

Coordinate-indexed addressing for per-request ROOT TTree I/O overhead

A practical, minimal-layer approach: reducing per-request I/O overhead in a TTree access pattern, measured on CMS Open Data with synTagma arithmetic addressing

Taeho Lee*
SSCCS Initiative
ssccs.org

August, 2026

Code
[ROOT fork](#)

Code
[Benchmark](#)

Whitepapers
[Tagma](#)

Abstract

ROOT is the petabyte-scale data analysis framework of the CERN community, and its TTree columnar format stores the events of nearly every physics analysis. A production bottleneck limits the read path: an analysis issues 372,000 singular reads averaging 4.6 KB, sustains an effective 33 KB/s, and runs for roughly 14 hours. The measured cause is per-request I/O overhead: each small read carries a system call, a cache lookup, and a context switch, while CPU and decompression stay idle in the remote production pattern.

This report measures a coordinate-indexed read path, built on the synTagma¹ coordinate engine, inside a fork of CERN's ROOT. The path replaces branch, basket, and cache traversal with closed-form coordinate arithmetic: an event addressed as (run, luminosity block, event number) resolves to a direct byte offset with no hash and no index scan. It is a working implementation: a self-contained store class plus byte-source and entry hooks, with existing analysis code, the TTree API, and the build chain unchanged. The dataset these rows measure is a single file, so the axis they populate is the event number; the coordinate reaches a dataset of many files in the later phases.

On the full CMS Run2016G DoubleMuon NanoAOD first file (2,315,223 events), the same medium, the coordinate path completes the read in 2.71 to 2.81 s against 187.0 to 188.2 s for the cache-disabled baseline, a 67x to 69x gain, serving every event with one aligned request. The memory-mapped variant issues zero application-level read system calls and completes in 1.55 to 1.61 s, a 117x to 121x gain, after a one-time conversion of 226.3 s. An analysis workload reproduces identical results on both paths, 20,861 selected events with histogram mean 132.696, at 69.2x. Those ratios are measured against a baseline that reads every branch of a compressed file, and the controls that separate the components narrow the claim: decompression removal accounts for 2.2x to 2.3x on the full file, the store is 13.1x faster than a baseline reading the same columns uncompressed, and on the two-column selection of the analysis workload ROOT's column-selective reader is 4.9x faster than the store. The store's cost does not fall with the number of columns an analysis selects, because the record is the addressing unit, so its advantage is column independence above a measured crossover near 11 columns, or about 6 with the store mapped. Served bytes are verified by a checksum, and analysis results by exact histogram equality. Every row is measured against a warmed source: the harness reads each local source through the page cache in an untimed pass before the timed pass, which is what makes the rows comparable and what the earlier single-run numbers did not hold constant.

The fork has since carried the path through the later phases, and the section Measured results, phases 2 to 6 is the current record. The store now carries the whole event rather than a projection, 974 scalars and 19 collections at two reads per event, and it compresses, 2.44 times, with the read path at 572 MB/s. The result that bears most

*Founder & Architect; Corresponding author: lee@ssccs.org

¹synTagma, Spatial coordinate space computing system docs.sccs.org/projects/syntagma, Pre-release, Apache 2.0.

on the signature this report opens with is the access pattern itself: the phase-1 rows above are all scans, while the production signature is an analysis reading selected events in list order. Read that way, over 2,000 events, the baseline pays 683 requests per event and 202.2 s, while the store reading the whole event, index record and slice with the fields delivered, pays two reads per event and 0.101 s, so the coordinate path wins there by 2,000 times rather than by tens; the block-compressed store is at 345 times. The store's per-event cost does not move with the order, and the penalty behind the baseline's number is a basket boundary rather than a slope. The coordinate also reaches a dataset rather than one file: one shard per run, 38 runs, a run resolved by arithmetic against the chain's run-branch scan of 11.1 s, at 18.1 ms per file to attach against the chain's forced 9.3 ms, with the dense grid carrying 1.24 times the event cells. The limits are stated with the result: a whole-event read is 1.5 times, because the entry layer, the store behind the tree interface delivering the fields, dominates; a narrow column selection is upstream's to win below the column crossover, near 42 scalar columns for the read path and near 810 with delivery; and $O(128)$, multi-terabyte samples, and the remote medium need infrastructure. The 128-core row is predicted rather than measured: the per-event request count is media-independent and settled at the scale measured, so the tendency holds there and only its wall-time magnitude needs the larger machine.

1 The TTree access pattern

The production case in the 2025 Fermilab CCESOP analysis is an analysis over a CMS NanoAOD dataset that performs 372,000 singular reads averaging 4.6 KB, at an effective 33 KB/s, over roughly 14 hours². The scale in bytes is small: 1.6 GB moves in seconds at raw bandwidth. The cost is per request.

The structural origin is the TTree columnar layout. A NanoAOD event is split across thousands of branches, about 153 data products over 8,190 branches in the configuration³, so one logical event scatters across many basket buffers. Reading one event issues many small reads; each carries a system call, a page cache lookup, and often a context switch. Multithreaded runs calling `GetEntry` out of order invalidate the `TTreeCache` and collapse vector reads into singular reads; beyond 100,000 clusters the cache degrades from 7 seconds to over an hour.

The baseline measurement, milestone M1 of the development plan⁴, reproduces the singular-read signature on the real dataset. With the `TTreeCache` disabled, the first 2,000 events read at 1.37 reads per event, 2,635 bytes per read, and the remote EOS run spends about 98.8 percent of wall time in I/O wait, with CPU below 2.5 percent. Decompression capacity headroom was measured at 18 to 44x across access patterns⁵; the bottleneck is per-request I/O wait.

A first-order cost model separates the terms per read request:

$$T_{\text{req}} = t_{\text{syscall}} + t_{\text{cache-lookup}} + t_{\text{copy}} + t_{\text{context}}$$

and the per-event cost of the baseline is

$$T_{\text{base}} = N \cdot R_{ev} \cdot T_{\text{req}}$$

where N is the event count and R_{ev} the measured reads per event. Any structural fix must reduce R_{ev} and T_{req} ; the coordinate approach reduces R_{ev} to 1 and, when mapped, T_{req} to a memory copy.

²The case is the 2025 Fermilab CCESOP analysis: 372,000 singular reads averaging 4.6 KB at an effective 33 KB/s over roughly 14 hours. The M1 measurement reproduced the signature on the CMS Run2016G DoubleMuon NanoAOD first file: 1.37 to 2.76 reads per event at 2.6 to 0.8 KB per read across cache-disabled slices.

³The branch count is the NanoAOD configuration from the CCESOP analysis. The file used in this project has 1,380 leaves; its smaller scatter lowers the full-file reads per event to 0.20 once baskets are resident, see the basket footnote.

⁴The project tracks the work in milestones M1 to M5, stated in the Direction: M1 instruments and measures the baseline read path, M2 to M4 build and integrate the coordinate-indexed store, and M5 releases the demonstration. This report covers the measured outcome; later references to the M1 signature and the M1 slice mean the cache-disabled baseline access pattern measured in M1.

⁵The headroom is the ratio of the decompression capacity (`TTreePerfStats ReadUZCP`) to the effective throughput, measured at 27x, 44x, and 18x on the cache-on, 2,000-entry cache-off, and 500-entry cache-off runs of M1.

2 The Innovation: Addressing Replaces Traversal

The coordinate store is a change of addressing model. ROOT traverses: an entry walks branches, baskets, and the cache to find its bytes, and each step can reach the medium. Tagma computes: the entry number is a coordinate, the coordinate resolves to a byte offset in closed form, and the record is served directly. The difference is structural. A cache reduces the number of medium accesses, but every miss still carries a system call; the mapped coordinate store removes the system call by construction, because the byte source is the mapping. No cache-size tuning can reach that point.

