Concurrency, Parallelism, Events, Asynchronicity, Oh My

threads, processes, queues & workers,
multiprocessing, gevent, twisted, yo momma

One thing at a time

per core

hyperthreading is a filthy lie

SMP/NUMA = think of multiple computers in one box

Concurrency vs Parallelism

flip-flops vs thongs

Parallelism

Speed-oriented
Physically parallel

Concurrency

Multiplexing operations
Reasoning about program structure

Von Neumann architecture

meet Von Neumann bottleneck

Multiple processes

```
wget -q -O- http://www.dreamhost.com/ \
| perl -ne 'for (/\b[[:upper:]]\w*\b/g) {print "$_\n"}' \
| tr '[:upper:]' '[:lower:]' \
| sort \
| uniq -c \
| sort -nr \
| head -5
```

waits for network

waits for disk

waits for disk

```
     30 dreamhost
     11 hosting
     10 wordpress
      9 web
      9 we
```

OS can schedule to avoid IO wait
(HT can schedule to avoid RAM wait)

Pipelines are lovely, but two-way communication is harder

IO waiting deadlock

```
# parent
data = child.stdout.read(8192)
```

```
# child
data = sys.stdin.read(8192)
```

"Blocking" on IO

Like refusing to participate
in a dinner conversation
because your glass is empty.

Nonblocking

"if I asked you to do X,
would you do it immediately?"

Asynchronous

abstraction on top of nonblocking;
queue outgoing data until writable,
notify of incoming data just read

Event loop

GUIs: "user did X, please react"
services: lots of async operations, concurrently

Callback-oriented programming

services: "received bytes B on connection C"

Deferred

Twisted API for
delayed-response
function calls

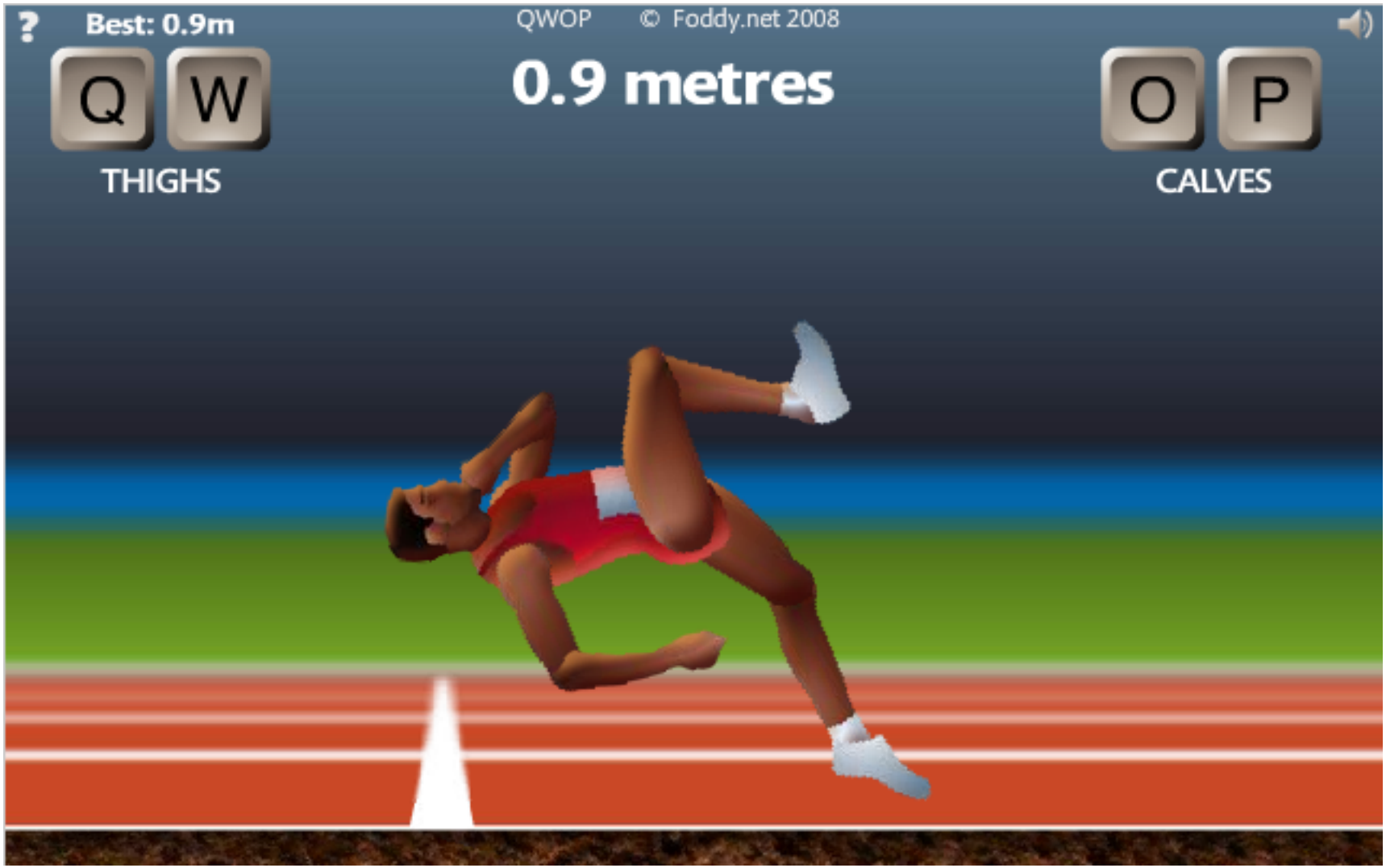
Deferred example

```
import something

def get_or_create(container, name):
    d = container.lookup(name)
    def or_create(failure):
        failure.trap(something.NotFoundError)
        return container.create(name)
    d.addErrback(or_create)
    return d

d = get_or_create(cont, 'foo')
def cb(foo):
    print 'Got foo:', foo
d.addCallback(cb)
```

