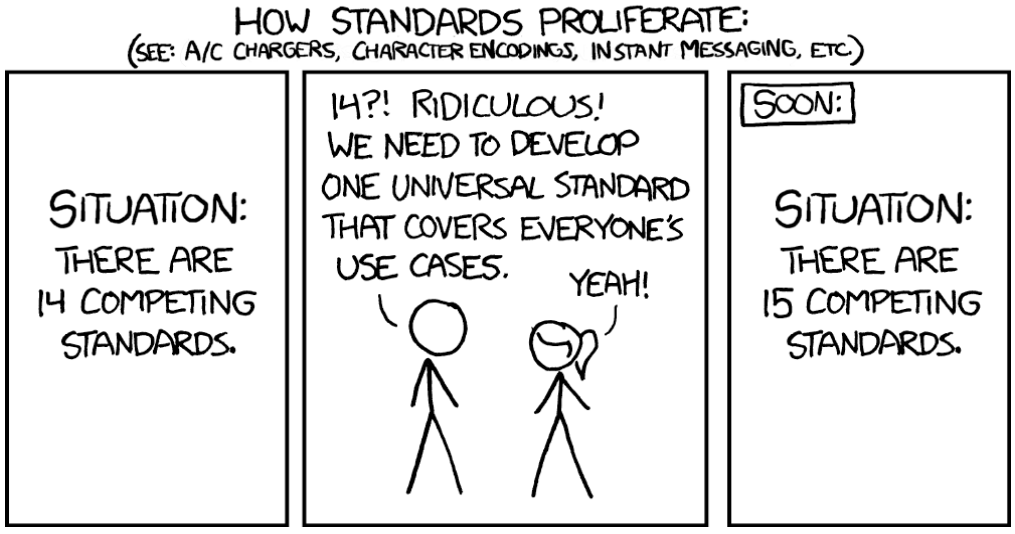


mzPeak: a next generation MS run file format

Background, challenges, questions for discussion

Nov 11, 2025 – HUPO 2025, Toronto

Joshua Klein, Tim Van Den Bossche, Samuel Wein

What is a run format?

- File format to store spectra and (some) metadata acquired during an MS Experiment.
- Vendor implementations are what we traditionally call “raw files”

Proprietary versus open formats

- Vendor formats are closed-source
  - ✓ High density, fast load and store time (by first party software)
  - ✓ Can change frequently with new tech and instruments
  - ✗ First party software preferred/exclusive
- Open formats were developed to enable exchange of data outside of vendor ecosystem
  - ✓ Allow comparison of data from different instruments
  - ✓ Usable by any third party software
  - ✓ Stable over long time span
  - ✓ Converters to and from proprietary files exist (MSConvert, ThermoRawFileParser)
  - ✗ Not always 1-to-1 mapping to vendor features and metadata
  - ✗ Larger than vendor files

The importance of open formats

- For researchers
  - Interoperability between different platforms
  - Increases diversity of available software tools
  - Long-term data preservation
- For vendors
  - Decreases cost of internal development
  - Easier to bring new software products to market
  - Cloud and AI ready

Timeline of community effort

- Sep 2023 – Public discussion at HUPO (Busan, South Korea)
- Oct 2023 – Start MZNext mailing list
- Jan 2024 – Launch poll
- May 2024 – First internal version of technical whitepaper
- Jun 2024 – Public discussion at ASMS
- Feb 2025 – Broader whitepaper mzPeak uploaded to Zenodo, request comments on PREreview for public review
- Apr 2025 – discuss mzPeak project (white paper + implementation) at HUPO-PSI
  & First meeting of technical committee
- May 2025 – Submission of white paper to JPR
- October 2, 2025 – Publication of "**mzPeak: Designing a Scalable, Interoperable, and Future-Ready Mass Spectrometry Data Format**"

mzML

- HUPO-PSI standard for storing experimental data
- Open XML-based standard
- First released in 2008, updated to 1.1.0 in 2009

mzML—a Community Standard for Mass Spectrometry Data *

Lennart Martens ‡ §, Matthew Chambers ¶, Marc Sturm ‖, Darren Kessner **, Fredrik Levander ‡‡ , Jim Shofstahl §§, Wilfred H. Tang ¶¶, Andreas Römpp ‖‖, Steffen Neumann [a], Angel D. Pizarro [b], Luisa Montecchi-Palazzi [c], Natalie Tasman [d], Mike Coleman [e], Florian Reisinger [c], Puneet Souda [f], Henning Hermjakob [c], Pierre-Alain Binz [g], Eric W. Deutsch [h]

Size and read speed issues in mzML

- MS data files have grown several orders of magnitude in size (high res, ion mobility, etc)
- XML does not support random access (unless indexed)
- Low read and write speed

Changes to the MS landscape

- Greater tolerance for non-"human-readable" formats
- Big tech led recognition of the importance of Open Source
- Scientist led recognition of the importance of FAIR science

JPR article

- Submitted in May
- Published at the beginning of October
- Describes the goals of the project and a very general roadmap
- >2000 views so far

Journal of Proteome Research > ASAP > Article

Open Access

Cite Share Jump to Expand

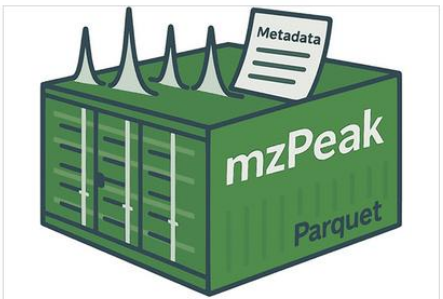
PERSPECTIVE | October 2, 2025

mzPeak: Designing a Scalable, Interoperable, and Future-Ready Mass Spectrometry Data Format

Tim Van Den Bossche, Theodore Alexandrov, Aivett Bilbao, Wout Bittremieux, Federico Ivan Brigante, Matthew Chase Chambers, Joshua Charkow, Eric Deutsch, Andrew W. Dowsey, Yasin El Abiead, Ralf Gabriels, Helge Hecht, Steffen Heuckeroth, Joshua A. Klein, Michael Knierman, Lennart Martens, Robert L. Moritz, Laura-Isobel McCall, Steffen Neumann, Yasset Perez-Riverol, Hannes L. Röst, Elliott J. Price, Jim Shofstahl, David L. Tabb, Julian Uszkoreit, Juan Antonio Vizcaíno, Mingxun Wang, Sander Willems, Dirk Winkelhardt, Oliver Kohlbacher*, and Samuel P. Wein

Open PDF | Supporting Information (1)

