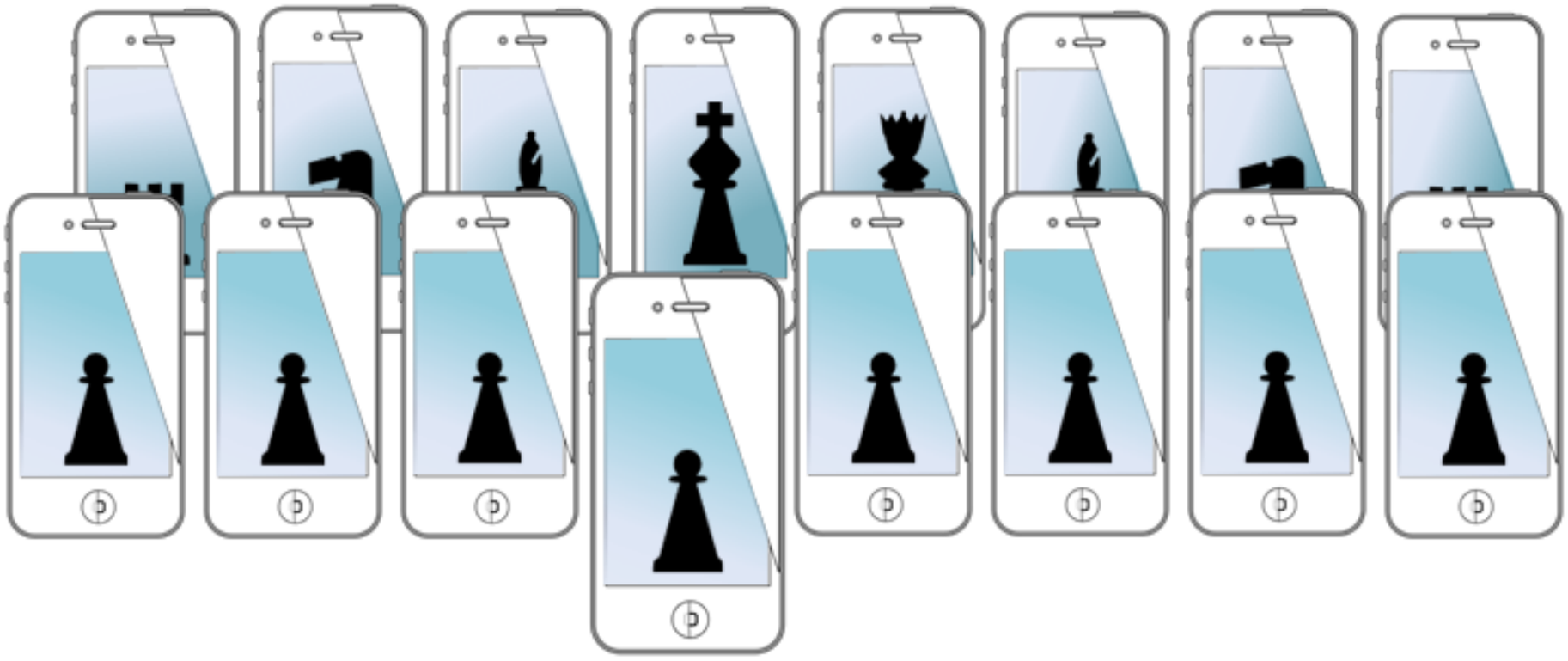
MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

tornado, pymongo, and http requests

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

course logistics

- A5 is due Friday
- start to think about the final project proposal

agenda

- tornado
- mongodb
- http requests in iOS

what are we doing?

- **preparing for A6**, need HTTP server that can:
  - accept (any) data
  - save it into a database
  - learn a (ML) model from that database
  - mediate queries and training of the model
- tornado is the event-driven architecture for interpreting the commands, routing the data, etc.
- our focus is building a deployment server, not an advanced ML algorithm (take DM or ML courses for that)

tornado web

- non-blocking web server
  - built for short-lived requests (pipelined)
  - and long lived connections
- built to scale
  - an attempt to solve the 10k concurrent problem
- has a python implementation
  - open sourced by Facebook after acquiring friendfeed.com
  - originally developed by the developers of gmail and google maps (the original releases)
- uses IOLoop and callback model

tornado web

also see this!
http://www.slideshare.net/kurtiss/tornado-web

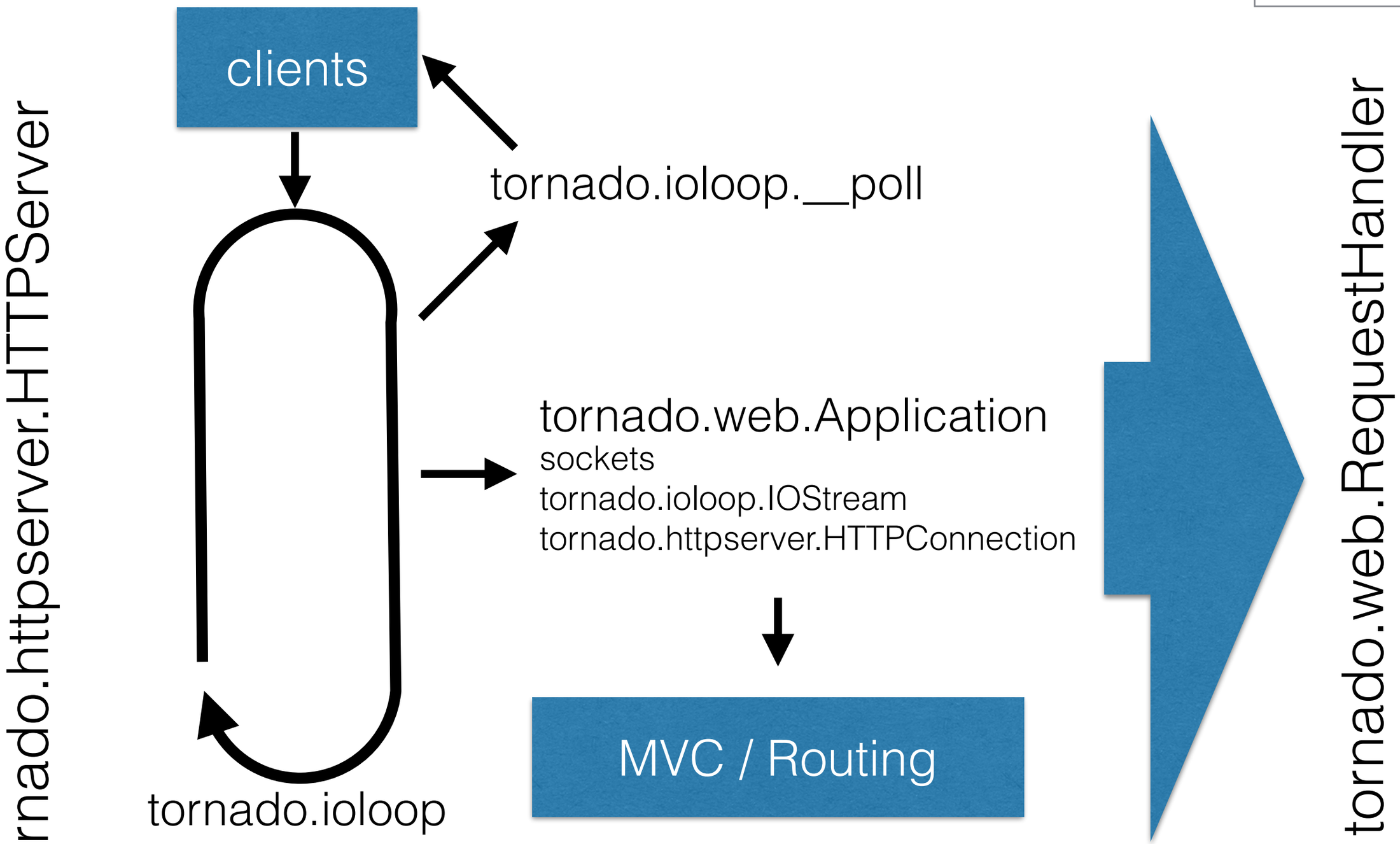
and this!
http://www.slideshare.net/gavinmroy/an-introduction-to-tornado?related=1

