Replicated Abstract Data Types

Kick-off meeting of ConcoRDanT project

Hyun-Gul Roh

17, Nov, 2010

Until now,

Commutative Replicated Data Types

have been introduced

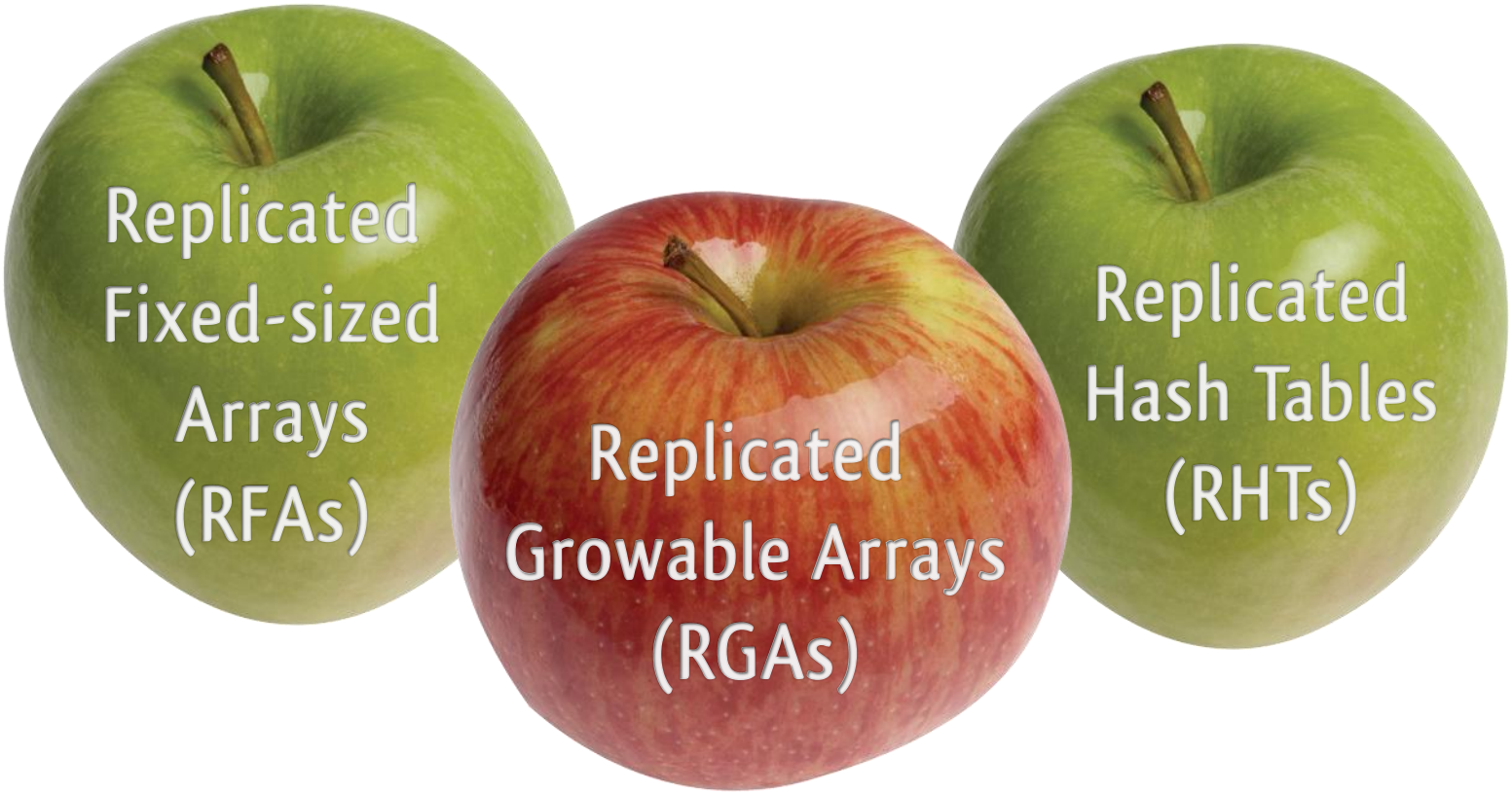
Since 2006, I proposed

Replicated Abstract Data Types

Replicated Abstract Data Types (RADTs)

