Continuous Performance Testing

Mark Price / @epickrram

Performance Engineer

Improbable.io

The ideal

System performance testing as a first-class citizen of the continuous delivery pipeline

Process

Process maturity

A scientific and rigorous survey

Process maturity

A ~~scientific and rigorous~~ survey

Process maturity

“As part of QA, the whole team logs on to the system to make sure it scales”

Process maturity

“We have some hand-rolled benchmarks that prove our code is fast”

Process maturity

“We use a well-known testing framework for our benchmarks”

Process maturity

“Our benchmarks are run as part of CI”

Process maturity

“Trend visualisations of system performance are available”

Process maturity

“There is a release gate on performance regression”

Increasing process maturity

Implies:

Higher maintenance cost

Greater confidence

Scopes

Performance test scopes

- Nanobenchmarks
- Microbenchmarks
- Component Benchmarks
- System performance tests

Nanobenchmarks

- Determine the cost of something in the underlying platform or runtime
- How long does it take to retrieve System.nanoTime()?
- What is the overhead of retrieving AtomicLong vs long?
- Invocation times on the order of 10s of nanoseconds

Nanobenchmarks

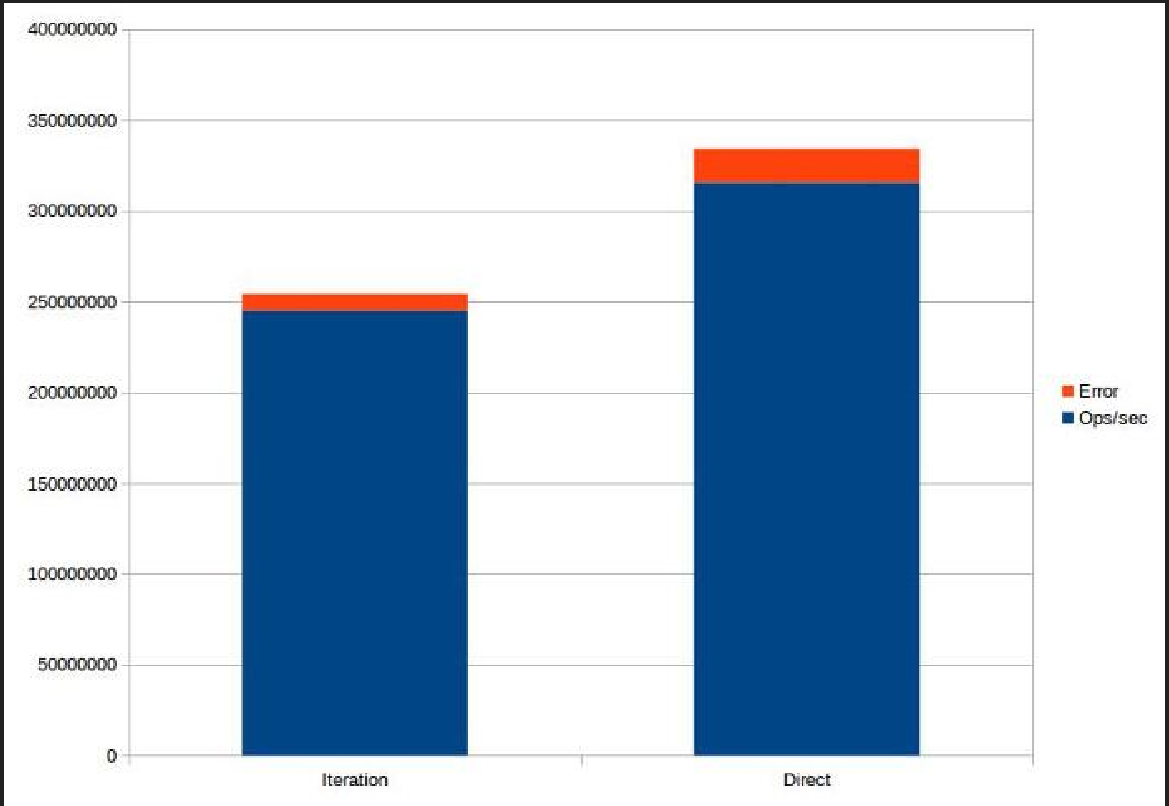
- Susceptible to jitter in the runtime/OS
- Unlikely to need to regression test these...
- Unless called **very** frequently from your code

Message callback

```
@Benchmark
@BenchmarkMode(Mode.Throughput)
@OutputTimeUnit(TimeUnit.SECONDS)
public void singleCallback(final Blackhole blackhole)
{
    callback.accept(blackhole);
}

@Benchmark
@BenchmarkMode(Mode.Throughput)
@OutputTimeUnit(TimeUnit.SECONDS)
public void singleElementIterationCallback(final Blackhole blackhole)
{
    for (Consumer<Blackhole> objectConsumer : callbackList)
    {
        objectConsumer.accept(blackhole);
    }
}
```

