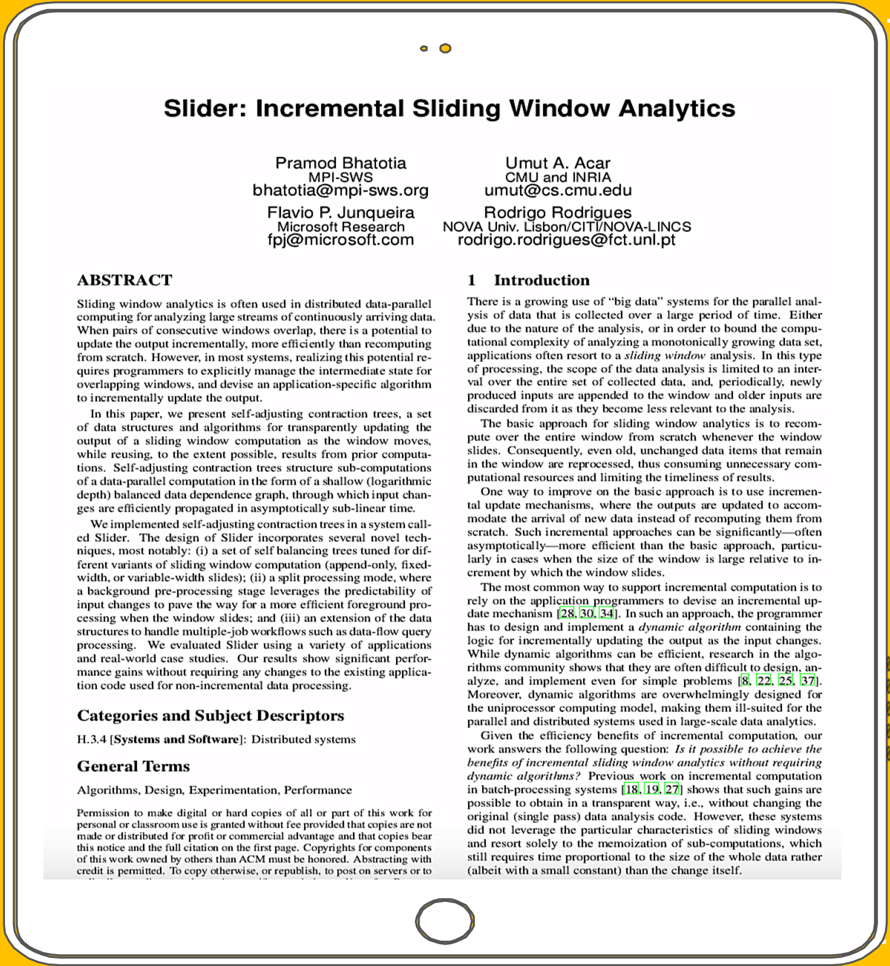


# Slider: Incremental Sliding Window Analytics

Pramod Bhatotia
MPI-SWS
bhatotia@mpi-sws.org

Umut A. Acar
CMU and INRIA
umut@cs.cmu.edu

Flavio P. Junqueira
Microsoft Research
fpj@microsoft.com

Rodrigo Rodrigues
NOVA Univ. Lisbon/CITI/NOVA-LINCS
rodrigo.rodrigues@fct.unl.pt

## ABSTRACT

Sliding window analytics is often used in distributed data-parallel computing for analyzing large streams of continuously arriving data. When pairs of consecutive windows overlap, there is a potential to update the output incrementally, more efficiently than recomputing from scratch. However, in most systems, realizing this potential requires programmers to explicitly manage the intermediate state for overlapping windows, and devise an application-specific algorithm to incrementally update the output.

In this paper, we present self-adjusting contraction trees, a set of data structures and algorithms for transparently updating the output of a sliding window computation as the window moves, while reusing, to the extent possible, results from prior computations. Self-adjusting contraction trees structure sub-computations of a data-parallel computation in the form of a shallow (logarithmic depth) balanced data dependence graph, through which input changes are efficiently propagated in asymptotically sub-linear time.

We implemented self-adjusting contraction trees in a system called Slider. The design of Slider incorporates several novel techniques, most notably: (i) a set of self balancing trees tuned for different variants of sliding window computation (append-only, fixed-width, or variable-width slides); (ii) a split processing mode, where a background pre-processing stage leverages the predictability of input changes to pave the way for a more efficient foreground processing when the window slides; and (iii) an extension of the data structures to handle multiple-job workflows such as data-flow query processing. We evaluated Slider using a variety of applications and real-world case studies. Our results show significant performance gains without requiring any changes to the existing application code used for non-incremental data processing.

## Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems

## General Terms

Algorithms, Design, Experimentation, Performance

## 1 Introduction

There is a growing use of "big data" systems for the parallel analysis of data that is collected over a large period of time. Either due to the nature of the analysis, or in order to bound the computational complexity of analyzing a monotonically growing data set, applications often resort to a *sliding window* analysis. In this type of processing, the scope of the data analysis is limited to an interval over the entire set of collected data, and, periodically, newly produced inputs are appended to the window and older inputs are discarded from it as they become less relevant to the analysis.

The basic approach for sliding window analytics is to recompute over the entire window from scratch whenever the window slides. Consequently, even old, unchanged data items that remain in the window are reprocessed, thus consuming unnecessary computational resources and limiting the timeliness of results.

