Andrey Nikolaev
RDTEX, Russia

Exploring mutexes, the Oracle RDBMS retrial spinlocks

MEDIAS - 2012
May 8-14

RDTEX
INFORMATION TECHNOLOGIES

Who am I

- Andrey.Nikolaev@rdtex.ru
- Graduated from MIPT in 1987
- 1987-1996 at COMPAS group, IHEP, Protvino
- RDTEX, First Line Support Center.
- http://andreynikolaev.wordpress.com

  "Latch, mutex and beyond"
- Specialize in Oracle performance tuning.
- Over 20 years of Oracle related experience as a research scientist, developer, DBA, performance consultant, lecturer…
- RUOUG member.

RDTEX
INFORMATION TECHNOLOGIES

Introduction

for non-Oracle

auditory

RDTEX
INFORMATION TECHNOLOGIES

Oracle RDBMS
performance improvements timeline:

v. 2 (1979): **the first commercial SQL RDBMS**

v. 3 (1983): the first database to support SMP

v. 4 (1984): read-consistency, Database Buffer Cache

v. 5 (1986): Client-Server, Clustering, Distributing Database, SGA

v. 6 (1988): procedural language (PL/SQL), undo/redo, latches

v. 7 (1992): Library Cache, Shared SQL, Stored procedures, 64bit

v. 8/8i (1999): Object types, Java, XML

v. 9i (2000): Dynamic SGA, Real Application Clusters

v. 10g (2003): Enterprise Grid Computing, Self-Tuning, **mutexes**

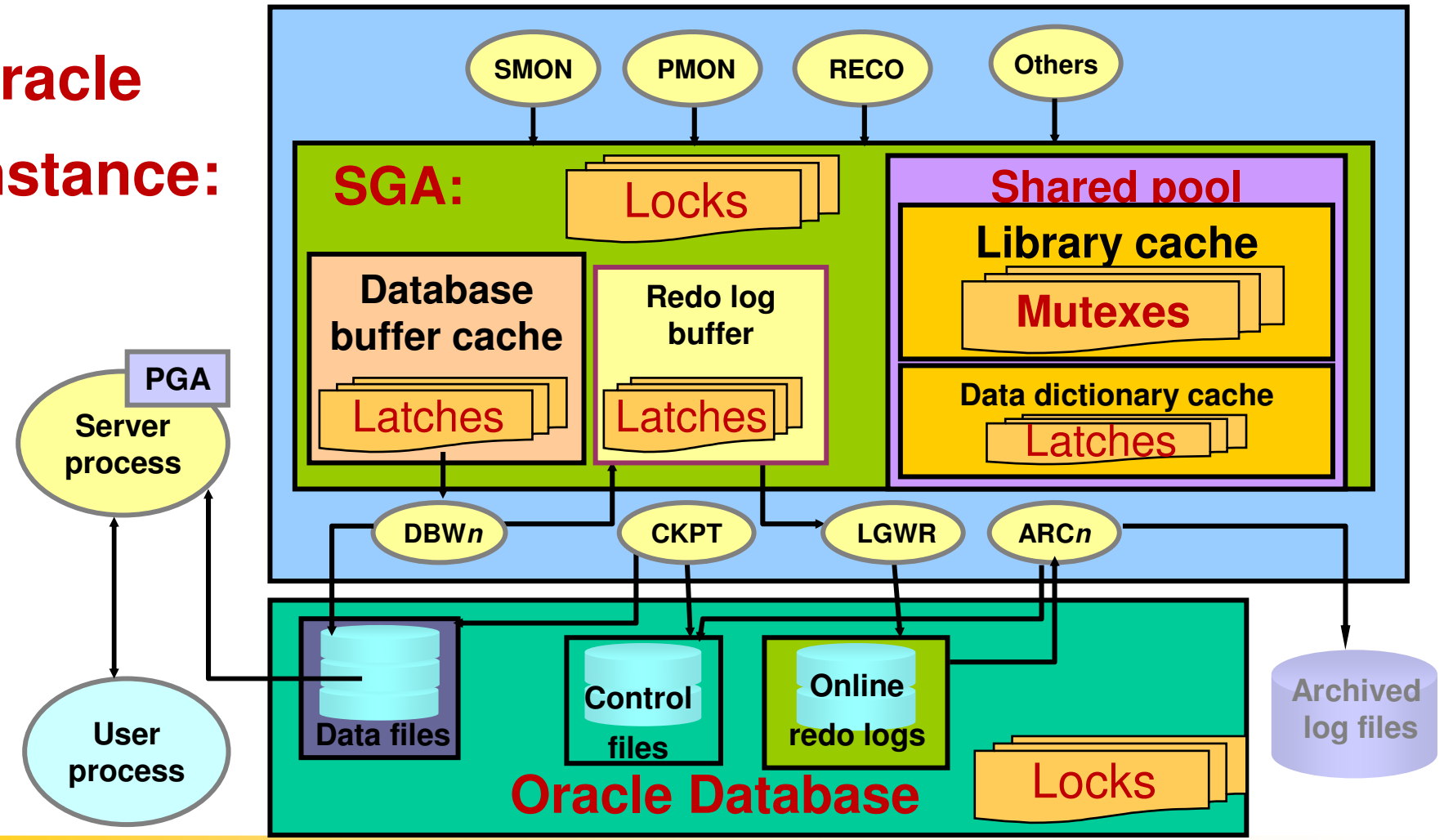
v. 11g (2008): Results Cache, SQL Plan Management, Exadata

v. 11gR2 (2011): … **Mutex wait schemes. Hot object copies**…

v. 12c (2012): *?Cloud? Not yet released … to be continued*

Oracle Database Architecture: Overview

Oracle instance:



Why Oracle needs Performance Tuning?

- More then 100 books on Amazon. *Need for mainstream science support!*
- Complex and variable workloads. Every database is unique.
- Complex internals. 348 "Standard" and 2655 "Hidden" tunable parameters.
- Complicated physical database and schema design decisions.
- Concurrency and Scalability issues.
- Insufficient developers education.
- "Database Independence" issues.
- Self-tuning anomalies. SQL plan instabilities.
- OS and Hardware issues.
- More than 10 million bug reports on MyOracleSupport.

Last year presentation about latches

- Year ago I discussed the most common Oracle spinlocks - latches:
  - How exclusive and shared latches works.
  - Exclusive latch spins 20000 cycles. Shared latch spins 4000 cycles.
  - Sometimes, it is worth to tune these numbers.
  - Processes requesting latch form a queue and use wait-posting.
- Today I will speak about KGX mutexes, the newest Oracle spinlocks:
  - Much less documented than needed to withstand mutex contention. Are one of the most complex DBA challenges today.
  - Various types and operations.
  - Sessions spin 255 cycles by default.
  - Session do not form the queue and retry spin attempts after sleeping.

Review of serialization mechanisms in Oracle

- "**Latches** are simple, low-level serialization mechanisms that coordinate multiuser access to shared data structures, objects, and files…"
- "A mutual exclusion object (**mutex**) is a low-level mechanism that prevents an object in memory from aging out or from being corrupted …"
- "Internal **locks** are higher-level, more complex mechanisms …"

Oracle® Database Concepts 11.2

- …"A mutex is similar to a latch, but whereas a latch typically protects a group of objects, a mutex protects a single object".
- KGX Mutexes appeared in latest Oracle versions inside Library Cache

