




ASH Architecture and Advanced Usage: Part 1

John Beresniewicz, Graham Wood, Uri Shaft
Oracle America



The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



Agenda

Part 1:

- The ASH Mechanism
- DB Time Estimates From ASH
- Top Activity and ASH Analytics
- Avoiding Mistakes

Part 2:

- The ASH Fix-up Mechanism
- Event Count Estimates From ASH
- ASH Forensics: Latency Outlier Detection
- Dataviz: Wait Class Latency Bubble Chart



The ASH Mechanism



Motivation for ASH in 10i

Performance diagnostics tool

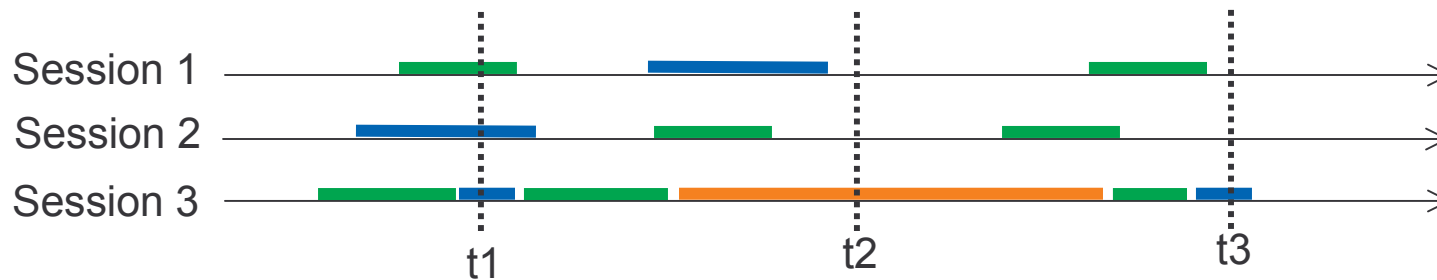
- Always on (unlike SQL trace)
- Keeps a history of what happened
- Keeps fine granularity of details

Other requirements:

- Efficient: very small performance penalty for using it
- Robust: no locks of any kind, won't stall
- Works well even when system is overloaded

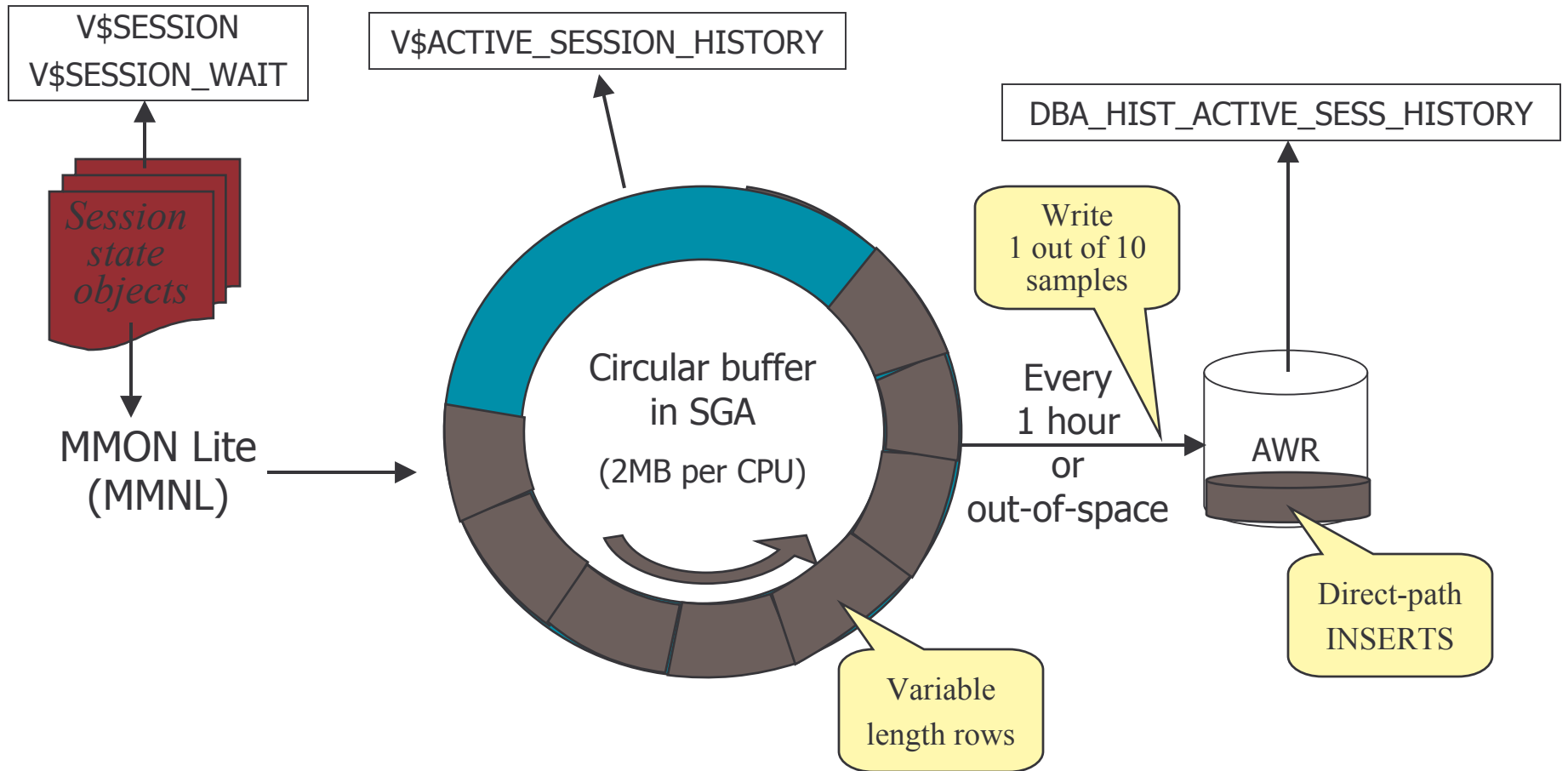
Active Session History (time based)

A sampling of session activity at regular intervals

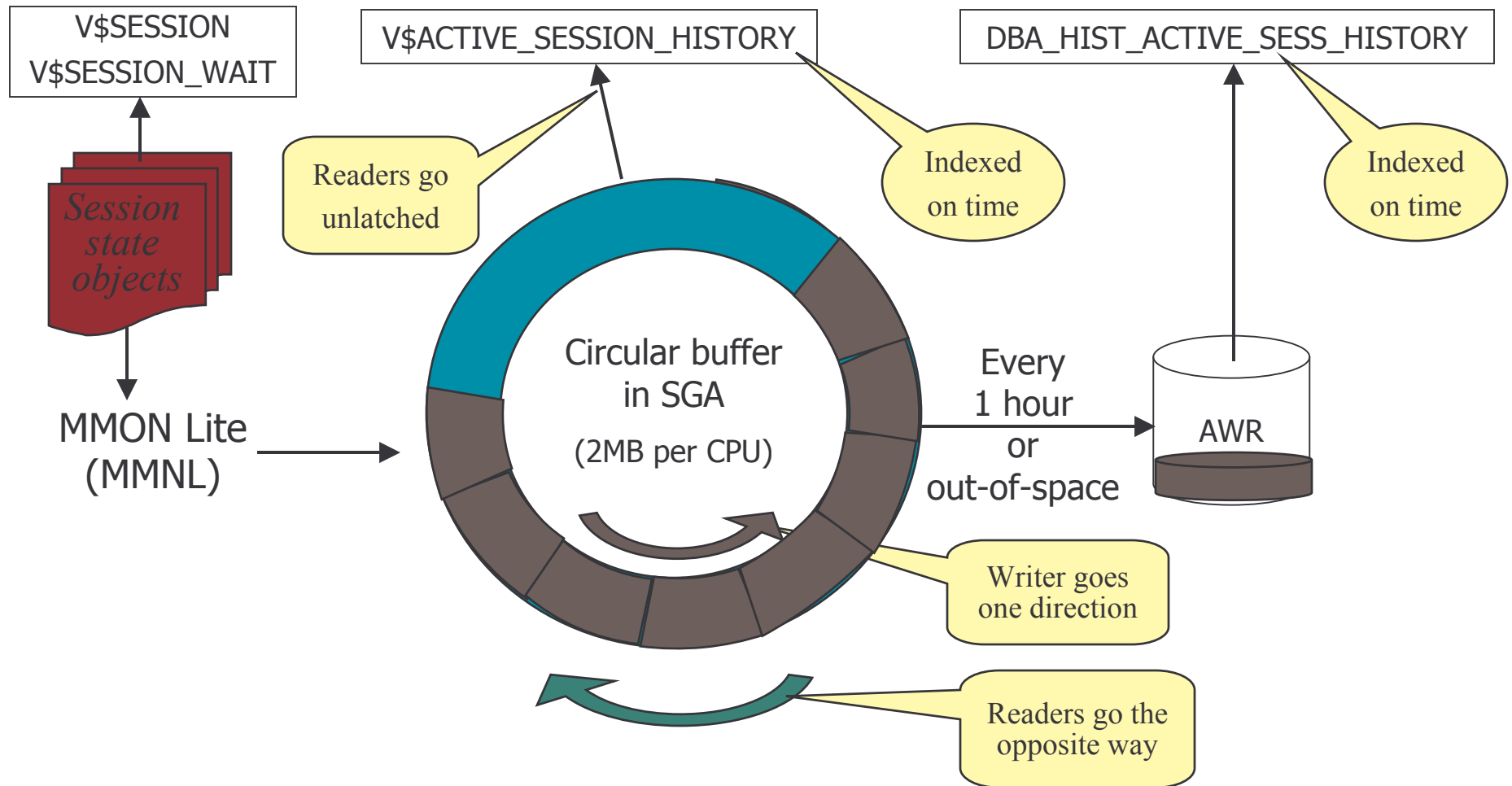


Session	Time	Duration	Activity	SQL	Object
1	t1	null	CPU	SQL1	null
2	t1	100ms	I/O	SQL2	EMP
3	t1	5ms	I/O	SQL1	EMP
3	t2	1sec	Row lock	SQL1	DEPT

ASH Architecture



ASH Architecture





Sampling Pseudo-code

1 FORALL SESSION STATE OBJECTS

2 IS SESSION CONNECTED?

NO => NEXT SESSION

YES:

3 IS SESSION ACTIVE?

NO => NEXT SESSION

YES:

4 MEMCPY SESSION STATE OBJ

5 CHECK CONSISTENCY OF COPY WITH LIVE SESSION

6 IS COPY CONSISTENT?

YES:

WRITE ASH ROW FROM COPY

NO:

IF FIRST COPY, REPEAT 4-6

ELSE => NEXT SESSION (NO ASH ROW WRITTEN)



Default Settings

- Sampling interval = 1000ms = 1 sec
- Disk filter ratio = 10 = 1 in 10 samples written to AWR
- ASH buffer size:
 - Min(Max (5% shared pool, 2% SGA), 2MB per CPU)
 - Absolute Max of 256MB

NOTE: the MMNL sampler session is not sampled

Control Parameters

- **_ash_size** : size of ASH buffer in bytes
 - K/M notation works (e.g. 200M)
- **_ash_sampling_interval** : in milliseconds
 - Min = 1, Max = 10,000
- **_ash_disk_filter_ratio** : every Nth sample to AWR
 - $\text{MOD}(\text{sample_id}, N) = 0$ where $N = \text{disk filter ratio}$
- **_sample_all** : samples idle and active sessions