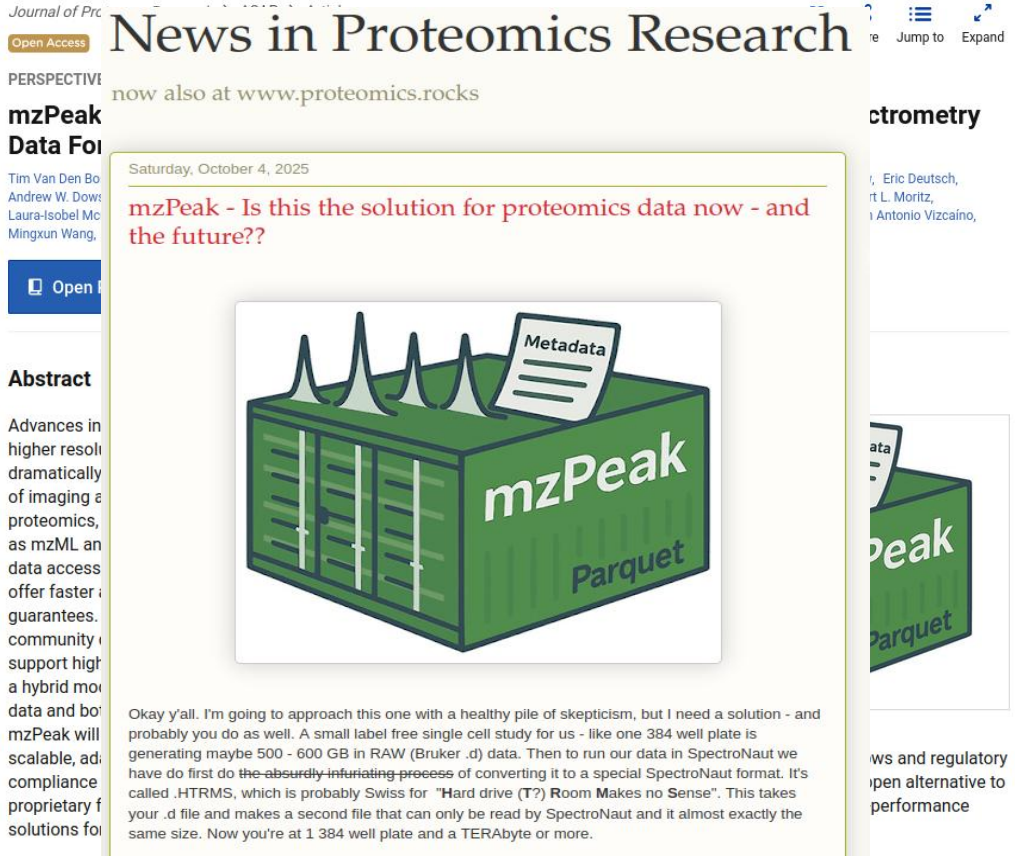
Abstract



Advances in mass spectrometry (MS) instrumentation, including higher resolution, faster scan speeds, and improved sensitivity, have dramatically increased the data volume and complexity. The adoption of imaging and ion mobility further amplifies these challenges in proteomics, metabolomics, and lipidomics. Current open formats such as mzML and imzML struggle to keep pace due to large file sizes, slow data access, and limited metadata support. Vendor-specific formats offer faster access but lack interoperability and long-term archival guarantees. We here lay the groundwork for mzPeak, a next-generation community data format designed to address these challenges and support high-throughput, multidimensional MS workflows. By adopting a hybrid model that combines efficient binary storage for numerical data and both human- and machine-readable metadata storage, mzPeak will reduce file sizes, accelerate data access, and offer a scalable, adaptable solution for evolving MS technologies. For researchers, mzPeak will support complex workflows and regulatory compliance through faster access, improved metadata, and interoperability. For vendors, it offers a streamlined, open alternative to proprietary formats. mzPeak aims to become a cornerstone of MS data management, enabling sustainable, high-performance solutions for future data types and fostering collaboration across the mass spectrometry community.

JPR article

- Submitted in May
- Published at the beginning of October
- Describes the goals of the project and a very general roadmap
- >2000 views so far
- +1 ProtemicsNews article



