CAP Theorem

Big Data Systems

Dr. Rubi Boim

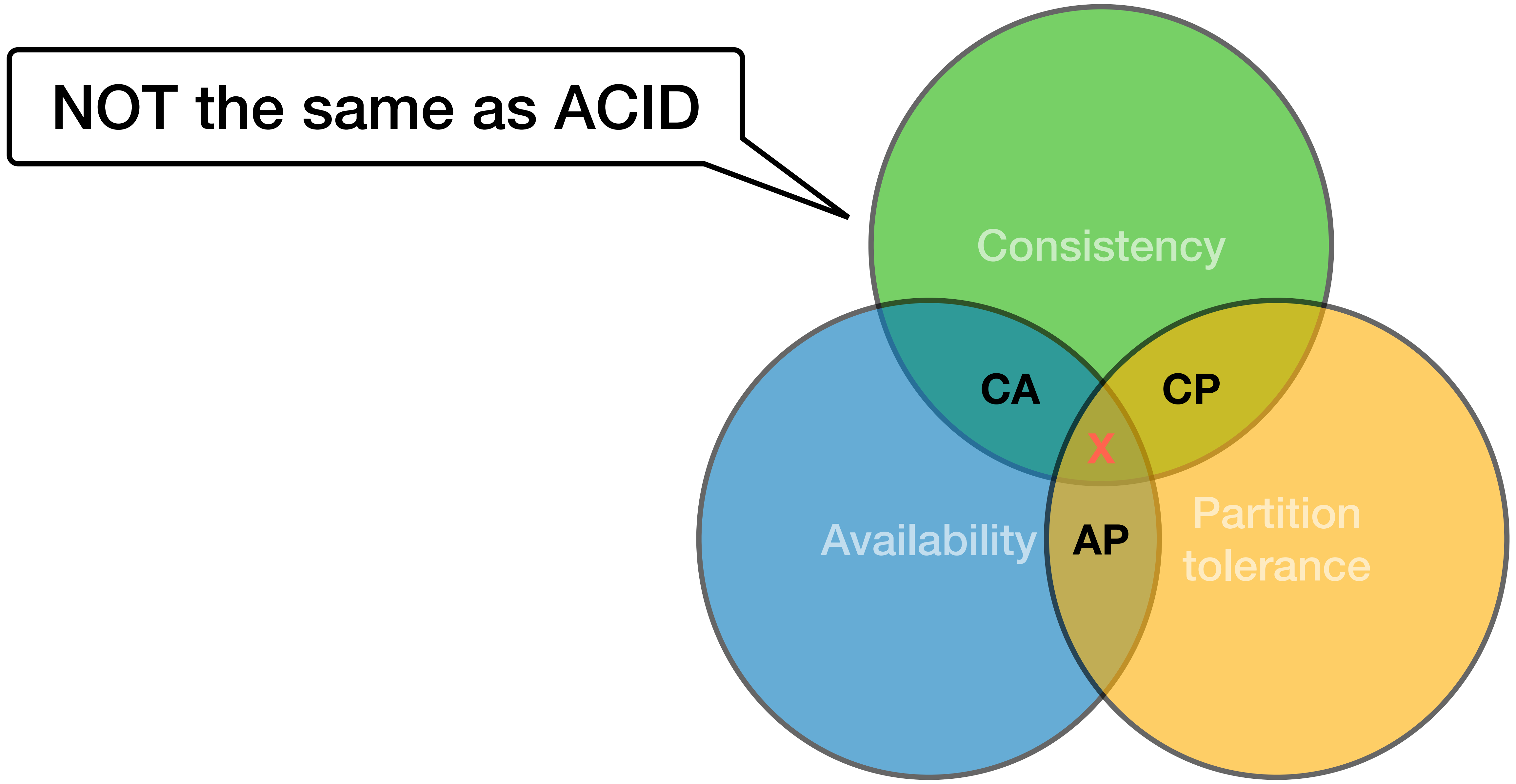
Motivation

We just learn it is “not trivial” to “go distributed”

- Data fragmentation
- Data distribution
- Data replication

- Things get (much) more complicated
- CAP Theorem - “Everything comes with a price”

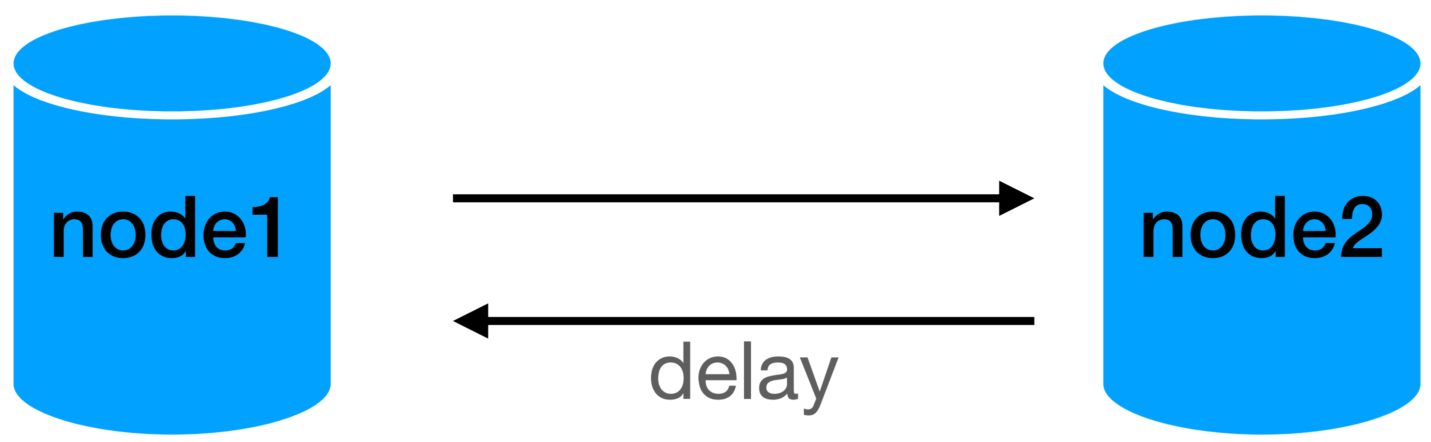
Some terms



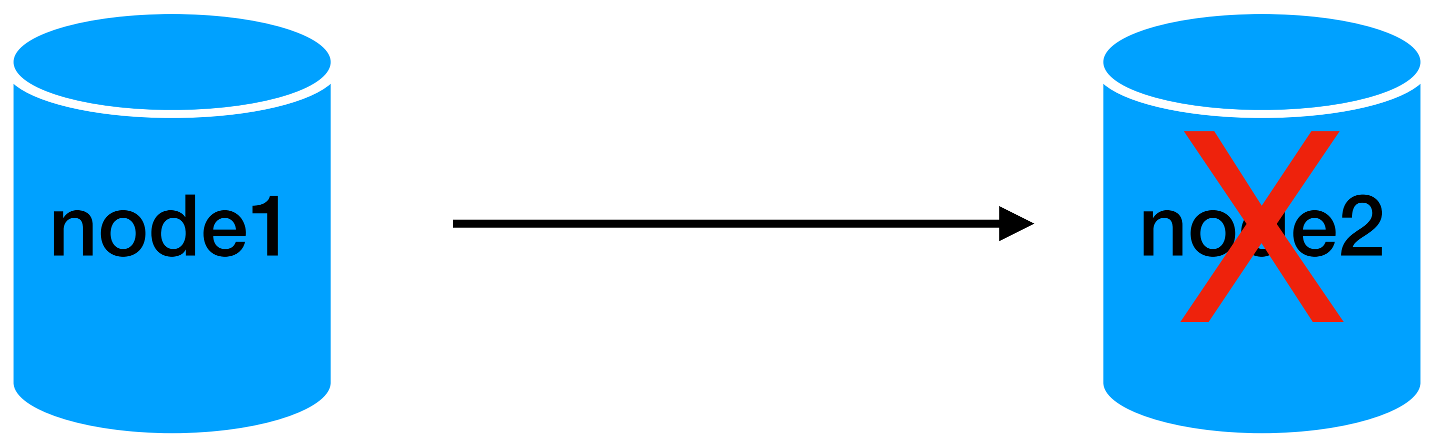
TLDR: You can only satisfy 2 out of 3 in a distributed database

Asynchronous network model

- Messages can be (randomly) delayed



- Can't distinguish between failed nodes and delayed messages in a finite amount of time

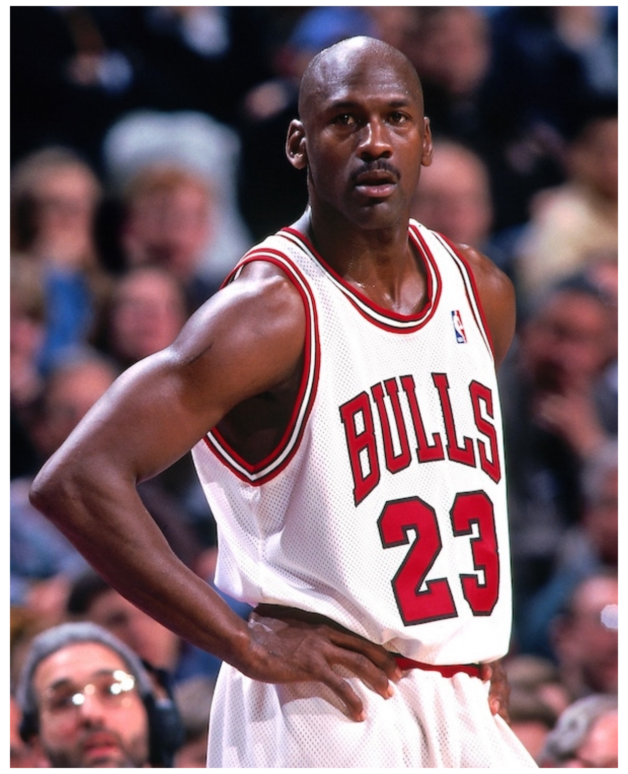


Consistency

- Every read receives the most recent write or an error

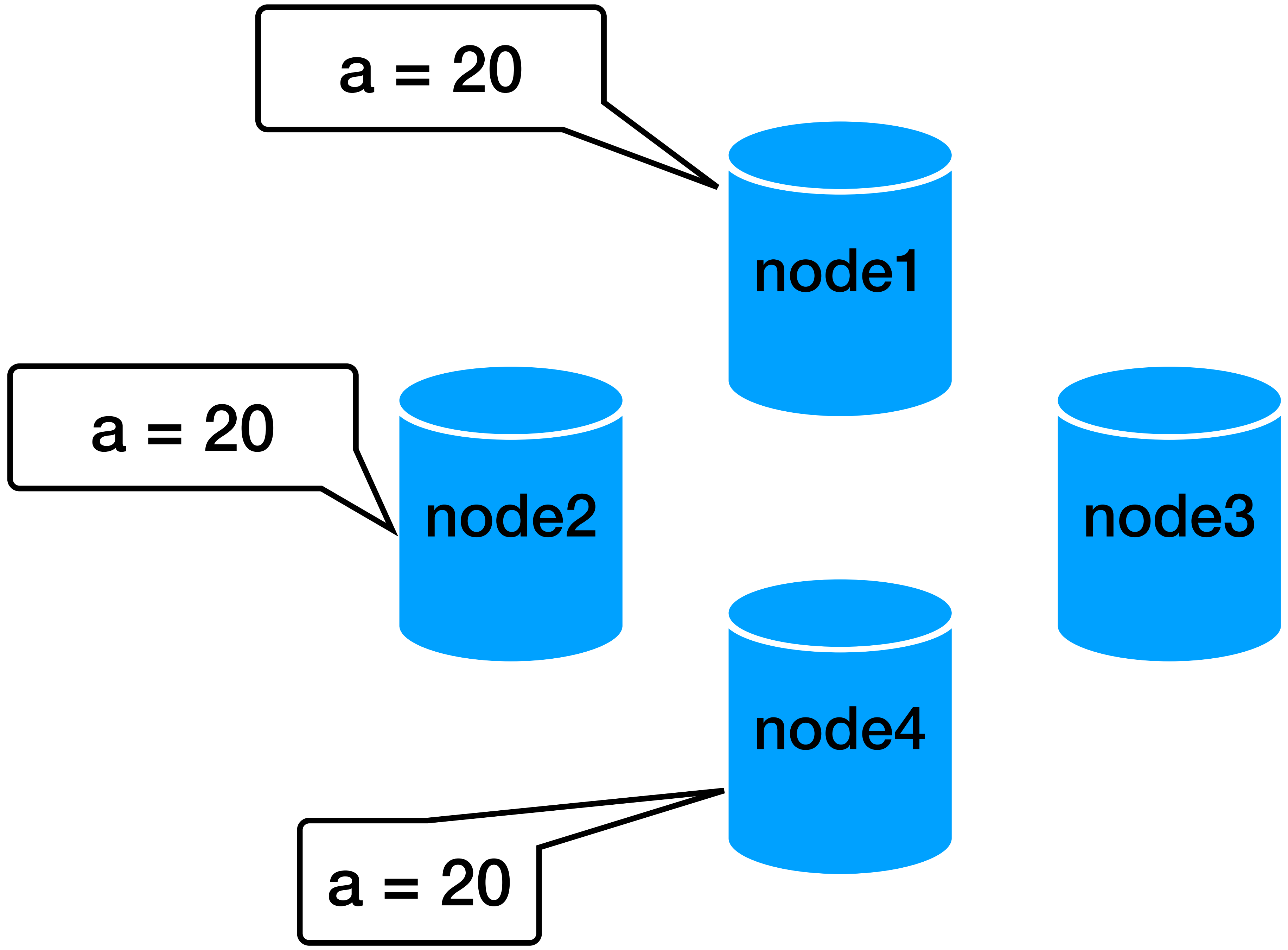
Consistency

- Every read receives the most recent write or an error



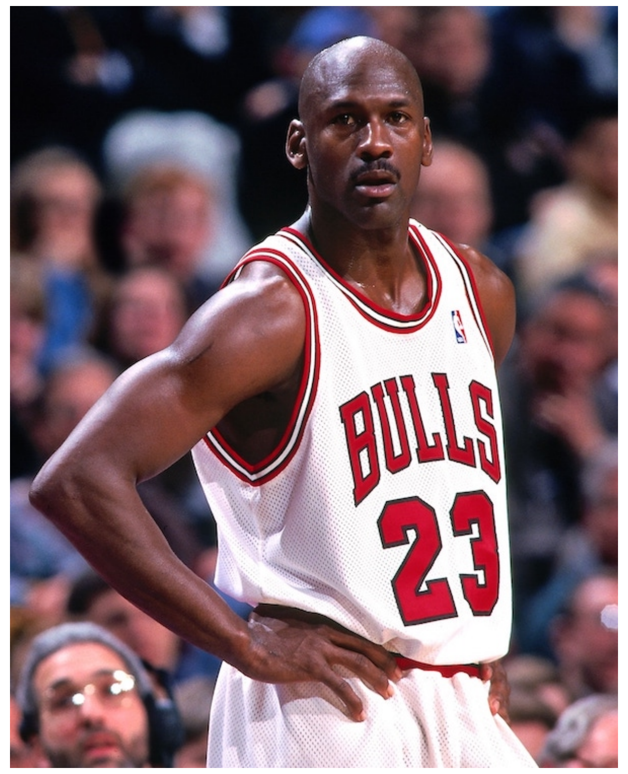
10:00: a = 20

* example for inconsistency



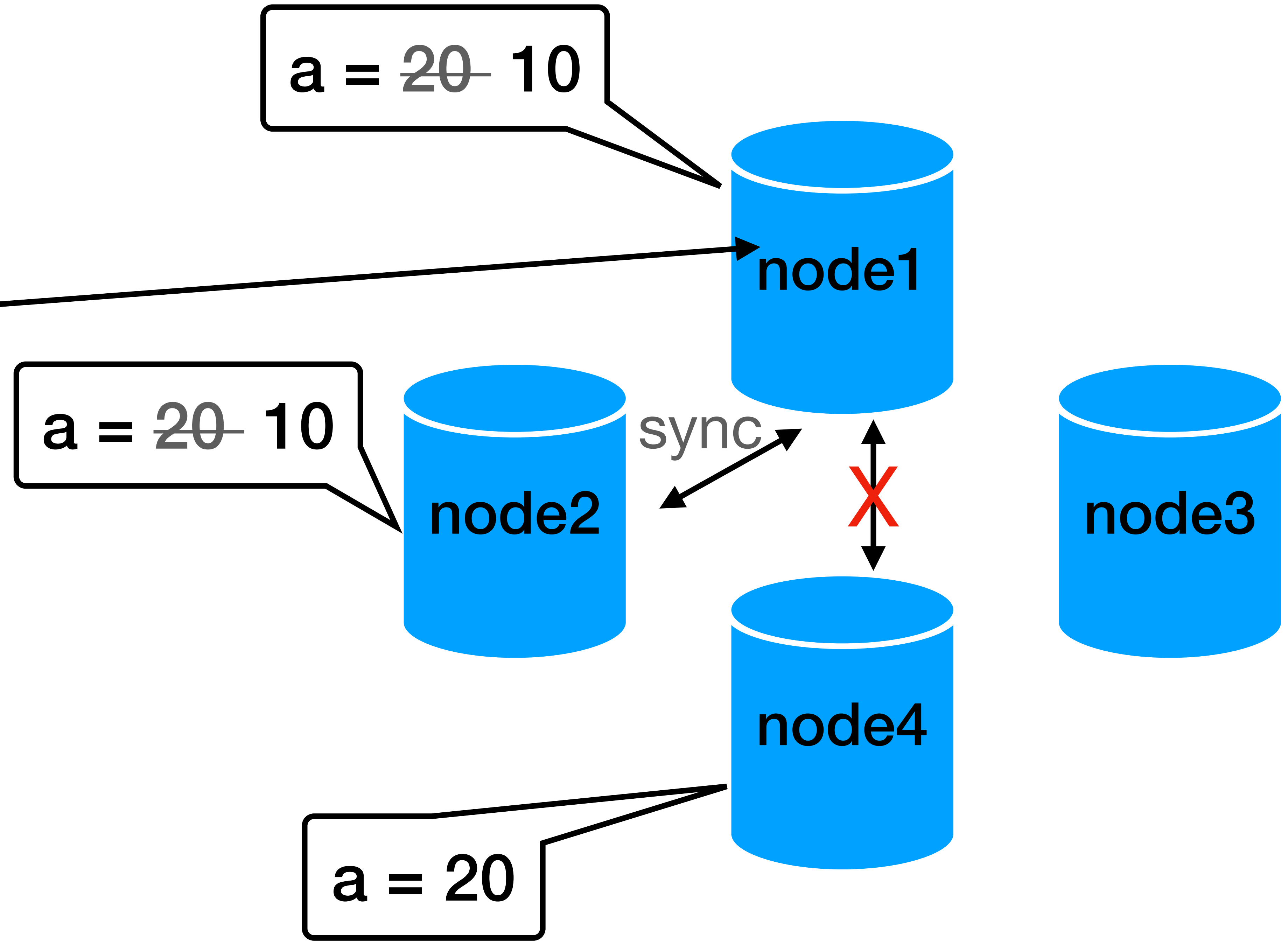
Consistency

- Every read receives the most recent write or an error



10:00: a = 20

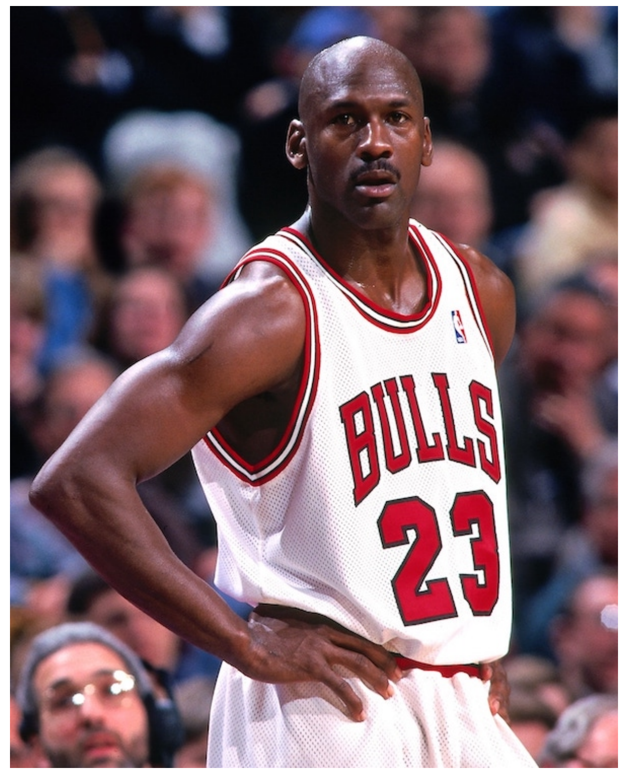
10:01: update a = 10



* example for inconsistency

Consistency

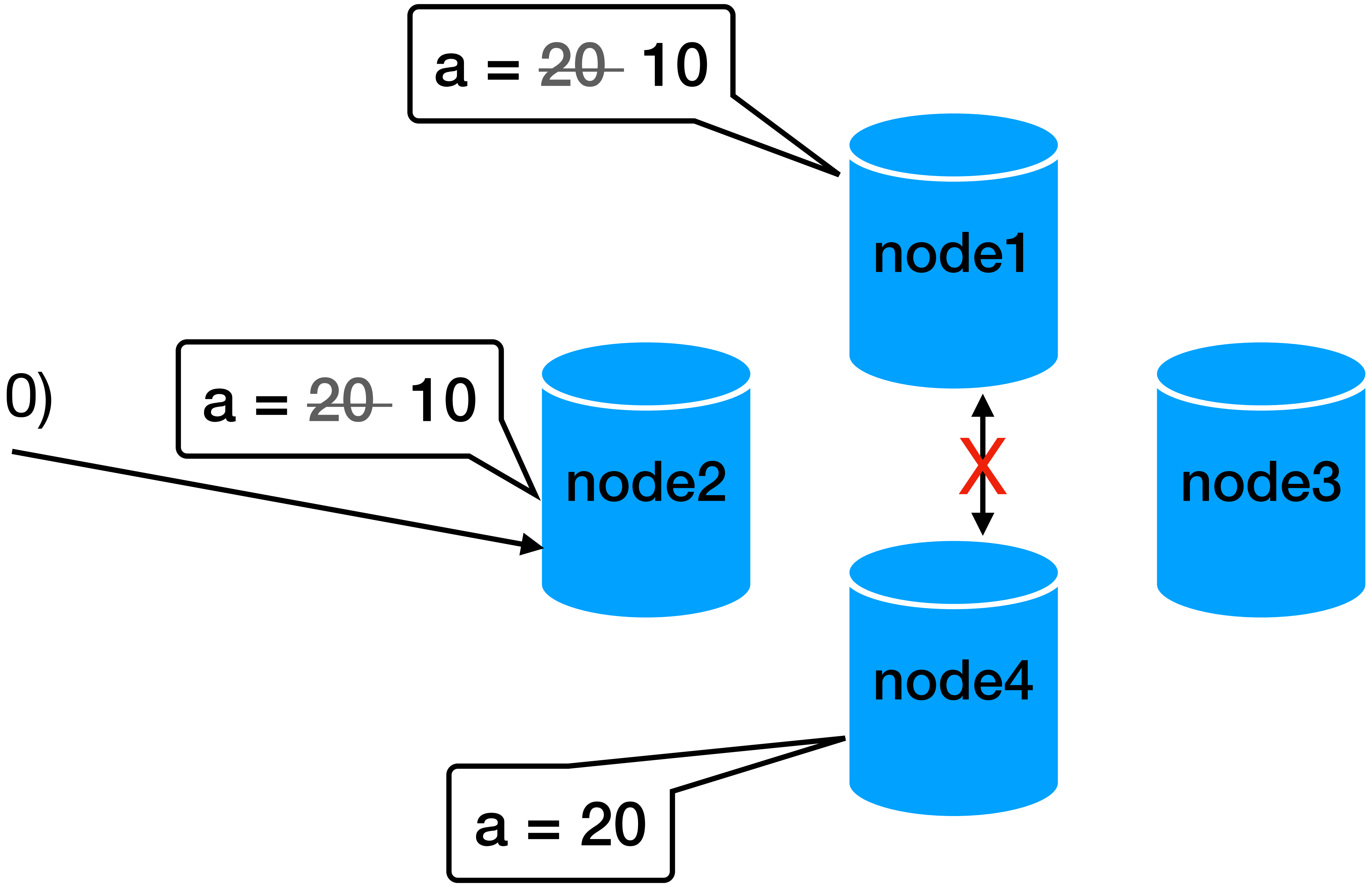
- Every read receives the most recent write or an error



