Practically and Theoretically Efficient Garbage Collection for Multiversioning

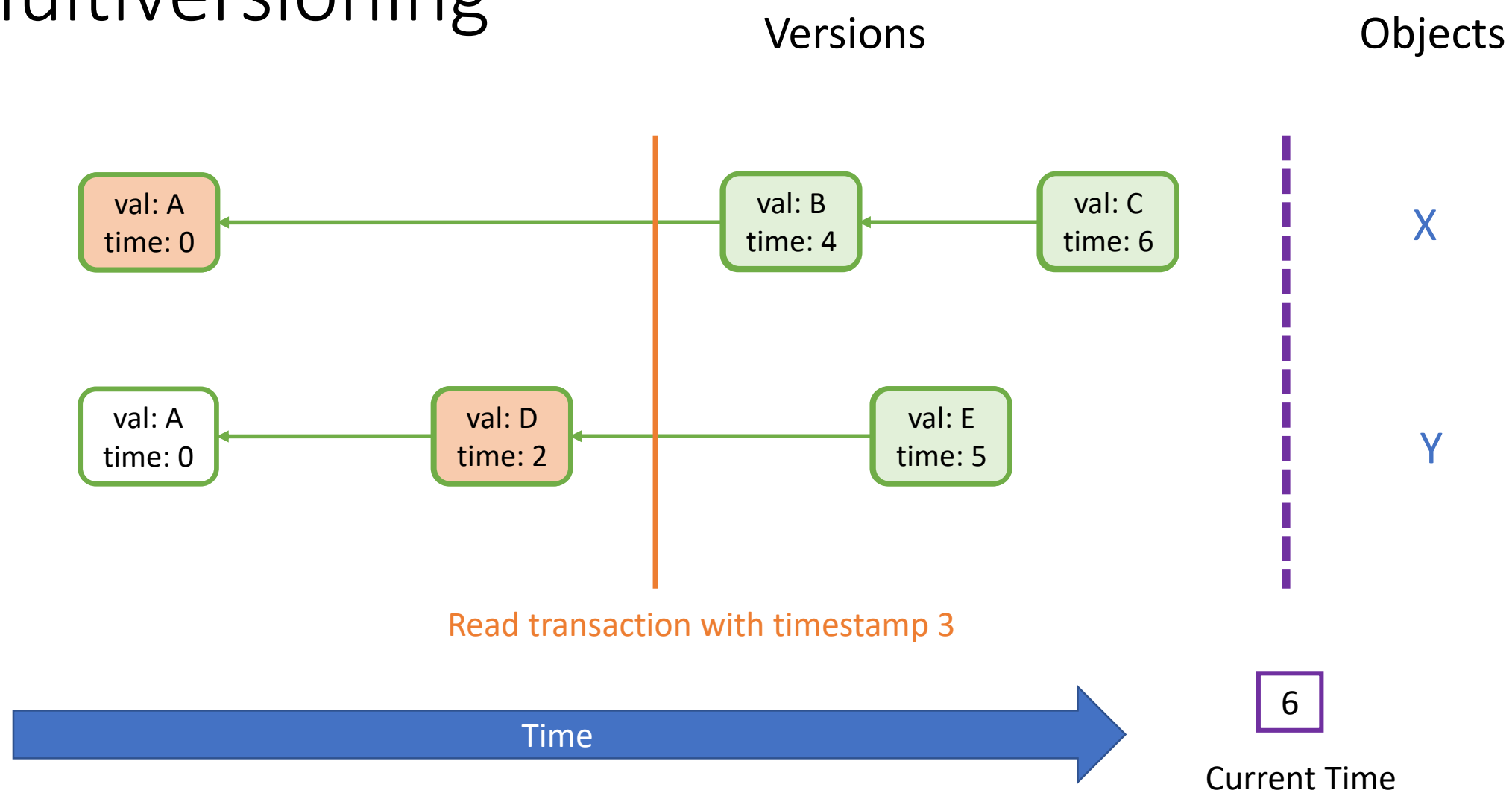
Yuanhao Wei, Guy E. Blelloch, Panagiota Fatourou, and Eric Ruppert

Motivation

- Multiversioning widely used:
  - Database systems
  - Software Transactional Memory **[Fernandes et al. PPoPP'11] [Lu et al. DISC'13]**
  - Concurrent data structures **[Fatourou et al. SPAA'19] [Wei et al. PPoPP'21] [Kobus et al. PPoPP'22] [Sheffi et al. OPODIS'22]**

