

Controlled conflict resolution for replicated document

Stéphane Martin, Mehdi Ahmed-Nacer, Pascal Urso
Score team - Université de Lorraine - CNRS – INRIA
LORIA, Nancy

Concordant/Streams meeting
Nantes France

Collaboration on Shared document

- Mutable shared data in distributed system
- CAP theorem [Brewer 2000]
 - strong Consistency
 - Availability
 - High responsiveness
 - Partition tolerance
 - Mobile - Disconnected work
- A solution : Optimistic Replication (Eventual consistency)

Optimistic replication

- A replica per application instance
- Immediately available for updates
- Updates sent to other replicas
- Consistency Model
 - (Strong) Eventual Consistency
- Replicated merge mechanism
 - Concurrent modifications
 - Possibly “conflicting”

Concurrency Conflicts

Replica A

- Removes an element
- Inserts a text
- Adds a paragraph *P*
- Sets a title
- ...

Replica B

- Removes the element and re-adds it (e.g. undo)
- Inserts another text at the position
- Moves the section *P*
- Sets a title
- ...

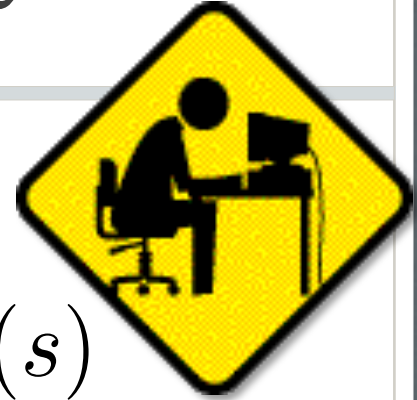
more complexity
=
more conflicts

Proof on ad-hoc complex systems

- Commutative Replicated Data Type

$\forall op_1, op_2 \in Op, S$ is state

$$op_1 || op_2 \Rightarrow [op_1, op_2](s) = [op_2, op_1](s)$$



- Operational Transformation $\forall op_1, op_2 \in Op, S$ is state

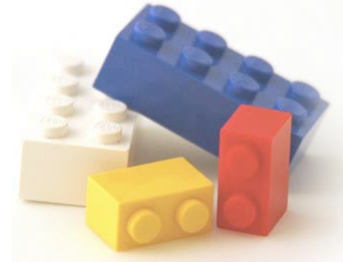
$$op_1 || op_2 \Rightarrow [op_1, IT(op_2, op_1)](s) = [op_2, IT(op_1, op_2)](s)$$

$$op \in Op,$$

$$IT(IT(op, op_1), IT(op_2, op_1)) = IT(IT(op, op_2), IT(op_1, op_2))$$

- More different operation \rightarrow more complex proof

Our proposition



- Decouples eventual consistency from conflict resolution
- Constructs a new data type using simpler one
- Layered approach
 - Bottom layers for eventual consistency
 - Layers per data type constraint

Conflicts to constraints

Replica A

- Removes an element
- Inserts a text
- Adds a paragraph P
- Sets a title
- ...

Replica B

- Removes the element and re-adds it (e.g. undo)
- Inserts another text at the same position
- Removes the section P belongs
- Sets a title
- ...

