# Performance Visualization of ROOT I/O on HPC Storage Systems

*Lightning Talks*

Rui Reis

06/09/2021

# Contextualization

*Current state-of-the-art*

- In order to analyse the tremendous amount of data generated from High Energy Physics (HEP) experiments, CERN uses the ROOT framework.
- Previous versions of ROOT used the TTtree data format, however, it will be soon replaced in v7 by RNTuple, an efficient columnar storage format developed by CERN.
- RNTuple also provides a set of metrics for the analysis of data ingestion performance, users can also add custom metrics, as desired.

```
ntuple->EnableMetrics();
```

```
RNTupleMetrics inner("inner");
auto ctr = inner.MakeCounter<RNTuplePlainCounter *>("plain", "s", "example 1");
```

# Contextualization

*Metrics simplicity*

Despite its capability to provide the user with useful insights, current RNTuple metrics constraint to counter aggregate type metrics, which are too simple in many scenarios (*e.g.*, when we need to analyse the distribution of a certain metric)

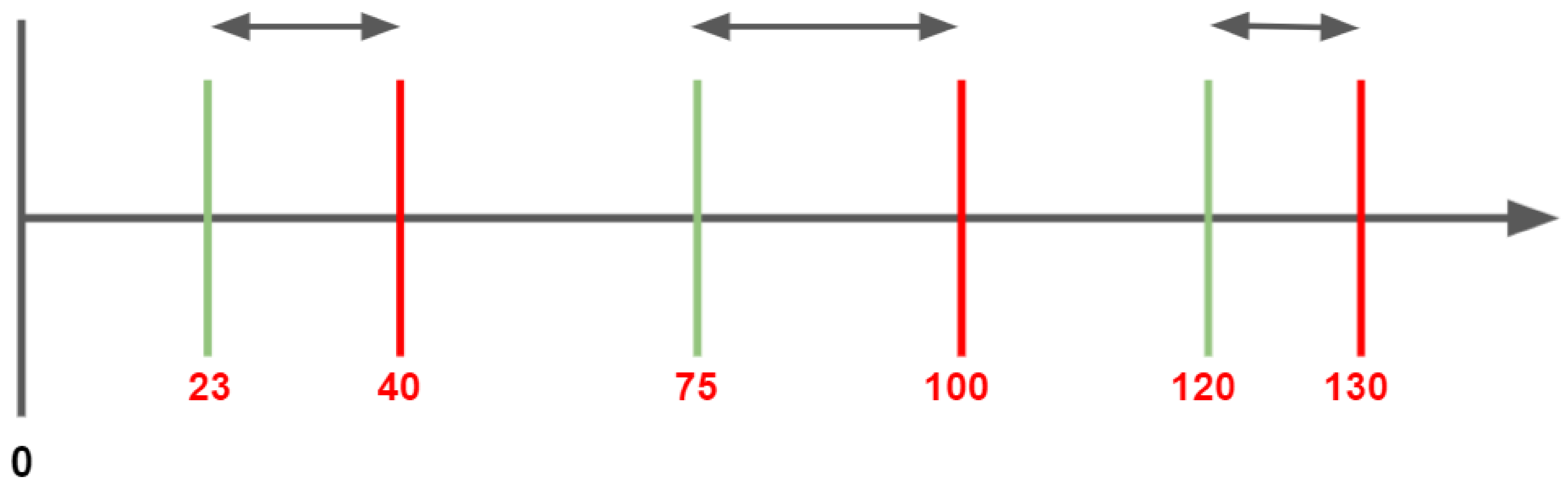
```
RNTupleReader.RPageSourceFile.nReadV||number of vector read requests|16
RNTupleReader.RPageSourceFile.nRead||number of byte ranges read|16
RNTupleReader.RPageSourceFile.szReadPayload|B|volume read from storage (required)|7451486
RNTupleReader.RPageSourceFile.szReadOverhead|B|volume read from storage (overhead)|3196
RNTupleReader.RPageSourceFile.szUnzip|B|volume after unzipping|8000000
RNTupleReader.RPageSourceFile.nClusterLoaded||number of partial clusters preloaded from storage|16
RNTupleReader.RPageSourceFile.nPageLoaded||number of pages loaded from storage|110
RNTupleReader.RPageSourceFile.nPagePopulated||number of populated pages|110
RNTupleReader.RPageSourceFile.timeWallRead|ns|wall clock time spent reading|2571172
RNTupleReader.RPageSourceFile.timeWallUnzip|ns|wall clock time spent decompressing|10002890
RNTupleReader.RPageSourceFile.timeCpuRead|ns|CPU time spent reading|4325000
RNTupleReader.RPageSourceFile.timeCpuUnzip|ns|CPU time spent decompressing|11030000
RNTupleReader.RPageSourceFile.bwRead|MB/s|bandwidth compressed bytes read per second|2899.332289
RNTupleReader.RPageSourceFile.bwReadUnzip|MB/s|bandwidth uncompressed bytes read per second|3111.421562
RNTupleReader.RPageSourceFile.bwUnzip|MB/s|decompression bandwidth of uncompressed bytes per second|799.768867
RNTupleReader.RPageSourceFile.rtReadEfficiency||ratio of payload over all bytes read|0.999571
RNTupleReader.RPageSourceFile.rtCompression||ratio of compressed bytes / uncompressed bytes|0.931436
```

