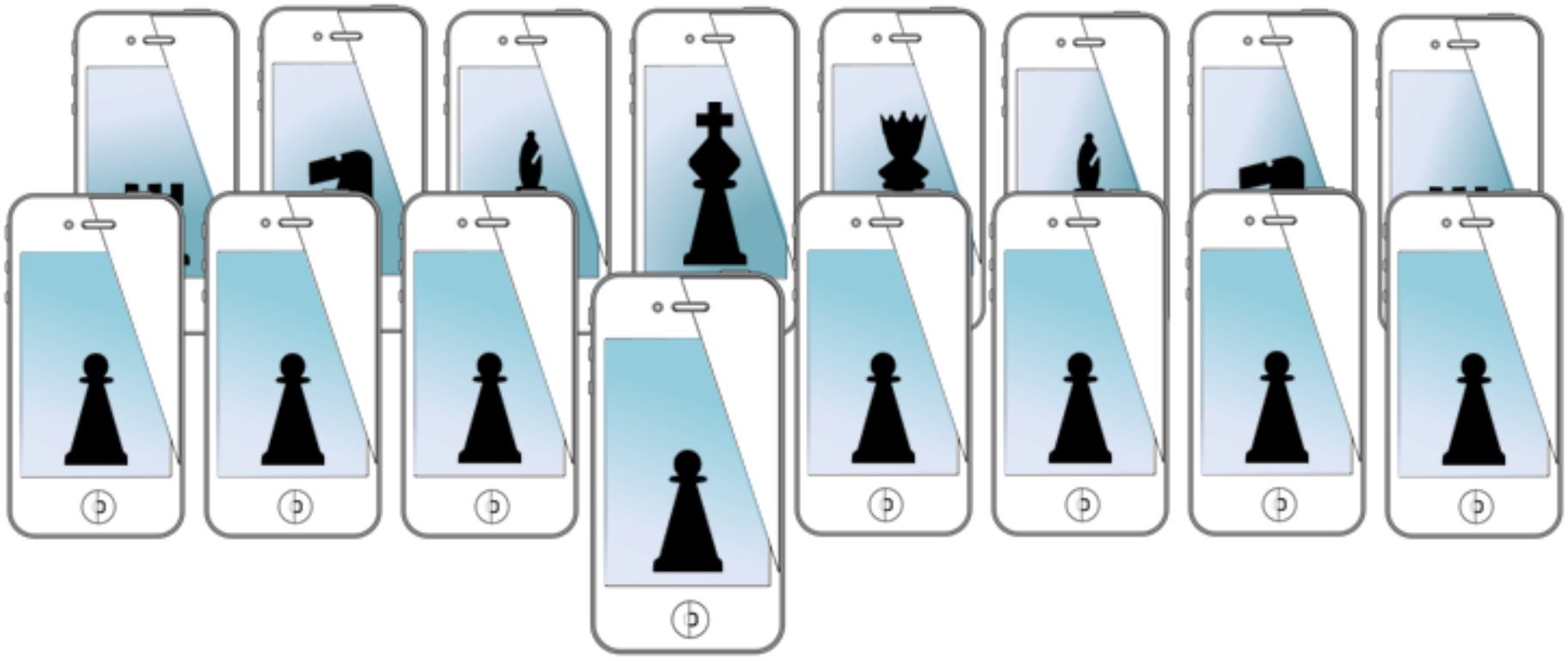
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

python crash-course, tornado

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

course logistics

- lab three due this week
  - motion and game
- two weeks after that
  - lab four due: images
- two weeks after that
  - lab five due: machine learning service
  - final project proposal due

agenda

- last time: OPENCV
- history of python
- syntax
  - pythonic conventions
  - simple examples
- web handling with tornado
- document databases

python

- Guido van Rossum

From wikipedia:

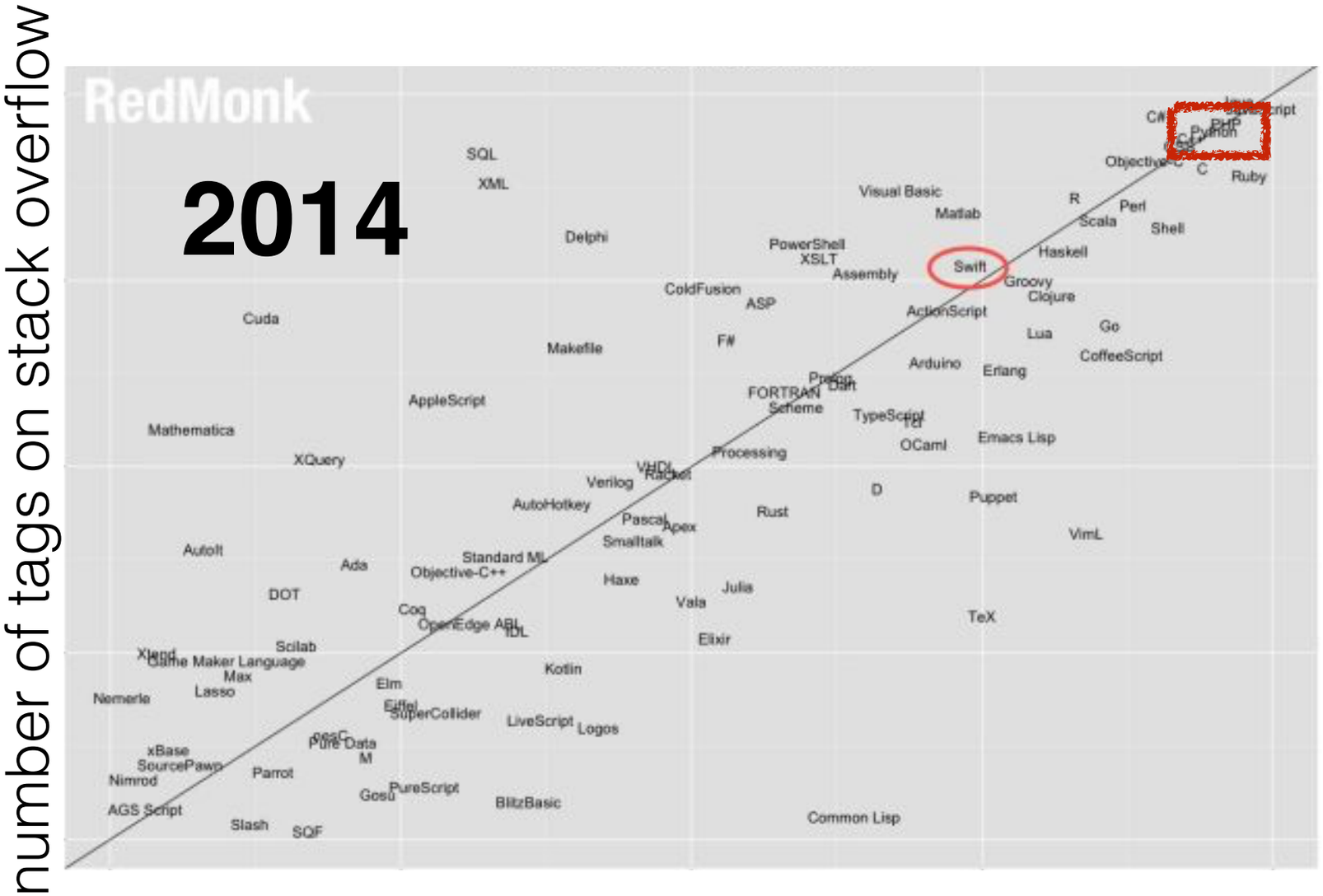
Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

-Guido van Rossum in 1996

python adoption

- appears in every programming top ten list

you (probably) should know it!



