PennState

Fixing Non-blocking Data Structures for Better Compatibility with Memory Reclamation Schemes

Md Amit Hasan Arovi, arovi@psu.edu, The Pennsylvania State University, USA

Ruslan Nikolaev, rnikola@psu.edu, The Pennsylvania State University, USA

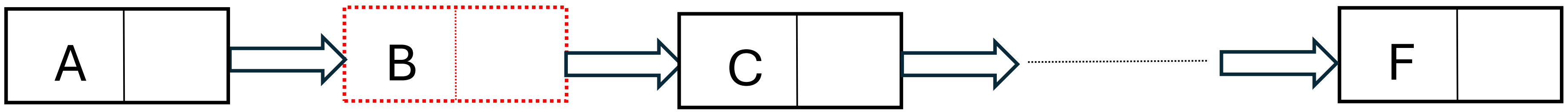
Non-blocking Data Structures

- Allow multiple threads to operate without mutual exclusion
- *Lock-free* data structures: at least one thread always makes progress
- *Wait-free* data structures: all threads always make progress in a finite number of steps
- Require safe memory reclamation (SMR) techniques
  - Epoch-Based Reclamation (EBR) is easy-to-use but has unbounded memory usage
  - Hazard Pointers (HP) is more difficult to use but is *robust*

Problems with Non-blocking Data Structures

- Many robust memory reclamation schemes (e.g., HP) fail to support **optimistic traversals** used in many lock-free algorithms:
  - Harris' linked list
  - The Natarajan-Mittal tree
  - Many others (skip lists, hash tables, etc)
- Workarounds are sometimes available with performance costs
  - Harris-Michael linked list
  - No such modification for the Natarajan-Mittal tree

Problem: Harris' Linked List with HP

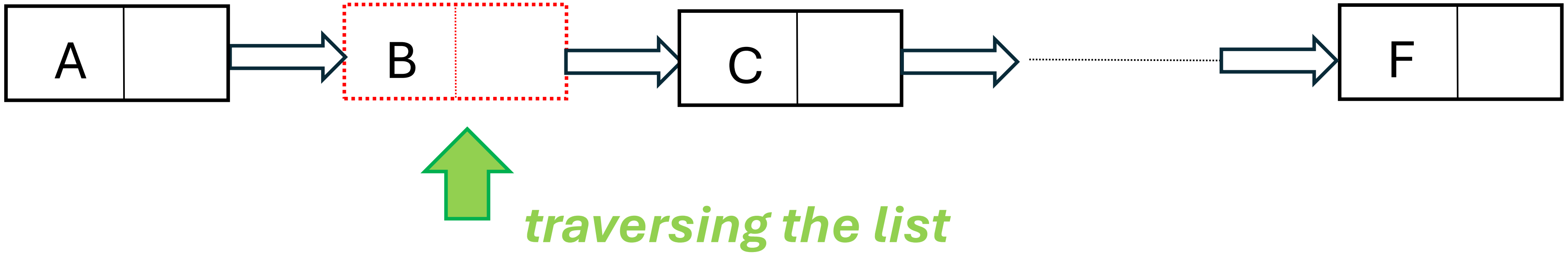


Thread 0: Delete(B) => Node B is marked for logical deletion

… Thread 0 is stalled

Problem: Harris' Linked List with HP

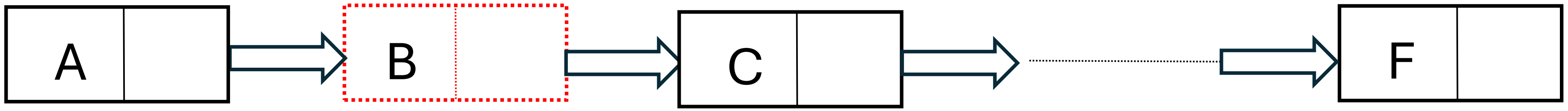
Thread 1 (Search)



Thread 1: Locate node F

Problem: Harris' Linked List with HP

Thread 1 (Search)

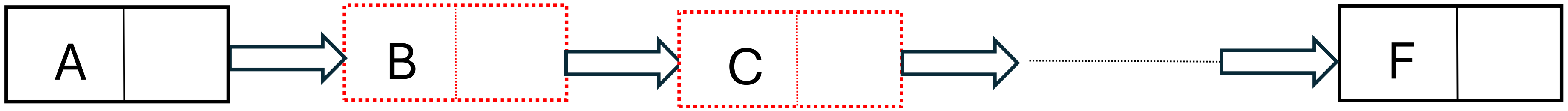


Thread 2 (Delete)

Thread 2: Delete(C)

Problem: Harris' Linked List with HP

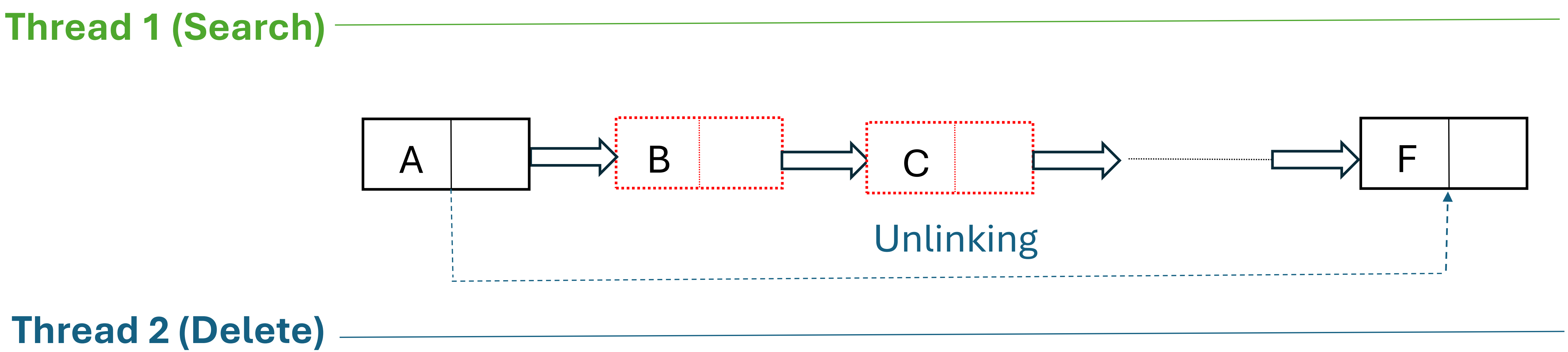
Thread 1 (Search)



Thread 2 (Delete)

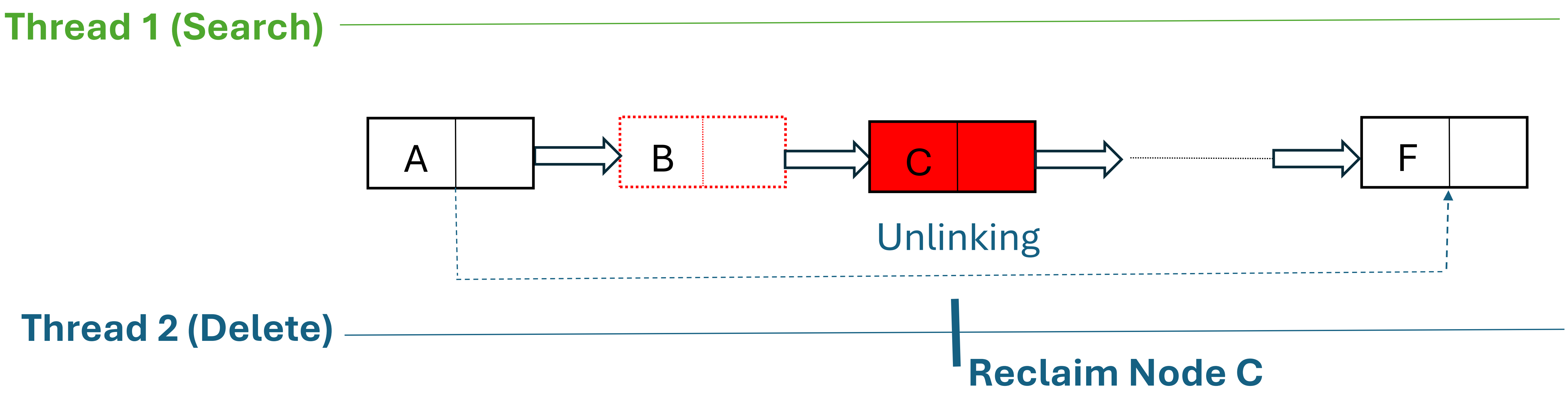
Thread 2: Node C is marked for logical deletion

