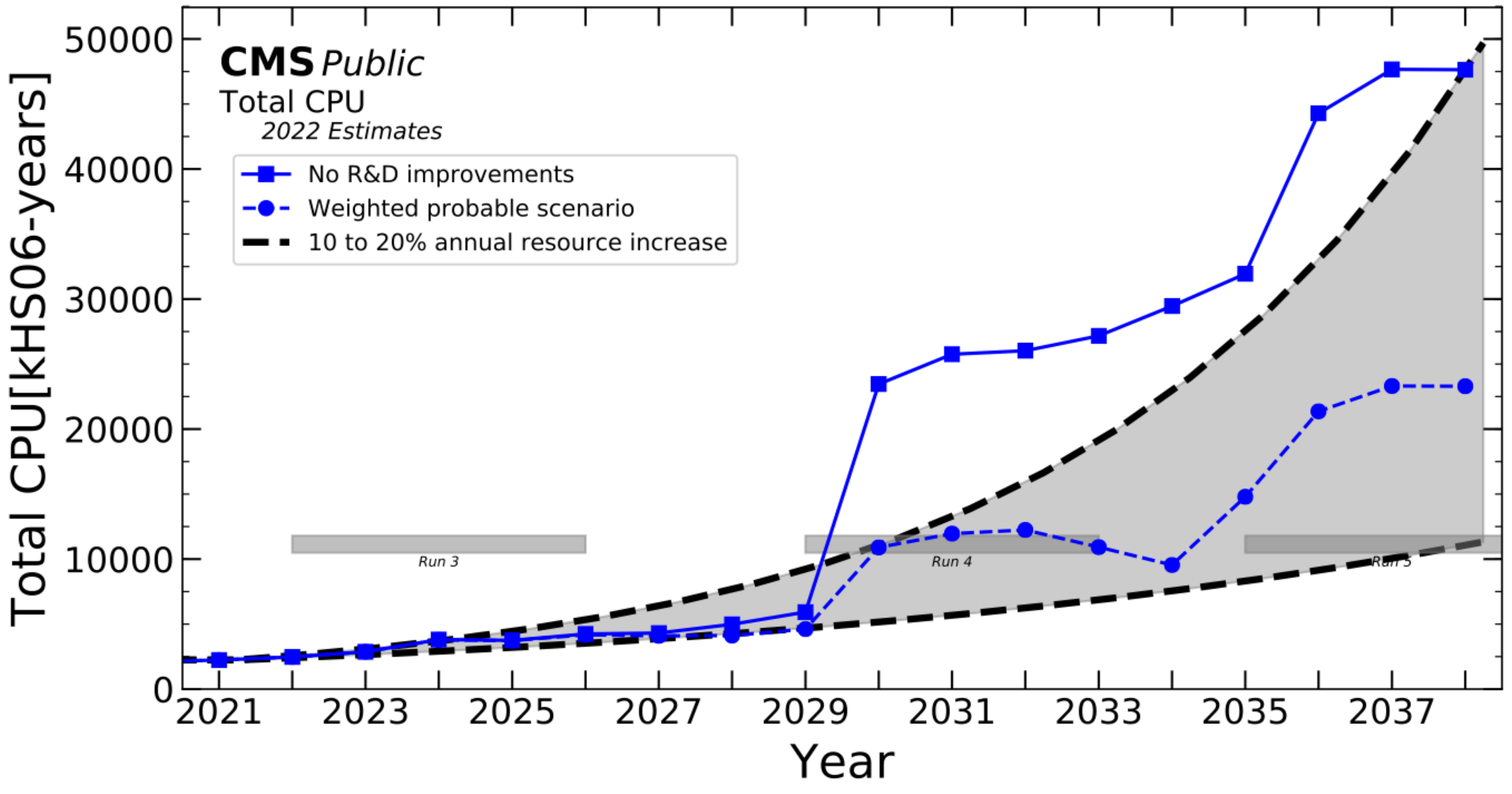
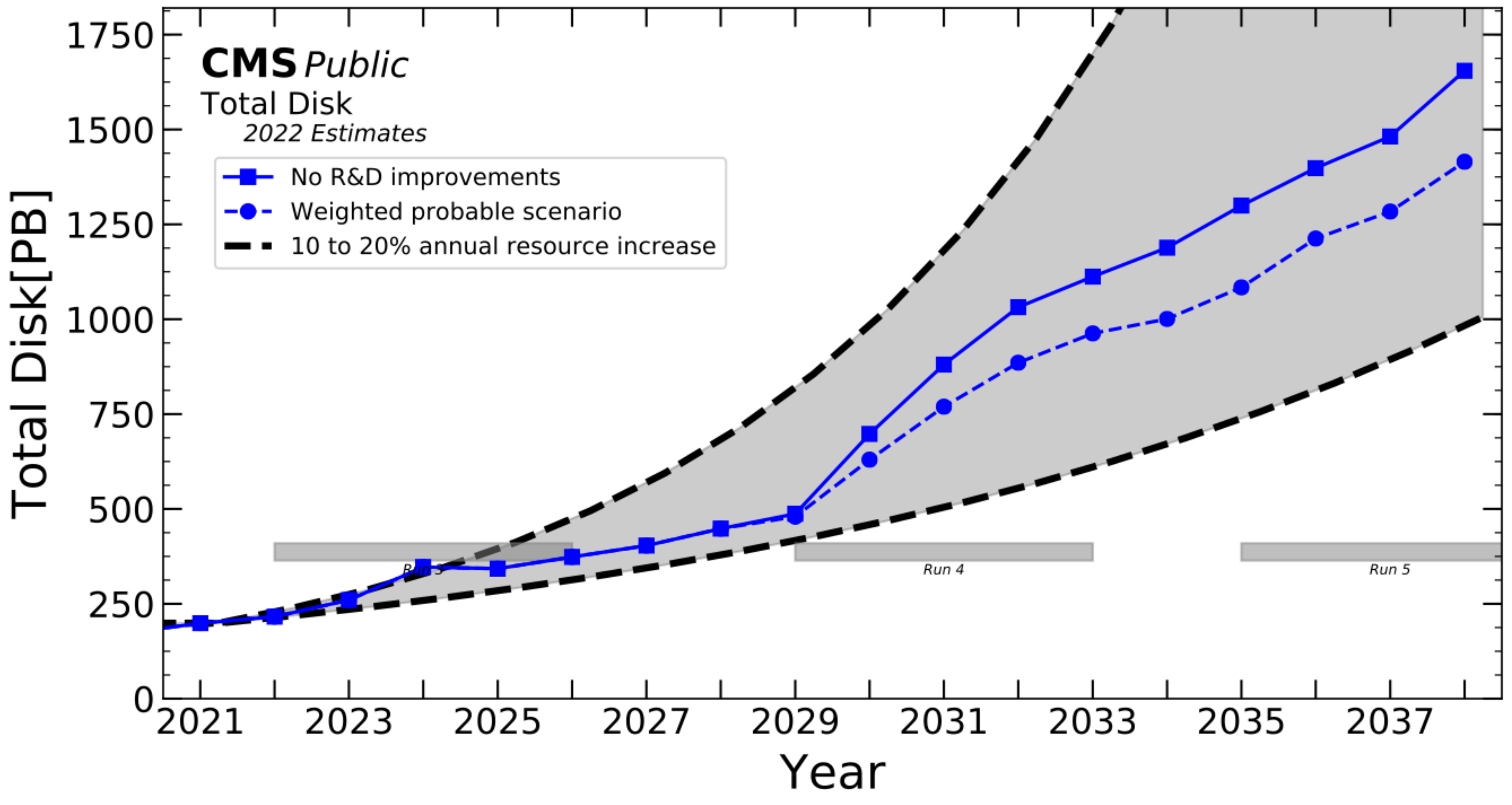
On-Demand Column-Joining for End-User Analysis

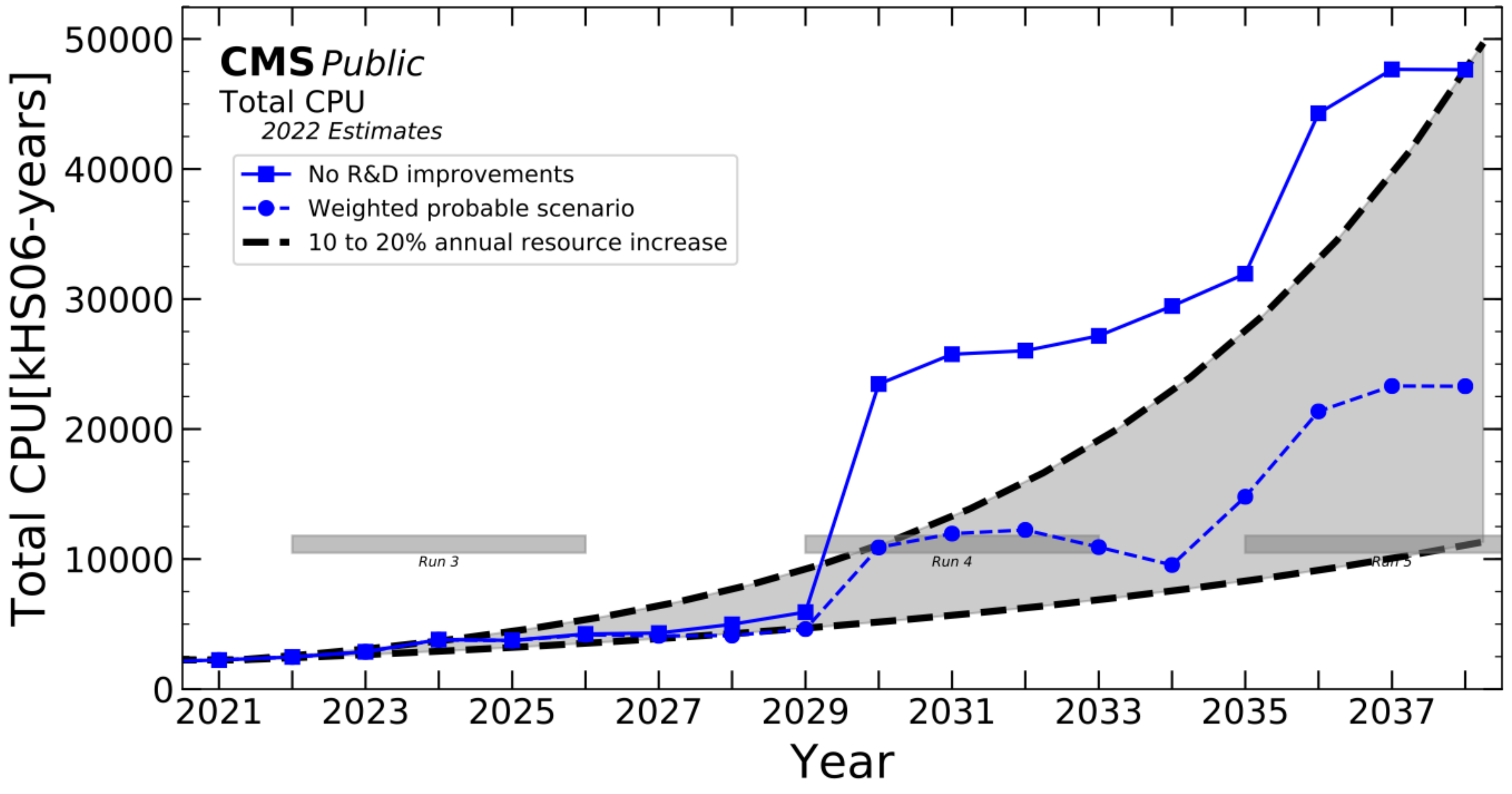
Nick Manganelli, Ben Galewsky, Burt Holzman
Lindsey Gray, Keith Ulmer

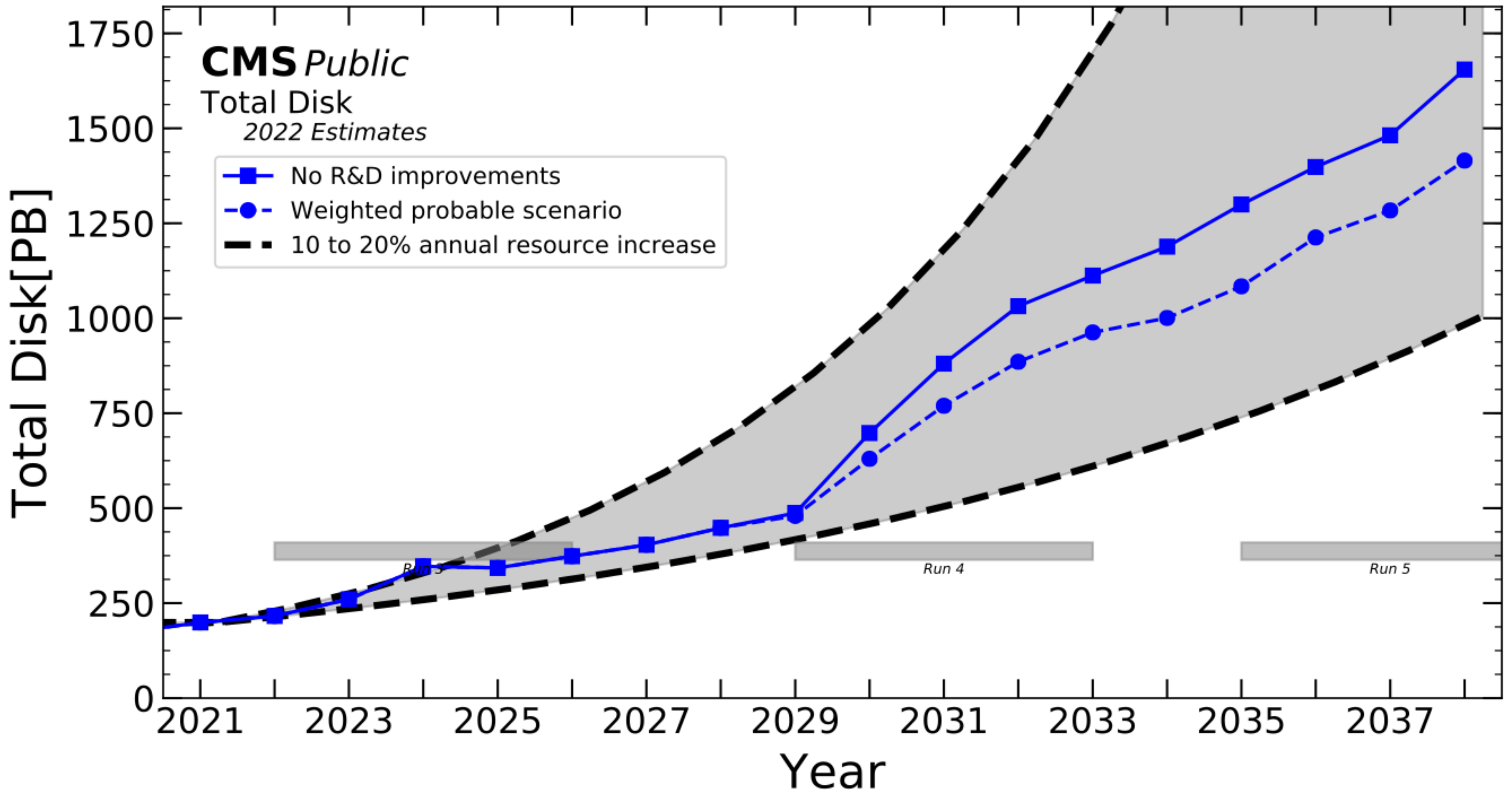
Conference on Computing in High Energy and Nuclear Physics
October 21 - 25 Krakow, Poland

Motivation


CMS Public
Total CPU
2022 Estimates
No R&D improvements
Weighted probable scenario
10 to 20% annual resource increase
Total CPU[kHS06-years]
Year
Run 3
Run 4


CMS Public
Total Disk
2022 Estimates
No R&D improvements
Weighted probable scenario
10 to 20% annual resource increase
Total Disk[PB]
Year
Run 4
Run 5