It's a set

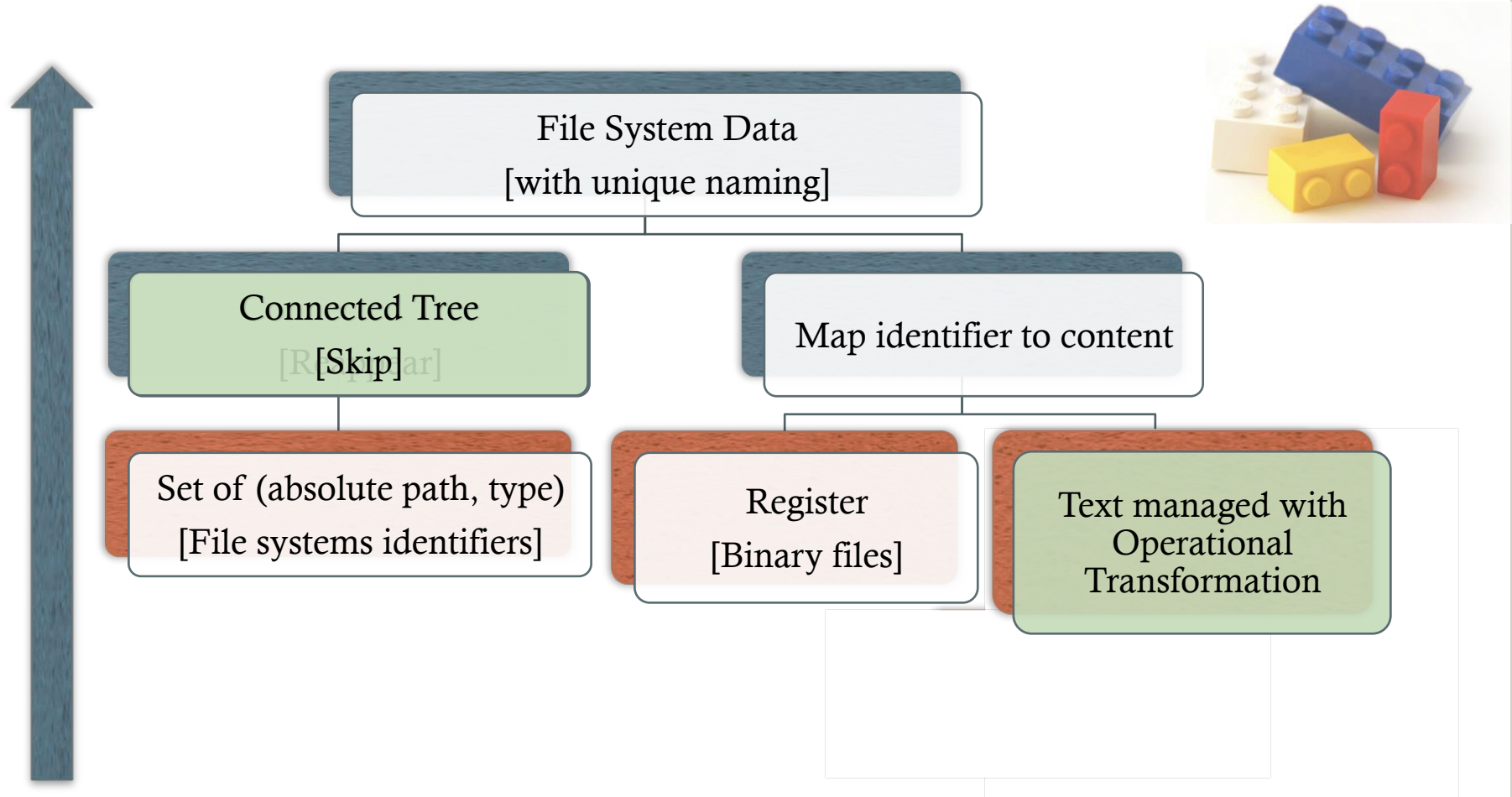
Elements are totally ordered

All elements are connected

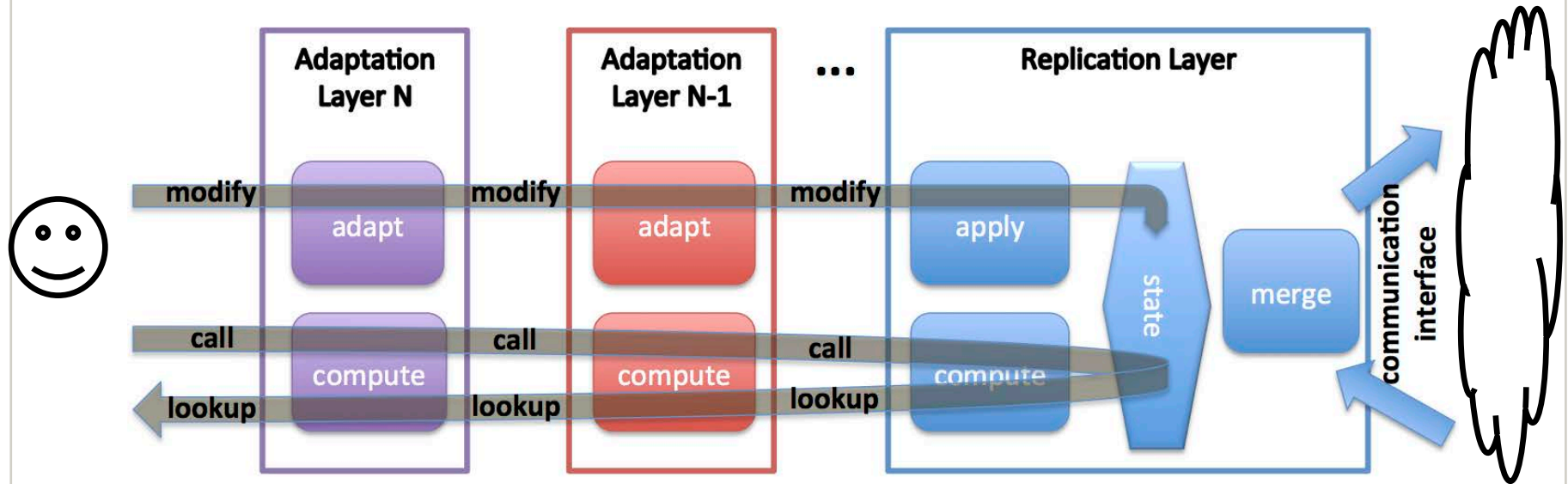
Only one title

Layered approach

Example : file system data

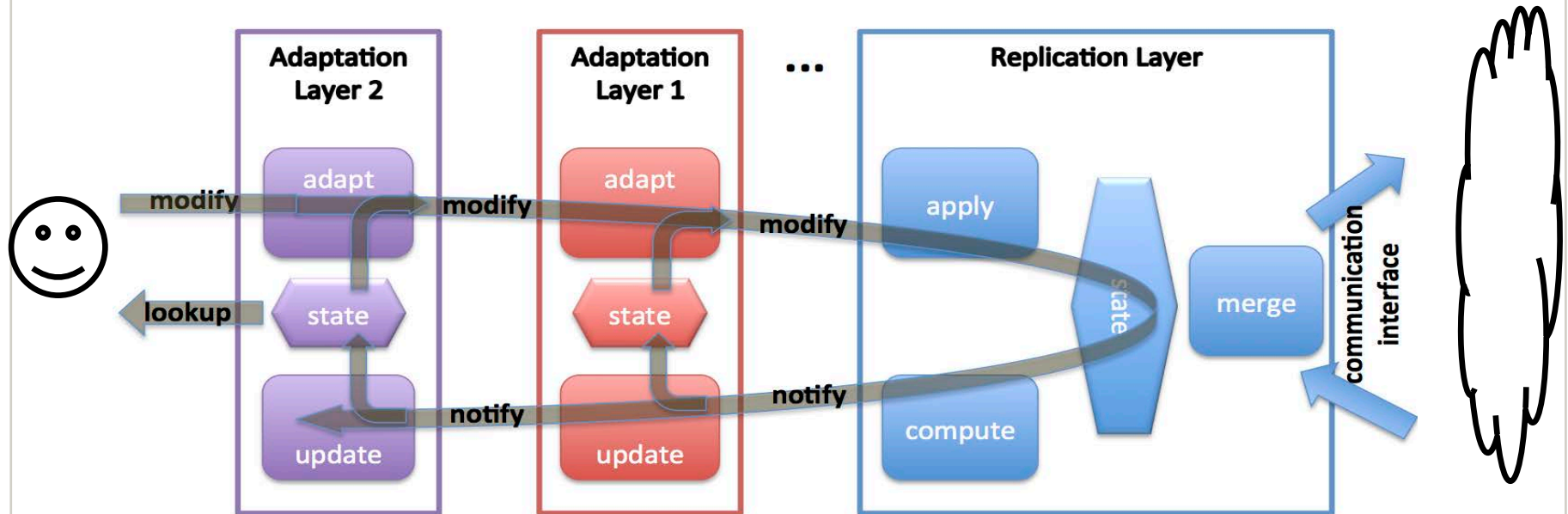


