LSR Tuning Today

- **Eugene S Hudders**
- **C\TREK Corporation**
- **PO Box 560100**
- **Montverde FL 34756-0100**
- **ehudders@ctrek.com**
- **(787) 397-4150**

Disclaimers

- **Remember YMMV**
- **Remember the political factor**
  - **“If it ‘ain’t’ broke, don’t fix it!”**
- **The following products are trademarks of IBM Corporation, Armonk, NY**
  - **CICS TS**
  - **COBOL LE**
  - **z/OS**
  - **VSAM**
  - **DB2**

Agenda

Introduction

- CICS TS VSAM Support
- Physical I/O
- CICS TS FC API Costs
- KSDS Structure and Terms
- The Robin Hood Theory

Traditional LSR Tuning

- Dynamic/Static Definition
- LSR Pool Measurement
- Buffer Allocation
- String/KL Allocation
- Buffer Monopolization
- Hiperspace Buffers

Agenda

Other LSR Tuning

- Fragmentation
- Buffer Alignment
- # of LSR Pools
- Non-Candidates for LSR

Hidden Tuning Areas

- VSAM CA Size
- CI/CA Splits
- Extents

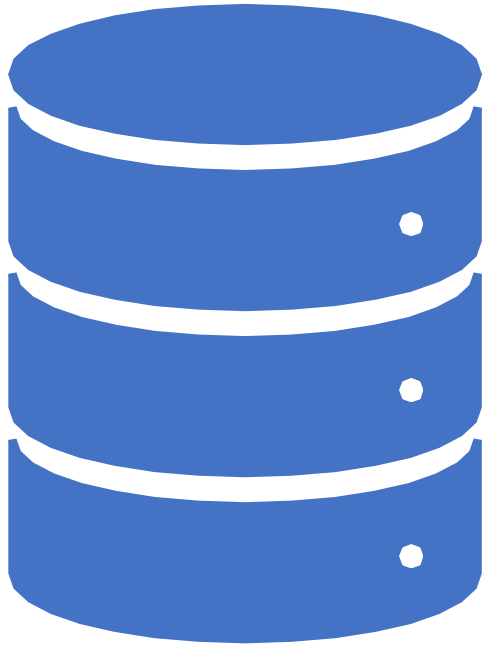
Closing

C\TREK Corporation 2019

INTRODUCTION

CICS TS VSAM SUPPORT

- CICS uses three techniques to handle VSAM files within CICS TS:
  - Non-Shared Resources (NSR)
  - Local Shared Resources (LSR)
  - Record Level Sharing (RLS)
- In recent years, new VSAM features announced for CICS have been LSR/RLS oriented
- The major difference between the three techniques lies in the "ownership" of the resources
  - NSR → resources are used exclusively by the file
  - LSR → resources are shared between participating files
  - RLS → resources are in a different address space (SMSVSAM) and requires a Coupling Facility (CF)
- CA splits tie up the main task TCB for NSR files
  - Consider the use of the CO TCB (Multi-processor)
  - Consider moving the file to LSR



C\TREK Corporation 2019

LSR Advantages

More efficient use of virtual storage versus NSR as resources are shared

Better look-aside hit ratio because the Sequence Set Index (SSI) are maintained in the buffer pool

Tends to be more self-tuning because buffers are allocated on a Least Recently Used (LRU) algorithm keeping the information for the more active files in buffers at the expense of lower activity files

Only one copy of a CI is allowed (better read integrity)

Can allocate up to 255 pools to segregate file access

- Provides more strings and buffers (VS is the limit)
- Best used by VSAM threadsafe (parallel access versus single thread)