V\$ASH_INFO (new in 11.2)

```
desc v$ash_info
```

Name	Null	Type
TOTAL_SIZE		NUMBER
FIXED_SIZE		NUMBER
SAMPLING_INTERVAL		NUMBER
OLDEST_SAMPLE_ID		NUMBER
OLDEST_SAMPLE_TIME		TIMESTAMP (9)
LATEST_SAMPLE_ID		NUMBER
LATEST_SAMPLE_TIME		TIMESTAMP (9)
SAMPLE_COUNT		NUMBER
SAMPLED_BYTES		NUMBER
SAMPLER_ELAPSED_TIME		NUMBER
DISK_FILTER_RATIO		NUMBER
AWR_FLUSH_BYTES		NUMBER
AWR_FLUSH_ELAPSED_TIME		NUMBER
AWR_FLUSH_COUNT		NUMBER
AWR_FLUSH_EMERGENCY_COUNT		NUMBER



ASH Dimensions

```
desc v$active_session_history
Name                               Null    Type
-----
SAMPLE_ID                           NUMBER
SAMPLE_TIME                         TIMESTAMP (3)
IS_AWR_SAMPLE                       VARCHAR2 (1)
SESSION_ID                           NUMBER
SESSION_SERIAL#                     NUMBER
SESSION_TYPE                       VARCHAR2 (10)
FLAGS                               NUMBER
USER_ID                             NUMBER
.
.
.
93 rows selected
```



SQL Dimensions

SQL Analysis

```
-----  
SQL_ID                                VARCHAR2 (13)  
IS_SQLID_CURRENT                       VARCHAR2 (1)  
SQL_CHILD_NUMBER                       NUMBER  
SQL_OPCODE                             NUMBER  
SQL_OPNAME                             VARCHAR2 (64)  
FORCE_MATCHING_SIGNATURE               NUMBER  
TOP_LEVEL_SQL_ID                       VARCHAR2 (13)  
TOP_LEVEL_SQL_OPCODE                   NUMBER  
SQL_PLAN_HASH_VALUE                    NUMBER  
SQL_PLAN_LINE_ID                       NUMBER  
SQL_PLAN_OPERATION                     VARCHAR2 (30)  
SQL_PLAN_OPTIONS                       VARCHAR2 (30)  
SQL_EXEC_ID                            NUMBER  
SQL_EXEC_START                         DATE  
PLSQL_ENTRY_OBJECT_ID                  NUMBER  
PLSQL_ENTRY_SUBPROGRAM_ID              NUMBER  
PLSQL_OBJECT_ID                        NUMBER  
PLSQL_SUBPROGRAM_ID                    NUMBER  
QC_INSTANCE_ID                         NUMBER  
QC_SESSION_ID                          NUMBER  
QC_SESSION_SERIAL#                     NUMBER
```



Evolution of SQL_ID

- 11.1.0.7+ it is the *currently executing SQL*
 - Except recursive server code SQL, triggers
- Previously there were various schemes to get the bottom user-level SQL
 - They all failed since it is very difficult to do.
- ENTRY_LEVEL columns added for applications with generic PL/SQL entry points and SQL underneath
 - Need to know what the application initially called



Wait Event Dimensions

Wait Event Analysis

EVENT	VARCHAR2 (64)
EVENT_ID	NUMBER
EVENT#	NUMBER
SEQ#	NUMBER
P1TEXT	VARCHAR2 (64)
P1	NUMBER
P2TEXT	VARCHAR2 (64)
P2	NUMBER
P3TEXT	VARCHAR2 (64)
P3	NUMBER
WAIT_CLASS	VARCHAR2 (64)
WAIT_CLASS_ID	NUMBER
WAIT_TIME	NUMBER
SESSION_STATE	VARCHAR2 (7)
TIME_WAITED	NUMBER



Blocking and Object Dimensions

Locking/Blocking Analysis

BLOCKING_SESSION_STATUS	VARCHAR2 (11)
BLOCKING_SESSION	NUMBER
BLOCKING_SESSION_SERIAL#	NUMBER
BLOCKING_INST_ID	NUMBER
BLOCKING_HANGCHAIN_INFO	VARCHAR2 (1)

Object Analysis

CURRENT_OBJ#	NUMBER
CURRENT_FILE#	NUMBER
CURRENT_BLOCK#	NUMBER
CURRENT_ROW#	NUMBER



Blocking Session Info

BLOCKING_SESSION_STATUS
BLOCKING_SESSION
BLOCKING_SESSION_SERIAL#
BLOCKING_INST_ID

Prior to 11.2:

- Finding blocking session could induce locking or crashes
- Therefore blocking session only in same instance, in some cases

From 11.2:

- The pre-11.2 path is taken for short local blocking
- Long blocking events (3 seconds in same instance, 10 seconds in RAC) get accurate hang information copied into ASH
- Information comes from Hang Manager (working behind the scenes)



What Is CURRENT_OBJ#

- DBA_OBJECTS.DATA_OBJECT_ID of the segment operated on when ASH is sampled
- It is only valid in specific wait events:
 - I/O events on data blocks
 - Cluster (global cache) events on data blocks
 - Row Locks
 - Table Locks
 - Buffer busy (and associated RAC events)

DANGER: the column is not cleared by session after waiting



Application Dimensions

Application Dimensions

SERVICE_HASH	NUMBER
PROGRAM	VARCHAR2 (48)
MODULE	VARCHAR2 (48)
ACTION	VARCHAR2 (32)
CLIENT_ID	VARCHAR2 (64)
MACHINE	VARCHAR2 (64)
PORT	NUMBER
ECID	VARCHAR2 (64)
CONSUMER_GROUP_ID	NUMBER
TOP_LEVEL_CALL#	NUMBER
TOP_LEVEL_CALL_NAME	VARCHAR2 (64)
CONSUMER_GROUP_ID	NUMBER
XID	RAW (8)
REMOTE_INSTANCE#	NUMBER
TIME_MODEL	NUMBER



Session Statistics (SQL Monitoring)

Session Statistics

TM_DELTA_TIME	NUMBER
TM_DELTA_CPU_TIME	NUMBER
TM_DELTA_DB_TIME	NUMBER
DELTA_TIME	NUMBER
DELTA_READ_IO_REQUESTS	NUMBER
DELTA_WRITE_IO_REQUESTS	NUMBER
DELTA_READ_IO_BYTES	NUMBER
DELTA_WRITE_IO_BYTES	NUMBER
DELTA_INTERCONNECT_IO_BYTES	NUMBER
PGA_ALLOCATED	NUMBER
TEMP_SPACE_ALLOCATED	NUMBER



Bit Vector Dimensions

Bitvec and Replay

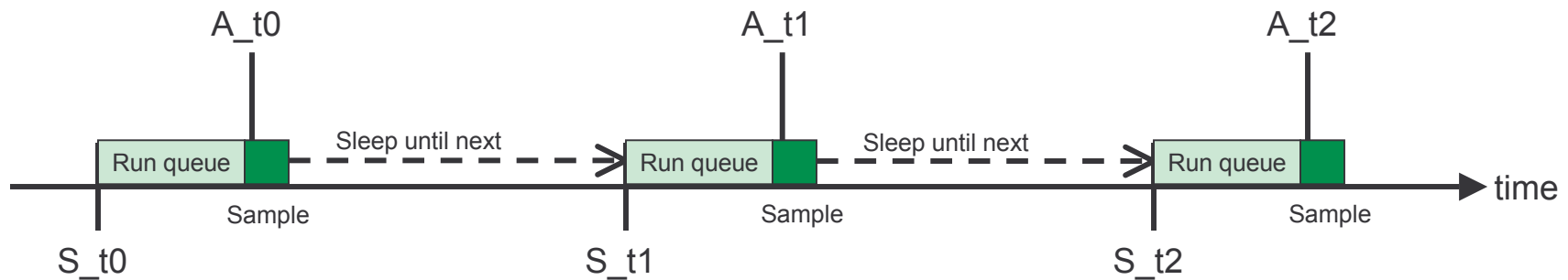
IN_CONNECTION_MGMT	VARCHAR2 (1)
IN_PARSE	VARCHAR2 (1)
IN_HARD_PARSE	VARCHAR2 (1)
IN_SQL_EXECUTION	VARCHAR2 (1)
IN_PLSQL_EXECUTION	VARCHAR2 (1)
IN_PLSQL_RPC	VARCHAR2 (1)
IN_PLSQL_COMPILATION	VARCHAR2 (1)
IN_JAVA_EXECUTION	VARCHAR2 (1)
IN_BIND	VARCHAR2 (1)
IN_CURSOR_CLOSE	VARCHAR2 (1)
IN_SEQUENCE_LOAD	VARCHAR2 (1)
CAPTURE_OVERHEAD	VARCHAR2 (1)
REPLAY_OVERHEAD	VARCHAR2 (1)
IS_CAPTURED	VARCHAR2 (1)
IS_REPLAYED	VARCHAR2 (1)



Why does ASH work when the server is CPU bound?

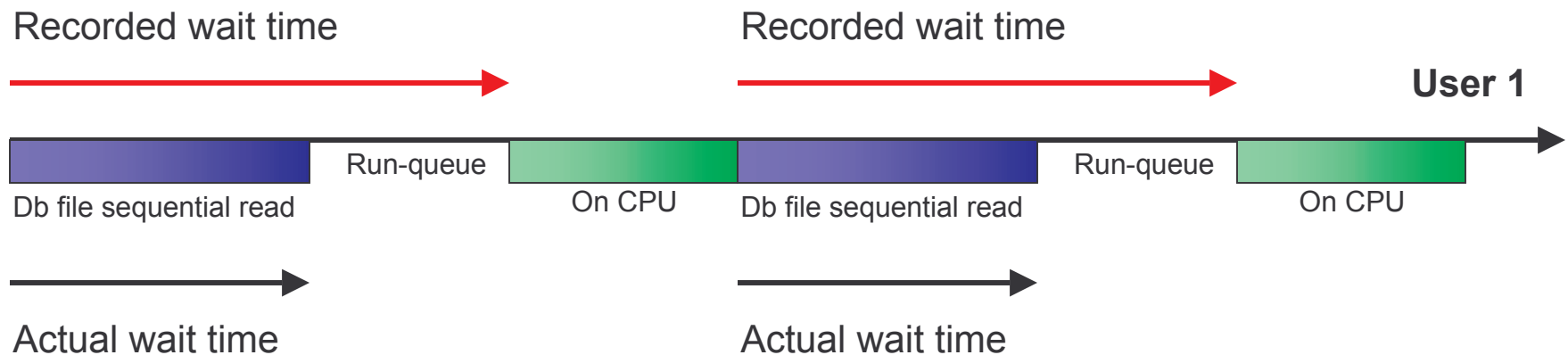
1. ASH sampler is very efficient and does not lock
 - Should complete a sample within a single CPU slice
2. After sampling, the sampler computes next scheduled sample time and sleeps until then
3. Upon scheduled wake-up, it waits for CPU (runq) and samples again
 - CPU bound sample times are shifted by one runq but intervals stay close to 1 second

ASH Sampler and Run-queue



If run queue times are consistent sampling interval will be preserved but sample times shifted

CPU Run-queue and Wait Latencies



Event wait time is inflated when host is CPU-bound, so ASH will sample more/longer waits



'ON CPU' and ASH

- ASH session status 'ON CPU' derived, not observed
 - Session is in a database call
 - Session is NOT in a wait event (idle or non-idle)
- Un-instrumented waits => 'ON CPU'
 - These are bugs and should be rare, but have happened
- Session on run queue may be 'WAITING' or 'ON CPU'
 - Depends on state prior to going onto run queue



ASH Pros and Cons

PROS

- Supports the DB Time method of performance analysis
 - Historically (AWR) for large or systemic problems
 - Recent (V\$) for emergency or monitoring
- Always available (modulo licensing restrictions)
- Minimal cost to server performance

CONS

- Queries become estimates
 - We query a sample and not the full data set
- **Difficult to form semantically meaningful queries**
 - Probability distribution of a query result matters



DB Time Estimates From ASH



Database Time and Active Sessions

- **Database Time** = Total time spent by sessions in the database server actively working (on CPU) or actively waiting (non-idle wait)
- **Active Sessions** = The number of sessions active (working or waiting) in the database server at a particular time
- **Average Active Sessions** = $\text{DB Time} / \text{Elapsed Time}$



ASH and Database Time

$$\text{DBT} \approx :T \bullet \sum S_i$$

DB Time is approximated by multiplying sampling interval (:T) by the total count of the samples

$$\text{DBT} \approx :T \bullet (\text{rows in ASH})$$

Each ASH row represents 1 :T slice of DB Time in the units of :T



Basic ASH Math

COUNT(*) = DB Time (secs)

GROUP BY ?

