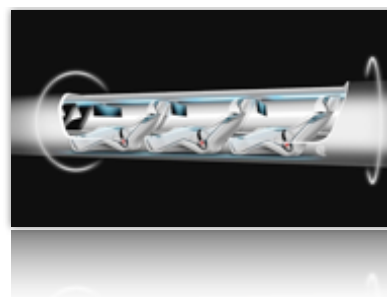
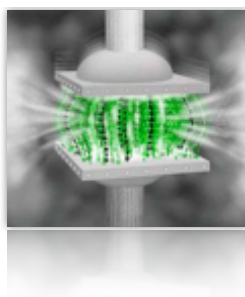


LZ4HC COMPRESSION

for ROOT and
IO Baseline Evaluation

ROOT IO Workshop - 6.12.2013

Andreas-Joachim Peters
IT-DSS-TD



=

compression x

speed x

size

• Overview of Compression Algorithms

- baseline for expectation







• Implementation

- format for multithreaded encoding and single-threaded decoding
- implementation in ROOT

• Results

- benchmarks for various Tree's
- IO baseline measurement

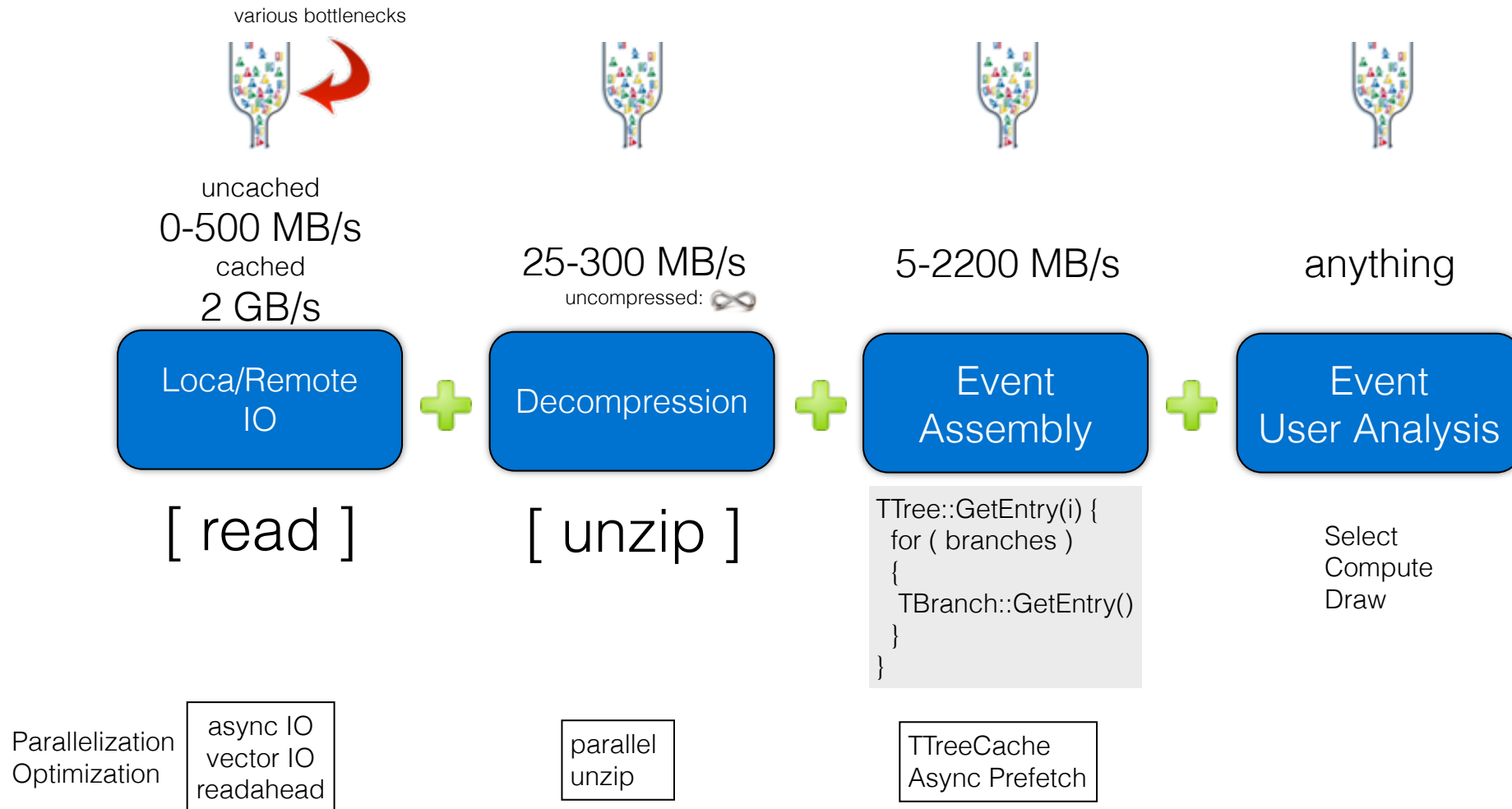
LZ4 is used by

- Operating Systems :  [Linux \(*\)](#),  [FreeBSD \(*\)](#),  [Illumos \(*\)](#)
- File Systems :  [OpenZFS \(*\)](#), [Hammer2 \(*\)](#), [LessFS \(*\)](#), [Grub](#)
- Big Data :  [Hadoop \(*\)](#),  [Cassandra \(*\)](#), [Rarelogic](#)
- Search Engine :  [Lucene \(*\)](#),  [Solr \(*\)](#)