= Replicated Data Structures

+ Optimistic Operations

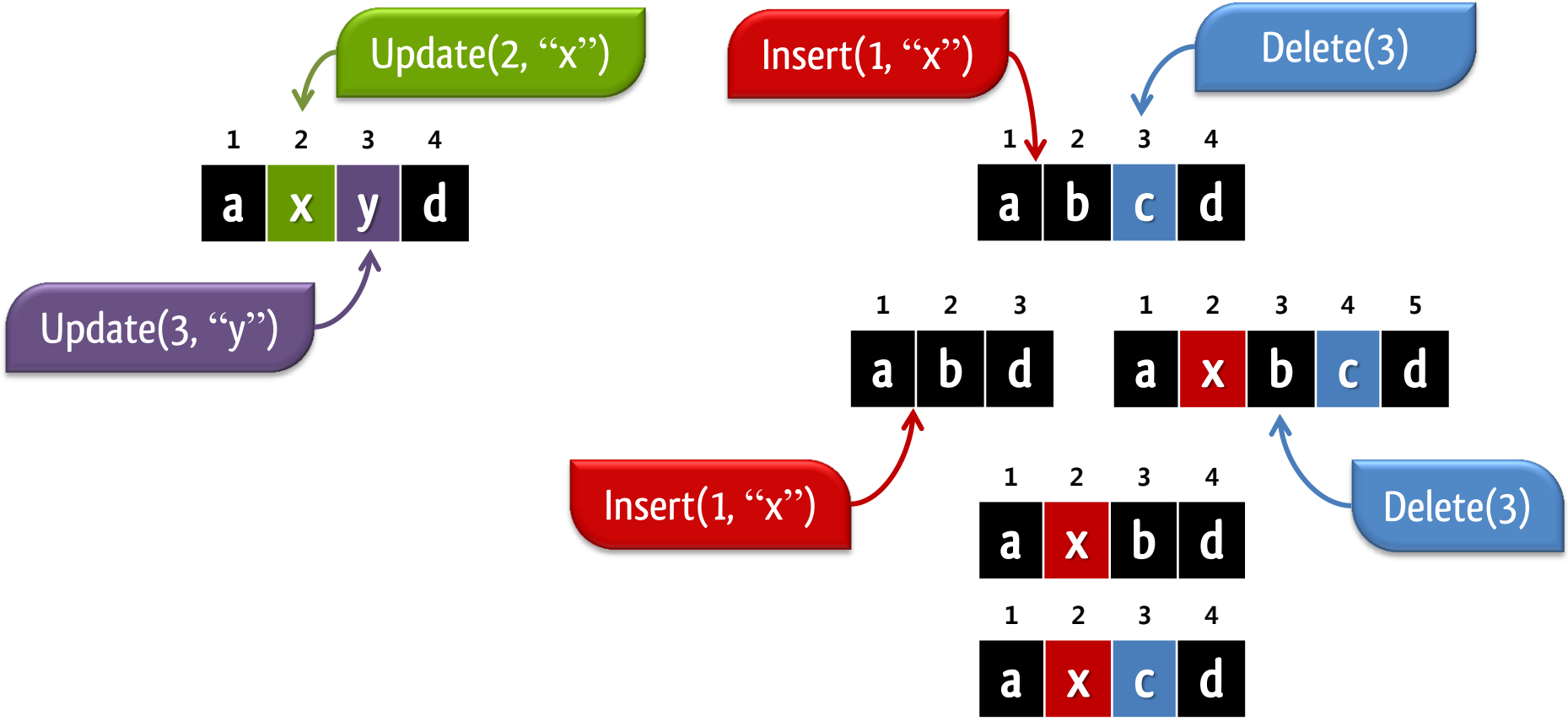


Replicated Growable Arrays (RGAs)

