The Amortized Analysis of a Non-blocking Chromatic Tree

Jeremy Ko
University of Toronto
OPODIS 2018

Motivation

Show that an implementation of a non-blocking balanced binary search tree has good amortized step complexity.

- Amortized step complexity = $\frac{\text{max number of steps performed in an execution}}{\text{number of operations invoked}}$

- Non-blocking: whenever there are active operations, one operation will eventually complete in a finite number of steps

Asynchronous Shared Memory Model

- Processes communicate using read, write, and CAS primitives
- Processes may crash

- Point Contention
  - $\dot{c}(op) =$ the maximum number of active operations at a point during operation op
  - $\dot{c}(\alpha) =$ the maximum number of active operations at a point during execution α

Related Work

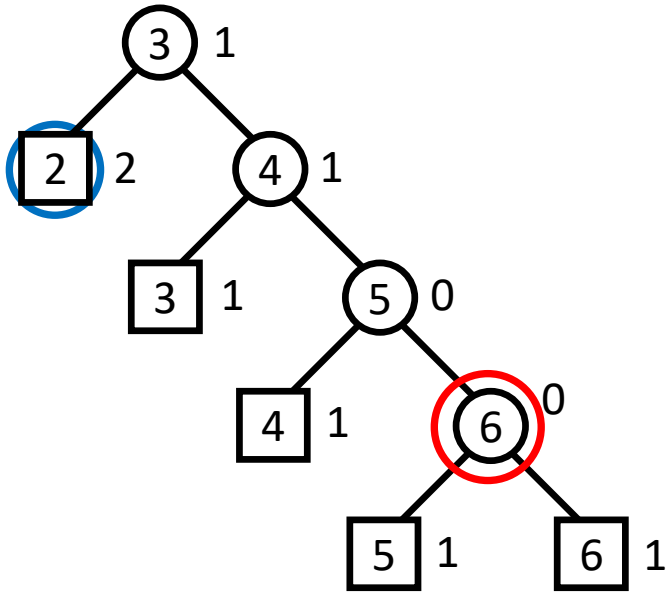
- Amortized analysis of non-blocking data structures:
  - [FR'04] Linked list of length $L(op)$: $O(\dot{c}(op) + L(op))$
  - [EFHR'14] Unbalanced binary search tree of height $h(op)$: $O(\dot{c}(op) + h(op))$

- Chromatic trees:
  - [NS'96] Introduce sequential chromatic tree
  - [BFL'95] Amortized analysis of a sequential chromatic tree
  - [BER'14] Implementation of a non-blocking chromatic tree using LLX and SCX

The Chromatic Tree

- A data structure for a dynamic set of elements, supporting $\textsc{Insert}(k)$, $\textsc{Delete}(k)$, and $\textsc{Find}(k)$
- Leaf-oriented binary tree with the following balance conditions:
  - The weights of leaves are not 0.
  - The sum of weights on the path from the root to each leaf are all the same.

- A node x with weight 0 and a parent with weight 0 has a red-red violation.
- A node x with weight $w > 1$ has $w - 1$ overweight violations.



