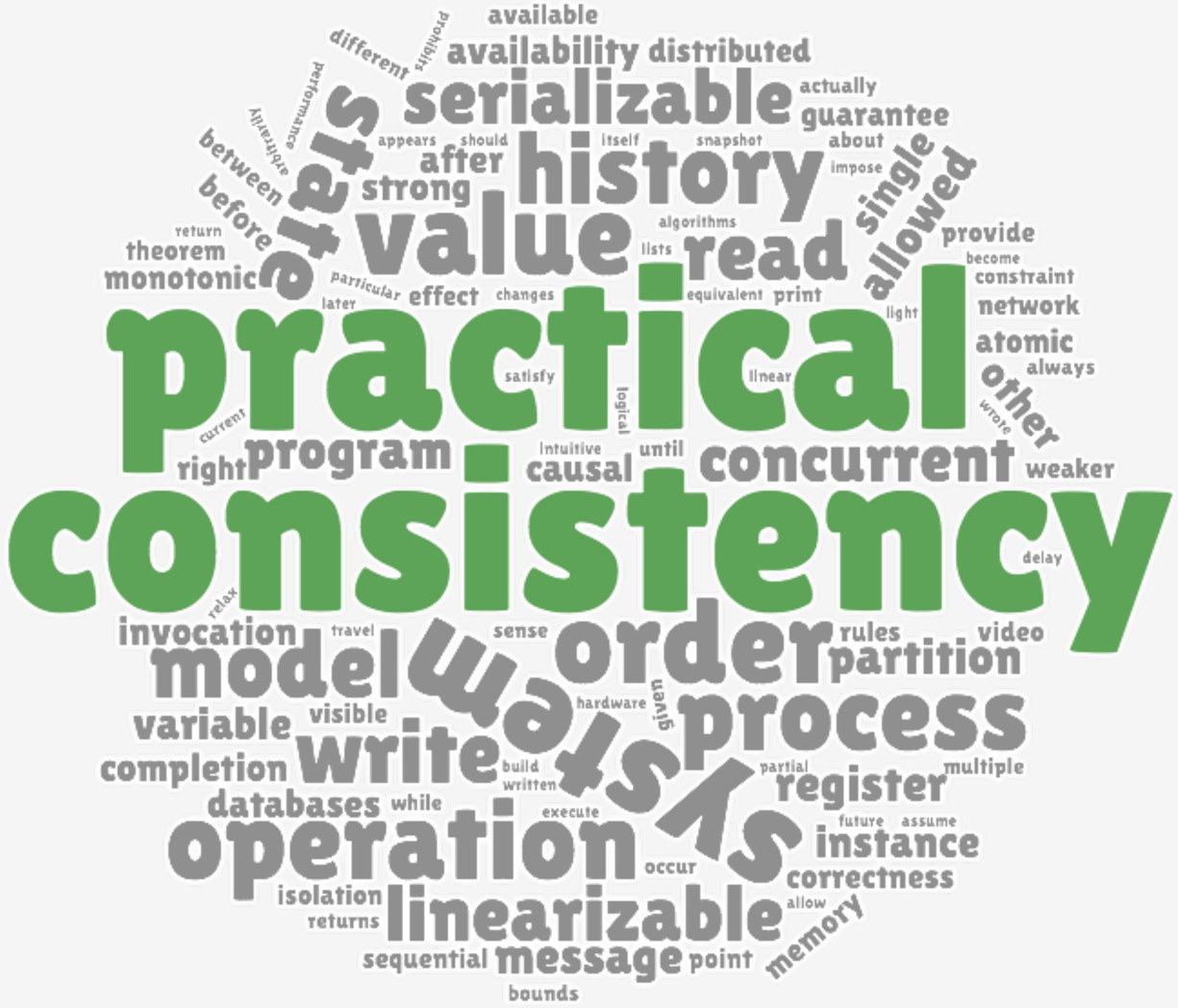


