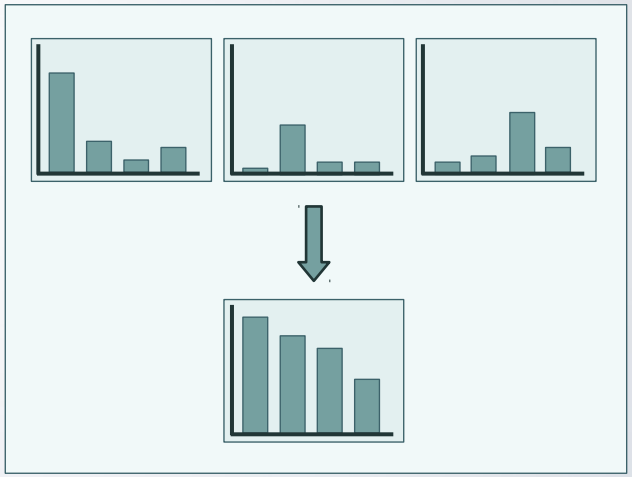
Exploring data merging methods for a distributed processing system

01.12.2021 Piotr Konopka, Barthélémy von Haller on behalf of the ALICE collaboration, CERN

Introduction



Introduction

Mergers!

ALICE

Gatherers!

ATLAS EXPERIMENT

DQM Consumer!*

* a part of it

CMS

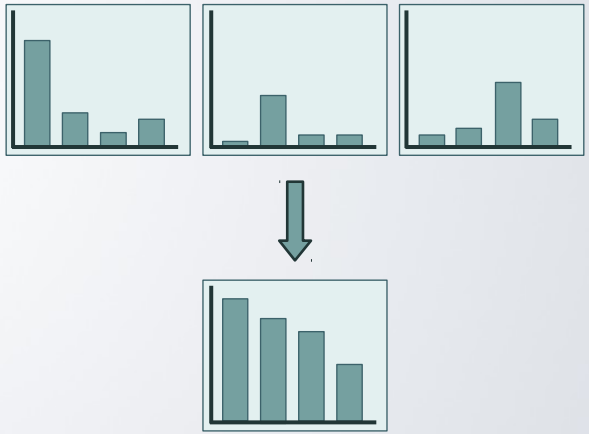
Histogram Adders!

LHCb

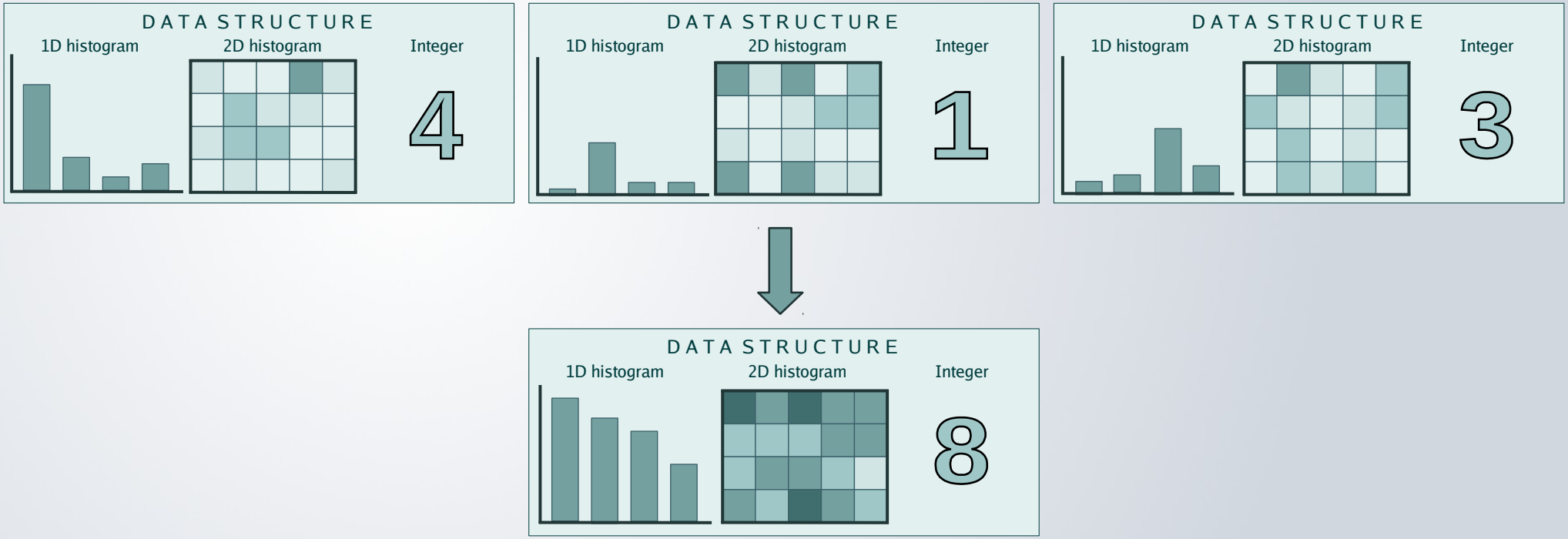
Reduce()!