- Database : [Hbase \(*\)](#), [Tokudb](#), [Delphix \(*\)](#), [infiniSQL \(*\)](#)
- Games :  [Battlefield 4 \(*\)](#),  [Sine Mora \(*\)](#),  [1000 Tiny Claws \(*\)](#)
- Graphics :  [Enlightenment](#),  [Xpra \(*\)](#), [Scaplib](#)
- Wave viewer :  [GTKWave](#)
- Caching : [Zram](#), [PHP Zend Optimizer](#)
- Backup : [Bareos \(*\)](#)

Core i5-3340M @2.7GHz, using the [Open-Source Benchmark by m^2 \(v0.14.2\)](#) compiled with GCC v4.6.1

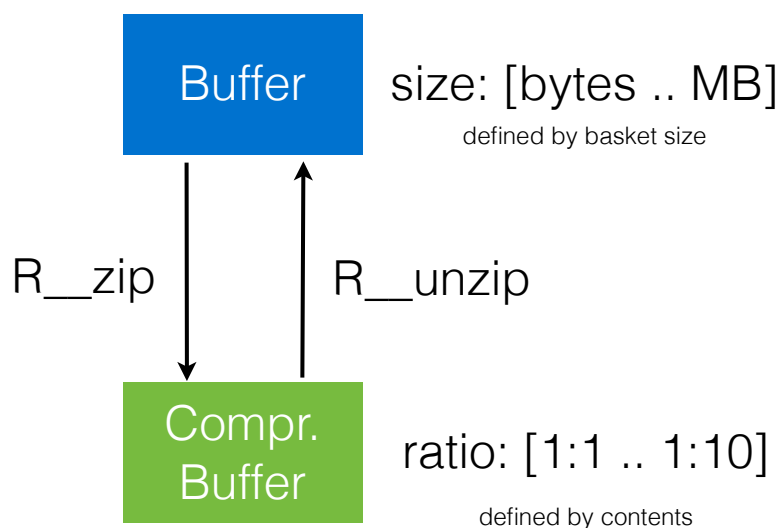
Algorithm	Compr. Ratio	Encoding Speed [MB/s]	Decoding Speed [MB/s]
LZ4	2.084	422	1820
Snappy	2.091	323	1070
ZLIB 1.2.8 level=1	2.730	65	280
ZLIB 1.2.8 level=6	3.099	21	300
LZ4HC	2.720	25	2080

Looking at this table: **LZ4HC** looks very **interesting** for workflows where $n(\text{read}) > n(\text{write})$... but the story is not at it's end ...