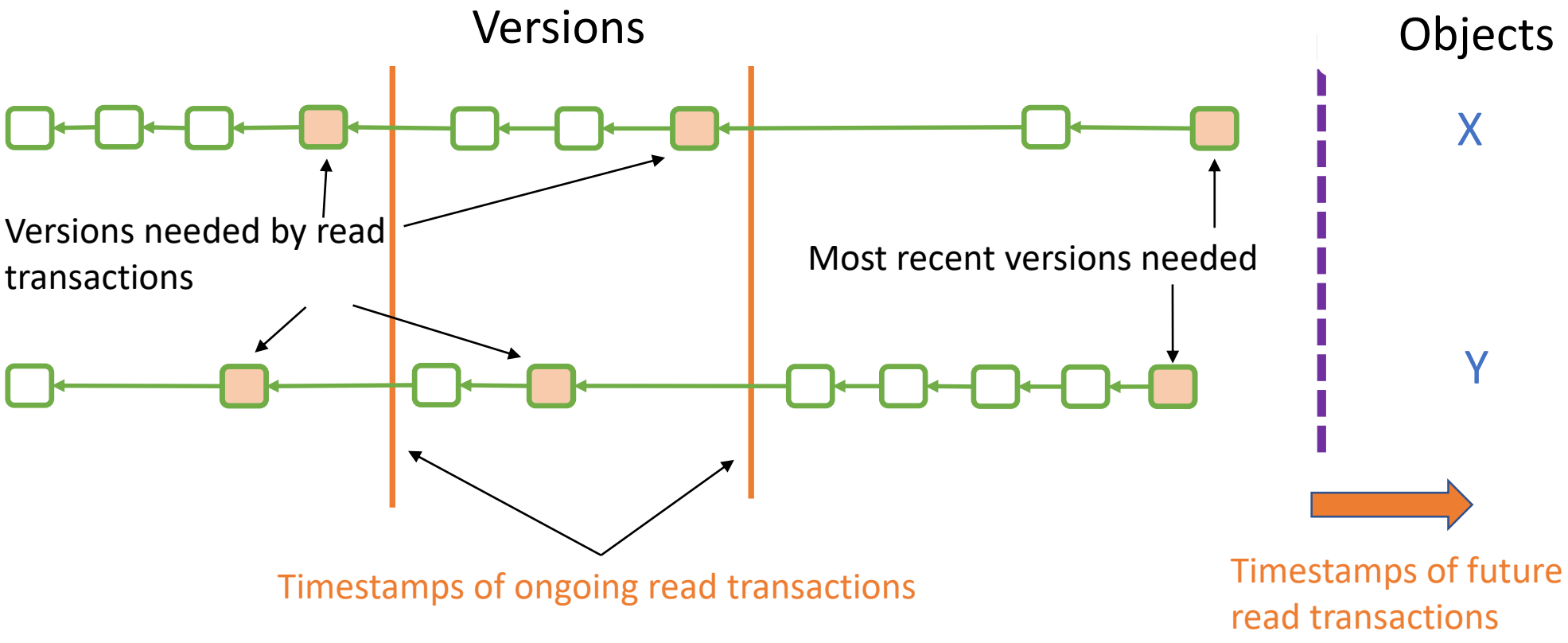
- High space usage ⇒ obsolete versions must be reclaimed
  - Observed to be a bottleneck in modern database systems **[Lee et al. SIGMOD'16] [Böttcher et al. VLDB'19]**

Multiversioning



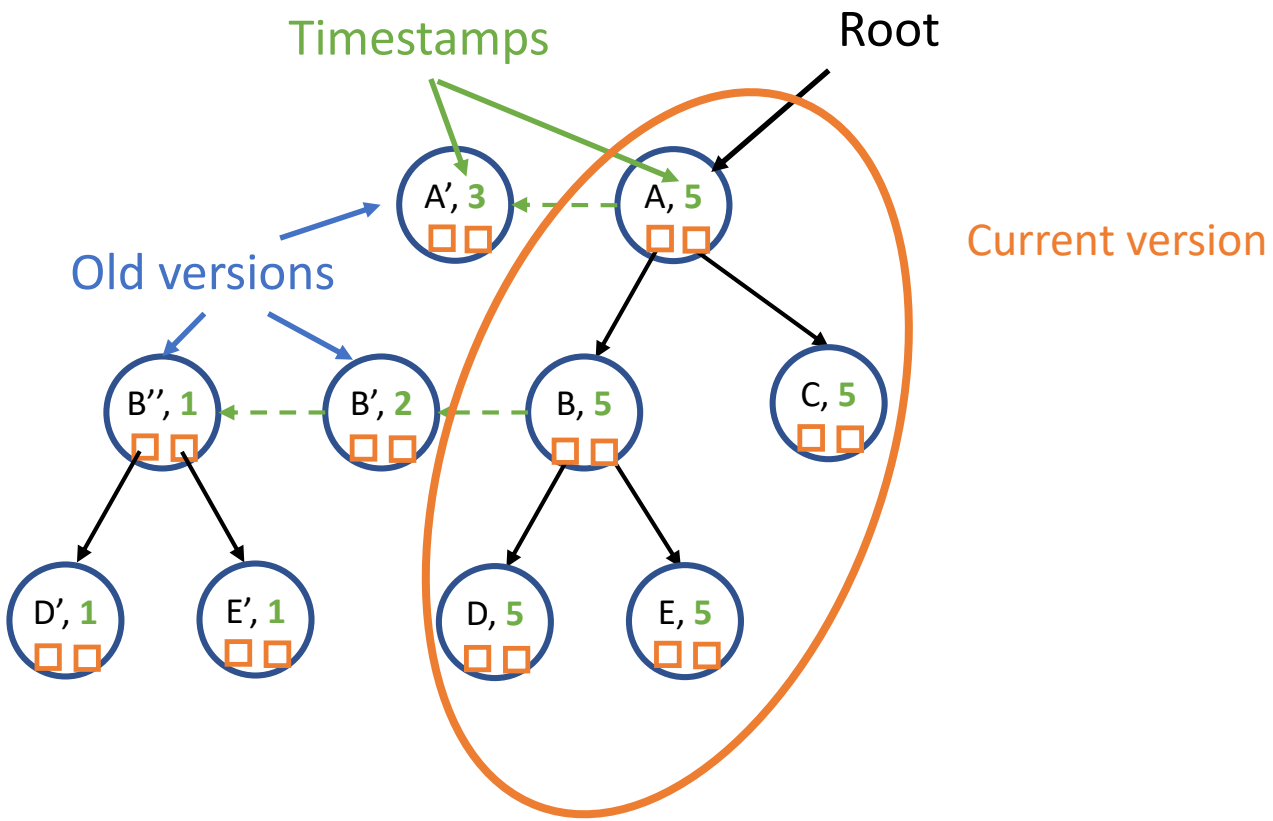
Multiversion Garbage Collection (MVGC)

Challenge: Identify and remove obsolete versions



Multiversion Data Structures

- Lock-free Multiversion Tree [PPoPP'21]
- Supports linearizable range queries
- Used in our experiments



Previous Work (MVGC)

Epoch Based Solutions

HyPer [SIGMOD'15]

VCAS [PPoPP'21]

Bundled References [PPoPP'22]

...

