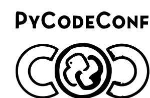
Cherry-Picking For Huge Success

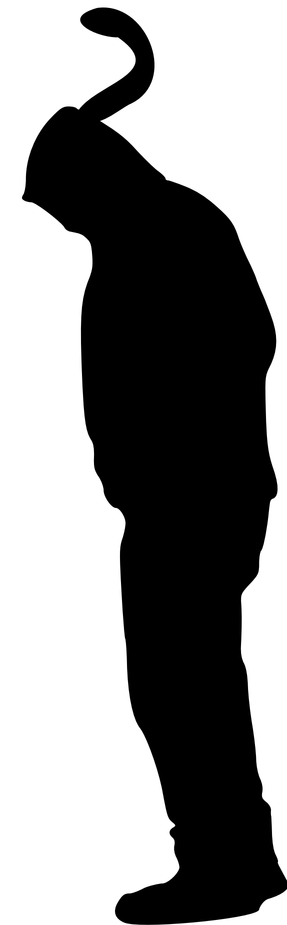
Armin Ronacher

PyCodeConf



Who Am I

- Armin Ronacher
- @mitsuhiko on Twitter/Github
- Part of the Pocoo Team
- Flask, Jinja2, Werkzeug, …



Preface

Framework / Programming language fights are boring.
Just use the best tool for the job.

The Problem

Consider Twitter

- 2006: Off the shelf Ruby on Rails application; static HTML; basic XML API
- Now: The API is the service the website itself is a JavaScript frontend to that API; everything is rate limited; Erlang/Java

Does Ruby Suck?

- No it does not.
- Neither does Python.
- Ruby / Python are amazing for quick prototyping.
- Expect your applications to change with the challenges of the problem.

Shifting Focus

- Expect your problems and implementations to change over time
- You might want to rewrite a part of your application in another language

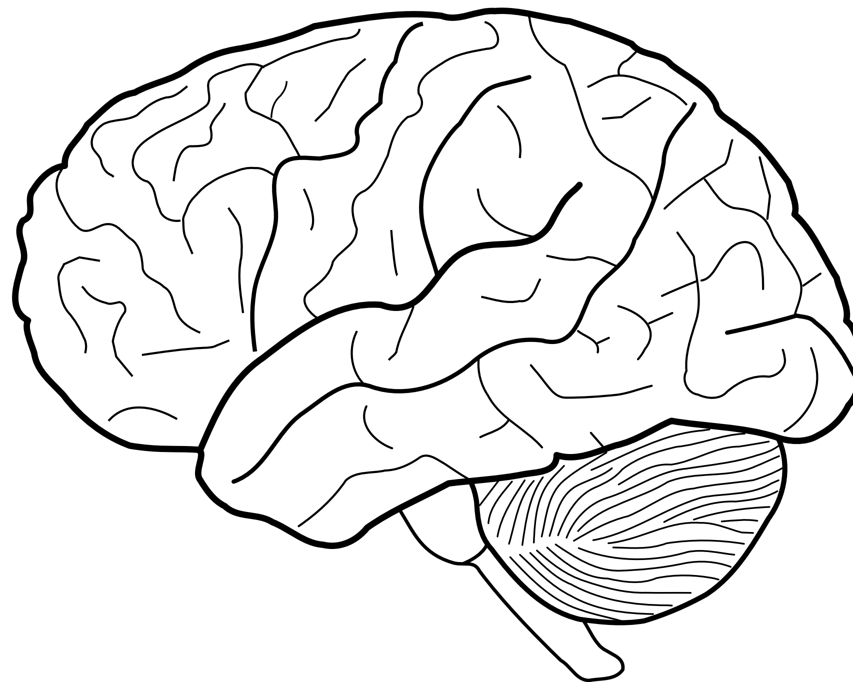
Proposed Solution

- Build smaller applications
- Combine these apps together to a large one

Cross Boundaries

- "Pygments is awesome"
- "I need Pygments in Ruby"
  - A: rewrite Pygments in Ruby
  - B: use a different syntax highlighter
  - C: Just accept Python and implement a service you can use from Ruby

