Treedoc

A Commutative Replicated Data Type Designed for Cooperative Text Editing

Nuno Preguiça (UNL)
Marc Shapiro (INRIA / LIP6)
Marek Zawirski (INRIA / LIP6)

STREAMS kick-off meeting
29 -30 November, 2010, Nancy

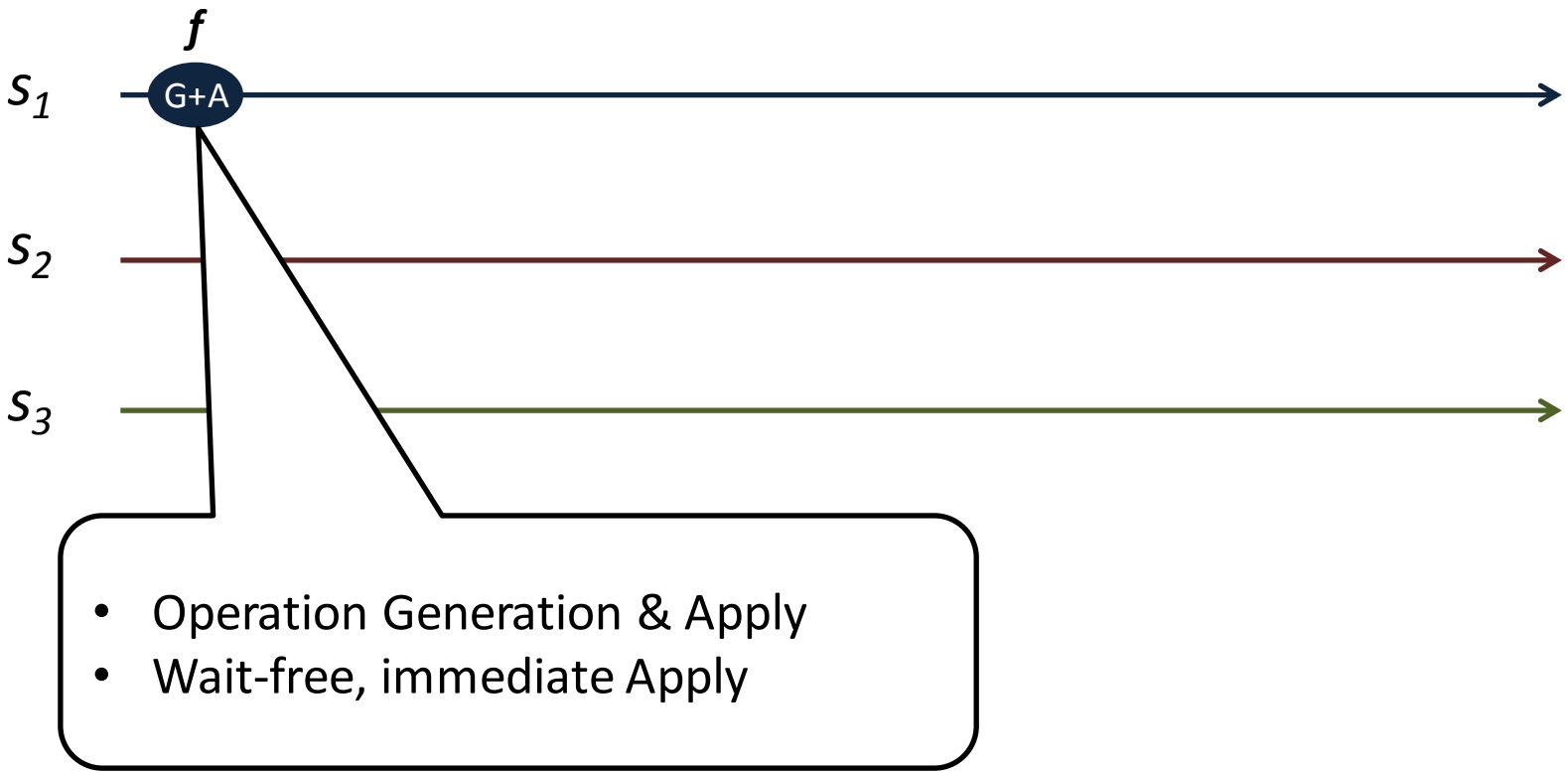
Summary

- Wait-free collaborative editing solution
- Eventually consistent *(CRDT)*
- Sequence
- Separates concurrently-inserted subsequences

