

Specification and Implementation of Replicated List

— The Jupiter Protocol Revisited

(Brief Announcement at PODC'2018)

Hengfeng Wei, Yu Huang, Jian Lu

Nanjing University

July 24, 2018



We claim one thing in this BA:

The **Jupiter** protocol [Nichols et al., 1995]¹ for replicated list **satisfies** the **weak list specification** [Attiya et al., 2016]².

¹David A. Nichols et al. (1995). “High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System”. In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. ACM, pp. 111–120.

²Hagit Attiya et al. (2016). “Specification and complexity of collaborative text editing”. In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. ACM, pp. 259–268.

We claim one thing in this BA:

The **Jupiter** protocol [Nichols et al., 1995]¹ for replicated list **satisfies** the **weak list specification** [Attiya et al., 2016]².

This was proposed as a conjecture in a PODC paper [Attiya et al., 2016].

¹David A. Nichols et al. (1995). “High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System”. In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. ACM, pp. 111–120.

²Hagit Attiya et al. (2016). “Specification and complexity of collaborative text editing”. In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. ACM, pp. 259–268.

Background for the Conjecture

Collaborative Text Editing Systems



(a) Google Docs



(b) Apache Wave



(c) Wikipedia



(d) \LaTeX Editor

Replication (for availability)



Replication (for availability)



- ▶ Replicas respond to user operations **immediately**
 - ▶ Updates are propagated **asynchronously**

List: to model the core functionality

$\text{INS}(a, p)$: Insert a at position p .

$\text{DEL}(p)$: Delete the element at position p .

READ : Return the list.

List: to model the core functionality

$\text{INS}(a, p)$: Insert a at position p .

$\text{DEL}(p)$: Delete the element at position p .

READ : Return the list.

To implement a highly available replicated list object.

Definition (Eventual Convergence (EC) [Ellis and Gibbs, 1989])

The lists at all replicas are identical *at quiescence*.



Definition (Strong Eventual Consistency (SEC) [Shapiro et al., 2011])

The lists at the replicas that *have executed the same set of user operations* are identical.

Definition (Eventual Convergence (EC) [Ellis and Gibbs, 1989])

The lists at all replicas are identical *at quiescence*.



Definition (Strong Eventual Consistency (SEC) [Shapiro et al., 2011])

The lists at the replicas that *have executed the same set of user operations* are identical.

Specify little on *intermediate states* going through by replicas.

Specification and Complexity of Collaborative Text Editing

Hagit Attiya
Technion

Sebastian Burckhardt
Microsoft Research

Alexey Gotsman
IMDEA Software Institute

Adam Morrison
Technion

Hongseok Yang
University of Oxford

Marek Zawirski^{*}
Inria & Sorbonne Universités,
UPMC Univ Paris 06, LIP6

Strong/weak list specification [Attiya et al., 2016]

Specify global properties on all states at all replicas.

Specification and Complexity of Collaborative Text Editing

Hagit Attiya
Technion

Sebastian Burckhardt
Microsoft Research

Alexey Gotsman
IMDEA Software Institute

Adam Morrison
Technion

Hongseok Yang
University of Oxford

Marek Zawirski*
Inria & Sorbonne Universités,
UPMC Univ Paris 06, LIP6

Strong/weak list specification [Attiya et al., 2016]

Specify global properties on all states at all replicas.

Proved: RGA [Roh et al., 2011] satisfies the strong list specification.

Conjecture: Jupiter [Nichols et al., 1995] satisfies the weak list specification.

Weak List Specification

Definition (Weak List Specification $\mathcal{A}_{\text{weak}}$ [Attiya et al., 2016])

Informally, $\mathcal{A}_{\text{weak}}$ requires the ordering between **elements that are not deleted** to be consistent across the system.

Definition (Weak List Specification $\mathcal{A}_{\text{weak}}$ [Attiya et al., 2016])

Informally, $\mathcal{A}_{\text{weak}}$ requires the ordering between **elements that are not deleted** to be consistent across the system.

