

www.qconferences.com
www.qconbeijing.com



QCon北京2014大会 4月25日—27日

伦敦 | 北京 | 东京 | 纽约 | 圣保罗 | 上海 | 旧金山

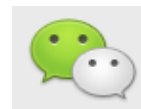
London · Beijing · Tokyo · New York · Sao Paulo · Shanghai · San Francisco

QCon全球软件开发大会

International Software Development Conference



@InfoQ



infoqchina

软件
正在改变世界!

特别感谢 QCon上海合作伙伴





Top 10 Performance Folklore

Martin Thompson - @mjpt777

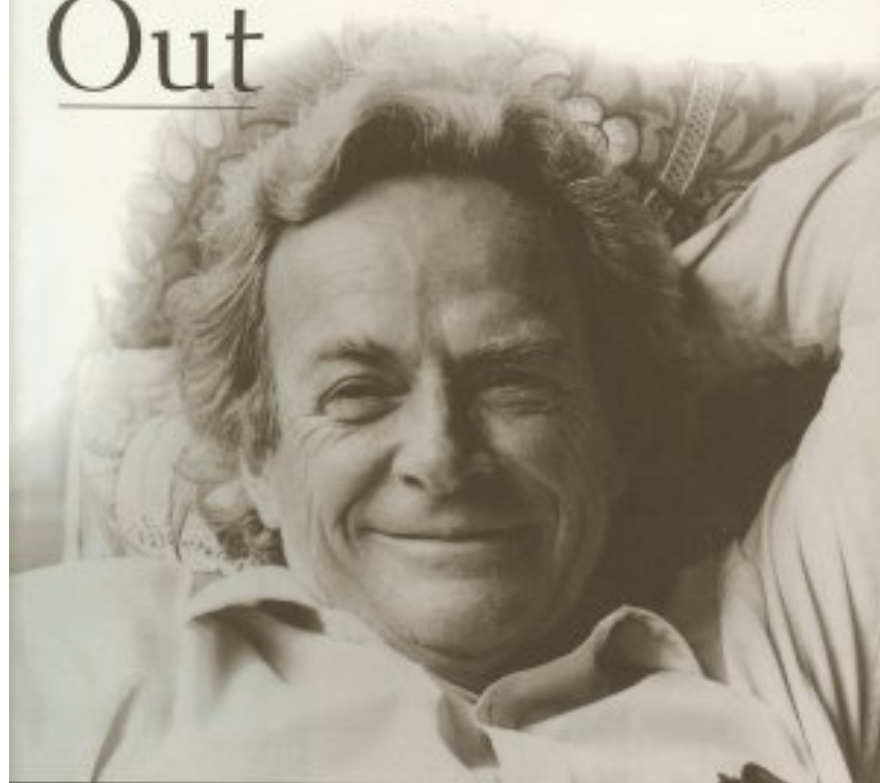


FANBOIS EVERYWHERE

| SEE FANBOI PEOPLE

"Feynman at his idiosyncratic, brilliant best."
—John Horgan, author of *The Undiscovered Mind*

The Pleasure of Finding Things Out



THE BEST
SHORT
WORKS OF

RICHARD P. FEYNMAN

Foreword by Freeman Dyson

“... as I’d like to show Galileo our world, I must show him something with a great deal of shame. If we look away from the science and look at the world around us, we find out something rather pitiful: that the environment that we live in is so actively, intensely unscientific.”

– Richard Feynman

***“How would you design an experiment
to answer that?”***

– Melville Arthur Feynman

Folklore

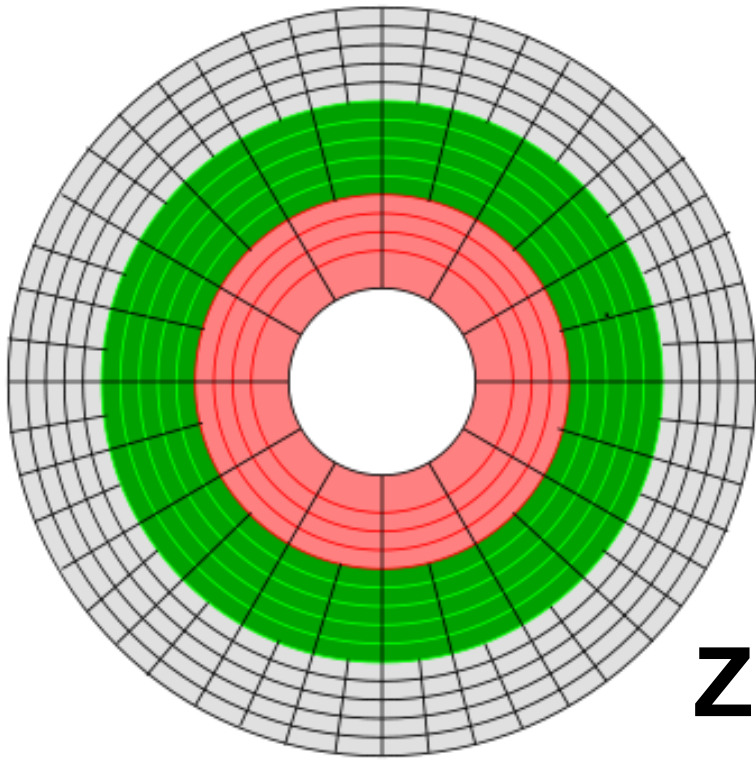
- *A body of widely held but false or unsubstantiated beliefs*



