SNAPSHOT-FREE, TRANSPARENT, AND ROBUST MEMORY RECLAMATION FOR LOCK-FREE DATA STRUCTURES

Ruslan Nikolaev and **Binoy Ravindran**

rnikola@vt.edu **binoy@vt.edu**

Systems Software Research Group

Virginia Tech, USA

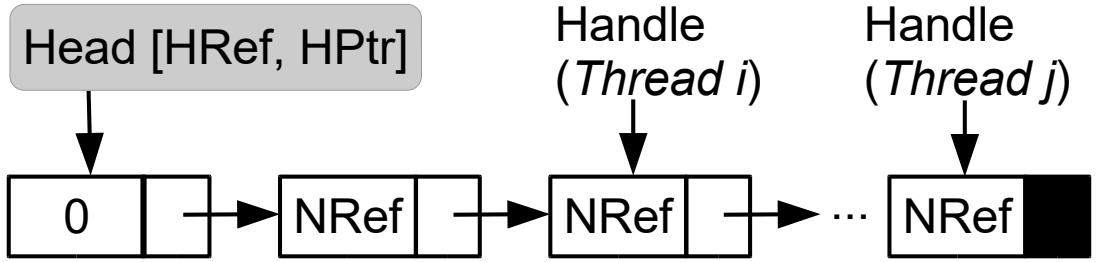
MEMORY RECLAMATION PROBLEM

- Concurrent programming is hard
  - Non-blocking (lock-free) data structures require special treatment of deleted memory objects
  - Garbage collectors are often impractical in C/C++ and lack suitable progress/performance properties

- Desirable properties for memory reclamation
  - ***Non-blocking progress***: avoid using locks
  - ***Robustness***: bounding memory usage even when threads are stalled or preempted
  - ***Transparency***: avoiding implicit assumptions about threads; threads can be created/deleted dynamically
  - ***Snapshot-freedom***: not taking snapshots of the global state to alleviate contention

