Agenda

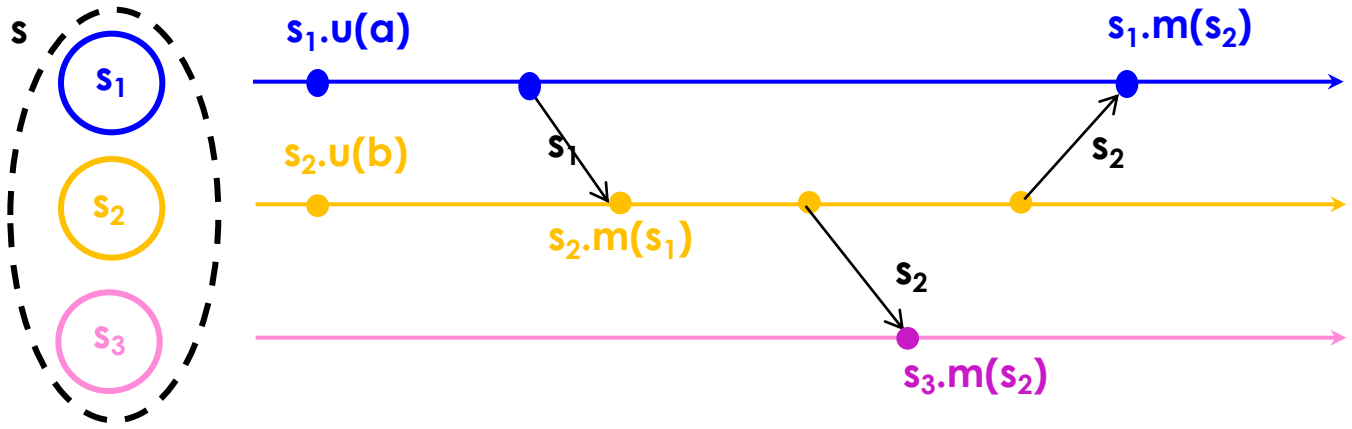
- Optimistic replication
  - CVS, Subversion
  - Duplicated databases
  - Operational transformation
  - Conflict-free Replicated Data Types (CRDT)

Conflict-free Replicated Data Types (CRDT) (*)

- Design operations to be commutative by construction
- Abstract data types
  - Designed to be replicated at multiple sites
  - Any replica can be modified without coordination
  - State convergence is guaranteed
- State-based and operation-based approaches

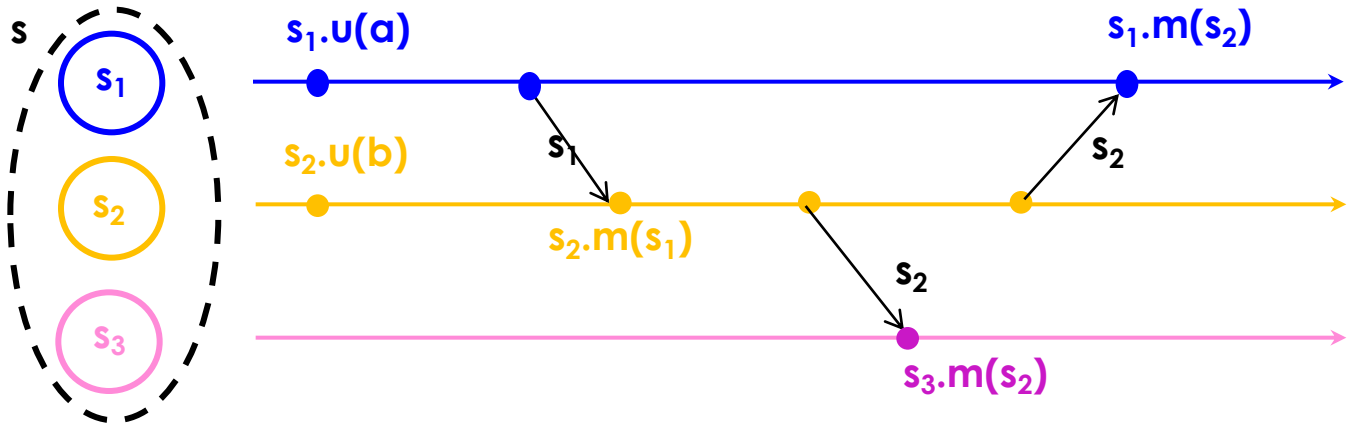
(*) M. Shapiro, N. Preguica, C. Baquero, M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types, Research Report, RR-7506, INRIA, 2011

Conflict-free Replicated Data Types
State-based Replication



- Replicated object: a tuple (S, s_0, q, u, m)
  - S: state domain
  - Replica at process p_i has state $s_i \in S$
  - s_0: initial state
- Each replica can execute one of following commands
  - q: query object's state
  - u: update object's state
  - m: merge state from a remote replica

Conflict-free Replicated Data Types
State-based Replication



- Algorithm
  - Periodically, replica at p_i sends its current state to p_j
  - Replica p_j merges received state into its local state by executing m
- After receiving all updates (irrespective of order), each replica will have same state

Conflict-free Replicated Data Types
Semi-lattice

- Partial order ≤ set S with a least upper bound (LUB), denoted ⊔
  - m = x ⊔ y is a LUB of { x, y } under ≤ if and only if
    ∀ m′, x ≤ m′ ∧ y ≤ m′ ⇒ x ≤ m ∧ y ≤ m ∧ m ≤ m′

- It follows that ⊔ is:
  - commutative: x ⊔ y = y ⊔ x
  - idempotent: x ⊔ x = x
  - associative: (x ⊔ y) ⊔ z = x ⊔ (y ⊔ z)

Conflict-free Replicated Data Types
Semi-lattice – Example on integers

- Partial order ≤ on set of integers
- Least upper bound ⊔ : max (maximum function)

- Therefore, we have:
  - commutative: max(x, y) = max(y, x)
  - idempotent: max(x, x) = x
  - associative: max(max(x, y), z) = max(x, max(y, z))

Conflict-free Replicated Data Types
Semi-lattice – Example on sets

- Partial order $\subseteq$ on sets
- Least upper bound $\sqcup$: $\cup$ (set union)

- Therefore, we have:
  - commutative: $A \cup B = B \cup A$
  - idempotent: $A \cup A = A$
  - associative: $(A \cup B) \cup C = A \cup (B \cup C)$