= Replicated Ordered Objects

+ Optimistic {Insert, Delete, Update}

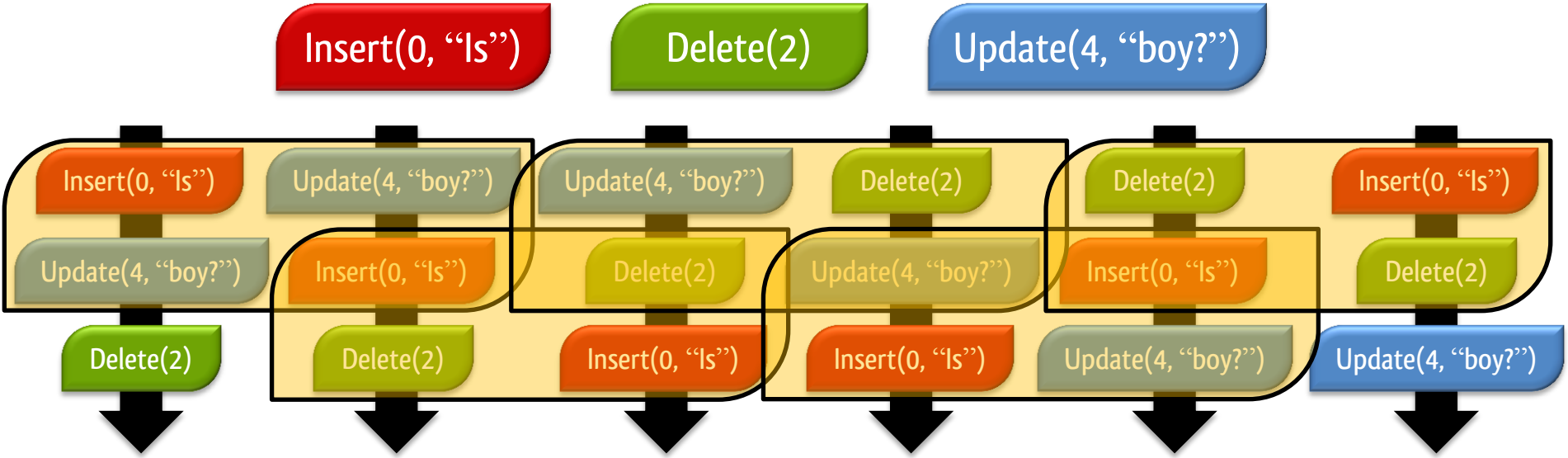
Concurrent Insert/Delete more likely affect (interfere) each other



Concurrent Inserts/Deletes distort the intentions of other operations

Operation Commutativity

The condition that **every pair** of **concurrent** operations are **commutative!!**

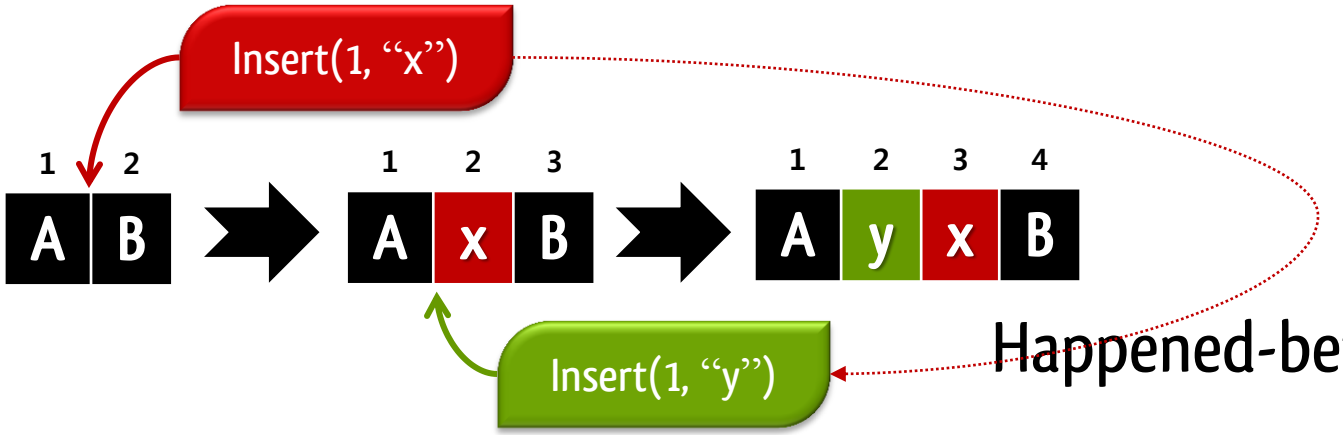


"It is **not** difficult to make **a pair** of concurrent operations commute"

"**But**, it is difficult to make **every pair** of concurrent **Insert/Delete** operations commute"

RGAs for Operation Commutativity and Intention

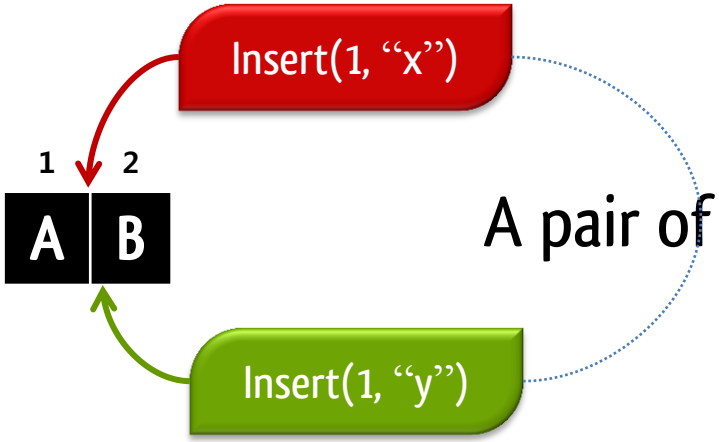
Precedence = whose intention has higher priority?



