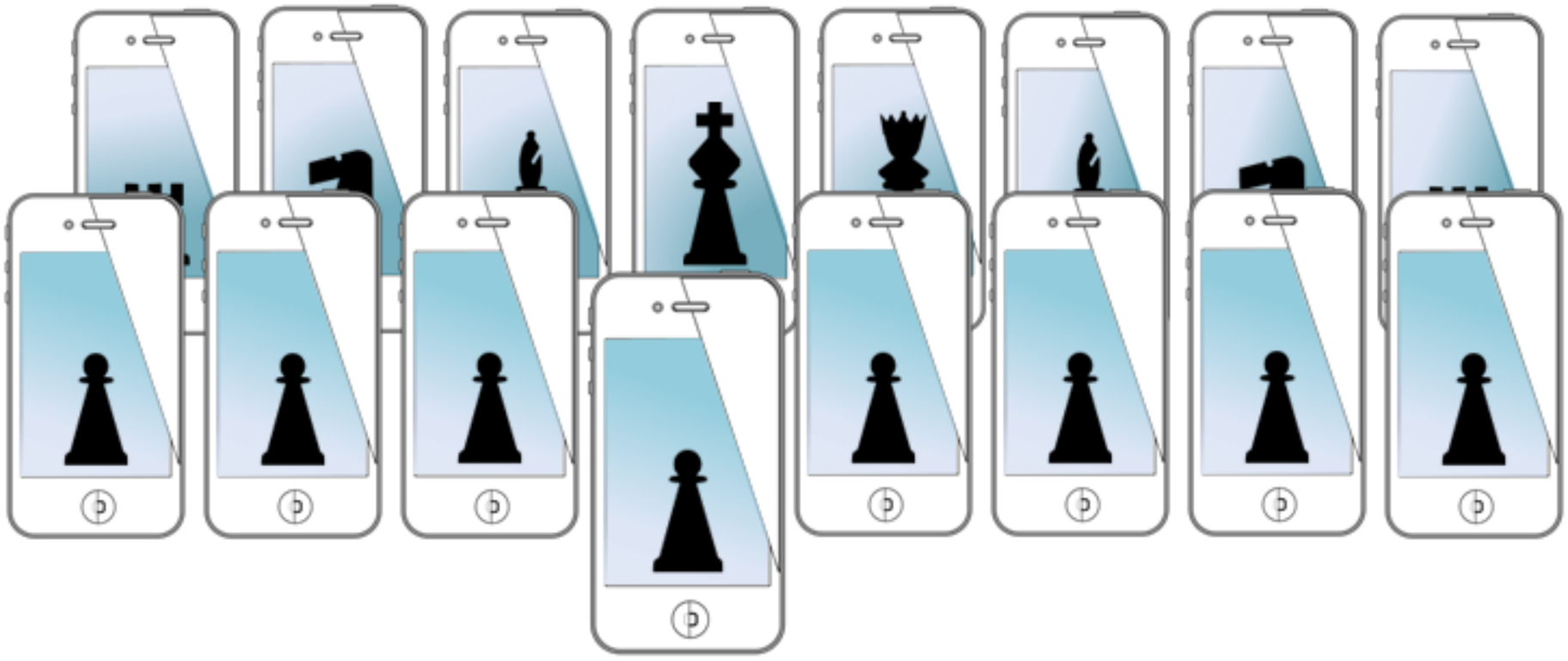
MOBILE SENSING LEARNING



CSE5323 & 7323

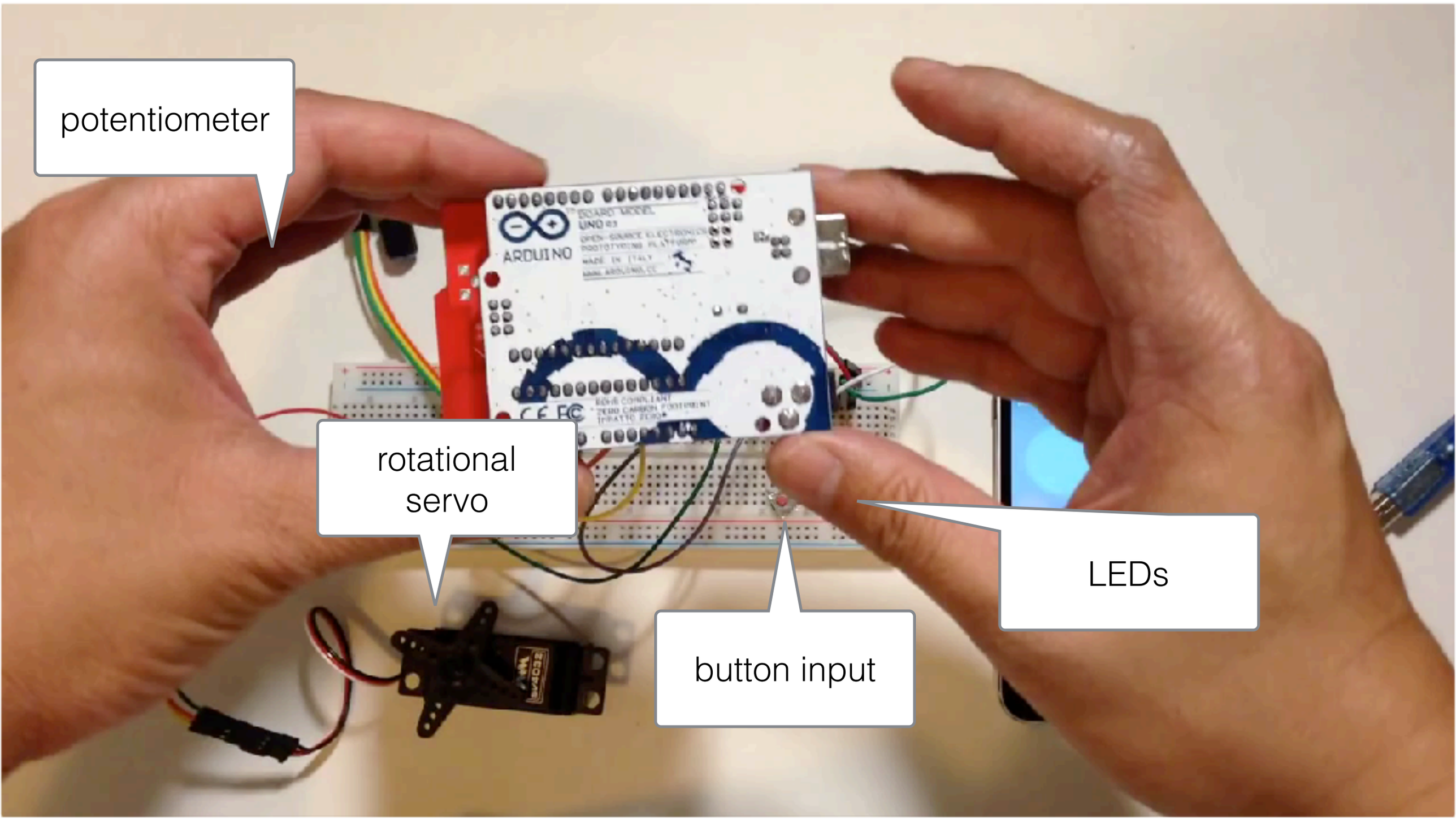
Mobile Sensing and Learning

Video Lecture: control and bluetooth

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

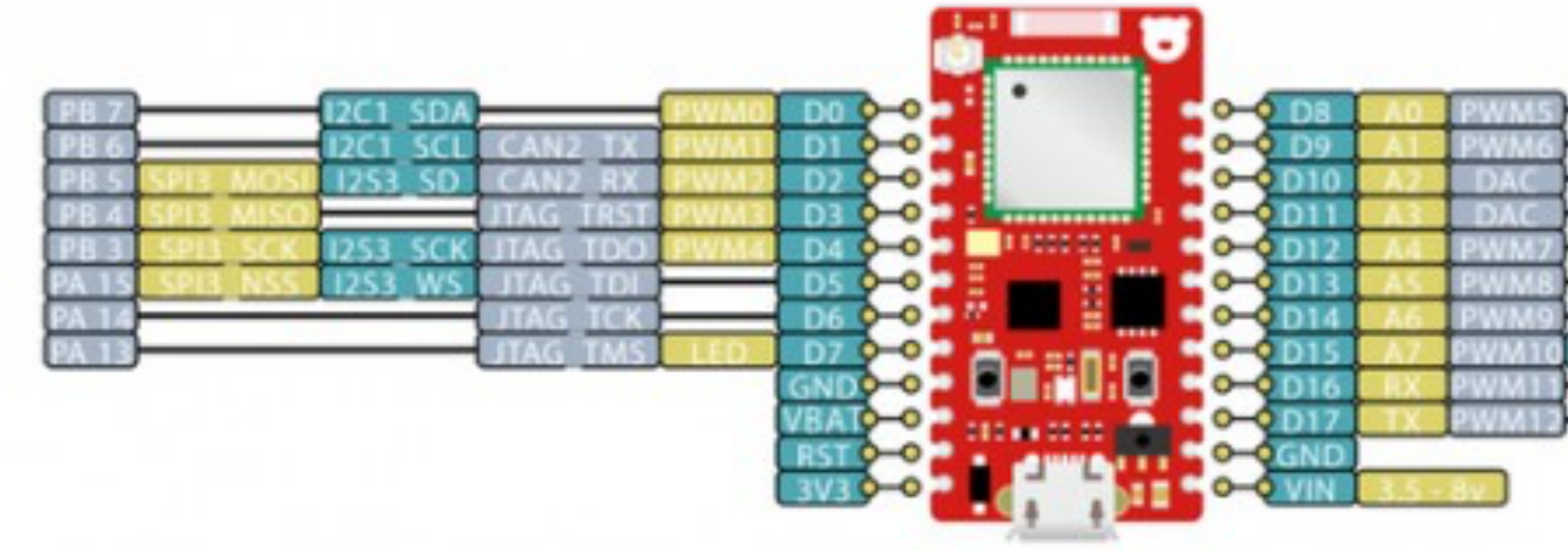
video agenda

- bluetooth communication
  - using microcontrollers
  - using core bluetooth on iOS


potentiometer
rotational servo
button input
LEDs

BLE +WIFi Duo

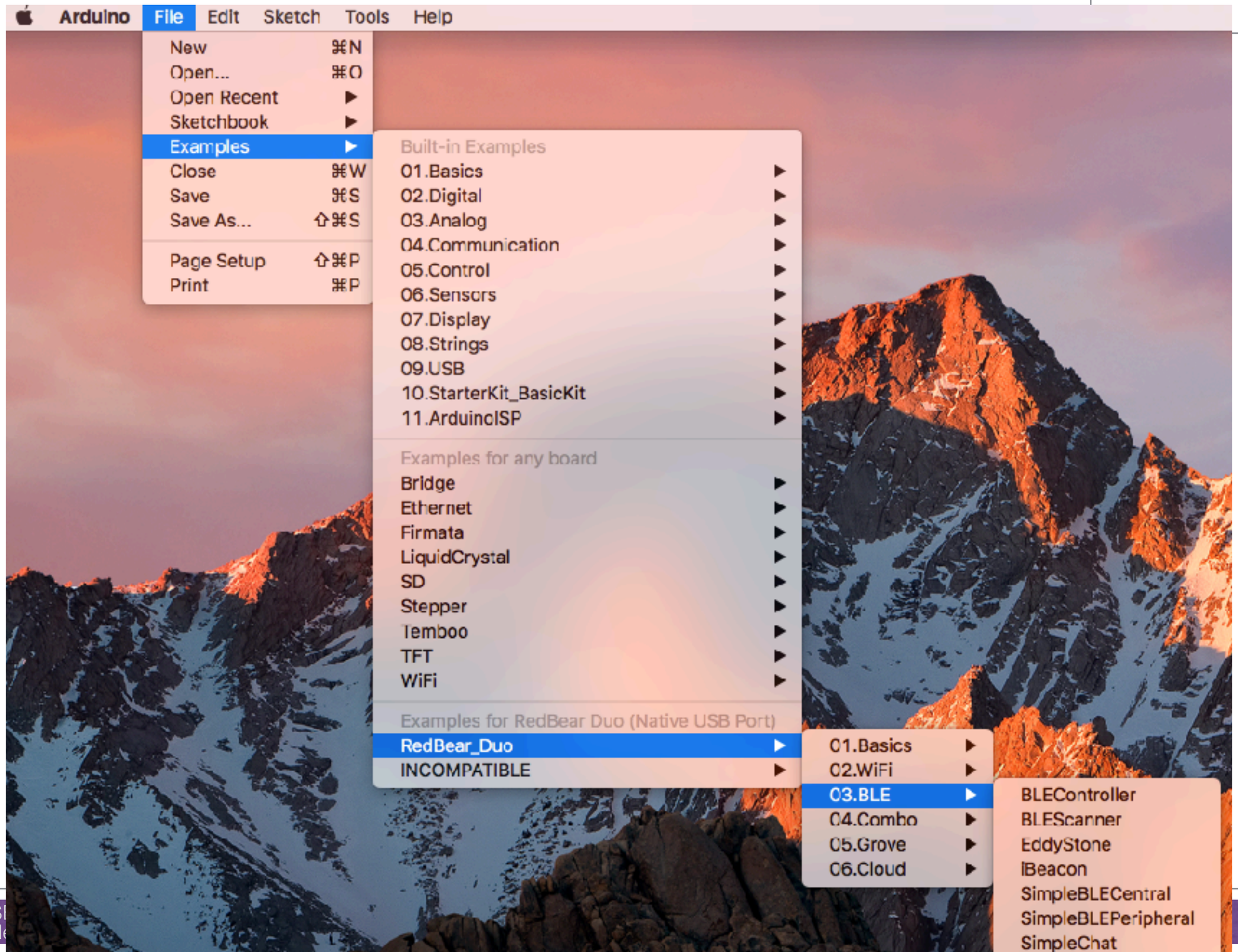
- BLE/WiFi made really easy
- takes care of all the protocol of BLE broadcasting



installing library

- instructions available at:
- https://github.com/redbear/Duo

BLE library examples