One way to improve on the basic approach is to use incremental update mechanisms, where the outputs are updated to accommodate the arrival of new data instead of recomputing them from scratch. Such incremental approaches can be significantly—often asymptotically—more efficient than the basic approach, particularly in cases when the size of the window is large relative to increment by which the window slides.

The most common way to support incremental computation is to rely on the application programmers to devise an incremental update mechanism [28, 30, 34]. In such an approach, the programmer has to design and implement a *dynamic algorithm* containing the logic for incrementally updating the output as the input changes. While dynamic algorithms can be efficient, research in the algorithms community shows that they are often difficult to design, analyze, and implement even for simple problems [8, 22, 25, 37]. Moreover, dynamic algorithms are overwhelmingly designed for the uniprocessor computing model, making them ill-suited for the parallel and distributed systems used in large-scale data analytics.

Given the efficiency benefits of incremental computation, our work answers the following question: *Is it possible to achieve the benefits of incremental sliding window analytics without requiring dynamic algorithms?* Previous work on incremental computation in batch-processing systems [18, 19, 27] shows that such gains are possible to obtain in a transparent way, i.e., without changing the original (single pass) data analysis code. However, these systems did not leverage the particular characteristics of sliding windows and resort solely to the memoization of sub-computations, which still requires time proportional to the size of the whole data rather (albeit with a small constant) than the change itself.

