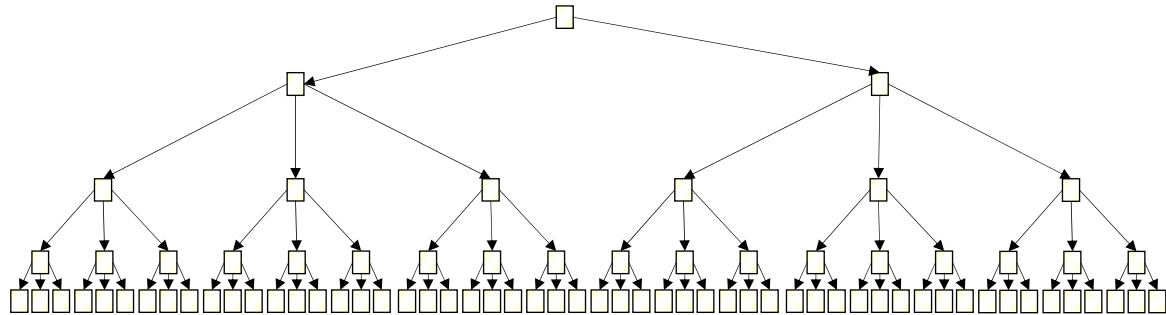


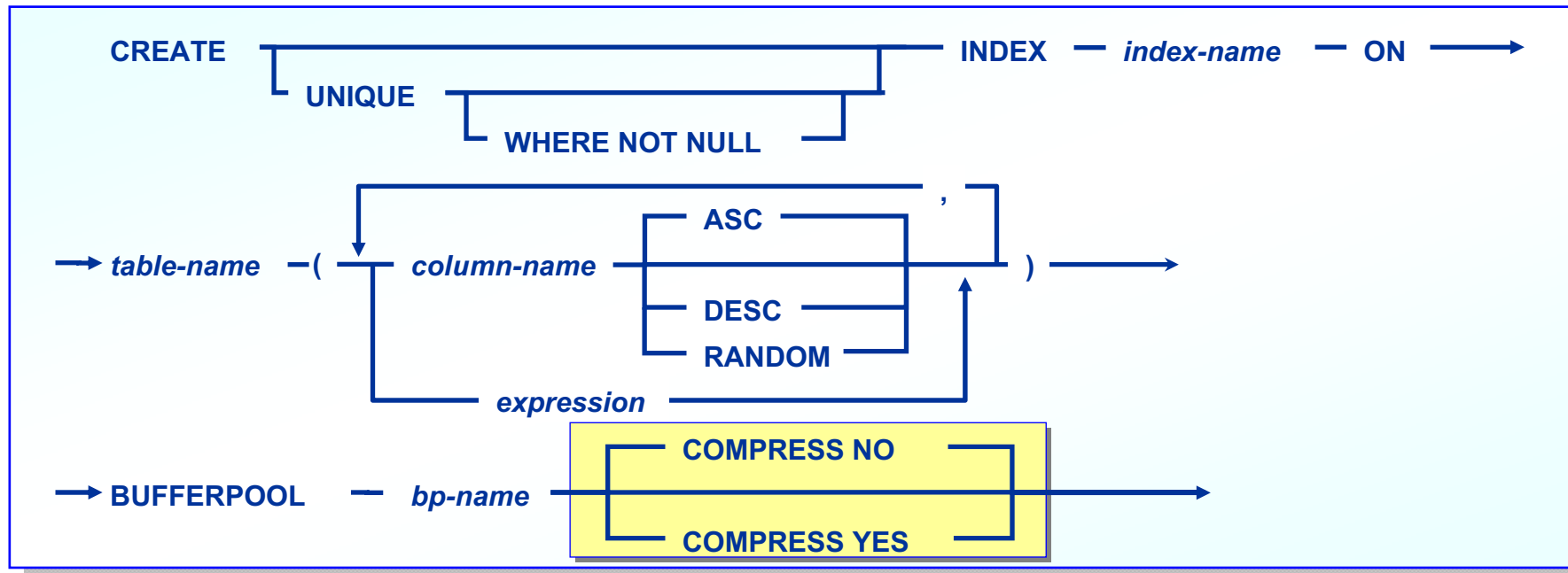
Index Compression



- Prior to V9, DB2 only supported compression for tablespaces
- In V9, we introduce Index Compression