10:00: a = 20

10:01: update a = 10

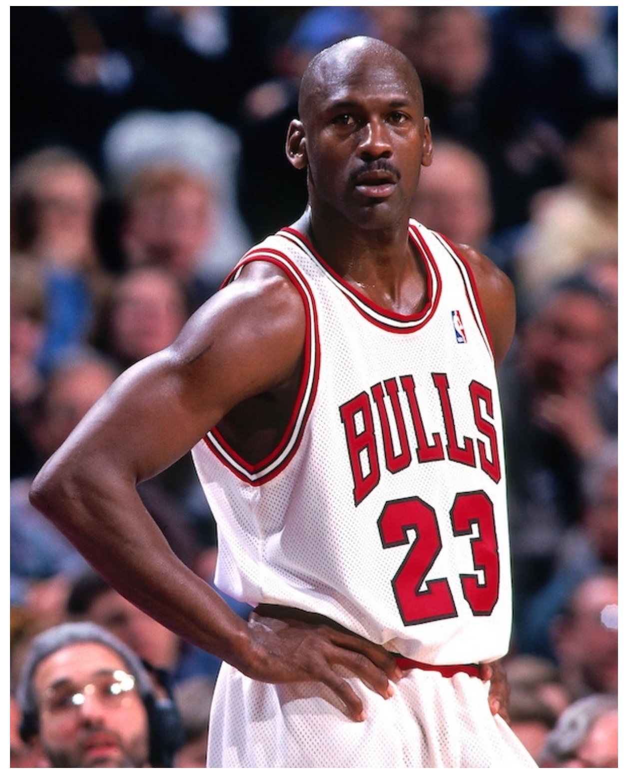
10:02: read a (value = 10)



* example for inconsistency

Consistency

- Every read receives the most recent write or an error

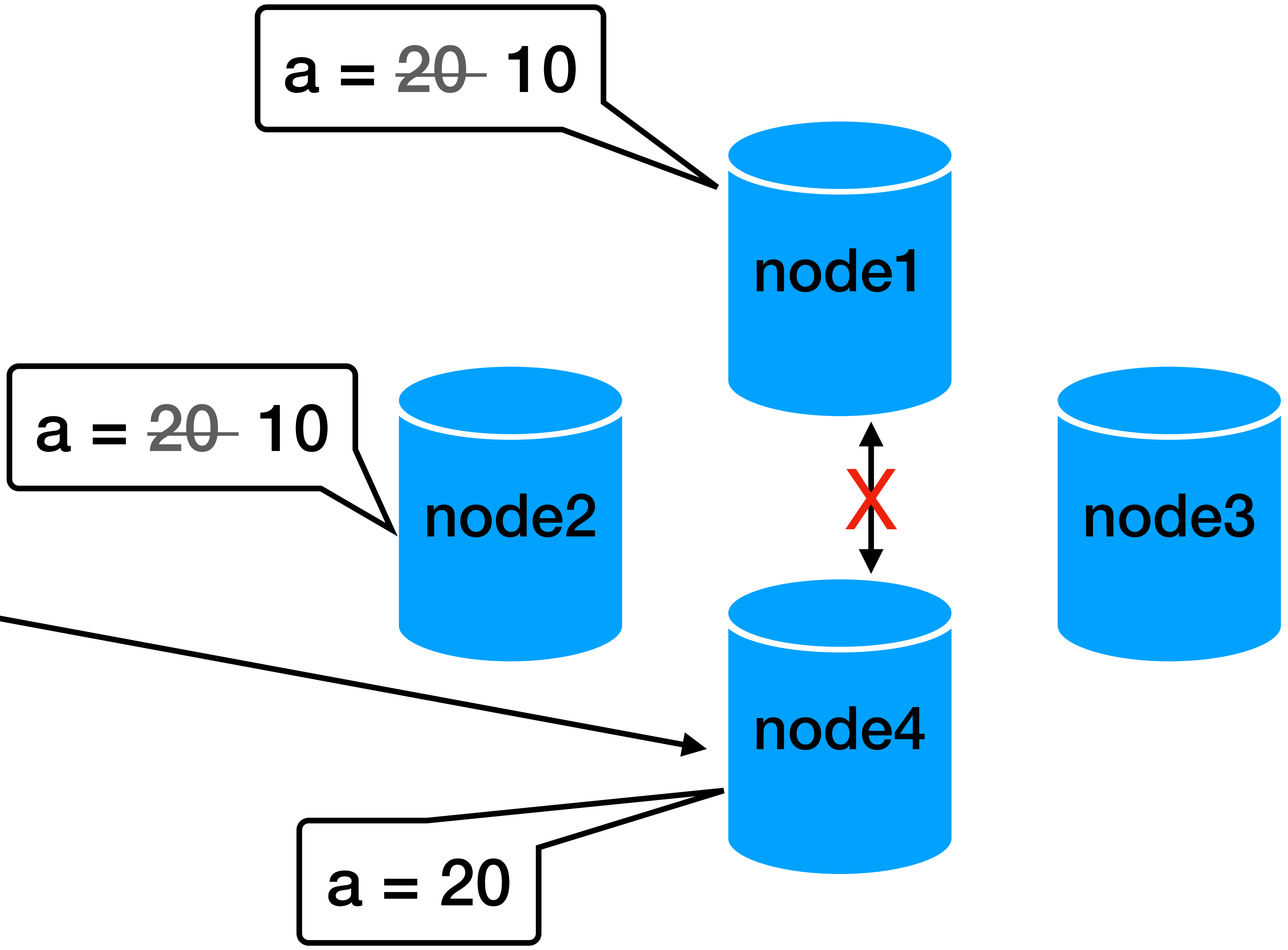


10:00: a = 20

10:01: update a = 10

10:02: read a (value = 10)

10:03: read a (value = 20)



* example for inconsistency

Consistency warning

Do not get confused with consistency from ACID

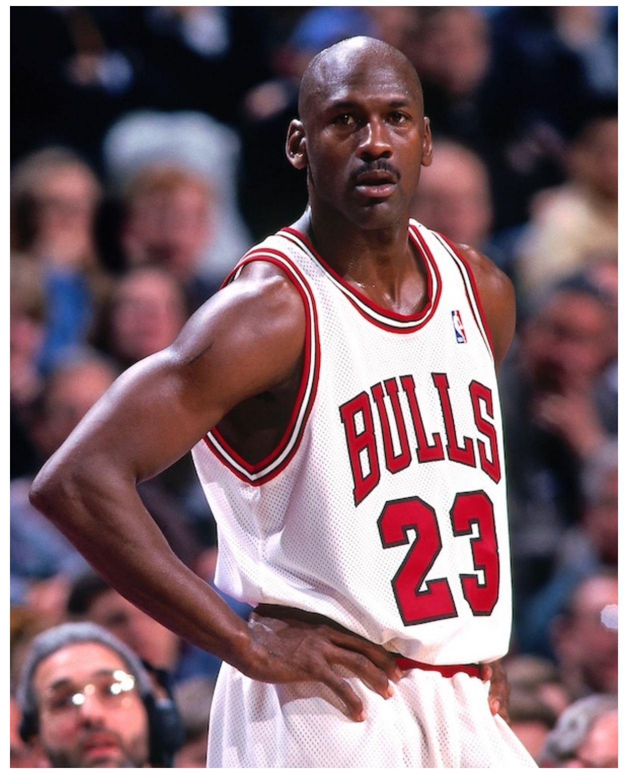
- **A**tomicity
- **C**onsistency
  correctness / referential integrity (foreign key)
- **I**solation
- **D**urability

Availability

- All requests (read/write) receives a non-error response
  <u>for reads there is no guarantee that it contains the most recent write</u>

Availability

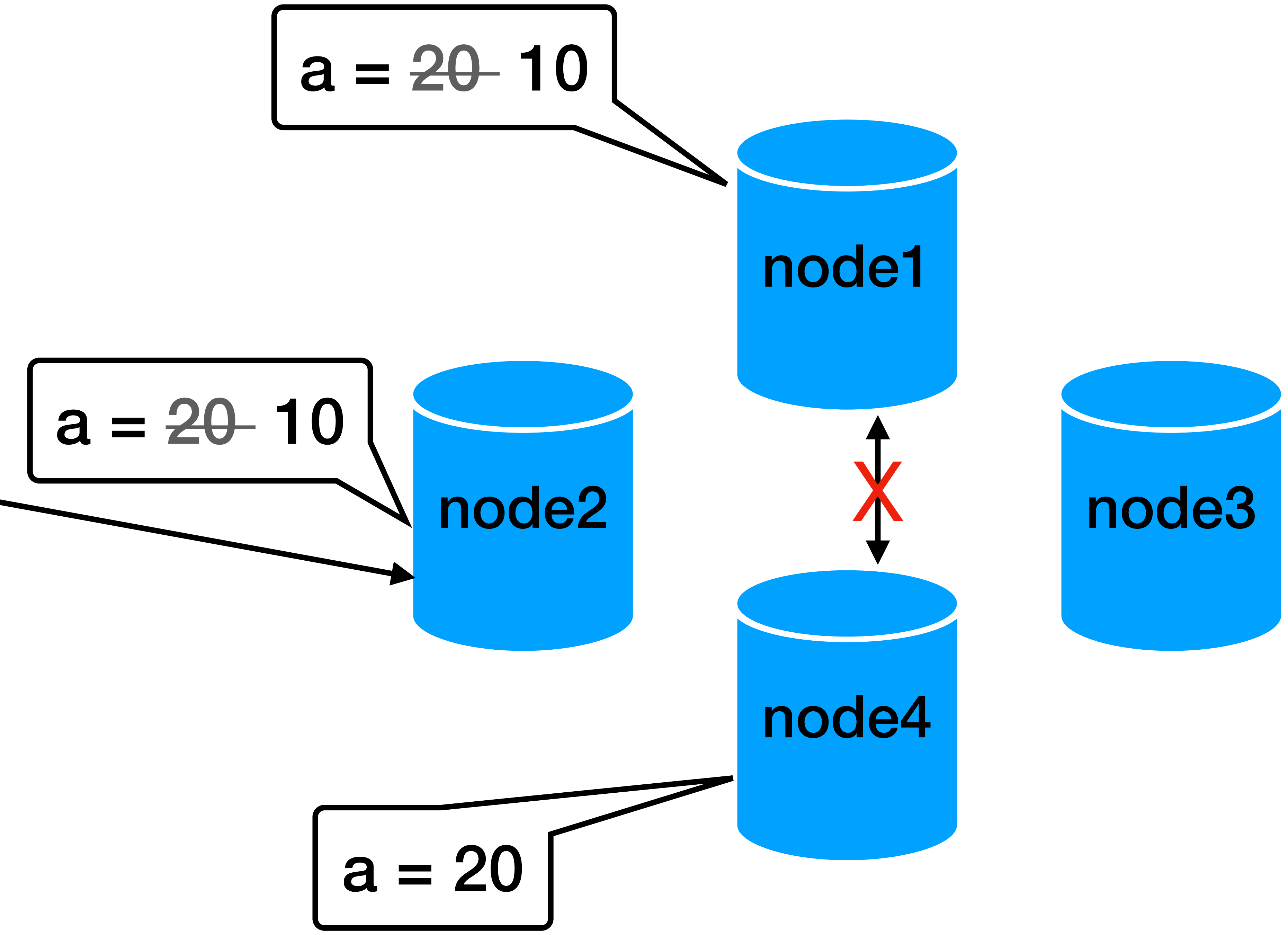
- All requests (read/write) receives a non-error response
  <u>for reads there is no guarantee that it contains the most recent write</u>



10:00: a = 20

10:01: update a = 10

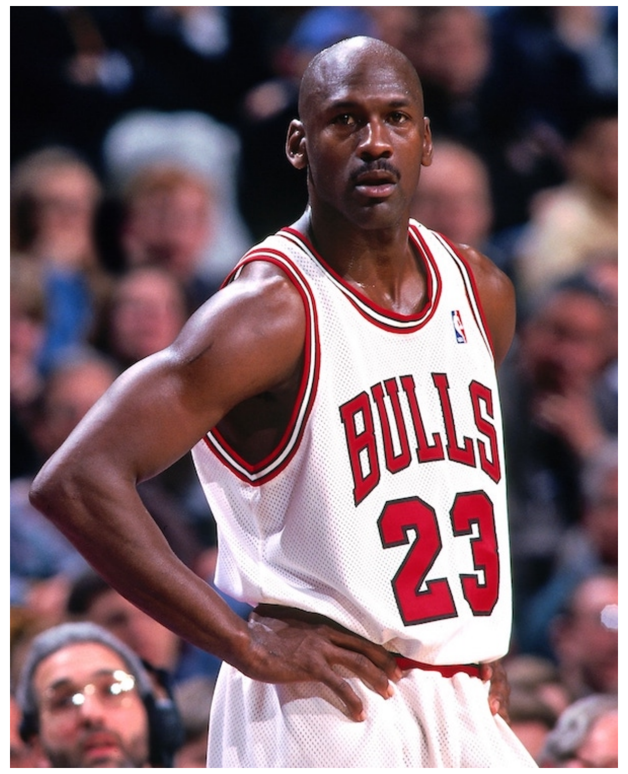
10:02: read a (value = 10)



* this is valid for high availability
(without consistency)

Availability

- All requests (read/write) receives a non-error response
  <u>for reads there is no guarantee that it contains the most recent write</u>



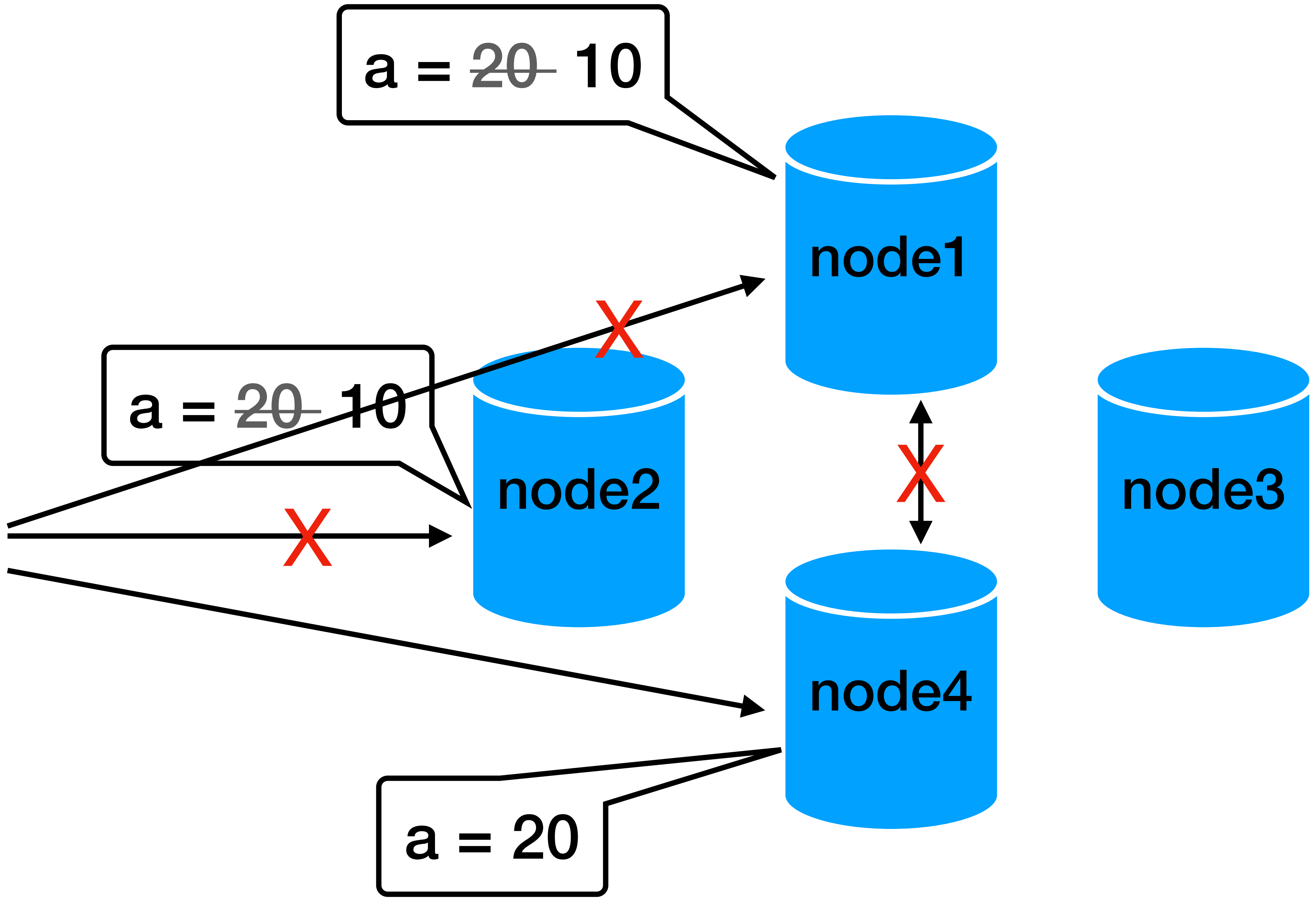
10:00: a = 20

10:01: update a = 10

10:02: read a (value = 10)

10:03: read a (value = 20)

* this is valid for high availability
  (without consistency)

