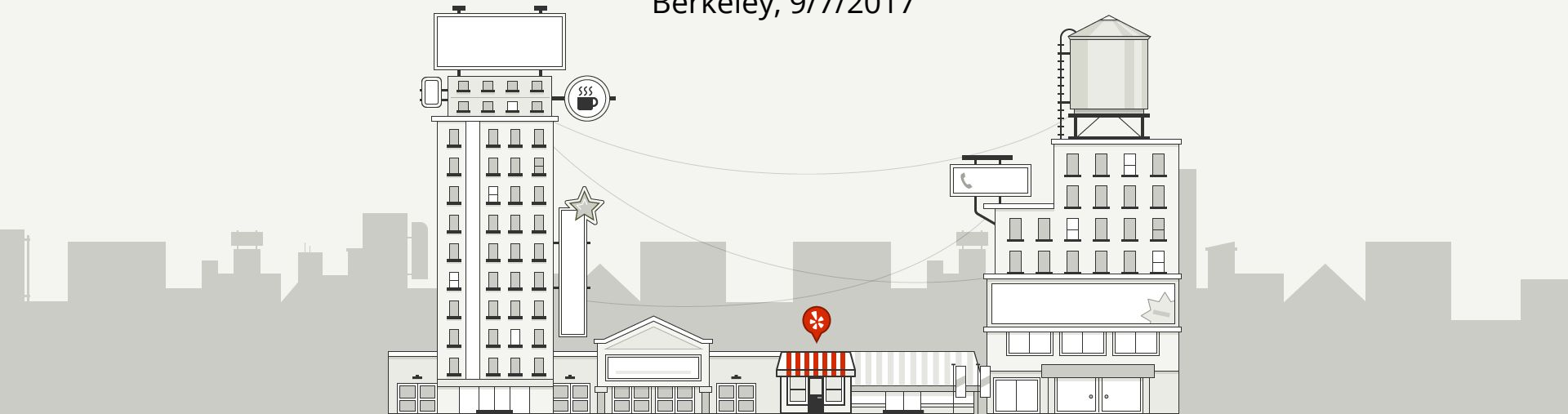
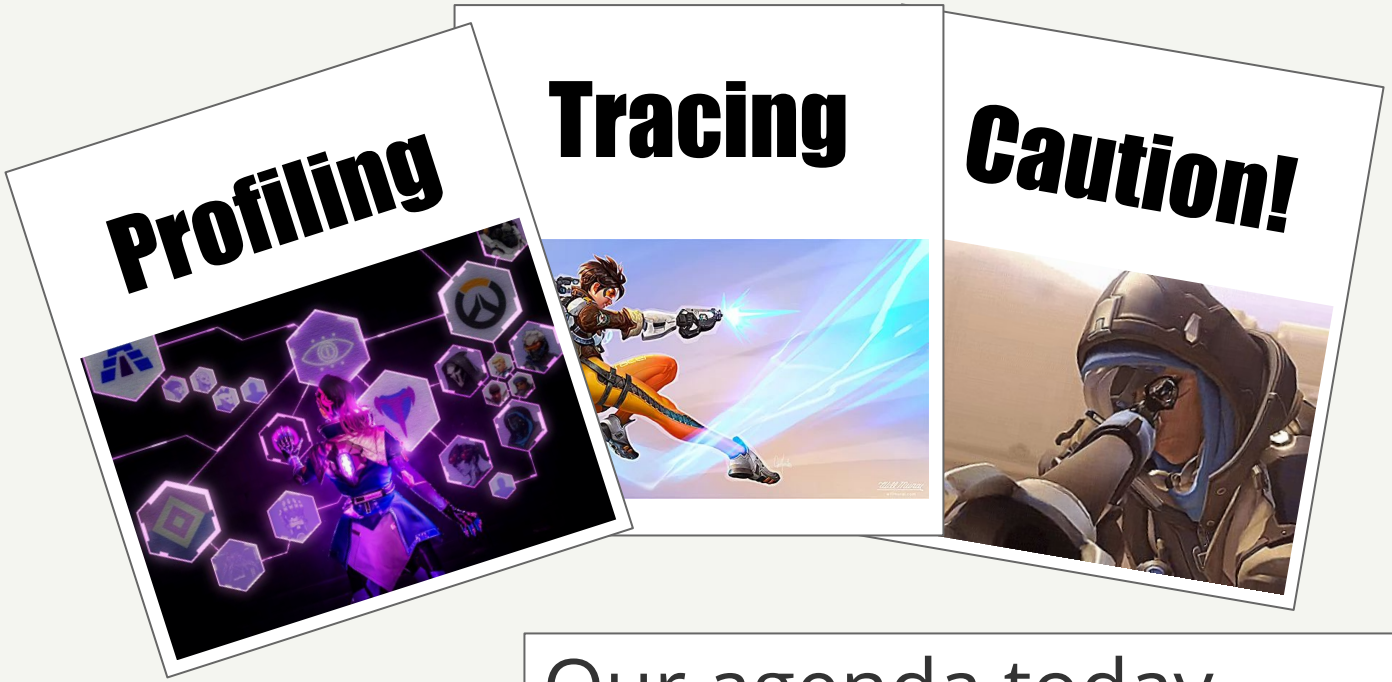
What’s slow?

Tools and Stories from Within Yelp’s Infrastructure

Arnaud Brousseau
Berkeley, 9/7/2017





Our agenda today

Profiling

Image credits: Pion Kim

What’s profiling?