For CERN the relevance is direct. The measured workload is a real CMS Open Data analysis file, the same access pattern the community documents as the 14-hour case. The researcher's interface is untouched: the same TTree API, the same analysis scripts, the same build chain, with the read path replaced behind the existing interfaces. The demonstration complements ROOT and RNTuple rather than competing with them: the coordinate store serves the existing TTree files in production, so analyses do not migrate, while RNTuple remains the forward path and this is the backward-compatible one. The measured evidence is open and reproducible from the fork and the runbook.

3 The Coordinate Approach

An event is a coordinate (run, luminosity block, event number). The canonical Tagma composition is the closed form

$$C(i, m, f) = U + AC00 + 588i + 28m + f, \quad 0 \leq i < 19, \quad 0 \leq m < 21, \quad 0 \leq f < 28$$

over a fixed 16-bit block with 11,172 valid states⁶. The coordinate-indexed store generalizes this to per-axis maxima. Let the layout be (R, L, E) for the exclusive bounds of the run, luminosity block, and event axes, and let s be the fixed record size in bytes. The linear index of a coordinate is the mixed-radix composition⁷

$$n(r, \ell, e) = (rL + \ell)E + e, \quad 0 \leq r < R, \quad 0 \leq \ell < L, \quad 0 \leq e < E$$

which is a bijection onto $[0, RLE)$, and the byte offset of the record is the product

$$\omega(r, \ell, e) = n(r, \ell, e) \cdot s$$

No hash function and no index scan are involved. The inverse, decomposing a linear entry number back into axes, is

$$e = n \bmod E, \quad \ell = \lfloor n/E \rfloor \bmod L, \quad r = \lfloor n/(LE) \rfloor$$

The read path cost becomes

$$T_{\text{coord}} = N \cdot (t_{\text{address}} + t_{\text{read}}), \quad t_{\text{address}} = O(1)$$

with t_{read} equal to one aligned record transfer, or a memory copy when the store is mapped.

⁶The 11,172 valid states are the product 19 by 21 by 28; the remaining 54,364 of the 65,536 representable 16-bit states are structurally invalid and detectable in constant time. See the Tagma white paper. docs.sscs.org/projects/syntaxtagma/tagma/index.pdf

⁷The mixed-radix composition generalizes the canonical form: setting (R, L, E) equal to $(19, 21, 28)$ and s equal to 1 recovers the canonical index. The bijection onto the range 0 to RLE minus 1 holds for any positive axis maxima; the constructor validates that the record count and extent fit a 64-bit word.

4 The Direction

The work follows a direction set before the measurements: the same stack, a different low-level implementation. The seam sits between the high-level interface and the file I/O layer; only the thin event-lookup layer is replaced, with the Tagma read path serving covered byte ranges by closed-form arithmetic before any disk access. The TTree API, the build system, the workflow, and ROOT's disk handling stay as they are. The goal is zero impact with the interior changed.

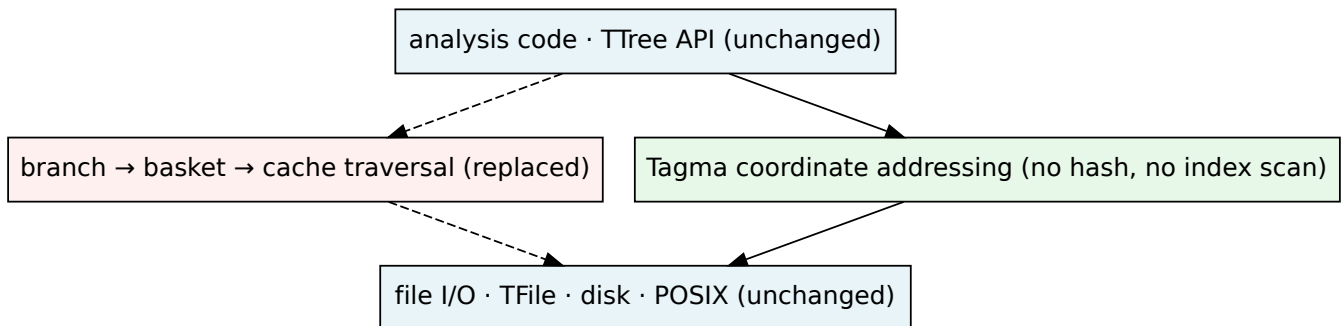


Figure 1: The implementation direction: the stack stays as it is, and only the thin event-lookup layer between the TTree API and the file I/O is replaced by Tagma coordinate addressing.

The implementation is delivered as three parts: the preparation tool that converts the dataset into the store, the thin byte-source layer that serves mapped data and passes everything else through, and the benchmark harness that runs the same workload against the ROOT baseline. Five milestones track the work: M1 measures the baseline read path, M2 converts the dataset, M3 builds and validates the byte-source layer, M4 benchmarks the analysis pattern, and M5 releases the demonstration and the measured comparison as open source.

If the measured comparison holds, the results provide the basis for discussing wider adoption and a potential contribution to the ROOT ecosystem.

5 The Fork Implementation

The implementation lives in the [ssccsorg/root](https://github.com/ssccsorg/root) fork. The byte-source layer and the entry layer are replaced behind the existing interfaces; everything else falls through unchanged.