Stateless Layers



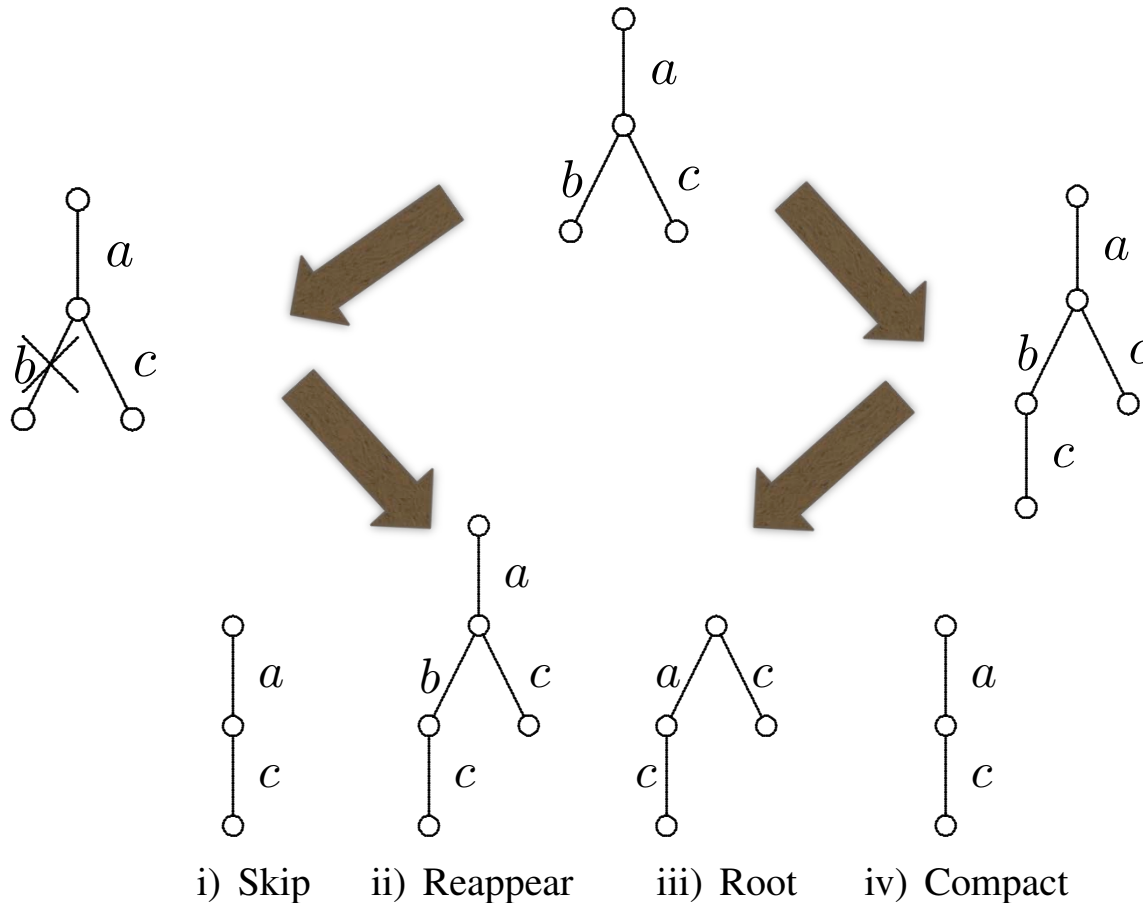
- Replication layer in charge of merge
- Adaptation layer computes a lookup
- Straight forward strong eventual consistency
- Fits state-based replication