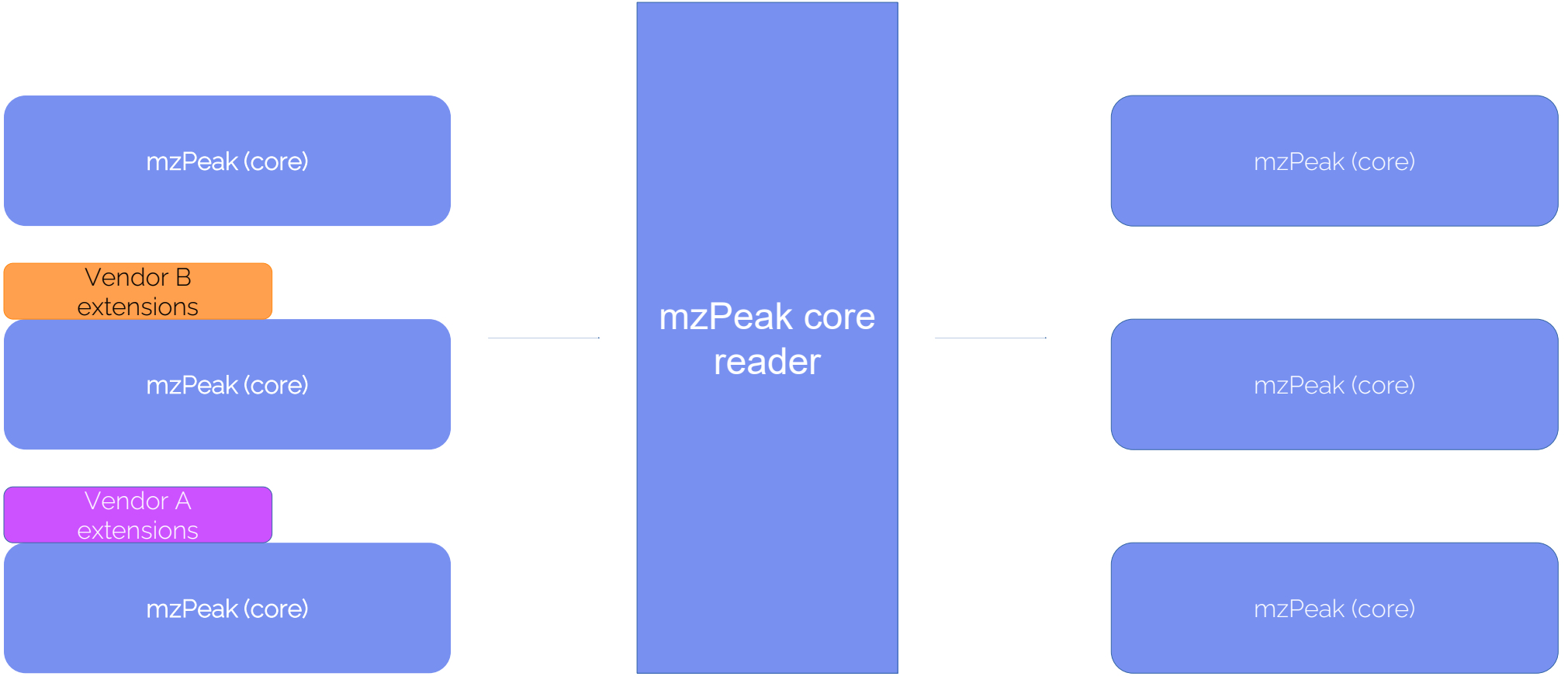
Key technical goals

- Single file on disk
  - Use an uncompressed zip file as a container for parquet tables
- “Peak-centric” long format
  - The fundamental units of measure is a m/z value and an intensity
  - future-~~proof~~ resistant against new dimensions of separation
- Uses existing controlled vocabulary
  - Add necessary terms for mzPeak specific storage
- Extendible
  - Vendors need to be able to store their own data and metadata
- Cloud native
  - Must be accessible via asynchronous IO so data can be stored in object storage

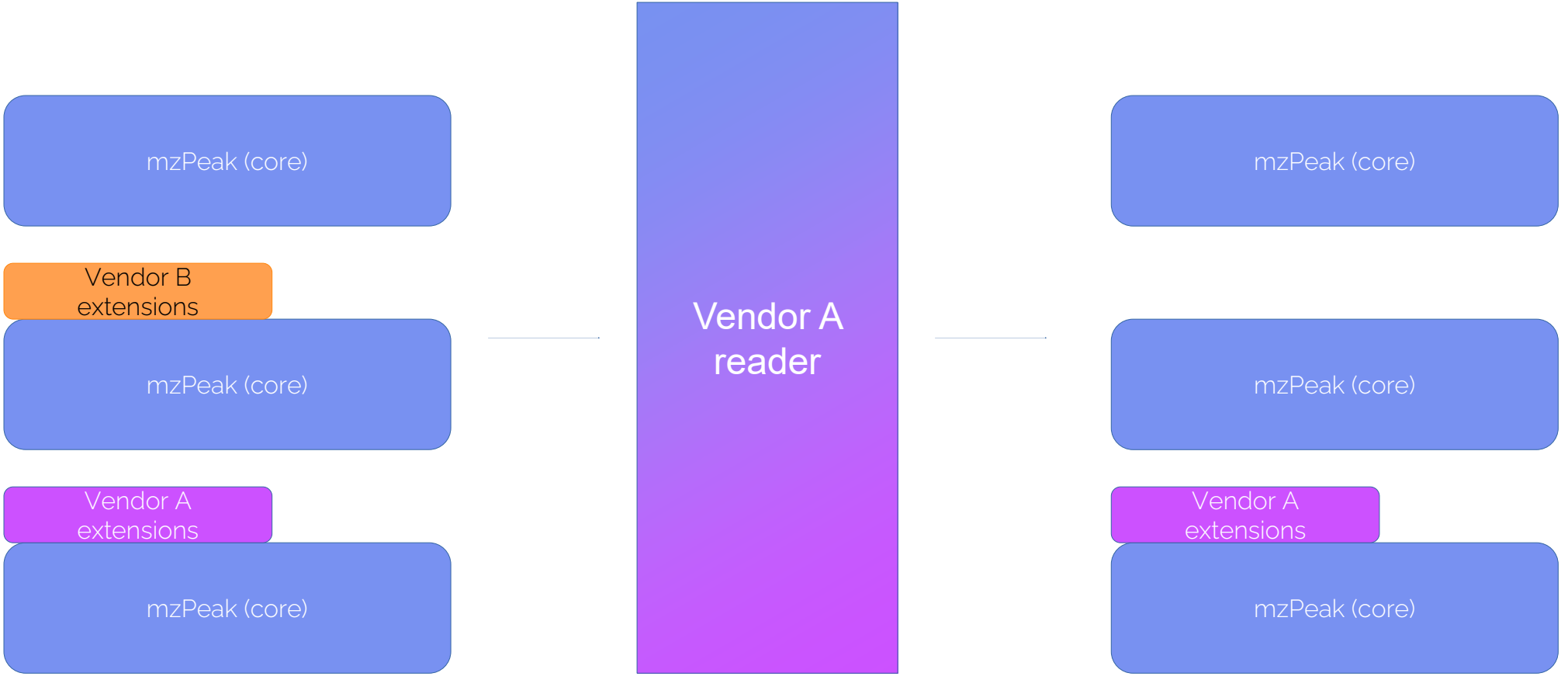
Extendibility

- Vendor raw files contain info that they care about preserving
  - Sometimes private and proprietary
  - mzPeak allows for additional data and metadata tables that can be proprietary without affecting readability of the rest of the data
  - Parquet support native encryption if a higher level of separation required
  - Goal: Lossless conversion from vendor files
- Rolling in SDRF
  - JSON objects can be embedded in mzPeak files keeping them together

