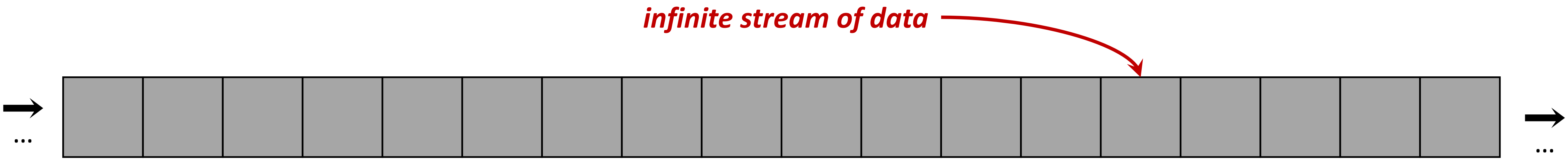
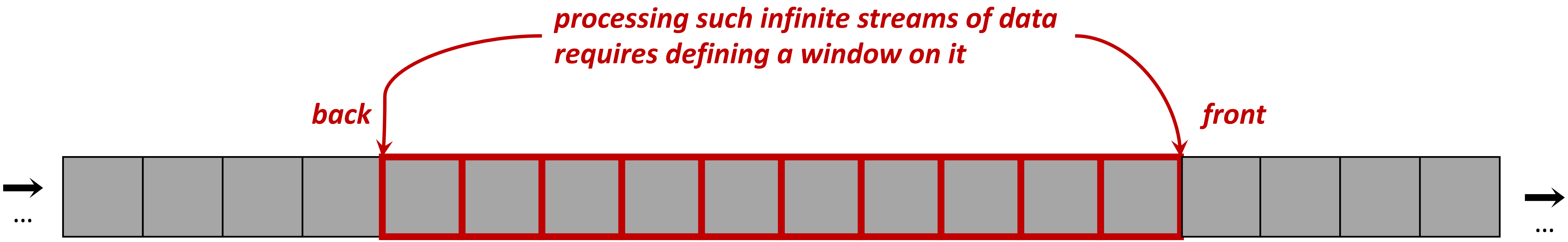
Optimal and General Out-of-Order Sliding Window Aggregation

Kanat Tangwongsan[#], Martin Hirzel[+], **Scott Schneider**[+]

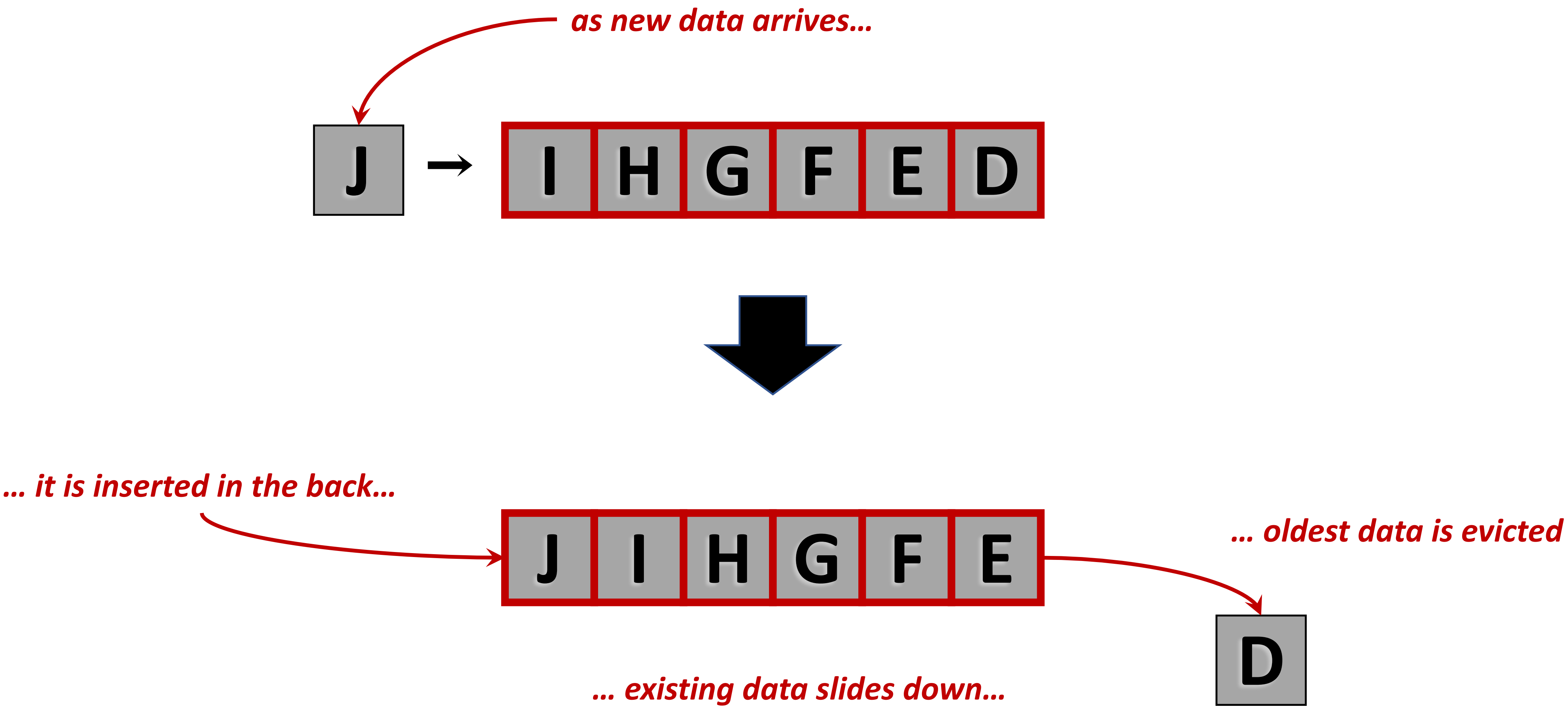
[#]*Mahidol University International College*
[+]*IBM T. J. Watson Research Center*

Windows on data streams




processing such infinite streams of data
requires defining a window on it
back
front
…
…