Scan Based Solutions

GMV [DISC'13]

Hana [SIGMOD'16]

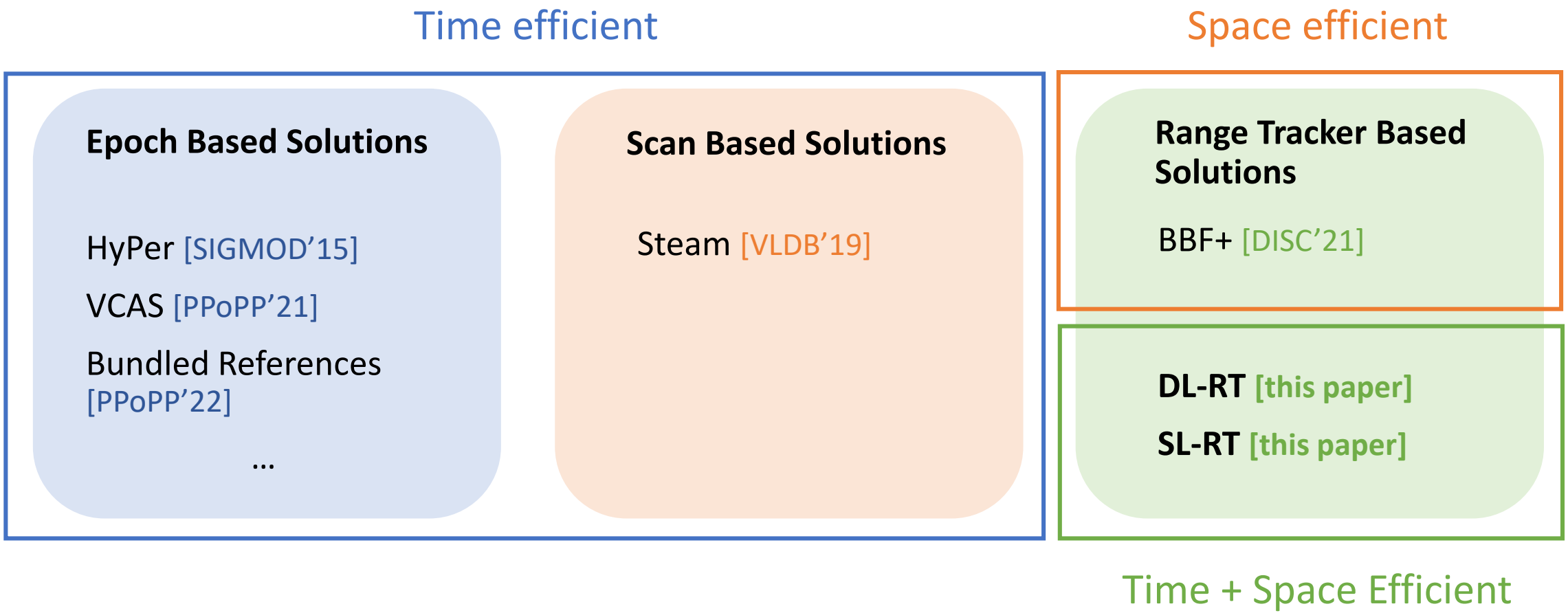
Steam [VLDB'19]

Range Tracker Based Solutions

BBF+ [DISC'21]

Our Contributions

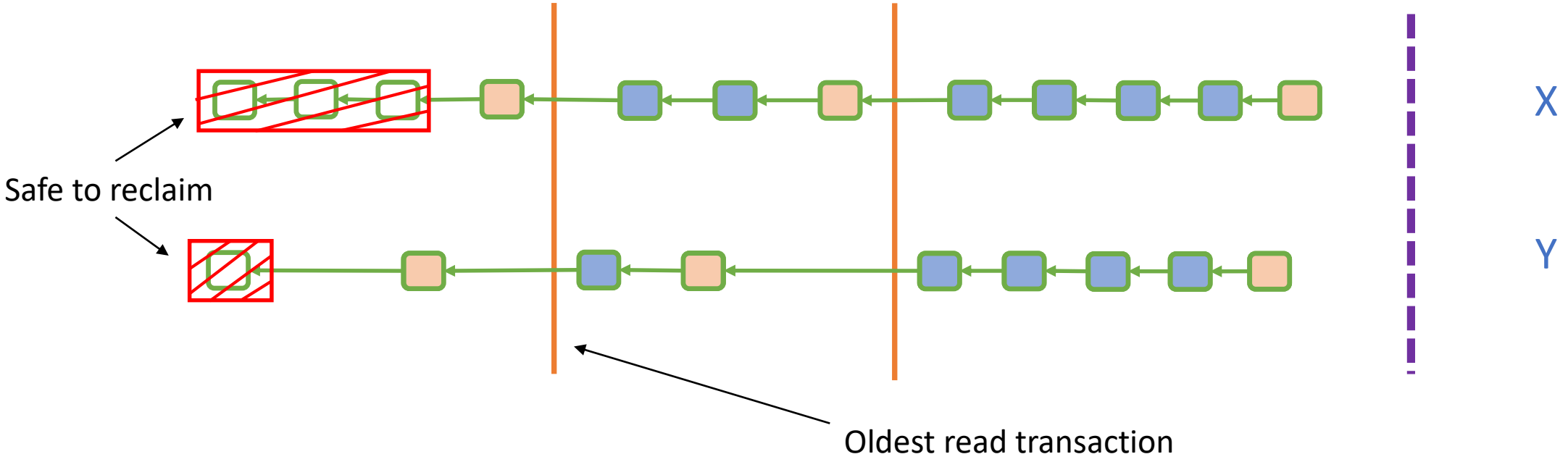
- 2 new MVGC schemes + apples-to-apples comparison



