UNIVERSITY OF CALIFORNIA SANTA CRUZ
SMU
Stanford
Technische Universität München
And now its time for a demo

working with your web server

1. `brew services start`



- we want to send data to our hosted server!
  - or any server for that matter
- need to form POST and GET requests from iOS
- we will use URLSession



```
fastapi run fastapi_example.py
```

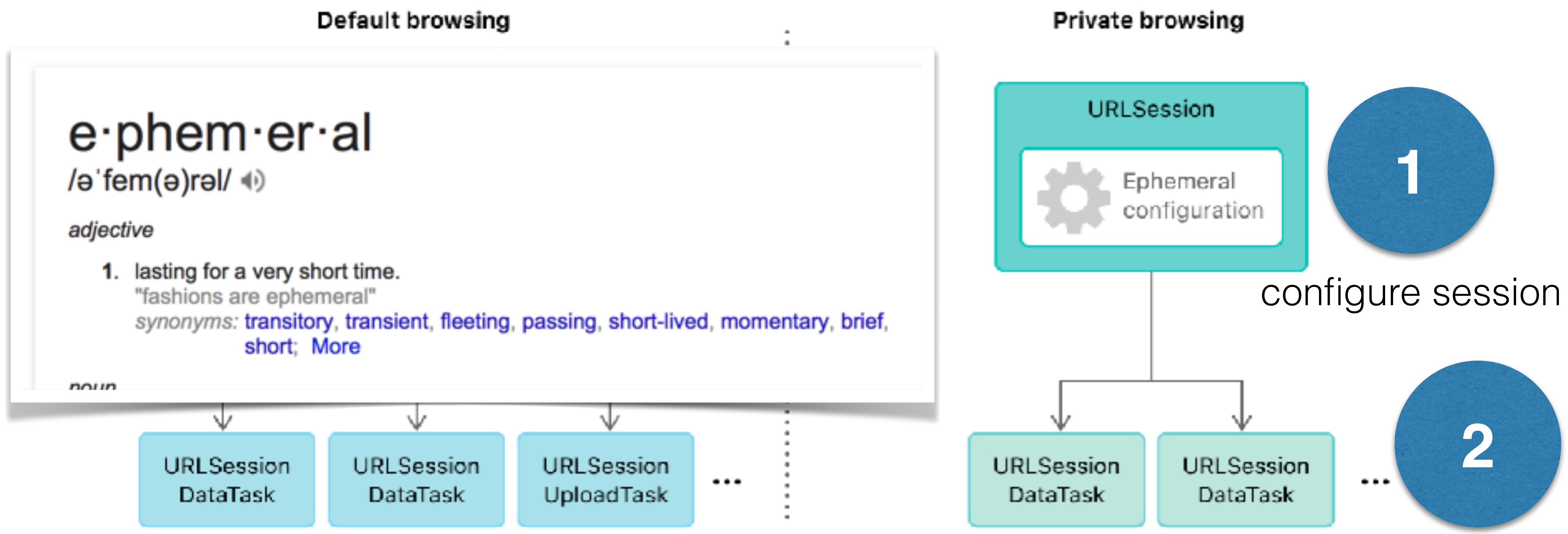
URLSession

- proper way to configure a session with a server
  - previous ways: `NSURLSession, NSURLConnection, sendAsynchronousRequest`
- you may see code for `initWithContentsOfURL`:
  - **never, never, never** use that for networking
- sessions are a huge improvement in iOS
  - and extremely powerful
  - the Stanford course talks about these (check it out)!
  - as promised, we will cover different topics than Stanford course

URLSession -> DataTask

- delegate model
- does authentication if you need it!
- implements pause / resume, tasks

do not cache
no cookies
do not store credentials



e·phem·er·al
/əˈfem(ə)rəl/
adjective
1. lasting for a very short time.
"fashions are ephemeral"
synonyms: transitory, transient, fleeting, passing, short-lived, momentary, brief, short; More
noun

configure session

configure task

configure a session

```
class ViewController: UIViewController, URLSessionDelegate {

    // MARK: Class Properties
    let operationQueue = OperationQueue()
```

delegation

custom queue

```
  //setup NSURLSession (ephemeral)
  let sessionConfig = URLSessionConfiguration.ephemeral

  sessionConfig.timeoutIntervalForRequest = 5.0
  sessionConfig.timeoutIntervalForResource = 8.0
  sessionConfig.httpMaximumConnectionsPerHost = 1

  self.session = URLSession(configuration: sessionConfig,
         delegate: self,
         delegateQueue:self.operationQueue)
```

configure a DataTask

- DataTask objects are common requests tied to a session
- configure task to **specify URL** and type of **request**
- we will use a **completion handler** to interpret response from Server
- **larger** downloads allow use of delegates
    - progress indicators, completion indicators

