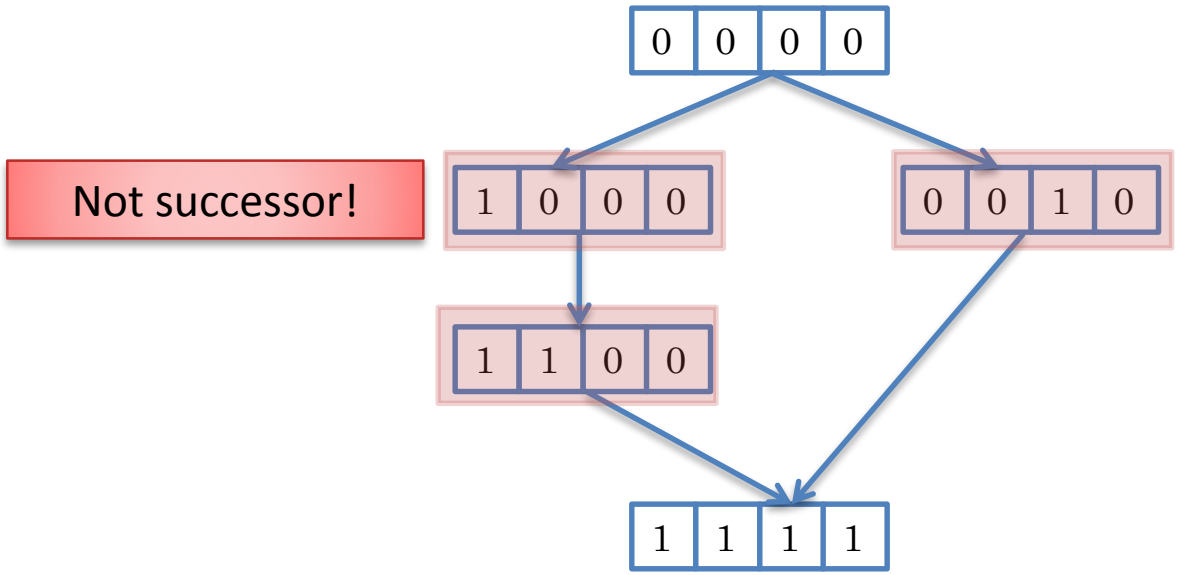
Operation Transformation

Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill
dewan@cs.unc.edu

Immediately Delivering Concurrent Messages

- A tree of message paths exist 
- Create vector time stamp and buffer for each leaf in the path
- When a message arrives see if its vector time stamp > one of the vector time stamps, put in the buffer for that vector time stamp
- Otherwise create a new vector time stamp and buffer (VectorTimeStampCopiedAndNewBufferCreated) deliver the message after flagging concurrency 

Concurrency: Different Successor



$v^1 = (a^1, .. a^n)$

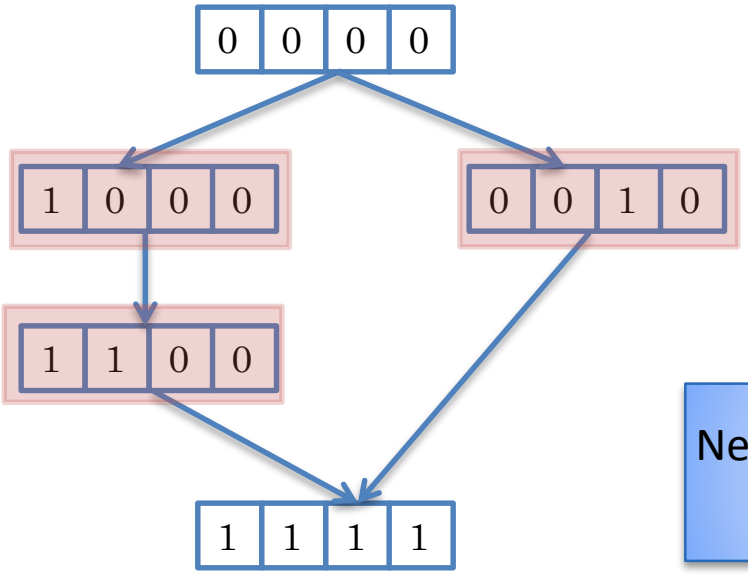
is a successor of

→

There exists $1 \leq i \leq n$, $a^i == 1 + b^i$

for all $j != i$, $a^j == b^j$

Concurrent Successors



Need to buffer messages along each path

$v^1 = (a^1, .. a^n)$

is a successor of

$v^2 = (b^1, .. b^n)$

→

There exists $1 \le i \le n$, $a^i == 1 + b^i$

for all $j != i$, $a^j <= b^j$

Delivering Concurrent Messages

- A DAG of message paths exist
- Create buffer for each leaf in the DAG
- When a message arrives find a buffer
  - See if a buffer exists in which the head message is not concurrent with this message
  - Otherwise create a new buffer
- Insert message at appropriate position in buffer
- Process buffer as before but now use a different successor function
  - $(b^1, b^2, \ldots b^n)$ is successor of $(a^1, a^2, \ldots a^n)$ if for some j, $b^j = a^j + 1$ and for all i!= j, $b^i <= a^i$
  - Takes into account that a message on a different path from an ancestor of the current time stamp node arrived

Delivering Concurrent Messages – Full Search

- A DAG of message paths exist
- Create a single buffer for all messages
- When a message arrives put it in the buffer at some position
- Process buffer as before
  - but now use a different successor function
    - $(b^1, b^2, \ldots b^n)$ is successor of $(a^1, a^2, \ldots a^n)$ if for some j, $b^j = a^j + 1$ and for all i != j, $b^i <= a^i$
    - Takes into account that a message on a different path from an ancestor of the current time stamp node arrived
  - Search the entire buffer rather than look at the head of the buffer

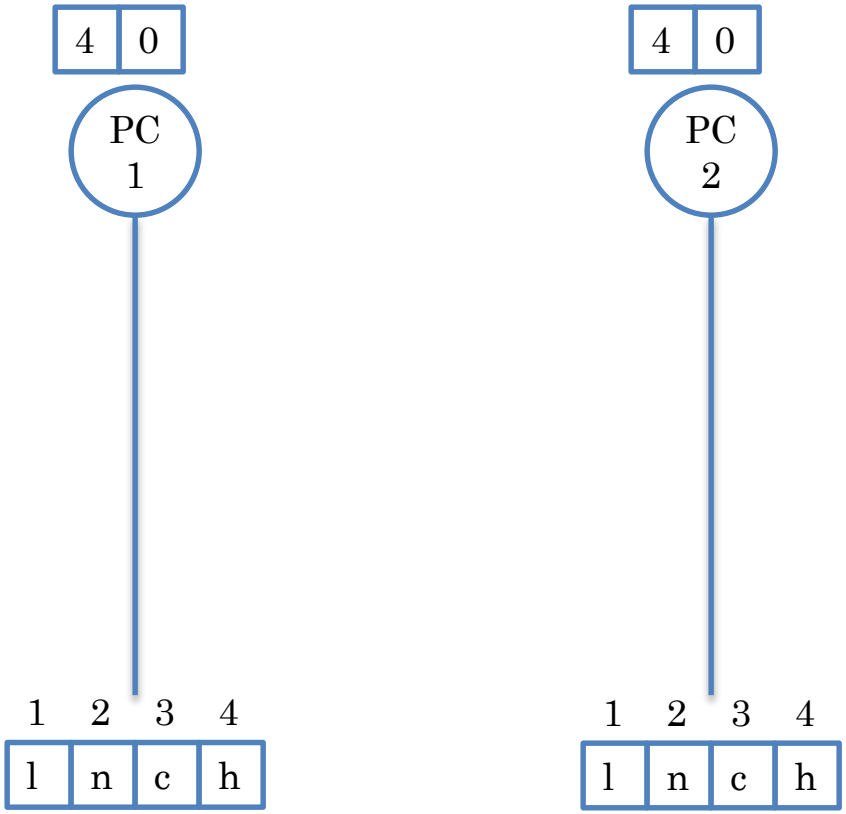
Operation Transformation

Delivering Concurrent Messages – Full Search

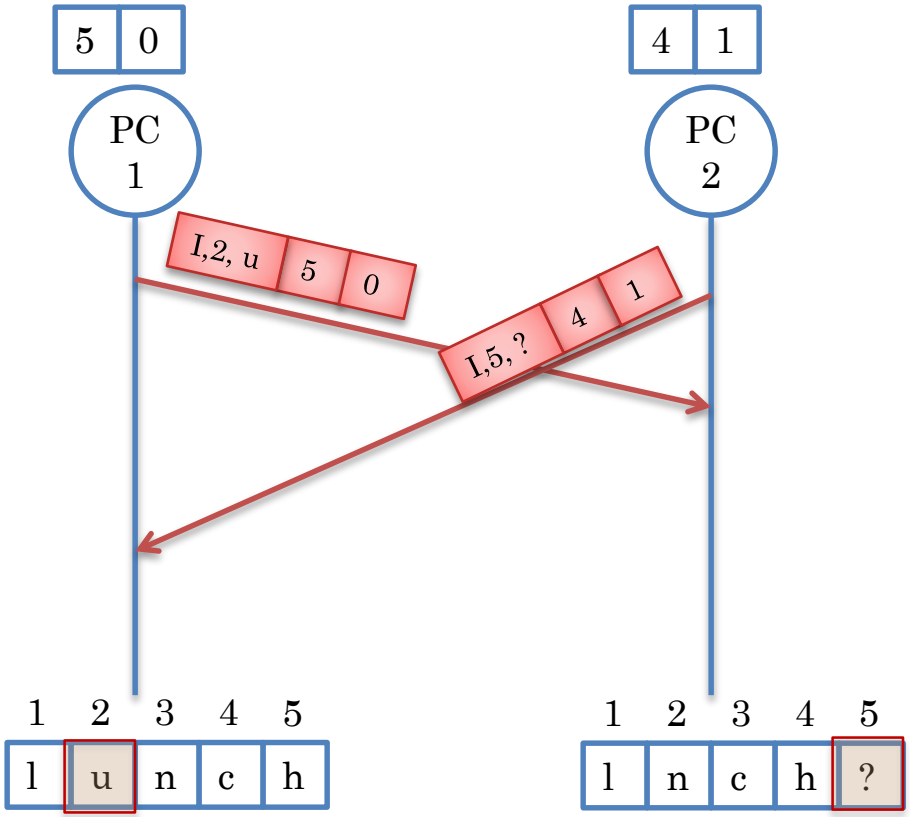
- A DAG of message paths exist
- Create a single buffer for all messages
- When a message arrives put it in the buffer at some position
- Process buffer as before
  - but now use a different successor function
    - $(b^1, b^2, \ldots. b^n)$ is successor of $(a^1, a^2, \ldots a^n)$ if for some j, $b^j = a^j + 1$ and for all i != j, $b^i <= a^i$
    - Takes into account that a message on a different path from an ancestor of the current time stamp node arrived
  - Search the entire buffer rather than look at the head of the buffer

Are causality guarantees among concurrent paths enough?

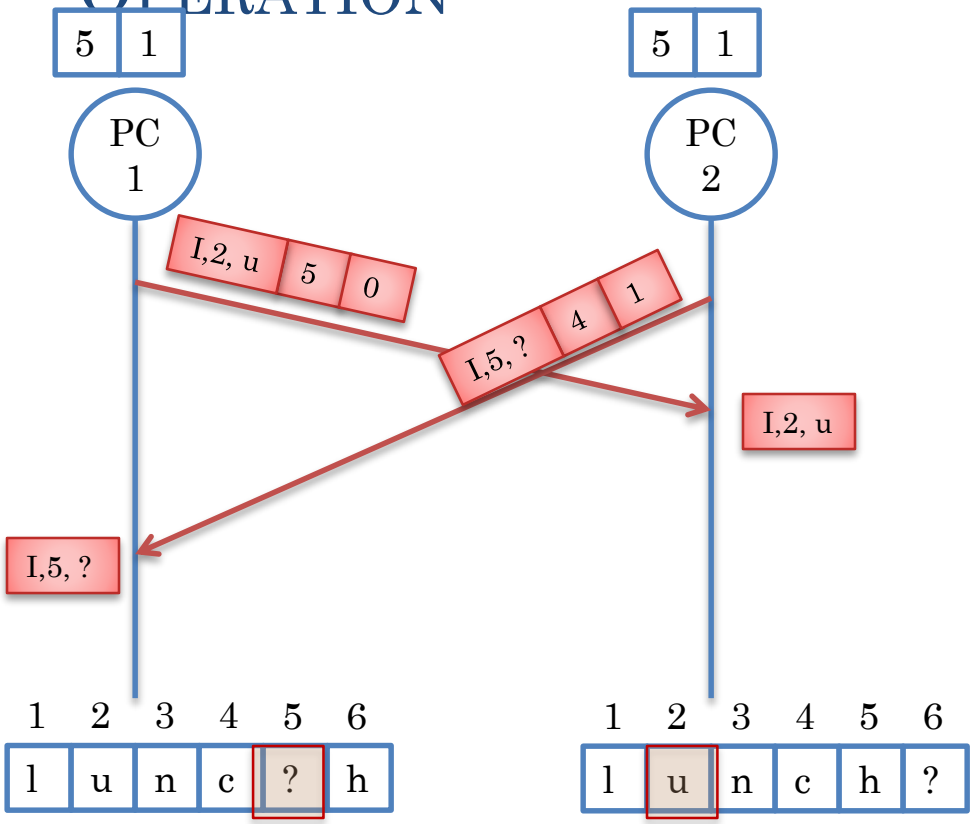
Concurrent Editing: Initial State



Concurrent Insertions



Immediately Delivering Conflicting Remote Operation



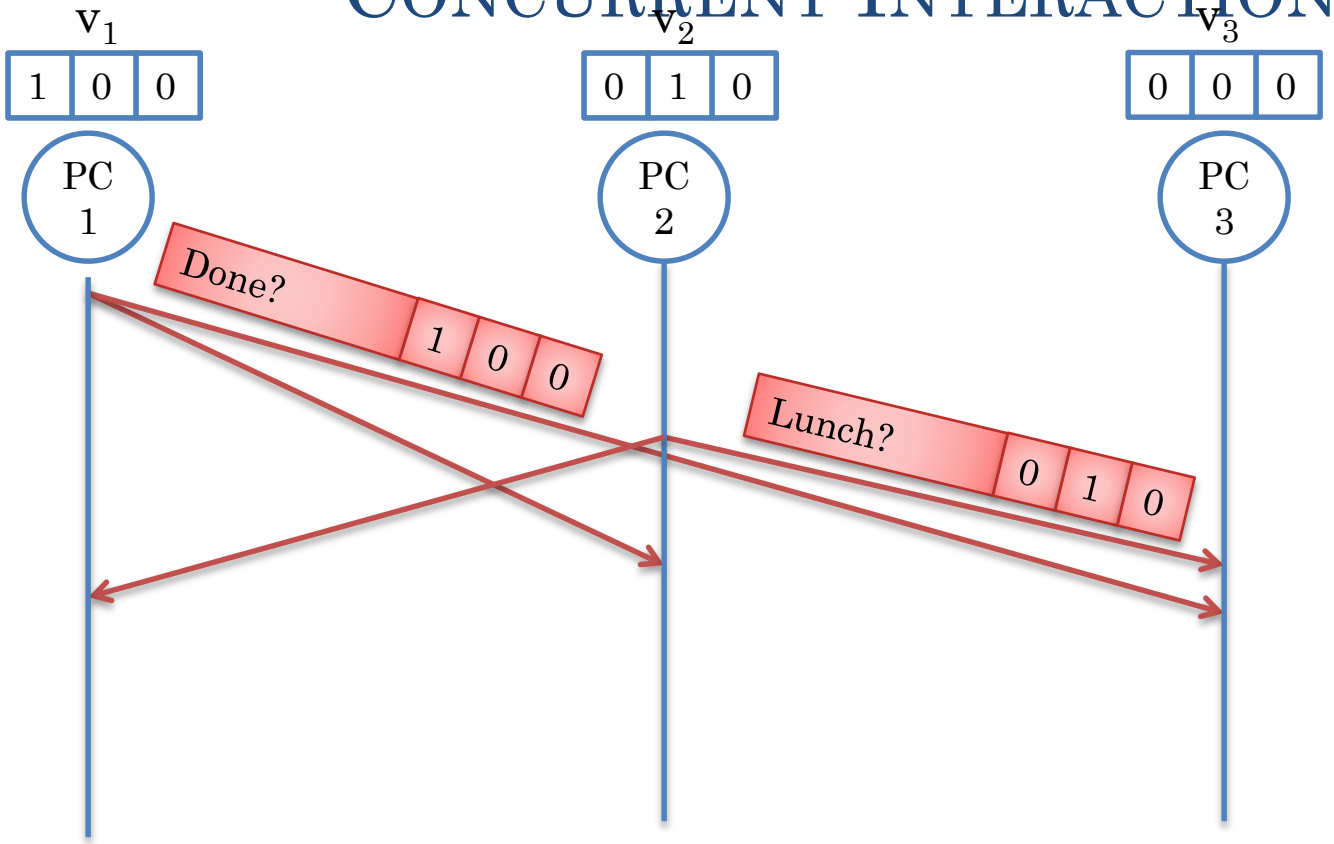
Partially Ordered Vector Time Stamps

$v = (x^1, .. x^n)$ at Site S^j → Site S^j has received x^i messages from Site S^i for all $1 \leq i \leq n$

$v^1 = (a^1, .. a^n)$ == $v^2 = (b^1, .. b^n)$	→	for all $1 \leq i \leq n$, $a^i == b^i$
$v^1 = (a^1, .. a^n)$ < $v^2 = (b^1, .. b^n)$	→	for all $1 \leq i \leq n$, $a^i \leq b^i$ for some $1 \leq i \leq n$, $a^i < b^i$
$v^1 = (a^1, .. a^n)$ || $v^2 = (b^1, .. b^n)$	→	for some $1 \leq i \leq n$, $a^i < b^i$ for some $1 \leq j \leq n$, $a^j > b^j$

Concurrent Input

Concurrent Interaction



Causal time stamps allow computer to determine concurrent actions

Do not impose a common or total order, which inherently does not exist.

Total order often important

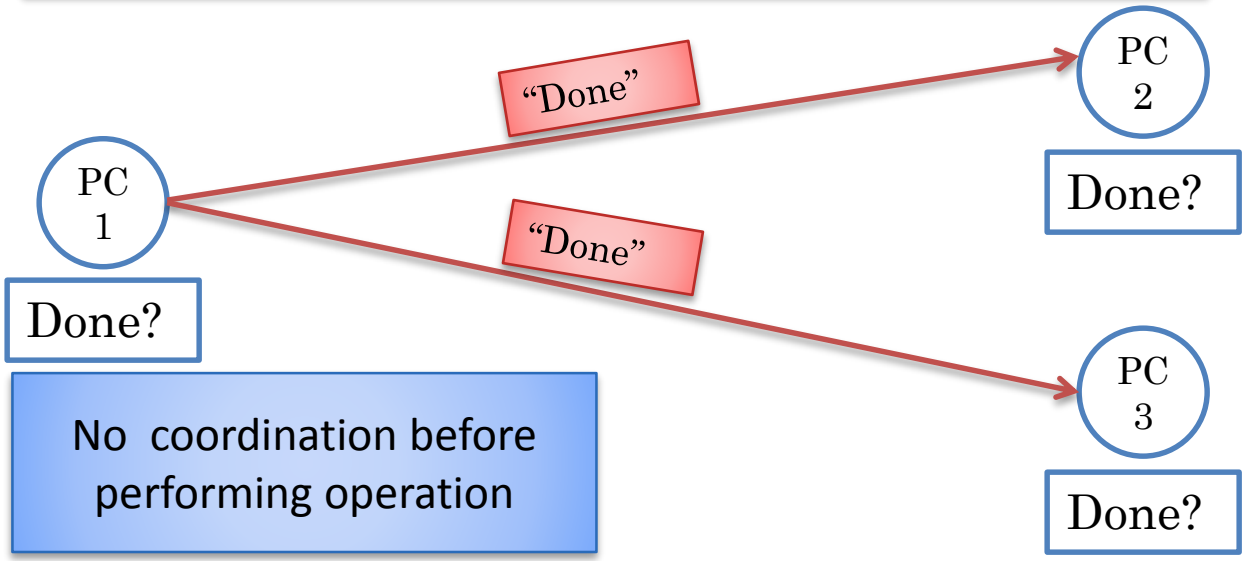
Broadcast supporting total order called atomic broadcast

Asynchronous Broadcast

1. Perform operation o

2. toOthers send the operation using peer to peer communication

3. Perform received operation (in causal order)



No coordination before performing operation

Atomic asynchronous, application-unaware broadcast impossible

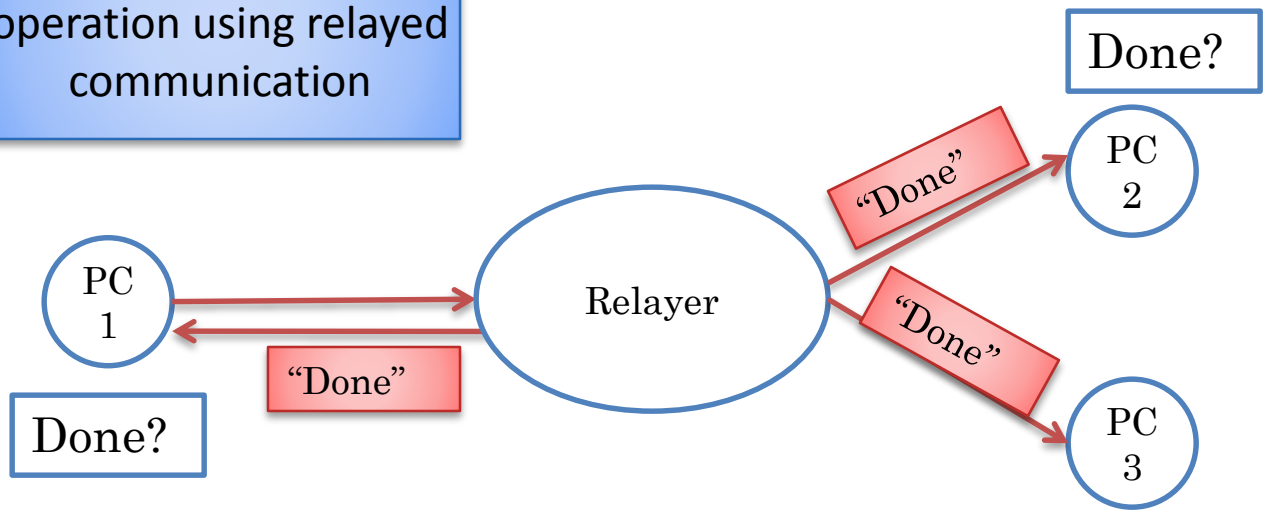
Synchronous and Coordination?

Synchronous Relayed Broadcast

1. Perform operation o

2. toAll send the operation using relayed communication

3. perform operation on its receipt

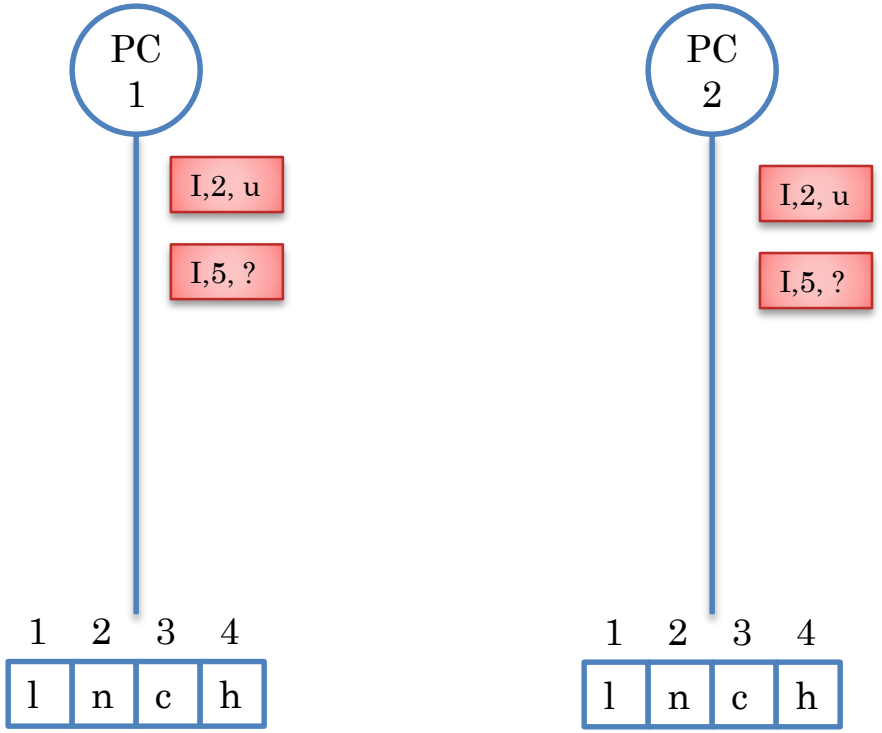


Each site performs operations in the same sequence assuming ordered unicast

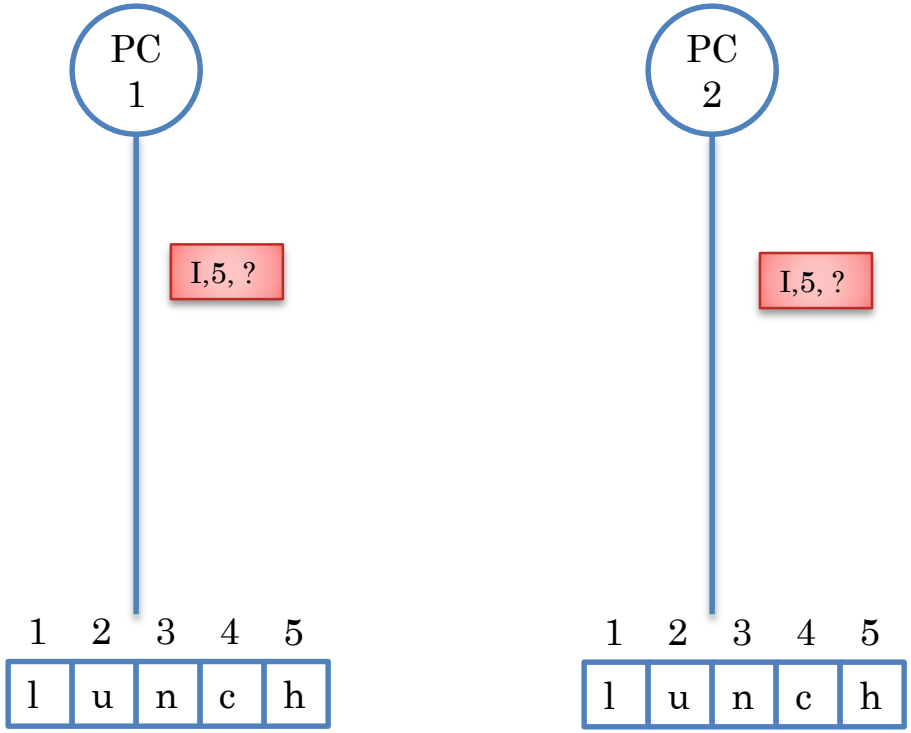
Done?

Delay (extra hop) state of operation issue not same as state of execution, though all sites are consistent

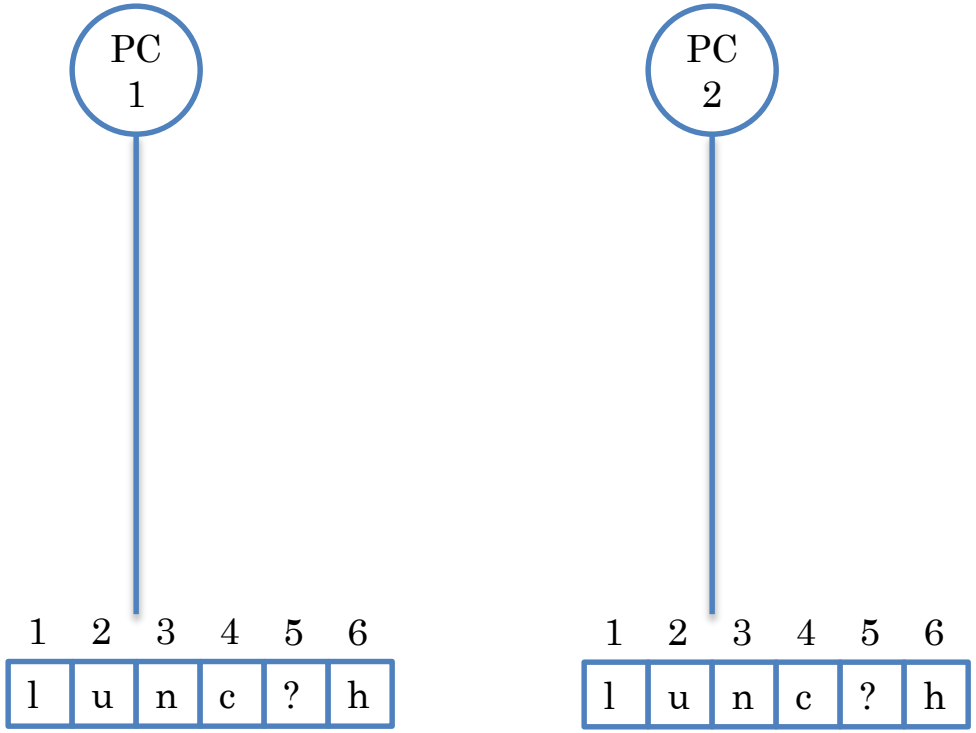
Ordering with Atomic Broadcast



First Operation Executes



Second Operation Executes

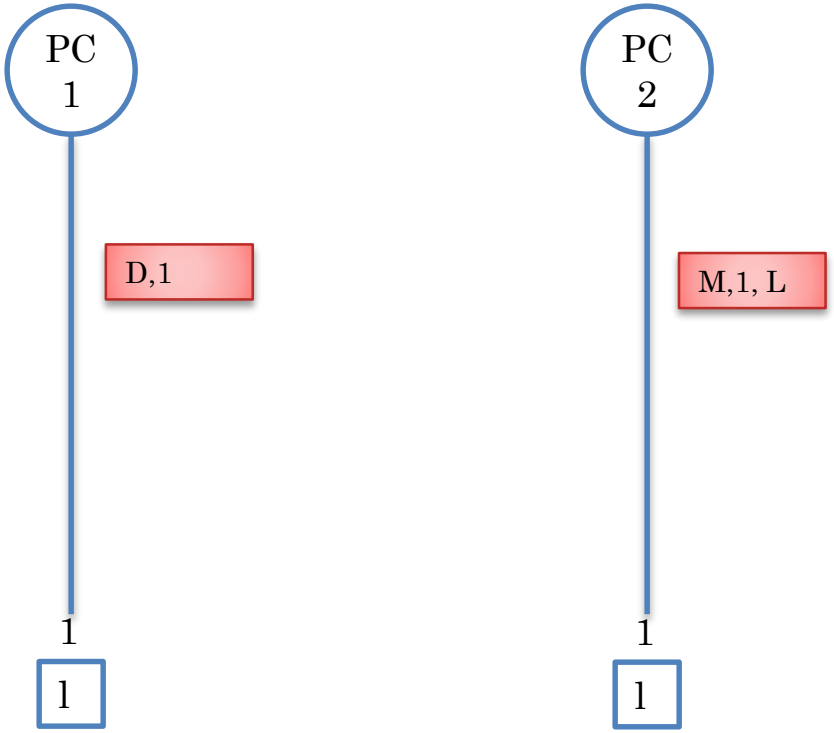


Common state but "intention" violation

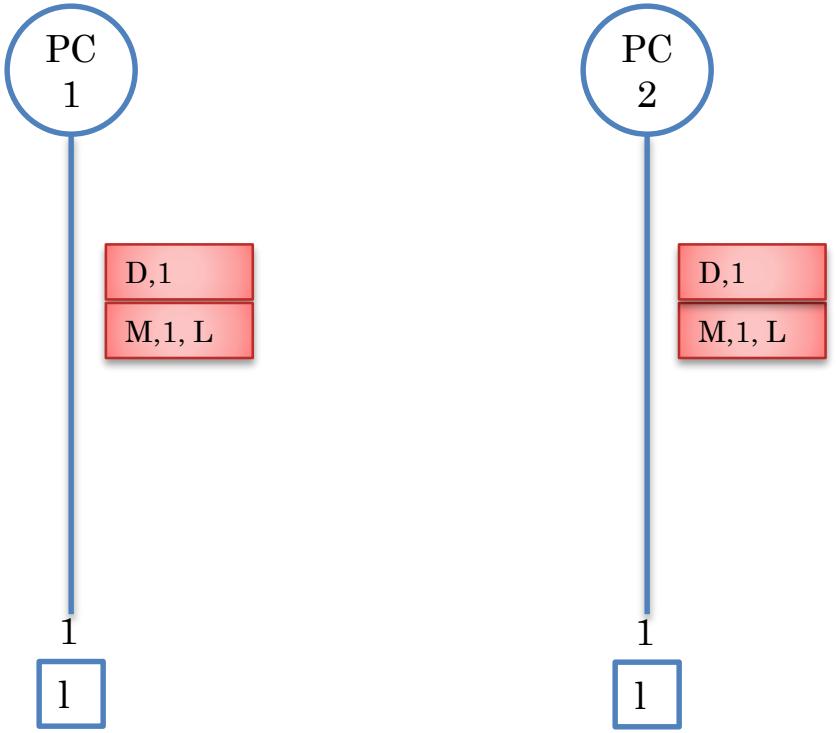
Context of operation not the same as when it was issued

Worse outcome than intention violation?

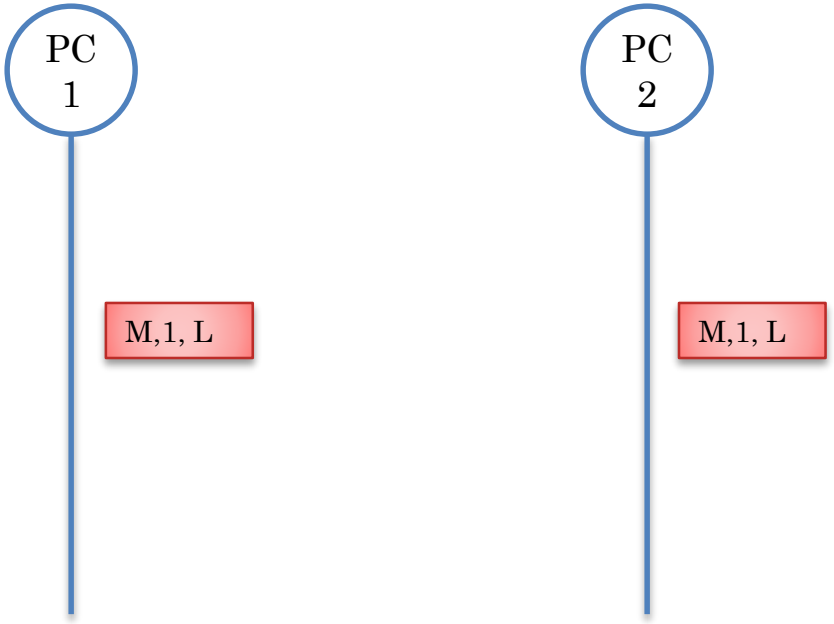
Concurrent Interaction: Delete, Modify



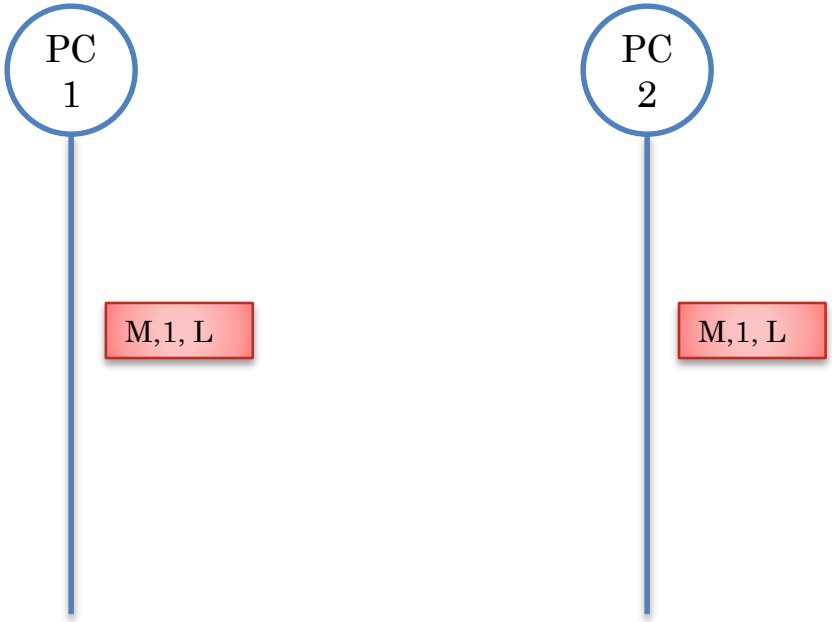
Atomic Broadcast Ordering



First Operation Executes



Second Operation Causes Exception



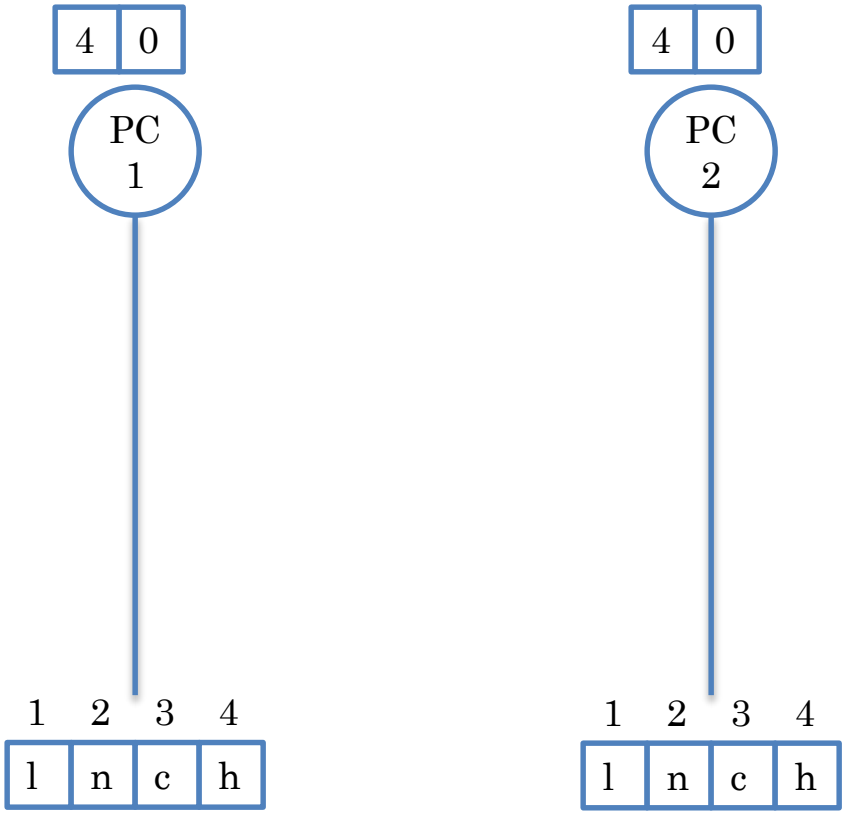
ArrayIndexOutOfBounds

Common behavior but exception

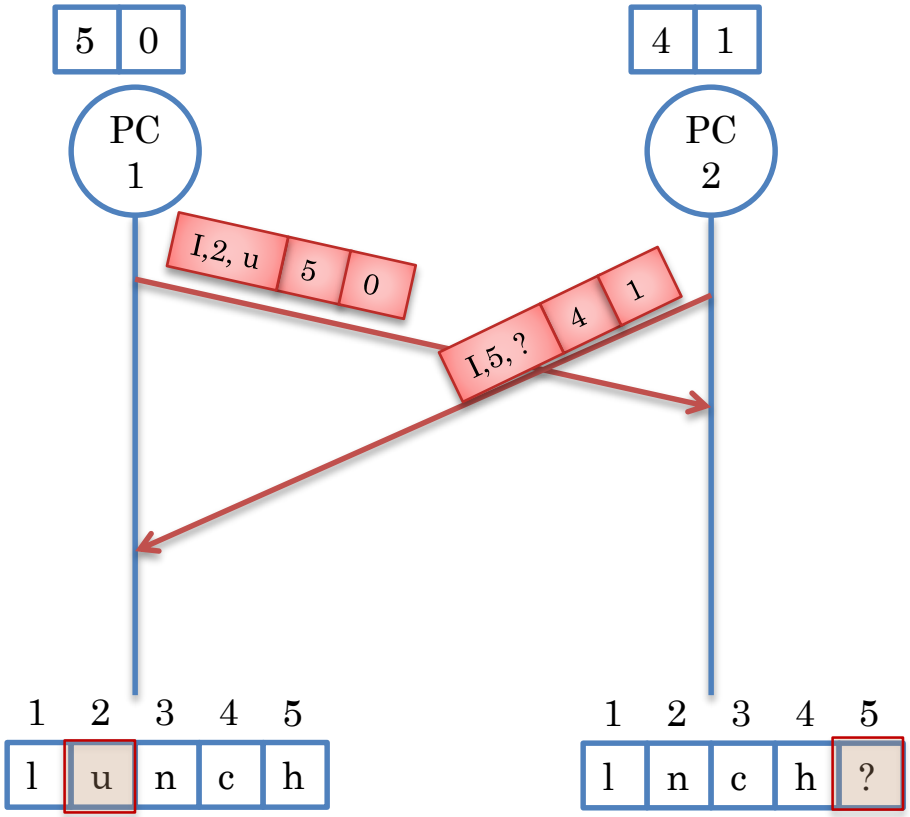
Context of operation not the same as when it was issued

Can concurrency control explain or fix this?

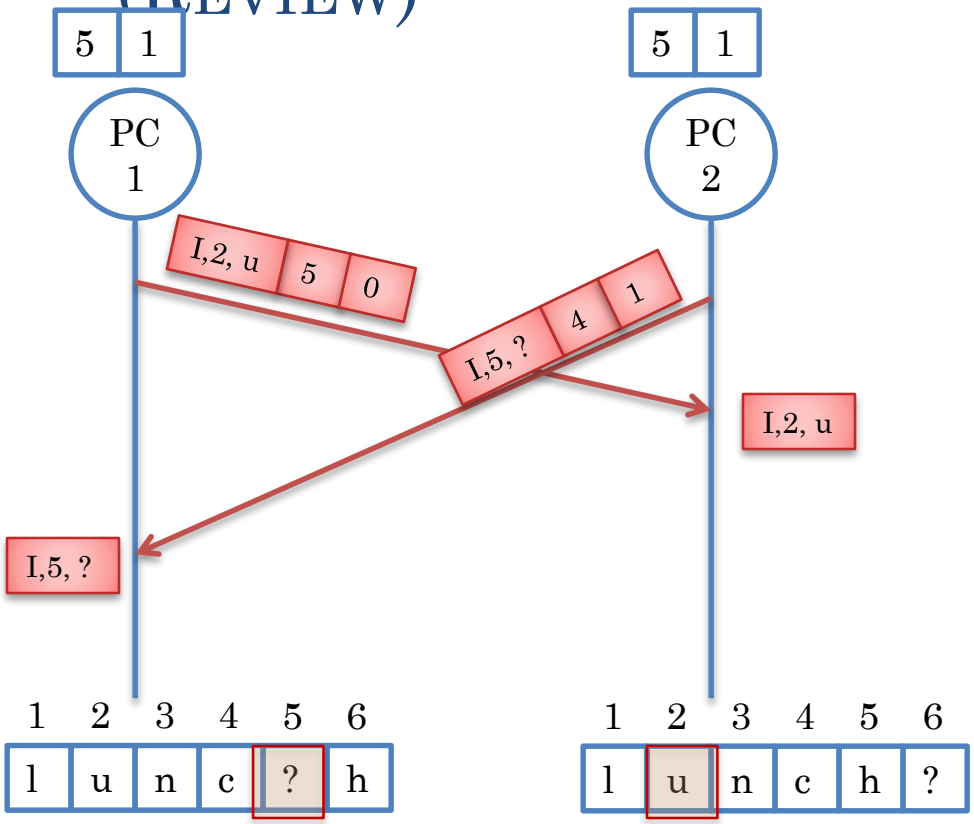
Concurrent Editing: Initial State (Review)



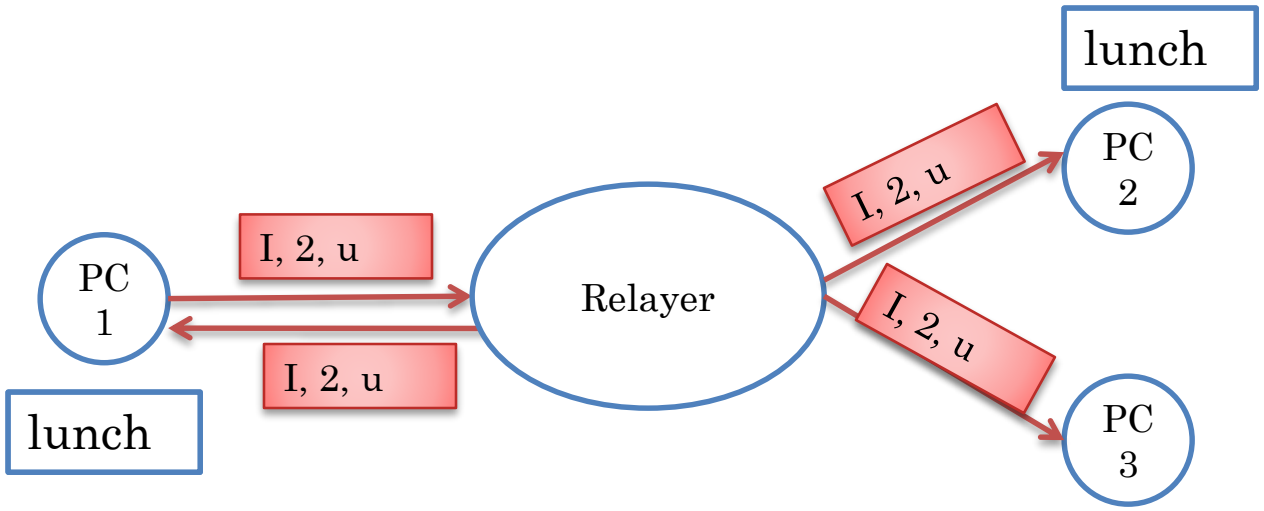
Concurrent Insertions (Review)



Delivering Conflicting Remote Operation (Review)

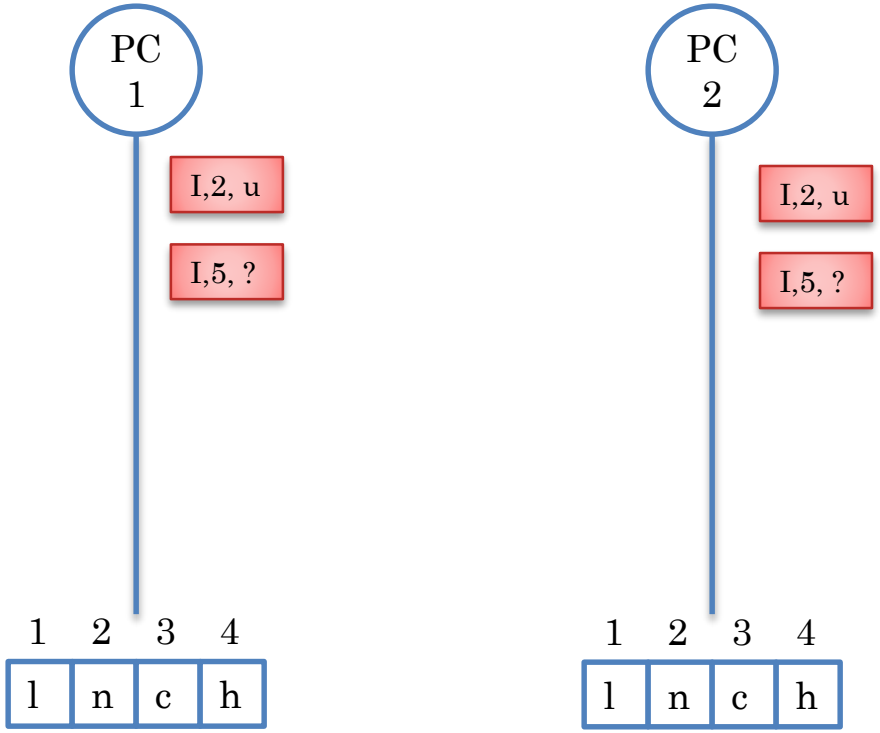


Synchronous Relayed Broadcast (Review)

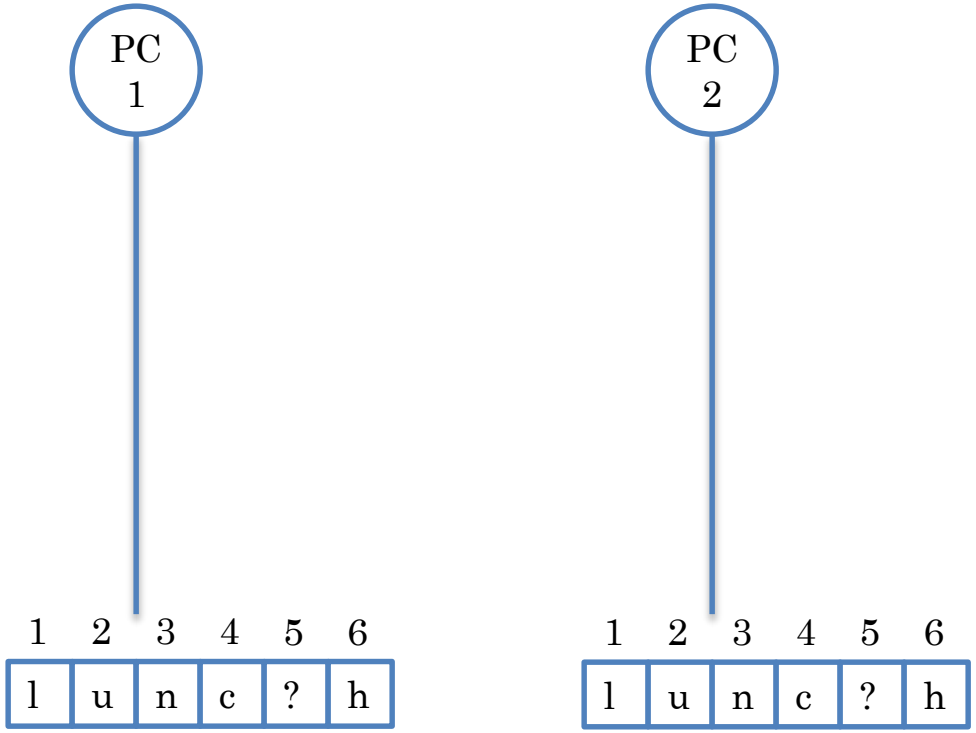


Delay (extra hop) state of operation issue not same as state of execution, though all sites are consistent