Conflict-free Replicated Data Types
Monotonic Semi-lattice Object

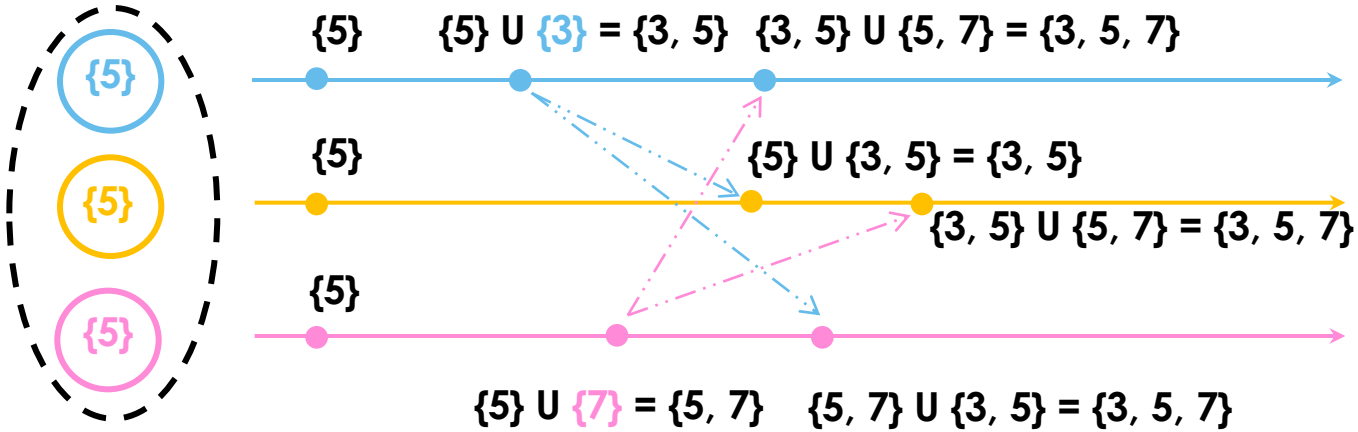
- A state-based object with partial order ≤, noted (S,≤, s_0, q, u, m), that has the following properties, is called a monotonic semi-lattice:
  1. Set S of values forms a semi-lattice ordered by ≤
  2. Merging state s with remote state s′ computes the LUB of the two states, i.e., s • m(s′) = s ⊔ s′
     (delivery order is not important)
  3. State is monotonically non-decreasing across updates, i.e., s ≤ s • u
     (updates have effect, no rollback)

Conflict-free Replicated Data Types
Convergent Replicated Data Type (CvRDT)

- Theorem: Assuming eventual delivery and termination, any replicated state-based object that satisfies the monotonic semi-lattice property is SEC

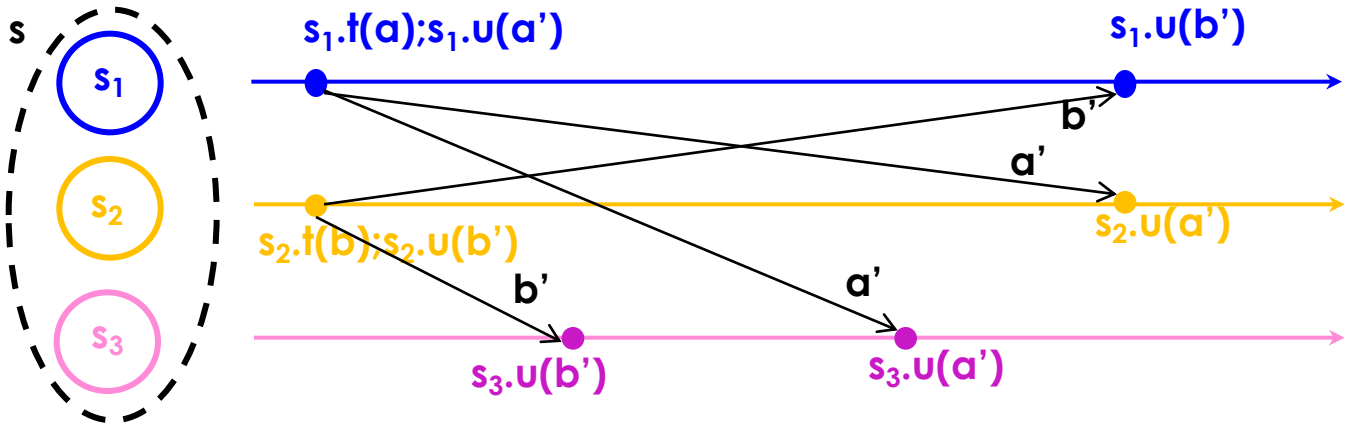
- Since:
  - Merge is both commutative and associative
    - We do not care about order
  - Merge is idempotent
    - We do not care about delivering more than once

Convergent Replicated Data Types
Example



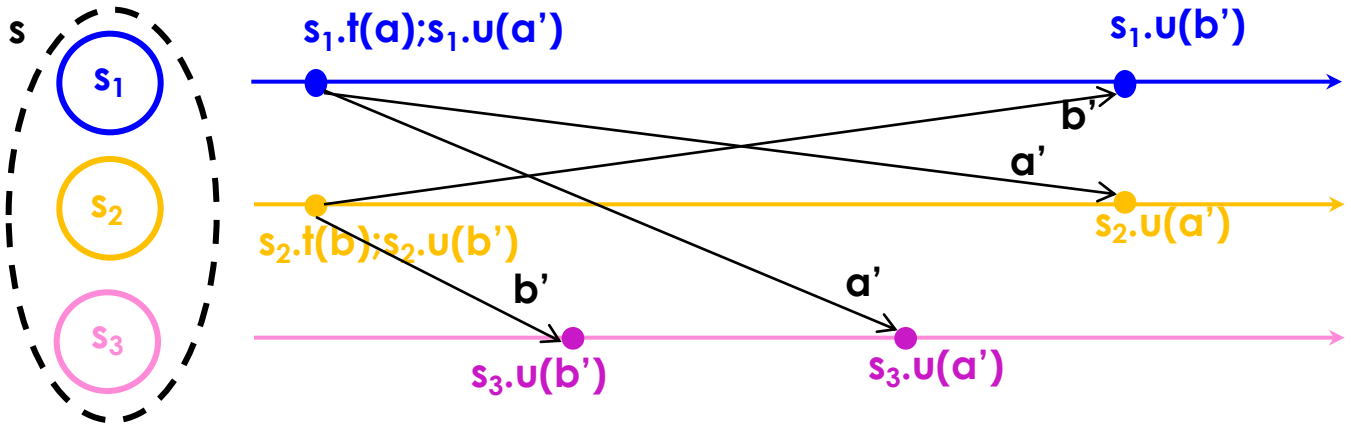
- Each replica can execute one of following commands
  - query q: returns entire set
  - update u: adds new element (e, α)to local set
  - merge m: compute unions between local set and remote set

Conflict-free Replicated Data Types
Operation-based Replication



- Replicated object: a tuple (S, s_0, q, t, u, P).
  - S: state domain
  - Replica at process p_i has state $s_i \in S$
  - s_0: initial state
- Each replica can execute one of following commands
  - q: query object's state
  - t: side-effect-free prepare-update method (at local replica)
  - u: effect-free update method (at all replicas)
  - P: delivery precondition