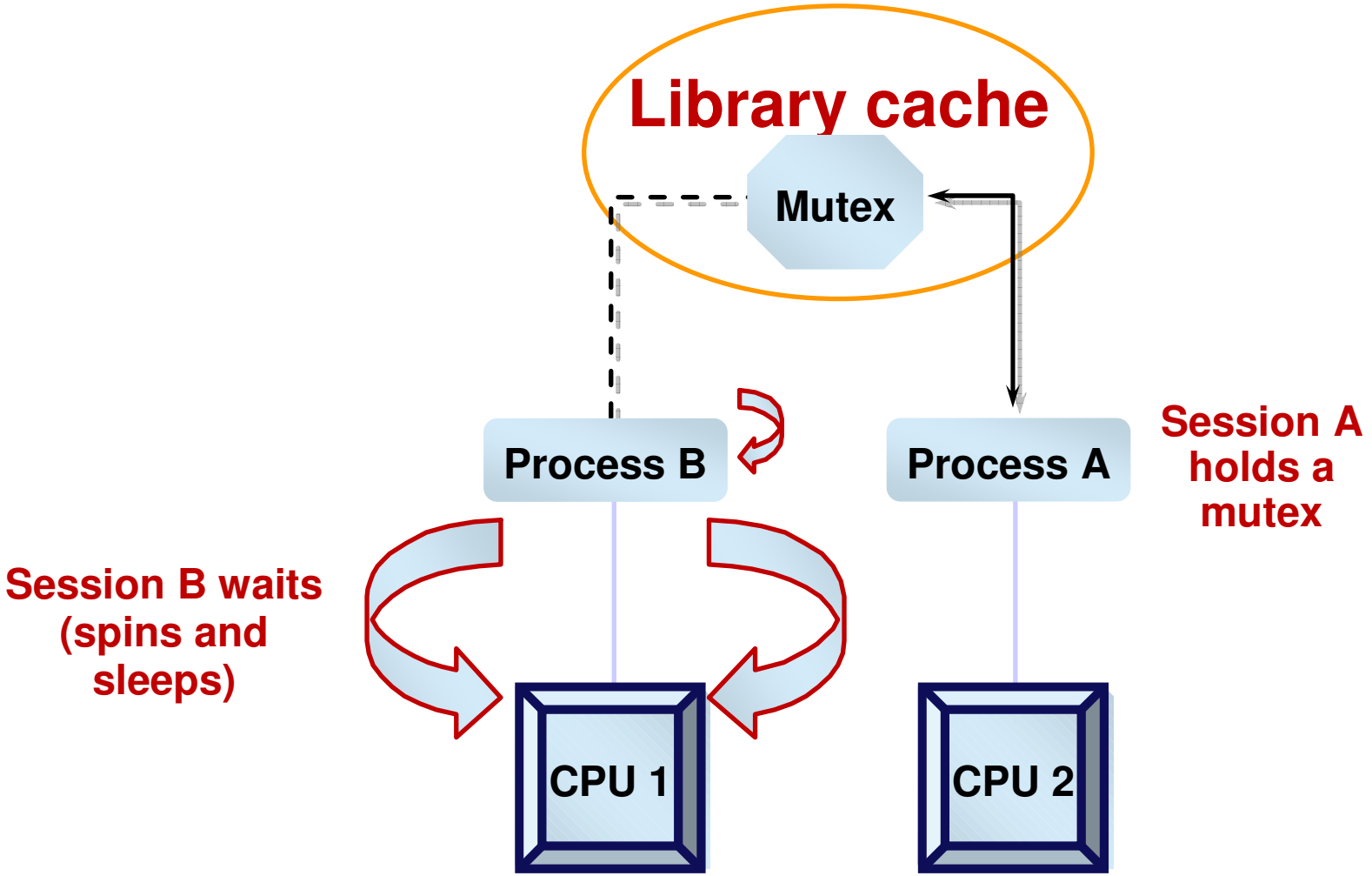
	Locks	Latches	Mutexes
Access	Several Modes	Types and Modes	*Operations*
Acquisition	FIFO	SIRO (spin) + FIFO	SIRO
SMP Atomicity	No	Yes	Yes
Timescale	> Milliseconds	Microseconds	SubMicroseconds
Lifecycle	Dynamic	Static	Dynamic

RDTEX INFORMATION TECHNOLOGIES

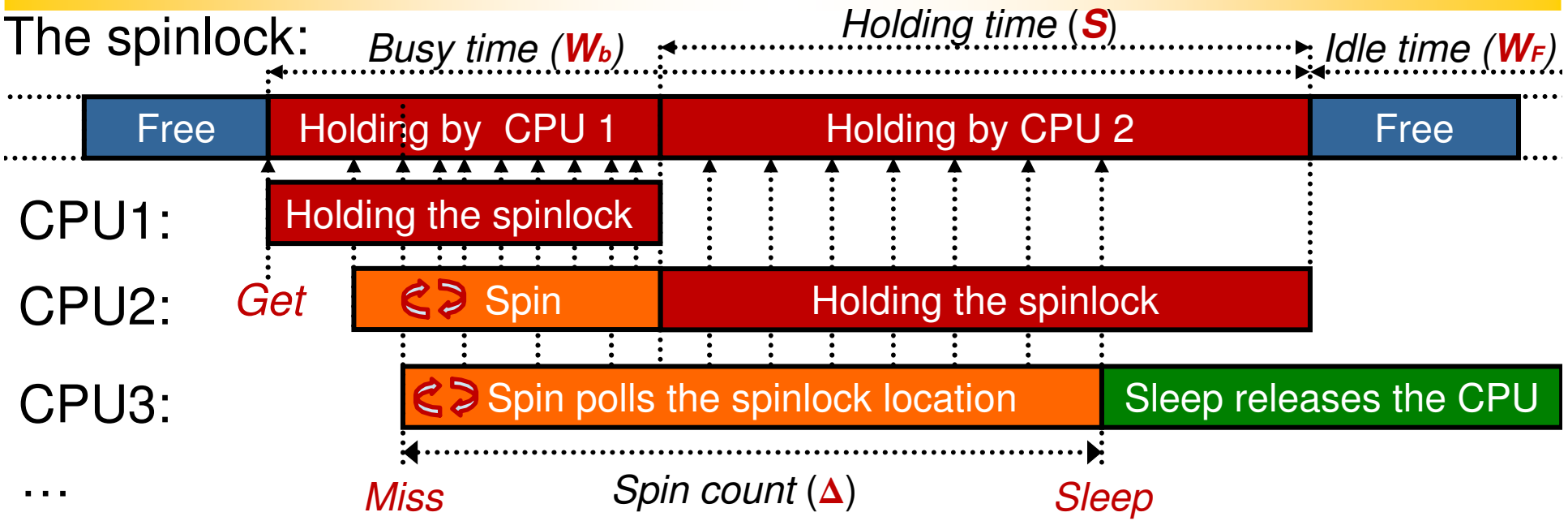
Classic spinlocks

- Latch and Mutex are spinlocks.
- Wiki: "… spinlock … waits in a loop repeatedly checking until the lock becomes available …". Uses atomic instructions.
- Introduced by Edsger Dijkstra [1] in 1965. Have been thoroughly investigated since that time [2].
- Many sophisticated spinlock realizations were proposed and evaluated (TS, TTS, Delay, MCS, Anderson,...).
- Two general types:
  - **System spinlock**. Kernel OS threads cannot wait. Major metrics: atomic operations frequency. Shared bus utilization.
  - **User spinlock**. Oracle latch and mutex. Average holding time is 1 us or less. It is more efficient to poll for a lock rather than preempt the thread doing 1 ms context switch. Metrics: CPU and elapsed times.

