OpenVMS/RMS Indexed File Tuning and the Million Dollar Bit

Hein van den Heuvel

Performance Engineer

HP, Enterprise Solutions Partners

HP WORLD 2003

Solutions and Technology Conference & Expo

Introduction and overview

- This presentation is based on, and much similar to, earlier Decus and CETS submissions (similar slides)
- The focus shifts to one specific tuning/design issue which has proven to have tremendous performance impact: Duplicate Key Chains.
- This presentation consists of:
  - General performance remarks
  - Picture of an indexed file on disk structure
  - Detailed tuning points including Duplicate Key Chains
  - Nice to knows (not presented)
  - File patching (not presented)

The Million Dollar bit

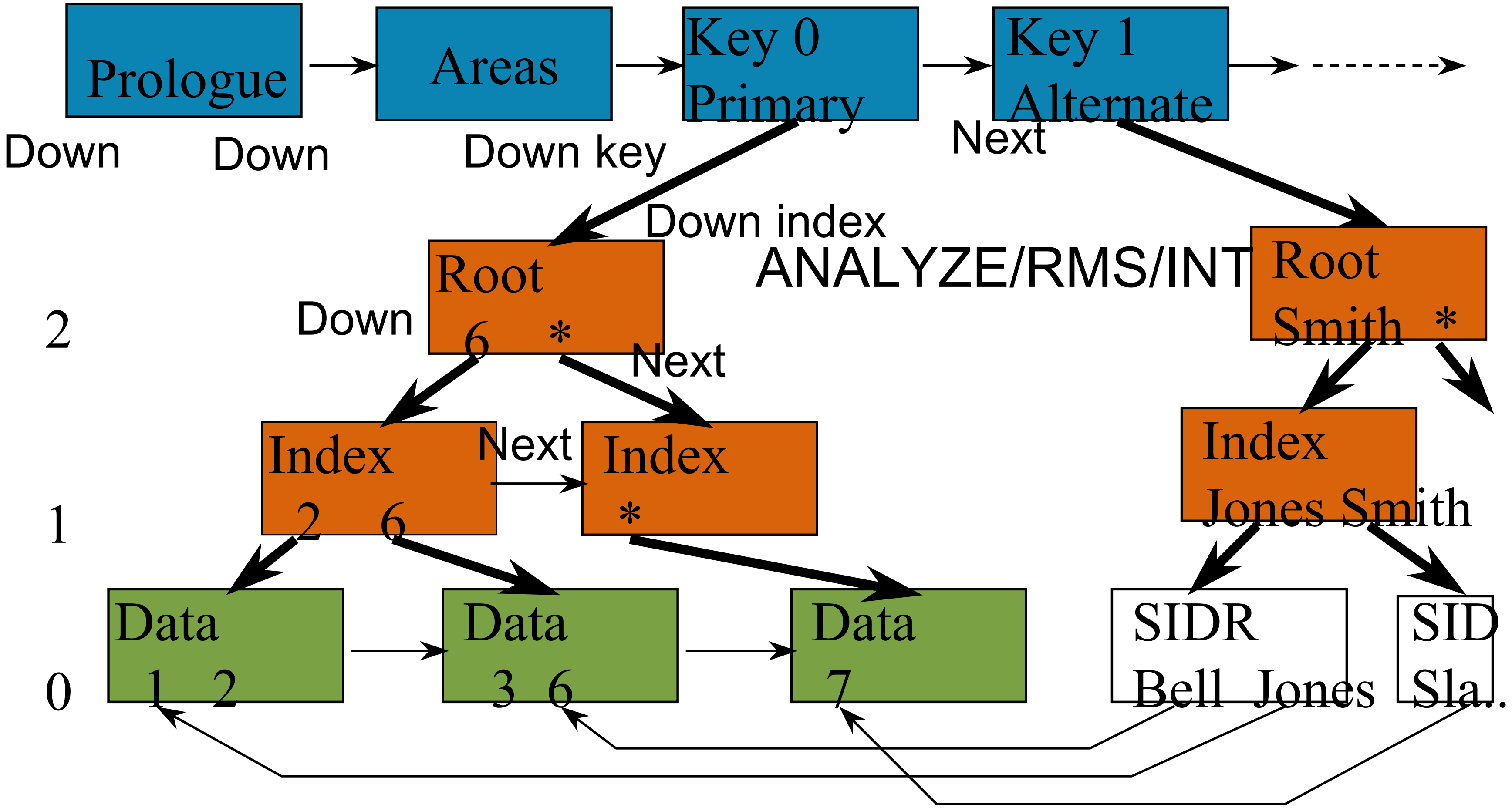
- The title is of course a bit of a teaser, but really for more than one customer a single bit set incorrectly has cost them more then a million dollars in oversized servers and lost production.
- The bit is called XAB$V_NUL and instructs RMS not to bother to maintain the index structure for one particular key value.
- Maintaining excessive duplicate keys (millions of duplicates for a single key value) can cost thousands of READ IOs causing a single record insert to take minutes instead of being a sub-second operation.
- See also **OpenVMS Technical Journal V2** (July 2003)

Get some, any, RMS training.

- If this was a 'Real' Database application (Oracle / SQLserver / MySQL), would you not have had:
  - a dedicated DB administrator
  - a handfull of specialized DB designers/programmers?
  - Invested in weeks (years?) of training
  - Hired mostly experienced database personel?

- RMS may be for free, but is still needs some TLC!
  - How much have you invested in RMS experience?
  - Used formal training? (HP, Bruden, Parsec)
  - Allow a engineer to build hands-on experience?

Anatomy of an Indexed File



Design For Performance,
Work with the numbers.

