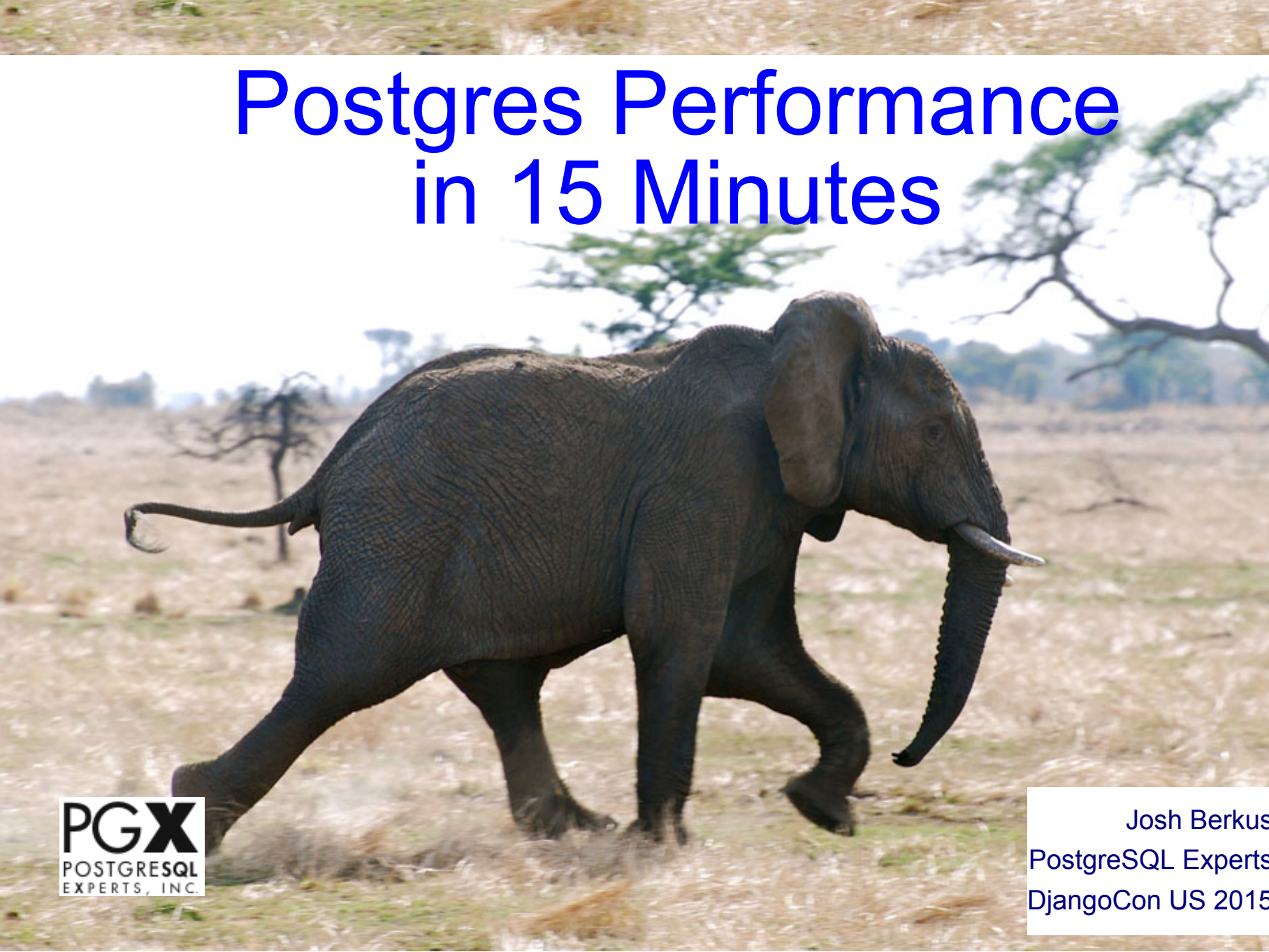
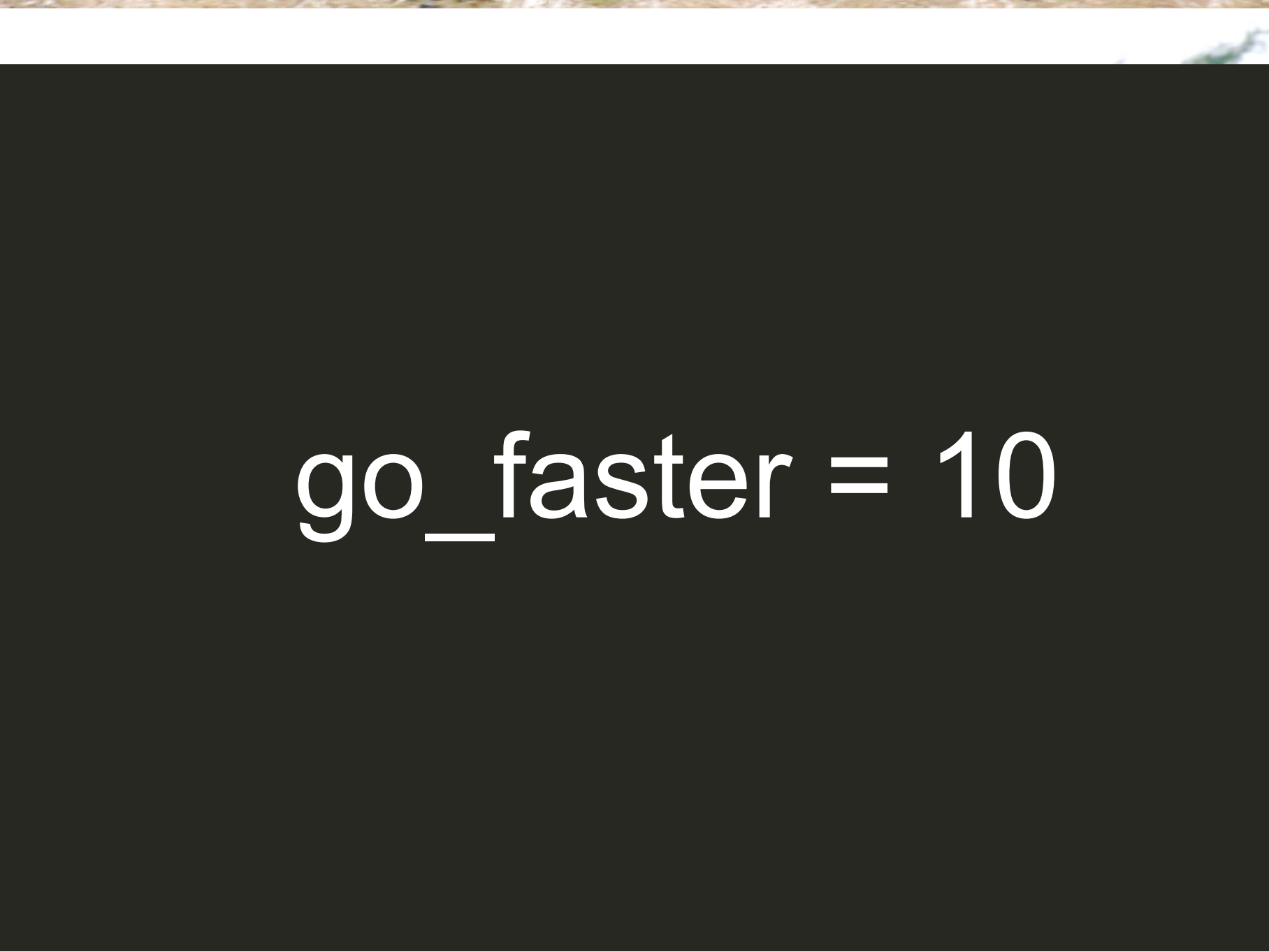
Postgres Performance in 15 Minutes