The spinlock in general

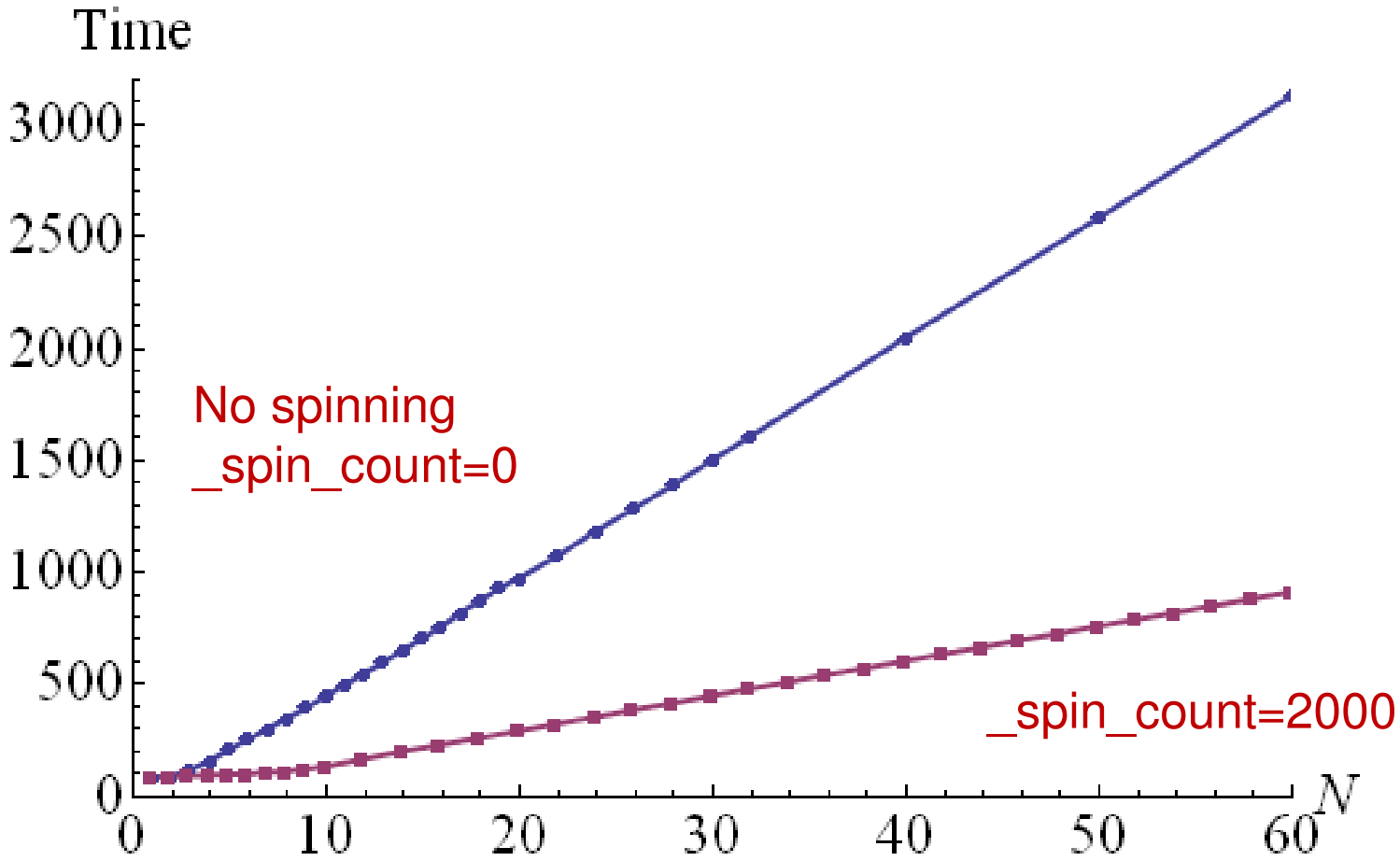


How Oracle spinlocks work



- Oracle spinlocks (latches and mutexes):
  - Use atomic hardware instruction for **Get.**
  - If **missed**, process spins by polling spinlock location during **spin get**.
  - Number of spin cycles is bounded by **spin count**.
  - In spin get not succeed, the process acquiring spinlock **sleeps**.
  - During the sleep the process may wait for already free spinlock.
- Oracle counts Gets and Sleeps and we can measure Utilization

The Need for Spin



Time to complete CBC latch contention testcase vs. number of threads.

The mutex statistics

Mutex requests arrival rate	$\lambda = \frac{\Delta\, gets}{\Delta\, time}$
Mutex sleeps rate	$\omega = \frac{\Delta sleeps}{\Delta time}$
Miss ratio estimation (PASTA)	$\rho = \frac{\Delta\, misses}{\Delta\, gets} \approx U_{X}$
Sleeps ratio	$\chi = \frac{\Delta sleeps}{\Delta misses} = \frac{\omega}{\lambda\rho}$
Avg. mutex holding time (Little's law)	$S = \frac{U}{\lambda}$
Mutex spin inefficiency	$\mathrm{k} = \frac{\Delta sleeps}{\Delta spins} = \frac{\chi}{\rho(1+\rho\chi)}$

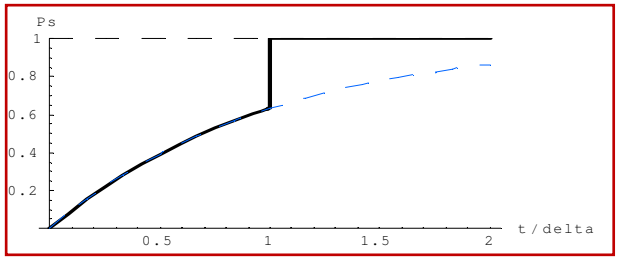
RDTEX INFORMATION TECHNOLOGIES

Mutex spin at moderate concurrency

- When no more then one process spin for mutex, the spin probes the mutex holding time distribution. The spin time p.d.f. is discontinuous at spin count:

$$P_{sg}(t_s < t) = \begin{cases} P_l(\tau_k < t) & \text{when} \quad t < \Delta \\ 1 & \text{when} \quad t \geq \Delta \end{cases}$$

$$p_{sg} = p_l(t)H(\Delta - t) + (1 - P_l(\Delta))\delta(t - \Delta)$$

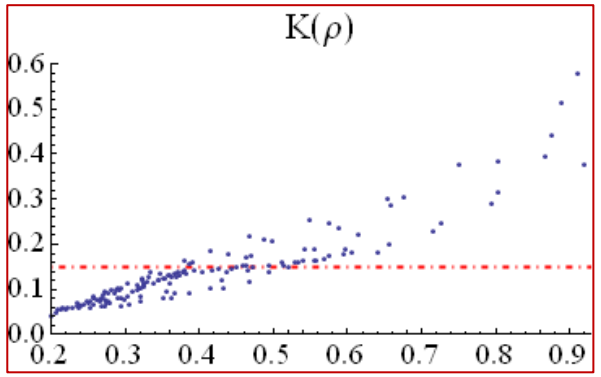


- According to standard renewal theory arguments the distribution of time until release is the transformed mutex holding time distribution:

$$p_l(t) = \frac{1}{S}(1 - P(t)) = \frac{1}{S}Q(t), \quad S = \langle t \rangle$$

- Spin inefficiency and average spin time are:

$$\begin{cases} k = \frac{1}{S}\int_{\Delta}^{\infty} Q(t)\mathrm{d}t \\ \Gamma_{sg} = \frac{1}{S}\int_{0}^{\Delta} \mathrm{d}t \int_{t}^{\infty} Q(z)\,\mathrm{d}z \end{cases}$$



- However, spin of many processes for mutex ($\rho \simeq 1$) became inefficient $K \simeq 1$

RDTEX INFORMATION TECHNOLOGIES

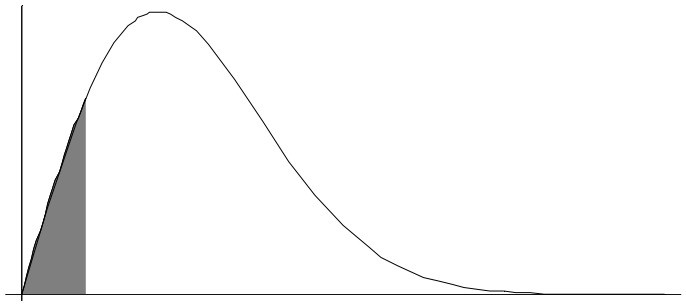
Low spin efficiency region

- To estimate effect of spinning, we can use approximate scaling rules depending on the value of:

K = "***spin inefficiency***"="***Sleeps per Spin***"

- If the spin is inefficient ***K***~1 then spin probes the mutex holding time distribution around the origin:

$$\begin{cases} \mathrm{k} = 1 - \frac{\Delta}{S} - \frac{1}{S}\int_0^{\Delta}(\Delta - t)p(t)\,\mathrm{d}t \\ \Gamma_{sg} = \Delta - \frac{\Delta^2}{2S} + \frac{1}{2S}\int_0^{\Delta}(\Delta - t)^2 p(t)\,\mathrm{d}t \end{cases}$$



- If processes do not release mutex immediately:

$$\begin{cases} \mathrm{k} = 1 - \frac{\Delta}{S} + O(\Delta^3) \\ \Gamma_{sg} = \Delta - \frac{\Delta^2}{2S} + O(\Delta^4) \end{cases}$$

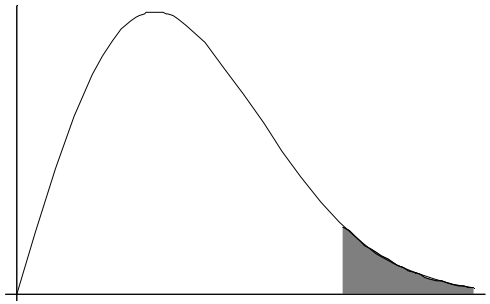
therefore:

When spin efficiency is low, doubling the spin count will double the number of efficient spins and also double the CPU consumption.

High spin efficiency region

- In high efficiency region the sleep cuts off the **tail** of spinlock holding time distribution:

$$\begin{cases} k = \frac{1}{S}\int\limits_{\Delta}^{\infty}(t-\Delta)p(t)\ \mathrm{d}t \\ \Gamma = \frac{\langle S^2\rangle}{2S} - \frac{1}{2S}\int\limits_{\Delta}^{\infty}(t-\Delta)^2 p(t)\ \mathrm{d}t = \frac{\langle S^2\rangle}{2S} - kT_r \end{cases}$$



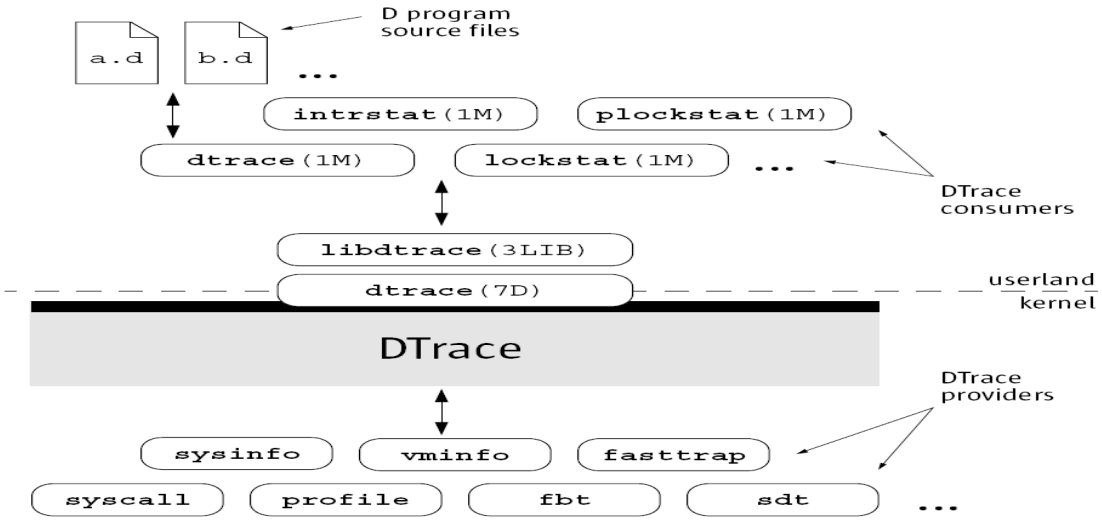
here ***Tr*** is the residual after-spin holding time.