yeah, this too!
http://www.slideshare.net/chebrian/introduction-to-tornado?related=2

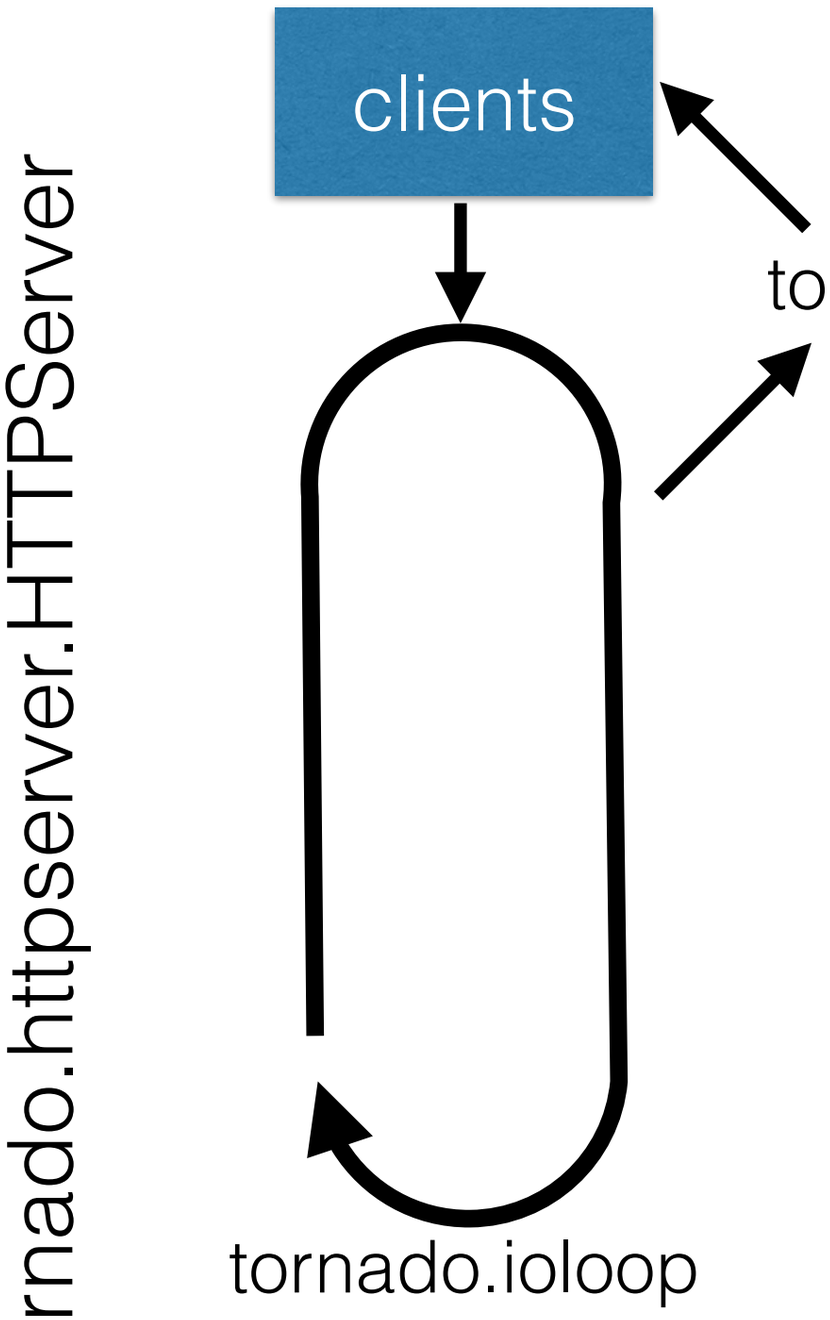
install tornado

- anaconda
  - conda install tornado
- pip
  - pip install tornado

tornado


clients
tornado.ioloop.__poll
tornado.httpserver.HTTPServer
tornado.ioloop
tornado.web.Application
sockets
tornado.ioloop.IOStream
tornado.httpserver.HTTPConnection
MVC / Routing
tornado.web.RequestHandler