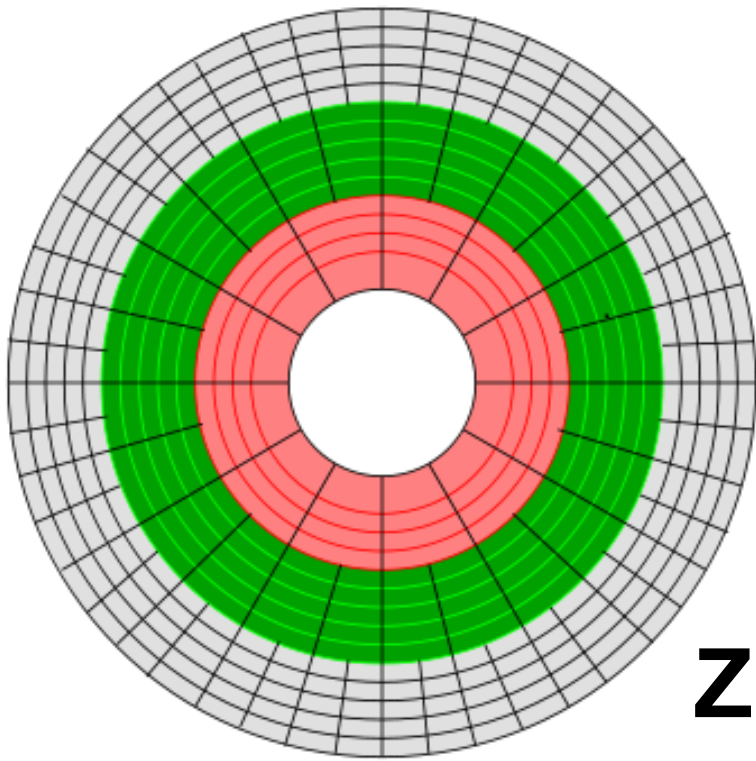
Performance Folklore

10

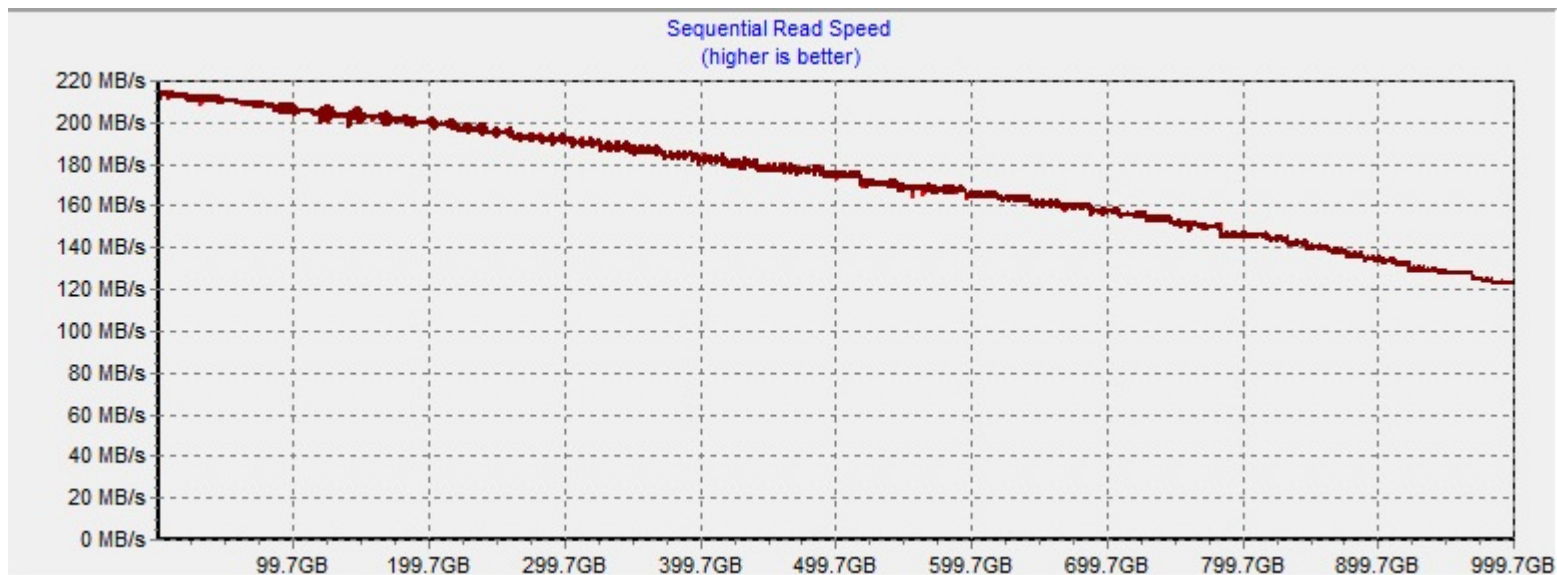
“Disk is random access”



Zone Bit Recording (ZBR)



Zone Bit Recording (ZBR)



$$\text{IO} = \text{Cmd} + \text{Seek} + \text{Rotation} + \text{Transfer}$$

IO = Cmd + Seek + Rotation + Transfer

Random 4K Block

~6ms = 0.1 + 3 + 3 + 0.02

IO = Cmd + Seek + Rotation + Transfer

Random 4K Block

~6ms = 0.1 + 3 + 3 + 0.02

~664KB/s vs. ~200MB/s !!!

