Contentious Small Tables

Neil Johnson

o·su·mo

Who Am I

Neil Johnson

o·su·mo

Independent Oracle DBA

@neiljdba

neil@osumo.co.uk

http://oraganism.wordpress.com/

Contentious Small Tables

Neil Johnson

o·su·mo

Background

Real issue seen on a busy payments processing system

The application:

- Two node RAC configuration
- RAC friendly - connection fail-over and major application tables hash partitioned
- "Application transactions" are time critical
- Individual application processes authorising trxns and heartbeating information back to a "state" table
- The "state" table is central to the story

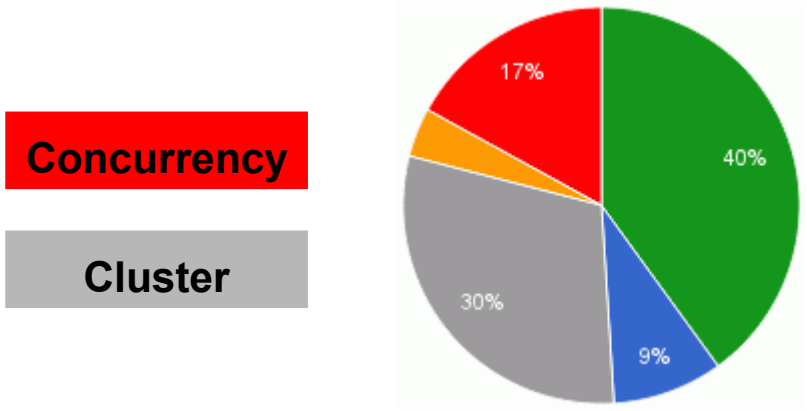
Issue Identification

o·su·mo

In load testing as part of extending the portfolio:

- Increased transaction rate
- Increased number of processes
- Transaction turnaround not fast enough

After interrogating ASH we could see a significant % of application time was spent on “Concurrency” and “Cluster” waits



Issue Identification - ASH

o·su·mo

Simple ASH query used to get session level information

```
select a.EVENT, NVL(wait_class,'CPU') wait_class
,count(*) waits
from v$active_session_history a
where a.session_id = &sess_id.
and SAMPLE_TIME between
                  to_date('&start_time.','dd-mon-yyyy hh24:mi')
           and    to_date('&end_time.','dd-mon-yyyy hh24:mi')
group by a.EVENT, NVL(wait_class,'CPU')
order by count(*) desc;
```

Could have used SQL*Trace for more accurate information

Issue Identification - ASH

Saw significant time on the following events:

11g:

Concurrency	buffer busy waits
Cluster	gc buffer busy acquire gc buffer busy release

10g:

Concurrency	buffer busy waits
Cluster	gc buffer busy

Issue Identification - ASH

o·su·mo

Drill down to object_id

```
gv$active_session_history.current_obj#
```

```
select parameter1
,      parameter2
from   v$event_name
where  name = :b1
order by 1;
```

Drill down to block

```
gv$active_session_history.p1
gv$active_session_history.p2
```

```
file#
block#
```

SQL ID

```
gv$active_session_history.sql_id
```

Issue Identification - ASH

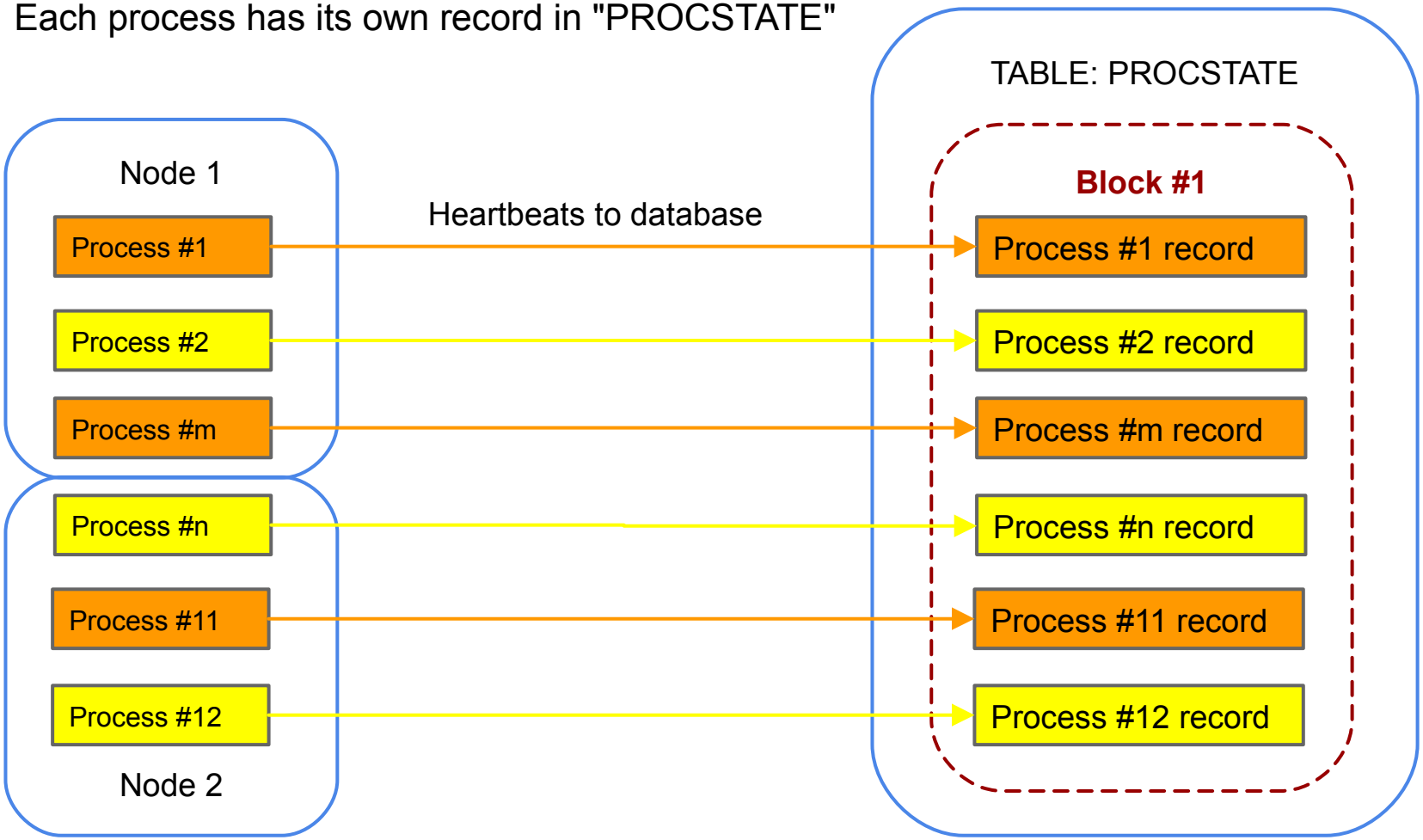
- ASH.P1 & ASH.P2 were the same for many of the undesirable wait events
- A small number of blocks were causing the majority of the contention
- Turned out to be DML on a very small table – the heartbeat table
- Focus on the DBA team to provide a solution without changing (wrapped) code

A (not very good) reconstruction

Application & "state" Table

o·su·mo

Two node RAC
Sessions evenly spread across both instances
Each process has its own record in "PROCSTATE"



Heartbeat "state" table

o·su·mo

```
SQL> desc procstate
 Name                      Null?    Type
 ------------------------- -------- -------------
 PROC_ID                   NOT NULL NUMBER(3)
 PROC_NAME                 NOT NULL VARCHAR2(10)
 PROC_STATUS               NOT NULL VARCHAR2(10)
 PROC_START                NOT NULL TIMESTAMP(6)
 LAST_HEARTBEAT                     TIMESTAMP(6)
 MSG_COUNT                          NUMBER
 ABORT                              CHAR(1)
```

Test Code Outline


```
for each process
    loop
        -- lock the heartbeat record
        select  rowid
        from    procstate
        where   proc_name = L_proc_name
        for update nowait;

        -- do some other brief work
        dbms_lock.sleep(0.02);

        -- heartbeat back to table
        update  procstate
        set     proc_status, msg_count, last_heartbeat
        where   rowid = L_rowid;

        -- end transaction
        commit;

        -- wait for more work to do
        dbms_lock.sleep(dbms_random.value(0.03,0.05));
    end loop;
```

The system used for the tests:

- 2 node RAC hosted on Oracle VirtualBox
- Dual core laptop
- Only 6GB of memory for host and 2 guests combined
- Single HDD for host, guests, Oracle software and DB
- Database and GI at 11.2.0.3

Needless to say - a slow configuration and side effects are exaggerated (this is good for me)

Test #1 - Two Node RAC Baseline

o·su·mo

12 application processes, 6 on each RAC instance

All rows in a single block

```
select dbms_rowid.ROWID_BLOCK_NUMBER(rowid) blockno
,      count(*)
from   procstate
group by dbms_rowid.ROWID_BLOCK_NUMBER(rowid);
```

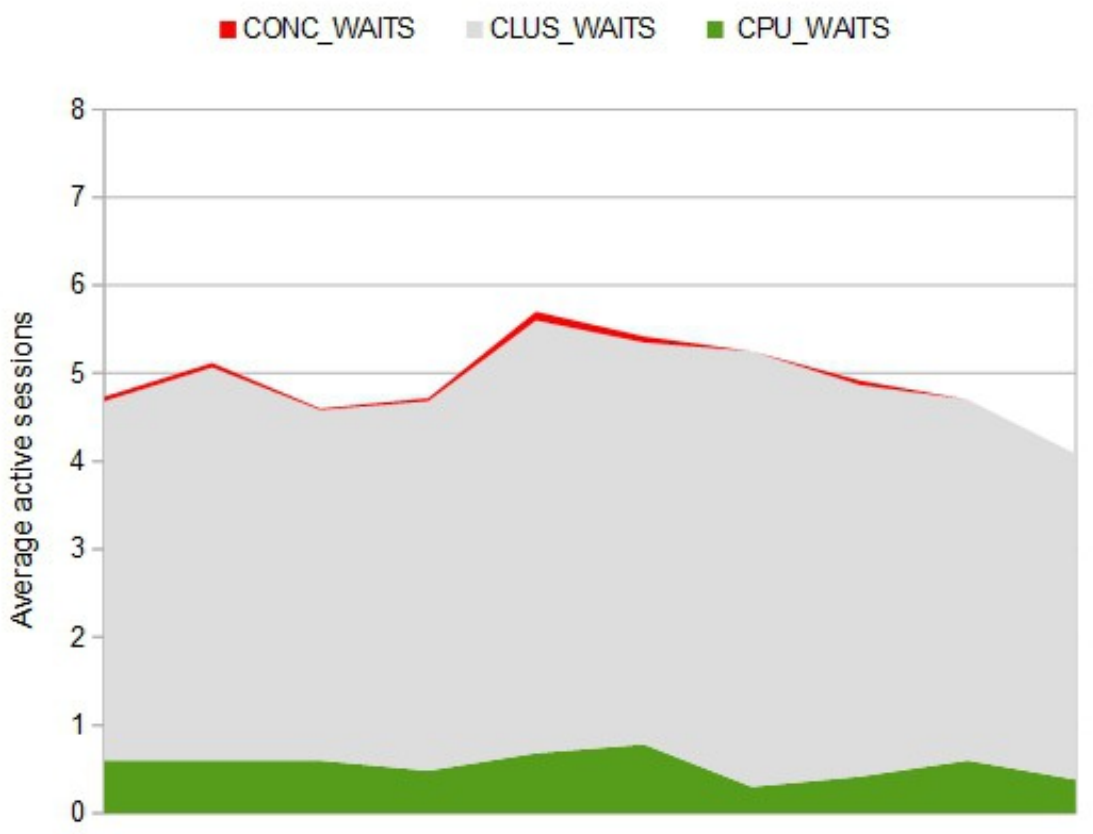
```
   BLOCKNO   COUNT(*)
---------- ----------
    239004         12
```

Test workload executed for 10 minute duration and captured ASH data

Results #1 - Two Node RAC

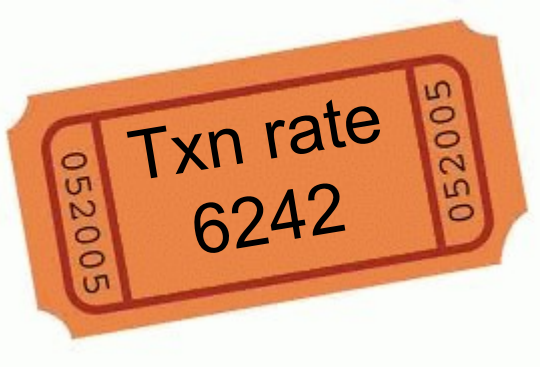
o·su·mo

EVENT	WAITS	P1	P2
gc buffer busy acquire	1012	5	239004
gc buffer busy release	755	5	239004
gc current block busy	373	5	239004
CPU	65	0	0



Almost all time spent on cluster waits

All on block 239004



Some Potential Solutions

Problem

Processes are fighting over a single data block

Potential Solutions

- Node affinity (move workload not rows)
- PCTFREE
- Partitioning
- MINIMIZE RECORDS_PER_BLOCK