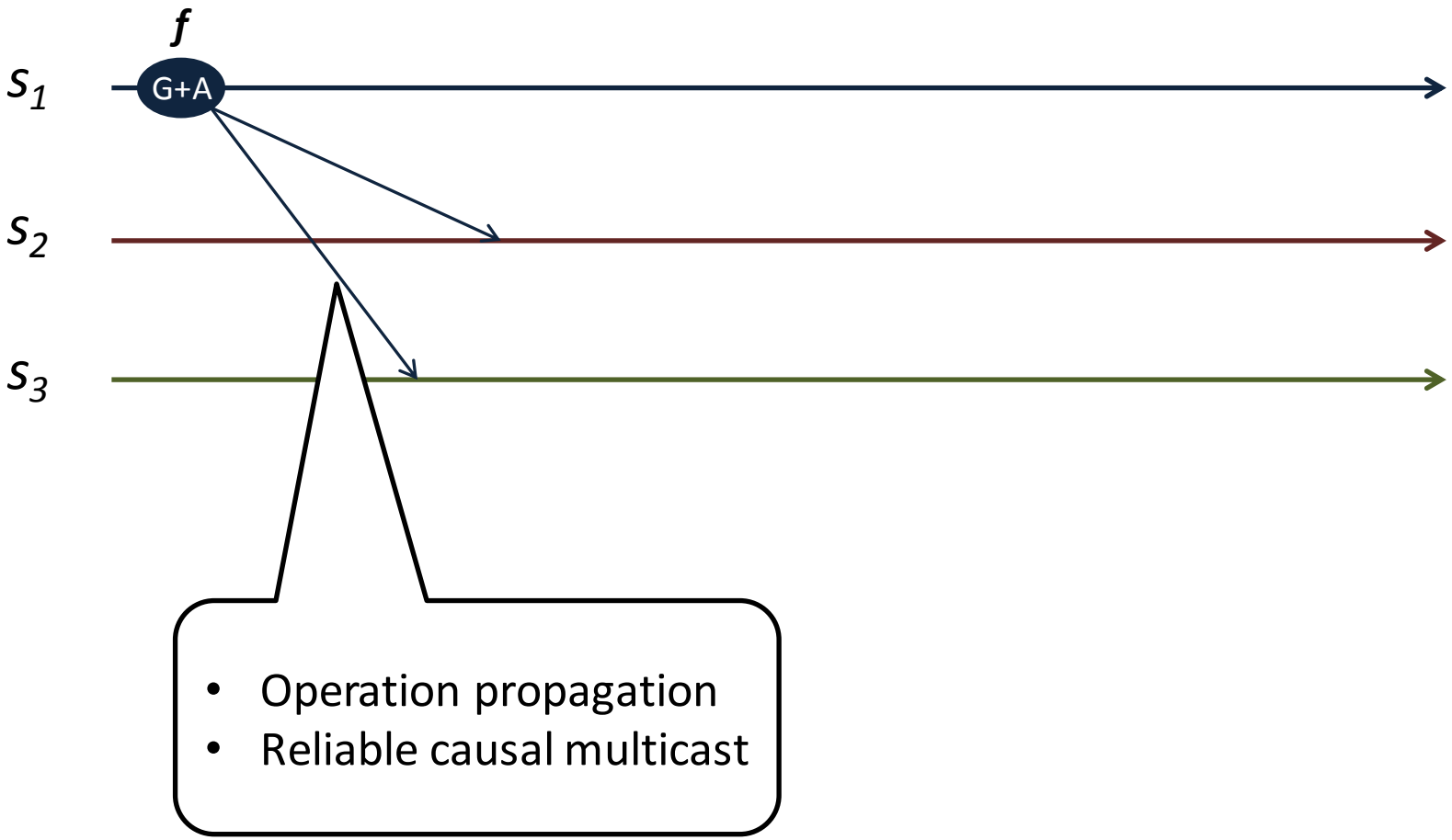
System model

- Dynamic set of sites (replicas)
- No permanent crashes

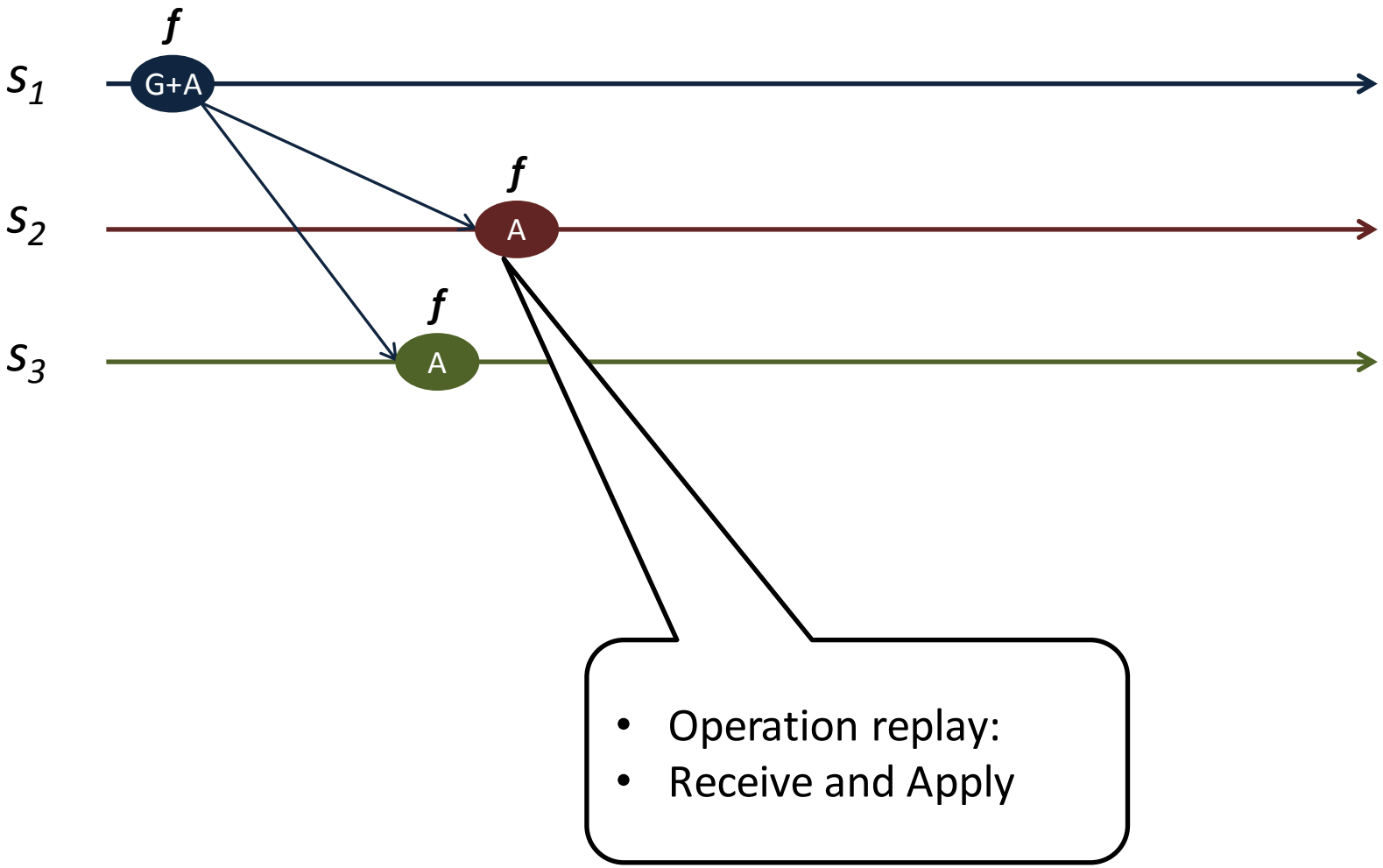
System model



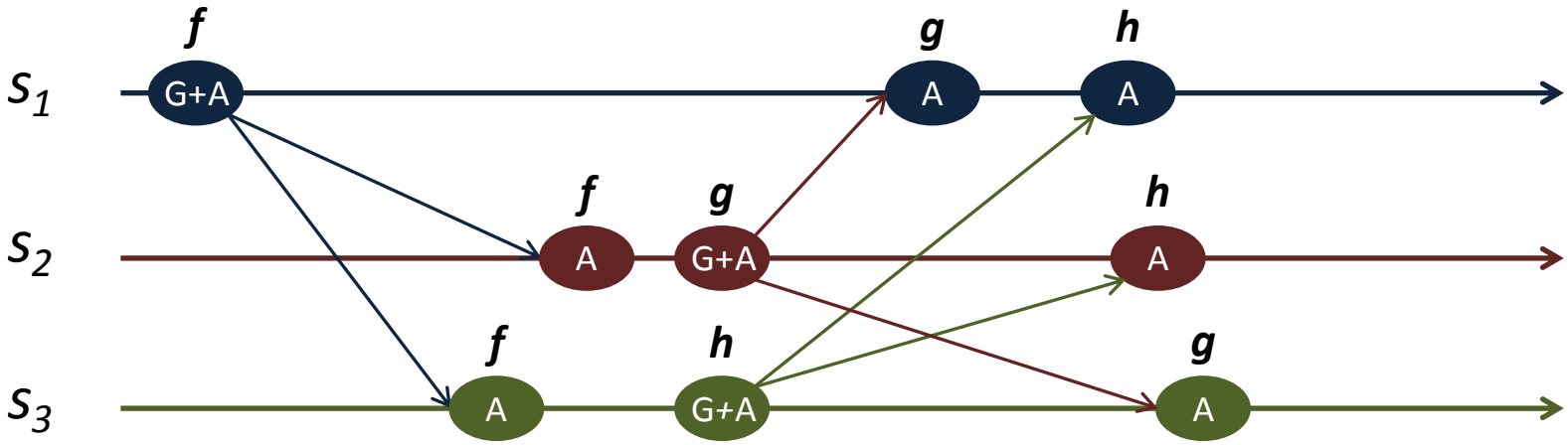
System model



System model



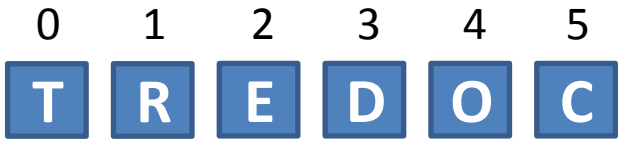
System model



- Concurrent operations **commute**
- Replicas converge

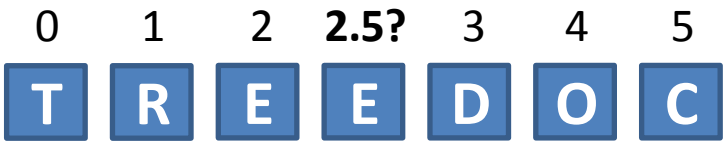
Treedoc – Sequential Buffer CRDT

- **Application view:**
  - Sequential buffer of atoms
  - *buffer.insert(index, atom)*
  - *buffer.remove(index)*
  - Inconvenient for replication



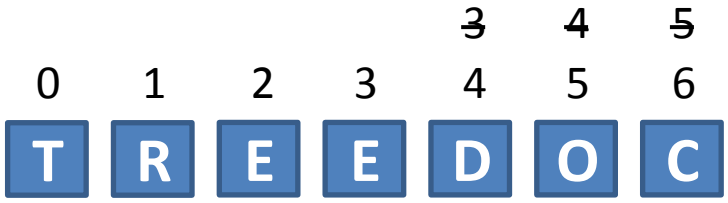