9

“CPUs are not getting any faster”

“CPUs are not getting any faster?”

**Text tokenization of
“Alice in Wonderland”**

“CPUs are not getting any faster?”

Text tokenization of “Alice in Wonderland”

| Micro Architecture | Model | Ops/sec |
|--------------------|---------------------|---------|
| Core | P8600 @ 2.40GHz | 1434 |
| Nehalem | E5620 @ 2.40GHz | 1768 |
| Sandy Bridge | i7-2720QM @ 2.20GHz | 2674 |

Nehalem 2.8GHz

=====

\$ perf stat <program>

| | | | |
|----------------|-------------------------|---|------------------------------|
| 6975.000345 | task-clock | # | 1.166 CPUs utilized |
| 2,065 | context-switches | # | 0.296 K/sec |
| 126 | CPU-migrations | # | 0.018 K/sec |
| 14,348 | page-faults | # | 0.002 M/sec |
| 22,952,576,506 | cycles | # | 3.291 GHz |
| 7,035,973,150 | stalled-cycles-frontend | # | 30.65% frontend cycles idle |
| 8,778,857,971 | stalled-cycles-backend | # | 38.25% backend cycles idle |
| 35,420,228,726 | instructions | # | 1.54 insns per cycle |
| | | # | 0.25 stalled cycles per insn |
| 6,793,566,368 | branches | # | 973.988 M/sec |
| 285,888,040 | branch-misses | # | 4.21% of all branches |
| 5.981211788 | seconds time elapsed | | |

Nehalem 2.8GHz

=====

\$ perf stat <program>

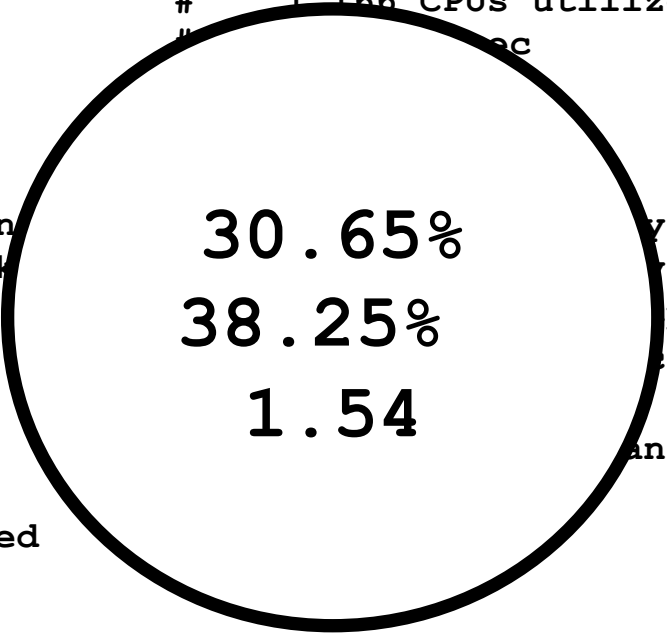
| | | |
|-----------------------------------|---|------------------------------|
| 6975.000345 task-clock | # | 1.166 CPUs utilized |
| 2,065 context-switches | # | 0.296 K/sec |
| 126 CPU-migrations | # | 0.018 K/sec |
| 14,348 page-faults | # | 0.002 M/sec |
| 22,952,576,506 cycles | # | 3.291 GHz |
| 7,111,111 stalled-cycles-frontend | # | 30.65% frontend cycles idle |
| 7,111,111 stalled-cycles-backend | # | 38.25% backend cycles idle |
| 1.54 instructions | # | 1.54 insns per cycle |
| 0.25 stalled-cycles | # | 0.25 stalled cycles per insn |
| 973.988 M/sec | # | 973.988 M/sec |
| 4.21% of all branches | # | 4.21% of all branches |
| 5.981211788 | | time elapsed |

Nehalem 2.8GHz

=====

\$ perf stat <program>

```
6975.000345 task-clock                               # 1.166 CPUs utilized
      2,065 context-switches                          # 0.000 CPUs utilized
      126 CPU-migrations                             # 0.000 CPUs utilized
     14,348 page-faults                               # 0.000 CPUs utilized
22,952,576,506 cycles                                cycles
  7,035,973,150 stalled-cycles-frontend               cycles idle
  8,778,857,971 stalled-cycles-backend               cycles idle
35,420,228,726 instructions                          cycle
                                                    cycles per insn
  6,793,566,368 branches                               branches
    285,888,040 branch-misses
5.981211788 seconds time elapsed
```



30.65%

38.25%

1.54

Sandy Bridge 2.4GHz

=====

\$ perf stat <program>

| | | |
|---------------------------------------|---|------------------------------|
| 5888.817958 task-clock | # | 1.180 CPUs utilized |
| 2,091 context-switches | # | 0.355 K/sec |
| 211 CPU-migrations | # | 0.036 K/sec |
| 14,148 page-faults | # | 0.002 M/sec |
| 19,026,773,297 cycles | # | 3.231 GHz |
| 5,117,688,998 stalled-cycles-frontend | # | 26.90% frontend cycles idle |
| 4,006,936,100 stalled-cycles-backend | # | 21.06% backend cycles idle |
| 35,396,514,536 instructions | # | 1.86 insns per cycle |
| | # | 0.14 stalled cycles per insn |
| 6,793,131,675 branches | # | 1153.565 M/sec |
| 186,362,065 branch-misses | # | 2.74% of all branches |
| 4.988868680 seconds time elapsed | | |

