Distributed Systems – Project 2025

Mars distributed storage system

stefano.genetti@unitn.it

erik.nielsen@unitn.it

Mission

- You have been enrolled by the European Space Agency (ESA).
- TASK: deployment of a distributed peer-to-peer network of data collection stations on the surface of Mars.
- The Martian extreme environment introduces considerable challenges! Stations can crash, disconnect, reboot or join the network unpredictably.

Project overview

- You will implement a key-value store (database)
- Every data item is identified by a unique key
- Main operations:

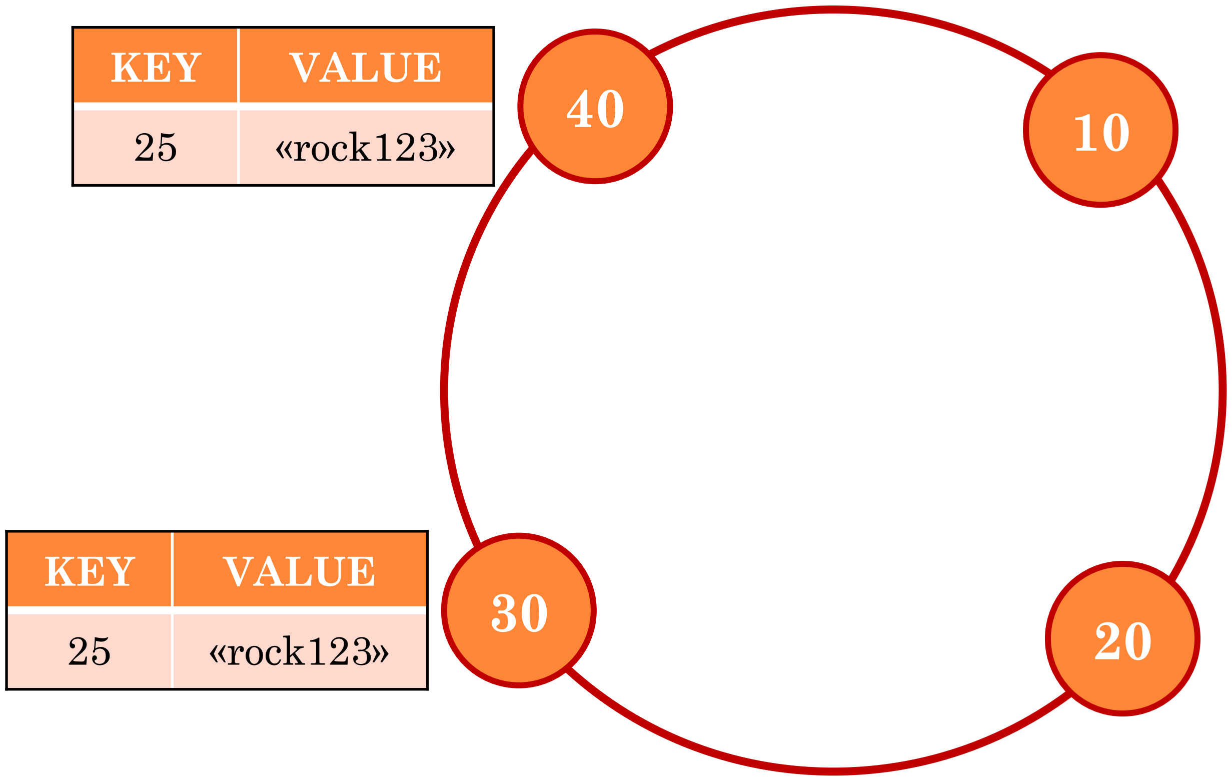
  update(key, value)

  get(key) → value
- Replication: several nodes store the same item. For reliability and accessibility.
- Data partitioning: not all nodes store all items. For load balancing.

KEY	VALUE
C	7
E	129
…	…
G	16

Network logical topology

- Both the storage nodes and data items have associated keys that form a circular space (**ring**).
- We will use integers for keys, strings for values.