bluetooth primer

- traditional bluetooth
  - short range wireless
  - low-medium latency
  - good for data like voice and audio
  - fairly power hungry
    - connection oriented
    - connection is maintained even when no data
- one million symbols per second
- but… cannot run from **coin cell battery** and limited to **seven** concurrent connections

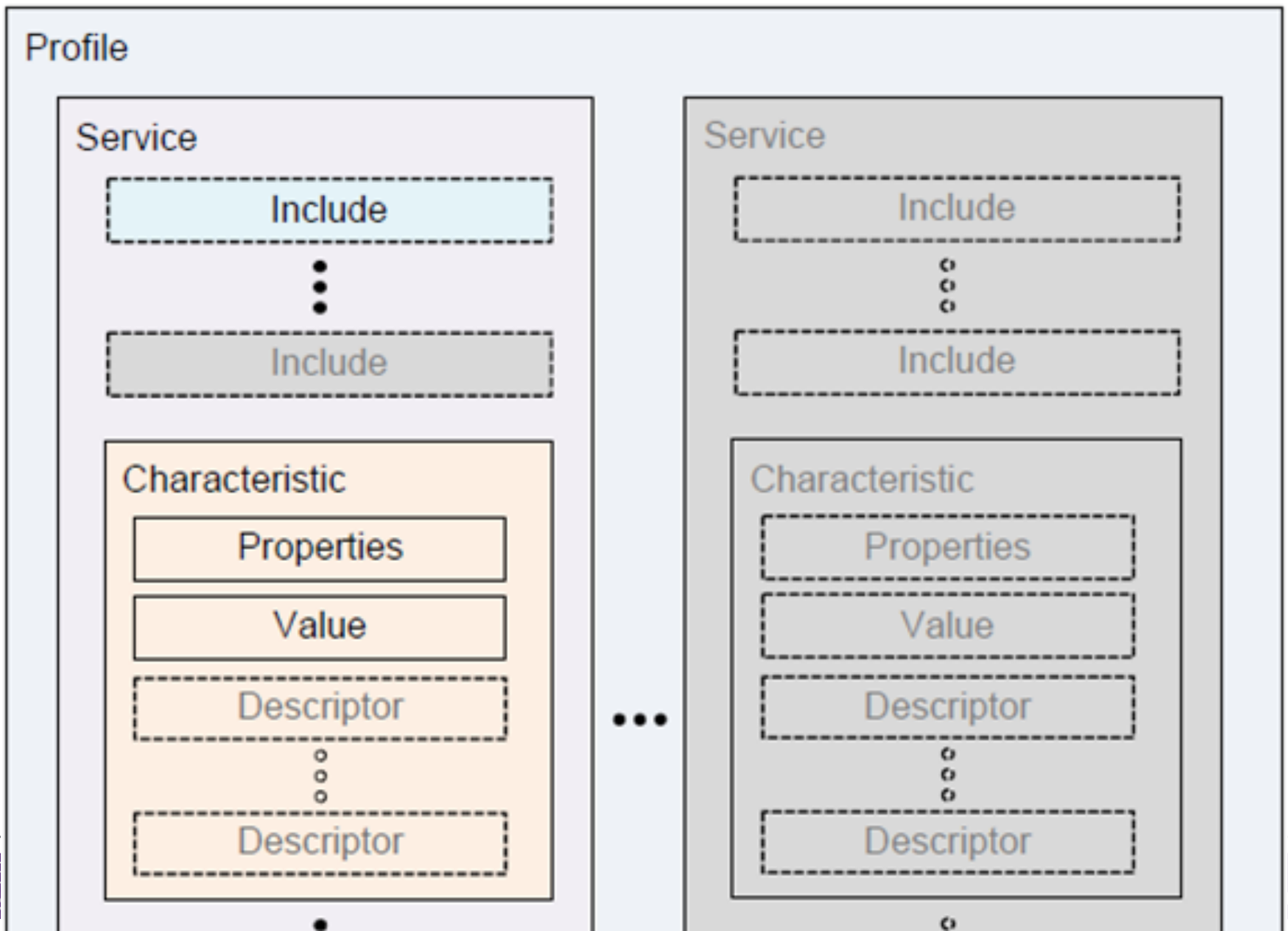
bluetooth low-energy BLE

- designed for the internet-of-things
- asynchronous client / server model
- low latency
  - ~3 ms start to finish
- optimized for short bursts of information
  - so not really audio, but that gets abused
- number of connections
  - greater than two billion