Supports Transaction Isolation

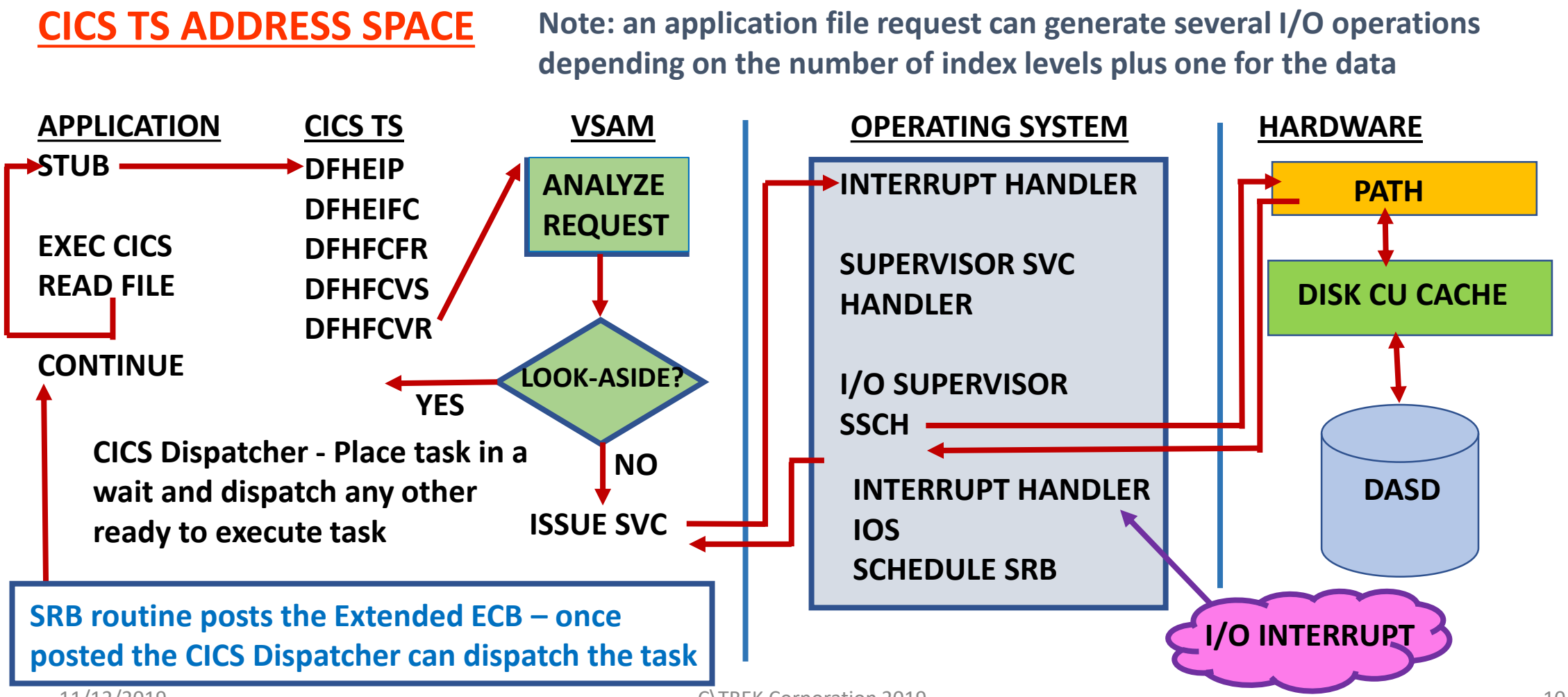
Physical I/O

- Physical I/O costs CPU cycles
  - Instruction Path
    - Application
    - CICS
    - VSAM
    - MVS
  - State Changes
- “Best I/O is the one that is not done”
- The key is to reduce physical I/O operations
  - Measured by look-aside hit ratio
  - CPU requirements for look-aside are much lower
- Misconception:
  - “Disk cache hit doesn’t generate physical I/O overhead”

Simplified Physical I/O Flow

- **Physical I/O generates CPU Utilization**
  - **CICS to**
  - **VSAM to**
  - **SVC Handler to**
  - **IOS**
  - **Start the I/O (SSCH) and eventually back to**
  - **VSAM and back to CICS to have the task wait for I/O completion**
  - **CICS continues to dispatch work or MVS wait (ICV and multiple wait)**
  - **----------I/O interrupt occurs----------**
  - **MVS handles and processes the I/O Interrupt**
  - **Schedules an SRB to the CICS address space (AS)**
  - **Dispatch the SRB when the AS is selected to run**
  - **CICS Dispatcher selects the waiting task to run (based on being ready and priority)**

The Very Big I/O Picture



- **For read operations, the VSAM I/O cost is not included because the need to access DASD depends on the workload. For the read operation to complete, both the index and data must be accessed. If the index or data are not in a buffer, an I/O operation is required for each level of index and one for the data.**
- **The relative number of instructions, in 1K instruction counts, for the I/O for each file type is as follows:**
  - **9.5 for a key-sequenced data set (KSDS)**
  - **9.5 for an entry-sequenced data set (ESDS)**
  - **8.2 for a relative record data set (RRDS)**
- **Source CICSTS55 Performance Guide Pages 176-178**

CICS TS File Control API Costs

CICS TS File Control API Costs

READ

KSDS	ESDS	RRDS	Data Table (CMT)
3.0	2.4	2.2	First: 1.5 Subsequent: 1.1

READ for UPDATE (Non-Recoverable)

KSDS	ESDS	RRDS
3.1	2.3	2.2

READ for UPDATE (Recoverable)

KSDS	ESDS	RRDS
5.5	4.3	4.2

REWRITE (Non-Recoverable)—has a VSAM I/O associated with the REWRITE

KSDS	ESDS	RRDS
10.2	10.1	10.1

Source CICSTS55 Performance Guide Pages 176-178

CICS TS File Control API Costs

REWRITE

- REWRITE (Recoverable)
  - KSDS ESDS RRDS
  - 10.4 10.3 10.3

WRITE

- WRITE (Non-Recoverable)—has a VSAM I/O associated with the WRITE (Does not include cost of CI/CA Split—additional Index I/O if split occurs)
  - KSDS ESDS RRDS
  - 12.9 11.1 10.9

WRITE

- WRITE (Recoverable)—has a VSAM I/O associated with the WRITE (Does not include cost of CI/CA Split—additional Index I/O if split occurs)
  - KSDS ESDS RRDS
  - 14.9 13.1 12.9

Source

- Source CICSTS55 Performance Guide Pages 176 to 178

CICS TS File Control API Costs

DELETE (Non-Recoverable)

KSDS	RRDS
12.5	11.5

DELETE (Recoverable)