Message callback



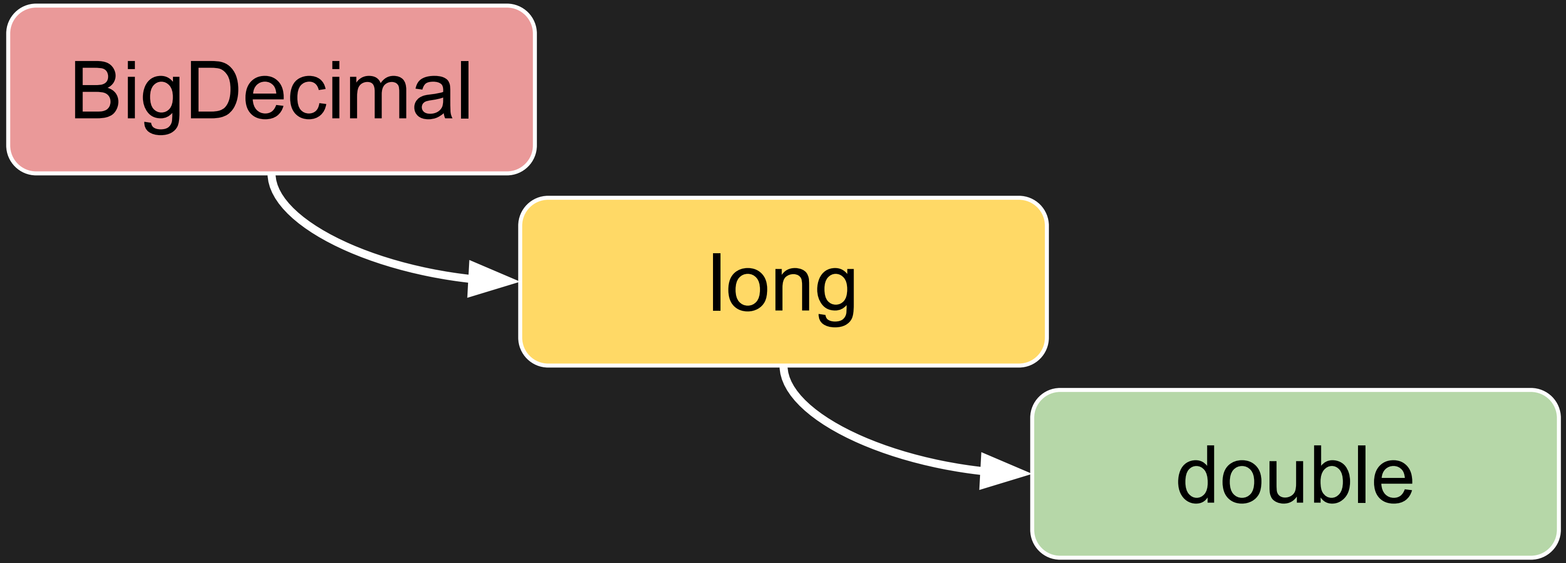
Microbenchmarks

- Test small, critical pieces of infrastructure or logic
- E.g message parsing, calculation logic
- These should be regression tests
- We own the code, so assume that we’re going to break it
- Same principle as unit & acceptance tests

Microbenchmarks

- Invaluable for use in optimising your code (if it is a bottleneck)
- Still susceptible to jitter in the runtime
- Execution times in the order of 100s of nanos/single-digit micros
- Beware bloat