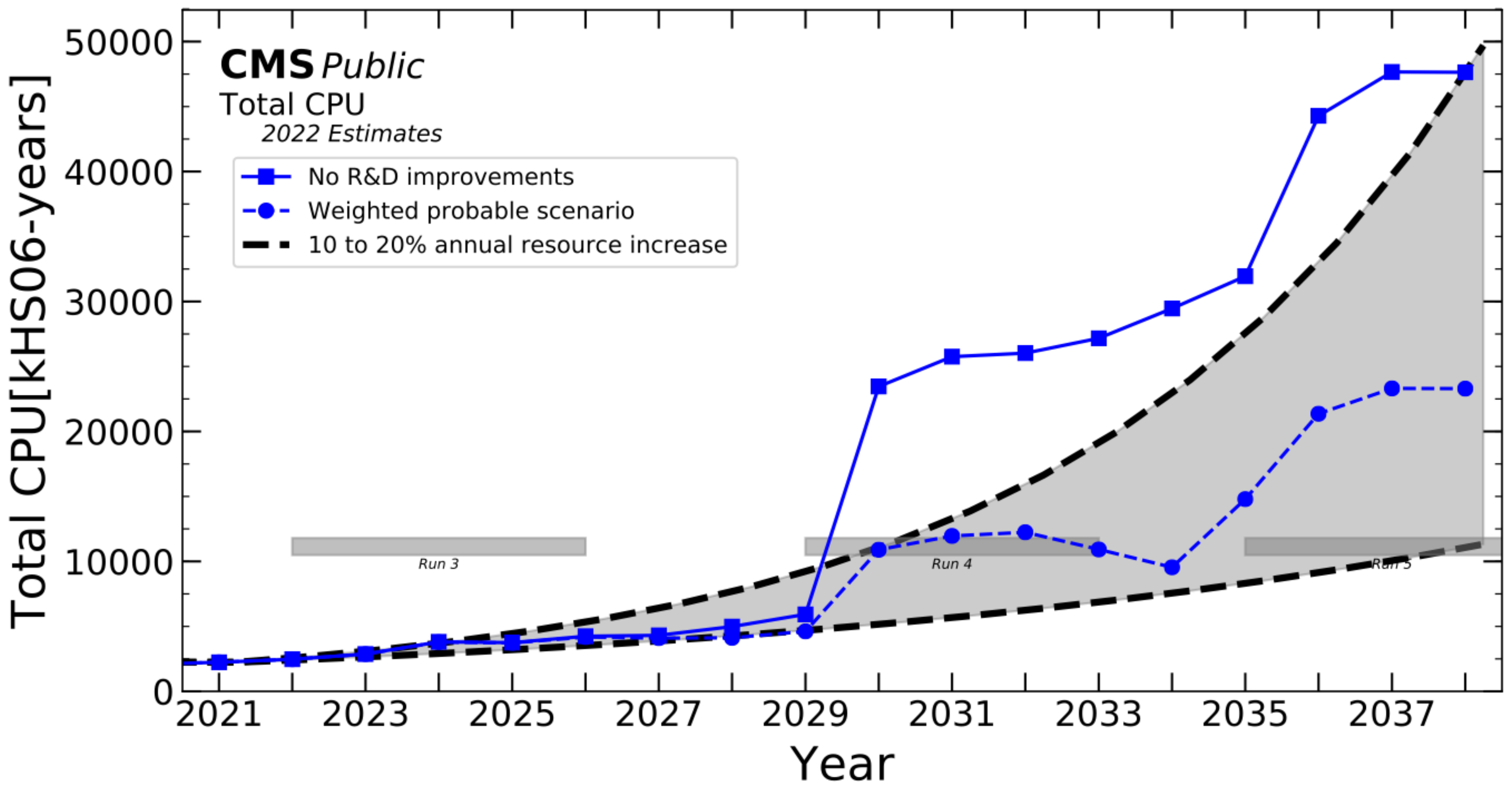


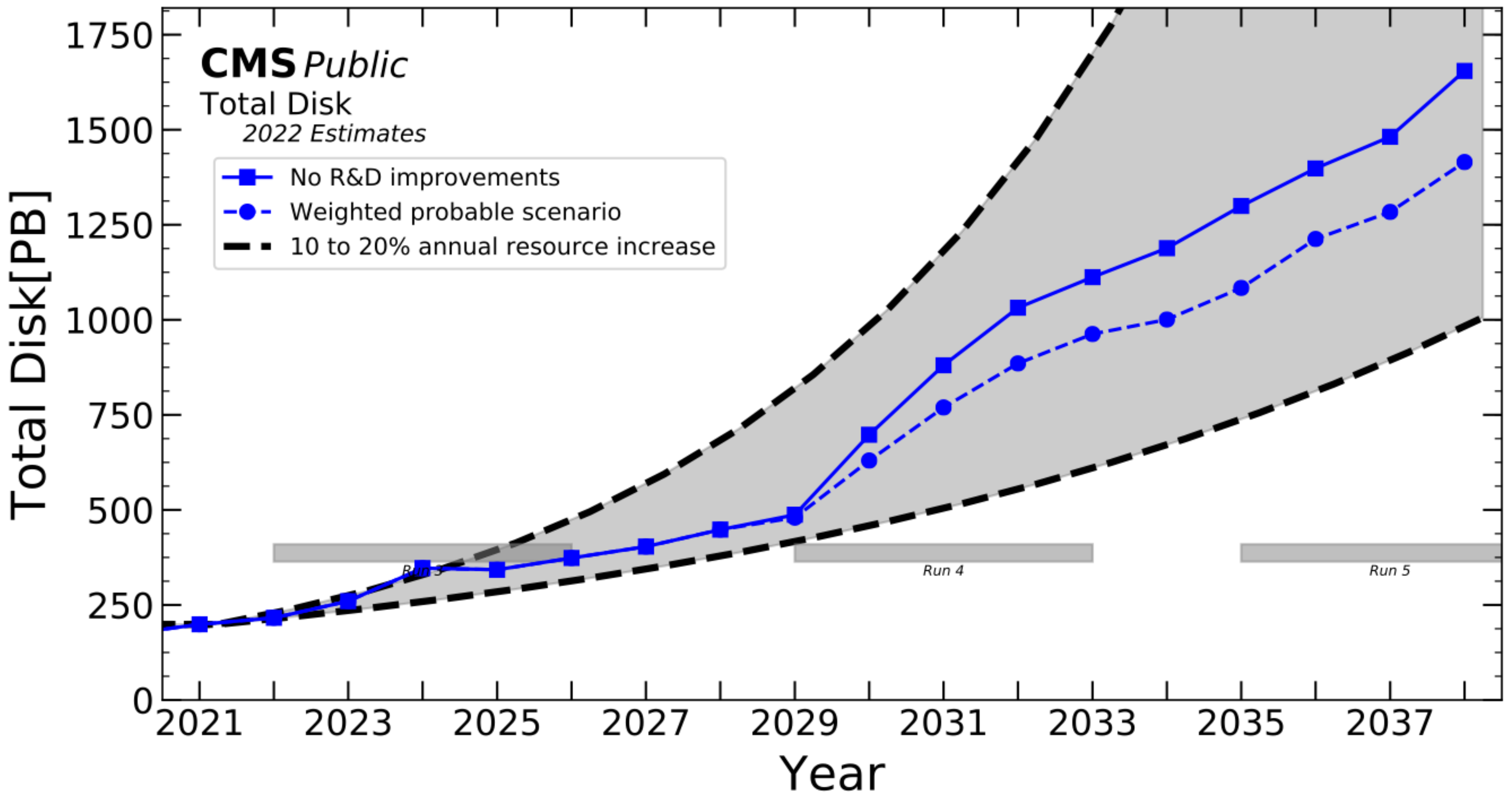


Motivation

- As the LHC moves into the HL-LHC era, the **volume of data to be stored and processed will grow significantly** (x10)
  - CMS AOD (450kB/event) - Limited Availability, high processing costs, data: C++ classes
  - MiniAOD (45kB/event) - Suitable for nearly all analyses, still large, still significant processing, data: C++ classes
  - NanoAOD(4kB/event) - Suitable for half of analyses, **analysis-ready**, **data: primitives**

Motivation





- As the LHC moves into the HL-LHC era, the **volume of data to be stored and processed will grow significantly** (x10)
  - CMS AOD (450kB/event) - Limited Availability, high processing costs, data: C++ classes
  - MiniAOD (45kB/event) - Suitable for nearly all analyses, still large, still significant processing, data: C++ classes
  - NanoAOD(4kB/event) - Suitable for half of analyses, **analysis-ready**, **data: primitives**
- Several competing needs create an impedance mismatch
  - Disk space comes at a premium
  - High throughput requires high availability and duplication across sites around the world
  - Traditional analysis workflows tend to **duplicate information** from large data-tiers (Mini/AOD) via custom “Ntuples”, in order to create more streamlined but self-contained input data - for the half of analyses able to use NanoAOD (a “generalized” ntuple), it’s nearly optimal and can obviate the need for intermediate Ntuples
  - Fast turnaround (for analyzers) is paramount to getting the science **done**!

Data Duplication in a Typical Analysis

Data Duplication in a Typical Analysis

- An analysis may be able to use NanoAOD(-like inputs), but must store expensive ML outputs
  - Typical approach: **Duplicate all necessary input data from NanoAOD + added ML** information into a custom NanoAOD (May permit dropping columns or certain events, but these decrease flexibility)

Data Duplication in a Typical Analysis

- An analysis may be able to use NanoAOD(-like inputs), but must store expensive ML outputs
  - Typical approach: **Duplicate all necessary input data from NanoAOD + added ML** information into a custom NanoAOD (May permit dropping columns or certain events, but these decrease flexibility)
- An analysis may have 90% of data needs met by NanoAOD, but the additional requirements drive it to use MiniAOD or AOD
  - Custom NanoAOD variant (superset of central variation) or custom NTuple format created from larger datatier (labor and compute-intensive), **duplicating** a significant amount of centrally-stored events in Nano and Mini formats (inefficient disk utilization)

Joins: A Way Forward

Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication

Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**

Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)

Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)

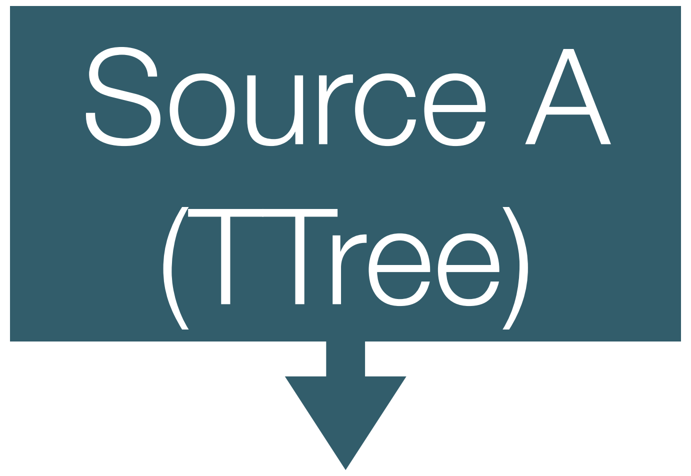
Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)

Joins: A Way Forward

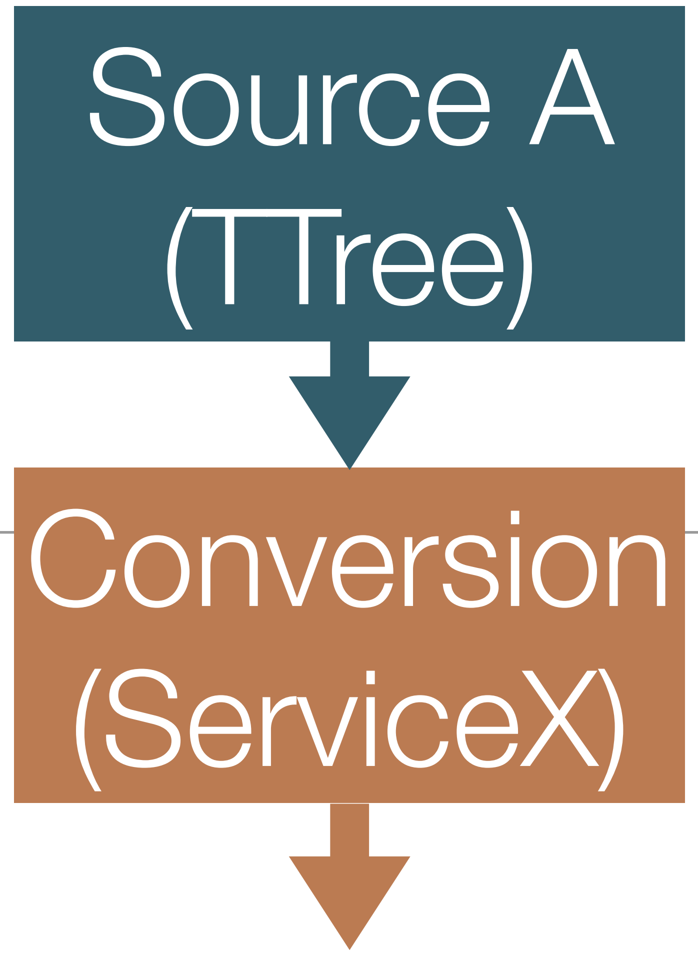
- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem

Joins: A Way Forward



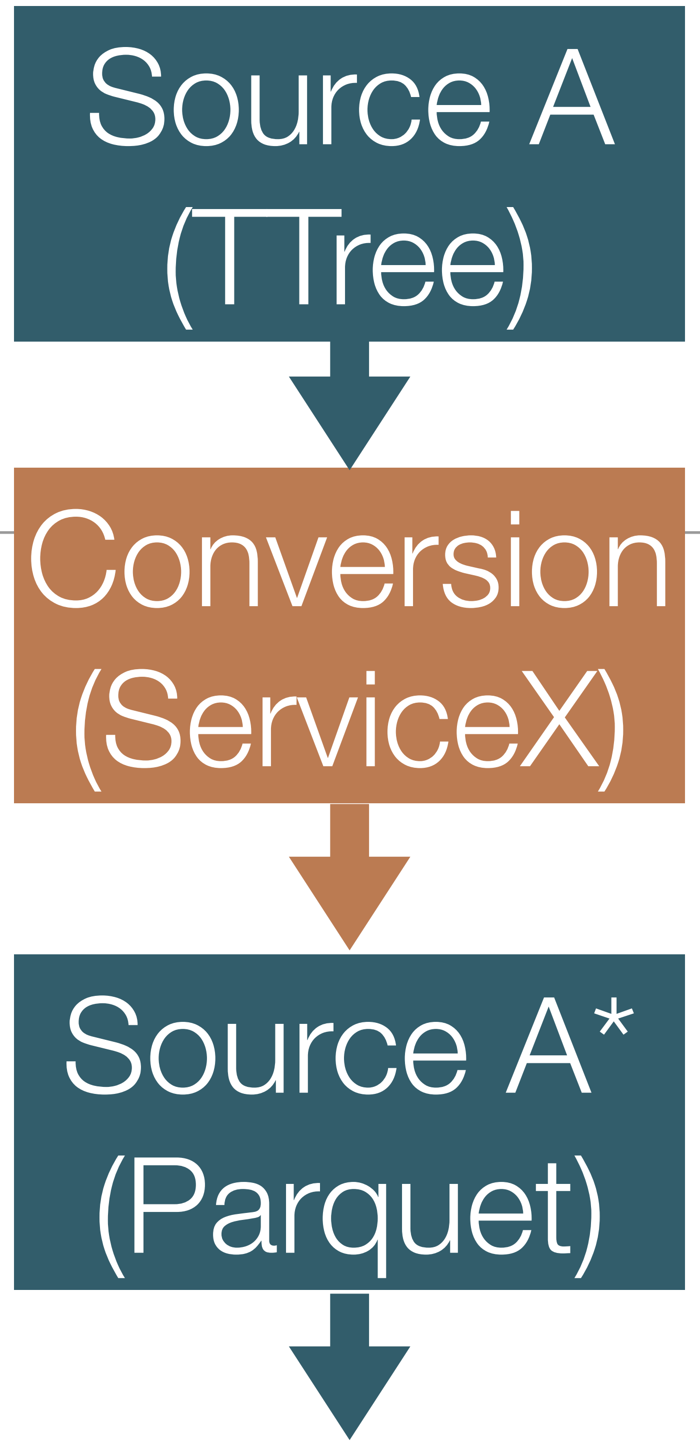
- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem

Joins: A Way Forward



- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem

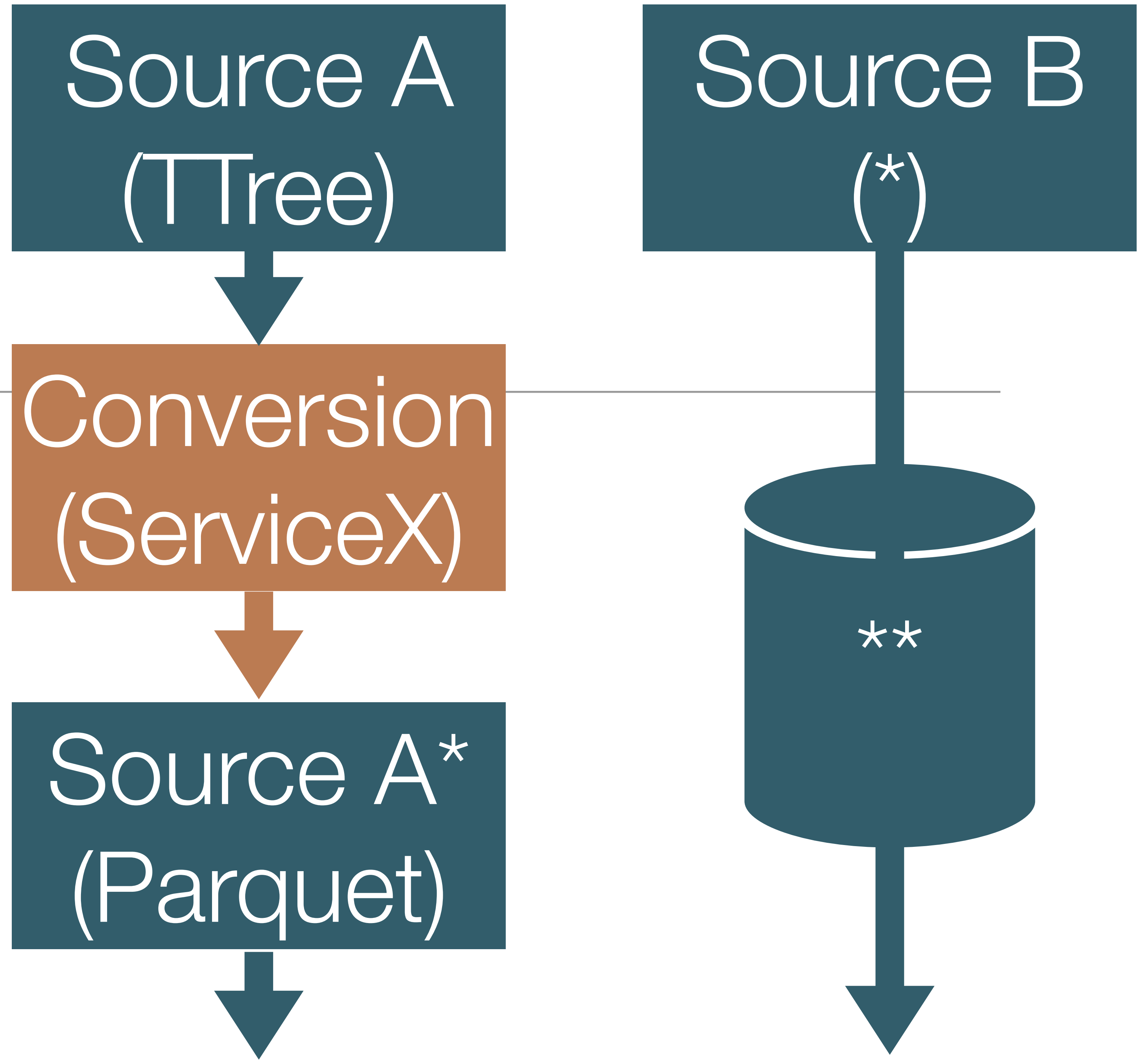
Joins: A Way Forward



- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem
- * Transient converted dataset ** Transformation as necessary

