PDSW24 | 2024-11-17

MOSAIC: DETECTION AND CATEGORIZATION OF I/O PATTERNS IN HPC APPLICATIONS

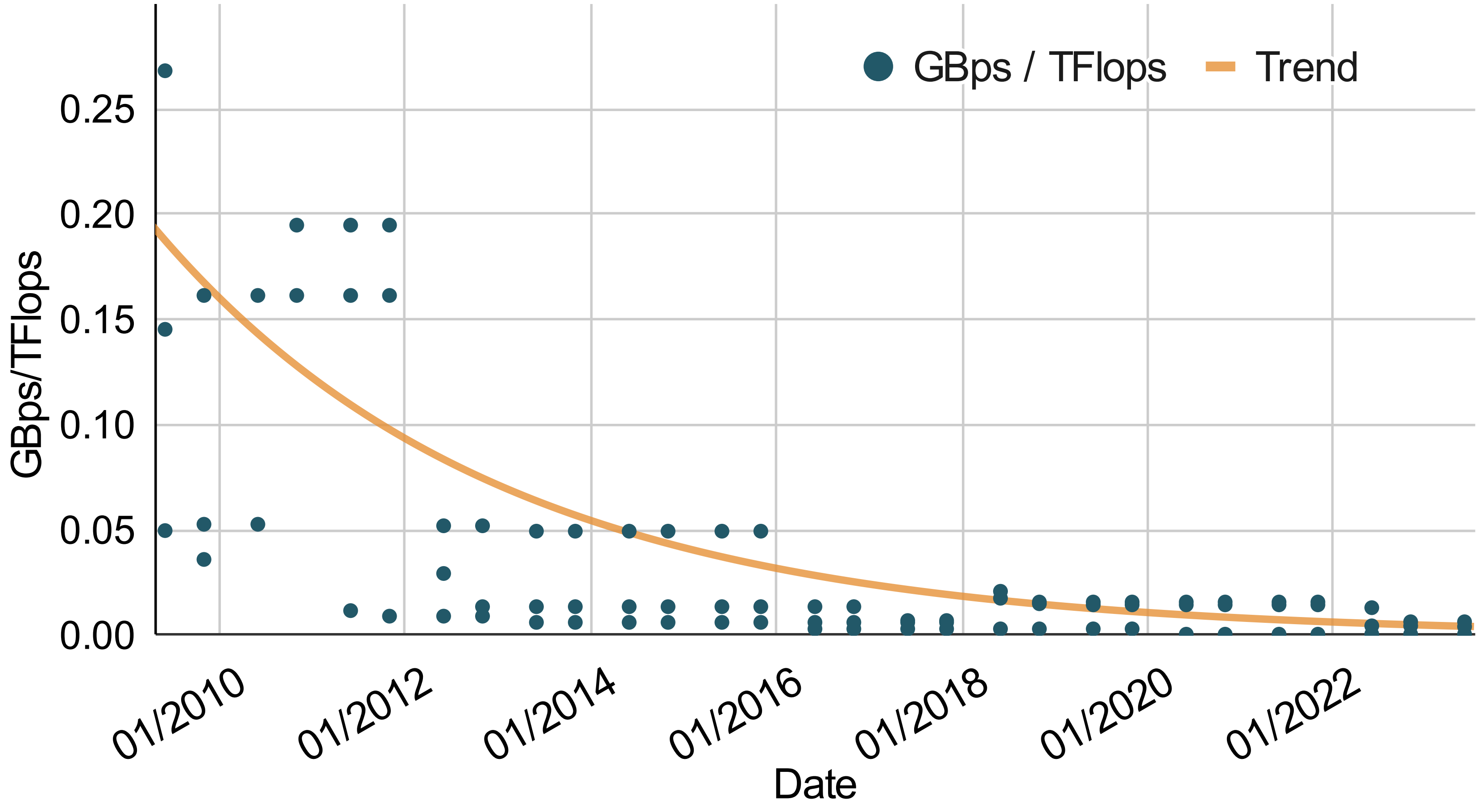
Théo Jolivel, François Tessier, Julien Monniot, Guillaume Pallez

Inria

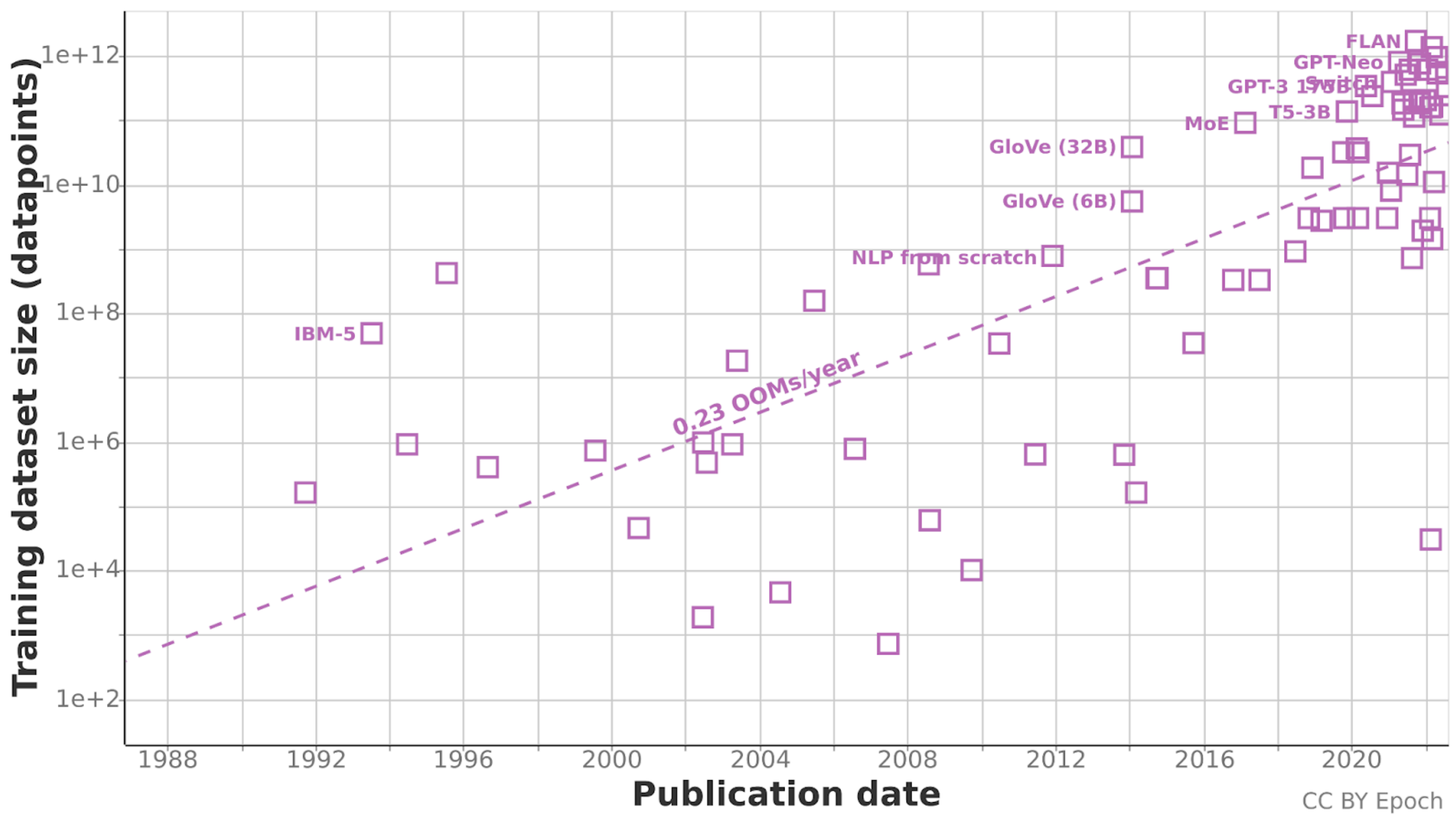
UMR IRISA

FRANCE 2030 RESEARCH PROGRAM DIGITAL FOR EXASCALE

Why do we need to optimize HPC storage systems?



Ratio of I/O bandwidth (GBps) to computing power (TFlops) of the top 3 supercomputers from the Top500 ranking over the past 15 years.



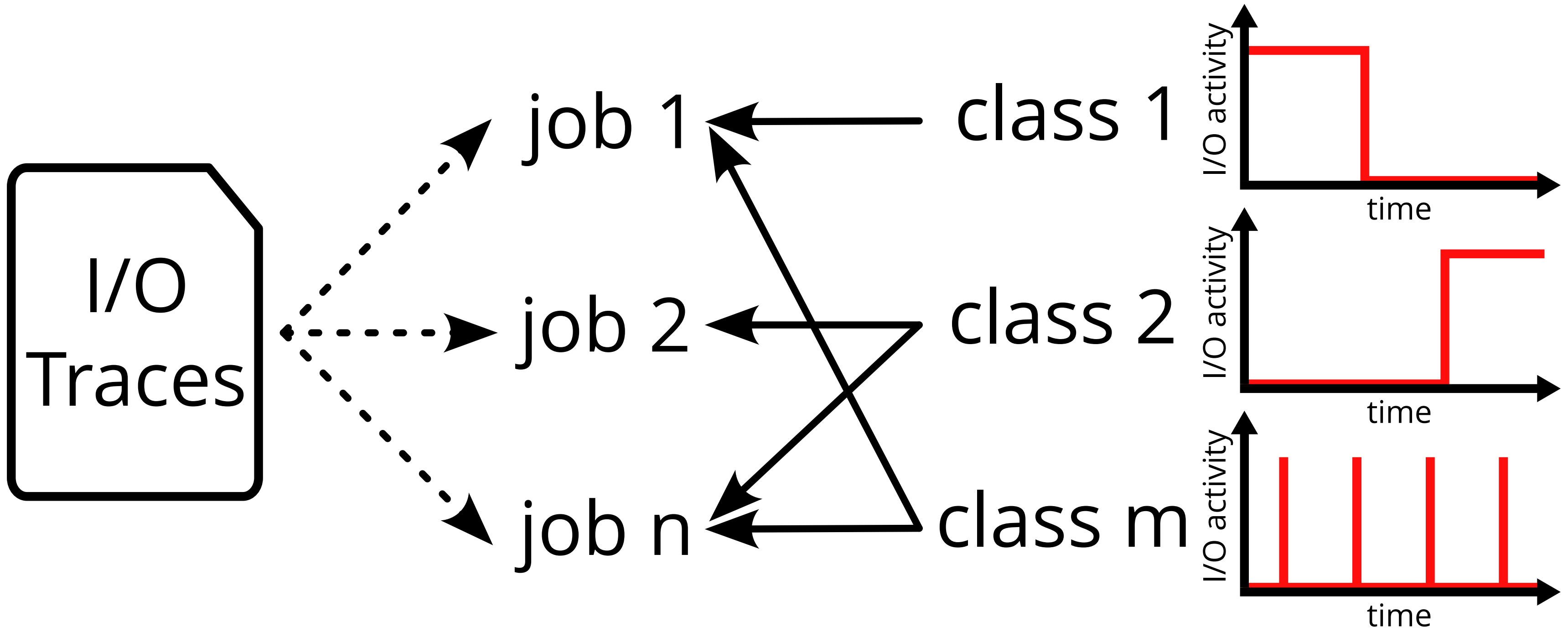
Evolution of training dataset's sizes for machine learning applications.
Pablo Villalobos and Anson Ho (2022) "Trends in Training Dataset Sizes"

How to optimize storage access on supercomputers?

Multiple ways of achieving I/O efficiency (non exhaustive):

- **Application level:** use of advanced I/O mechanisms in applications' codes.
- **I/O library level:** I/O optimizations inside libraries to transparently perform operations efficiently.
- **Scheduler level:** make smart scheduling decisions to avoid concurrency between jobs.
- **Storage level:** adopt I/O-aware policies (e.g. data distribution) within the parallel file system (PFS).

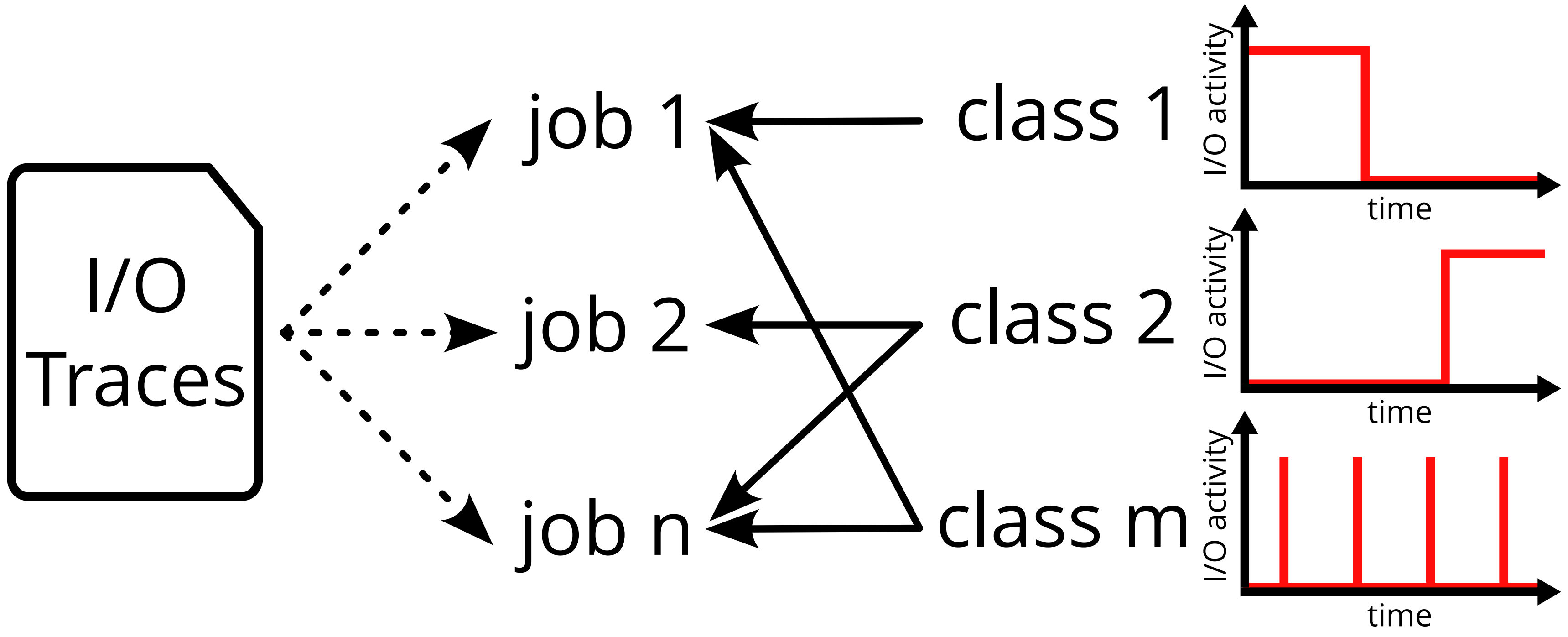
Fundamental prerequisite: have a good understanding of how HPC applications behave.



