



# The War on Latency

*Reducing Dead Time*  
*Kirk Pepperdine*  
*Principle*  
*Kodewerk Ltd.*

# Me

- *Work as a performance tuning freelancer*
- *Nominated Sun Java Champion*
- *[www.kodewerk.com](http://www.kodewerk.com)*
- *[kirk.blog-city.com](http://kirk.blog-city.com)*
- *[www.javaperformancetuning.com](http://www.javaperformancetuning.com)*
- *Other stuff (google if you care to)*



Java Performance Tuning  
Chania Crete  
May 18-21



## Public Service Announcement

*The resemblance of any opinion,  
recommendation or comment made  
during this presentation to performance  
tuning advice is merely coincidental*

## Latency Affects Abandonment

- *Shopzilla, 5 second improvement resulted in*
  - *25% increase in page view*
  - *10% increase in revenue*
  - *50% reduction in hardware*
- *Amazon reports every 100ms costs 1% in sales*

## Defining Latency

- *Time that elapses between a stimulus and the response to it*
  - *data latency (end user response time)*
  - *i/o latency (disk and network)*
  - *cache latency*
  - *synchronization*
- *Goal: find and minimize latency*

goal is to find and eliminate dead time or time spent waiting for something to happen

## The Box

- *Conceptional model of a system*
- *Visualize components of the system*
- *Visualize interactions between components*
- *Understand how each layer contributes latency*

### Actors

Usage patterns

### Application

Locks, external systems

### JVM/OS

Memory, Hardware management

### Hardware

CPU, Memory, Disk IO, Network

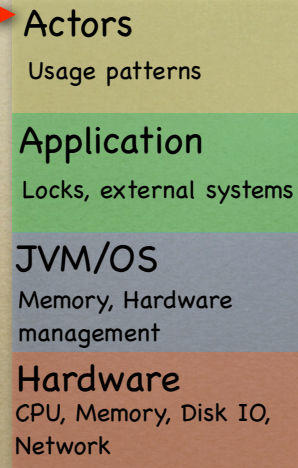
when components are good citizens, we'll experience good performance

when component are not good citizens, we'll experience poor performance

Look at monitoring data and ask, what does it mean in the box  
use that information to help guide our search for latency

# Latency and The Box

- *Defined by Usage Patterns*
- *drives load on the system*
- *Data latency shows up here*
- *response time*
- *Key measure of system performance*

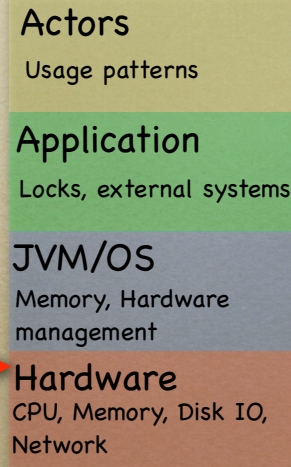


All performance decisions are guided by the user experience starting trigger and ending condition



# Latency and The Box

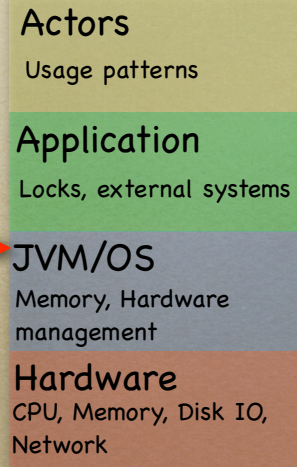
- *Bundle of non-sharable resources*
- *Defines finite capacity of the system*
  - *compute speeds*
  - *data capacities*
  - *data transfer speeds*



We can't go faster than our hardware  
nonsharable = Queuing  
Everything else will prevent us from going fas

# Latency and The Box

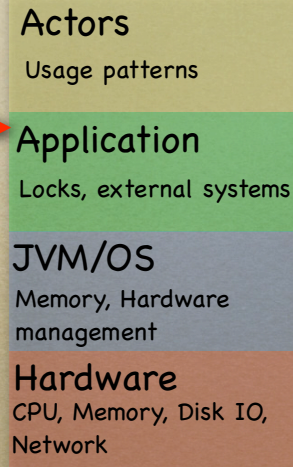
- *OS*
  - *hardware management and provisioning*
- *JVM*
  - *transform instructions into machine code*
  - *memory management*



memory management is the important item  
thread scheduling, interrupt handling, interacting with devices