UNIVERSITY OF WATERLOO

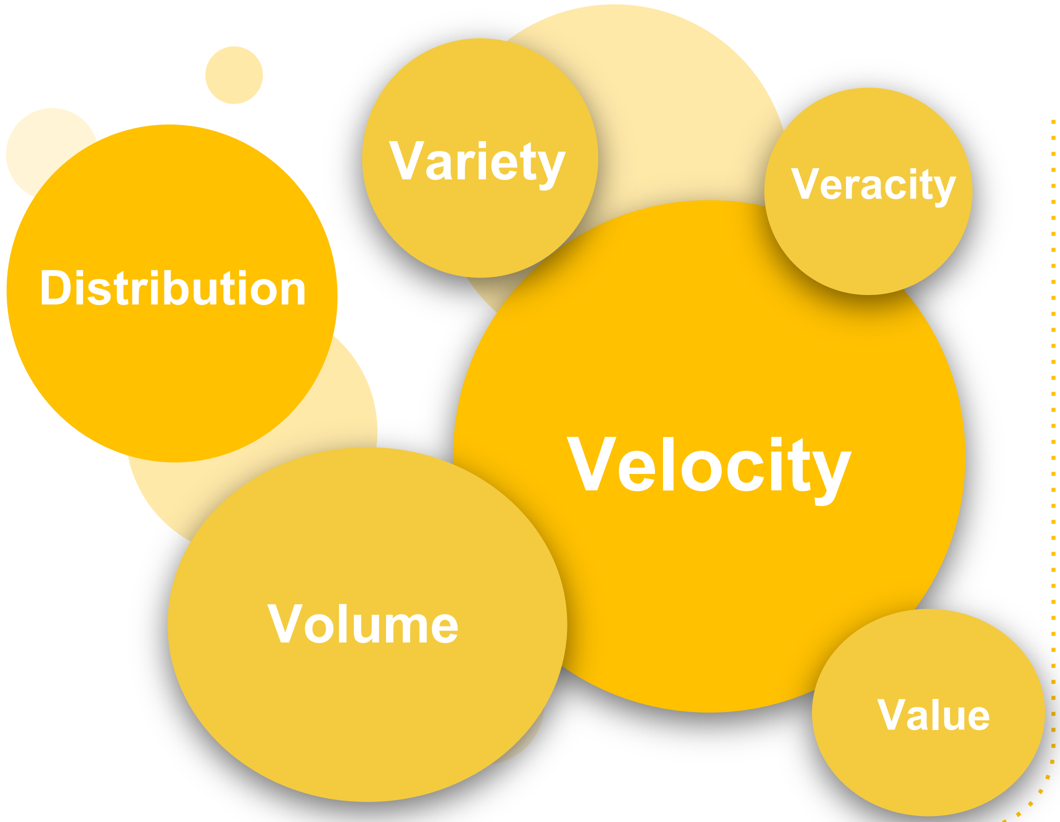
presented by

**Aida Sheshbolouki**

CS-848 - February 2019

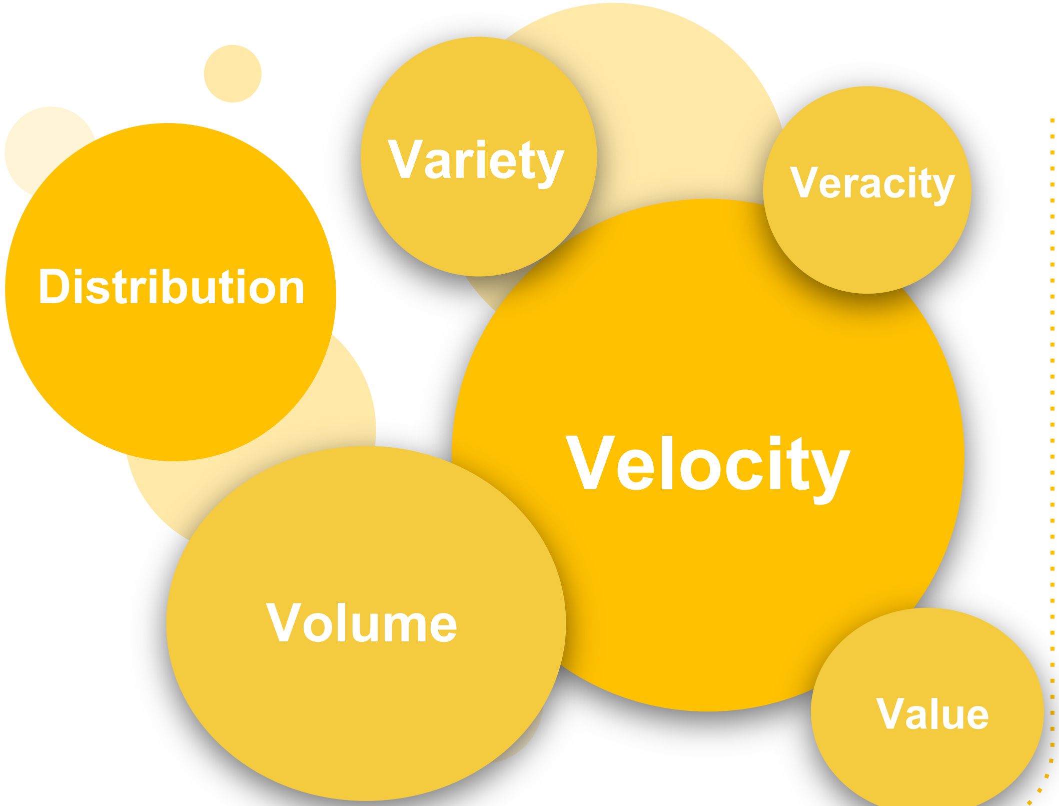
A quick recap.

## Where we are standing in **Big Data** platforms?



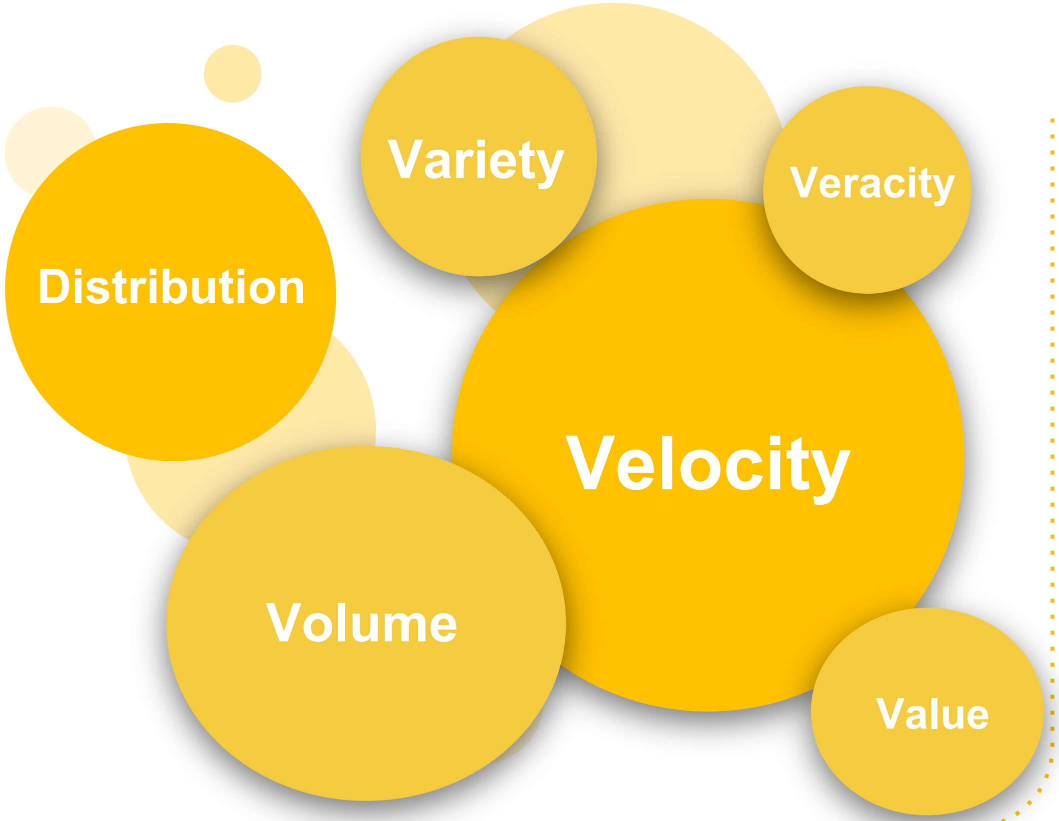
## Distributed Data Stores:

Tao, HDFS, GFS



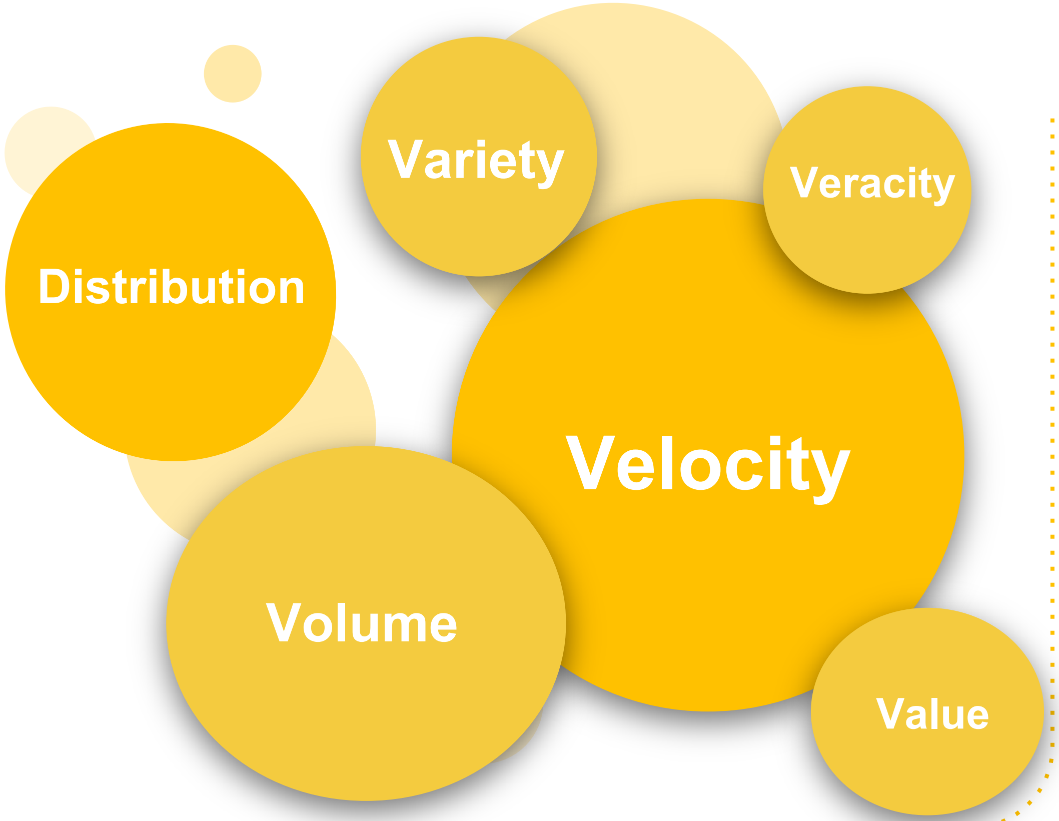
## Main Memory Systems:

Hekaton, Hyper, SAP HANA



## MapReduce based Data Management:

Spark, LinkedIn ecosystem, Hive



# Questions on Data Streams

**What are data streams?**

Unbounded sequence of values generated in real time

**How to manage and process data streams?**

Windowed analytics?

“ From the system’s point of view, it is infeasible to store an entire stream.

From the user’s point of view, recently arrived data may be more useful.

This motivates the use of windows to restrict the scope of continuous queries . ”

# Classifications of Window Models

**Direction of movement of endpoints**

- Fixed window
- Sliding window
- Landmark (append-only) window

**Window size**

- Logical or time-based Window
- Physical or count-based window
- Partitioned window
- Predicate window