Agnostic Code



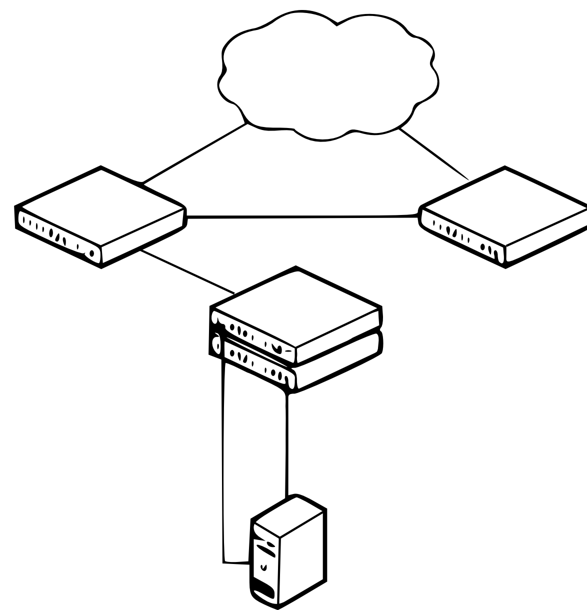
It only does Django

- You wrote a useful library that creates thumbnails?
- Don't make it depend on Django, even if you never want to switch from Django
- You might want to move the thumbnailing into a queue at one point and not need Django and your DB in your queue

Pass "X" in :-)

- Do not import "X"
- Store "X" on a class instance
- or pass "X" in as parameter
- Make "X" as specific as possible
- But not more than it has to be

Protocol Examples



Flask's Views

- Views can return response objects
- Response objects are WSGI applications
- No typecheck!
- Return any WSGI app

Flask + WSGI

```
from flask import Flask

app = Flask(__name__)


def hello_world_app(environ, start_response):
    headers = [('Content-Type', 'text/plain')]
    start_response('200 OK', headers)
    return ['Hello World!']


@app.route('/')
def index():
    return hello_world_app
```

Difflib

- Python's difflib module does not need strings, it works on anything that is iterable and contains hashable and comparable items.
- "X" == hashable and comparable
  - As specific as possible, but not too restrictive. Bad would be "X" == String

Consequences

- This came in very helpful when I had to diff HTML documents
- Parse into a stream of XML events — diff
- Render out inline HTML again with the differences wrapped in <ins>/<del>

Beauty in Design

- Genshi's XML stream's events is made of hashable, immutable objects
- The Stream is a Python iterable
- difflib can work with exactly that: hashable objects in a sequence
- *Goes well together, but was never designed to be used together*

Genshi's Stream

```
>>> from genshi.template import MarkupTemplate
>>> t = MarkupTemplate('<?xml version="1.0"?><test>'
... '<foo bar="baz"/></test>')
>>> g = iter(t.generate())
>>> g.next()
('XML_DECL', (u'1.0', None, -1), (None, 1, 0))
>>> g.next()
('START', (QName('test'), Attrs()), (None, 1, 21))
>>> g.next()
('START', (QName('foo'), Attrs([(QName('bar'), u'baz')])), (None, 1, 27))

...
```

Diffing XML

```
from genshi.template import MarkupTemplate
from difflib import SequenceMatcher

get_stream = lambda x: list(MarkupTemplate(x).generate())
a = get_stream('<?xml version="1.0"?><foo><a/></foo>')
b = get_stream('<?xml version="1.0"?><foo><b/></foo>')
matcher = SequenceMatcher(a=a, b=b)

for op, i1, i2, j1, j2 in matcher.get_opcodes():
    if op == 'replace':
        print 'del', a[i1:i2]
        print 'ins', b[j1:j2]
    elif op == 'delete':
        print 'del', a[i1:i2]
    elif op == 'insert':
        print 'ins', b[j1:j2]
```

Diff Result

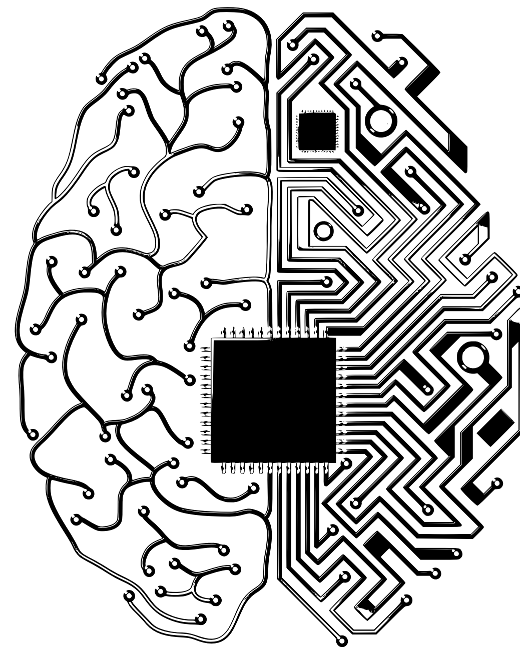
```
del    [('START', (QName('a'), Attrs()), (None, 1, 26)),
        ('END', QName('a'), (None, 1, 30))]
ins    [('START', (QName('b'), Attrs()), (None, 1, 26)),
        ('END', QName('b'), (None, 1, 30))]
```

