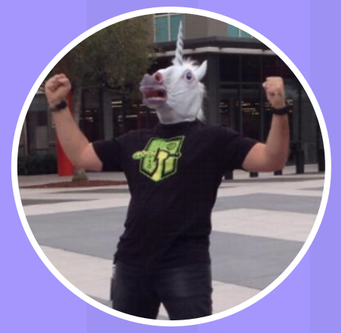
operating system analytics and host intrusion detection at scale

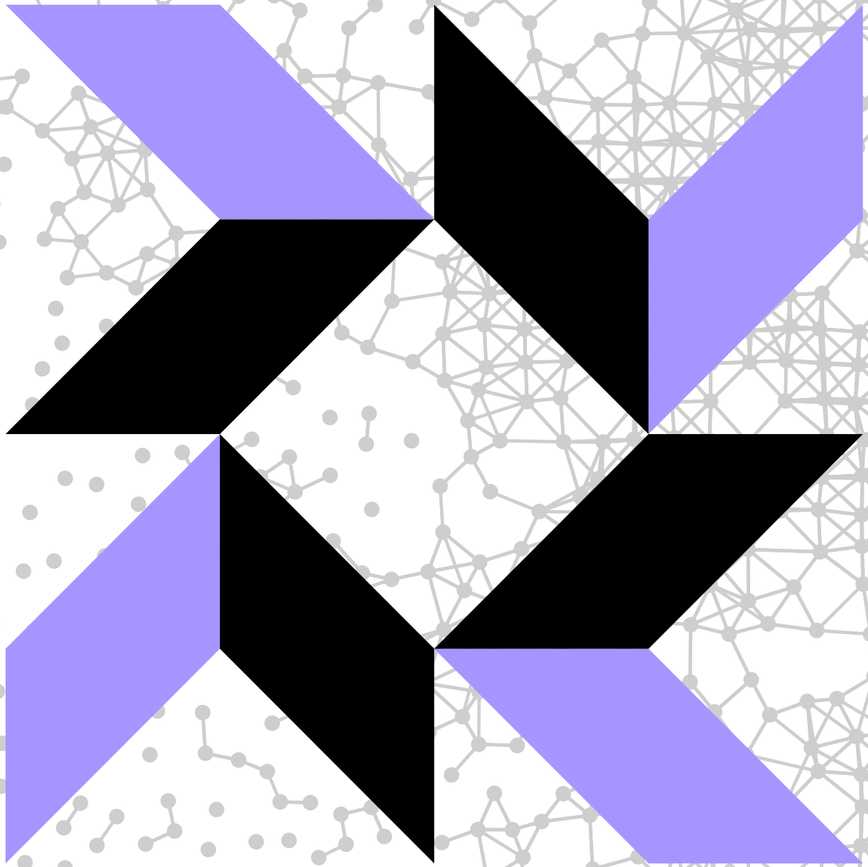
mike arpaia / facebook

mimeframe / facebook

ted reed / facebook



javier marcos / facebook



what's the problem?

we're all trying to catch attackers

it's a hard problem

- insider threats
  - espionage
- external threats
  - APT
  - hacktivists
  - mass malware
  - the list is endless

we're all deploying tools

we need help

- many of us have too many vendor products
- often times, a product solves too narrow of a use-case

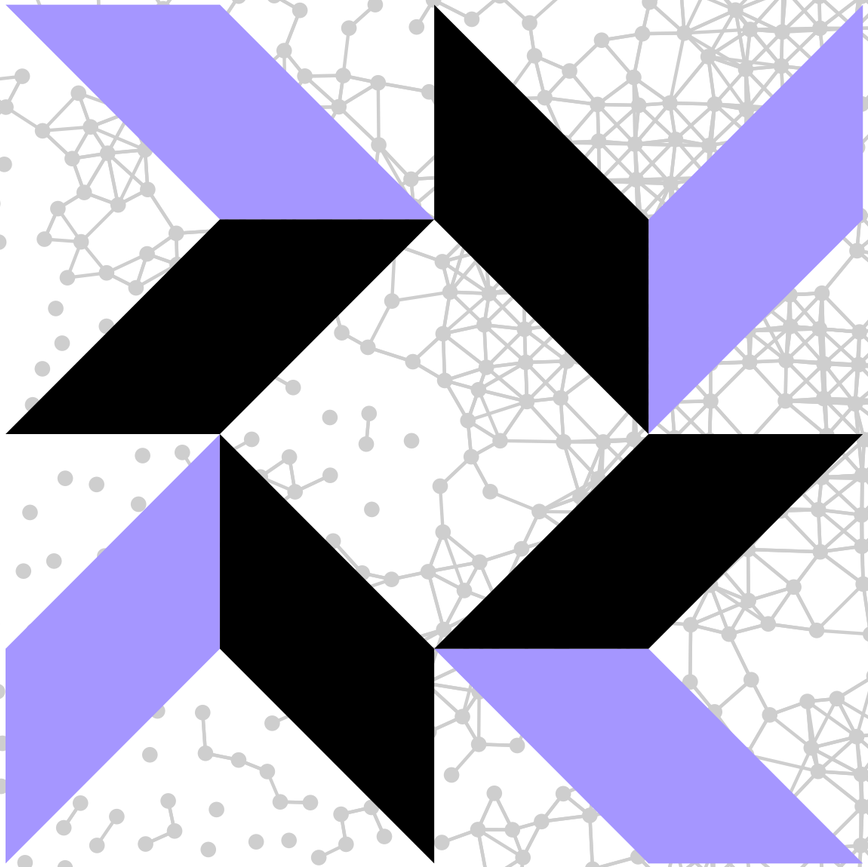
- new use-case? new vendor
  - mo' money and mo' problems

we live in a windows centric world

but, times are changing

- more OS X laptops
- most production infrastructure runs on Linux

- few are successfully instrumenting their OS X and Linux hosts
  - how would we solve that problem?