Pairwise state compatibility property:

*For any pair of list states, there **cannot** be two elements a and b such that a precedes b in one state but b precedes a in the other.*

Jupiter

S



C_1



C_2

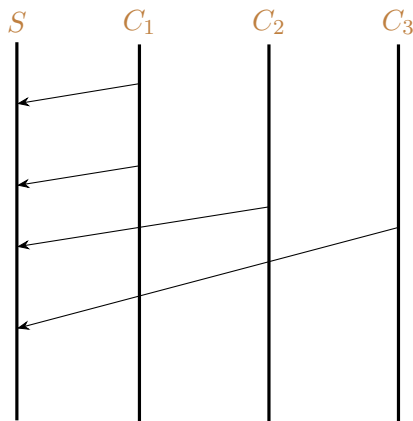


C_3



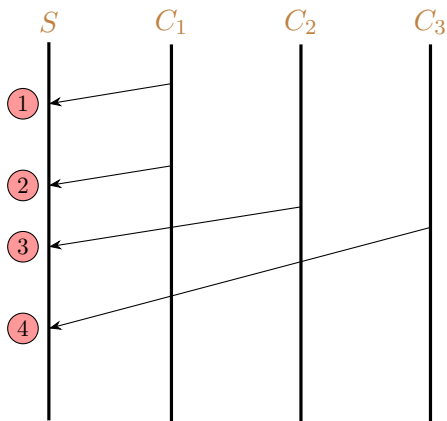
System model of Jupiter [Nichols et al., 1995]:

- ▶ client-server architecture



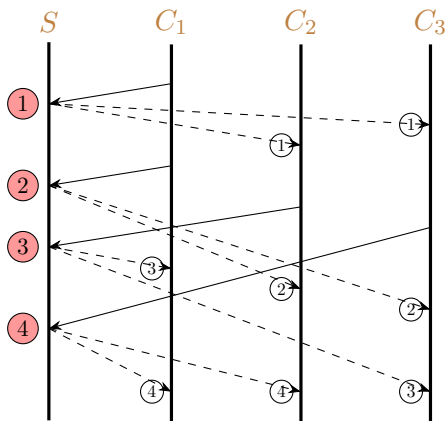
System model of Jupiter [Nichols et al., 1995]:

- ▶ client-server architecture
- ▶ client $\xrightarrow{\text{FIFO}}$ server



System model of Jupiter [Nichols et al., 1995]:

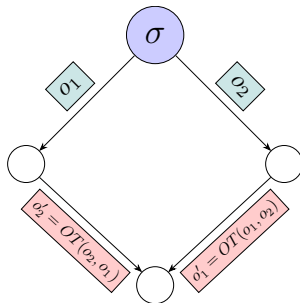
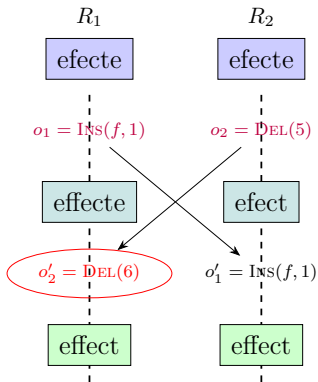
- ▶ client-server architecture
- ▶ client $\xrightarrow{\text{FIFO}}$ server
- ▶ totally ordered at the server



System model of Jupiter [Nichols et al., 1995]:

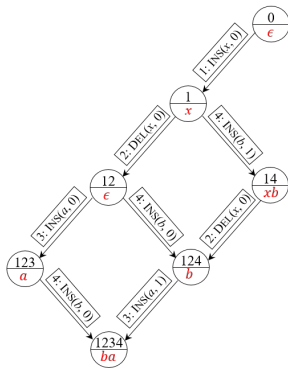
- ▶ client-server architecture
- ▶ client $\xrightarrow{\text{FIFO}}$ server
- ▶ totally ordered at the server
- ▶ server $\xrightarrow{\text{FIFO}}$ client

OT (Operational Transformation) [Ellis and Gibbs, 1989]

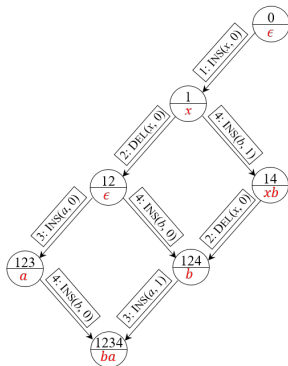


$$\sigma; o_1; o'_2 \equiv \sigma; o_2; o'_1$$

Jupiter uses **2D state spaces** [Xu, Sun, and Li, 2014]
to manage how and when to perform OTs.

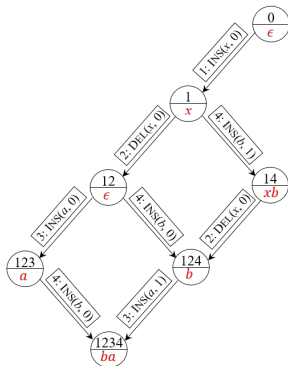


Jupiter uses **2D state spaces** [Xu, Sun, and Li, 2014]
to manage how and when to perform OTs.



