

# [544] Cassandra Replication

Tyler Caraza-Harter

# Learning Objectives

- walk a token ring (in Cassandra, or other consistent hashing implementation) to identify multiple nodes responsible for a given row (while potentially skipping duplicate nodes in the same "failure domain")
- tune read/write quorum requirements to achieve desired tradeoffs in availability, durability, and performance
- describe common approaches to eventual consistency and conflict resolution

# Outline

Replication

Quorum Reads/Writes

Conflict Resolution

Cassandra Demos

# Replication

We **replicate** (create multiple copies on different nodes) to improve **durability** – meaning we don't want data to be lost when nodes die.

Cassandra lets us choose a different **RF** (replication factor) for each keyspace:

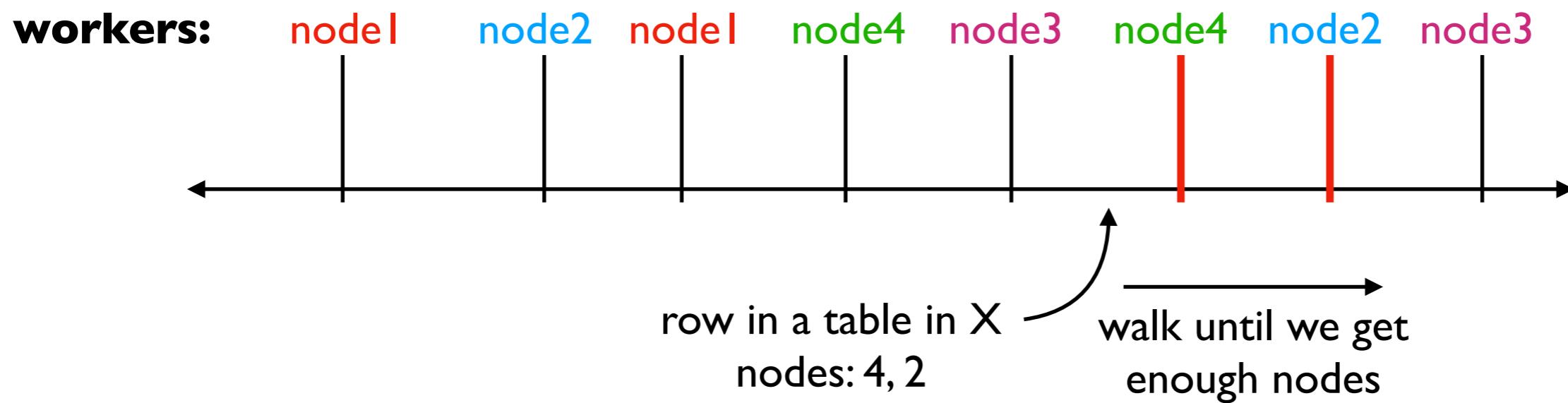
```
create keyspace X
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 2};
```

```
create keyspace Y
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 3};
```

# Replication

## Token Map:

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



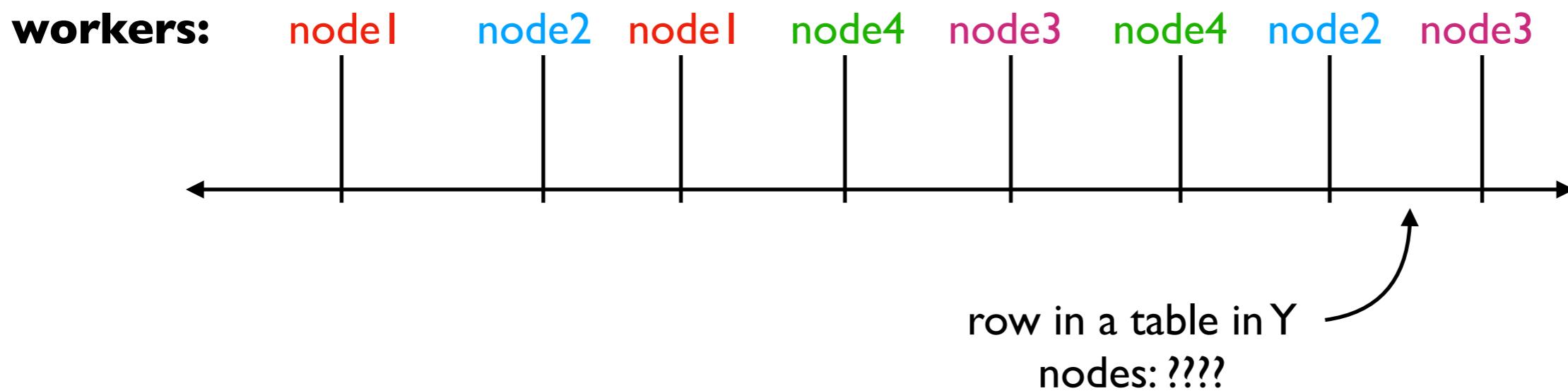
```
create keyspace X
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 2};
```

```
create keyspace Y
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 3};
```

# Replication

## Token Map:

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



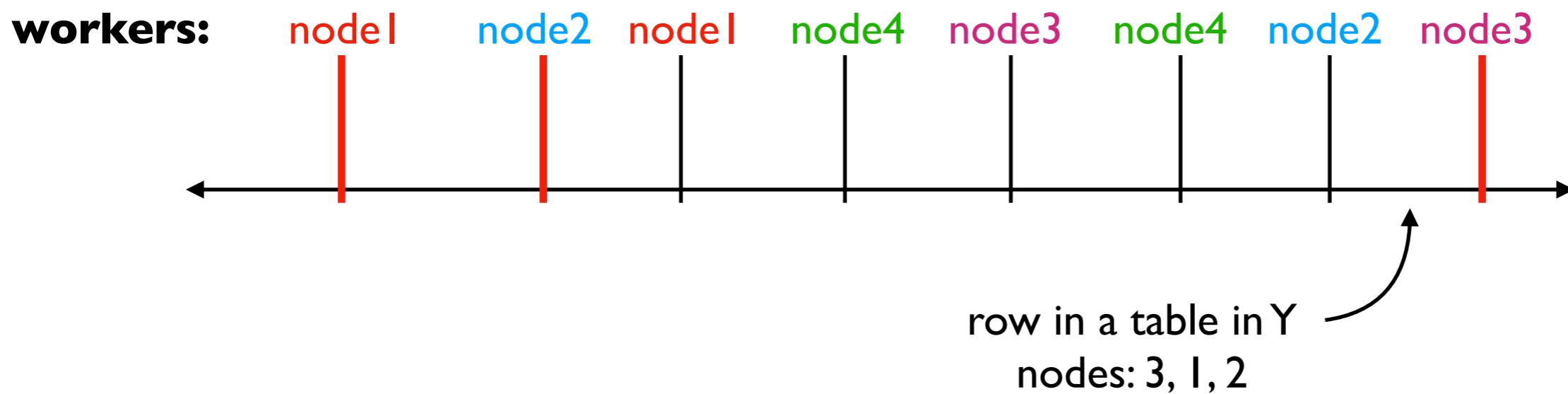
```
create keyspace X
with replication={'class': 'SimpleStrategy',
'replication_factor': 2};
```

```
create keyspace Y
with replication={'class': 'SimpleStrategy',
'replication_factor': 3};
```

# Replication

## Token Map:

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



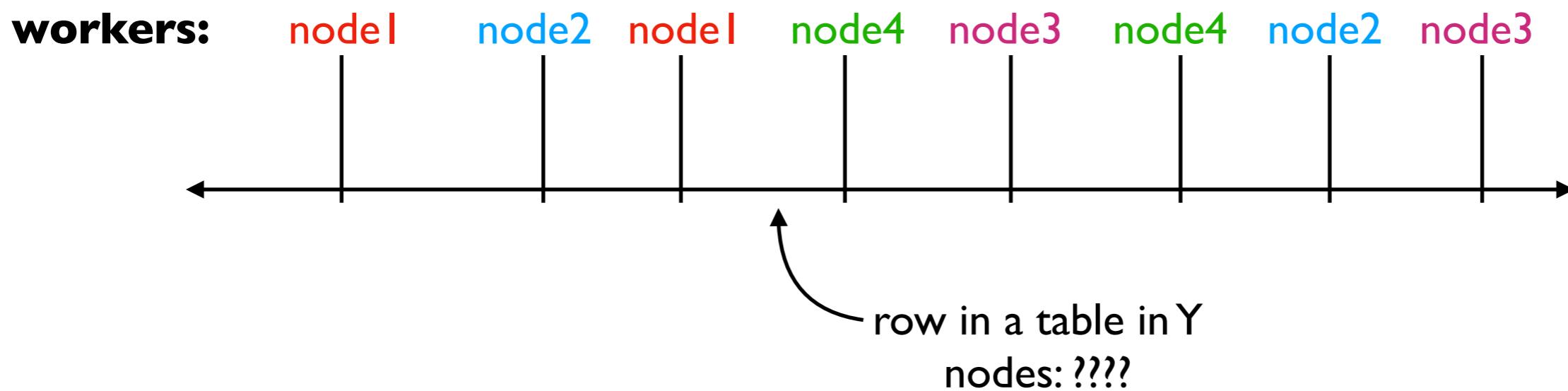
```
create keyspace X
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 2};
```

```
create keyspace Y
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 3};
```

# Replication

## Token Map:

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



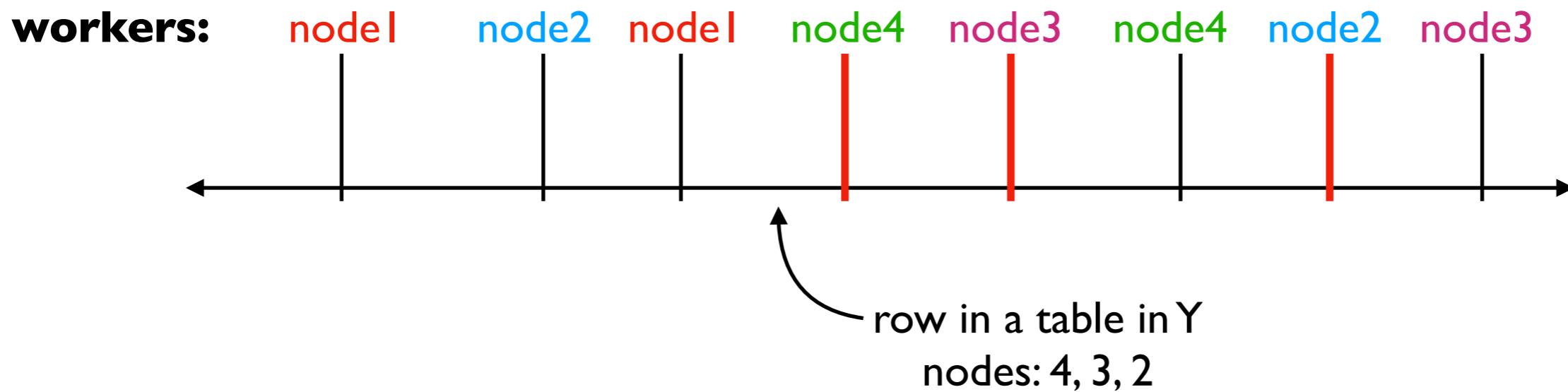
```
create keyspace X
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 2};
```

```
create keyspace Y
with replication={'class': 'SimpleStrategy',
                  'replication_factor': 3};
```

# Replication

## Token Map:

$\text{token}(\text{node1}) = \{\text{t1}, \text{t2}\}$   
 $\text{token}(\text{node2}) = \{\text{t3}, \text{t4}\}$   
 $\text{token}(\text{node3}) = \{\text{t5}, \text{t6}\}$   
 $\text{token}(\text{node4}) = \{\text{t7}, \text{t8}\}$



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

Cassandra can skip nodes as it "walks the ring".

