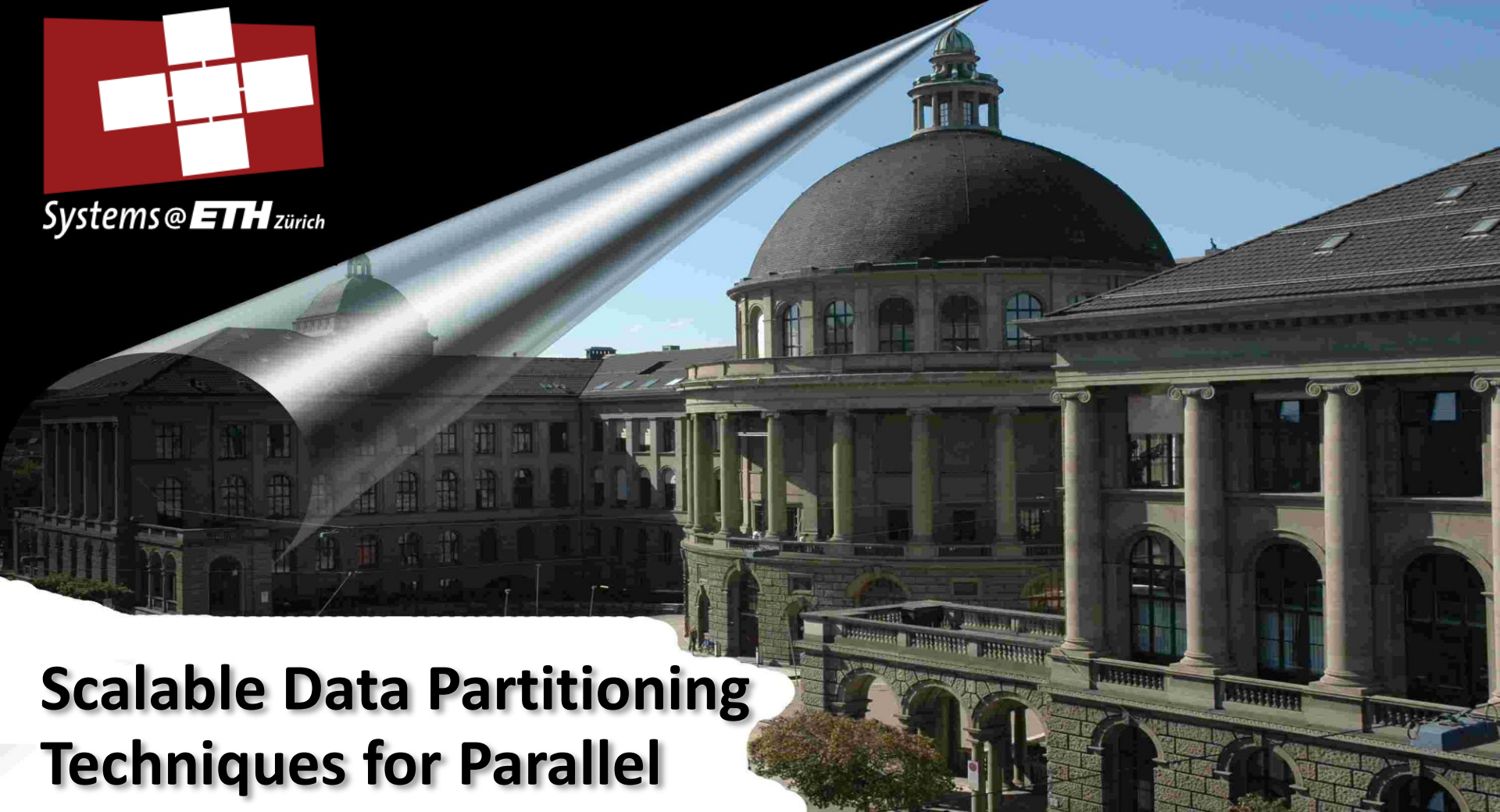


Systems@ETH zürich



Scalable Data Partitioning Techniques for Parallel Sliding Window Processing over Data Streams

