IBM GLOBAL SERVICES

B55

Implementing IMS Hardware Data Compression

Rich Lewis

IMS
Technical Conference

Sept. 27-30, 2004

Orlando, FL

RETURN TO INDEX

Abstract

Data compression can save DASD space, decrease I/Os, and improve the performance of your IMS databases. Implementing it can be easy. This session describes how compression is implemented in IMS. It provides advice on deciding which segments to compress, which compression routines to use, and which parameters to specify when implementing compression. It includes implementation using only the facilities in the IMS product as well as the use the IMS Hardware Data Compression Extended tool.

Agenda

- Overview of IMS Compression
  - IMS Exit Routine Options
- Hardware Data Compression
  - Use of dictionaries
- IMS Options with Compression
  - Key Compression vs. Data Compression
  - Split Full Function Segments
  - Setting Minimum Size for Segments
  - Adjusting Database Space Parameters
- Definition and Implementation
  - Defining Compression for a Segment
  - Implementation Steps
  - Using IMS Hardware Data Compression - Extended

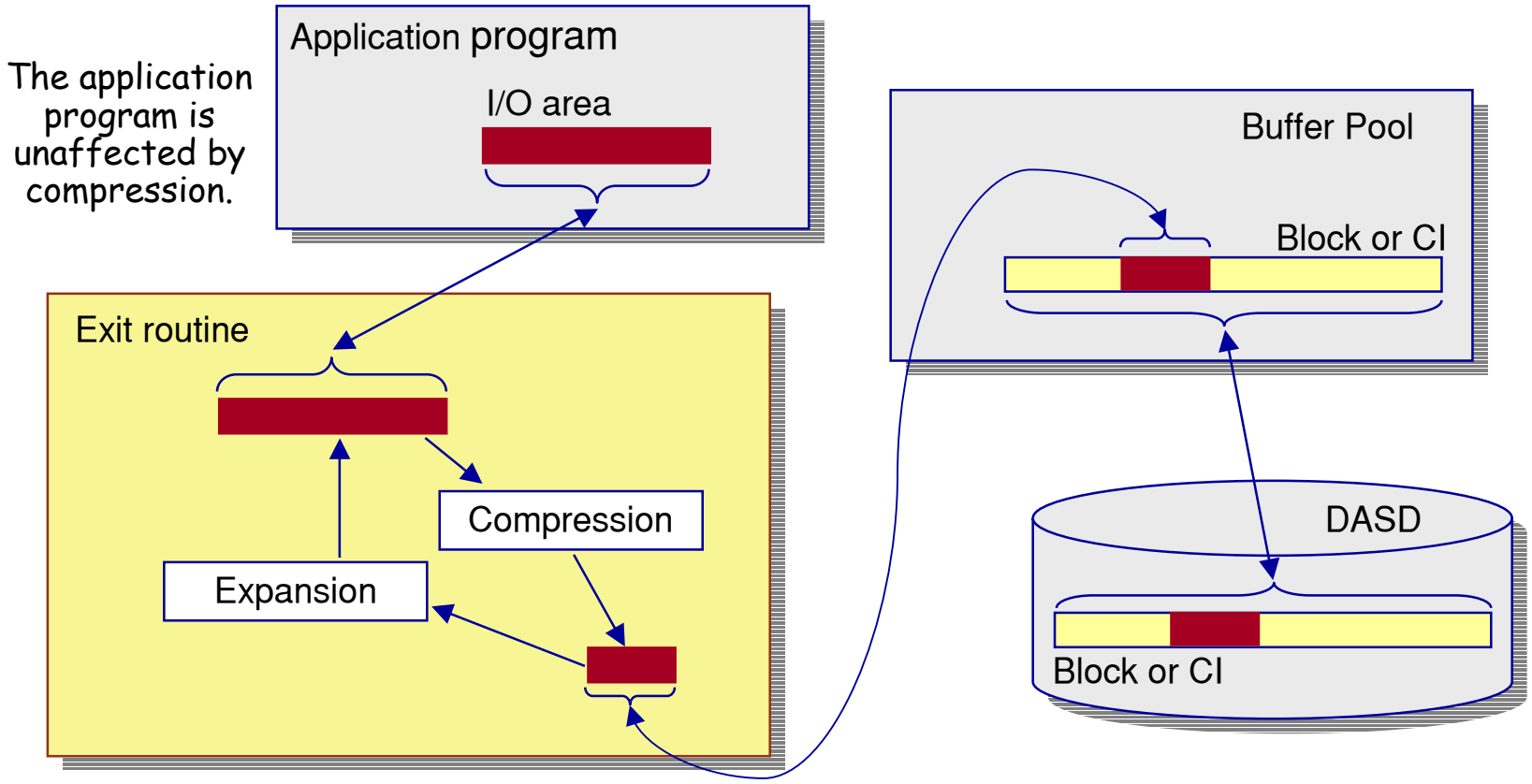
Overview of IMS Compression

Overview of IMS Compression

- Compression is implemented with segment edit/compression exit routines
  - Segments may share a routine
  - Different segments may have different routines
  - Some segments may be compressed while others are not compressed
- Compression is available for:
  - HDAM, HIDAM, HISAM, PHDAM, PHIDAM, and DEDB databases
- Compression is not available for:
  - HIDAM indexes, PHIDAM indexes, and secondary indexes
  - Logical children

Overview of IMS Compression

- Segment with compression



Exit Routine Options

- IMS supplied sample exit routine (DFSCMPX0)
  - Implements run length encoding (RLE)
    - Compresses string of a byte repeated four or more times to three bytes
      - Very effective for some times of data, such as data with many blanks
- Hardware data compression
  - IMS supplies facilities to generate exit routines using hardware data compression
    - Uses Ziv-Lempel technique
      - Compresses variable numbers of bytes to 12 bits
    - Dictionary is built to optimize compression for sample input data
- User written exit routine
  - Can use any technique
  - Rarely used

