Toward Low-Latency Java Applications

JavaOne 2014

by

John Davies | CTO

Kirk Pepperdine | CPC

Agenda / Notes

- Increasingly Java is being used to build applications that come with low-latency requirements.
  - To meet this latency requirements developers have to have a deeper understanding of the JVM and the hardware so their code works in harmony with it

- Recent trends in hard performance problems suggest the biggest challenge is dealing with memory pressure
  - Memory pressure

- This session demonstrates the memory cost of using XML parsers such as SAX and compares that with low-latency alternatives.

What is Latency

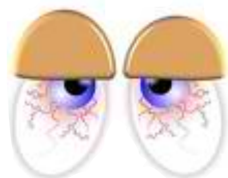
- The measure of time taken to respond to a stimulus

- Mix of active time and dead time
  - Active time is when a thread is making forward progress
  - Dead time is when a thread is stalled

Total Response Time = Service time + time waiting for service

What is Low Latency?

- Latency that is not noticeable by a human
  - Generally around 50ms
  - However missing video sync @ 16.7ms time intervals will cause eye fatigue



- Low latency for trading systems is faster than everyone that else
  - Generally a few ms or less
  - Generally the time taken to get through a network card

Why Do We Care About Latency

- There is no second place in anything that looks like an auction

- Less latency is perceived as better QoS
  - Customers or end users are less likely to abandon

Where is really matters!!!

- Front Office - The domain of High Frequency Trading (HFT)
  - Very high volume, from 50k-380k / sec
  - This is per exchange!

  - Latency over 10μS is considered slow
  - 10μS is just 3km in speed of light time!

  - Fix is a good standard but binary formats like ITCH, OUCH & OMNet are often better suited

  - Much of the data doesn’t even hit the processor. FPGA (Field-Programmable Gate Arrays), “smart network cards” do a lot of the work

Why it matters

- A world where 1ms is estimated to be worth over $100m
  - For that sort of money you program in whatever they want!
  - People who work here are almost super-human, a few make it big but most don't make it at all
- There is little place for Java and VM languages here, we need to move down the stack a little
  - We're not going to go here today, it's a world of customized hardware, specialist firmware, assembler and C

Sources of Latency

- Some you can rid of, some you can't
  - speed of light
  - hardware sharing (schedulers)
  - JVM safe-pointing
  - Application
- All hardware works in blocks of data
  - CPU: word size, cache line size, internal buses
  - OS: pages
  - Network: MTU
  - Disk: sector
- If your data fits into a block things will work well

Sources of Latency (JVM)?

- Safe-pointing
  - Called for when the JVM has to perform some maintenance
  - Parks application threads when the are in a safe harbor
  - State and hence calculation they are performing will not be corrupted

- Safe-pointing is called for;
  - Garbage Collection
  - Lock deflation
  - Code cache maintenance
  - HotSpot (de-)optimization
  - …..

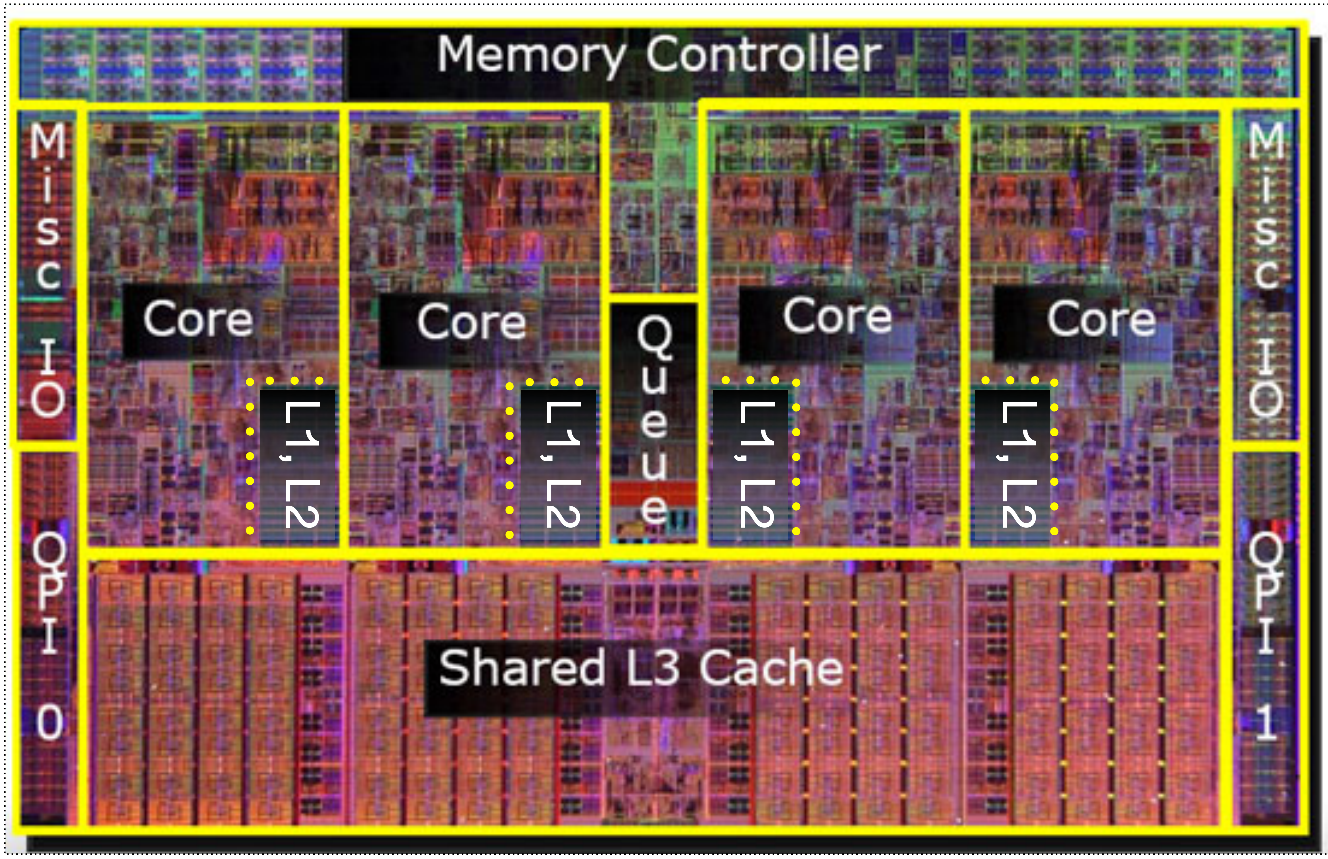
Puzzler

```
public void increment() {
    synchronized( this) {
        i++;
    }
}
```