# Challenges

- Current RNTuple metrics are too simple to provide viable information about the distribution of data.
- Which makes the following questions hard to answer:
  - What is the distribution of the size of read requests to load an entire ntuple cluster?
  - How can we know if our ntuple metrics are unevenly distributed?
  - How can we detect the existence of outliers in our metrics?
- Possible solutions need to be efficient and be able to construct histograms on-the-go.

# Solution #1

*User-Provided Set of Intervals*

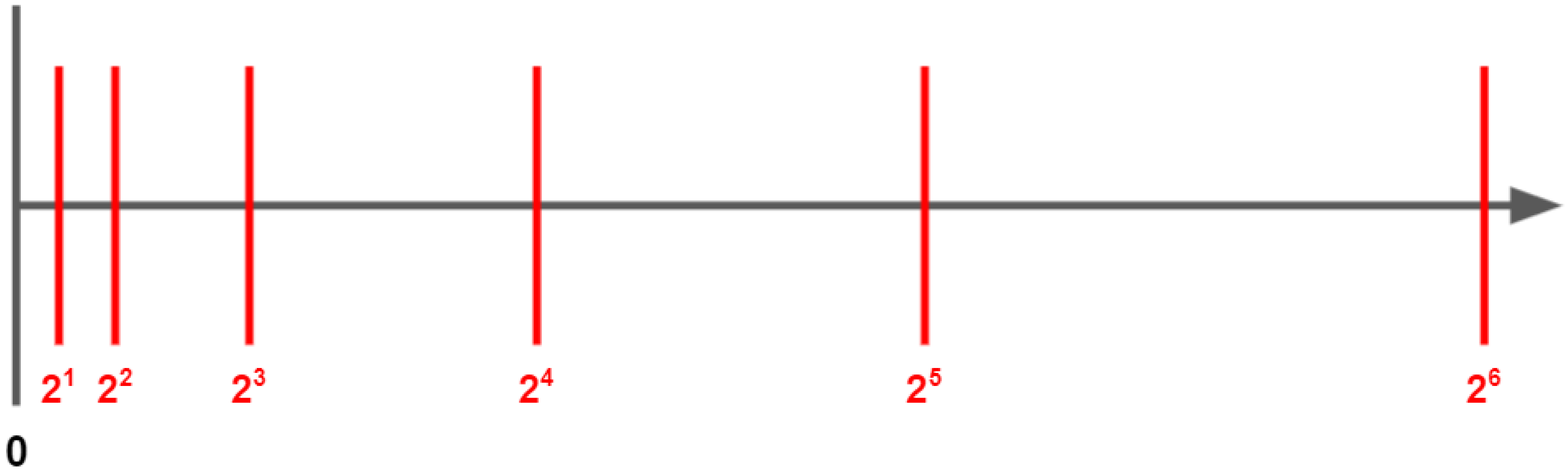



**Cons:**

- Requires knowledge of underlying data
- Unable to detect outliers
- Error-prone