a data scientist

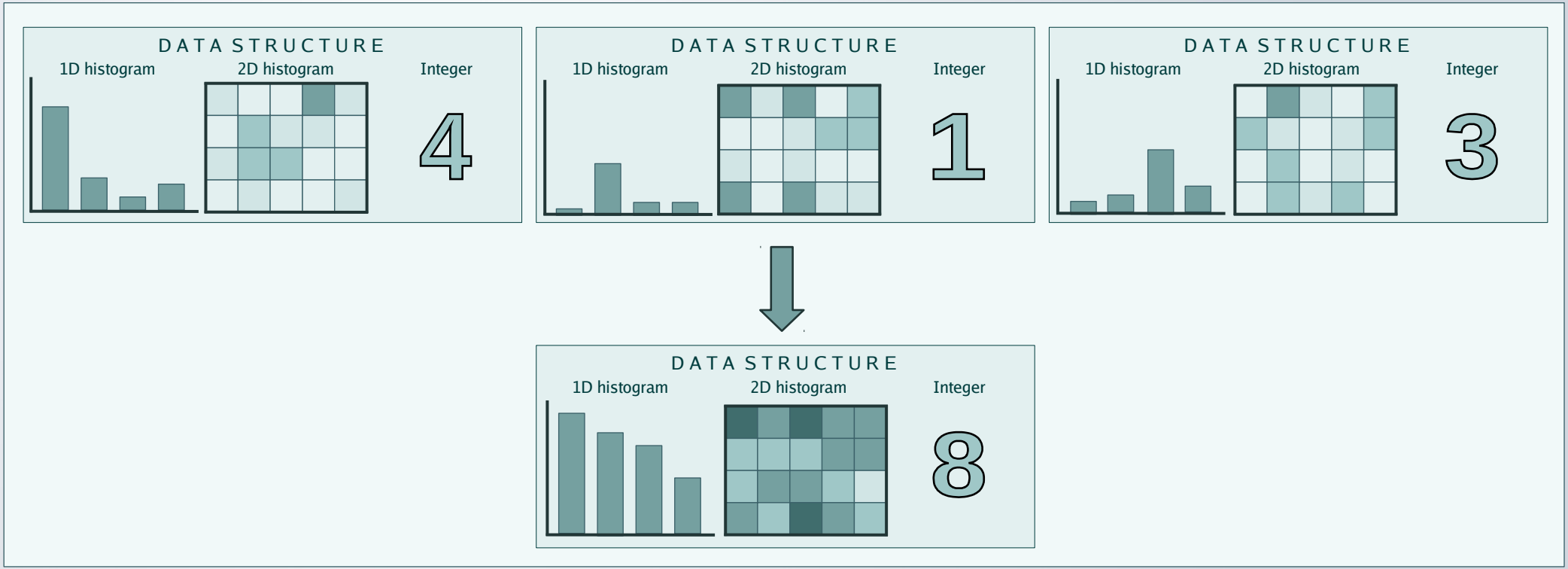
Isn’t it quite straightforward?



That is still quite straightforward though...

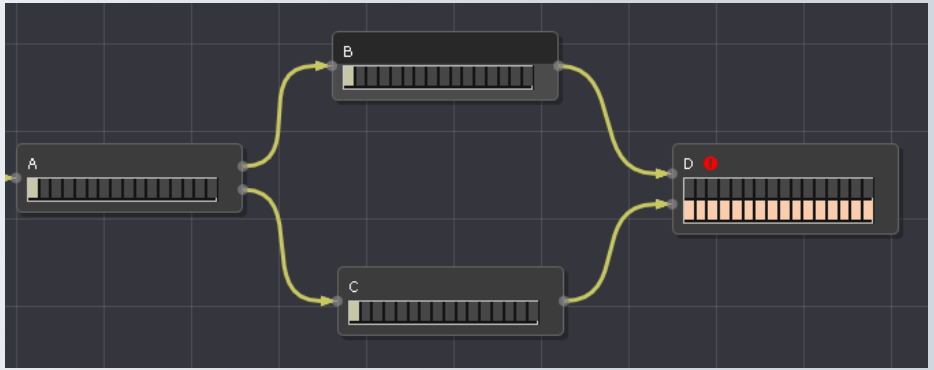


...but putting it in an application is not!



The new computing system in ALICE

- The Online–Offline (O^2) computing system:
  - Around 500 computing nodes
  - Asynchronous message–passing system
  - O(10000) processes in total
- Main use–case of Mergers:
  - Combine incomplete histograms created on parallel processing nodes (e.g. concerning different parts of a detector) in the Quality Control framework



The ALICE processing framework's debugging GUI

Investigation paths

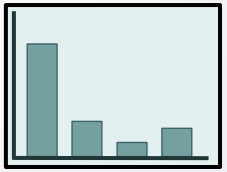
- Merge entire objects or deltas
- Moving windows
- Load balancing
- Optimal number of Merger layers
- Merging collections of objects as a performance improvement
- Preventing memory leaks

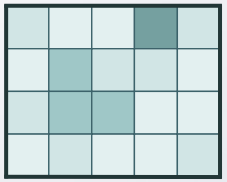
Types of objects being merged

Object size after merging

Shrinking

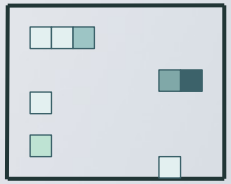
Fixed–size





Growing

A	B	C	D
432	z	0.1	true
432	x	0.2	false
543	c	0.4	false
54	v	0.5	false

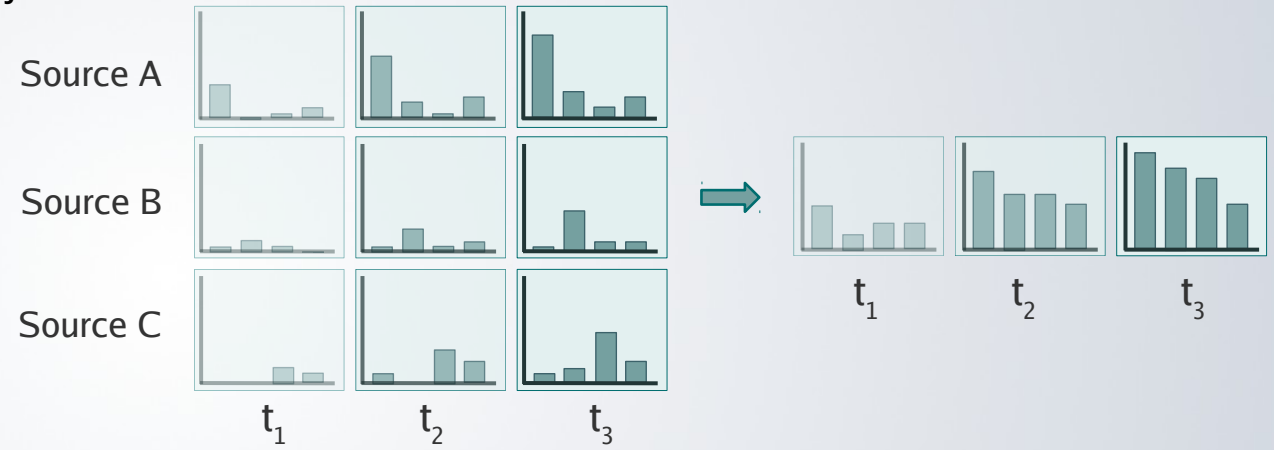


*Sparse histograms
(zero bins are not allocated)