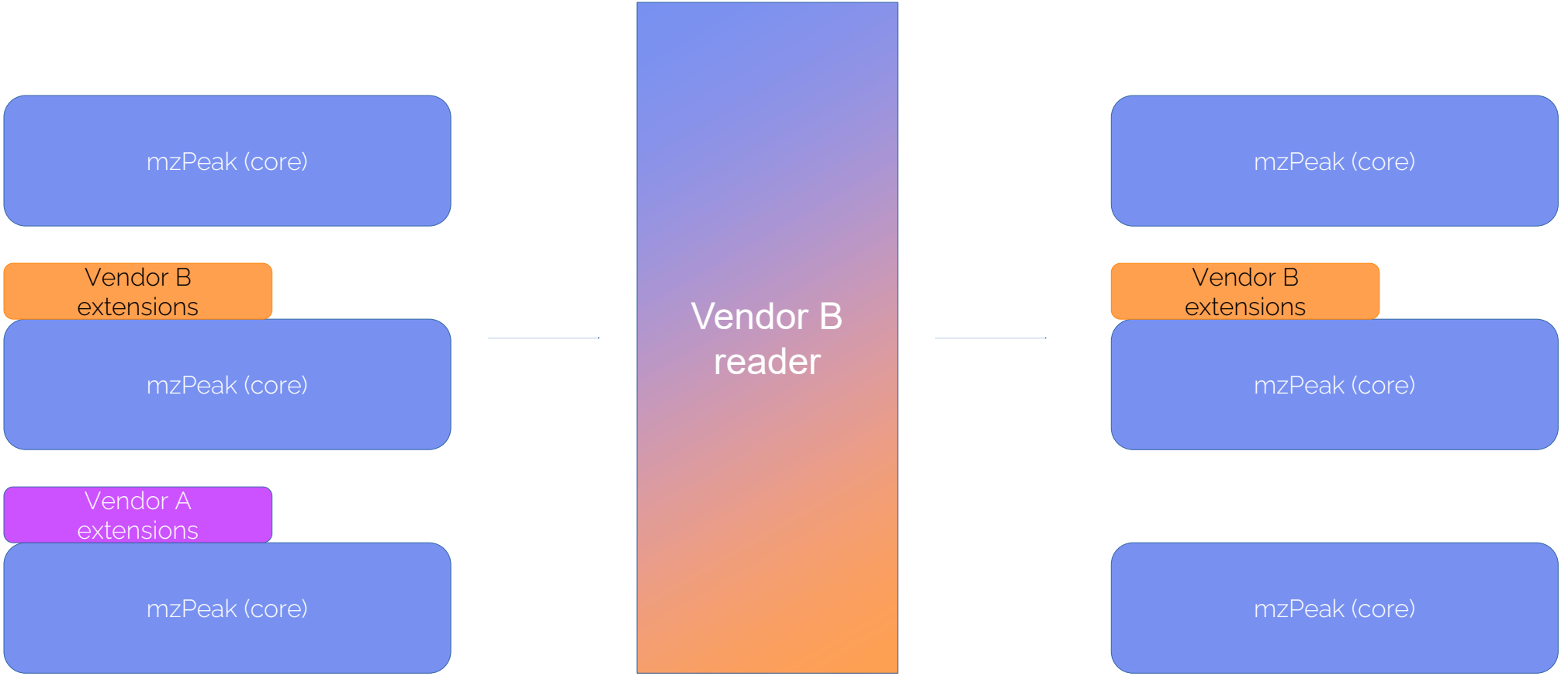
Extendability



Extendability



Extendability



Cloud nativeness

- Object storage is the future for large data
- mzPeak supports asynchronous IO over any backend that parquet supports
- Random access means that you only need to load the data you care about – saving hosting costs

Open Questions

- What should be required to be included in an mzPeak file?
- What languages are a priority?
- Can we convince vendors to build their formats on top of mzPeak?
- How do we get software developers involved?

Anatomy of an mzPeak archive

*.mzpeak

A file system directory, *uncompressed* ZIP archive, or a web address on the cloud

mzpeak_index.json

Describe the files in the archive

spectra_data.mzpeak

Parquet

Profile or centroid data defining the signal acquired for each spectrum

spectra_metadata.mzpeak

Descriptive metadata for each spectrum

chromatograms_metadata.mzpeak

Descriptive metadata for each chromatogram or trace

chromatograms_data.mzpeak

Profile data defining the signal acquired for each chromatogram

spectra_peaks.mzpeak

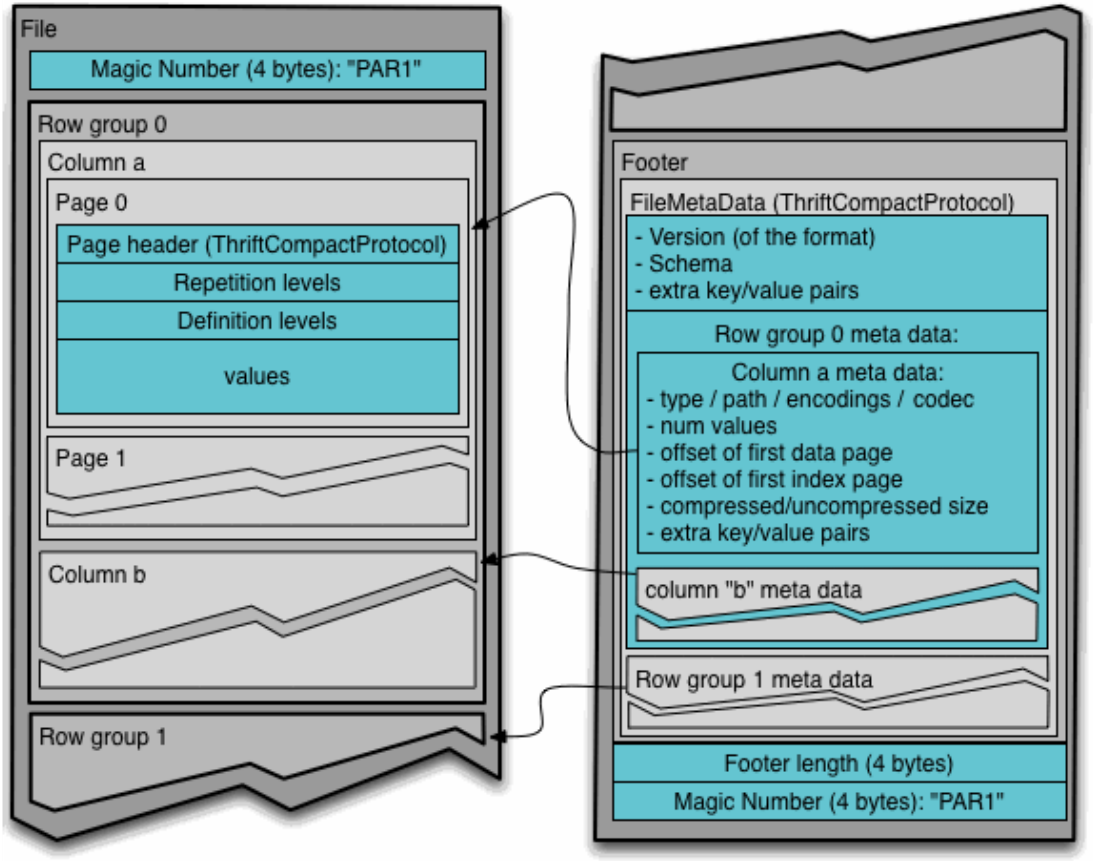
Optionally, store centroids for any profile spectrum separately, e.g. if a vendor RAW contains peaks and profiles

