Asynchronous rebalancing of a replicated tree

Marek Zawirski	**INRIA & UPMC, France**
Marc Shapiro	INRIA & LIP6, France
Nuno Preguiça	UNL, Portugal

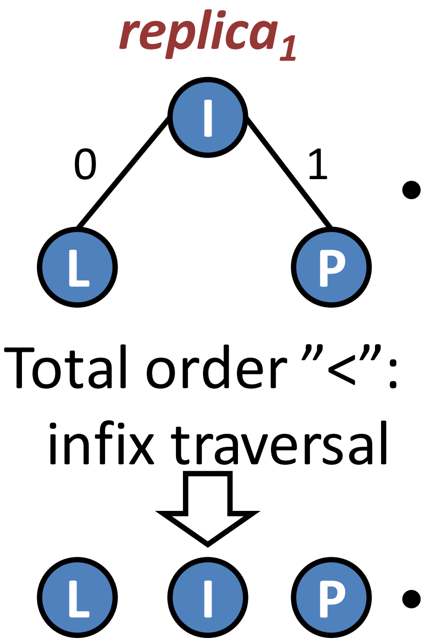
marek.zawirski@lip6.fr

ANR ConcoRDanT & STREAMS, Paris, June 2011

Summary

- Overview of Treedoc:
  - Abstractly, always-responsive replicated sequence
  - Built as a replicated ordering tree
- Problem faced:
  - Tree rebalanced on some replicas,
    while concurrently updated on others
- Approach:
  - *Catch-up* protocol to integrate rebalance on all replicas
- Novel *catch-up* algorithm using *symbolic positions*

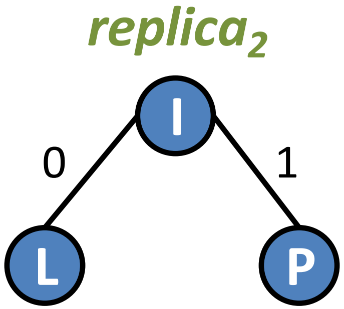
Treedoc – a replicated sequence

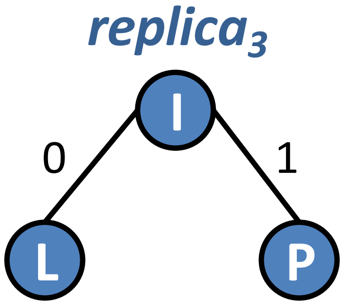


Total order ”<”:
infix traversal

- **Replicated representation**:
  - Grow-only binary tree
  - Stable, unique position ids

    P = *1*
- **Sequence of atoms**:
  - Ops: *read*, *addAt*, *removeAt*

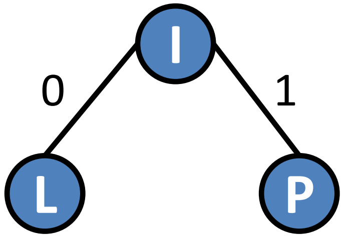




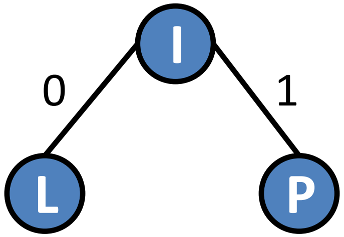
[Shapiro, Preguiça et. al, 2007, 2009]

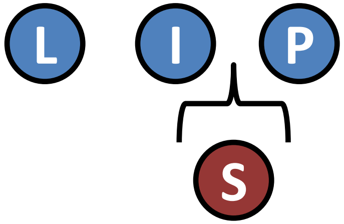
Treedoc – a replicated sequence

replica$_1$



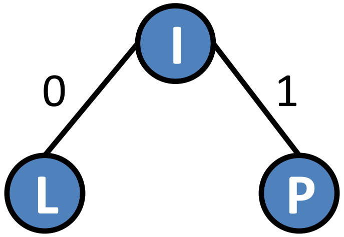
replica$_2$





addAt(S)   I < S < P

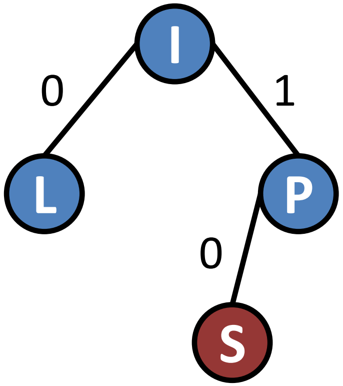
replica$_3$



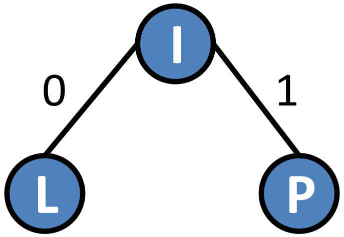
[Shapiro, Preguiça et. al, 2007, 2009]

Treedoc – a replicated sequence

replica$_1$

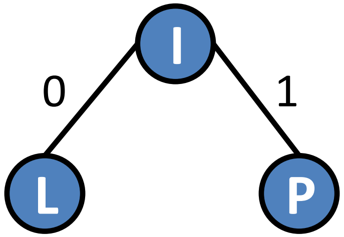


replica$_2$



L I S P

replica$_3$

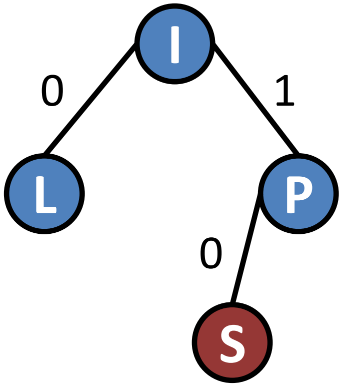


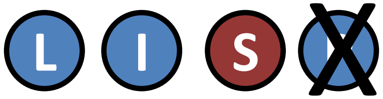
addAt(*10*, S)   I < S < P   (S = *10*)

[Shapiro, Preguiça et. al, 2007, 2009]

Treedoc – a replicated sequence

replica$_1$

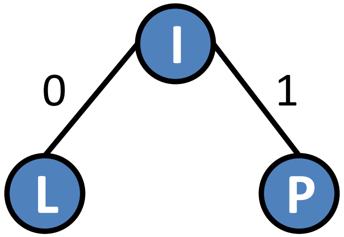




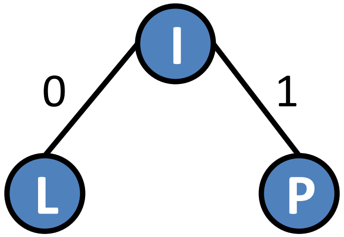
addAt(10, S*)*

removeAt(1, P*)*

replica$_2$



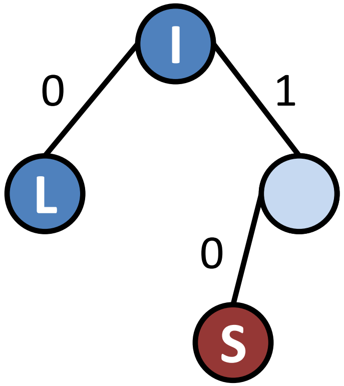
replica$_3$



[Shapiro, Preguiça et. al, 2007, 2009]

Treedoc – a replicated sequence

replica$_1$

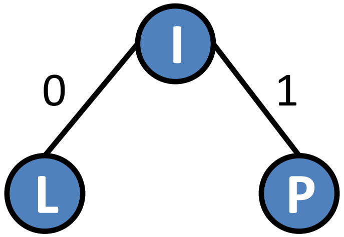


L I S

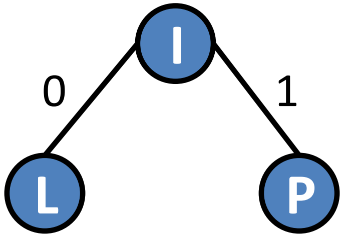
addAt(*10*, S)

removeAt(*1*, P)

replica$_2$



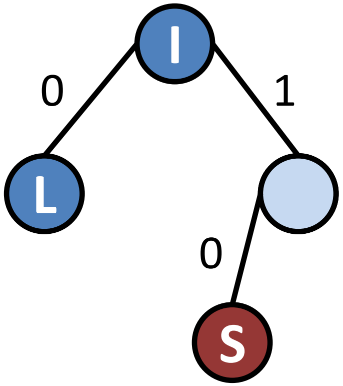
replica$_3$



[Shapiro, Preguiça et. al, 2007, 2009]

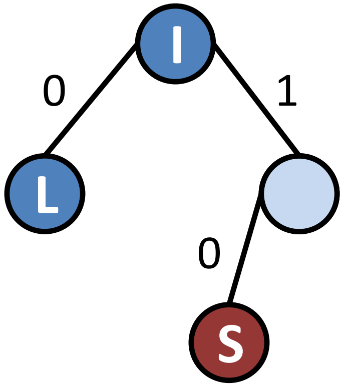
Treedoc – a replicated sequence

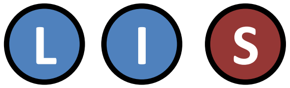
***replica*$_1$**



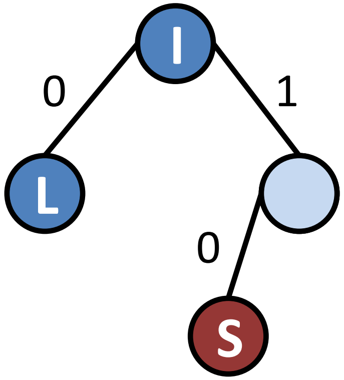
- **Operation-based replication:**
  - Immediate local execution
  - Propagate (cbcast) & replay

***replica*$_2$**





***replica*$_3$**

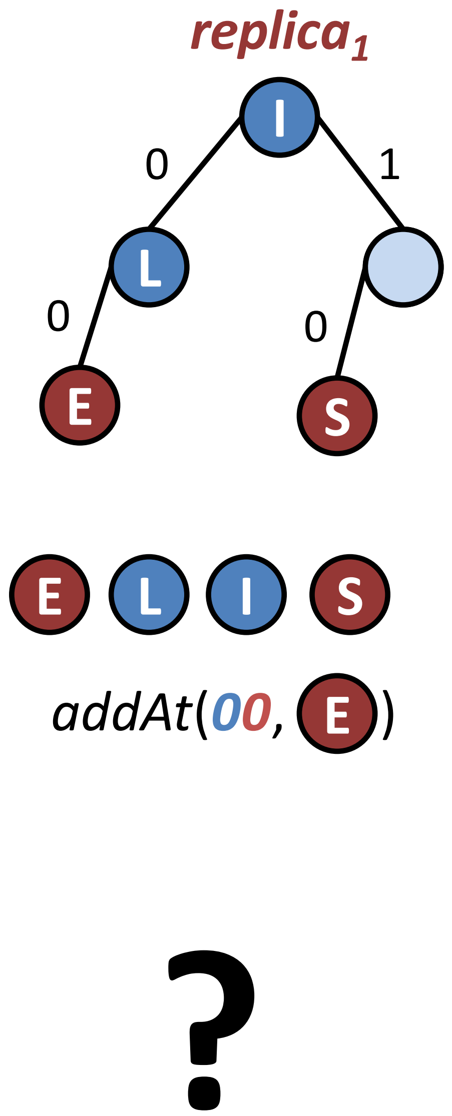


addAt(10, S)

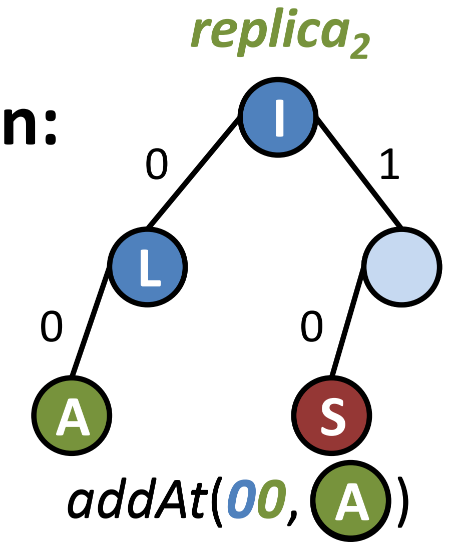
removeAt(1, P)