Problem: Harris' Linked List with HP



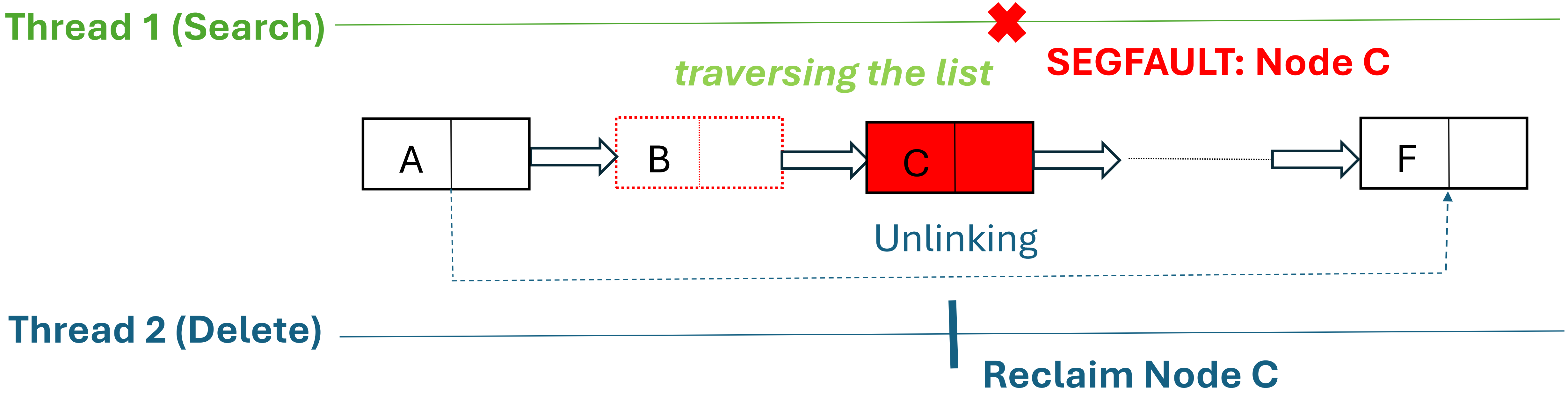
Thread 2 unlinks the entire chain of nodes between Node A and Node F (assuming *all* consecutive nodes in the chain are logically deleted)

Problem: Harris' Linked List with HP



Node C is Reclaimed by Thread 2 and returned to the OS

Problem: Harris' Linked List with HP

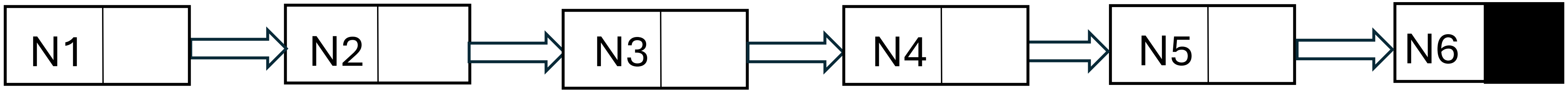


Thread 1 fails when accessing Node C

Safe Concurrent Optimistic Traversals (SCOT)

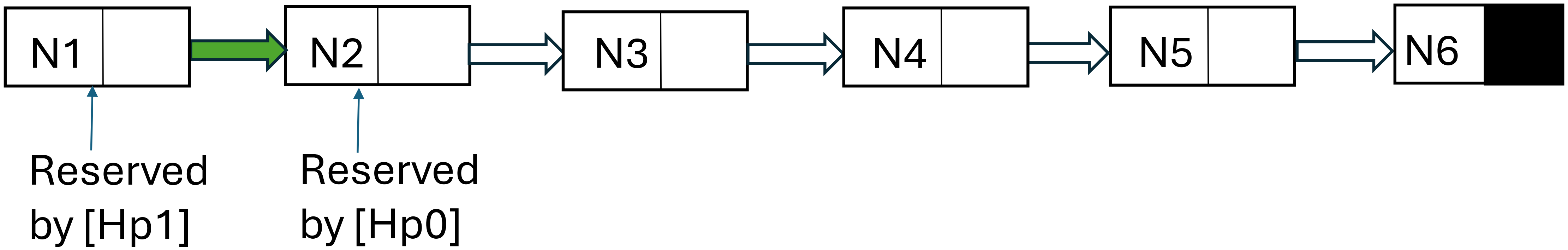
- Instead of fixing the SMR → fix the **data structure**
  - Resolves the robustness vs. applicability dilemma, ERA Theorem [PODC ’23]
- Redesign traversals to add **local validation**

SCOT: Harris' Linked List



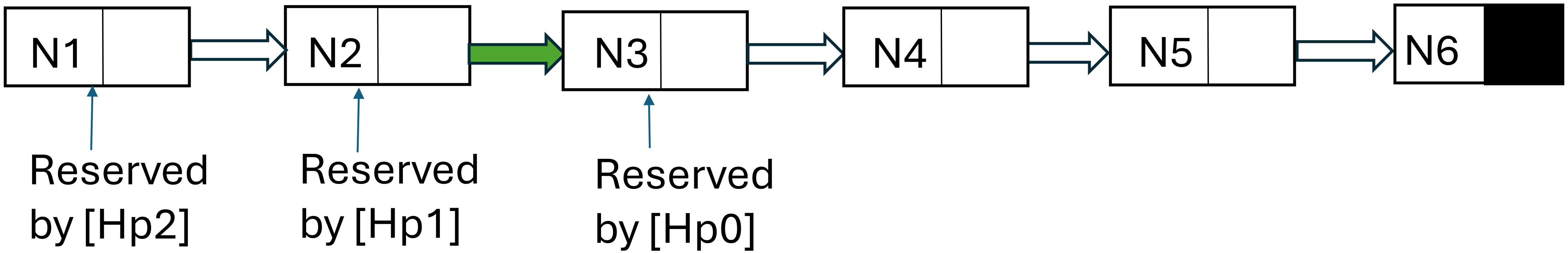
Initial State: List Contains Nodes N1-N6

SCOT: Harris' Linked List



Hp0: protects next
Hp1: protects curr
Hp2: protects prev (not available at the very beginning)

SCOT: Harris' Linked List



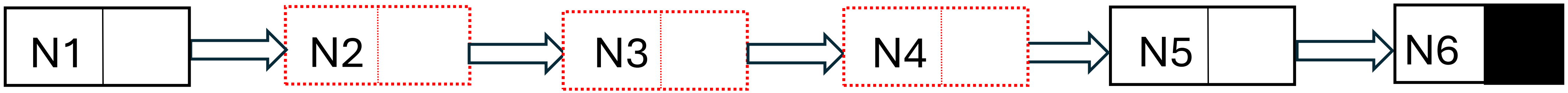
Moving hazard pointers when moving to the next iteration:

curr (N1) [Hp1] -> prev [Hp2]

next (N2) [Hp0] -> curr [Hp1]

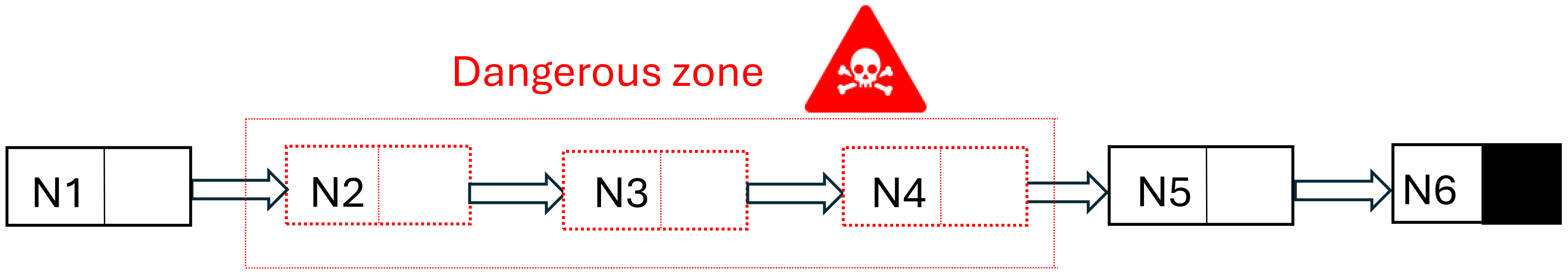
(new) next (N3) [Hp0]