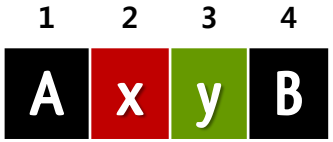
The same intentions:

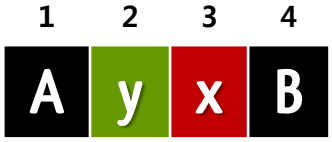
"Insert a new object **next to** the 1st object"

precedence of Insert(1, "x") < precedence of Insert(1, "y")



A pair of concurrent operations





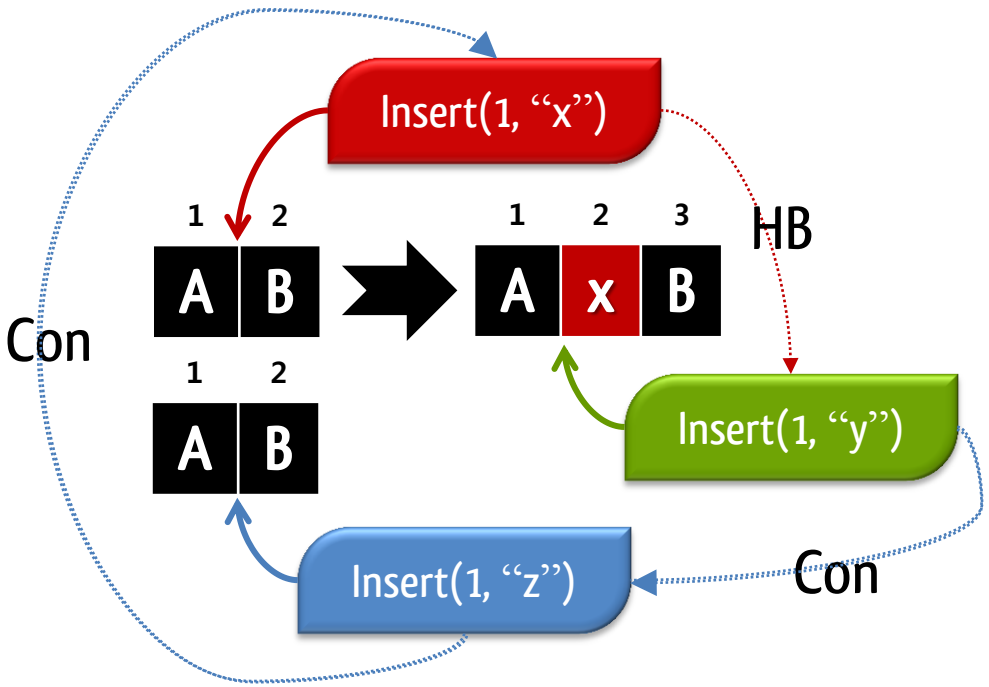
Precedence between a pair of concurrent operations

Insert(1, “y”) < Insert(1, “x”)

Insert(1, “x”) < Insert(1, “y”)

Define one precedence for consistency!!!

Precedence among multiple pairs of concurrent operations?



