Amazon Dynamo

DS 5110: Big Data Systems

Spring 2025

Lecture 19b

Yue Cheng

Some material taken/derived from:

- Princeton COS-418 materials created by Michael Freedman.
- Wisconsin CS 544 by Tyler Caraza-Harter.

Announcement

- This coming Thursday (04/24)
  - Final exam review

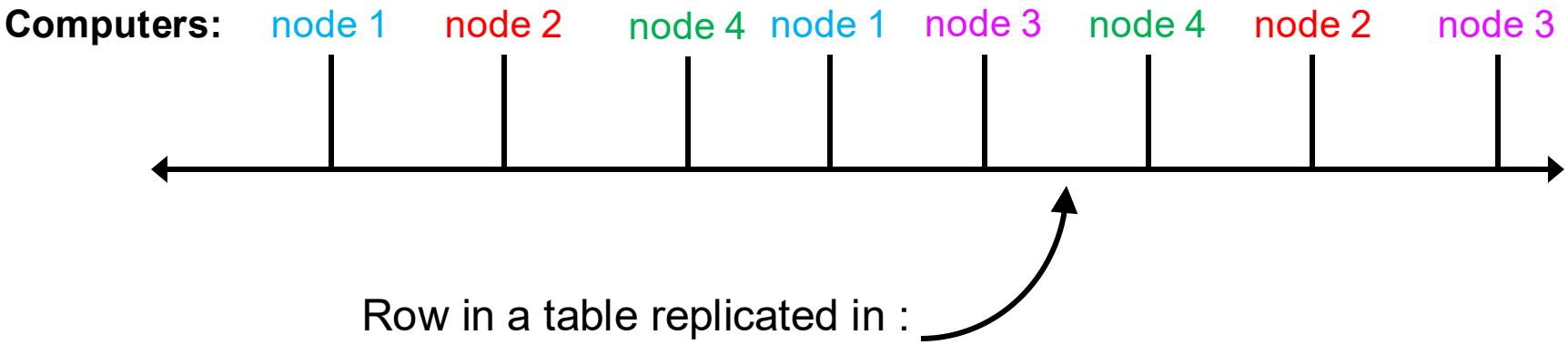
- Next Tuesday (04/29): Our final lecture of the semester
  - Invited speakers from Hugging Face XetHub
  - Xet recently acquired by Hugging Face (largest acquisition of Hugging Face!)
  - The Xet team is working on redesigning Hugging Face data storage infrastructure

Learning objectives

- Learn how Dynamo replicates data
  - Walk a token ring to identify multiple nodes responsible for a given key (row)
- Tune read and write quorum requirements to achieve desired tradeoffs in availability, durability, and performance
- Describe common approaches to eventual consistency and conflict resolution

Replication

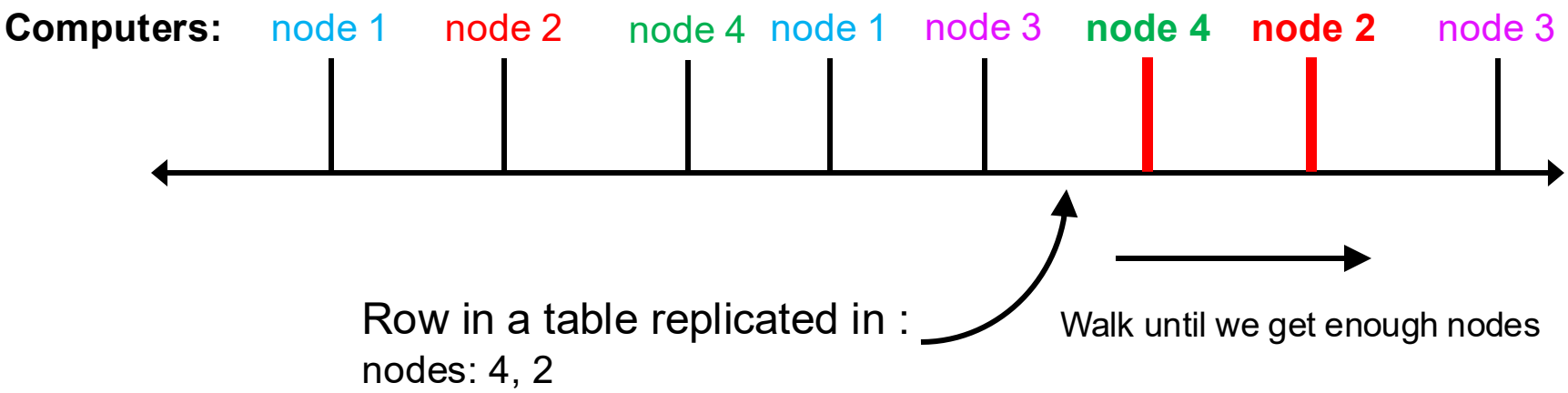
Token map:
token(node1) = {t1, t2}
token(node2) = {t3, t4}
token(node3) = {t5, t6}
token(node4) = {t7, t8}



Replication factor (RF) of N (where N == **2**)

Replication

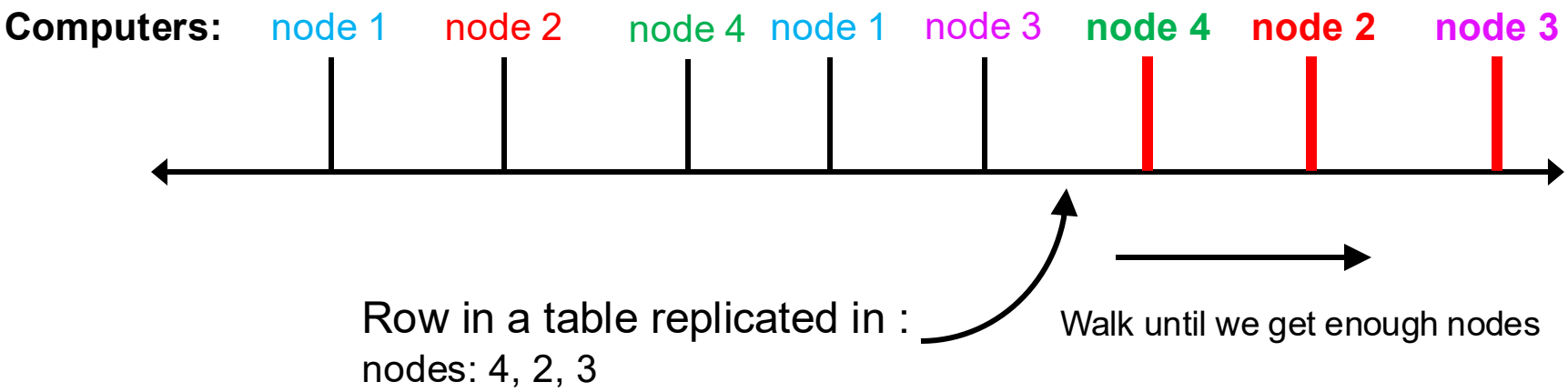
Token map:
token(node1) = {t1, t2}
token(node2) = {t3, t4}
token(node3) = {t5, t6}
token(node4) = {t7, t8}



RF = N (where N == **2**)

Replication

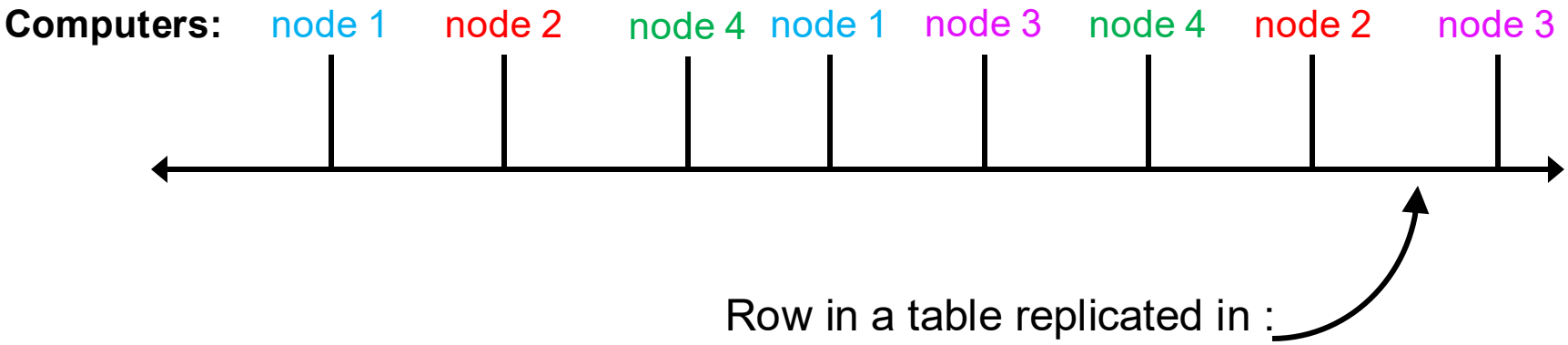
Token map:
token(node1) = {t1, t2}
token(node2) = {t3, t4}
token(node3) = {t5, t6}
token(node4) = {t7, t8}



RF = N (where N == **3**)

Replication

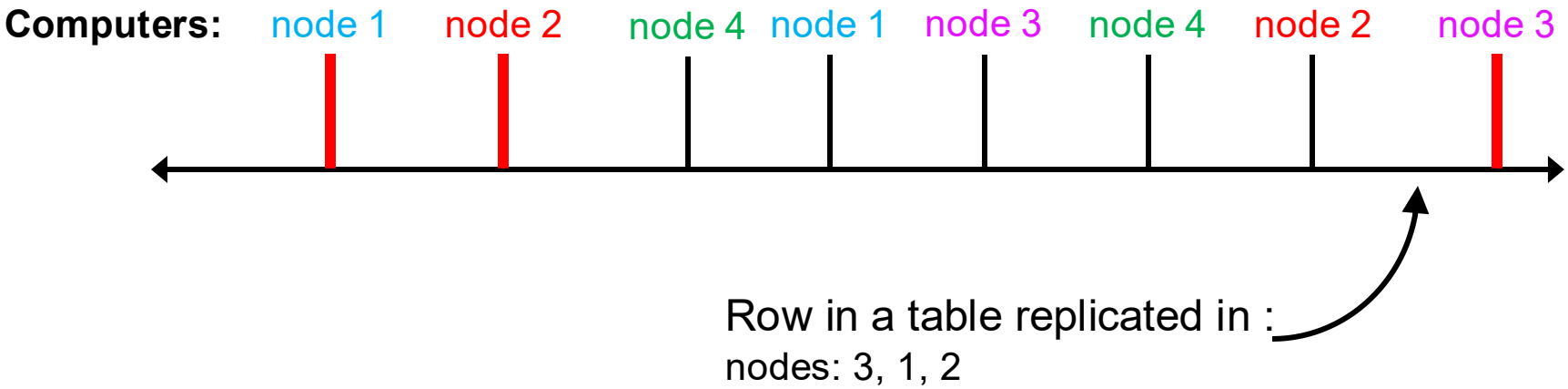
Token map:
token(node1) = {t1, t2}
token(node2) = {t3, t4}
token(node3) = {t5, t6}
token(node4) = {t7, t8}