Index Compression DDL

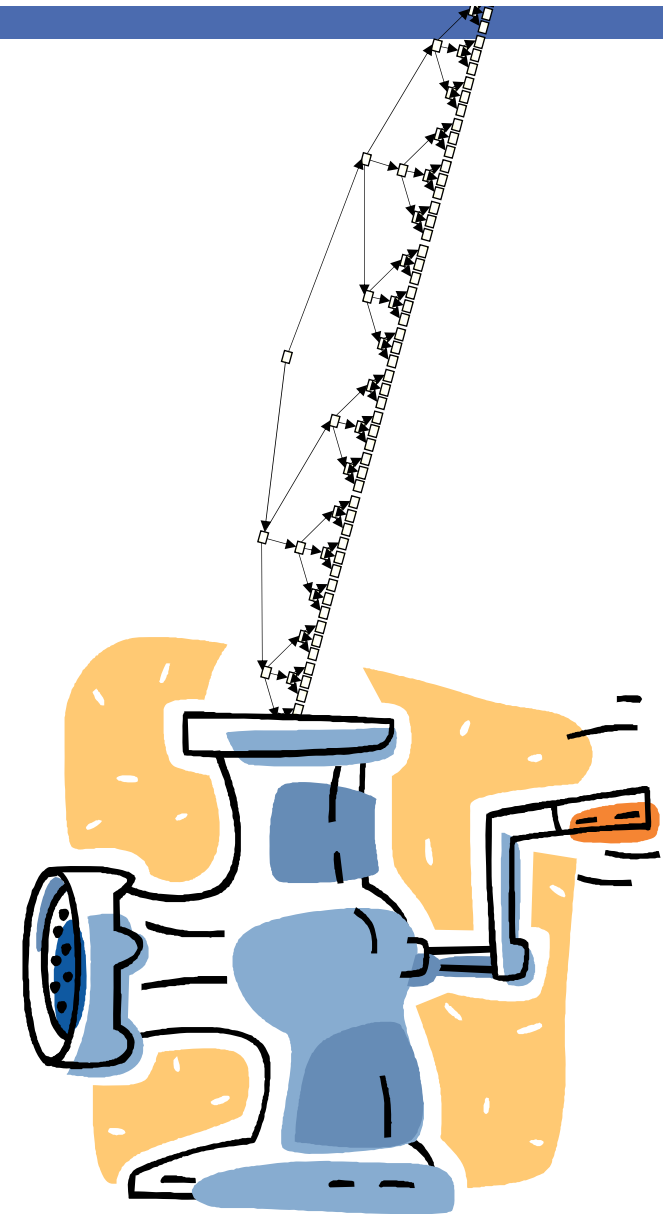


- You enable compression by CREATing or ALTERing the index **COMPRESS YES**
- Index Compression is ***not supported in some cases:***
 - For versioned indexes – **YET....**
 - For indexes in 4K bufferpools
- You may also ALTER an index to change the **COMPRESS** option
 - -676 SQLCODE for non-DB2 managed datasets with an incompatible CISIZE
 - -676 SQLCODE when altering to **COMPRESS YES** with 4K bufferpool
 - You can alter bufferpool and the **COMPRESS** option in the same statement