- Oracle normally operates in this region of small sleeps ratio $K < 0.1$
- Here spin count is greater than number of instructions protected $\Delta \gg S$
- The spin time is bounded by the "residual latch holding time" and spin count:

$$\Gamma_{sg} < \min\left(\frac{\langle S^2\rangle}{2S},\ \Delta\right)$$

Sleep prevents spinlock from burning CPU for spinning on heavy tail of holding time distribution.

DTrace. Solaris Dynamic Tracing framework



Allows us to investigate how Oracle mutexes perform in real time:

- Create triggers on any event inside Solaris and function call inside Oracle.

```
provider:module:function:name
pid1910:oracle:kgxExclusive:entry
```

- Write trigger bodies – actions. May read and change any address location.
- Can count the mutex spins, trace the mutex waits, perform experiments.
- Measure times and distributions up to microsecond precision.

How Oracle requests the mutex

Mutex contention testcases

• “Cursor: pin S” mutex contention arises when SQL executes concurrently at high frequency.

```
begin
 for i in 1..1000000
 loop
   execute immediate 'select 1 from dual where 1=2';
 end loop;
end;
```

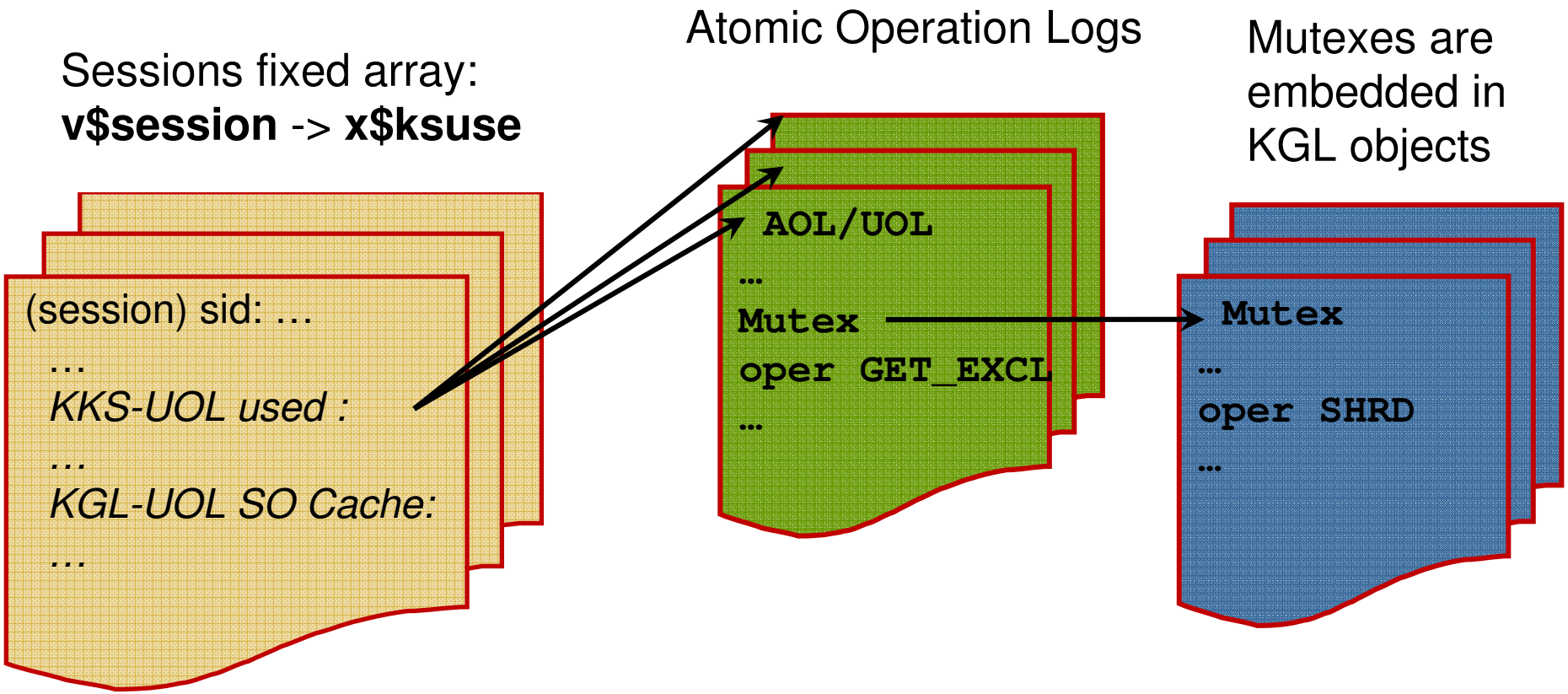
• Just set **session_cached_cursors=0** and add dozen versions of the SQL for "Cursor: mutex S" testcase.

•“Library cache: mutex X” contention arises when anonymous PL/SQL block executes concurrently at high frequency in 11g.

```
begin
 for i in 1..1000000
 loop
   execute immediate 'begin demo_proc();end;';
 end loop;
end;
```

• Many other mutex contention scenarios possible.

Mutex is holding by session:



- Each session has an array of references to the AOL/UOL it is using.
- Compare to latch, which is holding by the process.

Mutex types in 11.2.0.3

Type_id	Mutex_type	Protects:
7	*Cursor Pin*	*pin cursor*
8	hash table	Cursor management
6	Cursor Parent	…
5	Cursor Stat	…
4	Library Cache	Library cache
3	HT bucket mutex (kdlwl ht)	SecureFiles
2	SHT bucket mutex	…
1	HT bucket mutex	…
0	FSO mutex	…

- "**Cursor Pin**" mutexes act as pin counters for library cache objects (e.g. child cursors) to prevent their aging out of shared pool.
- "**Library cache**" cursor and bucket mutexes protect KGL locks and static library cache hash structures.
- The presentation will explore these two mutex types.
- "**Cursor Parent**" and "**hash table**" mutexes protect parent cursors during parsing and reloading.

Mutex structure in memory

```
SQL> oradebug peek 0x3F119B5A8 24
[3F119B5A8, 3F119B5CC) =
  00000016 00000001 0000001D 000015D7 382DA701 03000000 …
       SID    refcnt     gets    sleeps      idn    op
```

Mutex structure contains:

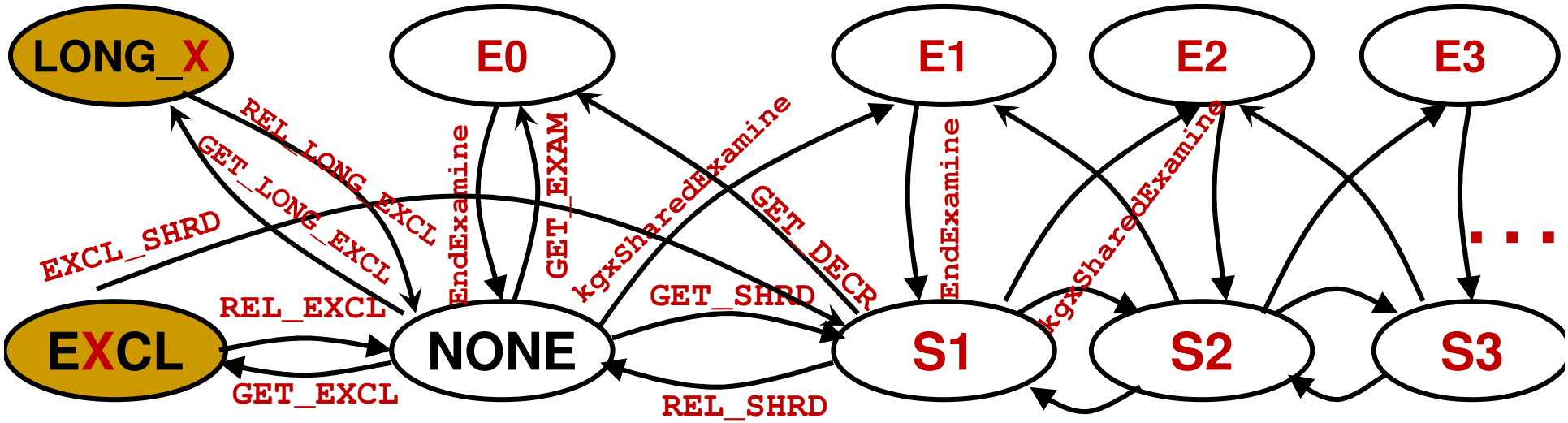
- Atomically modified value that consist of two parts:
  - "Holding SID." Top 4 bytes contain SID session currently holding the mutex eXclusively or modifying it.
  - "Reference count." Lower 4 bytes represent the number of sessions currently holding the mutex in Shared mode (or is in-flux).
- GETS - number of times the mutex was requested
- SLEEPS - number of times sessions slept for the mutex
- OP – Current mutex operation