RF = N (where N == **3**)

Replication

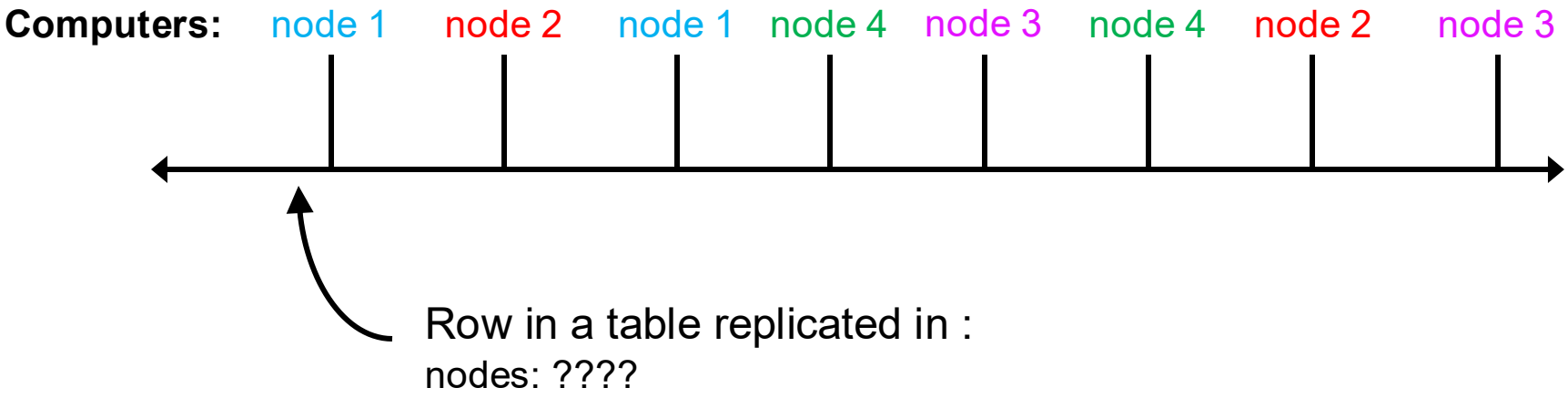
Token map:
token(node1) = {t1, t2}
token(node2) = {t3, t4}
token(node3) = {t5, t6}
token(node4) = {t7, t8}



Replication factor of N (where N == **3**)

Replication

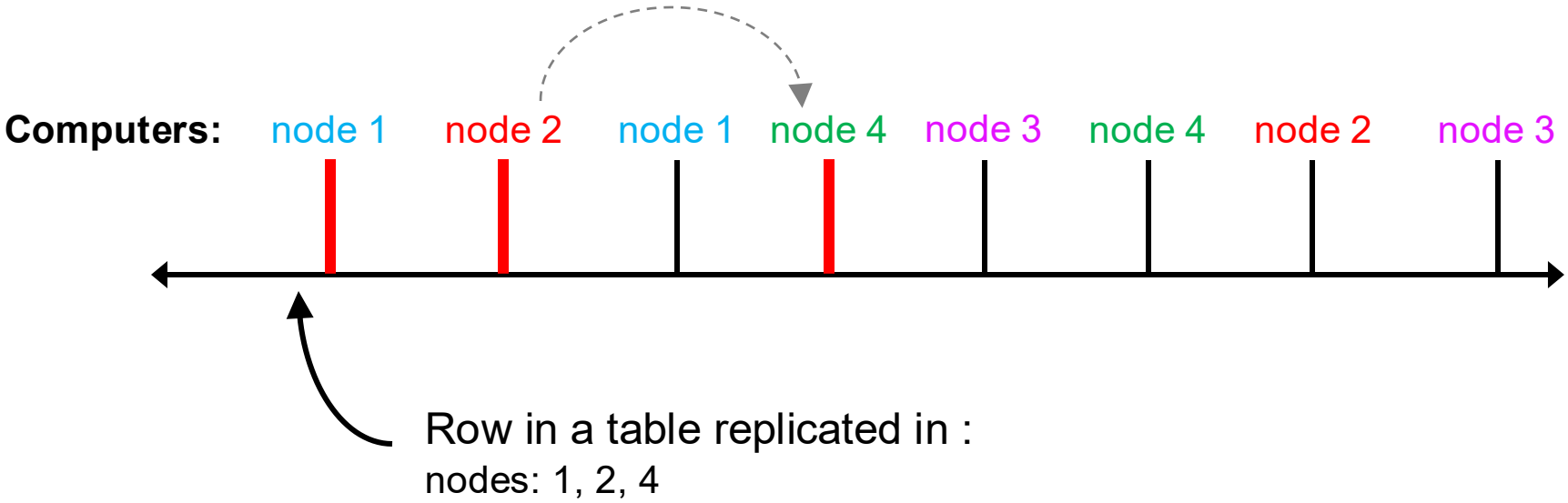
Token map:
token(node1) = {t1, t2}
token(node2) = {t3, t4}
token(node3) = {t5, t6}
token(node4) = {t7, t8}



RF = N (where N == **3**)

Replication

Token map:
token(node1) = {t1, t2}
token(node2) = {t3, t4}
token(node3) = {t5, t6}
token(node4) = {t7, t8}



RF = N (where N == **3**)

Important: Keeping multiple copies on vnodes on the same node provides little safety (when a node dies, all its vnodes die). Same **"failure domain"**.

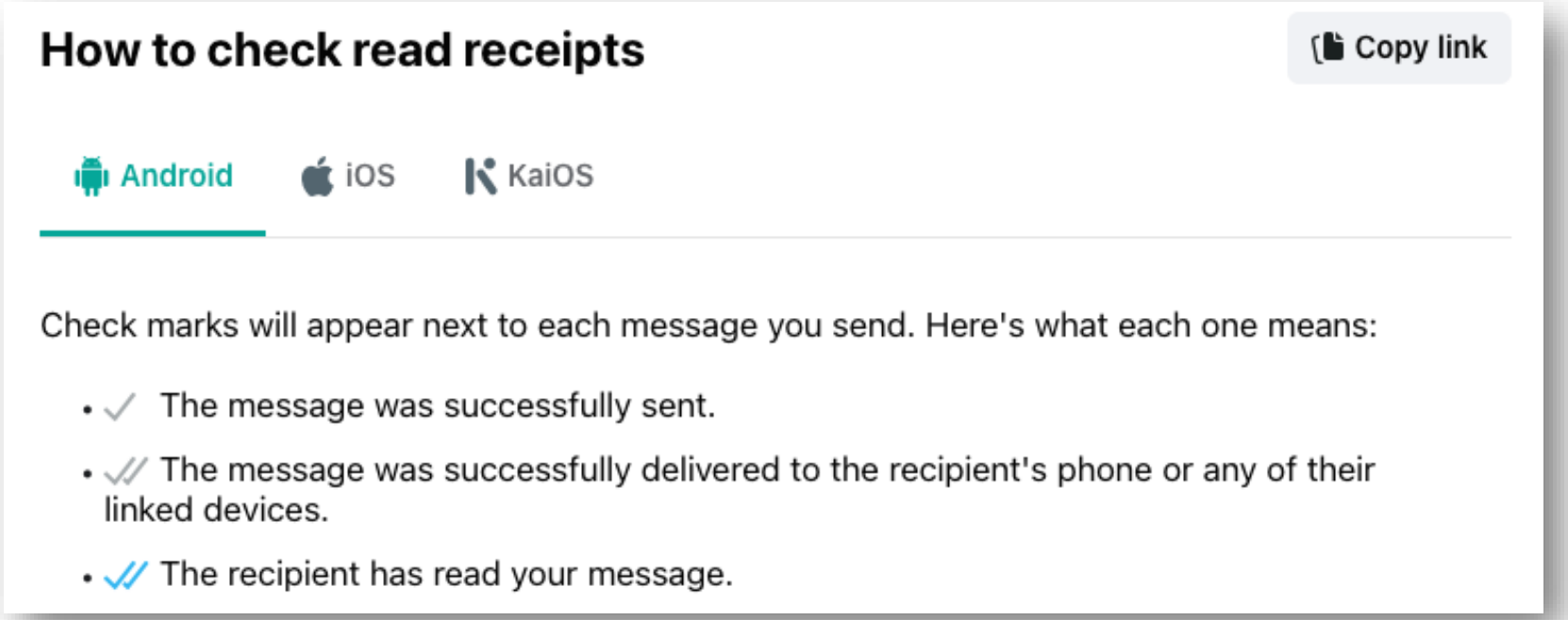
Dynamo skips nodes to ensure replicas reside on different nodes.

Write acks

- In distributed storage/database systems, an ***ack*** means our data is ***committed***

- "Committed" means our data is "**safe**", even if bad things happen. The definition varies system to system, based on what bad things are considered. For example:
  - A node could hang until rebooted; a node's disk could permanently fail
  - A rack could lose power; a datacenter could be destroyed