RedMonk Q320 Programming Language Rankings

RedMonk

Popularity Rank on Stack Overflow (by # of Tags)

Popularity Rank on GitHub (by # of Projects)

0 25 50 75 100

JavaScript
Java
Python
C#
PHP
C++
CSS
R
Objective-C
Swift
Ruby
TypeScript
Scala
PowerShell
Shell
Go
Perl
Haskell
Kotlin
Dart
Groovy
Lua
Rust
Clojure
Visual Basic
Matlab
Assembly
GCC Machine Description
Arduino
F#
ColdFusion
Cuda
Prolog
ASP
FORTRAN
CoffeeScript
QML
ActionScript
Erlang
Elixir
Julia
Scheme
Tcl
GLSL
OCaml
Processing
Common Lisp
VHDL
Racket
Verilog
RobotFramework
XQuery
Mathematica
Smarty
Puppet
Emacs Lisp
Liquid
FreeMarker
D
SystemVerilog
Pascal
Web Ontology Language
Coq
Standard ML
Apex
Elm
Solidity
Smalltalk
Haxe
Raku
SaltStack
Objective-C++
WebAssembly
PostScript
BitBake
Gherkin
Vala
Modelica
Crystal
Nix
PureScript
Nim
Reason
API Blueprint
Chapel
Hack
GDScript
Game Maker Language
HCL
Logos
ShaderLab
Jsonnet
Vim script
SourcePawn
GAP
Starlark
SQF

python disclaimers

- weakly typed variables (dynamic)
- its an interpreter (kinda)
  - loops are slow
  - until they are not (compile it)
- can't use parallel instructions natively
- kinda similar to swift, more than you know
- can be the glue for your different codebases

python releases

- 1.0 (up to 1.6)
  - basic python, complex numbers, lambdas
- 2.0 (up to 2.7.X, deprecated, but still used…)
  - unified types, made completely object oriented
- 3.0 (up to 3.X, updated recently)
  - eliminate multiple paradigms (kinda)
  - 2.x not necessarily compatible with 3.x

installation

- install anaconda
- use latest python 3
- use conda environments

python

- pick the IDE you want
- Jupyter (not an IDE, but good for editing)
- PyCharm, very good, supports breakpoints and watch
- XCode can also be used, but is more limited than pycharm

python

- many different coding “styles”
- “best” styles get the distinction of “pythonic”
  - ill formed definition
  - changes as the language matures
- pythonic code is:
  - simple and readable
  - uses dynamic typing when possible
- …or to quote Tim Peters…

python zen

type this

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass
Unless explicitly silenc                                    ss.
In the face of ambiguity                                    s way to do it.
There should be one-- ar                                    ou're Dutch.
Although that way may no
Now is better than never.
Although never is often better tha  right now.
If the implementation is hard to e plain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

get this

python is quirky
but, don't assume that means
it is not a **serious** tool

syntax, python 3

- numbers
  - int or float
- complex numbers

```
>>> 7*5
35
>>> 5/7
0.7142857142
>>> 7/5
1.4
>>> 7.0/5
1.4
```

```
>>> tmpVar = 4
>>> print (tmpVar)
4
>>> tmpVar/8
0.5
>>> tmpVar/8.0
0.5
```

```
>>> 1+1j
(1+1j)
>>> (1+1j)*5
(5+5j)
>>> 1+1j + 4
(5+1j)
```

syntax

- strings
  - immutable

```
>>> 'single quotes'
'single quotes'
>>> "double quotes"
'double quotes'
>>> 'here is "double quotes"'
'here is "double quotes"'
>>> 'here is \'single quotes\''
"here is 'single quotes'"
>>> "here are also \"double quotes\""
'here are also "double quotes"'
```

```
>>> someString = 'MobileSensingAndLearning'
>>> someString[:5]
'Mobil'
>>> someString[5:]
'eSensingAndLearning'
>>> someString+'AndControl'
'MobileSensingAndLearningAndControl'
>>> someString*3
'MobileSensingAndLearningMobileSensingAndLearningMobileSensingAndLearning'
>>> someString[-5:]
'rning'
>>> someString[:-5]
'MobileSensingAndLea'
>>> someString[5]
'e'
>>> someString[-1]
'g'
>>> someString[-2]
'n'
```

```
>>> someString[5] = 'r'

Traceback (most recent call last):
    File "<pyshell#32>", line 1, in <module>
        someString[5] = 'r'
TypeError: 'str' object does not support item assignment
```

syntax

- tuples
- lists
  - highly versatile and mutable
  - containers for anything

```
>>> aTuple = 45, 67, "not a number"
>>> aTuple
(45, 67, 'not a number')
```

immutable

```
>>> aList = ["a string",5.0,6,[4,3,2]]
>>> print(aList)
['a string', 5.0, 6, [4, 3, 2]]
>>> aList[0]
'a string'
>>> aList[2]
6
>>> aList[-1]
[4, 3, 2]
```

```
>>> anotherList = []
>>> i=0
>>> i+=1
>>> i
1
>>> while i<1000:
    anotherList.append(i)
    i+=i

>>> print anotherList
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

```
>>> len(aList)
4
>>> len(aList[-1])
3
>>> aList[0:1]=[]
>>> print(aList)
[5.0, 6, [4, 3, 2]]
>>> aList[0:2]=[]
>>> print(aList)
[[4, 3, 2]]
```

syntax loops

- for, while
  - indentation ~~matters~~ *is the only thing* that **matters**

```
i=0
while i<10:
    print (str(i) + ' is less than 10’)
    i+=1
else:
    print (str(i) + ' is not less than 10’)
```

```
0 is less than 10
1 is less than 10
2 is less than 10
3 is less than 10
4 is less than 10
5 is less than 10
6 is less than 10
7 is less than 10
8 is less than 10
9 is less than 10
10 is not less than 10
```

```
classTeams = ['Team','Monkey','CHC',
              'ThatGuyInTheBack',42]

for team in classTeams:
    print (team * 4)
else:
    print (‘ended for loop without break’)
```

```
TeamTeamTeamTeam
MonkeyMonkeyMonkeyMonkey
CHCCHCCHCCHC
ThatGuyInTheBackThatGuyInTheBackThatGuyInTheBackThatGuyInTheBack
168
ended for loop without break
```

syntax loops

- for, while
  - indentation ~~matters~~ *is the only thing* that **matters**

```
for i in range(10):
    print (i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
for j in range(2,10,2):
 print (j)
```

```
2
4
6
8
```

data structures

- **lists** can be used as a stack

```
>>> classTeams = ['Team','Monkey','CHC','ThatGuyInTheBack',42]
>>> classTeams.pop()
42
>>> classTeams.pop()
'ThatGuyInTheBack'
>>> classTeams.sort()
>>> classTeams
['CHC', 'Monkey', 'Team']
```

  - or can import queues
    - append(value)
    - pop_left(), deque first element
- **dictionaries**

```
>>> myDictionary = {"teamA":45,"teamB":77}
>>> myDictionary
{'teamA': 45, 'teamB': 77}
>>> myDictionary["teamA"]
45
```

lists and loops

- comprehensions

```
>>> timesFour = [x*x*x*x for x in range(10)]
>>> timesFour
[0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561]
```

```
from random import randint
grades = ['A','B','C','D','F']
teamgrades = [grades[randint(0,4)] for t in range(8)]
print (teamgrades)
```

```
['C', 'A', 'B', 'F', 'A', 'C', 'A', 'D']
```

can be nested as much as you like!

only **pythonic** if it makes the code **more readable**

```
>>> timesFour = {x:x*x*x*x for x in range(10)}
>>> timesFour
{0: 0, 1: 1, 2: 16, 3: 81, 4: 256, 5: 625, 6: 1296, 7: 2401, 8: 4096, 9: 6561}
```

can use comprehensions with dictionaries too!

lists and loops

reference slide

```
>>> timesFour = {x:x*x*x*x for x in range(10)}
>>> timesFour
{0: 0, 1: 1, 2: 16, 3: 81, 4: 256, 5: 625, 6: 1296, 7: 2401, 8: 4096, 9: 6561}
```

can use comprehensions with dictionaries too!