Sandy Bridge 2.4GHz

=====

\$ perf stat <program>

| | | |
|------------------------------|---|------------------------------|
| 5888.817958 task-clock | # | 1.180 CPUs utilized |
| 2,091 context-switches | # | 0.355 K/sec |
| 211 CPU-migrations | # | 0.036 K/sec |
| 14,148 page-faults | # | 0.002 M/sec |
| 19,026,552,897 cycles | # | 3.231 GHz |
| 26.90% frontend cycles idle | # | 26.90% frontend cycles idle |
| 21.06% backend cycles idle | # | 21.06% backend cycles idle |
| 1.86 insns per cycle | # | 1.86 insns per cycle |
| 0.14 stalled cycles per insn | # | 0.14 stalled cycles per insn |
| 1153.565 M/sec | # | 1153.565 M/sec |
| 2.74% of all branches | # | 2.74% of all branches |
| 4.988868680 | | |
| time elapsed | | |

Sandy Bridge 2.4GHz

=====

\$ perf stat <program>

| | | | | |
|----------------|-------------------------|---|---|-------------------|
| 5888.817958 | task-clock | # | 1 | 180 CPUs utilized |
| 2,091 | context-switches | # | | sec |
| 211 | CPU-migrations | | | |
| 14,148 | page-faults | | | |
| 19,026,773,297 | cycles | | | |
| 5,117,688,998 | stalled-cycles-frontend | | | cycles idle |
| 4,006,936,100 | stalled-cycles-backend | | | cycles idle |
| 35,396,514,536 | instructions | | | cycle |
| | | | | les per insn |
| 6,793,131,675 | branches | | | |
| 186,362,065 | branch-misses | | | anches |
| 4.988868680 | seconds time elapsed | | | |



8

“Memory is random access”



MEMORY SYSTEMS

Cache, DRAM, Disk

MK[®]
MORGAN KAUFMANN

BRUCE JACOB • SPENCER W. NG • DAVID T. WANG



Temporal



Temporal

Spatial



Temporal

Spatial



Pattern

Latencies measured by SiSoftware

Intel i7-3960X (Sandy Bridge E)

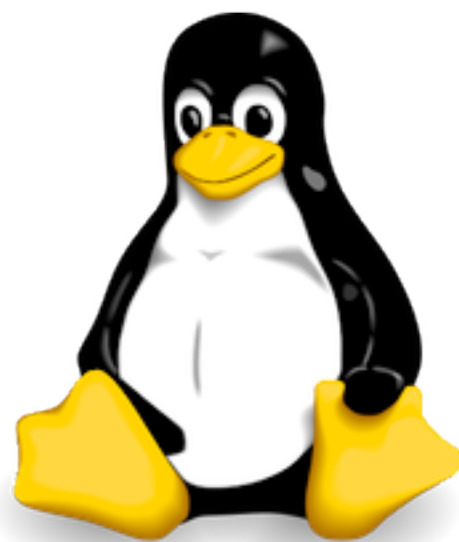
| | L1D | L2 | L3 | Memory |
|----------------|----------|-----------|-----------|---------|
| Sequential | 3 clocks | 11 clocks | 14 clocks | 6.0 ns |
| In-Page Random | 3 clocks | 11 clocks | 18 clocks | 22.0 ns |
| Full Random | 3 clocks | 11 clocks | 38 clocks | 65.8 ns |

**Memory Access Patterns
Really Matter!!!**



7

“Mac’s are a good development platform”







6

***“Garbage Collection takes away
the worry of memory
management”***

```
$ java -XX:+UnlockDiagnosticVMOptions \  
      -XX:+PrintFlagsFinal -version \  
      | wc -l
```

```
$ java -XX:+UnlockDiagnosticVMOptions \  
      -XX:+PrintFlagsFinal -version \  
      | wc -l
```

...

770

```
$ java -XX:+UnlockDiagnosticVMOptions \  
      -XX:+PrintFlagsFinal -version \  
      | wc -l
```

...

770

At least 272 are GC related!!!

G1 GC



Java™

G1GC





Native Heap – Beyond the city walls...





