zEDC: Huge Response Time Improvements in Compression

Anthony Sofia (atsofia@us.ibm.com)
Senior Software Engineer at IBM
August 14th 2015

#SHAREorg

Explosive Growth in Data

Every day over 2000 petabytes of data are created

Data Compression will become pervasive

- I/O throughput is struggling to keep up with increasingly data driven applications
- Batch workloads are accessing more data from disk and network connections
- Business opportunities can be lost due to the cost prohibitive nature of keeping data online

Data needs to be shared across different platforms

- Data is being exchanged among business partners
- Compression can substantially reduce the amount of data transferred
- Industry standard formats need to be used for transparent peer to peer communication

Compression solves problems in the enterprise

- Improves the effective throughput of data over storage and communication networks
- Allows more data to remain online for increased business value
- Reduces the amount of data for encryption operations
- Typically improves batch turnaround
- Make storage technology including Flash Memory more affordable

Storage technology, including Flash, more affordable with compression

IBM z Enterprise Data Compression (zEDC)

New data compression offering that can reduce resource usage

What is it?

- ✓ *zEDC Express is an IO adapter that does high performance industry standard compression*
- ✓ *Used by z/OS® Operating System components, IBM Middleware and ISV products*
- ✓ *Applications can use zEDC via industry standard APIs (**zlib** and **Java**™)*
- ✓ *Each zEDC Express sharable across 15 LPARs, up to 8 devices per CEC.*
- ✓ *Raw throughput up to **1 GB/s** per zEDC Express Hardware Adapter vs typical 50 MB a second in software*

What Changes?

It is time to revisit your decisions about compression.

- **Disk Savings:** Many people are already getting value from CMPSC compression and software compression today
- **Performance:** High throughput alternative to existing IBM System z® compression for large or active files.
- **Industry Standard:** Low cost compressed data exchange across all platforms
- **Pervasive:** Standard APIs allow quick adoption by middleware products running on System z

What is the Value?

New sources of customer value

- **QSAM/BSAM** can save up to 4x disk space and in some cases shorten elapsed time, reducing batch windows.
- **Business Partner Data Exchange** can have higher throughput with lower CPU cost
- **Managed File Transfer** saves up to 4x link bandwidth, and up to 80% elapsed time
- **ISV Products** deliver expanded customer value
- **Java for z/OS V7R1** accelerates common compression classes used by applications and middleware

z Systems Compression Technology Overview

Using the right hardware compression acceleration for each of your workloads

Compression Coprocessor	z Enterprise Data Compression
On Chip In every IBM eServer™ zSeries® today (and tomorrow) Mature: Decades of use by Access Methods and DB2® Work is performed jointly by CPU and Coprocessor Propriety Compression Format	PCIe Adapter New with IBM zEnterprise® EC12 GA2 and IBM zEnterprise BC12 Mature: Industry Standard with decades of software support Work is performed by the PCIe Adapter Standards Compliant (RFC1951)

Use Cases

Small object compression	Large Sequential Data	Industry Standard Data
▪ Rows in a database	▪ QSAM/BSAM Online Sequential Data ▪ Objects stored in a data base	▪ Cross Platform Data Exchange
Users	**Users**	**Users**
▪ VSAM for better disk utilization ▪ DB2 for lower memory usage ▪ The majority of customers are currently compressing their DB2 rows	▪ QSAM/BSAM for better disk utilization and batch elapsed time improvements ▪ SMF for increased availability and online storage reduction	▪ Java for high throughput standard compression via java.util.zip ▪ Encryption Facility for z/OS for better industry data exchange ▪ IBM Sterling Connect: Direct® for z/OS for better throughput and link utilization ▪ ISV support for increased client value

Sequential Data Compression with BSAM/QSAM and zEDC

Reduce the cost of keeping your sequential data online

- zEDC compresses data up to 4X, saving up to 75% of your sequential data disk space
- Capture new business opportunities due to lower cost of keeping data online

Better I/O elapsed time for sequential access

- Potentially run batch workloads faster than either uncompressed or QSAM/BSAM current compression

Sharply lower CPU cost over existing compression

- Enables more pervasive use of compression
- Up to 80% reduced CPU cost compared to tailored and generic compression options

Simple Enablement

- Use a policy to enable the zEDC

Example Use Cases

SMF Archived Data can be stored compressed to increase the amount of data kept online up to 4X

zSecure output size of Access Monitor and UNLOAD files reduced up to 10X and CKFREEZE files reduced by up to 4X