Caveat

This is a very specific case - a hot table block in a small segment

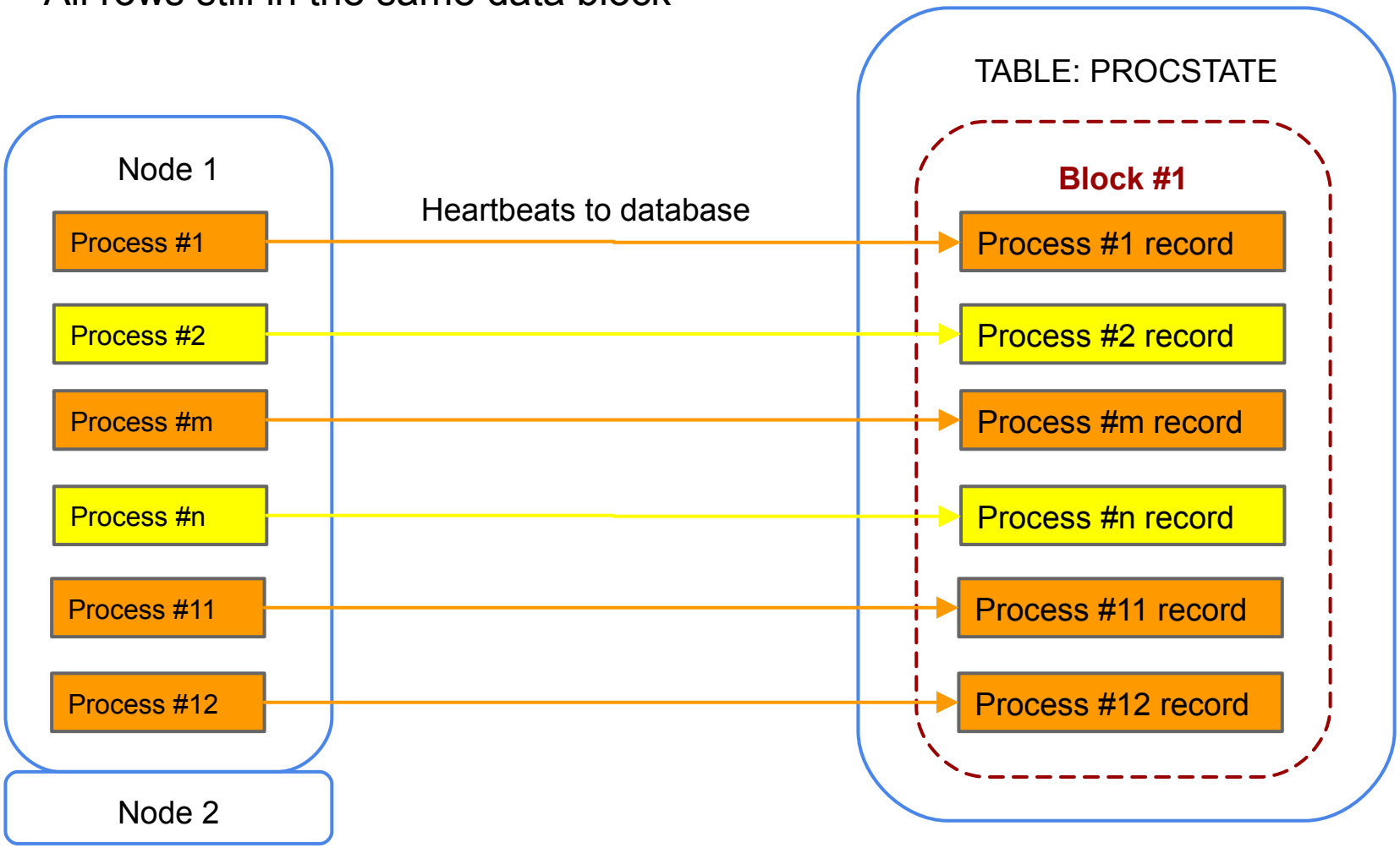
Node Affinity

Test #2 - Node Affinity

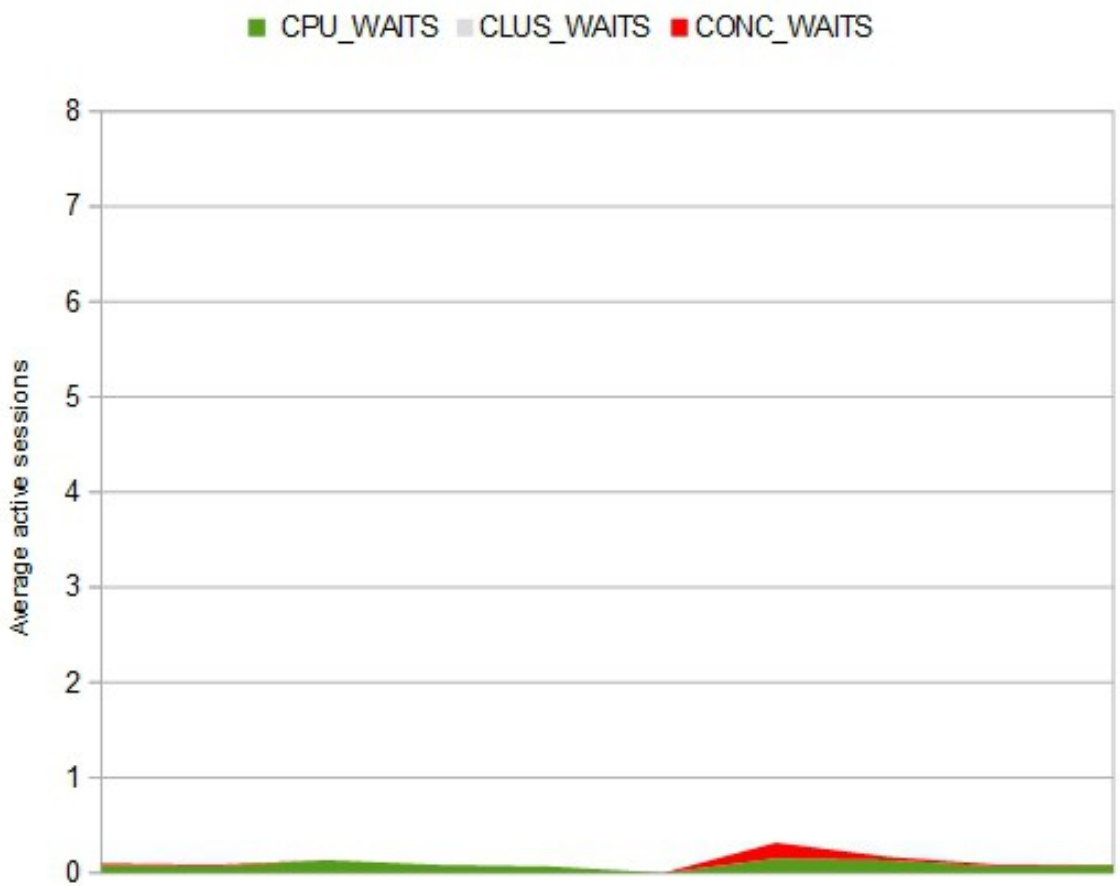
o·su·mo

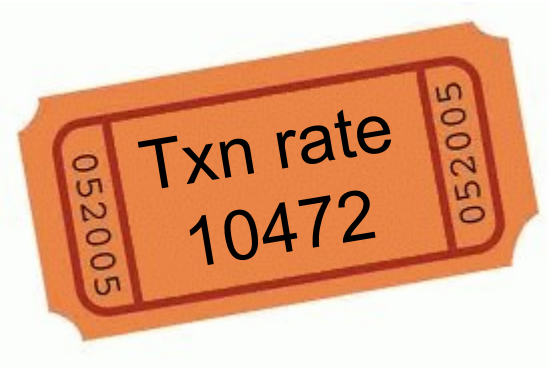
```
   BLOCKNO    COUNT(*)
---------- ----------
    239004          12
```

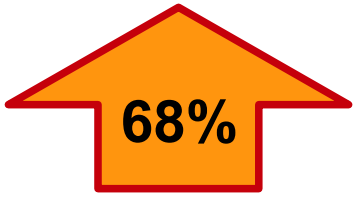
All sessions connected to first RAC instance
All rows still in the same data block



Results #2 - Node Affinity







EVENT	WAITS	P1	P2
CPU	52	0	0
buffer busy waits	13	5	239004
latch: cache buffers chains	1	2164194120	155

Results #2 - Node Affinity

The perfect remedy for global cache waits

but

The rest of the workload was fine running in RAC

Should we compromise application workload for a heartbeat?
For instrumentation?

“buffer busy waits” still possible

PCTFREE

Test #3 - PCTFREE

o·su·mo

PCTFREE *n*

My AVG_ROW_LEN was 46

I had a look at "René Nyffenegger's collection of things on the web:"
http://www.adp-gmbh.ch/ora/concepts/db_block.html

And came up with the following:

```
fixed hdr   + transaction hdr   + table directory    + row directory    = block hdr
57          + (23 * 2           + 4                  + (2 * 12)         = 131 bytes

Row overhead = 3
(blksize - blk hdr)     / (row length   + row overhead)
(8192    - 131)         / (46           + 3)              = 165.5 rows per block
```