Nodes represent states. Edges are labeled with operations.

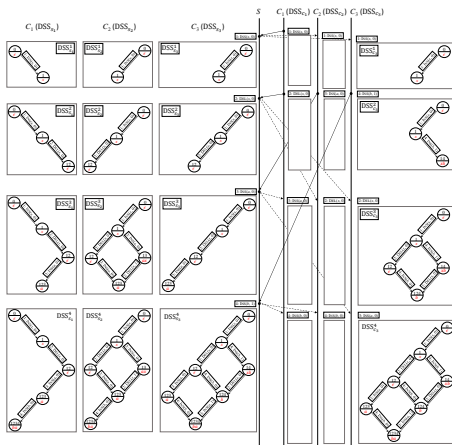
Jupiter uses **2D state spaces** [Xu, Sun, and Li, 2014]
to manage how and when to perform OTs.



Nodes represent states. Edges are labeled with operations.

2D: An operation from the same node is either LOCAL or GLOBAL.

Each **client** maintains a $2D$ state space.



The **server** maintains n ($= 3$) $2D$ state spaces, one for each client.

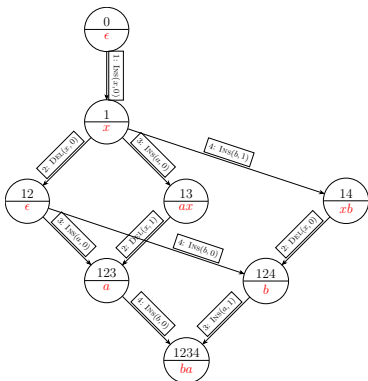
Global property on all replica states
specified by the weak list specification



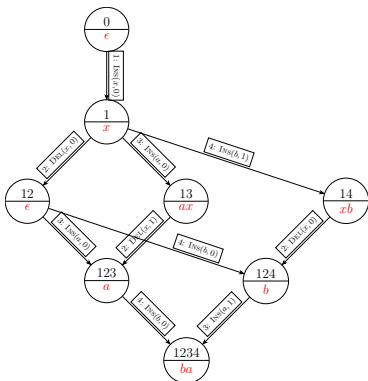
Local view each replica maintains in Jupiter

CJupiter (Compact Jupiter)

CJupiter maintains an n -ary ordered state space for each replica.

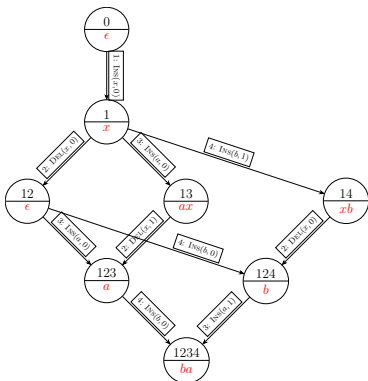


CJupiter maintains an n -ary ordered state space for each replica.



There can be more than two edges coming from the same node.

CJupiter maintains an n -ary ordered state space for each replica.

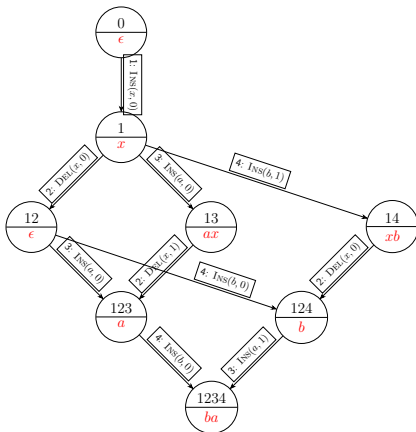


There can be more than two edges coming from the same node.

Edges from the same node are **totally ordered** by associated operations.

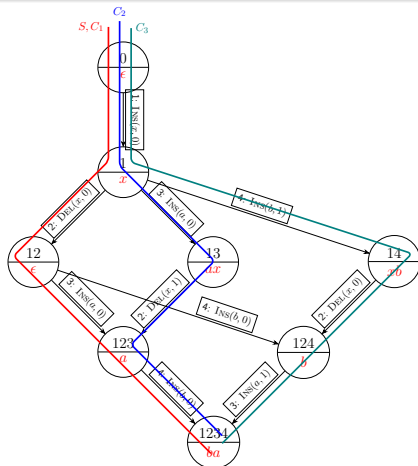
Proposition ($n + 1 \rightarrow 1$ (Informal))

At a high level, CJupiter maintains only **one** n -ary ordered state space.