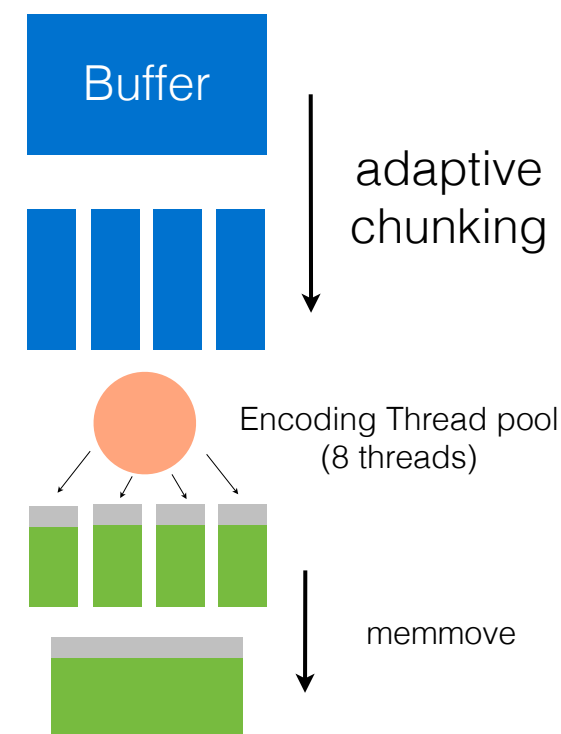


LZ4HC decoding is extremely fast [2 GB/s] and does not need parallelism while **encoding** is slow => try parallel approach with low code change impact in ROOT on lowest level (inside R_zip) ...