- RMS does (unfortunately) no magic (no optimizer)
- All RMS activity is rather predictable
- Calculate/predict IO resource needs
- Instrument application
  - Display records or work-units processed for batch.
  - `SYS$GETJPI / LIB$SHOW_TIMER`
  - `SET FILE /STATISTICS` followed by
    - `MONITOR RMS /ITEM=CACH` …
    - `RMS_STATS & RMS_TUNE_CHECK` tools on VMS Freeware
  - `SHOW PROCESS/ ACCOUNTING`
  - Compare Expected and Actual measurements

Production Systems

- Regular Converts for indexed files
  - Daily / Weekly / Monthly / Yearly based on usage
  - Combats both Internal and External fragmentation
  - Opportunity to tune
  - Delivers basic statistics: record count
  - Implied backup
  - Implied sanity check
  - Corruption might not yet show in production

Production Systems

- Periodic Analyze / Optimize
  - Index root level(s) still as planned ?
  - Bucket size still adequate ?
  - Number of (global) buffers still adequate ?
  - File size still as planned ?
- Standard tools
  - $ANALYZE/RMS $MONITOR…
  - $EDIT/FDL/NOINTERACTIVE
  - $DIRECTORY/FULL

Performance still bad ?

- Is there a bottleneck ?
  - CPU : $MONI MODE

    INTERRUPT = Devices and cluster locks

    MPSYNC = Kernel access serialization (locks?)

    KERNEL = Locks and QIO and logical names

    EXECUTIVE = RMS (or ORACLE or…)

    USER = real work!
  - IO : $MONI DISK, IO, PROC/TOPDIO

    Watch out for MONI DISK/TOPQ because serial reads will NOT cause a queue and yet still be an IO problem!
  - LOCKS: $MONI LOCK, DLOCK

Performance still bad ?

- No IO problem, No CPU problem, and still No Performance? Probably Serialization!
- Basic application understanding
  - "Master record with 'next key' value"
  - mailbox to go through?
- Hot files
  - DECps / Advise (Now C.A. previously VPA)
  - Viewpoint (Datametric)
  - DLB dynamic load balancer (TTI)

RMS Tuning : … Just Do It!

- RMS Tuning can be as EASY as
  - $set rms/system/indexed/buffer=20
  - $set rms/system/extend=2000
  - $set rms/system/sequent/buffer=4/block=32
  - $set file /global_buffer=50 *.DAT

RMS Tuning: Just Do It

- The 80/20 rule applies
  - A little tuning can help a whole lot
  - Ideal tuning requires a whole lot
- Just get it roughly right and your end-users will love you for it

Code does not break

- Many performance options are transparent
- Some require quota / resource adjustment

Code does not break

Parameter	Transparent?	Adjustment
Number of Buffers	Yes	ENQLM
Size of buckets	Yes	WSQUOTA
File Extend Quantity	Yes	Clustersize?
File/Area Placement	Yes	Spindles
Global Buffers	Yes	GBLPAGFIL, GBLPAGES,…

What to look for ? (Details on next slides)

- Bucket Size
- Number of Local and Global Buffers
- Duplicate Alternate Key Chains (SIDR)
- Number of Index Levels (depth)
- Allocation and Extend Sizes
- Placement in Space and Time
- Bucket splits
- Compression
- Deleted key ranges

What is a Bucket anyway?

- Unit of transfer to IO device
  - NOT directly dependent on CLUSTERSIZE
  - FINE Tuning with clustersize helps
- Contains Data or Index records
  - Records can NOT cross bucket boundaries
  - If records do not fit, a BUCKETSPLIT occurs
- Different sizes for different usage in file
  - Data / Level-1 Index / Rest-of-Index / Alternate keys
  - RMS memory buffers size = largest bucket size in file
- Bucket Size is a Permanent file attribute

Bucket Size Cheat Sheet

Bucket Size	SMALL	MEDIUM	BIG
Size (Blocks)	1 - 6	6 - 24	24+
Size (Records)	1 - 10	10 - 300	300+
Index Levels	4+	2 or 3	1 or 2
CPU time	0 ms	0.1 ms	0.5 ms

Bucket Size Cheat Sheet

Bucket Size	SMALL	MEDIUM	BIG
IO Transfer time @ 5 MB/Sec	0 ms	1 – 2 ms	3+ ms
Working Set	Minimal	Low Impact	Add Pages
Contention	Low	Medium	High
Disasters	Low risk Low impact	Low risk	Higher risk Big impact

What about buffers

- A buffer is a chunk of RMS maintained memory to read a bucket (or to write from)
- PUTs and UPDATEs need many buffers if multiple keys are in use
- Indexed file Sequential GETs need only 1 buffer
- Keyed GETs need 1 for each index level plus one for data (and perhaps for RRV)

What about LOCAL buffers?

- RMS was frugal : just 2 buffers! (pre 5.4) now ‘deepest index + 2’ is the default
- Set on CONNECT time (HLL OPEN call)
- Defaults can be set for at process and system level through $SET RMS /BUF…
  - but they apply to every file opened!
- Local buffer max = 254 through RAB
- Specify up to 32K buffers through XABitm

What about GLOBAL Buffers?

- GLOBAL BUFFERS REALLY WORK
- Share buffers by all accessors on a node
- Efficient in clusters (no change broadcasts)
- Expect to SAVE Memory.
  - » Sure, they use memory, but with many concurrent users they often save memory by requiring fewer local buffers per user (100 is less than 30 times 6)
- Great to cache (all) index buckets
  - » Primary and Alternate indexes alike

What about GLOBAL Buffers

- Great for read and write accessed files
  - PUTs first need to read, to know where to write.
  - Can write from global cache, avoiding next read
- Gotchas:
  - Up to VMS 7.2 (patch kit for 7.1) a per node serial sequential scan was used to locate buckets under protection of an exclusive, local, lock.

```
SHOW PROC/RMS=GBHSH  (VMS 7.2+)
```

  - Needs LOCKs to work, thus needs shared access
  - 'Deferred Write' option forces a local buffer copy
  - Not used for DCL opened (process permanent) files
  - Sequential readers may trash cache (bad citizens)

