# ROOT IO Update

Brian Bockelman

DIANA Meeting, 15 October 2017

# Goals for Today

- Summary last week's ROOT IO Workshop (includes contributions from inside and outside DIANA).
    - Compression progress.
    - Bulk IO.
    - Parallel file merging: TDataFrame and CMS progress.
- Outline of goals for the next year:
    - Improved compression.
    - Targets for 6.12.
- **Note**: skipping around a bit in the presentations, including some more forward-looking content.

**https://indico.fnal.gov/event/15154/**

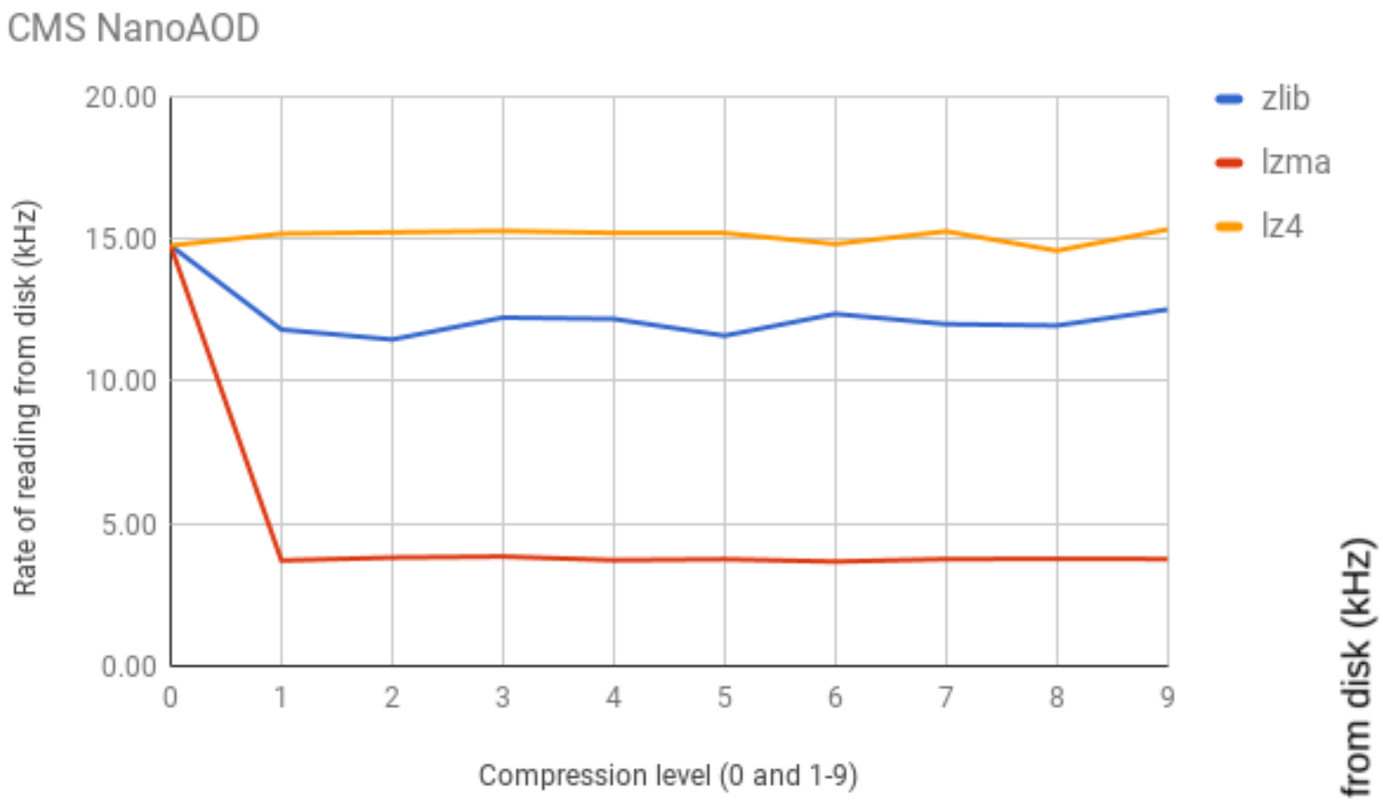
# Compression

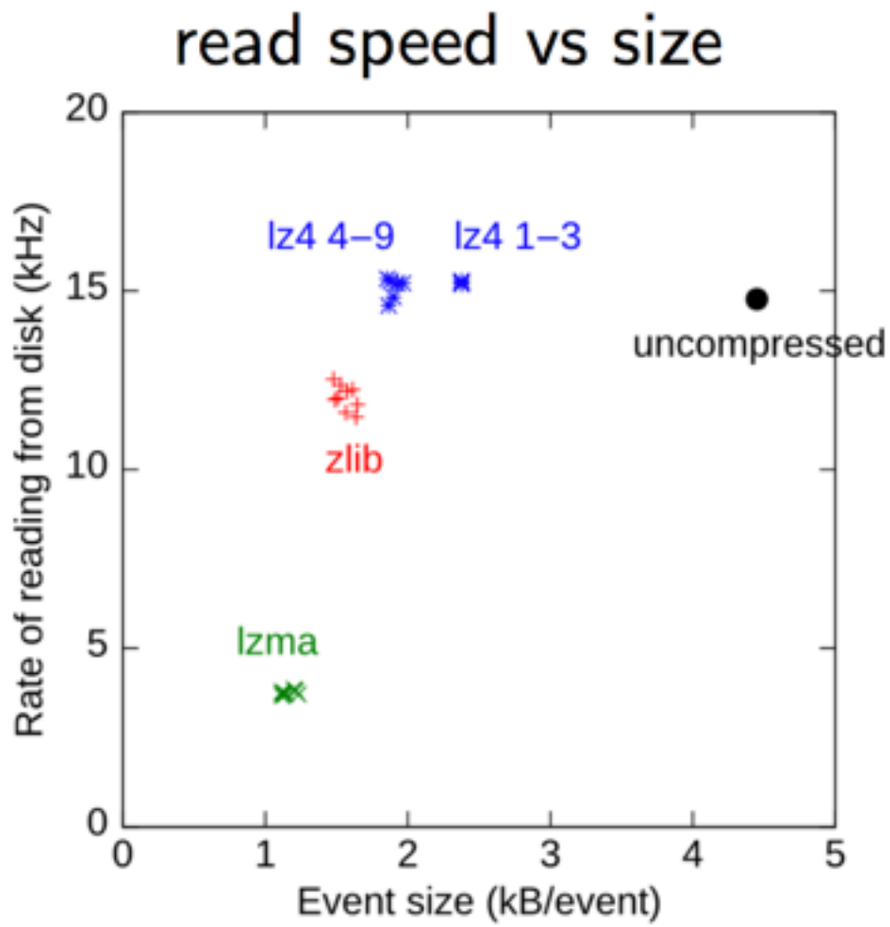
- We have been working to add support for LZ4 to ROOT IO — and backport to all active release branches.
    - Fixup test cases, build scripts, backport xxhash etc.
- Continuing our work to improve default zlib performance.
    - Patches from CloudFlare are 4x faster for compression for zlib-9 but only worked on x86-64 / new-ish processors.  David A started - and Oksana Shadura continued - work to making this usable on a wide variety of platforms.
