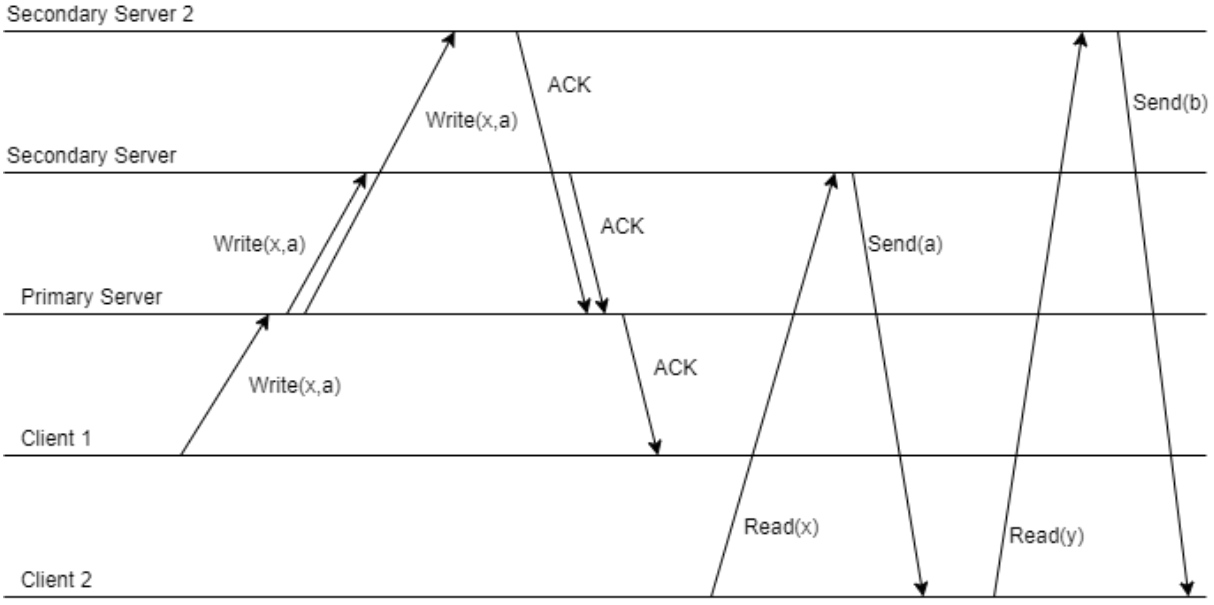
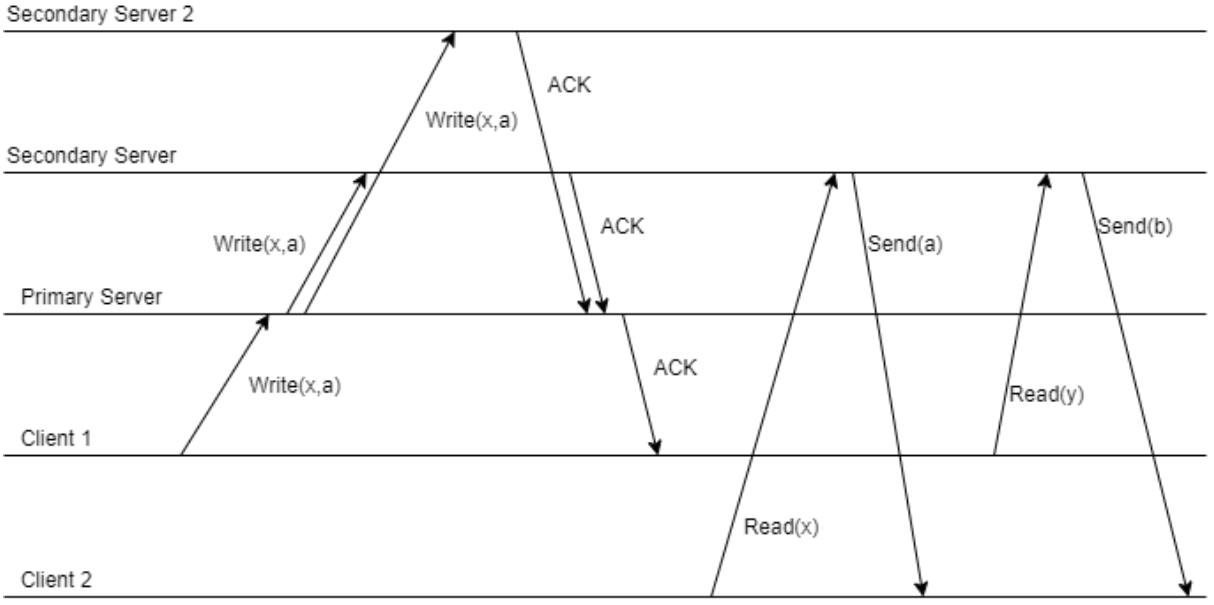
What is there to be done