desired properties

simple

performant and reliable

easy to integrate

flexible

simple

- no complex coding for users
- low-level details should be abstracted
- easy to use, deploy and maintain

performant and reliable

- host degradation is unacceptable
  - sane resource utilization over time
- company services should not be impacted
- extensive logging and metrics

easy to integrate

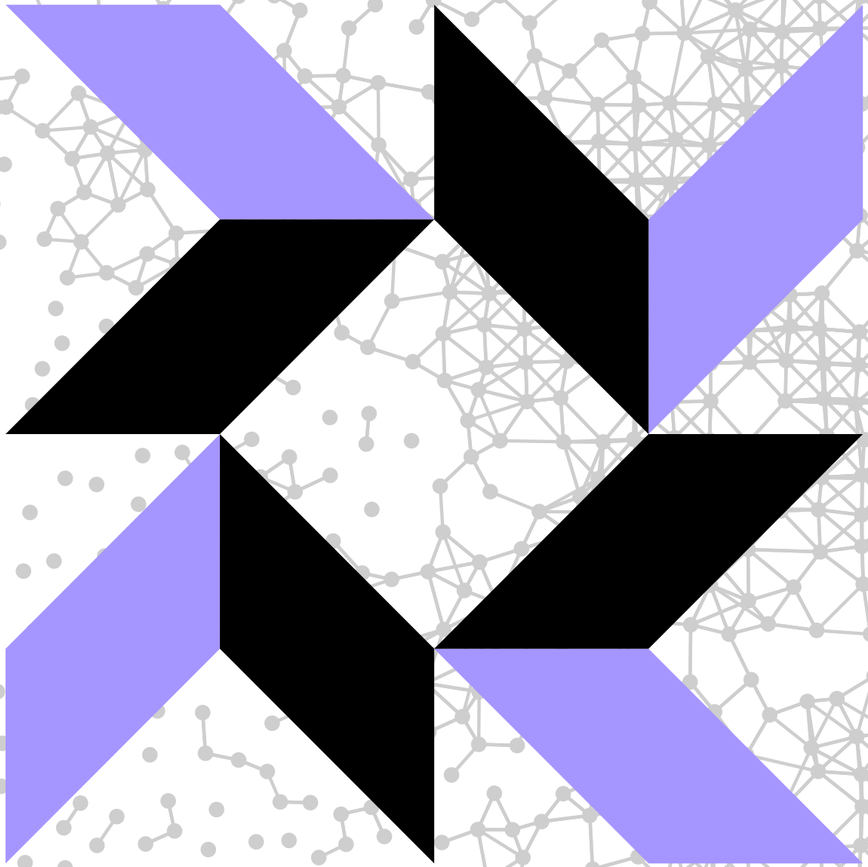
- every company has existing infrastructure
  - distributed configurations
  - real-time logging
  - data warehousing

- you should have the option to use existing infrastructure to help power your host instrumentation

flexible

- host instrumentation can help solve many problem domains
  - intrusion detection
  - vulnerability management
  - reliability
  - compliance
  - < insert domain here >

