Block Level Tuning; NYOUG 2005

Rich Niemiec, TUSC

(Thanks: Scott Martin, Tirth, Andy, Nitin, John, Kevin)

Audience Knowledge

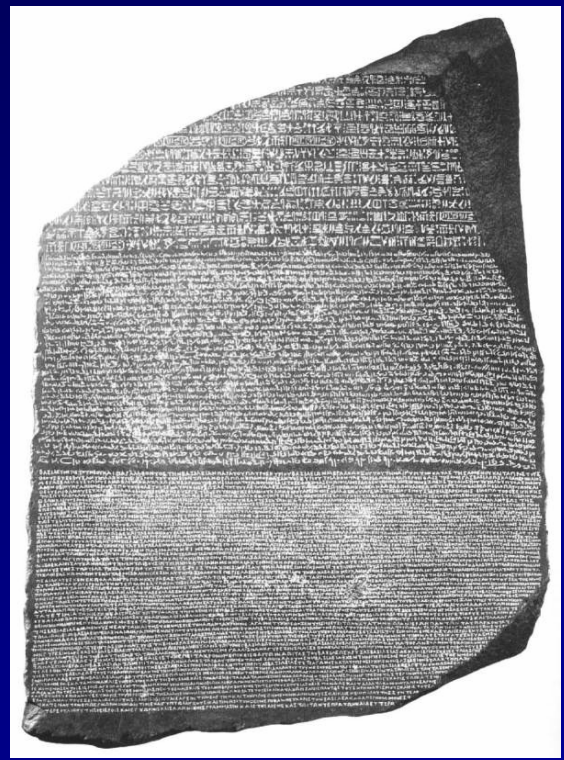
- Oracle8*i* Experience ?
- Oracle9*i* Experience ?
- Oracle9*i* RAC Experience?
- Oracle10*g* Experience?

- Goals
  - Overview of Block-level tools
  - Cool things you can do
- Non-Goals
  - Covering ALL exceptions

Overview

- Terminology
- What you're Waiting on
- Block Dumps & Block Information
- Transactions moving through Oracle
  - Buffer Cache
  - ITL & UNDO tie
  - Delayed block cleanout

Terminology



Statspack - Top Wait Events Things to look for…

Wait Problem	Potential Fix
Session Logical Reads	All reads cached in memory. Includes both consistent gets and also the db block gets.
Consistent Gets	These are the reads of a block that are in the cache. They are NOT to be confused with consistent read (cr) version of a block in the buffer cache (usually the current version is read).
Db block gets	These are block gotten to be changed. MUST be the CURRENT block and not a cr block.
Db block changes	These are the db block gets (above) that were actually changed.
Physical Reads	Blocks not read from the cache. Either from disk, disk cache or O/S cache; there are also physical reads direct which bypass cache using Parallel Query (not in hit ratios).

Statspack – Instance Activity

Statistic	Total	per Second	per Trans
branch node splits	7,162	0.1	0.0
consistent gets	12,931,850,777	152,858.8	3,969.5
current blocks converted for CR	75,709	0.9	0.0
db block changes	343,632,442	4,061.9	105.5
db block gets	390,323,754	4,613.8	119.8
hot buffers moved to head of LRU	197,262,394	2,331.7	60.6
leaf node 90-10 splits	26,429	0.3	0.0
leaf node splits	840,436	9.9	0.3
logons cumulative	21,369	0.3	0.0
physical reads	504,643,275	5,965.1	154.9
physical writes	49,724,268	587.8	15.3
session logical reads	13,322,170,917	157,472.5	4,089.4
sorts (disk)	4,132	0.1	0.0
sorts (memory)	7,938,085	93.8	2.4
sorts (rows)	906,207,041	10,711.7	278.2
table fetch continued row	25,506,365	301.5	7.8
table scans (long tables)	111	0.0	0.0
table scans (short tables)	1,543,085	18.2	0.5

Tuning the RAC Cluster Interconnect

RAC issues are the same times TWO!

Top 5 Timed Events

Event	Waits	Time (s)	% Total Ela Time
global cache cr request	820	154	72.50
CPU time		54	25.34
global cache null to x	478	1	.52
control file sequential read	600	1	.52
control file parallel write	141	1	.28

- Transfer times excessive from other instances in the cluster to this instance.
- Could be due to network problems or buffer transfer issues.

Current & CR Versions

- Buffer hash table x$bh holds headers (hash chain protected by a CBC latch) point to db_block buffers in memory.
- For a given block - Only one block is CURRENT and no more than 5 other CR versions of the block (as of V9).
- For DML, you need the CURRENT version.
- For query, you can use the CURRENT version if not being used and/or build a CONSISTENT READ (CR) version by applying and undo needed. This may include reading the ITL, mapping to the UNDO HEADER, but the ITL also maps directly to the UNDO BLOCK and applying the UNDO to get the correct CR version that you need.
- Links for the LRU & LRU-W (working set used for buffer replacement) are maintained in the buffer headers.

Biggest Problems

- The SQL, of course… especially reads of full indexes tables and others.
- Hot blocks… hot blocks can cause latching issues. Bad SQL or bad indexes causes hot blocks (scanning through the same large index). **Improved in 10g (shared latches).**
- Not enough freelists or not using ASSM.
- Not enough initrans for multiple DML to the same block (pctfree not high enough to auto-generate). Or too many (each ITL costs 24 bytes).
- Slow I/O subsystem or poor disk caching or not enough paths and readers/writers colliding.
- Not on latest version so can't use great new features!

What are you Waiting on?

Statspack - Top Wait Events Things to look for…

Wait Problem	Potential Fix
Sequential Read	Indicates many index reads – tune the code (especially joins); Faster I/O
Scattered Read	Indicates many full table scans – tune the code; cache small tables; Faster I/O
Free Buffer	Increase the DB_CACHE_SIZE; shorten the checkpoint; tune the code to get less dirty blocks, faster I/O, use multiple DBWR's.
Buffer Busy	Segment Header – Add freelists (if inserts) or freelist groups (esp. RAC). Use ASSM.

Statspack - Top Wait Events Things to look for…

Wait Problem	Potential Fix
Buffer Busy	Data Block – Separate 'hot' data; potentially use reverse key indexes; fix queries to reduce the blocks popularity, use smaller blocks, I/O, Increase initrans and/or maxtrans (this one's debatable) Reduce records per block.
Buffer Busy	Undo Header – Add rollback segments or increase size of segment area (auto undo)
Buffer Busy	Undo block – Commit more (not too much) Larger rollback segments/area. Try to fix the SQL.

Statspack - Top Wait Events Things to look for…

Wait Problem	Potential Fix
Enqueue - ST	Use LMT's or pre-allocate large extents
Enqueue - HW	Pre-allocate extents above HW (high water mark.)
Enqueue – TX	Increase initrans and/or maxtrans (TX4) on (transaction) the table or index. Fix locking issues if TX6. Bitmap (TX4) & Duplicates in Index (TX4).
Enqueue - TM (trans. mgmt.)	Index foreign keys; Check application locking of tables. DML Locks.

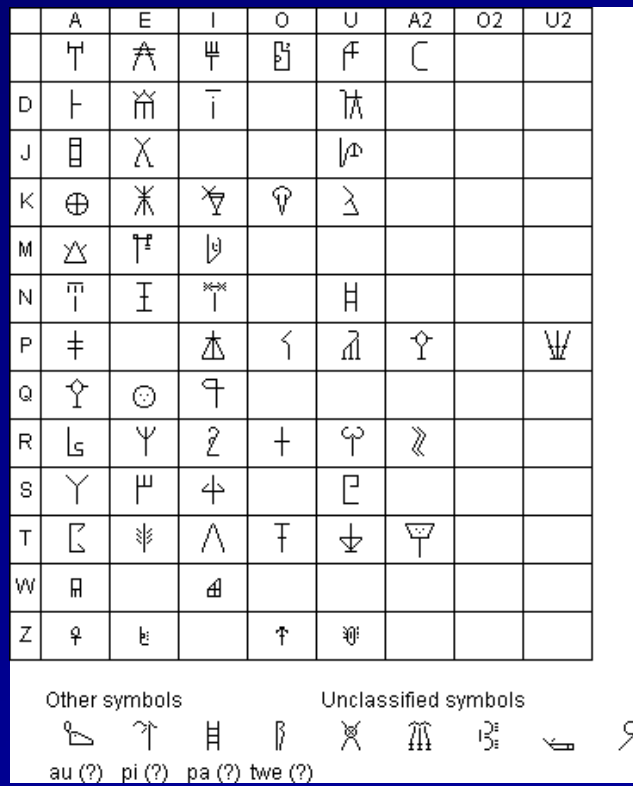
Statspack - Top Wait Events Things to look for…

Wait Problem	Potential Fix
CBC Latches	Cache Buffers Chains Latches – Reduce the length of the hash chain (less copies) by reducing block's popularity. Increase the latches by increasing buffers. Use Oracle SQ generator.
LRU Chain Latch	This latch protects the LRU list when a user needs the latch to scan the LRU chain for a buffer. When a dirty buffer is encountered it is linked to the LRU-W. When adding, moving, or removing a buffer this latch is needed.

Block Dumps

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com



Last Resort - Block Dumps


```
SQL> desc emp1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(4)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(9)
 MGR                                                NUMBER(4)
 HIREDATE                                           DATE
 SAL                                                NUMBER(7,2)
 COMM                                               NUMBER(7,2)
 DEPTNO                                             NUMBER(2)
```

Last Resort - Block Dumps

```
select  *
from    emp
where   ename = 'MILLER';
```

```
EMPNO ENAME   JOB       MGR  HIREDATE  SAL COMM DEPTNO
----------    ----------- ---------- ---------  -------- ---------- ----------
  7934    MILLER   CLERK   7782 23-JAN-82       1300                    10
```

Last Resort - Block Dumps

```
select  file_id, block_id, blocks
from    dba_extents
where   segment_name = 'EMP'
and     owner = 'SCOTT';

FILE_ID     BLOCK_ID     BLOCKS
----------  -----------  ----------
         1        50465           3
```

Last Resort - Block Dumps

```
ALTER SYSTEM DUMP DATAFILE 5 BLOCK 50465
   /
ALTER SYSTEM DUMP DATAFILE 5 BLOCK 50466
   /
ALTER SYSTEM DUMP DATAFILE 5 BLOCK 50467
   /
```

Or…

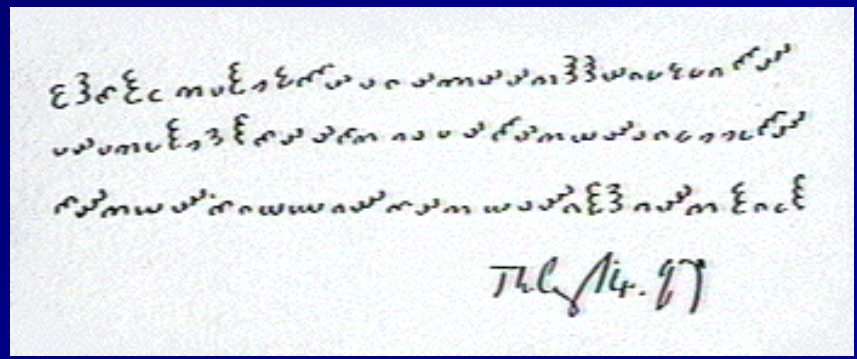
```
ALTER SYSTEM DUMP DATAFILE 5 BLOCK MIN
   50465 BLOCK MAX 50467;
```

(Puts output in user_dump_dest)

Block Dump… Getting the block number

```
select rowid,empno,
       dbms_rowid.rowid_relative_fno(rowid) fileno,
       dbms_rowid.rowid_block_number(rowid) blockno,
       dbms_rowid.rowid_row_number(rowid)   rowno, rownum,
       rpad(to_char(dbms_rowid.rowid_block_number(rowid), 'FM0xxxxxxx') || '.' ||
            to_char(dbms_rowid.rowid_row_number  (rowid), 'FM0xxx'    ) || '.' ||
            to_char(dbms_rowid.rowid_relative_fno(rowid), 'FM0xxx'    ), 18) myrid
from   emp1;

ROWID                  EMPNO     FILENO    BLOCKNO      ROWNO     ROWNUM
------------------ ---------- ---------- ---------- ---------- ----------
MYRID
------------------
AAAMfcAABAAAN0KAAA       7369          1      56586          0          1
0000dd0a.0000.0001

AAAMfcAABAAAN0KAAB       7499          1      56586          1          2
0000dd0a.0001.0001

AAAMfcAABAAAN0KAAC       7521          1      56586          2          3
0000dd0a.0002.0001
```

Block Dumps: Top Section



Block Dumps – Top Section

*** 2005-04-08 23:18:49.226
Start dump data blocks tsn: 0 file#: 1 minblk 56650 maxblk 56650
buffer tsn: 0 rdba: 0x0040dd4a (1/56650)
scn: 0x0000.003dfa58 seq: 0x01 flg: 0x00 tail: 0xfa580601
frmt: 0x02 chkval: 0x0000 type: 0x06=trans data
Block header dump: 0x0040dd4a
Object id on Block? Y
seg/obj: 0xce1c csc: 0x00.3dfa58 itc: 2 flg: O typ: 1 - DATA
fsl: 0 fnx: 0x0 ver: 0x01

#ITL's (see next slide)

Scn block was last changed at

ACTUAL Database Block Address

Changes to block w/i scn

Blocks that were dumped

Block Dumps – Top Section


```
Itl           Xid                  Uba                 Flag Lck        Scn/Fsc
0x01   0x0004.010.00000fba  0x0080003d.08b5.10  ----     4  fsc 0x009d.00000000
0x02   0x0004.016.00000fae  0x008000cc.08af.34  C---     0  scn 0x0000.003deb5b
```

ITL – 2 Interested Transaction Lists

Transaction ID
Undo#.slot#.wrap#
(Undo#,slot#,seq#)

UBA:
File.block(Undo dba).sequence.record
Undo block address where last change is recorded.

Rows Locked:
4 rows deleted for this xid in this block.

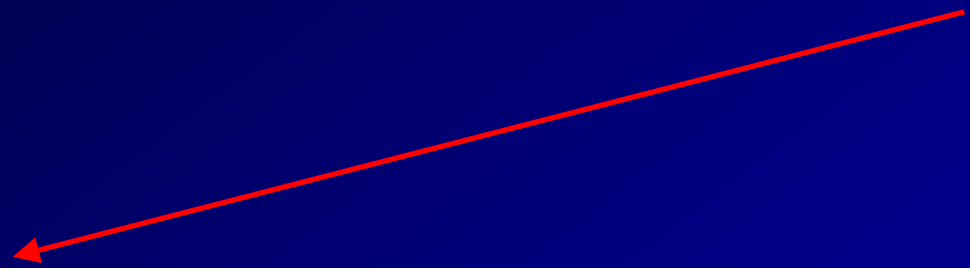
Block Dumps – Top Section

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x0004.010.00000fba	0x0080003d.08b5.10	----	4	fsc 0x009d.00000000
0x02	0x0004.016.00000fac	0x008000cc.08af.34	C---	0	scn 0x0000.003deb5b

Flag: No flag set then it's uncommitted (----)/(CBUT)
C---=Committed
-B-- = The UBA contains undo for this itl
--U-= Committed (scn is upper bound)…used by fast commits & delayed block cleanout has not occurred
---T = Transaction active at block cleanout SCN
-C-U- = Block cleaned by delayed block cleanout, and the rollback segment info overwritten. The scn will show the lowest scn that could be regenerated by the rollback segment.