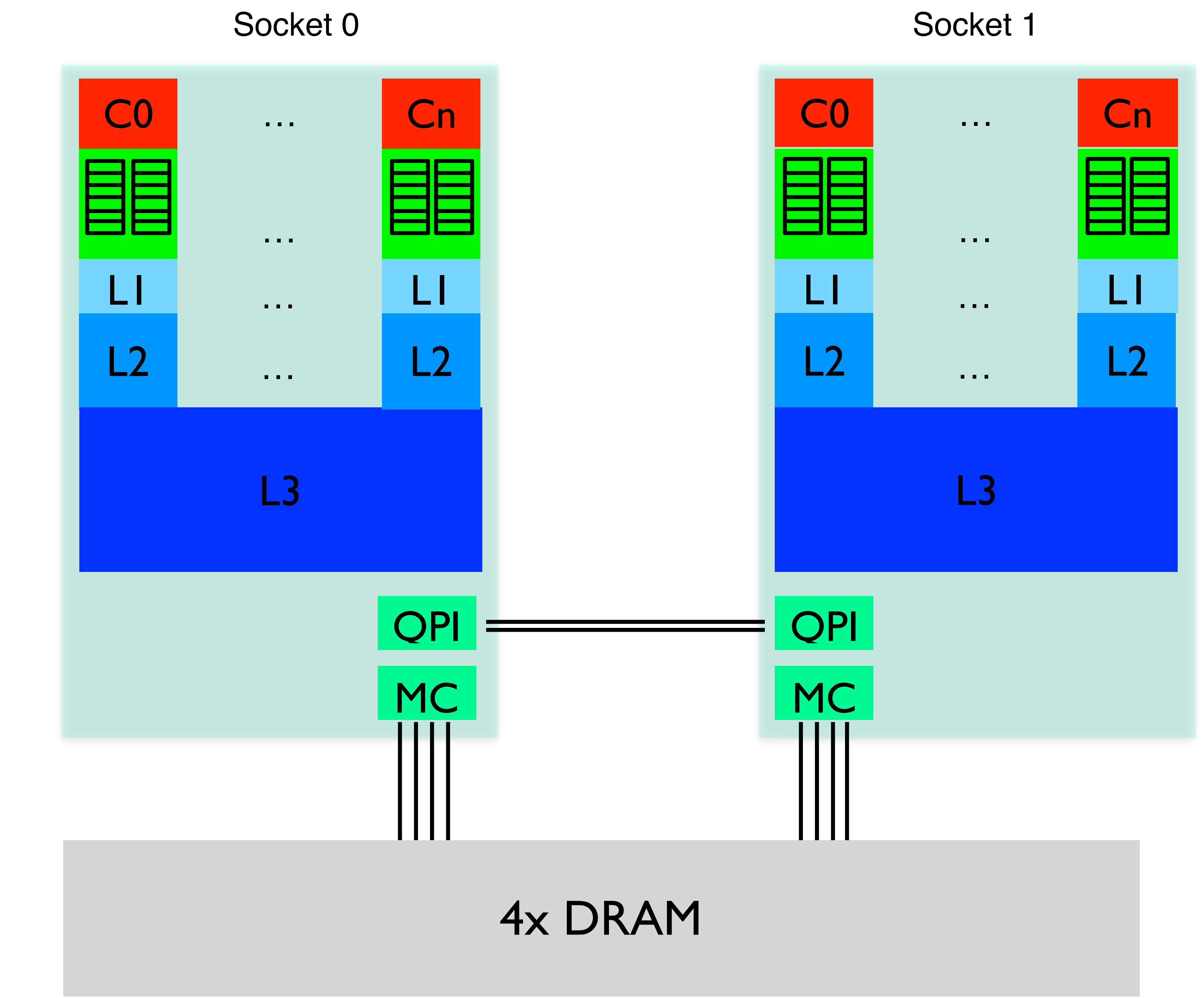
```
public synchronized void increment() {
        i++;
}
```

- Which is faster and why?

