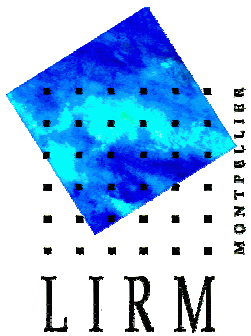
Asynchronous Reconciliation based on Operational Transformation for P2P Collaborative Environments

Michelle CART Jean FERRIÉ



Université Montpellier II

Montpellier (France)

LIRMM

The Context

Reconciling multiple distributed copies of an object

Synchronous methods:

- copies reconciled in real-time
- need broadcast protocol
- based on Operational Transformation (CSCW)
- need ordering mechanisms (state vectors,timestamps, sequencer, ..)

→ unsuitable for P2P environments

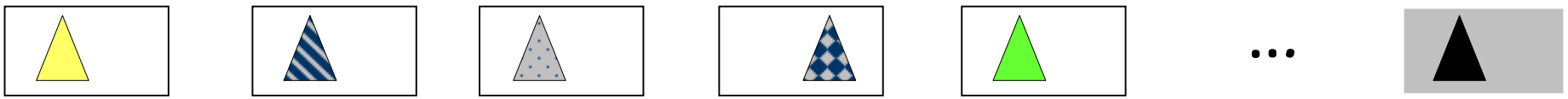
Asynchronous methods:

- pair-wise copy reconciliation
- need central component:
  - master copy (Copy/ Modify/ Merge paradigm)
  - particular site [Bayou, IceCube], sequencer [SO6]

→ unsuitable for P2P environments

Our Proposal

Multiple divergent copies of an object:



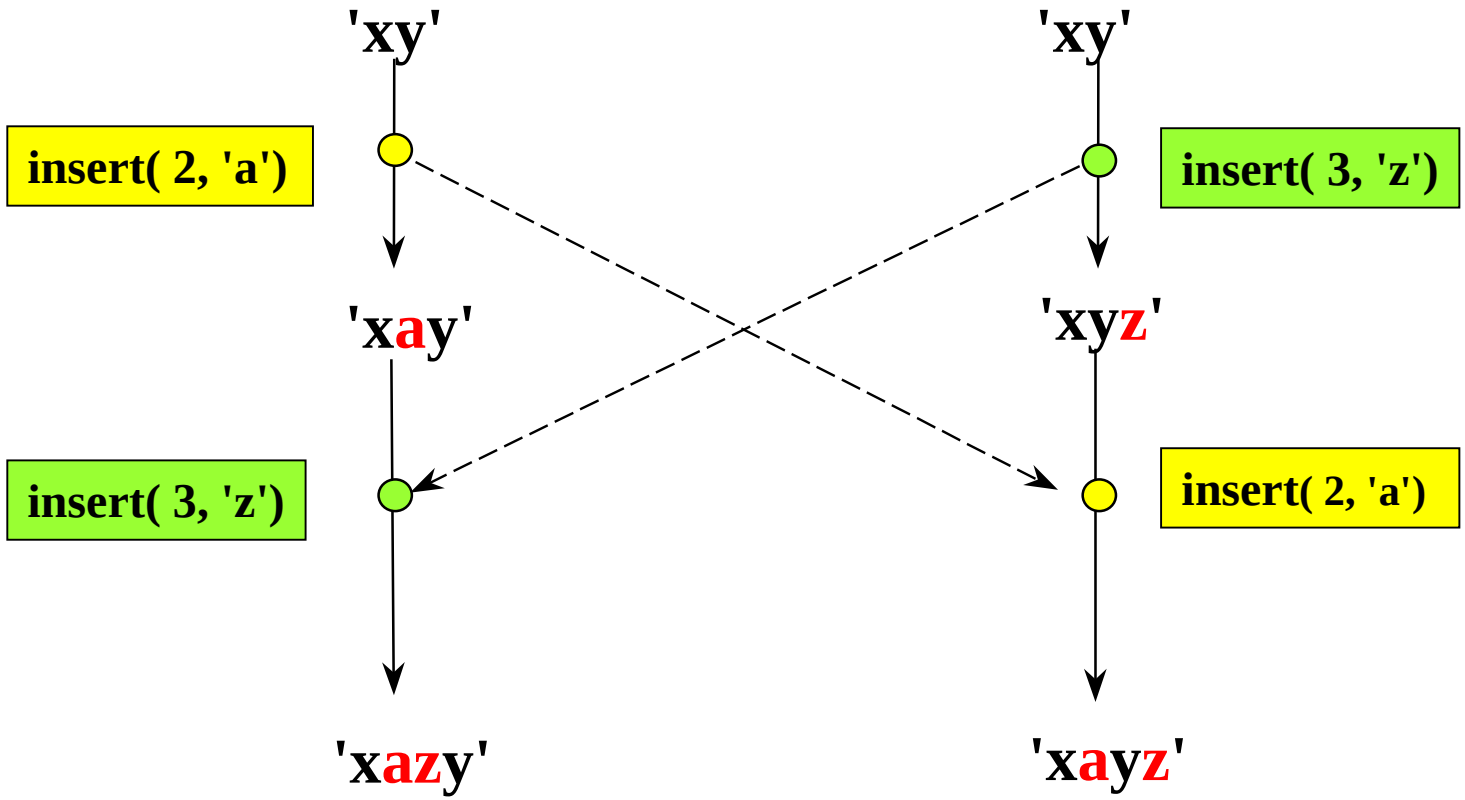
Reconciliation

- **ensuring "eventual consistency" of the copies**
- without rejecting any operation
- asynchronous (at any time)
- pair-wise (regardless of the pair, no privileged copy)
- group of an undetermined size
  (sites can dynamically join / leave the group)
- scalable to a P2P network
- **based on Operational Transformation**