Write acks: WhatsApp example

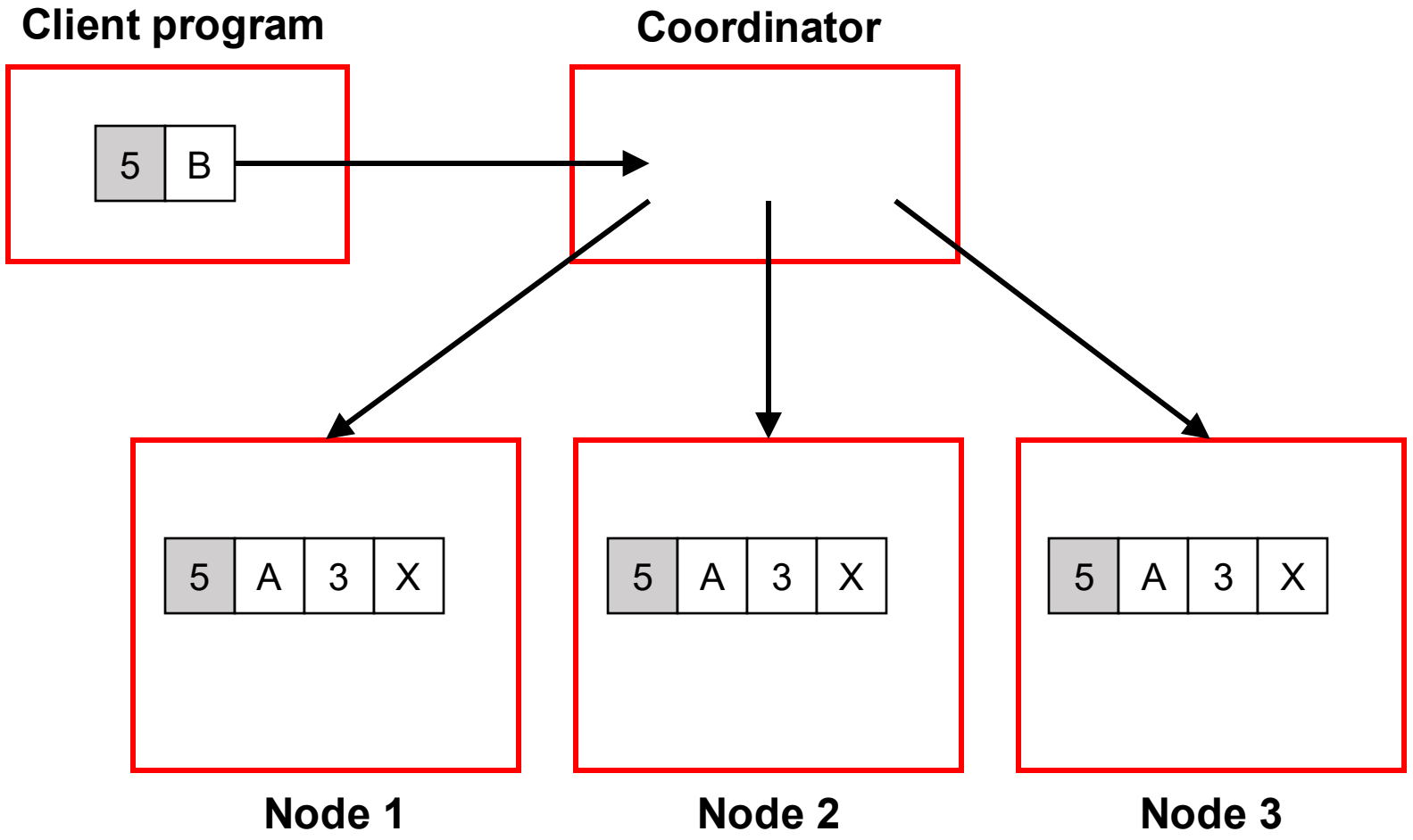


These are examples of "**acks**" (acknowledgments)

https://faq.whatsapp.com/665923838265756/?cms_platform=android&helpref=platform_switcher

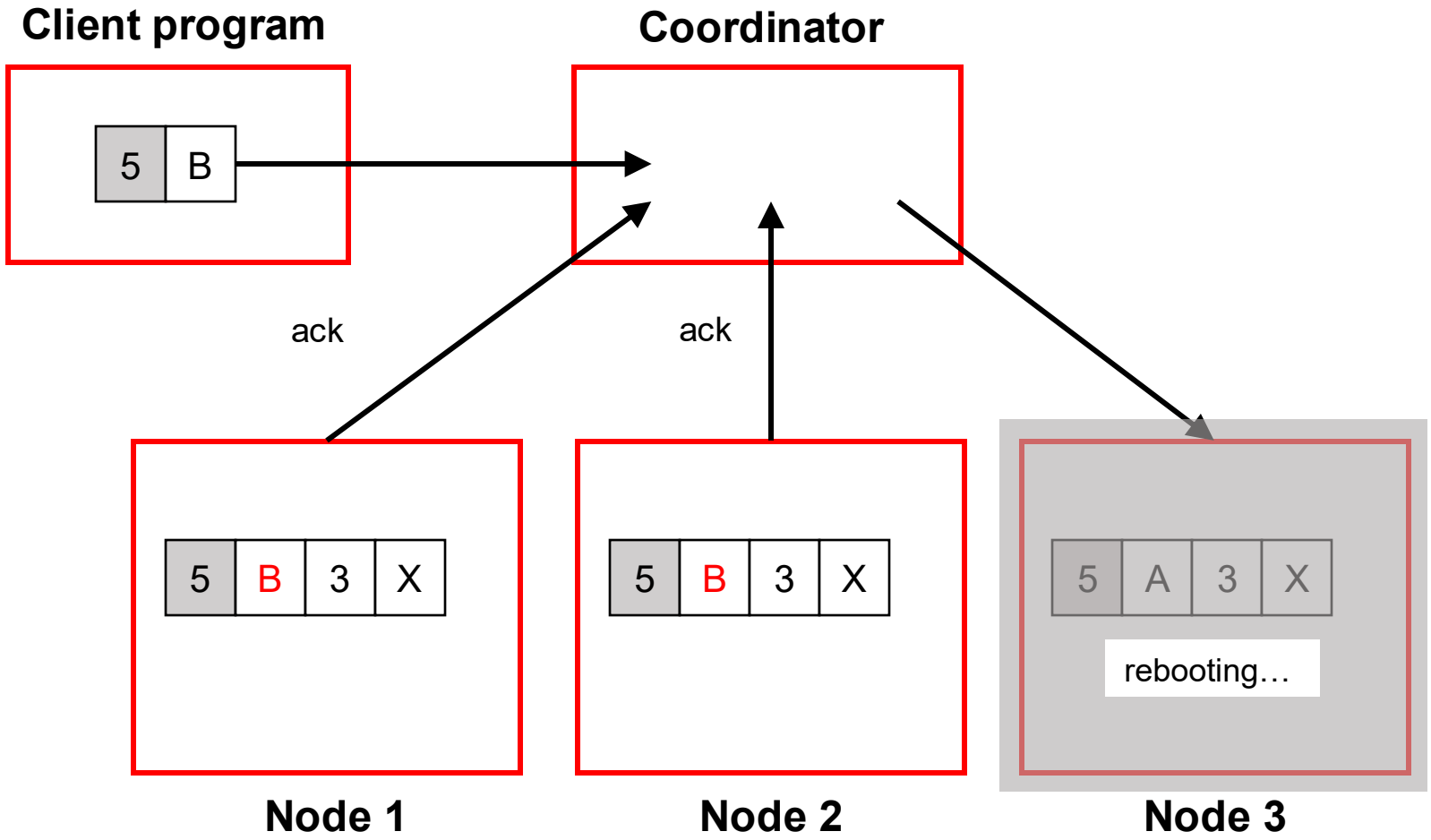
Dynamo writes

RF = 3. Coordinator will attempt to write data to all 3 replicas.



Dynamo writes

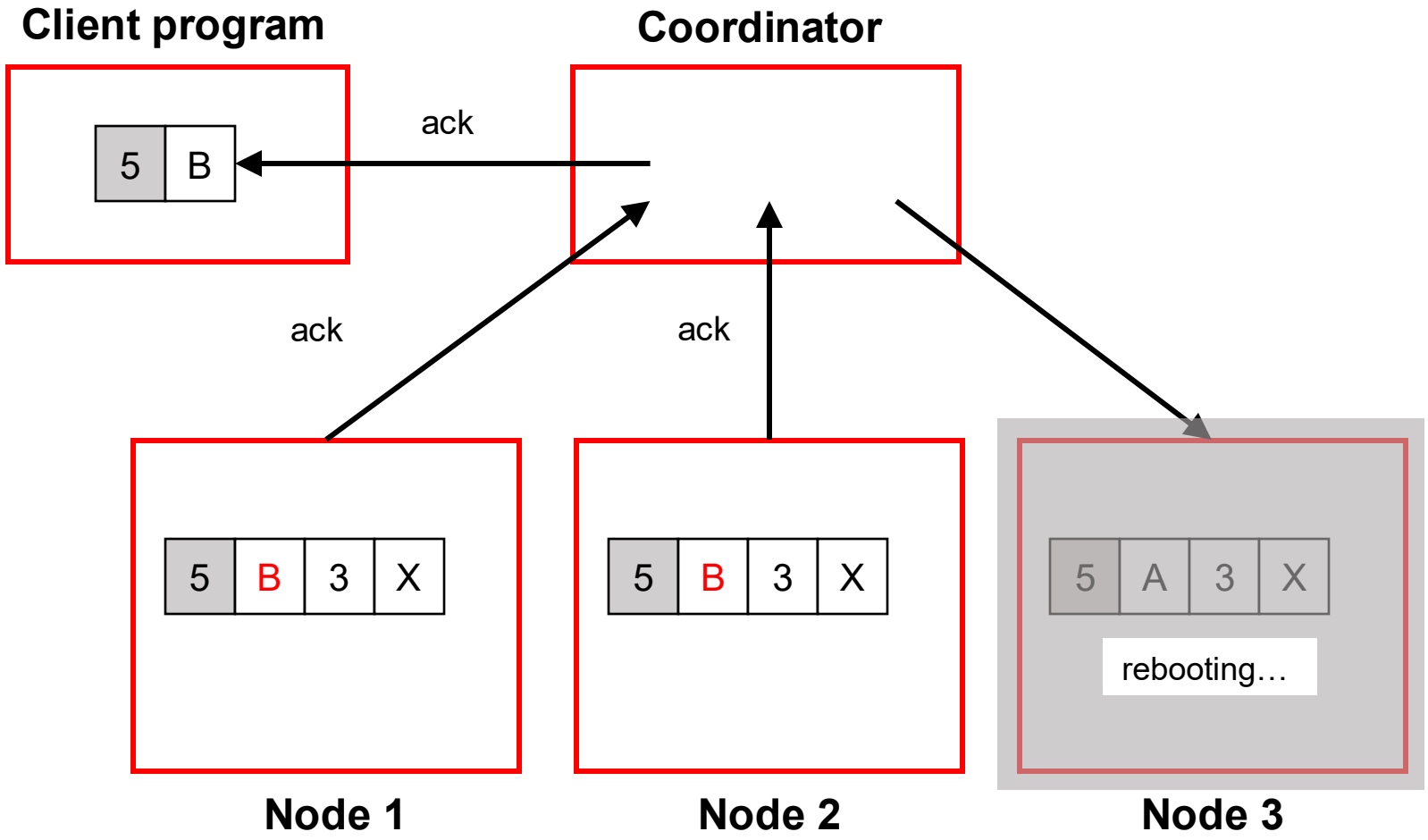
RF = 3. Coordinator will attempt to write data to all 3 replicas.



At what point should we send an ack back to the client?

