Zip-Ada

Part 1: Shrink your data to (almost) nothing with Trained Compression.

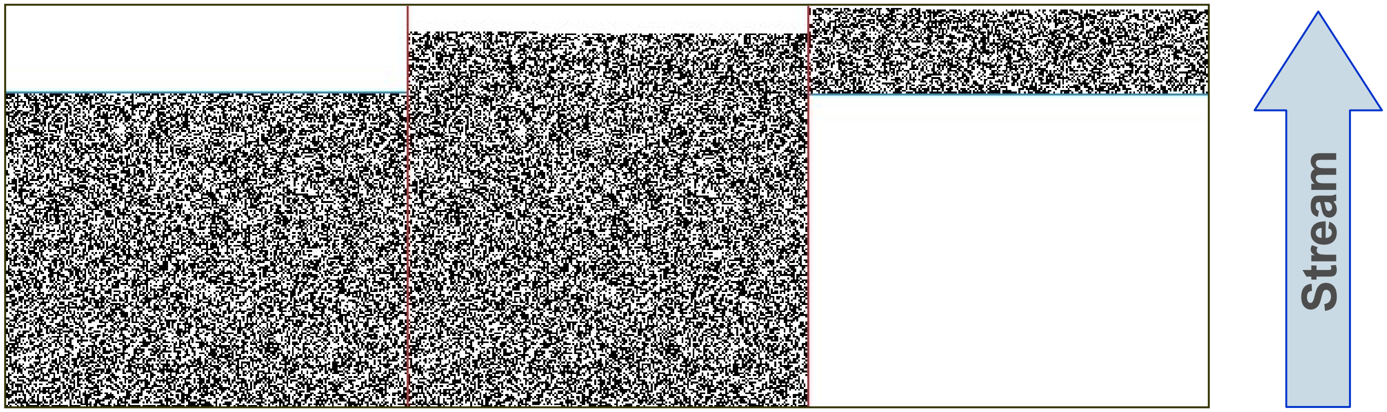
Alternative title (buzzwords version):

Enhance Your Data-Centric Compression To The Extreme Maximum By Leveraging Machine Learning 😉

Part 2: Overview of Zip-Ada – a complete open-source archive and compression library in Ada.

Gautier de Montmollin
FOSDEM 2019

Trained Compression – a little experiment…



a) Text 1, compressed

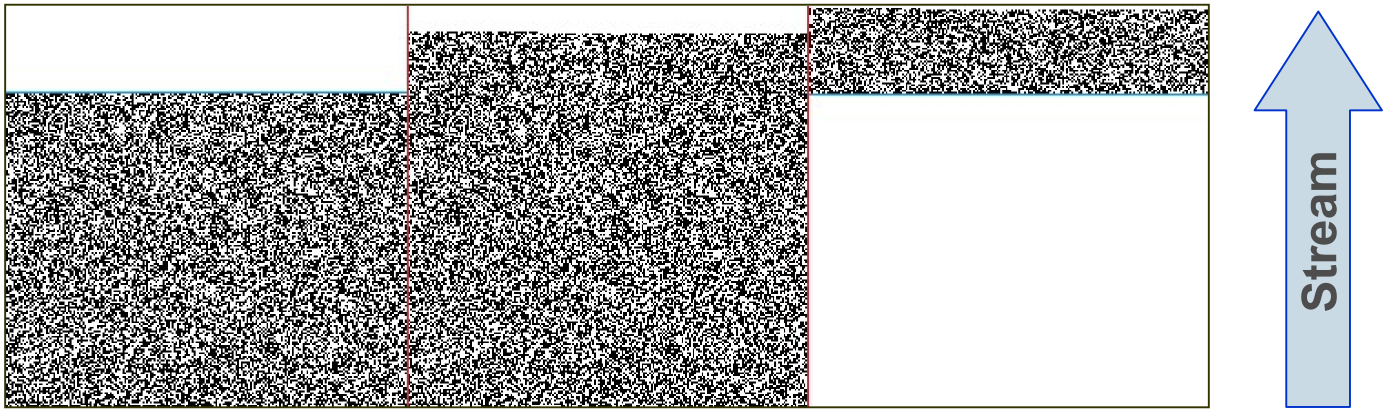
(streamed compression)

b) Text 1, then text 2, compressed

c) Text 2, compressed

- Compression: lzma.exe
- Text 1: zipada.txt
- Text 2: za_todo.txt

Trained Compression – a little experiment…



a) Text 1, compressed

(streamed compression)

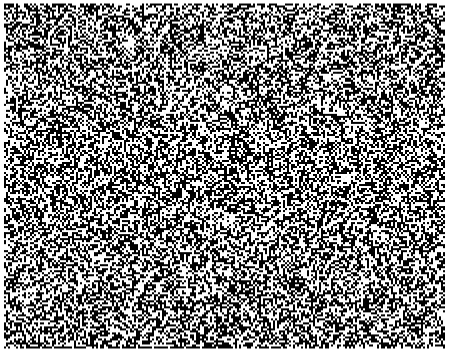
b) Text 1, then text 2, compressed

c) Text 2, compressed

1284 bytes (40% of original)

Δ = **951** bytes (29% of original)

a) ↔ b)



- Compression: lzma.exe
- Text 1: zipada.txt
- Text 2: za_todo.txt

Trained Compression – conclusion of the experiment

Compression is better with some training (*)…

How to implement it?

1) **Preset**: save the state of the “machine” (dictionary, probability model, …) corresponding to a training data. Restore state on use.

– or –

2) Use a **prefix data** for training the compression algorithm at run-time, just like in the experiment.

- Advantages:
  - You can leverage and reuse streamed compression algorithms “as-is”.
  - Extremely simple. No complex API, data structures, …
- Disadvantage: longer compression time (prefix data is wasted)