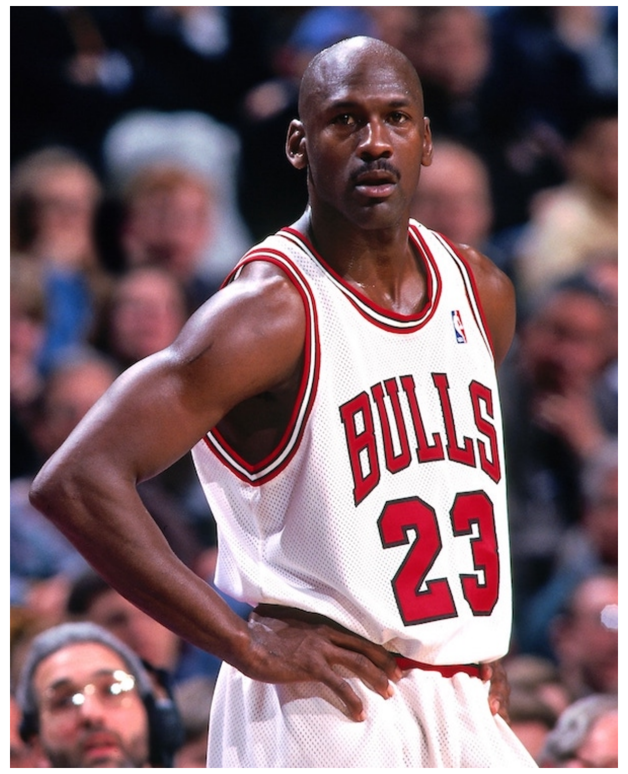


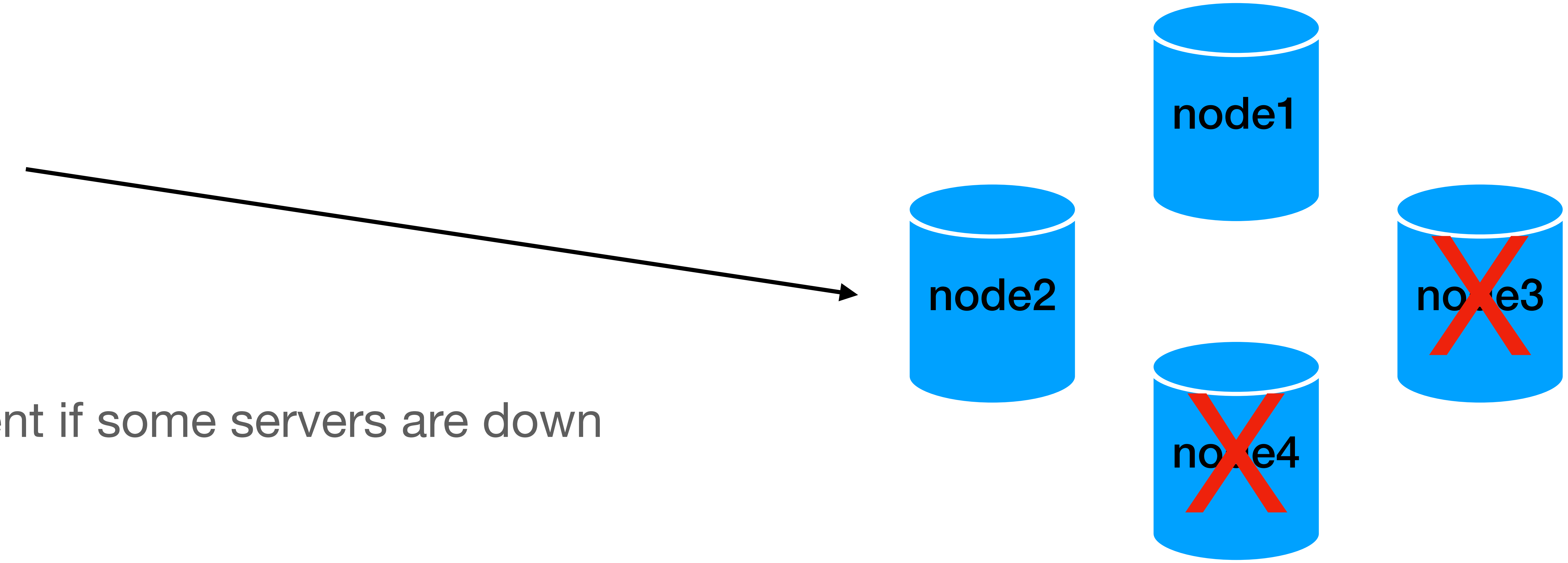
Partition tolerance

- The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network

Partition tolerance

- The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network





* success call event if some servers are down

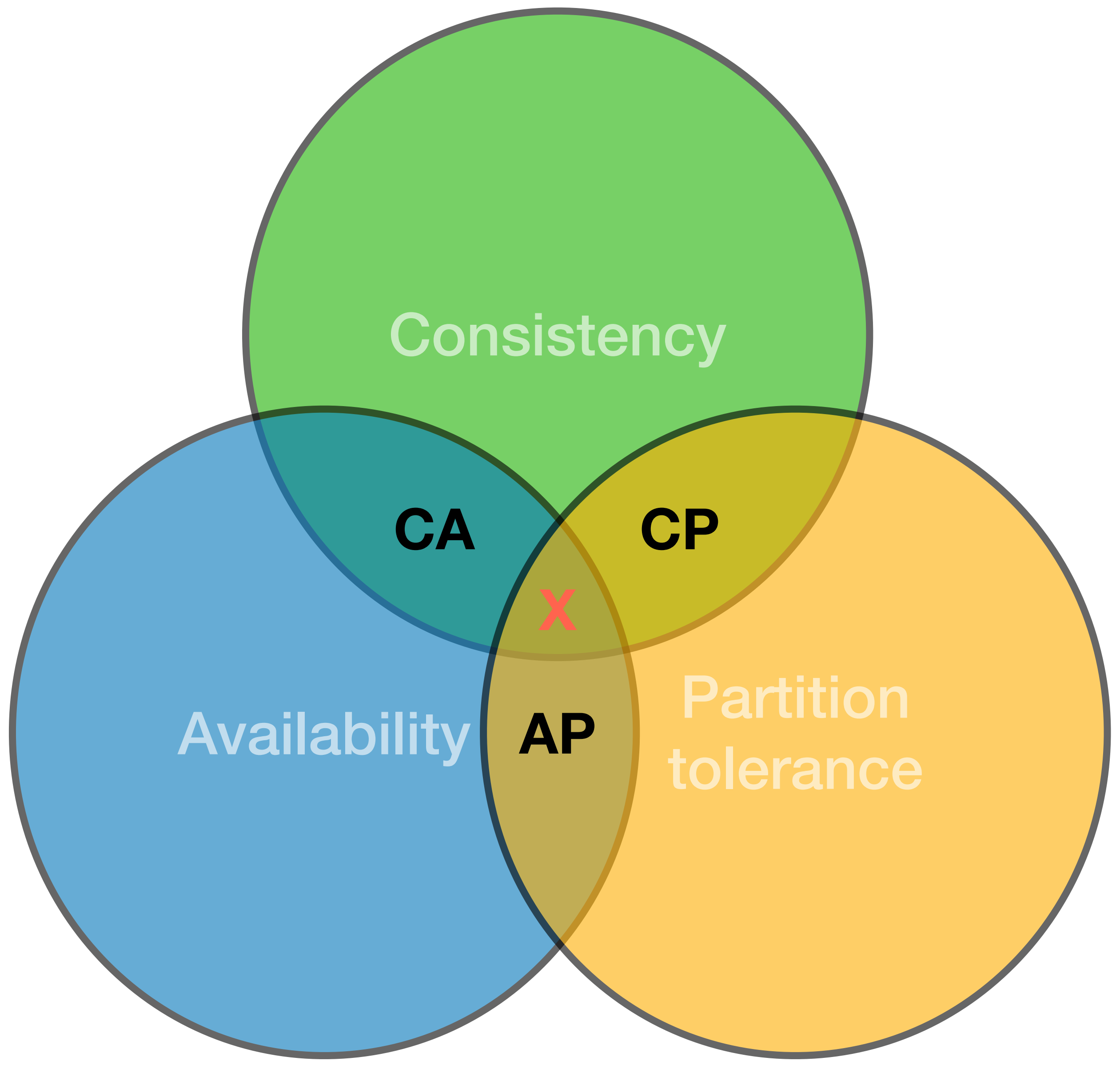
CAP Theorem

- For distributed data, it is impossible to satisfy more than two out of the three

- Consistency
  Every read receives the most recent write or an error

- Availability
  Every request receives a (non-error) response,
  without the guarantee that it contains the most recent write

- Partition tolerance
  The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network



CAP Theorem - in practice

No distributed system is safe from network failures.
—> we need to choose between CP and AP

In practice - If a node is down/unreachable we can:

- cancel the operation (CP)
- Return result with (maybe) inconsistency (AP)

CAP Theorem - why is it important?

- No free lunch for distributed systems

- This will be (among other stuff) a differentiator between different types of distributed databases and NoSQL systems
(not just how to model data, but how to write)

A bit more on Consistency

Consistency types

- Weak / Eventual consistency
  If we stop updating, the system will eventually be consistent

- Strong consistency
  consistent on all calls

Consistency types - different views

- From developer / application side
  how they observe updates?
  how it affects the application logic?

- From server side
  how can we detect / force consistency?

Consistency types - different views

- **From developer / application side**
  how they observe updates?
  how it affects the application logic?

- From server side
  how can we detect / force consistency?

Application side consistency

DNS Server

Which consistency type do we need?

Application side consistency

DNS Server

Weak / Eventual consistency

Application side consistency

Bank

Which consistency type do we need?

Application side consistency

Bank

Strong consistency

Application side consistency

Bank

Note that some “logic” is usually “eventual”

Strong consistency

Now with the CAP

DNS Server

Weak / Eventual consistency

Bank

Strong consistency

Should we prefer consistency or availability support?

Consistency types - different views

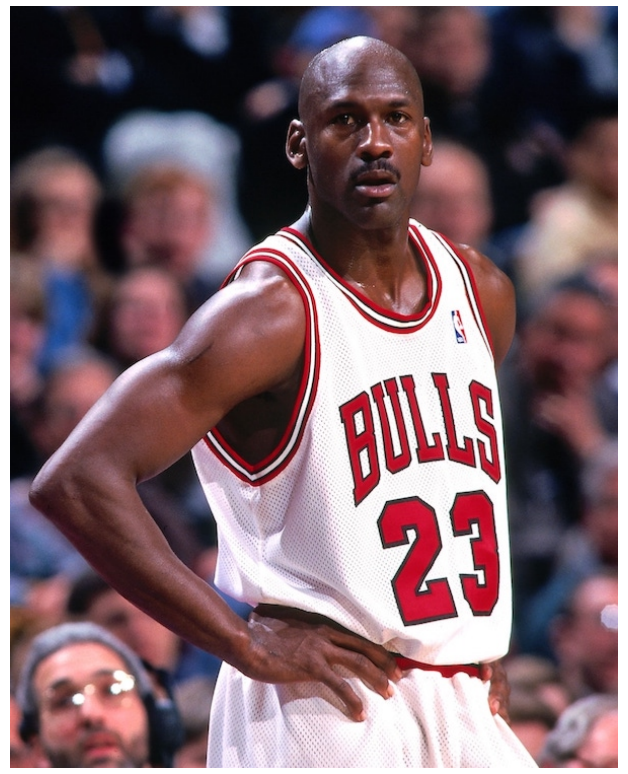
- **From developer / application side**
  how they observe updates?
  how it affects the application logic?

- **From server side**
  how can we detect / force consistency?

Discussion

Server side consistency

Discussion - How do we know if we satisfy consistency?
if one, two or more (how much?) are down

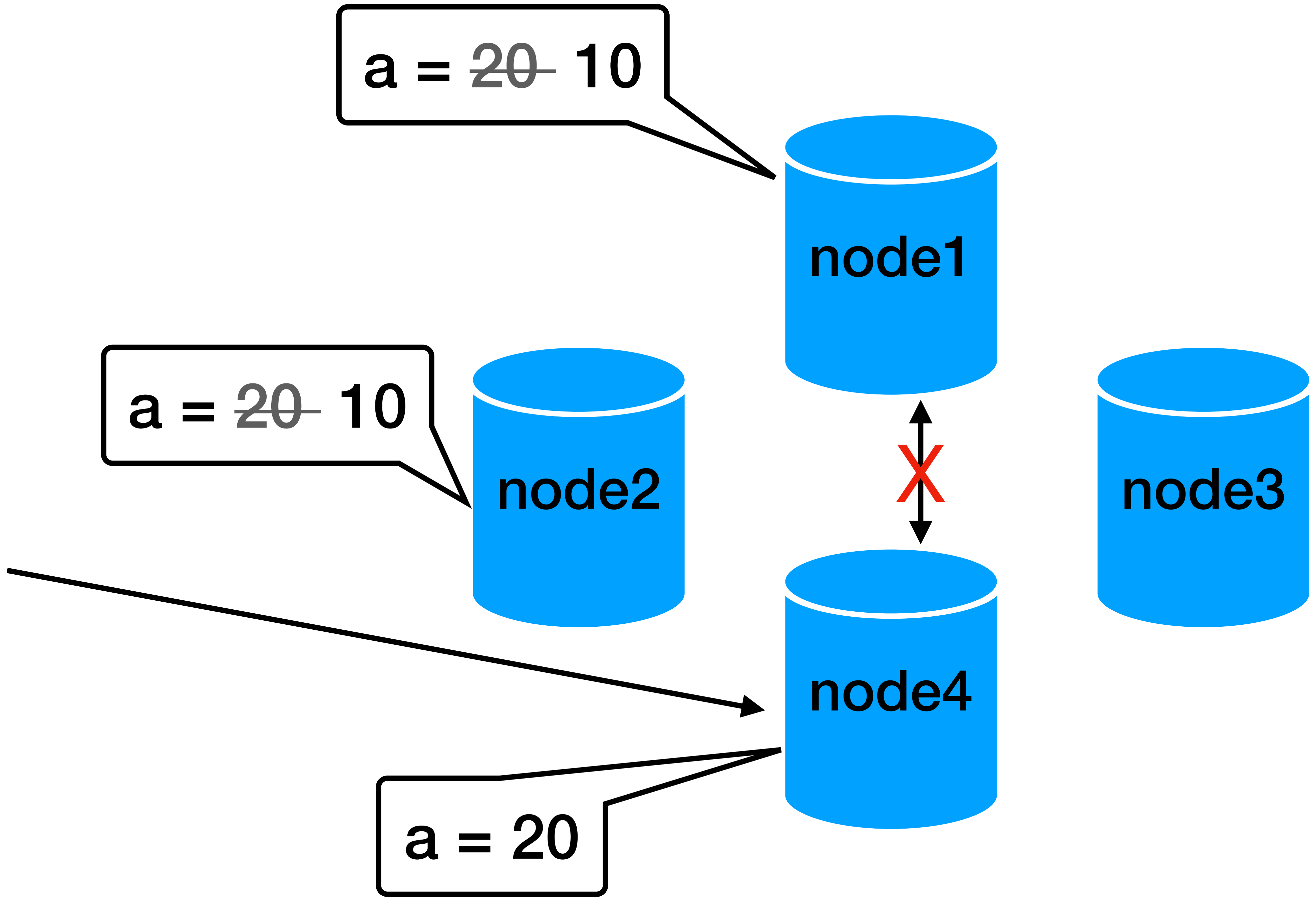


10:00: a = 20

10:01: update a = 10

10:02: read a (value = 10)

10:03: read a (value = 20)



Server side consistency

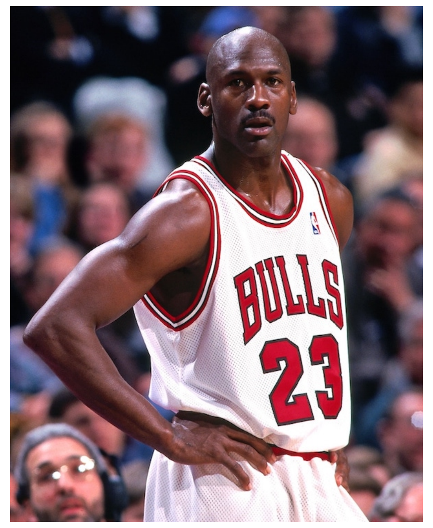
- **N** #nodes that store replicas of the data
- **W** #replicas that need to acknowledge the receipt of the update before the update completes
- **R** #replicas that are contacted for a read