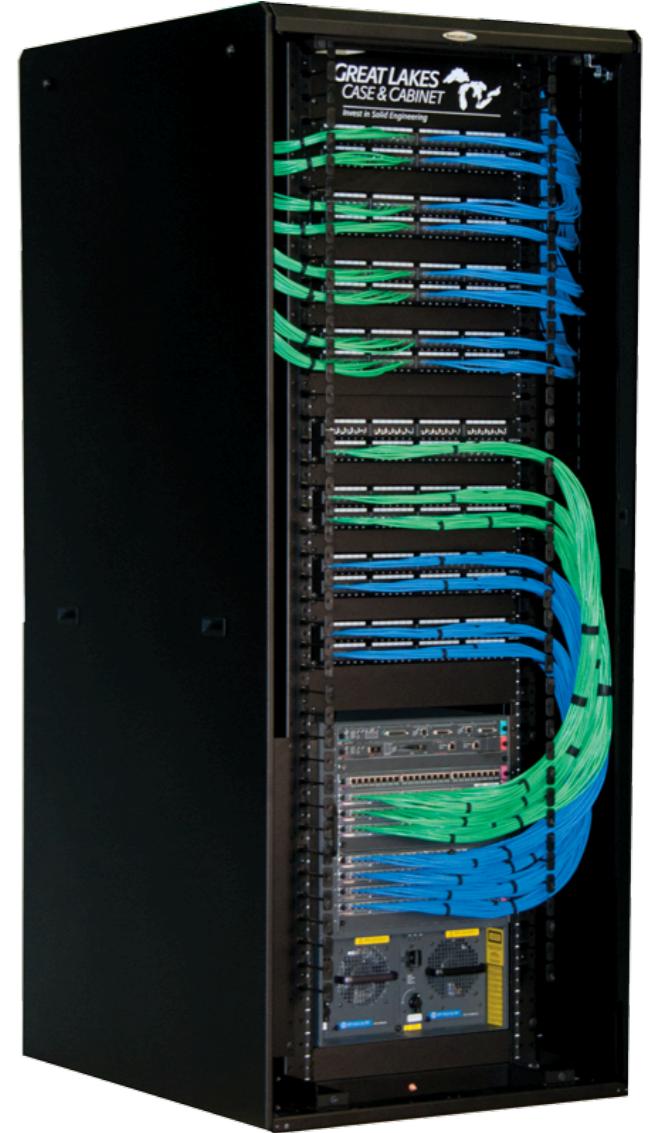
# Network Infrastructure



Server



Data Center



Rack

<https://www.dotmagazine.online/issues/digital-infrastructure-and-transforming-markets/data-center-models>

[https://buy.hpe.com/us/en/servers/proliant-dl-servers/proliant-dl10-servers/proliant-dl20-server/hpe-proliant-dl20-gen10-plus-e-2336-2-9ghz-6-core-1p-16gb-u-4sff-500w-rps-server/p/p4115-b21?ef\\_id=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd\\_JECmdyqNMAm2CKLSm\\_F-z6jTYDTQaAgMTEALw\\_wcB&s\\_kwcid=AL!13472!3!33!628972784!!g!318267171339!!1707918369!67076417419&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd\\_JECmdyqNMAm2CKLSm\\_F-z6jTYDTQaAgMTEALw\\_wcB](https://buy.hpe.com/us/en/servers/proliant-dl-servers/proliant-dl10-servers/proliant-dl20-server/hpe-proliant-dl20-gen10-plus-e-2336-2-9ghz-6-core-1p-16gb-u-4sff-500w-rps-server/p/p4115-b21?ef_id=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB&s_kwcid=AL!13472!3!33!628972784!!g!318267171339!!1707918369!67076417419&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNFjsg49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB)

[https://www.server-rack-online.com/g!910ent-4048sss.html?utm\\_medium=shoppingengine&utm\\_source=googlebase&utm\\_source=google&utm\\_medium=cpc&adpos=&scid=scplpg!910ent-4048sss&sc\\_intid=g!910ent-4048sss&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNEMYINPAA0RFGQIF0DsieCM6oh7i3kuJvJlpnmJAlOpAJ3RWTI1QMAaAqRnEALw\\_wcB](https://www.server-rack-online.com/g!910ent-4048sss.html?utm_medium=shoppingengine&utm_source=googlebase&utm_source=google&utm_medium=cpc&adpos=&scid=scplpg!910ent-4048sss&sc_intid=g!910ent-4048sss&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNEMYINPAA0RFGQIF0DsieCM6oh7i3kuJvJlpnmJAlOpAJ3RWTI1QMAaAqRnEALw_wcB)

# Correlated Failures

One server goes down, all of its vnodes are gone.



Server

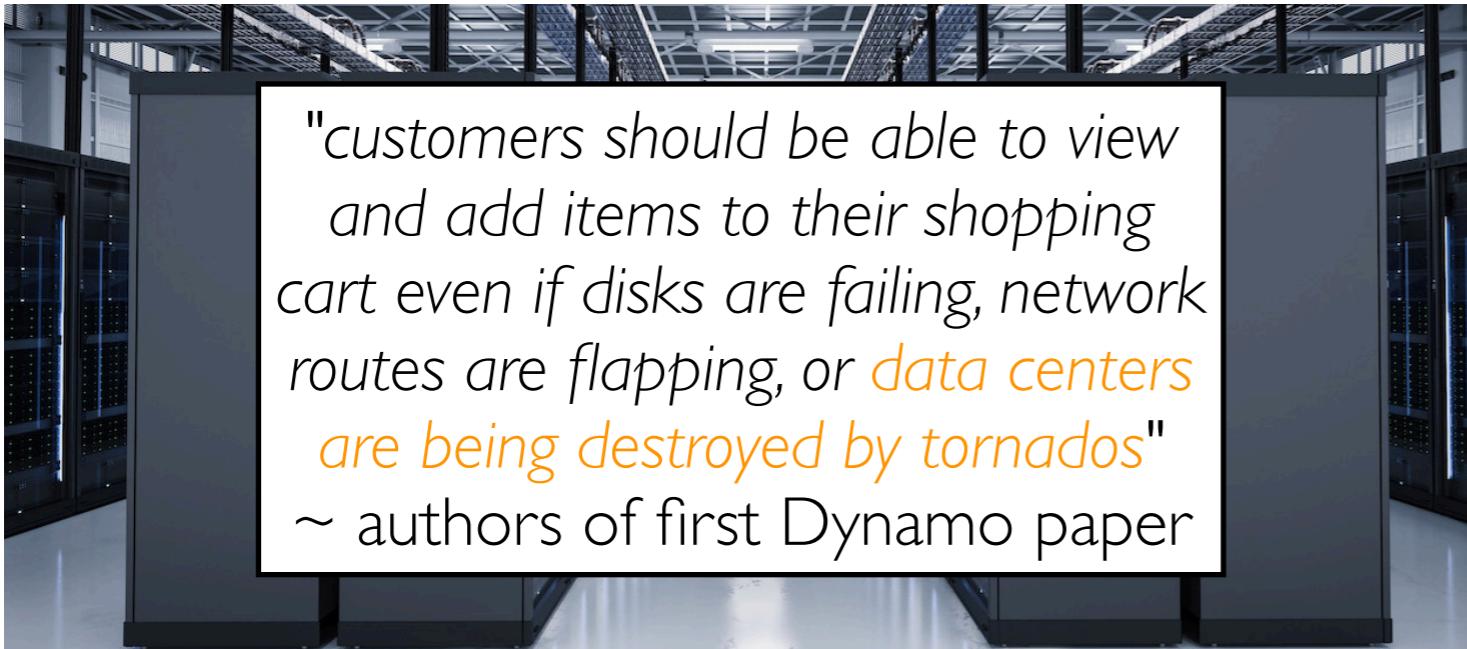
Whole-rack problems:

- top-of-rack switch fails
- rack's power supply fails



Rack

"customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or **data centers are being destroyed by tornados**"  
~ authors of first Dynamo paper



Data Center

<https://www.dotmagazine.online/issues/digital-infrastructure-and-transforming-markets/data-center-models>

[https://buy.hpe.com/us/en/servers/proliant-dl-servers/proliant-dl10-servers/proliant-dl20-server/hpe-proliant-dl20-gen10-plus-e-2336-2-9ghz-6-core-1p-16gb-u-4sff-500w-rps-server/p/p4115-b21?ef\\_id=Cj0KCQiAt66eBhCnARlsAKf3ZNFjs49UV6Zm33R7lkRqi-XOd\\_JECmdyqNMAm2CKLSm\\_F-z6jTYDTQaAgMTEALw\\_wcB;G:s&s\\_kwcid=AL!13472!3!33!628972784!!g!318267171339!!1707918369!67076417419&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNFjs49UV6Zm33R7lkRqi-XOd\\_JECmdyqNMAm2CKLSm\\_F-z6jTYDTQaAgMTEALw\\_wcB](https://buy.hpe.com/us/en/servers/proliant-dl-servers/proliant-dl10-servers/proliant-dl20-server/hpe-proliant-dl20-gen10-plus-e-2336-2-9ghz-6-core-1p-16gb-u-4sff-500w-rps-server/p/p4115-b21?ef_id=Cj0KCQiAt66eBhCnARlsAKf3ZNFjs49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB;G:s&s_kwcid=AL!13472!3!33!628972784!!g!318267171339!!1707918369!67076417419&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNFjs49UV6Zm33R7lkRqi-XOd_JECmdyqNMAm2CKLSm_F-z6jTYDTQaAgMTEALw_wcB)

[https://www.server-rack-online.com/g1910ent-4048sss.html?utm\\_medium=shoppingengine&utm\\_source=googlebase&utm\\_source=google&utm\\_medium=cpc&adpos=&scid=scplpg1910ent-4048sss&sc\\_intid=g1910ent-4048sss&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNEMYINPAA0RFGQIF0DsieCM6oh7i3kuJvJlpnmJAlOpAJ3RWTI1QMAaAqRnEALw\\_wcB](https://www.server-rack-online.com/g1910ent-4048sss.html?utm_medium=shoppingengine&utm_source=googlebase&utm_source=google&utm_medium=cpc&adpos=&scid=scplpg1910ent-4048sss&sc_intid=g1910ent-4048sss&gclid=Cj0KCQiAt66eBhCnARlsAKf3ZNEMYINPAA0RFGQIF0DsieCM6oh7i3kuJvJlpnmJAlOpAJ3RWTI1QMAaAqRnEALw_wcB)

# Replication Policy

Cassandra replication strategies are "pluggable", with a couple built-in options.