Mutex value and modes

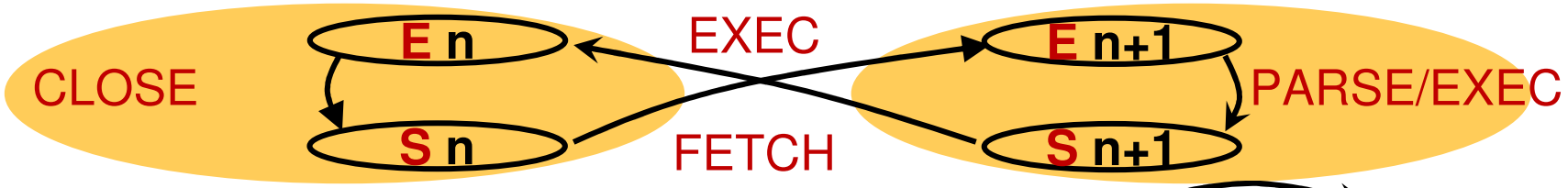
- One WORD (8- or 4-byte). Changed using single atomic CAS instructions.
- **S** mode:
  - Allows mutex to be held by several sessions simultaneously.
  - **0x00000000** | "**Reference Count**" represents the number of sessions holding the mutex.
- **X** mode:
  - Only one session can hold the mutex in exclusive mode.
  - "**Holder SID**" | **0x00000000**
- **E**xamine mode. Upper bytes are equal to holder SID. Lower bytes are nonzero and represent the number of sessions simultaneously holding the mutex in **S** mode.
  - **“Holding SID" | "Reference Count”. SHRD_EXAM** in traces**.**
  - Session can acquire mutex in **E** mode or upgrade it to **E** mode even if there are other sessions holding mutex in **S** mode.
  - No other session can change mutex at that time.

The mutex state diagram

- Experiments revealed the following mutex transitions diagram:



- Not all operations used by each mutex type. The "**Cursor Pin**" mutex pins the cursor in Library Cache for parse and execution in 8-like way:



- "**Library Cache**" mutex uses **X** mode only:
- "**Hash table**" mutexes utilize both **X** and **S**.

Mutex waits in Oracle Wait Interface

WAIT EVENT NAME	PARAMETER1	PARAMETER2	PARAMETER3
cursor: mutex X	idn	value	where
cursor: mutex S	idn	value	where
cursor: pin X	idn	value	where
cursor: pin S	idn	value	where
cursor: pin S wait on X	idn	value	where
library cache: mutex X	idn	value	where
library cache: mutex S	idn	value	where
SecureFile mutex	?	?	?

Get: \ Held:	S	X, LX	E
S	-	*mutex type* S	?
X	*mutex type* X	*mutex type* X	*mutex type* X
E	-	*mutex type* S wait on X	*mutex type* S

- This presentation will focus on most frequently observed "**cursor: pin S**" and "**library cache: mutex X**" waits.

RDTEX INFORMATION TECHNOLOGIES

Experimental setup to explore mutex wait

- Unlike the latch, details of mutex wait were never documented by Oracle.
- To explore the latch last year [7], I artificially acquired it calling **kslgetl** function. This is not possible for mutex.
- However, I can make mutex "busy" artificially:

```
SQL>oradebug poke <mutex address> 8 0x100000001
BEFORE: [3A9371338, 3A9371340) = 00000000 00000000
 AFTER: [3A9371338, 3A9371340) = 00000001 00000001
```

- This looks exactly like session with SID 1 is holding the mutex in **E** mode.
- I wrote several scripts that simulate a busy mutex in S, X and E modes:
  - One session artificially holds "**Cursor Pin**" mutex for **50s**.
  - Another session "statically" waits for "**cursor: pin S**" event during **49s**.
- This allowed me explore how Oracle waits for mutex.

Original Oracle mutex busy wait (2005)

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% Total Call Time	Wait Class
CPU time		59		93.5	
control file parallel write	384	11	28	17.0	System I/O
db file parallel write	436	6	13	9.0	System I/O
log file parallel write	161	4	24	6.2	System I/O
cursor: pin S	2,865,013	3	0	4.7	Other

- The waiting process consumed one of my CPUs completely.
- Millions of microsecond waits accounted for 3 s out of actual 49 s.

```
… spin 255 cycles
   yield()
  spin 255 cycles
   yield()
… repeated 1910893 times
```

- The session waiting for mutex spins and allows other processes to run.
- Wait time was the time spent off the CPU. If system has free CPU power, Oracle thought it was not waiting at all and mutex contention was invisible.
- If sessions held mutex for a long times, spinning resulted in waste of CPU.

Mutex true sleeps (patch 6904068)

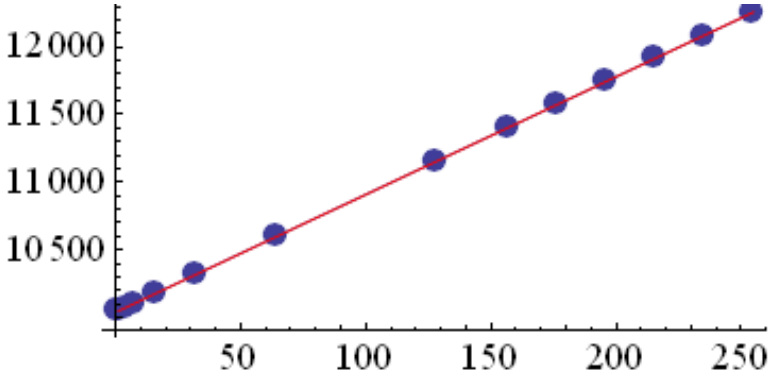
- If long "**cursor: pin S**" waits were consistently observed in Oracle 10.2-11.1," then your system did not have enough CPU for busy waiting.
- Oracle recognized this problem and provided the possibility to convert "busy" mutex wait into "true sleeps". This is legendary **Patch 6904068: High CPU usage when there are "cursor: pin S" waits**.
- We can adjust sleep time between spins with centisecond granularity.

```
SQL> alter system set "_first_spare_parameter"=1;
SQL> select 1 from dual where 1=2;
 kgxSharedExamine(…)
  spin 255 times
  semsys()    timeout=10 ms
 … repeated 4748 times
```

- The patch significantly decreases CPU consumption. Its drawback is larger elapsed time to obtain mutex**.**
- It make sense to install the patch **6904068** in 10.2-11.1 OLTP proactively.

Average mutex holding time

- **`mutex_statistics.sql`** – measures and computes statistics for given mutex address.
- Changing mutex spin count we can measure spin and yield times.