Inline Diffing HTML

- mitsuhiko/**htmldiff**

```
>>> from htmldiff import render_html_diff
>>> render_html_diff('Foo <b>bar</b> baz', 'Foo <i>bar</i> baz')
u'<div class="diff">Foo <i class="tagdiff_replaced">bar</i> baz</div>'
>>> render_html_diff('Foo bar baz', 'Foo baz')
u'<div class="diff">Foo <del>bar</del> baz</div>'
>>> render_html_diff('Foo baz', 'Foo blah baz')
u'<div class="diff">Foo <ins>blah</ins> baz</div>'
```

Interface Examples



Serializers

- pickle, phpserialize, itsdangerous, json
- Within the compatible set of types, they all work as drop-in replacements for each other

Example

```
>>> from itsdangerous import URLSafeSerializer
>>> smod = URLSafeSerializer('secret-key')
>>> smod.dumps([1, 2, 3])
'WzEsMiwzXQ.ss4nn3igDDAwxiqsWvj3EQ9FdIQ'
>>> smod.loads(_)
[1, 2, 3]
>>>
>>> import pickle as smod
>>> smod.dumps([1, 2, 3])
'(lp0\nI1\naI2\naI3\na.'
>>> smod.loads(_)
[1, 2, 3]
```

“What's your Point Armin?”

Loosely Coupled

- Small, independent pieces of code (both “libraries” and “apps”)
- Combine them with protocols and through interfaces
- This is how you can structure applications

Splitting up …

- … is not the problem
- Combining things together is

Mergepoints

- WSGI
- HTTP
- ZeroMQ
- Message queues
- A datastore
- JavaScript

WSGI

Overview

- Pros:
  - Every Python framework speaks it or can easily be ported to work on top of WSGI or to be able to host WSGI apps
- Cons:
  - Only works within Python
  - Often insufficient

The WSGI Env

- Apps that need request data can limit themselves to the data in the WSGI env
- That way they are 100% framework independent.
  - Good: env['PATH_INFO']
  - Bad: request.path_info

Middlewares

- Often overused
- Sometimes helpful though:
  - Debugging
  - Profiling
  - Dispatching to different applications
  - Fixing server / browser bugs

WSGI as Mergepoint

```
from myflaskapp import application as app1
from mybottleapp import application as app2
from mydjangoapp import application as app3

app = DispatchedApplication({
    '/':       app1,
    '/api':    app2,
    '/admin':  app3
})
```

Not merging?

- Correct: these applications are independent
- But what happens if we inject common information into them?

WSGI as Mergepoint

```
class InjectCommonInformation(object):

   def __init__(self, app):
       self.app = app

   def __call__(self, environ, start_response):
       db_connection = connect_database()
       user = get_current_user(environ, db_connection)
       environ['myapplication.data'] = {
           'current_user': user,
           'db':           db_connection
       }
       return self.app(environ, start_response)

app = InjectCommonInformation(app)
```

Problems with That

- Cannot consume form data
- Processing responses from applications is a complex matter
- Cannot inject custom HTML into responses easily due to the various ways WSGI apps can be written
- What if an app runs outside of the WSGI request/response cycle?

Libraries

- Werkzeug
- WebOb
- Paste

Django & WSGI

- Django used to do WSGI really badly
- Getting a documented WSGI entrypoint for applying middlewares
- Easy enough to pass out WSGI apps with the Django Response object

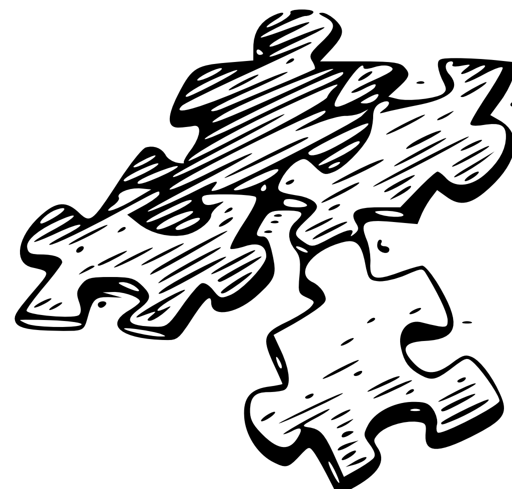
WSGI -> Django

```
from werkzeug.test import run_wsgi_app
from werkzeug.wrappers import WerkzeugResponse
from django.http import HttpResponse

def make_response(request, app):
    iter, status, headers = run_wsgi_app(app, request.META)
    status_code = int(status.split(None)[0])
    resp = HttpResponse(iter, status=status_code)
    for key, value in headers.iteritems():
        resp[key] = value
    return resp

def make_wsgi_app(resp):
    return WerkzeugResponse(resp, status=resp.status_code,
                            headers=resp.items())
```