How many GLOBAL Buffers

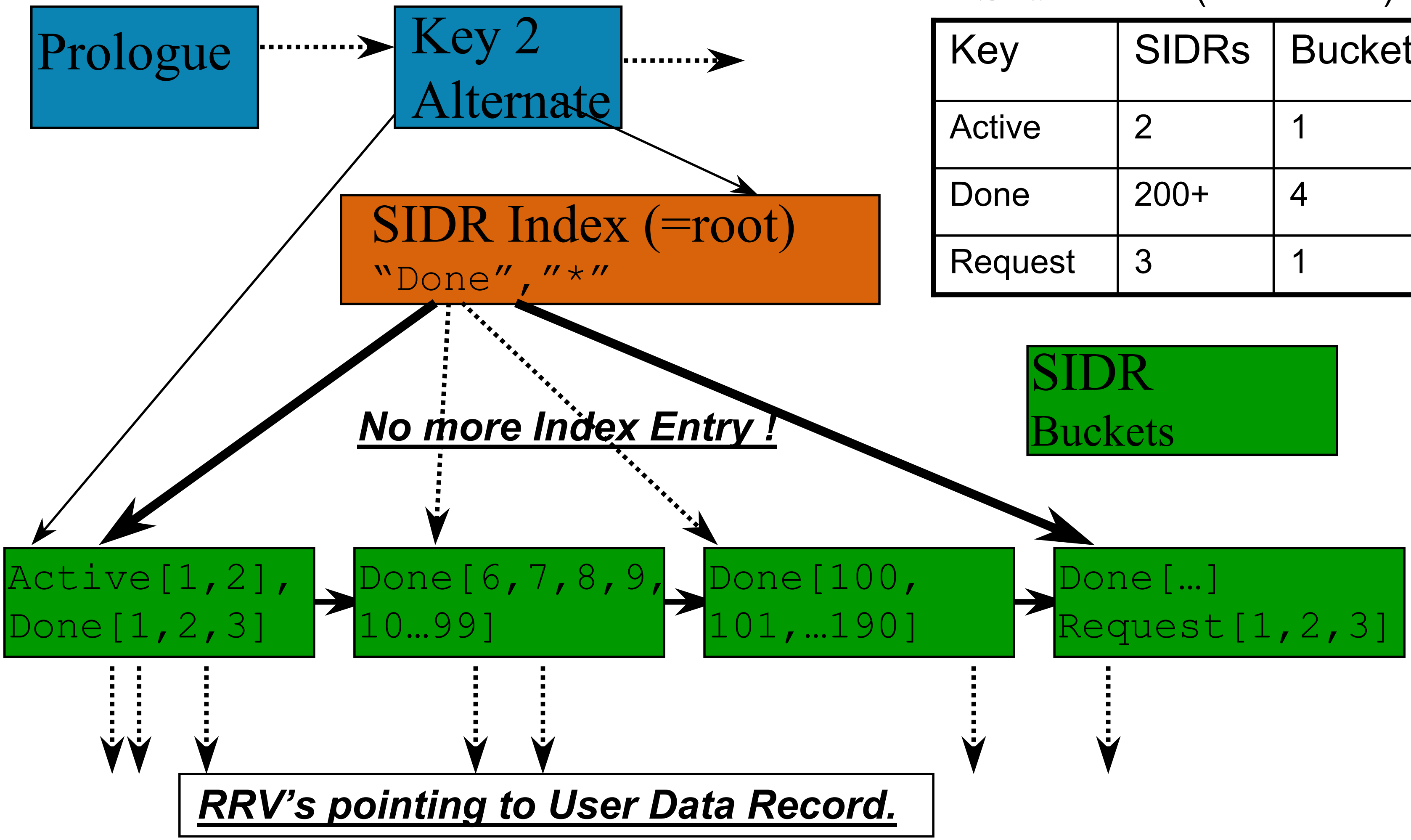
- ‘None’ is the only wrong answer.
- RMS Limit is 32K, Practical 200 - 2000 ?
- 42? One for each user?
- Cache (top of) index and then some?
- Goal is 80 - 99% hit rate.
  - But really the goal is an acceptable IO rate.
  - 99% hit rate on 5000 accesses/second is still 50 IO/second
  - 90% hit rate on 50 accesses/second is only 5 IO/second
- Treat as memory budget
  - You have xxxx pages in the bank to start with.
  - spent pages wisely, don’t spend all in one place.

Duplicate Chains, Overview.

- RMS stores duplicates as 7 byte RRV pointers in Secondary Index Data Records (SIDR)
- If SIDR is larger then Bucket Size a new Bucket is allocated
- RMS maintains duplicates in arrival order
  - This is deliberate documented behavior which some applications count on.
  - Last record added with key value ‘Request’ should become last entry in last SIDR with key ‘Request’
- New inserts must first find the end of target SIDR chain potentially requiring many read IOs

Duplicate Chains, Picture.

HP WORLD 2003
Solutions and Technology Conference & Expo



Prologue

Key 2
Alternate

SIDR Index (=root)
"Done","*"

No more Index Entry !

Active[1,2],
Done[1,2,3]

Done[6,7,8,9,
10…99]

Done[100,
101,…190]

Done[…]
Request[1,2,3]

SIDR
Buckets

RRV's pointing to User Data Record.

Sidr Data (100/bucket)

Key	SIDRs	Buckets
Active	2	1
Done	200+	4
Request	3	1

Duplicate Chains, Detection methods.

- Problems often seen on SHORT Alternate Keys
  - Equally possible on primary keys, but those are more often then not made unique or almost unique.
- High READ IO rate for an application that is supposed to be mostly writing.
- Very high XFC or IO-controller Cache Hit Rate.
- `ANALYZE/RMS/FDL` <your-file.idx>
- RMS_TUNE_CHECK tool on VMS Freeware
  - Used to be called ‘sidr’.
  - Suggestion: check .txt help file, and try –m argument

Duplicate Chains, Detection example 1 of 2.

- `ANALYZE/RMS/FDL/OUT=SYS$OUTPUT X.IDX`

```
ANALYSIS_OF_KEY 2
  :
  DATA_SPACE_OCCUPIED      1968
  DUPLICATES_PER_SIDR      969
  LEVEL1_RECORD_COUNT      9
```

  - Bucket size was 16
  - Therefore… 1968/16 = 123 SIDR buckets.
  - Just 9 index entries: 112 buckets with duplicates!
  - And… maximum SIDR size = 1168 entries

    (16*512 – 15 – compressed-key-size) / 7

Duplicate Chains, Detection example 2 of 2.