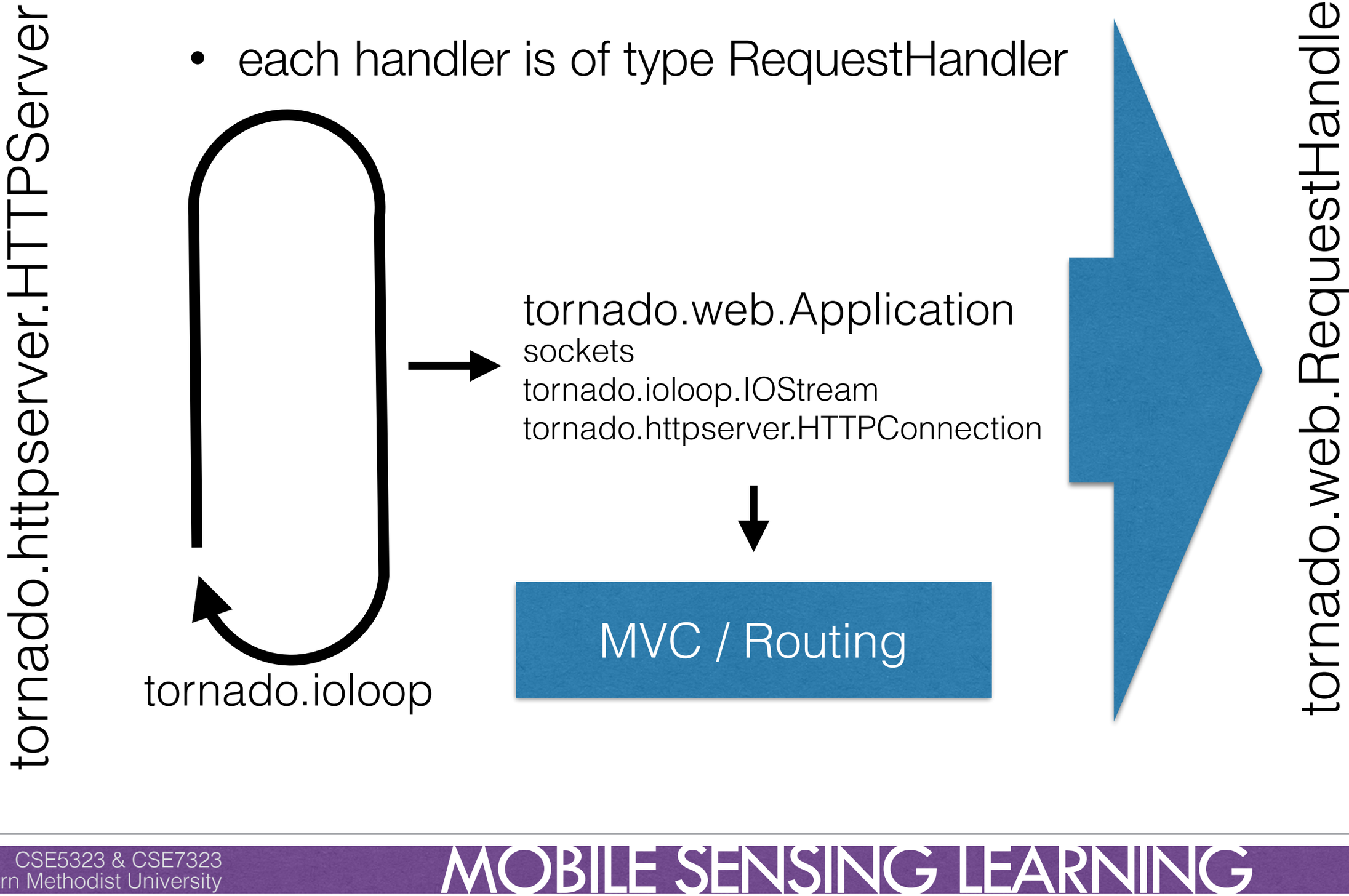
tornado



- edge triggered if possible
  - else becomes level triggered
- handles new connections
- handles new data from connection

tornado

- route URLs to different handlers
- each handler is of type RequestHandler



tornado example

- a very simple web server
- what is a get request?
  - a request for data fr[illegible]
  - URL contains any na[illegible]

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, MSLC World")

application = tornado.web.Application([
    (r"/", MainHandler),
])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

