Performance Testing Java Applications

Martin Thompson - @mjpt777

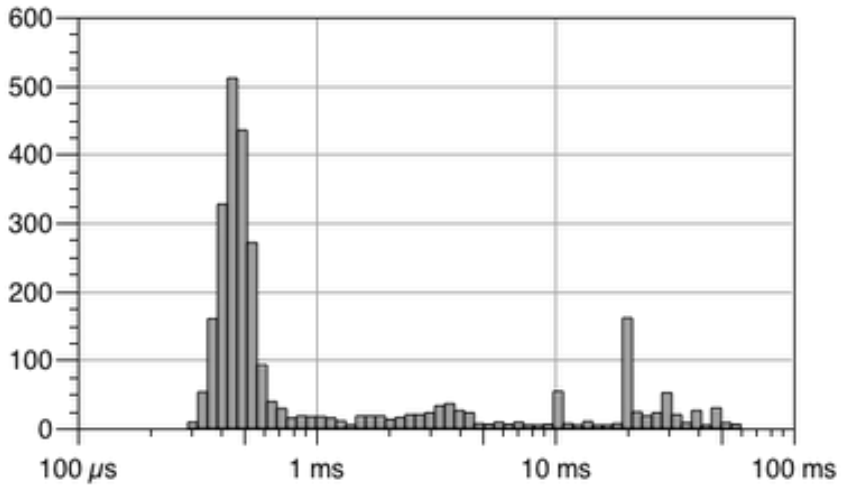
What is Performance?

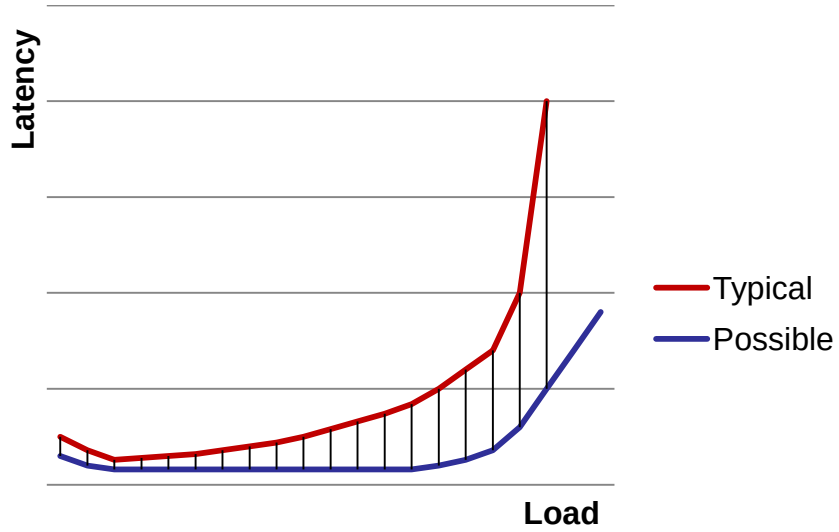
Throughput / Bandwidth

Latency / Response Time

Throughput vs. Latency

- How does an application cope under burst conditions?
- Are you able to measure queuing delay?
- Back-off strategies and other effects
- Amortise the expensive operations – Smart Batching