Hardware


Memory Controller
Misc IO
Core
Core
Queue
Core
Core
Misc IO
L1, L2
L1, L2
L1, L2
L1, L2
QPI 0
Shared L3 Cache
QPI 1

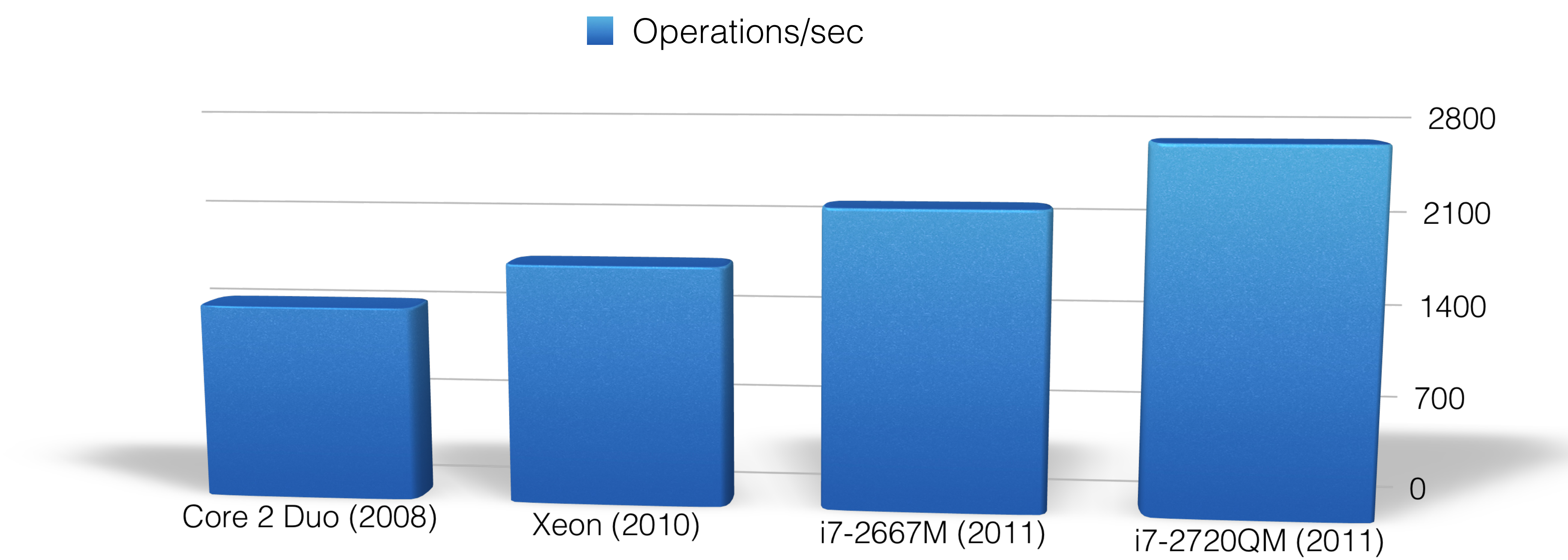
CPU



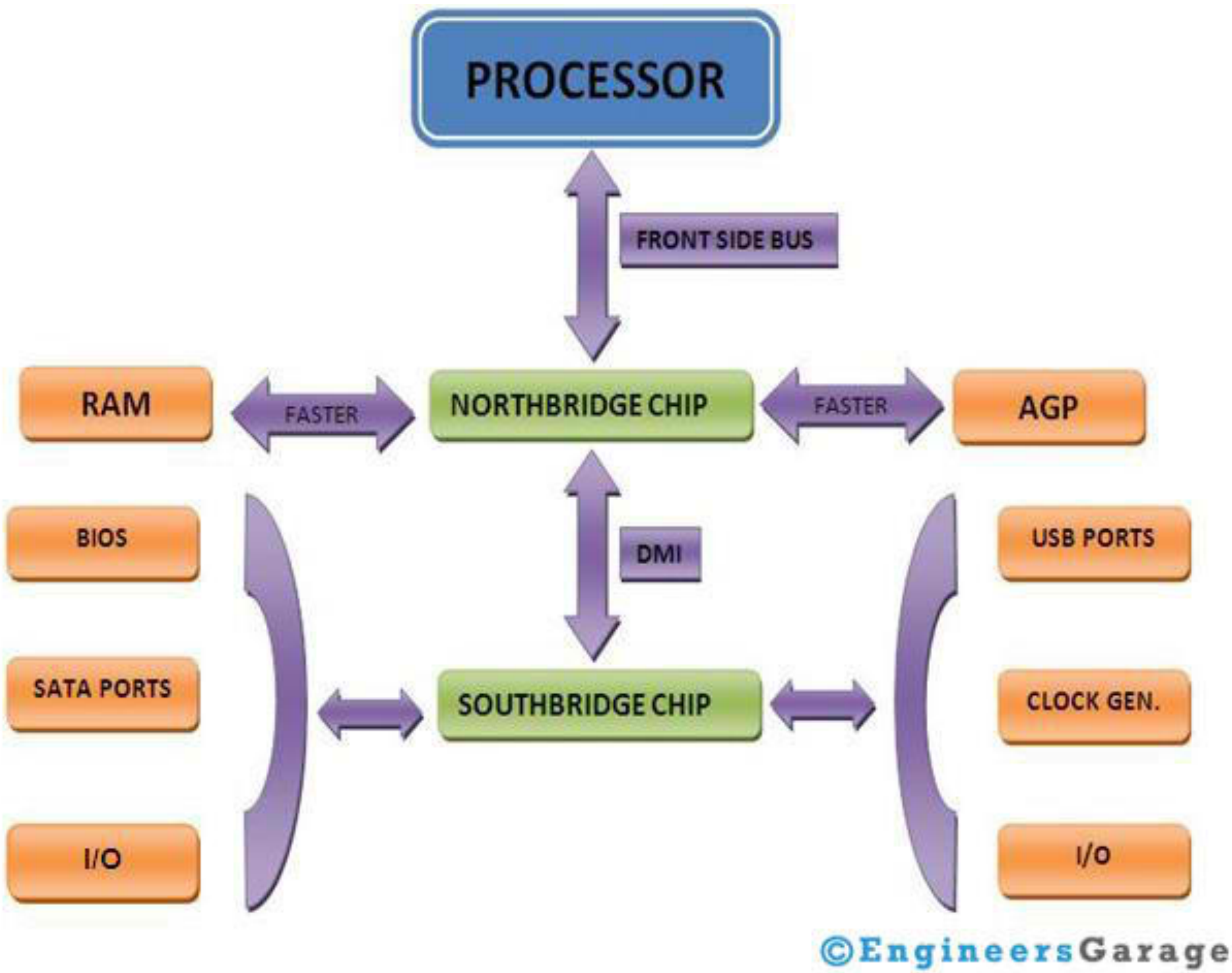
Moore's Law

- "The Free Lunch is Over" - Herb Sutter
  - Or is it?

- Martin Thompson's "Alice in Wonderland" text parsing



Hardware (bigger picture)

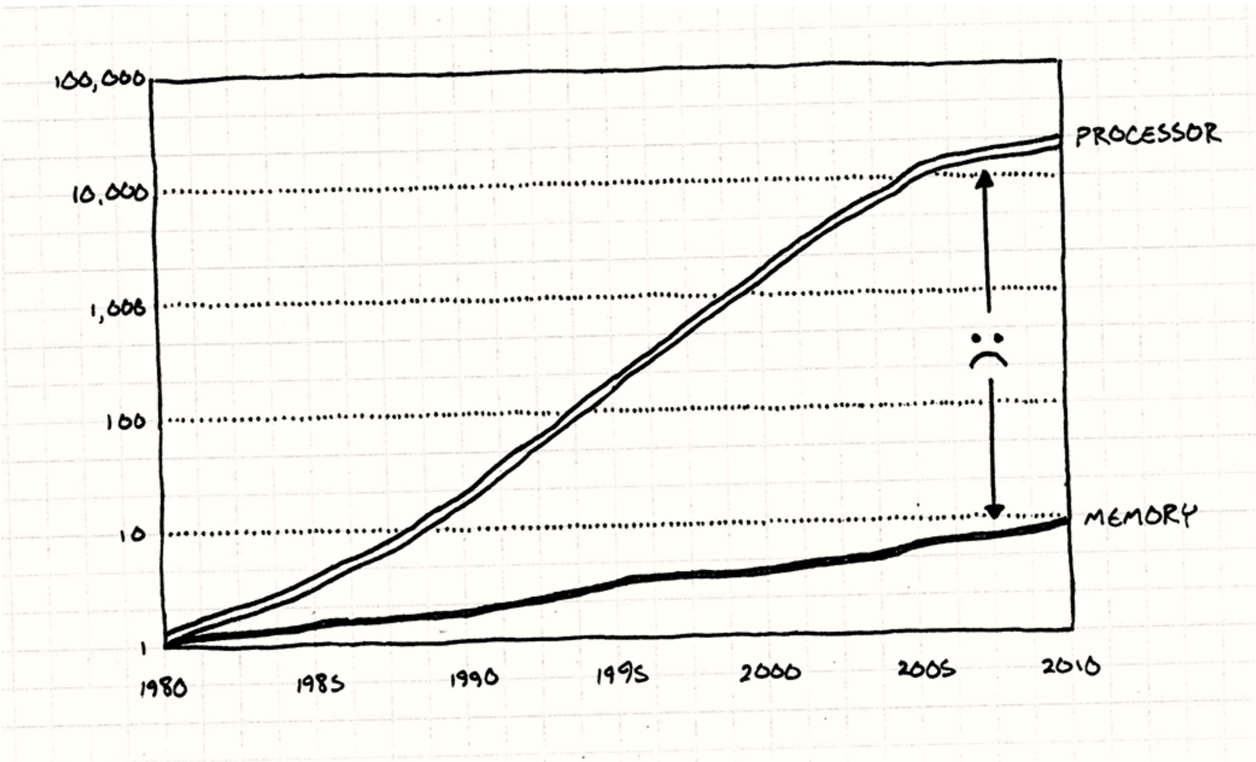


Time to Access Data

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM)	120 ns	6 min
Solid-state disk I/O (flash memory)	50-150 μs	2-6 days
Rotational Disk I/0	1-10 ms	1-12 months
Network SF to NY	40 ms	4 years
Network SF to London	81 ms	8 years
Network SF to Oz	183 ms	19 years
TCP packet retransmit	1-3 s	105-317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millenium
Hardware virtualization system reboot	40 s	4 millenium
Physical system reboot	5 m	32 millenium