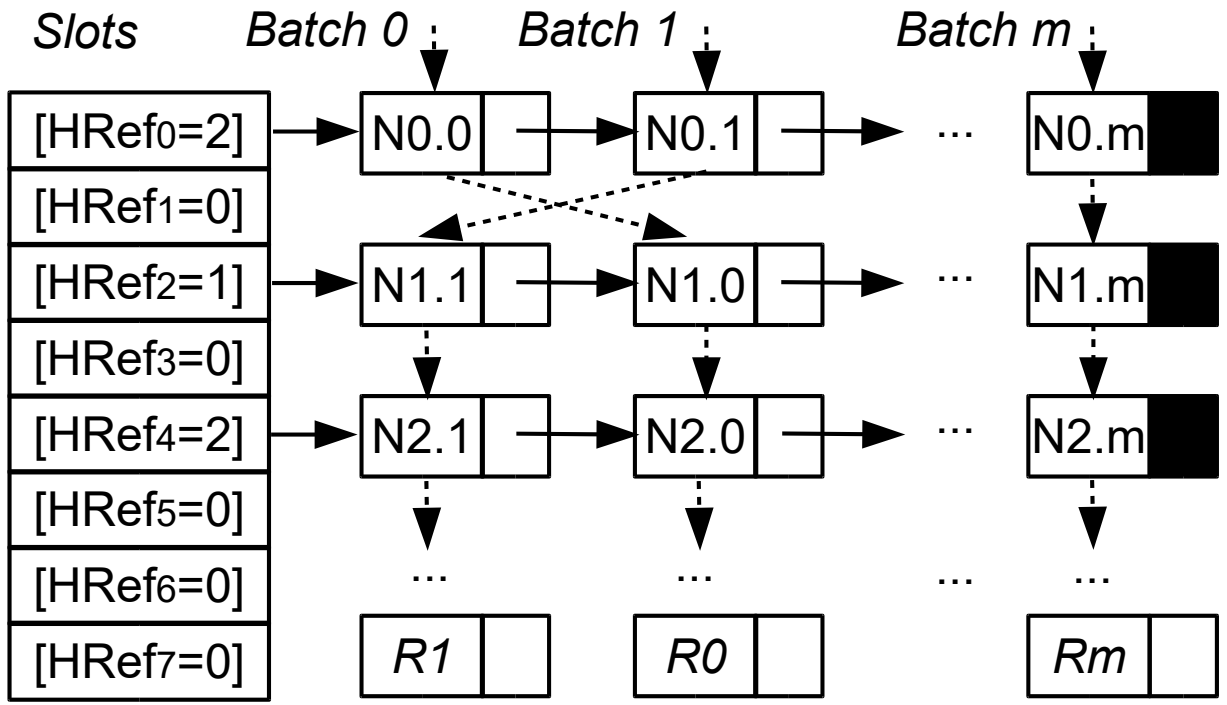
HYALINE: SINGLE LIST

- The main idea
  - Use ***special*** reference counting, which is triggered only when deleting objects
  - Deleted objects are appended to a global list
    - Each thread traverses a sublist from the very beginning of the list to the object pointed to by a special per-thread handle
    - The handle points to the part of the list when the thread started its operation



HYALINE: MULTIPLE LISTS

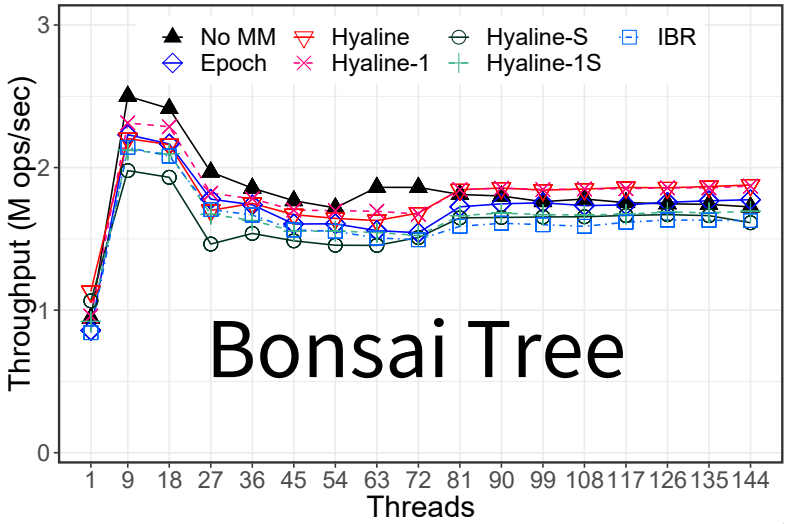
- The main idea
  - Use ***special*** reference counting, which is triggered only when deleting objects
  - Maintain multiple global lists of deleted objects to alleviate contention
    - Each list is used by a subset of threads
    - Delete an entire ***batch*** of objects
    - One reference counter for the entire batch