Block Dumps – Top Section

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x0004.010.00000fba	0x0080003d.08b5.10	----	4	fsc 0x009d.00000000
0x02	0x0004.016.00000fae	0x008000cc.08af.34	C---	0	scn 0x0000.003deb5b



Scn/fsc:

Fsc= free space Credit = 9d (hex) = 157 (decimal) bytes

Scn = System change (commit) number

Block Dumps – Top Section

data_block_dump,data header at 0x5a1125c

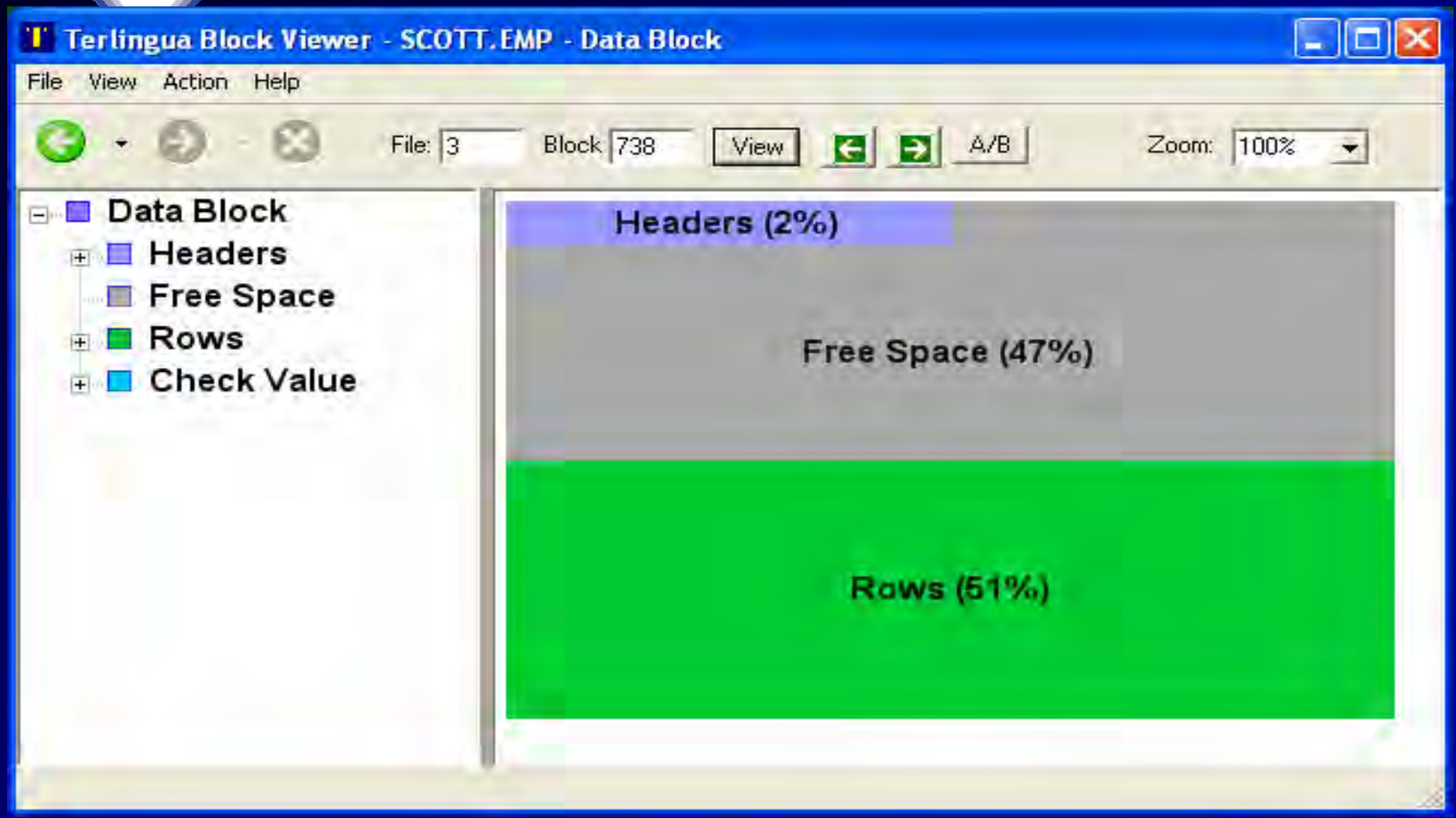
==========

tsiz: 0x1fa0	Total Size = fa0
	1 (hex) = 1 (decimal) x 16x16x16 = 4096
	F (hex) = 15 (decimal) x 16x16 = 3840
	A (hex) = 10 (decimal) x 16 = 160
	0 (hex) = 0 (decimal) x 1 = 0
	4096 + 3840 + 160 + 0 = 8096 (about 8K)
hsiz: 0x2e	Header Size = 2e = 46 bytes
pbl: 0x05a1125c	Pointer to block buffer holding the block
bdba: 0x0040dd4a	(Relative) database block address

Block Dumps – Data Section

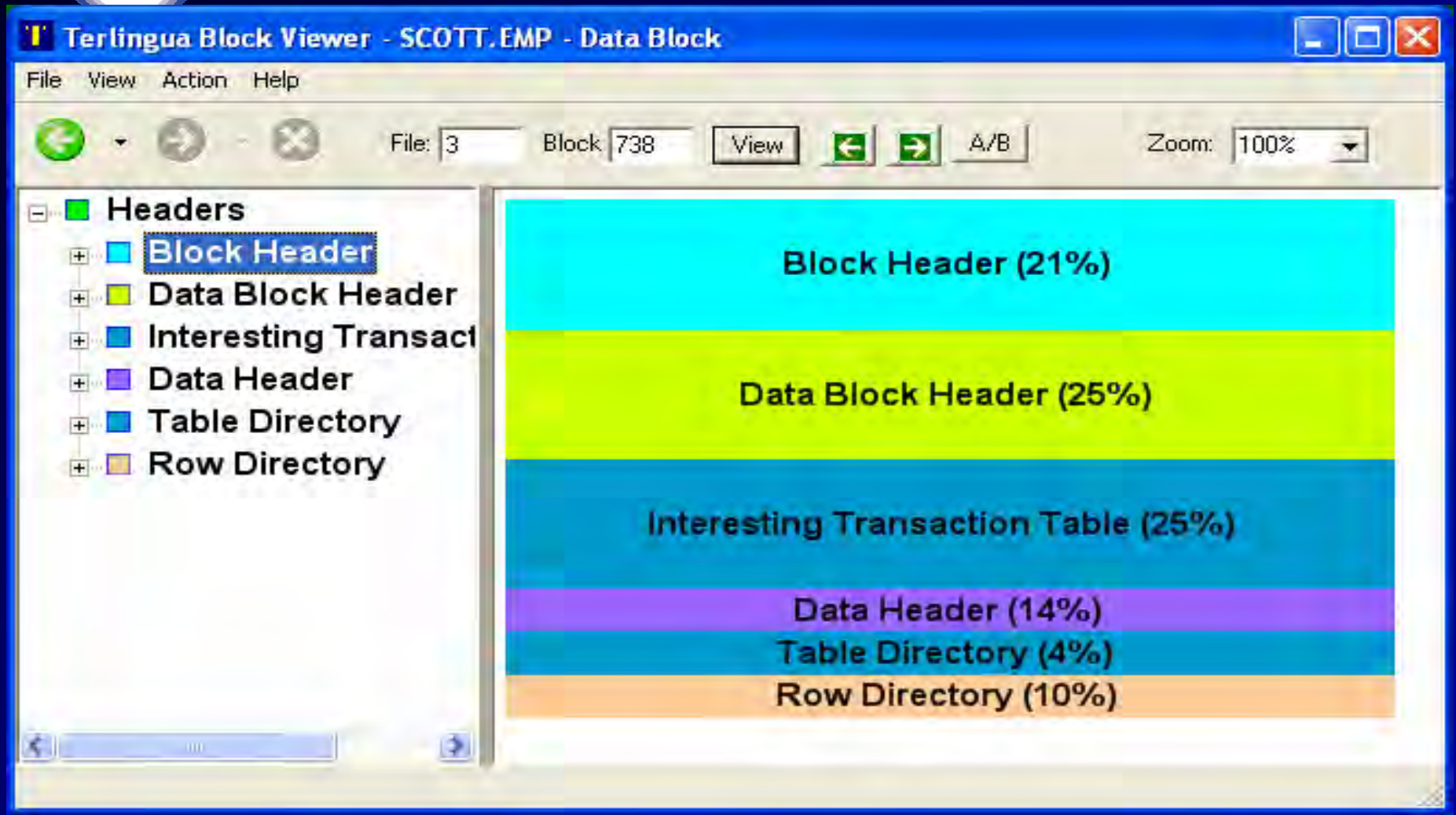
ntab=1		Number of tables= 1 (2+ for clusters)
nrow=14		Number of rows = 14
frre=-1		First free row index entry – 1 (add 1)
fsbo=0x2e		Free space begin offset
fseo=0x18fb		Free space end offset
avsp=0x1d3b		Available block space = 1d3b = 7483
tosp=0x1de0		Space avail. post commit=1de0=7648
0xe:pti[0]	nrow=14	offs=0 Table Info
0x12:pri[0]	offs=0x18fb	(6395) Row Info Record 0
0x14:pri[1]	offs=0x1921	(6433) Row Info Record 1
…		Row Info Records 2–13
block_row_dump:		Row Data is Next!

The familiar emp table Terlingua software

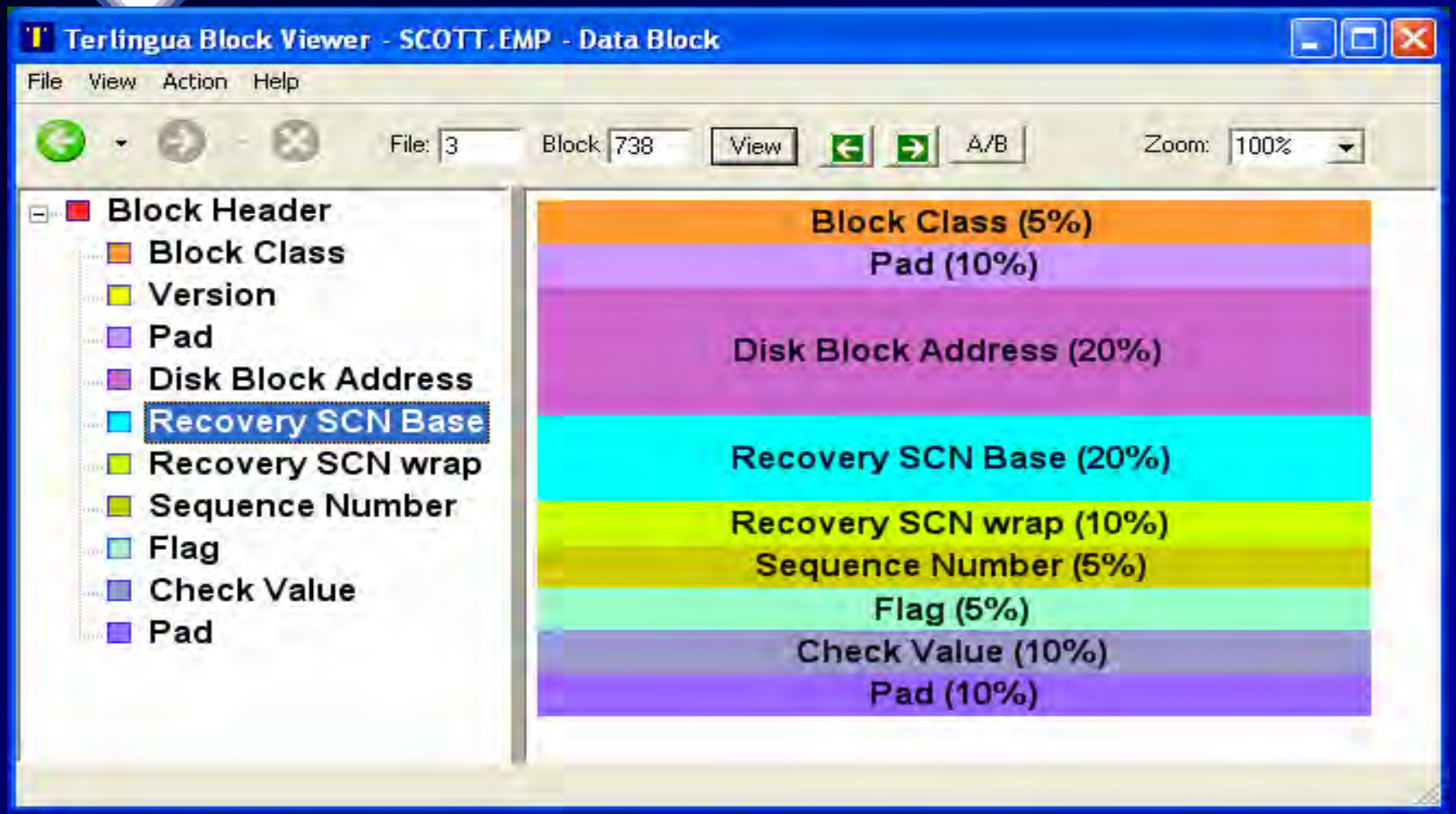


* Terlingua Software

Headers

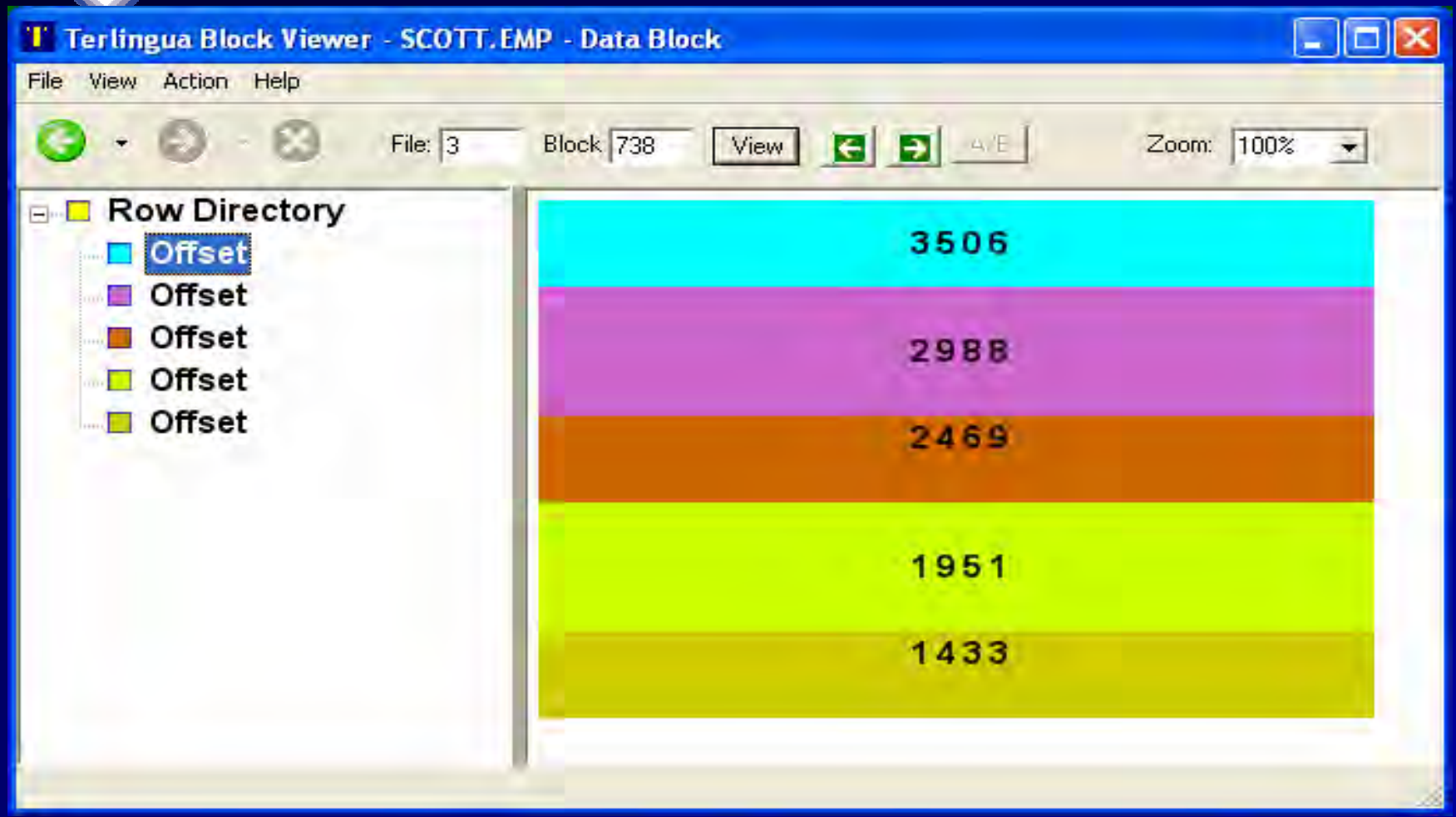

Terlingua Block Viewer - SCOTT.EMP - Data Block
File View Action Help
File: 3
Block 738
View
A/B
Zoom: 100%
Headers
Block Header
Data Block Header
Interesting Transact
Data Header
Table Directory
Row Directory
Block Header (21%)
Data Block Header (25%)
Interesting Transaction Table (25%)
Data Header (14%)
Table Directory (4%)
Row Directory (10%)

Block header



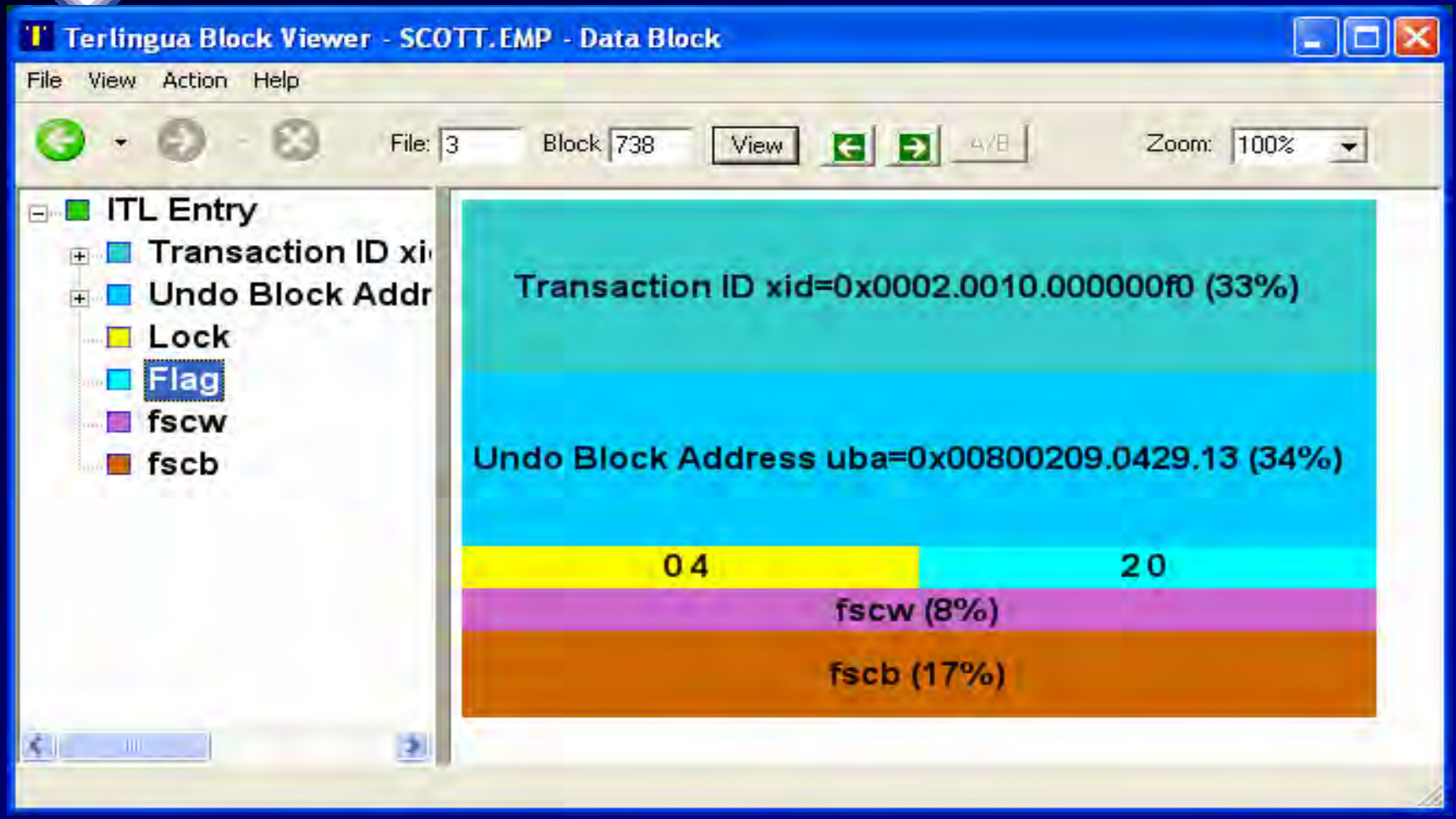
* Terlingua Software

Row directory



* Terlingua Software

Interested Transaction List (ITL)