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

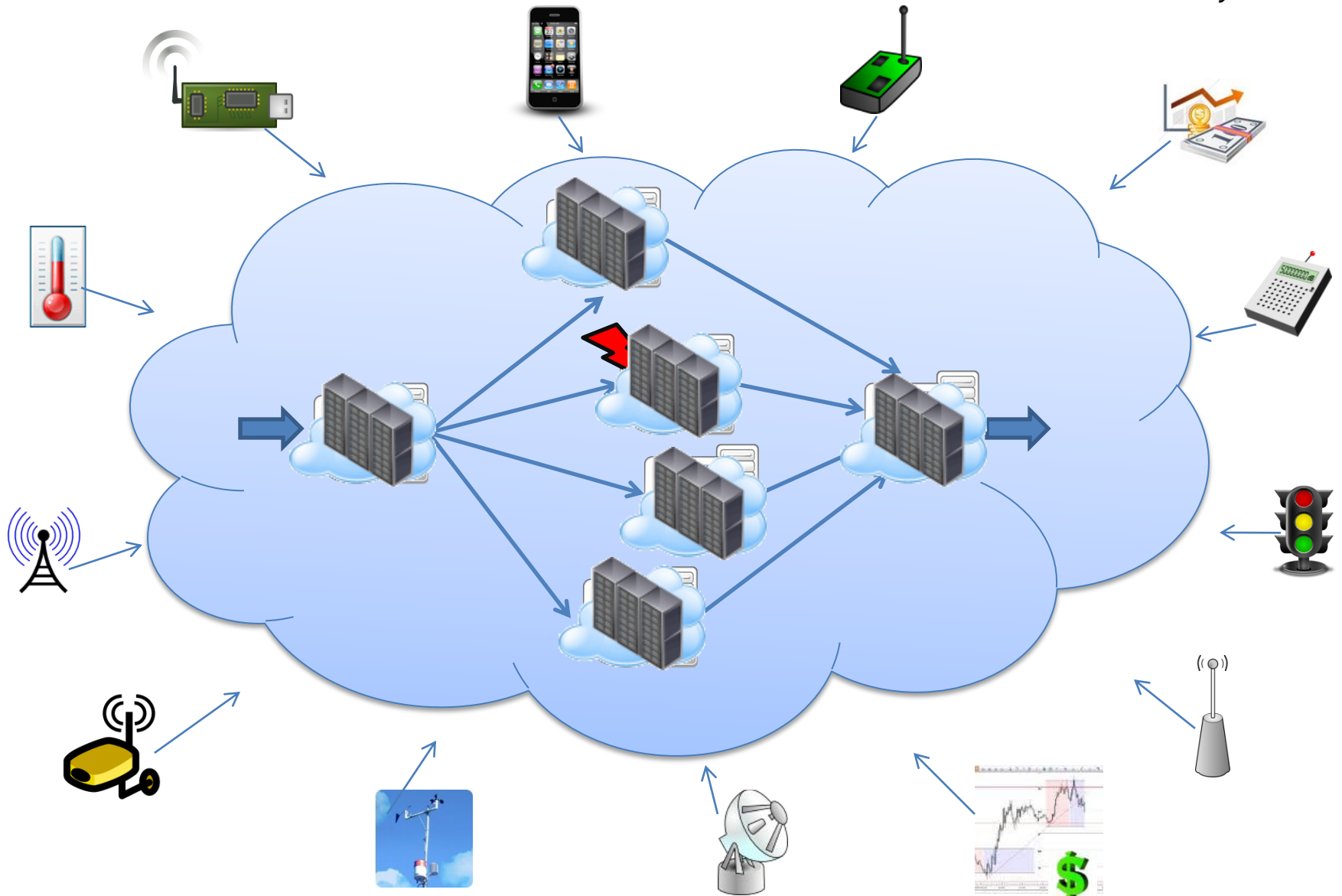
DMSN 2011

Cagri Balkesen & Nesime Tatbul

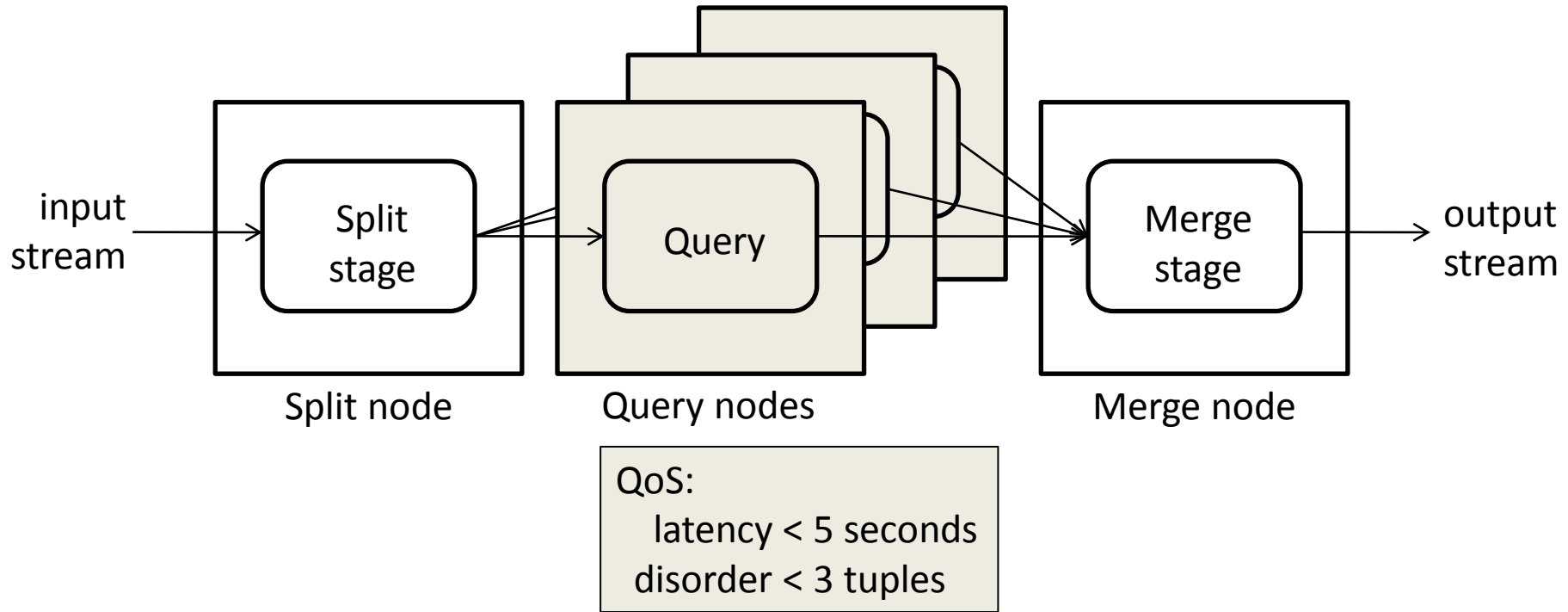
Talk Outline

- Intro & Motivation
- Stream Partitioning Techniques
 - Basic window partitioning
 - Batch partitioning
 - Pane-based partitioning
- Ring-based Query Evaluation
- Experimental Evaluation
- Conclusions & Future Work

Intro & Motivation



Architectural Overview



- Classical Split-Merge pattern from Parallel DBs
- Adjustable parallelism level, d
- QoS on max latency & order

Related Work: How to Partition?