Re-compute from scratch

Incremental Computations

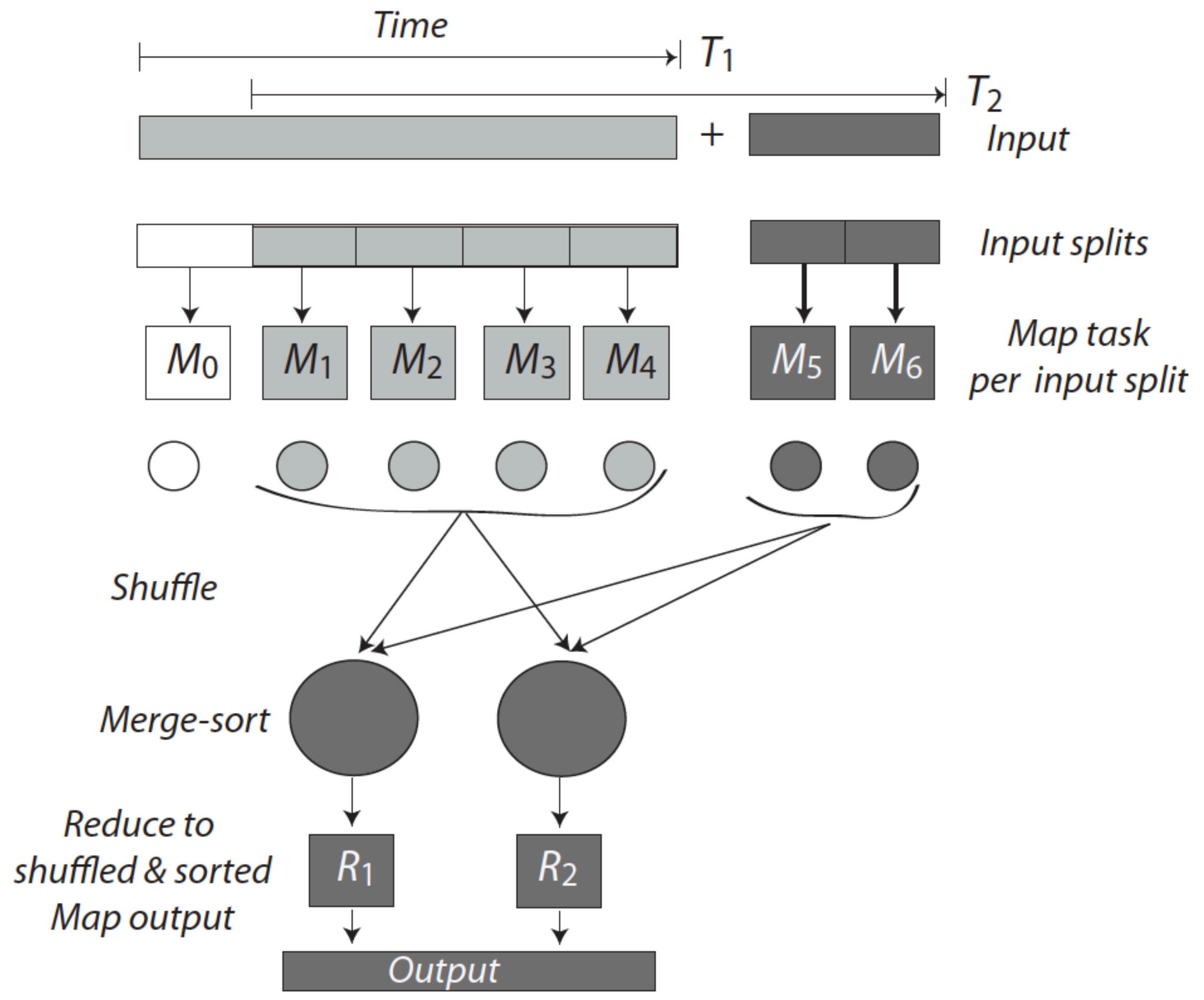
Dynamic Algorithms

?

## Slider

A system implementing **self-adjusting contraction trees**,
Supporting incremental updates efficiently and transparently in a distributed setting.

