Order out of Chaos: Proving Linearizability Using Local Views

Yotam Feldman
Tel Aviv University

Constantin Enea
University Paris Diterot

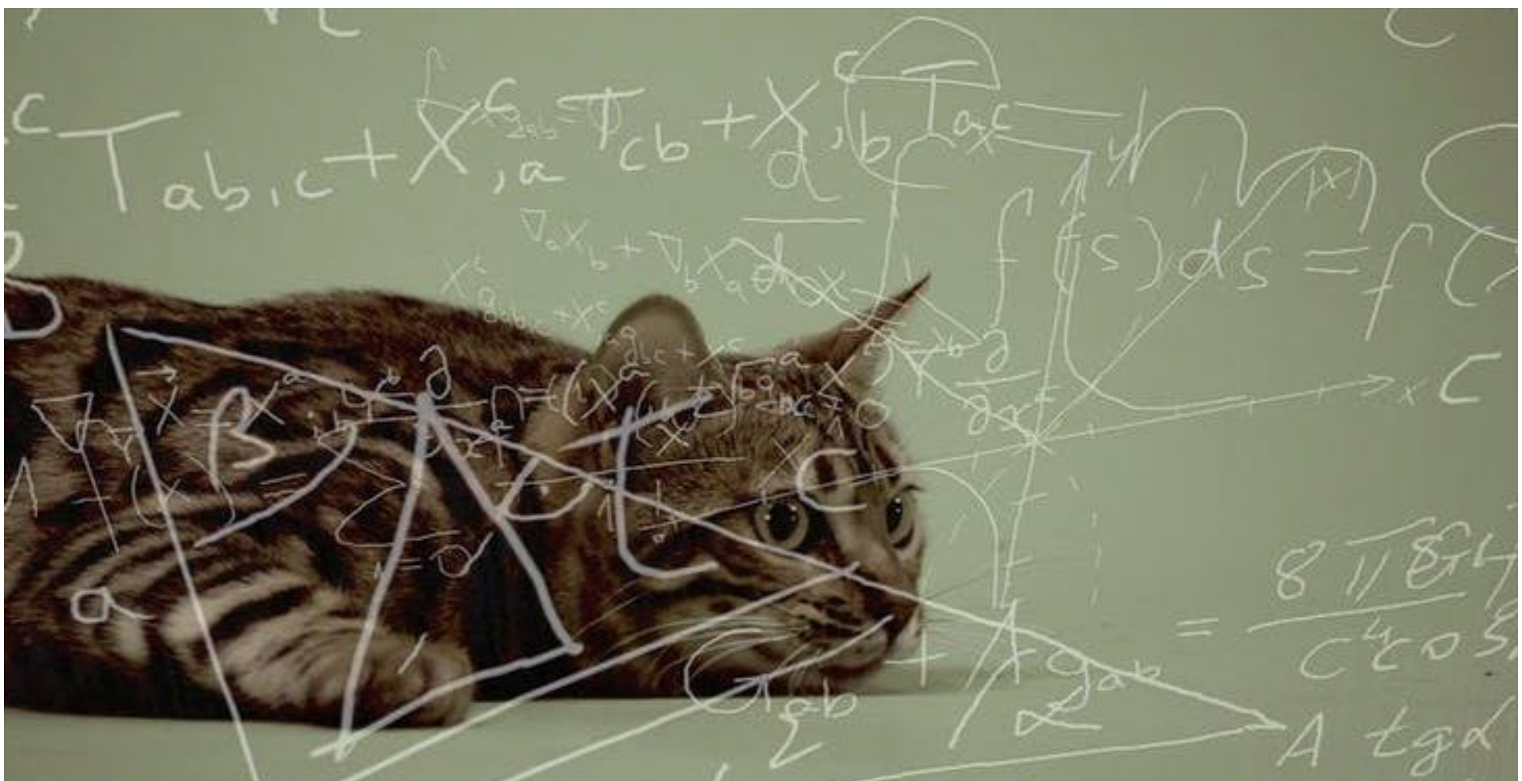
Adam Morrison
Tel Aviv University

Noam Rinetzky
Tel Aviv University

Sharon Shoham
Tel Aviv University

Concurrent Reasoning is Hard

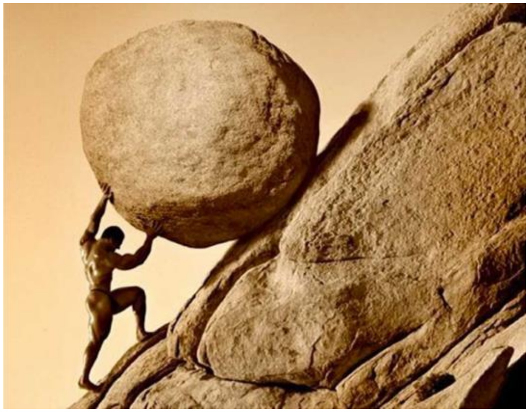
Goal: Prove linearizability of
highly-concurrent data structures



Optimistic Traversals

Good performance, hard proofs

- Lazy list [OPODIS’05]
- Lock-free skiplist [Fraser-2004]
- Contention-Friendly Tree [EuroPar’13, PPL’16]
- Lock free trees [PPOPP’14b, SPAA’12, PPOPP’14a, PODC’10, ICDCN’15]
- …



This Work

Main challenge: correctness of optimistic **traversals**

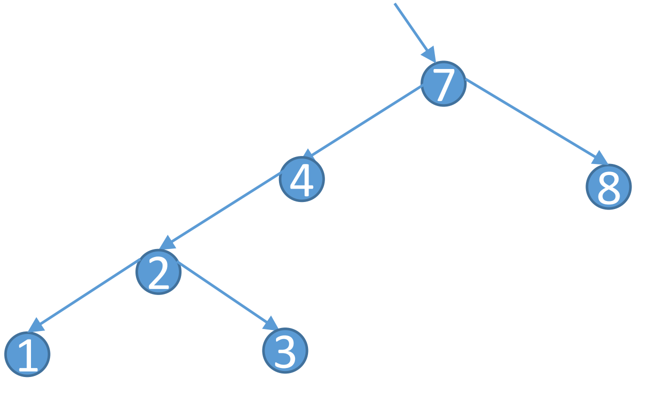
Traversal correctness **without interference** (sequential reasoning)

→

Traversal correctness **with interference** (concurrent reasoning)

Example: Lazy Binary Tree

- Variant of Contention-Friendly Tree [EuroPar'13, PPL'16]
- **Traversal does not take locks**
  - Used by contains, insert & delete
- Self balancing – rotation allocates a new node
- Marks logical & physical deletion by separate bits

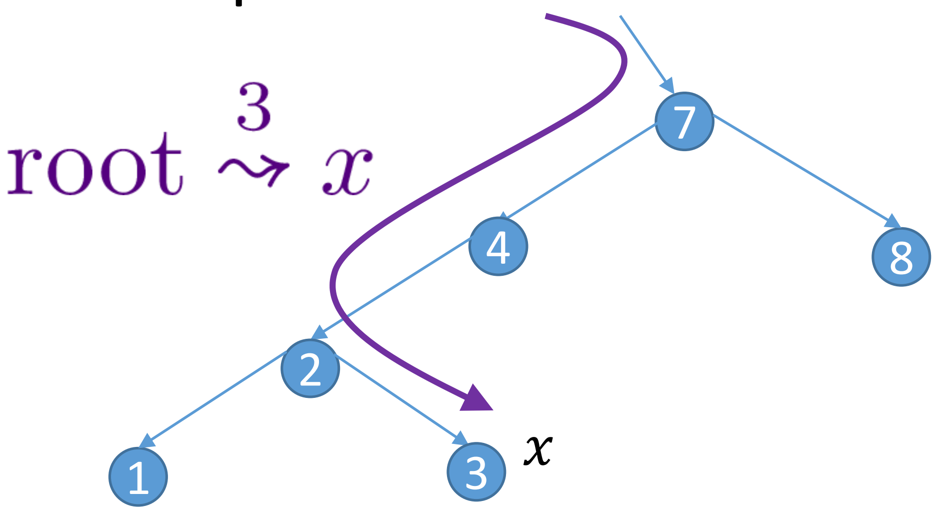


Linearizability Proof Uses Paths

- A tree represents a set using search paths:

$$\mathcal{A}(H) = \{k \in \mathbb{N} \mid \exists x.\ \mathrm{root} \overset{k}{\rightsquigarrow} x \wedge x.key = k \wedge \neg x.del\}$$

- Linearization point: set modified by at most one write in each operation



Linearizability Proof Uses Paths

- A tree represents a set using search paths:

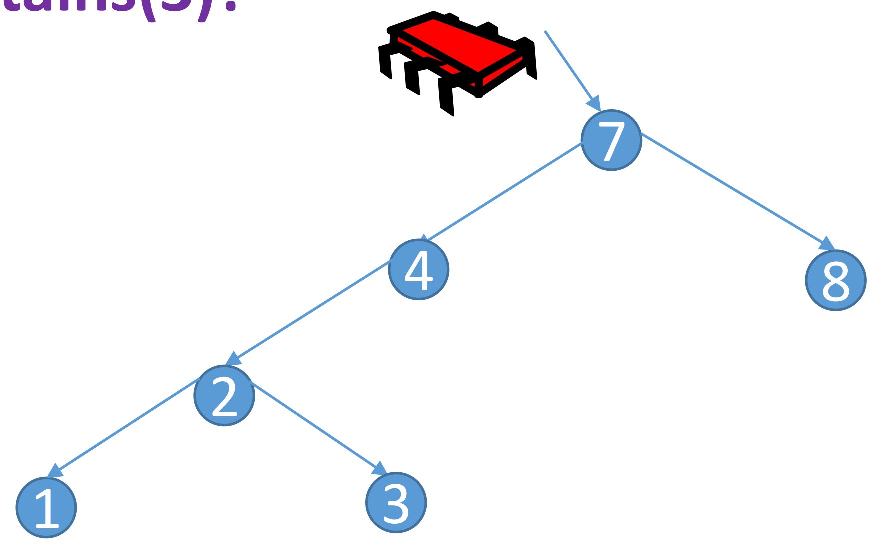
$$\mathcal{A}(H) = \{k \in \mathbb{N} \mid \exists x. \boxed{\mathrm{root} \overset{k}{\rightsquigarrow} x} \wedge x.key = k \wedge \neg x.del\}$$

Correct **traversals** required
for correct paths manipulation

- **Contains:** exists path to key/null at some point
- **Insert/delete:** add/remove path
- Maintenance operations (e.g. rotate):
  no observable effect

Traversals: Inconsistent View

contains(3)?

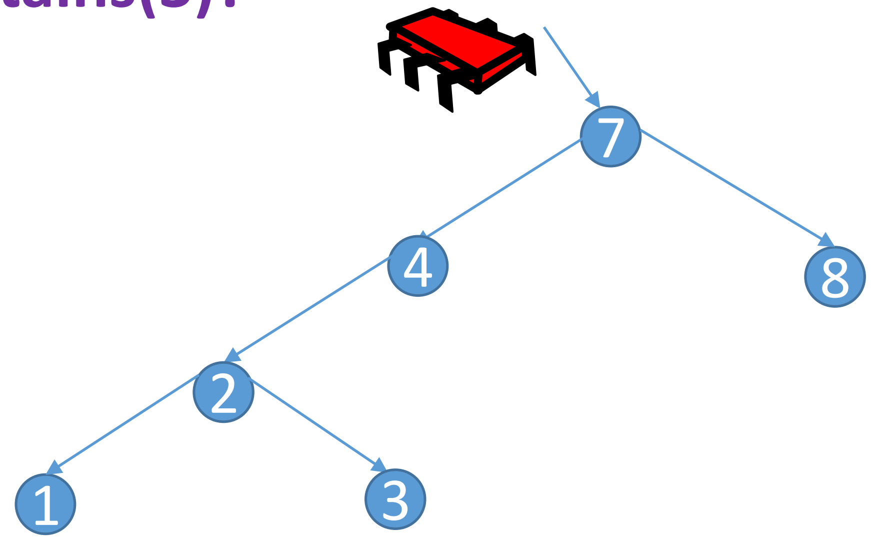


Traversals: Inconsistent View

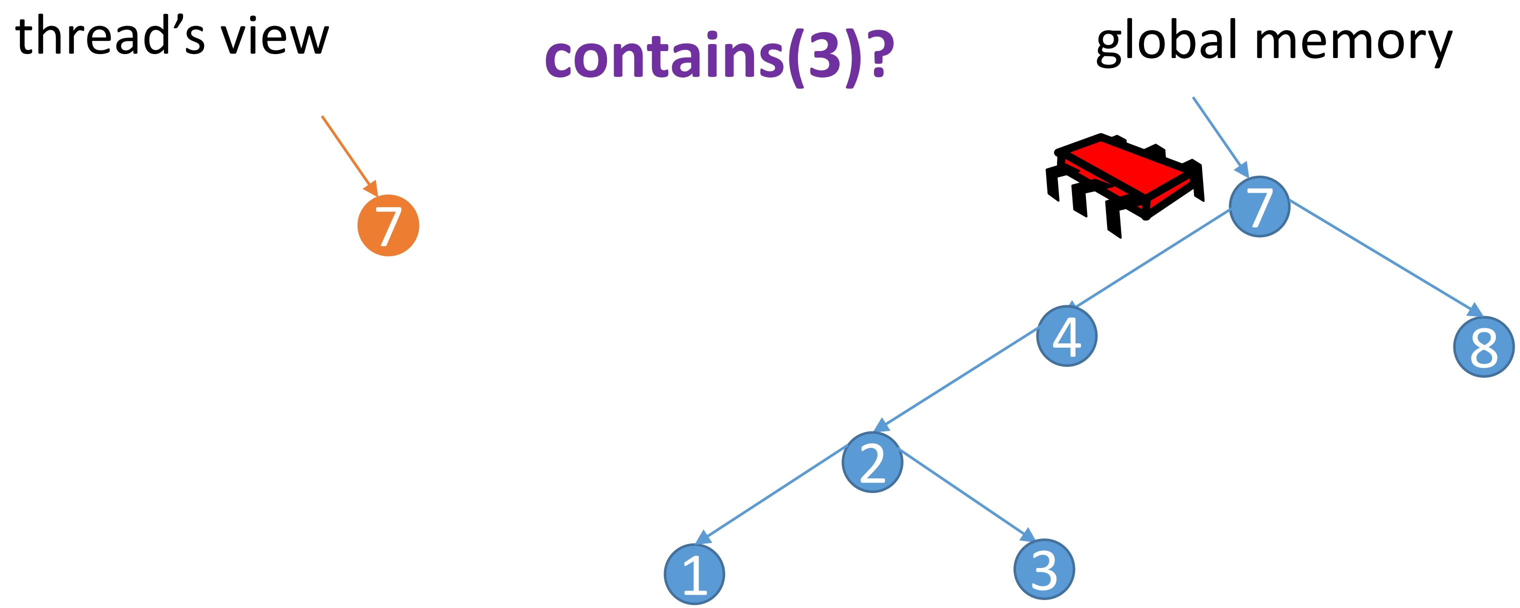
thread's view

contains(3)?

