

Research Paper/Article Discussion

“What’s Different with the New Google Docs: Making Collaboration Fast”

SDE Skills, Book Club

Reference book: Designing Data Intensive Applications
by Martin Kleppmann

Wednesday, September 2nd, 2020

Presented By: Dippy Aggarwal

This topic is listed as one of the references at the end of Chapter 5 (Replication) in the book.

Part 1:

<https://drive.googleblog.com/2010/05/whats-different-about-new-google-docs.html#:~:text=The%20new%20editor%20comes%20with,make%20this%20new%20functionality%20possible.>

Part 2: https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs_22.html

Part 3: <https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html>

Motivation behind research paper discussions and this article in particular

- Recognize the connections between fundamental ideas presented in the book with a real-world application/service/system.
- This article offers **good insights for a system design question** as well – Google docs/collaborative editing scenario.
- Covers several concepts from Chapter 5 (replication) including replication architecture, conflict resolution strategies.

Can you spot a connection(s) between the topic and the ideas in the replication chapter?

- Collaboration in Google Docs consists of sending changes from one editor to the server, and then to the other editors.
 - >> *Each editor acts like a leader. (Multi-leader replication architecture)*
- Collaboration is not quite as simple as sending changes to the other editors because people get out of sync.
 - >> *Conflict resolution techniques.*
- Real time collaborative editing adds further challenges around **latency**.
 - >> *All editors are capable of rapidly processing each other's changes in real time.*

Goals with collaborative editing

"EVENTUAL CONSISTENCY" ?

-eventual delivery

(eventually, every op seen by every node)

asynchronous: may deliver in any order

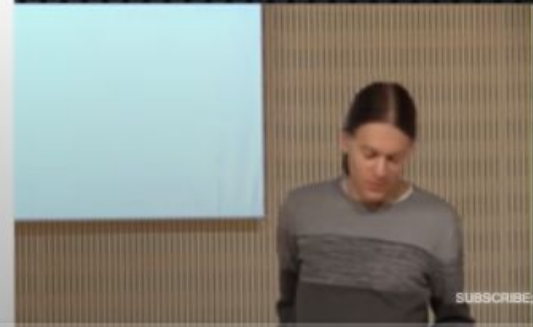
-convergence

(seen same ops \Rightarrow have same state)

-don't lose data

(otherwise it's easy 😊)

goto;
conference



Part I of collaborative editing

Introduction to **operational transformation**

Operational Transformation (OT) is an **algorithm/technique** for the **transformation of operations** such that **they can be applied to documents whose states have diverged**, bringing them both back to the **same state**.

+ non-blocking concurrent editing.

<https://medium.com/@srijancse/operational-transformation-the-real-time-collaborative-editing-algorithm-bf8756683f66>

<https://medium.com/@srijancse/how-real-time-collaborative-editing-work-operational-transformation-ac4902d75682>

Simple example to understand CE in Google Docs (the idea of operational transformation) - 1/2

GOOGLE DOCS (NUTSHELL)

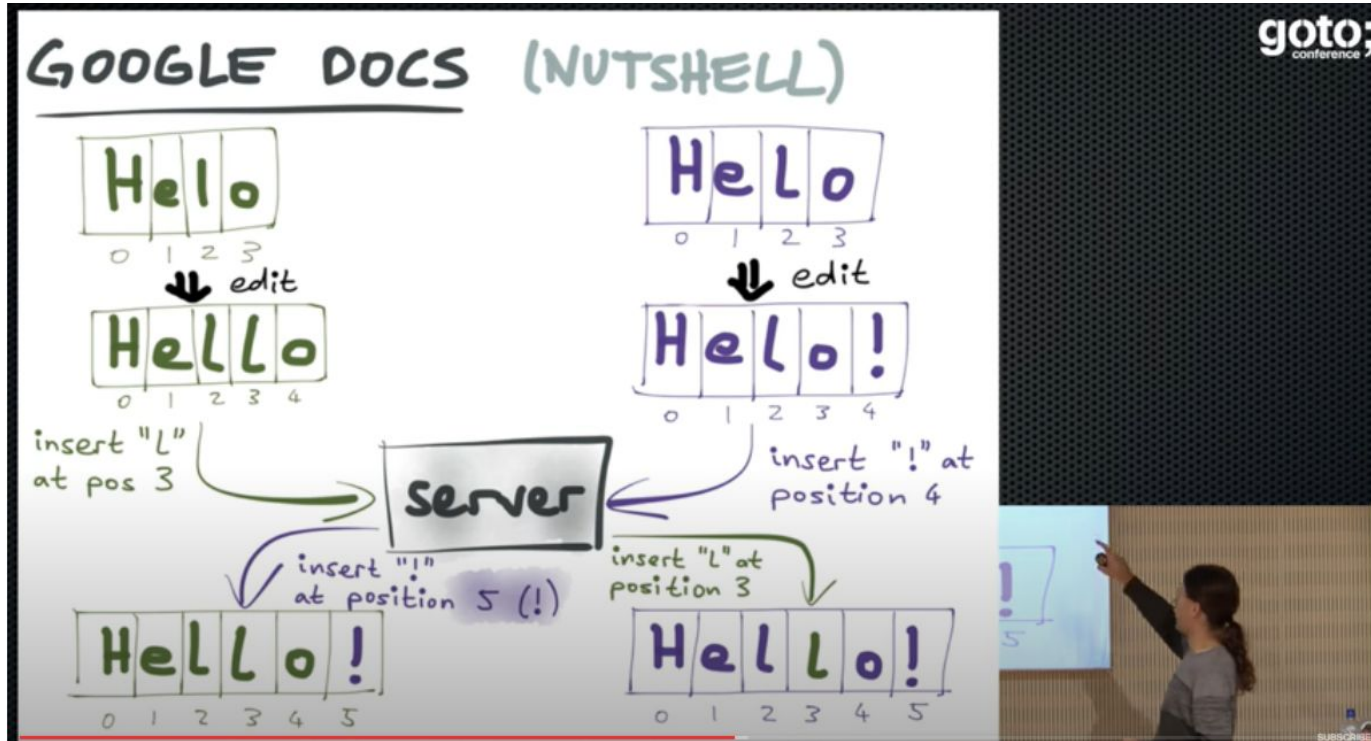
The diagram illustrates the concept of operational transformation (OT) in Google Docs. It shows two parallel editing scenarios:

- Left Scenario:** The initial text is "Hello" (indices 0-3). An "edit" operation is applied, resulting in the text "Hello" (indices 0-4).
- Right Scenario:** The initial text is "Hello" (indices 0-3). An "edit" operation is applied, resulting in the text "Hello!" (indices 0-4).