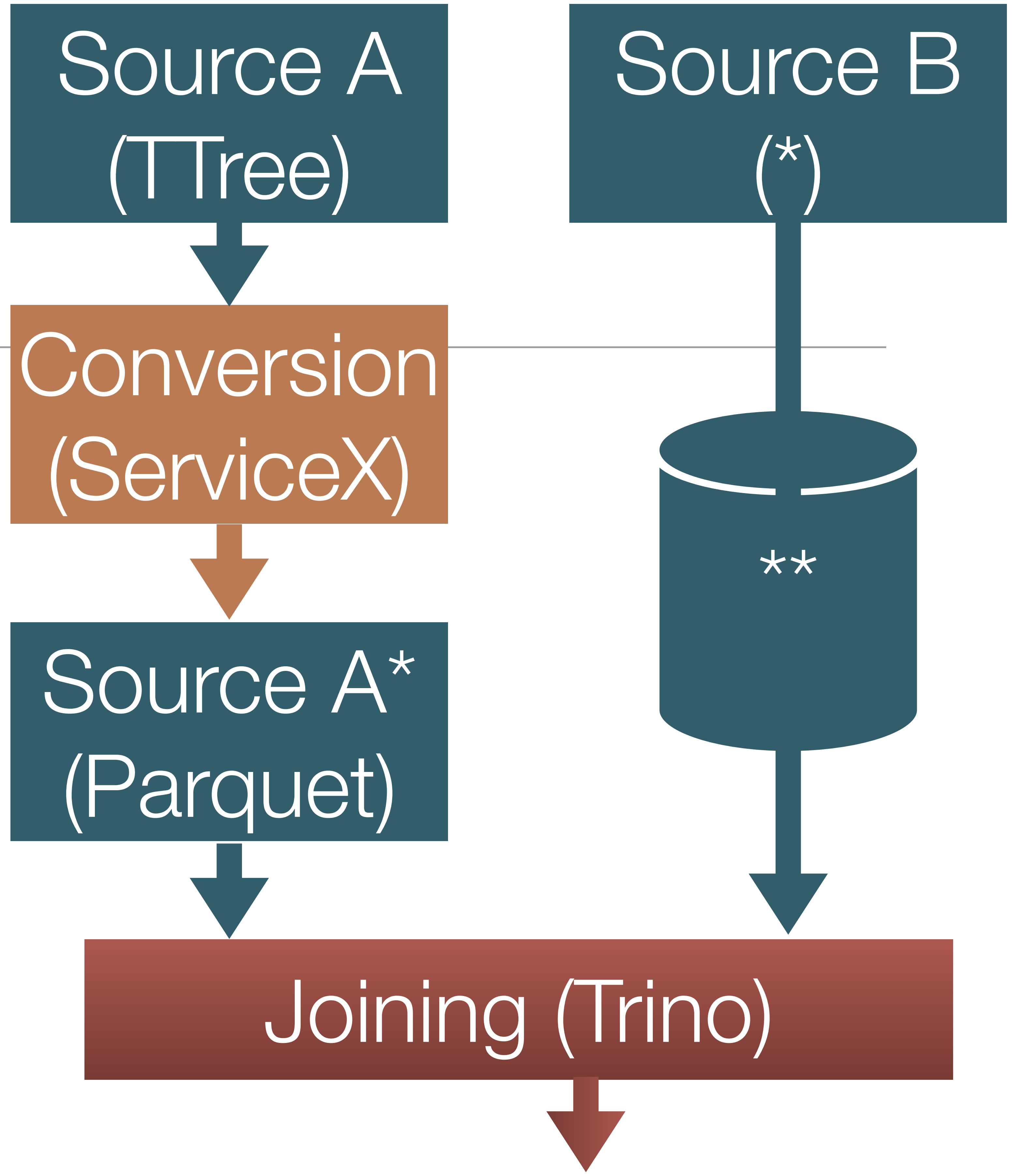
Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem
- * Transient converted dataset   ** Transformation as necessary



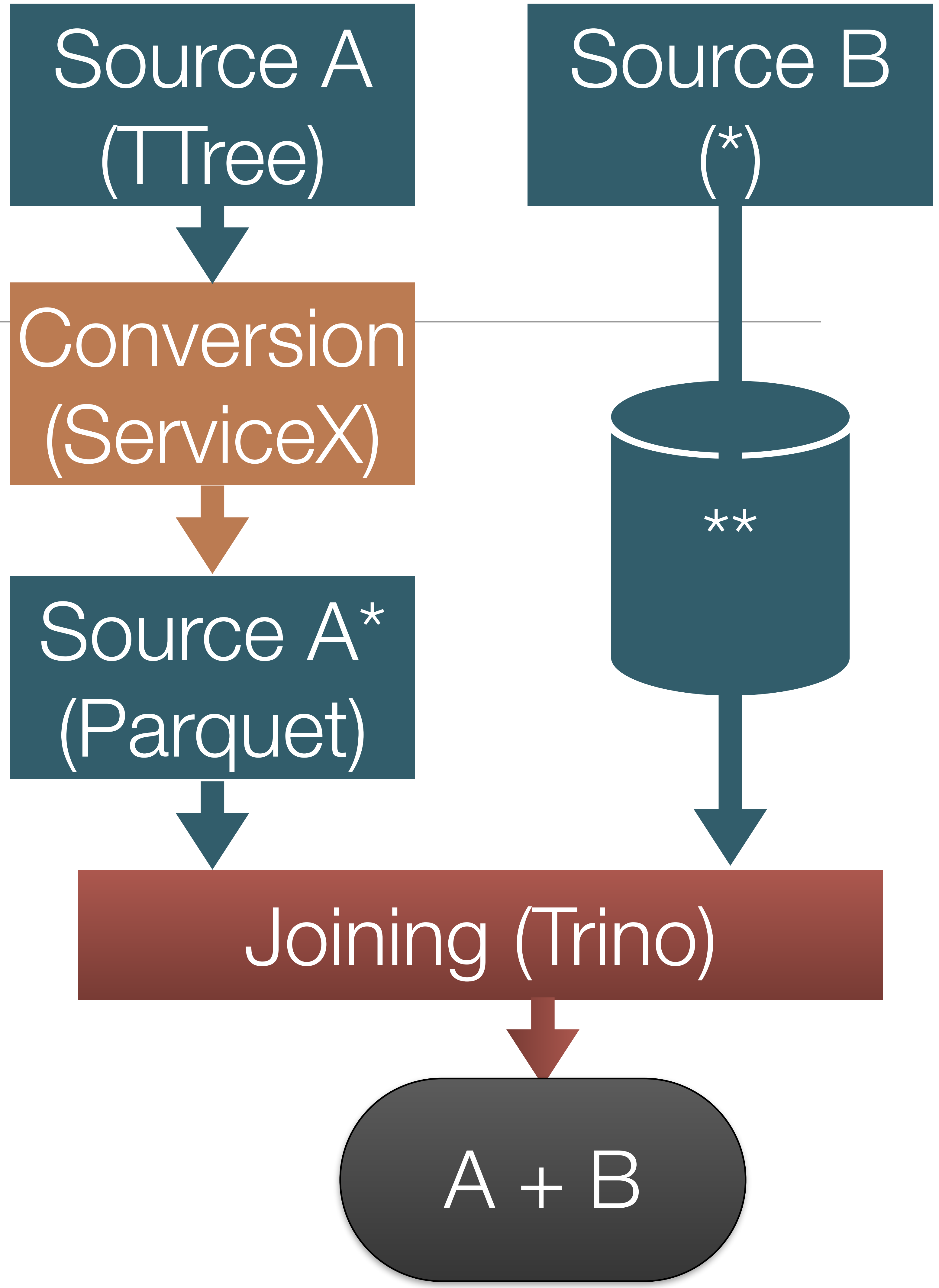
Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem
- * Transient converted dataset ** Transformation as necessary



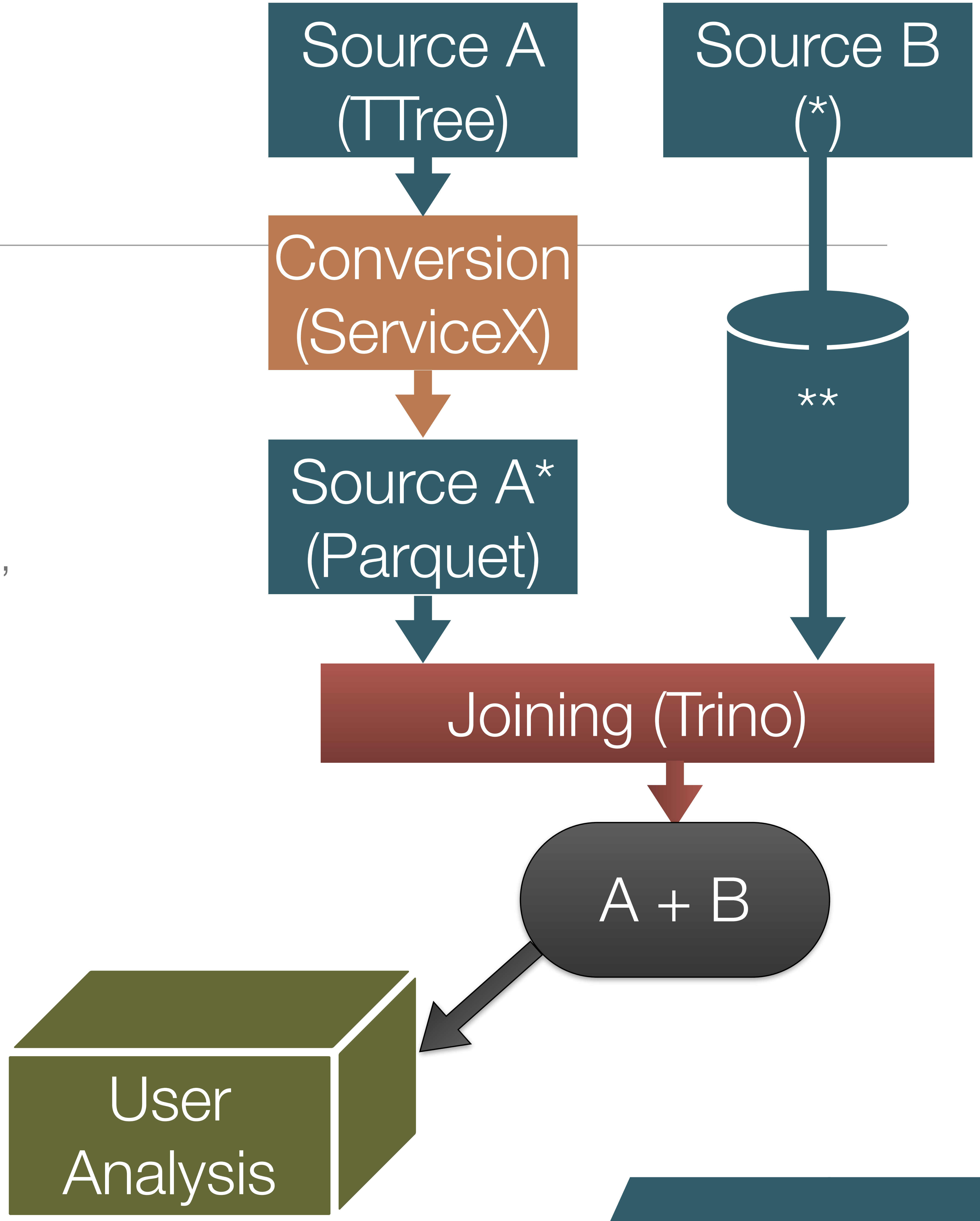
Joins: A Way Forward

- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem
- * Transient converted dataset   ** Transformation as necessary



Joins: A Way Forward

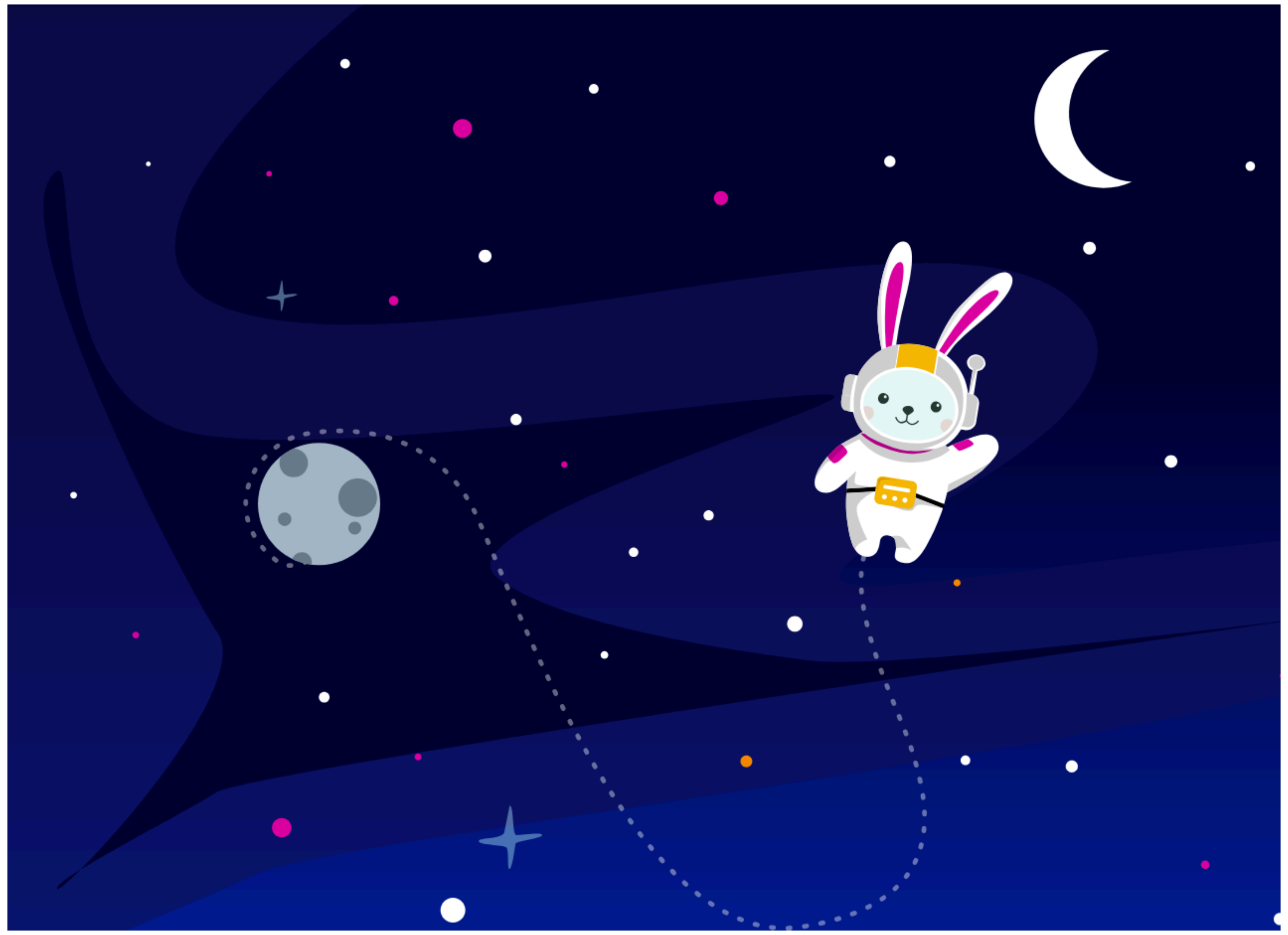
- The capability to join NanoAOD data on-demand with auxiliary information can obviate the need for much data duplication
  - Needs to be: **fast, scaleable, reliable**
  - Must support **ragged data** (vectors/arrays of primitives per-event, not just scalars)
  - Needs to support (in CMS context) NanoAOD + NanoAOD-like inputs (ML inference results, subsets of information derived from parent datatier like MiniAOD with minimal duplication)
  - Ideally should support joining with data from larger data-tiers in a near-seamless fashion (not requiring manual derivation)
  - Needs to be part of the scikit-hep part of the ecosystem
- * Transient converted dataset   ** Transformation as necessary



Trino (as described by o1-mini)

Trino is a high-performance, distributed SQL query engine designed for running interactive analytic queries against various data sources of all sizes. Originally developed by Facebook under the name **Presto**, Trino was forked in 2020 by the original creators to foster a more open and community-driven development model. Trino has since evolved into a robust, open-source project maintained by the **Trino Software Foundation**.

- **Key Features of Trino:**
  - Distributed Architecture
  - SQL Compatibility
  - Federated Querying
  - Performance Optimization
  - Extensibility and Customization:
    - Plugin Architecture: Users can develop **custom connectors** and functions to extend Trino's capabilities.
    - Community-Driven: Being open-source, it benefits from contributions and innovations from a broad community of developers and organizations.
  - Security and Access Control:
    - Authentication and Authorization: Supports various security protocols and integrates with enterprise security systems.
    - Data Encryption: Ensures data privacy through encryption in transit and at rest.