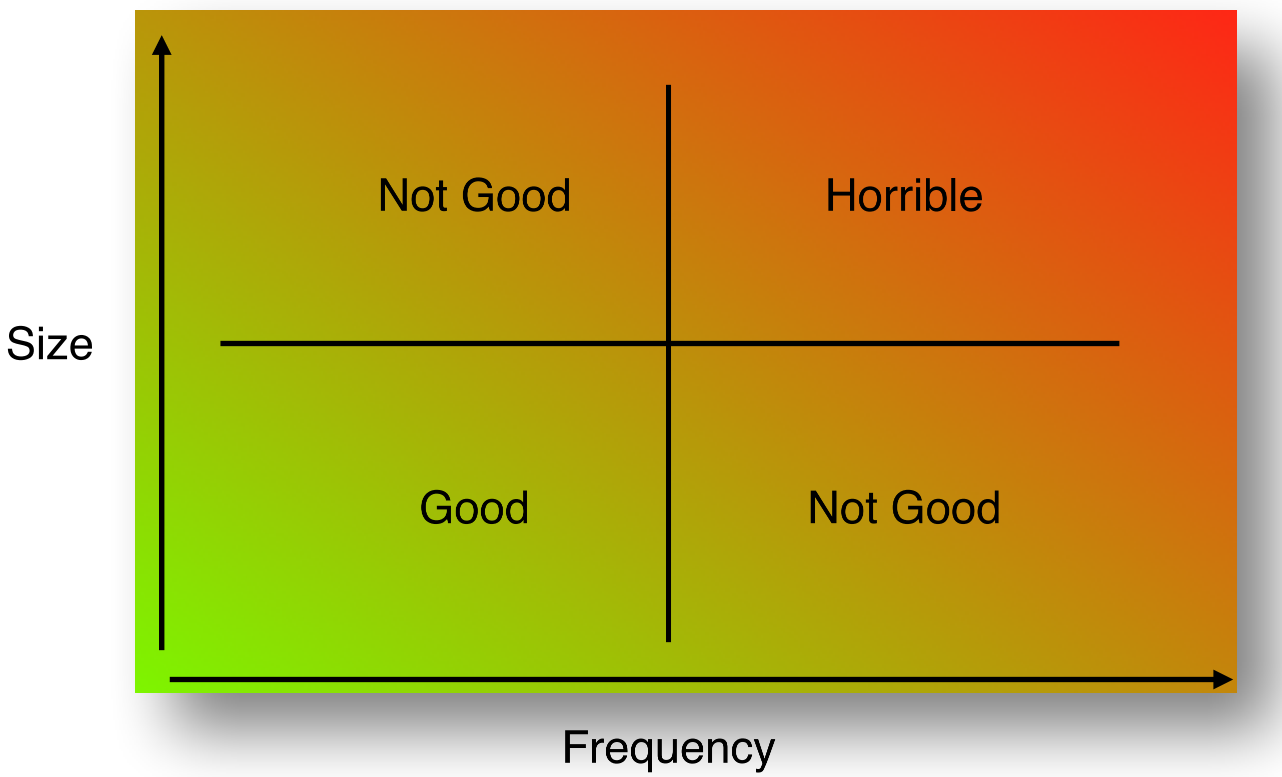
Memory Pressure

- Predictability helps the CPU remain busy
- Java heap is quite often not predictable
  - idles the CPU (micro-stall)