* Terlingua Software

Block Dump: Data Section

Block Dumps – output from udump

tab 0, row 13, @0x1b0b
tl: 39 fb: --H-FL-- lb: 0x0 cc: 8
col 0: [3] c2 50 23
col 1: [6] 4d 49 4c 4c 45 52
col 2: [5] 43 4c 45 52 4b
col 3: [3] c2 4e 53
col 4: [7] 77 b6 01 17 01 01 01
col 5: [2] c2 0e
col 6: *NULL*
col 7: [2] c1 0b…

Block Dumps – Data Section

DUMP OUTPUT:

tab 0, row 13, @0x1b0b

tl: 39 fb: --H-FL-- lb: 0x0 cc: 8 (row header)

Table = this data is for table 0

Row 13 = 14th Row (0-13 total rows)

Offset: 1b0b (in Hex) – Offset from header

tl: Total bytes of row plus the header = 39

Block Dumps – Data Section

DUMP OUTPUT:

tab 0, row 13, @0x1b0b

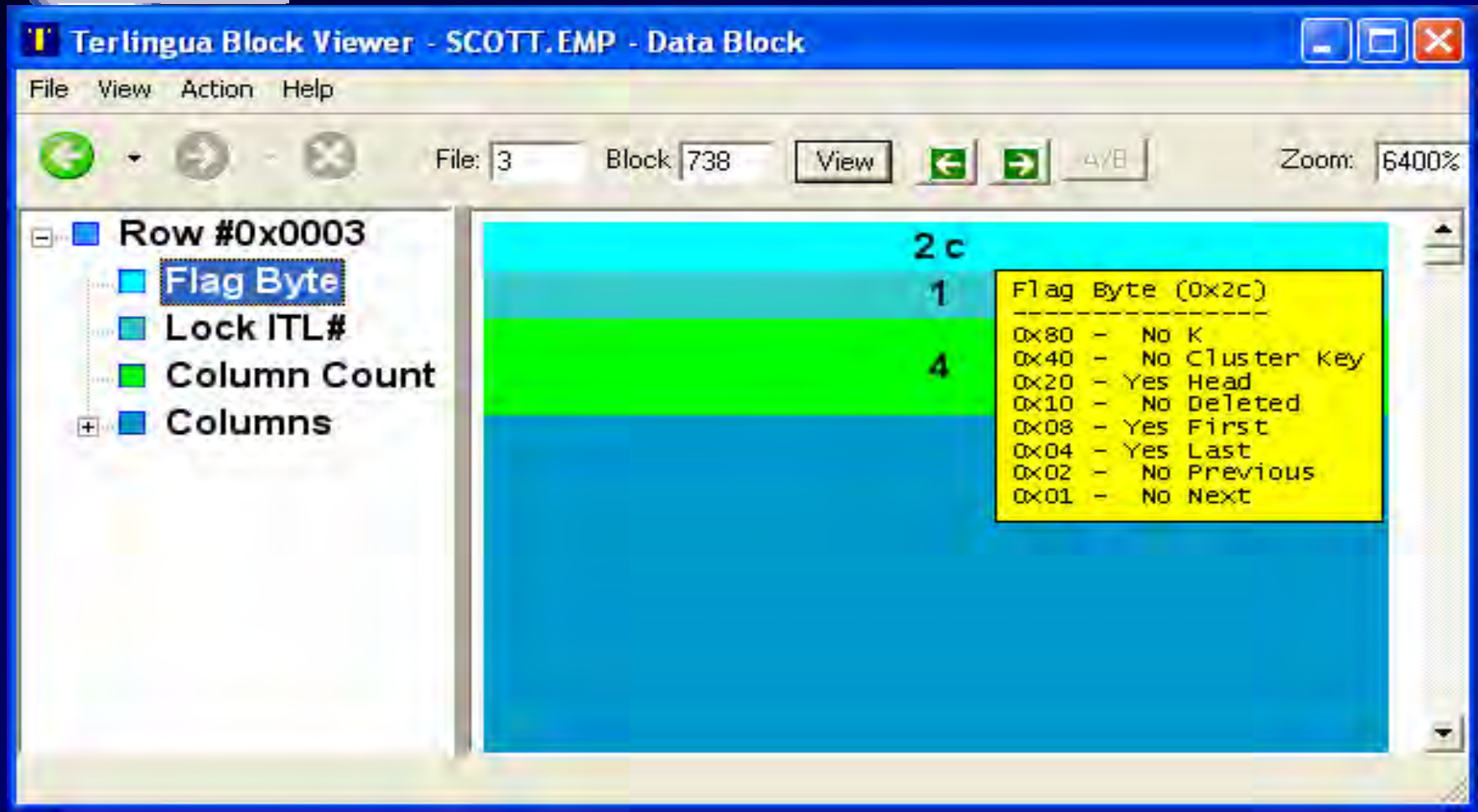
tl: 39 fb: --H-FL-- lb: 0x0 cc: 8

fb: --H-FL-- = flag byte; (-KCHDFLPN)

H = Head of row piece, F = First data piece, L=Last piece

D = Deleted; P= First column continues from previous piece (chaining) ; N= Last column continues in next piece;
K = Cluster Key; C = Cluster table member

Rows point back to itl



2c (Hex) = 00101100 (Binary) = --H-FL– (flags)

Block Dumps – Data Section

DUMP OUTPUT:

tab 0, row 13, @0x1b0b

tl: 39 fb: --H-FL-- lb: 0x0 cc: 8

Lb: lock byte is 1+ if this row is locked = 0 (unlocked)

cc: Column count = 8

Block Dumps – Deleted rows

DUMP OUTPUT (Deleted Row):

block_row_dump:

tab 0, row 0, @0x18fb

tl: 2 fb: --HDFL-- lb: 0x1

tab 0, row 1, @0x1921

tl: 2 fb: --HDFL-- lb: 0x1

Rows 1 &2 have been deleted! No row data is visible for the columns.

Block Dumps – Data Section

DUMP OUTPUT – EMPNO:

col 0: [3] c2 50 23

Hex to Decimal: Col0 = EMPNO = 7934

50 (Hex) = 80 (Decimal) – 1 = 79

23 (Hex) = 35 (Decimal) – 1 = 34

c2: Number in the thousands (c2 is exponent)

Block Dumps – Data Section

DUMP OUTPUT - ENAME:

col 1: [6] 4d 49 4c 4c 45 52

Hex to Character:

Col1 = ENAME = MILLER

4d (Hex) = M (Character)

49 (Hex) = I (Character)

4c (Hex) = L (Character)

4c (Hex) = L (Character)

45 (Hex) = E (Character)

52 (Hex) = R (Character)

Block Dumps – Data Section

DUMP OUTPUT - JOB:

col 2: [5] 43 4c 45 52 4b

Hex to Character:

Col2 = JOB = CLERK

43 (Hex) = C (Character)

4c (Hex) = L (Character)

45 (Hex) = E (Character)

52 (Hex) = R (Character)

4b (Hex) = K (Character)

Block Dumps – Data Section

DUMP OUTPUT – MGR:

col 3: [3] c2 4e 53

Hex to Decimal:

Col3 = MGR = 7782

4e (Hex) = 78 (Decimal) – 1 = 77

53 (Hex) = 83 (Decimal) – 1 = 82

Block Dumps – Data Section

DUMP OUTPUT - HIREDATE:

col 4: [7] 77 b6 01 17 01 01 01

Hex to Decimal: Col4 = HIREDATE = 23-JAN-82

77 (Hex) = 119 (Decimal) – 100 = 19

B6 (Hex) = 182 (Decimal) – 100 = 82

01(Hex) = 1 (Decimal) <month>

17 (Hex) = 23 (Decimal)

01 01 01 (Hex) = This is the Hour, Minute, Second

(none were entered when the date was entered…default)

Block Dumps – Data Section

DUMP OUTPUT - SAL:

col 5: [2] c2 0e

Hex to Decimal: Col5 = SAL = 1300

0e (Hex) = 14 (Decimal) – 1 = 13

c2 = add two zero's

Block Dumps – Data Section

DUMP OUTPUT - COMM:

col 6: *NULL*

Hex to Decimal:

NULL = NULL

Col6 = COMM = NULL

Block Dumps – Data Section

DUMP OUTPUT - DEPTNO:

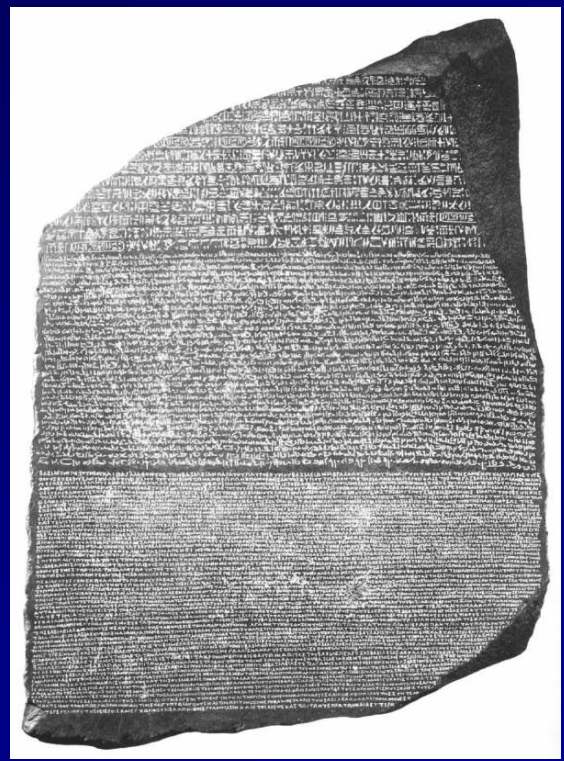
col 7: [2] c1 0b

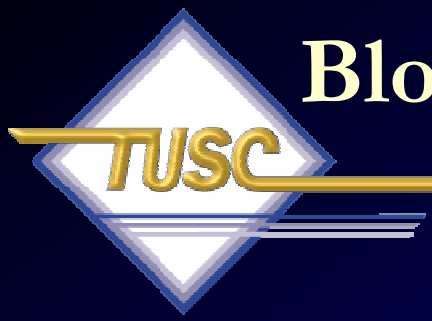
Hex to Decimal: Col7 = DEPTNO= 10

0B (Hex) = 11 (Decimal) – 1 = 10

c1 = number in the tens

Block Dump: Data Section – Other Ways





Block Dump using SELECT dump()

```
select   dump(ename) from emp1
where  ename='MILLER';

DUMP(ENAME)
----------------------------------------------------
Typ=1 Len=6: 77,73,76,76,69,82
```

Types: 1=varchar; 2=number; 12=date; 23=raw

Block Dump using SELECT dump()

Typ=1 Len=6: 77,73,76,76,69,82

Decimal to Character: ENAME = MILLER

77 (Decimal) = M (Character)

73 (Decimal) = I (Character)

76 (Decimal) = L (Character)

76 (Decimal) = L (Character)

69 (Decimal) = E (Character)

82 (Decimal) = R (Character)

Block Dump using SELECT dump() (convert it to HEX if you want)

select dump(ename.16) from emp1

where ename='MILLER';

DUMP(ENAME,16)

Typ=1 Len=6: 4d,49,4c,4c,45,52

Types: 1=varchar; 2=number; 12=date; 23=raw

Block Dump using SELECT dump() (Can even get the ename from the HEX!)

```
select  dump(ename,16), ename
from    emp1
where   dump(ename,16) like '%4d,49,4c,4c,45,52';
```

```
DUMP(ENAME,16)                          ENAME
--------------------------------------  ----------
Typ=1 Len=6: 4d,49,4c,4c,45,52          MILLER
```

Block Dump using SELECT dump()

```
select  dump(empno)
from    emp1
where   ename='MILLER';

DUMP(EMPNO)
--------------------------------------------------------------------------------
Typ=2 Len=3: 194,80,35
```

Decimal to number: EMPNO = 7934

194 (Decimal) = c2 (Hex)

80 (Decimal) – 1 = 79

35 (Decimal) – 1 = 34

Block Dump using SELECT dump()

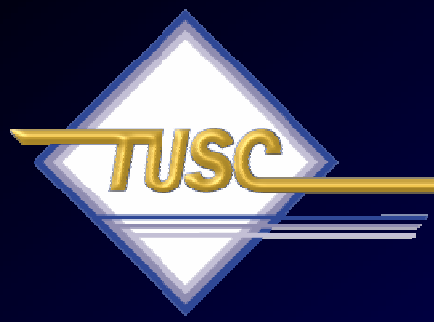
```
select  dump(hiredate)
from    emp1
where   ename='MILLER';

DUMP(HIREDATE)
--------------------------------------------------------------------------------
Typ=12 Len=7: 119,182,1,23,1,1,1
```

Decimal to Date: HIREDATE = 23-JAN-82

119 (decimal) – 100 = 19 (century)

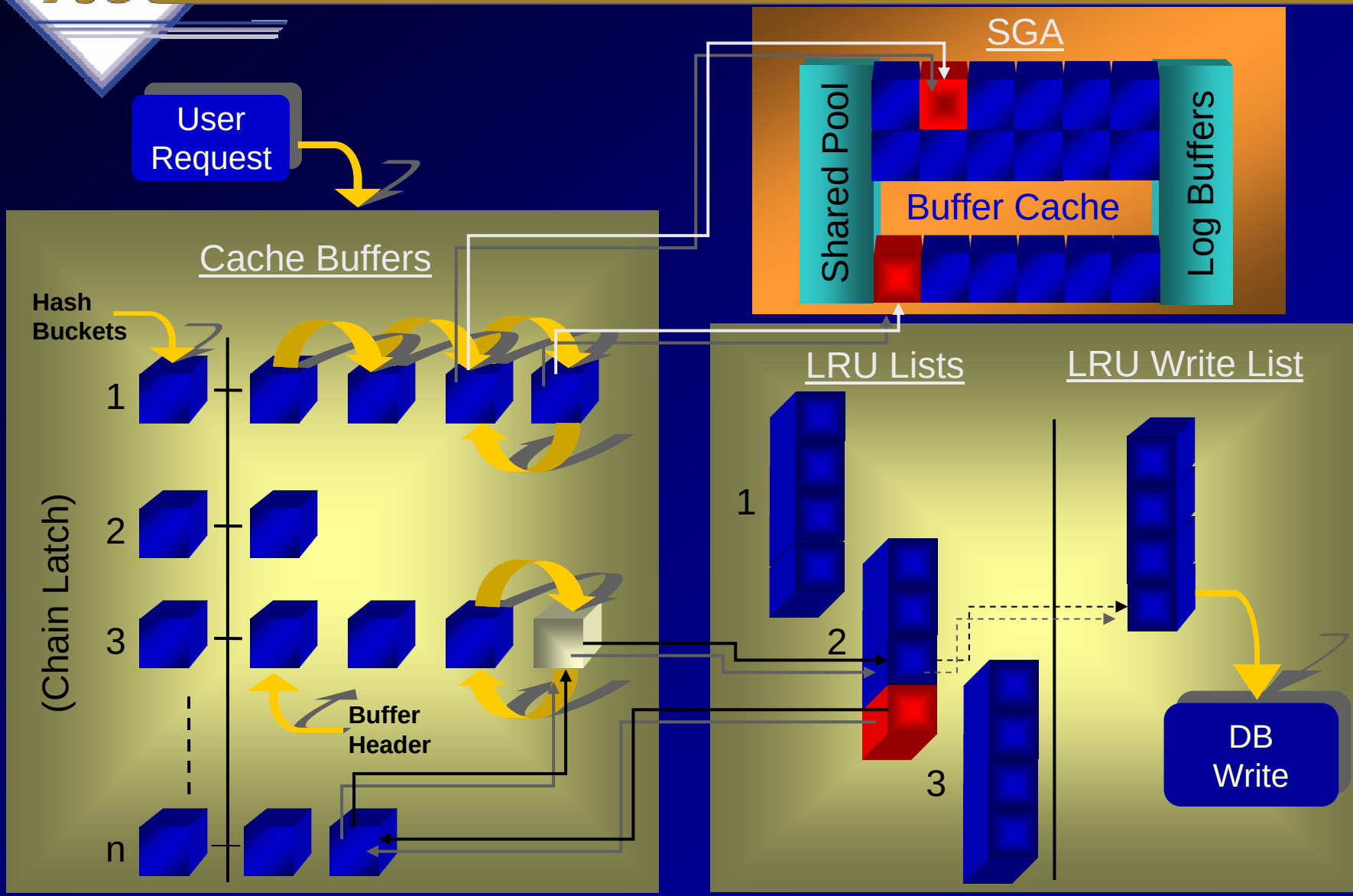
182 (decimal) – 100 = 82 (year)