Profiling is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the **frequency and duration of function calls.**

Source: wikipedia

Profiling your own Python code

⇒ Use Python's cprofile module!

```
$ python -m cProfile -o cprofile.data multiply.py 8123466 9119819
```

Profiling your own Python code

⇒ Use Python's cprofile module!

```
$ python -m cProfile -o cprofile.data multiply.py 8123466 9119819

8123466*9119819 ==> 74084539572654

$ python -m pstats cprofile.data
```

Profiling your own Python code

⇒ Use Python's cprofile module!

```
$ python -m cProfile -o cprofile.data multiply.py 8123466 9119819
8123466*9119819 ==> 74084539572654
$ python -m pstats cprofile.data
Welcome to the profile statistics browser.
cprofile.data% help
Documented commands (type help <topic>):
========================================
EOF  add  callees  callers  help  quit  read  reverse  sort  stats
```

Profiling your own Python code

⇒ Use Python's cprofile module!

```
cprofile.data% sort tottime
cprofile.data% stats
         9119834 function calls in 2.824 seconds
   Ordered by: internal time
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    1.777    1.777    2.823    2.823 multiply.py:7(multiply)
  9119819    0.839    0.000    0.839    0.000 multiply.py:4(add)
        1    0.207    0.207    0.207    0.207 {range}
        1    0.001    0.001    2.824    2.824 multiply.py:1(<module>)
        1    0.000    0.000    0.000    0.000 {print}
        …(more output)…
```

Profiling your own Python code

⇒ Use Python's cprofile module!

```
cprofile.data% sort tottime
cprofile.data% stats
         9119834 function calls in 2.824 seconds
   Ordered by: internal time
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    1.777    1.777    2.823    2.823 multiply.py:7(multiply)
  9119819    0.839    0.000    0.839    0.000 multiply.py:4(add)
        1    0.207    0.207    0.207    0.207 {range}
        1    0.001    0.001    2.824    2.824 multiply.py:1(<module>)
        1    0.000    0.000    0.000    0.000 {print}
        ...(more output)...
```

Profiling at Yelp

- We **also** use cprofile!
- We’re automatically collecting profiles for .01% of traffic on yelp.com
- We wrote a service to aggregate and visualize profile information
- It handles ~340Gb per day of profiling data, on average

yelp Profilistic

What's This?

Profilistic is an internal tool for viewing and aggregating Python profile data.

More Info

Visualize a cProfile

To visualize a Python profile, append **?cprofile_email=yourname@yelp.com** to any Yelp URL to get its profile emailed to you. Alternatively, here are 7 fresh sample profiles collected today:

- **admin**: cprofile-kafka/2017/08/22/cprofile/050acbd8bf6be1f6.gz
- **api**: cprofile-kafka/2017/08/22/cprofile/ac05c9c04ff5acad.gz
- **biz**: cprofile-kafka/2017/08/22/cprofile/014a7b0aa3fc1f84.gz
- **internalapi**: cprofile-kafka/2017/08/22/cprofile/ade96ac3cea4fd0a.gz
- **main**: cprofile-kafka/2017/08/22/cprofile/3dbbfe25c10a3d6c.gz
- **new_mobile**: cprofile-kafka/2017/08/22/cprofile/77c296c7c1e006dc.gz
- **partner_api**: cprofile-kafka/2017/08/22/cprofile/bc9a35e38cb1bb97.gz

Aggregate cProfiles

Yelp Main | Services

Earliest date inclusive 08/21/2017

Latest date inclusive 08/21/2017

Deploy Version	Site	Servlet	Action
deploy-oh-hi-markl_3f10f581fa deploy-tgif_77ee27f350 deploy-total-eclipse-of-the-heart_			

▸ Additional Filters

E-mail on completion Must be @yelp.com

Go!

Consumer (Archived) / CON-7880

Why is http://m.yelp.com/biz/b-nails-lincoln-city slow?

Agile Board | More

Details

Type:	Bug	Status:	DONE
Priority:	2 Do next	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	None
Component/s:	Performance		
Labels:	None		