Where samples are counted over some [time interval]



Non-default ASH Math

- For V\$ACTIVE_SESSION_HISTORY:

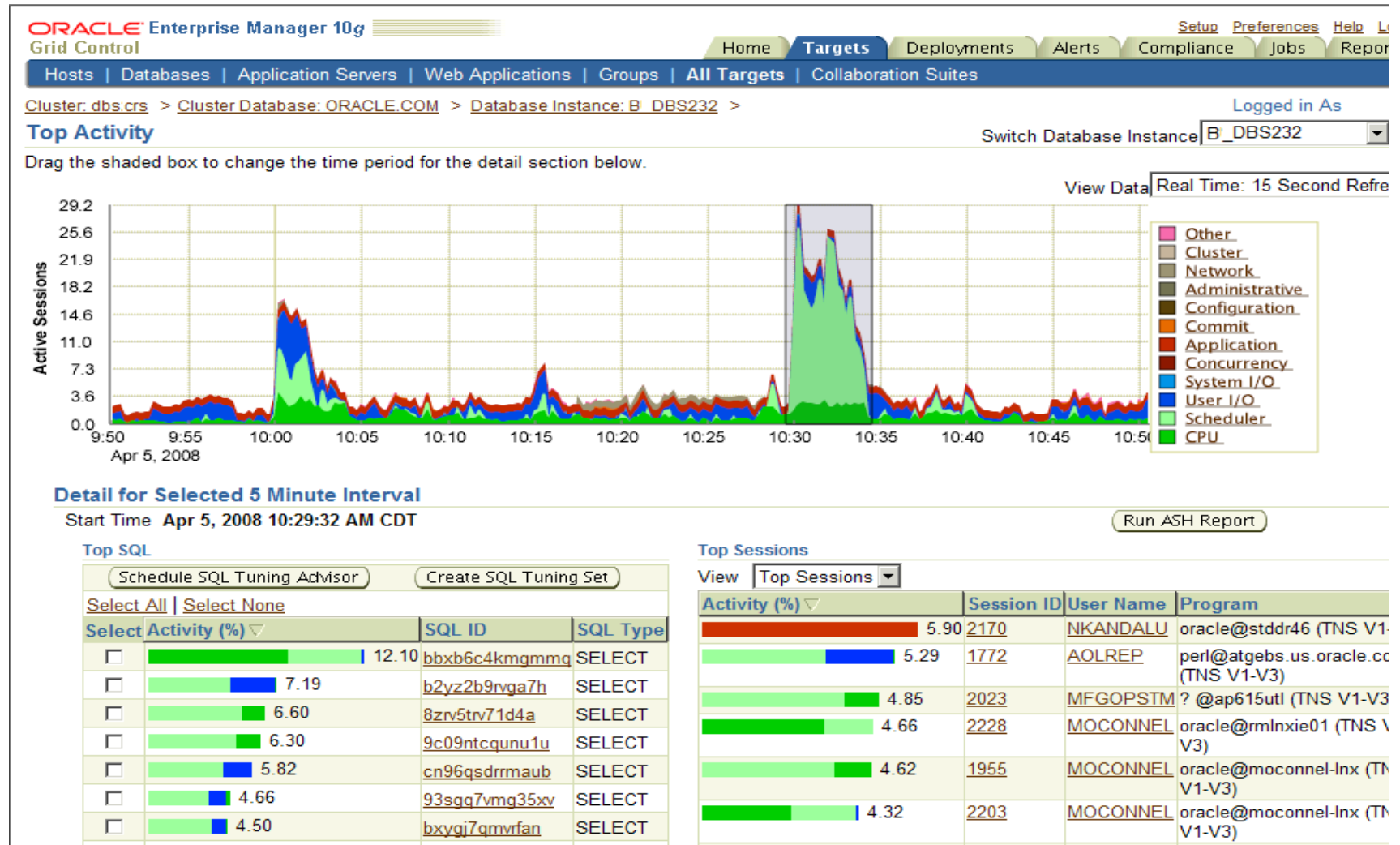
$$\text{SUM}(:T/1000) = \text{DB Time (secs)}$$

- For DBA_HIST_ACTIVE_SESS_HISTORY:

$$:F \times \text{SUM}(:T/1000) = \text{DB Time (secs)}$$

Where :T is sampling interval in millisecs and :F is the disk filter ratio

ASH Math: EM Top Activity

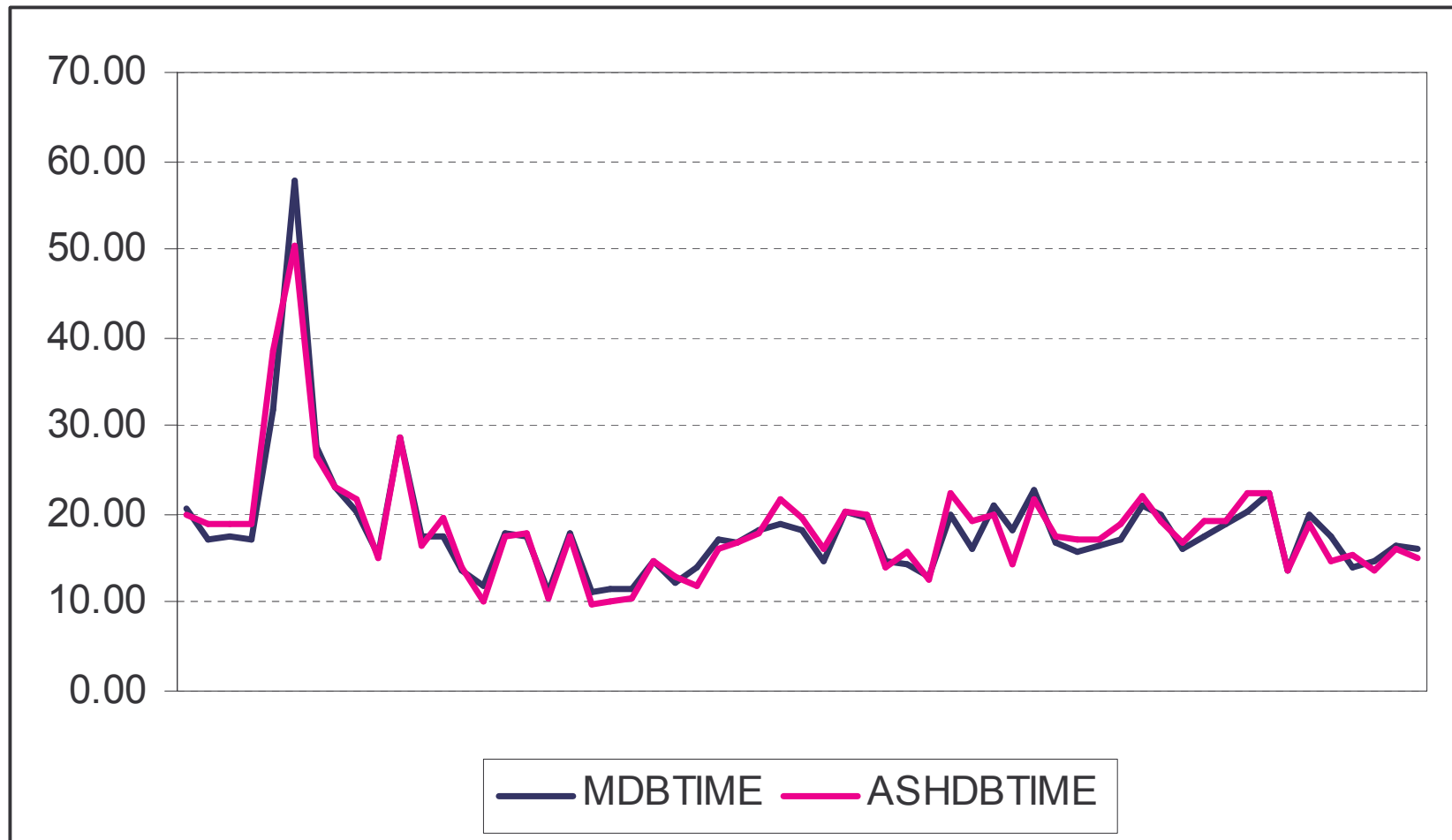




Compare ASH and Time Model

```
select M.end_time
       ,ROUND(M.value / 100,3) as Metric_AAS
       ,ROUND(SUM(DECODE(A.session_type,'BACKGROUND',1,0)) /
              ((M.end_time - M.begin_time) * 86400 ),3)
           as ASH_AAS
       ,COUNT(1)   as ASH_count
from
       v$active_session_history  A
       ,v$sysmetric_history      M
where
       A.sample_time between M.begin_time and M.end_time
       and M.metric_name = 'Database Time Per Sec' -- 10g metric
       and M.group_id = 2
group by M.end_time,M.begin_time, M.value
order by M.end_time;
```