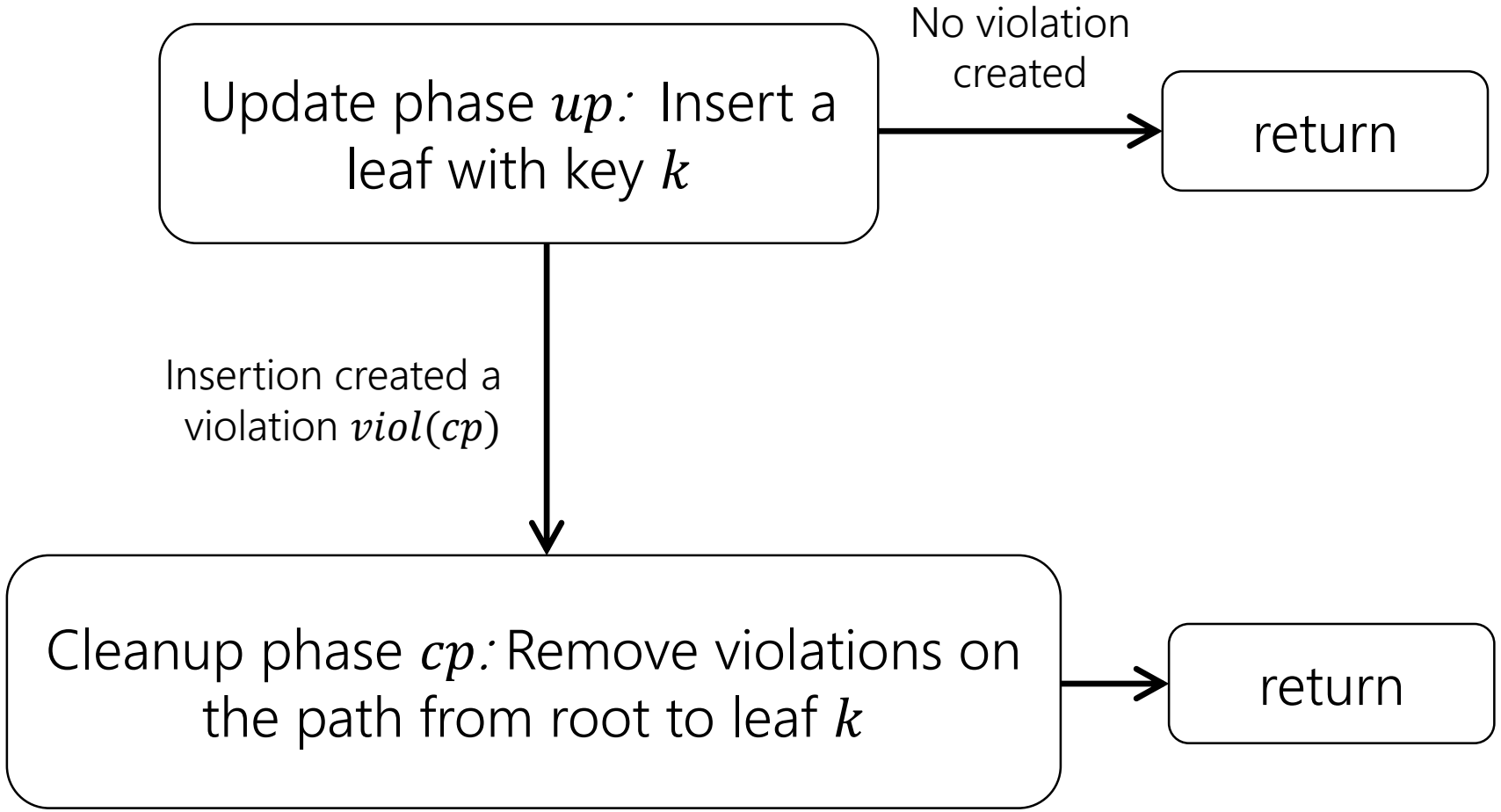
[BER'14] If there are c operations in progress and the chromatic tree contains n nodes, then its height is $O(c + \log n)$.

Challenges of the Amortized Analysis

- 11 different types of rebalancing transformations
- Update and rebalancing transformations may be concurrent
- The type and number of rebalancing transformations a process performs during an operation can depend on how the tree is changed by other processes
- Processes with high contention may perform an excessive amount of rebalancing

[BER'14] Chromatic Tree Implementation

- Insert(k) operation:



[BER'14] Chromatic Tree Implementation

- Cleanup phase cp divided into attempts, where each attempt consists of the following steps:
  1. Search the tree for the key k, pushing visited nodes on a stack
  2. If a node v containing a violation is found
     - Attempt to remove the violation using a rebalancing transformation
     - Backtrack to a node that is still in the tree
  3. If a leaf without a violation is found, return.

- Before cp finishes, the violation, $viol(cp)$, created during the update phase of cp's operation, is removed.

Main Theorem

- $n(op)$ = maximum number of elements in the chromatic tree during op

Theorem: The amortized number of steps made by an operation op in an execution α is $O(\dot{c}(\alpha) + \log n(op))$.