## SimpleStrategy

- all nodes are considered equal
- skips vnodes on same machine
- ignores rack and data center placement
- used in CS 544

## NetworkTopologyStrategy

- considers data centers and racks
- when walking the ring, some vnodes may be skipped to protect against various kinds of correlated failure

# Worksheet

# Outline

Replication

Quorum Reads/Writes

Conflict Resolution

Cassandra Demos

# Write Acks: WhatsApp Example

## How to check read receipts

 Android    iPhone    KaiOS

Check marks will appear next to each message you send. Here's what each one indicates:

- ✓ The message was successfully sent.
- ✅ The message was successfully delivered to the recipient's phone or any of their linked devices.
- ✅ The recipient has read your message.

<https://faq.whatsapp.com/665923838265756>

these are examples of "acks" (acknowledgements)

In distributed storage systems/databases, 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 data center could be destroyed

Obviously, no data is ever completely safe against any circumstance (e.g., comet strikes earth, leading to destruction of humankind and all our data centers).

# Write Acks: WhatsApp Example

**How to check read receipts**  Copy link

 Android    iPhone    KaiOS

Check marks will appear next to each message you send. Here's what each one

- ✓ The message was successfully sent.
- ✅ The message was successfully delivered to the recipient's phone or **any of their linked devices**.
- ✅ The recipient has read your message.

<https://faq.whatsapp.com/665923838265756>

stronger definition: **all** devices  
(in case one device fails)

these are examples of "**acks**" (acknowledgements)

In distributed storage systems/databases, 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 data center could be destroyed

Obviously, no data is ever completely safe against any circumstance (e.g., comet strikes earth, leading to destruction of humankind and all our data centers).

# Write Acks: WhatsApp Example

## How to check read receipts

 Android    iPhone    KaiOS

Check marks will appear next to each message you send. Here's what each one indicates:

- ✓ The message was successfully sent.
- ✅ The message was successfully delivered to the recipient's phone or any of their linked devices.
- ✅ The recipient has read your message. <https://faq.whatsapp.com/665923838265756>

these are examples of "acks" (acknowledgements)

two checks (in WhatsApp) mean the message reached the destination.

Does *only one check* mean the message has *NOT* reached the destination?

# Write Acks: WhatsApp Example

**How to check read receipts**

 Android    iPhone    KaiOS

Check marks will appear next to each message you send. Here's what each one indicates:

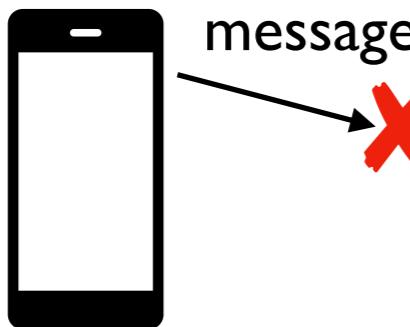
- ✓ The message was successfully sent.
- ✅ The message was successfully delivered to the recipient's phone or any of their linked devices.
- ✅ The recipient has read your message.

<https://faq.whatsapp.com/665923838265756>

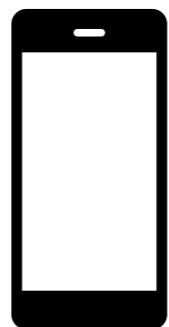
these are examples of "acks" (acknowledgements)

two checks (in WhatsApp) mean the message reached the destination.

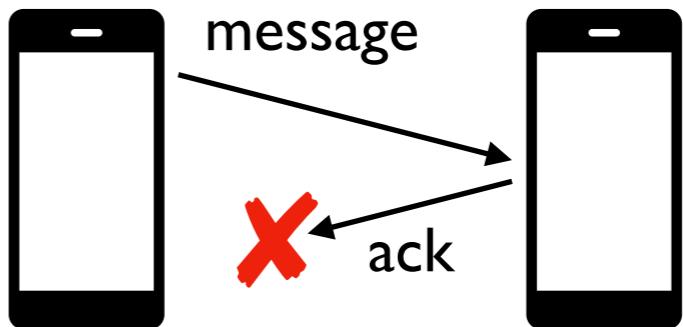
Does *only one check* mean the message has NOT reached the destination?



