

# Fully Automated Testing of the Linux Kernel

**Martin J. Bligh**

[mbligh@google.com](mailto:mbligh@google.com)

**Andy Whitcroft**

[apw@shadowen.org](mailto:apw@shadowen.org)

<http://test.kernel.org>

<http://test.kernel.org/autotest>

# Why fully automated testing?

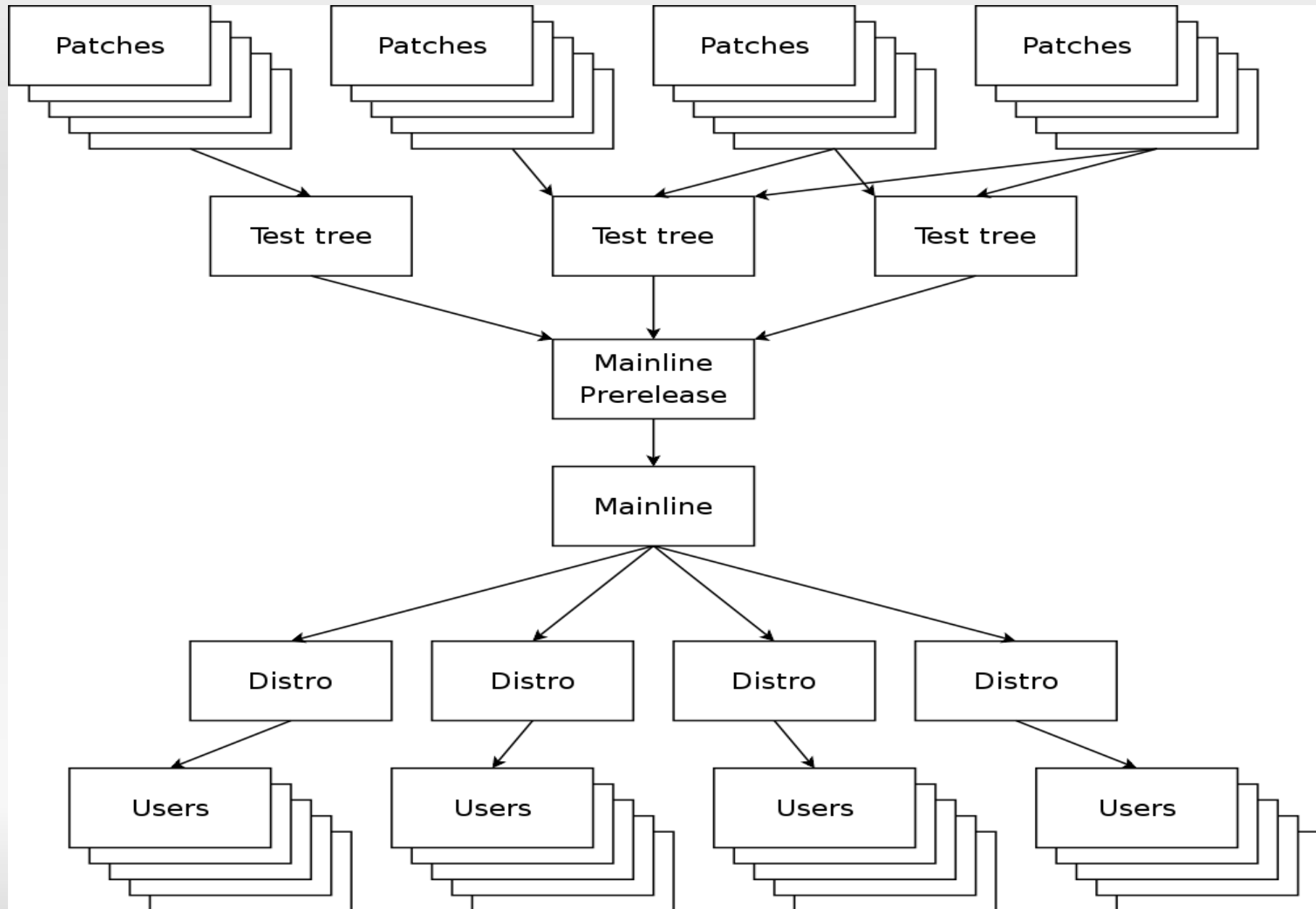
- Changes in the development model (no 2.7 tree)
- Increasing pace of development
- Linux more broadly deployed, higher expectations
- People are slow, and either expensive or unreliable.
- Automated testing is cheap, and fast.
- Consistent, available testers that don't disappear.
- Good debug setups - serial consoles, etc.
- Filter hardware problems by replication of machines.