1 = Month (January) ; 23 = Day



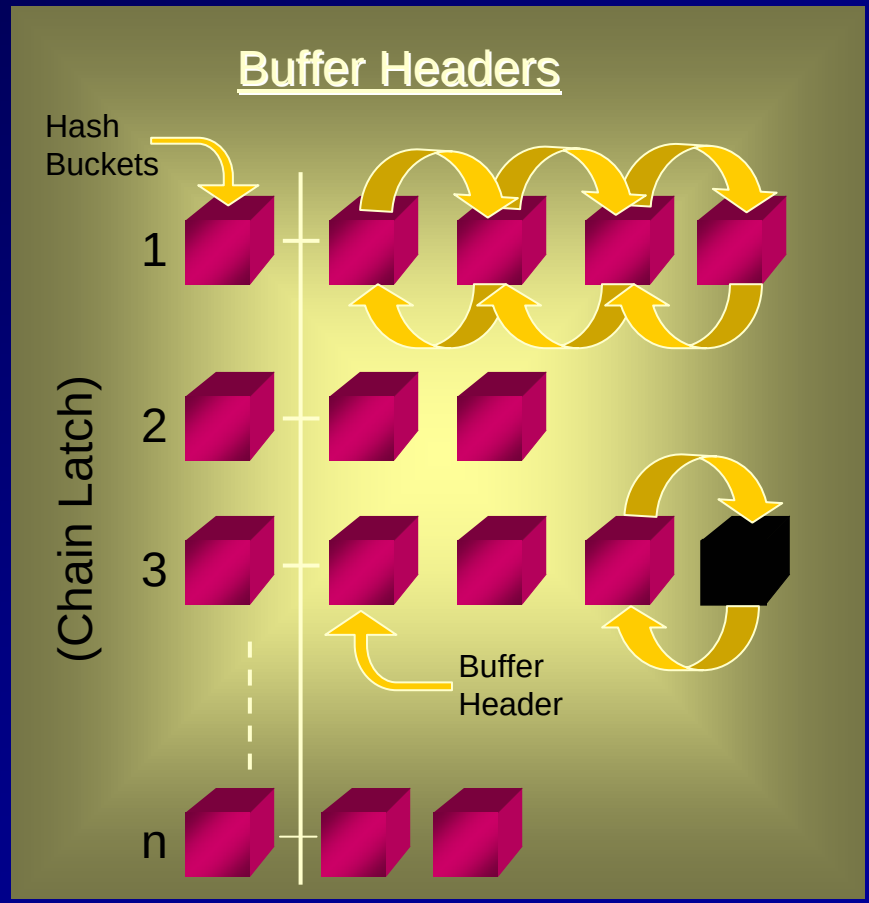
Transactions
Moving through Oracle

Row Level Locks



Working with Hash Buckets And Buffer Headers (not buffers)

- Users asks for a specific data block address.
- This is hashed with a hashing algorithm and placed in the hash bucket that it hashes to.
- It walks the hash chain using the cache buffers chain latch to find the block that it needs (curr or cr).
- There can be many versions of each block



_DB_BLOCK_HASH_BUCKETS and hashing data block addresses

Example: _DB_BLOCK_HASH_BUCKETS

(shouldn't have to change this in Oracle9i or 10g)

- Buffer hash table (x$bh) has all buffer headers for all db_block buffers.
- Buffer header ties to memory base address of the buffer.
- Buckets usually set to Prime(2*db_block_buffers)
- A prime number is often used to avoid hashing anomalies
- Objects dba (class) is hashed to a hash bucket on the hash chain
- Get enough hash buckets (_db_block_hash_buckets)
- Blocks assigned to a hash bucket and onto the hash chain
- Could have multiple blocks hashed to same chain (if both hot-issues)
- Can have multiple versions of a block on same chain
- When block is replaced (based on LRU chain) new block comes in and could be (probably will be) hashed to a different hash chain.

Example: Emp Table

Consider a user querying emp:

- First block of emp may go to chain #1
- Second block of emp may go to chain #55
- If second block of emp is updated and also has several readers than we'll get more copies. LRBA – Lowest Redo Block Address (last redo applied) for dirty block.
- Chain #55 may now have a current block and 2 CR blocks all with the same dba (data block address)
- For a given block - Only one block is CURRENT and no more than 5 other CR versions of the block (as of V9).
- All buffer headers tie to LRU, LRU-W and other LRU's (many in 10g) used for buffer replacement.

Additional LRU's / Faster!!

- LRU Main block replacement list
- LRU-W Old dirty buffers and reco/temp
- LRU-P Ping Buffer list / RAC
- LRU-XO Buffers to be written for drop/truncate
- LRU-XR Buffers to be written for reuse range
- Thread CKPT Thread Checkpoint Queue
- File CKPT File Checkpoint Queue
- Reco CKPT Reco Checkpoint
- LRU-MAIN & LRU-AUX help LRU

Query all buffer headers (state):

```
col status for a6
select   state,
 decode(state, 0, 'FREE',          /* not currently is use */
               1, 'XCUR',          /* held exclusive by this instance */
               2, 'SCUR',          /* held shared by this instance */
               3, 'CR',            /* only valid for consistent read */
               4, 'READ',          /* is being read from disk */
               5, 'MREC',          /* in media recovery mode */
               6, 'IREC',          /* in instance(crash) recovery mode */
               7, 'WRITE',         /* being written */
               8, 'PIN') status, count(*)   /* pinned */
from  x$bh
group by state;

STATE  STATUS COUNT(*)
------ ------ ----------
     1 XCUR         2001
     3 CR              3
```

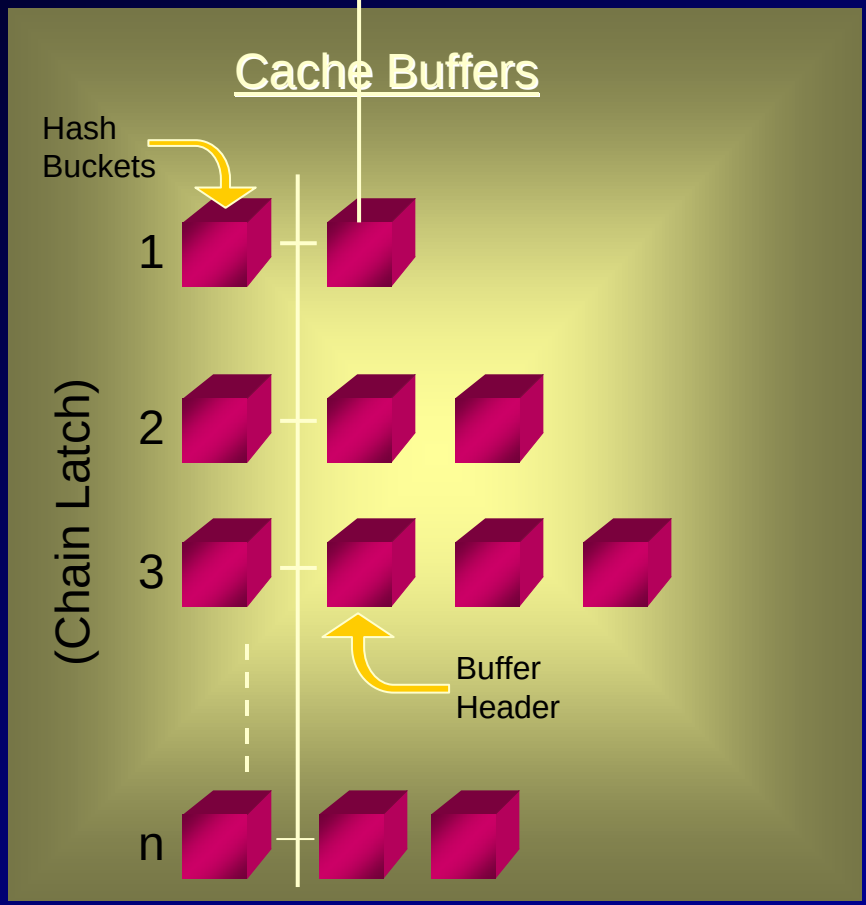
EMP1 is Block#: 56650 (all rows are in this block)

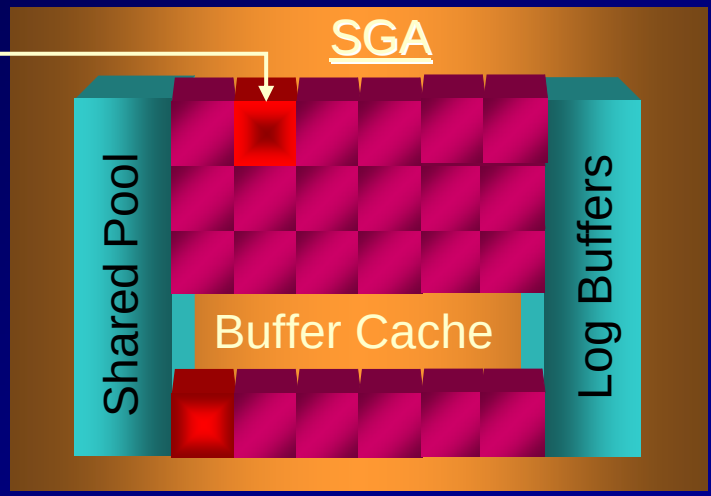

```
select rowid,empno,
       dbms_rowid.rowid_relative_fno(rowid) fileno,
       dbms_rowid.rowid_block_number(rowid) blockno,
       dbms_rowid.rowid_row_number(rowid)   rowno, rownum,
       rpad(to_char(dbms_rowid.rowid_block_number(rowid), 'FM0xxxxxxx') || '.' ||
            to_char(dbms_rowid.rowid_row_number  (rowid), 'FM0xxx'    ) || '.' ||
            to_char(dbms_rowid.rowid_relative_fno(rowid), 'FM0xxx'    ), 18) myrid
from   emp1;
ROWID                  EMPNO     FILENO    BLOCKNO      ROWNO     ROWNUM
------------------ ---------- ---------- ---------- ---------- ----------
MYRID
------------------
AAAM4cAABAAAN1KAAA       7369          1      56650          0          1
0000dd4a.0000.0001

AAAM4cAABAAAN1KAAB       7499          1      56650          1          2
0000dd4a.0001.0001
…
…
AAAM4cAABAAAN1KAAN       7934          1      56650         13         14
0000dd4a.000d.0001


14 rows selected.
```

Let's watch the EMP1 buffer header (So far it's clean and only 1 copy)

```
select    lrba_seq, state, dbarfil, dbablk, tch, flag, hscn_bas,cr_scn_bas,
    decode(bitand(flag,1), 0, 'N', 'Y') dirty,          /* Dirty bit */
    decode(bitand(flag,16), 0, 'N', 'Y') temp,          /* temporary bit */
    decode(bitand(flag,1536), 0, 'N', 'Y') ping, /* ping (to shared or null) bit */
    decode(bitand(flag,16384), 0, 'N', 'Y') stale,      /* stale bit */
    decode(bitand(flag,65536), 0, 'N', 'Y') direct,     /* direct access bit */
    decode(bitand(flag,1048576), 0, 'N', 'Y') new       /* new bit */
from      x$bh
where     dbablk = 56650
order by dbablk;
```

```
  LRBA_SEQ      STATE    DBARFIL     DBABLK        TCH       FLAG   HSCN_BAS
---------- ---------- ---------- ---------- ---------- ---------- ----------
CR_SCN_BAS D T P S D N
---------- - - - - - -
         0          1          1      56650          0   35659776 4294967295
         0 N N N N N N
```





Only ONE block on the Hash Chain!

Let's watch the EMP1 buffer header (Delete a row)

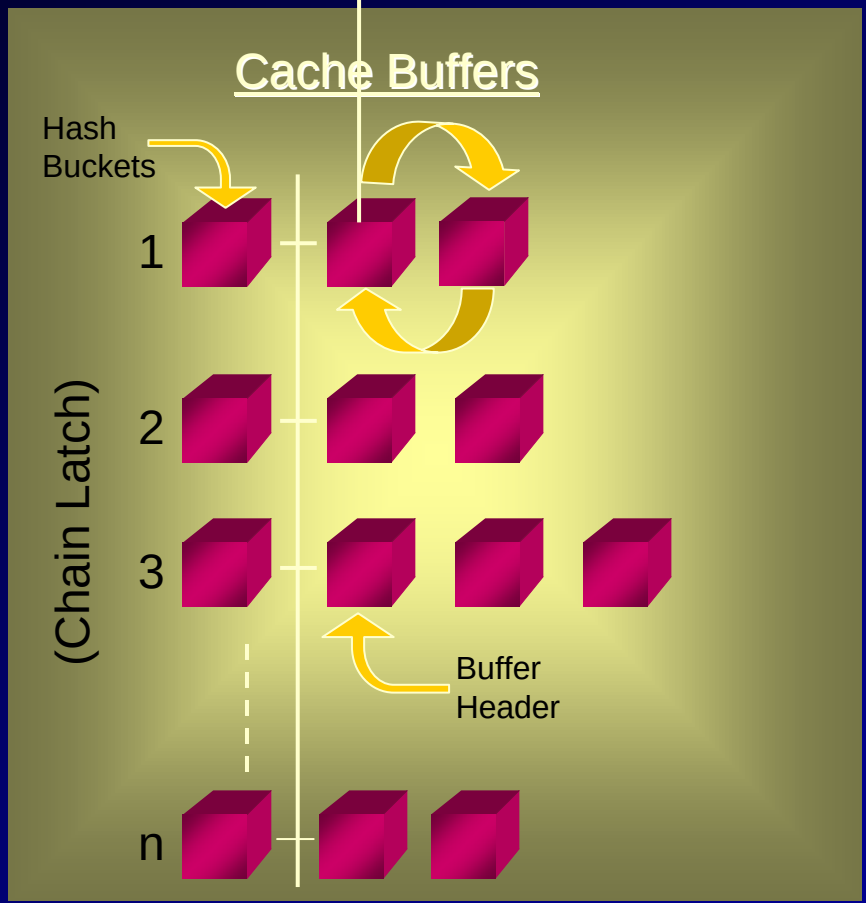
delete from emp1
where comm = 0;

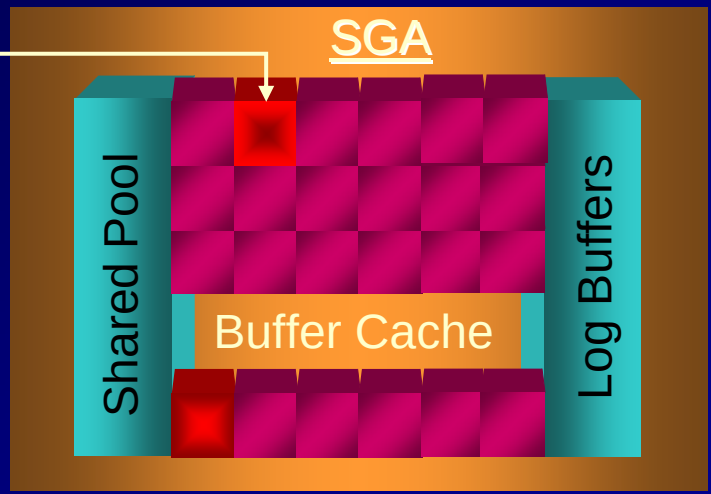
one row deleted.

Let's watch the EMP1 buffer header (Make some changes 2 copies)

```
select    lrba_seq, state, dbarfil, dbablk, tch, flag, hscn_bas,cr_scn_bas,
    decode(bitand(flag,1), 0, 'N', 'Y') dirty,          /* Dirty bit */
    decode(bitand(flag,16), 0, 'N', 'Y') temp,          /* temporary bit */
    decode(bitand(flag,1536), 0, 'N', 'Y') ping, /* ping (to shared or null) bit */
    decode(bitand(flag,16384), 0, 'N', 'Y') stale,      /* stale bit */
    decode(bitand(flag,65536), 0, 'N', 'Y') direct,     /* direct access bit */
    decode(bitand(flag,1048576), 0, 'N', 'Y') new       /* new bit */
from      x$bh
where     dbablk = 56650
order by dbablk;
```

```
  LRBA_SEQ      STATE    DBARFIL     DBABLK        TCH       FLAG   HSCN_BAS
---------- ---------- ---------- ---------- ---------- ---------- ----------
CR_SCN_BAS D T P S D N
---------- - - - - - -
         0          1          1      56650          1       8200 4294967295
         0 N N N N N N

         0          3          1      56650          2     524288          0
   4347881 N N N N N N
```





Hash Chain is now TWO! One is a CR and the other is Current.

V$Transaction now has our record (created when transactions have undo)

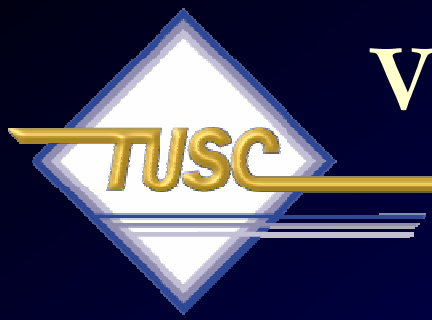

```
SELECT t.addr, t.xidusn USN, t.xidslot SLOT, t.xidsqn SQL, t.status,
       t.used_ublk UBLK, t.used_urec UREC, t.log_io LOG,
       t.phy_io PHY, t.cr_get, t.cr_change CR_CHA
FROM   v$transaction t, v$session s
WHERE t.addr = s.taddr;

ADDR             USN        SLOT         SQL STATUS                UBLK
-------- ---------- ---------- ---------- ---------------- ----------
      UREC         LOG         PHY     CR_GET     CR_CHA
---------- ---------- ---------- ---------- ----------
69E50E5C           5         42         652 ACTIVE                     1
         1          3          0          3          0
```

USN is the Undo Segment Number (rollback segment ID)

SLOT is the slot number in the rollback segment's transaction table.

SQN (Wrap) is the sequence number for the transaction.

USN+SLOT+SQN are the three values that uniquely identifies a transaction XID



V$Transaction

UBAFIL is the file for last undo entry

UBLK is block for last undo entry (find out how many undo blocks).

UBASQN is the sequence no of the last entry.

UREC is the record number of the block(shows how many table and index entries the transaction has inserted, updated or deleted.

If you are doing an INSERT or DELETE, then you will see that UREC is set to <number of indexes for this table> + how many rows you inserts/deletes. If you UPDATE a column then UREC will be set to <number of indexes that his column belongs to> * 2 + number of updated rows (so if the column belongs to no index, then UREC is set to the number of rows that was updated).