KSDS	RRDS
14.5	13.5

BROWSING

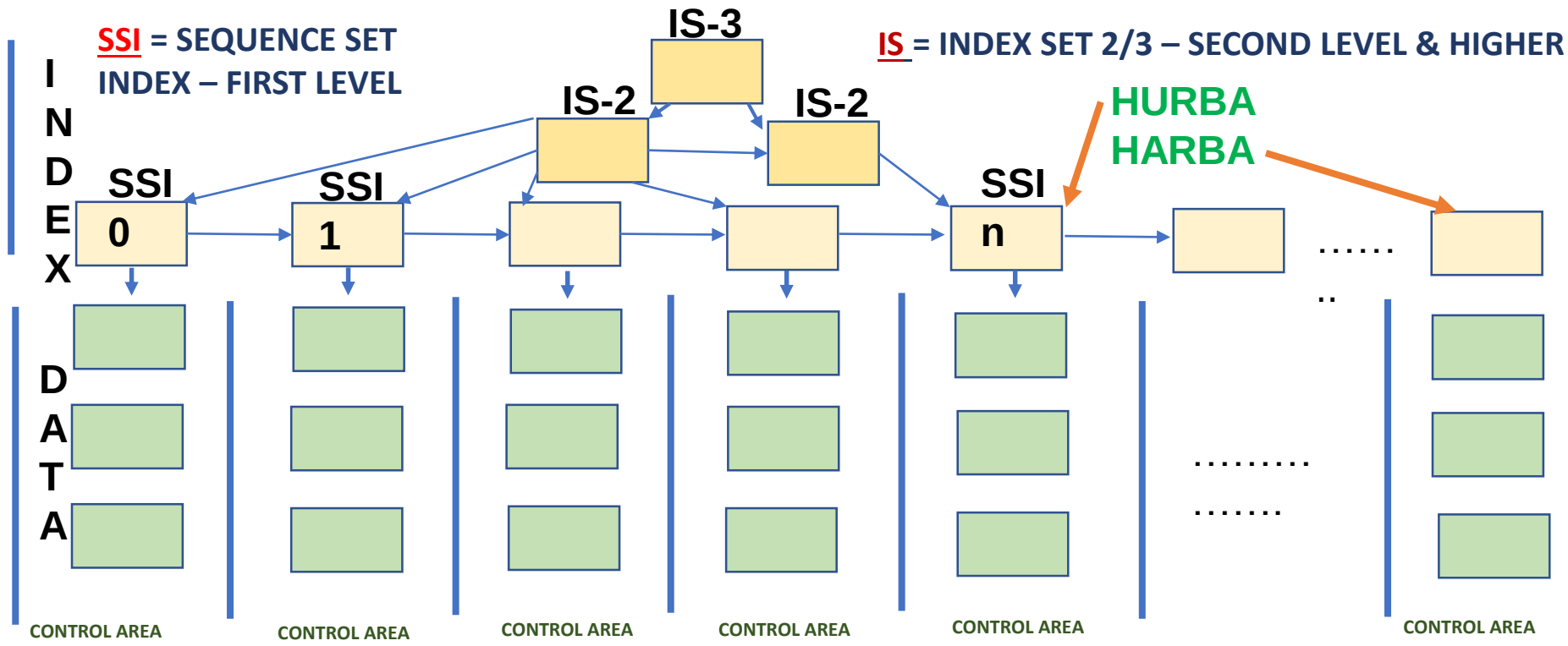
STARTBR	READNEXT	READPREV	RESETBR	ENDBR
3.1	1.5	1.6	2.6	1.4

UNLOCK

- **Path is 0.7**

Source CICSTS55 Performance Guide Pages 176 to 178

KSDS Structure and Terms



The Robin Hood Theory

- **Tuning LSR files is simply the opposite of what Robin Hood did in Sherwood Forest**
  - **Robin Hood stole from the rich to give to the poor**
- **In LSR you will steal from the poor to give to the rich!!!**
  - **Poor → low to medium activity files**
  - **Rich → most active files**
- **So, the major contribution made by low/medium activity files in LSR is to provide their resources so that higher activity files can use them**
  - **Unfortunately, this is the cruel reality**

Traditional LSR Tuning

Dynamic Versus Static Pool Definition

DYNAMIC DEFINITION

- Advantages
  - Quick implementation and installation
  - Reduces system programmer intervention
    - No need to determine VSAM CISZ vs. buffer size
    - No need to determine the maximum key length
    - No need to determine the number of strings needed

STATIC DEFINITION

- Advantages
  - Separate buffer pools for data and index components
    - No CISZ contention between data and index
  - Can optimize for buffers that have higher activity
  - Can optimize string and maximum key size
  - Faster CICS initialization
  - Can allocate Hiperspace buffers (if applicable)

Dynamic Versus Static Pool Definition

DYNAMIC DEFINITION

- Disadvantages
  - Possible contention between data and index buffers (combined pool)
    - Can hide bad data/index performers
  - Number of strings and buffers allocated are based on a % not on file activity
    - Can result in string over-allocation
  - Slow initialization
  - Cannot allocate Hiperspace buffers (if applicable)

STATIC DEFINITION

- Disadvantages
  - Requires system programming intervention
    - Need to determine # of strings, # and buffer sizes and maximum key length
    - Exposed to error
    - Requires planning
    - Possible solution
      - 255 strings
      - Max KL 255
      - Define a value for all buffer sizes

LSR Pool Definition

- **Recommendation**
  - **Define LSR pools explicitly**
  - **Initially bring the system up dynamically to get an idea of the buffers and strings required and the maximum key length (test environment)**
    - **Using the buffers defined, use the definition to define the data component**
    - **Initially, use the same definition for the index component**
    - **Run transactions and determine actual buffers used**
    - **Using this information adjust buffers required for the data and index components**
    - **Ensure a definition of a safety valve buffer of 32K, if none defined**
    - **Use performance monitor or CICS statistics (STAT or EOD)**
  - **Once in production, monitor and adjust as required**

LSR Pool Measurement

LSR pool effectiveness is measured on look-aside hit ratios

Generally accepted ratios are:

- **Data component → 80%+**
- **Index Component → 95%+**
- **Combined → 93%**

Some installations have higher objectives

- **Important thing is to establish what is acceptable for your installation**

Look-aside hit ratio is usually improved by adjusting buffers assigned

Improve the index hit ratio first because you access the index component more than the data component in direct read operations

Total number of index CIs is lower than for the data component

Index CISZ are usually smaller in size than the data component CIs

- **So, real/virtual storage required to improve index hit ratio is less**

LSR Pool Measurement

- **Data buffer tuning is highly dependent on the file's access patterns**
  - **Good look-aside hit ratios for the data component usually requires a substantial amount of storage to obtain an 80%+ hit ratio**
  - **The major cause is that the data component for all the files is usually very large (vs. the index component)**
  - **Good look-aside hit ratios usually result in files with:**
    - **Sequential activity**
    - **Read for Update/Rewrite/Delete activity**
    - **Concentrated read activity**
- **LSR buffer look-aside % can be misleading**
  - **The % specified does not mean that every file is getting that % (remember Robin Hood)**
  - **The look-aside % is the average of all the files using that buffer**

