Realistic Load Testing

Setting up a realistic testing environment
using PostgreSQL and Python

Who Are You?

- DBA—Architecture
- DBA—Administration
- Database Developers
- Application Developers
- Web Developers
- Managers
- Other?

Why Are You Here?

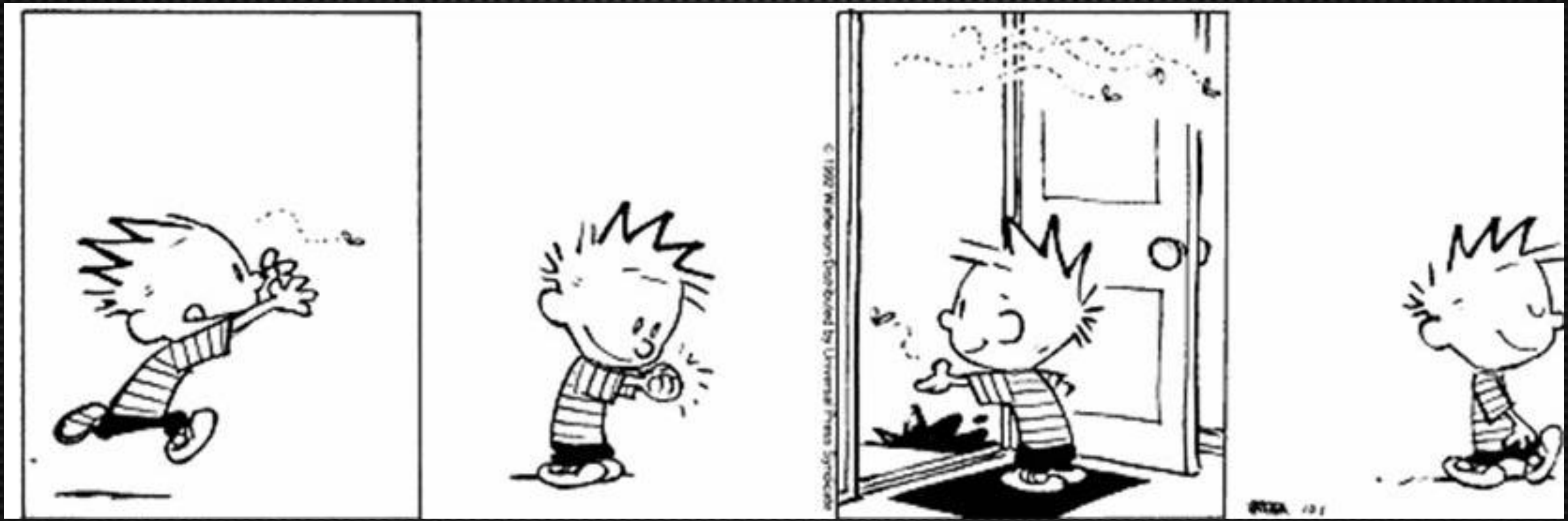
- You want a process of testing to gain confidence in application release
- You want to have proof of meeting performance goals to stakeholders
- You have an SLA that demands certain performance expectations
- You want to see another approach to load testing

Why Load Testing?

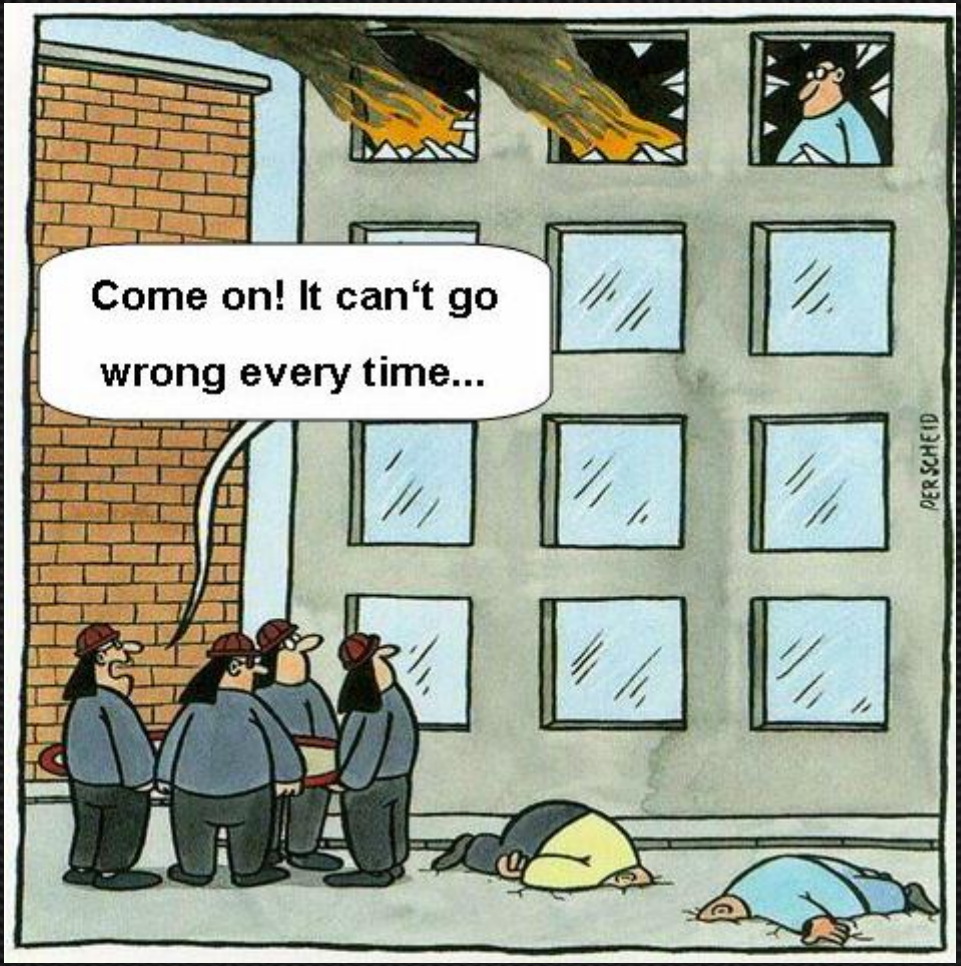
- Detect software failures
- Detect performance thresholds
- Detect integration failures
- Detect proper configurations
- Optimization of Hardware / Software
- Determine if application is Good-To-Go

The purpose of Load Testing is to *simulate a system load* and measure the *user experience* to determine if the performance goals were met.

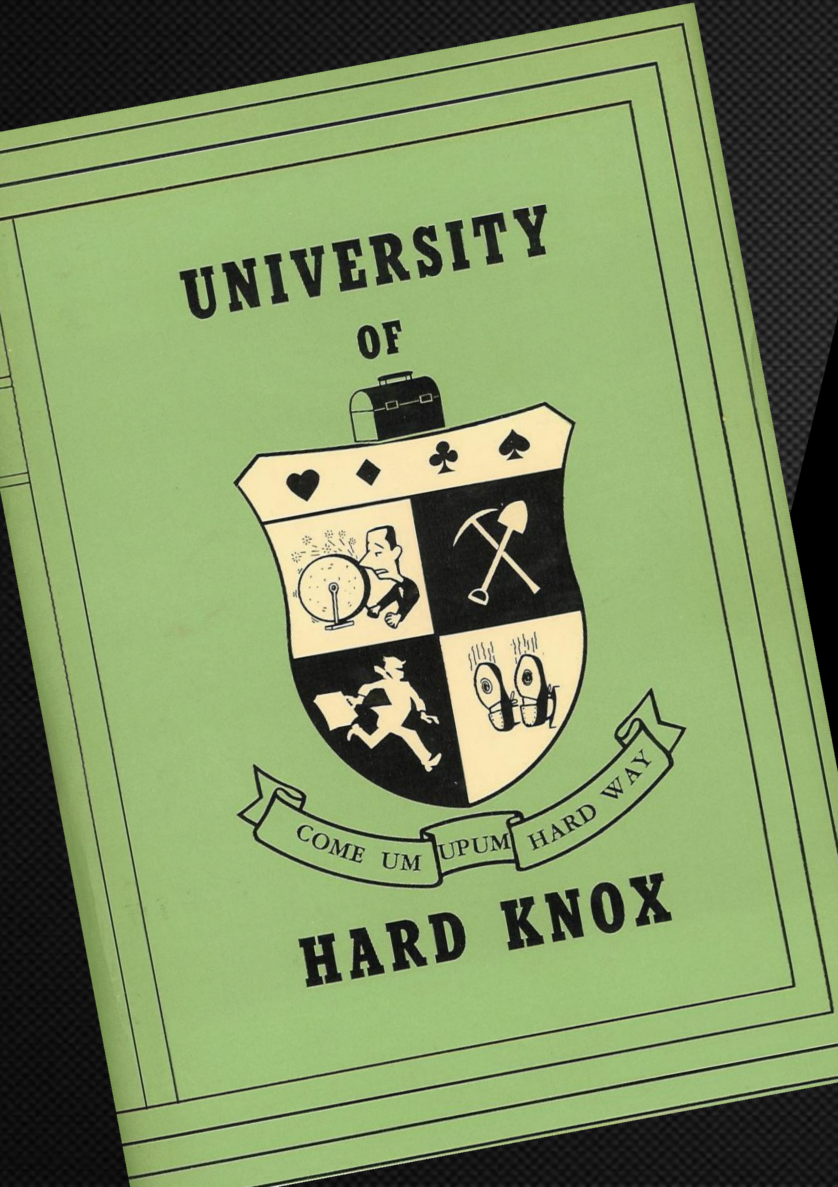
The purpose of Integration Testing is to *test performance* and *identify problems* that occur when all processes used to provide the *user experience* are combined as a system.