Sliding windows



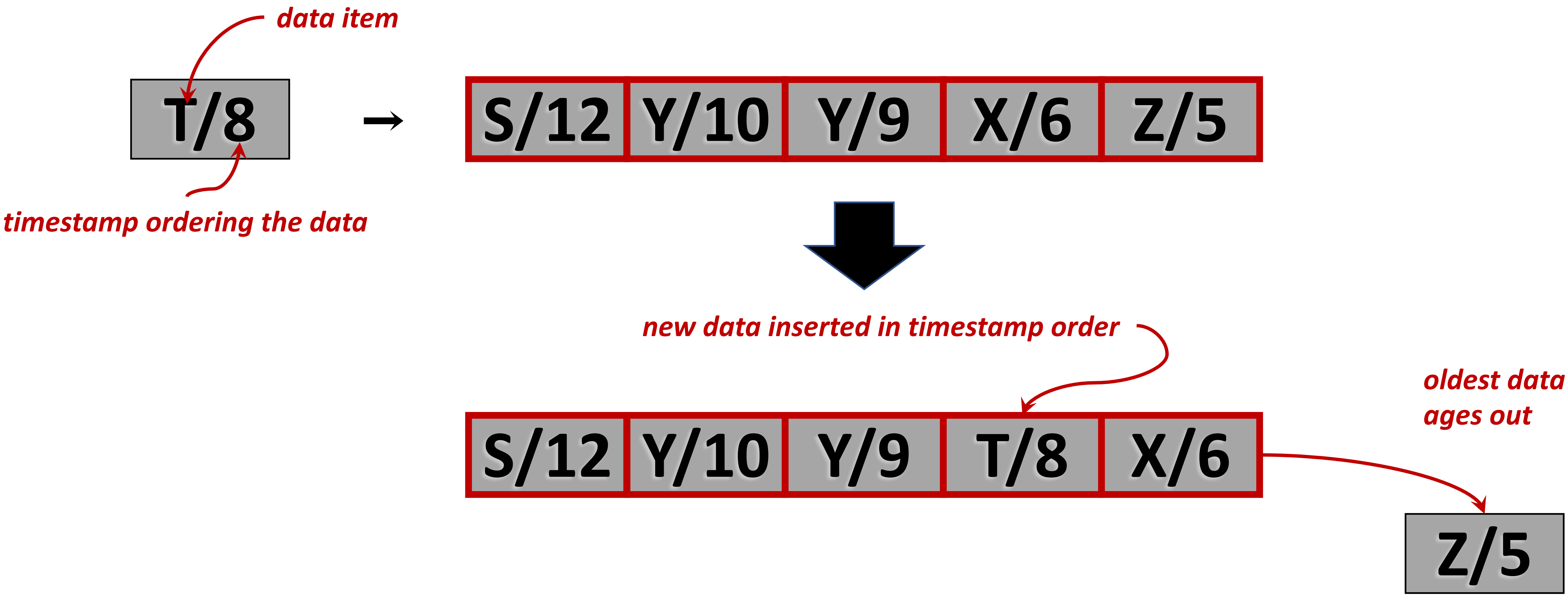
as new data arrives…

… it is inserted in the back…

… oldest data is evicted

… existing data slides down…

Real data tends to have *timestamps*



data item

T/8

timestamp ordering the data

→

S/12 Y/10 Y/9 X/6 Z/5

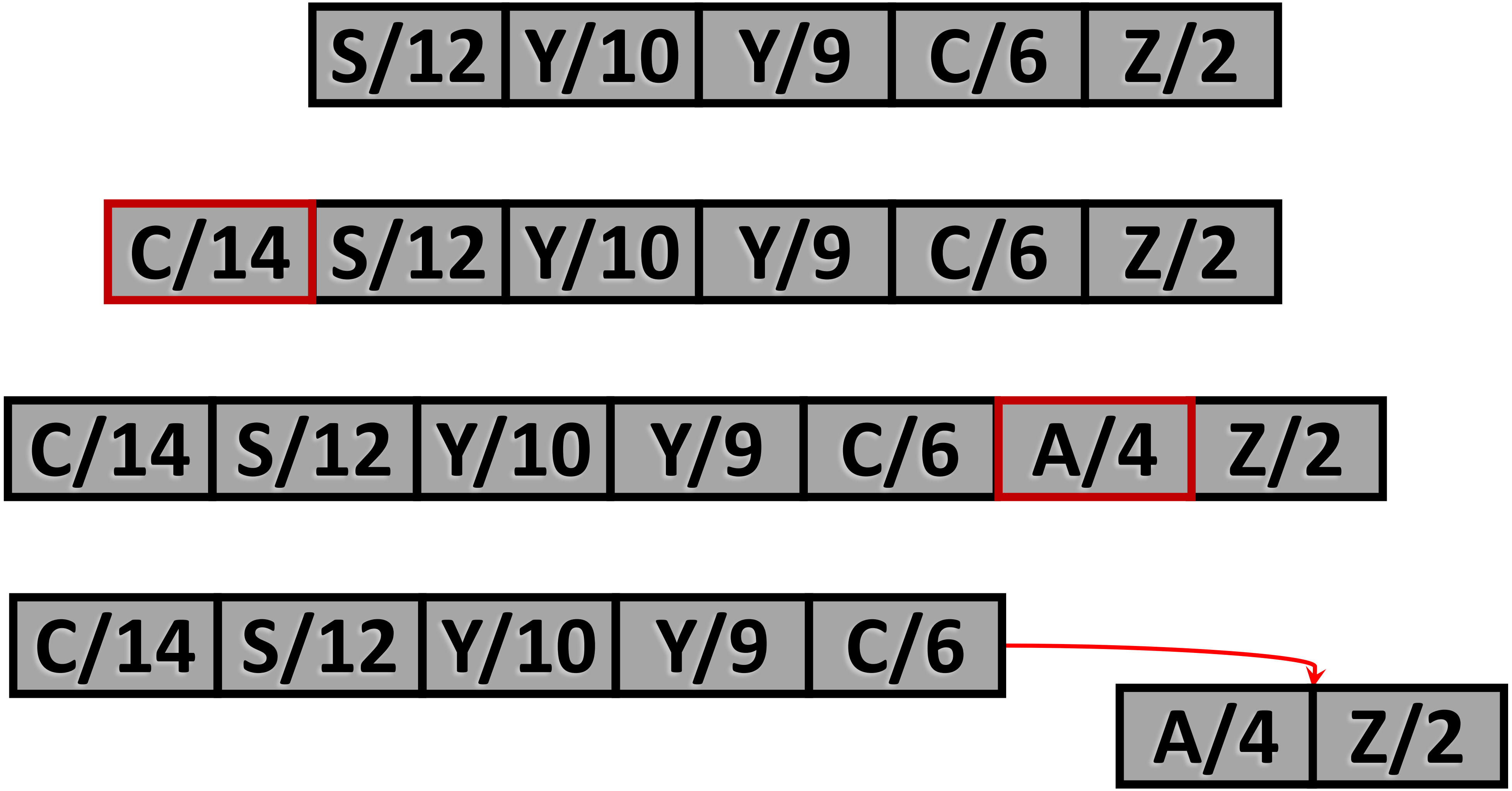
new data inserted in timestamp order

S/12 Y/10 Y/9 T/8 X/6

oldest data ages out

Z/5

A query on the window is an *aggregation*



S/12 | Y/10 | Y/9 | C/6 | Z/2

min: C
mincount: 1

C/14 | S/12 | Y/10 | Y/9 | C/6 | Z/2

min: C
mincount: 2