Risk analysis - long vs double



BigDecimal

long

double

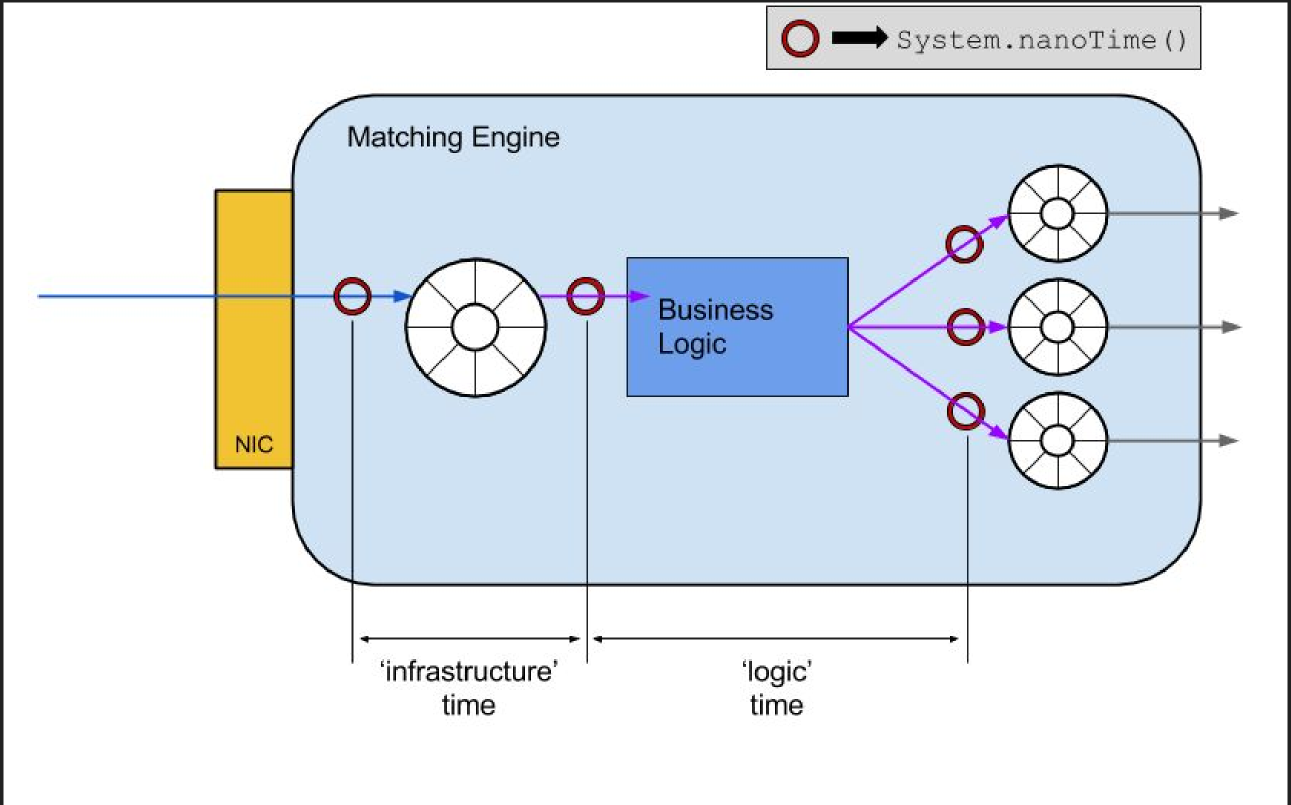
Component benchmarks

- ‘Service’ or ‘component’ level benchmarks
- Whatever unit of value makes sense in the codebase
- Wire together a number of components on the critical path
- We can start to observe the behaviour of the JIT compiler (i.e. inlining)

Component benchmarks

- Execution times in the 10s - 100s of microseconds
- Useful for reasoning about maximum system performance
- Runtime jitter less of an issue, as things like GC/de-opts might start to enter the picture
- Candidate for regression testing

Matching Engine - no-ops are fast!



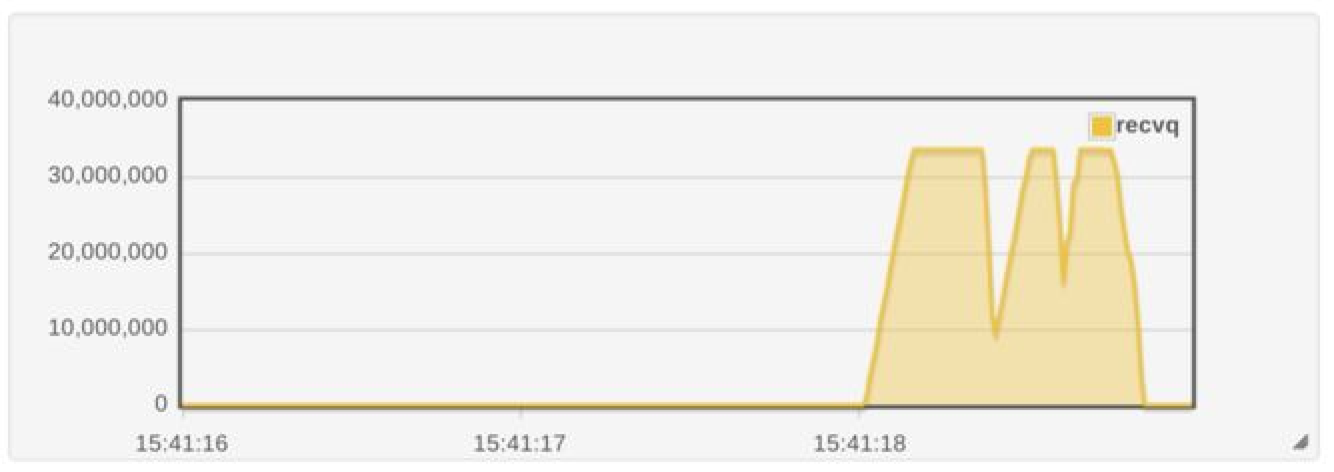
System performance tests

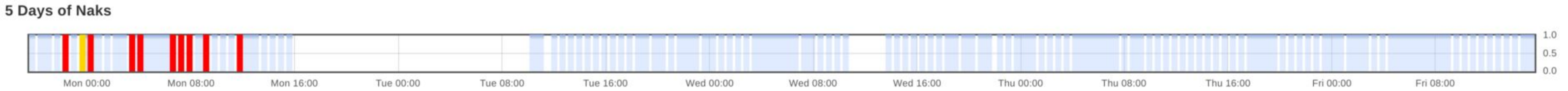
- Last line of defence against regressions
- Will catch host OS configuration changes
- Costly, requires hardware that mirrors production
- Useful for experimentation
- System recovery after failure
- Tools developed for monitoring here should make it to production

System performance tests

- Potentially the longest cycle-time
- Can provide an overview of infrastructure costs (e.g network latency)
- Red-line tests (at what point will the system fail catastrophically)
- Understand of interaction with host OS more important
- Regressions should be visible