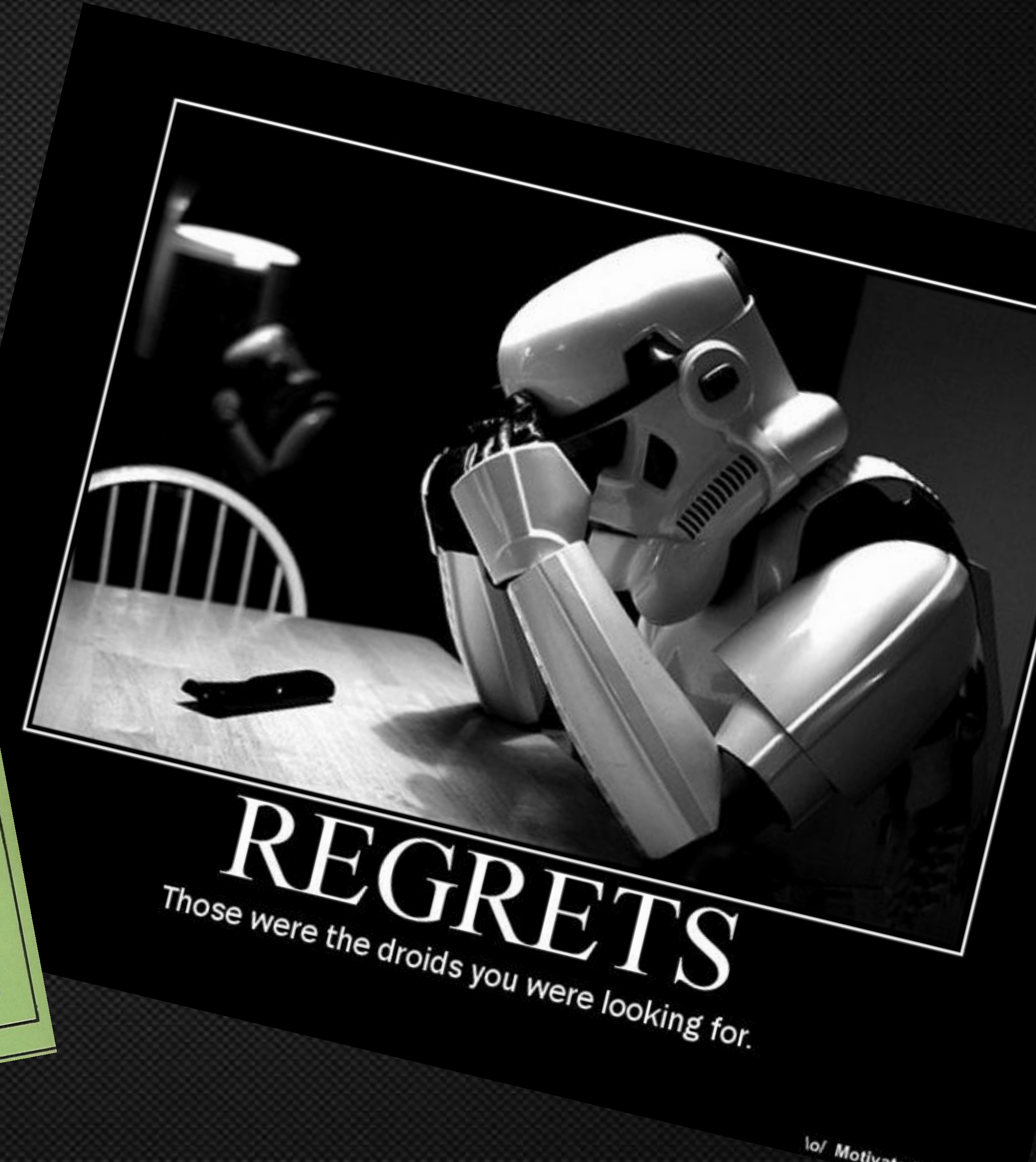


© 1992 Watterson/Distributed by Universal Press Syndicate


Come on! It can‘t go wrong every time...
PERSCHEID

School of Hard Knocks

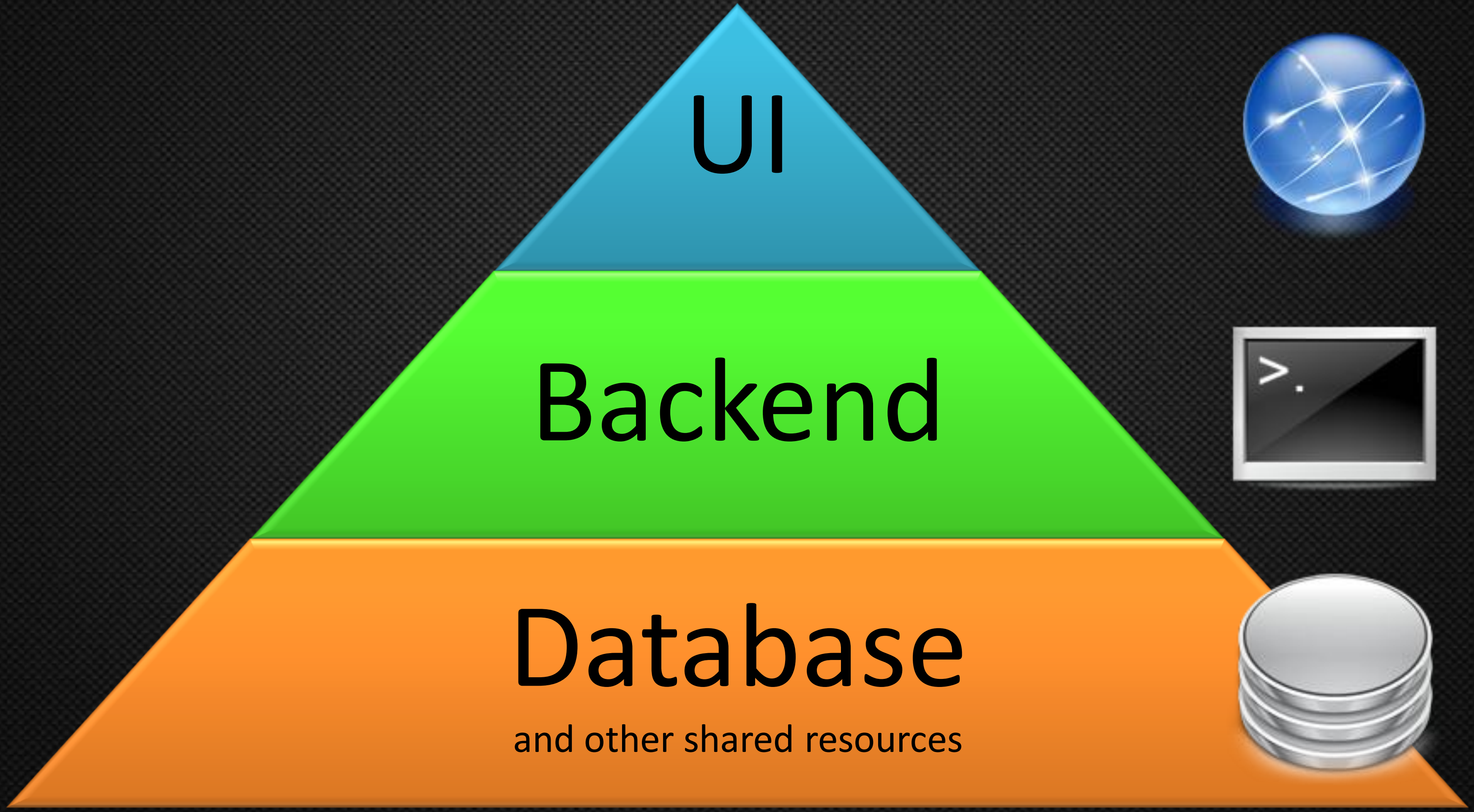




What Will We Cover?

- Shortfalls of FLOSS benchmark tests
- Identifying Test Components
- Identifying Realistic Loads
- Identifying the Dataset
- Developing Tests and Procedures
- PostgreSQL Functions for Tests
- Python Scripts for Tests
- Helpful Tools

Typical Application Stack



UI

Backend

Database

and other shared resources

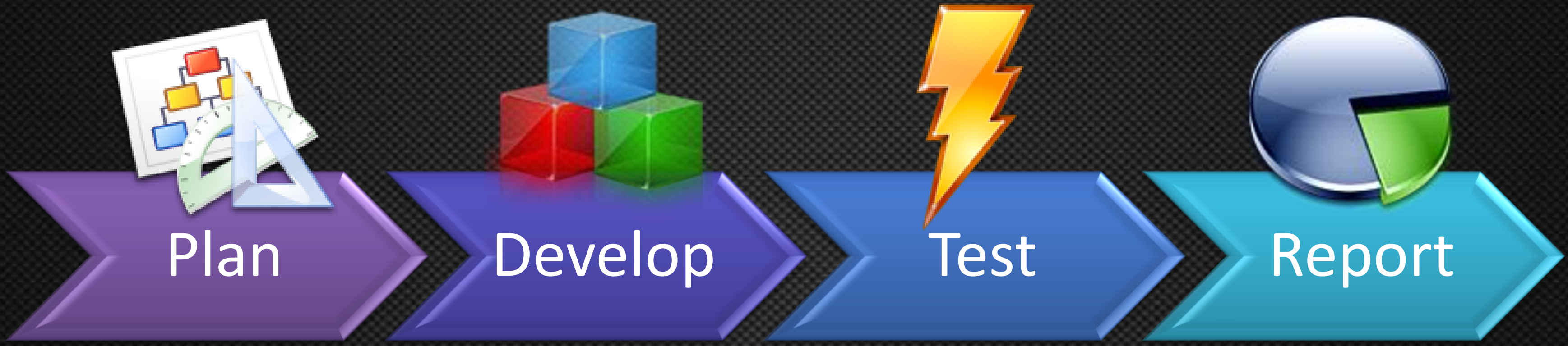
Shortfalls of FLOSS Tests

- It is not your application
- Results are not always explained
- Reporting can be cryptic or unhelpful
- Does not fit all your needs
- Does not use a realistic data set

Example FLOSS Testing Tools

- Database
  - pgbench
  - Tsung (Erlang)
  - pgReplay
- Backend
  - Included application tests
- Frontend
  - Tsung
  - jMeter
  - Ab
  - Siege
  - Selenium
  - Funkload

The Test Process



Plan

Develop

Test

Report

Identifying Test Components

Database + Backend + Frontend

Identifying Test Components

Database

- Are there complex architectural designs that warrant special testing (e.g. Triggers, Functions) for performance?
- Are there specific tables that are heavily written to that may affect performance?
  - Partitioned Tables
  - Data Warehousing

Identifying Test Components

Database *continued*

- Are there Materialized Views that may affect performance (e.g. Eager)?
- Are there Indexes that may affect performance?
- What does the data look like?
  - Typical dataset
  - Average row size
- Identify "heavy" queries

Identifying Test Components

Backend

- What functions of the backend need to be reproduced programmatically?
- What does the data need to look like?
- What scheduled tasks exist?
  - CRON jobs
  - Daemon processes
  - Scheduled events
    - Backups
    - Audits
    - Reports

Identifying Test Components

Frontend

- What functions of frontend need to be reproduced programmatically?
- What does a typical end user do?
  - Record end user interaction
  - Record time usage
  - Identify activities done 80% of the time

Identifying Realistic Loads

Identifying Realistic Loads

Which Perspective?

