Three Types of Table Compression, Part 2

A tale about a room with two doors in plain view, and a hidden forgotten third door…

hroug

hrvatska udruga oracle korisnika

Tim Gorman

Delphix

Friday, 17-Oct 2014

Yesterday's agenda

- **The story behind the story**
- **Overview of data compression**
- **Overview of table data compression in Oracle database**
  - **Review of related concepts within Oracle database**
    - **Internal block and row formats**
    - **Cluster tables, row-chaining, and direct-path loads**
- **Details of BASIC/OLTP and HCC table compression**
  - **De-duplication compression (basic and OLTP)**
  - **Hybrid Columnar Compression (HCC)**
- **Trailing NULL columns**
  - **The rest of the story**

Today's agenda

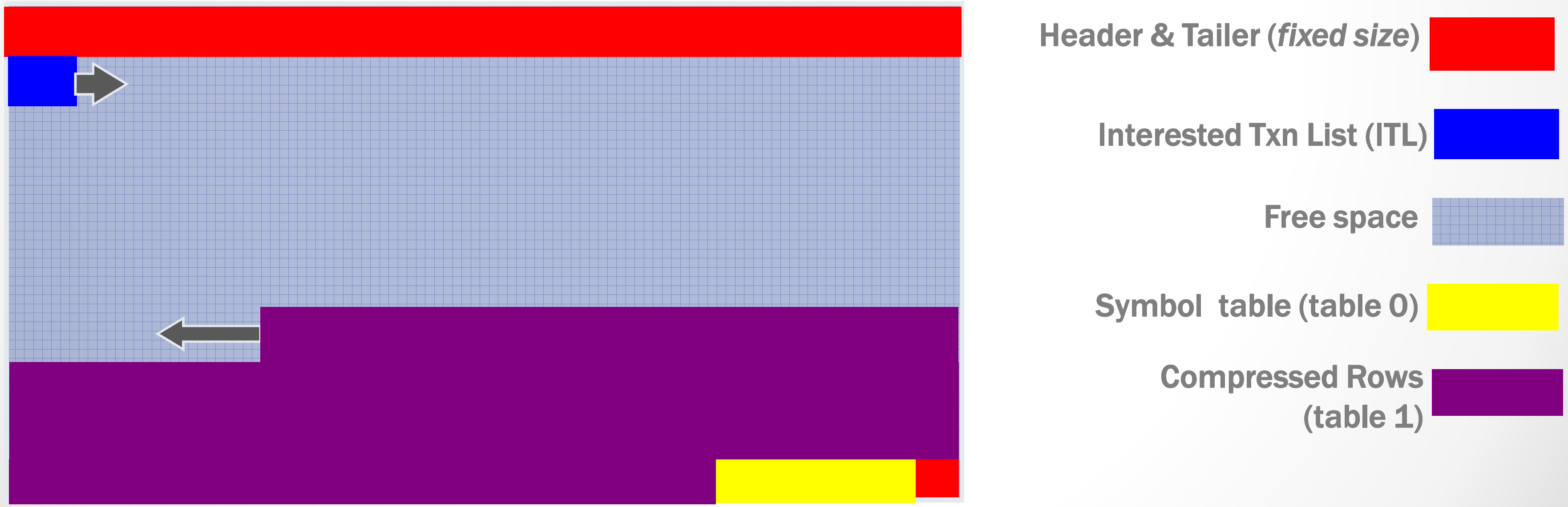
- The story behind the story
- Overview of data compression
- Overview of table data compression in Oracle database
  - Review of related concepts within Oracle database
    - Internal block and row formats
    - Cluster tables, row-chaining, and direct-path loads
- Details of BASIC/OLTP and HCC table compression
  - De-duplication compression (basic and OLTP)
  - Hybrid Columnar Compression (HCC)
- Trailing NULL columns
  - The rest of the story

COMPRESS BASIC

- Symbol table is implemented as a 2nd table in the block
  - Just like a clustered tables
- Each entry in symbol table contains repetitive data values
  - One or more columns per entry
    - If two or more rows contain the same data values in the one or more contiguous columns, then this will be represented and replaced by an entry in the symbol table

COMPRESS BASIC

- Database block layout illustration
  - Distinct data values stored once in symbol table
  - Basic compression only occurs on direct-path INSERT
    - Conventional INSERT, UPDATE leave NOCOMPRESS rows