1. **Working in group, represent four examples that illustrate the proposals of our colleague: 1a, 1b, 2a and 2b.**

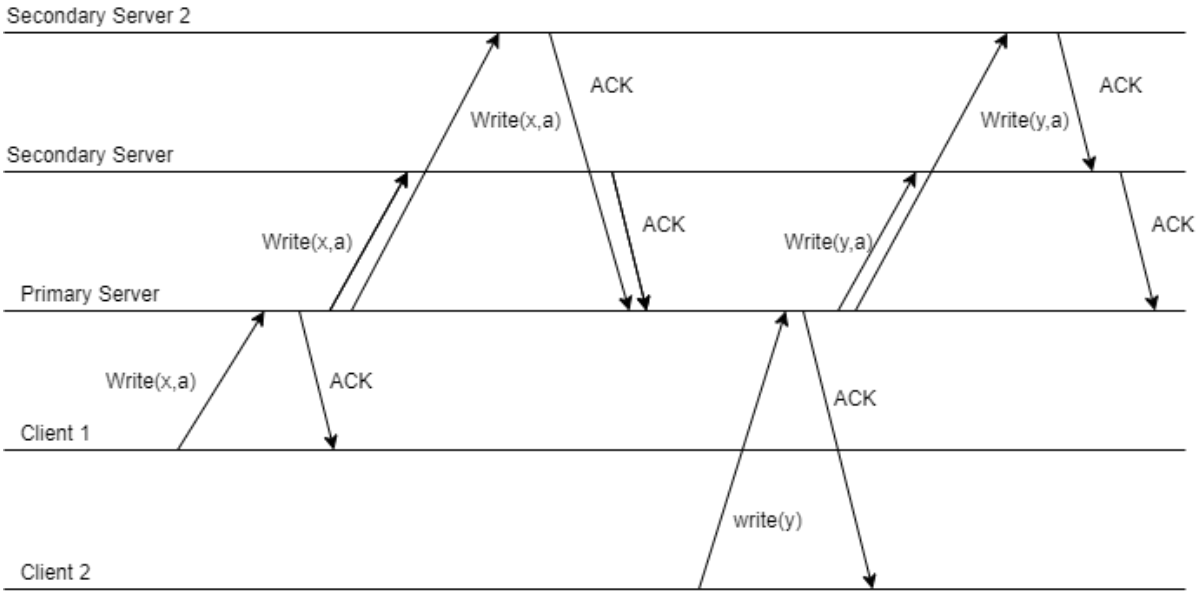
   1a - Read operations to an arbitrary secondary server. Write operations to the primary one, and disseminate.