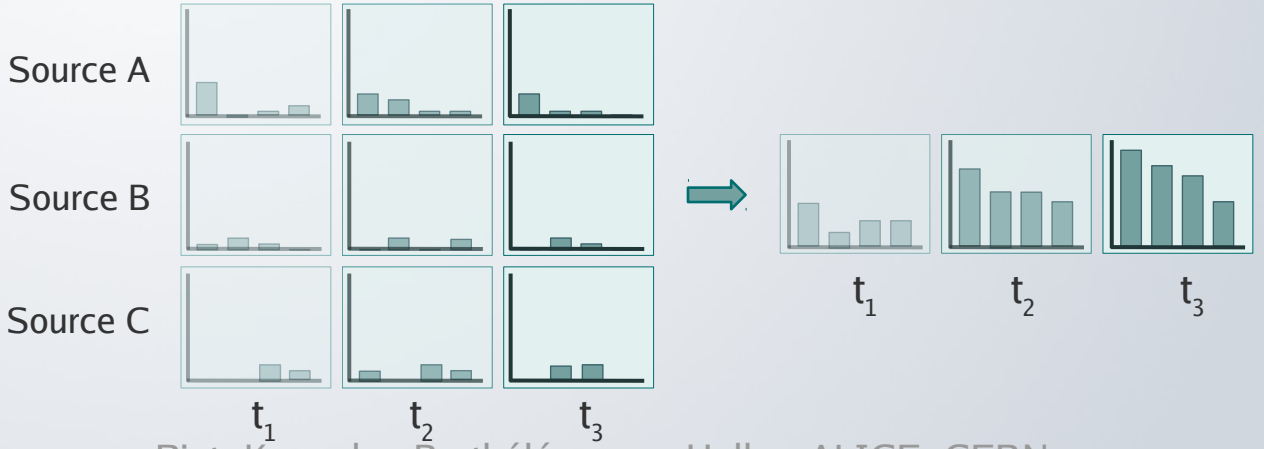
Variable

Entire objects or deltas?