Dynamo writes

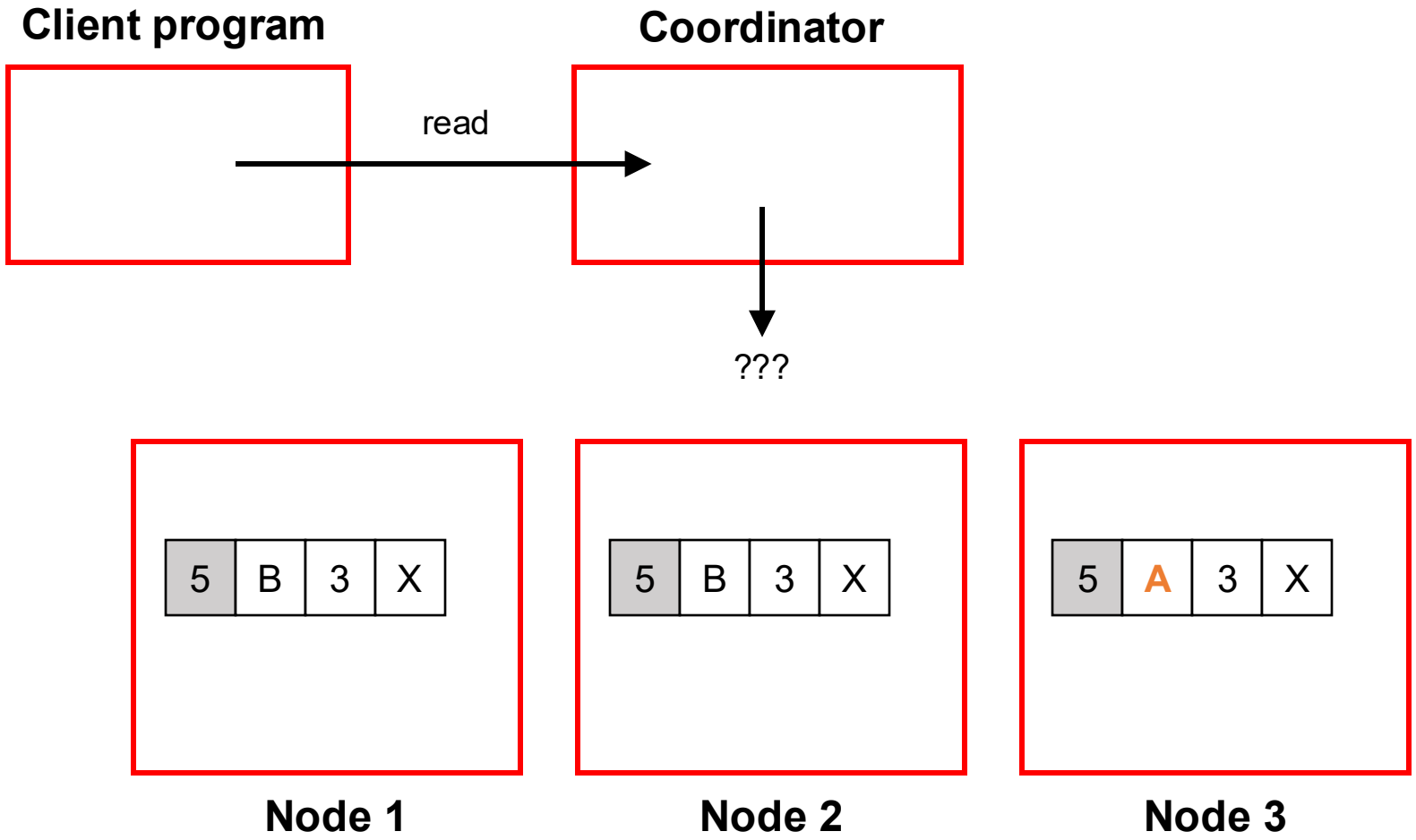
RF = 3. Coordinator will attempt to write data to all 3 replicas.



At what point should we send an ack back to the client?
Configurable: **W = 2** lets coordinator ack now, and data is fairly safe.

Dynamo reads

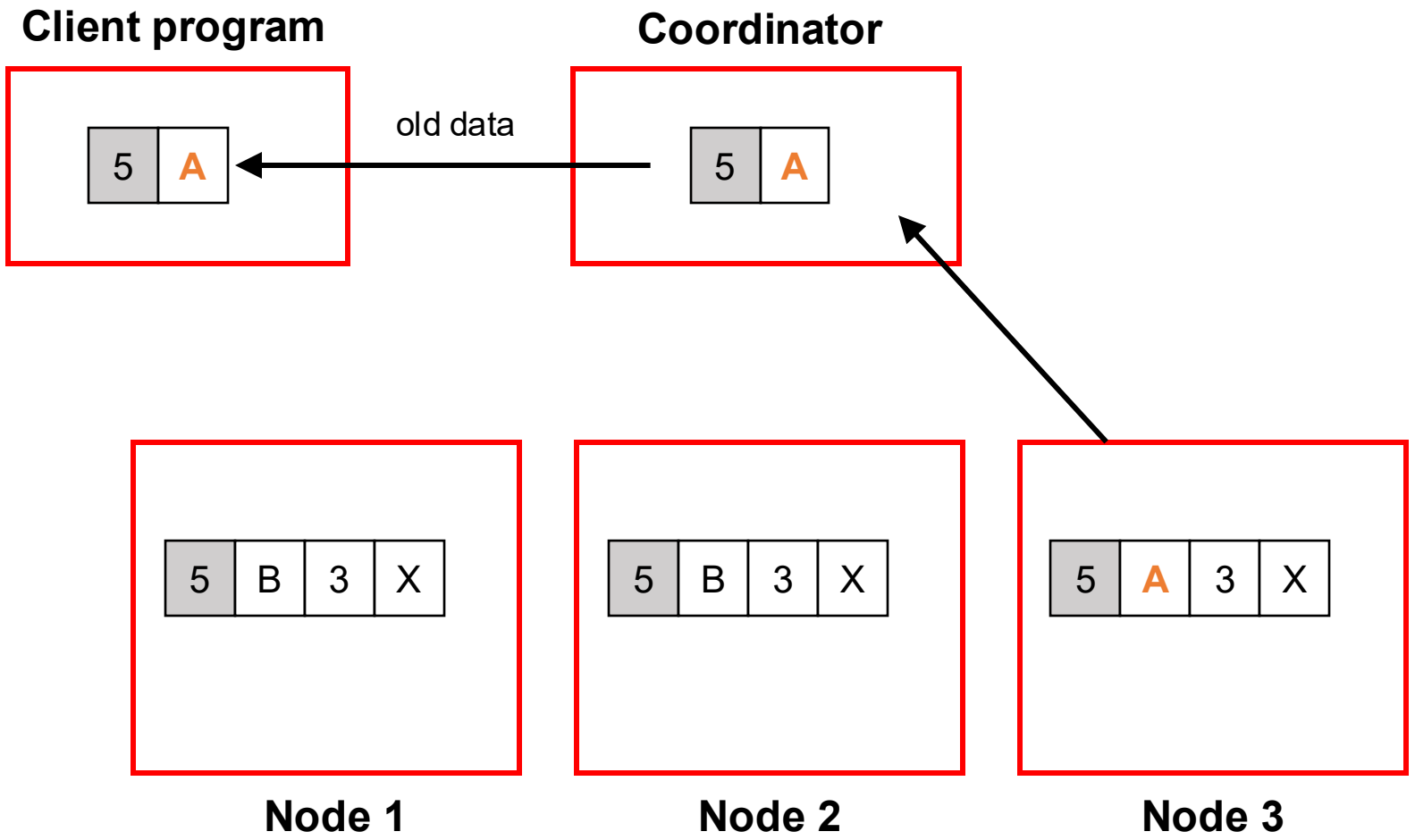
RF = 3



HDFS reads go to one replica. What if Dynamo tries that?

Dynamo reads

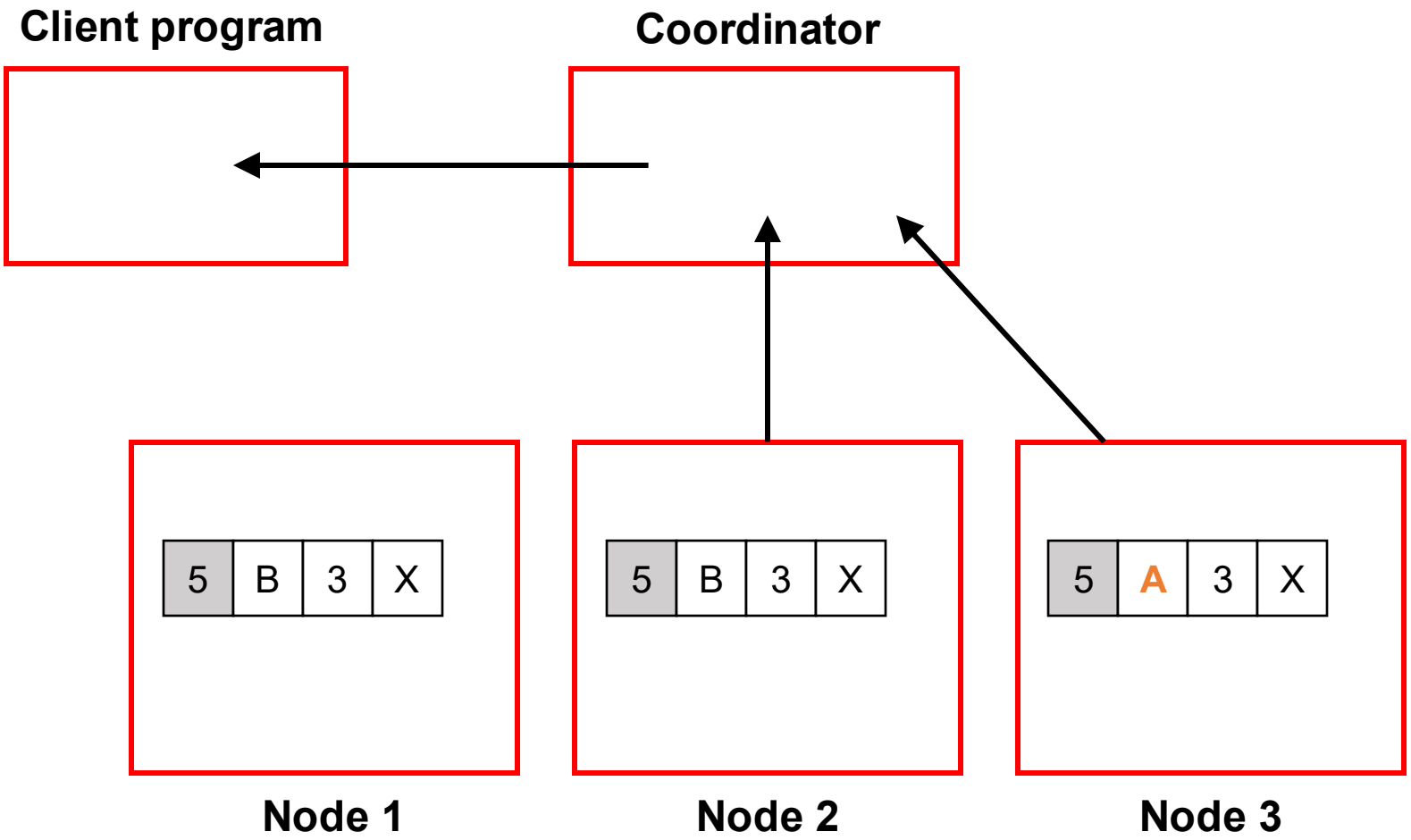
RF = 3



HDFS reads go to one replica. What if Dynamo tries that?