- Typical nocontention values for mutex holding time **S** in exclusive mode on some platforms:

	Library cache	Cursor pin	spin time	yield() time
Exadata X2-2 ¼	0.3-5 us	0.1-2 us	1.8 us	0.7 us
Niagara T2000	2.5-12 us	3.2-11 us	8.7 us	9.5 us

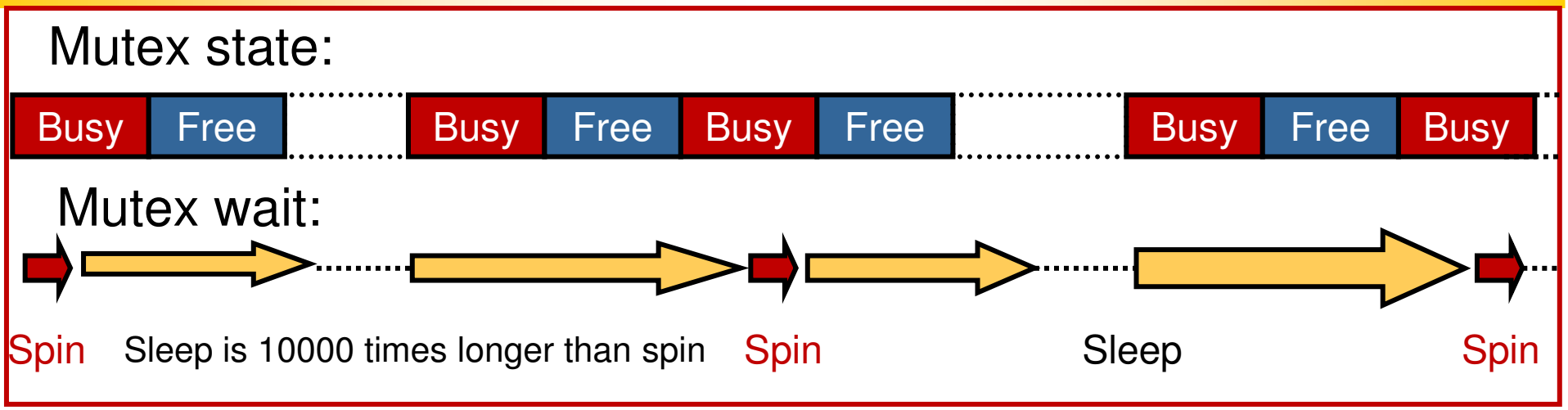
- Compare these microsecond times with default mutex sleep of 1 cs duration.
- Since it is hard for us, humans, to visualize milli- and microseconds, I found useful the following illustration.

Imagine a Time Microscope:

Reality:	**One million times zoomed:**
1 us	1 sec
1 ms	17 min
1 sec	11.5 days
Light speed (300000 km/sec)	Sonic speed (300 m/sec)
CPU tick (3 GHz)	0.0003 sec
Normal "**Library cache" mutex** holding time	**~ 0.3-5 sec**
Max spin time for mutex (255 spins)	**~1.8 sec**
"Library Cache" latch holding time was 10-20us	*~ 10-20 sec*
Max spin time for exclusive latch (20000 spins)	*~50 sec*
Min interval between mutex gets	~ 2 sec
OS context switch time (10us-1ms)	*10 sec – 17 min*

RDTEX
INFORMATION TECHNOLOGIES

Mutex waits under the Time Microscope



- Mutex wait with patch 6904068:
  - **Spin** for mutex during **1.8 second.**
    - **Sleep** for **2 hours 46 min** (1cs) in hope that congestion will dissolve.
  - **Spin** again during 1.8 seconds.
    - **Sleep** again for 2 hours 46 min.

    …
- Sleeps duration is much longer than normal mutex correlation time.
- Each time waiting process awakens, it should observe independent picture.
- Why it waits so long? Most OS can not sleep less than 10ms by default.

"Mean Value Analysis" of mutex retrials

- MVA is an elegant approach for queueing models [4,5]. Though not applicable directly to non-Markovian mutex, it can be used for estimations.
- According to PASTA, request arriving with frequency λ finds mutex busy with probability ρ and goes to orbit (sleeps!) for time T with probability ρk.
- The waiting time consist of spin and sleep: $W = W_s + W_{orb}$
- The process acquires the mutex during repeating spins:

$$W_s = \rho\Gamma + (k\rho)\rho\Gamma + (k\rho)^2\rho\Gamma + \ldots = \frac{\rho}{1-k\rho}\Gamma$$

- It retries from orbit while the mutex is busy and idle:

$$W_{orb} = W_b + W_i$$

- In steady state the busy mutex wait time should be: $W_b = L_{orb}S + \rho k T_r$
- According to Little's law: $L_{orb} = \lambda W_{orb}, \quad L_b = \lambda W_b, \quad \rho = \lambda S$
- Flows per second to and from the orbit should be balanced:

$$k\lambda\rho + k\frac{\lambda W_b}{T} = \frac{\lambda W_{orb}}{T}$$

MVA for mutex with patch 6904068

- Therefore, the average summary wait time for mutex with patch 6904068 may be estimated as:

$$W = \frac{\rho}{(1-k\rho)}\left(\Gamma + k\left(T + kT_r\right)\right)$$

- Unlike the queueing theory, the Oracle Wait Interface does not treat the first spin as a part of wait:

$$W_o = \frac{k\rho}{(1-k\rho)}\left(T + \rho\Gamma + kT_r\right)$$

and average OWI (and AWR) mutex wait duration becomes:

$$\overline{w}_o = \frac{1}{(1-k\rho)}\left(T + \rho\Gamma + kT_r\right)$$

- Normally in Oracle 11.2 the huge sleep ***T*** dominates in the above formulas and limits the mutex wait performance.

$$T \sim 10^4 \times \{\Gamma, \Delta, T_r\}, \quad K \sim 0.1$$

- Such estimations do not account for OS scheduling and are not applicable when number of active processes exceeds the number of CPUs.

11.2.0.2.2 Mutex waits diversity

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
cursor: pin S	1	50	49562	92.80	Concurrency
db file sequential read	381	6	16	11.47	User I/O
DB CPU		4		7.05	

- Oracle 11.2 Wait Interface computes "**wait time**" as a duration between the first sleep and successful mutex acquisition. This results in **single wait**.
  - We still observe millions of sleeps in **v$mutex_sleep_history**.
- Allows one of three concurrency wait schemes and introduced 3 parameters to control the mutex waits:
  - **_mutex_wait_scheme** – Which wait scheme to use.
    - 0 – Always YIELD.
    - 1 – Always SLEEP for **_mutex_wait_time**.
    - 2 – Default. **Exponential Backoff** with maximum sleep **_mutex_wait_time**.
  - **_mutex_spin_count** - the number of times to spin. Default value is 255
  - **_mutex_wait_time** – sleep timeout depending on scheme. Default is 1.

Simply SLEEPS. _mutex_wait_scheme 1

- In mutex wait scheme 1 session repeatedly requests 1 ms sleeps:

```
SQL> select 1 from dual where 1=2;
kgxSharedExamine(…)
  yield()
  pollsys()   timeout=1 ms    call repeated 25637 times
```

- **_mutex_wait_time** parameter controls sleep timeout in **milliseconds**.
- The millisecond sleep will be effectively rounded to centisecond on most platforms (Solaris, Windows, AIX, latest HP-UX).
- This scheme differs from **patch 6904068** by one additional spin and **yield()** cycle at the beginning (and another syscall). To estimate its effect suppose that spins are independent:

$$W_1 \approx \frac{\rho}{(1-k\rho)}\left(\Gamma + k\left(kT + kT_r\right)\right)$$

- The additional spin effectively reduces wait time ***T*** multiplying it by ***k*** and increase the performance.

Default "Exponential Backoff" scheme 2

- Surprisingly, DTrace shows that there is no exponential backoff by default. Session repeatedly sleeps with 1 cs duration:

```
yield() call repeated 2 times
semsys()   timeout=10 ms    repeated 4237 times
```

- To reveal exponential backoff one need to increase the **_mutex_wait_time.** This parameter control maximum wait time in centiseconds.
- Default mutex wait scheme 2 parameters result in behavior that differs from patch 6904068 by two **yield()** syscalls at the beginning.
- These two spins and yields change the mutex wait performance drastically.
- In moderate concurrency region they effectively reduce centisecond wait time ***T*** by squared mutex spin inefficiency:

$$W_2 \approx \frac{\rho}{(1-k\rho)}\left(\Gamma + k\left(k^2 T + kT_r\right)\right)$$