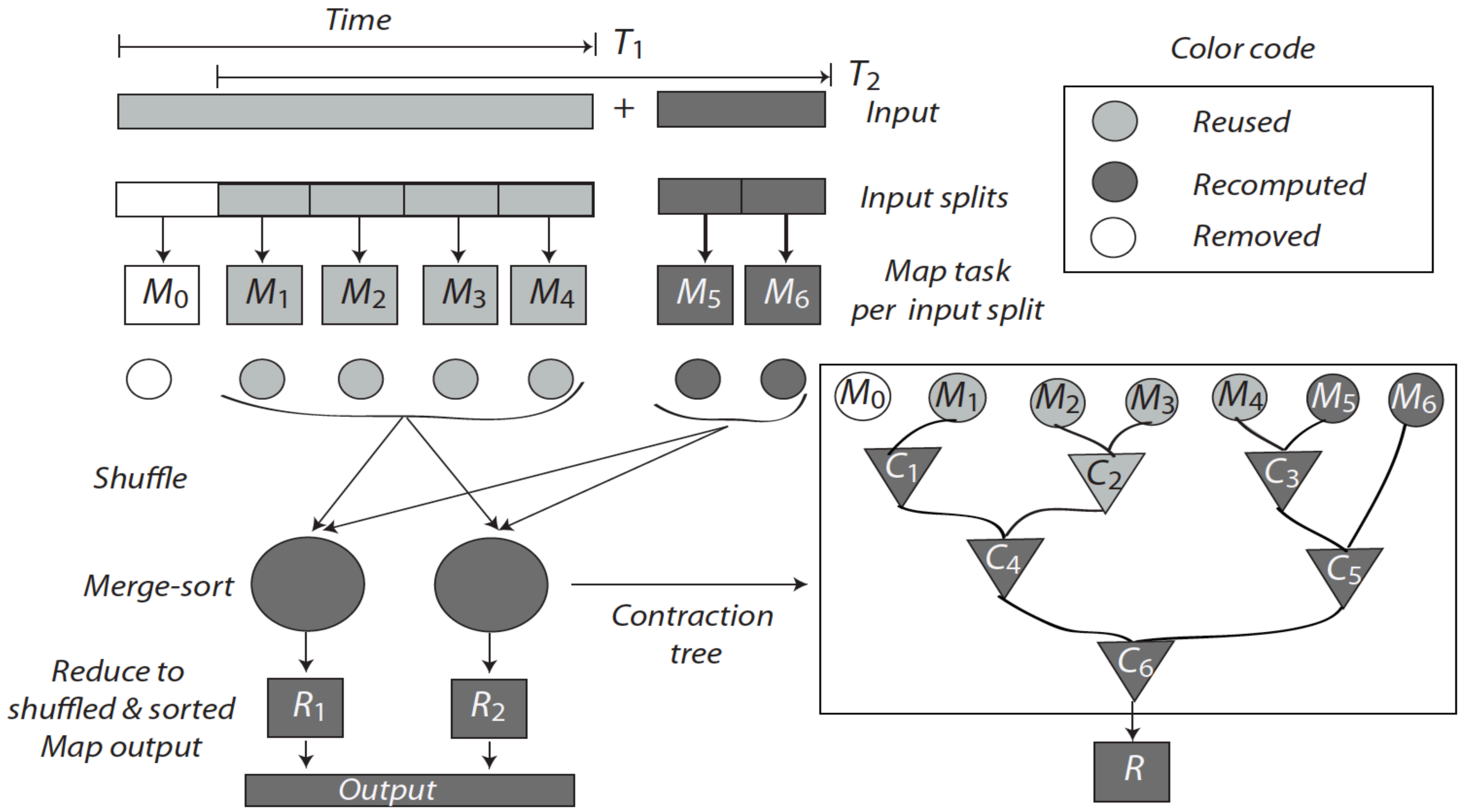
## Strawman Design



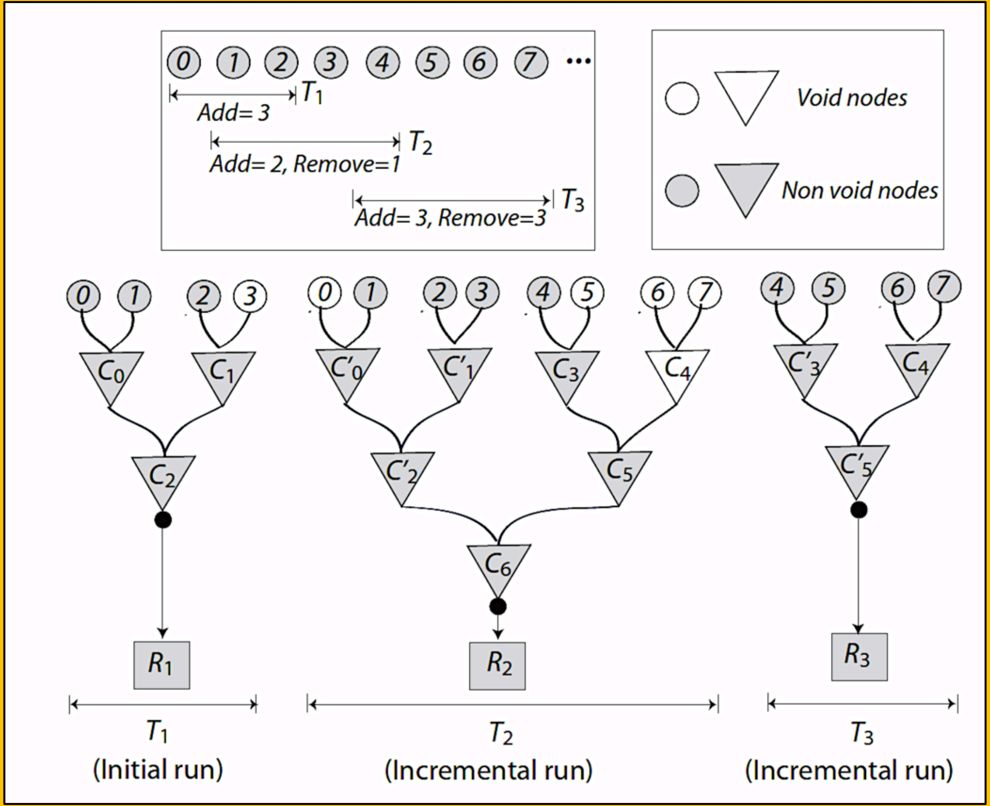
**Self-Adjusting Computation Model**

**+**

**Data Parallel Programming Model**

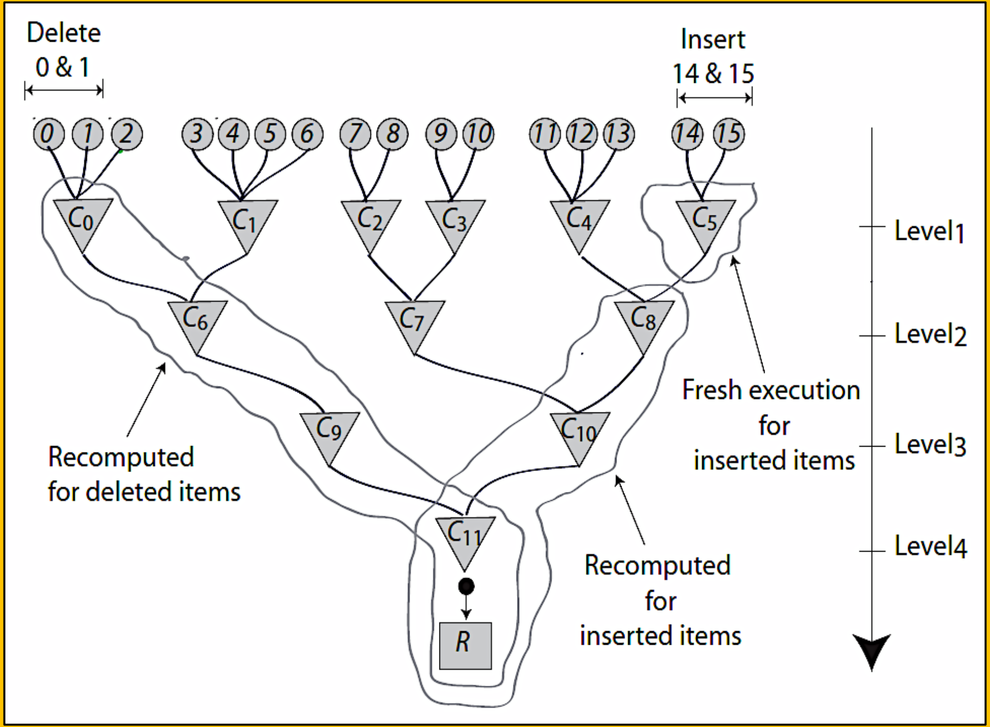


Time
$T_1$
$T_2$
Input
+
Color code
Reused
Recomputed
Removed
Input splits
$M_0$
$M_1$
$M_2$
$M_3$
$M_4$
$M_5$
$M_6$
Map task per input split
Shuffle
Merge-sort
Contraction tree
$C_1$
$C_2$
$C_3$
$C_4$
$C_5$
$C_6$
$R$
Reduce to shuffled & sorted Map output
$R_1$
$R_2$
Output