Benchmarking Trino

Source File 1

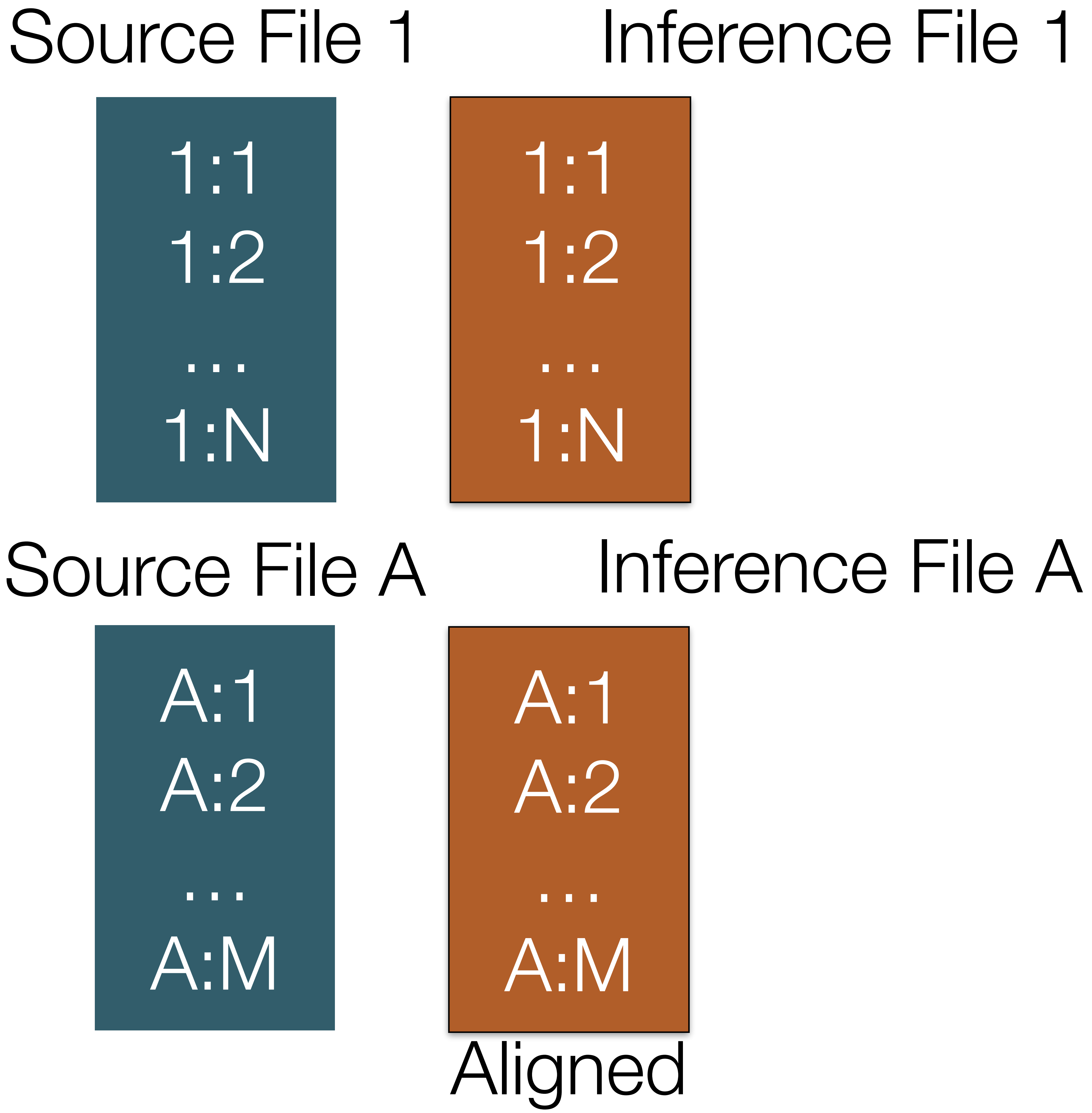
1:1
1:2
…
1:N

Source File A

A:1
A:2
…
A:M

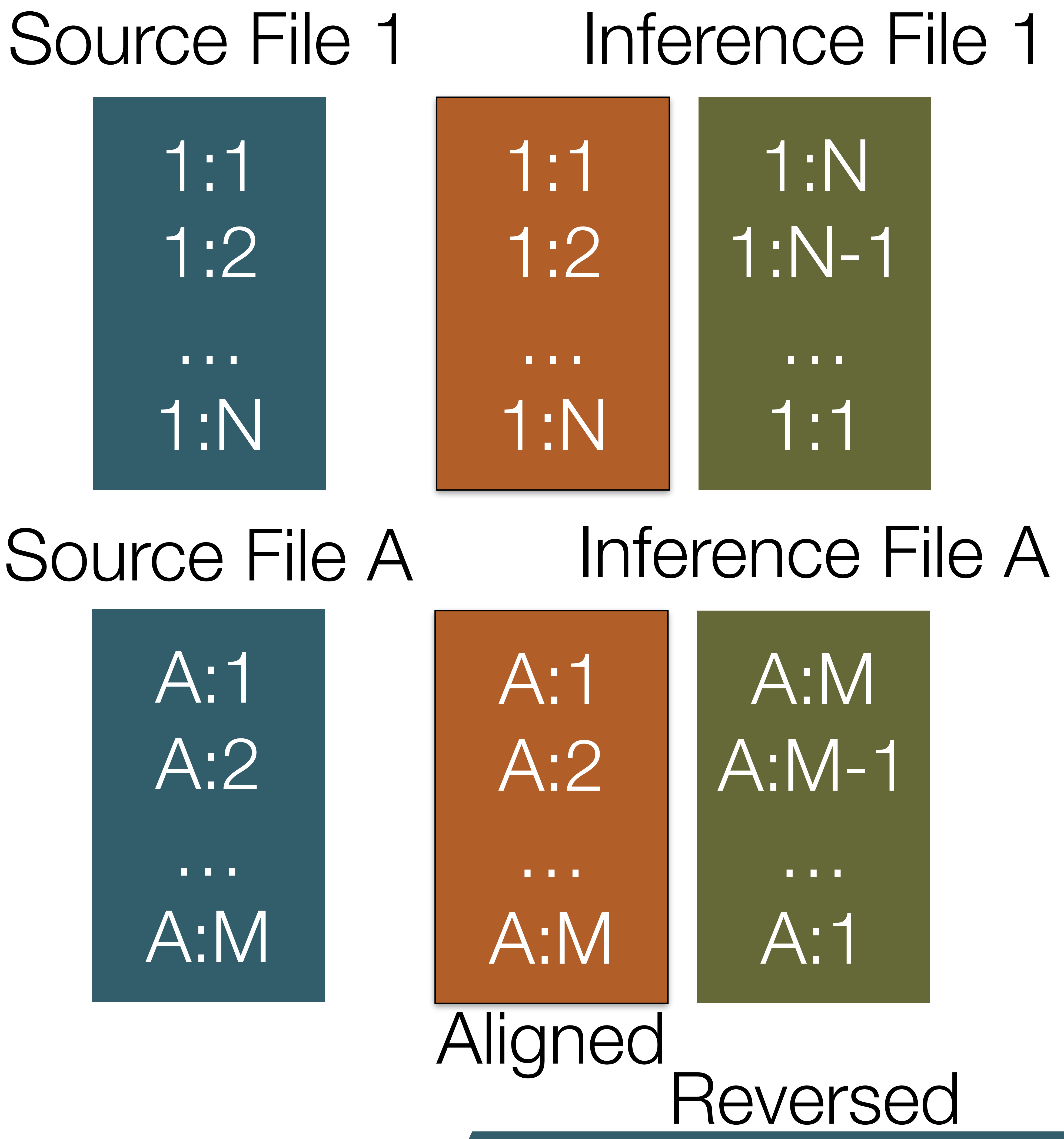
- We've prepared several benchmark datasets using CMS OpenData
  - Source datasets ranging from ~2GB to ~500GB (converted to parquet, ZSTD:5, full NanoAOD)
  - GNN Inference (parquet, 8 scalar-float columns / event)
    - fully-aligned, intra-file-reversed, intra-file-shuffled, globally-shuffled variants
  - Testing various combinations (from a few scalar columns from source + inference, to dozens of ragged fields in source + all 8 inference columns)

Benchmarking Trino



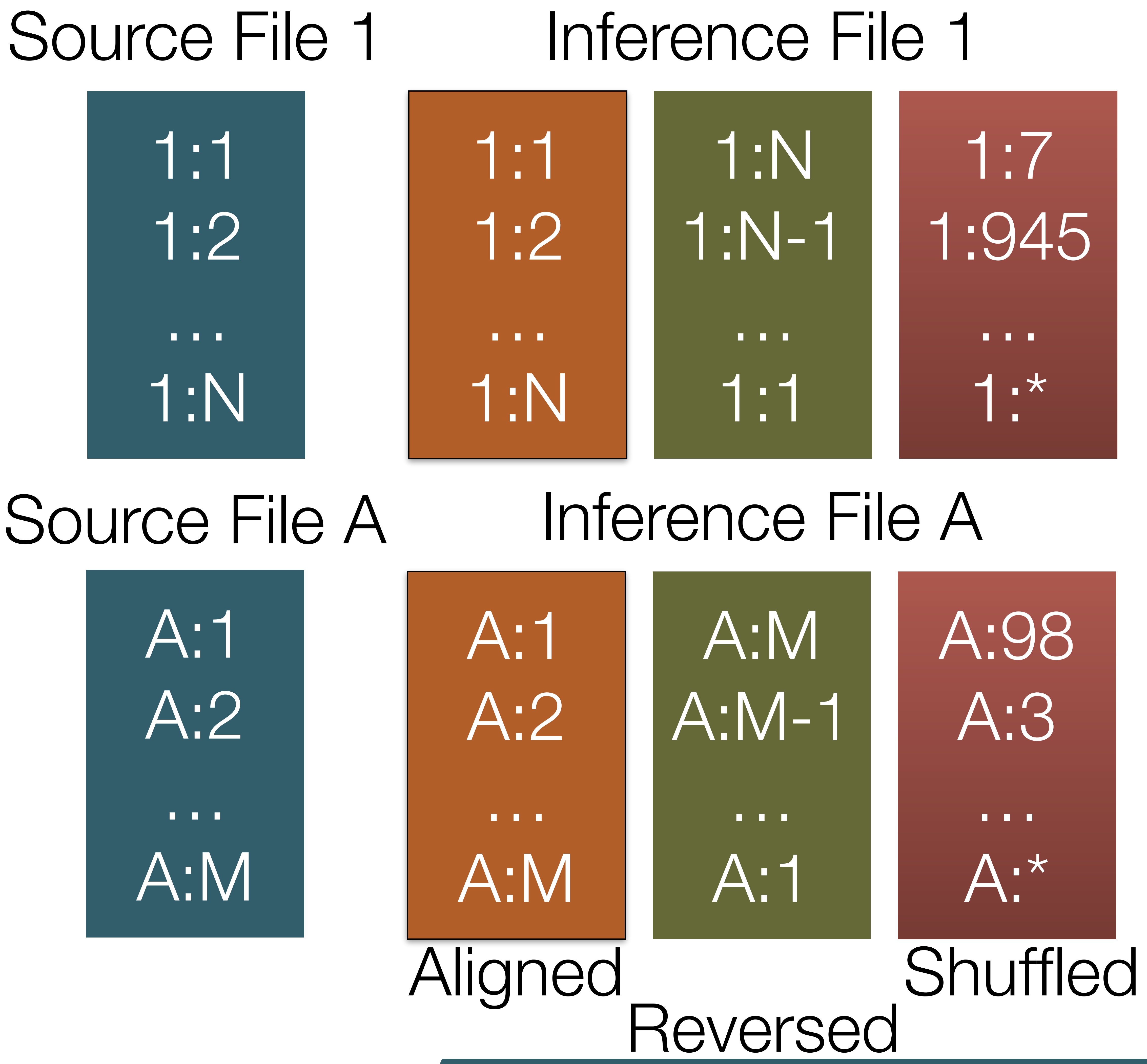
- We've prepared several benchmark datasets using CMS OpenData
  - Source datasets ranging from ~2GB to ~500GB (converted to parquet, ZSTD:5, full NanoAOD)
  - GNN Inference (parquet, 8 scalar-float columns / event)
    - fully-aligned, intra-file-reversed, intra-file-shuffled, globally-shuffled variants
  - Testing various combinations (from a few scalar columns from source + inference, to dozens of ragged fields in source + all 8 inference columns)

Benchmarking Trino



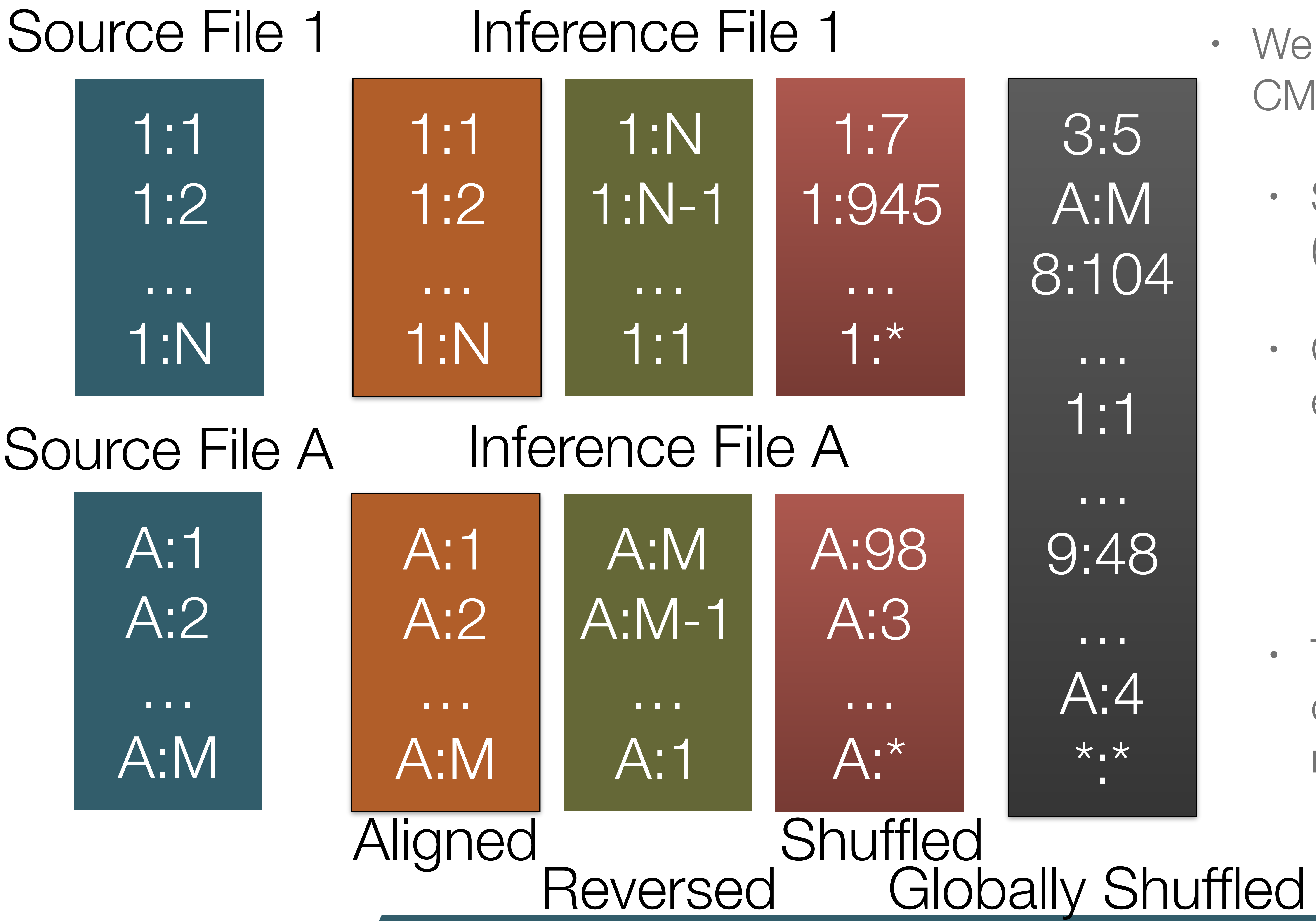
- We've prepared several benchmark datasets using CMS OpenData
  - Source datasets ranging from ~2GB to ~500GB (converted to parquet, ZSTD:5, full NanoAOD)
  - GNN Inference (parquet, 8 scalar-float columns / event)
    - fully-aligned, intra-file-reversed, intra-file-shuffled, globally-shuffled variants
  - Testing various combinations (from a few scalar columns from source + inference, to dozens of ragged fields in source + all 8 inference columns)

Benchmarking Trino



- We've prepared several benchmark datasets using CMS OpenData
  - Source datasets ranging from ~2GB to ~500GB (converted to parquet, ZSTD:5, full NanoAOD)
  - GNN Inference (parquet, 8 scalar-float columns / event)
    - fully-aligned, intra-file-reversed, intra-file-shuffled, globally-shuffled variants
  - Testing various combinations (from a few scalar columns from source + inference, to dozens of ragged fields in source + all 8 inference columns)

Benchmarking Trino



- We've prepared several benchmark datasets using CMS OpenData
  - Source datasets ranging from ~2GB to ~500GB (converted to parquet, ZSTD:5, full NanoAOD)
  - GNN Inference (parquet, 8 scalar-float columns / event)
    - fully-aligned, intra-file-reversed, intra-file-shuffled, globally-shuffled variants
  - Testing various combinations (from a few scalar columns from source + inference, to dozens of ragged fields in source + all 8 inference columns)