Blah Blah

Deduced my rowsize < 1% of available space

Therefore PCTFREE 99 is the best I can do

Test #3 - PCTFREE


```
create table procstate
(
...
) pctfree 99;
```

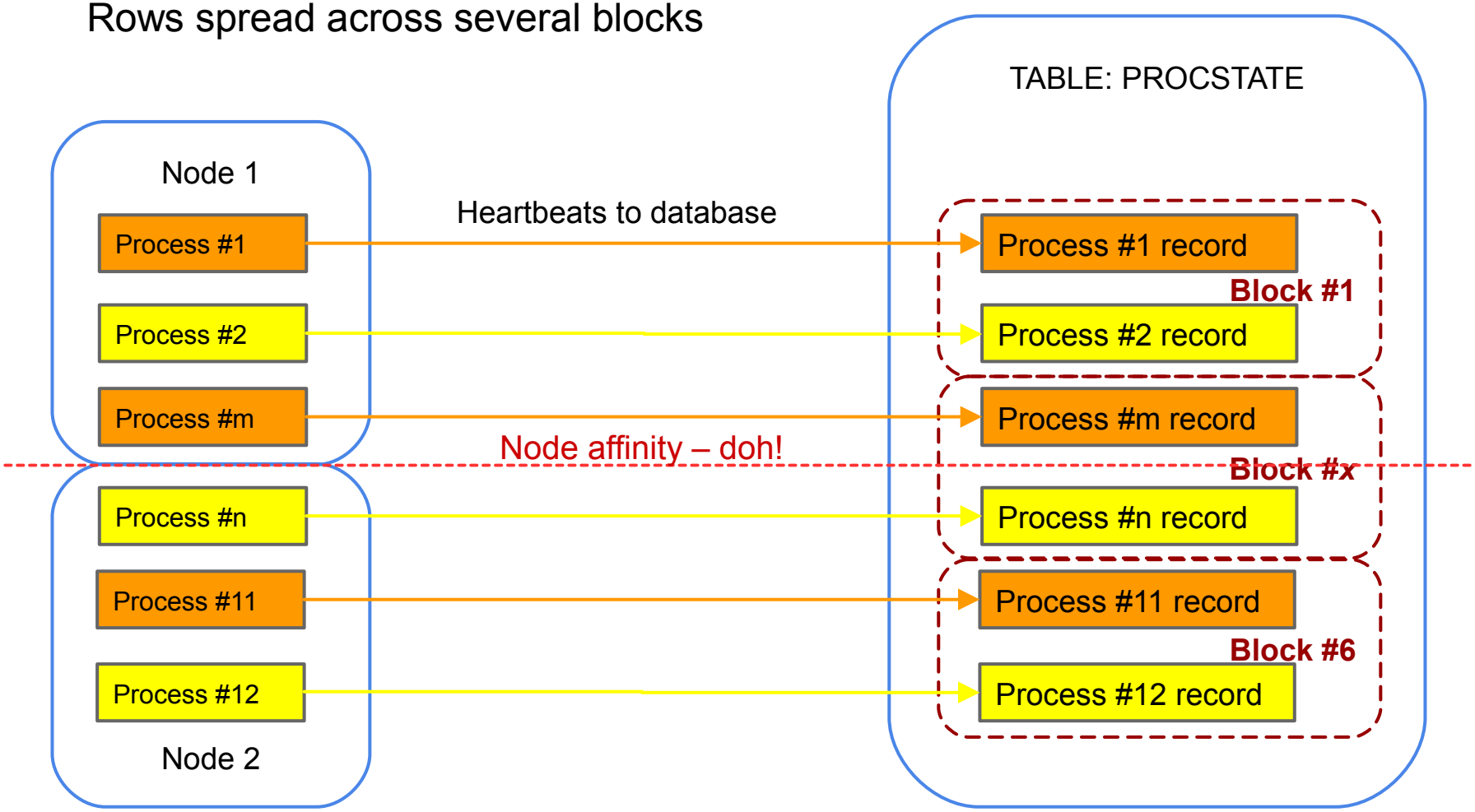
Ended up with two rows per block

BLOCKNO	COUNT(*)
239021	2
239040	2
239022	2
239020	2
239019	2
239023	2

Test #3 – PCTFREE, spot the mistake

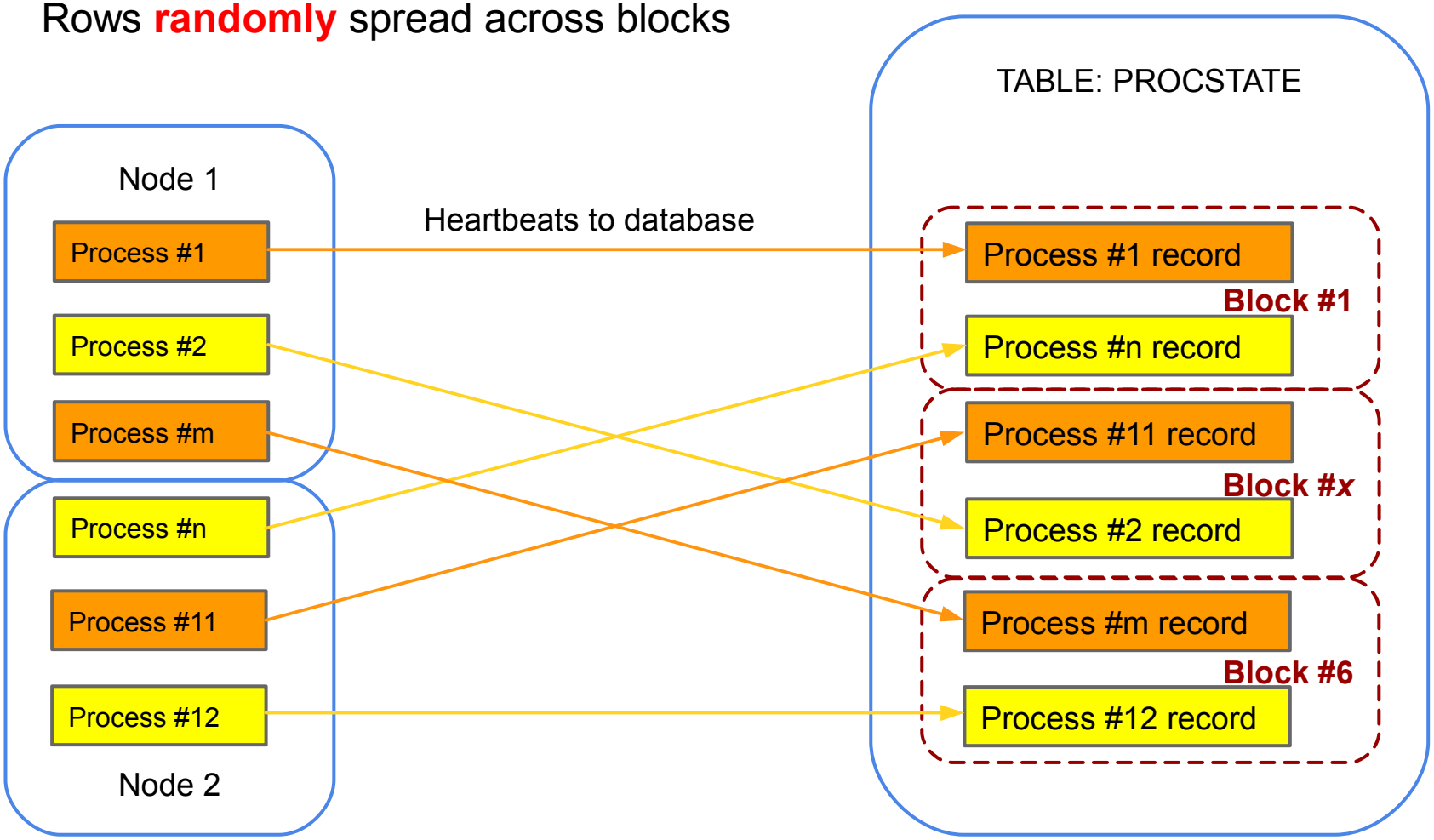
o·su·mo

2 node RAC
Sessions evenly spread across instances
Rows spread across several blocks



Test #3 - PCTFREE

2 node RAC
Sessions evenly spread across instances
Rows **randomly** spread across blocks

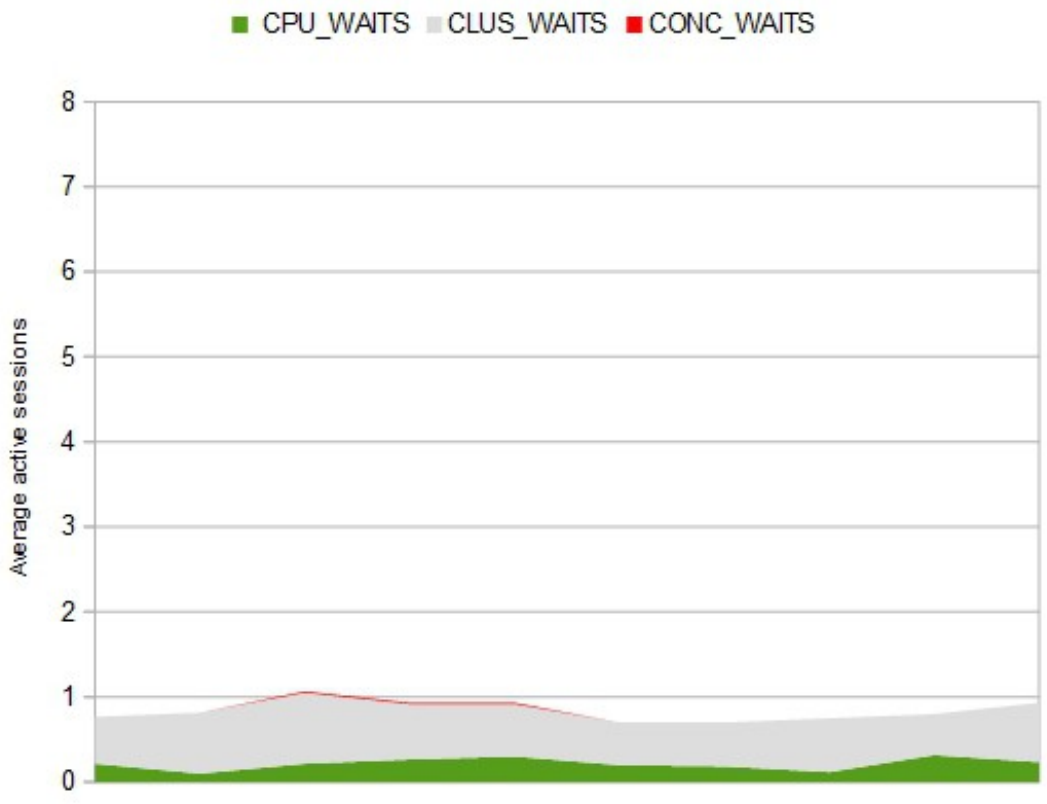


Results #3 - PCTFREE

o·su·mo

A small amount of contention

EVENT	WAITS	P1	P2
gc cr block busy	106	5	239020
gc cr block busy	95	5	239040
CPU	90	0	0
gc current block busy	54	5	239040
gc current block busy	51	5	239020



A big improvement

Hash Partitioning

Test #4 - Hash Partitioning

o·su·mo

Hash partition with power of 2 above desired number of partitions/processes

```
create table procstate
(
...
)
partition by hash (proc_id)
partitions 16;
```

BLOCKNO	COUNT(*)
173610	1
203944	1
235058	2
233010	1
198960	1
239154	1
212402	1
147240	1
237096	1
237618	1
236082	1

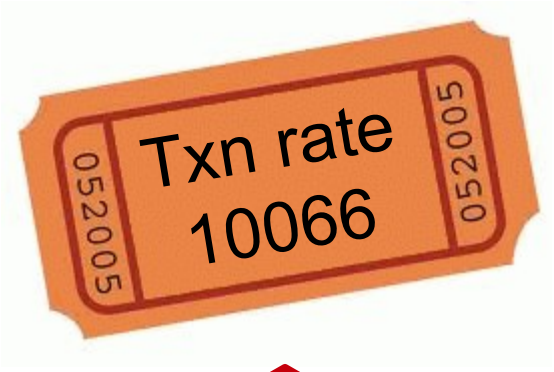
- Rows distributed across blocks
- Some collisions possible on small data sets