- `RMS_TUNE_CHECK Y.IDX`

```
   :
   Duplicate count, Buckets, Key value
   ----------------------------------
        1759748         4045     000000000
             46            1     292164044
             27            1     211941745
             25            1     211147595
```

  - Every new record inserted with key value 0000000000 will require 4000+ read IOs and 1 (or more) Write.
  - Most new records will have that key value… how else did those 1.7 millions duplicates get there?

Duplicate Chains, Solutions 1 of 3.

- DROP the Key! Do you really use it?
  - Consider SORTING primary data as alternative.
  - What is the business value of a query 'find the first of the 2 million records without an appointment'. Now. And the next, and the next…
- Apply NULL KEY VALUE
  - Set 'the million dollar bit' : `xab$v_nul = 1`
  - Set 'null value' byte `xab$b_nul = 32` (for 'zero')
  - Sample FDL command to set 'space' as null key value:

    ```
    NULL_KEY      yes
    NULL_VALUE    32
    ```
  - NO index entry made if each byte of the key for the records added equals the 'null value' byte
  - NO application code change needed. Just re-convert!
    - Restricted use. May need to define application data value.

Duplicate Chains, Solutions 2 of 3.

- Increase the Bucket Size (for the alternate key)
  - IO Size is not such an important factor in the Cost to do an IO. It's the number of IOs that matter.
  - RMS Maximum bucket size is 63
  - If the current bucket size is 12 or less, then duplicate key read IO count can be divided by 5 or more.
  - Edit FDL file, change bucket size in the AREA for the key with a problem, and re-convert.

Duplicate Chains, Solutions 3 of 3.

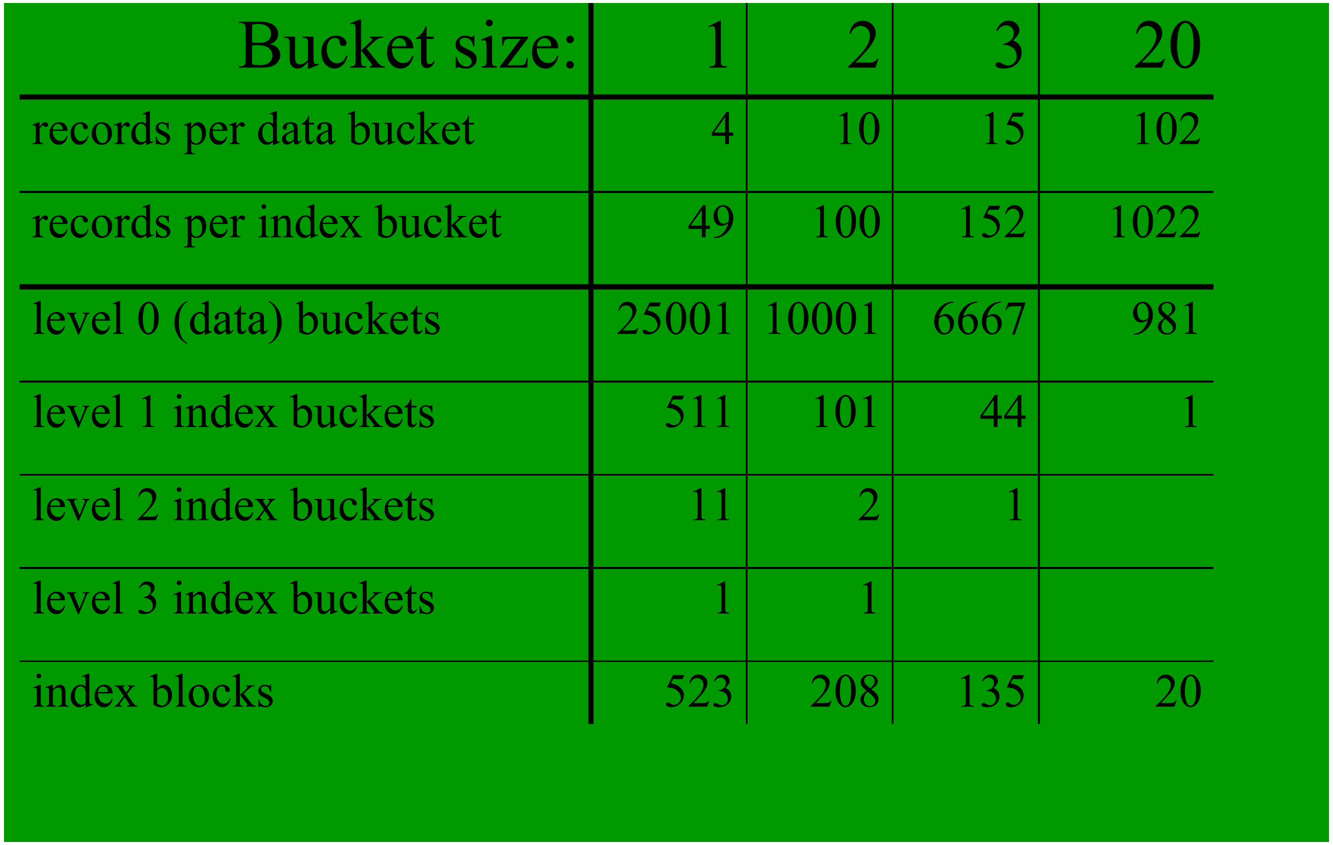
- Add (ordered) key segment to ‘de-duplicate’
  - Goal is NOT to make each key unique,
  - Goal is just to avoid buckets full of duplicates.
  - A few (hunderds!) duplicates is just fine (even efficient!)
  - Physical data need not be changed, just the key definition to add a segment `(XAB$W_POSx, XAB$B_SIZx)`
  - The new segment *will* change the sorting order. Ok?
  - Sample: change STATE to STATE + ZIP(9)
  - Sample: add low bytes from primary as segment to alternate key to divide average dup count by 100?