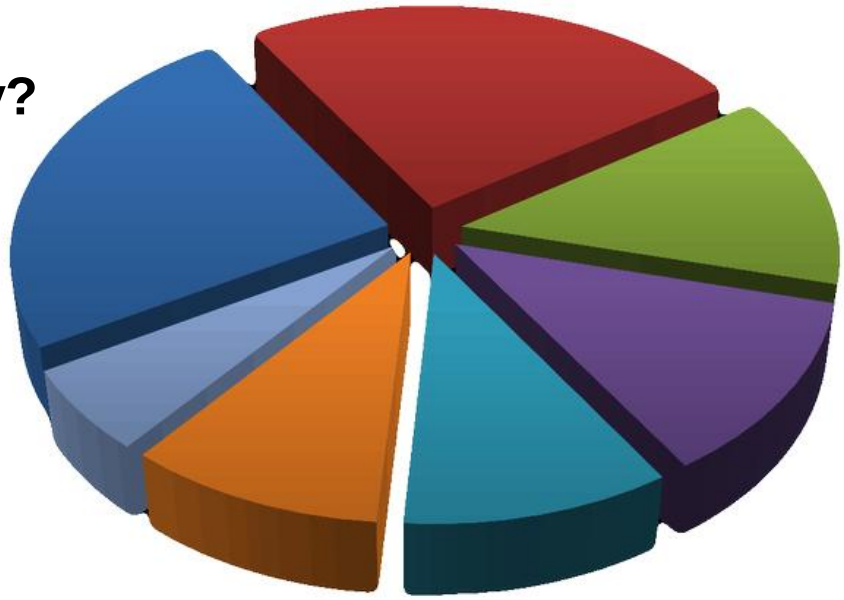
Performance Requirements

Performance Requirements

- What throughput and latency does your system require?
  - Do you need to care?
  - How will you be competitive?
  - Does performance drive business?
- Investments start from a business plan
  - Work with the business folk
  - Set Transaction Budget limits
- As the business scales does the software scale in an economic fashion?
  - Don’t limit design options

Decompose the Transaction Budget

- How much time is each layer in the architecture allowed?
- Do all technology choices pay their way?
  - Think Aircraft or Spacecraft!
- Profile to ensure budget is enforced
- What happens when throughput increases?
  - Queuing delay introduced?
  - Scale out at constant latency?



X µs Total with Breakdown

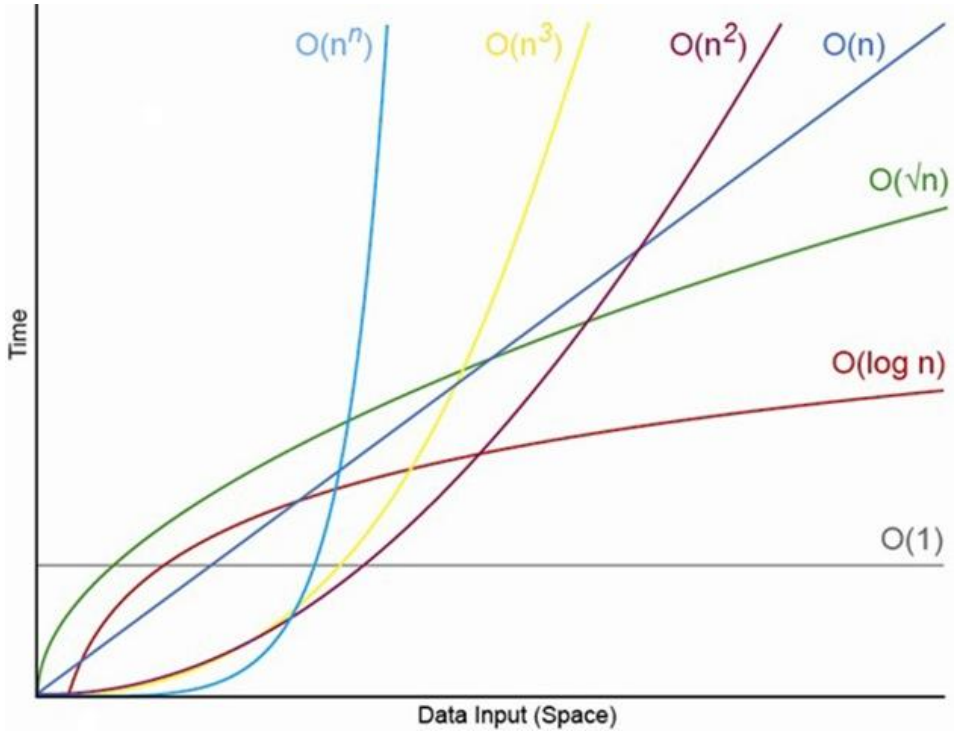
How can we test Performance?

