

# Pragmatic Django Performance: The 20% that matters

- Performance matters
- Mistakes to avoid
- Smallest changes, largest impact

# hello

- Wes Winham @weswinham, [gplus.to/weswinham](https://plus.google.com/u/0/weswinham)
- Responsible for product development at PolicyStat
- ~6 year old Django app, ~1 million monthly page views
- Lots of lessons learned the hard way

Performance Matters

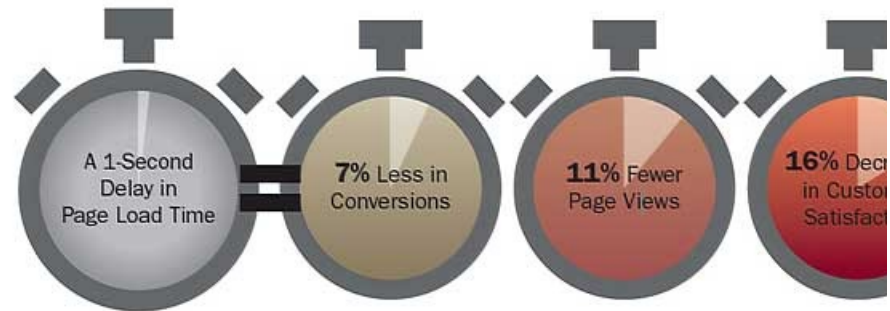


photo: [cloudreviews.com](http://cloudreviews.com)

The magic number now  
is now somewhere  
between 300 to 250ms

- Harry Schum, Speed specialist for MS

# Study Results

- Bing- 2s == 4.3% less revenue
- Yahoo- 400ms == 5-9% less traffic
- Google- 400ms == ~1% less searching

But we're busy!

# Not that busy

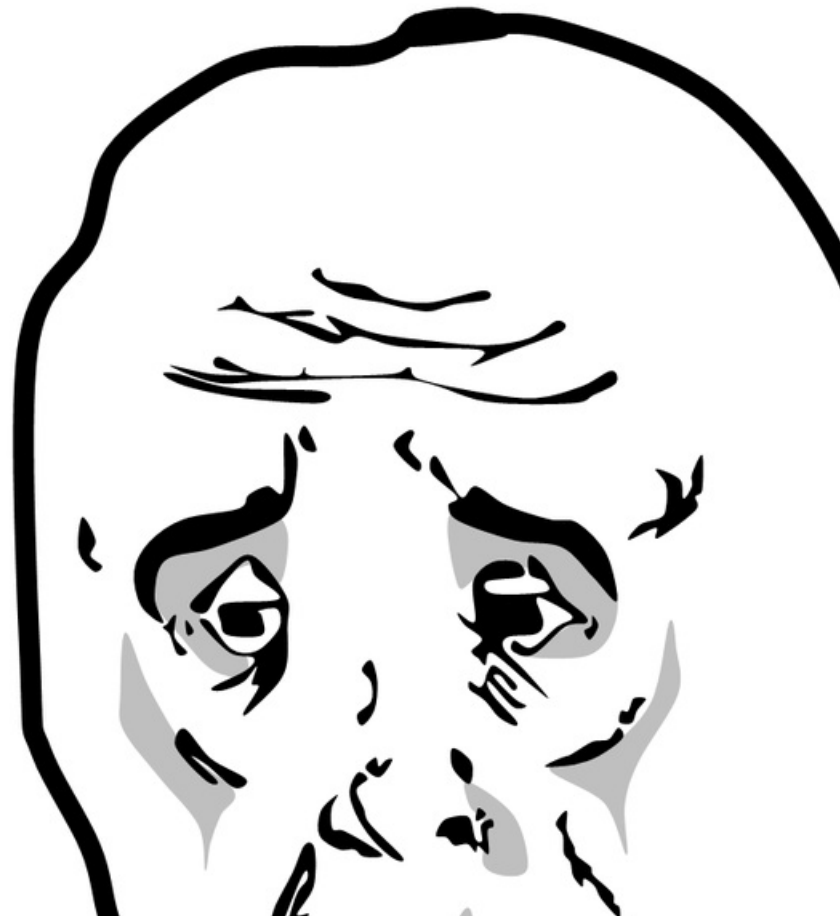
- Many cheap/easy "optimizations"
- Easy once you notice them
- One-time improvements with big payoffs