SCOT: Harris' Linked List



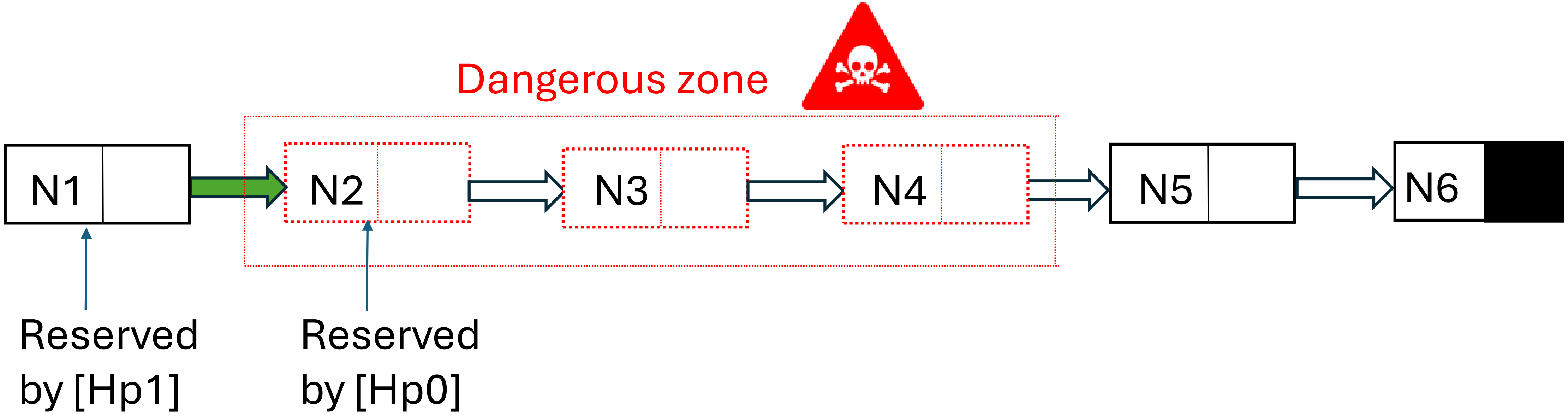
Node N2-N4 are logically deleted

SCOT: Harris' Linked List



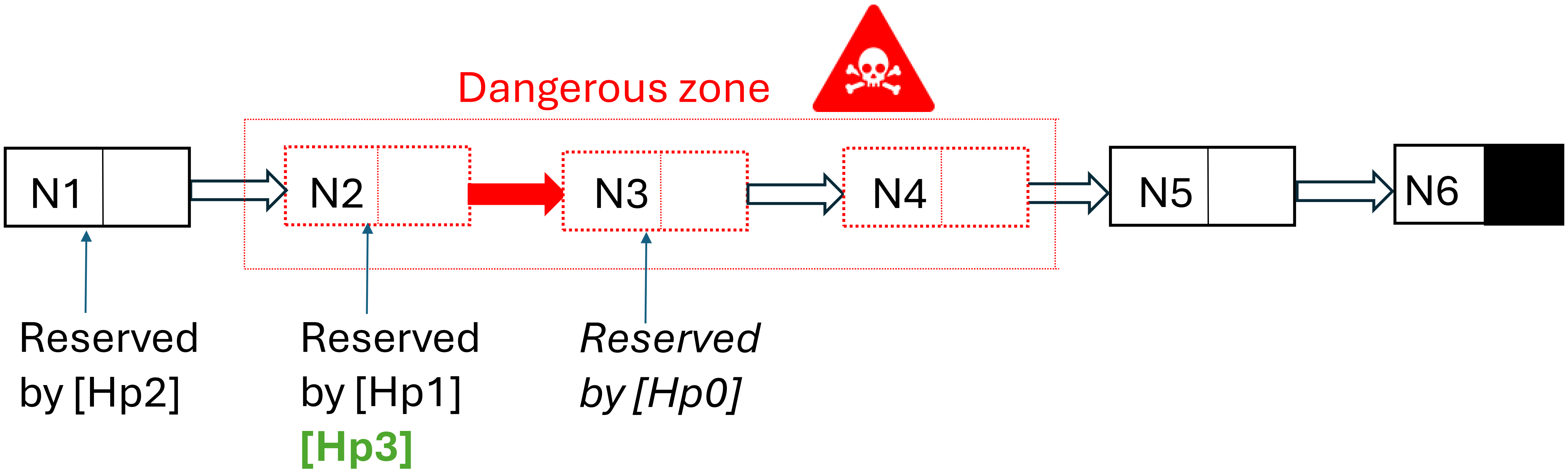
Need to be careful while traversing the dangerous zone

SCOT: Harris' Linked List



N1 and N2 protected by hazard pointers' reservations

SCOT: Harris' Linked List



Next destination N3

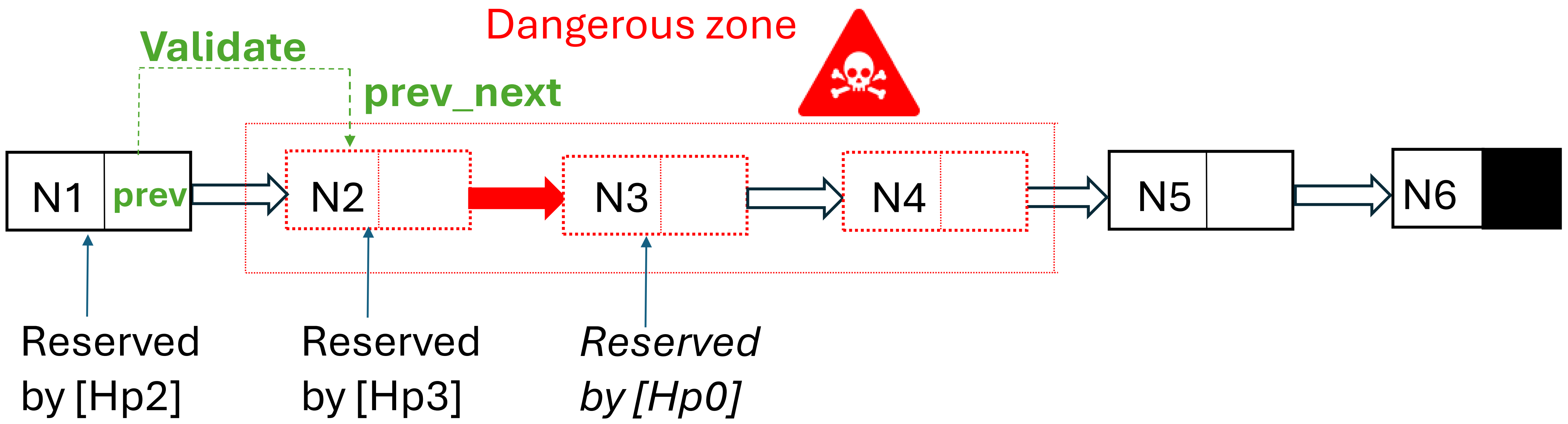
N1 is the last safe node and protected by Hp2

N2 is the first unsafe node protected by **Hp3**

Hp3 is an extra hazard pointer which protects the 1st unsafe node

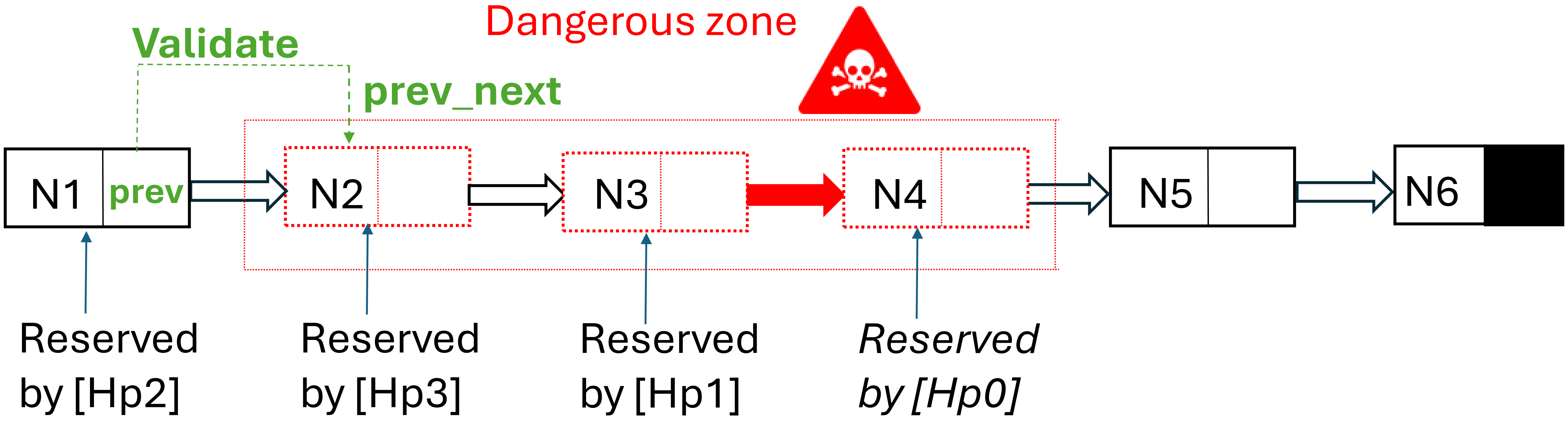
Hp2 protects the last safe node

SCOT: Harris' Linked List



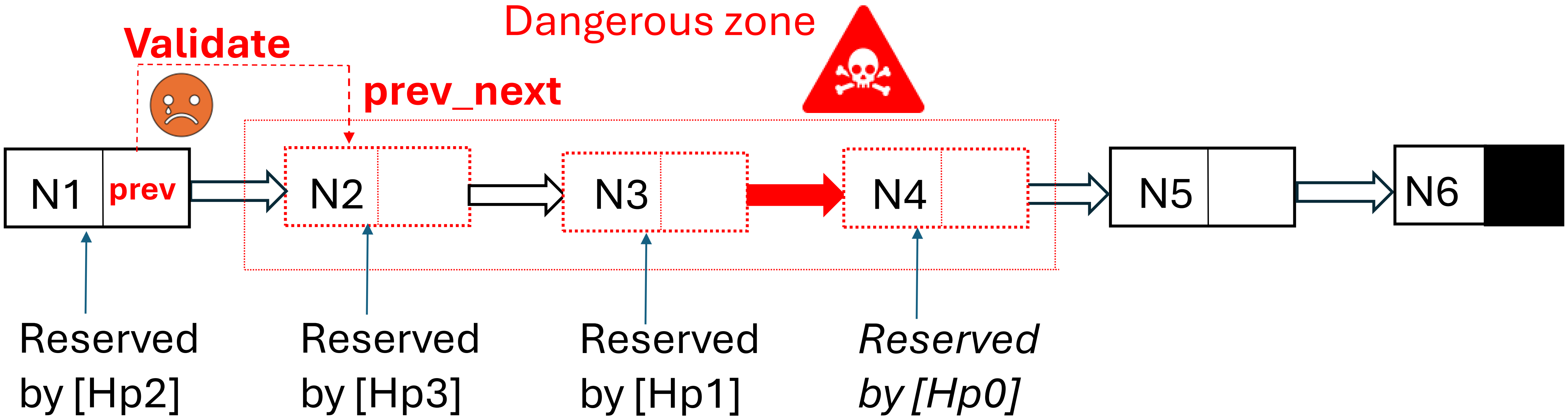
Validate (*prev = prev_next) after reserving N3 (Hp0)