URLSessionDataTask: GET

`dataTaskWithURL:completionHandler:(Data?,URLResponse?,Error?)`

URL as String

GET arguments as String

```
let baseURL = “\(SERVER_URL)/GetRequestURL" + query

let getUrl = URL(string: baseURL)
let request: URLRequest = URLRequest(url: getUrl!)

let dataTask : URLSessionDataTask = self.session.dataTask(with: request,
    completionHandler:{(data, response, error) in
        print("Response:\n%@",response!)
})

dataTask.resume() // start the task
```

must call, or stays suspended

URLSessionDataTask: POST

```
dataTaskWithURL:completionHandler:(Data?,URLResponse?,Error?)
```

```
    // create a custom HTTP POST request
    let baseURL = "\(SERVER_URL)/PostUrl"
    let postUrl = URL(string: "\(baseURL)")
    var request = URLRequest(url: postUrl!)

    let requestBody:Data? = UIImageJPEGRepresentation(image, 0.25);

     request.httpMethod = “POST"
    request.httpBody = requestBody

    let postTask : URLSessionDataTask = self.session.dataTask(with: request,
            completionHandler:{(data, response, error) in

     })

     postTask.resume() // start the task
```

could be any data

…we want this to be JSON…

JSON serialization

Dictionary

- serialize in iOS

```
let requestBody = try JSONSerialization.data(withJSONObject: jsonUpload,
          options:JSONSerialization.WritingOptions.prettyPrinted)
```

- parse in iOS

```
let jsonDictionary: Dictionary =
      try JSONSerialization.jsonObject(with: data!,
       options: JSONSerialization.ReadingOptions.mutableContainers) as! Dictionary
```

- parse in fastapi (taken care of for you with pydantic)

```
@app.post(
    "/labeled_data/",
    response_description="Add labeled data",
    response_model=LabeledDataPoint,
   response_model_by_alias=False,
)
```

fastapi + iOS demo

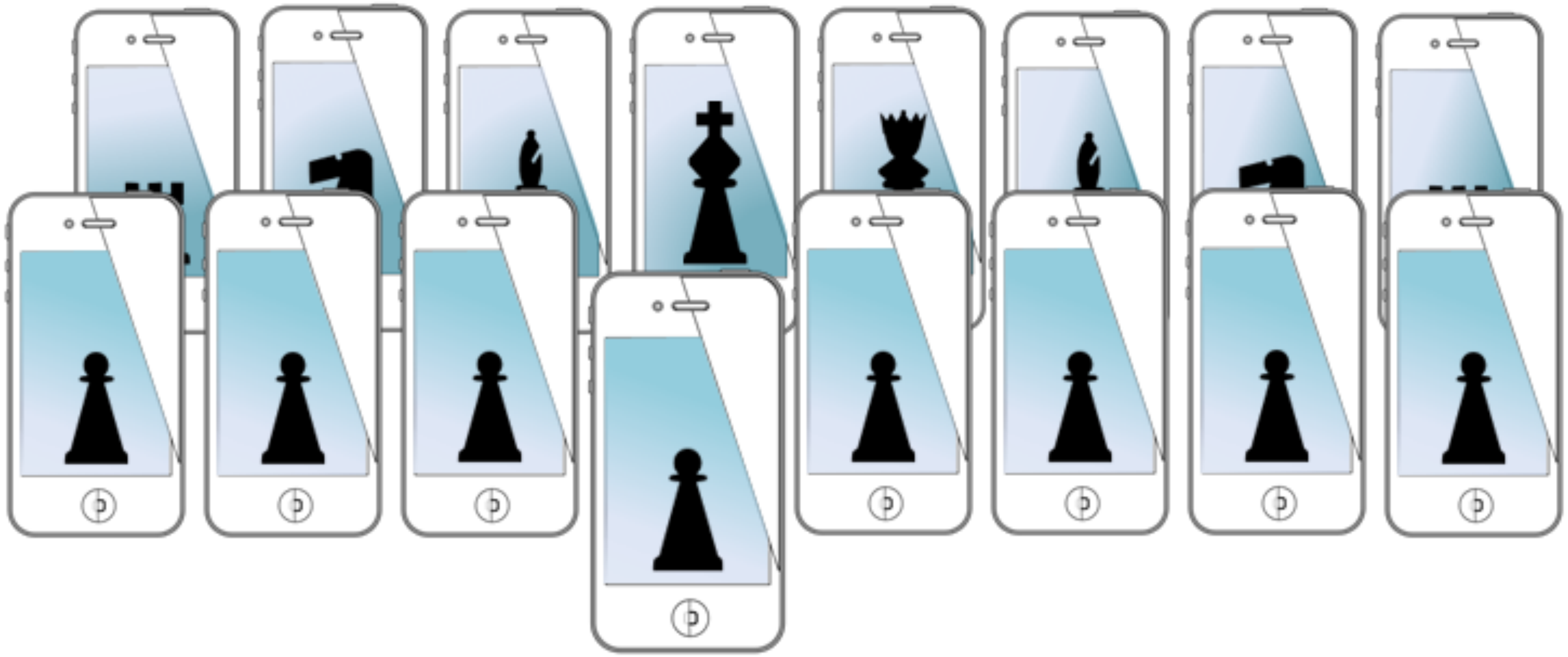
`HTTPSwiftExample.xcodeproject` (branch TestFastAPI)

`fastapi_example.py`

1. create marvel chars
2. access fastapi server
3. do POST with JSON in, JSON out

Cal Poly
San Luis Obispo
Drexel University
ingésup
Plymouth University
RMIT University
University of California Santa Cruz
SMU
Stanford
Technische Universität München
And now its time for a demo

MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

fastapi, pymongo, and http requests

Eric C. Larson, Lyle School of Engineering,
Computer Science, Southern Methodist University