If W+R > N then strong consistency is guaranteed

If W+R <=N then weak / eventual consistency

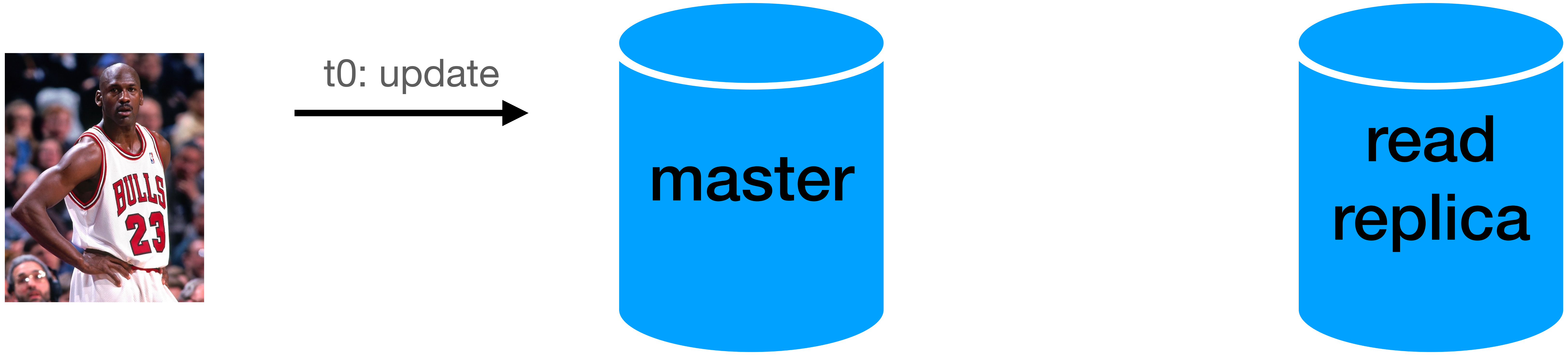
Server side consistency - example 1

- Master + read replica RDBMS



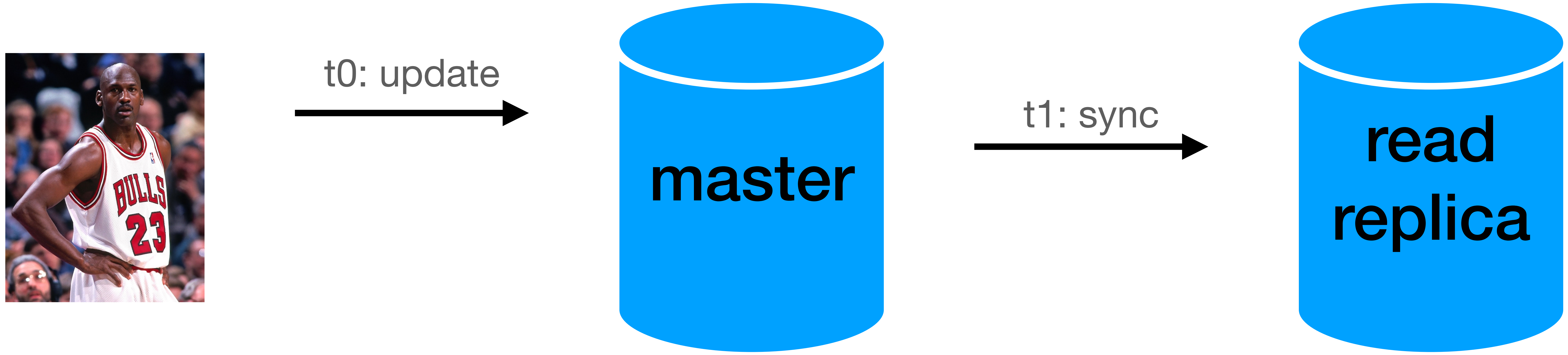
Server side consistency - example 1

- Master + read replica RDBMS



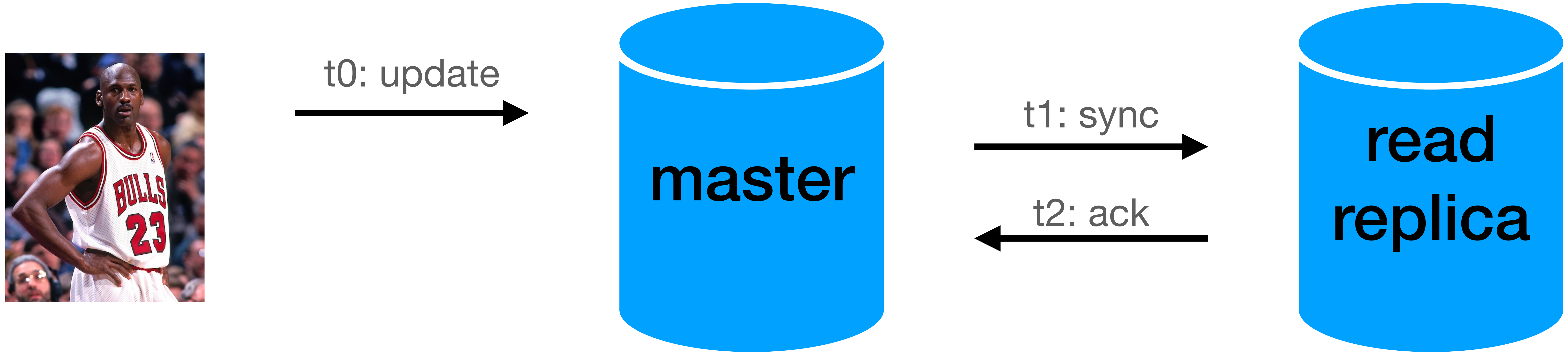
Server side consistency - example 1

- Master + read replica RDBMS



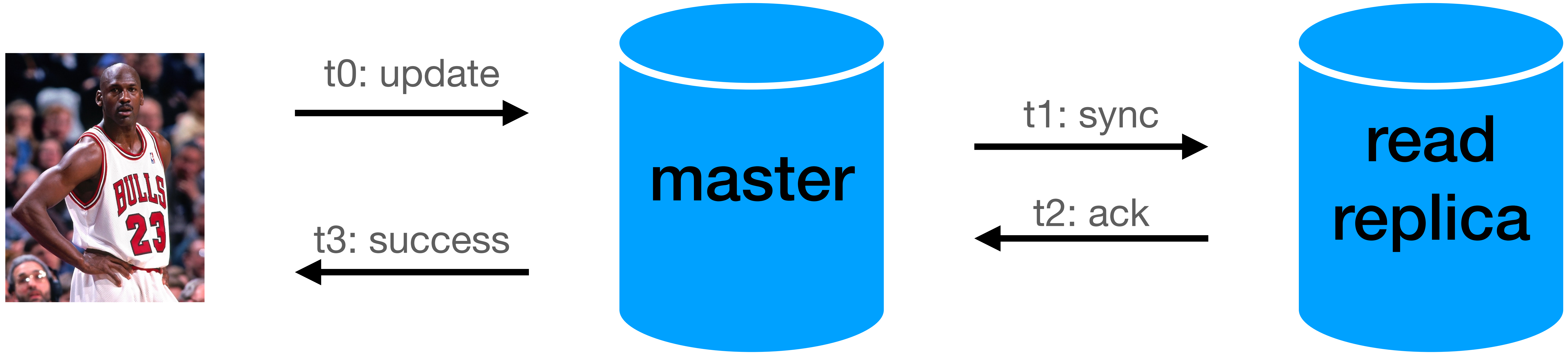
Server side consistency - example 1

- Master + read replica RDBMS



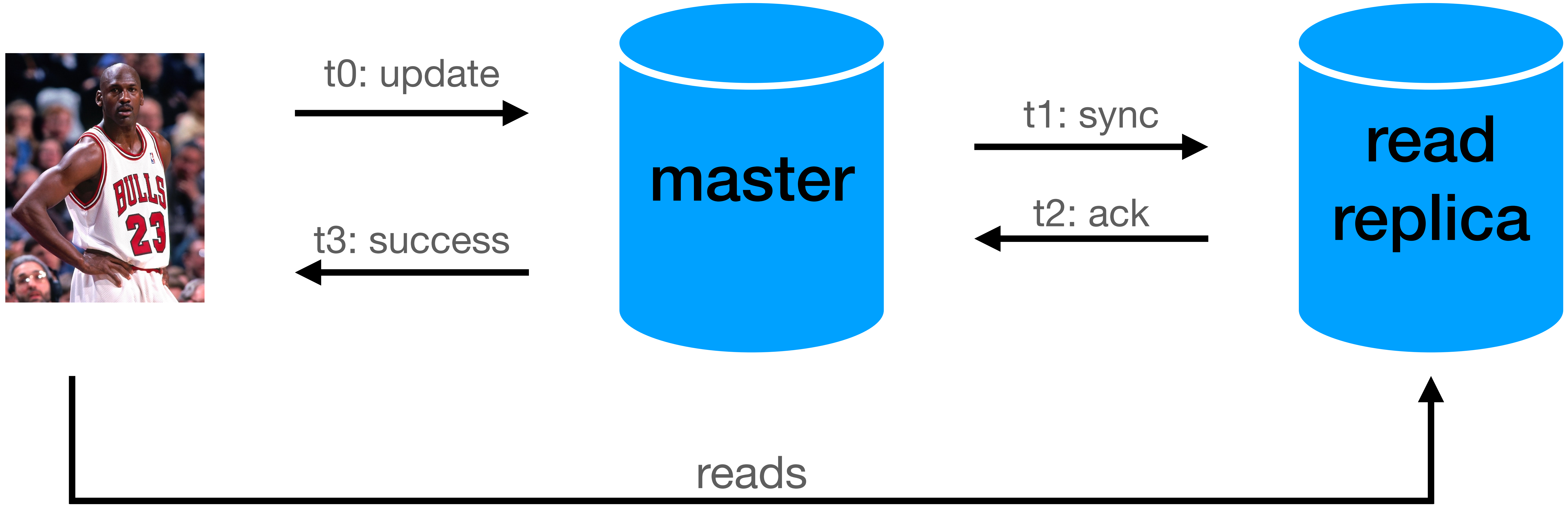
Server side consistency - example 1

- Master + read replica RDBMS



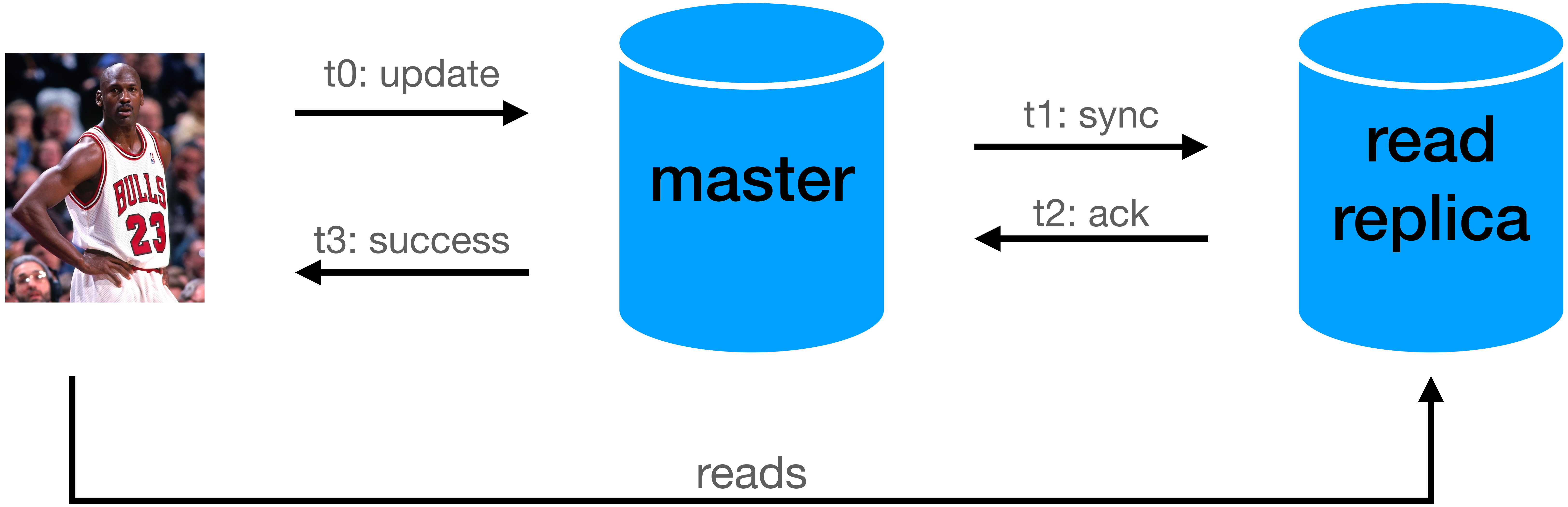
Server side consistency - example 1

- Master + read replica RDBMS



Server side consistency - example 1

- Master + read replica RDBMS

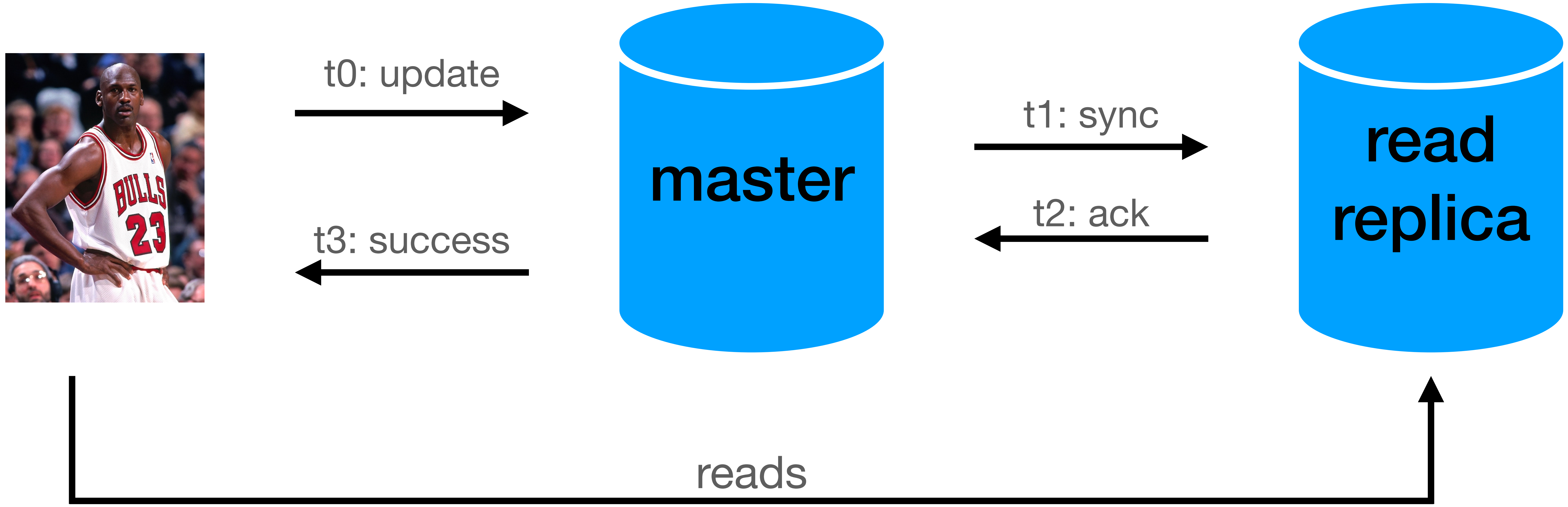


W (2) + R (1) > N (2)

strong consistency

Server side consistency - example 1

- Master + read replica RDBMS

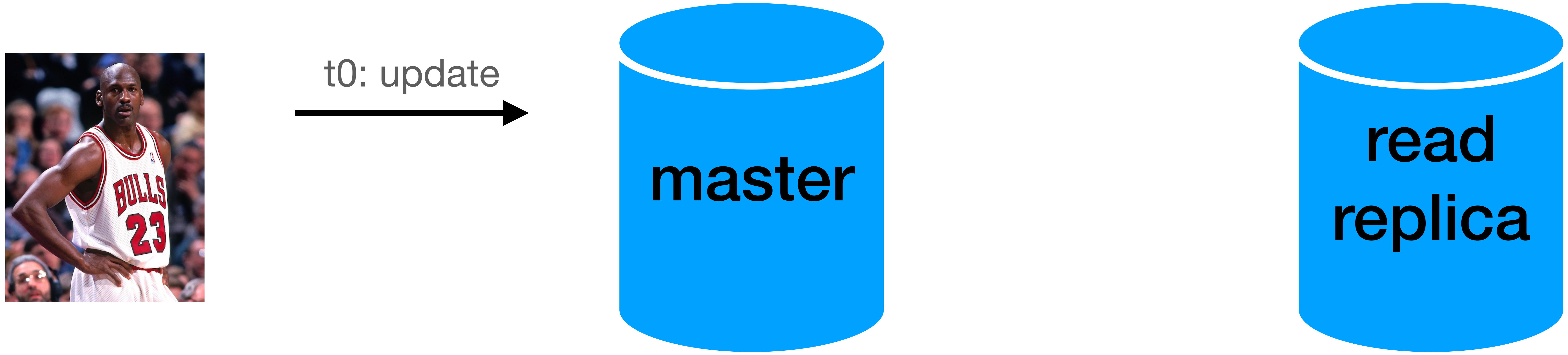


W (2) + R (1) > N (2)
strong consistency

What happens if the read replica fails?

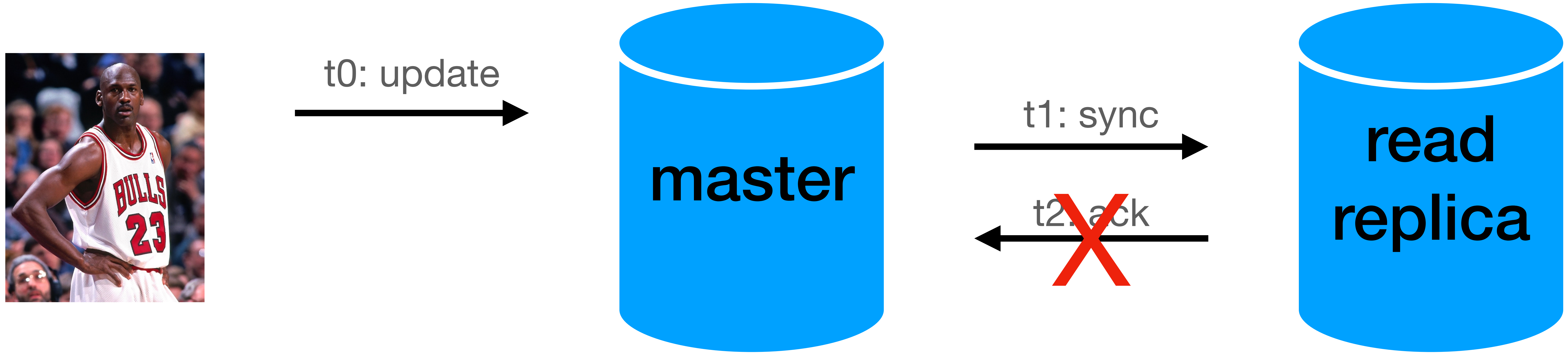
Server side consistency - example 1

- Master + read replica RDBMS



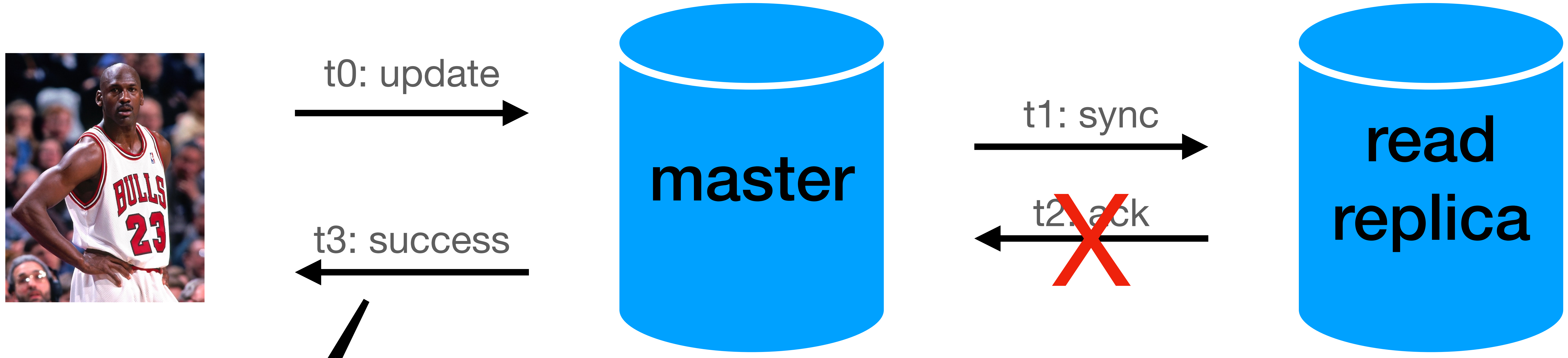
Server side consistency - example 1

- Master + read replica RDBMS



Server side consistency - example 1

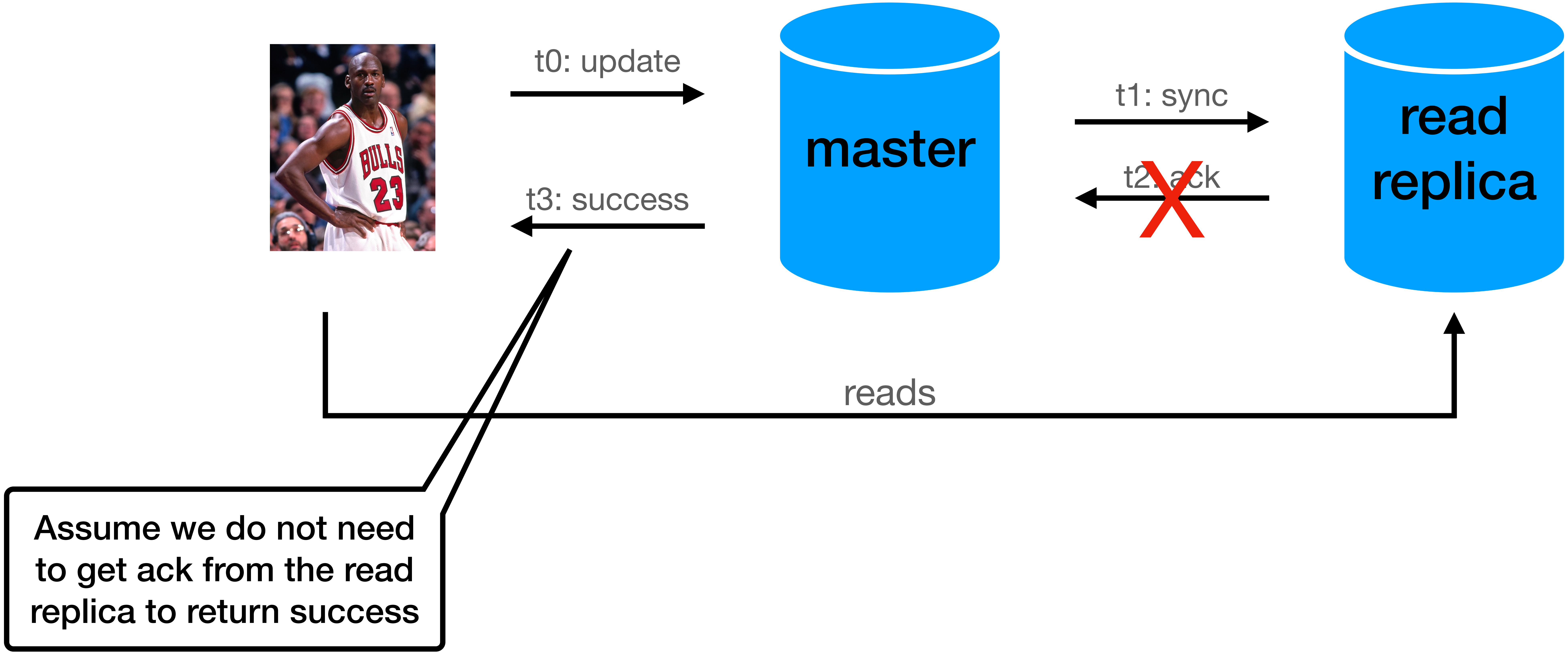
- Master + read replica RDBMS



Assume we do not need to get ack from the read replica to return success

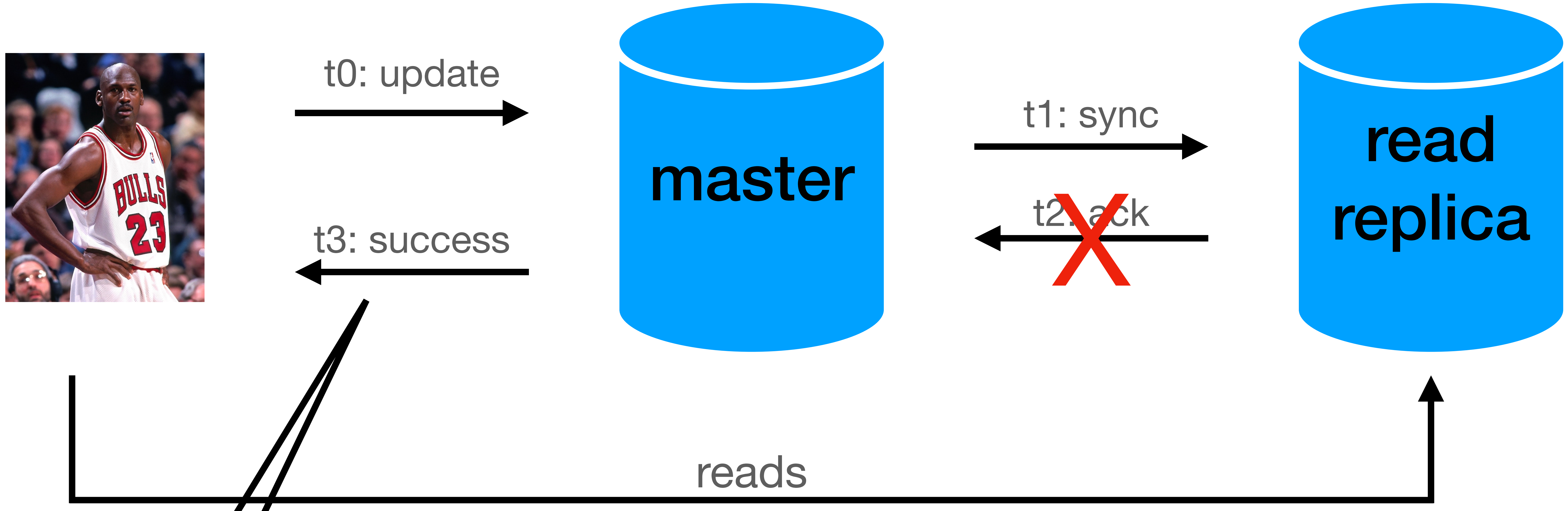
Server side consistency - example 1

- Master + read replica RDBMS



Server side consistency - example 1

- Master + read replica RDBMS



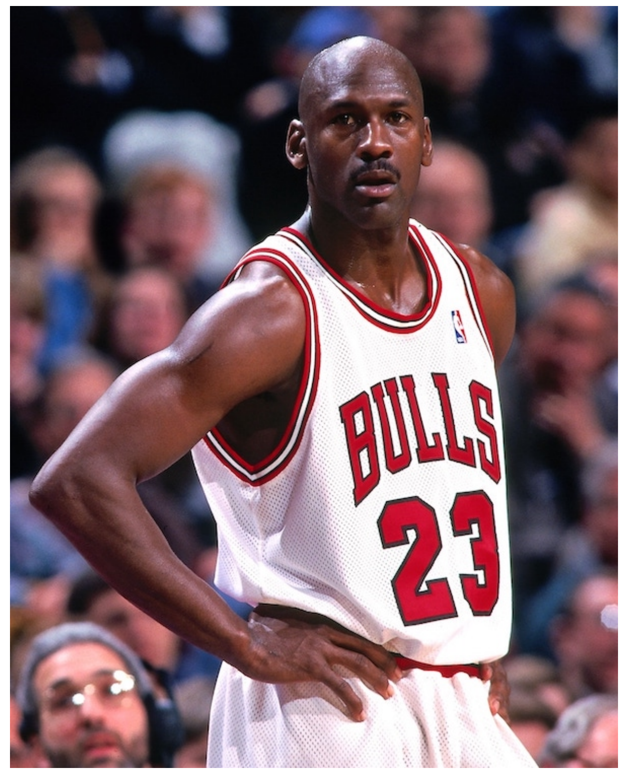
Assume we do not need to get ack from the read replica to return success

W (1) + R (1) <= N (2)

weak / eventual consistency

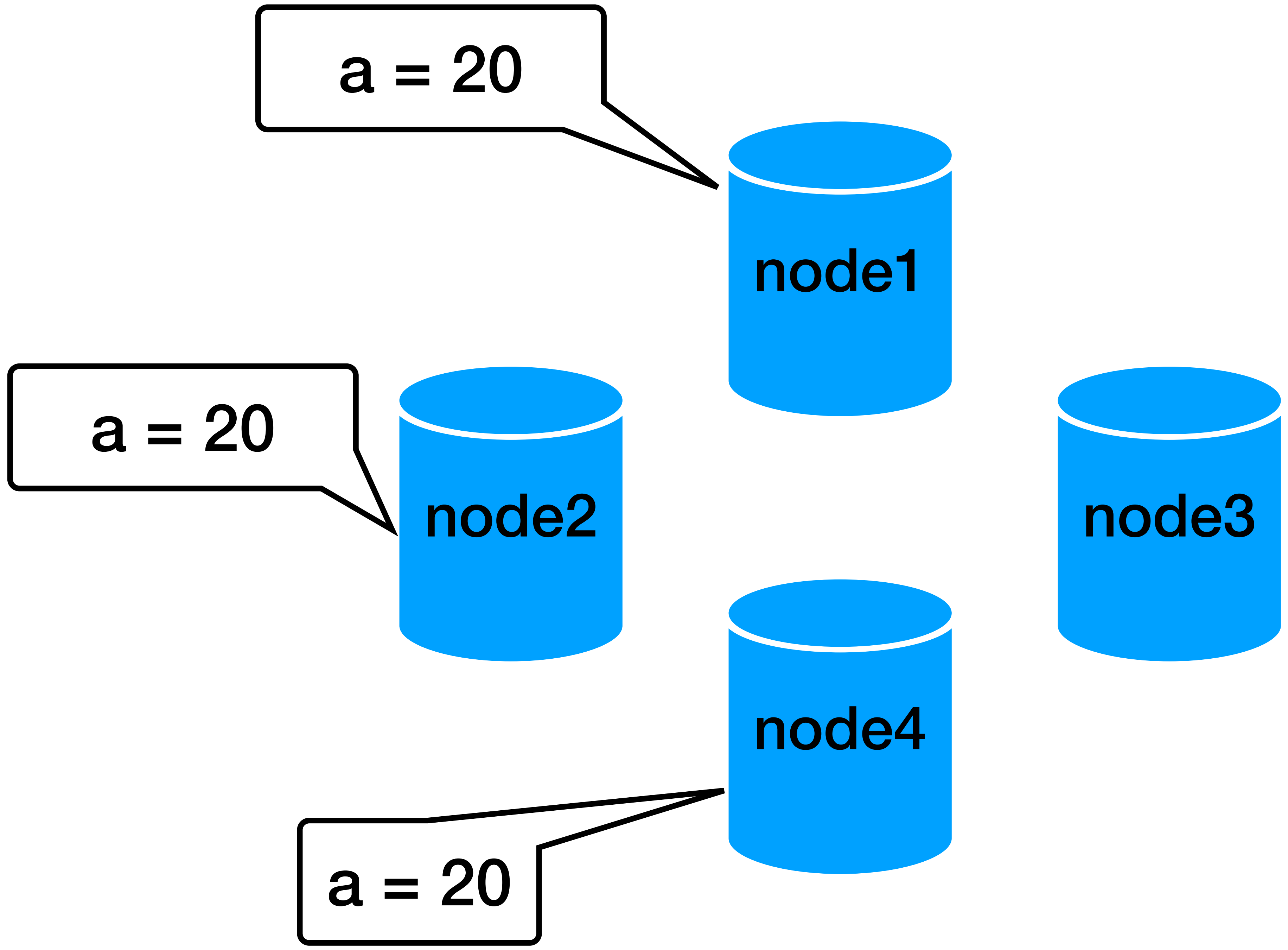
Server side consistency - example 2

- Distributed database, set to performance (availability)
  updates other nodes asynchronously