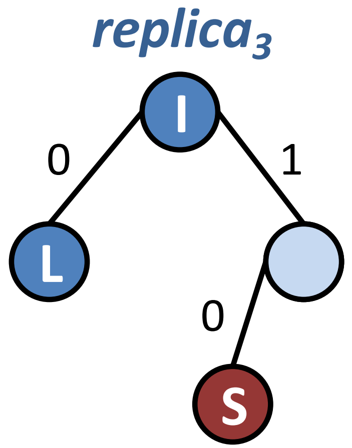
[Shapiro, Preguiça et. al, 2007, 2009]

Treedoc – a replicated sequence



- **Operation-based replication:**
  - Immediate local execution
  - Propagate (cbcast) & replay



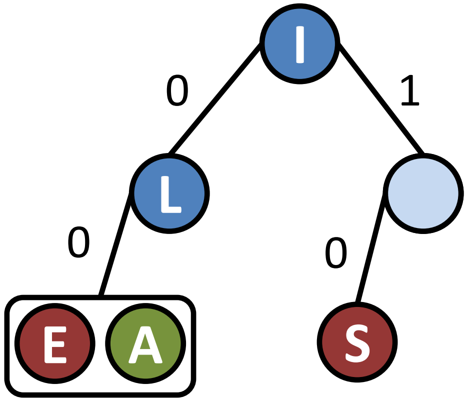


[Shapiro, Preguiça et. al, 2007, 2009]

Treedoc – a replicated sequence

- **Operation-based replication:**
  - Immediate local execution
  - Propagate (cbcast) & replay

replica$_1$



E A L I S

addAt(*00*, E)

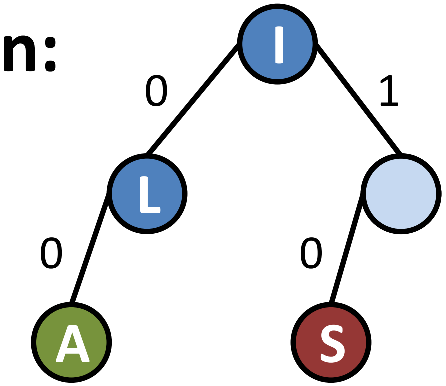
addAt(*00*, A)

Predefined order:

red $<_c$ **green** $<_c$ **blue**…

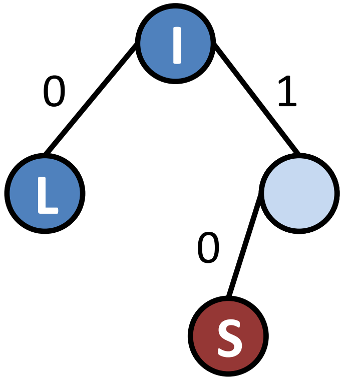
[Shapiro, Preguiça et. al, 2007, 2009]

replica$_2$

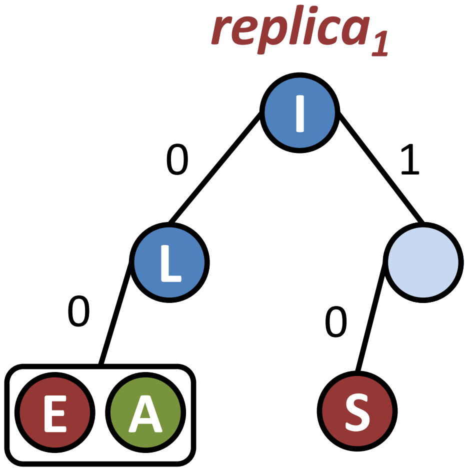


addAt(*00*, A)

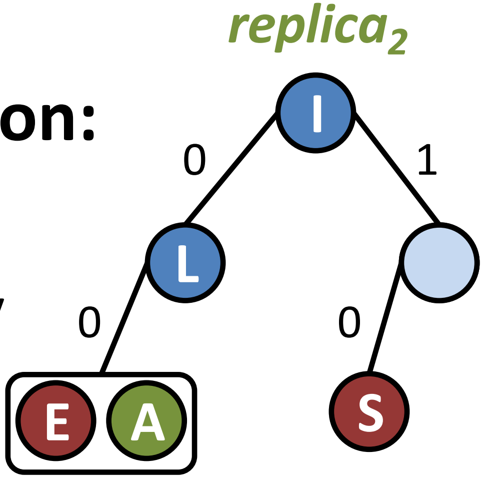
replica$_3$

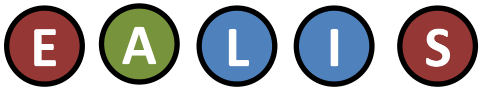


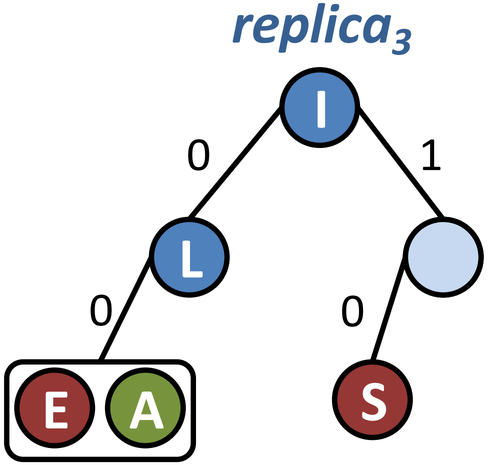
Treedoc – a replicated sequence



- **Operation-based replication:**
  - Immediate local execution
  - Propagate (cbcast) & replay
  - Concurrent commute
  - Eventually consistent



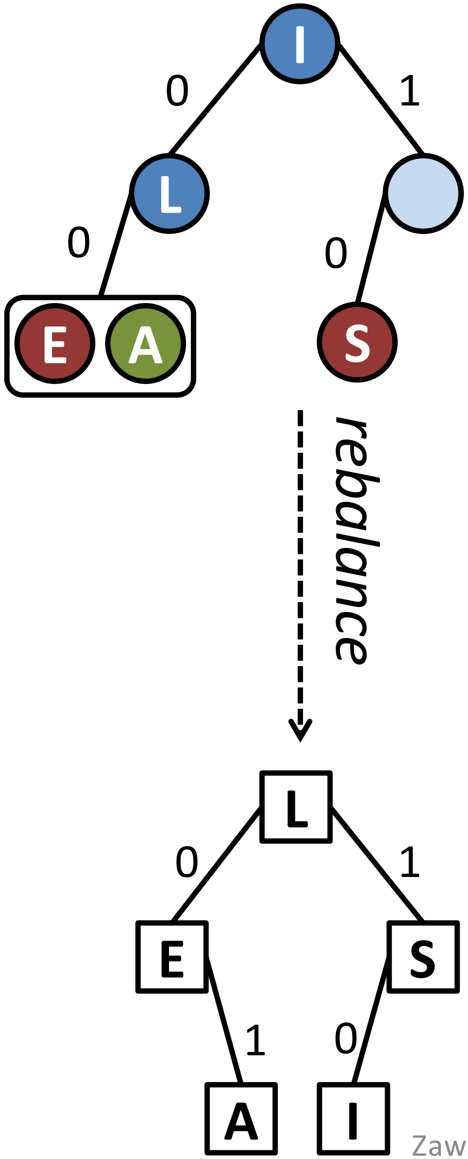




Predefined order:
red $<_c$ **green** $<_c$ **blue**…

[Shapiro, Preguiça et. al, 2007, 2009]

The tree rebalance problem



- **With time tree gets worse and worse**
  - Unbalanced, empty nodes, lot of colors…
  - Various negative impacts
- **Tree rebalance:**
  - Create minimal tree from nonempty nodes
  - Keep order "<"
  - Use single color (white)
  - **New ids epoch (rectangles), incompatible**
- **Challenge:**
  - **Ensuring identical rebalance across replicas without costly consensus or lost updates**

The core-nebula architecture

Idea: limit consensus to a smaller number of replicas

[Leția et. al, 2009]

Divide replicas into two disjoint sets:

CORE

- a stable group
- execute tree operations & agree on *rebalance*

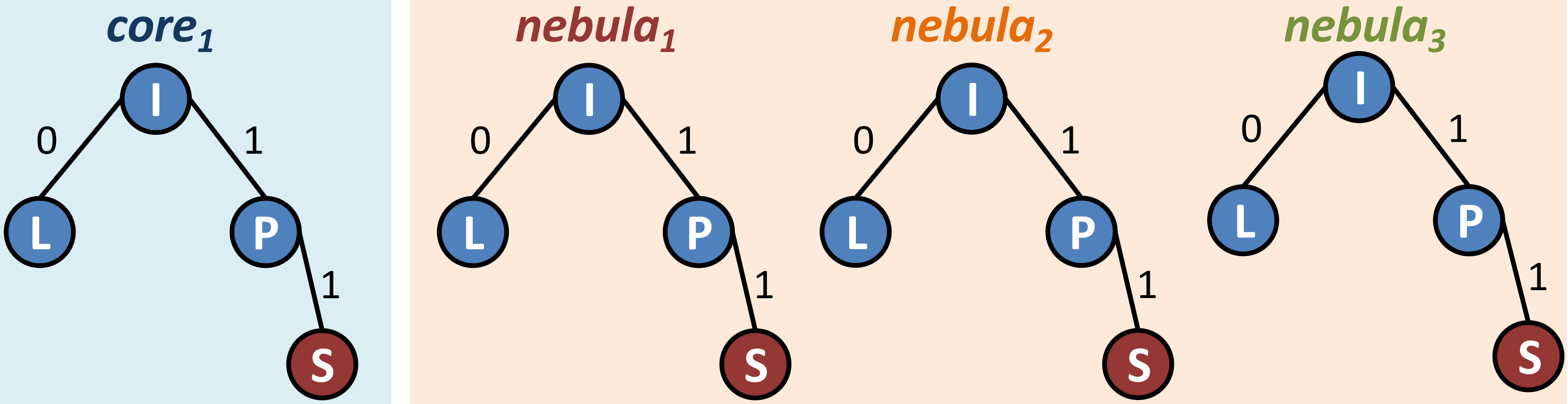
✓ easier agreement

NEBULA

- sites join & leave, dynamic
- generate tree operations
- learns about *rebalance*
- perform ***catch-up*** protocol to integrate conc. changes

✓ never blocked

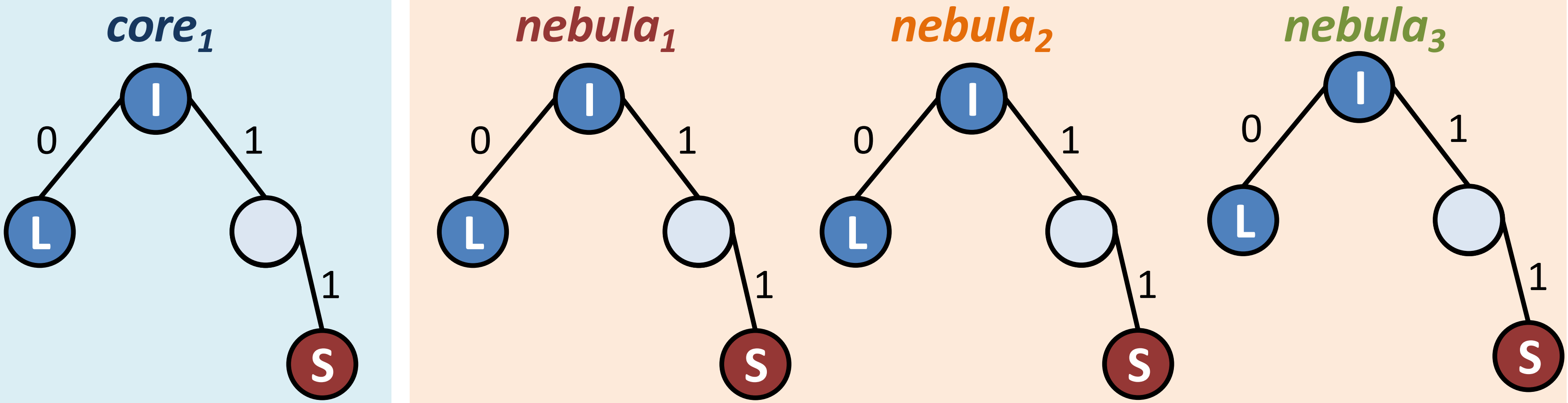
Rebalance in core, catch-up from nebula



addAt(11, S)