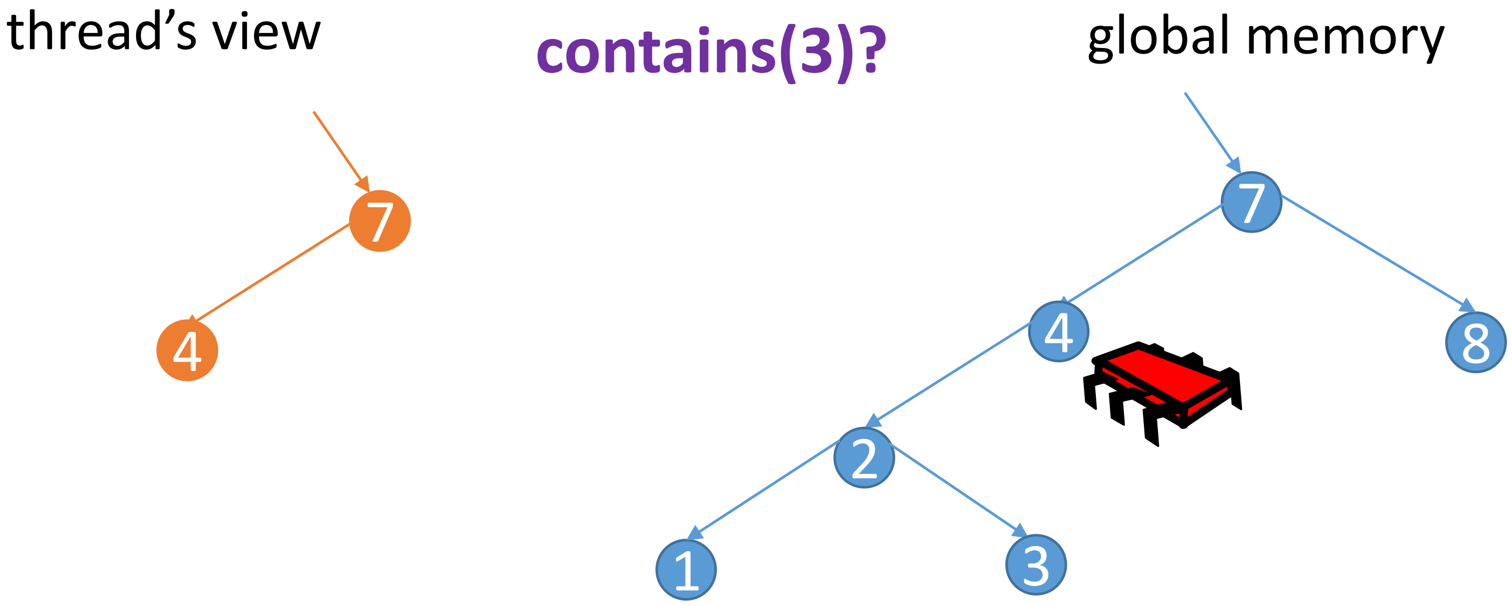
global memory



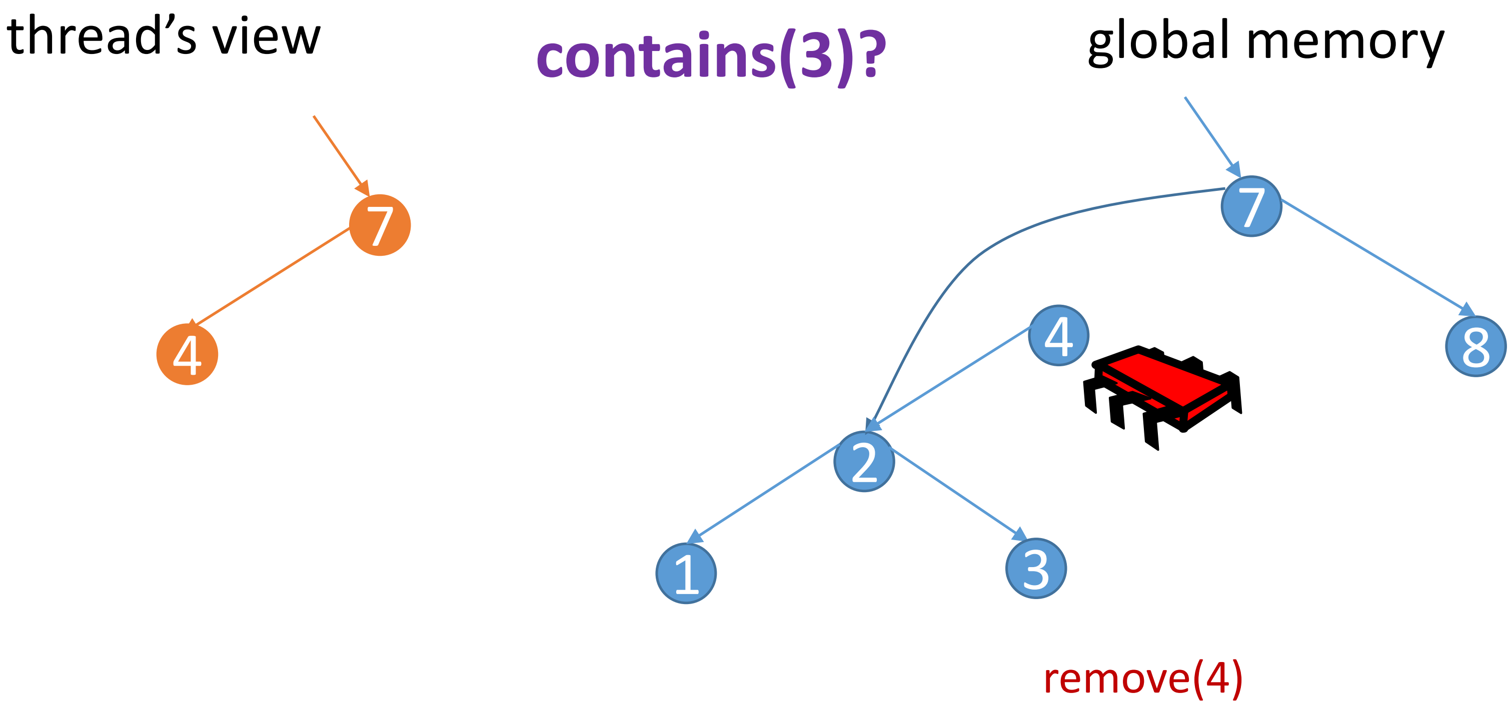
Traversals: Inconsistent View



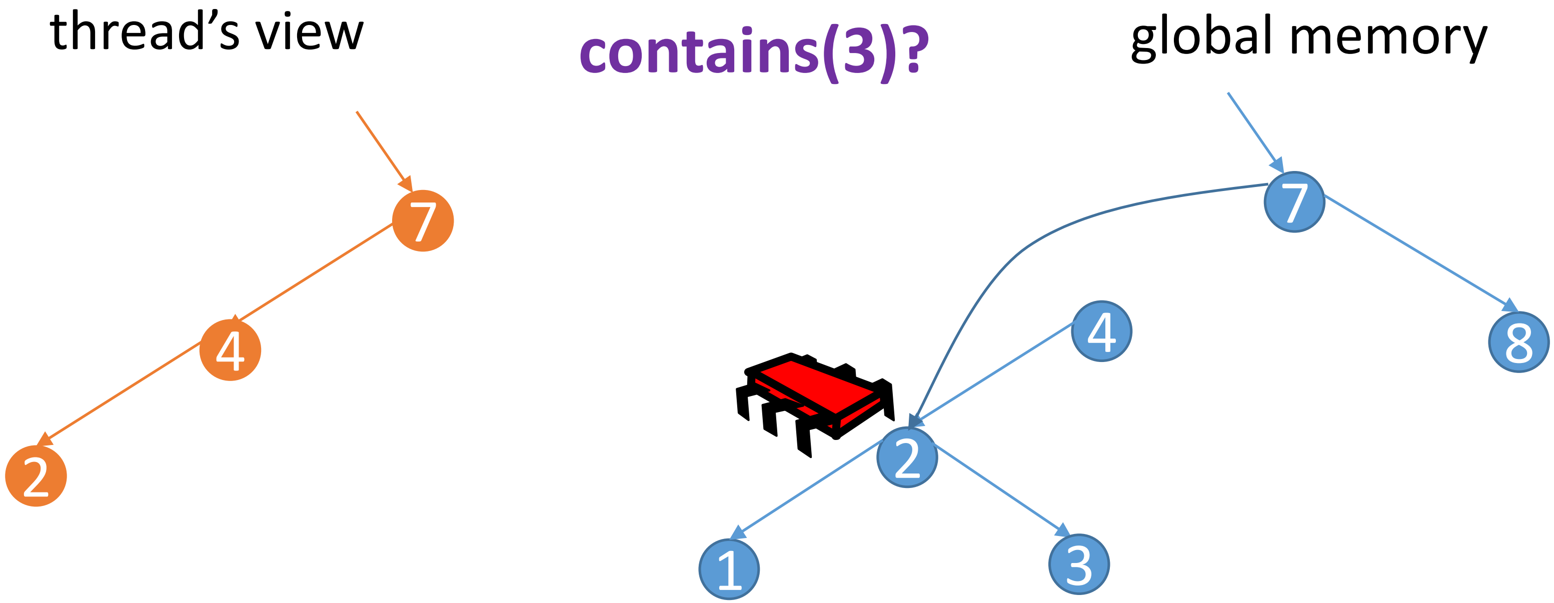
Traversals: Inconsistent View



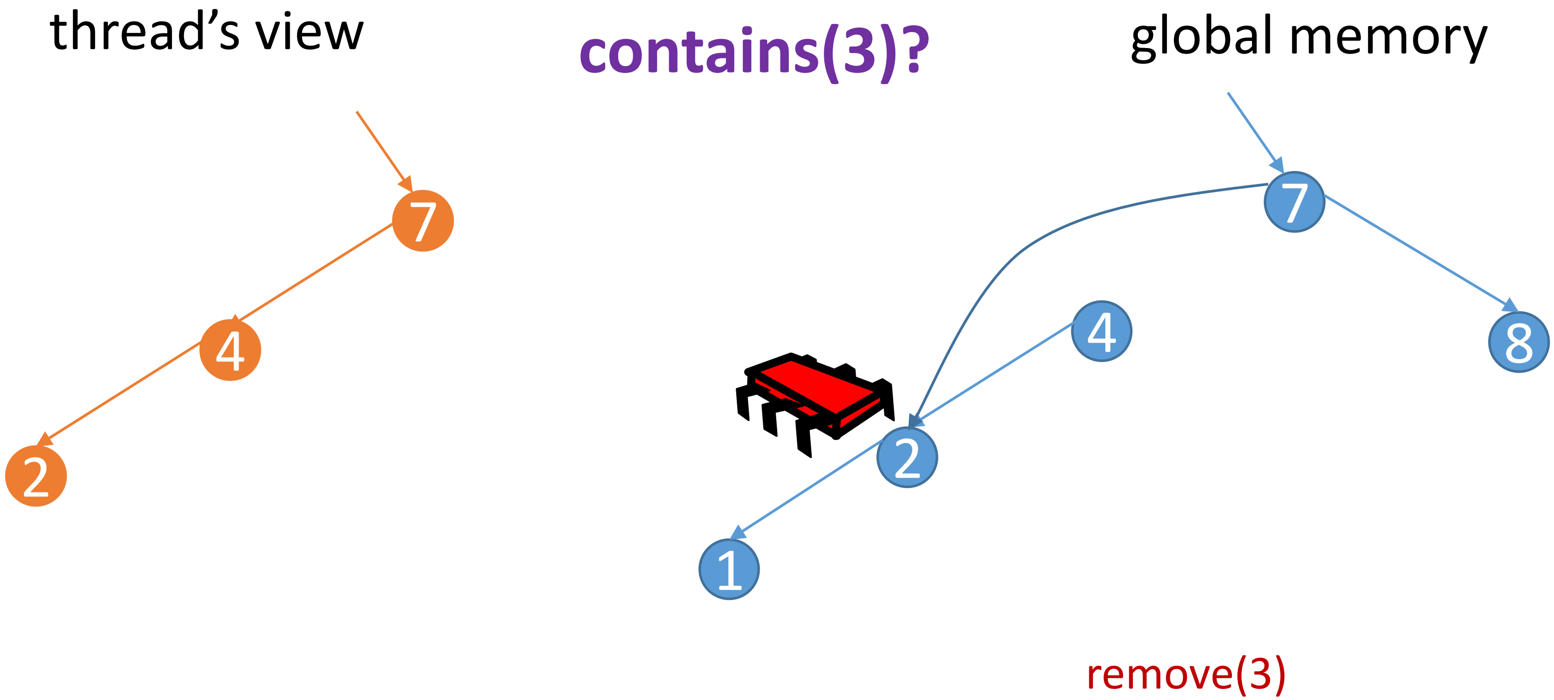
Traversals: Inconsistent View



Traversals: Inconsistent View

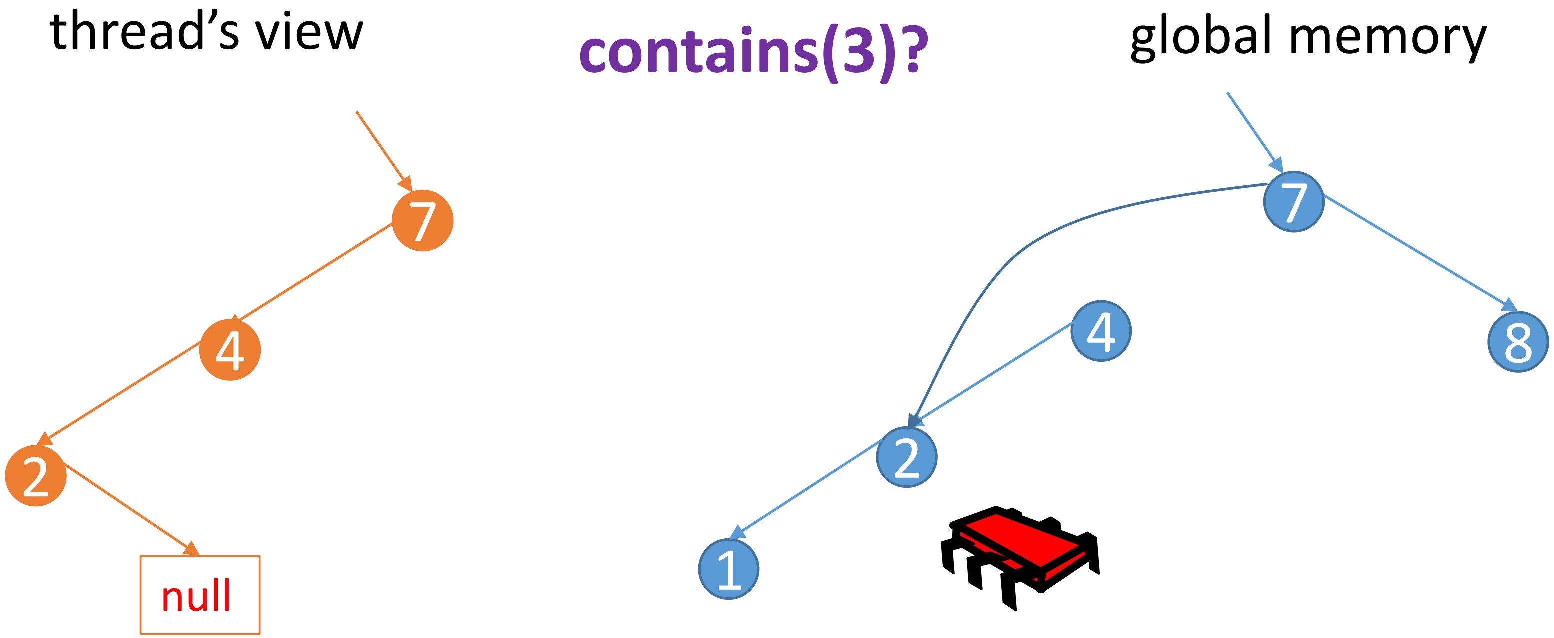


Traversals: Inconsistent View