# Solves world peace and hunger?

No.

- Skilled developers, careful development.
- Static code analysis
- Regular and rigorous code review
- Functional, regression, performance, and stress testing.
- If we can kill off 20% of bugs ... that's great.
- 1% is still useful!
- It's a tool to give you leverage, not a magic wand.

# Code flow of the Linux Kernel



# Why test upstream?

Test early. Test often.

- Prevent replication of bad code into other code bases
- Fewer users are exposed to the bug
- Code is still fresh in the authors mind
- Change isn't interacting with subsequent changes
- Code is at the top of the stack - easy to remove.
- You're going to hit the same bugs eventually anyway ...
- Goal is to run the SAME tests upstream as downstream

# State of the testing union

- Hardware vendors
- Distros
- ISVs
- LTP, STP, ABAT/autobench.
- [test.kernel.org](http://test.kernel.org)
- Developers are busy
- Fixing bugs is boring

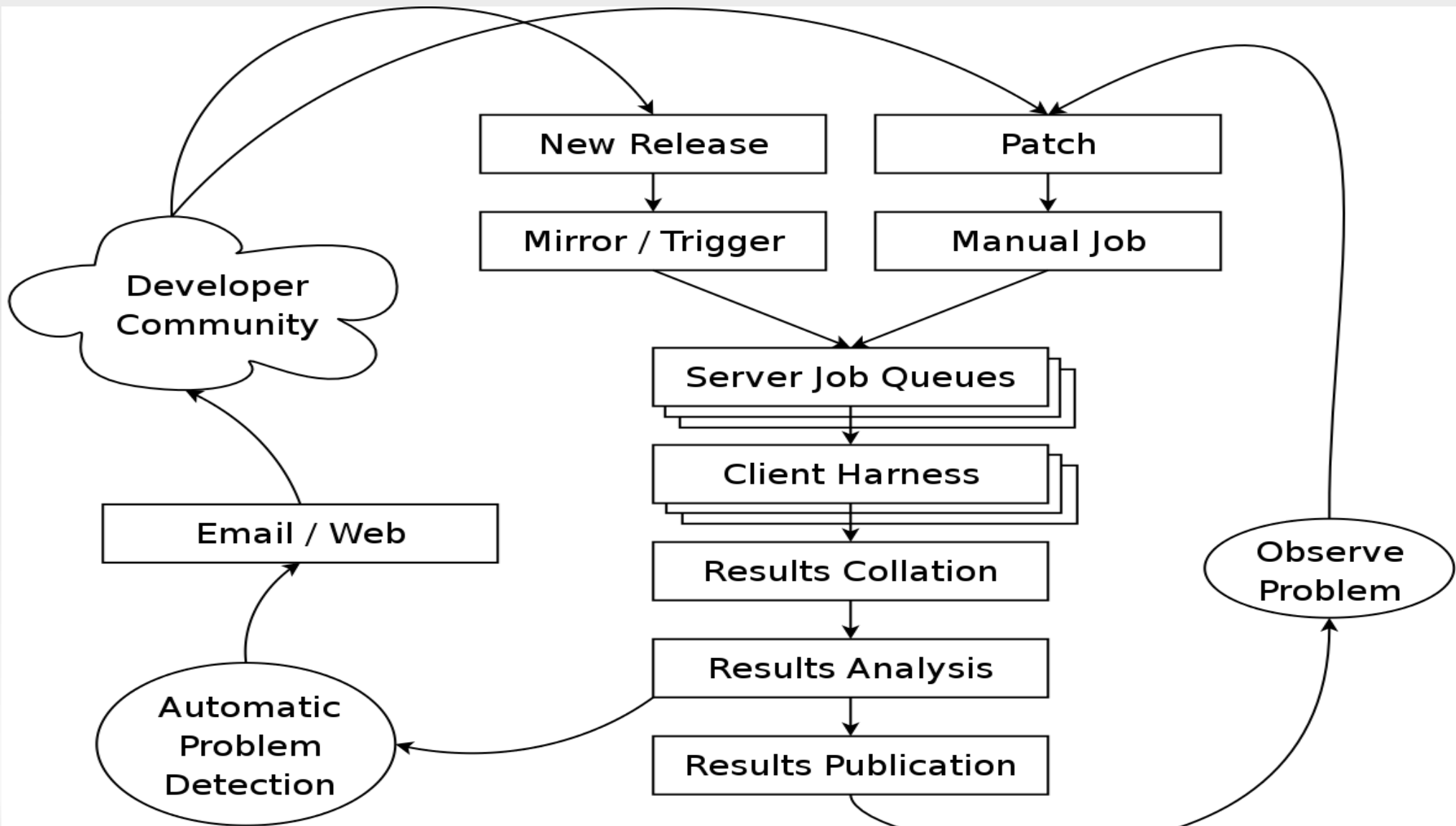
# What needs to change?