Header & Tailer (*fixed size*) 🟥

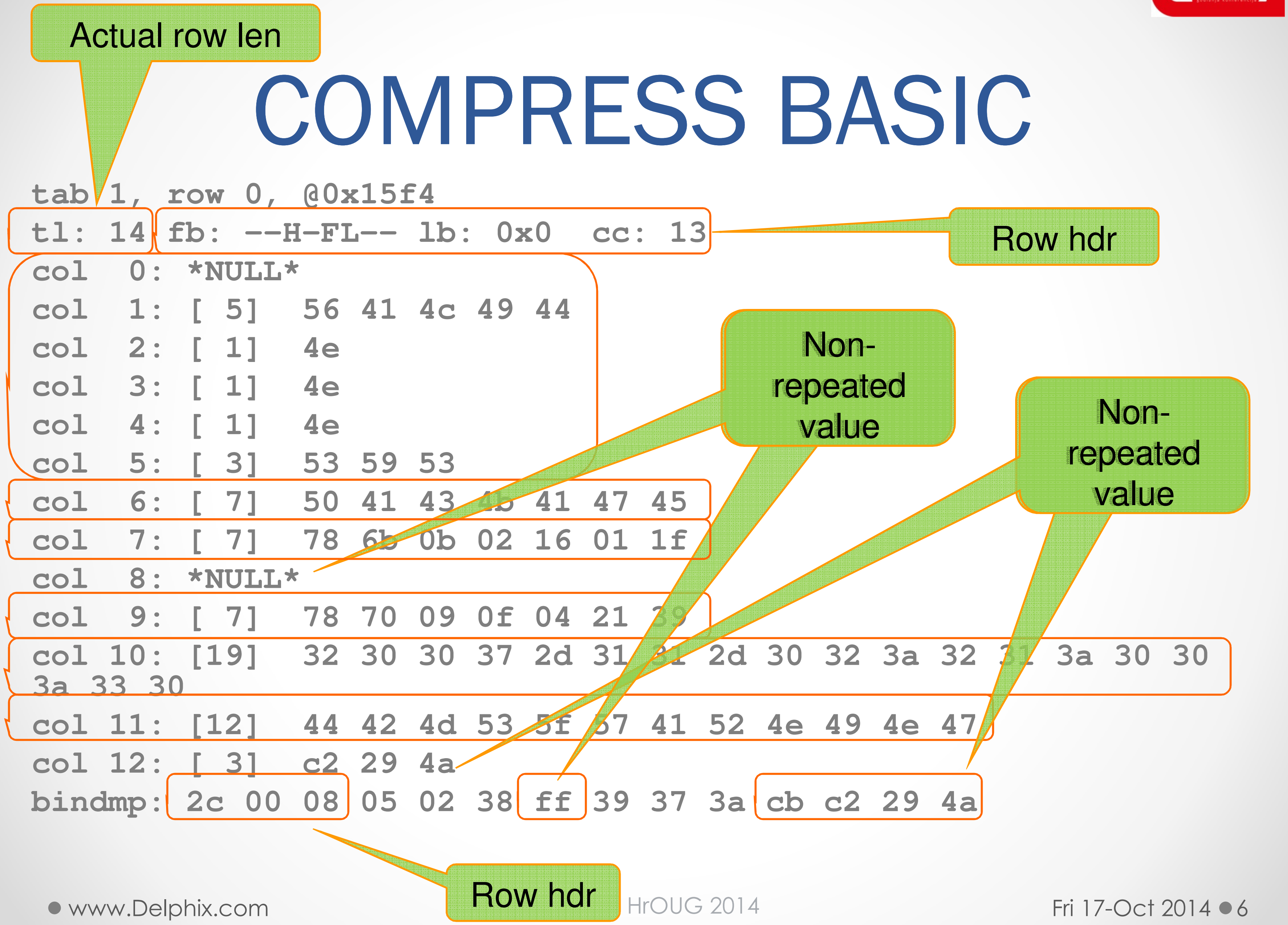
Interested Txn List (ITL) 🟦

Free space ▨

Symbol table (table 0) 🟨

Compressed Rows (table 1) 🟪

COMPRESS BASIC



```
tab 1, row 0, @0x15f4
tl: 14 fb: --H-FL-- lb: 0x0  cc: 13
col  0: *NULL*
col  1: [ 5]  56 41 4c 49 44
col  2: [ 1]  4e
col  3: [ 1]  4e
col  4: [ 1]  4e
col  5: [ 3]  53 59 53
col  6: [ 7]  50 41 43 4b 41 47 45
col  7: [ 7]  78 6b 0b 02 16 01 1f
col  8: *NULL*
col  9: [ 7]  78 70 09 0f 04 21 39
col 10: [19]  32 30 30 37 2d 31 31 2d 30 32 3a 32 31 3a 30 30
3a 33 30
col 11: [12]  44 42 4d 53 5f 57 41 52 4e 49 4e 47
col 12: [ 3]  c2 29 4a
bindmp: 2c 00 08 05 02 38 ff 39 37 3a cb c2 29 4a
```