Types of Performance Testing

1. Throughput / Bandwidth Testing
2. Latency / Response Testing
3. Stress Testing
4. Concurrent / Contention Testing
5. Endurance / Soak Testing
6. Capacity Testing

Understand Algorithm Behaviour

- Need to model realistic scenarios
  - Read to write ratios
  - Distribution across data sets
  - No single entity / item tests!
- Model based on production
- Are unbounded queries allowed?
  - Deal in manageable chunks

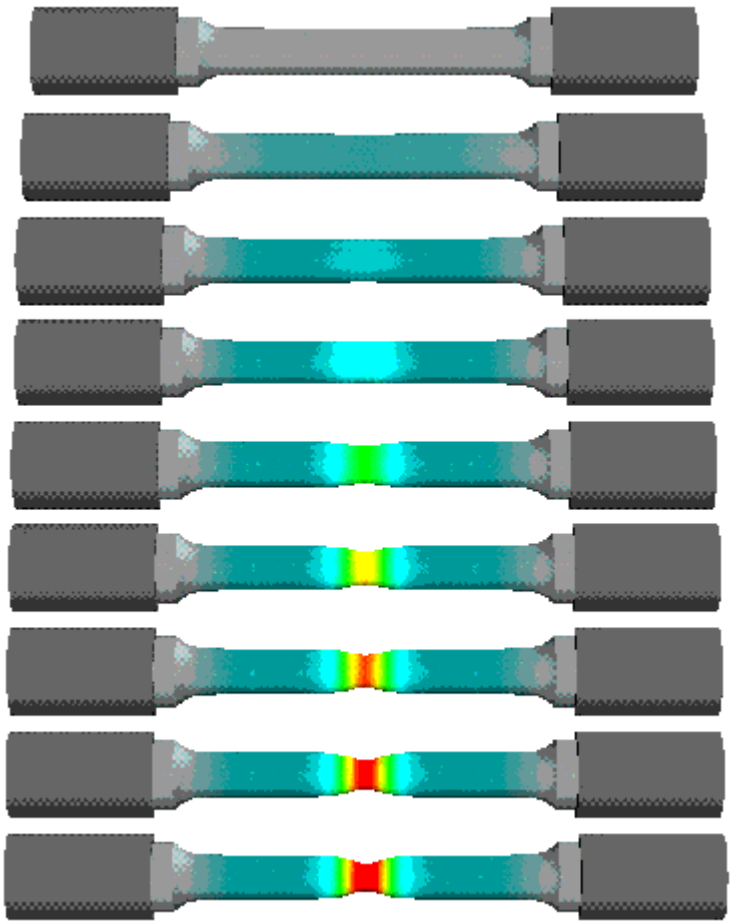


The “Onion”

- Separation of Concerns is key
  - Layer your architecture
- Test individual components
- Test assemblies of components with a focus on interactions
  - Beware micro-benchmarking!
- Test core infrastructure
  - Useful for catching upgrade issues
- Same patterns at different levels of scale

Know Your Platform/Infrastructure

- **Stress test until breaking point**
  - Do things degrade gracefully?
  - Do things crash?
  - Order of the algorithms?
- **What are the infrastructure capabilities?**
  - Profile to know relative costs of components
  - Operations Per Second
  - Bandwidth
  - Latency
  - Endurance
- **What happens when redundant components take over?**



When should we test Performance?

Performance Test & Profile

"Premature optimization is the root of all evil"
– Donald Knuth / Tony Hoare

- What does "optimization" mean?
  - Specialisation vs. Flexibility?
  - Very different from knowing your system capabilities
  - Test / profile early and often

- Integrate performance testing to CI
- Monitor production systems
- Change your development practices...

Development Practices

- Performance *"Test First"*
- Red, Green, Debug, Profile, Refactor...
  - A deeper understanding makes you faster
- Use *"like live"* pairing stations
- Don't add features you don't need
- Poor performance should fail the build!

Performance Testing in Action

The Java Pitfalls

- Runtime Compiler
  - JIT & On Stack Replacement (OSR)
  - Polymorphism and In-lining
  - Dead code elimination
  - Race conditions in optimisation
- Garbage Collection
  - Which collector - Dev vs. Production
  - Skewed results from pauses
  - Beware “Card Marking”
- Class Loading

Micro Benchmarking

- Framework should handle warm up
- Representative Data Sets
  - Vary set size
- Key Measures
  - Ops Per Second (per thread)
  - Allocation rates
- Concurrent Testing
  - Scaling effects with threads
  - Queuing effects

```
public class MyBenchmark
     extends Benchmark
{
     public void timeMyOp(int reps)
     {
          int i = reps + 1;
          while (--i != 0)
          {
               MyClass.myOperation();
          }
     }
}
```