- having a single solution reduces cognitive overhead and time spent



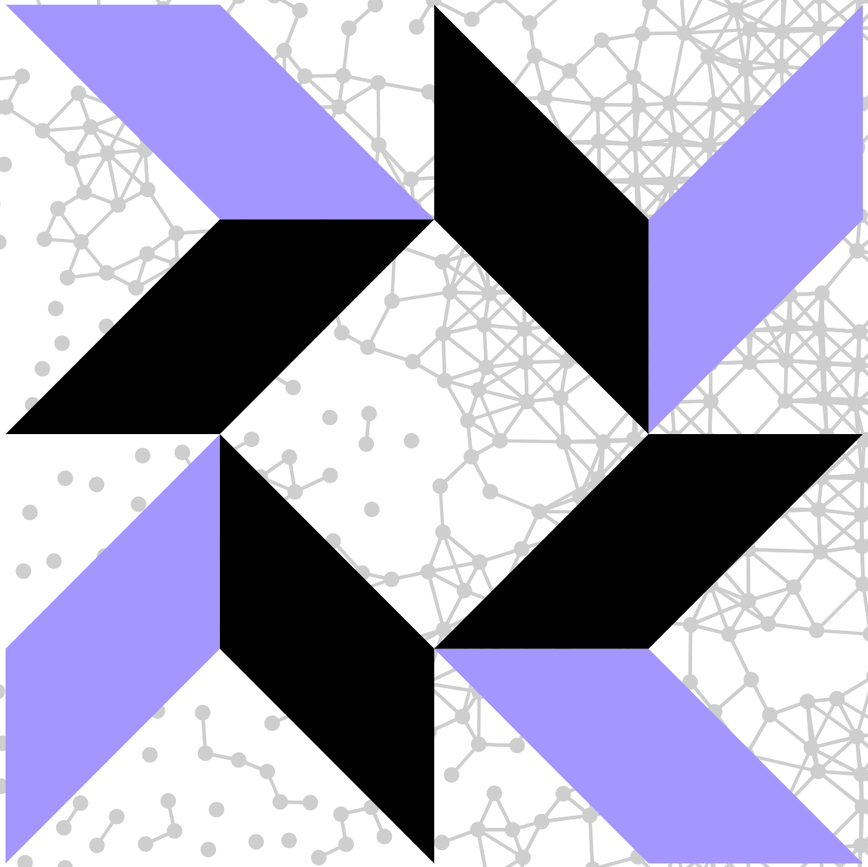
osquery

osquery

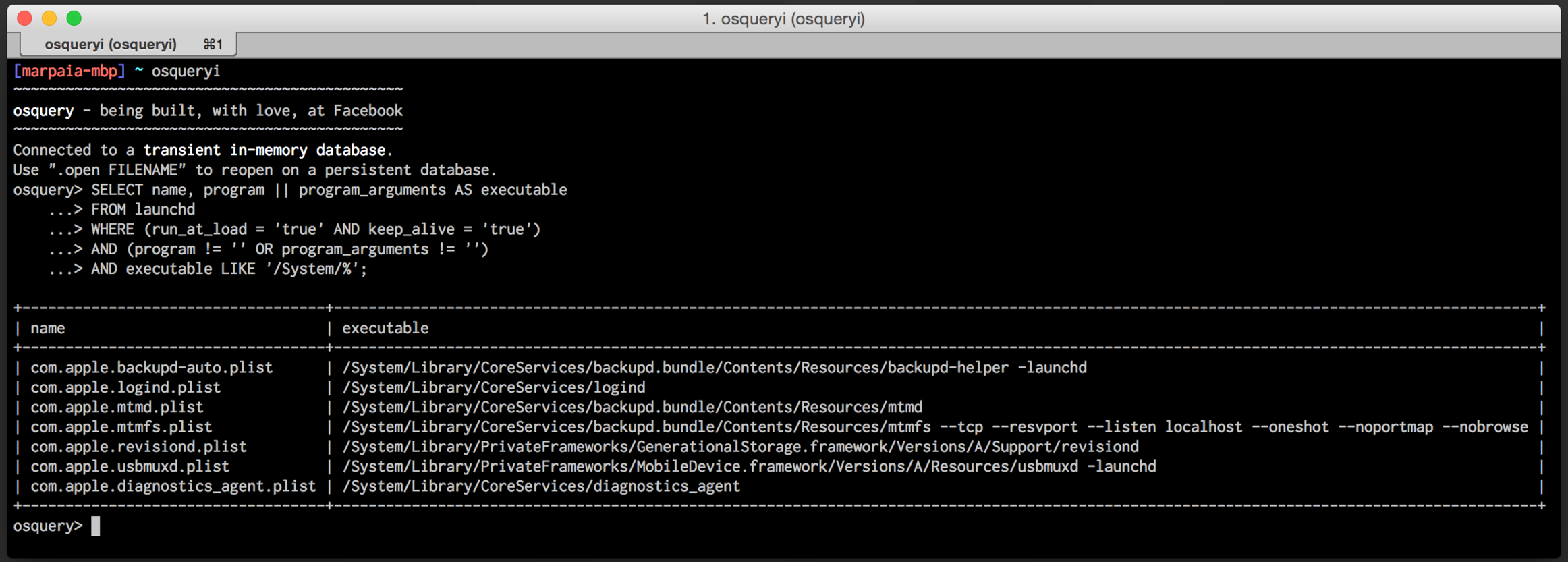
SQL for your infrastructure