C/14 | S/12 | Y/10 | Y/9 | C/6 | A/4 | Z/2

min: A
mincount: 1

C/14 | S/12 | Y/10 | Y/9 | C/6

A/4 | Z/2

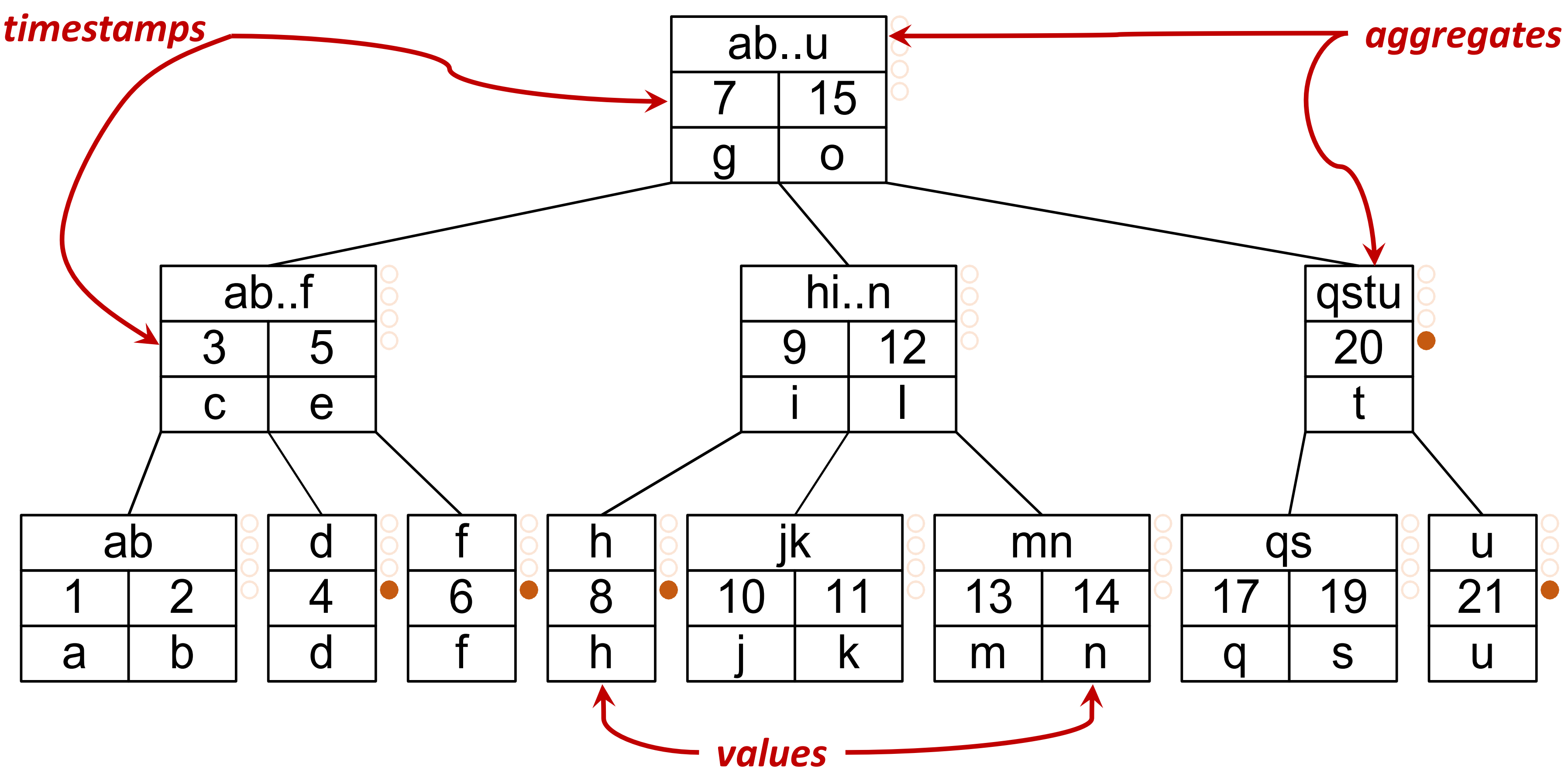
min: C
mincount: 2

Problem statement

- We want a data structure for a sliding window that can:
  - **insert** data items with timestamps that are out-of-order by distance *d* in amortized *O(*log *d)*, reducing to *O(*1*)* when *d*=0
  - **evict** data items based on timestamps in amortized *O(*log *d)*, reducing to *O(*1*)* when *d*=0
  - **query** the aggregations on the window in worst-case *O(*1*)*
- How?
  - B-Trees! (heavily modified)