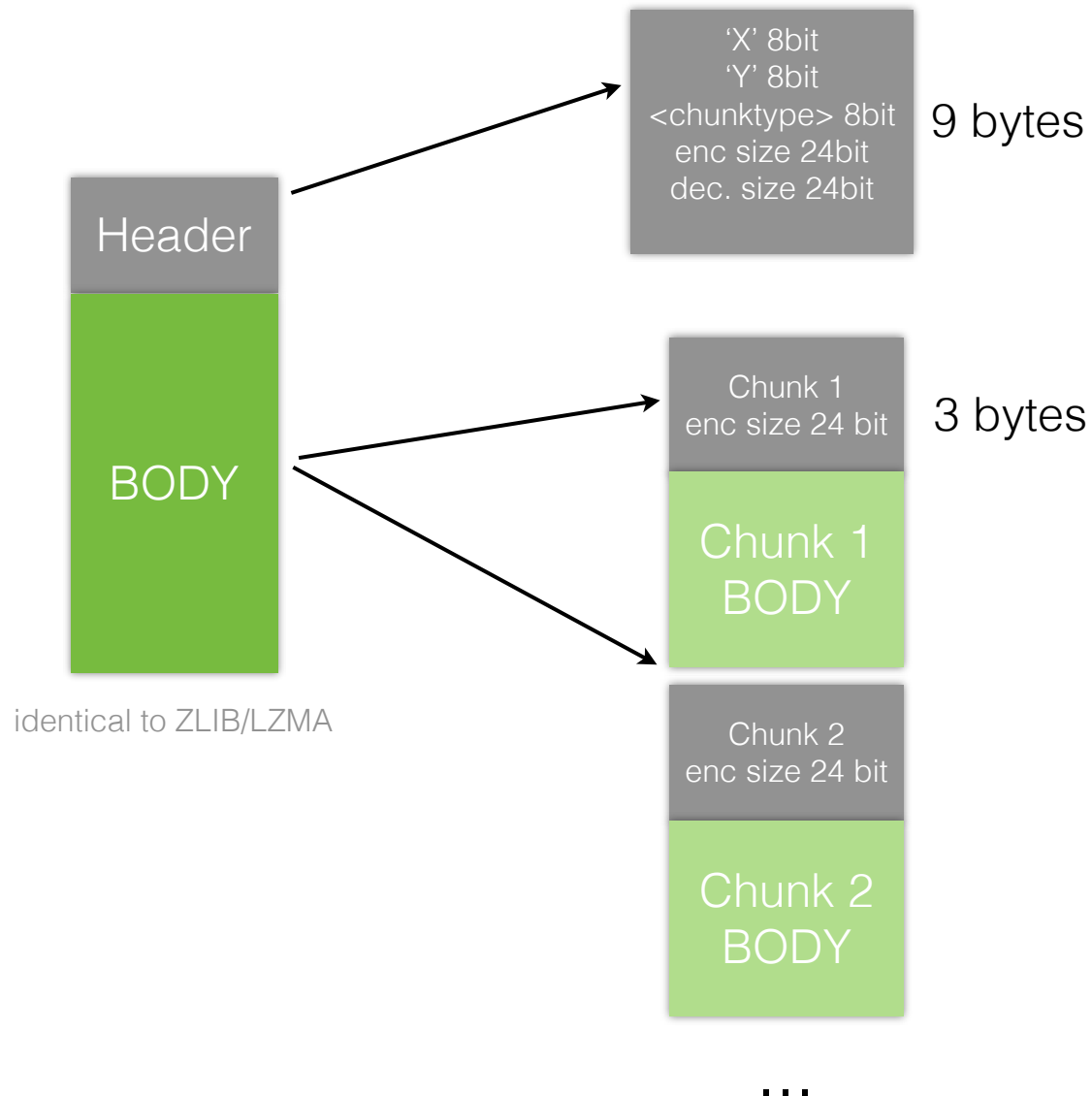
ZLIB/LZMA Compression in ROOT



LZ4(HC) Compression in ROOT



ROOT LZ4 COMPRESSION



<chunktype>

ID	Size
1	64k
2	128k
3	256k
4	512k
5	1M

ROOT compression is part of **libCore** which has no access to threading.

First prototype used *C++11 threads*, now using native ROOT threads + semaphores.

Single threaded encoding and decoding is implemented in class **ZipLZ4** under `root/core/lz4/in libCore`.

Multi threaded encoding is implemented in class **ZipLZ4mt** in `root/io/io/src/` installing a singleton pointer on load starting eight worker threads.


[the construction and destruction of the thread pool is currently tied to libRIO - to be reviewed]

Compression code is ~2.5k lines in C (6 files) from <https://code.google.com/p/lz4/> [LZ4 r108]

Javascript implementation available
<https://github.com/pierrec/node-lz4>

Event.root eventexe 100000 1

Basket-Size: 50kb - 2.5M ; 21 Leaves

Reference 