How to optimize storage access on supercomputers?

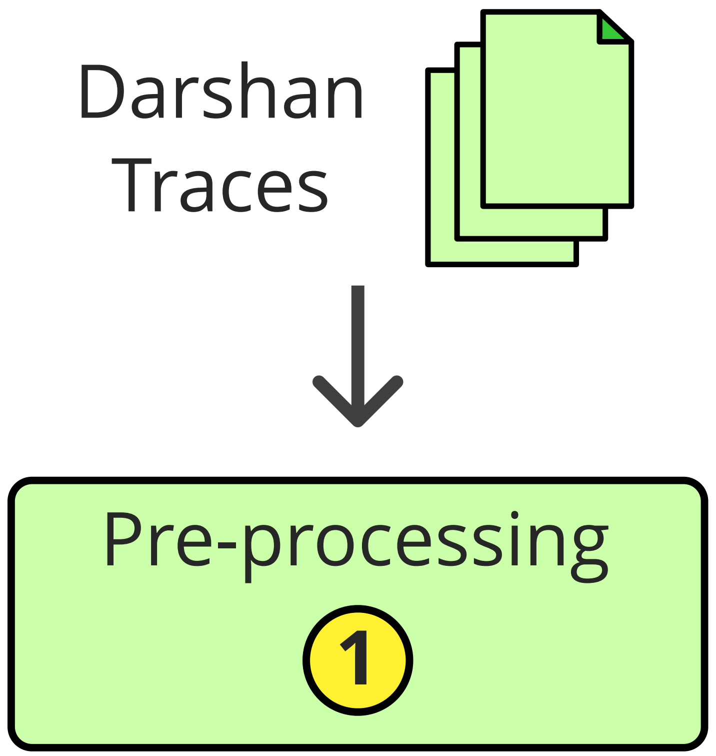
We introduce **MOSAIC**: an I/O-aware categorization tool able to describe access patterns from I/O traces, including: access temporality, periodicity and metadata overhead.

Fundamental prerequisite: have a good understanding of how HPC applications behave.



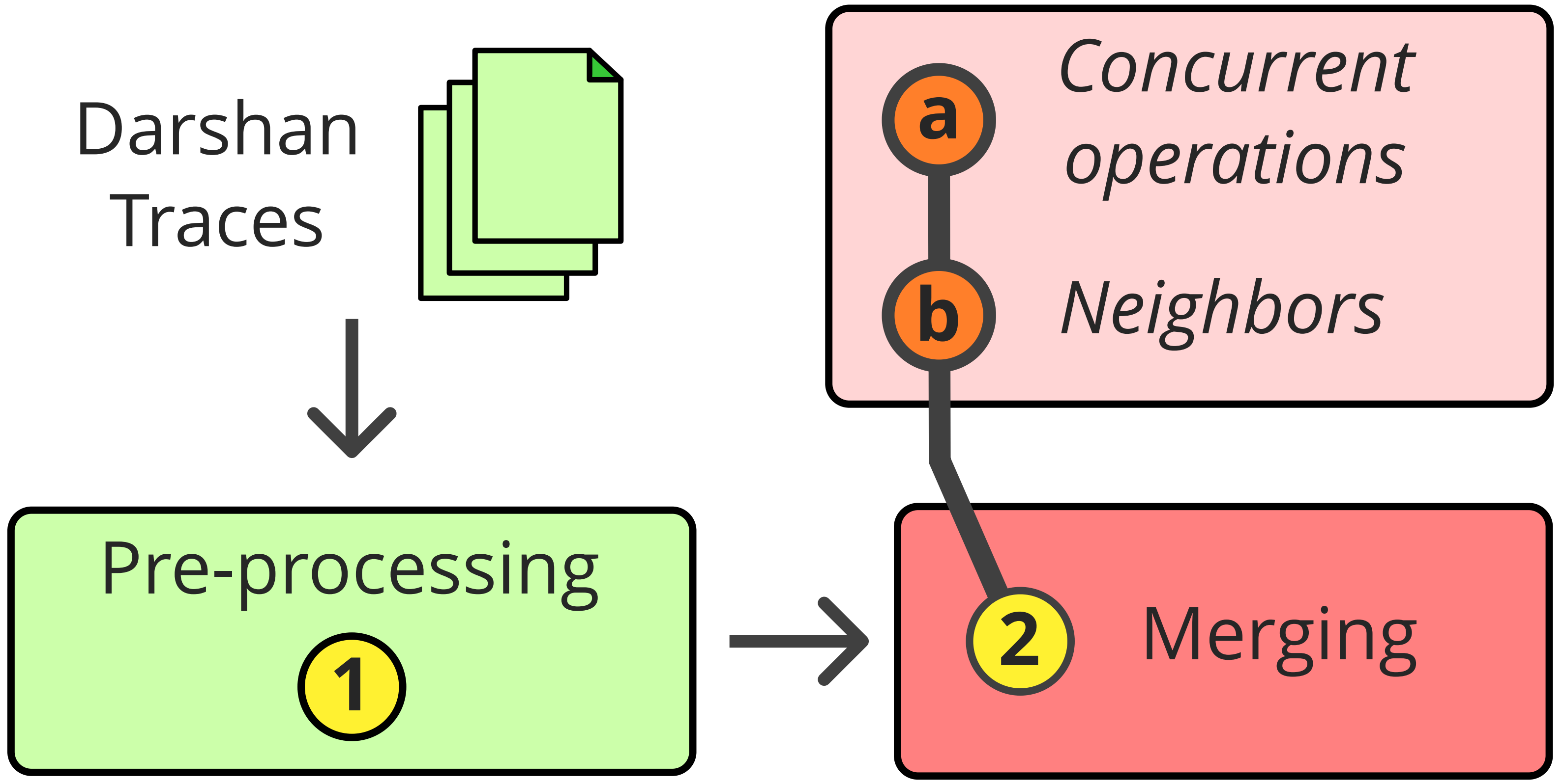
TRACE PROCESSING METHODOLOGY

MOSAIC trace processing workflow



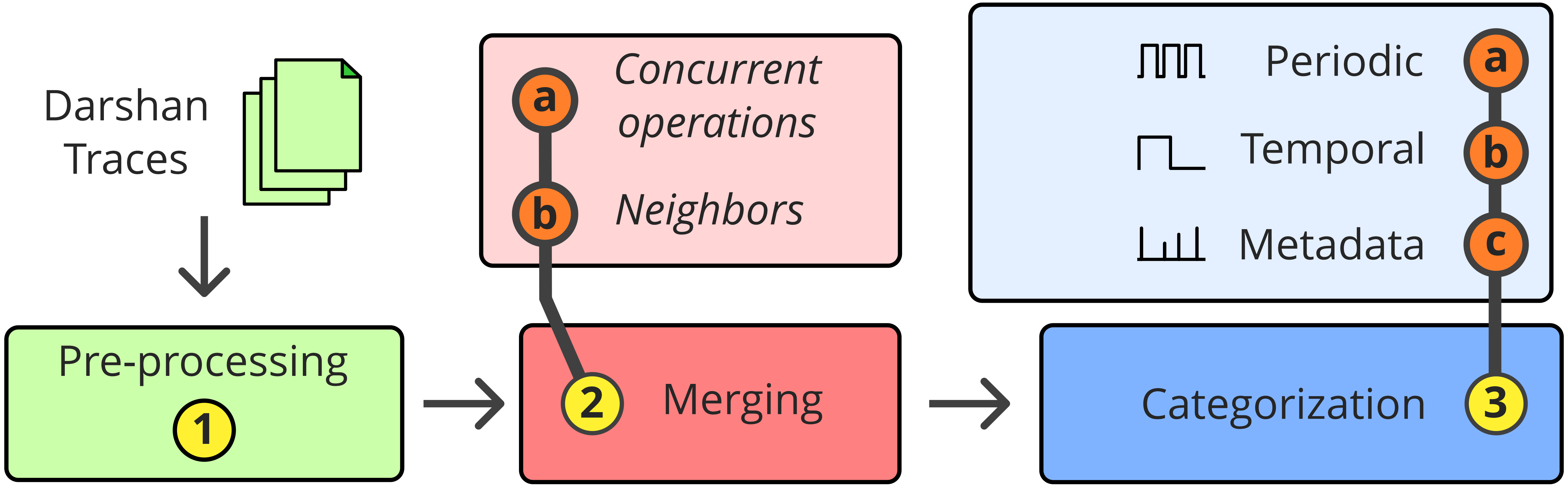
Pre-processing: go through every Darshan trace file to retain only one representative trace for each unique execution (user, execution cmd).

MOSAIC trace processing workflow



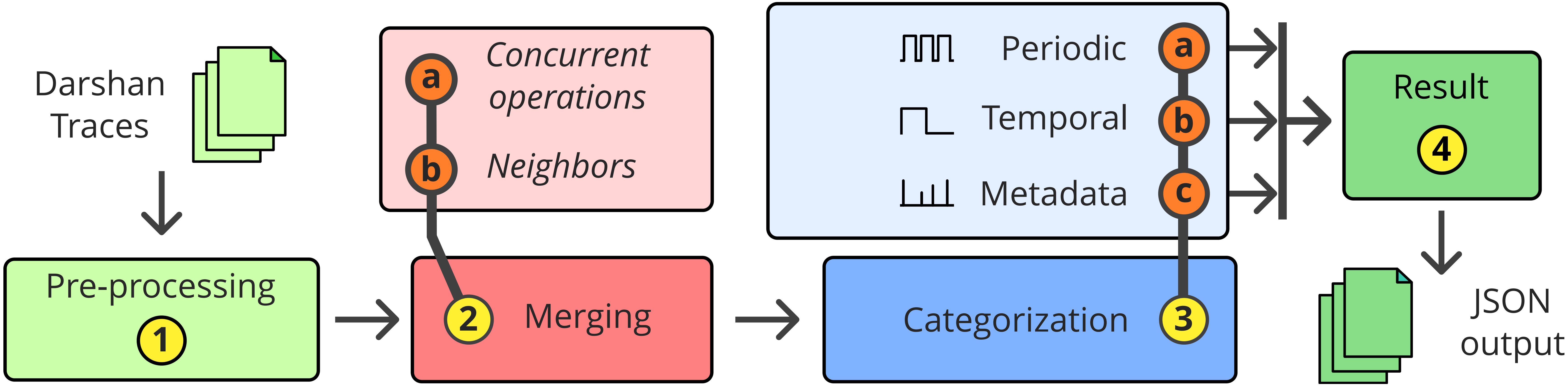
Merging: fuse concurrent operations and close neighbors to have a linear list of operations (the start of the next operation is after the end of the previous one).

MOSAIC trace processing workflow



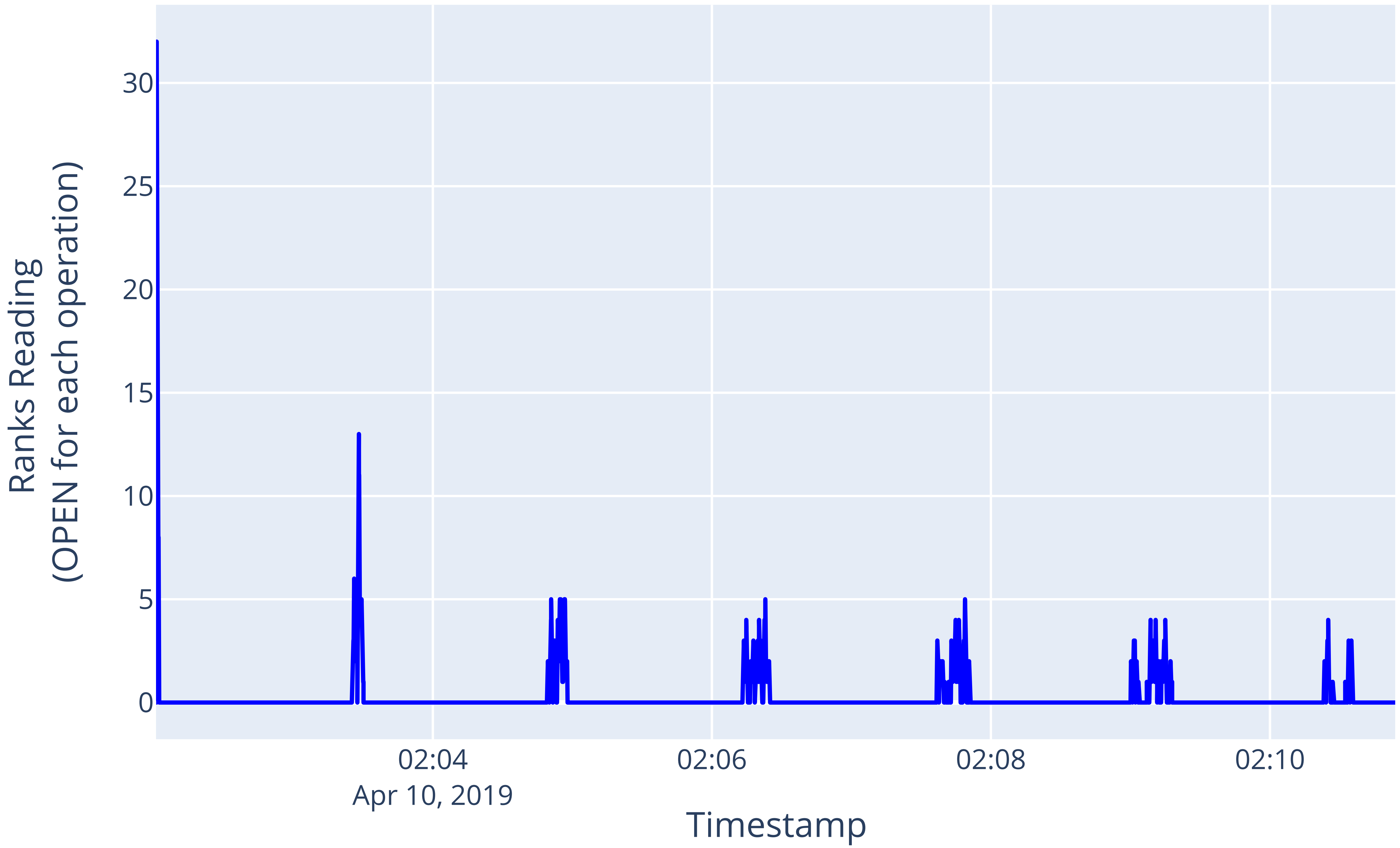
Categorization: process the traces to assign classes about metadata patterns, operation periodicity and access temporality.