Any other files you like

Store any other files you want alongside the other data files as long as they do not end in ".mzpeak" or = "mzpeak_index.json"

Why Parquet

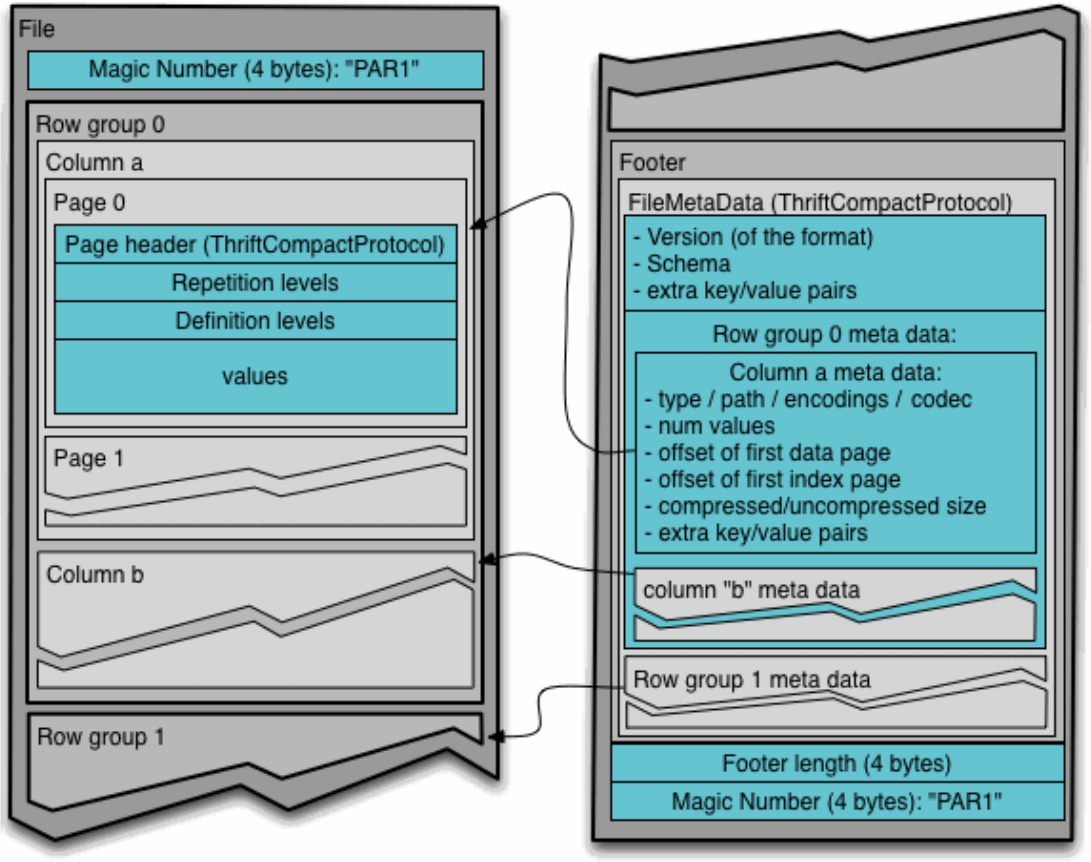
- Columnar blocked encoding and compression.
- Embedded indices make range queries fast and *combining* filters makes it even faster.
- Flexible schemas let you add more columns without impacting other readers or having to sacrifice data types.
- Stability and support across many, many languages via the Apache Foundation.
- Strong industry buy-in from cloud providers and analytics companies.



https://parquet.apache.org/docs/file-format/

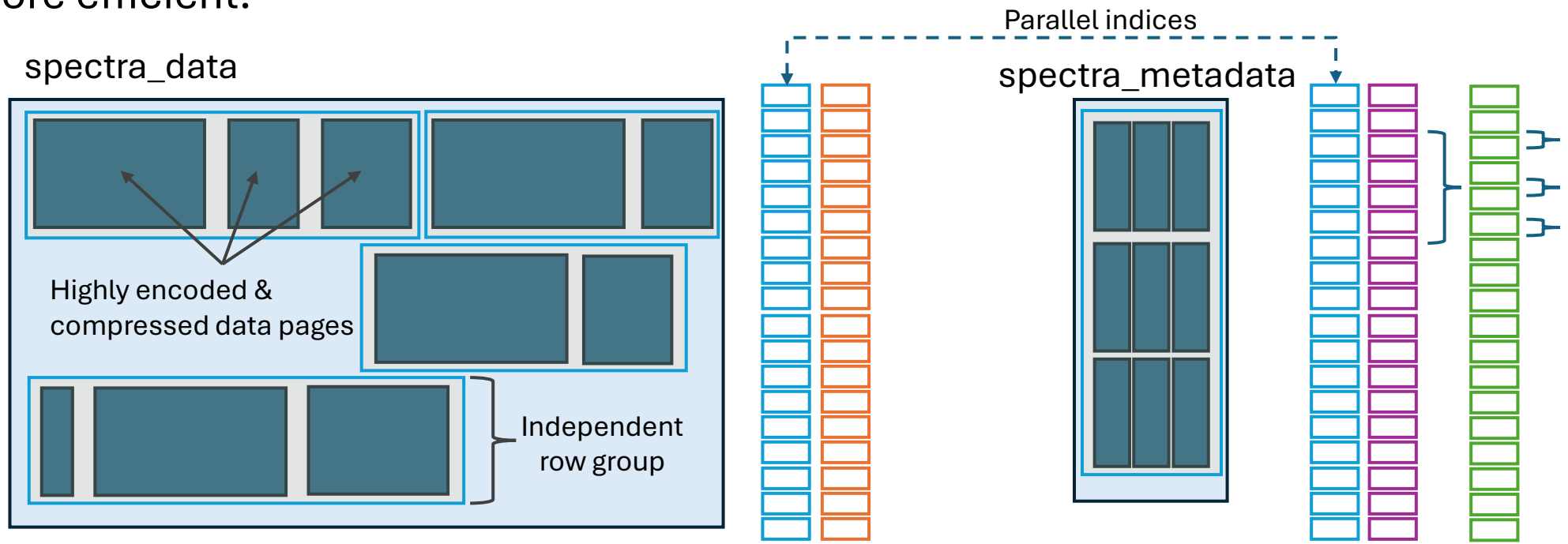
Columnar blocked compression

- Every field/column is compressed in chunks, but the compression and encoding can be tailored to the data type or content.
- If you don’t want to read a column, you can skip all costs to read its bytes, let alone decoding and decompressing it.
- There are several built-in encodings that are *transparent*, so automatic indices see the raw data, not the encoded data.
  - Byte shuffling for floating point numbers
  - Run length encoding for repetitive values
  - Dictionary encoding for large common values (or repetitive ones)
- Support for popular fast compression algorithms like Zstandard