Buffer Allocation

- Some installations prefer to define “x” number of buffers to all 11 LSR buffer sizes whether there is a file that can use it or not
  - This results in virtual storage allocation that will not be used “day in day out, 365 days a year, every year”
    - For example, suppose you have no file that can use a 16K buffer, but you allocated 40 buffers (640K) in case one day a 16K file appears
      - The allocation is a “magic number”, that is, you really don’t know or predict the activity that this new file is going to generate
    - Wouldn’t it be better if you took that 640K and converted it to thirty-two 20K buffers that will be used every day?
      - If one day the file does appear, at least you have thirty-two buffers you allocated to use for the file
- It is important to note that buffer allocation to files in LSR are not controlled by the number of buffers assigned to the file
  - A file can have more buffers assigned than the number in the FCTE definition

String Allocation

- Each pool can have up to 255 strings
- Usually tuned when you get a wait on strings condition
- There are 2 types of wait on strings for LSR
  - Wait on string related to the number of strings allocated to the file
  - Wait on string related to the number of strings allocated to the pool
- String allocations are controlled by CICS
- Objective should be to have the LSR string assignment somewhere between 50% to 60% of the peak string usage

Maximum Key Size

- The maximum key size is 255 bytes
- As LSR control blocks are shared, the maximum key length must be defined (PLH control block)
- If the maximum key size specified for the LSR pool is too small, the file will not open
- To avoid this situation, many installations define the maximum key size as 255
- The actual virtual storage cost depends on the number of strings

VS Costs

Current LSR pool has 60 strings and a maximum KL of 100

Performed a benchmark to determine the cost of specifying the maximum key length (255) and maximum strings (255)

- Step 1 was to recycle and use the current settings 60/100
- Step 2 was to recycle and use 255/255 for strings and maximum KL
- Difference was 195 strings and 155 bytes for the KL

Results	Step 1 (MB)	Step 2 (MB)	Difference (KB)
• Initialization	856,568	856,216	352K
• Open all files	855,728	855,140	588K

Estimated cost

- 195 strings * 900 (PLH) → 175,500 bytes
- 195 strings * 155 (increased KL) → 30,225 bytes
- Total 205,725 bytes

Buffer Monopolization

An area that must be monitored is the possibility of a file monopolizing a buffer in a pool

- **The problem is that CICS does not provide information regarding the number of LSR buffers being used by a file**
- **The statistics provided indicate the activity, but this does not translate into number of buffers**
  - **A file could perform 100K accesses to the file, but this does not translate into number of buffers as it could be that the access is to one or a few buffers**

Several options are there to resolve this situation

- **Move the file to a separate LSR pool**
- **Increase the number of buffers to reach the file's point of Diminishing Return**
  - **Once you reach this point, other files will have access to buffers**

What Happens If You need to Add a File and the Buffers or Max KL Are Not Defined?

- **If the required buffer and/or maximum KL is not defined:**
  - **Use the next bigger buffer size available and open the file**
  - **If no bigger buffer available, send message and file is not opened**
  - **Possible solutions**
    - **Always define 32K buffers (if not already defined) for index and data components to protect against this happening**
      - **Major objection is having VS that is not used**
    - **Change the LSR pool number to one not in use and open the file into a pool that is dynamically created**
      - **Increase number of strings (255) to get more buffers allocated**
- **If the file's KL exceeds the maximum KL specified**
  - **Performed a benchmark to determine the cost of specifying the maximum key length (255) and maximum strings (255)**
- **If these options are not acceptable, prepare a good explanation for your manager as to why you are going to have to recycle the LSR pool**

Hiperspace Buffers

- Hiperspace buffers were designed to use Expanded Storage (ES)
  - ES worked like a very fast synchronous paging device
    - ES was less expensive than real storage
    - ES was 4K addressable (not byte addressable like real storage)
- z/Architecture does not support ES
  - In order to maintain this functionality, ES is simulated using real storage by z/OS
- CICS supports Hiperspace buffers in LSR
  - However, you are using real storage to simulate ES
  - Moving real to real overhead
  - Better to allocate the equivalent Hiperspace buffers into the regular LSR buffers
- You may want to use Hiperspace buffers under the following conditions if enough RS exists
  - You need more than 32K buffers of a specific size
  - You are running low on the region VS availability