First rule of  
optimization:

No premature  
optimization

It affects to lots of  
developers



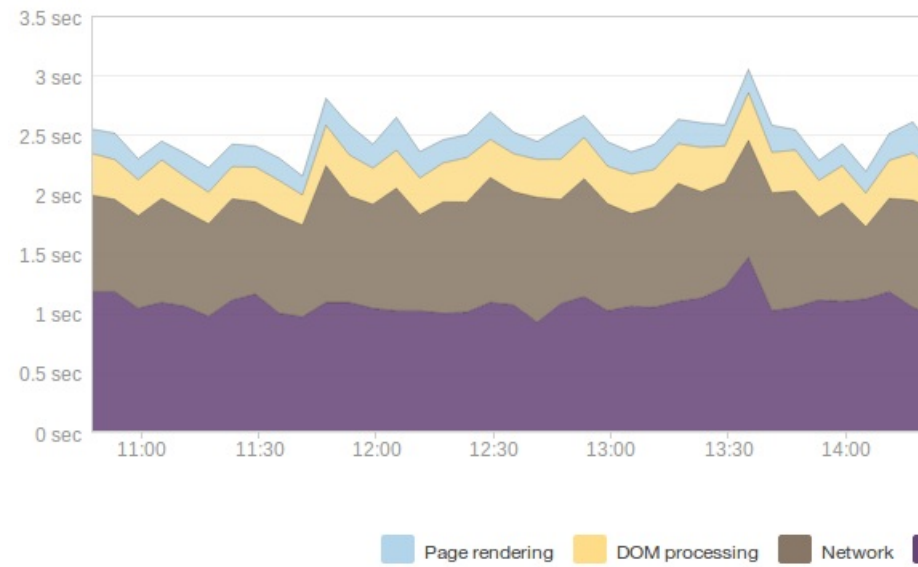
# Choose obvious wins!

- Don't add code complexity
- Don't make things fragile
- But you should be monitoring (New Relic)

Advice by (bad)  
example

Anti-pattern 1: Ignore  
the front end

Browser page load time ▾ ⓘ





# Our easiest wins are on the front end!

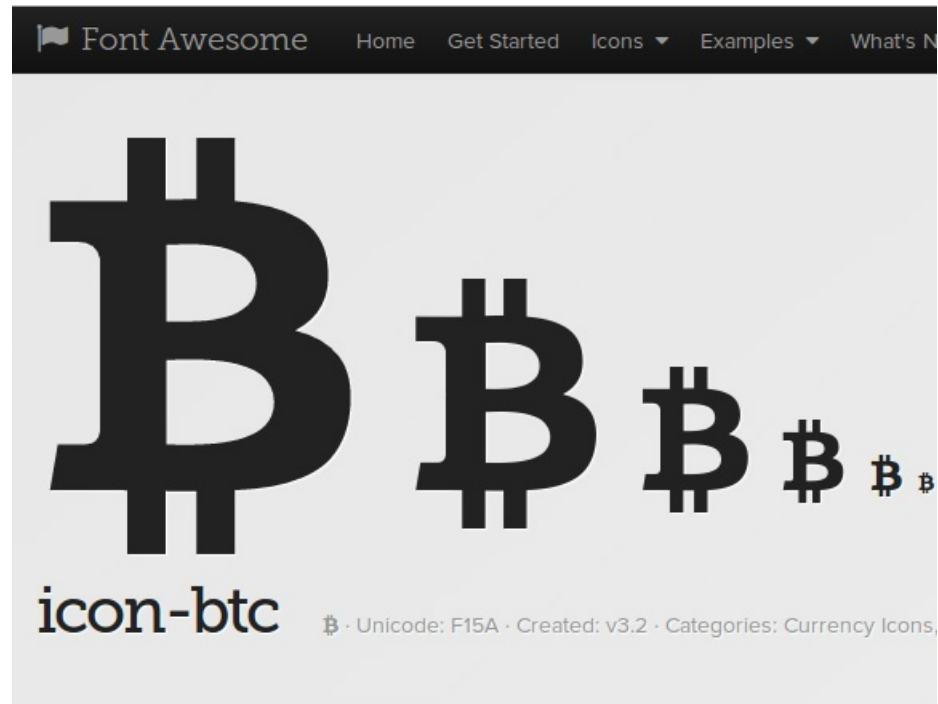
- Server side <50% of time
- Usually <25%