Conflict-free Replicated Data Types
Operation-based Replication



- Algorithm
  - Updates are delivered to all replicas
  - Use causally-ordered broadcast communication protocol, i.e., deliver every message to every node exactly once, w.r.t. happen-before order
  - Happen-before: updates from some replica are delivered to all recipients in the order they happened (effectively delivery precondition, P; delivery of u at a replica may be delayed until P is true)
  - Note: concurrent updates can be delivered in any order

Conflict-free Replicated Data Types
Commutativity Property

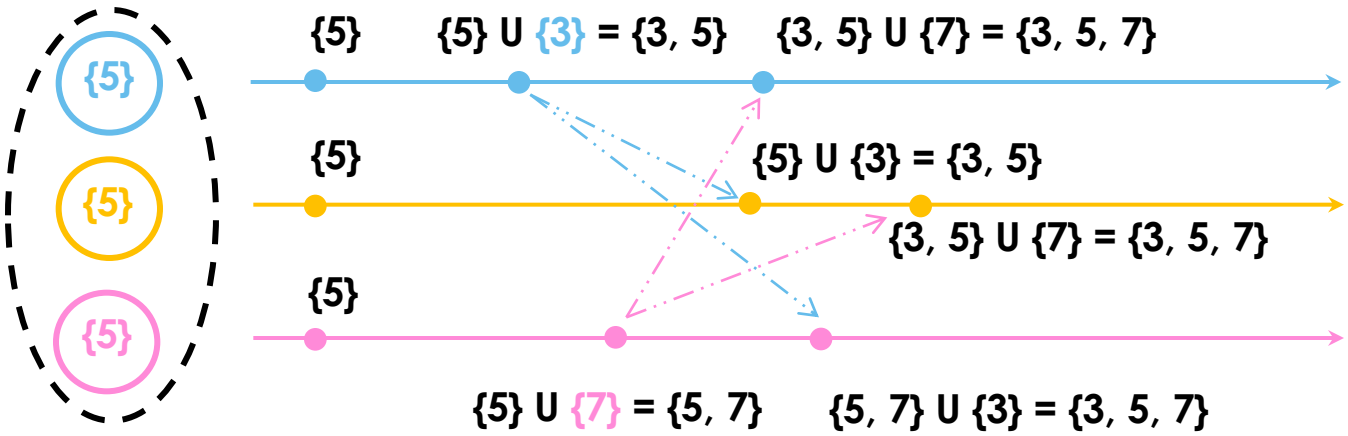
- Updates (t, u) and (t′, u′) commute, if and only if for any reachable replica state s where both u and u′ are enabled:
  - u (resp. u′) remains enabled in state s • u′ (resp. s • u)
  - s • u • u′ ≡ s • u′ • u

- Commutativity holds for concurrent updates

Conflict-free Replicated Data Types
Commutative Replicated Data Type (CmRDT)

- **Theorem:** Assuming causal delivery of updates and method termination, any replicated op-based object that satisfies the commutativity property for all concurrent updates is SEC

Commutative Replicated Data Types
Example



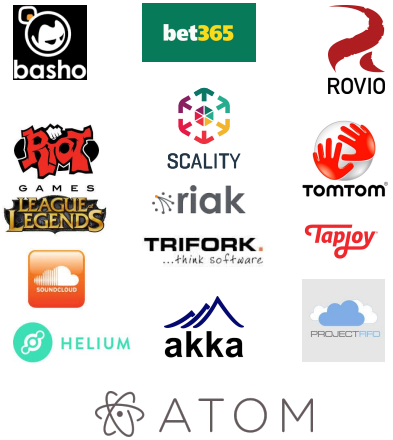
- query q: returns entire set
- prepare method t: adds new element (e,α) to local set
- update u: add delta to any remote replica

Consistency Maintenance

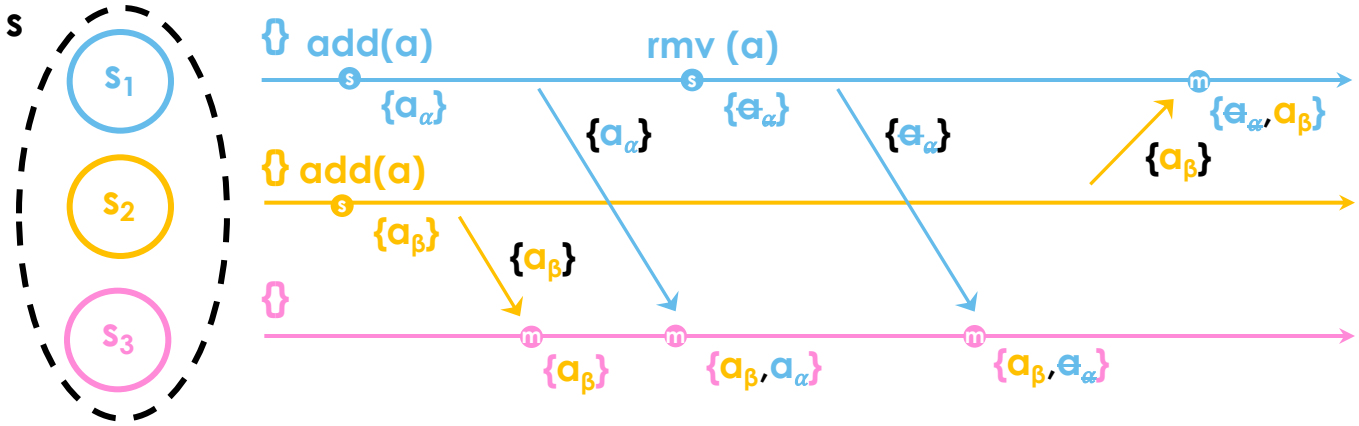
Conflict-free Replicated Data Types (CRDT)

- Register
  - Last-Writer Wins
  - Multi-Value
- Set
  - Grow-Only
  - 2-Phase
  - **Observed-Remove**
  - Observed-Update-Remove

- Map
- **Counter**
- Graph
  - Directed
  - Monotonic DAG
  - Edit graph
- **Sequence**

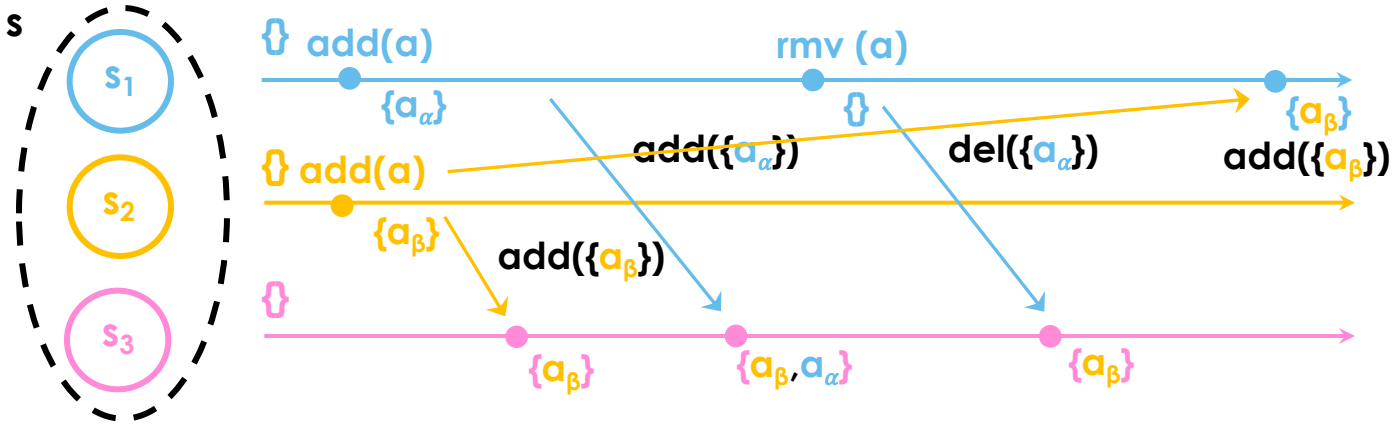


Conflict-free Replicated Data Types
Observed-Remove Set (CvRDT)