Traversals: Inconsistent View

contains(3)?

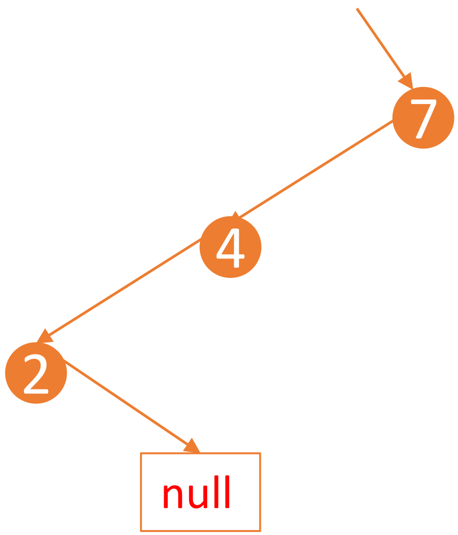


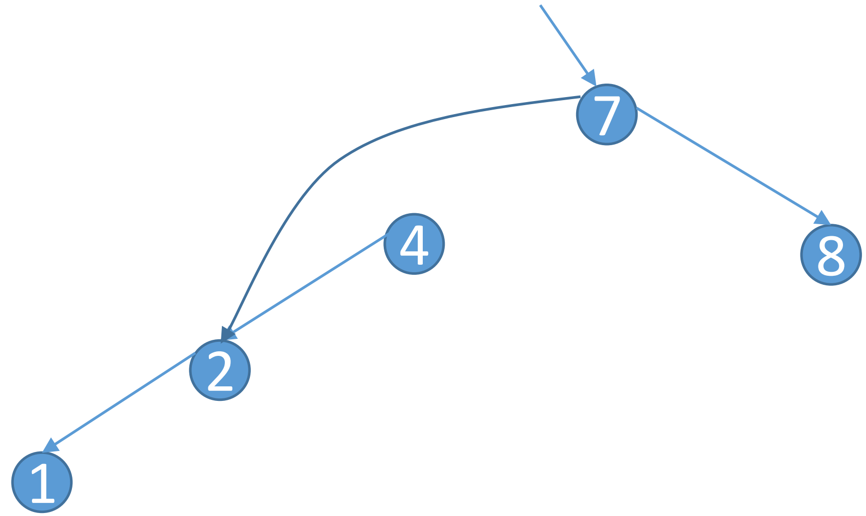
Traversals: Inconsistent View

thread's view

contains(3)? no

global memory





$$\text{root} \overset{k}{\rightsquigarrow} \mathit{null}$$

contains(3): **false**

$$\diamondminus\left(\text{root} \overset{k}{\rightsquigarrow} \mathit{null}\right)$$