MOSAIC trace processing workflow



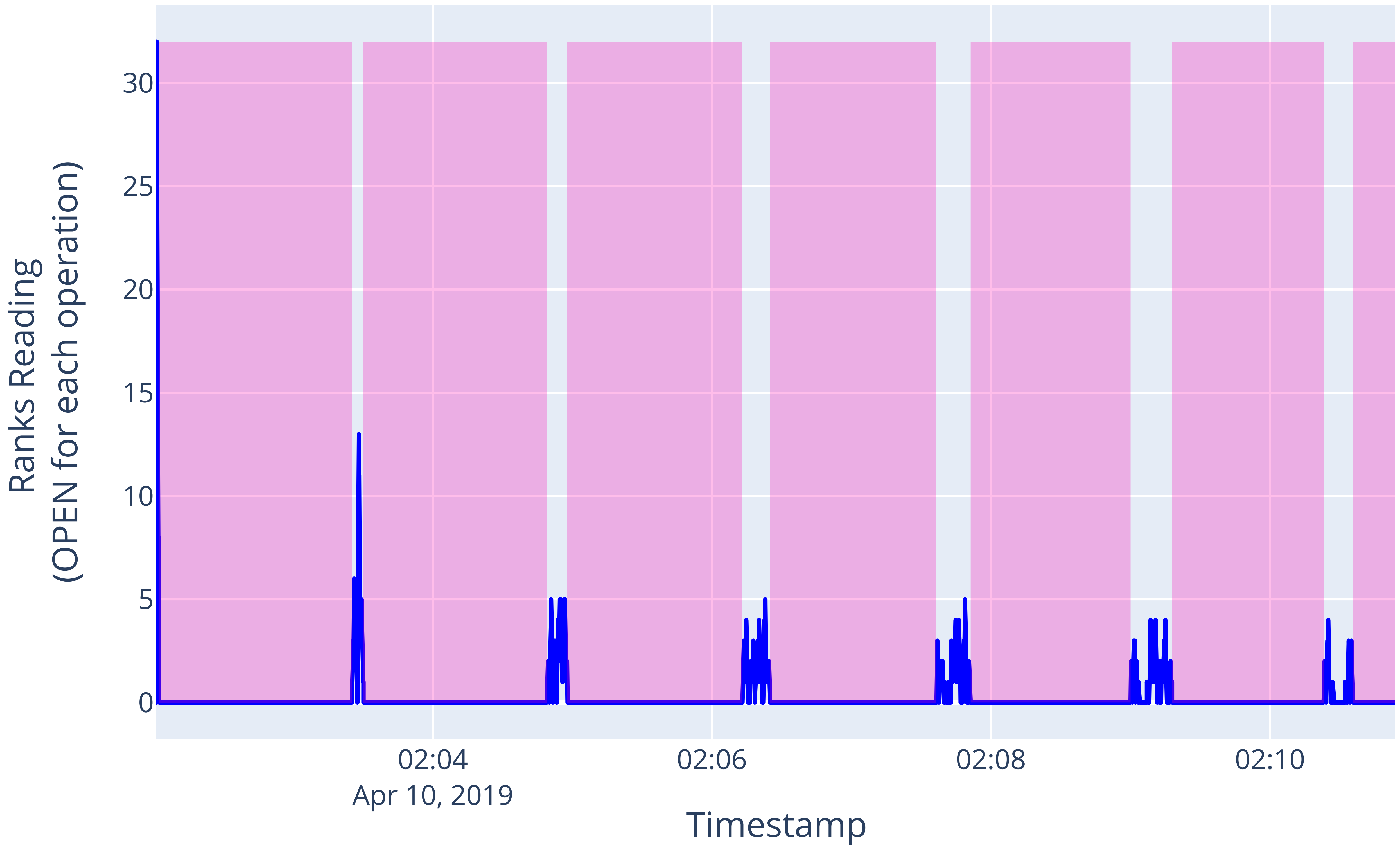
Export: save the results in a JSON file for each processed trace. It contains execution metadata, assigned classes, and list of operations with periodicity.

Merging of neighboring operations



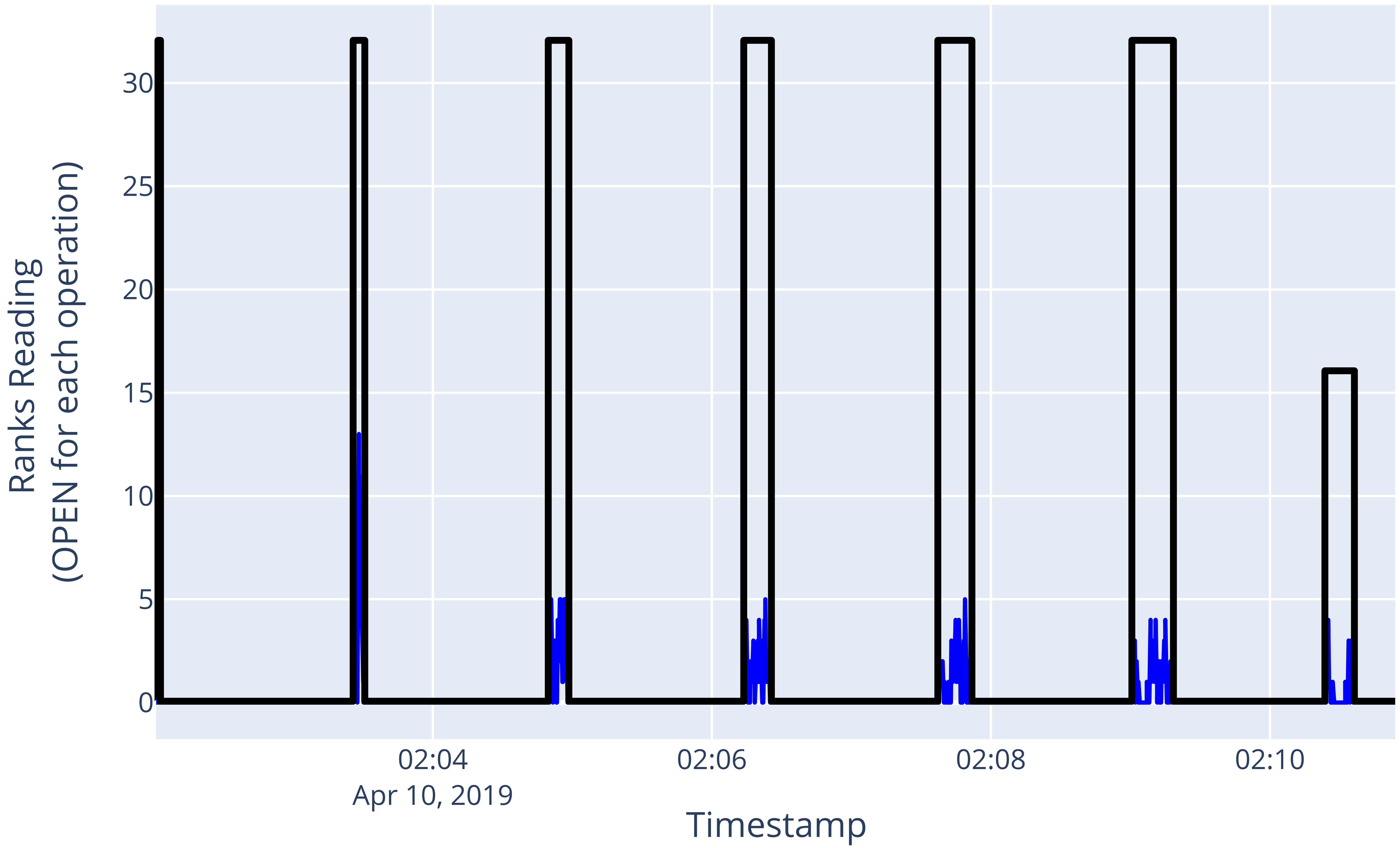
1. Take the list of operations.
2. Find the inactive periods higher than a fraction of the average idle time between two operations.
3. Merge operations between those periods.

Merging of neighboring operations



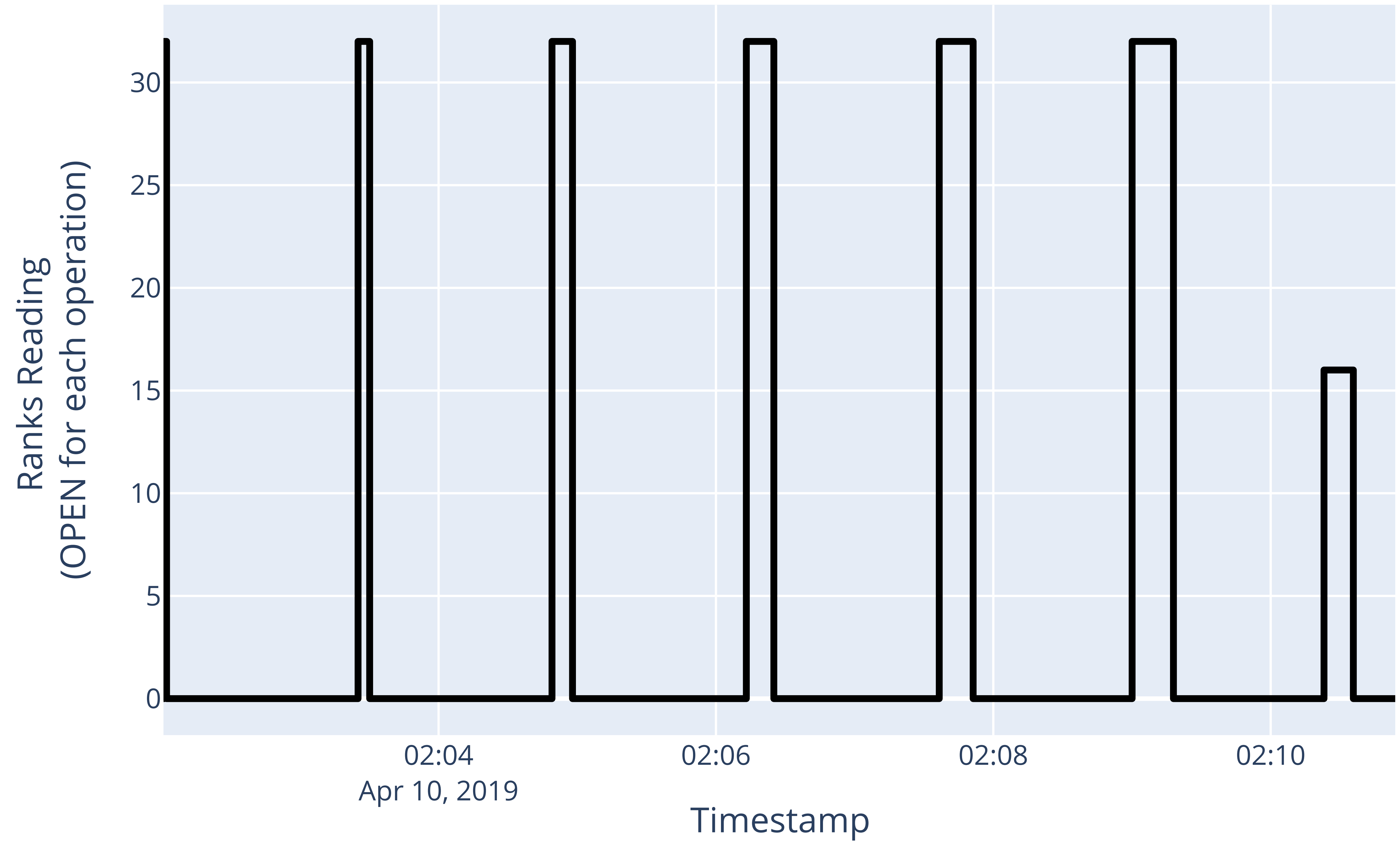
1. Take the list of operations.
2. Find the inactive periods higher than a fraction of the average idle time between two operations.
3. Merge operations between those periods.