- Payload: (A,R) - added/removed sets of *(element, unique-token)*
- Operations*:*
  add(e): A := A U *{(e,α)}*
  remove(e): R := R U *{(e,-)∈A}* remove all unique elements observed
  lookup(e): ∃*(e,-)∈A\R*
  merge(S, S'): *(A U A', R U R')*
- *{true} add(e)* ‖ *remove(e) {e∈S}*

Conflict-free Replicated Data Types
Observed-Remove Set (CmRDT)



- Payload:
  S = {(e,α), (e,β), (e,γ), …} where α, β, γ,… are unique tokens
- Operations:
  add(e): $S := S \cup \{(e,\alpha)\}$ where α is a fresh unique token
  lookup(e): $\exists\alpha: (e,\alpha) \in S$
  remove(e): $R := \{(e,\alpha) \mid \exists\alpha\ (e,\alpha) \in S\}$ (at source) no tombstones
  $S := S \setminus R$
  {true} add(e) ‖ remove(e) {e ∈ S}

Conflict-free Replicated Data Types
P-Counter (CvRDT)

- Payload:
  - P = [int, int, …]
- Operations:
  - value(): $\sum_i P[i]$
  - increment(): P[MyID]++
  - merge(S,S'): $S \sqcup S' = [\ldots,max(s.P[i],s'.P[i]),\ldots]_i$
- Positive

Conflict-free Replicated Data Types
PN-Counter (CvRDT)

- Payload:
  - P = [int, int, …],
    N = [int, int, …]
- Operations:
  - value(): $\sum_i P[i] - \sum_i N[i]$
  - increment(): P[MyID]++
  - decrement(): N[MyID]++
  - merge(S,S'): $S \sqcup S' = ([\ldots,max(s.P[i],s'.P[i]),\ldots]_i,$
    $[\ldots,max(s.N[i],s'.N[i]),\ldots]_i)$
- Positive or negative

Conflict-free Replicated Data Types (CRDT) CvRDT vs. CmRDT

- Both approaches are equivalent
  - A state-based object can emulate an operation-based object, and vice-versa
- **Operation-based:**
  - More efficient since you only ship small updates
  - But require exactly once causally-ordered broadcast
- **State-based:**
  - Only require reliable broadcast
  - Communication overhead of shipping the whole state
- **Delta State-based**:
  - Small messages
  - Dissemination over unreliable communication channels

Conflict-free replicated data type (CRDT) (Text) Sequence

- Document = linear sequence of elements
- Unique position identifiers
  - Each element has a unique identifier
  - Identifier remains constant for the lifetime of the document
  - Dense total order of identifiers consistent with element order:
    $\forall\ id_x , id_y : id_x < id_y \Rightarrow \exists\ id_z : id_x < id_z < id_y$
- Real numbers require an infinite precision
- Different approaches for generating identifiers: Treedoc, Logoot

Logoot (*)

- Identifier:

<p_1,s_1,h_1>. <p_2,s_2,h_2> ··· <p_k,s_k,h_k>

p_i integer

s_i site identifier

h_i logical clock at site s_i

<0,NA,NA>

<87,1,0>

<87,1,0>.<111,6,7>

<89,4,5>

<MAX,NA,NA>

CSCW conference

March 19-23, 2011

Hangzhou,China

(*) Stéphane Weiss, Pascal Urso and Pascal Molli. *Logoot : a Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks*. In ICDCS , Montreal, Quebec, Canada , June 2009

Logoot

<0,NA,NA>

<1,1,1>.<5,2,4>.<4,6,2>

<4,2,1>.<2,4,6>.<3,3,6>

<6,3,4>

<MAX,NA,NA>

CSCW conference

March 19-23, 2011

Hangzhou,China

- Site 9 wants to insert 5 lines between first 2 lines
- Take first components <1,1,1> and <4,2,1>
  - 3 identifiers: <1,9,h>, <2,9,h> and <3,9,h>
- Take first 2 components <1,1,1>.<5,2,4> and <4,2,1>.<2,4,6>
  - 26 identifiers
    - **<1,1,1>.<{5-9},9,h>**
    - **<2,9,h>.<{1-9},9,h>**
    - **<3,9,h>.<{1-9},9,h>**
    - **<4,2,1>.<{1},9,h>**

Choose 5 identifiers

Logoot

<0,NA,NA>

<1,1,1>.<5,2,4>.<4,6,2>

<4,2,1>.<2,4,6>.<3,3,6>

<6,3,4>

<MAX,NA,NA>

CSCW conference

March 19-23, 2011

Hangzhou,China

- Remote insertion: Insert(<2,9,1>.<2,9,2>, “Computer Supported Cooperative Work”)
- Use binary search algorithm