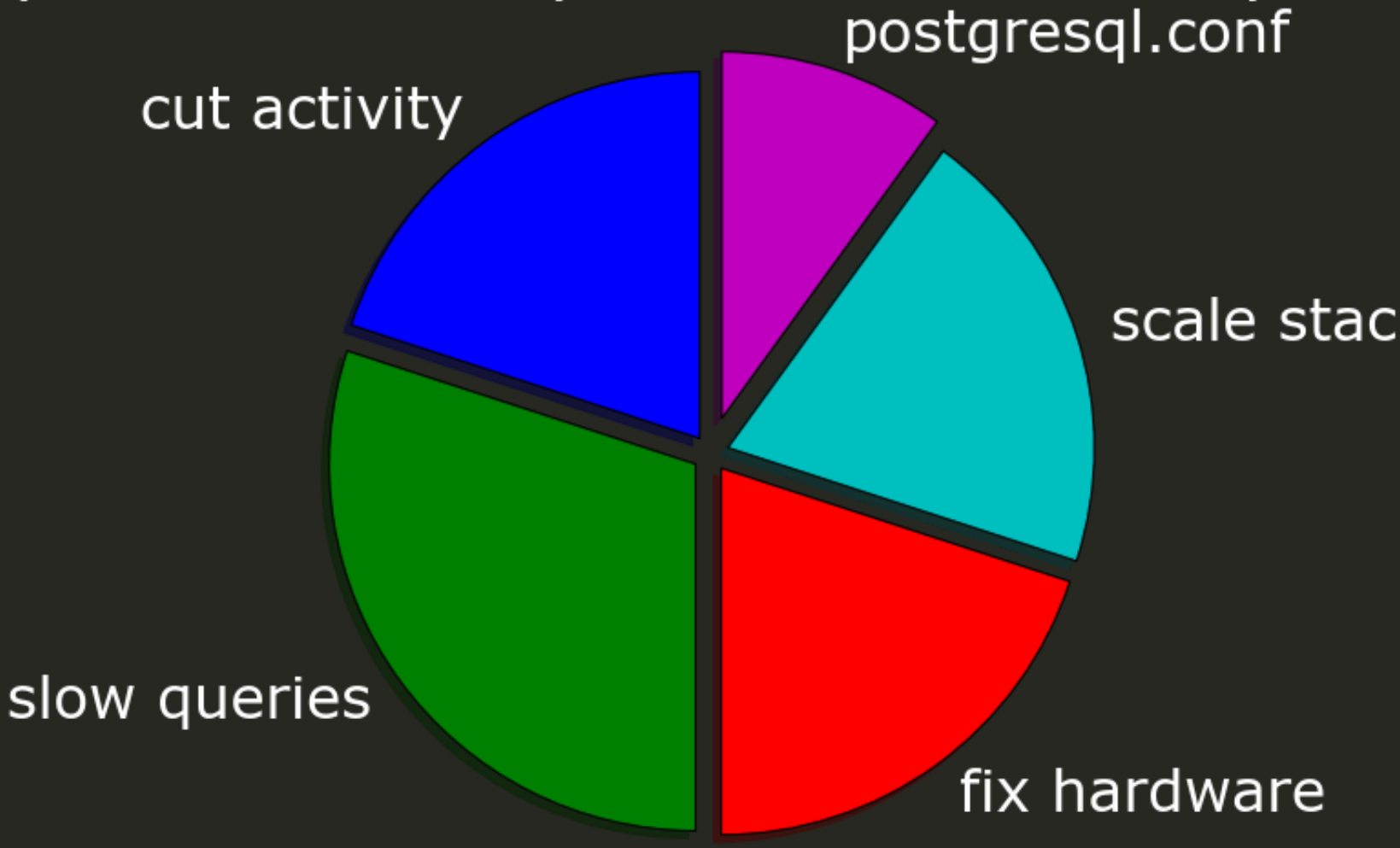


PGX
POSTGRESQL
EXPERTS, INC.

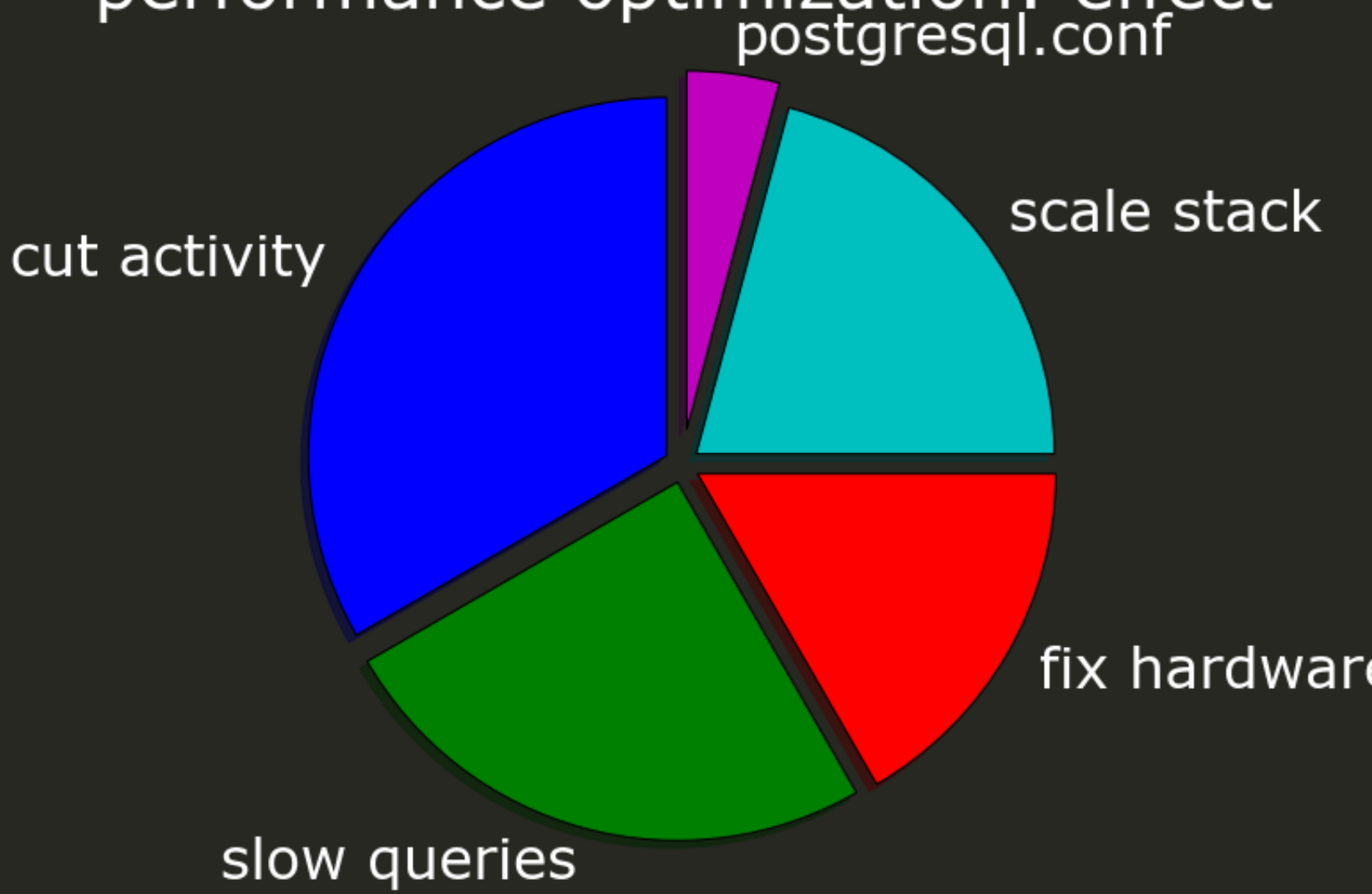
Josh Berkus
PostgreSQL Experts
DjangoCon US 2015


go_faster = 10

performance optimization: time spent



performance optimization: effect



Do Less

do I need the database to answer that?