Amortized Analysis Overview

- Steps taken during update phases and cleanup phases can be counted separately

- The steps taken during a cleanup phase cp:

$$steps(cp) = O\big(attempts(cp) + pushes(cp)\big).$$

Steps taken during transformation attempts

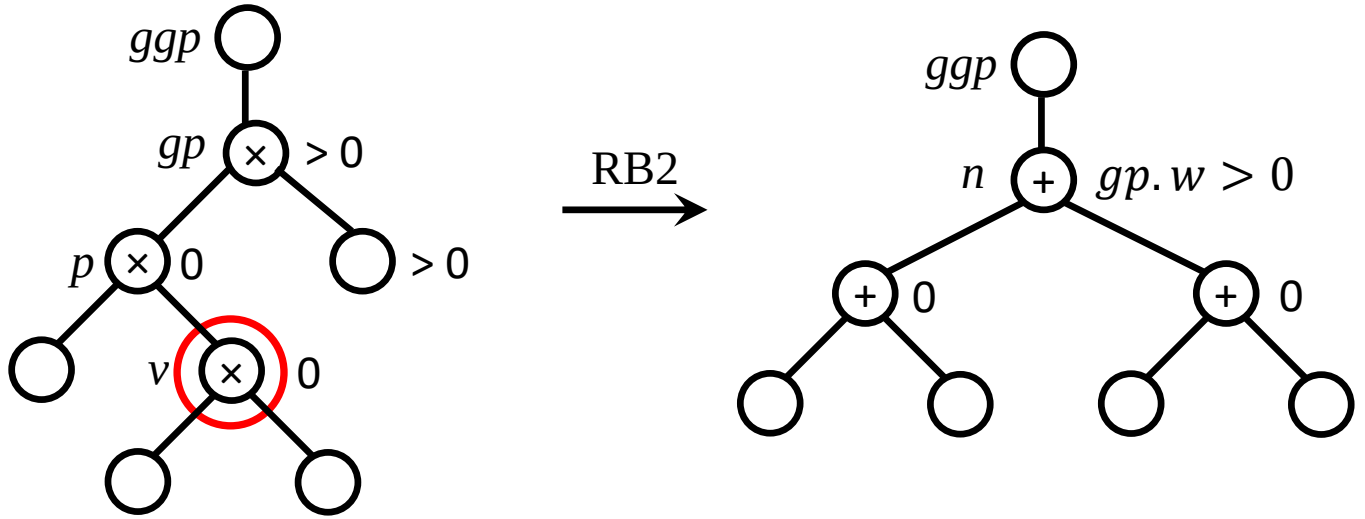
Steps taken during searches

Rebalancing Transformations

- Every rebalancing transformation is centered at a particular violation.

- Let $rebal\big(viol(cp)\big)$ be the number of successful rebalancing transformations centered at the violation, $viol(cp)$, created by cp's operation.

Example: RB2 transformation centered at the red-red violation at node v.



Amortized Analysis Overview

- The steps taken during a cleanup phase cp is:

$$steps(cp) = O\big(attempts(cp) + pushes(cp)\big).$$

- Show that

$$\sum_{cp \in \alpha} pushes(cp) \leq \sum_{cp \in \alpha} [3attempts(cp) + h(cp) + 10\dot{c}(cp) \cdot rebal\big(viol(cp)\big)] \qquad (1)$$

$$\sum_{cp \in \alpha} attempts(cp) \leq \sum_{cp \in \alpha} [78h(cp) + 312\dot{c}(cp) + 5008\dot{c}(cp) \cdot rebal\big(viol(cp)\big) + 357] \qquad (2)$$

Amortized Analysis Overview

[BFL'95] If $i > 0$ insertions and d deletions are performed on an initially empty chromatic tree, then at most $3i + d - 2$ rebalancing transformations occur.