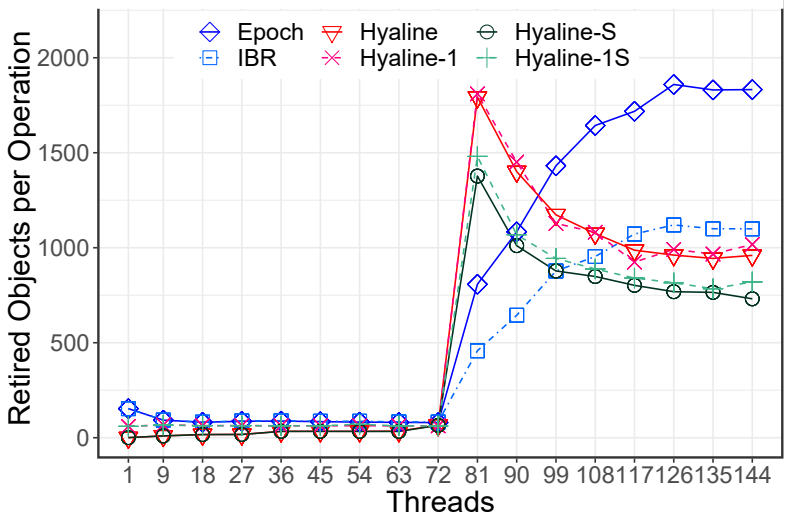


COMPARISON

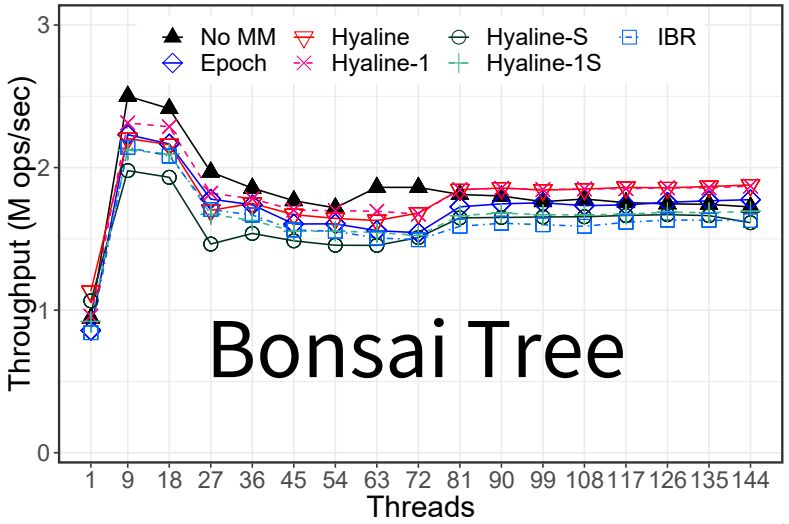
Scheme	Performance	Snapshot-Free	Robust	Transparent	Extra Memory	API complexity
Reference Counting	Very Slow	Yes	Yes	Partially (swap)	Double each pointer	Harder, Intrusive
Hazard Pointers	Slow	No	Yes	No (deletion)	1 word	Harder
Epoch Based Reclamation	Fast	Yes	No	No (deletion)	1 word	Very Easy
Hazard Eras	Medium	No	Yes	No (deletion)	3 words	Harder
Interval Based Reclamation	Fast	No	Yes	No (deletion)	3 words	Medium
Hyaline	Fast	Yes	No	Yes	3 words	Very Easy
Hyaline-1	Fast	Yes	No	Almost	3 words	Very Easy
Hyaline-S	Fast	Yes	Yes	Yes	3 words	Medium
Hyaline-1S	Fast	Yes	Yes	Almost	3 words	Medium