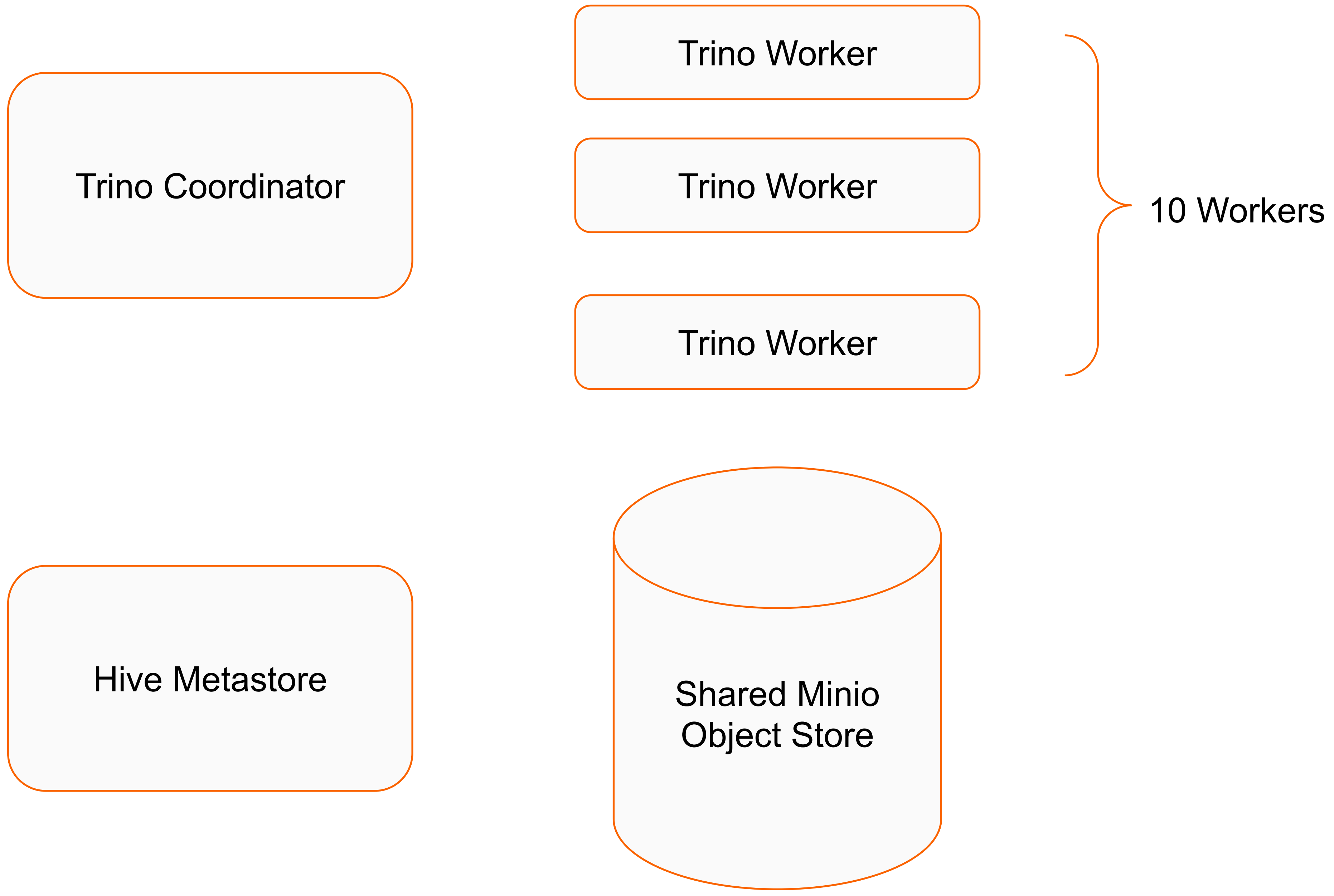
Cluster Configuration

See more in Ben's slides:
https://indico.cern.ch/event/1369601/contributions/5883602/

- Heterogeneous Cluster
    - 3 Nodes: Intel Xeon Silver 4212 @ 2.20GHz (22 cores) - 88GB RAM - Ceph on NVME storage
    - 1 Node: Intel Xeon Silver 4210 @ 2.20GHz (39 cores) - 100GB RAM - Ceph on NVME storage
- Shared resource
    - 10 Workers for Trino on Cluster

Exploratory Environment at FNAL

- Deployed in FNAL OpenShift cluster
- Shared Mino Object Store



Small Dataset Join Benchmark

```
CREATE TABLE single_top_s_chan_join
WITH (
  format = 'PARQUET',
  external_location = 's3a://servicex/nanotest/parquet/AGC/single_top_s_chan/join_out/'
)
AS
  SELECT single_top_s_chan.run, single_top_s_chan.event, single_top_s_chan.luminosityBlock,
    Electron_pt,
    Electron_eta,
    Electron_phi,
    Electron_cutBased,
    Electron_ip3d,
    Electron_sip3d,
    Electron_mass,
    Electron_pfRelIso03_all,
    Electron_pfRelIso03_chg,
    Muon_pt,
    Muon_eta,
    Muon_phi,
    Muon_mass,
    Muon_tightId,
    Muon_ip3d,
    Muon_sip3d,
    Muon_pfRelIso04_all,
    Jet_mass,
    Jet_pt,
    Jet_eta,
    Jet_phi,
    Jet_jetId,
    Jet_btagCSVV2,
    Jet_btagDeepFlavB,
    Jet_btagDeepFlavCvB,
    Jet_btagDeepFlavCvL,
    Jet_btagDeepFlavQG,
    Jet_chEmEF,
    Jet_chHEF,
    Jet_muEF,
    Jet_neEmEF,
    Jet_neHEF
    Jet_puIdDisc,
    Jet_qgl,
    Jet_rawFactor,
    Jet_bRegCorr,
    Jet_bRegRes,
    Jet_electronIdx1,
    Jet_electronIdx2,
    Jet_muonIdx1,
    Jet_muonIdx2,
    GNN_p1, GNN_p2, GNN_p3, GNN_p4
  FROM single_top_s_chan
  JOIN single_top_s_chan_infer ON
    single_top_s_chan.run = single_top_s_chan_infer.run AND
    single_top_s_chan.luminosityBlock = single_top_s_chan_infer.luminosityBlock AND
    single_top_s_chan.event = single_top_s_chan_infer.event;
```

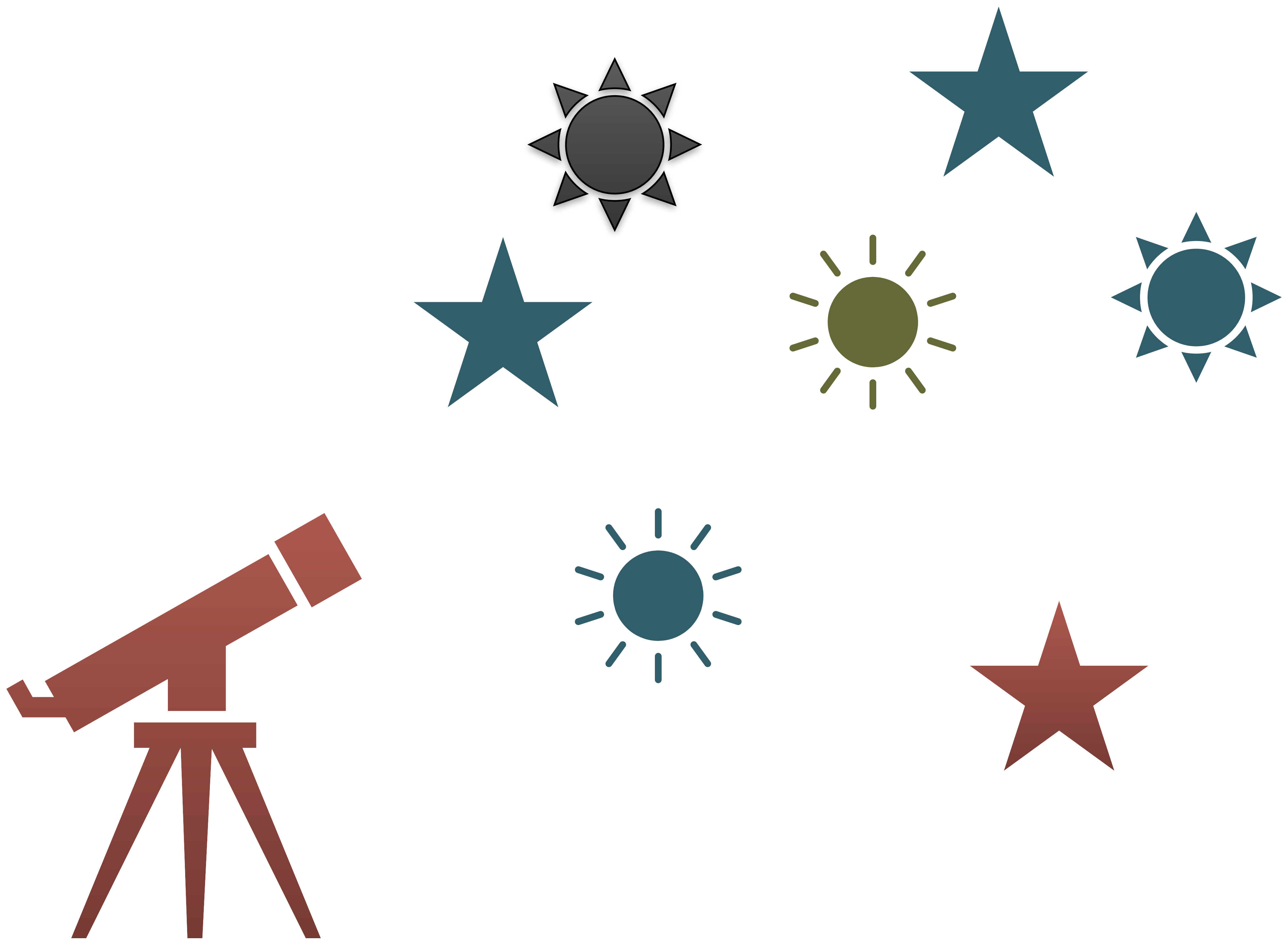
- Dataset with 2.8M events (rows), ragged primitive data for most columns (3.7GB source + 93MB inference)
- Encouraging result: seemingly **invariant* to permutations** being joined

2.2M events/s

```
single_top_s_chan_infer
    Query 20240927_183428_00028_au69m, FINISHED, 11 nodes
    Splits: 862 total, 862 done (100.00%)
    16.64 [34.4M rows, 1.48GB] [2.07M rows/s, 91.3MB/s]

    trino:servicex> select count(*) from single_top_s_chan_join;
      _col0
    ---------
     2867199
    (1 row)

single_top_s_chan_infer_reversed
    Query 20240927_184139_00033_au69m, FINISHED, 11 nodes
    Splits: 862 total, 862 done (100.00%)
    14.54 [34.4M rows, 1.48GB] [2.37M rows/s, 104MB/s]

    trino:servicex> select count(*) from single_top_s_chan_join;
      _col0
    ---------
     2867199
    (1 row)

single_top_s_chan_infer_intrafileshuffle
    Query 20240927_184318_00036_au69m, FINISHED, 11 nodes
    Splits: 862 total, 862 done (100.00%)
    15.49 [34.4M rows, 1.5GB] [2.22M rows/s, 98.9MB/s]

    select count(*) from single_top_s_chan_join;
      _col0
    ---------
     2867199
    (1 row)

single_top_s_chan_infer_globalshuffle
    Query 20240927_184518_00039_au69m, FINISHED, 11 nodes
    Splits: 859 total, 859 done (100.00%)
    15.00 [34.4M rows, 1.5GB] [2.29M rows/s, 102MB/s]

    select count(*) from single_top_s_chan_join;
      _col0
    ---------
     2867199
```

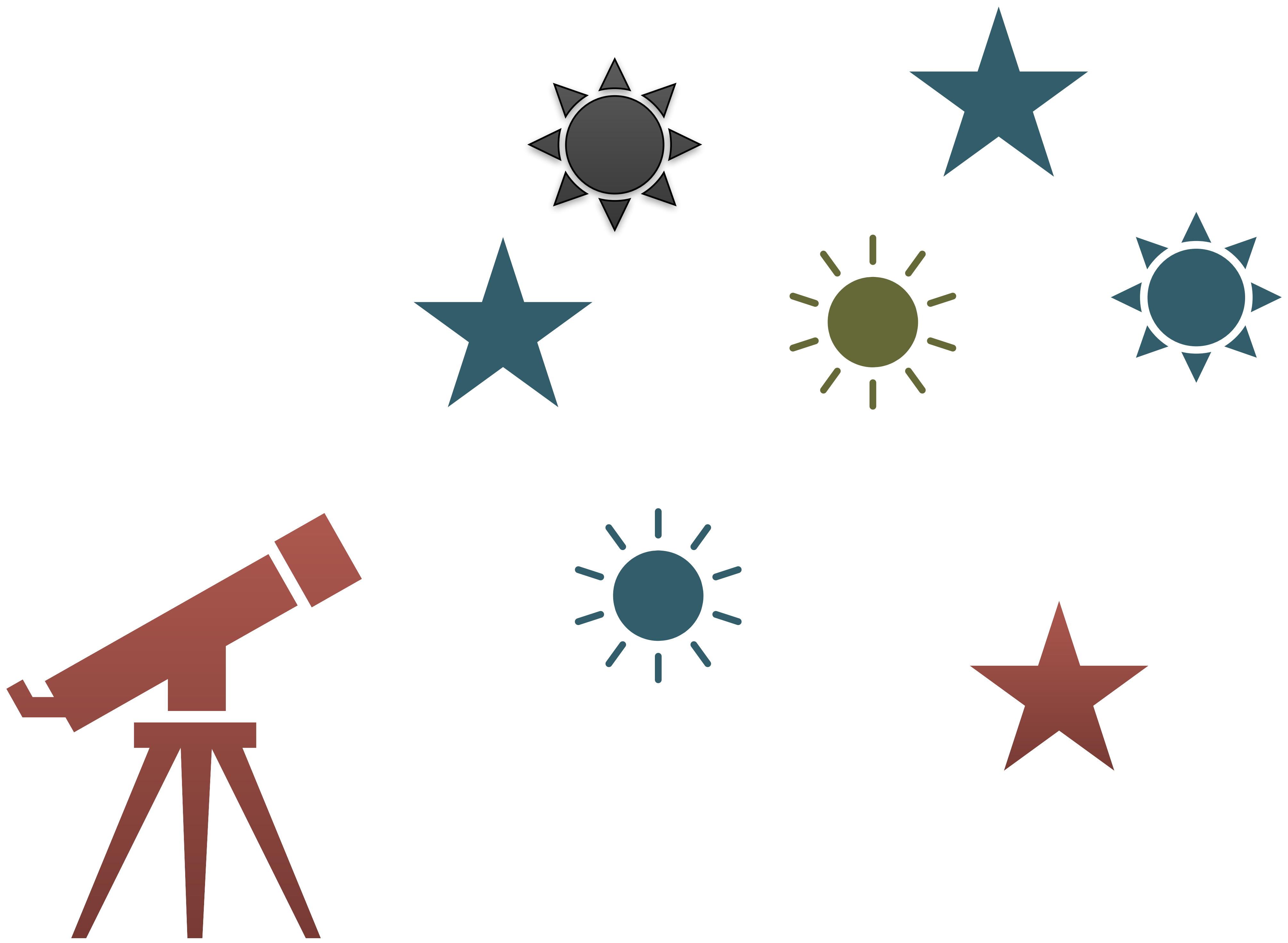
* stat fluctuations, warm-caching to be eliminated as sources of differences in high-stat testing

“All that glitters is not gold”



Large Dataset Join Benchmark

“All that glitters is not gold”