theoretically

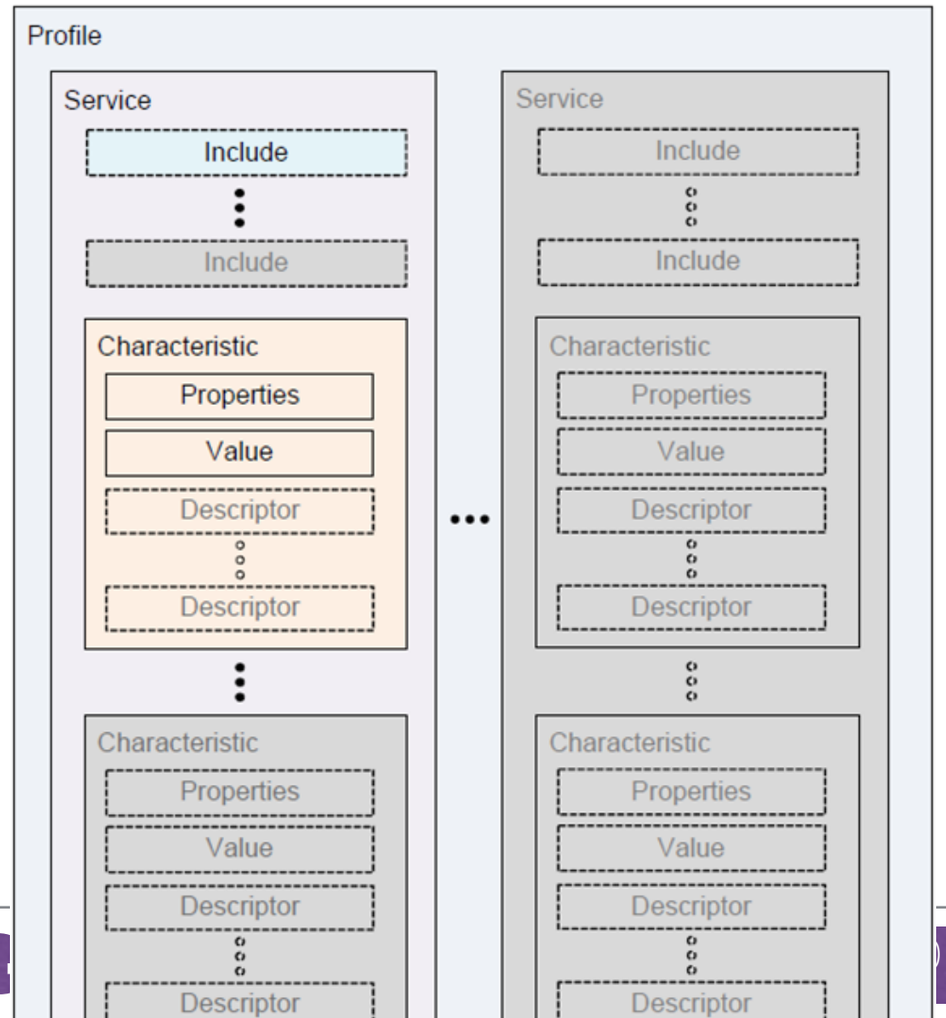
bluetooth low-energy BLE

- data transfer can be triggered by events
- read at any time by a client or server
- interface protocol is Generic Attribute (GATT) Protocol



GATT protocol

- top level: profile that includes services the device uses
- each service: composed of references and characteristics
- each characteristic:
  - type (universally unique ID (UUID, 128 bit)
  - value (typically data)
  - properties



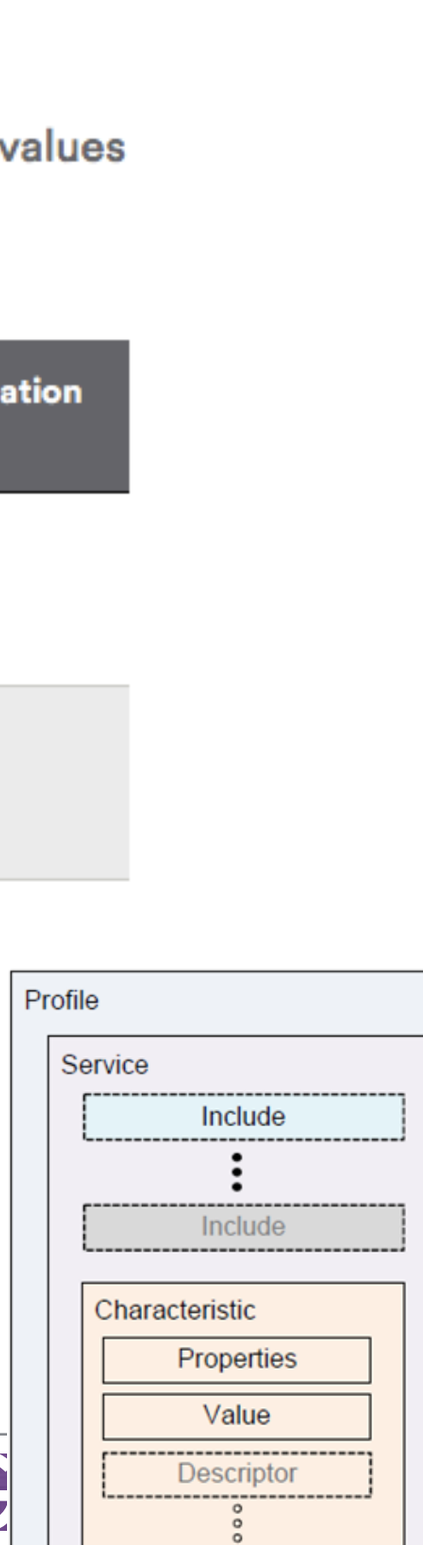
https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

GATT protocol

GATT Characteristics

Characteristics are defined attribute types that contain a single logical value. All Assigned Numbers values on this page are normative.

Name	Uniform Type Identifier	Assigned Number	Specification
Aerobic Heart Rate Lower Limit	org.bluetooth.characteristic.aerobic_heart_rate_lower_limit	0x2A7E	GCD
Aerobic Heart Rate Upper Limit	org.bluetooth.characteristic.aerobic_heart_rate_upper_limit	0x2A84	GCD