Page fault stalls

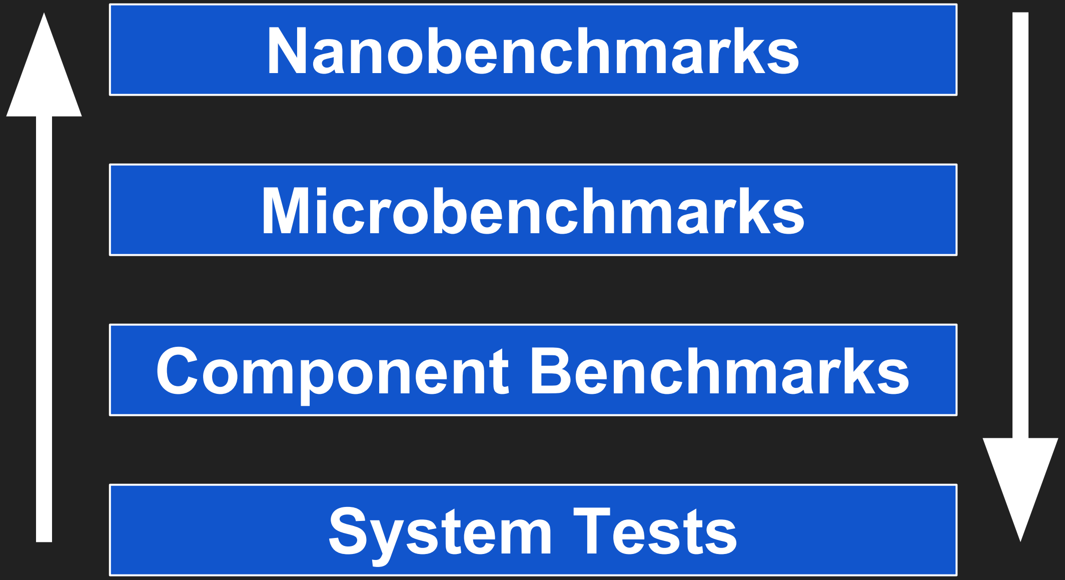




http://digihippo.net/?p=241

Performance testing trade-offs

- Faster feedback
- System jitter magnified
- Fewer moving parts
- Stability



Nanobenchmarks

Microbenchmarks

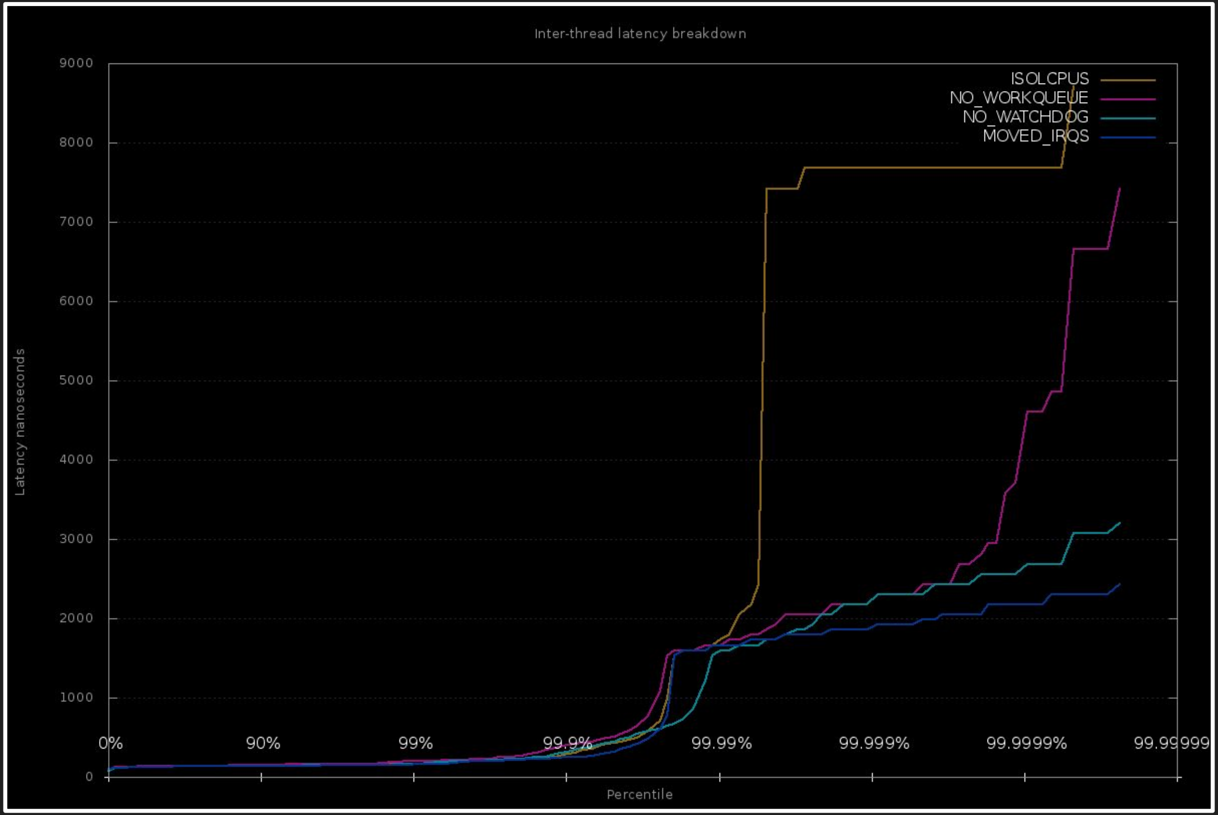
Component Benchmarks

System Tests

- Slower feedback
- Hardware cost
- Maintenance cost
- KPI/SLA indicator
- Realism