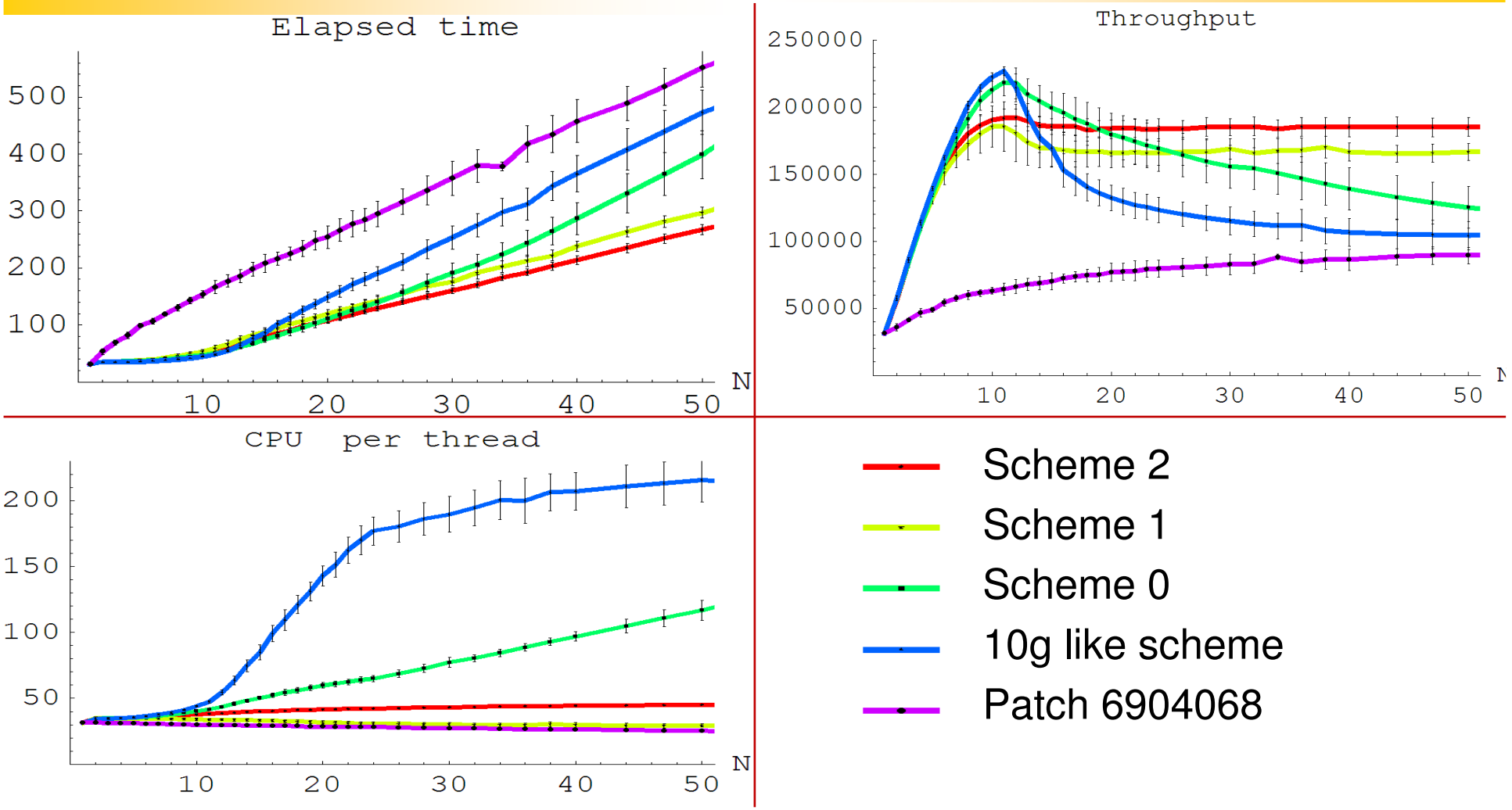
Classic YIELDS. _mutex_wait_scheme 0

- Differs from aggressive mutex waits used in previous versions by 1ms sleep after each 99 yields.

```
kgxSharedExamine()
 yield() call repeated 99 times
 pollsys() timeout=1 ms
 yield() call repeated 99 times
 pollsys() timeout=1 ms
…
```

- This 1 ms sleep significantly reduces CPU consumption and increases robustness in high contention.
- Sleep duration and yield frequency are tunable by **_wait_yield_mode**, **_wait_yield_sleep_time_msecs** and **_wait_yield_sleep_freq** parameters.
- The scheme is very flexible. It allows almost any combination of yield and sleeps including 10g and patch **6904068** behavior [8].

Comparison of mutex wait schemes



"**Library cache: mutex X**" testcase on 12 Cores (24 SMT) X2-2 ¼ Exadata.

RDTEX INFORMATION TECHNOLOGIES

Comparison of mutex wait schemes

- Default scheme 2 is well balanced in all concurrency regions.
- Previous 10.2-11.1 algorithm:
  - Had the fastest performance in medium concurrency workloads.
  - However, its throughput fell down when number of contending threads exceeds number of CPU cores. CPU consumption increased rapidly.
  - This excessive CPU consumption starves CPUs and impacts other database workloads.
- **Patch 6904068** results in very low CPU consumption, but the largest elapsed time and the worst throughput.
- All the contemporary 11.2.0.2.2 mutex schemes consume less CPU than before.
  - Scheme 1 should be used when the system is constrained by CPU.
  - Scheme 0 has the throughput close to 10.2 in medium concurrency region and should be used if you have free CPU resources.

Mutex Contention

Contention diagnostics

Little's law : $U = \lambda S$

- Mutex contention occurs when the mutex is requested by several sessions at the same time. Can be consequence of:
  - Long mutex holding time **S** due to:
    - high version count,
    - bugs causing long mutex holding time,
    - CPU starvation and preemption …
  - High mutex exclusive **U**tilization due to frequent requests caused by:
    - high SQL and PL/SQL execution rate,
    - bugs causing excessive mutex requests …
- Mutex statistics helps to diagnose what actually happens.
- Latest Oracle versions include fixes for many mutex related bugs and:
  - Flexible mutex **wait schemes.**
  - Mutex **spin count tuning**
  - **Cloning** of hot library cache objects.

Let it spin

- The spinlocks were designed to spin. Let them do it!
- Long mutex holding time may cause the mutex contention.
- Default **_mutex_spin_count =255** may be too small. *1.8 sec under time microscope.*
- Spinning may alleviate this.
- My experiments showed that frequently mutex holding time distribution has exponential tail:

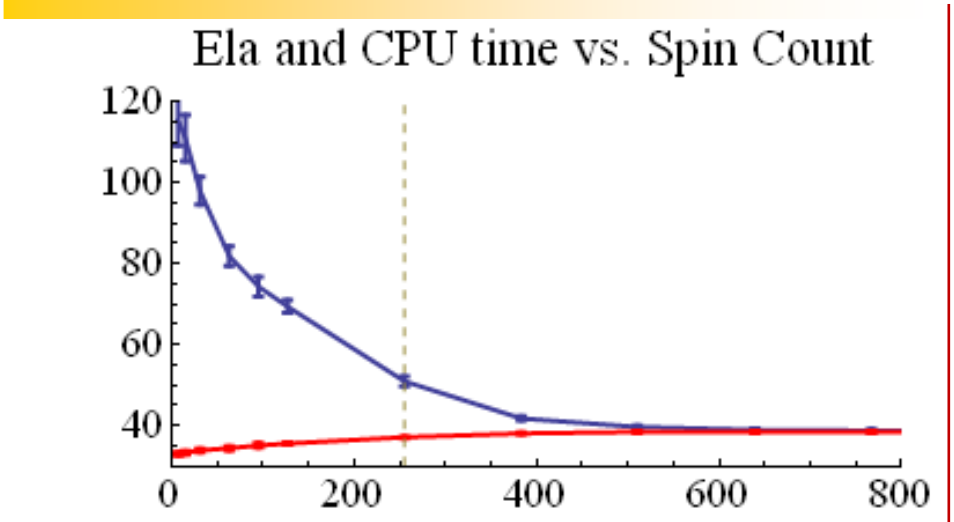
$$\begin{cases} Q(t) \sim C\exp(-t/\tau) \\ \mathrm{k} \sim C\exp(-t/\tau) \\ \Gamma_{sg} \sim \frac{\langle S^2 \rangle}{2S} - C\tau\exp(-t/\tau) \end{cases}$$

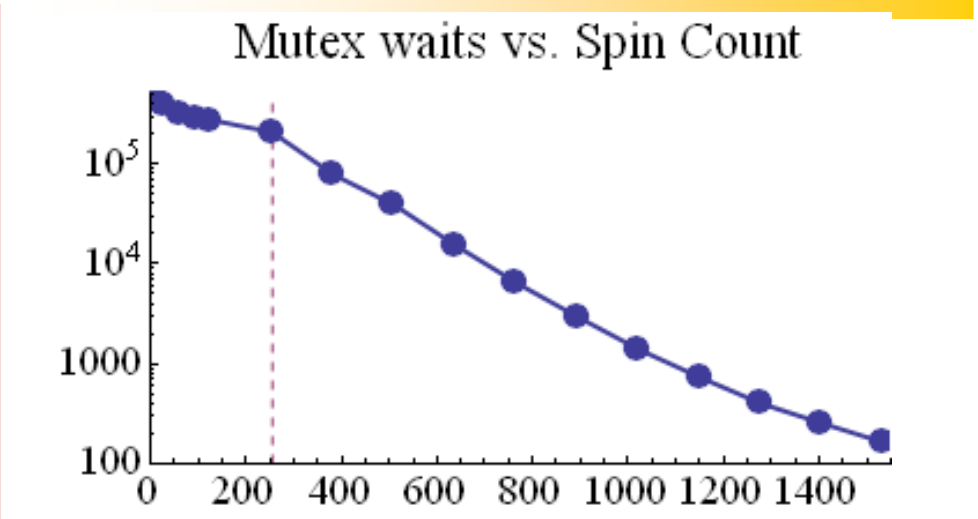
- My previous work [7] proposed approximate scaling rule to estimate effect of spin count tuning when "spin inefficiency" ***k <<0.1:***