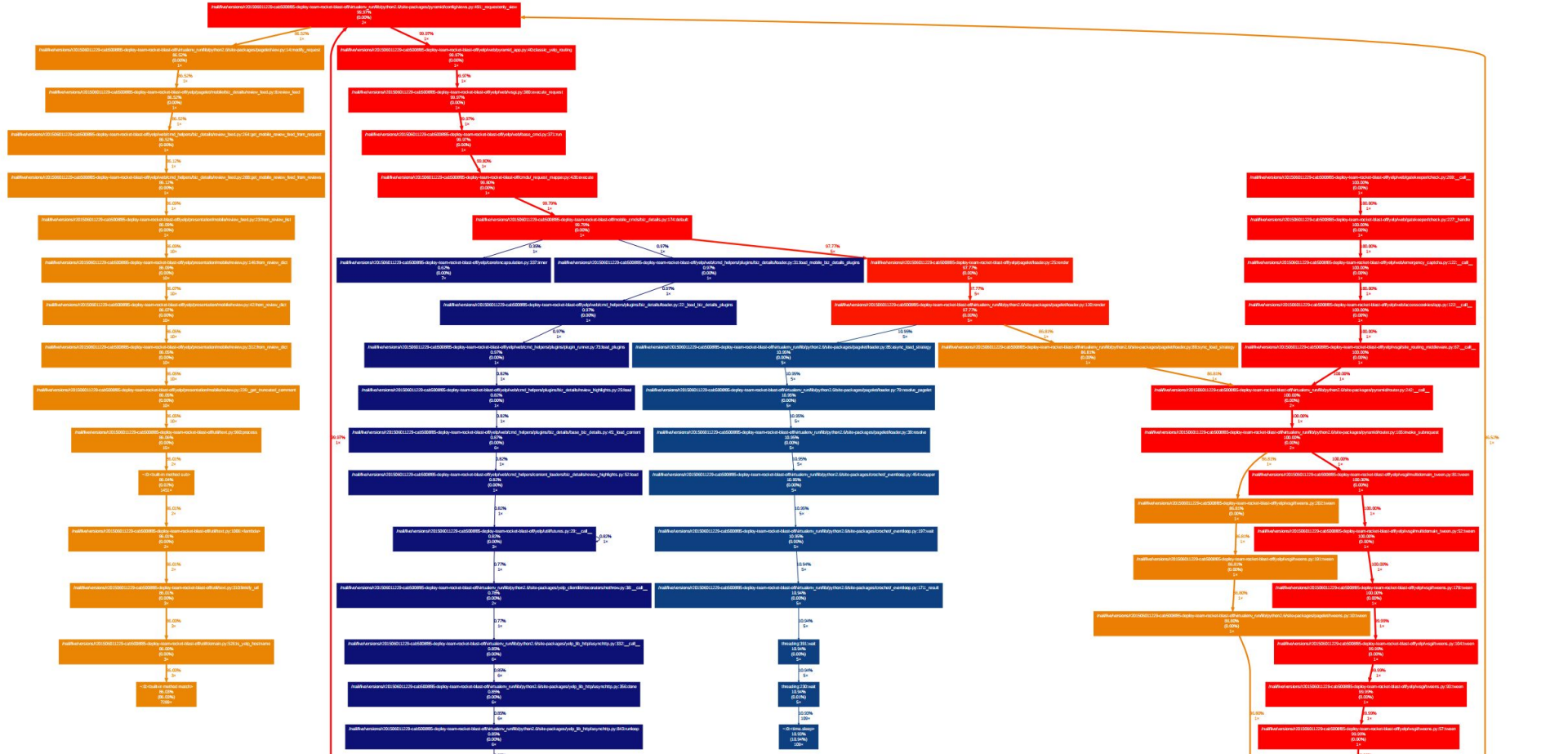
Description

See https://dowser.yelpcorp.com/#/dashboard/elasticsearch/tmp_slow_biz_details_page

Repro: load http://m.yelp.com/biz/b-nails-lincoln-city

It seems to be only slow on m.yelp. www loads decently fast!

I can repro this in an anonymous browser window so it's not caused by logged in cookie or dirty session.



```
664 yelp_hostname_regex = re.compile(
665     r"""
666         ^(                          # start of string
667             [^%(not_regname)s]+\.  # the not-not pattern allows for unicode characters.
668         )*                          # any number of domain segments
669         (%(yelp_domains)s)          # one of the known yelp domains
670         \.?                         # optionally followed by a dot (?!)
671         $                           # end of string
672     """ % dict(
673         yelp_domains='|'.join(re.escape(d) for d in country_to_base_domain.values()),
674         not_regname=RFC3986.re.not_regname + "\%",
675     ),
676     re.VERBOSE | re.UNICODE | re.IGNORECASE,
677 )
```

One of the reviews on that page:

“Was.attended.quickly.I.had.a.good.time.talking.with.everyon.I.was.very.happy.with.the.service.I.know.I.will.surely.miss.this.place.when.I.move.I.went.in.there.for.simple.&.I.got.cute.simple.I.recommend.this.place”

```
$ time python -c '\
    import config.domain; \
    config.domain.yelp_hostname_regex.match(
        "a.b.c.d.e.f.g.h.i.j.k.l.m.o.p.q.r.s.t.u.v.w.x.y.z"\
    )'
real    0m8.392s
user    0m8.299s
sys     0m0.056s
```

```
$ time python -c '\
    import config.domain; \
    config.domain.yelp_hostname_regex.match(
        "a.b.c.d.e.f.g.h.i.j.k.l.m.o.p.q.r.s.t.u.v.w.x.y.z"\
    )'
real    0m8.392s
user    0m8.299s
sys     0m0.056s
```

The fix? One character!

649 lines

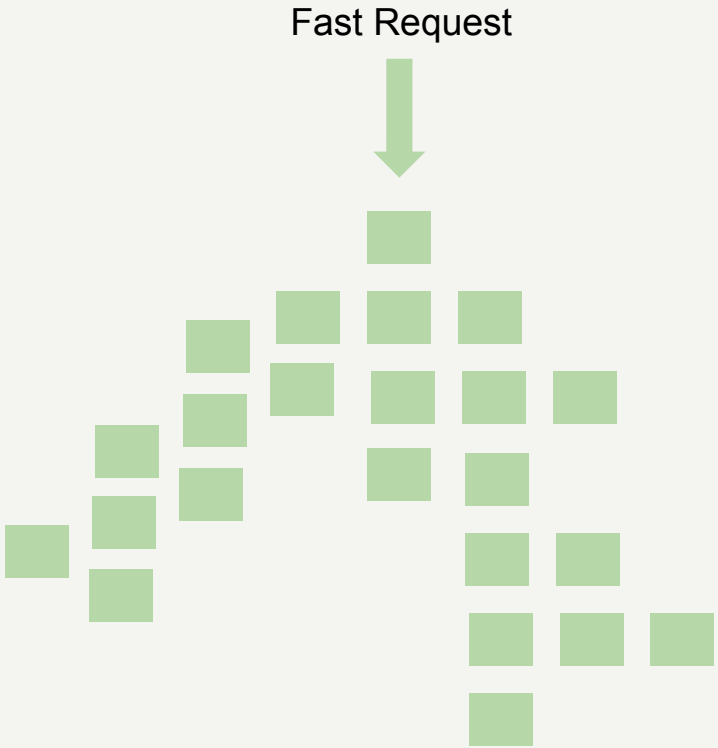
20 — def consume_base_domain(cls, domain_string):