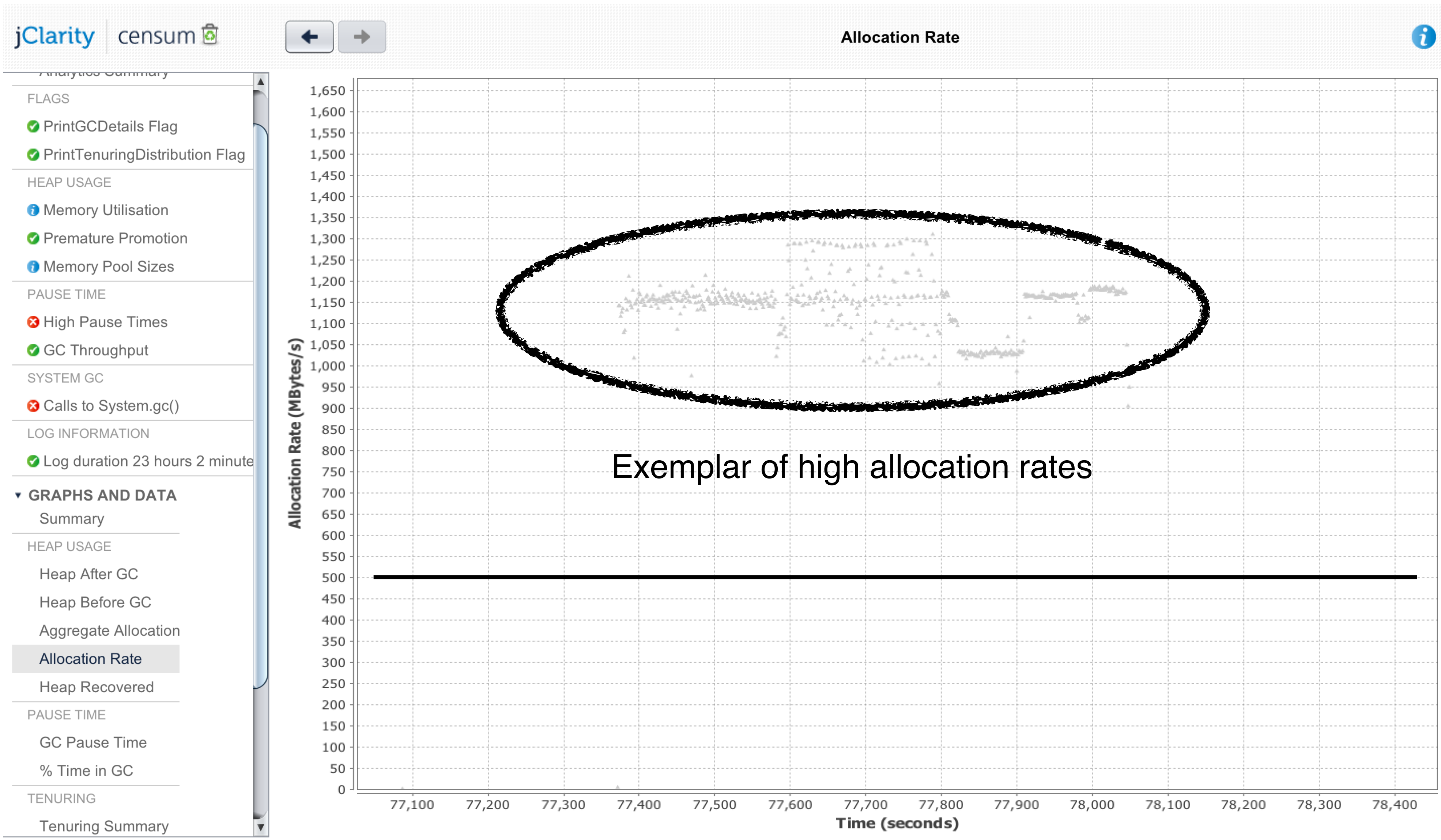


Memory Pressure

- Rate at which the application churns through memory



Allocation Rates Before



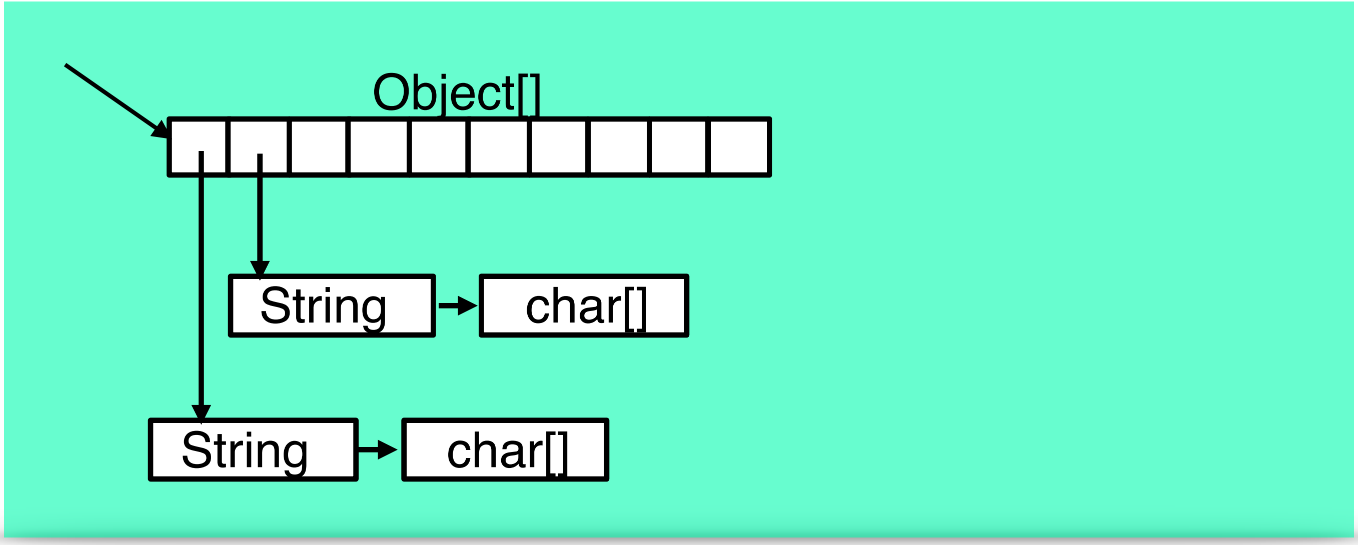
Memory Layout

- Proper memory layouts promotion dead reckoning
  - Single fetch to the data
  - Single calculation to the next data point
  - Processors turn on pre-fetching

- Java Objects form an undisciplined graph
  - OOP is pointer to the data
  - A field is an OOP
    - Two hops to the data
    - Most likely cannot dead-reckon to the next value
      - Think iterator over a collection

  - An array of objects is an array of pointers
    - (at least) two hops to the data

Object Layouts



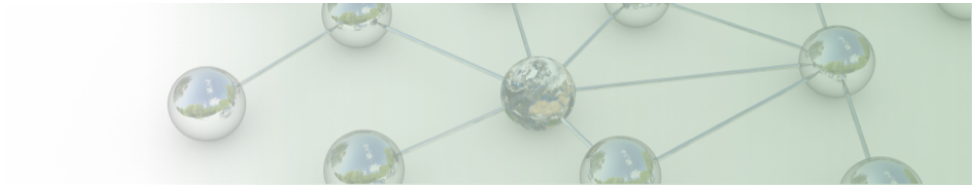
Java Memory Layout

- Solution: we need more control over how the JVM lays out memory
- Risk: if we have more control it's likely we'll shoot ourselves in the foot
- One answer: StructuredArray (Gil Tene and Martin Thompson)

What is the problem?