AZUL
S Y S T E M S

5

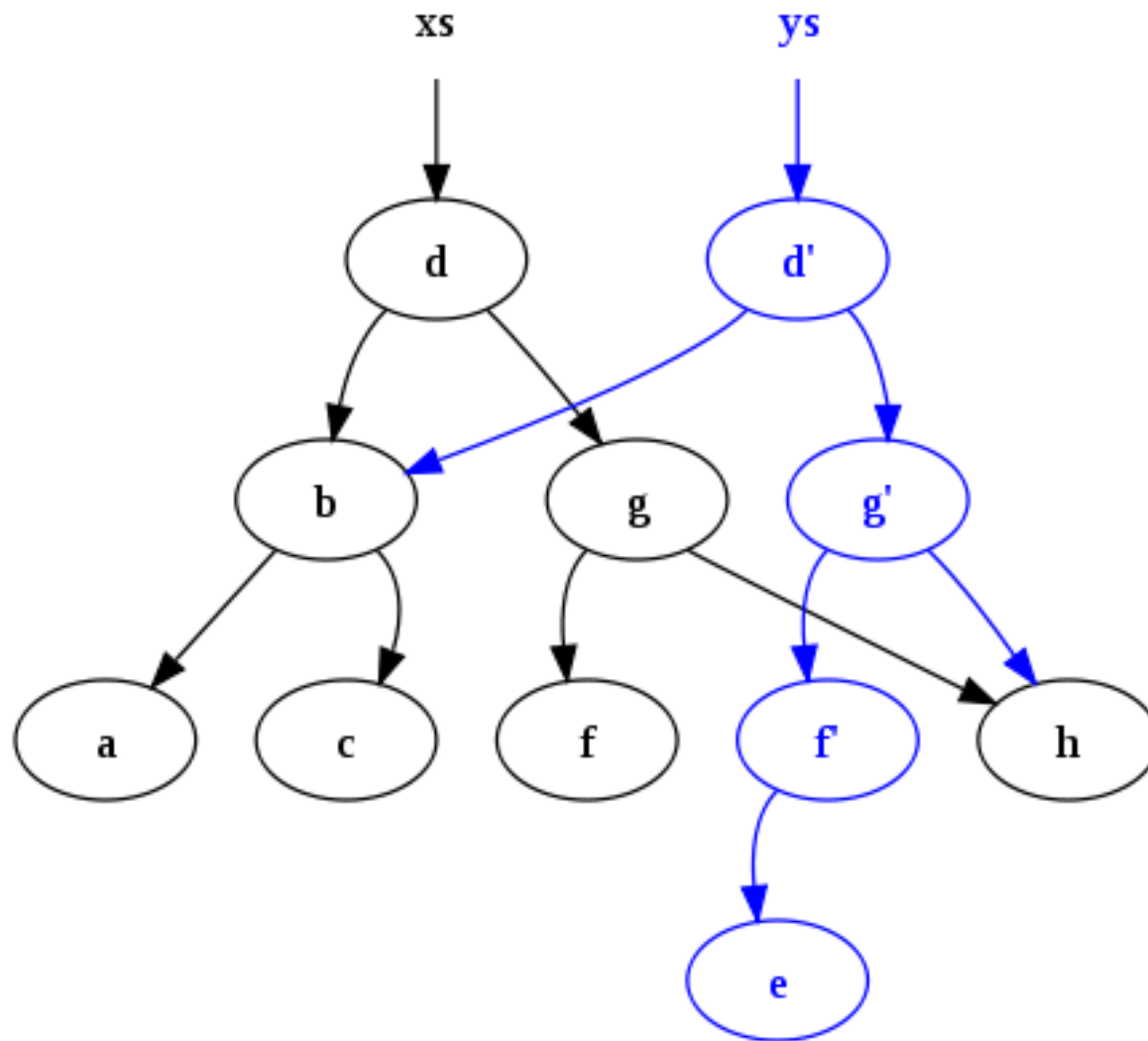
***“Functional Programming solves
the concurrency problem”***

“The runtime can optimize immutable values...”