    - Interesting finding: there are 2-3 versions of zlib in ROOT.  The one used at any given time varies based on how the user utilizes ROOT.
    - Prior to merging “CloudFlare patches”, working to either get ROOT to *one* version of zlib — or at least making sure the same one is used consistently.

# Compression - LZ4

**https://indico.fnal.gov/event/15154/contribution/8/material/slides/0.pdf**

More importantly: reading is as fast as uncompressed




- **LZ4 performs similarly to uncompressed data.**
- **LZ4 write speed similar to current ROOT default.**
- **LZ4 file size ~15% larger**

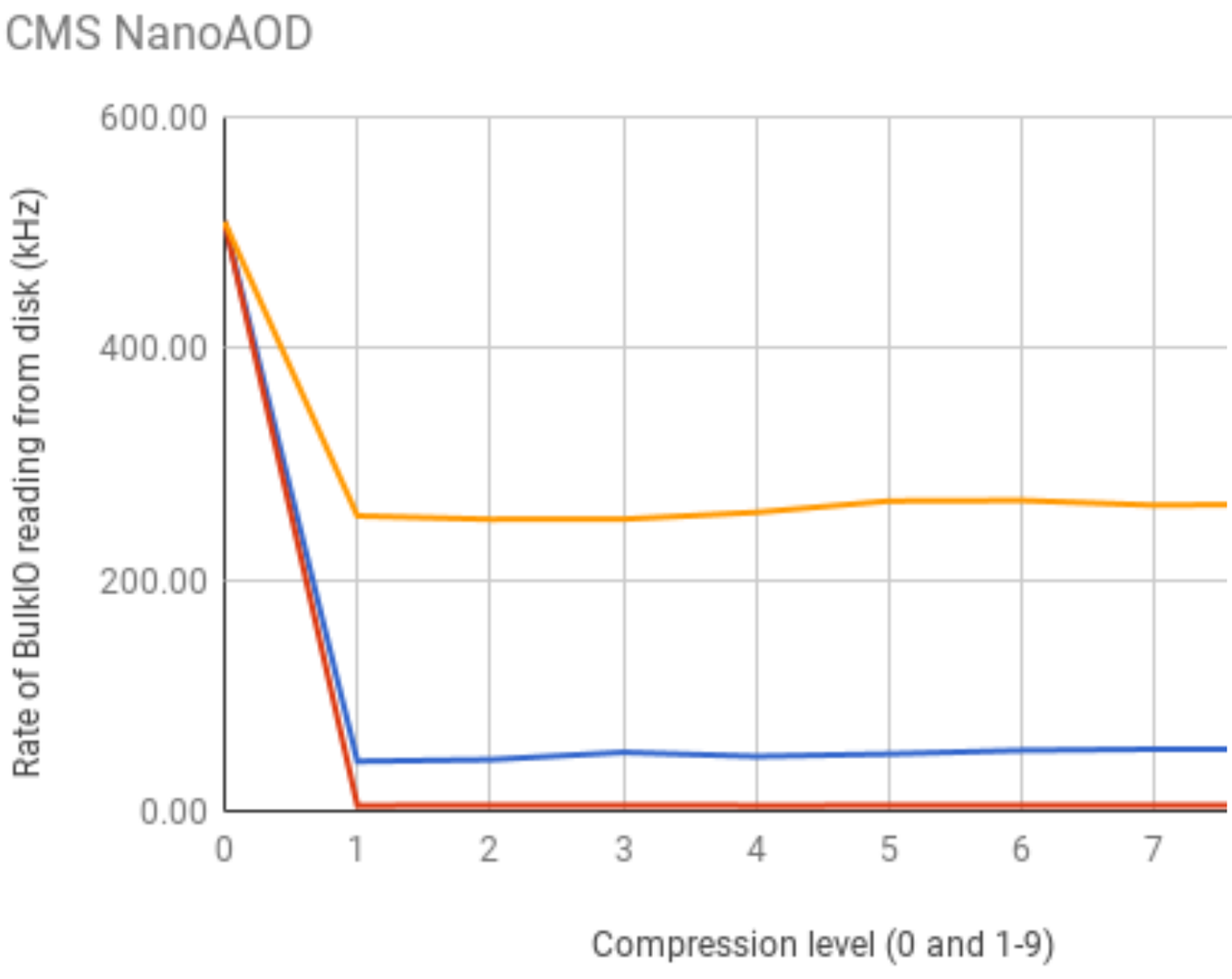
# Bulk IO Progress

- **Reminder**: Bulk IO is an approach that invokes ROOT libraries once per "event cluster" (~hundreds of events) instead of per-event.  Only applicable for simple object types, but potentially huge speedups.