(*) similar to Machine Learning, for predictions

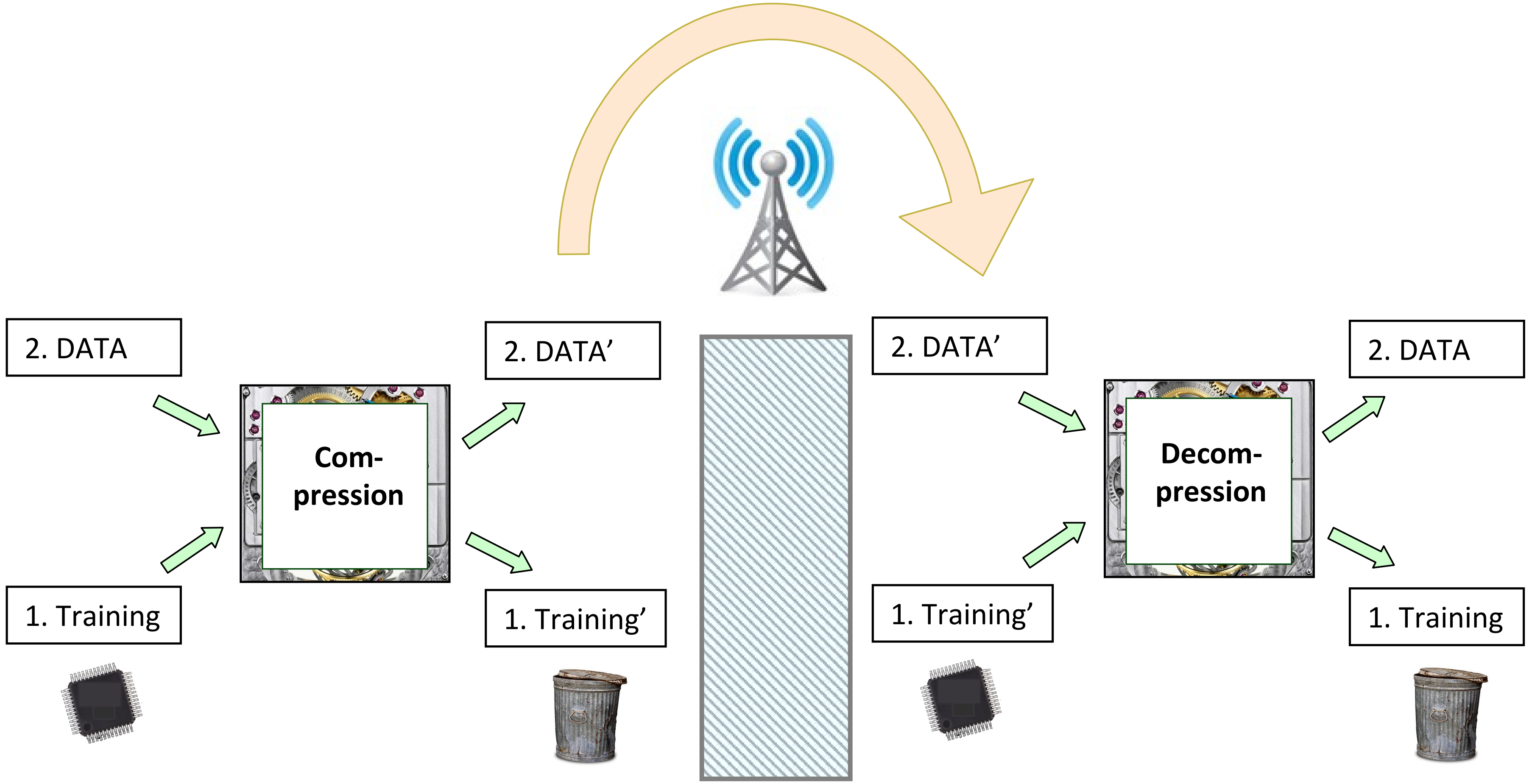
Trained Compression – purpose

You anticipate having a large or even an indefinite amount of data files which will be all similar to a sample known in advance (the training data).

Extra compression can save further storage and transmission time compared to untrained compression.

Trained Compression – workflow of approach 2) with prefix data

Our package, Trained_Compression, can be plugged onto any (preferably adaptive) streamed compression scheme.



2. DATA

1. Training

Com-pression

2. DATA’

1. Training’

2. DATA’

1. Training’

Decom-pression

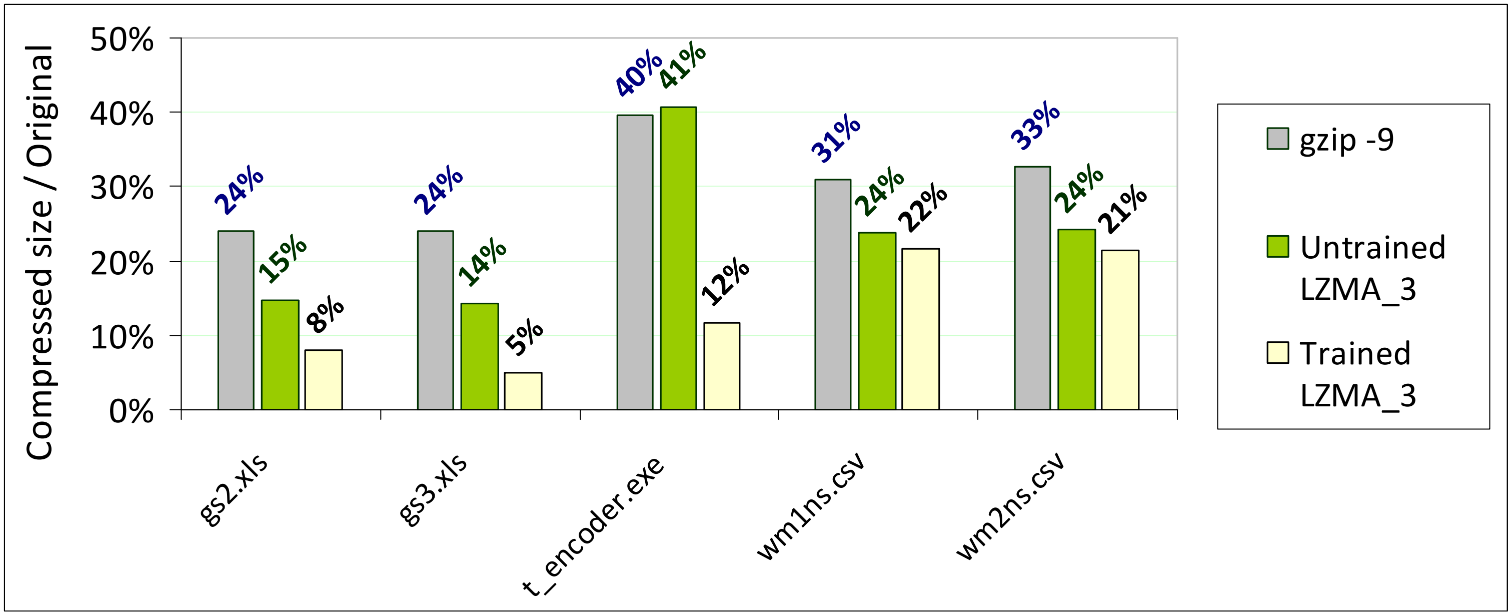
2. DATA

1. Training

Trained Compression – specification

```
--  Universal Trained Compression
---------------------------------

with Interfaces;

package Trained_Compression is

  subtype Byte is Interfaces.Unsigned_8;

  ------------------------------
  --  Encoding - compression  --
  ------------------------------

  generic
    type Data_Bytes_Count is range <>;
    --  Input of training or data bytes:
    with function Read_Uncompressed_Training return Byte;
    with function Read_Uncompressed_Data return Byte;
    with function More_Uncompressed_Data_Bytes return Boolean;
    --  Output of compressed data:
    with procedure Write_Compressed_Byte (B : Byte);
    --
  procedure Encode (Train_Uncompressed, Skip_Compressed : Data_Bytes_Count)

  --------------------------------
  --  Decoding - decompression  --
  --------------------------------

  generic
    type Data_Bytes_Count is range <>;
    --  Input of training or data bytes:
    with function Read_Compressed_Training return Byte;
    with function Read_Compressed_Data return Byte;
    --  Output of compressed data:
    with procedure Write_Decompressed_Byte (B : Byte);
    --
  procedure Decode (Train_Compressed, Skip_Decompressed : Data_Bytes_Count)

end Trained_Compression;
```

Trained Compression – results



Training file name	Data file name
gs1.xls	gs2.xls
gs1.xls	gs3.xls
trained_decoder.exe	trained_encoder.exe
wsbase.csv	wm1ns.csv
wsbase.csv	wm2ns.csv

Part 1: Shrink your data to (almost) nothing with Trained Compression.