- Content-sensitive
 - FluX: Fault-tolerant, load balancing Exchange [1,2]
 - Use group-by values from the query to partition
 - Need explicit load-balancing due to skewed data
- Content-insensitive
 - GDSM: Window-based parallelization (fixed-size tumbling wins) [3]
 - Win-Distribute: Partition at window boundaries
 - Win-Split: Partition each win into equi-length subwins
- The Problem:
 - How to handle sliding windows?
 - How to handle queries without group-by or a few groups?

[1] Flux: An Adaptive Partitioning Operator for Continuous Query Systems, ICDE'03

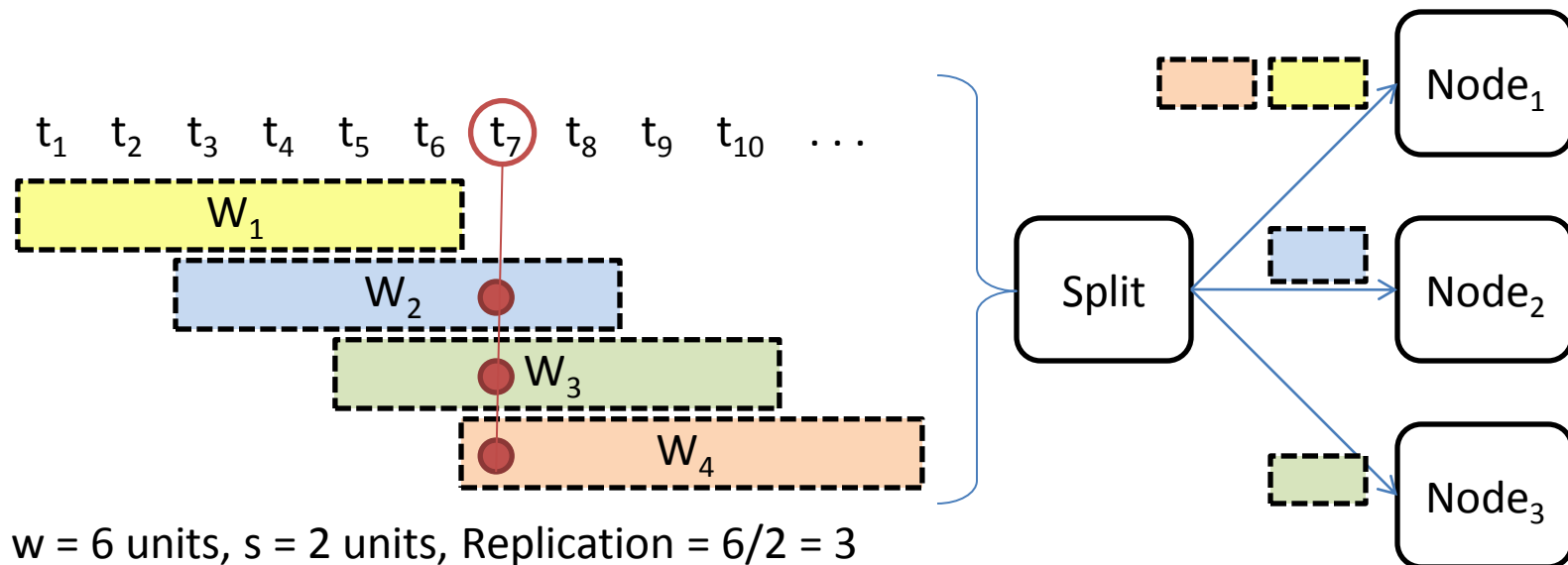
[2] Highly-Available, Fault-Tolerant, Parallel Dataflows, SIGMOD '04

[3] Customizable Parallel Execution of Scientific Stream Queries, VLDB '05

Stream Partitioning Techniques

Approach 1: Basic Sliding Window Partitioning

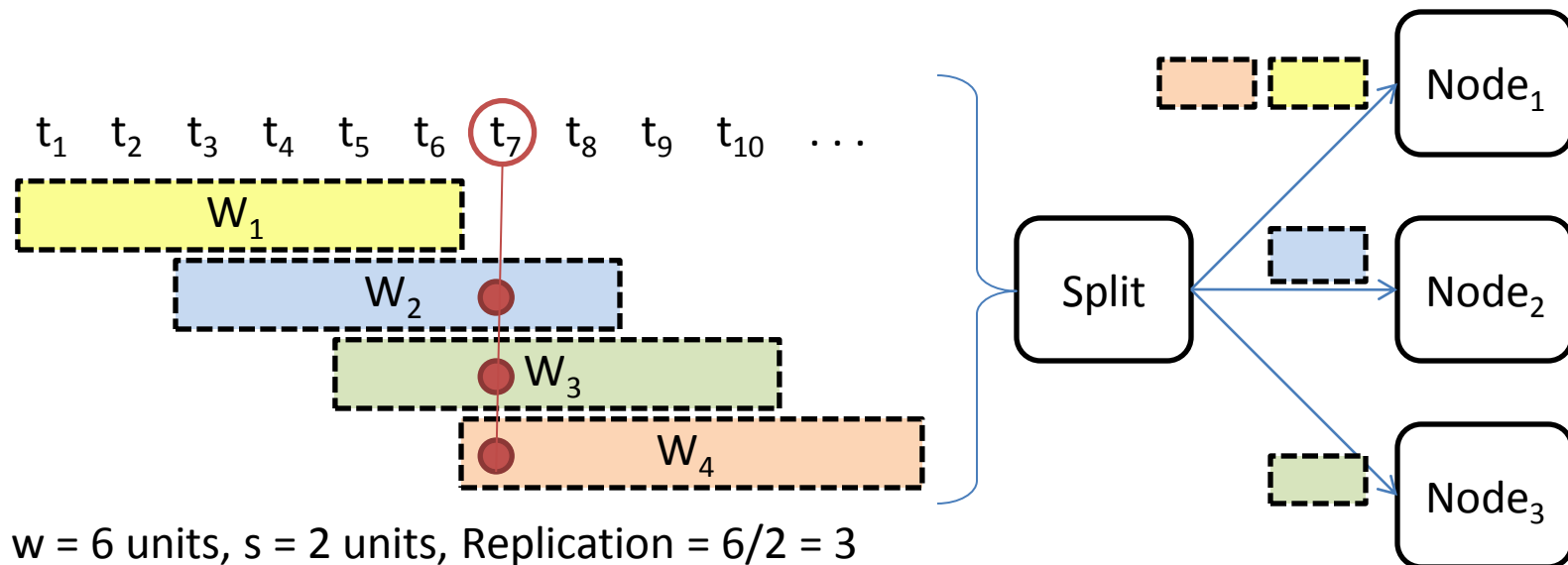
- Independently processable chunking
 - Window aware splitting of the stream
- Each window has an id & tuples are marked
 - *(first-winid, last-winid, is-win-closer)*
- Tuples are replicated for each of their windows