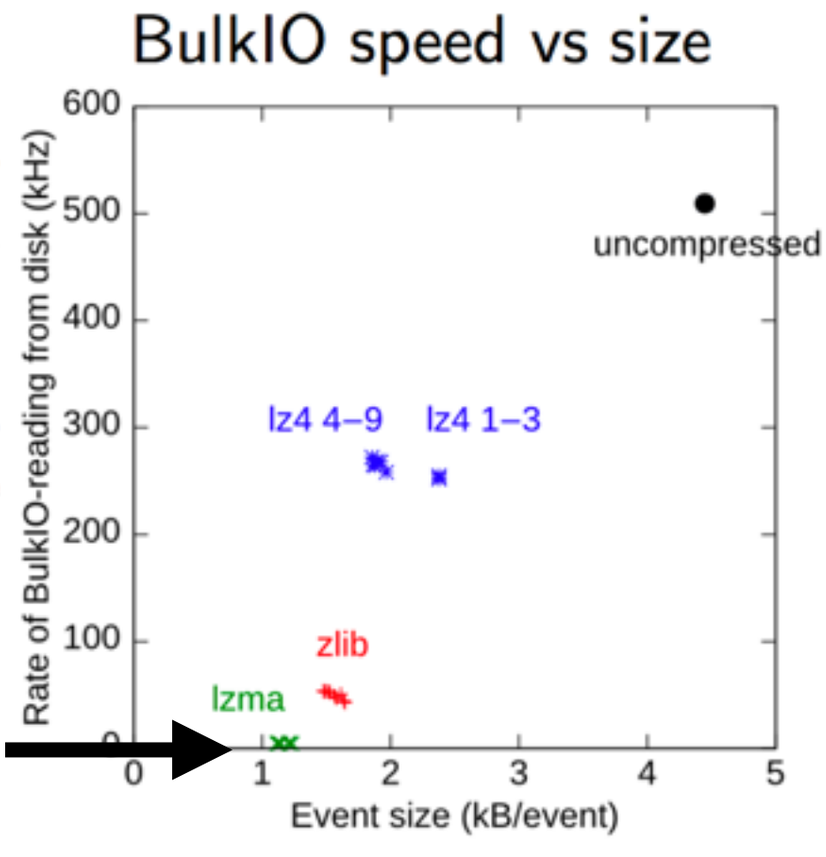
- Over the last summer, bulk IO…
    - Matured enough to build two high-level interfaces (Python/numpy and TTreeReader-like).
    - Got enough functionality to do realistic performance comparisons.
    - Got into a reviewable state and put in as a PR.
- Aiming to get this into 6.14 release.
- What follows are some slides from Jim and Oksana on performance tests

# Bulk IO - >10x benefit

And BulkIO reading is super-fast: serious penalty for LZMA




**100x penalty for LZMA**



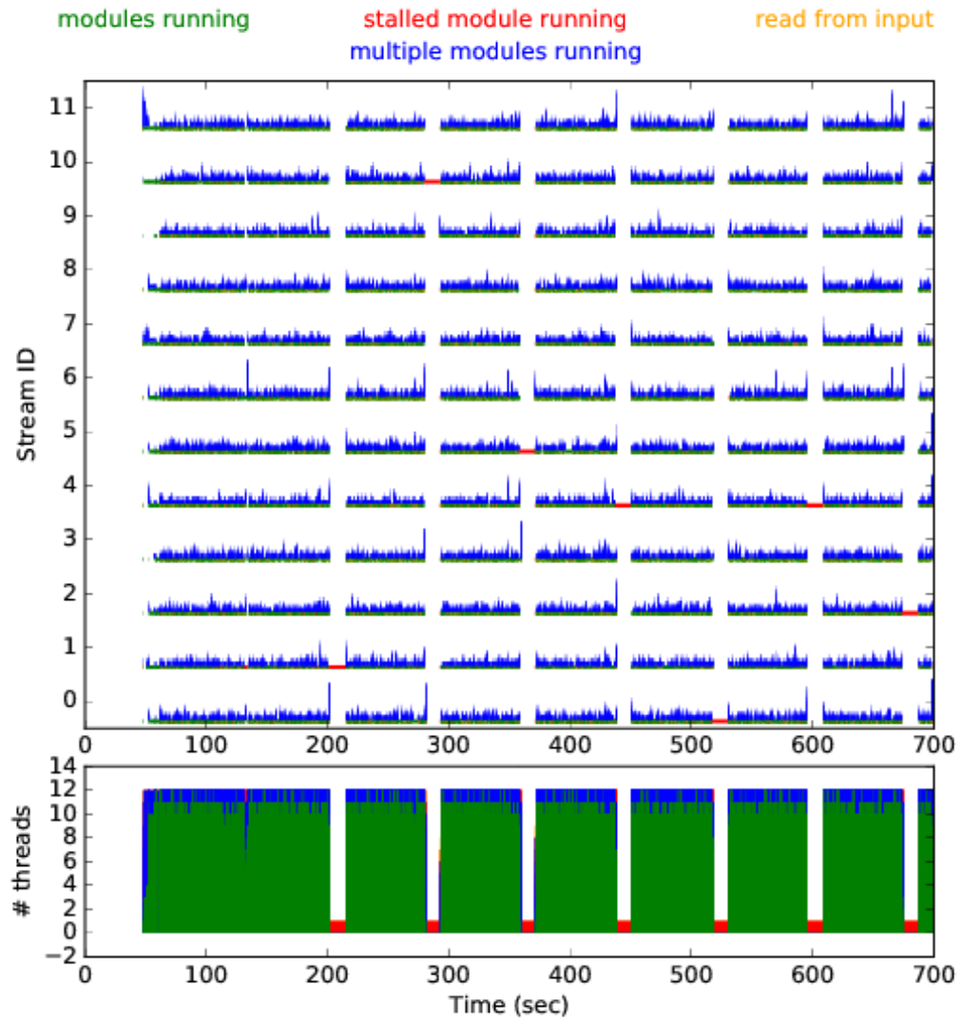
# CMS Parallel Merging



## ROOT I/O limits CMS scaling

CMS production jobs are multithreaded

- Production jobs currently use 4 cores with 4 framework event streams
- Output is handled by "one" modules that can only be active on one thread at a time
- ROOT output is the dominant source of output stalls
  - We lose efficiency with more than 4 cores, preventing us going to 8 cores
- Compression is the principal bottleneck
  - Especially for AOD and MINIAOD data compress with LZMA