```
from random import randint

teams = ['CHC','Team','DoerrKing','MCVW','etc.']
grades = ['A','B','C','D','F']
teamgrades = {team:grades[randint(0,4)] for team in teams}
teamgrades

{'etc.': 'F', 'CHC': 'A', 'DoerrKing': 'B', 'MCVW': 'B', 'Team': 'A'}
```

pop quiz!

- add the numbers from 0 to 100, not including 100

```
sumValue = 0
for i in range(100):
    sumValue += i

print (sumValue)
print (sum(range(100)))
print (100*(100-1)/2)
```

more pythonic?

or use real math

now, print the **index** and **value** of elements in a list

```
list = [1,2,4,7,1,5,6,8]

for i in range(len(list)):
    print (str(list[i]) + " is at index " + str(i))

for i,element in enumerate(list):
    print (str(element) + " is at index " + str(i))
```

```
1 is at index 0
2 is at index 1
4 is at index 2
7 is at index 3
1 is at index 4
5 is at index 5
6 is at index 6
8 is at index 7
```

more pythonic

conditionals

reference slide

- if, elif, else, None, is, or, and, not, ==

```
a=5
b=5

if a==b:
    print (“Everybody is a five!”)
else:
    print (“Wish we had fives…”)
```

```
Everybody is a five!
```

```
a=327676
b=a

if a is b:
    print (“These are the same object!”)
else:
    print (“Wish we had the same objects…")
```

```
These are the same object!
```

```
a=327676
b=327675+1

if a is b:
    print (“These are the same object!”)
else:
    print (“Wish we had the same objects…")
```

```
Wish we had the same objects
```

```
a=5
b=4+1

if a is b:
    print (“Everybody is a five!”)
else:
    print (“Wish we had fives…")
```

small integers are cached
strings behave the same

```
Everybody is a five!
```

conditionals

reference slide

```
teacher = "eric"

if teacher is not "Eric":
    print (“Go get the prof for this class!”)
else:
    print (“Welcome, Professor!”)
```

```
Go get the prof …
```

```
teachers = ["Eric","Paul","Ringo","John"]

if "Eric" not in teachers:
    print (“Go get the prof for this class!”)
else:
    print (“Welcome, Professor!”)
```

```
Welcome!
```

```
teachers = ["Eric","Paul","Ringo","John"]
shouldCheckForTeacher = True

if "Eric" not in teachers and shouldCheckForTeacher:
    print (“Go get the prof for this class!”)
elif shouldCheckForTeacher:
    print (“Welcome, Professor!”)
else:
    print (“Not checking”)
```

```
Welcome!
```

functions

- def keyword
  - like c, must be defined before use

```
def show_data(data):
    # print the data
    print (data)

some_data = [1,2,3,4,5]
show_data(some_data);
```

```
[1, 2, 3, 4, 5]
```

```
def show_data(data,x=None,y=None):
    # print the data
    print data
    if x is not None:
        print (x)
    if y is not None:
        print (y)

some_data = [1,2,3,4,5]
show_data(some_data);
show_data(some_data,x='a cool X value')
show_data(some_data,y='a cool Y value',x='a cool X value')

def get_square_and_tenth_power(x):
    return x**2,x**10

print (get_square_and_tenth_power(2))
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
a cool X value
[1, 2, 3, 4, 5]
a cool X value
a cool Y value

(4, 1024)
```

debugging

- the python debugger
  - http://docs.python.org/2/library/pdb.html
  - if you have not used it, I just changed your life
- `import pdb, pdb.set_trace()`
- command line arguments
  - `s`(tep), `c`(ontinue), `n`(ext), `w`(here), `l`(ist), `r`(eturn), `j`(ump)
  - and much more… like `print, p, pp`
  - can set numbered break points by running from python window
    - `python -m pdb your_function.py`

python demos

- more demos:

http://sandbox.mc.edu/~bennet/python/code/index.html?utm_source=twitterfeed&utm_medium=twitter

classes

- multiple inheritance
- “self” is always passed as first argument

```
class BodyPart(object):
    def __init__(self,name):
        self.name = name;


class Heart(BodyPart):
    def __init__(self,rate=60,units="minute"):
        self.rate = rate
        self.units= units
        super().__init__("Heart")

    def __str__(self):
        print (“name:" + str(self.name) + " has " + str(self.rate) + " beats per " + self.units)


myHeart = Heart(1,"second")
print(myHeart)
```

python generators

- kinda like static variables
- used to create iterables
- lots more that you can do, like send in values

```
def get_primes(number):
    while True:
        if is_prime(number):
            yield number
        number += 1
```

yield allows iteration

```
total = 2
for next_prime in get_primes(3):
    if next_prime < 2000000:
        total += next_prime
    else:
        break
```

https://jeffknupp.com/blog/2013/04/07/improve-your-python-yield-and-generators-explained/

python syntax “with”

reference slide

- the “with” statement
- defines an “enter” and an “exit” protocol
- used commonly for opening files, where “open” adopts the “with” protocol

```
file = open("/some_file.txt")
try:
    data = file.read()
finally:
    file.close()



with open("/some_file.txt") as file:
    data = file.read()
```

python decorators

reference slide

- wrap your method inside another method
- the wrapper changes some functionality

```
from time import sleep

def sleep_decorator(function):
    def wrapper(*args, **kwargs):
        sleep(2)
        return function(*args, **kwargs)
    return wrapper

@sleep_decorator
def print_number(num):
    return num

print(print_number(222))

for num in range(1, 6):
    print(print_number(num))
```

used a **bunch** in web applications before python 3.5

python async/await

- new in python 3.5: awaitable objects

co-routine:
awaitable methods

```
import asyncio

async def nested():
    return 42

async def main():
    # Nothing happens if we just call "nested()".
    # A coroutine object is created but not awaited,
    # so it *won't run at all*.
    nested()

    # Let's do it differently now and await it:
    print(await nested())  # will print "42".

asyncio.run(main())
```

tasks:
awaitable objects

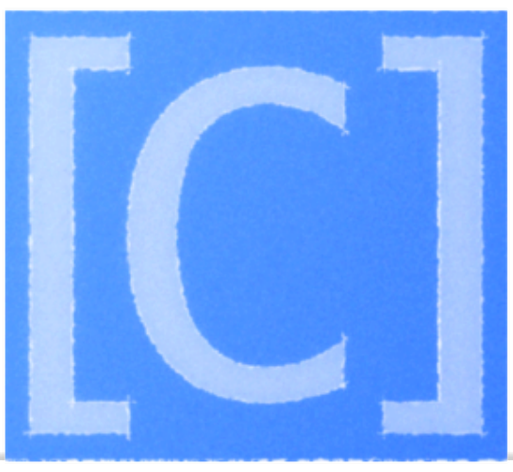
```
import asyncio

async def nested():
    return 42

async def main():
    # Schedule nested() to run soon concurrently
    # with "main()".
    task = asyncio.create_task(nested())