contains(3): **false**

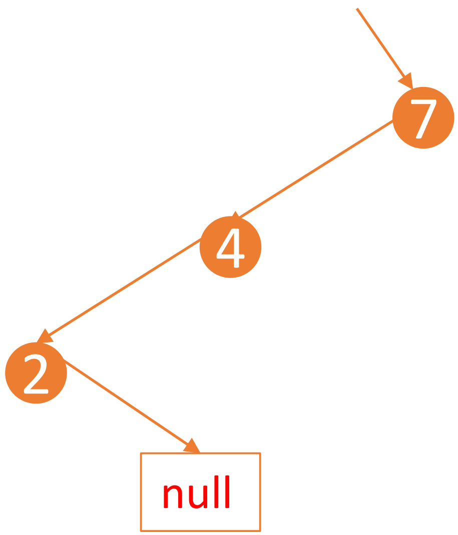
at some point

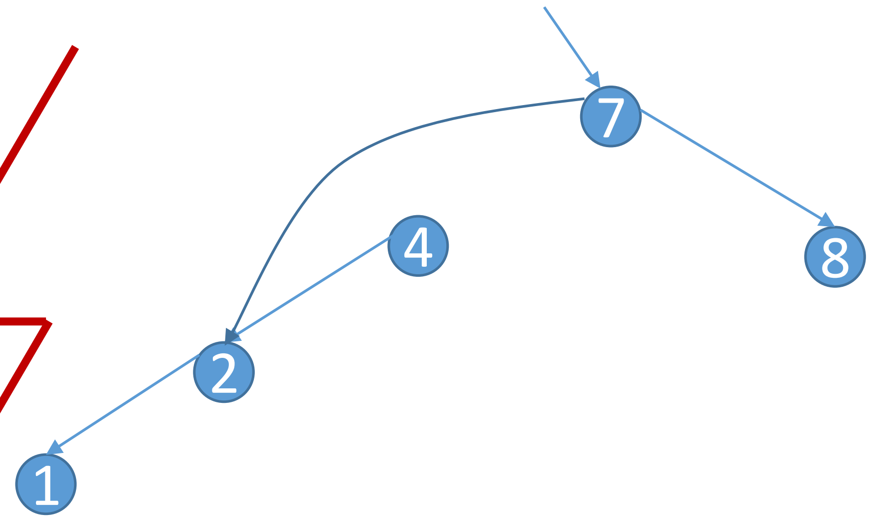
Traversals: Inconsistent View

thread's view

contains(3)? no

global memory





$$\mathrm{root} \overset{k}{\rightsquigarrow} null$$

contains(3): **false**

$$\diamondminus(\mathrm{root} \overset{k}{\rightsquigarrow} null)$$

contains(3): **false**

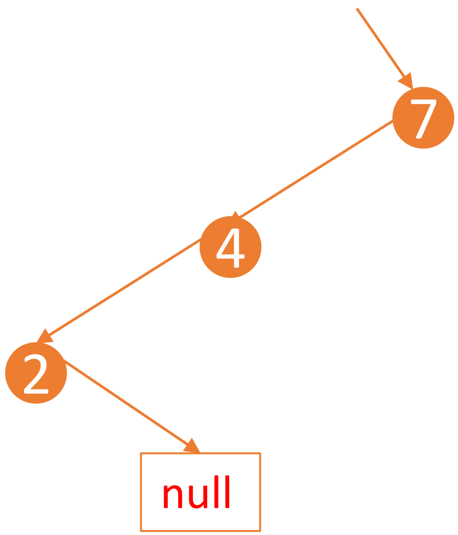
at some point

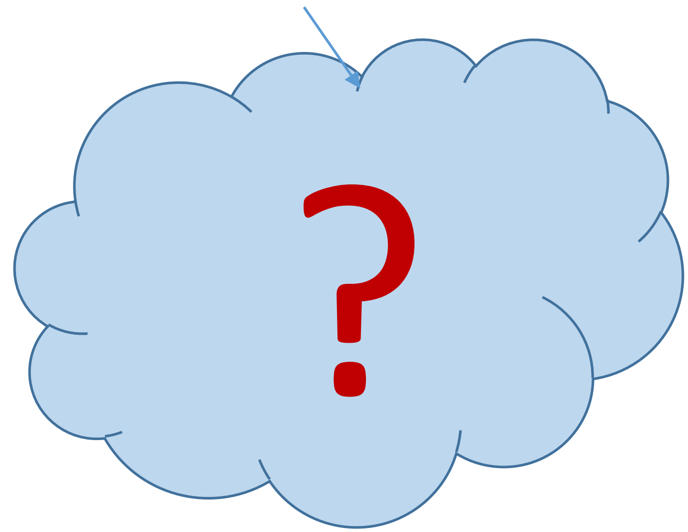
Traversals: Inconsistent View

thread's view

contains(3)? no

global memory





$$\mathrm{root} \overset{k}{\rightsquigarrow} \mathit{null}$$

contains(3): **false**

$$\diamondminus(\mathrm{root} \overset{k}{\rightsquigarrow} \mathit{null})$$

contains(3): **false**

at some point

Traversal Correctness Problem

Prove the **existence** of a **search path**
in **real memory**
at some point during the traversal

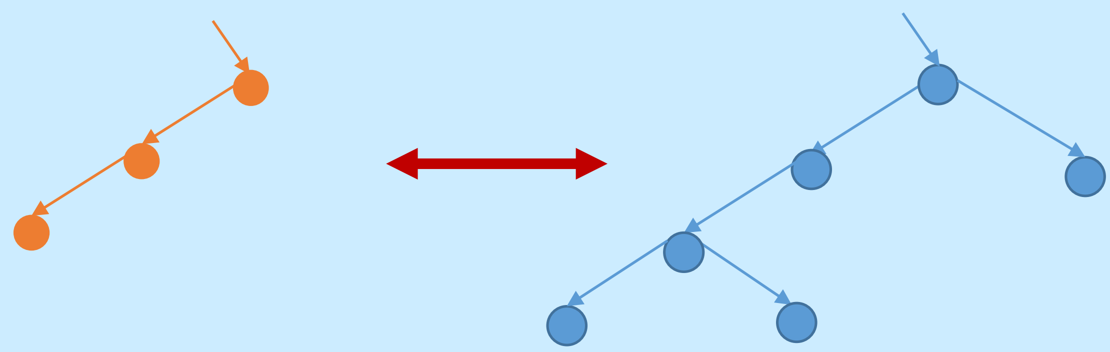
- The heart of the linearizability proof:
- **Contains:** $\diamondminus(\mathrm{root} \overset{k}{\leadsto} x)$
- **Insert:** $\diamondminus(\mathrm{root} \overset{k}{\leadsto} x)$+unmarked
- Full details in the paper

$\diamondminus(\mathrm{root} \overset{k}{\leadsto} x)$

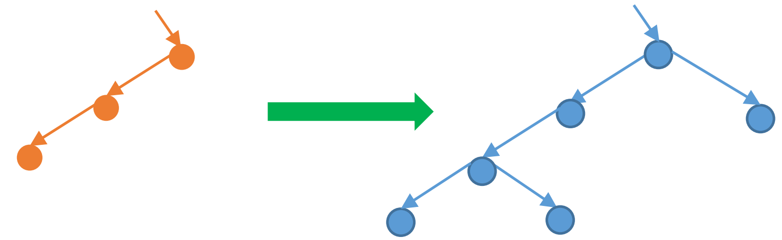
Traversal Correctness Problem

Prove the **existence** of a **search path**
in **real memory**
at some point during the traversal

How to prove traversal correctness?



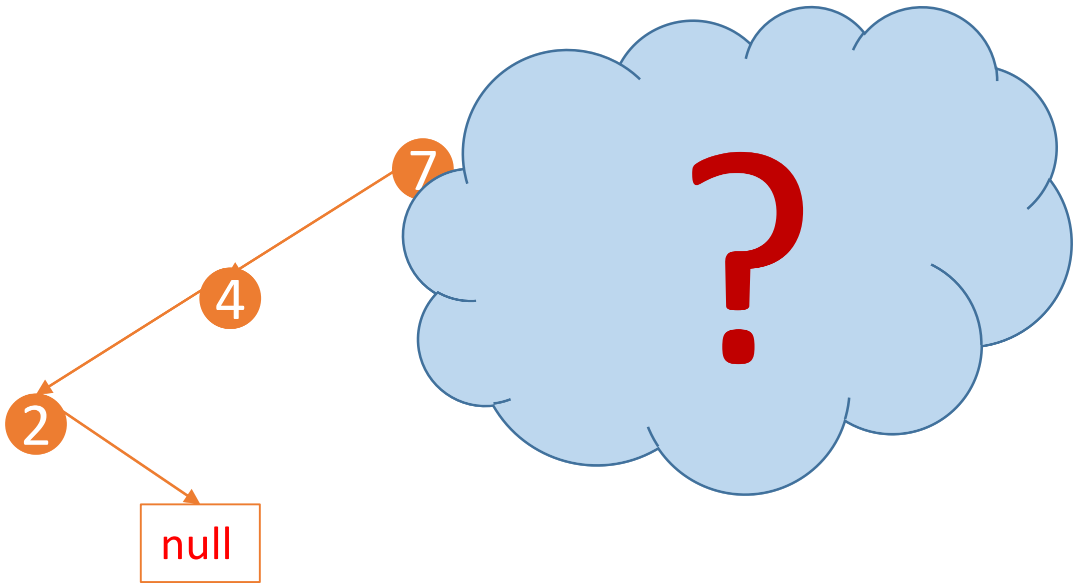
Our Approach



Traversal correctness **without interference** (sequential reasoning) → Traversal correctness **with interference** (concurrent reasoning)

- Apply the technique to prove the linearizability of
  1. Variant of Contention-Friendly self-balancing binary tree [EuroPar’13, PPL’16]
  2. Lazy list and Optimistic list [OPODIS’05]
  3. Lock-free list
  4. Lock-free skiplist

Local View Argument

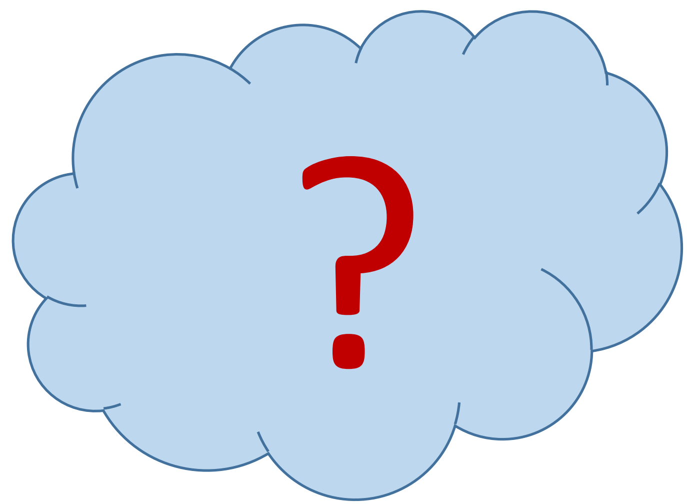


$$\Diamond(\mathrm{root} \overset{k}{\rightsquigarrow} x)$$

$$\mathrm{root} \overset{k}{\rightsquigarrow} x \quad \Longrightarrow$$

Local View Argument

Sequential reasoning to show traversal correct without interference



$$\mathrm{root} \overset{k}{\leadsto} x \quad \Longrightarrow \quad \diamondminus(\mathrm{root} \overset{k}{\leadsto} x)$$

Local View Argument

Sequential reasoning to show traversal correct without interference

Correctness of concurrent traversals

$$\mathrm{root} \overset{k}{\leadsto} x \quad \Longrightarrow \quad \Diamond(\mathrm{root} \overset{k}{\leadsto} \mathit{null})$$