https://indico.fnal.gov/event/15154/contribution/7/material/slides/0.pdf

# CMS Parallel Merging



## CMS Implementation

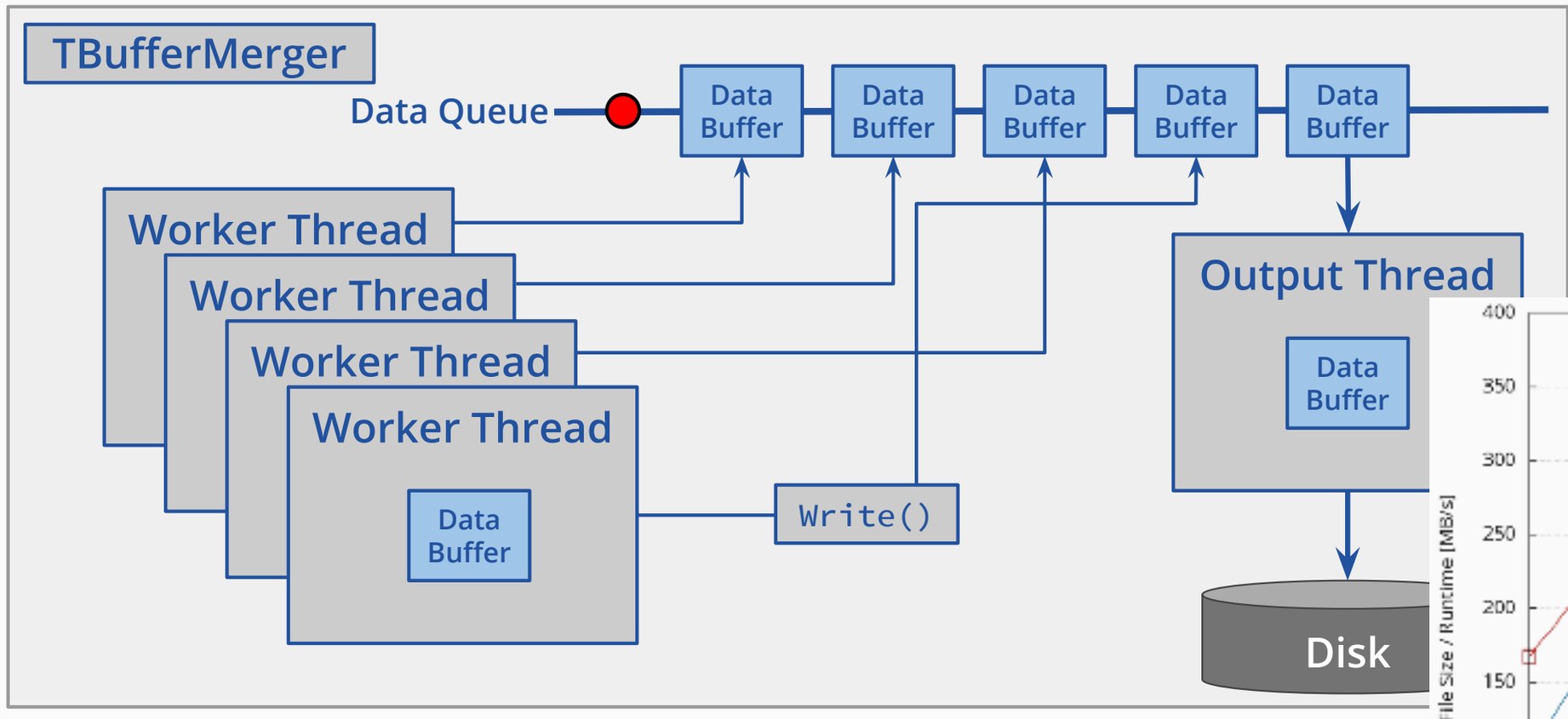
Refactored the CMS output module

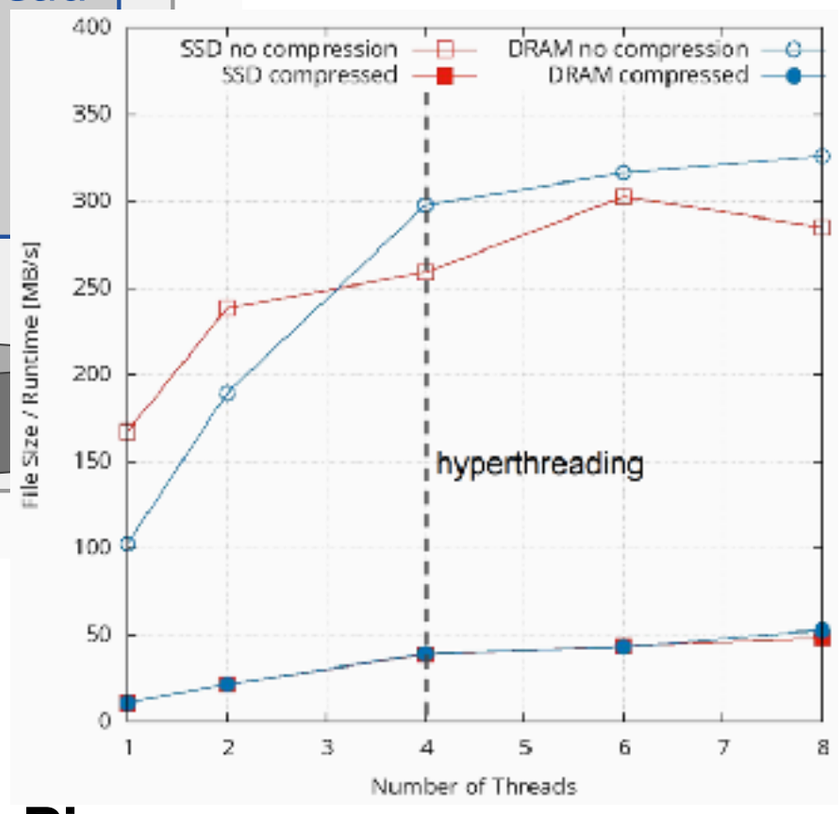
- Kept single-threaded ("one") output module for cases that are IO bound
- Factored out common bookkeeping code
- **Chris Jones implemented a new "limited" module type**
  - Normal "stream" and "global" modules have parallelism limited only by the thread count; "limited" modules have explicitly limited parallelism
  - Goal is to only have as many TBufferMerger buffers as necessary, not one for every thread
- **Parallel output module uses a tbb::concurrent_priority_queue to manage a pool of output buffers**