# Latency and The Box

- *Translates user intent into a sequence of instructions*
- *Protects non-sharable soft resources*
  - *lock induced latency*
- *Interacts with external systems*



All performance decisions are guided by the user experience  
External systems may show up as a kernel problem or as parked threads  
thread pools as this level

# Finding Latency

- *Trigger*
  - *actors experience poor response time*
- *Action*
  - *find the dominating consumer of the CPU*

## Actors

Usage patterns

## Application

Locks, external systems

## JVM/OS

Memory, Hardware management

## Hardware

CPU, Memory, Disk IO, Network

All performance decisions are guided by the user experience

# Dominating Consumer

- *Application*
- *JVM*
- *OS*
- *No dominating consumer*
- *Monitor cpu (both user and system) and GC activity*

## Actors

Usage patterns

## Application

Locks, external systems

## JVM/OS

Memory, Hardware management

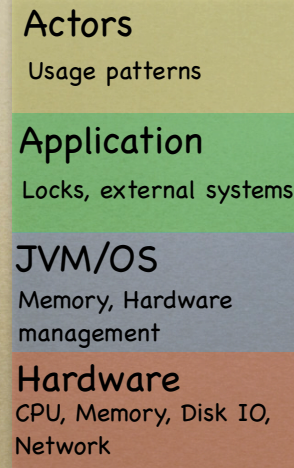
## Hardware

CPU, Memory, Disk IO, Network

All performance decisions are guided by the user experience

# Applicaton as Dominator

- *CPU user time is high*
- *Efficient Java memory management*
- *Object creation rates are reasonable*



1.2G/sec on this machine

# Localizing Latency

- *JVM dominates when*
  - *GC throughput is low*
    - *less than 90%*
  - *high full to partial GC ratio*
  - *object creation rates are high*

## Actors

Usage patterns

## Application

Locks, external systems

## JVM/OS

Memory, Hardware management

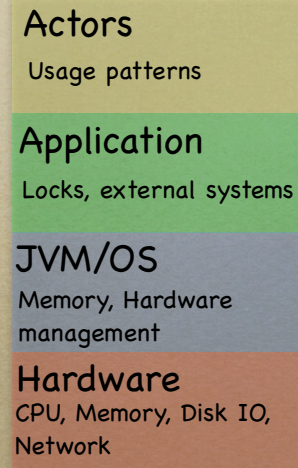
## Hardware

CPU, Memory, Disk IO, Network

1.2 gigs is about all this machine will tolerate

# Localizing Latency

- *OS dominates when system cpu*
- *exceeds 10%*
- *is 50% or greater than that of user cpu time*



1.2 gigs is about all this machine will tolerate



# Localizing Latency

- *No dominating consumer means threads are parked waiting for something*
- *calls to external systems*
- *locks*
- *thread pool starvation*

## Actors

Usage patterns

## Application

Locks, external systems

## JVM/OS

Memory, Hardware management

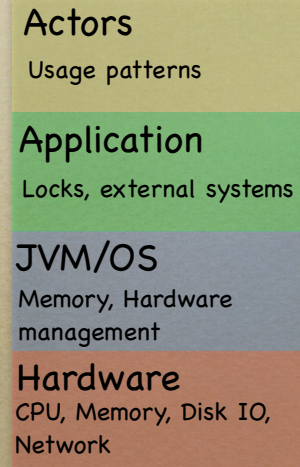
## Hardware

CPU, Memory, Disk IO, Network

1.2 gigs is about all this machine will tolerate

# Diagnosing Latency

- *Application - execution profile*
- *JVM*
  - *gc tuning*
  - *memory profiling*
- *OS - thread dumps and/or execution profiling*