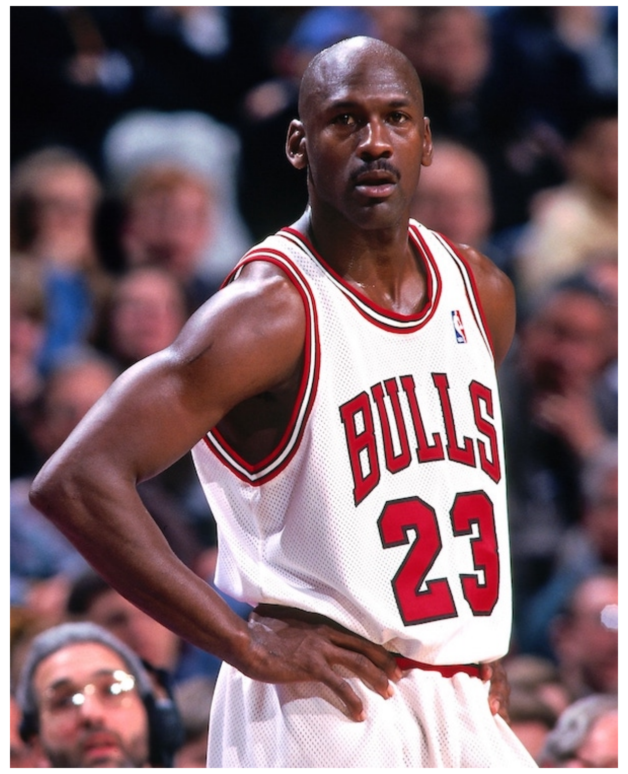
10:00: a = 20

* example for availability



Server side consistency - example 2

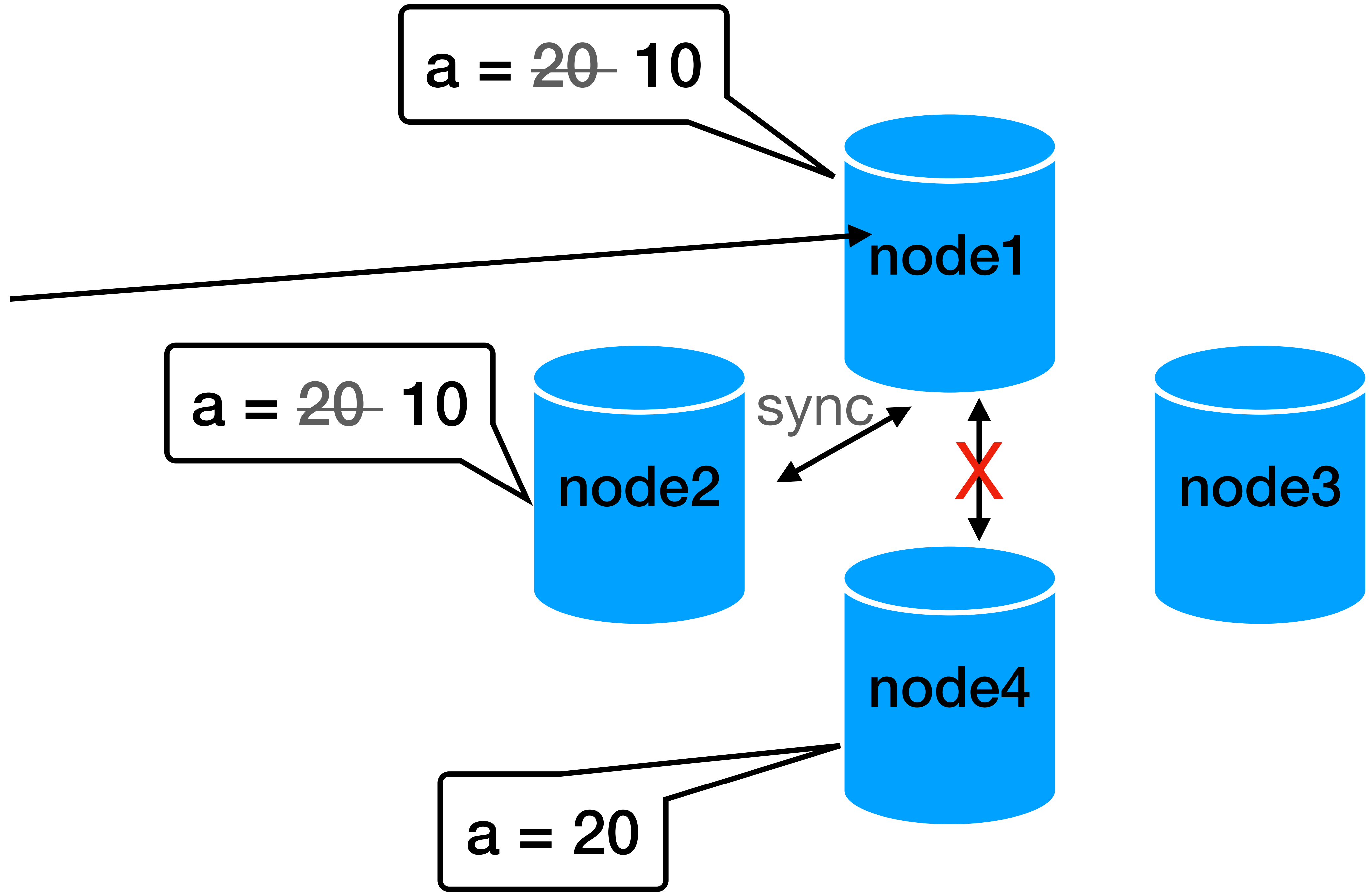
- Distributed database, set to performance (availability)
  updates other nodes asynchronously



* example for availability

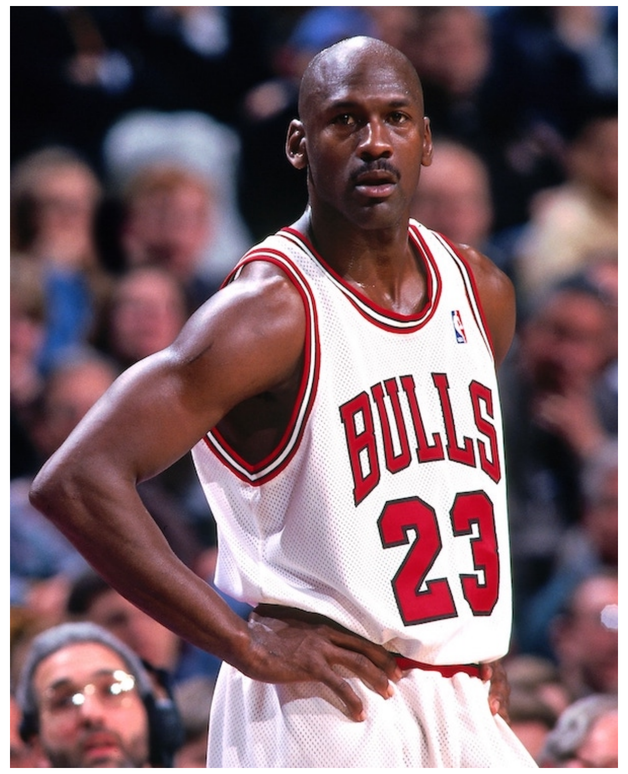
10:00: a = 20

10:01: update a = 10



Server side consistency - example 2

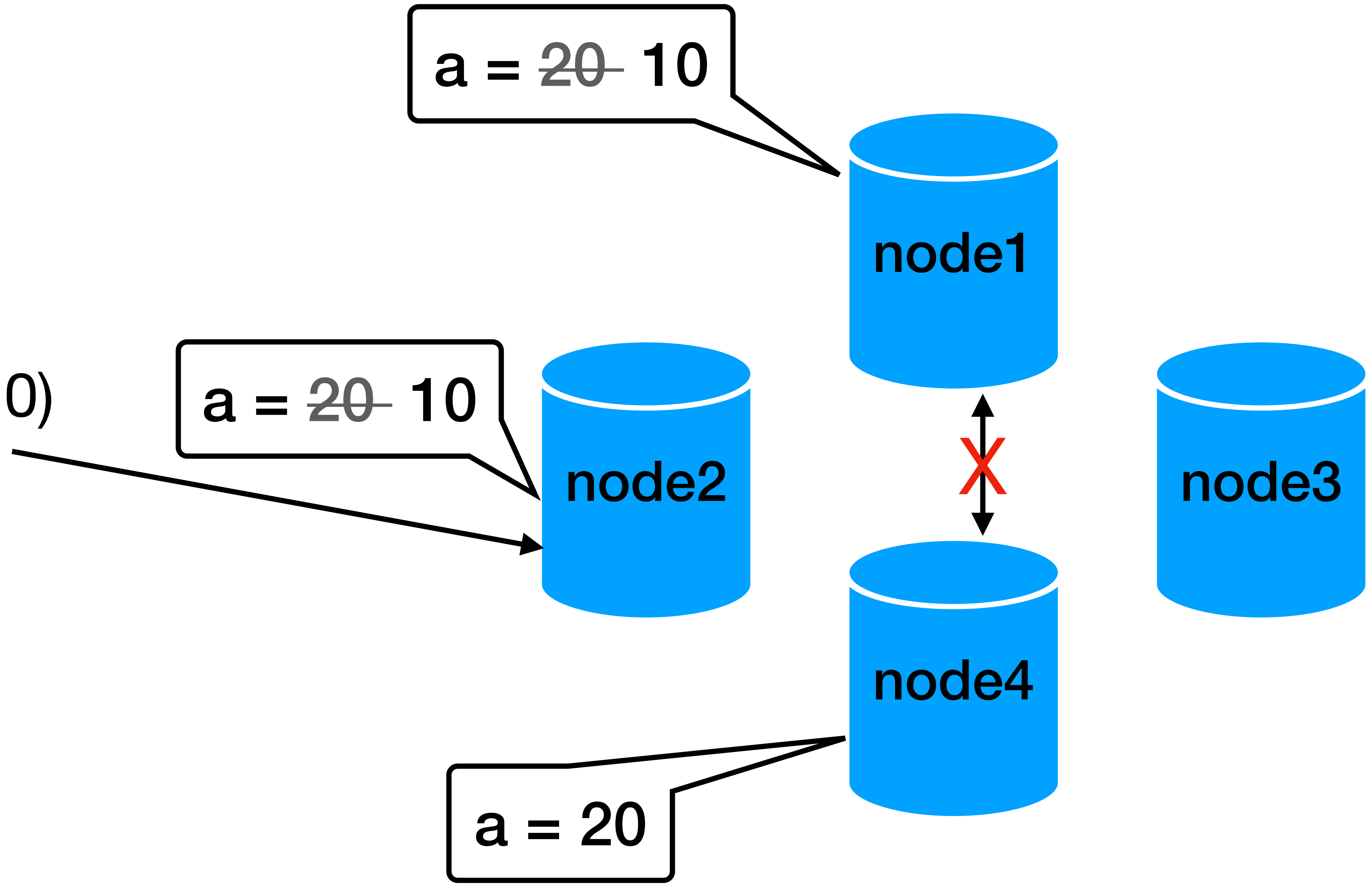
- Distributed database, set to performance (availability)
  updates other nodes asynchronously



10:00: a = 20

10:01: update a = 10

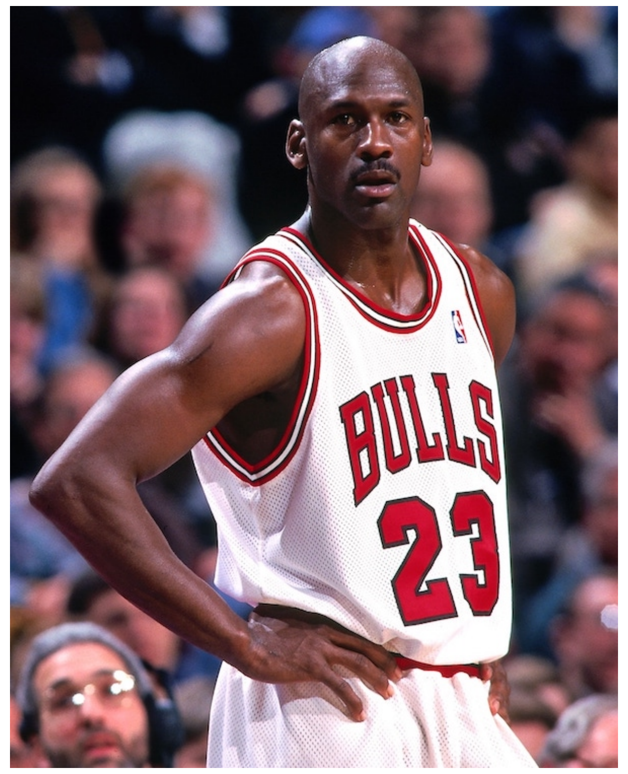
10:02: read a (value = 10)



* example for availability

Server side consistency - example 2

- Distributed database, set to performance (availability)
  updates other nodes asynchronously

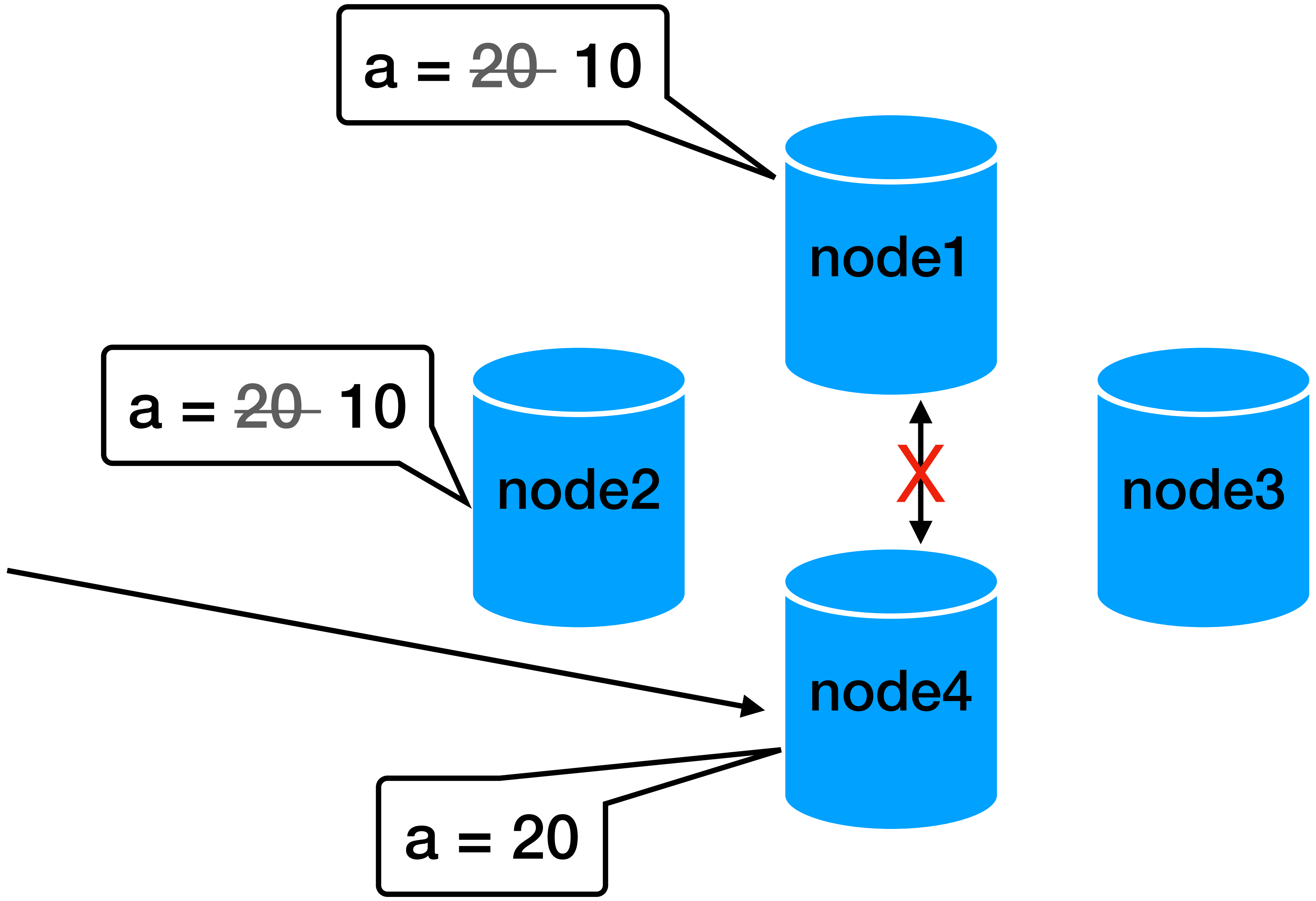


10:00: a = 20

10:01: update a = 10

10:02: read a (value = 10)

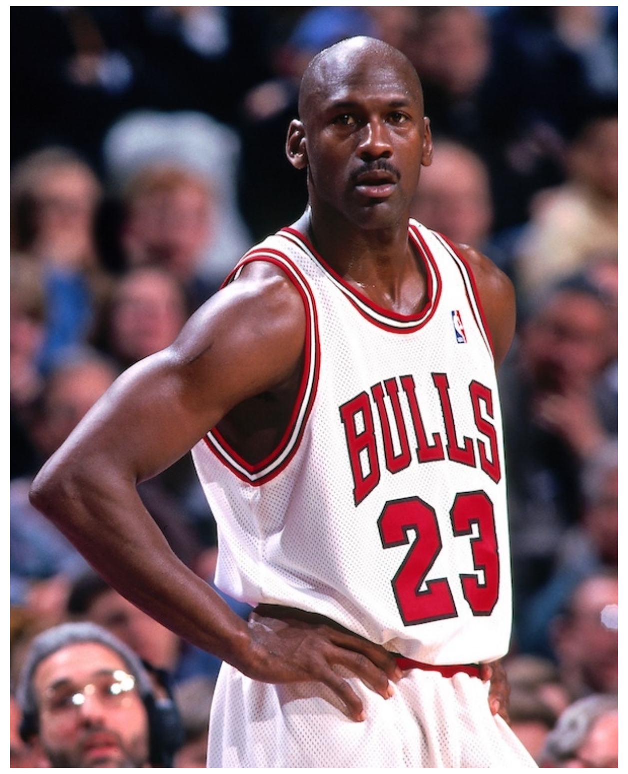
10:03: read a (value = 20)



* example for availability

Server side consistency - example 2

- Distributed database, set to performance (availability)
  updates other nodes asynchronously

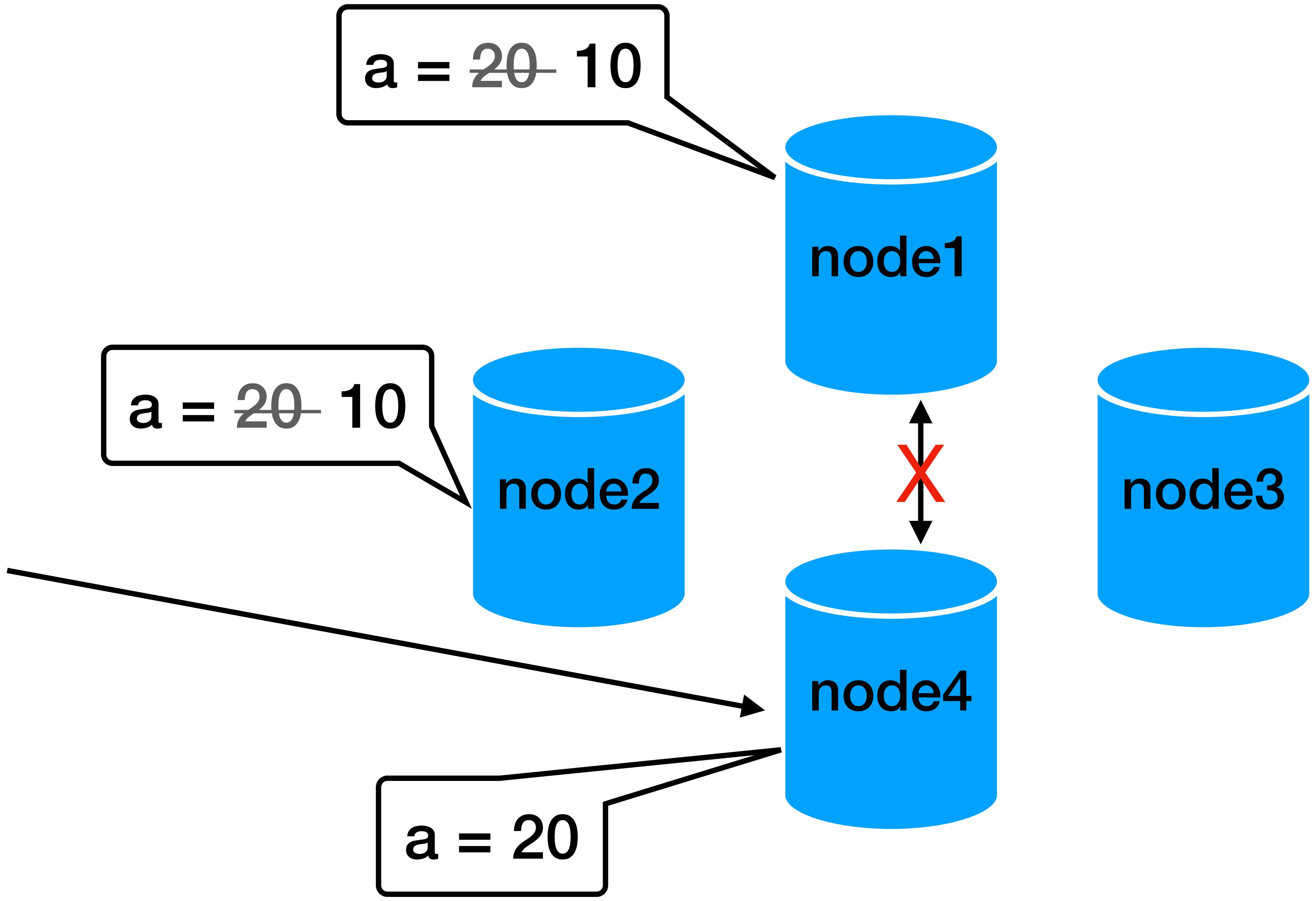


10:00: a = 20

10:01: update a = 10

10:02: read a (value = 10)

10:03: read a (value = 20)