Part 2: Overview of Zip-Ada – a complete open-source archive and compression library in Ada.

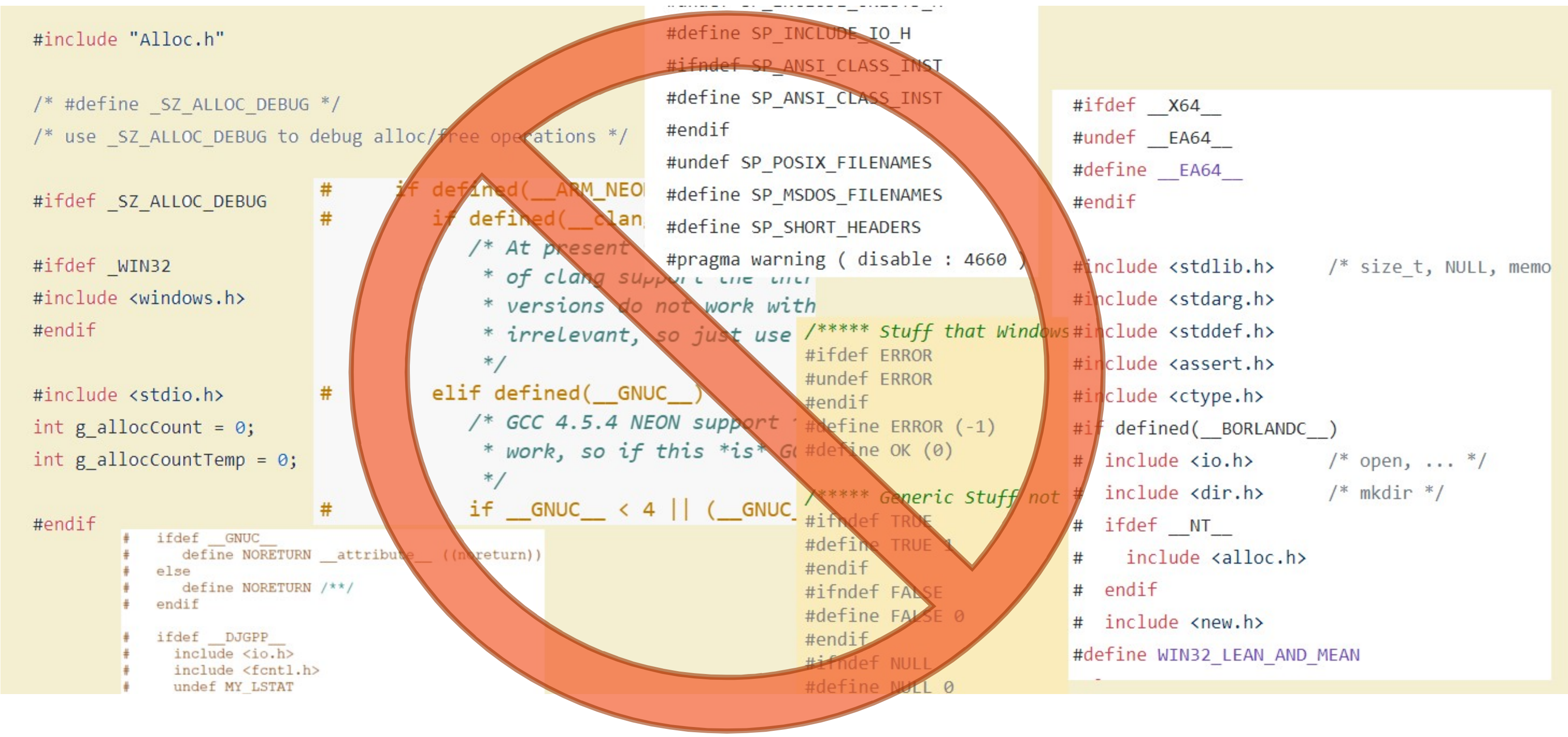
Fully open-source – no black-box

Fully in Ada – no interfacing headaches

Fully portable – no preprocessing, no conditionals

Portability – source level

One set of sources, zero conditionals.



```
#include "Alloc.h"

/* #define _SZ_ALLOC_DEBUG */
/* use _SZ_ALLOC_DEBUG to debug alloc/free operations */

#ifdef _SZ_ALLOC_DEBUG

#ifdef _WIN32
#include <windows.h>
#endif

#include <stdio.h>
int g_allocCount = 0;
int g_allocCountTemp = 0;

#endif
```

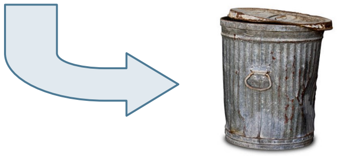
```
#       if defined(__ARM_NEON
#         if defined(__clang
            /* At present
             * of clang support the intr
             * versions do not work with
             * irrelevant, so just use
             */
#         elif defined(__GNUC__)
            /* GCC 4.5.4 NEON support
             * work, so if this *is* G
             */
#           if __GNUC__ < 4 || (__GNUC_
```

```
#   ifdef __GNUC__
#      define NORETURN __attribute__ ((noreturn))
#   else
#      define NORETURN /**/
#   endif

#   ifdef __DJGPP__
#     include <io.h>
#     include <fcntl.h>
#     undef MY_LSTAT
```

```
#define SP_INCLUDE_IO_H
#ifndef SP_ANSI_CLASS_INST
#define SP_ANSI_CLASS_INST
#endif
#undef SP_POSIX_FILENAMES
#define SP_MSDOS_FILENAMES
#define SP_SHORT_HEADERS
#pragma warning ( disable : 4660 )
```

```
/***** Stuff that Windows
#ifdef ERROR
#undef ERROR
#endif
#define ERROR (-1)
#define OK (0)

/***** Generic Stuff not
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#ifndef NULL
#define NULL 0
```

```
#ifdef __X64__
#undef __EA64__
#define __EA64__
#endif

#include <stdlib.h>     /* size_t, NULL, memo
#include <stdarg.h>
#include <stddef.h>
#include <assert.h>
#include <ctype.h>
#if defined(__BORLANDC__)
#  include <io.h>       /* open, ... */
#  include <dir.h>      /* mkdir */
#  ifdef __NT__
#    include <alloc.h>
#  endif
#  include <new.h>
#define WIN32_LEAN_AND_MEAN
```



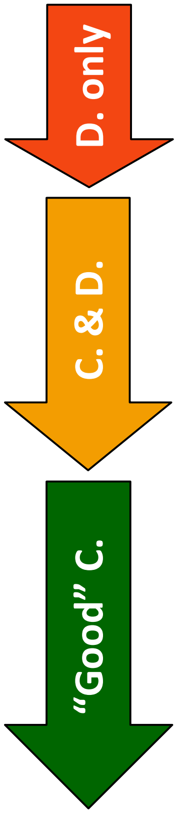
Portability – external components: none

- **No dependency** (on other Ada packages, nor on external libraries)
- No interfacing needed
- No worries with linker formats, 32 vs. 64 bit, etc.