SCOT: Harris' Linked List



Validate (*prev = prev_next) after reserving N4 (Hp0)

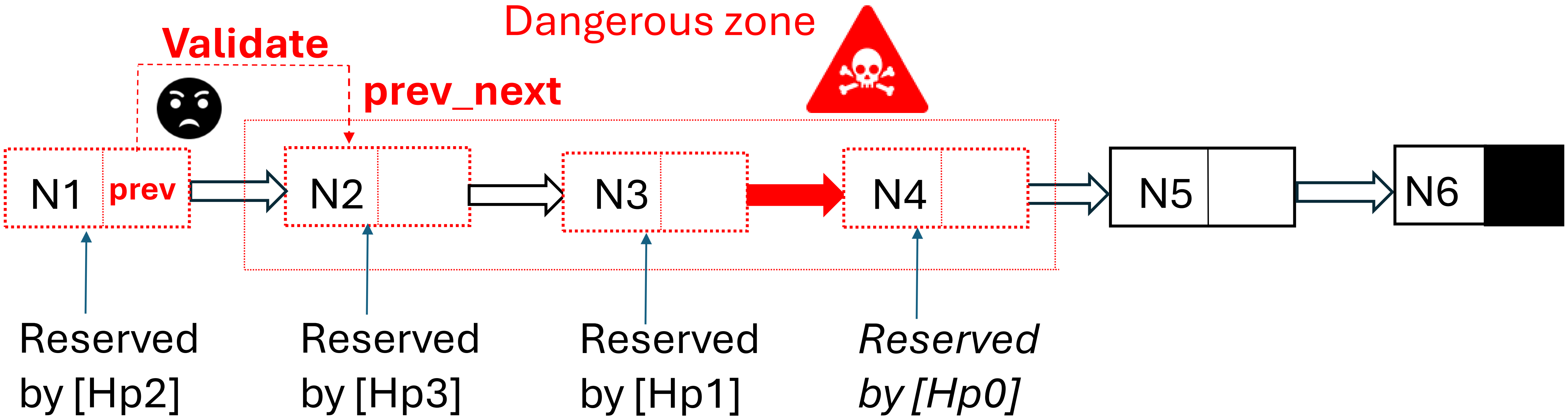
SCOT: Recovery



What if (*prev = prev_next) validation fails due to a new node being inserted or the chain of logically deleted nodes being already eliminated by a concurrent thread?

We start from the last safe node (N1)

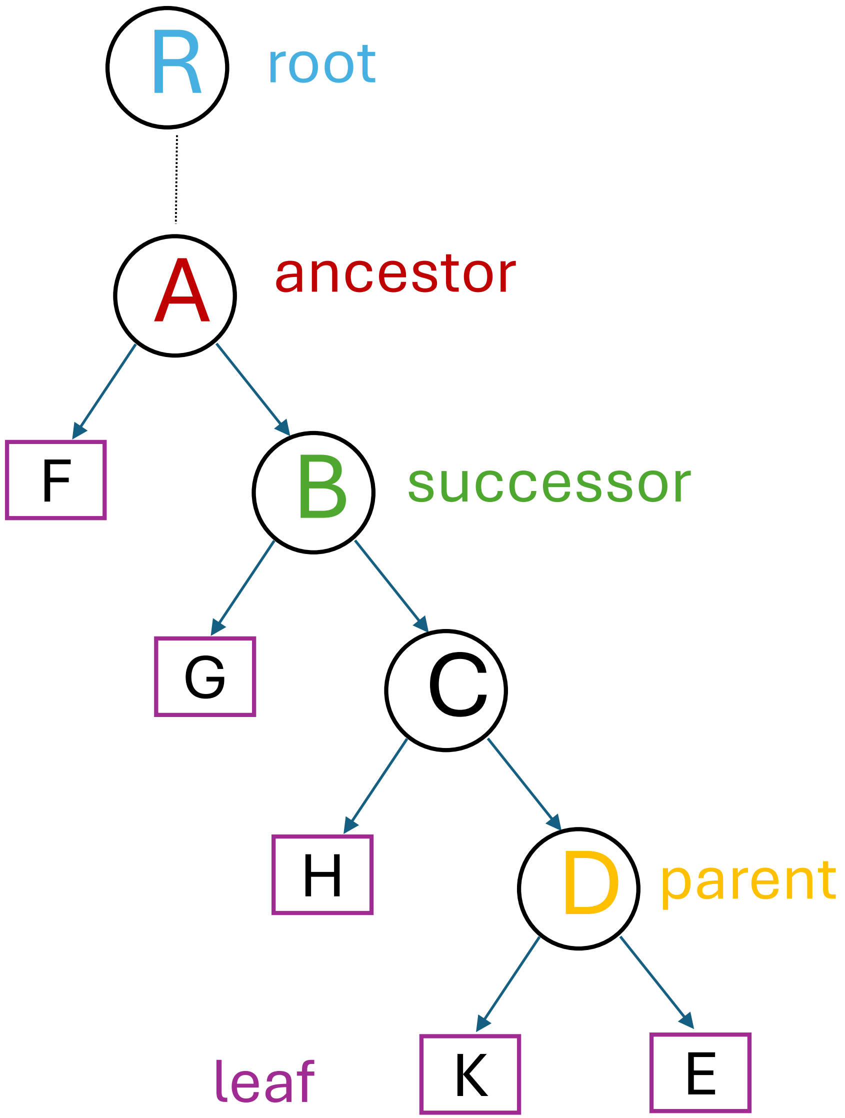
SCOT: Recovery



What if the last safe node (N1) is also logically deleted?