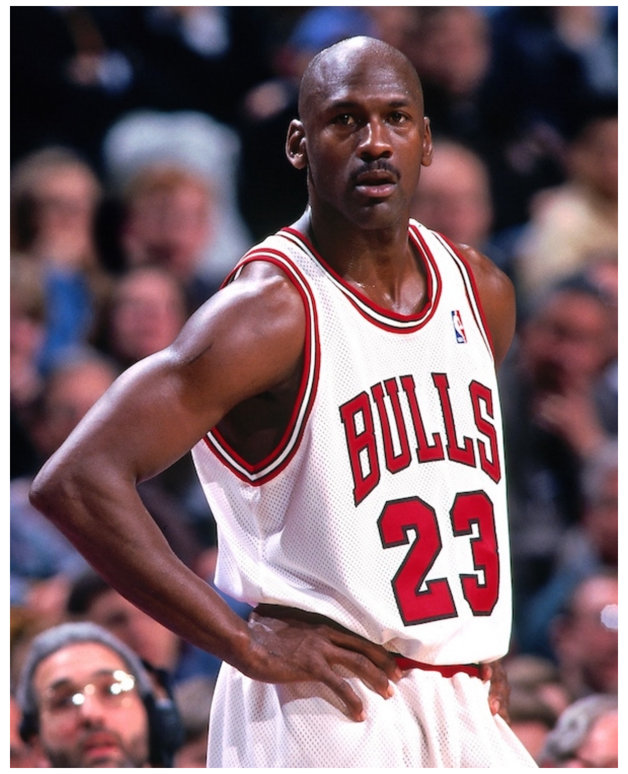
* example for availability

W (1) + R (1) <= N (3)

weak / eventual consistency

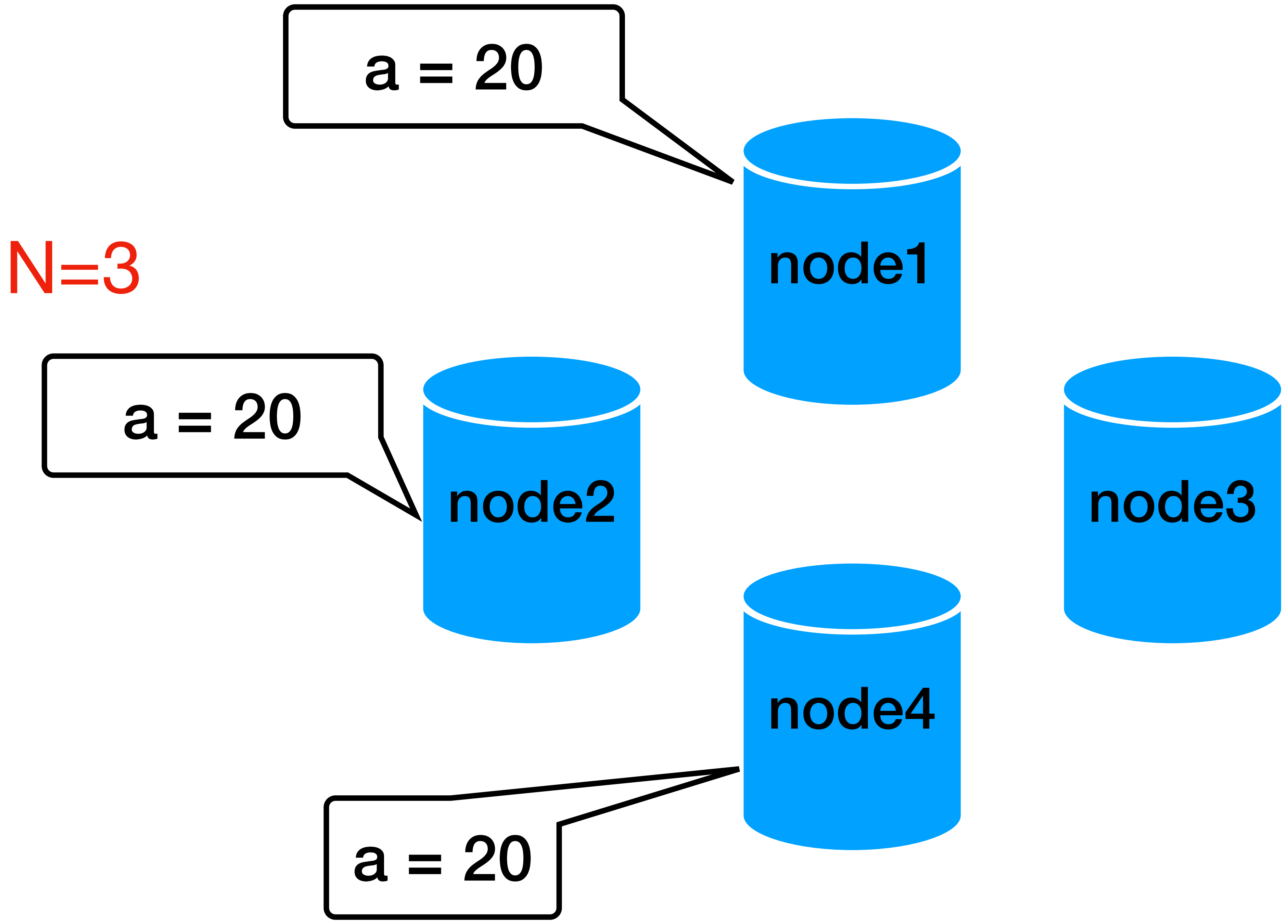
Server side consistency - example 3

- Distributed database, set to consistency
  updates & reads needs quorum ack



10:00: a = 20

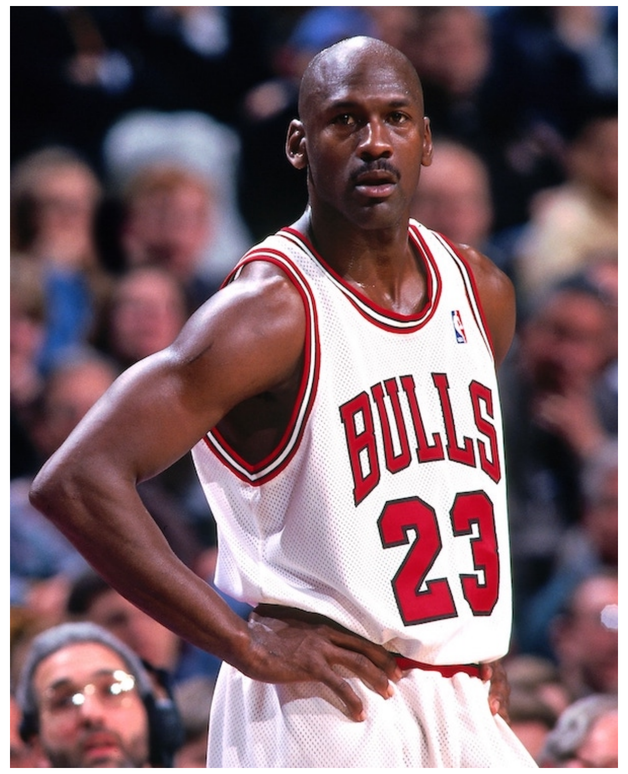
* example for consistency



Server side consistency - example 3

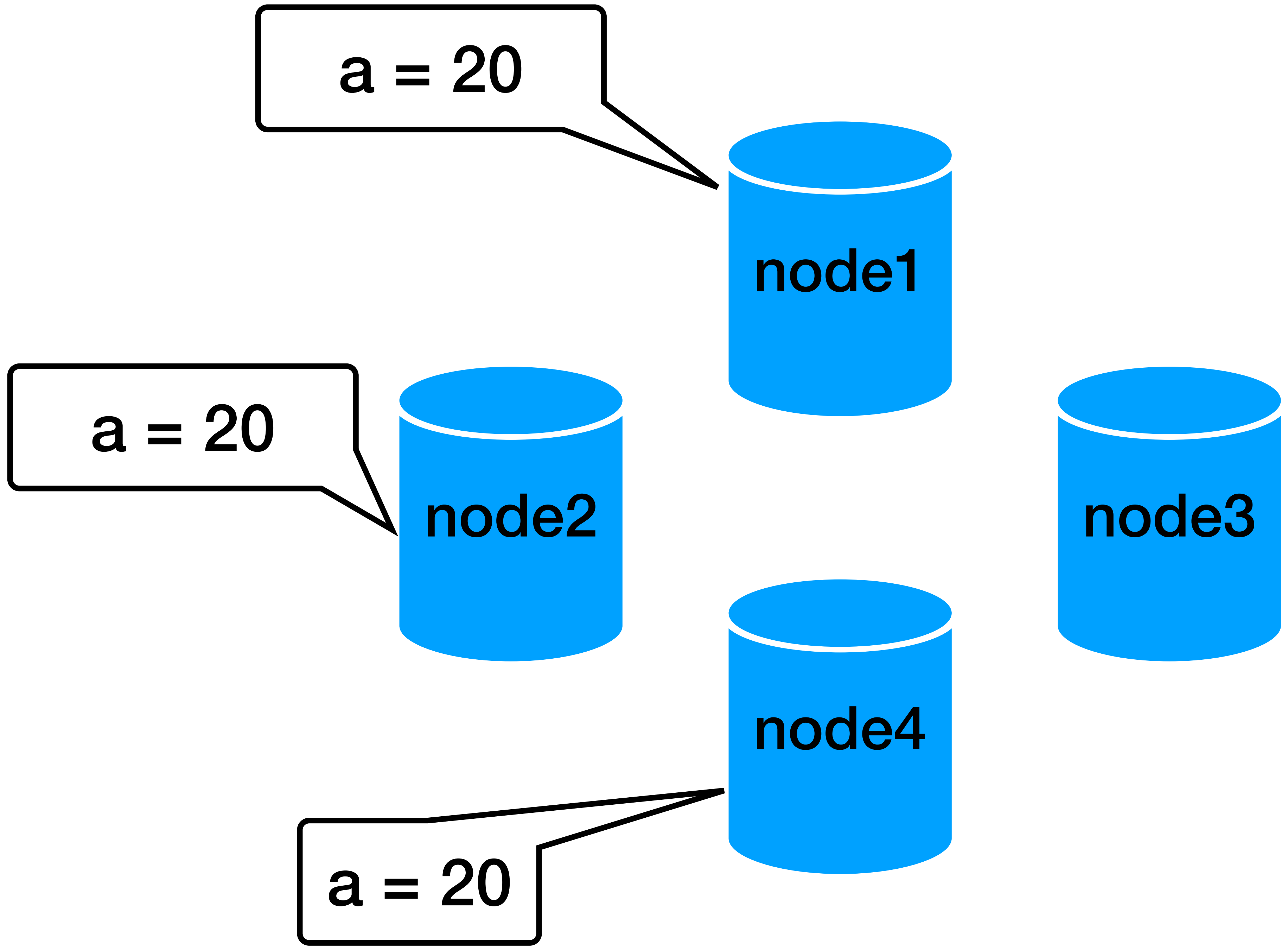
- Distributed database, set to consistency
  updates & reads needs quorum ack

What is quorum ack?



10:00: a = 20

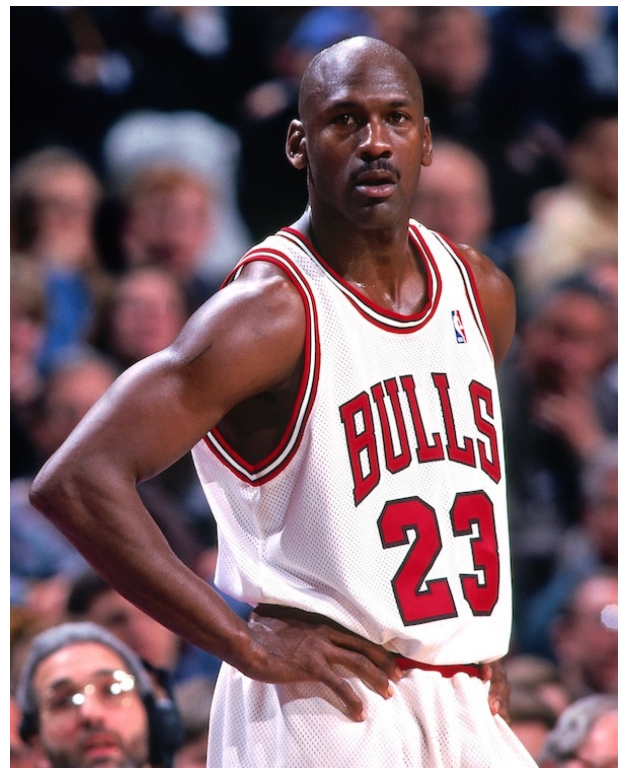
* example for consistency



Server side consistency - example 3

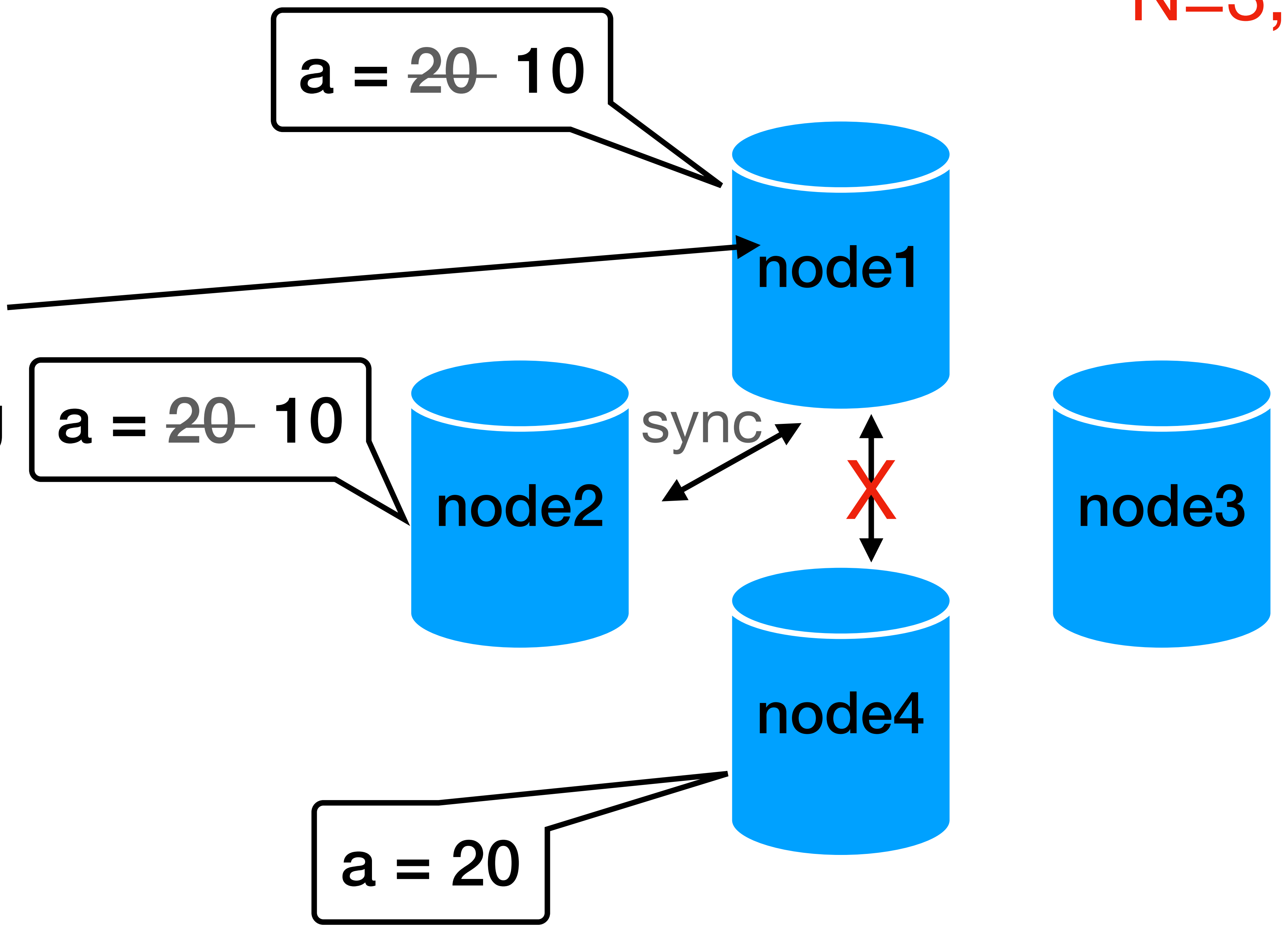
- Distributed database, set to consistency
  updates & reads needs quorum ack

N=3, W=2, R=2



10:00: a = 20

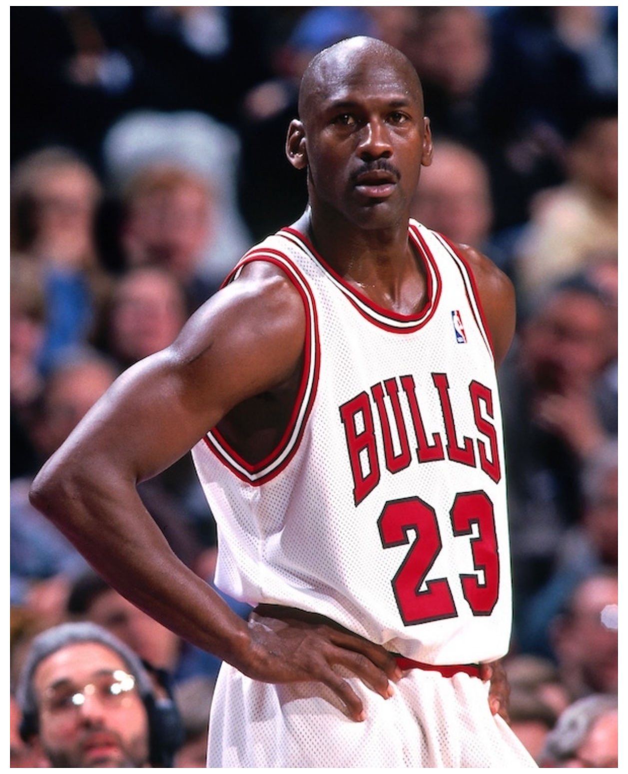
10:01: update node1 a = 10
—> node2 returned ack
node4 is not responding
—> return success



Server side consistency - example 3

- Distributed database, set to consistency
  updates & reads needs quorum ack

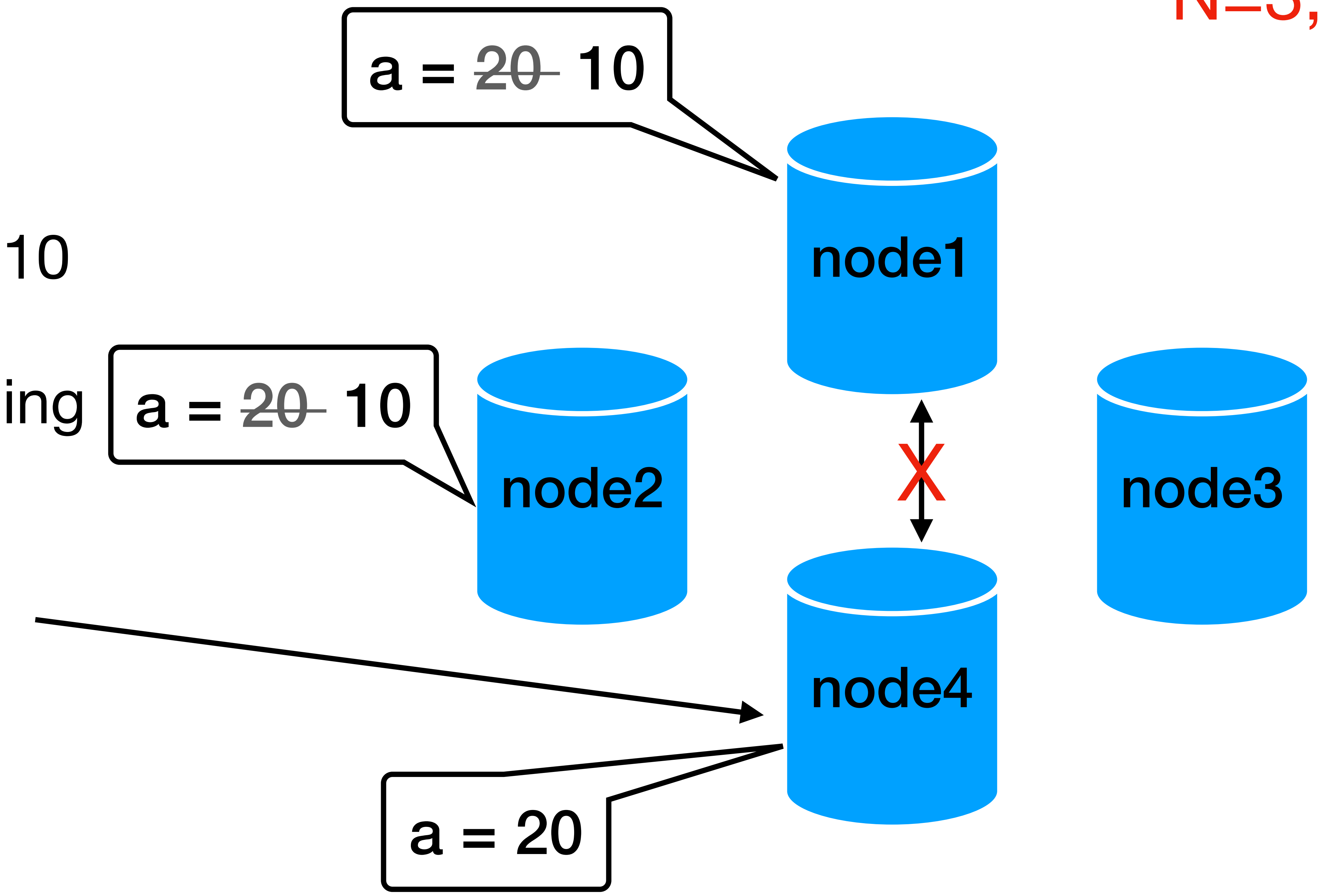
N=3, W=2, R=2



10:00: a = 20

10:01: update node1 a = 10
—> node2 returned ack
      node4 is not responding
—> return success

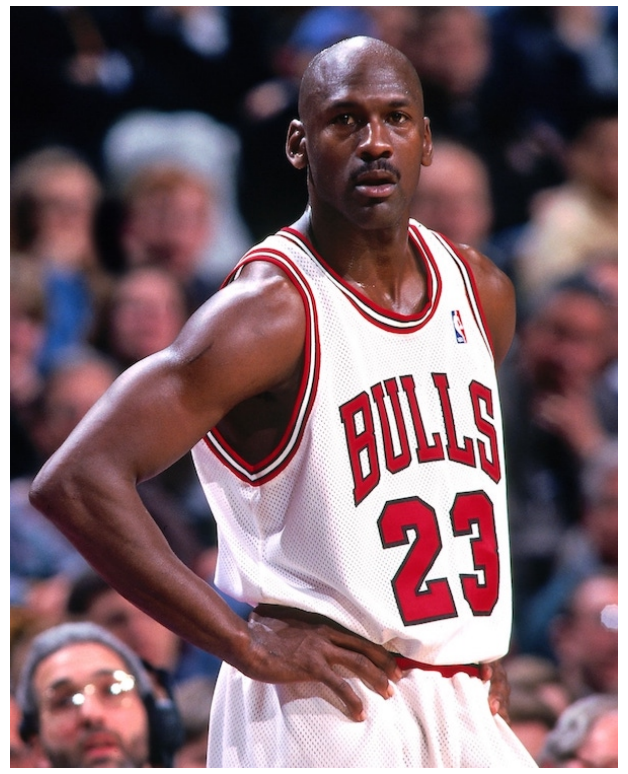
10:02: read node4 (a=20)