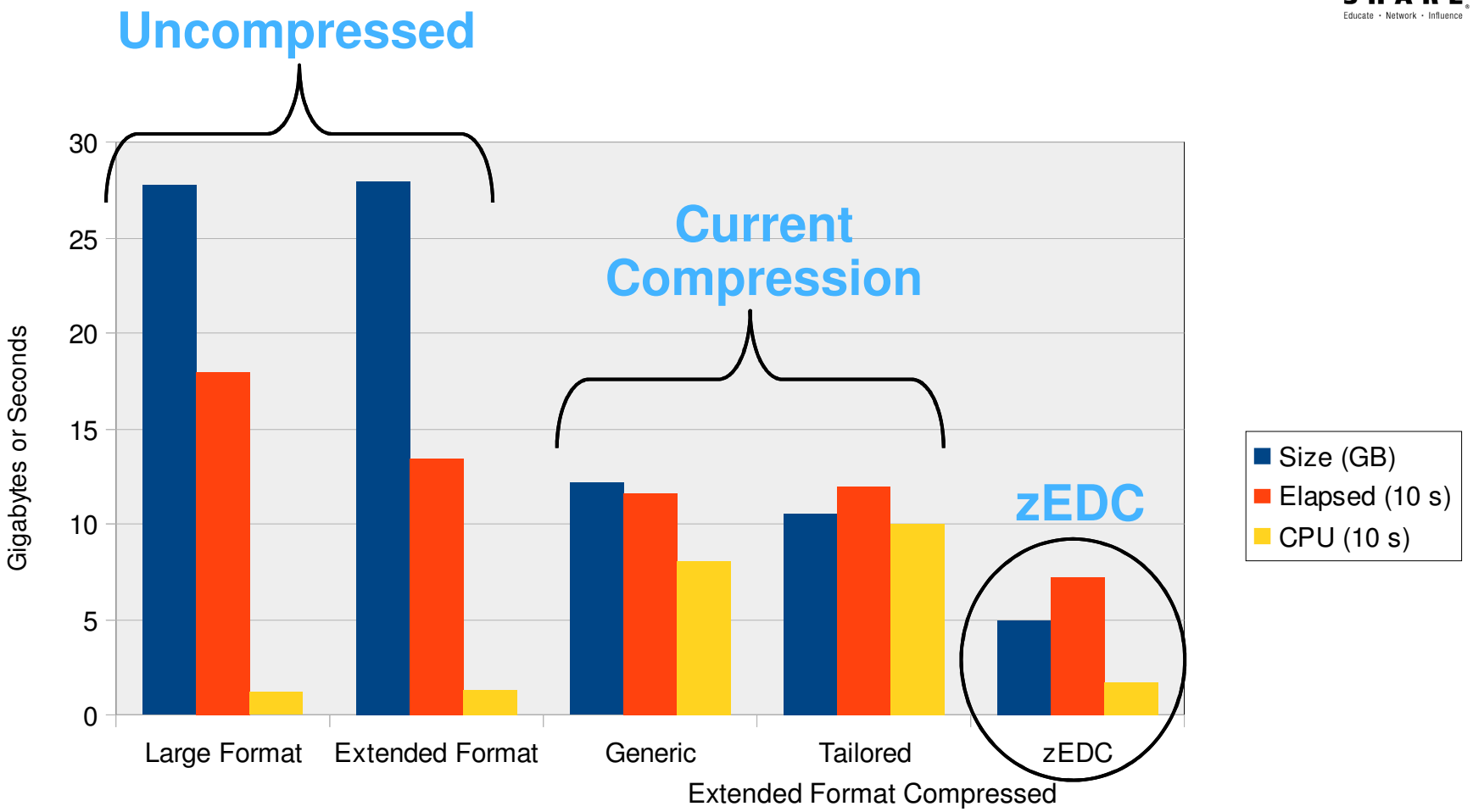
Up to 5X more ***XML*** data can be stored in sequential files

The ***IBM Employee Directory*** was stored in up to 3X less space

z/OS SVC and Stand Alone DUMPs can be stored in up to 5X less space

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.

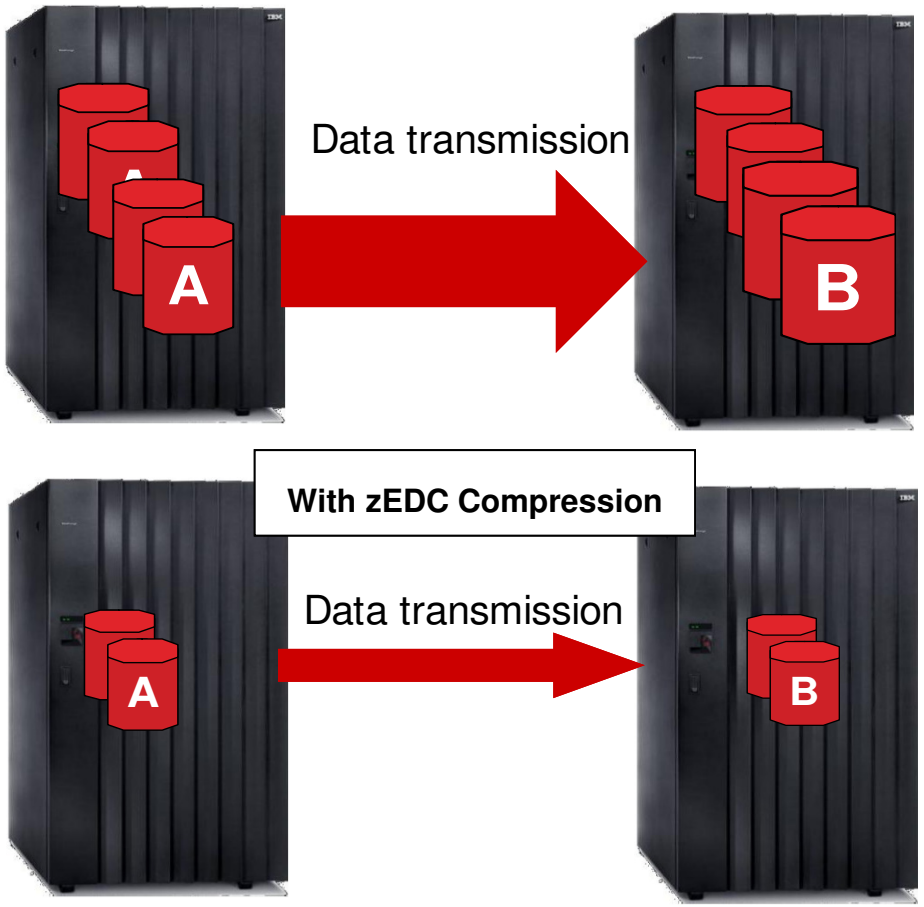
Sequential Data Compression with zEDC – Value!



Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.

Data Replication

- Replication technologies which move data in physical format can take advantage of the reduced storage requirements of data compressed with zEDC.
  - Significant amounts of zEDC compressed data can reduce the amount of data transferred as well as the elapsed time to complete the transfer.



DFSMShsm and DFSMSdss Usage of zEDC

- **DFSMSdss DUMP** command
  - In addition to existing COMPRESS and HWCOMPRESS options on DUMP command, new **ZCOMPRESS (REQUIRED | PREFERRED | NONE)** option will take advantage of zEDC compression.
    - Accepted for all FULL, TRACKS, physical and logical DATASET backups to DASD and tape
    - The use of zEDC for backups can be restricted using a new facility class profile: STGADMIN.ADR.DUMP.ZCOMPRESS

- **DFSMShsm** will use the **DFSMSdss zEDC** support (via **ZCOMPRESS(PREFERRED)**) in
  - Migrate/Recall
  - Backup/Recover
  - Full Volume DUMP
  - Recover and FRRECOV from DUMP

  Exception: zEDC will not be used during migration or backup functions when DFSMShsm is the data mover. Partitioned Data Sets will utilize the standard DFSMShsm compaction methodology in place.

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.

zEDC with WebSphere MQ for z/OS V8

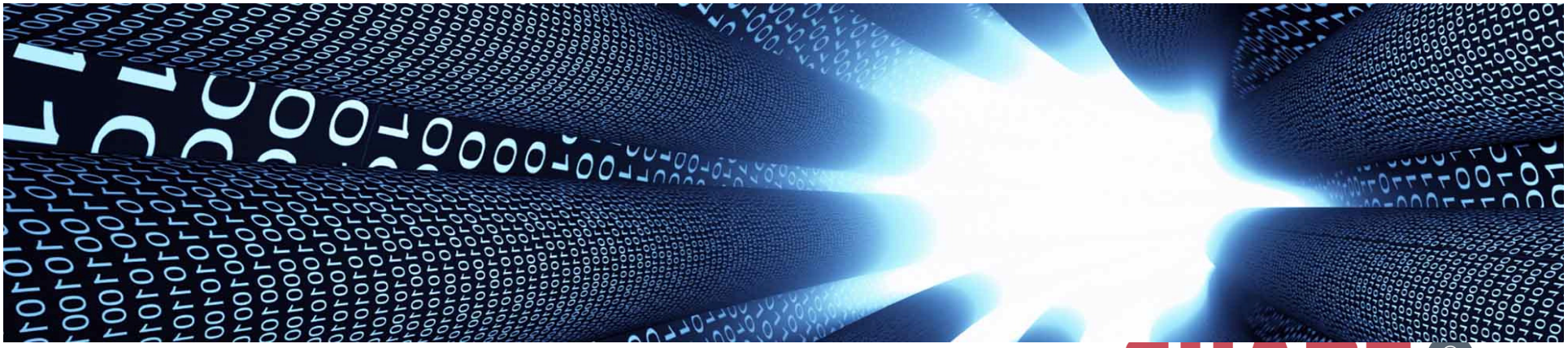
WebSphere MQSeries® has always provided compression options for message data passed over MQ channels via the COMPMSG attribute.

The existing zlib options are the following:

- ZLIBFAST - Message data compression is performed using the zlib compression technique. A fast compression time is preferred.
- ZLIBHIGH - Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

Starting with WebSphere MQ for z/OS V8 the COMPMSG(ZLIBFAST) attribute will now use zEDC when available to perform compression and decompression of message data.

This support is ideal for channels that handle large, 32KB requests.



zEDC with SMF Logstream Recording

Alleviate SMF constraints across the entire life cycle of a record using compression technology

Record Creation

Records created by z/OS, DB2, CICS® and applications are all written to SMF

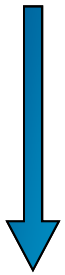
Record Buffering

zEDC Express can be used to compress SMF data resident in the z/OS System Logger cutting down on logger storage requirements.

Record Extraction

SMF can extract compressed data from logger faster than uncompressed data.

Targeting a compressed QSAM/BSAM data set for long-term archival can also optimize disk space. zEDC, tailored or generic compression can be used depending on the requirements



Record Archival

Reading SMF data from a compressed data sets can increase the performance of applications that access that data.

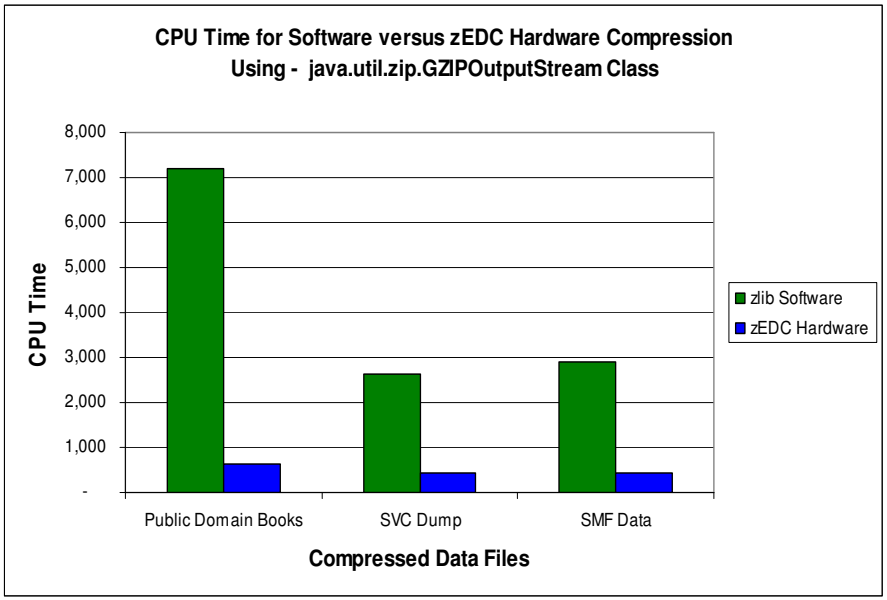
- Store up to **4x** less data in System Logger
- Logger CPU usage reduced by up to **30%**
- Up to **15%** reduction in elapsed time for SMF extraction from Logger
- SMF data stored in zEDC compress BSAM can save up to **4x** in archived SMF data size
- Programs reading SMF data from a zEDC compressed data set can see an elapsed time reduction