The "goto; conference" logo is visible in the top right corner of the diagram area.

All edits boil down to three basic types of changes: **inserting** text, **deleting** text, and **applying styles** to a range of text.

Simple example to understand CE in Google Docs (the idea of operational transformation) -2/2



Each editor transforms incoming changes so that they make sense relative to the local version of the document.

Operational transformation:

for handling **concurrent operations**, we use the **transform** function that **takes two operations** that have been **applied to the same document state** (but on **different clients**) and **computes a new operation** that can be applied after the second operation and **that preserves the first operation's intended change**.

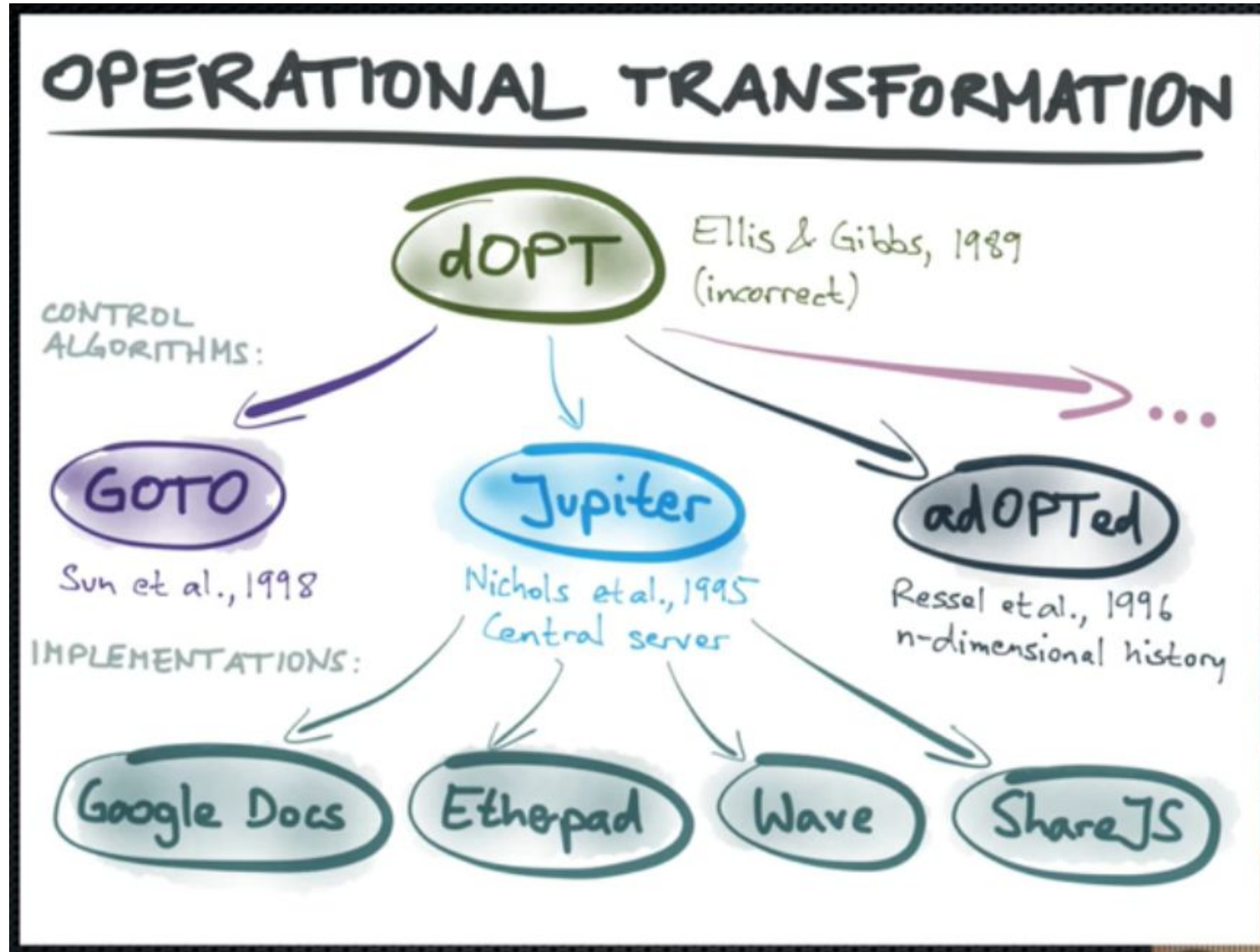
In the above scenario - left client's change was applied first at the server and then server transformed the second client's change (operational transformation in play!).

You may wonder what will happen if second client's change (insert "I") was applied first.

Answer: That's very well possible if both the updates arrive at same time and second client's update is picked first.

What's important to note - whichever order changes are picked, all the clients will see a consistent view of final data.