# Solution #2

*Log Scale*

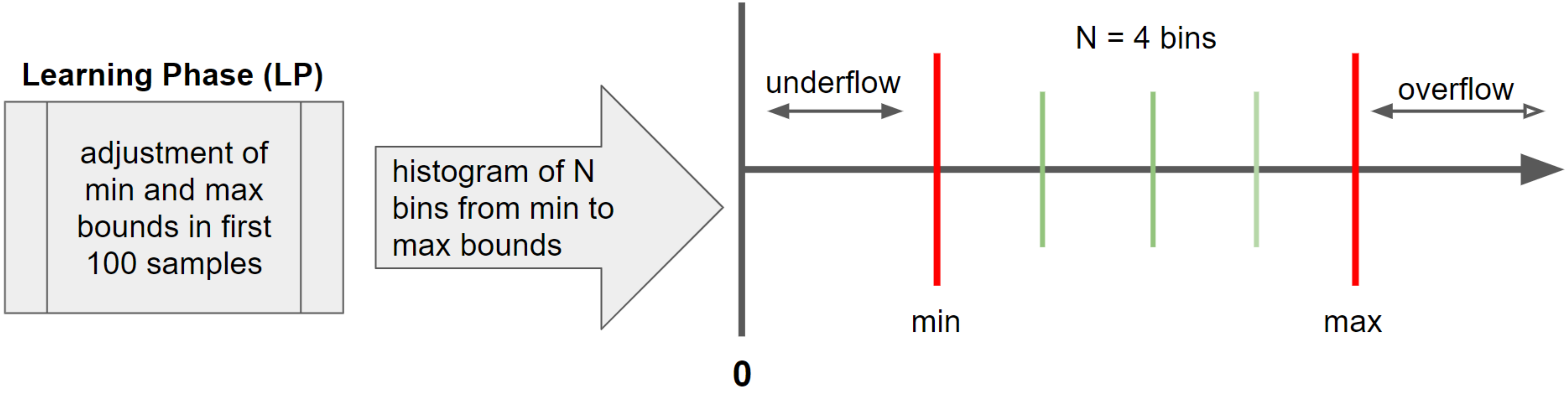



**Cons:**

- Amplitude of intervals is exponentially large
- Hard to interpret the meaning of histogram output
- Able to detect some outliers, depending on scale