Usage

```
from my_wsgi_app import application
from wsgi_to_django import make_response


def my_django_view(request):
    return make_response(request, application)
```

HTTP



Overview

- Pros:
  - Language independent
  - Cacheable
- Cons:
  - Harder to work with than WSGI
  - Complex specification
  - Same problems as WSGI

Proxying

- Write three different apps
- Let nginx do the proxying
- The more HTTP you speak, the better

Cool Things

- If all your services speak HTTP properly you can just put caching layers between them
- HTTP can be debugged easily (curl)
- Entirely language independent

Suggestion

- Let your services speak HTTP.
- You need syntax highlighting with Pygments but your application is written in Ruby? Write a small Flask app that exposes Pygments via HTTP

Libraries

- Python-Requests
- Your favorite WSGI Server (gunicorn, CherryPy, Paste etc.)
- Tornado, Twisted

ZeroMQ

ØMQ

Not a Queue

- ZeroMQ is basically sockets on steroids
- Language independent
- Different usage patterns:
  - push/pull
  - pub/sub

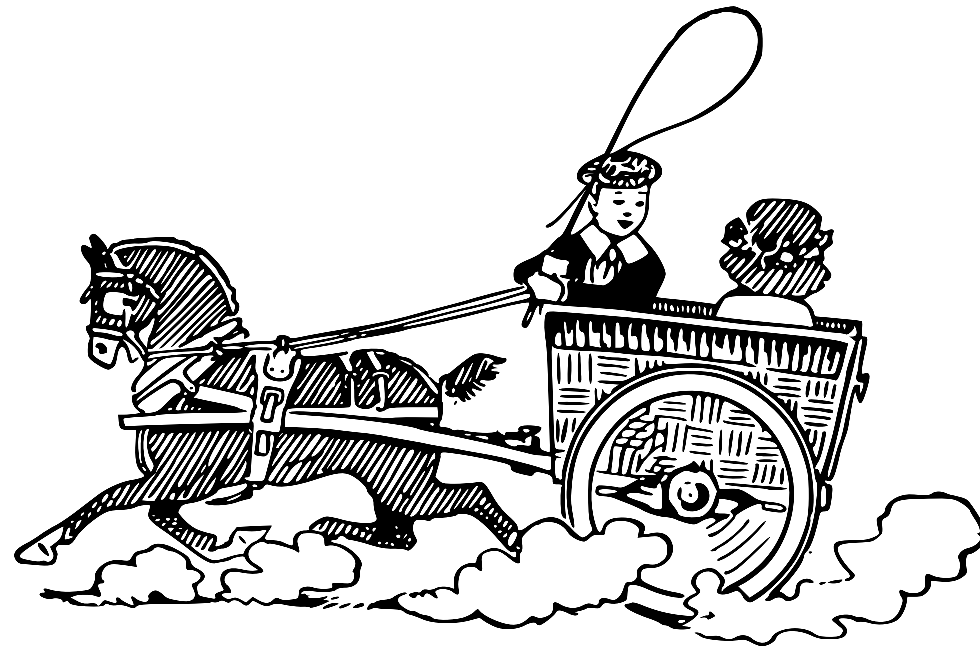
ZeroMQ vs HTTP

- ZeroMQ is easier to use than HTTP
- You however don't get the nice caching
- On the plus side you can dispatch message to many subscribers
- ZeroMQ abstracts the bad parts of sockets and HTTP away from you (timeouts, EINTR, etc.)