If USED_UBLK and USED_UREC are decreasing each time you query, then the transaction is rolling back. When USED_UREC zero, the rollback is finished.

Dump the block

Dump the block

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x0005.02a.0000028c	0x008000af.02b6.01	----	1	fsc 0x0029.00000000
0x02	0x0004.016.00000fae	0x008000cc.08af.34	C---	0	scn 0x0000.003deb5b

ITL – 2 Interested Transaction Lists

Committed Transaction

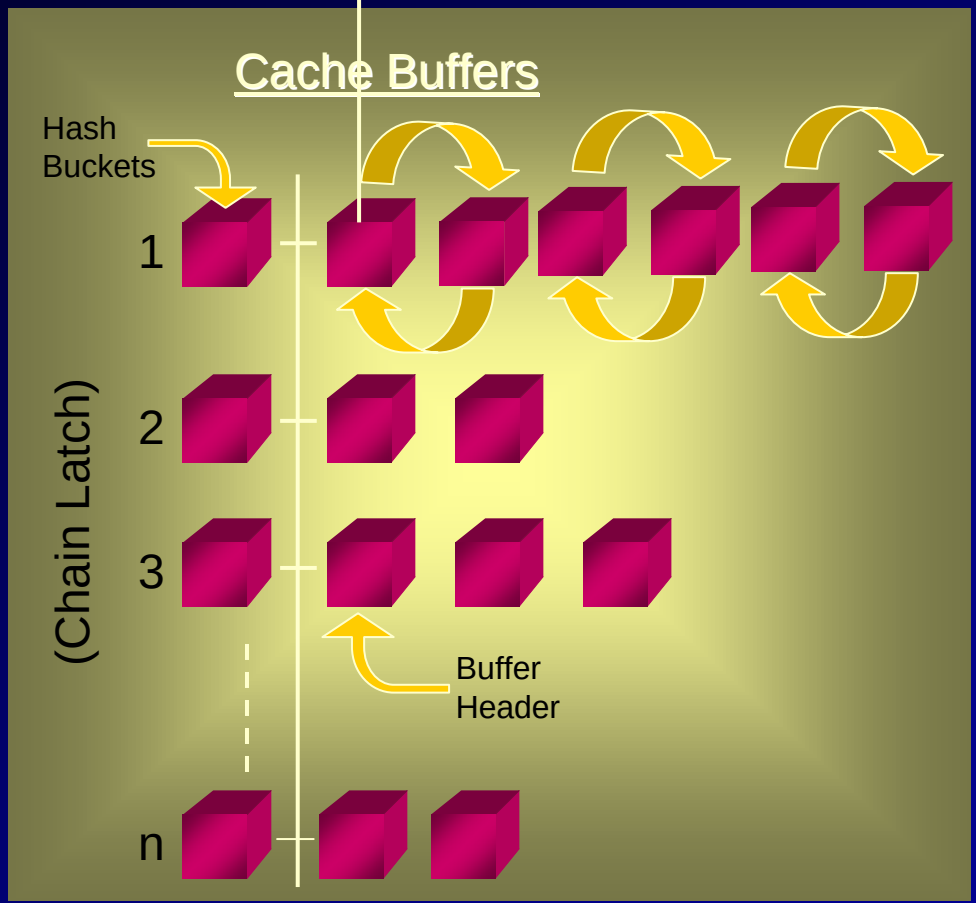
The row I deleted is still locked; fsc is 0x29 = 41 bytes

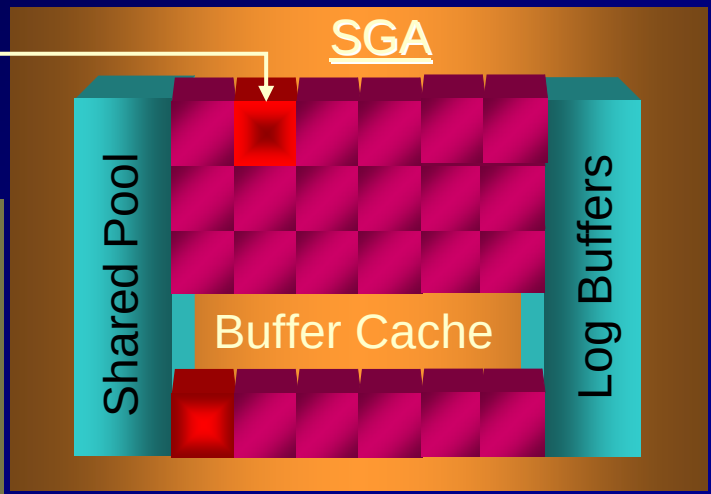
Transaction ID
Undo 5 = 5 (decimal)
Slot 2a = 42 (decimal)
SEQ 28C = 652

UBA:
File.block.sequence.record
Undo block address where last change is recorded.

Insert in 3 other sessions & drive x$bh up to the max of 6 versions of block


```
 LRBA_SEQ      STATE    DBARFIL     DBABLK        TCH        FLAG   HSCN_BAS
--------- ---------- ---------- ---------- ---------- ---------- ----------
CR_SCN_BAS D T P S D N
--------- - - - - - -
         0          3          1      56650          1     524416          0
   4350120 N N N N N N

         0          3          1      56650          1     524416          0
   4350105 N N N N N N

       365          1          1      56650          7   33562633    4350121
         0 Y N N N N N

         0          3          1      56650          1     524416          0
   4350103 N N N N N N

         0          3          1      56650          1     524416          0
   4350089 N N N N N N

         0          3          1      56650          1     524288          0
   4350087 N N N N N N
```





Hash Chain is now SIX long! Five CR and the one Current.

Why only 6 versions of a Block?

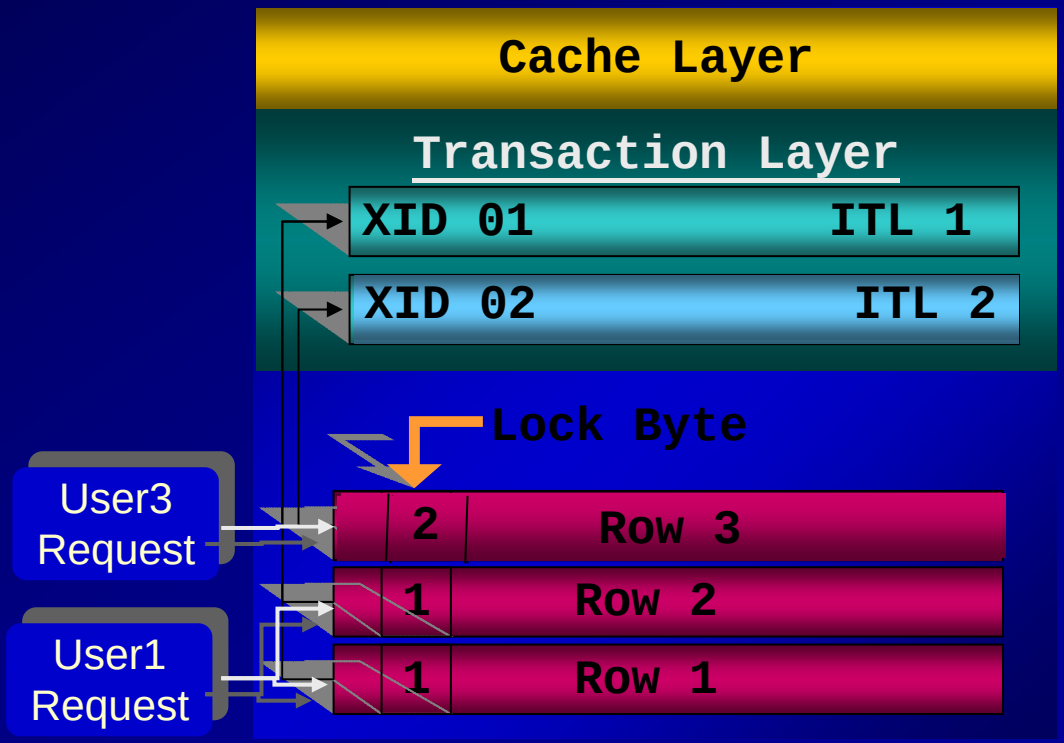
```
select      a.ksppinm, b.ksppstvl, b.ksppstdf, a.ksppdesc
from        x$ksppi a, x$ksppcv b
where       a.indx = b.indx
and         substr(ksppinm,1,1) = '_'
and         ksppinm like  '%&1%'
order by    ksppinm;

KSPPINM
--------------------------------------------------------------------------------
KSPPSTVL
--------------------------------------------------------------------------------
KSPPSTDF
---------
KSPPDESC
--------------------------------------------------------------------------------
_db_block_max_cr_dba
6
TRUE
Maximum Allowed Number of CR buffers per dba
```

What happens after we roll everything back – x$bh Still an LRBA:


```
 LRBA_SEQ      STATE    DBARFIL     DBABLK        TCH        FLAG   HSCN_BAS
--------- ---------- ---------- ---------- ---------- ---------- ----------
CR_SCN_BAS D T P S D N
---------- - - - - - -
         0          3          1      56650          1     524416          0
   4350120 N N N N N N

         0          3          1      56650          1     524416          0
   4350105 N N N N N N

       365          1          1      56650         11   35659777    4350702
         0 Y N N N N N

         0          3          1      56650          1     524416          0
   4350103 N N N N N N

         0          3          1      56650          1     524416          0
   4350089 N N N N N N

         0          3          1      56650          1     524288          0
   4350087 N N N N N N

6 rows selected.
```

User 1 – Updates Row# 1&2
User 2 updates Row 3

- *User1 updates a row with an insert/update/delete – an ITL is opened and xid tracks it in the data block.*
- *The xid ties to the UNDO header block which ties to the UNDO data block for undo.*
- *If user2 wants to query the row, they create a clone and rollback the transaction going to the undo header and undo block.*
- *If user3 wants to update same row (they wait). If user 3 wants to update different row then they open a second ITL with an xid that maps to an undo header that maps to an undo block.*



Create EMP2 ('MILLER'/'ALLEN')

```
create table emp2
as select * from emp1
where ename in ('MILLER','ALLEN');

select   empno, ename, job
from   emp2;

 EMPNO  ENAME  JOB
------------  ------------   --------------------
      7499  ALLEN   SALESMAN
      7934  MILLER  CLERK
```

Get the Blockno for EMP2

```
select rowid,empno,
       dbms_rowid.rowid_relative_fno(rowid) fileno,
       dbms_rowid.rowid_block_number(rowid) blockno,
       dbms_rowid.rowid_row_number(rowid)   rowno, rownum,
       rpad(to_char(dbms_rowid.rowid_block_number(rowid), 'FM0xxxxxxx') || '.' ||
       to_char(dbms_rowid.rowid_row_number  (rowid), 'FM0xxx'    ) || '.' ||
       to_char(dbms_rowid.rowid_relative_fno(rowid), 'FM0xxx'    ), 18) myrid
from   emp2;
```

```
ROWID                 EMPNO      FILENO     BLOCKNO       ROWNO      ROWNUM
------------------ ---------- ---------- ---------- ---------- ----------
MYRID
------------------
AAANB2AABAAAOHSAAA       7499          1      57810          0           1
0000e1d2.0000.0001

AAANB2AABAAAOHSAAB       7934          1      57810          1           2
0000e1d2.0001.0001
```

Dump the EMP2 block (Partial)

Alter system dump datafile 2 block 57810;
System Altered.

```
*** SESSION ID:(154.36) 2005-04-25 23:24:22.865
Start dump data blocks tsn: 0 file#: 1 minblk 57810 maxblk 57810
buffer tsn: 0 rdba: 0x0040e1d2 (1/57810)
scn: 0x0000.00432370 seq: 0x02 flg: 0x04 tail: 0x23700602
frmt: 0x02 chkval: 0xb205 type: 0x06=trans data
Block header dump:  0x0040e1d2
 Object id on Block? Y
 seg/obj: 0xd076  csc: 0x00.43236f  itc: 3  flg: -  typ: 1 - DATA
     fsl: 0  fnx: 0x0 ver: 0x01

 Itl           Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0xffff.000.00000000    0x00000000.0000.00  C---    0    scn 0x0000.0043236f
0x02   0x0000.000.00000000  0x00000000.0000.00  ----    0    fsc 0x0000.00000000
0x03   0x0000.000.00000000  0x00000000.0000.00  ----    0    fsc 0x0000.00000000
```

Dump the EMP2 block (Partial)

block_row_dump:
tab 0, row 0, @0x1f5d
tl: 43 fb: --H-FL-- lb: 0x0 cc: 8
col 0: [3] c2 4b 64
col 1: [5] 41 4c 4c 45 4e
col 2: [8] 53 41 4c 45 53 4d 41 4e
col 3: [3] c2 4d 63
col 4: [7] 77 b5 02 14 01 01 01
col 5: [2] c2 11
col 6: [2] c2 04
col 7: [2] c1 1f

A L L E N

Dump the EMP2 block (Partial)

tab 0, row 1, @0x1f36
tl: 39 fb: --H-FL-- lb: 0x0 cc: 8
col 0: [3] c2 50 23
col 1: [6] 4d 49 4c 4c 45 52
col 2: [5] 43 4c 45 52 4b
col 3: [3] c2 4e 53
col 4: [7] 77 b6 01 17 01 01 01
col 5: [2] c2 0e
*col 6: *NULL**
col 7: [2] c1 0b
end_of_block_dump

M I L L E R

Update 'MILLER' to 'SMALL'

```
update emp2
set    ename = 'SMALL'
where ename = 'MILLER';

select  empno, ename, job
from   emp2;

 EMPNO  ENAME  JOB
------------  ------------  --------------------
      7499  ALLEN  SALESMAN
      7934  SMALL  CLERK
```

Dump the EMP2 block (Partial)

Alter system dump datafile 2 block 57810;
System Altered.

Start dump data blocks tsn: 0 file#: 1 minblk 57810 maxblk 57810
buffer tsn: 0 rdba: 0x0040e1d2 (1/57810)
scn: 0x0000.00432794 seq: 0x05 flg: 0x00 tail: 0x27940605
frmt: 0x02 chkval: 0x0000 type: 0x06=trans data
Block header dump: 0x0040e1d2
Object id on Block? Y
seg/obj: 0xd076 csc: 0x00.43236f itc: 3 flg: O typ: 1 - DATA
fsl: 0 fnx: 0x0 ver: 0x01

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0xffff.000.00000000	0x00000000.0000.00	C---	0	scn 0x0000.0043236f
0x02	0x0004.02a.000012ff	0x00800353.0a9e.07	----	1	fsc 0x0001.00000000
0x03	0x0000.000.00000000	0x00000000.0000.00	----	0	fsc 0x0000.00000000

Here is the ITL with our Transaction

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x02	0x0004.02a.000012ff	0x00800353.0a9e.07	----	1	fsc 0x0001.00000000

Transaction ID
Undo 4 = 4 (decimal)
Slot 2a = 42 (decimal)
Seq 12ff = 4863 (decimal)

UBA:
File.block.sequence.record
Undo block address where last change is recorded.

The row that was updated is still locked; fsc is 0x1 = 1 bytes

Find the Segment & Location of RBS

```
select  segment_name
from  dba_rollback_segs
where segment_id = 4;

SEGMENT_NAME
------------------------------
_SYSSMU4$

select  header_file, header_block
from  dba_segments
where segment_name = '_SYSSMU4$';

HEADER_FILE HEADER_BLOCK
----------- ------------
          2           57
```

Dump the UNDO Header (Partial) Transaction Table (last modified blk)!

Alter system dump datafile 2 block 57;
System Altered.

```
TRN TBL::

  index  state cflags  wrap#    uel         scn            dba (uba)
   parent-xid    nub     stmt_num    cmt
  ------------------------------------------------------------------
   ------------------------
   0x00     9     0x00  0x12ff  0x0001  0x0000.00432687  0x0080034e
   0x0000.000.00000000  0x00000001   0x00000000  1114490213
   0x01     9     0x00  0x12ff  0x0002  0x0000.0043269d  0x0080034e
   0x0000.000.00000000  0x00000001   0x00000000  1114490272
...
   0x29     9     0x00  0x12ff  0x0028  0x0000.004327a4  0x00800353
   0x0000.000.00000000  0x00000001   0x00000000  1114490829
   0x2a    10     0x80  0x12ff  0x0002  0x0000.00432795  0x00800353
   0x0000.000.00000000  0x00000001   0x00000000  0
…
   End dump data blocks tsn: 1 file#: 2 minblk 57 maxblk 57
```

Dump the UNDO Header (Partial) Transaction Table!


```
index  state cflags  wrap#    uel         scn            dba
----------------------------------------------------------------
0x2a      10    0x80  0x12ff  0x0002  0x0000.00432795  0x00800353
```

State:
State 10 is Uncommitted

The scn for uncommitted (ours is) or committed transactions

Slot:
The Slot was 2a which was the 42nd in the list.

Wrap/Seq:
The Wrap is 12ff

The dba of the undo which we need to look in.

The ITL again… (fyi to see UBA)

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x02	0x0004.02a.000012ff	0x00800353.0a9e.07	----	1	fsc 0x0001.00000000

Transaction ID
Undo 4 = 4 (decimal)
Slot 2a = 42 (decimal)
Seq 12ff = 4863 (decimal)

UBA:
File.block.sequence.record
Undo block address where last change is recorded.