Treedoc – Sequential Buffer CRDT

- **Application view:**
  - Sequential buffer of atoms
  - *buffer.insert(index, atom)*
  - *buffer.remove(index)*
  - Inconvenient for replication



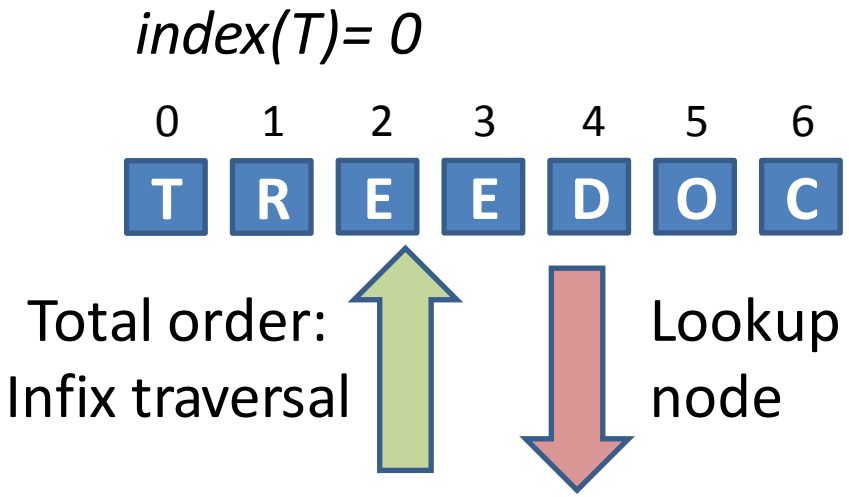
Treedoc – Sequential Buffer CRDT

- **Application view:**
  - Sequential buffer of atoms
  - *buffer.insert(index, atom)*
  - *buffer.remove(index)*
  - Inconvenient for replication

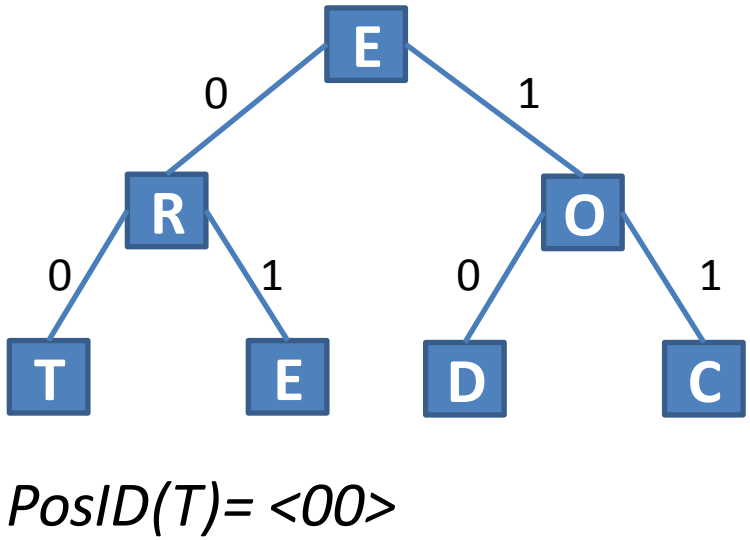


Treedoc – Sequential Buffer CRDT

- **Application view:**
  - Sequential buffer of atoms
  - *buffer.insert(index, atom)*
  - *buffer.remove(index)*
  - Inconvenient for replication