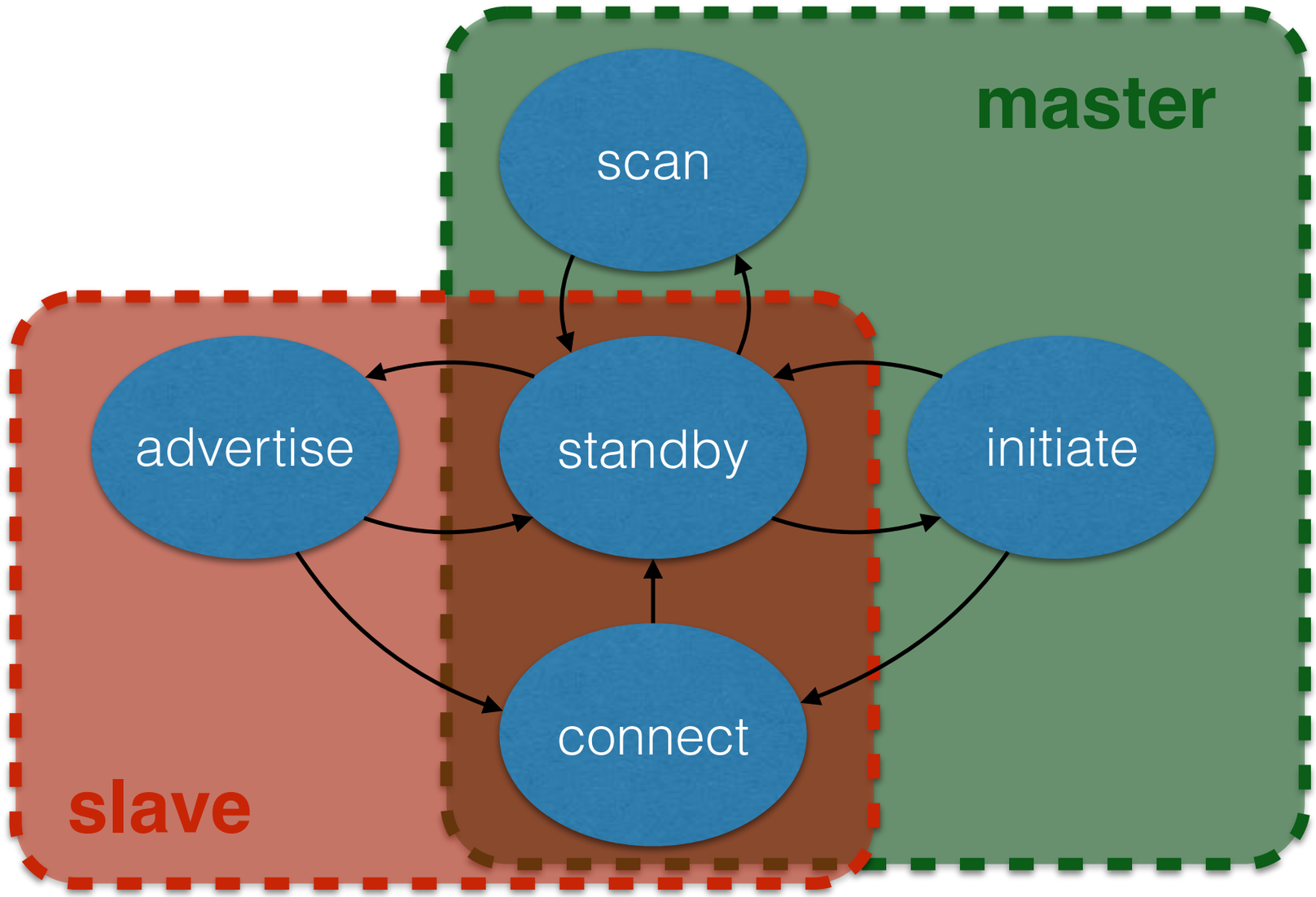
https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

code review demo

- define services
- setup characteristics supported for each service
- setup callbacks for transmit
- setup callbacks for receive
- loop()

BLE connections

- 3 bands for advertising
- 37 bands for data transmission



who's who?

peripheral

slave

advertise

central

master

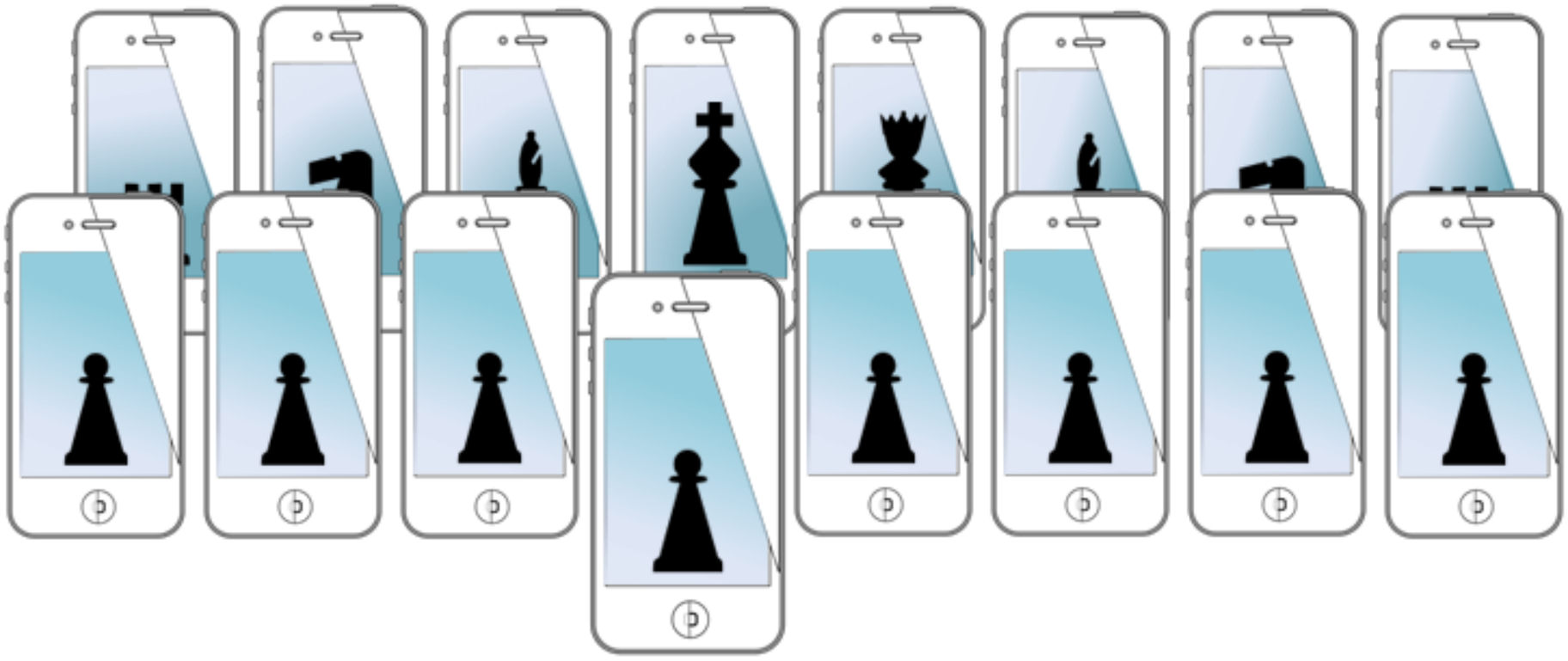
initiate

BLE in iOS

- we will use the red bear API
- instantiate BLE object
- connect to advertisers
- use delegation for responding to incoming data
- add as framework to project and that's it
- download from their github page
- or checkout the example repository from class GitHub page

iOS demo time

MOBILE SENSING LEARNING



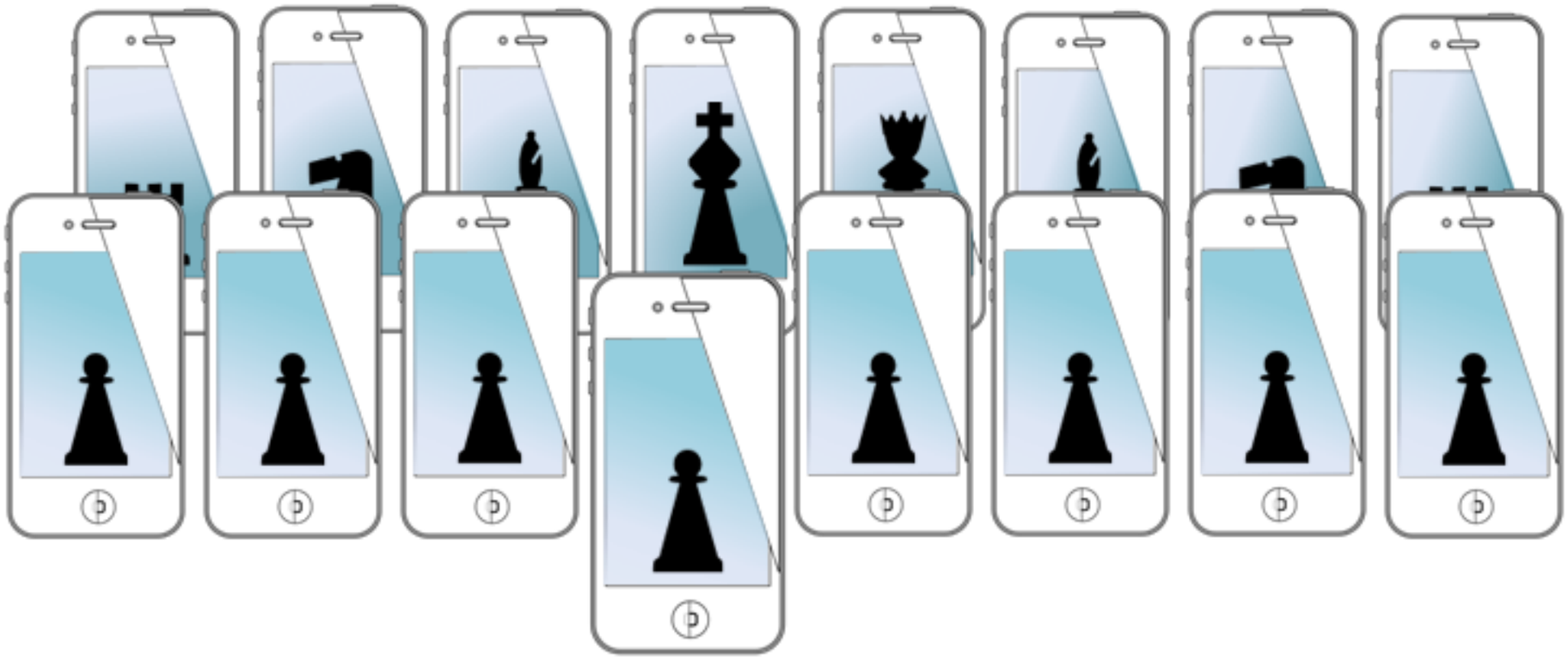
CSE5323 & 7323

Mobile Sensing and Learning

Video Lecture: control and bluetooth

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

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

week nine: python crash-course, tornado

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

course logistics

- A5 is due next week
  - sensors okay?
  - other issues?