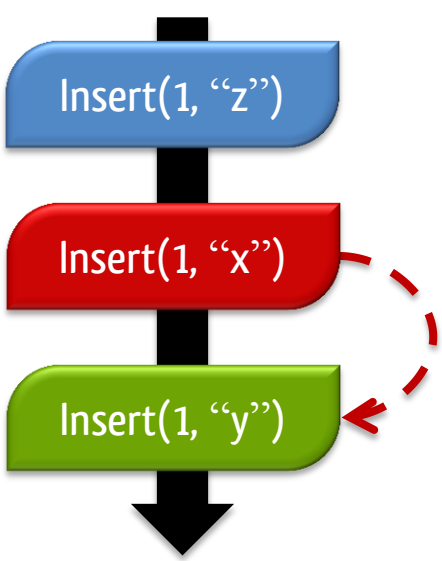
Happened-before operations

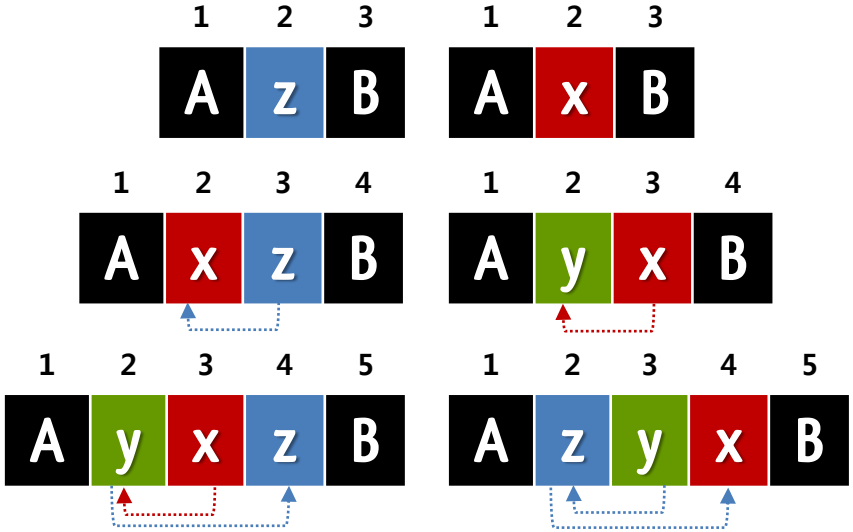
Insert(1, "x") < Insert(1, "y")

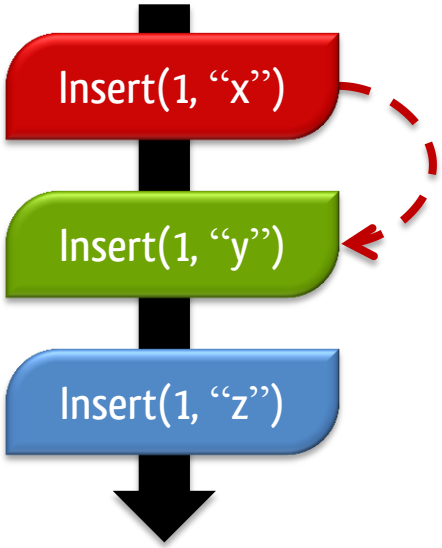
Concurrent operations

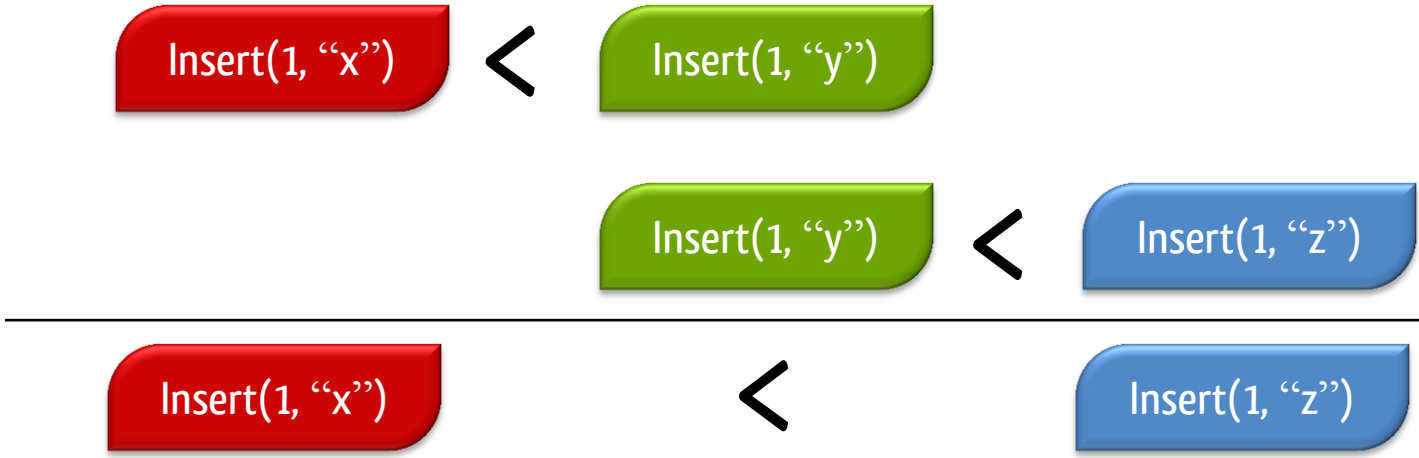
Insert(1, "y") < Insert(1, "z")

Insert(1, "z") < Insert(1, "x")









Precedence Transitivity (PT)

Recall

Allow **every** site to execute operations in a **different** order (Not a total order)

➔ Operation Commutativity

Make **a pair** of concurrent operations **commute** ➔ Precedence

➔ Make **every pair** of concurrent operations **commute**

➔ Precedence Transitivity

Significance of Precedence Transitivity (PT)