Thus,

$$\sum_{cp \in \alpha} rebal\big(viol(cp)\big) \leq 3i + d - 2\,.$$

This, (1), and (2) imply

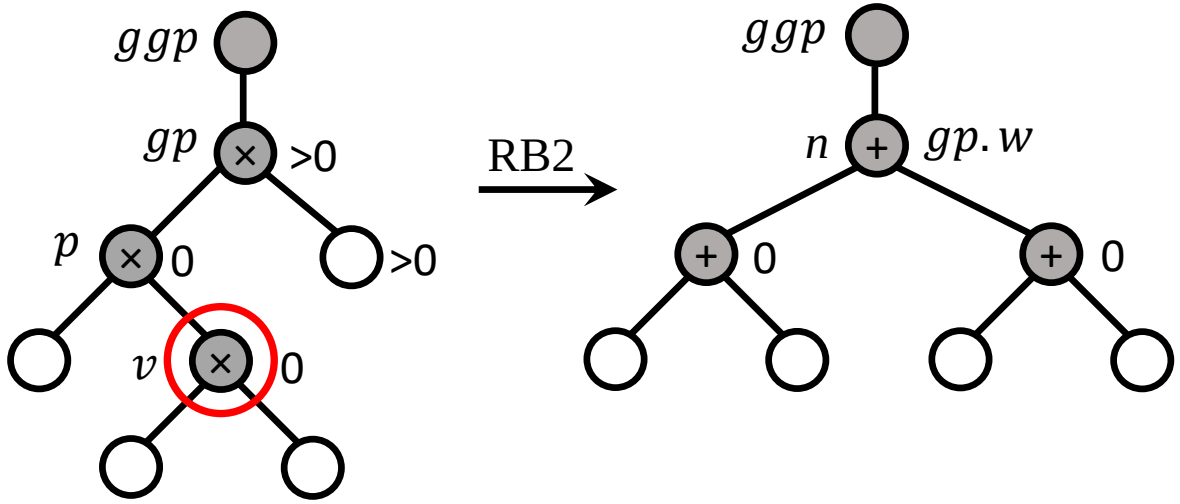
$$\sum_{op \in \alpha} steps(op) = O\left(\sum_{op \in \alpha} [\dot{c}(\alpha) + \log n(op)]\right).$$

Rebalancing Transformations

[BER'14] LLX and SCX primitives used to perform tree transformations guarantee non-blocking progress.

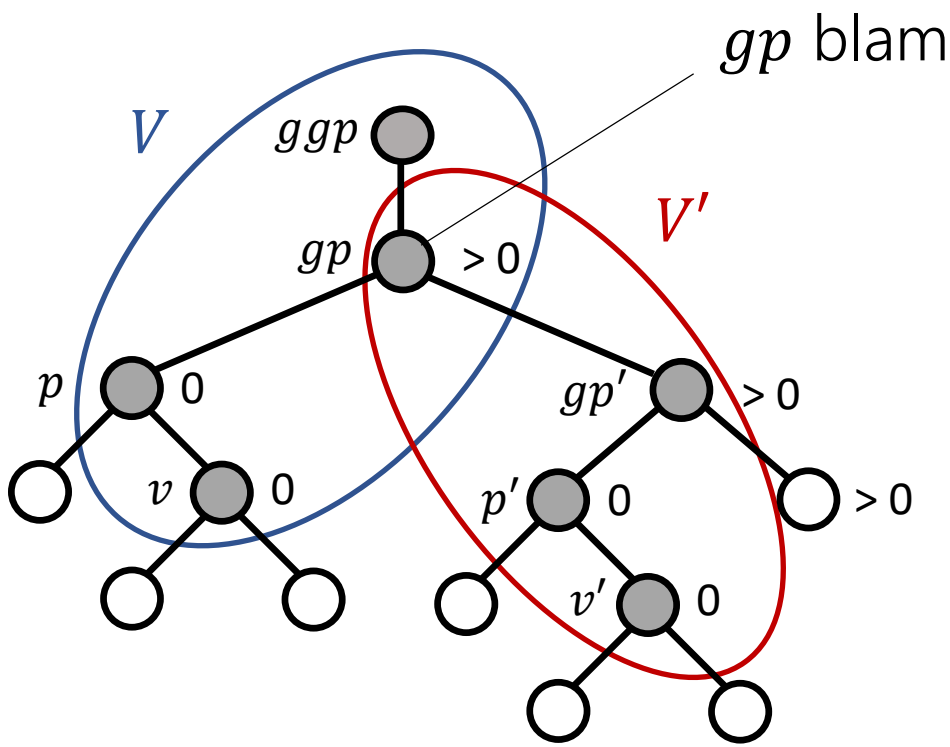
TryRebalance(V), where $V = \{ggp, gp, p, v\}$:

1. Mark gp, p, and v for removal
2. Update a child pointer of ggp to point to a new node n



Unsuccessful Rebalancing Transformations

- An instance of TryRebalance(V) may fail due to an instance of TryRebalance(V') by a different process, where $V \cap V' \neq \emptyset$, and blames some node $x \in V'$.
- A failed TryRebalance(V) aborts its current transformation, and may help the concurrent TryRebalance(V') either complete or abort.

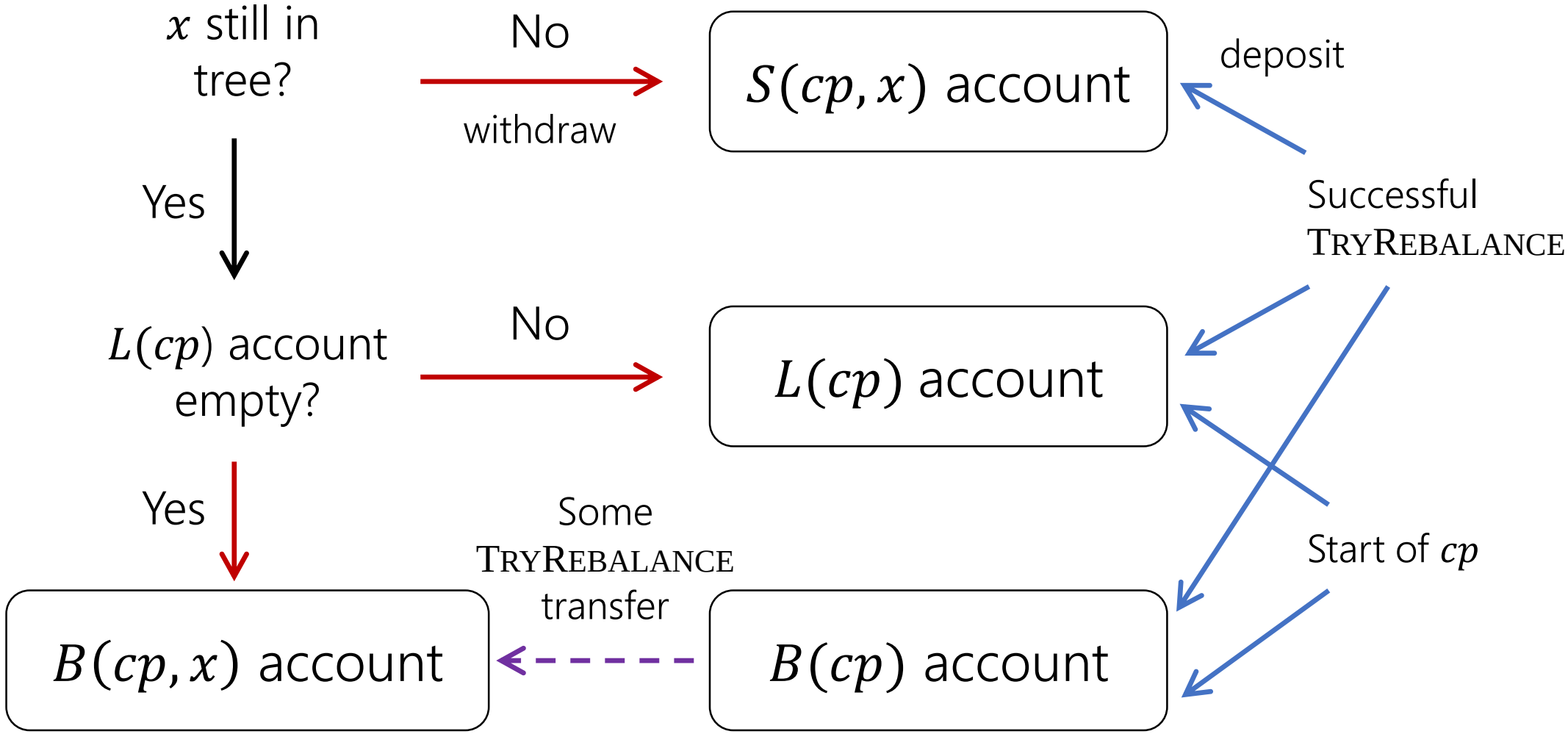