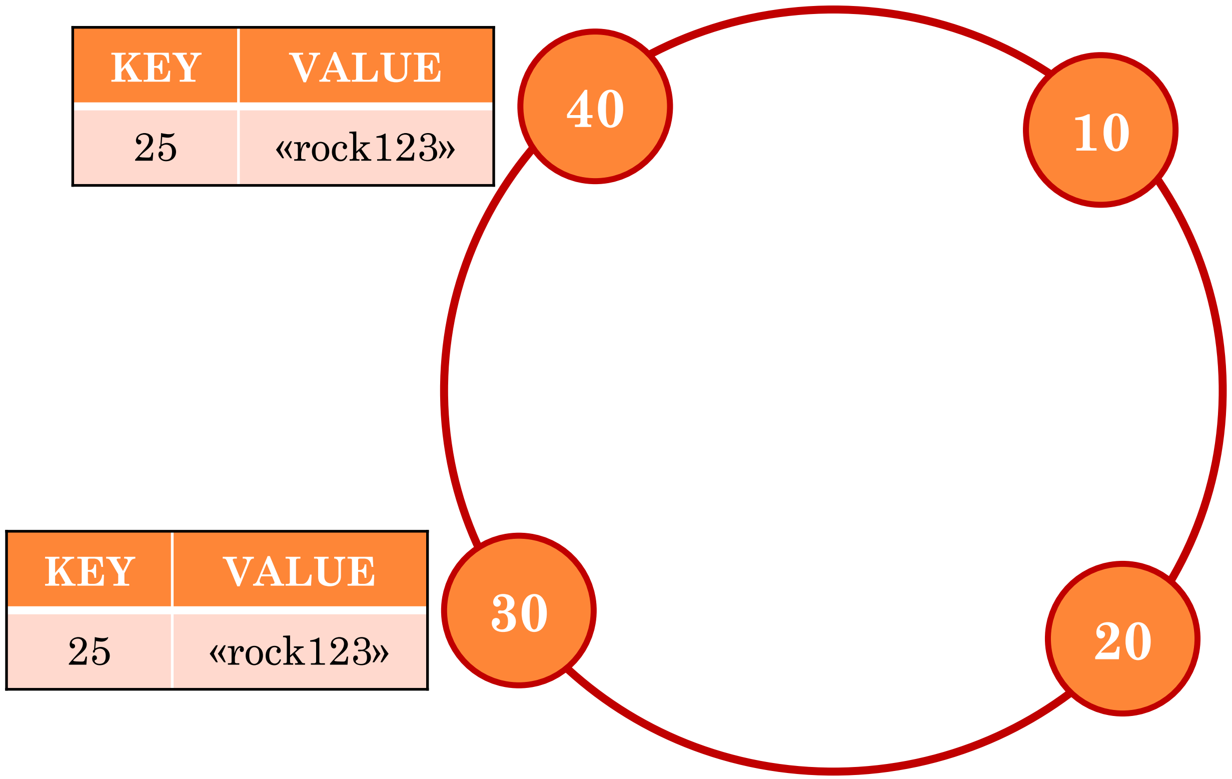


KEY	VALUE
25	«rock123»

KEY	VALUE
25	«rock123»

Network logical topology

- A data item with key K is stored by the N nearest clockwise nodes.
- For example, if N=2, a data item with key 25 will be stored at nodes 30 and 40.

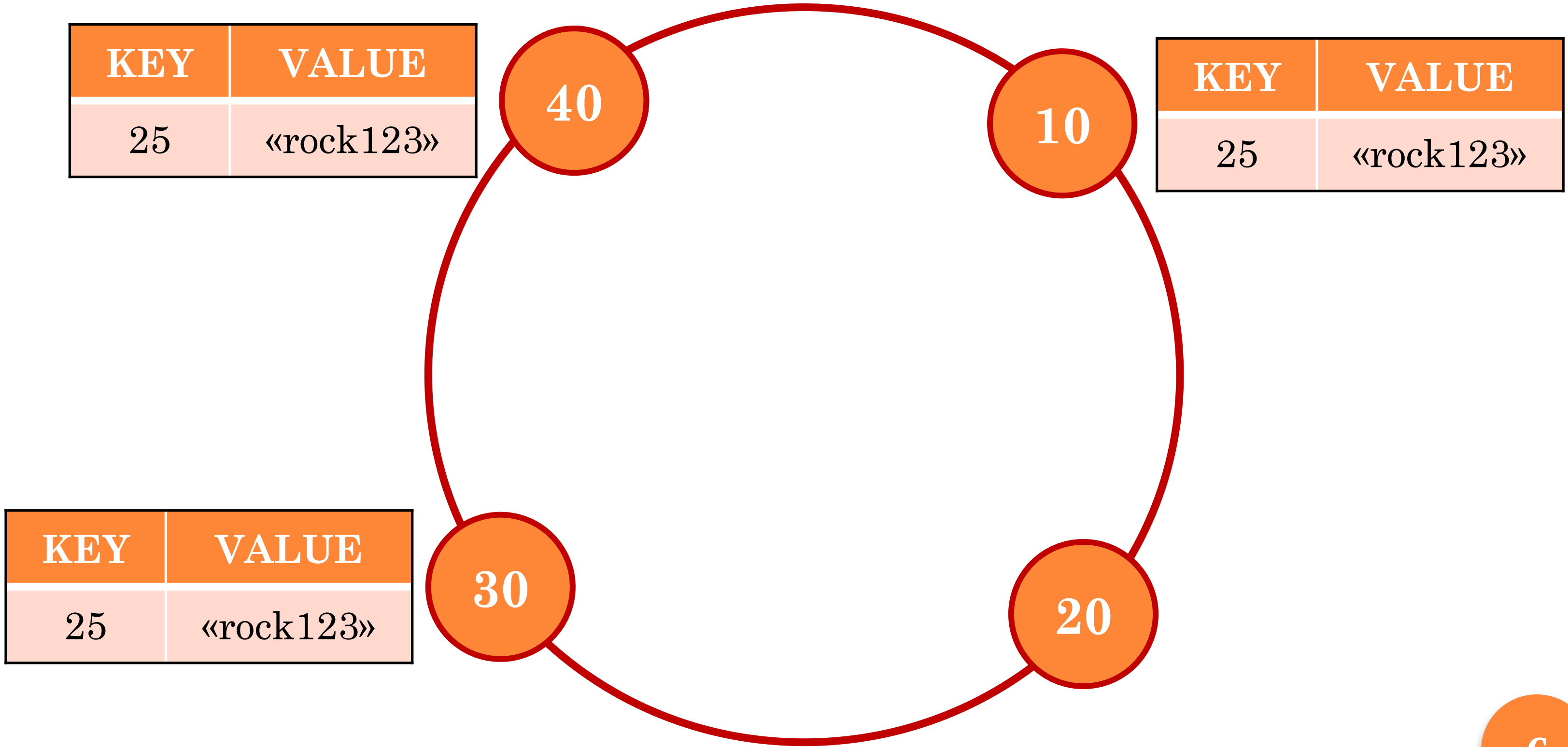


KEY	VALUE
25	«rock123»

KEY	VALUE
25	«rock123»

NETWORK LOGICAL TOPOLOGY

- A data item with key K is stored by the N nearest clockwise nodes.
- For example, if N=3, a data item with key 25 will be stored at nodes 30, 40, 10. If we need to go beyond the maximum node key, we wrap around.