1.2 gigs is about all this machine will tolerate

# Diagnosing Latency

- *No dominating consumer*
- *what is keeping threads out of the CPU?*

## Actors

Usage patterns

## Application

Locks, external systems

## JVM/OS

Memory, Hardware management

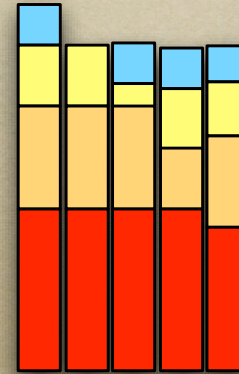
## Hardware

CPU, Memory, Disk IO, Network

debuggable question

## Big Gains First

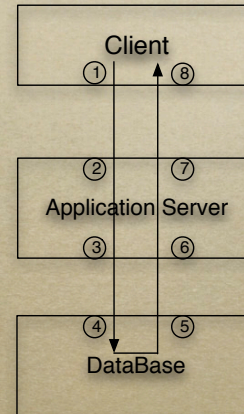
- *How can we remove 100ms from 500ms time budget*
- *100ms servlet*
- *150ms business logic*
- *250ms EJB*
- *500ms DB*



focus on layer with largest contribution

# Time Budgets

- *Build a layer by layer, component by component time budget*
- *5-4 DB response time*
- *6-3 Apps view of DB response time*
- *etc.....*



dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

## Common Sources of Latency

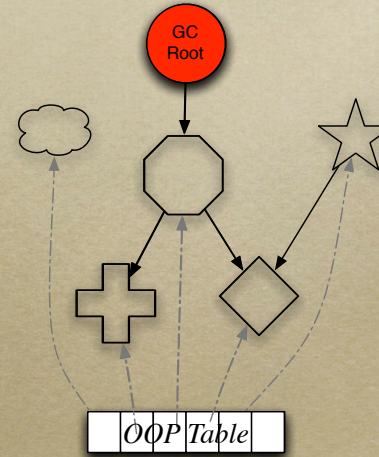
- *Java Memory Management*
- *Network I/O (JDBC)*
- *Disk I/O (Logging)*
- *Shared data structures*

# Java Memory Management

- *Java heap allocated out of C heap*
  - *one large contiguous piece of RAM*
- *Objects are allocated out of Java heap*
- *Java heap fills up triggering a garbage collection cycle*
  - *mark and sweep*

# Mark & Sweep GC

- *Traverse OOP table*  
*clear mark bit in each*  
*object*

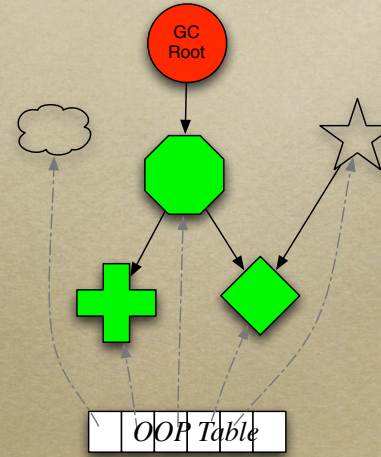


compaction?



# Mark & Sweep GC

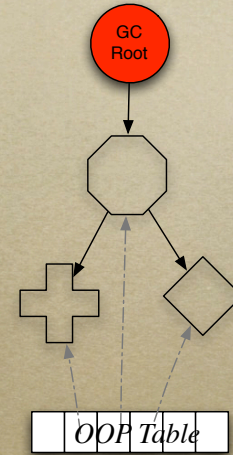
- *From GC root mark all reachable objects*



compaction?

# Mark & Sweep GC

- *Traverse OOP table releasing all unmarked objects.*

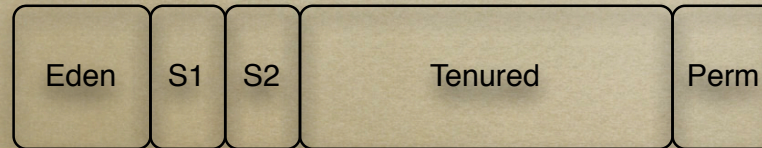


compaction?