Two overlapping RB2 transformations

Accounting Method Overview

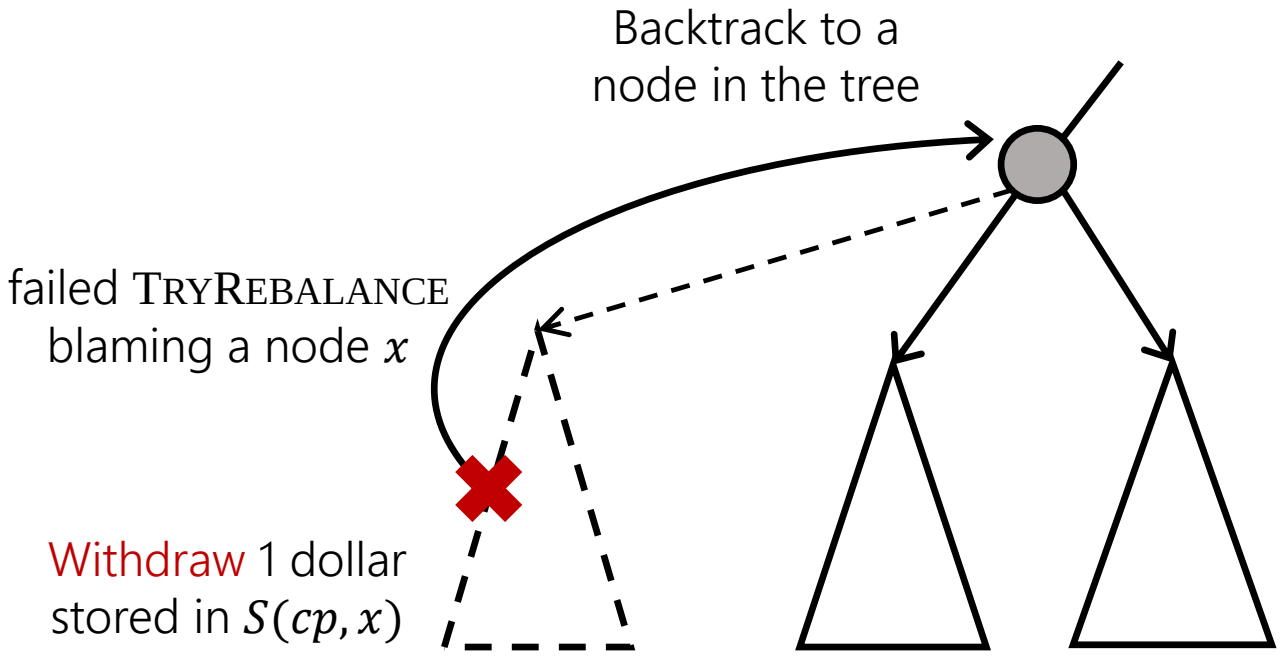
- Each cleanup phase cp owns a number of bank accounts:
  - cp deposits a total of $O(h(cp) + rebal(viol(cp)) \cdot \dot{c}(cp))$ dollars into its own accounts and accounts owned by concurrent operations.
  - Every failed cleanup attempt of cp withdraws 1 dollar from one of its own bank accounts.
  - The bank accounts have non-negative balance.