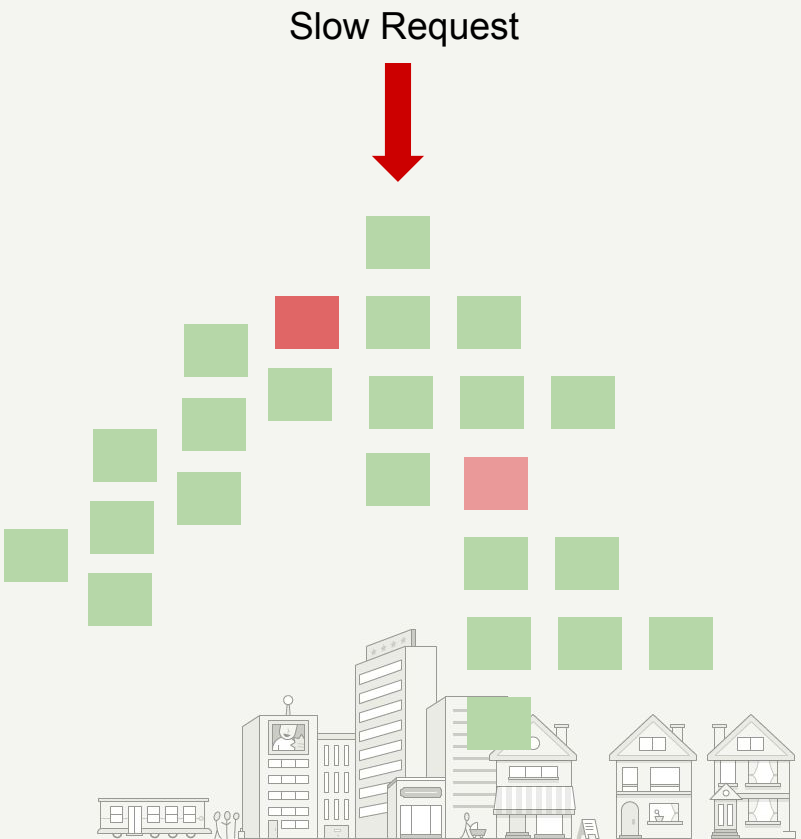
```
650
651 # log parsing stuff
652 yelp_hostname_regex = re.compile(
653     r"""
654         ^(                          # start of string
655             [^%(not_regname)s]+\.   # the not-not pattern allows for unicode characters.
656         )*                          # any number of domain segments
657         (%(yelp_domains)s)          # one of the known yelp domains
658         \.?                         # optionally followed by a dot (?!)
659         $                           # end of string
660     """ % dict(
```

```
650
651 # log parsing stuff
652 yelp_hostname_regex = re.compile(
653     r"""
654         ^(                          # start of string
655             [^.%(not_regname)s]+\.  # the not-not pattern allows for unicode characters.
656         )*                          # any number of domain segments
657         (%(yelp_domains)s)          # one of the known yelp domains
658         \.?                         # optionally followed by a dot (?!)
659         $                           # end of string
660     """ % dict(
```