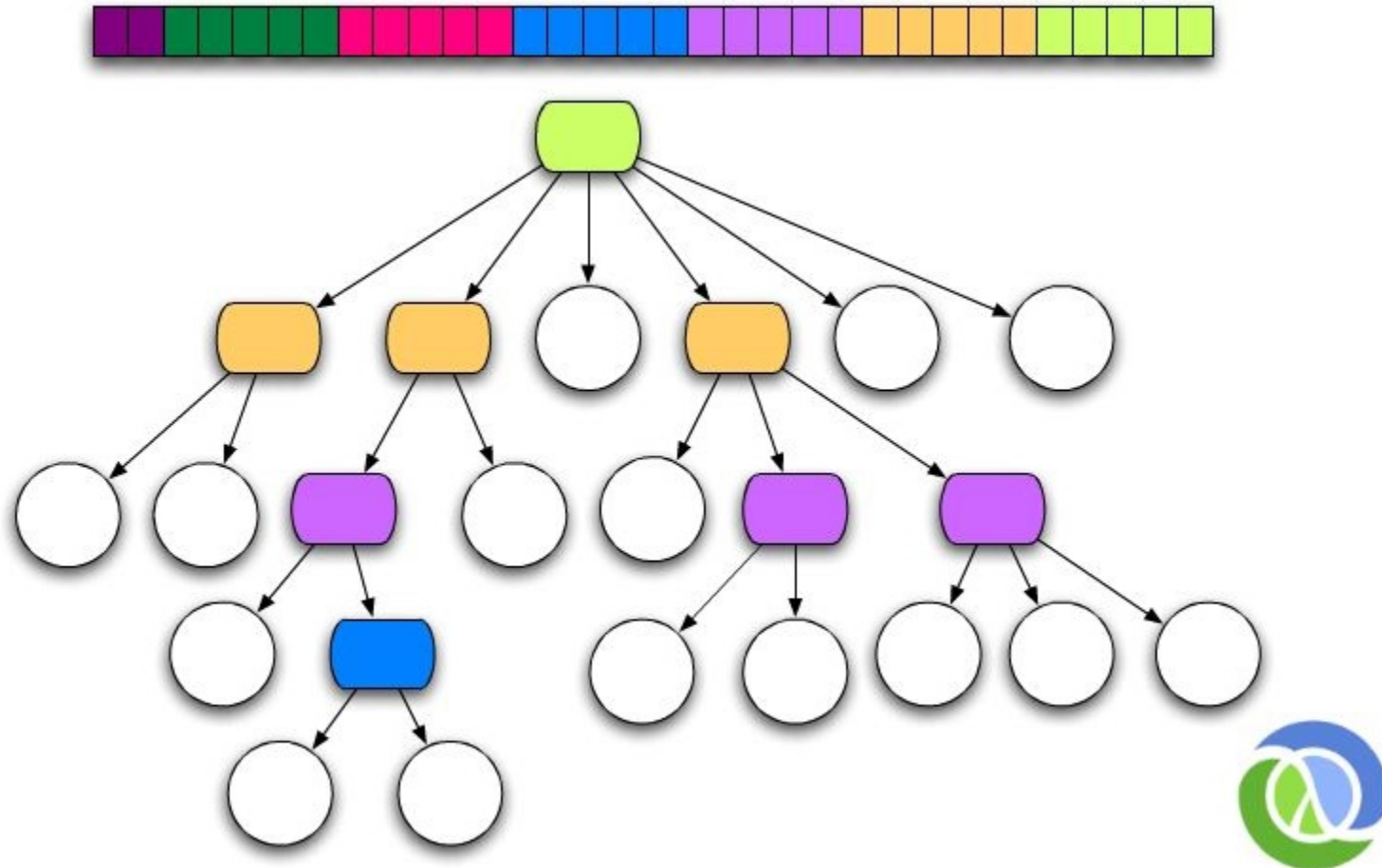
– FP Fanboi

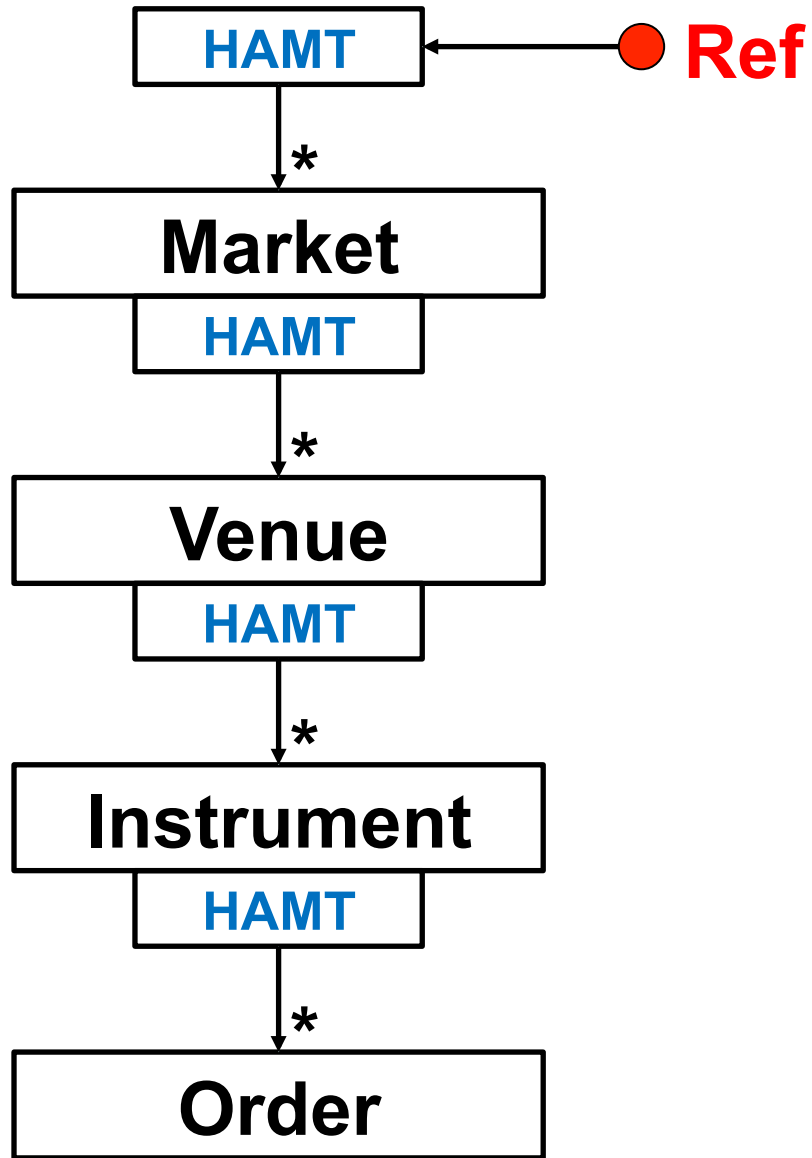


Persistent Data Structures

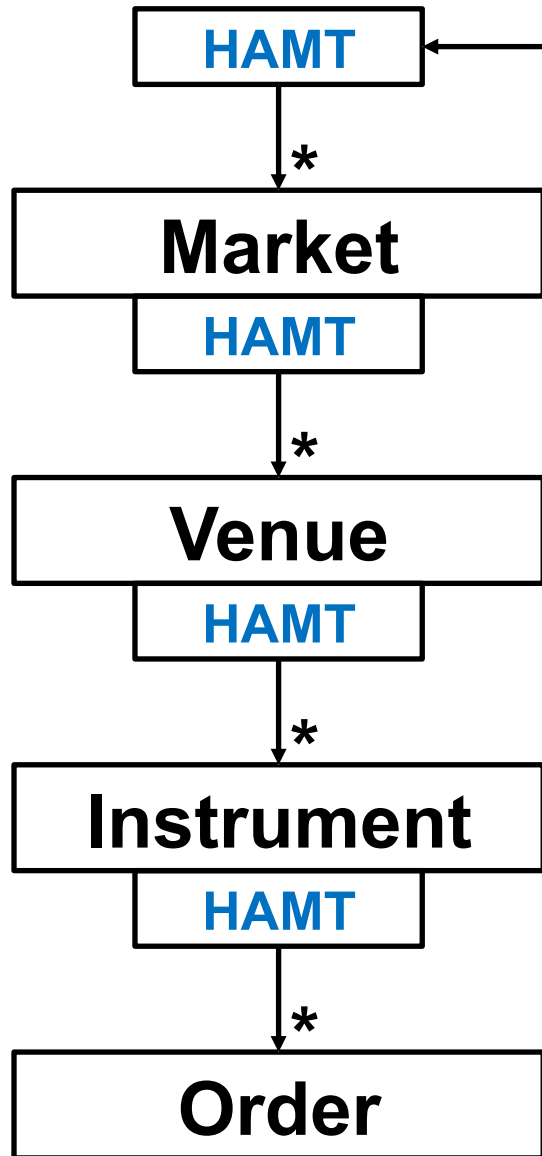


Hash Array Mapped Trie (Bagwell)



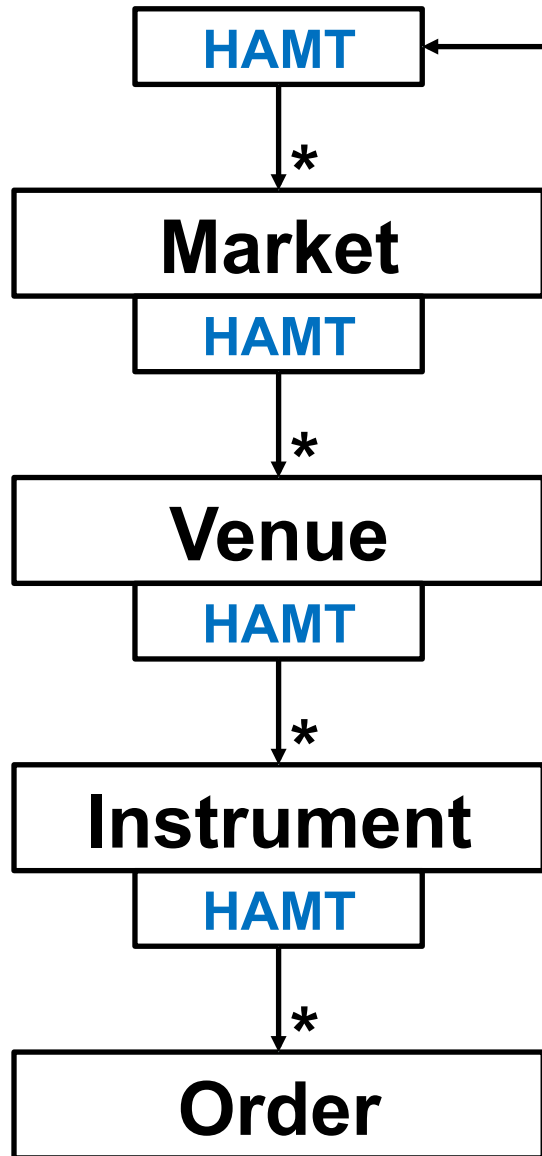