use SQL queries to explore OS state

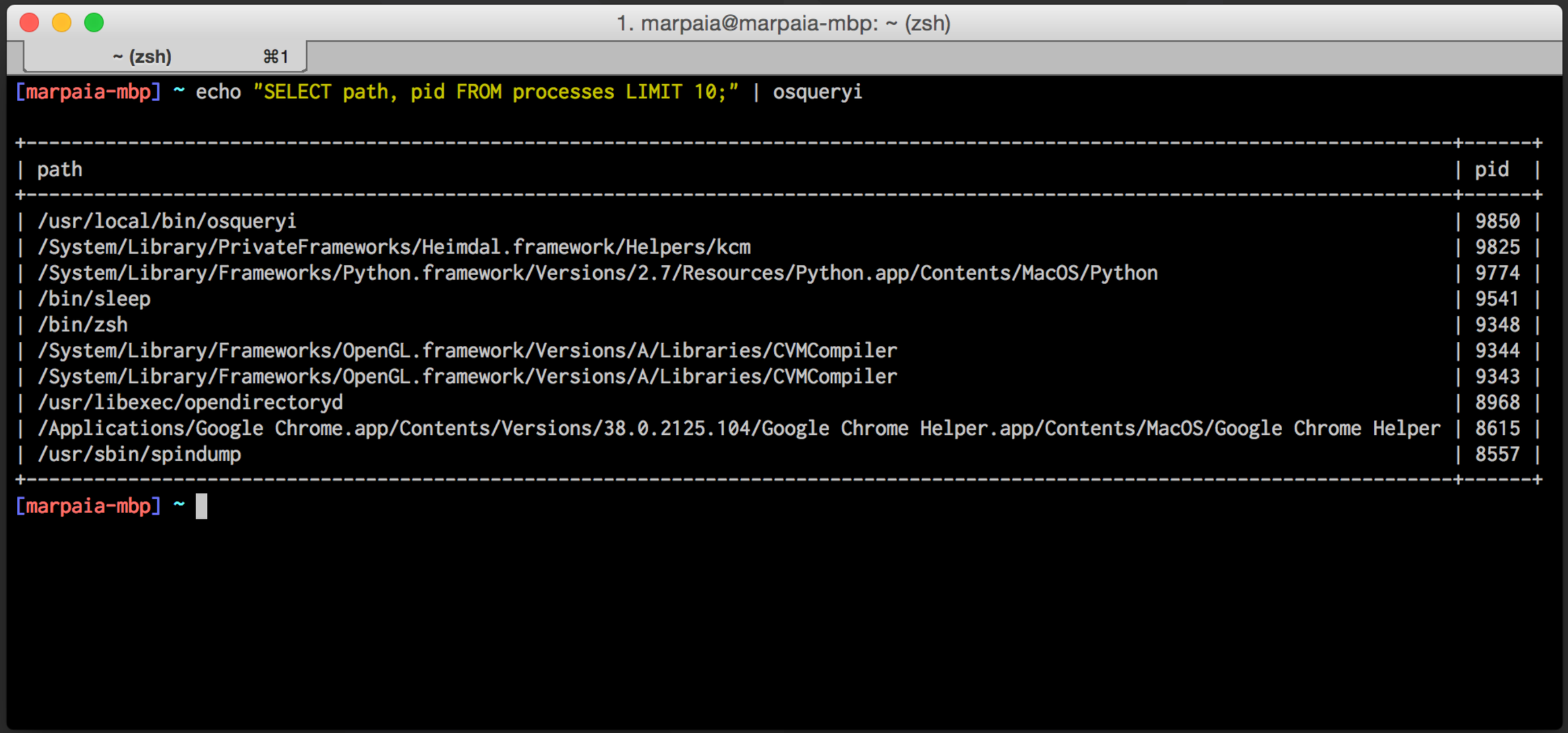
- running processes
- loaded kernel modules
- active network connections
- route table
- firewall settings
- installed software
- and more