- Any pair of replicas can exchange operations in the same epoch

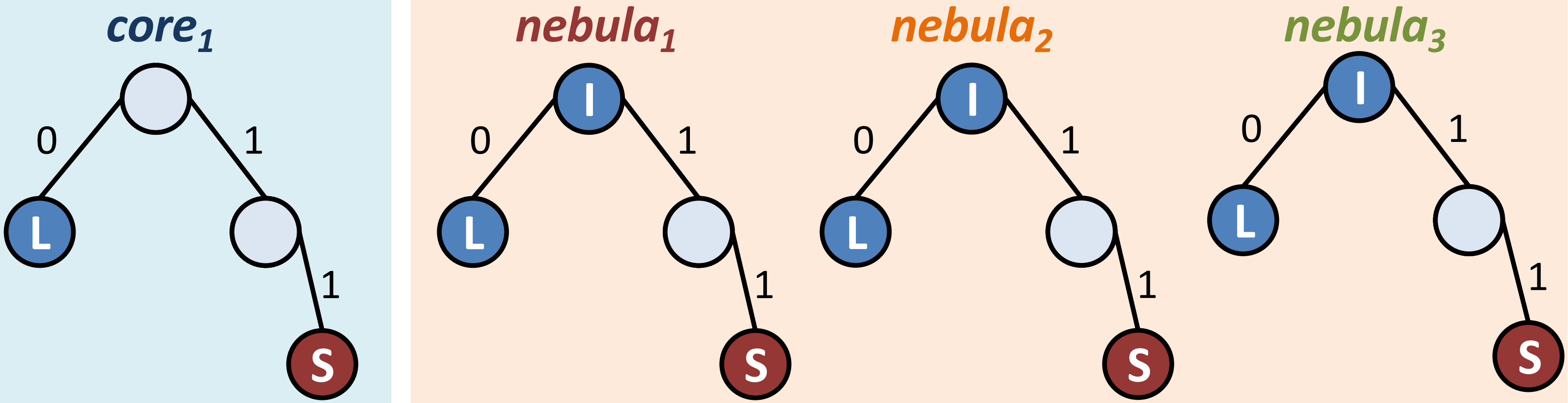
Rebalance in core, catch-up from nebula



removeAt(1, P)

- Any pair of replicas can exchange operations in the same epoch

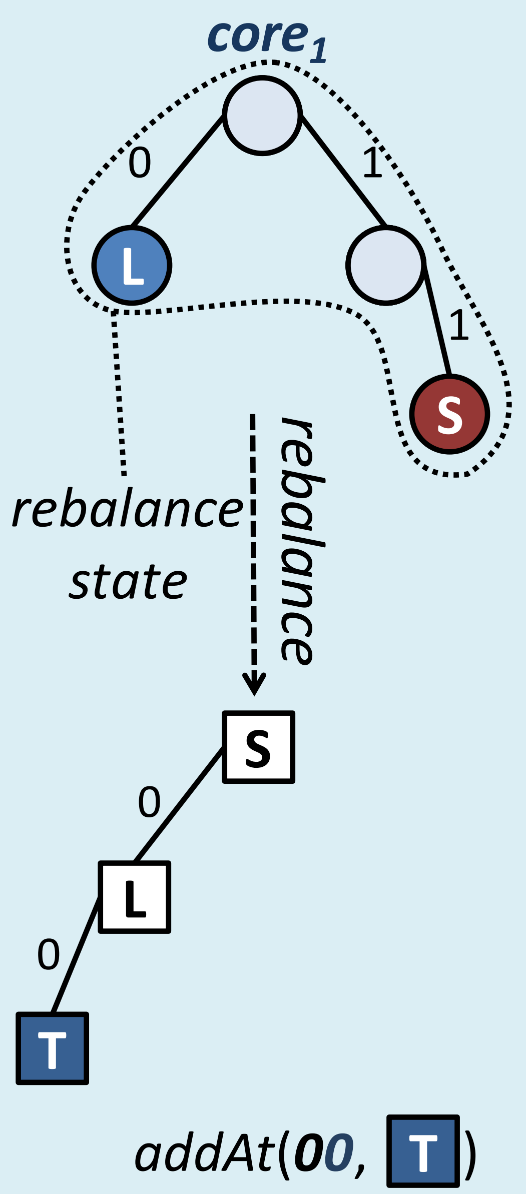
Rebalance in core, catch-up from nebula

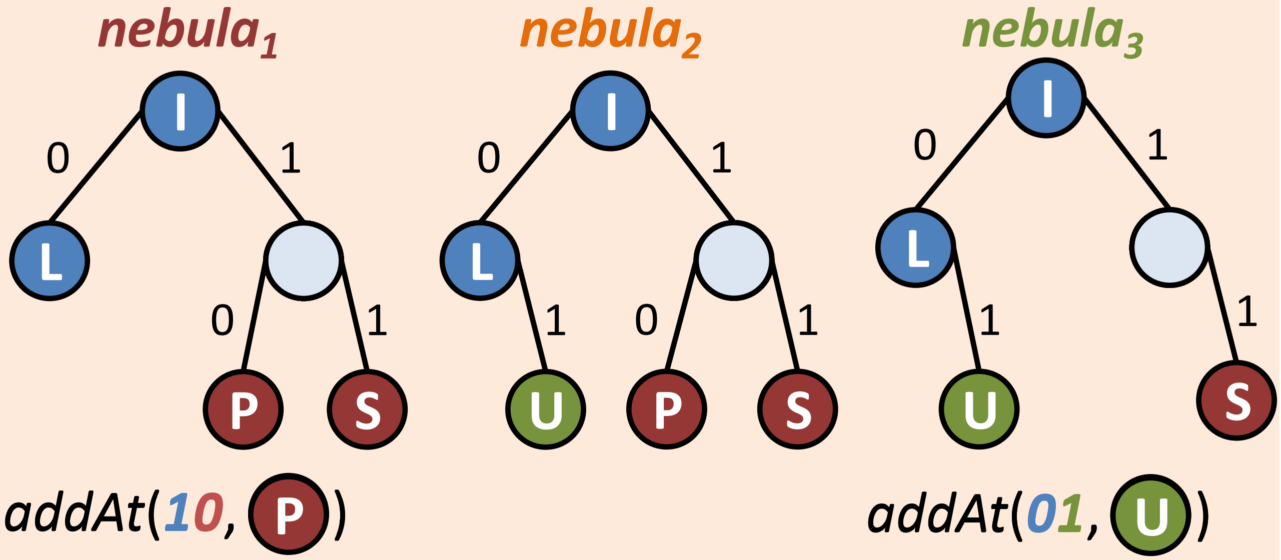


removeAt(⊤, I)

- Any pair of replicas can exchange operations in the same epoch

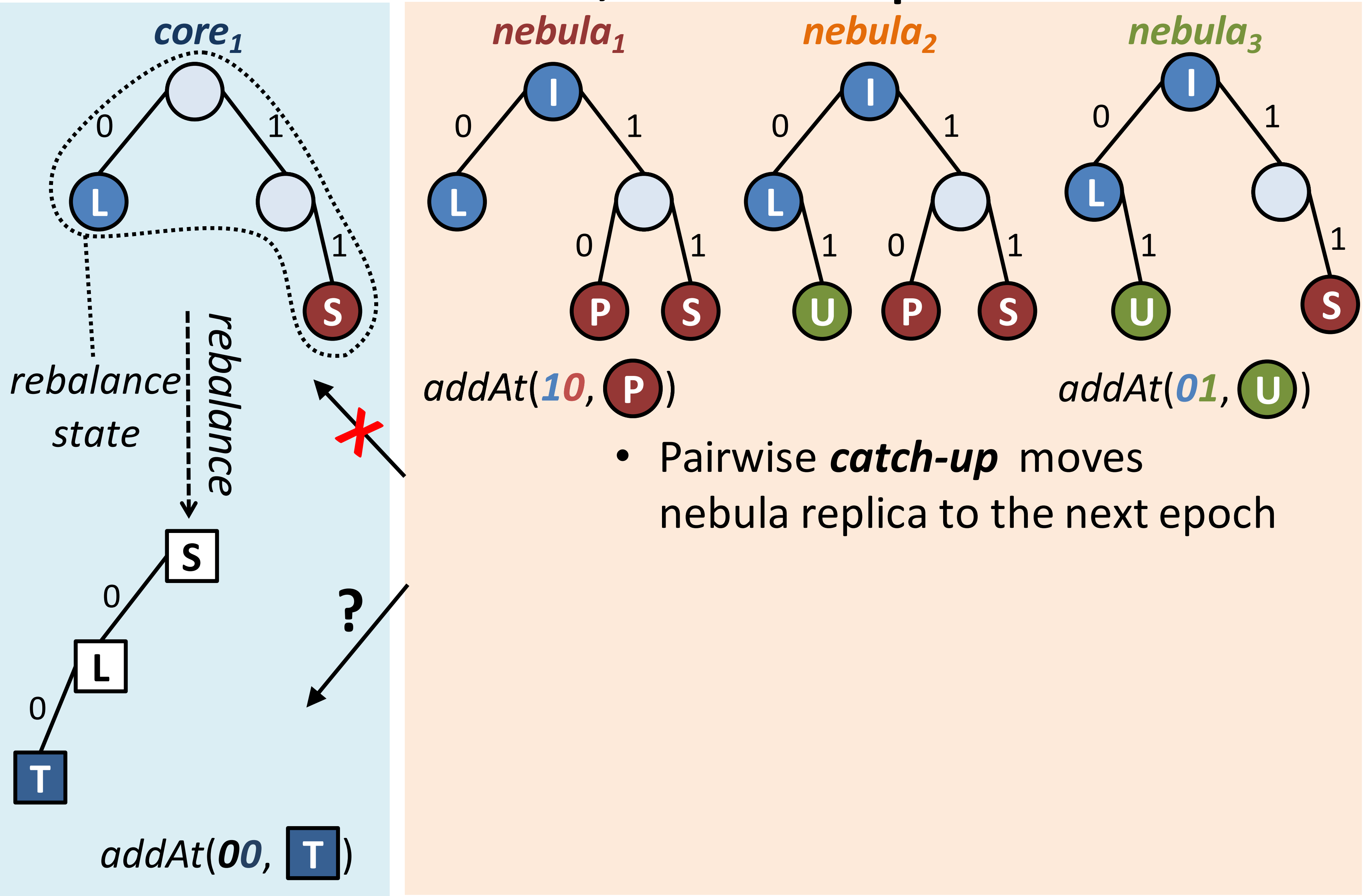
Rebalance in core, catch-up from nebula





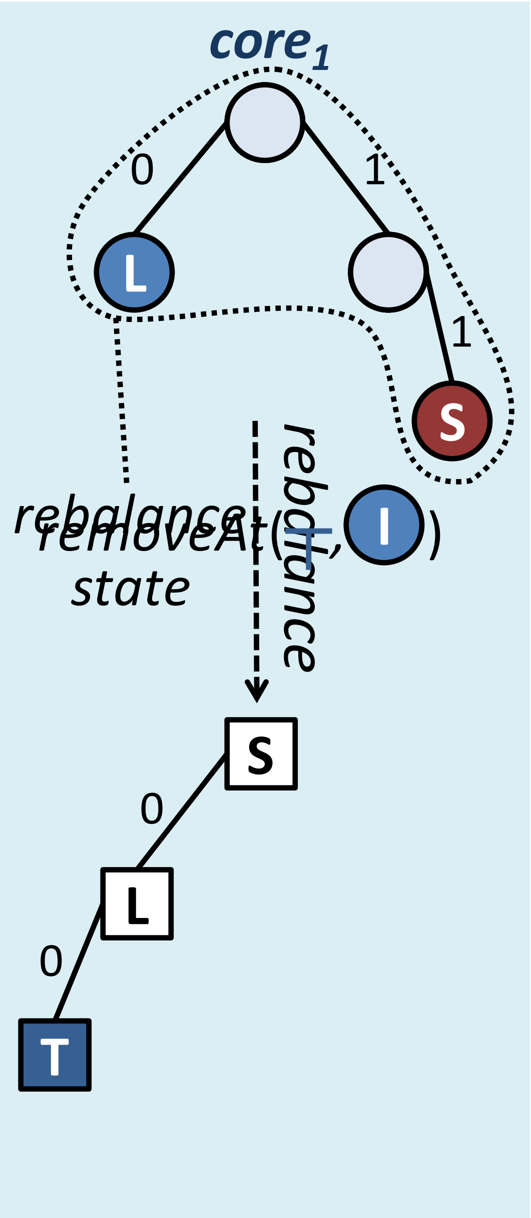
- Any pair of replicas can exchange operations in the same epoch
- *rebalance@core* initiates new *epoch*
- *rebalance@core* and *operations@core* inherently concurrent to *ops@nebula*!