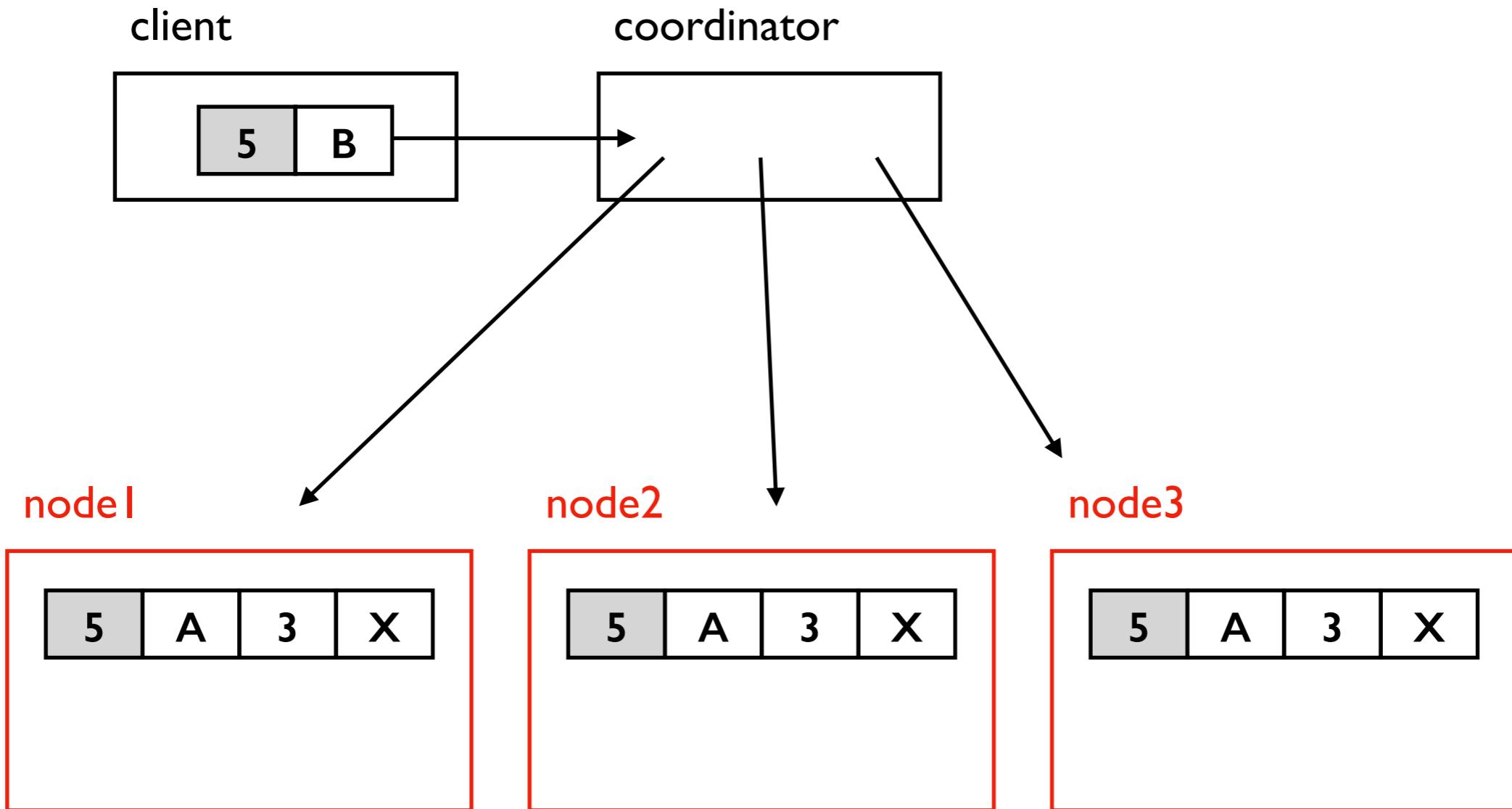
scenario 1



scenario 2

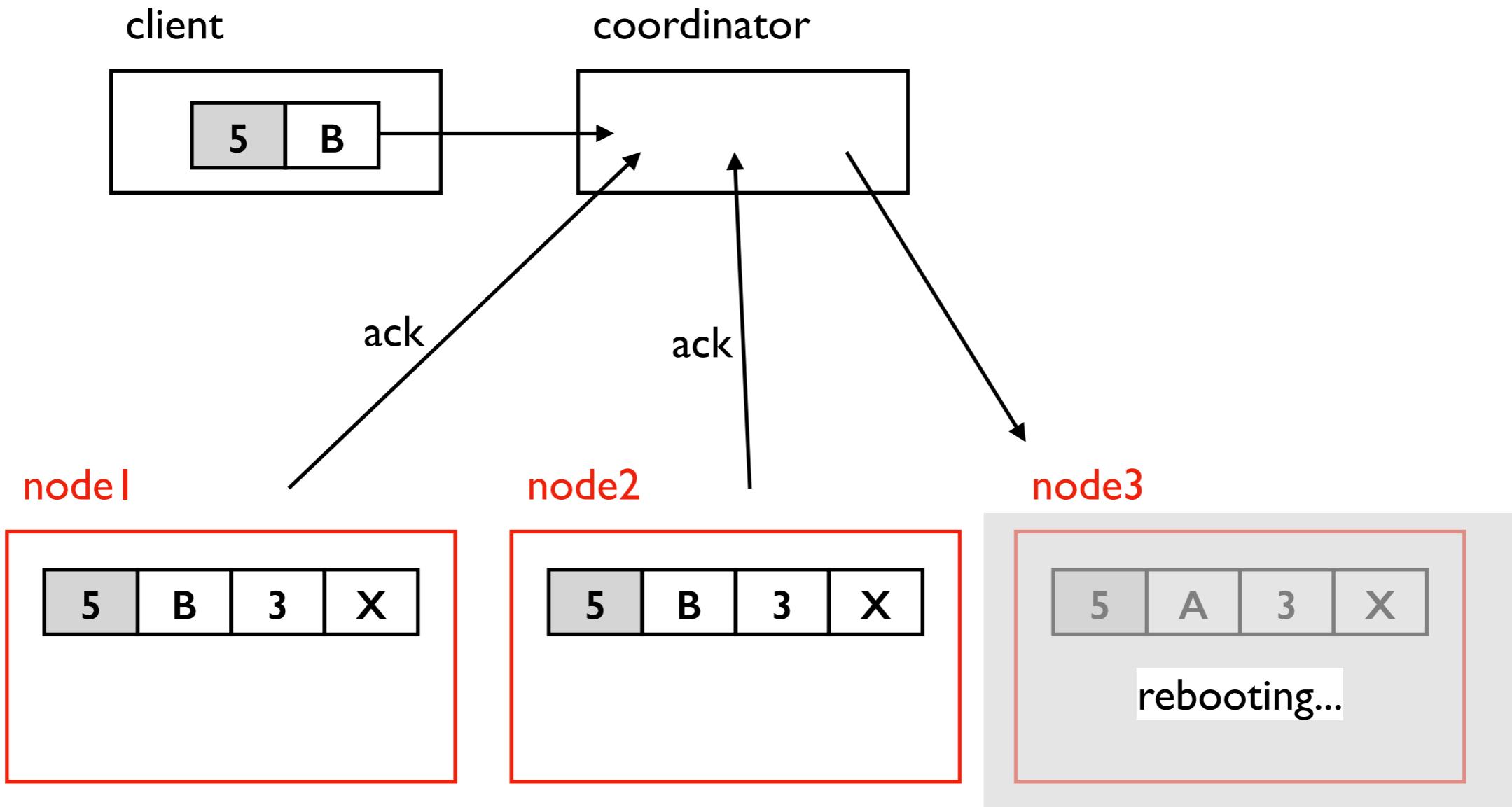


# Cassandra Writes



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

# Cassandra Writes

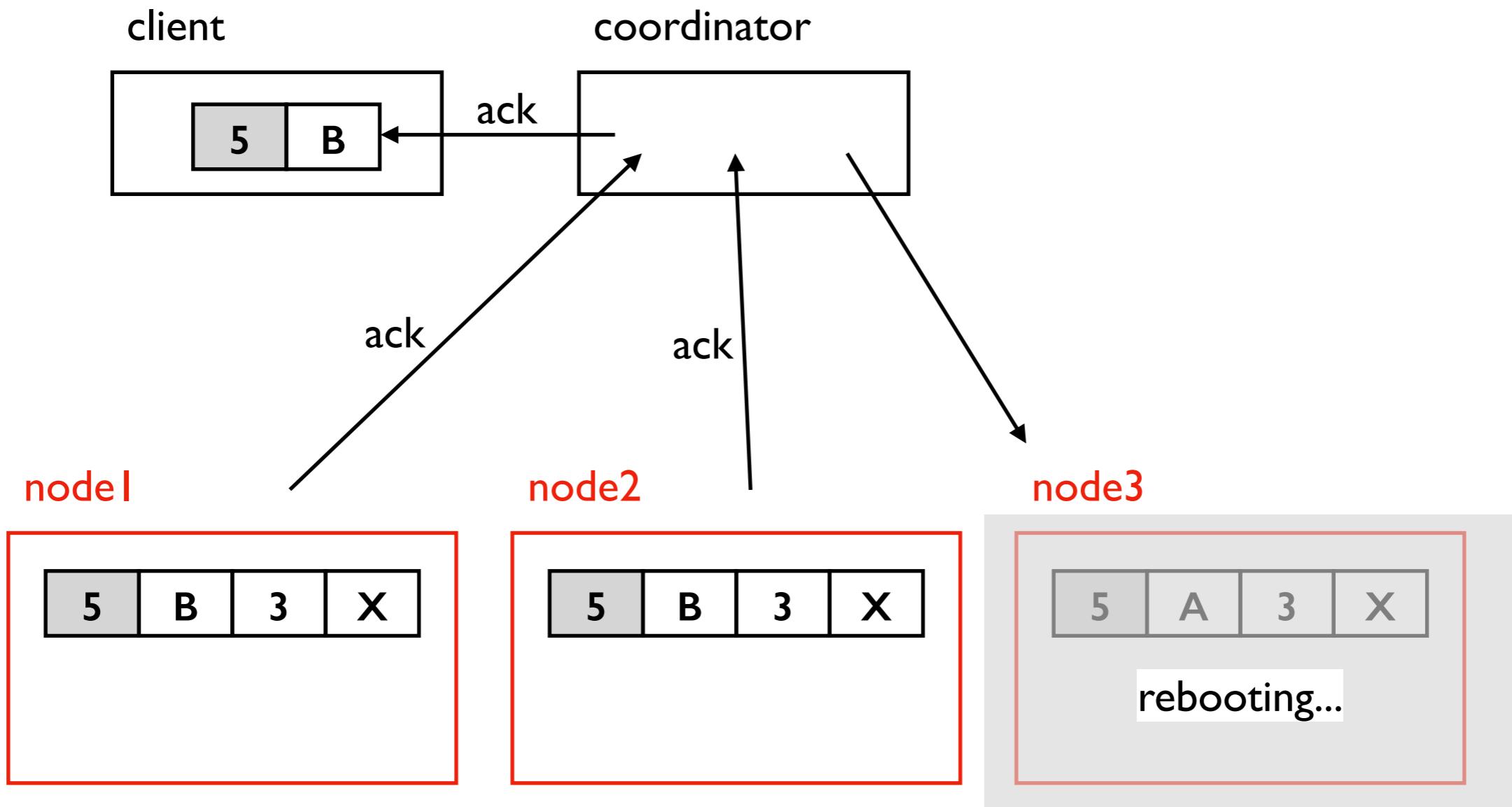


Say RF=3. Coordinator will attempt to write data to all 3 replicas.

Nodes send acks to coordinator.

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

# Cassandra Writes



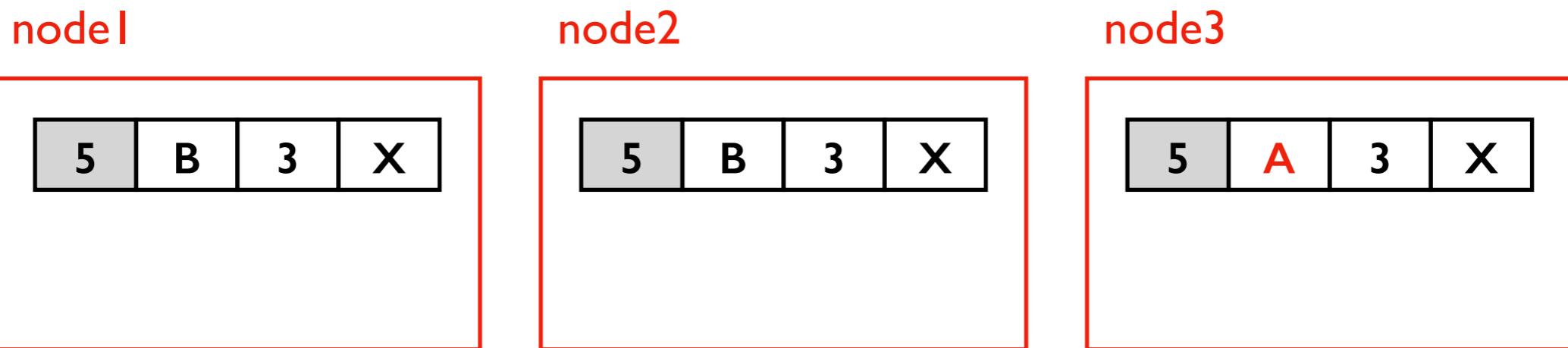
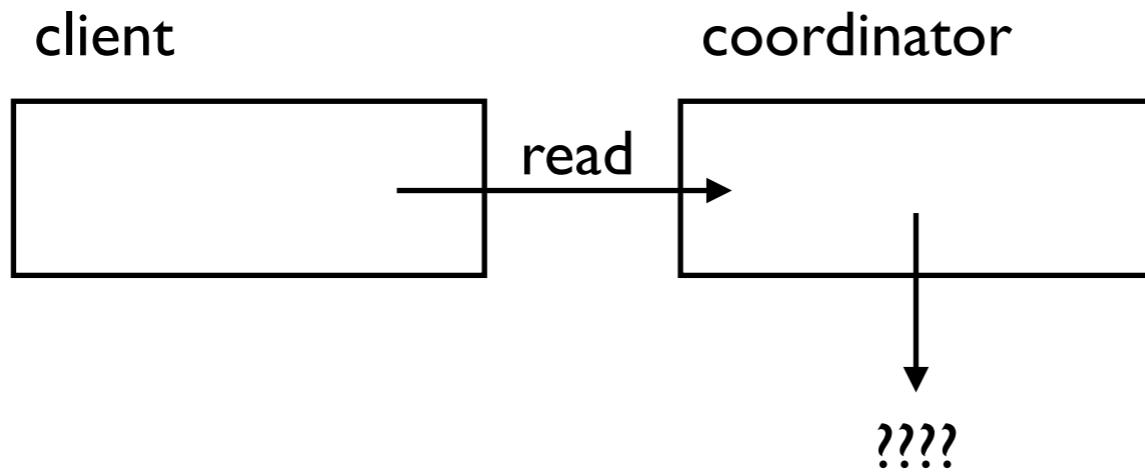
Say RF=3. Coordinator will attempt to write data to all 3 replicas.

Nodes send acks to coordinator.