# Front end 20%: Reverse Proxy

- Lives in front of uwsgi/mod\_wsgi
- Protects you from slow clients
- Serves your media crazy-fast

## Front end 20%: GZIP

- Tell nginx to GZIP everything
- Remove django gzip middleware
- Cut network transfer in half, for free



After you get **up and running**, you can place Font Awesome icons just about anywhere with

₿ icon-btc

## Front end 20%: Use a font icon

- HTTP requests for single icons slow you down
- FontAwesome etc look better, anyway

## Front end 20%: Merged media

- Use django-compressor (etc)
- CSS at the top, JS at the bottom
- Fewer HTTP requests win!

```
# my_template.html
```

```
{% load compress %}
```

```
{% compress css %}
```

```
<link rel="stylesheet"
```

```
    href="/static/css/one.css" type="text/css">
```

```
<link rel="stylesheet"
```

```
    href="/static/css/two.css" type="text/css">
```

```
{% endcompress %}
```

Status Text	Type	Initiator	Size Content	Time Latency
200 OK	text/html	Other	6.4 KB 20.0 KB	1.14 s 1.14 s
200 OK	text/css	<a href="#">pstatdev.policystat.com/:29</a> Parser	17.8 KB 77.2 KB	380 ms 378 ms
200 OK	text/css	<a href="#">pstatdev.policystat.com/:35</a> Parser	5.4 KB 21.6 KB	99 ms 98 ms
200 OK	text/javascript	<a href="#">pstatdev.policystat.com/:62</a> Parser	0 B 91.7 KB	36 ms 30 ms
200 OK	text/javascript	<a href="#">pstatdev.policystat.com/:63</a> Parser	0 B 197 KB	62 ms 35 ms
200 OK	image/jpeg	<a href="#">pstatdev.policystat.com/:139</a> Parser	9.4 KB 9.0 KB	419 ms 419 ms
200 OK	application/x-javas...	<a href="#">pstatdev.policystat.com/:476</a> Parser	2.3 KB 5.3 KB	29 ms 28 ms
200 OK	image/gif	<a href="#">pstatdev.policystat.com/:143</a> Parser	4.1 KB 3.8 KB	30 ms 30 ms
200 OK	image/gif	<a href="#">pstatdev.policystat.com/:193</a> Parser	1.4 KB 1.0 KB	63 ms 62 ms
200 OK	image/gif	<a href="#">pstatdev.policystat.com/:260</a> Parser	901 B 563 B	86 ms 85 ms
200 OK	application/x-javas...	<a href="#">(index):485</a> Script	8.1 KB 20.5 KB	160 ms 159 ms
200 OK	application/x-woff	<a href="#">pstatdev.policystat.com/:1</a> Parser	43.2 KB 42.6 KB	20 ms 18 ms
200 OK	application/json	<a href="#">mixpanel.2.js:33</a> Script	469 B 1 B	34 ms 34 ms
200 OK		<a href="#">(index):616</a> Script	3.0 KB 3.0 KB	10 ms 10 ms



## Anti-pattern 2: Database black box

# Ways to kill your database

- Don't use a cache or read slave
- What's an index?
- Who needs SQL?