- **Internal state representation:**
  - Grow-only tree of atoms
  - *tree.insert(PosID, atom)*
  - *tree.remove(PosID)*
  - Stable, unique positions *PosID*
  - Commutativity

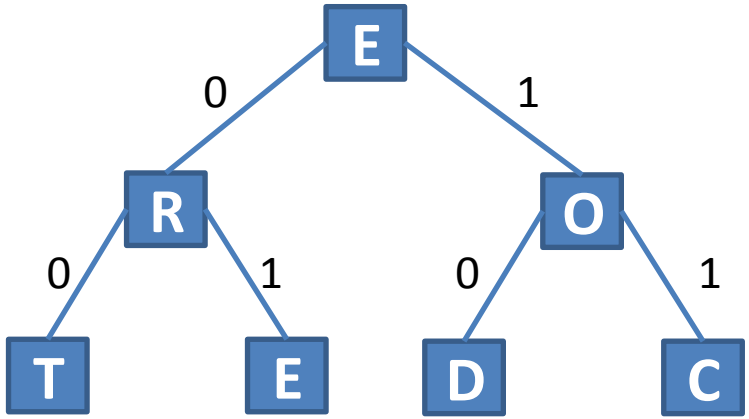


Operations on Treedoc: *insert*

buffer.insert(index, atom) -> tree.insert(PosID, atom)

- Create a new **leaf node** *PosID* such that it corresponds to *index* (it is always possible)
- Put an *atom* there
- Propagate to other replicas

e.g. buffer.insert(4, "S")



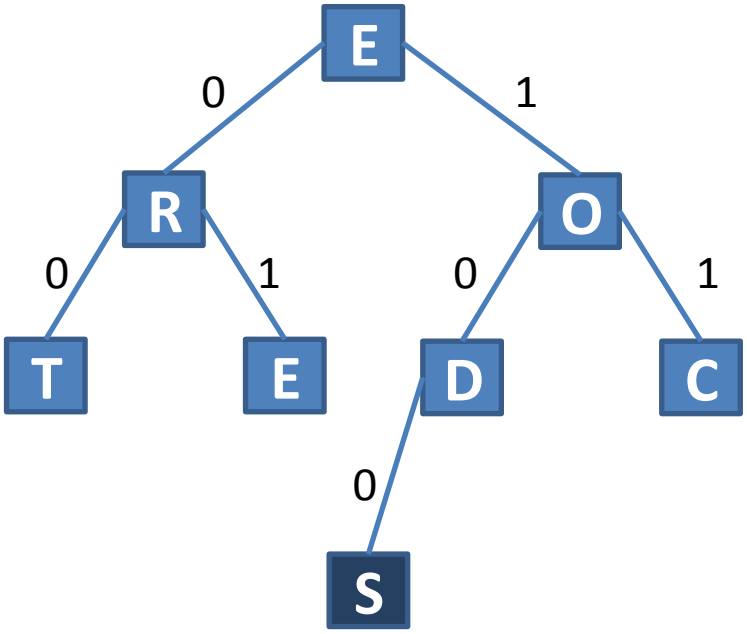
Operations on Treedoc: *insert*

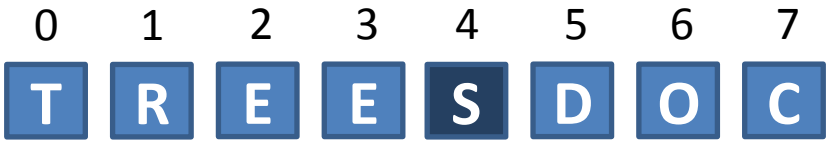
buffer.insert(index, atom) -> tree.insert(PosID, atom)

- Create a new **leaf node** *PosID* such that it corresponds to *index* (it is always possible)
- Put an *atom* there
- Propagate to other replicas

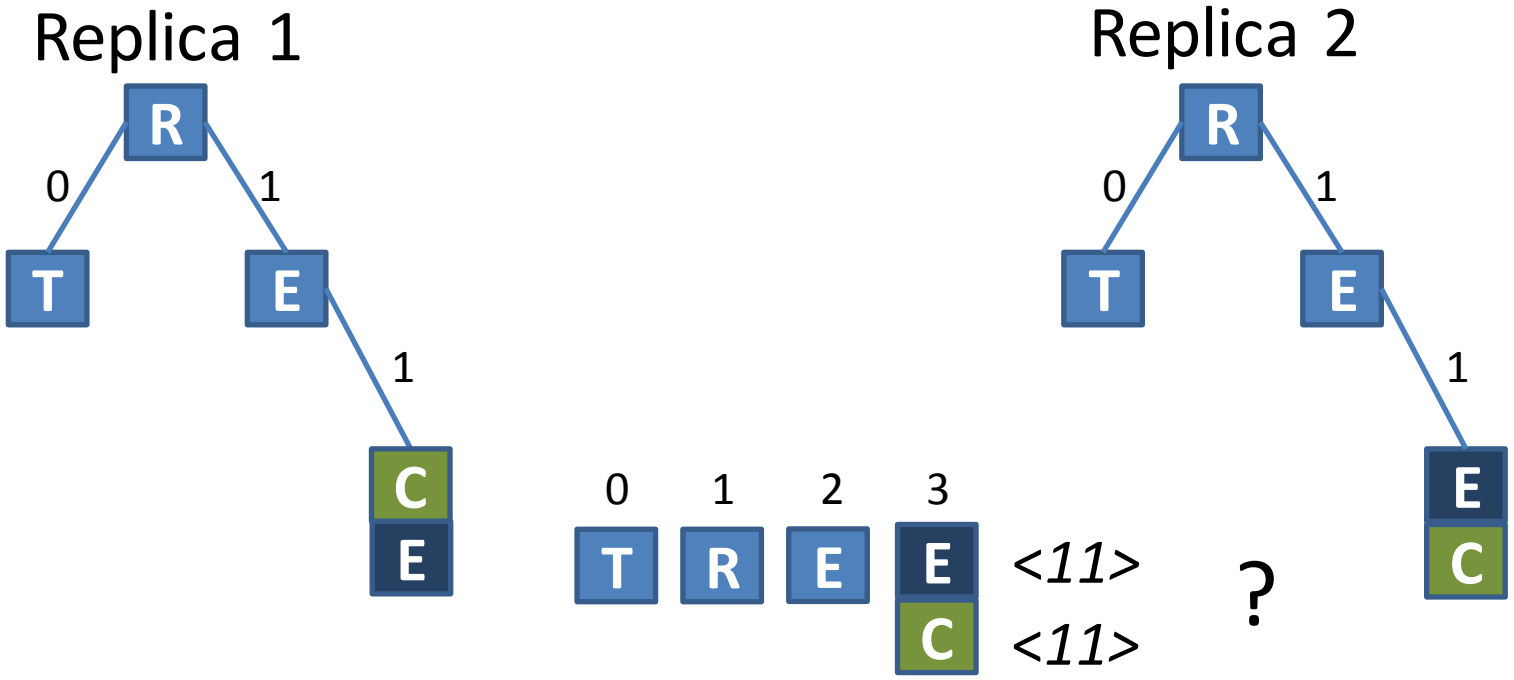
e.g. buffer.insert(4, "S")

tree.insert(<100>, "S")