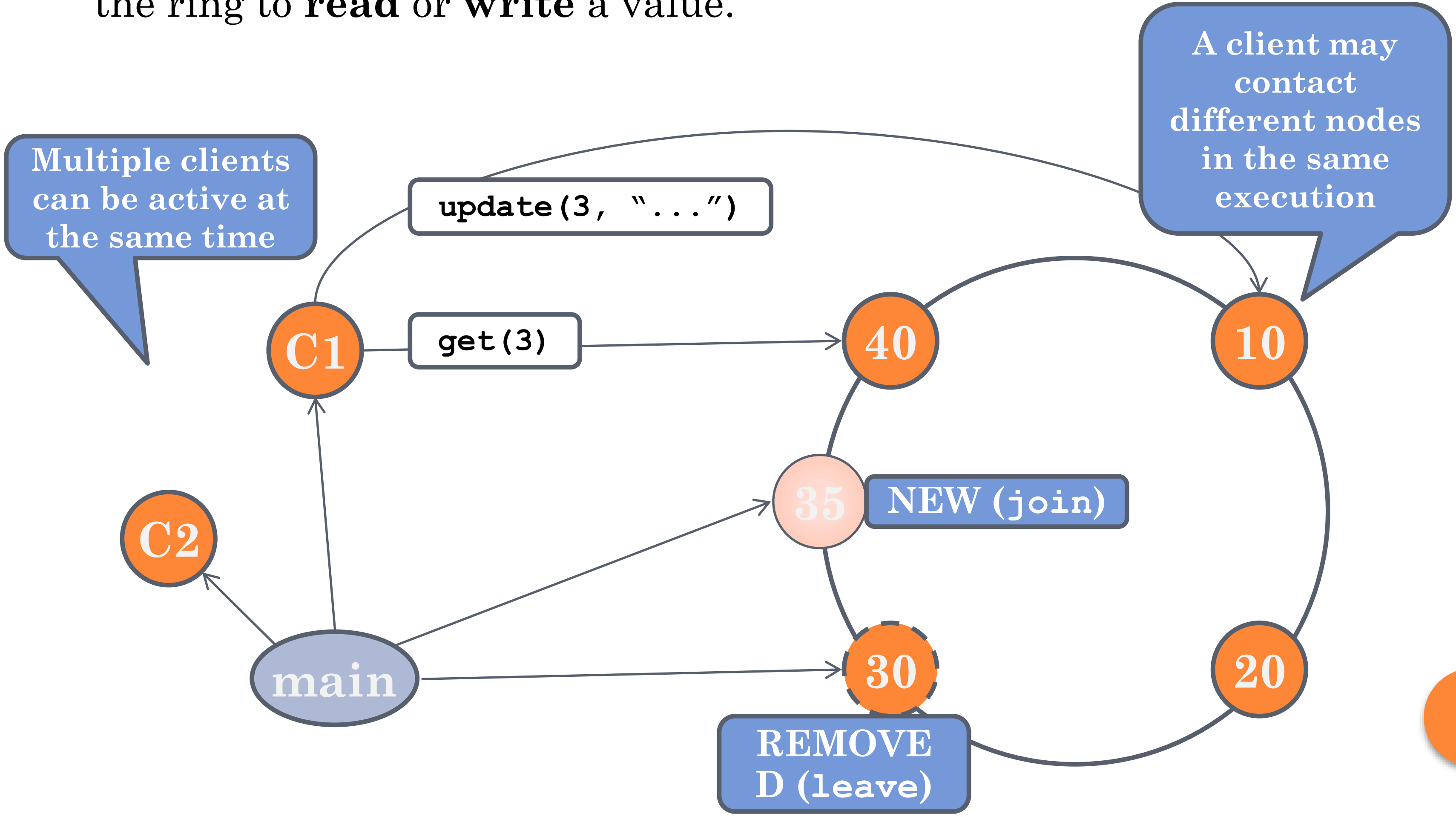
KEY	VALUE
25	«rock123»

KEY	VALUE
25	«rock123»

KEY	VALUE
25	«rock123»

Types of actors and supported operations

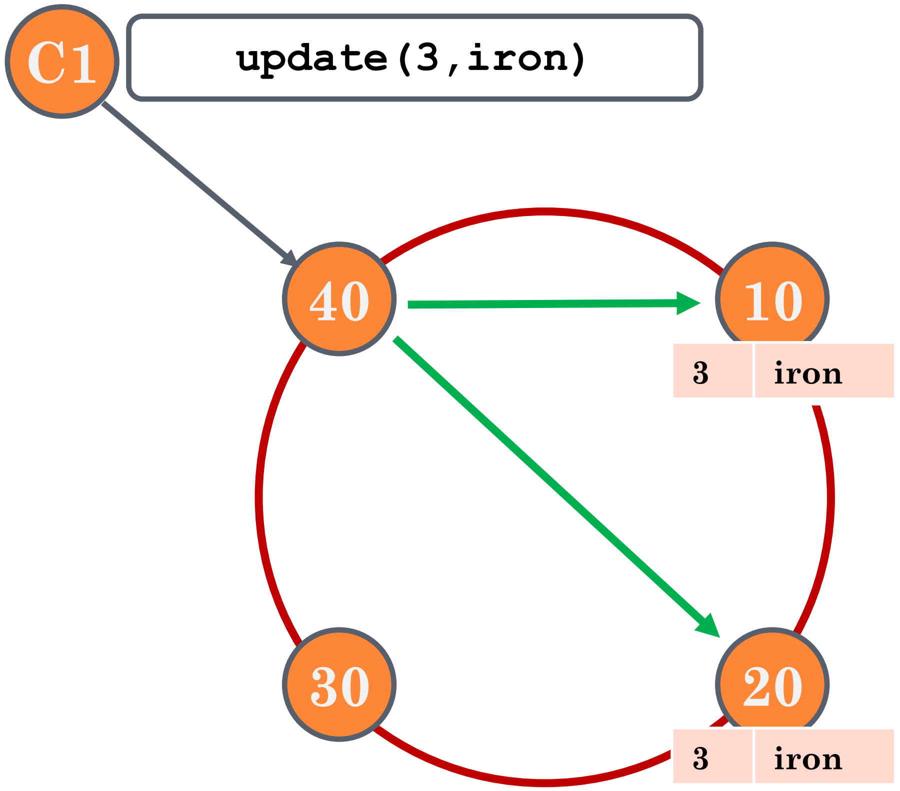
- You must support the creation of clients, new storage nodes (**join**), and storage node removal (**leave**).
- A different type of actor, the client should be able to contact any node in the ring to **read** or **write** a value.