We start from the beginning

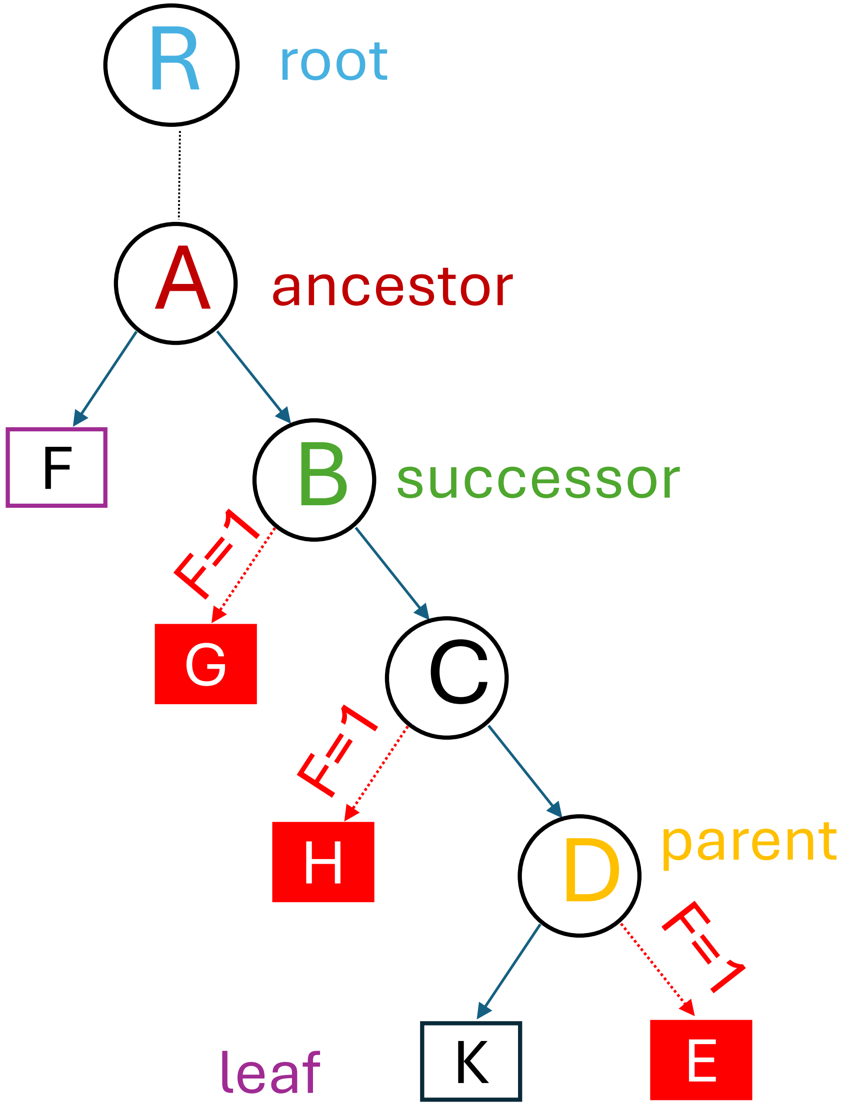
SCOT: The Natarajan-Mittal Tree



Leaf nodes contain actual keys

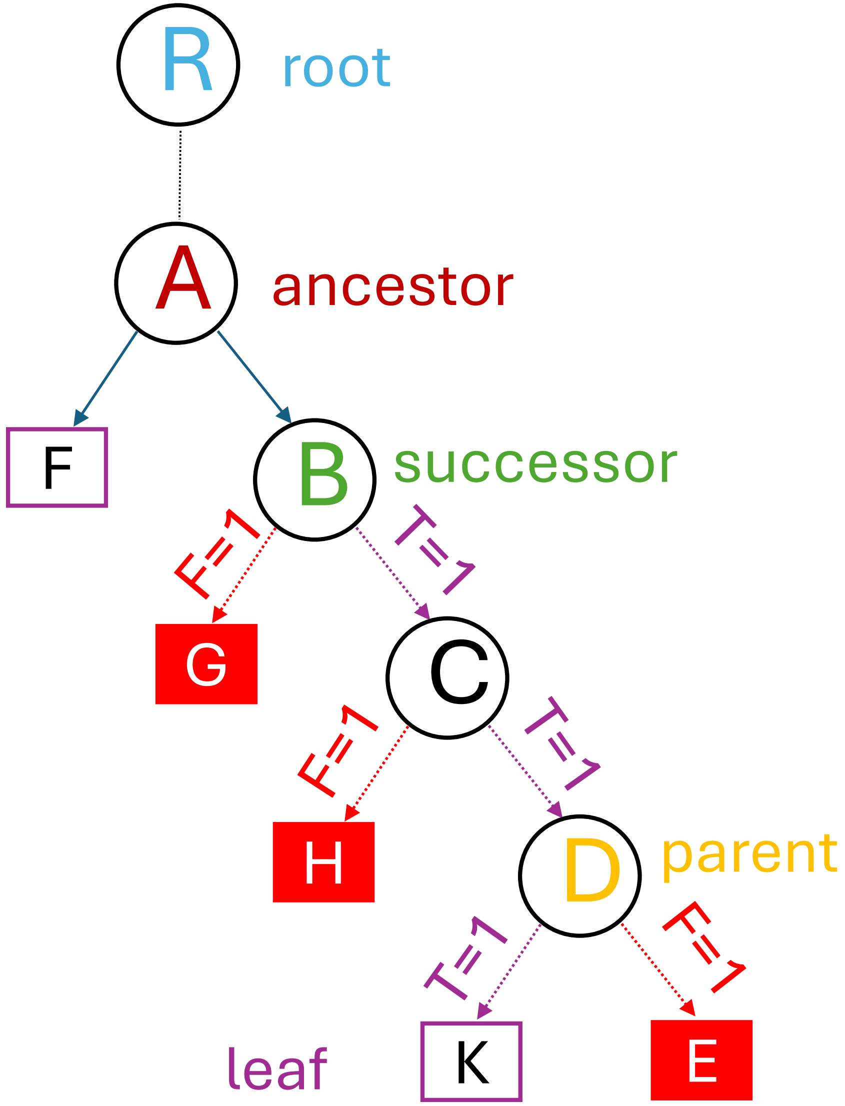
Keys in Internal nodes are used for traversal

SCOT: The Natarajan-Mittal Tree



Leaf nodes G, H, E are deleted (flagged, F=1)

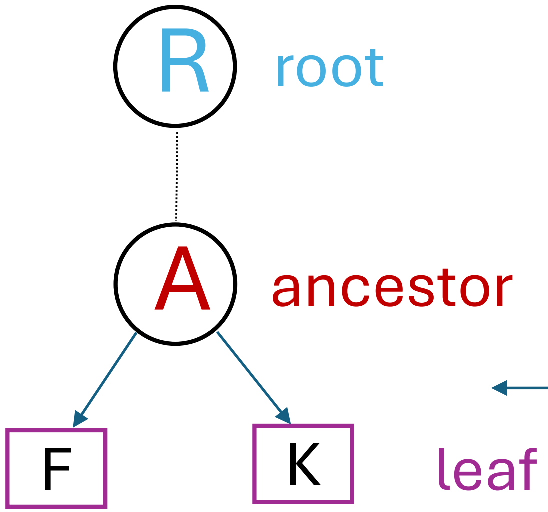
SCOT: The Natarajan-Mittal Tree



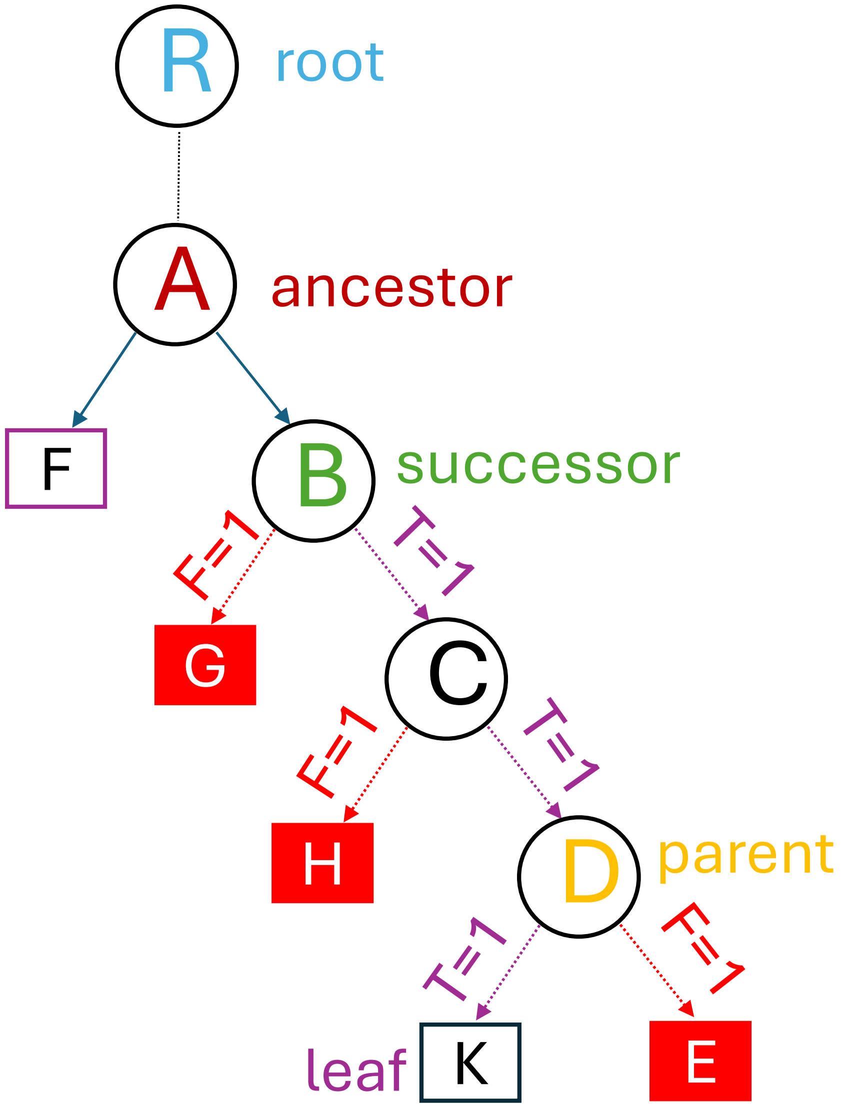
When leaf nodes G, H, E are deleted (flagged, F=1)

Siblings (B-C, C-D, D-K) are tagged (T=1)

SCOT: The Natarajan-Mittal Tree



Ancestor’s right link is changed from successor (B) to the remaining leaf node (K)



An important observation: a chain of consecutively tagged edges can be eliminated with one CAS operation by updating ancestor’s link from successor to the remaining leaf node

successor node is the last untagged node

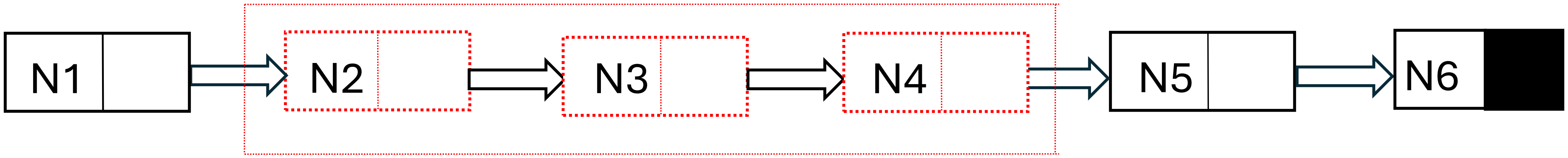
SCOT: The Natarajan-Mittal Tree

- We allocate 5 hazard pointers to protect nodes in the underlying search procedure: current, leaf, parent, successor, and ancestor. The current node points to the lowest node that is currently considered

- After each HP reservation of the current node, if the corresponding node is flagged or tagged, we need to verify that ancestor still points to successor

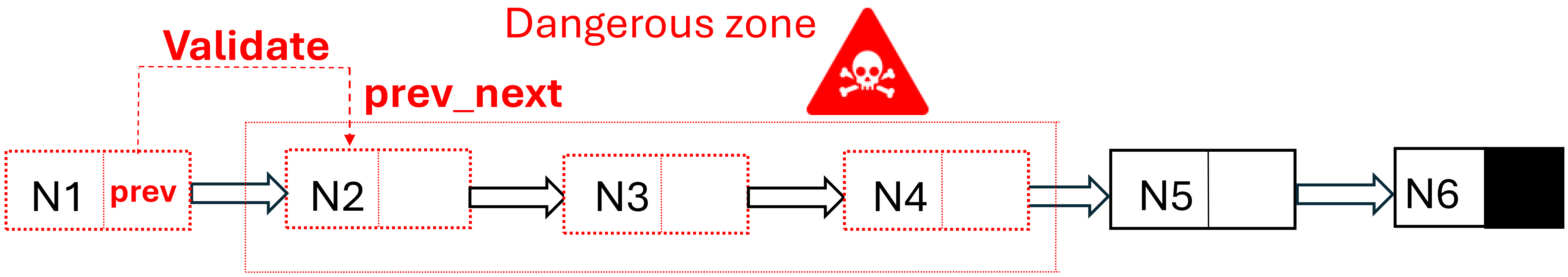
- If ancestor points to some other node or successor becomes tagged, we need to restart from the very beginning

SCOT: Wait-free traversal: Harris' Linked List



The original data structure with EBR allows wait-free read-only traversals, no need to restart from the beginning

SCOT: Wait-free traversal: Harris' Linked List



If validation fails due to the data structure state change and the last safe node is logically deleted, we need to restart from the beginning; guarantees lock-free but not wait-free progress.