Ordering with Atomic Broadcast (Review)



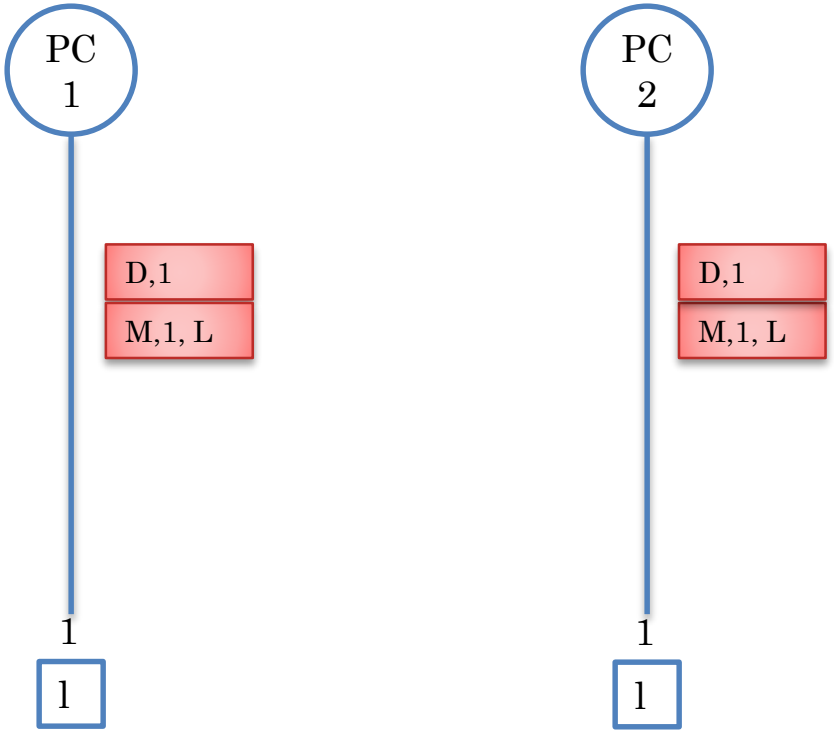
Second Operation Executes (Review)



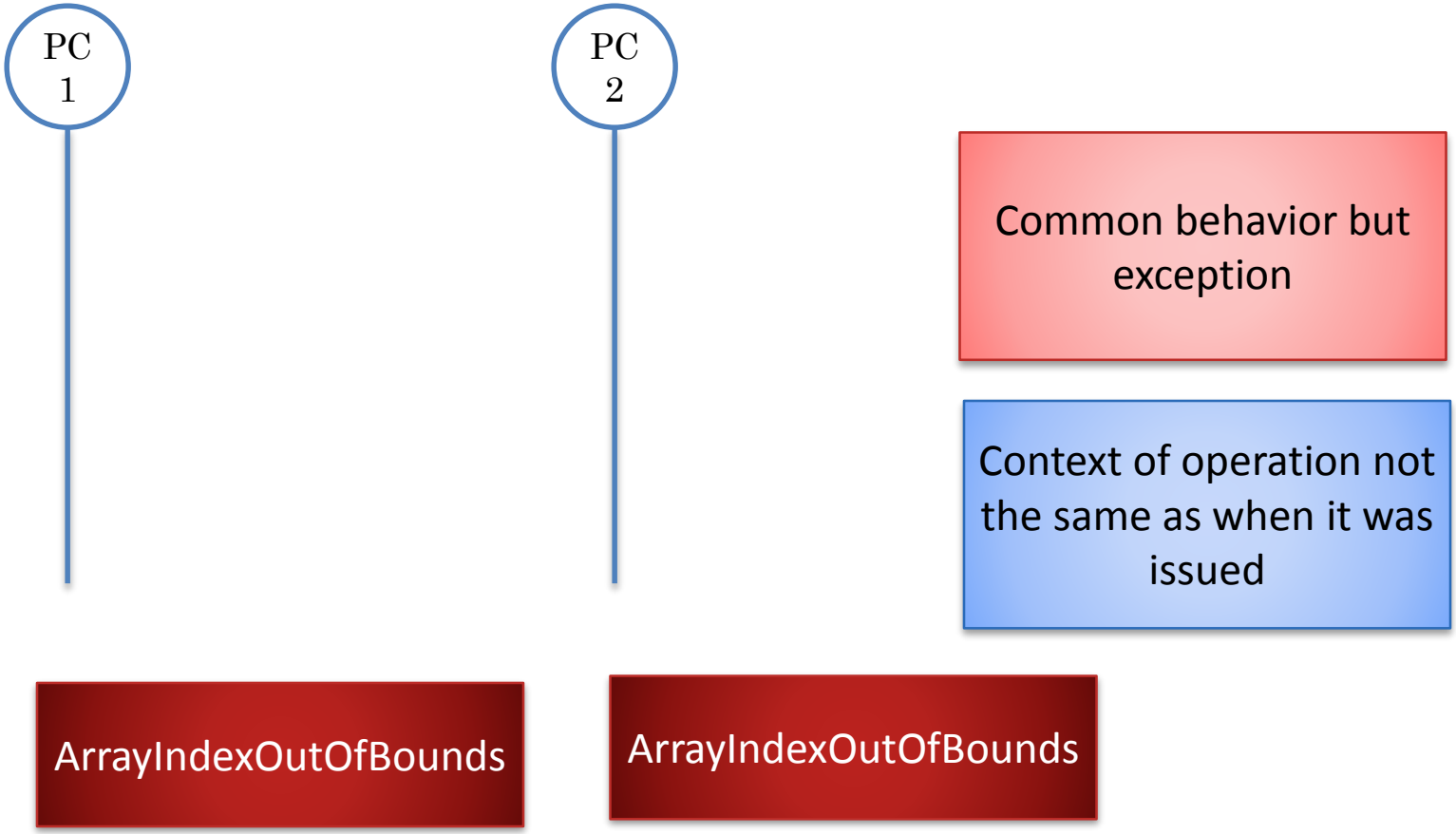
Common state but "intention" violation

Context of operation not the same as when it was issued

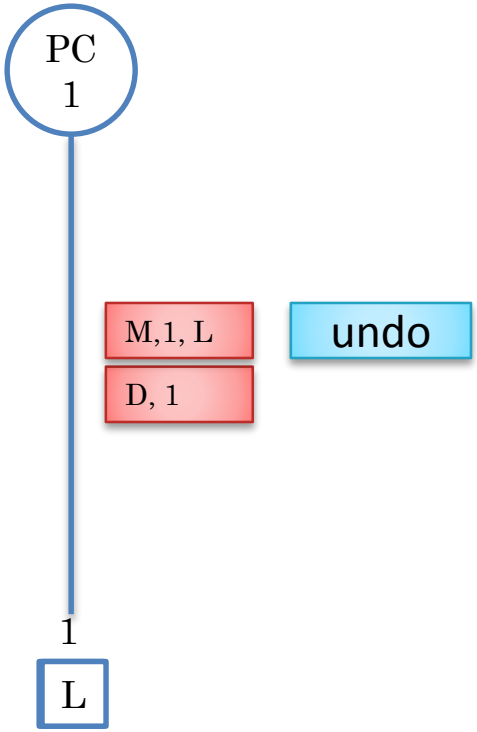
Atomic Broadcast Ordering



Second Operation Causes Exception

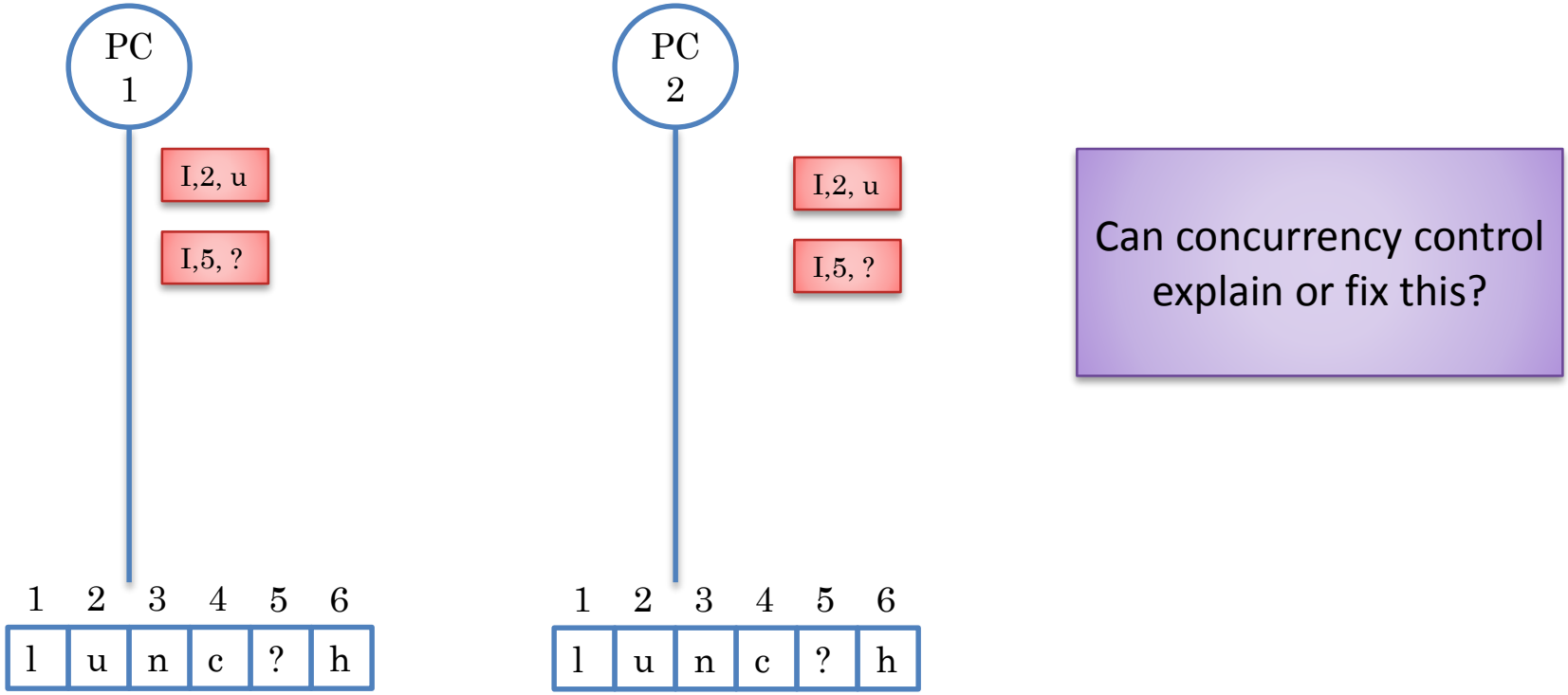


Single-User Case: Selective Undo



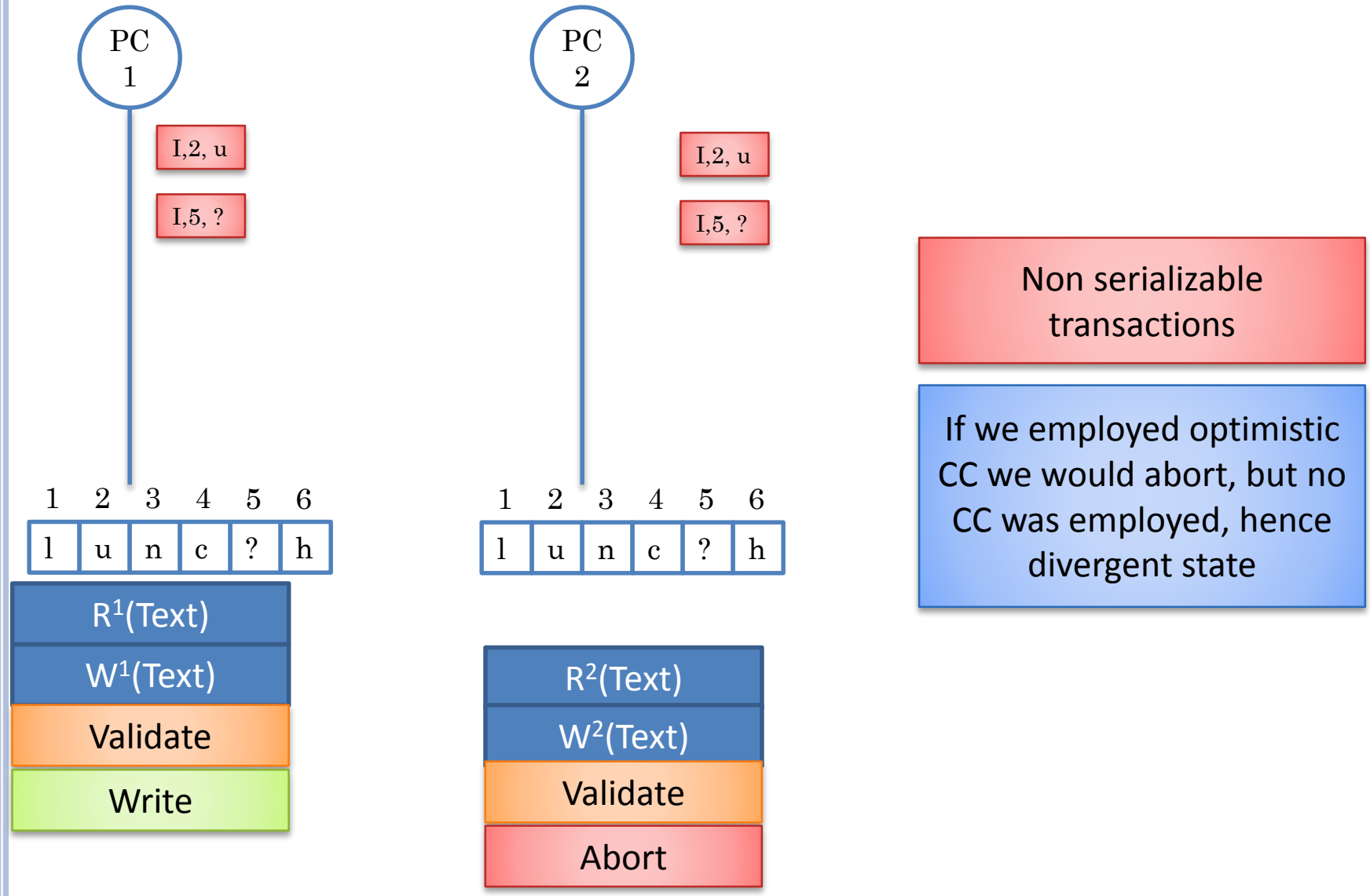
Undoing non last operation is done in a context different from the one in which it was executed

Concurrency Control?



Can concurrency control explain or fix this?

Optimistic Transaction



Validation/Checking Time (Review)

- Early
  - Pessimistic
- Late
  - Optimistic
- Merging

Early vs. late validation (Review)

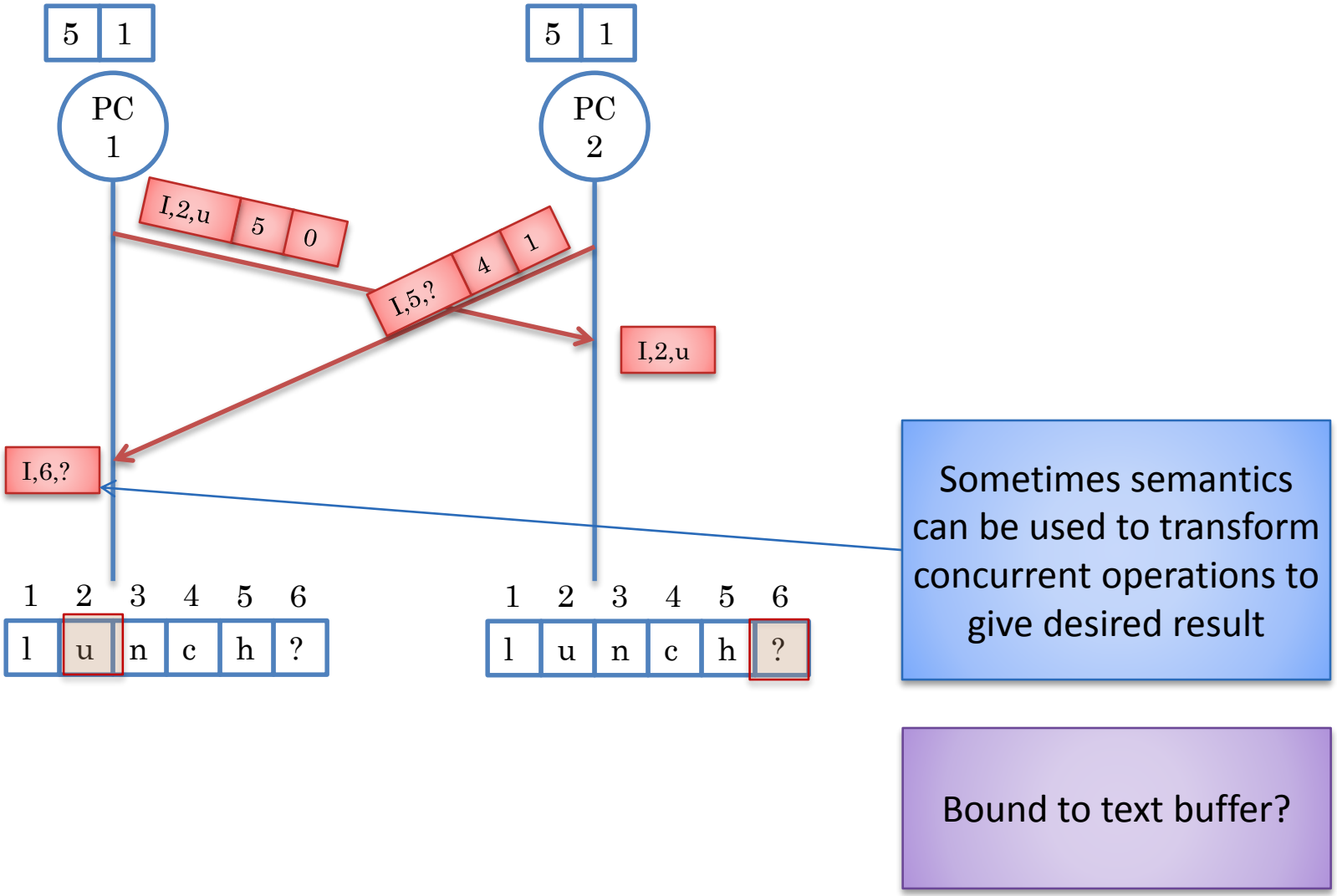
- **Per-operation checking and communication overhead**
- No compression possible.
- Prevents inconsistency.
- Tight coupling: incremental results shared
- Not functional if disconnected
  - Unless we lock very conservatively, limiting concurrency.

- No per-operation checking, communication overhead
- Compression possible.
- **Inconsistency possible resulting in lost work.**
- Allows parallel development.
- Functional when disconnected.

Merging (Review)

- Like optimistic
  - Allow operation to execute without local checks
- But no aborts
  - **Merge conflicting operations**
  - **E.g. insert 1,a || insert 2, b = insert 1, a; insert 3, b || insert 2, b; insert 1,** a
- **Serializability not guaranteed**
  - **Ignore reads**
  - New transaction to replace conflicting transactions
  - Strange results possible
    - E.g. concurrent dragging of an object in whiteboard
- **App-specific**

Transforming Remote Operation



Sometimes semantics can be used to transform concurrent operations to give desired result

Bound to text buffer?

Assume Indexed Sequence Data Type

1	Element 1
2	Element 2
3	Element 3

Arbitrary element type

1	Lunch?
2	Lunch?
3	No, Yes

String element type (message sequence)

1	2	3	4	5
l	u	n	c	h

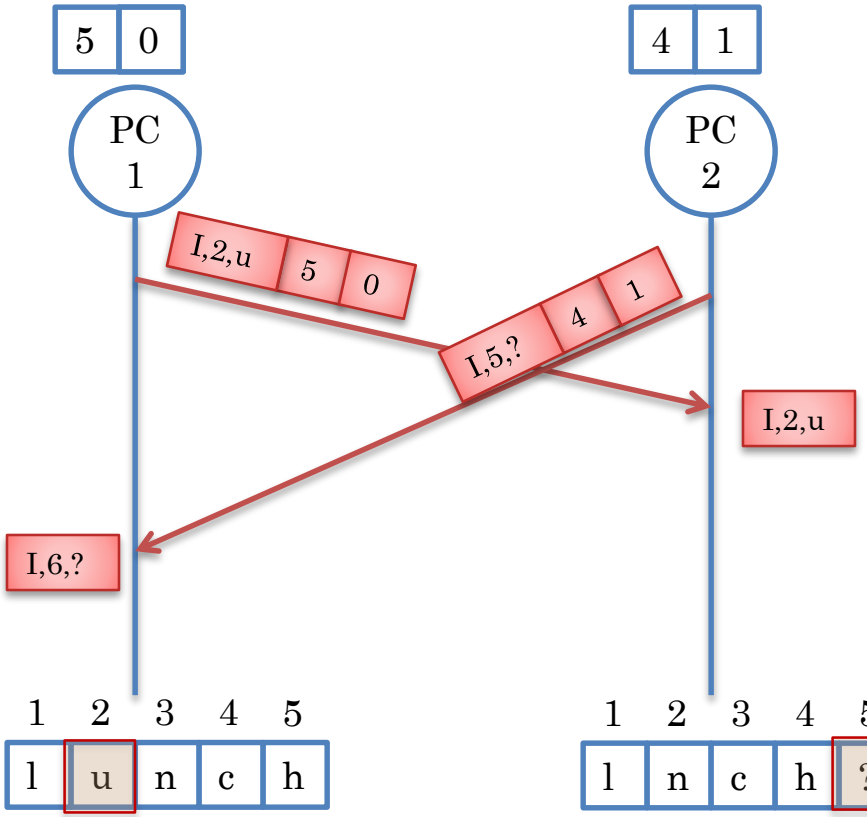
char element type (text editors)

I(nsert) (index, element)

D(elete)(index))

Operations

Concurrent Inorder Interaction



Assume only two sites

Assume messages are received in order from other site

No need for received buffer

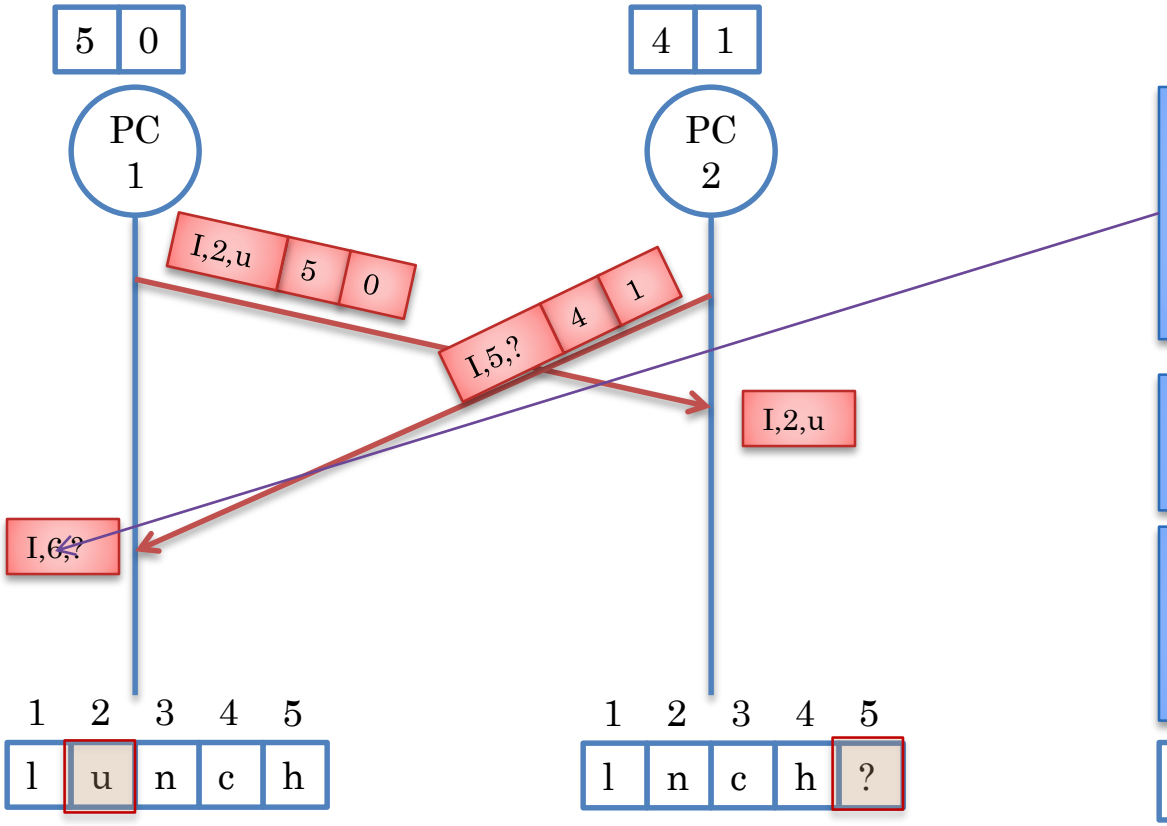
Still need time stamp to discover concurrency

General rule for transforming operation

Received Buffer$_1$

Received Buffer$_2$

Transformation Function



Increment index of remote operation if it has higher index before processing it

Remote operation, R transformed!

Based on index of concurrent local operation, L

R^T = Transform (R, L)

Apply R^T instead of R at local site

R Transformed if R^T != R

Need local buffer to store L rather than remote buffer

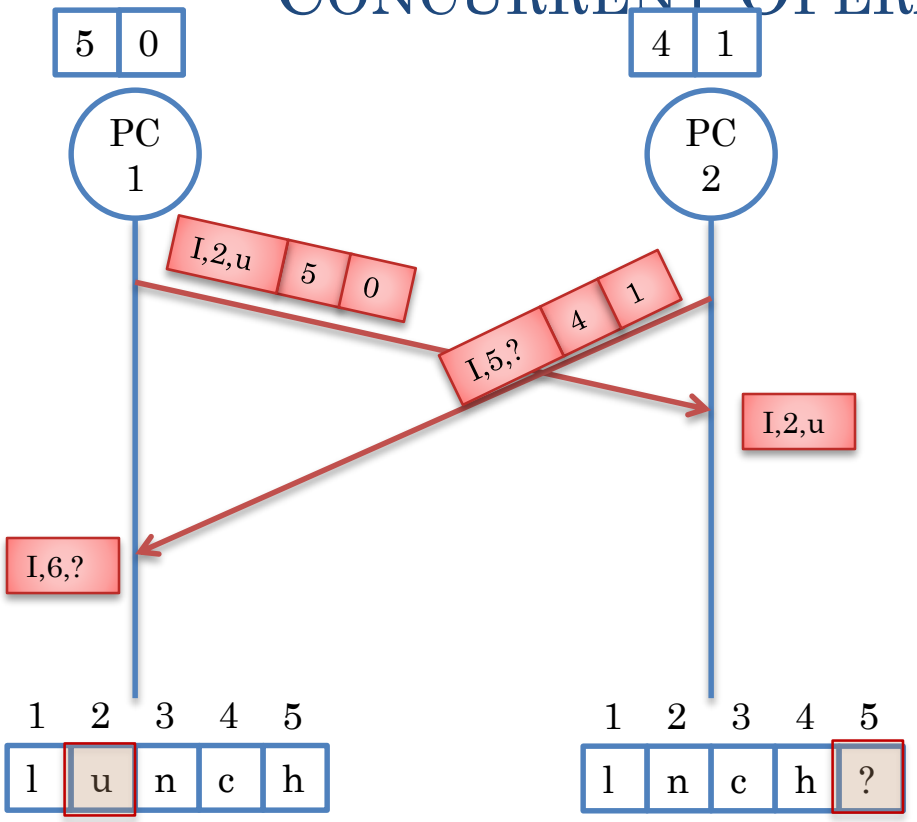
Local Buffer$_1$

I,2,u	5	0

Local Buffer$_2$

I,5,?	4	1

Control Algorithm: Single Local Concurrent Operation



Given Remote op R, concurrent with exactly one local op L

R^T = Transform (R, L)

Execute R^T

Site.TimeStamp.increment(R.site)

OT System Components

Transformation function: Handles single local and concurrent operations

Control algorithm: Calls transformation function, processed buffer and local time stamps

Both must be correct.

Inclusion Transformation

```
Operation Transform (Operation R,  Operation L)  {
        if (R.type == Insert && L.type == Insert)
                return TransformInsertInsert (R,L);
        else ….
}
```

```
Operation TransformInsertInsert (InsertOperation R, InsertOperation L)  {
        Operation R^T = R.deepCopy();
        if (R.index > L.index)
                R^T.index = R.index + 1
        return R^T ;
}
```

Transform includes effect of second operand on first operand

Other uses in which first and second operands are not remote and local operations

Correct?

Correctness criterion?

Names indicate we include effect of earlier executed local operation on later received concurrent remote operation

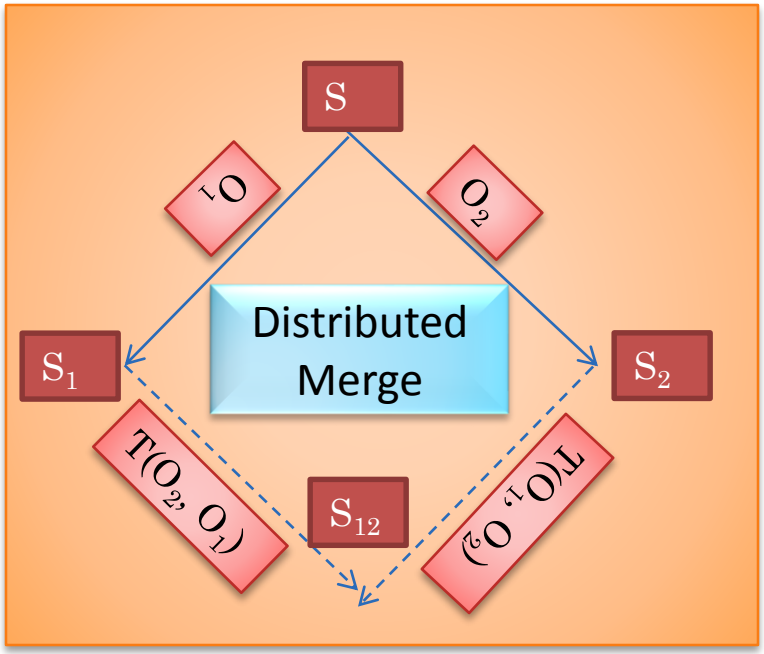
Called inclusion transformation

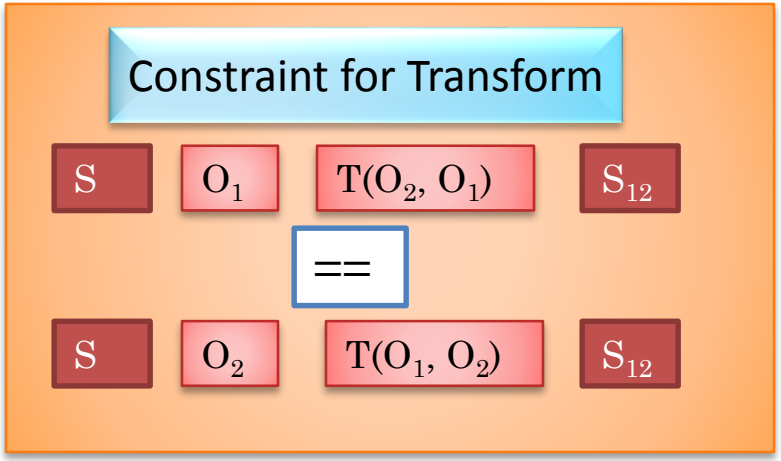
Correctness Criterion

```
Operation Transform (Operation R,  Operation L)  {
        if (R.type == Insert && L.type == Insert)
                return TransformInsertInsert (R,L);
        else ….
}
```

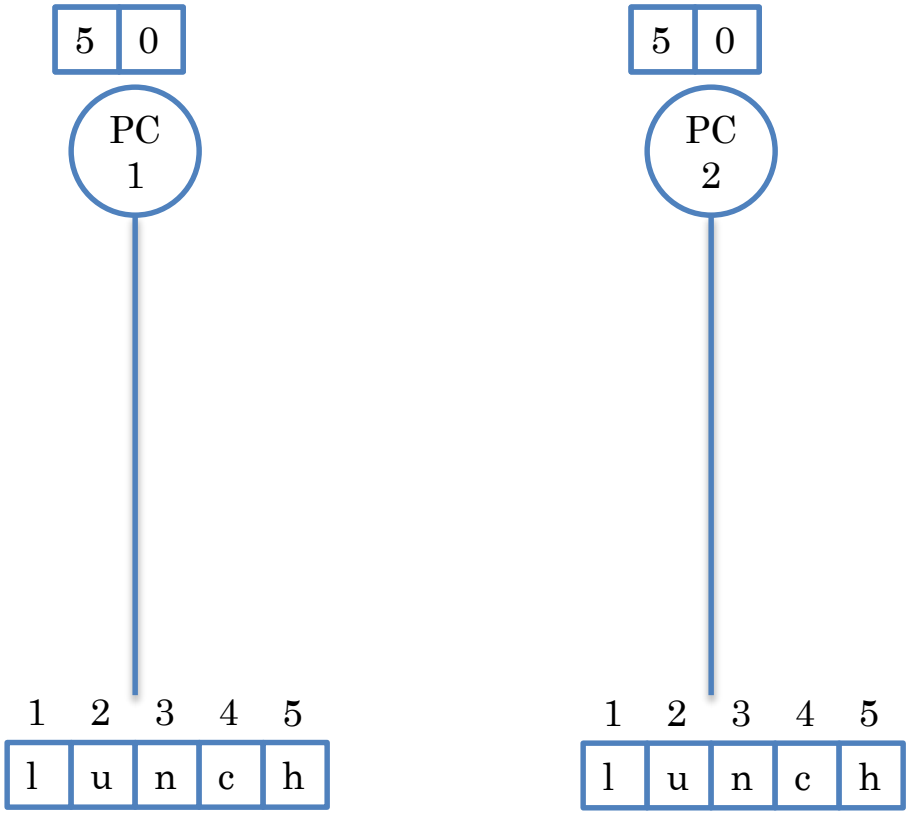
==?

```
Operation TransformInsertInsert (InsertOperation R, InsertOperation L)  {
        Operation R^T = R.deepCopy();
        if (R.index > L.index)
                R^T.index = R.index + 1
        return R^T ;
}
```

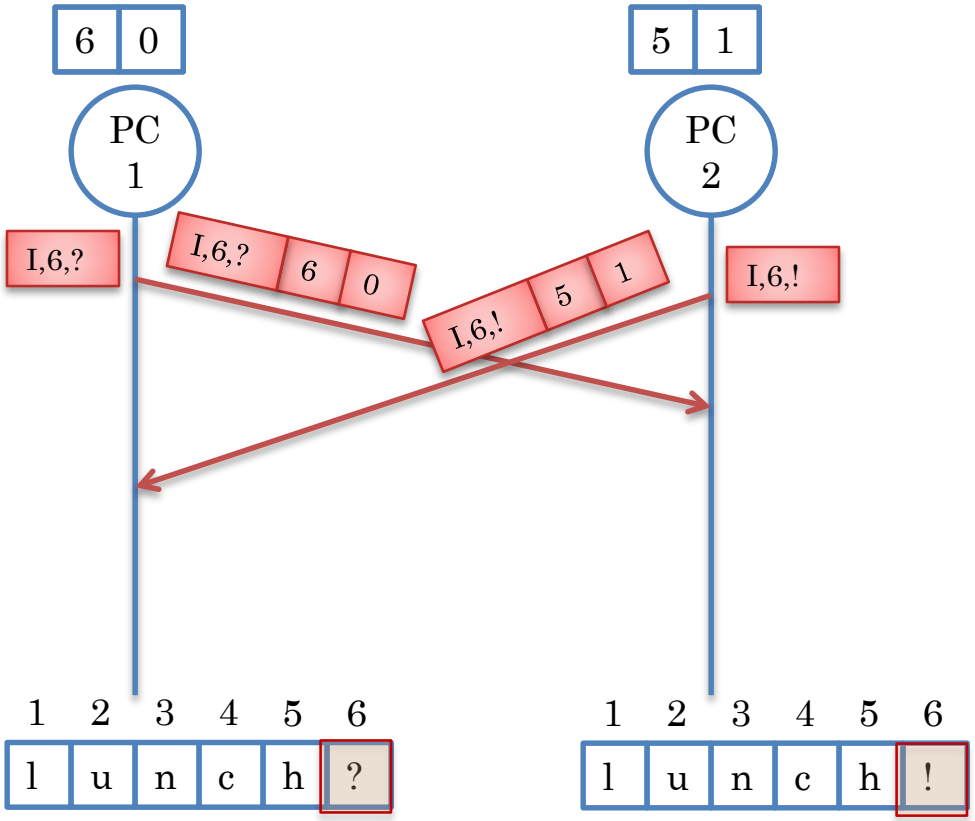




Initial State



Insertion at same index



Local Buffer$_1$

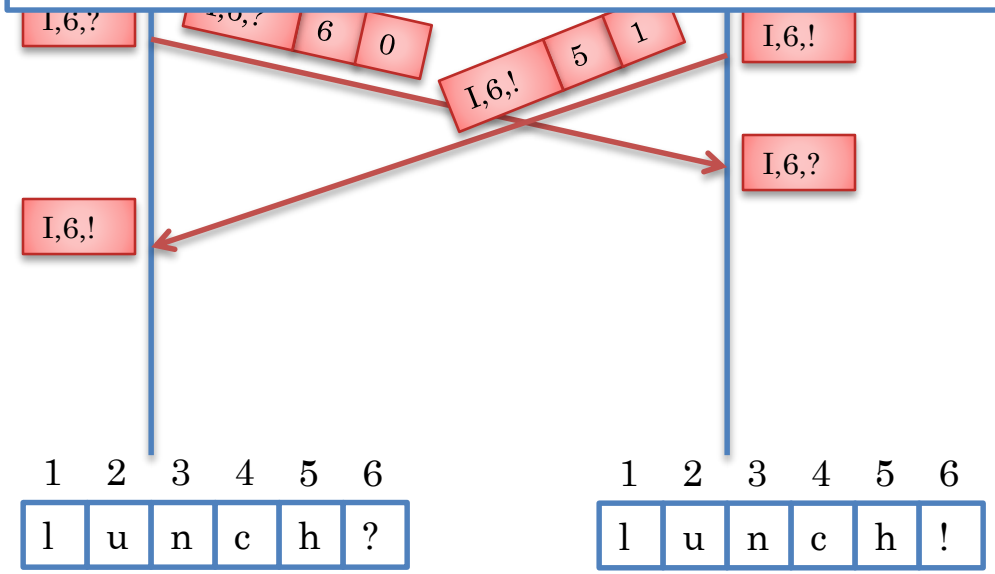
I,6,?	6	0

Local Buffer$_2$

I,6,!	5	1

Insertion at same index

```
InsertOperation TransformInsertInsert (InsertOperation Remote, InsertOperation L ocal)  {
        Operation Remote^T = Remote.clone();
        if (Remote.index > Local.index)
                Remote^T.index = Remote.index + 1
        return Remote^T ;
}
```



Neither operation is transformed

Local Buffer$_1$

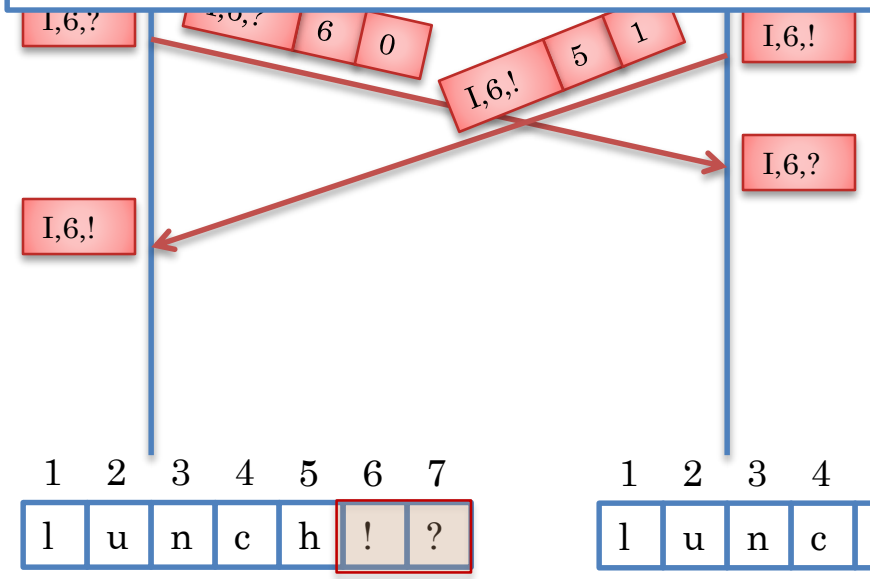
I,6,?	6	0

Local Buffer$_2$

I,6,!	5	1

Insertion at same index Algorithm

```
InsertOperation TransformInsertInsert (InsertOperation Remote, InsertOperation L ocal) {
        Operation Remote^T = Remote.clone();
        if (Remote.index > Local.index)
                Remote^T.index = Remote.index + 1
        return Remote^T ;
}
```



Neither operation is transformed

Inconsistency

Local Buffer$_1$

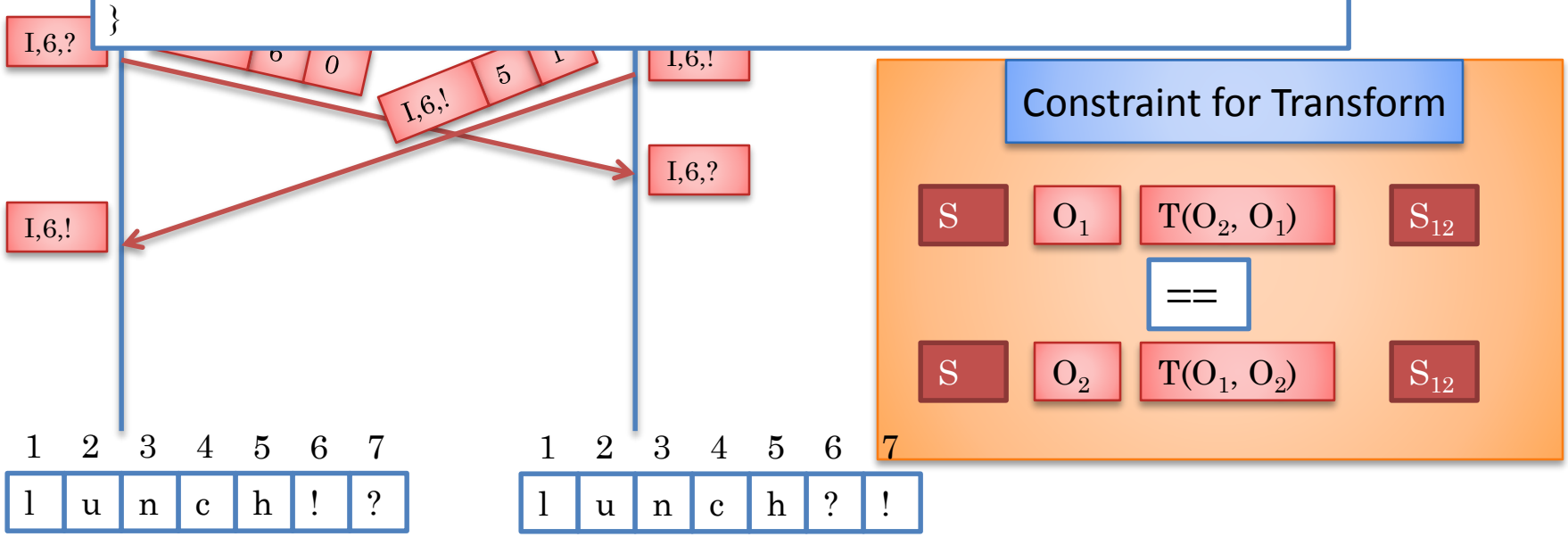
I,6,?	6	0

Local Buffer$_2$

I,6,!	5	1

Insertion at same index: Error

```
Operation TransformInsertInsert (InsertOperation R, InsertOperation L) {
        Operation R^T = R.deepCopy();
        if (R.index > L.index)
                R^T.index = R.index + 1
        return R^T ;
}
```



Constraint for Transform

S | O_1 | $T(O_2, O_1)$ | S_{12}

==

S | O_2 | $T(O_1, O_2)$ | S_{12}

Local Buffer$_1$

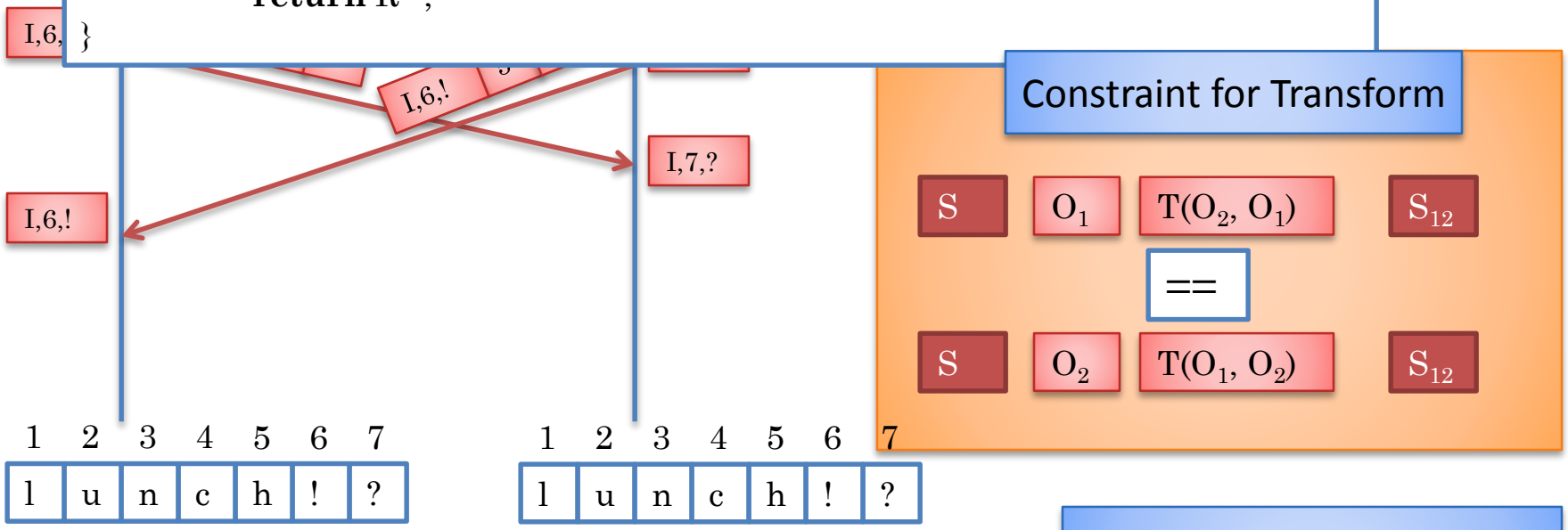
I,6,?	6	0

Local Buffer$_2$

I,6,!	5	1

Priority-Based

```
InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L) {
        Operation R^T = deepClone();
        if ((R.index > L.index) | |
           (R.index === L.index  &&  R.id  < L.id))
                R^T.index = R.index + 1;
        return R^T ;
}
```



Constraint for Transform

S	O_1	$T(O_2, O_1)$	S_{12}
==			
S	O_2	$T(O_1, O_2)$	S_{12}

Local Buffer$_1$

I,6,?	6	0

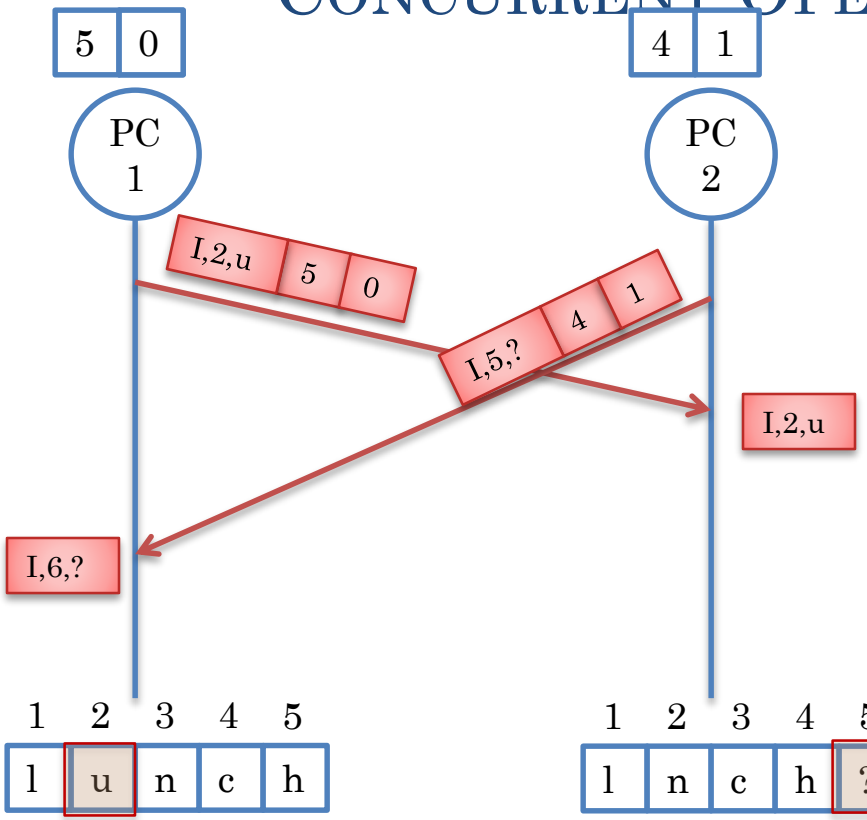
Local Buffer$_2$

I,6,!	5	1

Insert concurrent text at some position in order of priority

Can two remote operations be transformed wrt to the same local operation?