Compare ASH and Time Model



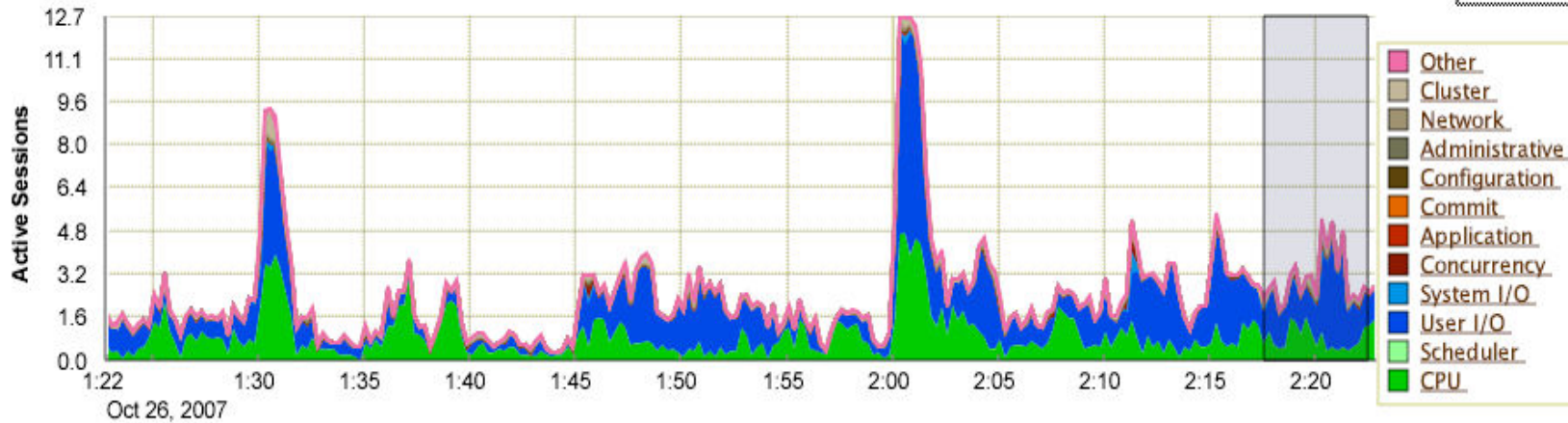
Compare Estimated to Actual DB Time

Top Activity

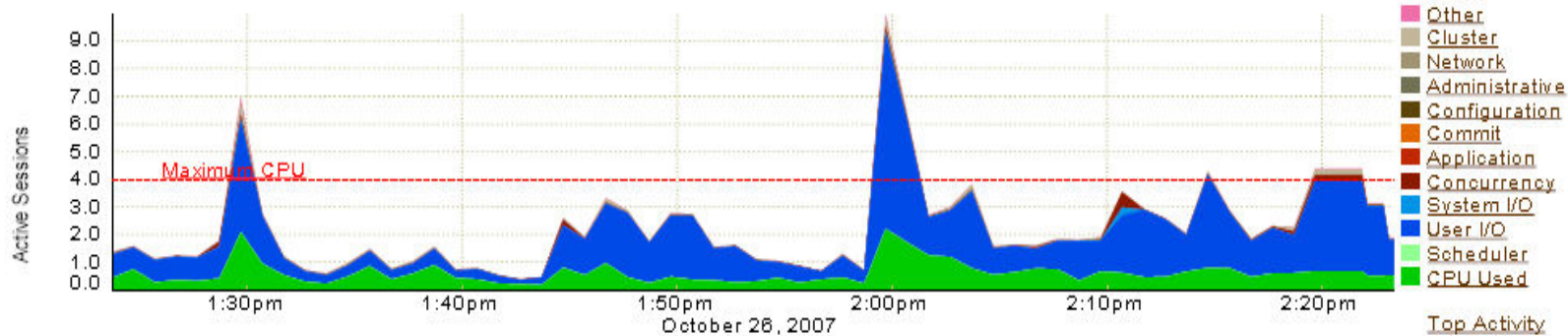
Switch Database Instance

Drag the shaded box to change the time period for the detail section below.

View Data

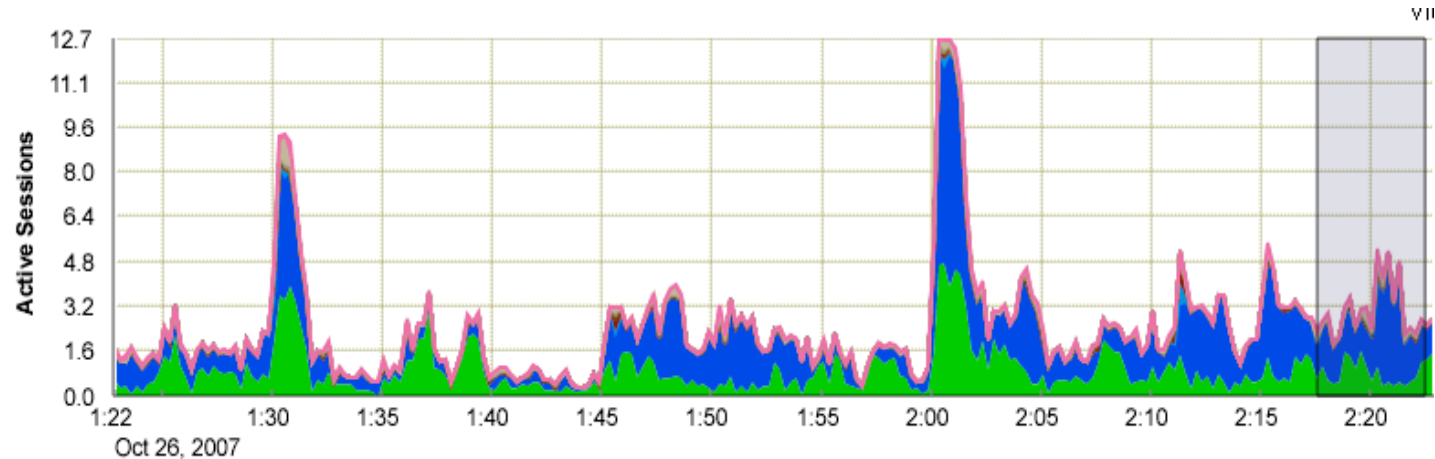


Average Active Sessions



Instance Disk I/O

Understanding Active Session Charts



- Chart of Average Active Sessions over time
 - Broken down by Wait Class
 - Green = CPU, Blue = I/O
- Area under curve = amount of DB Time
- Usage Model: “Click on the big stuff”



Active Sessions and DB Time

- The number of active sessions at any time is the rate of change of the “DB Time function” at that time

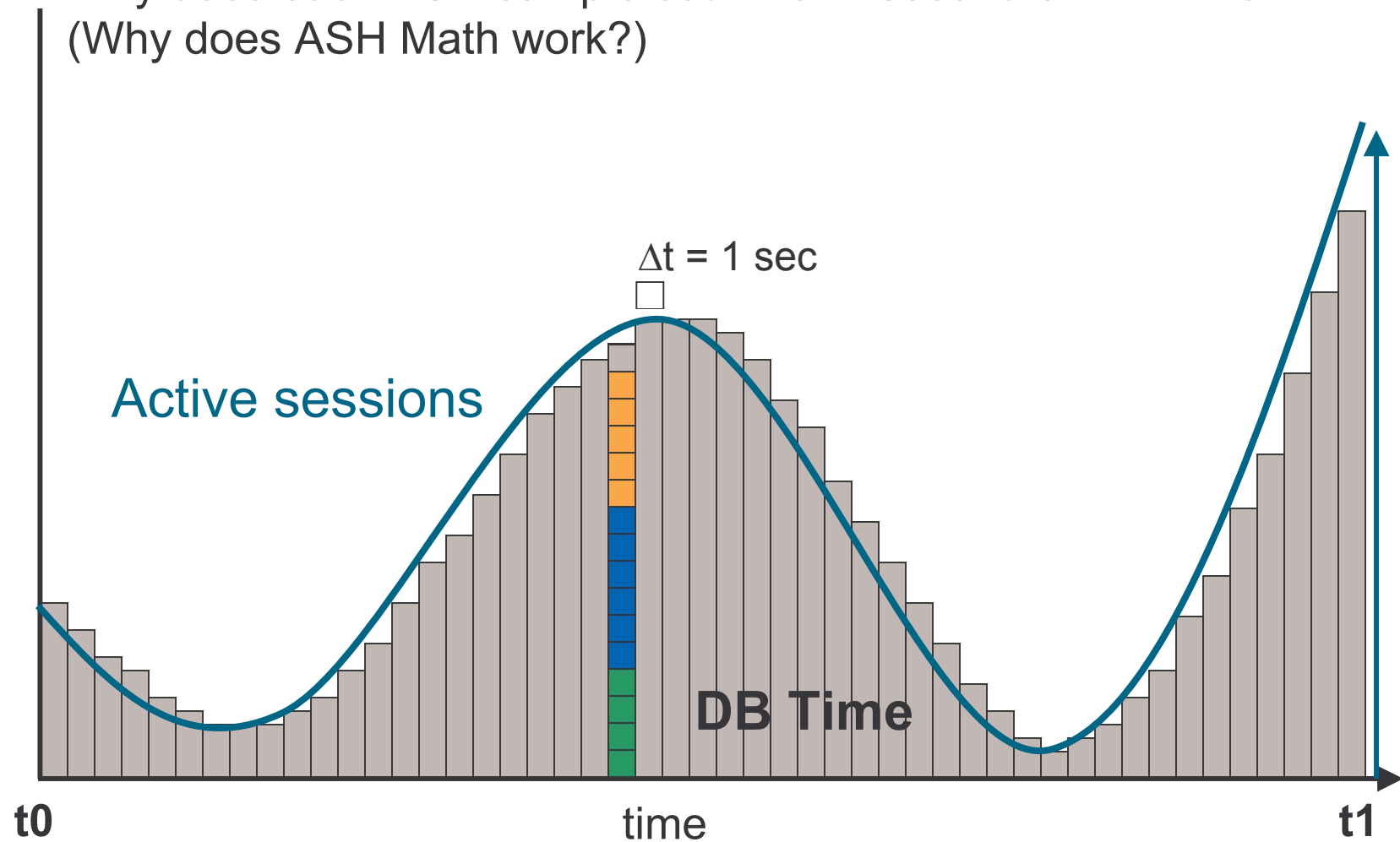
$$\delta DBtime / \delta t = ActiveSessions$$

- DB Time is the integral of the Active Session function

$$DBtime = \int_{t_0}^{t_1} ActiveSessions$$

Active Sessions and DB Time

Why does each ASH sample count for 1 second of DB Time?
(Why does ASH Math work?)



ASH DB Time Query (basic ASH math)

- What are the **top SQL by database time** in the last 5 minutes?

```
SELECT * FROM
  (SELECT NVL(SQL_ID, 'NULL') as SQL_ID
        ,SUM(1)                as DBtime_secs
    FROM V$ACTIVE_SESSION_HISTORY
   WHERE sample_time > SYSDATE - 5/1440
   GROUP BY SQL_ID
   ORDER BY 2 DESC
  )
WHERE rownum < 6;
```

SUM(:T/1000) must be substituted for SUM(1) to account for non-default sampling

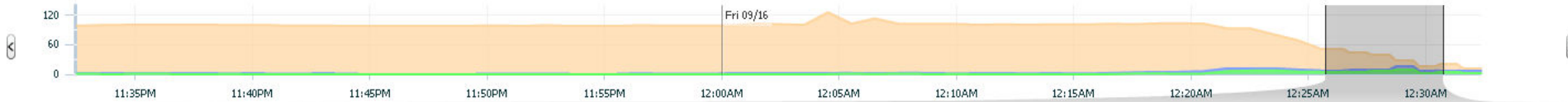


ASH Math: ASH Analytics

ASH Analytics

Save Mail Full Screen Refresh 15 seconds Stop Refresh

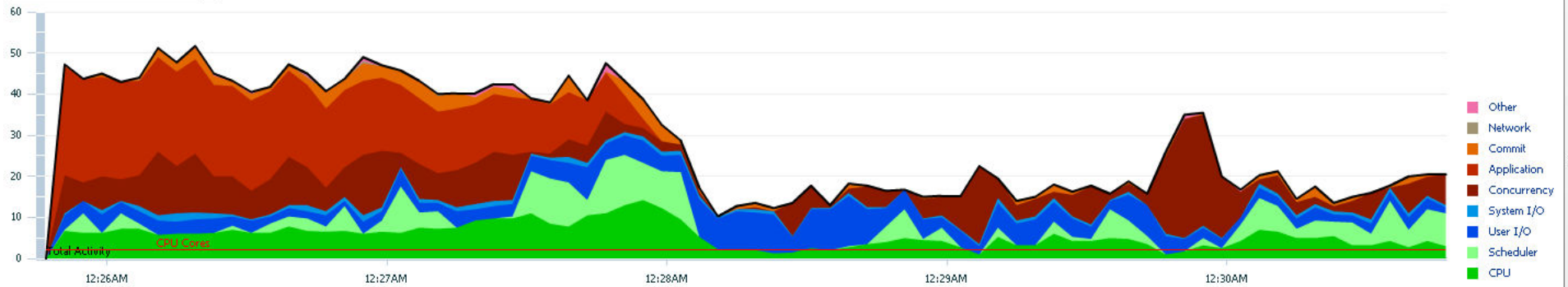
Hour Day Week Month Max Calendar Custom



Filters None

Activity Load Map

Wait Class Show Total Activity CPU Cores



SQL ID by Wait Class Schedule SQL Tuning Advisor Create SQL Tuning Set

Select	SQL ID	Activity (Average Active Sessions)
<input type="checkbox"/>	fd042b5v0y25t	7.56
<input type="checkbox"/>	9q7k9nbpvk8pv	1
<input type="checkbox"/>	632qqfpdwh0bc	.99
<input type="checkbox"/>	3sm97>9kdzyhn	.95
<input type="checkbox"/>	64fgf2dfr9f6n	.8