<< CAS Failure? >>

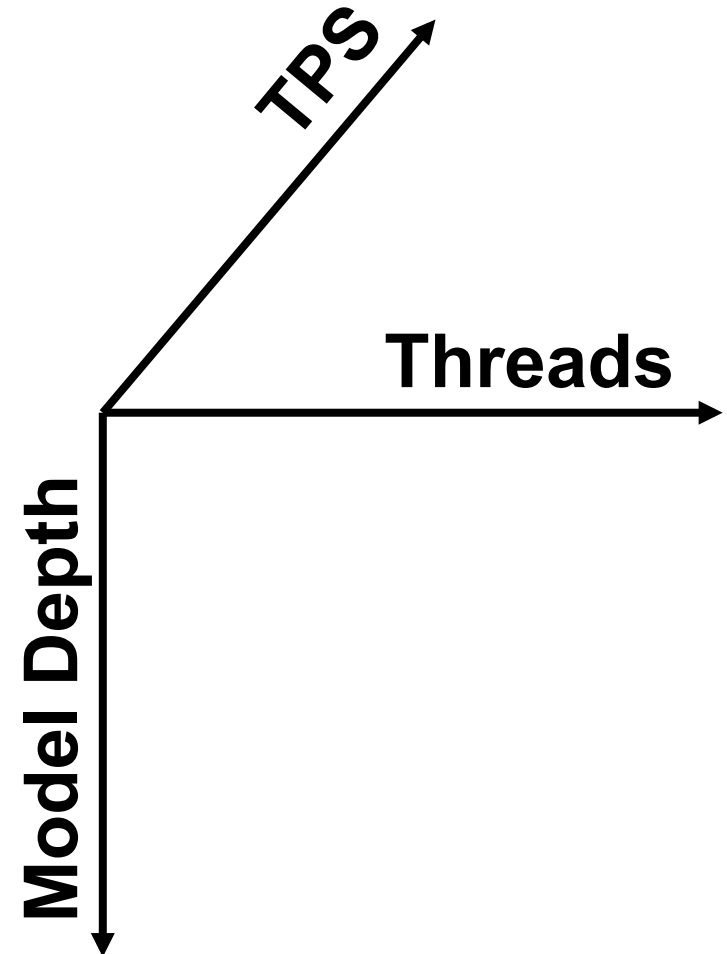


● Ref

<< CAS Failure? >>



● Ref





10 X

Throughput

10 X
Throughput

&&

**Garbage
Collection**



4

“Domain Models do not perform...”