Incremental Layers



- Adaptation layer state is updated
- No concurrency management in adaptation layer
 - “all local”
- Fits operation-based replication
- Proof is more difficult. (proof that it's equivalent of stateless layers)

Adaptation layer example: Trees policies

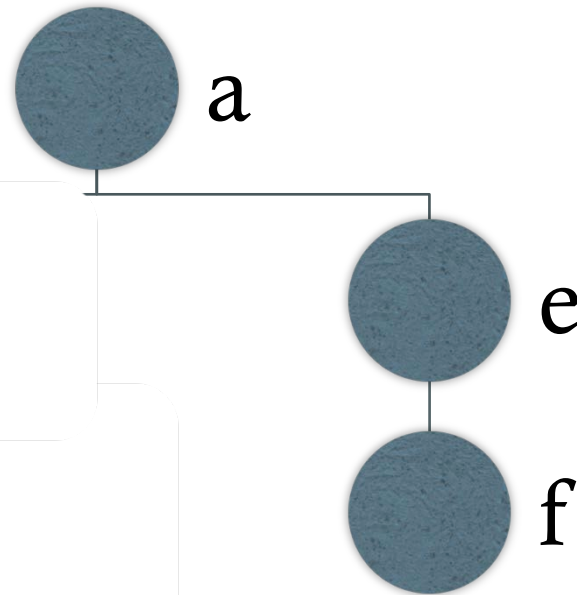


Incremental reappear policy

- Mark reappeared nodes – “ghosts”
- Path added in the inner set
 - Creates *recursively* missing father(s)
 - Creates node if needed (else unmark)
- Path removed in the inner set
 - If corresponding node got no child
 - Removes it
 - If it was the last child of a reappeared node, removes *recursively* its father
 - Else marks it



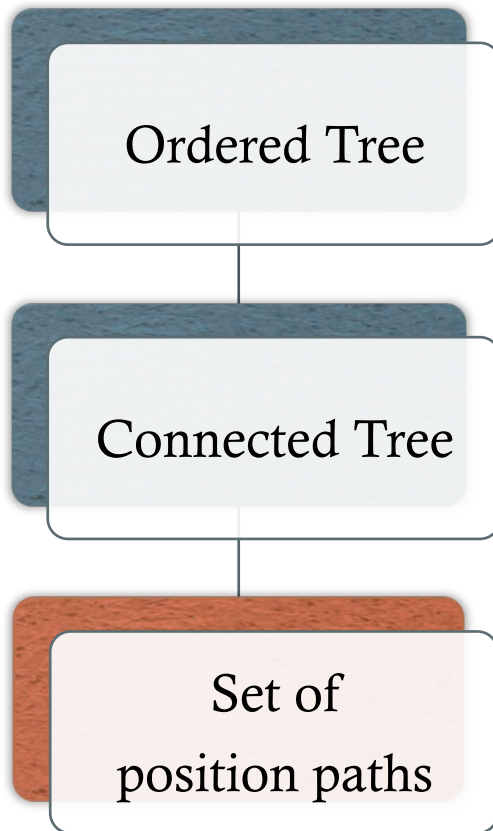
Example of incremental reappear policy



- Del(ab)
- Del(abc)
- Del(abd)
- Add(abc)

Ghost without children is deleted

Ordered Trees



- Position path are list of couple (position, label)
- Position are unique, totally ordered and define in a dense space
- Several existing possibility
 - Logoot [Weiss et al 09],
 - FCEdit [Martin, Lugiez 09],
 - Treedoc [Preguiça et al 09],
 - PPS [Wu et al 10], ...