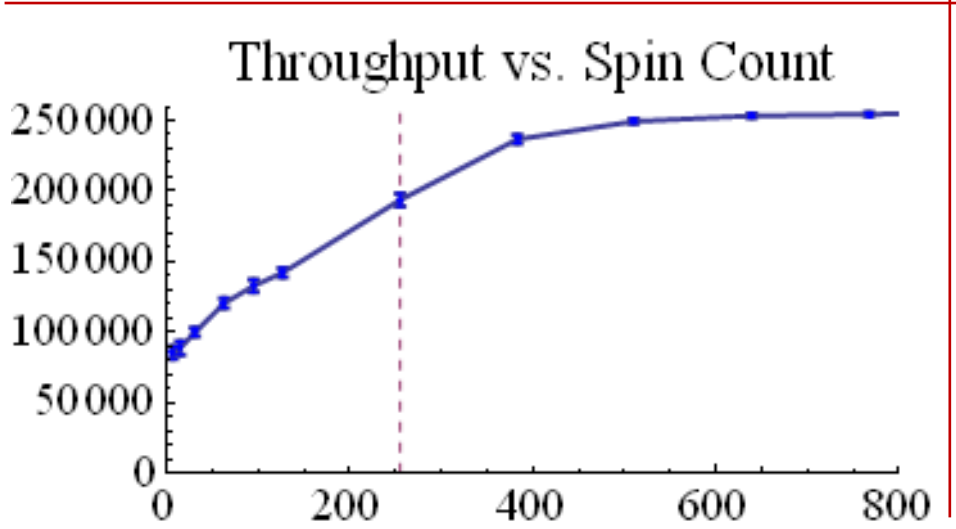
Doubling the spin count will square the "spin inefficiency" and add *k*'th part to the CPU consumption.

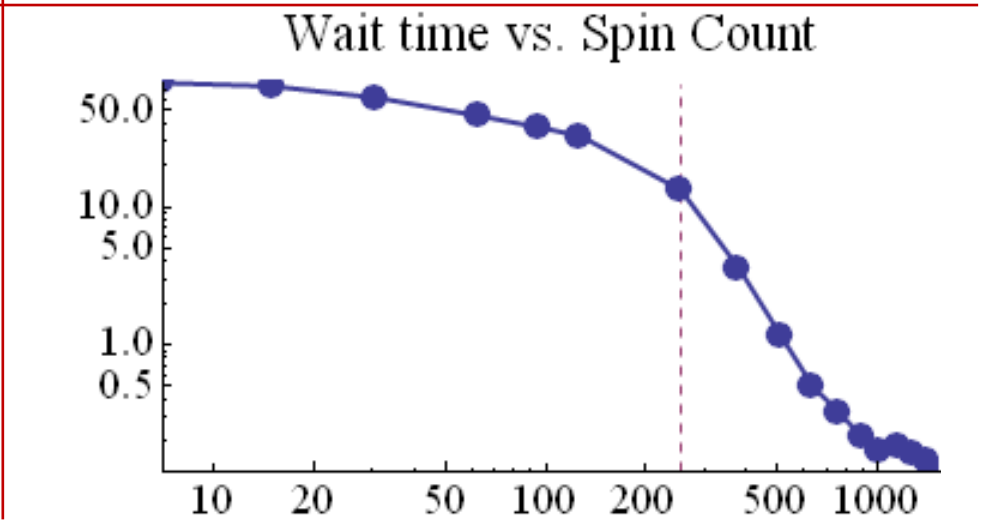
If the spin is already efficient, it is worth to increase the spin count.

Library cache mutex contention testcase







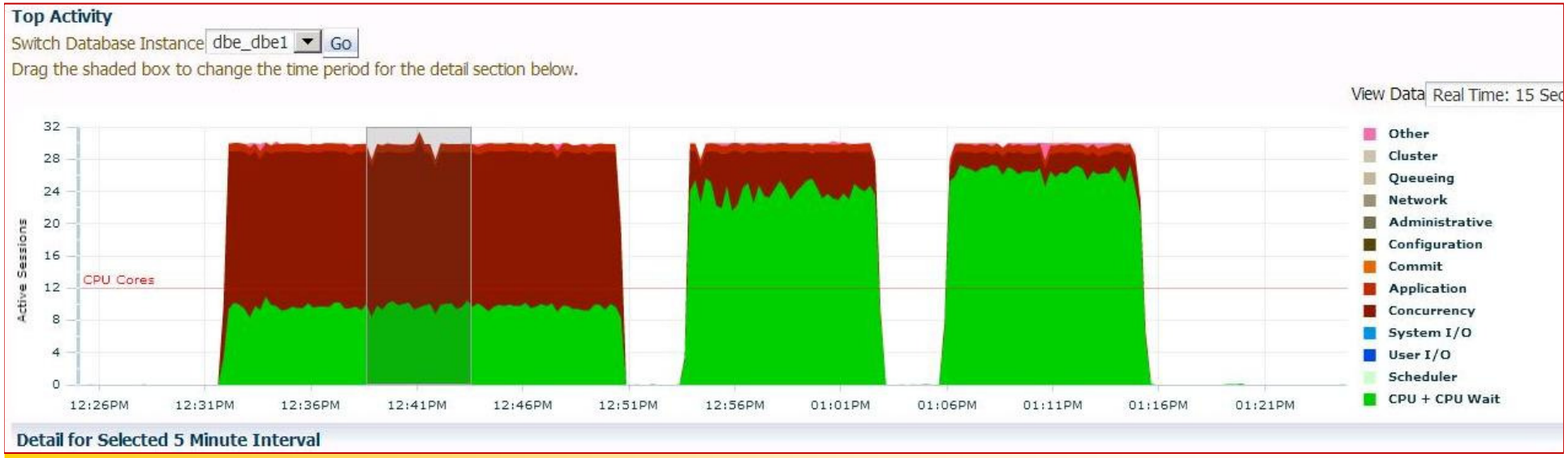


Waits, times and throughput vs. **_mutex_spin_count**

RDTEX
INFORMATION TECHNOLOGIES

"Divide and Conquer" the mutex contention

- Contention for some highly utilized objects can be divided between multiple copies of object in the library cache [8].
- If each session would have its own copy of object there would be no contention:
- **_kgl_hot_object_copies** parameter controls the maximum number of copies.
- **_kgl_debug** or dbms_shared_pool.markhot() mark library cache objects as candidates for cloning.



RDTEX INFORMATION TECHNOLOGIES

Mutex SMP scalability:

- If exclusive mutex utilization is ρ_1 in single CPU environment.
- Then in N CPU server latch utilization will be $\rho_N \approx N\rho_1$. This can be problematic:
  - If double CPU system held mutex for only 1% of time.
  - 48 CPU server with the same per-CPU load will hold mutex for 25%.
  - 128 CPU Cores server will suffer huge mutex contention.
  - What about 1024 SMT?
- This is also known as **"Software lockout"** [5]. It may substantially affect contemporary multi-core servers.
- High granularity mutexes, NUMA pools, and hot objects copies mechanism should overcome this intrinsic spinlock scalability restriction.

Bibliography

1. Edsger Dijkstra. **Solution of a Problem in Concurrent Programming Control** CACM. 1965.
2. M. Herlihy, N. Shavit, **The Art of Multiprocessor Programming.** 2008.
3. Steve Adams, **Oracle8i Internal Services for Waits, Latches, Locks, and Memory**. 1999.
4. M. Reiser, S. Lavenberg, **Mean-Value Analysis of Closed Multichain Queuing Network.** Journal of the ACM 27 (2): 313. 1980.
5. J.R. Artalejo, J.A.C. Resing, **Mean Value Analysis of Single Server retrial Queues**. APJOR 27(3): 335-345. 2010
6. R. Johnson, et al. **A new look at the roles of spinning and blocking**. Proc. of 5th Workshop on Data Management on New Hardware. 2009
7. A. Nikolaev, **Exploring Oracle RDBMS latches using Solaris DTrace.** Proc. of MEDIAS 2011 Conf., **http://arxiv.org/abs/1111.0594v1** 2011
8. My blog, **http://andreynikolaev.wordpress.com**

Q/A?

- Questions?
- Comments?

Acknowledgements

- Thanks to Professor S.V. Klimenko for kindly inviting me to MEDIAS 2012 conference.
- Thanks to RDTEX CEO I.G. Kunitsky for financial support.
- Thanks to RDTEX Technical Support Centre Director S.P. Misiura for years of encouragement and support of my investigations.
- Thanks to my colleagues for discussions and all our customers for participating in the mutex troubleshooting.

RDTEX
INFORMATION TECHNOLOGIES

Thank You!

Andrey Nikolaev

http://andreynikolaev.wordpress.com

Andrey.Nikolaev@rdtex.ru

RDTEX, Moscow, Russia

www.rdtex.ru

RDTEX
INFORMATION TECHNOLOGIES