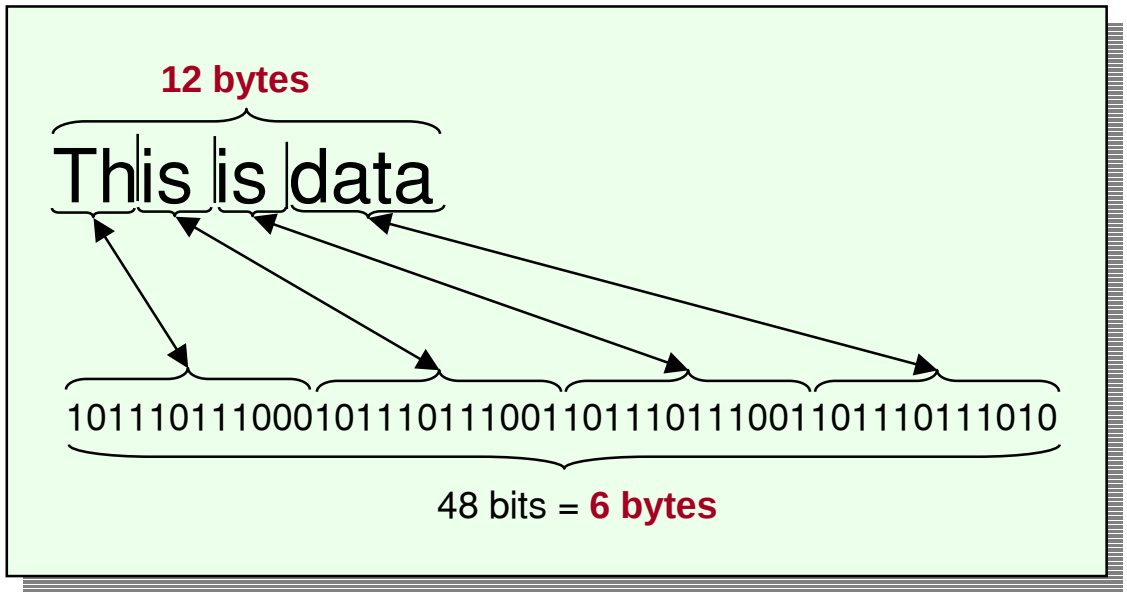
Hardware Data Compression

Hardware Data Compression

- CPU hardware is used to assist in compression and expansion
  - Lessens CPU overhead of compression and expansion
- Ziv-Lempel technique is used
  - Dictionary is used to map data for compression and expansion
  - User must build dictionary
  - IMS supplies capability to build dictionaries

An Example of Ziv-Lempel Compression

- Strings of bytes are compressed to 12 bits
  - Dictionary maps
    - Variable length byte strings to fixed length (12) bit values (compression)
    - Bit values to byte strings (expansion)



Dictionary

Compression

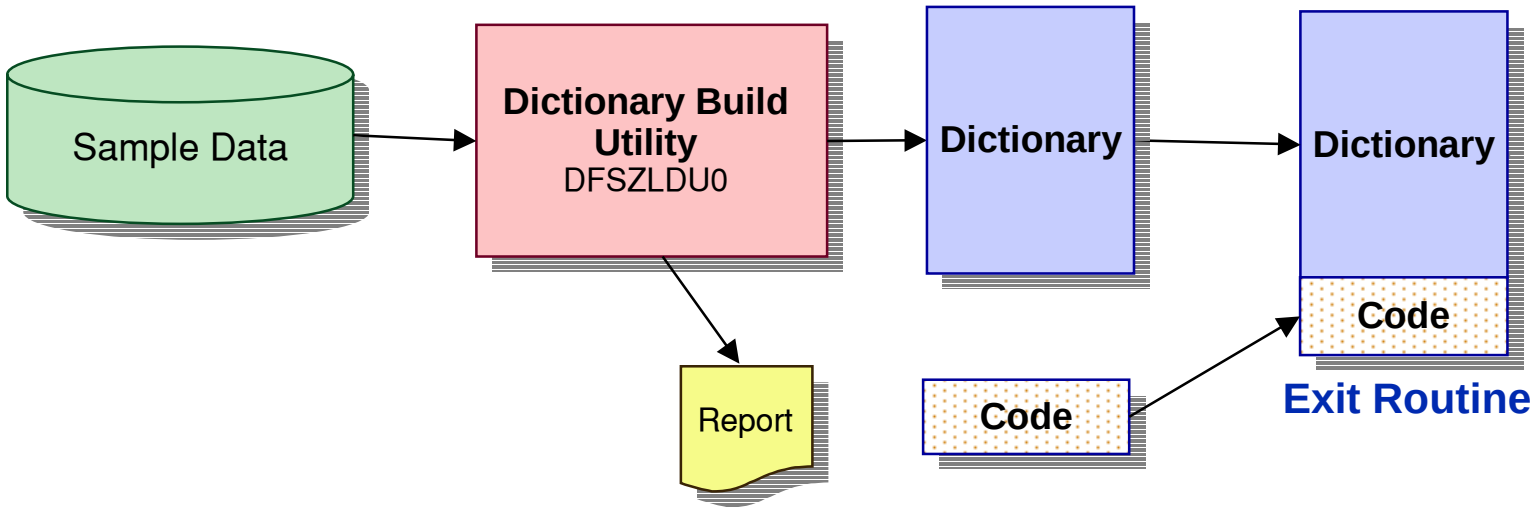
'data'	101110111010
'is '	101110111001
'Th'	101110111000

Expansion

101110111000	'Th'
101110111001	'is '
101110111010	'data'

Hardware Data Compression

- Building an exit routine which uses hardware data compression
  - Dictionary build routine reads sample data and creates a dictionary
  - The dictionary is bound (link edited) with stub code to build an IMS segment edit/compression exit routine