new class, inherit from RequestHandler

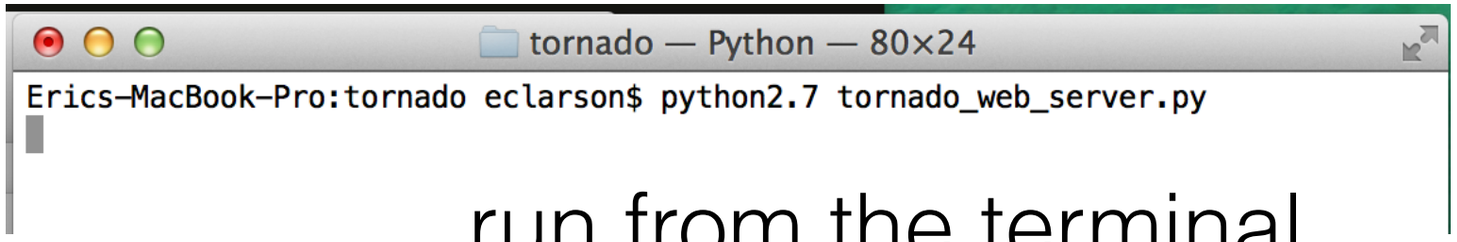
override get request handling

tuple with URL and handler

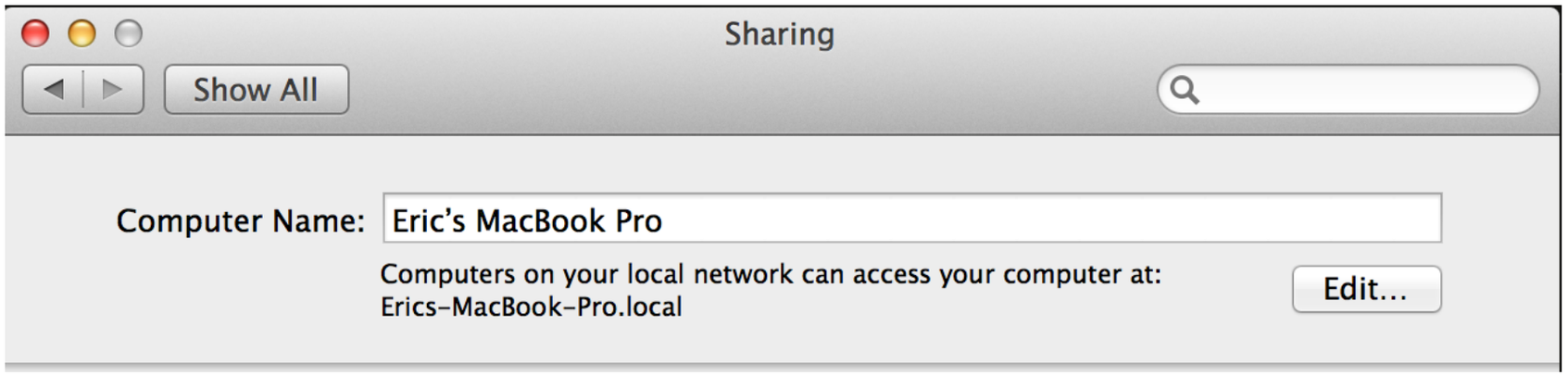
listen on 8888

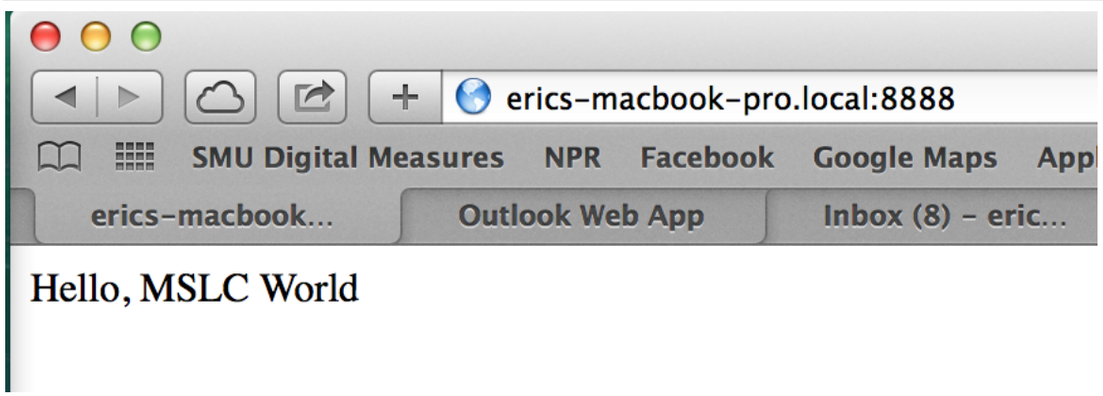
start the IO loop

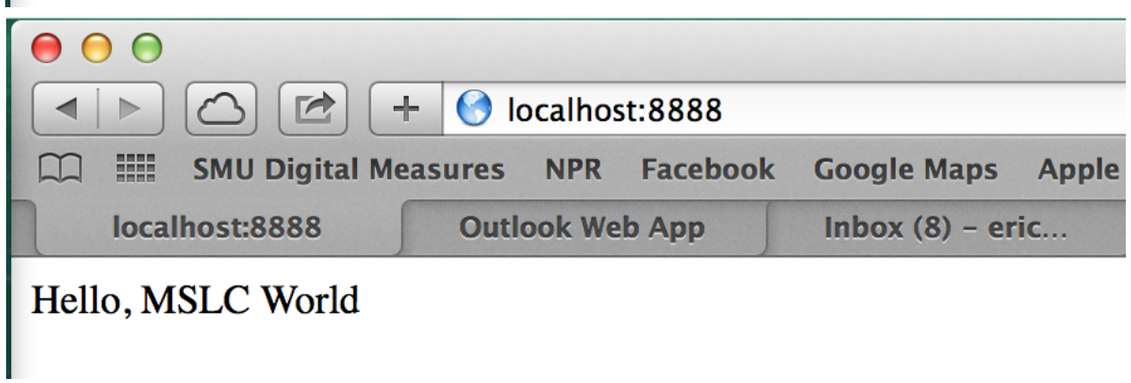
tornado example



run from the terminal







tornado

- get requests with arguments

```
class GetExampleHandler(tornado.web.RequestHandler):
    def get(self):
        arg = self.get_argument("arg", None, True) # get arg
        if arg is None:
            self.write("No 'arg' in query")
        else:
            self.write(str(arg)) # spit back out the argument
```

- how many connections?
  - one front end of Tornado~3,000 concurrent
  - with nginx and four instances of tornado
    - anywhere from 9,000-17,000
  - caveat: as long as you do not block the thread!

blocking example

reference slide

```
import tornado.ioloop
import tornado.web
import tornado.httpclient

flickrSearch = ‘https://www.flickr.com/services/rest/?
method=flickr.photos.getRecent&api_key=API_KEY'

class SearchHandler(tornado.web.RequestHandler):
   def get(self):
    self.write("Searching on Flickr!”)

    http_client = tornado.httpclient.HTTPClient()
    response = http_client.fetch(flickrSearch)

    self.write(" and we got a response! \n\n")
    self.write(response.body.replace("<", " "))
```

http://www.slideshare.net/moret1979/nginx-tornado-17k-reqs?next_slideshow=1

non-blocking example

reference slide

```
import tornado.ioloop
import tornado.web
import tornado.httpclient

flickrSearch = 'https://www.flickr.com/services/rest/?
method=flickr.photos.getRecent&api_key=MYSECRETKEY'

class SearchHandler(tornado.web.RequestHandler):
   @tornado.web.asynchronous
   def get(self):
    self.write("Searching on Flickr!")

    http_client = tornado.httpclient.AsyncHTTPClient()
    response = http_client.fetch(flickrSearch, callback=self.handle_response)

  def handle_response(self,response):
    self.write(" and we got a response! \n\n")
    self.write(response.body.replace("<", " "))
    self.finish()
```

decorator:
do not call finish!

http://www.slideshare.net/moret1979/nginx-tornado-17k-reqs?next_slideshow=1

sub-classing application

```
# tornado imports
import tornado.web
from tornado.web import HTTPError
from tornado.httpserver import HTTPServer
from tornado.ioloop import IOLoop
from tornado.options import define, options

# Setup information for tornado class
define("port", default=8000,
       help="run on the given port", ty

CUSTOM CLASSES AND DEFINITIONS

def main():
    '''Create server, begin IOLoop
    '''
    tornado.options.parse_command_line()
    http_server = HTTPServer(Application(), xheaders=True)
    http_server.listen(options.port)
    IOLoop.instance().start()

if __name__ == "__main__":
    main()
```

custom class for handling requests

sub-classing application

```
# Utility to be used when creating the Tornado server
# Contains the handlers and the database connection
class Application(tornado.web.Application):
    def __init__(self):
        '''Store necessary handlers,
           connect to database
        '''

        handlers = [(r"/[/]?",
                           BaseHandler),
                    (r"/Test[/]?",
                        examplehandlers.TestHandler),
                    (r"/DoPost[/]?",
                    MORE HANDLERS AND URL PATHS
                    ]

        settings = {'debug':True}
        tornado.web.Application.__init__(self, handlers, **settings)

        SETUP DATABASE

    def __exit__(self):
        self.client.close()
```

I wrote the base handler for you

handlers should subclass it

more to come in a moment

post versus get

Compare GET vs. POST

The following table compares the two HTTP methods: GET and POST.

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

credit: w3schools.com

BaseHandler Demo

- check out what it does
  - built for analyzing and writing back json
  - implements both get and post requests
  - put this in the main python file to access these:

```
# custom imports
from basehandler import BaseHandler
import examplehandlers
```

post versus get

- if we are sending data to a server for processing
  - of unknown length
  - of many different formats
  - possibly in a multi-part file
- should we use post or get requests?
- why?

post requests

- identical handling code in python
- in our implementation, return json

convenience function I wrote for you!