“Object models do not perform...”

– Translation

“Indirection is bad...”

– Distillation





Cohesion

=>

Cache Friendly

Feature Envy
=>
Cache Misses

Clean Code

=>

Good Performance

Specialised Indices for access paths

Choose your data structures well

3

“Go parallel to scale...”



GOLDEN HAMMER

When you have a golden hammer, everything looks like a nail.



Locks

Work Pools

Map Reduce

Fork Join

Parallel Collections

Single Threaded Rocks!

... apply messaging passing ...

... then consider pipelining ...



**Go parallel only
when you really need too!**

2

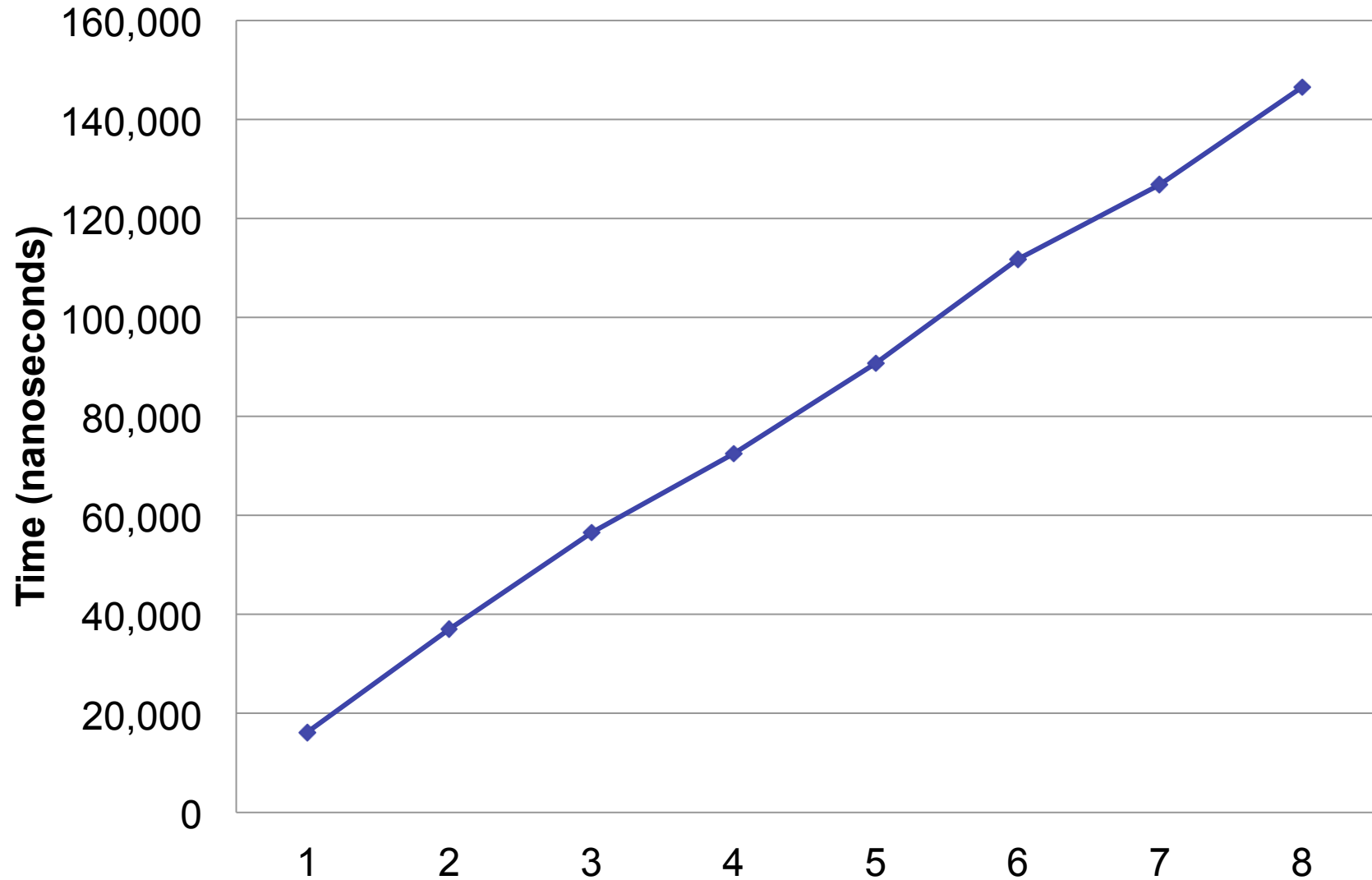
“Logging is cheap...”



Should a logger be able to generate data faster than a disk can write?

**Should more threads be able to
generate more data to write?**

Average Log Event Duration



Should logging be synchronous?

Should we really be logging Strings?

Should logging need guards?

How reliable should logging be?

1

***“Parsing code is highly
optimised...”***

SOAP

JSON

XML

FIX

HTTP

**How much text is not
UTF-8 or ASCII?**

**We need primitive support for
dealing with bytes and chars**

**What's horrible in the JDK when
parsing?**

“Where do I start?”



Performance Testing



Questions?

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

Twitter: @mjpt777

***“Any intelligent fool can make things bigger,
more complex, and more violent.***

***It takes a touch of genius, and a lot of courage,
to move in the opposite direction.”***

- Albert Einstein