Proposition ($n + 1 \rightarrow 1$ (Informal))

At a high level, CJupiter maintains only **one** n -ary ordered state space.



Each replica behavior corresponds to a **path** going through this state space.

Theorem (Equivalence)

Under the same schedule, the behaviors of corresponding replicas in CJupiter and Jupiter are the same.

Theorem (Equivalence)

Under the same schedule, the behaviors of corresponding replicas in CJupiter and Jupiter are the same.

At the server side:

Proposition ($n \leftrightarrow 1$ (Informal))

*The single n -ary ordered state space at the server side in CJupiter is a **compact representation** of n 2D state spaces at the server side in Jupiter.*

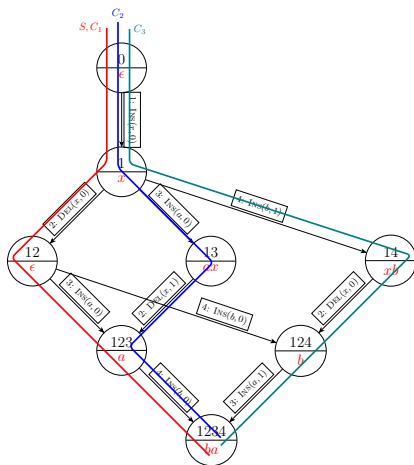
At the client side:

Proposition ($1 \leftrightarrow 1$ (Informal))

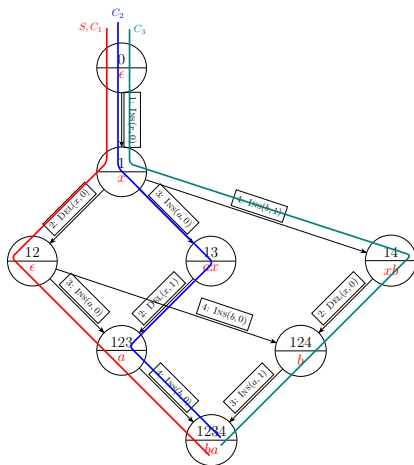
*Jupiter is **slightly optimized in implementation** at clients by eliminating redundant OTs than CJupiter.*

CJupiter Satisfies the Weak List Specification

We study a single n -ary ordered state space which provides a global view of all possible replica states.



We study a single n -ary ordered state space which provides a global view of all possible replica states.

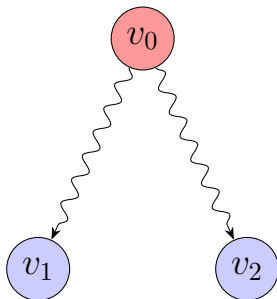


To show the pairwise state compatibility property in three steps.

- 1 Take any two nodes/states v_1 and v_2 .

Lemma (LCA (Lowest Common Ancestor))

*Each pair of states in the n -ary ordered state space has a **unique** LCA.*

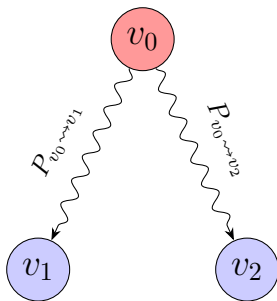


$$v_0 = \text{LCA}(v_1, v_2)$$

- 2 Consider the paths to v_1 and v_2 from their LCA v_0 .

Lemma (Disjoint Paths)

The set of operations $O_{v_0 \rightsquigarrow v_1}$ along $P_{v_0 \rightsquigarrow v_1}$ is *disjoint* from the set of operations $O_{v_0 \rightsquigarrow v_2}$ along $P_{v_0 \rightsquigarrow v_2}$.

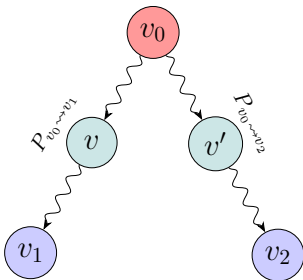


$$v_0 = \text{LCA}(v_1, v_2)$$

- 3 Consider the states in these two paths.

Lemma (Compatible Paths)

Each pair of states consisting of one state v in $P_{v_0 \rightsquigarrow v_1}$ and the other v' in $P_{v_0 \rightsquigarrow v_2}$ are **compatible**.



$$v_0 = \text{LCA}(v_1, v_2)$$

In particular,
 v_1 and v_2 are compatible.