LIRMM

Operational Transformation: recall

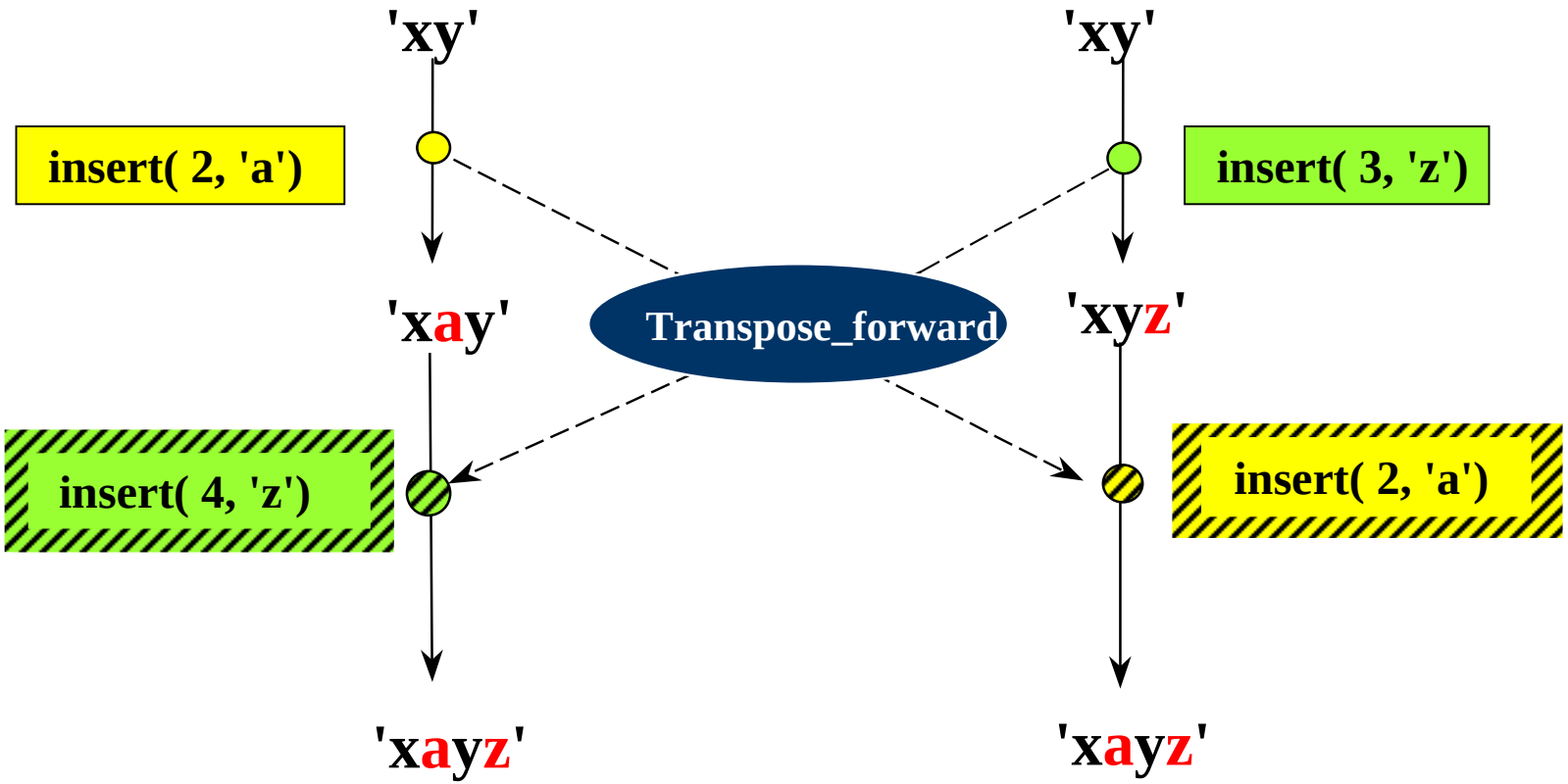
Used to achieve copy convergence when concurrent and non-commutative operations are executed on copies in different orders



LIRMM

Operational Transformation: recall

Used to achieve copy convergence when concurrent and non-commutative operations are executed on copies in different orders



Transformation functions specified (before execution)
Conflicting situations automatically solved (during execution)

LIRMM

Operational Transformation: example

Transpose_forward (insert(p_1, c_1) , insert(p_2, c_2)) =

case p_1 ? p_2 of

$p_1 < p_2$: return insert(p_2 +1, c_2);

$p_1 > p_2$: return insert(p_2, c_2);