Approach 1: Basic Sliding Window Partitioning

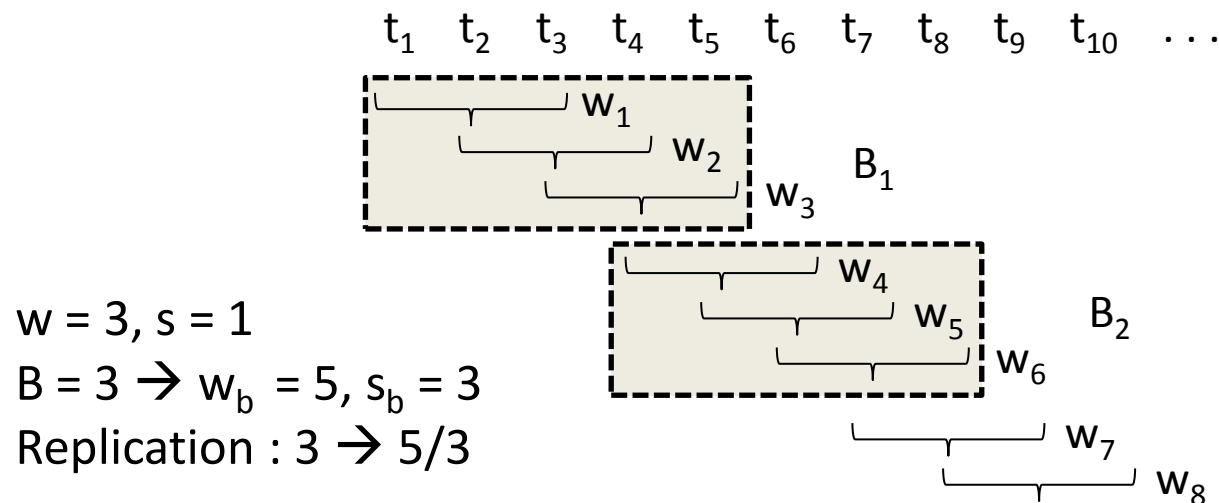
The Problem with Basic sliding window partitioning:

- Tuples belong to many windows depending on slide
- Excessive replication of tuples for each window
- Increase in output data volume of split



Approach 2: Batch-based Partitioning

- Batch several windows together to reduce replication
- “Batch-window”: $w_b = w + (B - 1) * s$; $s_b = B * s$
 - All the tuples in a batch go to the same partition
 - Only tuples overlapping btw. batches are replicated
- Replication reduced to w_b / s_b partitions instead of w / s



Definitions:

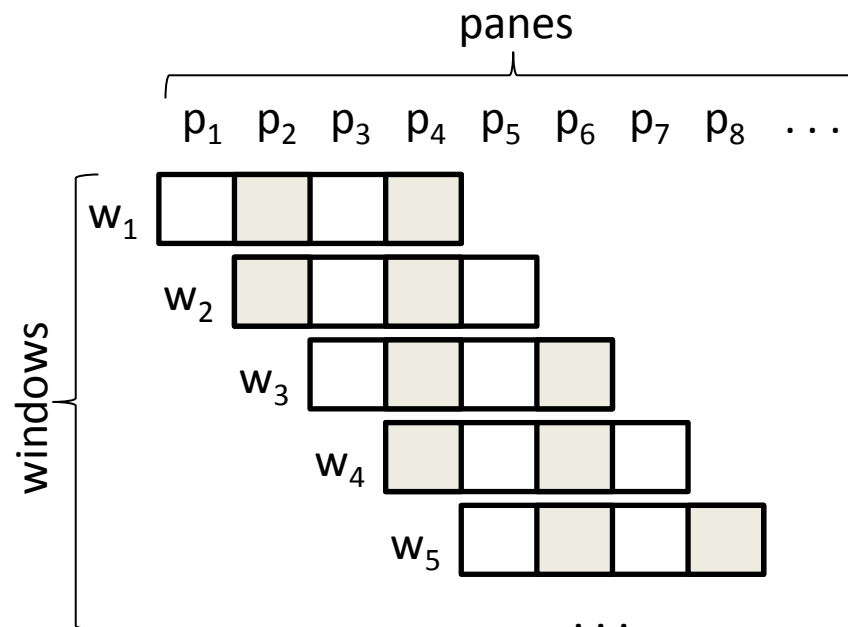
w : window-size

s : slide-size

B : batch-size

The Panes Technique

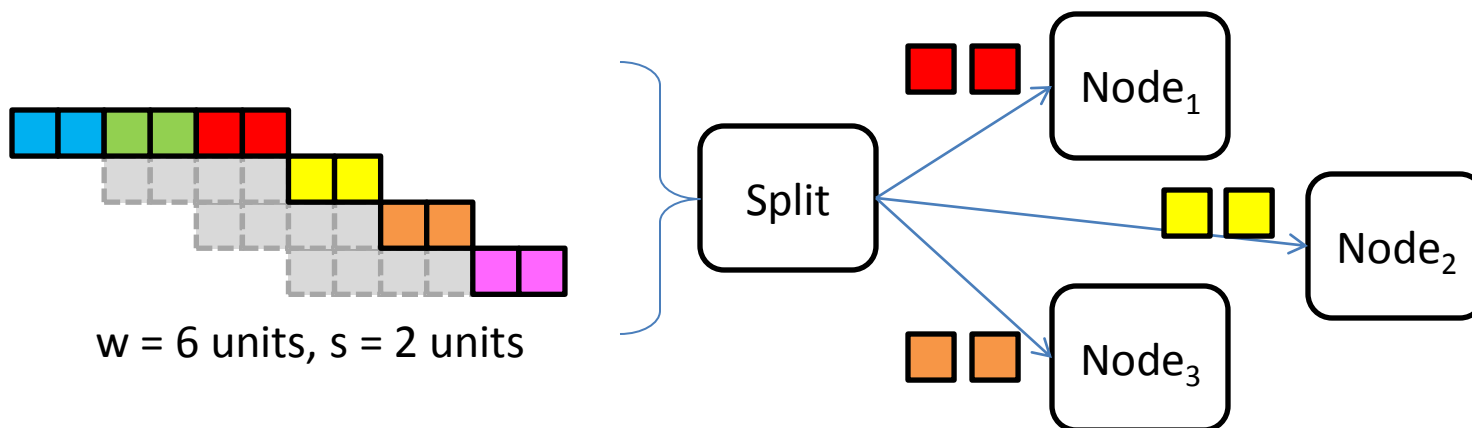
- Divide overlapping windows into disjoint panes
- Reduce cost by sub-aggregation and sharing
- Each window has $w/\text{gcd}(w,s)$ panes of size $\text{gcd}(w,s)$
- Query is decomposed: pane-level (**PLQ**) & window-level (**WLQ**) queries



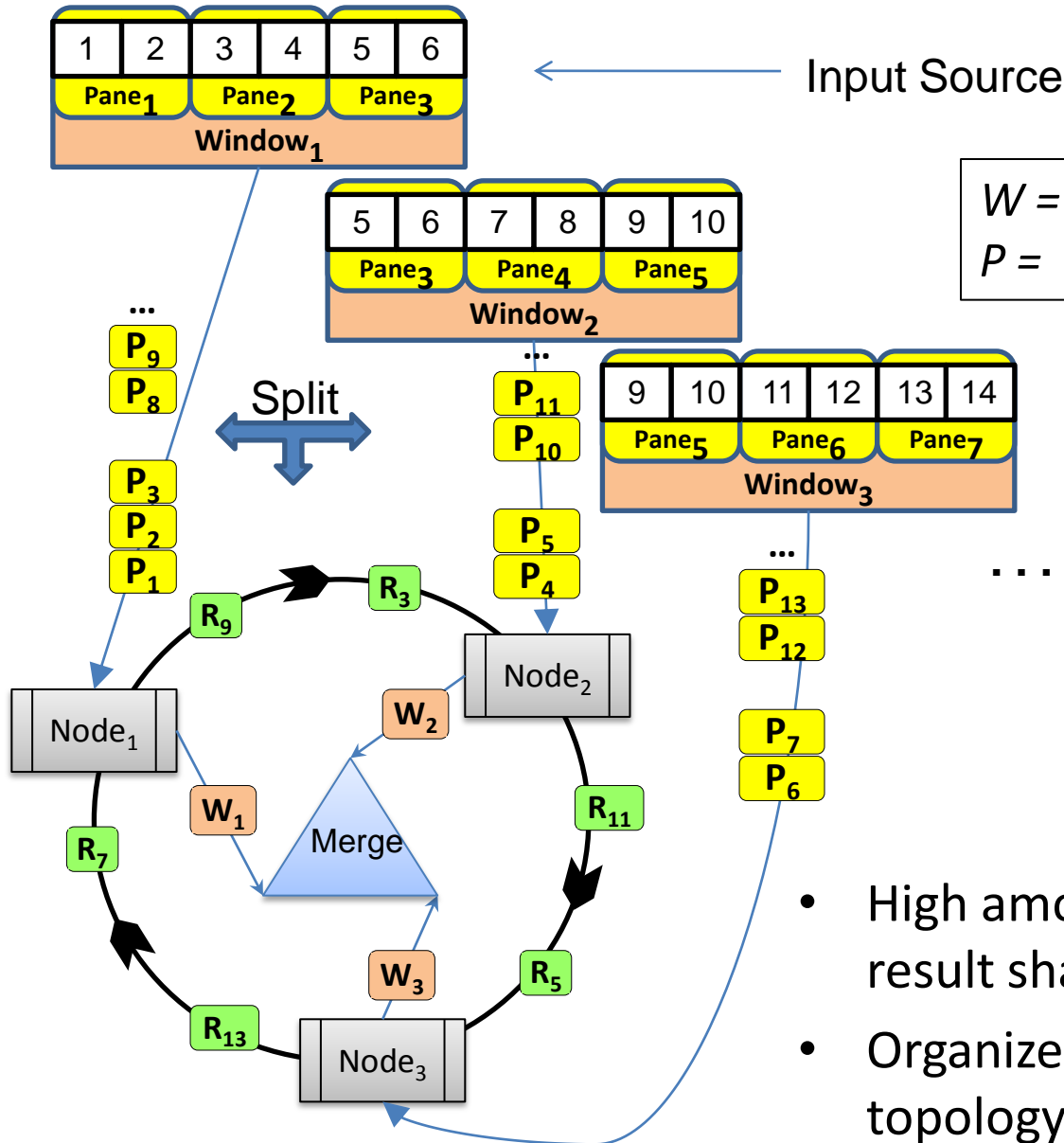
[1] No Pane, No Gain: Efficient Evaluation of Sliding Window Aggregates over Data Streams, SIGMOD Record '05