Local View Argument

Order & Preservation

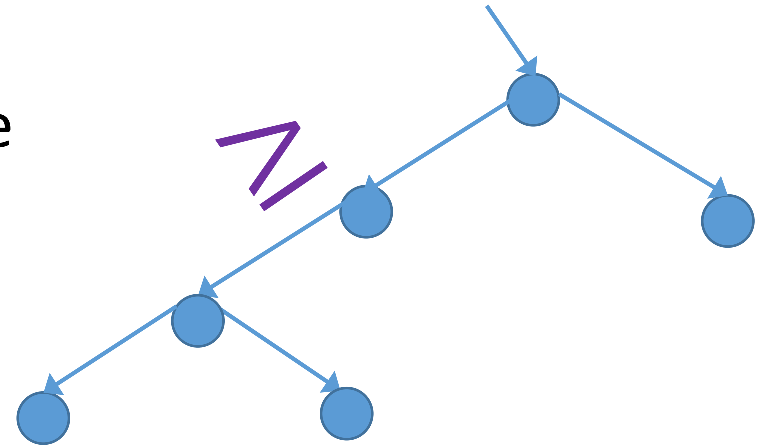
Sequential reasoning to show traversal correct without interference

Correctness of concurrent traversals

$$\mathrm{root} \overset{k}{\rightsquigarrow} x \quad \Longrightarrow \quad \diamondminus(\mathrm{root} \overset{k}{\rightsquigarrow} null)$$

Requirement 1: Order on Memory

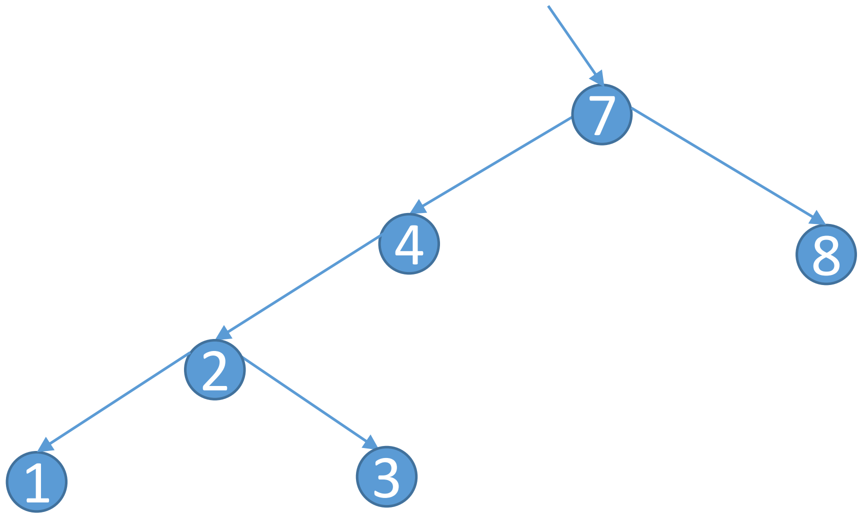
- Partial order $\leq_H$ on **memory** induced by the **pointers** in the global state H (**not** order on the data)



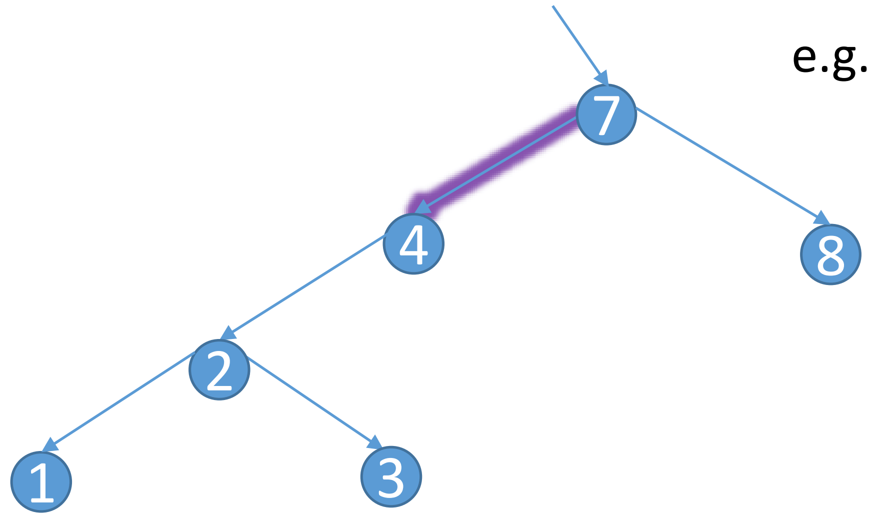
- Reads follow the order
- Search paths follow the order
- **Temporal acyclicity**: the order is not violated across intermediate states