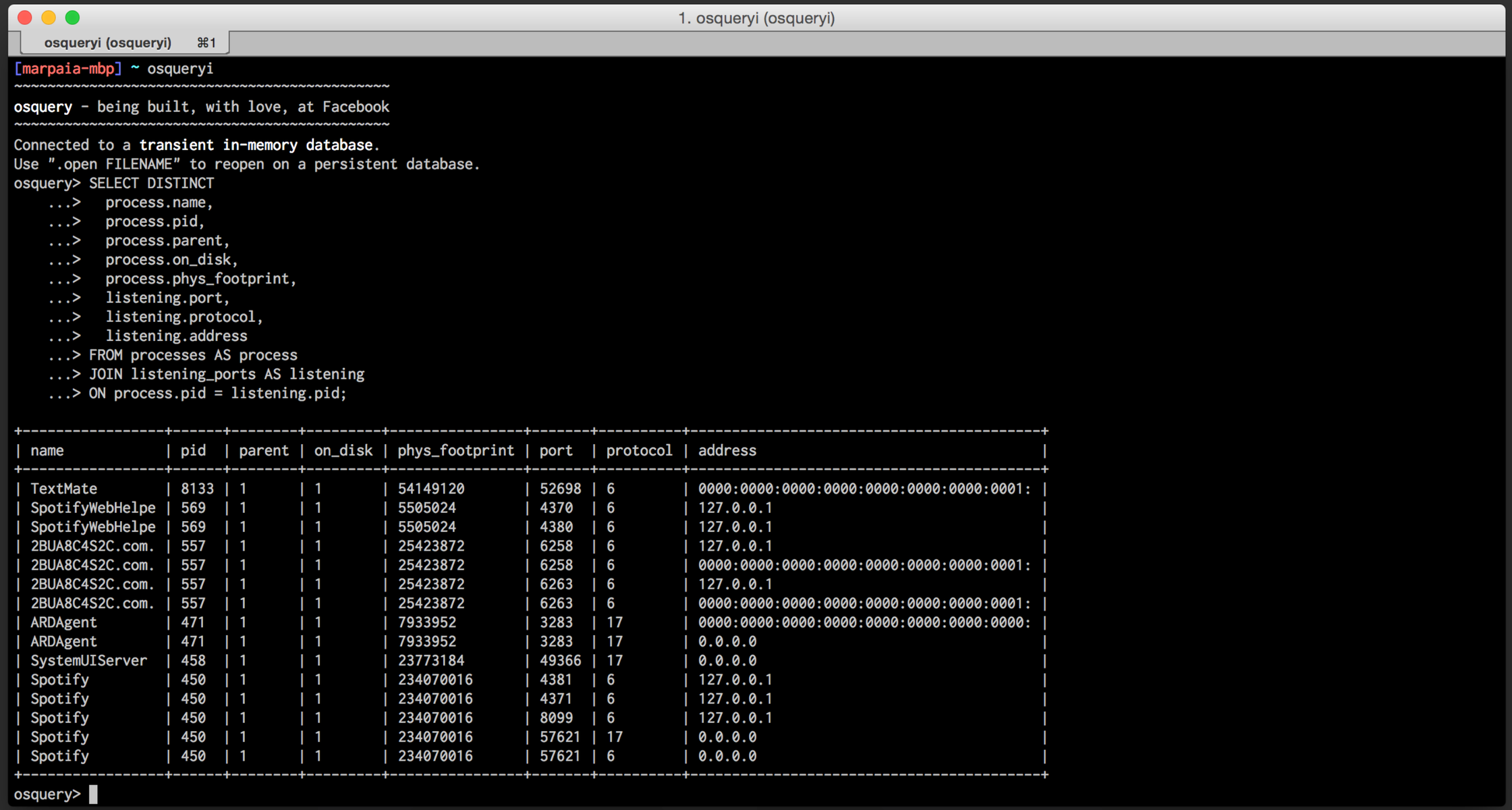
osqueryi



LaunchDaemons which run a binary at boot



running processes



processes listening on ports

many tables are available

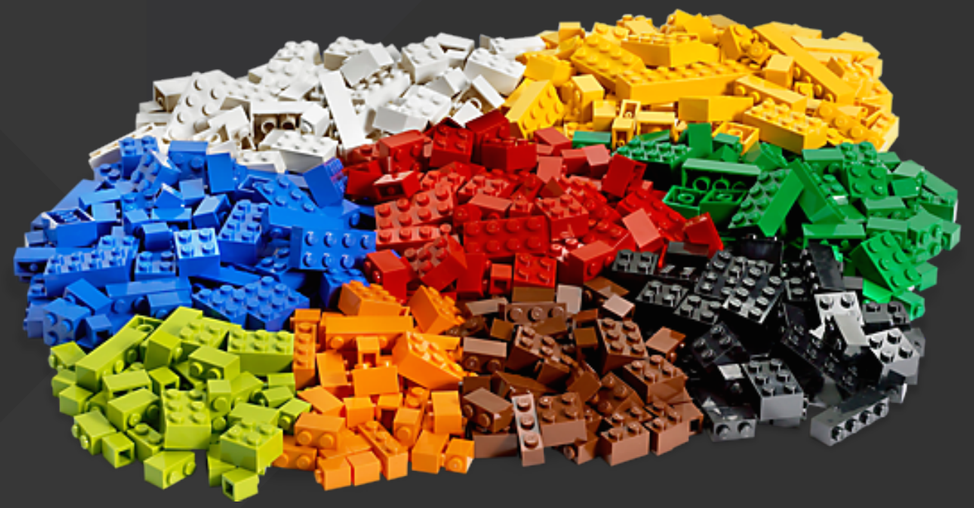
more tables are being written every day

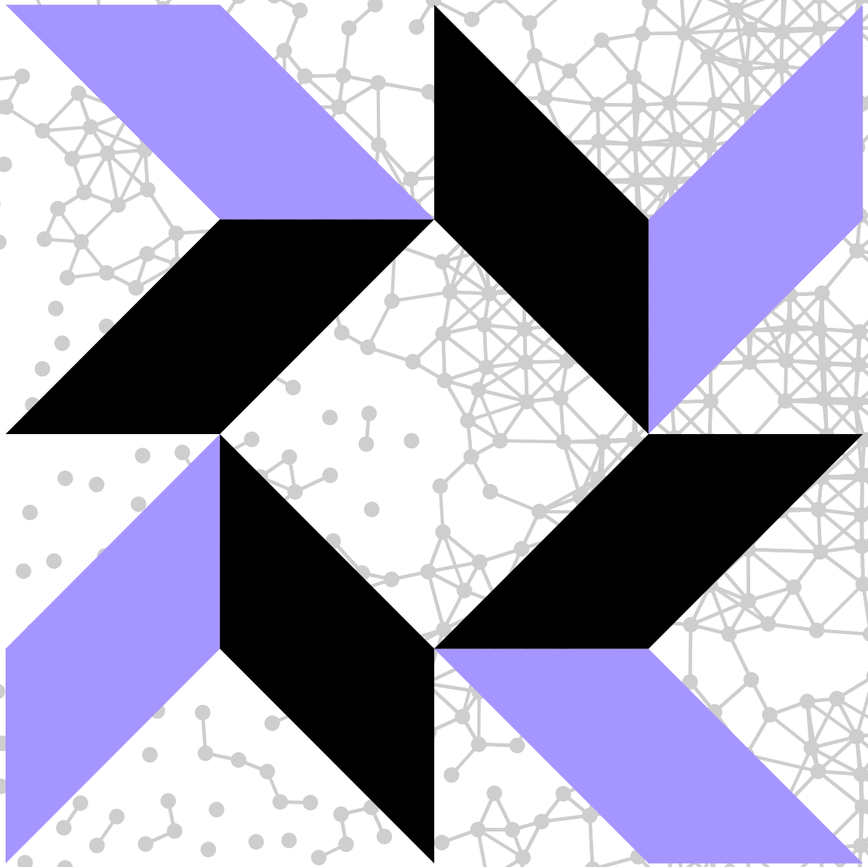
- alf
- alf_exceptions
- alf_explicit_auths
- alf_services
- apps
- ca_certs
- cpuid
- etc_hosts
- groups
- homebrew_packages
- interface_addresses
- interface_details
- kextstat
- last
- launchd
- listening_ports
- nvram
- osx_version
- passwd_changes
- processes
- routes
- suid_bin
- time
- users

use simple tables, together

osquery enables complex analysis by allowing users to join and aggregate across several simple tables