## GC Optimizations

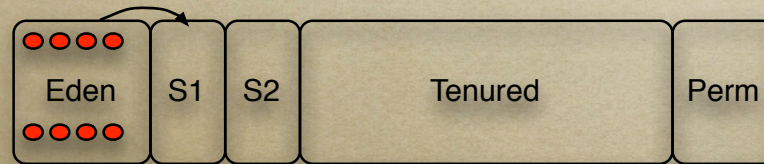
- *Parallel GC (throughput)*
- *Concurrent GC (pause time)*
- *Incremental*
- *Weak generational hypothesis*
  - *generational GC*
  - *G1GC*

# Generation Spaces



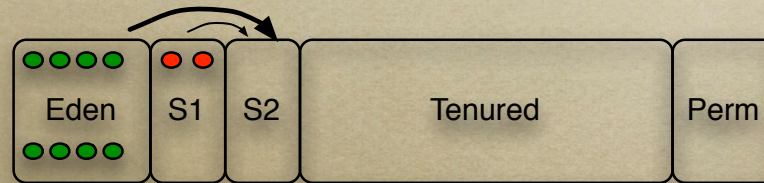
dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

# Generational Spaces



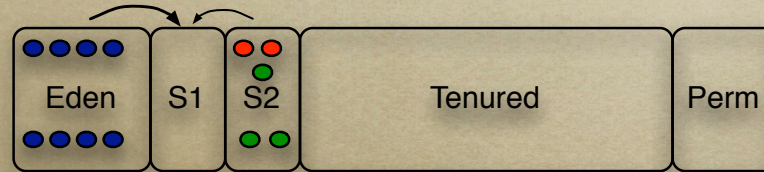
dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

# Generational Spaces



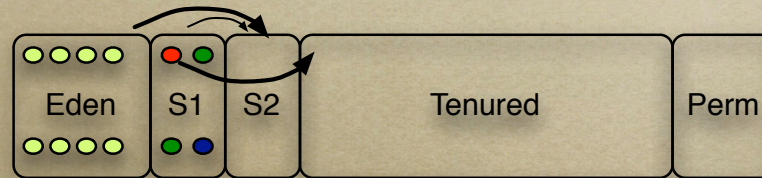
dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

# Generational Spaces



dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

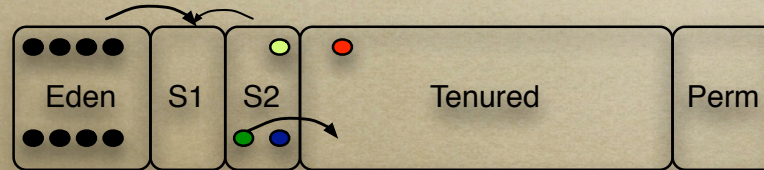
# Generational Spaces



dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

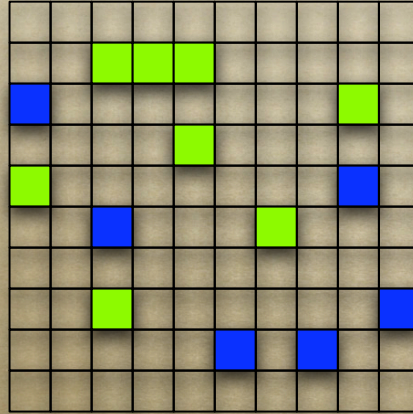


# Generation Spaces



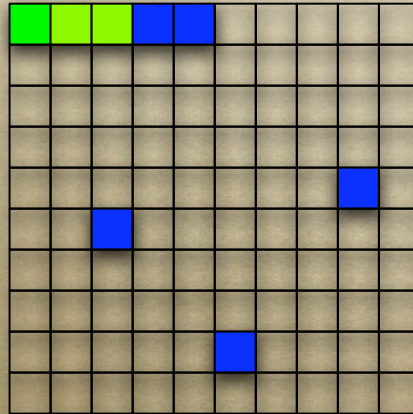
dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

G1GC



dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

G1GC



dominating consumer tells us the nature of the problem  
time budgets tell us where the problem is