- The perspective you choose *changes how you will test* and what the results will say about *performance from that perspective*.
- Examples
  - User
  - Account/Client
  - Geographical Location
  - Object of Interest
  - Combination of the above

Identifying Realistic Loads

How Many Users?

- Internal Application
  - Identify size of organization or team
  - Identify forecasted growth
- External Application
  - Identify size of current user base
  - Identify forecasted growth
  - Estimate based on Marketing or Business Plan
  - Estimate based on competitor's or other aspiration's current user base

Identifying Realistic Loads

- How Many Users? *continued*
  - Fixed Usage
    - Subscription based limit
    - Hardware based limit
    - Software based limit
    - I.T. based limit

Identifying Realistic Loads

How Much Data?

- Using the Marketing and Business Plan
  - Forecast Perspective’s Base and Usage
    - At Launch
    - At 1 year
    - At 3 year
  - Forecast Users
    - At Launch
    - At 1 year
    - At 3 year

Identifying Realistic Loads

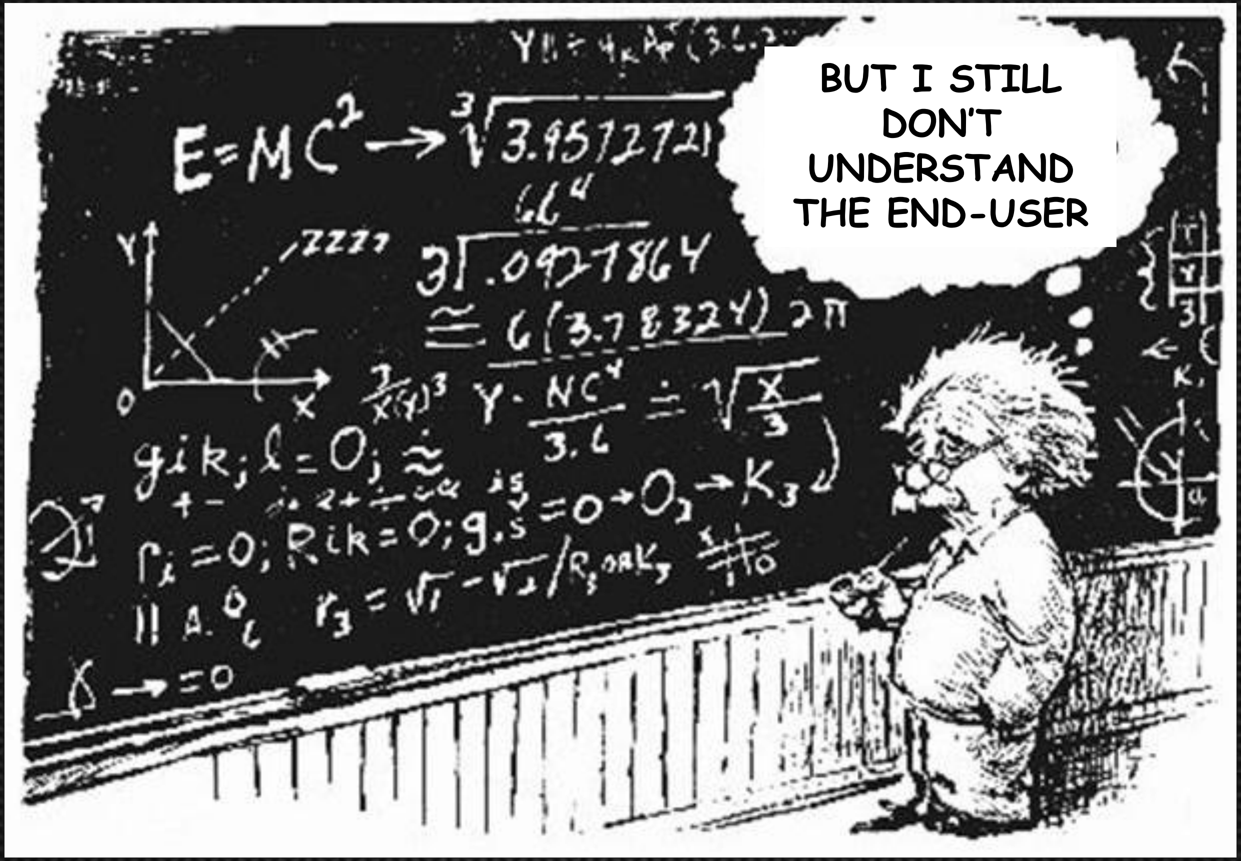
Customers	200	600	1200	3000	6000	10000
RTUs	400	1200	2400	6000	12000	20000
Devices	1300	3900	7800	19500	39000	65000

- A typical use session is about 8 minutes (480 s)
- We assume a 12 hour peak use time during the day (43,200 s)
- We assume # of users = # Contacts (3000)

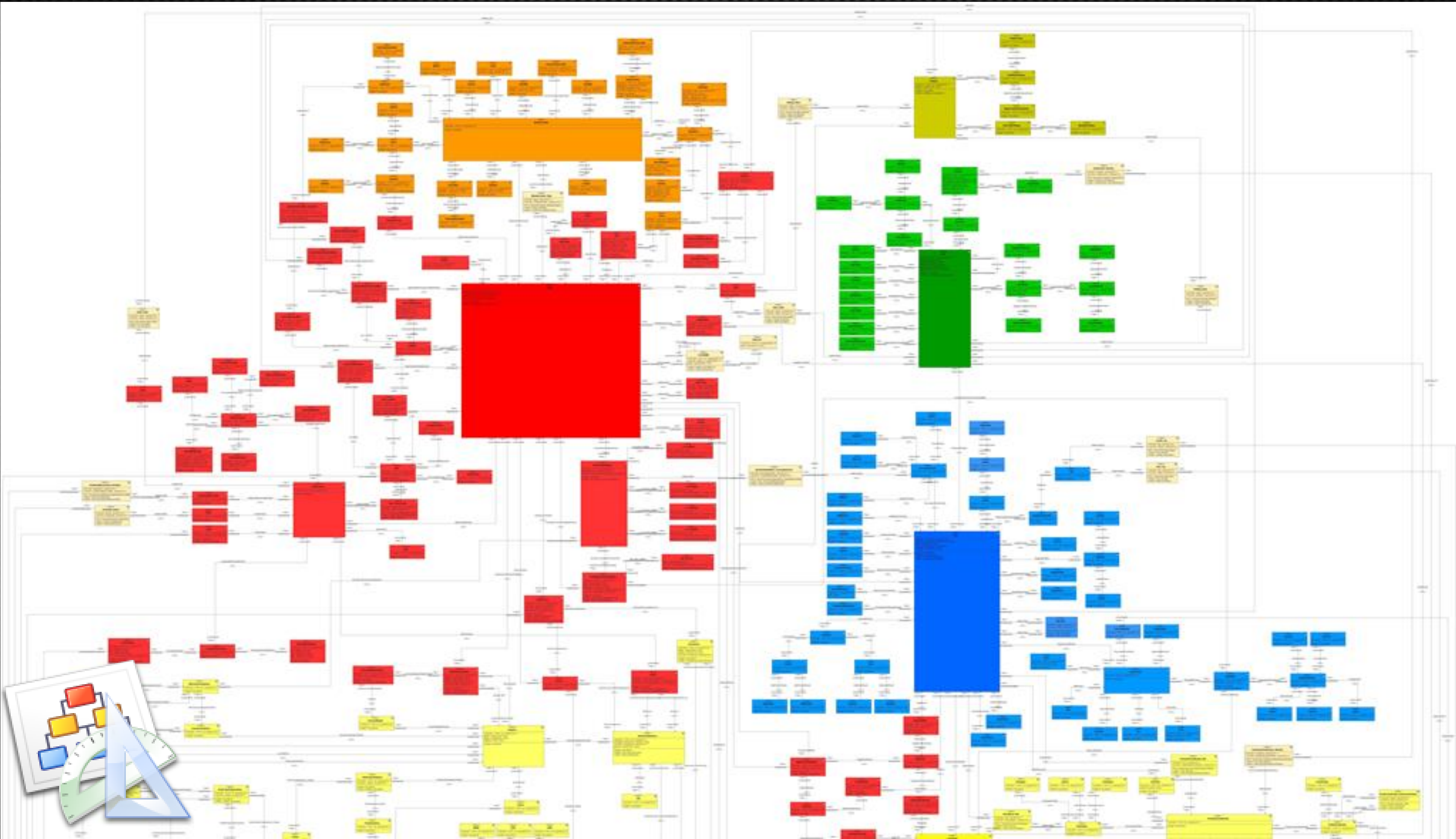
$$\frac{\text{<peak time>}}{\text{<users>}} = \text{<average rate of new users>} \qquad \text{OR} \qquad \frac{43200\text{ s}}{3000} = 14.4\text{ s}$$

480 s / 14.4 s = 33.333 concurrent users

waveRoute


E=MC²
BUT I STILL DON'T UNDERSTAND THE END-USER

Identifying the Dataset



Identifying the Dataset

Which Data to Use?