Multiple clients can be active at the same time
A client may contact different nodes in the same execution
update(3, "...")
get(3)
C1
C2
main
40
10
35
NEW (join)
30
20
REMOVED (leave)

Request coordinator

- Clients may contact any node to read/write data with any key: the chosen node is the coordinator



Node 40 is the *coordinator* for C1's request

Nodes know all their peers and, therefore, can compute who is responsible for the key and pass the request

Replication

- The same data item is stored by N nodes
- **When reading an item**, the coordinator requests data from all N nodes, but answers to the client as soon as a **read quorum R** $\leq$ N of them reply
- **When writing**, the coordinator tries to contact all N nodes, but completes the write as soon as a **write quorum W** $\leq$ N of them reply

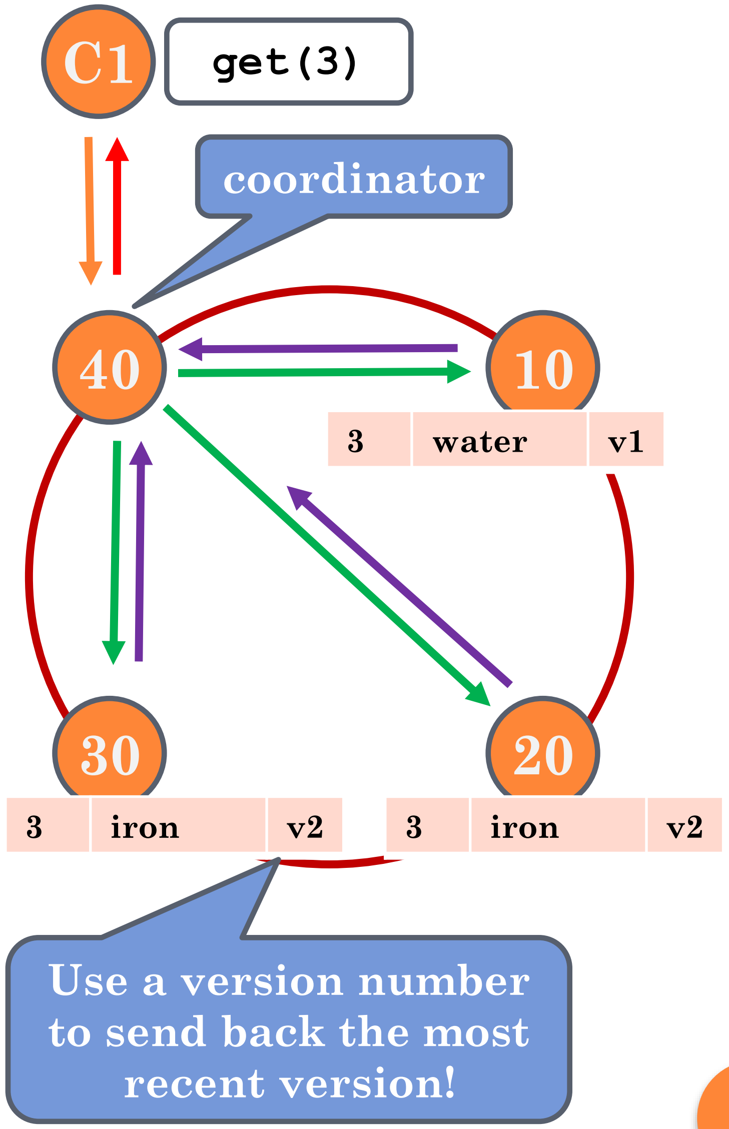
In your project, set R and W correctly according to N, and implement read and write operations that provide sequential consistency

The result is the same as if the operations by all processes were executed in some sequential order, and the operations by each process appear in program order

Read operation

- Let N=3, R=2

1. C1 contacts node 40
2. 40 contacts N replicas
3. Other nodes reply
4. As soon as R replies arrive, 40 sends the item to C1



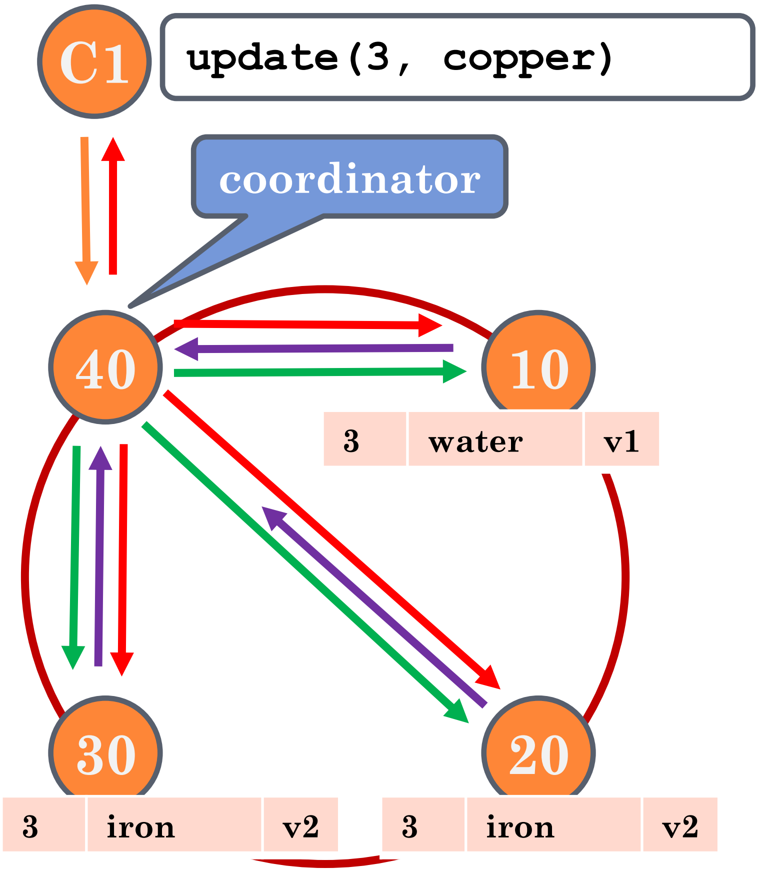
Write operation

- Let N=3, W=2

1. C1 contacts node 40
2. 40 contacts N replicas
3. Other nodes reply
4. As soon as W replies arrive, 40 sends the item to C1 and updates the item at all N nodes:

3	copper	v3

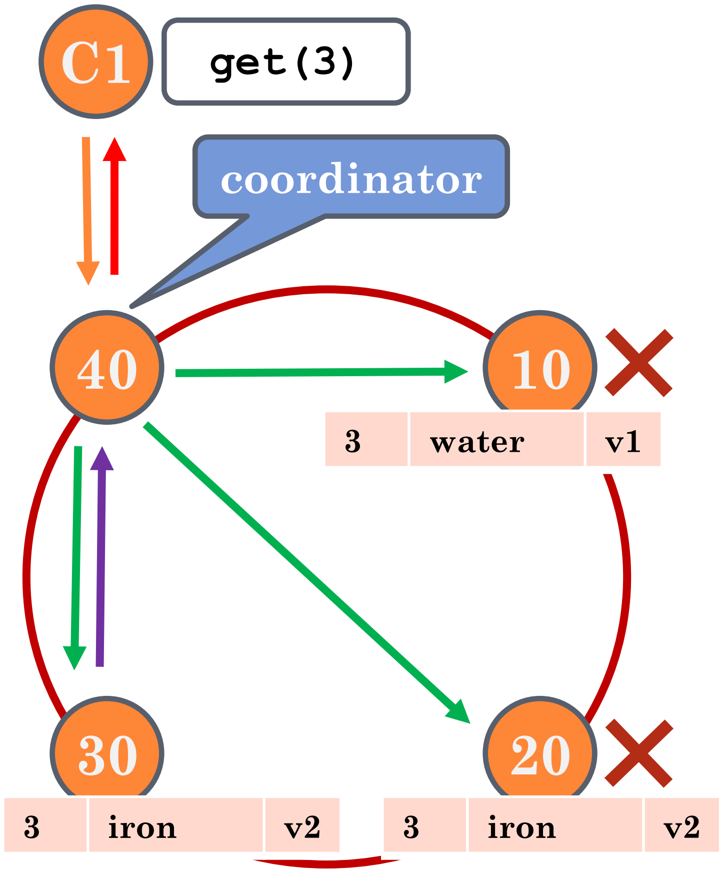
Update the version!



No quorum?

- Let N=3, R=2, T=1s

1. C1 contacts node 40
2. 40 contacts N replicas
3. Not enough nodes reply within 1 second due to crashes…
4. Timeout *T* expired.
   40 sends back an error message to the client



Sequential consistency: Hints

- What if a write is requested while processing of another write on the same key?

Avoid write/write conflicts or you may have different values for the same data version!

- Once a client has observed some value, it can never observe a value prior to it…
- …so, what if a read overlaps with a write operation? Can it happen that the new value is read by a client while some nodes have yet to receive the update? If the client reads the new value and then reads again, can you ensure the old value is not the one returned?

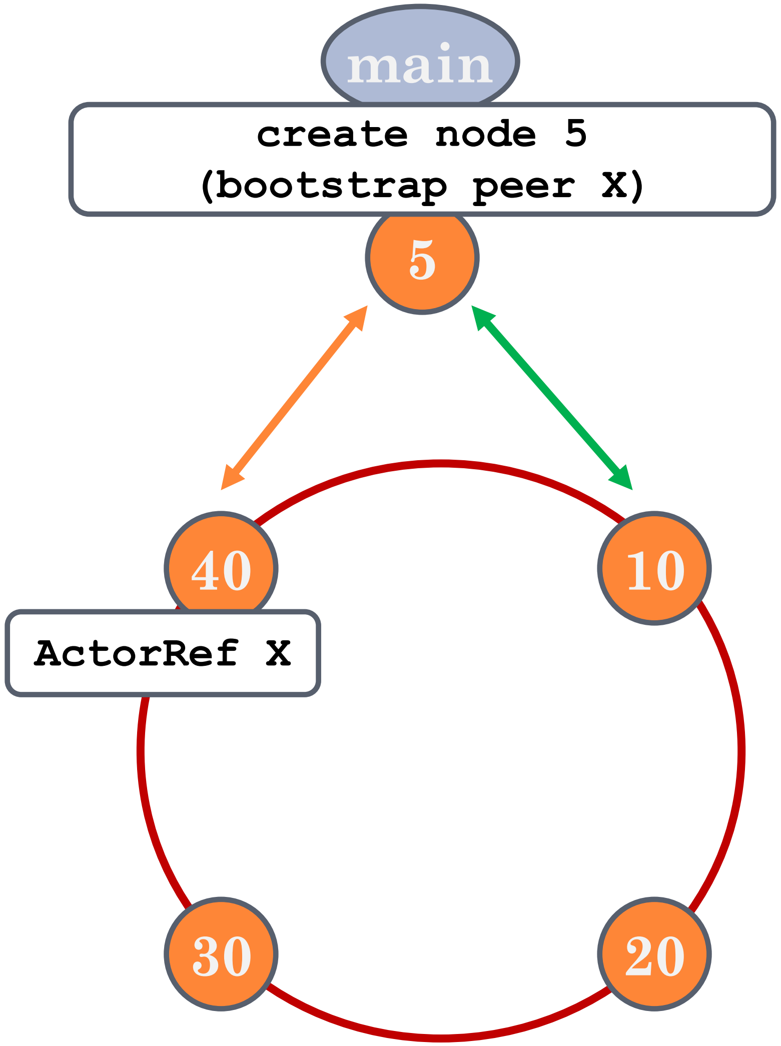
Like with writes, reject reads that may break sequential consistency!