- We are great at sharing some code
- Terrible at sharing tests, and test results
- Open, pluggable client, share tests (autotest)
- More people testing upstream
- Collation of results from multiple sources (tko)
- Push results back to community (email + web)
- **FIX THE BUGS!**

# What is test.kernel.org (tko)?

- A collation, publication, and analysis engine.
- Currently only fed by proprietary IBM ABAT / autobench harness
- Switching over to autotest
- Will take collated input from many sources.

# TKO (test.kernel.org) workflow



# Design goals - client test harness

- Modular, simple, well defined APIs.
- Separate tests from core
- Powerful and flexible.
- Simple - low barrier to entry for test development
- Distributed, scalable development and maintainership
- Maintainable code
- Robust, consistent results. Good error handling.
- Isolate hardware failures.
- Consistent, easy to analyse results format

# Test harness features

## Wide range of functionality

- **Build**
- Static Verification (e.g. gcc, sparse, coverity)
- **Boot**
- Debug runs (turn on CONFIG\_DEBUG\_FOO)
- Functional / unit tests
- Performance tests
- Stress tests.
- Profiling and debugging.

# What tests are supported?

## Tests / Testsuites

aiostress, bonnie, cpu\_hotplug, cyclicttest, dbench, dbt2, fio, fs\_mark, fsx, interbench, iozone, isic, kernbench, libhugetlbf, lmbench, ltp, netperf2, pi\_tests, pktgen, reaim, rmaptest, crashme, selftest, sleeptest, sparse, stress, tbench, tiobench, unixbench, xmtest

## Profilers

catprofile, lockmeter, oprofile, readprofile

It only takes about 10 minutes to add a new test

# Example test - dbench

```
import test
from autotest_utils import *

class dbench(test.test):
    version = 1

    # http://samba.org/ftp/tridge/dbench/dbench-3.04.tar.gz
    def setup(self, tarball = 'dbench-3.04.tar.gz'):
        tarball = unmap_url(self.bindir, tarball, self.tmpdir)
        extract_tarball_to_dir(tarball, self.srcdir)
        os.chdir(self.srcdir)

        system('./configure')
        system('make')

    def execute(self, iterations = 1, nprocs = count_cpus(), args = ""):
        for i in range(1, iterations+1):
            args = args + ' -c '+self.srcdir+'/client.txt'
            args += ' %s' % nprocs
            system(self.srcdir + '/dbench ' + args)

    # Do a profiling run if necessary
```

# Multi-machine tests

## barrier support

- See netperf2 test for an example
- Barriers provide a basic synchronization point
- Name the barrier
- List all members of the barrier
- Specify a timeout (in seconds)
- Everybody does “rendevous” on that barrier