- Fixed Data
  - Informational data
  - Lookup tables
- Perspective Static Data
  - Identify tables required
  - Identify average number of rows and data

Identifying the Dataset

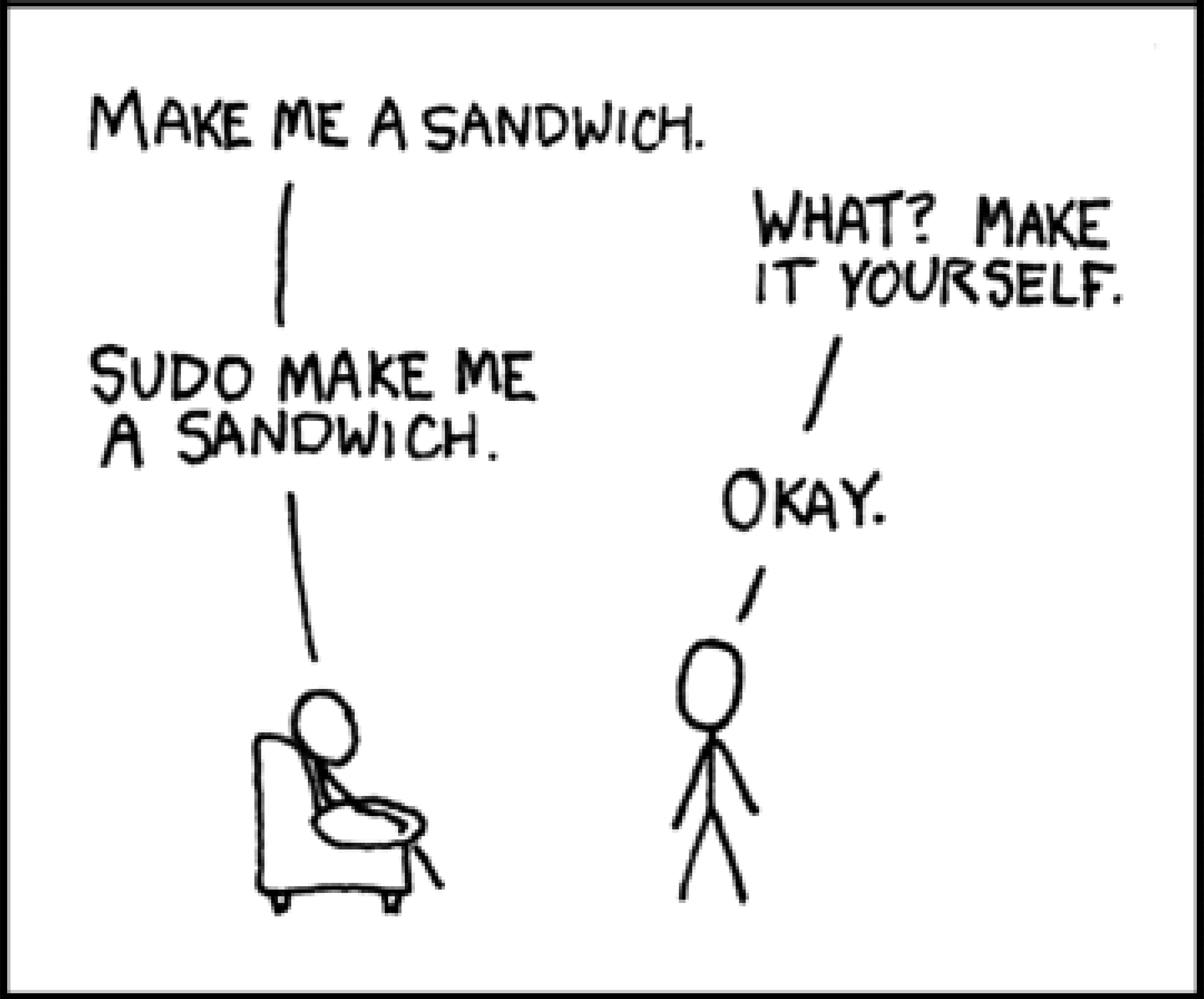
Which Data to Use? *continued*

- Historical Data
  - Partitioned Tables
  - Logs
  - Audit Tables

Identifying the Dataset

- The test Dataset can now be based on
  - Perspective
  - Marketing and Business Plan Forecasts
  - Which Data
  - Calculated estimate of
    - Perspective units
    - Average rows in tables per each Perspective unit
    - Average data column size per row per each Perspective unit

Develop Tests



Develop Database Tests

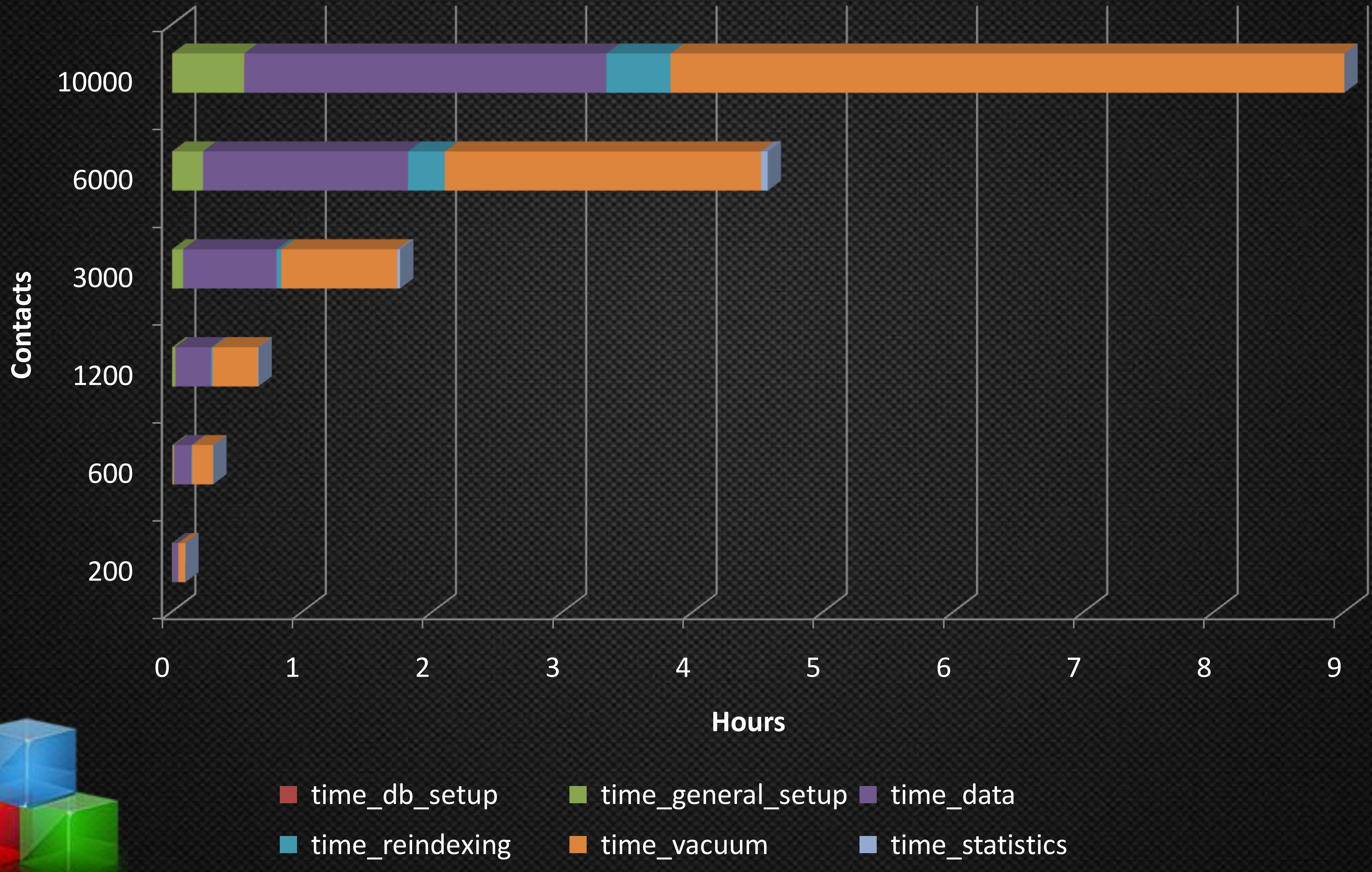
Functions

- Generation Functions
  - Perspective unit
  - Historical data
  - Index create/drop