    # "task" can now be used to cancel "nested()", or
    # can simply be awaited to wait until it is complete
    await task

asyncio.run(main())
```

futures
gathering awaitable routines

```
async def main():
    await function_that_returns_a_future_object()

    # this is also valid:
    await asyncio.gather(
        function_that_returns_a_future_object(),
        some_python_coroutine()
    )
```

why are we learning python?

- its the glue for:
  - tornado
  - mongodb
  - http requests in iOS



what are we doing?

- **preparing for A5**, 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)

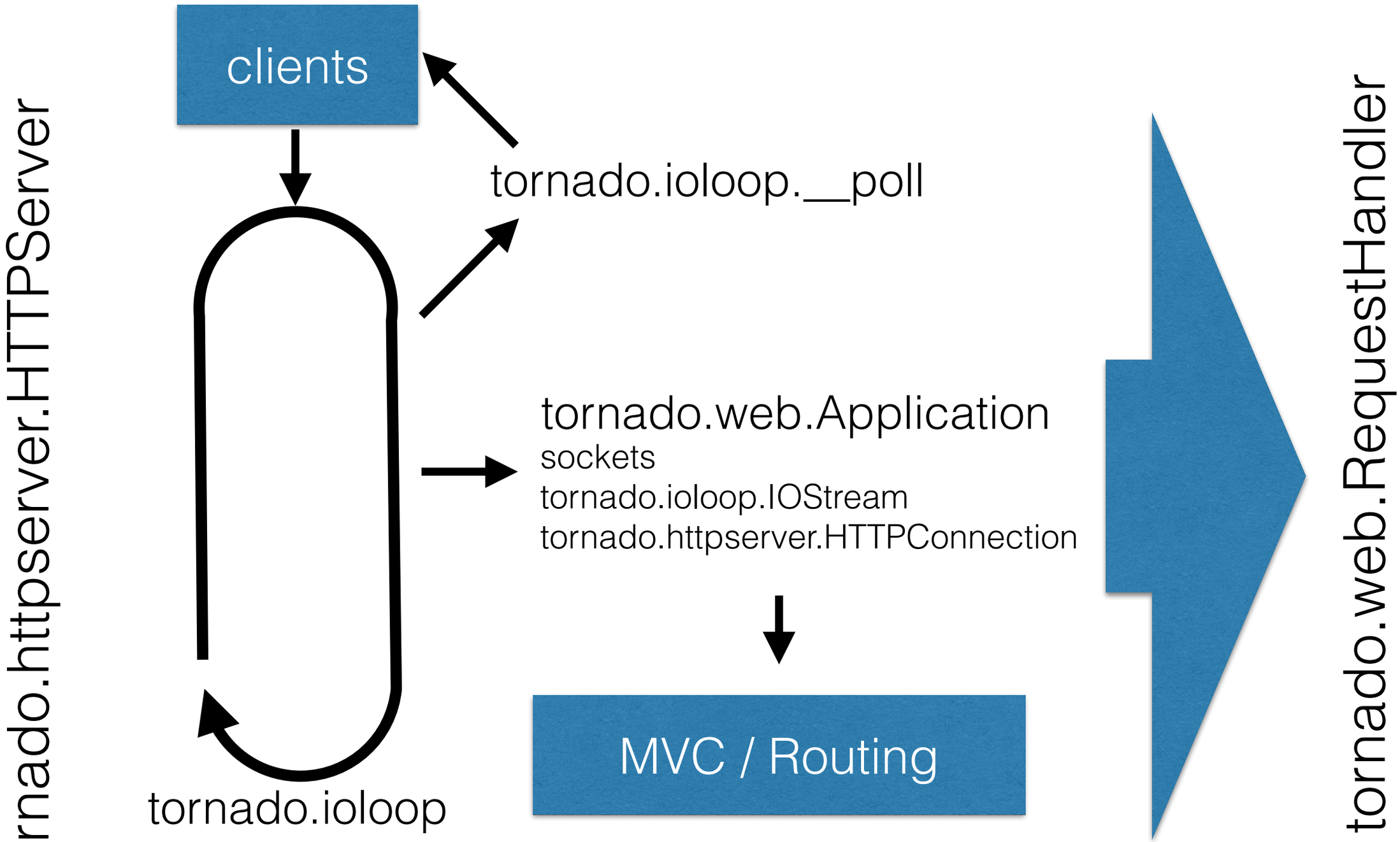
tornado web

- 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

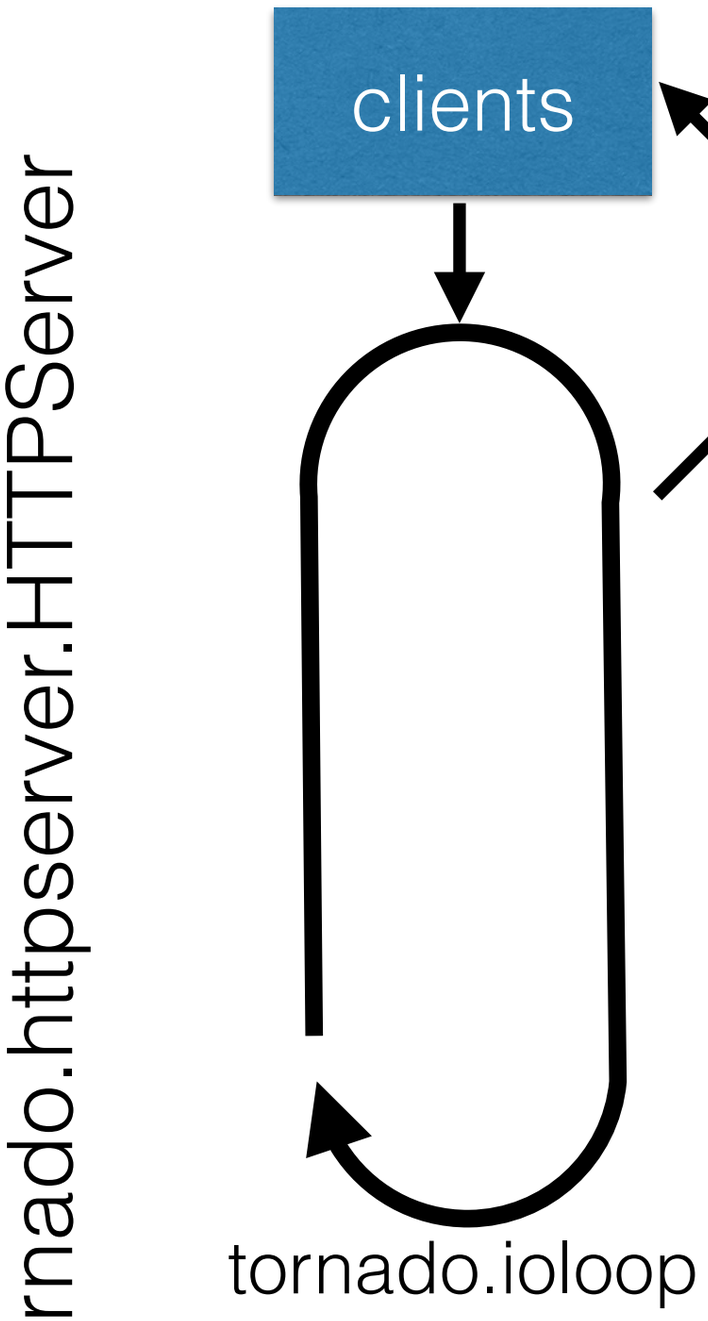
install tornado

- 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

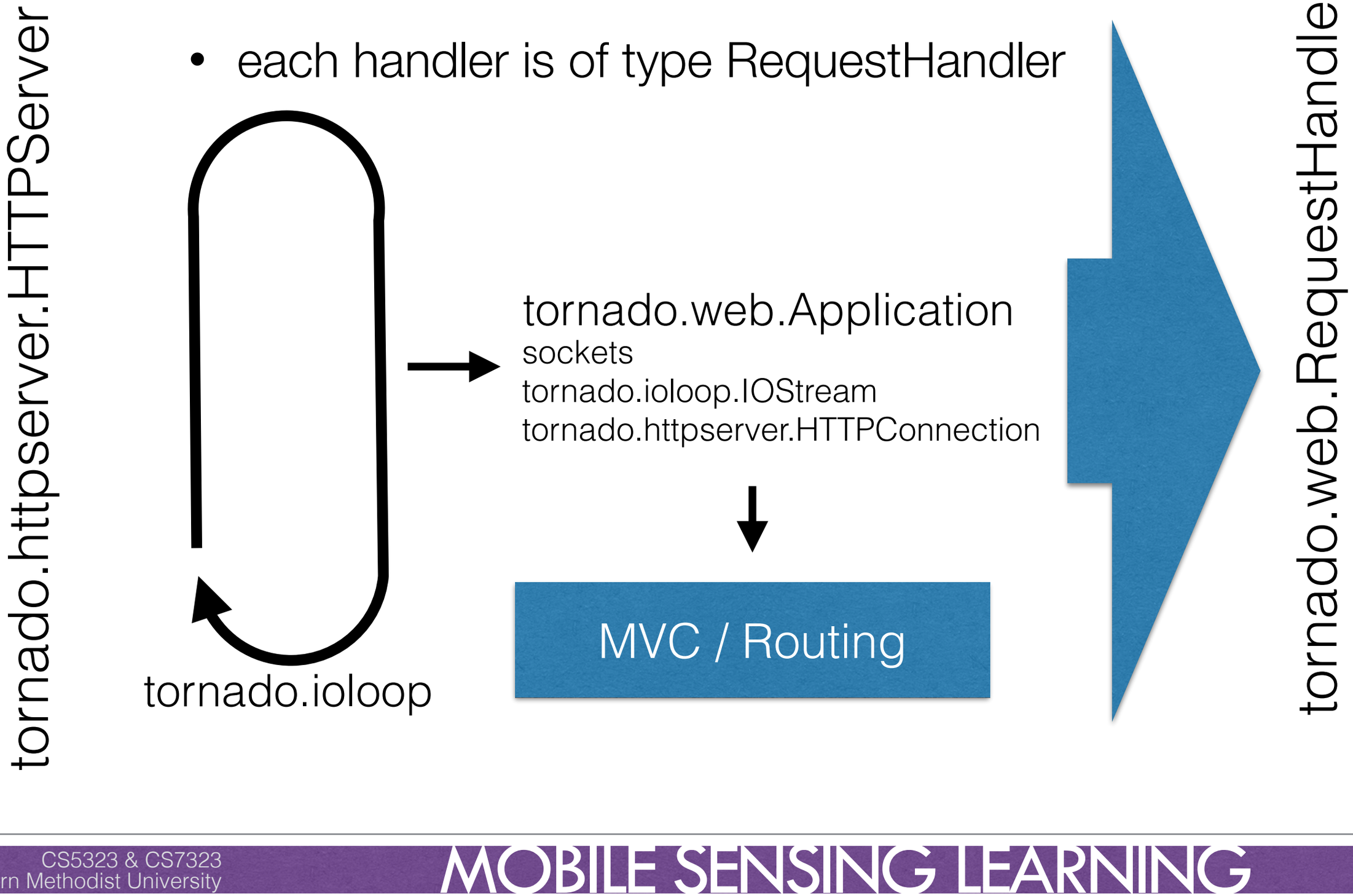
tornado



- 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
  - URL contains any na