Building a Dictionary

- IMS supplies a utility to build dictionaries (DFSZLDU0)
  - Reads sample data
    - Any sequential data set with variable length records
      - HD Unload utility output may be used
        - All segments in data set will be used to build dictionary
      - Users may create their own sample data
        - Restrict data to that which will be compressed
    - IMS Hardware Data Compression Extended (HDCE) tool
      - Reads HD Unload output
      - Reads Image Copies
      - Has option to select only instances of specified segments
  - Dictionary can be built from 200K to 400K bytes of data
    - No need to read megabytes or gigabytes of data
    - Once dictionary is filled, build process ignores the rest of input

Dictionary Strategies

1. Use generic dictionary supplied by HDCE tool
   - Developed from English text data
     - Sufficient for many segments
     - Typically provides 40 to 60 percent compression
   - Avoids building and managing dictionaries
2. Build a few enterprise dictionaries
   - Examples:
     - Text input
     - Numeric input
     - Text and numeric input
   - Limited number of dictionaries to manage
   - Good compression for all segment types
3. Build unique dictionary for each compressed segment
   - Excellent compression
   - Higher DBA management costs
4. Combination of the above

Evaluating Dictionaries

- Dictionary build utility (DFSZLDU0) reports potential savings
  - Uses dictionary it builds to compress sample data
- HDCE tool can examine existing dictionaries
  - Analyzes compression of dictionary on sample data
    - Sample data may be HD Unload file, Image Copy, or user file
  - Answers questions:
    - Will the generic dictionary be adequate for this segment?
    - Will another existing dictionary be adequate for this segment?
    - Has the data changed so much that I need a new dictionary?

IMS Options with Compression

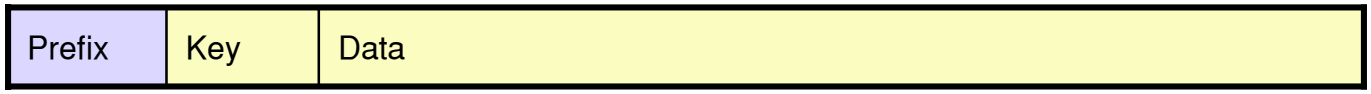
Key Compression vs. Data Compression

- Key compression
  - Compresses key and the rest of the data in the segment
  - Not available for DEDBs and HISAM
  - Compresses all of the segment
  - Causes significant overhead for some DL/I calls
    - Keys must be expanded before IMS can examine them
    - Especially dependent segments with long twin chains

- Data compression
  - Compresses only the data following the key
  - Available for all database types
  - Less compression than key compression
  - More efficient for some DL/I calls
    - Especially dependent segments with long twin chains