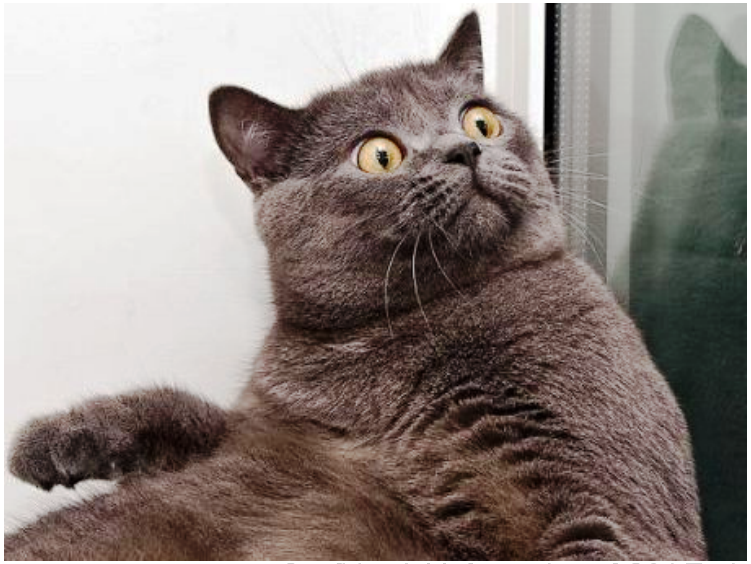
- SDO is a binary codec for XML documents
  - reduces 7k documents to just under 400 bytes
- Requirement: improve tx to 1,000,000/sec/core
  - baseline: 200,000 tx/sec/core
- Problem: allocation rate of 1.2GB/sec
- Action: identify loci of object creation and altered application to break it up
- Result: eliminated ALL object creation. Improved tx rate to 5,000,000/sec/core

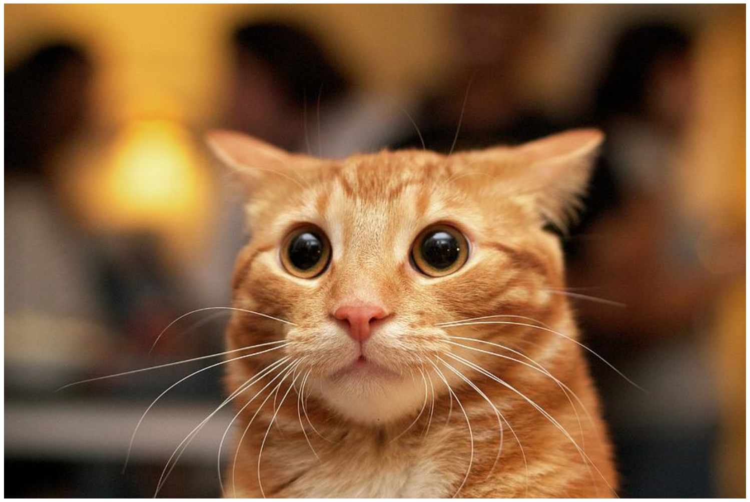
We're good!!!!!



2,500%

Improvement





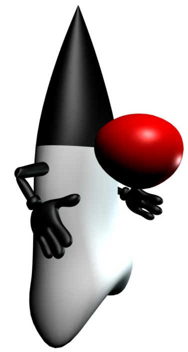
Memory Footprint of SDO

- SDOs were designed for two main purposes
  - Reduce memory footprint - by storing data as byte[] rather than fat Objects
  - Increase performance over "classic" Java Objects
- Java is in many cases worse than XML for bloating memory usage for data
  - A simple "ABC" String takes 48 bytes!!!
- We re-wrote an open source Java Binding tool to create a binary codec for XML (and other) models
- We can reduce complex XML from 8k (an FpML derivative trade) and 25k as "classic" bound Java to under 400 bytes
  - Well over 50 times better memory usage!

Same API, just binary

- Classic getter and setter vs. binary implementation
- Identical API

```java
@Override
public Date getTradeDate() {
    return tradeDate;
}

@Override
public void setTradeDate(Date tradeDate) {
    this.tradeDate = tradeDate;
}
```



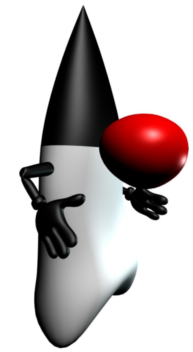
```java
@Override
public Date getTradeDate() {
    long date = wordFromBytesFromOffset(8);
    date *= 86_400_000L; // milliseconds in a day
    return new Date(date);
}

@Override
public void setTradeDate(Date tradeDate) {
    long date = tradeDate.getTime();
    date /= 86_400_000L; // milliseconds in a day

    data[8] = (byte)(date >>> 8);
    data[9] = (byte)(date);
}
```

Just an example…

```java
@Override
public Date getTradeDate() {
    long date = wordFromBytesFromOffset(8);
    date *= 86_400_000L; // milliseconds in a day
    return new Date(date);
}

@Override
public void setTradeDate(Date tradeDate) {
    long date = tradeDate.getTime();
    date /= 86_400_000L; // milliseconds in a day

    data[8] = (byte)(date >>> 8);
    data[9] = (byte)(date);
}
```