Control Algorithm: Single Local Concurrent Operation



Given Remote op, R, concurrent with exactly one local op L

R^T = Transform (R, L)

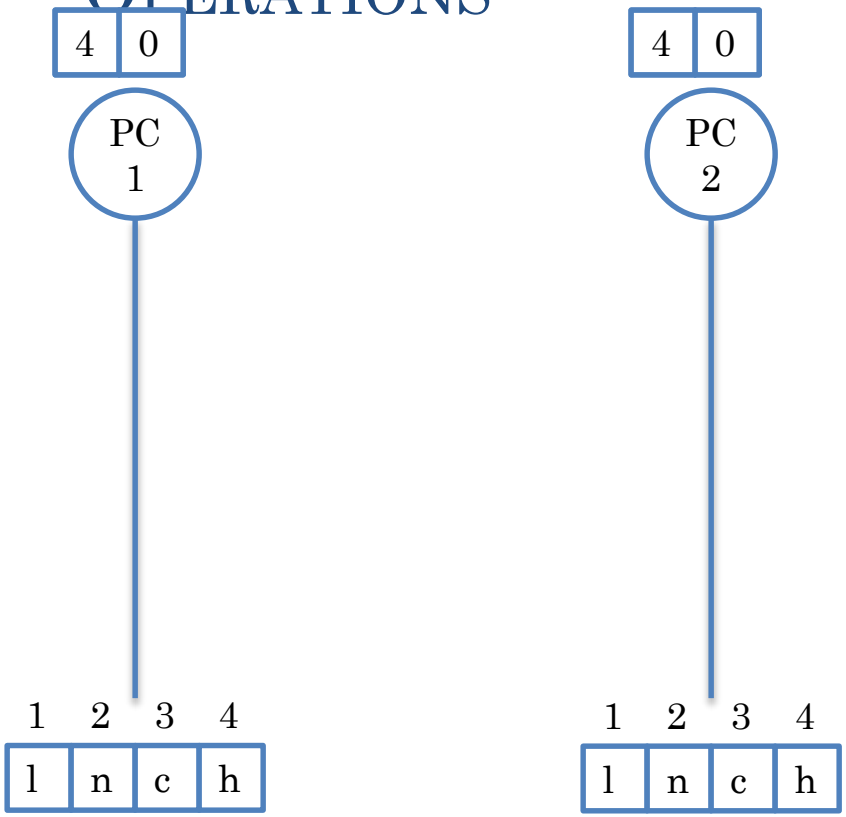
Execute R^T

Site.TimeStamp.increment(R.site)

Local Buffer$_1$

Local Buffer$_2$

Multiple Transformed Remote Concurrent Operations

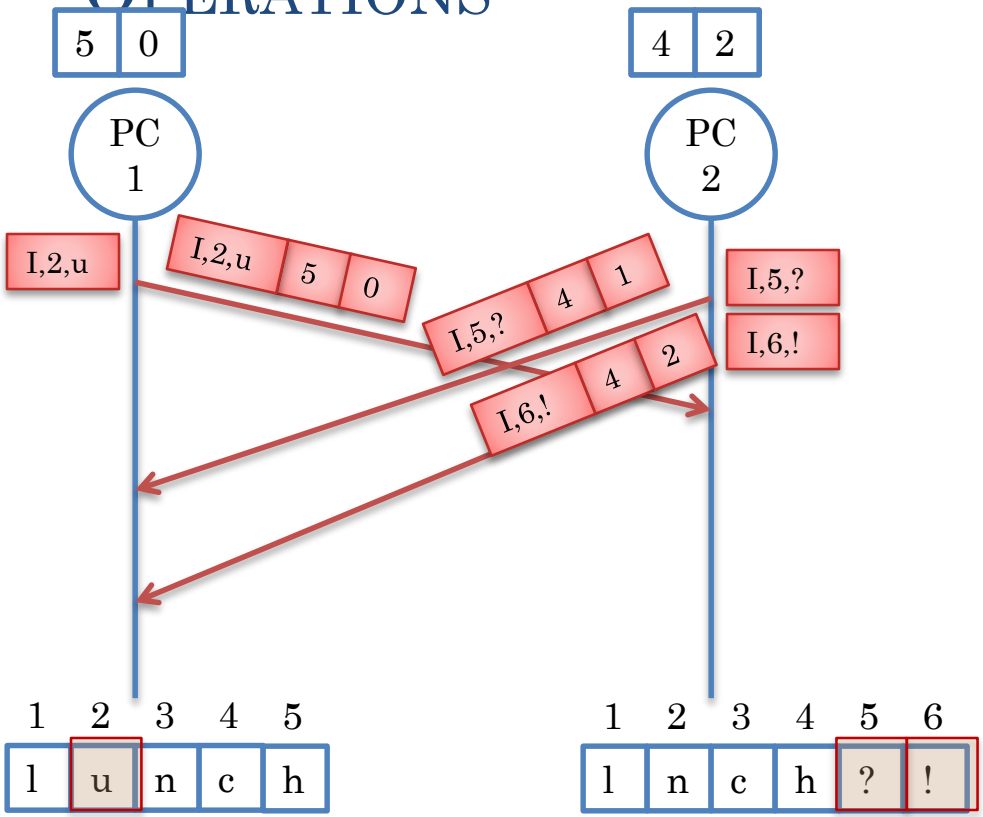


A remote site can execute multiple operations that are concurrent wrt to local buffer

Local Buffer$_1$

Local Buffer$_2$

Multiple Transformed Remote Concurrent Operations



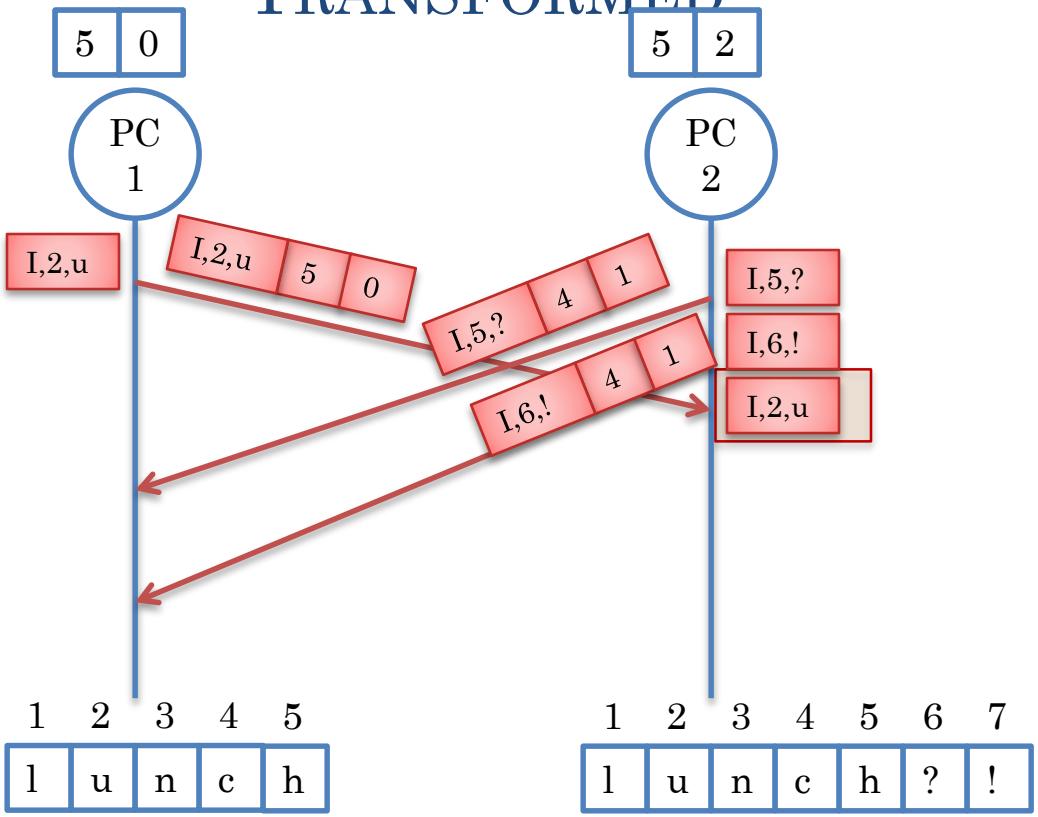
Local Buffer$_1$

I,2,u	5	0

Local Buffer$_2$

I,5,?	4	1
I,6,!	4	2

Site 1 Operation Arrives and is not Transformed



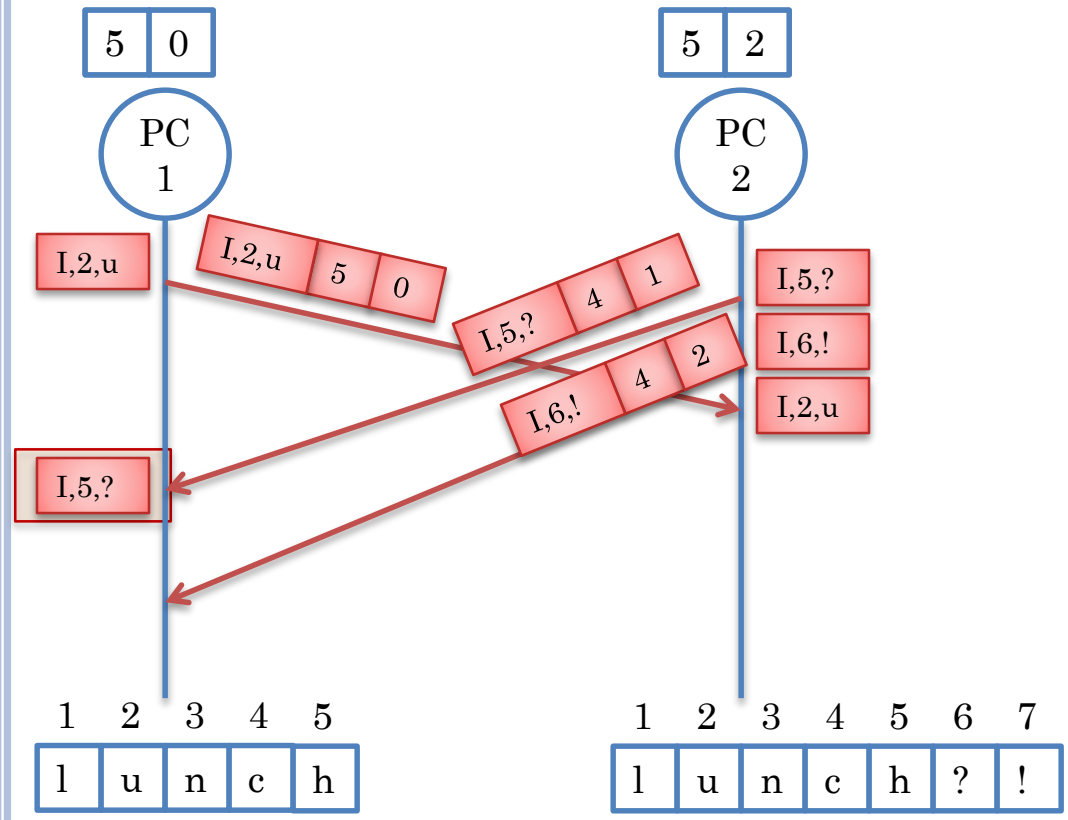
Local Buffer$_1$

I,2,u	5	0

Local Buffer$_2$

I,5,?	4	1
I,6,!	4	2

First Site 2 Operation Arrives



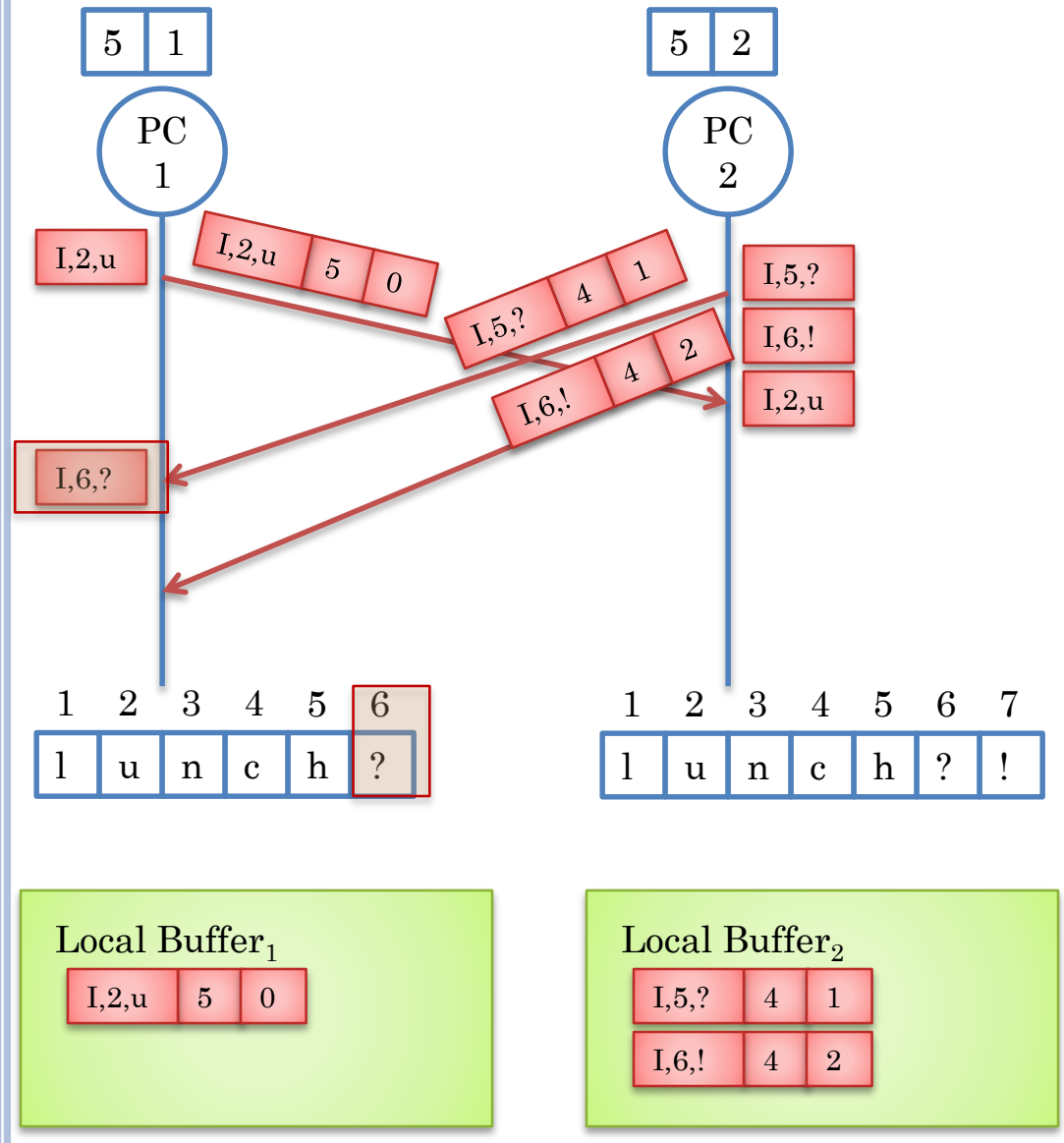
Local Buffer$_1$

I,2,u	5	0

Local Buffer$_2$

I,5,?	4	1
I,6,!	4	2

First Operation Transformed



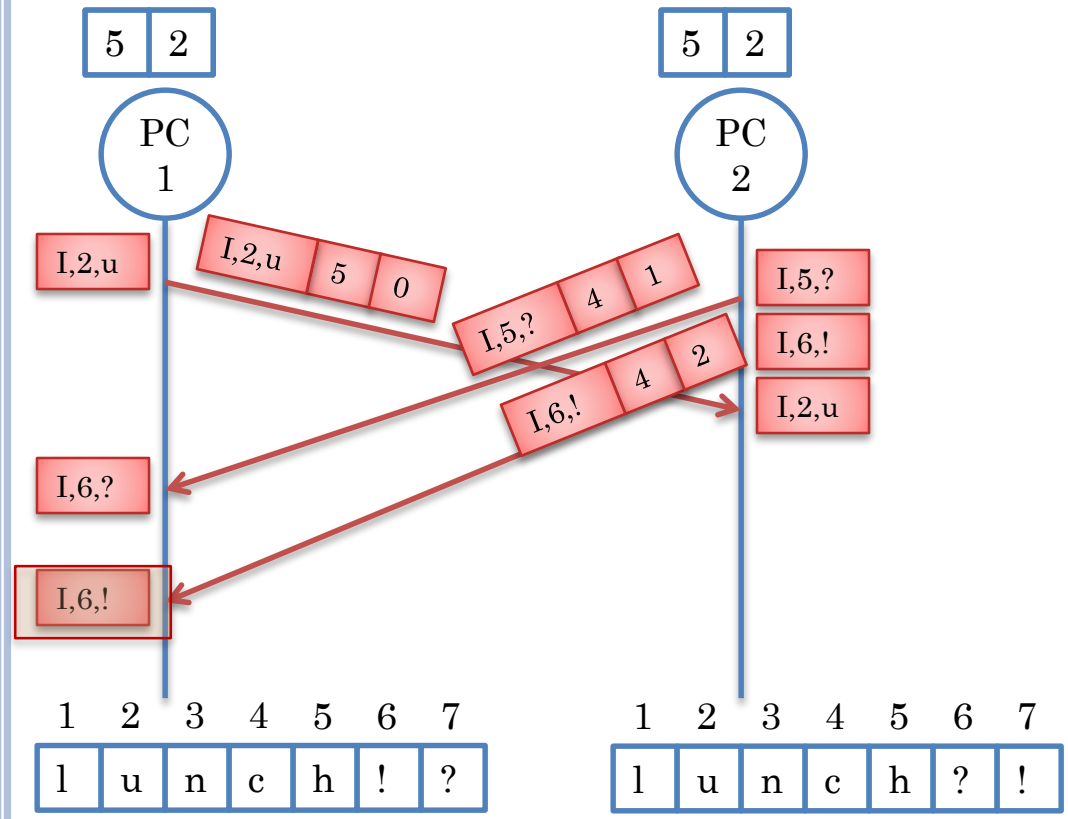
Local Buffer$_1$

I,2,u	5	0

Local Buffer$_2$

I,5,?	4	1
I,6,!	4	2

Second Operation Arrives



Multiple remote operations transformed with respect to same local operation

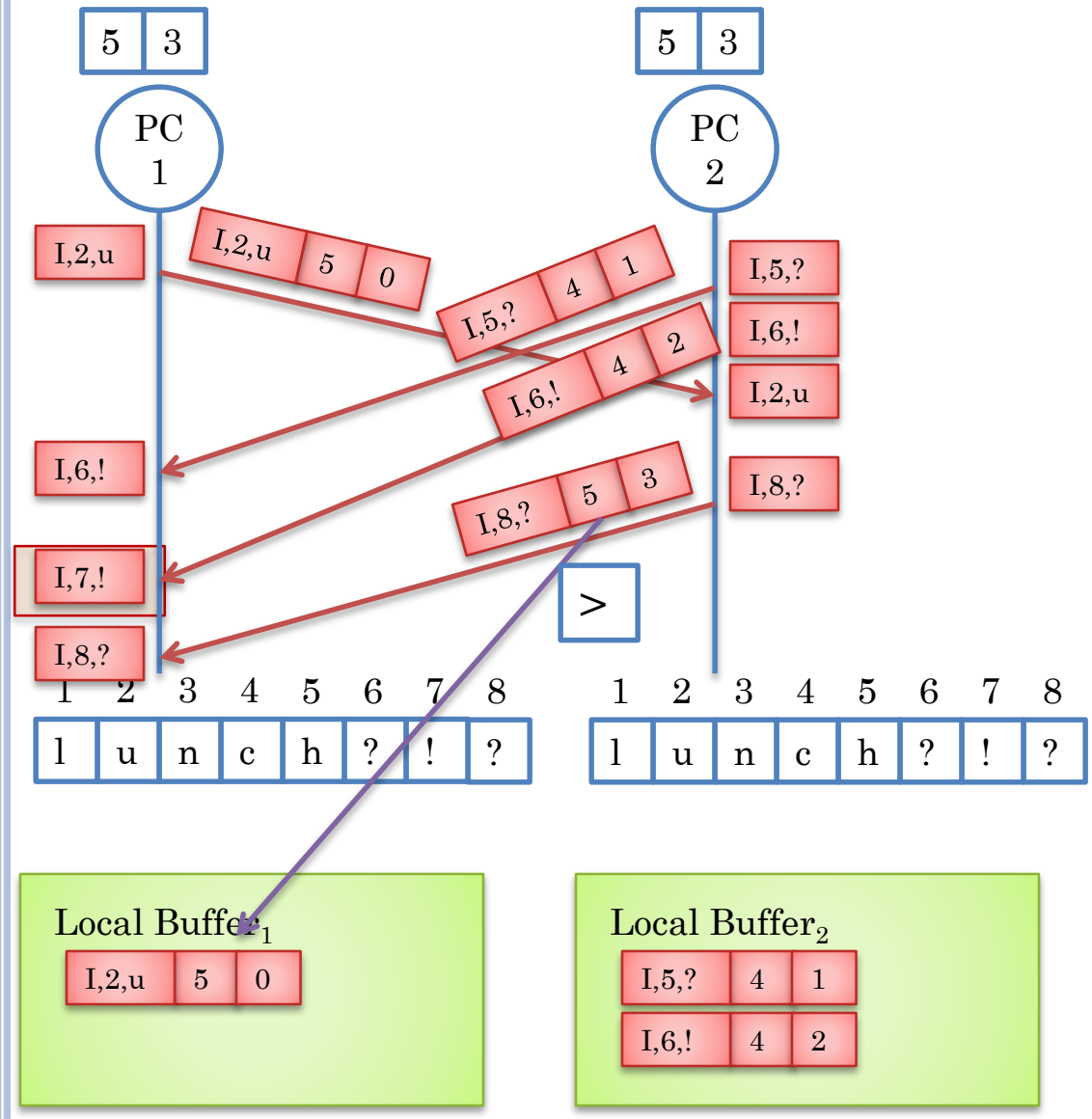
Local Buffer$_1$

I,2,u	5	0

Local Buffer$_2$

I,5,?	4	1
I,6,!	4	2

Buffer Cleanup



Multiple remote operations transformed with respect to same local operation

How long should local buffer be kept?

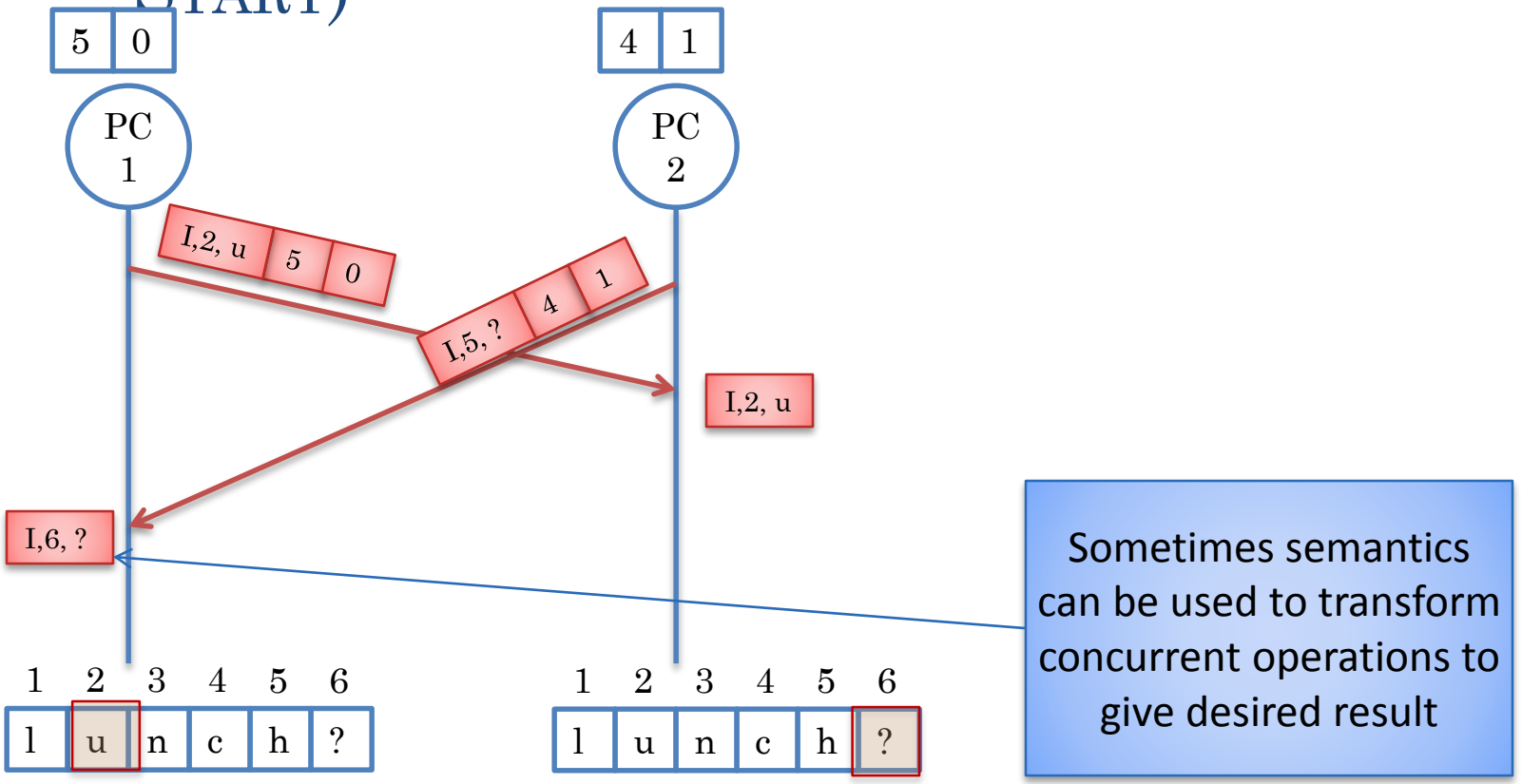
No need to transform

If local.timestamp < msg.timestamp

Each subsequent message has larger time stamp

Remove all locals from buffer with time stamp smaller than time stamp of received message

Transforming Remote Operation (Review Start)



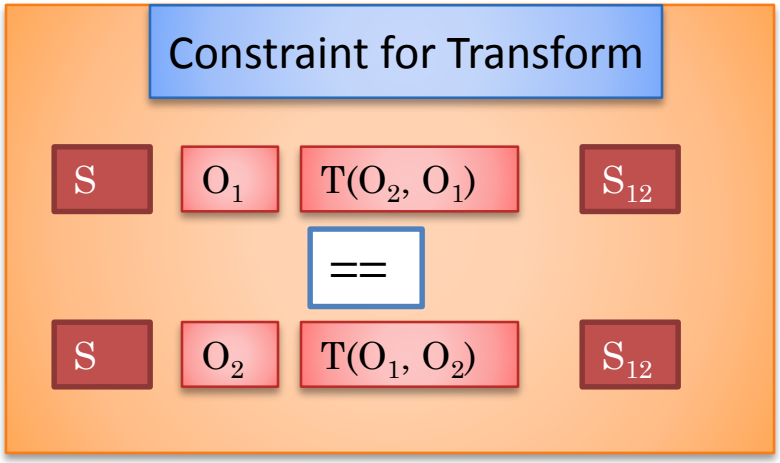
OT System Components

Transformation function: Handles single local and concurrent operations

Control algorithm: Calls transformation function, processed buffer and local time stamps

Both must be correct.

Priority-Based Transformation




```
InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L)  {
        Operation R^T = deepClone();
        if ((R.index > L.index) | |
           (R.index === L.index  &&  R.id  < L.id))
                R^T.index = R.index + 1;
        return R^T ;
}
```

Control Algorithm: Single Local Concurrent Operation

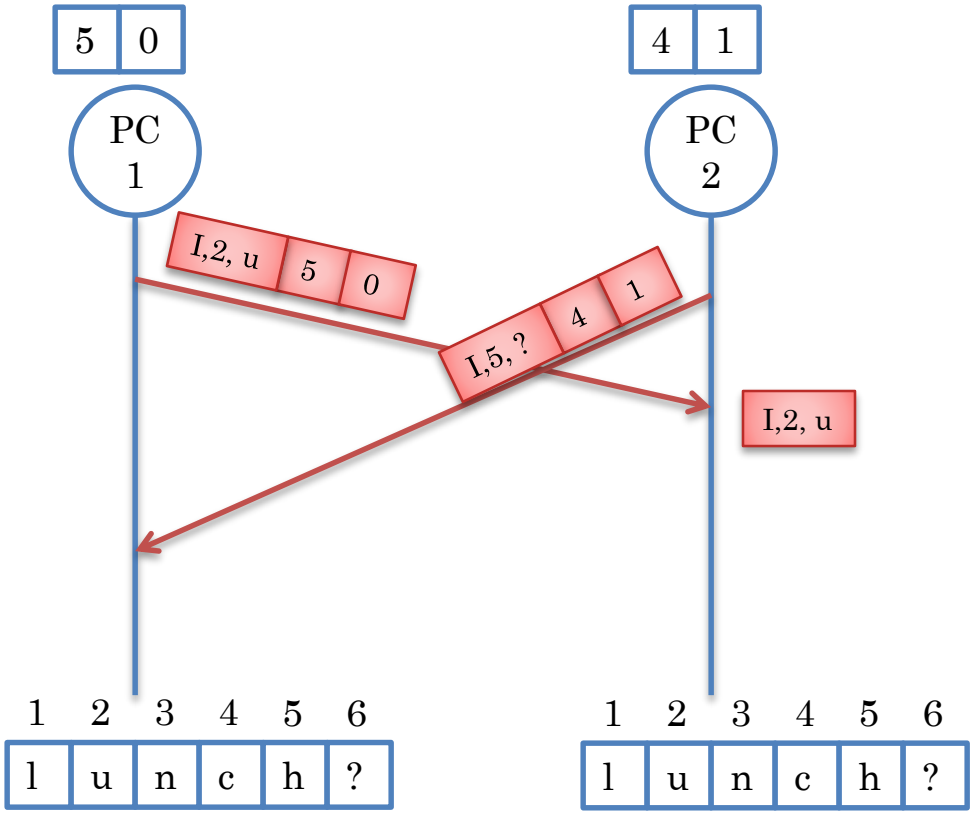
Given Remote op R, concurrent with exactly one local op L

R^T = Transform (R, L)

Execute R^T

Site.TimeStamp.increment(R.site)

Need For Local Buffer



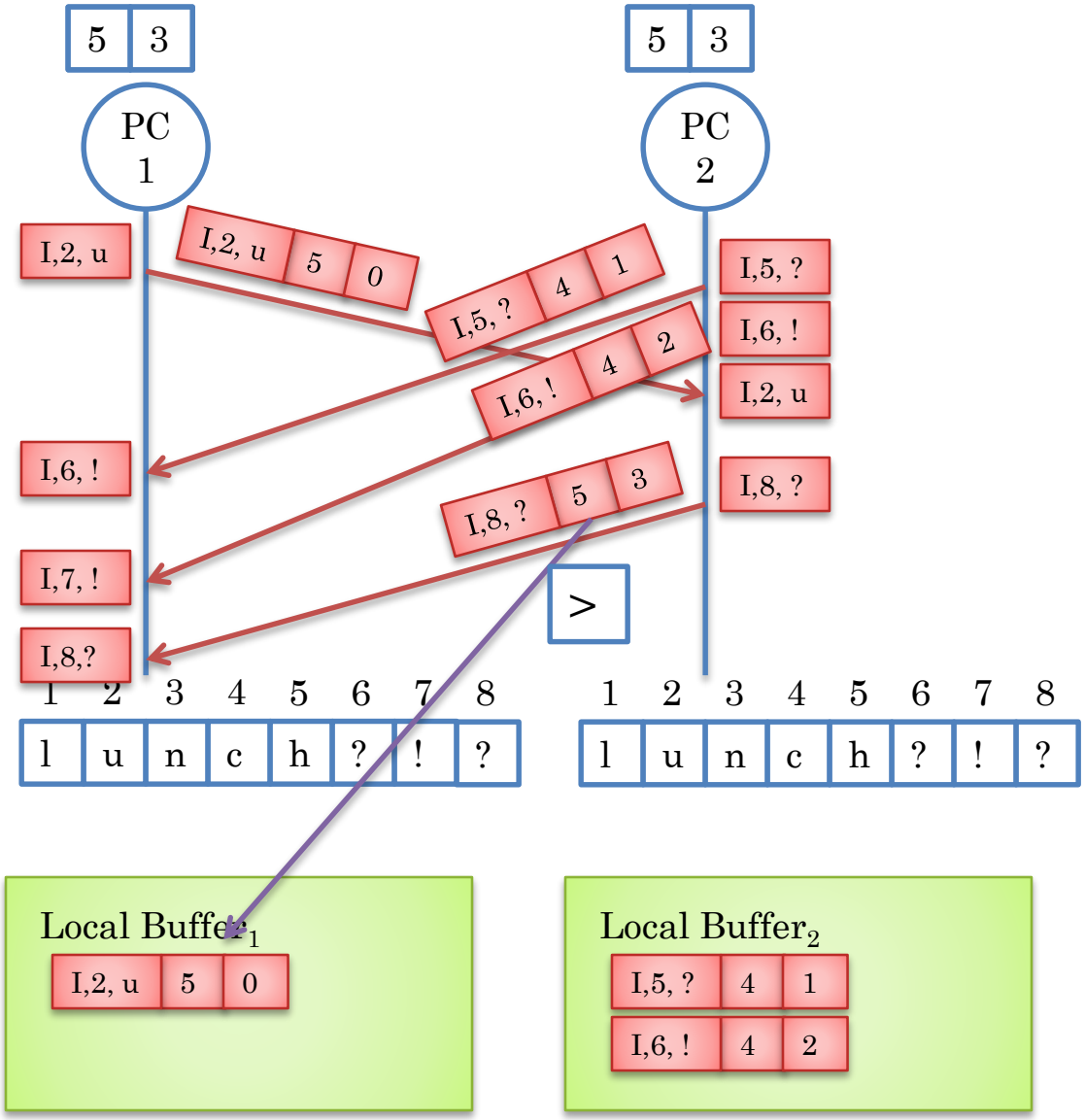
Local Buffer$_1$

I,2, u	5	0

Local Buffer$_2$

I,5, ?	4	1

Buffer Cleanup (Review End)



Multiple remote operations transformed with respect to same local operation

How long should local buffer be kept?

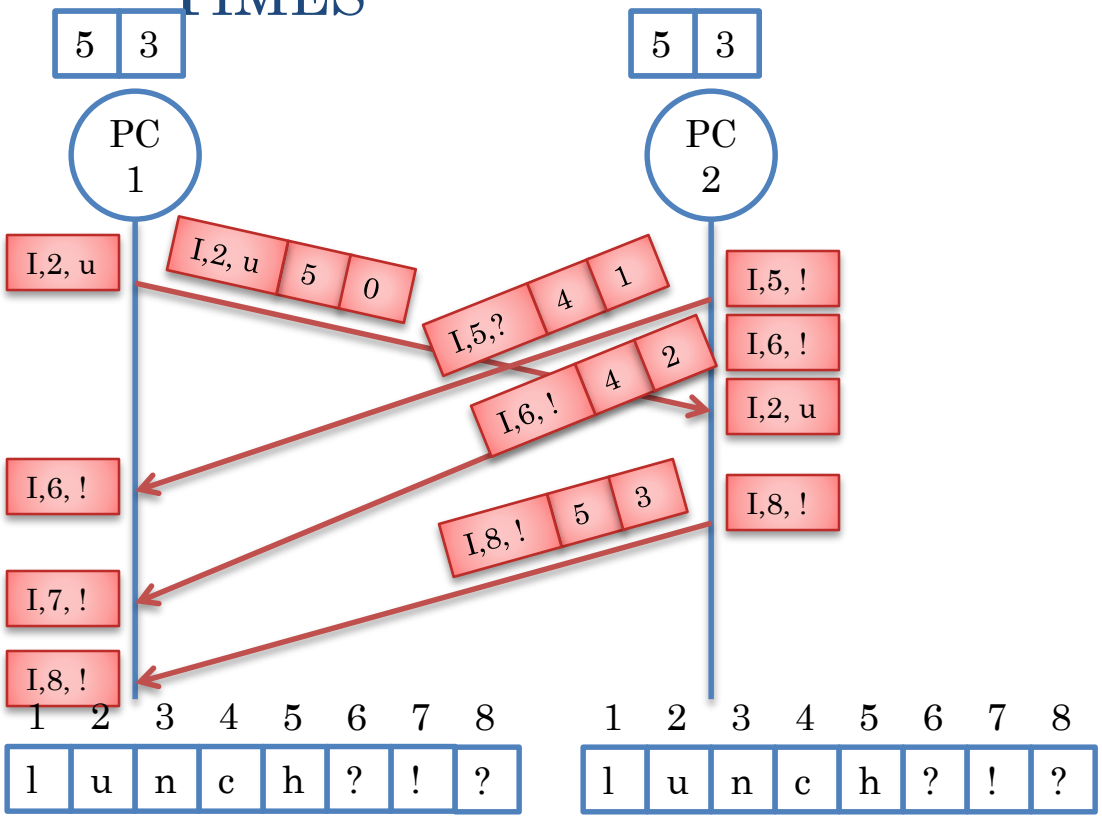
No need to transform

If local.timestamp < msg.timestamp

Each subsequent message has larger time stamp

Remove all locals from buffer with time stamp smaller than time stamp of received message

Dual: Operation Transformed Multiple Times



Multiple remote operations transformed with respect to same local operation

Dual?