caching

- results cache
- redis
- memcached
- CDN

some data access anti-patterns

polling

```
while True:
    j = check_for_job(pid)
    if j is None:
        continue
```

data you already have

```
curUser = Users.object.filter(pk=sessionUser).values()
curUserID = curUser[0]["id"]
```

data you already have

```
SELECT id FROM users
WHERE id = ?
```

data you don't need

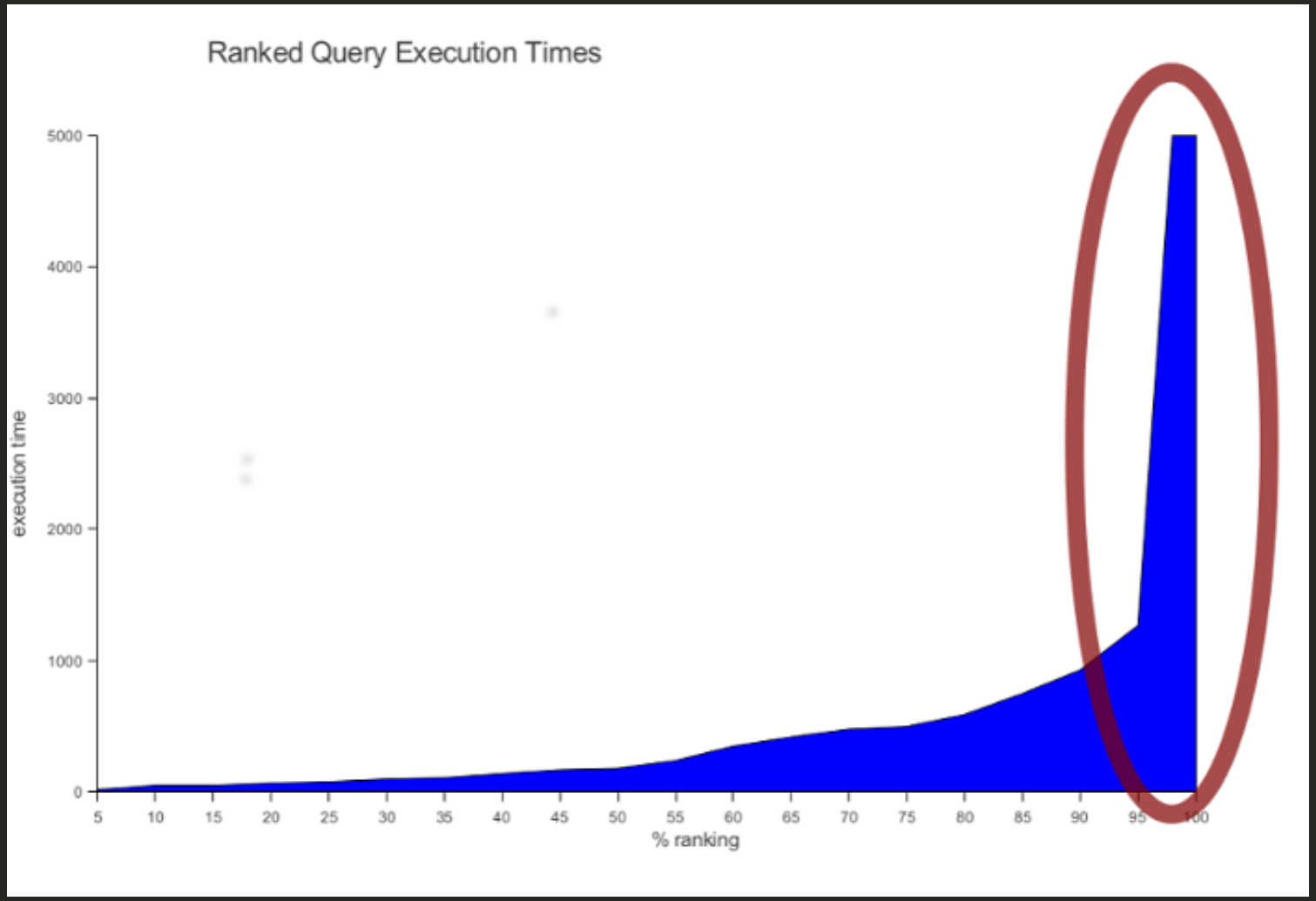
```
AllProfiles = Profile.objects.all().order_by('-updated')
LastProfile = AllProfiles[0]
```