- simple tables have many advantages
  - easier to write
  - easier to maintain
  - can be used in many contexts





osqueryd

osqueryd

daemon for low-level host monitoring

know how the results of a query change over time

- schedule a query on your hosts via a config

- the daemon takes care of periodically executing your queries
  - buffers results to disk and generates a log of state changes
  - logs results for aggregation and analytics

RocksDB

host event pub/sub stream

event-based operating system introspection

subscribe to key OS events to create dynamically growing tables

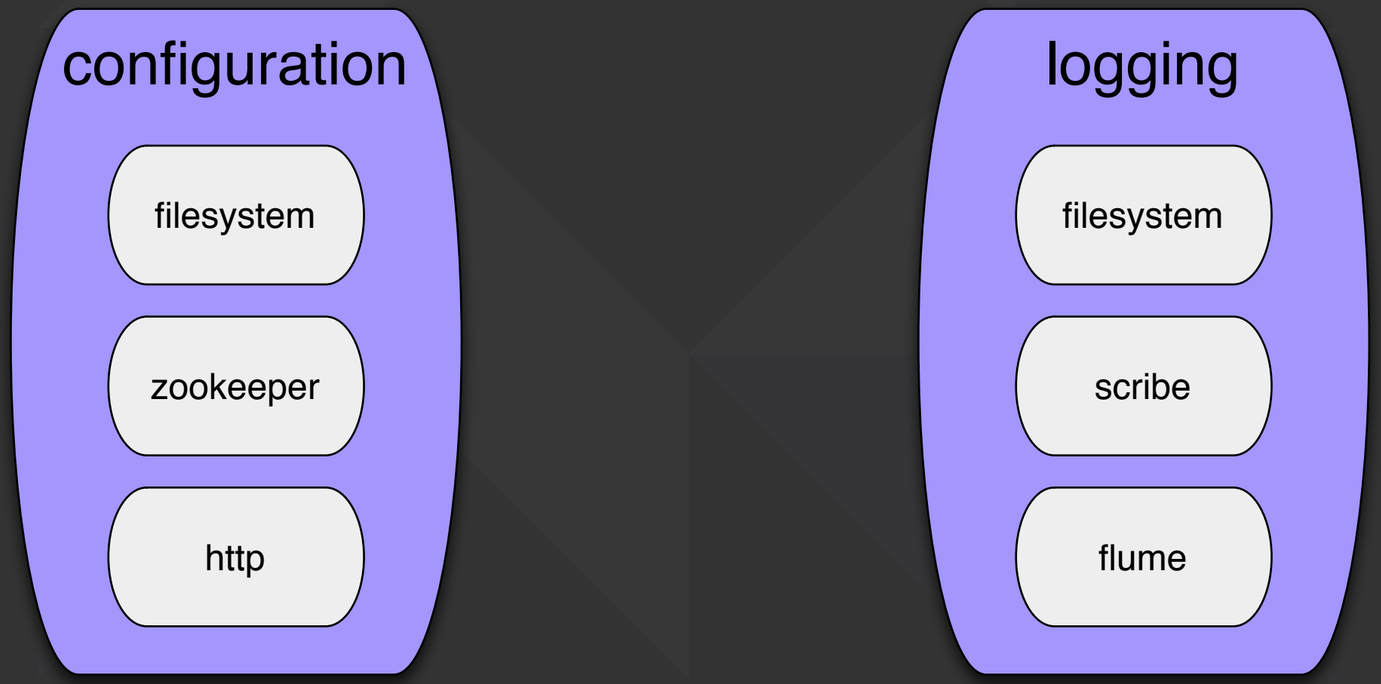
- subscribe to “publishers”
  - filesystem changes (inotify, FSEvents)
  - network setting changes (SCNetwork)