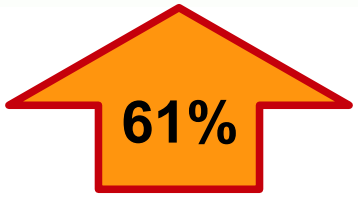
Results #4 - Hash Partitioning

o·su·mo

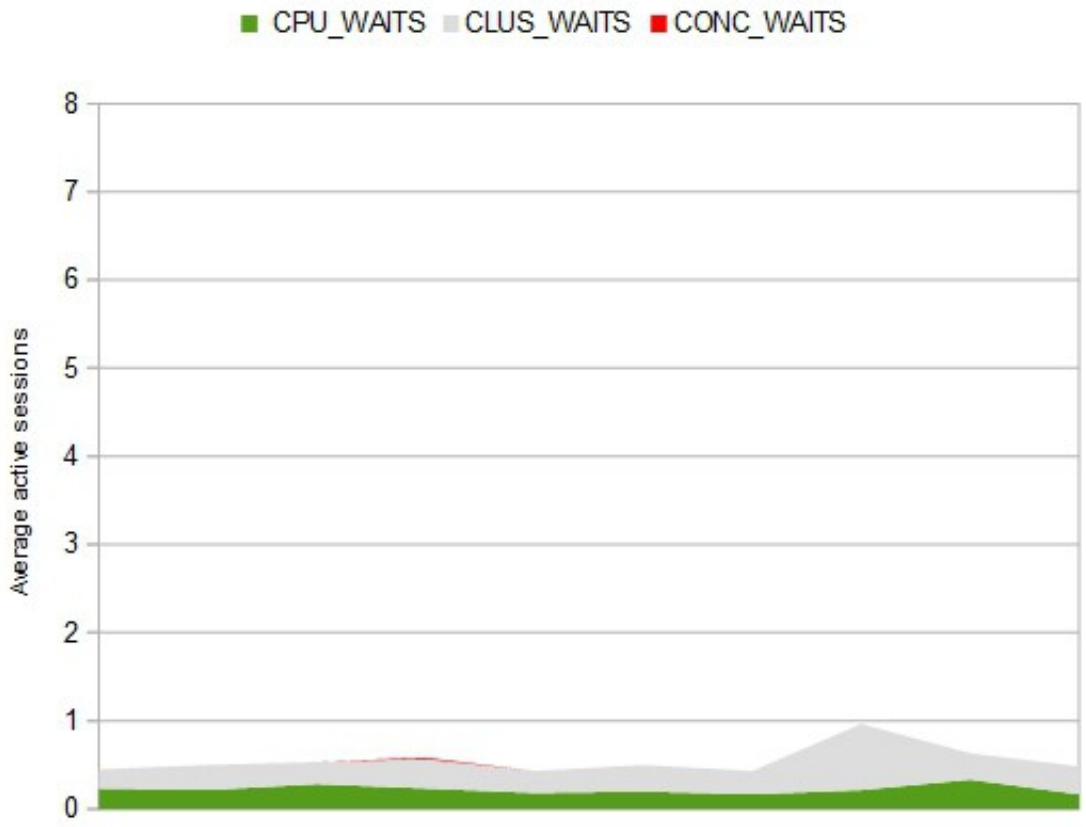
Probably the most obvious solution

Enterprise Edition option

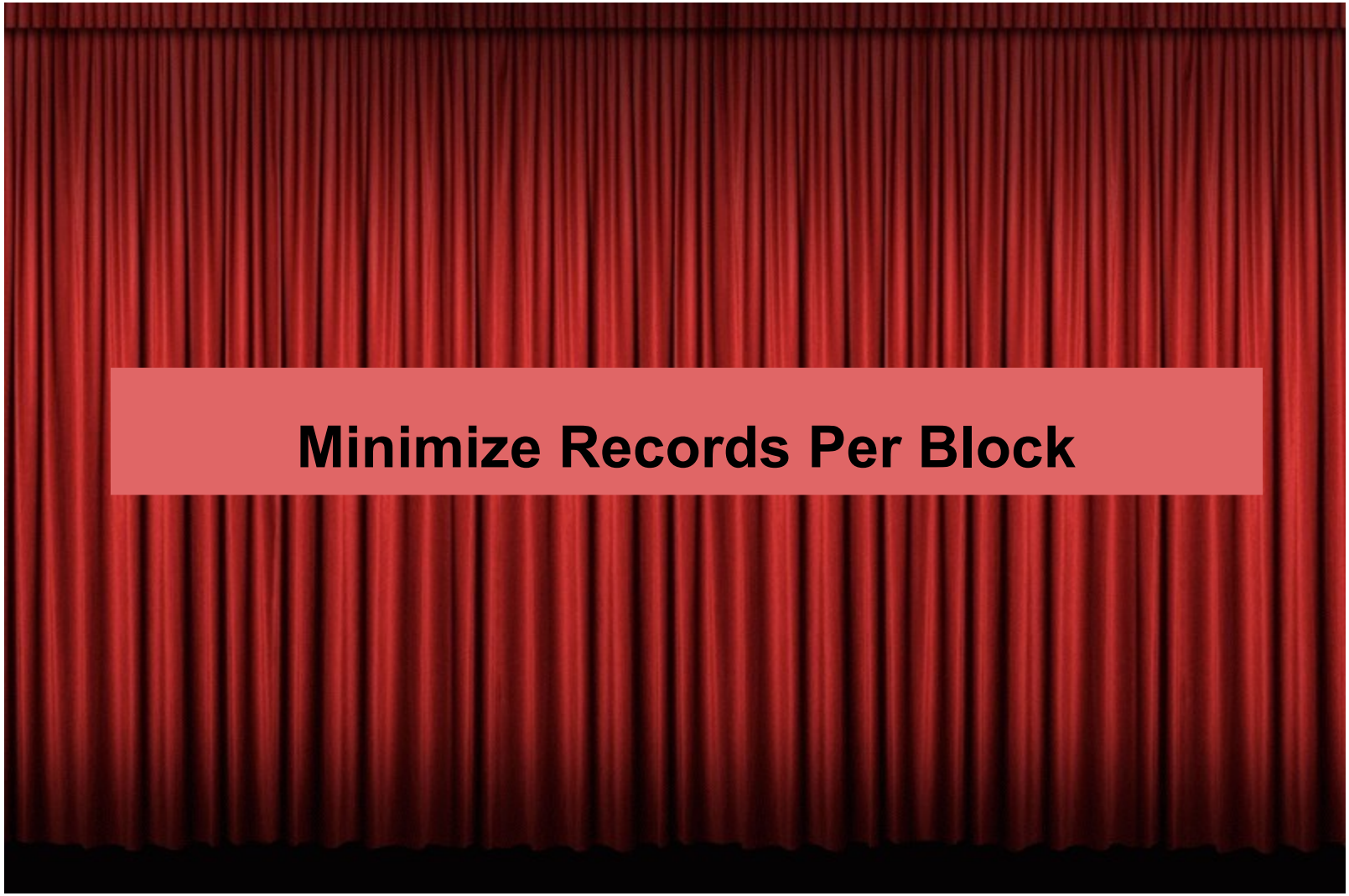




EVENT	WAITS	P1	P2
CPU	102	0	0
gc cr block busy	87	5	235058
gc current block busy	60	5	235058
CPU	32	5	235058
gc buffer busy release	20	5	235058



Interlude



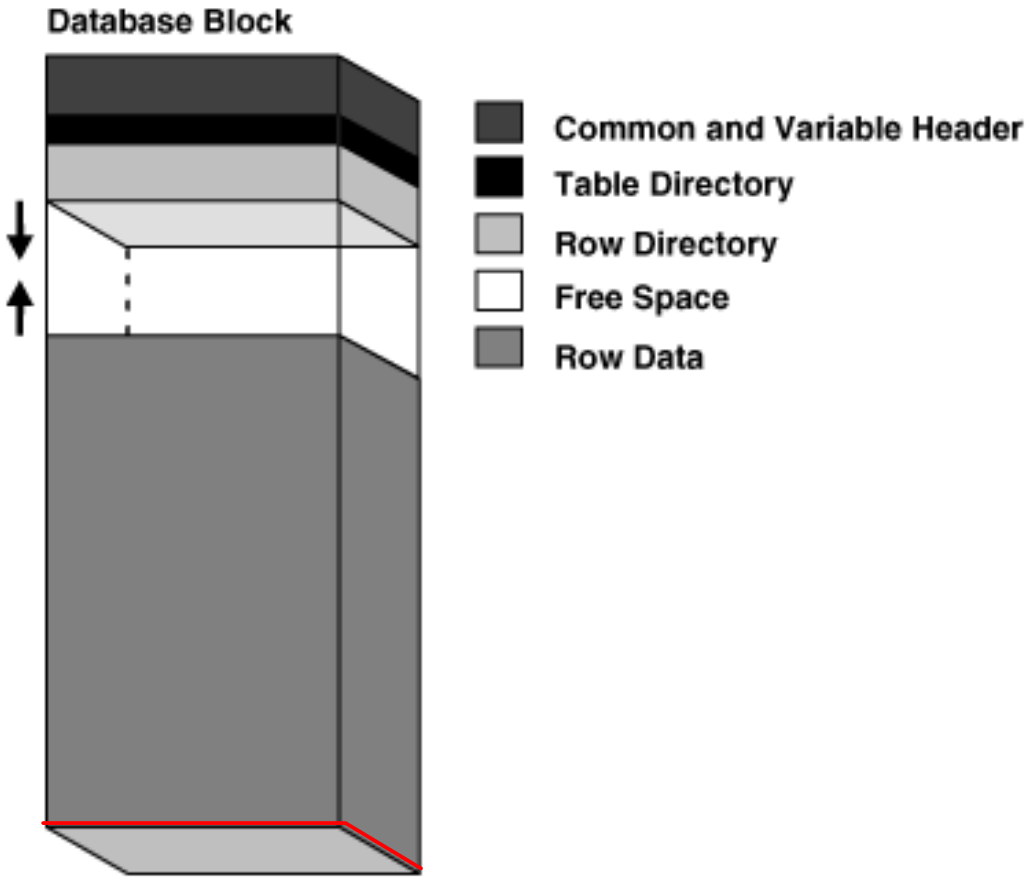
Minimize Records Per Block

- `ALTER TABLE … MINIMIZE RECORDS_PER_BLOCK;`
- Scans a table keeping track of highest row count in any block
- Used to manipulate the Hakan factor
- The Hakan factor limits the number of rows a block will accept
- Hakan factor is stored in sys.tab$.spare1

```
select spare1 from sys.tab$ where obj# = :id;

    SPARE1
----------
       736
```

- The Hakan factor is already set for all your heap tables
- sys.tab$.spare1 is a multi-use column

Minimize Records Per Block



http://docs.oracle.com/cd/E11882_01/server.112/e25789/logical.htm#i4902

Minimize Records Per Block

- A number of different factors affect the Hakan factor
- Block size has a big impact

```
create table tab4k (num1 number)
tablespace users4k;
```

Block Size	Hakan Factor*
4k	364
8k	736
16k	1481
32k	2971

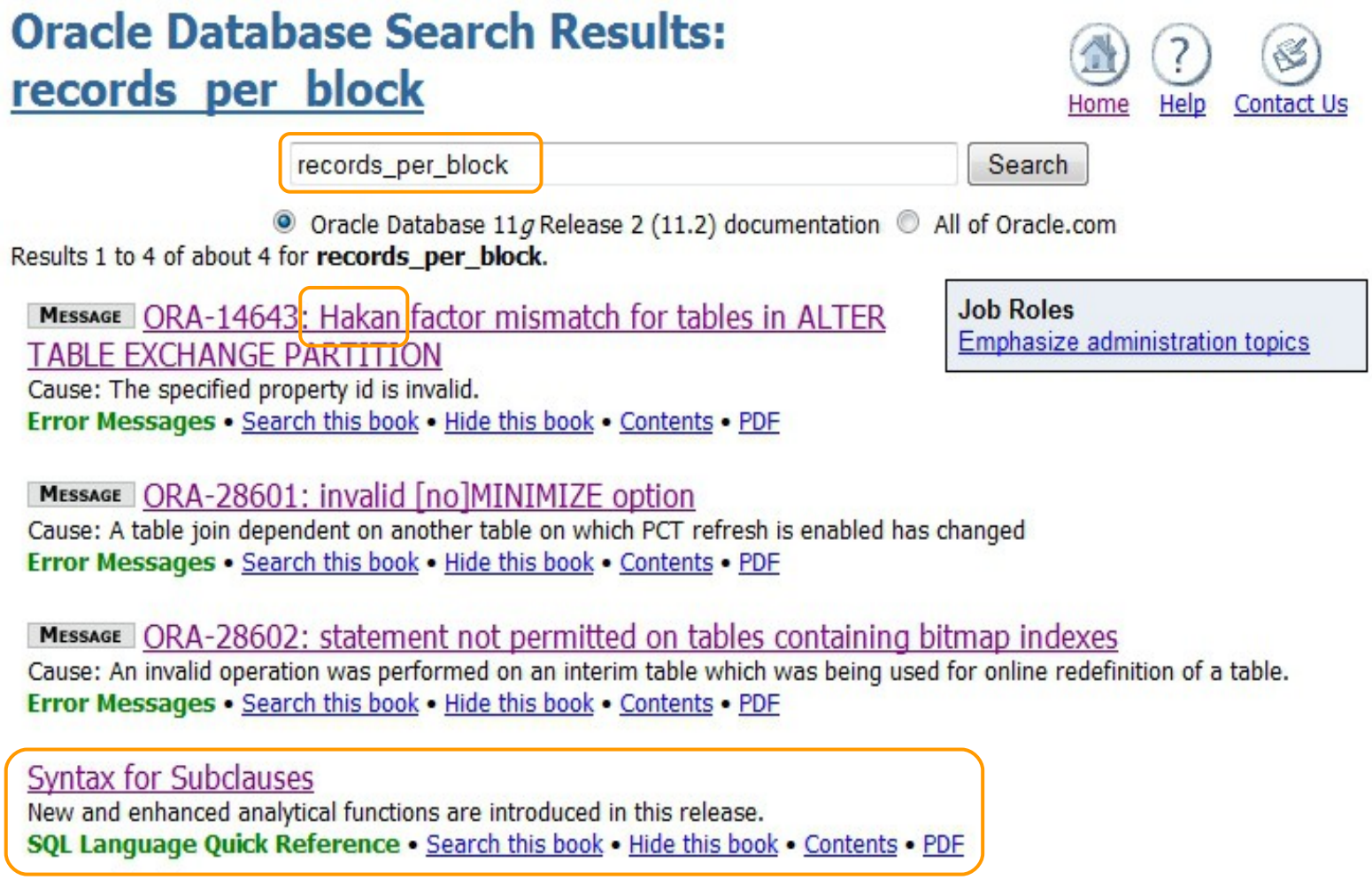
- There is little written about MRPB and the Hakan factor

*this is a very simple single column table

Minimize Records Per Block

o·su·mo

What does the Oracle 11g documentation say...



Minimize Records Per Block

Only entry in documentation when searching for "records_per_block" is in the SQL Language Quick Reference:

records_per_block_clause

```
{ MINIMIZE | NOMINIMIZE } RECORDS_PER_BLOCK
```

No description or advice

Minimize Records Per Block

o·su·mo

Surely it's under ALTER TABLE:

http://docs.oracle.com/cd/E11882_01/server.112/e41084/statements_3001.htm#SQLRF53325

records_per_block_clause

The `records_per_block_clause` lets you specify whether Oracle Database restricts the number of records that can be stored in a block. This clause ensures that any bitmap indexes subsequently created on the table will be as compressed as possible.

...

MINIMIZE Specify `MINIMIZE` to instruct Oracle Database to calculate the largest number of records in any block in the table and to limit future inserts so that no block can contain more than that number of records.

Can be found when search for "records_per_block_**clause**" - not typically what you are going to search for!

Minimize Records Per Block

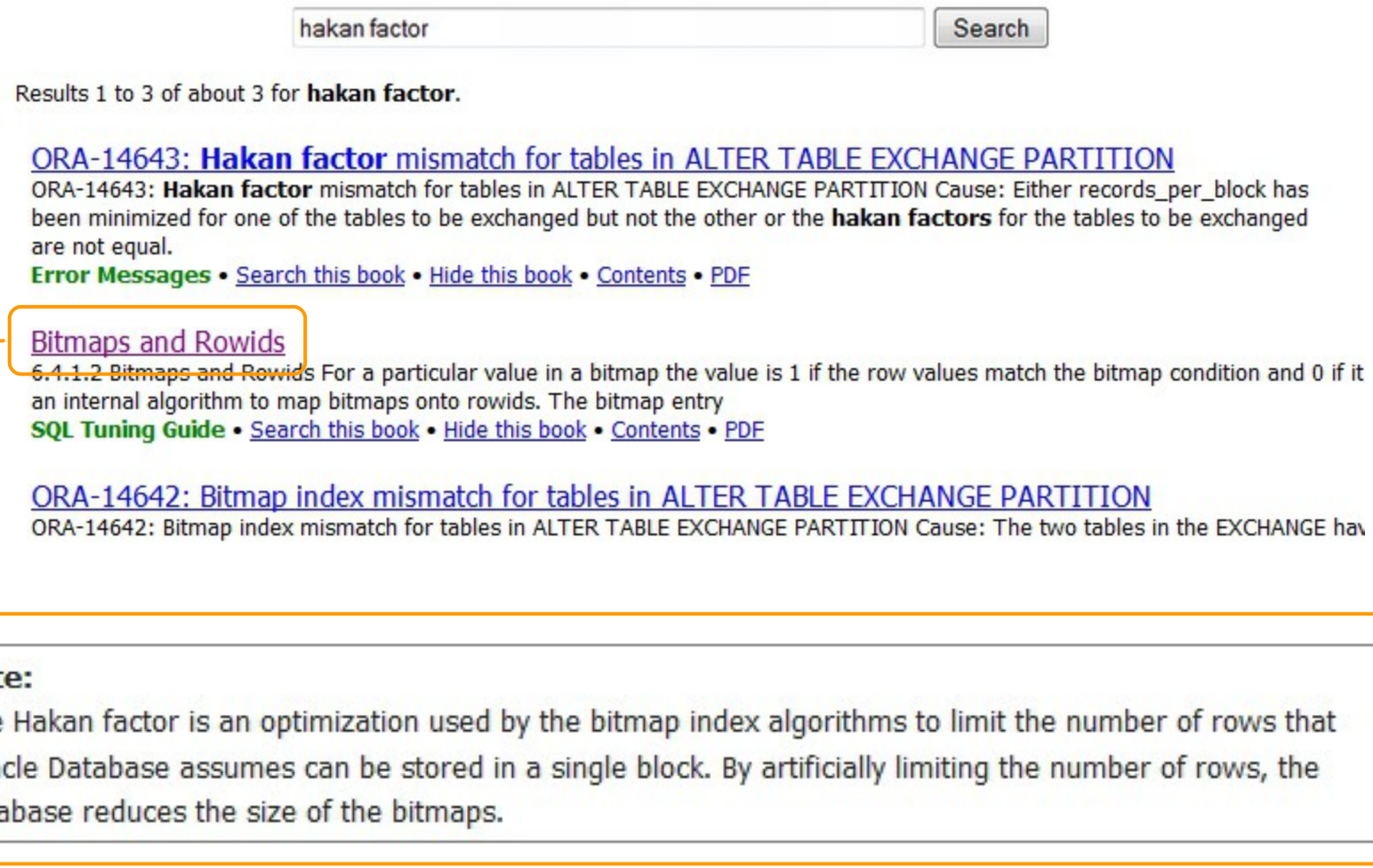
o·su·mo

11g docs search for “Hakan factor”:

Minimize Records Per Block

o·su·mo

12c docs search for “Hakan factor”:



Minimize Records Per Block

- Command primarily for use with bitmap indexes
- Can be used to dramatically reduce the size of a bitmap index (in specific cases)
- Nice example on Richard Foote's blog

http://richardfoote.wordpress.com/2011/07/19/bitmap-indexes-minimize-records_per_block-little-wonder/

Minimize Records Per Block

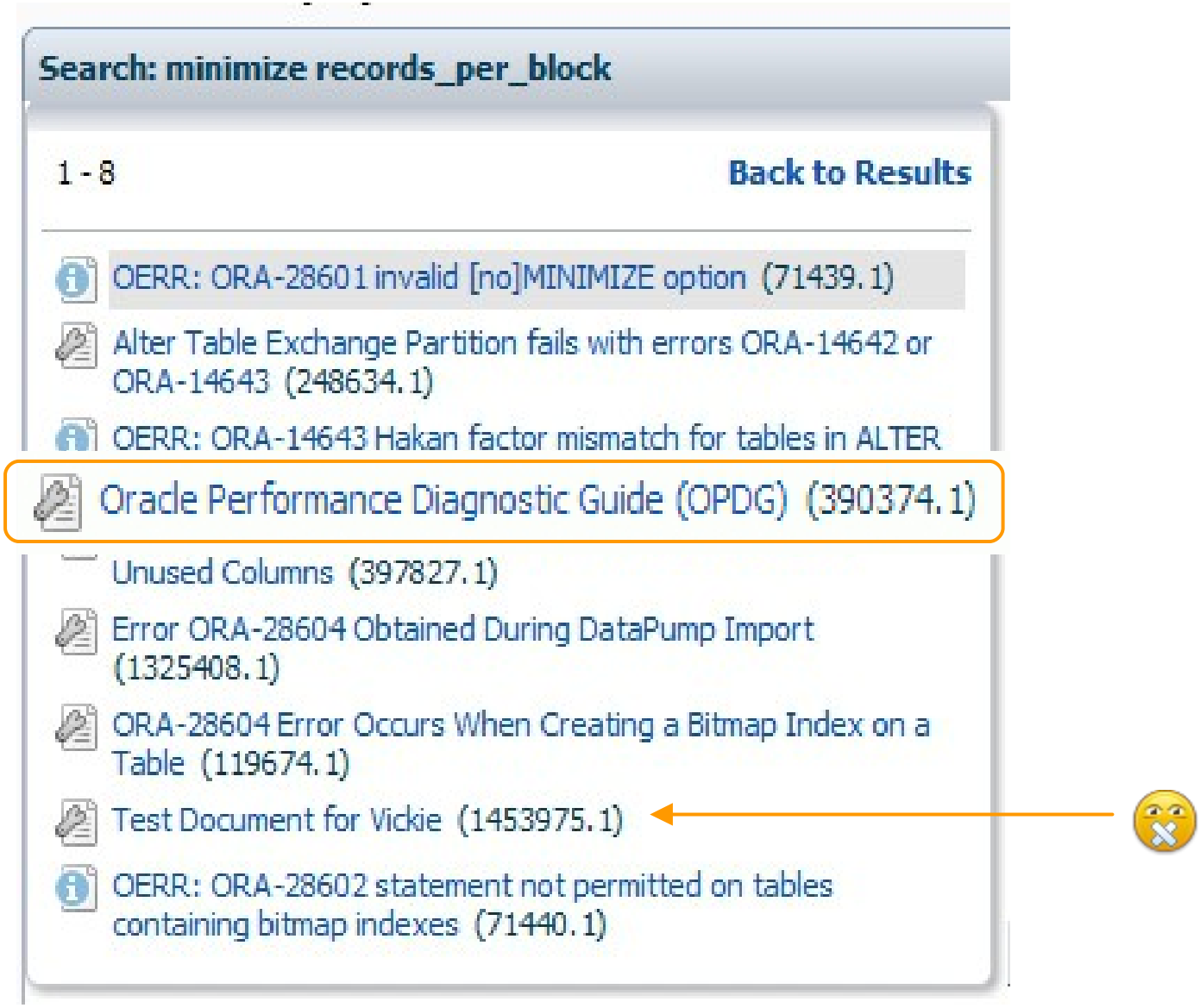
o·su·mo

M.O.S:

Minimize Records Per Block

o·su·mo

M.O.S:



Minimize Records Per Block

o·su·mo

Oracle Performance Diagnostic Guide (OPDG) [ID 390374.1]

Cause Identified: Hot blocks due to concurrently accessing a popular block

Solution Identified: Spread out the rows over more blocks

*Either the query must be executed less often or the **rows need to be spread out among more blocks**.*

...

*Rows can be spread out by rebuilding the table using a **larger value for PCTFREE**. Another way to spread rows out is to **make use of the table option, MINIMIZE RECORDS_PER_BLOCK***

Minimize Records Per Block - How

Oracle Performance Diagnostic Guide (OPDG) [390374.1]

- 1. Export the table
- 2. Truncate the table
- 3. Insert the desired number of rows per block. E.g., if you only want 10 rows per block, insert just 10 rows.
- 4. Alter the table to set minimize records_per_block setting. E.g.,

```
Alter table stock_prices minimize records_per_block;
```

- 5. Delete the rows you inserted in step 3
- 6. Import the table

Minimize Records Per Block - Where

So what does MRPB do to the Hakan factor?

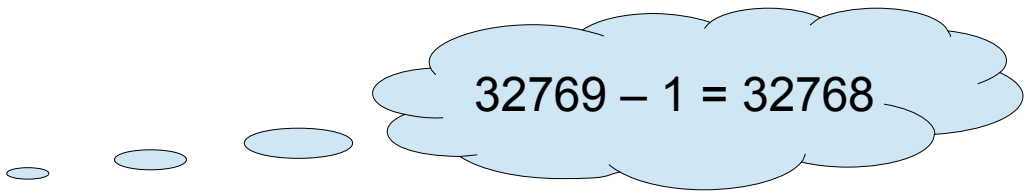
```
select spare1 from sys.tab$ where obj# = :id;
```

```
    SPARE1
----------
       736
```

```
alter table <table_name> minimize records_per_block;
```

```
select spare1 from sys.tab$ where obj# = :id;
```

```
    SPARE1
----------
     32769
```



Minimize Records Per Block - Where

BITAND()

From the docs:
http://docs.oracle.com/cd/E16655_01/server.121/e17209/functions021.htm#SQLRF00612

"*The BITAND function treats its inputs and its output as vectors of bits; the output is the bitwise AND of the inputs.*"

The making of a nice example

```
SELECT BITAND(
    BIN_TO_NUM(1,1,0),
    BIN_TO_NUM(0,1,1)) "Binary"
  FROM DUAL;

    Binary
----------
         2
```

Minimize Records Per Block - Where


```
BIN_TO_NUM(1,1,0),
           &  &  &
BIN_TO_NUM(0,1,1))
           =  =  =
           0  1  0
```

Binary “0 1 0” = Decimal “2”

Minimize Records Per Block - Where

Sounds more complex than it is

Anyone who's used CHMOD in numeric mode will have used similar principles

e.g.

```
chmod 761 myfile

decimal:   7     6     1

binary:   111   110   001

flag:     rwx   rw-   --x
```

Minimize Records Per Block - Where


```
select  spare1
    ,   least(1,BITAND(spare1,32768)) c32k
    ,   least(1,BITAND(spare1,16384)) c16k
    ,   least(1,BITAND(spare1,8192)) c8k
    ,   least(1,BITAND(spare1,4096)) c4k
    ,   least(1,BITAND(spare1,2048)) c2k
    ,   least(1,BITAND(spare1,1024)) c1k
    ,   least(1,BITAND(spare1,64)) c64
    ,   least(1,BITAND(spare1,32)) c32
    ,   least(1,BITAND(spare1,16)) c16
    ,   least(1,BITAND(spare1,8)) c8
    ,   least(1,BITAND(spare1,4)) c4
    ,   least(1,BITAND(spare1,2)) c2
    ,   greatest(0,BITAND(spare1,1)) c1
from    sys.tab$
where obj# = (select obj# from sys.obj$
              where name = 'PROCSTATE');
```

least(1,BITAND(spare1,32768))

Minimize Records Per Block - Where

o·su·mo

tab$.spare1 values with Minimize Records Per Block on...

1 row

SPARE1	C32K	C16K	C8K	C4K	C2K	C1K	C512	C256	C128	C64	C32	C16	C8	C4	C2	C1
32769	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

2 rows

SPARE1	C32K	C16K	C8K	C4K	C2K	C1K	C512	C256	C128	C64	C32	C16	C8	C4	C2	C1
32769	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

3 rows

SPARE1	C32K	C16K	C8K	C4K	C2K	C1K	C512	C256	C128	C64	C32	C16	C8	C4	C2	C1
32770	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

4 rows

SPARE1	C32K	C16K	C8K	C4K	C2K	C1K	C512	C256	C128	C64	C32	C16	C8	C4	C2	C1
32771	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Minimize Records Per Block - Where

o·su·mo

It looks like 16th bit used to record MRPB usage and the bits to the right (+1) specify the maximum number of rows permitted per block.

```
create table tab4k (num1 number) tablespace users4k;

SPARE1 C32K C16K  C8K  C4K  C2K  C1K C512 C256 C128  C64  C32  C16   C8   C4   C2   C1
------ ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
   364    0    0    0    0    0    0    0    1    0    1    1    0    1    1    0    0

create table tab8k (num1 number) tablespace users8k;

SPARE1 C32K C16K  C8K  C4K  C2K  C1K C512 C256 C128  C64  C32  C16   C8   C4   C2   C1
------ ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
   736    0    0    0    0    0    0    1    0    1    1    1    0    0    0    0    0

create table tab16k (num1 number) tablespace users16k;

SPARE1 C32K C16K  C8K  C4K  C2K  C1K C512 C256 C128  C64  C32  C16   C8   C4   C2   C1
------ ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  1481    0    0    0    0    0    1    0    1    1    1    0    0    1    0    0    1

create table tab32k (num1 number) tablespace users32k;

SPARE1 C32K C16K  C8K  C4K  C2K  C1K C512 C256 C128  C64  C32  C16   C8   C4   C2   C1
------ ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  2971    0    0    0    0    1    0    1    1    1    0    0    1    1    0    1    1
```

Minimize Records Per Block

o·su·mo

SQL*Trace of “ALTER TABLE … MINIMIZE RECORDS_PER_BLOCK” shows:

```
select max(sys_op_rpb(rowid)) from "TCTDBS"."MYHAK"
```

This function (or dbms_rowid.rowid_row_number()) potentially indicates why values lag behind the actual row count

```
select rownum
, dbms_rowid.rowid_relative_fno(rowid) file_no
, dbms_rowid.rowid_block_number(rowid) blk_no
, dbms_rowid.rowid_row_number(rowid) row_no
, sys_op_rpb(rowid) row_rno2
from tctdbs.myhak;
```

ROWNUM	FILE_NO	BLK_NO	ROW_NO	ROW_RNO2
1	11	18676	0	0
2	11	18676	1	1
3	11	18676	2	2
4	11	18676	3	3

Minimize Records Per Block

A block dump confirms rows numbered from zero onwards

```
alter system dump datafile 11 block 18676;


data_block_dump,data header at 0x110a69464
===============
...
nrow=4
...
block_row_dump:
tab 0, row 0, @0x1efe
...
tab 0, row 1, @0x1f4a
...
tab 0, row 2, @0x1e64
...
tab 0, row 3, @0x1eb0
...
end_of_block_dump
```

Why store “1” when we only have row zero?

My guess is that for the primary use of Bitmap Indexes, the difference between 1 and 2 rows is irrelevant, plus storing “0” is untidy

Minimize Records Per Block - IMPDP

o·su·mo

Import Doesn't Do As I Expected

Minimize Records Per Block - IMPDP

o·su·mo

```
select spare1 from sys.tab$ where obj# = :b1;

    SPARE1
----------
     32769

select dbms_rowid.rowid_block_number(rowid) blk_no
, count(*)
...

    BLK_NO   COUNT(*)
---------- ----------
     71373          2
...
     18614          2

101 rows selected.
```

= MRPB of 2 rows

Minimize Records Per Block - IMPDP

o·su·mo

What if we drop rather than truncate the table?