Merging of neighboring operations



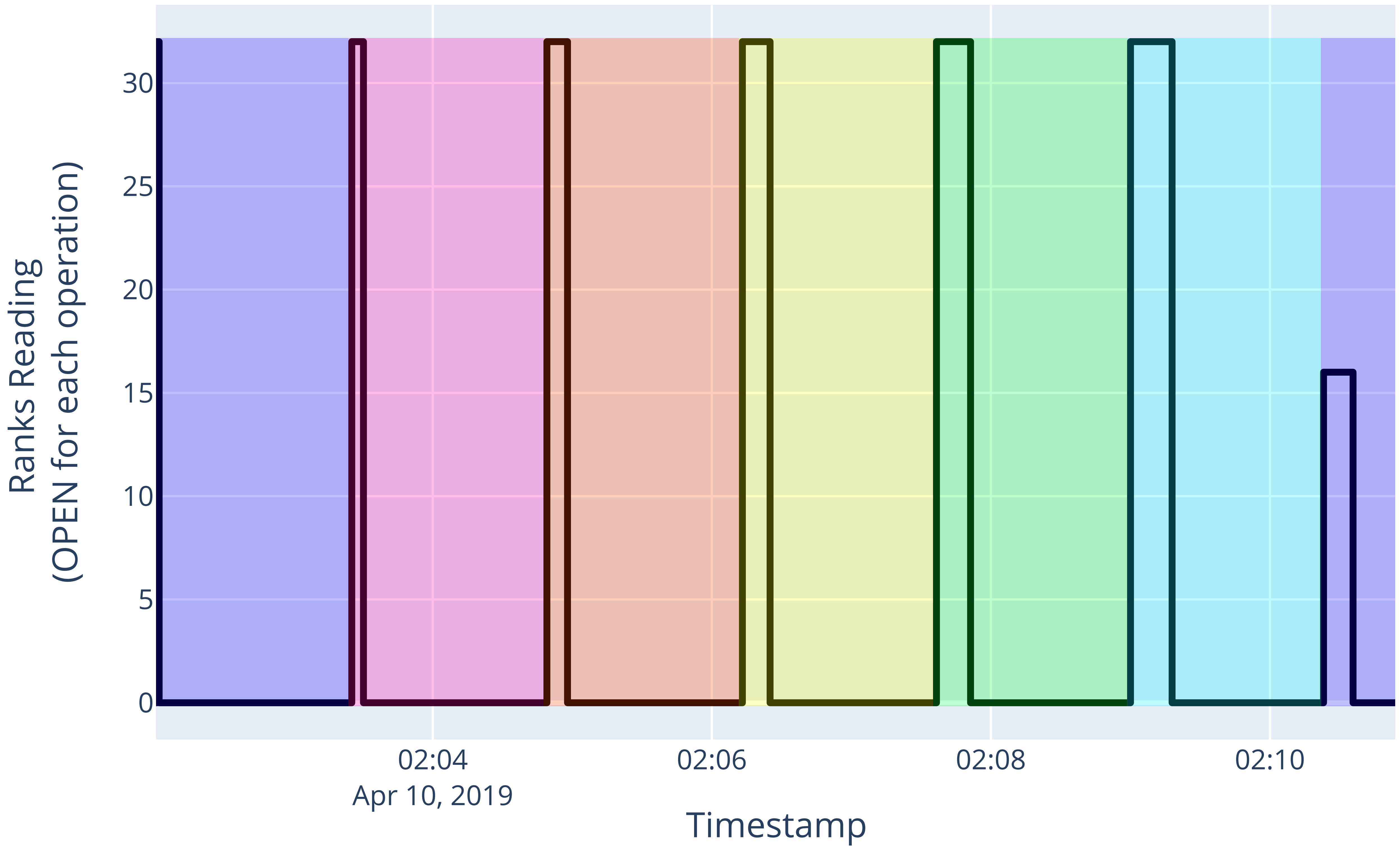
1. Take the list of operations.
2. Find the inactive periods higher than a fraction of the average idle time between two operations.
3. Merge operations between those periods.

Detection of recurring operations



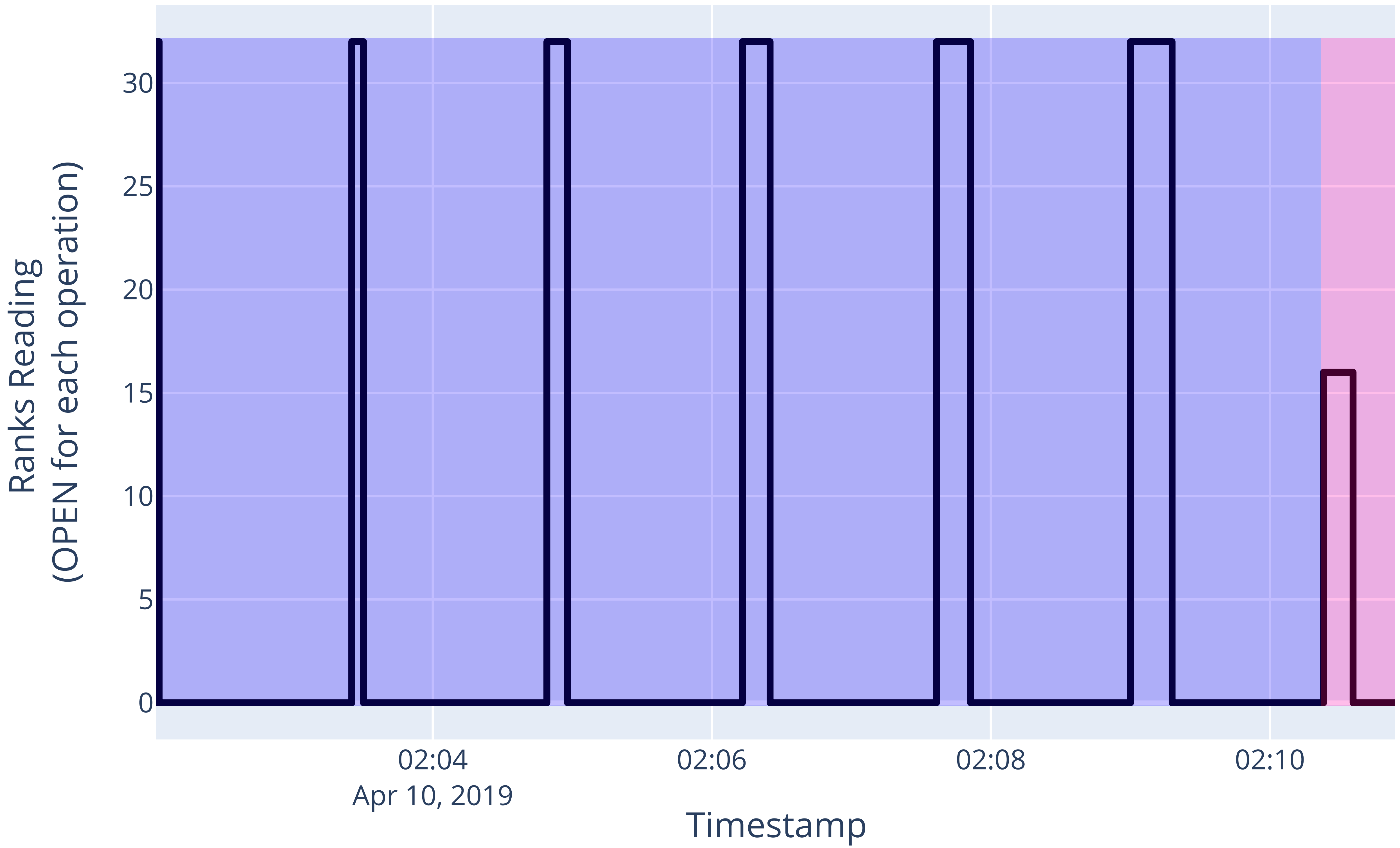
1. Take the list of merged operations.
2. Create segments from the start of a merged operation to the start of the next one.
3. Run MeanShift clustering algorithm to group similar segments. All the segments in the same group are considered as being part of the same periodic operation

Detection of recurring operations



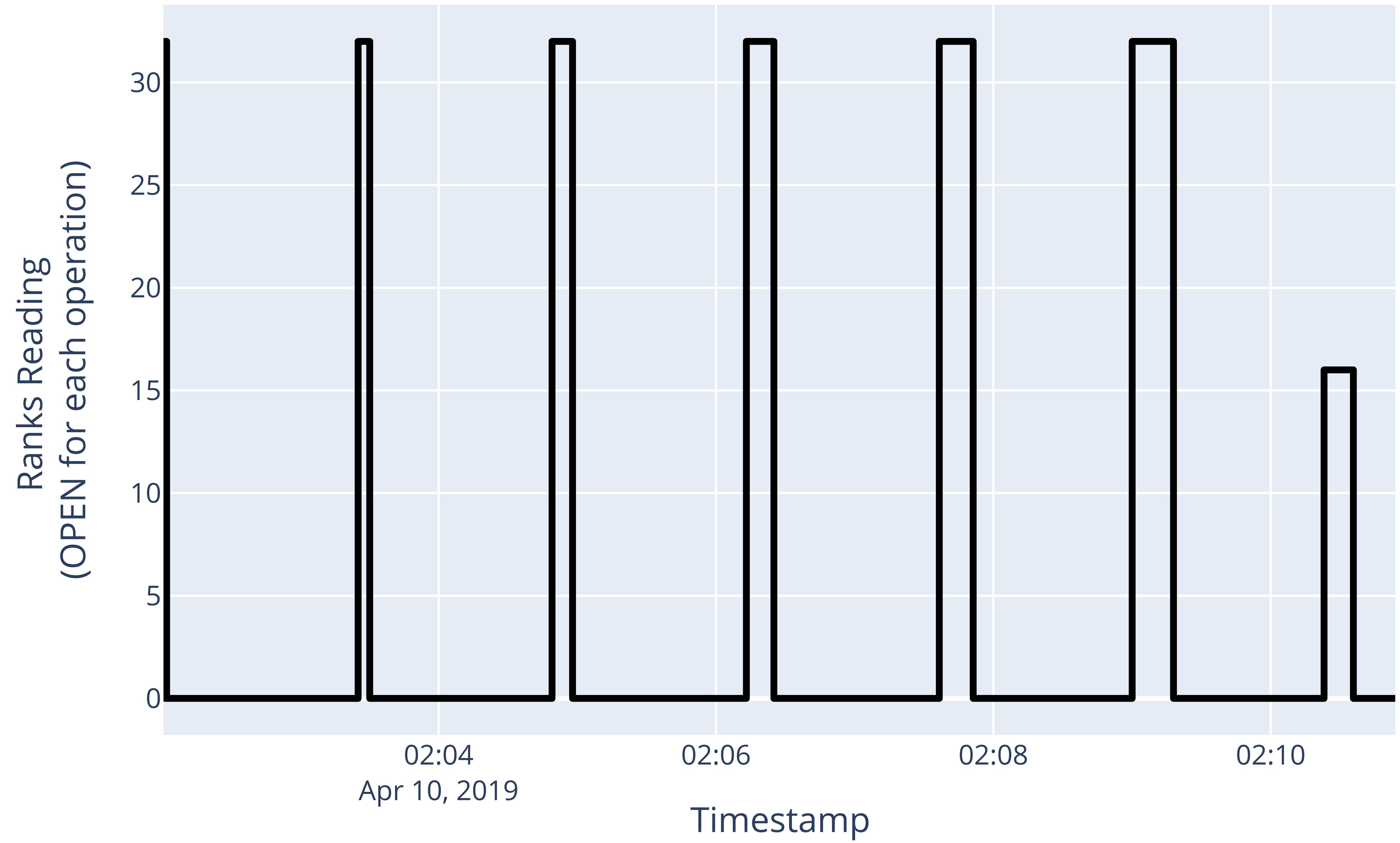
1. Take the list of merged operations.
2. Create segments from the start of a merged operation to the start of the next one.
3. Run MeanShift clustering algorithm to group similar segments. All the segments in the same group are considered as being part of the same periodic operation

Detection of recurring operations



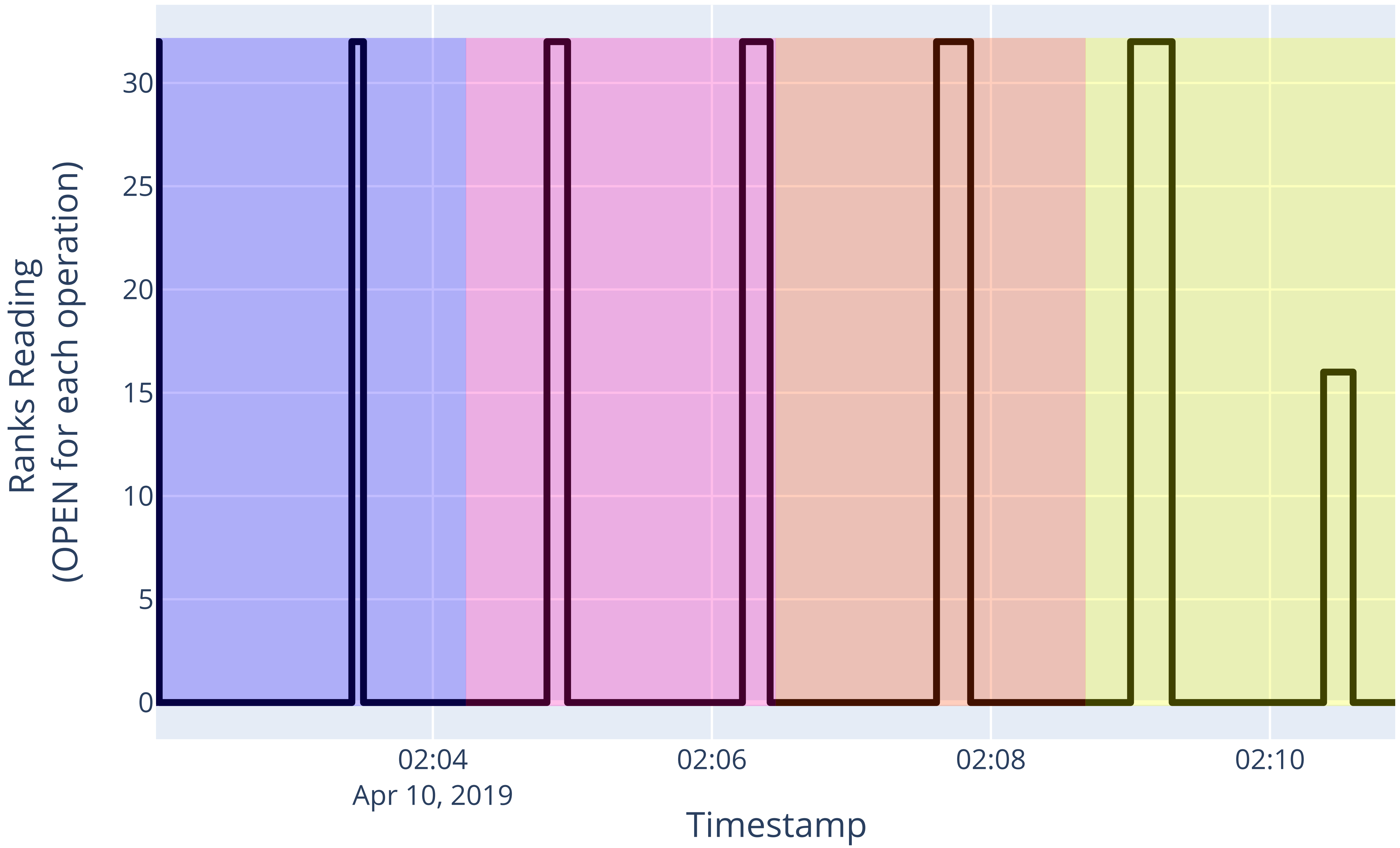
1. Take the list of merged operations.
2. Create segments from the start of a merged operation to the start of the next one.
3. Run MeanShift clustering algorithm to group similar segments. All the segments in the same group are considered as being part of the same periodic operation