Paying for Failed TryRebalance(V) Blaming Node x



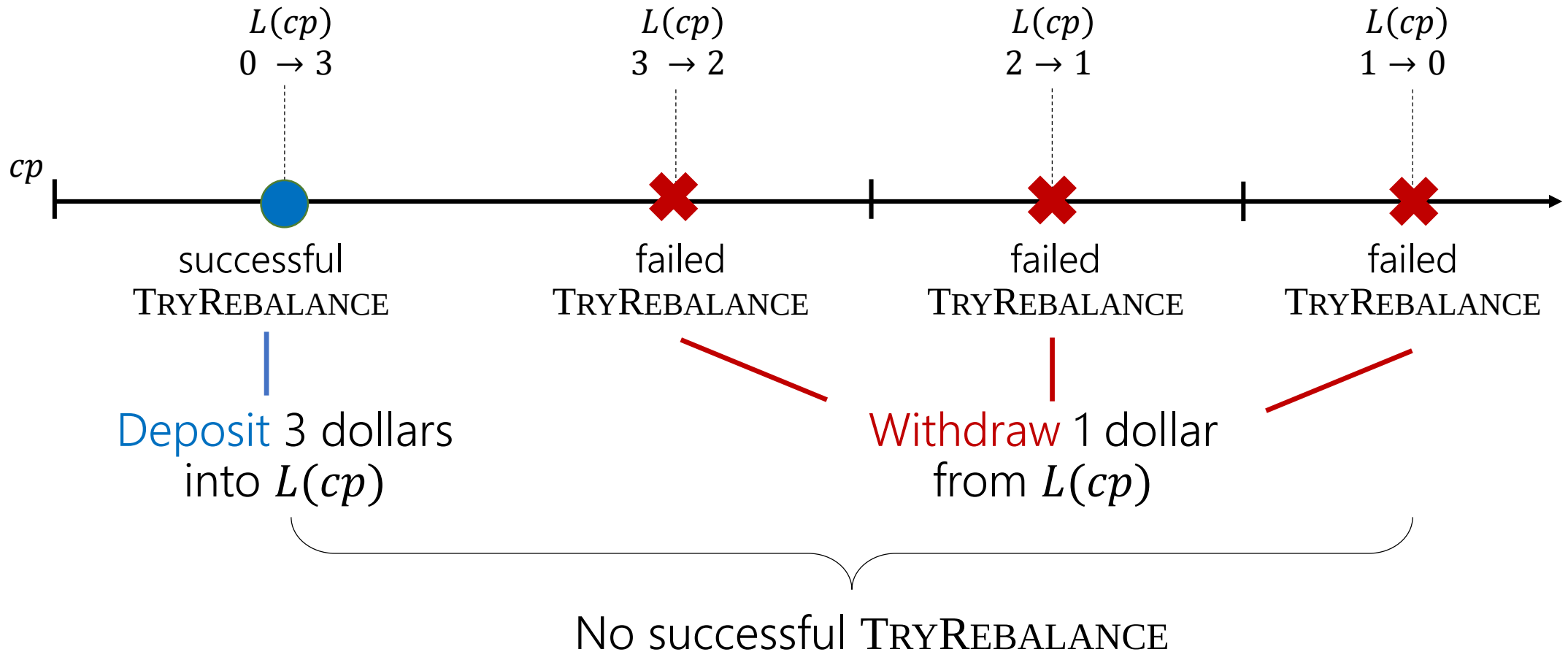
Paying for an Attempt When x is No Longer in the Tree

- When x is removed from the tree by a successful TryRebalance, deposit 1 dollar into the $S(cp, x)$ accounts of each concurrent cleanup phase cp