```
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

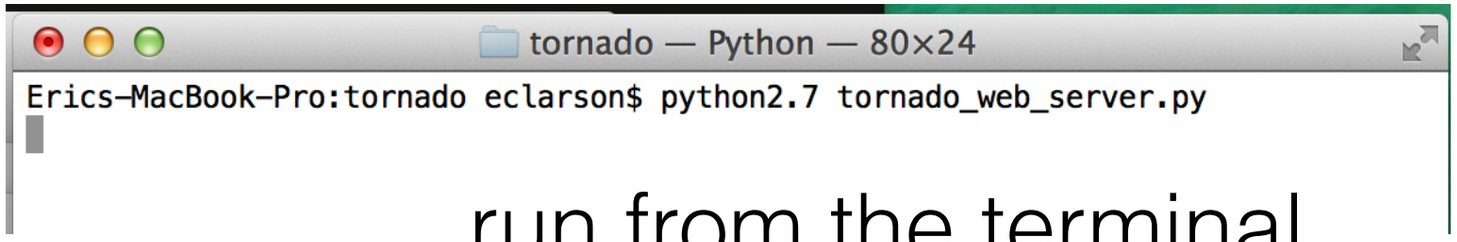
override get request handling

tuple with URL and handler

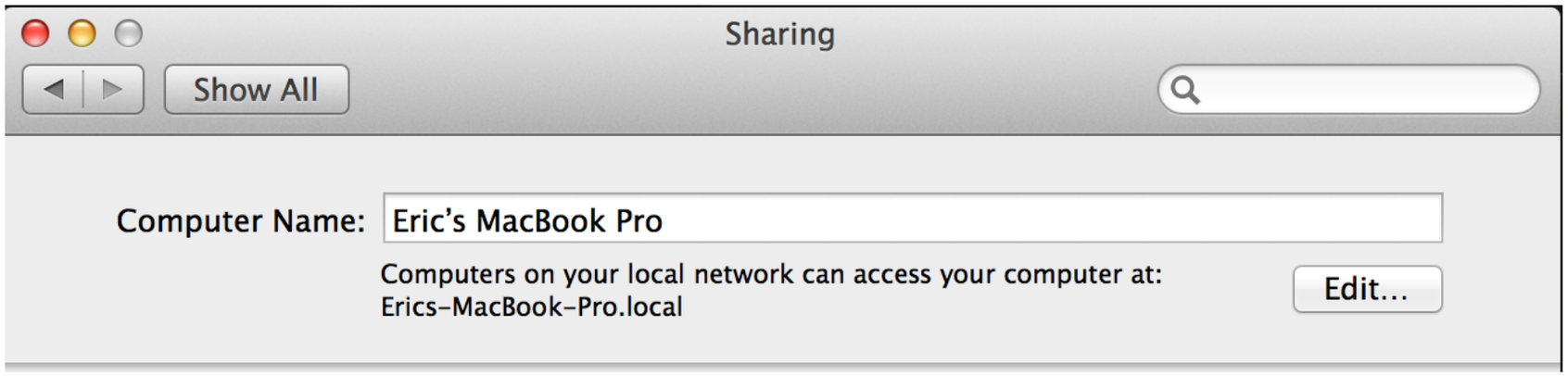
listen on 8888

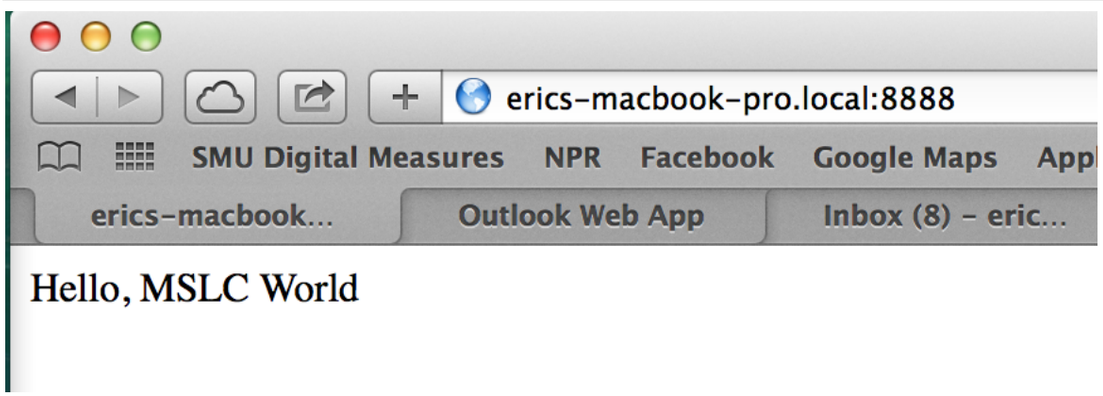
start the IO loop

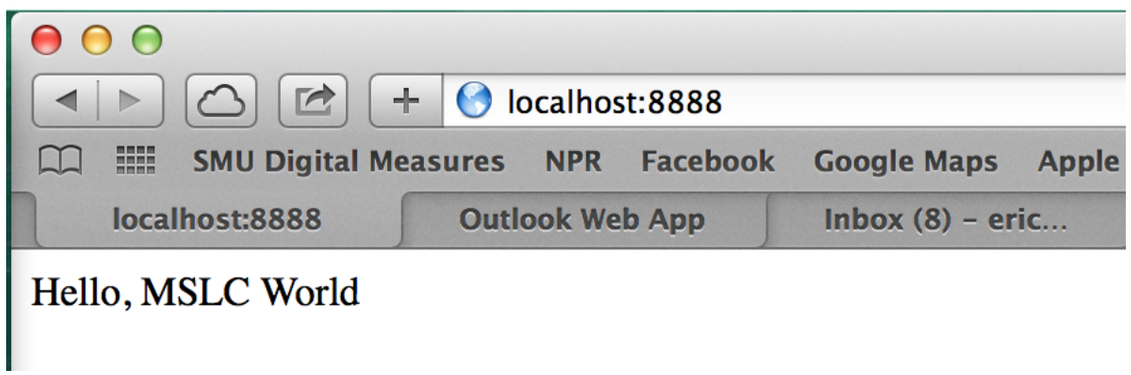
tornado example



run from the terminal





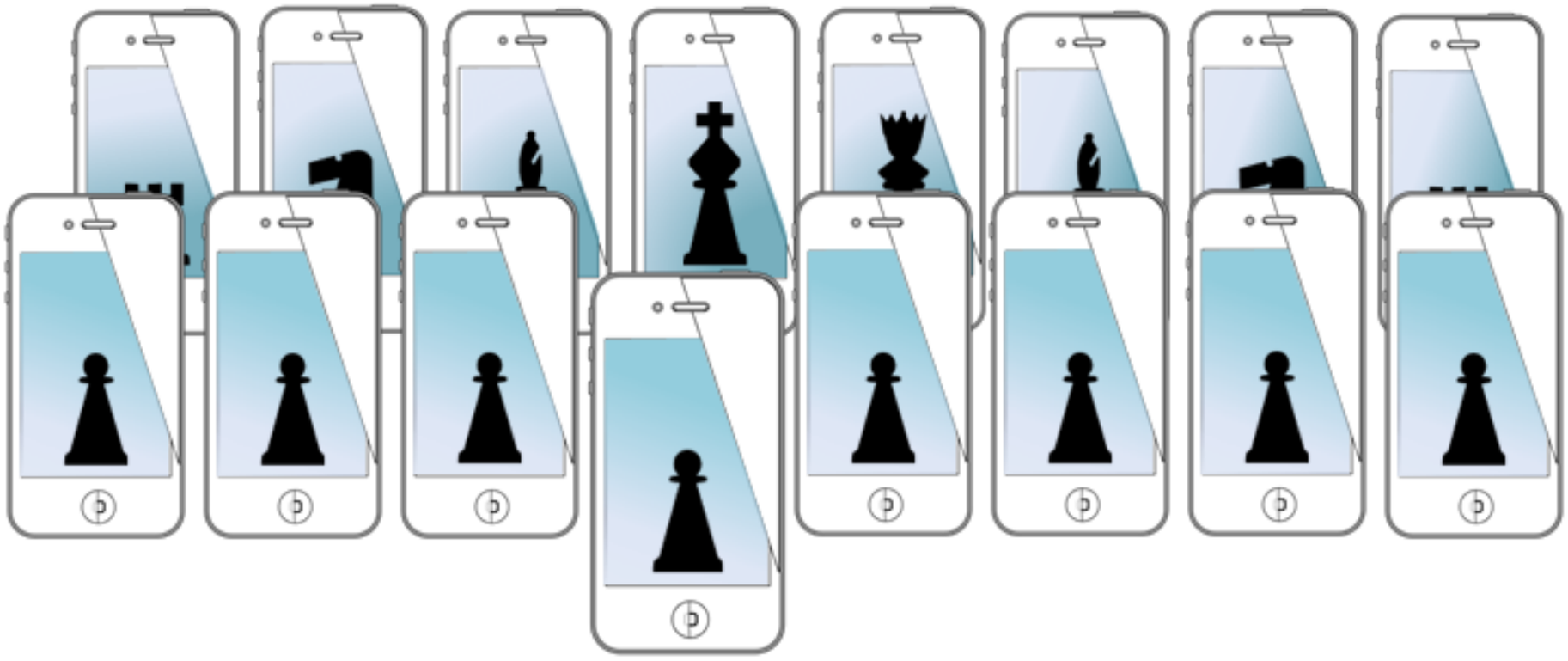


for next time…

- more python examples
- tornado
- pymongo
- in preparation for
  - proper http requests in iOS
  - numpy
  - turi-create

install these on your mac!

MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

tornado, pymongo, and http requests

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

course logistics/agenda

- start to think about the final project you want to propose
- agenda:
  - continue tornado
  - pymongo