Actual row len

Row hdr

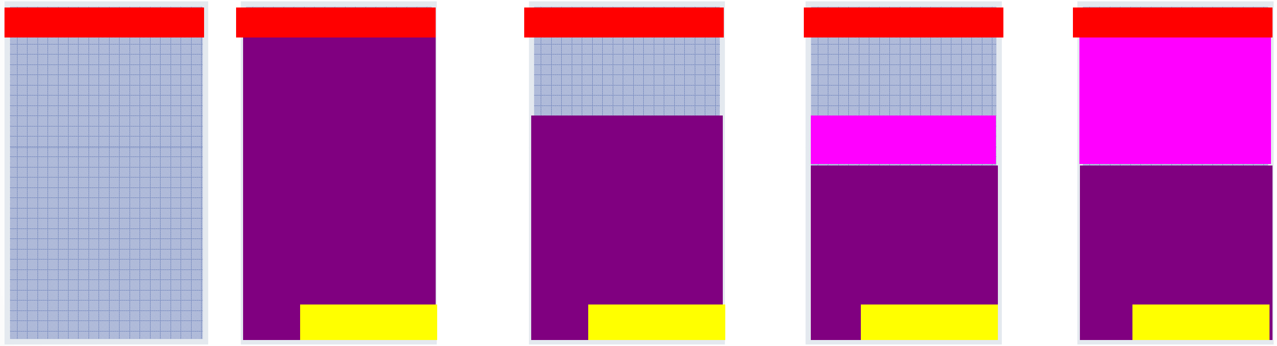
Non-repeated value

Non-repeated value

Row hdr

BASIC lifecycle

- Data lifecycle with basic compression
  - Normal DML operations as well as direct-path supported



Time A

Empty

Time B

Compressed
on direct-path
APPEND INSERT

Time C

Rows
deleted

Time D

Rows
updated

Time E

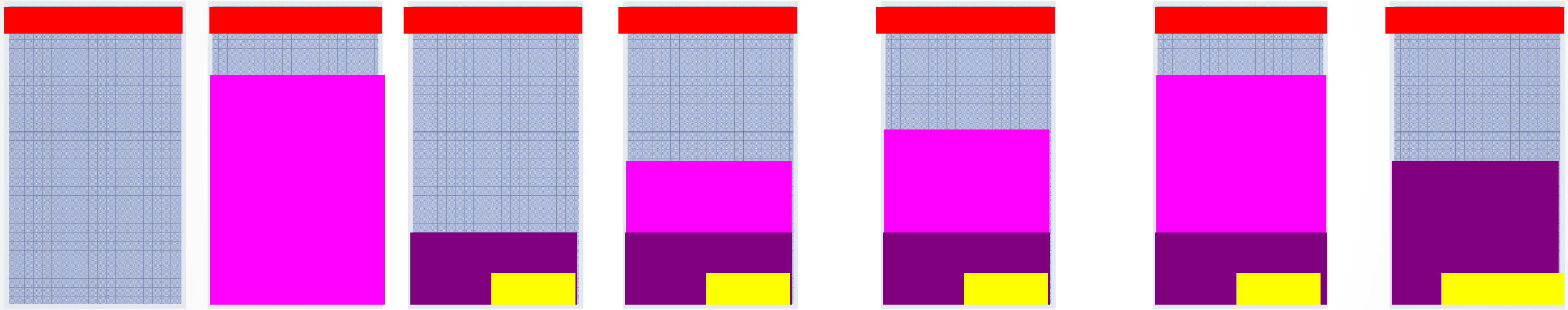
Rows
inserted

COMPRESS FOR OLTP

- Same compression method as BASIC
- The difference is how it is managed
  - Rows are initially inserted as NOCOMPRESS
  - When block would be marked FULL
    - session instead compresses the contents of the block
  - Compression operation remains a *direct-path* operation
- Supports both *conventional-path* as well as *direct-path* INSERT and MERGE operations
  - All UPDATE and DELETE operations are conventional-path, also supported here