Compression example 1

- Key is in the first field of the segment
  - Segment without compression



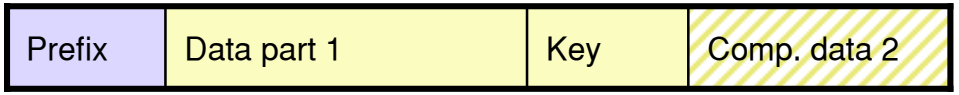
  - Data compression

  - Key compression

Compression example 2

- Key is not in the first field of the segment
  - Segment without compression

  - Data compression



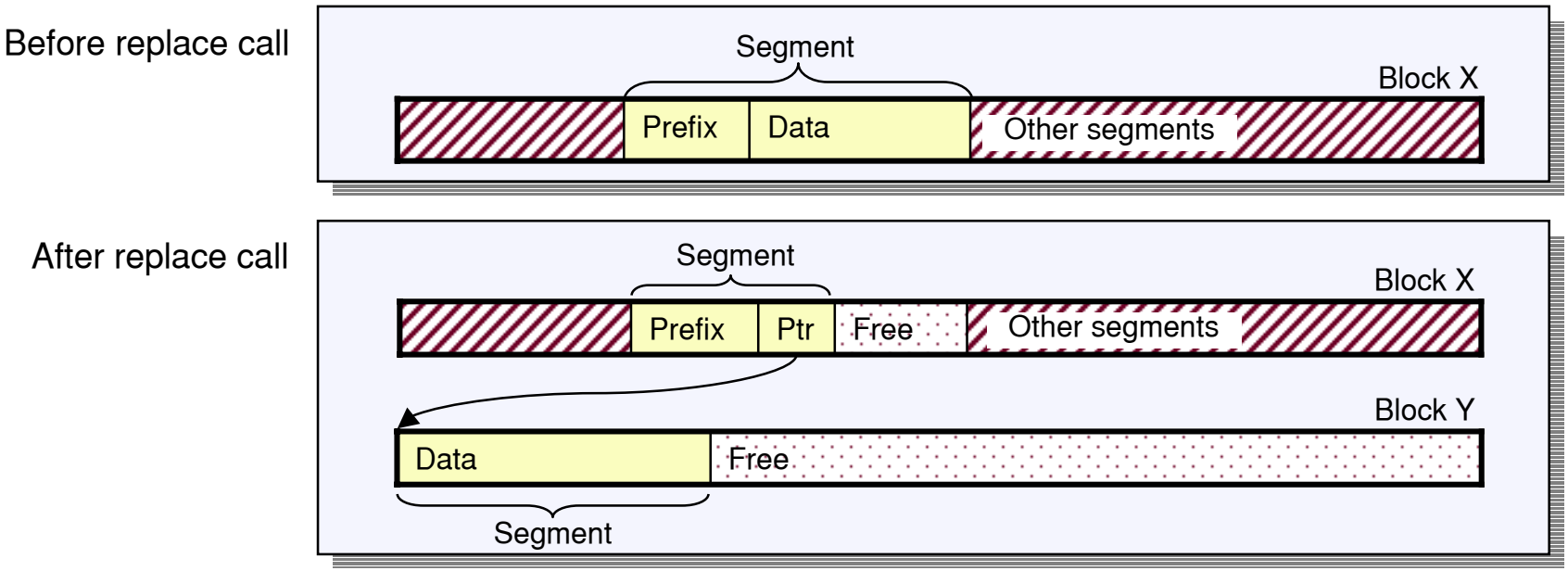
  - Key compression

Key Compression vs. Data Compression

- Should you use key compression?
  - Factors to consider:
    - Size of the key
    - The location of the key in the segment
    - Processing done against the segment
      - Will many keys have to be expanded for searches?
        - Long dependent segment twin chains?