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

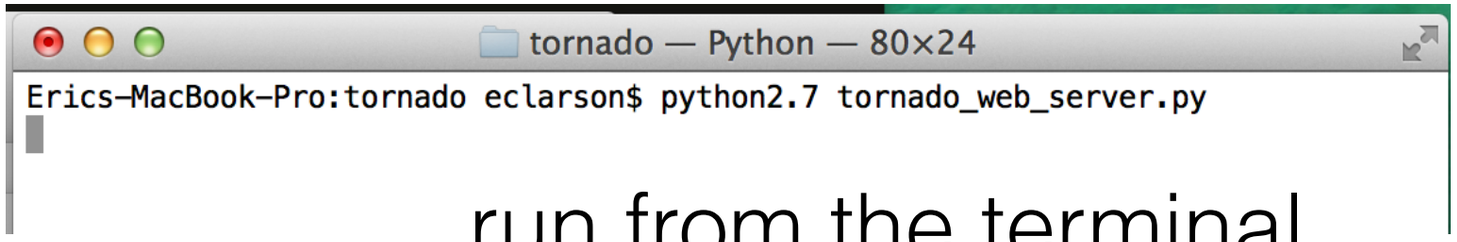
override get request handling

tuple with URL and handler

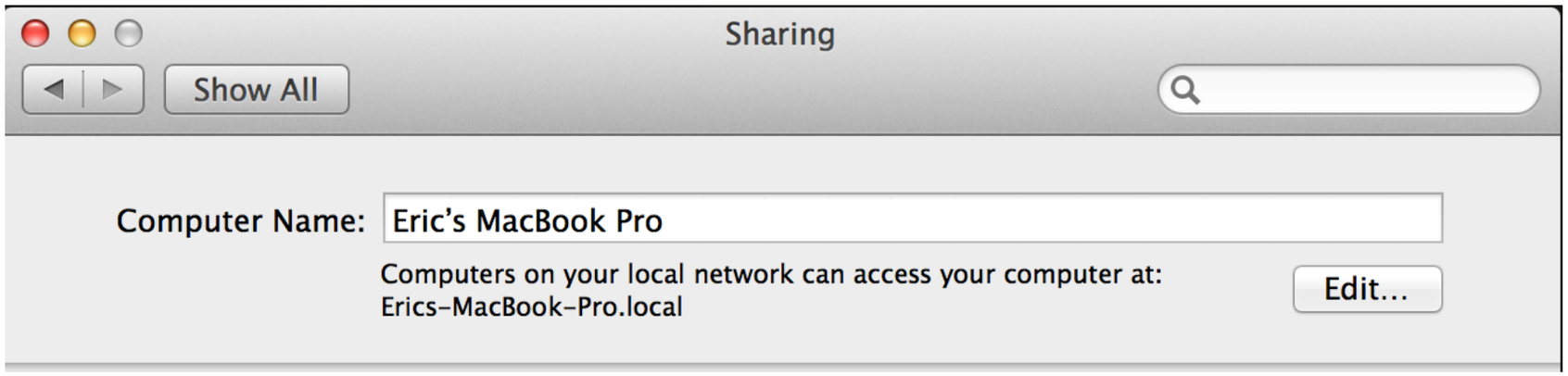
listen on 8888

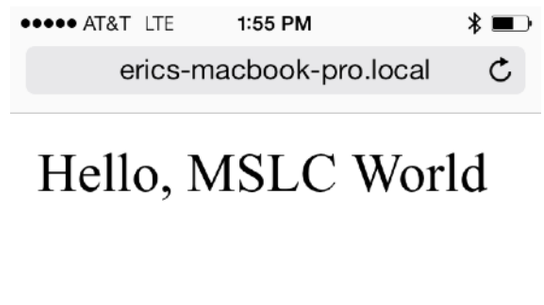
start the IO loop

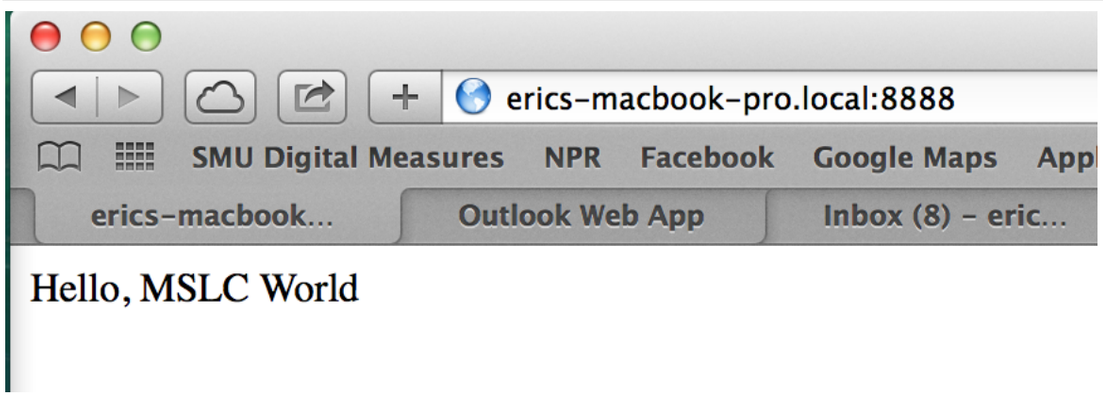
tornado example

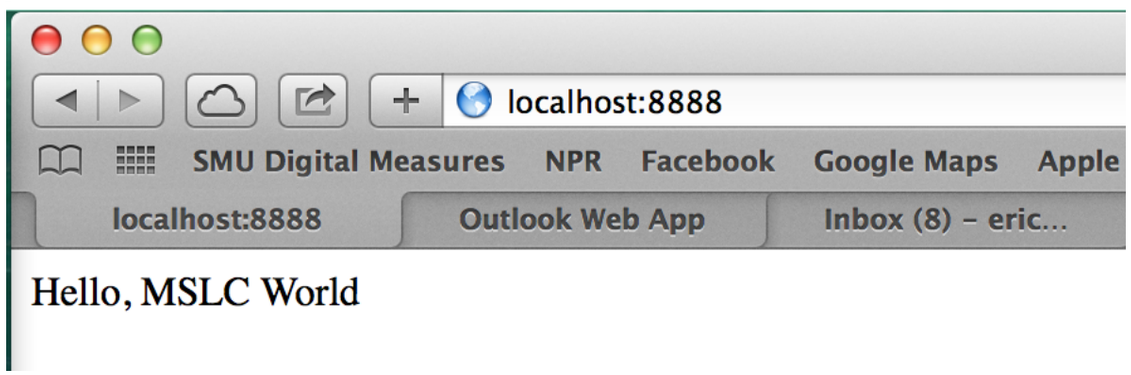


run from the terminal









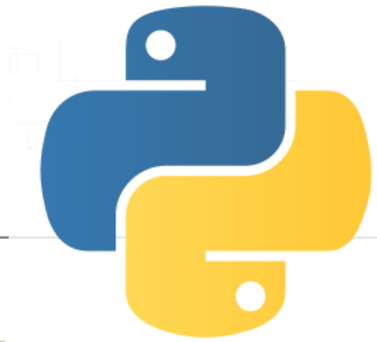
tornado

- 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



```
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

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

Run the blocking code on another thread. When asynchronous libraries are not available, `concurrent.futures.ThreadPoolExecutor` can be used to run any blocking code on another thread. This is a universal solution that can be used for any blocking function whether an asynchronous counterpart exists or not:

```
class ThreadPoolHandler(RequestHandler):
    async def get(self):
        for i in range(5):
            print(i)
            await IOLoop.current().run_in_executor(None, time.sleep, 1)
```

```
self.write(response.body.replace( <
```

allow return to IOLoop
if supported by function

https://www.tornadoweb.org/en/stable/faq.html

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", type=int)

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()
```

We need to write the Application class to meet desired functionality

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 written for you!