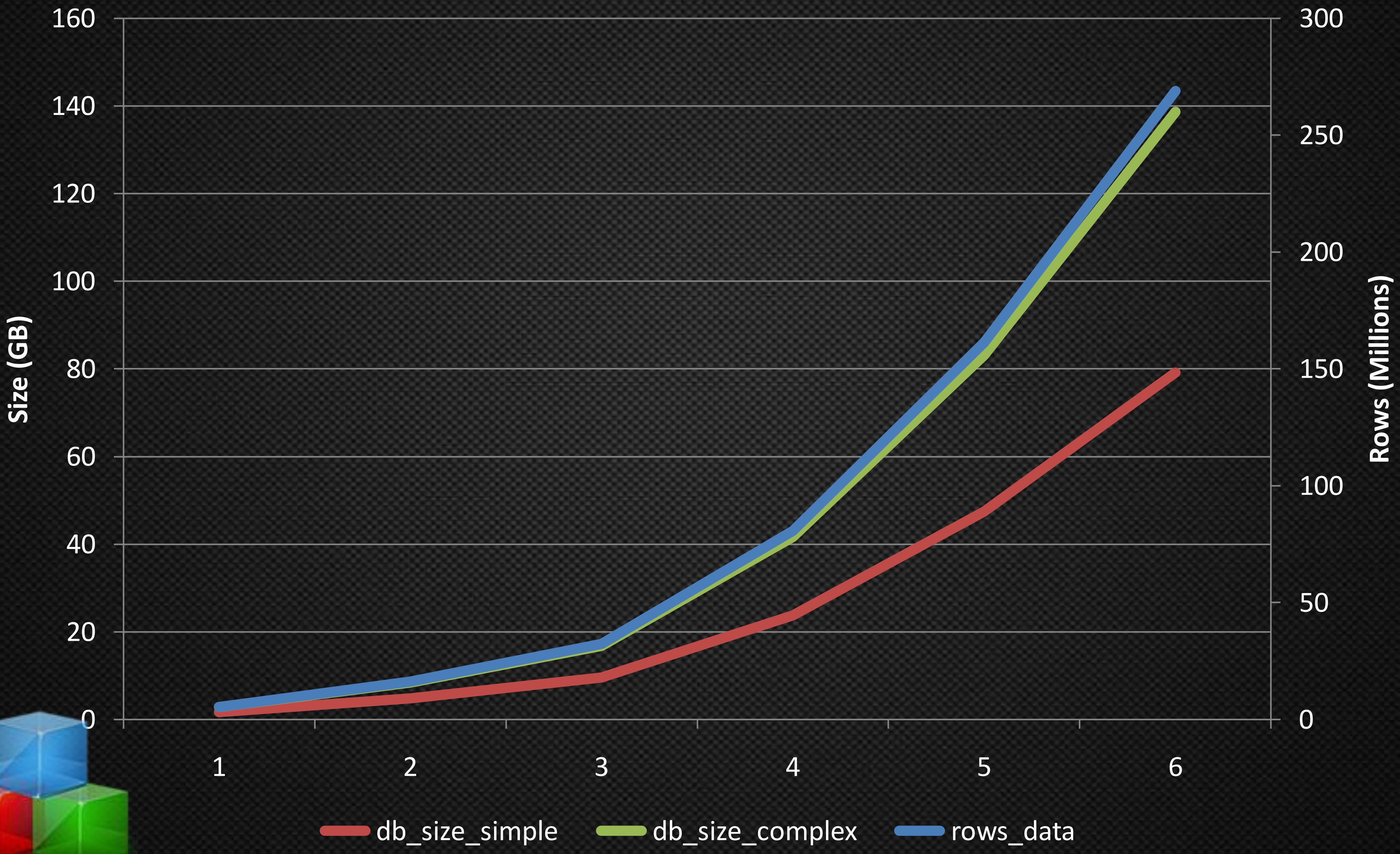
Schema

- Build Scripts
  - Make database template schema plus core data

It Takes Time



It Takes Resources



Develop Database Tests

Tips

- Make multi-process (cores)
- Prefix test functions with “test_”
- Run functions as superuser
  - use SET ROLE in function if necessary
- Make nested functions
  - Function of functions helps simplify tests scripts
- Prepare for partitioned tables

setup.py

```
import sys

if len(sys.argv) <= 2:
        inform = "False"
else:
        inform = sys.argv[2]

test = sys.argv[1]
cpu_cores = 4
poll_interval = 2.5
years = 1

if inform.lower() == "true":
        inform = True
else:
        inform = False
```

gen_history.py

```
#!/usr/bin/env python
import time
import setup
import pgdb                 #postgreSQL wrapper
import subprocess

def timestamp():
    """Returns formatted timestamp MM.DD.YYYY HH:MI:SS"""
    lt = time.localtime(time.time())
    return "%02d.%02d.%04d %02d:%02d:%02d" % (lt[2], lt[1], lt[0], lt[3],
   lt[4], lt[5])

start_overall = time.time()
```

gen_history.py

```
# drop/create db
ts_create = timestamp()
print "Creating wre_test_base wre_test_%s Database" % setup.test, ts_create
start_create = time.time()
subprocess.call('dropdb wre_test_%s' % setup.test, shell=True, stdout =
   subprocess.PIPE)
subprocess.call('createdb -E UTF8 -O wre_test -T wre_test wre_test_%s' %
   setup.test, shell=True, stdout = subprocess.PIPE)
# connect to database
my_db = pgdb.database("host='localhost' port='5432' dbname='wre_test_%s'
   user='postgres' password=‘sOmEaWeSoMeHaSh'" % setup.test)
end_create = time.time()
print "Finished Creating wre_test_base wre_test_%s Database ( %s s)" %
   (setup.test, int(end_create - start_create))
print ""
```

gen_history.py

```
# generate contacts, etc
ts_setup = timestamp()
print "Setting Up %s RTUs, Contacts, Devices, and Metatdata" % setup.test,
   ts_setup
start_setup = time.time()
subprocess.call('psql -c "SELECT * FROM test_gen_setup(%s);" wre_test_%s' %
   (setup.test, setup.test), shell=True, stdout = subprocess.PIPE)
end_setup = time.time()
print "Finished Setting Up %s RTUs, Contacts, Devices, and Metatdata  ( %s
   s)" % (setup.test, int(end_setup - start_setup))
print ""
```

gen_history.py

```
# generate historical data
ts_data = timestamp()
print "Begin Generating Historical Data", ts_data
start_data = time.time()
# remove old partitions
subprocess.call('psql -c "SELECT test_remove_partitions();" wre_test_%s' %
   setup.test, shell=True, stdout = subprocess.PIPE)
# create partitions
subprocess.call('psql -c "SELECT test_gen_history_ddl(%s);" wre_test_%s' %
   (setup.years, setup.test), shell=True, stdout = subprocess.PIPE)
# disable trigger for last# generate data
subprocess.call('psql -c "ALTER TABLE wre_test.my_partitioned_table DISABLE
   TRIGGER a_insert_my_partitioned_table_last_upsert_trigger;" wre_test_%s'
   % setup.test, shell=True, stdout = subprocess.PIPE)
```

gen_history.py

```
# generate data
processes = []
for x in range (setup.cpu_cores):
    print "    starting process %s" % (x+1)
    temp = subprocess.Popen('psql -c "SELECT * FROM
   test_gen_history_by_cpu(%s, %s, %s, %s);" wre_test_%s' %
   (setup.cpu_cores, x+1, setup.poll_interval, setup.years, setup.test),
   shell=True, stdout = subprocess.PIPE)
    processes.append(temp)
for process in processes:
    process.wait()
```

gen_history.py

```
# populate last
subprocess.call('psql -c "INSERT INTO wre_test.my_partitioned_table_last
   (fk_id,  val, updated) SELECT some_id, val, CURRENT_TIMESTAMP FROM t1;"
   wre_test_%s' % setup.test, shell=True, stdout = subprocess.PIPE)
# enable trigger for last
subprocess.call('psql -c "ALTER TABLE wre_test.my_partitioned_table
   ENABLE TRIGGER a_insert_my_partitioned_table_last_upsert_trigger;"
   wre_test_%s' % setup.test, shell=True, stdout = subprocess.PIPE)
end_data = time.time()
print "Finished Generating Historical Data (", int(end_data - start_data),
   "s)"
print ""
```

gen_history.py

```
# reindex
ts_reindex = timestamp()
print "Start Reindexing", ts_reindex
start_reindex = time.time()
my_db.execute_sql("SELECT * FROM test_gen_history_indexes(%s)",
   setup.years)
db_return = my_db.fetchone()
# create indexes
lines = []
# loop through subprocess.Popen()...
for line in db_return['test_gen_history_indexes']:
    temp = subprocess.Popen('psql -c "%s" wre_test_%s' % (line,
   setup.test), shell=True, stdout = subprocess.PIPE)
    lines.append(temp)
for index in lines:
    index.wait()
end_reindex = time.time()
print "Finished Reindexing (", int(end_reindex - start_reindex), "s)"
print ""
```

gen_history.py

```
# vacuum db
ts_vacuum = timestamp()
print "Start Vacuum Analyze", ts_vacuum
start_vacuum = time.time()
subprocess.call('vacuumdb -z wre_test_%s' % setup.test, shell=True, stdout
   = subprocess.PIPE)
end_vacuum = time.time()
print "Finished Vacuum Analyze (", int(end_vacuum - start_vacuum), "s)"
print ""
```

gen_history.py

```
# do statistics
ts_stat = timestamp()
print "Start Generating Statistics", ts_stat
start_stat = time.time()
my_db = pgdb.database("host='localhost' port='5432' dbname='wre_test_%s'
   user='postgres' password='sOmEaWeSoMeHaSh'" % setup.test)
my_db.execute_sql("SELECT simple, complex FROM dbsize")
db_return = my_db.fetchone()
my_db.execute_sql("SELECT count(*) AS total FROM
   wre_test.my_partitioned_table")
db_rows = my_db.fetchone()
end_stat = time.time()
print "Finished Generating Statistics (", int(end_create - start_create),
   "s)"
print ""
```

gen_history.py

```
# done
ts_finish = timestamp()
end_overall = time.time()
print "========== TEST COMPLETE =========="
print "Started:            ", ts_create
print "Finished:           ", ts_finish
print "Overall Time:       ", int(end_overall - start_overall), "s"
print "    Database Setup: ", int(end_create - start_create), "s"
print "    General Setup:  ", int(end_setup - start_setup), "s"
print "    Historical Data:", int(end_data - start_data), "s"
print "    Reindexing:     ", int(end_reindex - start_reindex), "s"
print "    Vacuum Analyze: ", int(end_vacuum - start_vacuum), "s"
print "    Statistics:     ", int(end_stat - start_stat), "s"
print "Database Size"
print "    Simple:         ", db_return['simple']
print "    Complex:        ", db_return['complex']
print "Data Rows:          ", db_rows['total']
```

gen_history.py

```
f = open('wre_test_%s.log' % setup.test, "w")
f2 = open('wre_tests.log', "a")
f.write("ts_start, ts_finish, time_overall, time_db_setup,
   time_general_setup, time_data, time_reindexing, time_vacuum,
   time_statistics, db_size_simple, db_size_complex, rows_data\n")
f.write("%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s\n" % (ts_create,
   ts_finish, int(end_overall - start_overall), int(end_create -
   start_create), int(end_setup - start_setup), int(end_data - start_data),
   int(end_reindex - start_re$
f2.write("%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s\n" %
   (setup.test, ts_create, ts_finish, int(end_overall - start_overall),
   int(end_create - start_create), int(end_setup - start_setup),
   int(end_data - start_data), int(end_r$
f.close()
f2.close()

if setup.inform:
        subprocess.call ("echo 'Finished Generating Database for %s
   Contacts'|/usr/sbin/sendmail 8885551212@vtext.com" % setup.test,
   shell=True)
```

fulltest.sh

```
cp wre_tests.log.clean wre_tests.log
python gen_history.py 200
dropdb wre_test_200
python gen_history.py 600
dropdb wre_test_600
python gen_history.py 1200
dropdb wre_test_1200
python gen_history.py 3000
dropdb wre_test_3000
python gen_history.py 6000
dropdb wre_test_6000
python gen_history.py 10000
dropdb wre_test_10000
echo 'Finished WRE historical data test'|/usr/sbin/sendmail
   8885551212@vtext.com
```

Perspective Units

Develop Backend Tests

If you have backend processes, you may need to create scripts that will provide the necessary input or events that will trigger the backend process.