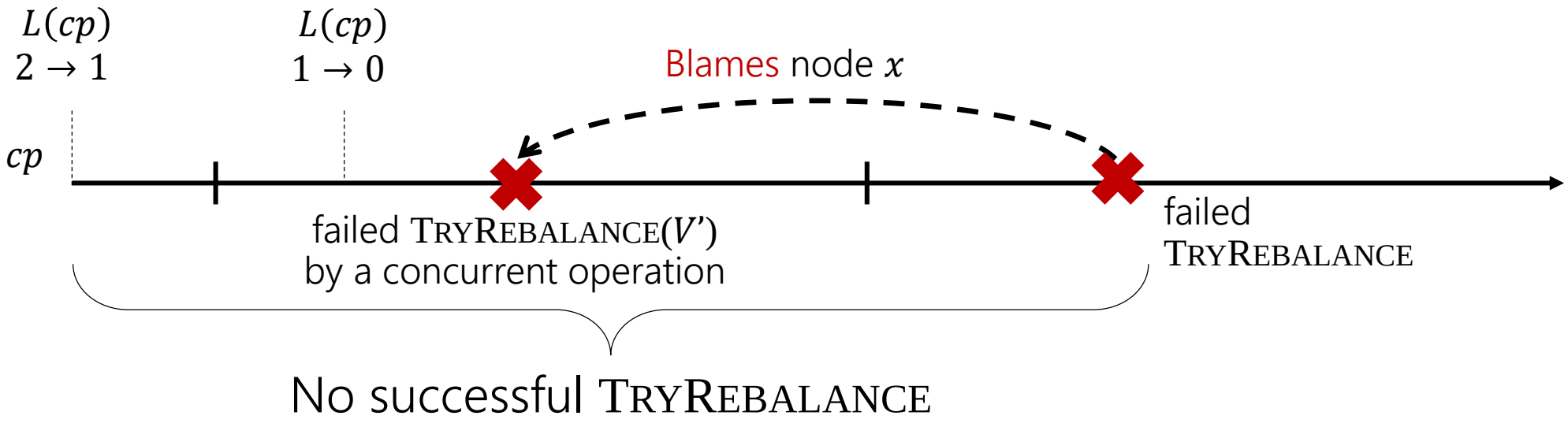


Paying for Failed TryRebalance using $L(cp)$

- Every rebalancing transformation adds 3 dollars to $L(cp)$ for every concurrent cleanup phase cp

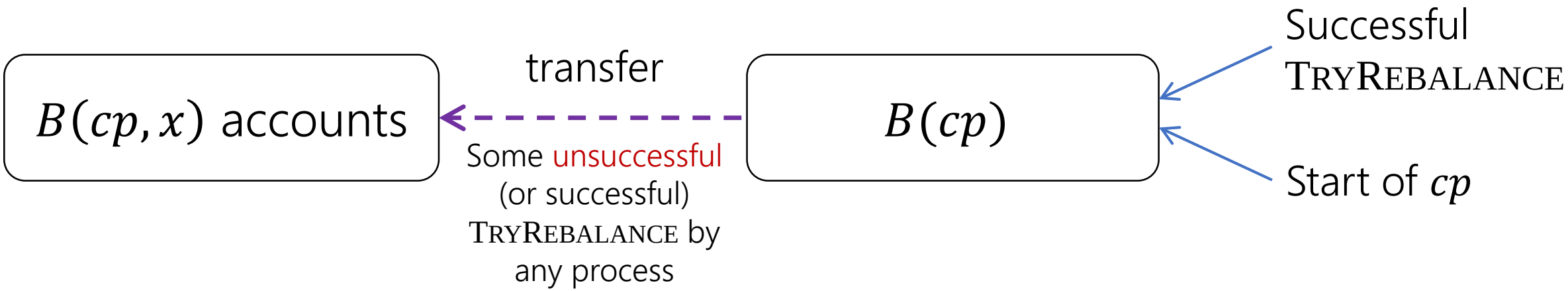


Paying for Failed TryRebalance using $B(cp, x)$



- The failed TryRebalance blames a node x, which was recently modified by a different process performing a failed TryRebalance(V')
- In this case, TryRebalance(V') transfers 1 dollar from $B(cp)$ to $B(cp, x)$.
- The TryRebalance blaming x withdraws this dollar from $B(cp, x)$.

Transfers from the $B(cp)$ Account



Must show:

- enough dollars were transferred from $B(cp)$ and
- not too many instances of TryRebalance withdraw from $B(cp, x)$

so that $B(cp, x)$ remains non-negative.

Conclusion

Theorem: The amortized number of steps made by an operation op in an execution α is $O(\dot{c}(\alpha) + \log n(op))$.

Open Problems:

- Prove an implementation of a chromatic tree has amortized step complexity $O\big(\dot{c}(op) + \log n(op)\big)$, or
- Give a lower bound of $\Omega\big(\dot{c}(\alpha) + \log n(op)\big)$

Future Work:

- Show how the amortized analysis can be extended to other balanced binary search trees