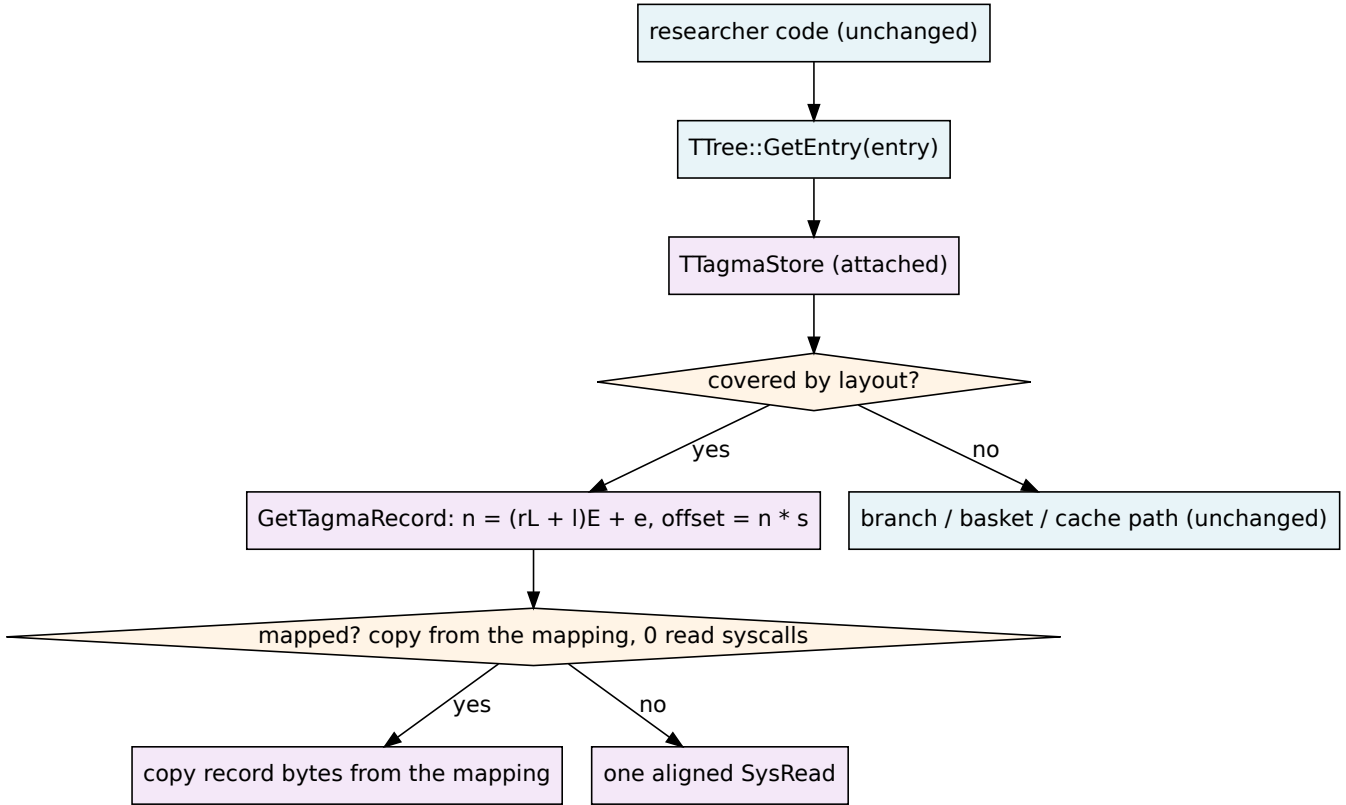


Figure 2: The coordinate seam in the TTree read path. Covered entries are served through the store by closed-form arithmetic; everything else falls through to the ordinary path unchanged.

The store class `TTagmaStore` is self-contained C++17 and carries the layout validation: the axis maxima and the record size must be nonzero, the record count must fit in a 64-bit word, and the store extent must not overflow⁸. `MapFile` attaches a read-only `mmap` of the store file following the canonical `CoordSpaceM` discipline from the syntagma reference: the region is demand-paged once with sequential locality, and covered reads copy from the map without an application-level read system call⁹.

The `TFile::ReadBuffer` hook serves a request when it resolves to exactly one fixed-width record: the position is record-aligned, the length equals the record size, and the decomposed axes fall inside the layout bounds. The `TTree::GetEntry` short circuit serves covered entries through the same store, keeping the record in the tree and exposing it through `GetTagmaRecordBuffer` and `GetTagmaRecordSize`.

A preparation tool converts a real dataset into the store: each event becomes one fixed-width record holding the first 320 leaf values serialized as doubles, with a sidecar checksum and a layout sidecar that maps field names to record offsets. The 320 is the record capacity at the measured 2,560-byte record size, so the store holds at most that many of the tree's 1,380 leaves, taken in tree order. The selection tests the static leaf length, which a variable-length array also reports as one, so 276 of the 320 fields are arrays from which only the leading element is written and 44 are scalar leaves¹⁰. The fixed-width projection therefore carries a scalar subset plus the leading value of each array, and it does not carry the collections themselves.

⁸The constructor rejects layouts with a zero axis or record size, a record count that overflows, or an extent that overflows, so `Compose` and `Offset` can never overflow for in-bounds coordinates. The entry layer additionally bounds the record size to 1 GiB so the length fits the `Int_t` argument of `TFile::ReadBuffer`.

⁹The mapping is read-only and private. The kernel demand-pages each page once on first touch; zero read system calls means zero application-level read calls, while the page faults are kernel-internal.

¹⁰The preparation tool selects leaves whose static length is one and serializes each value as a double, so a variable-length array branch enters the record through its leading element. Of the 320 fields, 276 have a leaf count in the file and 44 do not. The two analysis columns, `MET_pt` and `nMuon`, have no leaf count, so the analysis measurement is unaffected.

6 Benchmark Methodology

The workload is the M1 pattern: read every event of the CMS Run2016G DoubleMuon NanoAOD first file, tree Events, 2,315,223 events, 2,155,974,646 bytes, with the TTreeCache disabled in the baseline. The baseline is the current ROOT read path, branch to basket to cache to file, with no store attached; the coordinate hooks are dormant null checks when no store is present, and the branch, basket, cache, and file layers are unchanged from upstream¹¹. The coordinate rows read the fixed-width store converted from the same events, 2,560-byte records, 320 scalar leaves. Both paths call the same API, `GetEntry`, so the comparison isolates the interior read path. The layout in the phase-1 measured rows holds a single populated axis: the run and lumi extents are one and the event axis carries the entry count, so the composition evaluates to the entry index times the record size. The mixed-radix composition over nonzero axis extents runs the same code and is verified against the reference engine over the canonical lattice, and the dataset-level layout whose axis extents come from the data is delivered and measured in the dataset subsection of the later results.

The environment is a single Arm Firestorm 3.2 GHz core¹², macOS, out-of-source Release build with `treeplayer` and testing enabled. Absolute times are machine-specific; the request and system call counts are media-independent.

The reported metrics are wall and CPU time, request count, coordinate-served count, bytes moved, read system call count, and cache efficiency. Correctness is verified two ways: the benchmark checks that the store holds exactly the converted records and that the bytes the coordinate paths serve match the conversion-time checksum; the analysis workload checks that the same selection and histogram over both paths produce identical results.

7 Measured Results

7.1 Full dataset, same medium

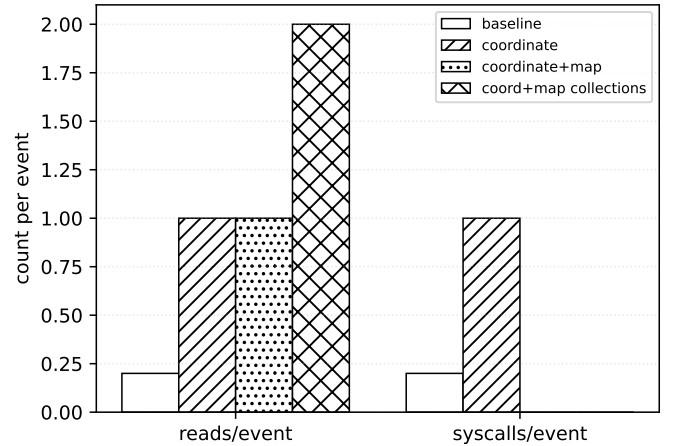
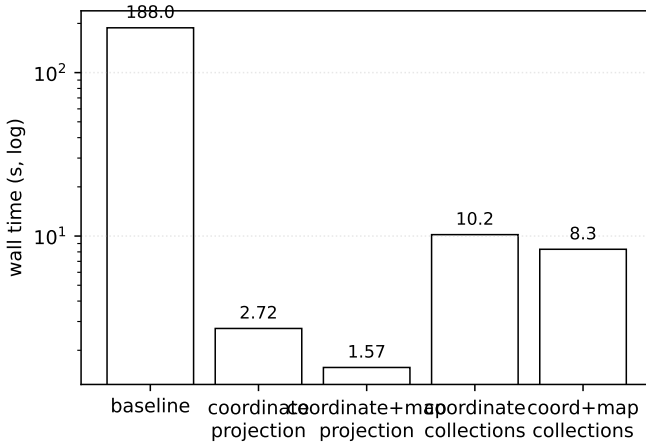


Figure 3: Full-dataset wall time, same medium, log scale. The median of three warmed runs for the projection and one run for the collections. The baseline reads the compressed file through the ordinary path; the coordinate rows read the raw store, the projection at 2,560 bytes and the collections at 1,280.

Figure 4: Requests and read system calls per event. The mapped path serves every event with one aligned request and zero application-level read system calls; the collection store serves the whole event, the index record and the event's slice, in two.