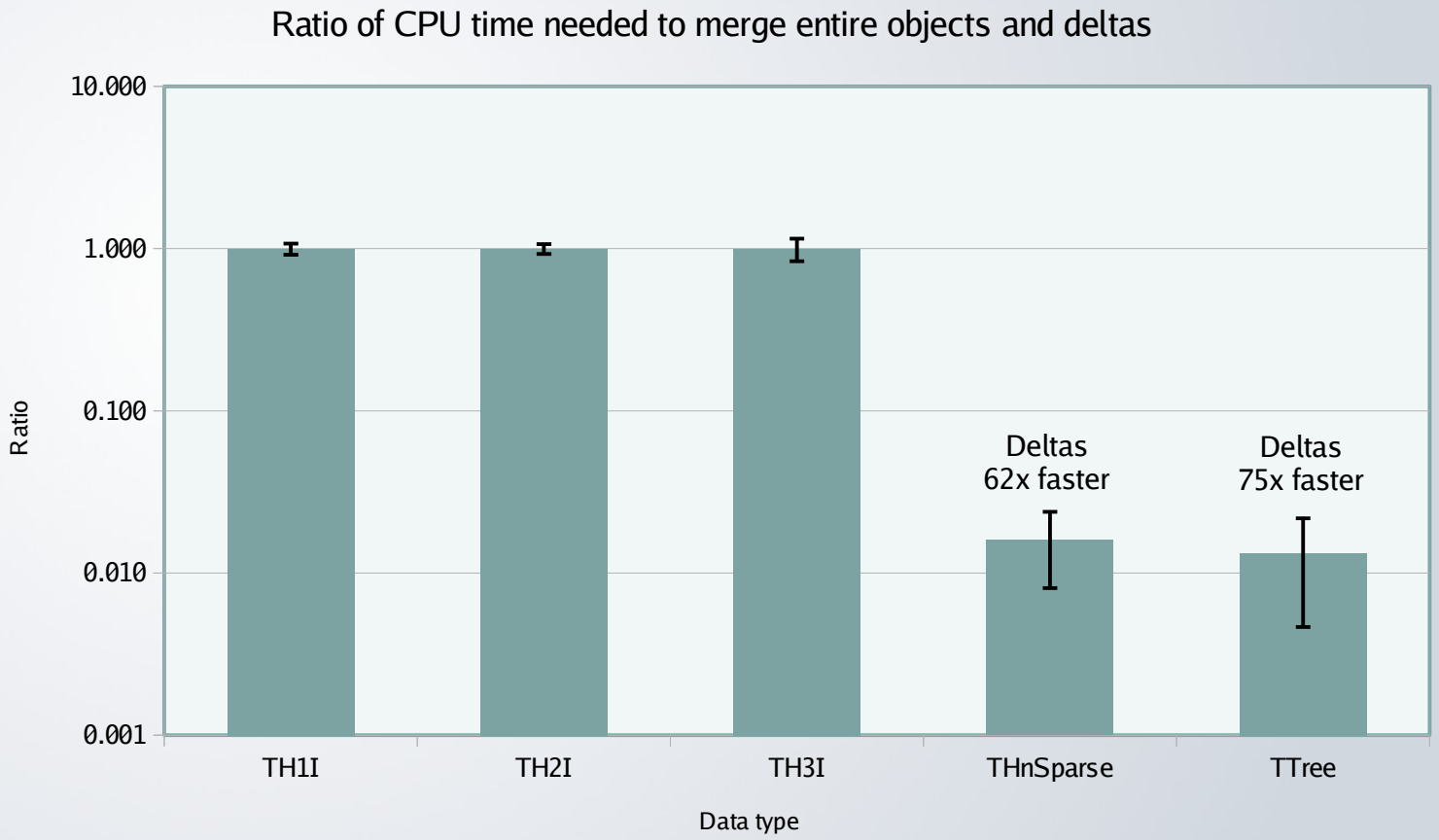
Merging entire objects



Merging deltas



Merging performance for entire objects and 1/100 deltas

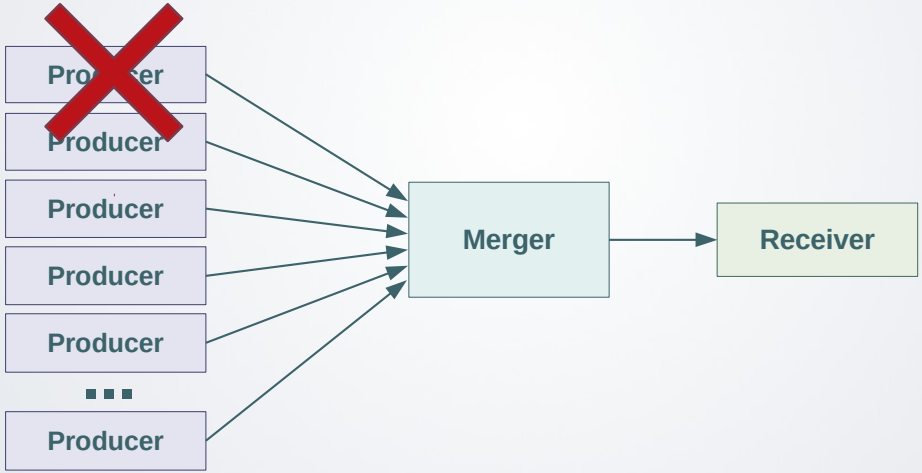


Entire objects or deltas - comparison

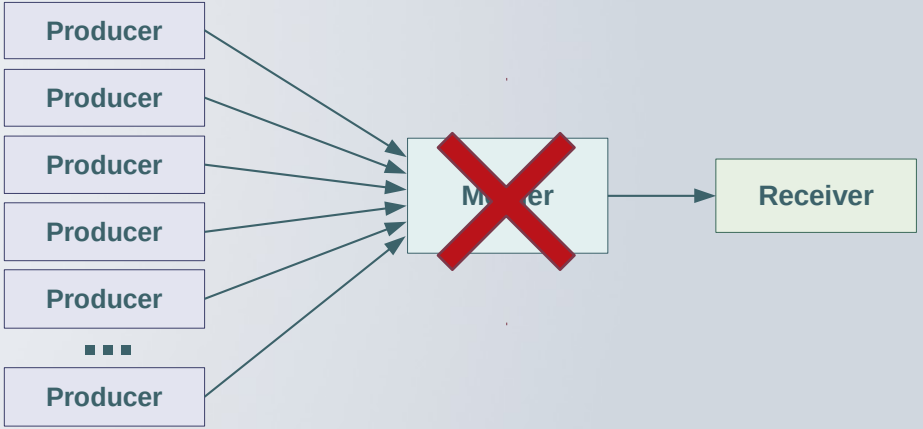
Property	Entire objects	Deltas
Supports objects which cannot be reset	yes	no
Lower resource usage for growing objects	no	yes

Entire objects or deltas - recoverability

One of the sources dies:



Merger dies:

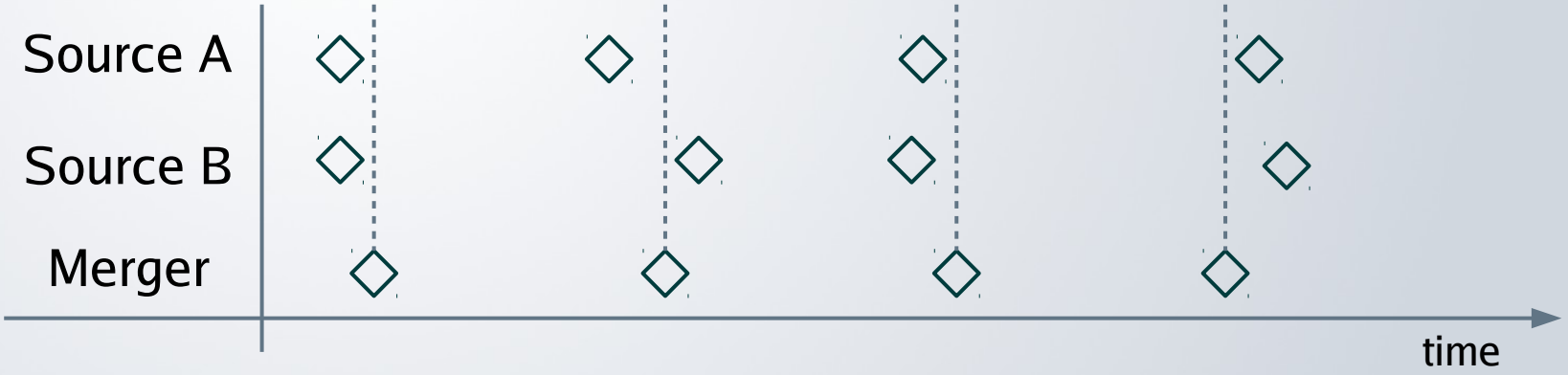


Entire objects or deltas - comparison

Property	Entire objects	Deltas
Supports objects which cannot be reset	yes	no
Lower resource usage for growing objects	no	yes
Recoverability of Merger by design	yes	no
Recoverability of data sources by design	no	yes

Entire objects or deltas - Asynchronicity

- No guarantee that a message will arrive on time
- Especially cumbersome for merging entire objects:
  - Merging should be performed just before publication
  - The last version of each incomplete object should be cached