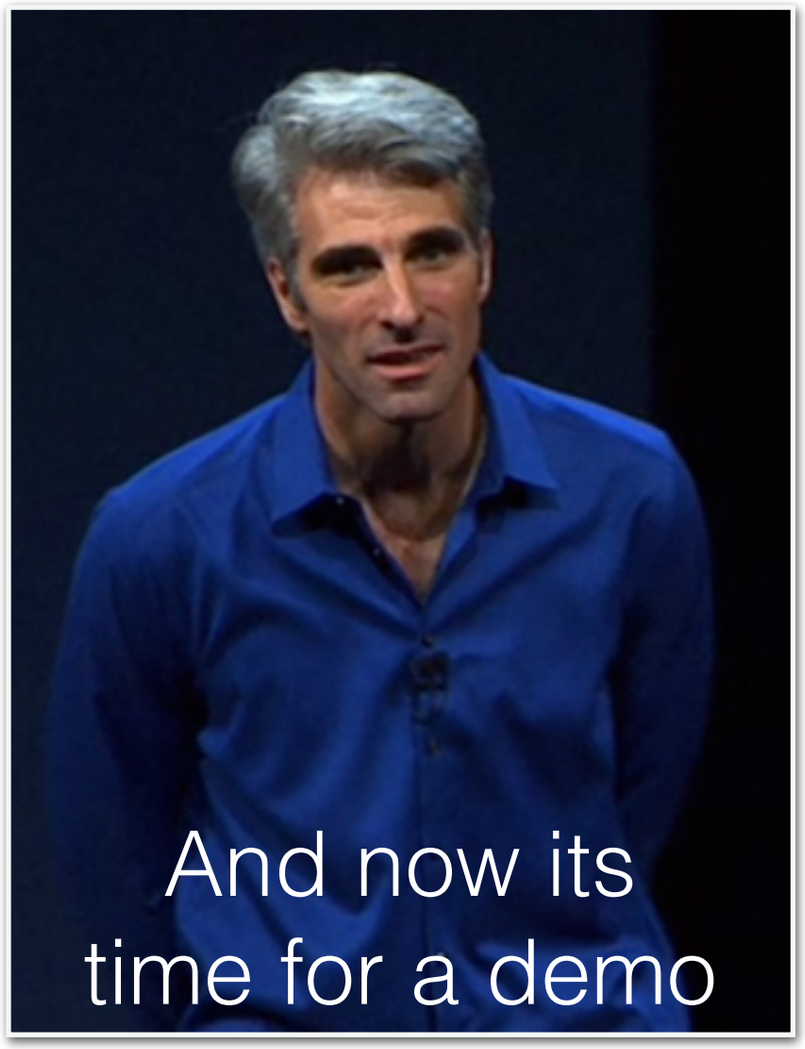
```
class PostHandler(BaseHandler):
  def get(self):
   '''respond with arg1*2
   '''
   arg1 = self.get_float_arg("arg1",default="none");
   self.write("Get from Post Handler? " + str(arg1*2));

  def post(self):
   '''Respond with arg1 and arg1*4
   '''
   arg1 = self.get_float_arg("arg1",default=1.0);
   self.write_json({"arg1":arg1,"arg2":4*arg1});
```

convenience function I wrote for you!

tornado examples

- with everything up except the database
- note that quick database queries are “okay” to block on



mongodb

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

mongodb install

- install it
- http://www.mongodb.org/downloads
- to run single server database:
  - make a directory for the db
    - like data/db
- run mongodb
  - ./mongod --dbpath “<path to db>”
- its running! localhost
-

make sure this exists!

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.

A MongoDB document.

The advantages of using documents are:

- Documents (i.e. objects) correspond to native data types in many programming languages.
- Embedded documents and arrays reduce need for expensive joins.
- Dynamic schema supports fluent polymorphism.

docs and collections

Database: MSLC_creations

limit on document size: ***16MB***

apps_collection

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

{
  app: “StepCount",
  users: 45,
  rating: 2.6,
}
....

{
  app: "shattner",
  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: "shattner",
  members: [ "Bill", “Will”, “Tom” ],
  numApps: 1,
  website: “shattner.com”,
}
```

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
```

relational equivalent of a table

create this database, if it does not exist

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

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

get collections

pymongo

- insertion

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

unique key, _id

- update

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

where ever this key is…

equal to this

set

this key to this value

insert if it does not exist

pymongo

- find one datum in database

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

its a list!

access the result

sort with this key

last element

- loop through all results

```
f=[];
for a in db.some_collect.find({"keyIWant":valueOfKeyIWant}):
    f.append( str(a['keyToGrabDataWith']) )
```

- 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({"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({"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(insert_list)
```

```
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'}
```

mongodb and binary data

- want to store binary data more than 16MB?
- use `gridfs`, its real simple
- use `put()` and `get()` instead of `insert()` and `find()`
  - `get()` returns a “file-like” object, so you can read in chunks

```
> from pymongo import MongoClient
> import gridfs
> db = MongoClient().gridfs_ex
> fs = gridfs.GridFS(db)

> a = fs.put("hello world”)
> fs.get(a).read()
'hello world'
```

```
> b = fs.put(fs.get(a),
       filename="foo", bar="baz")
> out = fs.get(b)
> out.read()          'hello world'
> out.filename          u'foo'
> out.bar               u'baz'
> out.upload_date
datetime.datetime(...)
```

http://api.mongodb.com/python/current/examples/gridfs.html

mongodb and binary data

- want to store binary data more than 16MB?
- use `gridfs`, its real simple
- use `put()` and `get()` instead of `insert()` and `find()`
  - `get()` returns a “file-like” object, so you can read in chunks

```
for grid_out in fs.find({"filename": "foo.txt"},
                        no_cursor_timeout=True):
    data = grid_out.read()