¹¹The fork's footprint on the baseline path is two dormant null checks and one read-system-call counter increment per operation. Over the full run this is bounded below one millisecond against a 187.0 to 188.2 s wall time, about five orders of magnitude below measurement noise, so the baseline matches the upstream ROOT read path. The measured read pattern, 1.37 reads per event at 2,635 bytes per read on the M1 slice, is produced by the unchanged branch, basket, and cache logic, and matches the 372,000 by 4.6 KB singular-read scale. A cross-check against a build without the coordinate changes is expected to confirm identical numbers.

¹²The core is an Arm Firestorm implementation at 3.2 GHz, Armv8.5-A. Absolute times are specific to this core; the request and system call counts are core and media independent.

The full comparison runs on local disk for both paths. The served checksum matches the conversion checksum (233,262,869,086), so the coordinate paths read the real converted event bytes¹³¹⁴. The baseline row reads every leaf's baskets and moves 2.15 GB, the file size, while the coordinate rows move 5.93 GB of raw records holding the store's 320 fields. The two rows do not move the same payload. The measured machine holds both files in the page cache, so the mapped row measures a memory copy of resident data rather than an access to the medium.

Path	wall (s)	cpu (s)	reads	reads/event	syscalls/event	bytes/read	MB/s	vs baseline
baseline	187.0-188.2	183.6-185.9	470,131	0.20	0.20	4,573	11.5	1x
coordinate	2.71-2.81	2.69-2.73	2,315,223	1.00	1.00	2,560	2,190	66.9-69.1x
coordinate+map	1.55-1.61	1.54-1.58	2,315,223	1.00	0.00	2,560	3,819	116.6-120.5x

The component controls, measured with the same warm-source protocol on the same events and the same medium:

Control row	wall (s)	cpu (s)	reads	bytes moved
baseline, cache off	187.0-188.2	183.6-185.9	470,131	2,149,961,076
baseline, compression 0	81.1-85.7	74.1-74.7	387,007	10,852,850,385
baseline, compression 0, the store's columns active	35.935	29.150	118,680	6,127,346,663
baseline, compressed, the store's columns active, 32 MB cache	111.111	107.200	1,985	1,404,068,431
coordinate	2.71-2.81	2.69-2.73	2,315,223	5,926,970,880
coordinate+map	1.55-1.61	1.54-1.58	2,315,223	5,926,970,880

Decompression removal is worth 2.20 to 2.31x of the compressed baseline's wall time, 102 to 106 s of 187.0 to 188.2 s, and 109 to 112 s of its CPU time. The rest of the gain is the read path. The payload-equal comparison is the uncompressed row with the store's columns active, which moves 3.4 percent more bytes than the store: the store is 13.1x faster, or 22.8x mapped. Against ROOT's best configuration for that payload, the same columns on the compressed file with the cache enabled, the gain is 40.5x, or 70.5x mapped. The 67x to 69x of the table above is measured against the all-branch row, which reads 1,380 leaves for a payload the store holds in 320 fields.

Two properties of these rows set what they support. The baseline is the term that moves with the machine state: three runs on a settling page cache measured it at 248.979, 202.469 and 182.609 s against 187.0 to 188.2 s in steady state, so a ratio quoted without the protocol carries that spread. The coordinate rows are the term that moves with

¹³The checksum is the 64-bit sum of every record byte, accumulated with wrap-around. The conversion tool writes it to a sidecar; the benchmark recomputes it from the bytes the coordinate paths serve, and any mismatch fails the run.

¹⁴The rows of the full-dataset table are the committed benchmark artifact in the fork, `benchmarks/tagma/result/reference.json`. It is tracked under a directory whose per-run results stay ignored, and the file records the run timestamp and the code commit it belongs to. Wall, CPU, request, and system call counts in this report match that artifact. The later-phase rows sit in the same file: `phases_2_6` carries the collection store, the compressed store, the scatter rows, and the thread rows, and `step_2_dataset` carries the shard partition and the dataset bench rows, each entry naming its tool and the layer the row reads at.

the source's residency: 2.79, 3.56 and 4.05 s unwarmed against 2.71 to 2.81 s warmed, and 1.06, 1.71 and 1.55 to 1.61 s for the mapped row. That 1.06 s is the low outlier of the nine samples of that row: the same row measured 1.71 to 1.79 s unwarmed and 1.55 to 1.61 s warmed, and the condition behind the 1.06 s was not recorded, so the ratio it implies is not reproduced.

7.2 Cache behavior, signature slice and remote access

With the TTreeCache enabled on a fresh file open, the first 10,000 events read in 8 read calls at 1,943 KB per read, cache efficiency 0.917, miss rate 0.083¹⁵. This is the ideal sequential case; the cache degradation under out-of-order multithreaded reads is a Phase 5 item in the Status section.

The 2,000-event slice on the same medium measures the signature established above: baseline 1.37 reads per event at 2,635 bytes per read in 0.466 to 0.483 s wall. The remote EOS baseline over the same slice runs 100.1 s, I/O-wait bound¹⁶, versus 0.002 to 0.003 s (coordinate) and 0.001 s (mapped) on the local store. The slice times on the coordinate paths are sub-millisecond; the derived ratios are indicative microbenchmark-scale values.

7.3 Analysis workload

The analysis workload applies the same selection, MET_pt above 100 GeV and at least one muon, and fills the same MET_pt histogram on both paths, over the full dataset. Both columns are scalar leaves with no leaf count, so the projection carries them faithfully.

Path	wall (s)	selected events	histogram mean	vs baseline
analysis baseline, every branch	187.9	20,861	132.696	1x
analysis coordinate	2.72	20,861	132.696	69.2x

The selection counts and the histogram means are identical, which verifies that the coordinate path reproduces the baseline analysis exactly. The ratio in this table compares against a baseline that reads all 1,380 leaves for a selection that needs two.

Measured on the same events, the two columns read through RDataFrame, which reads the active columns through their branches and never walks the tree's branch list per event, take 0.558 s. The coordinate path takes 2.715 s on the same machine, which is also what it spends on the whole record, so on a two-column analysis it is 4.9x slower than the reader, and 1.6x slower for the mapped path at 1.577 s. A two-column analysis is the case where the coordinate store is weakest: the store reads the whole 2,560-byte record whatever the analysis selects, while a column-selective reader reads two columns. The same reader costs 10.849 s for the store's 44 scalar columns, against the store's 2.715 s for 320 fields, and the per-event costs of the two readers give a crossover near 11 columns, or near 6 with the store mapped¹⁷.

7.4 Synthetic workload

The synthetic workload, 20,000 events of 2,560-byte records with a 3-branch scattered tree, isolates the mechanism without the dataset:

¹⁵The efficiency is TTreeCache::GetEfficiency, the fraction of requested blocks found in the cache; the miss rate is its complement. The pass measures the ideal sequential case on a fresh file open.

¹⁶Remote reads go through the XRootD network plugin, which does not call TFile::SysRead, so the system call counter reads zero for the remote baseline; the request count and bytes moved still apply.

¹⁷Per event the column-selective reader costs 0.022 microseconds plus 0.112 per column, from the two-column and 44-column rows, and the coordinate path costs 1.54 microseconds, or 0.74 mapped, independent of the column count. The crossover is a first-order estimate from those two points, not a fitted model. The store's per-event cost includes the benchmark's byte-by-byte checksum over the 2,560-byte record.