- Same Ada toolset for monitoring processing flow and memory footprint.
- Ada streams and exceptions.

OpenVMS	Intel Itanium (64 bit)	GNU - **GNAT**
AIX	Power7 (64 bit)	
MS Windows 9x;NT,2K,XP+	Intel x86 (32 bit)	
MS Windows x64	Intel x64 (64 bit)	
Linux	Intel x86 (32 bit)	
Linux	Intel x86_64 (64 bit)	
Linux on PS3	Cell (64 bit)	
Linux on Raspberry Pi	ARM	
Mac OS X	PowerPC (64 bit)	
Mac OS X	Intel x64 (64 bit)	
Solaris	SPARC (32 or 64 bit)	
Solaris	Intel x64 (64 bit)	
OpenBSD	(one of several)	
FreeBSD	Intel x86 (32 bit)	
FreeBSD	Intel x64 (64 bit)	
Android 2.3+	ARM	
MS Windows x64	Intel x64 (64 bit)	PTC - **ObjectAda**
MS Windows NT+	Intel x86 (32 bit)	SofCheck - **AdaMagic**
MS Windows NT+	Intel x64 (64 bit)	
Linux	Intel x86 (32 bit)	
Mac OS X	PowerPC (64 bit)	
Mac OS X	Intel x64 (64 bit)	
Solaris	SPARC (32 or 64 bit)	
Solaris	Intel x64 (64 bit)	

Overview – milestones


D. only

C. & D.

“Good” C.

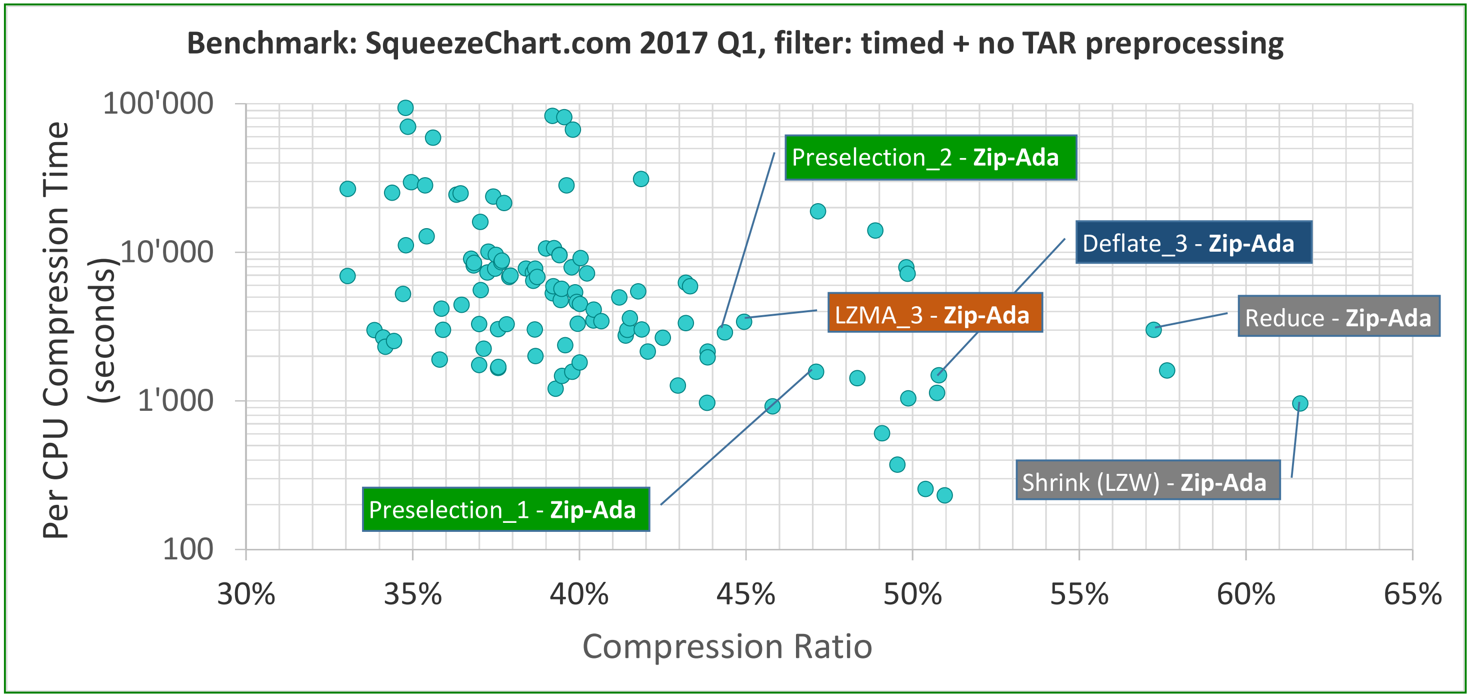
- **1999**: started with **Decompression** only
- **2007**: SourceForge hosting, **http://unzip-ada.sf.net/**
- **2007**: added **1st Compression** method (Shrink / LZW)
- **2008**: streams support, Zip.Create (**contrib.** NXP semiconductors)
- **2009**: added BZip2 decompression (*)
- **2010**: profiling, stream performance, UTF-8 (**contrib.** Romans CAD)
- **2011**: developed a simple Deflate method for Compression
- **2014**: added LZMA Decompression (*) from reference decoder
- **2016**: developed an advanced **Deflate** Compression method
- **2016**: developed a **LZMA** Compression method (*)
- **2018**: Trained Compression package (standalone)

(*) can be used standalone

Comparison. What is a “good” compression in general ?

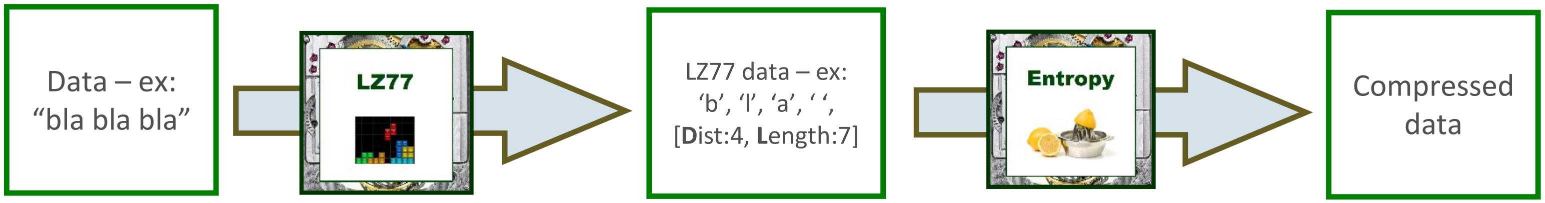
No easy answer: there are multiple criteria!