Index Compression – How it works

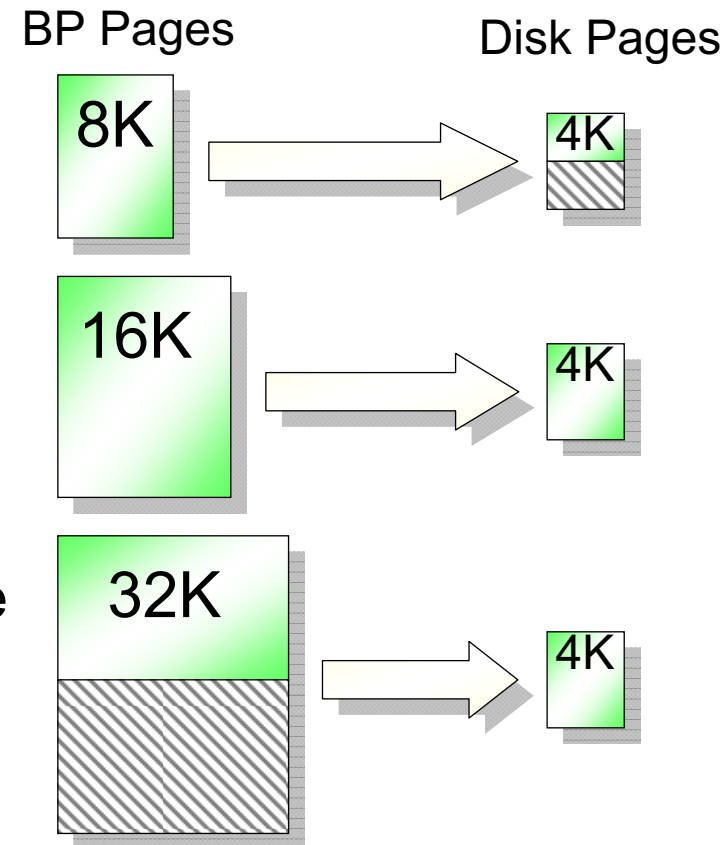
- There is ***NO COMPRESSION DICTIONARY***
 - Compression is done on the fly
 - Indexes will be compressed when initially populated
 - A Hybrid approach is used
- In the bufferpool, pages are physically expanded to the page size of the bufferpool
 - 8K, 16K, or 32K
- On disk, the pages are always 4K



Index Compression – How it works

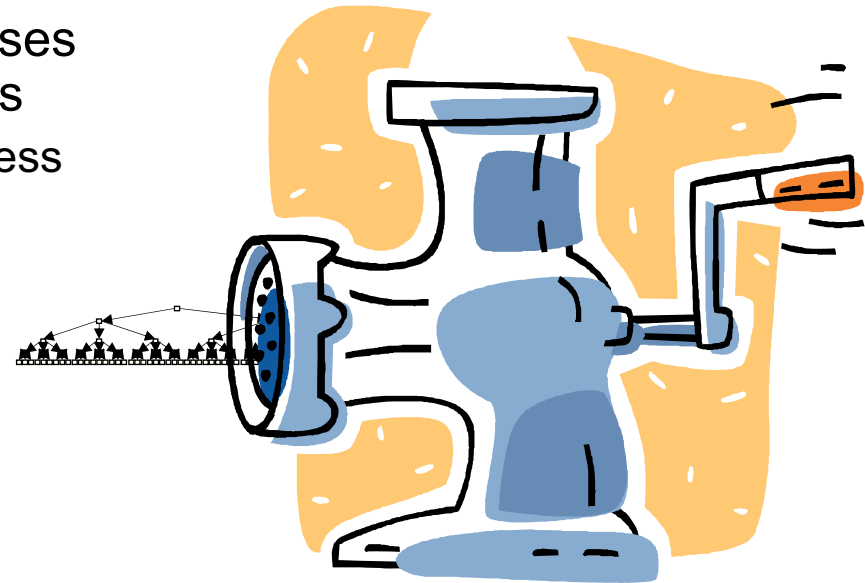
- **The compression you can achieve is limited by the bufferpool you use**
 - For 8K, up to 51% compression
 - For 16K, up to 76% compression
 - For 32K, up to 88% compression
- DB2 ensures that the bufferpool pages will fit onto the pages on disk.
- If the data doesn't compress well, and you use large page sizes, you may waste bufferpool space
- **It is VERY IMPORTANT to choose an appropriate page size**
- What other factors affect the choice of page size for compressed indexes???