- `expdp tables=tctdbs.myhak ...`
- `SQL> drop table tctdbs.myhak;`
- `impdp tables=tctdbs.myhak ...`
-

```
    SPARE1
----------     =    32856 – 32768 + 1    =    MRPB of 89 rows
     32856
```

```
    BLK_NO    COUNT(*)
---------- ----------
     18619          89
     18621          29
     18620          83
```

Minimize Records Per Block - IMPDP

- IMPDP remembered to modify the Hakan factor
- But modified it **after** loading data
- Meaning if we move data using DataPump after using MRPB then we need to be careful
- I didn't expect this but it makes sense from the bitmap index angle
- IMP has the same behaviour

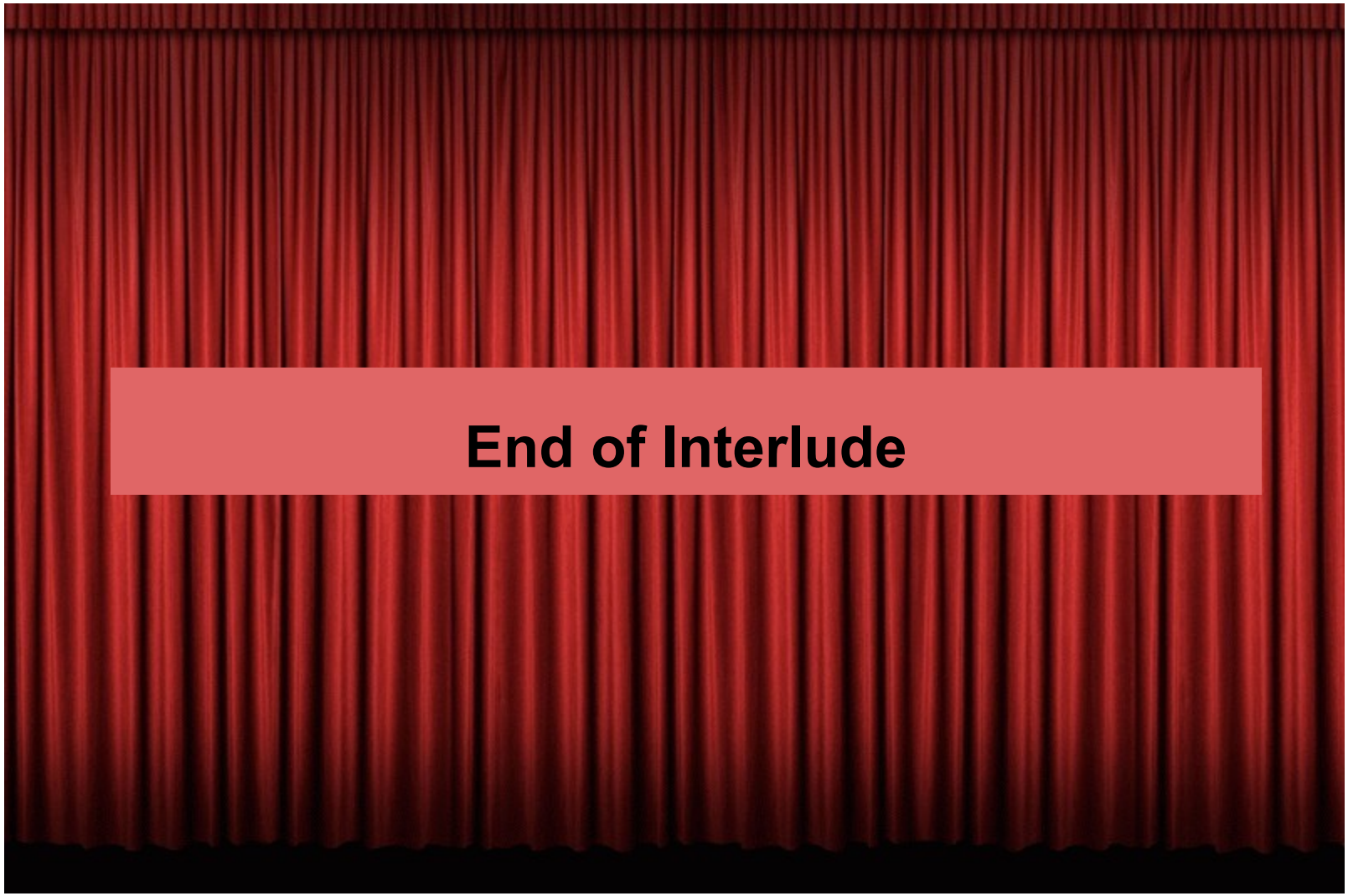
Minimize Records Per Block

In summary:

- Not well documented or publicised
- Barely visible in the data dictionary
- Values in tab$.spare1 are not human friendly
- Can be (mis)used to limit the maximum number of rows per block
- Manipulates the Hakan factor
- Recommended as a “hot block” solution in M.O.S
- NOMINIMIZE to reset to default Hakan factor
- IMP/IMPDP may not do as expected

This is one ugly duckling

But this is also one geeky duckling


End of Interlude

Minimize Records Per Block

Test #5 - Minimize Records Per Block

o·su·mo

a) Load the desired number of rows per block

```
  COUNT(*)
----------
         1
```

b) Modify the Hakan factor

```
alter table procstate minimize records_per_block;
```

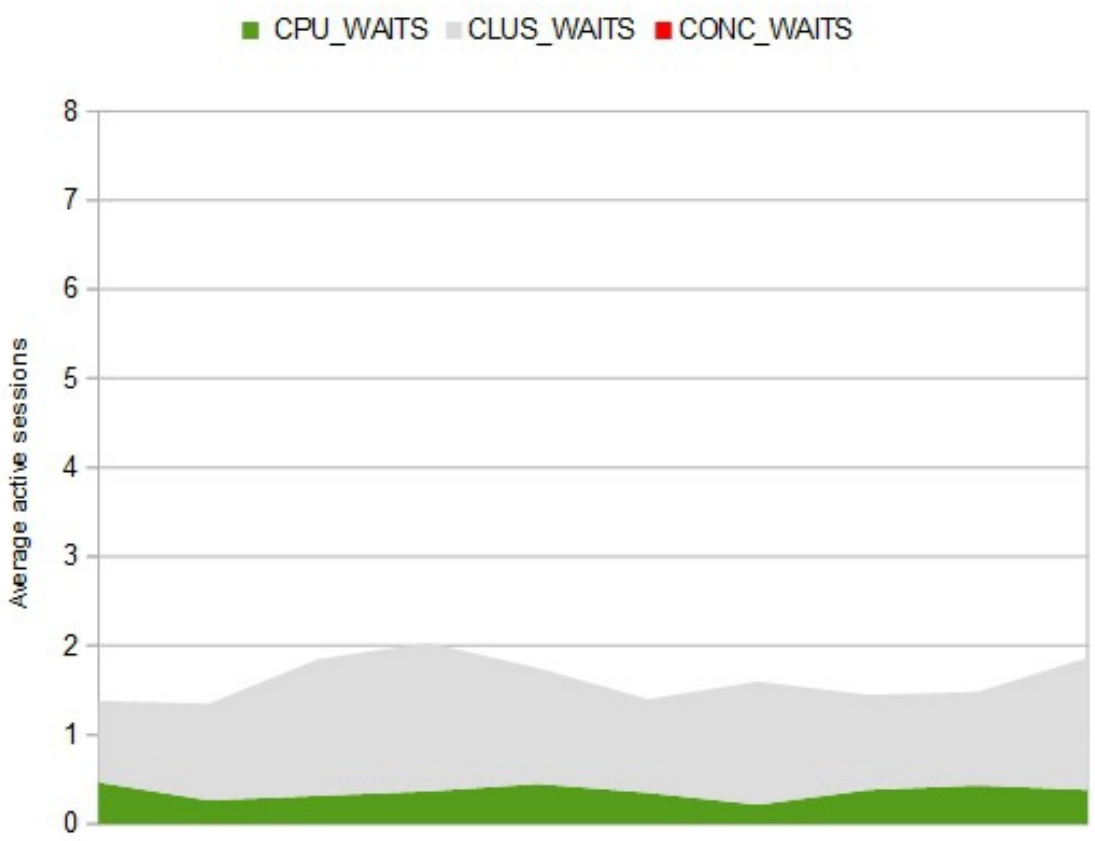
c) Delete rows and load correct data

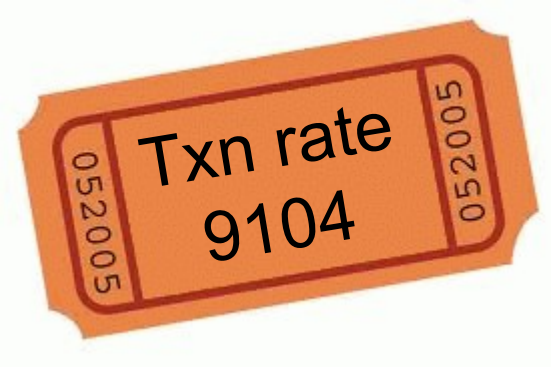
```
   BLOCKNO   COUNT(*)
---------- ----------
    239046          2
    239044          2
    239043          2
    278663          2
    239045          2
    239047          2
```

Results #5 - Minimize Records Per Block

o·su·mo

EVENT	WAITS	P1	P2
gc cr block busy	136	5	239046
CPU	122	0	0
gc cr block busy	109	5	239044
gc cr block busy	98	5	239043
gc cr block busy	94	5	239045
gc current block busy	55	5	239043





46%

Test #5 - Minimize Records Per Block

o·su·mo

Negatives?

1. Not well documented or publicised
2. Barely visible in the data dictionary
3. Values in tab$.spare1 are not human friendly

Any positives?

1. Does as it says on the tin
2. Vanity?

In hindsight I think I should have used PCTFREE

Bonus Solution - Change The Code

o·su·mo

Recently had the opportunity to see the application from this story again

Heartbeats are now every second not every trxn

Oracle Performance Diagnostic Guide (OPDG) [ID 390374.1]

Solution Identified: Spread out the rows over more blocks

... the query must be executed less often ...

A gold star for the development team

Can we do better?

All solutions so far have > 1 row per block*

Test	Average Rows Per Block
Node Affinity	12
PCTFREE 99	2
Hash Partitioning	1.1
Hakan Factor	2

Can we do better?
What other options can we think of?

* in the real world any of the solutions covered already would be perfectly fine

Interval Partitioning

Test #6 - Interval Partitioning

o·su·mo

```
create table procstate
(
...
)
partition by range (proc_id) interval (1)
(partition p0 values less than (1));

select partition_name
from user_tab_partitions
where table_name = 'PROCSTATE';
```

```
PARTITION_NAME  HIGH_VALUE
--------------- -----------
P0              1
```

Test #6 - Interval Partitioning

o·su·mo

Insert the 12 rows for the test...

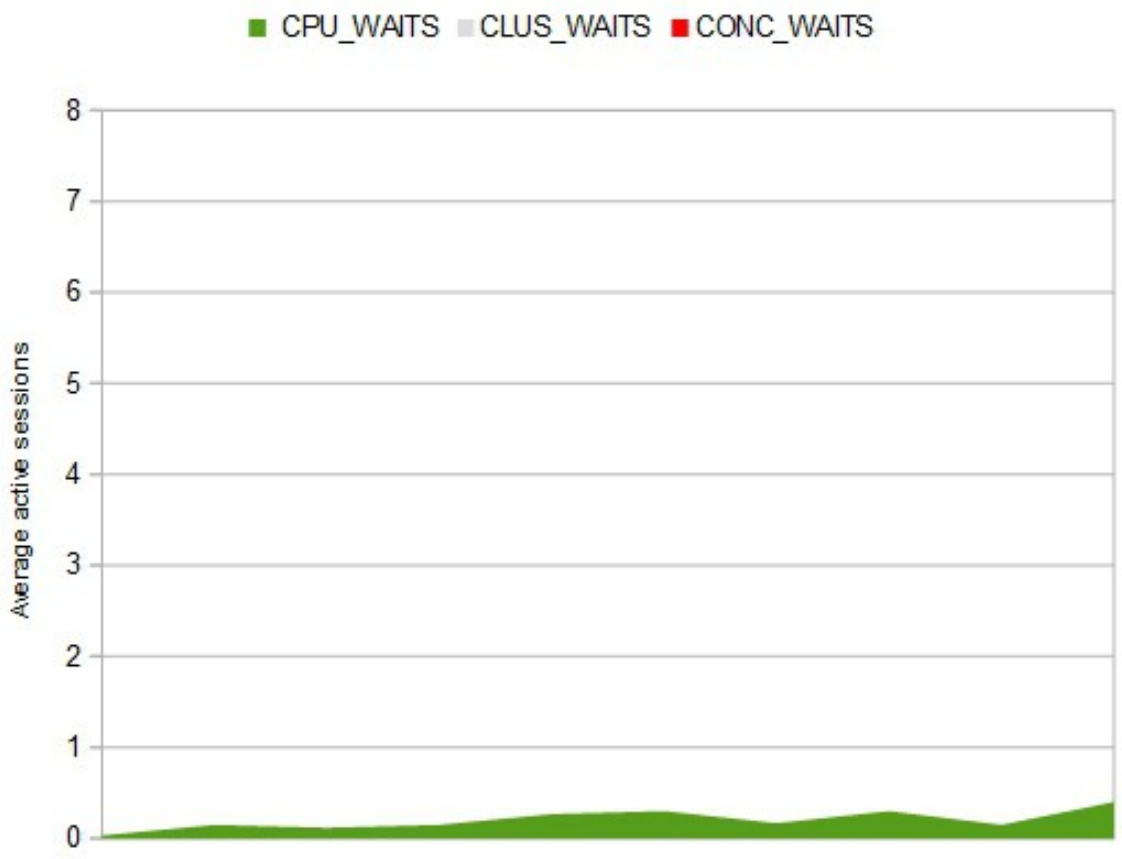
PARTITION_NAME	HIGH_VALUE
P0	1
SYS_P1029	8
SYS_P1030	5
SYS_P1031	13
SYS_P1032	11
SYS_P1033	10
SYS_P1034	12
SYS_P1035	9
SYS_P1036	4
SYS_P1037	2
SYS_P1038	3
SYS_P1039	7
SYS_P1040	6