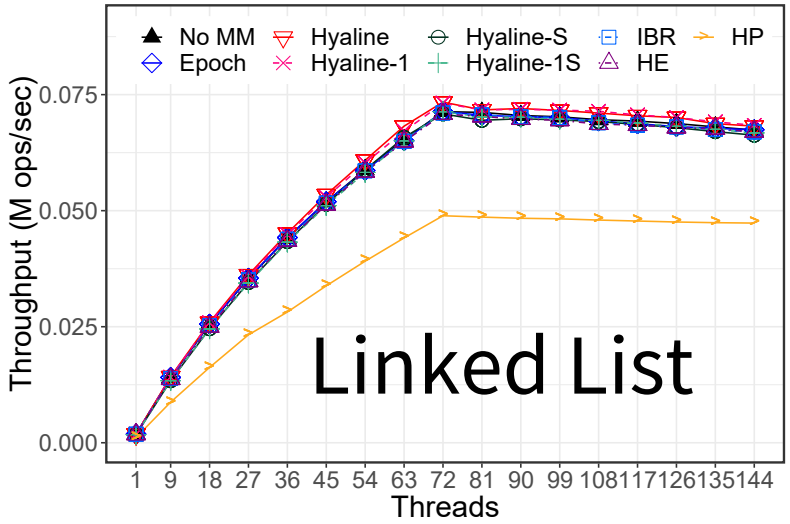
EVALUATION

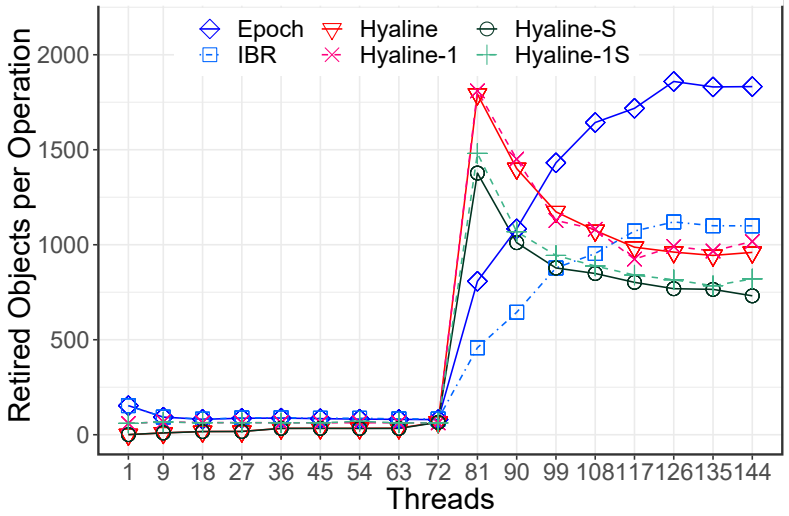

No MM
Hyaline
Hyaline-S
IBR
Epoch
Hyaline-1
Hyaline-1S
Bonsai Tree
Throughput (M ops/sec)
Threads
0
1
2
3
1 9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144

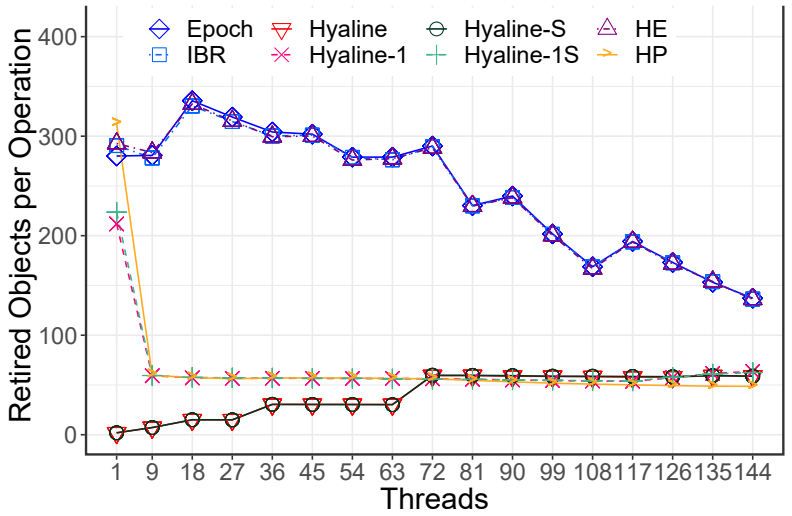

Epoch
Hyaline
Hyaline-S
IBR
Hyaline-1
Hyaline-1S
Retired Objects per Operation
Threads
0
500
1000
1500
2000
1 9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144

EVALUATION

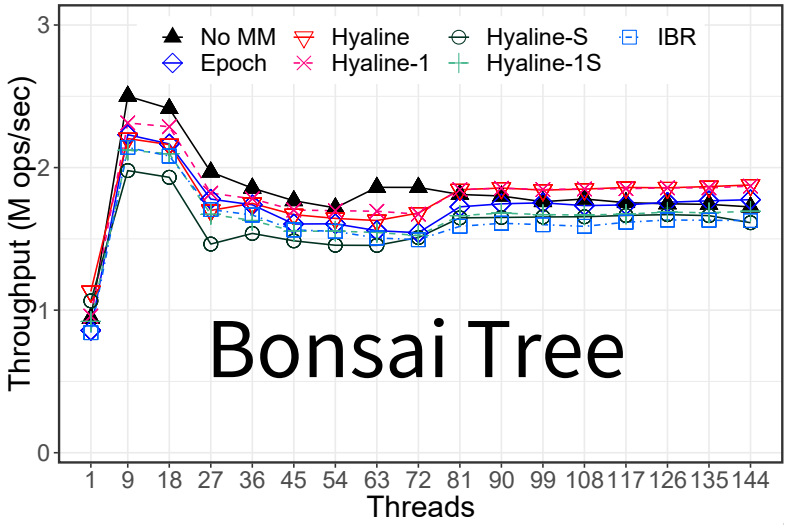

No MM
Epoch
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
IBR
Throughput (M ops/sec)
Threads
Bonsai Tree