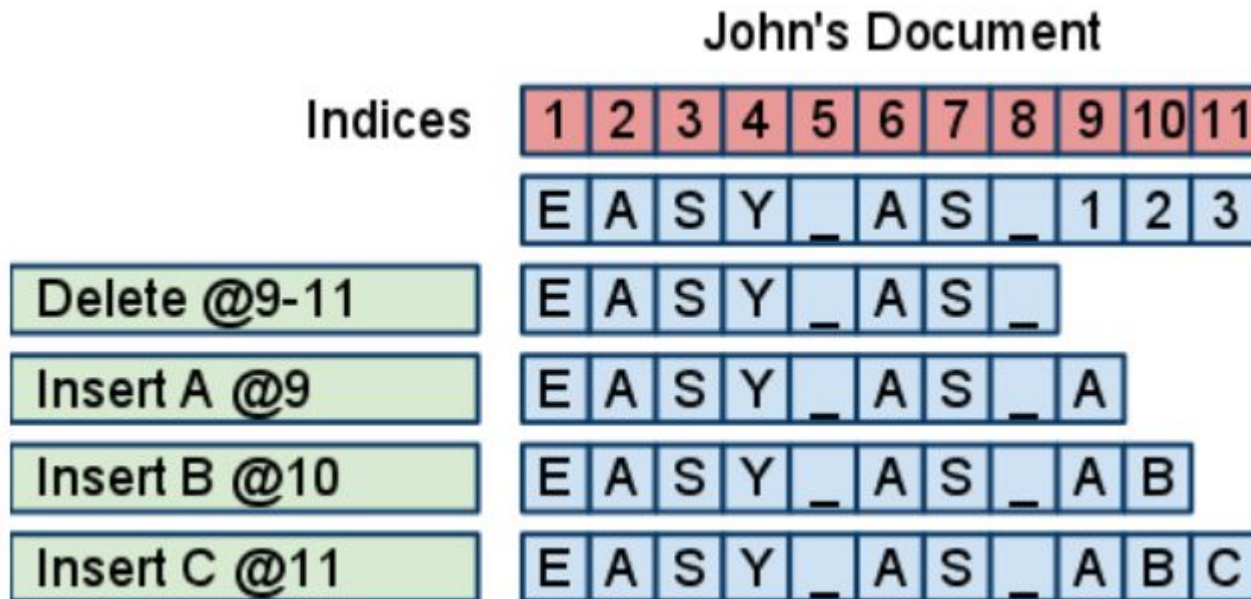
Operational transformation idea dates back to 1989!



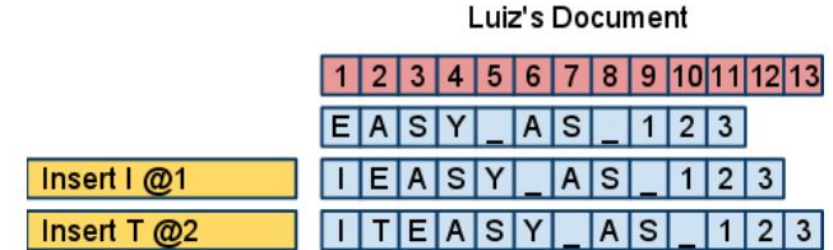
Example#2 (OT and collaborative editing)

Suppose that a document edited by John and Luiz initially reads; **EASY AS 123**

John (represented by green) changes the document to **EASY AS ABC**



Suppose as John is typing, Luiz (represented by yellow) begins to change his document to **IT'S EASY AS 123**. He first inserts the **I** and the **T** at the beginning of the document:



Suppose Luiz naively applies John's first change {DeleteText @9-11}:



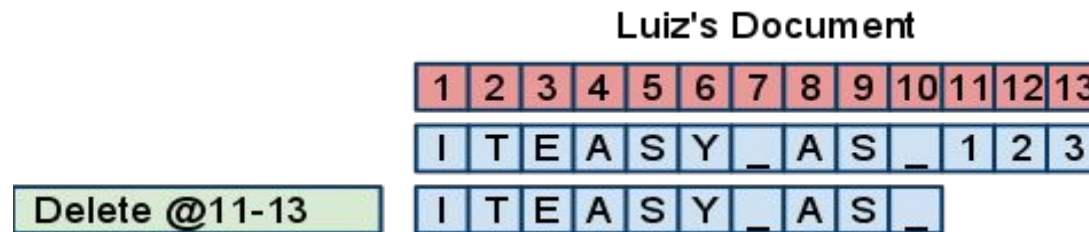
He deleted the wrong characters!

What went wrong?

Luiz had two characters at the beginning of the doc that John was never aware of. So the location of John's change was wrong relative to Luiz's version of the document.

To avoid this problem, **Luiz must transform John's changes** and **make them relative to his local document**. In this case, when Luiz receives changes from John he needs to know to shift the changes over by two characters to adjust for the **IT** that Luiz added.

Once he does this transformation and applies John's first change, he gets:

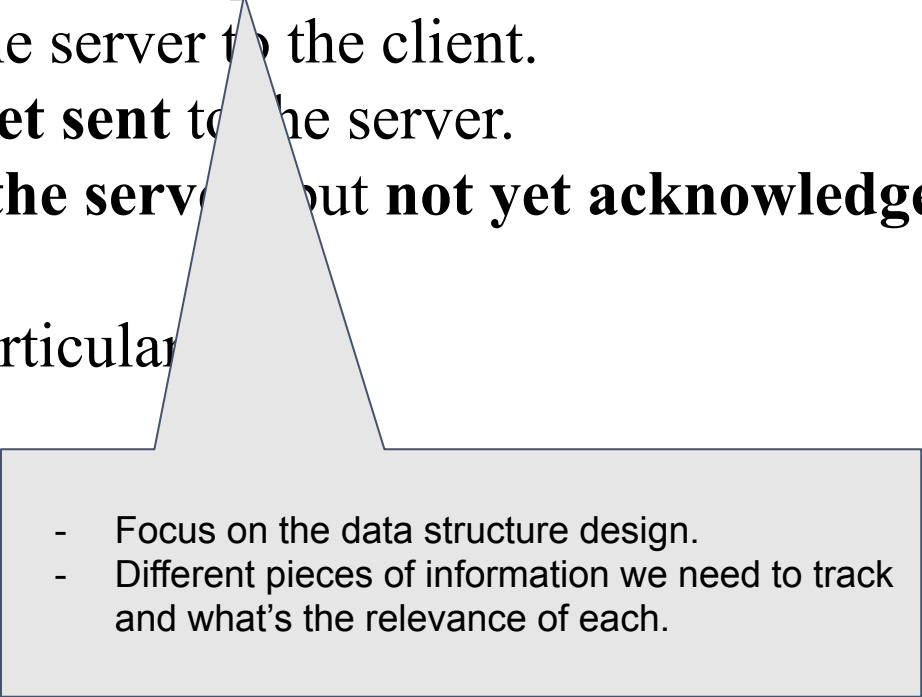


The algorithm that we use to handle these shifts is called **operational transformation (OT)**. If OT is implemented correctly, it guarantees that once all editors have received all changes, everyone will be looking at the same version of the document.