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

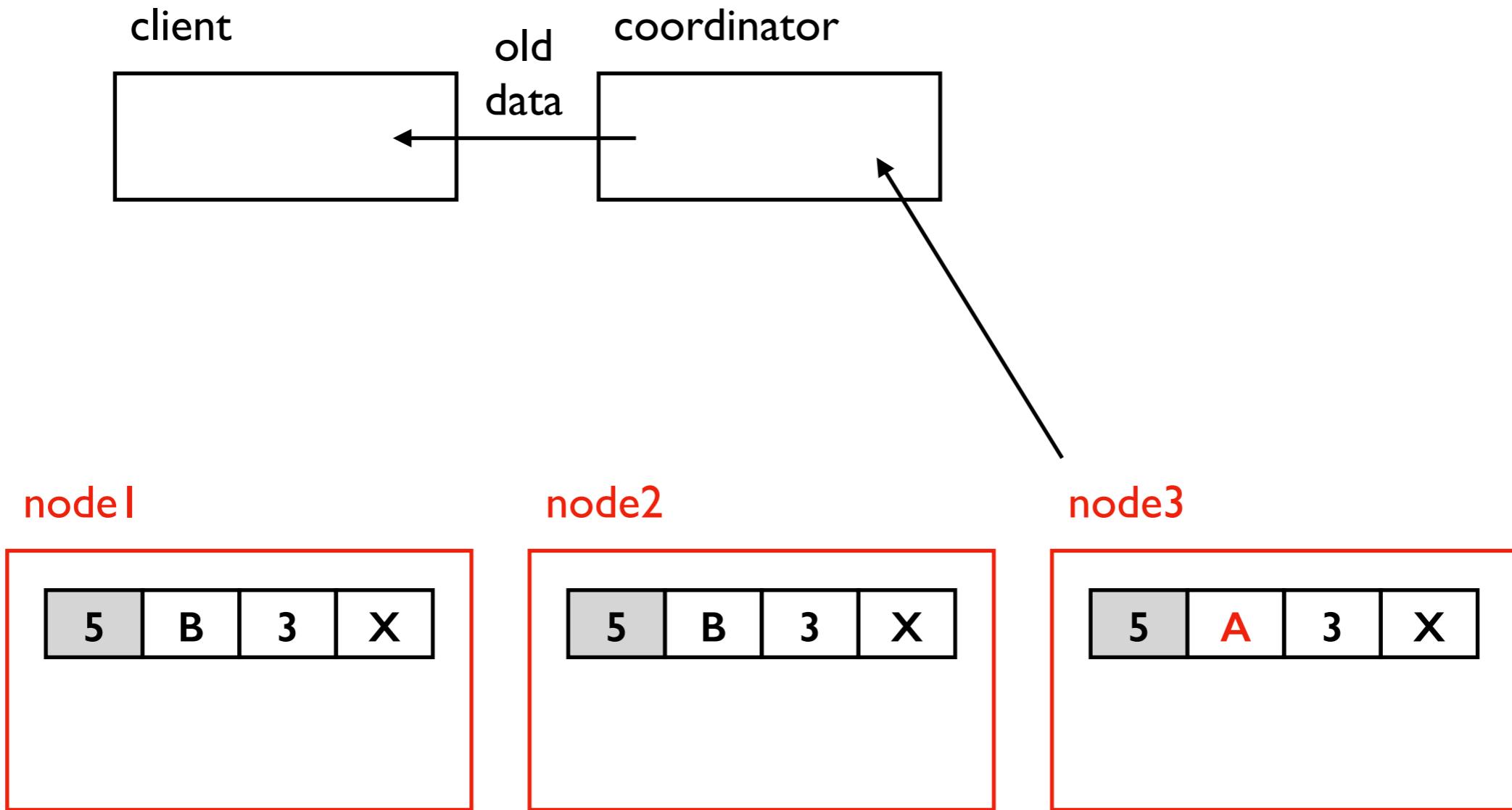
Configurable. W=2 lets coordinator ack now, and data is fairly safe.

# Cassandra Reads



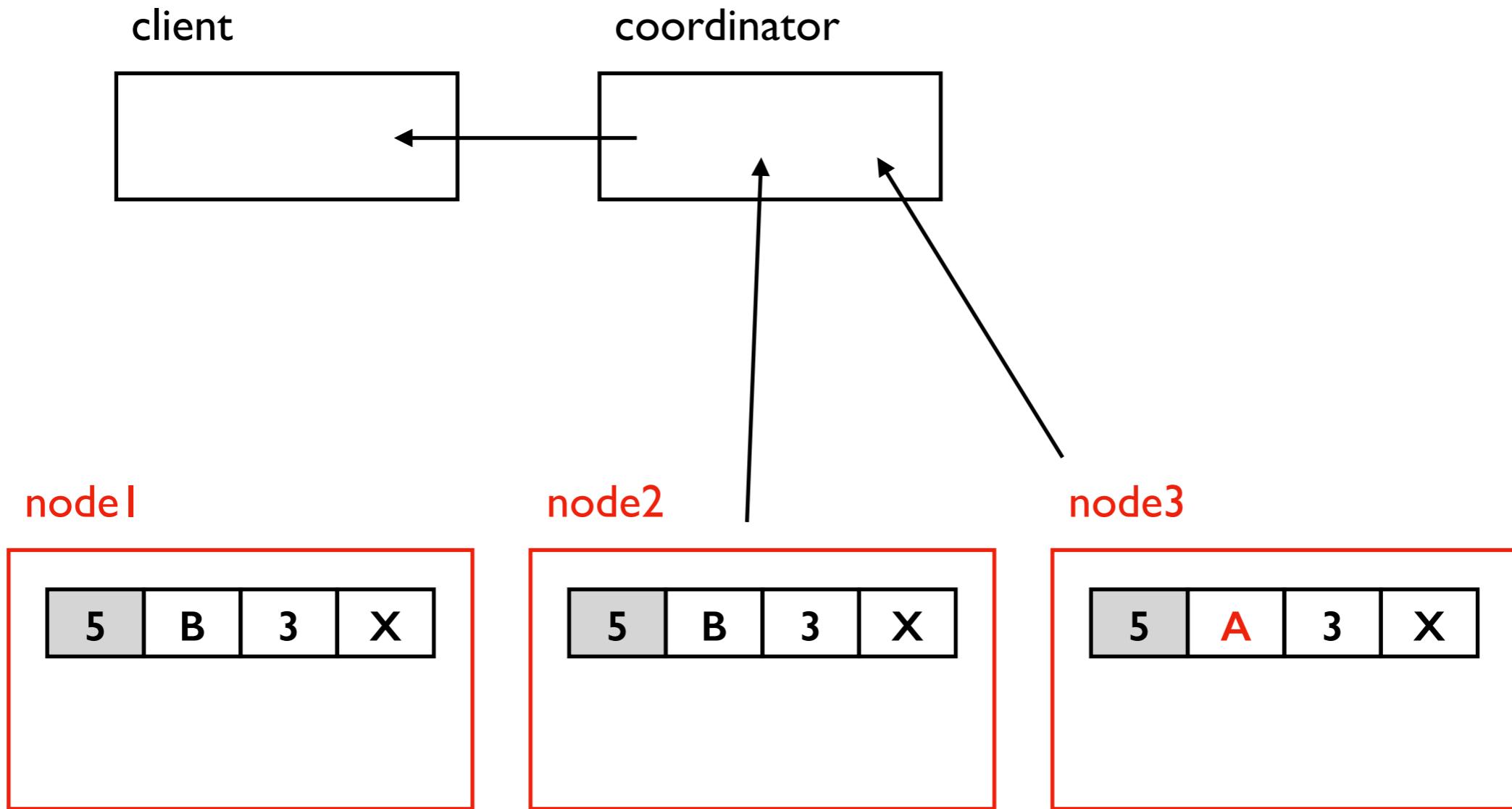
HDFS reads go to one replica. What if Cassandra tries that?

# Cassandra Reads



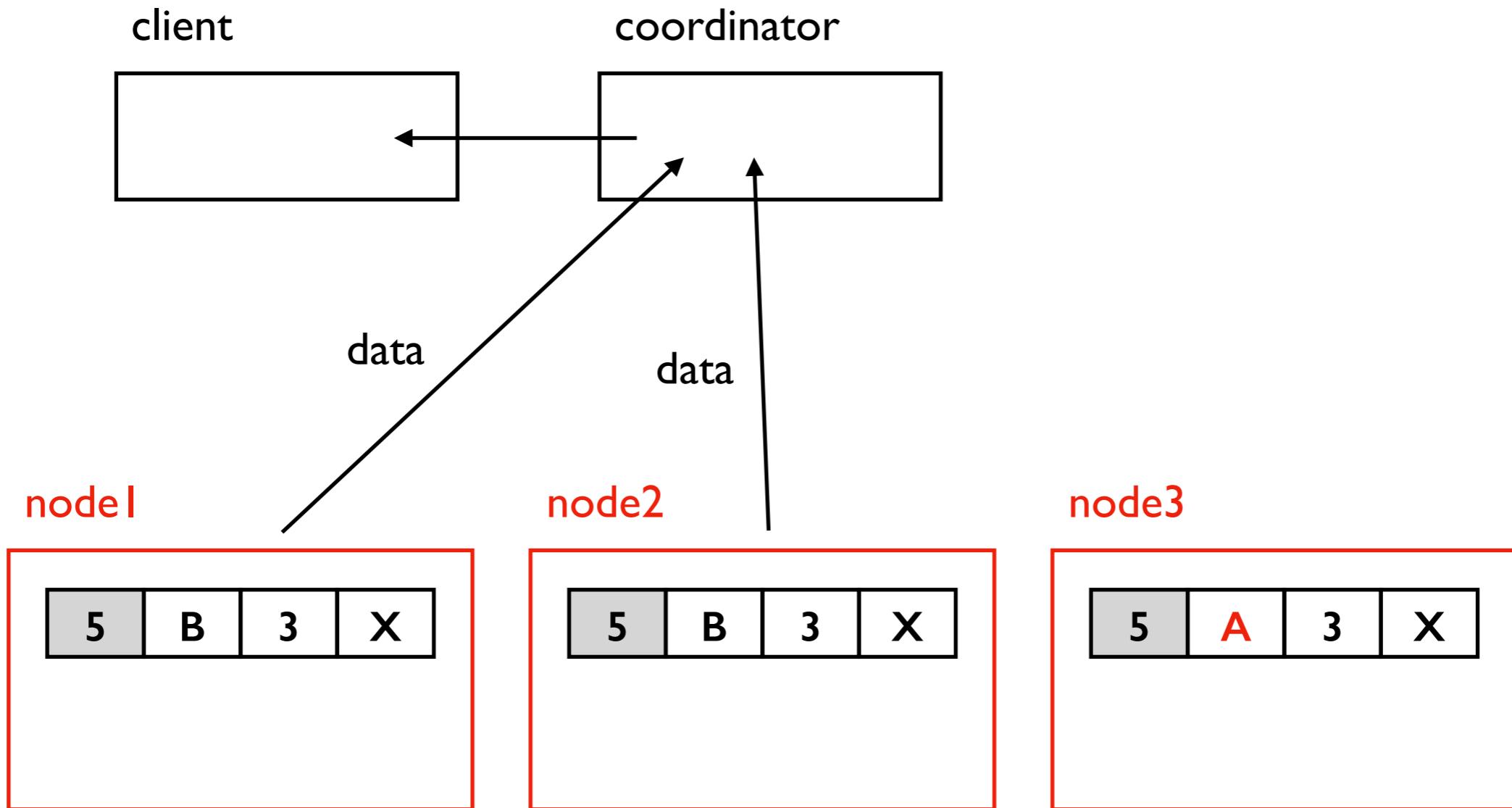
HDFS reads go to one replica. What if Cassandra tries that?