- Virtual Users
- Virtual Perspective Units
- Virtual Entities

Develop Frontend Tests

- We will concentrate on web-based applications
- I   Funkload
  - Reports are excellent for web
  - Charts show thresholds
- Tsung is a close second
  - Reports are a bit vague
  - Highly Scalable

Run Tests

Run Tests

- Generate test database
- Generate historical data
- Turn PostgreSQL logging on
- Start backend tests and let them ramp up
- Start frontend test
- Complete tests
- Turn PostgreSQL logging off

Generate Test Report

- Our Reporting Tools
  - pgFouine
  - Funkload
  - Log files
  - Microsoft Office
- Our Monitoring Tools
  - OpenNMS
  - Yet Another Monitor (YAM)—in house

pgFouine

- Normal pgFouine reporting

```
php pgfouine.php -memorylimit 3840 -file pgsql -top 40 -
   report queries.html=overall,bytype,slowest,n-
   mosttime,n-mostfrequent,n-slowestaverage -report
   hourly.html=overall,hourly -report
   errors.html=overall,n-mostfrequenterrors -format html-
   with-graphs
```

- Since we are dealing with large files, we need to split out to chunk* files

```
split --lines=1000000 pgsql.log chunk
```

pgFouine

- Example of how to do pgFouine CSV report

```
cat chunk/chunk* |~/pgfouine/pgfouine-
  1.2/pgfouine.php -memorylimit 3750 - -report
  chunk*_queries.csv=csv-query -format text
```

- Example of how to combine all CSV into one full CSV

```
cat chunk*_queries.csv >> queries.csv
```

pgFouine

- Make a new database and table to put the CSV data into

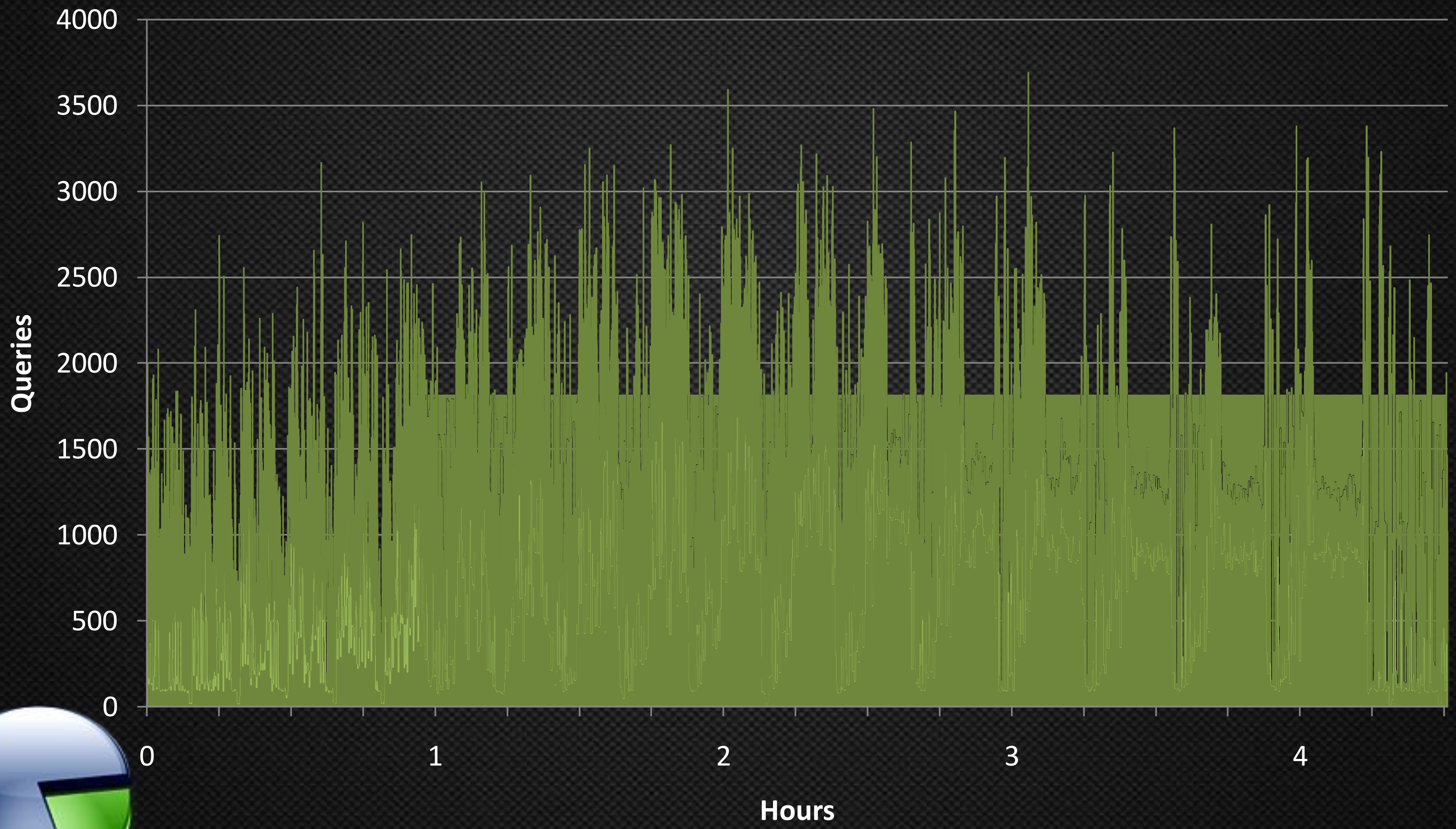
```
CREATE DATABASE mytest encoding 'UTF8';
CREATE TABLE log (
id integer,
date timestamp,
connection_id integer,
database text,
"user" text,
duration float,
query text);
```

pgFouine

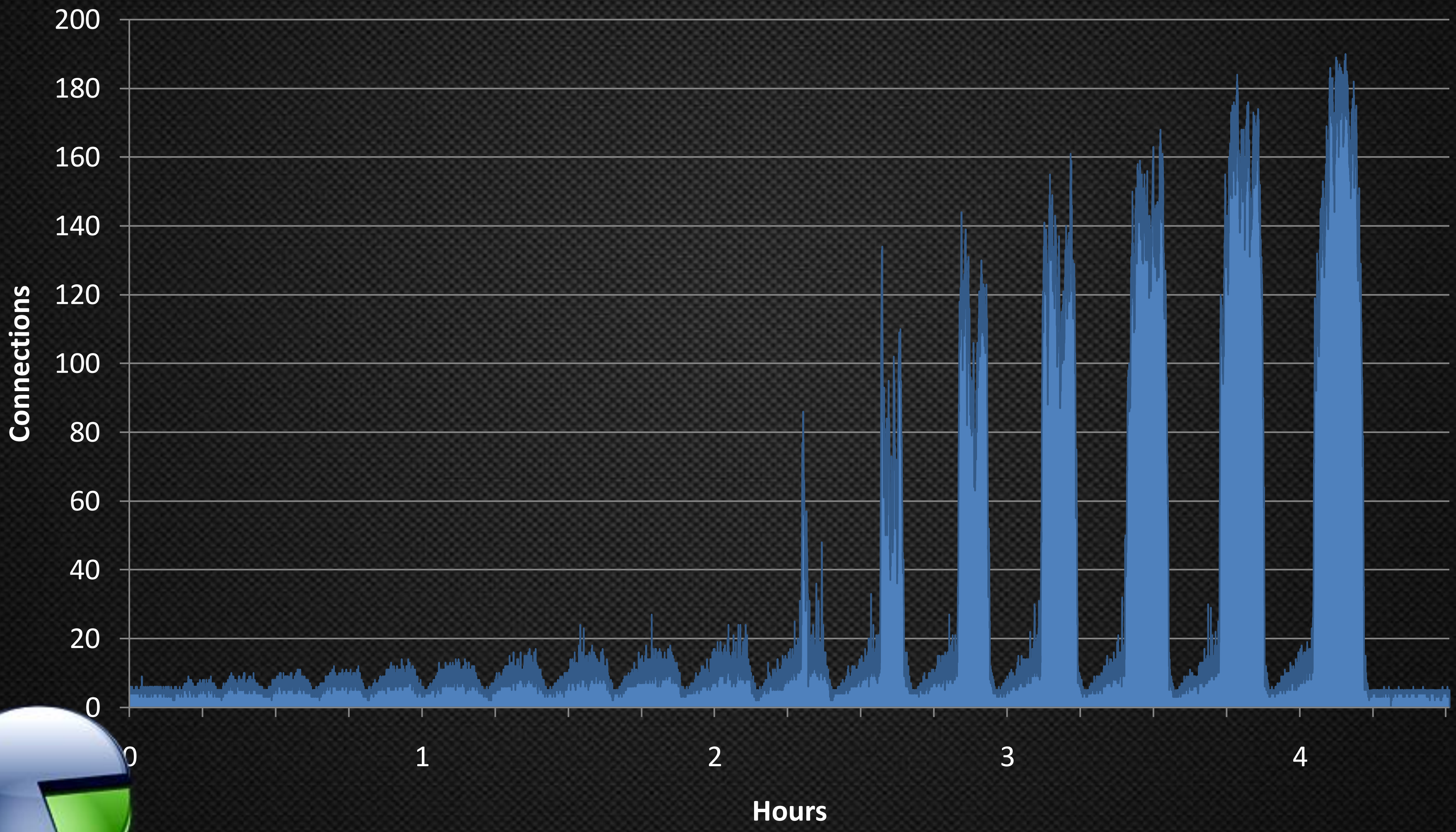
- Log into the database and run

```
COPY log FROM 'queries.csv' WITH CSV;
ALTER TABLE log ADD COLUMN type CHAR(1) DEFAULT
   'S';
UPDATE log SET type='I' WHERE query ~*
   '^insert.*$';
UPDATE log SET type='D' WHERE query ~*
   '^delete.*$';
UPDATE log SET type='U' WHERE query ~*
   '^update.*$';
UPDATE log SET type='O' WHERE query NOT ILIKE
   'select%' AND query NOT ILIKE 'insert%' AND query
   NOT ILIKE 'update%' AND query NOT ILIKE
   'delete%';
CREATE INDEX test_date_idx ON log(date);
CREATE INDEX test_database_idx ON log(database);
CREATE INDEX test_connection_idx ON
   log(connection_id);
CREATE INDEX test_type_idx ON log(type);
```