A remote operation transformed with respect to multiple local operations

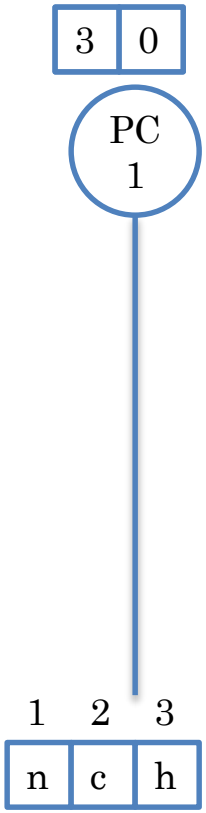
Local Buffer$_1$

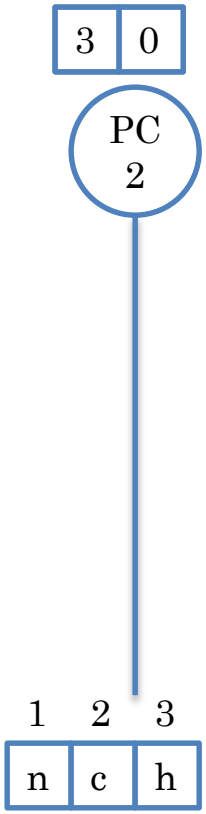
I,2, u	5	0

Local Buffer$_2$

I,5, ?	4	1
I,6, !	4	2

Initial State

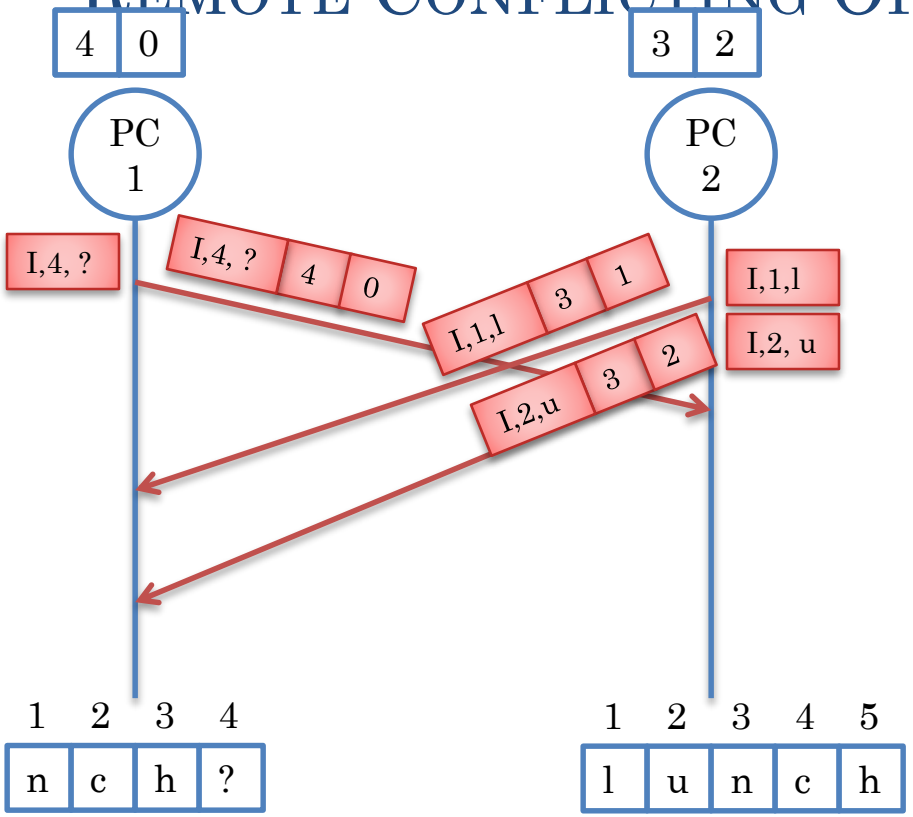




Local Buffer$_1$

Local Buffer$_2$

Multiple Concurrent Local Operations wrt Remote Conflicting Operation



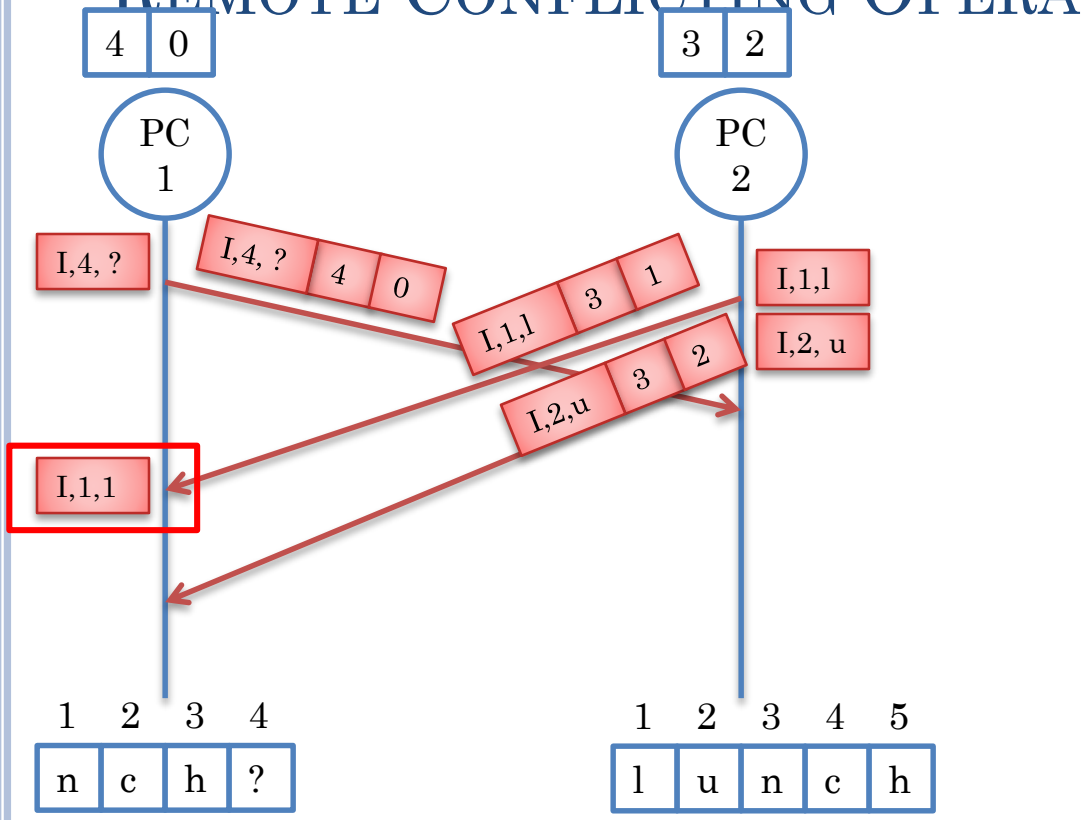
Local Buffer$_1$

I,4, ?	4	0

Local Buffer$_2$

I,1,l	3	1
I,2, u	3	2

Multiple Concurrent Local Operations wrt Remote Conflicting Operation



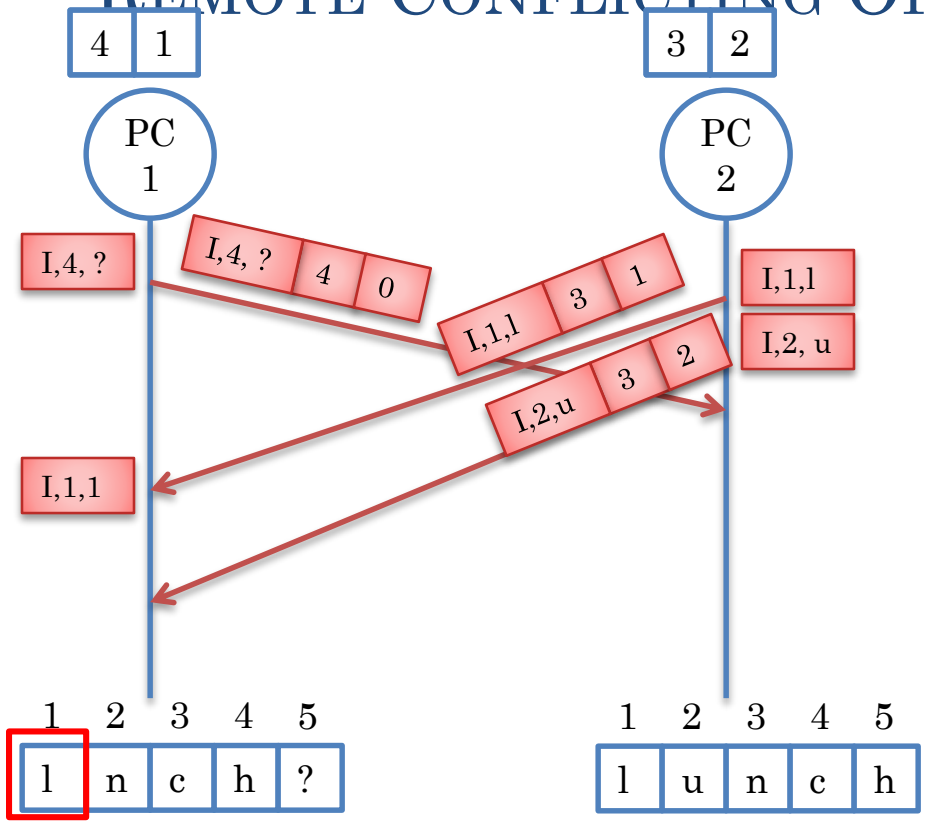
Local Buffer$_1$

I,2, u	4	0

Local Buffer$_2$

I,1,l	3	1
I,2, u	3	2

Multiple Concurrent Local Operations wrt Remote Conflicting Operation



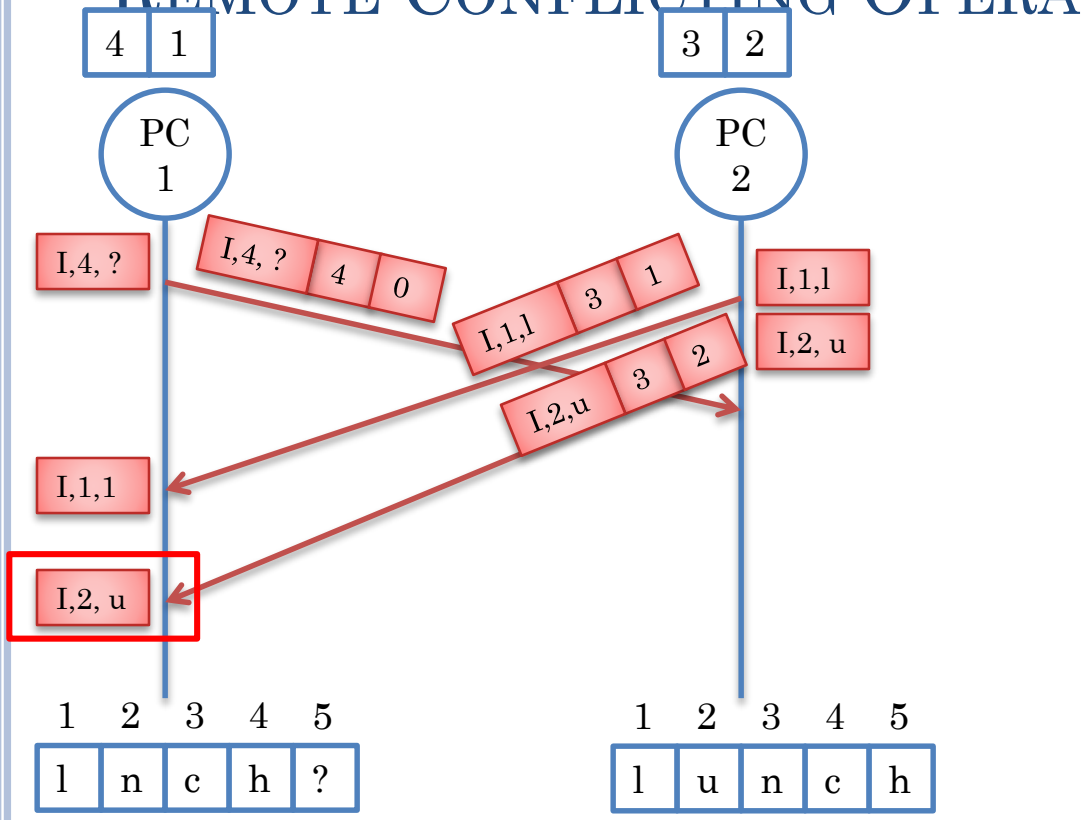
Local Buffer$_1$

I,4, ?	4	0

Local Buffer$_2$

I,1,l	3	1
I,2, u	3	2

Multiple Concurrent Local Operations wrt Remote Conflicting Operation



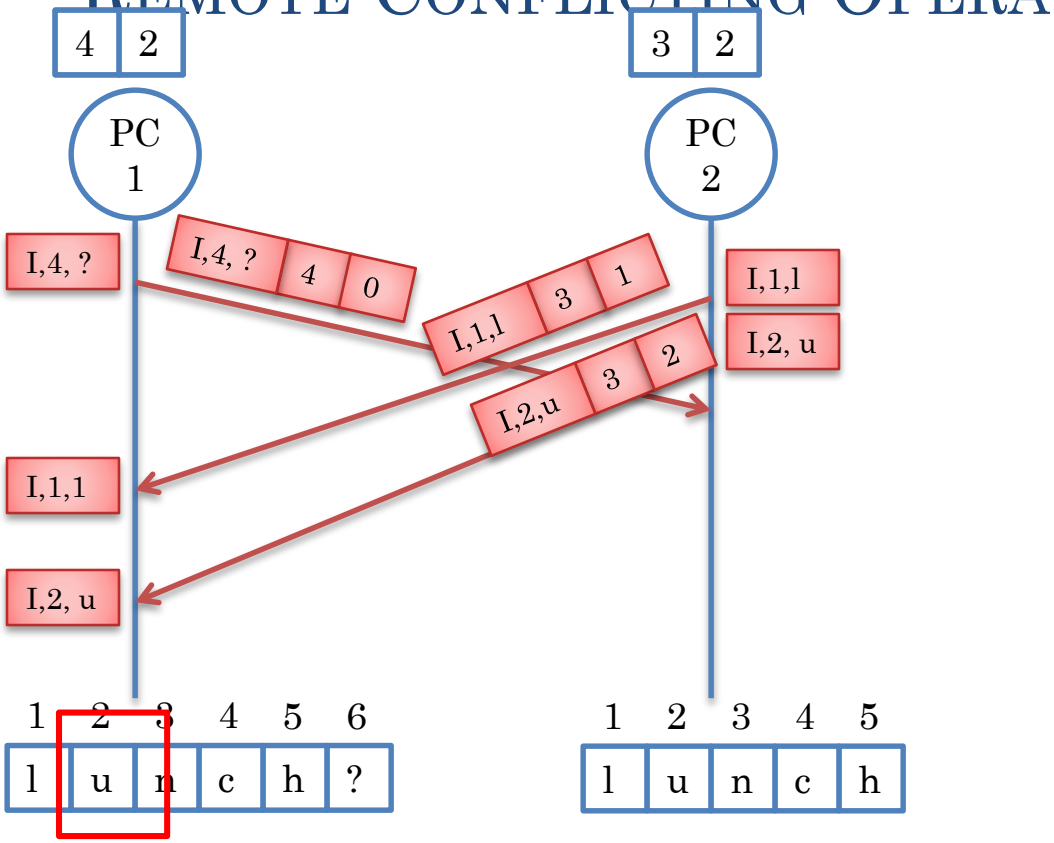
Local Buffer$_1$

I,4, ?	4	0

Local Buffer$_2$

I,1,l	3	1
I,2, u	3	2

Multiple Concurrent Local Operations wrt Remote Conflicting Operation



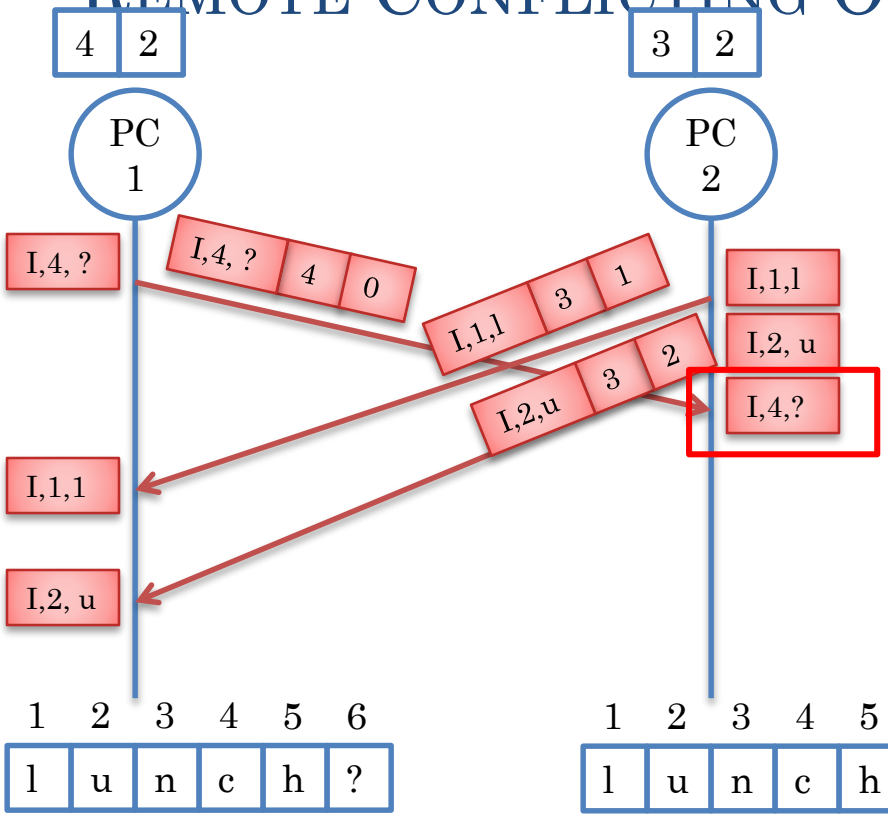
Local Buffer$_1$

I,4, ?	3	0

Local Buffer$_2$

I,1,l	3	1
I,2, u	3	2

Multiple Concurrent Local Operations wrt Remote Conflicting Operation



Local Buffer$_1$

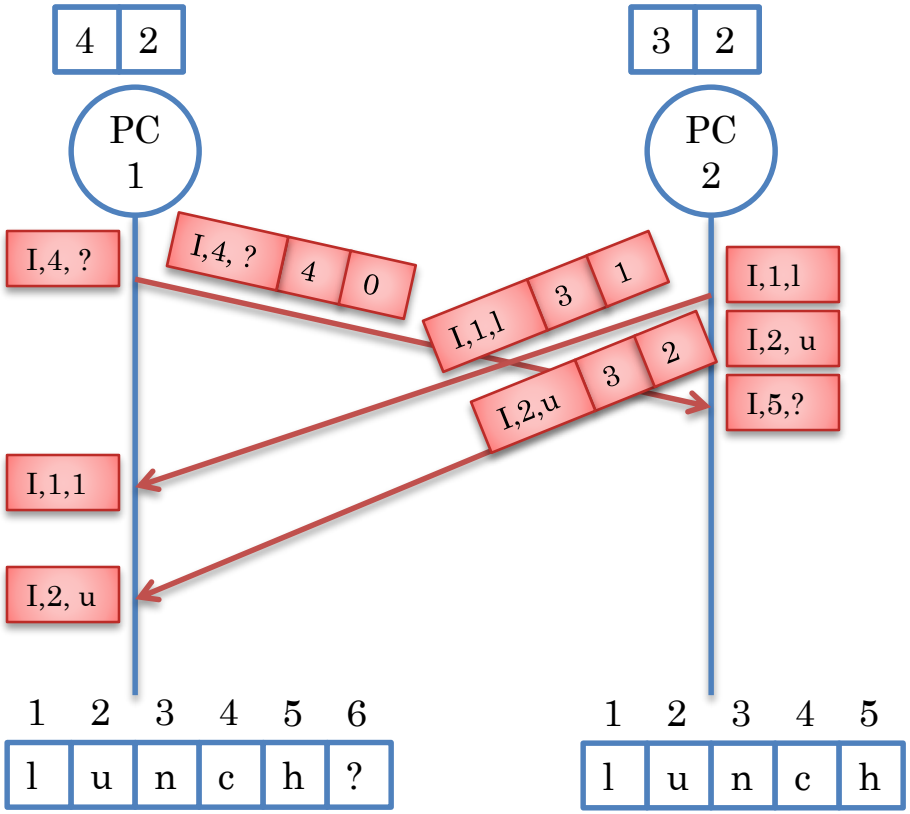
I,4, ?	3	0

Local Buffer$_2$

I,1,l	3	1
I,2, u	3	2

First Transformation

Transform wrt to first concurrent local operation



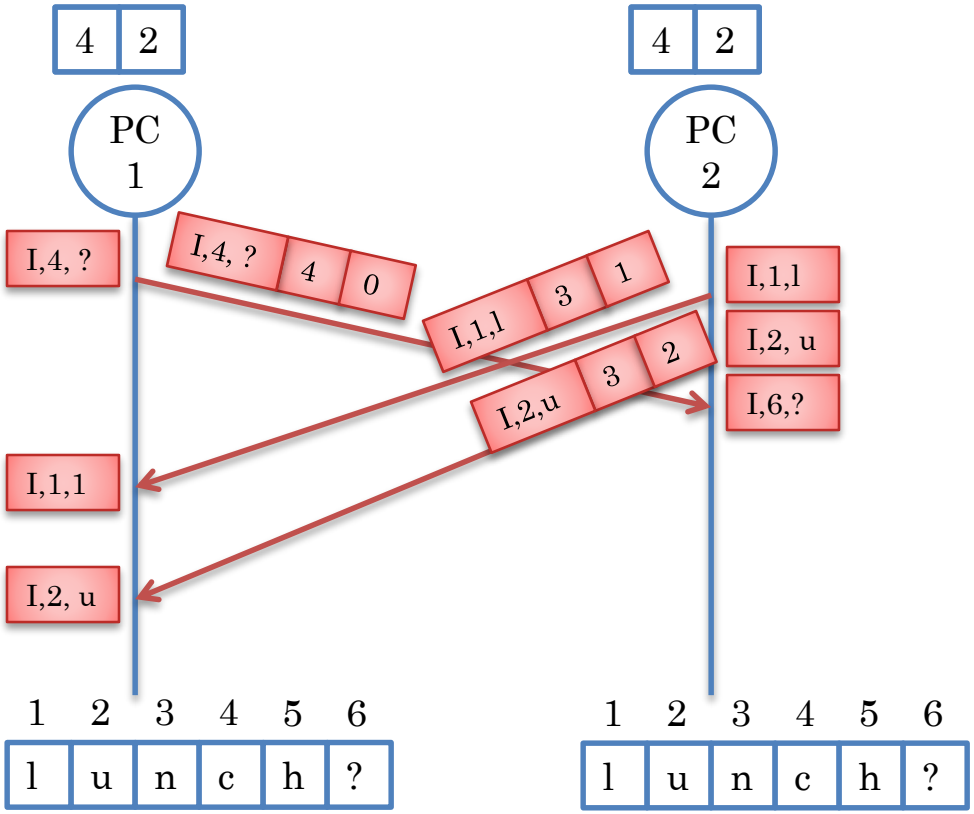
Local Buffer$_1$

I,4, ?	3	0

Local Buffer$_2$

I,1,1	3	1
I,2, u	3	2

Second Transformation/Control Algorithm



Local Buffer$_1$

I,4, ?	3	0

Local Buffer$_2$

I,1,1	3	1
I,2, u	3	2

Transform wrt to second concurrent local operation

A remote operation transformed with respect to multiple local operations

Run transform function with respect to all concurrent operations in the local log: Transform (Transform (Transform (R, L1), L2) …LN)

Control algorithm now handles multiple concurrent/local operations using Transform function addressing single concurrent remote/local operation

Control Algorithm: Single Local Concurrent Operation

Given Remote op, R, concurrent with exactly one local op L

R^T = Transform (R, L)

Execute R^T

Site.TimeStamp.increment(R.site)

Algorithm for multiple local concurrent operations?

Control Algorithm: Multiple Concurrent Local Operations

Given Remote op, R, concurrent with local ops L^1, L^2, .. L^N

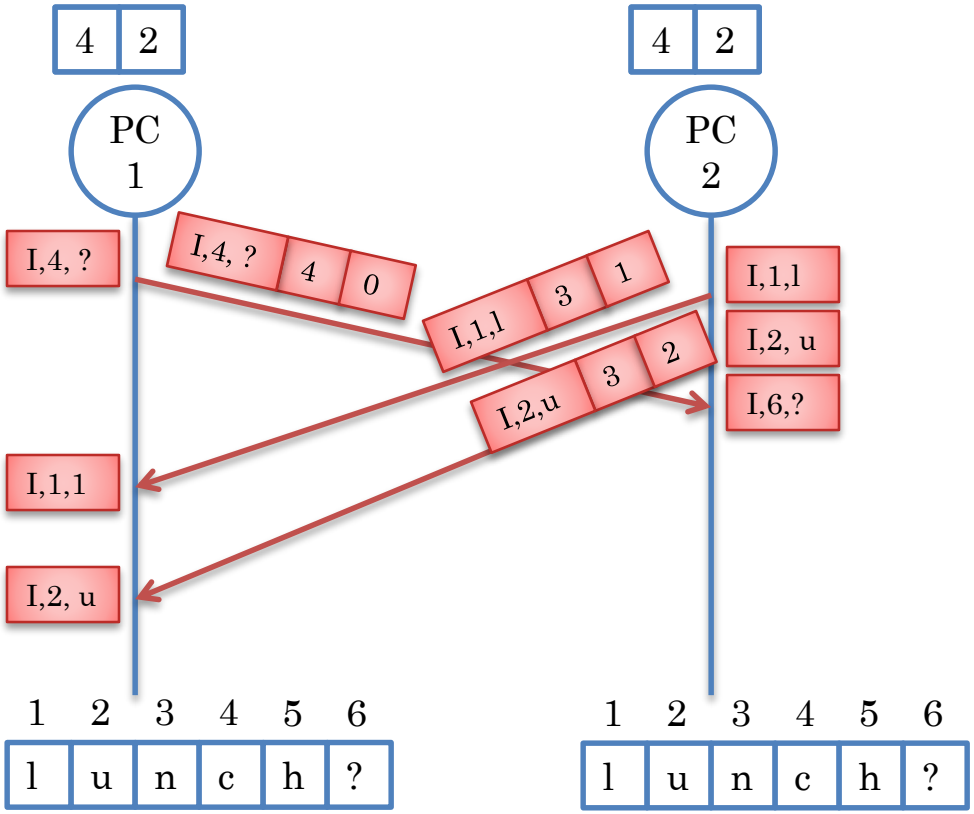
For each L

R = Transform (R, L)

Execute R

Site.TimeStamp.increment(R.site)

Single Site Transformation



All examples so far involved transformation(s) at one site

Transformation at both sites?

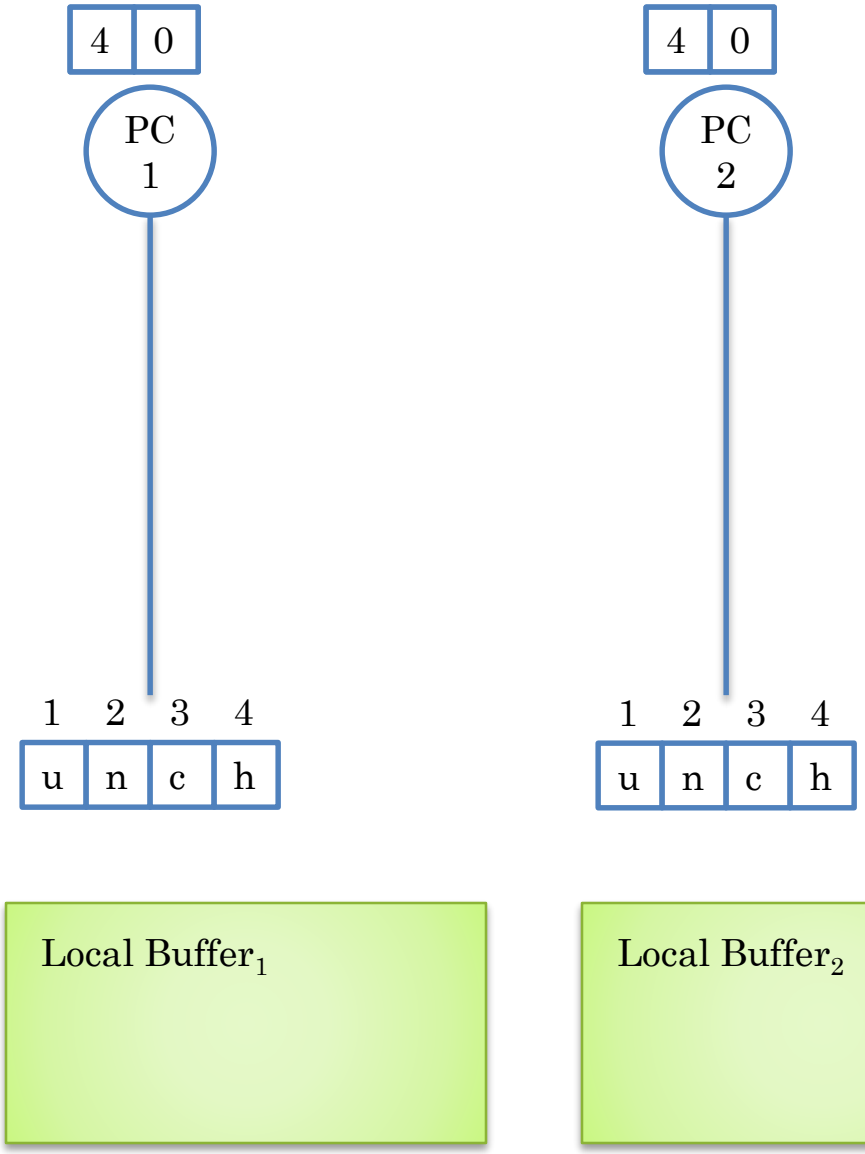
Local Buffer$_1$

I,4, ?	3	0

Local Buffer$_2$

I,1,1	3	1
I,2, u	3	2

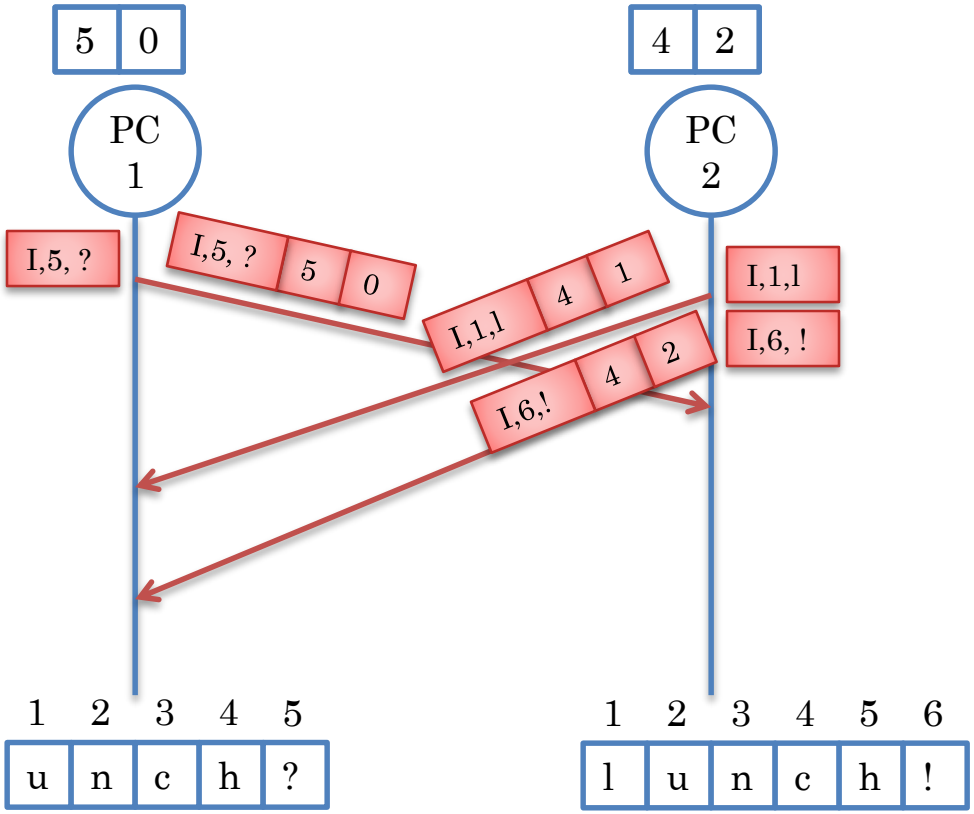
Initial State



Local Buffer$_1$

Local Buffer$_2$

Multiple Concurrent Remote Operations



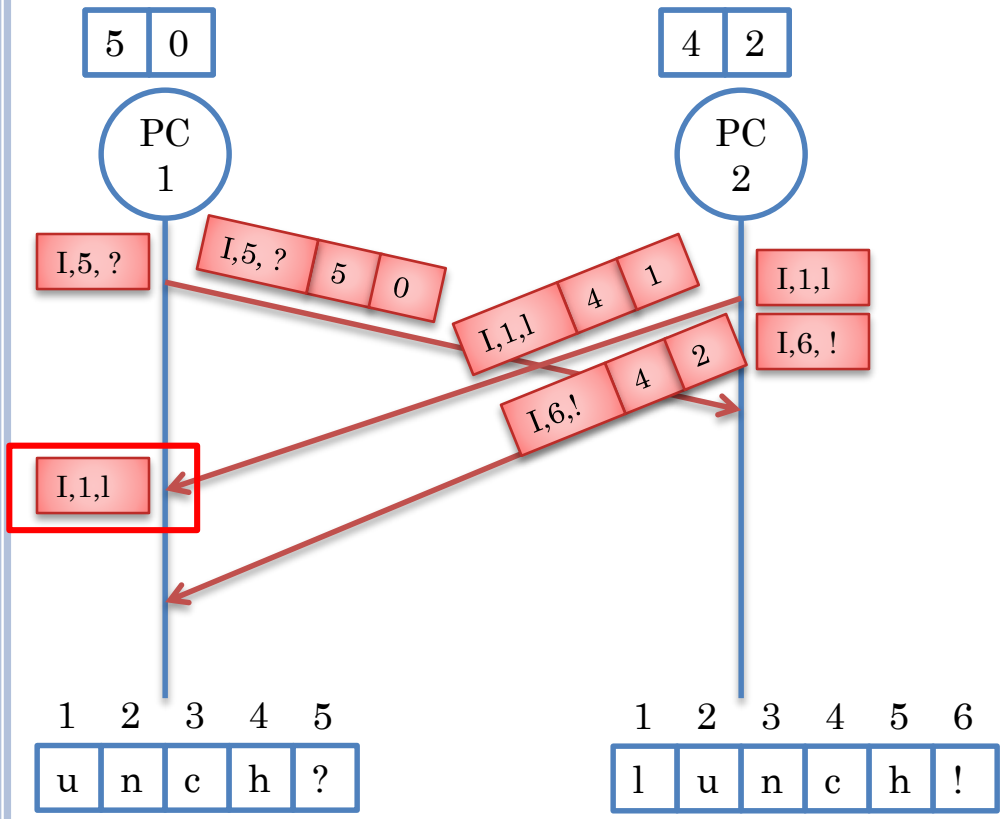
Local Buffer$_1$

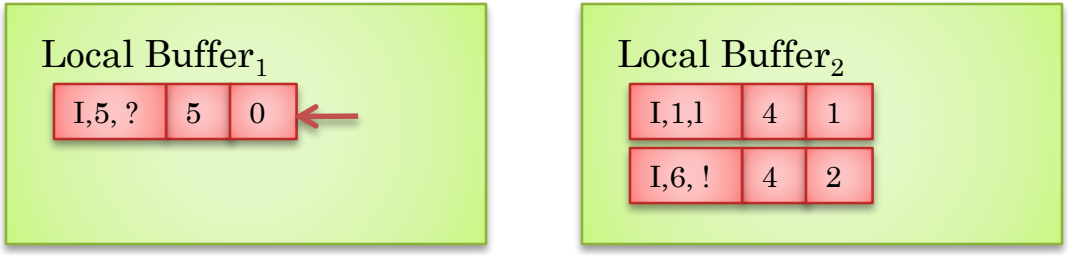
I,5, ?	5	0

Local Buffer$_2$

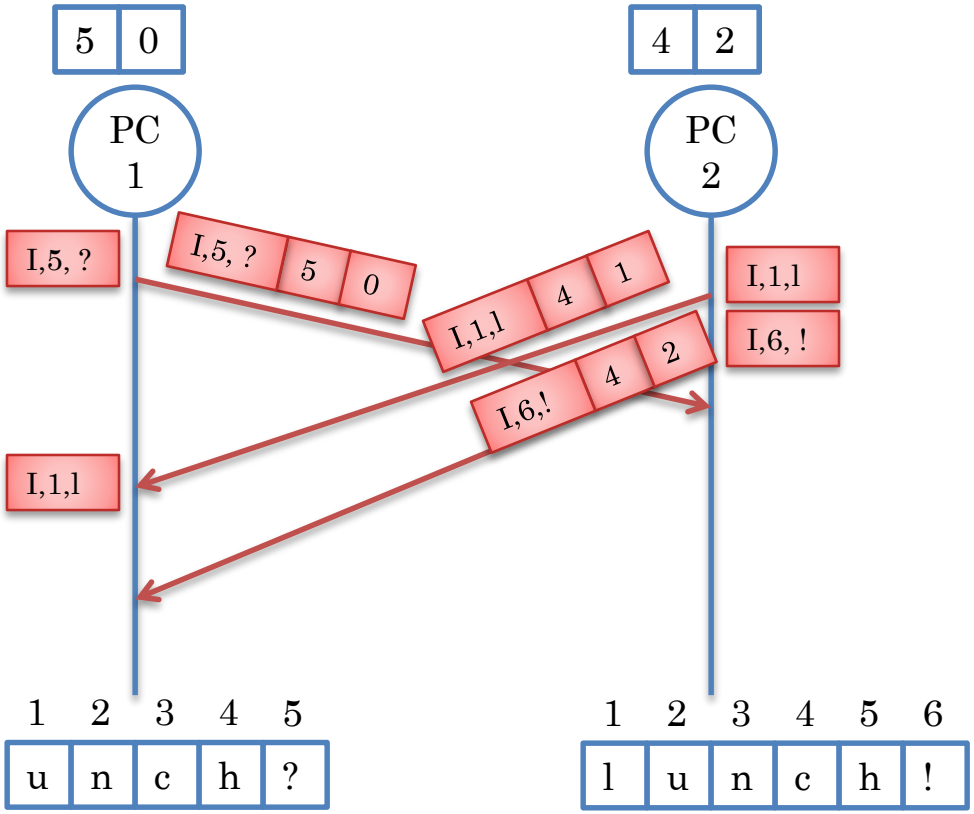
I,1,l	4	1
I,6, !	4	2

Arrival at Site 1





Result of Transform



Local Buffer$_1$

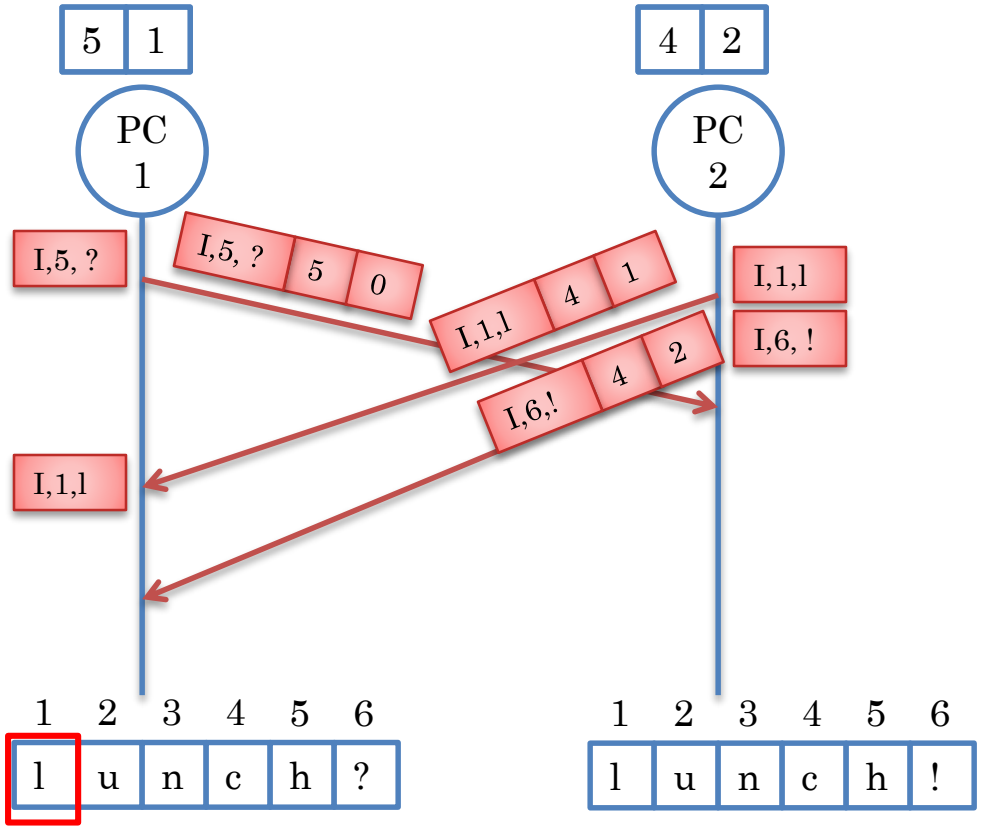
I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Remote operation not transformed

Untransformed Application



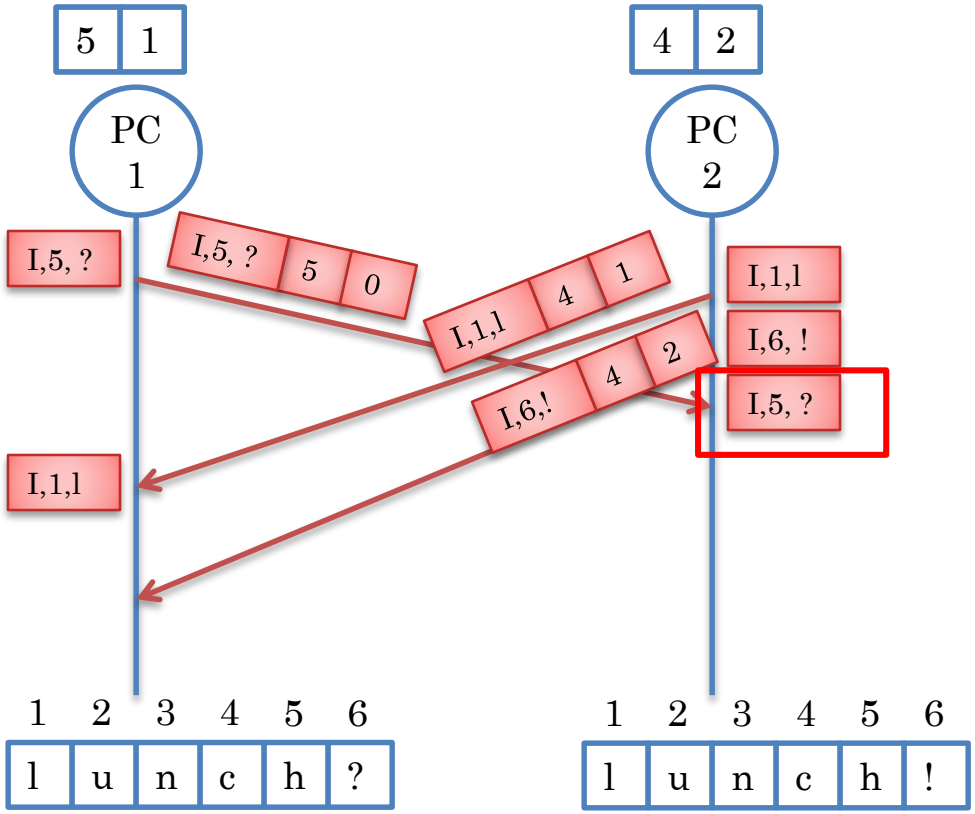
Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Arrival at second site



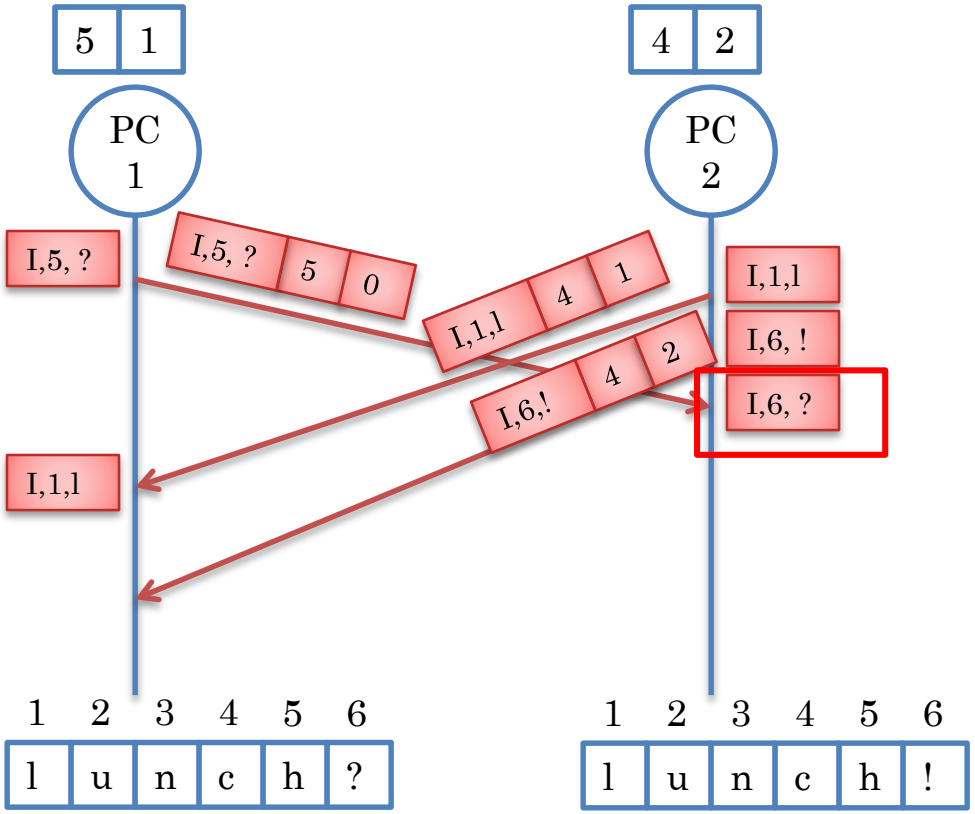
Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Transformed wrt to First Local Operation



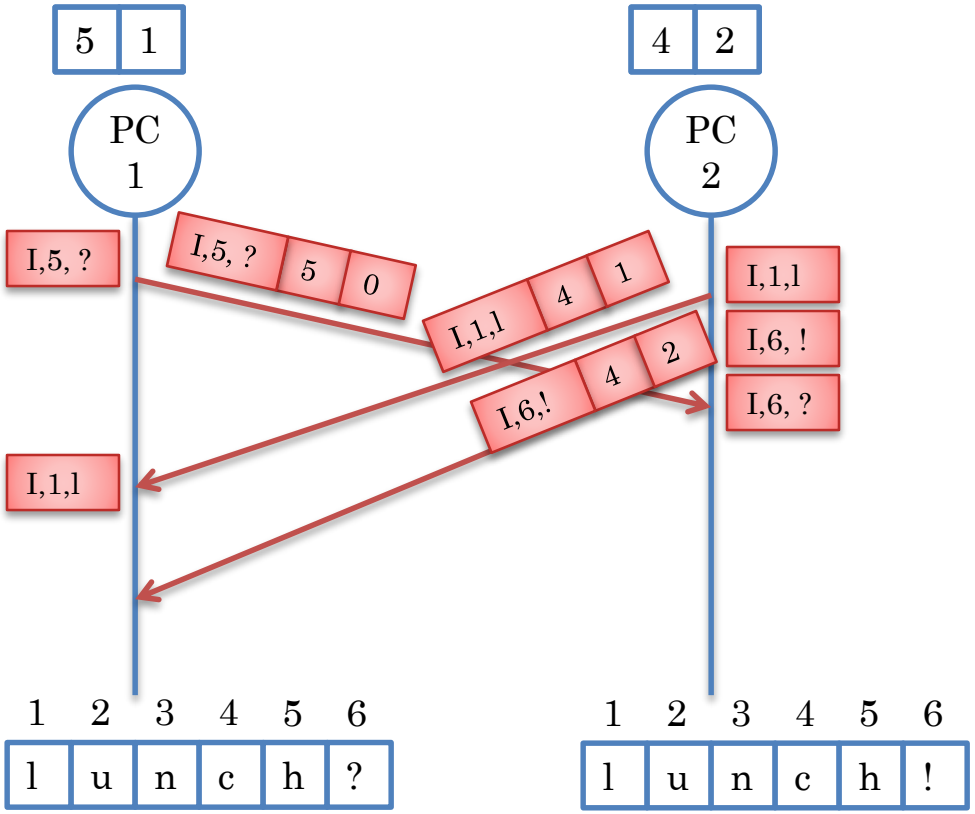
Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Examining Second Local Operation



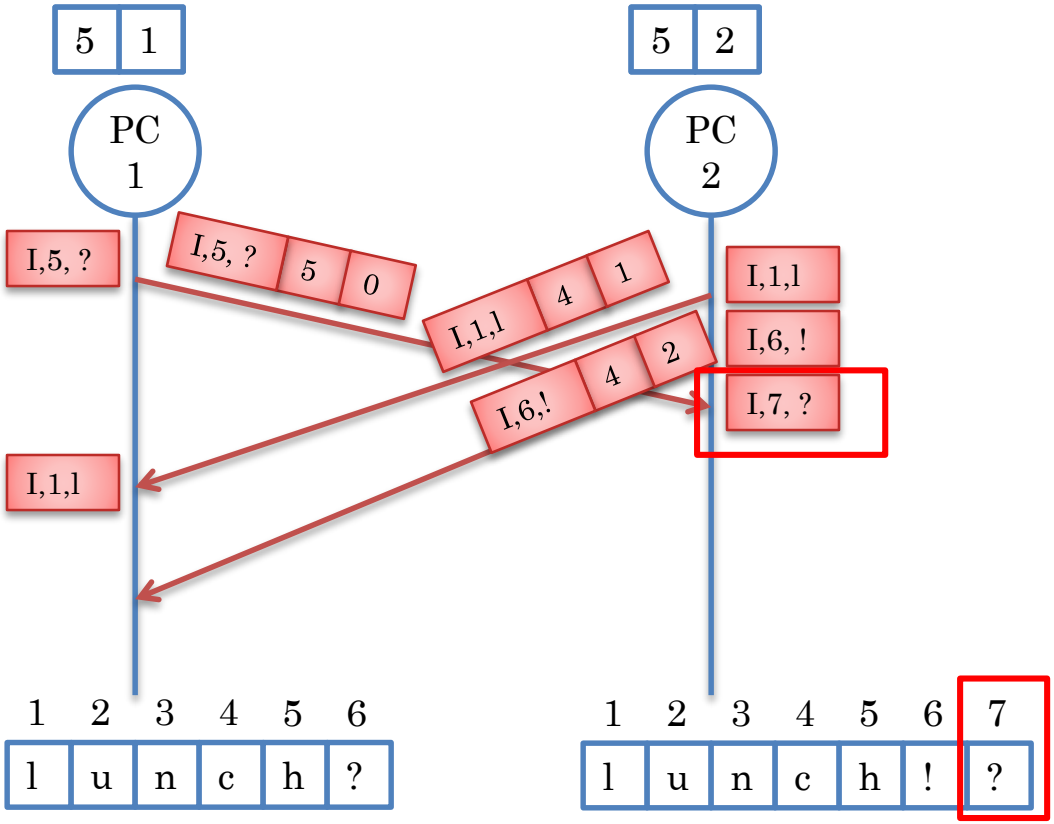
Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Second Transformation and Application



Local Buffer$_1$

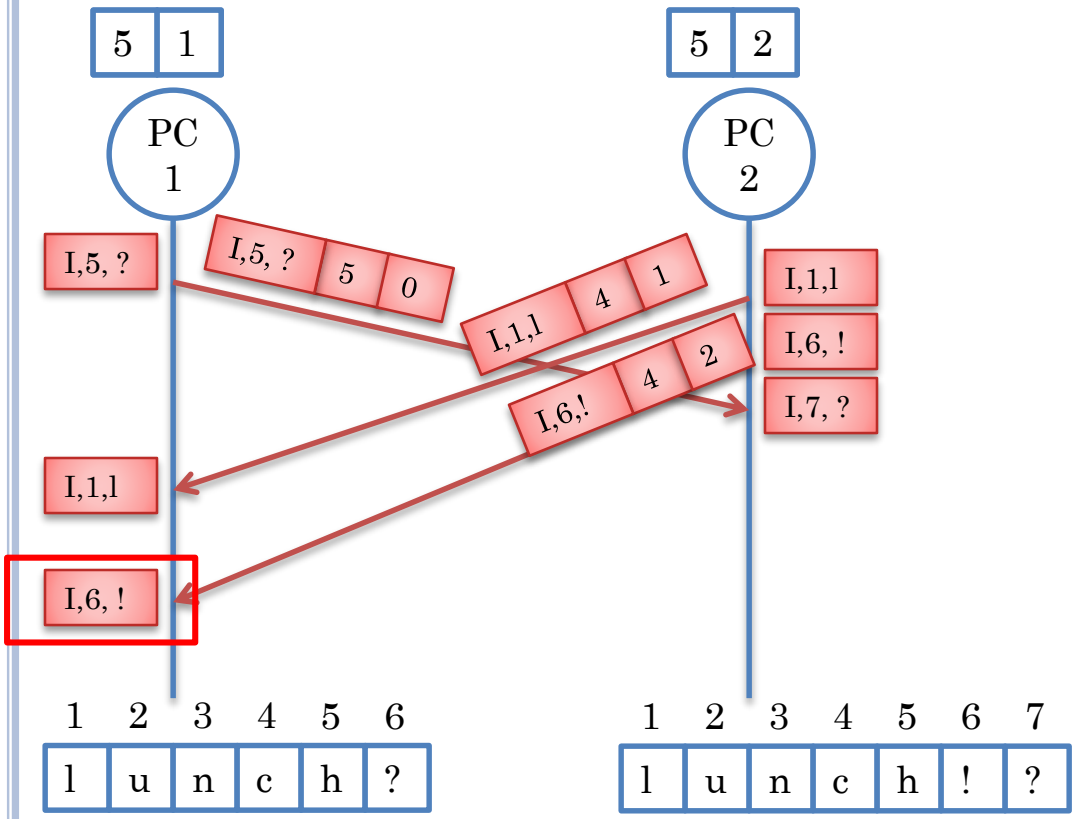
I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

User 1 < User 2

Second Arrival at Site 1



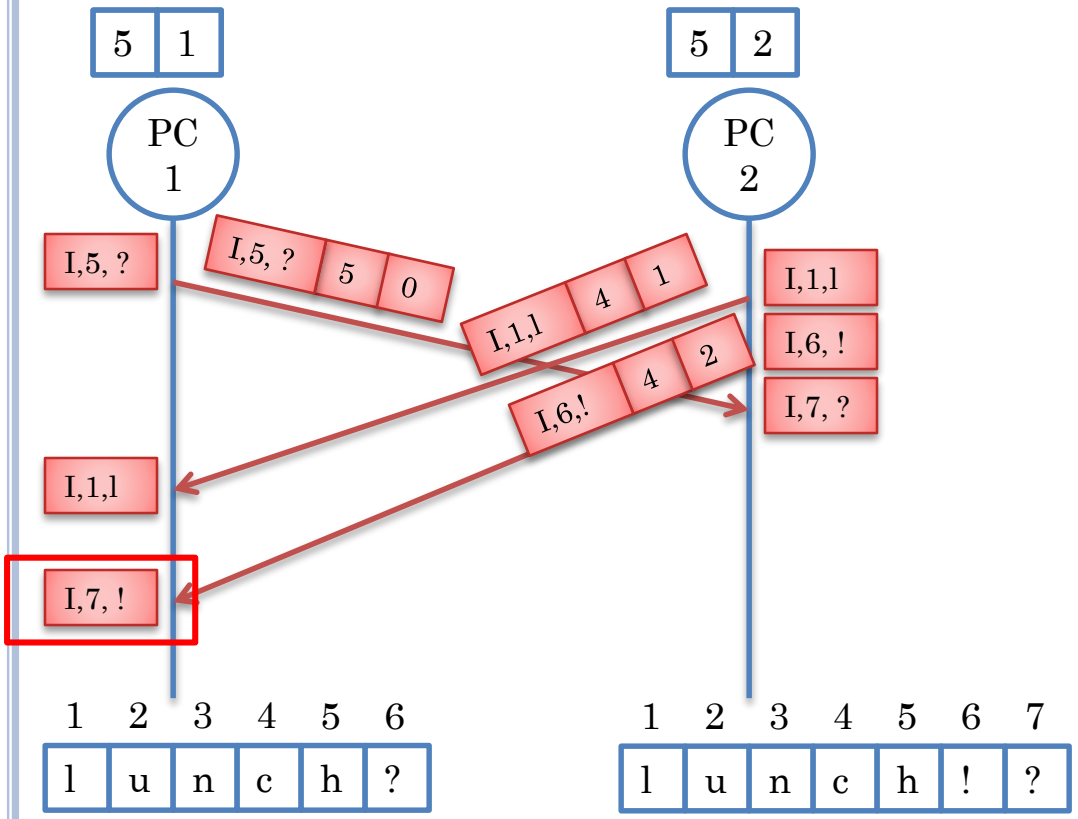
Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Operation Transformed



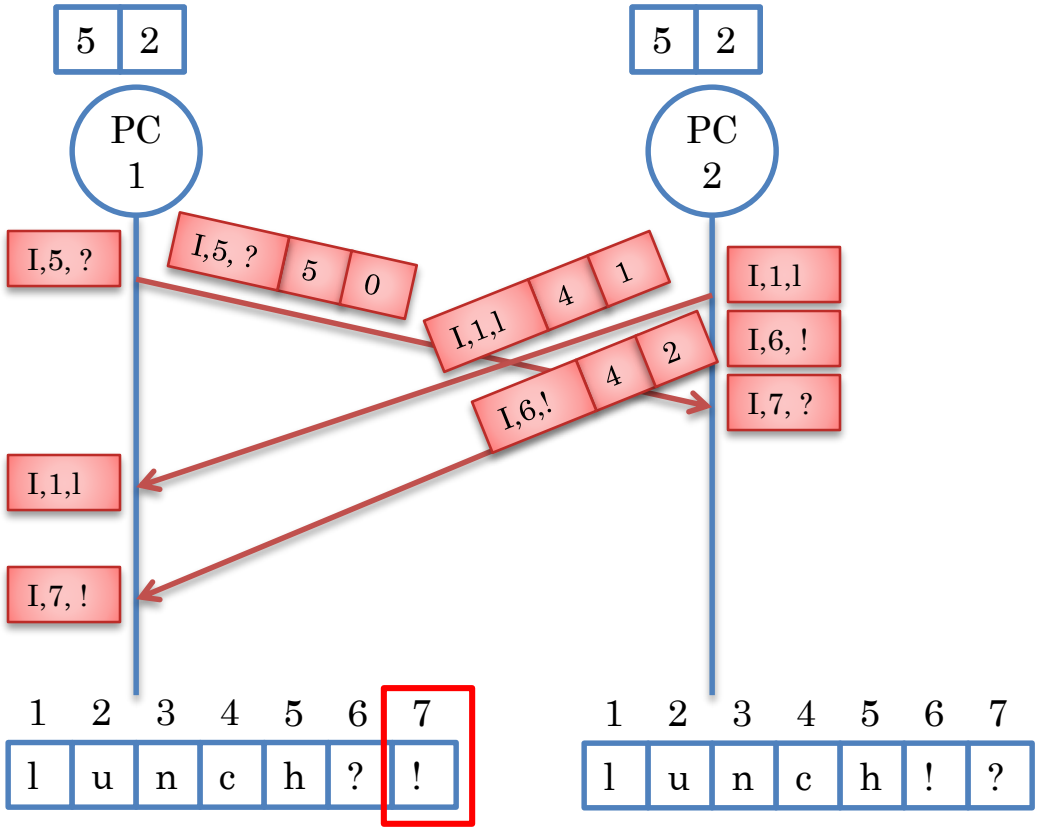
Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Application of Transformed Operation



Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

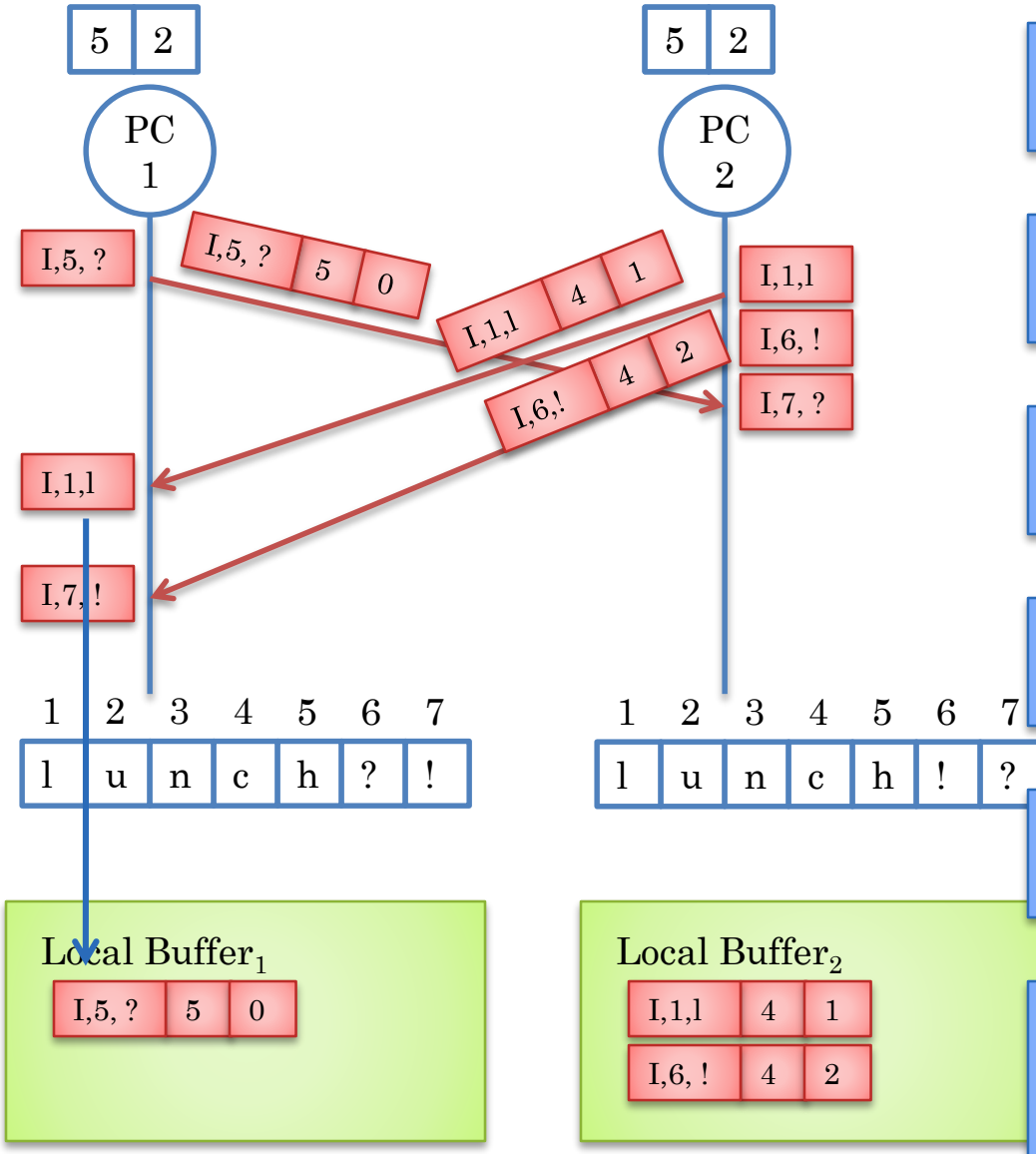
I,1,l	4	1
I,6, !	4	2

Inconsistency!

Never compared user ids at site 1

What went wrong?

What Went Wrong?