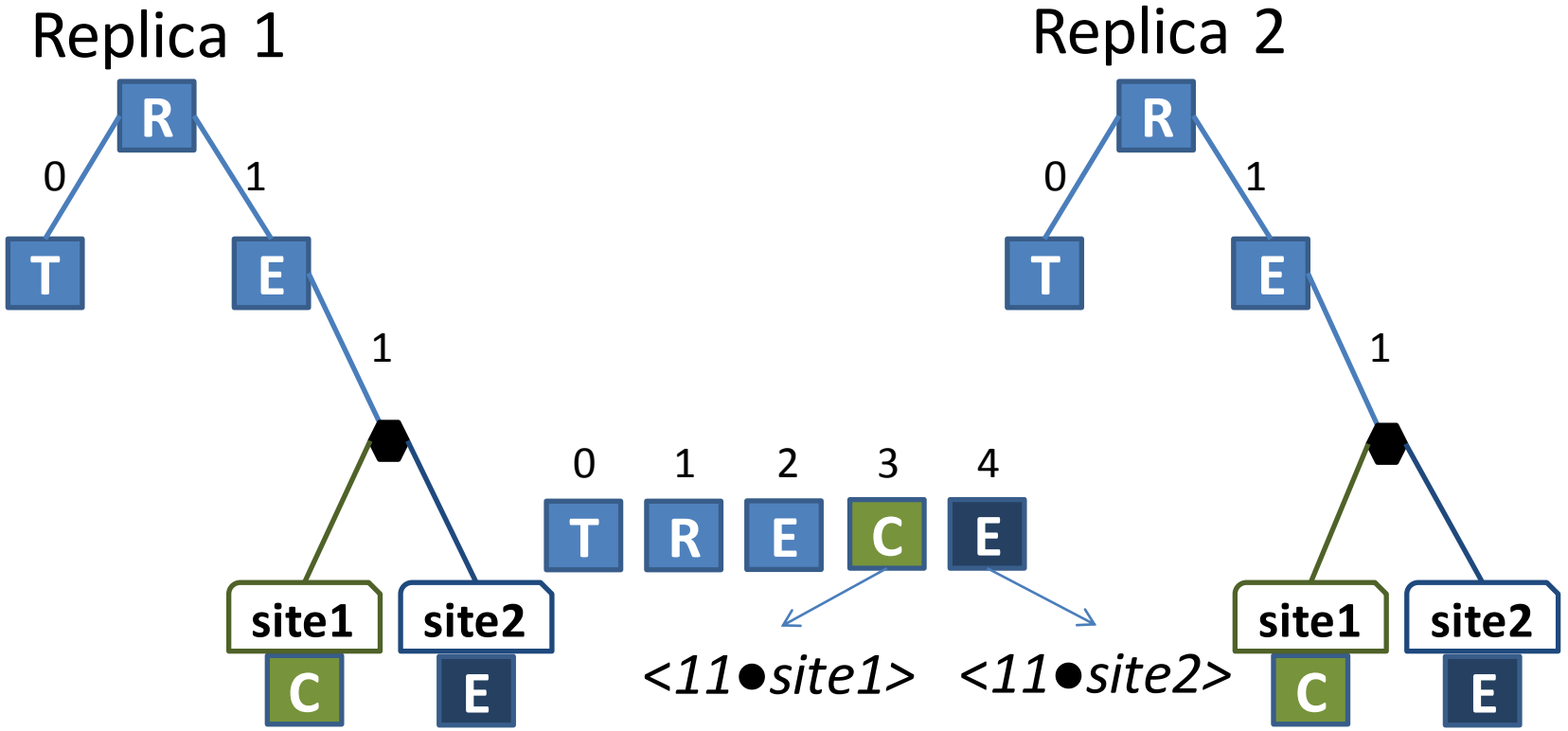




Operations on Treedoc: conc. *insert*



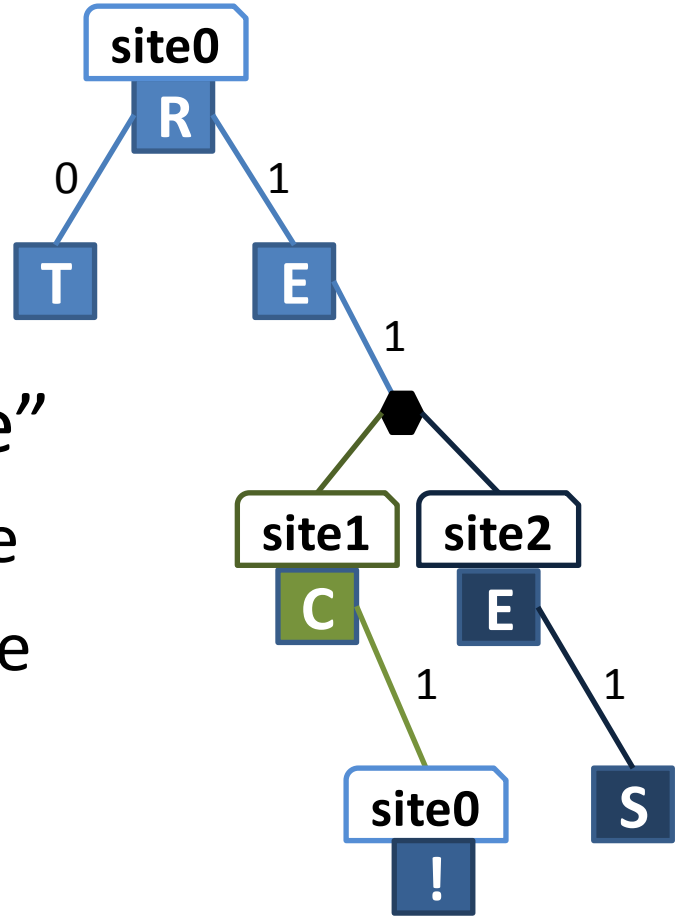
Operations on Treedoc: conc. *insert*

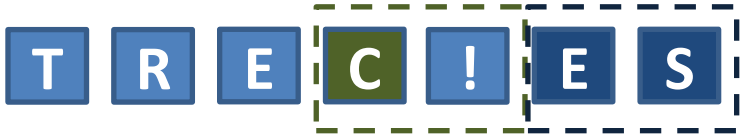


- Always include unique ID in inserted tree node
  - In case of "conflict" -> resolves into arbitrary order
- Could be a unique site ID