- query the history of your host, as it evolves

plugin system

for config distribution and data infrastructure

- simple plugin API
- specify your plugins at runtime with a command-line flag



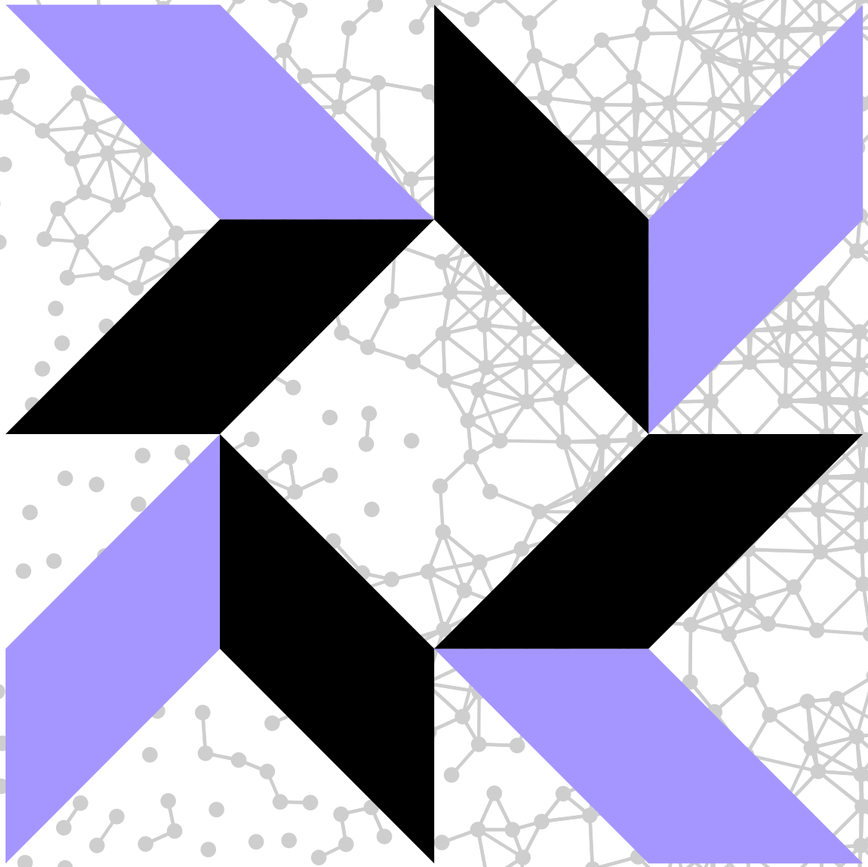
```
#include "osquery/logger/plugin.h"
#include <glog/logging.h>

namespace osquery {

class GlogPlugin : public LoggerPlugin {
 public:
  Status logString(const std::string& message) {
    LOG(INFO) << message;
    return Status(0, "OK");
  }
};

REGISTER_LOGGER_PLUGIN("glog", std::make_shared<osquery::GlogPlugin>());
}
```

registering a glog plugin



tables

creating tables is easy

easily define what your tables “look like” in Python and use C++ to implement what a full-table scan would return

- the Python is used to generate faster C++ code transparently
- you write a single C++ function which implements a full-table scan


2. marpaia@marpaia-mbp: ~ (zsh)
~ (zsh) ⌘1
[marpaia-mbp] ~ echo "SELECT * FROM time;" | osqueryi
+------+---------+---------+
| hour | minutes | seconds |
+------+---------+---------+
| 0 | 58 | 30 |
+------+---------+---------+
[marpaia-mbp] ~

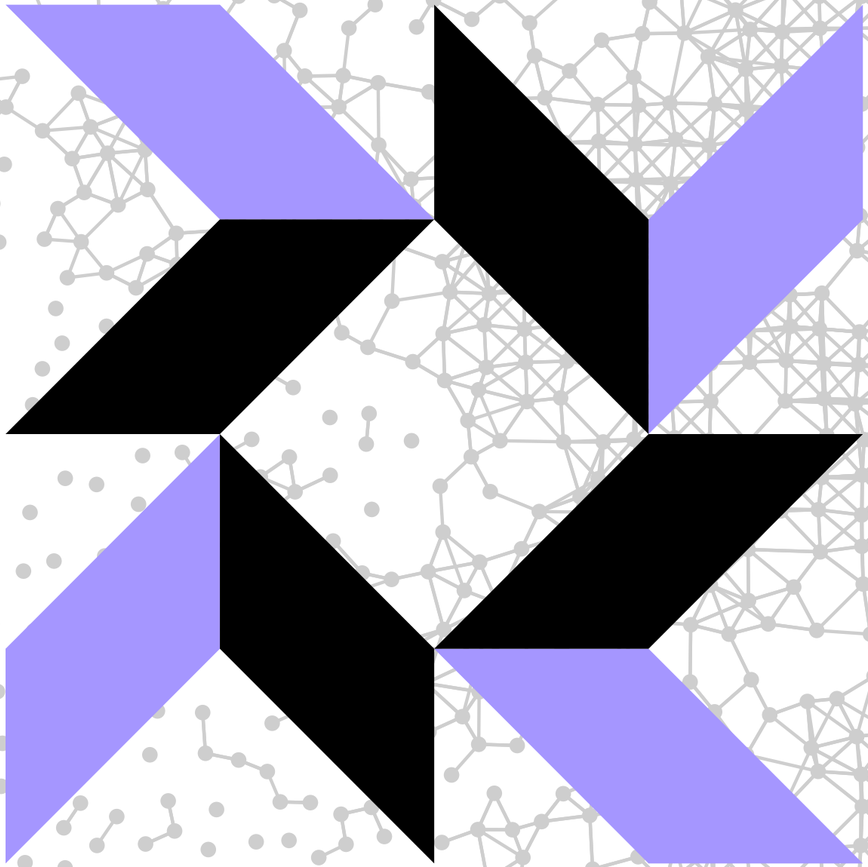
```
table_name("time")
schema([
    Column("hour", INTEGER),
    Column("minutes", INTEGER),
    Column("seconds",INTEGER),
])
implementation("time@genTime")
```

```
namespace osquery {
namespace tables {

QueryData genTime() {
  QueryData results;
  struct tm* now = localtime(time(0));

  Row r;
  r["hour"] = INTEGER(now->tm_hour);
  r["minutes"] = INTEGER(now->tm_min);
  r["seconds"] = INTEGER(now->tm_sec);
  results.push_back(r);

  return results;
}
}
}
```