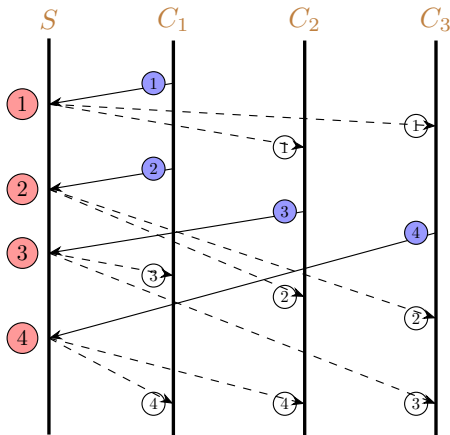
Thank
You!



- Attiya, Hagit et al. (2016). "Specification and complexity of collaborative text editing". In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. ACM, pp. 259–268.
- Ellis, C. A. and S. J. Gibbs (1989). "Concurrency Control in Groupware Systems". In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. SIGMOD '89. ACM, pp. 399–407.
- Nichols, David A. et al. (1995). "High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System". In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. ACM, pp. 111–120.
- Roh, Hyun-Gul et al. (2011). "Replicated Abstract Data Types: Building Blocks for Collaborative Applications". In: *J. Parallel Distrib. Comput.* 71.3, pp. 354–368.
- Shapiro, Marc et al. (2011). "Conflict-free Replicated Data Types". In: *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*. SSS'11. Springer-Verlag, pp. 386–400.
- Xu, Yi, Chengzheng Sun, and Mo Li (2014). "Achieving Convergence in Operational Transformation: Conditions, Mechanisms and Systems". In: *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work*. CSCW '14. ACM, pp. 505–518.

Backup

It is still challenging to achieve convergence despite the server.



Serializability may not be desirable.

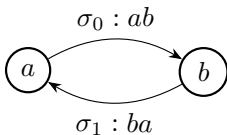
It does not imply that clients process operations in the same order.

$$\forall \sigma, \sigma' : a, b \in \sigma \cap \sigma' \implies (a \prec_{\sigma} b \iff a \prec_{\sigma'} b)$$

$(\sigma, \sigma' : \text{list}; \quad a, b : \text{element}; \quad \prec_{\sigma} : \text{precedes})$

$\sigma_0 : ab$

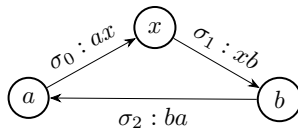
$\sigma_1 : ba$



$\sigma_0 : ax$

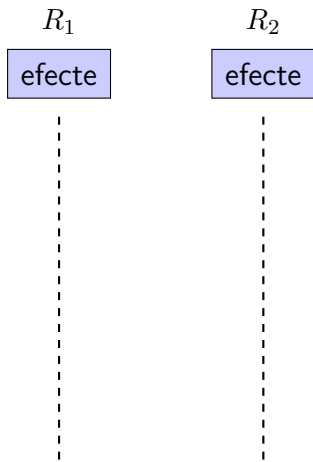
$\sigma_1 : xb$

$\sigma_2 : ba$

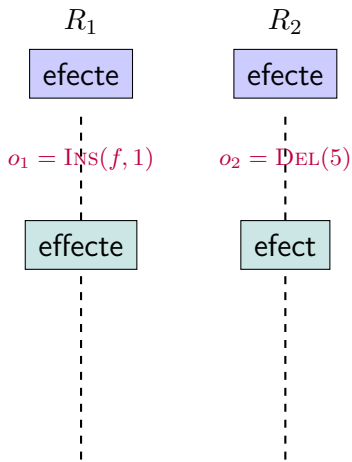


OT (Operational Transformation) [Ellis and Gibbs, 1989]

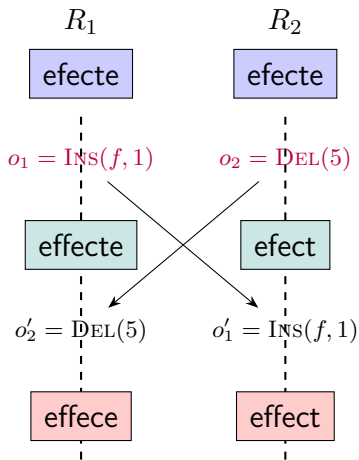
OT (Operational Transformation) [Ellis and Gibbs, 1989]