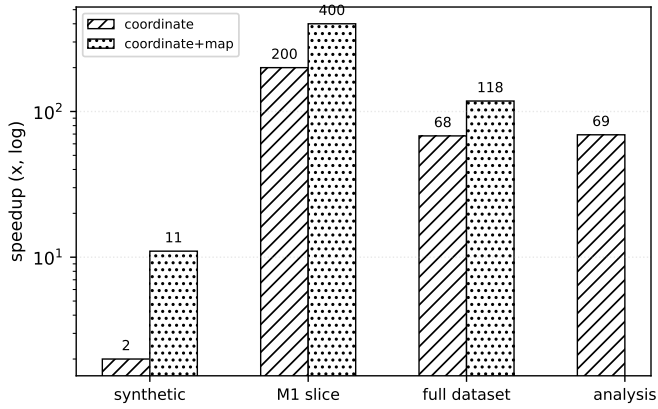


Figure 5: Speedup over the baseline across measured conditions, log scale. The mapped row is the coordinate+map path; the analysis row is the analysis workload.

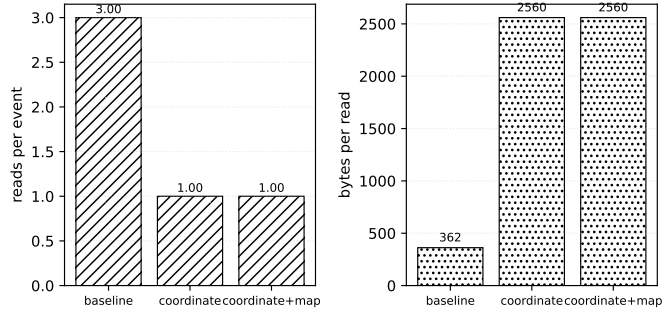


Figure 6: The synthetic mechanism: the baseline scatters one event into three reads of about 362 bytes (compressed baskets); the coordinate paths serve one aligned read of the full 2,560-byte record.

Path	wall (s)	reads/event	syscalls/event	vs baseline
baseline	0.128-0.138	3.00	3.00	1x
coordinate	0.026-0.078	1.00	1.00	1.6-5.3x
coordinate+map	0.011-0.027	1.00	0.00	4.7-12.6x

The coordinate rows move 51 MB in 20,000 aligned reads, so their 0.03 to 0.08 s wall is set by per-syscall cost and timer resolution rather than by the read path. The counts per event are the stable result here, and the ratio column is indicative.

8 Measured results, phases 2 to 6

The section above measures the phase-1 projection. The fork has since carried the read path through the later phases, and this section is that record, on the same machine and the same source file, with the dataset subsection below extending the partition to one file per run. The harness in the fork's `benchmarks/tagma` reproduces every row, and `tagma_bench.C` is the entry point over the tool set.

8.1 The store carries the collections

The phase-1 store holds a fixed-width projection of the event: of its 320 fields, 276 are variable-length arrays truncated to their leading element. The read path now carries them whole. The conversion tool derives a schema from the tree, one scalar field per scalar numeric leaf and one collection per variable-length array leaf bounded by its count leaf, preserves the leaf types instead of widening every value to double, and writes a self-describing store: 974 scalar fields and 19 collections, an index record of 1,280 bytes, a data region of 4,118,805,545 bytes and a payload of 7,082,290,985 bytes over 2,315,223 events, converted in 1,963.3 s. The whole event is two reads per event, the index record and the event's slice, and it reads back byte for byte through the ordinary leaves.

Against the same cache-disabled baseline, the read path moves 7,082,290,985 bytes in 10.2 s against the baseline's 176.7 s, and a scalar-only selection, which the reader serves from the index record alone, in 4.2 s, 42 times, or 54 times with the store mapped. The whole-event row with the branches active runs 117.6 s: the read path is 10.2 s of it and the branch delivery, about 50 microseconds per event at this single-file scale, is the rest. Delivery, not the read path, decides that comparison.

8.2 The store compresses under the same addressing

The phase-1 store is raw and moves 2.76 times more bytes than the compressed baseline, which this report names as the phase-3 gap. The store now splits its payload into fixed blocks, compresses each block on its own, and closes the file with the same descriptor at a higher version, the block size carried in it. The address arithmetic is untouched, so the file hook, the branches, and the leaves do not know the difference, and TTagmaBlockSource decompresses the blocks a read touches. The payload closes to 2,897,372,833 bytes, 2.44 times, in 75.4 s. That ratio is the store's payload against its own raw form. Against the baseline file, 2,155,974,646 bytes, the compressed store, 2,897,372,833 bytes, is 1.34 times, down from the raw collection store's 3.28 times: the file gap narrows and the store remains the larger file on disk. The two compressors differ: the baseline takes its uncompressed tree, 10,852,850,385 bytes, to 2,155,974,646, 5.0 times, while the store closes the smaller raw form, 7,082,290,985 bytes, by 2.44 times.

Read on its own, with no entry layer, the whole payload through the byte source takes 12.4 s at 572 MB/s, against the raw store's 14.6 s and the baseline's 177.4 s. Compression therefore keeps the read-path advantage and shrinks the file; it is not a trade against the read, and the whole-event row that suggests otherwise is the delivery layer, which costs about 50 microseconds per event at single-file scale whatever the order and is measured directly in the scatter subsection below. A block size of 8 KB, one line of the format, cuts the scattered read below to 16 microseconds per event against 222, and costs the file almost nothing, 2.44 to 2.37 times.

8.3 Event-selected reads, the regime this report motivates

This report opens on the production signature, 372,000 singular reads averaging 4.6 KB over fourteen hours, and the per-request cost that produces it. That signature is an analysis reading selected events, in list order, and every phase-1 row is a scan. Measured in list order over 2,000 events on the same medium, the 8 KB block store, one run. The index rows copy the addressing unit, the index record; the entry rows read the whole event, the index record and the slice with the 1,380 field branches delivered, so they are layer-matched and payload-matched with the baseline, which drives TTree::GetEntry with no branch addresses. The entry rows are gated: before they are timed, the delivered event and MET_pt values are compared with the baseline file over 32 sampled entries, and the gate passed with no mismatch.

order	row	seconds	reads	reads/event
sequential	store mapped	0.003	2,000	1.00
sequential	store block	0.005	2,000	1.00
sequential	store entry	0.109	4,000	2.00
sequential	store block entry	0.182	4,000	2.00
sequential	baseline	0.453	7	0.004
sequential	baseline+cache	0.468	16	0.008
scattered	store mapped	0.000	2,000	1.00
scattered	store block	0.031	2,000	1.00
scattered	store entry	0.101	4,000	2.00
scattered	store block entry	0.586	4,000	2.00
scattered	baseline	202.209	1,365,153	682.6
scattered	baseline+cache	204.010	16	0.008

The entry rows settle the claim at the layer the baseline reads at: reading the whole event costs 0.101 s scattered against 0.109 s sequential, two reads per event, so the store's cost does not move with the order, and against the baseline's 202.2 s over the same events that is 2,000 times, payload for payload and delivery for delivery, or 345 times for the compressed store. The store's per-event cost is 50 microseconds, the figure the whole-event scan rows measure too, which places the store's remaining cost in the delivery layer rather than in the read path; the dataset subsection measures that layer's scale condition.

The scatter penalty is a basket boundary, not a slope. The first basket of the measured file covers entries 0 to 999, and a scattered list inside it reads like a scan, 6 requests at 500 and at 1,000 events, 0.076 to 0.093 s; a list that spans a

second basket collapses, 1,365,153 requests at 2,000 events, because a jump between baskets invalidates the current basket of each of the 1,380 branches. The store's count is flat across that threshold.

The cache is not the answer under scatter. A 64 MB TTreeCache collapses the read calls to 16 and leaves the time where it was, 204.0 s against 202.2, so the scattered cost is the traversal and the decode of every event, and a cache cannot hold the working set. The store removes that cost by addressing.