User Session by Wait Class

User Session	Activity (Average Active Sessions)
1:1272,46823	1
1:1203,32745	.77
1:1258,16979	.72
1:1179,62907	.69
1:42,10989	.65

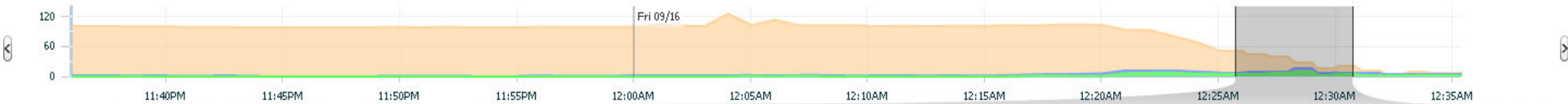


Multi-dimensional ASH Math

ASH Analytics

Save Mail Full Screen Refresh 15 seconds Stop Refresh

Hour Day Week Month Max Calendar Custom



Filters None

Activity Load Map
Mode: Advanced Wait Class: Wait Event: SQL ID

Application	User I/O	Concurrency	CPU	Scheduler
eng: TX - row lock contention	db file sequential read	latch: shared pool	CPU + Wait for CPU	resmgr:cpu quantum
fd42bsv8y25t	Others 9q7k9nbpvk8pv 64fgf2drf9f6n 14fks3ympphn 43kc2 cvn5 26xbr 9wy 632qqfp awk3 btxy	Others fas26aqfudzff 4a7blag2vx 4p3r 3sm ny0g 97x9 288m dnv fzbu6 8h53 agk0 3hs9	Others 3sm97x9kdzyhn 3curfrq crzpkk fas26aq fudaff 3mwr01 qttp4c2	Others 9q7k9nbpvk8pv 14fks3ympphn 5ptjh5mu3g0k 64fgf2drf9f6n 4p3rny0g5b 632qqfpdwh0tx a9t927jd agvtt 7k4hpq97
	read by other session 64fgf2drf9f6n 632qqfpdwh0tx	cursor: pin S wait on X cursor: pin S Others 57rksw9931w6h dfjx 3sm97 x9kdz 2m	library cache: 3m97x9kdzyhn 3curfrq crzpkk fas26aq fudaff 3mwr01 qttp4c2	Commit log file sync Other latch free 3sm97x9kdzyhn 4y
	db file parallel read 099wtpc2 7n553 23j60 6nxaj x0g dg21rbss 14fks3ympphn 9q7k9nbpvk8pv 2m8c3 0t9w	library cache loc buffer b latch: row c Others 23j606 fykb 4471 3sm9 28k4 7x9k 74 Dwx8	latch: row c 57xckg84505w7 8h53vj0t9qha9 3m97x9kdzyhn 3curfrq crzpkk fas26aq fudaff 3mwr01 qttp4c2 5ptjh5mu3g0k 7vhbt gw9bf 810 2r68zg fag7qb9	Commit log file sync Other latch free 3sm97x9kdzyhn 4y

Dimensions 1 2 3 Sample Size 1 2258



“Click on the big stuff”

- EM usage model for DB Performance
 - “click on the largest integral estimate of DB Time”
- The size of colored areas corresponds directly to the important quantity
- ASH Math 3 Dim combinations $\sim 90^3 = 729,000$



ASH Timing for Nano-operations

- Some important operations are still too frequent and short-lived for timing
 - No wait event for “bind” operations
- A session-level bit vector is updated in binary fashion before/after such operations
 - Much cheaper than timer call
- The session bit vector is sampled into ASH
- “ASH Math” allows us to estimate time spent in these un-timed transient operations



ASH Queries: Estimates Over Samples

- ASH does not contain all activity data, just a sample
 - Multiple mechanisms would produce different results from same true activity
- Aggregates over ASH data are random variables
 - They estimate some property of the true activity
 - **Unbiased estimator** means the expected value = true property value
- Time sampling means ASH event samples are biased over `TIME_WAITED`
 - 100ms event 10x more likely to sample than 10ms event



Sample Size Matters

- Statistics over larger data sets have more strength
 - Increases confidence that sample represents reality
 - Rigorous sample sizing discussion is beyond scope
- ASH aggregates over fewer rows lose strength
 - Beware of drawing conclusions from highly filtered data
- EM and AWR defaults were chosen carefully:
 - 5-minute Top Activity detail window is 300 V\$ samples
 - 30-minute historical mode is 180 AWR samples



Common Mistakes (Bad ASH Math)

~~SUM(TIME_WAITED)~~

This does not compute total wait time in the database since ASH does not sample all waits

~~AVG(TIME_WAITED)~~

This does not estimate the average event latencies because of sampling bias toward longer events

ASH is neither a random sample nor a complete sample of TIME_WAITED by session events






ORACLE IS THE INFORMATION COMPANY



ASH Architecture and Advanced Usage: Part 2

John Beresniewicz, Graham Wood, Uri Shaft
Oracle America



The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



Agenda

Part 1:

- The ASH Mechanism
- DB Time Estimates From ASH
- Top Activity and ASH Analytics
- Avoiding Mistakes

Part 2:

- The ASH Fix-up Mechanism
- Event Count Estimates From ASH
- ASH Forensics: Latency Outlier Detection
- Dataviz: Wait Class Latency Bubble Chart

Bad ASH Math

- SQL observed using 9 secs of CPU every 10 secs

ORACLE Enterprise Manager Cloud Control 12c

Enterprise Targets Favorites History

lin22.us.oracle.com

Oracle Database Performance Availability Schema Administration

Top Activity > SQL Details: 9q7k9nbpvk8pv

SQL Details: 9q7k9nbpvk8pv

Switch to SQL ID Go View Data Real Time: Manual Refresh Refresh SQL Worl

Text

```
SELECT NVL(SUM(TIME_WAITED/1000), 0)
FROM SYS.DBA HIST ACTIVE SESS HISTORY S, SYS.WRH$_SEG_STAT_OBJ SO, SYS.WRM$_SNAPSHOT SN , SYS.OBJ$ OE
WHERE SN.SNAP_ID = S.SNAP_ID AND SN.DBID = S.DBID AND SN.BEGIN_INTERVAL_TIME >= (CURRENT_TIMESTAMP -
AND S.WAIT_CLASS = 'User I/O' AND S.WAIT_TIME = 0 AND SO.DATAOBJ# = :B1 AND OBJ.OBJ# = SO.OBJ# AND OE
```

Details

Select the plan hash value to see the details below. Plan Hash Value 2106978214

Statistics Activity Plan Plan Control Tuning History SQL Monitoring

Summary



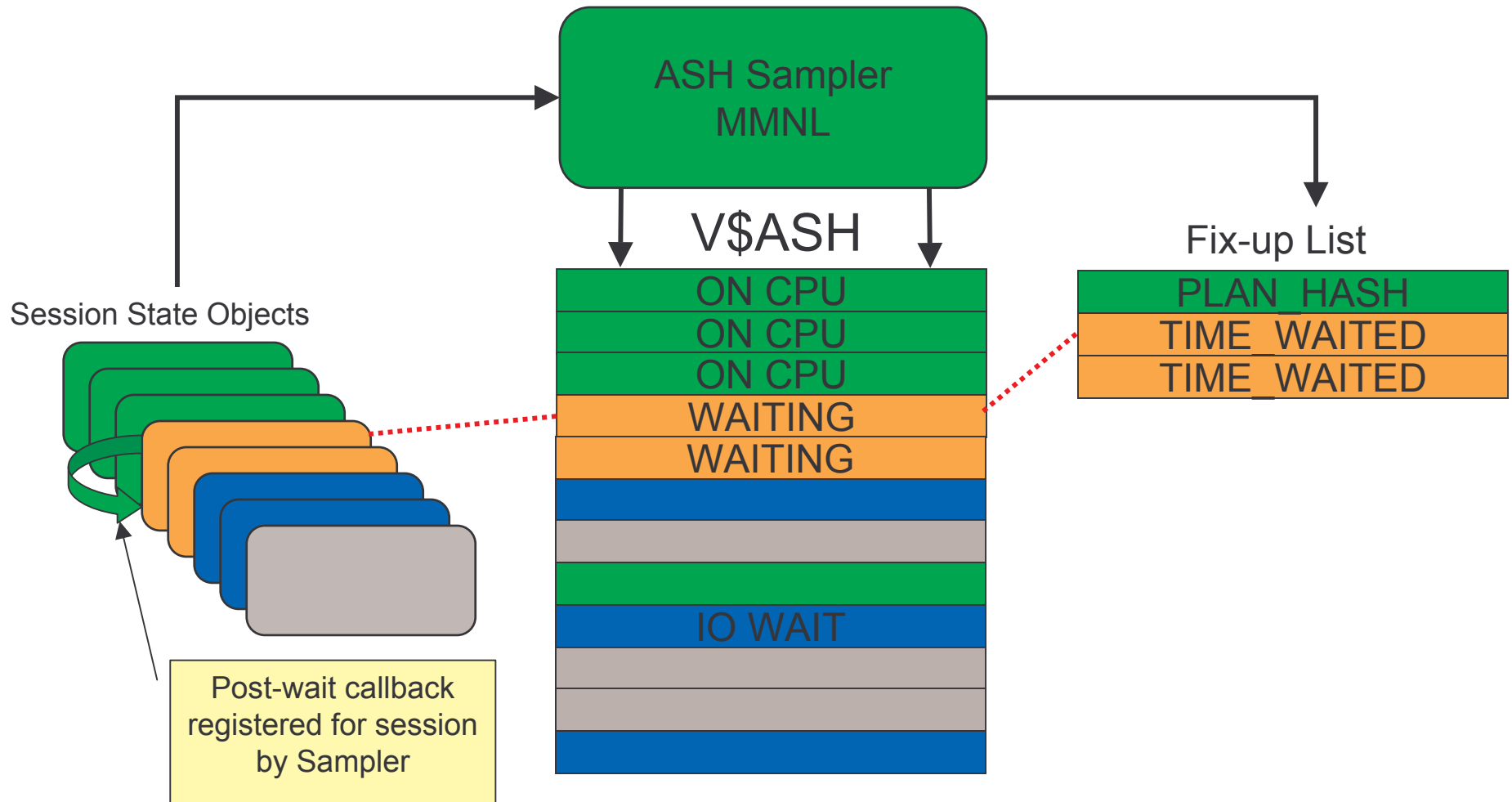
The ASH Fix-up Mechanism



The ASH “Fix-up”