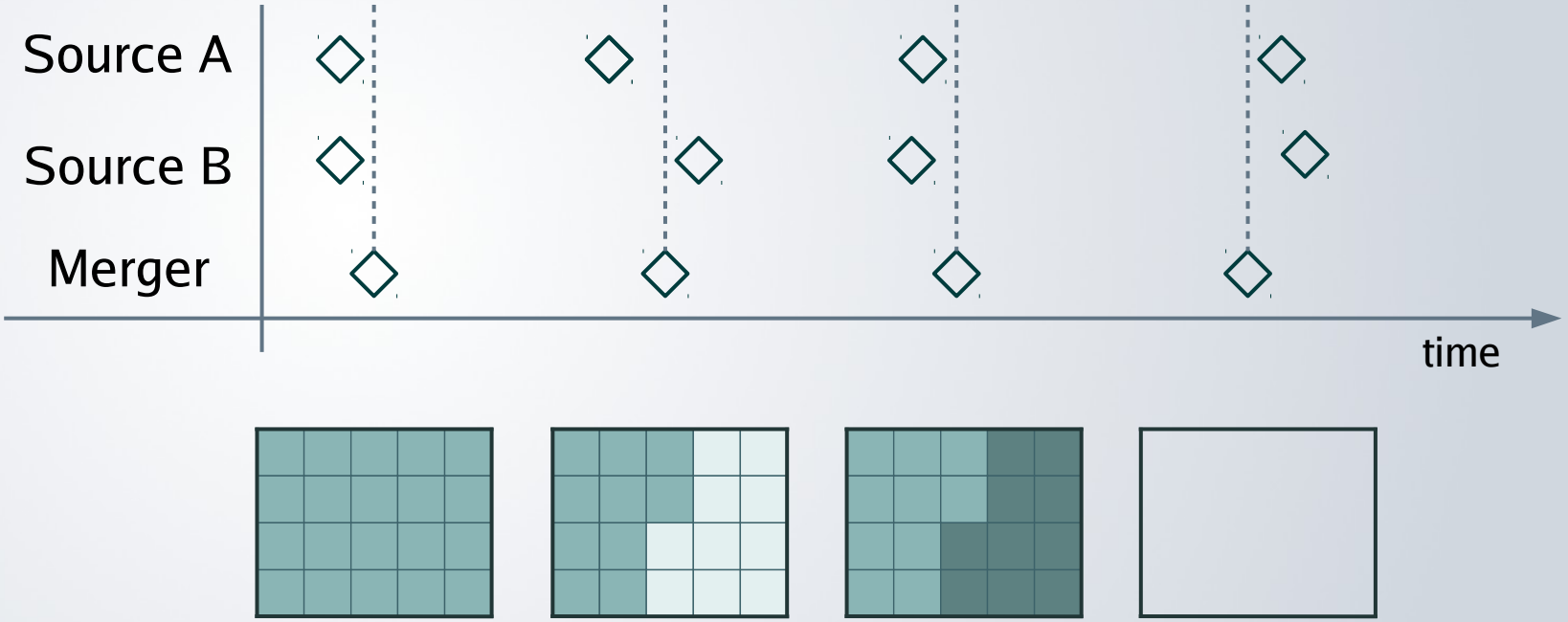


Entire objects or deltas - comparison

Property	Entire objects	Deltas
Supports objects which cannot be reset	yes	no
Lower resource usage for growing objects	no	yes
Recoverability of Merger by design	yes	no
Recoverability of data sources by design	no	yes
No need for distinguishable data sources	no	yes
No need for cache	no	yes
Merging can be performed at object arrival	no	yes

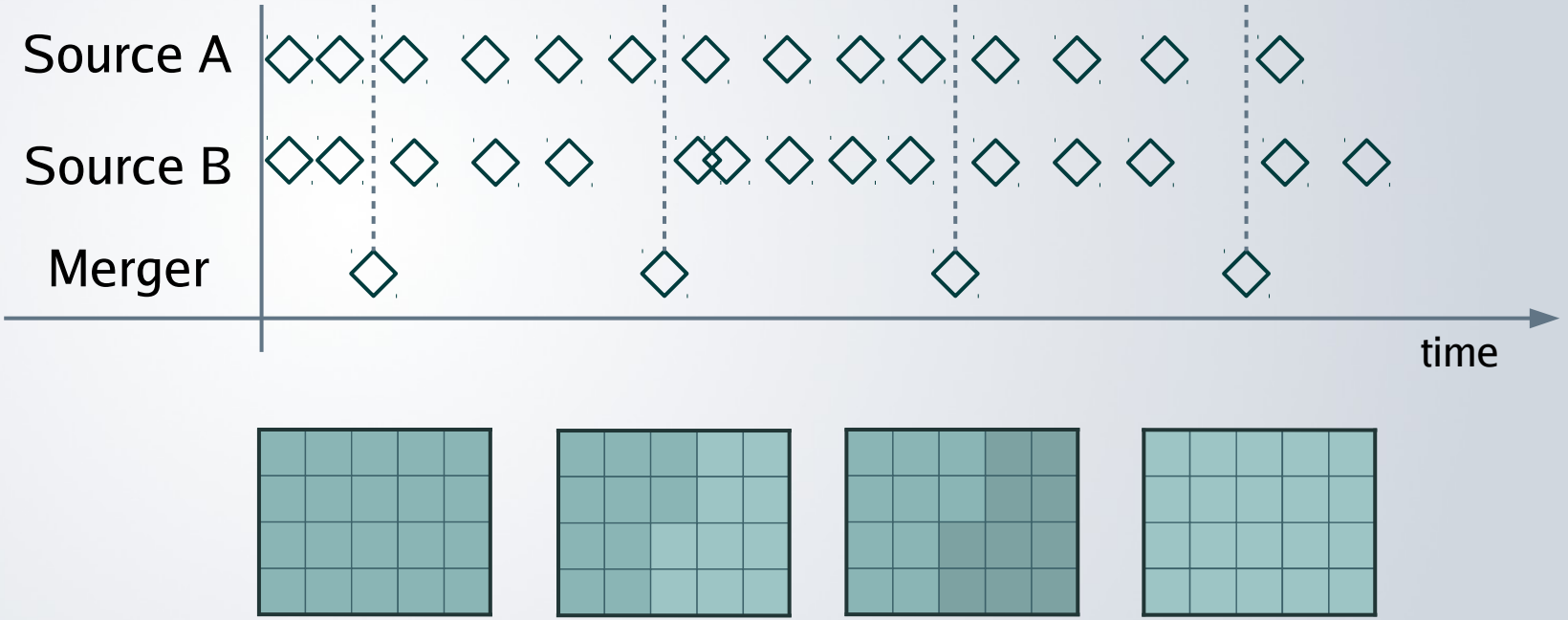
Conclusion: use deltas where possible

Implementing moving windows



Instead of the detector state, plots show fluctuations of dataflow

Implementing moving windows



Publishing merged objects less frequently than data sources reduces dataflow fluctuations