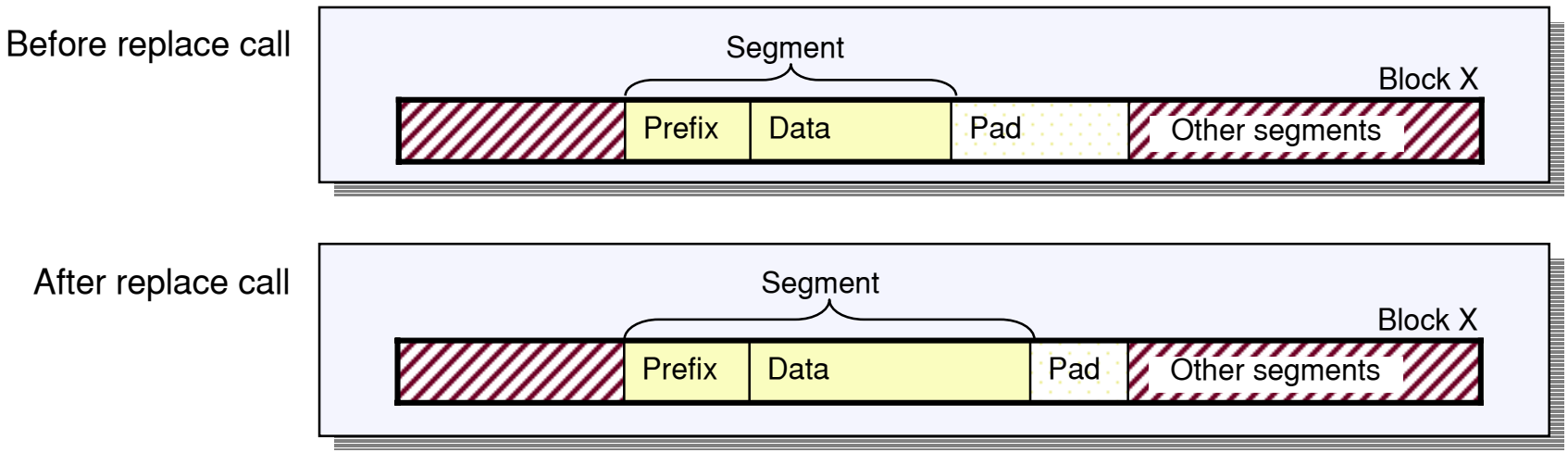
Split Full Function Segments

- Split full function segments are caused by some replace calls
  - Insufficient space for segment with increased length
    - Split segments have prefix and data parts in different locations
    - Split segments may require extra I/Os
- Fast Path DEDB segments are never split
  - The entire segment is moved to a new location



Minimum Sizes for Full Function Segments

- You may want to set a minimum size for full function segments
  - Segment is padded to the minimum size
  - Can help avoid “split segments”



Specifying Minimum Sizes for Segments

- Full function fixed length segments
  - COMPRTN= parameter of SEGM statement
    - Specify PAD for fifth positional subparameter
    - Specify minimum size in fourth positional subparameter
  - Example specification:

    `COMPRTN=(MYRTN,DATA,INIT,150,PAD)`
    - Minimum size is 150 bytes
- Full function variable length segments
  - BYTES= parameter of SEGM statement
    - Specify the minimum size in the second subparameter
  - Example specification

    `BYTES=(500,150)`
    - Minimum size is 150 bytes
  - You may need to adjust minimum size to get sufficient space savings

Minimum Sizes for DEDB Segments

- Variable length DEDB segments
  - Minimum size on DBD SEGM statement is minimum size for application program
    - Compression routine may make segment smaller on DASD
- Fixed length DEDB segments
  - Size on DBD SEGM statement is size for application program
    - Compression routine may make segment smaller on DASD

Adjusting Database Space Parameters

- Database space parameters may need to be adjusted
  - Smaller segments will change how space is used
- You might want fewer HALDB partitions
- You might want fewer DEDB areas
- You might want smaller root addressable area/part
  - Smaller space requirements
  - Sequential processes read all of the blocks/CIs
    - In (P)HDAM root addressable area
    - In DEDB root addressable part

Adjusting HDAM and PHDAM Parameters

- RMNAME parameters on DBD statement or PHDAM partition definition
  - Number of blocks in root addressable area (RAA)
    - A smaller number may be used
    - You may want more free space
  - Number of RAPs per block in RAA
    - If RAA size is decreased, the number of RAPs per block should be increased
      - Maintain the same number of RAPs per root
  - 'bytes' parameter may need to be adjusted
    - 'bytes' is typically used to preserve space in RAA for other records
    - Space is RAA will be used differently

Adjusting DEDB Parameters

- ROOT parameters on AREA statement
  - You may want a smaller number of UOWs
    - For root addressable part
    - For independent overflow
    - For both
- UOW parameters on AREA statement
  - You may want smaller UOWs
  - You may want a smaller number of CIs used for dependent overflow

When Will Compression Reduce I/Os?