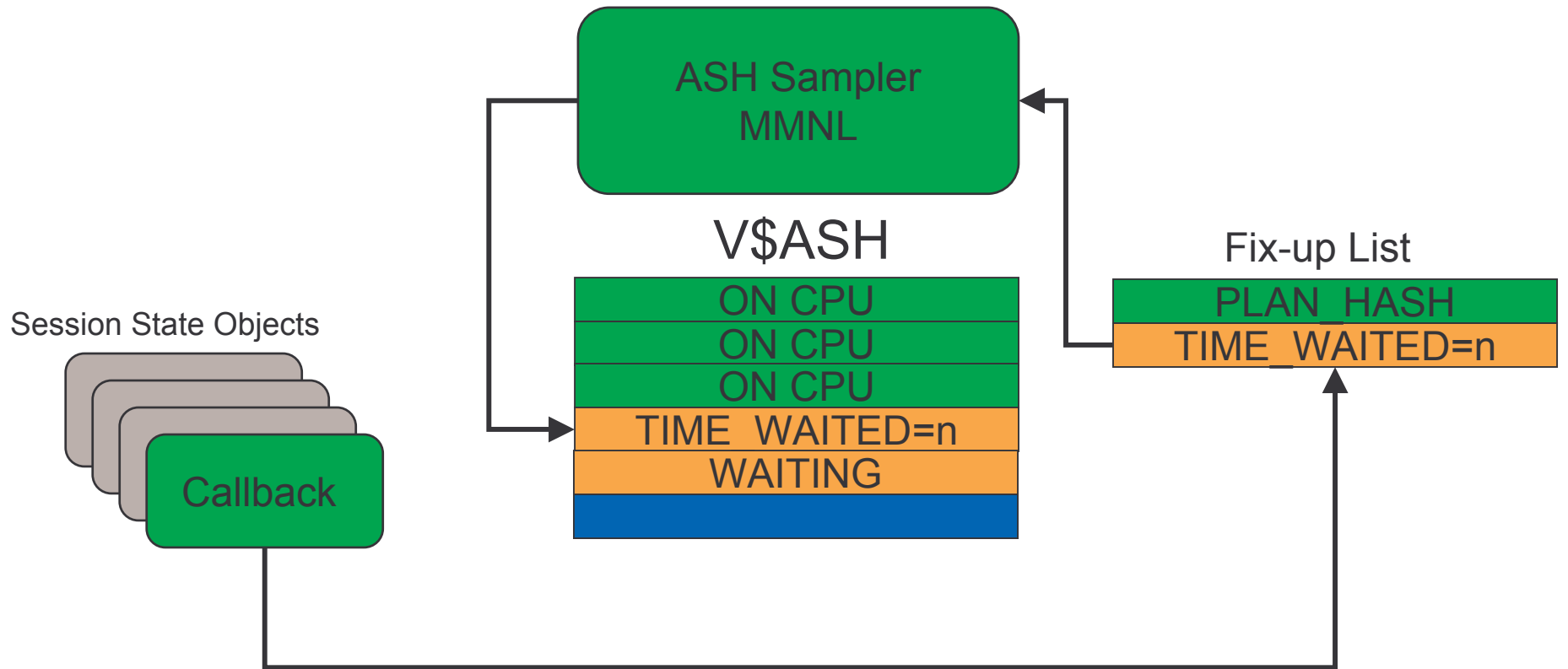
- ASH columns may be unknown at sampling time
 - TIME_WAITED: session is still waiting
 - PLAN_HASH: session is still optimizing SQL
 - GC events: event details unknown at event initiation
 - Certain time model bit vector columns
- ASH “fixes up” data during subsequent sampling
 - TIME_WAITED fixed up in first sample after event completes
 - Long events: last sample gets fixed up time_waited (all others stay 0)
- Querying current ASH may return un-fixed rows
 - Should not be a problem generally

ASH Fix-up 1: Sampling





ASH Fix-up 2: Fixing Up





Event Count Estimates From ASH



Correcting Bad ASH Math

- Why is `AVG(TIME_WAITED)` so commonly seen?
- Need to identify average latencies for events
 - Has a direct impact on performance
 - Some latencies originate externally (e.g. I/O)
- Turns out `TIME_WAITED` can be used in latency estimates
 - (it's just not that simple)

Estimating Event Counts With ASH

:T = ASH sampling interval (**millisec**)

TIME_WAITED = length of sampled event (**microsec**)

N = Number of events the ASH sample represents

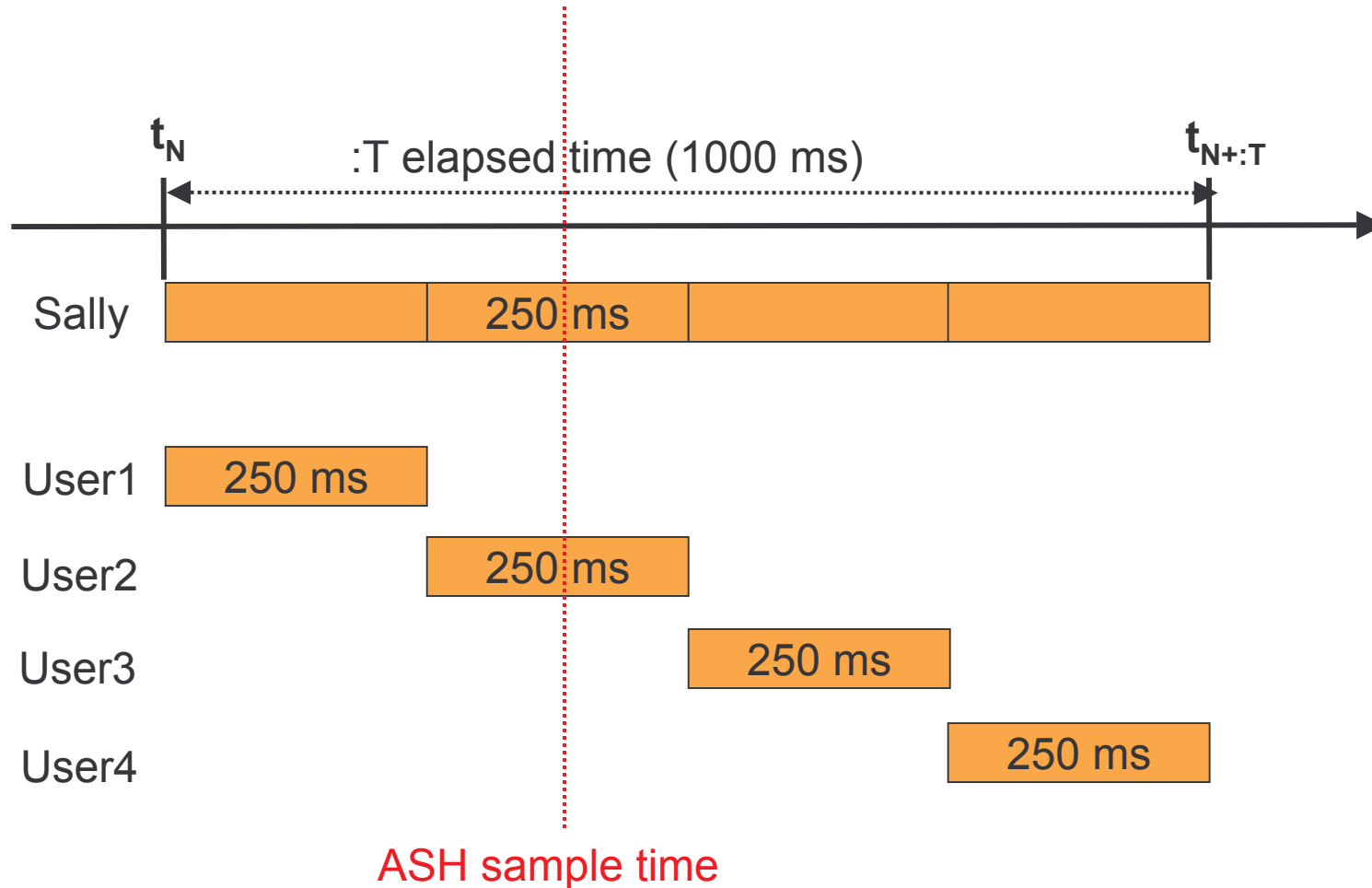
$$:T \sim \text{TIME_WAITED} / 1000 * N$$

OR

$$N \sim :T * 1000 / \text{TIME_WAITED}$$

Matching time units correctly is critical to correctness

Why is Count ~ :T / Time_Waited?



Sally's row in ASH estimates $:T/250 \text{ ms} = 4$ waits during time interval t_N to t_{N+T}

ASH Event Count Query (1)

- Top 5 objects by I/O requests last 15 minutes

```
SELECT W.*,O.object_name FROM
(SELECT current_obj#
        ,ROUND(SUM(1000000/time_waited)) as est_IOWaits
 FROM   V$ACTIVE_SESSION_HISTORY
 WHERE  sample_time > SYSDATE - 15/1440
        AND time_waited > 0 -- fixed up samples only
        AND event IN ('db file sequential read'
                      , 'db file scattered read')
 GROUP BY current_obj#
 ORDER BY 2 DESC
 ) W
, DBA_OBJECTS O
 WHERE o.data_object_id = W.current_obj#
       AND ROWNUM < 6;
```

This query works when all waits are “short”



Long Events

- Events that are longer than :T are always sampled
 - No sampler “bias” for these
 - Event may be sampled multiple times (e.g. 3 second enqueue)
- Only final (“fixed-up”) ASH row for long events has `TIME_WAITED > 0`
- The event count of long events is known:
 - 1 for each row with `TIME_WAITED >= :T`
- The DB Time estimate is still `COUNT(*)`

ASH Event Count Query (2)

- Top I/O event objects adjusted for long events

```
SELECT * FROM
(SELECT
  current_obj#
  ,ROUND(SUM(CASE WHEN time_waited >= 1000000 THEN 1
                ELSE (1000000/time_waited)
                END))          as est_IOWaits
  FROM V$ACTIVE_SESSION_HISTORY
 WHERE sample_time > SYSDATE - 15/1440
       AND time_waited > 0
       AND wait_class = 'User I/O'
 GROUP BY current_obj#
 ORDER BY 2 DESC
)
WHERE ROWNUM < 6;
```




Estimating Event Latencies

1. We estimate total DB time on events
2. We can now estimate total event occurrences
3. Therefore, we can compute average latency:

$$\text{Est_avg_latency_ms} = \text{est_Dbtime_ms} / \text{est_waits}$$



User I/O Event Latencies (1)

```
SELECT event
       ,ROUND(est_DBtime_ms/est_waits,1)
           as est_avg_latency_ms
FROM
  (SELECT event
       ,ROUND(SUM(GREATEST(1, 1000000/time_waited)))
           as est_waits
       ,SUM(GREATEST(1000, time_waited/1000))
           as est_DBtime_ms
   FROM V$ACTIVE_SESSION_HISTORY
  WHERE sample_time > SYSDATE - 15/1440
        AND time_waited > 0      -- fixed-up events only
        AND wait_class = 'User I/O'
   GROUP BY event
  )
ORDER BY 2 DESC;
```

User I/O Event Latencies (2)

```
SELECT event
       ,ROUND(est_DBtime_ms/est_waits,1)
           as est_avg_latency_ms
FROM
  (SELECT event
       ,ROUND(SUM(CASE WHEN time_waited >0
                     THEN GREATEST(1, 1000000/time_waited)
                     ELSE 0 END ))      as est_waits
       ,SUM(1000)                        as est_DBtime_ms
  FROM V$ACTIVE_SESSION_HISTORY
 WHERE sample_time > SYSDATE - 15/1440
       -- NOTE: all samples, no time_waited>0
       AND wait_class = 'User I/O'
  GROUP BY event
  )
ORDER BY 2 DESC;
```



DEMO?