Requirement 1: Temporal Acyclicity

Requirement 1: Temporal Acyclicity

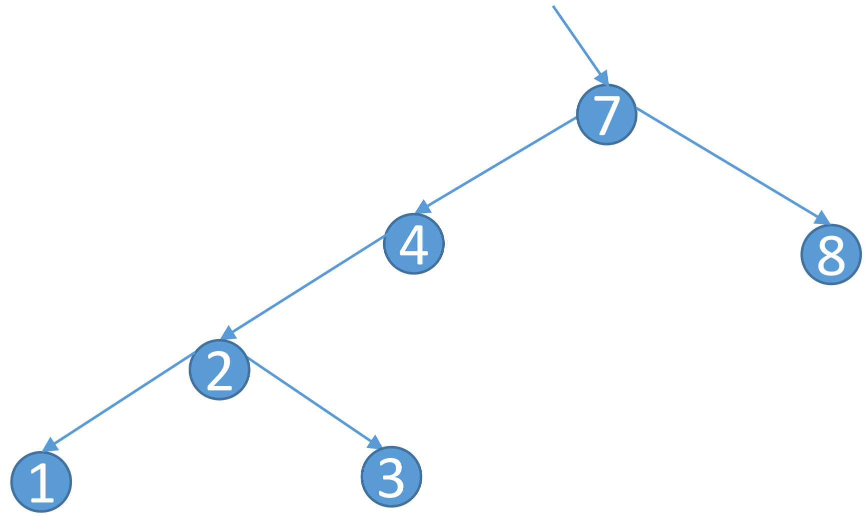


Requirement 1: Temporal Acyclicity

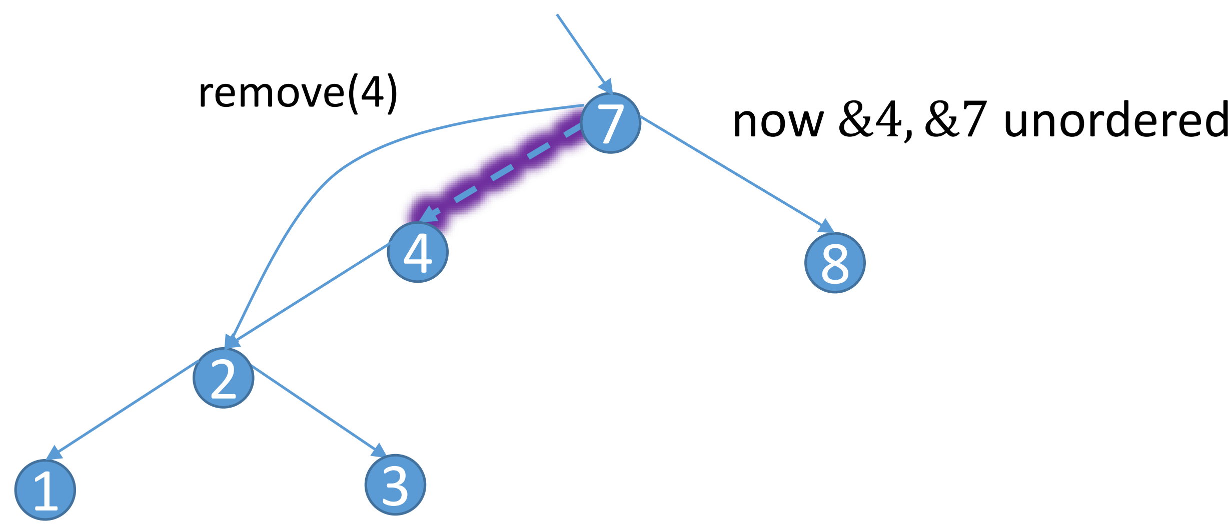


e.g. $\&4 > \&7$

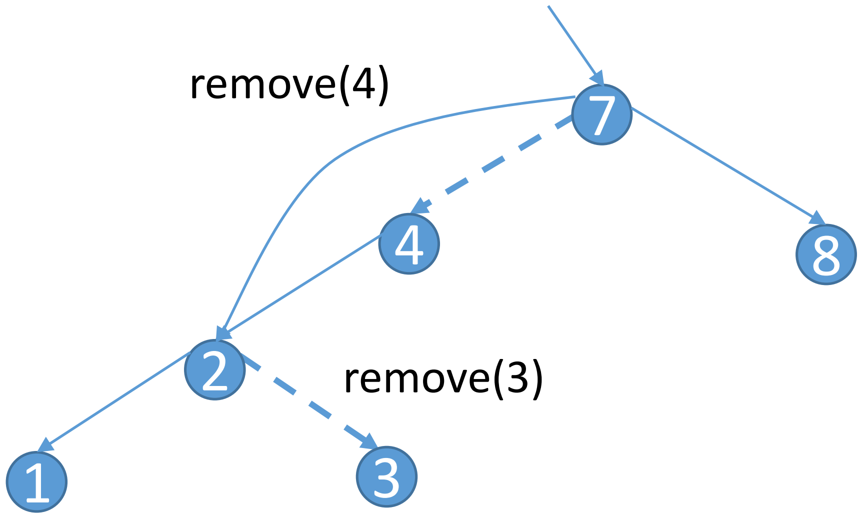
Requirement 1: Temporal Acyclicity



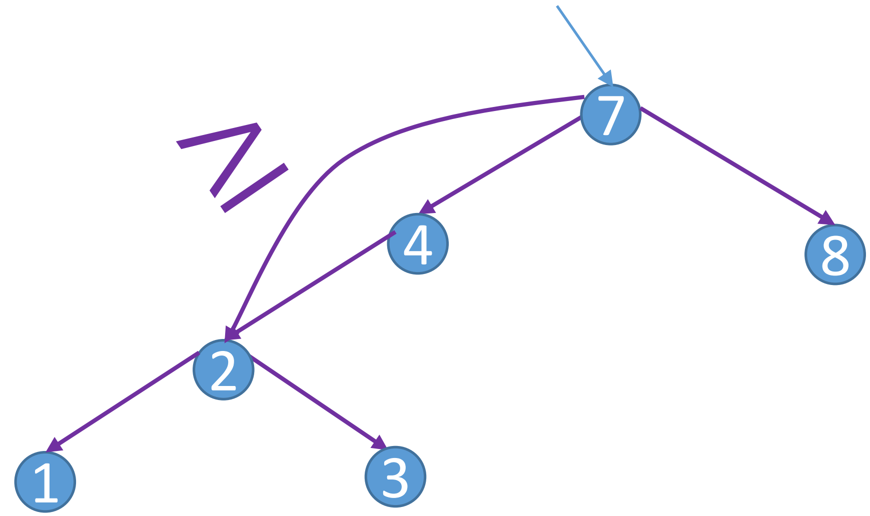
Requirement 1: Temporal Acyclicity



Requirement 1: Temporal Acyclicity



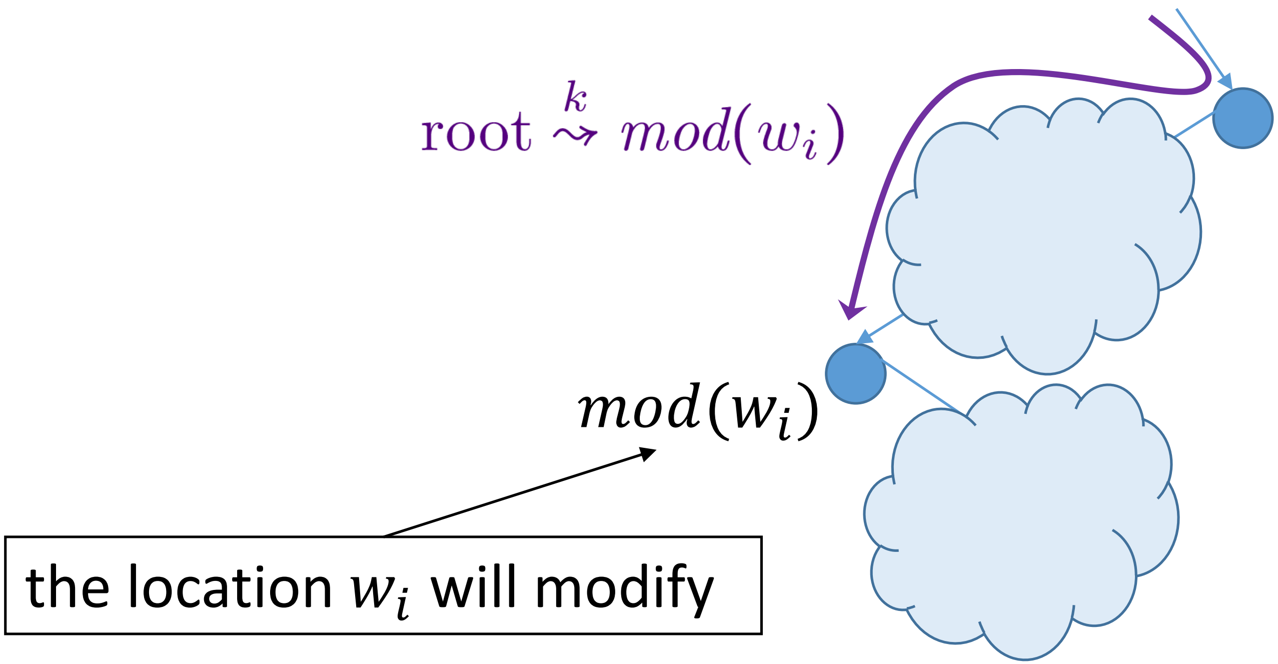
Requirement 1: Temporal Acyclicity



<u>Requirement</u>: no cycles in accumulated order

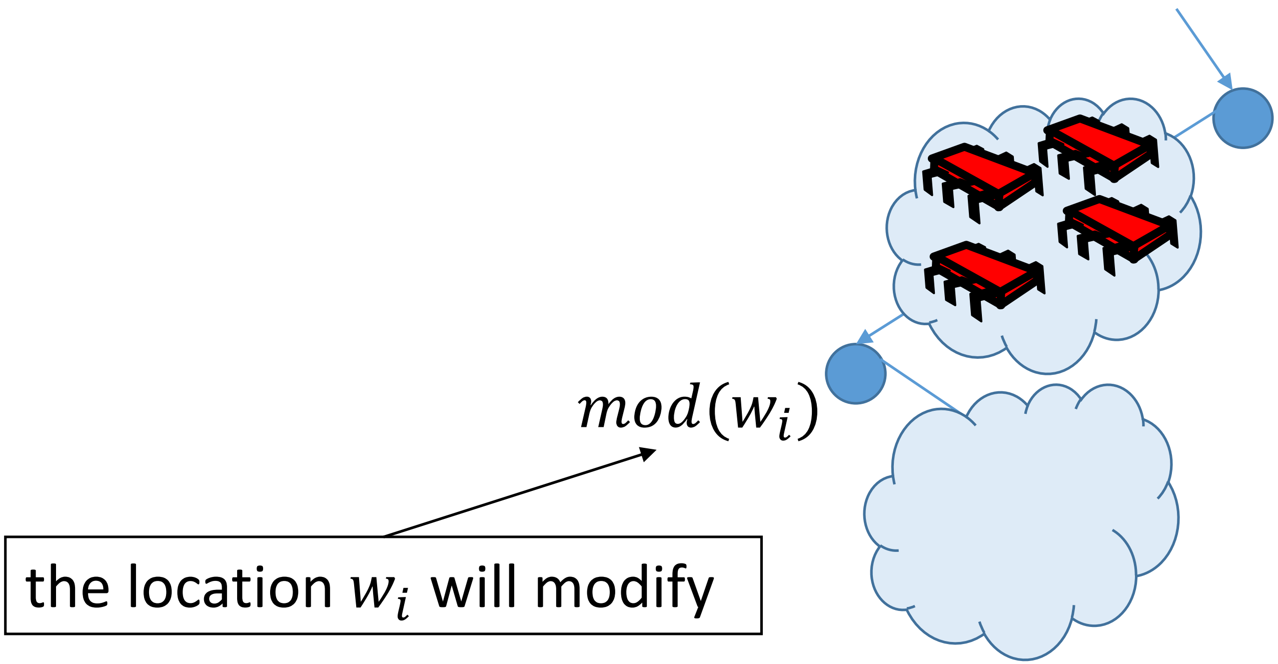
Requirement 2: Preservation

Writes do not destroy search paths to locations to which there could be later writes