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels. (BSAM)
Disclaimer: These results are based on projections and measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels (Logger)

zEDC with Java

Transparent enablement of the *java.util.zip* package enables high throughput compression and decompression

- zEDC java.util.zip.Deflater in memory test improved elapsed time up to 55x and CPU time up to 240x when compared to zlib software compression.



Java with zEDC can be used for

- Application Business Partner Data Exchange
- HTTP Responses for Web Services
  - Servlet Filters
  - WebSphere® Web Services component
- Large objects that are serialized and stored
- Processing gzip or zip files
- Exploited through standard Java APIs java.util.zip* in the latest releases of Java 7.0.0, and Java V7R1
  - Java application to compress files using java.util.zip.GZIPOutputStream class
    - Up to 90% reduction in CPU time using zEDC hardware versus zlib software
    - Up to 74% reduction in Elapsed time using zEDC hardware versus zlib software

* Not all java.util.zip classes exploit zEDC

Disclaimer: Results are based on internal controlled measurements using java.util.zip.Deflater on data already in memory. Results may vary based on the application's use of java.util.zip classes and other work done by the application

zEDC with Java Application – Encryption Facility for z/OS

Increased throughput and functionality for standard compliant business partner data exchange. The Encryption Facility (EF) for z/OS can now use zEDC to compress and decompress data

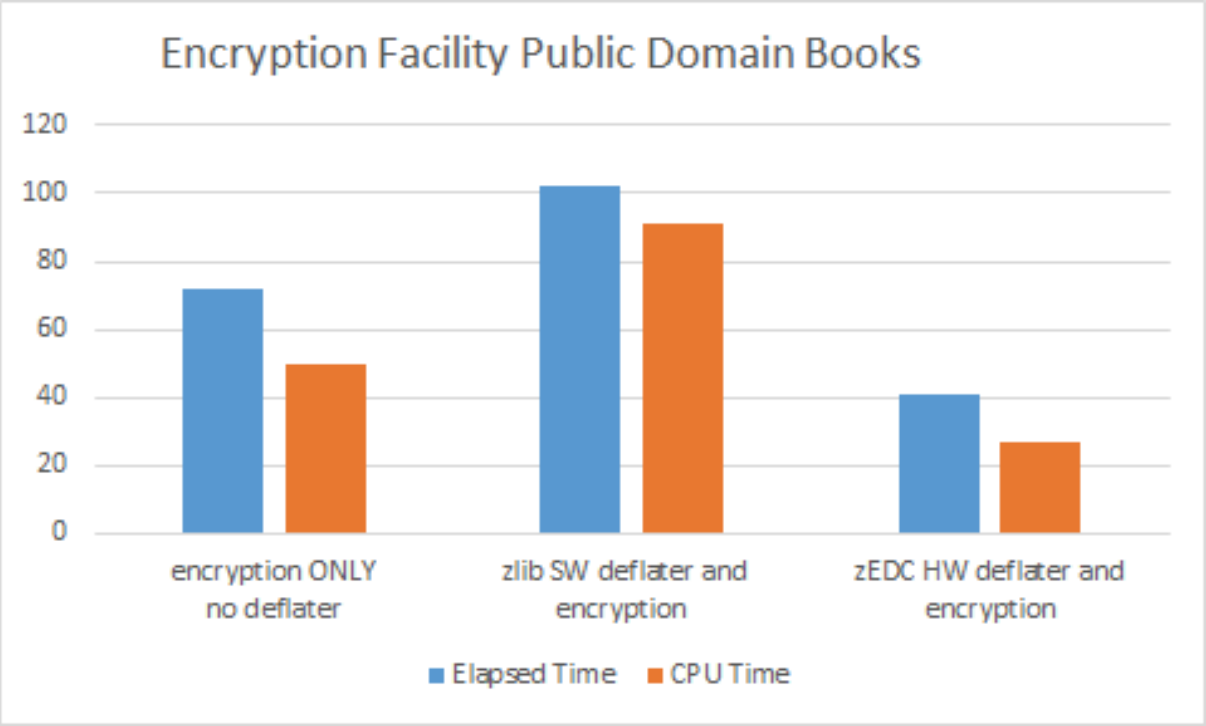
1. Before performing an encryption and after performing a decryption
2. As a stand-alone operation NEW

Compressing data with zEDC enables higher throughput than no compression or software compression

zEDC can provide IBM EF users reductions of up to 60% in elapsed time and up to 70% in CPU time for environments where compression is already in use

For IBM EF users not already using compression, compression with zEDC can provide IBM EF users a reduction of up to 44% in elapsed time and up to 46% in CPU times

Results based on files containing public domain books. Results may vary by customer based on individual workload, data, configuration, and software levels.



Disclaimer: Results based on internal controlled measurements using IBM Encryption Facility for files containing public domain books. Results may vary by customer based on individual workload, data, configuration and software levels.