most_recent_three = fs.find().sort(
                        "uploadDate", -1).limit(3)
```

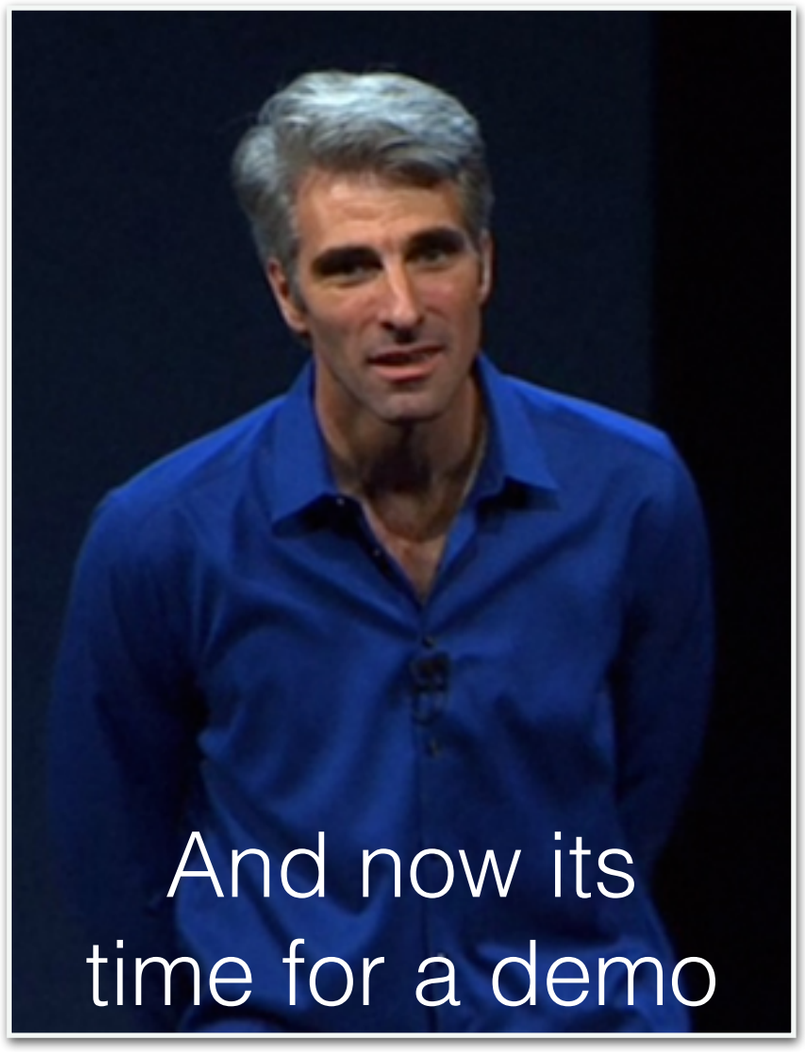
http://api.mongodb.com/python/current/examples/gridfs.html

mongodb + tornado

- we will use pymongo and tornado
  - mongodb runs localhost, tornado mediates access
  - good for learning
  - but real product would need load balancing (nginx)

mongodb + tornado

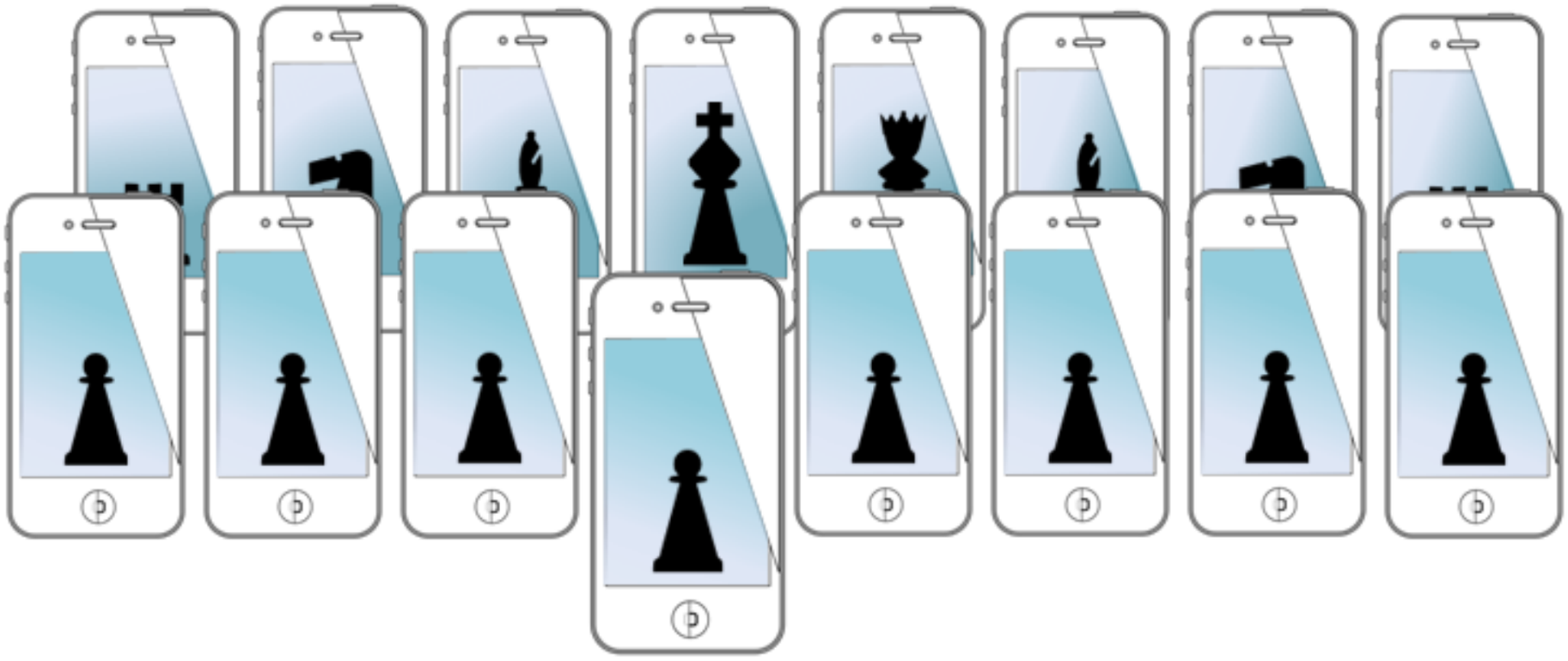
- demo:
  - store data inside mongodb with each http request



and add **something** to it

- ifconfig | grep "inet "

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

tornado, pymongo, and http requests

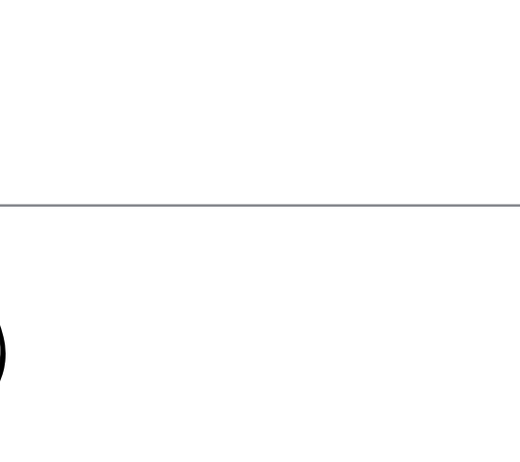
Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

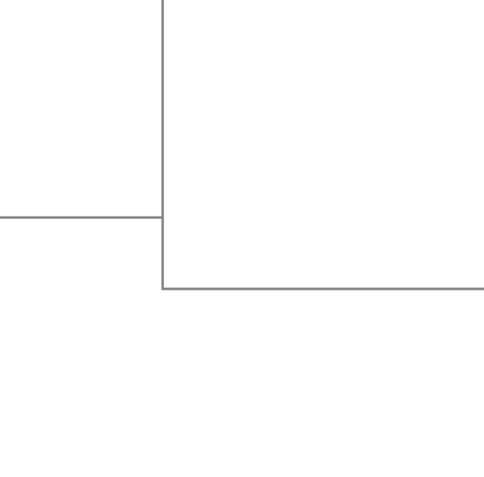
course logistics

- start to think about the final project proposal

agenda

- *finish tornado (done!)*
- *mongodb (done!)*
- http requests in iOS
- project proposals





working with your web server

- 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 NSURLSession

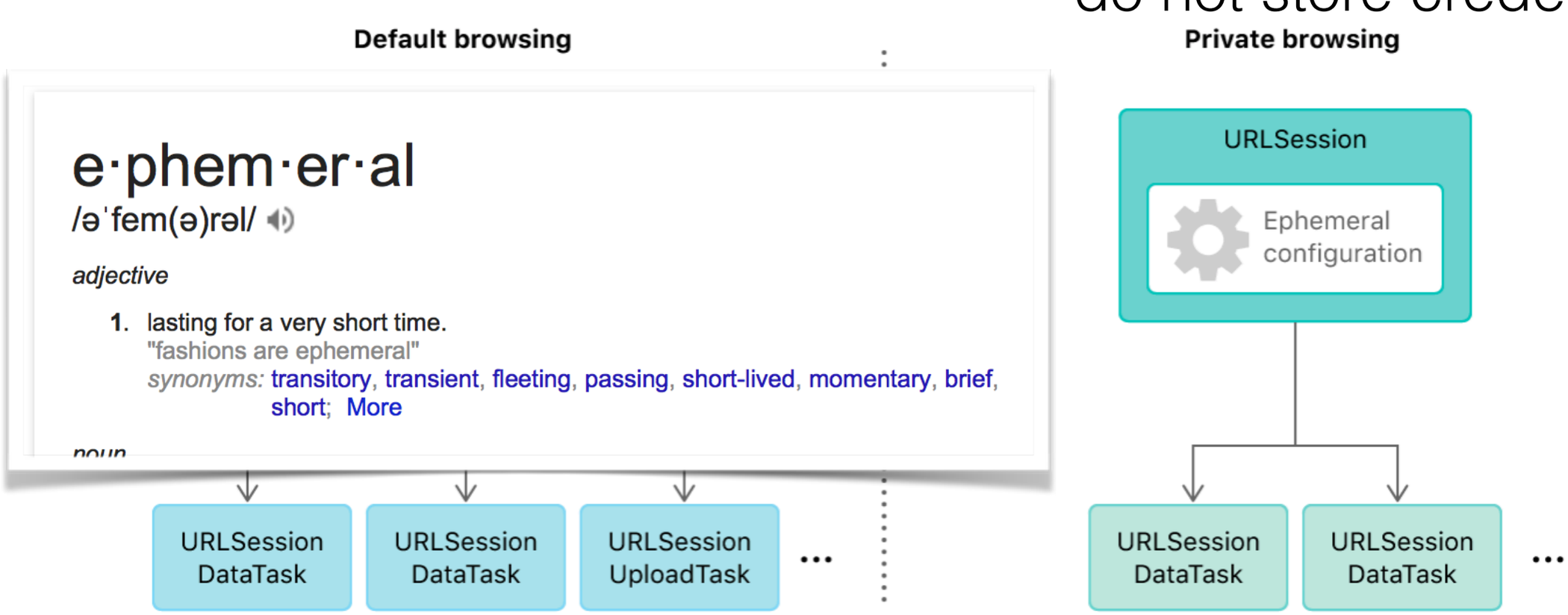
NSURLSession

- proper way to configure a session with a server
- new format starting in iOS7
  - old way was to use `NSURLConnection`
  - before that was to use `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

- delegate model
- does authentication if you need it!
  - we won't use that though — who would hack our server?
- implements pause / resume, tasks

do not cache
no cookies
do not store credentials



configure a session