  - Priority is set so that the available TBufferMerge with the most entries is used, to prefer filling buffers quickly
  - Minimizes tail and synchronization effects (vs. FIFO/round-robin)



# TBufferMerger

## TBufferMerger Class



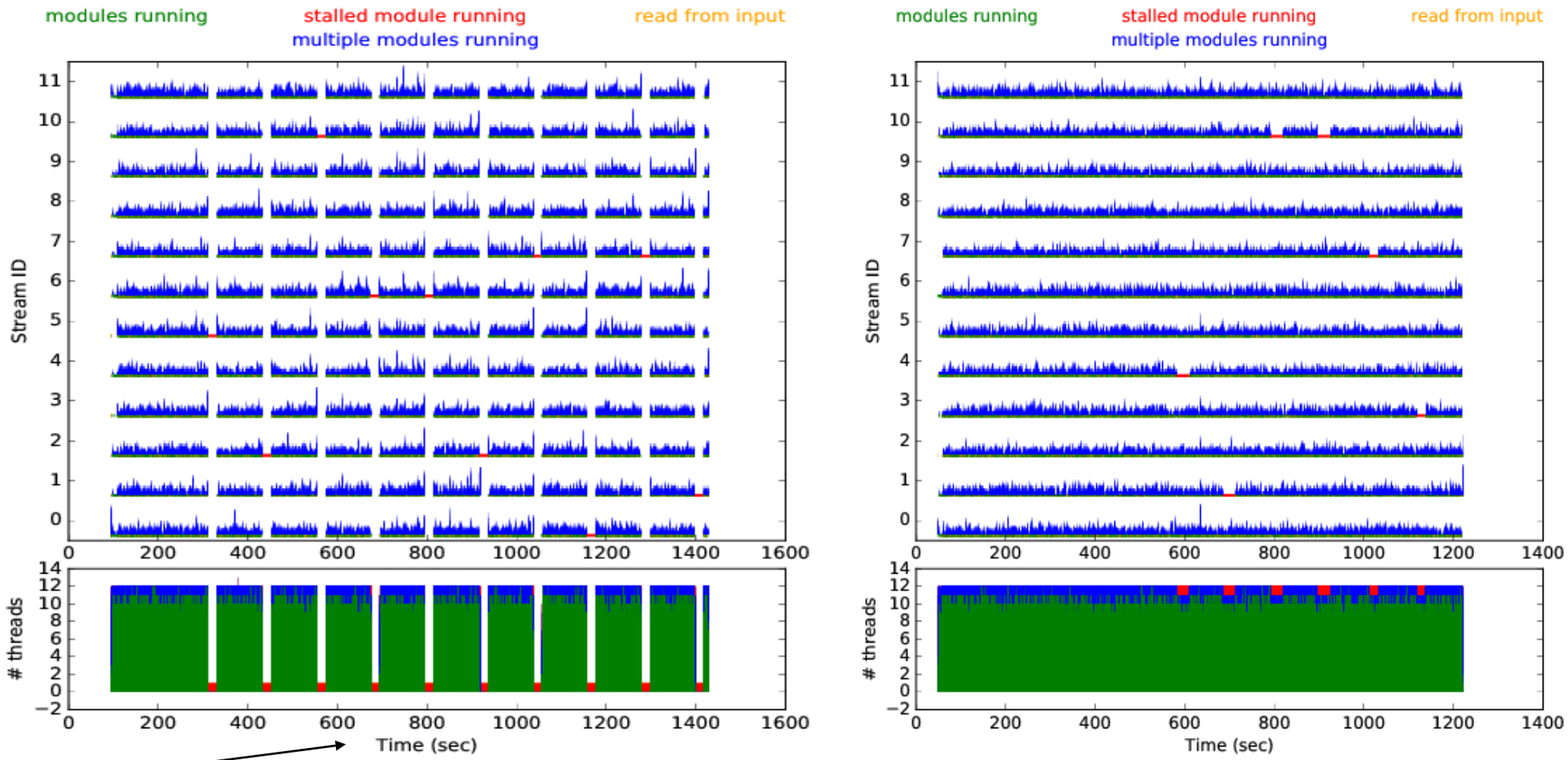

**Work by G. Amadio, P. Canal, and D. Piper**

**https://indico.fnal.gov/event/15154/contribution/9/material/slides/0.pdf**

# CMS Parallel Merging



## Stall Graph Comparison, LZMA 9




Note scale change



**At 12 threads, scaling efficiency goes from 90.7% to 96.9%**

# Other: Using Object Stores for petabyte-scale TFiles.


Idea #2 (this talk). Keep ROOT data as they are, but put individual *TBaskets* in the object store. TFile/TTree subclasses fetch data from the object store instead of seeking to file positions.