<0,NA,NA>

<1,1,1>.<5,2,4>.<4,6,2>

<2,9,1>.<2,9,2>

<4,2,1>.<2,4,6>.<3,3,6>

<6,3,4>

<MAX,NA,NA>

CSCW conference

Computer Supported Cooperative Work

March 19-23, 2011

Hangzhou,China

Logoot

<0,NA,NA>

<1,1,1>.<5,2,4>.<4,6,2>

<2,9,1>.<2,9,2>

<4,2,1>.<2,4,6>.<3,3,6>

<6,3,4>

<MAX,NA,NA>

CSCW conference

Computer Supported Cooperative Work

March 19-23, 2011

Hangzhou,China

- Remote deletion: Delete(<4,2,1>.<2,4,6>.<3,3,6>)
- Use binary search algorithm

<0,NA,NA>

<1,1,1>.<5,2,4>.<4,6,2>

<2,9,1>.<2,9,2>

<6,3,4>

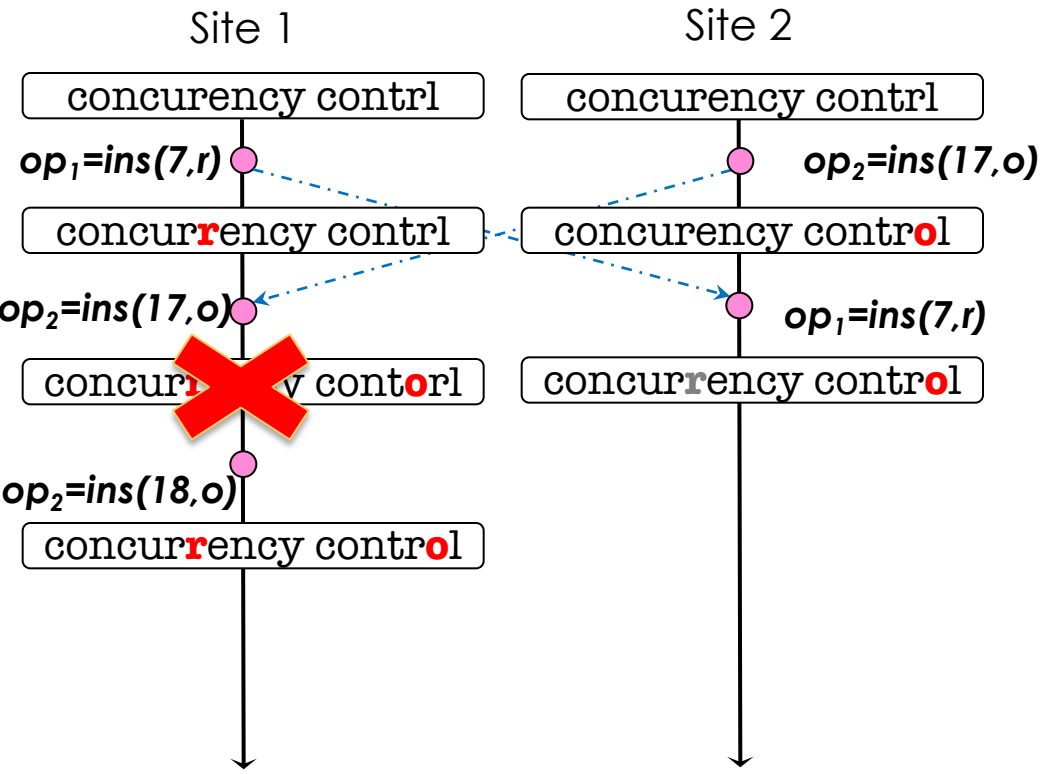
<MAX,NA,NA>

CSCW conference

Computer Supported Cooperative Work

Hangzhou,China

Operational Transformation (OT)

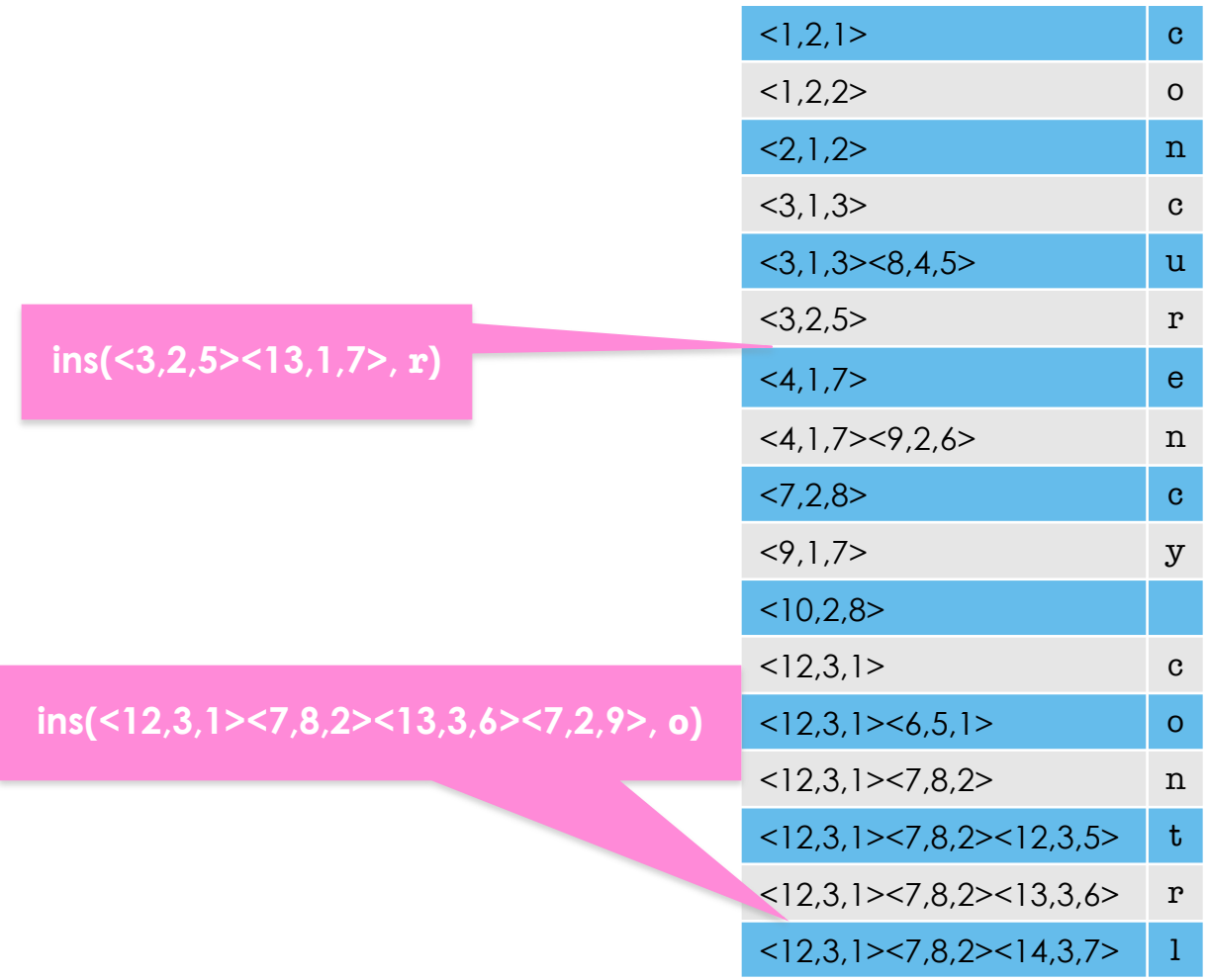


- Transforms non commuting operations to make them commute
- Genericity
- Time complexity

  Average: O(H.c) H: #ops

  Worst case: $O(H^2)$ c: avg. #conc. ops
- Difficult to write correct transformation functions
- State vectors used for detecting concurrency ⇒ scalability limitations
- **Not very suitable for large scale peer-to-peer collaboration**