$p_1 = p_2$: if $c_1 = c_2$ then return id

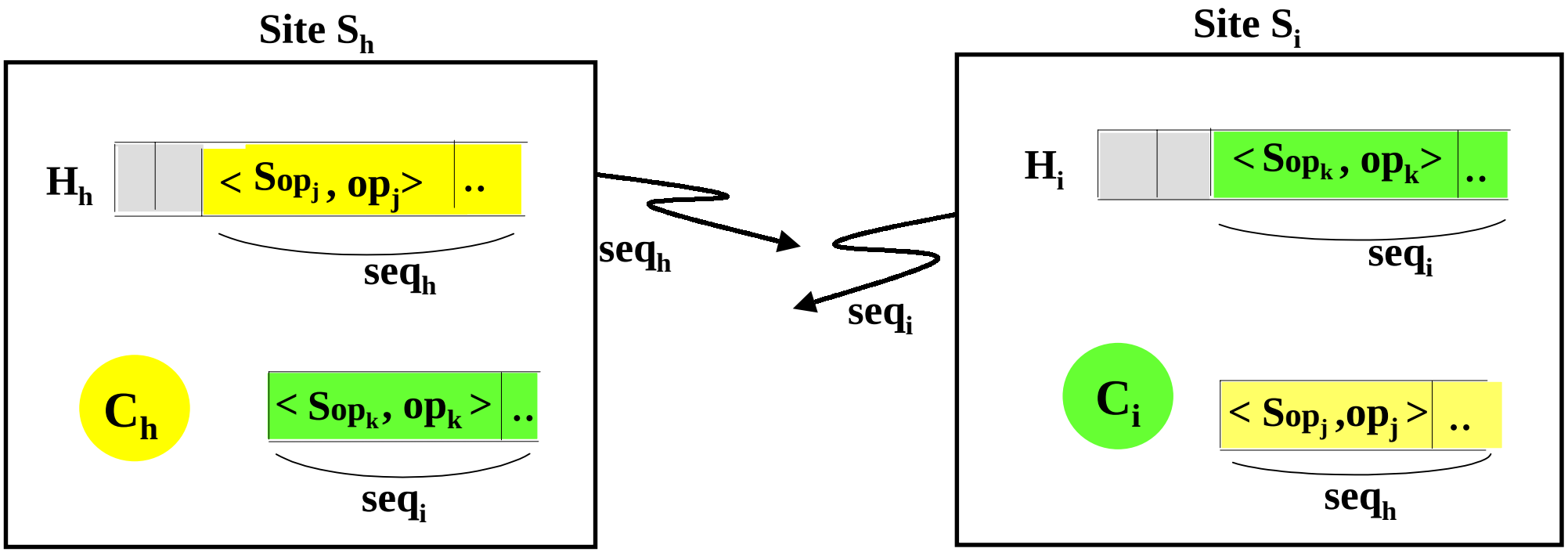
else

if pr(c_2) > pr(c_1)
then return insert(p_2, c_2)
else return insert(p_2+1, c_2);
endif;

endif;

endcase

The Model (1)



History H: succession of operations $< S_{op}, op >$ that leads from the initial state to the current state of the copy **C**.

S_{op}: generator site of op

op: operation executed on the copy C associated to H

Reconciliation transmission of sequences seq of operations: $seq_h \rightarrow S_i$, $seq_i \rightarrow S_h$

LIRMM

The Model (2)

Relationships between operations $\langle S_{op_j}, op_j \rangle$ and $\langle S_{op_k}, op_k \rangle$

Causal precedence "happen before"

$op_j \xrightarrow{c} op_k$ if

- $S_{op_j} = S_{op_k}$ and op_j generated before op_k
- $S_{op_j} \neq S_{op_k}$ and op_k generated after the execution of op_j on S_{op_k}
- $\exists\, op_m : (op_j \xrightarrow{c} op_m)$ and $(op_m \xrightarrow{c} op_k)$

Concurrency

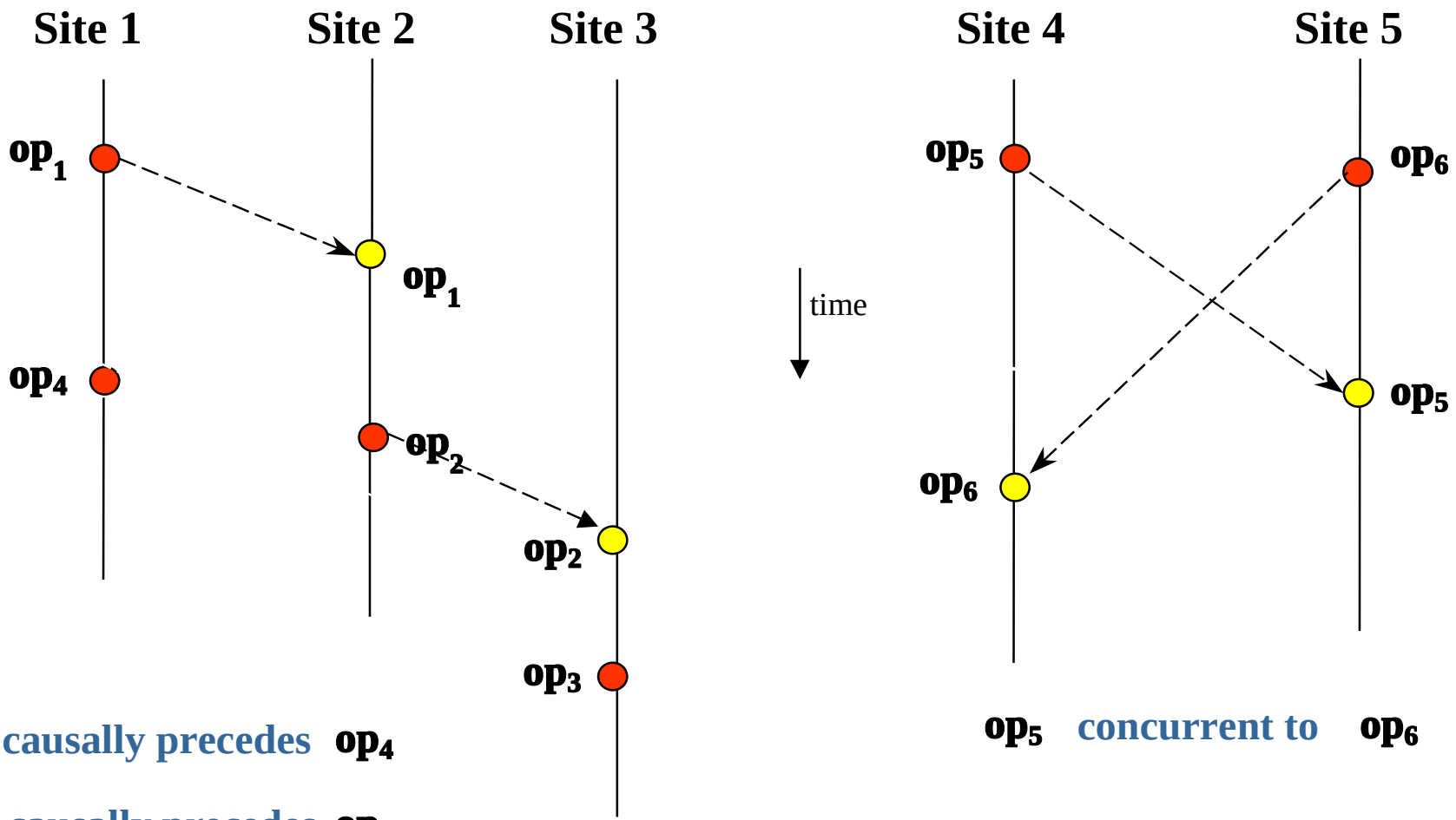
$op_j \;//\; op_k$ not $(op_j \xrightarrow{c} op_k)$ and not $(op_k \xrightarrow{c} op_j)$