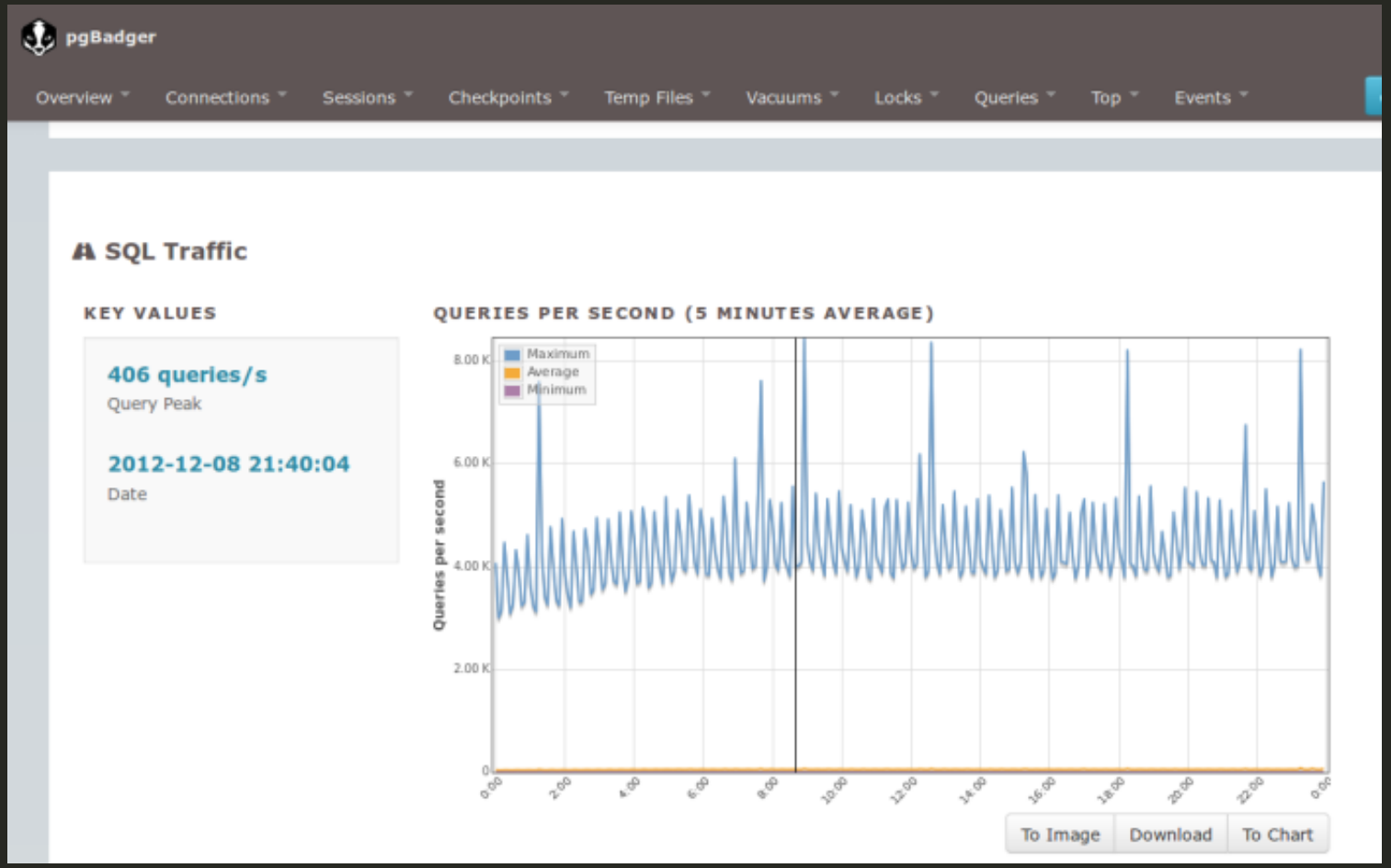
join loops

```
for Player in Roster:
    myGames =
      Games.object.filter(player_id=Player["id"]).values(
```

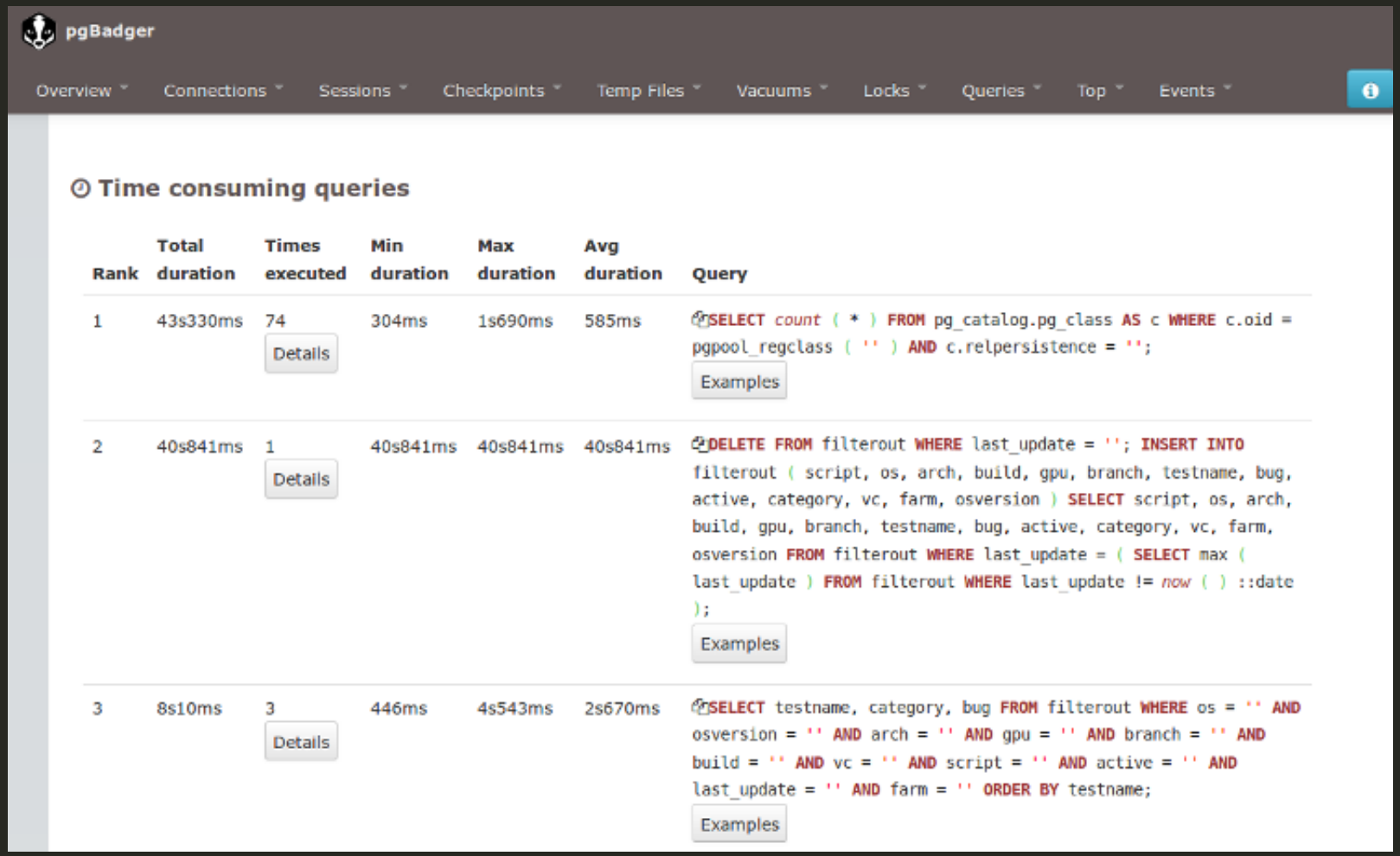
```
SELECT * FROM games WHERE player_id = ?
-- repeat 200X
```