## Talking Points

- *Young generational guarantee*
- *Fragmentation*
  - *compaction phase*
- *Sizing to avoid disruptive pauses*
  - *pause time goals*
  - *throughput goals*

## Talking Points

- *Space efficiency*
  - *zombies*
- *Completeness*
  - *floating garbage*
- *Object nepotism*
  - *tenured garbage*

## Bad Stuff

- *Unintentional object retention*
  - *Object with no semantic meaning to the application is never released*
- *Loitering objects*
  - *objects that will go away long after you want them to*
- *Local caches*

## Things That Help

- *Narrow scope of all variables*
  - *fits to weak generational hypothesis*
- *Don't swap during GC*
  - *lock VM into memory*
- *Improve object locality*
  - *use large pages*

## Benchmarking GC

<i>Mix Pressure</i>	<i>Parallel Parallel</i>	<i>Parallel CMS</i>	<i>G1</i>
<i>old</i>	7775	11138	32800
<i>young</i>	1406	1302	3400
<i>object creation</i>	7275	7195	20835



## I/O

- *Interactions with devices that are 1000s of orders of magnitudes slower than local interactions*
- *Threads suspended waiting for I/O*
  - *no dominating consumer*
- *Thrash on I/O*
  - *OS becomes the dominating consumer*

## Disk I/O

- *Mechanical device optimized for chunky sized sequential reads*
- *Use buffered input/output*
- *Reduce load*
- *Compress data (trade CPU for disk)*
- *Stripe to increase throughput*

# Unix Kernel Counters

procs		memory			swap		io				system			cpu		
r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	wa	id	
2	1	207740	98476	81344	180972	0	0	2496	0	900	2883	4	12	57	27	
0	1	207740	96448	83304	180984	0	0	1968	328	810	2559	8	9	83	0	
0	1	207740	94404	85348	180984	0	0	2044	0	829	2879	9	6	78	7	
0	1	207740	92576	87176	180984	0	0	1828	0	689	2088	3	9	78	10	
2	0	207740	91300	88452	180984	0	0	1276	0	565	2182	7	6	83	4	
3	1	207740	90124	89628	180984	0	0	1176	0	551	2219	2	7	91	0	
4	2	207740	89240	90512	180984	0	0	880	520	443	907	22	10	67	0	
5	3	207740	88056	91680	180984	0	0	1168	0	628	1248	12	11	77	0	
4	2	207740	86852	92880	180984	0	0	1200	0	654	1505	6	7	87	0	
6	1	207740	85736	93996	180984	0	0	1116	0	526	1512	5	10	85	0	
0	1	207740	84844	94888	180984	0	0	892	0	438	1556	6	4	90	0	

# Network

- *Responsible for vast majority of IPC*
- *Caching to avoid*
- *Data set size matches network frame size*
- *Validate hardware configurations*
- *Diagnose with thread dump*

# Unix Kernel Counters

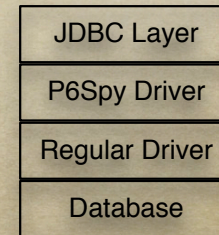
procs			memory				page				faults			cpu			
r	b	w	avm	free	re	at	pi	po	fr	de	sz	in	sy	cs	us	sy	id
6	5	0	6788303	15995602	429	105	0	0	0	0	0	43506	223636	35870	12	11	76
11	5	0	6798760	15996291	484	105	0	0	0	0	0	41273	224496	39269	11	13	76
11	5	0	6798760	15995053	434	102	0	0	0	0	0	41525	229932	40548	13	12	75
7	5	0	6734414	15993987	469	129	0	0	0	0	0	42806	238258	41581	12	12	75
7	5	0	6734414	15984722	984	134	0	0	0	0	0	41240	255757	42089	18	16	66
8	5	0	6753286	15986598	1117	190	0	0	0	0	0	41852	289565	42430	17	15	68
8	5	0	6753286	15993458	638	127	0	0	0	0	0	41050	246921	41123	12	12	75
10	6	0	6694867	15993275	442	112	0	0	0	0	0	41117	234697	41337	12	12	77
10	6	0	6694867	15992895	417	116	0	0	0	0	0	39506	226170	40361	12	12	75
9	5	0	6686343	15992543	420	124	0	0	0	0	0	39809	227487	40447	12	12	76
9	5	0	6686343	15991552	476	112	0	0	0	0	0	41320	233457	41181	12	12	76
11	5	0	6669621	15991648	426	104	0	0	0	0	0	39712	213137	36657	10	11	78
11	5	0	6669621	15992502	406	102	0	0	0	0	0	41535	212687	32910	9	11	80
7	5	0	6699466	15992379	393	102	0	0	0	0	0	39843	195238	29802	10	9	80
7	5	0	6699466	15992379	340	97	0	0	0	0	0	39377	186153	27820	9	9	81