What is DEFLATE and GZIP?

The DEFLATE file format is defined by the IETF RFC1951 document. The generation of the DEFLATE data is up to each implementer.

There is a combination of two processes:

LZ77 (Lempel-Ziv 1977) – Provide pattern matching via a 32k rolling window in the data. As matches are found they are replaced with a back reference to the match.

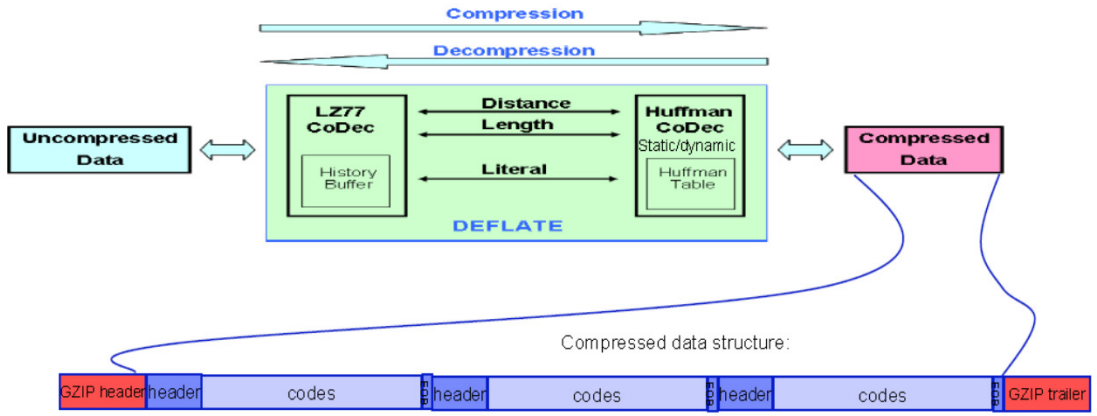
Huffman Coding – Encodes the symbols in the file into a set of bit patterns where the most used symbols get the smallest bit patterns.

There are two types of approaches for **Huffman Coding**

Static or Fixed Huffman – A predefined alphabet is used to encode the symbols. This alphabet is defined in RFC1951.

Dynamic Huffman – The Huffman Tree is defined based on the symbols in the stream. The Huffman Tree alphabet is embedded in the DEFLATE block.

The file format is a BIT aligned file; meaning that symbols do not fall on byte boundaries.



Application Programming with zEDC

- Several application programming interfaces exist
  - Authorized APIs – Requires supervisor state, key0 execution. Provides direct access to zEDC – Not covered in this presentation!
  - zlib - Problem State, requires LE runtime
  - Java – Provided by the java.util.zip package

zlib

zlib is a widely used open source C library that provides compression and decompression. It supports RFC1950 (ZLIB), RFC1951 (DEFLATE), and RFC1952 (GZIP).

zlib supports a streaming model such that files can be compressed/decompressed in chunks.

Things to know about using zlib with zEDC

- The address space using zEDC needs READ access to the FPZ.ACCELERATORS.COMPRESSION SAF resource
- The FIRST deflate() or inflate() request must be at least as large as the minimum threshold setup in IQPPRMxx
- The window size for deflate must be 32k
- The _HZC_COMPRESSION_METHOD environmental variable can be used to force software even when zEDC is available

zlib Configuration Options

The IQPPRMxx member in SYS1.PARMLIB can be used to adjust internal settings for zlib behavior. For example

```
ZEDC,DEFMINREQSIZE=5,MAXSEGMENTS=7
```

This member will set the minimum input size of a deflate request via zlib to 5Kb and will set the maximum internal buffer size to 7 16Mb segments. The current settings and buffer usage can be displayed with the D IQP command:

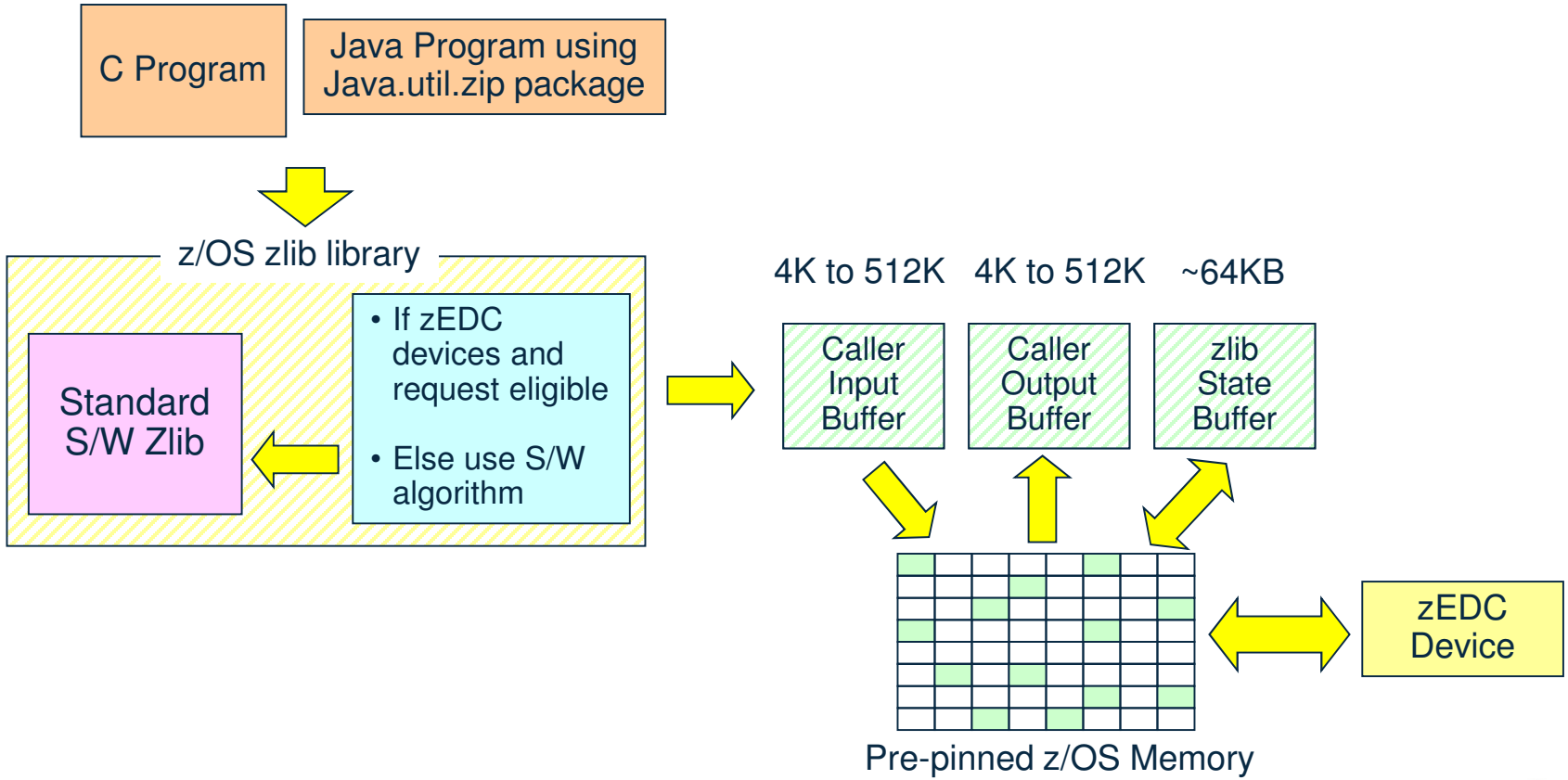
```
D IQP
IQP066I 10.12.37 DISPLAY IQP 364
 zEDC Information
  MAXSEGMENTS:                    7   (112M)
  Previous MAXSEGMENTS:           4   (64M)
  Allocated segments:             1   (16M)
  Used segments:                  0   (0M)
  DEFMINREQSIZE:                  5K
  INFMINREQSIZE:                 16K
  Feature Enablement:    Enabled
```

Updated values from IQPPRMxx

zlib internal structure

If the request can not be performed using zEDC then the software zlib code will be used for the request.

The zEDC requests are not done directly from user storage; the input data is copied into pre-allocated buffers and the output data is copied from these buffers back to user storage.



Using zlib with CEEPIPI

- The CEEPIPI interface can be used to create an LE runtime to execute zlib functions in
- This allows the IBM provided zlib to be used in many environments that may not have an LE runtime
- The `_HZC_COMPRESSION_METHOD` environmental variable can be set with the ENVAR keyword in the runtime options

Using zlib with CEEPIPI (cont)

- The pre-linker can be used to define custom short names for the zlib routines

```
//PLKED EXEC PGM=EDCPRLK,PARM='NOER,OMVS,MAP,NODYNAM,OE'

…

//ZLIB     DD  PATH='/usr/lpp/hzc/lib/libzz.a'

…

//SYSIN    DD  *
   RENAME  'inflate',INFLATE
   RENAME  'inflateEnd',INEND
   RENAME  'inflateReset',INRES
   RENAME  'inflateInit',ININ
   RENAME  'inflateHwAvail',HWAVAIL
   LIBRARY ZLIB
```

Using zlib with CEEPIPI (cont)

- The short names can be defined as external entry points and setup for CEEPIPI via the PIT

```
          EXTRN ININ2
          EXTRN HWAVAIL
          EXTRN INFLATE
          EXTRN INRES
          EXTRN INEND

PPTBL     CEEXPIT  ,
          CEEXPITY ,ININ2+X'80000000'
          CEEXPITY ,HWAVAIL+X'80000000'
          CEEXPITY ,INFLATE+X'80000000'
          CEEXPITY ,INRES+X'80000000'
          CEEXPITY ,INEND+X'80000000'
          CEEXPITS ,
```

Using zlib with CEEPIPI (cont)

- The `init_sub` function can be used to create an LE runtime for calling zlib routines
  - This provides a calling environment with minimal overhead and that keeps the memory between calls to preserve the zlib state
- The same environment created with `init_sub` can be used for multiple streams.
- Either deflateReset or inflateReset can be used between streams rather than performing a delfateEnd/inflateEnd and deflateInit/inflateInit
- If the LE runtime is terminated then any active zstrm is invalid
  - zlib mallocs areas and links them from the zstrm which would no longer be valid

Using zlib with CEEPIPI (cont)

- The `call_sub` function can be used to execute the individual zlib routines
  - Remember, this is C so not everything is pass by reference!
  - Lets look at a parameter list for calling inflate, defined as

```
ZEXTERN int ZEXPORT inflate OF((z_streamp strm, int flush));
```

  - Lets use a flush mode of Z_FINISH in this example

The zstrm area is storage obtained by you!

Java and zEDC with java.util.zip

Java 7.1 provides zEDC access via the java.util.zip Inflater and Deflater classes. The same conditions that apply to zlib also apply to Java.