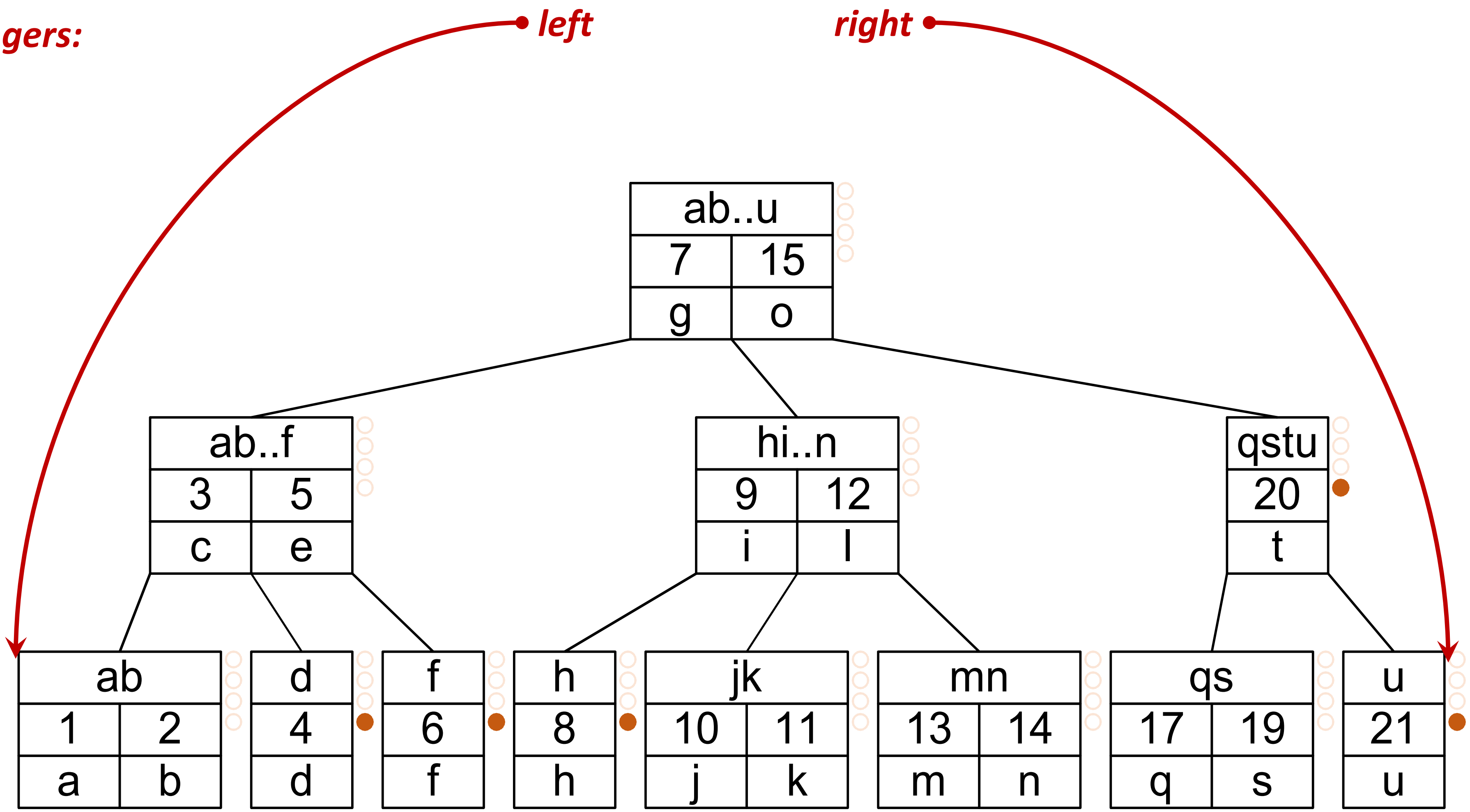
FiBA: Finger B-Tree Aggregator

start with a B-Tree modified with aggregates:



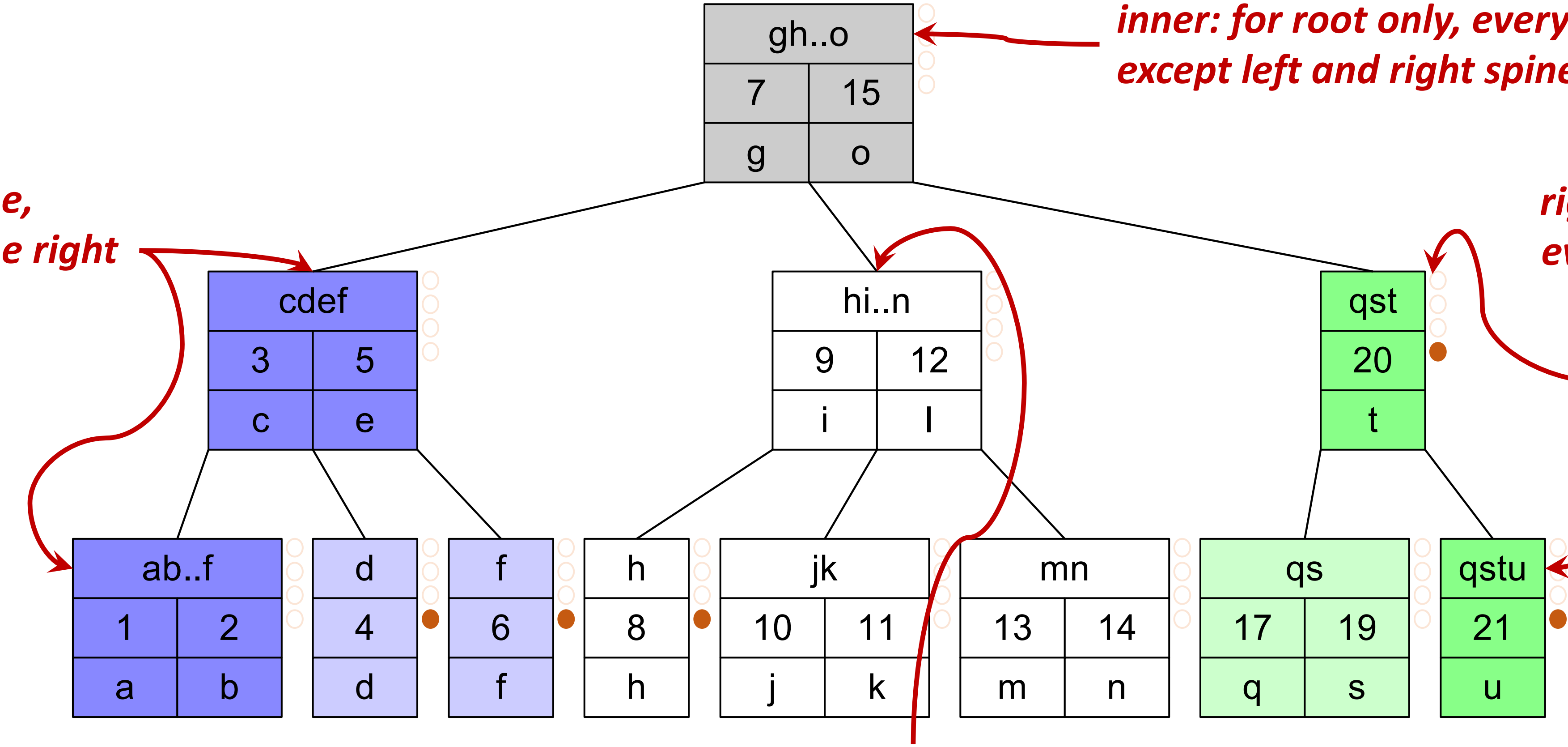
FiBA: Finger B-Tree Aggregator

add fingers:



FiBA: Finger B-Tree Aggregator

define position-aware aggregates:



inner: for root only, everything below except left and right spines

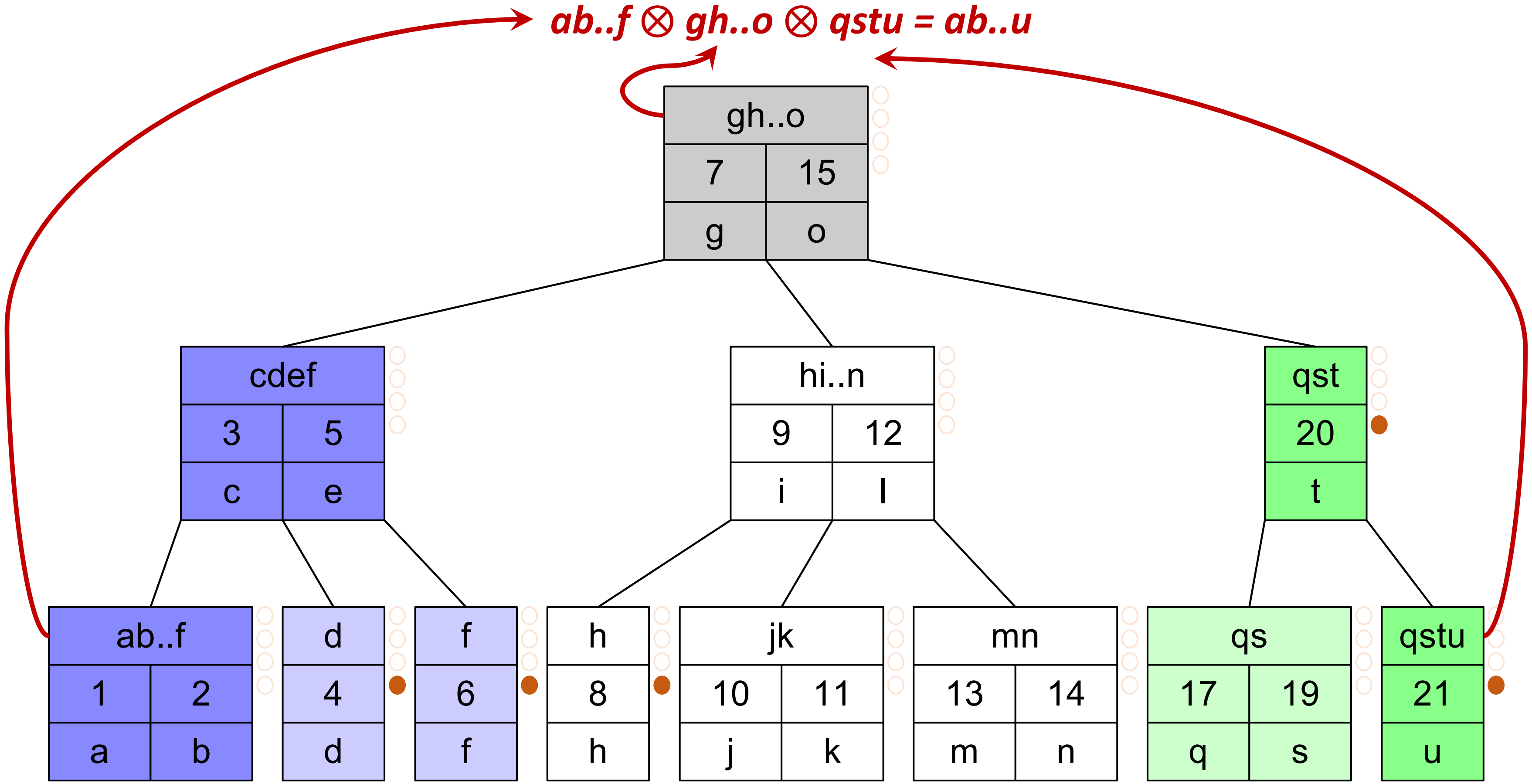
left: for left spine, everything to the right and above

right: for right spine, everything to the left and above

up: for middle, everything below

FiBA: Finger B-Tree Aggregator

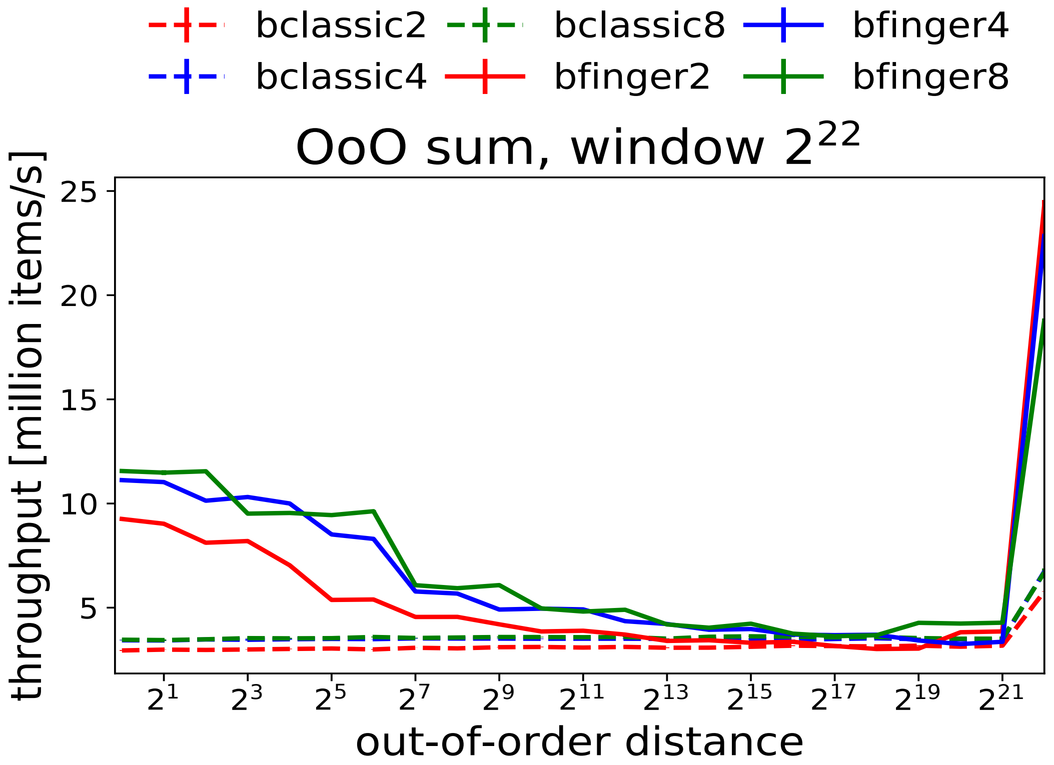
answer queries by combining left finger, root and right finger:

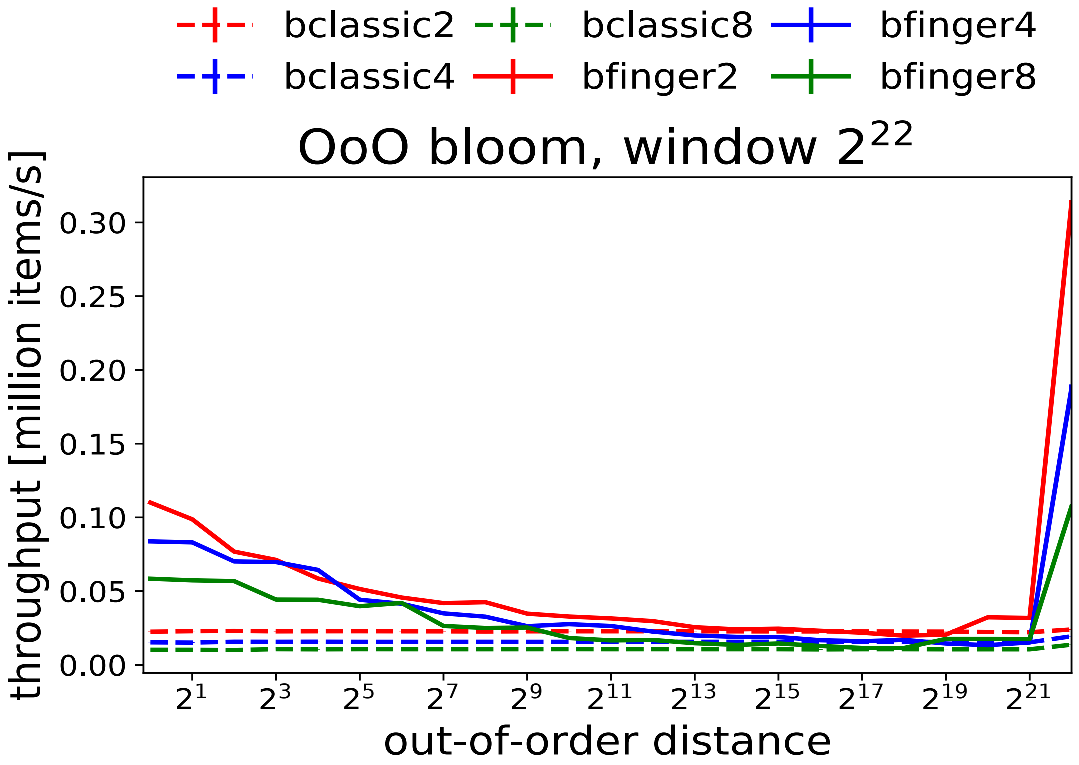


FiBA: Intuition for why it works

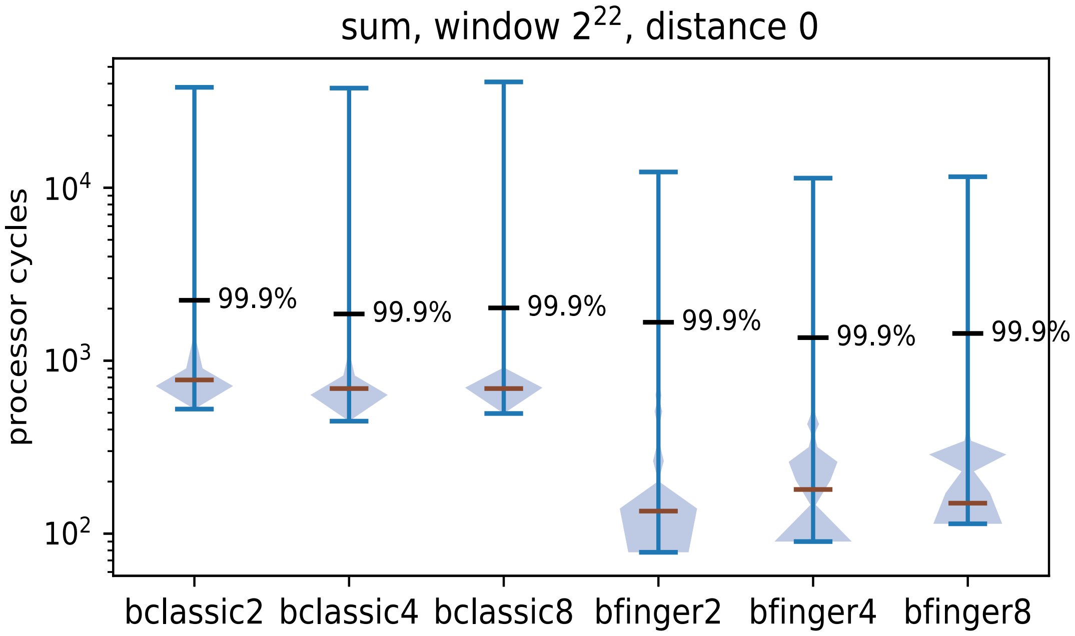
- **Fingers** allow *O(*1*)* time access to oldest and youngest
  - without fingers, searching for a data item would be standard *O(*log *n)*
  - but we are dealing with a time-based window, where we are biased towards inserting at the young end and evicting from the old end
- Specially defined aggregates **shield** sections of the tree
  - updating a value in one section of the tree is unlikely to cause repairs to aggregates elsewhere
- Choice of min and max arity plus lazy splitting and merging **avoids** unnecessary tree rebalancing