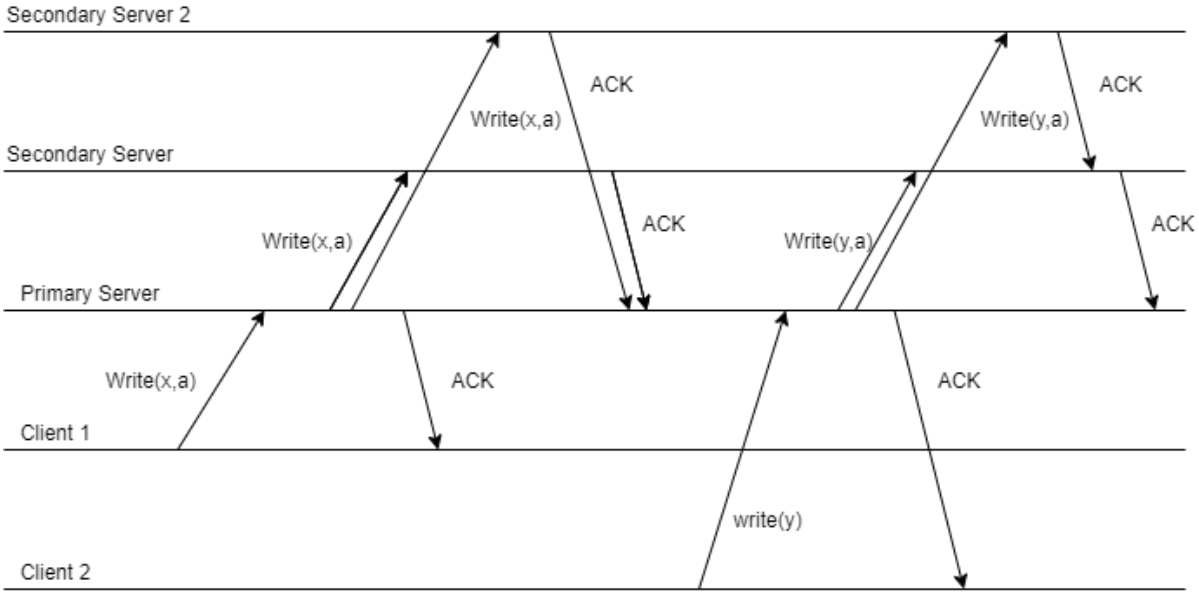
   1b - All read operations to a specific secondary server. Write operations to the primary one, and disseminate.



2a - Treat the client request on the primary, and respond to the client before disseminating to the secondary.



2b - Treat the client request on the primary, and respond to the client after disseminating to the secondary.



2. **We will focus first on Proposition 1. Discuss in the group how the new passive replication technique behaves in both alternatives (a) and (b). Do these modifications affect the properties of order and atomicity? Is the consistency criterion of the original technique lowered? What happens if failures occur?**

   It will affect atomicity if we go by straight definition of all replicas having to handle the request, but considering that a consultation doesn't change the system it wouldn't matter if we didn't run the request in all servers; it can however lower the consistency due to the order, if we don't implement a semaphore system or an equivalent in the primary it becomes possible that the consultation will be processed by another server as the primary is making changes to the data, and our second request getting outdated information. If failures occur, depending on how we implement the primary,it would either have to be processed by the primary or it will need to send the request to yet another server. For proposition A in which any server handles the request we could by timeout send the request to another server and then return the result to the client, but in proposition B in which the secondary is always used we would be forced to either use the primary or return an error which would break the failure transparency.

3. **Now, consider Proposition 2 and analyze it in the same way as the previous one.**

   In either scenario the property of atomicity and order remains as the primary server will attend the requests and dictate the order to the backups, if the backups fail they will be disconnected until they can be updated and put back online, and if its the primary the one that fails it will timeout on the client side and a new primary will be chosen, and the changes to the primary crashed server will be ignored complying with the atomicity requirements(all or nothing).

   Where there is significant change between a and b proposals is after failures. In the case of proposal a responding to the request before disseminating the results can in the case of crash on the primary mean that the client will assume a successful request and as previously stated the primary server changes will be ignored thus leading to an inconsistent state. In proposal b this can be somewhat mitigated as the primary will wait to send the requests to the other servers before sending the ack message to the client. In this scenario if the primary fails a backup can take over with the changes done by the client, provided of course that we devised a method to make sure that all of them have made the changes and not just some(these last ones can be made unavailable until they update their contents).