Precedence

$op_j \xrightarrow{}_H op_k$ op_j appears before op_k in history H

LIRMM

The model (2): example



op_1 causally precedes op_4

op_1 causally precedes op_2

op_1 causally precedes op_3

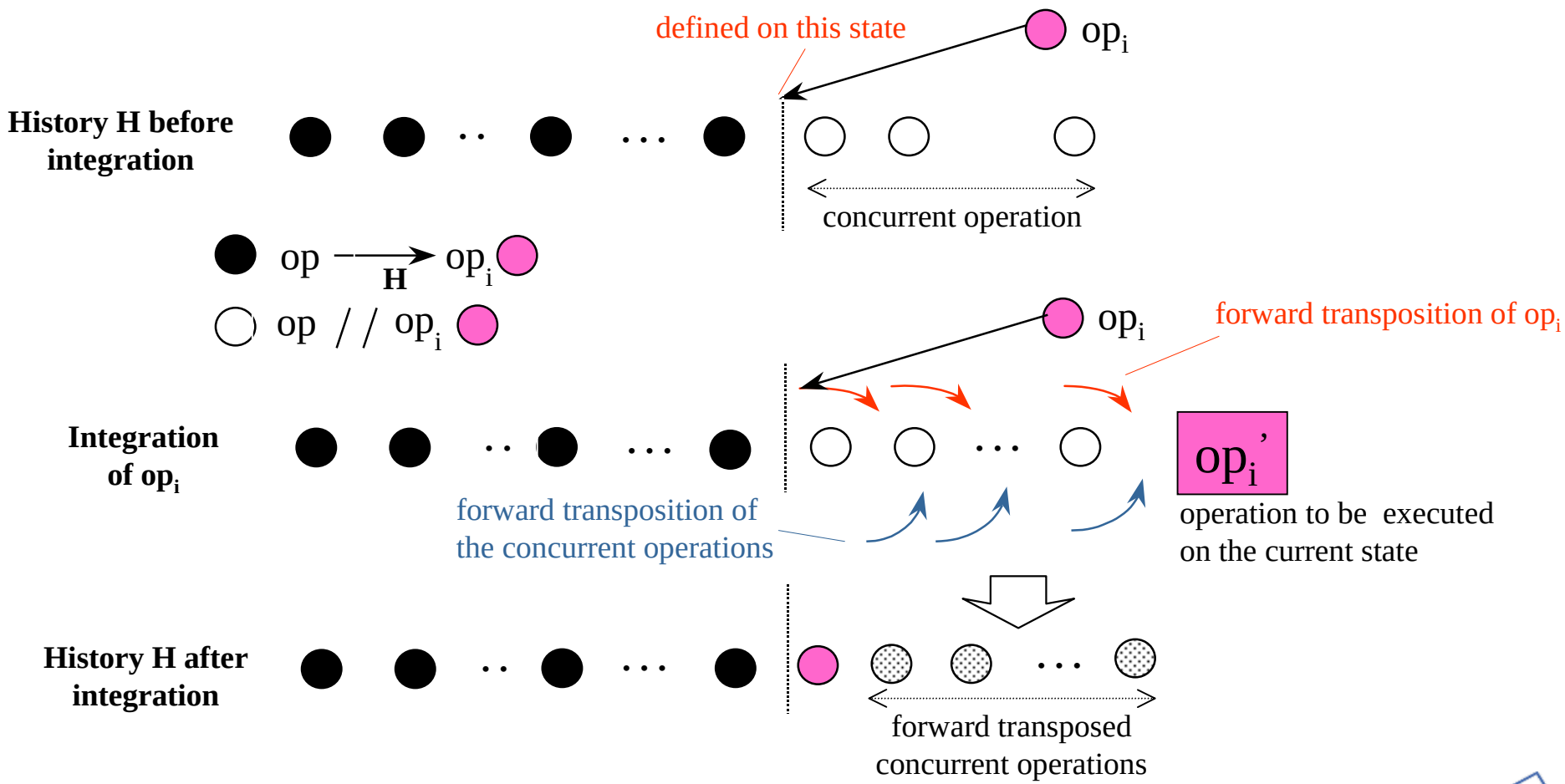
op_5 concurrent to op_6

local operation
remote operation

LIRMM

Integration of an operation

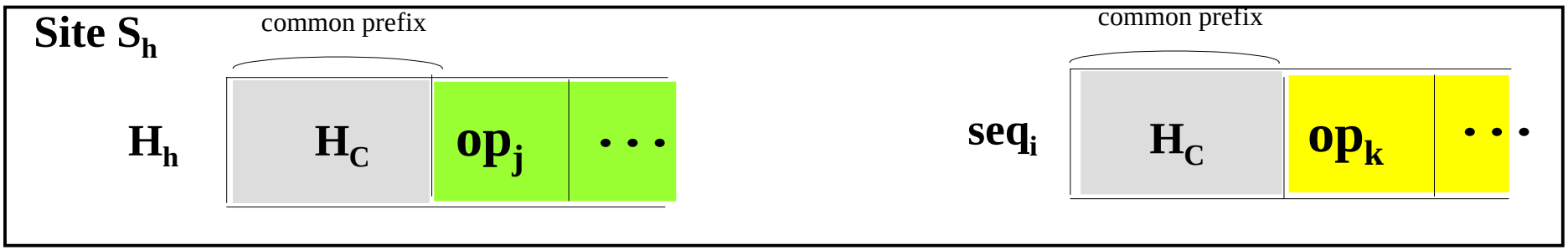
Integration of the operation op_i into the history H of a site