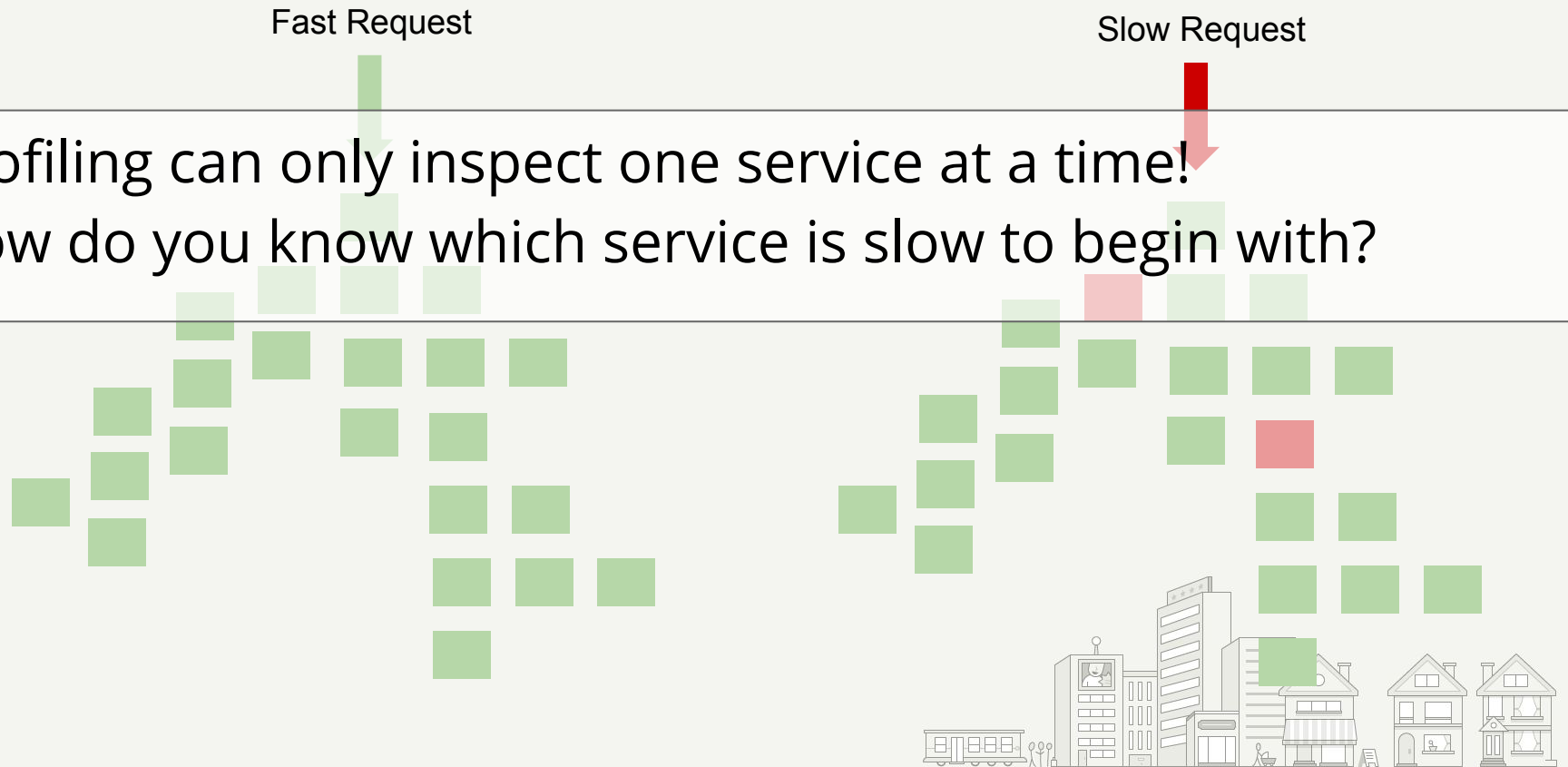
20 — 141 lines

The limits of profiling





The limits of profiling



- Profiling can only inspect one service at a time!
- How do you know which service is slow to begin with?

Tracing

Image Credits: Will Murai

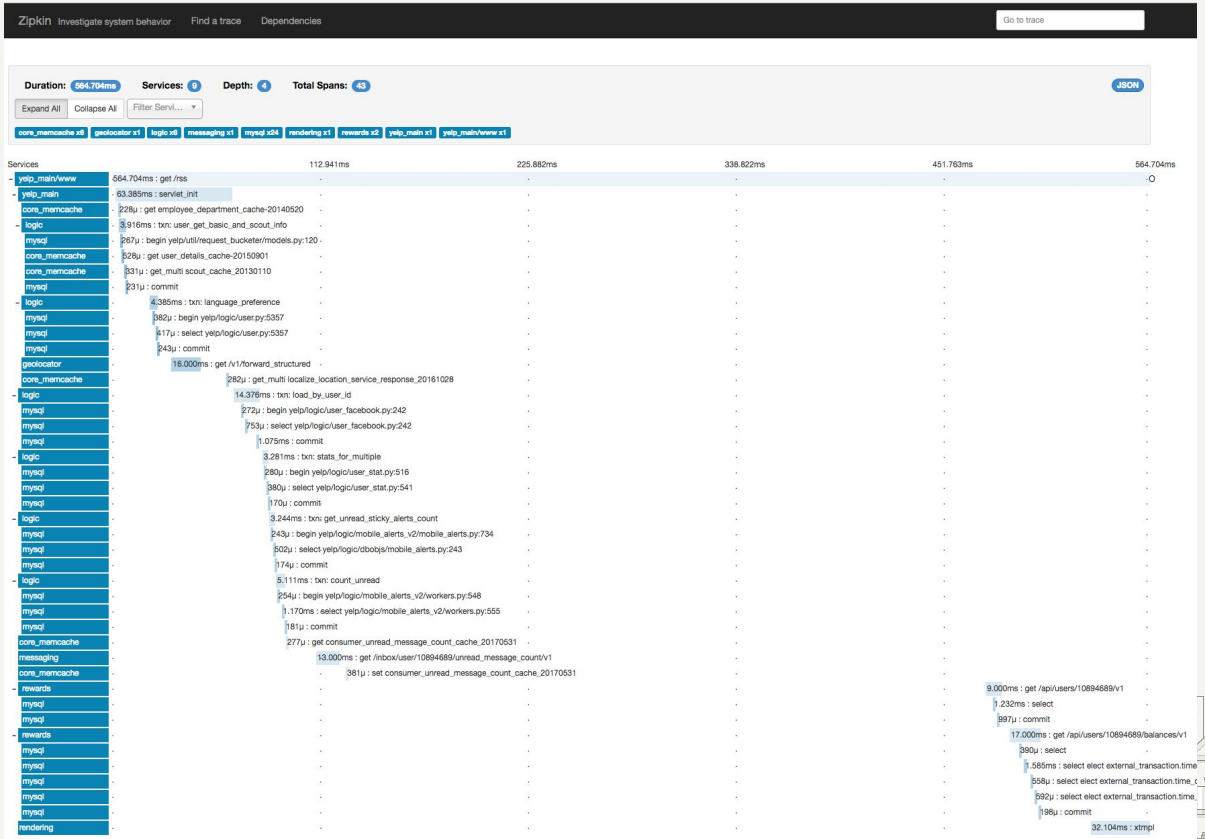
What is tracing?