1. **Compression ratio** (compressed size / uncompressed size)
2. Per-CPU **Decompression time** (dep.: format and compressed size)
3. Per-CPU **Compression time** (dep.: format symmetry, algorithm, effort)
4. **Memory footprint** (Decompression only, or both Comp. & Decomp.)



Deflate & LZMA formats – Zip-Ada implementations – 2016

- In two phases: combines **LZ77** (font-end) and **Huffman trees** or **range encoding** (entropy back-end).



Data – ex:
“bla bla bla”

LZ77

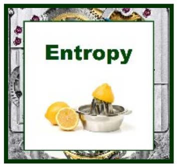
LZ77 data – ex:
‘b’, ‘l’, ‘a’, ‘ ‘,
[**D**ist:4, **L**ength:7]

Entropy

Compressed
data

- **Deflate** is multi-block with compression structure header for each block; not adaptive within a block
- **LZMA** (1st version) is single-block but adaptive (probability model adapted continuously with the stream)

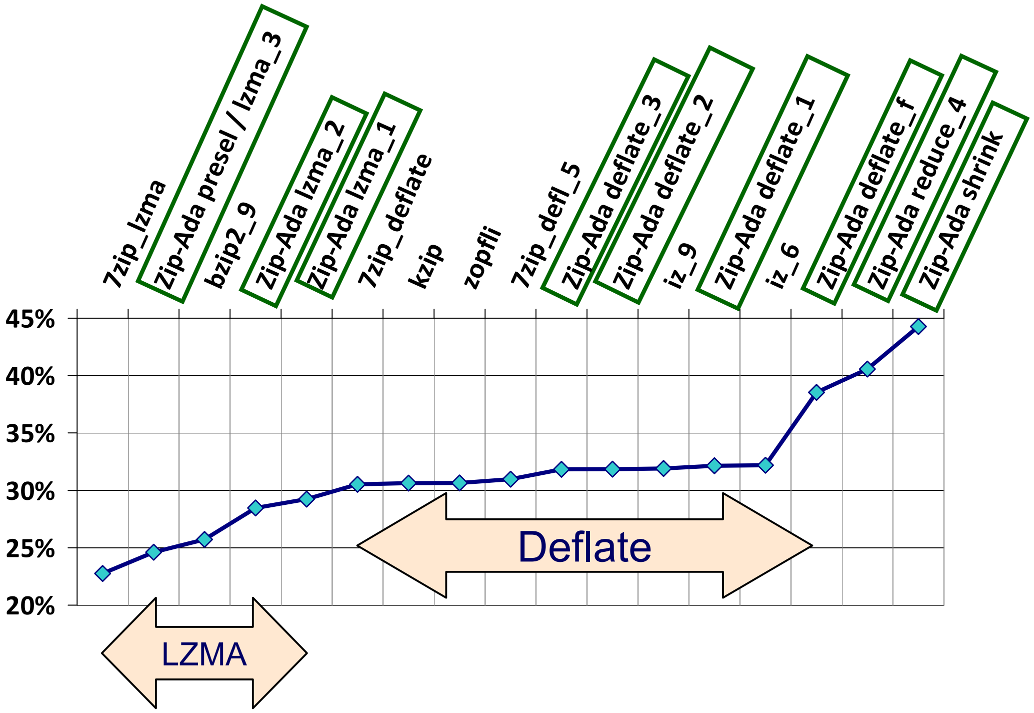
Deflate & LZMA formats – Zip-Ada implementations – 2016



- Generic collection of translated string matchers (BTW: can be used standalone), including Info-Zip/zlib implementation – the latter, very fast, is used for our Deflate **and** LZMA_1 and LZMA_2 methods.
- Entropy encoding programmed from scratch.

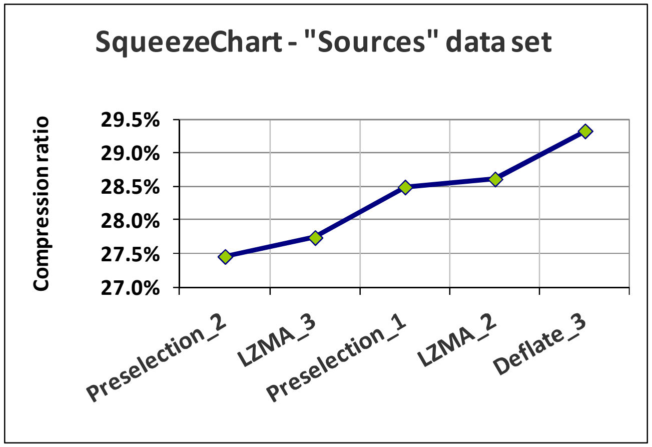
Silesia corpus			
Date / Size	% compr	Name	Deflate bench
48'240'494	22.8%	7zip_lzma	-25.44%
52'169'187	24.6%	presel / lzma_3	-19.37%
54'509'539	25.7%	bzip2_9	-15.75%
60'346'016	28.5%	lzma_2	-6.73%
61'970'916	29.2%	lzma_1	-4.22%
64'698'142	30.5%	**7zip_deflate**	**0.00%**
64'921'533	30.6%	kzip	+0.35%
64'949'384	30.6%	zopfli	+0.39%
65'636'076	31.0%	7zip_defl_5	+1.45%
67'462'614	31.8%	deflate_3	+4.27%
67'506'579	31.9%	deflate_2	+4.34%
67'634'472	31.9%	iz_9	+4.54%
68'110'939	32.1%	deflate_1	+5.27%
68'230'447	32.2%	iz_6	+5.46%
81'667'070	38.5%	deflate_f	+26.23%
85'991'264	40.6%	reduce_4	+32.91%
93'826'501	44.3%	shrink	+45.02%
211'938'580	100.0%	original data	

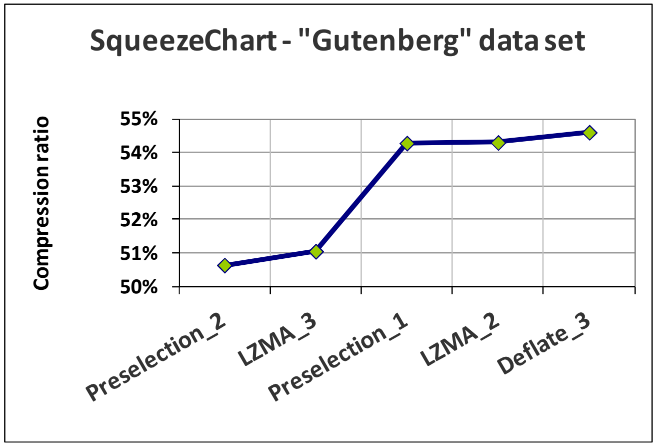
Green = Zip-Ada



“Preselection” algorithm-picking method for Zip archives

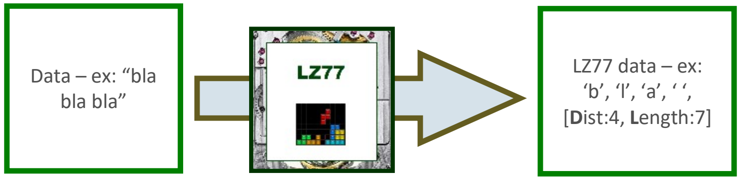
- Entries in Zip files are compressed individually
- **LZMA** is adaptive and needs some warm-up phase to have its large probability model adapted to data – it works better on large, homogeneous data.
- Indeed, **Deflate** usually beats LZMA on data smaller than 9000 bytes (empirical threshold).
- idea: select **Deflate** for small data streams, **LZMA** for large ones.