Measurement

System jitter is a thing



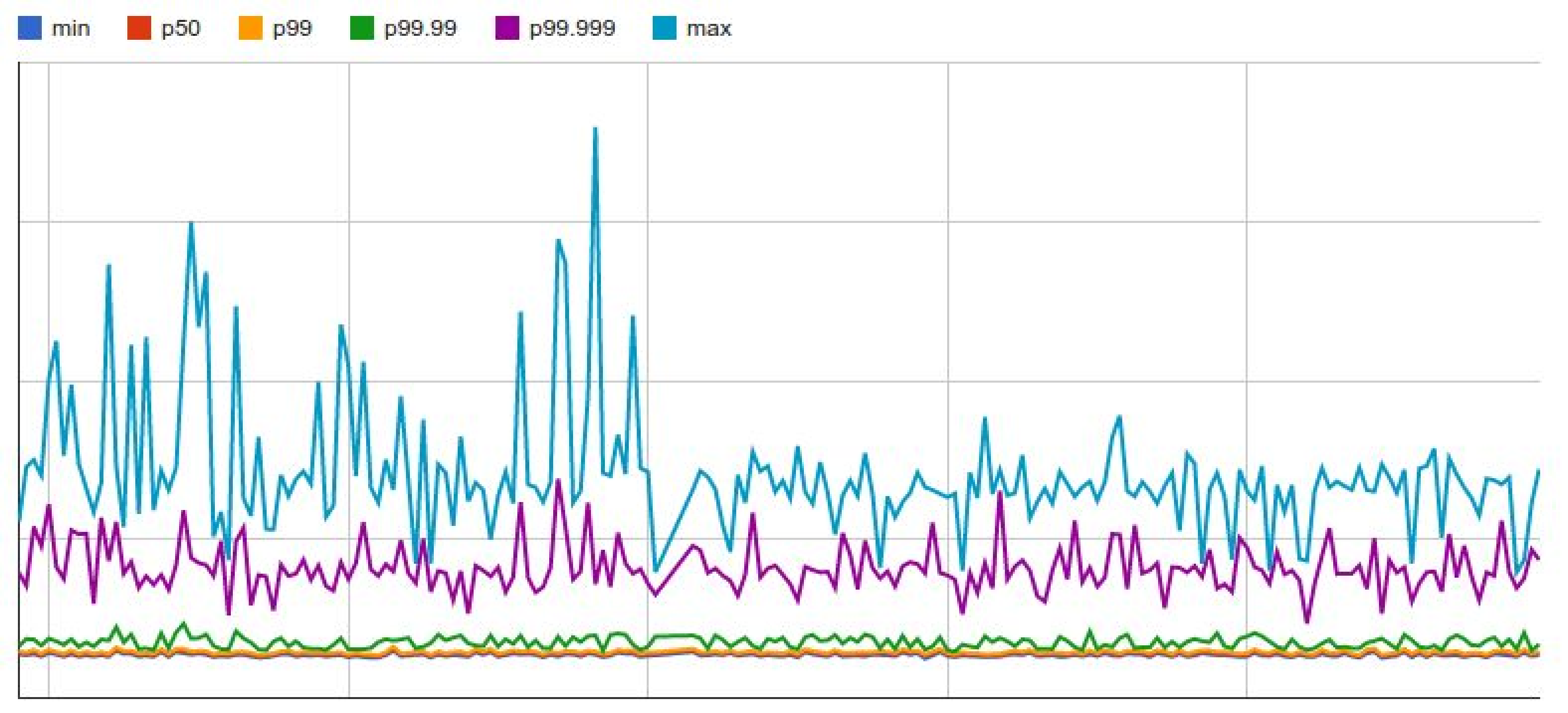
Reducing runtime jitter

Histogram of invocation times (via JMH)

Run-to-run variation

Large error values around average

Reducing runtime jitter



Measurement apparatus

Use a proven test-harness

If you can’t:

Understand coordinated omission

Measure out-of-band

Look for load-generator back-pressure

Production-grade tooling

Monitoring and tooling used in your performance environment should be productionised

Containers and the cloud

Measure the baseline of system jitter

Network throughput & latency: understand what is an artifact of our system and what is the infrastructure

End-to-end testing is more important here since there are many more factors at play adding to latency long-tail

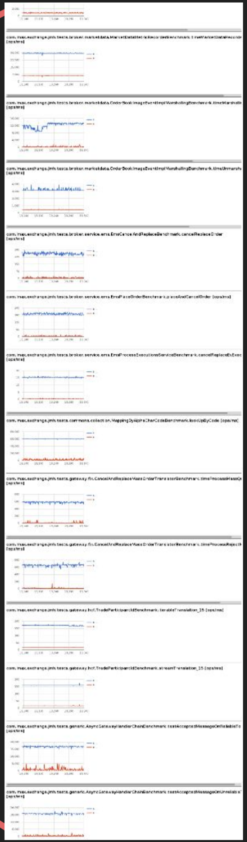
Reporting

Charting

“Let’s chart our benchmark results so we’ll see if there are regressions”

Charting

Charting



com.max.exchange.jmh.tests.broker.marketdata.MarketDataMetricRecorderBenchmark.timeMarketDataRecorde [ops/ms]

com.max.exchange.jmh.tests.broker.marketdata.OrderBookImageEventImplMarshallingBenchmark.timeMarshallir [ops/ms]

com.max.exchange.jmh.tests.broker.marketdata.OrderBookImageEventImplMarshallingBenchmark.timeUnmarsh [ops/ms]

com.max.exchange.jmh.tests.broker.service.ems.EmsCancelAndReplaceBenchmark.cancelReplaceOrder [ops/ms]

com.max.exchange.jmh.tests.broker.service.ems.EmsPlaceOrderBenchmark.placeAndCancelOrder [ops/ms]

com.max.exchange.jmh.tests.broker.service.ems.EmsProcessExecutionsServiceBenchmark.cancelReplaceByExec [ops/ms]

com.max.exchange.jmh.tests.commons.collection.MappingByAlphaCharCodeBenchmark.lookUpByCode [ops/ms]

com.max.exchange.jmh.tests.gateway.fix.CancelAndReplaceMassOrderTranslatorBenchmark.timeProcessMassO [ops/ms]

com.max.exchange.jmh.tests.gateway.fix.CancelAndReplaceMassOrderTranslatorBenchmark.timeProcessRejecti [ops/ms]