Index Depth

- Each level to traverse may need an IO
- Each level requires a (temporary) LOCK
- Locking ‘amplified’ with global buffers
- Bucket locking is cluster wide
- Binary search minimizes lock duration
- ‘Flat’ files are often best but may cause too much contentions. Compensate extra levels through more buffers

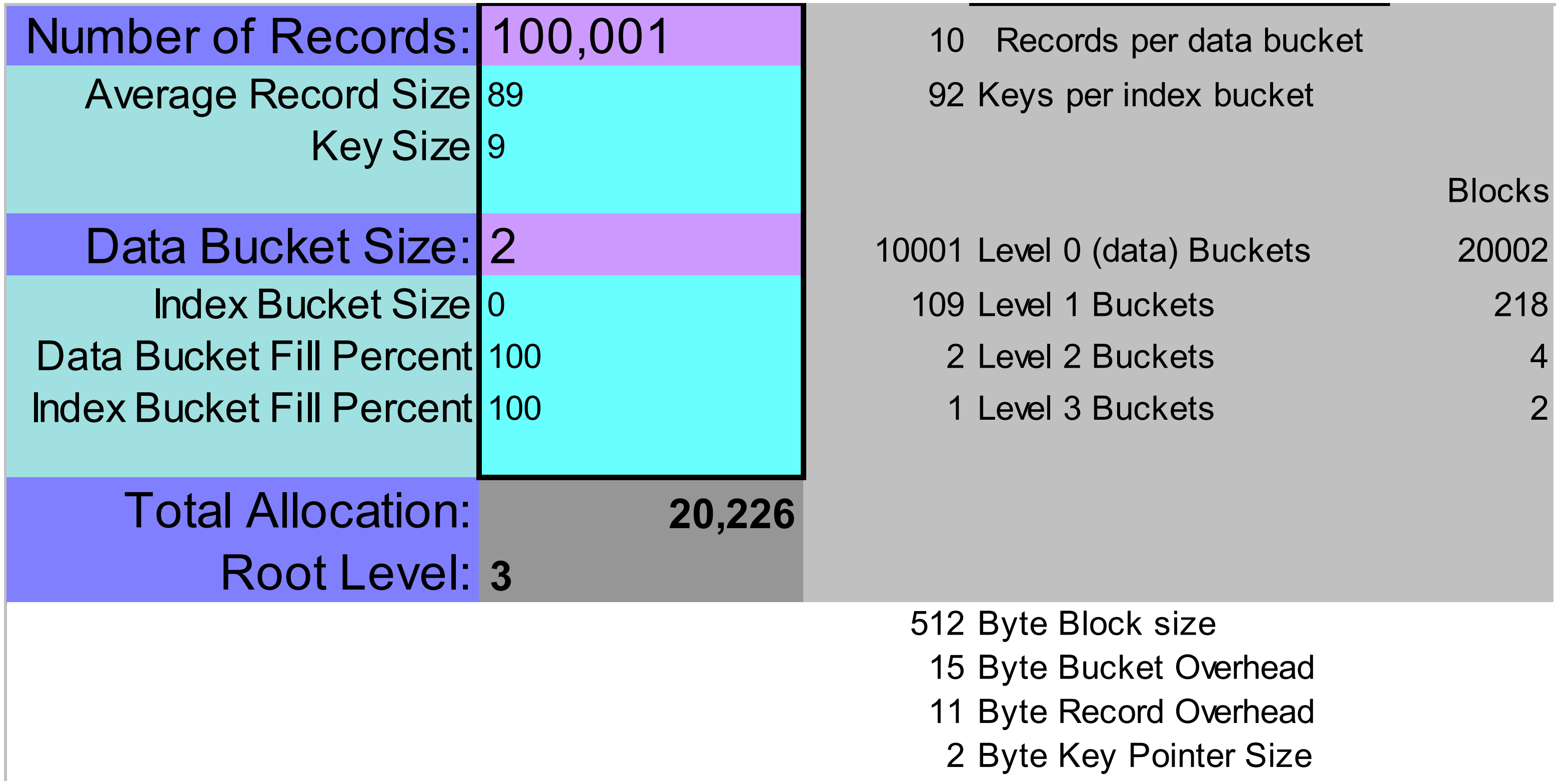
Index Depth Cheat Sheet Sample

100,001 records; recordsize =100; keysize =10; overhead included



Bucket size:	1	2	3	20
records per data bucket	4	10	15	102
records per index bucket	49	100	152	1022
level 0 (data) buckets	25001	10001	6667	981
level 1 index buckets	511	101	44	1
level 2 index buckets	11	2	1	
level 3 index buckets	1	1		
index blocks	523	208	135	20

Index Depth Excel spread sheet



Number of Records:	100,001	10	Records per data bucket	
Average Record Size	89	92	Keys per index bucket	
Key Size	9			
				Blocks
Data Bucket Size:	2	10001	Level 0 (data) Buckets	20002
Index Bucket Size	0	109	Level 1 Buckets	218
Data Bucket Fill Percent	100	2	Level 2 Buckets	4
Index Bucket Fill Percent	100	1	Level 3 Buckets	2
Total Allocation:	20,226			
Root Level:	3			
		512	Byte Block size	
		15	Byte Bucket Overhead	
		11	Byte Record Overhead	
		2	Byte Key Pointer Size	

- ***Click on spreadsheet to activate***

Adequate Extends

- High price if done wrong
  - Not acceptable to run out of disk space while in production
  - Frequent file system (XQP) requests each requiring several IOs (worse with High Water Marking)
  - Fragmentation
    - One bucket requiring multiple clusters = SPLIT IO
    - Unwarranted disk head movements
- Very easy to do right
  - Rely primarily on adequate allocation
  - If file needs to grow, allow significant growth
    - Gotcha: Maximum extend is 65536

Compression

- DATA and KEY compression is goodness
- Save SPACE and save CPU TIME
  - Fewer data buckets => fewer IO
  - Fewer index keys => fewer index levels
  - More effective caches
  - Less memory to walk
- Remember to verify effectiveness
  - FDL has no smarts to detect negative compression
- Index compression often discouraged as it prohibits binary searches in index buckets