# Cassandra Reads



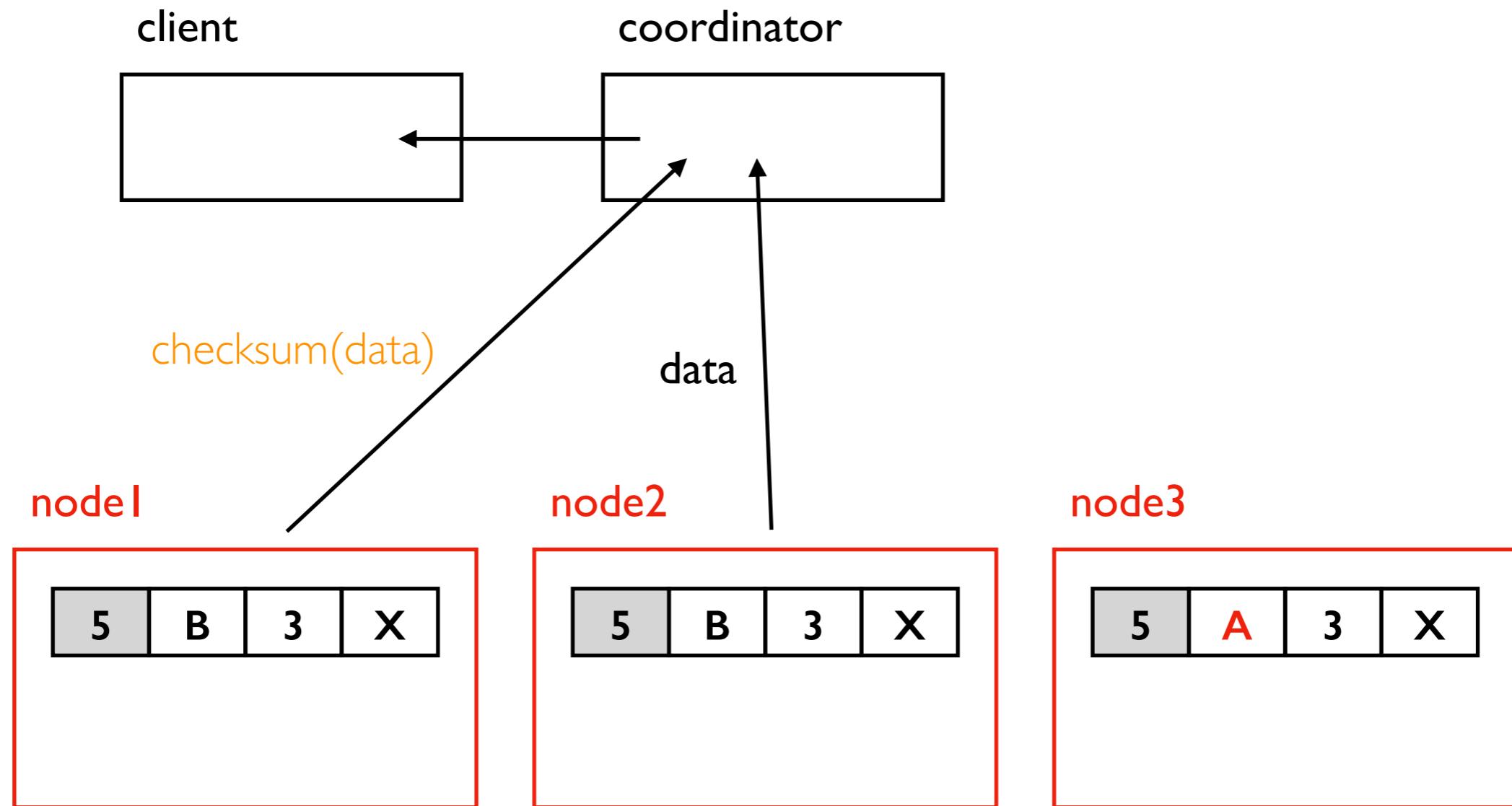
Read from R replicas (configurable). Here R=2.  
Hopefully at least one of the replicas has new data.

# Cassandra Reads



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

# Cassandra Reads



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

Improvement: read one copy, and only request checksum from others.

A **checksum** (like md5) is a hash function where collisions are extremely rare and hard to find.

# When $R+W > RF$

$RF=3$

---

$W=2$

---

$R=2$

---

node1

5	B	3	X
---	---	---	---

node2

5	B	3	X
---	---	---	---

node3

5	A	3	X
---	---	---	---

When  $R+W > RF$ , the replicas read+written will **overlap**. If your write is acked, you'll see it when you read later.

There are some caveats (related to ring membership and something called "hinted handoff") not covered in 544.

# Tuning R and W

Say RF=3

W=3, R=1

- reads are highly available and fast -- only need one replica to respond before we can get back to the client!
- writes will not succeed (from the client's perspective) if even one node is down. But the data may still get recorded on some nodes.

W=1, R=3

- writes are highly available and fast -- only need one replica to respond before we can get back to the client!
- reads will not return data when even one node is down.
- risky: if the one node that took the write fails permanently, we'll lose committed data