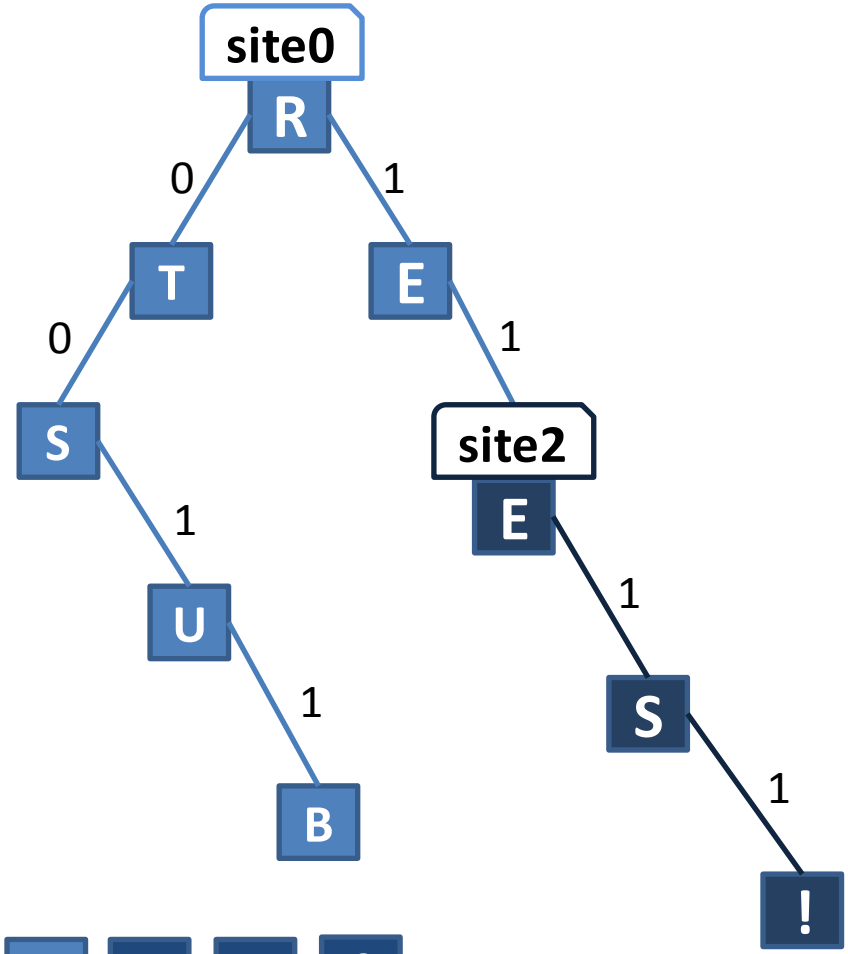
Treedoc: A Layered Tree

- Treedoc composed of layers:
  - Compact binary tree
  - Sparse site ID layer
- Site ID denotes “private space”
  - Space owner uses compact tree
  - Others create new private space
  - Separates concurrent inserts





Treedoc: Independent Private Spaces



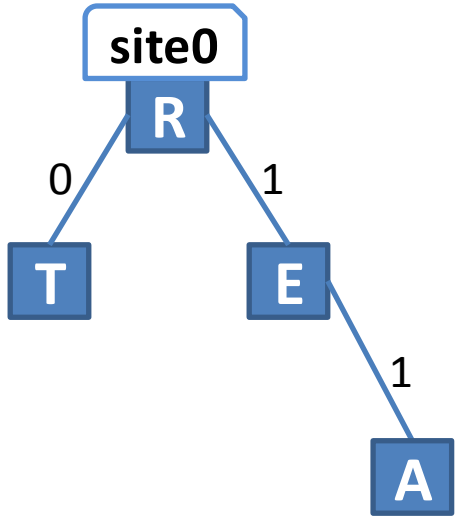
S U B T R E E S !