# JDBC

- *Monitor JDBC calls*
  - *frequency and duration*
- *Reconcile response times with those reported by DB*
- *Many tools (commercial and OSS)*

# P6Spy

- *Sourceforge ([www.p6spy.org](http://www.p6spy.org))*
- *JDBC proxy*
  - *logs JDBC traffic*
- *Visualized with IronEye*



# P6Spy

IronGrid SQL

File View Server Help

Connect Disconnect Config Purge Import Export About

Legend: Preparation Execution Retrieval  
Slowest Most Run Both

Filtering (click to open)

SQL	Count	Avg Time	Max Time
SELECT LASTUSERID, OIACOMPANY, CODE, SHORTNAME, CCRULES, OIACC...	4124	2	16
SELECT LASTUSERID, OIACOMPANY, OIROOT, OROUPTYPE, CODE, HEAD...	3199	2	31
SELECT LASTUSERID, OIAPARENT, OIIDLANGUAGE, TEXT, OBJECTS, OBJEC...	3093	2	16
SELECT LASTUSERID, OIACOMPANY, OISEQUENCE, OIBTYPE, VALUEFROM...	3093	2	16
SELECT LASTUSERID, OIAGROUP, OIACOSTTYPE, ITEMPOSITION, OBJECTTS...	3093	9	16
SELECT LASTUSERID, OIAPARENT, OIIDLANGUAGE, TEXT, OBJECTS, OBJEC...	3093	2	16
SELECT LASTUSERID, OIAGROUP, OIACOSTTYPE, ITEMPOSITION, OBJECTTS...	3093	2	125
SELECT LASTUSERID, OIACOMPANY, OIROOT, OROUPTYPE, CODE, HEAD...	2070	2	31
SELECT LASTUSERID, OIACOMPANY, OIROOT, OROUPTYPE, CODE, HEAD...	2069	2	32
SELECT LASTUSERID, OIACOMPANY, OIROOT, OROUPTYPE, CODE, HEAD...	2069	2	32
SELECT LASTUSERID, OIAPARENT, OIIDLANGUAGE, TEXT, OBJECTS, OBJEC...	2065	2	32
SELECT LASTUSERID, OIAPARENT, OIIDLANGUAGE, TEXT, OBJECTS, OBJEC...	2065	2	32
SELECT LASTUSERID, CODE, OIIFRMA, VALIDFROM, VALIDUNTL, OBJECTS...	2063	2	16
SELECT LASTUSERID, BUDGETVAR, OIACONVERSIONTABLE, OIAPERIODDST...	2063	2	16
SELECT LASTUSERID, CODE, OIIFRMA, VALIDFROM, VALIDUNTL, OBJECTS...	2063	2	16
SELECT LASTUSERID, OIAGROUP, OIACOSTTYPE, ITEMPOSITION, OBJECTS...	2062	2	16
SELECT LASTUSERID, OIIBUDGETROW, OIACURRENCY, BALANCEDATE, AM...	1031	2	16
UPDATE COSTOROUPTYPE_V SET LASTUSERID = ?, OIAGROUP = ?, OIACOSTT...	1031	1	16
INSERT INTO COSTOROUPTYPE_V (LASTUSERID, OIAGROUP, OIACOSTTYPE, I...	1031	1	16
SELECT LASTUSERID, OIACOSTTYPE, OIIVARIANTE, CALCULATED, OIIBAMO...	1031	2	31
UPDATE COSTTYPEBASE_V SET LASTUSERID = ?, OIACOMPANY = ?, CODE = ?...	1031	2	47
SELECT LASTUSERID, LANGUAGE, RESOURCEMODULE, PARENT, COMPONE...	15	1	16
SELECT LASTUSERID, SETTINGUSER, SETTINGFILE, SETTINGSECTION, SETT...	15	1	16
SELECT LASTUSERID, CODE, LANGUAGE, ISDEFAULT, LANGUAGETEXTS, HAS...	6	3	16