assignment 5

- Reads and displays data from two or more sensors/hardware attached to the arduino
  - one sensor/input must use analog voltage (e.g., as simple as a potentiometer)
  - the other inputs(s) can be analog, digital, or binary output (e.g., as simple as a button)
  - the display of the sensor data should be more than just a text label
  - Create a protocol for encoding and interpretting the data on the iPhone
- Sends two or more control commands to the microcontroller that change the behavior of the arduino (i.e., the arduino interpretes the commands and changes something)
  - make the controls change something noticeable in the operation of the Arduino
  - the output must make use of a PWM signal (implemented in hardware)
- The Arduino sketch should also:
  - use one or more interrupts
    - for example, use a button as input
    - the interrupt must change something noticeable in the operation of the Arduino
    - use proper debouncing
  - use digital outputs (GPIO)
    - for example to control LED(s), pin 13 is already setup for this
  - use PWM (in hardware, look at which GPIO pins can do this)
  - use the ADC (for the analog input part)

agenda

- history of python
- syntax
  - pythonic conventions
  - simple examples
- if time:
  - web handling
  - document databases

python

- Guido van Rossum

From wikipedia:

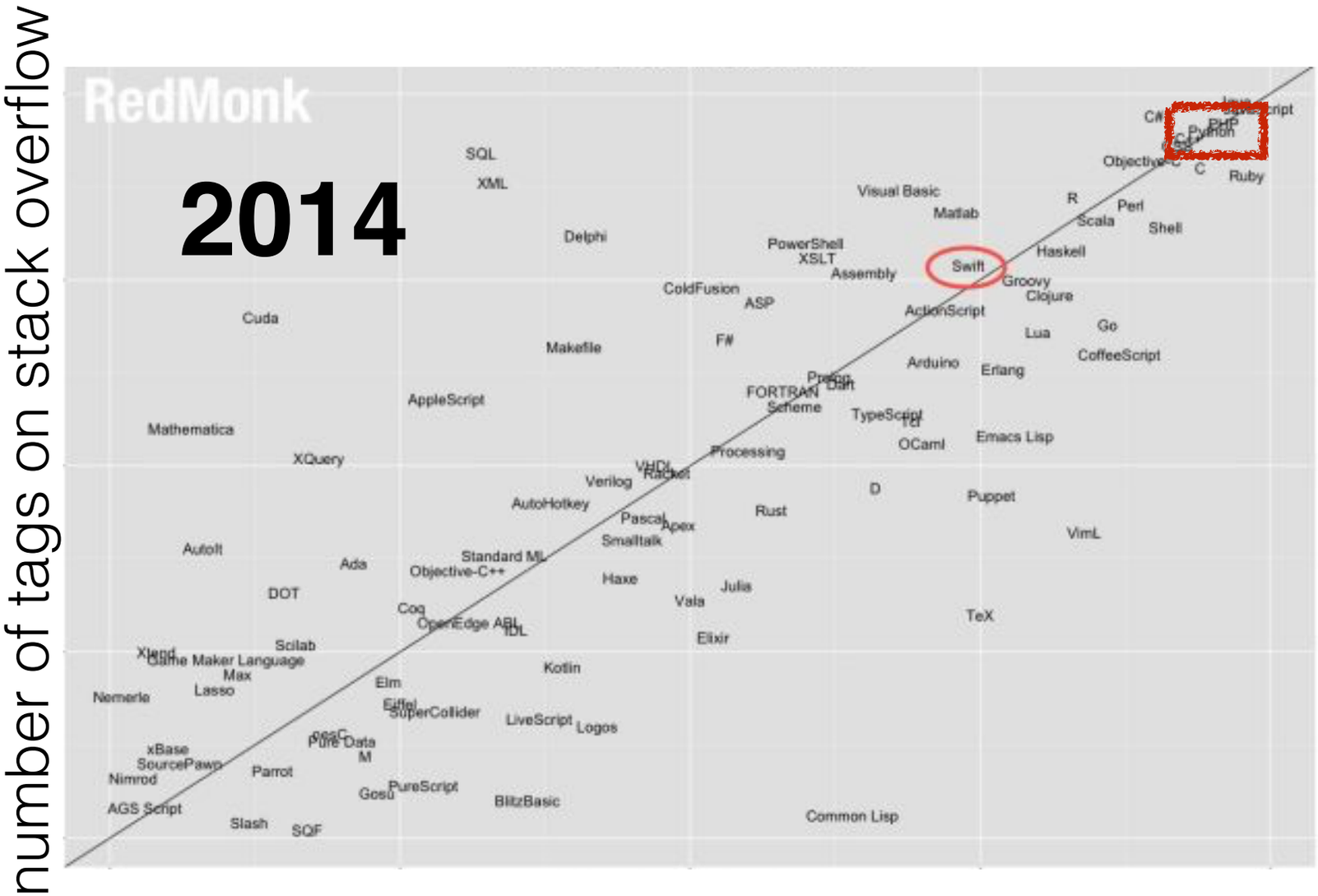
Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language 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

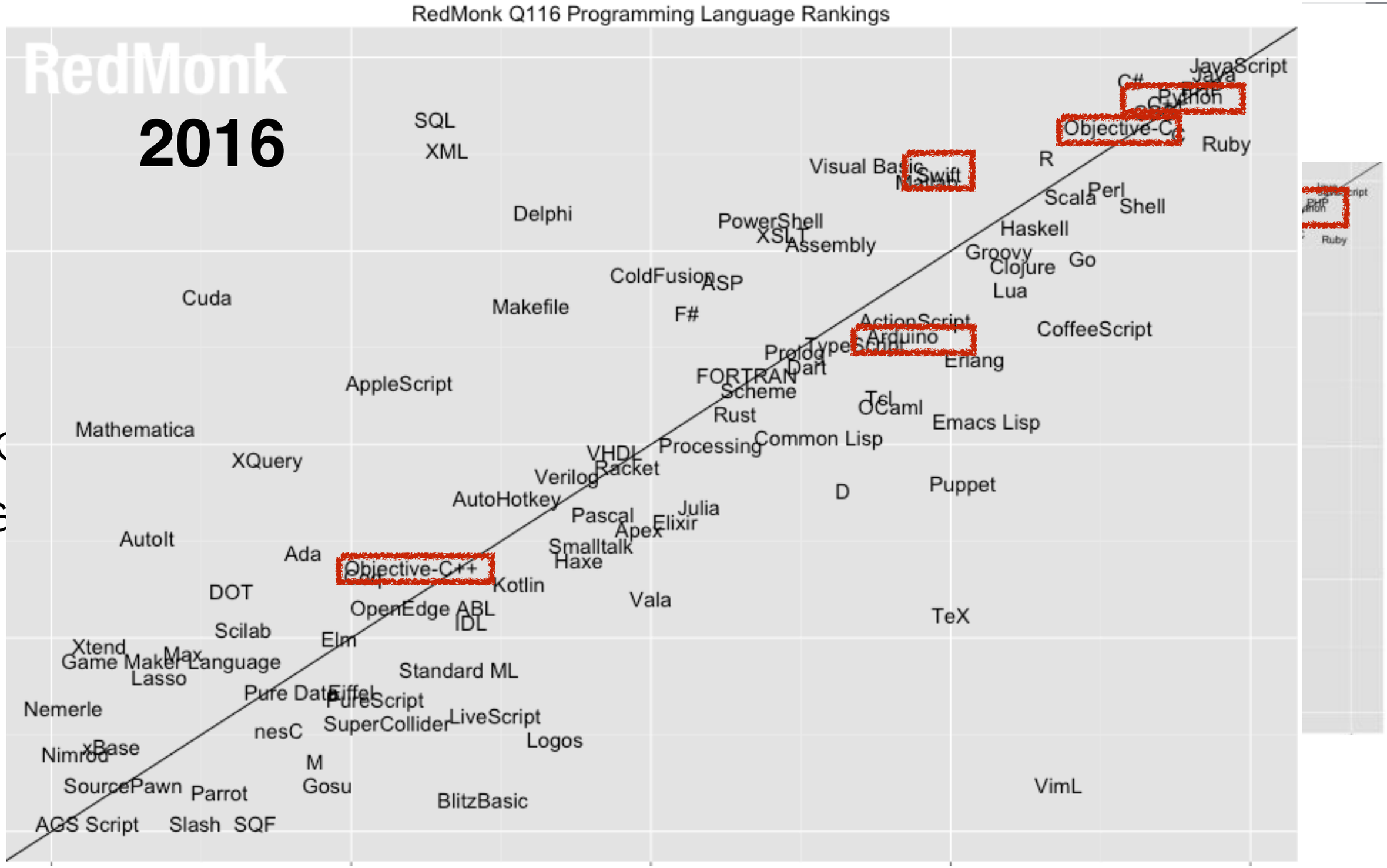
- appears in every programming top ten list

you (probably) should know it!