1. Presents the same TFile/TTree interface to users; **old scripts still work.**
2. But data replication, storage class, and caching are handled by the object store with columnar granularity.
3. Branches are shared transparently across derived datasets: all trees are friends.
4. The logic of sharing, reference counting branches, managing datasets, etc. must all be implemented in ROOT; only ROOT understands how to combine branches.

(the "ROOT becomes the database" approach)

**https://indico.fnal.gov/event/15154/contribution/10/material/slides/0.pdf**

**Generated lots of good discussion!**
**Focused on potential simplifications to show concepts.**

# Up-and-coming: Forward Compatibility Breaks

- ROOT 6.12 will introduce a new mechanism for detecting forward-compatibility breaks.
    - This way, if a file was written with feature XYZ, older versions of ROOT will detect they can't support it give a clear error message.  (Instead of crash or return incorrect data…)
- Current plan is to introduce experimental features (disabled by default) through the ROOT6 series.  ROOT7 will enable a few of these by default.

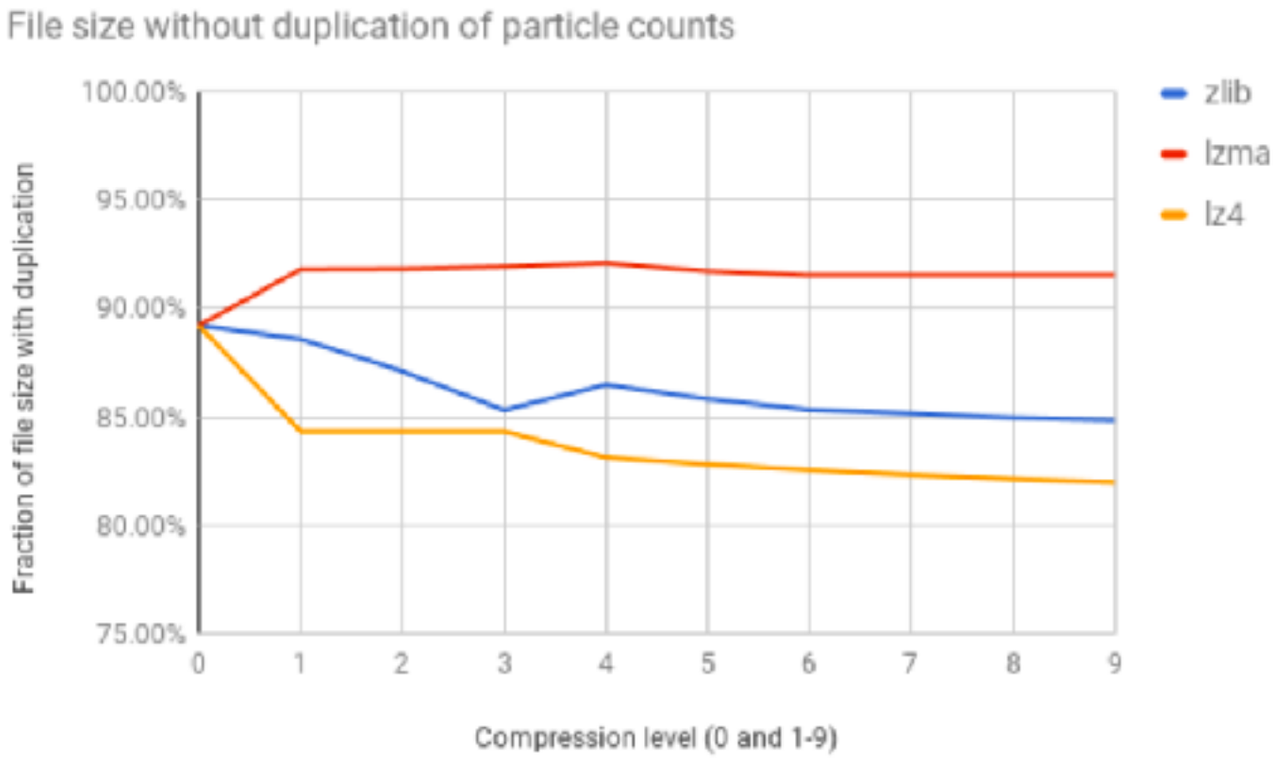
# Example: Skipping Entry Offsets

- For some object types, ROOT cannot predict the number of serialized bytes (Think: dynamically-sized arrays). Hence, the branch contains an "entry offset array" to save where each entry is in a basket.

    - For many "split" objects, this is written once per attribute — and can be calculated from a different branch (Think: dynamically-sized array of `int`).

- Forward compatibility break: skip writing out these arrays when they aren't needed.