Operations on Treedoc: *remove*

- Remove atom
- Keep *unused* node

  (alternative designs possible)
- Ignore in application view

  e.g. *buffer.remove(1)*

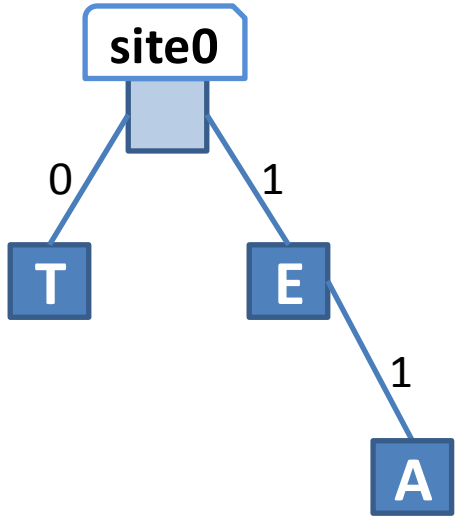


Operations on Treedoc: *remove*

- Remove atom
- Keep *unused* node

  (alternative designs possible)
- Ignore in application view

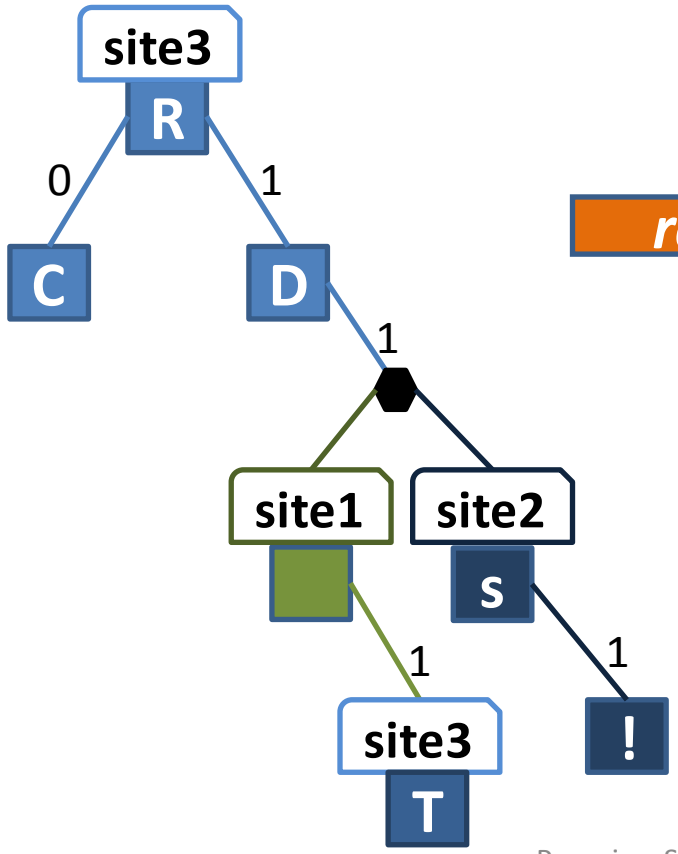
  e.g. *buffer.remove(1)*

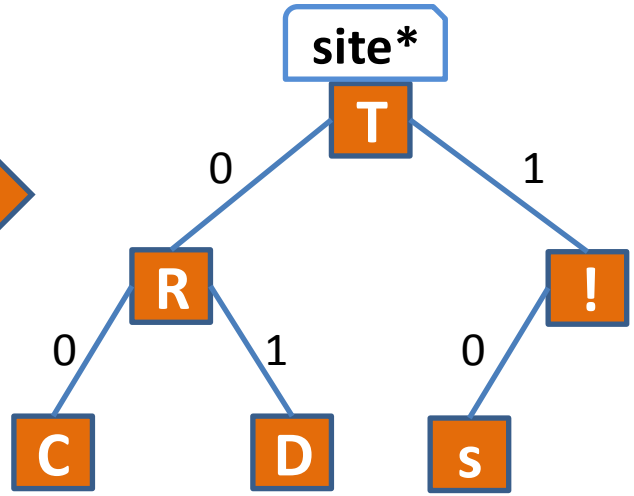


Why Is Tree Rebalance Required?

Tree may become unbalanced
Unused nodes waste space
Number of site IDs increases

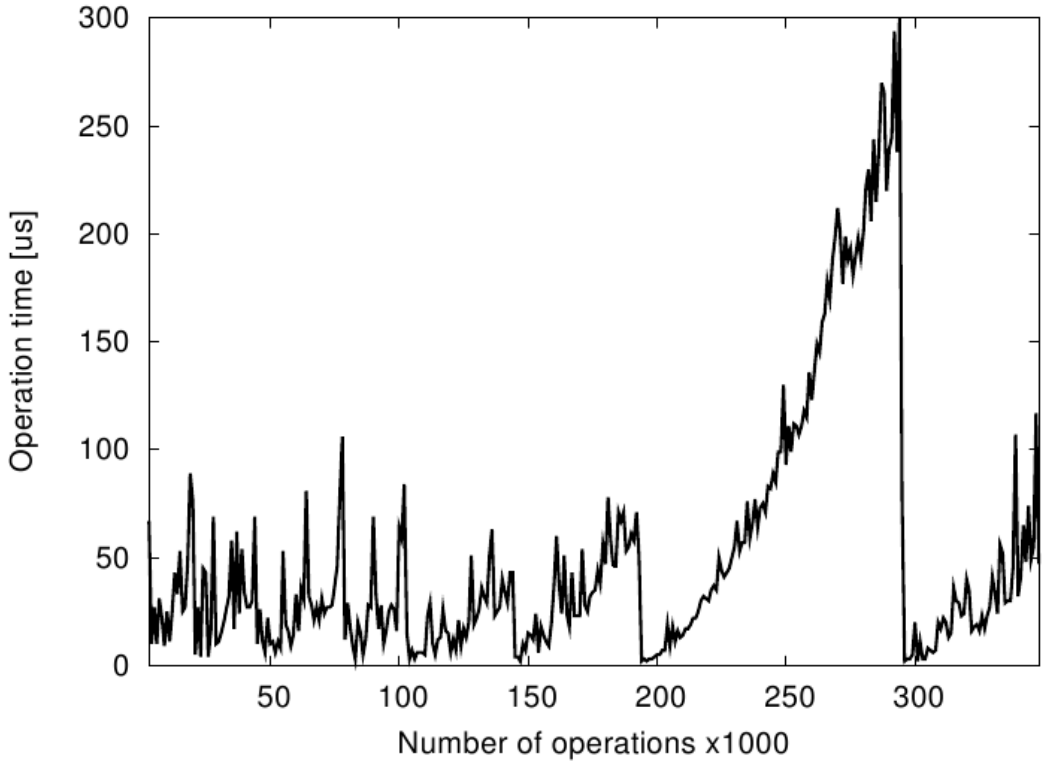
Access time grows
PosID length grows (communication cost)





- Unused nodes discarded
- Minimal tree maintaining identical order
- Single space – compact tree