# Database 20%: Memcached + Johnny- Cache

- Dead-simple caching
- Stale reads never hit your DB
- Just works

```
# settings.py
```

```
MIDDLEWARE_CLASSES = (  
    'johnny.middleware.LocalStoreClearMiddlew  
    'johnny.middleware.QueryCacheMiddleware  
    # ...  
)
```

## Database 20%: Slow Query Log

- Tells you when you're bad
- Can still mostly use the ORM

Enable the Slow  
Query Log

```
# /etc/mysql/conf.d/slow_query.cnf
```

```
slow_query_log = 1
```

```
long_query_time = .1
```

```
log_queries_not_using_indexes = 1
```

Read the Slow Query  
Log



```
# Time: 130823 13:45:48
# User@Host: live0[live0] @ URL[IP]
# Query_time: 10.454075  Lock_time: 0.000000
  Rows_sent: 20  Rows_examined: 149654
use polycystat;
SET timestamp=1377265548;
```

# How bad was it?

# Query\_time: 10.454075 Lock\_time: 0.00008  
Rows\_sent: 20 Rows\_examined: 149654

# When was it?

SET timestamp=1377265548;

```
SELECT `auth_user`.`id`, ...snip...  
FROM `auth_user` LEFT OUTER JOIN `pstat_  
  ON (`auth_user`.`id` = `pstat_profile`.`user_id`  
INNER JOIN `pstat_tenant`  
  ON (`pstat_profile`.`tenant_id` = `pstat_tenant`.`tenant_id`)
```

```
WHERE (`pstat_profile`.`is_guest` = 0
      AND `pstat_profile`.`tenant_id` IN (
        301, ...snip... 356)
      AND `auth_user`.`is_superuser` = 0
      AND `auth_user`.`is_active` = 1
)
```

# Where clause across tables

- Can't index across tables with MySQL
- Your DB chooses *one* index per table
- A JOIN key counts as *one*
- Easy to filter superuser and inactive in python

```
ORDER BY `auth_user`.`last_name` ASC,  
`auth_user`.`first_name` ASC,  
`auth_user`.`username` ASC  
LIMIT 20;
```

# Complex ORDER BY

- No index there
- Creates a temporary table!
- Beware of default `order_by`



# Database fixes

- Add an index `tenant_id, is_guest`
- Python-side filtering `is_superuser, is_active`
- Sort on indexes `last_name, first_name, username`

## Anti-pattern 3: Default Django Settings

## Django Settings 20%: Cache Templates

- Template evaluation is fast
- Disk I/O can be slow
- Cache templates in memory!

```
TEMPLATE_LOADERS = (  
    (  
        'django.template.loaders.cached.Loader',  
        (  
            'django.template.loaders.filesystem.Loader'  
            'django.template.loaders.app_directories.Lc  
        )  
    ),  
)
```

Anti-pattern 4: Query  
all the things!

## Query 20%: Batch queries

- Don't do queries when iterating in templates
- Get things in one big query, where possible
- `select_related()` is usually your friend

```
# views.py
```

```
context = {  
    'my_things': MyThing.objects.filter(foo=bar)  
}
```

```
# my_template.html
```

```
<table>  
{% for my_thing in my_things %}  
    <tr>  
        <td>{{ my_thing.name }}</td>  
        <td>{{ my_thing.other_thing.name }}</td>  
    <tr>  
{% endfor %}  
</table>
```

See the fail?



```
# views.py
```

```
context = {  
    'my_things': MyThing.objects.filter(  
        foo=bar  
    ).select_related('other_thing__name')  
}
```

Automate catching  
this fail!

```
# views.py
```

```
def test_query_growth(self):  
    expected_queries = FuzzyInteger(10, 13)
```

```
    # Make 5 MyThing objects
```

```
    with self.assertNumQueries(expected_queries):  
        # Load the view
```

```
    # Make 5 more MyThing objects
```

```
    with self.assertNumQueries(expected_queries):  
        # Load the view again
```

# Focus on the easy wins

- No extra complexity
- Make them a habit
- The other 20% is much harder