open source

work on osquery with us

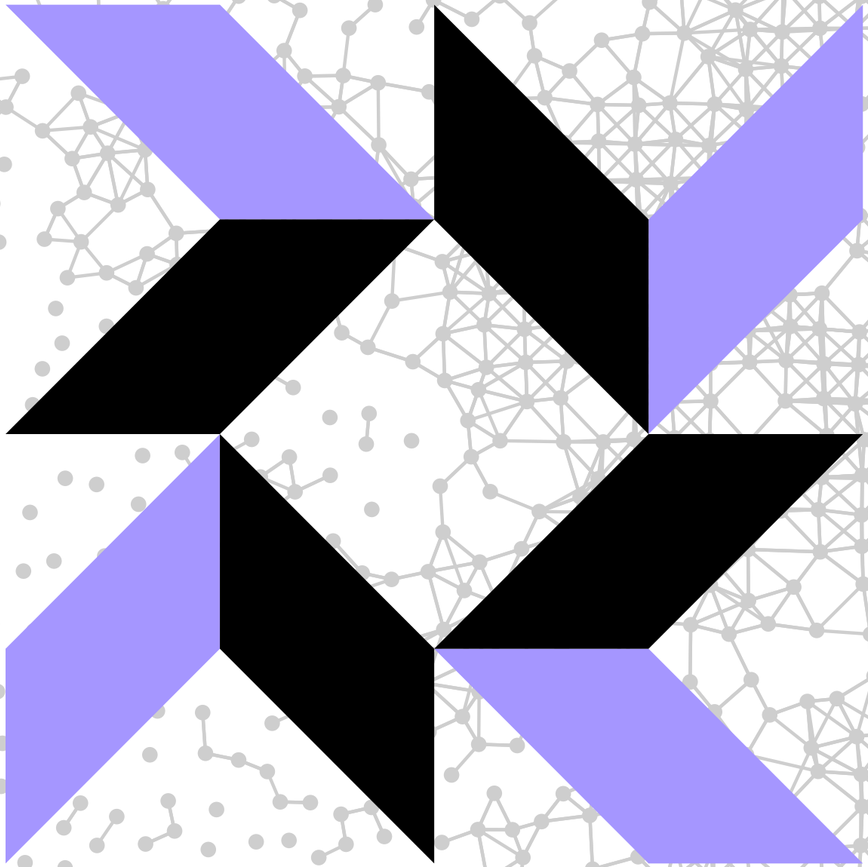
all development happens in the open, on GitHub

the problem that osquery solves isn't unique to facebook

- https://github.com/facebook/osquery
- http://osquery.io

this journey is 1% finished: get involved

- we're excited to take on future challenges in the open
- let's build together



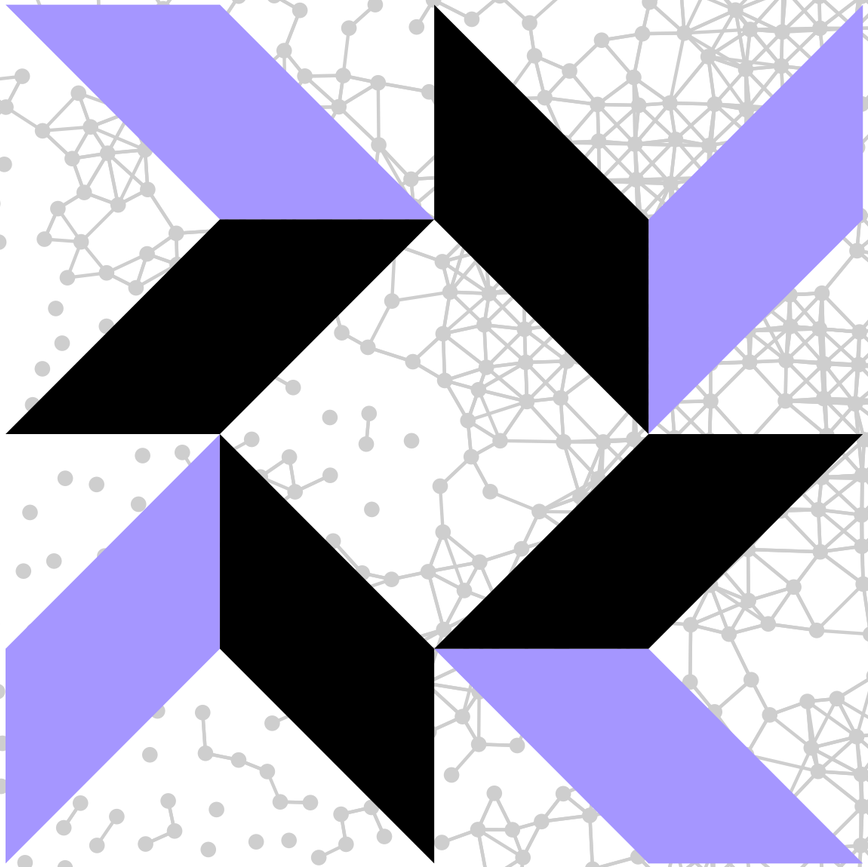
what's next?

what we're working on

contribute and help us build an awesome tool

there's a team of great engineers at facebook that are actively working on making osquery awesome for everyone

- more tables
- ad-hoc remote queries
- kernel modules for lower-level behavior monitoring
- deep systems visibility



questions

http://osquery.io