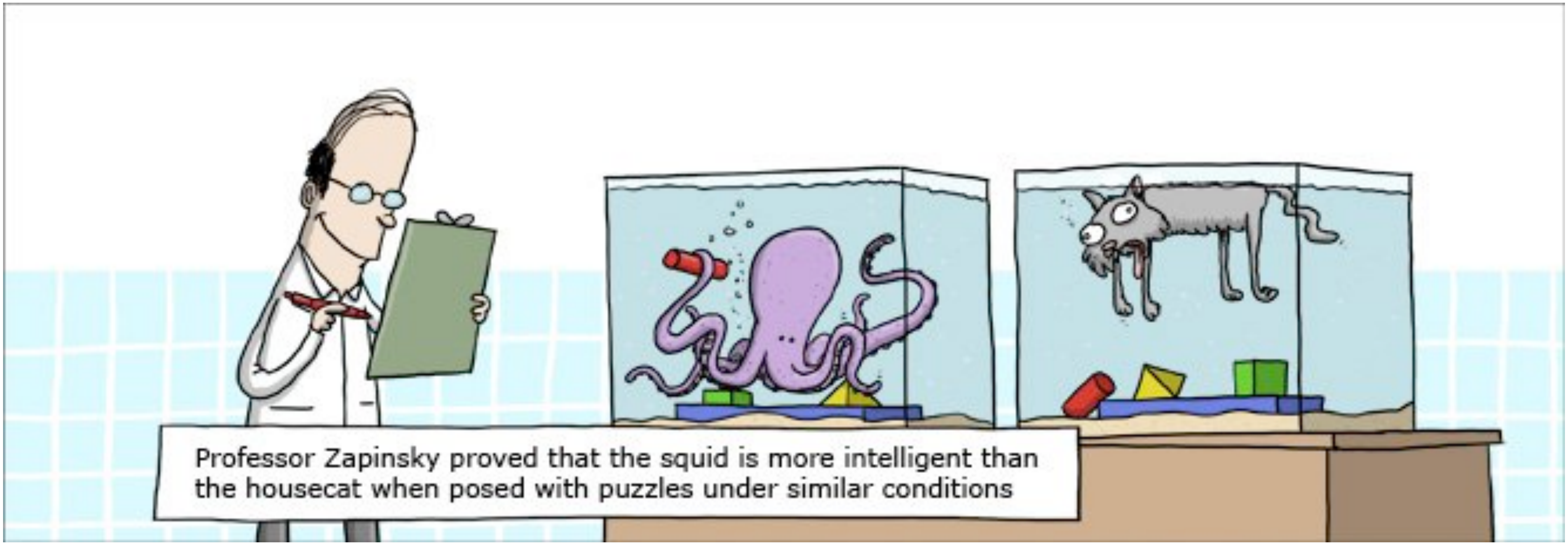
Did I mention … The Same API

- This is a key point, we're changing the implementation not the API
- This means that Spring, in-memory caches and other tools work exactly as they did before

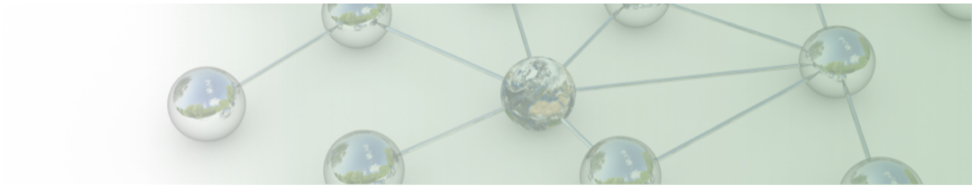


```java
@Override
public Date getTradeDate() {
    long date = wordFromBytesFromOffset(8);
    date *= 86_400_000L; // milliseconds in a day
    return new Date(date);
}

@Override
public void setTradeDate(Date tradeDate) {
    long date = tradeDate.getTime();
    date /= 86_400_000L; // milliseconds in a day

    data[8] = (byte)(date >>> 8);
    data[9] = (byte)(date);
}
```

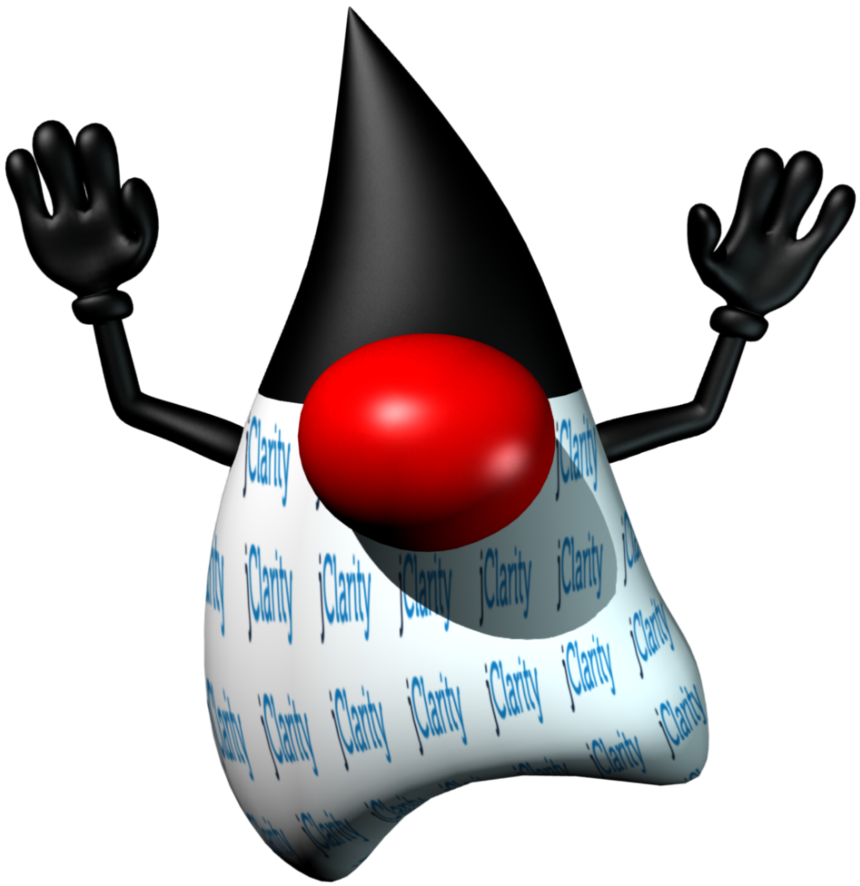
Demo



- Professor Zapinsky proved that the squid is more intelligent than the housecoat when posed this puzzles under similar conditions



Questions?



For more information please contact Kirk Pepperdine (@kcpeppe)
or John Davies (@jtdavies)

Code & more papers will be posted at http://sdo.c24.biz