Other LSR Tuning Areas

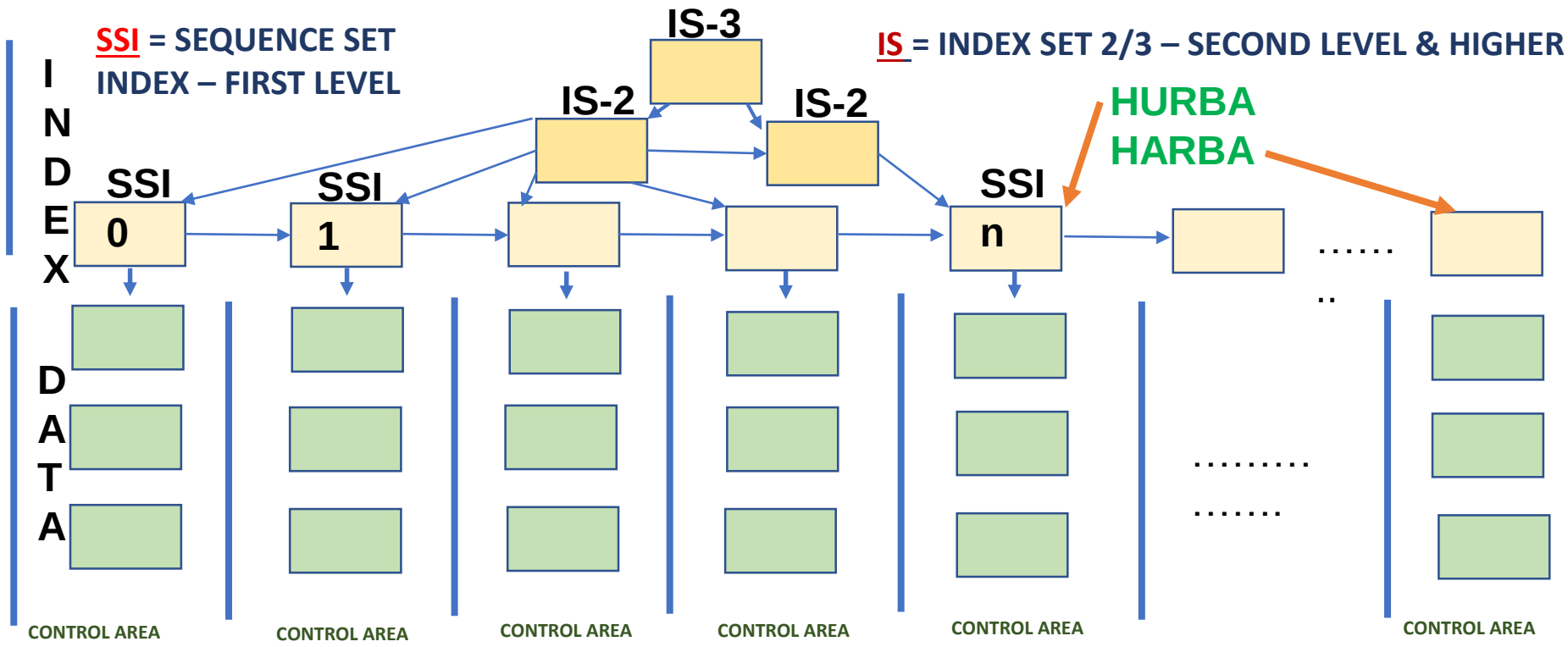
Fragmentation

- **Fragmentation represents the lost space due to the difference in the CISZ and the LSR buffer assigned to handle the CI**
  - **The major cause of fragmentation is that VSAM has 28 different CISZ available while CICS LSR only supports 11 buffer sizes**
  - **Some data component fragmentation may be acceptable such as using an 18K CISZ (non-VSAM/E) to obtain the best disk utilization. In this case, you would use a 20K buffer with a 2K or 10% cost of virtual/real storage fragmentation**
  - **Other fragmentation is not acceptable such as using a 16K buffer to cover a 10K CISZ because the user did not define a 12K buffer**
  - **Adjusting the CISZ to a data component buffer size may have some advantages. For example, increasing a CISZ from 5.5K to use an 8K buffer**
    - **Takes advantage of 2.5K (31%) lost virtual/real storage**
    - **Can add or adjust free space without increasing the amount of VS/RS used by CICS LSR**

Fragmentation

- There may be some advantages to increasing the index component size to match the LSR buffer size
  - For example, increasing the CISZ from 1.5K to 2K (or 2.5K, 3K and 3.5K to 4K)
    - For large files this may reduce the number of IS index records in the file
      - Less IS records, less buffers are required in LSR to support the file
    - May provide a cushion for potential key compression problems that may occur within the SSI of a file
  - Be especially wary of using a large data CISZ (e.g., 26K) for large files that result in VSAM assigning a small index CISZ (e.g., .5K)
    - This may result in more IS records requiring more LSR buffers to support the file
- Recommendation: Ensure that you do not have the bad fragmentation in your LSR pool

KSDS Structure and Terms



Control Interval Size (CISZ) vs. Buffer Size

VSAM	LSR POOL	LSR HIPERCACHE	VSAM RLS (Cache/Memory)
0.5K	0.5K	4.0K	2.0K
1.0K	1.0K	8.0K	4.0K
1.5K	2.0K	12.0K	6.0K
2.0K	4.0K	16.0K	8.0K
2.5K	8.0K	20.0K	10.0K
3.0K	12.0K	24.0K	12.0K
3.5K	16.0K	28.0K	14.0K
4.0K	20.0K	32.0K	16.0K
4.5K	24.0K		18.0K
5.0K	28.0K		20.0K
5.5K	32.0K		22.0K
6.0K			24.0K
6.5K			26.0K
7.0K			28.0K
7.5K			30.0K
8.0K			32.0K
10.0K			
12.0K			
14.0K			
16.0K			
18.0K			
20.0K			
22.0K			
24.0K			
26.0K			
28.0K			
30.0K			
32.0K			

VSAM Cluster → Possible 28 possibilities
CICS LSR Definitions → 11 possibilities
CICS LSR Hipercache Definitions → 8 possibilities
VSAM RLS Cache/Memory → 16 possibilities

Note the possibility of buffer fragmentation due to the differences in VSAM CISZ availability vs. CICS TS LSR Buffer availability

RLS Information: Terri Menendez "VSAM RLS Best Practices"

Buffer Page Boundary Alignment

- This is a minor recommendation to avoid fragmentation
- LSR buffers are allocated in 4K increments on a 4K boundary
- Small buffer sizes should be allocated in multiples of 4K
  - .5K buffer → multiple of 8 (8 * .5K = 4K)
  - 1K buffer → multiple of 4 (4 * 1K = 4K)
  - 2K buffer → multiple of 2 (2 * 2K = 4K)
- For example, supposed you requested ten .5K buffers. LSR would allocate 8K of which it would use 5K losing 3K
  - Increasing the allocation to 16 buffers would gain 6 additional buffers without increasing the amount of storage

Number of LSR Pools

Two schools of thought on this issue (could be more)

- School 1 → use as many pools as possible so files can be segregated to reduce buffer contention and/or interference
- School 2 → define as few pools as possible (preferably 1) so that resources can be used more efficiently

Considerations

- LRU algorithm works best with a larger number of buffers
- Do you allocate a "fudge factor" to each pool's definition?
- Are the files continuously used? What happens to the resources when the file is in low activity?
- Unless you are using VSAM threadsafe, access to the different pools are single threaded via the QR TCB