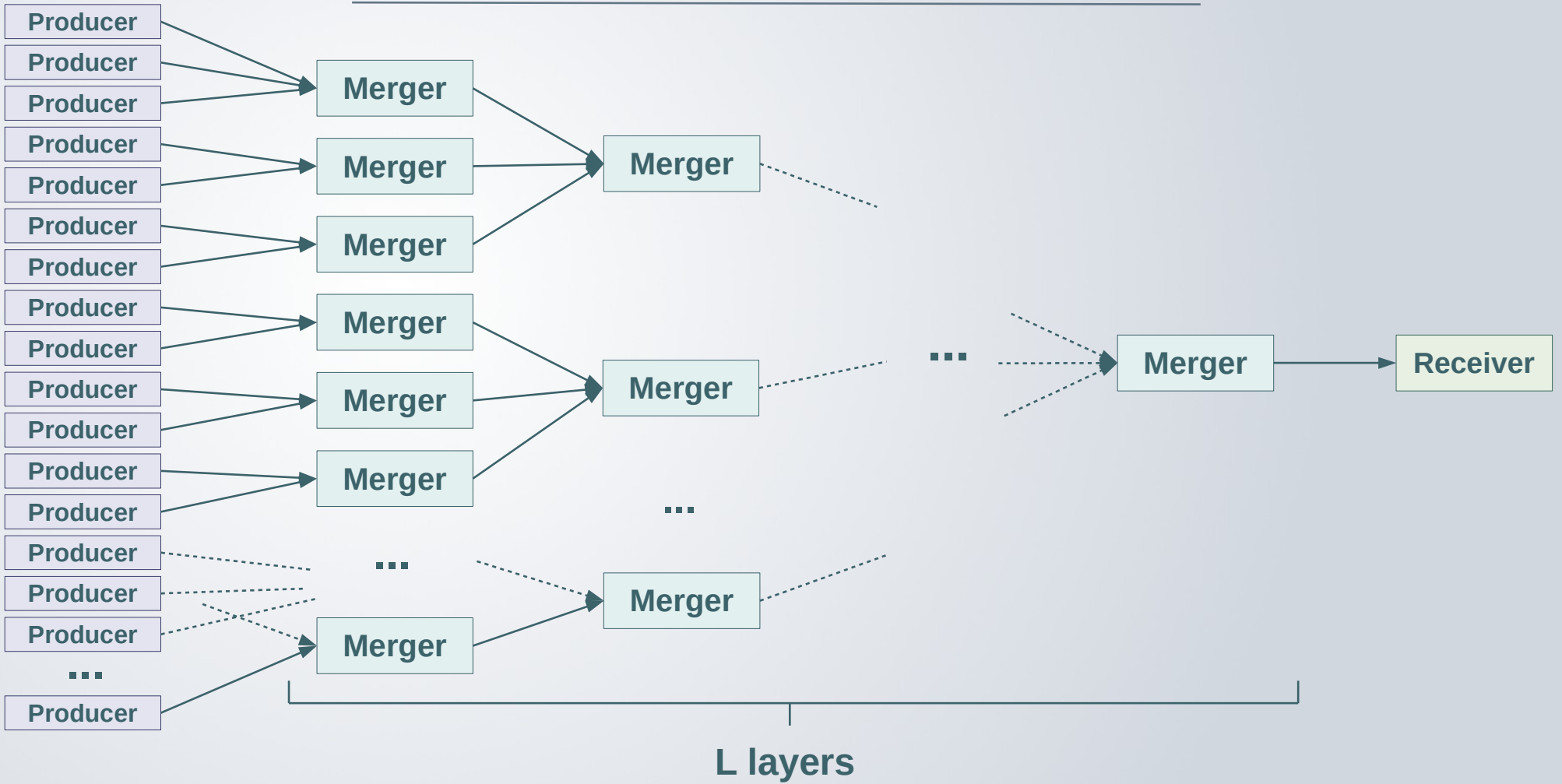
Load balancing

CPU, memory and bandwidth spikes

Load balancing

Shifting publication time in phase evens out Merger's input load

Multiple layers of Mergers



Multiple layers of Mergers – calculations

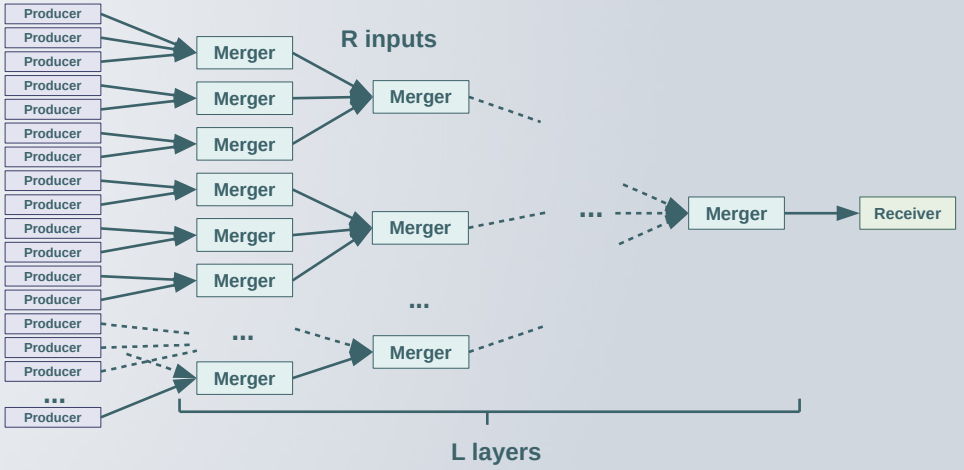
Number of mergers in i^{th} layer, where M_0 – number of producers:

- With L as the number of layers

$$M_i = \lceil M_0^{\frac{L-i}{L}} \rceil$$

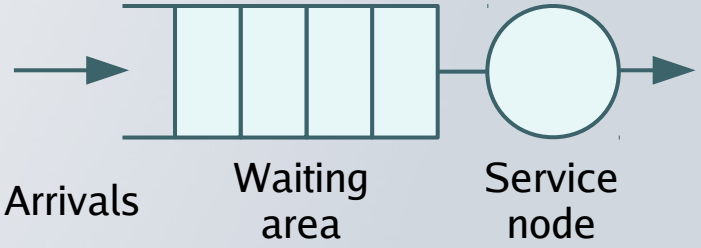
- With R as the reduction factor

$$M_i = \lceil \frac{M_{i-1}}{R} \rceil$$

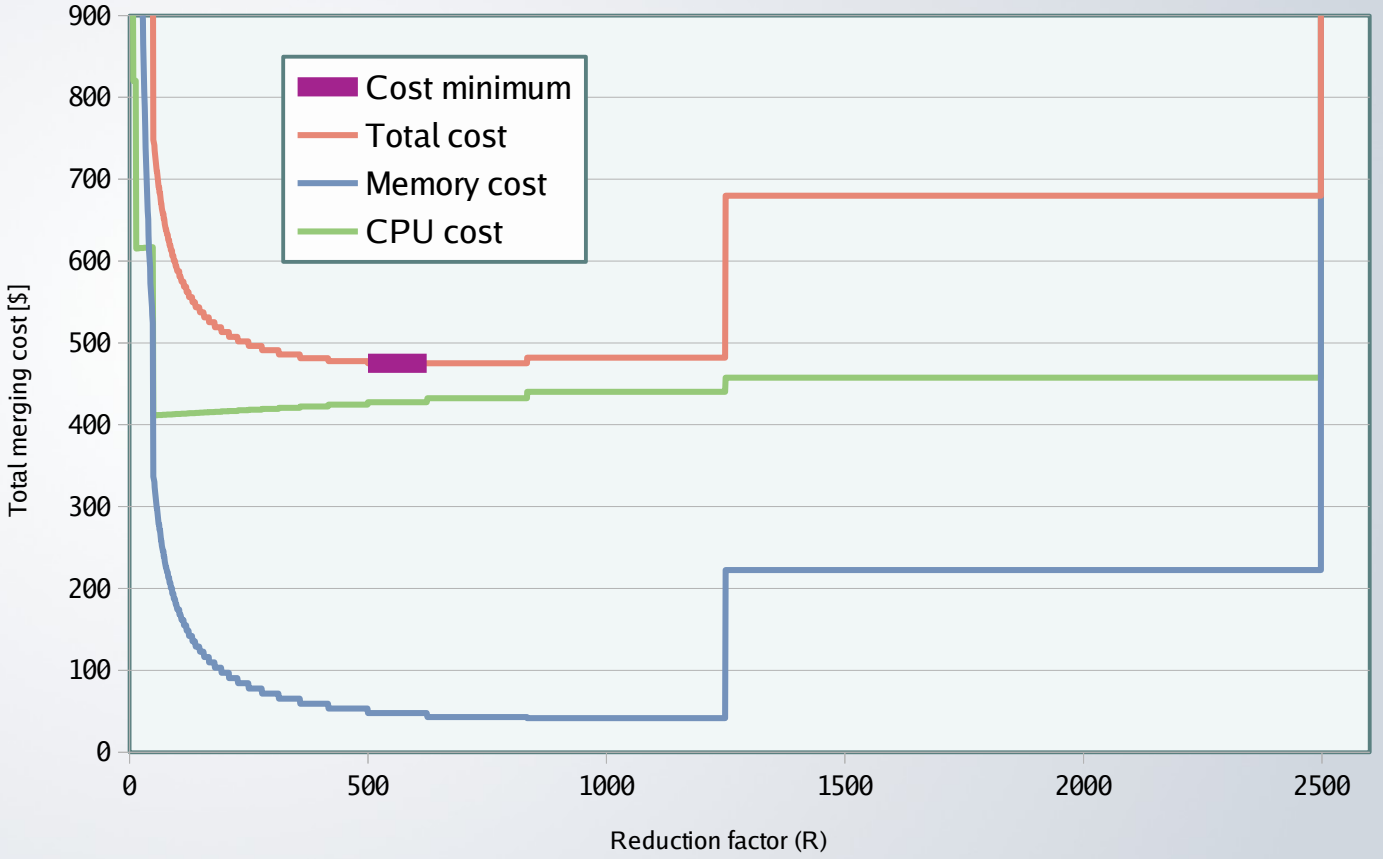


Multiple layers of Mergers - model

- Calculate the total cost of CPU and memory including:
  - Memory consumption of an idle Merger
  - Memory consumption of input queues size (*M/D/1 queue)*
  - CPU time needed to perform merging
- M/D/1 queue:
  - M: Poisson arrival process
  - D: Deterministic service time
  - 1: One server