8.4 The coordinate addresses a dataset

The step after the phases above asks whether the coordinate reaches a dataset rather than one store. The collection store is split into one shard per run, each shard declaring the luminosity blocks its run holds and the greatest block count, and the same partition is held as ROOT files with the manifest both sides share. Measured on the M1 file, 38 runs, the largest run 281,515 events, sources warmed:

subject	operation	seconds	note
store	attach	0.687	18.1 ms per file, descriptor read, parse, map
chain	attach	0.002	0.05 ms per file, names only
chain	headers forced	0.355	9.3 ms per file, the cost the store's attach pays up front
chain	select by run	11.1	no run index, the range comes from a scan of the run branch
manifest	read	10.3	one file, all branches, no branch addresses, 36 microseconds per event
store	read path	1.6	942,333,903 bytes at 600 MB/s, no delivery
store	entry layer	29.0	78.9 microseconds per grid cell over 368,008 cells, 54.5 over the first 2,000

The coordinate resolves a run to its shard with no scan, which is the dataset level claim: the chain (TChain over the ROOT shards) has no run index, so the same question costs it a scan of the run branch, 11.1 s over the 38 files. Two costs on the store side are recorded rather than argued: the descriptor is parsed per file and repeats the field table the shards share, 18.1 ms against the chain's forced 9.3 ms, and the dense grid, the cells the shard axes lay out, carries 2,862,780 cells for 2,315,223 events, 1.24 times, the largest run 368,008 for 281,515, 1.31; the cells the events do not occupy, the padding, are 19.1 percent of it, and the walk pays them as well as the bytes. The entry layer is 2.8 times the chain's per-event read over the same run, and its per-cell cost rises with the scale of the walk, 54.5 microseconds over the first 2,000 cells against 78.9 over 368,008, a rise this run does not attribute. The axes are slots while the record carries the physical run, luminosity block, and event number: over the physical luminosity block values a dense axis would carry 14,710,139 cells, 84.3 percent of them padding, 5.14 times the grid the slot axes lay out, and the alternative is a block table of 2,348 entries, so where the slot-to-physical mapping lives is open.

8.5 Thread scaling

Every worker with its own file object and a disjoint range, the store row through the byte source and the baseline row through TTree::GetEntry with no branch addresses, on the same disk:

threads	store_s	store_MB/s	baseline_s	baseline+cache_s	baseline/store
1	13.04	543	186.4	183.8	14.3
2	9.86	718	93.7	92.7	9.5
4	5.32	1331	50.3	53.7	9.4
8	2.86	2481	31.9	32.2	11.2

Both scale, the store 4.6 times and the cache-less baseline 5.8 over eight threads, so the store's lead narrows from 14.3 to 11.2 times rather than widening: the store is bandwidth bound and the baseline I/O bound. The cache neither helps nor hurts, because each worker reads a contiguous range and the cache is redundant under sequential access, so the concern that threads degrade a shared cache is not reproduced here; a shared cache or a scattered, remote pattern would be needed, and that pairing is open. The machine has ten cores, so the row ends at eight, and the 128-core measurement itself needs a larger system and is not taken.

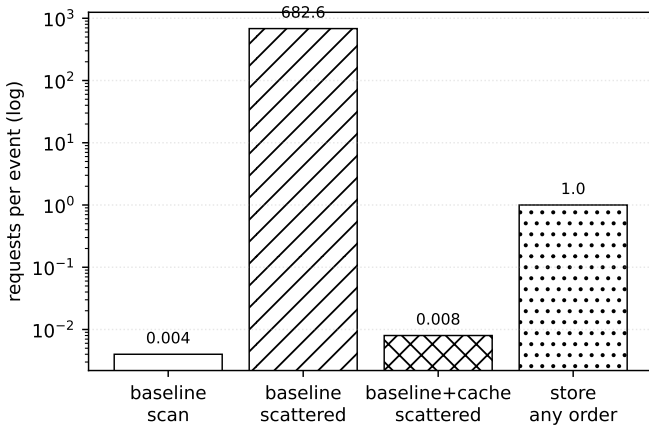


Figure 7: Event-selected reads, 2,000 events, same medium, requests per event. The store answers each event in one or two requests whatever the order; the baseline pays 683 requests per event when the order is scattered against 0.004 requests per event when it is a scan, and a 64 MB cache collapses its calls to 0.008 requests per event without moving its time, 204.0 s against 202.2.

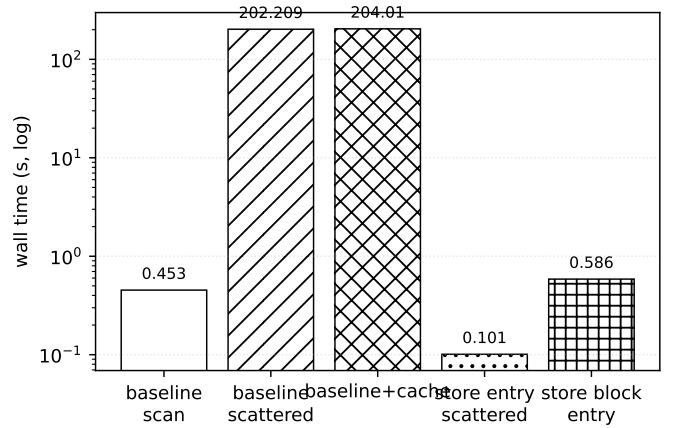


Figure 8: The same events by wall time, log scale. The baseline runs 202.2 s scattered against 0.453 s over a scan, the cache leaves it at 204.0 s, and the store reads the whole event in 0.101 s, 0.586 s through the block source, against 0.003 s for the index record alone.

What the small sample gives is a deterministic prediction. The store answers every event in one or two requests whatever the order and whatever the thread count, while the baseline's scattered request count collapses once the entry set spans a basket, 6 requests at 500 and at 1,000 events inside the file's first basket against 1,365,153 at 2,000, because a jump between baskets invalidates the current basket of each branch and a cache cannot hold the working set. The request count is media-independent, so the tendency is settled at the scale measured and it does not close with more cores: the store's per-event cost stays one or two requests and the baseline's stays hundreds. Only the wall-time magnitude at 128 cores depends on that machine's bandwidth, and that needs the machine.

9 Cost Analysis

The measured results separate two bottleneck regimes. On the remote EOS baseline, the bottleneck is per-request I/O wait (about 98.8 percent of wall time); the coordinate path removes requests and, when mapped, read system calls. On the local full-file baseline, the bottleneck is decompression CPU (cpu 183.6 to 185.9 s of wall 187.0 to 188.2 s, 98.2 percent); the coordinate store holds raw fixed-width records and removes decompression from the read path. Both removals are structural rather than incremental: the request and system call counts are deterministic properties of the addressing scheme, and the raw record layout removes the decompression step by construction.

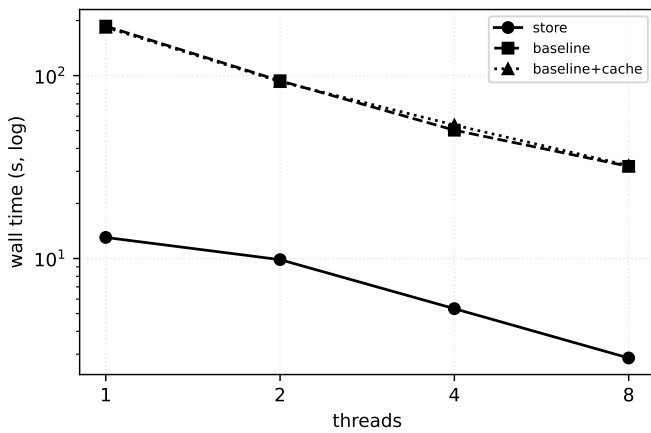


Figure 9: Thread scaling, the store against the baseline, every worker with its own file object and a disjoint range. Both scale, the store 4.6 times and the cache-less baseline 5.8 over eight cores, so the lead narrows from 14.3 to 11.2 times; the cache neither helps nor hurts.