Detection of activity's temporality



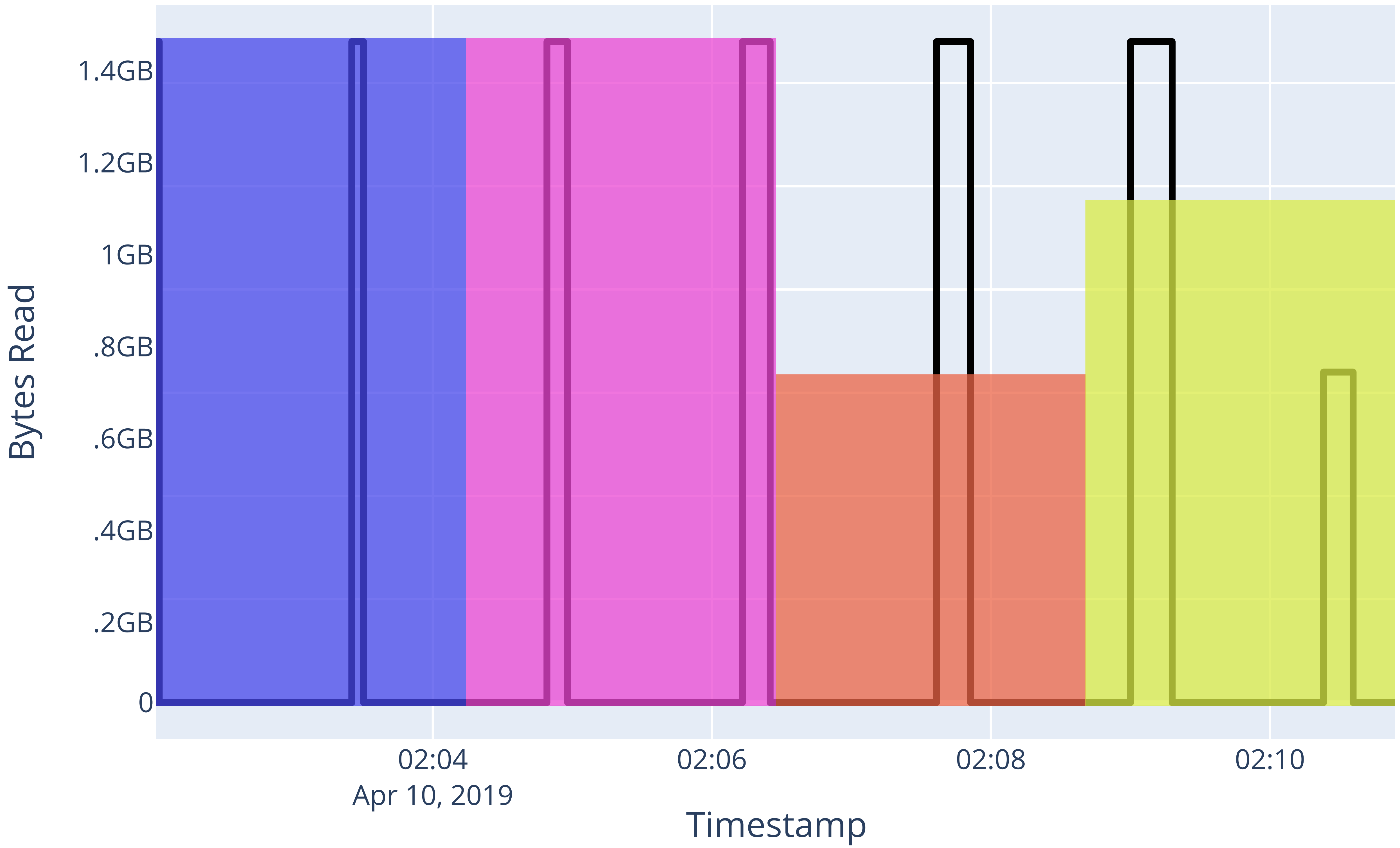
1. Take the list of merged operations.
2. Split the trace into four chunks of equal length.
3. Compute the amount of Bytes operated in each chunk.

Detection of activity's temporality



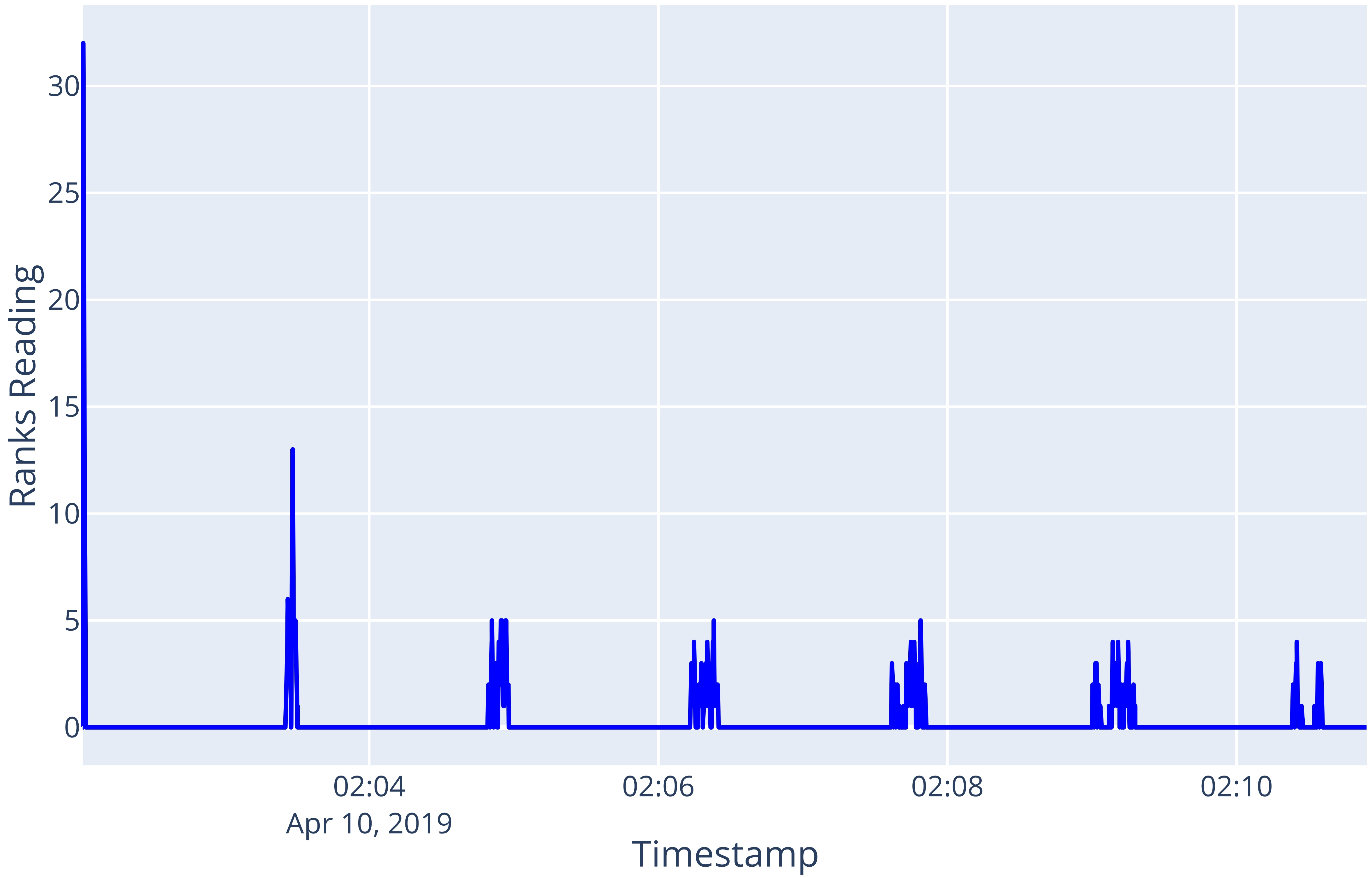
1. Take the list of merged operations.
2. Split the trace into four chunks of equal length.
3. Compute the amount of Bytes operated in each chunk.

Detection of activity's temporality



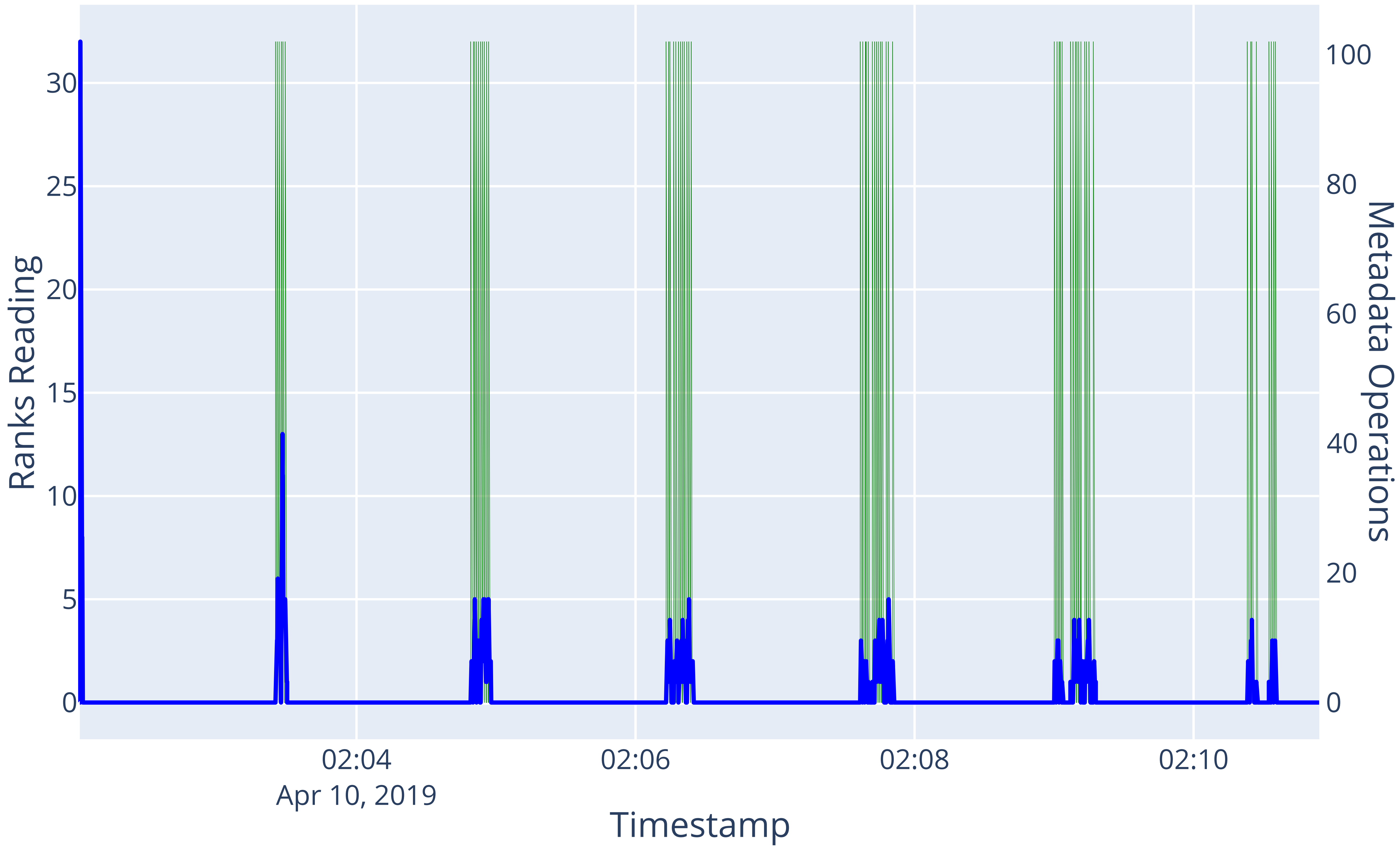
1. Take the list of merged operations.
2. Split the trace into four chunks of equal length.
3. Compute the amount of Bytes operated in each chunk.