SCOT: Wait-free traversal: Harris' Linked List

- Our approach is specifically tailored for traversals and uses a per-thread record entry.
- Avoids nonstandard instructions (only a regular CAS) and complexity related to dynamically-allocated descriptors (OPODIS '12, PPoPP '14).

Search ⟶ Request_Help ⟶ Slow_Search

SCOT: Wait-free traversal: Harris' Linked List

- Our approach is specifically tailored for traversals and uses a per-thread record entry.
- Avoids nonstandard instructions (only a regular CAS) and complexity related to dynamically-allocated descriptors (OPODIS '12, PPoPP '14).

Search ⟶ Request_Help ⟶ Slow_Search

Insert / Delete ⟶ Help_Threads $\xrightarrow{\text{True}}$ Slow_Search

SCOT: Wait-free traversal: Harris' Linked List

Input request (T_{helpee})

Value	IsInput
slow-path cycle #	1
bit 63:1	bit 0

Output result (T_{helpee})

Value	IsInput
result True/False	0
bit 63:1	bit 0

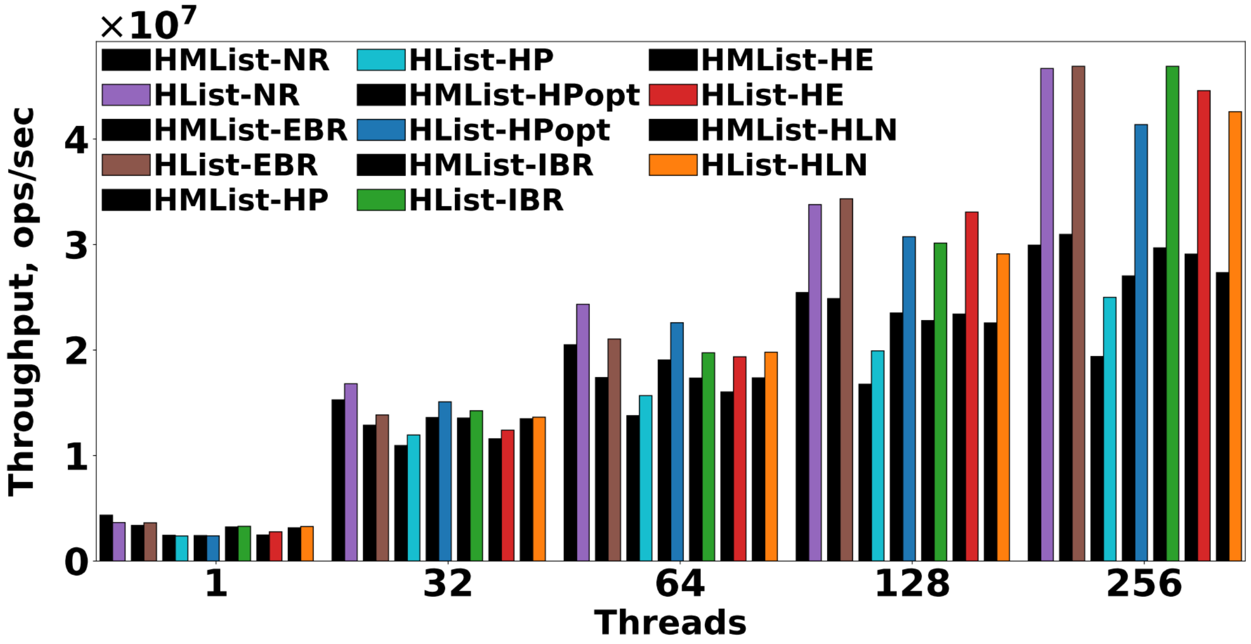
Output is produced by both helpee and helpers

Evaluation Setup

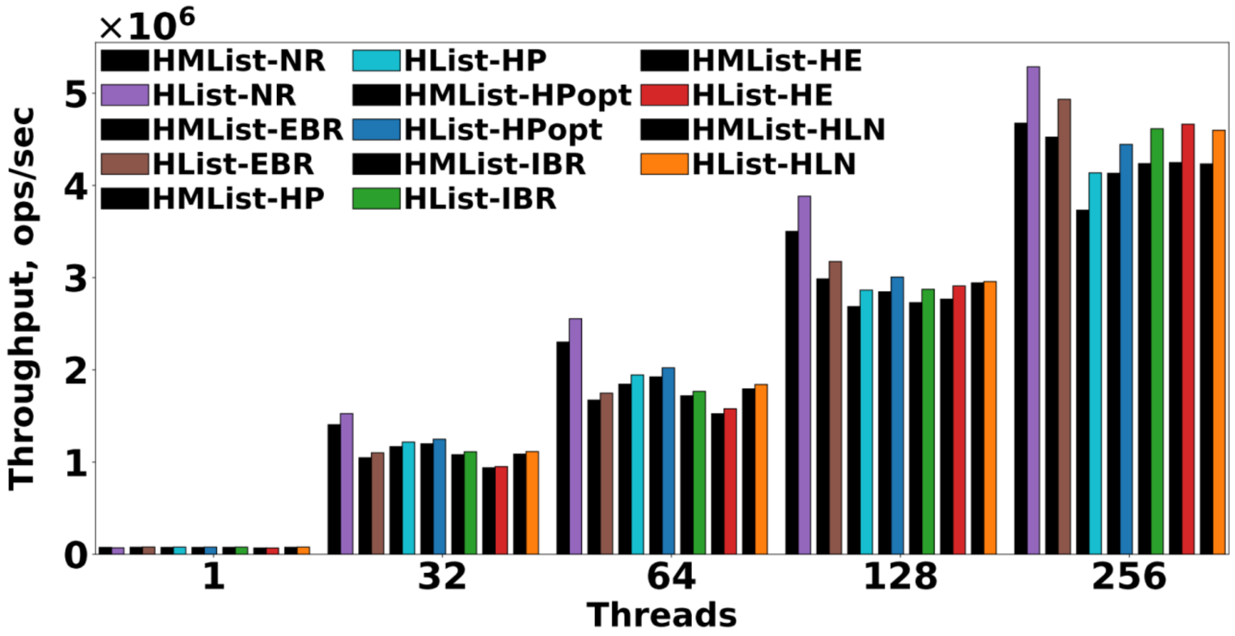
- AMD EPYC 9754, 128 cores, 256 hardware threads, 384 GiB of RAM

- SMR Schemes
    - No-Reclamation (**NR**) baseline which leaks memory
    - Epoch-Based Reclamation (**EBR**)
    - Hazard Pointers (**HP**): TPDS ’04
    - Hazard Eras (**HE**): SPAA ’17
    - Interval-Based Reclamation (**IBR**): PPoPP ’18
    - Hyaline-1S (**HLN**): PODC ’19, PLDI ’21

Evaluation: Harris vs. Harris-Michael list (Throughput)