Assume Keys compress 4:1



Index Compression – What to consider when choosing a page size

- The compressability of the data
 - DSN1COMP helps estimate compression ratios
- The Workload/Usage Patterns on the index
 - Pages are decompressed on read and compressed on write
 - Synchronous I/Os for Random Accesses may incur synchronous CPU penalties
 - Larger pages take longer to decompress
 - Prefetch I/O for index scanners can decompress asynchronously
 - Still uses CPU, but doesn't affect elapsed time
 - In fact, ***compression may improve elapsed time for I/O bound queries***



Index Compression – Page Size Considerations

■ Bufferpool considerations

- Only Leaf Pages are compressed
 - Only 4K of the non-leaf buffers will be used
 - Reductions in non-leaf bufferpool space don't apply for compressed indexes
 - Compressed indexes will behave much like non-compressed indexes with 4K pages
 - Index non-leaf data typically occupies a small percentage of the index space
- Hit Ratio in the bufferpool
 - If you get high buffer hit ratios, the compression overhead is minimized
 - As long as the page remains in the bufferpool, it stays uncompressed



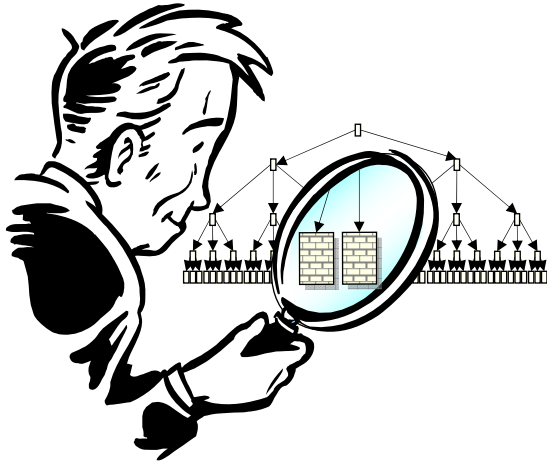
■ Rules of Thumb

- Consider using compression for warehouse applications
- Be aware of the CPU overhead for OLTP applications

- *The best way to understand the impact is to try it out!*



DSN1COMP



```
//DSN1COMP JOB 'USER=$$USER','<USERNAME:JOBNAME>',  
//      MSGCLASS=A,MSGLEVEL=(1,1),  
//      CLASS=A,USER=ADMF001,PASSWORD=GOODSTUF,  
//      REGION=4096K  
//JOBLIB  DD DSN=USER.TESTLIB,DISP=SHR  
//      DD DSN=DB2A.SDSNLOAD,DISP=SHR  
/*ROUTE PRINT STLVM14.BLYE  
//*****  
/*  STEP1 - Run DSN1COMP on the index  
//*****  
//STEP1 EXEC  
PGM=DSN1COMP,PARM='LEAFLIM(1000)'  
//SYSPRINT DD SYSOUT=A  
//SYSDUMP  DD SYSOUT=A  
//SYSABEND DD SYSOUT=A  
//SYSUT1   DD DSN=TESTCAT.DSNDBD.DB1.I1.I0001.A001,DISP=SHR  
//SYSUT2   DD SYSOUT=A
```

- DSN1COMP provides an **estimate** of how well an index will compress
- Estimates how the bufferpool will behave for given page sizes
 - i.e. will buffer space be wasted for the index
- The LEAFLIM option limits how many leaf pages are examined
 - If not specified, all leaf pages in the dataset will be used



DSN1COMP – Example Table

Sample Table for Compression Tests

- Randomly and sequentially Generated Data
- Large Tablespace
- 1 Million Rows
- 7 columns

–RegionId	Integer	10 distinct values
–StoreId	Integer	50 distinct values
–DeptId	Integer	50 distinct values
–BigSeqNo	BigInt	+1/row – Unique Seq.
–State	Char(16)	50 distinct values
–City	Char(16)	50 distinct values
–Date	Date	+1 day/1000 rows on avg

```
CREATE TABLE T1(  
    regionid INTEGER,  
    storeid  INTEGER,  
    deptid   INTEGER,  
    bigseqno BIGINT,  
    state    VARCHAR(32),  
    City     VARCHAR(32),  
    date     DATE )  
PARTITION BY (regionid)  
    ( PART 1 VALUES (100),  
      PART 2 VALUES (200),  
      PART 3 VALUES (300),  
      PART 4 VALUES (400) )  
IN DB1.TS1;
```