Dynamo reads

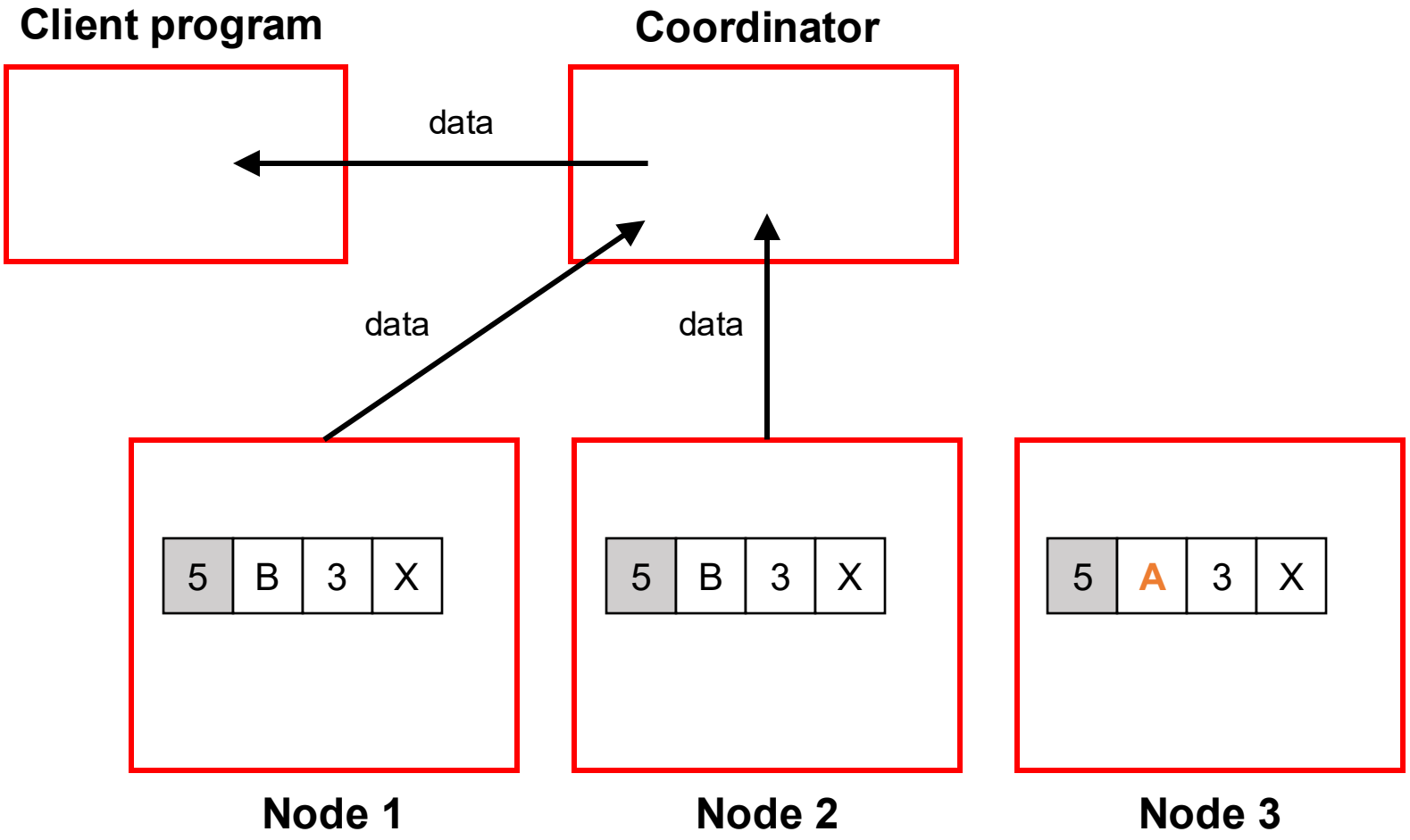
RF = 3



Read from **R** replicas (R is configurable). Here R = 2.
Hopefully at least one of the replicas has new data.

Dynamo reads

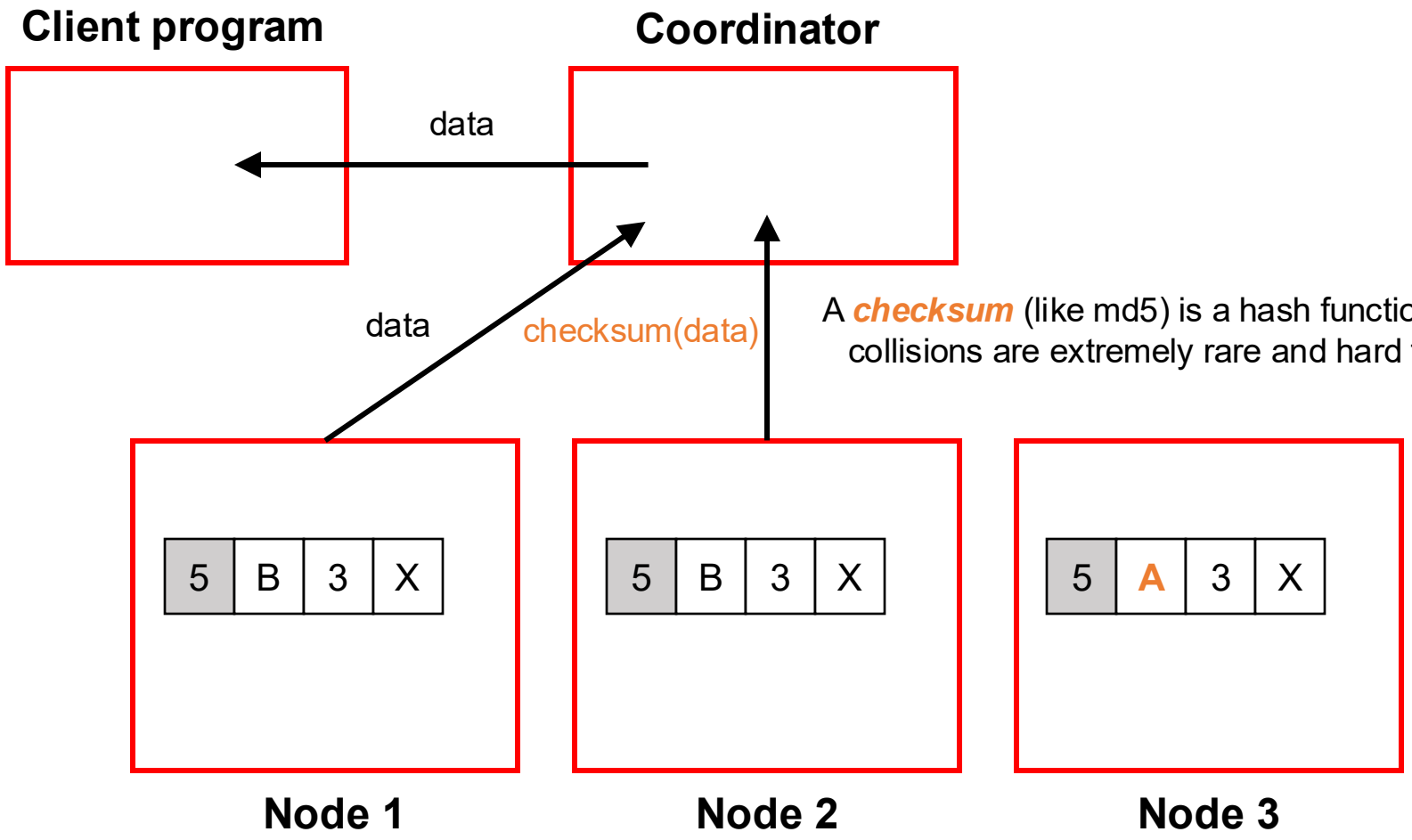
RF = 3



R = 2 means we'll often read identical data from two replicas (wasteful)

Dynamo reads

RF = 3



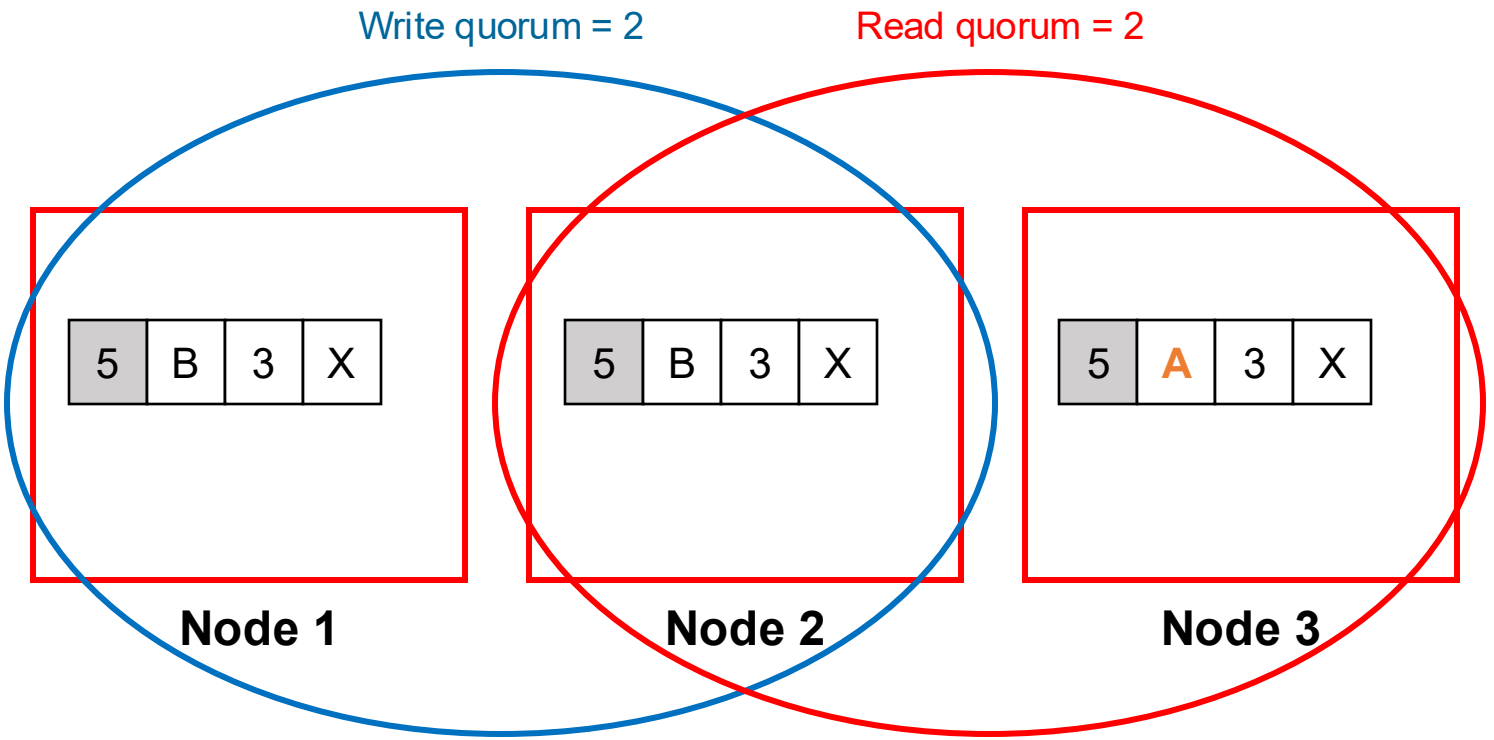
R = 2 means we'll often read identical data from two replicas (wasteful)

Optimization: Read one copy, and only request checksum from others.