Compressor	Level	Ratio	Compression Speed	Decompression Speed
GZIP	1	2.15 L=1 2.27 L=6 2.35 L=9	21 10 1.9	88
LZ4	>1	1.68	52	178
LZ4HC	1	2.02	12	188
LZ4HC (mt)	1	2.02	32	188
LZMA	1	2.71	7	24
uncompressed	0	1.0	57	200



NTUPLE	File Size [b]	Default Compr. Type	Default Compr. Ratio	Branch /Leafs	#Events	Default Read (s)	LZ4HC Read [s]	LZ4HC IO rate [MB/s]	Size change	CPU usage change	IO rate change
ATLAS SUSY	4.5G	ZLIB	3.84	7K	55K	496s	445s	11.8	+17,0%	-10.5%	+28%
10%						138s	64s	82.2		-53.7%	+583%
ATLAS HIGGS	856M	ZLIB	2.47	5.8K	12K	62s	54s	17.8	+11,4%	-13%	+12%
10%						22s	8.8s	110		-60%	+130%
ALICE	230M	ZLIB	5.4	423	657	12.4s	9.1s	26.2	+10%	-26%	+45%
CMS Higgs Events	2.5 GB	ZLIB	5.04	305	1.4M	213s L=9 229s L=1	229s	13.3	+20 % L=9 +0 % L=1	+4% +0%	+13% +0%
CMS PHOTON	3 GB	ZLIB	4.21	5k	14K	570s			+21%	N.N	N.N
CMS USER NTUPLE	1.8G	ZLIB	2.6	14	81M	110s	90s	24	+22%	-18%	+50%
LHCB	1.5G	LZMA	3.0	76	232k	264s 190s zlib	119s	19.1	+50% +5% zlib	-55% -37% zlib	+230% +71%