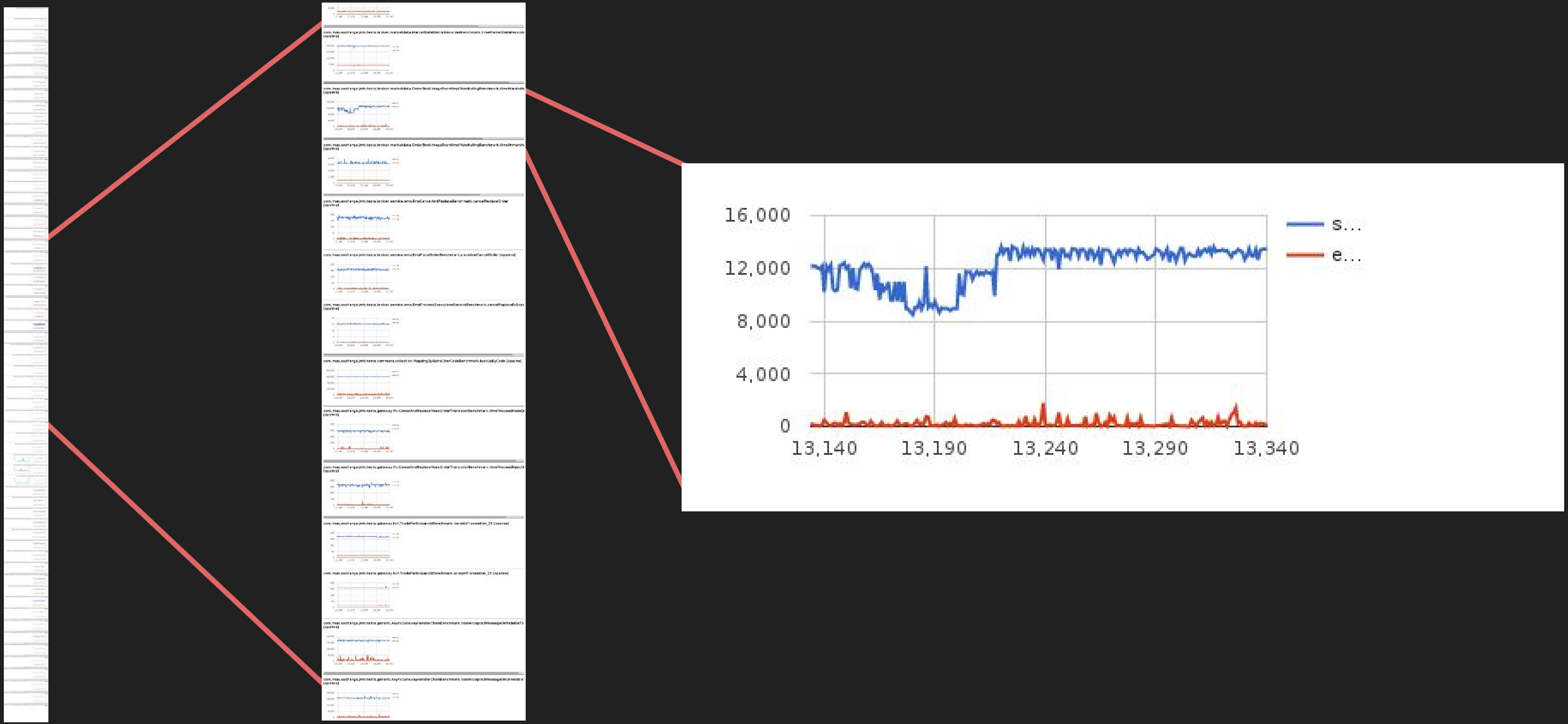
com.max.exchange.jmh.tests.gateway.hcf.TradeParticipantIdBenchmark.iterableTranslation_15 [ops/ms]

com.max.exchange.jmh.tests.gateway.hcf.TradeParticipantIdBenchmark.streamTranslation_15 [ops/ms]

com.max.exchange.jmh.tests.generic.AsyncGatewayHandlerChainBenchmark.testAcceptedMessageOnReliableTo [ops/ms]

com.max.exchange.jmh.tests.generic.AsyncGatewayHandlerChainBenchmark.testAcceptedMessageOnUnreliable [ops/ms]