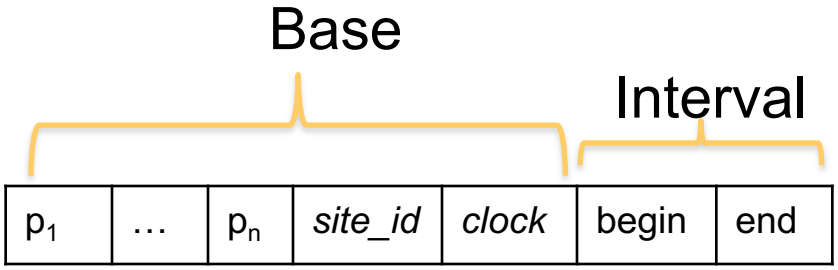
Conflict-free Replicated Data Types (CRDT)

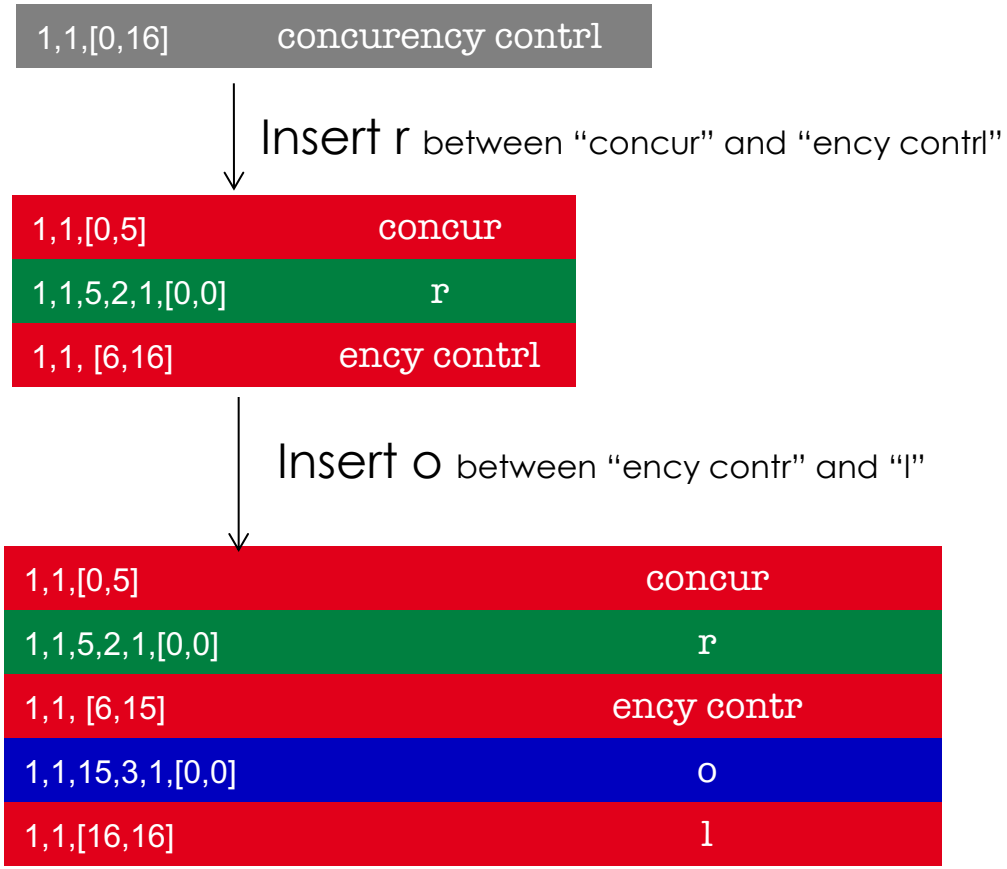


- Time complexity
  - Average: O(k.log(n))
  - Worst case: O(H.log(H))
  - H: #ops
  - n: doc. size (non deleted chars.)
  - k: avg. size of Logoot identifier
- No need for concurrency detection
- Identifiers storage cost
- New design for each data type
- **Suitable for large-scale collaboration**

Conflict-free Replicated Data Types (CRDT) LogootSplit

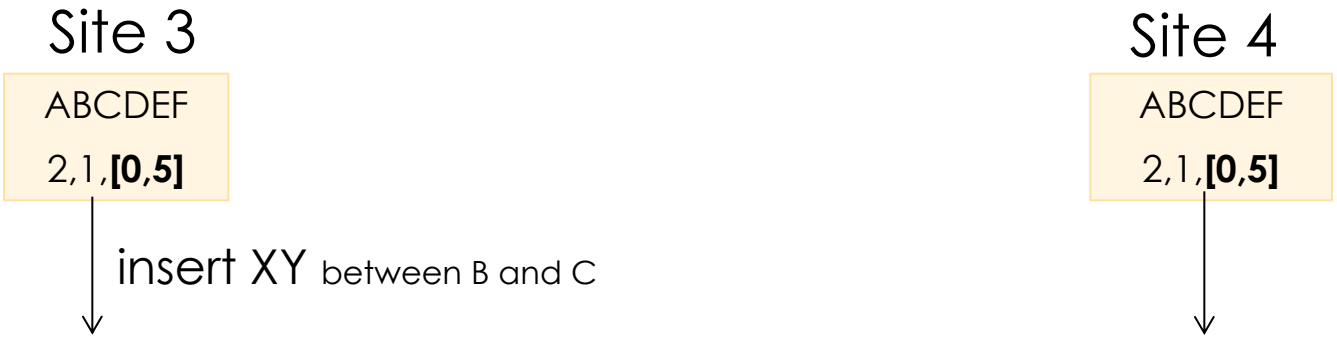
LogootSplit identifiers



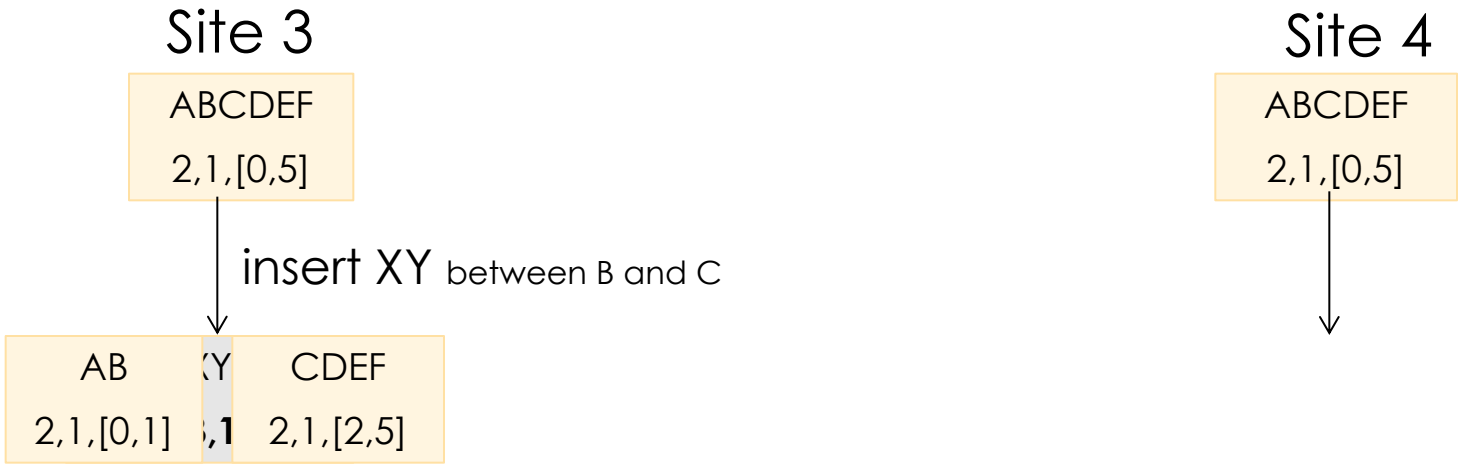


André, L. et al., "Supporting Adaptable Granularity of Changes for Massive Scale Collaborative Editing" CollaborateCom 2013