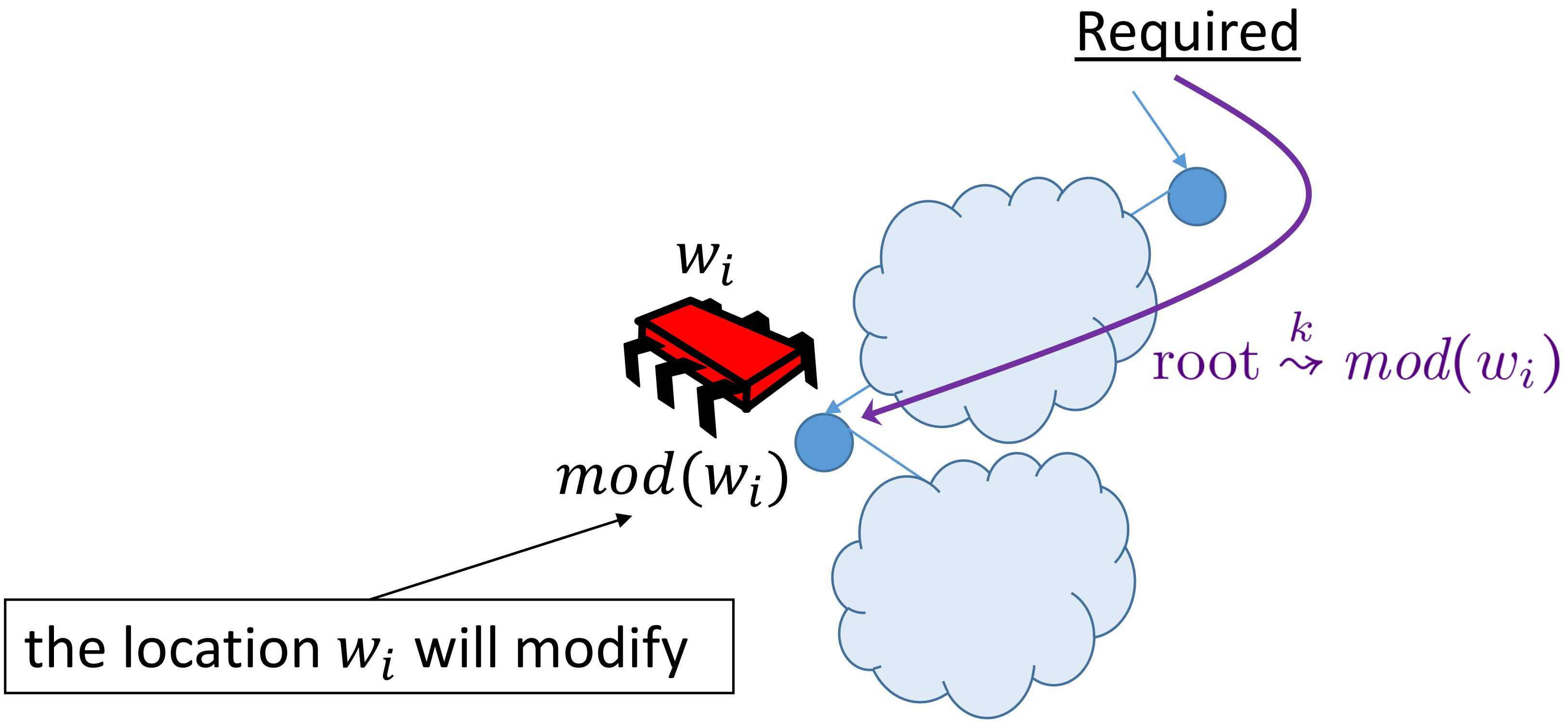
Requirement 2: Preservation

Writes do not destroy search paths to locations to which there could be later writes



Requirement 2: Preservation

Writes do not destroy search paths to locations to which there could be later writes



Requirement 2: Preservation

Writes do not destroy search paths to locations to which there could be later writes

Required

In example: nodes are **marked removed** when their **search reachability is reduced**;
writers lock and **check the marked** bit

$\overset{k}{\rightsquigarrow} mod(w_i)$

$mod(w_i)$

the location w_i will modify

Local View Argument

Order & Preservation

Sequential reasoning to show traversal correct without interference

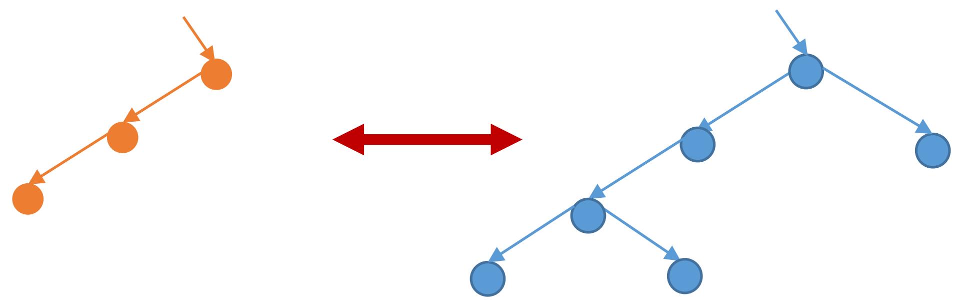
Correctness of concurrent traversals

$$\mathrm{root} \overset{k}{\rightsquigarrow} x \quad \Longrightarrow \quad \diamondminus(\mathrm{root} \overset{k}{\rightsquigarrow} x)$$

Idea: Consistency from Ordering

- Run with no interference on a related heap

thread's view

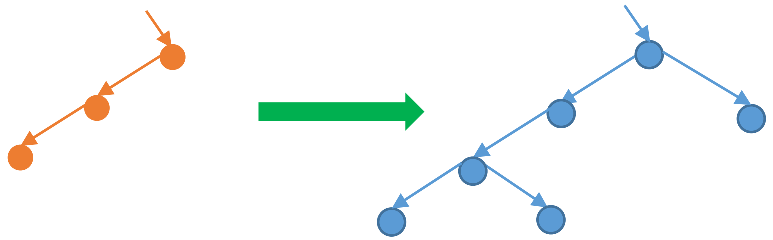


global memory

$H_r = w_{i_1} \ldots w_{i_k}(H_0)$ $\quad$ $H_c = w_1 \ldots w_n(H_0)$

- Writes on H_r are forward-agreeing with H_c
- Preservation complements the backwards path
- Full details in the paper

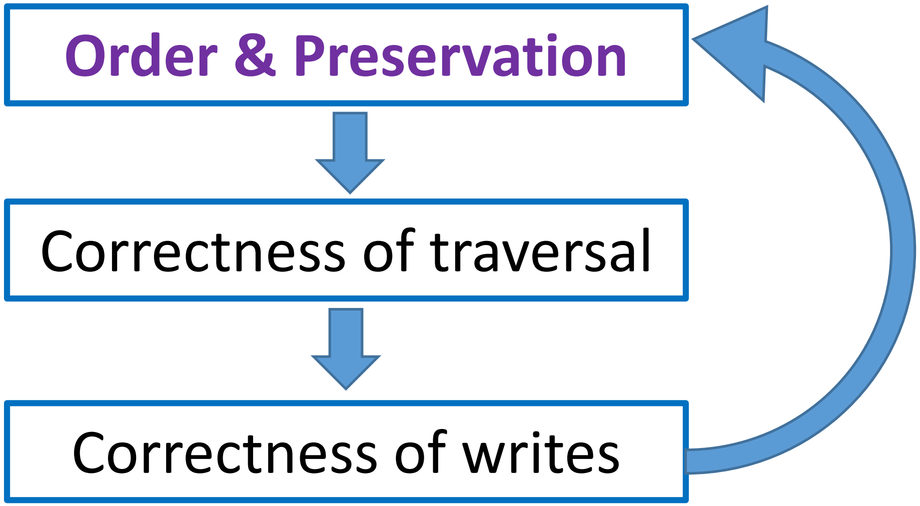
Summary



- Concurrent traversal correctness: Existence of search path in the past
- **Order** & **preservation** facilitate proof by **sequential reasoning**
- Reasoning on interleavings of writes completes the linearizability proof
- Challenges: backtracking, in-traversal validation

Establishing the Conditions

- In our examples, the proof of the conditions is trivial
- Valid to rely on local view arguments by induction on the sequence of writes



Some Related Work

- General linearizability proof methods and program logics – see paper for references
- Peter W. O'Hearn, Noam Rinetzky, Martin T. Vechev, Eran Yahav, Greta Yorsh:
  Verifying linearizability with hindsight. PODC 2010: 85-94
    - Proof of the lazy list by the hindsight lemma
    - Specific instance of our extension to path+field
- Kfir Lev-Ari, Gregory V. Chockler, Idit Keidar:
  A Constructive Approach for Proving Data Structures' Linearizability. DISC 2015: 356-370
    - Uses sequential reasoning on the data structure
    - Relies on **base points** – which our technique can establish
- Dennis E. Shasha, Nathan Goodman:
  Concurrent Search Structure Algorithms. ACM Trans. Database Syst. 13(1): 53-90 (1988)
    - Framework for proving linearizability of search data structures
    - 3 templates for traversal correctness, proofs require concurrent reasoning on interleavings of reads and writes

References

- [Fraser-04] Keir Fraser:
  Practical lock-freedom. PhD thesis
- [OPODIS’05] Steve Heller, Maurice Herlihy, Victor Luchangco, Mark Moir, William N. Scherer III, Nir Shavit:
  A Lazy Concurrent List-Based Set Algorithm. OPODIS 2005: 3-16
- [PODC’10] Faith Ellen, Panagiota Fatourou, Eric Ruppert, Franck van Breugel:
  Non-blocking binary search trees. PODC 2010: 131-140
- [SPAA’12] Shane V. Howley, Jeremy Jones:
  A non-blocking internal binary search tree. SPAA 2012: 161-171
- [EuroPar’13] Tyler Crain, Vincent Gramoli, Michel Raynal:
  A Contention-Friendly Binary Search Tree. Euro-Par 2013: 229-240
- [PPOPP’14b] Trevor Brown, Faith Ellen, Eric Ruppert:
  A general technique for non-blocking trees. PPOPP 2014: 329-342
- [PPOPP’14a] Aravind Natarajan, Neeraj Mittal:
  Fast concurrent lock-free binary search trees. PPOPP 2014: 317-328
- [ICDCN’15] Arunmoezhi Ramachandran, Neeraj Mittal:
  A Fast Lock-Free Internal Binary Search Tree. ICDCN 2015: 37:1-37:10
- [PPL’16] Tyler Crain, Vincent Gramoli, Michel Raynal:
  A Fast Contention-Friendly Binary Search Tree. Parallel Processing Letters 26(3): 1-17 (2016)