SqueezeChart - "Sources" data set


SqueezeChart - "Gutenberg" data set

Bonus: generic standalone LZ77 encoder (string matcher)

```ada
--  Standalone LZ77 compression (encoding) package.
---------------------------------------------------

with Interfaces;

package LZ77 is

  type Method_Type is (LZHuf, IZ_4, IZ_5, IZ_6, IZ_7, IZ_8, IZ_9, IZ_10, BT4);

  subtype Byte is Interfaces.Unsigned_8;

  generic
    Method: Method_Type;
    --  Input of data:
    with function  Read_byte return Byte;
    with function  More_bytes return Boolean;
    --  Output of LZ-compressed data:
    with procedure Write_literal (b: Byte);
    with procedure Write_DL_code (distance, length: Integer);
  procedure Encode;

end LZ77;
```


Data – ex: “bla bla bla”

LZ77 data – ex: ‘b’, ‘l’, ‘a’, ‘ ‘, [**D**ist:4, **L**ength:7]

Bonus: generic standalone LZMA encoder

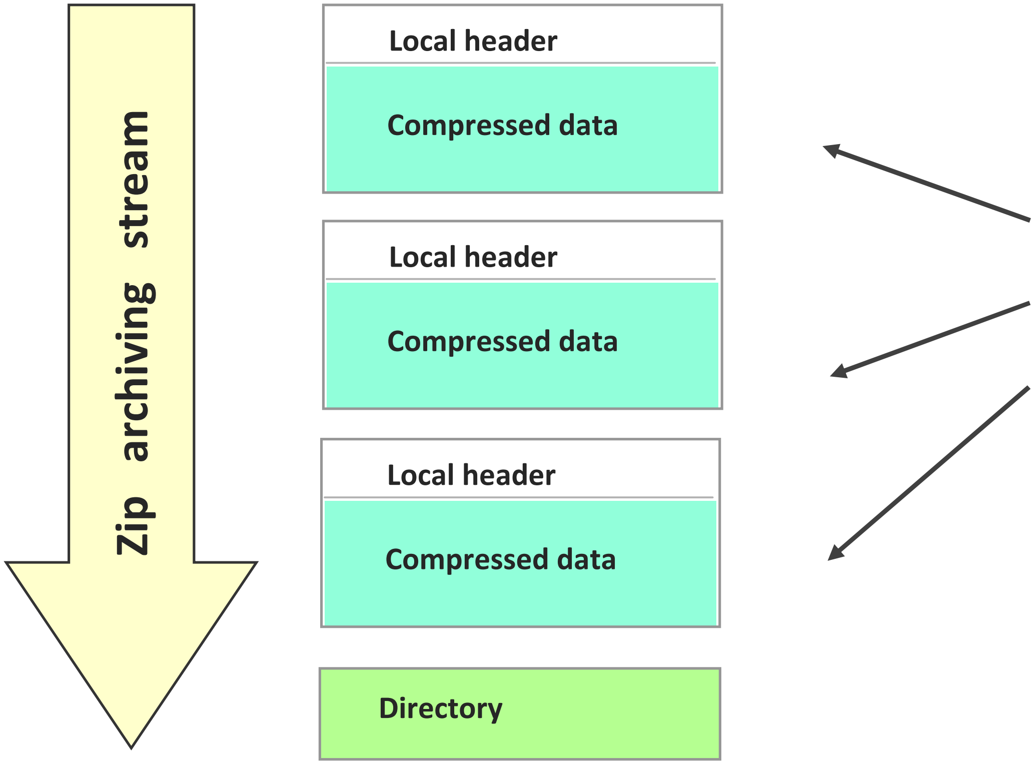
```
package LZMA.Encoding is

  type Compression_level is (Level_0, Level_1, Level_2, Level_3);

  generic
    --  Input of data:
    with function  Read_Byte return Byte;
    with function  More_Bytes return Boolean;
    --  Output of LZMA-compressed data:
    with procedure Write_Byte (b: Byte);
    --
  procedure Encode(
    ... --  [ parameters with default values ]
  );

end LZMA.Encoding;
```

Bonus: generic standalone LZMA decoder

```
generic
  --  Input:
  with function Read_Byte return Byte;
  --  Output:
  with procedure Write_Byte (b: Byte);

package LZMA.Decoding is

  type LZMA_Hints is record
    ...
  end record;

  procedure Decompress(hints: LZMA_Hints);

  ...

end LZMA.Decoding;
```

Annex – The Zip archive format

- Origin: Phil Katz's PKZIP (~ 1989) – old, limited… but used everywhere.
- **Multi-file** data archive container format with compression.
- **Open** regarding compression formats: Store, LZW, Deflate, LZMA, …



Individual compression formats: Zip-Ada takes advantage of this with "Preselection" methods.

References

1. **Zip-Ada** web site **http://unzip-ada.sf.net/**

2. **AZip** web site **http://azip.sf.net/** (AZip is a GUI archive manager using Zip-Ada)

3. **Squeeze Chart:** large and varied corpus: 5 GB; 21,532 files; web site: **http://www.squeezechart.com/**

4. [Deflate] **A fast and space-economical algorithm for length-limited coding**, Katajainen J., Moffat A., Turpin A. (1995), Lecture Notes in Computer Science, vol 1004. Springer, Berlin, Heidelberg