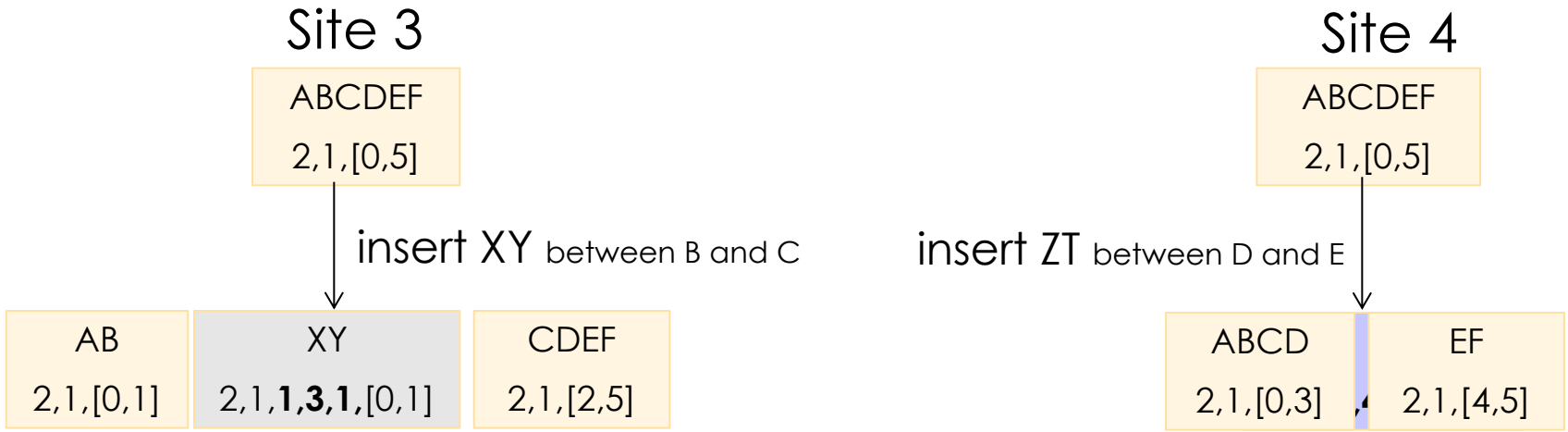
LogootSplit



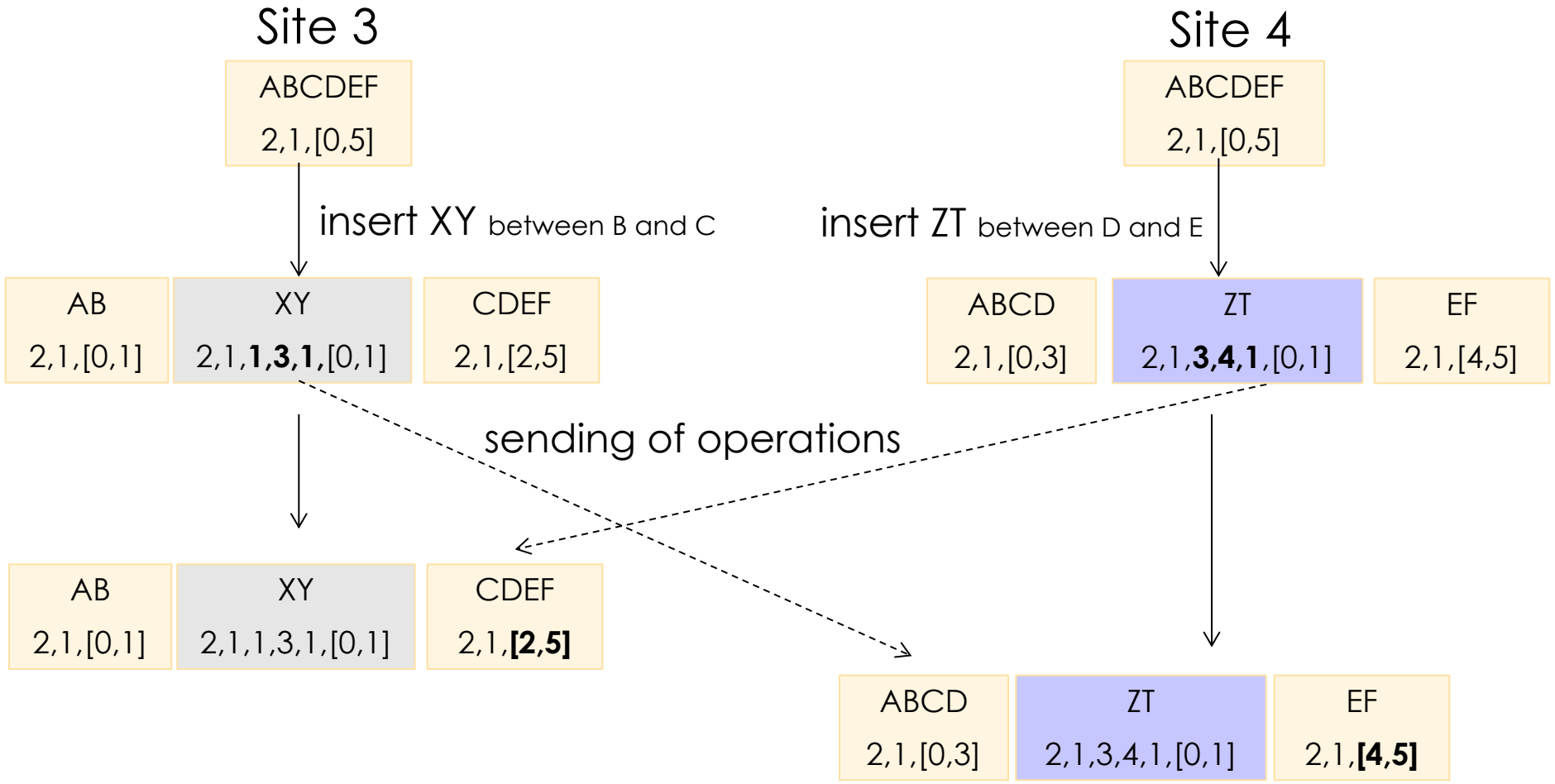
LogootSplit



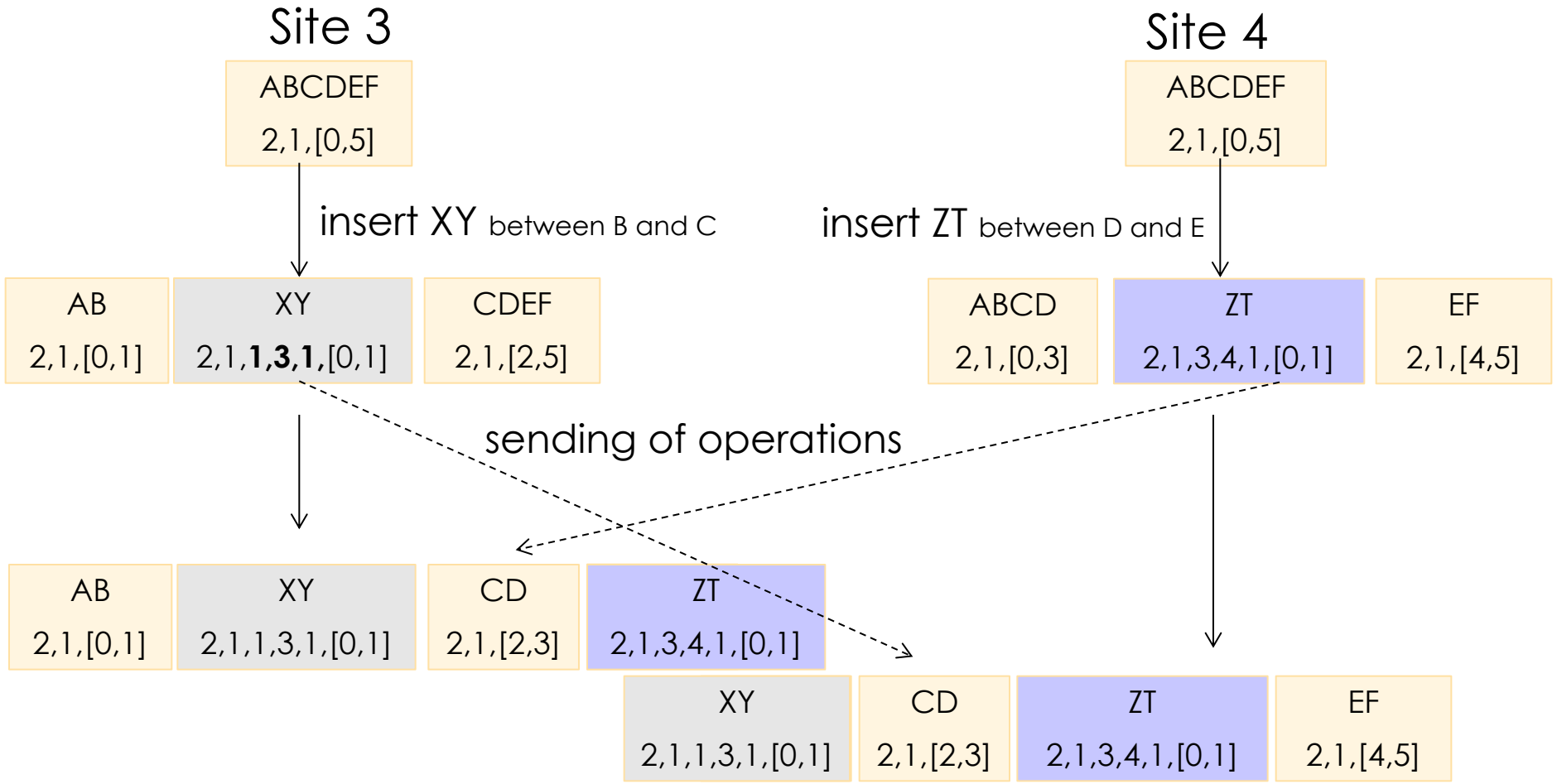
LogootSplit



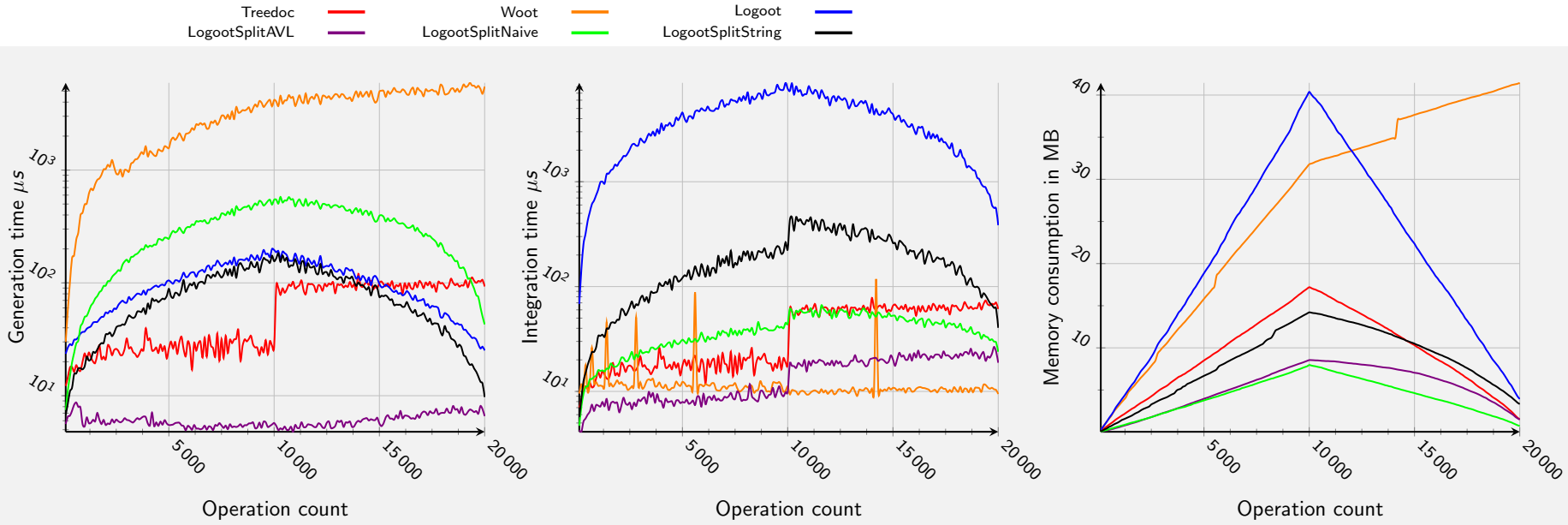
LogootSplit



LogootSplit



LogootSplit Performance Comparison



Generation Time **Integration Time** **Memory Consumption**

Random block (avg. 50 char.) insertion/deletion

First 80% insertions, then 20% insertions

Delays in MUTE

Delays in GoogleDocs