Charting



Charting

Make a computer do the analysis

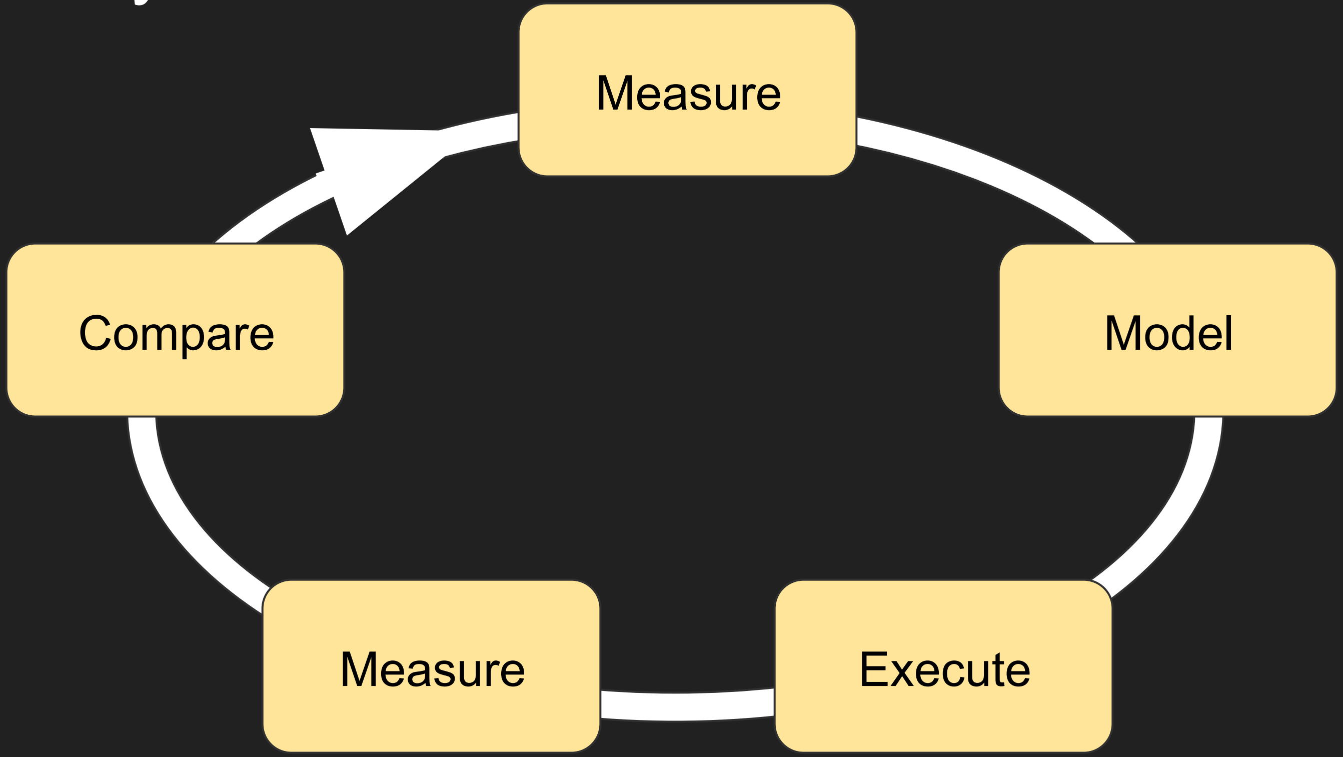
We automated manual testing, we should automate regression analysis

Then we can selectively display charts

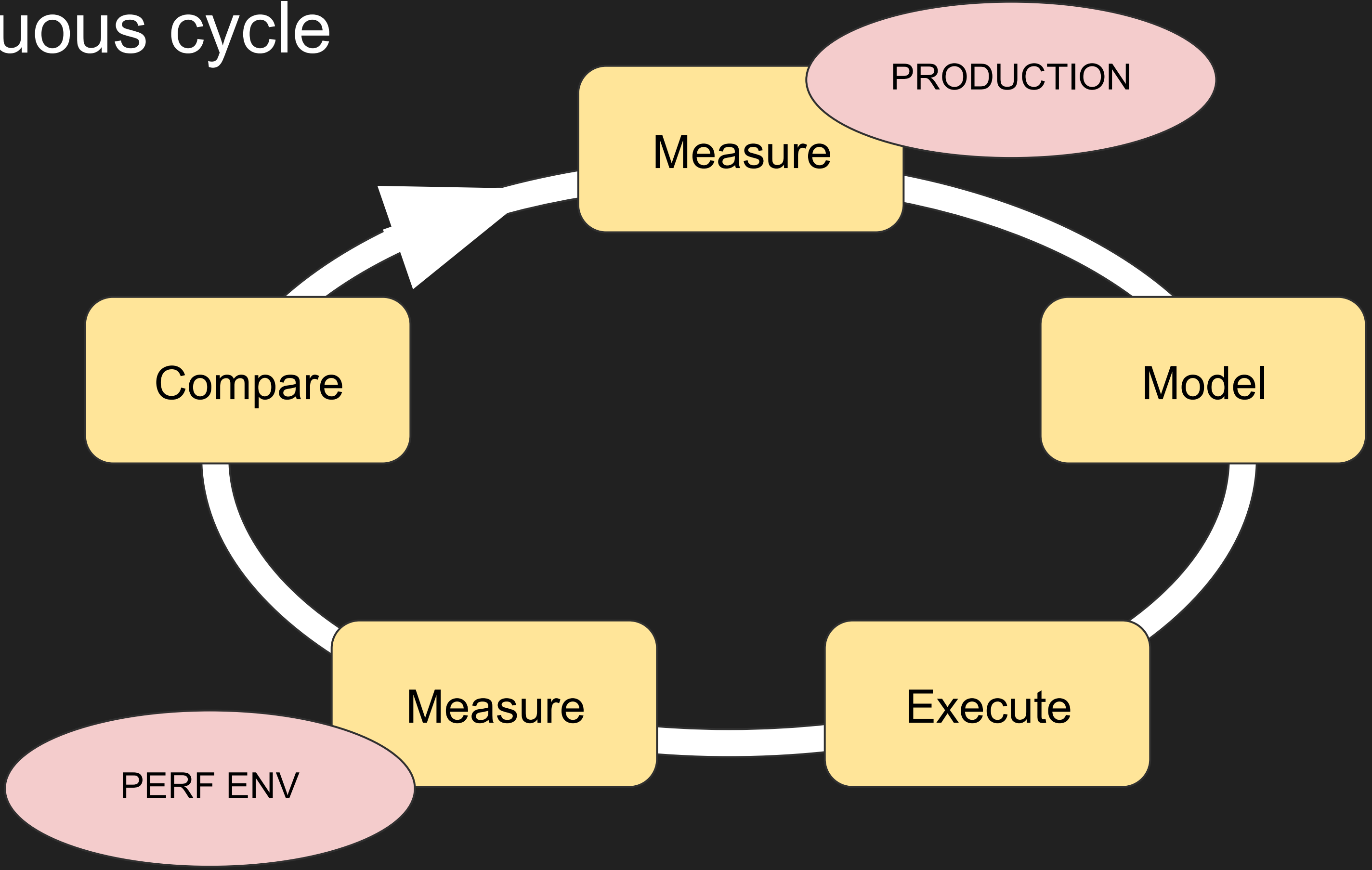
Explain the screen in one sentence, or break it down