FOR OLTP lifecycle

- Data lifecycle with advanced compression
  - Normal DML operations as well as direct-path supported



Time A

Empty

Time B

Newly inserted uncompressed data

Time C

Compressed

Time D

Compressed with newly uncompressed data inserted or updated

Time E

Compressed with newly uncompressed data inserted or updated

Time F

Compressed with newly uncompressed data inserted or updated

Time G

Re-compressed

BASIC vs FOR OLTP

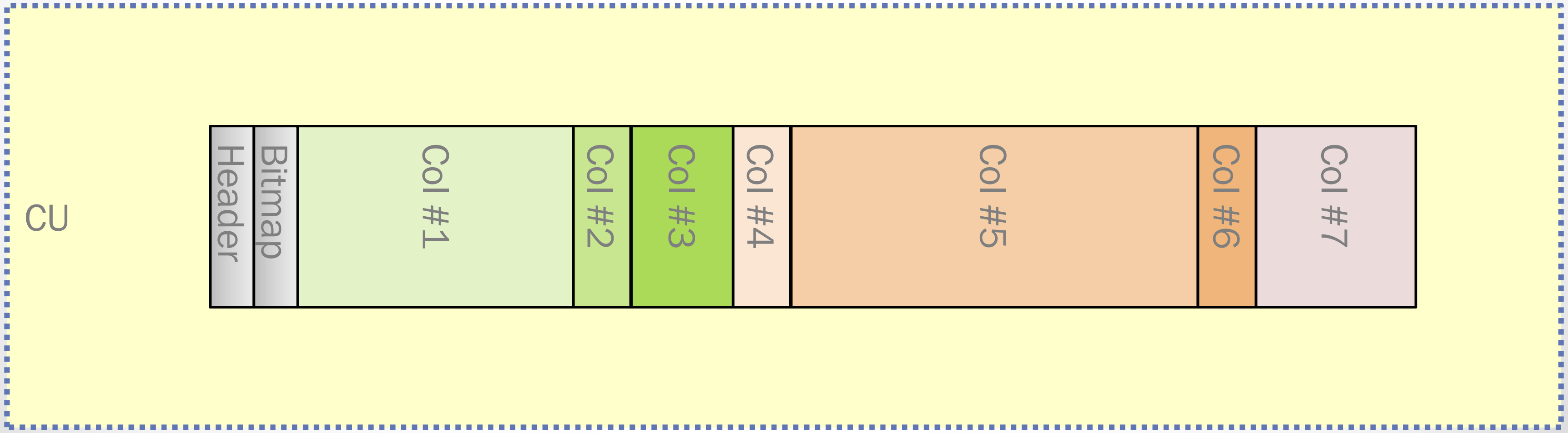
- BASIC supports direct-path INSERT and MERGE only
    - Use-cases include…
        - Data warehouse ETL into tables which are loaded and then never modified
            - INSERT /*+ APPEND */ /*+ APPEND_VALUES */
        - DDL operations
            - ALTER TABLE … MOVE [PARTITION], ALTER TABLE … [SPLIT|MERGE] PARTITION, CREATE INDEX, ALTER INDEX … REBUILD, SQL*Loader
- FOR OLTP supports all DML (i.e. INSERT, UPDATE, DELETE, MERGE)
    - Use-cases include…
        - Any applications
        - Tables which are modified frequently and constantly

HCC

- Built in to the base database 11gR2 and above
  - But only available on Oracle storage (i.e. Exadata, ZFS, and Pillar)
- Columnar storage pivots the idea of row storage
  - Each entry represents the values of a column across many rows
  - Rather than each entry representing values in a row across many columns
- Hybrid (*not true*) columnar storage
  - Each set of column values does contain values not all the rows in the table
    - Covers a limited set of rows only
- Advantages:
  - Achieve greater compression ratio
    - Compressing similar types of data, rather than different types of data
    - Less metadata, more payload
  - SELECT and UPDATE operations in SQL are column oriented
- Disadvantages:
  - Relational databases manage generally transactions by row
    - Row locks exist, but column locks do not exist
  - INSERT and DELETE operations are row oriented