# Solution #3

*Active Learning Phase*

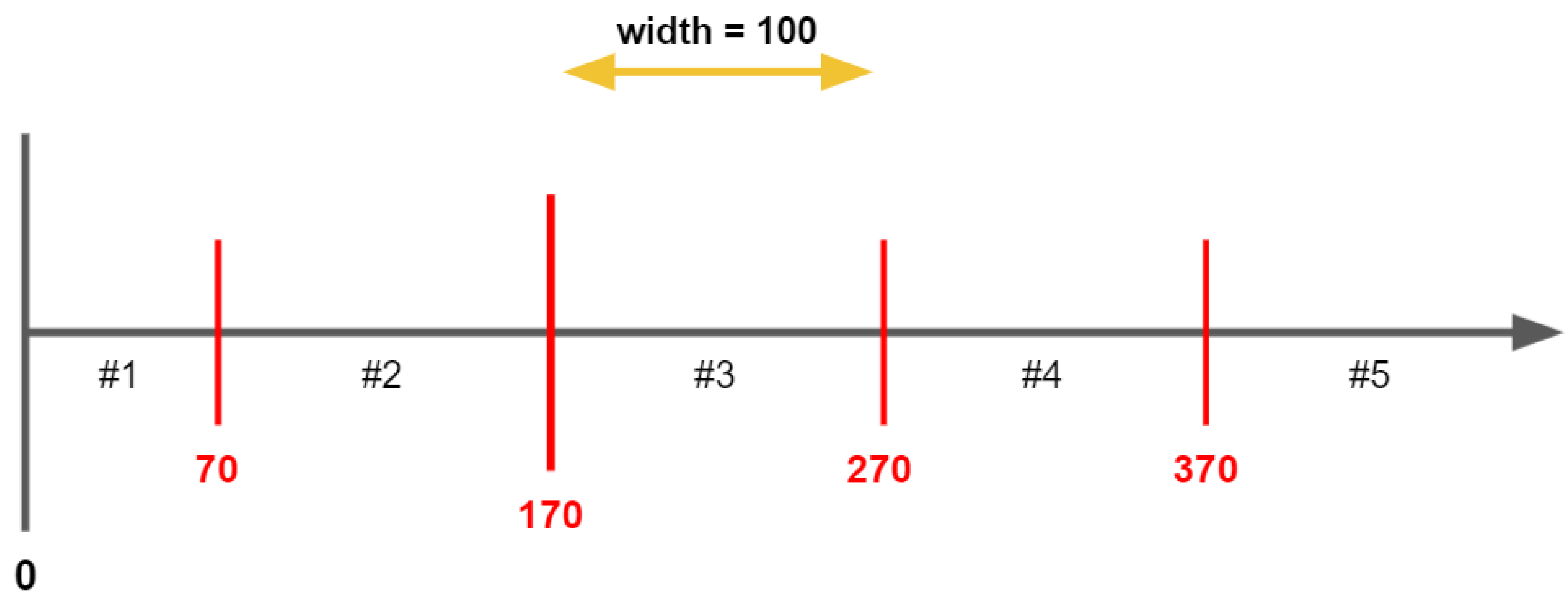



**Cons:**

- Heavily dependant on the distribution of samples in the LP
- Occurrence of outliers in LP deeply affects efficiency of histogram
- Can’t effectively separate outliers from real distribution

# Solution #4

*Fixed Width Bins*



# Solution #4

*Calculating the bin key (Fill Algorithm)*

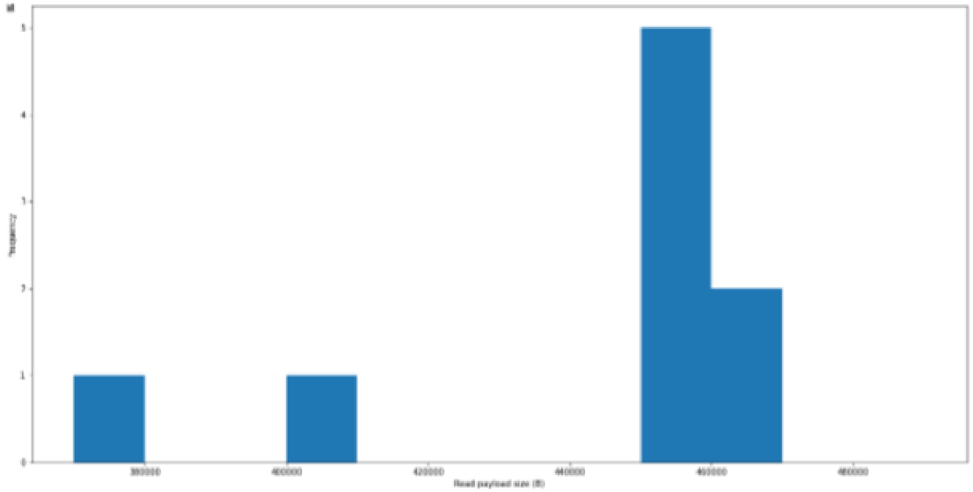
- If a new value, N, is **greater or equal** to the **offset**, then:
  - Key = (N – offset) / width + #{below offset bins} + 1
- Else:
  - Key = #{below offset bins} – (offset – N) / width
- Examples, width=100, offset=170:
  - N = 178 ⇔ key = (178 – 170) / 100 + 2 + 1 = 3
  - N = 384 ⇔ key = (384 – 170) / 100 + 2 + 1 = 5
  - N = 105 ⇔ key = 2 – (170 – 105) / 100 = 2
  - N = 69 ⇔ key = 2 – (170 – 69) / 100 = 1