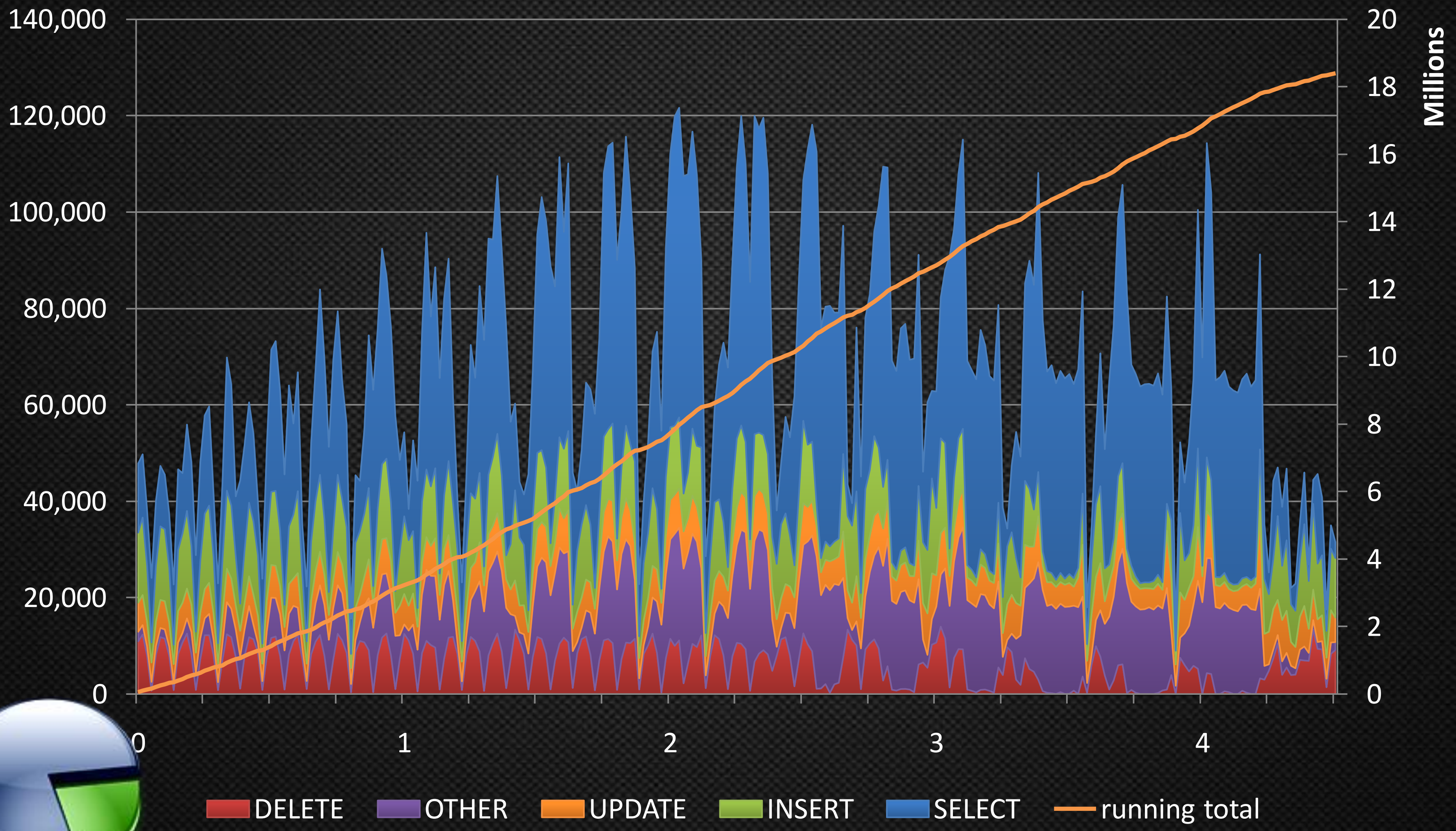
Database TPS



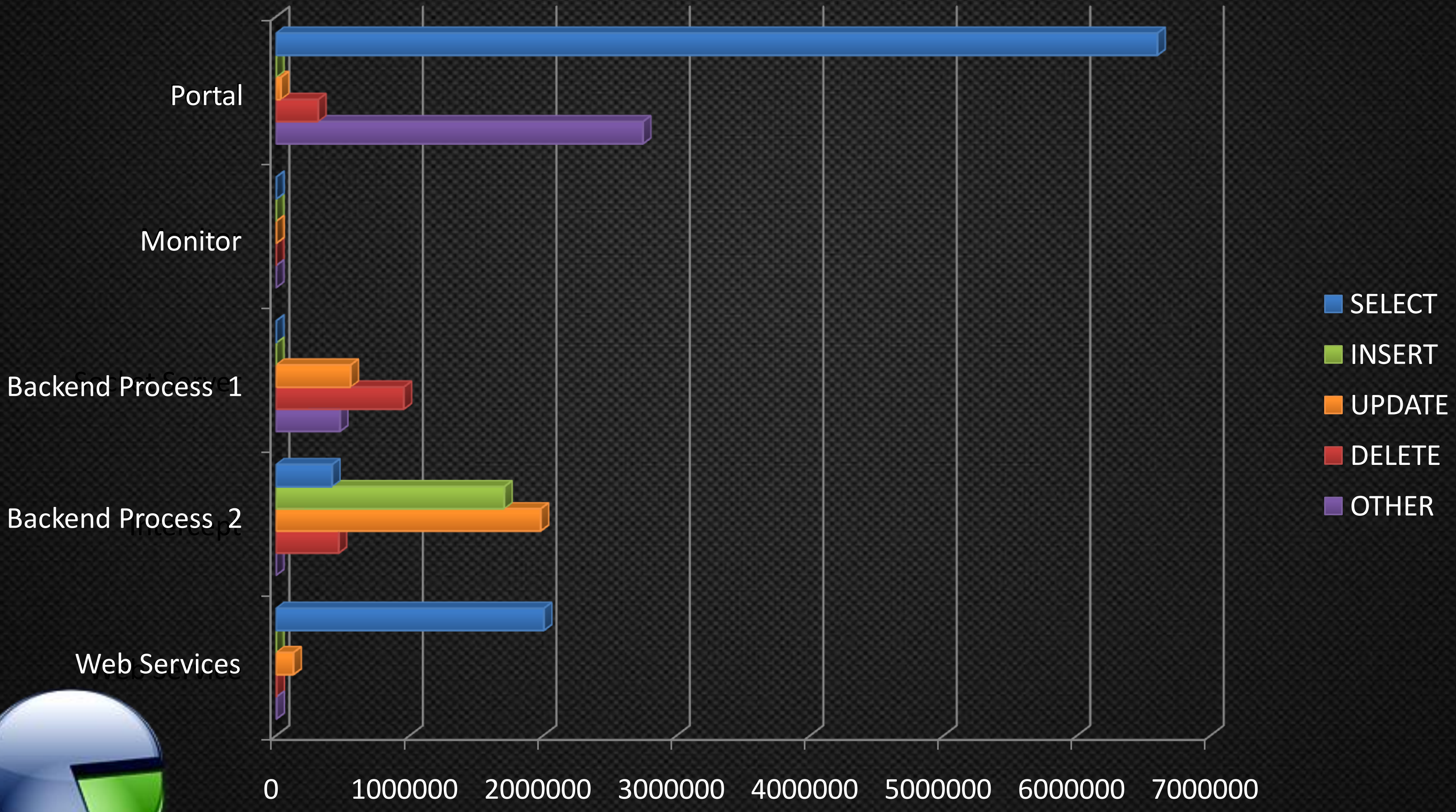
Database CPS



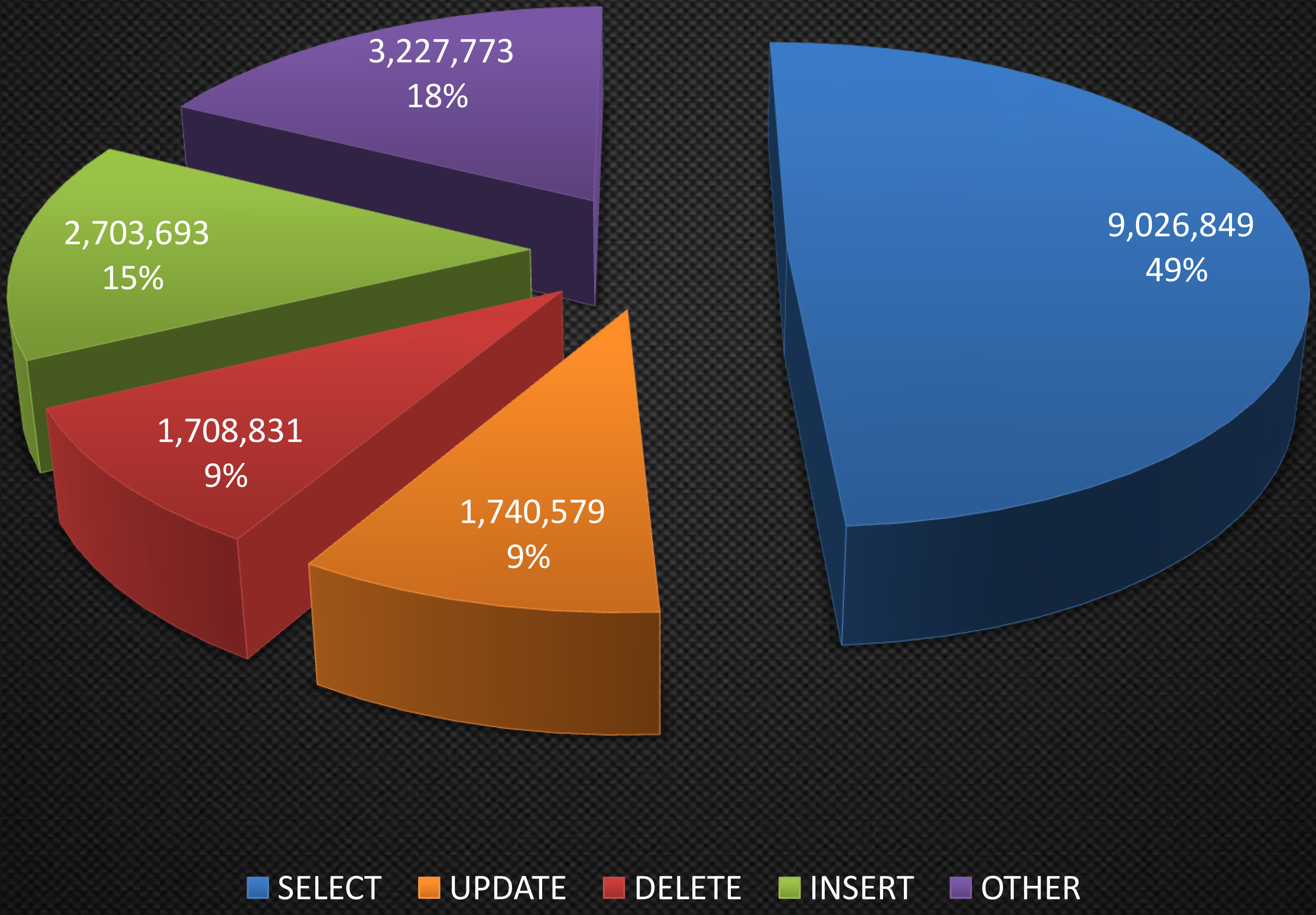
Database TPM by Type



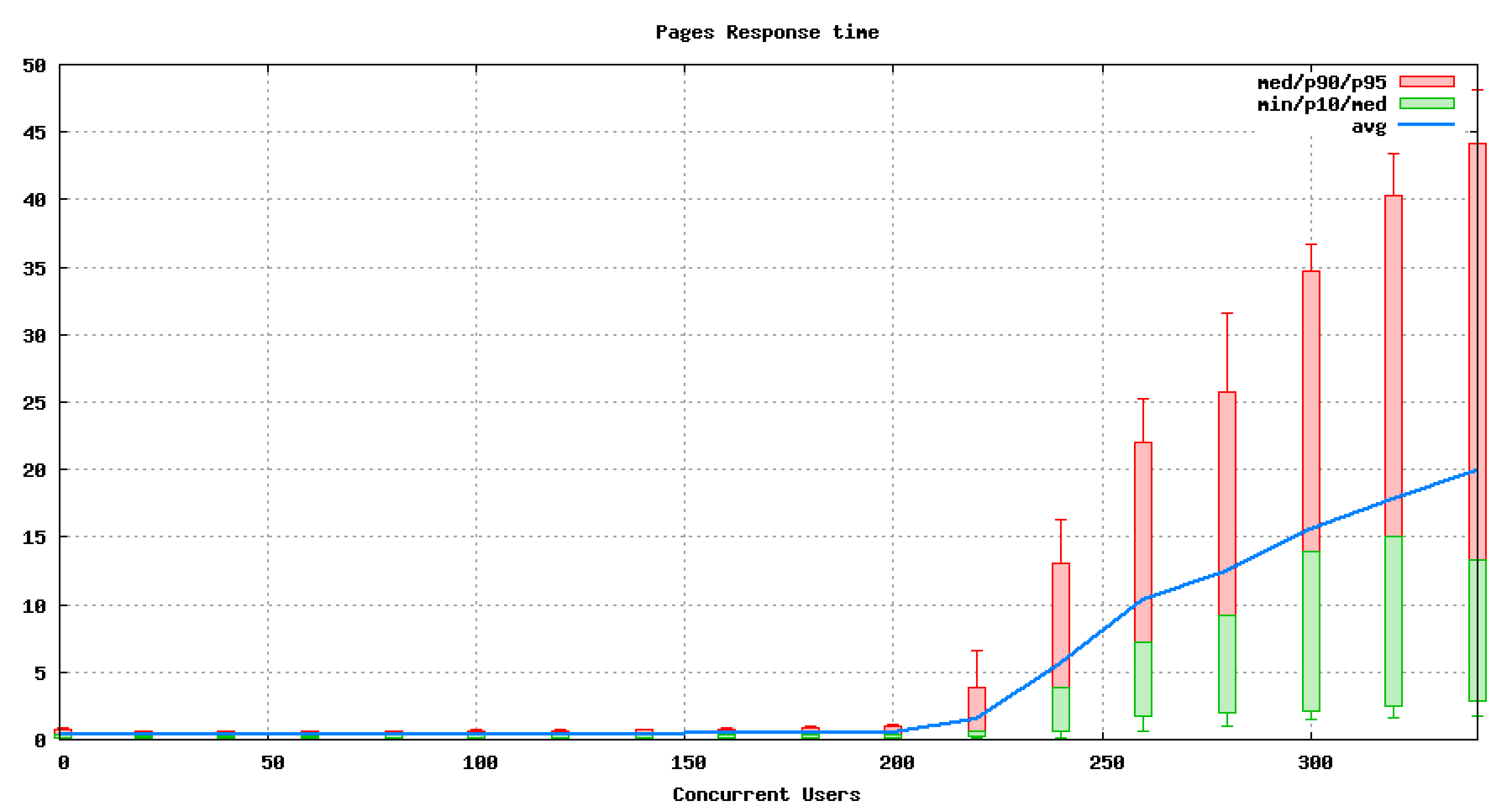
Database Queries by Function



Database Queries by Type



Funkload Report



Identifying Realistic Loads

Customers	200	600	1200	3000	6000	10000
RTUs	400	1200	2400	6000	12000	20000
Devices	1300	3900	7800	19500	39000	65000

Database	PASS	PASS	PASS	PASS	PASS	FAIL
Backend	PASS	PASS	PASS	PASS	FAIL	FAIL
Frontend	PASS	PASS	PASS	PASS	PASS	FAIL

waveRoute

Things Learned Along the Way

DO

- Make everything scriptable and repeatable
- Time everything
- Keep notes of what you did and the results
- Spend the time to get quantifiable numbers
- Log everything possible
- Make a baseline dataset to get fastest query results and fastest user experience for comparrison

Things Learned Along the Way

DO

- Run these tests on the *actual* production system or equivalent
- Make optimizations *after* you complete tests
- Request hardware if it is the bottleneck
- Re-run tests after optimizations or core changes to prove goals
  - We ran the Full Integration Test (F.I.T.) ~5 times

Things Learned Along the Way

DO

- Combine with monitoring
  - OpenNMS
  - Nagios
  - Zabbix
  - Reconnoiter / Circonus (talk to xzilla)
  - Home-brewed