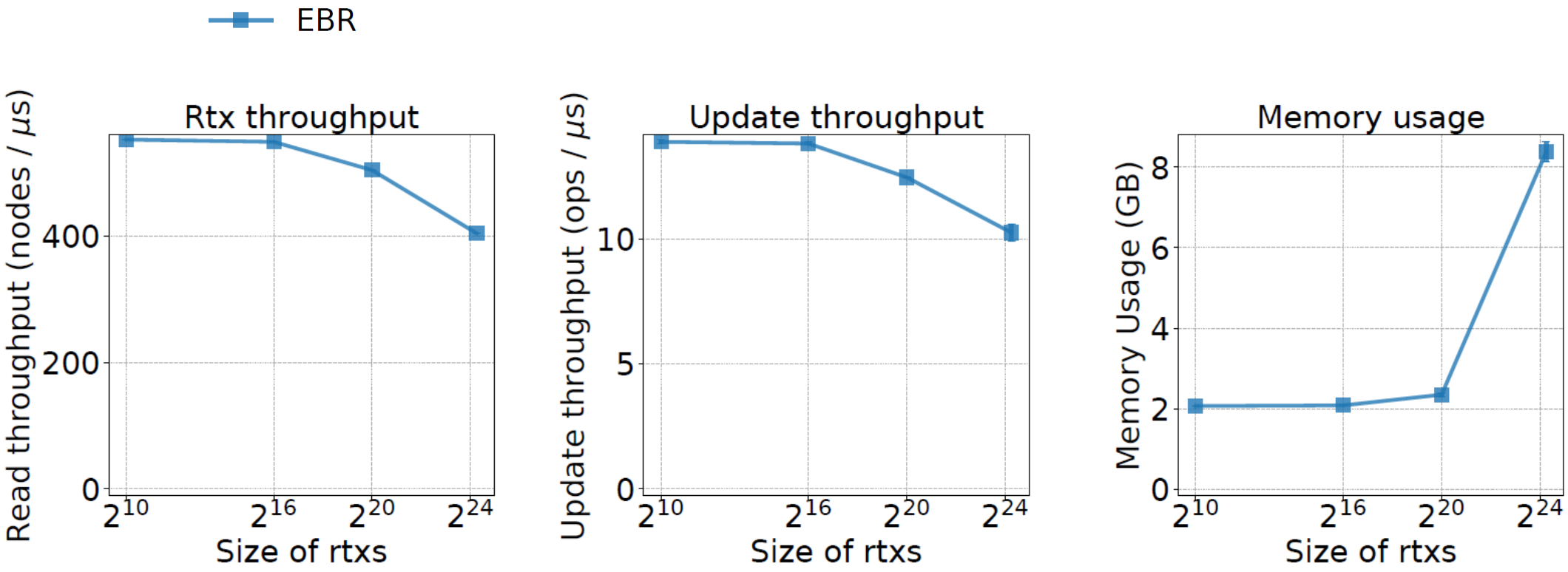
Epoch Based Solutions

- Track oldest ongoing read transaction
- Reclaim any version overwritten before it

- Pros: Fast, easy to implement
- Cons: unable to reclaim intermediate versions



Experiment Results (Java): Epoch Based

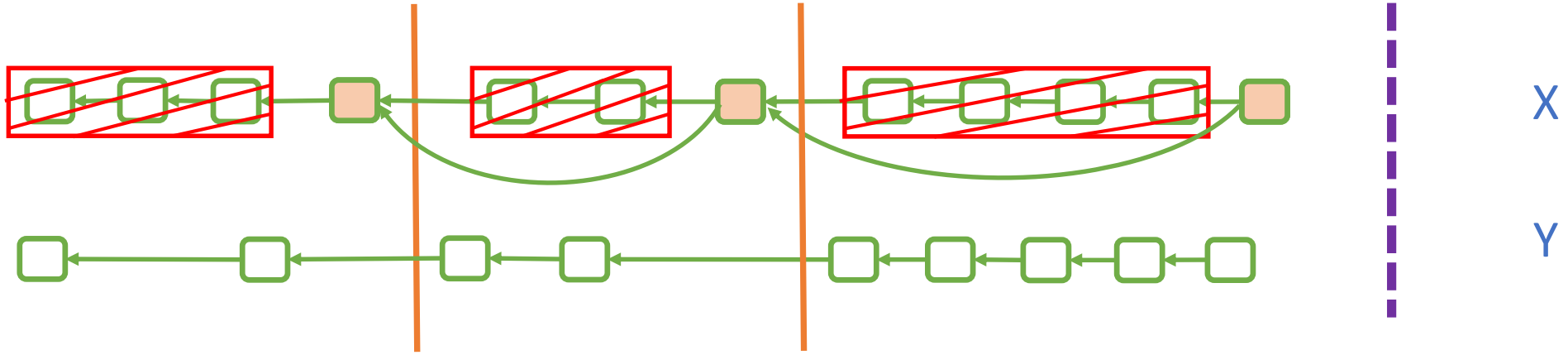


Workload: Tree initialized with 10M keys
80 range query threads
40 update threads