Experiments

- Evaluate performance of the approach
 - Overhead of layers and connection algorithms
 - 4 connection policies : skip, reappear, root, compact
 - 2 positions identifier : Logoot, WOOTH
- *Skip policy* compared to
 - Operational Transformation (OT) [Ressel et al 96]
 - “Naïve” tree based on TTF [Oster et al 06]
 - TreeOPT [Ignat, Morrie 03]
 - Conflict-free Replicated Data Types (CRDT) [Shapiro et al 11]
 - Tree CRDT : FCEdit [Martin, Lugiez 09]

Experiments Set up

- Open-source Benchmark (Java)

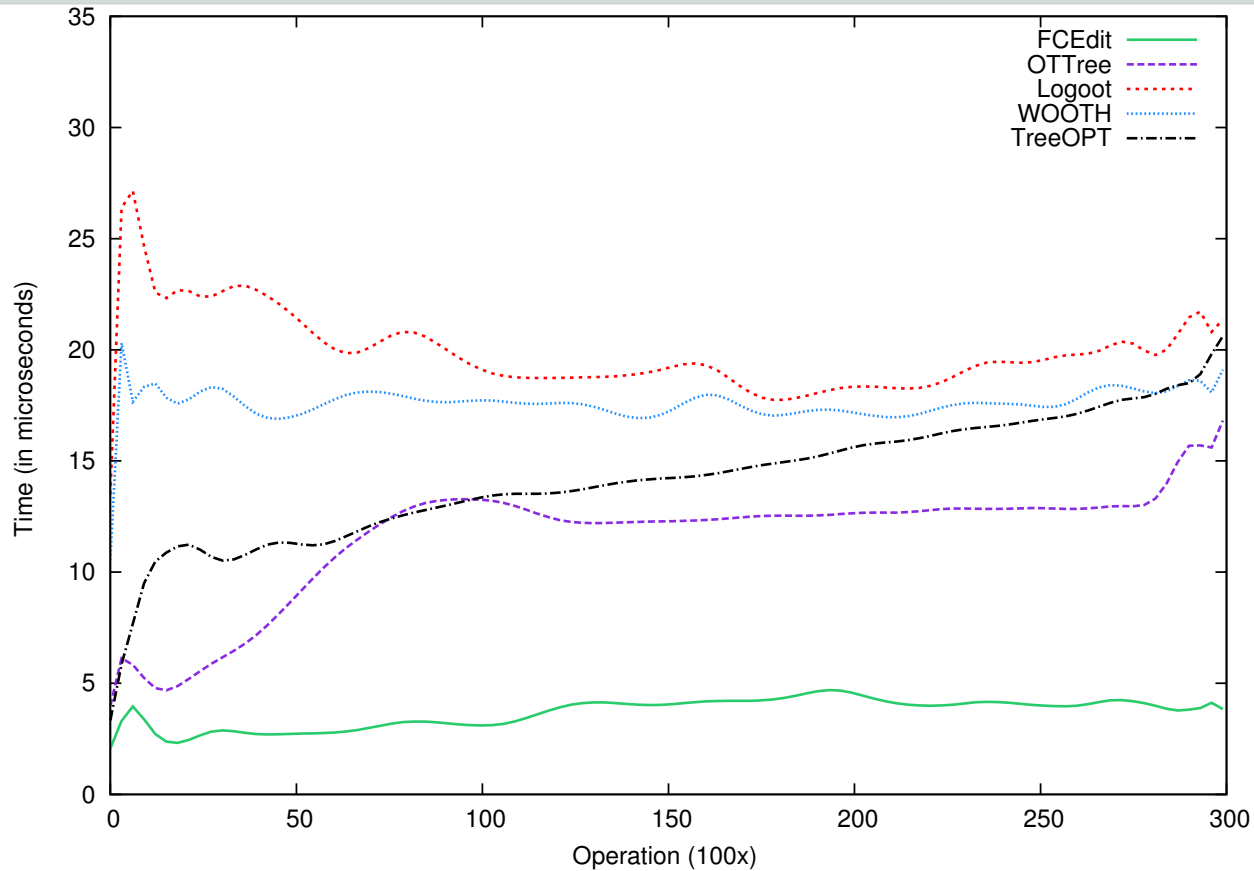
<http://github.com/score-team/replication-benchmark>

- Layered implementation

```
new PositionIdentifierTree(new WordTree(new ReappearPolicy(), new CounterSet()))
```

- One randomly generated trace (available)
 - 30,000 operations
 - Random paths
 - 80% insertions
- Intel(R) Xeon(R) 5160 dual-core
 - 3 executions