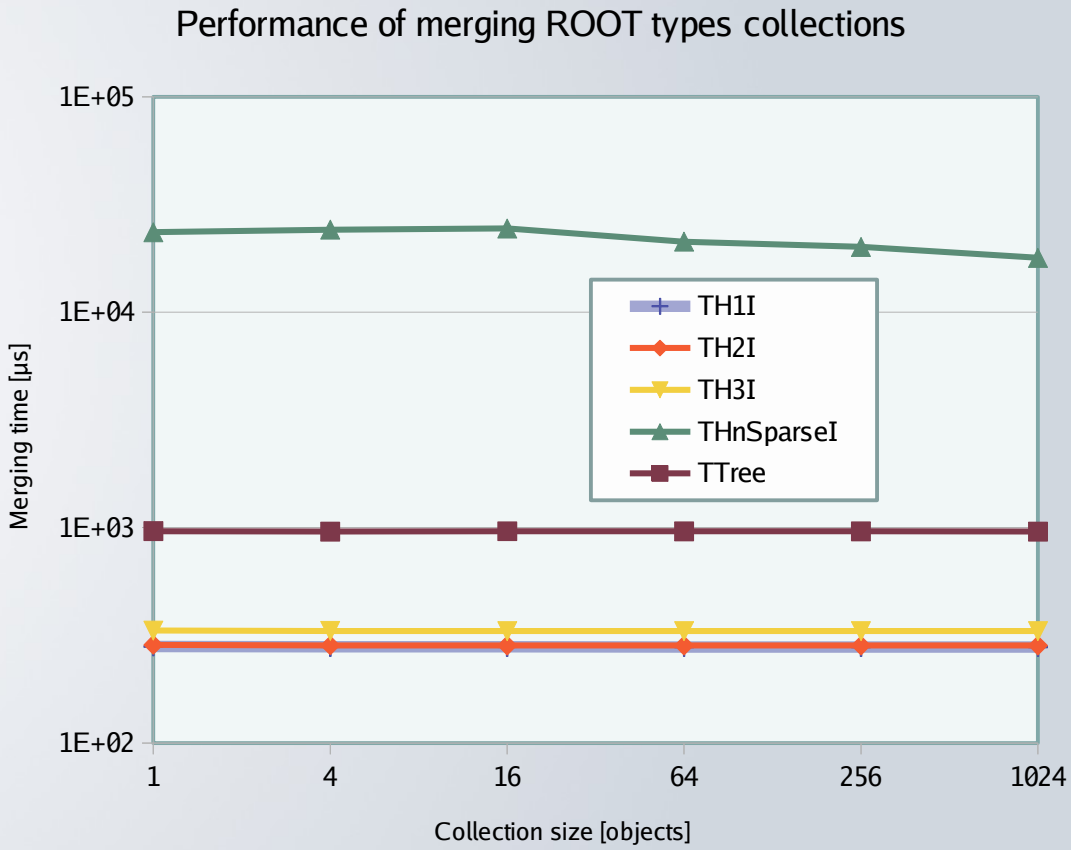
Multiple layers of Mergers - model



Using more layers might be more efficient, even if less seems enough

Collections of ROOT objects

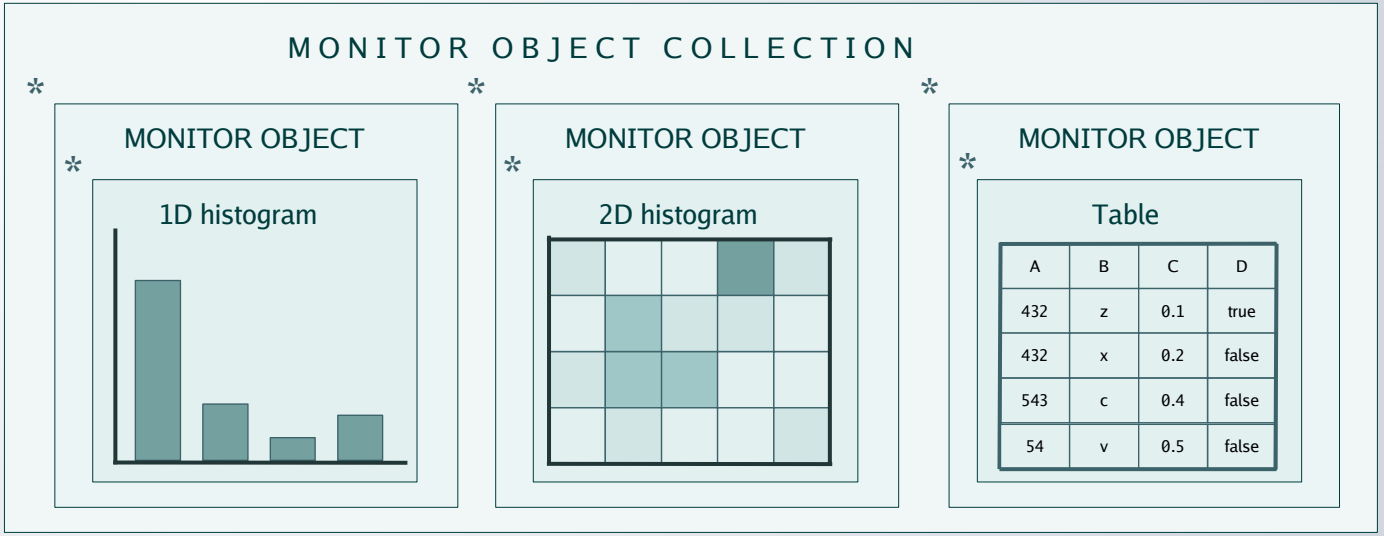
- ROOT objects offer an interface to merge collections of objects
- Standard histograms
  - No influence
- Sparse histograms
  - Merging 1024 objects in a collection is ~25% faster than individually
- Trees (tables)
  - No influence



Merging collections does not provide significant benefits

Preventing memory leaks

- ROOT objects have flags which determine ownership of other objects
- Ownership flags might be incorrectly set after deserializing a standard ROOT object
- Allow to implement a PostDeserialization() method for custom objects to allow for setting flags correctly



Object layout in the Quality Control framework

Assume that object ownership is set incorrectly after deserialization

Summary

- We lack some common language to talk about this topic
- Merging deltas is generally more efficient
- When implementing moving windows, publishing merged objects less frequently reduces dataflow fluctuations
- Shifting publication time in phase evens out Merger’s input load
- Using more layers might be more efficient, even if less seems enough
- Merging ROOT collections is usually as efficient as merging objects individually
- Assuming incorrect memory resource ownership after deserialization helps preventing memory leaks