python adoption

- **2016**



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
  - unless you use IPython
- more similar to swift than you know
  - kinda
- 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 2.7.9, updated in January)
  - unified types, made completely object oriented
- 3.0 (up to 3.4.3, updated in February)
  - eliminate multiple paradigms (kinda)
  - 2.x not necessarily compatible with 3.x

installation

- install anaconda
- use python 3.5
- use conda environments

python

- hello world!
- from a script and from an interpreter

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
In the face of ambiguity                                    ss.
There should be one-- ar                                    s way to do it.
Although that way may no                                    ou're Dutch.
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

```
>>> 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

- 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
```

small integers are cached
strings behave the same

```
a=5
b=4+1

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

```
Everybody is a five!
```

conditionals

```
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(Heart,self).__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 syntax “with”

- 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 generators

- kinda like static variables
- functions used to iterate through some process
- lots more that you can do, like send in values

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


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

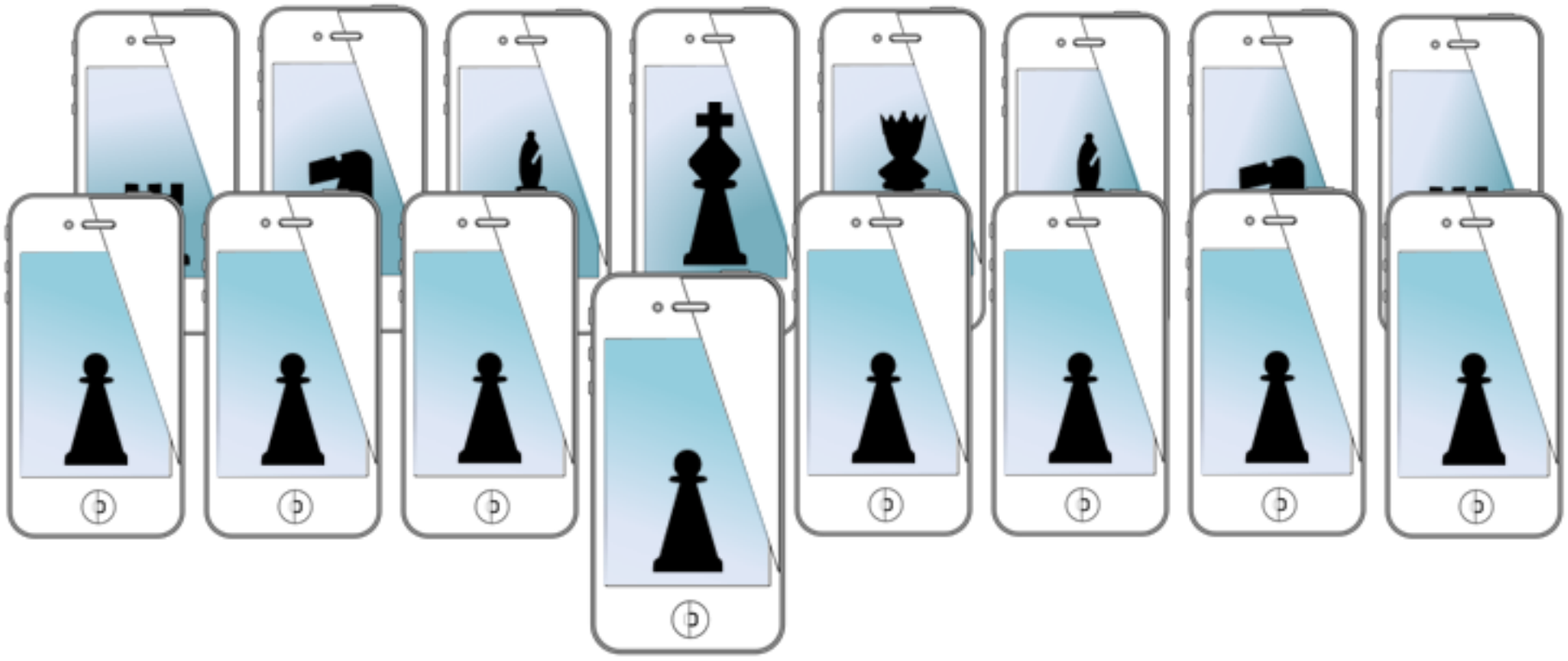
python decorators

- 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

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

week nine: tornado and pymongo

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

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

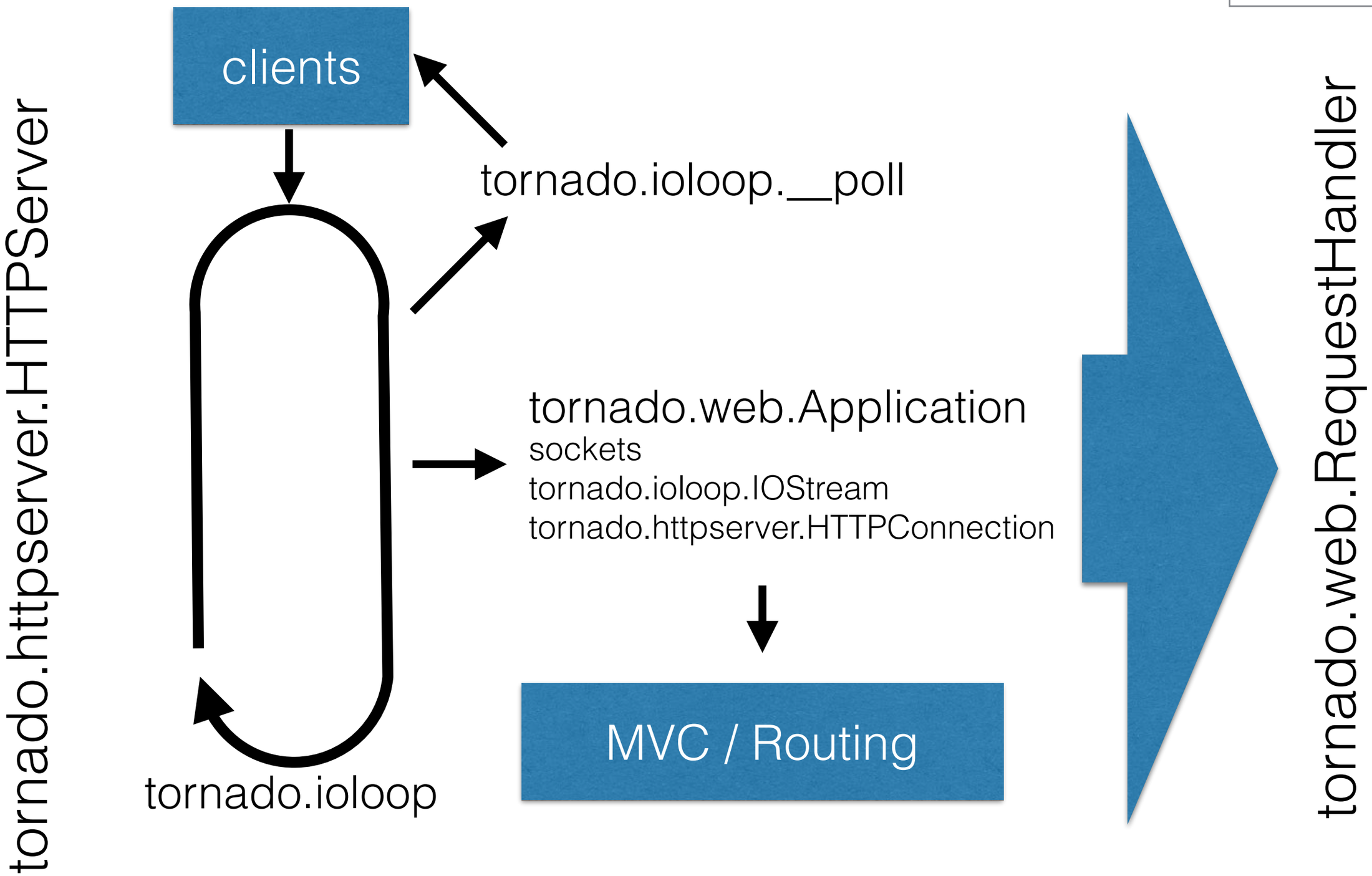
and this!
http://www.slideshare.net/gavinmroy/an-introduction-to-tornado?related=1