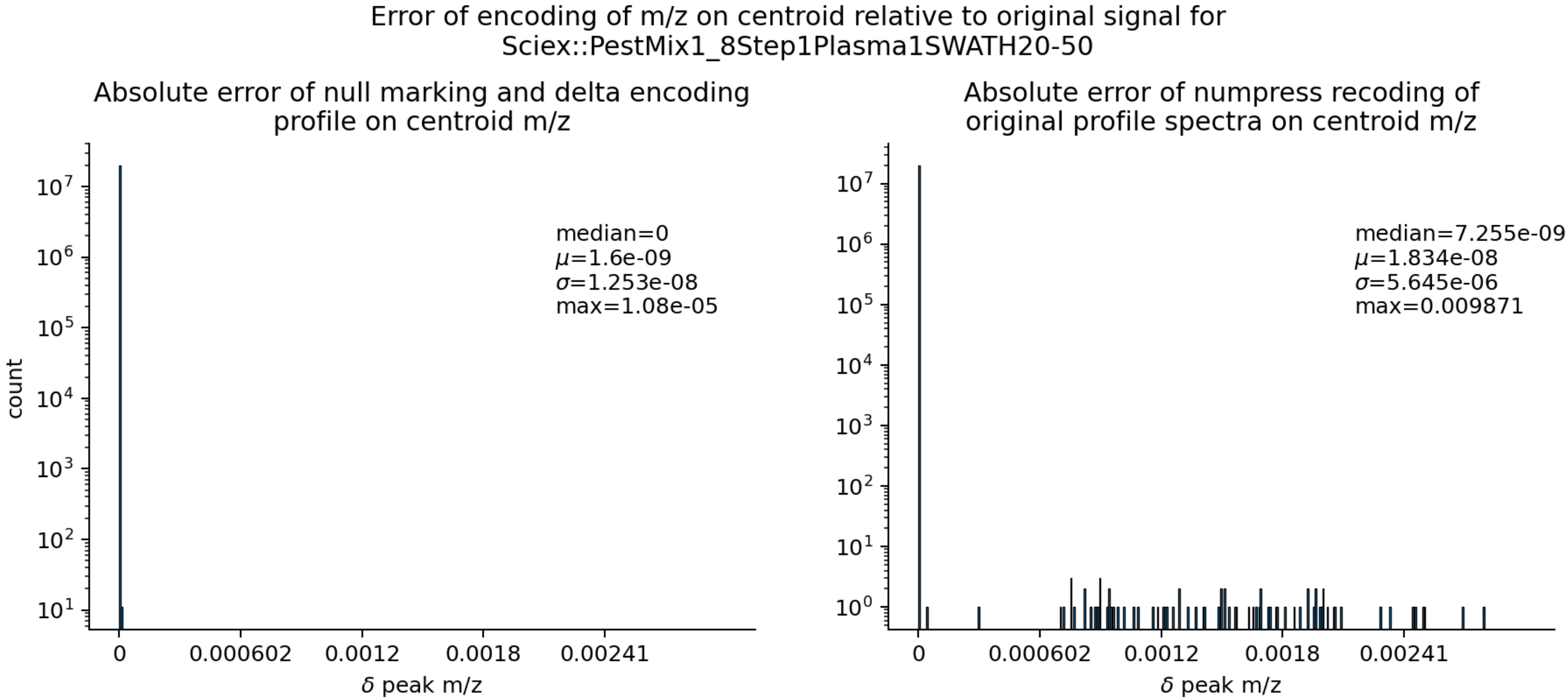


Indices and sparse reads

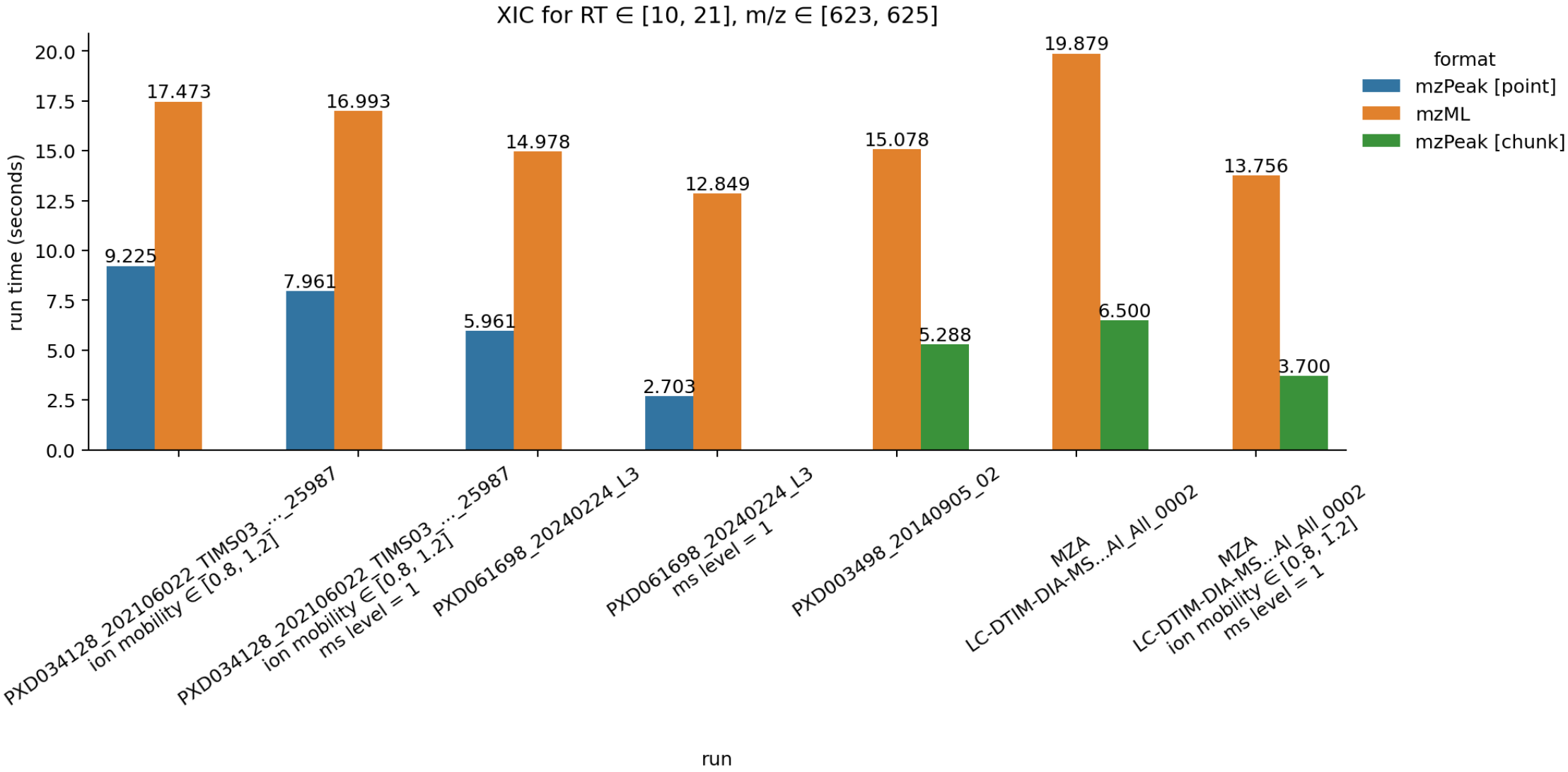
- Parquet supports range indices (min, max) at the row group and data page level. If you can isolate your target entries to a subset of them, you can skip all the other work.
- Eagerly load one column to evaluate your constraints and use *that* to decide how to sparsely read the rest of your data.
- By combining filters, e.g. start time > 5 AND start time <10 AND MS level = 1 makes it even more efficient.