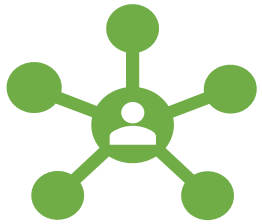
Number of LSR Pools

There are 255 LSR pools available in CICS TS

Originally, access to the LSR buffers used a sequential search which required more CPU when you increased the number of buffers

- **So, you were given 8 pools to distribute this load**

Access to the LSR buffers was changed and now uses a hashing algorithm that eliminates the concern over the length of the search caused by number of buffers



Remember LSR is Local Shared Resources

Everyone contributes to the sharing of resources

Number of LSR Pools

- When should we consider using more pools?
  - Data Tables
    - Any output operations go against the VSAM file
    - LSR pool is used for look-aside before going to the disk
    - ROT = 90%+ activity should be read/browse operations
    - Problem – due to the low activity of a data table going to the LSR pool, the odds are very high that the file's buffer are no longer in the LSR pool forcing an access to disk
    - So, you should define all your data tables into a separate pool where they only compete with other data tables for the LSR buffers
  - VSAM threadsafe files (multiprocessor environment)
    - You are not limited to only using QR TCB and can use an L8/L9 TCB
    - To attain overlap access you would need to separate files across several pools

C\TREK Corporation 2019

LSR Pool Non-Candidates

SHARE Option 4 files should not be in LSR

- **Direct reads require the reading of the CI from disk ignoring the fact that the CI is already in a buffer**

Files that do not follow Command Level Guidelines

- **STARTBR, READNEXT, READNEXT etc. READ for UPDATE (non-RLS)**

Files with a very high CA split activity

- **Use NSR to add BUFND to accelerate split**
- **Cost maybe lower look-aside hit ratio**

Hidden Tuning Areas

VSAM CA Size

- **CA size (CASZ) is an important tuning option**
- **CASZ is indirectly defined through the primary and secondary allocation**
  - **Bad CASZ can occur for any file that is incorrectly defined**
    - **However, it is more prone to happen for small files (less than 1 cylinder)**
    - **Valid CASZ**
      - **Track Managed Storage (TMS) → 1 to 15 tracks**
        - **Striped Data Set → 16 tracks**
      - **Cylinder Managed Storage (CMS) → 1, 3, 5, 7, 9 and 15 tracks (EAV – Extended Address Volumes)**
        - **Multi Cylinder Units (MCU) → 21-cylinder allocations (315 tracks)**
  - **Incorrect setting of the CASZ can result in unnecessary index CIs**
    - **Unnecessary index CIs would require buffers**
  - **The rule you must remember is that all data CAs must be the same size throughout the file (primary and secondary)**
    - **If requested space is in CYLINDERS, the CASZ will be 1 cylinder**
    - **If requested in TRACKS, use the Highest Common Denominator**
    - **Be careful when using RECORDS, can lead to bad CASZ**

C\TREK Corporation 2019

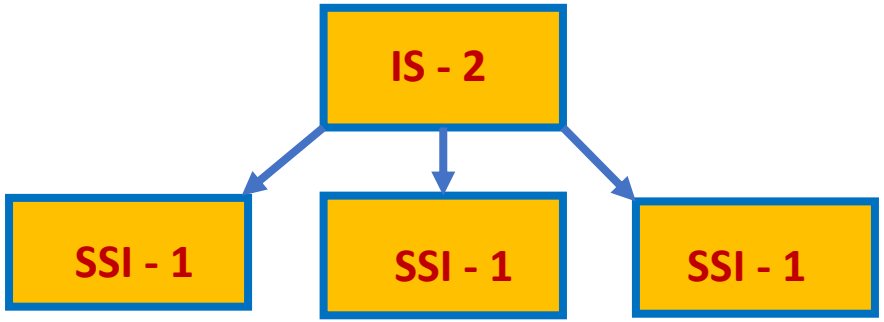
VSAM CA Size

- **Request space in cylinders**
- **If you need to ask space for a small file in tracks, make the primary = the secondary to ensure the maximum CASZ to reduce the number of indices**
  - **Examples**
    - **TRACKS (7 7)**
    - **TRACKS (3 3)**
    - **TRACKS (10 10)**
- **If requesting space in RECORDS, make sure the primary and secondary generate the maximum CASZ**
  - **Uses the Average Record Length to determine the space**
  - **Do you really know what the average record length is for a variable file?**

C\TREK Corporation 2019

VSAM CA Size

TRACKS (3 1) → If all three tracks are used, generates 2 index levels and 4 index records
Requires 4 index buffers



TRACKS (3 3) → If all three tracks are used, generates 1 index level and 1 index record
Requires 1 index buffer

CI/CA Splits

CI/CA splits are the result of adding new records or extending the length of variable length records

Splits result in physical I/O operations

- **CI splits – several**
- **CA splits – many**
- **The Good, the Bad and the Ugly**
  - **The Good – VSAM file can continue to operate accepting the split**
  - **The Bad – many I/O operations can result, especially for a CA split**
  - **The Ugly – if an extent needs to be acquired …**

Besides the actual number of physical I/O operations required to service the split, the cost of adding a new extent (data and/or index) because of additional I/O activity to the catalog, VVDS and VTOC

CI/CA splits can create free space that cannot be used or has a very low possibility of being used

CI/CA Splits

However, there is a hidden cost to CI/CA splits that is not normally discussed

CI splits take a single CI and convert it to two CIs which are approximately 50% filled

- **Instead of being able to access the data in one buffer, you now would need two buffers (half-full) to access the same amount of data**
- **This is a hidden cost of CI splits**
- **Important for most active files**

CA splits come as a result of not having a free CI in the CA, causing the CA split