Detection of metadata activity



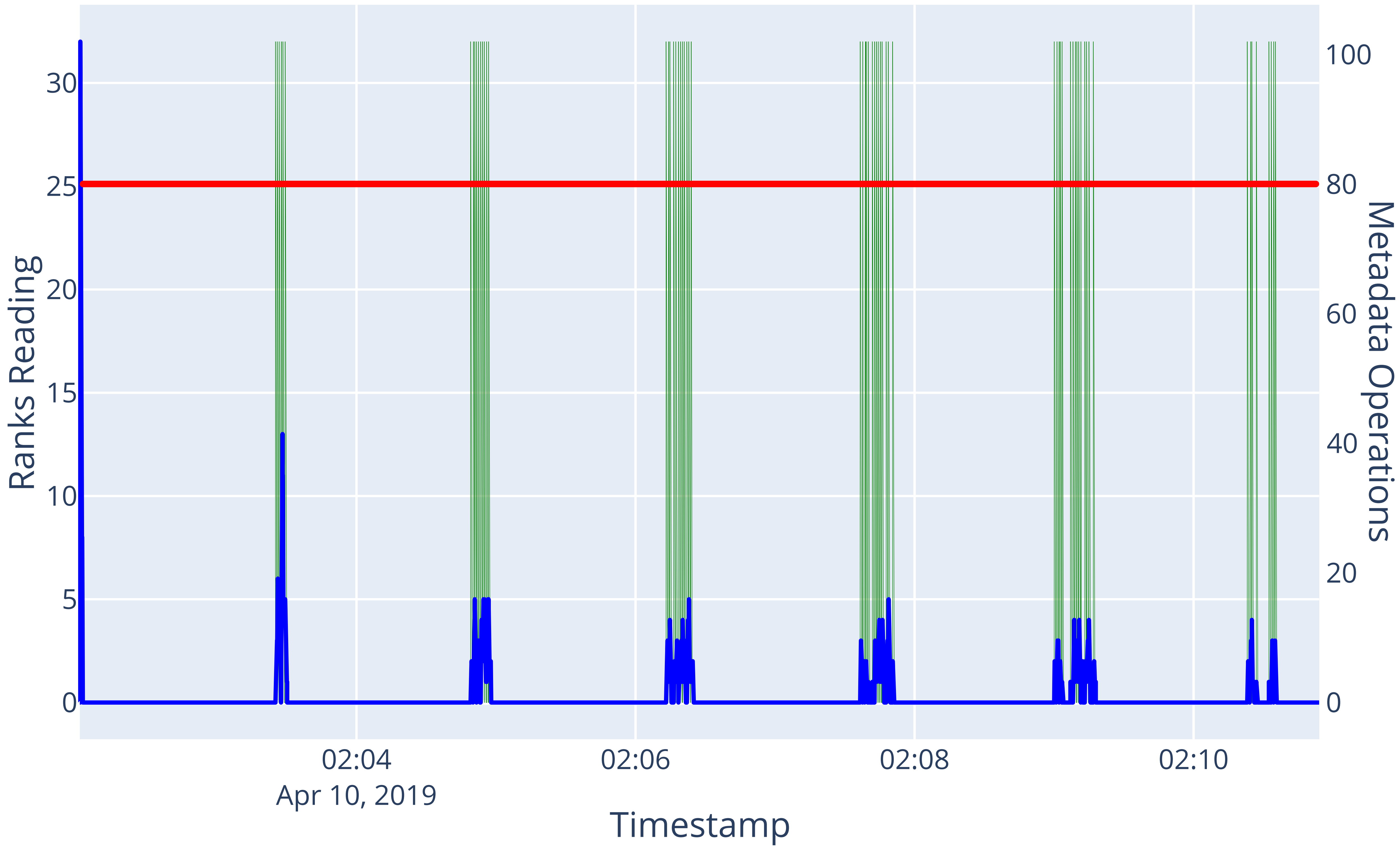
1. Take the list of operations.
2. For each operation, put the OPEN and SEEK operations at the beginning of it, and the CLOSE at the end.
3. Assigns the classes if one or multiple spikes (over a threshold) are present, if multiple operations are done in a window of multiple seconds.

Detection of metadata activity



1. Take the list of operations.
2. For each operation, put the OPEN and SEEK operations at the beginning of it, and the CLOSE at the end.
3. Assigns the classes if one or multiple spikes (over a threshold) are present, if multiple operations are done in a window of multiple seconds.

Detection of metadata activity



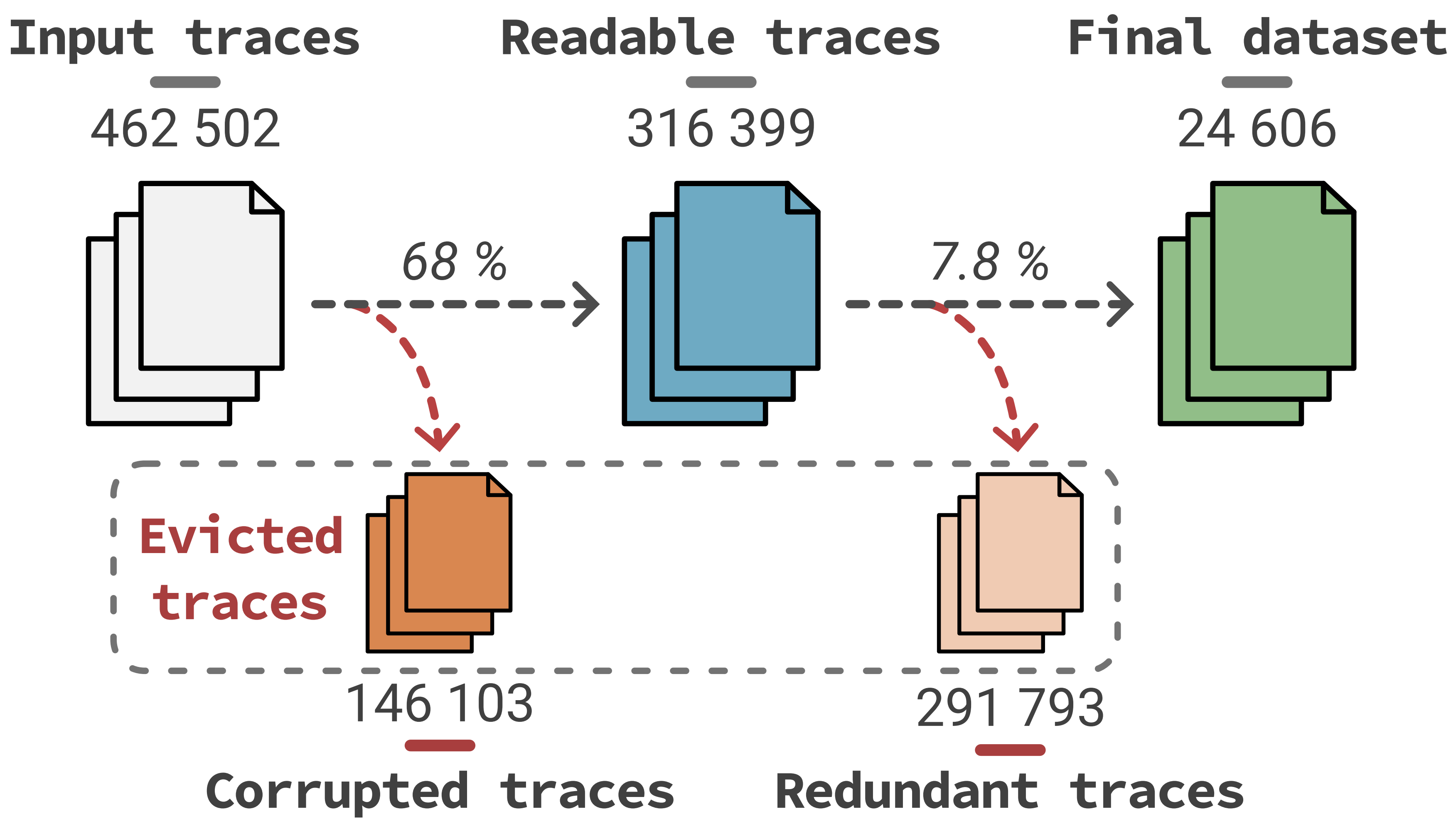
1. Take the list of operations.
2. For each operation, put the OPEN and SEEK operations at the beginning of it, and the CLOSE at the end.
3. Assigns the classes if one or multiple spikes (over a threshold) are present, if multiple operations are done in a window of multiple seconds.

RESULTS FROM BLUE WATERS' TRACES (YEAR 2019)

Blue Waters

- **Petascale supercomputer** managed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois.
  - Operated from 2013 until late 2021.
  - 27k nodes, 49k CPUs
  - **25PB of storage** managed by Lustre
- By default, all jobs running on Blue Waters were **monitored by Darshan**. Traces from 2013 to 2019 are publicly available.
- The traces contain the full list of operations, meaning one can estimate the I/O activity produced by monitored jobs.

Trace Processing flow (Blue Water, year 2019)

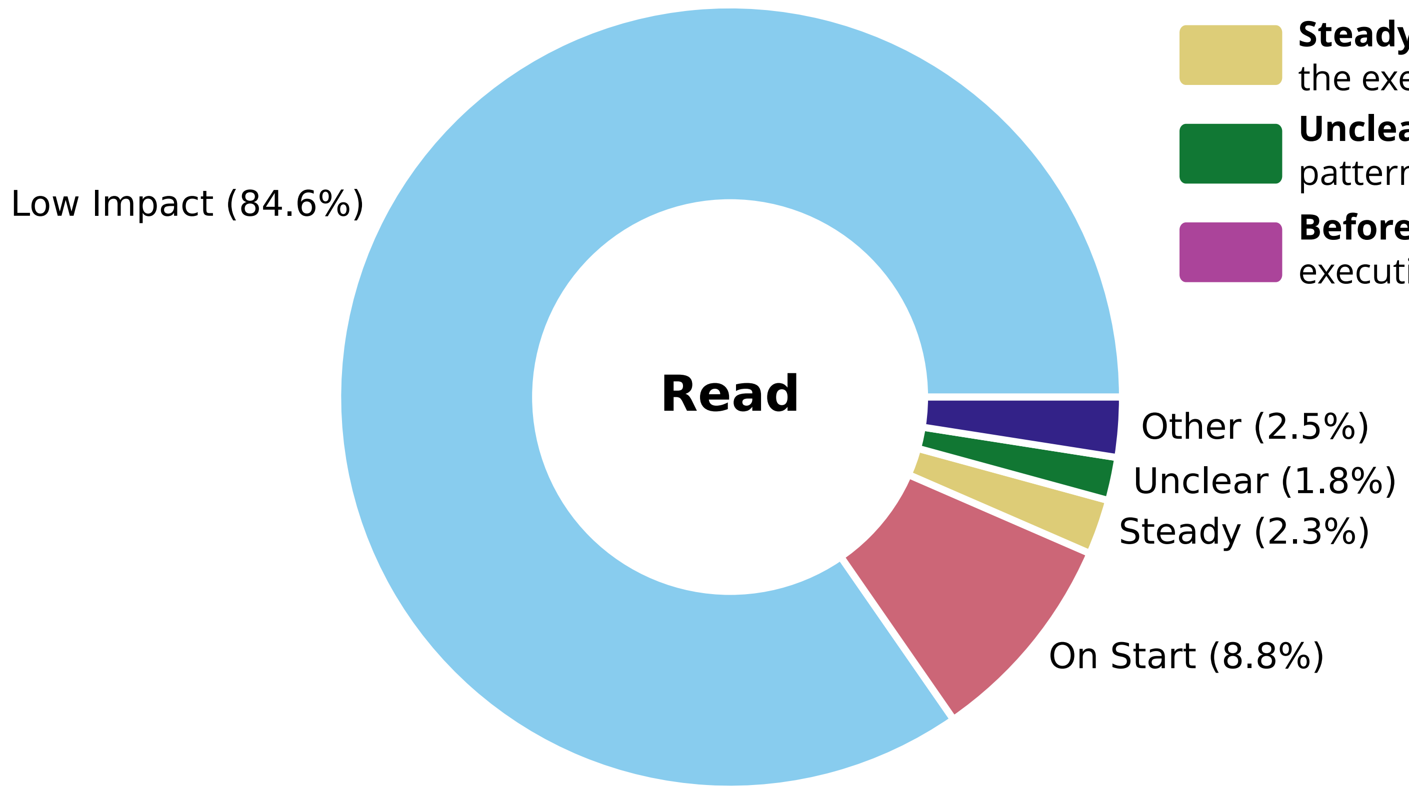


Temporality classes distribution (read)