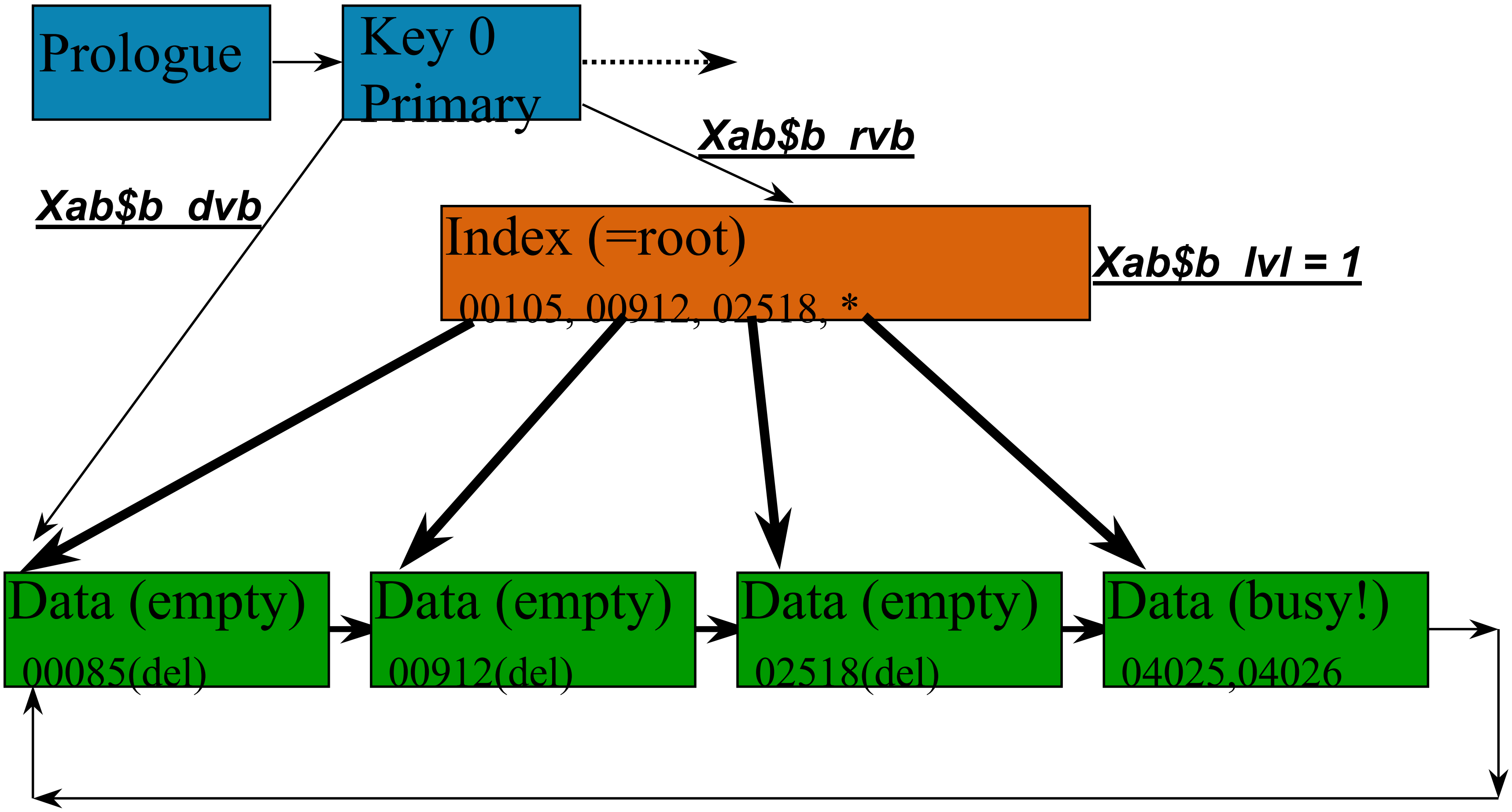
Deleted key ranges

- Deleted records are purged from the data bucket.
  - Space can be re-used for records with similar keys.
  - deleted ‘ID’ remains gone, allowing alternate key, and RFA, access to conclude the record was deleted.
  - With ‘Fast Delete’ a future alternate key access will remove the associated alternate key.
  - Exception: The last key is never deleted, but left
- The Primary key index is never removed.
  - Buckets remain ‘reserved’ for key range from index.
  - CONVERT/RECLAIM designed to clean up (off-line!)
- PROBLEM: ‘GET First Record’ may travers empty buckets

Deleted key ranges (Continued)

- Example: “work queue” file with date & time key.
  - If work needs to be done, a record is inserted.
  - Worker process takes action and deletes record.
  - At end of day, no records are left.
- If worker falls behind, new records fill up bucket and ‘spill’ into next bucket. The first bucket wil never be re-used for data, because keys increase.
- **Workaround / Solution:**
  - Maximum bucket size (63) may cache queue longer, adding fewer buckets. (watch out for RMS AIJ)
  - Remember last record processed, used KGE.
    - Does NOT have to be exact, just update when crossing bucket boundary (peek into RFA? Update every 100 records or seconds?)

Deleted key ranges (picture)



Prologue

Key 0 Primary

Xab$b_rvb

Xab$b_dvb

Index (=root)

00105, 00912, 02518, *

Xab$b_lvl = 1

Data (empty)

00085(del)

Data (empty)

00912(del)

Data (empty)

02518(del)

Data (busy!)

04025,04026

What about IO

- Eliminate it: Caches & Application design
- A files place in space and time is critical
  - Just because it fits, does not make it the right place.
- Spread it out
  - Multiple disks per application
  - Hardware or software striping
  - Bound volume sets (area placement)

    Yes, you can bind a Solid State disk with a real disk
  - Shadow sets (notably for read intensive)

What about IO

- Speed it up
  - Solid state disks (DECram, EZnn)
  - Faster disks (10,000 rpm now available)
  - Shorter seeks by reducing area on disk
  - Faster area on zoned disks

    RZ29 has 16 zones from 67 to 135 sectors per track.

    Some 50% of the data lives in 30% of the seek range
  - Track read ahead caching
  - Writeback caching (notably with battery backup)

Bucket Splits

- Necessary evil… CONVERT regularly
  - RMS keeps records in primary key order
  - RMS maintains single pointer from original record file address (RFA) to actual location after split
- Cause by random inserts or updates changing the effective record size
- Try to add records in primary key order
- Fill factor used to avoid or ease splits
  - leave room for a few records per bucket
  - Localized multiple inserts? Split will make room.