resource-hungry queries


Ranked Query Execution Times
execution time
5000
4000
3000
2000
1000
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
% ranking


pgBadger
Overview
Connections
Sessions
Checkpoints
Temp Files
Vacuums
Locks
Queries
Top
Events
SQL Traffic
KEY VALUES
406 queries/s
Query Peak
2012-12-08 21:40:04
Date
QUERIES PER SECOND (5 MINUTES AVERAGE)
Maximum
Average
Minimum
Queries per second
8.00 K
6.00 K
4.00 K
2.00 K
0
0:00
2:00
4:00
6:00
8:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
0:00
To Image
Download
To Chart

pgBadger


pgBadger
Overview
Connections
Sessions
Checkpoints
Temp Files
Vacuums
Locks
Queries
Top
Events
Time consuming queries
Rank
Total duration
Times executed
Min duration
Max duration
Avg duration
Query
1
43s330ms
74
Details
304ms
1s690ms
585ms
SELECT count (*) FROM pg_catalog.pg_class AS c WHERE c.oid = pgpool_regclass ('') AND c.relpersistence = '';
Examples
2
40s841ms
1
Details
40s841ms
40s841ms
40s841ms
DELETE FROM filterout WHERE last_update = ''; INSERT INTO filterout (script, os, arch, build, gpu, branch, testname, bug, active, category, vc, farm, osversion) SELECT script, os, arch, build, gpu, branch, testname, bug, active, category, vc, farm, osversion FROM filterout WHERE last_update = (SELECT max (last_update) FROM filterout WHERE last_update != now ( ) ::date);
Examples
3
8s10ms
3
Details
446ms
4s543ms
2s670ms
SELECT testname, category, bug FROM filterout WHERE os = '' AND osversion = '' AND arch = '' AND gpu = '' AND branch = '' AND build = '' AND vc = '' AND script = '' AND active = '' AND last_update = '' AND farm = '' ORDER BY testname;
Examples

pgBadger

improving slow queries

- add indexes
- fix filter expressions
- analyze/autoanalyze

text searching

startswith: requires varchar_pattern_ops

istartswith: requires function index or CItext

icontains: use TSearch or trigrams

transaction pitfalls

bad: bundling reads into a transaction

worse: holding a transaction open while rendering

worst: holding a transaction open while waiting for user input

locking

check for "waiting" queries:

```
SELECT * FROM pg_stat_activity WHERE waiting;
```

does your program logic have multiple threads update the same data at the same time?

adequate hardware

and virtual hardware

You can't outperform inadequate hardware

IO is more important than you think

postgres writes all the time

- COMMITs
- replication
- "hint bits"
- statistics

IOPS + throughput == performance

optimizing IO

own HW: use SSDs, HW RAID

cloud: get more IOPS, EBS-optimized

also: No More EXT3! (or HFS or NTFS)

and: some Linux Kernel issues (3.2, 3.5)

RAM usage is thresholded

good: enough to cache "working set"