Approach 3: Pane-based Partitioning

- Mark each tuple with pane-id + win-id
 - Treat panes as tumbling window with $w_p = s_p = \gcd(w,s)$
- Route tuples to a node based on pane-id
- Nodes compute PLQ with pane tuples
- Combine all PLQ results of a window to form WLQ
 - Need for an organized topology of nodes
 - We propose organization of nodes in a ring



Ring-based Query Evaluation



$W = 6, S = 4$ tuples
 $P = \text{GCD}(6,4) = 2$ tuples

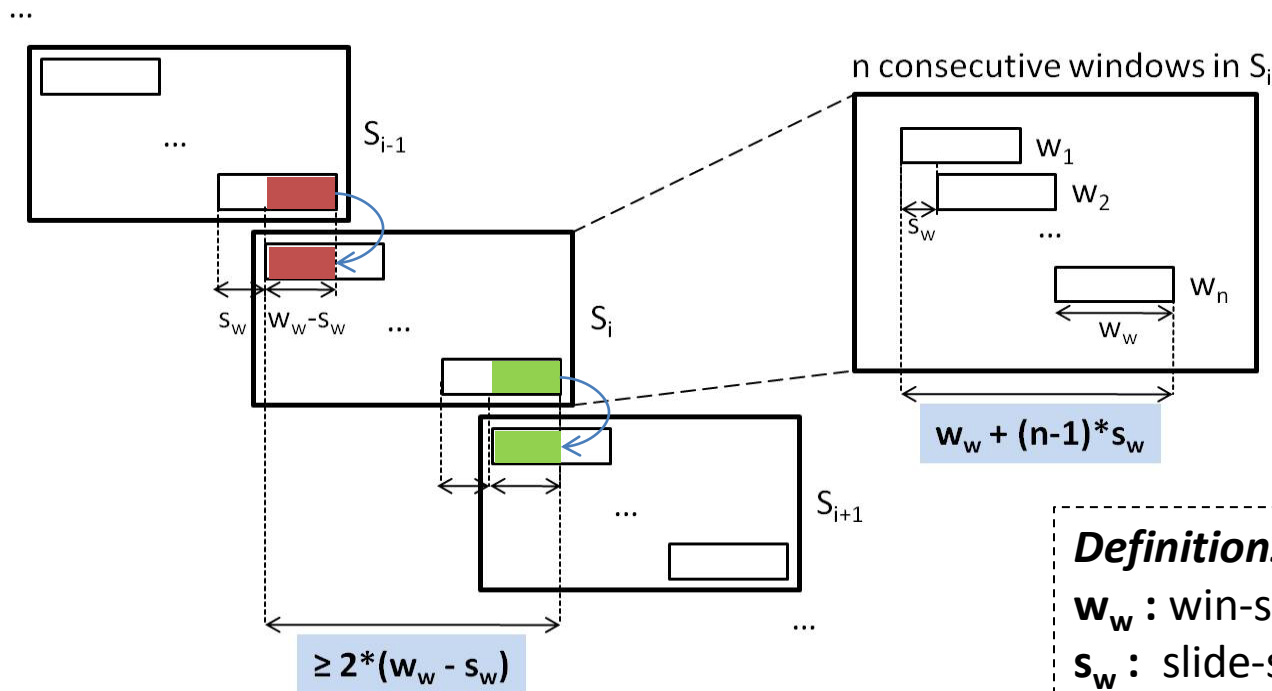
- High amount of pipelined result sharing among nodes
- Organized communication topology

Assignment of Windows and Panes to Nodes

- All pane results only arrive from predecessors
- Pane results sent to successor is only local panes