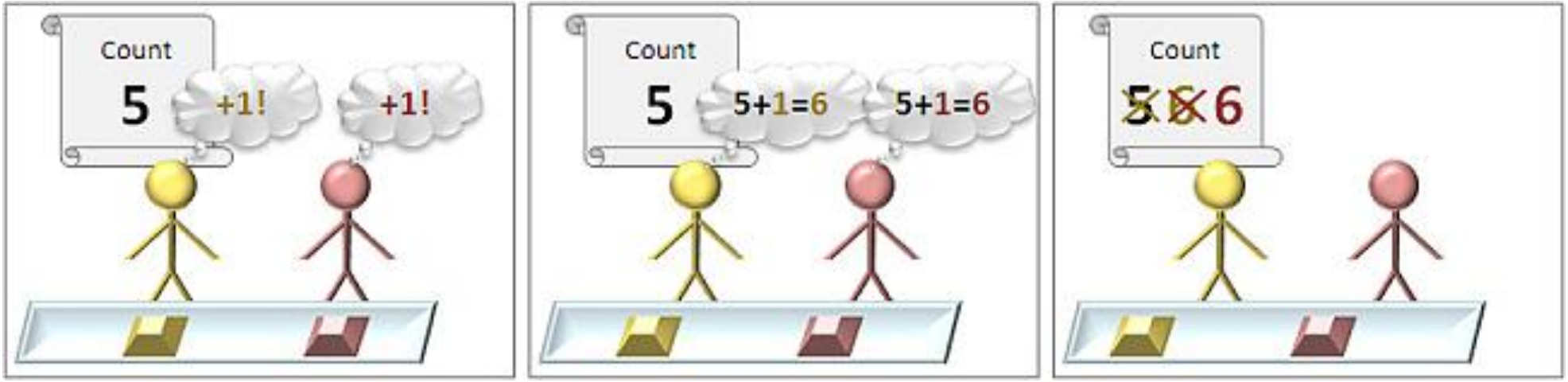
Anatomy Of A Micro Benchmark

```
public class MapBenchmark
    extends Benchmark
{
    private int size;
    private Map<Long, String> map = new MySpecialMap<Long, String>();

    private long[] keys;
    private String[] values;

    // setup method to init keys and values

    public void timePutOperation(int reps)
    {
        for (int i = 0; i < reps; i++)
        {
            map.put(keys[i], values[i]);
        }
    }
}
```