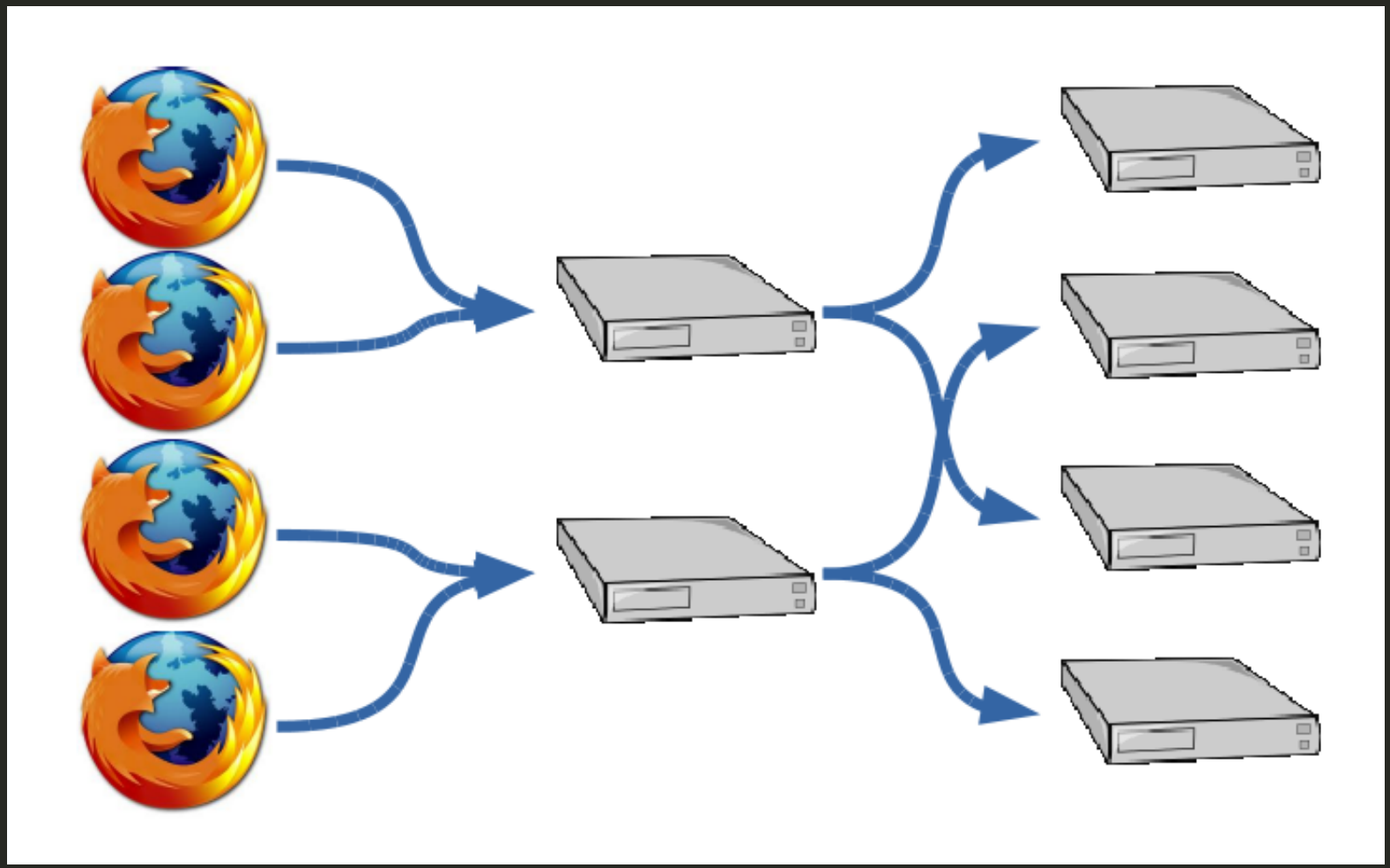
better: cache the whole database

best: DB fits in shared_buffers

some AWS tips

- use GP2 + extra GB
- PAAS != performance
- check zone distance

scaling infrastructure



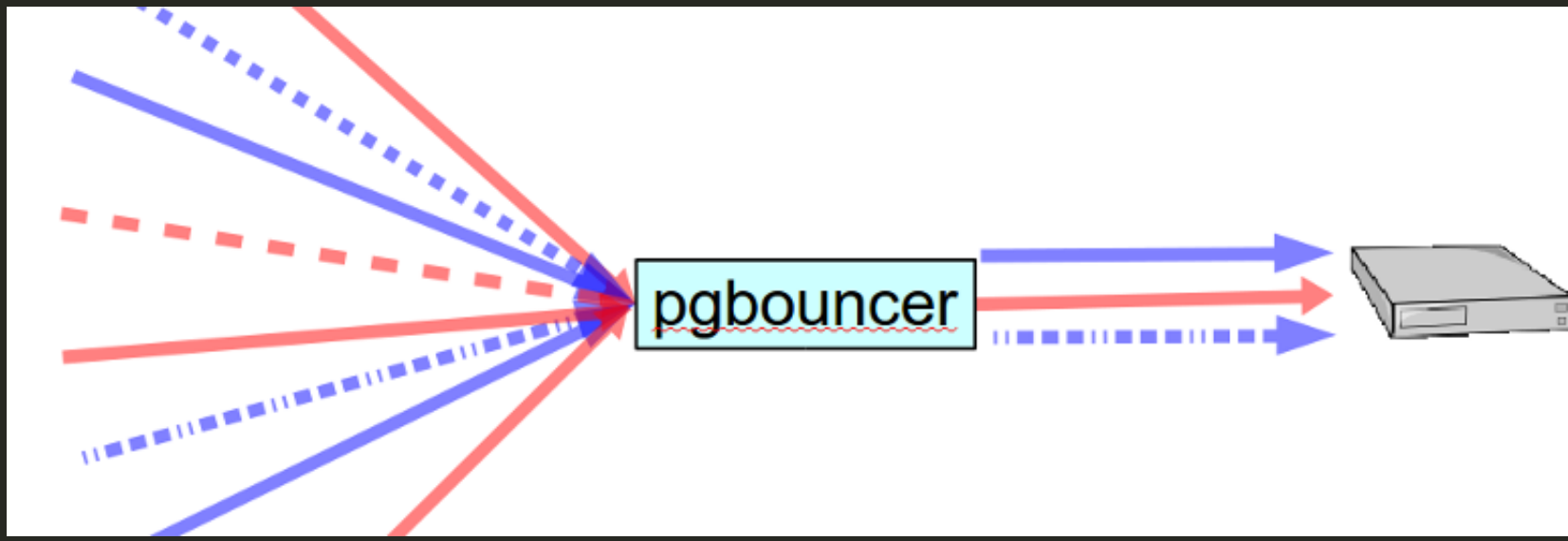
the easy stuff

use the latest version of PostgreSQL
perf improvements in every release

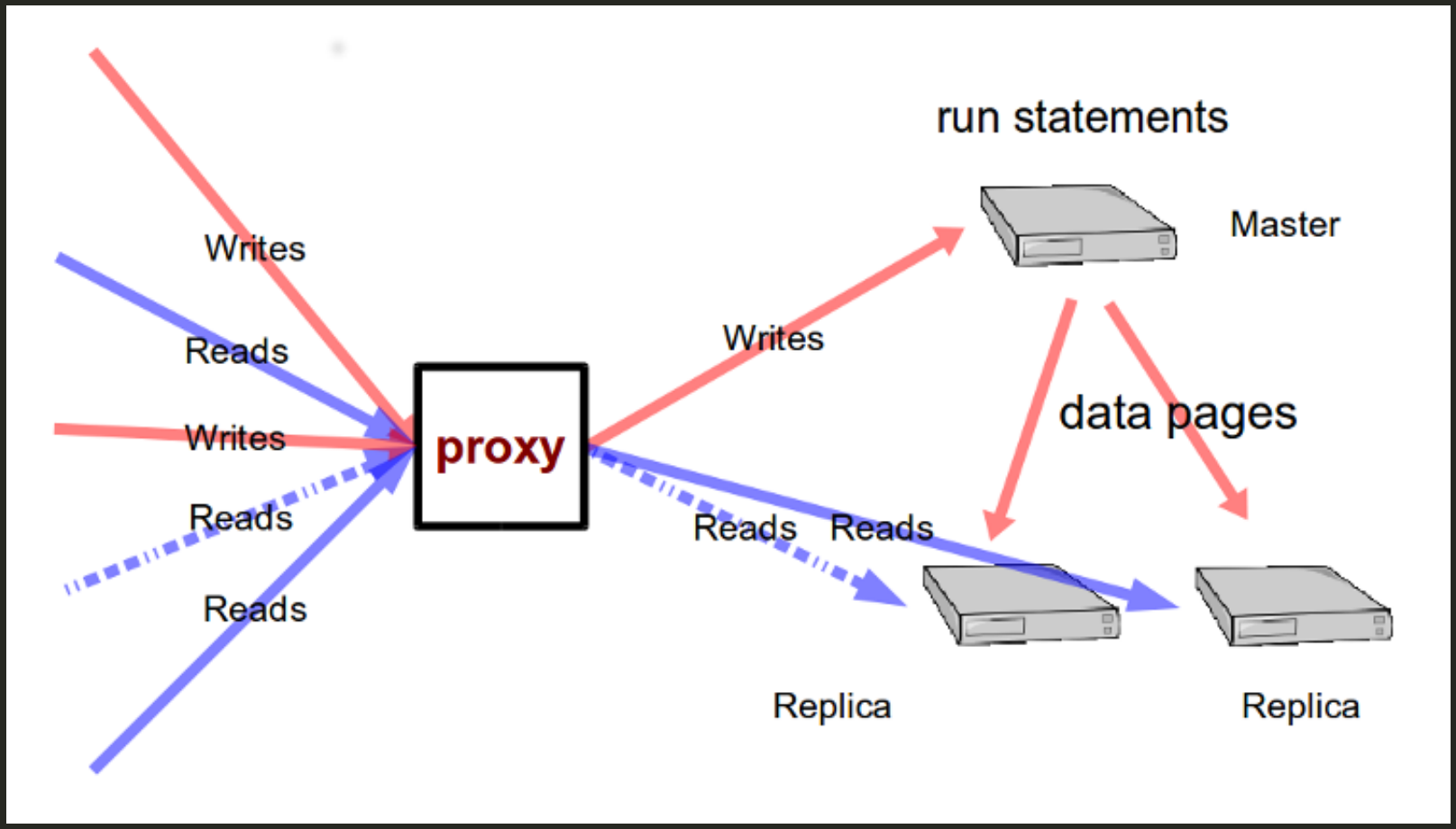
put PostgreSQL on its own server/instance
databases do not share well

use pgbouncer for connection pooling

```
pool_mode = transaction
```



event-based pooling



load balance to read slaves

load balancing using routes

```
DATABASES = {
'master': {
    'NAME': 'master',
    'ENGINE': 'django.db.backends.pgsql'
},
'replica1': {
    'NAME': 'replica1',
    'ENGINE': 'django.db.backends.pgsql'
} ...
```

load balancing special workloads

- reporting
- cache refresh
- queueing (separate DB)

postgresql.conf

memory use

```
shared_buffers = 2GB
# RAM/4 up to 8GB

work_mem = 32MB
# 8MB to 32MB: web
# 128MB to 1GB: reporting
# limit: RAM/(max_connections/2)
```

more memory use

```
effective_cache_size = 6GB
# 3/4 of RAM

wal_buffers = 64MB
# just set it

maintainence_work_mem = 512MB
# RAM/32
# more for reporting
```

WAL

```
checkpoint_segments = 64
# make WAL bigger
# space / 32MB

checkpoint_completion_target = 0.9
```

more settings

```
stats_temp_directory = '/mnt/ramdisk'
# helps with latency

random_page_cost = 1.5
# for AWS, SSD

effective_io_concurrency = 4
# for AWS, SSD, RAID
```

logging settings

```
log_connections = on
log_disconnections = on
log_temp_files = 1kB
log_lock_waits = on
log_checkpoints = on
log_min_duration_statement = 0
```

recap

- Do Less querying
- fix resource-hungry requests
- get adequate hardware
- scale your infrastructure
- tune the config a little

questions?

more www.pgexperts.com
jberkus: www.databasesoup.com

more austinPUG Thursday night:
events: www.meetup.com/austinpug

Postgres Open:
Sept 16-18, Dallas

pgConfSV:

PGX
POSTGRESQL
EXPERTS, INC.