This example (try/catch blocks removed) shows the critical buffer sizes

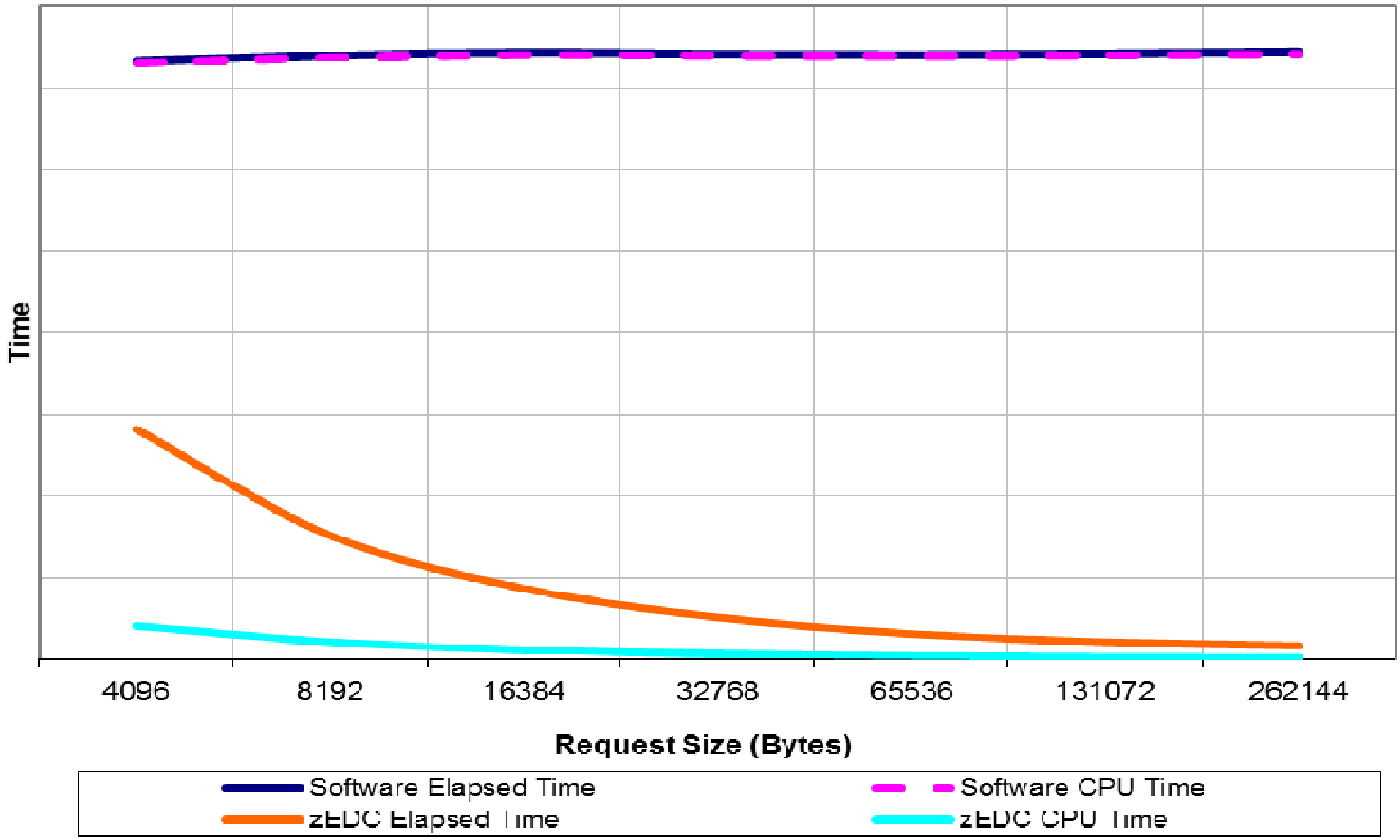
```
byte buffer[] = new byte[64 * 1024];
byte outputFile[];



input = new FileInputStream(argv[0]);
output = new ByteArrayOutputStream();
gzStream = new GZIPOutputStream(output, 4096);

for(;;) {
          readBytes = input.read(buffer);

          if(readBytes < 0) {
                break;
          }
          else {
                gzStream.write(buffer, 0, readBytes);
          }
}
```

64Kb input buffer for deflate(). This must reach the threshold

4Kb output buffer for deflate()

Data is read from an uncompressed file and written to a compressed file

zlib – Buffer Size Matters!



Disclaimer: Based on projections and/or measurements completed in a controlled environment.
Results may vary by customer based on individual workload, configuration and software levels.

Setting Buffer Sizes with Java

For the `Inflater` and `Deflater` classes the *input buffer* size is the size of the parameter passed via the `setInput` method.

For the `GZIPInputStream`, `DeflaterInputStream` and `InflaterInputStream` classes a constructor is provided which allows the *input buffer* size for the deflate or inflate operation to be specified. The buffer passed to the `read` method determines the size of the *output buffer*.

For the `GZIPOutputStream` and `DeflaterOutputStream` classes a constructor is provided which allows the *output buffer* size for deflate and inflate operations to be specified. For these classes the size of the buffer passed to the `write` method sets the *input buffer* size.

