Optimizing ROOT IO For Analysis

Brian Bockelman, Zhe Zhang, *University of Nebraska-Lincoln*
Jim Pivarski, *Princeton*

Why?

ROOT IO is an incredibly flexible format!

It is used for the storage of many petabytes of archival HEP experiment data; it can easily store the complex objects that correspond to the experiment's data models.

We propose the **analysis is a distinct IO use case** and to explore specialization.

What do users want?

Speed! Users may iterate across their data many times – and science can't proceed until the IO has finished.

What can they sacrifice?

Disk space: The input to a typical analysis can often fit on a single hard drive (an experiment's data may take up thousands).

Complex data: Analysis events are often drastically simplified when compared to full experiment frameworks.

Approach

We look for techniques that tradeoff modest sacrifices in disk space and event complexity for drastic increases in speed.

We explore alternate decompression techniques – particularly the **LZ4 algorithm** – that focus on read performance over compression ratio.

We propose a new TTree API that invokes the ROOT IO code *once per cluster* instead of *once per event*.

Branches, Baskets, Events, Clusters: Oh My!

- **Event**: The *event* is the atomic unit of work. Think rows in a table. Each event is composed of multiple objects.
- **Branch**: There are similar objects in each event – these are organized as *branches*. Think columns in a table.
- **Basket**: When serialized, ROOT writes (and compresses) the objects in the same branch – and from many contiguous events – into a *basket*.
- **Cluster**: All the data from a group of events is written contiguously as part of an event *cluster*.

Turbocharging ROOT with Bulk IO

What is Bulk IO?

Bulk IO is a set of techniques and APIs we developed for ROOT that allow the user to deserialize a large set of events at a time.

Why Bulk IO?

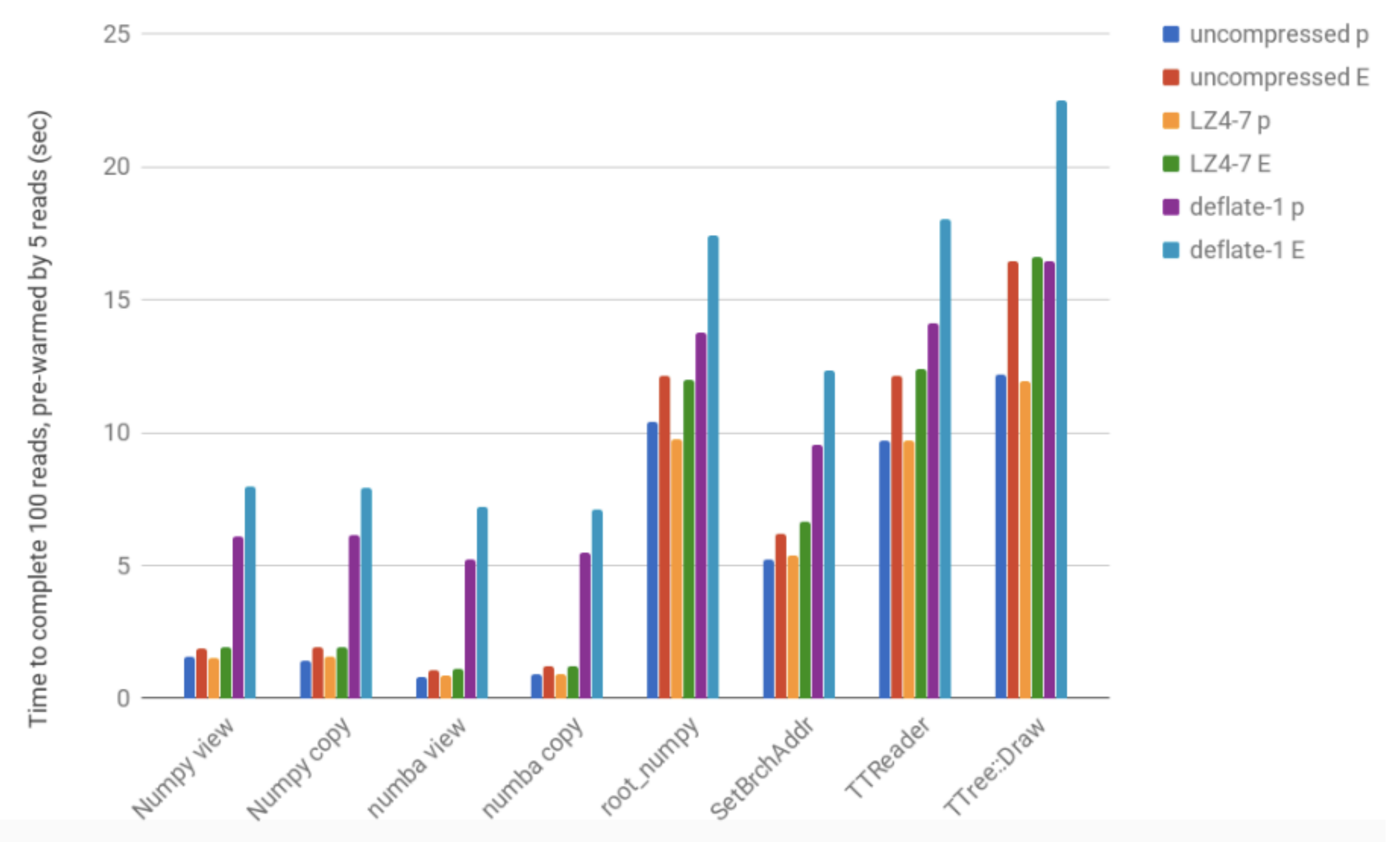
For small, simple events the overhead of ROOT library calls is much larger than the cost of serialization itself. By returning an entire basket of deserialized objects to the user, .