HCC

- Compression unit (CU) is a logical data structure
  - ○ Header
    - Offsets and lengths of column entries
  - ○ Bitmap
    - Identifies deleted or updated (migrated) rows
  - ○ Column entries
    - Data values for N rows of an individual column
    - Each column entry compressed separately using specified compression algorithm (LZO, LZIP, or BZIP2)

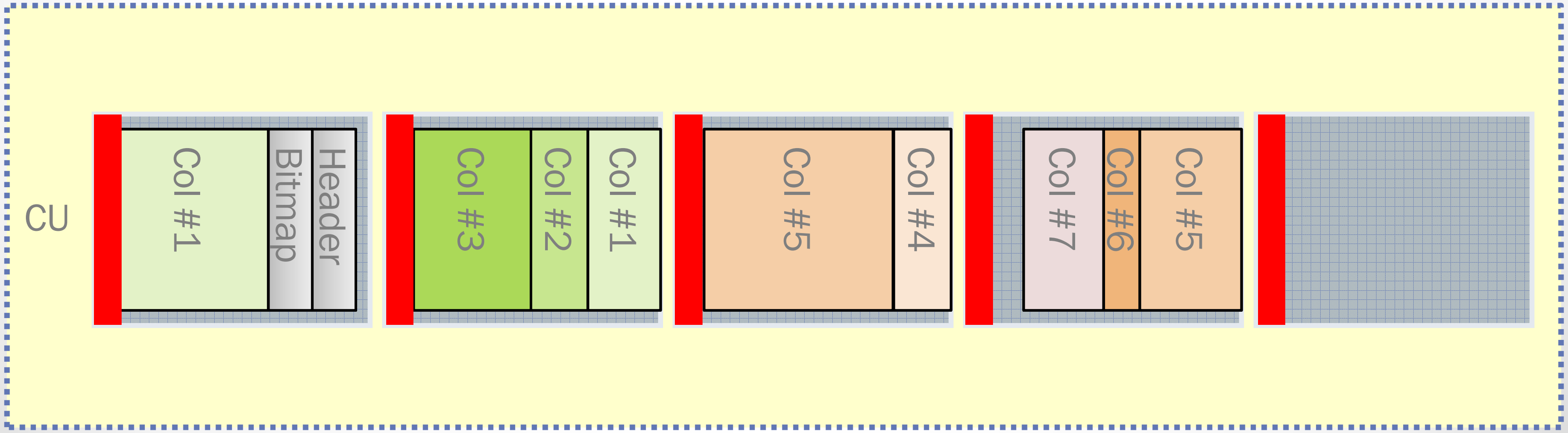


HCC

- Entire CU is stored as a single chained row entry
  - CU can be broken into chunks at any point, then chained across rows
- Online references:

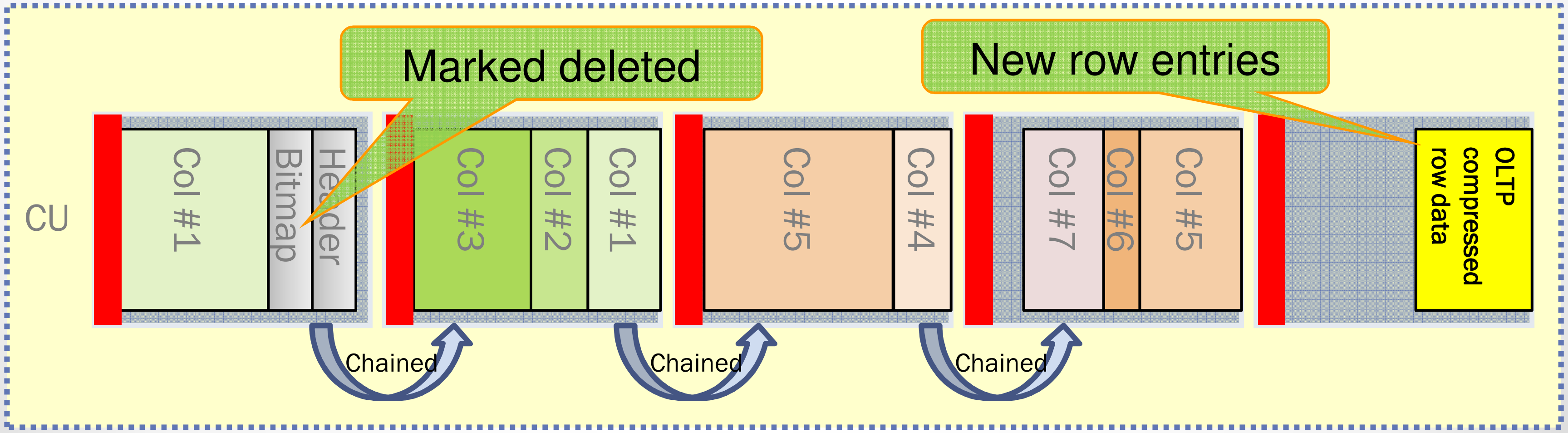
J Lewis http://jonathanlewis.wordpress.com/2012/07/20/compression_units/