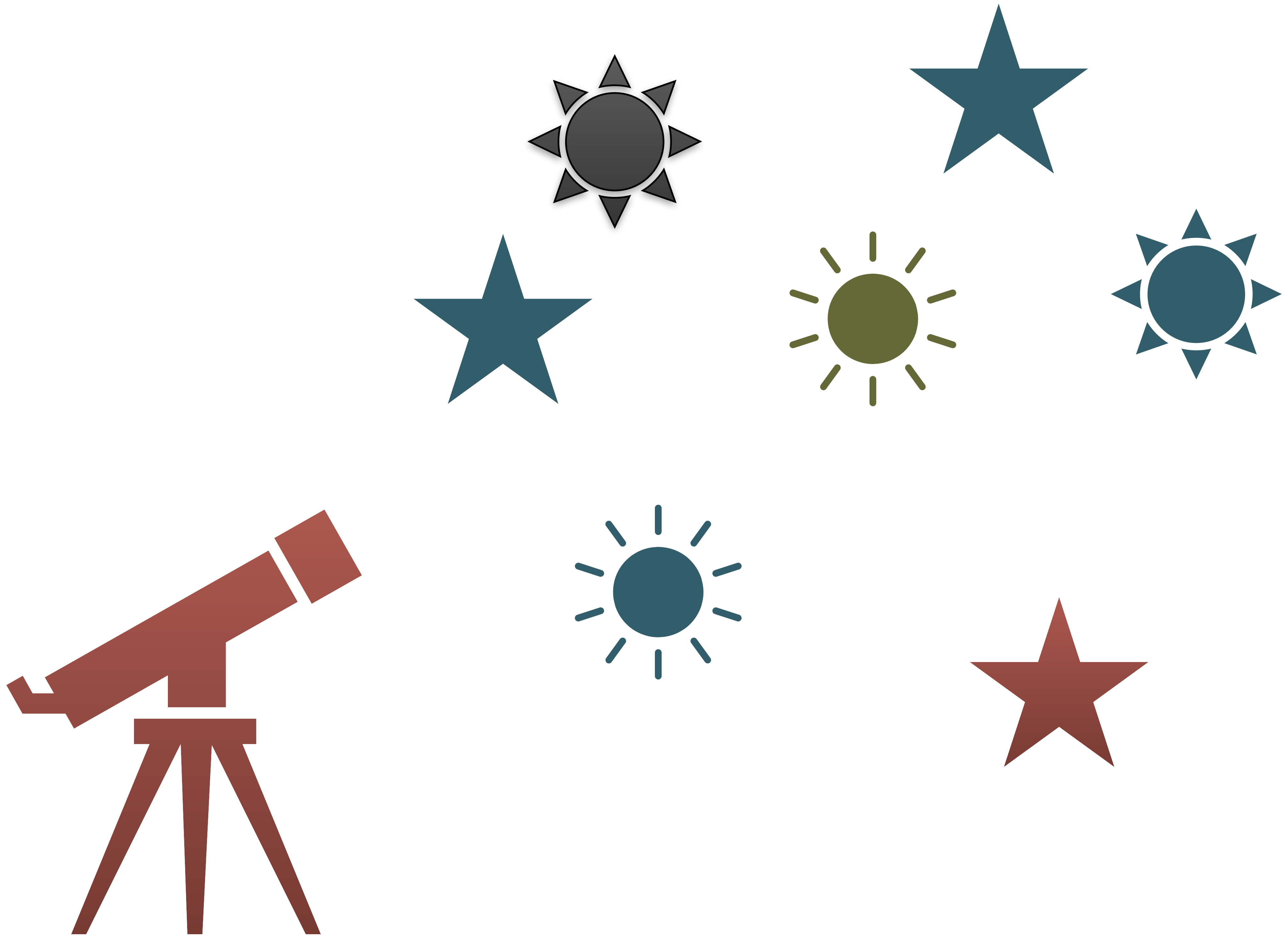


Large Dataset Join Benchmark

- Largest dataset of 500GB (ttbar) crashed trino deployment*

Large Dataset Join Benchmark

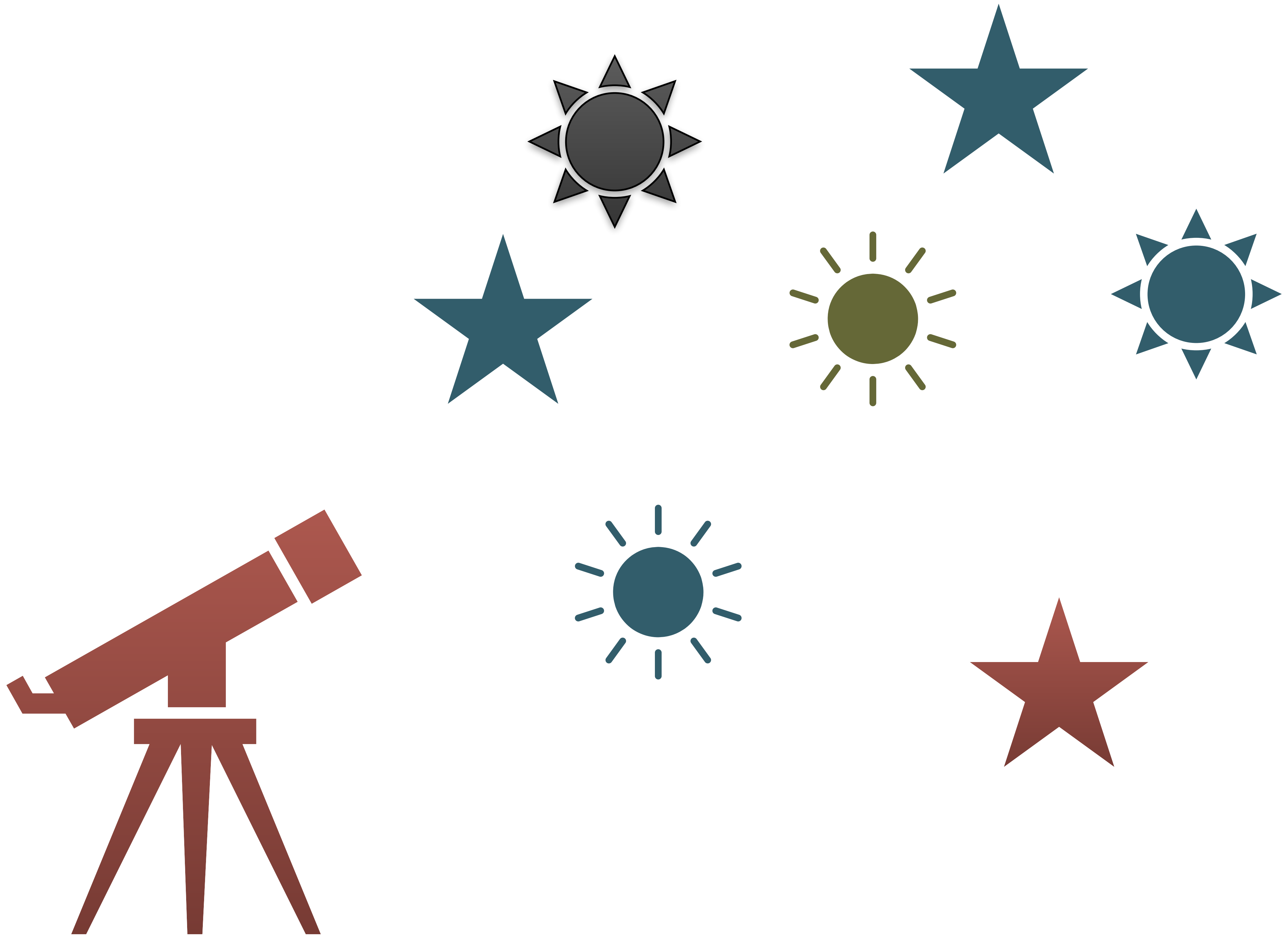
"All that glitters is not gold"



- Largest dataset of 500GB (ttbar) crashed trino deployment*

* Incorrect resource limits set for trino in OKD

“All that glitters is not gold”



Large Dataset Join Benchmark

- Largest dataset of 500GB (ttbar) crashed trino deployment*

- May indicate that partitioning the joins into dask-task sized elements will be a necessity to use trino as our distributed SQL engine (already a desirable element, as described later)

* Incorrect resource limits set for trino in OKD

Benchmark of “analysis” read-speed

Stats computed over 7 runs, 5 loops (timeit)

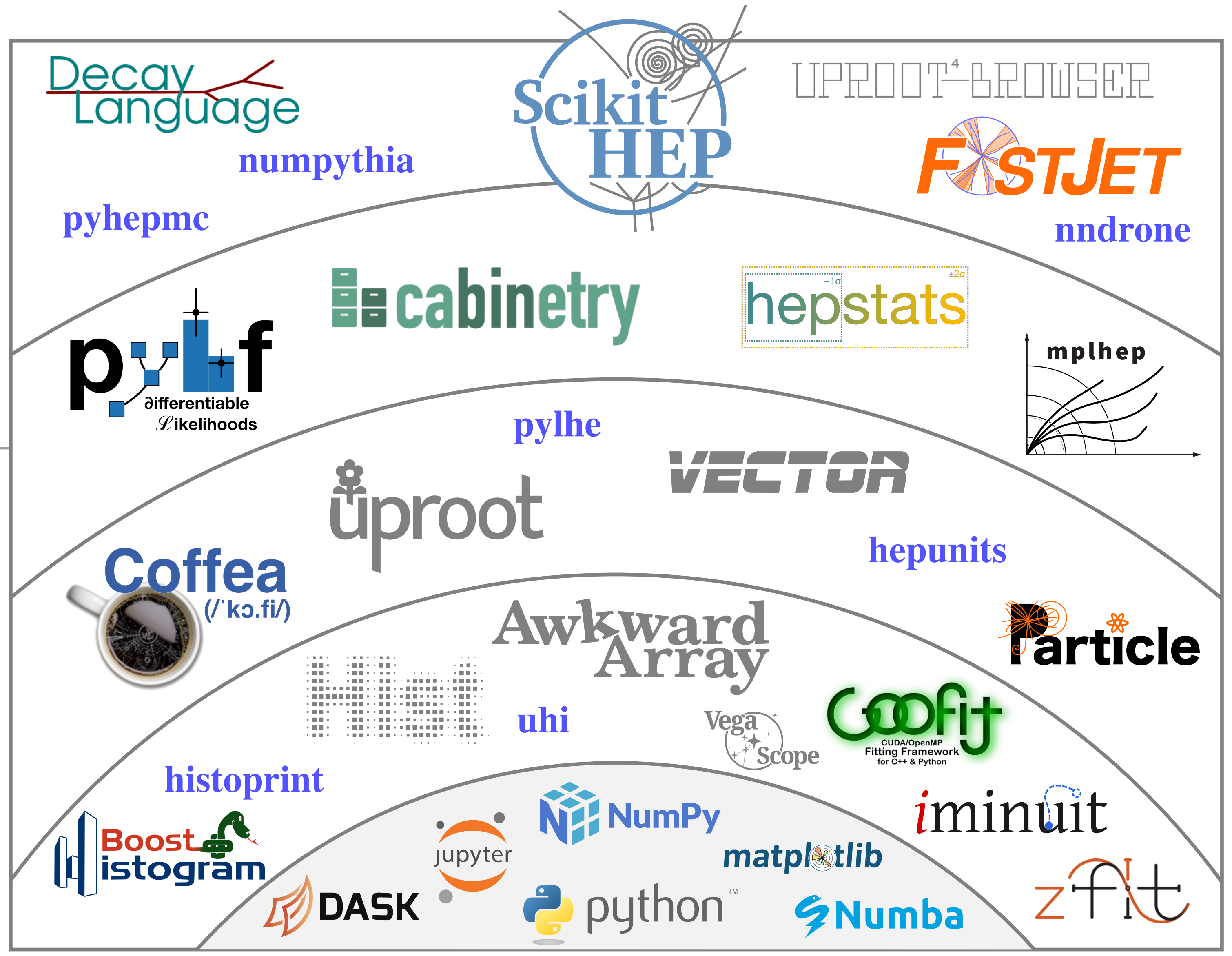
1,334,428 events processed

Benchmark Output Analysis Formats

Test of a simple pseudo-analysis running on various root TTree and parquet files, in various compression schemes

Format	Source	How Made	Compression	Time
ROOT	AGC	N/A	ZLIB:1	11.7 s ± 389 ms
ROOT	converted	root hadd	LZMA:9	11.9 s ± 551 ms
ROOT	converted	root hadd	ZSTD:5	10.9 s ± 477 ms
Parquet	converted	hepconvert	ZSTD:5	4.73 s ± 61 ms
Parquet	joined	trino	GZIP	3.99 s ± 129 ms

Coffea analysis



- coffea brings together individual scikit-hep elements needed for a full analysis, and provides **schema-application, corrections, scaleout**(-patterns)
  - and often the first place something is prototyped and tested before being spun out into it's own package
- Tightly integrated with dask:
  - User's analysis code is broadcast over datasets to create **task graphs**
  - Typetracer setup per dataset records operations (lazy, no data executed on)
  - Task graphs are distributed to compute resources to execute
  - Results returned to user's client (histograms, small arrays, locations of output root/parquet files, …)
- Task graphs are key: allow programmatic optimization of analysis, understanding **necessary inputs** as mapped to **requested outputs**

Joins R&D (End-to-End Processing with dask)

Joins R&D (End-to-End Processing with dask)

Source A
(Format A)

Source B
(Format B)

Source C?
(Format C)

Joins R&D (End-to-End Processing with dask)

Source A
(Format A)

Source B
(Format B)

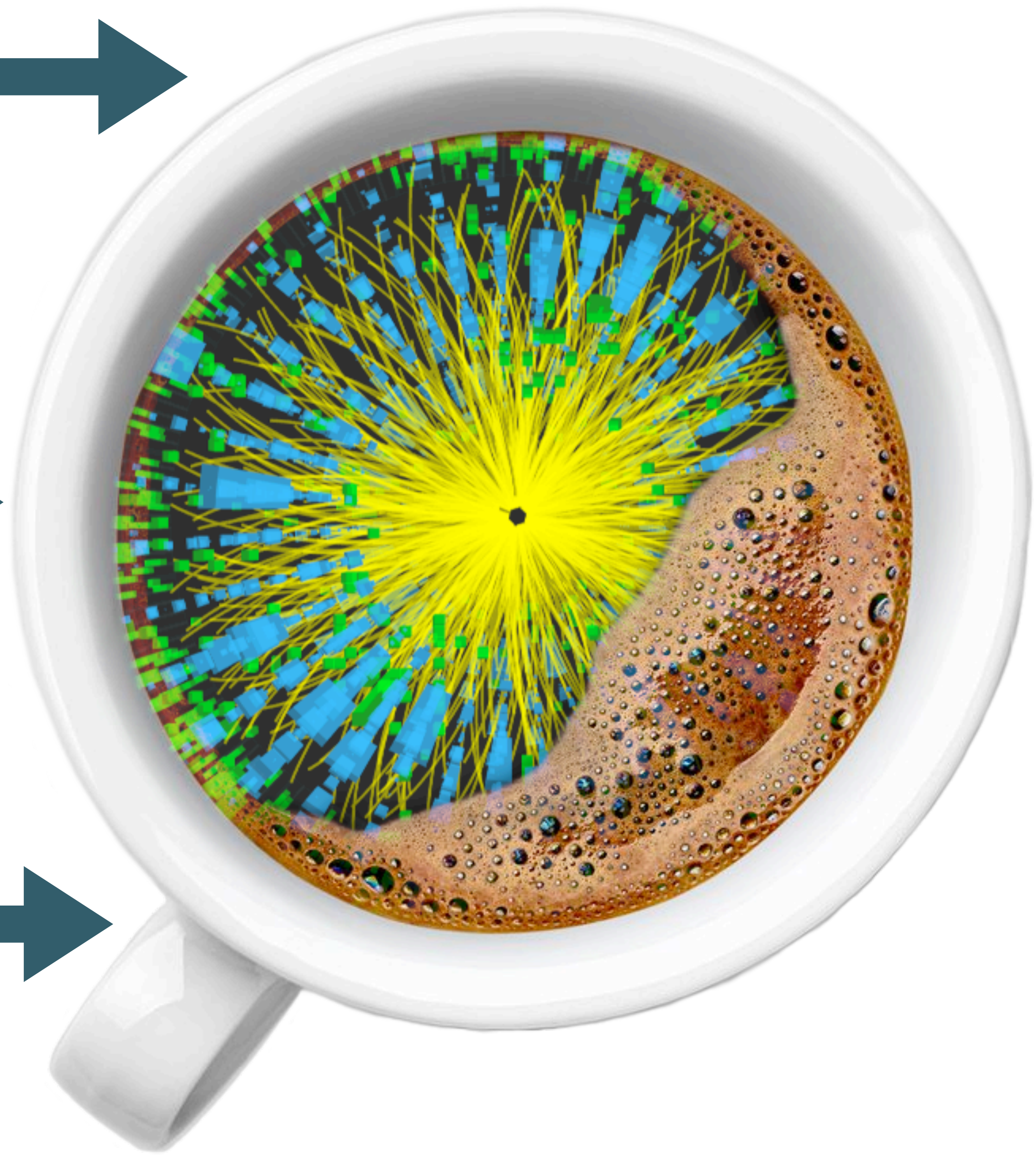
Source C?
(Format C)

Joins R&D (End-to-End Processing with dask)

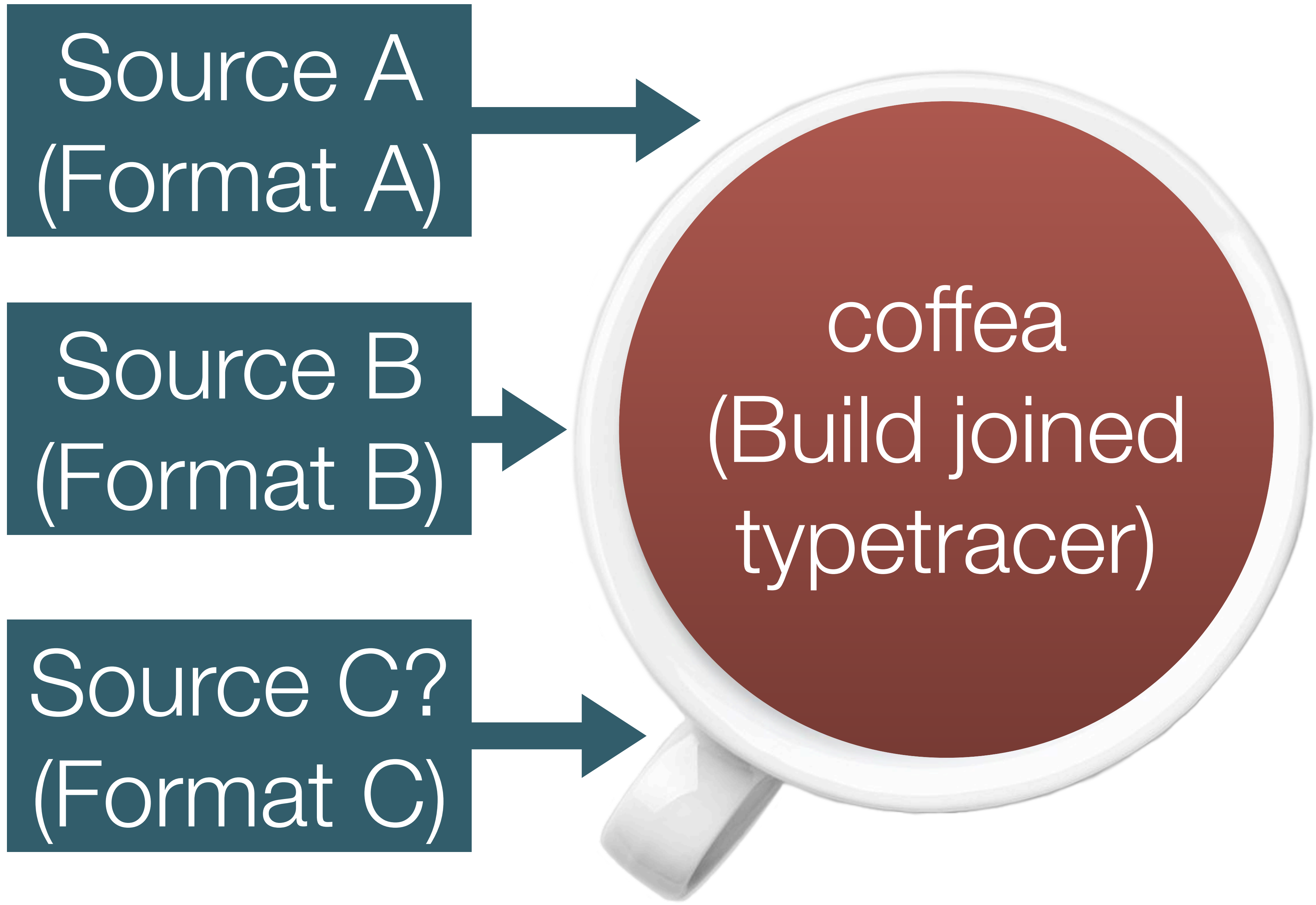
Source A
(Format A)