Twisted



Use to poke fun at Node.js fanboys

concurrent.futures (PEP 3148)

Python fails to learn from Twisted
More like Java every day
Twisted dev not taken by surprise

Co-operative scheduling

like the nice guy at the ATM,
letting you go in between
because he still has
20 checks to deposit

Preemptive scheduling

the CPU gets taken away from you

(plus you usually get punished for hogging)

So.. back to processes

Communication overhead

(most of the time, just
premature optimization)

Creation cost

(5000 per second on a
900MHz Pentium 3,
don't know if anyone's cared since..)

Idea: Communicate by sharing memory

Threads

... now you have two problems

Unix has mmap you know

(aka share memory by sharing memory)

<rant>Voluntarily giving up that precious MMU counts as pretty stupid in my book...</rant>

<rant>Threads originate from operating systems with bad schedulers and VM</rant>

GIL

Python threads
are only good
for concurrency,
not for parallelism

Multiprocessing

Ingenious hack to emulate thread API with multiple processes.

Eww, pickles.

Do not trust the pickle

```
def process(work):
    while True:
        item = work.get()
        try:
            r = requests.get(item['url'])
            if r.status_code != 200:
                print 'Broken link {source} -> {url}'.format(**item)
                continue
            level = item['level'] - 1
            if level > 0:
                if r.headers['content-type'].split(';', 1)[0] != 'text/html':
                    print 'Not html: {url}'.format(**item)
                    continue
                doc = html5lib.parse(r.content, treebuilder='lxml')
                links = doc.xpath(
                    '//html:a/@href | //html:img/@src',
                    namespaces=dict(html='http://www.w3.org/1999/xhtml'),
                    )
                for link in links:
                    if link.startswith('#'):
                        continue
                    url = urlparse.urljoin(item['url'], link, allow_fragments=False)
                    parsed = urlparse.urlsplit(url)
                    if parsed.scheme not in ['http', 'https']:
                        print 'Not http: {url}'.format(url=url)
                        continue
                    work.put(
                        dict(
                            level=level,
                            url=url,
                            source=item['url'],
                            ),
                        )
        finally:
            work.task_done()
```

```
import html5lib
import multiprocessing
import requests
import urlparse
```

```
def main():
    work = multiprocessing.JoinableQueue(100)
    work.put(
        dict(
            level=2,
            url='http://www.dreamhost.com/',
            source=None,
            ),
        )
    processes = [
        multiprocessing.Process(
            target=process,
            kwargs=dict(
                work=work,
                ),
            )
        for _ in xrange(10)
        ]
    for proc in processes:
        proc.start()

    work.join()

    for proc in processes:
        proc.terminate()
    for proc in processes:
        proc.join()


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

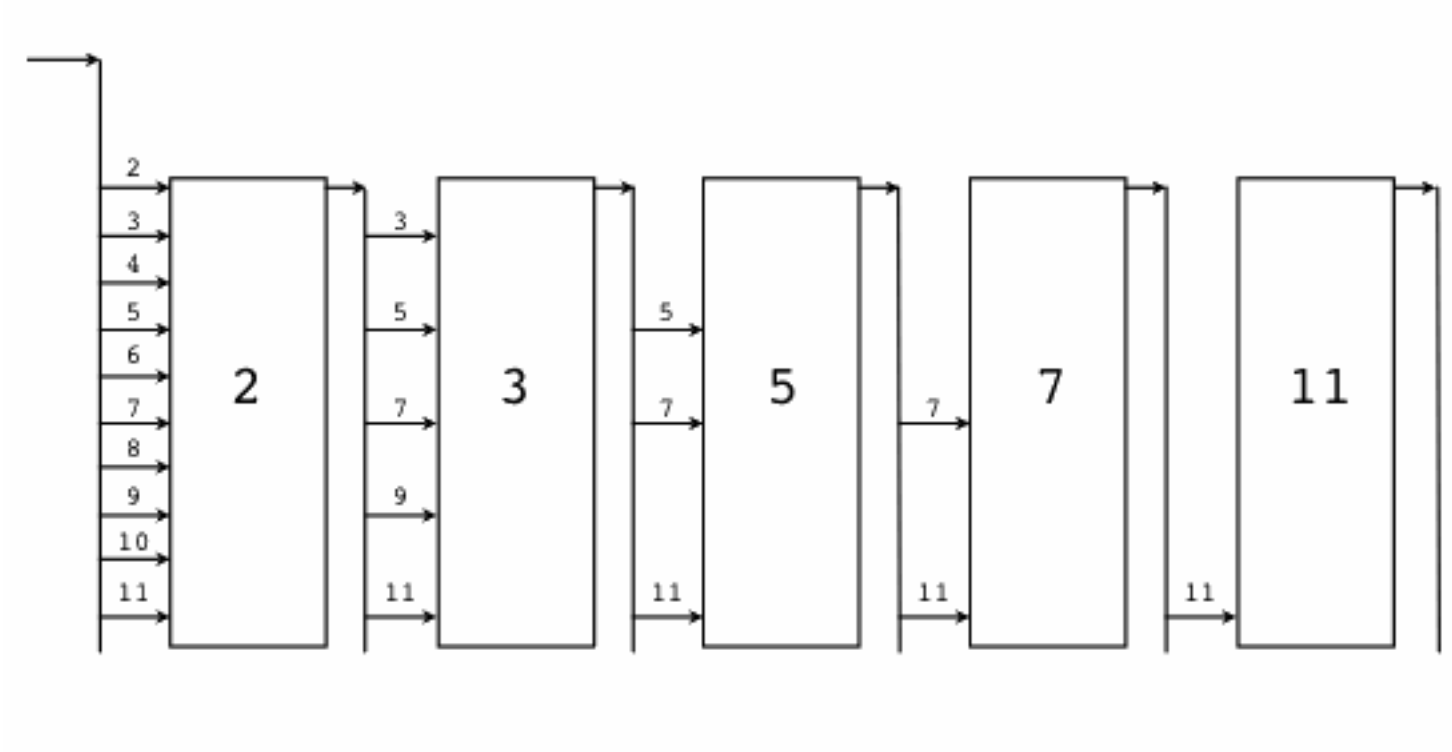
Communicating Sequential Processes

more about the thought process
than the implementation

"Share memory by communicating"

Radically less locking;
radically less locking bugs.

Prime number sieve with CSP



Prime number sieve with Gevent