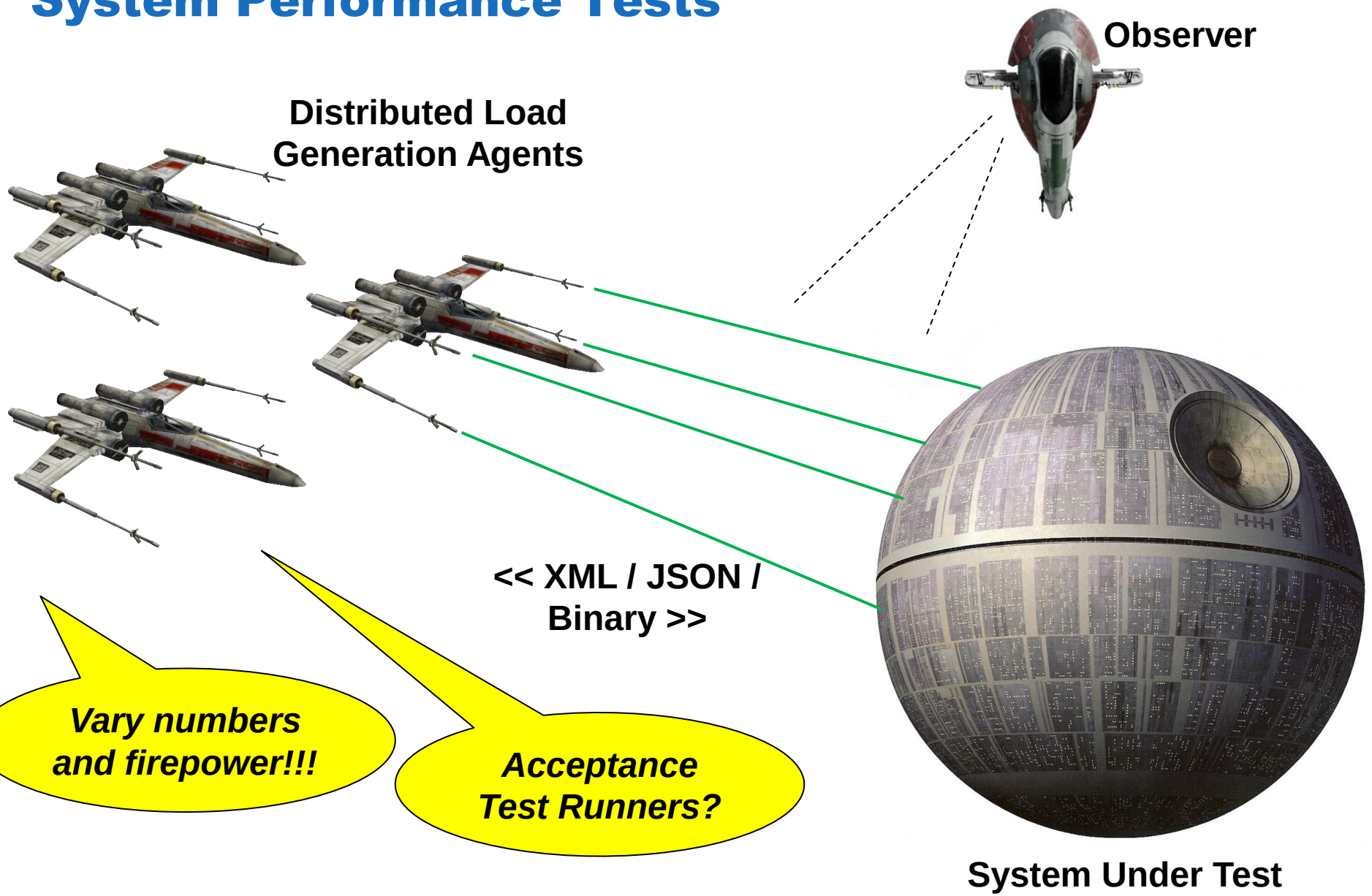
Performance Testing Concurrent Components

- **Straight Performance Tests**
  - **Ramp number of threads for plotting scaling characteristics**
  - **Measure Ops / Sec throughput – Averages vs. Intervals**
  - **Measure latency for queuing effects**

- **Validating Performance Tests**
  - **Check invariants and sequences**



System Performance Tests



System Performance Testing Analysis

- Build a latency histogram for given throughput
  - Disruptor Histogram, HdrHistogram
  - Investigate the outliers!

- Gather metrics from the system under test
  - Design system to be instrumented
  - Don’t forget the Operating System
  - Plot metrics against latency and throughput
  - Capacity planning from this is possible

- Generate micro-bursts
  - They show up queuing effects at contention points
  - Uncover throughput bottlenecks

Got a Performance Issue?

Performance Profiling

- Java Applications
  - JVisualVM, YourKit, Solaris Studio, etc
  - What is the GC doing?
  - Learn bytecode profiling
- Operating System
  - htop, iostat, vmstat, pidstat, netstat, etc.
- Hardware
  - Perf Counters – perf, likwid, VTune
- Follow Theory of Constraints for what to tackle!