Part II: Collaborative editing in Google Docs

To collaborate in Google Docs, **each client keeps track of four pieces of information:**

- The number of the most recent revision sent from the server to the client.
- Any changes that have been made locally and **not yet sent** to the server.
- Any **changes** that have been **made locally, sent to the server** but **not yet acknowledged** by the server.
- The current state of the document as seen by that particular

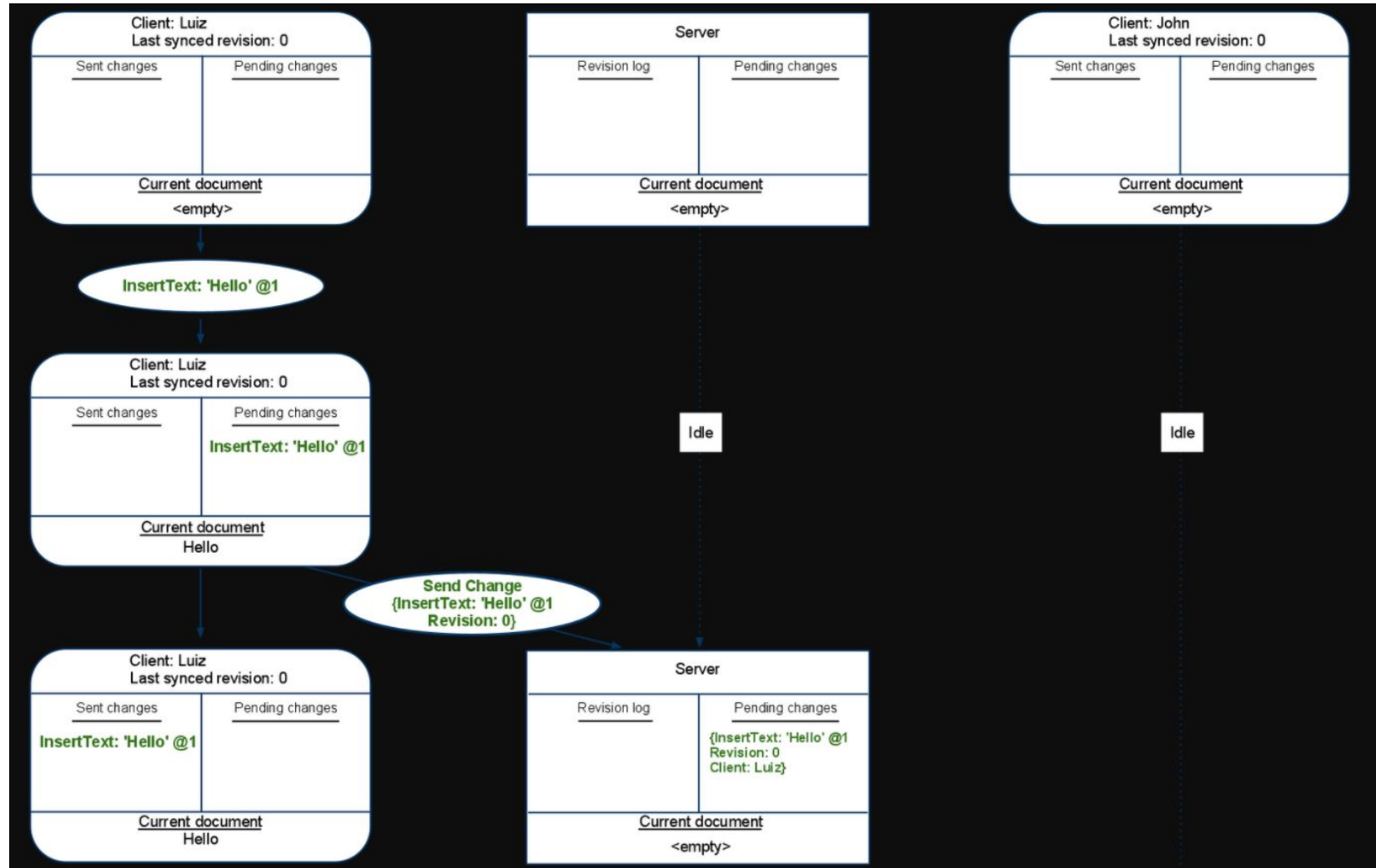
- 
- Focus on the data structure design.
 - Different pieces of information we need to track and what's the relevance of each.

Part II: Collaborative editing in Google Docs

The **server remembers three things**:

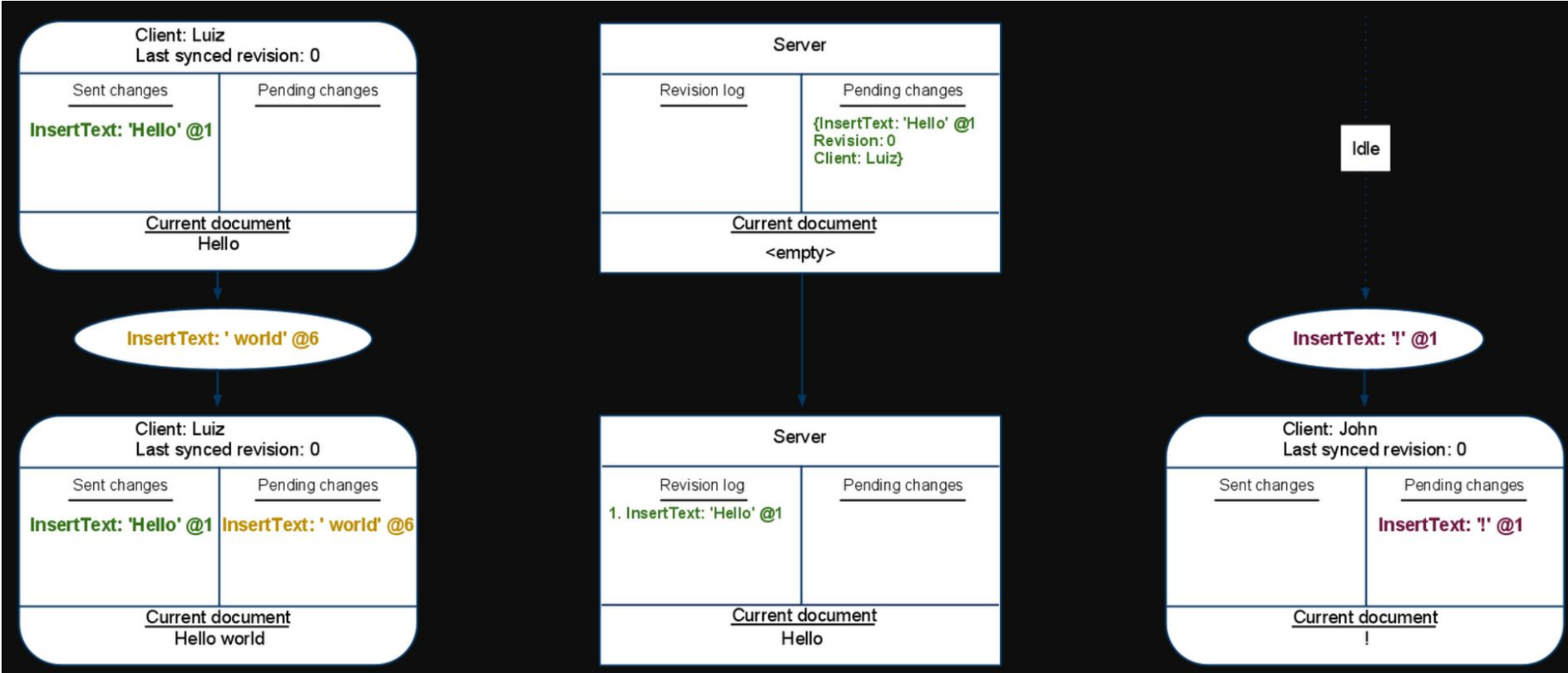
- The list of all changes that it has received but not yet processed.
- The complete history of all processed changes (called the revision log).
- The current state of the document as of the last processed change.

#1 Luiz starts by typing the word Hello at the beginning of the document.



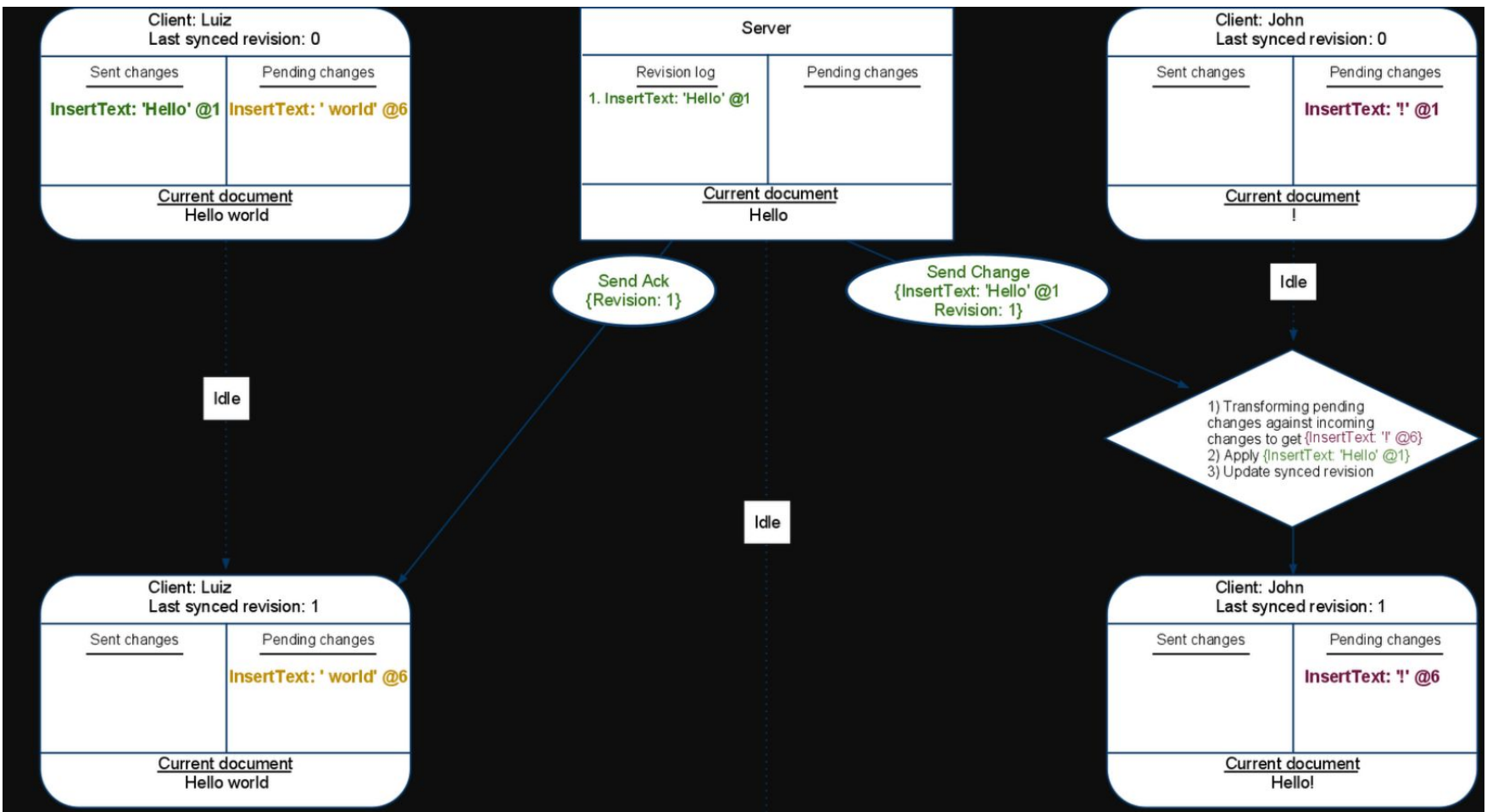
Luiz's client added the edit to his list of pending changes. He then sent the change to the server and moved the change into his list of sent changes.