The row that was updated is still locked; fsc is 0x1 = 1 bytes

Find the Segment & Location of RBS

```
SELECT DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE(
          TO_NUMBER('00800353','XXXXXXXXX')) UFILE
FROM    DUAL;

UFILE
----------
         2


SELECT DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK(
          TO_NUMBER('00800353','XXXXXXXXX')) BLOCK
FROM    DUAL

BLOCK
-----------
       851

Alter system dump datafile 2 block 851;
System altered.
```

Dump the UNDO Block (Partial)


```
**********************************************************************
UNDO BLK:
xid: 0x0004.02a.000012ff  seq: 0xa9e cnt: 0x7   irb: 0x7   icl: 0x0   flg: 0x0000
 ...
Rec #0x7  slt: 0x2a  objn: 53366(0x0000d076)  objd: 53366  tblspc: …
uba: 0x00800353.0a9e.04 ctl max scn: 0x0000.00432655 prv tx scn: 0x0000.00432656
txn start scn: scn: 0x0000.00432731 logon user: 0
 prev brb: 8389454 prev bcl: 0
KDO undo record:
KTB Redo
op: 0x03  ver: 0x01
op: Z
KDO Op code: URP row dependencies Disabled
  xtype: XA flags: 0x00000000  bdba: 0x0040e1d2  hdba: 0x0040e1d1
itli: 2  ispac: 0  maxfr: 4863
tabn: 0 slot: 1(0x1) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 1
col  1: [ 6]  4d 49 4c 4c 45 52
```

Here's the UNDO: M I L L E R

Update 'ALLEN' to 'BIG'

```
update emp2
set    ename = 'BIG'
where ename = 'ALLEN';

select  empno, ename, job
from   emp2;

 EMPNO  ENAME  JOB
------------  -------------  --------------------
      7499  BIG          SALESMAN
      7934  SMALL    CLERK
```

The ITL again… (fyi to see UBA)

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0xffff.000.00000000	0x00000000.0000.00	C---	0	scn 0x0000.0043236f
0x02	0x0004.02a.000012ff	0x00800353.0a9e.08	----	2	fsc 0x0003.00000000
0x03	0x0000.000.00000000	0x00000000.0000.00	----	0	fsc 0x0000.00000000

<u>UBA:</u>
We're now at Record 8.

Two updates for the same user uses the same ITL. Now save 3 bytes if we commit.

Dump the UNDO Block (Partial)

```
*************************************************************************
UNDO BLK:
xid: 0x0004.02a.000012ff  seq: 0xa9e cnt: 0x7   irb: 0x7   icl: 0x0   flg: 0x0000
 ...
* Rec #0x7  slt: 0x2a  objn: 53366(0x0000d076)  objd: 53366  tblspc: 0(0x00000000)
*----------------------------
uba: 0x00800353.0a9e.04 ctl max scn: 0x0000.00432655 prv tx scn: 0x0000.00432656
txn start scn: scn: 0x0000.00432731 logon user: 0
KDO undo record:
KTB Redo
...
col  1: [ 6]  4d 49 4c 4c 45 52                    UNDO RECORD: M I L L E R
* Rec #0x8  slt: 0x2a  objn: 53366(0x0000d076)  objd: 53366  tblspc: 0(0x00000000)
*----------------------------
KDO undo record:
KTB Redo
op: C  uba: 0x00800353.0a9e.07
...
col  1: [ 5]  41 4c 4c 45 4e                       UNDO RECORD:  A L L E N
End dump data blocks tsn: 1 file#: 2 minblk 851 maxblk 851
```

Now insert some records as user2

insert into emp2
*select * from emp1';*

14 rows created.

Alter system dump datafile 1 block 57810;

System altered.

Here is the ITL with our Transaction

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0xffff.000.00000000	0x00000000.0000.00	C---	0	scn 0x0000.0043236f
0x02	0x0004.02a.000012ff	0x00800353.0a9e.08	----	2	fsc 0x0003.00000000
0x03	0x0005.00e.0000029b	0x008005b2.02c9.19	----	14	fsc 0x0000.00000000

Transaction ID

Now there are 2 ITL's in use.

Undo 4,5 = 4,5 (decimal)

UBA:

2 Undo Headers are used & 2 Undo blocks are referenced.

2 rows were updated on one ITL and 14 are inserted on the other.

Let's check V$TRANSACTION & match it up to ITL (no need to dump)

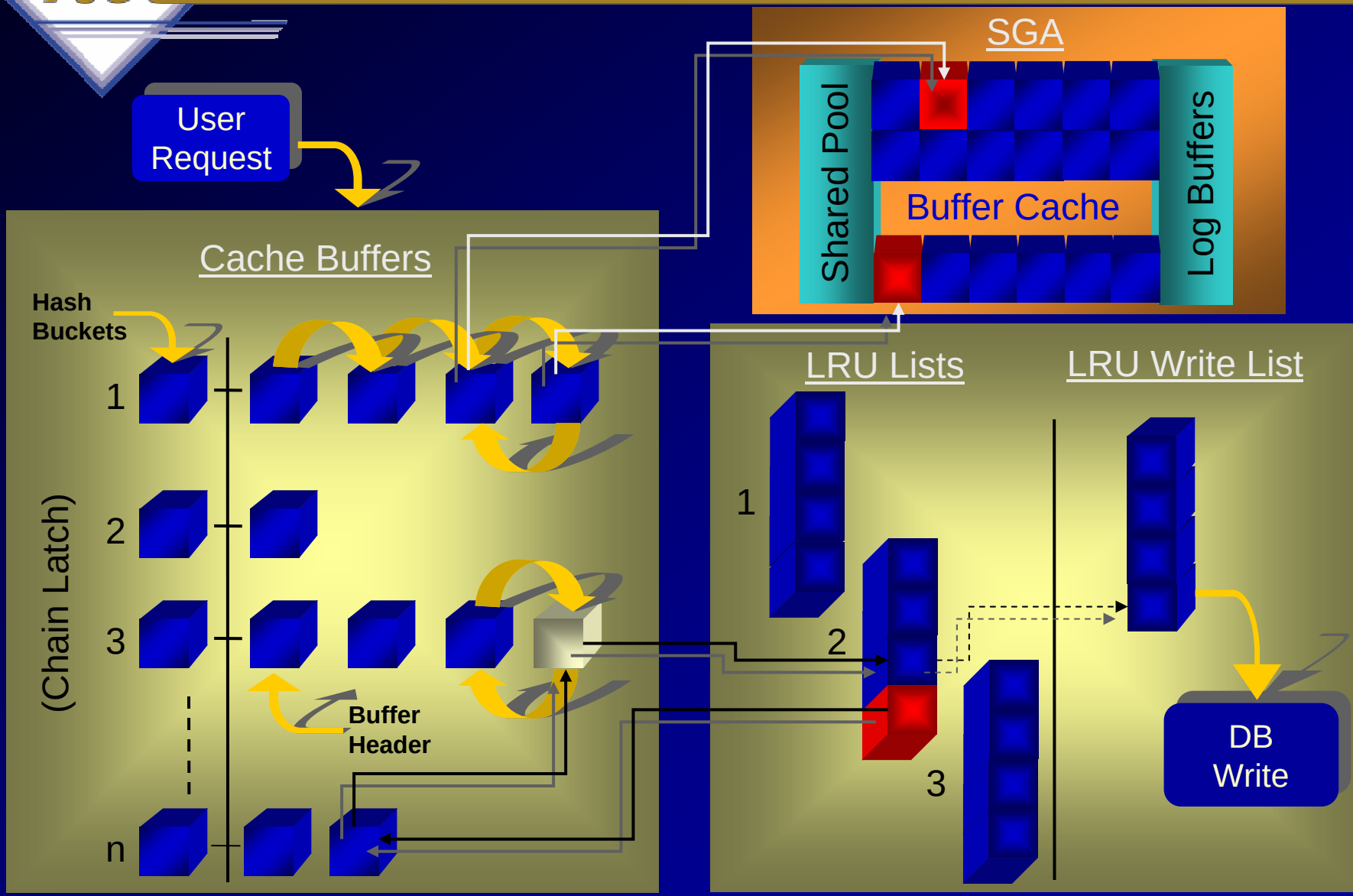
```
select   xidusn, xidslot, xidsqn, ubafil, ubablk, ubasqn, ubarec
from     v$transaction t, v$session s
where    t.ses_addr = s.saddr;
```

XIDUSN	XIDSLOT	XIDSQN	UBAFIL	UBABLK	UBASQN	UBAREC
4	42	4863	2	851	2718	8
5	14	667	2	1458	713	25

```
4.42.4863 = 4.2a.12ff      2.851.2718.8  = 800353.a9e.8
5.14.667 = 5.e.29b         2.1458.713.25 = 8005b2.2c9.19
```

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x02	0x0004.02a.000012ff	0x00800353.0a9e.08	----	2	fsc 0x0003.00000000
0x03	0x0005.00e.0000029b	0x008005b2.02c9.19	----	14	fsc 0x0000.00000000

Row Level Locks



Commit EVERYTHING!

Commit; (all sessions)

Alter system dump datafile 1 block 57810;

System altered.

Itl	*Xid*	*Uba*	*Flag*	*Lck*	*Scn/Fsc*
0x01	*0xffff.000.00000000*	*0x00000000.0000.00*	*C---*	*0*	*scn 0x0000.0043236f*
0x02	*0x0004.02a.000012ff*	*0x00800353.0a9e.08*	*----*	*2*	*fsc 0x0003.00000000*
0x03	*0x0005.00e.0000029a*	*0x008005b2.02c9.19*	*----*	*14*	*fsc 0x0000.00000000*

Why no Change (show uncommitted)?? Delayed Block Cleanout! (Usually fast commit)

Delayed block cleanout…

*Select * from emp2;*
(delayed block cleanout is how redo can be generated from a select)

Alter system dump datafile 1 block 57810;
System altered.

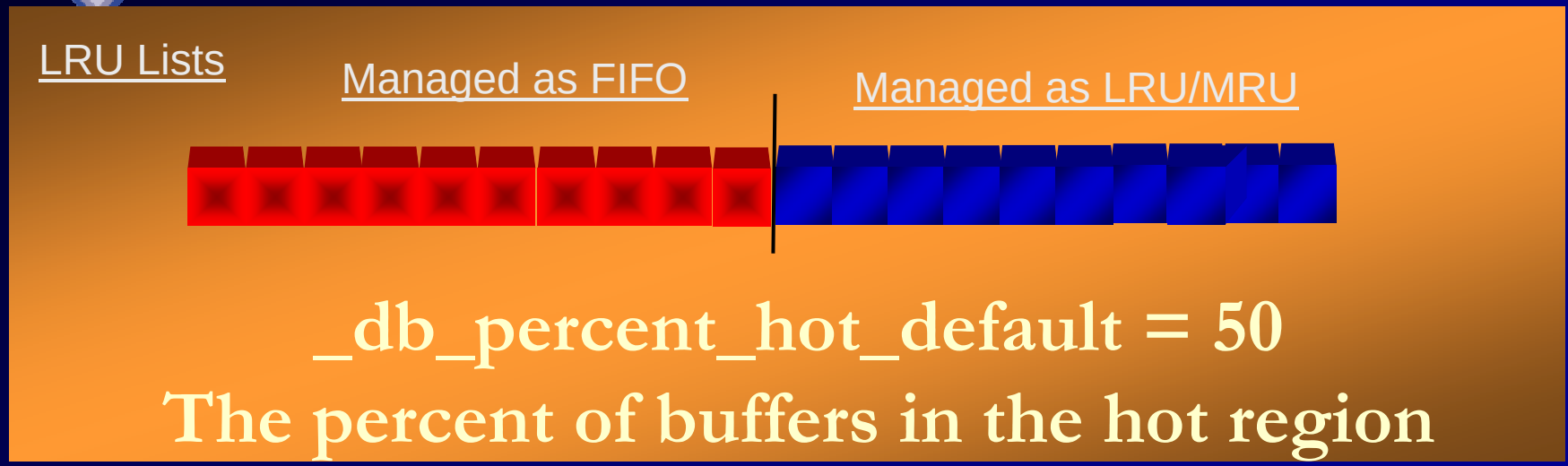
Itl	*Xid*	*Uba*	*Flag*	*Lck*	*Scn/Fsc*
0x01	*0xffff.000.00000000*	*0x00000000.0000.00*	*C---*	*0*	*scn 0x0000.0043236f*
0x02	*0x0004.02a.000012ff*	*0x00800353.0a9e.08*	*C---*	*0*	*scn 0x0000.0043469f*
0x03	*0x0005.00e.0000029a*	*0x008005b2.02c9.19*	*C---*	*0*	*scn 0x0000.004346a3*

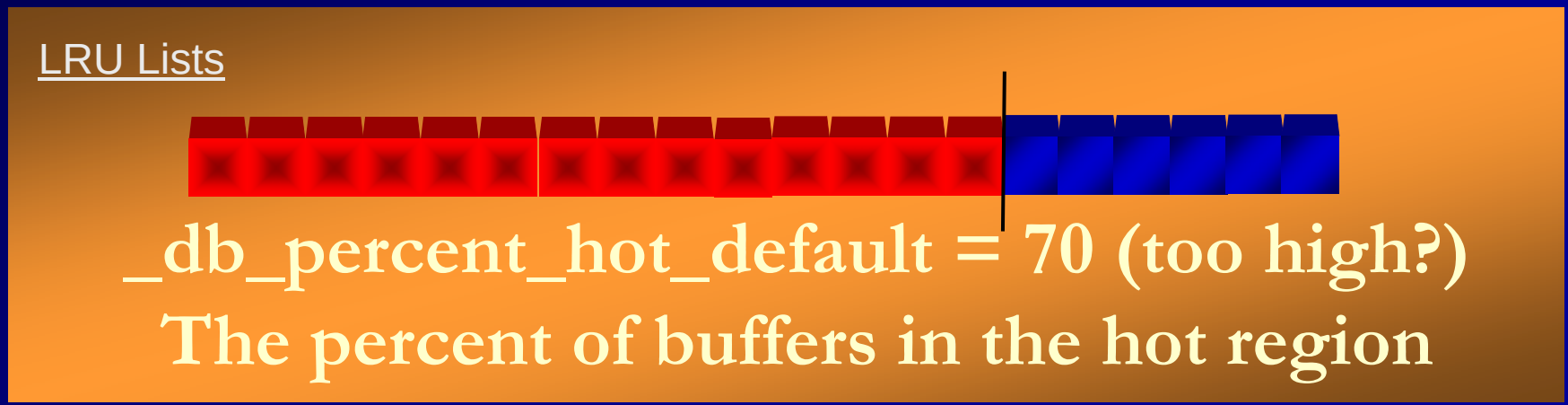
All records now show as committed.

Decoding the Hot/Cold Regions (fyi only)

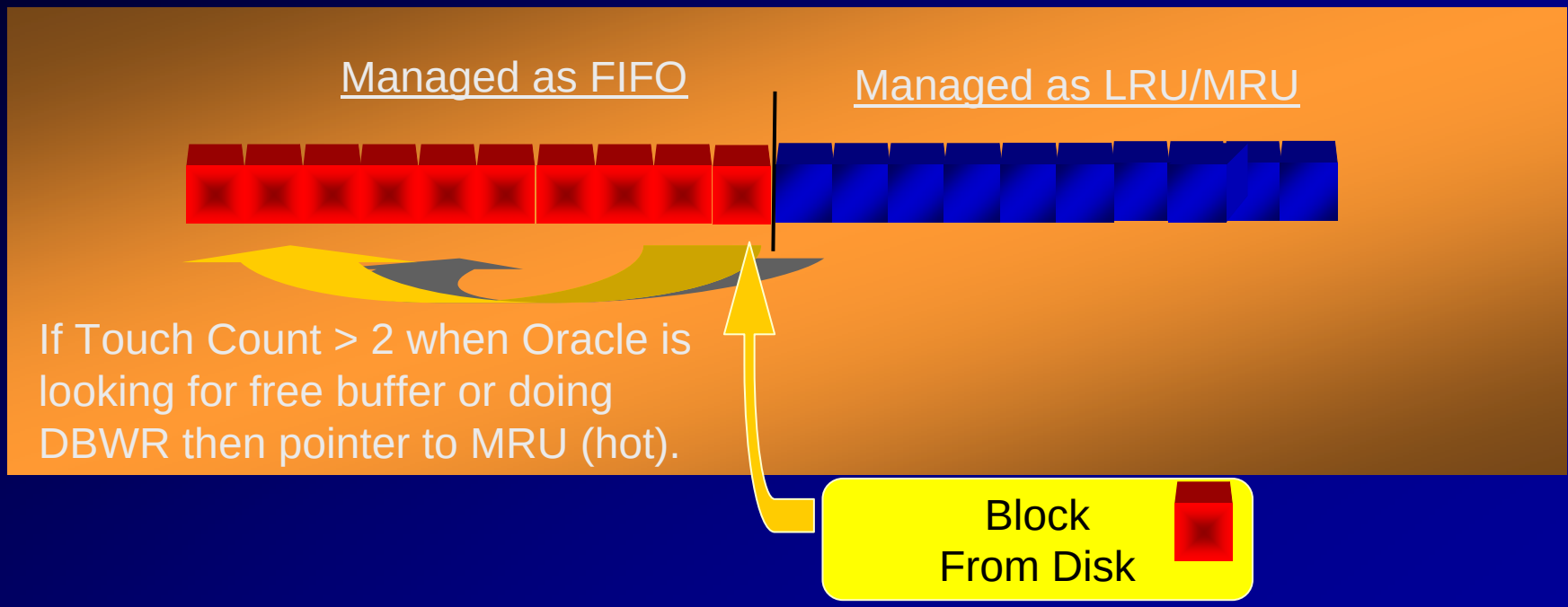
The Indus Script

The percent of buffers in the hot region
A look at pointers in the LRU





A look at pointers in the LRU



Altering the Hot/Cold LRU
Really advanced tuning!

1. _db_percent_hot_default (50) – The percent of buffers in the hot region.
2. _db_aging_touch_time* (3) – Seconds that must pass to increment touch count again. (Higher - less LRU movement)
3. _db_aging_hot_criteria (2) – Threshold to move a buffer to the MRU (hot) end of LRU chain.
4. _db_aging_stay_count (0) – **Touch count reset to this when moved to MRU (hot) end. Set=0 even if it was 200 previously!
5. _db_aging_cool_count (1) – Touch count reset to this when moved to LRU (cold) end. Set=1 even if it was 200 previously!