Performance Testing Lessons

Mechanical Sympathy

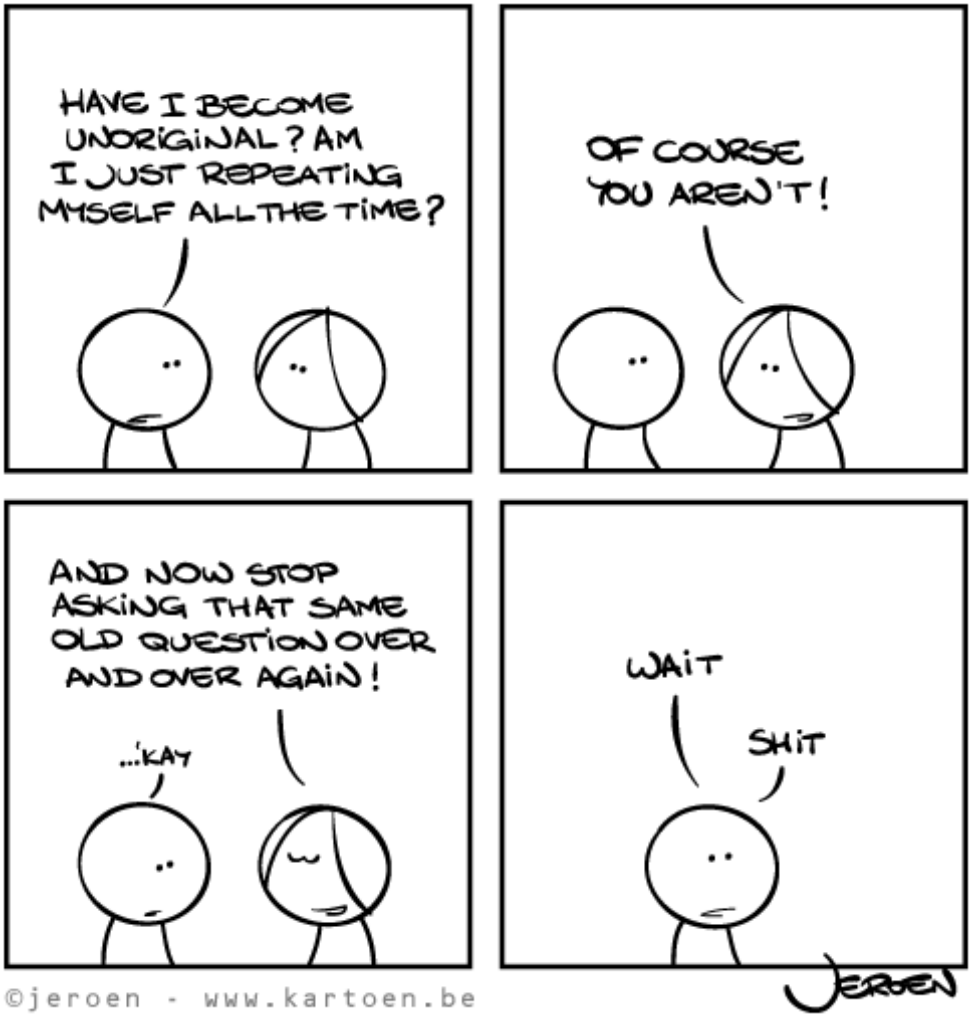
- Java Virtual Machines
  - Garbage Collection
  - Optimization
  - Locks
- Operating Systems
  - Schedulers
  - Virtual Memory
  - File Systems & IO
- Hardware
  - Hardware capabilities and interactions
  - Profile the counters for greater understanding

The Issues With “Time”

- NTP is prone to time correction
  - Careful of `System.currentTimeMillis()`
- Monotonic time not synchronised across sockets
  - `System.nanoTime()` is monotonic
  - RDTSC is not an ordered instruction
- Not all servers and OSes are equal
  - Pre Nehalem TSC was not invariant
  - Older OSes and VT can be expensive
  - Resolution is highly variable by OS/JVM

Beware Being Too Predictable

- CPU Branch Prediction
  - Fake Orders
  - Taking same path in code
- Cache hits
  - Disk loaded into memory
  - Memory loaded into CPU cache
  - Application level caching



Beware YAGNI

The “Performance Team” Anti-Pattern

- They struggle to keep up with rate of change
- Performance testing is everyone's responsibility
- Better to think of a “Performance Team” as a rotating R&D exercise
- Performance specialists should pair on key components and spread knowledge

Lame Excuses - “It’s only …”

- It is only start-up code...
  - MTTF + MTTR
- It is only test code...
  - Feedback cycles!
- It is only UI code...
  - Read “High Performance Web Sites” by Steve Souders

Questions?

Blog: http://mechanical-sympathy.blogspot.com/

Twitter: @mjpt777

Links:

https://github.com/giltene/HdrHistogram

https://github.com/LMAX-Exchange/disruptor/blob/master/src/main/java/com/lmax/disruptor/collections/Histogram.java

https://code.google.com/p/caliper/

http://grinder.sourceforge.net/

http://www.javaworld.com/javaworld/jw-08-2012/120821-jvm-performance-optimization-overview.html