LIRMM

Basic Principle of MOT2

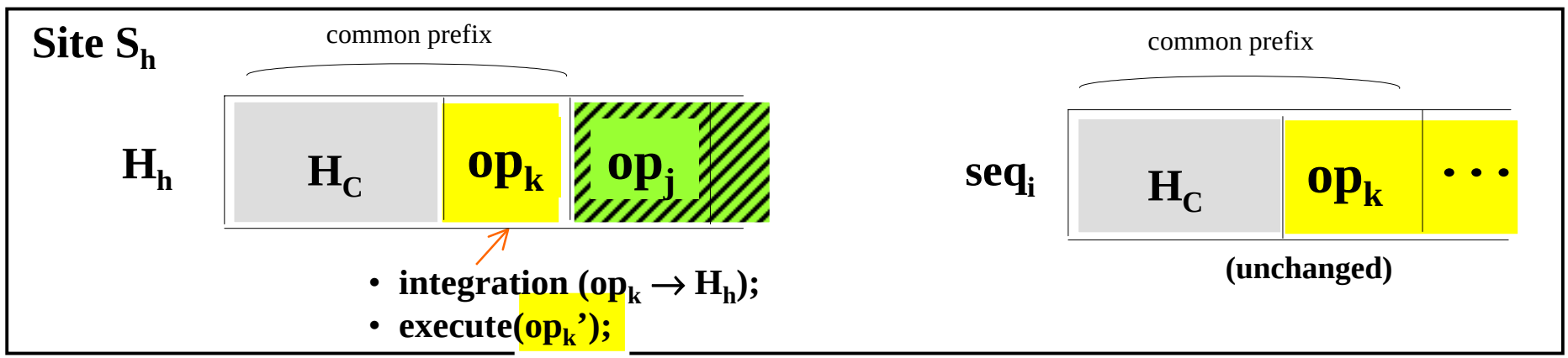
("Merge based on Operational Transformation")



The order between 2 operations to be merged depends on the order defined among the generator sites of these operations.

$$S_{op_k} < S_{op_j}$$

generator site of op_k — generator site of op_j



LIRMM

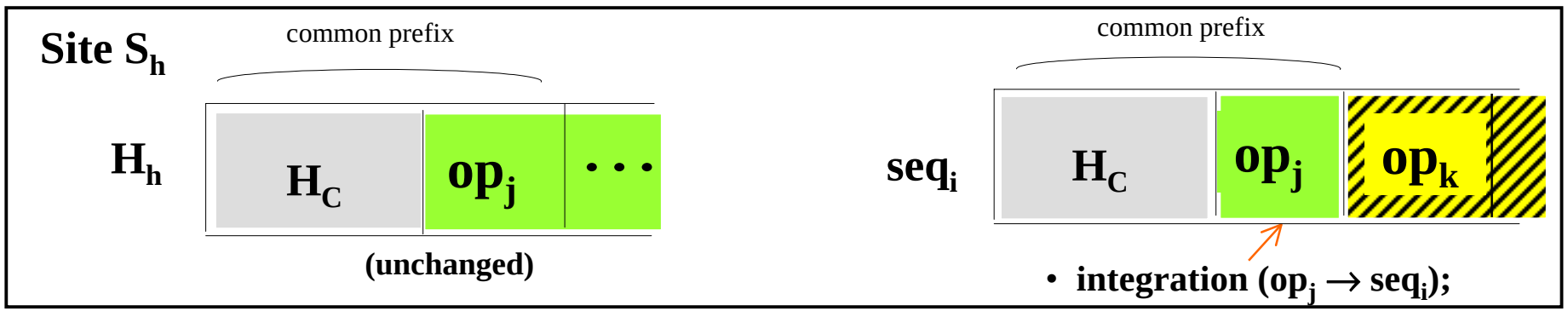
Basic Principle of MOT2

("Merge based on Operational Transformation")

The order between 2 operations to be merged depends on the order defined among the generator sites of these operations.