*Setting parameter 1 (above) lower, we increase hanging on to older buffers and setting it higher will cause a flush sooner. (*Error in description)*

Altering the Hot/Cold LRU
Really advanced tuning!

6. _db_aging_freeze_cr (FALSE) – Setting this to TRUE will make cr (consistent read) buffers too cold to keep in the cache.
7. _db_percent_hot_keep (0) – Percent of keep buffers considered hot (in hot region). Old LRU algorithm!
8. _db_percent_hot_recycle (0) – Percent of recycle buffers considered hot (in hot region). Old LRU algorithm!

** *If* _db_aging_stay_count => _db_aging_hot_criteria *then touch count is set to ½ it's current count instead of setting it to the* _db_aging_stay_count when moved to the hot end of LRU.

- FTS, FFIS (multi-block) are put on the cold end of the LRU.

Queries that may help you find solutions your problems…FYI ONLY

Great V$ - V$SEGMENT_STATISTICS


```
select object_name, statistic_name, value
from v$segment_statistics
where value > 100000
order by value;
```

OBJECT_NAME	STATISTIC_NAME	VALUE
ORDERS	space allocated	96551
ORDERS	space allocated	134181
ORDERS	logical reads	140976
ORDER_LINES	db block changes	183600

Great V$ - V$SYSTEM_EVENT

Update a row as one user and then update the same row as another user (waiting):

```
select    event, total_waits, total_timeouts
from      v$system_event
where     event like '%enq%'
```

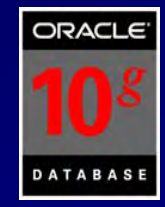
EVENT	TOTAL_WAITS	TOTAL_TIMEOUTS
enq: PR - contention	2	0
enq: RO - fast object reuse	46	0
enq: TX - row lock contention	337	336
enq: WF - contention	4	2
enq: JS - slave enq get lock1	2	1

Great V$ - V$SESSION in 10g has it all!


```
select   username, sid, event, seq#, seconds_in_wait,
         wait_time, p1 , p2, p3, st
from     V$SESSION
where    NOT(event like 'SQL%')
and      NOT(event like '%message%')
and      NOT(event like '%otimer%')
and      NOT(event like '%pipe get%')
order by wait_time desc, event;
```

V$SESSION EVEN HAS: row_wait_obj#, row_wait_file#, row_wait_block#, row_wait_row#

Great V$ - V$SESSION has it all!




```
USERNAME                         SID EVENT
------------------------------ ---------- ----------------------------
      SEQ# SECONDS_IN_WAIT  WAIT_TIME         P1         P2         P3
---------- -------------- ---------- ---------- ---------- ----------
STATE
----------------
                                   159 Queue Monitor Wait
        23         787822          0          0          0          0
WAITING
SYS                                149 enq: TX - row lock contention
        19            901          0 1415053318     327699        652
WAITING

                                   162 jobq slave wait
         1              1          0          0          0          0
WAITING

                                   147 wakeup time manager
         1         787822          0          0          0          0
WAITING
```

x$ksqst – Kernel service enqueue statistics by type (no waits yet)


```
select ksqsttyp type, ksqstsgt gets, ksqstwat waits
from x$ksqst
where ksqstsgt > 0
order by waits , gets
```

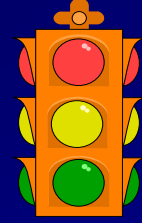
TYPE	GETS	WAITS
CU	6313	0
HW	9091	0
TX	55309	0
TM	101973	0
CF	132720	0

x$bh – Cache buffers chains issues

- *To identify a heavily accessed buffer chain look at the latch statistics for cache buffers chains latch under V$LATCH_CHILDREN and match this to X$BH:*

```
select   dbarfil, dbablk, class, state
from     x$bh
where    hladdr = '<latch address from v$latch_children>';
```

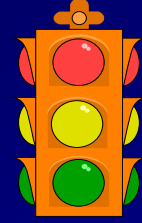
X$ - Finding Specific waits



Buffer Busy Waits or Write Complete Waits Events:

```
SELECT    /*+ ordered */ sid, event, owner, segment_name,
          segment_type,p1,p2,p3
FROM      v$session_wait sw, dba_extents de
WHERE     de.file_id = sw.p1
AND       sw.p2 between de.block_id
          and de.block_id+de.blocks – 1
AND       (event = 'buffer busy waits'
          OR event = 'write complete waits')
AND       p1 IS NOT null
ORDER BY event,sid;
```

X$ - Finding Specific waits



```
col name for a20
col p1 for a10
col p2 for a10
col p3 for a10
select  event#,name,parameter1 p1,parameter2 p2,parameter3 p3
from   v$event_name
where name in ('buffer busy waits', 'write complete waits')
 EVENT# NAME                 P1         P2         P3
------- -------------------- ---------- ---------- ----------
    143 write complete waits file#      block#
    145 buffer busy waits    file#      block#     id
```

Combining Trace and Wait Events!

- Querying the V$SESSION_WAIT table gives a point in time view of the waits
- To record all the waits use
  - **Alter session set events '10046 trace name context forever, level *12*';**
- In the trace file you'll see entries such:
  - **WAIT #2: nam='db file sequential read' ela= 0 p1=1 p2=1135 p3=1**
    - **#2 - is the SQL that caused the wait**
    - **P1 - file#, P2 - block#, p3 - number of block read**

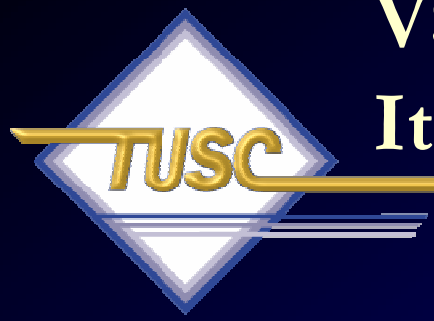
Lots of latch free waits in this one…

```
WAIT #2: nam='latch free' ela= 0 p1=-
  2147423252 p2=105 p3=0
WAIT #2: nam='latch free' ela= 0 p1=-
  2147423252 p2=105 p3=1
WAIT #2: nam='latch free' ela= 0 p1=-
  1088472332 p2=106 p3=0
WAIT #2: nam='latch free' ela= 0 p1=-
  2147423252 p2=105 p3=0
WAIT #2: nam='latch free' ela= 0 p1=-
  2147423252 p2=105 p3=1
```

Checking this event …

```
select  event#, name, parameter1 p1, parameter2 p2,
        parameter3 p3
from    v$event_name
where name in ('latch free');

EVENT# NAME                 P1         P2         P3
------ -------------------- ---------- ---------- ----------
     3 latch free           address    number     tries
```

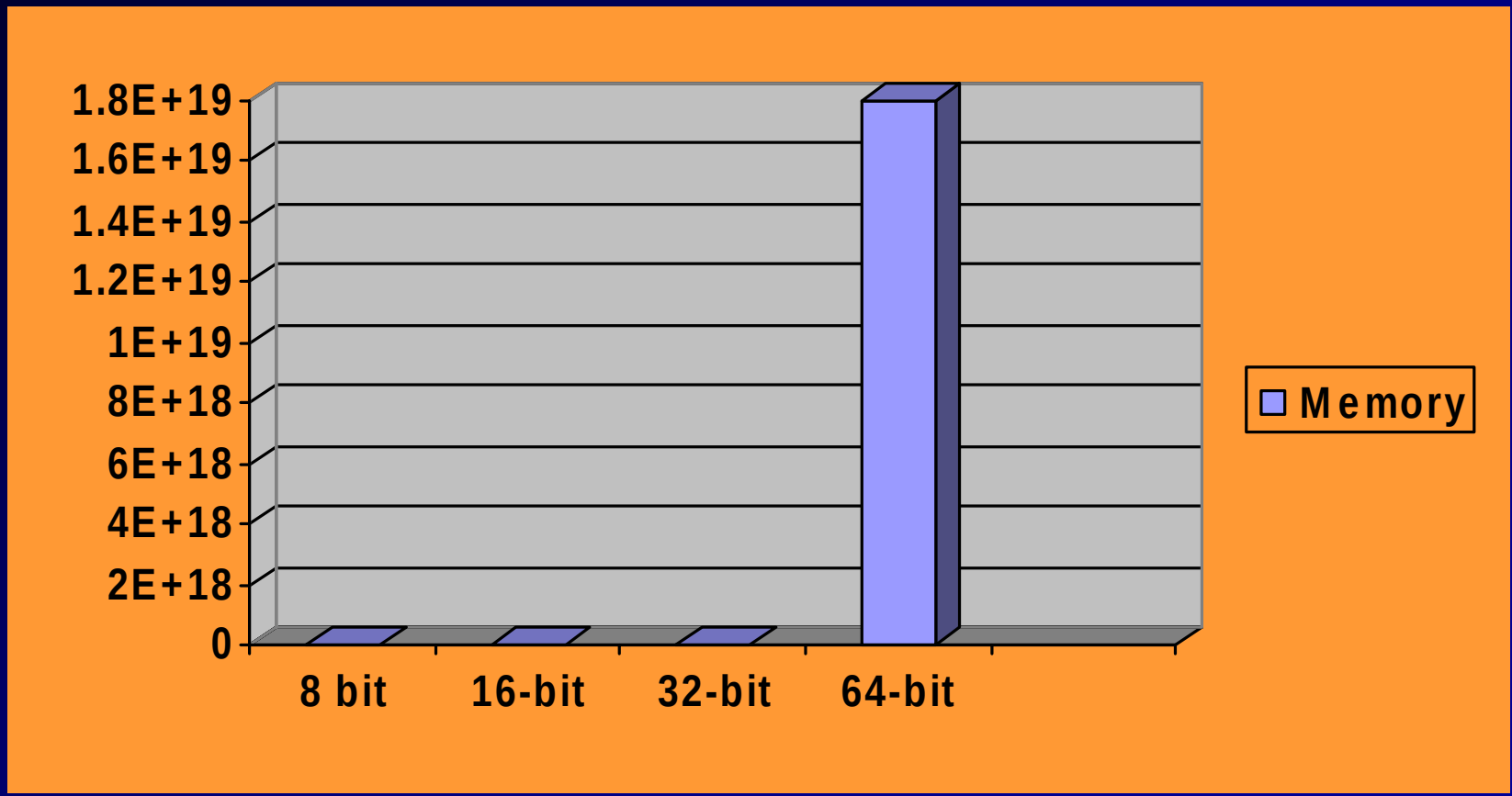
V$ - Top 10 as % of All
It's always the SQL…




```
select sum(pct_bufgets) percent
from (select rank() over ( order by buffer_gets desc )     as rank_bufgets,
   to_char(100 * ratio_to_report(buffer_gets) over         (), '999.99')
    pct_bufgets
from   v$sqlarea )
where  rank_bufgets < 11;

PERCENT
-----------
     97.07
```

If Time Permits… the Future!

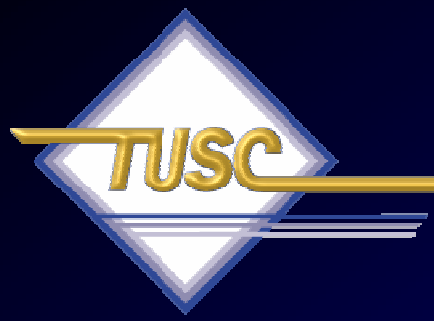
Directly Addressable Memory



64-Bit advancement of Directly addressable memory

	Address Direct	Indirect/Extended
• 4 Bit:	16	(640)
• 8 Bit:	256	(65,536)
• 16 Bit:	65,536	(1,048,576)
• 32 Bit:	4,294,967,296	
• 64 Bit:	18,446,744,073,709,551,616	

- When the hardware physically implements the theoretical possibilities of 64-Bit, things will dramatically change….
  …moving from 32 bit to 64 bit will be like moving from 4 bit to 32 bit or like moving from 1971 to 2000 overnight.

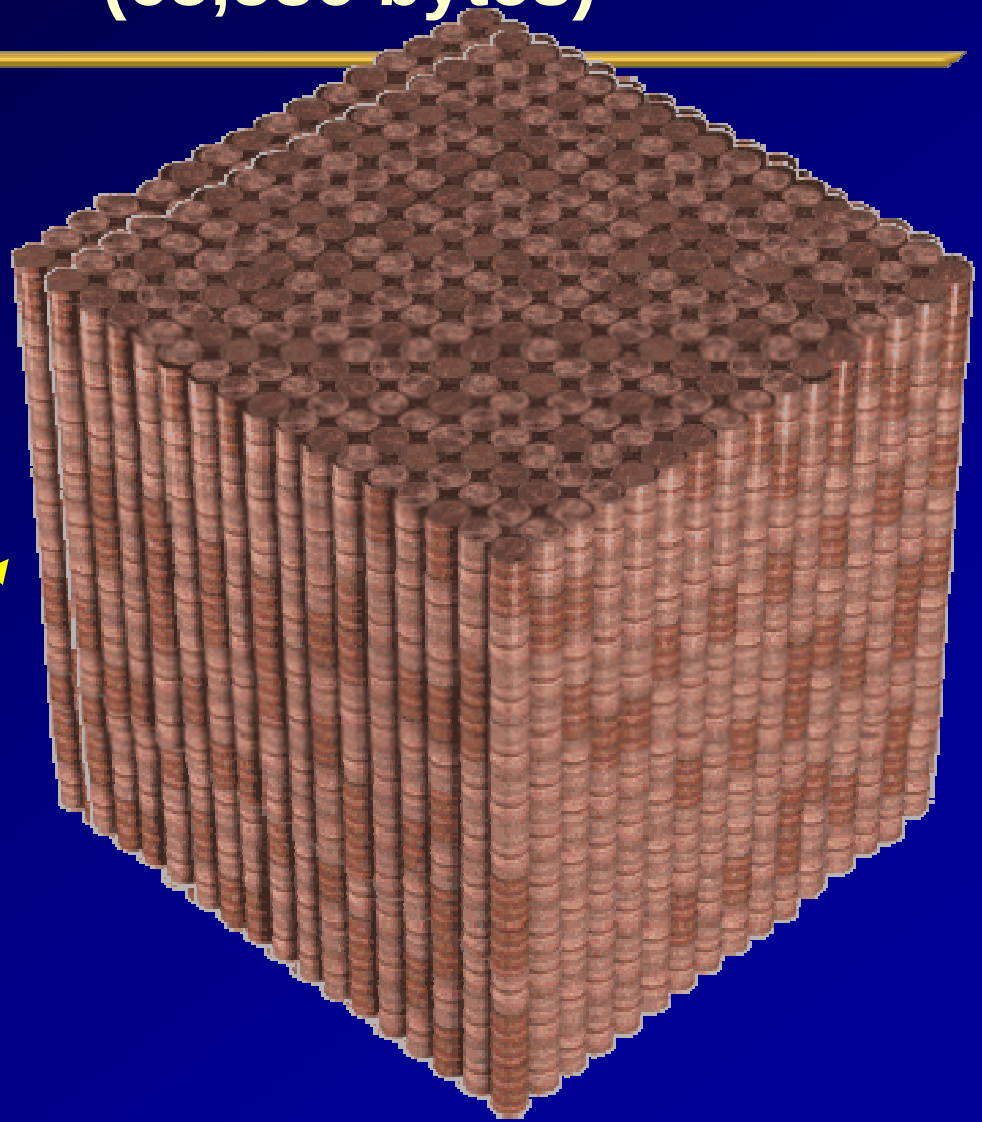


16 – Bit
(65,536 bytes)

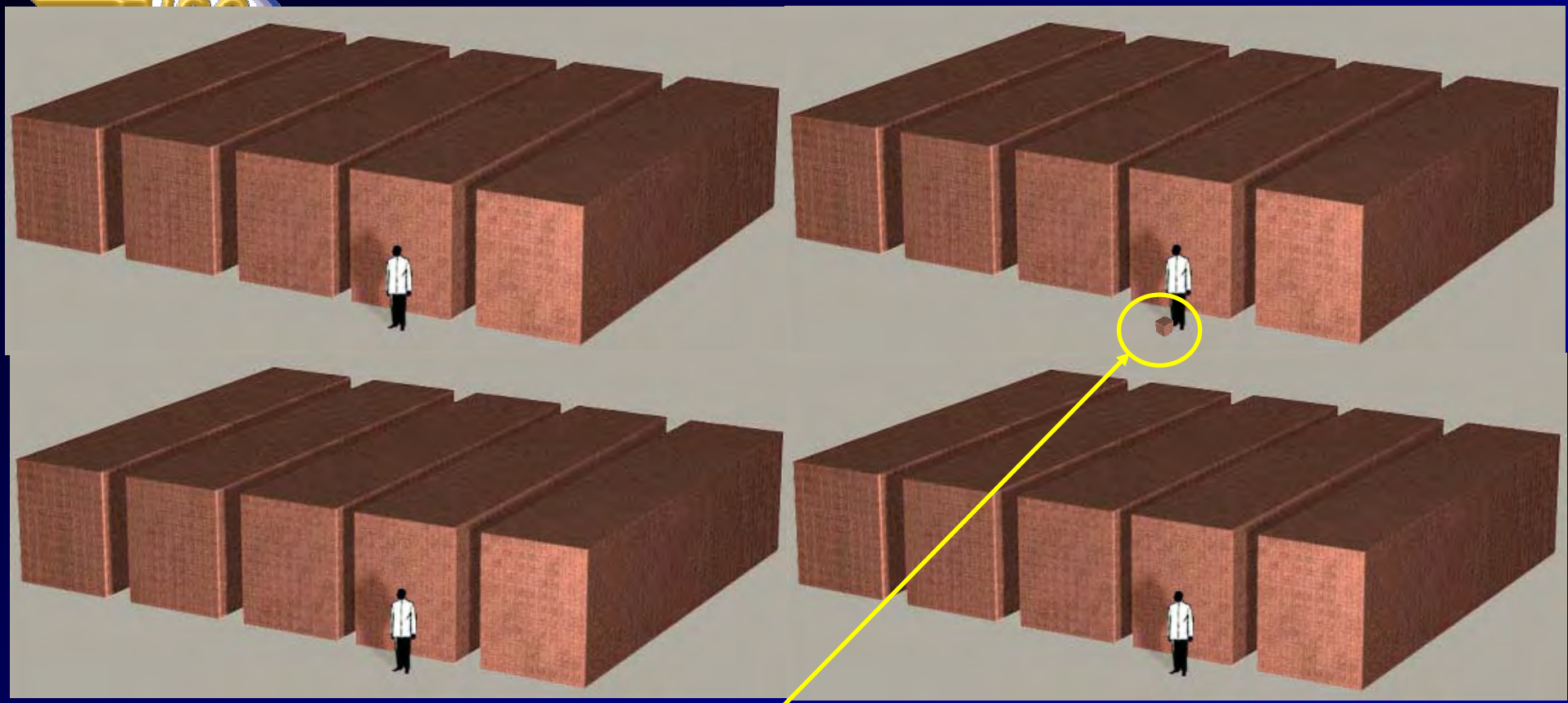
What if directly addressable memory was pennies?