Never compared site ids at site 1

Effect of remote I, 1, 1 on local I, 5, ? not recorded

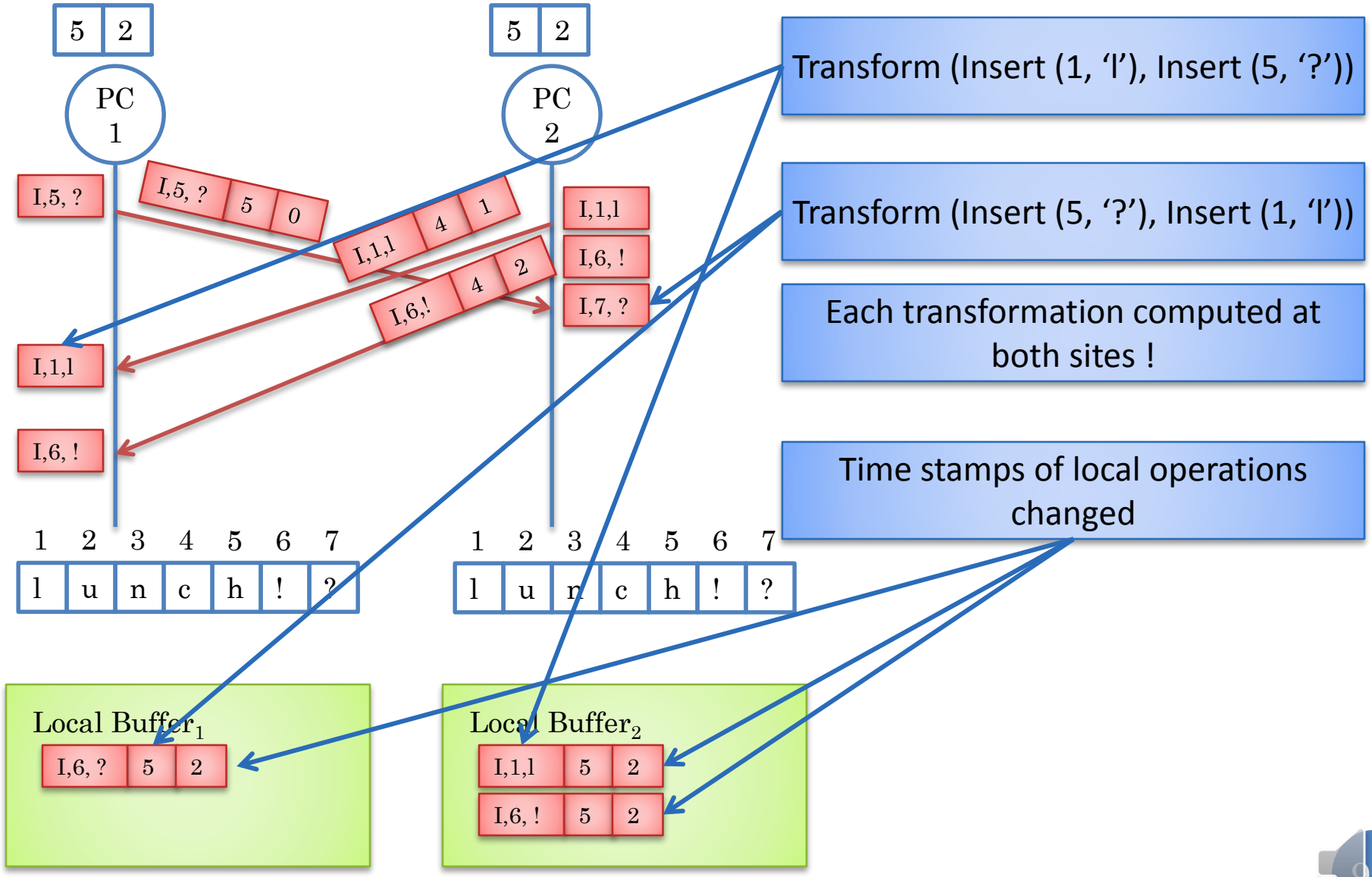
Must change time stamp and if necessary operands of local operation

Transform gives effect of an operation on another

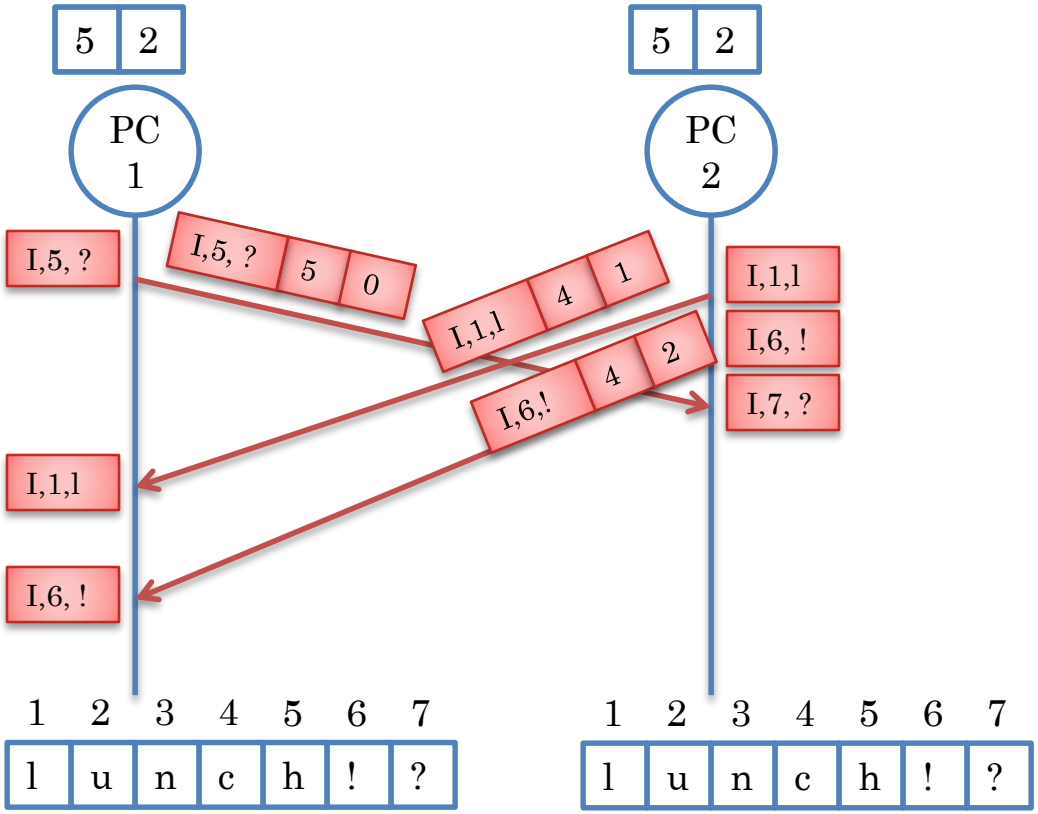
Used it so far to get effect of local operation on remote operation

Need to also use it to determine effect of remote operation on local operation

Running Transform in Pairs



Previous Control Algorithm



Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

Execute R

Site.TimeStamp.increment(R.site)

Local Buffer$_1$

I,6, ?	5	2

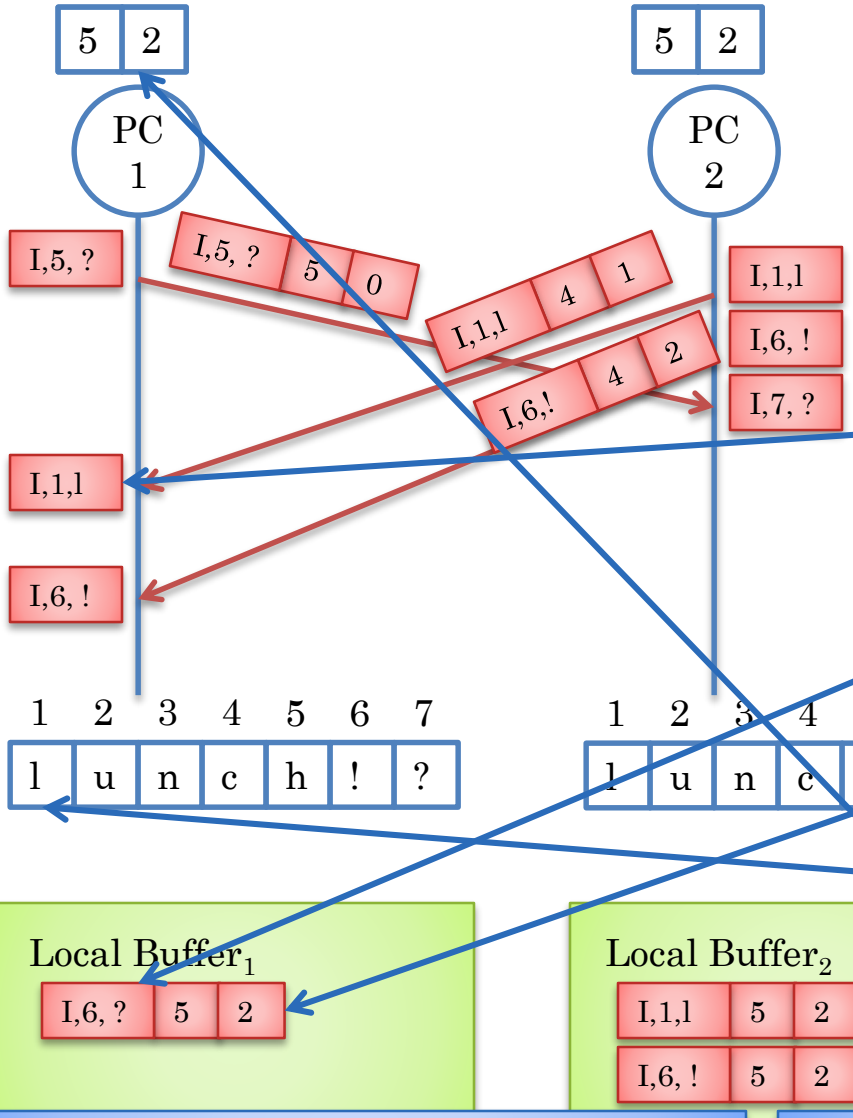
Local Buffer$_2$

I,1,1	5	2
I,6, !	5	2

Effects of L^1, L^2, .. L^N included in R

Effects of R must also be included in L^1, L^2, .. L^N

New Control Algorithm



Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

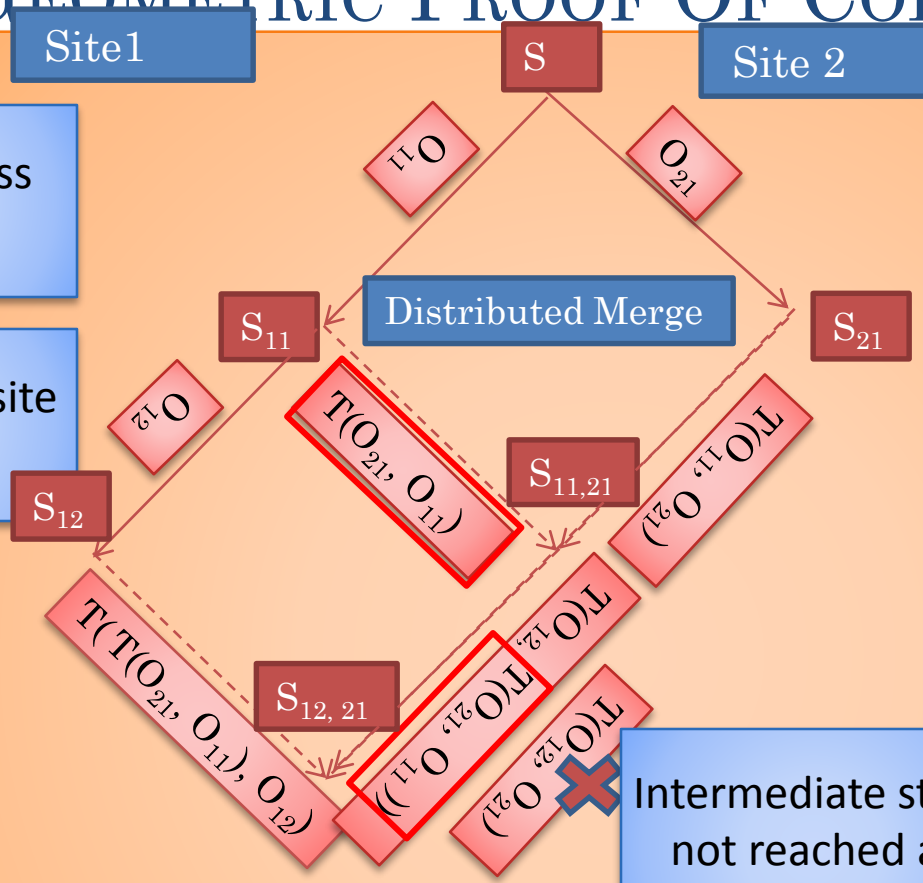
Execute R

Site.TimeStamp.increment(R.site)

Effects of L1, L2, .. LN included in R

Effects of R included in L1, L2, .. LN

Geometric Proof Of Correctness



SW : process site 1 op

SE: process site 2 op

One-step process

Given two sequences of concurrent ops, transformation merge process can be derived

Perpendiculars to two non dashed lines meet at a unique point

Intermediate state not reached at both sites

Multiple one-step processes to reach state

Without undo

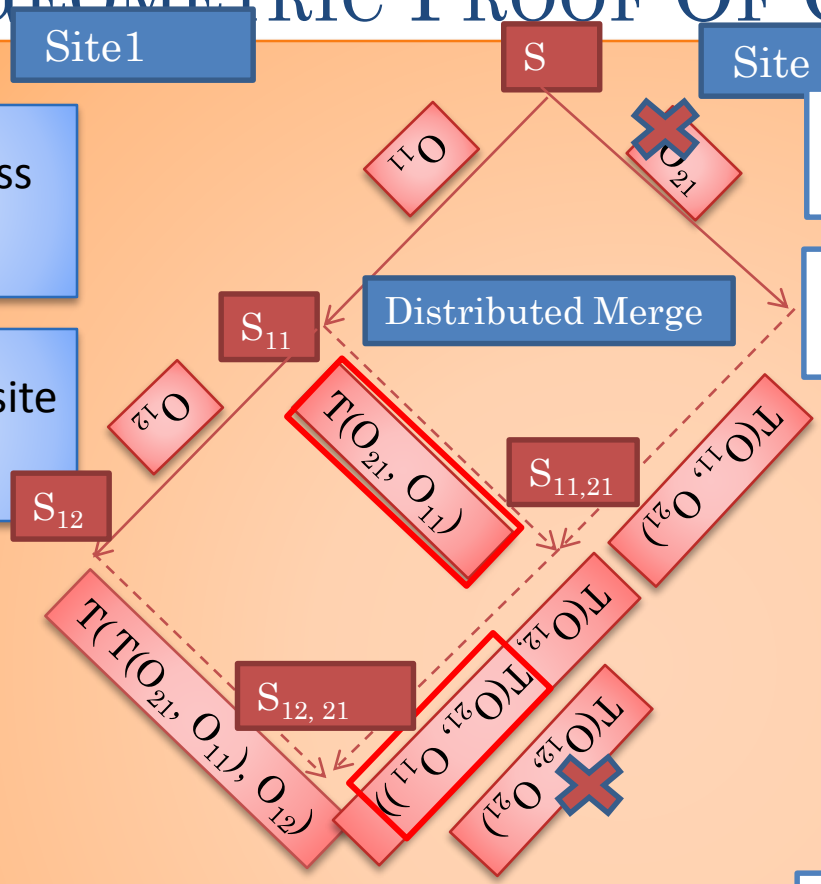
Local op not changed

Including effect of multiple locals on a remote at site 1

Including effect of multiple remotes on a single local at site 2

Same transformation computed at both sites

Geometric Proof Of Correctness



SW : process site 1 op

SE: process site 2 op

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

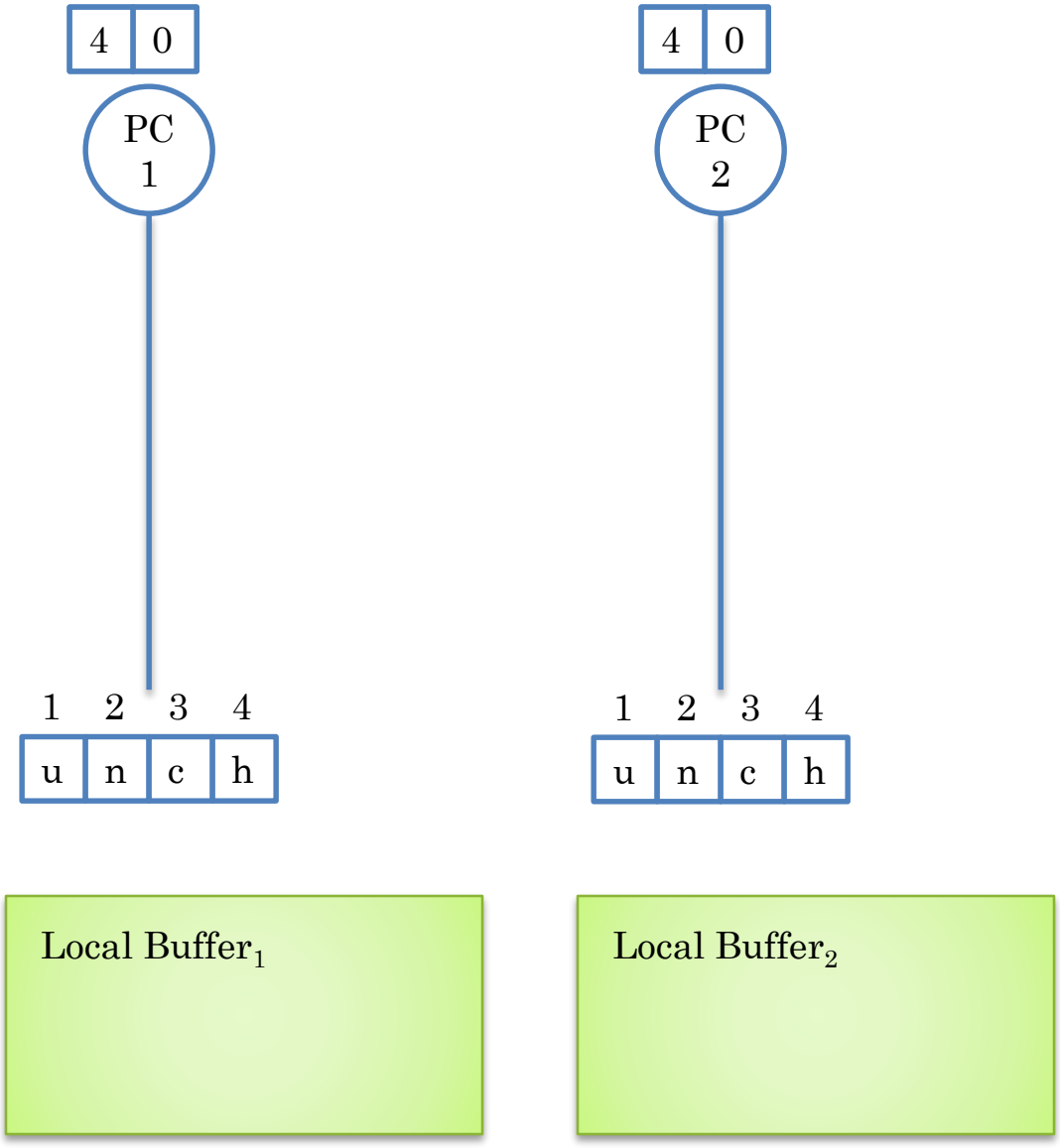
Execute R

Site.TimeStamp.increment(R.site)

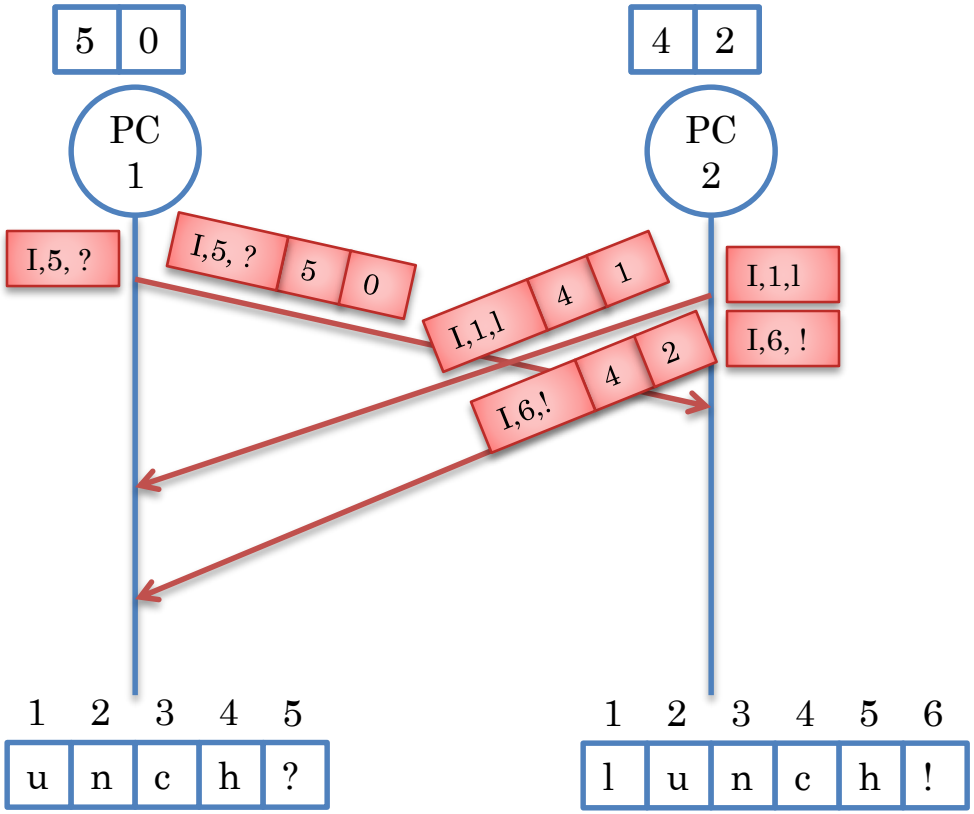
Including effect of multiple locals on a remote at site 1

Including effect of multiple remotes on a single local at site 2

Re-Run with Transformations



Concurrent Interaction



Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,1	4	1
I,6, !	4	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

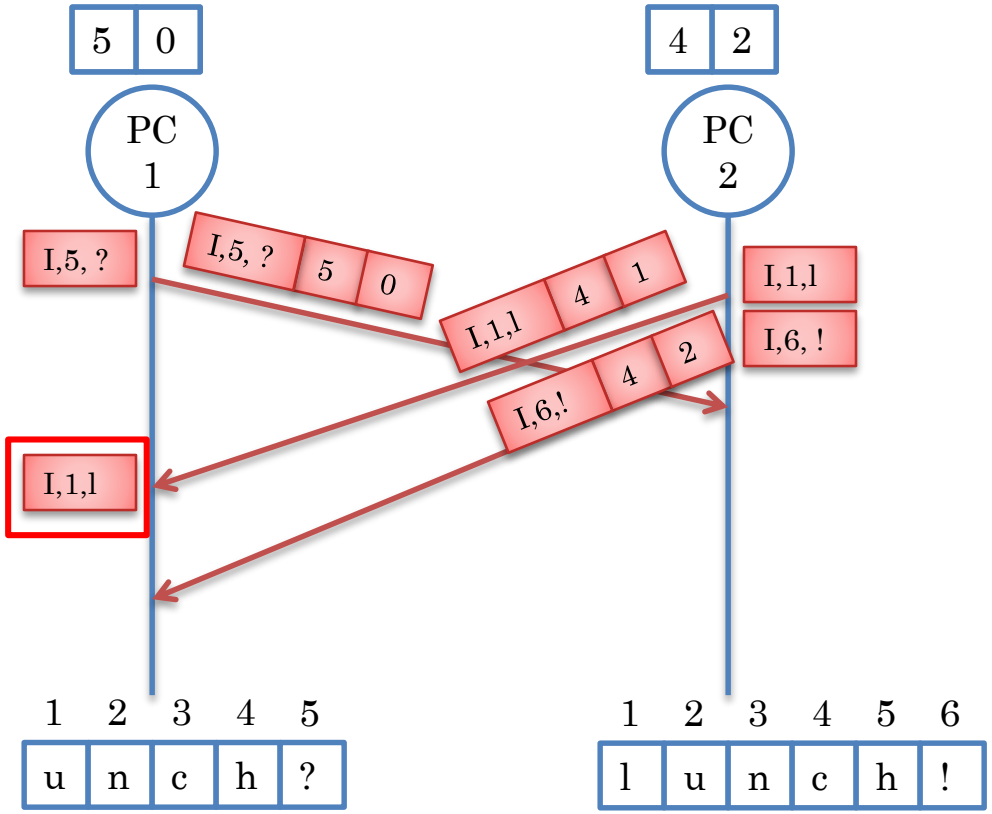
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Remote I, 1, l arrives at Site 1



Local Buffer$_1$

I,5, ?	5	0

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

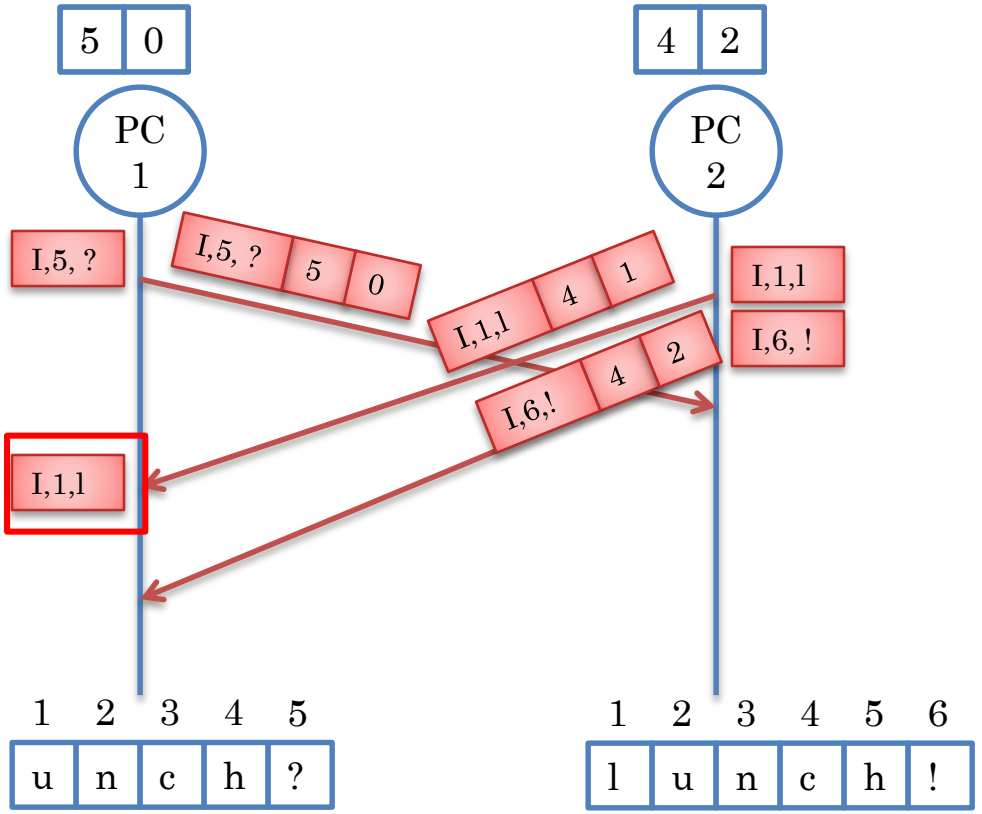
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Remote Not Transformed



Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Local Buffer$_1$

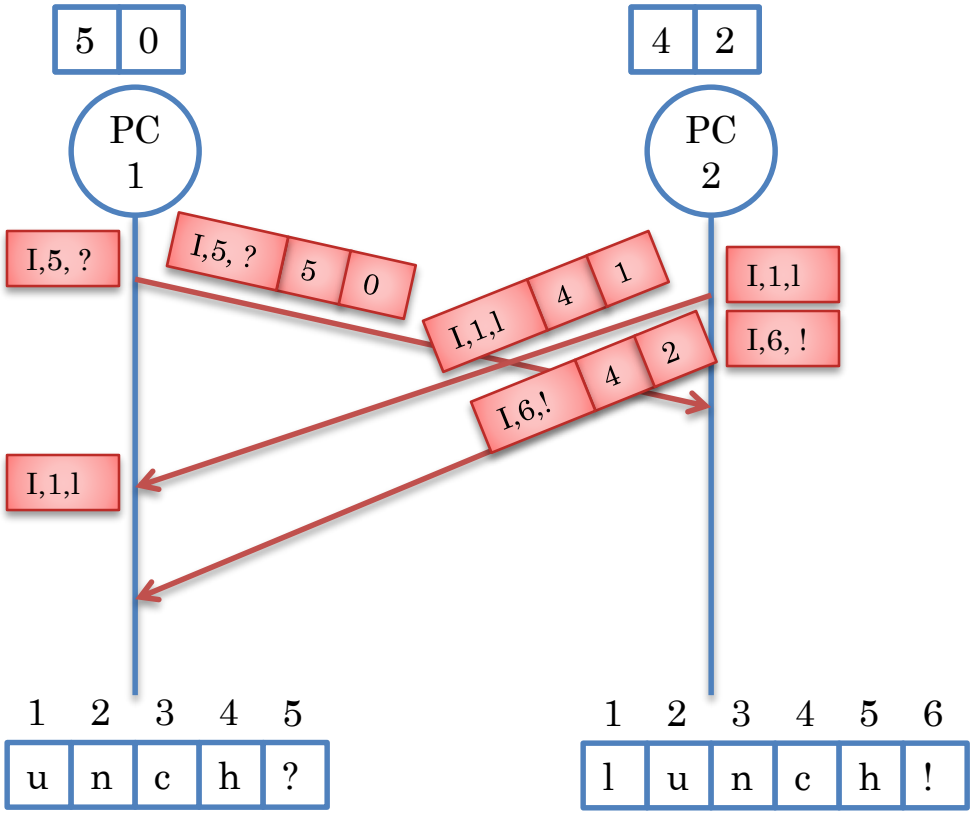
I,5, ?	5	0

Local Buffer$_2$

I,1,1	4	1
I,6, !	4	2

Remote not transformed

Local Transformed



Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Local transformed

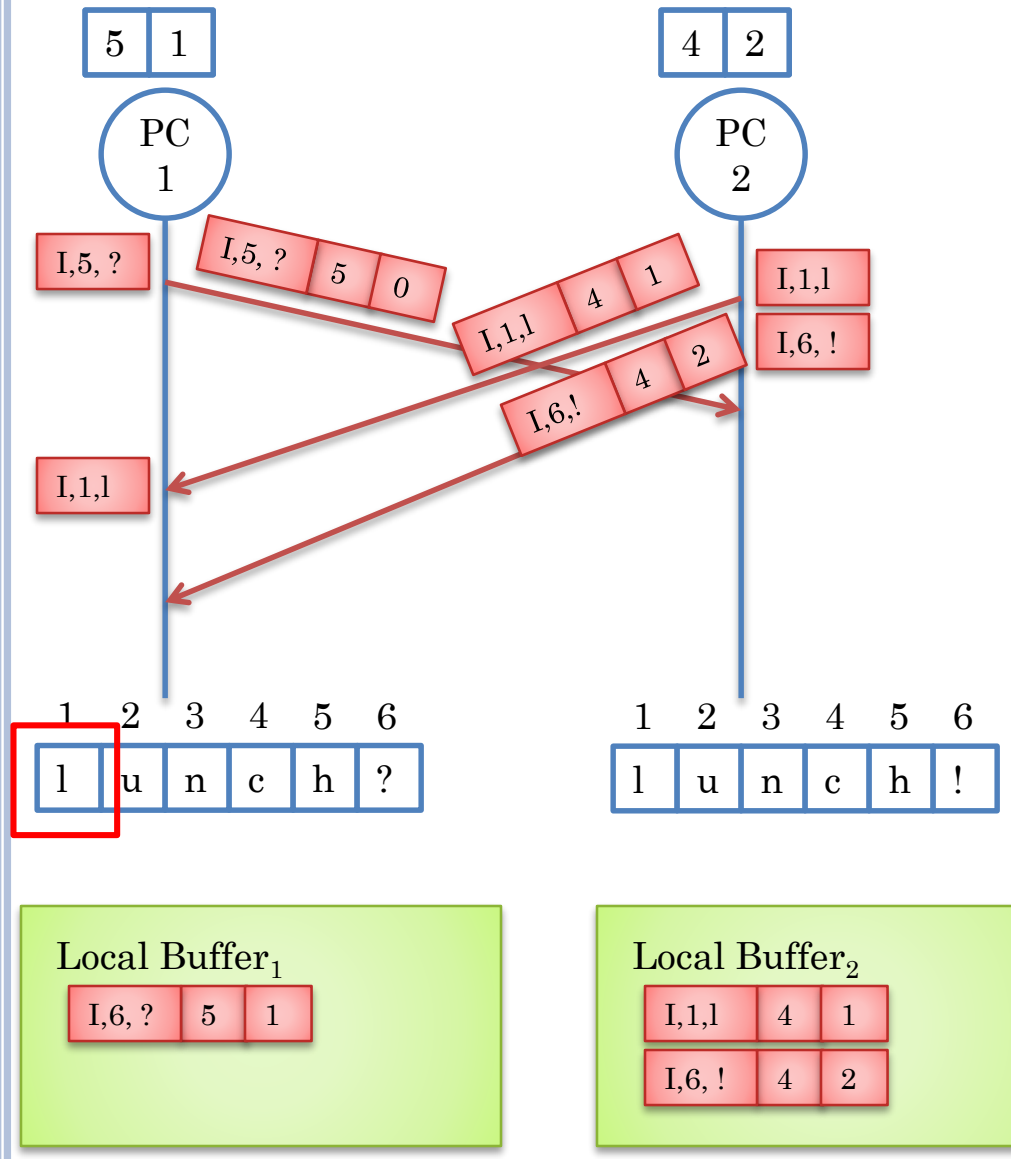
Local Buffer$_1$

I,6, ?	5	1

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Remote Applied



Local Buffer$_1$

I,6, ?	5	1

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

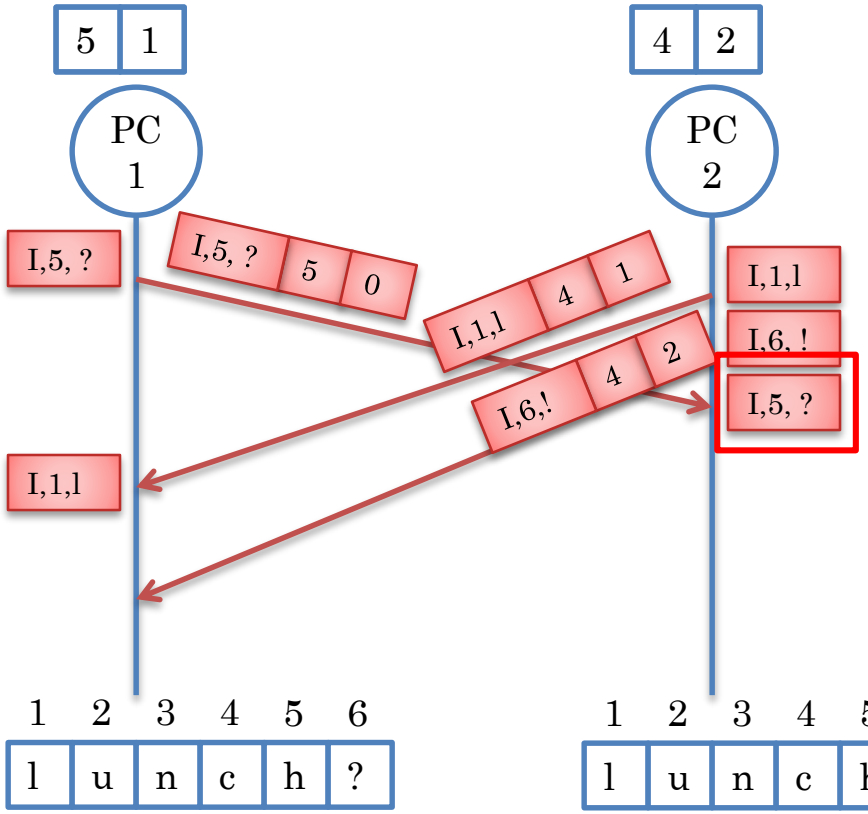
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

I,5, ? Arrives at Site 2



Local Buffer$_1$

I,6, ?	5	1

Local Buffer$_2$

I,1,l	4	1
I,6, !	4	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

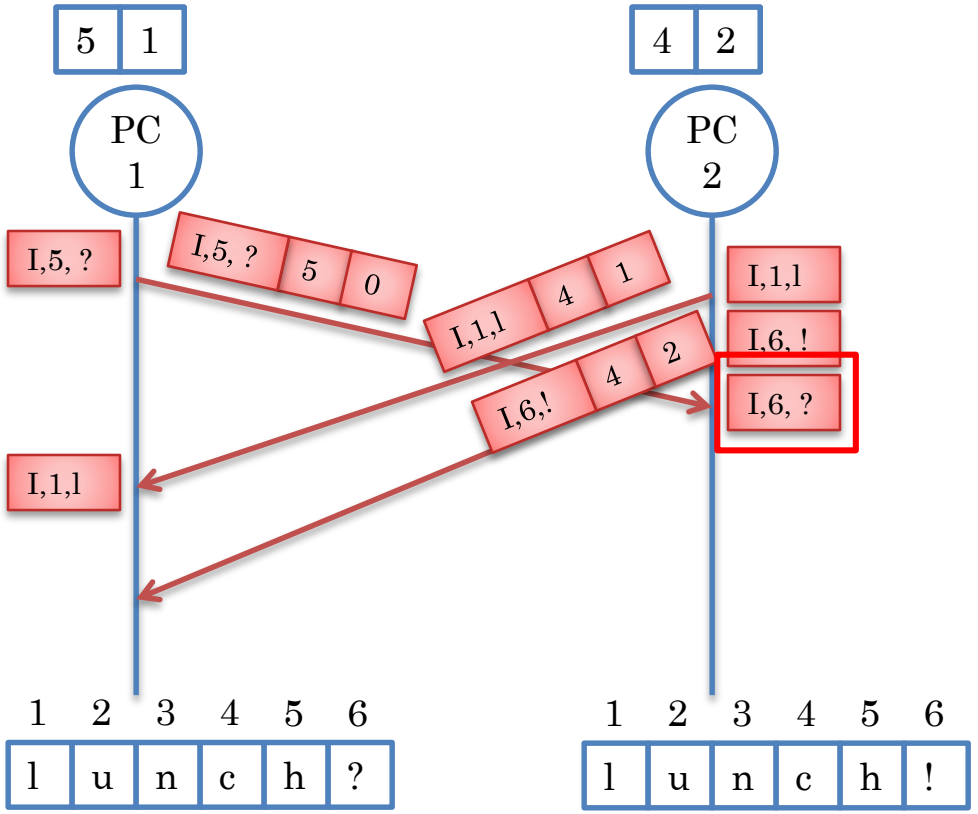
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Remote Transformed But Not First Local



Local Buffer$_1$

I,6, ?	5	1

Local Buffer$_2$

I,1,1	5	1
I,6, !	4	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

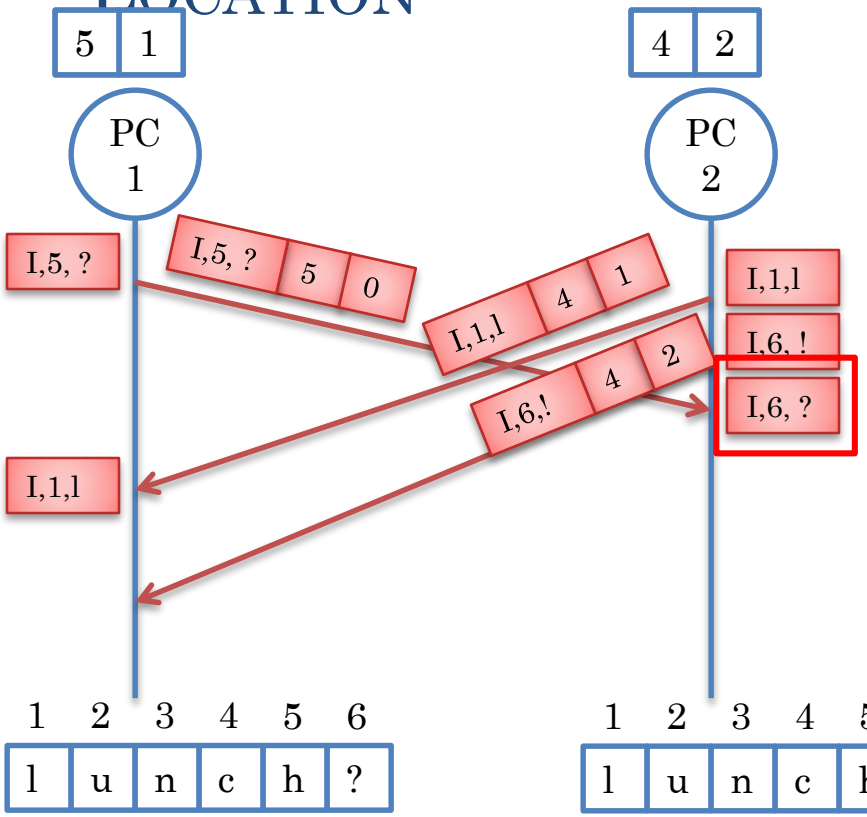
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Compared with Second Local (I. 6, !) at same Location



Local Buffer$_1$

I,6, ?	5	1

Local Buffer$_2$

I,1,1	5	1
I,6, !	4	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

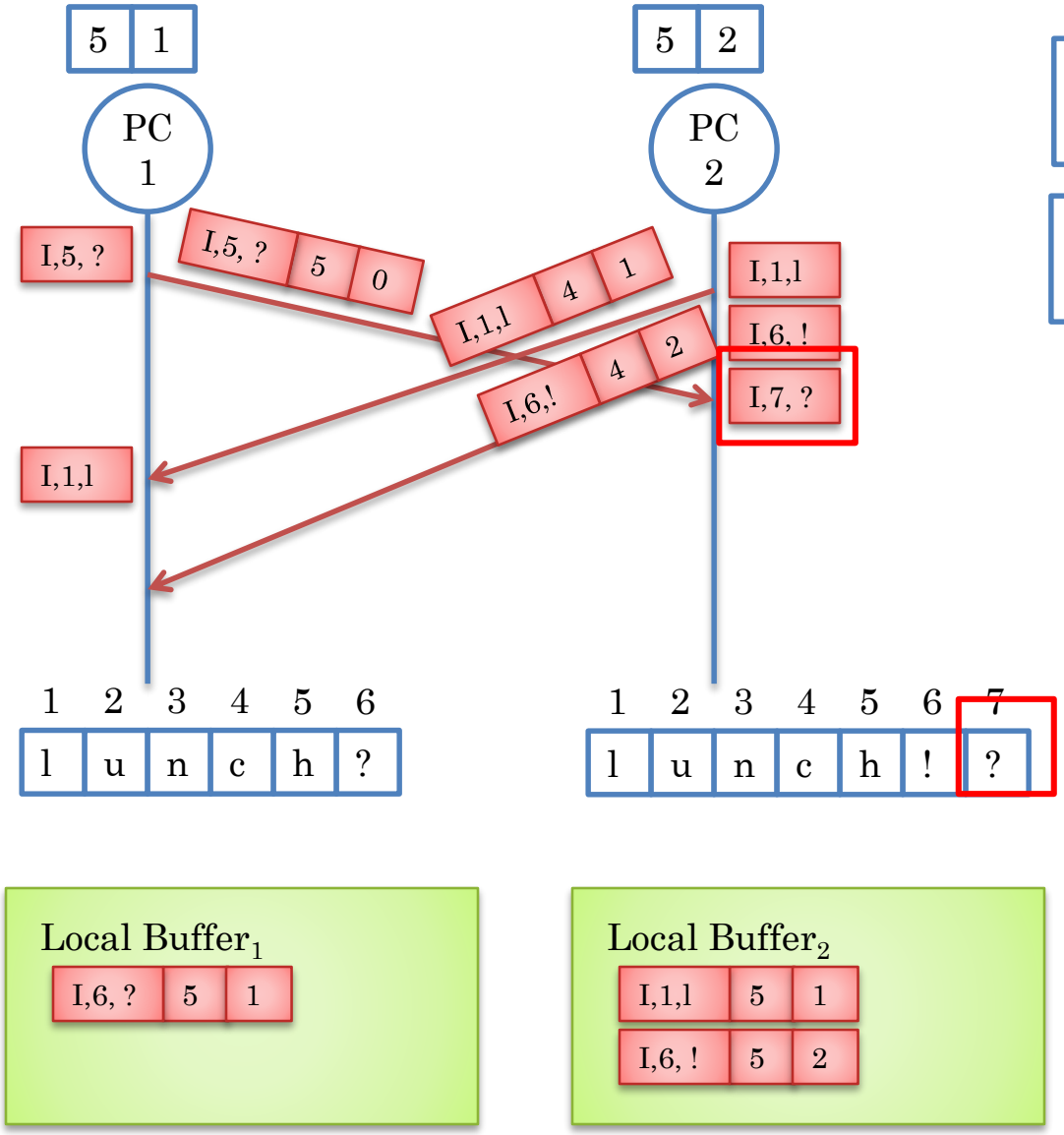
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Transformation and Application



Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

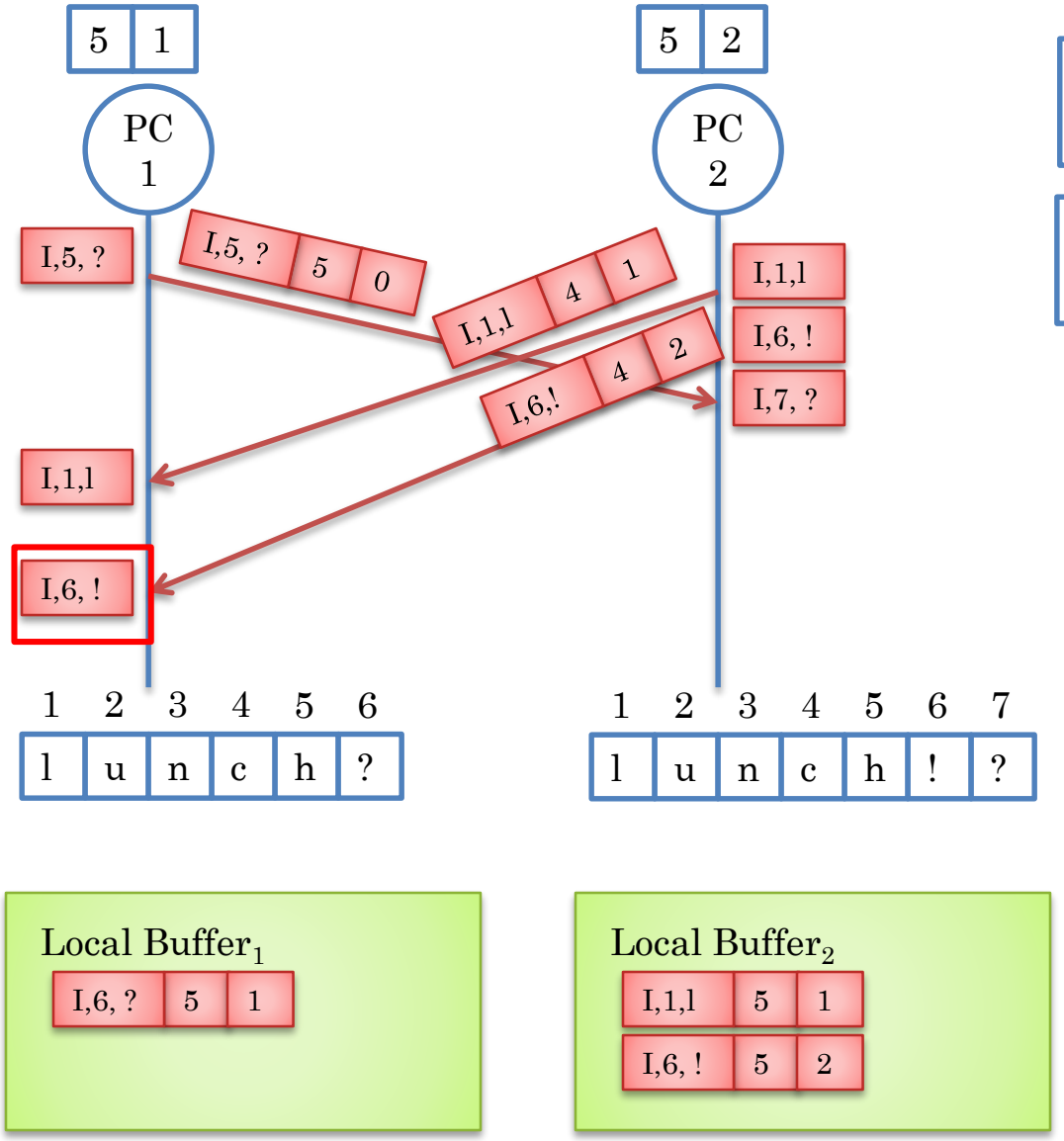
Local Buffer$_1$

I,6, ?	5	1

Local Buffer$_2$

I,1,1	5	1
I,6, !	5	2

Second Remote Operation Arrives at Site 1



Local Buffer$_1$

I,6, ?	5	1

Local Buffer$_2$

I,1,1	5	1
I,6, !	5	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

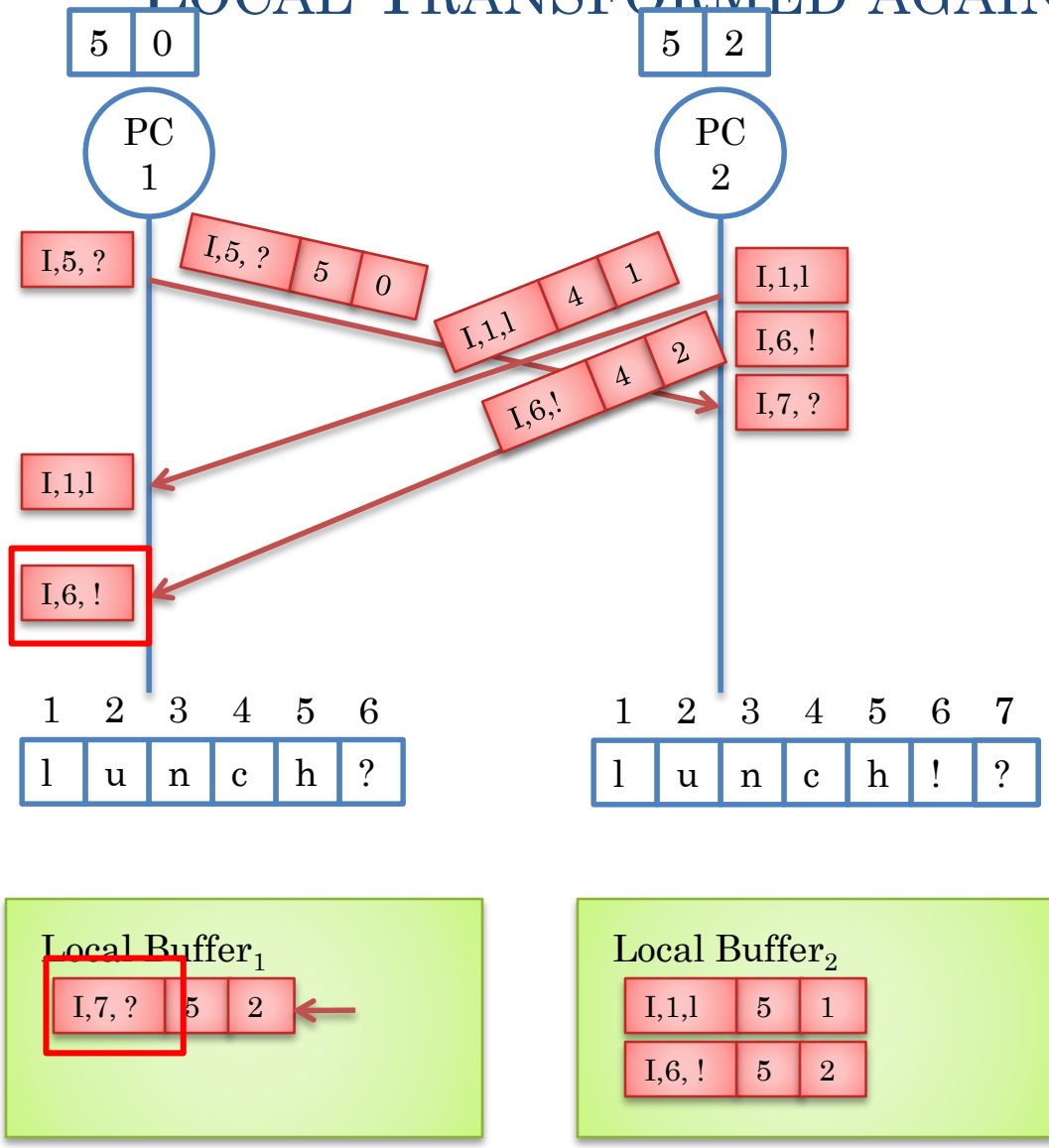
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Lower Priority Previously Transformed Local Transformed again but not Remote



Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

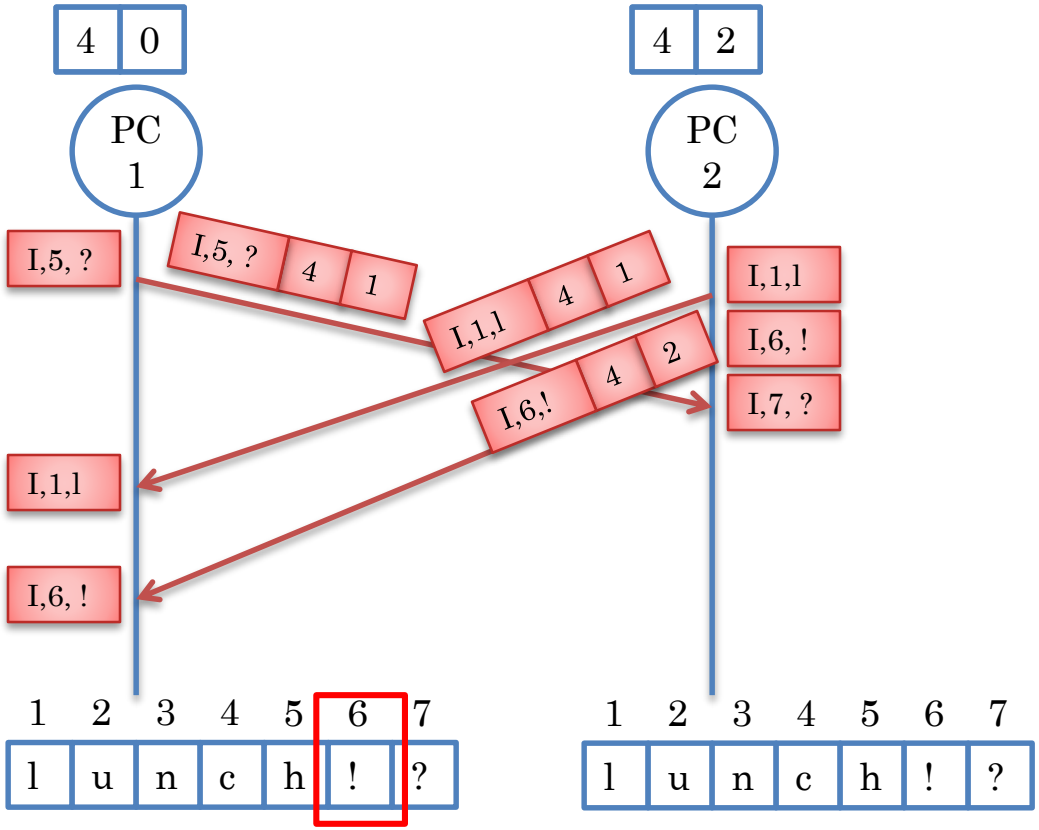
L = Transform (L, R)

L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

Untransformed Remote Applied



Local Buffer$_1$

I,7,?	5	2

Local Buffer$_2$

I,1,1	5	1
I,6, !	5	2

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R = Transform (R, L)

L = Transform (L, R)

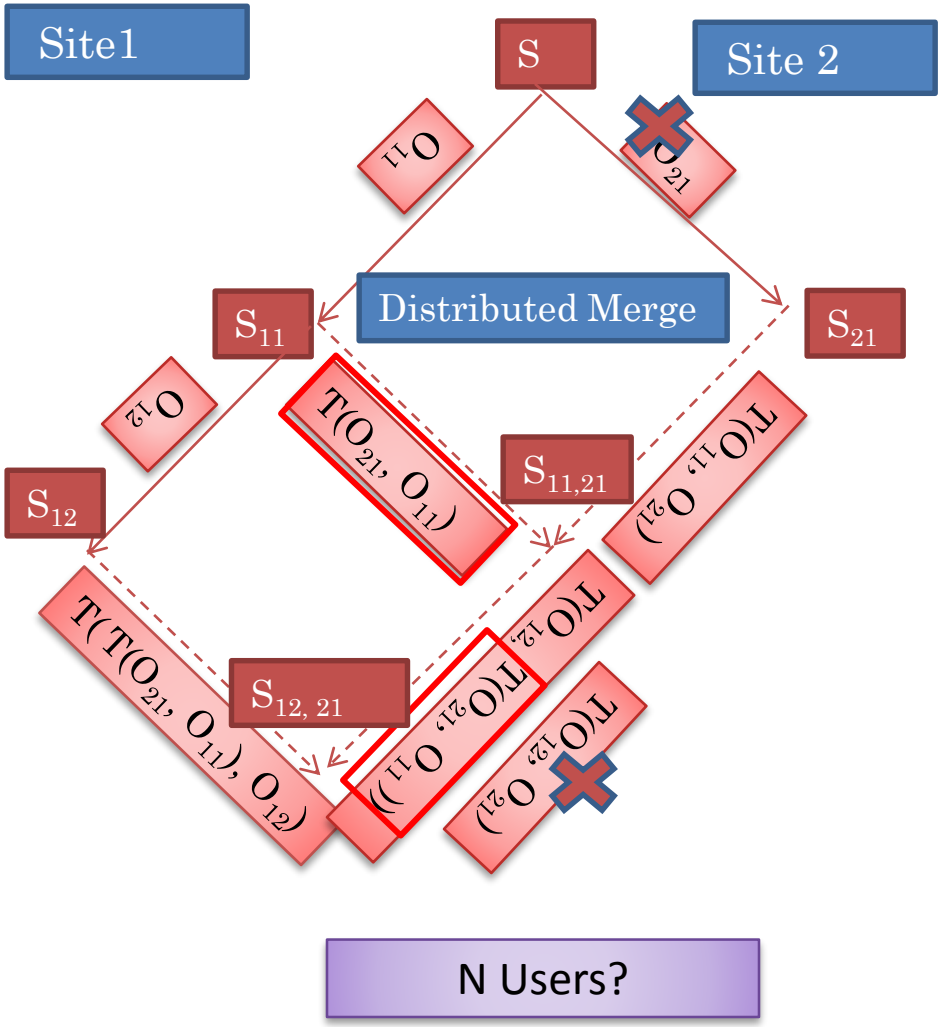
L.TimeStamp.increment(R.site)

Execute R

Site.TimeStamp.increment(R.site)

N Users?