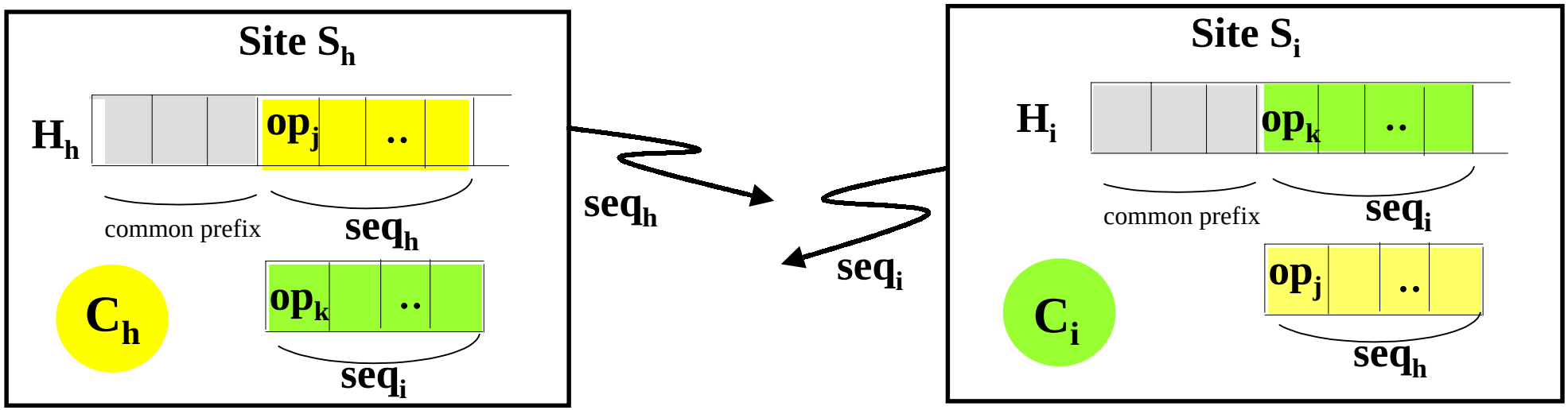
$$S_{op_j} < S_{op_k}$$

generator site of op_j — generator site of op_k



LIRMM

The MOT2 Merge Algorithm



while .... **loop**

case S_j ? S_k **of**

$S_k < S_j$: integration ($op_k \rightarrow H_h$) ; execute(op_k');

$S_j < S_k$: integration ($op_j \rightarrow seq_i$);

$S_k = S_j$: ; -- $op_j = op_k$

end loop ;

while .... **loop**

case S_j ? S_k **of**

$S_k < S_j$: integration ($op_k \rightarrow seq_h$);

$S_j < S_k$: integration ($op_j \rightarrow H_i$) ; execute(op_j');

$S_k = S_j$: ; -- $op_j = op_k$

end loop ;

Algorithm symmetrical and generic

LIRMM

The MOT2 Algorithm

procedure **MOT2** (H_k , H_j) ;
{Look for the prefix H_C common to H_k and H_j : détermine the index i_S of the last operation of H_C}
    i := i_S + 1 ;
while (i ≤ size_H_k) and (i ≤ size _H_j) loop
        < $Idop_k$, Sop_k , op_k > := H_k[i] ;
        < $Idop_j$, Sop_j , op_j > := H_j[i] ;
        case Sop_k ? Sop_j of
            Sop_k < Sop_j : **Integration** (H_j , i, < $Idop_k$, Sop_k , op_k >) ; {op_k integrated into H_j}
            Sop_j < Sop_k : **Integration** (H_k, i, < $Idop_j$, Sop_j , op_j >) ; {op_j integrated into H_k}
            Sop_k = Sop_j : ; {Operation already present in H_k and H_j}
        endcase ;
        i := i + 1 ;
endloop ;

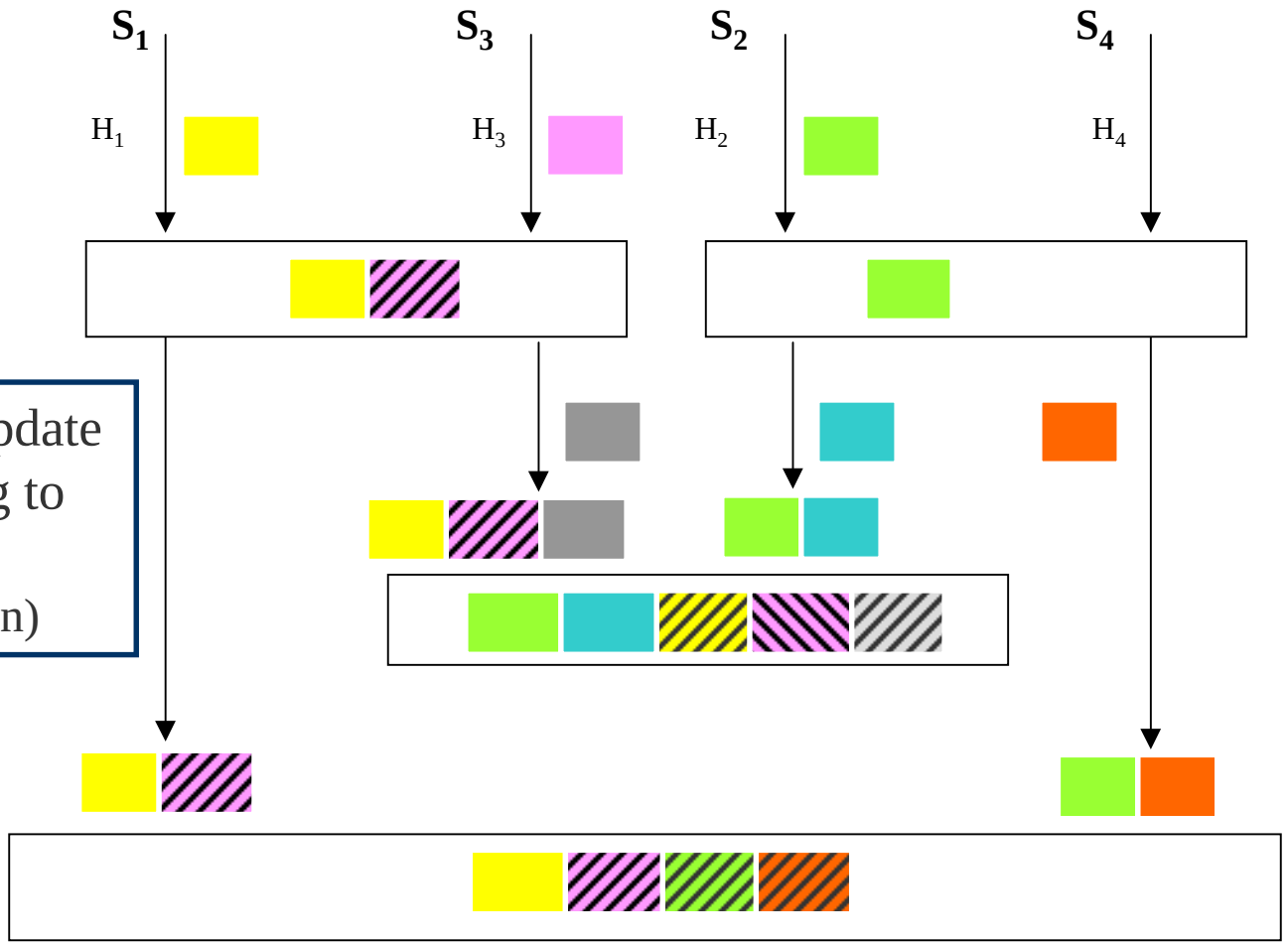
{The end of H_k or H_j has been reached}
while i ≤ size _H_j loop {End of H_k : append the remainder of H_j→ H_k}
        < $Idop_j$, Sop_j, op_j > := H_j[i] ;
        Append (H_k, < $Idop_j$, Sop_j , op_j >) ;
        i := i + 1 ;
endloop ;