SQL Statement Syntax

```

OIACOSTINGQUANTITY, ISASSIGNEDTOALLOC,
CITYPR, OIACONSURABILITY,
OIACOSTINGGROUPSIR, VARIABLEPERCENT,
IIRREDEPENDENTVAR, OBJECTS, OBJECTID
FROM COSTTYPEBASE_V WHERE OBJECTID = ?
    
```

Rows Returned: 0, max. avg. min.

Time Performance: 0, max. avg. min.

Count: 0, 2,000, 4,000

Count: 0, 2,000, 4,000

Data loaded from C:\local\p6\p6sp6.log

Not Connected



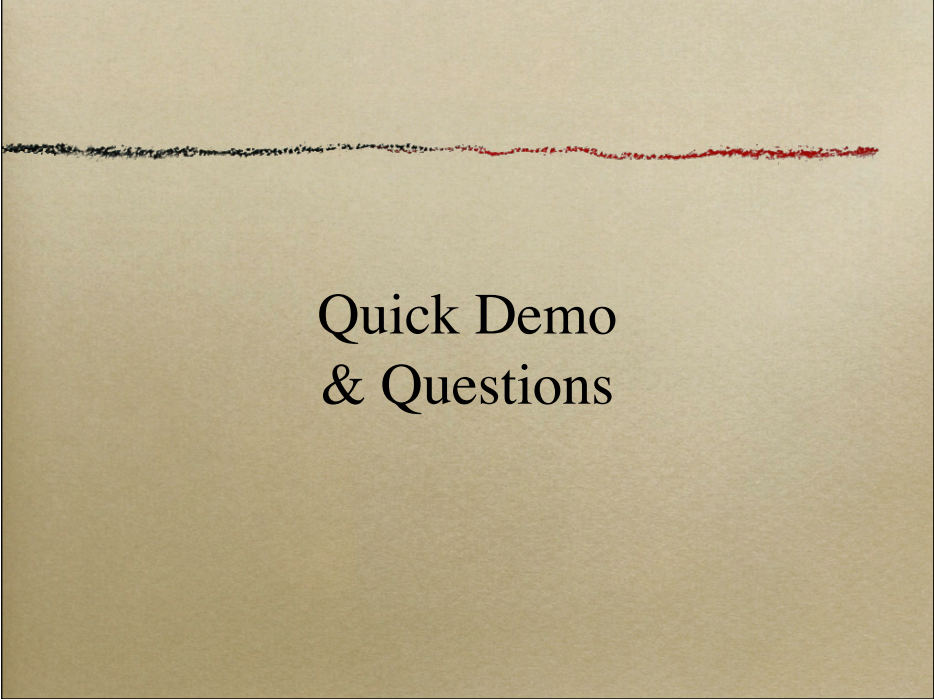
## Shared Data Structures

- *Data mutated by multiple threads must be synchronized (locked)*
- *Drive up rates of context switching*
  - *increase pressure on thread scheduler*
  - *not cover the costs of the context switch*
- *OS will be the dominating consumer*

Java locks will push the problem into application CPU burn  
can make it harder to find

## Finding Lock Contention

- *Thread and lock profilers*
  - *many vendor and OSS implementations*
- *Thread dumps*
  - *jstack (or visualvm)*
  - *TDA (Thread dump Analysis)*



## Quick Demo & Questions

- 1) GC Log viewing, followed by allocation stack traces
- 2) Thread dump followed by TDA