Cluster size

- Fine-tuning. Exercise in trading speed for space.
- More important with underlying STRIPING
  - EDIT/FDL makes buckets too large for large cluster.
  - Make CHUNK, CLUSTER and BUCKET SIZE all have nic factors, notably for small chunks (swxcr).

    Chunk=16, Cluster=64 Bucket=8, NOT bucket = 17
  - Each AREA = XQP Extend. Will be cluster aligned
  - Buckets will not cross into extends

    By default the data buckets in the first AREA 0 extend are unaligned following the file PROLOGUE. Can be fixed by using area 0 for top index, area 1 for data.

HP WORLD 2003

Solutions and Technology Conference & Expo

Interex, Encompass and HP bring you a powerful new HP World.

interex
shared knowledge • shared power

encompass
AN HP USER GROUP

hp
invent

Assorted Nice to Knows

- Convert starts duplicate chains in their own buckets, mostly goodness but…
- Use normal editor for FDL files
- EDIT/FDL/NOINTER takes TWO inputs:
  - File design: Defines keys and such. Do not touch.
  - Statistics: really only uses 3 inputs!
    - Record count: Set to anticipated value
    - average record size: Keep from old stats
    - cluster size: Set to small ‘nice’ value: 12? 6?

Assorted Nice to Knows

- Use ANALYZE/SYSTEM to peak at global buffers and hit & miss counters if no file stats.
  - SDA> SHOW PROC/RMS=(GBH,GBDSUM)

- Tune system files
  - sysuaf, rightslist, mail_profile, mail.mai files,...

Assorted Nice to Knows

- Use ‘idle’ program to maintain global buffers when files are not always open. Keep them warm
  - Drive with file of files
  - Consider touching ‘desirable’ records on startup
  - Optionally add check index levels/allocations
- Use SHR=NIL to avoid locking
- $GET for first record walks index. Cache its RFA in application?

Assorted Nice to Knows

- Records are stored in order by Primary key.
  - Assume record has name and number as keys
  - If frequent sequential or ‘generic’ search by name is needed and only random access by number then, contrary to popular believe, name should be primary
- If significant portion of the file is read by alternate key order, then it will often be more efficient to walk entire file by primary
  - With random distribution, each alternate key need an IO to find right data bucket. By contrast, each primary bucket read will return several records

HP WORLD 2003

Solutions and Technology Conference & Expo

Interex, Encompass and HP bring you a powerful new HP World.

interex
shared knowledge • shared power

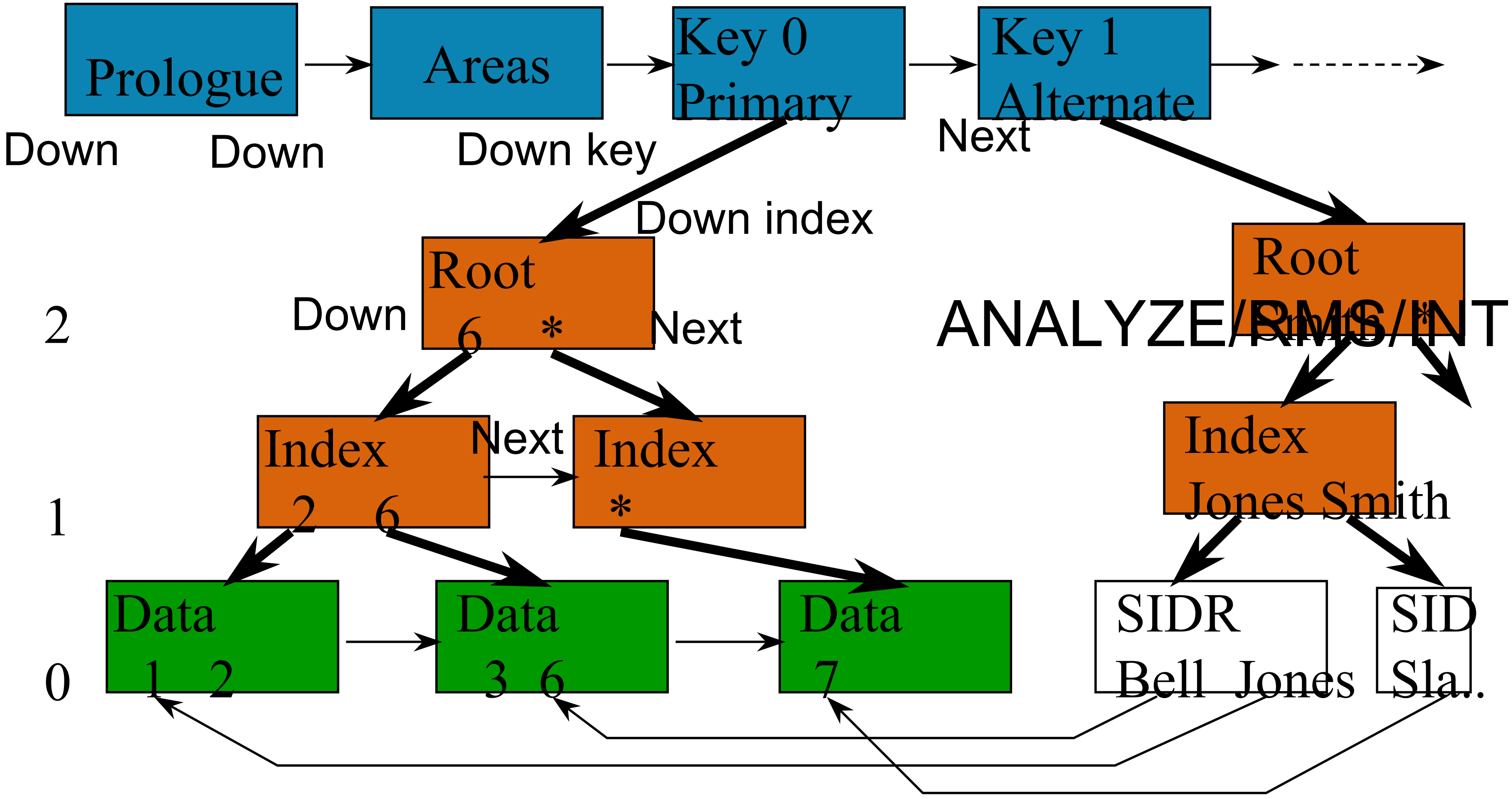
encompass
AN HP USER GROUP

hp
invent

Patching up broken files

- The remainder of this presentation was the main contents presented at CETS200 in session 705