Present a solution to achieve operation commutativity

without

History of operations,

Deriving total order of objects or Totally ordered dense indexing scheme

Operation Commutativity

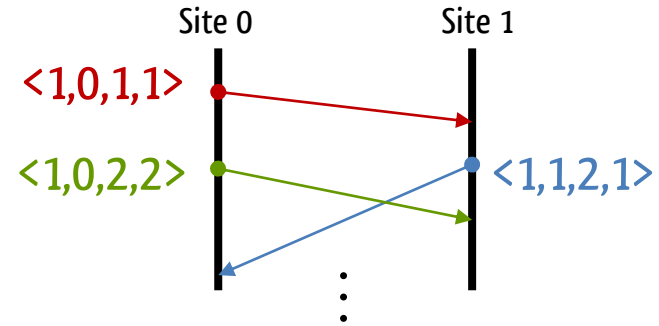
➔ A principle **only** for **concurrent** operations

Precedence Transitivity

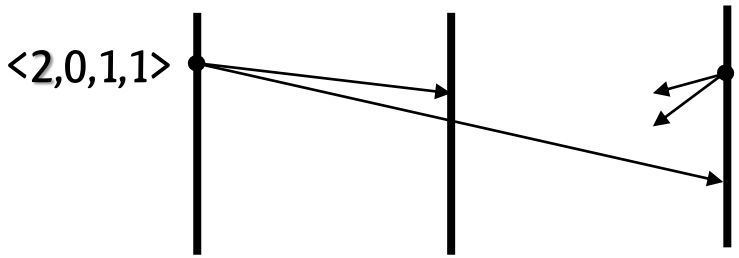
➔ A principle for the **relationship** among **happened-before** and **concurrent** operations

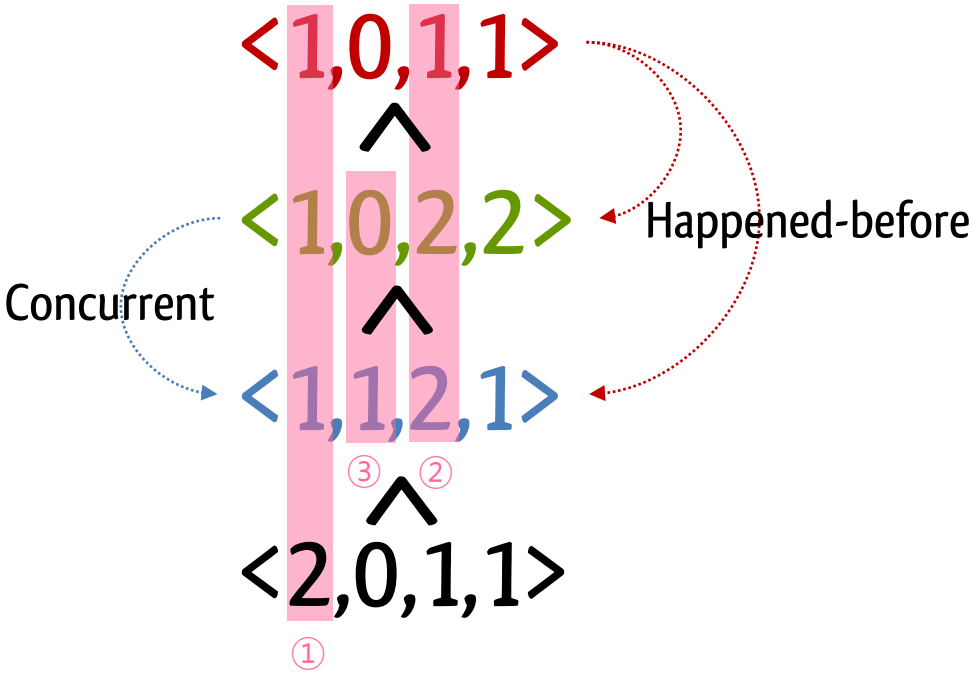
S4Vector <int ssn, int sid, int sum, int seq>

ssn: session number
sid: site ID
sum: sum of a version vector
seq: reserved for purging tombstones