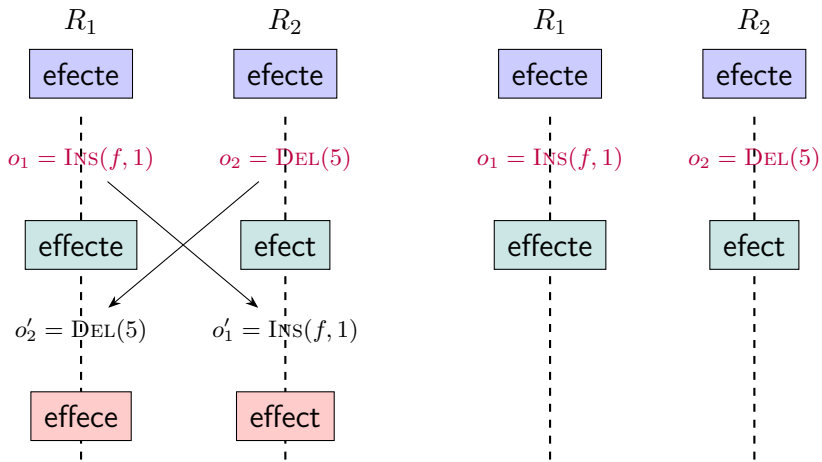
OT (Operational Transformation) [Ellis and Gibbs, 1989]



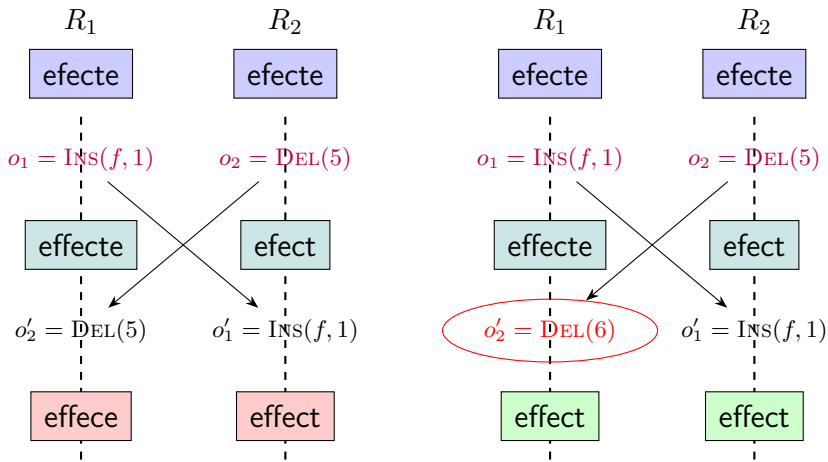
OT (Operational Transformation) [Ellis and Gibbs, 1989]



OT (Operational Transformation) [Ellis and Gibbs, 1989]



OT (Operational Transformation) [Ellis and Gibbs, 1989]



OT functions for a replicated list object [Ellis and Gibbs, 1989]

$$OT\left(\text{INS}(a_1, p_1, pr_1), \text{INS}(a_2, p_2, pr_2)\right) = \begin{cases} \text{INS}(a_1, p_1, pr_1) & p_1 < p_2 \\ \text{INS}(a_1, p_1 + 1, pr_1) & p_1 > p_2 \\ \text{NOP} & p_1 = p_2 \wedge a_1 = a_2 \\ \text{INS}(a_1, p_1 + 1, pr_1) & p_1 = p_2 \wedge a_1 \neq a_2 \wedge pr_1 > pr_2 \\ \text{INS}(a_1, p_1, pr_1) & p_1 = p_2 \wedge a_1 \neq a_2 \wedge pr_1 \leq pr_2 \end{cases}$$

$$OT\left(\text{INS}(a_1, p_1, pr_1), \text{DEL}(-, p_2, pr_2)\right) = \begin{cases} \text{INS}(a_1, p_1, pr_1) & p_1 \leq p_2 \\ \text{INS}(a_1, p_1 - 1, pr_1) & p_1 > p_2 \end{cases}$$

$$OT\left(\text{DEL}(-, p_1, pr_1), \text{INS}(a_2, p_2, pr_2)\right) = \begin{cases} \text{DEL}(-, p_1, pr_1) & p_1 < p_2 \\ \text{DEL}(-, p_1 + 1, pr_1) & p_1 \geq p_2 \end{cases}$$

$$OT\left(\text{DEL}(-, p_1, pr_1), \text{DEL}(-, p_2, pr_2)\right) = \begin{cases} \text{DEL}(-, p_1, pr_1) & p_1 < p_2 \\ \text{DEL}(-, p_1 - 1, pr_1) & p_1 > p_2 \\ \text{NOP} & p_1 = p_2 \end{cases}$$

Consider a replicated system with n ($= 3$) clients.