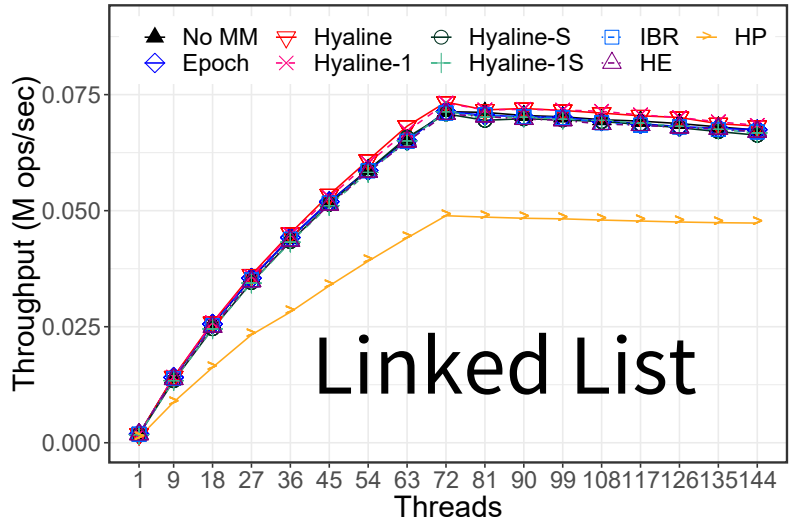

No MM
Epoch
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
IBR
HE
HP
Throughput (M ops/sec)
Threads
Linked List

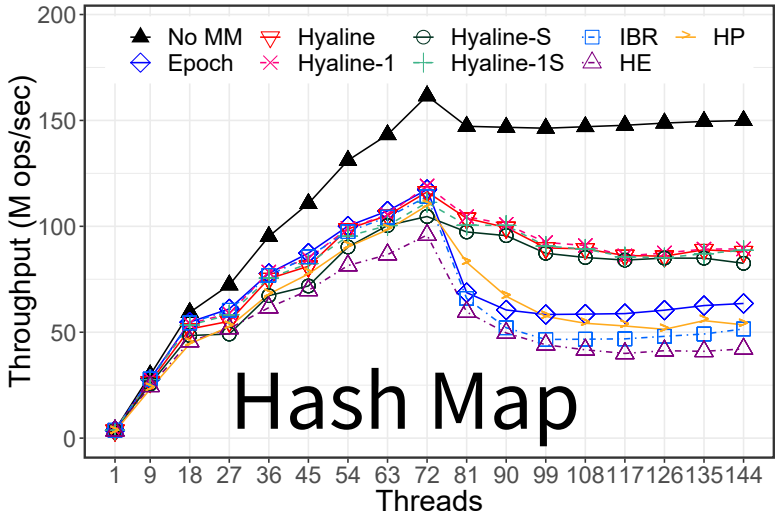

Epoch
IBR
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
Retired Objects per Operation
Threads

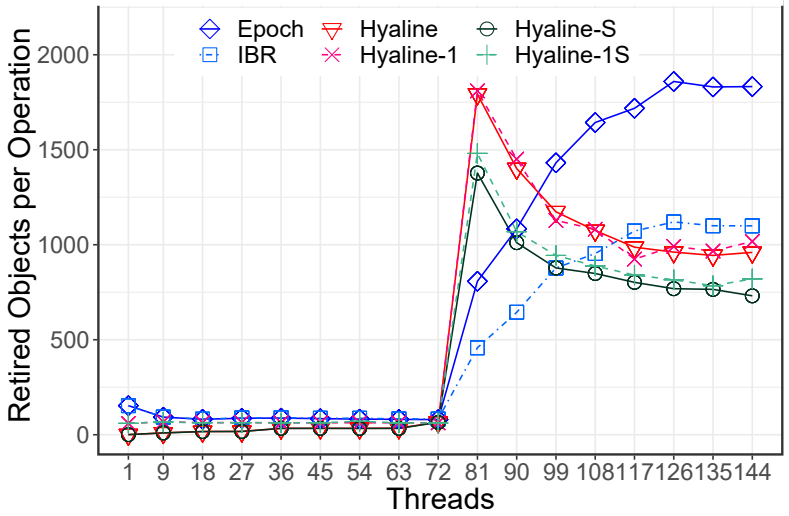

Epoch
IBR
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
HE
HP
Retired Objects per Operation
Threads