**SOFTWARE.
HARDWARE.
COMPLETE.**

ORACLE



Lessons Learned (so far)

- ASH is a time-based and not event-based sample of database activity
- ASH is an excellent representation of activity history
- How to estimate and rank DB Time spent over ASH dimensions using basic ASH Math
- ASH estimates underlie key DB Time performance analysis use cases exposed by EM
- The ASH fix-up is a critical (and unique) mechanism
- How to estimate event counts and latencies using `TIME_WAITED`



ASH Forensics: Latency Outlier Detection



ASH Forensics

Forensics: scientific analysis of physical evidence (as from a crime scene)

- Using ASH to understand and/or diagnose very specific incidents (crimes) that have happened
- Usually very tightly focused by time or session or both
- This is probably what you are doing with ASH now
 - ASH Report can be narrowly scoped for forensic analysis
- We will discuss a more advanced use case



Preliminary Caution

- ASH rows are based on latchless (and therefore possibly dirty) reads of session attributes
 - There is negligible (but nonzero) chance of within-row inconsistency
 - There is also some chance of sample-level inconsistency
- Looking for “outliers” and other individual ASH rows may be subject to these risks
- The risks are very small, but if you see something in ASH that seems unbelievable, it may just be wrong



Motivating Use Case

- Unusually long wait events (outliers) suspected to trigger cascade-effect performance incidents
- RAC performance experts claim EM Top Activity not helpful as aggregation masks outlier events
- Can we see if ASH has sampled any such events?
 - If none observed does not mean they have not happened
 - ASH sampling is biased toward longer events



V\$EVENT_HISTOGRAM

- Histogram buckets of event wait times
- Captures statistical distribution of wait times
- All events since instance startup counted in some bucket
- Exponential time bucketing scheme captures long-tail distributions efficiently



V\$EVENT_HISTOGRAM

```
SQL> desc v$event_histogram
```

Name	Type
-----	-----
EVENT#	NUMBER
EVENT	VARCHAR2 (64)
WAIT_TIME_MILLI	NUMBER
WAIT_COUNT	NUMBER
LAST_UPDATE_TIME	VARCHAR2 (64)



I/O Event Histogram

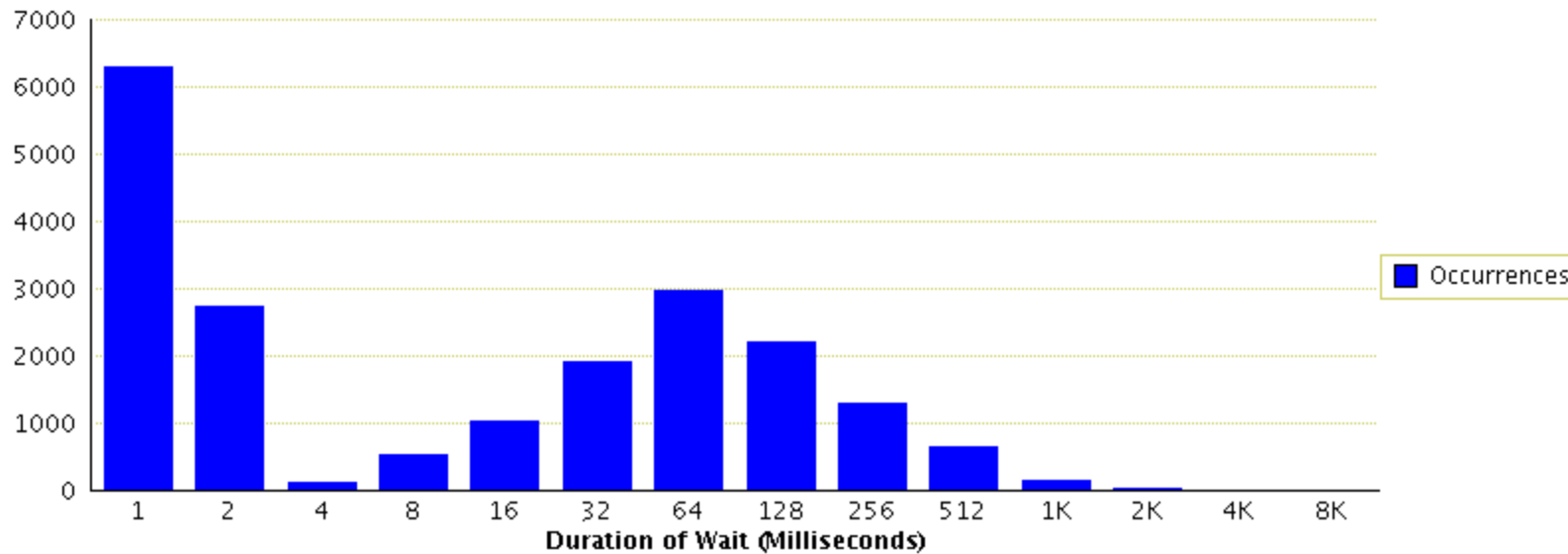
Database: [database](#) > [Active Sessions Waiting: User I/O](#) > Histogram for Wait Event: db file scattered read

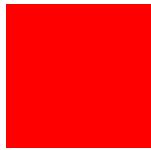
[View Data](#) [Logged in As](#)
[Real Time](#) [Manual Refresh](#)

Histogram for Wait Event: db file scattered read

Page Refreshed **Nov 17, 2004 12:58:43**

Wait Event Occurrences Per Duration





Latch Wait Event Histogram

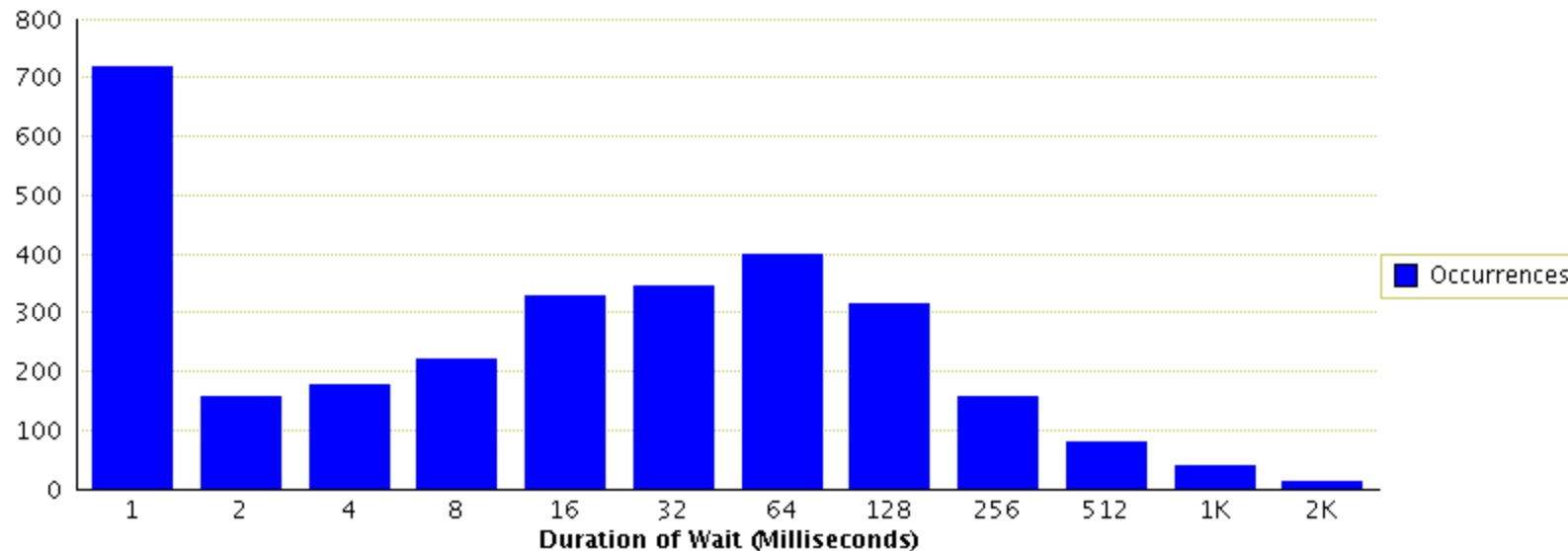
Database: database > [Active Sessions Waiting: Concurrency](#) > Histogram for Wait Event: latch: library cache

View Data [Logged in As](#)
[Real Time](#) [Manual Refresh](#)

Histogram for Wait Event: latch: library cache

Page Refreshed **Nov 17, 2004 1:01:36**

Wait Event Occurrences Per Duration





Histogram Math

- Histograms capture probability distribution of TIME_WAITED by event over this startup cycle

$$\Pr(\textit{time_waited} < \textit{bucket}_N) = \frac{\sum_{\textit{bucket} < N} \textit{WaitCount}}{\sum_{\textit{allbuckets}} \textit{WaitCount}}$$

Significance of Histogram Buckets

$$\textit{Significance}_{\textit{bucket}N} = 1 - \left(\frac{\sum^{\textit{bucket} \geq N} \textit{WaitCount}}{\sum_{\textit{allbuckets}} \textit{WaitCount}} \right)$$

- Cumulative distribution function of TIME_WAITED probabilities represented by the histograms
- Every event in the bucket has at least this significance



Defining “Outlier Events”

- Events with low probability of occurrence
- Events with high significance value
- Q: Has ASH sampled any such events?

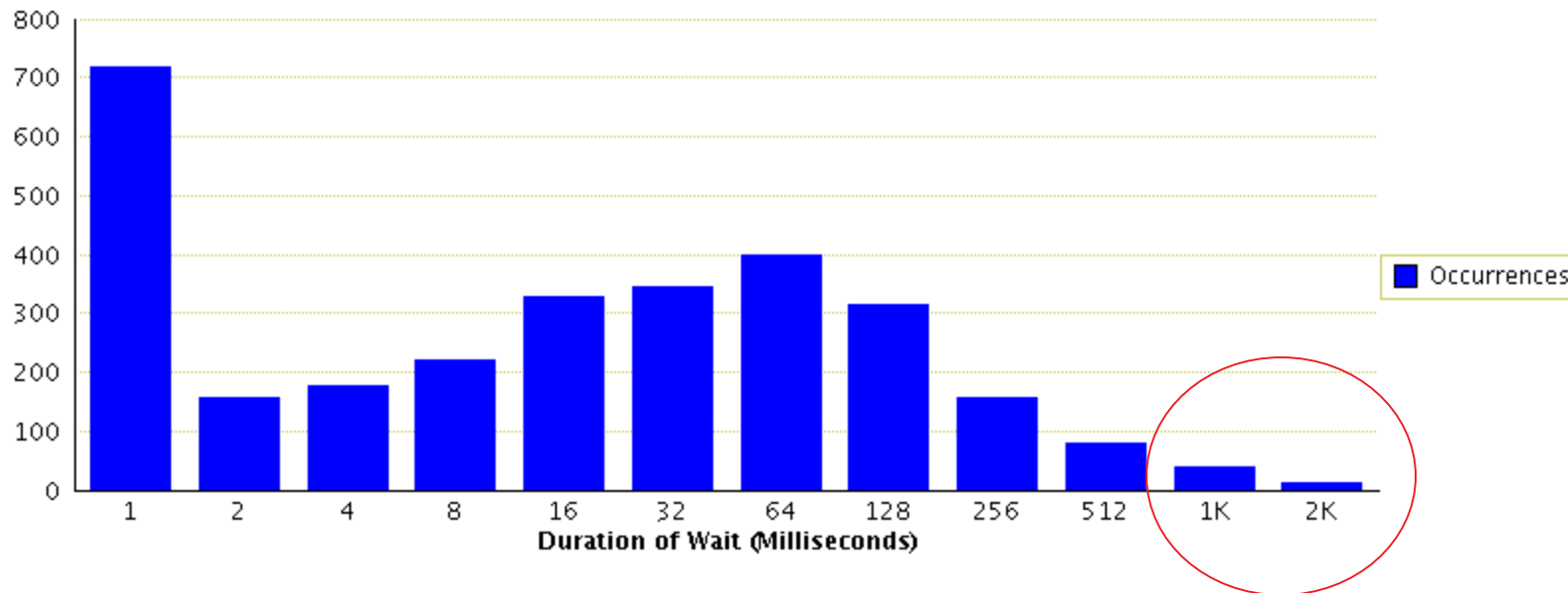


“Outlier” = “Unusual”

Histogram for Wait Event: latch: library cache

Page Refreshed **Nov 17, 2004 1:01:36**

Wait Event Occurrences Per Duration





Finding Outlier Events in ASH

- Which ASH rows (if any) represent wait events with significantly long `TIME_WAITED` against the event histogram record?

