3 Mini Talks of 10 minutes each

OS/application inversion
Anil Madhavapeddy

Living Databases
Pavlo Baron

Why do we still not give a @#%! about testing?!
Trisha Gee

QCon

The OS/App Inversion: escaping POSIX to bring Git to the datacentre

QCon New York Mini talk

Anil Madhavapeddy (speaker)
with Thomas Gazagnaire and Benjamin Farinier
University of Cambridge Computer Laboratory

June 7, 2015

Common features every distributed system needs

- **Persistence** for fault tolerance and scaling
- **Scheduling** of communication between nodes
- **Tracing** across nodes for debugging and profiling

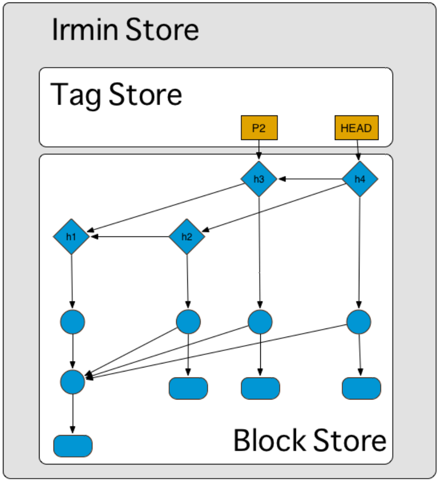
Most distributed systems run over an operating system, and so are stuck with the OS kernel exerting control. We use *unikernels*, which are application VMs that have complete control over their resources.

What if we just used Git?

- **Persistence**
  - `git clone` of a shared repository across nodes
  - `git commit` of local operations in the node
- **Scheduling**
  - `git pull` to receive events from other nodes
  - `git push` to publish events to other nodes
- **Tracing and Debugging**
  - `git log` to see global operations
  - `git checkout` to roll back time to a snapshot
  - `git bisect` to locate problem messages
- **New Problems**
  - `git rebase` needed for distributed garbage collection
  - Shelling out to `git` is slow and lacks control

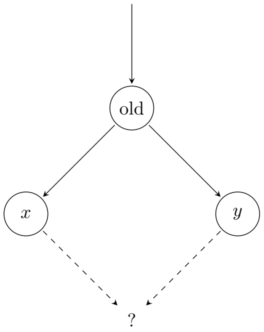
Irmin, large-scale, immutable, branch-consistent storage

- Irmin is a library to **persist** and **synchronise distributed data structures** both on-disk and in-memory
- It enables a style of programming very similar to the **Git workflow**, where distributed **nodes fork, fetch, merge and push** data between each other
- The general idea is that you want every active node to get a **local** (partial) **copy of a global database** and always be very explicit about how and when data is shared and migrated


Irmin Store
Tag Store
P2
HEAD
h3
h4
h1
h2
Block Store

```
type t = ...
(** User-defined contents. *)
type result = [ `Ok of t |
   `Conflict of string ]

val merge: old:t → t → t →
   result
(** 3-way merge functions. *)
```


$$\begin{tikzcd} & \arrow[d] & \\ & \text{old} \arrow[dl] \arrow[dr] & \\ x \arrow[dr, dashed] & & y \arrow[dl, dashed] \\ & ? & \end{tikzcd}$$

Irmin Features

- Still pre 1.0, but several useful datastructures such as distributed queues and efficient ropes.
- HTTP REST for remote clients, or library via OCaml.
- JavaScript compilation for pure browser operation.
- Bidirectional operation, so `git` commits map to Irmin commits from any direction.
- Open source at `https://irmin.io`
- **Want to know more?** I'm giving a full talk on this on Friday at 1015 titled "Functional Distributed Programming with Irmin"!

Teaser: Xen Toolstack using Irmin

`https://www.youtube.com/watch?v=DSzvFwIVm5s`

Why Do We Still Not We Give A @#$! About Testing?

Trisha Gee @trisha_gee

Developer Advocate at JetBrains and OCD Geek

2010 IT Project Success Rates

Project Style	Successful	Challenged	Failure
Ad-hoc	49%	37%	14%
Iterative	61%	28%	11%
Agile	60%	28%	12%
Traditional	47%	36%	17%

Dr Dobbs, 2010

…when it comes to quality, 40% prefer to deliver on time and on budget and 57% prefer to deliver high-quality, easy-to-maintain systems.

Software Project Success Rates

Successful	32%
Challenged	44%
Failure	24%

Gartner, 2012

So, Why Don’t We Test?

Focus is on delivering features

Boring

Too much effort for too little value

"I don't know what it's supposed to do"

Code wasn't designed for testing

Too Hard

Just a box to tick

Framework gets in the way

"It's only a prototype"

The code is self describing

Takes too much time

I'm the only maintainer

Manual Process

Why Should We Care?

Documentation

Freedom to Refactor

Knowledge Sharing

Usability

Help us with Design

So we know what doesn't work

Identify missing features

So we know it works!

What Can We Do?

Tap into Motivation

- OCD
- Frame it as a difficult problem
- Gamification
- Make it as easy as possible
- Lead by example
- Exploration (what if…?)
- Set as primary task
- Add to Objectives
- Break emotional attachment to code

We Don’t Have To Care About Testing

We Need To Encourage Behaviour That Prioritises Quality

“living” database

@pavlobaron

speaking of
streaming, reactive,
realtime, near time,
near realtime etc.

here is how all things are supposed to look in the reactive context

here is how current database access methods look in the reactive context

here is how current databases look in the reactive context

9322537031465

it’s time to rethink databases

“living” database

some of the ideas are
partially implemented
in CEP, NoSQL, classic
DBMS, Rx etc.,
but not as a whole

storage is an aspect

storage is raw

everything is
ordered by time

everything is on channels

queries are continuous

views are materialised and continuous

results are published

results are limited,
timed-out or ignored
by consumer

database quantifies harvest

changes are published both ways

everybody fires and forgets,
but gets reminded
when things go wrong

consumers react

it needs far more than
just support in
the database driver

it needs the database
to be an active part
of the overall data flow

“living” database

Questions