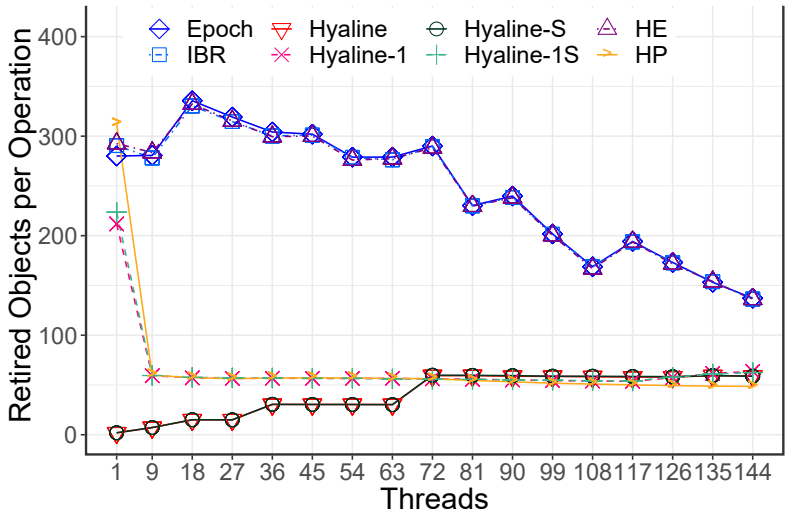
EVALUATION

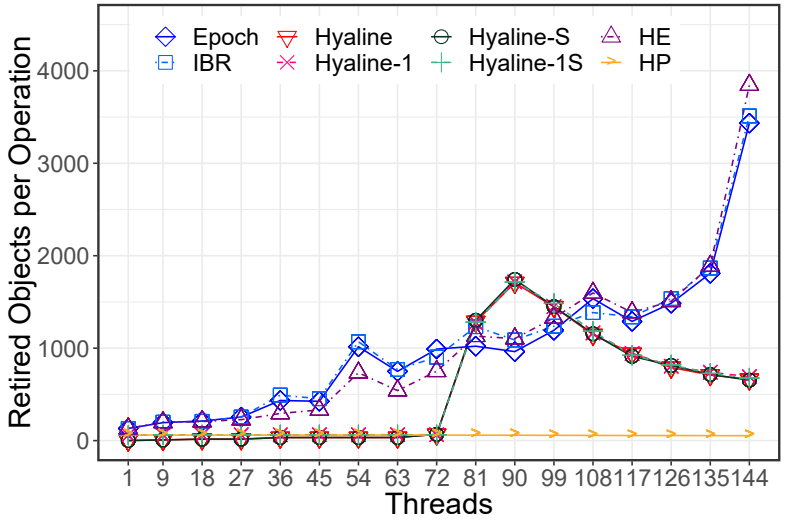

No MM
Epoch
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
IBR
Throughput (M ops/sec)
Threads
Bonsai Tree


No MM
Epoch
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
IBR
HE
HP
Throughput (M ops/sec)
Threads
Linked List


No MM
Epoch
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
IBR
HE
HP
Throughput (M ops/sec)
Threads
Hash Map


Epoch
IBR
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
Retired Objects per Operation
Threads


Epoch
IBR
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
HE
HP
Retired Objects per Operation
Threads


Epoch
IBR
Hyaline
Hyaline-1
Hyaline-S
Hyaline-1S
HE
HP
Retired Objects per Operation
Threads

AVAILABILITY

- Hyaline's code and the benchmark are open-source and available at
  - https://github.com/rusnikola/lfsmr

- Additional paper appendices are available at
  - https://arxiv.org/pdf/1905.07903.pdf

THANK YOU!