rtx = read transaction

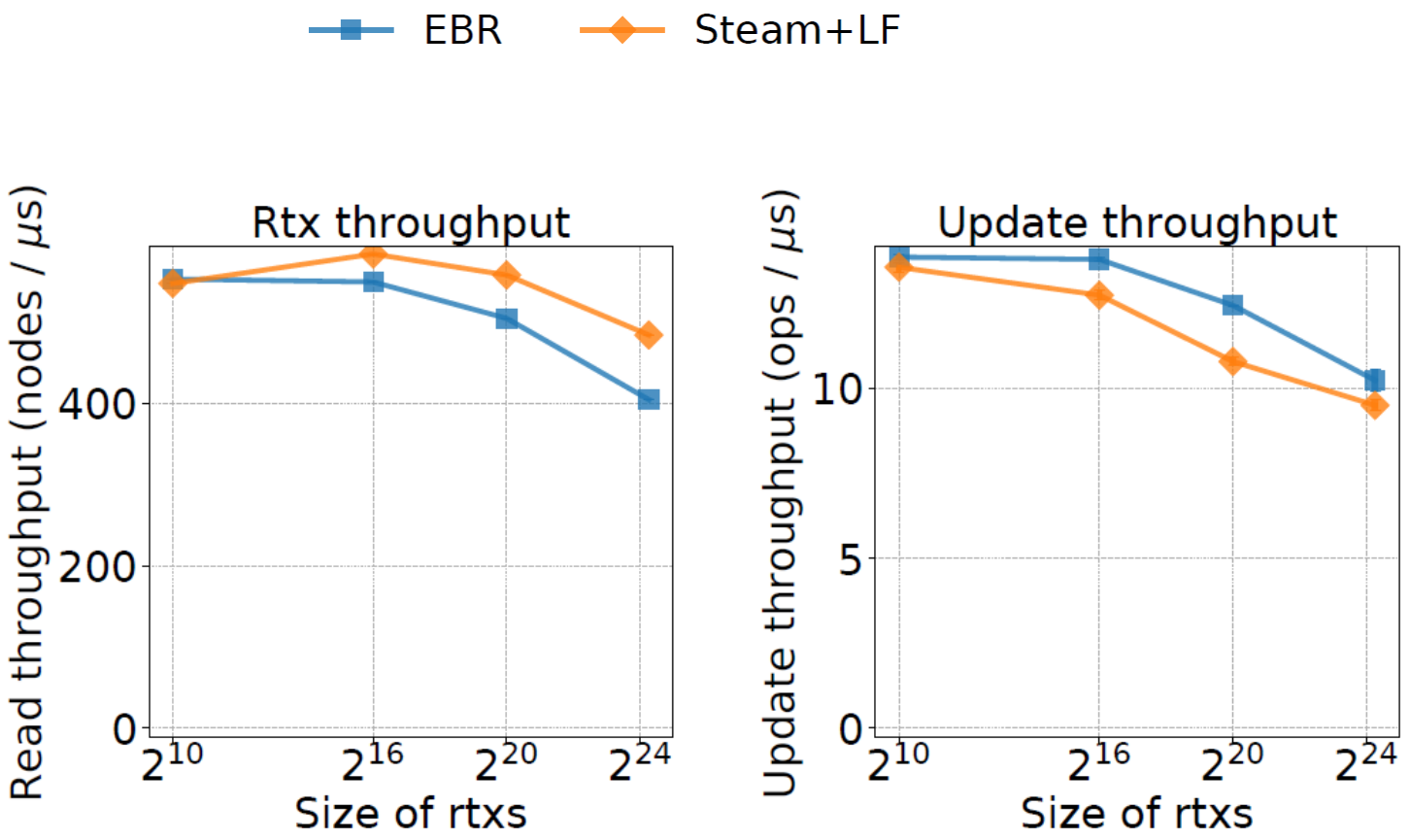
Scan Based Solutions

- Steam [VLDB'19]: Whenever a new version is added -> scan and compact its version list



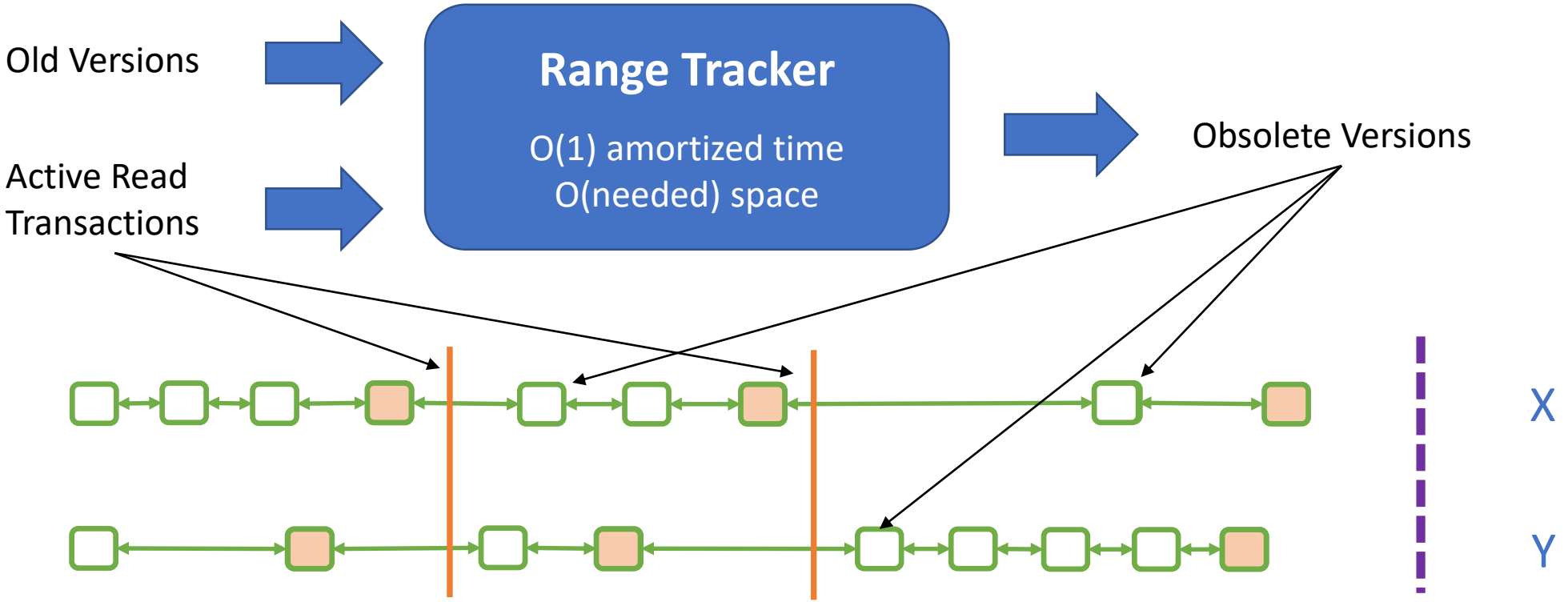
- Locks entire version list when compacting
- We improved Steam with a new lock-free singly-linked version list (Steam+LF)