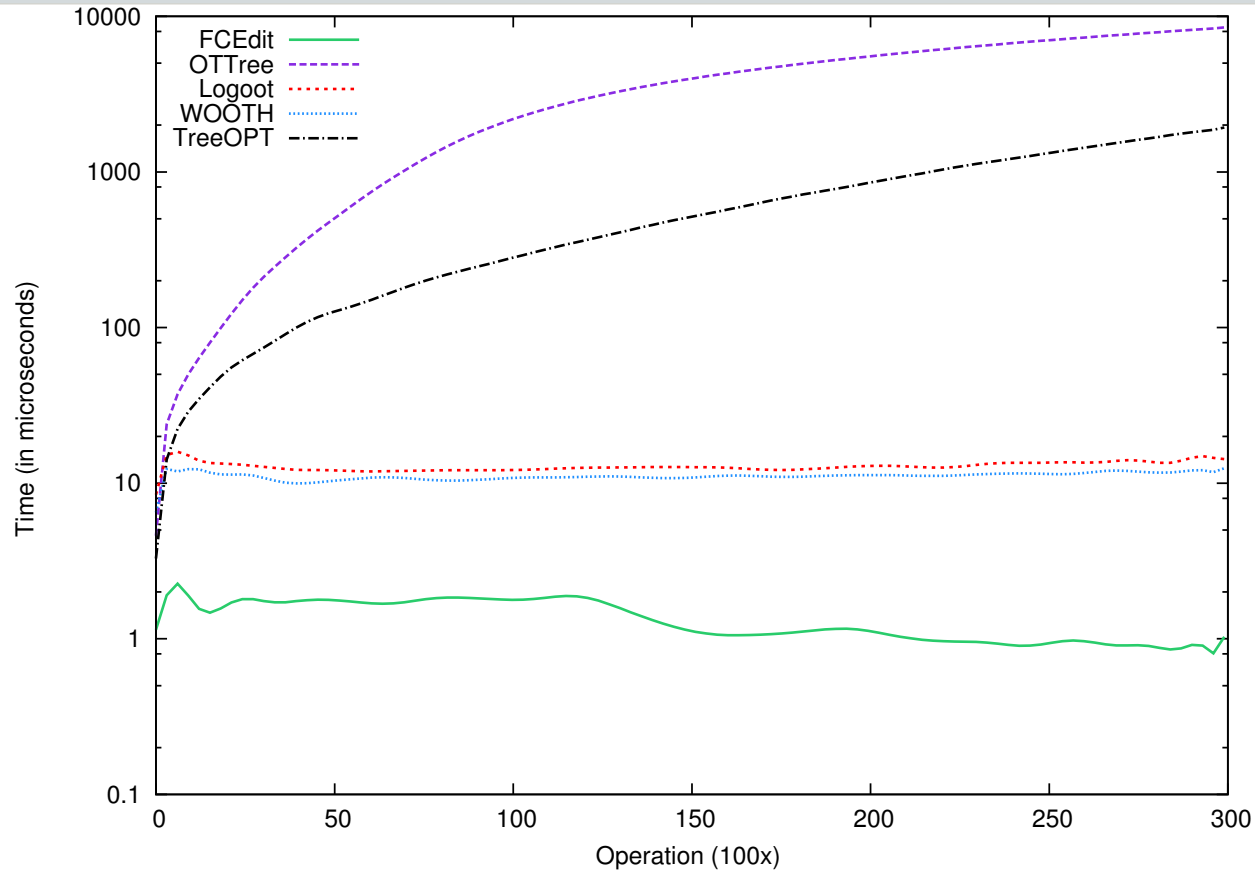
Experiments : Local execution time skip policy