# Self-Adjusting Folding Trees

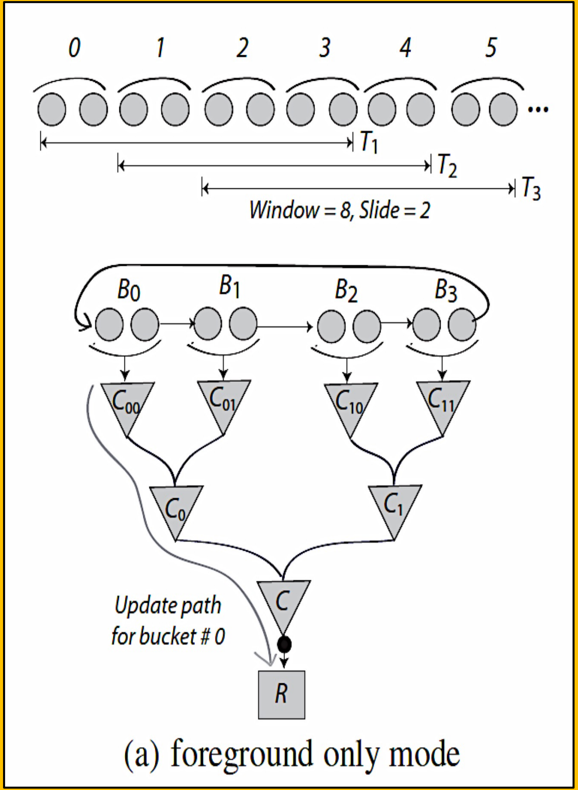
**Variable-width slides**
**rare changes of window size**