- Tracing == profiling, for distributed systems
- This lets us follow a request from the moment it enters our infrastructure up until a response is returned

Tracing at Yelp

- We use Zipkin, an open-source project originally built by Twitter engineers
- Zipkin is a system to log, collect and aggregate information about traced requests

A normal Zipkin trace



A normal Zipkin trace

Performance / PERF-297

Reservation provider cache hit many times in short succession on /search

Edit | Comment | Assign | More | Backlog | Selected for Development | Workflow

Details

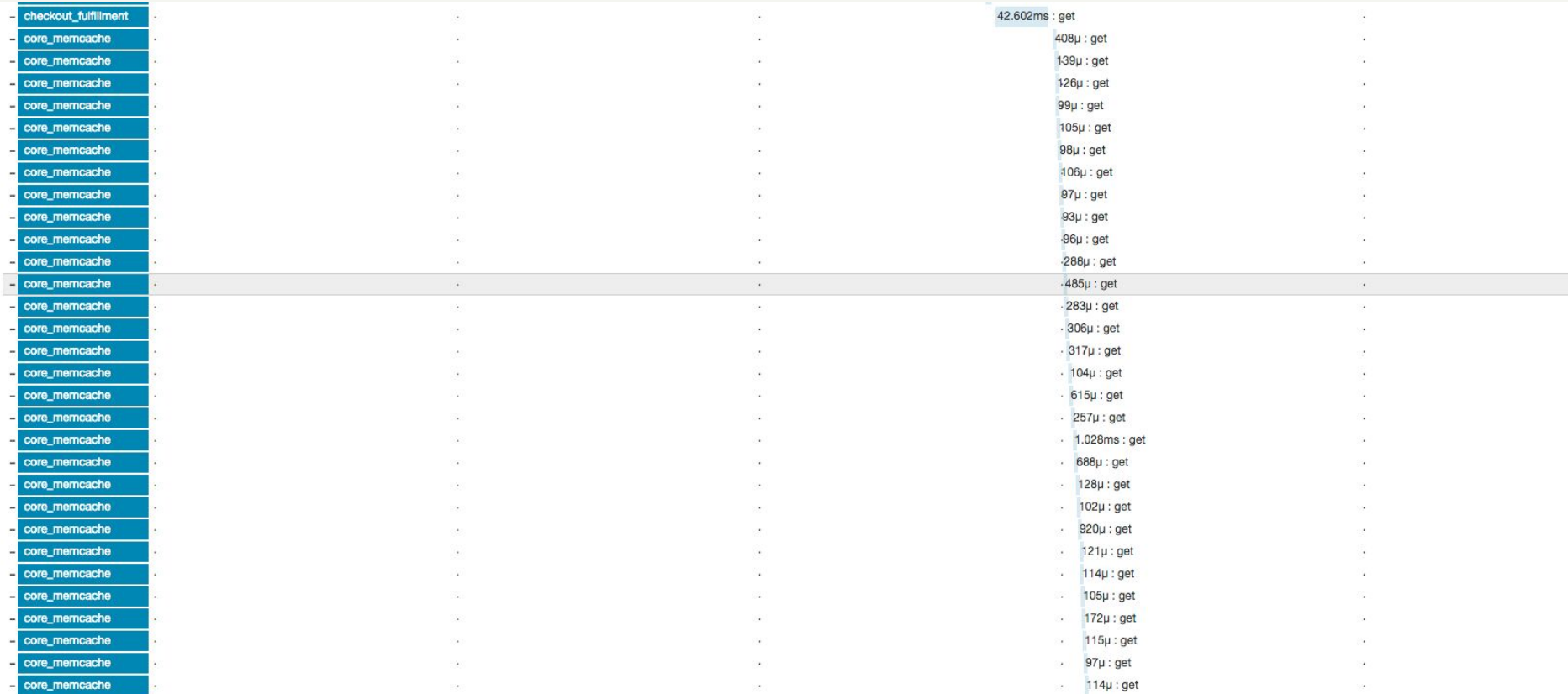
Type:	Improvement	Status:	DONE (View Workflow)
Priority:	2 Do next	Resolution:	Done
Component/s:	Tuning		
Labels:	None		
Epic Link:	Memcache Optimizations		