W=2, R=2

- relatively balanced approach

W=1, R=1

- speed+availability more important than correct data

# Worksheet

# Outline

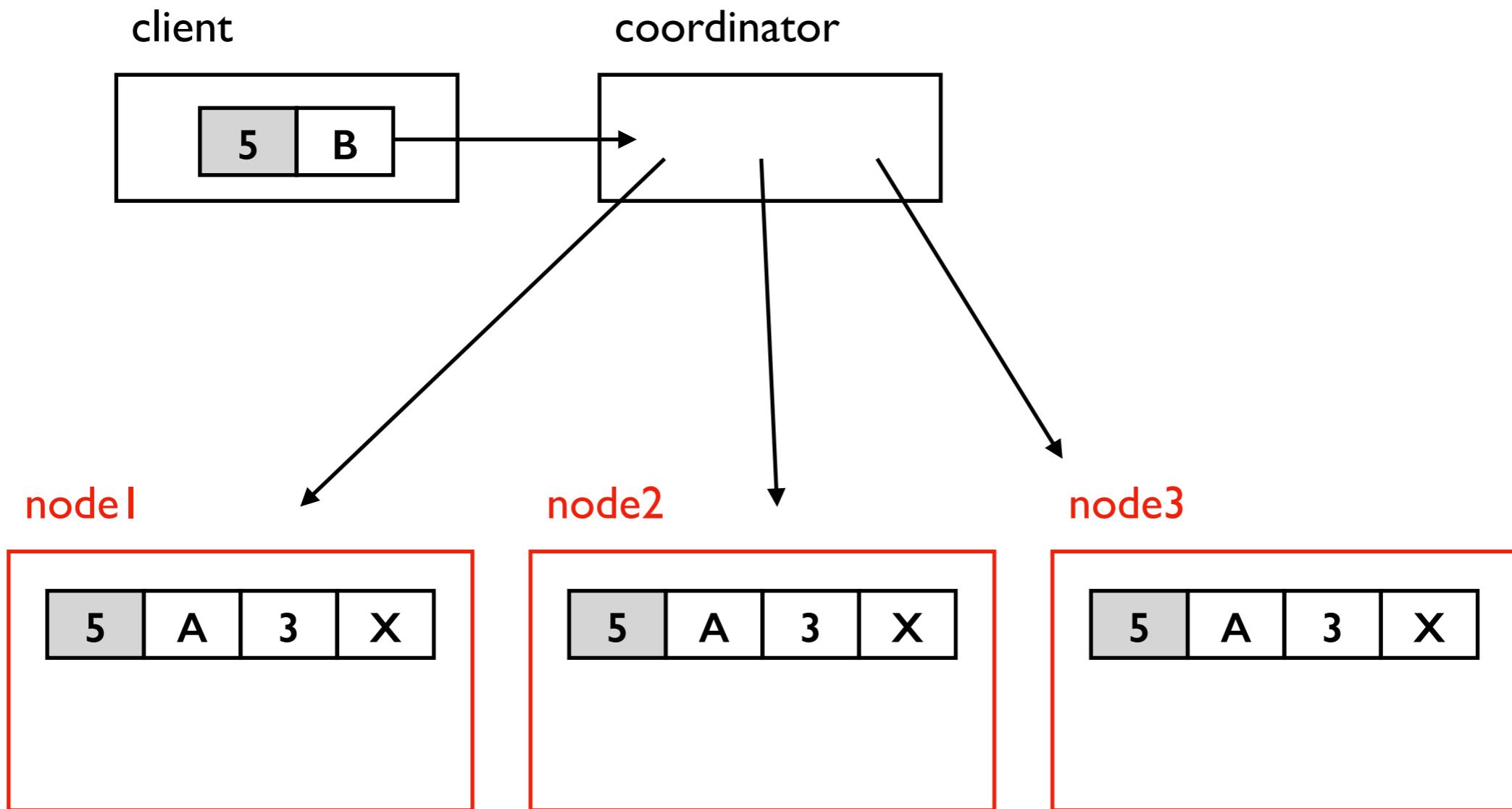
Replication

Quorum Reads/Writes

**Conflict Resolution**

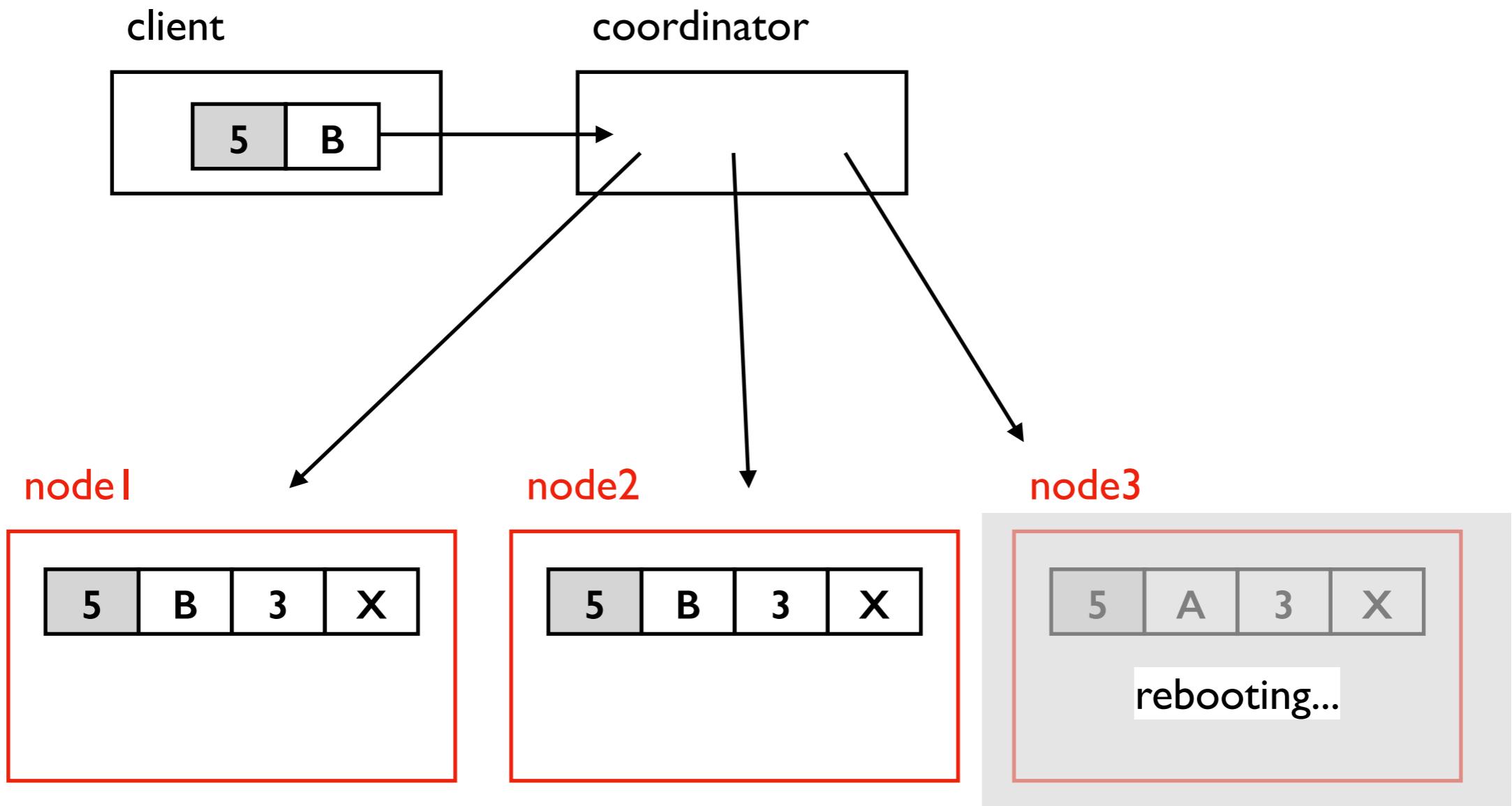
Cassandra Demos

# Getting Conflicting Versions



Let RF=3, R=2, W=2

# Getting Conflicting Versions



Let RF=3, R=2, W=2

# Getting Conflicting Versions

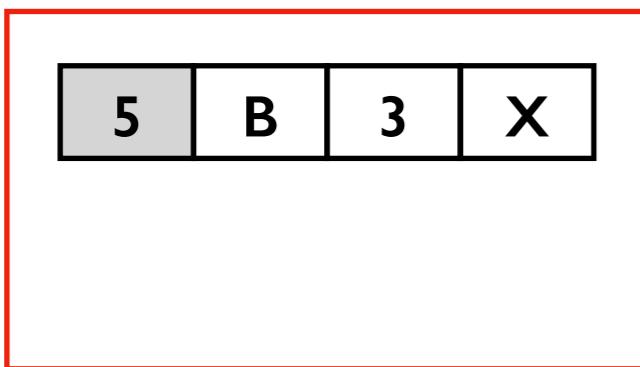
# client



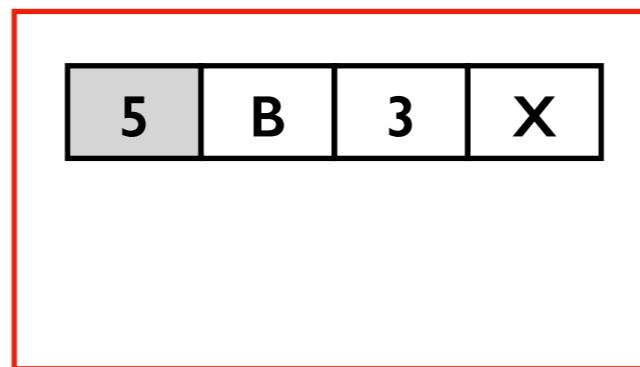
## coordinator



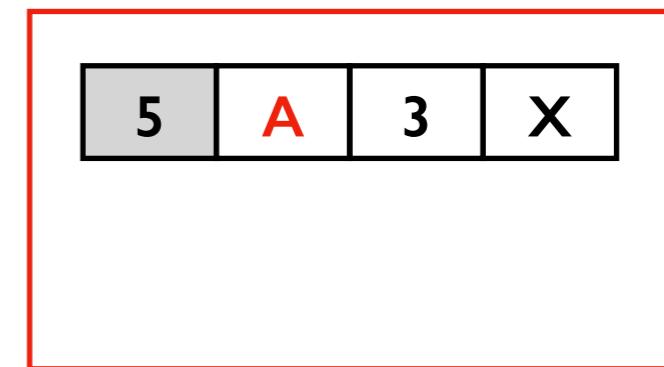
node |



node2

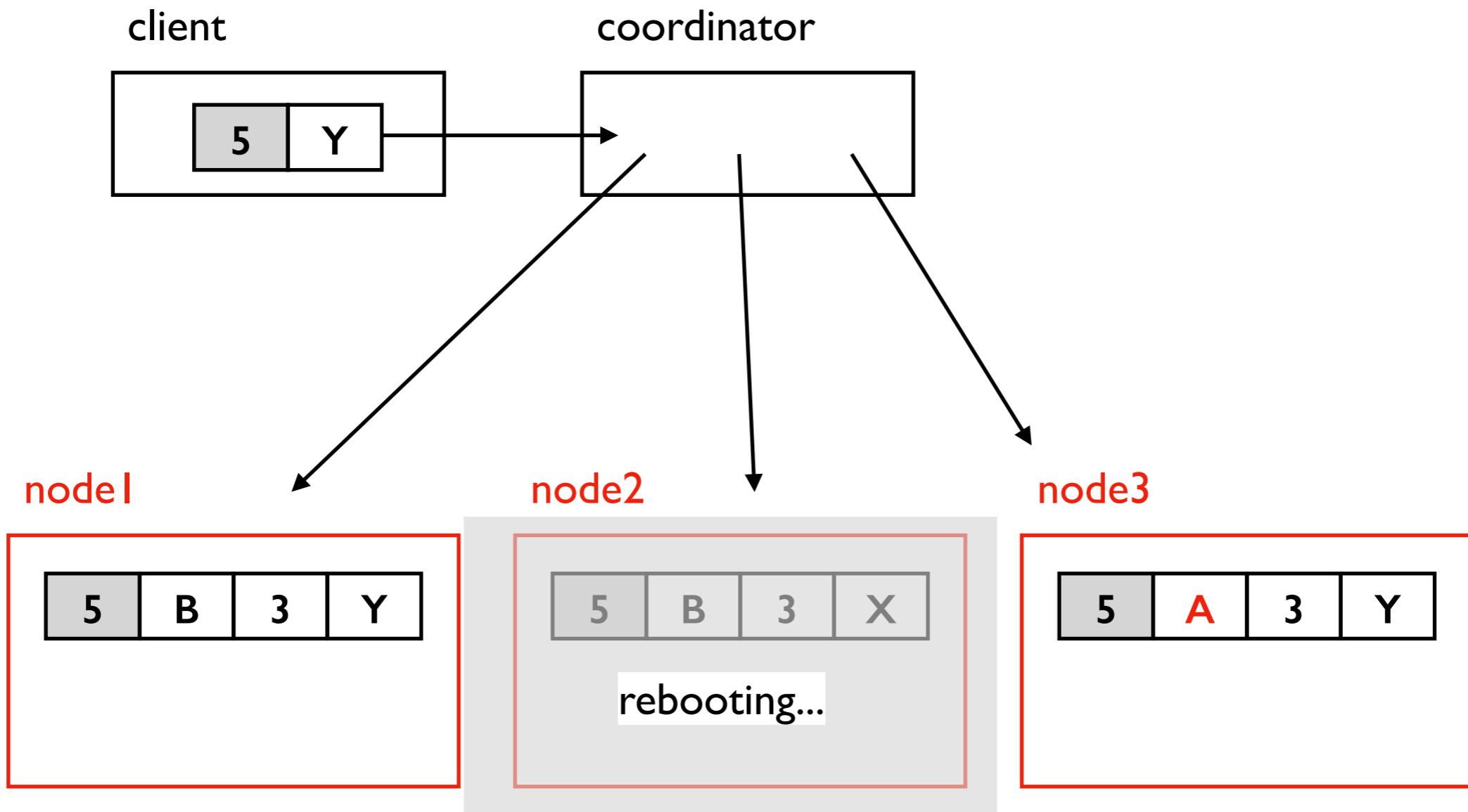


node3



Let RF=3, R=2, W=2

# Getting Conflicting Versions



Let RF=3, R=2, W=2

# Getting Conflicting Versions

# client



## coordinator



node |

5	B	3	Y
---	---	---	---

node2

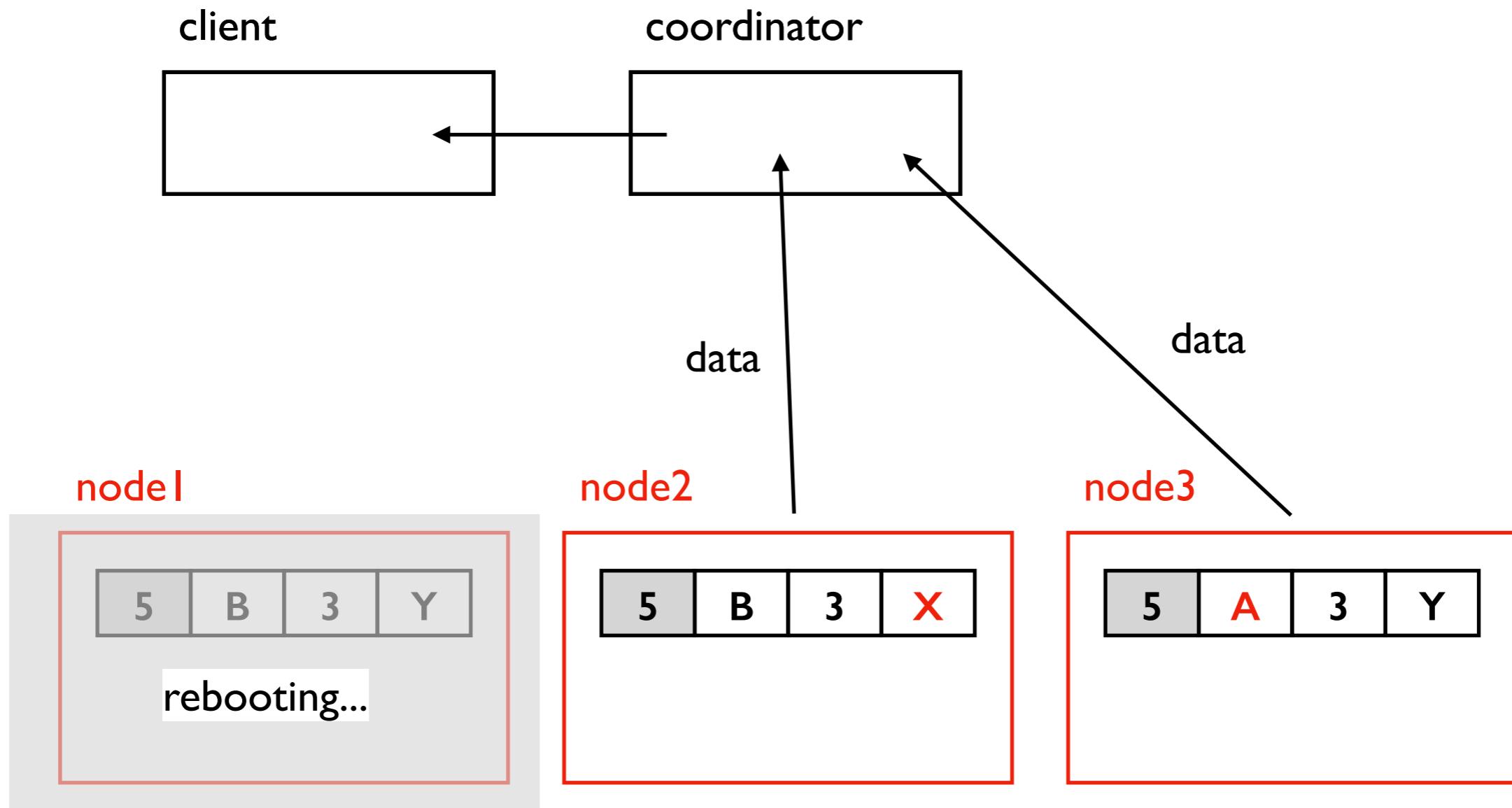
5	B	3	X
---	---	---	---

node3