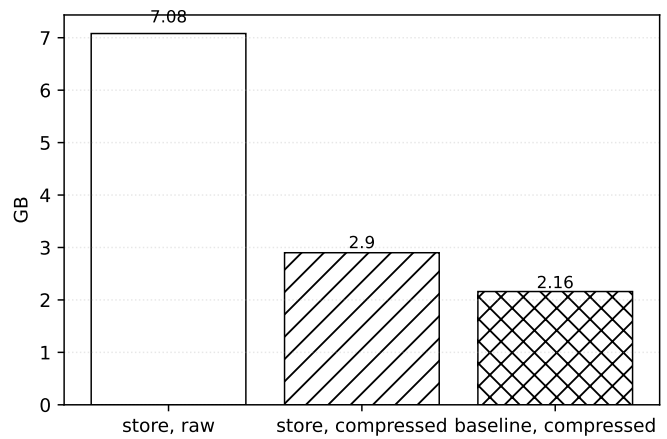


Figure 10: The store under compression, 256 KB blocks, zlib level 1. The store's raw payload, 7,082,290,985 bytes, closes to a 2,897,372,833-byte compressed store, 2.44 times its own raw form, while the baseline file is 2,155,974,646 bytes, so the compressed store remains 1.34 times the baseline file. The read path on its own is 12.4 s at 572 MB/s against the raw store's 14.6 s and the baseline's 177.4 s. An 8 KB block cuts the scattered read to 16 microseconds per event against 222 and costs almost nothing, 2.44 to 2.37 times.

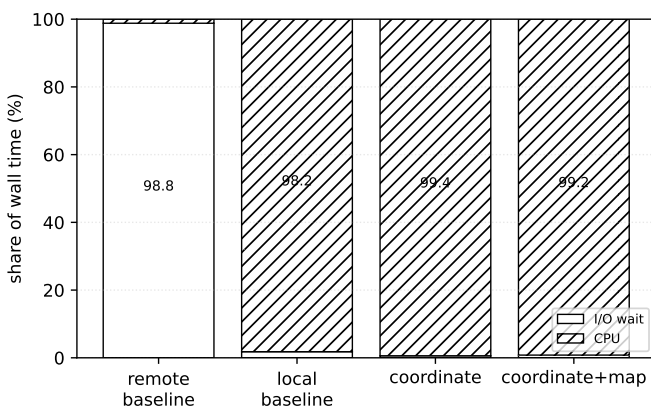


Figure 11: Wall-time composition across the measured paths. The remote baseline is I/O-wait bound; the local baseline and the coordinate paths are CPU bound, with decompression dominating the local baseline and the mapped path running entirely in CPU.

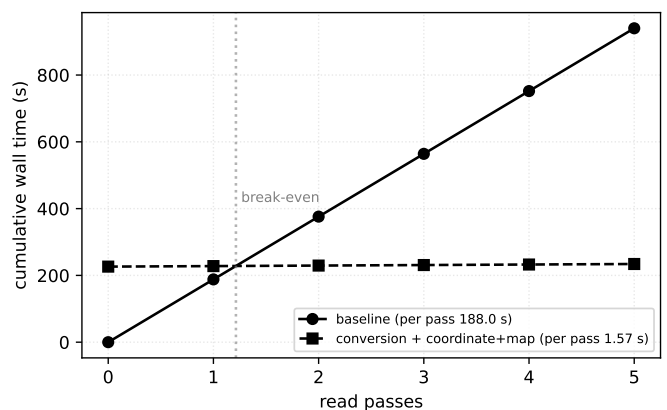


Figure 12: Cumulative wall time over repeated read passes. The one-time conversion (226.3 s) amortizes against the per-pass baseline cost (187.0 to 188.2 s); the mapped coordinate path breaks even by the second pass and then accrues the gap per pass.

At full scale the baseline request count drops to 0.20 per event because basket buffers stay resident in memory after the first cluster¹⁸; the full-file speedup is therefore dominated by decompression removal and aligned sequential access, while the request-count reduction from 1.37 to 1.00 per event is the M1 signature slice and the remote case.

The bytes-per-read asymmetry is intentional. The baseline reads 2,149,961,076 compressed bytes in 470,131 requests; the coordinate paths read 5,926,970,880 raw bytes in 2,315,223 requests. The coordinate paths move 2.76x more bytes and still finish 67x to 69x and 117x to 121x faster on the full file, where decompression removal and aligned sequential access dominate, and the mapped path removes the application-level read system call entirely. Compression semantics remain a later-phase item, specified in the Status section.

The speedup ratios vary with the condition, from 1.6x to 5.3x on the synthetic microbenchmark to about 150x to 240x on the same-media slice, because the baseline cost composition changes with the access pattern and cache state, and both those rows sit at a scale where per-syscall cost and timer resolution dominate. The stable claims are the request count (1.37 to 1.00 per event) and the system call count (1.37 to 0.00 per event), which are media-independent. All wall-time ratios are scoped to the access pattern, the cache-disabled baseline, the same medium, and the warm-source protocol of the measured results.

9.1 Conversion

The one-time conversion of the full dataset into the store costs 226.3 s wall, 220.5 s CPU, producing 5,926,970,880 bytes across 2,315,223 records. The conversion is comparable to one baseline read pass (187.0 to 188.2 s) and amortizes: after conversion, every read pass costs 1.55 to 1.61 s (mapped) instead of 187.0 to 188.2 s, so the break-even point falls within the second analysis pass.

10 Correctness Verification

The checksum verification closes the loop between the converted store and the served bytes: the sum over every record byte at conversion time equals the sum over the bytes the coordinate paths serve, the value reported in the measured results. The analysis workload verifies semantic equivalence: the same selection and histogram over both paths produce identical selected counts and identical histogram means. The unit tests additionally verify the store arithmetic over the full 19 by 21 by 28 lattice¹⁹ against the canonical engine, the memory-backed read path (payload correctness with zero application-level read system calls), and the layout rejection paths. On the dataset side, the shard tool's self-check resolves 2,348 slot coordinates, one per block, and reads each record back with no mismatch, and the unit tests cover the partition by run and luminosity block (`gtest-io-io-tagma-dataset`, `PartitionsOneRunByLuminosityBlock`).

11 Status

11.1 Phase 1: complete

The fixed-width coordinate read path is implemented, built into the fork, and verified end to end. Phase 1 scope:

- Fixed-width event records addressed as a coordinate and resolved by closed-form arithmetic; the measured rows populate the event axis alone, and the multi-axis composition is verified against the reference engine over the canonical lattice without being exercised by a dataset-level layout
- The read path replaced behind the existing TTree and TFile interfaces; the researcher API, analysis code, and build chain are unchanged

¹⁸TBranch keeps the current basket buffer in memory; sequential entries within a cluster reuse it, so the cache-disabled baseline reads 0.20 times per event at full scale. The 1.37 to 2.76 reads per event signature is the cold-cache and cluster-boundary regime.

¹⁹The lattice has 19 by 21 by 28 cells, equal to 11,172, the canonical scale. The test compares TTagmaStore::Compose against the canonical reference engine over every cell.