- **All approaches are below 30 μ s**

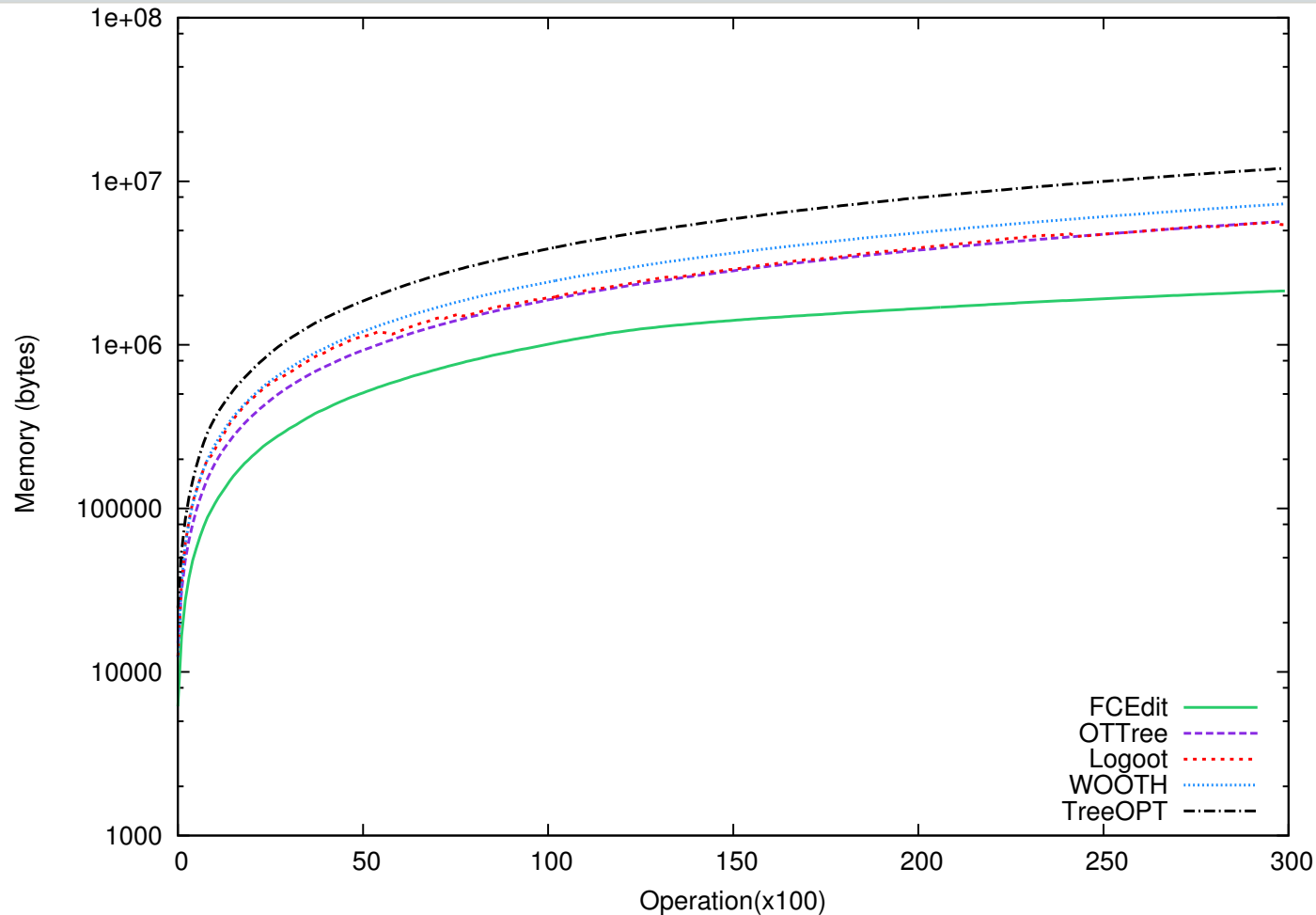
Experiments : Remote execution time

Skip policy (log scale)

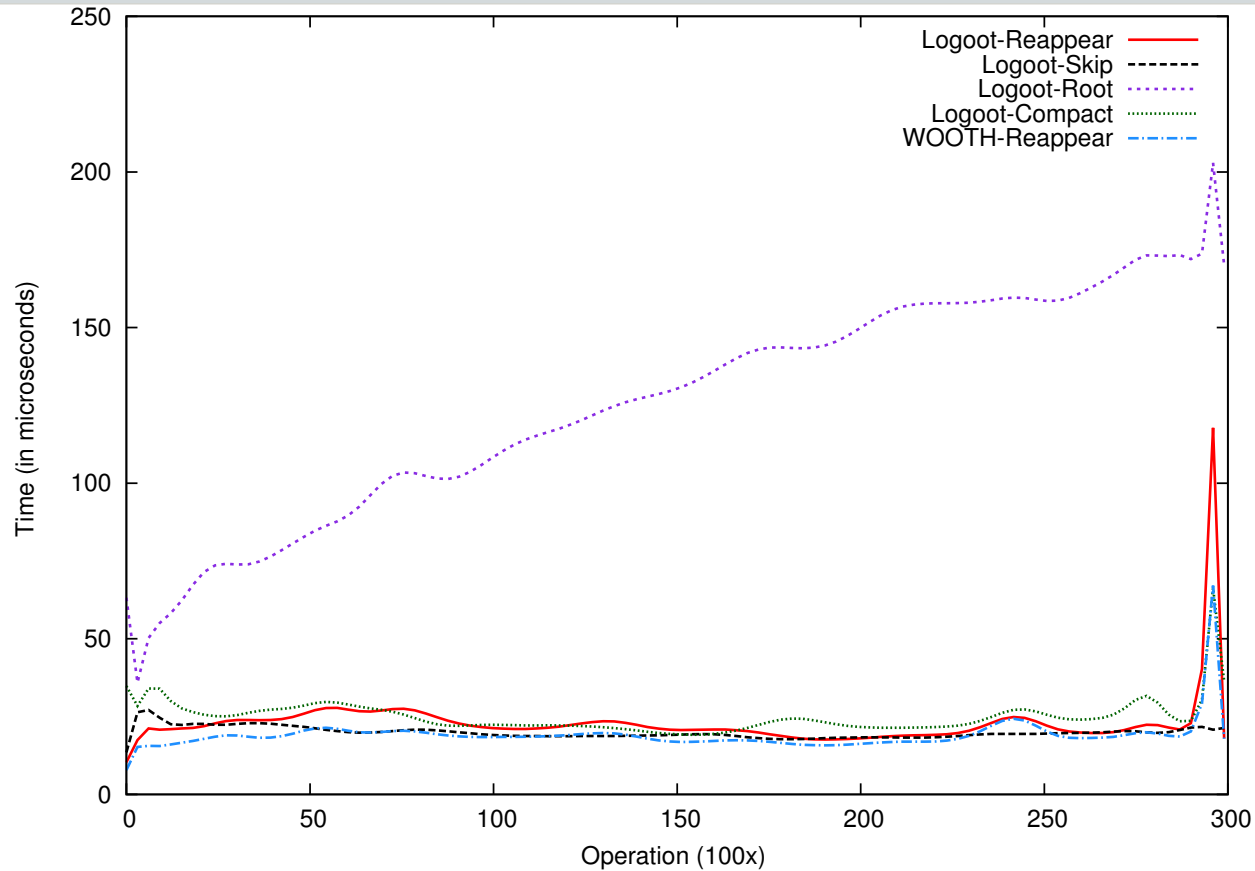


- Layers are below 30 μs

Experiments : Memory Skip policy (log scale)