- Each node is assigned n consecutive windows

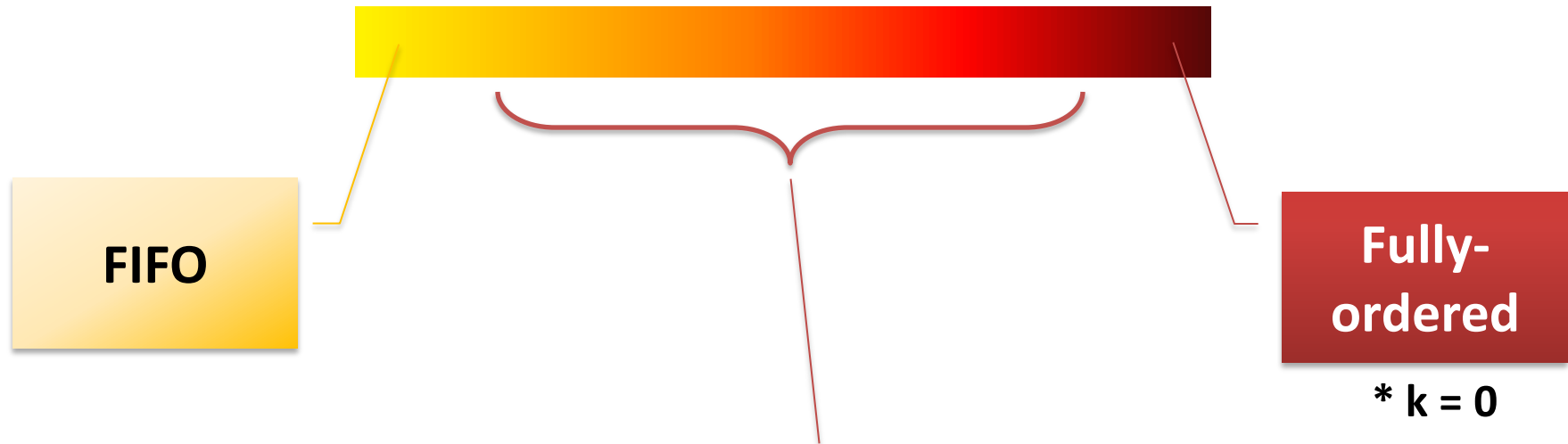
- Min n st. $w_w + (n - 1) * s_w \geq 2 * (w_w - s_w) \implies n \geq \frac{w_w - s_w}{s_w}$



Definitions:

w_w : win-size in # of panes
 s_w : slide-size in # of panes

Flexible Result Merging

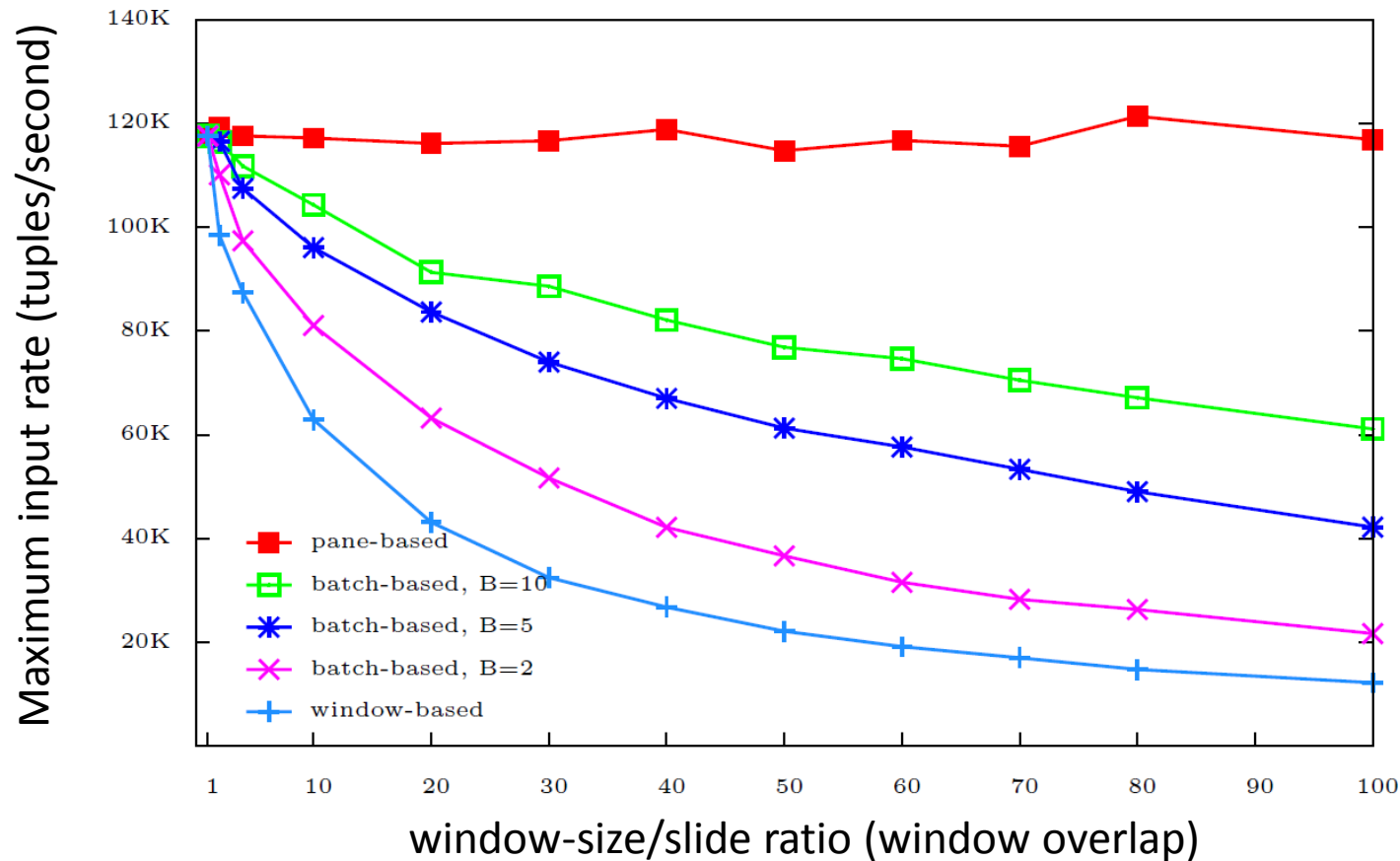


k-ordered: k-ordering constraint [1],
certain disorder allowed

Defn: For any tuple s , s' arrives at
least $k+1$ tuples after s st. $s'.A \geq s.A$