Insert Statement for table

INSERT INTO T1

```
with dt(regionid, storeid, deptid, bigseqno,
      state, City, date, rstate, rcity, r1000 ) as
( SELECT int(rand(10)*10), int(rand(11)*50), int(rand(12)*50), 1,
      'California', 'American Canyon', current date,
      int(rand(15)*50), int(rand(16)*50), int(rand(17)*1000)
FROM SYSIBM.SYSDUMMY1
UNION ALL
SELECT int(rand(1)*10), int(rand(2)*50), int(rand(3)*50), bigseqno+1,
      case rstate
        when 0 then 'Alabama'           when 1 then 'Alaska'
        when 2 then 'Arizona'          when 3 then 'Arkansas'
        when 4 then 'California'        when 5 then 'Colorado'
        when 6 then 'Conneticut'        when 7 then 'Delaware'
        when 8 then 'Florida'           when 9 then 'Georgia'
        when 10 then 'Hawaii'           when 11 then 'Idaho'
        when 12 then 'Illinois'         when 13 then 'Indiana'
        when 14 then 'Iowa'             when 15 then 'Kansas'
        when 16 then 'Kentucky'         when 17 then 'Louisiana'
        when 18 then 'Maine'            when 19 then 'Maryland'
        when 20 then 'Massachusetts'    when 21 then 'Michigan'
        when 22 then 'Minnesota'        when 23 then 'Mississippi'
        when 24 then 'Missouri'         when 25 then 'Montana'
        when 26 then 'Nebraska'         when 27 then 'Nevada'
        when 12 then 'New Hampshire'    when 29 then 'New Jersey'
        when 30 then 'New Mexico'        when 31 then 'New York'
        when 32 then 'North Carolina'    when 33 then 'North Dakota'
        when 34 then 'Ohio'              when 35 then 'Oklahoma'
        when 36 then 'Oregon'            when 37 then 'Pennsylvania'
        when 38 then 'Rhode Island'      when 39 then 'South Carolina'
        when 40 then 'South Dakota'      when 41 then 'Tennessee'
        when 42 then 'Texas'             when 43 then 'Utah'
        when 44 then 'Vermont'           when 45 then 'Virginia'
        when 46 then 'Washington'       when 47 then 'West Virginia'
        when 48 then 'Wisconsin'        when 49 then 'Wyoming'
        else 'Unknown'
      end,
end,
```

```
case rcity
  when 0 then 'Adelanto'
  when 2 then 'Alameda'
  when 4 then 'Alhambra'
  when 6 then 'Altura'
  when 8 then 'American Canyon'
  when 10 then 'Anderson'
  when 12 then 'Antioch'
  when 14 then 'Arcadia'
  when 16 then 'Arroyo Grande'
  when 18 then 'Arvin'
  when 20 then 'Atherton'
  when 22 then 'Auburn'
  when 24 then 'Avenal'
  when 26 then 'Bakersfield'
  when 12 then 'Banning'
  when 30 then 'Beaumont'
  when 32 then 'Bell Gardens'
  when 34 then 'Belmont'
  when 36 then 'Benicia'
  when 38 then 'Beverly Hills'
  when 40 then 'Biggs'
  when 42 then 'Blue Lake'
  when 44 then 'Bradbury'
  when 46 then 'Brea'
  when 48 then 'Brisbane'
  else 'Unknown'
end,
date + case r1000 when 0 then 00000001. else 00000000. end,
int(rand(21)*50), int(rand(22)*50), int(rand(23)*1000)
FROM dt
WHERE bigseqno < 100000
)
select regionid, storeid, deptid, bigseqno,
      state, city, date
from dt;
```



DSN1COMP Output Part 1

**CREATE UNIQUE INDEX I1
ON T1 (bigseqno)**

PCTFREE 0;

- Large Tablespace
- 1 Million Rows
- 1 Key Column
 - BigSeqNo BIGINT
 - +1/row – Unique Seq.