Anatomy of an Indexed File



ANALYZE/RMS/INT

Bucket header
$libr/extr=$BKTdef sys$library:lib.mlb

Offset	Type	Description
0, x-1	Char	Bucket Check Byte
1	Char	Index level or Area indicator
2	Word	VBN Address Sample (low 16 bits)
4	Word	Free Space Offset (end of data)
6	Word	Next Record Id (for prologue 3)
8	Long	Next Bucket VBN
12	Byte	Bucket Level (0 = data)
13	Byte	Flags (LAST, ROOT, PointerSize)

Record header
(Variable, Compression)
$libr/extr=$IRCdef sys$library:lib.mlb

Offset	Type	Description
0	Byte	Control Byte (2=Valid, DEL,RRV)
1	Word	Record ID
3	Word	RRV ID (original ID)
5	Long	RRV VBN (original VBN)
9	Word	Record Length
11	Byte	Key length as stored here
12	Byte	Count of front bytes (previous key)
13	Chars	KEY data (variable count)
Data	Chars	RECORD data (variable count)

Excuses for corruption

- Hardware failure: Disk, controller, cable.
- Partial IO due to Power-failure
- stop/id (amplified with deferred write usage)
- Software failure
  - RMS: NO known problems since 6.0
  - Experimental defrag tool?
  - Privileged code (write logical block)
  - Stupid code: $READ/$WRITE indexed files.
  - Experimental data caching tool?

Basic PATCH strategy

- Use tools to locate problem zone
- Goal is to be able to CONVERT the file
- Concentrate on DATA buckets only
- Concentrate on bringing bulk of file back
    - Trade-off between TIME and DATA
- Worry about touching up remainder later.
    - Use BACKUPs and application reports to reconstruct any lost records if needed.

Basic PATCH Analysis tools

- ANALYZE/RMS/INTERACTIVE
  - Drill down structure as per first slide.
- DUMP
  - /BLOCK=(START:x, COUNT:y)
  - /RECORD=(START:x, COUNT:y)
- SEARCH, DIFF, EDT
  - Quick tests for linear read.
- DCL: READ/KEY
  - Quick test for random access read

Basic PATCHing tools

- COPY / BACKUP : Work on test file first!
- PATCH… on VAX in cluster.
  - ‘vested’ Alpha version is floating around
- DCL on file in SEQUENTIAL-512 bytes mode
- Reading (CONVERT) by alternate key
- ZAP
- COPY_BLOCK
- ‘binary’ file editor ‘rms-edt?’
- CONVERT… when all is well again.

DCL as PATCHing tool

- Flip file from indexed to sequential (and back later) in order to read /write a block at a time.
  - `set file/att=(org=seq,rfm=fix,mrs=512,lrl=512)`
  - `set file/att=(org=idx,rfm=var,mrs=x,lrl=y)`
- DCL File and symbol manipulations:
  - `open/read/write file filename.dat`
  - `key[0,32]=vbn_number          !Binary key value`
  - `read/key=&key file record`

    Use & to postpone symbol substitution, quotes are a problem
  - `record[x*8,y*8]=z`

    replace y bytes at offset x by value z
  - `write/update/symbol file record`
  - `close file`

ZAP as PATCHing tool

- Simple Macro tool to read, update, format, write buckets.
- Take your time to study outputs. ‘dense’, but all you need is there.
- Uses DBG as GUI
  - Define dbg$decw$display “ “
  - bucket buffer pointed to by ‘buf’ and R2

```
DBG> examin/octaword @r2
DBG> deposit/byte @r2=0
DBG> go   … back to prompt, format and write.
```

Click on icon to get source

COPY_BLOCK as PATCHing tool

- Simple C tool to move a block/bucket between files.
- May need to update source to your liking (input VBN).
- Can be used to ‘clone’ a good (but old) bucket from a backup over a broken file bucket.
- Adjust checkbytes… if you are so lucky

Click on icon to get source

Basic PATCH strategies

- Adjust check byte
  - Loose nothing?!
- Patch ‘around’ broken bucket(s)
  - Loose a bucket of records
- Adjust next free byte value
  - Loose a end portion of bucket
- Construct deleted record over bad blocks
  - Loose a middle portion of bucket
- reconstruct bucket header
  - Loose nothing?

‘easy’

↓

‘hard’

Basic Check byte correction

If somehow the beginning of an updated bucket was written, but not the end, you may only have to make the check-bytes match

- Needed for most other steps.
- Really easy to do.
- Easiest to adjust byte-0 to match last one

  (even though last one is really the bad one)
- Done! ‘if you would only be so lucky’

Patch ‘around’ broken bucket(s)

If the bucket, or a series of buckets really ‘looks like a mess’ (DUMP), then maybe just give up on that data.

- Fairly easy to do: Quick & dirty
- Set next-vbn in bucket before bad zone to point to first valid bucket after bad zone.
- **hope** that buckets are adjacent.
- Verify prior and next bucket VBN with pointers from the index bucket above it.

Adjust next free byte value

The beginning of an updated bucket was written, but not the end. The free byte points into the end. The new data abruptly flows into old data blocks.

- Fairly easy patch.
- ‘Eye-ball’ dump to find good/back boundary.
- ANAL/INT/RMS
  - POSITION/BUCKET broken-vbn
  - NEXT 9999: run into broken record
  - BACK : find start of last good record
  - Calculate end of last good record.

Construct deleted record over bad blocks

- Experts only. A lot of work.
- Easy to get RMS to loop if done wrong
- Fake all but FLAG (deleted) and SIZE
- Fake KEY with data key compression (Tricky!)
- Maximum data recovery chance!

reconstruct bucket header

If somehow the beginning of a bucket was overwritten (zeroed out? Text file?)

- Al lot of the header data is ‘redundant’
  - first check from last
  - Since we are only interested in the data level make level=0, and area=0
  - vbn sample from vbn address
  - flags = 0 (not last bucket is it?)
  - next bucket from index (or adjacent value)
- Construct deleted record to span into good zone.