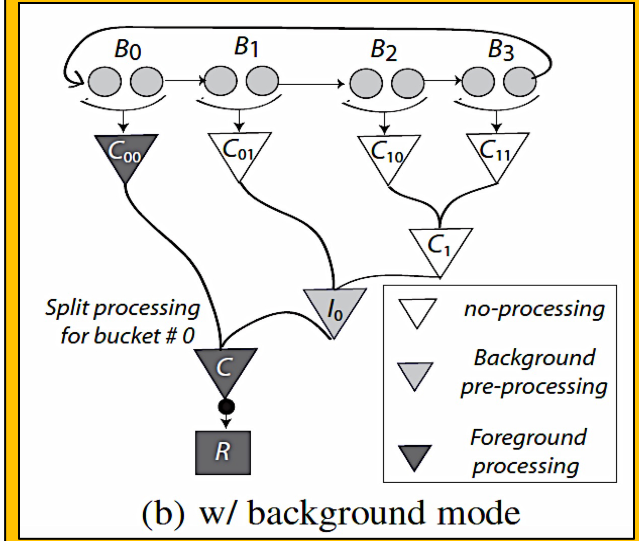


## Randomized Folding Trees

Variable-width slides
Frequent changes of window size

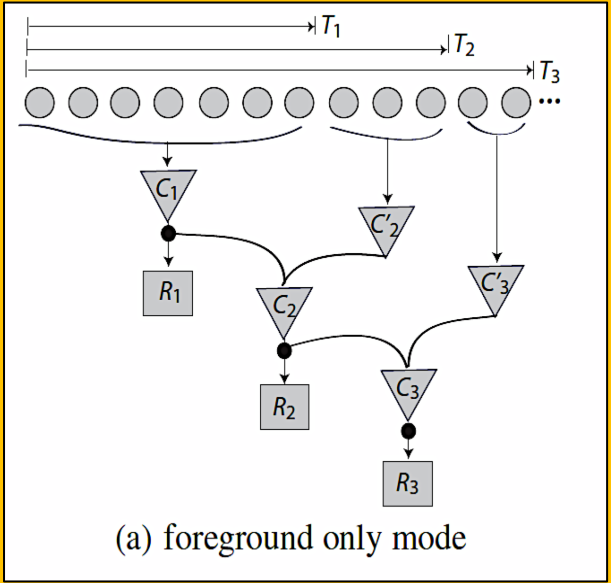
2



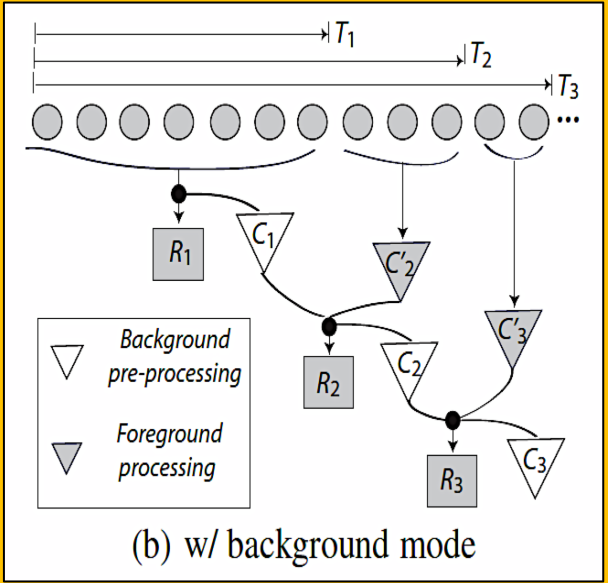


## Rotating Contraction Trees

Fixed-width slides
Split processing



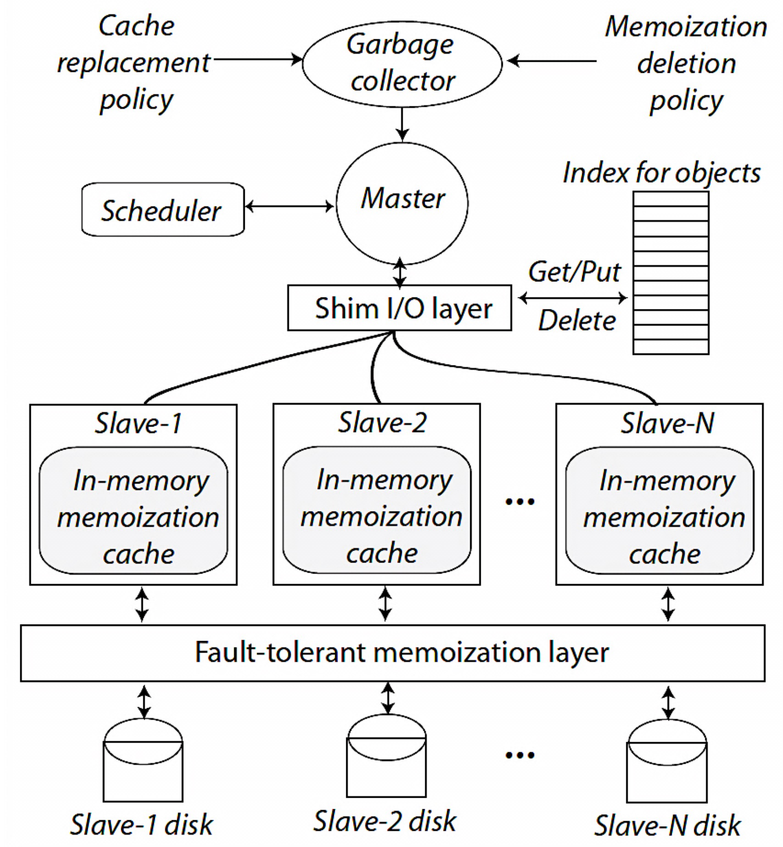
(a) foreground only mode



(b) w/ background mode

# Coalescing Contraction Trees

**Append-only Windows**
**Split processing**



## Slider Architecture

# Conclusions

**Idea.** Taking advantage of Contraction Trees for efficient update and organized dependency graph (**Stable Window Computations**)

☺

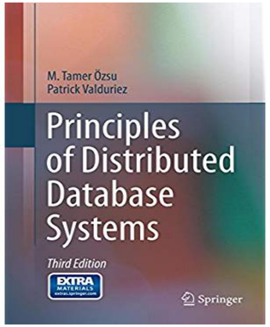
**Performance gains** w.r.t computing from scratch and Strawman approach.
**Effective optimizations:** Split-processing, scheduler modifications, Data-flow query interface, In-memory distributed caching.

☹

**Performance Overhead.** High runtime overhead for data-intensive applications.

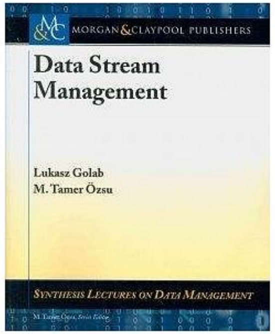
**Space Overhead.** High space overhead for specific data-intensive applications especially in variable -width windows.

## References



M. Tamer Özsu
Patrick Valduriez
Principles of Distributed Database Systems
Third Edition
EXTRA MATERIALS
Springer

Özsu, M. Tamer, and Patrick Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.



MORGAN & CLAYPOOL PUBLISHERS
Data Stream Management
Lukasz Golab
M. Tamer Özsu
SYNTHESIS LECTURES ON DATA MANAGEMENT

Golab, Lukasz, and M. Tamer Özsu. "Data Stream Management." *Synthesis Lectures on Data Management* 2, no. 1 (2010): 1-73

acm Association for Computing Machinery

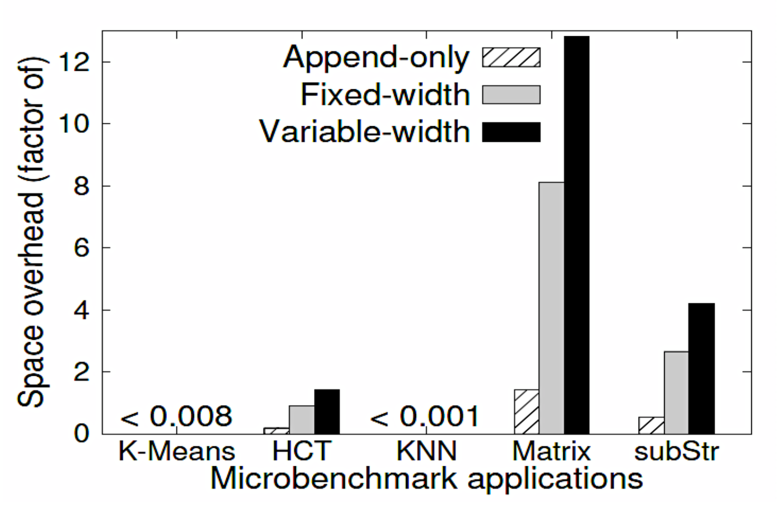
Garofalakis, Minos. “Streaming Analytics in 1st Europe Summer School: Data Science” ACM, 2017

***The simple act of paying attention can take you a long way.***

*-Keanu Reeves*

*Thanks for your attention!*

## Discussion



How can garbage collection mechanisms compensate for the space overhead of data-intensive applications?

What are differences between the architectures of Spark Stream and Slider ?

Slider supports aggregation functions with associative property. What about other computations?