```
class ViewController: UIViewController, URLSessionDelegate {

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

delegation

will reuse session

```
  //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)
```

custom queue

configure a session

- other options:

`ephemeralSessionConfiguration`

`defaultSessionConfiguration`

use global cache, cookies, and credential storage objects

`backgroundSessionConfiguration`

make my session respond to push notifications, launch my app, if needed, handle download completion

configure a task

- tasks are common requests tied to a session
- they give a way 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

- common to use for GET requests
- uses blocks for completion

```
dataTaskWithURL:completionHandler:
```

String

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

URLDownloadTask

reference slide

- sub-class of URLSessionDataTask
- many delegate methods for getting progress

```
    NSURLSessionDownloadTask *downloadTask = [self.session downloadTaskWithURL:[NSURL
URLWithString:@"someurlfordownloadingimages.com/coolimage"]
          completionHandler:^(NSURL *location, NSURLResponse *response, NSError *error) {
                if(!error)
                {
                    UIImage *img = [UIImage imageWithData:[NSData dataWithContentsOfURL:location]];
                }
            }];
    [downloadTask resume];
```

could use delegate instead of completion handler

```
-(void)URLSession:(NSURLSession *)session
     downloadTask:(NSURLSessionDownloadTask *)downloadTask
didFinishDownloadingToURL:(NSURL *)location

-(void)URLSession:(NSURLSession *)session
     downloadTask:(NSURLSessionDownloadTask *)downloadTask
     didWriteData:(int64_t)bytesWritten
totalBytesWritten:(int64_t)totalBytesWritten
totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite
```

URLSessionDataTask

- common to use for PUT/POST requests
- need to setup HTTP request (default is GET)

`uploadTaskWithRequest:fromData:completionHandler`

```
// 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

URLDataTask

reference slide

- can also use this for PUT or POST requests
- highly similar to uploadTask
  - but you don't get the delegate methods for progress

```
// create a custom HTTP POST request
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:postUrl];
[request setHTTPMethod:@"POST"];

NSData *imageData = UIImageJPEGRepresentation(image, 0.25);
[request setHTTPBody:imageData];

NSURLSessionDataTask *dataTask =
[self.session dataTaskWithRequest:request
        completionHandler:^(NSData *data, NSURLResponse *response, NSError *error) {