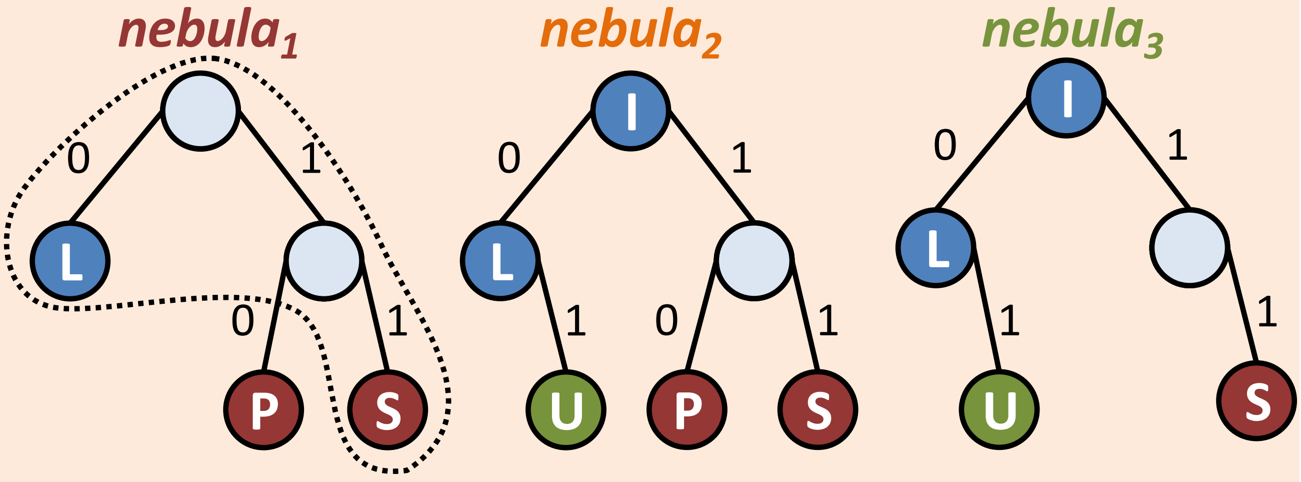
Rebalance in core, catch-up from nebula



- Pairwise ***catch-up*** moves nebula replica to the next epoch

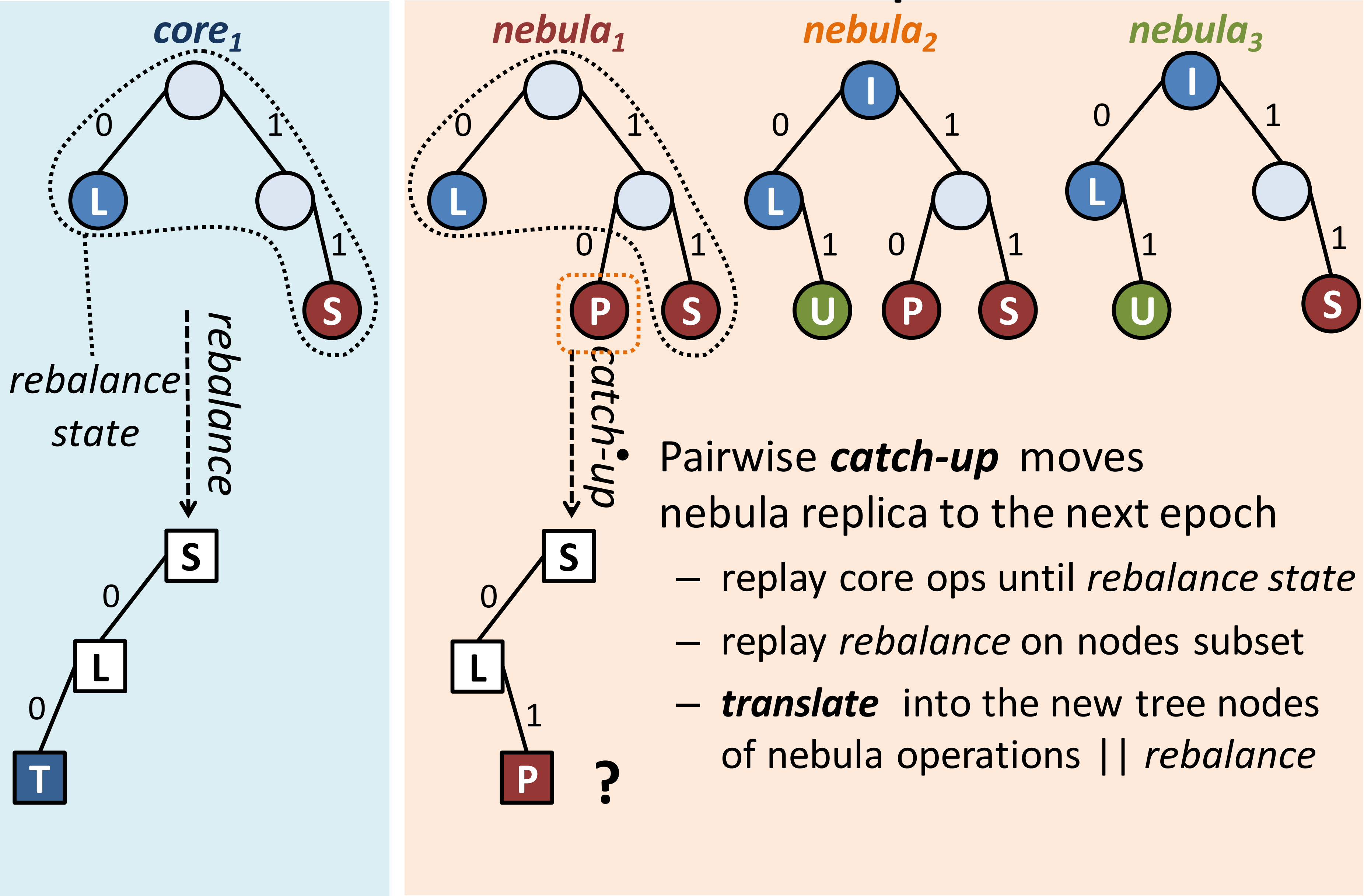
Rebalance in core, catch-up from nebula





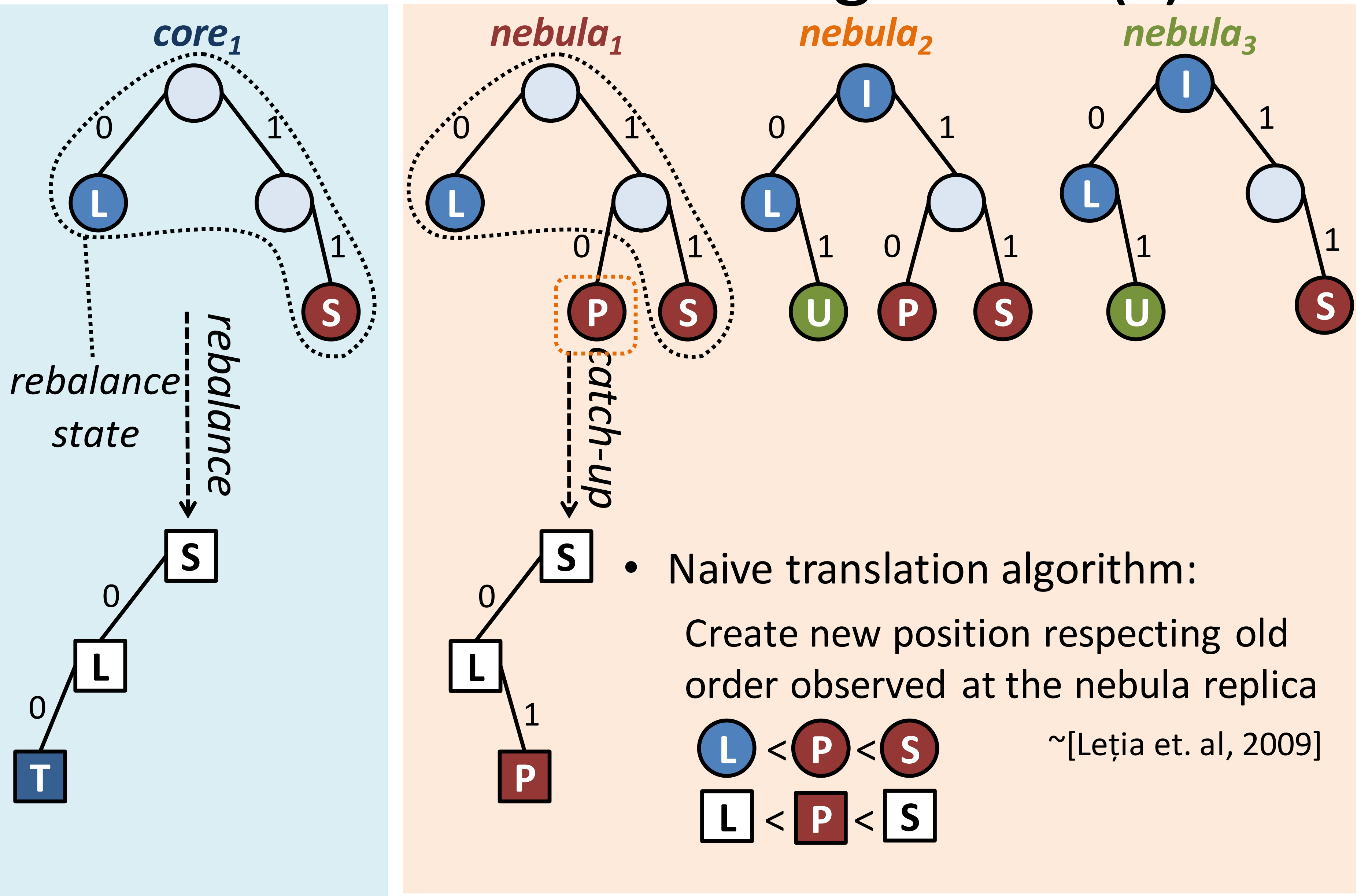
- Pairwise ***catch-up*** moves nebula replica to the next epoch
  - replay ops until *rebalance state*

Rebalance in core, catch-up from nebula



- Pairwise ***catch-up*** moves nebula replica to the next epoch
  - replay core ops until *rebalance state*
  - replay *rebalance* on nodes subset
  - ***translate*** into the new tree nodes of nebula operations || *rebalance*

Naive translation algorithm(s)