Experiment Results: Steam+LF



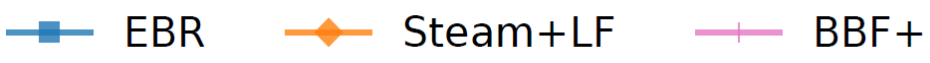
Range Tracker Based Solutions

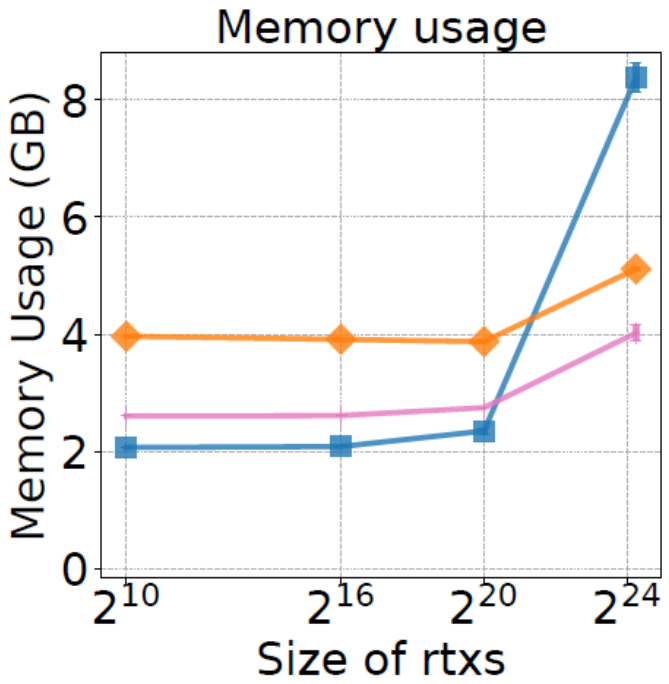
BBF+ [Ben-David et al., DISC'21]



BBF+ uses doubly-linked version lists to allow removal of obsolete versions

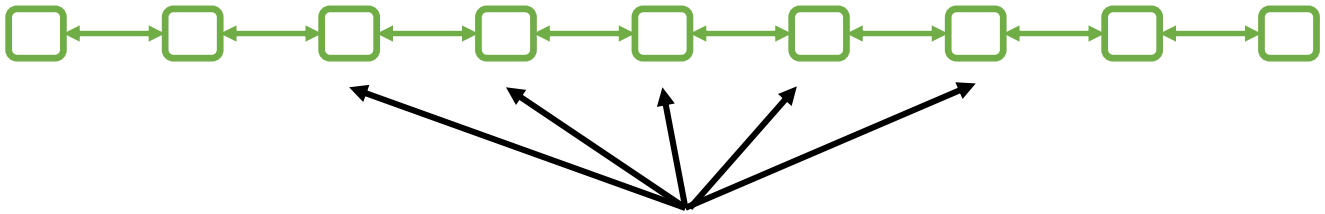
Experiment Results: BBF+





New MVGC Schemes

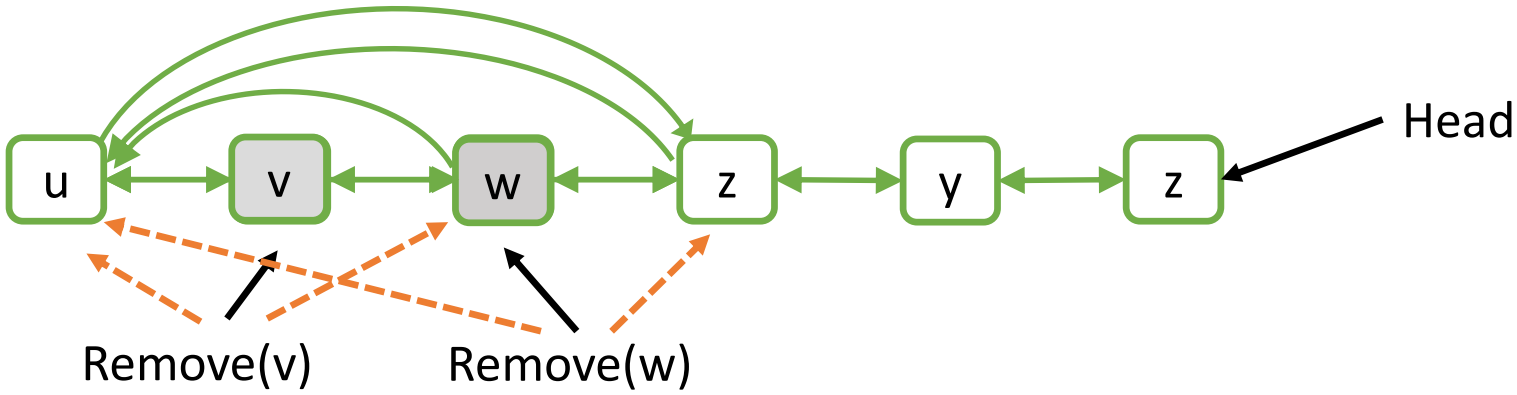
- Use range tracker to get good space efficiency
- Time efficiency: BBF+ is over optimized for worst-case



- DL-RT: Range tracker + new doubly-linked version list
- SL-RT: Range tracker + scan + new singly-linked version list