4. **Finally, what happens if we try to combine proposals 1 and 2 in the same technique? Do this analysis in the same way as the previous ones.**

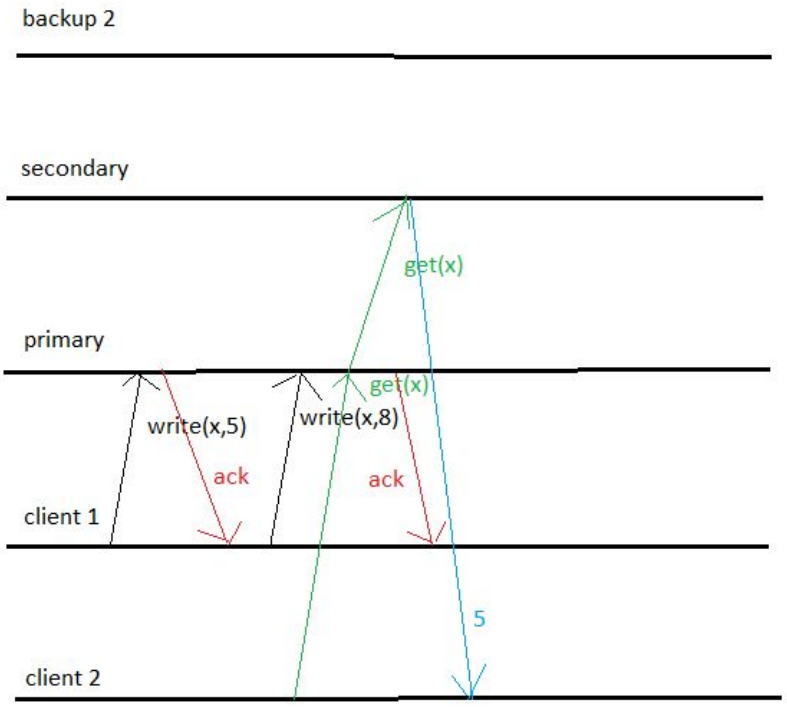
   If we combine these 2 proposals with their sub proposals we end up with a system in which the primary doesn't handle the consultations nor it waits for acknowledgement that the backups have updated their content according to the changes on the primary.

   We would end up with issues of consistency arising from a lack of clear order. Since we handle requests from other servers without waiting for the task at the primary, we could be pulling outdated data from the backup/secondary server(breach of order). Since only consults are independent we won't have problems with atomicity as outlined in the previous analysis (either the primary crashes and the changes are rolled back with a backup becoming primary or the primary fails after updating some backups and our consensus algorithm maintains the changes) this provided of course that we implement 2b and not 2a.

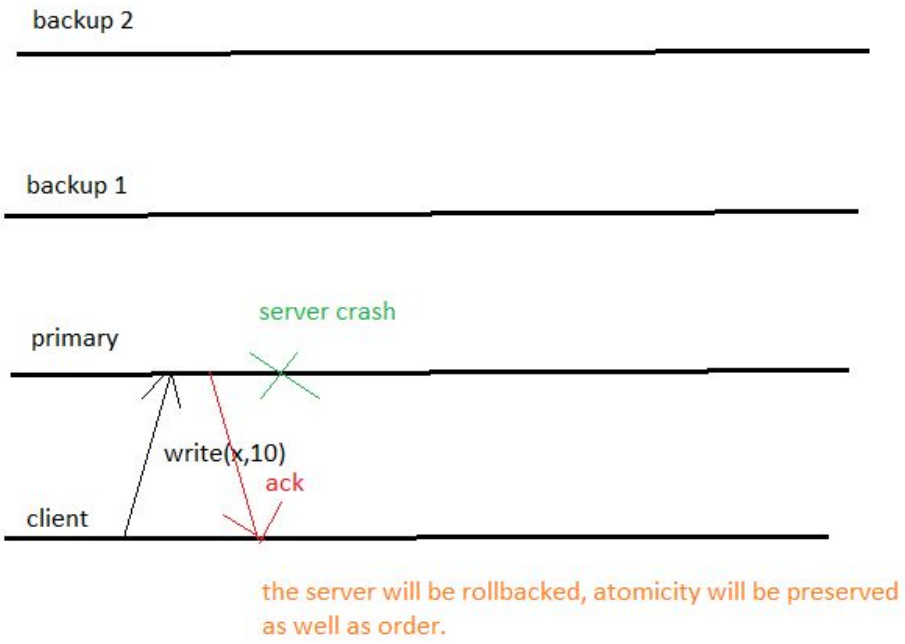
5. **Use chronograms to represent cases that illustrate if the properties of order and/or atomicity are violated.**

Proposal 1

Violation of order leading to inconsistency



Proposal 2



the server will be rollbacked, atomicity will be preserved as well as order.