Peter Brink http://www.slideshare.net/Enkitec/hybrid-columnar-compression-in-a-nonexadata-system



HCC

- When DML is performed on compressed data
  - INSERT
    - Inserted as a new row entry using OLTP compression
  - UPDATE
    - Marked as deleted in the bitmap entry of CU, then inserted as a new row entry using OLTP compression
  - DELETE
    - Marked as deleted in the bitmap entry of CU
- Deleted data is not removed, simply marked “deleted”



HCC

- More block dump output…

```
...
data_block_dump,data header at 0x2b8bbc16e67c
===============
tsiz: 0x1f80
hsiz: 0x1c
pbl: 0x2b8bbc16e67c
     76543210
flag=-0------
ntab=1
nrow=1
frre=-1
fsbo=0x1c
fseo=0x1f
avsp=0x3
tosp=0x3
...
```

#tables = 1, #rows = 1

Free space begin offset and end offset only 2 bytes apart

HCC

```
        mec_kdbh9ir2=0x0
                      76543210
        shcf_kdbh9ir2=----------
                  76543210
        flag_9ir2=--R-----      Archive compression: Y
                fcls_9ir2[0]={ }
0x16:pti[0]     nrow=1   offs=0
0x1a:pri[0]     offs=0x1f
block_row_dump:
tab 0, row 0, @0x1f
tl: 8033 fb: ------PN lb: 0x0  cc: 1
nrid:  0x04001491.0
col  0: [8021]
Compression level: 00 (Out of range)
 Length of CU row: 8021
kdzhrh: --------START_CU:
 00 00 1f 55 00 4c c7 01 f3 9b 62 b5 3f 7d bc 88 88 86 83 e1 c6 4e 91 01 72
...
```