When R + W > RF

RF = 3

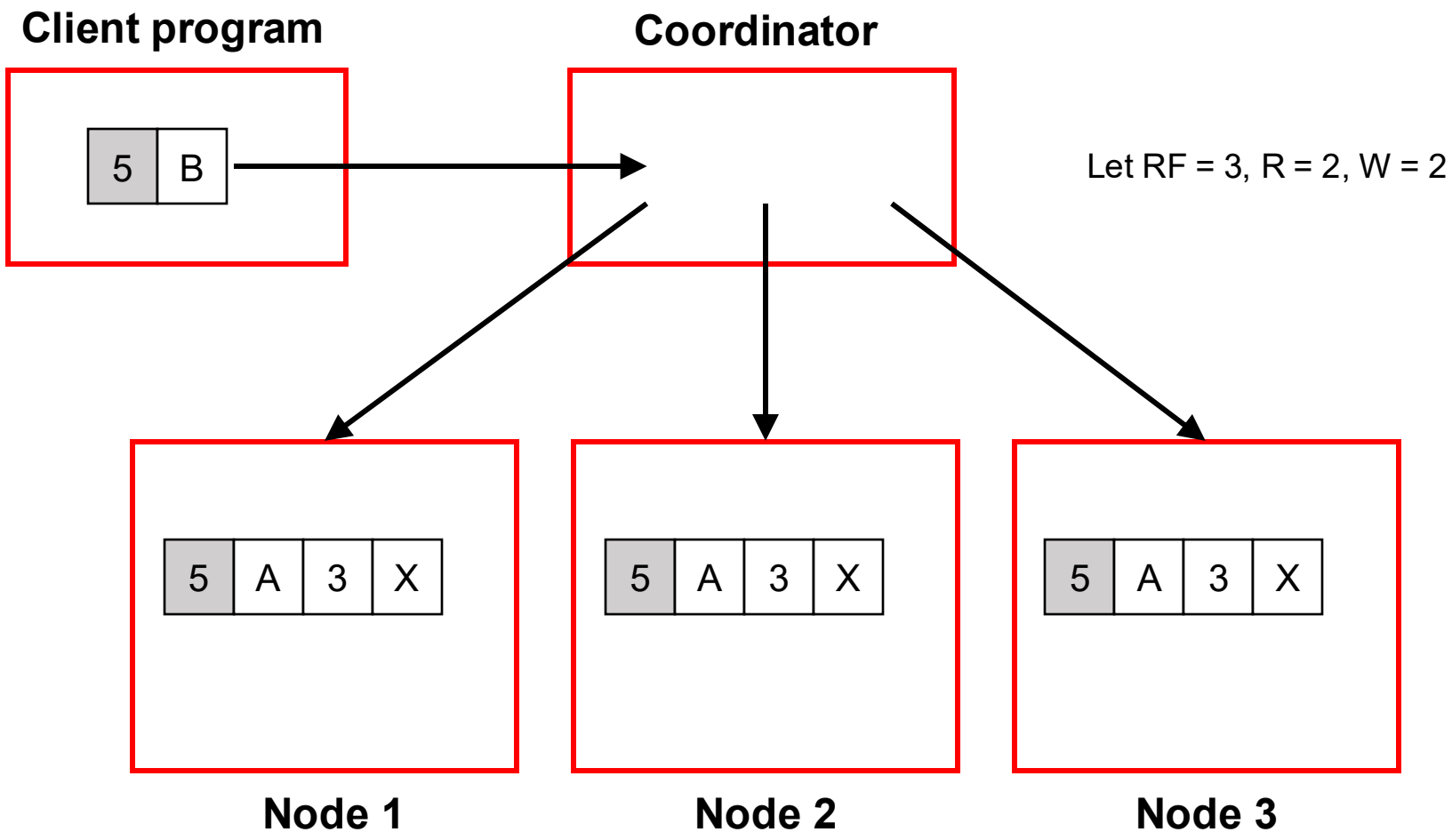
When R + W > RF, the replicas read + written will **overlap**.



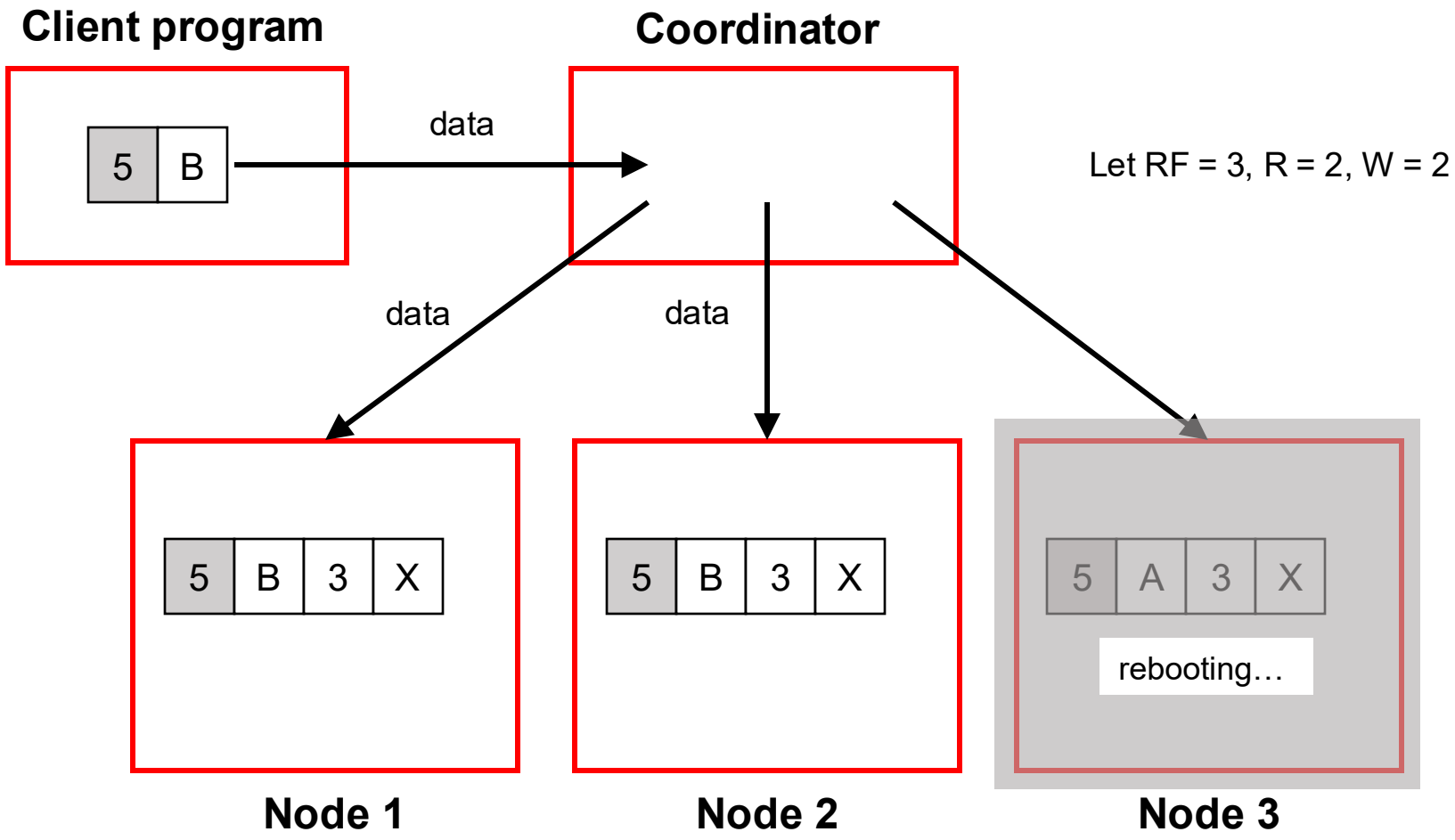
Tradeoff: Tuning R and W

RF	R	W	Behavior
3	**2**	**2**	**Parameters from the Dynamo paper:** **Relatively balanced configuration;** **Good durability, good R/W latency**
3	3	1	Slow reads, **weak durability, fast writes** Writes are highly available, therefore fast; Reads will not return data even if one node is down; reads may fail; Risk: If the one node that took the write fails permanently, we'll lose committed data.
3	1	3	**Slow writes,** strong durability, **fast reads** Reads are highly available, therefore fast, but have weak consistency; Writes are slow (from client's perspective) as they involve writing to three replicas.
3	3	3	More likely that **reads see all prior writes**?
3	1	1	Read quorum **doesn't overlap** write quorum Speed + availability more important than consistency