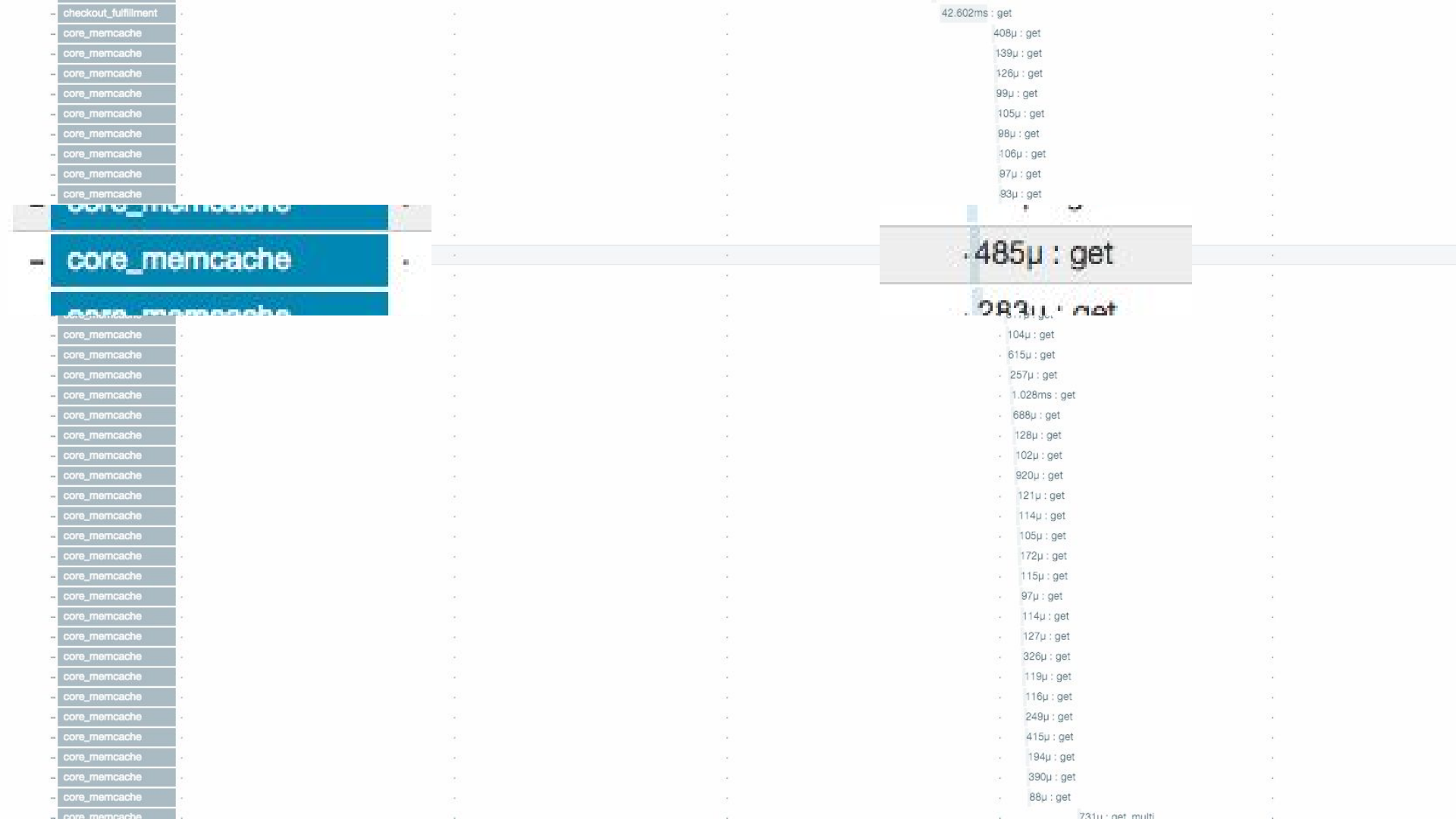
Description

Zipkin-traced memcache calls in yelp-main revealed that this code, called in the context of a /search hit, hits the reservation memcache many many times in short succession. Here's the relevant Zipkin information (from this trace).

This isn't a huge perf problem - in this example all these individual calls totaled 33ms - but we should still be able to do this much more efficiently with a `get_multi` call. Could probably get it down to ~2ms or so, with much less network traffic and log line chatter.

(Part of) the abnormal Zipkin trace




checkout_fulfillment
core_memcache
42.602ms : get
408µ : get
139µ : get
126µ : get
99µ : get
105µ : get
98µ : get
106µ : get
97µ : get
93µ : get
485µ : get
104µ : get
615µ : get
257µ : get
1.028ms : get
688µ : get
128µ : get
102µ : get
920µ : get
121µ : get
114µ : get
105µ : get
172µ : get
115µ : get
97µ : get
114µ : get
127µ : get
326µ : get
119µ : get
116µ : get
249µ : get
415µ : get
194µ : get
390µ : get
88µ : get

The problem? O(n) memcache calls

```
1 cache = CacheSystem('mycache')
2
3 for review in reviews:
4     render(review_cache[review['id']])
5           #^^^^^^^^^^^^^^^^^^^^^^^^^^ One network call! For each review!
6
```

Solution? GET_MULTI

```
1 cache = CacheSystem('mycache')
2
3 reviews_list = [review['id'] for review in reviews]
4 render(review_cache.get_multi(review_list))
5       #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ A single network call
6
```

Caution!

Image Credits: Blizzard Inc.

Avoid premature optimizations

- Make your code work
- Make your code clean
- THEN measure
- THEN make your code faster (if it’s too slow)

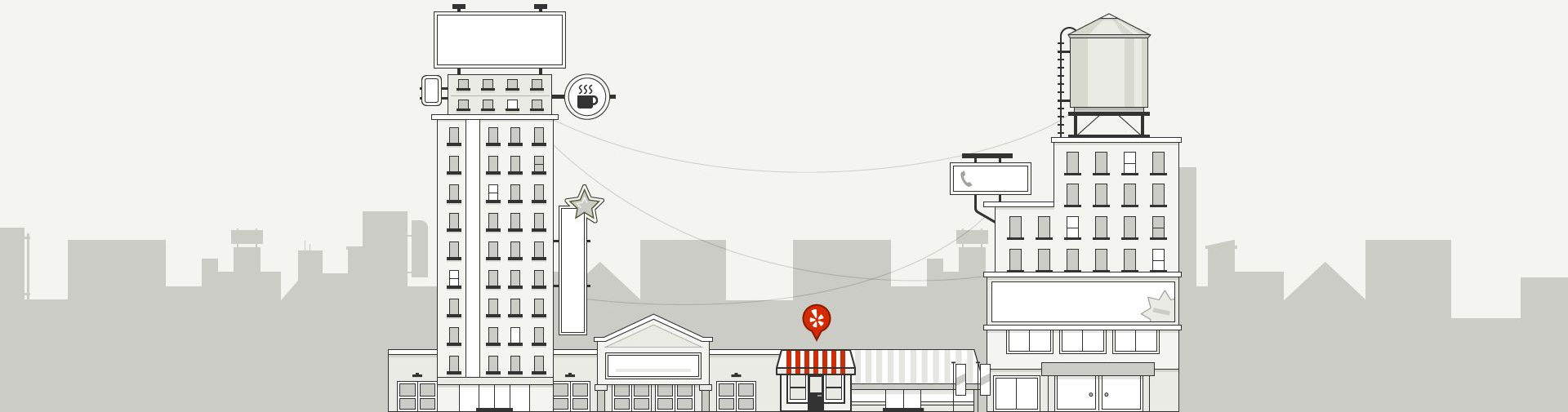
Monitoring, alerting

- Yelp has many many pages/views. Looking at profile/traces of all of them isn't scalable
- If something gets slower, someone should notice, quickly!