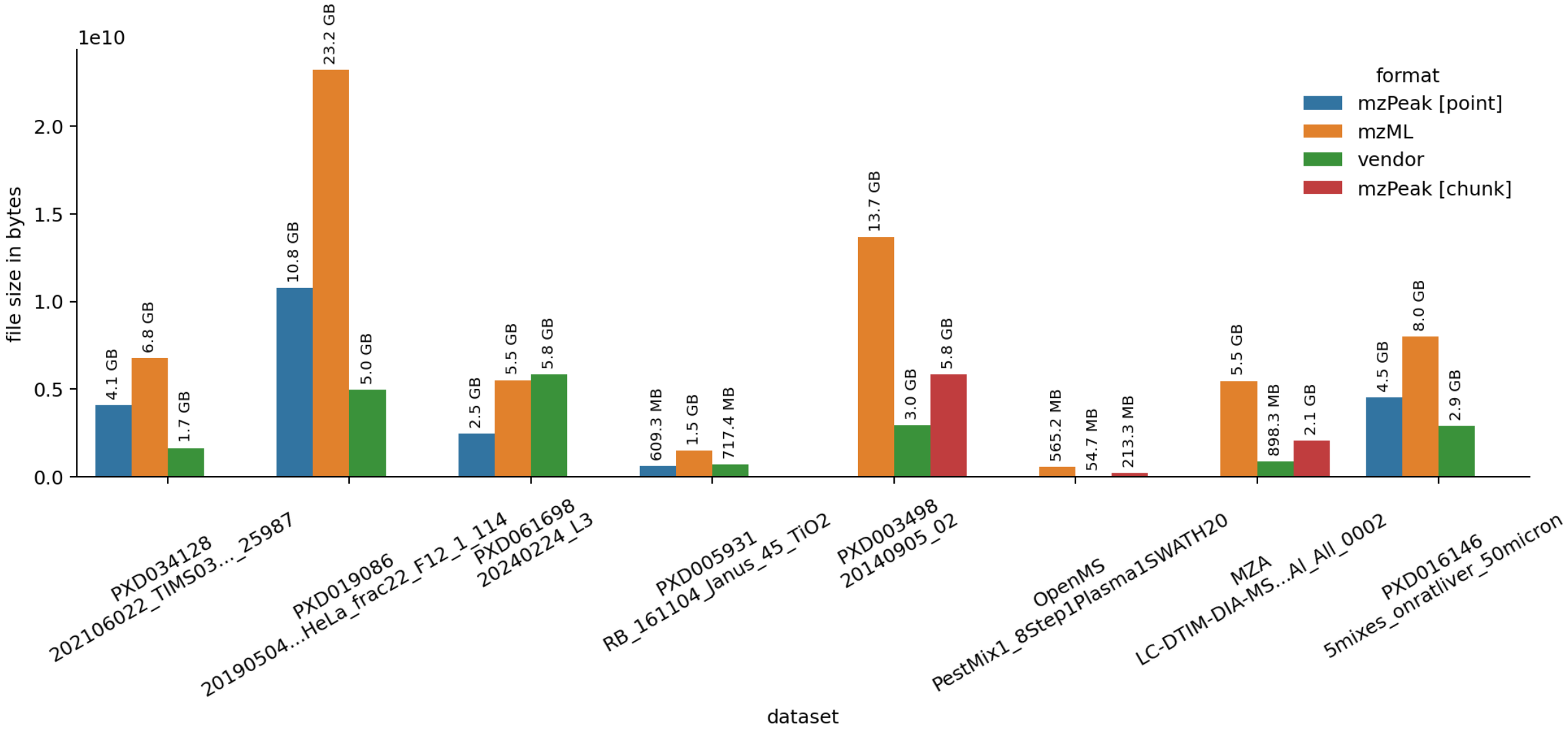
A better “lossy” compression for sparse data



Example: Faster XIC extraction



Example: Excellent Compression



#spectrum_metadata

```
CREATE TABLE spectrum (
   'index' bigint,
   id varchar,
   ms_level smallint,
   'time' real,
   polarity smallint,
   mz_signal_continuity varchar,
   spectrum_type varchar,
   number_of_data_points int,
   parameters parameter[],
   data_procesing_ref int,
   number_of_auxiliary_arrays int,
   auxiliary_arrays auxiliary_array[],
   mz_delta_model numeric[],

   total_ion_current numeric,
   base_peak_mz numeric,
   base_peak_intensity numeric,
   lowest_observed_mz numeric,
   highest_observed_mz numeric,
   ...
);

CREATE TYPE param_value AS (
   'integer' integer,
   'float' numeric,
   'bool' boolean,
   'string' varchar
);

CREATE TYPE parameter AS (
   'name' varchar,
   accession varchar,
   'value' param_value,
   unit varchar,
);
```

```
CREATE TABLE selected_ion (
   spectrum_index bigint,
   precursor_index bigint,
   selected_ion_mz numeric,
   charge_state int,
   intensity numeric,
   parameters parameter[],
   ...
);

CREATE TABLE scan (
   spectrum_index bigint,
   scan_start_time numeric,
   preset_scan_configuration int,
   filter_string varchar,
   ion_injection_time numeric,
   instrument_configuration_ref int,
   parameters parameter[],
   scan_windows scan_window[],
   ...
);
```

Flexible schemas with a common core

Define multiple tables within metadata files. A minimal core of columns are required, but additional columns can be added.