Length of row = 8033

Flag “PN”: cont’d from Previous, cont’ing to Next

Length of CU chunk = 8021

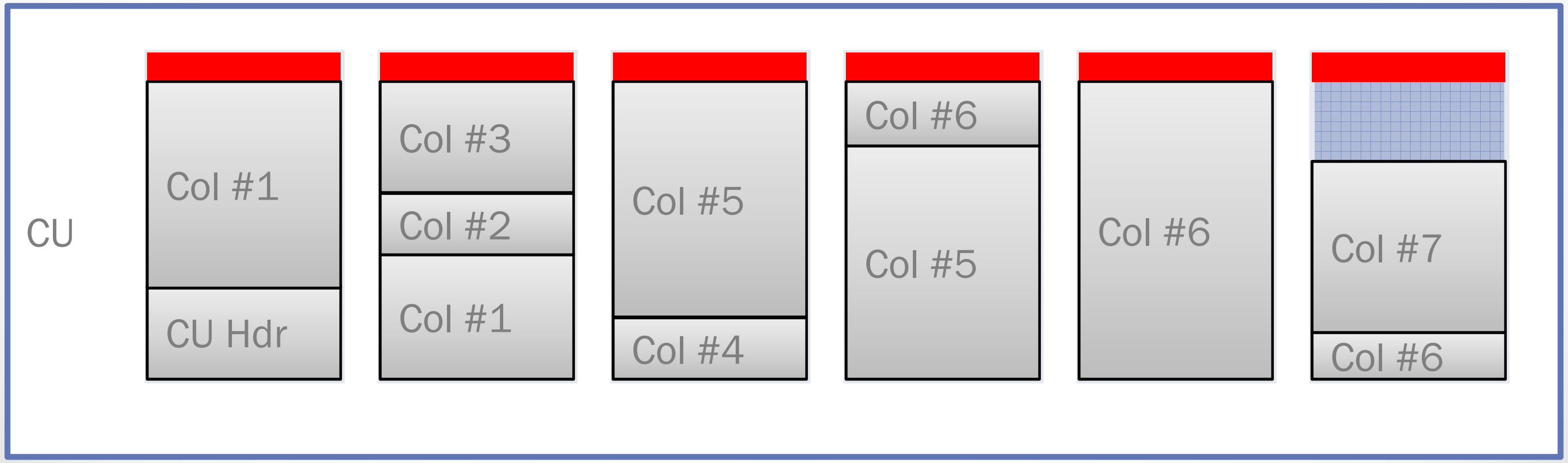
HCC

- COMPRESS FOR QUERY [LOW | HIGH]
  - Faster decompression for more frequent query usage
  - Lower compression ratio
- COMPRESS FOR ARCHIVE [LOW | HIGH]
  - Slower decompression for less frequent query usage
  - Higher compression ratio

Level	Algorithm	Expected compression ratio	Load Method
BASIC	Dedup	2:3 → 1:4 (60% → 25%)	Direct-path
OLTP	Dedup	2:3 → 1:4 (60% → 25%)	Any
QUERY LOW	LZO	1:5 → 1:10 (20% → 10%)	Direct-path
QUERY HIGH	ZLIB	1:10 → 1:18 (10% → 6%)	Direct-path
ARCHIVE LOW	ZLIB	1:12 → 1:20 (8% → 5%)	Direct-path
ARCHIVE HIGH	BZIP2	1:15 → 1:30 (6% → 3%)	Direct-path

HCC

- Hybrid columnar storage has usage implications
  - <u>Query</u>:
    - "`select col2`" will perform two (2) LIOs
    - "`select *`" will perform six (6) LIOs
  - <u>Upshot</u>:
    - let developers and ad-hoc query-writers know that columnar storage implies no wildcards for columns



DBMS_COMPRESSION

- Procedure GET_COMPRESSION_RATIO
  - Assists in determining if compression is worthwhile
  - Creates a temporary table with the specified type of compression, populated with a specified number of rows, returns actual compression statistics
- Function GET_COMPRESSION_TYPE
  - Determines how the specified row is compressed (or not)

GET_COMPRESSION_RATIO

```
declare
    v_blkcnt_cmp        number;      v_blkcnt_uncmp      number;
    v_row_cmp           number;      v_row_uncmp         number;
    v_cmp_ratio         number;      v_comptype_str      varchar2(4000);
begin
    DBMS_COMPRESSION.GET_COMPRESSION_RATIO(
        scratchtbsname => 'TOOLS',
        ownname => 'PROD_OWNER', tabname => 'ORDER_ACTIVITY', partname       => NULL,
        comptype     => DBMS_COMPRESSION.COMP_FOR_OLTP, /* QUERY_LOW|HIGH, ARCHIVE_LOW|HIGH */
        blkcnt_cmp   => v_blkcnt_cmp,      blkcnt_uncmp   => v_blkcnt_uncmp,
        row_cmp      => v_row_cmp,         row_uncmp      => v_row_uncmp,
        cmp_ratio    => v_cmp_ratio,       comptype_str   => v_comptype_str);
    dbms_output.put_line('Blocks compressed:           ' || v_blkcnt_cmp);
    dbms_output.put_line('Blocks uncompressed:         ' || v_blkcnt_uncmp);
    dbms_output.put_line('Rows per block compressed:   ' || v_row_cmp);
    dbms_output.put_line('Rows per block uncompressed: ' || v_row_uncmp);
    dbms_output.put_line('Compression Ratio:           ' || v_cmp_ratio);
    dbms_output.put_line('Comment:                     ' || v_comptype_str);
end;
/
```

Trailing NULLCOLs

- A form of compression that exists in all current versions of Oracle…
  - Takes advantage of how columns are stored within rows
    - Row
      - Row-header :: column-piece [:: column-piece …]
      - Column-piece
        - Non-NULL data values
          - Length :: data
        - NULL data values
          - Non-trailing placeholder = 0xFF
          - Trailing NULL values are not stored

Trailing NULLCOLs

- A form of compression that exists in all current versions of Oracle…
  - ○ Takes advantage of how columns are stored within rows
    - Row
      - ○ Row-header :: column-piece [:: column-piece …]
      - ○ Column-piece
        - Non-NULL data values
          - ○ Length :: data
        - NULL data values
          - ○ Non-trailing placeholder = 0xFF
          - ○ **Trailing NULL values are not stored**