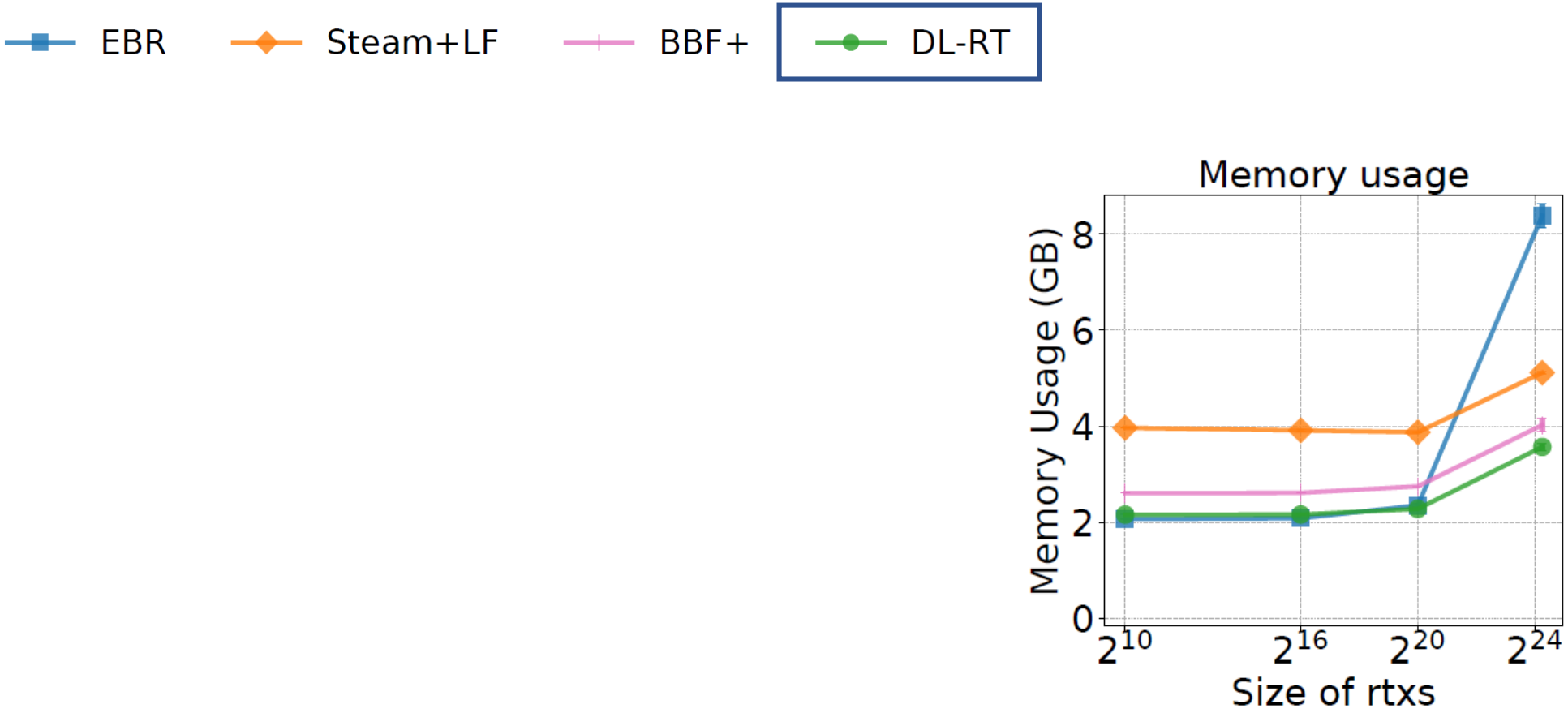
Practical Doubly-Linked Version List

- To remove a node:
  1. Mark it as deleted by setting a flag
  2. Traverse in both directions until you find an unmarked node on each side
  3. Make them point to each other (using compare_and_swap)

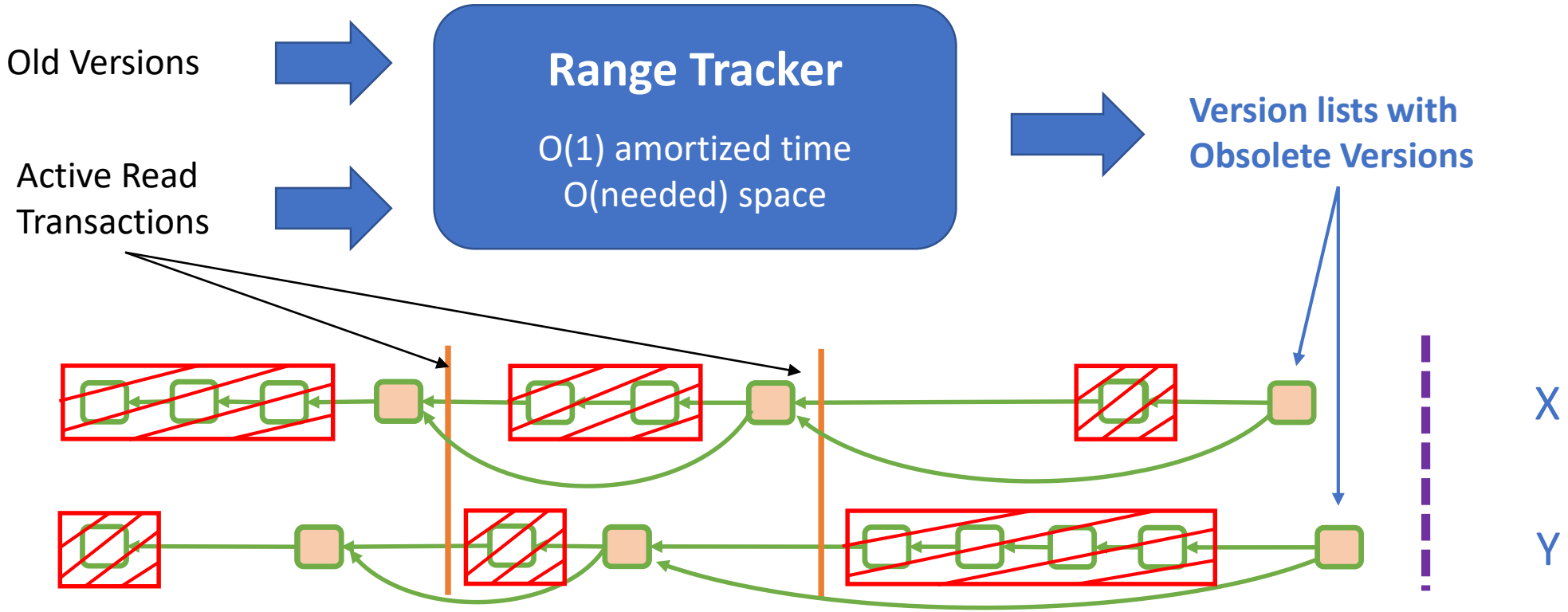


- Algorithm simple, correctness subtle

Practical Doubly-Linked Version List (DL-RT)