Columns may be structural or map onto controlled vocabulary terms using an inflection rule:
`<CVID>_<ACCESSION>_<NAME>`

#spectrum_data

```
CREATE TABLE point (
   spectrum_index bigint,
   mz numeric,
   intensity numeric,
   ...
);
```

Signal Data Disposition: Tradeoffs Available

Point layout

spectrum index	m/z array	intensity array
1	213.2	1002
1	506.9	500
1	758.0	405
...	...	...
2	329.1	50
2	516.5	5002
2	783.8	302

Sparse, fast, and easy to search and slice across all arrays

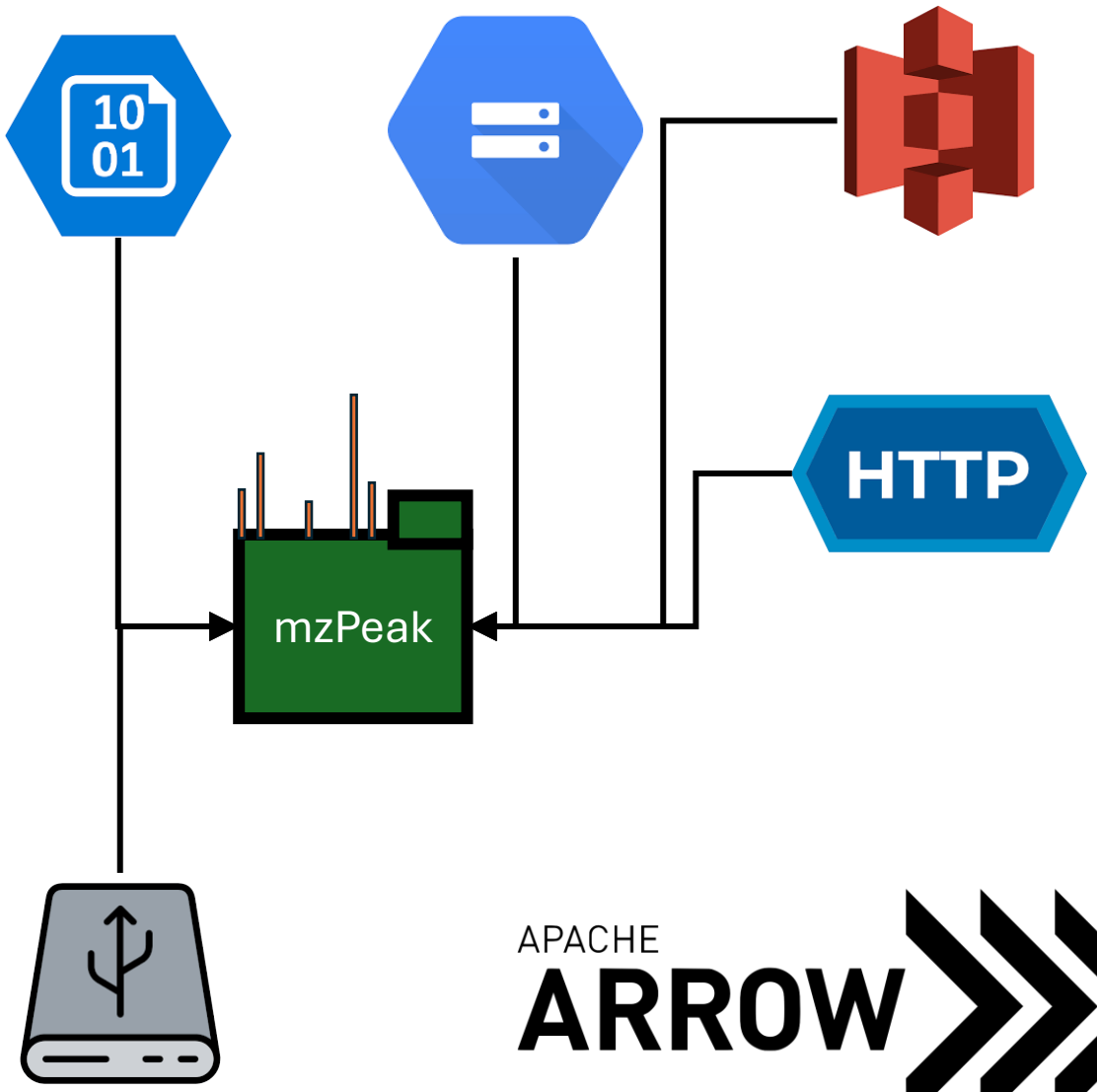
Chunk layout

Dense, easier to compress and still searchable, but slower to filter granularly

spectrum index	m/z array start	m/z array end	m/z array chunk values	intensity array
1	200.0	250.0	[0.0013, ..., 0.0013]	[...]
1	250.0	300.0	[0.0014, ..., 0.0014]	[...]
1	500.0	550.0	[0.0014, ..., 0.0015]	[...]
...	...	...	...	...
2	200.0	250.0	[0.0013, ..., 0.0013]	[...]
2	350.0	400.0	[0.0014, ..., 0.0014]	[...]
2	400.0	450.0	[0.0013, ..., 0.0014]	[...]

Read from anywhere with the Arrow Filesystem Abstraction

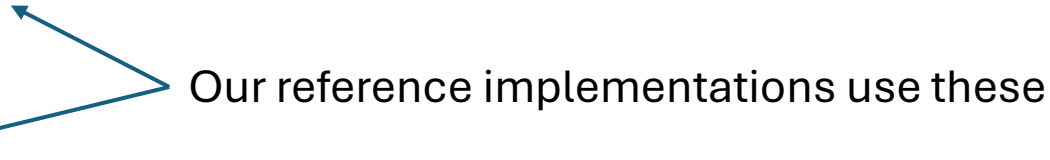
- Read from disk
- Read from a byte array in RAM or mem-mapped file
- Read from an object store in the Cloud like AWS S3
- Read from an HTTP server with Content-Range support



APACHE ARROW

Core features of Parquet are supported in many languages

- C++*
  - **Python**
  - R
  - …
- **Rust***
- Java*
- C#
- Go*
- JavaScript
- Julia



* - Official Apache implementations