Improvement

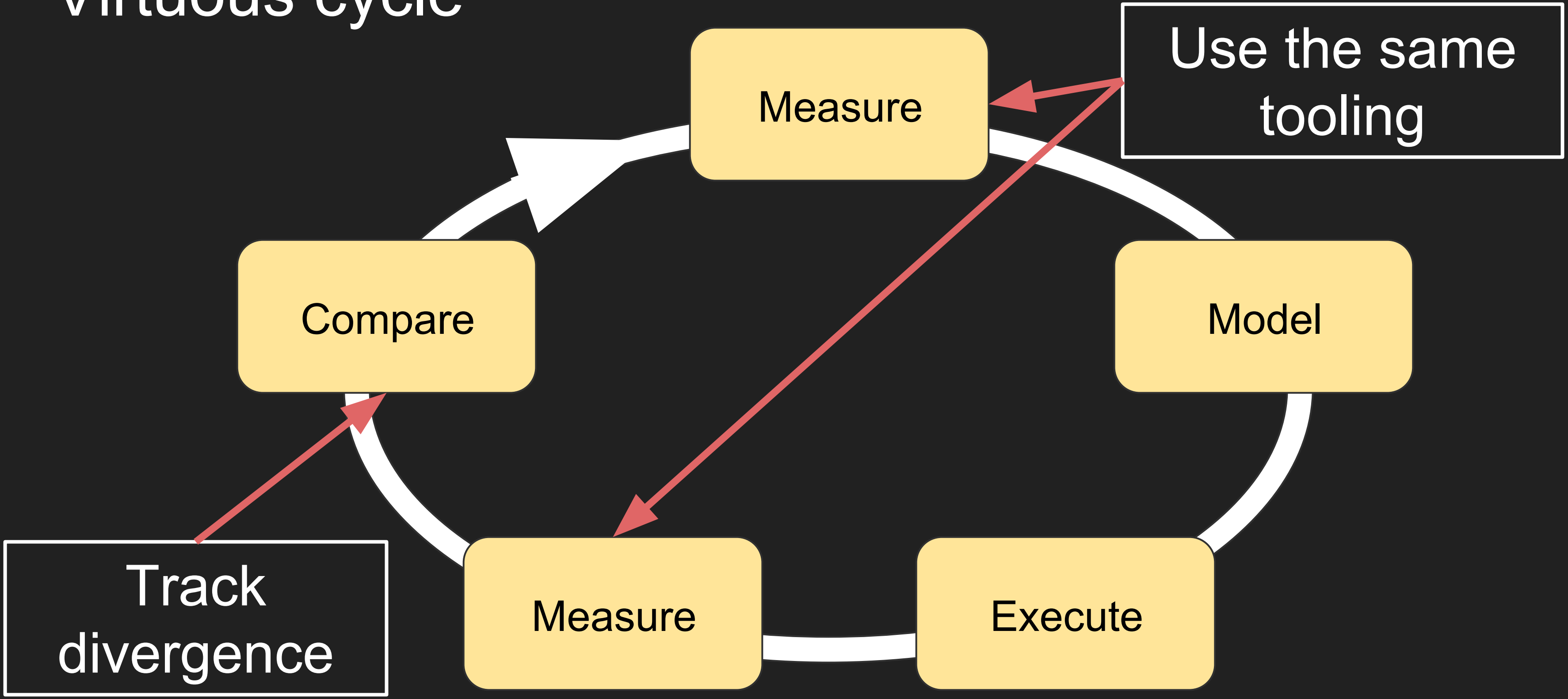
Virtuous cycle



Virtuous cycle



Virtuous cycle



Measure

Model

Execute

Measure

Compare

Use the same tooling

Track divergence

Regression tests

If we find a performance issue, try to add a test that demonstrates the problem

This helps in the investigation phase, and ensures regressions do not occur

Be careful with assertions

In a nutshell...

Key points

Use a known-good framework if possible

If you have to roll your own: peer review, measure it, understand it

Data volume can be oppressive, use or develop tooling to understand results

Test with realistic data/load distribution

Key points

Are we confident that our performance testing will catch regressions before they make it to production?

Thank you!

- @epickrram
- https://epickrram.blogspot.com
- recruitment@improbable.io