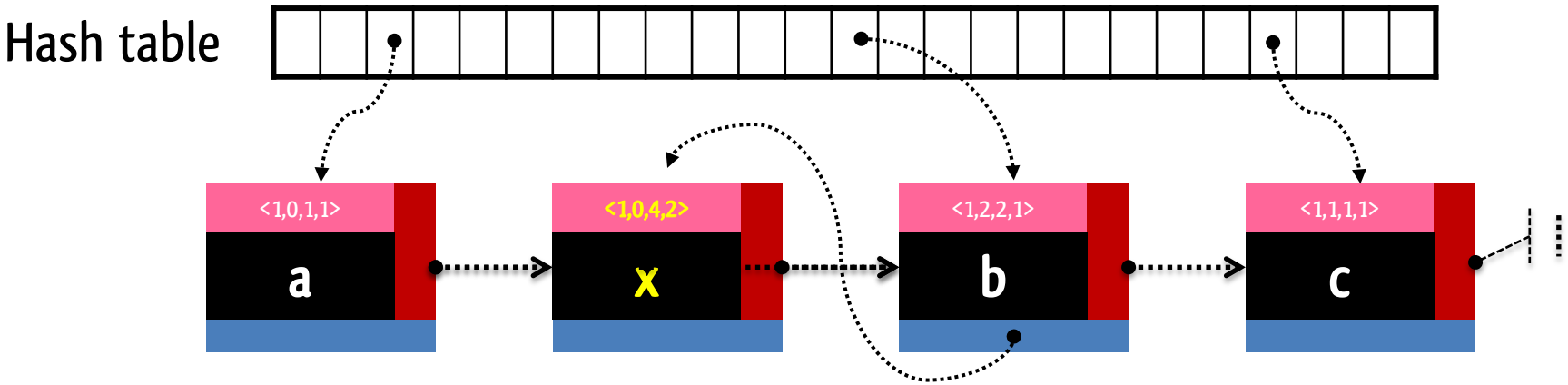
A new session begins
with the same initial data structures





S4Vector Index (SVI) scheme

Adopt a **linked list** with a **hash table**



Insert(1, "x") with <1,0,4,2>

Local operation

Insert(<1,0,1,1>, "x")