Two step process:

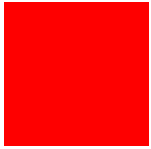
1. Compute event histogram bucket significance
2. Join ASH to histograms and filter by significance

Step 1: Compute Bucket Significance

```
WITH EH$stats
as
(select
    EH.*
    ,ROUND(1 - (tot_count - bucket_tot_count + wait_count) / tot_count,6)
                                                as event_bucket_siglevel
from
    (select event#
        ,event
        ,wait_time_milli
        ,wait_count
        ,ROUND(LOG(2,wait_time_milli))           as event_bucket
        ,SUM(wait_count) OVER (PARTITION BY event#) as tot_count
        ,SUM(wait_count) OVER (PARTITION BY event# ORDER BY wait_time_milli
                                RANGE UNBOUNDED PRECEDING)
                                                as bucket_tot_count
        from v$event_histogram
    ) EH
)
```

Step 2: Join ASH to Buckets and Filter

```
select
    EH.event_bucket
    ,ASH.sample_id
    ,ASH.session_id
    ,EH.event_bucket_siglevel as bucket_siglevel
    ,ASH.event
    ,ASH.time_waited/1000 ASH_time_waited_milli
    ,ASH.sql_id
from
    EH$stats EH
    ,v$active_session_history ASH
where
    EH.event# = ASH.event#
    and EH.event_bucket_siglevel > &siglevel
    and EH.event_bucket = CASE ASH.time_waited
                            WHEN 0 THEN null
                            ELSE TRUNC(LOG(2,ASH.time_waited/1000))+1
                            END
order by
sample_id, event, session_id
;
```



Dataviz: Wait Class Latency Bubble Chart



Motivation

- Why is DB Time for I/O accumulating faster?
 - More I/O events happening?
 - Longer I/O latencies?
- How are latency and volume of events changing over time in the system?
 - Try to understand system characteristics and dynamics better

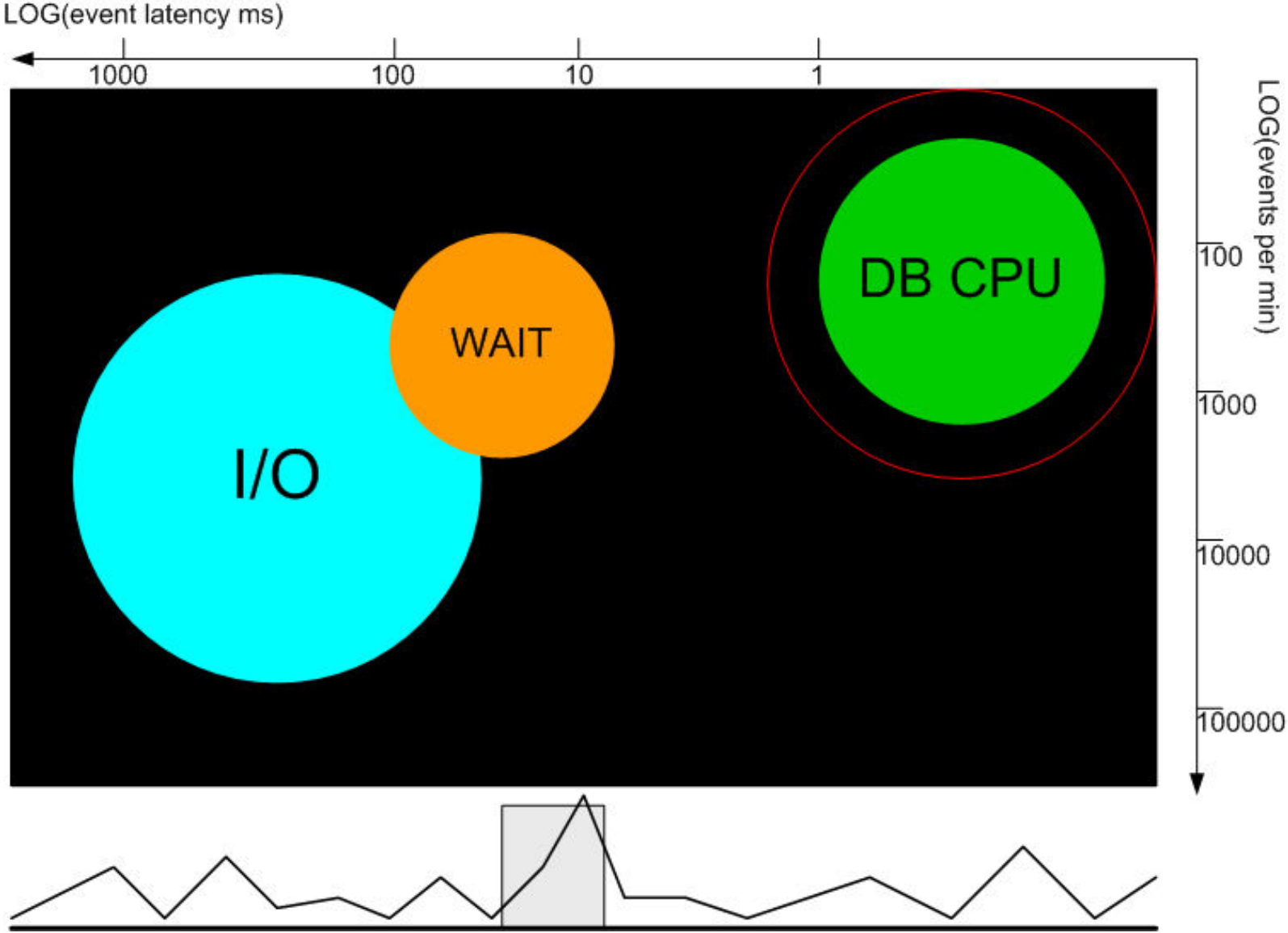


Inspiration

- New ASH Math and Event Latencies
 - Estimate event counts from ASH data
 - $\text{DB Time} / \text{Event count} = \text{Avg Latency}$
- Hans Rosling's TED talk:
 - http://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen.html



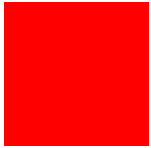
Mockup #1



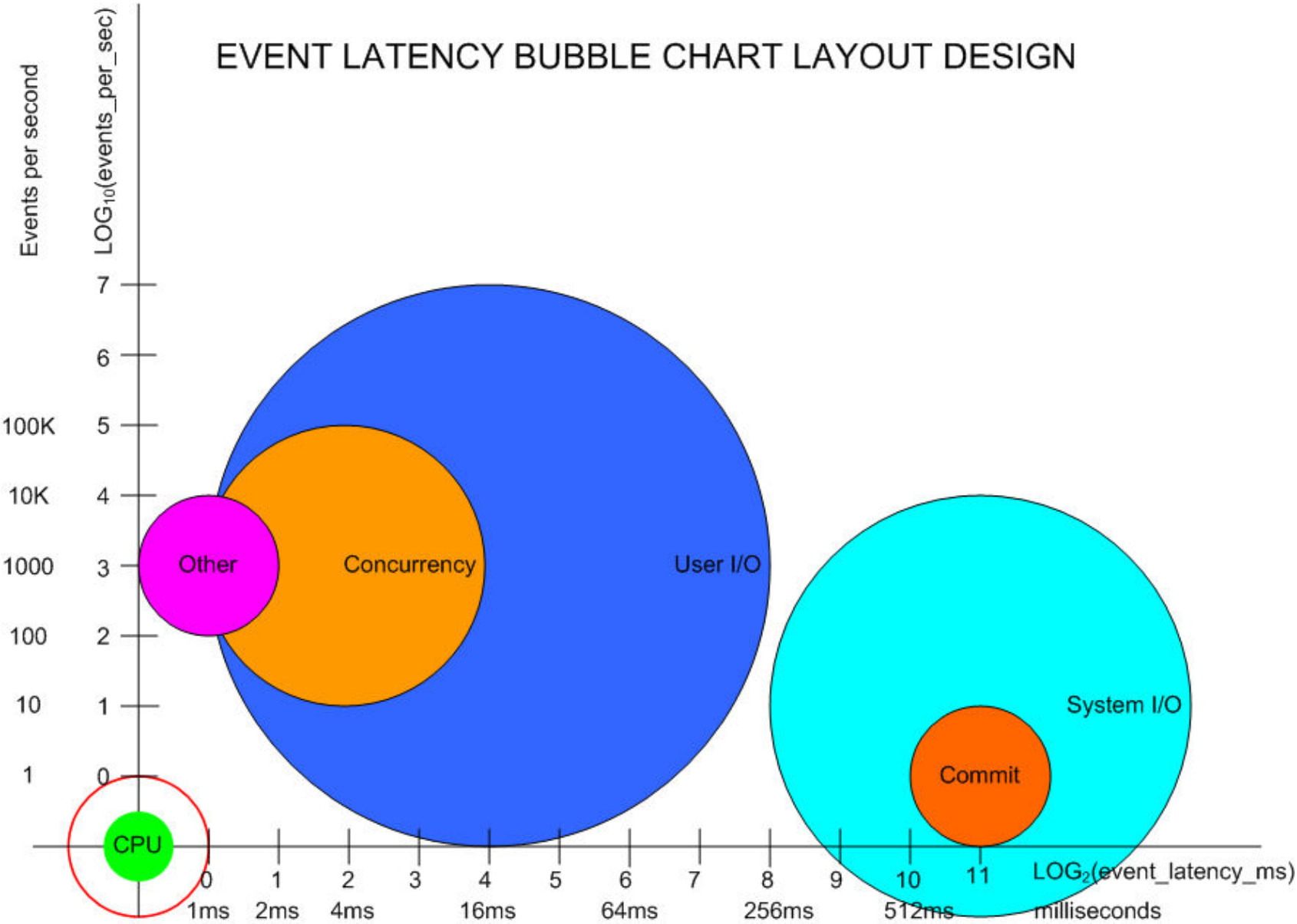


Uri's observation

- Bubble area and position on the grid are deterministic with each other
- Drawing the bubble allows visual comparison of DB time
- We can use the observation to draw an exact mockup



Mockup #2





DEMO?

**SOFTWARE.
HARDWARE.
COMPLETE.**

ORACLE





ORACLE IS THE INFORMATION COMPANY