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

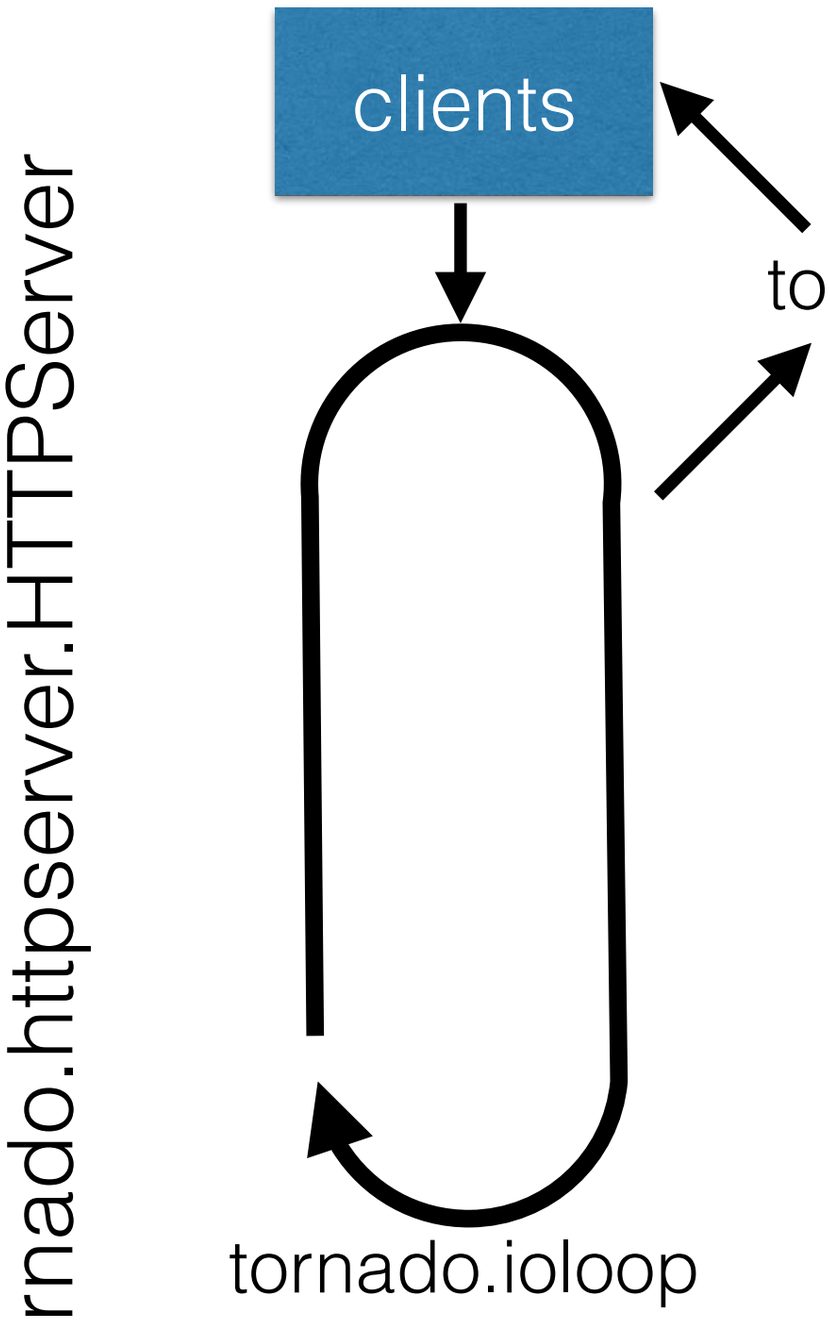
install tornado

- 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

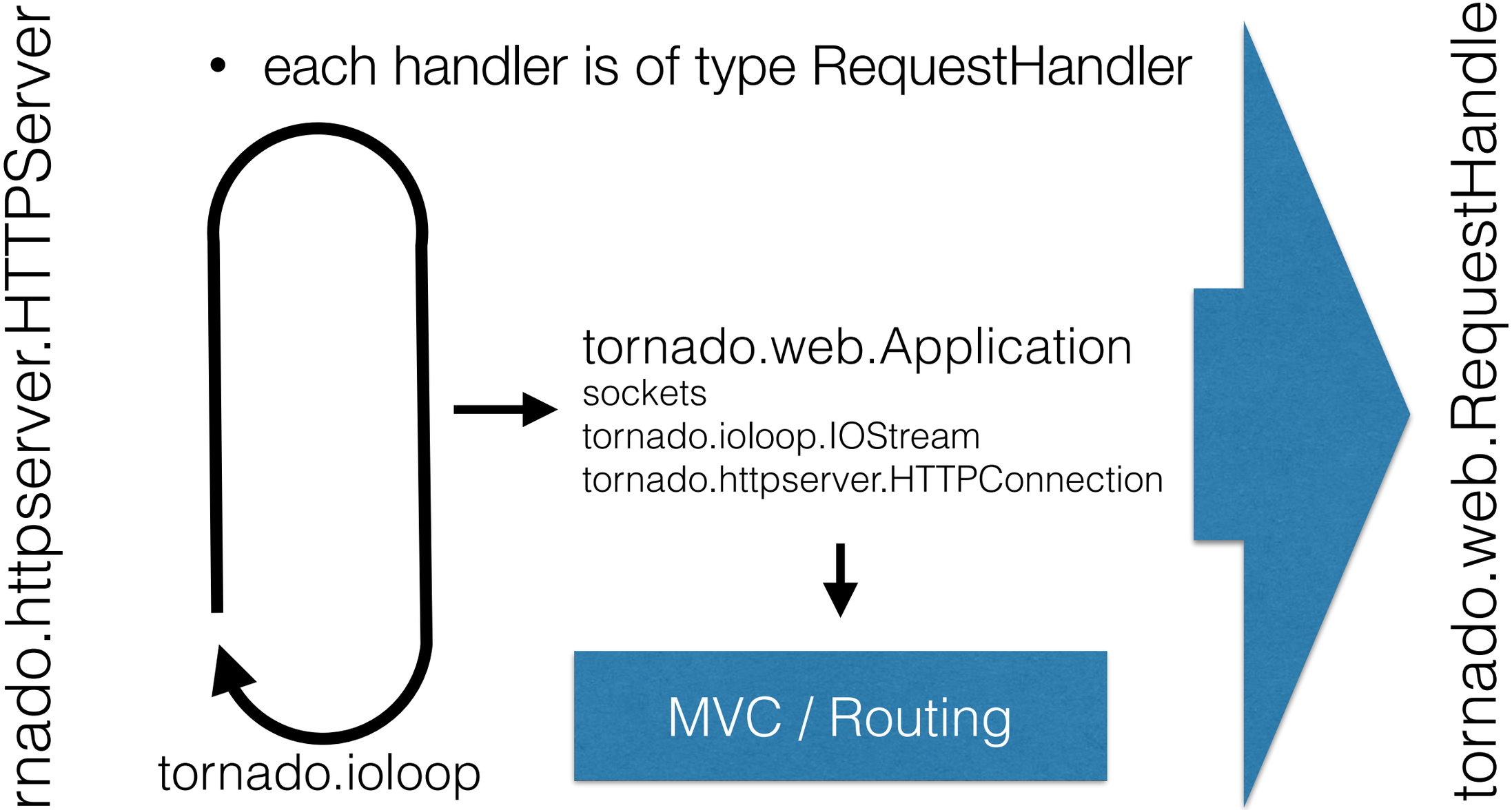
tornado



- 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

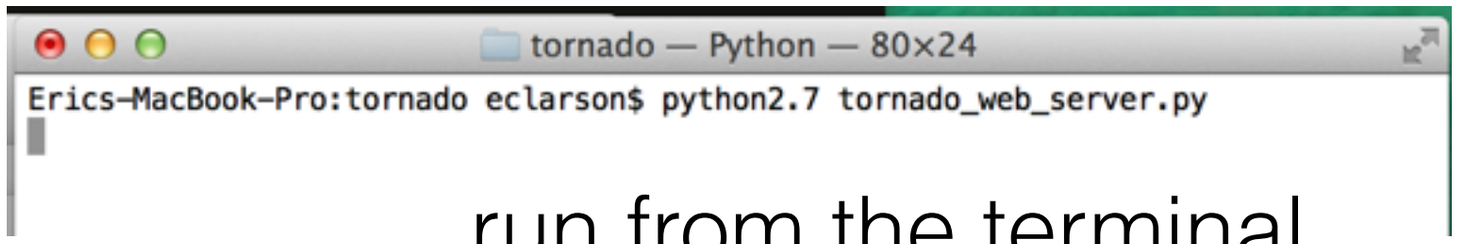
override get request handling

tuple with URL and handler

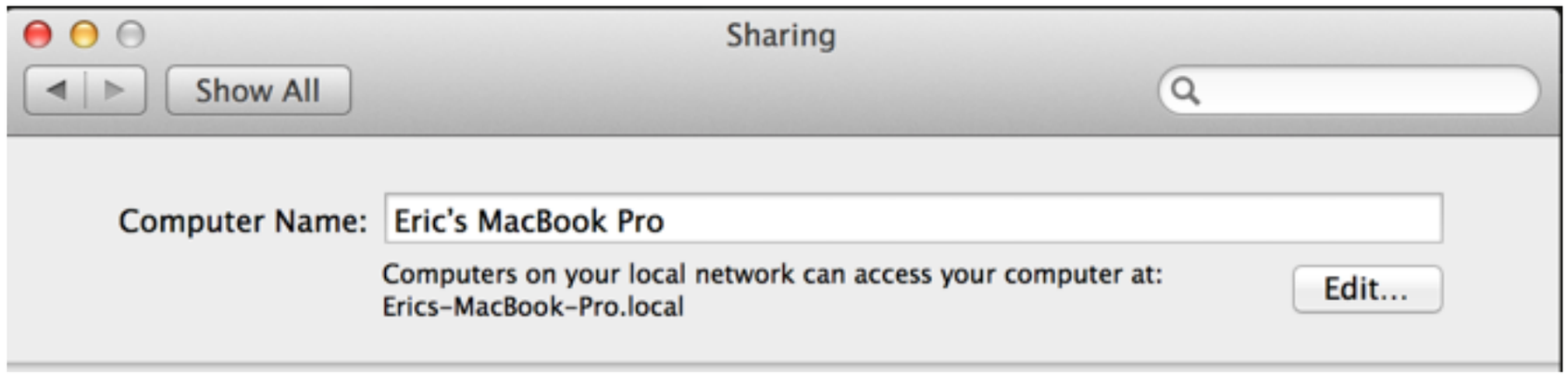
listen on 8888

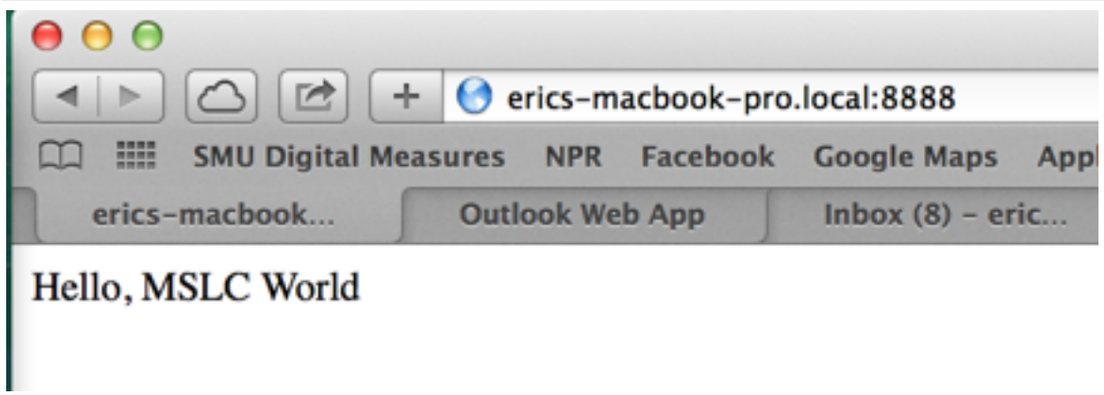
start the IO loop

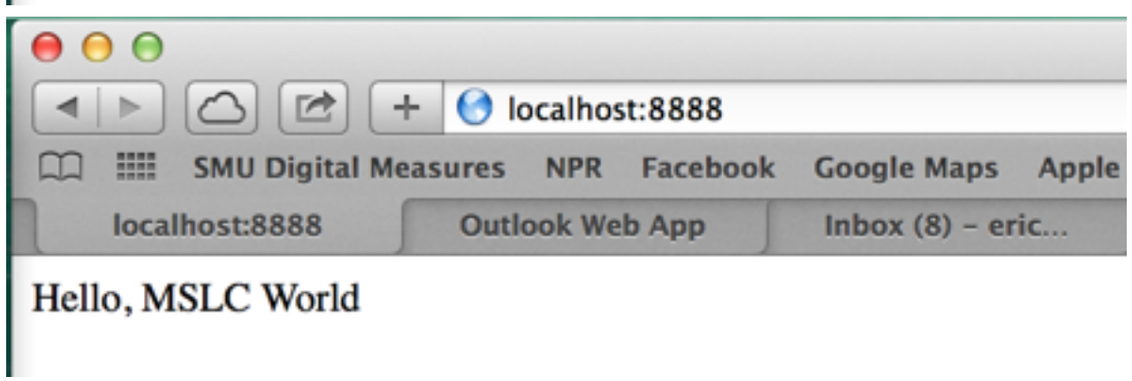
tornado example



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