while i ≤ size _H_k loop {End of H_j : append the remainder of H_k→ H_j}
        < $Idop_k$, Sop_k , op_k > := H_k[i] ;
        Append (H_j , < $Idop_k$, Sop_k , op_k >) ;
        i := i + 1 ;
endloop ;
end MOT2 ;

LIRMM

Example: Reconciliation without using MOT2 (1)

(reconciliation without a master site)



Free propagation of update operations according to reconciliations
(epidemic propagation)

Reconciling copies of sites S_1 and S_3 ?

LIRMM

Example: Reconciliation without using MOT2 (2)

Impossible to reconcile these histories !

Histories to be merged

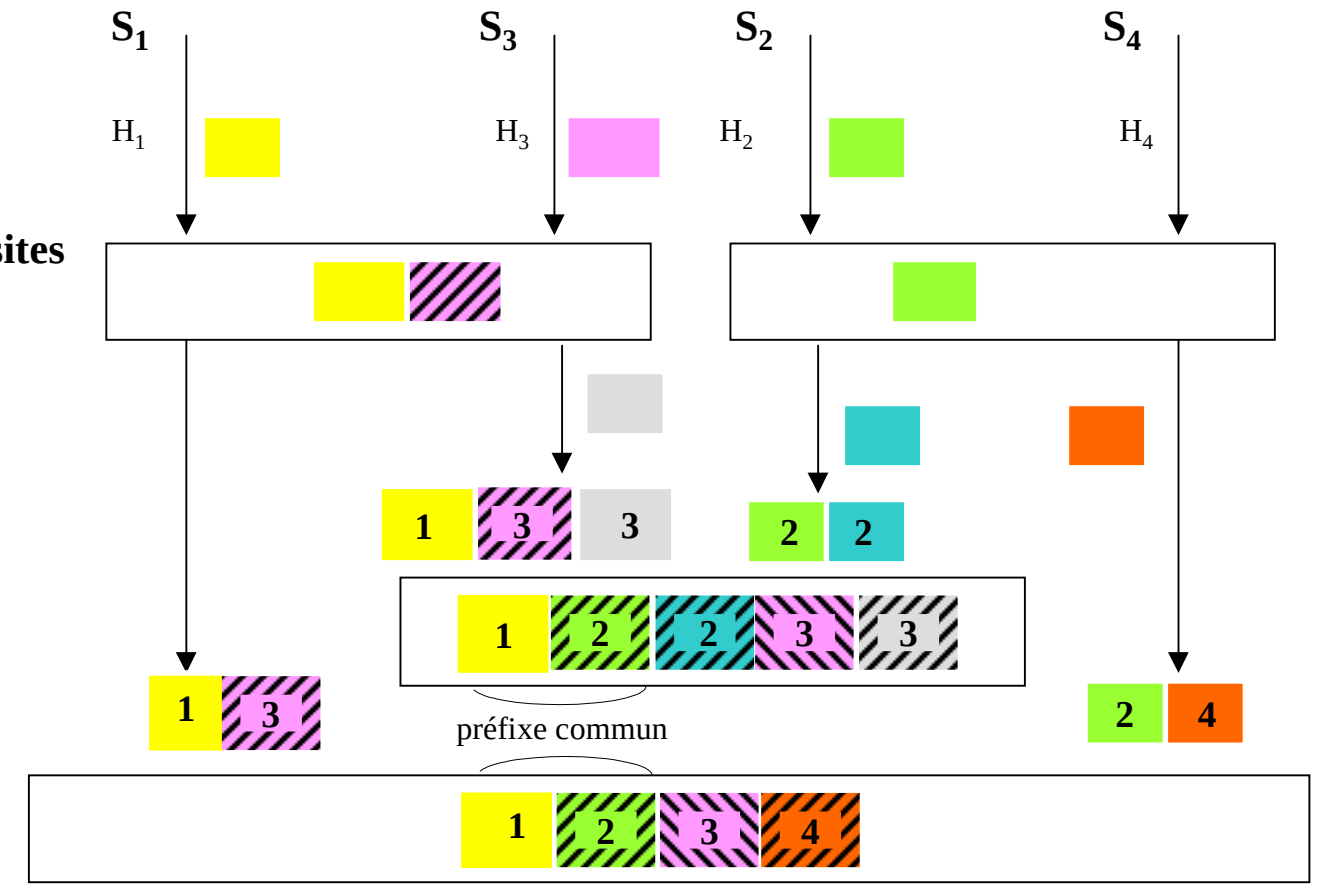
We cannot reconcile these histories because, although they have common operations (, et), they **do not appear**:

- in common prefix to both histories,
- in the same order,
- in the same form (because of forward transpositions).