Things Learned Along the Way

DO NOT

- Skimp out on the logging
  - Sanity checks to explain anomalies
- Think it will be a simple process and something quickly achievable in a day
- Ignore the Suits and Marketroids
- Ignore the SLA

Hard Work Pays Off

- 80% speed increase on website
- 85% bandwidth reduction
- Discovered bottlenecks
- Discovered thresholds
- Fully redundant and scalable system
- Better understanding of what our database is actually doing

Changes Made as a Result

- Apache + APC + mod_php → Lighttpd + eAccelerator + FastCGI + PHP5
- Load balanced servers as needed due to discovered thresholds
- Backend application no longer caching
- Web Servers split according to task
- Virtualized servers reconfigured for resources needed instead of guessing

Changes Made as a Result

- Purchased more hardware
- Database Disk Schedule Elevator set for best performance
  - {NOOP | CFQ | Deadline}

What We Covered

- Shortfalls of FLOSS benchmark tests
- Identifying Test Components
- Identifying Realistic Loads
- Identifying the Dataset
- Developing Tests and Procedures
- PostgreSQL Functions for Tests
- Python Scripts for Tests
- Helpful Tools

For More Information

Funkload – http://funkload.nuxeo.org/

Tsung – http://tsung.erlang-projects.org/

OpenNMS– http://www.opennms.org/

Blog– http://digicondev.blogspot.com/

Email– conradz@gmail.com