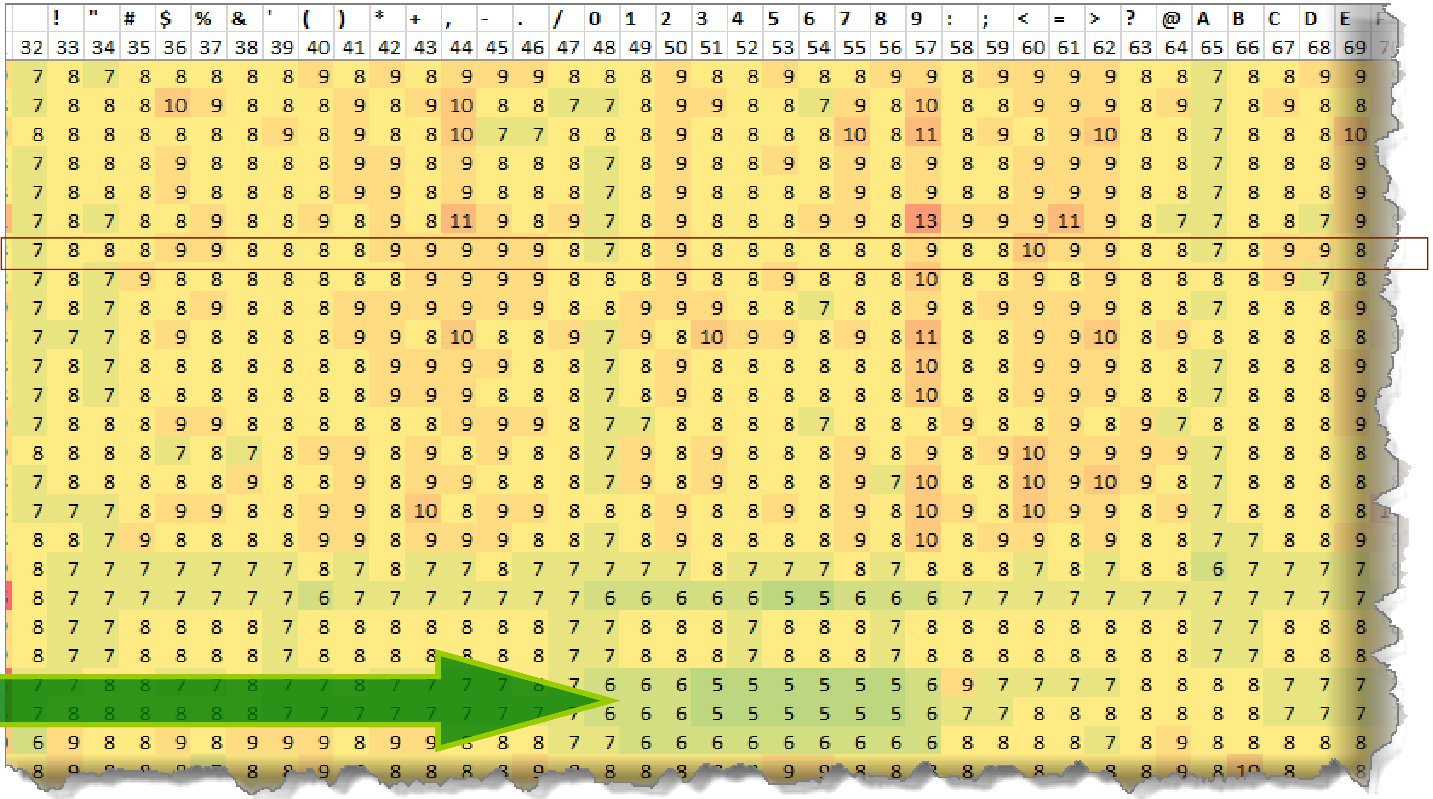
5. **DEFLATE Compressed Data Format Specification version 1.3**, P. Deutsch, 1996, **https://www.ietf.org/rfc/rfc1951.txt**

6. [LZMA] **Range encoding: an algorithm for removing redundancy from a digitized message**, G. N. N. Martin, Video & Data Recording Conference, Southampton, UK, July 24-27, 1979.

7. **Zip file format specification: https://support.pkware.com/display/PKZIP/APPNOTE**

Annex – The Deflate format – Taillaule algorithm

Huffman trees are uniquely determined by the set of *bit lengths* (the travel from root to leaf for each symbol). Consequently, only *bit lengths* are stored as compression structures. Our single-pass algorithm detects changes in the data stream “on the fly” by comparing *bit length* vectors. L1 norm seems the best, so far.



	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
7	8	7	8	8	8	8	8	9	8	9	8	9	9	9	8	8	8	9	8	8	9	8	8	9	9	8	9	9	9	9	8	8	7	8	8	9	9
7	8	8	8	10	9	8	8	8	9	8	9	10	8	8	7	7	8	9	9	8	8	7	9	8	10	8	8	9	9	9	8	9	7	8	9	8	8
8	8	8	8	8	8	8	9	8	9	8	8	10	7	7	8	8	8	9	8	8	8	8	10	8	11	8	9	8	9	10	8	8	7	8	8	8	10
7	8	8	8	9	8	8	8	8	9	9	8	9	8	8	8	7	8	9	8	8	9	8	9	8	9	8	8	9	9	9	8	8	7	8	8	8	9
7	8	8	8	9	8	8	8	8	9	9	8	9	8	8	8	7	8	9	8	8	8	8	9	8	9	8	8	9	9	9	8	8	7	8	8	8	9
7	8	7	8	8	9	8	8	9	8	9	8	11	9	8	9	7	8	9	8	8	8	9	9	8	13	9	9	9	11	9	8	7	7	8	8	7	9
7	8	8	8	9	9	8	8	8	8	9	9	9	9	9	8	7	8	9	8	8	8	8	8	8	9	8	8	10	9	9	8	8	7	8	9	9	8
7	8	7	9	8	8	8	8	8	8	8	9	9	9	9	8	8	8	9	8	8	9	8	8	8	10	8	8	9	8	9	8	8	8	8	9	7	8
7	8	7	8	8	9	8	8	8	9	9	9	9	9	9	8	8	9	9	9	8	8	7	8	8	9	8	9	9	9	9	8	8	7	8	8	8	9
7	7	7	8	9	8	8	8	8	9	9	8	10	8	8	9	7	9	8	10	9	9	8	9	8	11	8	8	9	9	10	8	9	8	8	8	8	8
7	8	7	8	8	8	8	8	8	8	9	9	9	9	8	8	7	8	9	8	8	8	8	8	8	10	8	8	9	9	9	8	8	7	8	8	8	9
7	8	7	8	8	8	8	8	8	8	9	9	9	8	8	8	7	8	9	8	8	8	8	8	8	10	8	8	9	9	9	8	8	7	8	8	8	9
7	8	8	8	9	9	8	8	8	8	8	8	9	9	9	8	7	7	8	8	8	8	7	8	8	8	9	8	8	9	8	9	7	8	8	8	8	9
8	8	8	8	7	8	7	8	9	9	8	9	8	9	8	8	7	9	8	9	8	8	8	9	8	9	8	9	10	9	9	9	9	7	8	8	8	8
7	8	8	8	8	8	9	8	8	9	8	9	9	8	8	8	7	9	8	9	8	8	8	9	7	10	8	8	10	9	10	9	8	7	8	8	8	8
7	7	7	8	9	9	8	8	9	9	8	10	8	9	9	8	8	8	9	8	8	9	8	9	8	10	9	8	10	9	9	8	9	7	8	8	8	8
8	8	7	9	8	8	8	8	9	9	8	9	9	9	8	8	7	8	9	8	8	8	8	9	8	10	8	9	9	8	9	8	8	7	7	8	8	9
8	7	7	7	7	7	7	7	8	7	8	7	7	8	7	7	7	7	7	8	7	7	7	8	7	8	8	8	7	8	7	8	8	6	7	7	7	7
8	7	7	7	7	7	7	7	6	7	7	7	7	7	7	7	6	6	6	6	6	5	5	6	6	6	7	7	7	7	7	7	7	7	7	7	7	7
8	7	7	8	8	8	8	7	8	8	8	8	8	8	8	7	7	8	8	8	7	8	8	8	7	8	8	8	8	8	8	8	8	7	7	8	8	8
8	7	7	8	8	8	8	7	8	8	8	8	8	8	8	7	7	8	8	8	7	8	8	8	7	8	8	8	8	8	8	8	8	7	7	8	8	8
7	7	8	8	7	7	8	7	7	8	7	7	7	7	8	7	6	6	6	5	5	5	5	5	5	6	9	7	7	7	7	8	8	8	8	7	7	7
7	8	8	8	8	8	8	7	7	7	7	7	7	7	7	7	6	6	6	5	5	5	5	5	5	6	7	7	8	8	8	8	8	8	8	7	7	7
6	9	8	8	9	8	9	9	9	8	9	9	8	8	8	7	7	6	6	6	6	6	6	6	6	6	8	8	8	8	7	8	9	8	8	8	8	8
8	9	[illegible]	[illegible]	[illegible]	[illegible]	8	8	9	[illegible]	8	8	8	8	9	[illegible]	8	8	8	[illegible]	[illegible]	9	9	8	8	[illegible]	8	[illegible]	8	[illegible]	8	8	9	8	10	8	[illegible]	8