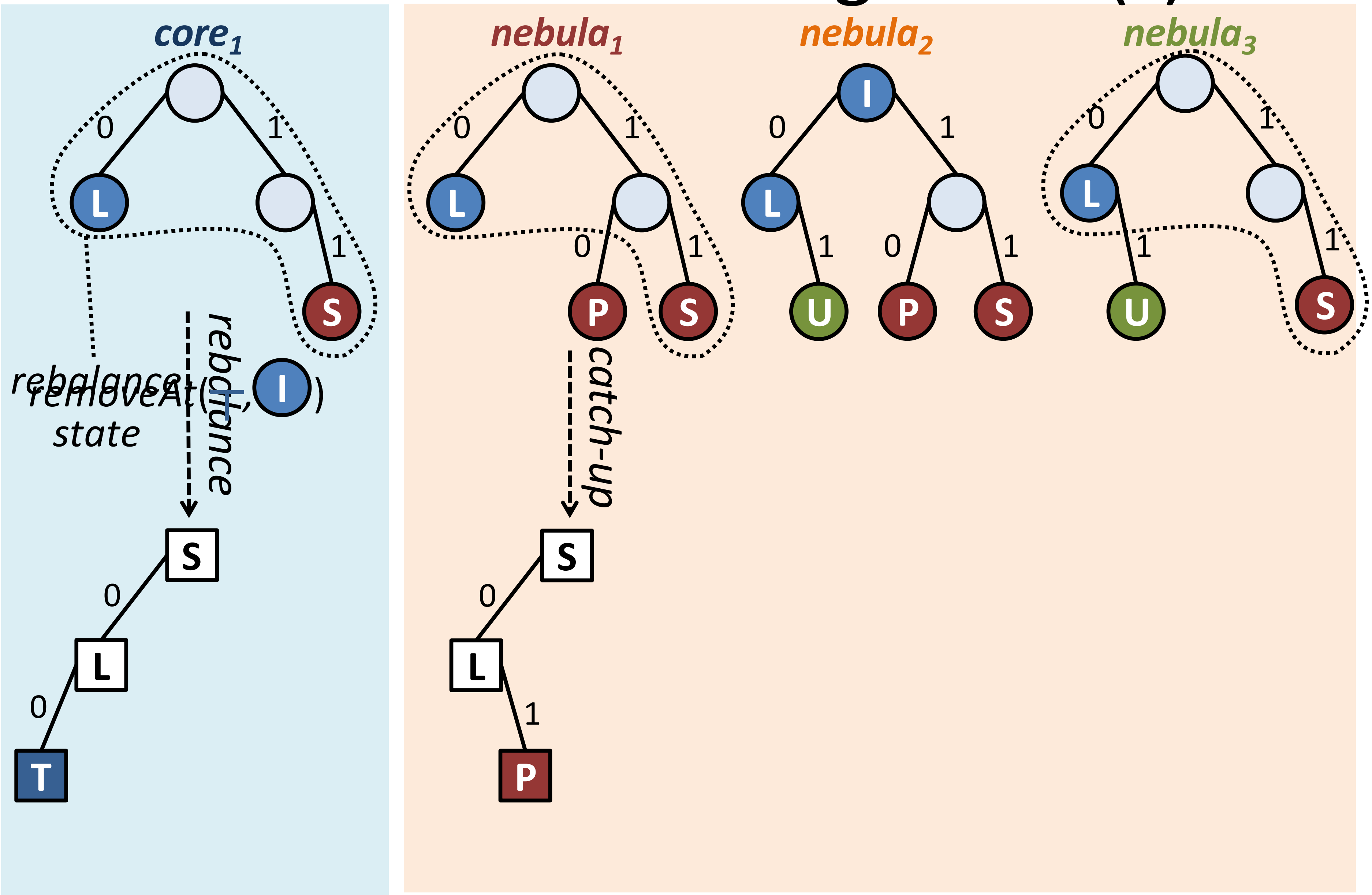
- Naive translation algorithm:

  Create new position respecting old order observed at the nebula replica

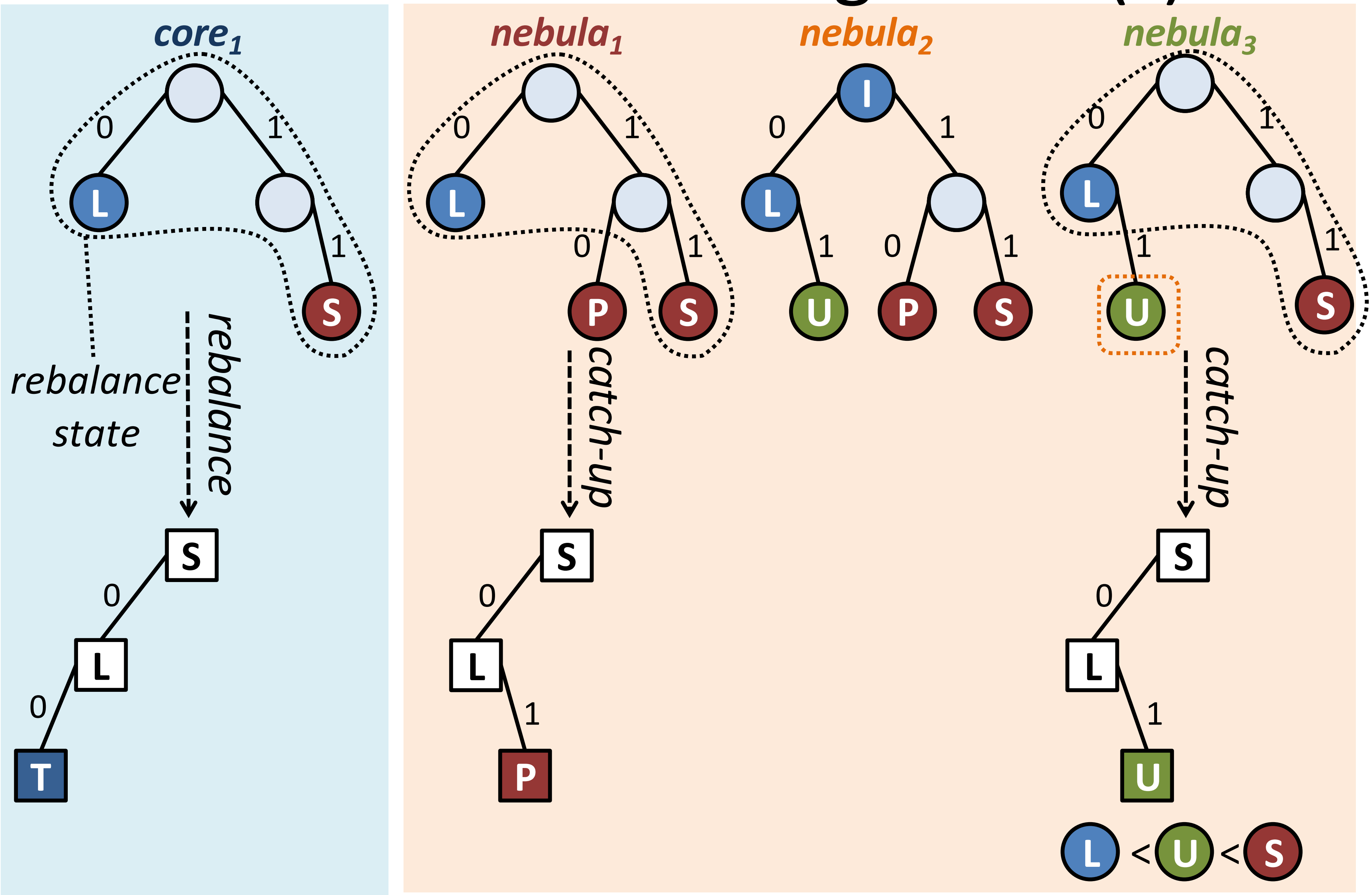
  L < P < S ~[Leția et. al, 2009]

  L < P < S

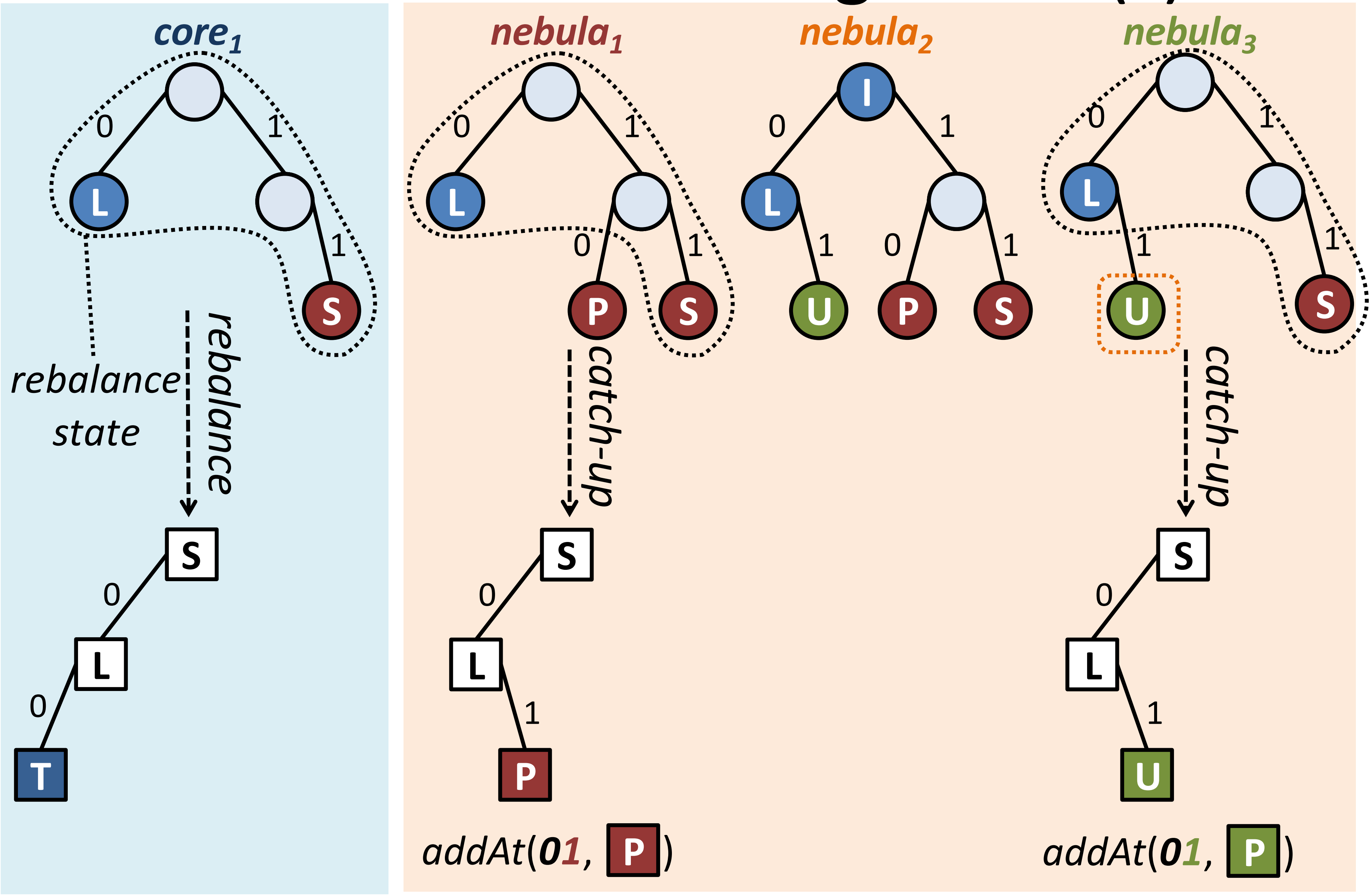
Naive translation algorithm(s)



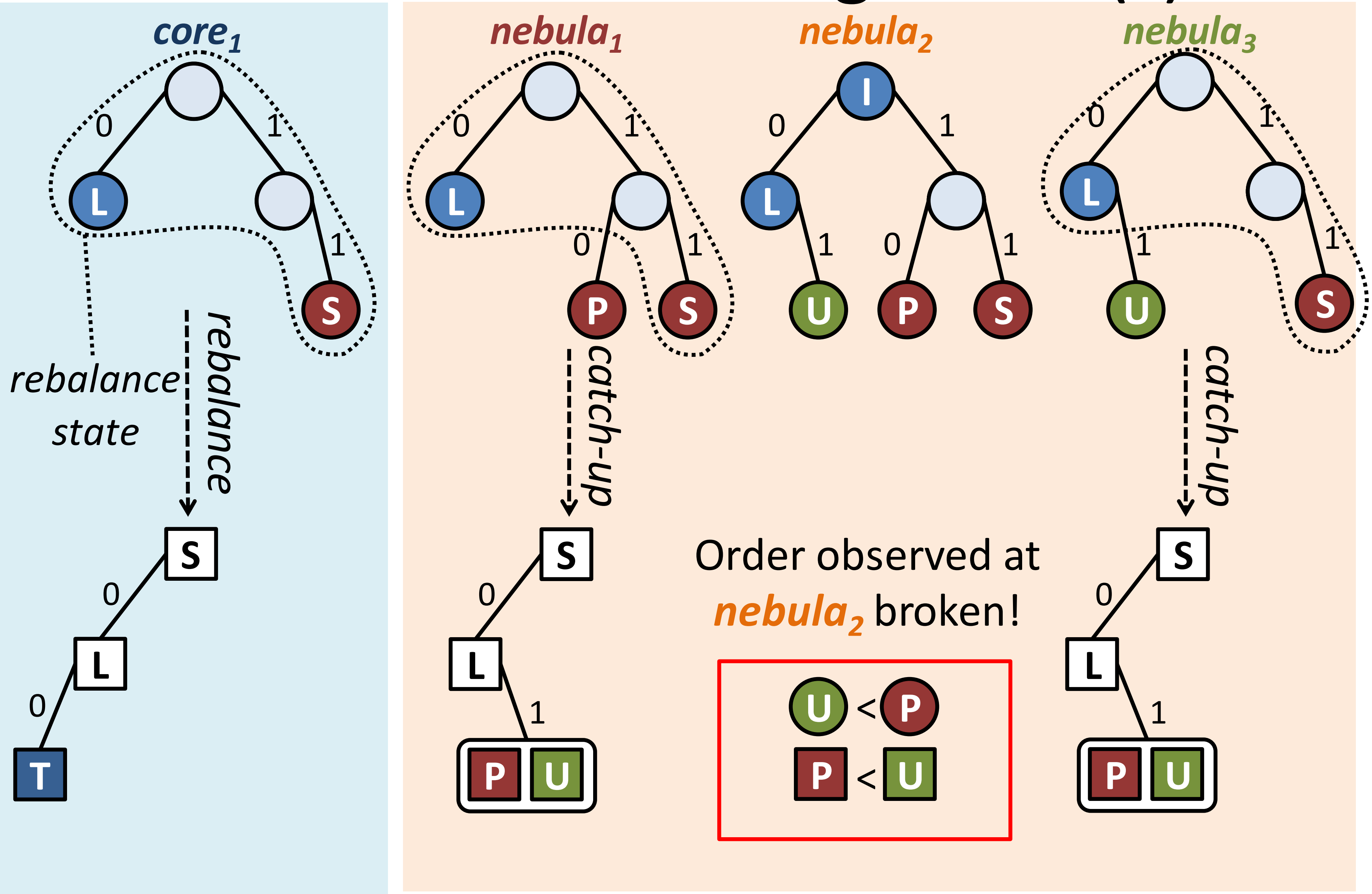
Naive translation algorithm(s)



Naive translation algorithm(s)



Naive translation algorithm(s)



Order observed at *nebula*$_2$ broken!