Luiz continues to type, adding the word **world** to his document.
At the same time, John types an **!** in his empty version of the document (remember he has not yet received Luiz's first change).



Luiz's {InsertText ' world' @6} change was placed in the pending list and wasn't sent to the server because we never send more than one pending change at a time.

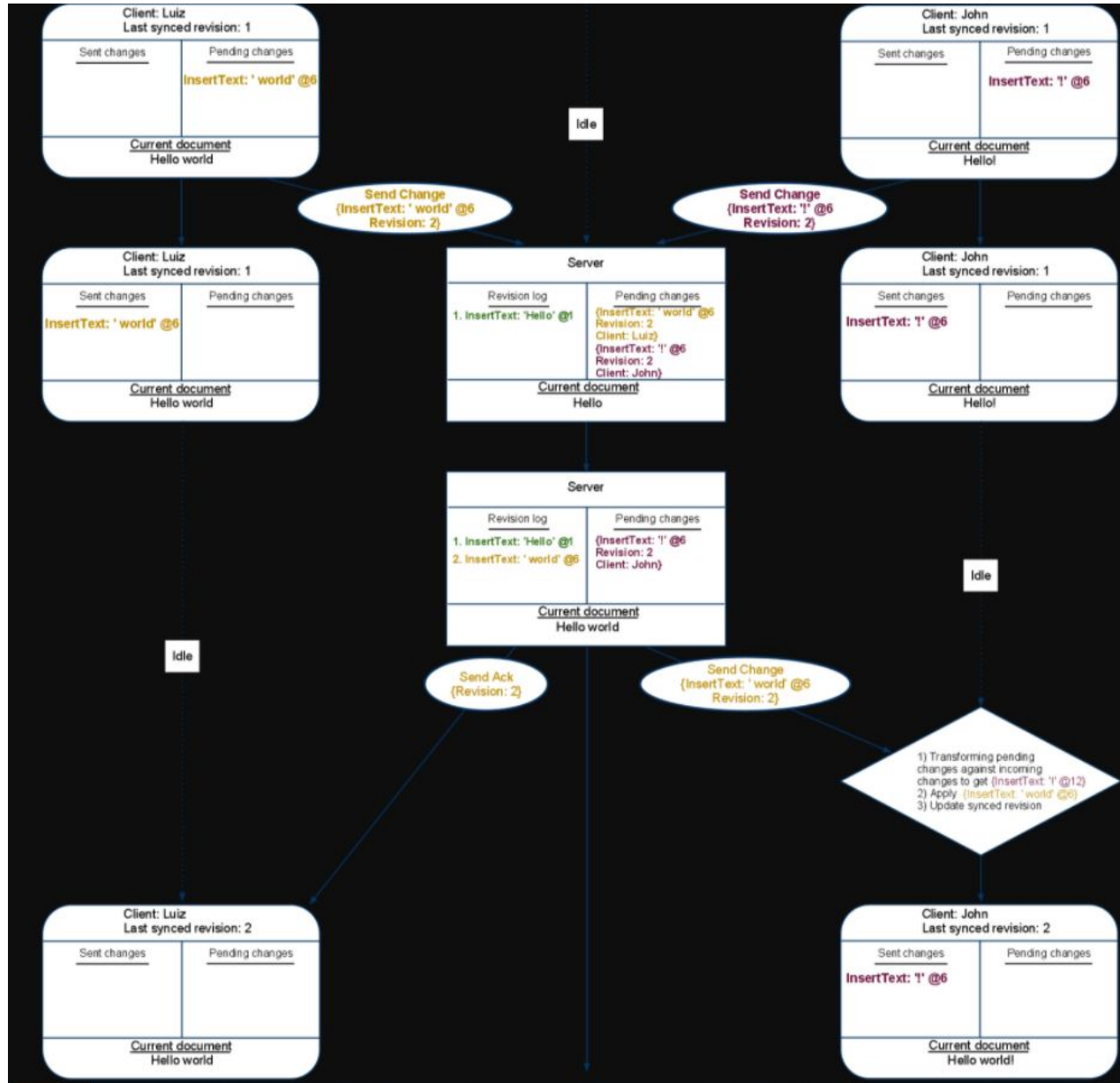
Until Luiz receives an acknowledgement of his first change, his client will keep all new changes in the pending list. Also notice that the server stored Luiz's first change in its revision log.