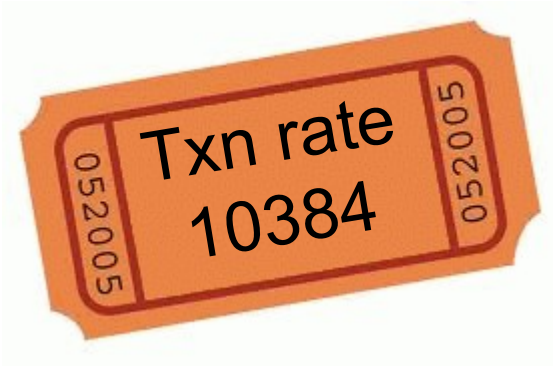
BLOCKNO	COUNT(*)
249650	1
256818	1
246578	1
255794	1
254770	1
252722	1
245554	1
247602	1
251698	1
250674	1
248626	1
253746	1

Results #6 - Interval Partitioning

o·su·mo

EVENT	WAITS	P1	P2
CPU	121	0	0
latch free	2	2157329264	94
CPU	1	2179275824	29





66%

Test #6 - Interval Partitioning

Benefit over hash partitioning - new processes will generate new partitions automatically

Still cost option and Enterprise Edition

11g only

List Partitioning & Row Movement

Test #7 – Row Movement

Typically used for FLASHBACK TABLE or SHRINK SPACE however...

“The ENABLE ROW MOVEMENT clause is specified to allow the automatic migration of a row to a new partition if an update to a key value is made that would place the row in a different partition.”

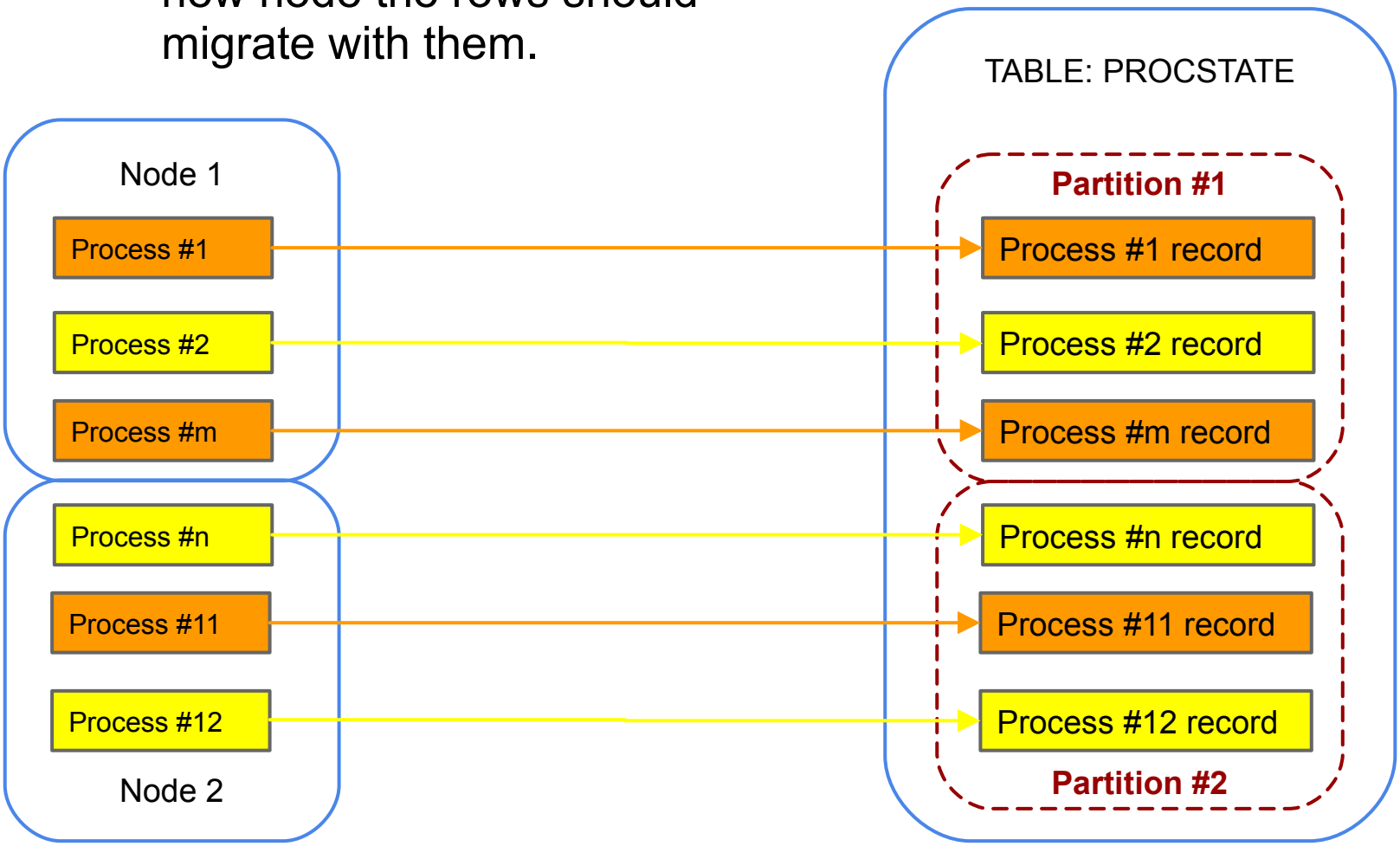
http://docs.oracle.com/cd/E11882_01/server.112/e25523/part_admin001.htm#i1006455

```
CREATE TABLE sales
...
PARTITION
...
ENABLE ROW MOVEMENT;
```

Test #7 – Row Movement

o·su·mo

If processes are started on a new node the rows should migrate with them.



Test #7 – Row Movement

o·su·mo

```
create table procstate
(  proc_id   number(2)    not null
...
,  instance  number(2)
      default
          to_number(
              sys_context('USERENV','INSTANCE')
          ) not null
)
partition by list (instance)
( partition p1 values (1)
, partition p2 values (2))
enable row movement;
```

Test #7 – Row Movement

o·su·mo

```
create or replace trigger procstate_trg
before update on procstate for each row
when (nvl(old.instance,0) <>
 to_number(sys_context('USERENV','INSTANCE')))
begin
 :new.instance :=
 to_number(sys_context('USERENV','INSTANCE'));
end;
/
```

HEALTH WARNING

Not recommending use of triggers – just keeping with the spirit of not modifying code.

Triggers are usually a bad idea.

http://www.oracle.com/technetwork/issue-archive/2008/08-sep/o58asktom-101055.html

Test #7 – Row Movement

o·su·mo

```
select dbms_rowid.ROWID_BLOCK_NUMBER(rowid) blockno, count(*)
from procstate group by dbms_rowid.ROWID_BLOCK_NUMBER(rowid);

   BLOCKNO   COUNT(*)
---------- ----------
    257842         12
```

All rows in one block

```
DATA_OBJECT_ID SUBOBJECT_NAME
-------------- --------------
         17846 P1
         17847 P2
```

Two partitions

```
select dbms_rowid.rowid_object(ROWID) part_id
, min(proc_id) min_id, max(proc_id) max_id
from procstate
group by dbms_rowid.rowid_object(ROWID);

   PART_ID     MIN_ID     MAX_ID
---------- ---------- ----------
     17846          1         12
```

All rows in partition “P1”

Test #7 – Row Movement

o·su·mo

```
[oracle@racn1 ~]$ ./procwork.sh 1 6
[oracle@racn2 ~]$ ./procwork.sh 7 12

   PART_ID      MIN_ID      MAX_ID
---------- ---------- ----------
     17846          1           6
     17847          7          12
```

Proc's 1-6 on “racn1”
Proc's 7-12 in “racn2”

Rows 1-6 in “P1”
Rows 7-12 in “P2”

```
[oracle@racn1 ~]$ ./procwork.sh 7 12
[oracle@racn2 ~]$ ./procwork.sh 1 6

   PART_ID      MIN_ID      MAX_ID
---------- ---------- ----------
     17846          7          12
     17847          1           6
```

Proc's 7-12 on “racn1”
Proc's 1-6 in “racn2”

Rows 7-12 in “P1”
Rows 1-6 in “P2”

Test #7 – Row Movement

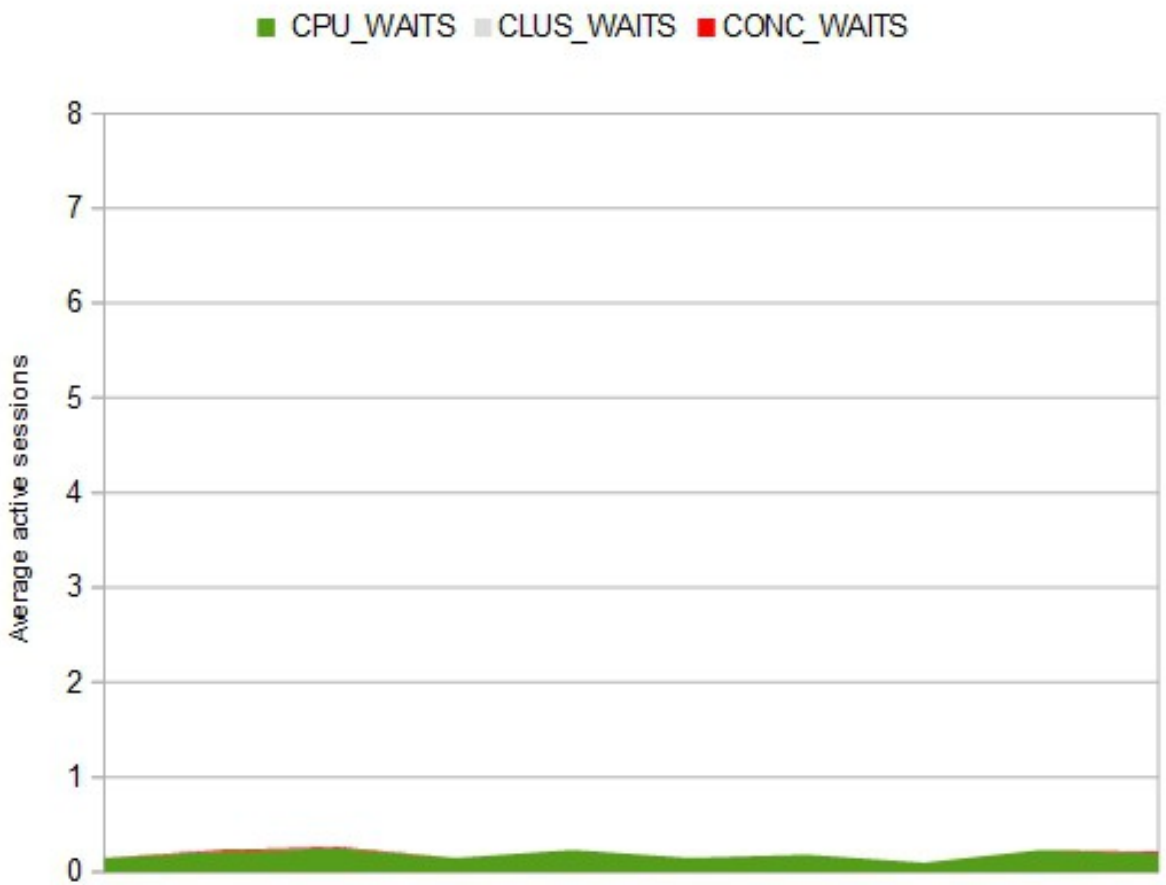
- With row movement an update of the partition key essentially becomes a delete and an insert
- Indexes will be updated when rows move therefore increased overhead
- Enabling row movement is just giving your blessing – not an overhead in itself
- As always use with care
- FAQ: Row Movement Common Questions and Problems (Doc ID 1518567.1)
- After row movement my rows were spread across their own blocks hence similar results as previous tests