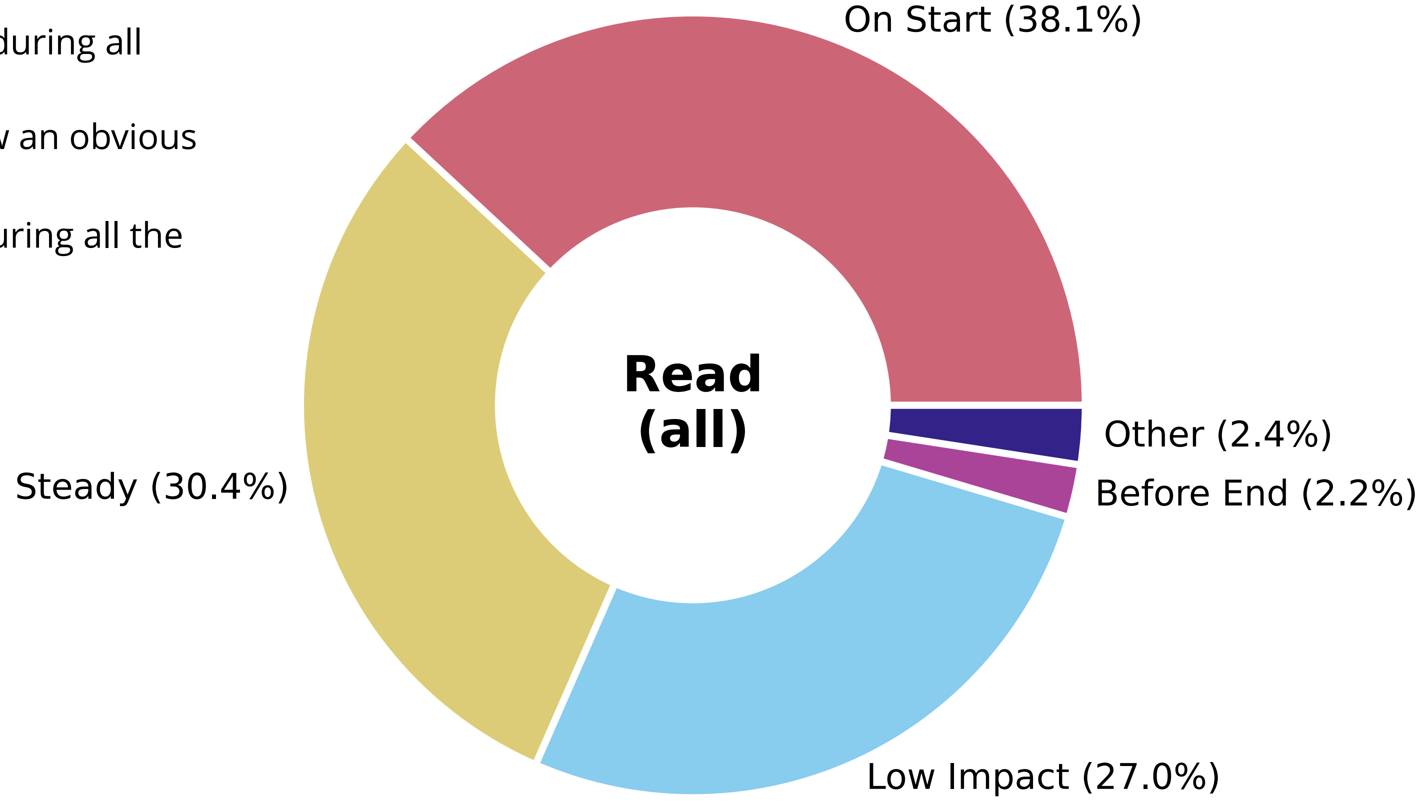
Read: distribution of categorized representative traces

Read (all): estimation of the repartition of classes for all the traces in the dataset (weighting of each representative trace by the number of files it represents)

- **Low Impact**: less than 100MiB read
- **On Start**: most read operations are performed at the execution's start (first 25%)
- **Steady**: read operations are performed during all the execution
- **Unclear**: read operations does not follow an obvious pattern
- **Before End**: read operations are done during all the execution except for the end (last 25%)



Most represented read classes for categorized traces.



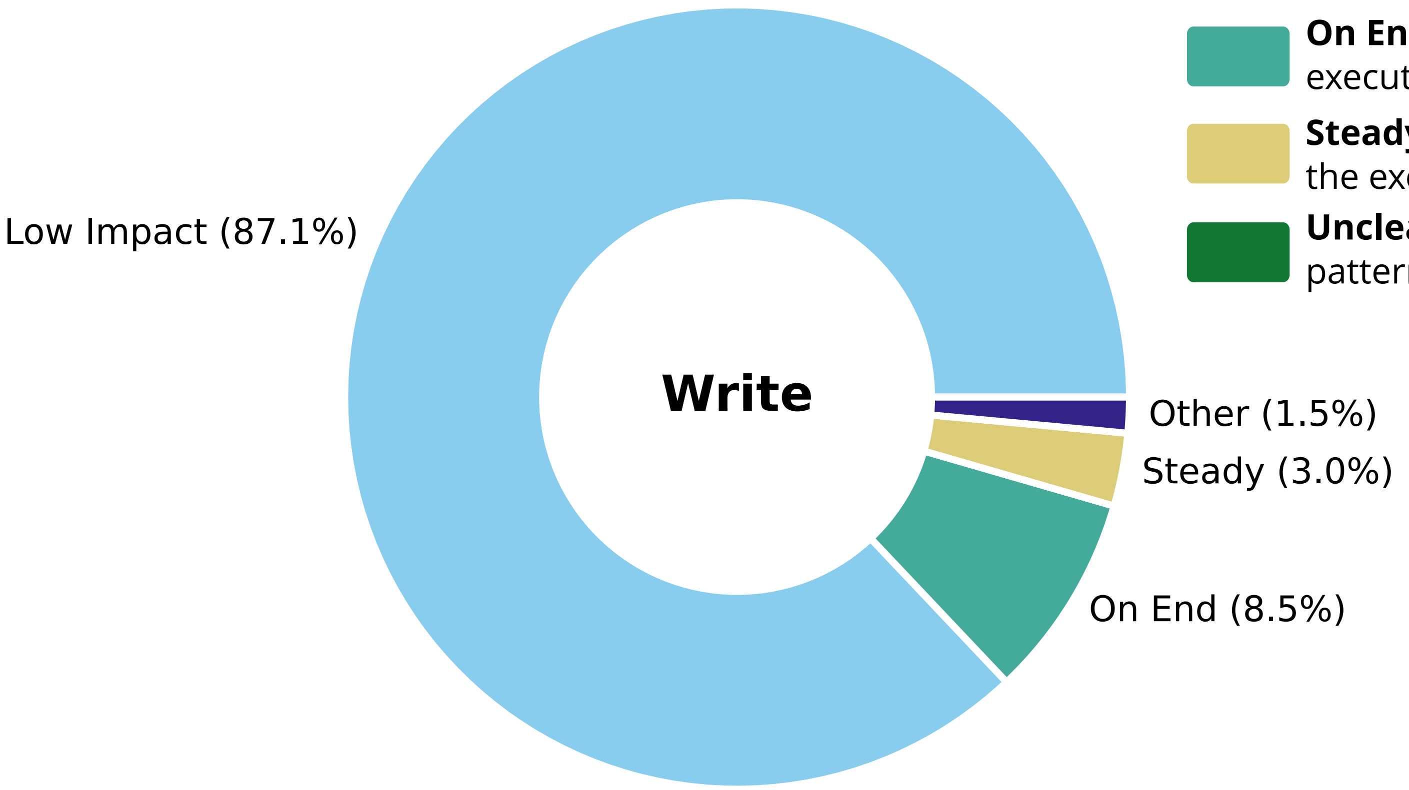
Estimation of the most represented read classes for all traces.

Temporality classes distribution (write)

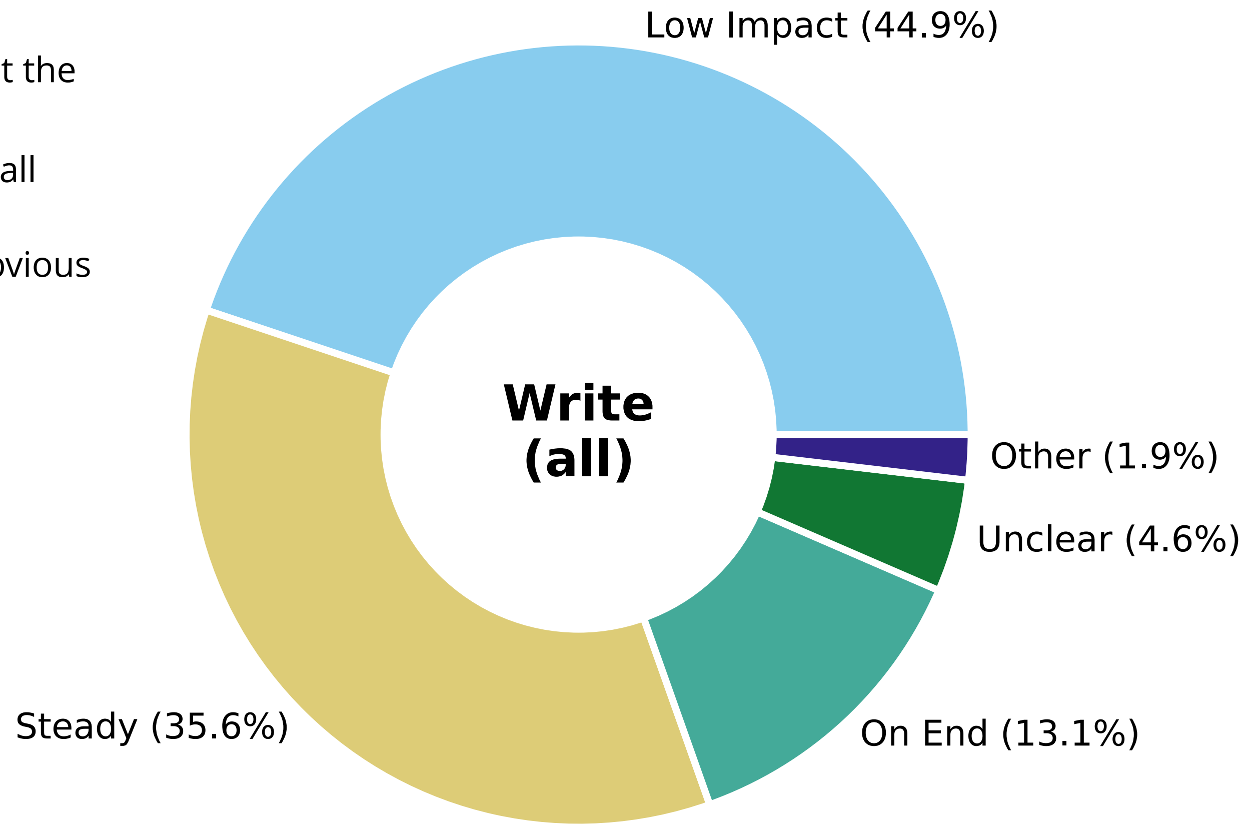
Write: distribution of categorized representative traces

Write (all): estimation of the repartition of classes for all the traces in the dataset (weighting of each representative trace by the number of files it represents)

- **Low Impact**: less than 100MiB written
- **On End**: most write operations are performed at the execution's end (last 25%)
- **Steady**: write operations are performed during all the execution
- **Unclear**: write operations does not follow an obvious pattern



Most represented write classes for categorized traces.



Estimation of the most represented write classes for all traces.

Periodicity classes distribution

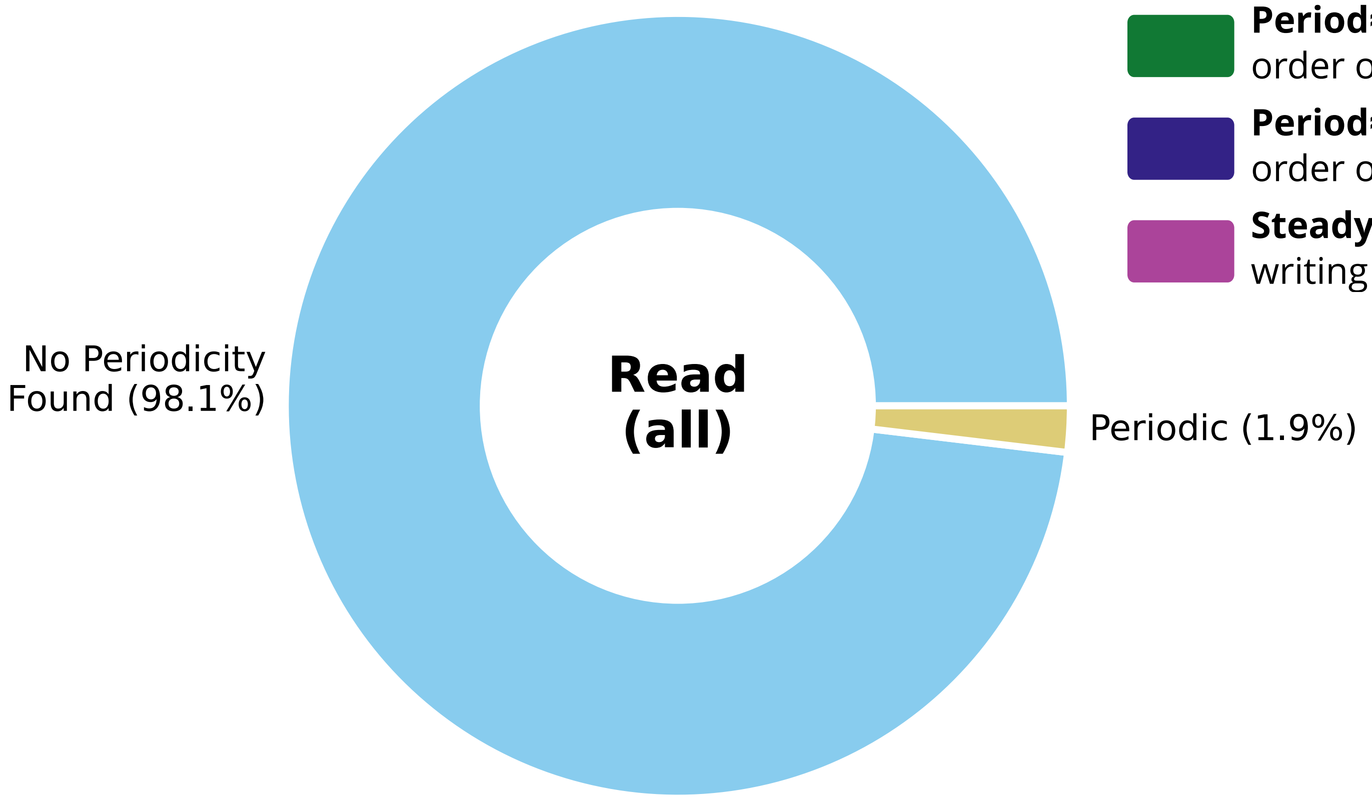
No Periodicity Found: no repeating operation detected

Periodic: at least 3 times more periodic operations (≥2 segments grouped) than non periodic ones