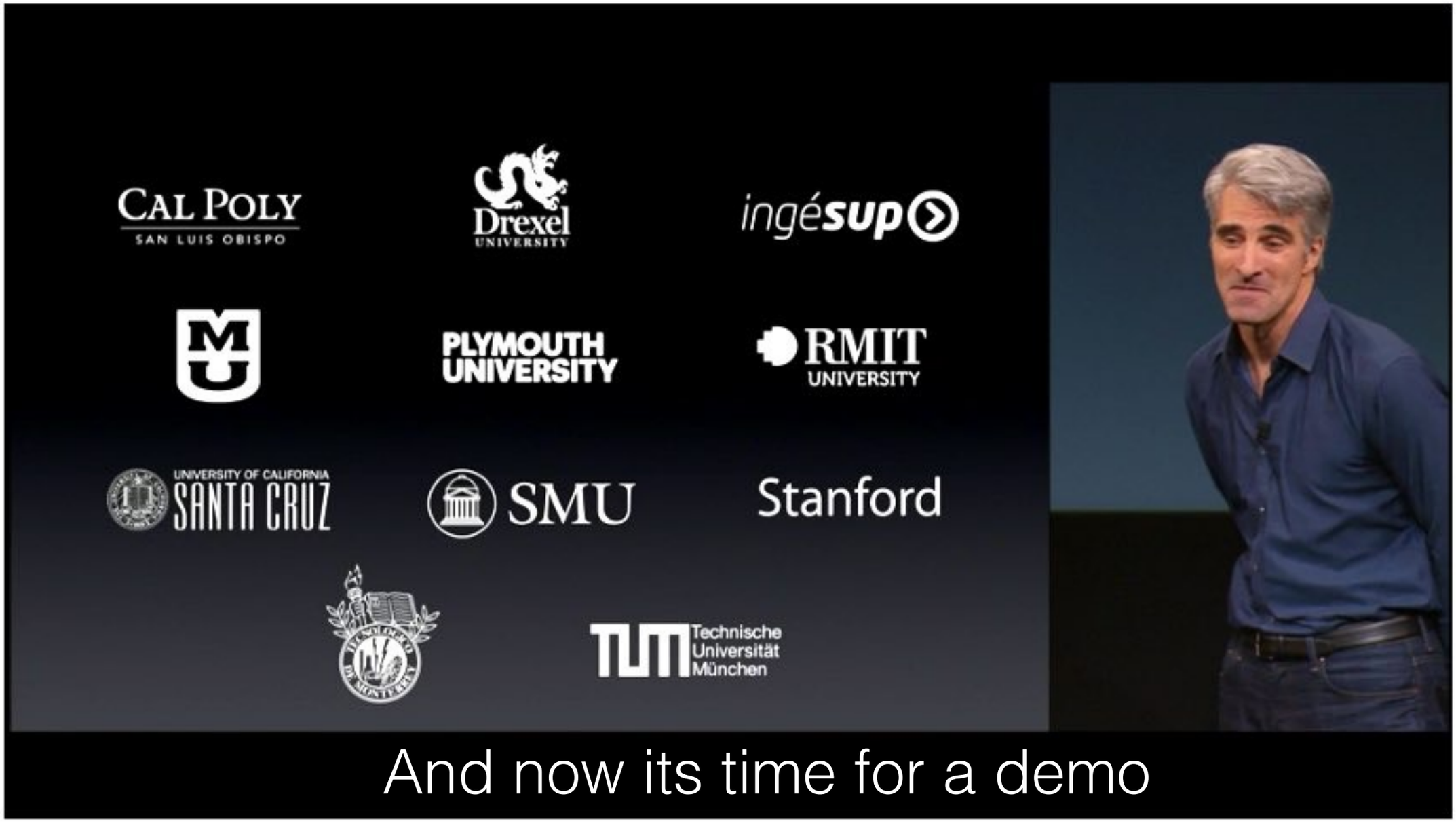
```
class PostHandler(BaseHandler):
  def get(self):
   '''respond with arg1*2
   '''
   arg1 = self.get_float_arg("arg1",default="none");
   self.write("Get from Post Handler? " + str(arg1*2));

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 written for you!

tornado examples

ifconfig | grep "inet "

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



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

- 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
- you can also **run as a service** (`./mongo`)

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)
  - and asynchronous calls (decorators or async/await)

```
import motor
db = motor.MotorClient().test

@gen.coroutine
def loop_example(collection):
    cursor = db.collection.find()
    while (yield cursor.fetch_next):
        doc = cursor.next_object()
```

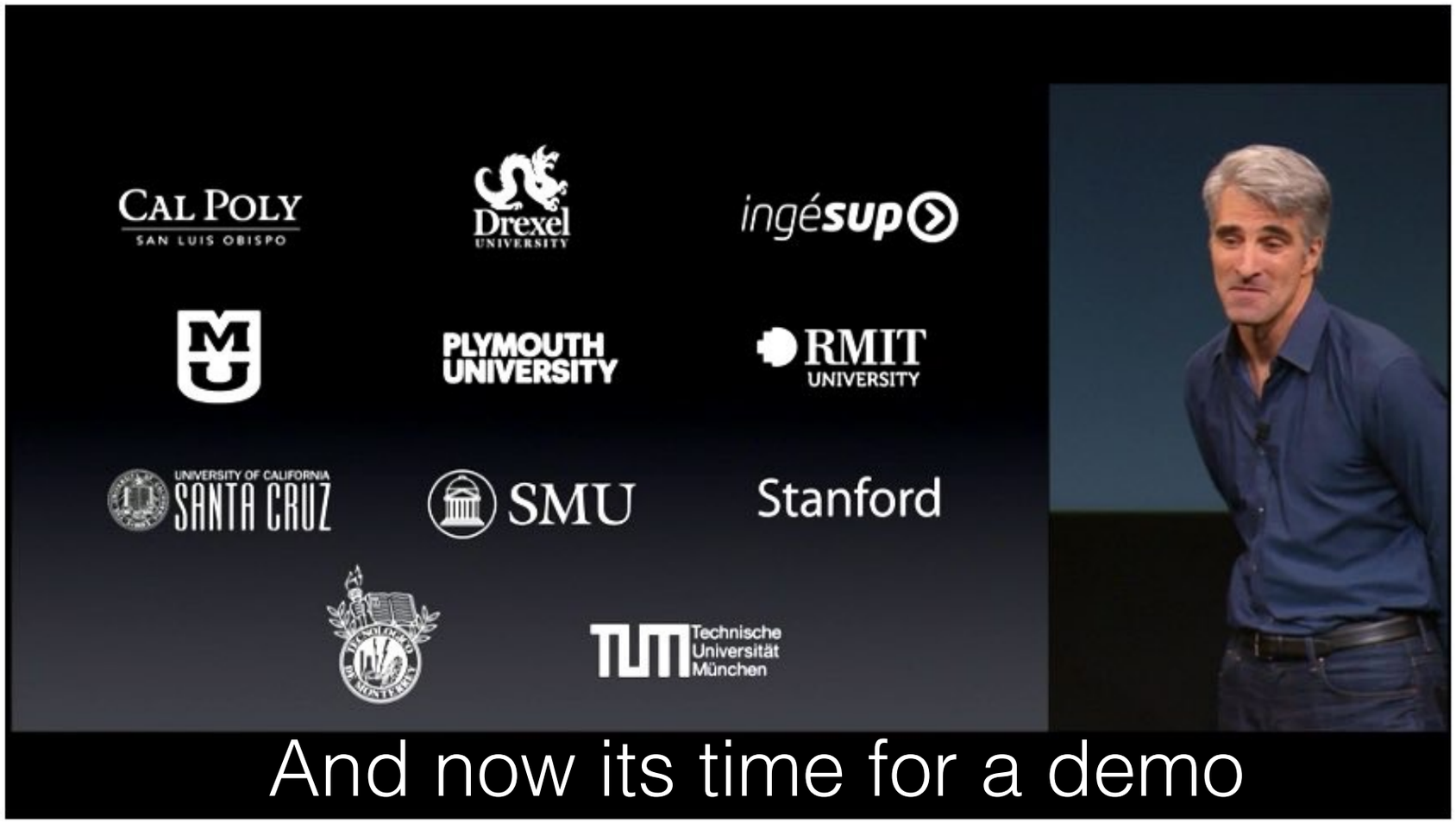
the motor package
is good for this!

https://motor.readthedocs.io/en/stable/

mongodb + tornado

ifconfig | grep "inet "

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



and add ***something*** to it