Towards correct *translate*: requirements

1. **Order-preserving**
   - For every X, Y the order is preserved between epochs:

     X < Y => X < Y

2. **Deterministic**
   - For every X , *nebula*$_i$, *nebula*$_j$, X is translated identically:

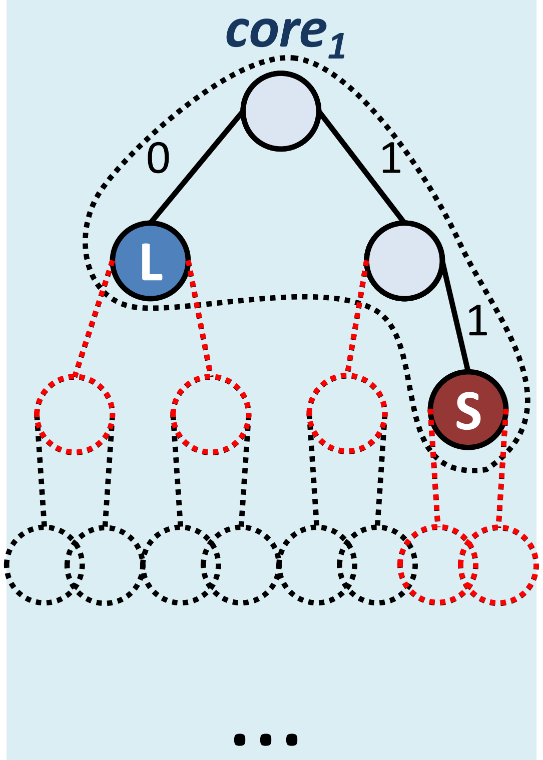
     X @*nebula*$_i$ = X @*nebula*$_j$

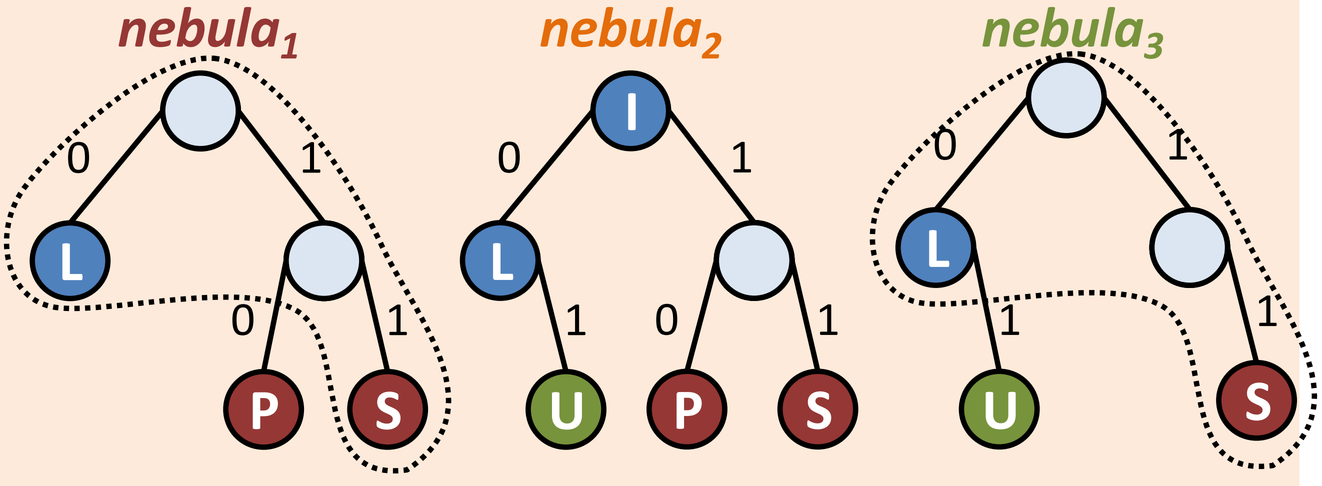
3. **Non-disruptive**
   - For every X created by *addAt* and Y created by *translate*:

     X != Y

Solution: designate all cases in advance using *rebalance state*!

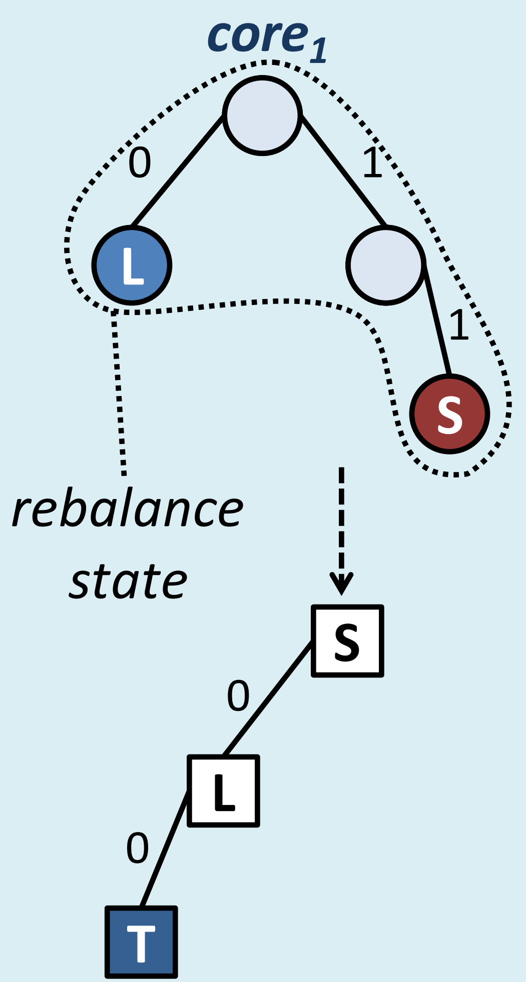
R-Translate: abstract view (simplified)

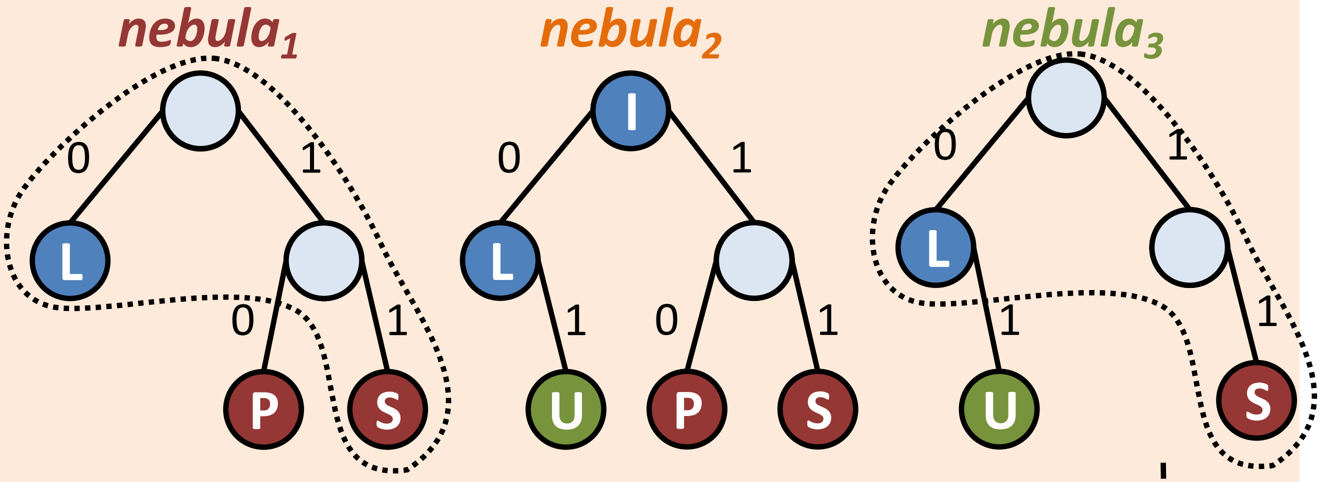




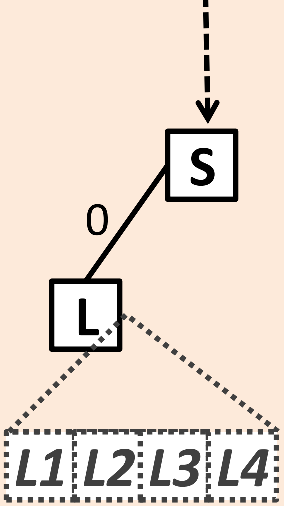
- Set of potential input to translate:
  - Ordered by “<“ relation
  - Infinite => hard to designate cases
- Set of roots of potential input:
  - Ordered and finite!
  - Enough to consider only roots
  - Ignore colors (equivalence classes)