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

flickrSearch = 'https://www.flickr.com/services/rest/?
method=flickr.photos.getRecent&api_key=9787477e45fec5e4f16ab9cbb60c3447'

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

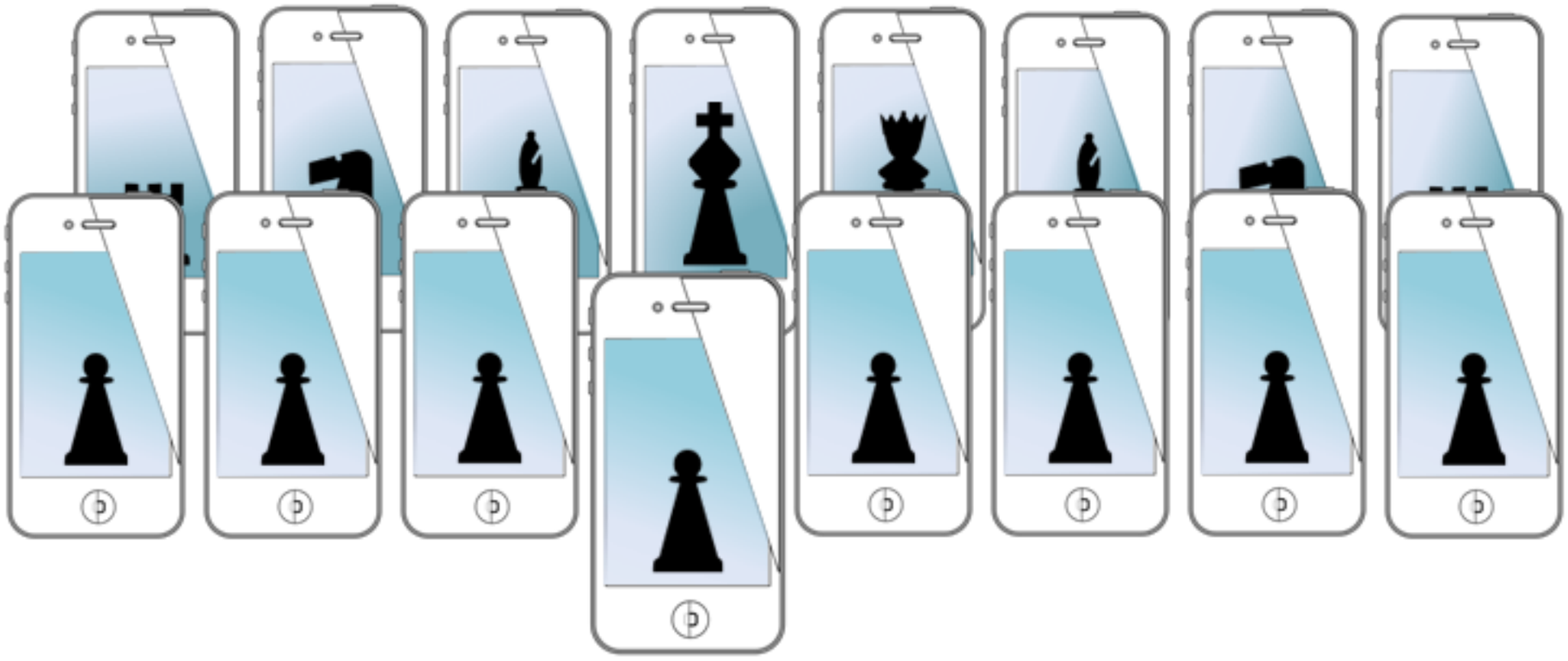
non-blocking example

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

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

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

week nine: python crash-course, tornado

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