Source B
(Format B)

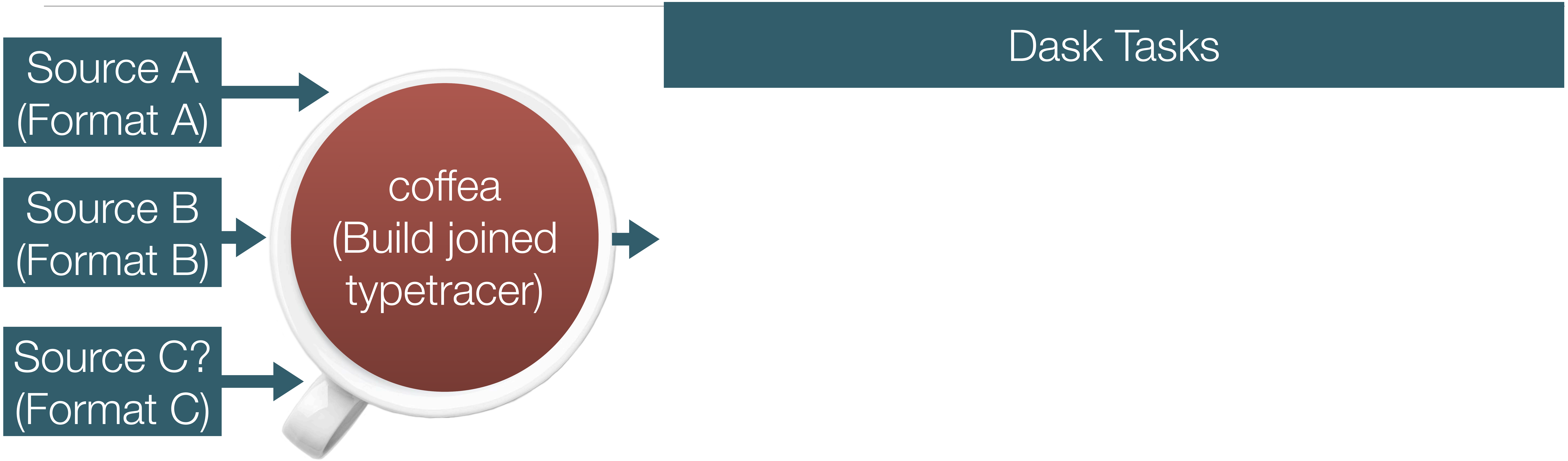
Source C?
(Format C)



Joins R&D (End-to-End Processing with dask)



Joins R&D (End-to-End Processing with dask)



Joins R&D (End-to-End Processing with dask)

Source A
(Format A)

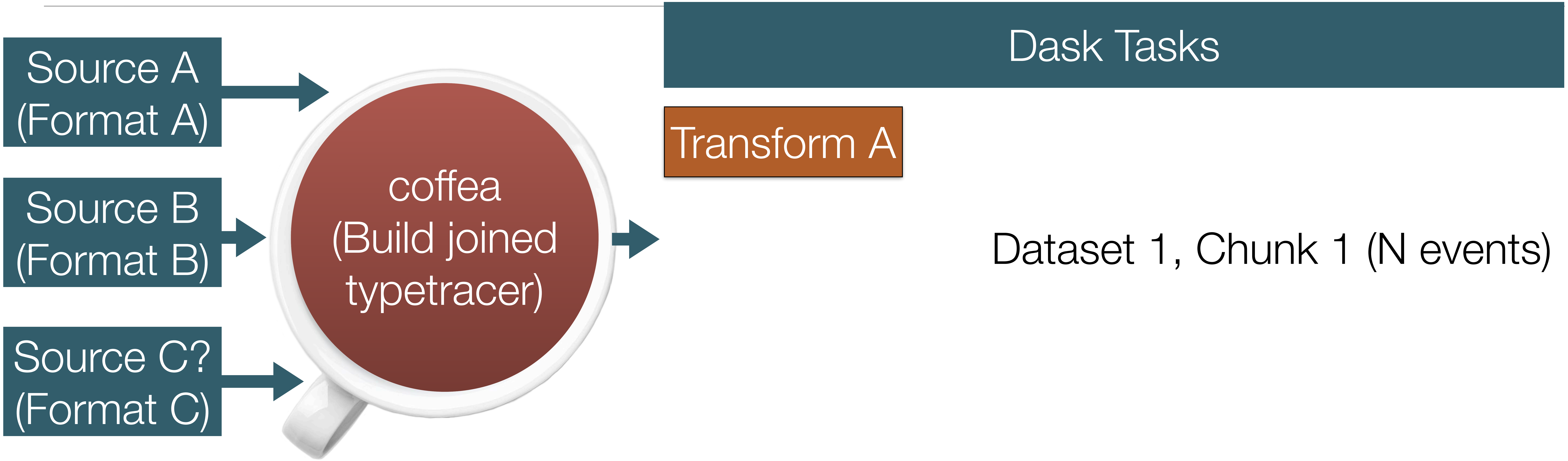
Source B
(Format B)

Source C?
(Format C)

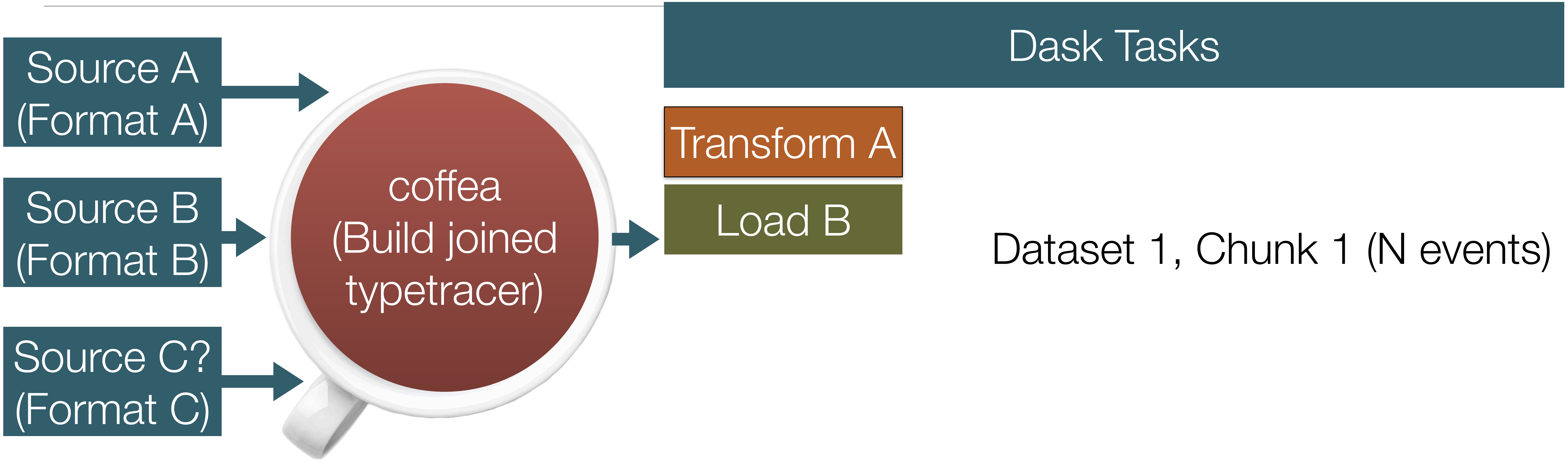
Dask Tasks

Dataset 1, Chunk 1 (N events)

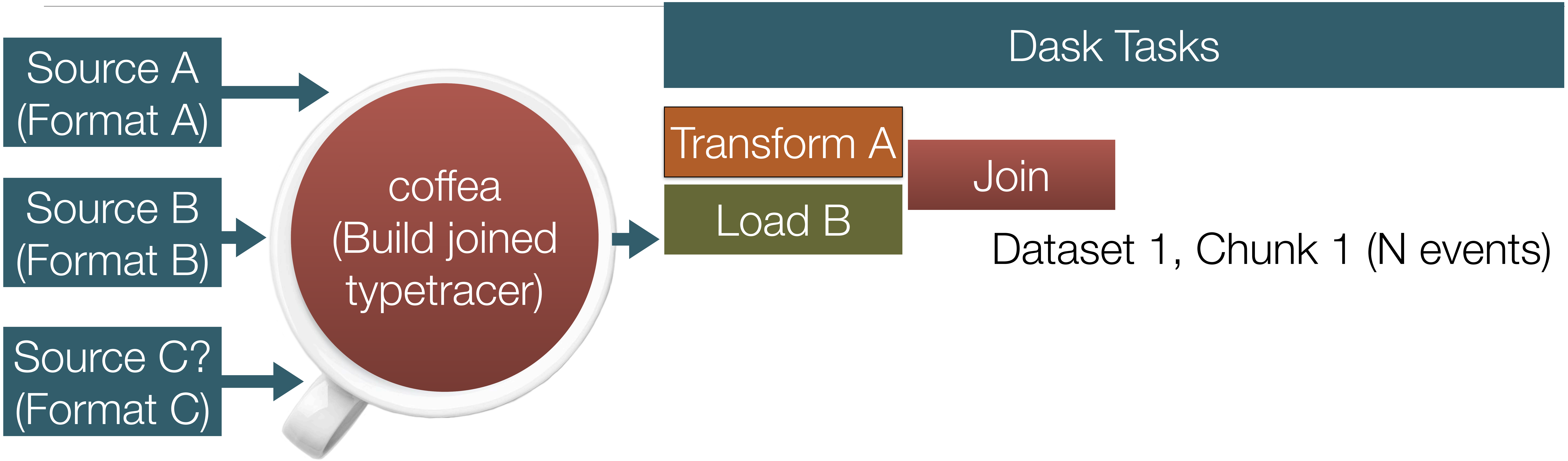
Joins R&D (End-to-End Processing with dask)



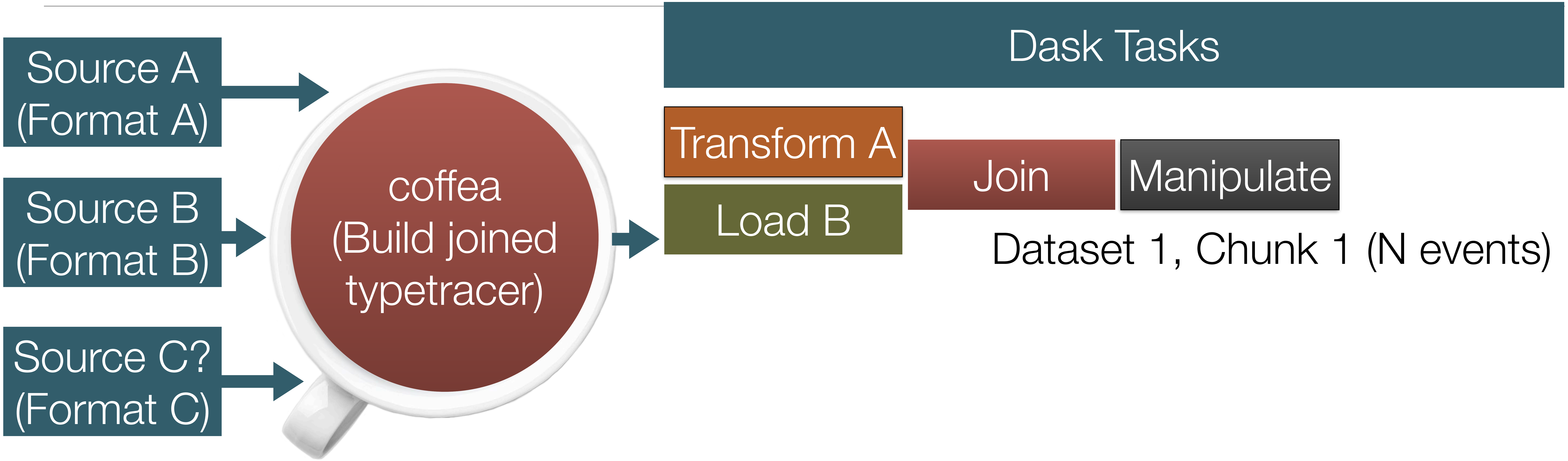
Joins R&D (End-to-End Processing with dask)



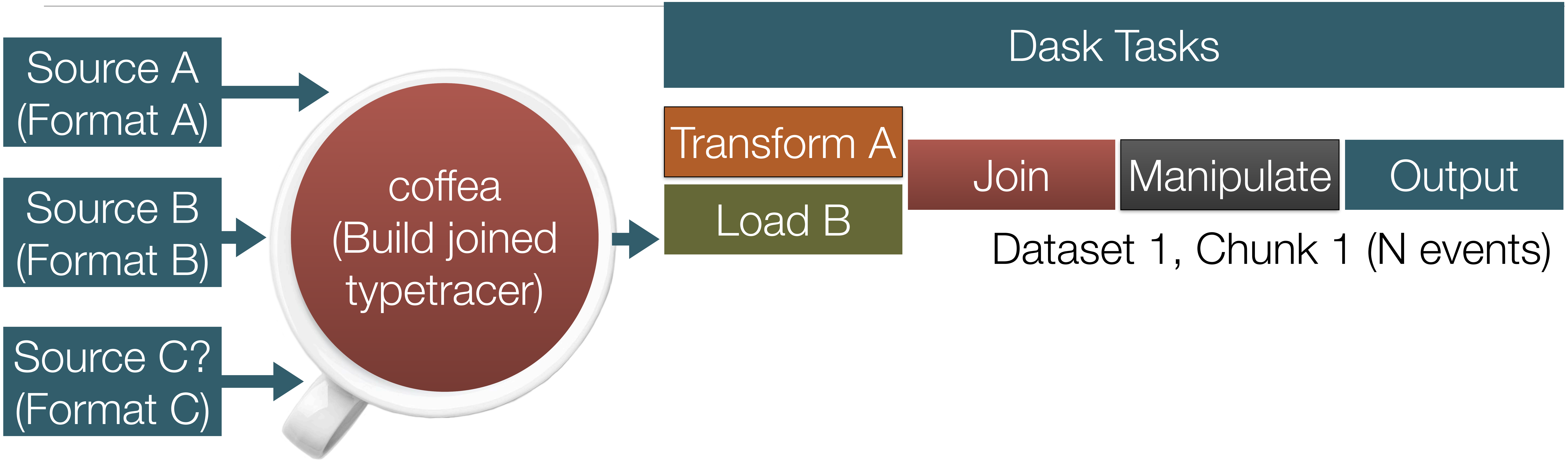
Joins R&D (End-to-End Processing with dask)



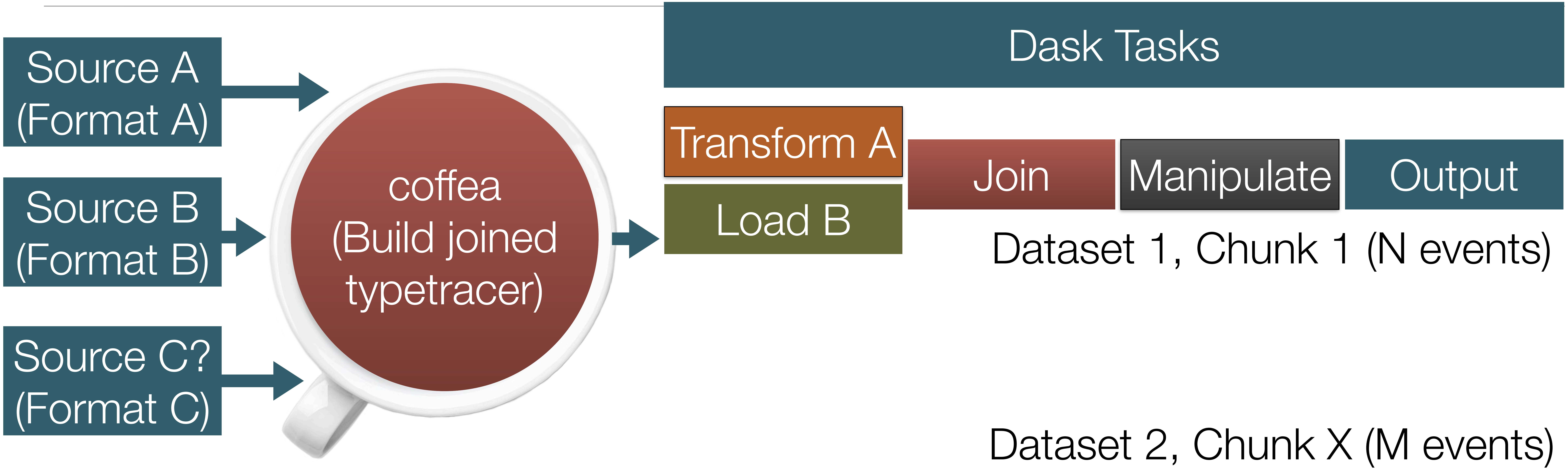
Joins R&D (End-to-End Processing with dask)