Algorithm and Proof Of Correctness



Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R^T = Transform (R, L)

L = Transform (L R)

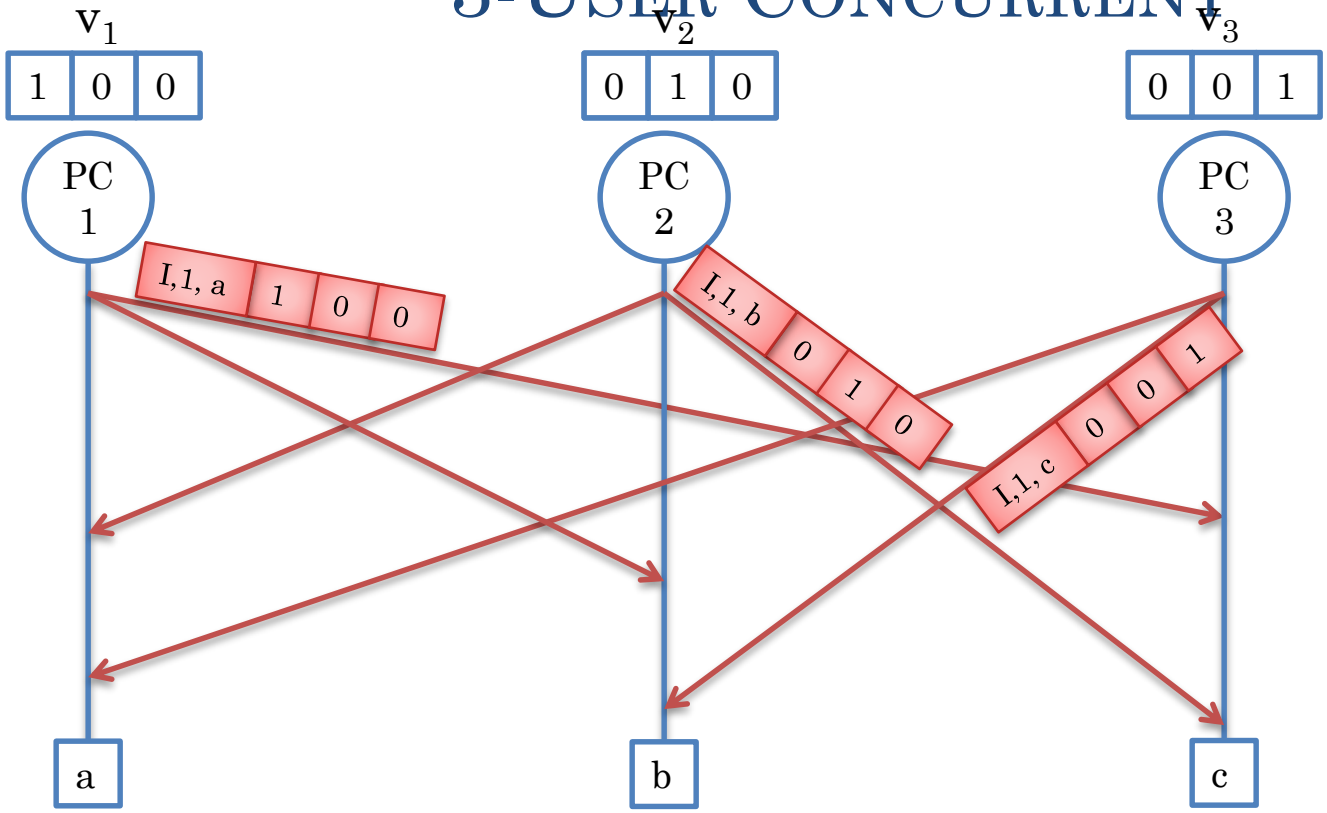
L.TimeStamp.increment(R.site)

R= R^T

Execute R

Site.TimeStamp.increment(R.site)

3-User Concurrent



Local Buffer$_1$

I,1, a	1	0	0

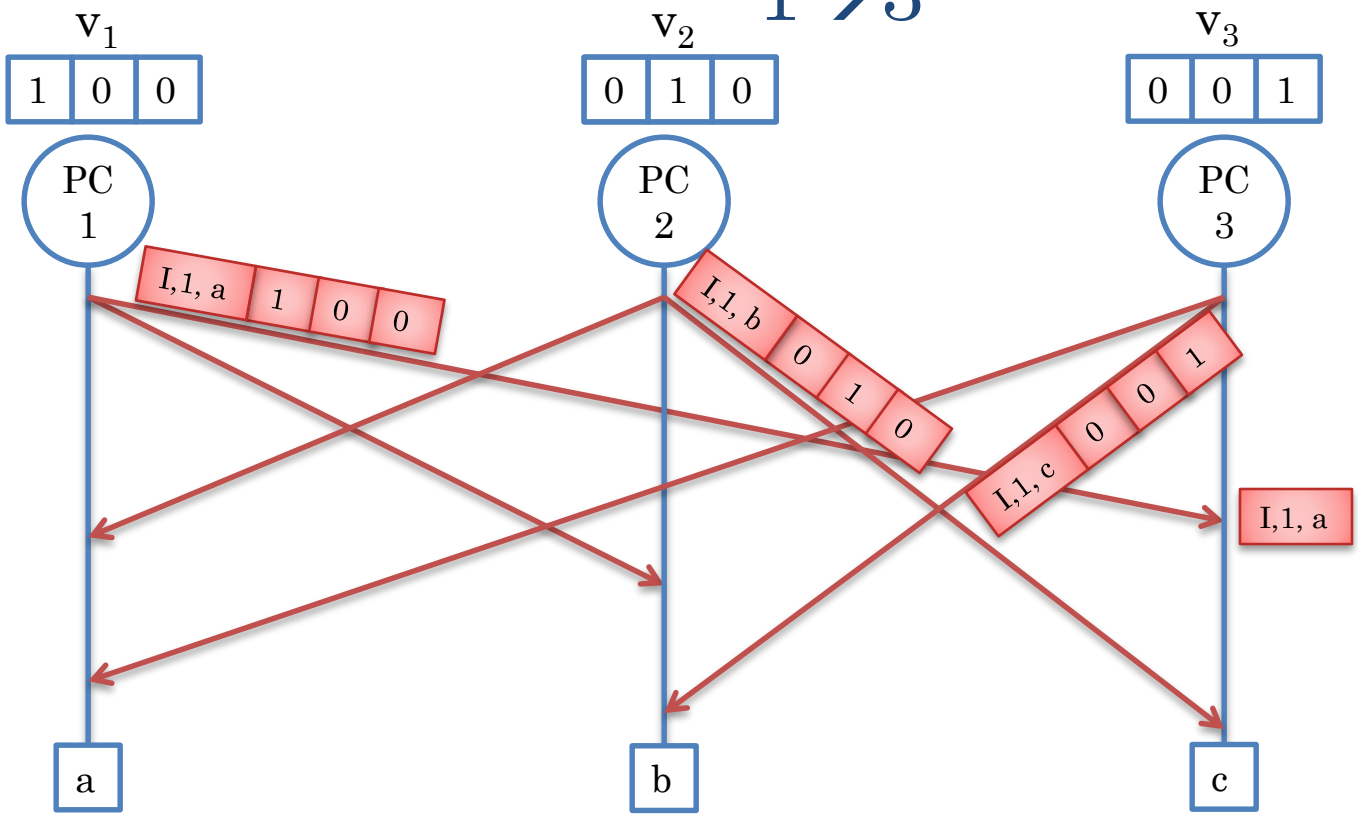
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	0	1

1→3



Local Buffer$_1$

I,1, a	1	0	0

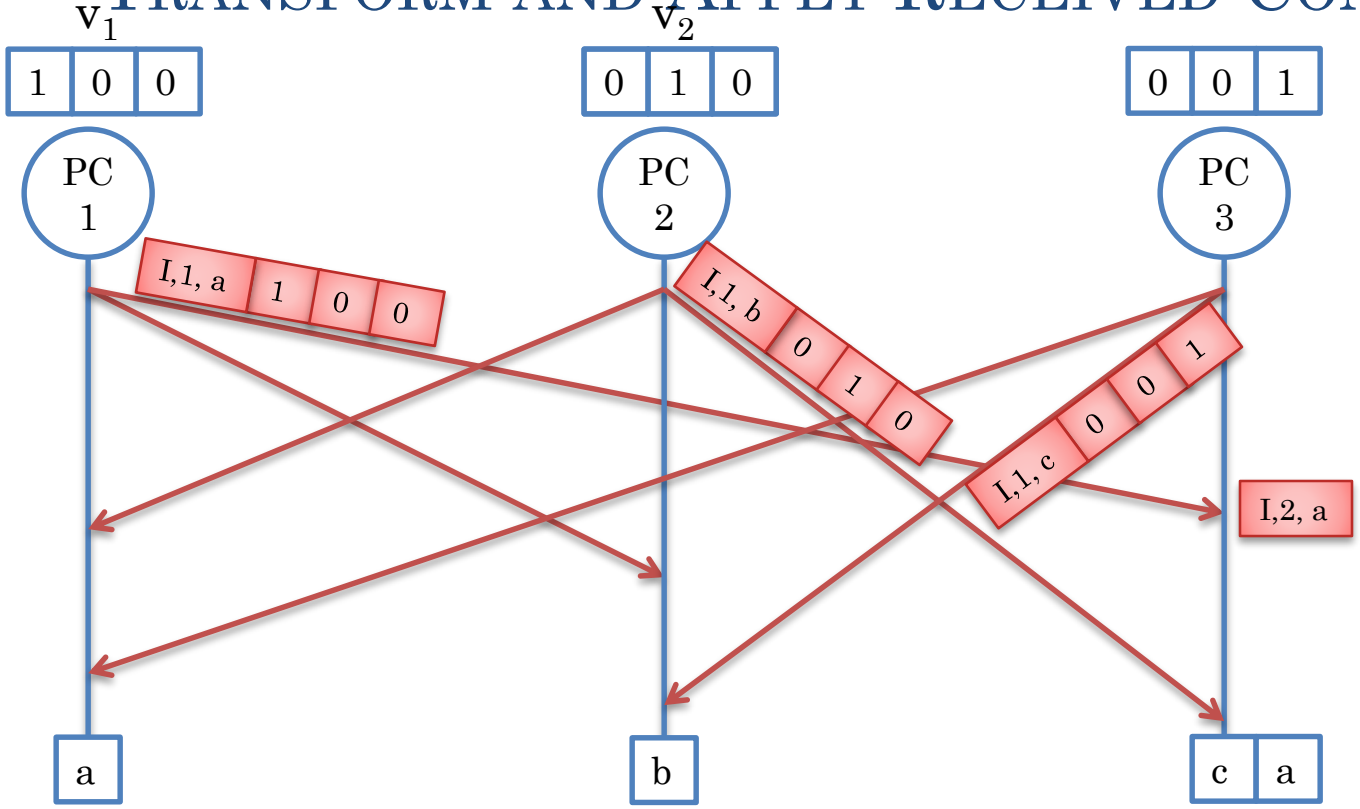
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	0	1

Transform and Apply Received Command



Local Buffer$_1$

I,1, a	1	0	0

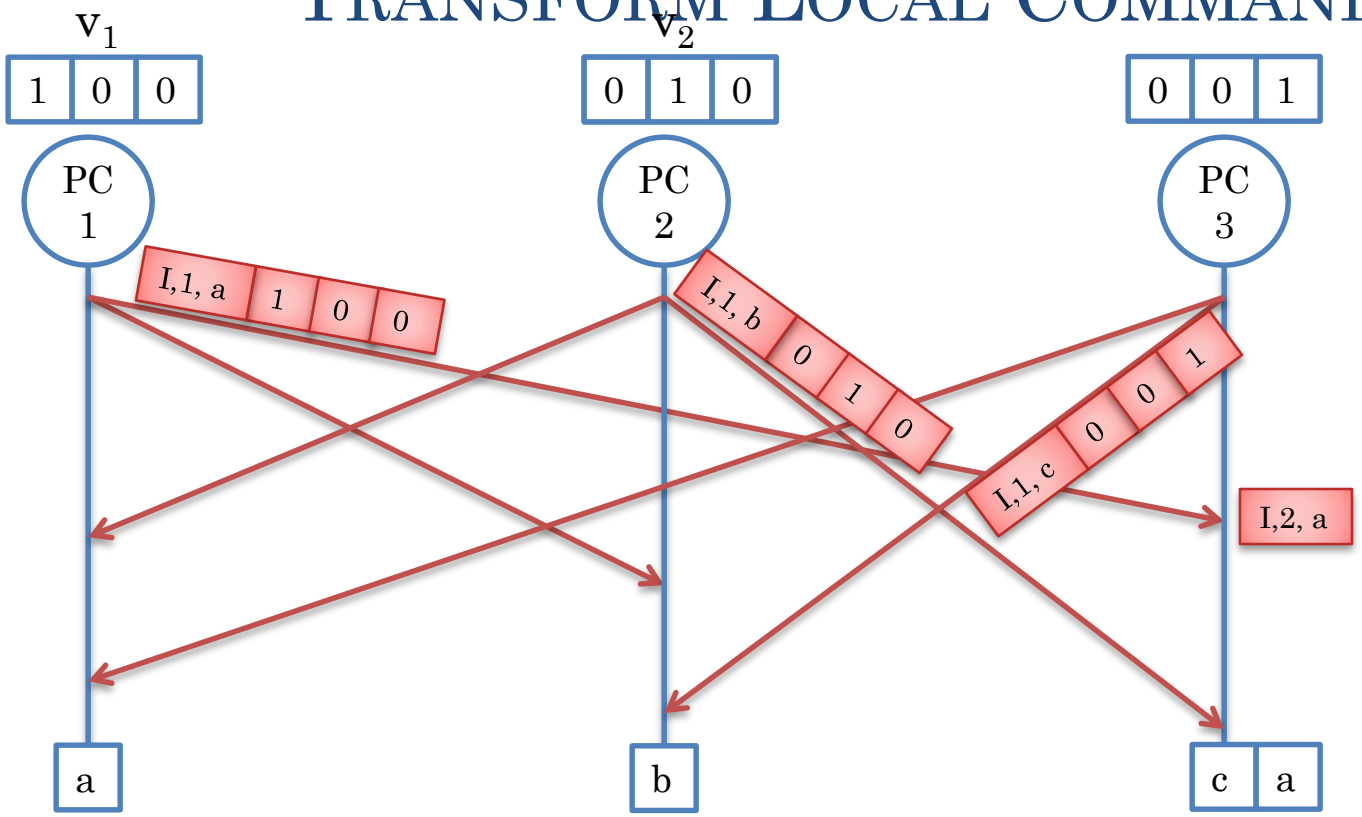
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	0	1

Transform Local Command



Local Buffer$_1$

I,1, a	1	0	0

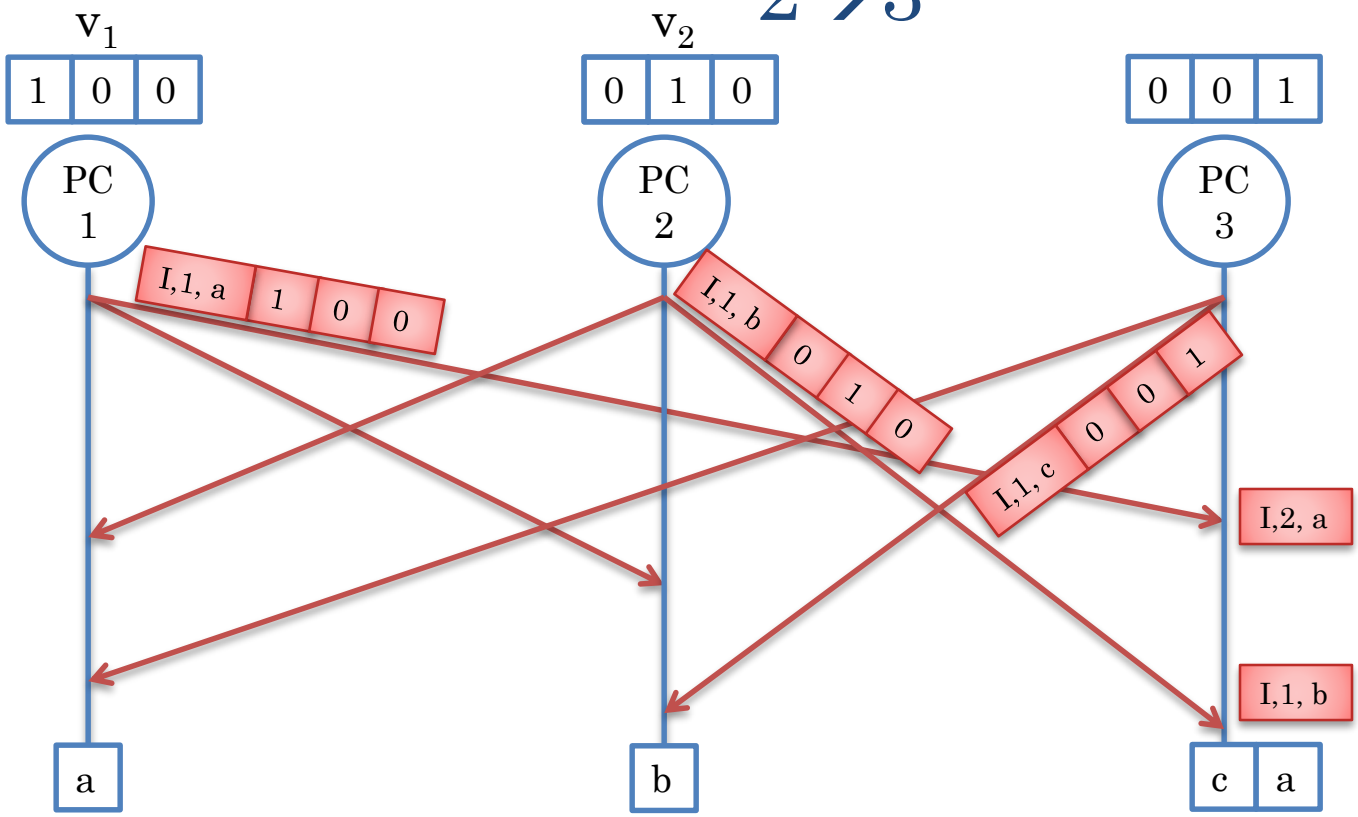
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	1	0	1

2→3



Local Buffer$_1$

I,1, a	1	0	0

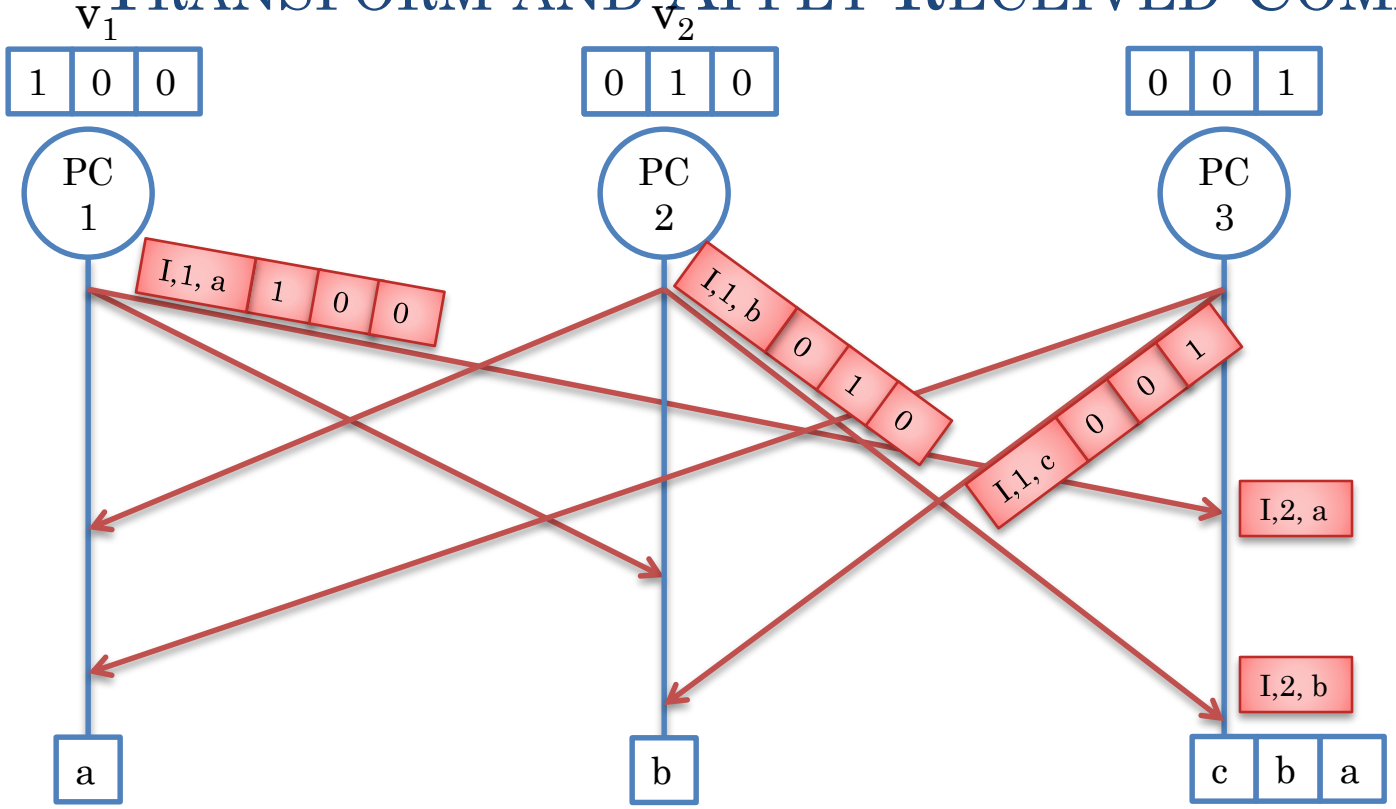
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	1	0	1

Transform and Apply Received Command



Local Buffer$_1$

I,1, a	1	0	0

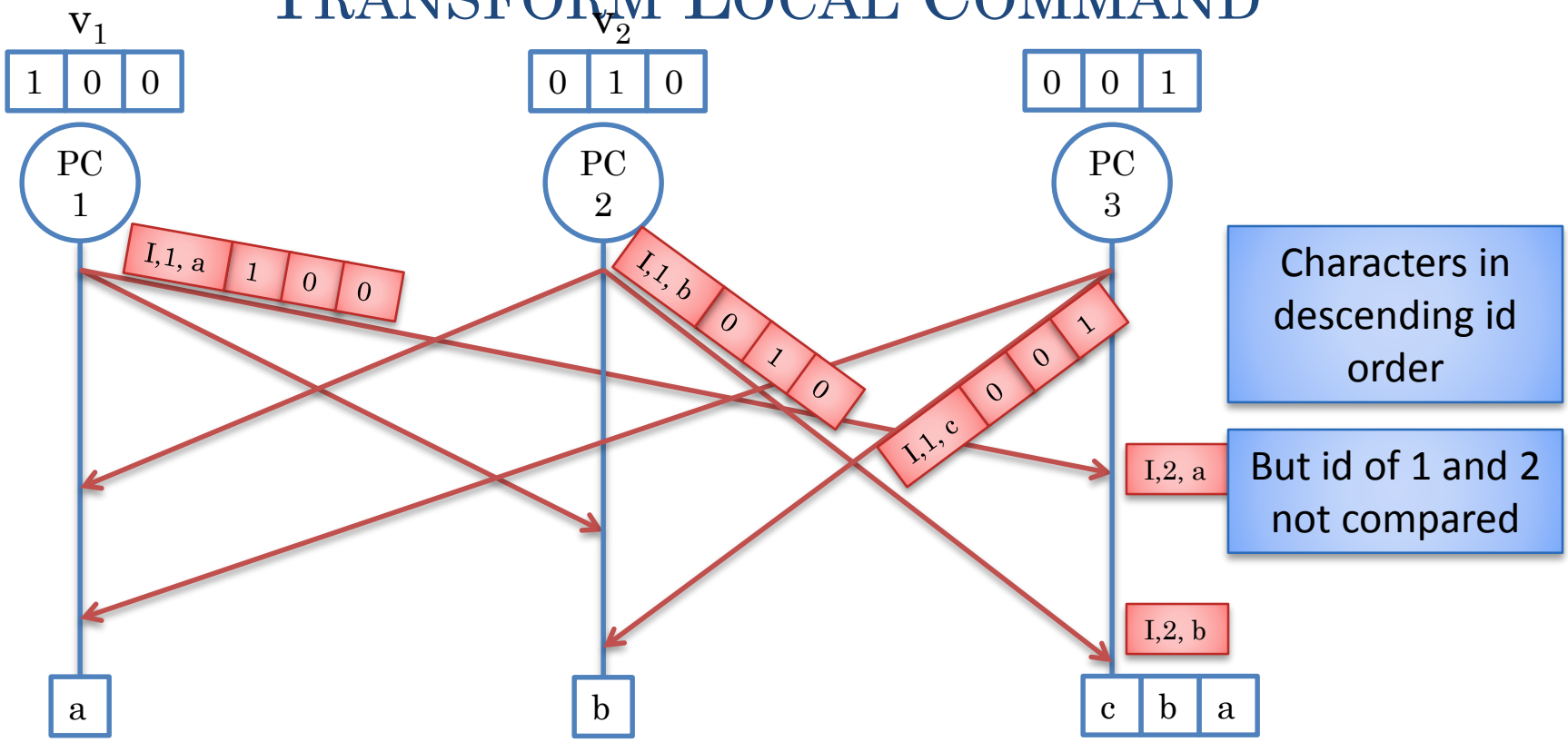
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	1	0	1

Transform Local Command



Characters in descending id order

But id of 1 and 2 not compared

Local Buffer$_1$

I,1, a | 1 | 0 | 0

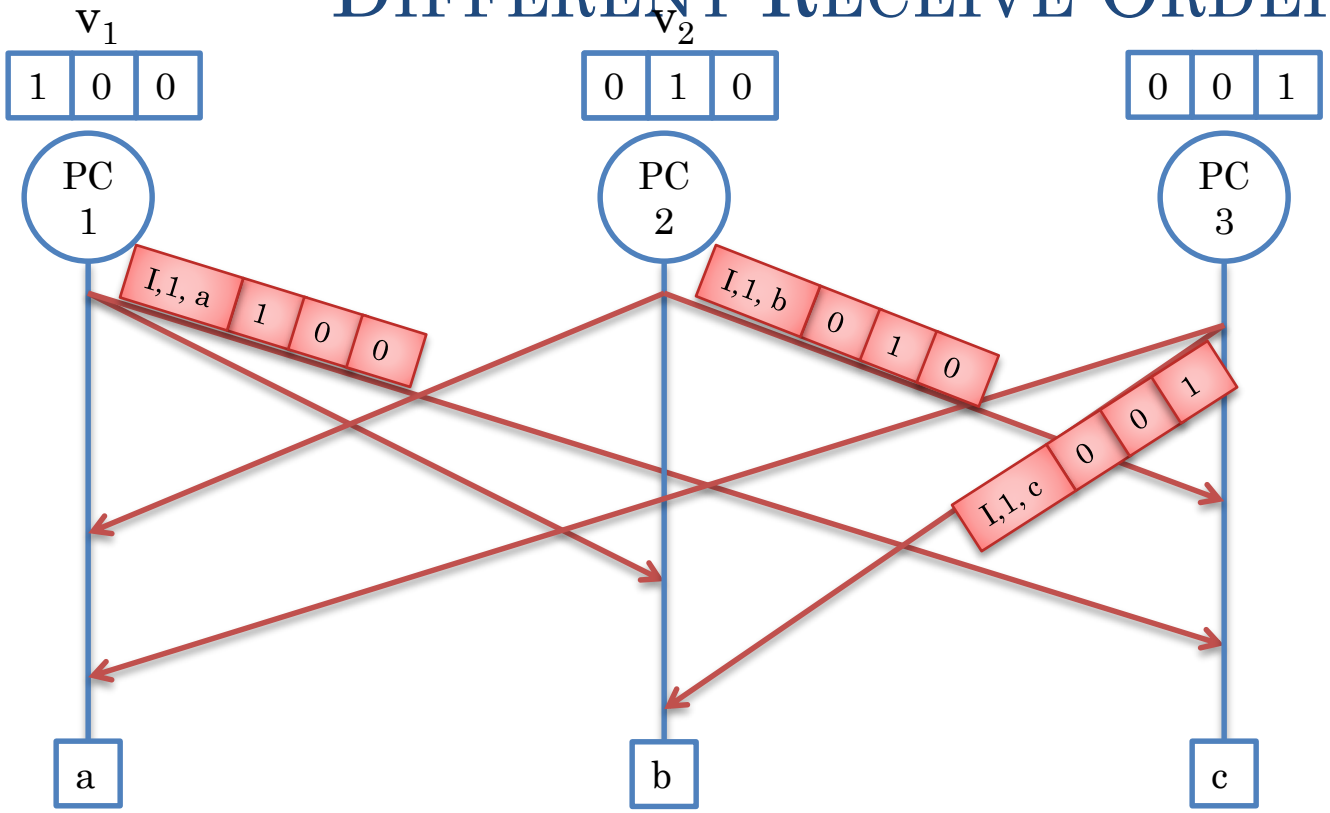
Local Buffer$_2$

I,1, b | 0 | 1 | 0

Local Buffer$_3$

I,1, c | 1 | 1 | 1

Different Receive Order



Local Buffer$_1$

I,1, a	1	0	0

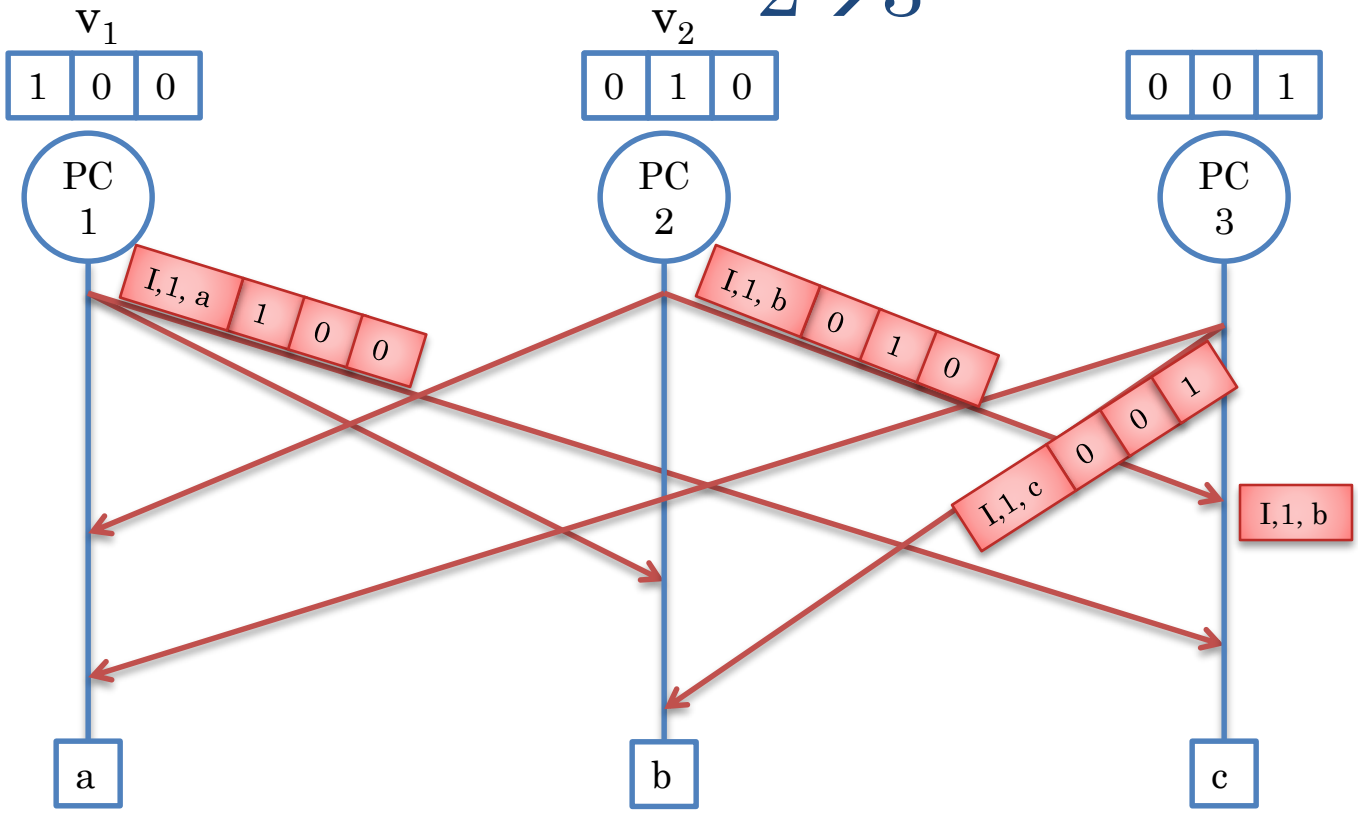
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	0	1

2→3



Local Buffer$_1$

I,1, a	1	0	0

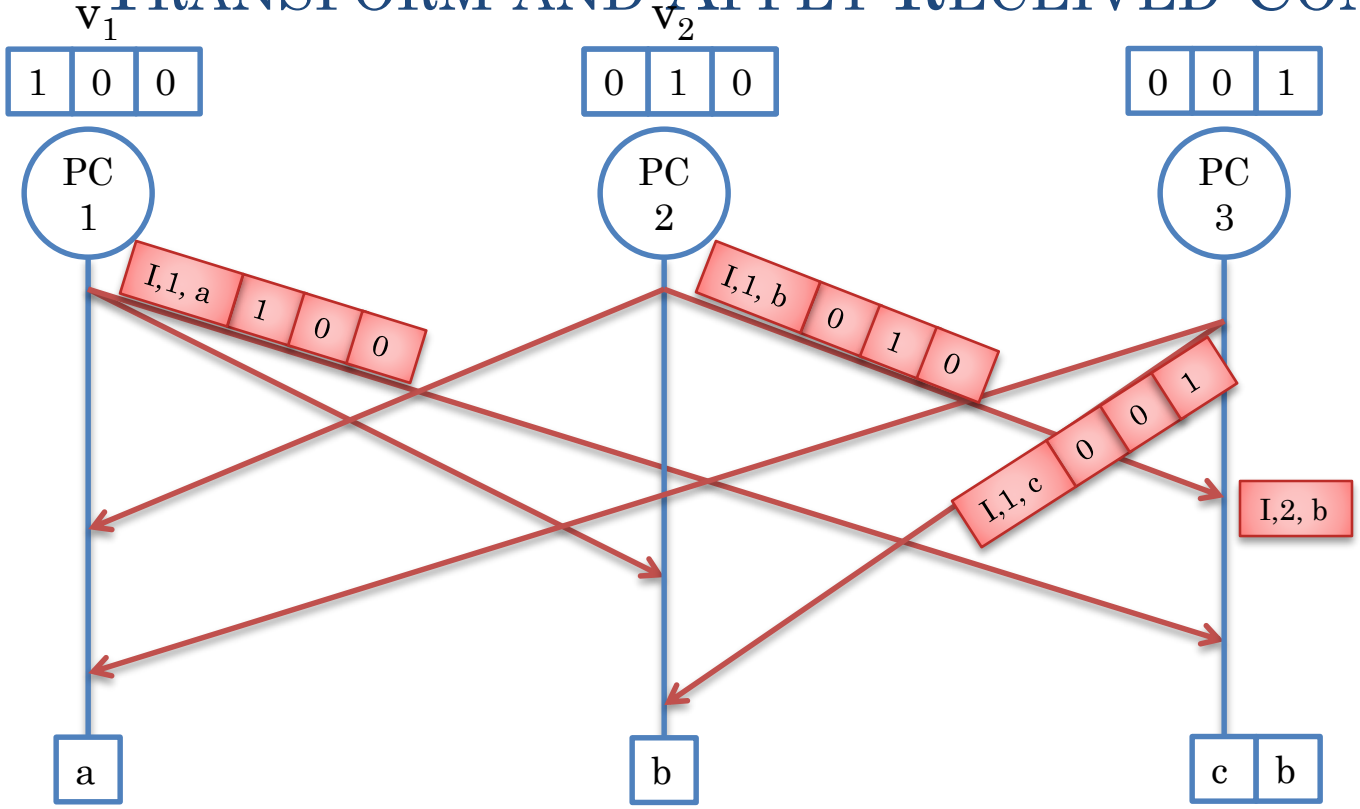
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	0	1

Transform and Apply Received Command



Local Buffer$_1$

I,1, a	1	0	0

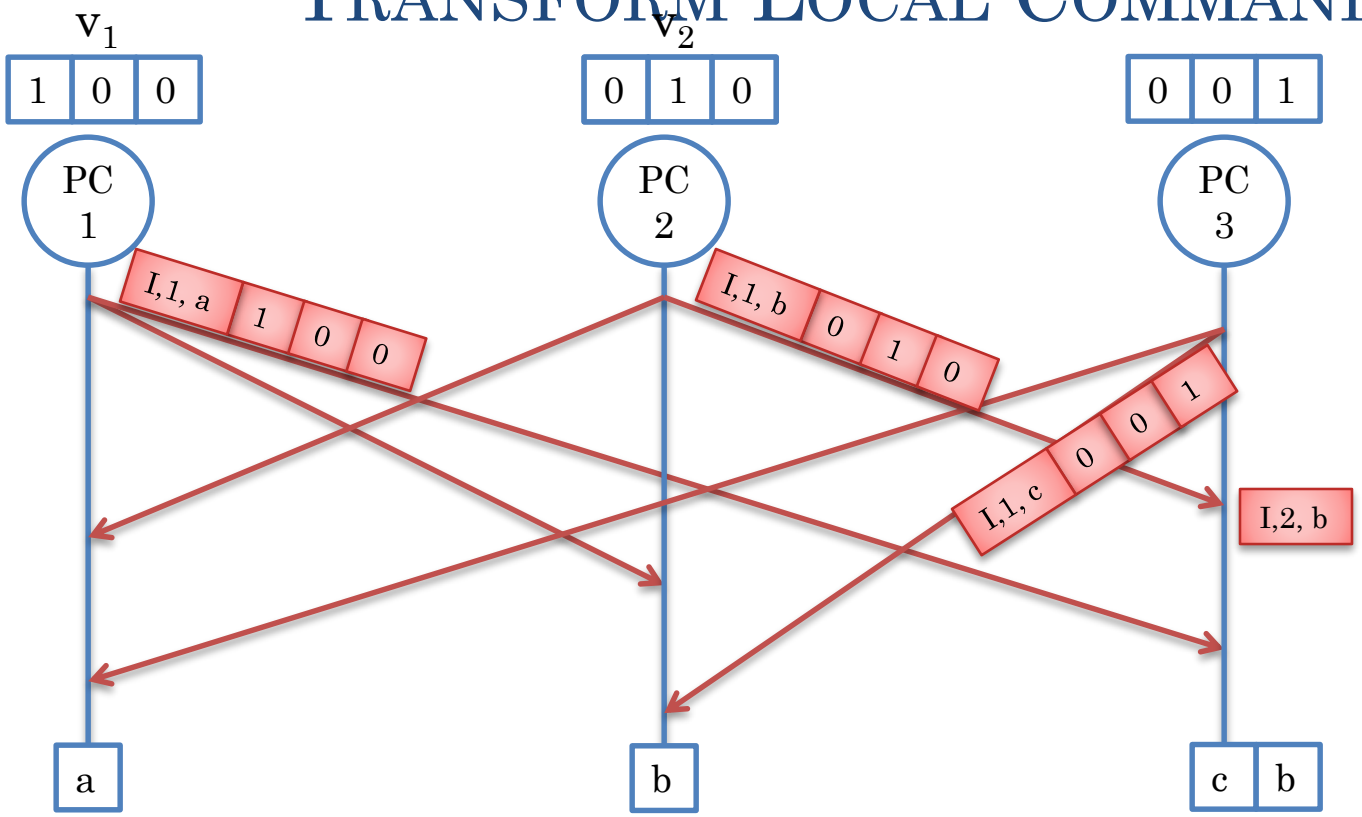
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	0	1

Transform Local Command



Local Buffer$_1$

I,1, a	1	0	0

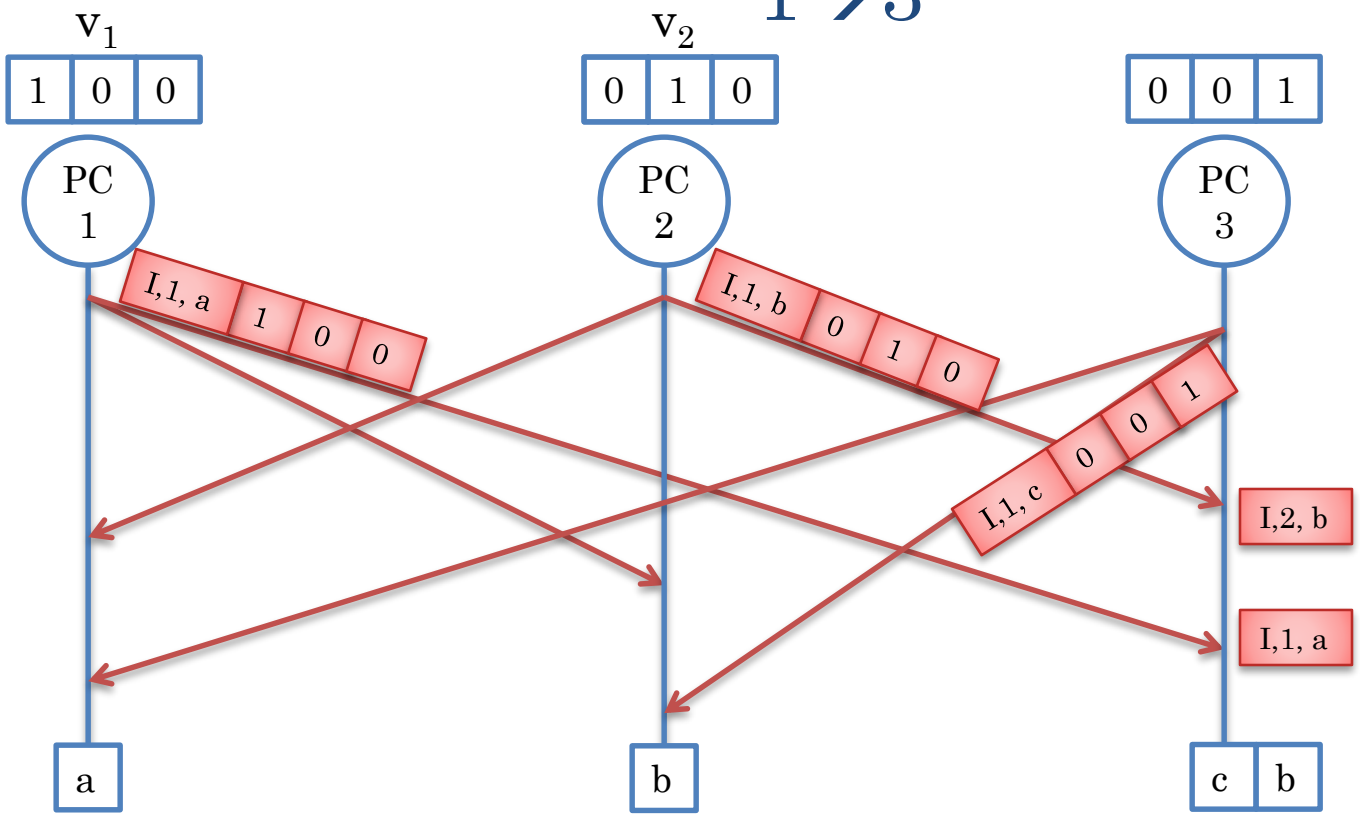
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	1	1