- **However, splits can also affect the index component requiring additional indices and therefore, more LSR buffers**

CI/CA Splits

- Recommendations
  - Best defense against splits is % of free space allocated
    - Can help defer/reduce splits
    - Ensure that % free space allocated will result in saving space to store a record in the CI or leave a free CI in the CA
    - Analyze if changing the CISZ combined with a different free space can help reduce splits
  - Reorganization
    - Reorganizing the file will eliminate the splits but ….
      - You may wind up with taking splits again because you eliminated the free space created by the splits
      - The major keys that indicate that you may want to reorganize are:
        - CI splits represent 30% of the file
        - CA splits represent 30% of the file
        - Total free space within the HURBA exceeds 30%

Extents

- VSAM allows for a file to have multiple extents
  - Extent processing has improved over time (Space Constraint Relief –DFSMS)
    - 123 extents per volume
    - Total number of extents 7,257 (123*59)
    - Extent consolidation
- The cost of an extent occurs when obtaining an extent because the process involves accessing and updating the catalog, VVDS and the VTOC
  - The data set is serialized during a CI/CA split while the extent is being processed
    - The data set is serialized for CI/CA splits/reclaims for VSAM,
    - The data set is serialized for CA splits/reclaims for RLS
  - This can take time depending on the workload

Note: Many thanks to Ms. Terri Menendez of IBM for providing valuable information for this section

Extents

- However there is an additional hidden cost processing the data set
  - Each extent results in a control block called an EDB (Extent Definition Block) that consists of a header (X'20' bytes long) and an entry for each extent (X'20 bytes long)
    - Contains extent information starting/ending CCHH and starting/ending RBA
  - These extents have a pointer to another control block LPMB (Logical to Physical Mapping Block) (X'28' bytes long)
    - Physical characteristics of the device
      - CA size in tracks
      - # of CIs per CA
      - Physical record size
      - # Physical records per track
  - To reduce the overhead of searching for the correct EDB, VSAM creates an extent index control block
    - Works like a hashing technique used by CICS

Extents

- DSN DFHCMACD CICSTS55.DFHCMACD BASE CLUSTER
- CL DATA NAME CICSTS55.DFHCMACD.DATA LOCAL
- CL INDX NAME CICSTS55.DFHCMACD.INDEX VARIABLE BLOCKED
- VSAM KSDS OPEN ENABLED READ
- DATA EXTENTS 8 DEVICE 3390 TOTAL TRACKS ALLOC 285
- XTN STR-CCHH END-CCHH START-RBA END--RBA TRACKS VOLID UCB-ADDR UNIT
- 0 01040000 010F000E 0 9,215,999 180 OSW00G 0256DCE8 0D90
- 1 00510000 0051000E 9,216,000 9,983,999 15 OSW00G 0256DCE8 0D90
- 2 00730000 0073000E 9,984,000 10,751,999 15 OSW00G 0256DCE8 0D90
- 3 00780000 0078000E 10,752,000 11,519,999 15 OSW00G 0256DCE8 0D90
- 4 00980000 0098000E 11,520,000 12,287,999 15 OSW00G 0256DCE8 0D90
- 5 00CE0000 00CE000E 12,288,000 13,055,999 15 OSW00G 0256DCE8 0D90
- 6 00CF0000 00CF000E 13,056,000 13,823,999 15 OSW00G 0256DCE8 0D90
- 7 00D20000 00D2000E 13,824,000 14,592,000 15 OSW00G 0256DCE8 0D90

EXTENTS

```
7F2E2EE8  00000000  90000020 00000008 00000000 00000000 ................
7F2E2EF8  00000010  00000000 00000000 0000E400 7F2A90D8 ..........U."..Q
7F2E2F08  00000020  01040000 010F000E 00000000 0000012B ................
7F2E2F18  00000030  0256DCE8 00000000 0000E400 7F2A90D8 ...Y......U."..Q
7F2E2F28  00000040  00510000 0051000E 0000012C 00000144 ................
7F2E2F38  00000050  0256DCE8 00000001 0000E400 7F2A90D8 ...Y......U."..Q
7F2E2F48  00000060  00730000 0073000E 00000145 0000015D ...............)
7F2E2F58  00000070  0256DCE8 00000002 0000E400 7F2A90D8 ...Y......U."..Q
7F2E2F68  00000080  00780000 0078000E 0000015E 00000176 ...........;....
7F2E2F78  00000090  0256DCE8 00000003 0000E400 7F2A90D8 ...Y......U."..Q
7F2E2F88  000000A0  00980000 0098000E 00000177 0000018F .q...q..........
7F2E2F98  000000B0  0256DCE8 00000004 0000E400 7F2A90D8 ...Y......U."..Q
7F2E2FA8  000000C0  00CE0000 00CE000E 00000190 000001A8 ...............y
7F2E2FB8  000000D0  0256DCE8 00000005 0000E400 7F2A90D8 ...Y......U."..Q
7F2E2FC8  000000E0  00CF0000 00CF000E 000001A9 000001C1 ...........z...A
7F2E2FD8  000000F0  0256DCE8 00000006 0000E400 7F2A90D8 ...Y......U."..Q
7F2E2FE8  00000100  00D20000 00D2000E 000001C2 000001DA .K...K.....B....
7F2E2FF8  00000110  0256DCE8 00000007 00000000 00000000 ...Y............

7F2A90D8  00000000  91880028 00000019 0000C800 00002800 jh........H.....
7F2A90E8  00000010  000F000F 00050003 004B0C00 0000FF20 ................
7F2A90F8  00000020  00062F58 81AAD300 32C9C4C1 D4D9C5E3 ....a.L..IDAMRET
```