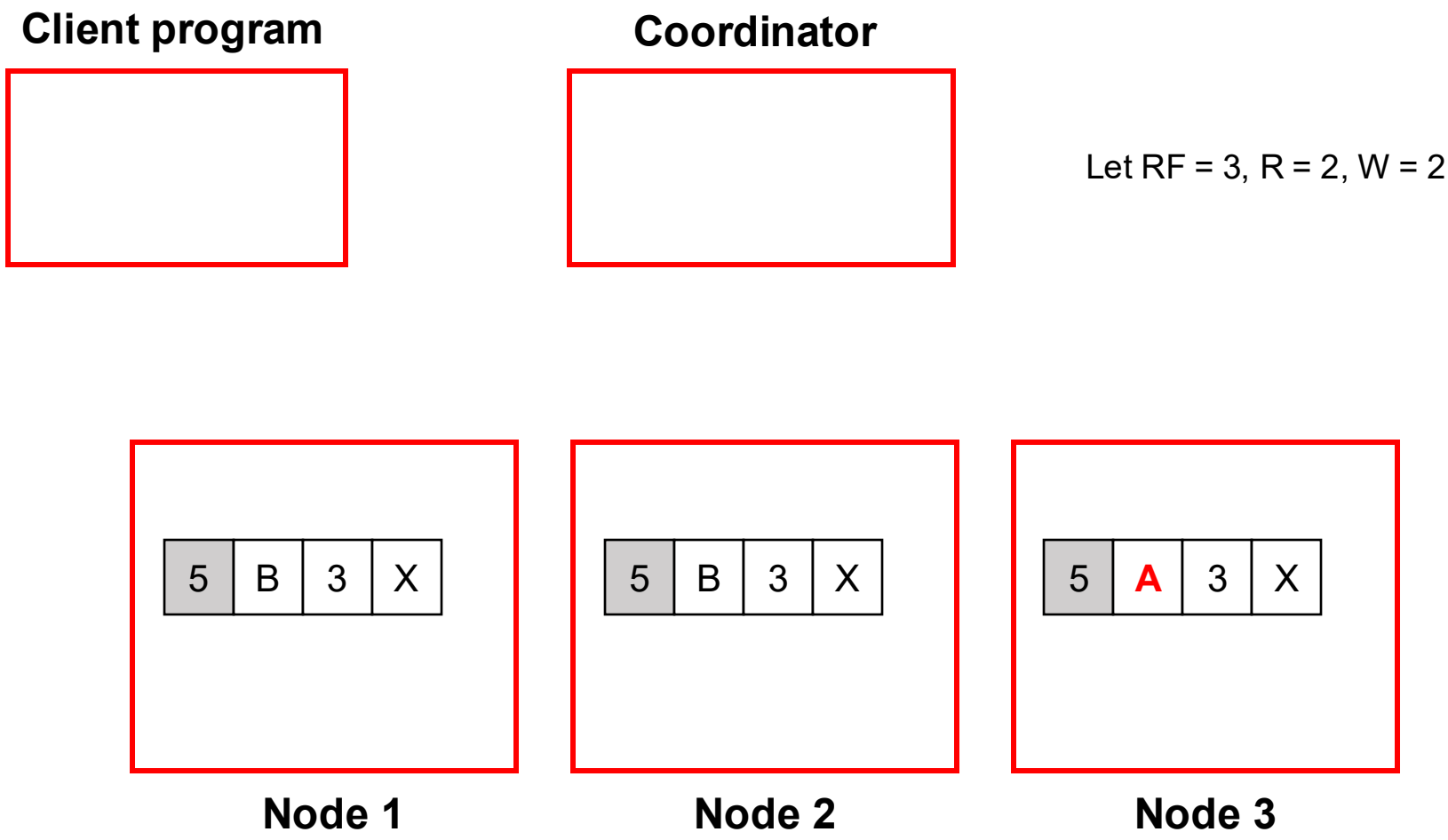
Getting conflicting versions



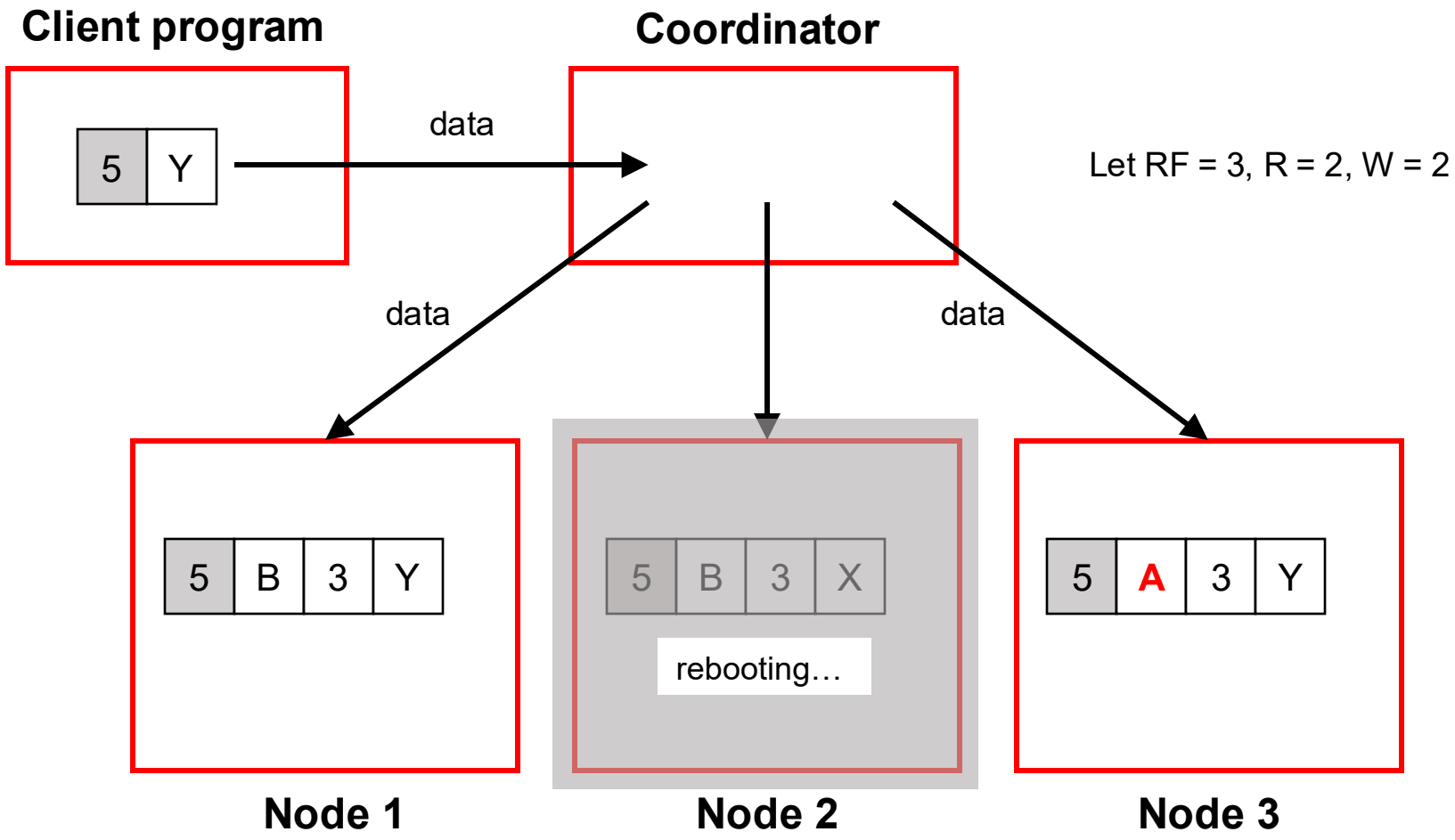
Getting conflicting versions



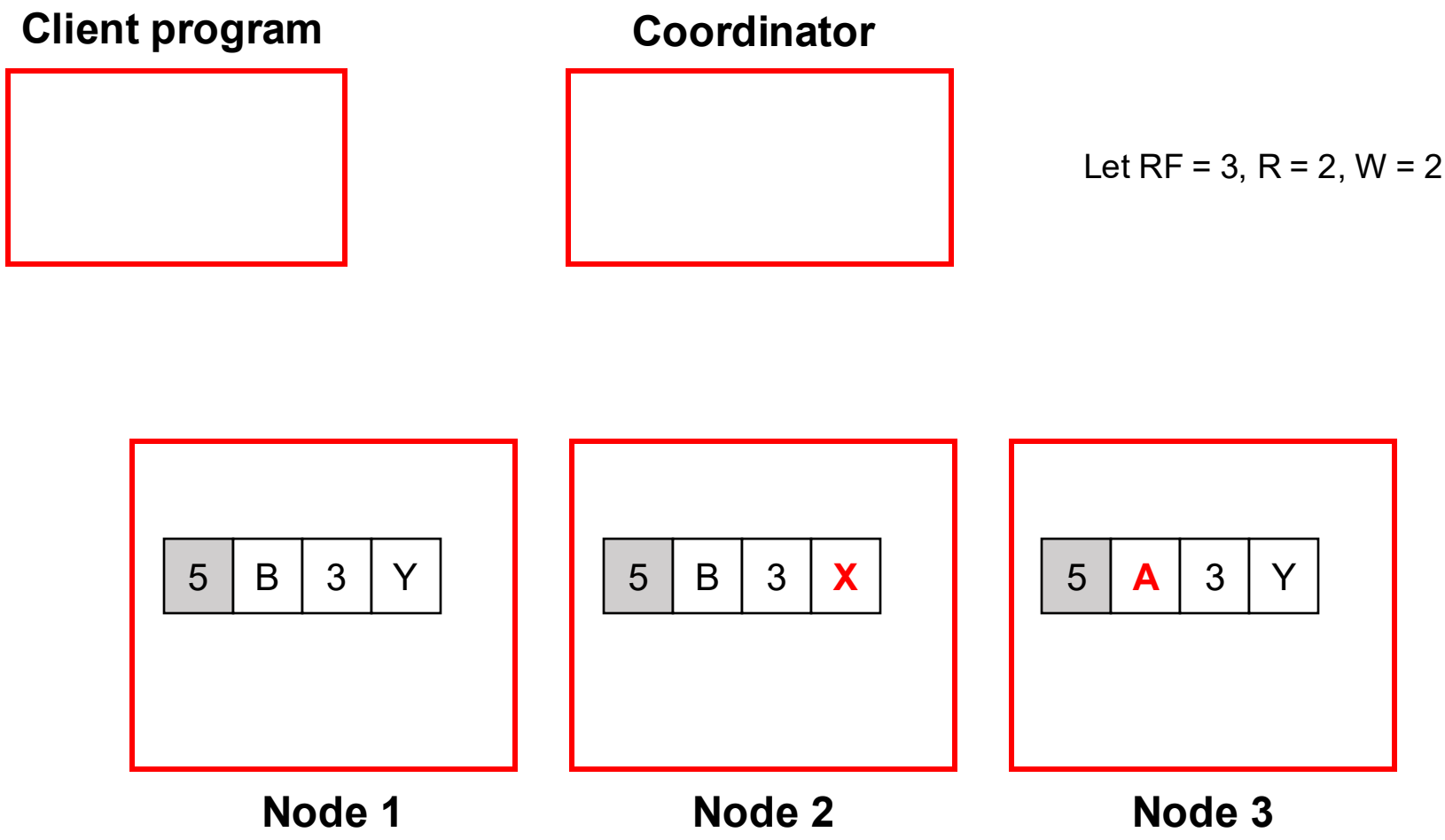
Getting conflicting versions



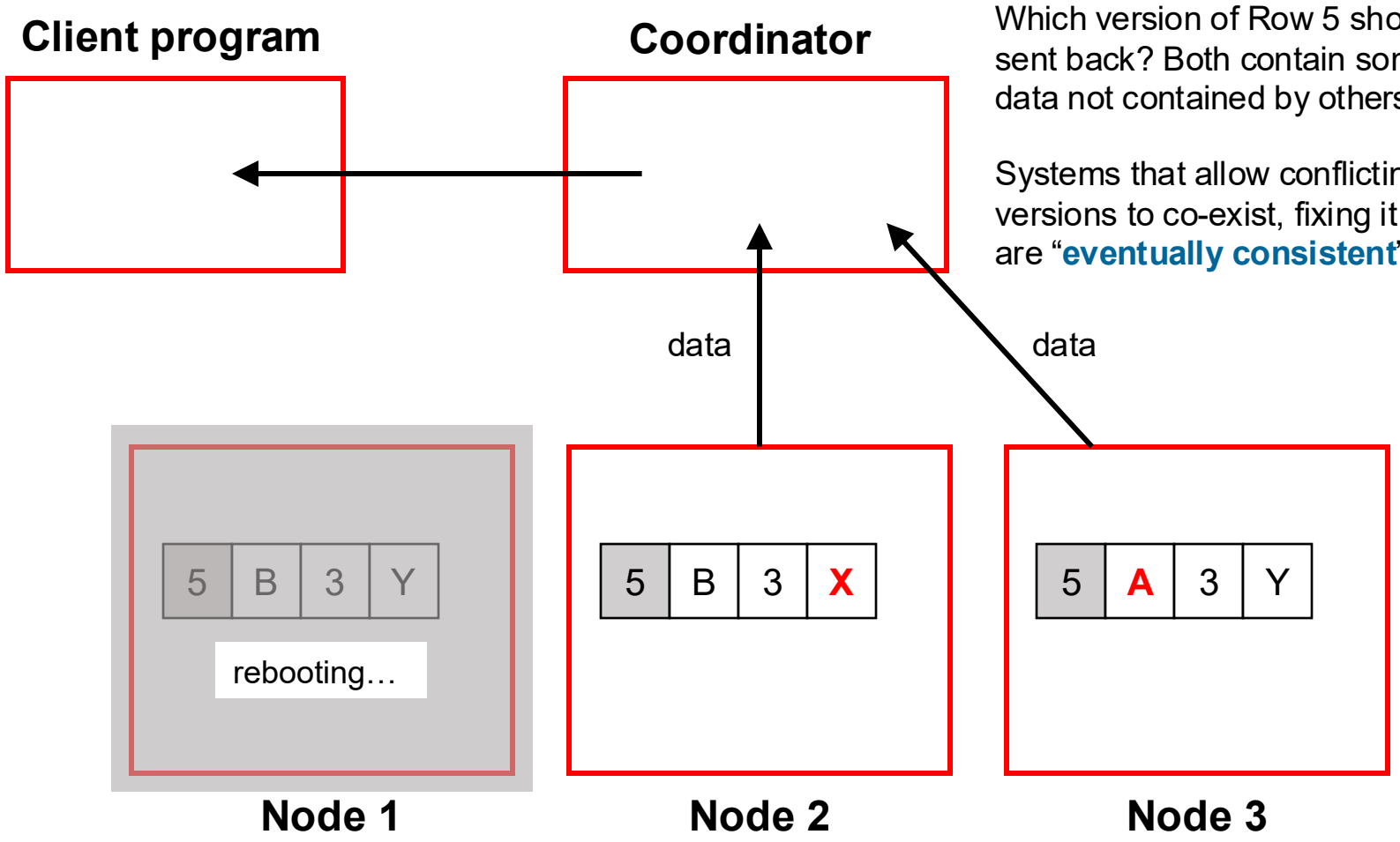
Getting conflicting versions



Getting conflicting versions



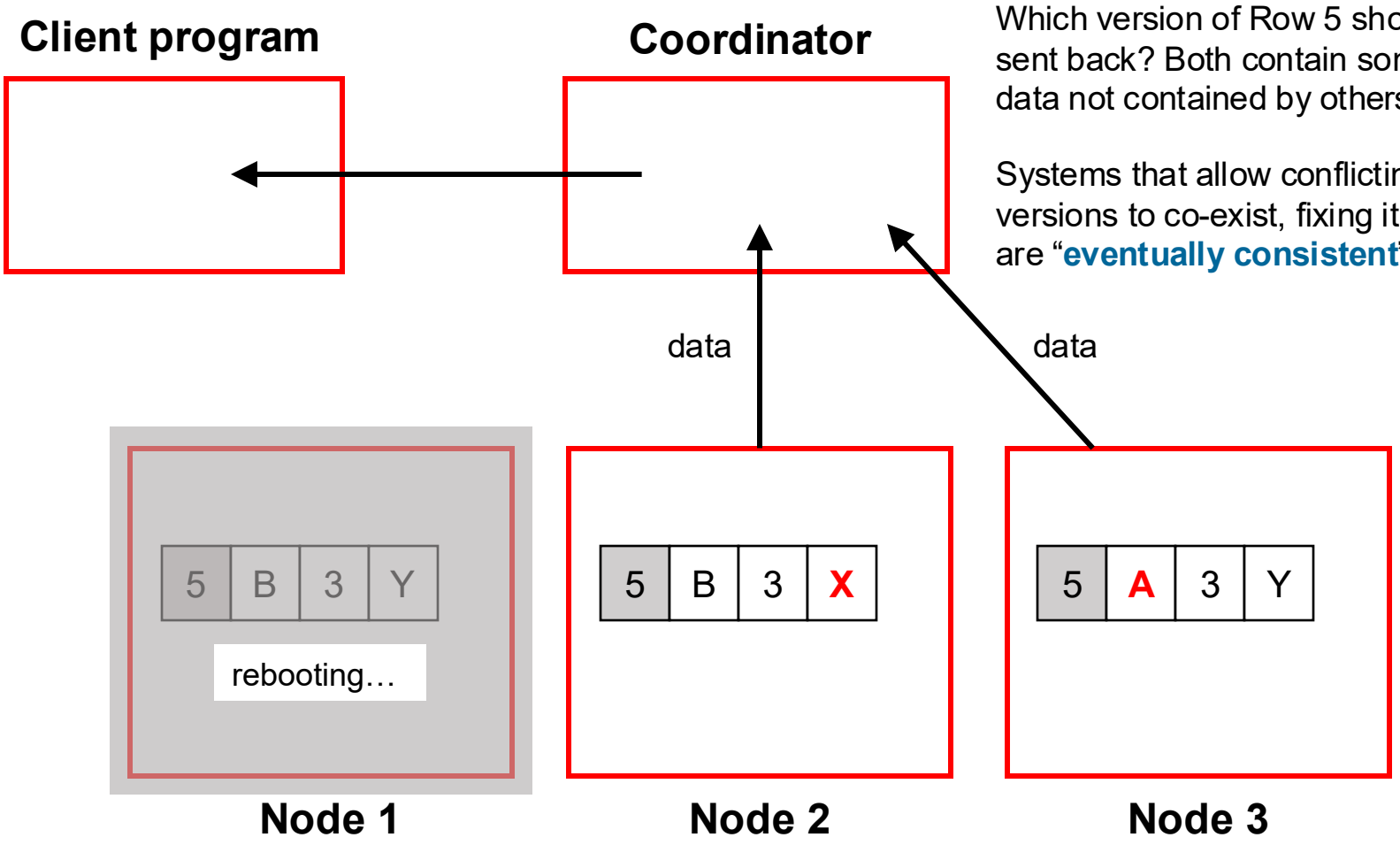
Getting conflicting versions



Which version of Row 5 should be sent back? Both contain some new data not contained by others.

Systems that allow conflicting versions to co-exist, fixing it up later are "**eventually consistent**".

Getting conflicting versions