R-Translate & symbolic positions

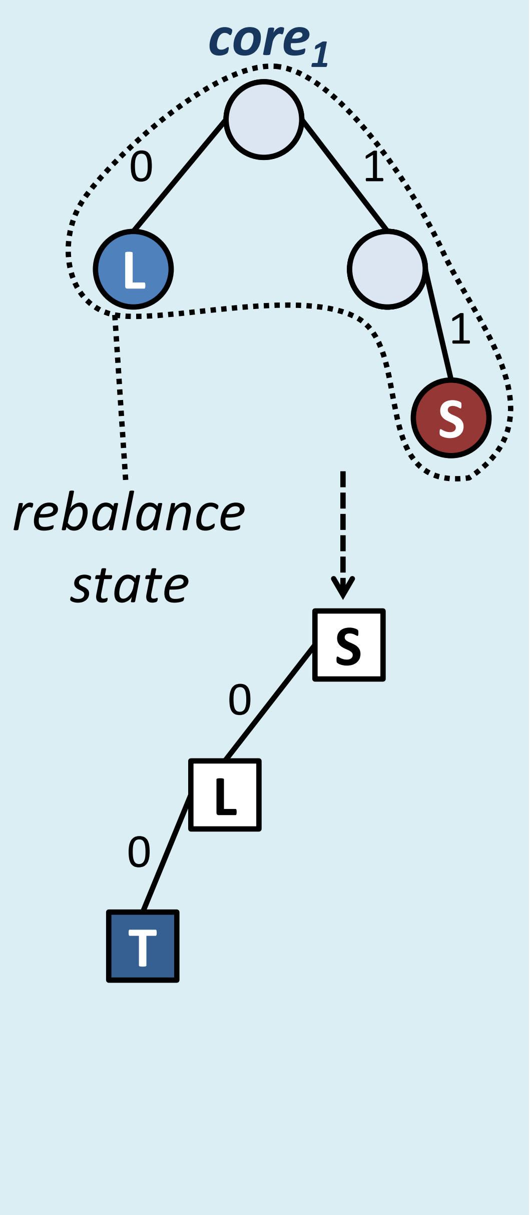


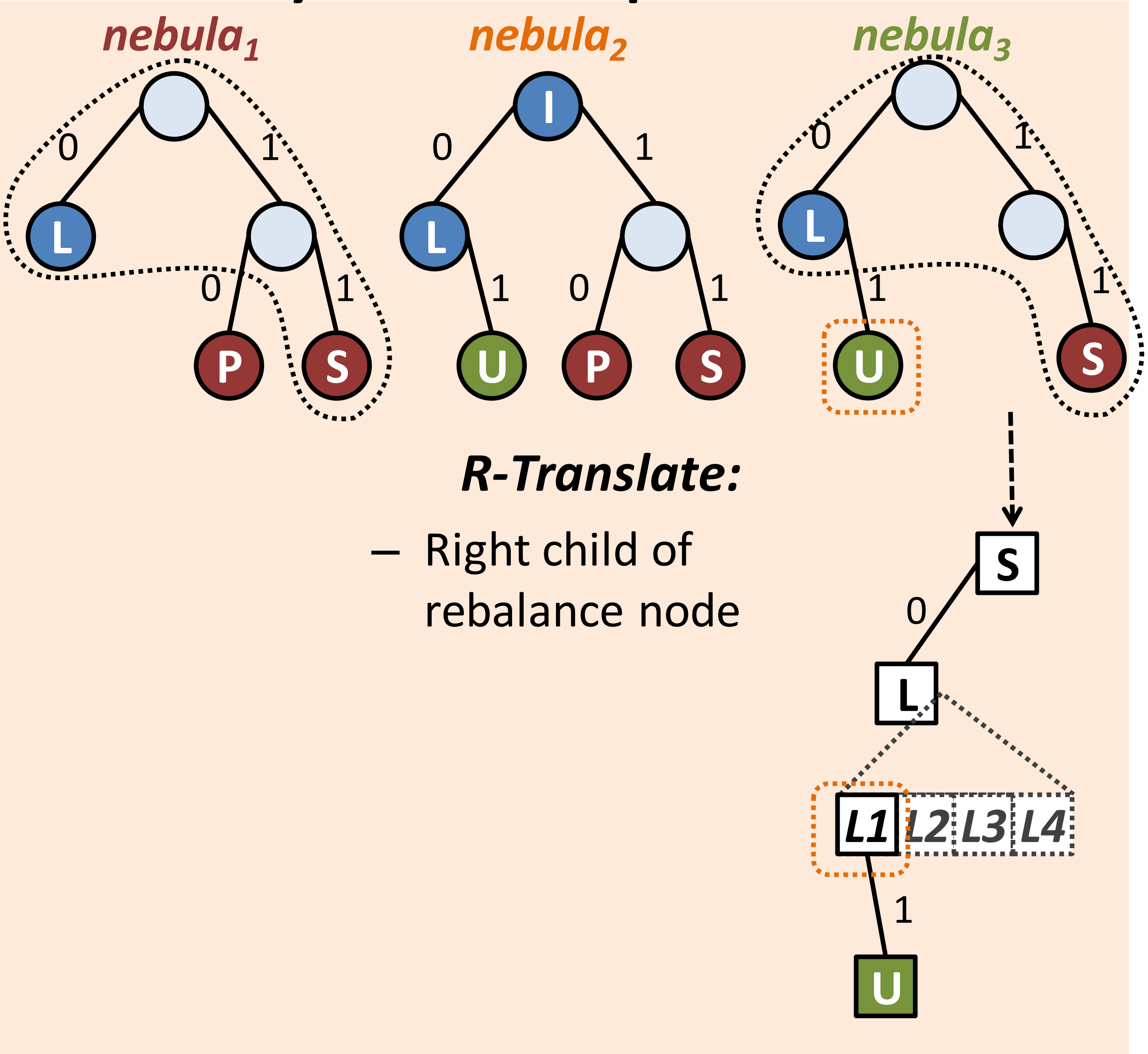


- *Symbolic position* *Xi* :
  - Designated position for every potential *translate*
  - X < *X1* < *X2* … < Xi_m < Y
  - Allocated for *rebalance nodes* in proper number
  - Materialized on *translate*



R-Translate & symbolic positions

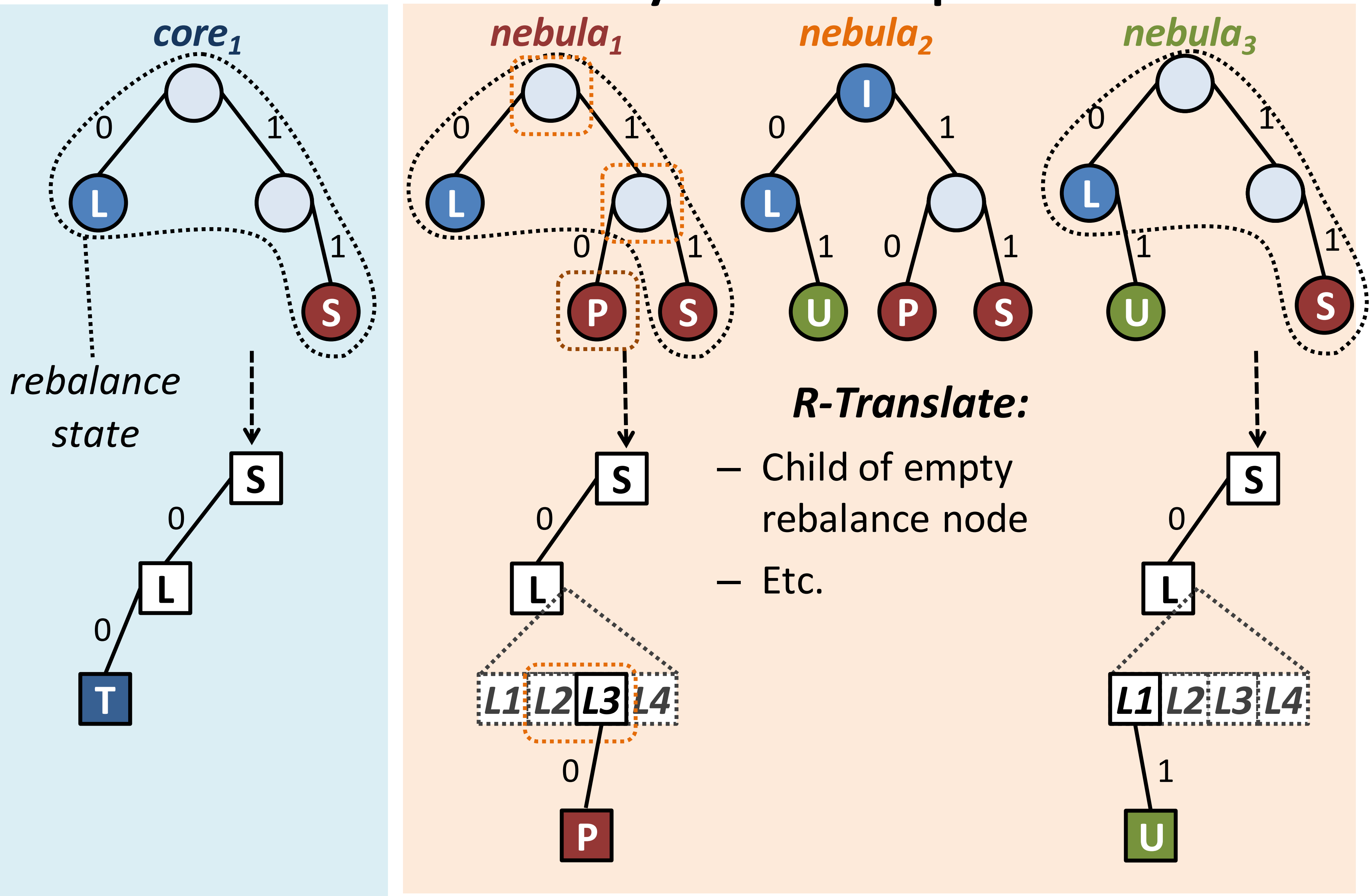




R-Translate:

– Right child of rebalance node

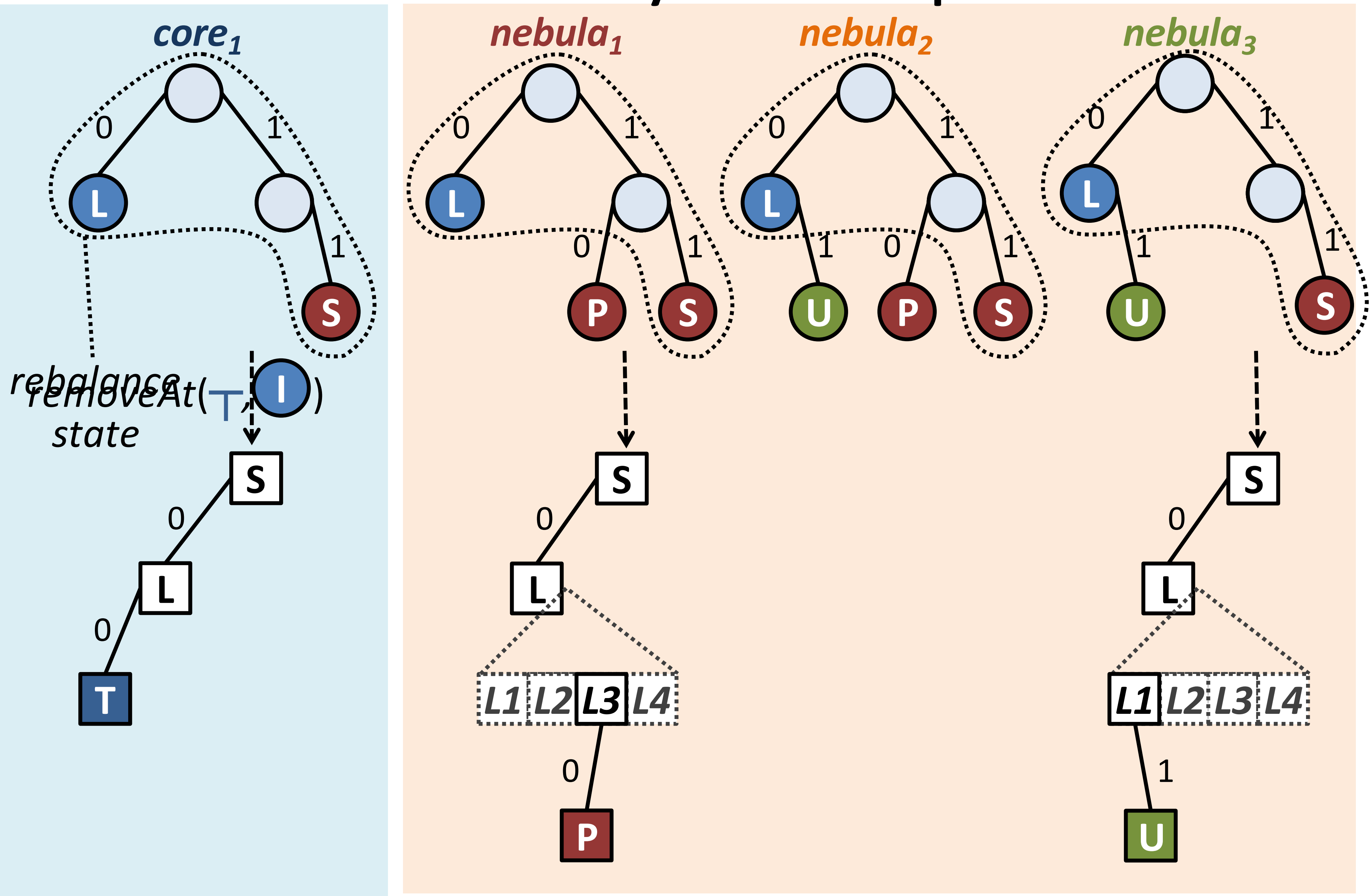
R-Translate & symbolic positions



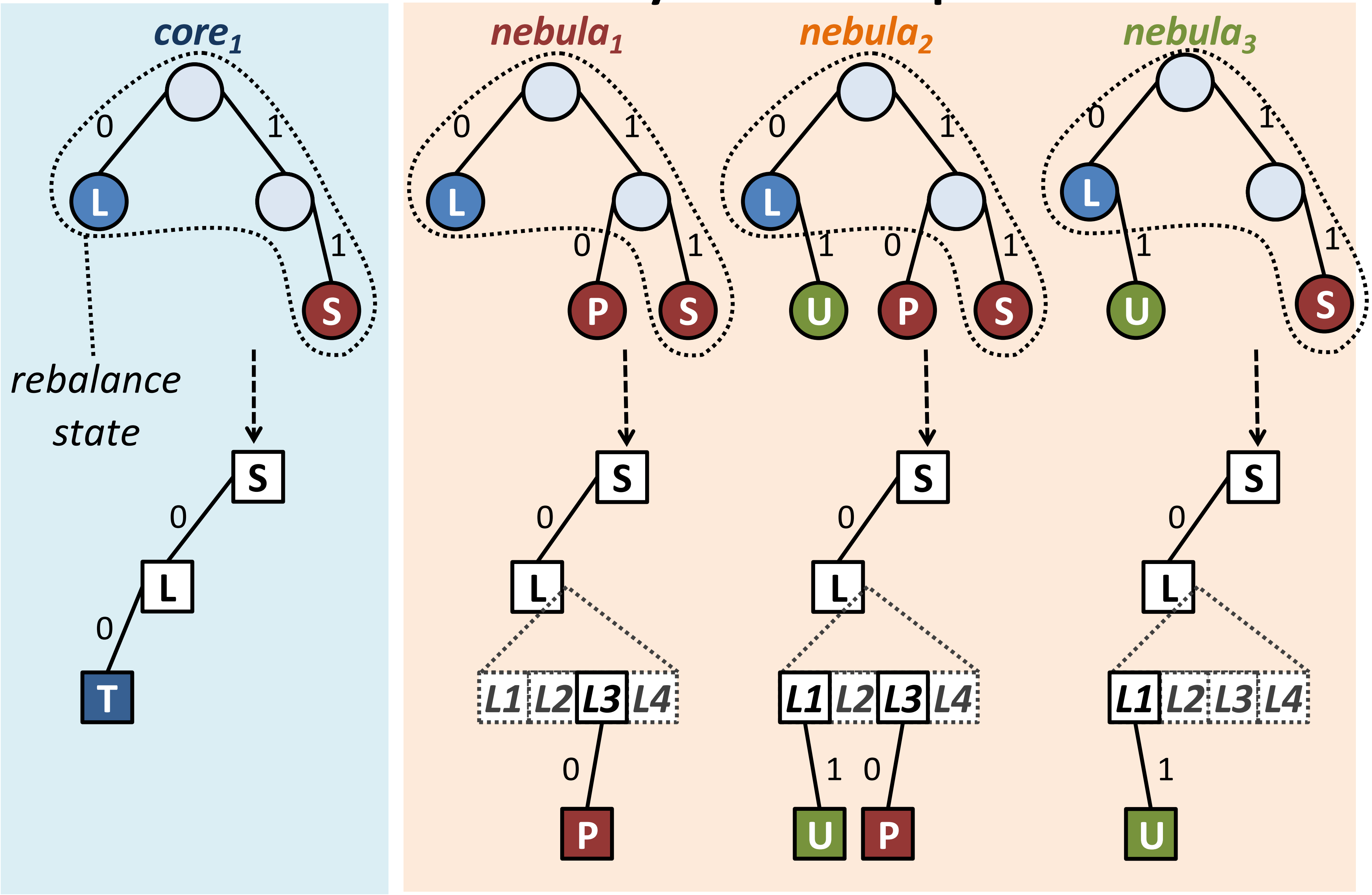
R-Translate:

- Child of empty rebalance node
- Etc.

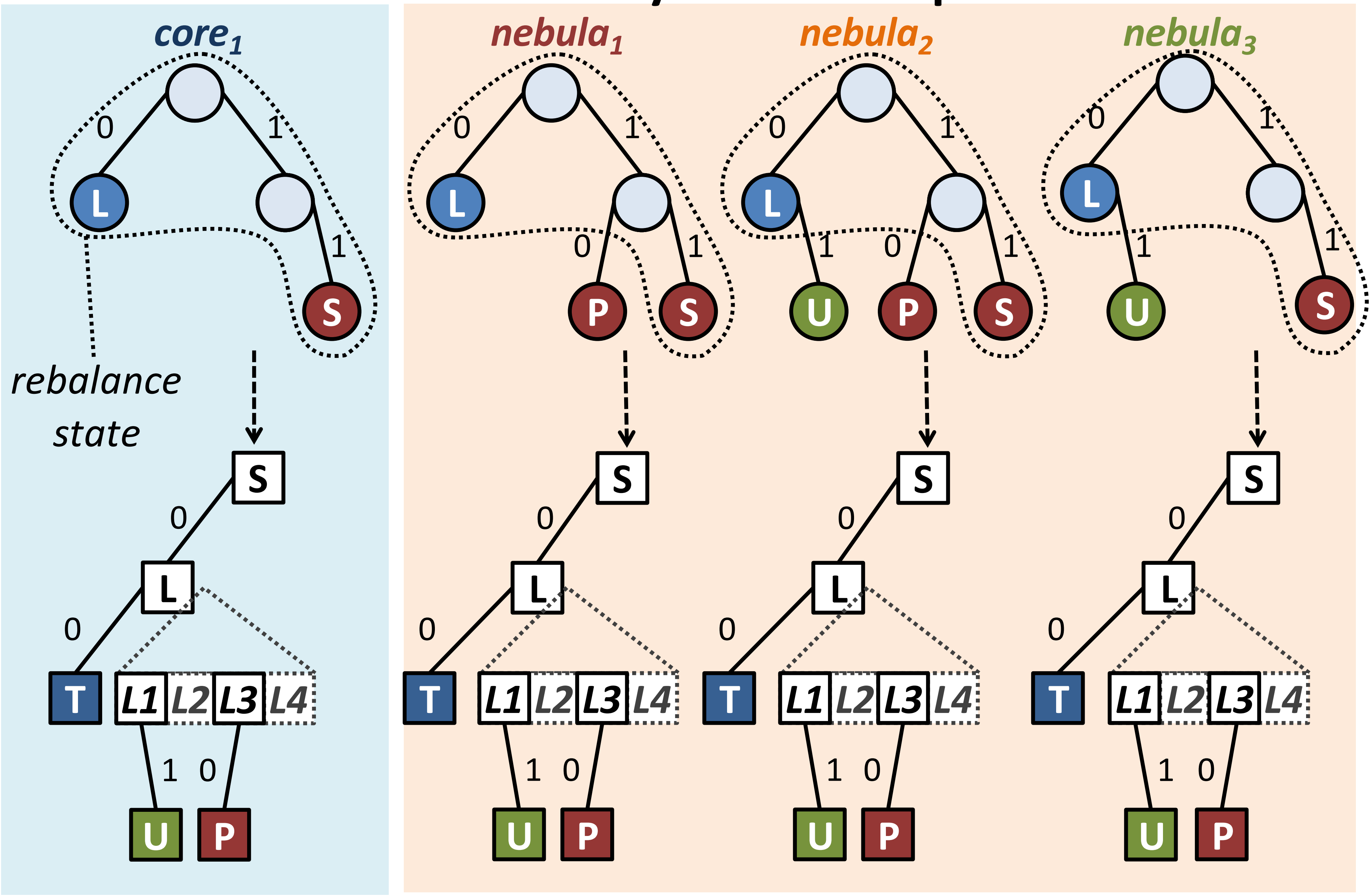
R-Translate & symbolic positions



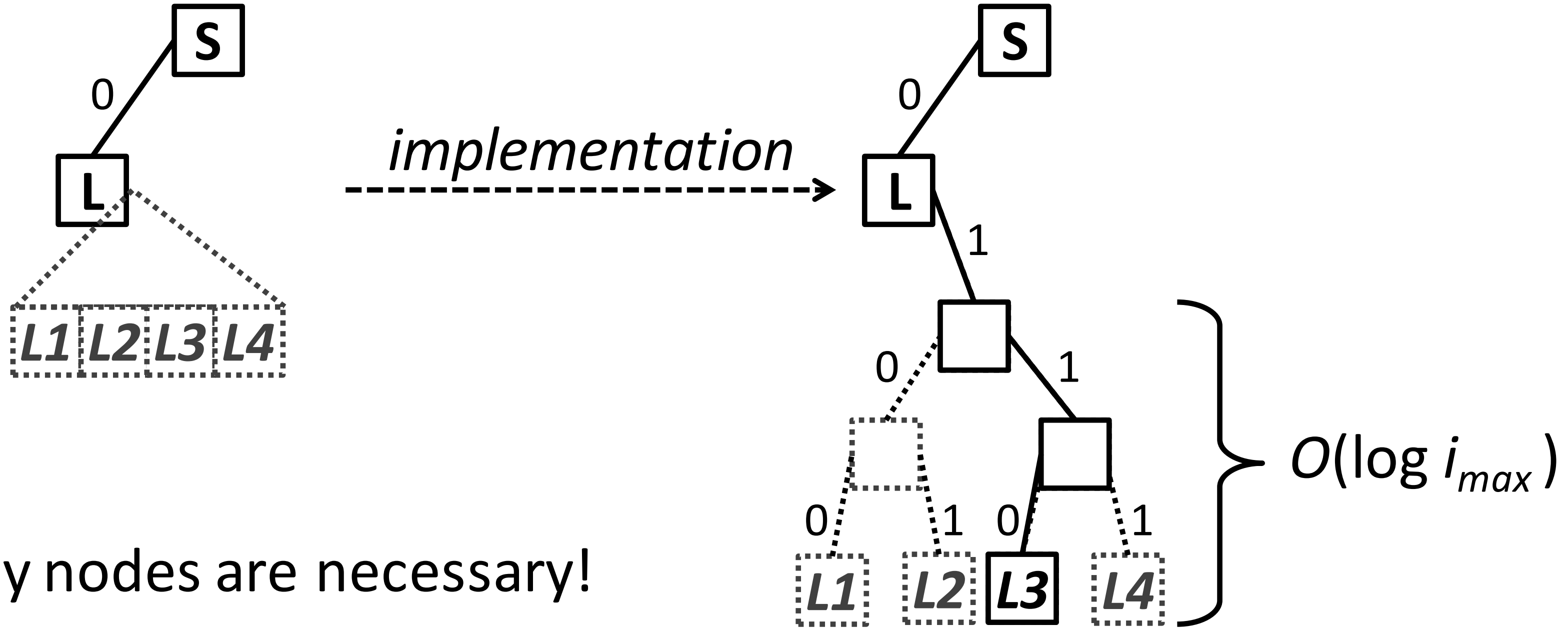
R-Translate & symbolic positions



R-Translate & symbolic positions



R-Translate: symbolics implementation



- Empty nodes are necessary!
- **Do we still discard more empty nodes than introduce?**
  - No update concurrent to rebalance => no empty nodes
    T1: No new operations => tree is minimal in 2 epochs
  - Concurrent update => create only empty nodes on the path
  - Encode *symbolic positions* as a balanced tree & other opt.
    C1: O(*n*) symbolics for *n*-size tree.
    C2: O(*n*) utilized symbolics create at most O(*n*) empty nodes

Summary

- Problem faced:
  - Tree rebalanced in some replicas (new ids), while concurrently updated in others (using old ids)
- Approach:
  - *Catch-up* protocol to integrate rebalance on all replicas
- Novel *R-translate* algorithm:
  - Identify and utilize *rebalance state,* use *symbolic positions*
  - Prototype catch-up implementation
- Future work?
  - Evaluation of *symbolic positions* implementation
  - Formal order-preservation proof

Appendix: the unbalance problem

- Use sparse tree and heuristic to assign *PosID* [Weiss et. al, ‘09] or Treedoc with similar heuristics [Shapiro, Preguiça et. al, ’09]
  - Work on evaluated workload; at the cost of possible anomaly
- Use list instead of a tree [Roh et. al, ‘10]
  - Different costs and convergence characteristics?
- Rebalance the tree [Shapiro, Preguiça et. al, ‘09]
  - System-wide consensus; inherent limitations
  - The core-nebula idea [Leția et. al ‘09]; incorrect translation
- **This work brings:**
  - More formalization of the core-nebula for asynchronous systems
  - Flaws revealed in naive algorithms
  - Translation requirements statement
  - Novel *R-translate* algorithm and first prototype implementation