Random Thoughts

- ZeroMQ hides connection problems
- Blocks on lack of connectivity
- You might have to build your own broker

Message Queues



It might take a while

- Move long running tasks outside of the request handling process
- Possibly dispatch it to different machines
- But: It can be an entirely different code that processes the queue entry, different language even

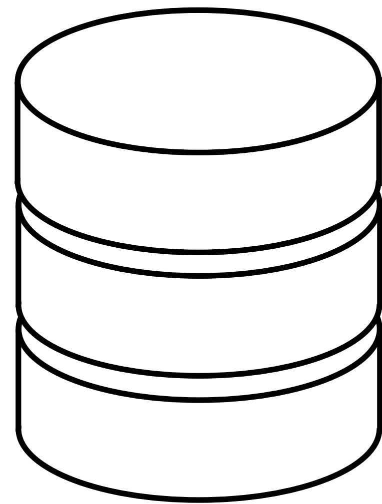
Queues

- Accessor library: Celery
- AMQP (RabbitMQ)
- Redis
- Tokyo Tyrant

Various Things

- Don't expect your calls to be nonblocking
- Greatly simplifies testing!
- Build your own queue > no queue
- Redis queues are a good start

A Datastore



The Obvious One

- Use the same datastore for two different applications.
- For as long as everybody plays by the rules this is simple and efficient.

Classical Example

- Flask application
- Django Admin

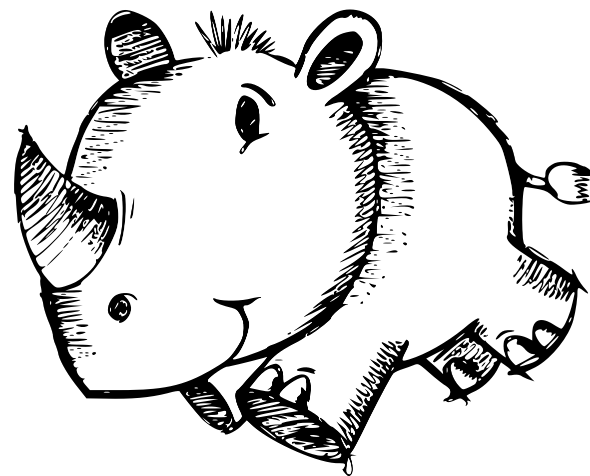
Redis

- A datastore
- Remote datastructures!
- Can easily be used as a queue
- Simple interface, bindings for every language
- Python pushes, Java pulls and executes

Bash Queue Consumer

```
#!/bin/bash
QUEUE_NAME=my_key

while :
do
  args=`redis-cli -d $'\t' blpop $QUEUE_NAME 0 | cut -f2`
  ./my-script $args
done
```

JavaScript



It's awesome

- Geeks hate JavaScript
- The average users does not care at all
- Why do we hate JavaScript?
  - Language us ugly
  - Can be abused for things we think are harmful (user tracking)

Ugly Language

- Accept it
- Use CoffeeScript
- it's the C kind of ugly, not the PHP one

Can be abused

- So can cars, bittorrent etc.
- Grow up :-)

Google's Bar

- That Google bar on top of all their products?
- You can implement that in JavaScript only
- Fetch some JSON
- Display current user info
- Application independent

Is it used?

- Real world example: xbox.com
- Login via live.com
- Your user box on xbox.com is fetched entirely with JavaScript
- Login requires JavaScript, no fallback

DICE's Battlelog

- Made by DICE/ESN for Battlefield 3
- Players join games via their Browser
- The joining of games is triggered by the browser and a token is handed over to the game.
- Browser plugin hands over to the game client.

Technologies

- Python for the Battlelog service
- JavaScript for the frontend
- Java for the push service
- C++ for the Game Client and Server
- HTTP for communication

Other Things

- JavaScript can efficiently transform the DOM
- You can do things you always wanted to do no the server side but never could because of performance or scaling considerations
- Instantly updating page elements!
- backbone.js

Testing

- JavaScript testing only sucks for others
- You control the service, you know the API endpoints. Speak HTTP with them
- HtmlUnit has pretty good JavaScript support
- Selenium supports HtmlUnit

Processes

Daemons

- Yes, you need to keep them running
- Yes it can be annoying
- systemd / supervisord help

SYSTEMD

- Socket is managed by the OS
- Your application activates on the first request to that socket
- Restart applications, clients queue up in the OS
- Python's socket module does not operate on arbitrary file numbers before 3 (AFAIK)

Processes+

- But processes are a good idea on Unix:
  - Different privileges
  - You can shoot down individual pieces without breaking the whole system
  - You can performance tune individual things better
  - No global lock :-)

Python 3

- libpython2 and libpython3 have clashing symbols
- You cannot run Python 2 and Python 3 in the same process
- ZeroMQ / HTTP etc. are an upgrade option

!Q&A?

lucumr.pocoo.org/talks/