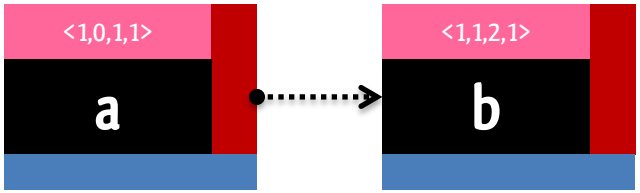
Remote operation

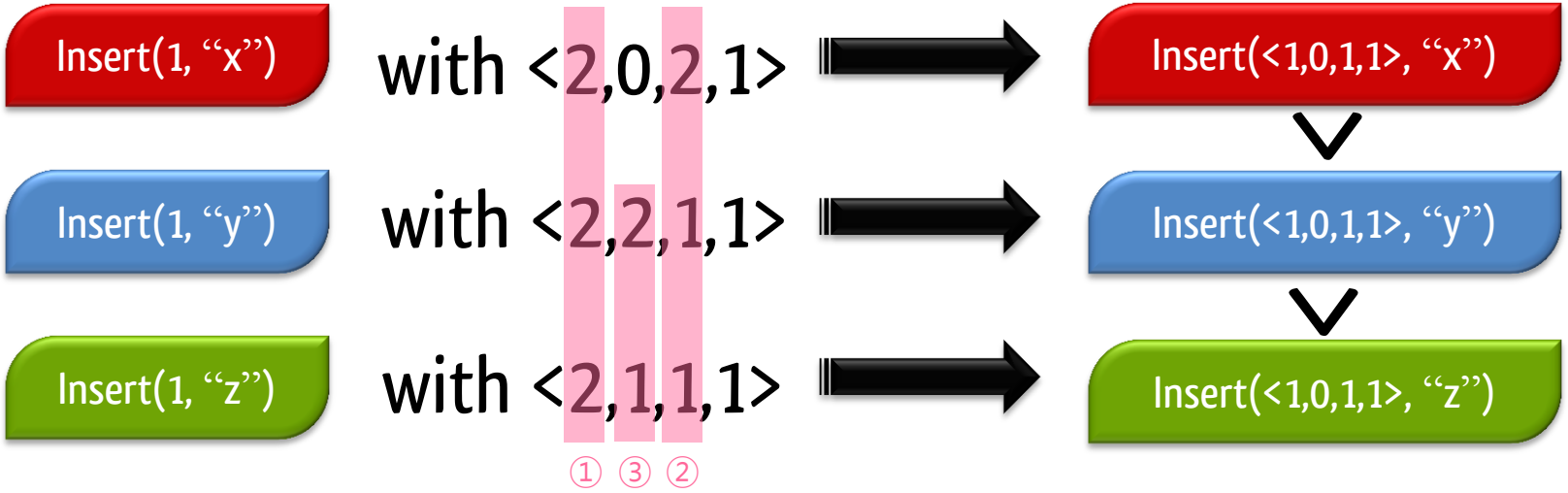
Preserve **intentions**

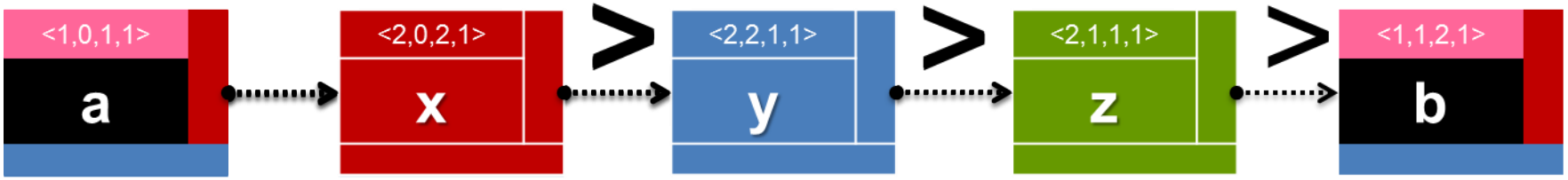
Boost **performance**

of **remote** operations

Concurrent Inserts







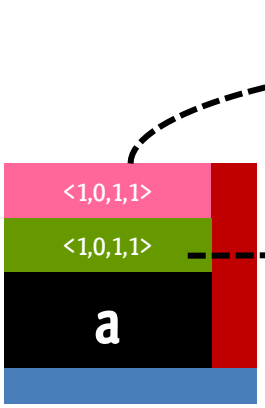
Summary of RGA implementation

A Delete makes a tombstone

Concurrent Inserts

Concurrent Updates

follow transitivity of S4Vectors



Immutable after an Insert

Mutable by Update/Delete

A Delete always wins An update

Overwhelming performance of RGAs

Algorithms	Local operations	Remote operations
RGAs	O(N) or †O(1)	O(1)
ABT	O(|H|)	O(|H|²)
SDT	O(1)	O(|H|²) or §O(|H|³)
TTF	O(N) or ‡O(1)	O(|H|²+N)
WOOT	#O(N²) and ¶O(1)	#O(N³) and ¶O(N)
Treedoc	O(logN)	O(logN)

N: the number of objects or characters,
|H|: the number of operations in history buffer,
†: local pointer operations,
‡: the caret operations,
§: worst-case complexity,
#: WOOT insertion operation,
¶: WOOT deletion operation.

Remote operations

Local operations with 1000 objects

120 µs

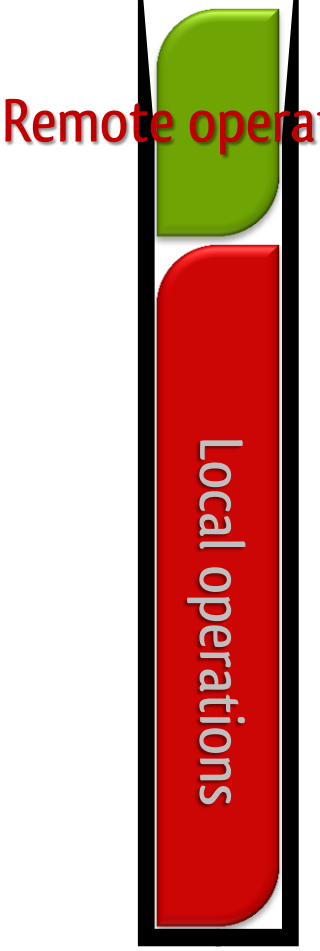
Experiments on Pentium 4 2.8Ghz

2 sites

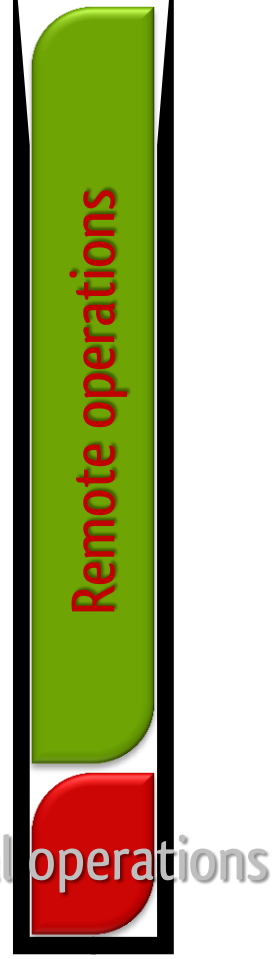
32 sites

Which performance is important

for Scalability?



O(1)



RGAs

Consistency of Insert / Delete

Intention preservation

Precedence Transitivity + SVI scheme

Good performance

Scalability