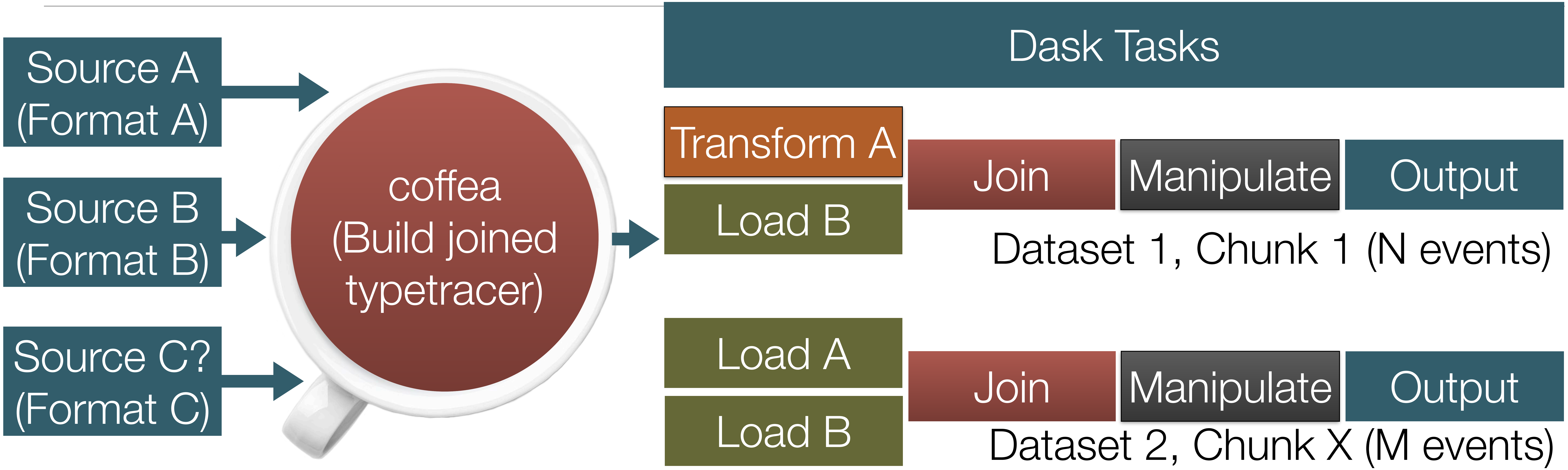
Joins R&D (End-to-End Processing with dask)



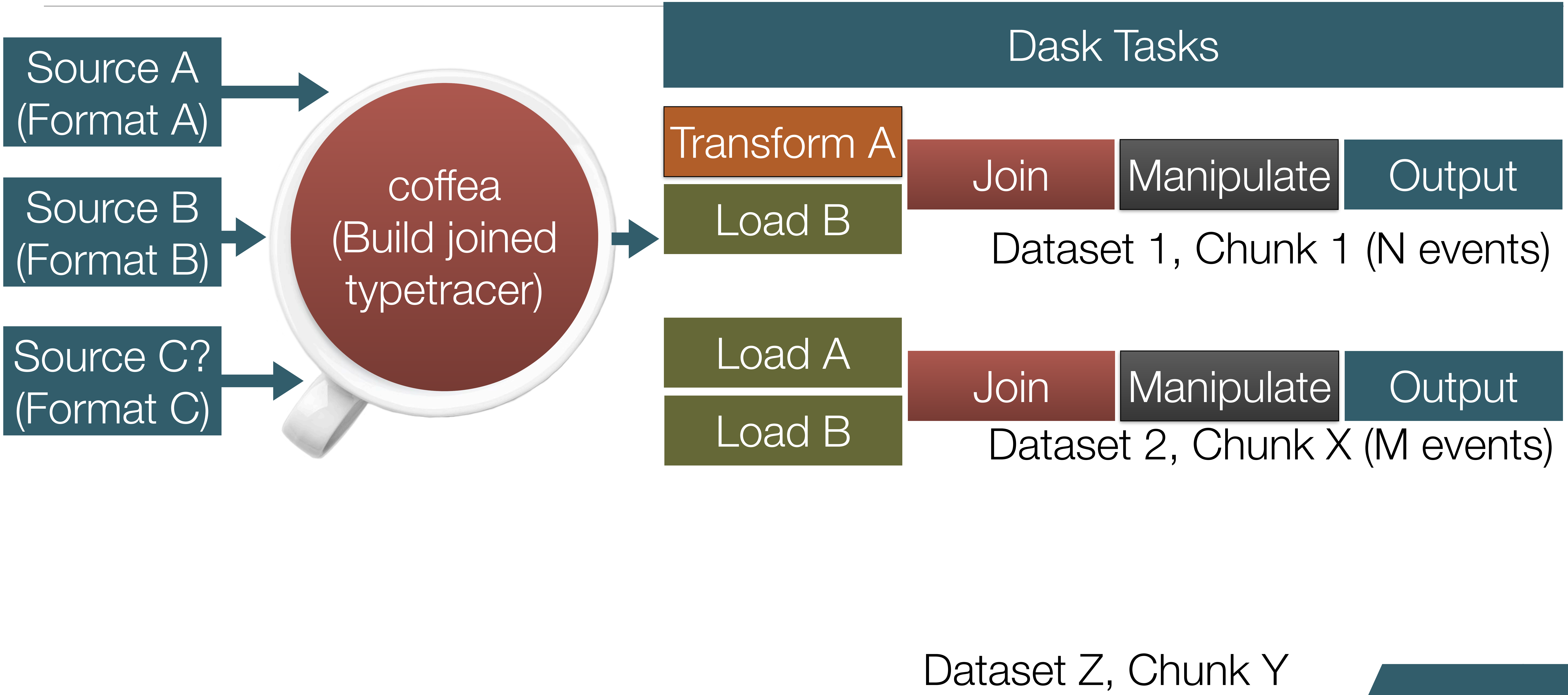
Joins R&D (End-to-End Processing with dask)



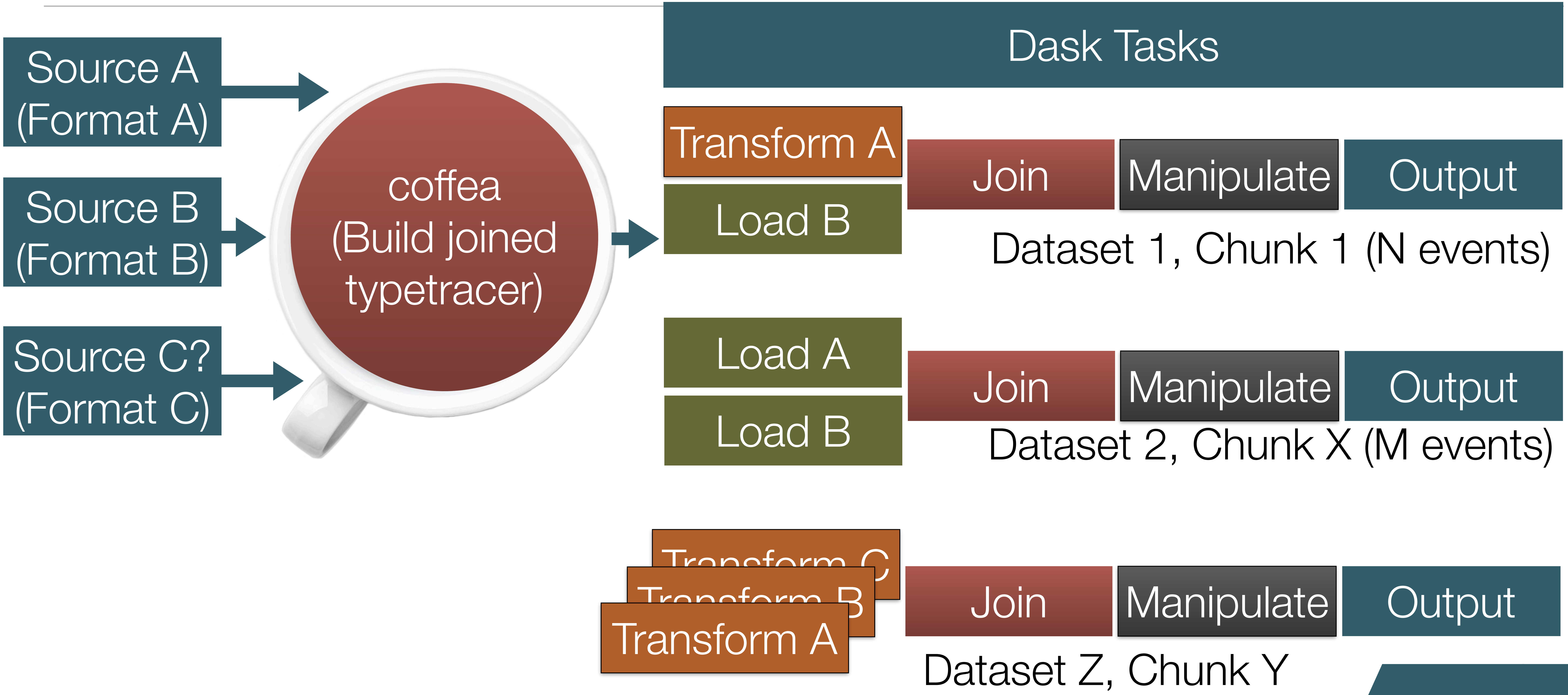
Joins R&D (End-to-End Processing with dask)



Joins R&D (End-to-End Processing with dask)



Joins R&D (End-to-End Processing with dask)



Ongoing and Future Developments

Ongoing and Future Developments

- Baseline and Optional Targets of US CMS HL-LHC R&D Program (2024) in collaboration with IRIS-HEP
  - Building Typetracer for pseudo-joined data
  - Generation of ServiceX conversion tasks
  - Building Trino join queries, embedding of joined-output into daskified analysis
  - Updated and expand ServiceX MiniAOD conversion/selection of auxiliary information to be joined with NanoAOD (Stretch Goal)

Ongoing and Future Developments

- Baseline and Optional Targets of US CMS HL-LHC R&D Program (2024) in collaboration with IRIS-HEP
    - Building Typetracer for pseudo-joined data
    - Generation of ServiceX conversion tasks
    - Building Trino join queries, embedding of joined-output into daskified analysis
    - Updated and expand ServiceX MiniAOD conversion/selection of auxiliary information to be joined with NanoAOD (Stretch Goal)
- Funded by LPC Distinguished Researcher Program (2025)
    - Native Ceph Object Store usage (Currently object -> file -> object)

Ongoing and Future Developments

- Baseline and Optional Targets of US CMS HL-LHC R&D Program (2024) in collaboration with IRIS-HEP
    - Building Typetracer for pseudo-joined data
    - Generation of ServiceX conversion tasks
    - Building Trino join queries, embedding of joined-output into daskified analysis
    - Updated and expand ServiceX MiniAOD conversion/selection of auxiliary information to be joined with NanoAOD (Stretch Goal)
- Funded by LPC Distinguished Researcher Program (2025)
    - Native Ceph Object Store usage (Currently object -> file -> object)
- Future developments within US CMS and IRIS-HEP
    - Kafka integration for streaming output
    - RNTuple support in trino

Specific Example Use-cases

Specific Example Use-cases

- Augmenting with objects from MiniAOD datatier
  - Adding ParticleFlow candidates for reclustering
  - Full set of ML taggers

Specific Example Use-cases

- Augmenting with objects from MiniAOD datatier
  - Adding ParticleFlow candidates for reclustering
  - Full set of ML taggers
- Caching Experiment-wide objects (systematic variations, CMS ParticleTransformer, FlashSim?)
  - Versioning potential

Specific Example Use-cases

- Augmenting with objects from MiniAOD datatier
  - Adding ParticleFlow candidates for reclustering
  - Full set of ML taggers
- Caching Experiment-wide objects (systematic variations, CMS ParticleTransformer, FlashSim?)
  - Versioning potential
- Caching Analysis-specific non-ML (derived quantities, systematic variations) and ML results (event, object classifiers)

Funding Acknowledgements

This work was performed with support of the U.S. CMS Software and Computing Operations Program under the U.S. CMS HL-LHC R&D Initiative. This work was partially supported by Fermilab operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the Department of Energy, and by the National Science Foundation under grant ACI-1450377 and Cooperative Agreement PHY-1120138. Additional support came from the Department of Energy DE-SC0010005 grant.

This project is supported by the National Science Foundation under Cooperative Agreements OAC-1836650 and PHY-2323298. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

BACKUP

US CMS Software and Computing - HL-LHC R&D (WBS4)

2 Grand Challenges for HL-LHC Computing and Software

In order to effectively focus and structure the U.S. CMS R&D efforts, we organize the innovation, research, and development needs for HL-LHC computing into the four *Grand Challenges* that encompass the advances needed for HL-LHC computing to succeed:

(1) **Modernizing Physics Software and Improving Algorithms**

Exploit novel algorithms, including ML/AI, reduce algorithmic complexity, increase computational intensity, and provide core software infrastructure to enable effective use of modern hardware and accelerators. The work is organized in the following work packages:

- Core Software Framework and Software Portability
- Establish Performance Metric and Performance Baseline for Physics Software
- U.S. Contributions to the Charged Particle Tracking Software
- U.S. Contributions to Software for High Granularity Calorimeter
- U.S. Contributions to CMS Advanced Algorithms Work

(2) **Building Infrastructure for Exabyte-Scale Datasets**

Build infrastructure to archive, store, transfer, and provide access to exabyte-scale datasets. Explore data lakes and custodial storage: establish a technology and cost model for custodial/archival storage facilities which manages operations costs, and optimizes hardware costs. Orchestrate computational services and data access, provide intelligent

FriendTrees

- Traditional ROOT workflows have some functionality that can alleviate the data duplication issues, via the FriendTree mechanism
  - Supports joining entry-by-entry events in two separate (groups of) files
  - Can accommodate some situations with non-aligned (out-of-order) joins with BuildIndex (major + minor keys, i.e. run + event)
    - https://root.cern/doc/master/classTTree.html#a3f6b5bb591ff7a5bd0b06eea6c12b998
    -

O1-mini's description of TTrees, Parquet, RNTuple

2. Technology Descriptions

a. ROOT TTrees

ROOT TTrees are a fundamental data structure within the ROOT framework, widely used in HEP for storing and analyzing large datasets.

- Structure: TTrees store data in a hierarchical, tree-like structure with branches representing different variables (columns).
- Features:
  - Custom Compression: Supports various compression algorithms tailored for HEP data.
  - Provenance Tracking: Maintains metadata and provenance information essential for reproducible research.
  - Integration with ROOT: Seamless integration with ROOT's data analysis tools and C++/Python interfaces.

b. Apache Parquet

Apache Parquet is an open-source, columnar storage format optimized for performance and interoperability across different data processing frameworks.

- Structure: Stores data in column chunks, enabling efficient compression and encoding schemes.
- Features:
  - Language Agnostic: Supports multiple programming languages (e.g., Java, C++, Python).
  - Wide Ecosystem Support: Integrates with big data tools like Apache Spark, Hive, and Pandas.
  - Efficient Compression: Utilizes advanced compression techniques to reduce storage footprint.

c. ROOT RNTuple

ROOT RNTuple is a newer addition to the ROOT framework, designed to address some limitations of TTrees and to provide a modern, efficient, and scalable storage solution.

- Structure: Similar to TTrees but optimized for better performance and scalability.
- Features:
  - Asynchronous I/O: Enhances read/write performance by supporting non-blocking operations.
  - Better Parallelism: Designed to leverage multi-threading and distributed computing environments.
  - Improved Data Model: Offers a more flexible and robust data schema compared to TTrees.

Trino Use Cases (as described by o1-mini)

- **Use Cases of Trino**
  - **Interactive Analytics**:
    - Ideal for business intelligence (BI) tools and dashboards that require **real-time or near-real-time query responses**.
  - **Data Lake Querying**:
    - Facilitates querying large-scale data lakes stored in platforms like Amazon S3, Google Cloud Storage, or HDFS without the need for data movement.
  - **Federated Data Access**:
    - Enables organizations to query and analyze data spread across multiple heterogeneous systems from a single interface.
  - **ETL and Data Transformation**:
    - Can be used in Extract, Transform, Load (ETL) processes to transform and move data between systems efficiently.

Model o1-mini says…

Trino in the Context of Data Storage Technologies

Given your previous interest in **ROOT TTrees**, **Apache Parquet**, and **ROOT RNTuple**, it's valuable to understand how Trino interacts with these and other columnar storage formats:

- **Apache Parquet**:
  - **Integration**: Trino has robust support for querying Parquet files, allowing users to perform SQL queries directly on Parquet-formatted data stored in data lakes or distributed file systems.
  - **Performance**: Leveraging Parquet's columnar storage, Trino can efficiently read only the necessary columns, reducing I/O and speeding up query execution.
- **ROOT TTrees and RNTuple**:
  - **Integration**: While Trino is primarily designed to work with widely-adopted data formats like Parquet, CSV, JSON, and various databases, integrating with specialized formats like ROOT TTrees or RNTuple may require custom connectors or data transformation steps.
  - **Use in HEP**: In High Energy Physics (HEP), where ROOT formats are prevalent, Trino could be used alongside data processing pipelines that convert ROOT data to more Trino-friendly formats for broader analysis and integration with other data sources.