For the CMS Photon file I missed class libraries to read it after conversion. Running **CloneTree** resulted in the ATLAS cases in 5-10% larger files using ZIP default compression [basket size optimization?] **CloneTree is incredibly slow!!!**

ROOT LZ4 COMPRESSION



LZ4HC compressed trees are **not always faster** to read than ZLIB. If overall network IO is a bottleneck LZ4HC is already ruled out.

It is not completely transparent to understand the different behavior of LZ4HC and ZLIB for the tested tree's: the usability of LZ4HC depends strongly on the input data.

In general it would be good to have a **fast conversion** function in ROOT re-compressing baskets (in parallel) with a different algorithm e.g. it takes 20 minutes to convert the Higgs Event tree (2.5 MB/s).

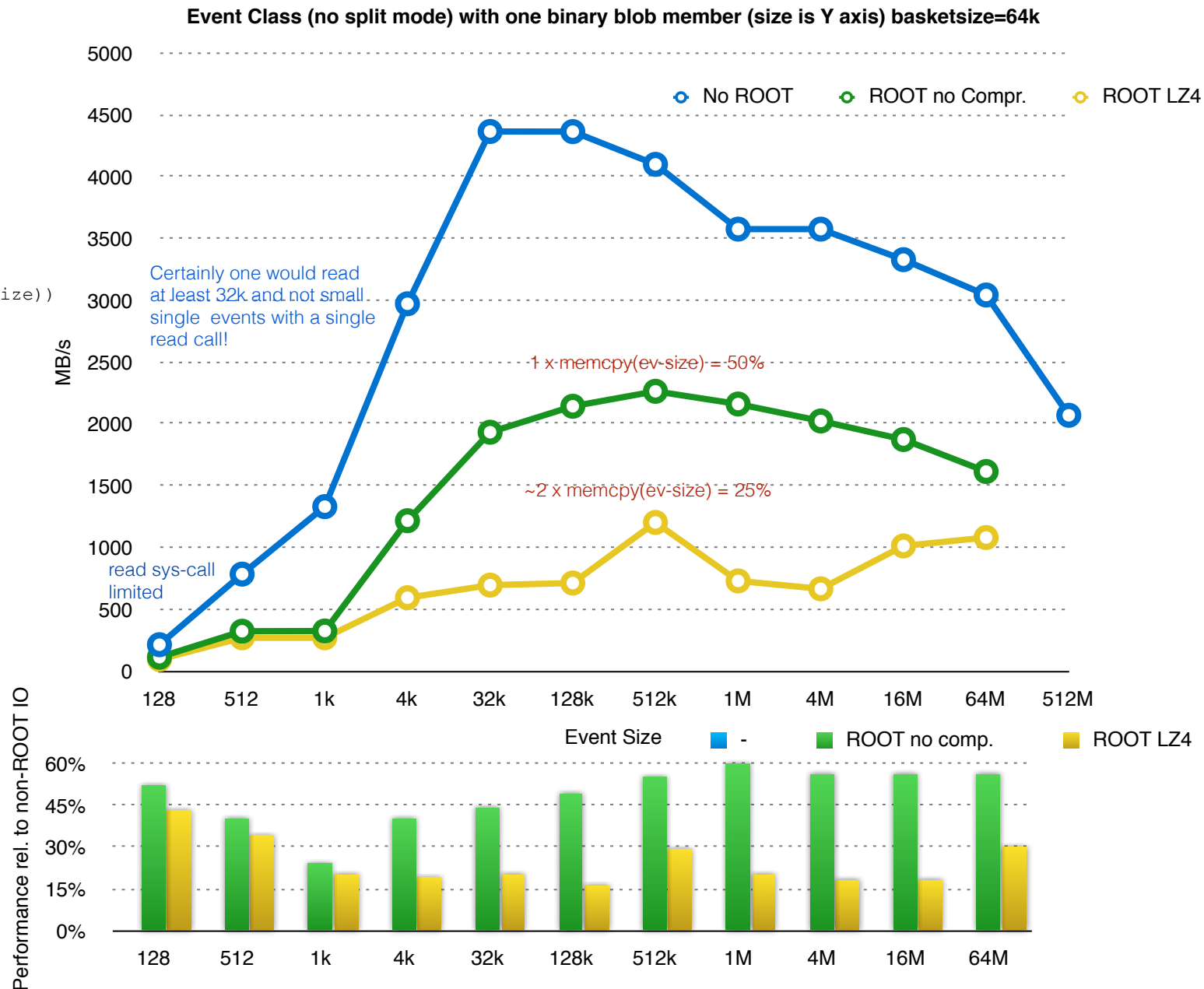
ROOT should implement a fast **benchmark probe function** showing the performance results for a subset of events in a given tree.