4 – Bit
(16 bytes)

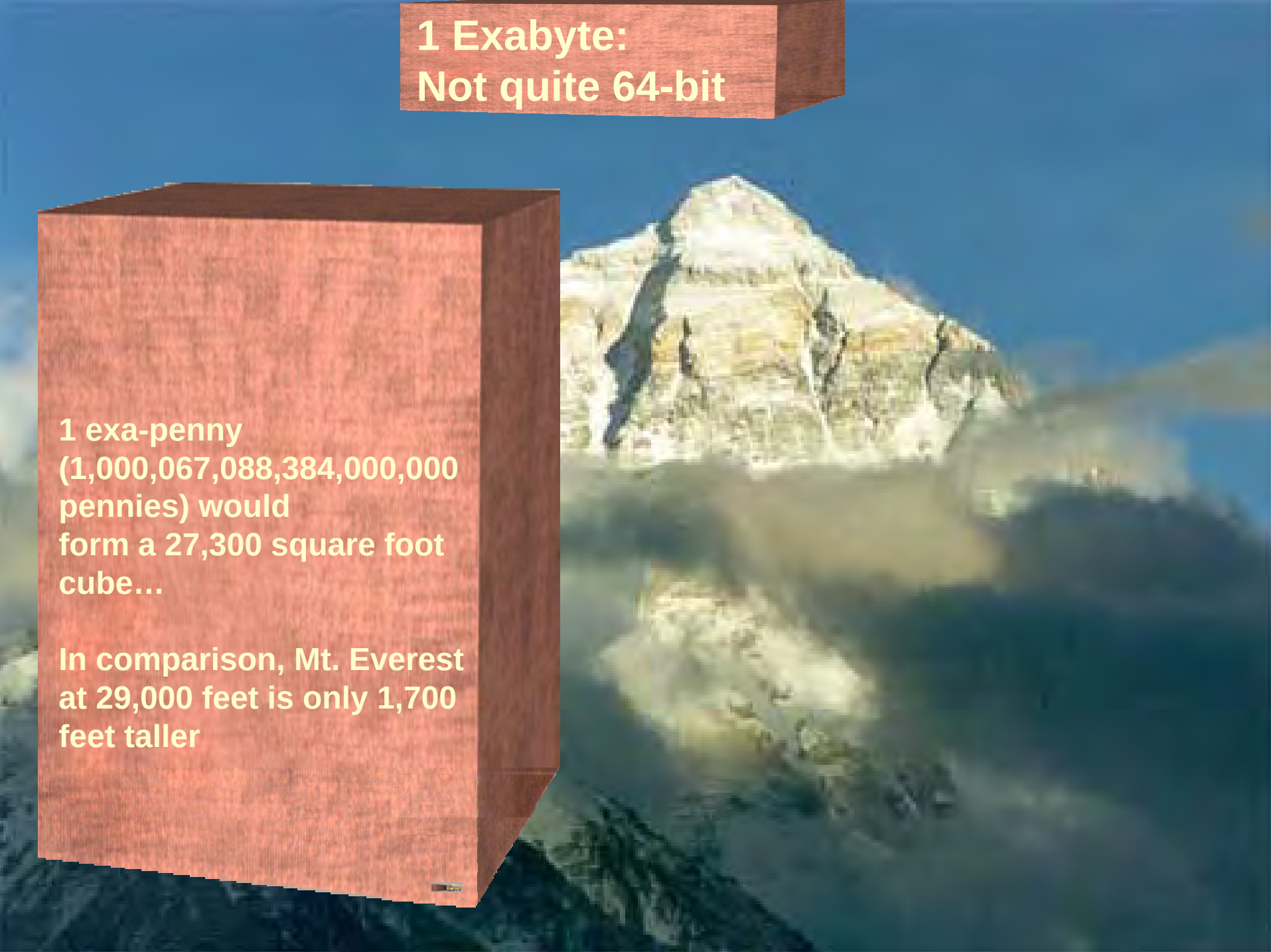
8 – Bit
(256 bytes)



32 – Bit
(4,294,967,296 bytes)



Here is the square from the last slide for 16-bit (a bit smaller)


1 Exabyte:
Not quite 64-bit
1 exa-penny (1,000,067,088,384,000,000 pennies) would form a 27,300 square foot cube…
In comparison, Mt. Everest at 29,000 feet is only 1,700 feet taller

64 – Bit
(18,446,744,073,709,551,616 **bytes)**

16 exa-pennies would easily dwarf even Mt. Everest!

8 Exabytes: Look what fits in one 10g Database!

- 1000 Internets (8P each)

or

- 400,000 Libraries of Congress (20T each and 17-18 million books in each)

or

- 8 Billion Movies on CD (1 G each)

or

- 8 Billion Pickup Trucks of Documents (1G each)

or

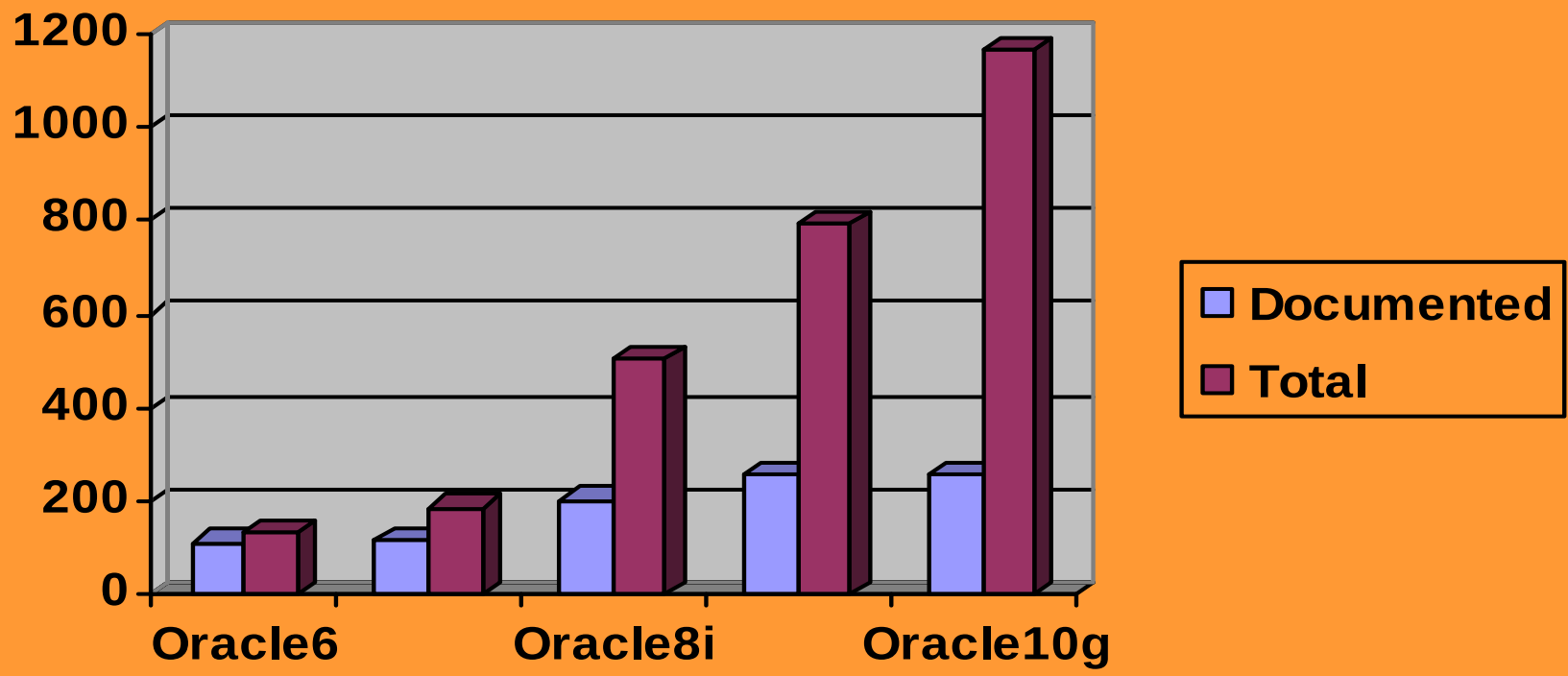
- 1 Mount Everest filled with Documents (approx.)

or

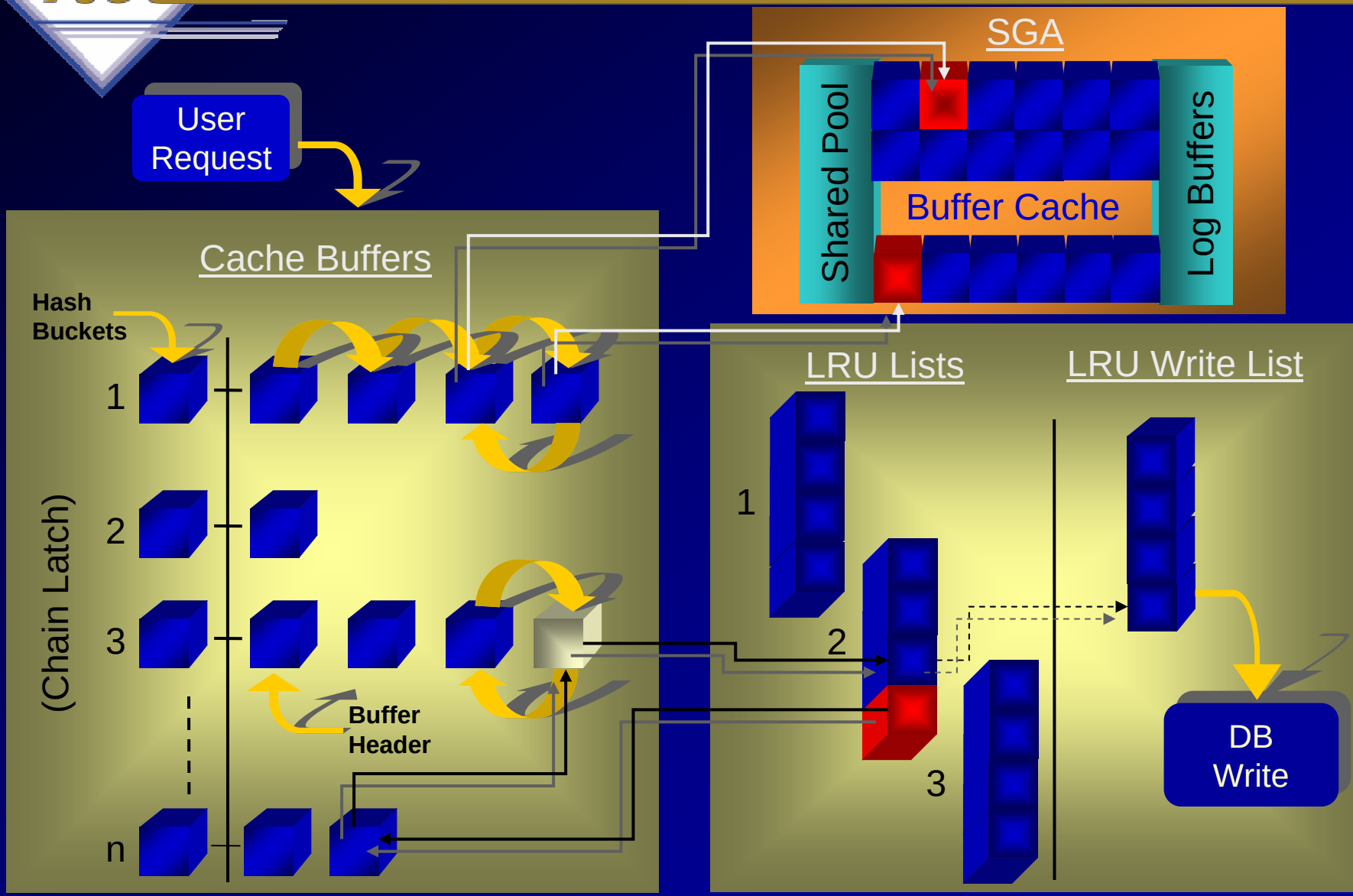
- Every piece of data produced each year (2E-5.6E)

Init.ora Parameters

Documented & Total



Row Level Locks



The Init.ora over the years

Documented & Undocumented

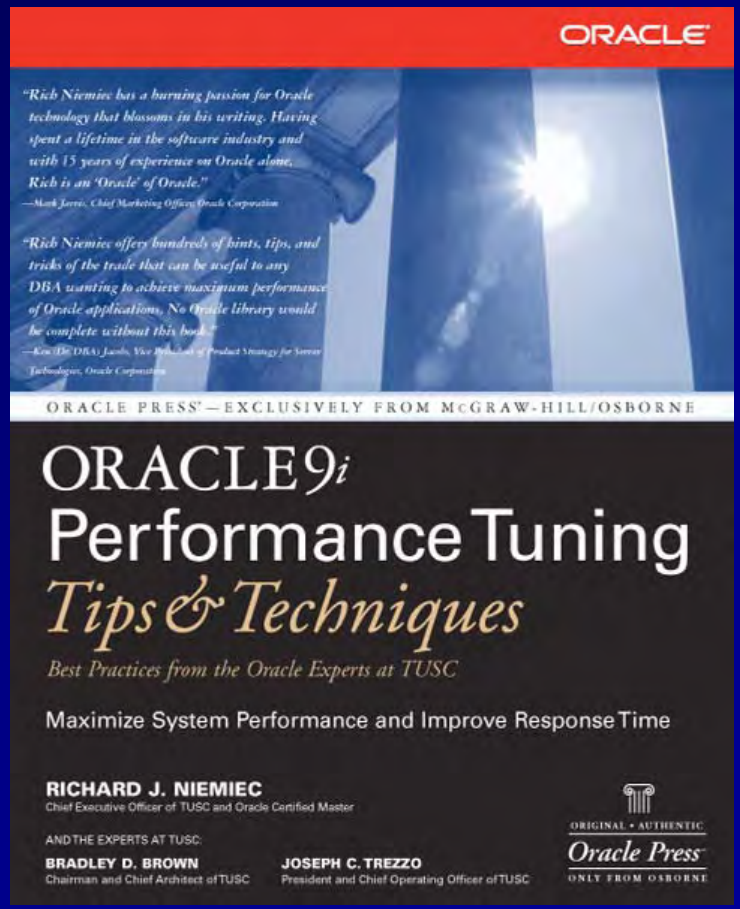
Version	Doc.	Undoc.	Total
6	111	19	130
7	117	68	185
8.0	193	119	312
8.1	203	301	504
9.0	251	436	687
9.2	257	540	797
10.1.0.2	255 (-1%)	911 (+69%)	1166 (+42%)

We Covered:

- Terminology
- What you're Waiting on
- Block Dumps & Block Information
- Transactions moving through Oracle
  - Buffer Cache
  - ITL & UNDO tie
  - Delayed block cleanout

For More Information

- www.tusc.com
- *Oracle9i Performance Tuning Tips & Techniques; Richard J. Niemiec; Oracle Press (May 2003)*



"If you are going through hell, keep going" - Churchill

"We make a Living by what we get; We make a Life by what we give."

References

- *Oracle9i Performance Tuning Tips & Techniques,* Rich Niemiec
- Block Level Reading Tool is from Terlingua Software
- The Machinations of Oracle, Scott Marin, Terlingua software
- *All about Oracle's touch count algorithm, Craig Schallahamer*
- *Mark Bobak, Kevin Gilpin, Jonathon Lewis, Metalink Notes*
- *EM Grid Control 10g;* otn.oracle.com, Oracle Corporation
- *Oracle Enterprise Manager 10g: Making the Grid a Reality*; Jay Rossiter, Oracle Corporation
- http://www.kokogiak.com/megapenny
- Oracle 10g documentation
- www.tusc.com, www.oracle.com, www.ixora.com, www.laoug.org, www.ioug.org, technet.oracle.com
- *Special thanks to Steve Adams, Brad Brown, Kevin Gilpin, and Joe Trezzo.*
- *Oracle OLN Advanced Tuning Class, Oracle Corporation*
- www.lookuptables.com
- Scaling Oracle8i by James Morle
- Reading a block dump, TUSC
- http://www.ixora.com.au
- http://www.jlcomp.demon.co.uk/

TUSC Services

- Oracle Technical Solutions
  - Full-Life Cycle Development Projects
  - Enterprise Architecture
  - Database Services
- Oracle Application Solutions
  - Oracle Applications Implementations/Upgrades
  - Oracle Applications Tuning
- Managed Services
  - 24x7x365 Remote Monitoring & Management
  - Functional & Technical Support
- Training & Mentoring
- Oracle Authorized Reseller

Copyright Information

- Neither TUSC, Oracle, NYOUG, nor the author guarantee this document to be error-free. Please provide comments/questions to rich@tusc.com.
- TUSC © 2005. This document cannot be reproduced or used without expressed written consent from an officer of TUSC, but NYOUG may reproduce or copy this for conference use.

Contact Information

Rich Niemiec: rich@tusc.com

www.tusc.com