Server side consistency - example 3

- Distributed database, set to consistency
  updates & reads needs quorum ack

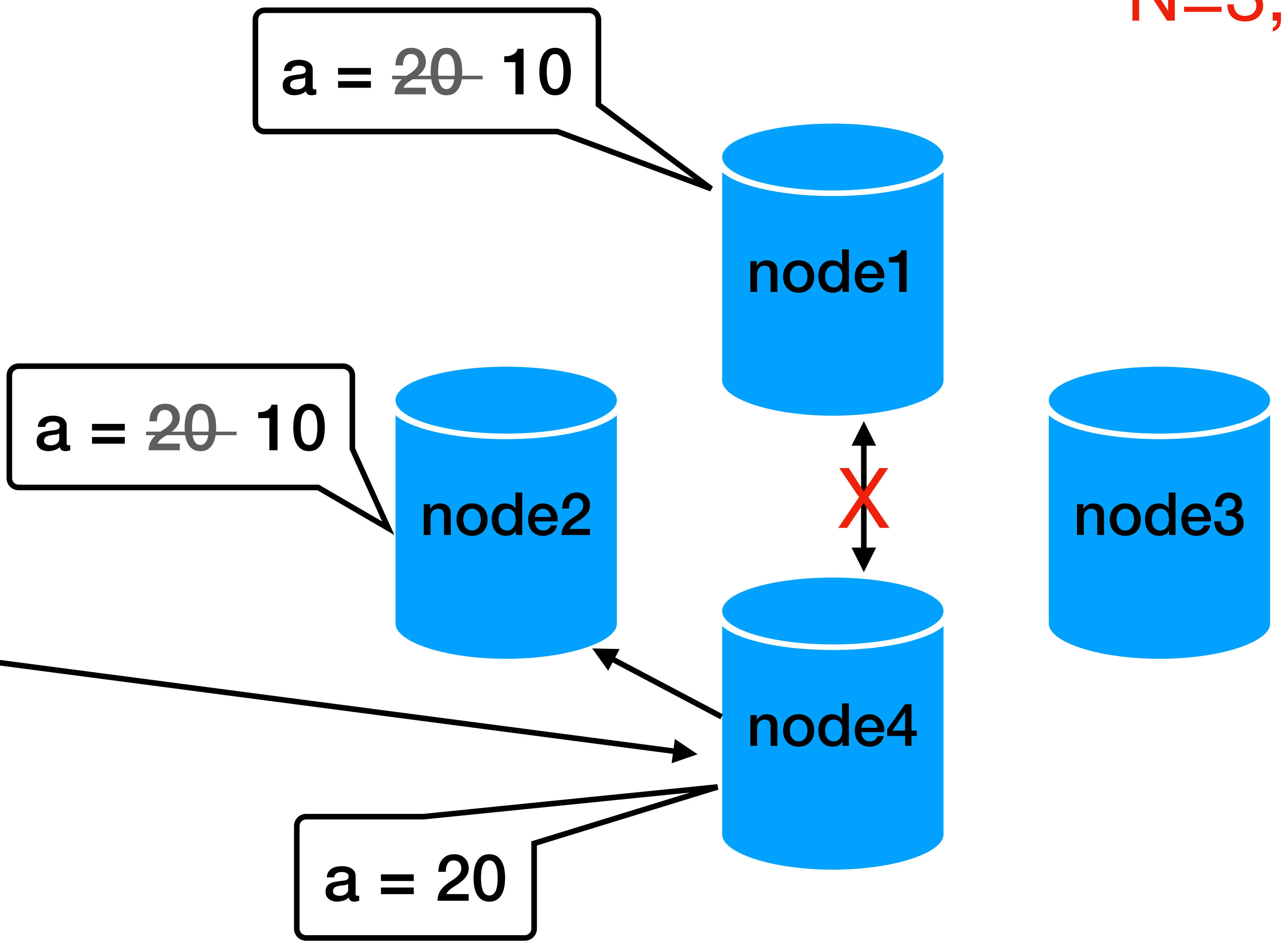
N=3, W=2, R=2



10:00: a = 20

10:01: update node1 a = 10
—> node2 returned ack
      node4 is not responding
—> return success

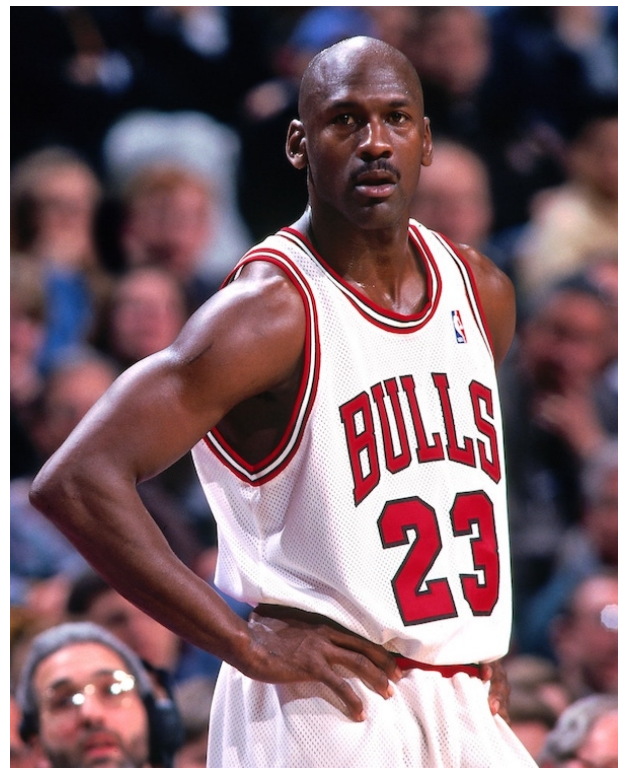
10:02: read node4 (a=20)
—> read node2 (a=10)



Server side consistency - example 3

- Distributed database, set to consistency
  updates & reads needs quorum ack

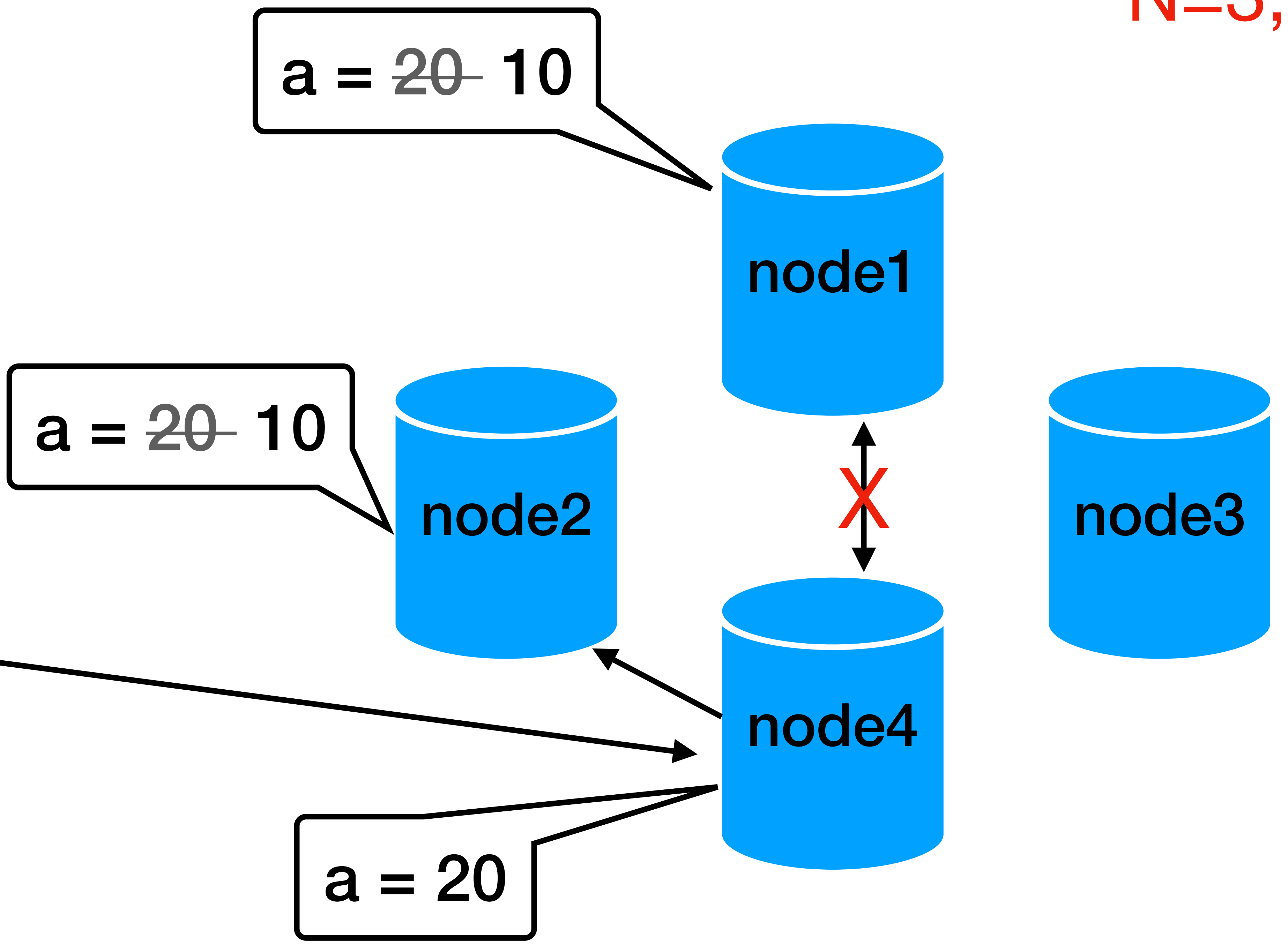
N=3, W=2, R=2



10:00: a = 20

10:01: update node1 a = 10
—> node2 returned ack
     node4 is not responding
—> return success

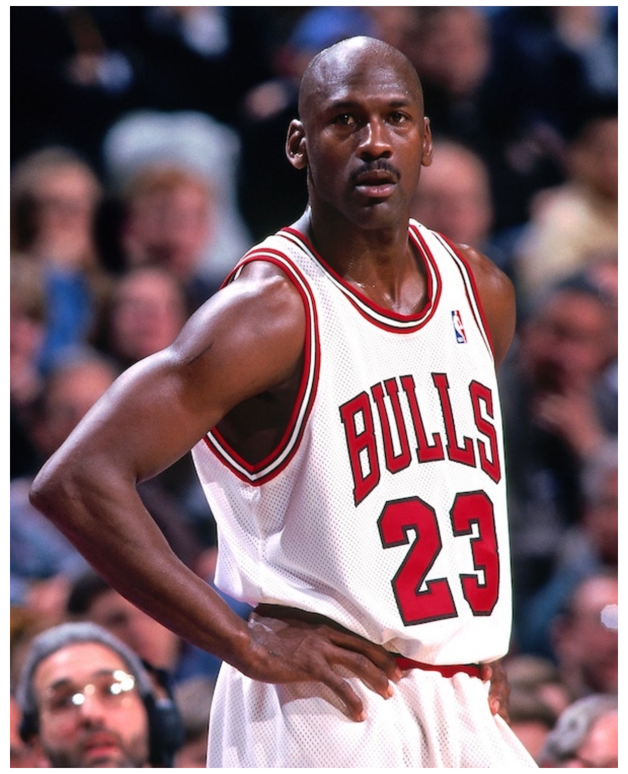
10:02: read node4 (a=20)
—> read node2 (a=10)
—> there is NO quorum



Server side consistency - example 3

- Distributed database, set to consistency
  updates & reads needs quorum ack

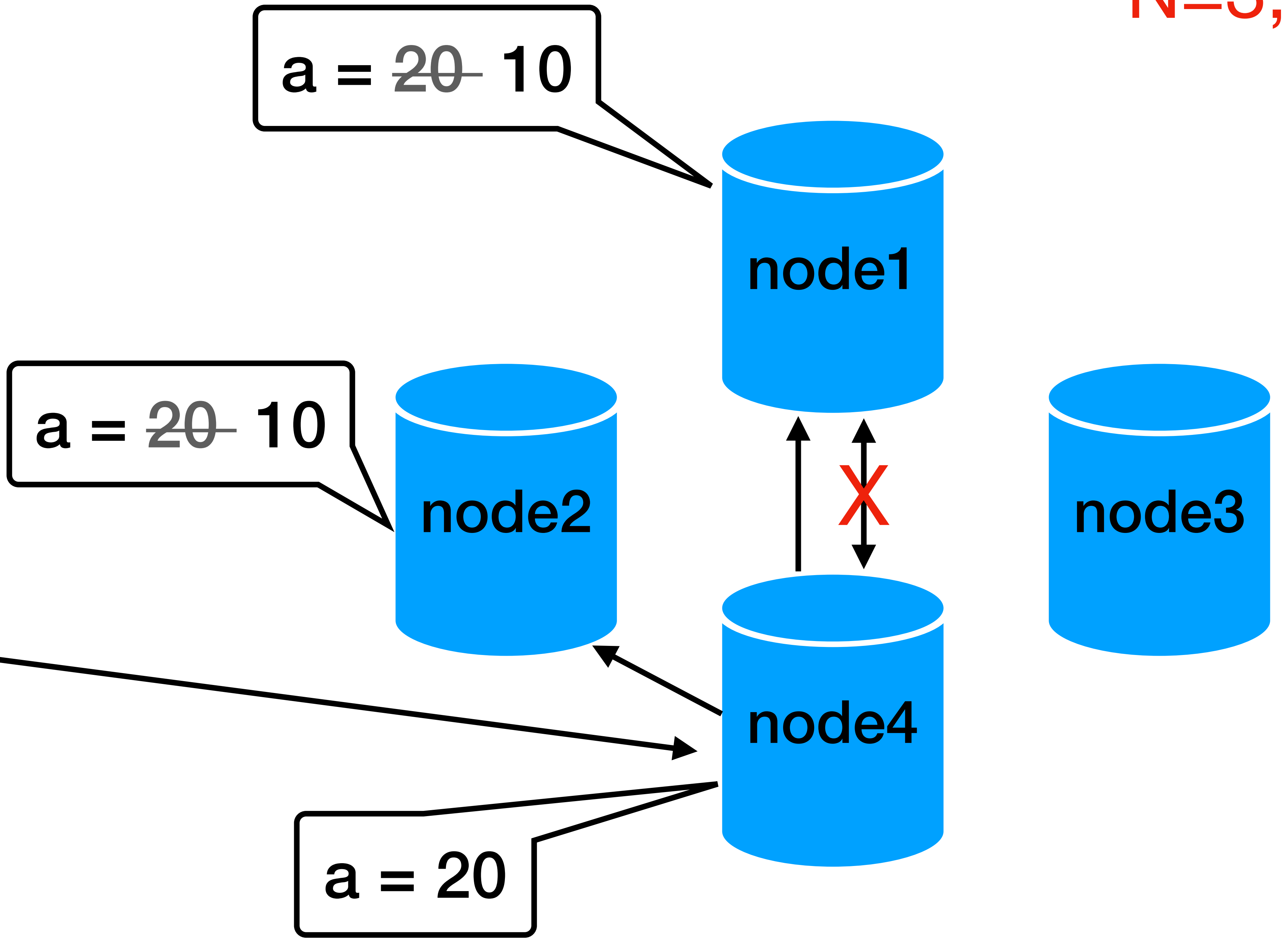
N=3, W=2, R=2



10:00: a = 20

10:01: update node1 a = 10
—> node2 returned ack
node4 is not responding
—> return success

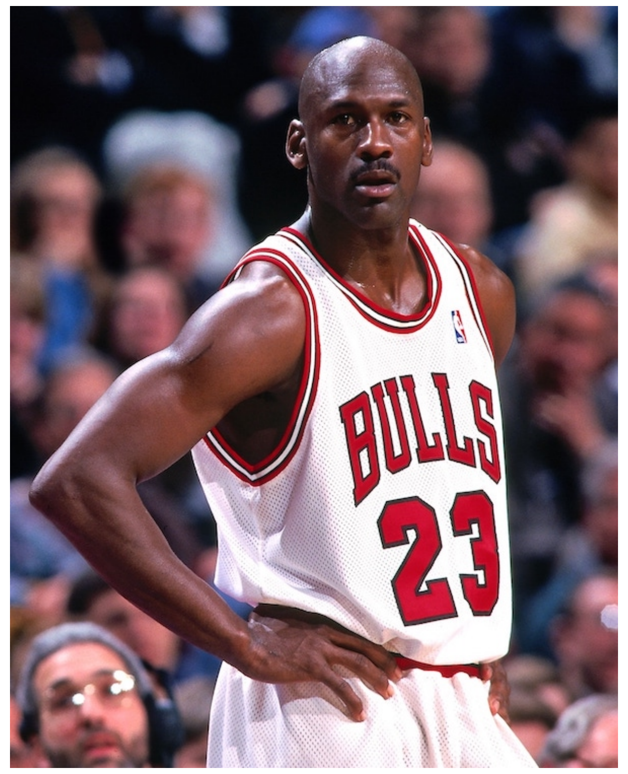
10:02: read node4 (a=20)
—> read node2 (a=10)
—> there is NO quorum
—> in node1 a=10



Server side consistency - example 3

- Distributed database, set to consistency
  updates & reads needs quorum ack

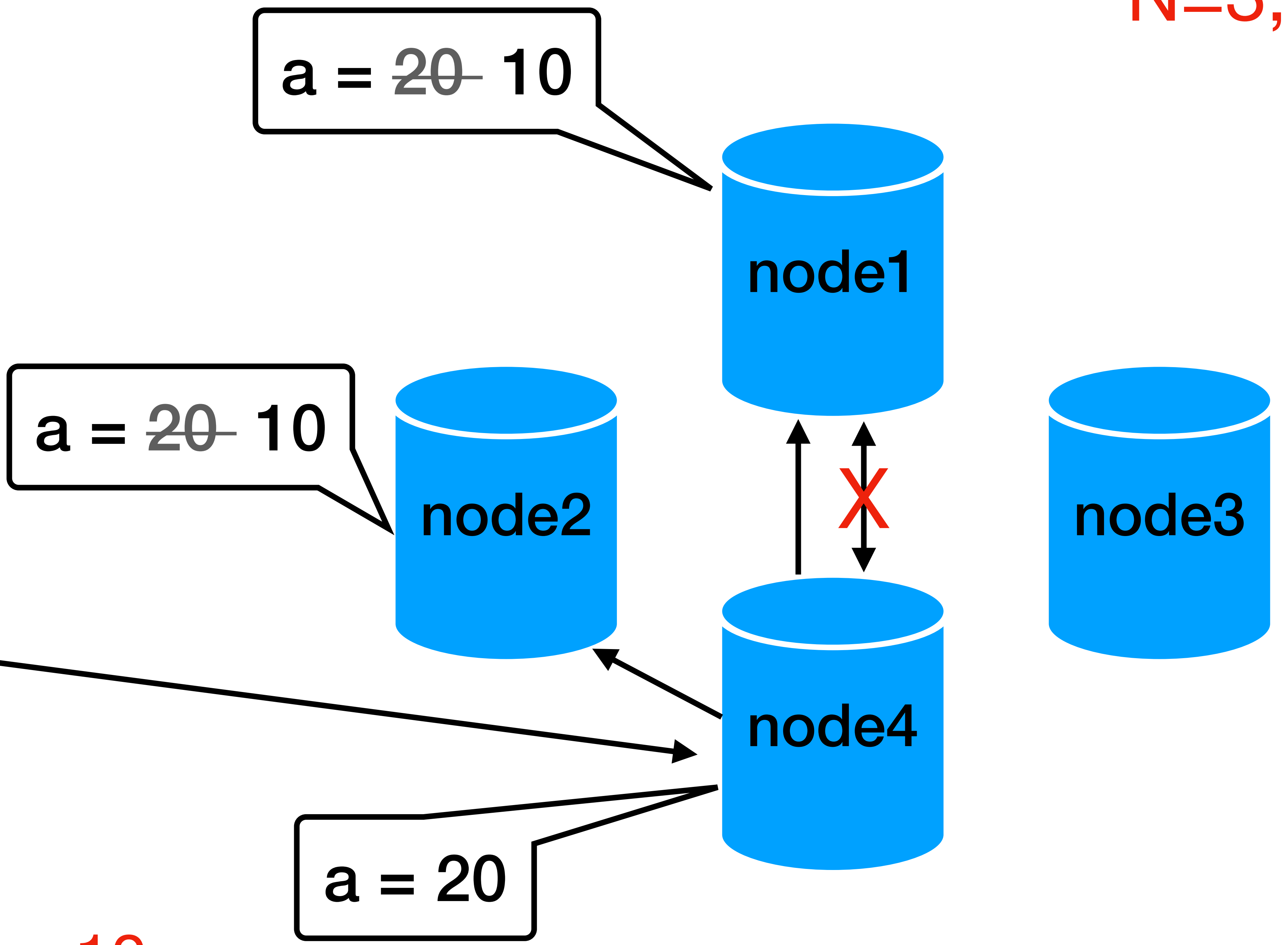
N=3, W=2, R=2



10:00: a = 20

10:01: update node1 a = 10
—> node2 returned ack
node4 is not responding
—> return success

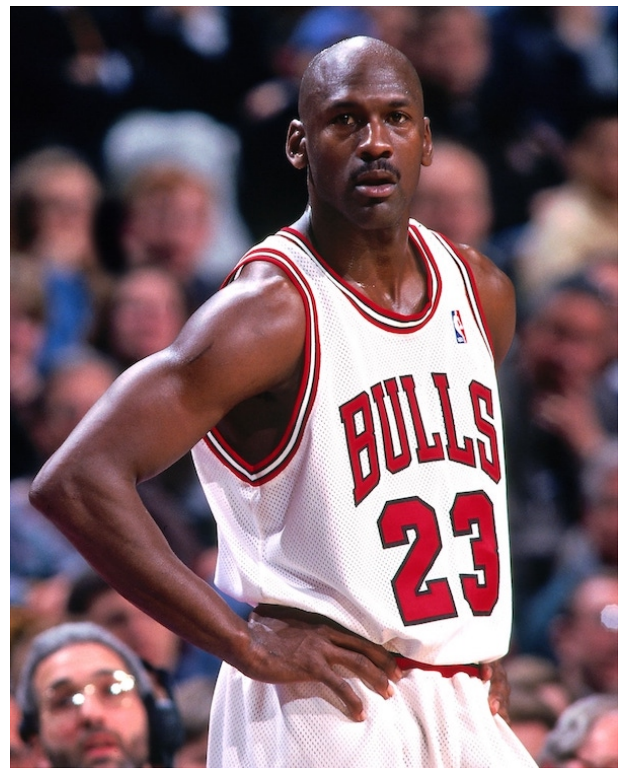
10:02: read node4 (a=20)
—> read node2 (a=10)
—> there is NO quorum
—> in node1 a=10
—> there is a quorum, return a=10