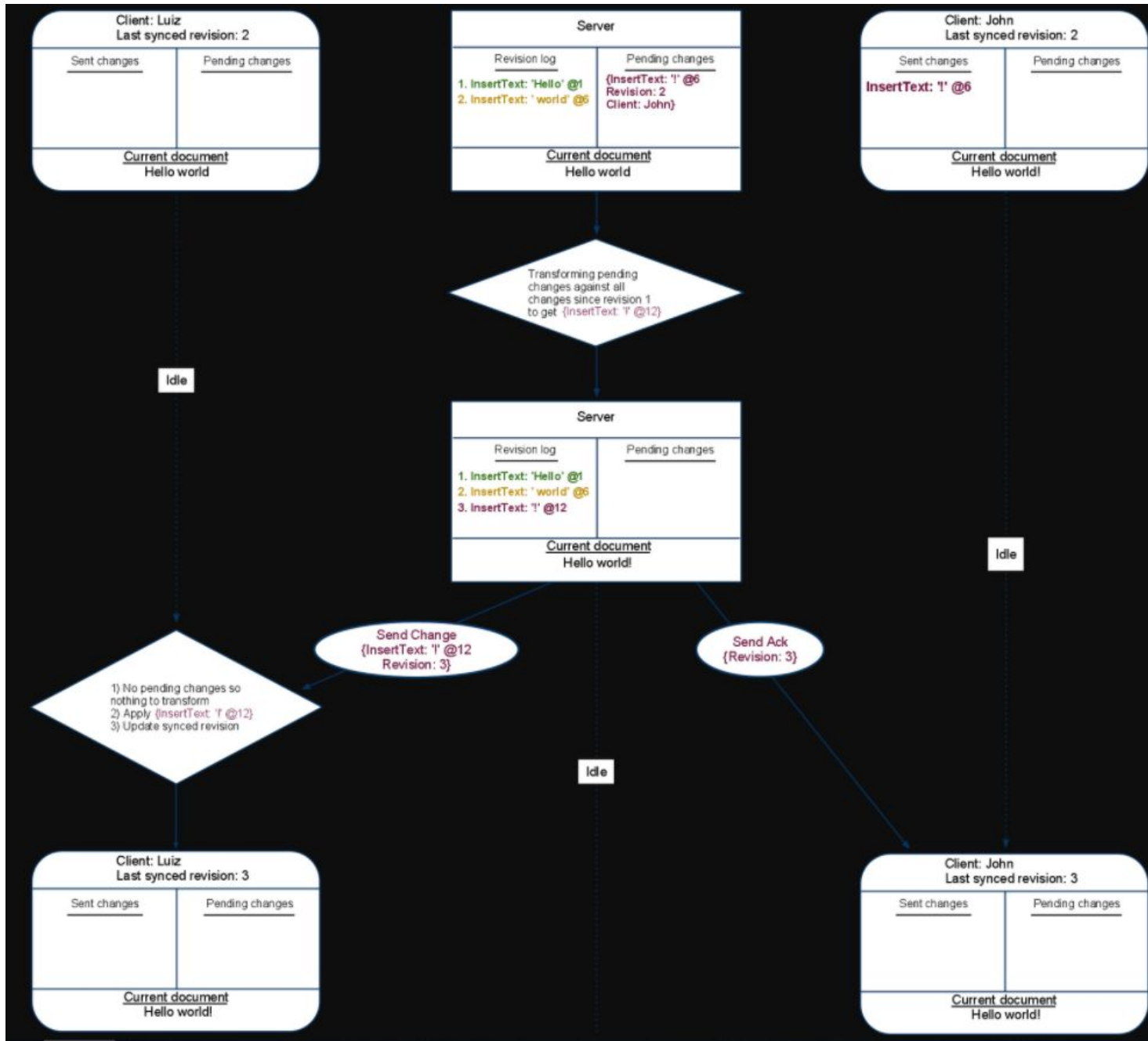
Next, the server will send John a message containing Luiz's first change and it will send Luiz a message acknowledging that it has processed that first change.

John received Luiz's edit from the server and used [operational transformation](#) (OT) to transform it against his pending {InsertText '!' @1} change. The result of the transformation was to shift the location of John's pending change by 5 to make room at the beginning of the document for Luiz's Hello. Notice that both Luiz and John updated their last synced revision numbers to 1 when they received the messages from the server. Lastly, when Luiz received the acknowledgement of his first change, he removed that first change from the list of sent changes.

Next, both Luiz and John are going to send their unsent changes to the server.



The server got Luiz's change before John's so it processed that change first. An acknowledgement of the change was sent to Luiz. The change itself was sent to John, where his client transformed it against his still pending `{InsertText '!' @1}` change.



The first thing the server did, was to transform John's sent change against all the changes that have been committed since the last time John synced with the server. In this case, it transformed John's change against Luiz's {InsertText ' world' @6}. The result shifted the index of John's change over by 6. This shift is identical to the transformation John's client made when it first received Luiz's {InsertText 'Hello' @1}.

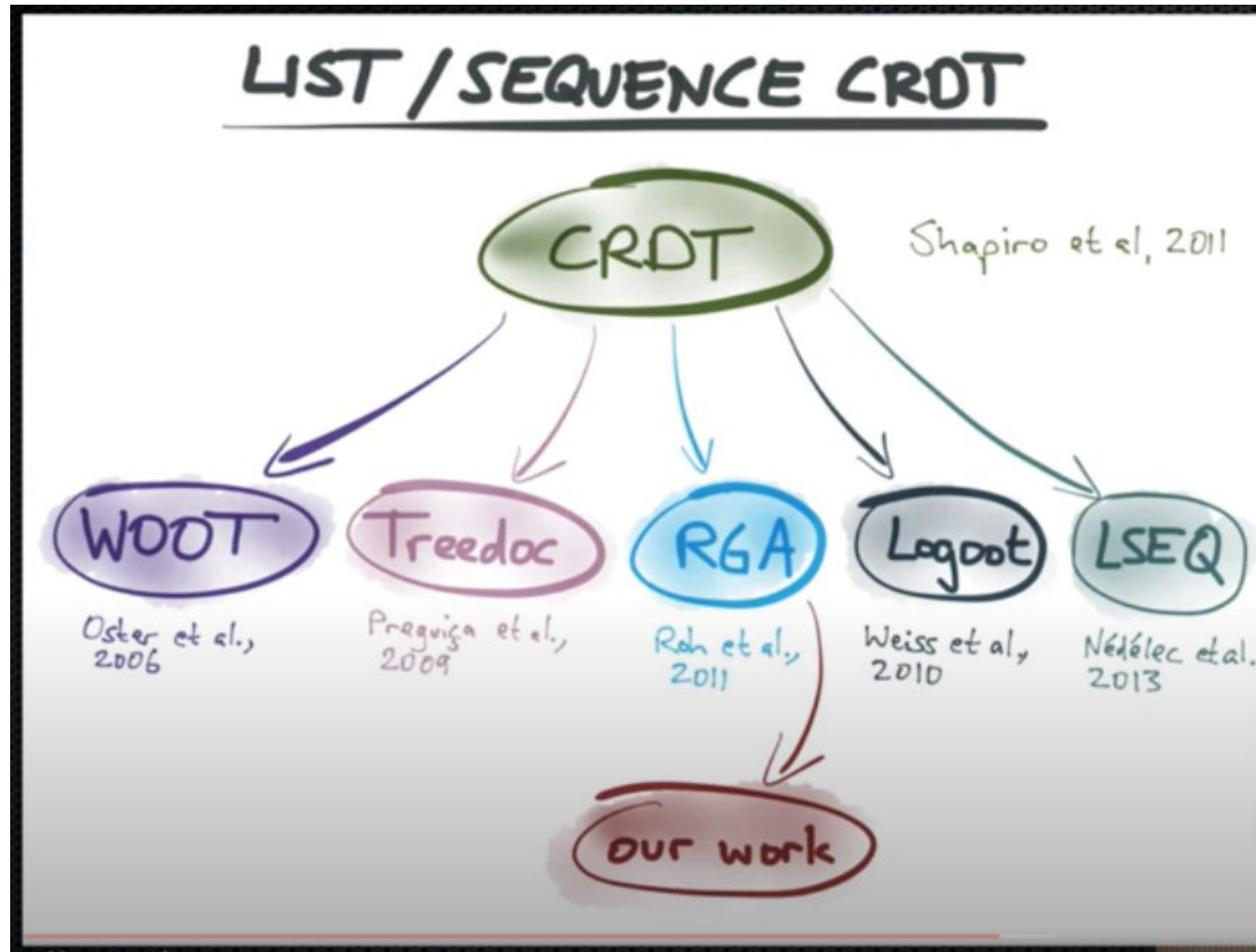
Characteristics/Advantages of OT and Collaboration model