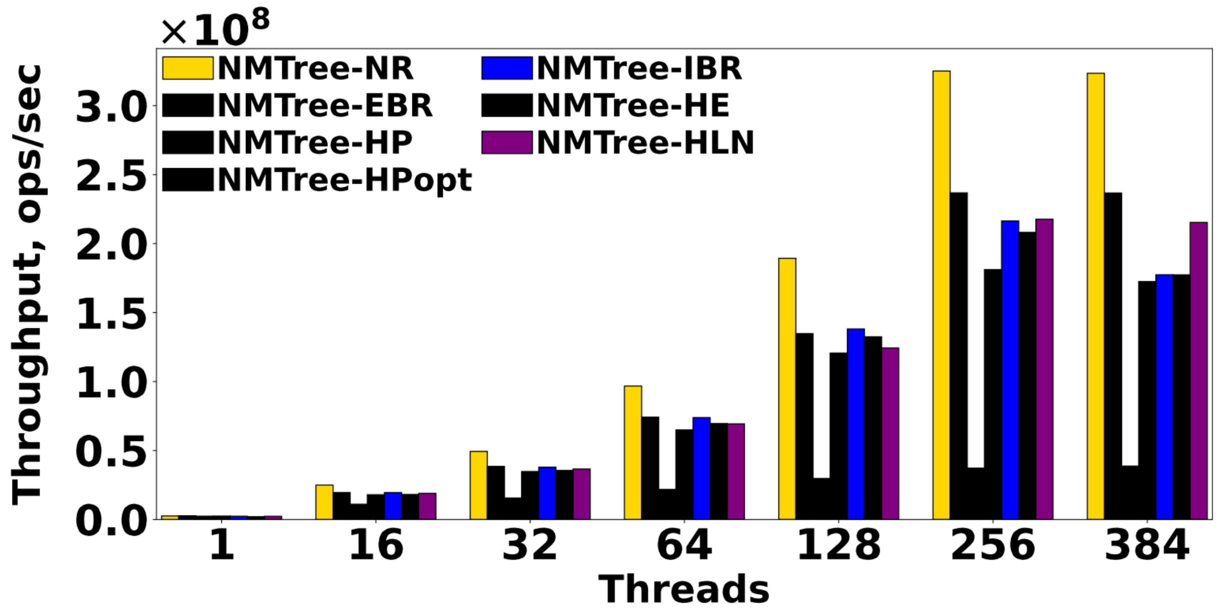


(a) Key Range = 512

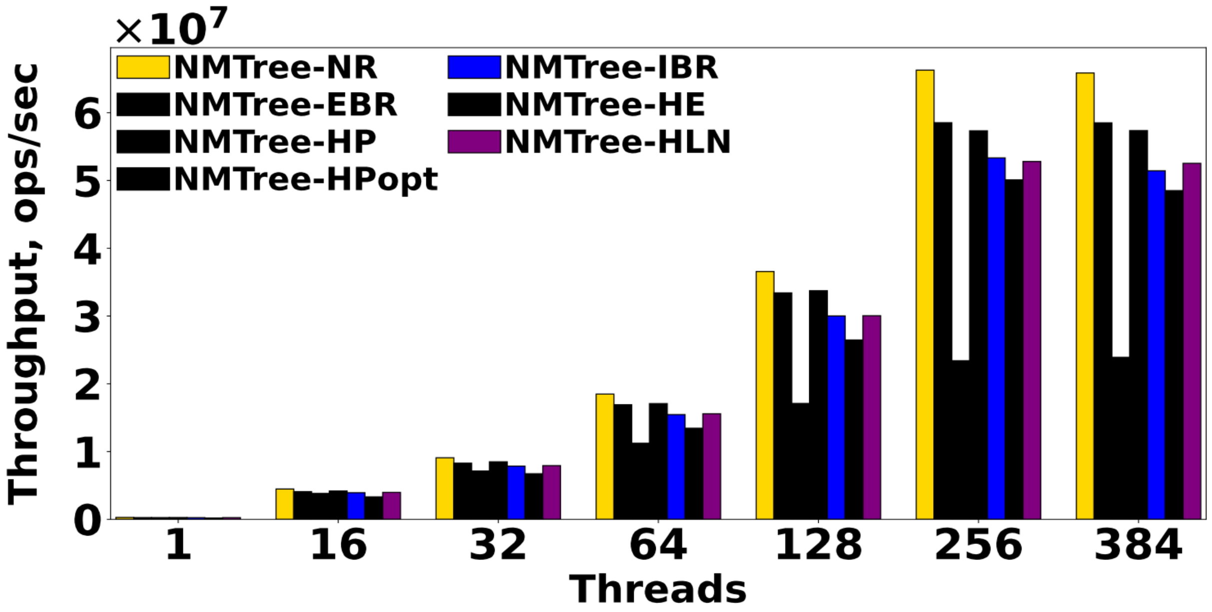


(b) Key Range = 10,000

Evaluation: The Natarajan-Mittal tree (Throughput)



(a) Key Range =100,000



(b) Key Range = 50,000,000

Code Availability

- Code is open-source and available:

  GitHub Repository: https://github.com/rusnikola/scot

  Zenodo DOI: https://doi.org/10.5281/zenodo.17707898

Code Availability

- Code is open-source and available:

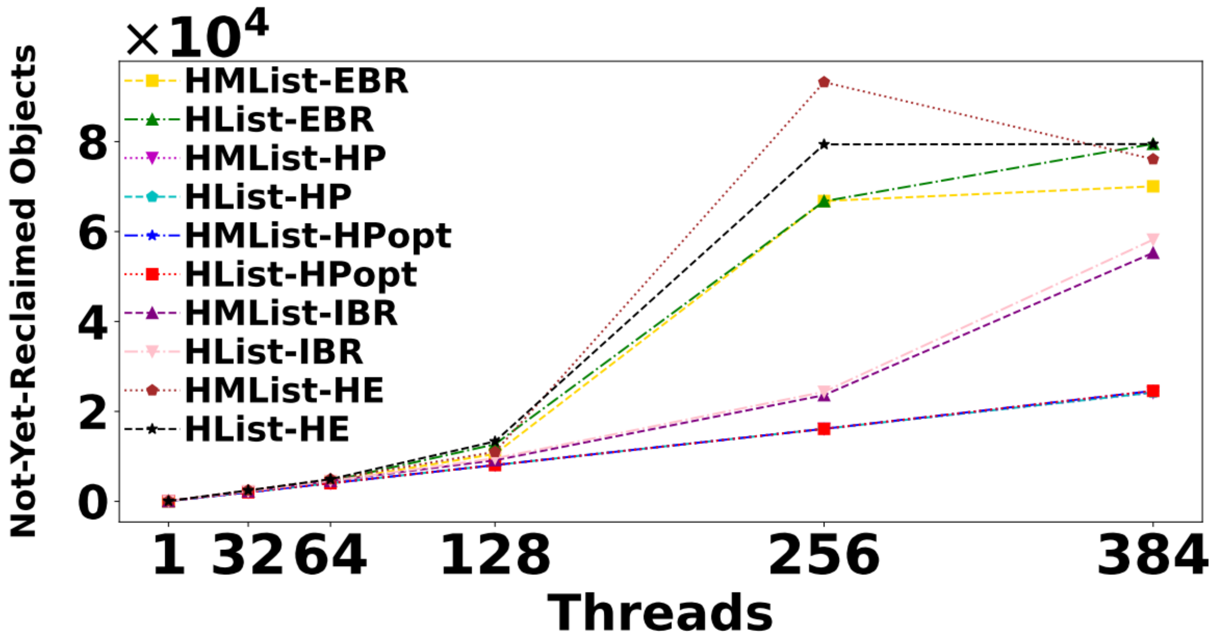
  GitHub Repository: https://github.com/rusnikola/scot

  Zenodo DOI: https://doi.org/10.5281/zenodo.17707898

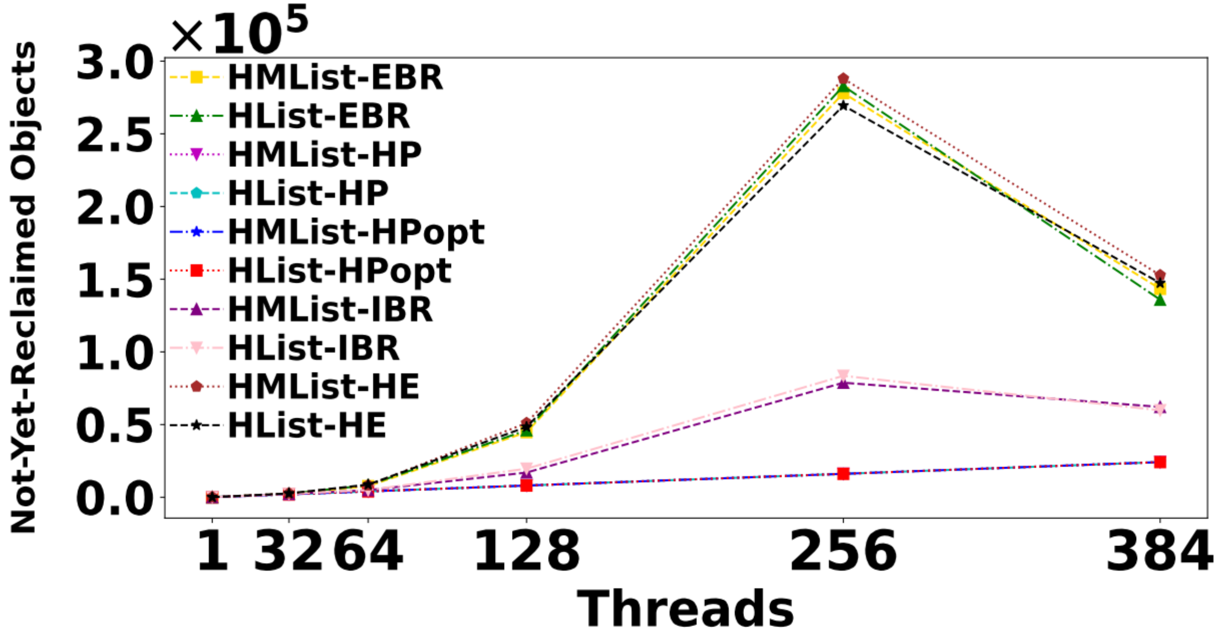
Thank You!

Questions?