# Sample #1

*ROOT I/O - Tutorial #5*

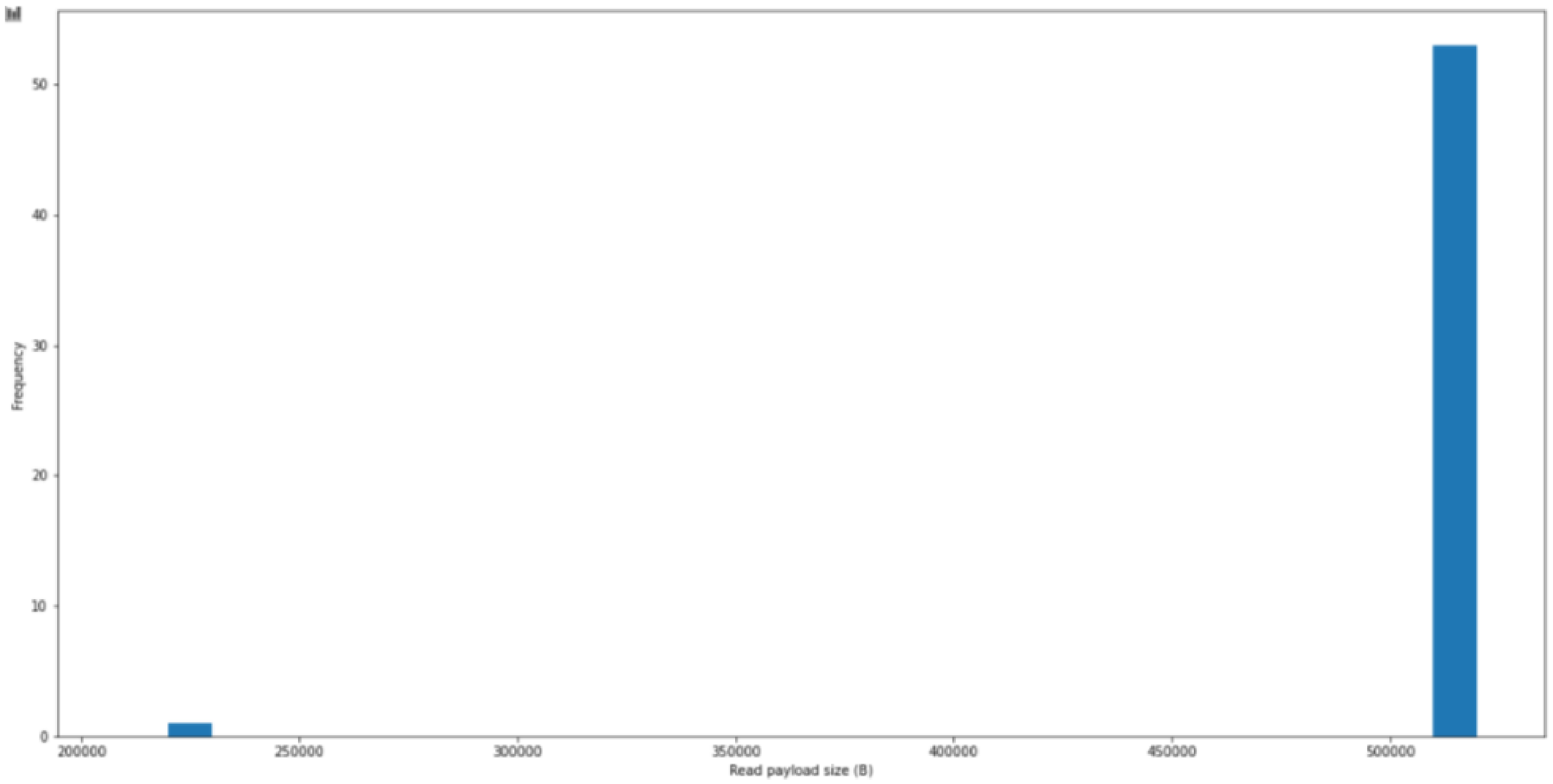
After the desired analysis, the histogram content can be dumped as a CSV and fed to external plotting utilities for visual analysis.

```
lower_bound,upper_bound,count
370000,379999,1
400000,409999,1
450000,459999,5
460000,469999,2
490000,499999,7
```



# Sample #2

*Convert LHC 1 run open data from TTree to RNtuple*



# Conclusion

- Performance visualization can easily allow a detailed analysis of the underlying metrics.
- Histogram output format can be easily ingested by external plotting utilities.
- More information can be found on the PR: [ntuple] Performance visualization improvements by ruipreis · Pull Request #8880 · root-project/root (github.com)

# QUESTIONS?

*ruipedronetoreis12@gmail.com*