Further improvements can be achieved by returning serialized events to the user and allowing the compiler to inline deserialization in the event loop

Why Not Bulk IO?

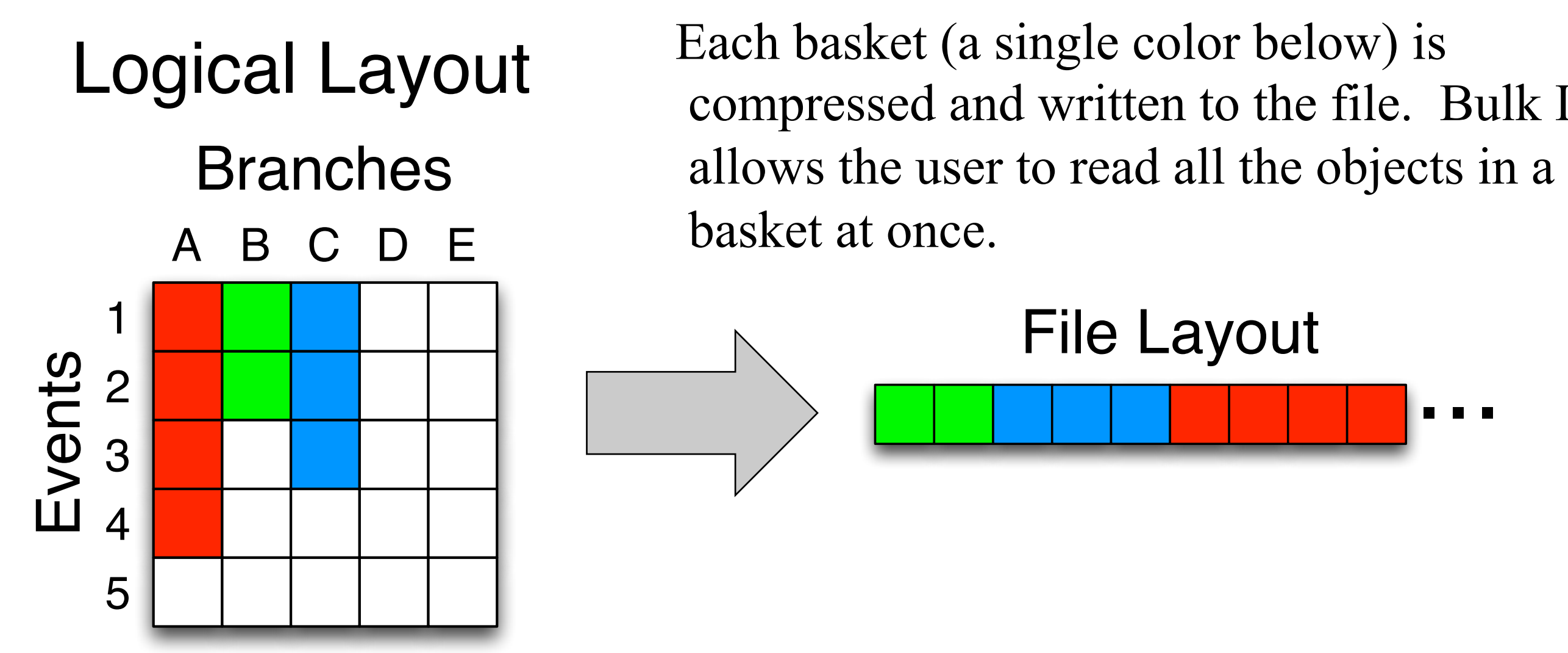
Complex objects involving references or from polymorphic classes require expensive lookups to deserialize. In these cases, the library overheads are minimal and bulk IO provides little benefit.