- Saves about 18% in file size for LZ4 (CMS NanoAOD)

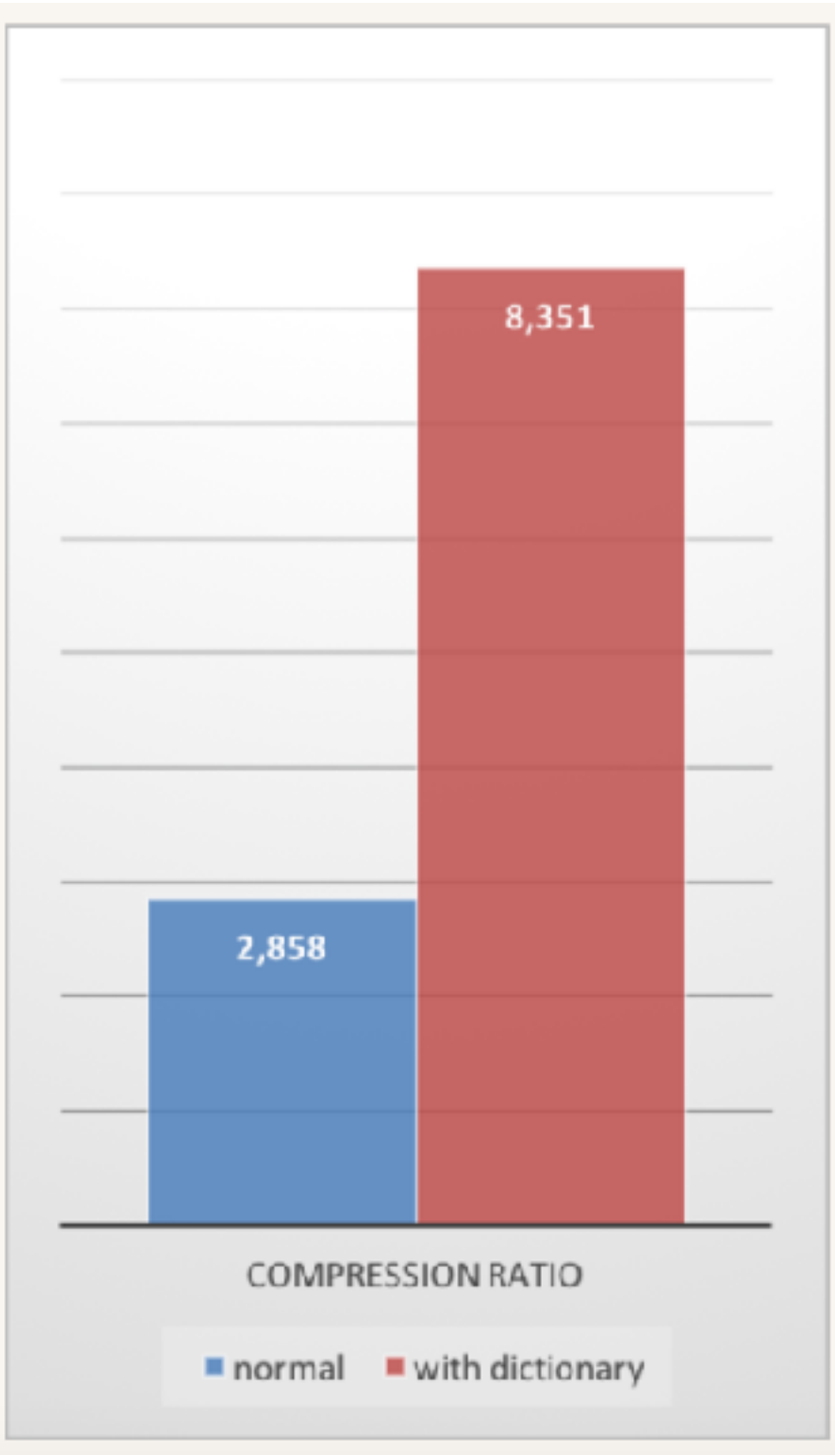


# ZSTD\

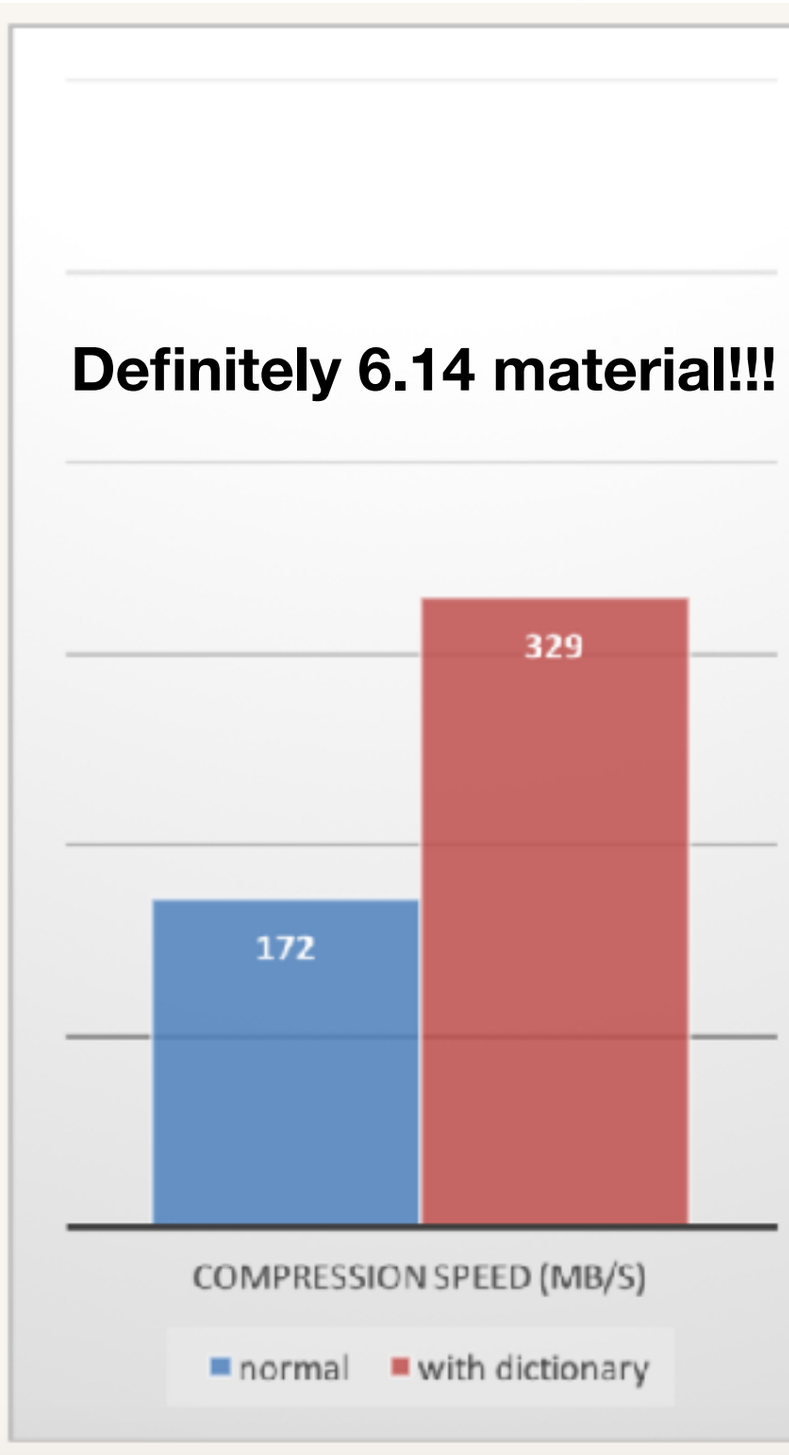
- ZSTD is an interested new compression algorithm because it has a rich API for generating and using compression dictionaries.
    - Facebook developers report massive speedups and compression ratio improvements when using dictionaries (almost a 3x improvement in compression ratio!) on a corpus of 10,000 entries of 1KB each.
- If we can get *anything near that*, then it would be a huge improvement for ROOT.
    - **Idea**: after the first event cluster, analyze the buffer and write out a separate compression dictionary.
    - No clue how much of Facebook's success can be repeated in ROOT, but appears worth investigating this winter.