Trailing NULLCOLs

- Case study
  - Oracle's Demantra product
    - Application for demand management, sales and operations planning, projections, and what-if analysis
  - Central fact table is named SALES_DATA
    - Frequently customized with additional columns
  - SALES_DATA had over 750 columns and 250m rows
    - All analytic queries performed FULL table scans on SALES_DATA, over and over and over and over…
  - It turned out that the SALES_DATA table had only 40-50 out of 750 columns populated on average
    - DBMS_STATS showed average row length of 766 bytes

Trailing NULLCOLs

- Compress?
  - Database is 10gR2
    - Couldn't use BASIC compression because SALES_DATA is frequently updated by Demantra application
      - Even OLTP compression would not work well
    - More than 255 columns as well
- Solution
  - Rebuild SALES_DATA with columns ordered by NUM_NULLS descending
    - Then load all rows into the new table
    - Average row length dropped from 766 to 102 bytes
      - 7:1 compression ratio
    - Total table size dropped from 190 Gb to about 26 Gb

Trailing NULLCOLs

- OK, but we need a way to determine if a table would benefit from such a rebuild
  - ○ Without having to test it
- Procedure CARL (Calculate Average Row Length)
  1. Queries rows and calculates current average row length
  2. Sorts columns by NUM_NULLS DESC from DBA_TAB_COLUMNS view
  3. Recalculate average row length
- Download from http://EvDBT.com/scripts/
  - ○ Script "carl.sql"
  - ○ Prerequisites…
    - CARL relies on **good** column statistics
    - Uses DBMS_OUTPUT package to output results
      - ○ Enable SERVEROUTPUT ON in SQL*Plus

Prerequisites and storage

- TRAILING NULLCOLS compression
  - No prerequisites for database version, server platform, or storage prerequisites
  - The only prerequisite is a large number of frequently-NULL columns and an application that does not perform “blind” SELECT and INSERT statements
- BASIC compression
  - Database version 9iR2 or above, no server platform or storage prerequisites
  - Support DW/BI applications best
- OLTP compression
  - Database version 11gR1 or above with licensing for Advanced Compression option
  - No server platform or storage prerequisites
  - Supports OLTP applications best, DW/BI applications probably less well
- HCC compression
  - Database version 11gR2 or above, only on Oracle storage (Exadata, ZFS, Pillar)
  - Supports all types of applications, but DW/BI applications most effectively

Points to ponder…

- **What happens when you attempt to access HCC data on non-Oracle (a.k.a. non-HCC-enabled) storage?**
  - ORA-64307 "hybrid columnar compression is not supported for tablespaces on this storage type"
- **Luis Moreno Campos' blog**
  - http://ocpdba.wordpress.com/2011/05/06/recover-hcc-compressed-tables-to-non-exadata-storage/
  - Testing with RMAN, moving HCC data from Exadata to non-Oracle storage
  - RMAN backup and restore operations are successful. *Why*?
  - INSERTs are successful. *Why*?
  - SELECTs, UPDATEs, and DELETEs fail. *Why*?
  - ALTER TABLE … MOVE is successful. *Why*?

Summary

- Multiple ways to compress table data
  - Two ways are provided and supported by Oracle
    - in certain versions, some need patching
  - One way is possible by understanding how Oracle stores data
- Compression can improve performance
  - Understand each and every type of compression and how they work
- Compression is primarily intended for dormant data
  - But Oracle has done a good job to handle volatile data well also
  - Please note how uncompressed data within compressed segments are handled
    - BASIC/OLTP: uncompressed row data is preceded by flag bytes
    - HCC: modifications to compressed data handled as OLTP compress

References

- Jonathan Lewis - http://jonathanlewis.wordpress.com/2012/07/20/compression_units/ http://jonathanlewis.wordpress.com/2011/10/04/hcc/
- Peter Brink - http://www.slideshare.net/Enkitec/hybrid-columnar-compression-in-a-nonexadata-system
- Graham Thornton - http://www.orafaq.com/papers/dissassembling_the_data_block.pdf
- Uwe Hesse - http://uhesse.com/2011/09/12/dbms_compression-example/

hroug

hrvatska udruga oracle korisnika

- Email: tim.gorman@delphix.com
- Blog: http://EvDBT.com/
  - Papers: http://EvDBT.com/papers/
  - Scripts: http://EvDBT.com/scripts/
- Twitter: @TimothyJGorman
- Mobile: +1 (303) 885-4526

DELPHIX