         }];
```

JSON serialization

- parse in tornado

```
import json

class JSONPostHandler(BaseHandler):
   def post(self):
    '''Parse some posted data
    '''
     data = json.loads(self.request.body)
```

- parse in iOS

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

the output in both scenarios is a dictionary

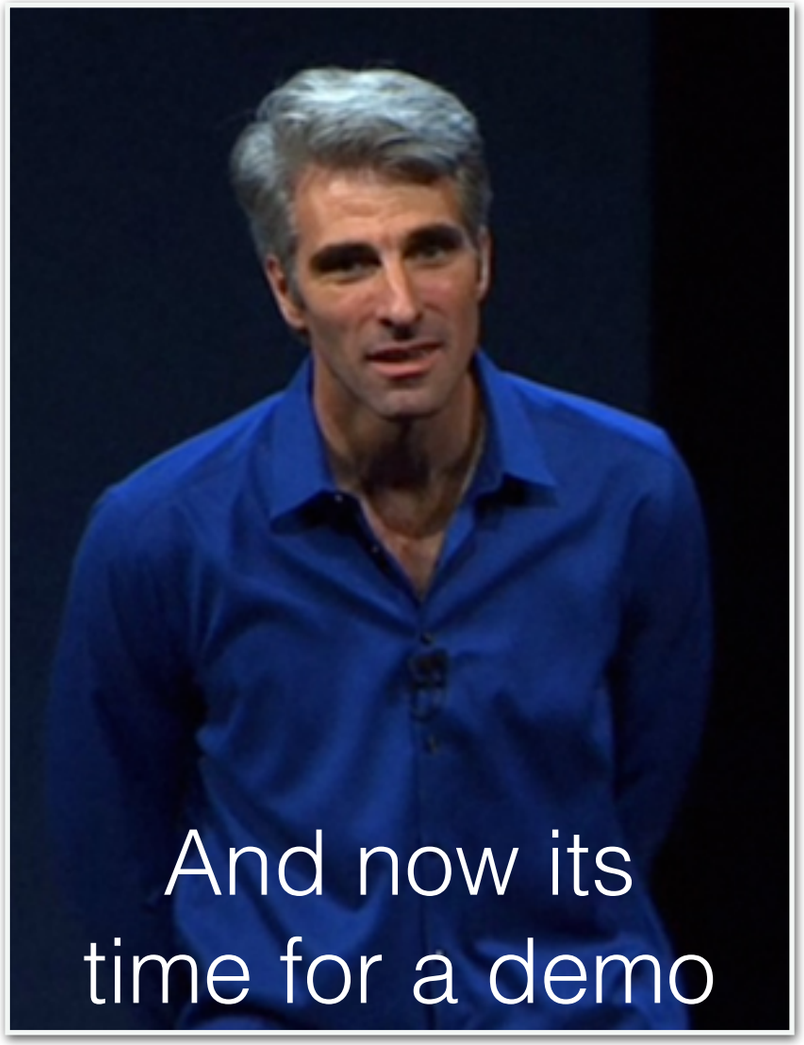
Dictionary

- serialize in iOS

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

tornado + iOS demo

- send a GET request, handle query in tornado
- do POST with GET-like query
- do POST with JSON in, JSON out



before talking about proposals for next time…

- next time: basics of machine learning
  - machine learning as a service
    - install scikit-learn (batteries included with anaconda)
- next next time: **in-class-assignment** with our own restful API
  - using ML and networking to do cool things…
  - some code has changed since filming, but mostly the same

project proposal

Final Project Proposal

All final projects should be approved by the instructor via the final project proposal. This is a description explaining the overall idea, which labs the project builds from, and a list of four or more design constraints that the design will meet. Turn in the project proposal via canvas or talk it over with the instructor.

If your group is opting into the "mother of all demos" (see explanation below) your proposal must specify this (you can opt out later, but you cannot opt in). Note that MOD projects should have more difficult constraints. The instructor may tighten constraints for those that wish to opt into the MOD.

exceptional final project example

JukeboxHero

We will build an iOS application that works as a guitar pedal. We are aiming for users to be musicians who want play along and practice songs from their music library. The phone will allow a “Jam Session.” Users connect their guitar and listen via headphones or external speakers. It will have five different guitar effects including distortion and chorus. Some effects will have varying levels of tuning (*i.e.*, the amount of distortion).

The app will play songs from the user’s music library and will consult our web server for information about the songs (like the tuning of the guitar, so the user can play along). It will allow the users to tune their guitar to the song being played. We will allow control over the effects through CoreMotion. For example, moving the phone with your foot controls a predefined progression of different effects.

An example of the closest app on the app store is <here>.

The constraints to be met in the project are:

1. Recognition of a “foot” gesture. We will validate our recognition by teaching the instructor to “select” through the foot gesture, then allowing the instructor to use the “foot select” to change effects. We will have at least 90% recognition.
2. Five different effects will be implemented. At least two will be tunable. We will evaluate by showing a progression of effects through external speakers and connecting a guitar during the final demonstration.
3. We will have a tuner in the application for tuning the strings of a guitar. We will demonstrate this with a guitar during the final demonstration. We will also show the output of a real tuner, for reference. The phone tuner and the purchased tuner will be no more than 1 Hz apart.
4. last constraint — the web server. **What might this look like?**

past final projects 2014

past final projects 2015

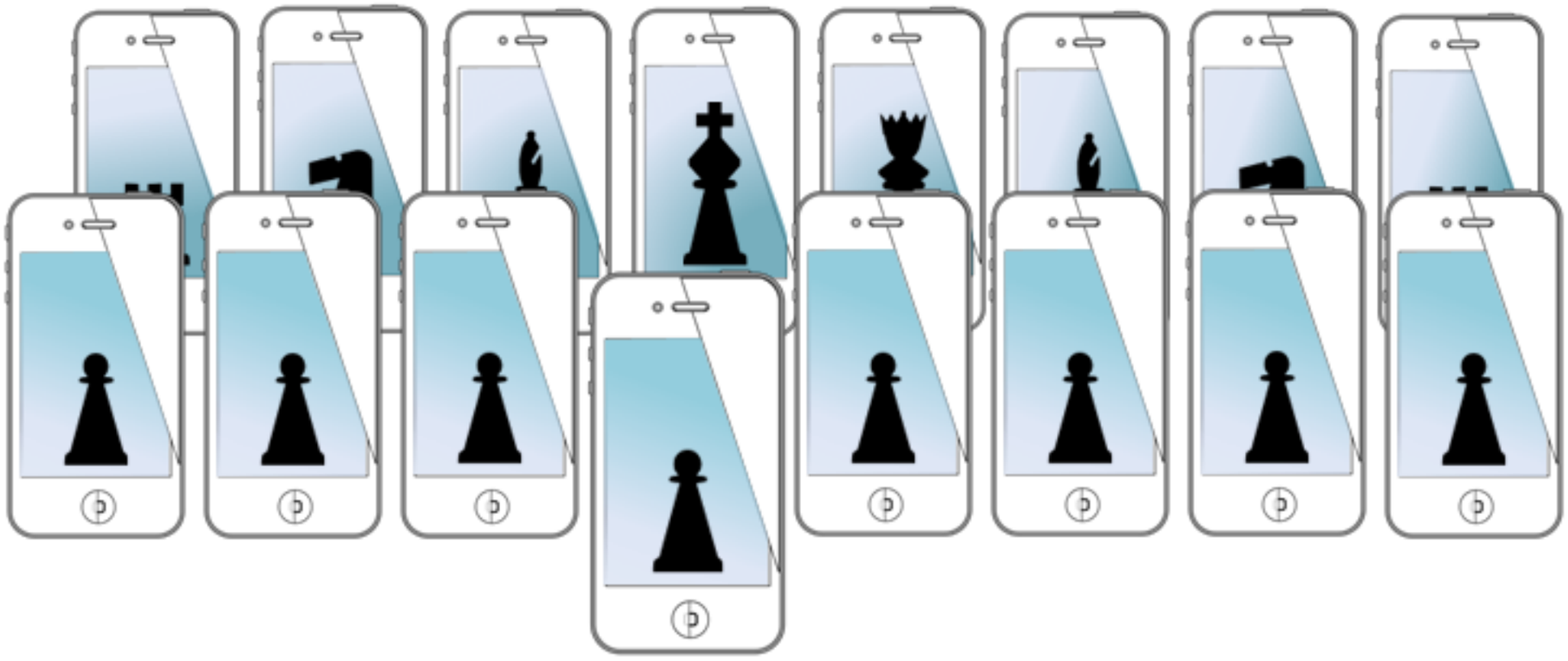
past final projects 2015

past final projects 2016

past final projects 2017

past final projects 2017

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

tornado, pymongo, and http requests

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