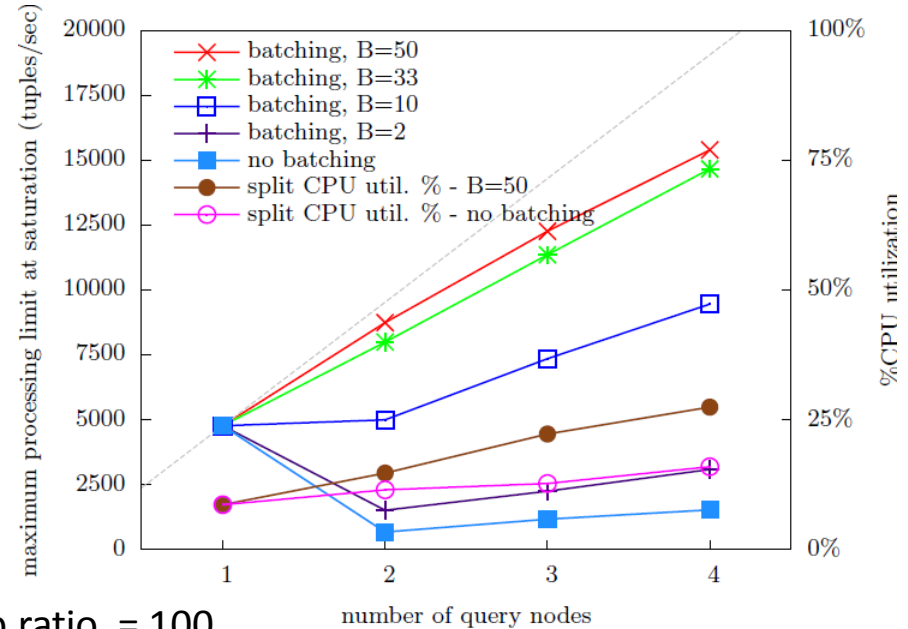
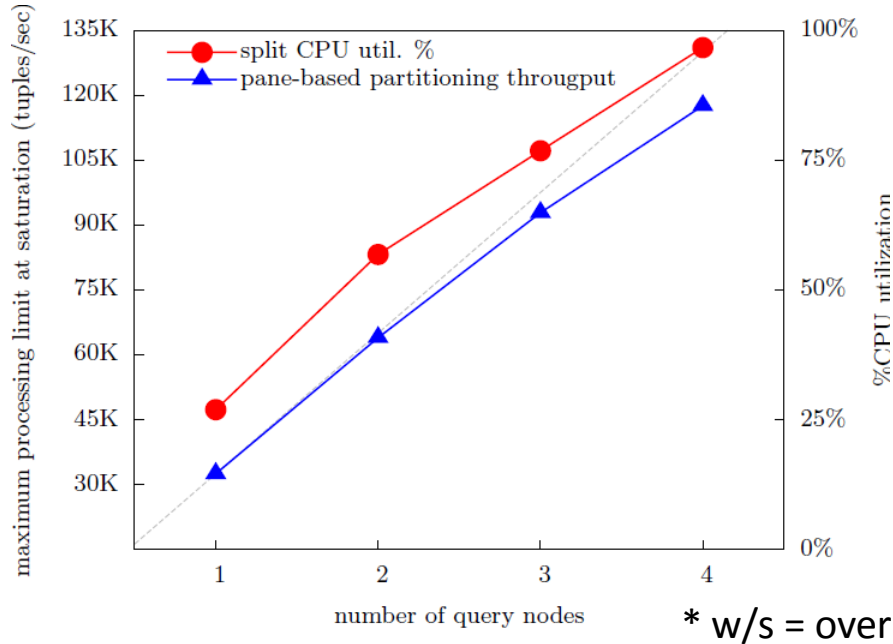
- Implementation of techniques in Borealis
- Workload adapted from Linear Road Benchmark
 - Slightly modified segment statistics queries
 - Basic aggregation functions with different window/slide ratios

Scalability of Split Operator



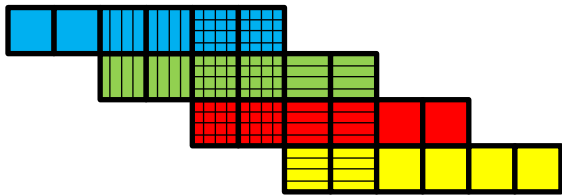
- Pane-partitioning: cost & tput constant regardless of overlap ratio
- Window & batch –partitioning: cost \uparrow and tput \downarrow as overlap \uparrow
- Excessive replication in window-partitioning is reduced by batching

Scalability of Partitioning Techniques

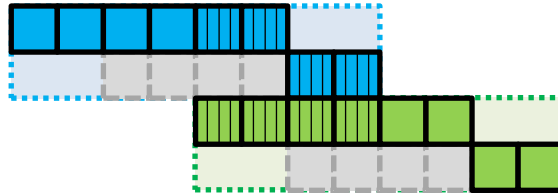


- Pane-based scales close to linear until split is saturated
 - per tuple cost is constant
- Window & batch based: extremely high replication
 - Split is not saturated, but scales very slowly

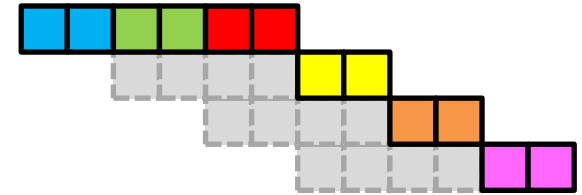
Summary & Conclusions



1) Window-based



2) Batch-based



3) Pane-based

- Pane-partitioning is the choice of partitioning
 - Avoids tuple replication
 - Incurs less overhead in split and aggregate
 - Scales close to linear

Ongoing & Future Work

- Generalization of the framework
- Support for adaptivity during runtime
- Extending complexity of query plans
- Extending performance analysis & experiments

Thank You!