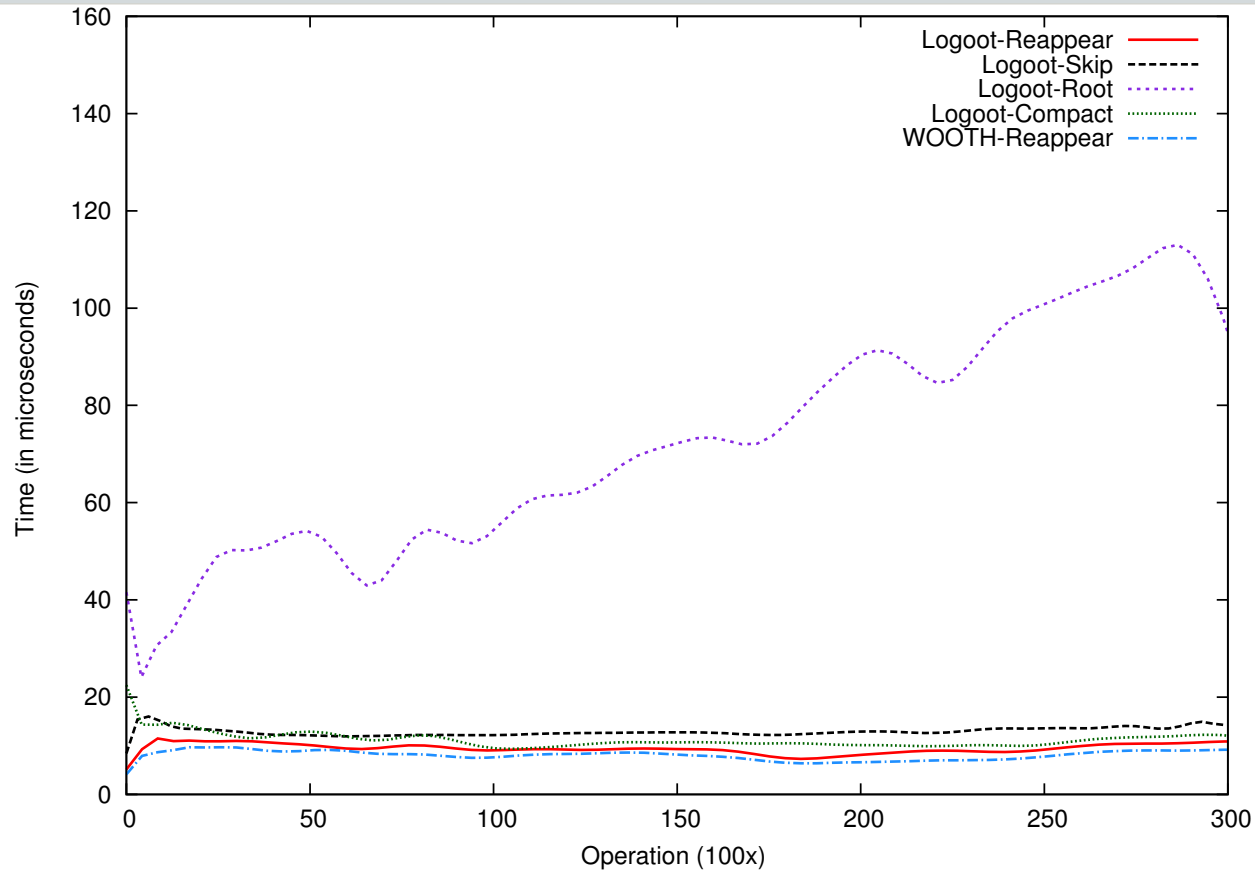


Experiments : Local execution times policies



- All policies (except root) are below 120 μ s

Experiments : Remote execution times policies



- **All policies (except root) are below 20 μ s**

Related Work



- Replicated data types
 - Ad-hoc, OT, CRDT, ...
 - We use them as replication layer
- Generic replication frameworks
 - Bayou [Terry et al 95], IceCube [Kermarrec et al 01]
 - Do not scale well since centralized
- Collaborative applications
 - DVCS (git, darcs, hg, ...), replicated file systems (Fuse,
 - Some conflicts have to be resolved by the users

Conclusion



- Benefit of our approach
 - Behavior modularity
 - More complex data types (XML, ...)
 - “Conflict outside the data” : separate awareness from consistency
- Performances
 - Very stable
 - Comparable to ad-hoc approaches

Future Work



- *Experiments on large-scale real data*
 - *Open-sources DVCS traces*
- *User file-system (FUSE)*
- Incremental DTD correction layers
- Effective awareness support
- Formal proof of equivalence between incremental and stateless policies