LZ77 symbols

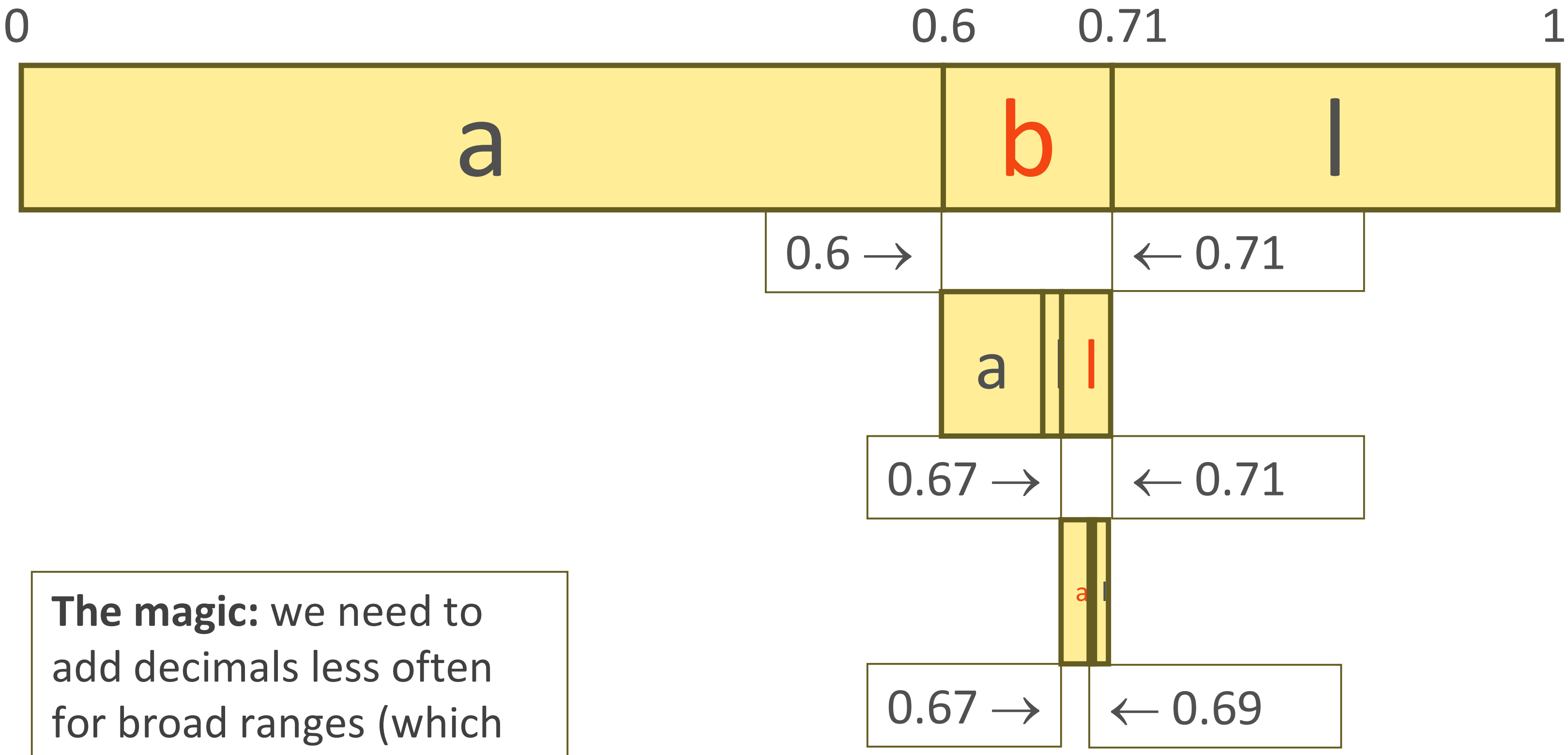
Optimal bit lengths based on statistics of chunks of LZ77 data.

1 row = 1 vector of bit lengths for a data chunk

Frequent symbol
⇒ Low bit length
⇒ Better compression

Range encoding

Example with a restricted alphabet: a, b, l. Widths are proportional to average frequencies in the English language. We want to encode “bla”.

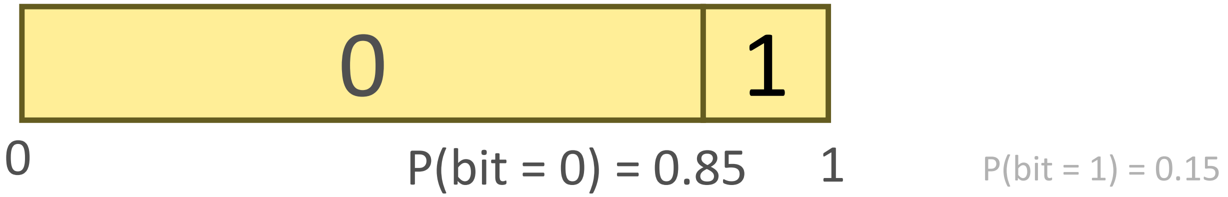


The magic: we need to add decimals less often for broad ranges (which are more probable): “aaaa” = [0, 0.1296]

Then: “bla” = [0.67, 0.69]

Range encoding *in LZMA*

- Only 0’s and 1’s. The interval may be *expanded* after a bit output.



- Many **contextual probability sets** used. Here, for literals:
  - **Previous bits in a byte** (end effect: 256 subintervals, one prob. for each byte value)
  - **Value of previous byte** (Markov predictor)
  - **Position of the byte** modulo up to 16 (good for structured data or Unicode text)
  - ⇒ each bit uses *one* of **8,388,608** probabilities (max configuration) !
- Default, neutral probability is 0.5, then adapted with a factor (~1.03) on each output.
- Max probability ~0.985: in the best case, compressed output is **~0.03 bit per uncompressed bit** – that, only the for “MA” part, it is on top of the “LZ” compression !