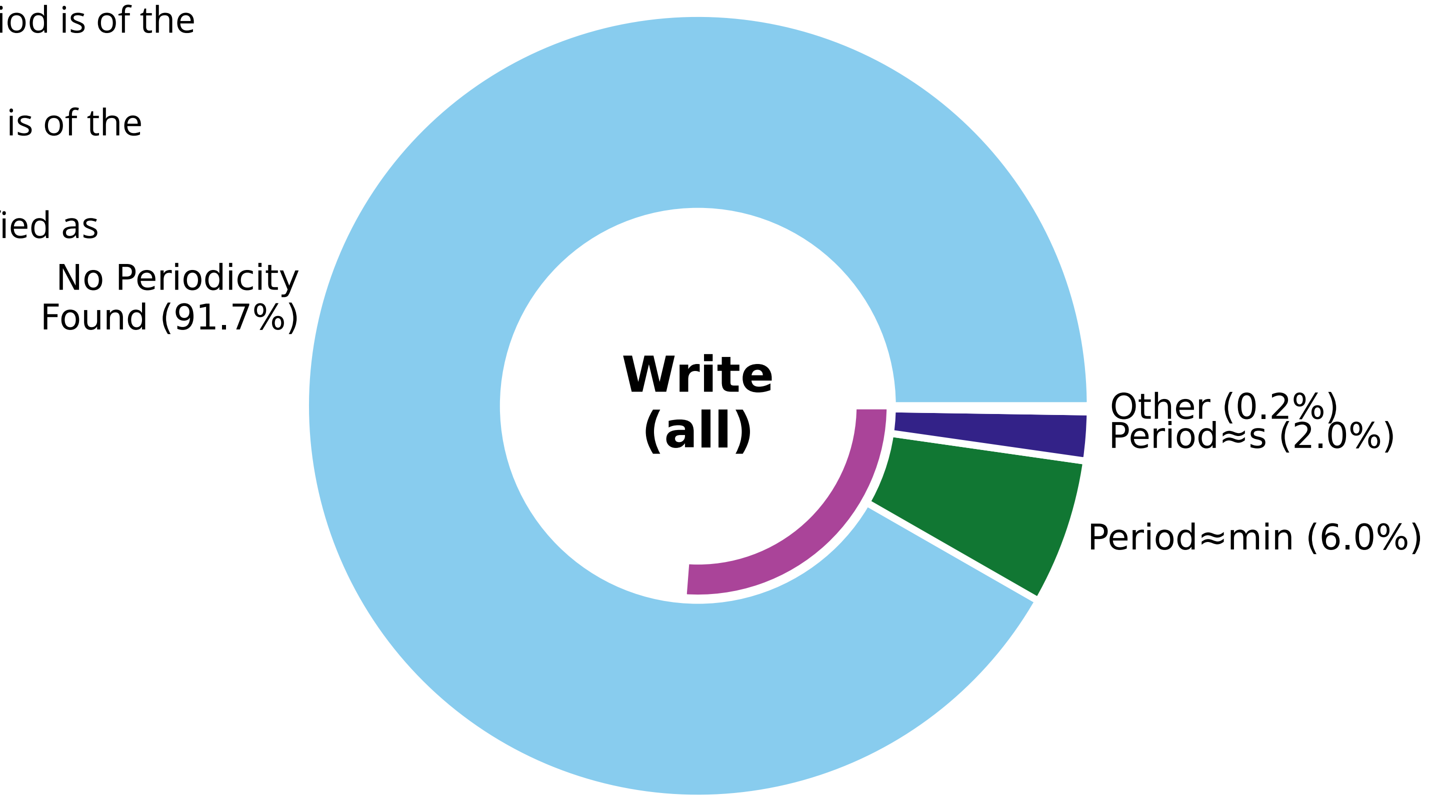
Period≈min: periodic, the average period is of the order of magnitude of the minute

Period≈s: periodic, the average period is of the order of magnitude of the second

Steady Write: amount of traces classified as writing during all the execution

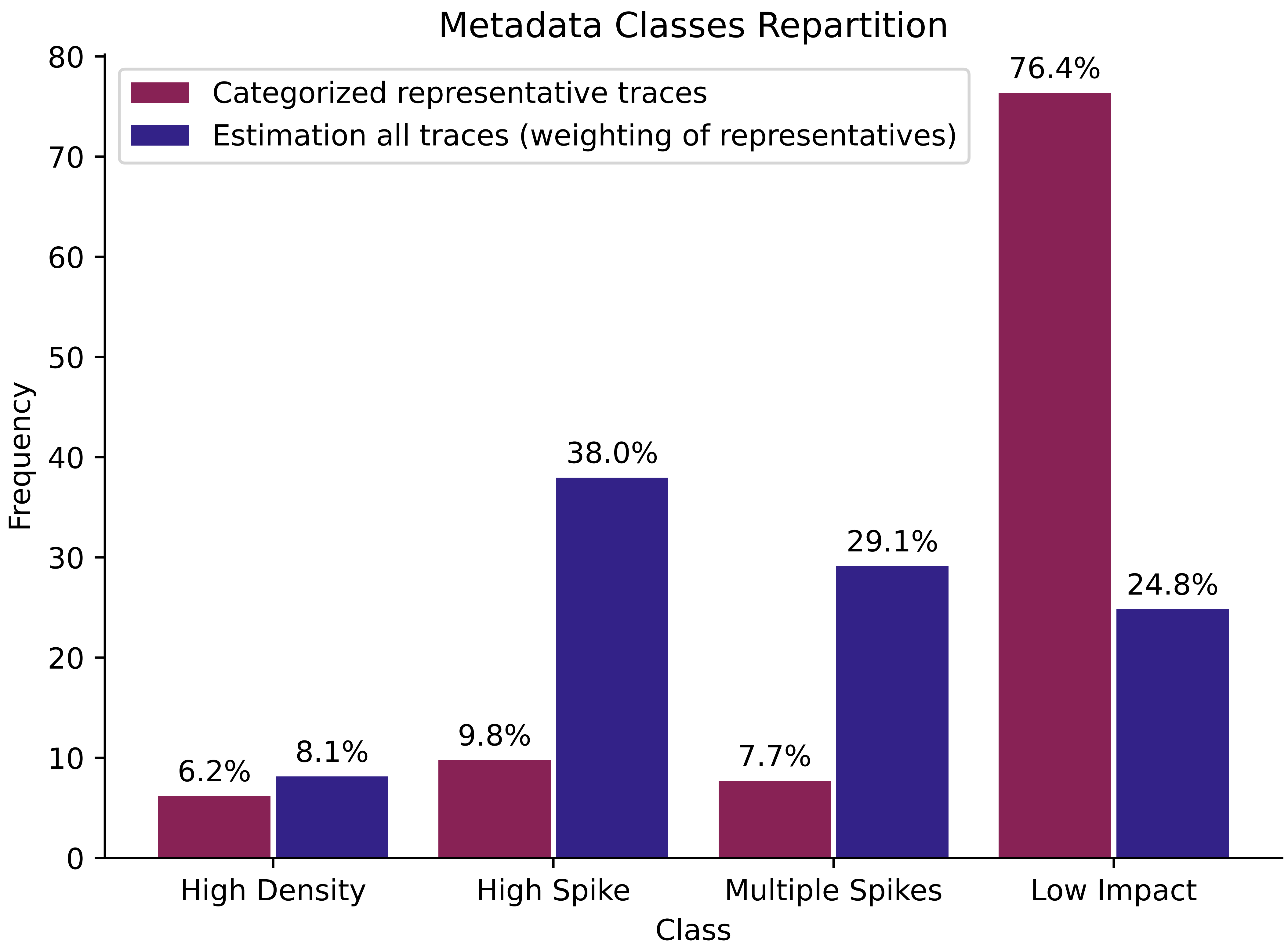


Estimation of the executions with periodic read operations.



Estimation of the most represented periodic write classes for all traces.

Metadata classes distribution



Classes frequently assigned together

1-metadata_high_density
2-metadata_high_spike
3-metadata_multiple_spikes
4-read_insignificant
5-read_on_start
6-read_periodic
7-read_periodic_s
8-read_steady
9-write_insignificant
10-write_on_end
11-write_periodic
12-write_periodic_h
13-write_periodic_min
14-write_steady

Some notable associations:

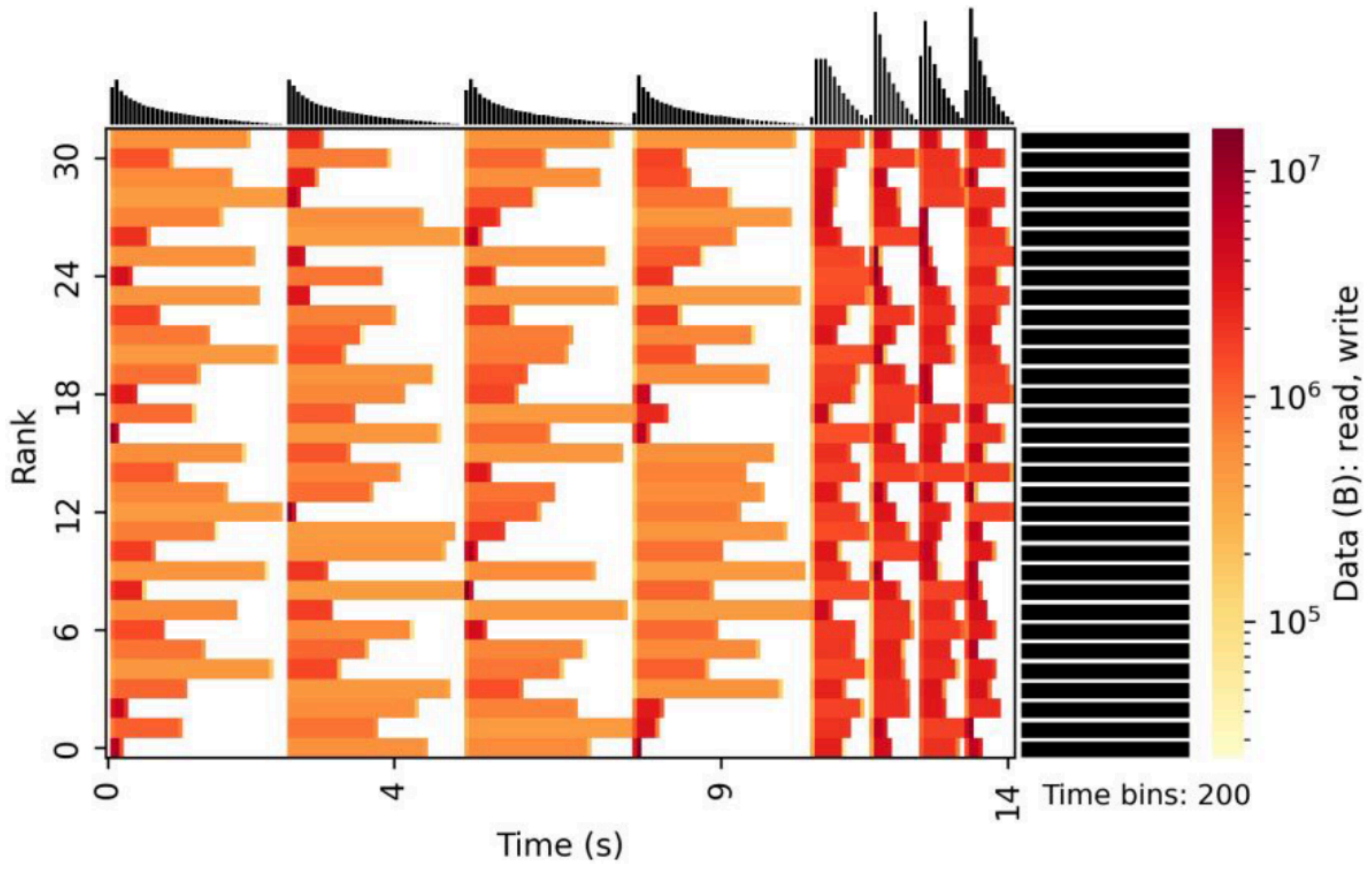
- **95%** of applications having no impactful read activity also has no impactful write activity (4-9).
- **66%** of applications reading at the start also writes at the end (5-10).
- Periodic reads mostly have a period around the second (**62%**, 6-7), whereas periodic writes usually have a period around the minute or longer (**74%**, 11-12 and 11-13).

Conclusion

Contributions:

- **A methodology to detect recurring operations based on clustering**: we have worked on a clustering-based algorithm that groups operations based on their characteristics.
- **A set of classes to describe I/O patterns**: we defined a set of high-level classes characterizing the access patterns found in I/O traces.
- **The implementation of a Python tool to categorize Darshan traces**: we created MOSAIC, a tool that automatically processes Darshan traces to assign classes and get insights about the way HPC applications access the storage system.
- **A case study using year 2019 of the Blue Waters Darshan dataset:** we categorized the traces from 2019 in the Blue Waters dataset and analyzed the distribution of classes to see how the storage system was used by the applications.

Limitations



Example of Darshan heatmaps.
Source: Exascale Computing Project

Main limitations:

- **Operation aggregation**: operations are aggregated in Darshan traces from the first open to the last close of a file by a rank. This loss of accuracy can lead to an underestimation of periodic operations.
- **Recent dataset availability**: there are very few publicly available Darshan datasets for a whole system. We did not find a Darshan dataset recent enough to integrate heatmaps from, which could solve the issue raised by operation aggregation.

Future Work and Perspectives

Future Work:

- **Improve the selection of representative traces**: for some applications the selection of a representative trace is an hard task. We plan on improving the methodology by clustering Darshan traces of a single application based on their size and select one representative trace per group.
- **Implementation of unsupervised classification**: the patterns are classified according to classes defined beforehand. We plan to add an automatic classification step to group traces automatically.
- **Further improve the definition of thresholds**: some thresholds were defined empirically from the Blue Waters dataset. We plan to work on a better way to define them.

Perspective:

- **I/O-optimized scheduling**: feed the I/O classes to a scheduler when submitting a job. The scheduler can then adopt strategies to avoid contention on the PFD (e.g. don't start at the same time 2 jobs that read at start).

THANK YOU FOR YOUR ATTENTION!

Access this presentation

YOU HAVE I/O TRACES? WE WANT THEM!

If you have access to I/O traces (Darshan or others) and can share them, don't hesitate to reach us (theo.jolivel@inria.fr) so we discuss how we can work together!