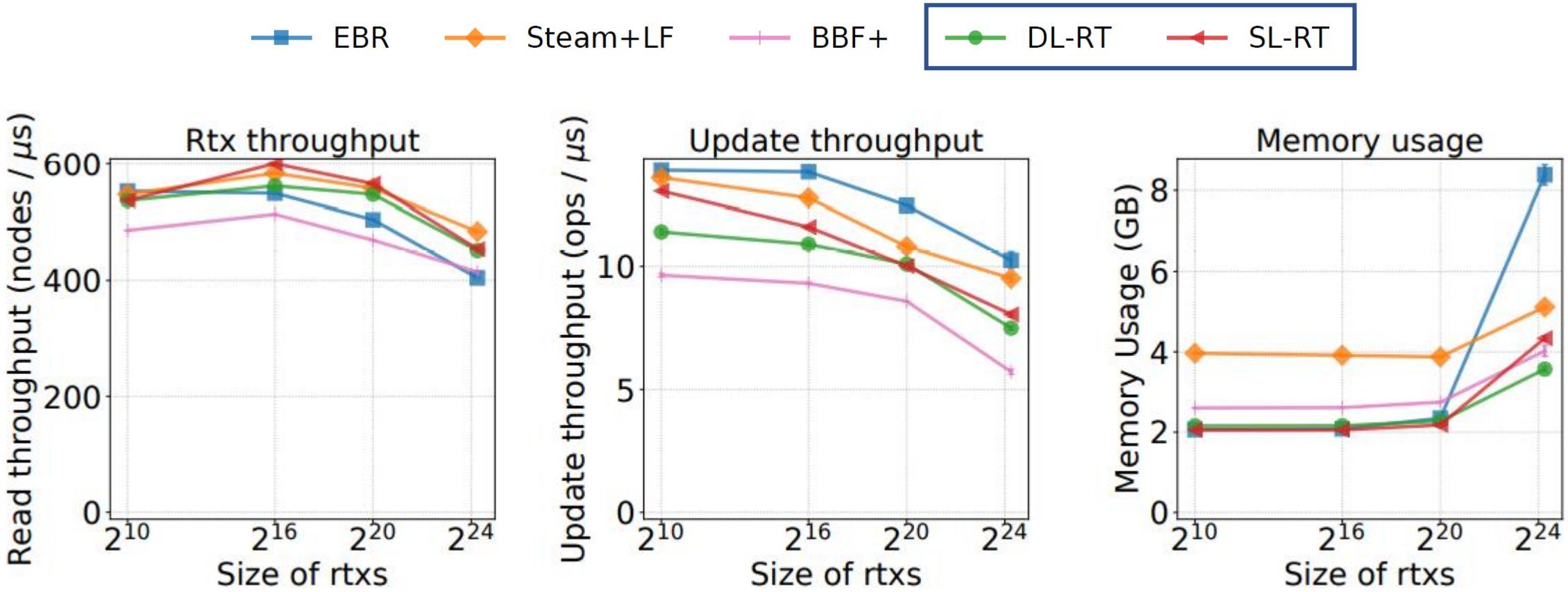
Range Tracker + Scan Based Solutions



New lock-free, singly-linked version list

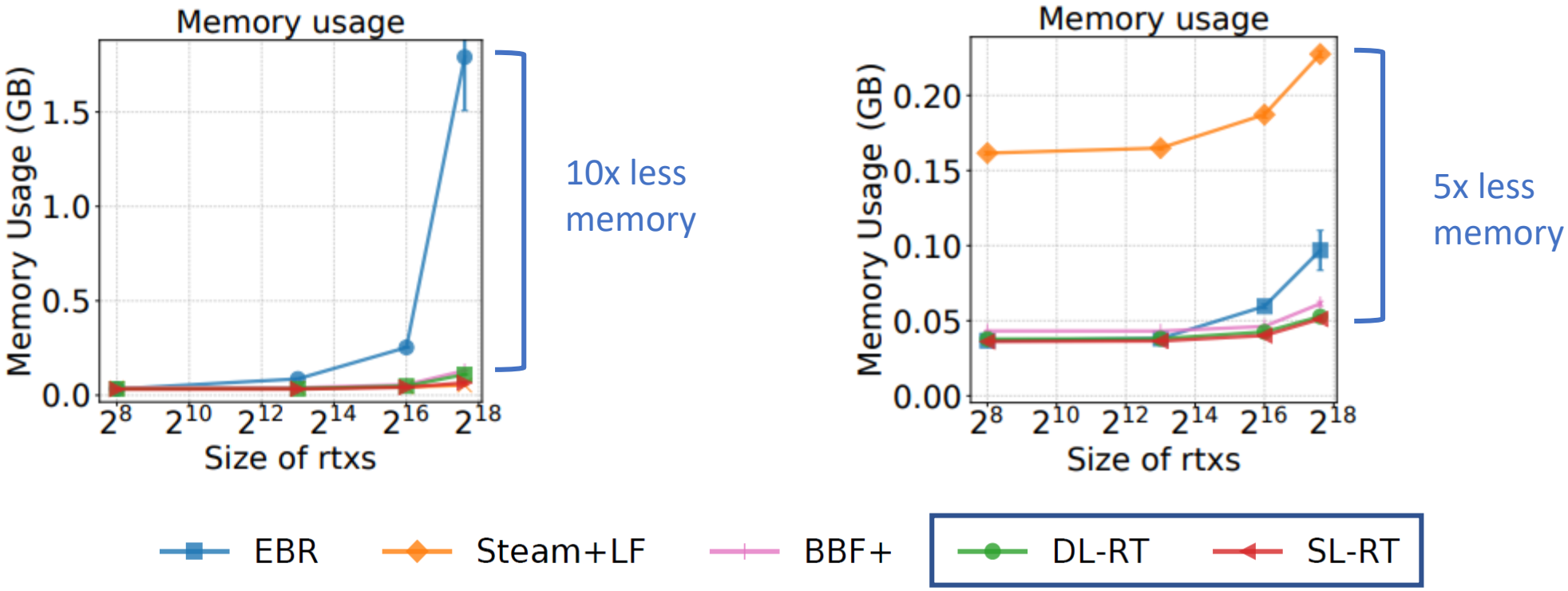
Range Tracker + Scan (SL-RT)

New MVGC schemes



Worst-case Space Bounds

- DL-RT and SL-RT maintain O(needed) versions => robust to workload
- Better worst-case memory usage than EBR and Steam



Conclusion

- Two new MVGC schemes: DL-RT and SL-RT
  - Fast and space efficient in practice
  - Strong space bounds in theory
  - New lock-free doubly-/singly- linked version lists
- Experimental comparison between MVGC schemes
- Our code is available on GitHub: https://github.com/cmuparlay/ppopp23-mvgc