1→3



Local Buffer$_1$

I,1, a	1	0	0

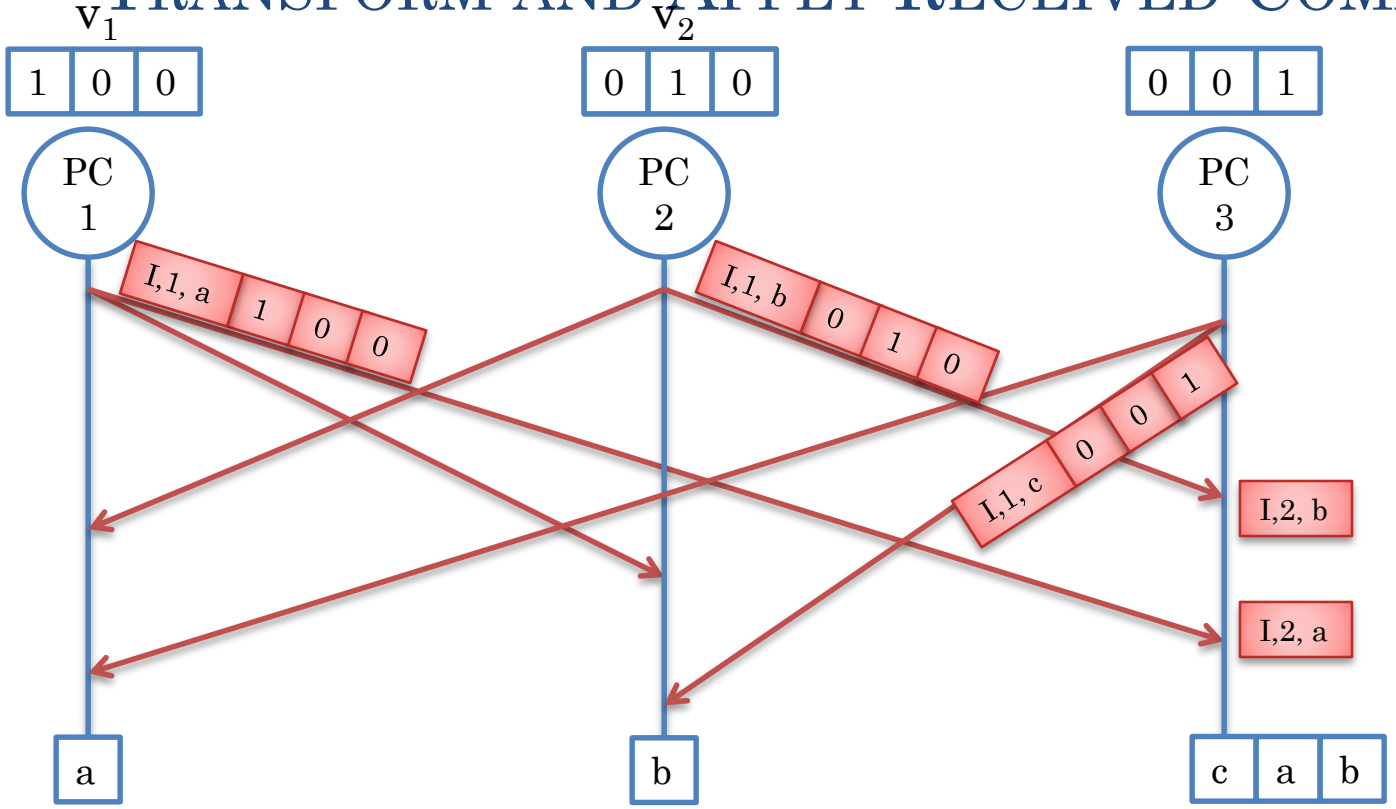
Local Buffer$_2$

I,1, b	0	1	0

Local Buffer$_3$

I,1, c	0	1	1

Transform and Apply Received Command



Local Buffer$_1$

I,1, a | 1 | 0 | 0

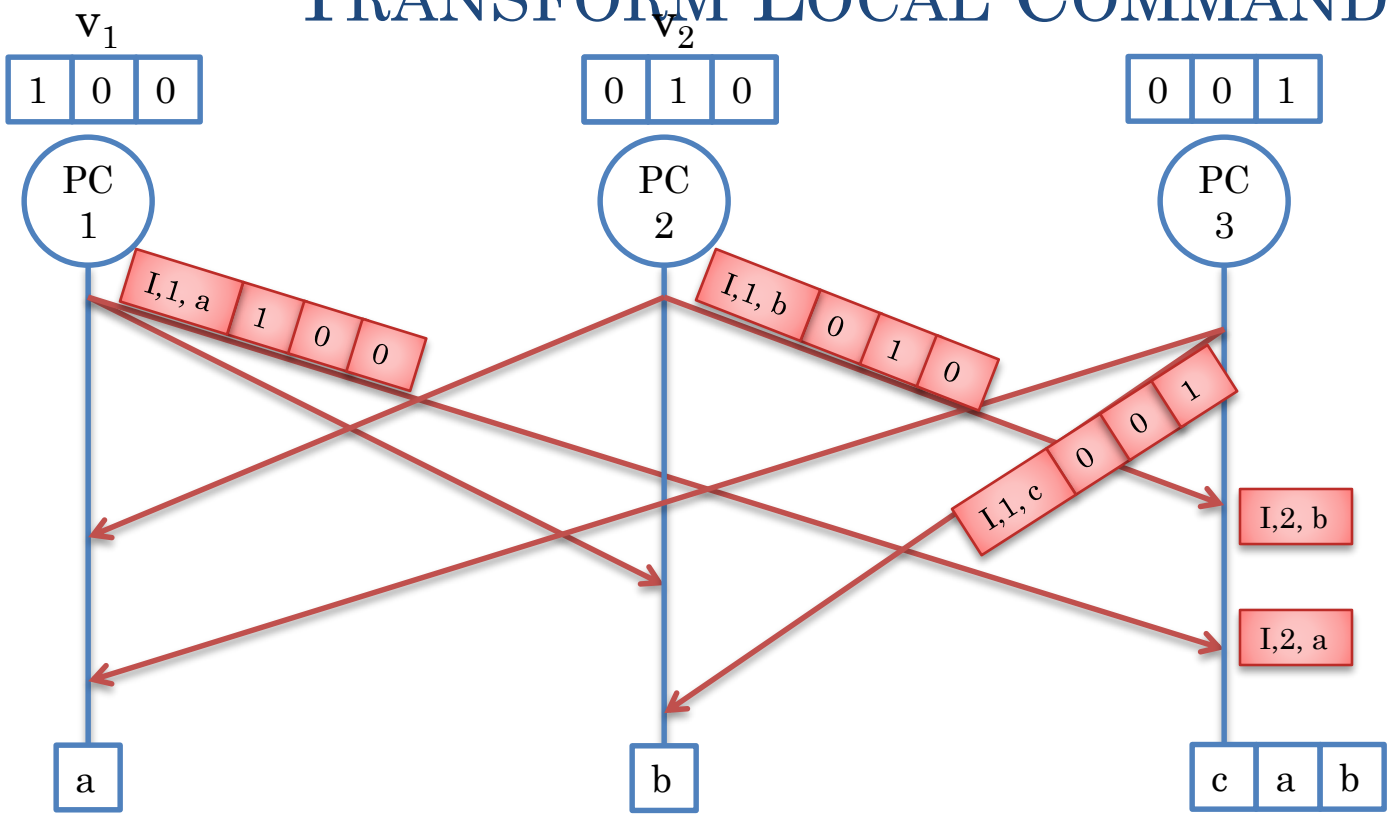
Local Buffer$_2$

I,1, b | 0 | 1 | 0

Local Buffer$_3$

I,1, c | 0 | 1 | 1

Transform Local Command



Local Buffer$_1$

I,1, a	1	0	0

Local Buffer$_2$

I,1, b	0	1	0

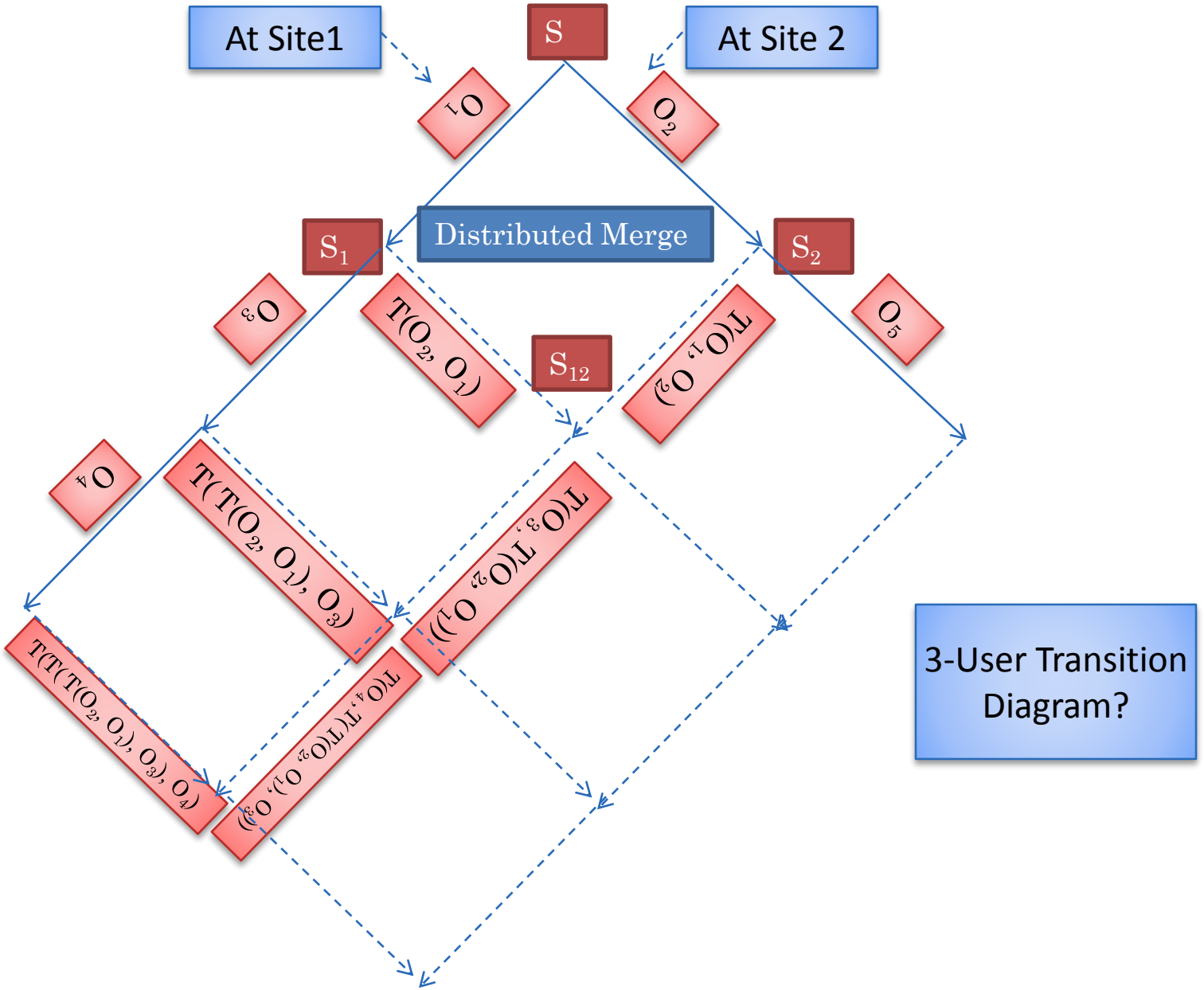
Local Buffer$_3$

I,1, c	1	1	1

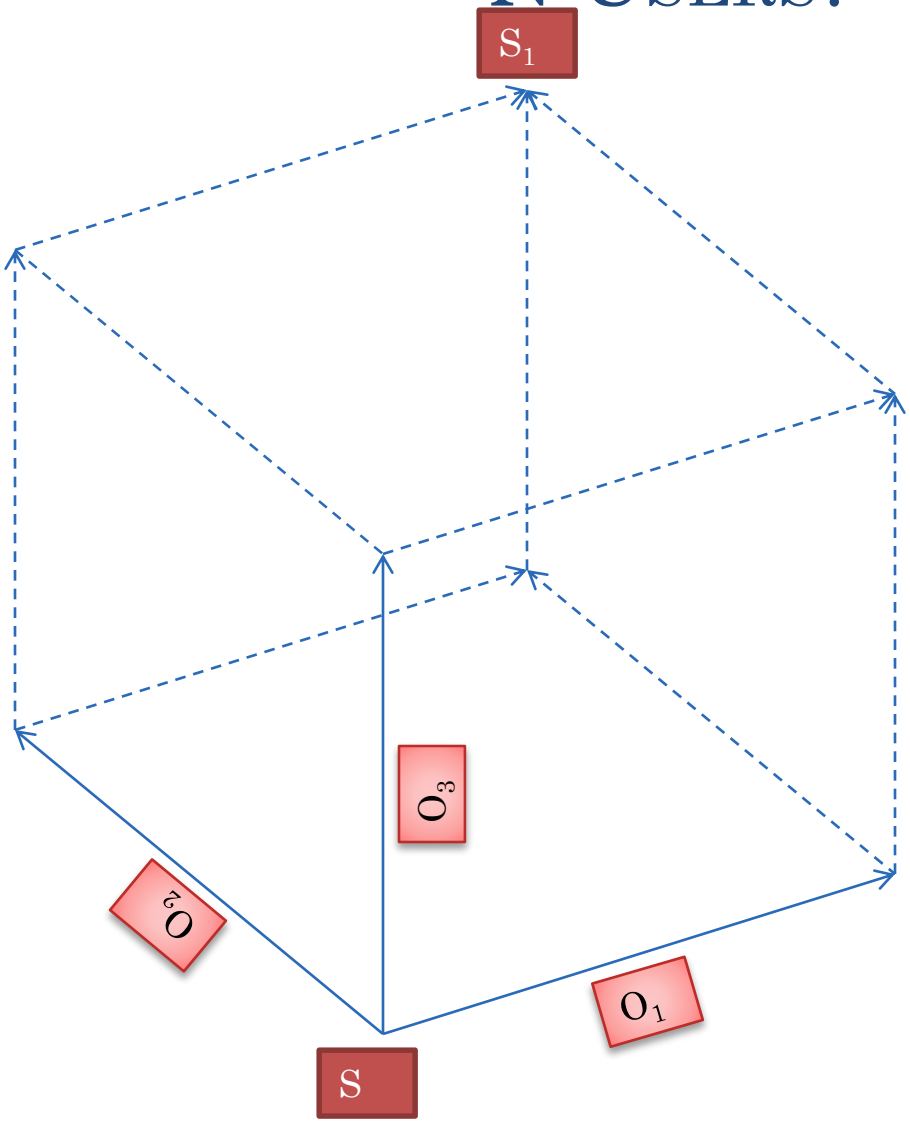
Problem with 2 Users

- Order of concurrent messages influences output.
- Same output not guaranteed at a single site.
- Same output not guaranteed at all sites.
- Problem independent of whether local operation is transformed.
- To understand better, need state transition diagram

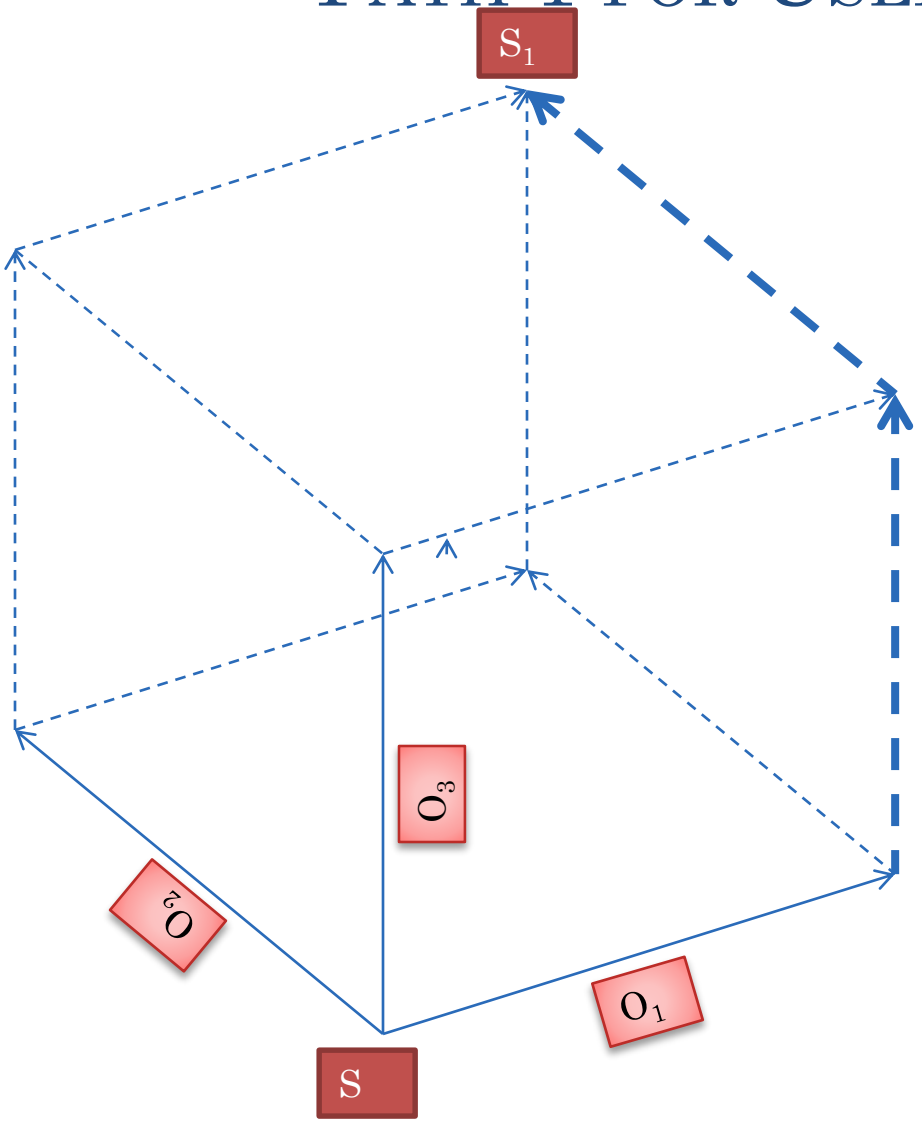
Multiple Remote Concurrent Operations



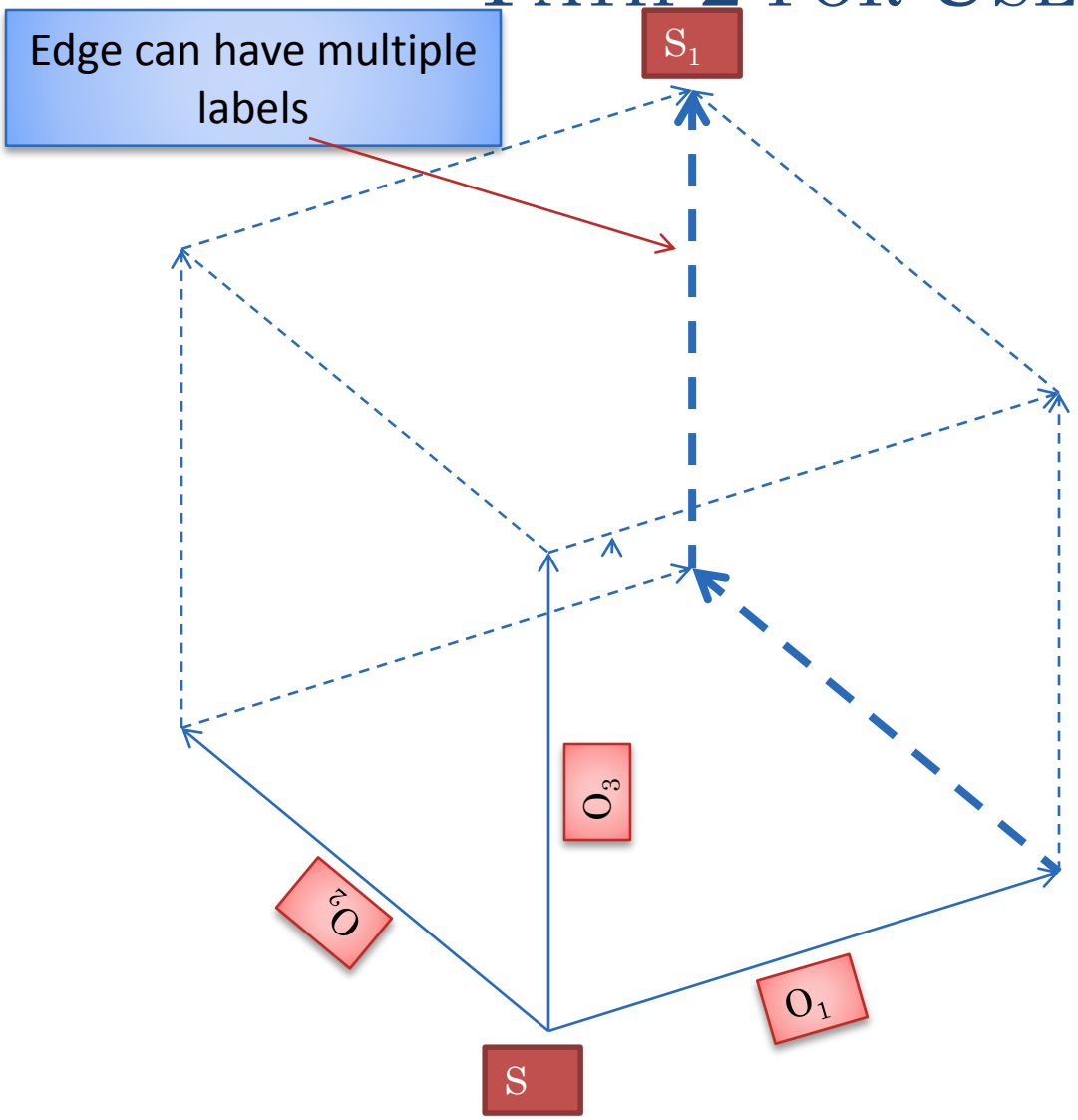
N-Users?



Path 1 for User 1



Path 2 for User 1



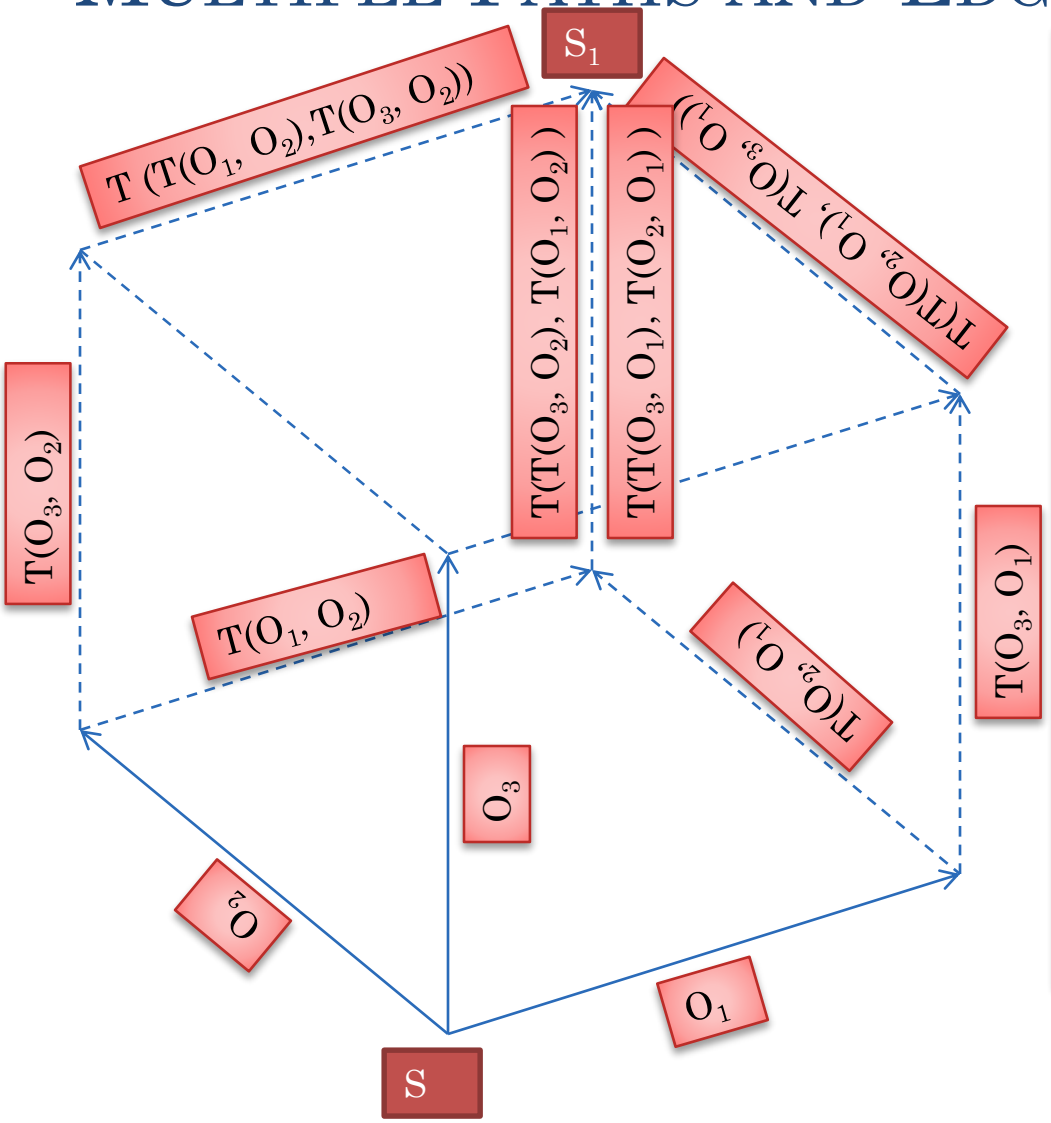
The two paths must give equivalent results.

In our example, our transformation functions did not!

Necessary condition for new transformation functions?

Each edge should have a unique label

Multiple Paths and Edge Labels



TP1

O_1	$T(O_2, O_1)$
==	
O_2	$T(O_1, O_2)$

TP2

$T(T(O_3, O_2), T(O_1, O_2))$

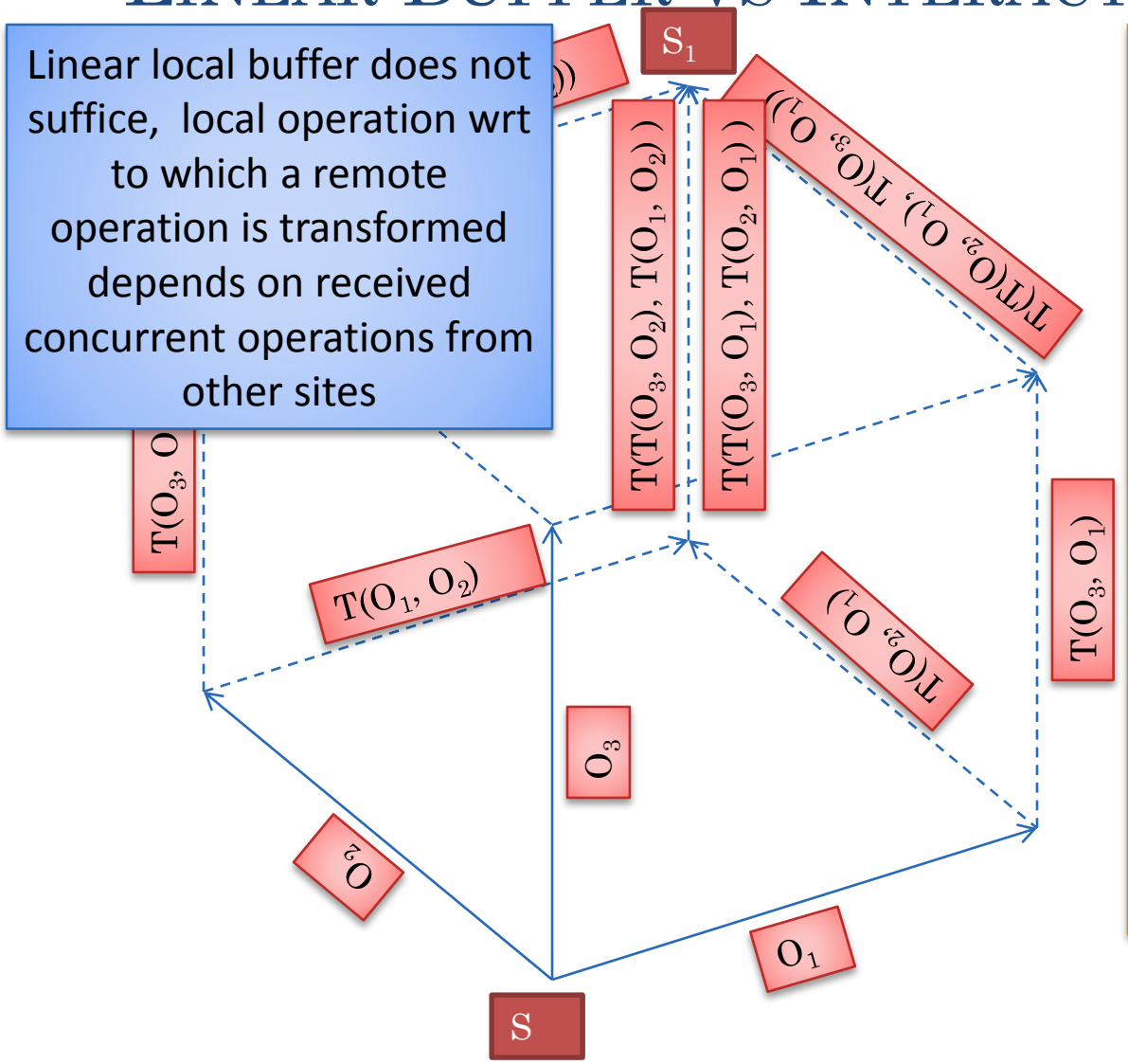
==

$T(T(O_3, O_1), T(O_2, O_1))$

Constraints for Transform

Linear Buffer vs Interaction Model

Linear local buffer does not suffice, local operation wrt to which a remote operation is transformed depends on received concurrent operations from other sites



TP1

O_1 $T(O_2, O_1)$

==

O_2 $T(O_1, O_2)$

TP2

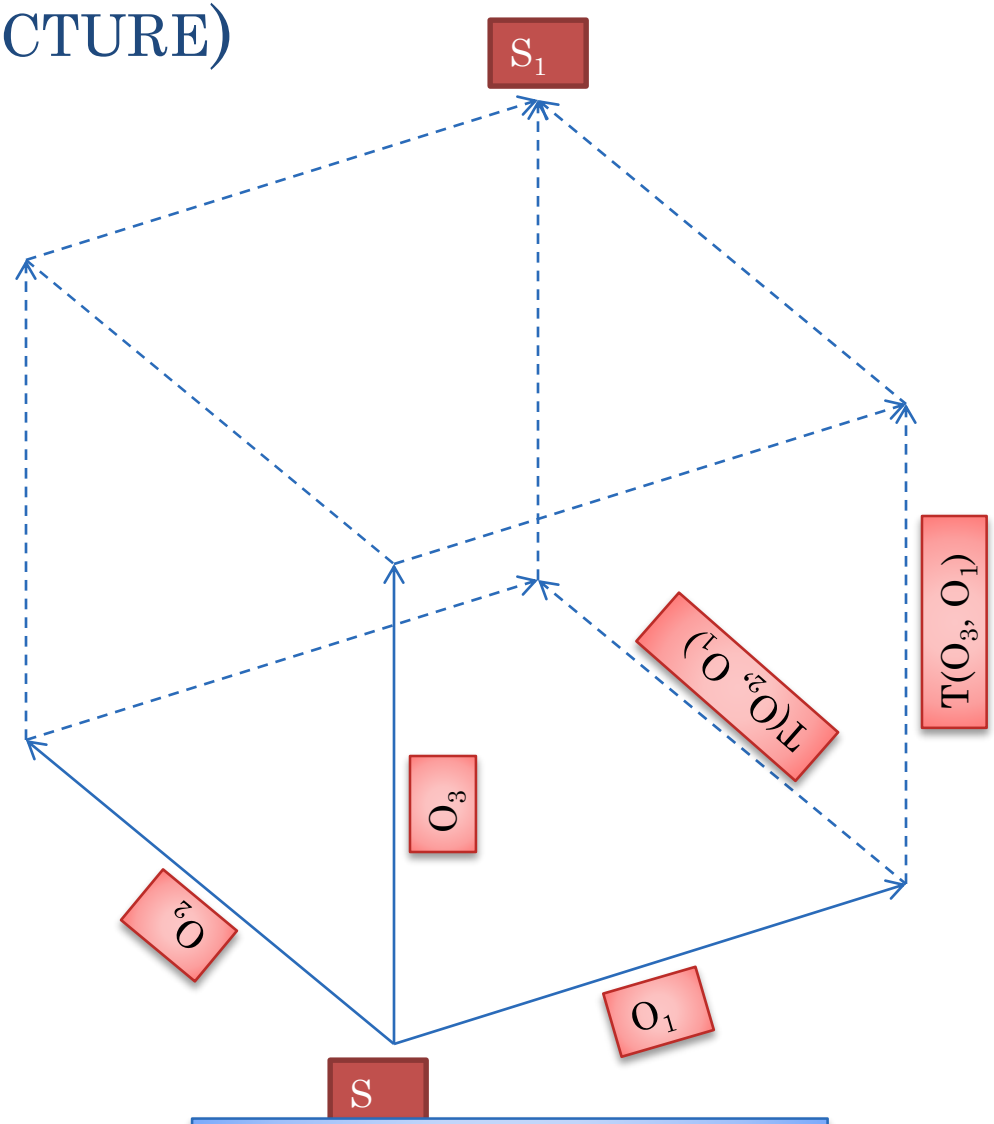
$T(T(O_3, O_2), T(O_1, O_2))$

==

$T(T(O_3, O_1), T(O_2, O_1))$

Constraints for Transform

Linear Buffer vs Interaction Model (Post Lecture)



O_1 no longer in buffer to compute $T(O^2, O^1)$

Multiple concurrent paths from a vertex, must store path

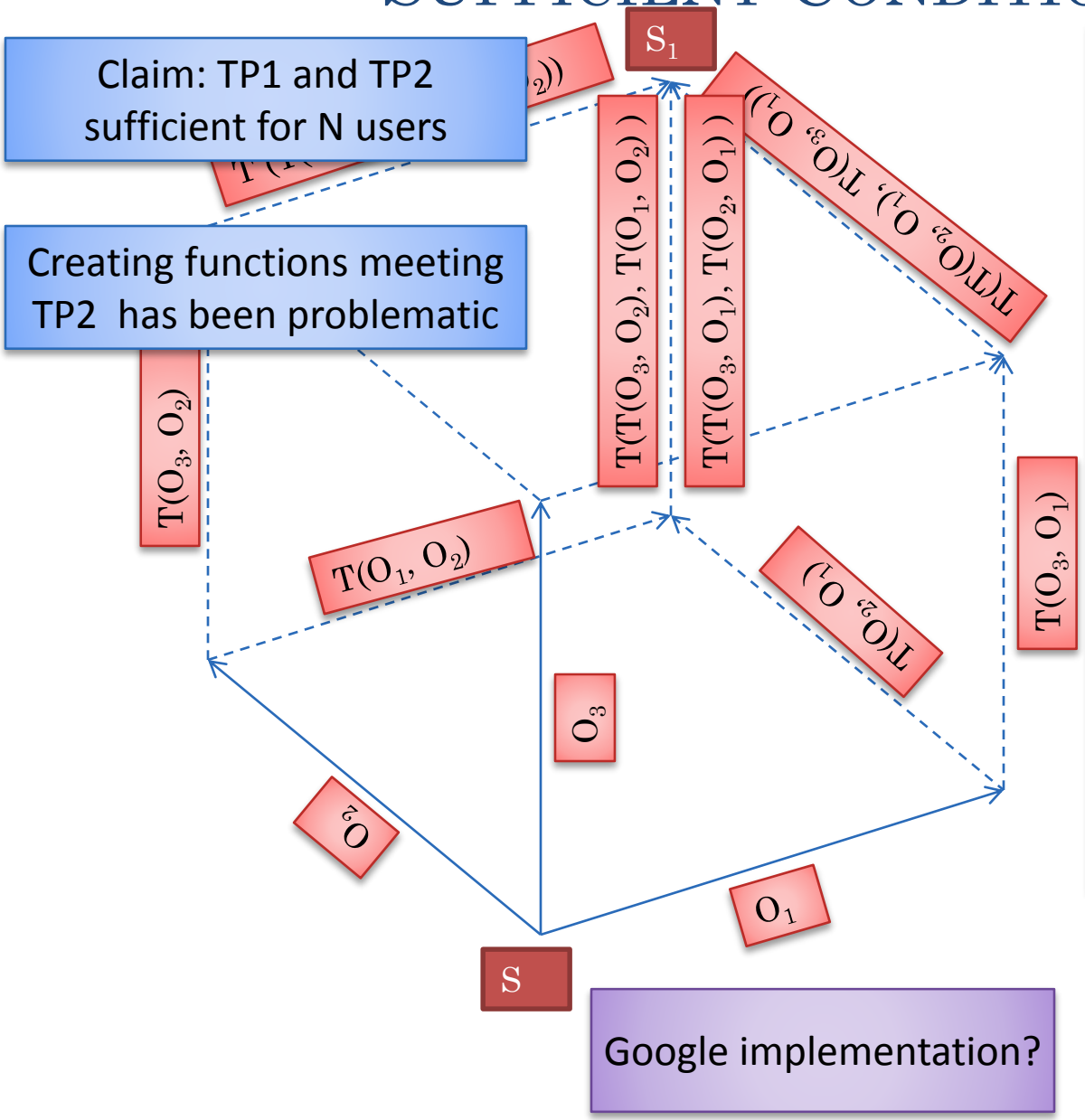
$T(O_1, O_3)$

Local Buffer$_1$

Sufficient Conditions?

Claim: TP1 and TP2 sufficient for N users

Creating functions meeting TP2 has been problematic



Google implementation?

TP1

O_1 $T(O_2, O_1)$

==

O_2 $T(O_1, O_2)$

TP2

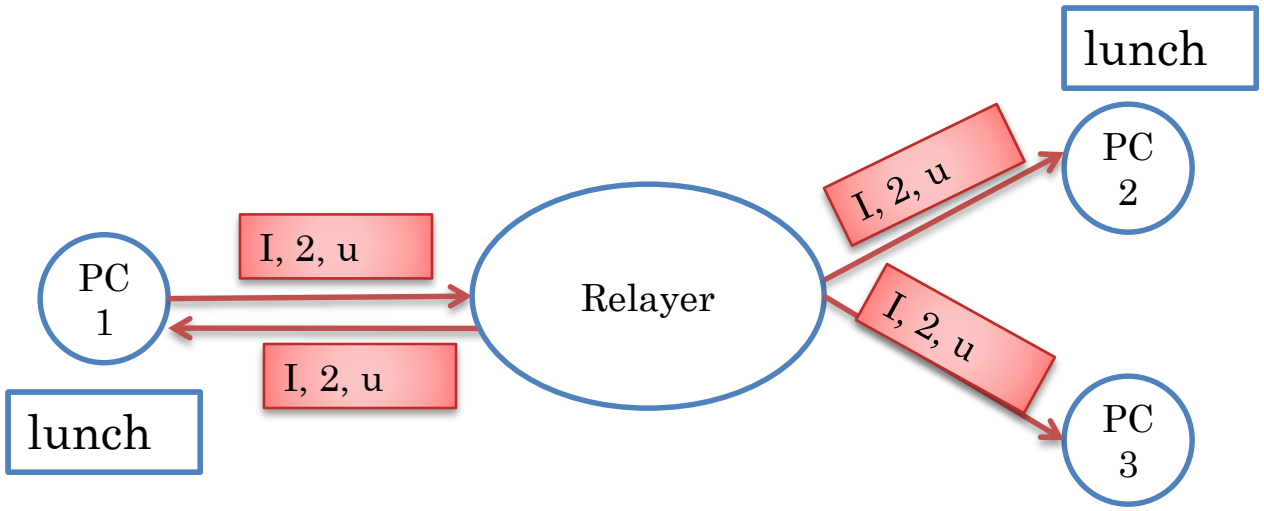
$T(T(O_3, O_2), T(O_1, O_2))$

==

$T(T(O_3, O_1), T(O_2, O_1))$

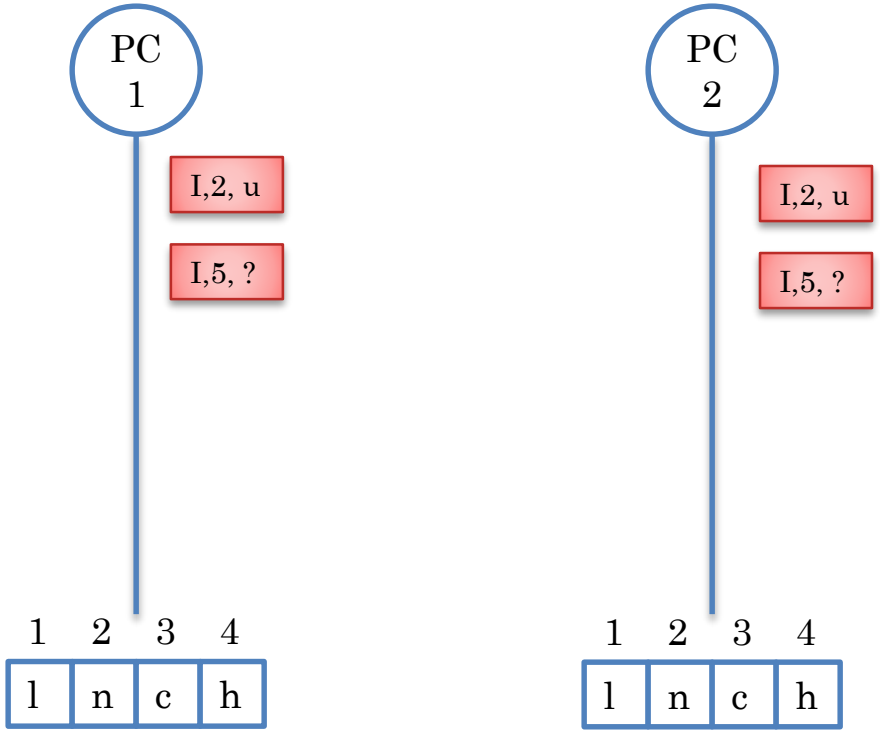
Constraints for Transform

Synchronous Relayed Broadcast (Review)

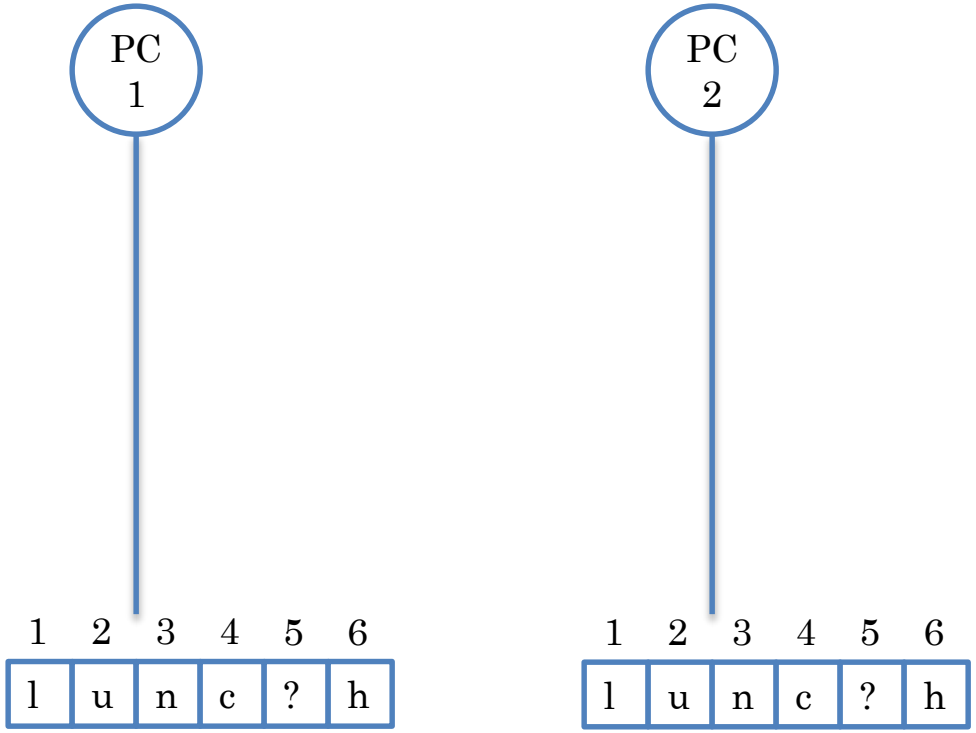


Delay (extra hop) state of operation issue not same as state of execution, though all sites are consistent

Ordering with Atomic Broadcast (Review)



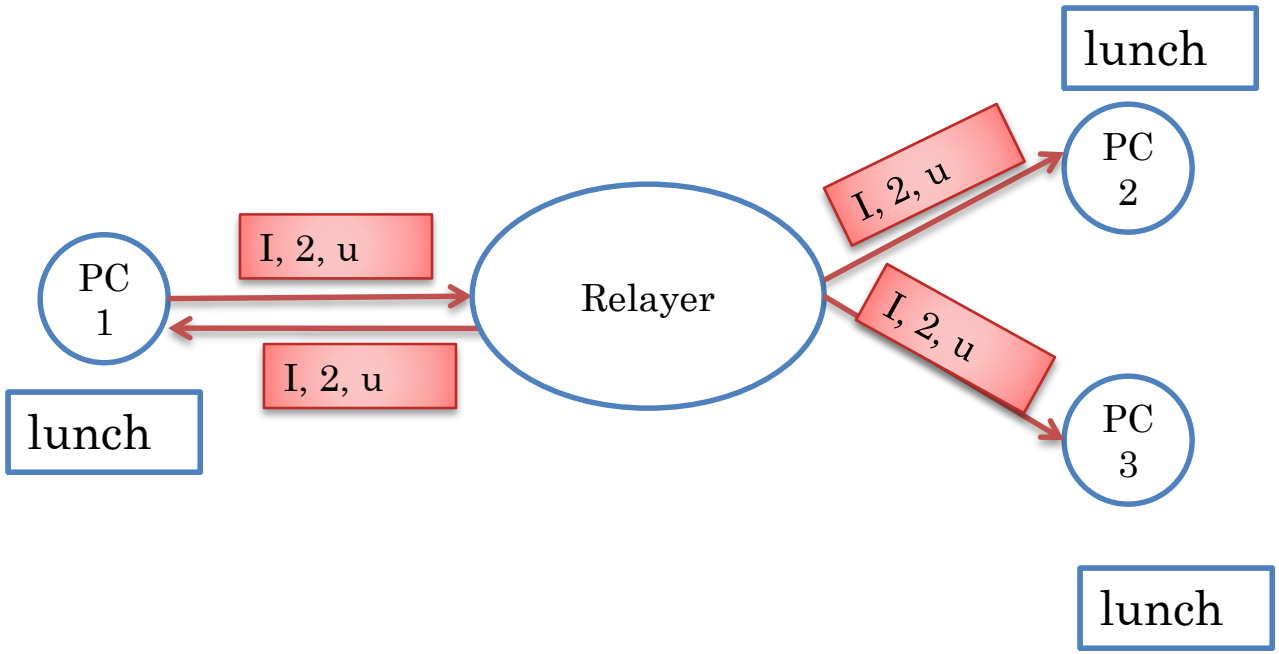
Second Operation Executes (Review)