```
from gevent import monkey; monkey.patch_all()
import gevent
import gevent.queue
import itertools

def generate(queue):
    """Send the sequence 2, 3, 4, ... to queue."""
    for i in itertools.count(2):
        queue.put(i)

def filter_primes(in_, out, prime):
    """Copy the values from in_ to out, removing those divisible by prime."""
    for i in in_:
        if i % prime != 0:
            out.put(i)

def main():
    ch = gevent.queue.Queue(maxsize=10)
    gevent.spawn(generate, ch)
    for i in xrange(100):
        # Print the first hundred primes.
        prime = ch.get()
        print prime
        ch1 = gevent.queue.Queue(maxsize=10)
        gevent.spawn(filter_primes, ch, ch1, prime)
        ch = ch1
```

Look ma, no threads!

monkey.patch_all()

Coroutines

```
def foo():
    a = yield
    b = yield
    yield a+b

f = foo()
next(f)
f.send(1)
print f.send(2)
```

Coroutines #2

```
def ticktock():
    messages = ['tick', 'tock']
    while True:
        cur = messages.pop(0)
        new = yield cur
        if new is not None:
            print '# Okay, {new} not {cur}.' \
                .format(new=new, cur=cur)
            cur = new
        messages.append(cur)
```

```
t = ticktock()
print next(t) # tick
print next(t) # tock
print next(t) # tick
t.send('bip')
# Okay, bip not tick.
print next(t) # bip
print next(t) # tock
print next(t) # bip
print next(t) # tock
t.send('bop')
# Okay, bop not tock.
print next(t) # bop
print next(t) # bip
print next(t) # bop
print next(t) # bip
print next(t) # bop
print next(t) # bip
```

Laziness without hubris

```
with open(path) as f:
    words_on_lines = (line.split() for line in f)
    words = itertools.chain.from_iterable(words_on_lines)
    lengths = (len(word) for word in words)
    (a, b) = itertools.tee(lengths)
    count = sum(1 for _ in a)
    summed = sum(b)
    avg = summed / count
```

Go watch Jesse Noller's PyCon talks

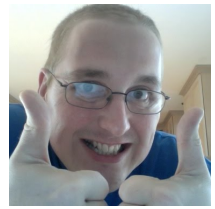
Thank you!

Questions?

fair use from http://en.wikipedia.org/wiki/Gil_Grissom

Image credits:

<3 CC http://www.flickr.com/photos/pedromourapinheiro/2122754745/



fair use from https://plus.google.com/115662513673837016240

self-portrait of anonymous macaque http://www.techdirt.com/articles/20110706/00200314983/monkey-business-can-monkey-license-its-copyrights-to-news-agency.shtml

<3 CC http://www.flickr.com/photos/bitzcelt/2841983698/