5	A	3	Y
---	---	---	---

Let RF=3, R=2, W=2

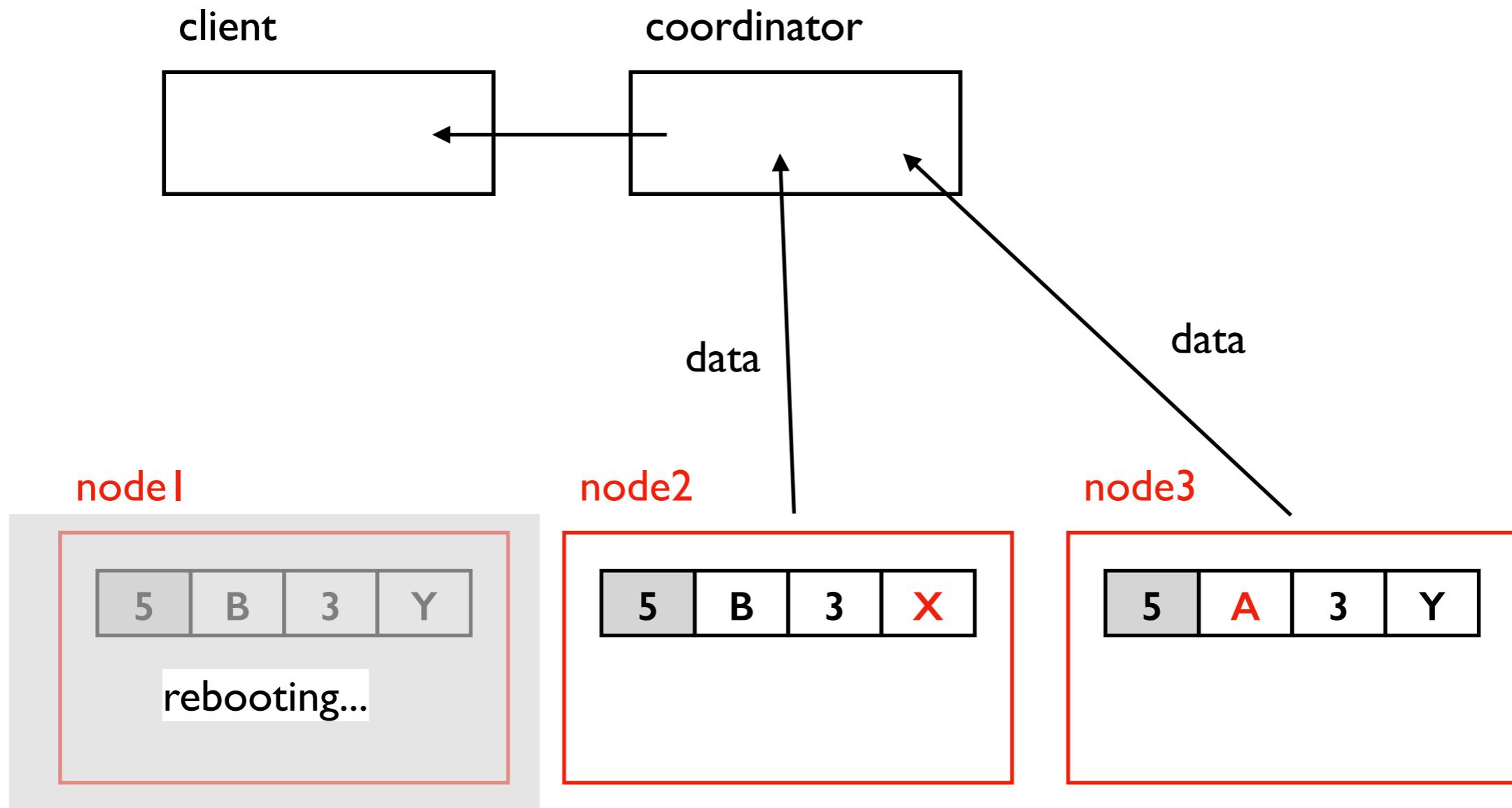
# Getting Conflicting Versions



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

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

# Getting Conflicting Versions

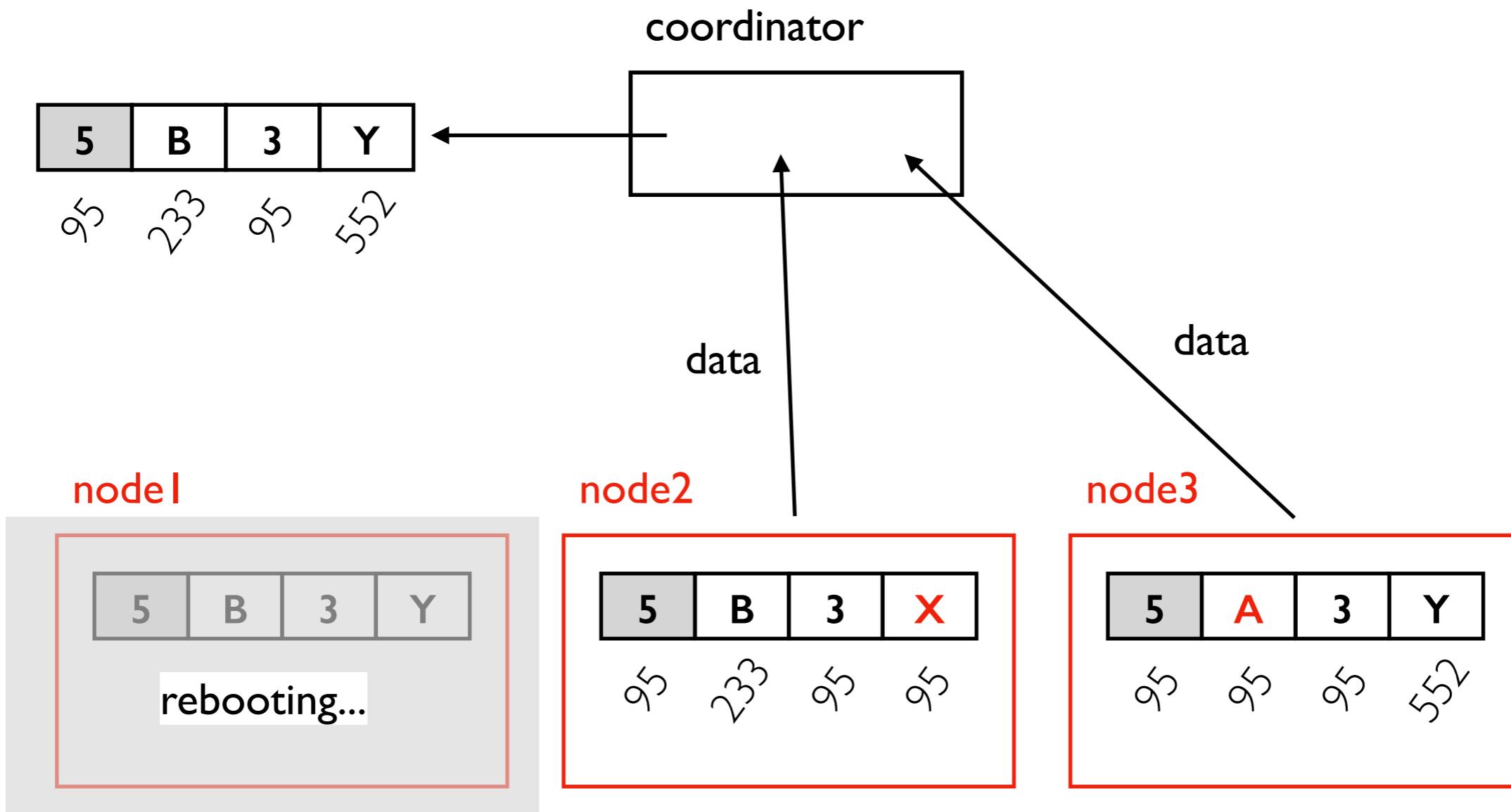


## Approaches:

- send all version back to the client, which will need specialized conflict resolution code
- automatically combine them into a new row, and write that (if possible) to all replicas

Dynamo supports both. Cassandra uses second approach.

# Timestamps



**Every cell of every table has a timestamp:**

- approximate (since clocks of nodes in a cluster are never perfectly in sync)
- policy is LWW (last writer wins), meaning prefer newer data
- Cassandra lets you query the timestamp of each cell

# Outline

Replication

Quorum Reads/Writes

Conflict Resolution

**Cassandra Demos**