- Measured on the full CMS Run2016G DoubleMuon NanoAOD first file against the cache-disabled baseline: 66.9x to 69.1x (coordinate) and 116.6x to 120.5x (mapped, zero application-level read system calls), with a one-time conversion of 226.3 s. Those ratios are against the all-branch baseline; the component controls are in the measured-results section, and the payload-equal figure is 13.1x with 22.8x mapped
- Correctness verified by a checksum over the served bytes and exact analysis-result equality, plus unit tests against the reference engine over the full lattice
- Released as open source in the [ssccsorg/root](https://github.com/ssccsorg/root) fork, with the benchmark harness and the runbook

11.2 Phase 2 to 6: delivered

The phases below are delivered in the fork, and the measured record is the section Measured results, phases 2 to 6. The statements below were written while the phases were planned and are kept as the plan of record.

The component controls have run. Decompression removal accounts for 2.20x to 2.31x of the measured ratio on the full file and the read path for the rest. The store's advantage is column independence rather than a universal speedup: 13.1x against an uncompressed baseline reading the same columns, 40.5x against the compressed baseline reading those columns with the cache enabled, and 4.9x slower than RDataFrame on the two-column selection of the analysis workload. The crossover is near 11 columns, or near 6 with the store mapped. The phase that moves that boundary is the first one below, because 276 of the store's own 320 fields are arrays that the projection truncates to their leading element.

- P2: variable-length collections (jets, tracks). The store must support per-event record lengths; the fixed-width record is the addressing foundation, and this is the phase that decides whether the projection can carry the collections rather than their leading value
- P3: compression semantics. The raw phase-1 store moves 2.76x more bytes than the compressed baseline; block-level or selective compression must preserve direct record addressing at the PB scale
- P4: production-scale end-to-end measurement. The 14-hour workload, including the remote access pattern, is not reproduced; the full local file is the current measured scale
- P5: cache-degradation scenario. Out-of-order multithreaded reads and the beyond-100,000-cluster cache collapse are not reproduced; the measured 0.917 cache efficiency is the ideal sequential case
- P6: portability and independent verification. The mapped path requires a Unix-like platform; a Linux build cross-check is planned

Dataset level, delivered and measured. TTagmaDataset partitions the run and luminosity block axes across store files; tagma_shard.C writes one shard per run, each declaring its own axes, tagma_root_shard.C splits the ROOT tree into one file per run with the manifest both sides share, and tagma_dataset_bench.C measures attach, select, and read over the partition. The coordinate resolves a run to its shard with no scan, and the chain has no run index, so the same question costs it a scan of the run branch, 11.1 s over the 38 files. The measured rows are the dataset subsection of the results above.

12 Boundaries

- The mapped path requires a Unix-like platform (mmap)
- The source file (2.16 GB) and the store (5.93 GB) both fit the measured machine's page cache, so the mapped row reports a memory copy of resident data; a sample larger than memory is required before any mapped row can be read as a storage claim
- The store serves a fixed-width projection: at most 320 of the tree's 1,380 leaves, of which 44 are scalar leaves and 276 are variable-length arrays truncated to their leading element, while the baseline reads every branch
- The store's cost per event does not fall with the number of columns an analysis selects, so it is faster than a column-selective reader above a crossover near 11 columns, or near 6 with the store mapped, and slower below it
- The dataset rows cover one partition: 38 runs of one file, one shard per run, on the warmed local medium

- The dataset's run and lumi axes are slots while the record carries the physical values, so the grid the slot axes lay out is 19.1 percent padding; where the slot-to-physical mapping lives is open, and a dense axis over the physical luminosity block values would be 84.3 percent padding, 5.14 times that grid
- The entry layer's per-cell cost rises with the scale of the walk, 54.5 microseconds over the first 2,000 cells of a shard against 78.9 over 368,008; the rise is not attributed
- All results are measured and reported as they are, with no projection ahead of measurement
- The demonstration complements ROOT and RNTuple; the existing read paths remain in place

13 Conclusion

The coordinate-indexed read path replaces branch, basket, and cache traversal with closed-form arithmetic, and after the later phases the measured picture is sharper than the phase-1 projection it started from. The store carries the whole event, 974 scalars and 19 collections at two reads per event. It compresses, 2.44 times, with the read path at 572 MB/s. And the regime this report motivates, an analysis reading selected events in list order rather than a scan, is where the mechanism shows: the store reads the whole event, fields delivered, in 0.101 s scattered against 0.109 s sequential over 2,000 events, two reads per event, while the baseline pays 683 requests per event and 202.2 s in the scattered order against 0.004 requests and 0.453 s over a scan. That is 2,000 times, payload for payload and delivery for delivery, or 345 times for the compressed store, against 12 to 14 times on the scan's read path. The scatter penalty is a basket boundary the store's flat request count does not share.

The phase-1 rows remain the measured record of the projection, and that depth is real: 2.71 to 2.81 s against 187.0 to 188.2 s, or 1.55 to 1.61 s mapped, with the same controls and the same boundaries. What the later phases change is the scope of the claim. The whole-event read is 1.5 times, not tens of times, because the delivery layer costs about 50 microseconds per event at single-file scale, and the dataset subsection measures how it moves with the walk's scale. A narrow column selection is upstream's to win below the column crossover, near 42 scalar columns for the read path and near 810 with delivery. A cache does not rescue the scattered baseline, but thread scaling does narrow the lead, 14.3 to 11.2 times over eight cores. And O(128), multi-terabyte samples, and the remote medium need infrastructure the measurement machine does not have. The 128-core row is therefore not measured, but it is predicted and not guessed: the per-event request count is media-independent and settled at the scale measured, one or two for the store whatever the order against hundreds for the baseline once its entries span a basket, so the tendency holds at 128 cores and the gap does not close. Only the wall-time magnitude there depends on the machine's bandwidth and needs that machine.

The coordinate also reaches a dataset rather than one file. One shard per run, 38 runs, and a run resolves to its shard with no scan, where the chain has no run index and scans the run branch for 11.1 s; the attach cost is 18.1 ms per file against the chain's forced 9.3 ms, and the dense grid carries 1.24 times the cells the events need, which the walk pays as well as the bytes. Two things stay open: the entry layer's per-cell cost rises with the scale of the walk, 54.5 microseconds over a shard's first 2,000 cells against 78.9 over 368,008, and the axes are slots while the record carries the physical values, so where the slot-to-physical mapping lives decides what a dense axis over the physical luminosity block values would cost: 5.14 times the grid the slot axes lay out, 84.3 percent of it padding.

The same shape was measured at another layer, on this machine: a conjunctive resolution over a coordinate product space holds at 12.7 to 14.3 microseconds per query from 1,000 to 200,000 entries while its reference scan rises from 1.5 to 298.2, crossing near 10,000.²⁰ The mechanisms differ and the magnitudes do not transfer; the cross-check corroborates the shape of the claim, not this read path.

The numbers are media-independent where they are counts, and the counts are where the mechanism lives. The fork, the harness, and the runbook are the starting point: `benchmarks/tagma/tagma_bench.C` is the entry point, `tagma_harness.C` holds the measured record, whose dataset rows come from `tagma_shard.C`, `tagma_root_shard.C`, and `tagma_dataset_bench.C`, and `FRONTS.md` holds the open fronts, a column-major layout to move the crossover, the delivery layer, the walk's per-cell cost, and the revision of the claim itself.

²⁰The board is nex-derive, the FIH goal space of the nexus project (docs.sccs.org/projects/nexus/apps/derive): a conjunctive query is a word-wise intersection over a fixed coordinate product rather than closed-form byte addressing over a store, and the board lives in memory. The matched counts reproduce the published page, and the rerun was on the same machine as the rows above.