Server side consistency - example 3

- Distributed database, set to consistency
  updates & reads needs quorum ack

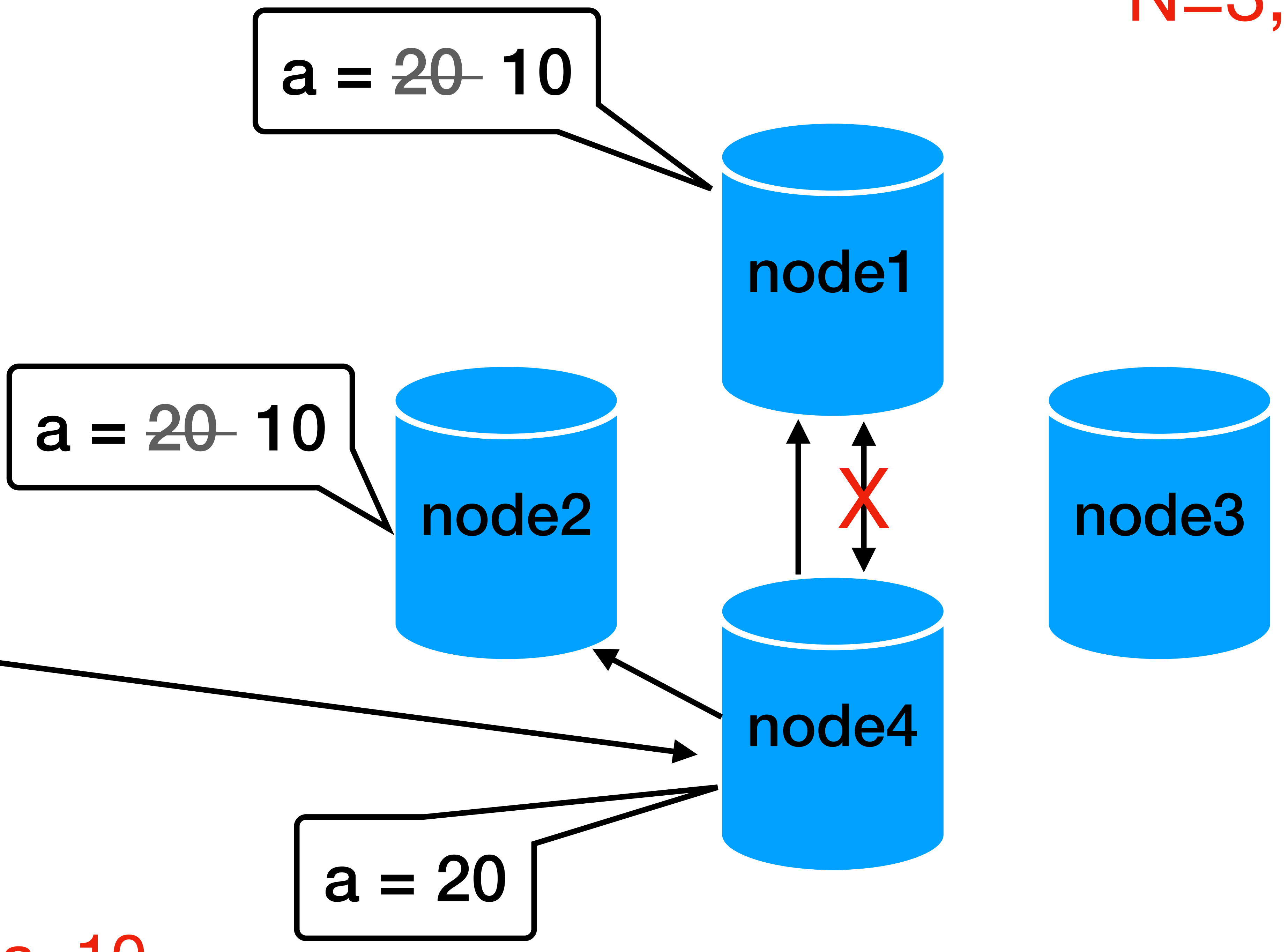
N=3, W=2, R=2



10:00: a = 20

10:01: update node1 a = 10
—> node2 returned ack
    node4 is not responding
—> return success

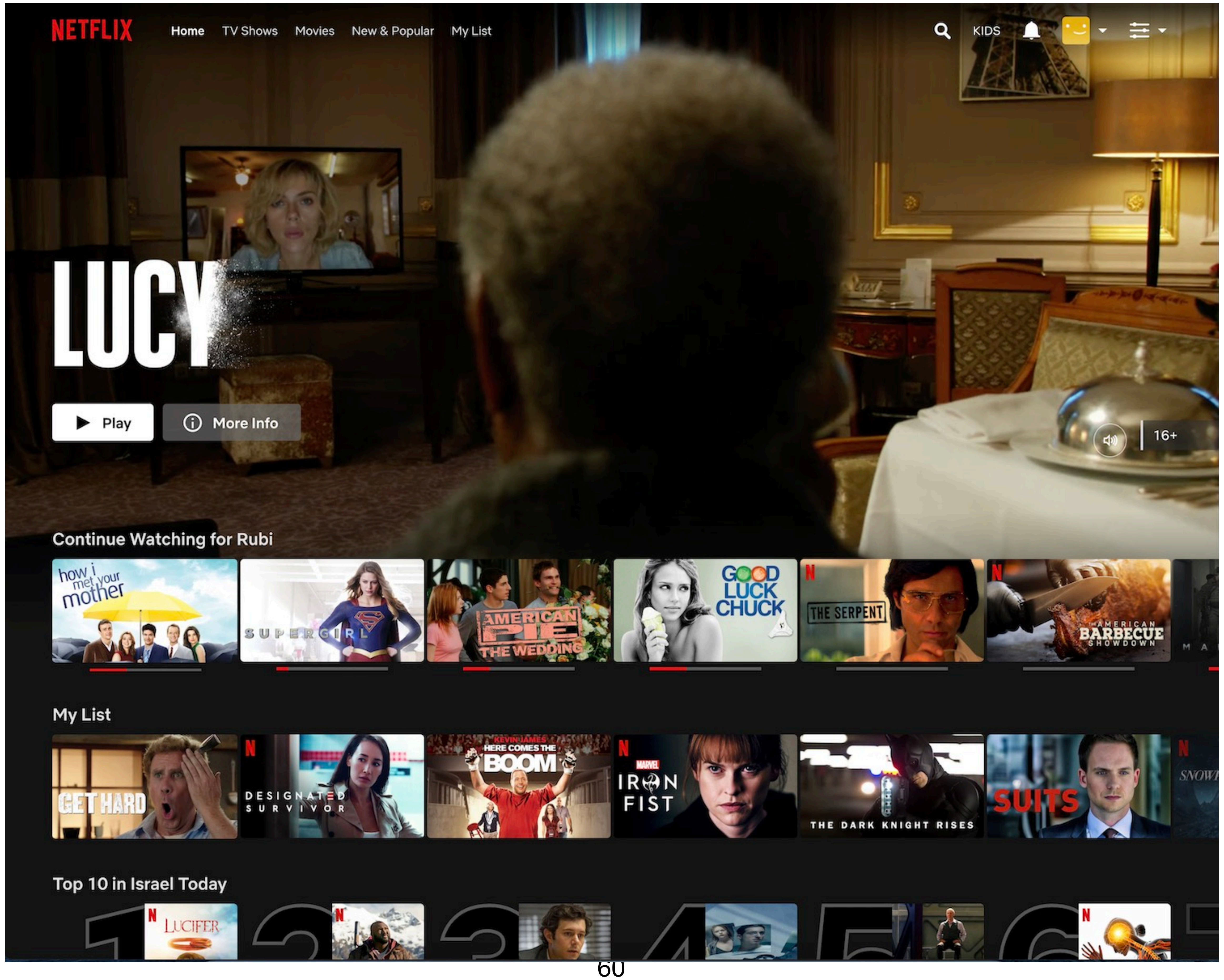
10:02: read node4 (a=20)
—> read node2 (a=10)
—> there is NO quorum
—> in node1 a=10
—> there is a quorum, return a=10



W (2) + R (2) > N (3)
strong consistency

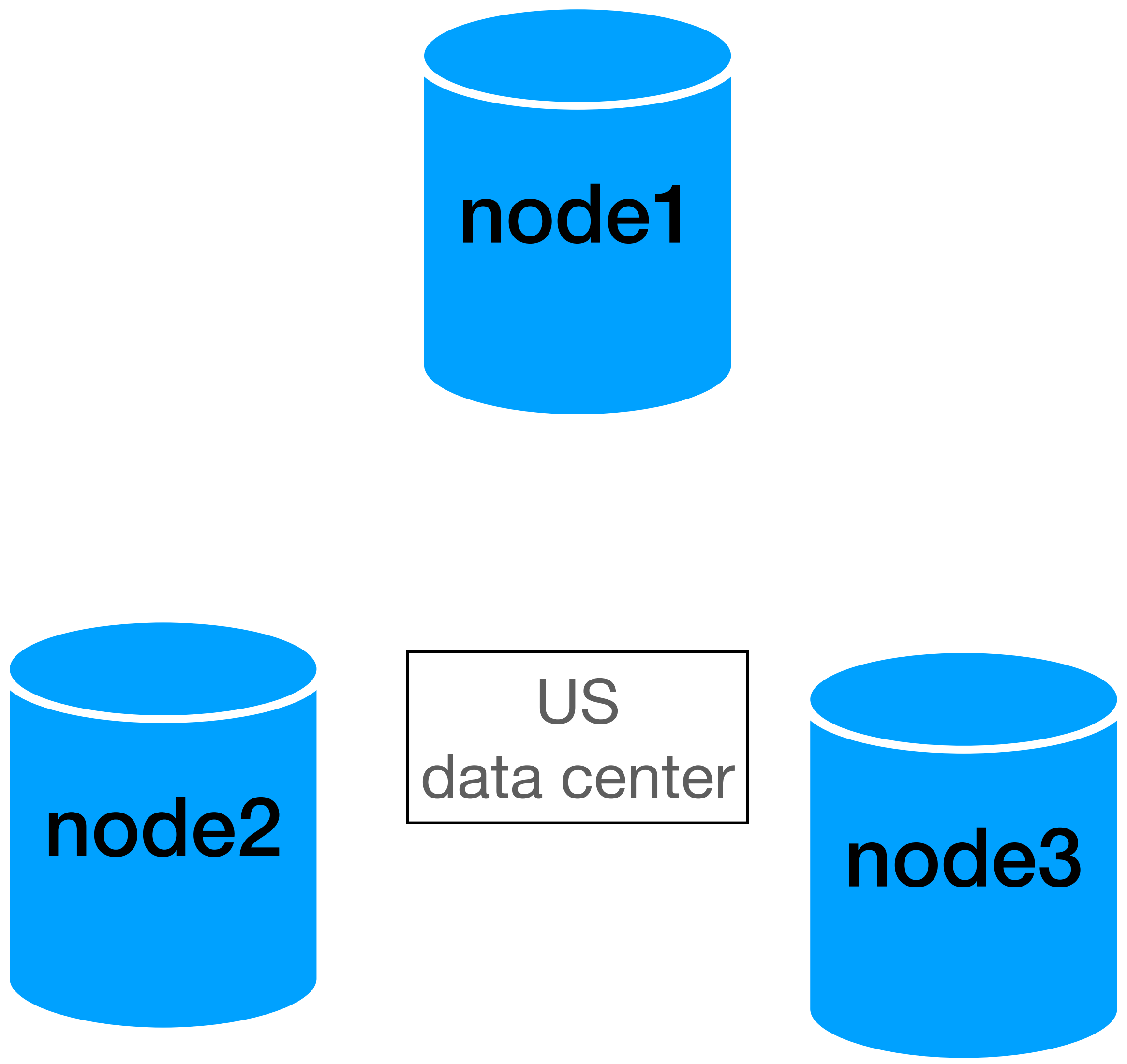
Server side consistency - example 4

- Distributed database, multi data center



Server side consistency - example 4

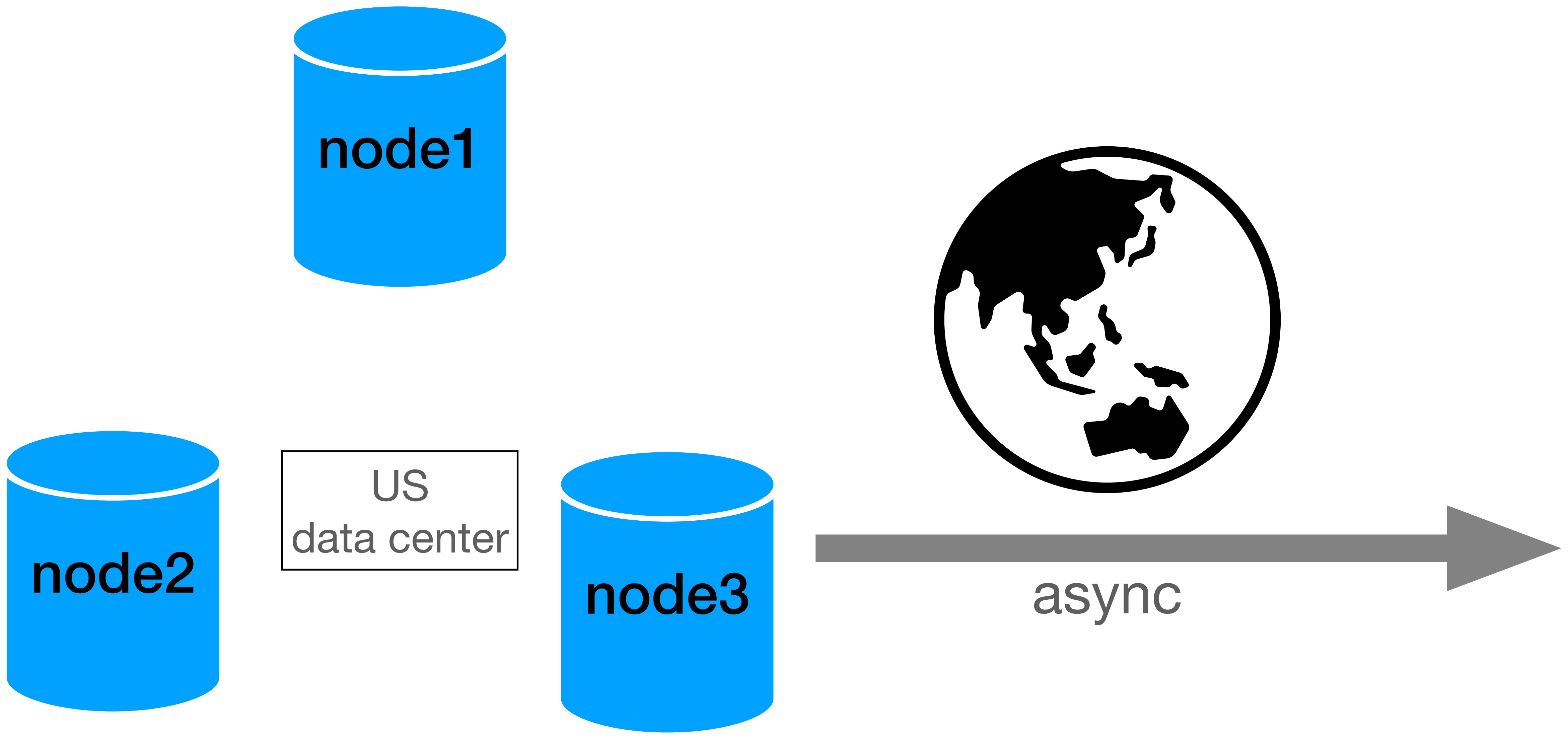
- Distributed database, mixed consistency
  updates needs quorum ack in the same datacenter
  single ack from remote datacenter



W (2) + R (2) > N (3)
strong consistency
within a local data center

Server side consistency - example 4

- Distributed database, mixed consistency
  updates needs <u>quorum ack in the same datacenter</u>
  single ack from remote datacenter



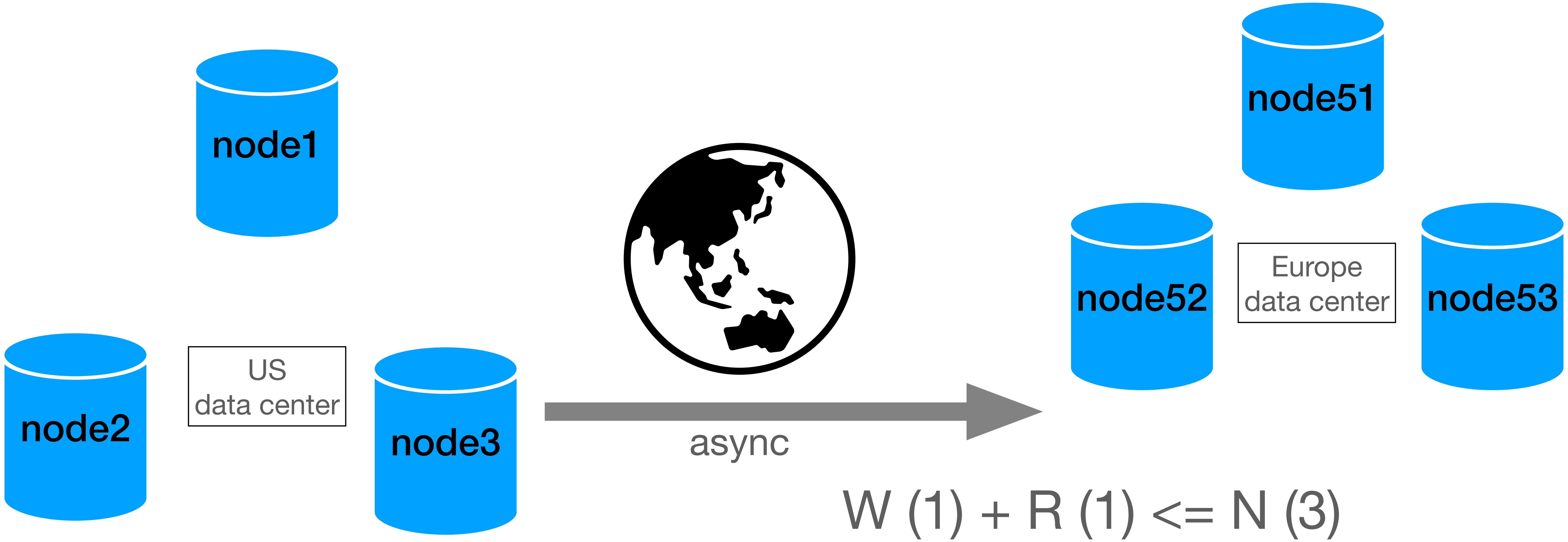
W (2) + R (2) > N (3)

strong consistency
within a local data center

Server side consistency - example 4

- Distributed database, mixed consistency

updates needs <u>quorum ack in the same datacenter</u>
single ack from remote datacenter



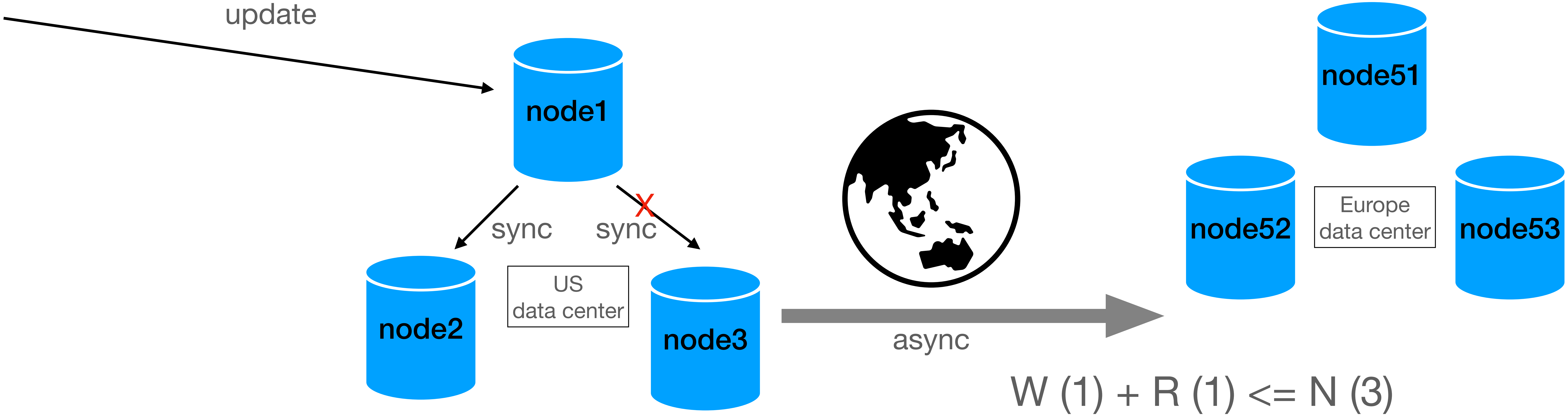
W (2) + R (2) > N (3)
strong consistency
within a local data center

W (1) + R (1) <= N (3)
eventual consistency
across data centers

Server side consistency - example 4

- Distributed database, mixed consistency

updates needs <u>quorum ack in the same datacenter</u>
single ack from remote datacenter

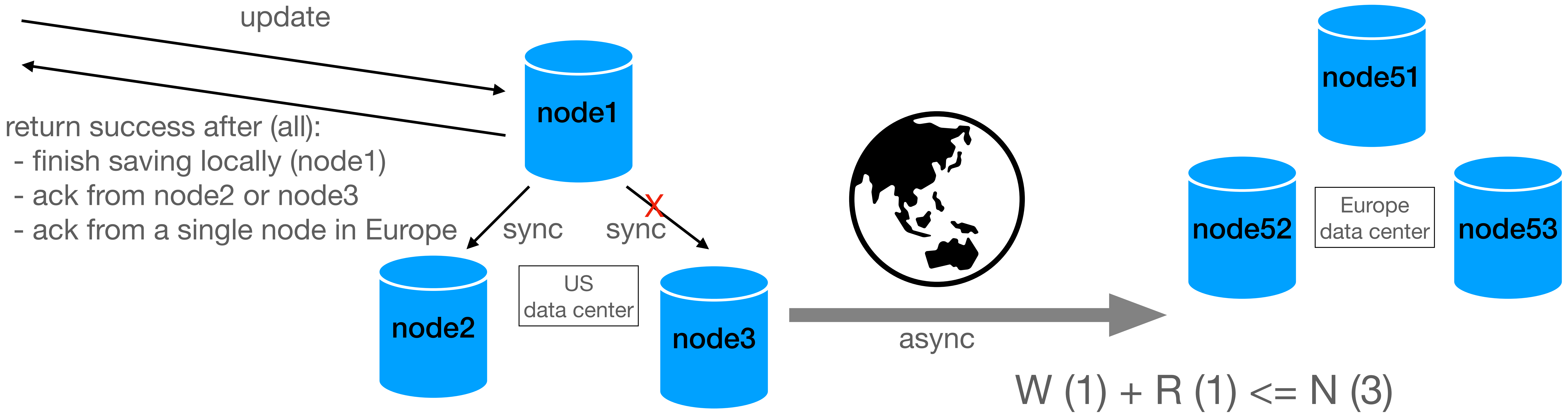


W (2) + R (2) > N (3)
strong consistency
within a local data center

W (1) + R (1) <= N (3)
eventual consistency
across data centers

Server side consistency - example 4

- Distributed database, mixed consistency
  updates needs <u>quorum ack in the same datacenter</u>
  single ack from remote datacenter



W (2) + R (2) > N (3)
strong consistency
within a local data center

W (1) + R (1) <= N (3)
eventual consistency
across data centers

Summary - CAP Theorem

- No distributed system is safe from network failures.
  —> we need to choose between CP and AP

Summary - CAP Theorem

- No distributed system is safe from network failures.
  —> we need to choose between CP and AP
- If a node is down/unreachable we can:
  - cancel the operation (CP)
  - Return result with (maybe) inconsistency (AP)
- Multi data center adds more options