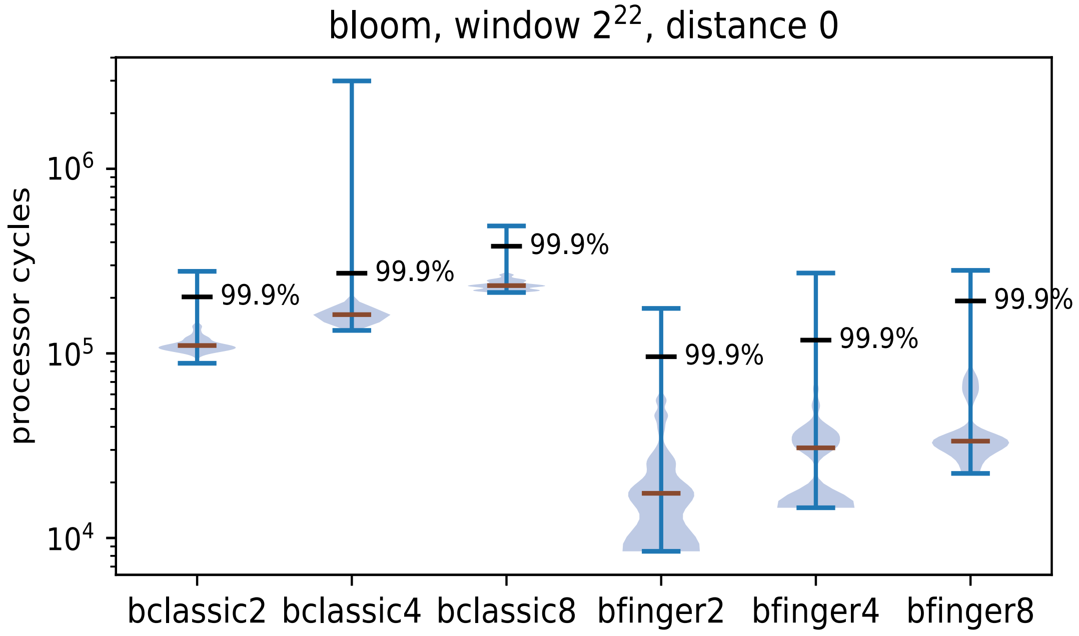
Out-of-Order Throughput



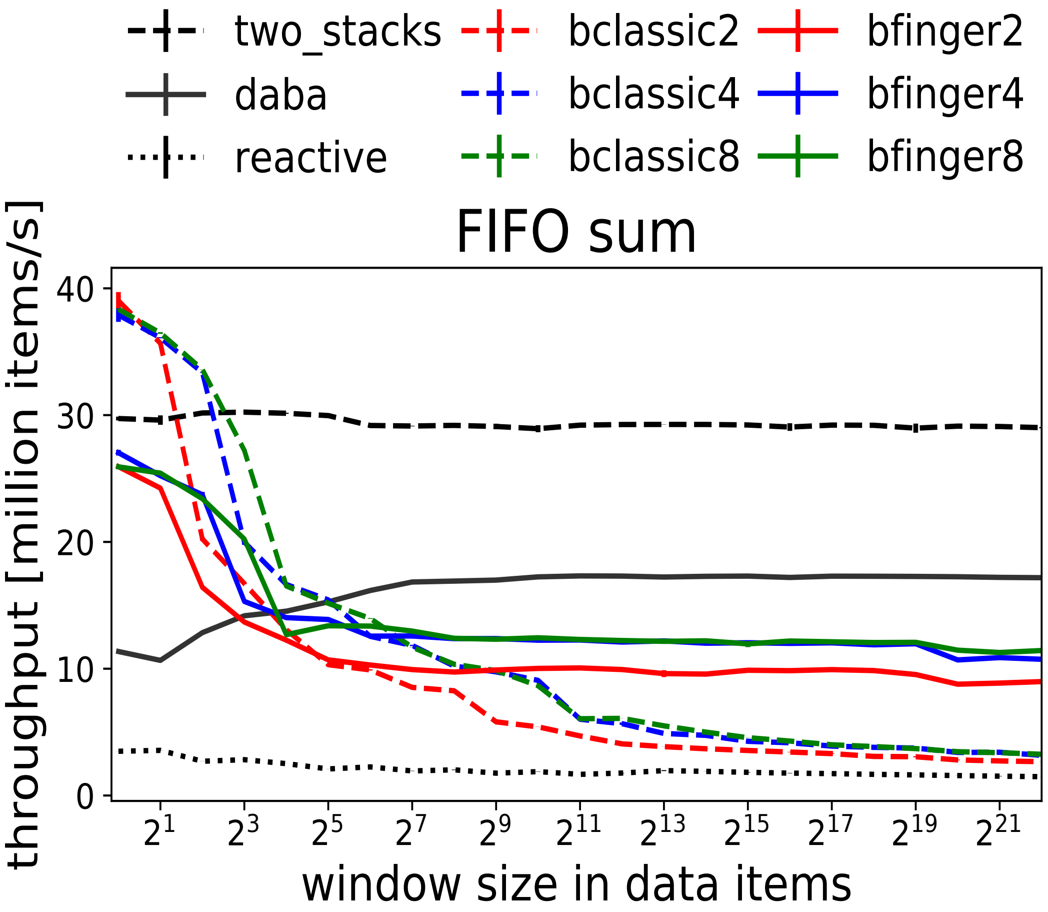


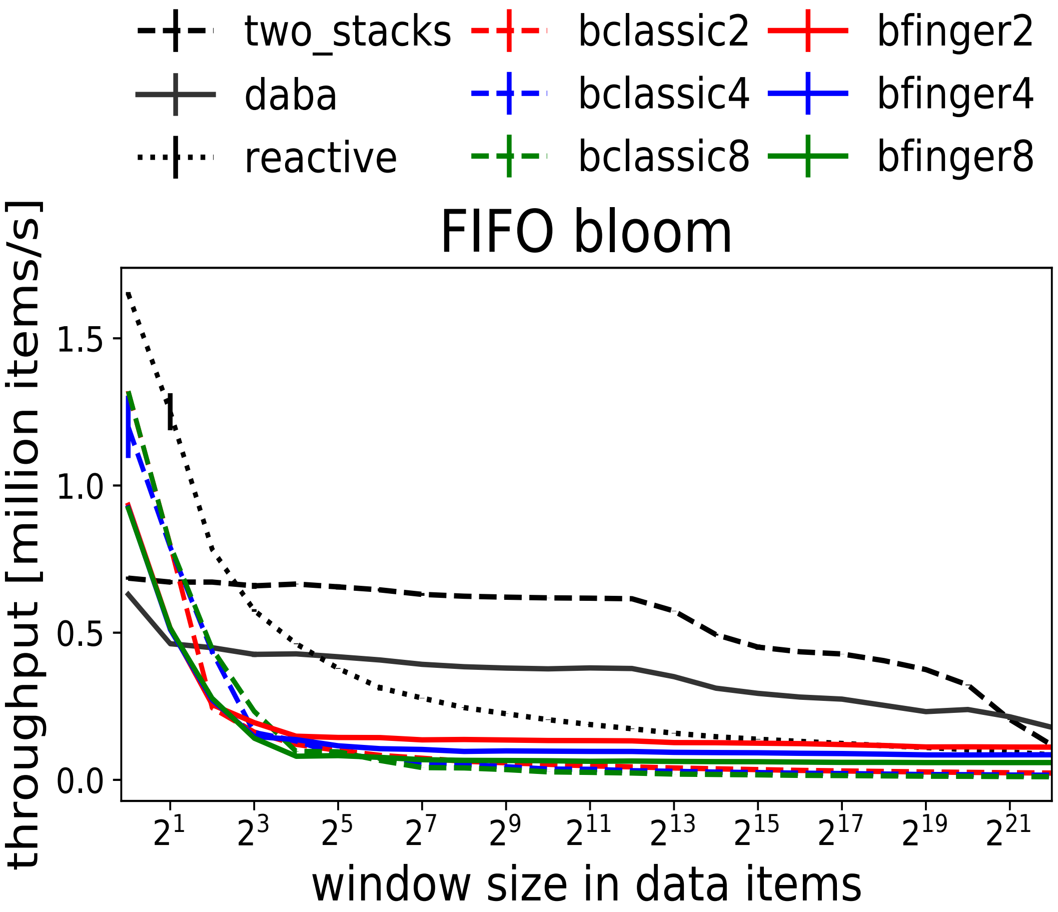
In-Order Latency





In-Order Throughput

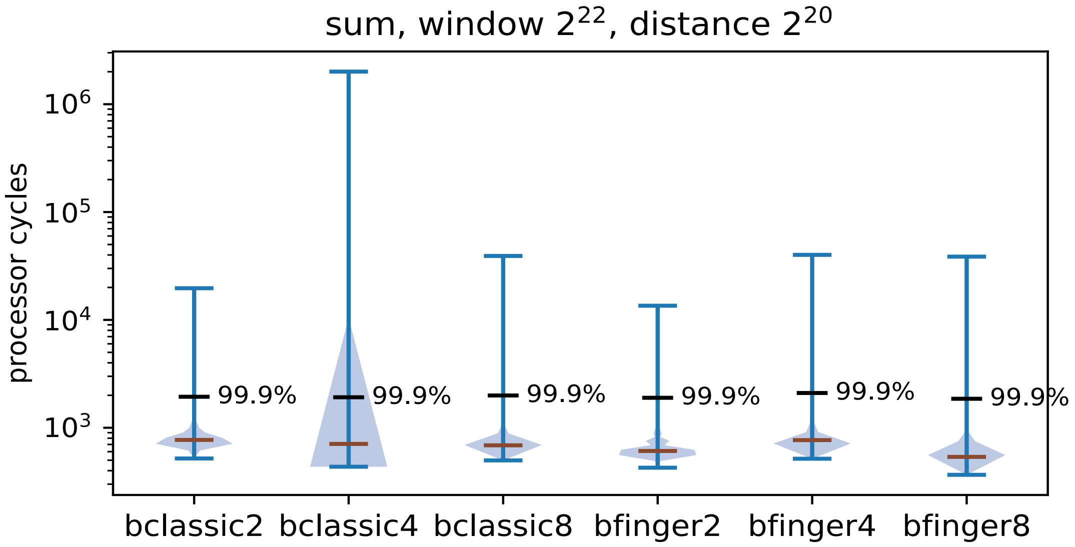


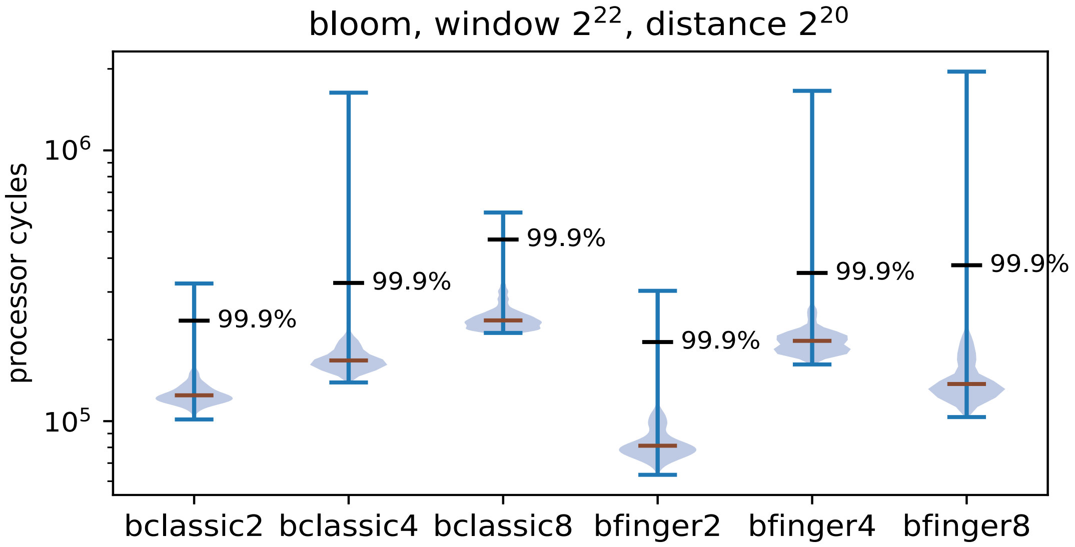


Questions?

Backup

Out-of-Order Latency


sum, window 2^{22}, distance 2^{20}
processor cycles
10^6
10^5
10^4
10^3
bclassic2
bclassic4
bclassic8
bfinger2
bfinger4
bfinger8


bloom, window 2^{22}, distance 2^{20}
processor cycles
10^6
10^5
bclassic2
bclassic4
bclassic8
bfinger2
bfinger4
bfinger8