Evaluation: Harris vs. Harris-Michael list (Avg. Number of Not-Yet-Reclaimed Objects)

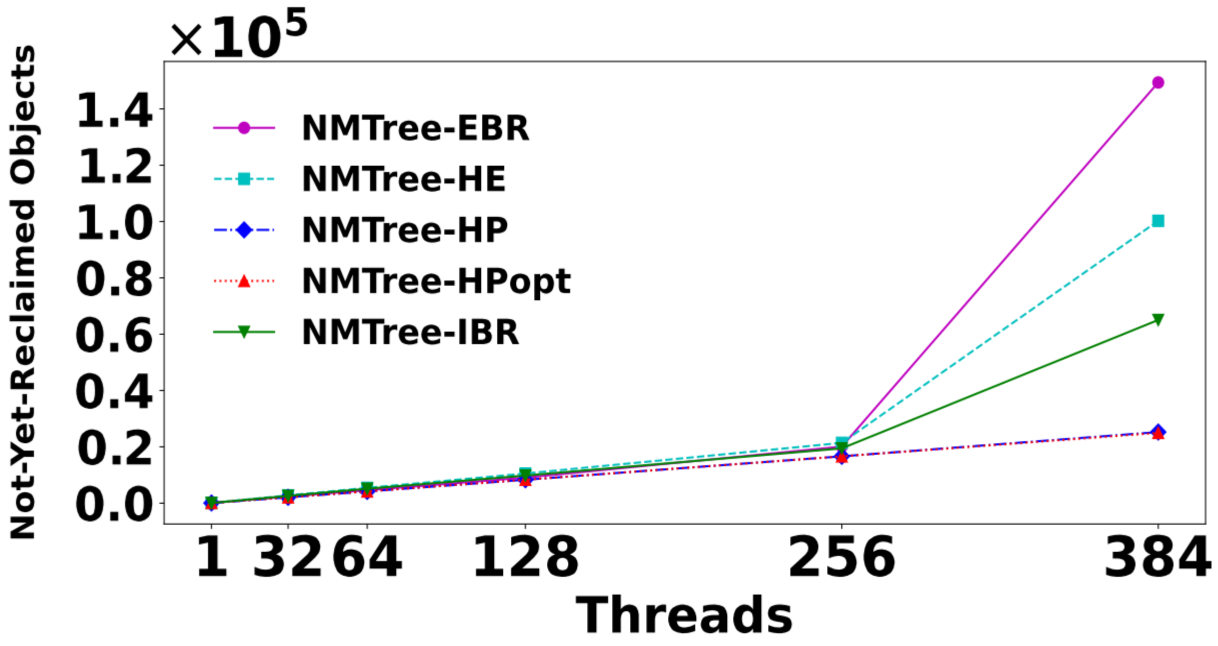


(a) Key Range = 512

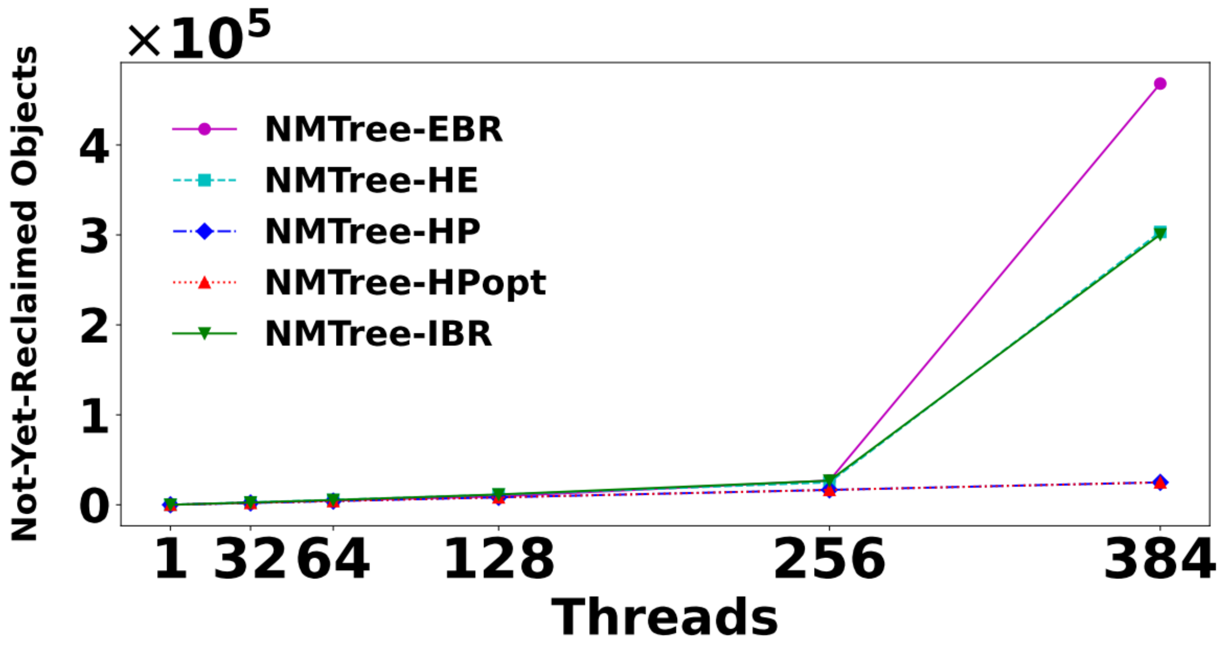


(b) Key Range = 10,000

Evaluation: The Natarajan-Mittal tree (Avg. Number of Not-Yet-Reclaimed Objects)

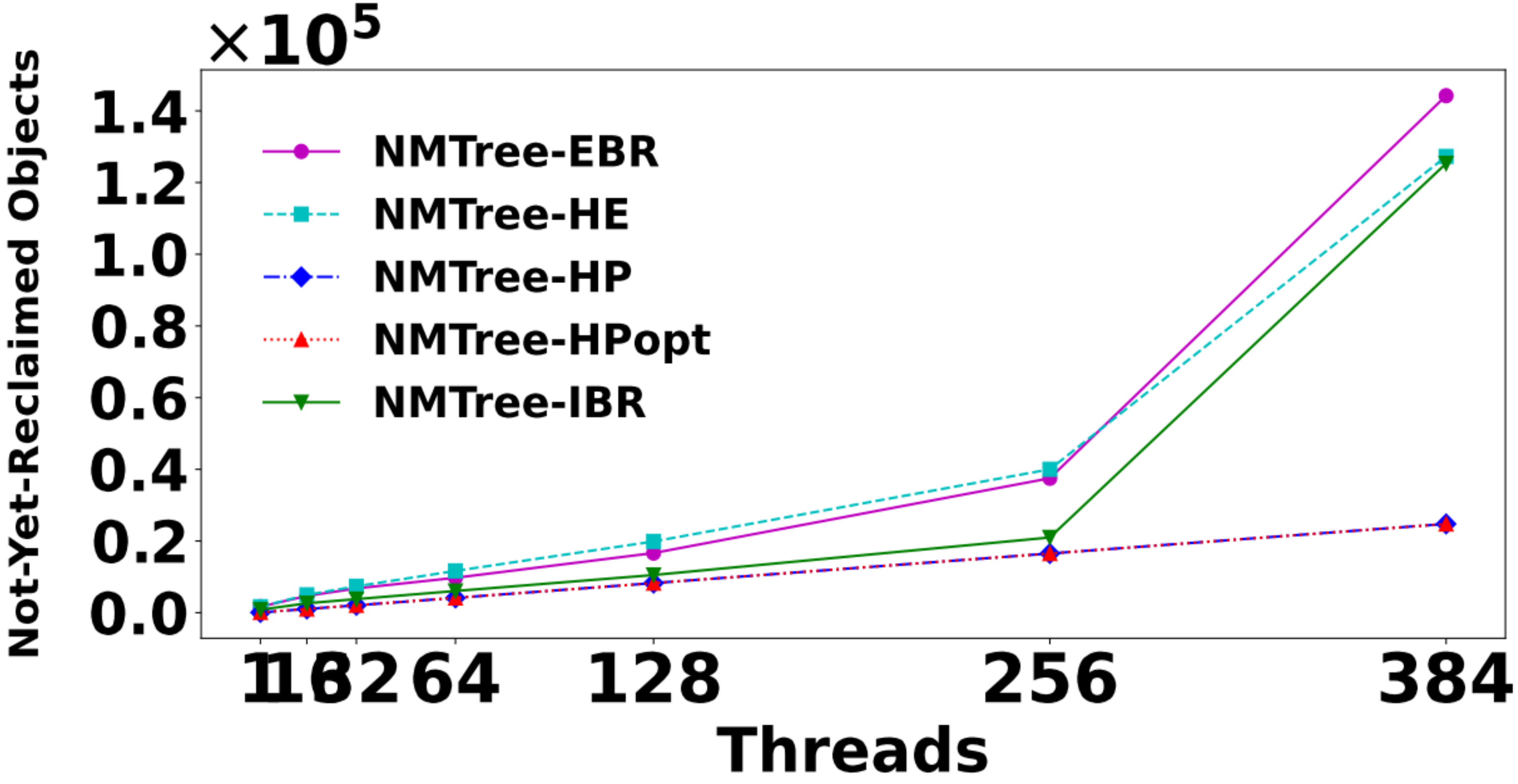


(a) Key Range = 128



(b) Key Range = 100,000

Evaluation: The Natarajan-Mittal tree (Avg. Number of Not-Yet-Reclaimed Objects)



(c) Key Range = 50,000,000