New zEDC HealthCheck

- Checks for a specific number of zEDC Express devices
  - Different thresholds exist for different severities
- Triggered as devices are brought online and offline to the system
  - This check is not triggered on a time interval
- Shipping with OA48434

New zEDC HealthCheck – Example Reports

- Sample report that reports a medium severity when less than 2 devices are available and a high severity when no devices are available
- In this example there are greater than 2 zEDC Express devices available

```
  Display  Filter  View  Print  Options  Search  Help
 ------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY HWAM_ZEDC_DEVICE_AVAILABILITY LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===>                                          SCROLL ===> PAGE
********************************** TOP OF DATA ***********************************
CHECK(IBMHWAM,HWAM_ZEDC_DEVICE_AVAILABILITY)
SYSPLEX:    LOCAL     SYSTEM: SY1
START TIME: 07/28/2015 08:50:56.944840
CHECK DATE: 20150303  CHECK SEVERITY: HIGH-DYNAMIC
CHECK PARM: DEVTHRESH_HIGH(1),DEVTHRESH_MED(2)


FPGH0002I No zEDC Express device availability exceptions have been
detected.

END TIME: 07/28/2015 08:50:56.947024  STATUS: SUCCESSFUL
******************************** BOTTOM OF DATA **********************************
```

New zEDC HealthCheck – Example Message

- Messages are issued when the exception is triggered
  - FPGH0001E for low severity exception
  - FPGH0002I for successful execution
- In this example there 20 devices available however the installation-specified threshold is 60

```
SY1  HZS0001I CHECK(IBMHWAM,HWAM_ZEDC_DEVICE_AVAILABILITY):
FPGH0001E The number of zEDC Express devices available is
20 and less than the installation-specified threshold of 60
```

zEDC RMF Reporting

New RMF™ report shows the utilization of each device.

RMF Postprocessor Interval Report : PCIE Activity Report

RMF Version : z/OS V2R1 SMF Data : z/OS V2R1
Start : 02/24/2014-05.48.00 End : 02/24/2014-05.48.44 Interval : 00:45:000 minutes

Hardware Accelerator Activity

Function ID	Time Busy %	Request Execution Time	Std Dev for Request Execution Time	Request Queue Time	Std Dev for Request Queue Time	Request Size	Transfer Rate T
0013	0.689	7.78	0.417	15.0	0.953	24.3	21.5

Hardware Accelerator Compression Activity

Function ID	Compression Request Rate	Compression Throughput	Compression Ratio	Decompression Request Rate	Decompression Throughput	Decompression Ratio	B
0013	885	11.6	1.14	0	0		64

The percent of this interval where this specific zEDC Express device was executing requests

Compression ratio of all requests serviced by zEDC. This will span all usage of the zEDC Express devices

Average request queue time in Microseconds for this device.

zEDC SMF30 Section

New zEDC section in the SMF30 record provides per-job zEDC usage statistics

– Shipped with OA45767

- Also track OA48268 (currently open) to address an issue with the Queue and Request time fields

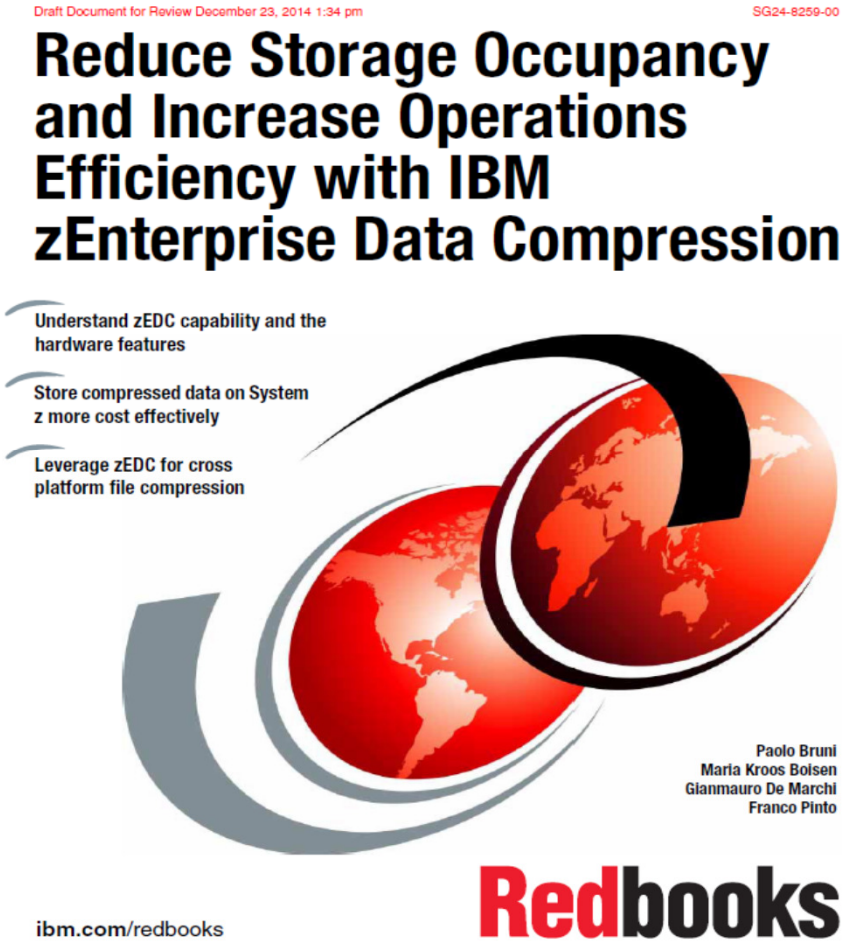
– The following statistics are provided in the new section

- Number of requests
- Number of Problem State (zlib) requests
- Total Queue Time
- Total Execute Time
- Uncompressed Input Bytes
- Compressed Output Bytes
- Compressed Input Bytes
- Uncompressed Output Bytes

zEDC Redbook

- **Provides overview of the technology**
- **Covers configuration of SMF, QSAM/BSAM and DFHSM/DFDSS**
- **Examples using zBNA**
- **Available Now on www.redbooks.ibm.com:**

  http://www.redbooks.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg248259.html?Open



References

- zEnterprise Data Compression – What is it and how do I use it? Share in Anaheim -- Session 15081

Thank You!

Complete your session evaluations online at www.SHARE.org/Orlando-Eval