DSN1944I DSN1COMP INPUT PARAMETERS
PROCESSING PARMS FOR INDEX DATASET:
NO LEAFLIM WAS REQUESTED
DSN1940I DSN1COMP COMPRESSION REPORT

4,220 Index Leaf Pages Processed

1,000,000 Keys Processed

1,000,000 Rids Processed

16,602 KB of Key Data Processed

7,837 KB of Compressed Keys Produced

- **$7,837 / 16,602 = 47.2\%$**
 - Doesn't count non-leaf or page header overhead
- This gives an idea of how well the keys compress
- More detailed analysis follows....



DSN1COMP Output Part 2

```
CREATE UNIQUE INDEX I1
ON T1 (bigseqno)
  - BigSeqNo    BIGINT
    +1/row - Unique Seq.
```

- Note these are approximations
 - May be a few percent off
 - Depends on how keys are laid out on pages
 - Doesn't consider non-leaves
 - Don't expect exact numbers
- **For this index, 8K might be a good choice**

EVALUATION OF COMPRESSION WITH DIFFERENT INDEX PAGE SIZES:

8 K Page Buffer Size yields a
51 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
49 % of the original index's Leaf Page Space
No Bufferpool Space would be unused

16 K Page Buffer Size yields a
53 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
47 % of the original index's Leaf Page Space
46 % of Bufferpool Space would be unused to
ensure keys fit into compressed buffers

32 K Page Buffer Size yields a
53 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
47 % of the original index's Leaf Page Space
73 % of Bufferpool Space would be unused to
ensure keys fit into compressed buffers



DSN1COMP Example 2

CREATE INDEX I2 ON T1 (

**regionid, storeid,
deptid, bigseqno) ...;**

- RegionId Integer
10 distinct values
- StoreId Integer
50 distinct values
- DeptId Integer
50 distinct values
- BigSeqNo BigInt
+1/row – Unique Seq.

- **For this index, 16K would be a good choice**

EVALUATION OF COMPRESSION WITH DIFFERENT INDEX PAGE SIZES:

**8 K Page Buffer Size yields a
51 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
49 % of the original index's Leaf Page Space
No Bufferpool Space would be unused**

**16 K Page Buffer Size yields a
76 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
24 % of the original index's Leaf Page Space
No Bufferpool Space would be unused**

**32 K Page Buffer Size yields a
78 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
22 % of the original index's Leaf Page Space
43 % of Bufferpool Space would be unused to
ensure keys fit into compressed buffers**



DSN1COMP Example 3

CREATE INDEX I5

ON T1 (regionid, state, city, bigSeqno)

- RegionId Integer
10 distinct values
- State Char(16)
 - 50 distinct values
- City Char(16)
 - 50 Distinct values
- BigSeqNo BigInt
+1/row – Unique Seq.

- 16K and 32K are both options
- **What's more Important to you?**
 - Disk space or Bufferpool space

EVALUATION OF COMPRESSION WITH DIFFERENT INDEX PAGE SIZES:

8 K Page Buffer Size yields a
51 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
49 % of the original index's Leaf Page Space
No Bufferpool Space would be unused

16 K Page Buffer Size yields a
76 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
24 % of the original index's Leaf Page Space
No Bufferpool Space would be unused

32 K Page Buffer Size yields a
85 % Reduction in Index Leaf Page Space
The Resulting Index would have approximately
15 % of the original index's Leaf Page Space
16 % of Bufferpool Space would be unused to
ensure keys fit into compressed buffers



Index Compression Summary

- No Compression Dictionary is used
- The choice of bufferpool is very important
 - Too small and you limit your compression ratio
 - Too large and you waste bufferpool space
 - DSN1COMP helps you choose
- Effects on system performance vary
 - Compress/Decompress is done at I/O time
 - May be synchronous or asynchronous
 - For in-memory indexes, overhead is minimal
 - Once the pages are decompressed, they remain decompressed in the bufferpool