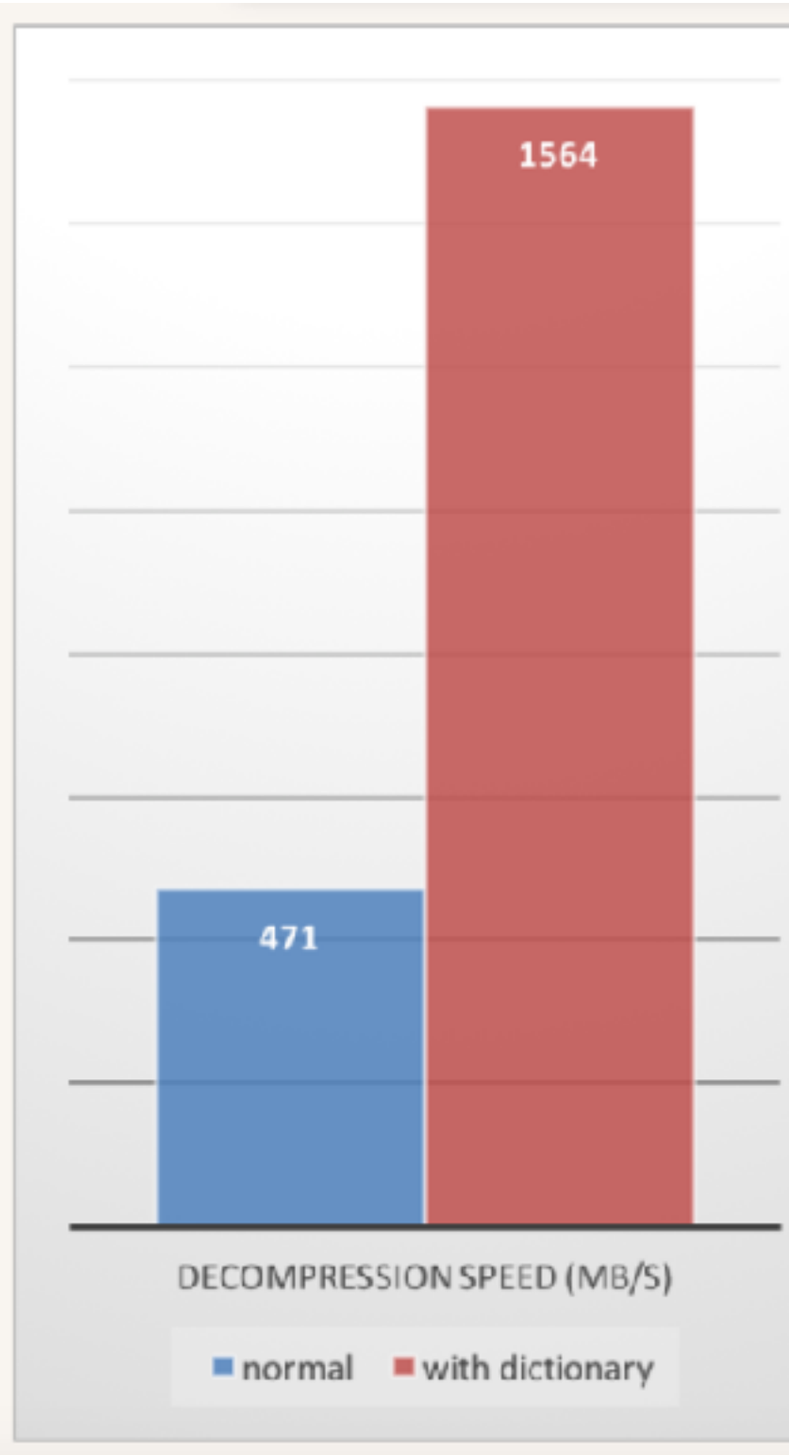
# ZSTD - With Dictionaries

**Source: http://facebook.github.io/zstd/**




**Definitely 6.14 material!!!**



# Up-and-Coming:
## Release Plans

- 6.12:
    - Forward compatibility break mechanism.
    - Skip offset entry writing.
    - Parallel, asynchronous unzipping.
    - Performance / locking improvements inspired TBufferFile.
- Likely 6.14:
    - Bulk IO initial version.
    - Improved zlib
    - ZSTD (or further out?)