- Collaboration is fast. At all times, **every editor can optimistically apply their own changes locally without waiting for the server to acknowledge those changes.** This means that the speed or reliability of your network connection doesn't influence how fast you can type.
- Collaboration is accurate. There is **always enough information for each client to merge collaborators' changes** in the same deterministic way.
- Collaboration is efficient. The **information that is sent over the network is always the bare minimum** needed to describe what changed.
- Collaboration complexity is constant. The server does not need to know anything about the state of each client. Therefore, the complexity of processing changes does not increase as you add more editors.
- Collaboration is distributed. Only the server needs to be aware of the document's history and only the clients need to be aware of uncommitted changes. This division spreads the workload required to support real time collaboration between all the parties involved.

Do you recognize any issues with Operational transformation?

- Central server (Single point of failure).
- Server needs to decrypt the message.

CRDT - address limitations of OT

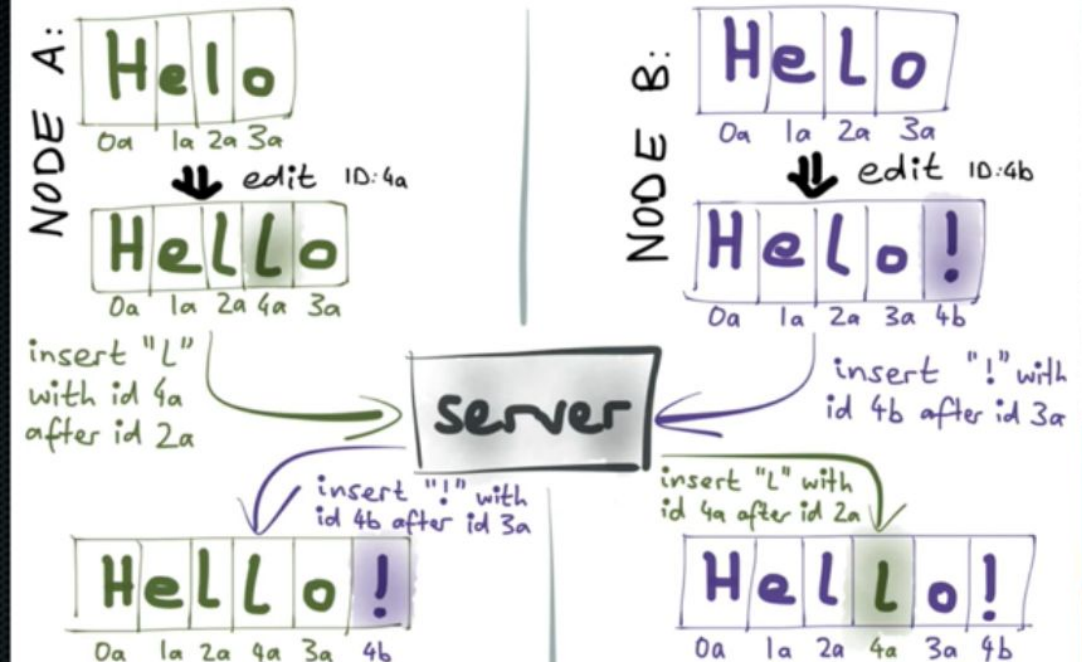


CRDT - 2/2

ORDERED LIST CRDT (NUTSHELL)



ORDERED LIST CRDT (NUTSHELL)



Lessons learned

Check your understanding

Can you answer/reason about the following areas.

- Distinguishing between different replication architectures (single-leader, multi-leader, leaderless). Why is Google docs collaborative editing not a leaderless architecture?
- Can you enumerate few approaches to conflict resolution?
- Can you explain operational transformation algorithm in plain english? What scenarios is it used in? How does it work? What are the inputs and output of this algorithm? What are its limitations?
- How does CRDT help to overcome limitations around Operational transformation?

Reference talk

The image is a screenshot of a presentation slide. The slide has a white background with a black border. In the top right corner, there is a logo that says "goto; conference". The main content of the slide is handwritten text. At the top left, the word "Conflict" is written in red, with a yellow starburst shape behind it. Below it, the word "eventual" is written in black. To the right of "eventual", the words "resolution for" are written in black. Below "resolution for", the word "consistency" is written in green, enclosed in a green cloud-like shape. A horizontal line separates the title from the author information. Below the line, the name "MARTIN KLEPPMANN" is written in black, followed by the email address "@martinkl" on the next line. To the right of the email address, the text "University of Cambridge Computer Laboratory" is written in black, and "TRVE DATA" is written in black below it. In the bottom right corner, there is a small video inset showing a woman standing in front of a screen. The screen displays the text "University of Cambridge Computer Laboratory" and "TRVE DATA".

goto;
conference

Conflict
eventual resolution for
consistency

MARTIN
KLEPPMANN
@martinkl

University of Cambridge
Computer Laboratory
TRVE DATA

University of Cambridge
Computer Laboratory
TRVE DATA

GOTO 2016 • Conflict Resolution for Eventual Consistency • Martin Kleppmann

<https://www.youtube.com/watch?v=yCcWpzY8dIA>