EDB Header

EDBs

LPMB

X’12C’ * X’7800’ = X’8CA000’ → 9,216,000

Extents

- Avoid taking new extents online
  - Ensure proper space allocation
    - Specifically, make sure you are not meek requesting secondary allocations
      - For example, avoid requests like CYL(200 2)
      - Instead CYL(100 25)
      - If you are going to pay the price of a new extent, make sure you get enough space to avoid coming back continually
  - Try to reduce CA splits by requesting proper Free Space %
    - May require changing the data component CISZ
  - Reorganize the file to improve the free space allocation and/or the data component CISZ

Free Space Example

Current file has a 4K CISZ with a CI free space of 25% and a fixed record length of 1000

- A 4K CISZ results in 180 CI/CA
- 25% CI free space (4096 * .25 = 1024) or one free record per CI
- This results in leaving space for 180 records per CA (180 * 1)

Change the file CISZ to 18K and a free space of 22%

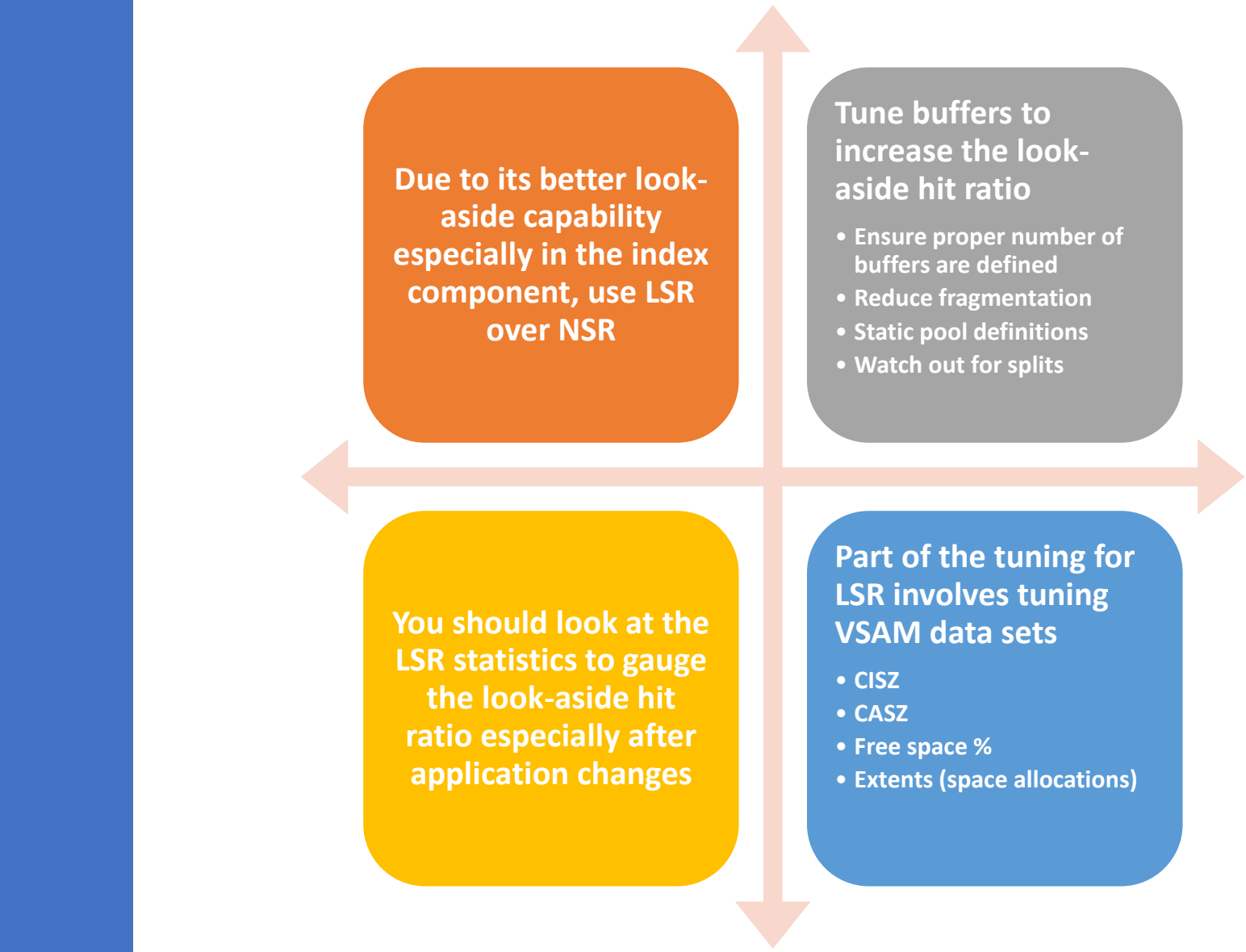
- An 18K CISZ results in 45 CI/CA
- 22% CI free space (18432 * .22 = 4055) or 4 free records per CI
- This results in leaving space for 180 records per CA (45 * 4)

Free Space Example

- **Space Requirement 4K**
  - **Records/CI = (4096 – (4096 * .25) - 10) / 1000 = 3**
  - **CIs required = (100000 / 3) = 33333.33 → 33334**
  - **Cylinders required = (33334 / 180) = 185.18 → 186 cylinders**
- **Space requirement 18K**
  - **(Records/CI = (18432 – (18432 * .22) – 10) / 1000 = 14**
  - **CIs required = (100000 / 14) = 7142.857 → 7143**
  - **Cylinders required = (7143 / 45) = 158.73 → 159 cylinders**
- **Difference 186 – 159 = 27 cylinders less or 14.5% space cylinders**

CLOSING

Closing


Due to its better look-aside capability especially in the index component, use LSR over NSR
Tune buffers to increase the look-aside hit ratio
• Ensure proper number of buffers are defined
• Reduce fragmentation
• Static pool definitions
• Watch out for splits
You should look at the LSR statistics to gauge the look-aside hit ratio especially after application changes
Part of the tuning for LSR involves tuning VSAM data sets
• CISZ
• CASZ
• Free space %
• Extents (space allocations)

Questions