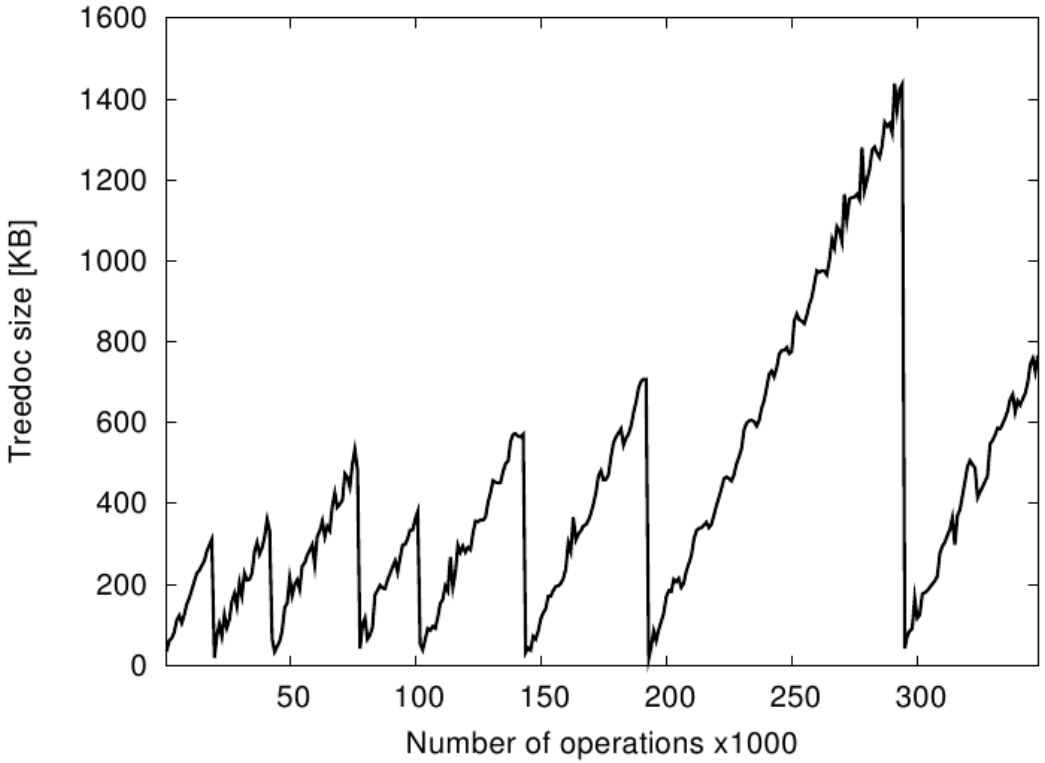
Measurements: GWB Wikipedia page



[Letia 09]
old impl.

Sequential replay of Wikipedia traces
(atom: paragraph)

Measurements: GWB Wikipedia page



[Letia 09]
old impl.

Rebalance is beneficial, but ***non-commutative*** with edits.
Identifiers are changed => consensus is required.
Consensus => blocking operations, difficult in dynamic system.

The Core-Nebula Architecture

Idea: limit consensus to a smaller number of sites

Sites are divided into two disjoint sets:

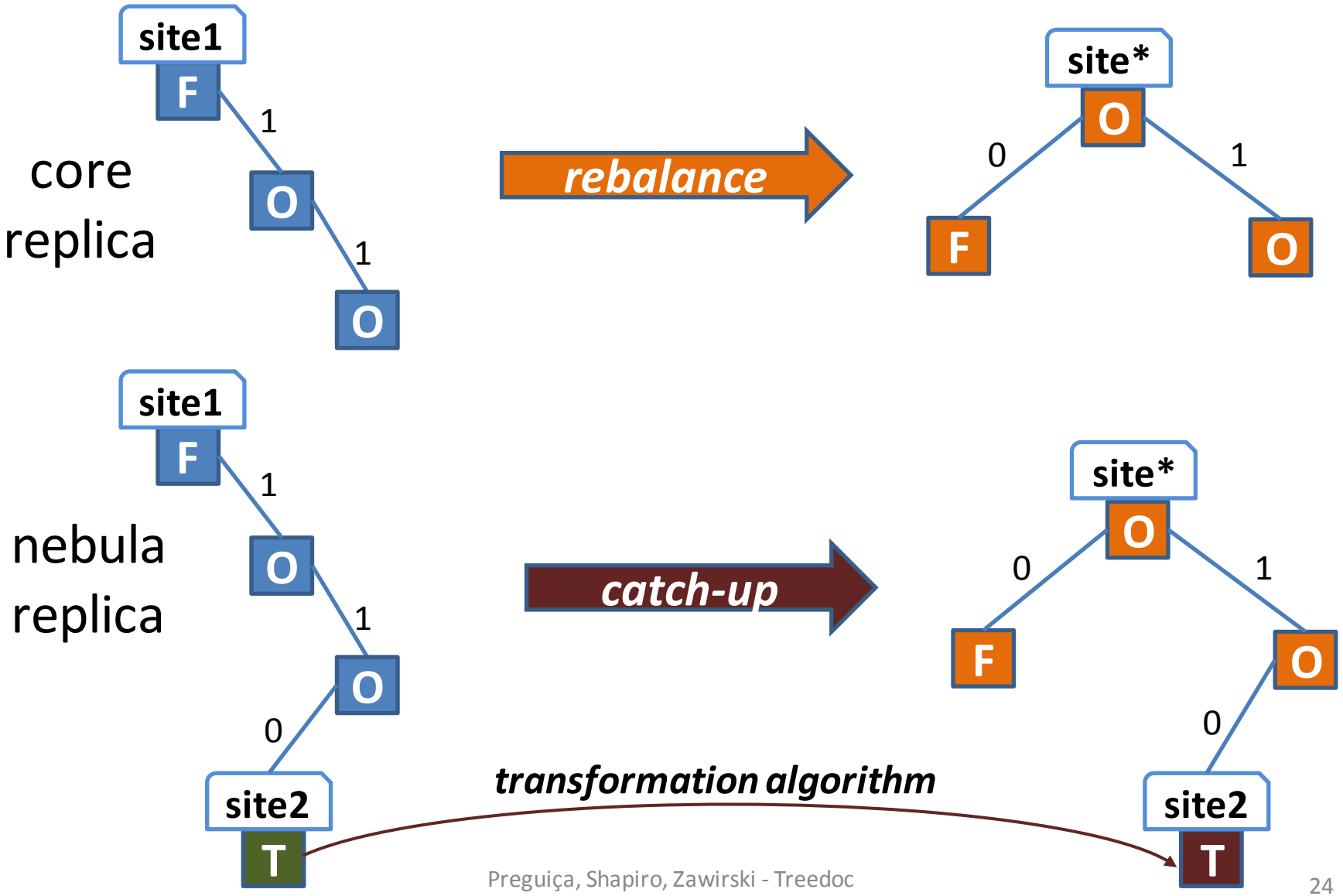
CORE

- a group managed by membership protocol
- stable
- both perform tree operations & agree on tree *rebalance*
- easier agreement

NEBULA

- sites freely join and leave
- dynamic
- perform tree operations only
- Informed about *rebalance,* perform ***catch-up*** protocol to integrate conc. changes

The Core-Nebula: catch-up



Current status

- Prototype core-nebula implementation:
  - 1 core node, simplified communication
  - 5 KLOC Java code (1.5 KLOC core-nebula code)
  - Test suite (boundary cases)

Work in progress:

  - Core-nebula: evaluation, proof generalization?
  - Combining private spaces and core-nebula (verification)
  - Optimization of *PosIDs* encoding
  - More theory on private spaces?

Summary

- Wait-free collaborative editing solution
- Eventually consistent *(CRDT)*
- Sequence
- Separates concurrently-inserted subsequences