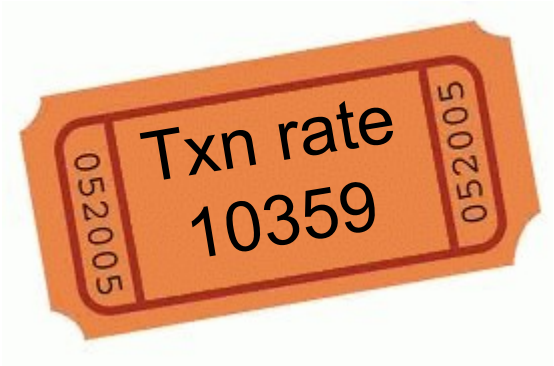
BLOCKNO	COUNT(*)
194836	1
174391	1
174386	1
195118	1
195041	1
174397	1
195183	1
174396	1
195053	1
174399	1
194988	1
174398	1

Results #7 – Row Movement

o·su·mo

EVENT	WAITS	P1	P2
CPU	121	0	0
latch: enqueue hash chains	1	2179275824	29





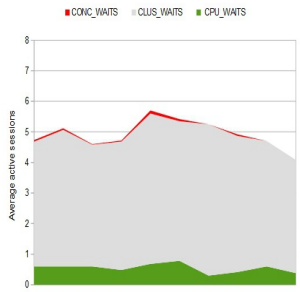
66%

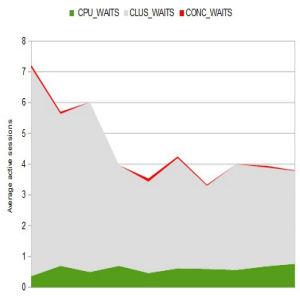

12
C

12c Results

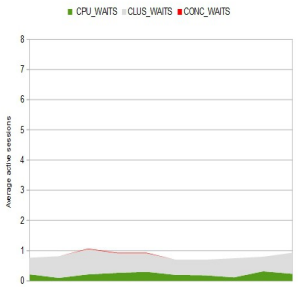
o·su·mo

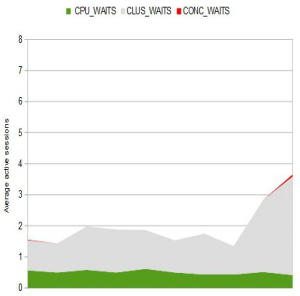
Two node heap



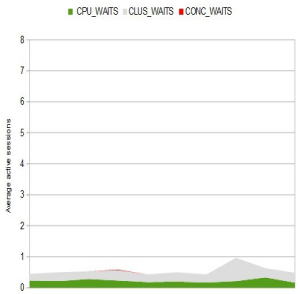


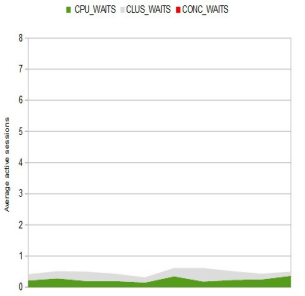
PCTFREE 99



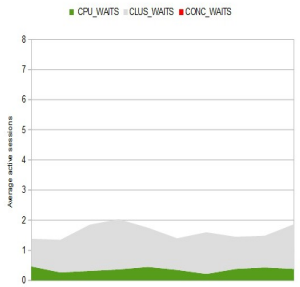


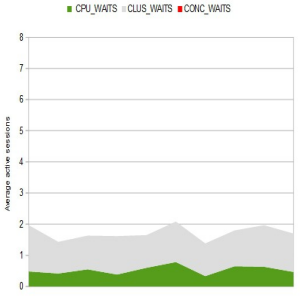
Hash Partitioned



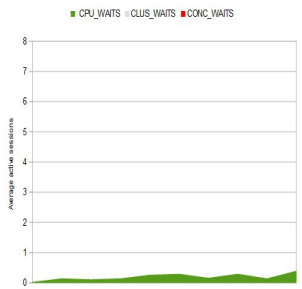


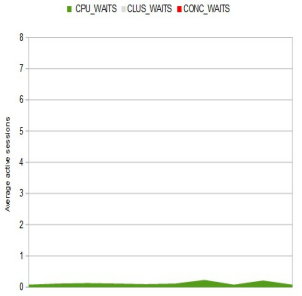
MRPB



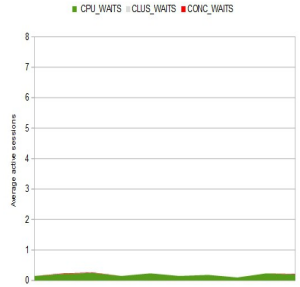


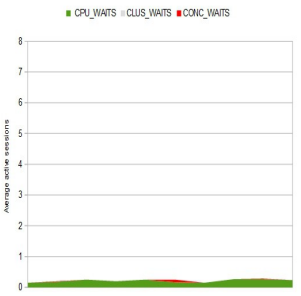
Interval Partitioned





Row Migration Partitioned





Invisible Padding Column

Test #8 – Invisible Padding Column

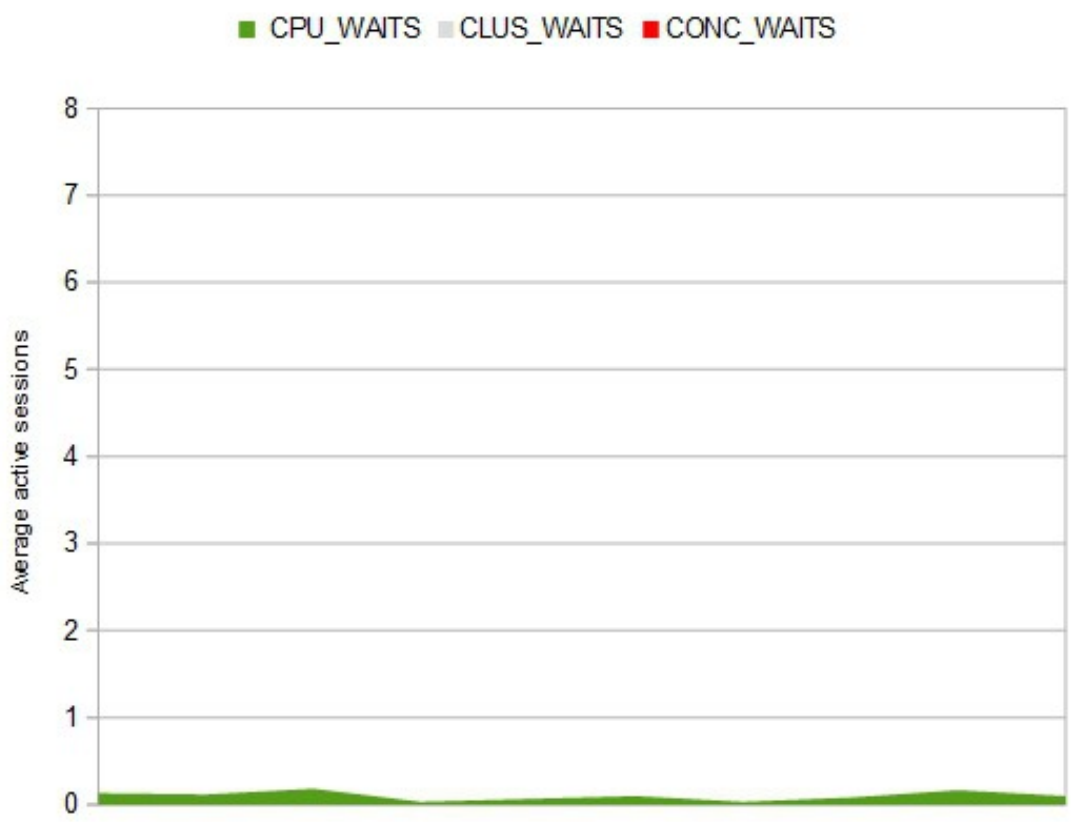
o·su·mo

```
create table procstate
(
...
,  padcol   char(100) invisible default 'x'
) pctfree 99;

desc procstate

 Name                      Null?    Type
 ------------------------- -------- -------------
 PROC_ID                   NOT NULL NUMBER(3)
 PROC_NAME                 NOT NULL VARCHAR2(10)
 PROC_STATUS               NOT NULL VARCHAR2(10)
 PROC_START                NOT NULL TIMESTAMP(6)
 LAST_HEARTBEAT                     TIMESTAMP(6)
 MSG_COUNT                          NUMBER
 ABORT                              CHAR(1)
 PADCOL                             CHAR(100)
```

Test #8 – Invisible Padding Column

o·su·mo

BLOCKNO	COUNT(*)
1208	1
1174	1
1172	1
1215	1
1209	1
1212	1
1214	1
1173	1
1175	1
1210	1
1211	1
1171	1



Summary

- Many ways to reduce the density of rows
  - Partitioning
  - PCTFREE
  - MINIMIZE RECORDS_PER_BLOCK
  - Padding columns
- Not many reasons why you'd want to
- Results in this presentation are for an indication only – they are exaggerated
- In this particular case I'd go with PCTFREE, but the other options are all good food-for-thought

The End

@neiljdba

neil@osumo.co.uk

Audience Contributions...

- <12c hide a padding column in a view (T.Hasler)
- Insert padding rows using a trigger (D.Dawson)
- APPEND_VALUES hint on insert (M.McKay-Dirden)
- Single table hash cluster (J.Lewis)

Single Table Hash Cluster

Test #9 – Single Table Hash Cluster

o·su·mo

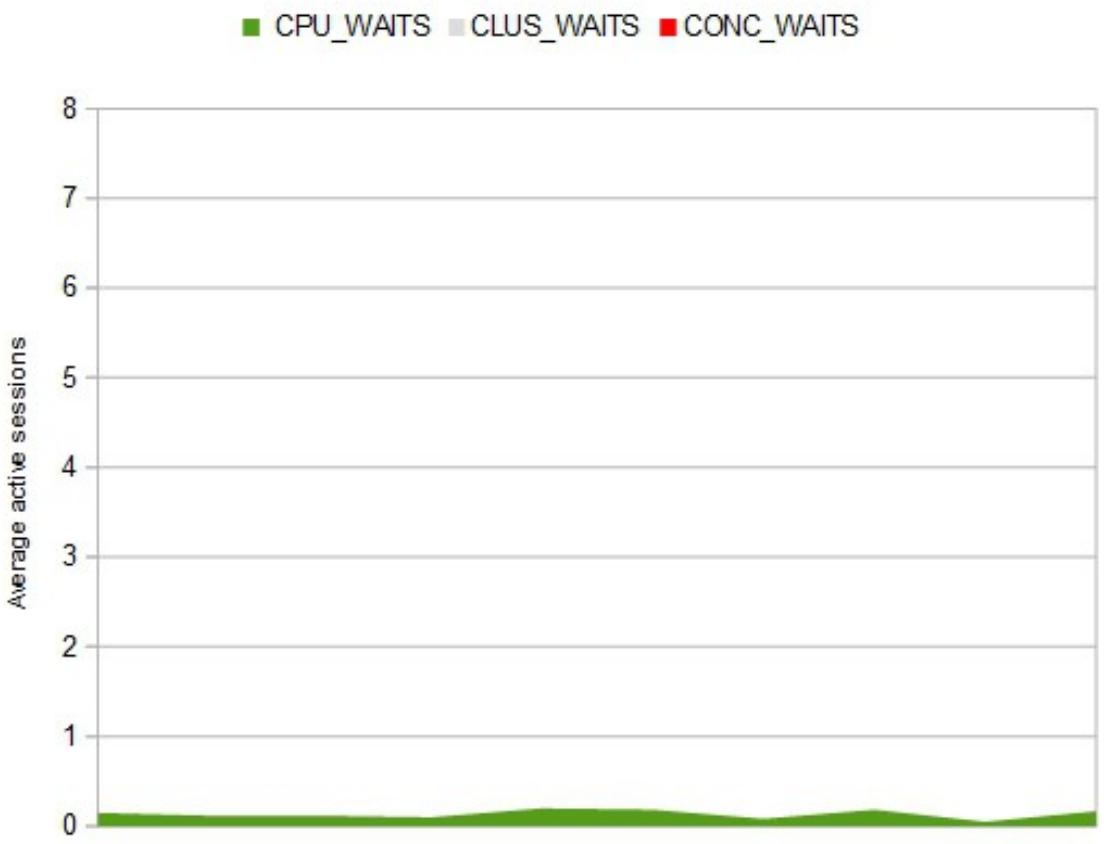
```
create cluster procstate_cluster
(proc_id number(3))
size 8000 single table hashkeys 100;

create table procstate
(
...
,
)
cluster procstate_cluster(proc_id);
```

```
    BLOCKS
----------
       110
```

Test #9 – Single Table Hash Cluster

o·su·mo

BLOCKNO	COUNT(*)
172398	1
172361	1
239067	1
172402	1
172365	1
239079	1
172389	1
172352	1
239075	1
172356	1
239071	1
172394	1



The End

o·su·mo

@neiljdba

neil@osumo.co.uk

Test code:
http://www.osumo.co.uk/share/contentious-small-tables-testcode.txt

Oracle Doc link for found when searching for "records_for_block_clause":
http://docs.oracle.com/cd/E11882_01/server.112/e26088/statements_3001.htm#i2085284

Oracle Performance Diagnostic Guide (OPDG) [ID 390374.1]

Bitmap Index Internals:
http://www.juliandyke.com/Presentations/BitmapIndexInternals.ppt

Bitmap Indexes & Minimize Records_Per_Block (Little Wonder)
http://richardfoote.wordpress.com/2011/07/19/bitmap-indexes-minimize-records_per_block-little-wonder/