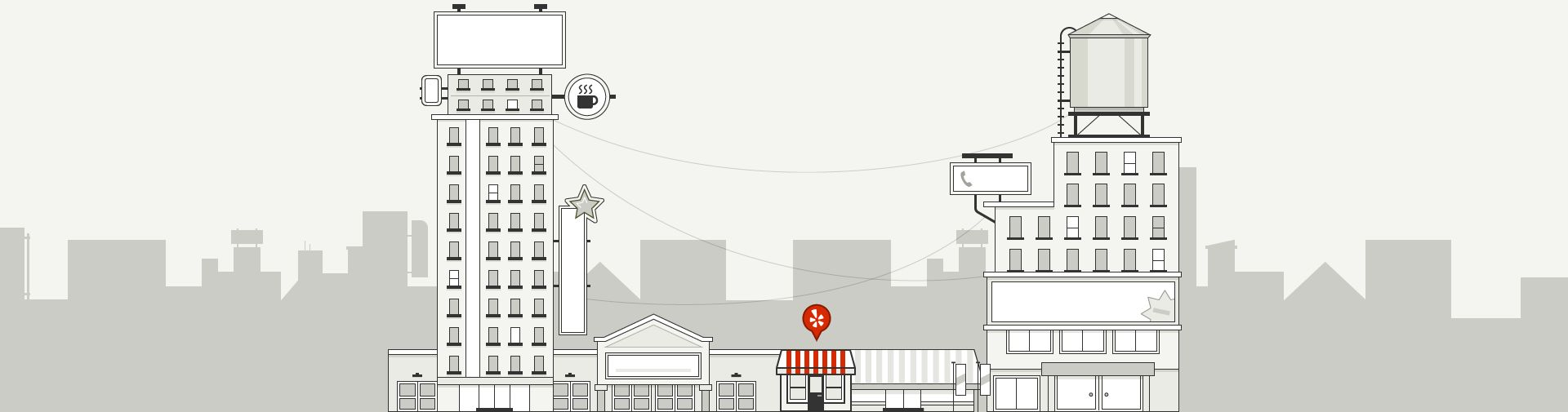
⇒ Automated monitoring/alerting is crucial

⇒ I can go into more details if you're interested, during Q&A

Q&A time!



Extra slides



```python
from __future__ import print_function
import sys

def add(a, b):
    return a + b

def multiply(a, b):
    total = 0
    for i in range(b):
        total = add(total, a)
    return total

if __name__ == "__main__":
    a = int(sys.argv[1])
    b = int(sys.argv[2])
    result = multiply(a, b)
    print('{a}*{b} ==> {result}'.format(a=a, b=b, result=result))

```

A “convenient” abstraction: CacheSystem

```
1  class CacheSystem(object):
2      def __init__(self,...):
3          # omitted for brevity
4
5
6      def __getitem__(self, key):
7          Return self.client.get_value(self.cache_name, key)
8
9  cache = CacheSystem(‘mycache’)
10 print(cache[‘mykey’]) # returns the associated value!
11
```

Monitoring & alerting deets

- App performance: custom logs
- Web performance: window.performance.timings
- Server performance: logs at multiple levels, then aggregation and display with SignalFx
- External monitoring with Catchpoint, a third-party tool to let us ping our site/apps from everywhere around the world (an enterprise version of webpagetest)

4426

You can't parse [X]HTML with regex. Because HTML can't be parsed by regex. Regex is not a tool that can be used to correctly parse HTML. As I have answered in HTML-and-regex questions here so many times before, the use of regex will not allow you to consume HTML. Regular expressions are a tool that is insufficiently sophisticated to understand the constructs employed by HTML. HTML is not a regular language and hence cannot be parsed by regular expressions. Regex queries are not equipped to break down HTML into its meaningful parts. so many times but it is not getting to me. Even enhanced irregular regular expressions as used by Perl are not up to the task of parsing HTML. You will never make me crack. HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow *it is too late it is too late we cannot be saved* the trangession of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) *dear lord help us how can anyone survive this scourge* using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes *using regex* as a tool to process HTML establishes a bre*ach between this world* and the dread realm of corrupt entities (like SGML entities, but *more corrupt) a mere glimp*se of the world of reg**ex parsers for HTML will ins**tantly transport a p*rogrammer's consciousness i*nto a wo*r*ld of ceaseless screaming, he comes~~, the pestilent sli~~thy regex-infection wi**ll devour your HT**ML parser, application and existence for all time like Visual Basic only worse *he comes he com*es *do not f*ight h**e comes, hi**s unholy radiańcé de*stroying all enlightenment, HTML tags **leaking from your eyes like liq**uid p*ain, the song of regular expr~~ession parsing~~ will ext*inguish the voices of mor**tal man from the sp**here I can see it can you see it it is beautiful t*he f `inal snuf` fing o*f the lie**s of Man ALL IS LOST AL**L IS LOST the *pony he com*es he comes~~the comes the~~ ich*or permeates a*ll MY FAC*E MY FACE oh god n**o NO NOOOO N**O stop t*he an*gles are **not real ZALGO IS TONY THE PONY HE COMES**

Have you tried using an XML parser instead?