Item repartitioning

- When a node joins or leaves, the system should move data items around accordingly
- Crashes do not count as leaving: no need to implement a failure detection mechanism and move items if a node cannot be reached

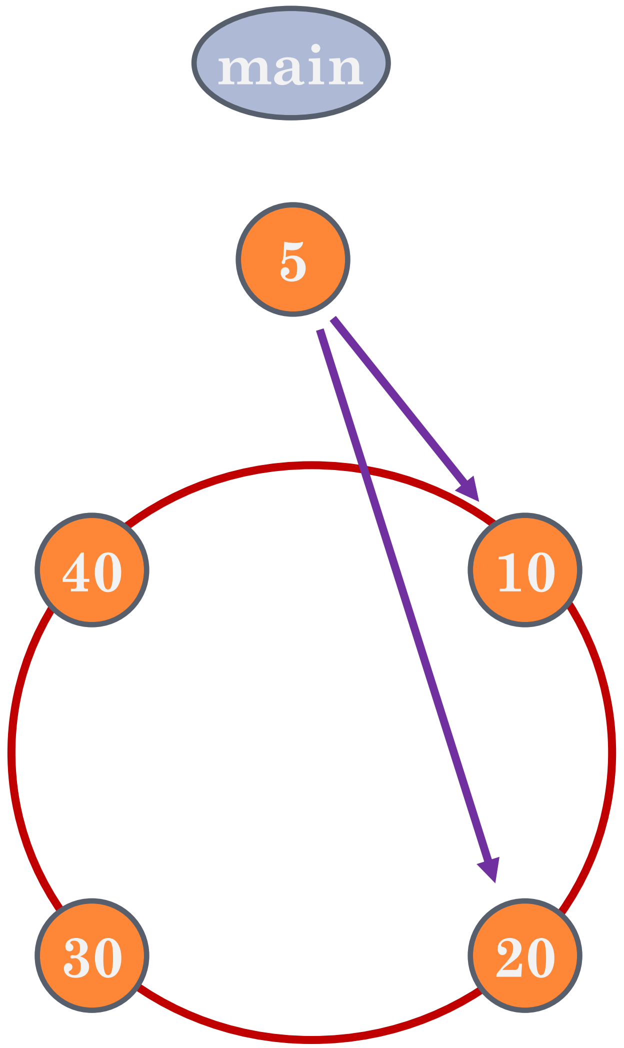
Joining operation

1. Joining node 5 can contact any node to request the list of nodes in the system
2. Based on the list of nodes, node 5 knows 10 holds all items node 5 is responsible for, and asks node 10 for those items