Micro benchmark shows order-magnitude speedup…



… with bulk IO consistently faster across access methods.

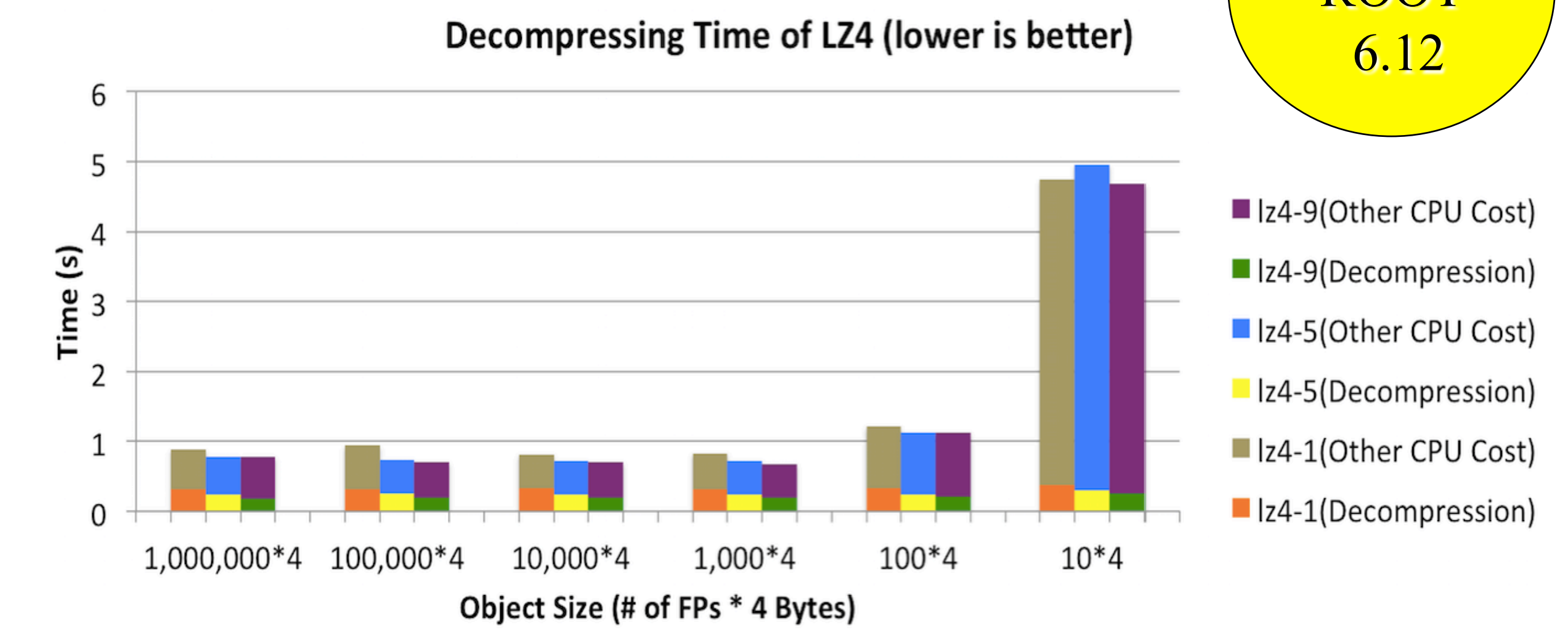
ROOT IO: An Illustration



Each basket (a single color below) is compressed and written to the file. Bulk IO allows the user to read all the objects in a basket at once.