Events:

Events are filled with random bytes and compress 1:1.45

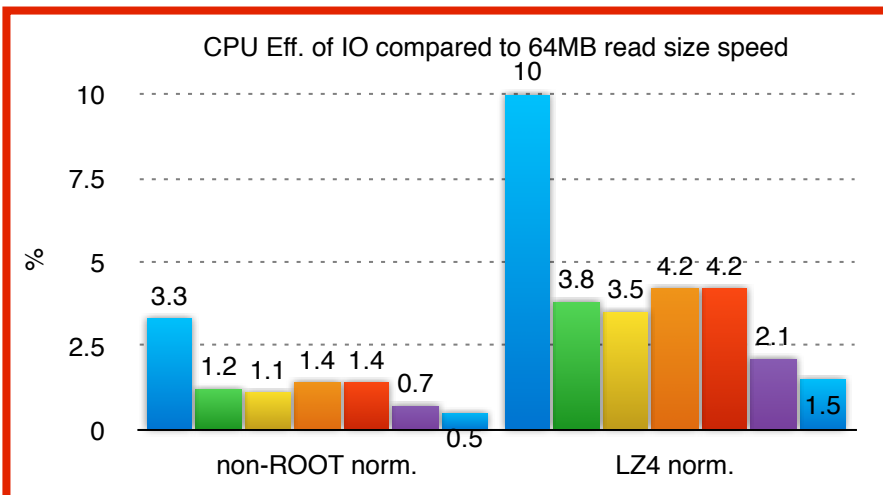
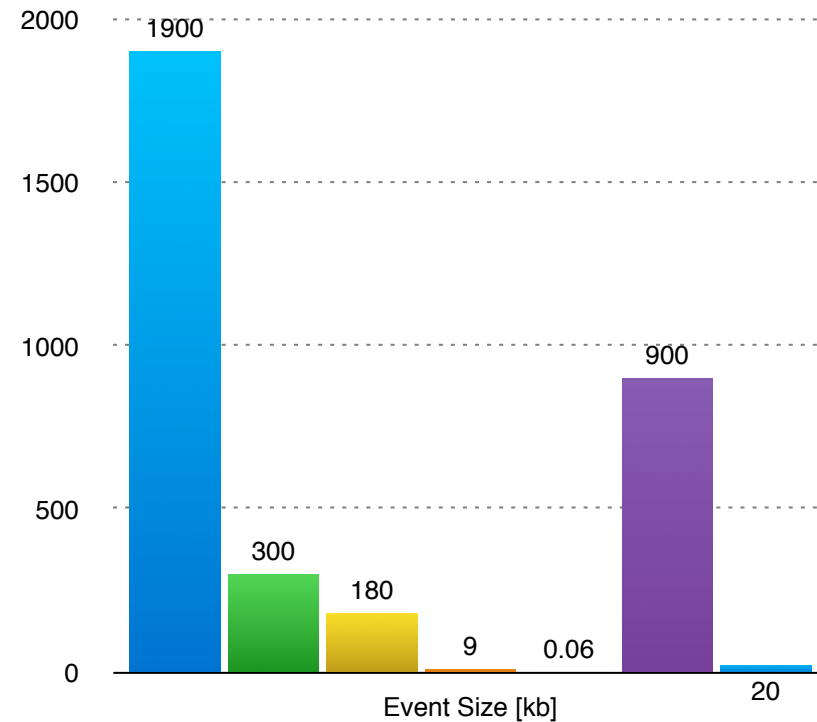
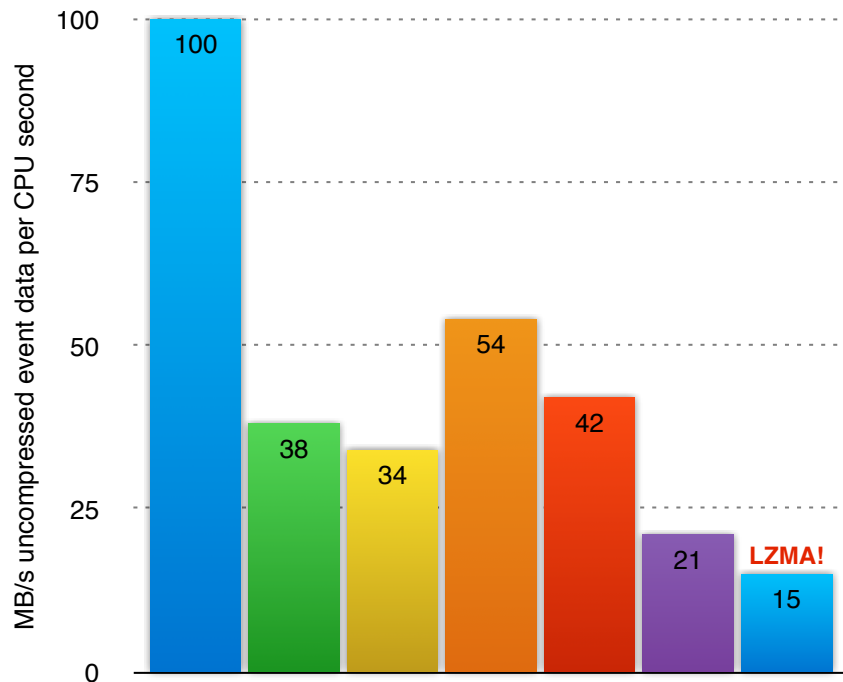
No-ROOT:

```
open (file)
while (read(ev-size))
close (file)
```



ALICE AOD ATLAS SUSY ATLAS HIGGS CMS HIGGS CMS USER CMS PHOTON LHC MDST

100% CPU Read IO



This is an area to invest more work. This inefficiency wastes CPU cycles to arrange data in memory not to analyze them => re-evaluate the cost of minimal data size and framework flexibility.

LZ4HC compression trades inferior compression for lower CPU usage.

It seems to be a good choice in use cases with **lower number** of **branches** or **partial** event **reads**.

Less compression and faster decompression results in higher bandwidth requirements to reach 100% CPU usage.

To gain from multithreaded LZ4HC **encoding** basket sizes must be at least of the order of several 64kb - otherwise multithreading does not result in faster compression.

It might be interesting to apply the **multithreaded encoding & decoding to LZMA** which gives the best compression at a low performance. It helps single client performance but certainly costs CPU.

With LZ4HC compression the **inefficiency in the event assembly** becomes more evident. This should be a **focus of the future** with the goal not just to parallelize it (using even more CPU) but **to reduce the CPU needed to assemble events** in memory (maybe never really convert them in C++ objects - just proxy).