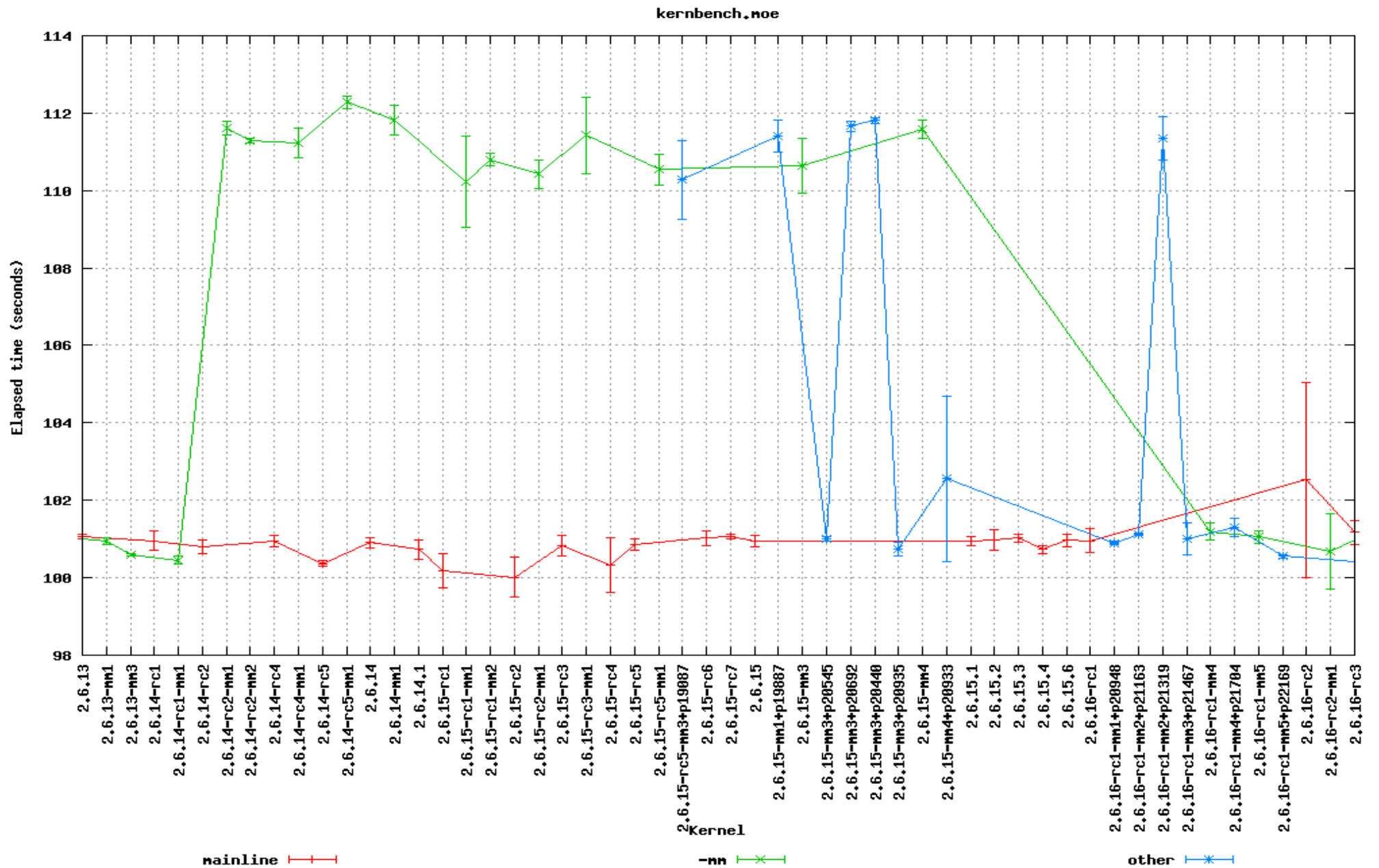
# Why Python?

- Bitter experience with previous projects written in bash and perl
- Easy to understand and modify (not write-only)
- Easy to maintain
- Easy to learn
- Simple language syntax – fewer hidden errors
- Exception handling
- Powerful
- Has a wide library of modules to leverage
- The spacing "issue"

# This is only the tip of the iceberg

- Static analysis
- Comparisons across releases
- Automated bisection search
- Simple server-side engine
- Better failure capture / debug tools
- Debug Options
  - NMI watchdog / alt+sysrq
  - Crashdump
- Collate results from variety of machines

# Performance analysis



# What help do we need?

- Download the test harness and experiment with it
- Need tests that find real bugs!
- Itp, Imbench need work
- Support for Xen
- More machines running tests
- It's beta code
- Results analysis / comparison

# Thanks to ...

- Andy Whitcroft
- IBM, Google, OSL, OSDL
- The autotest development community