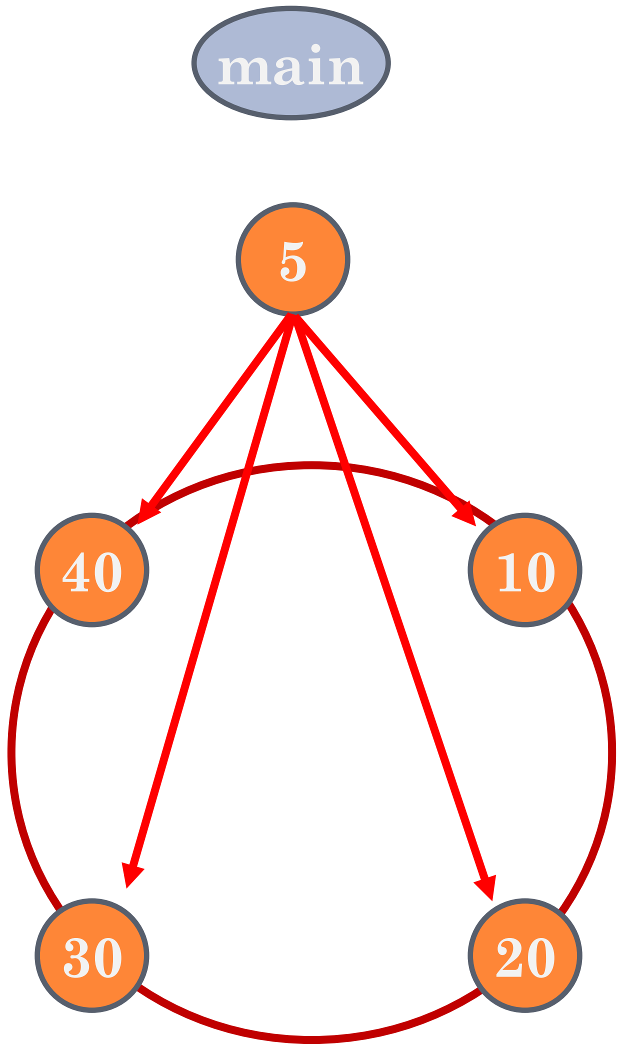
Joining operation

3. Node 5 reads the items it is responsible for from the current replicas to ensure it finds the most updated version

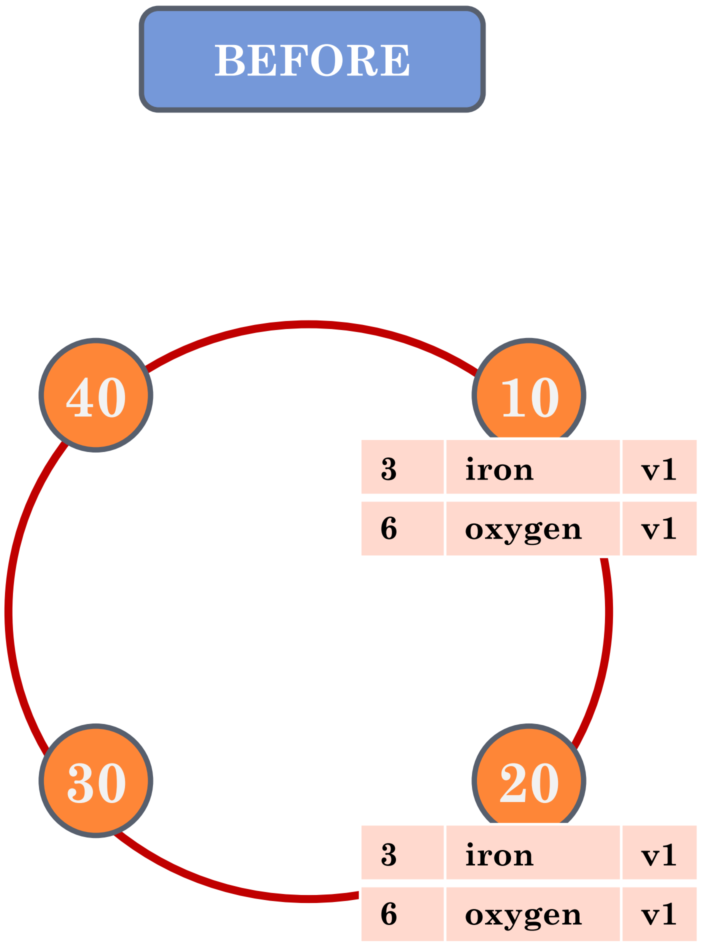


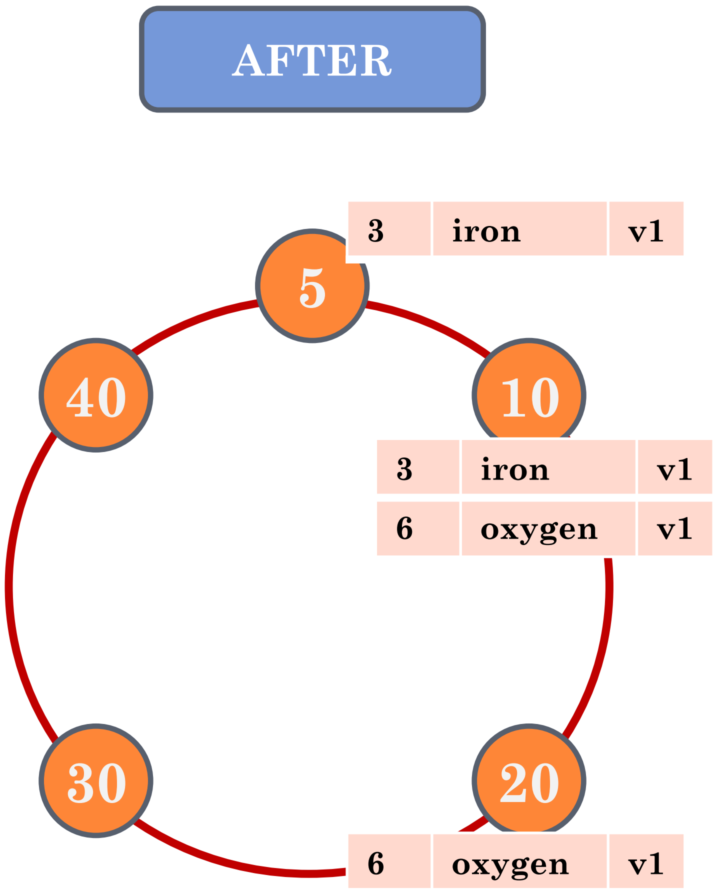
Joining operation

4. Node 5 announces itself to the whole system, and other nodes remove items they are not responsible for anymore

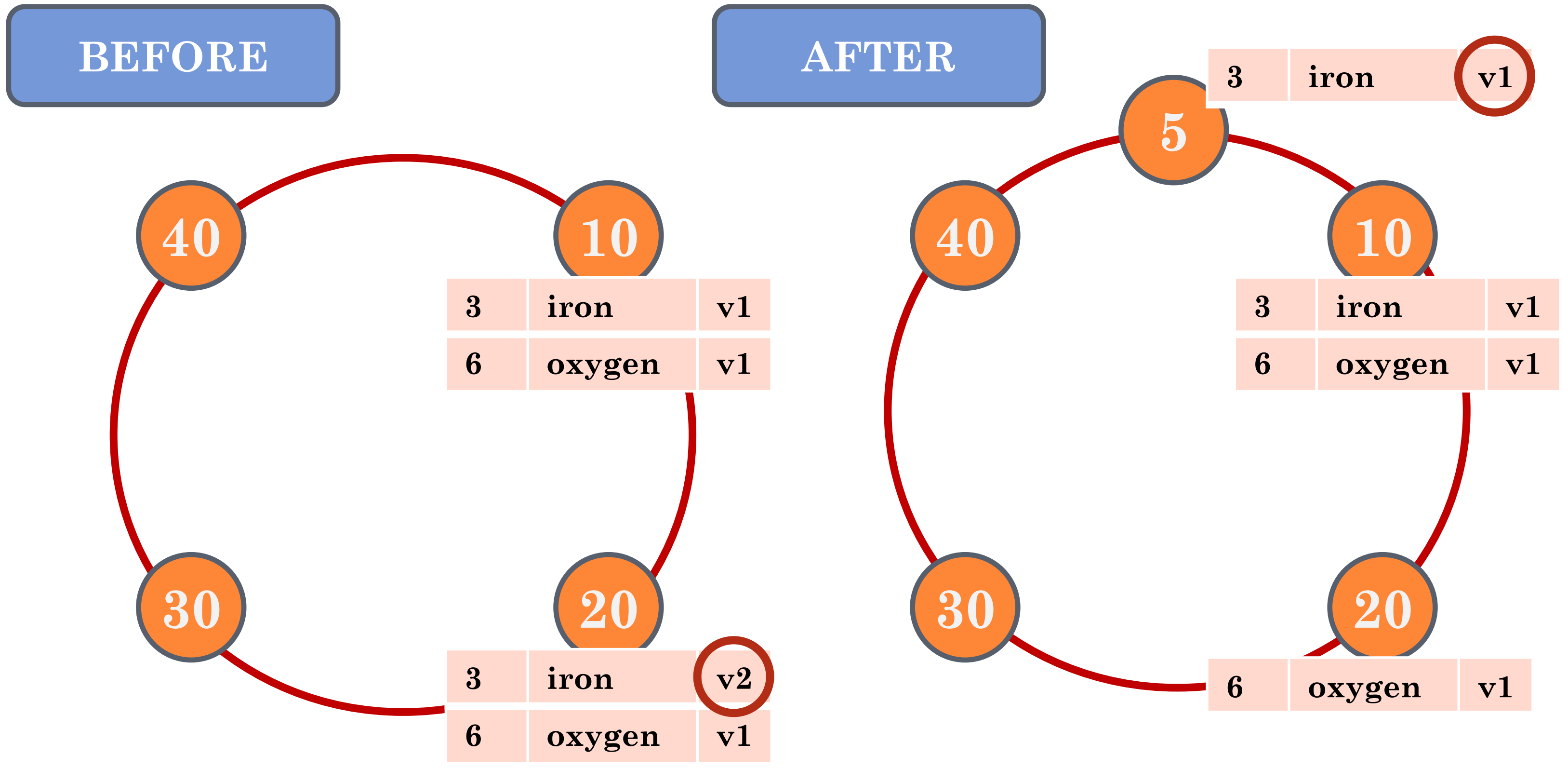


Joining operation (example N=2)





JOINING OPERATION (WHY READING?)

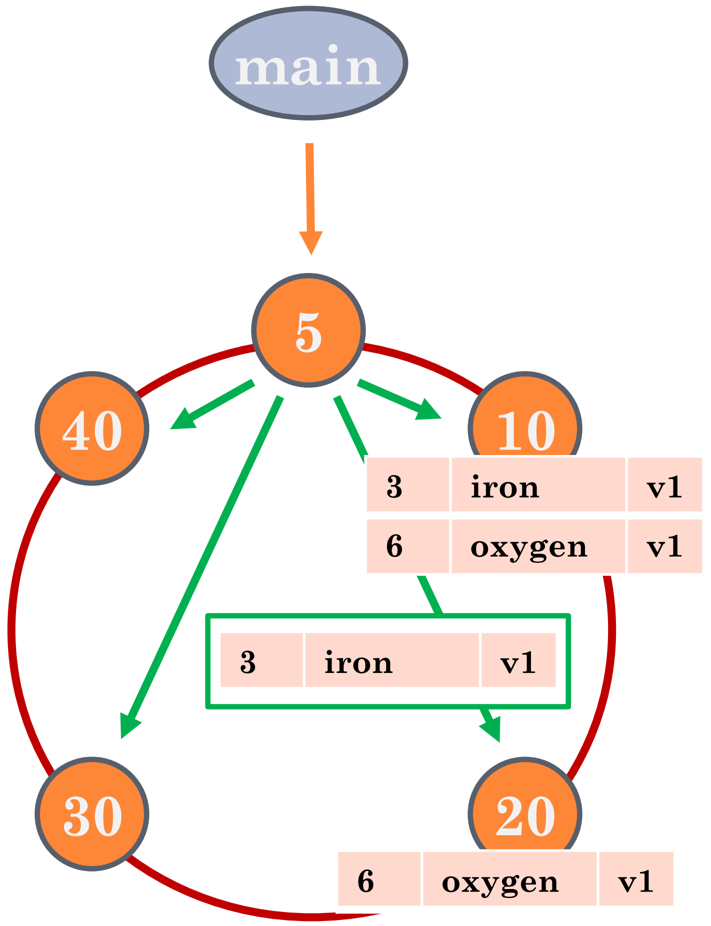


Node 5 may retrieve an item k from its next neighbor in the ring, 10. It is possible that item k hold by 10 has an old version. If 5 just announces itself as a new node, another replica (20) will drop item k.
If item k at node 20 had the most recent version, the procedure ends up replacing a new item with an old item, breaking quorum assumptions.

LEAVING OPERATION

1. The main may request a node to leave
2. The node announces to every other node that it is leaving, and passes its data items to any node that becomes responsible for them

Do not send all data items to all nodes, send only if needed



RECOVERY OPERATION

Storage nodes may simulate a "crash", but recover when requested

1. The recovering node asks the current list of nodes to the node specified in the recovery request
2. Upon discovering which nodes are in the system…

   it forgets the items it is no longer responsible for,

   due to new nodes joining the storage network

   and requests the items it became responsible for

   due to nodes leaving the storage network

MAIN ASSUMPTIONS

- A node may serve multiple clients, and the network may be requested to operate on the same item keys in parallel

Hint: requests may fail if they potentially compromise sequential consistency

- Nodes join, leave, crash or recover one at a time when the network is idle. Operations might resume while one or more nodes are still in crashed state.
- Replication parameters are specified at compile time

Refer to the description document for more details

Project report

- Structure of the project
- Main design choices
  - How do you keep track of ongoing operations?
  - What are the main messages used in the system?
  - How do you simulate crash and recovery?
- Discussion of the implementation
  - Does it provide sequential consistency? How?
  - Do your strategies for read/write and join/leave use the minimum amount of messages? For join and leave, do you ensure data is only sent to nodes responsible for it?
  - Are there any additional assumption about the system that you needed to make?

Grading

- The whole feature set is worth **6 points**
- A reduced implementation without replication, item versions and support for node recovery is worth **3 points**
- Work in pairs! (But grades are individual!)