- Sequential processing
  - Smaller records require fewer blocks
    - Assumes that (P)HDAM and DEDB parameters are adjusted
    - Fewer blocks implies fewer I/Os
  - Compression is usually great for sequential batch work
- Random processing
  - A smaller database record may require fewer blocks/CIs
    - If so, I/O savings occur
  - More data per block implies more data in buffers
    - May decrease probability of I/O
  - Compression is sometimes beneficial for random processing
- Reducing I/Os reduces CPU usage
  - Could offset the CPU cost of compression

Definition and Implementation

Specifying a Compression Routine in the DBD

- Segment Edit/Compression routine is specified on SEGM statement
  - Full function fixed length segments:

```
                        ~¬,DATA¬®
Ê¬COMPRTN=¬¬(¬¬routinename¬¬¬¬,KEY
¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬)¬Í
                                    □¬,INIT¬¯
□¬,size¬¬¬¬¬¬¬¬«
□¬,PAD¬¯
```

  - Full function variable length segments:

```
                        ~¬,DATA¬®
Ê¬COMPRTN=¬¬(¬¬routinename¬¬¬¬,KEY
¬¬¬¬¬¬¬¬¬¬¬¬¬¬)¬Í
□¬,INIT¬¯
```

  - Fast Path DEDB segments

```
                        ~¬,DATA¬¬®
Ê¬COMPRTN=¬¬(¬¬routinename¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬
¬¬¬)¬Í
□¬,INIT¬¯
```

Used to specify a minimum size

Exit routine is driven at initialization and termination. Required with Hardware Data Compression

Data compression is the default. Key compression is optional for full function.

Specifying a Segment Size in the DBD

- Size is specified on SEGM statement
  - Fixed length segments:

```
Ê¬BYTES=bytes
```

    - Do not change the size specification when adding a compression routine.

  - Variable length segments:

```
Ê¬BYTES=¬¬(maxbytes,minby
tes)
```

    - DEDBs: Do not change the size specification when adding a compression routine.
    - Full function:
      - Adjust the minimum size if needed to avoid splitting segments or to save space
      - Do not change the maximum size specification when adding a compression routine except in very rare circumstances
        - See next page

Compression Routines Might Not Compress

- Compression routines might not compress
  - All exit routines might expand instead of compress highly unusual data
    - IMS supplied routines might add a maximum of one byte to segment size
- Handling segments which are not compressed (i.e. expanded)
  - Fixed length segments and DEDB variable length segments
    - These segments may exceed specified size by up to 10 bytes
  - Full function variable length segments
    - These segments may not exceed specified maximum size
    - Maximum size might need to be increased by 1 byte
      - Very rarely required, maybe never required in the “real world”

Hardware Data Compression Implementation Steps

1. Evaluate which segments to compress in a database
2. Determine which dictionary to use
   - Generic (supplied by HCDE tool)
   - Enterprise
   - Customized
3. Create dictionary if required
4. Link-edit dictionary with IMS-supplied base exit routine code
   - This produces an exit routine
5. Unload the database using old DBD
6. Modify DBD to specify exit routine for segment(s)
7. Run ACBGEN for the modified DBD
8. Reload the database using the modified DBD
9. Image copy the database data sets

IMS Hardware Data Compression Extended Tool

- IMS productivity aid for DBAs
  - Simplifies the implementation and management of compression for IMS
  - ISPF based tool
    - Analyzes segment data
      - Input: HD Unload or Image Copy
      - Output: Report on potential compression
        - Uses temporary dictionary built from input data
    - Extracts segment data for building dictionaries
      - Input: HD Unload or Image Copy
      - Output: File for each segment type selected
    - Builds dictionary and creates exit routine from extracted data
    - Compresses data (reload with DBD specifying exit routine)
    - Examines existing dictionaries
      - Reports how they would compress input data

Summary

Implementing Hardware Data Compression

- Hardware Data Compression is powerful
  - Compression of 50% to 75% is typical
  - Included in IMS product
- IMS Options with Compression
  - Evaluate key compression and minimum segment sizes
  - Adjust space parameters if necessary
- Definition and Implementation
  - IMS Hardware Data Compression – Extended tool simplifies the implementation and management of compression

- White Paper: A Guide to IMS Hardware Data Compression
  - http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100416