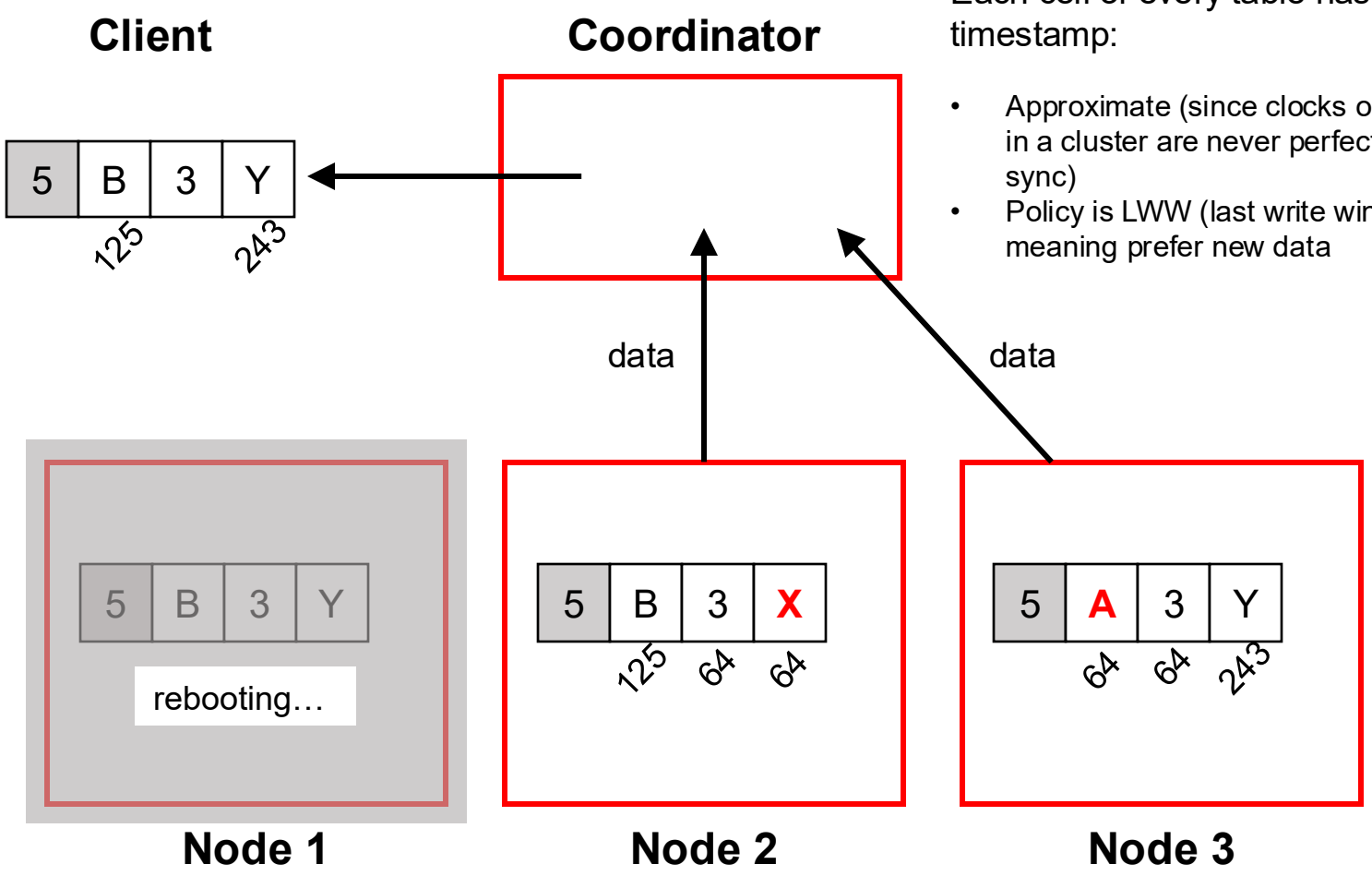
Which version of Row 5 should be sent back? Both contain some new data not contained by others.

Systems that allow conflicting versions to co-exist, fixing it up later are "**eventually consistent**".

Approach:

- Send all versions back to client, which will need specialized conflict resolution code
- Automatically combine them into a new row, and write that (if possible to all replicas)

Timestamps (logical clock)



Each cell of every table has a timestamp:

- Approximate (since clocks of nodes in a cluster are never perfectly in sync)
- Policy is LWW (last write wins), meaning prefer new data