LZ4: A New Approach for Compression

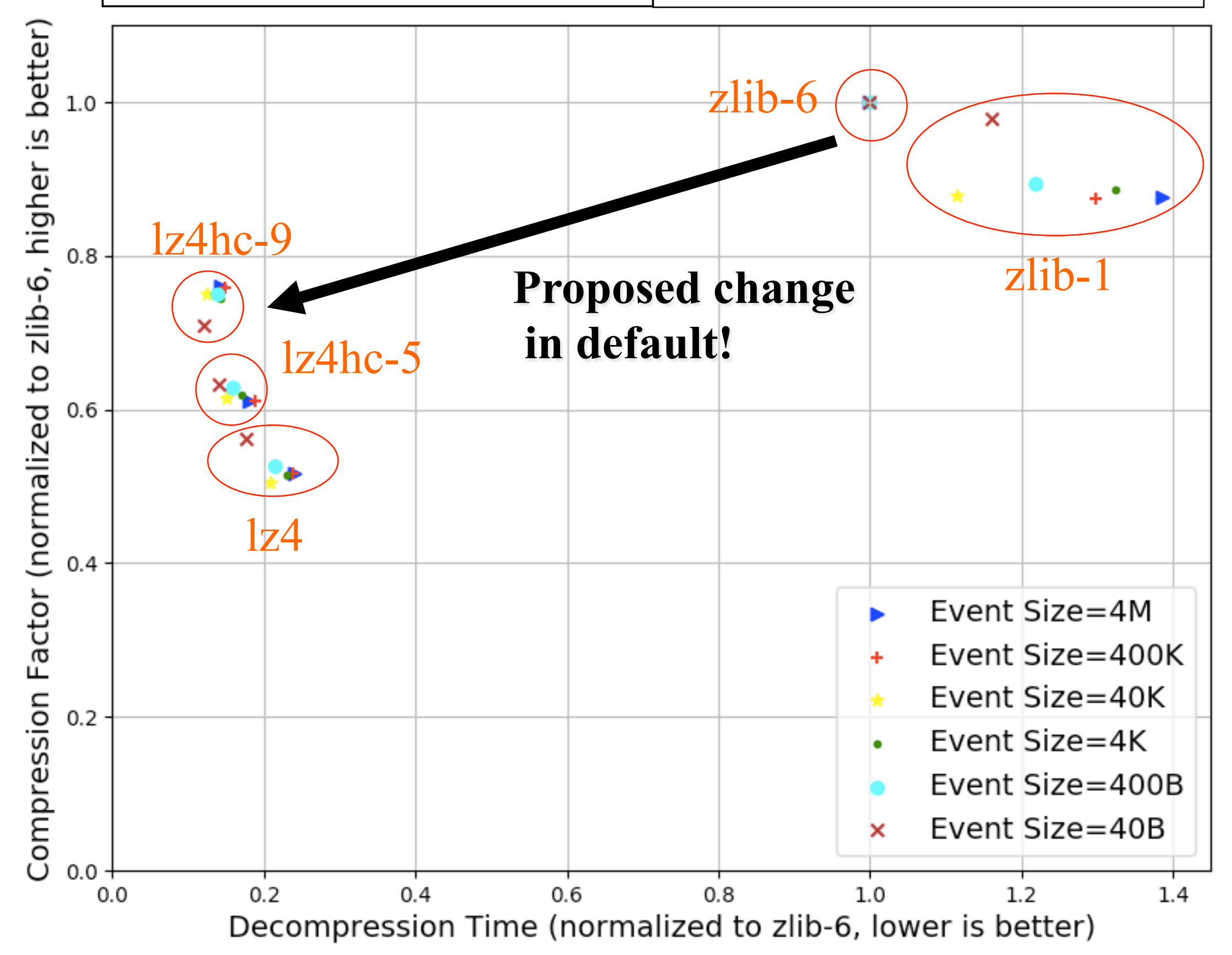
New! ROOT 6.12



Decompression contributes to around ¼ of total CPU cycles for events size between 4KB and 4MB. But as event size goes to 40 bytes, deserialization and other costs dominate.

Depending on the event content, with LZ4, we see:

95%	15%
performance of reading **uncompressed** data	Larger files compared to **zlib-based** compression



But That's Not All!

- Analysis can be made faster by using more cores! See the Track 1 presentation "**Increasing Parallelism in the ROOT I/O subsystem**" at 17:20 on Thursday.
- Join the weekly ROOT IO meetings to help this effort move forward!

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1450323. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.