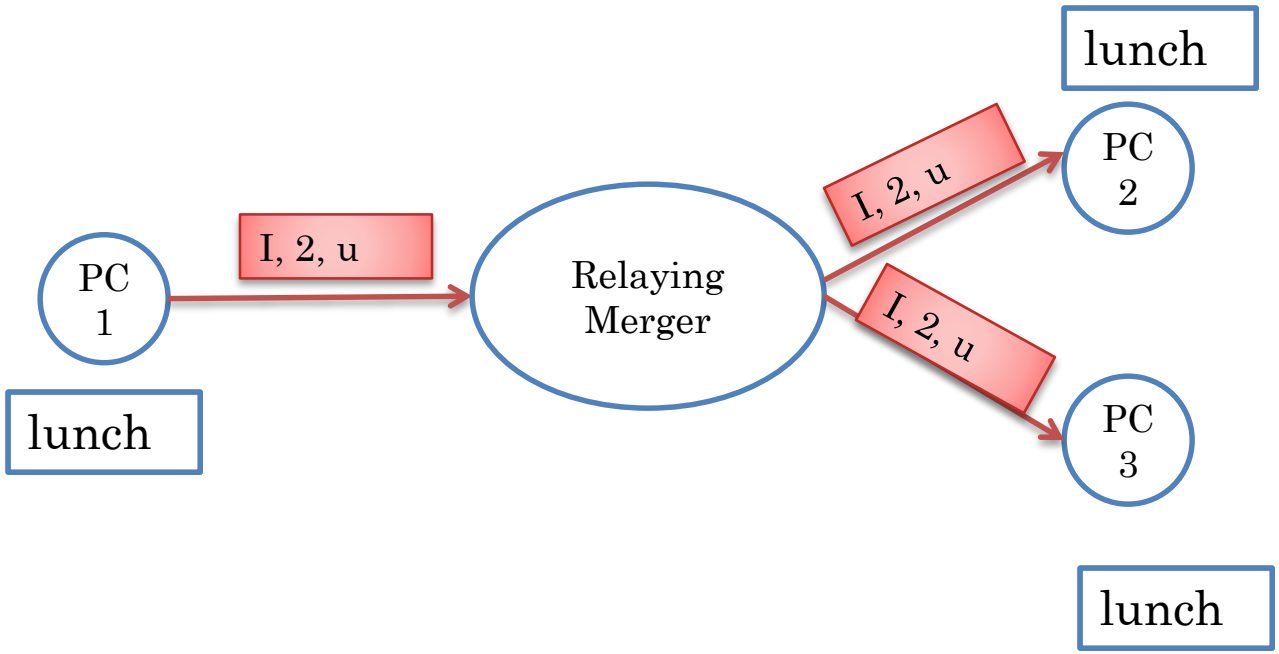
Common state but "intention" violation

Context of operation not the same as when it was issued

Synchronous Relayed Broadcast



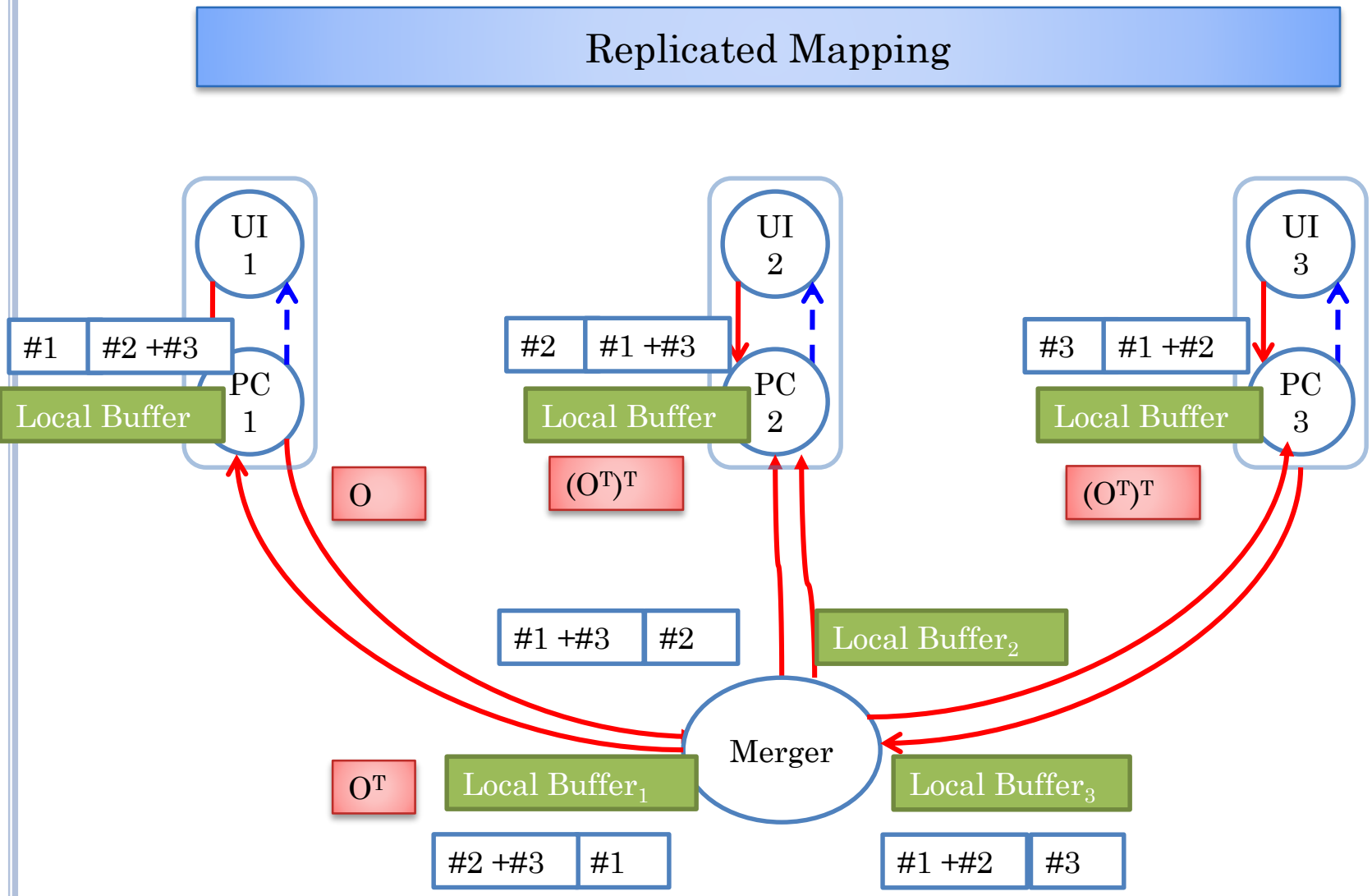
Asynchronous Merged Relayed Broadcast



2 to N Users

- Can do one N-user merge
- Can do N 2-User messages
  - Through a server
  - Each client is consistent with the server
  - Implies each client is consistent with the server
- But server does not issue any operations
  - For each client, server operations are those issued by other clients

Replicated Architecture With Central Merger : Local, Remote Time Stamp



Client and Server Receive Algorithm

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R^T = Transform (R, L)

L = Transform (L R)

L.TimeStamp.increment(R.site)

R= R^T

Execute R

Site.TimeStamp.increment(R.site)

Client

Given Remote op, R, concurrent with local ops L^1, L^2, .. LN

For each L

R^T = Transform (R, L)

L = Transform (L R)

L.TimeStamp.increment(R.site)

R= R^T

For all other sites assume server executed R

Site.TimeStamp.increment(R.site)

Server

Transform Operation for Client-Server Case

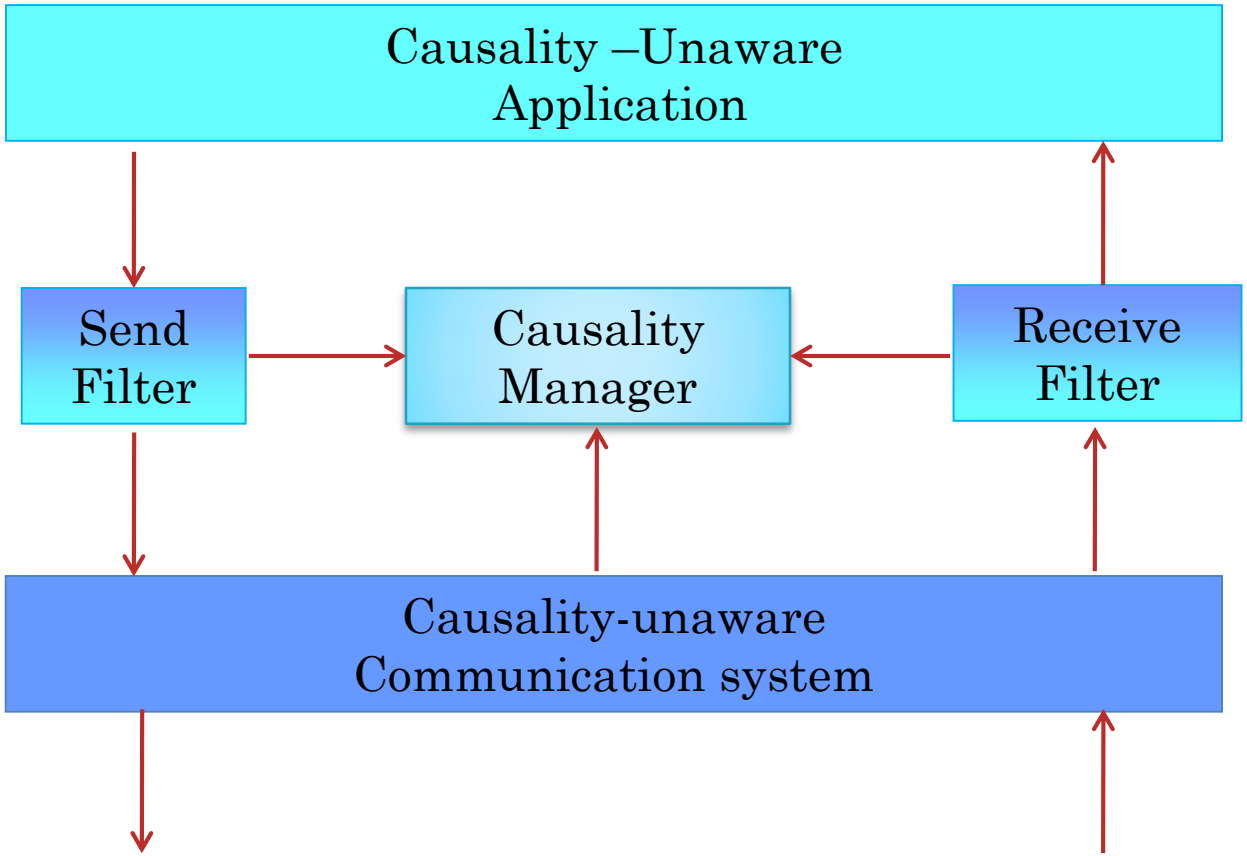
P2P

```
InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L) {
        Operation R^T = deepClone();
        if ((R.index > L.index) | |
           (R.index === L.index  && R.id  < L.id))
                R^T.index = R.index + 1;
        return R^T ;
}
```

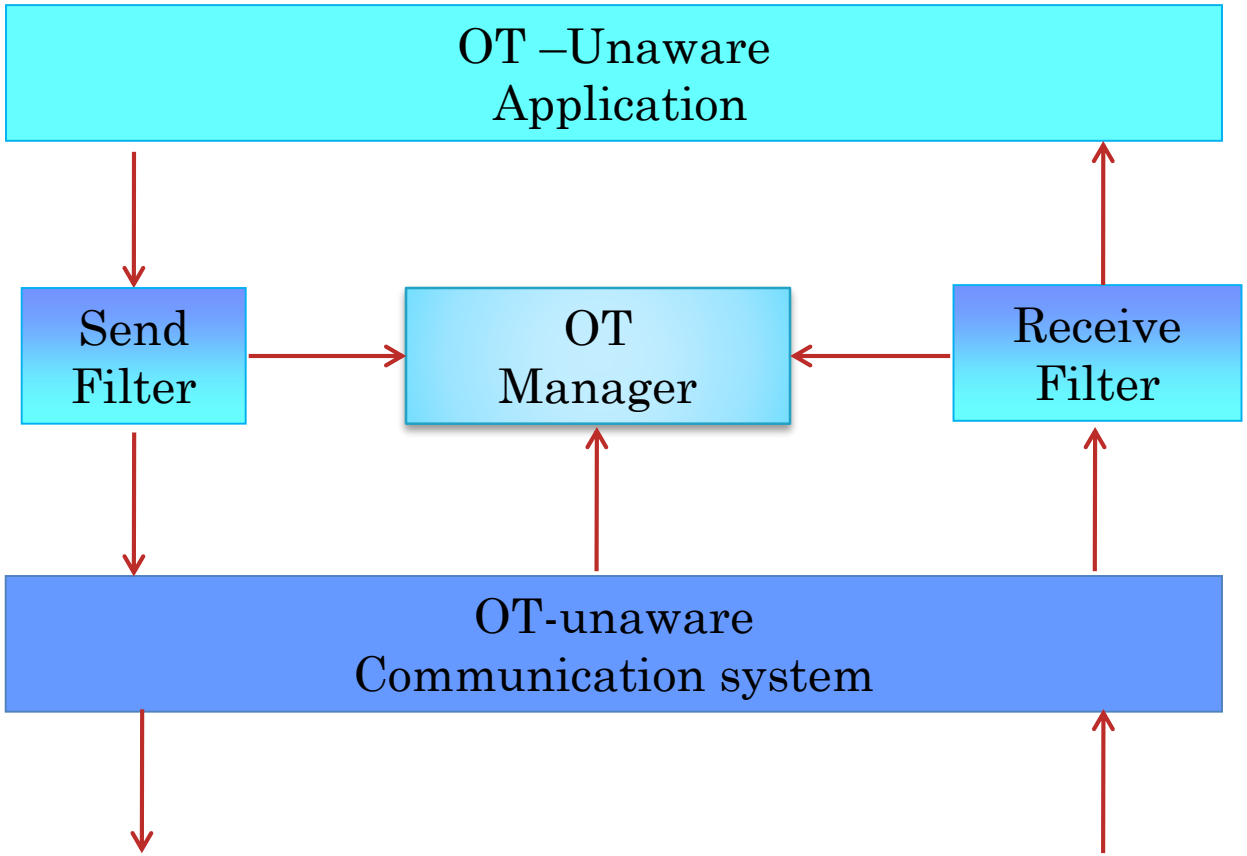
Client-Server

```
InsertOperation TransformInsertInsert (InsertOperation R, InsertOperation L) {
        Operation R^T = deepClone();
        if ((R.index > L.index) | |
           (R.index === L.index  && !R.isServer()))
                R^T.index = R.index + 1;
        return R^T ;
}
```

Causality Manager



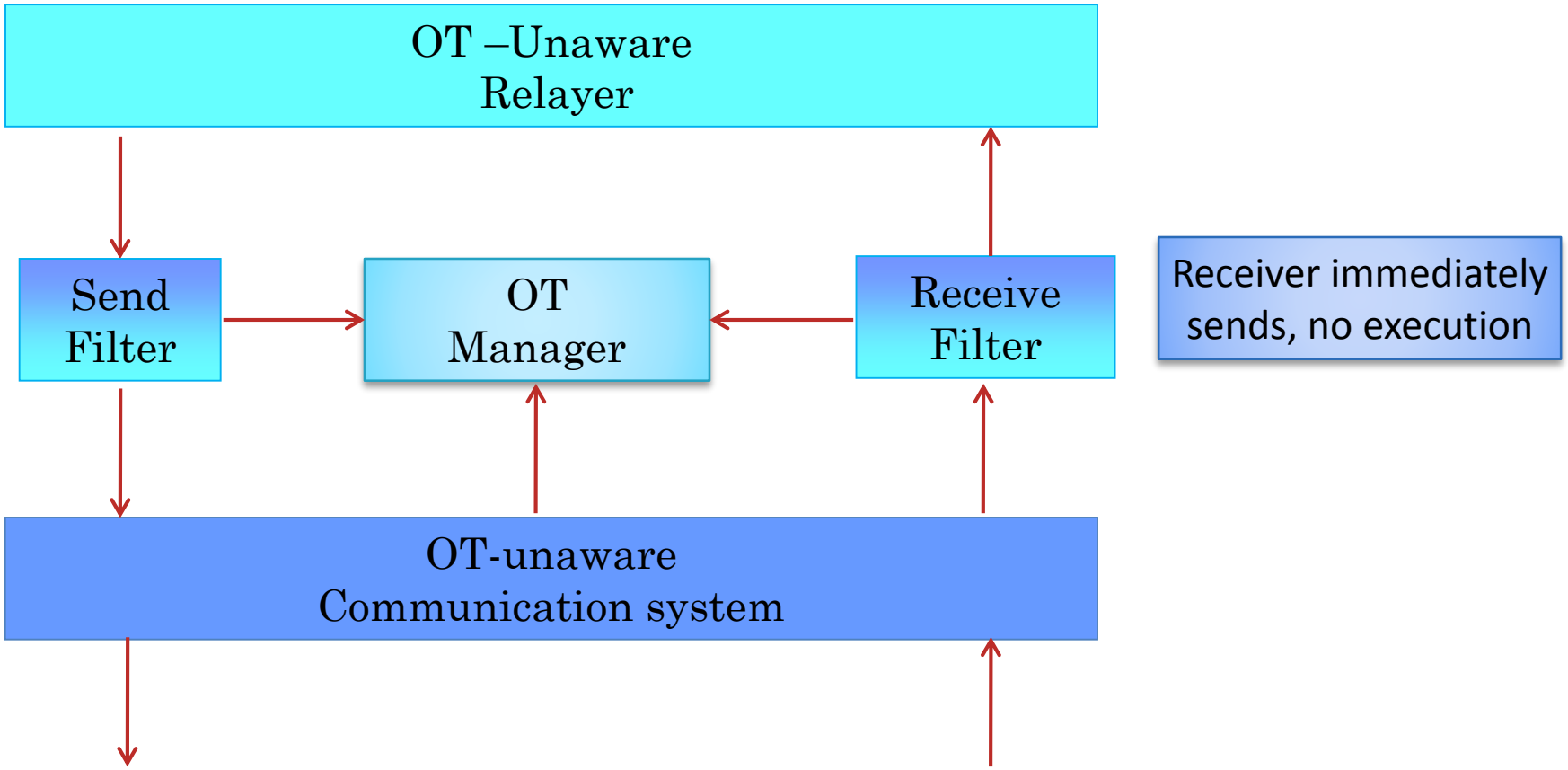
Client OT Manager



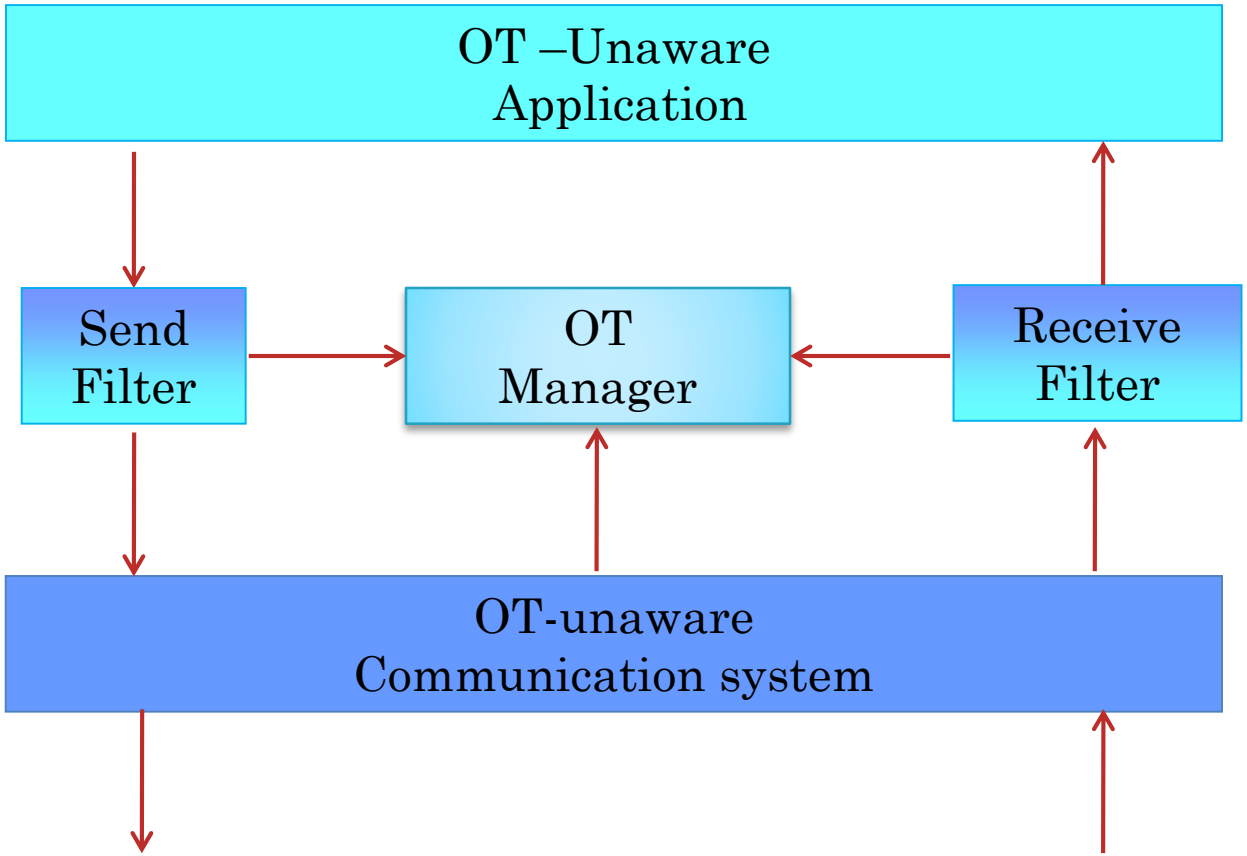
Single OT Manager?

One for each sequence

Server OT Manager



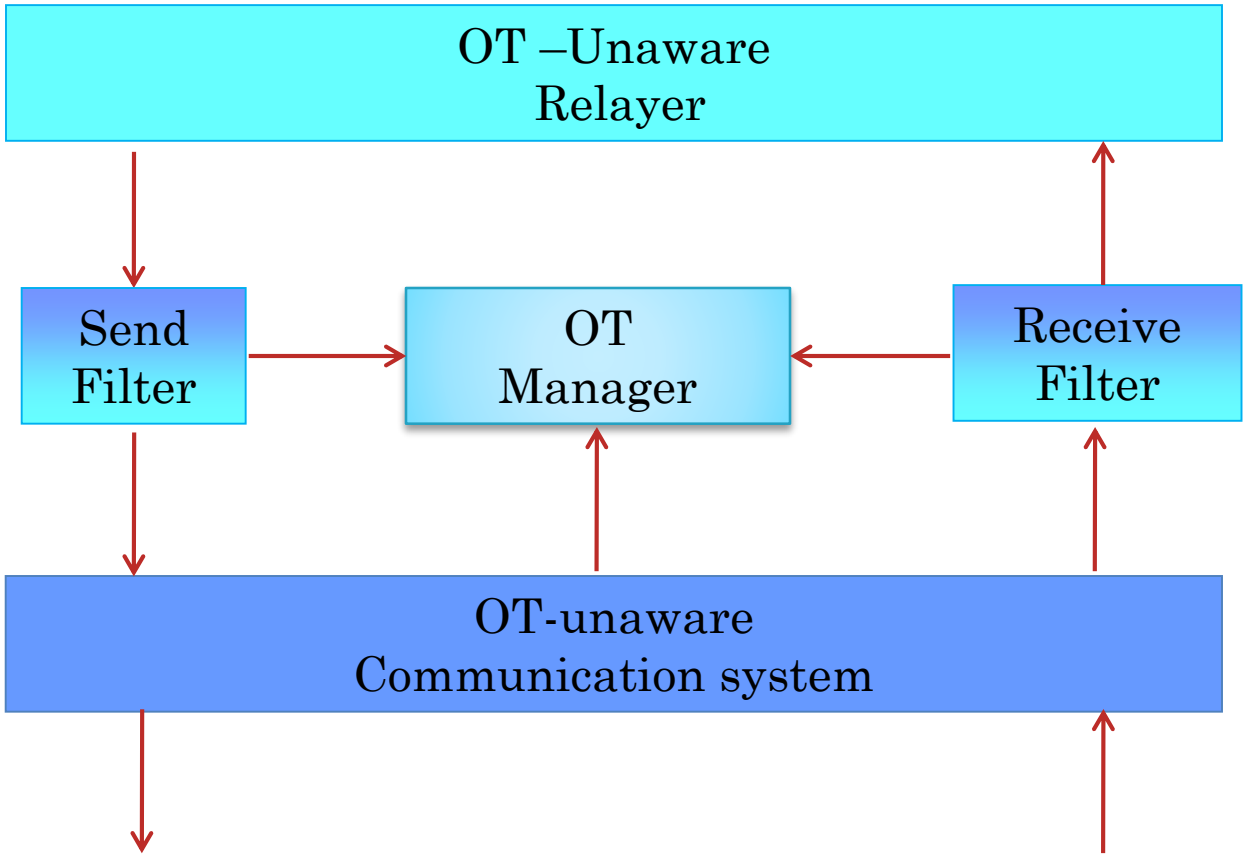
Client OT Manager (Review)



Single OT Manager?

One for each sequence

Single Server Filter



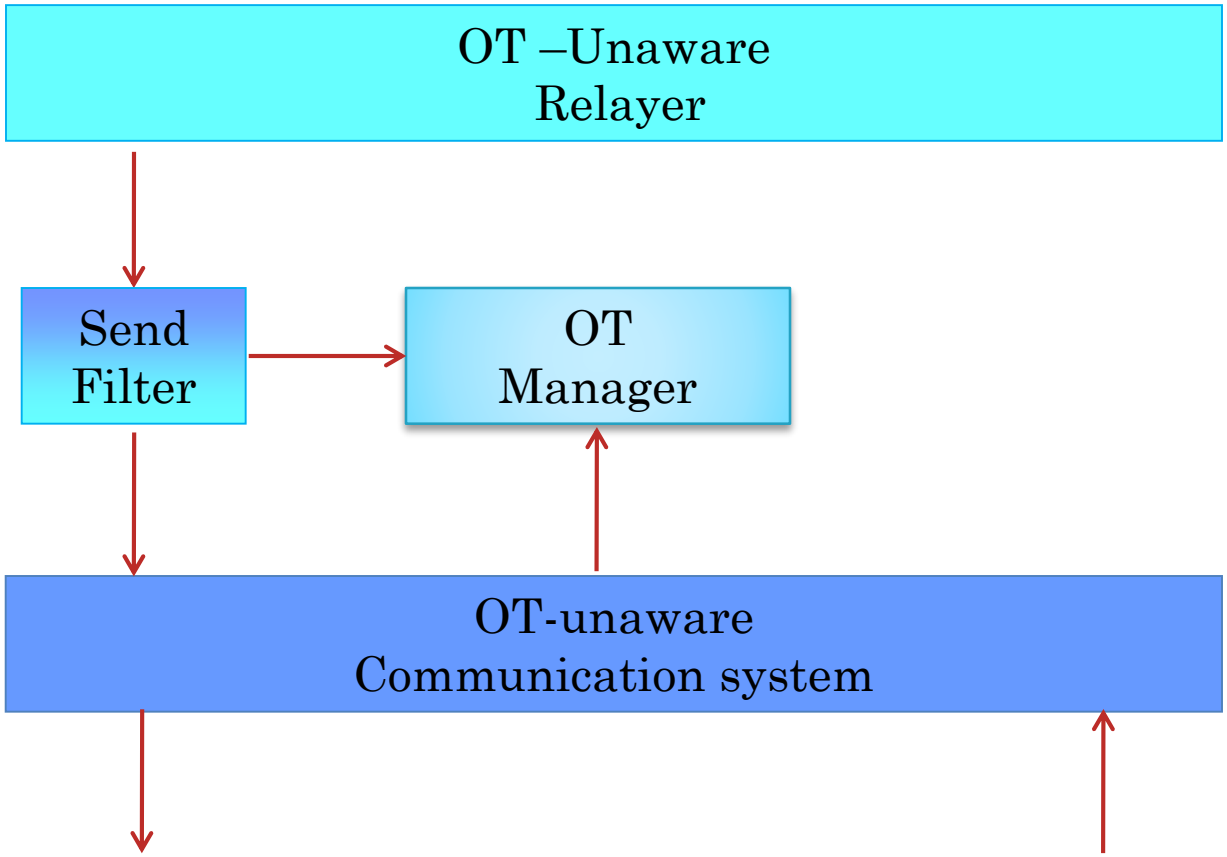
Receiver immediately sends, no execution

Can use only one filter

Receiver does not know identities of destinations but sender may (in current implementation filter called before multicasting)

Single send filter sufficient

Server OT Managers



One OT Manager for each client and sequence

How to attach send filter to server?

Message Filter Interface

```
public interface MessageFilter<MessageType> {
    public void setMessageProcessor (MessageProcessor<MessageType>
newVal;
    public void filterMessage(MessageType message);
}
```

Called by communication system when new message to be filtered available

Next stage in pipeline, processing the filtered message

ReceivedMessage or SentMessage

Called by communication system when pipeline setup

Server Message Filter Interface

```
public interface ServerMessageFilter extends MessageFilter<SentMessage> {
  public void userJoined(String aSessionName, String anApplicationName,
String userName);
  public void userLeft(String aSessionName, String anApplicationName,
String userName);

}
```

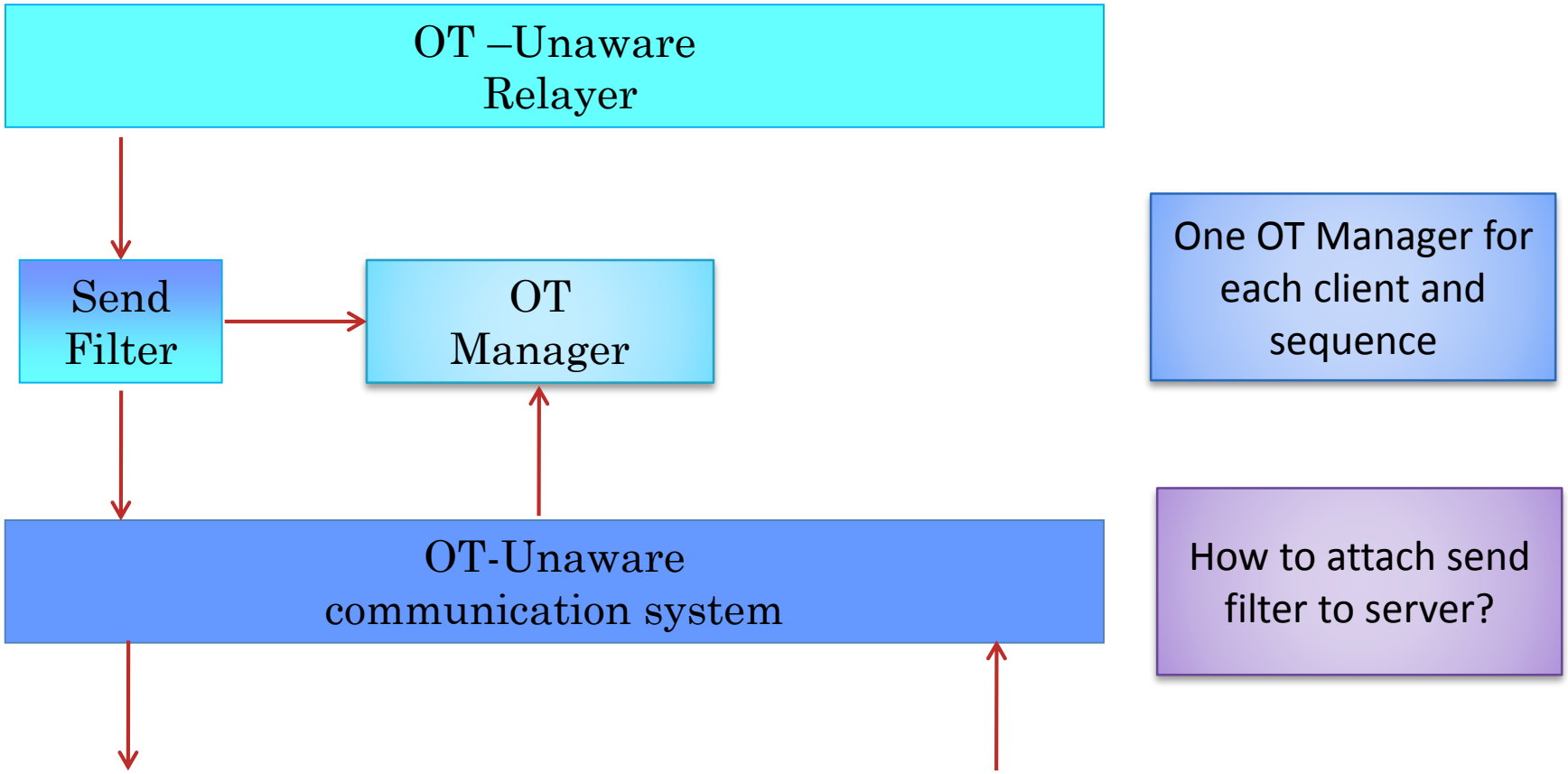
Server Factory Interface

```
public interface ServerMessageFilterCreator {
  ServerMessageFilter getServerMessageFilter();
}
```

Send Filter (Factory) Selector or Abstract Factory

```
public class SentMessageFilterSelector {
  static MessageFilterCreator<SentMessage> filterFactory =
     new AMessageForwarderCreator<SentMessage>();
  public static MessageFilterCreator<SentMessage> getMessageFilterCreator() {
    return filterFactory;
  }
  public static void setMessageFilterCreator(
    MessageFilterCreator<SentMessage> theFactory) {
    filterFactory = theFactory;
  }
}
```

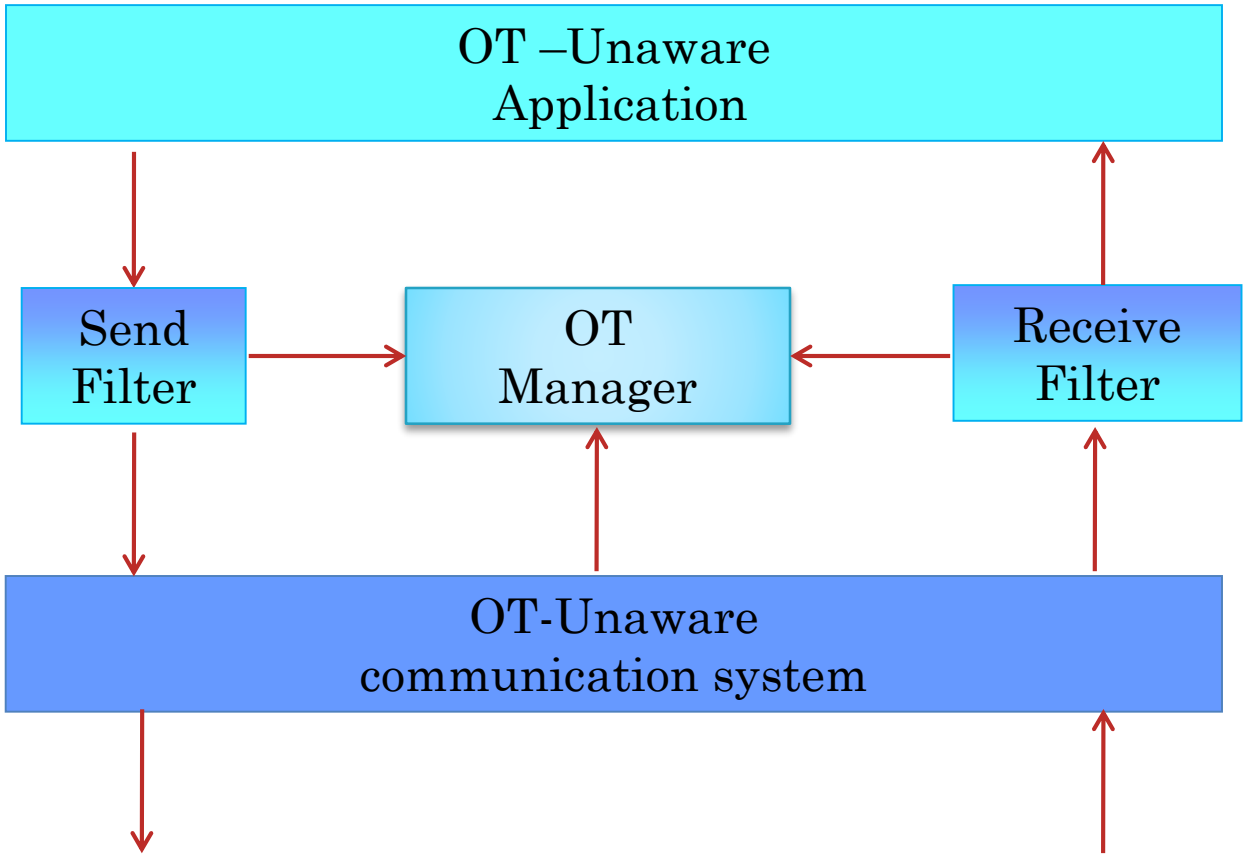
Server OT Managers



One OT Manager for each client and sequence

How to attach send filter to server?

Client OT Manager



Single OT Manager?

One for each sequence

Client Initialization

Init

Create <List, OT Manager> Mapping ListOTManager

For each List, L

ListOTManager(L) ← new OT Manager (ClientName, Not Server)

Create Client Send and Receive Filter Factories, passing them OTManager so they can pass them to the two filters

Send Filter Traceable Steps

Send Filter
On each user edit about OT List L
Ask ListOTManager(L) to time stamp edit
OTListEditSend the timestamped edit through message processor
Ask ListOTManager(L) to store copy of sent message

As in causality must ensure changing site time stamp does not change message time stamp

Receive Filter Traceable Steps

Receive Filter
On each OTListEditReceived for list L received (through server)
OTListEditFlipped time stamp
Ask OTManager(L) to transform received edit
Pass transformed edit to message processor

OT Manager: Init

Init (User Name, IsServer)

InitialOTTimeStampCreated

OT Manager: Send Steps

Time stamp edit

LocalSiteCountIncremented

Timestamp edit with local time stamp

Store Sent Message

MessageBuffered

OT Manager: Receive Filter Communication

Process received timestamped edit

For each local message not concurrent with received edit

Local MessageUnBuffered

For each local buffered concurrent edit L

L= TransformationResult from Transform(L, R)

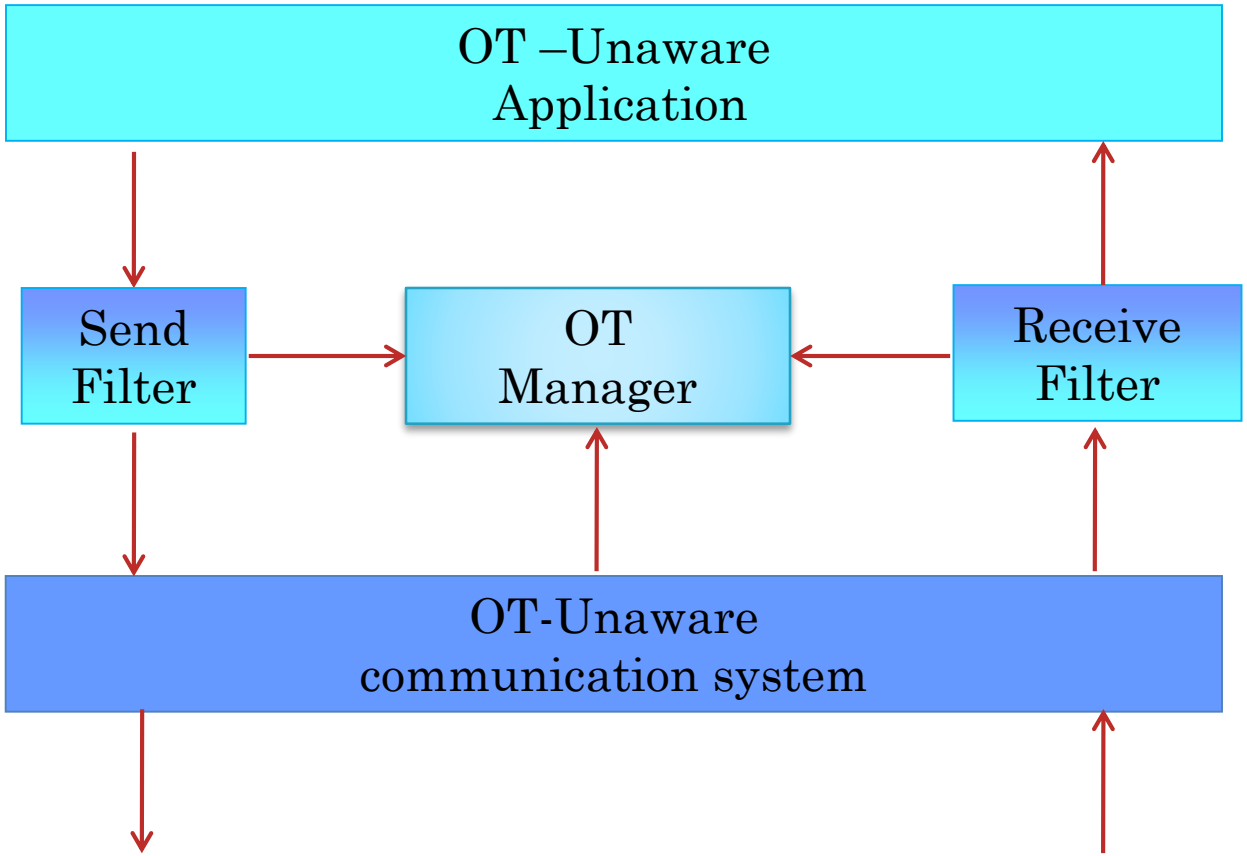
R= TransformationResult from Transform(R, L)

OTListEditRemoteCountIncremented in L

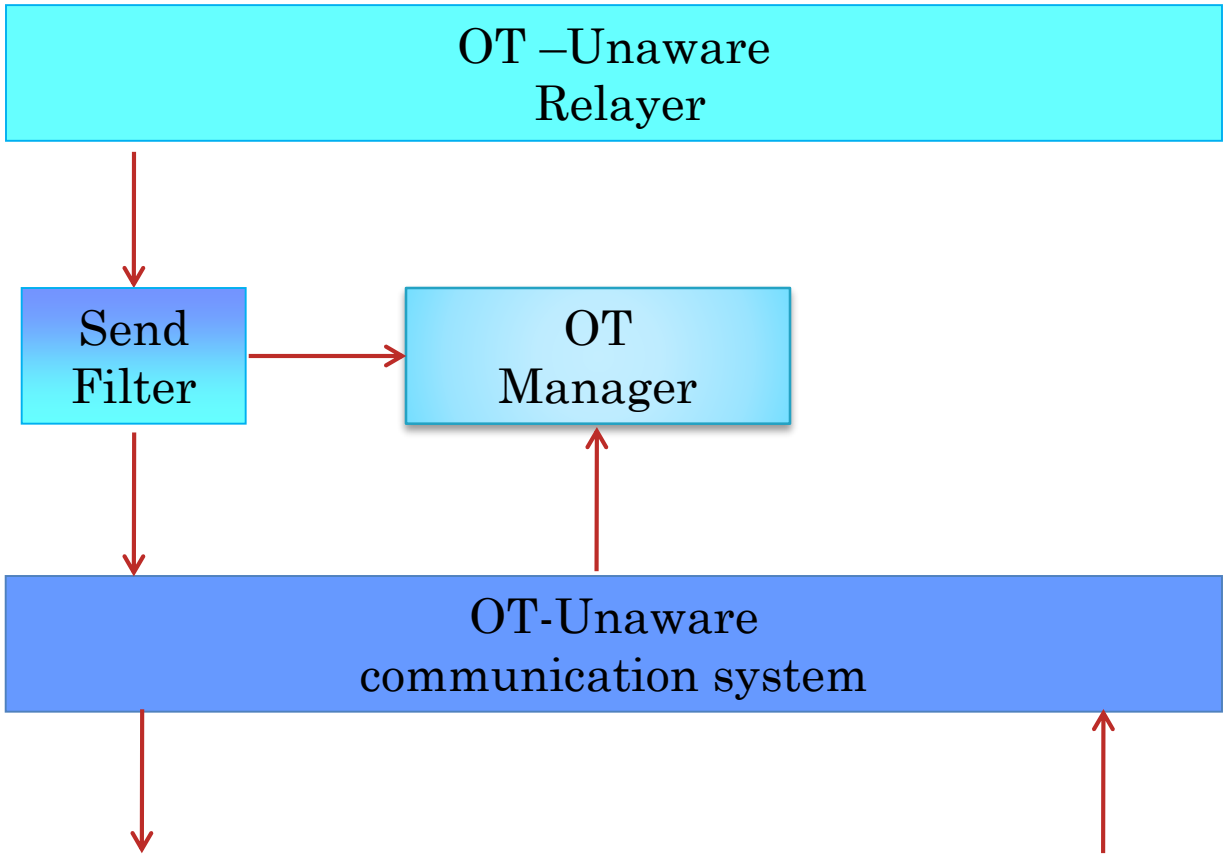
RemoteSiteCountIncremented

User name in trace step is name of user who executed the operation, for local edit, the local user, for remote edit, the remote user (exact name, not server)

Client OT Manager



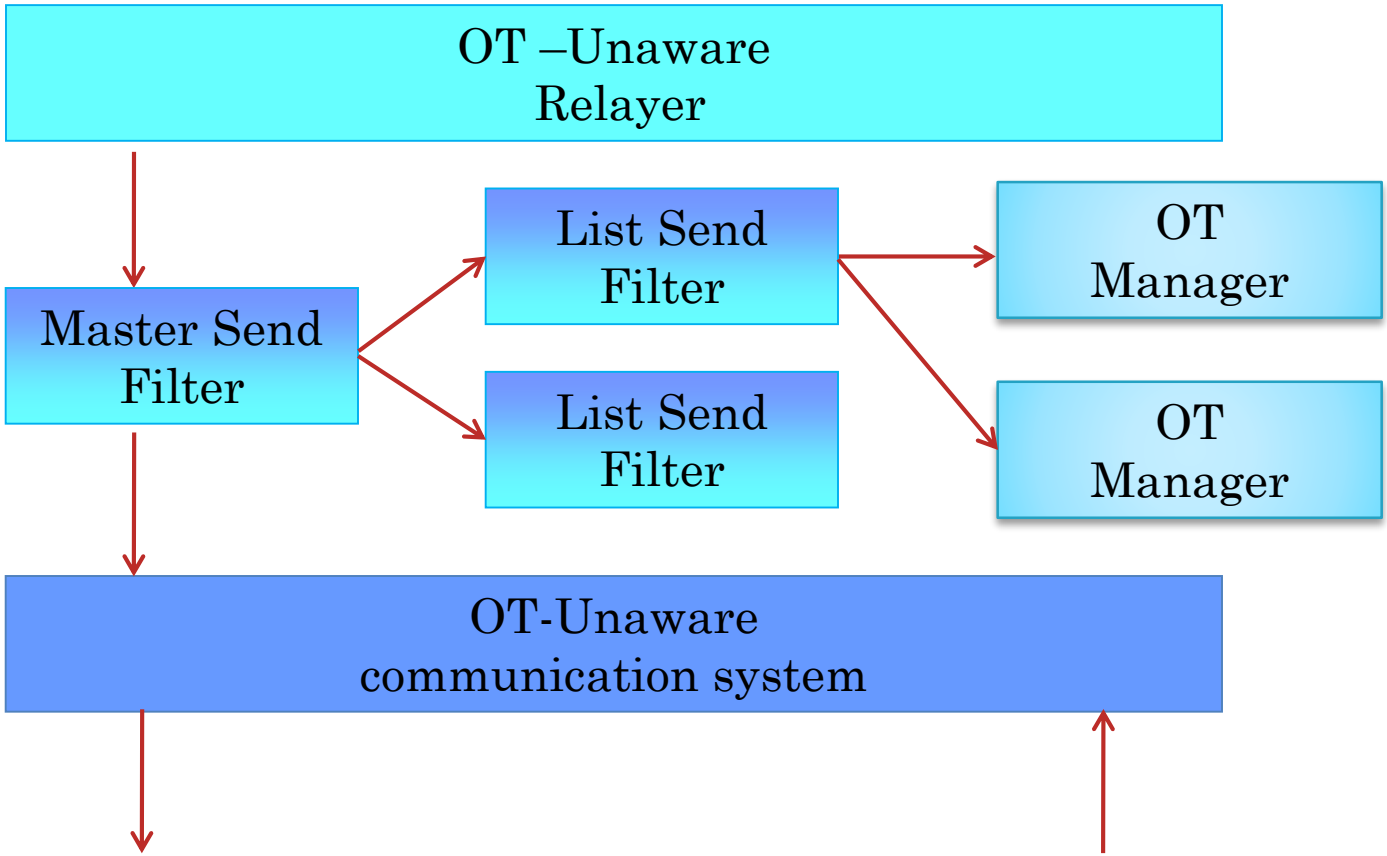
Server OT Managers



One OT Manager for each client and sequence

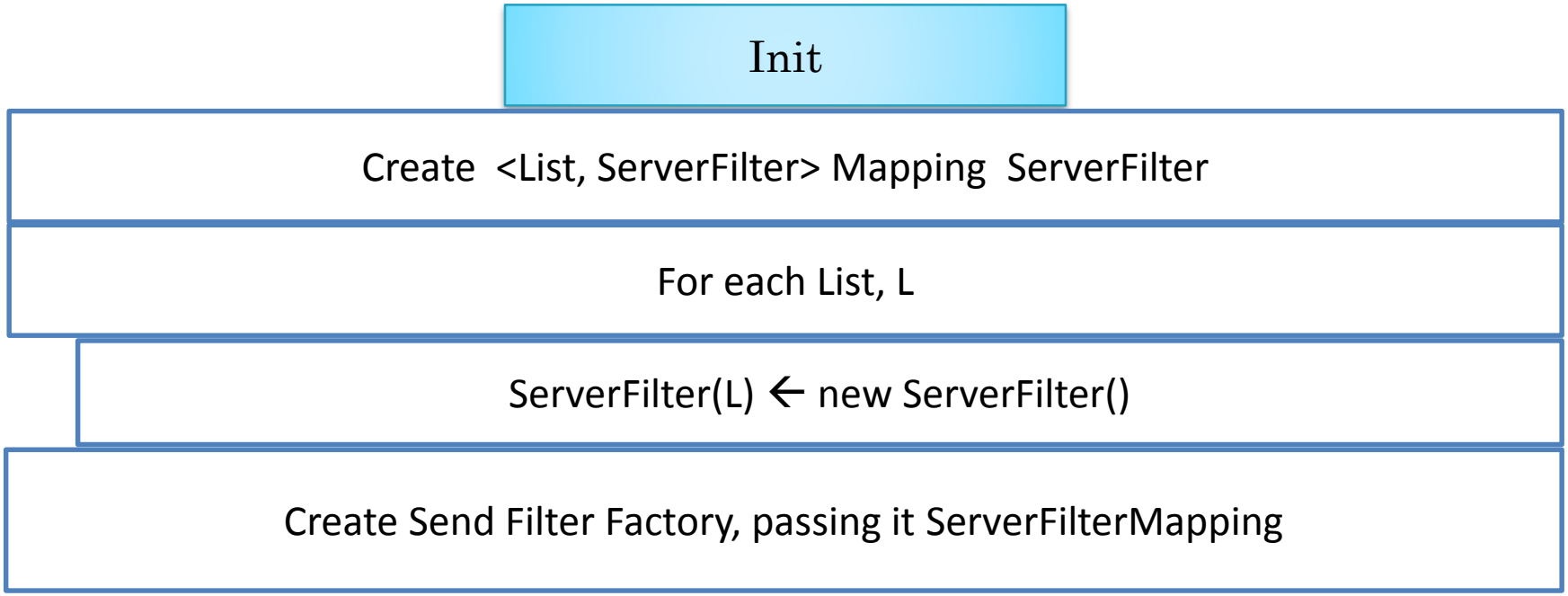
May want to extend one list implementation to multiple lists

Filter Composition



Could have used this architecture (master/delegate filters) also for clients, but client filters were simple and there were two of them, so it is not clear creating two additional master filters for clients is worth it

Server Initialization



Server Message Filter Interface

```
public interface ServerMessageFilter extends MessageFilter<SentMessage> {
  public void userJoined(String aSessionName, String anApplicationName,
String userName);
  public void userLeft(String aSessionName, String anApplicationName,
String userName);

}
```

Master Server Send Filter: Forwarding

Join

For each list L

ServerFilter(L).userJoined()

Leave

For each list L

ServerFilter(L).userLeft()

Set Message Processor

For each list L

ServerFilter(L).setMesageProcessor()

Master Server Send Filter

New Message

On each client edit of list L

ServerFilter(L).filterMessage()

Server Send Filter

Init

Create <User, OTManager> Mapping UserOTManager

Join

On join of each user U

UserOTManager(U) ← new OTManager(U, Is Server)

Server Send Filter

New Message

On each OTListEditReceived from Sending User S

OTListEditFlipped time stamp

Ask UserOTManager(S) to create transform received edit, O^T

For each user R in UserOTManager other than S

Create unicast copy of message containing, O^{TR}
By calling ASentMessage.toSpecificUser(message, R)

Ask UserOTManager(R) to time stamp O^{TR}

Send timestamped edit through message processor

Ask ListOTManager(R) to store copy of sent message

Centralized Algorithm

- Assume all users merge through central server
  - Output is produced locally immediately
- Server keeps local buffer and timestamp for each client
- Each client treats server as second user and sends it each command
- Instead of applying (possibly transformed) command to its local state server sends time-stamped command to each remote client
- Client transforms it further if it has executed concurrently
- Client assumes command executed directly by server
- Each client consistent with server, and thus with each other client
- Unique ordering of all commands from remote machines

History

- dOPT (Distributed Operation Transformation) – Ellis and Gibbs '89
  - Did not transform local operation
  - Had known problem with multiple users
- adOPTed (Ressel et al '96)
  - Transformed local operation
  - Give conditions for N-user replicated merging
- Jupiter (Nichols, Curtis, et al '95)
  - Centralized merging
  - Inventors of LiveMeeting
  - Implemented in GoogleWave