LIRMM

Example: Reconciliation by using MOT2 (3)

Order between the generator sites
$S_1 < S_2 < S_3 < S_4$



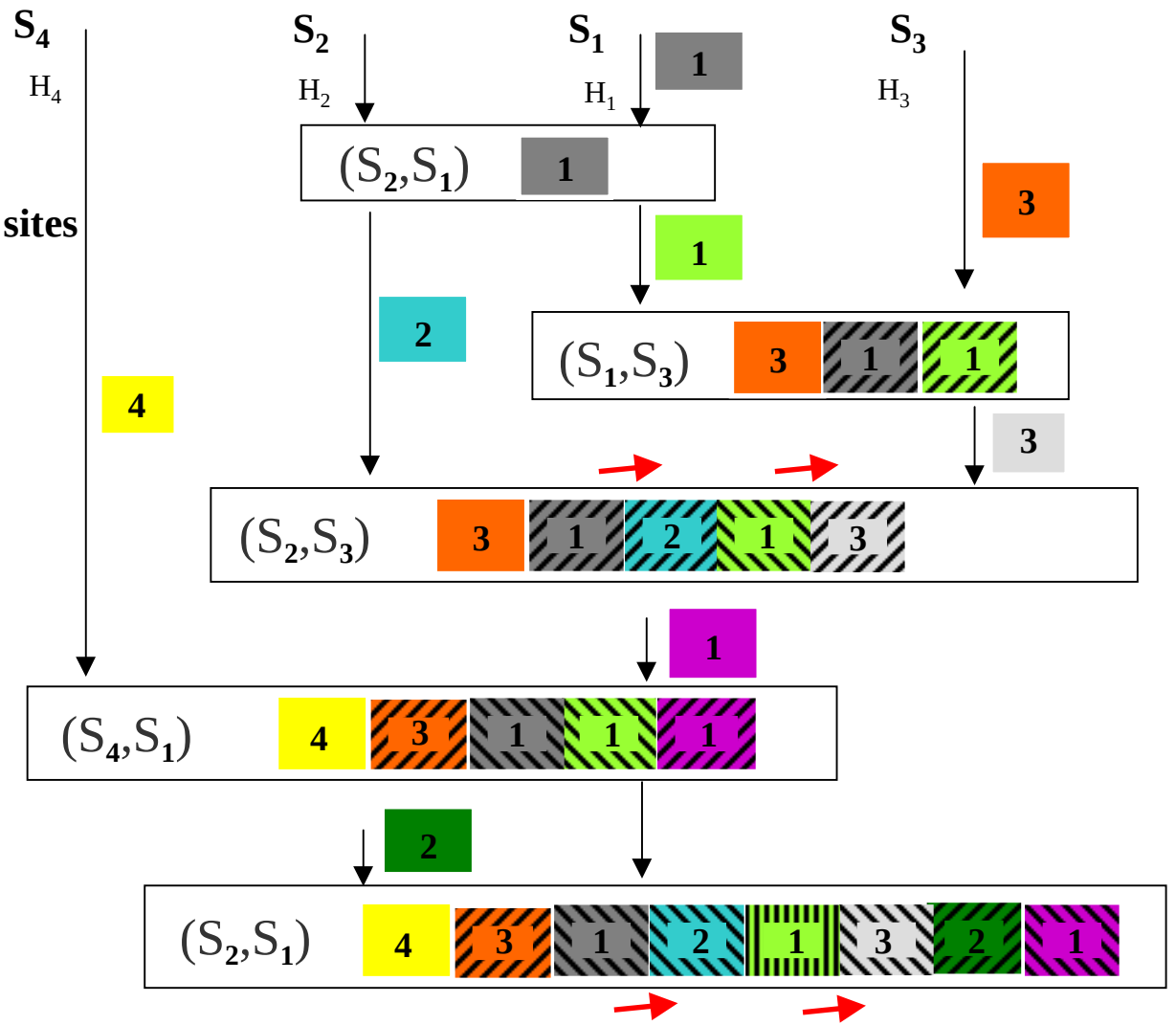
Reconciling S_1 (or S_4) and S_3 (or S_2) :

LIRMM

Example: Reconciliation by using MOT2 (4)

Order between the generator sites

$S_4 < S_3 < S_2 < S_1$



→ expresses causal precedence

LIRMM

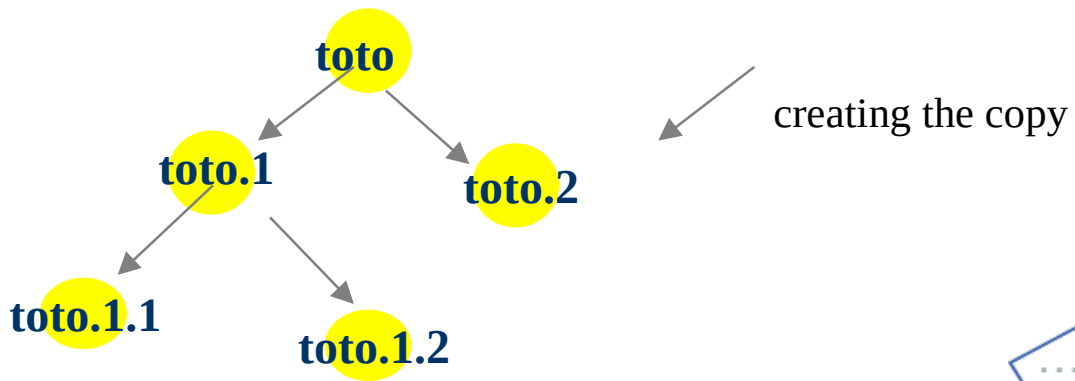
Ordering relation between sites / copies

Aim → Name of (site / copy) is unique and not re-allocated

How to obtain that ?

Site name = @IP → Problem in P2P when @IP allocated dynamically

Name of copy → Hierarchical identification schema



LIRMM

Conclusion

The MOT2 Merge Algorithm:

- reconciles pair-wise, any number of copies,
  while achieving convergence
- enables free propagation of update operations
- is based on Operational Transformation
- constructs a unique global order between operations
  (respecting the causal precedence relation)
  without requiring any ordering mechanism
  (state vectors,timestamps, sequencer, ..)
  without any centralized component
  without needing broadcast protocol
  without needing consensus protocol

→ scalable to a P2P collaborative environment