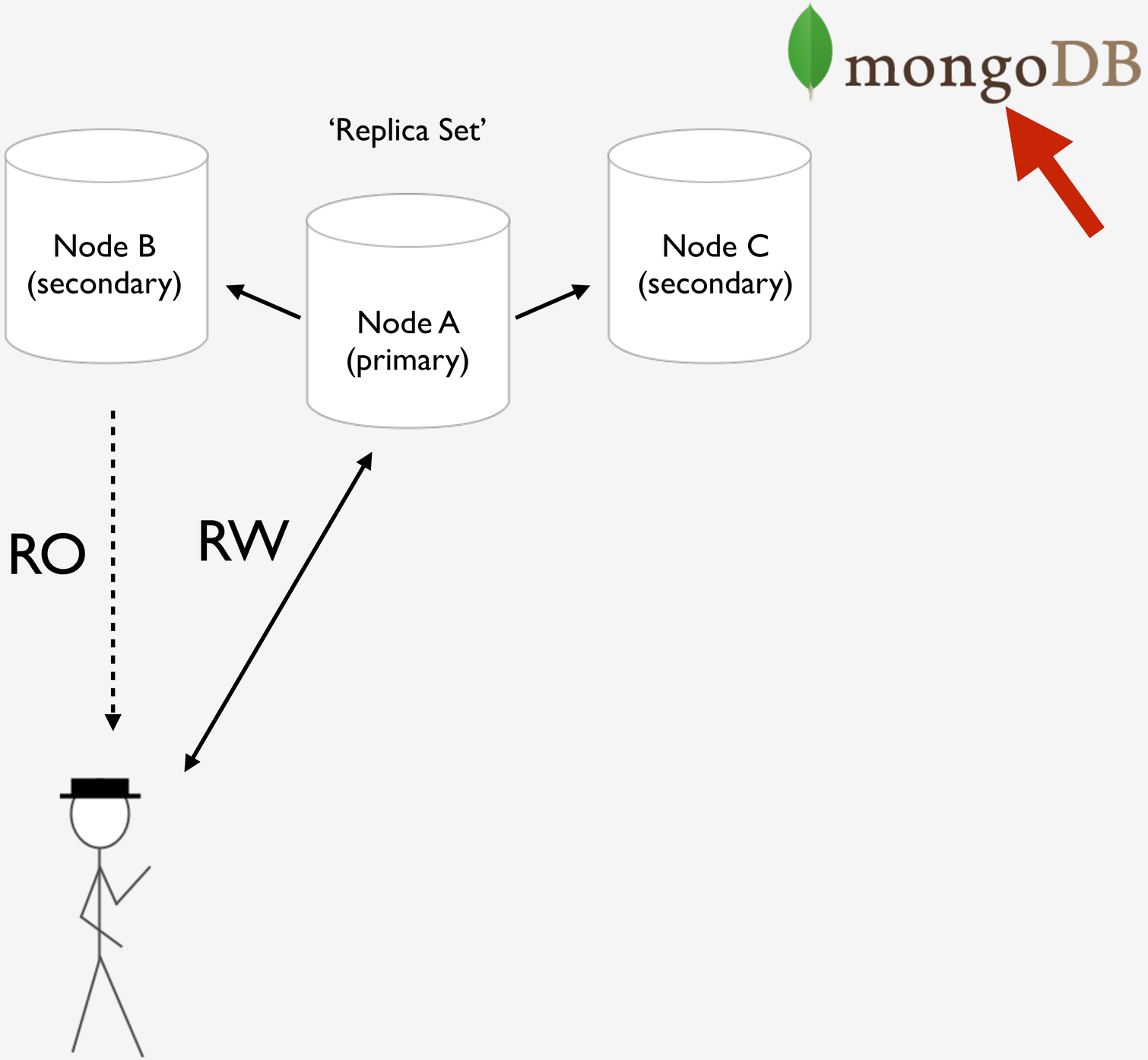
David Golden

mongoDB

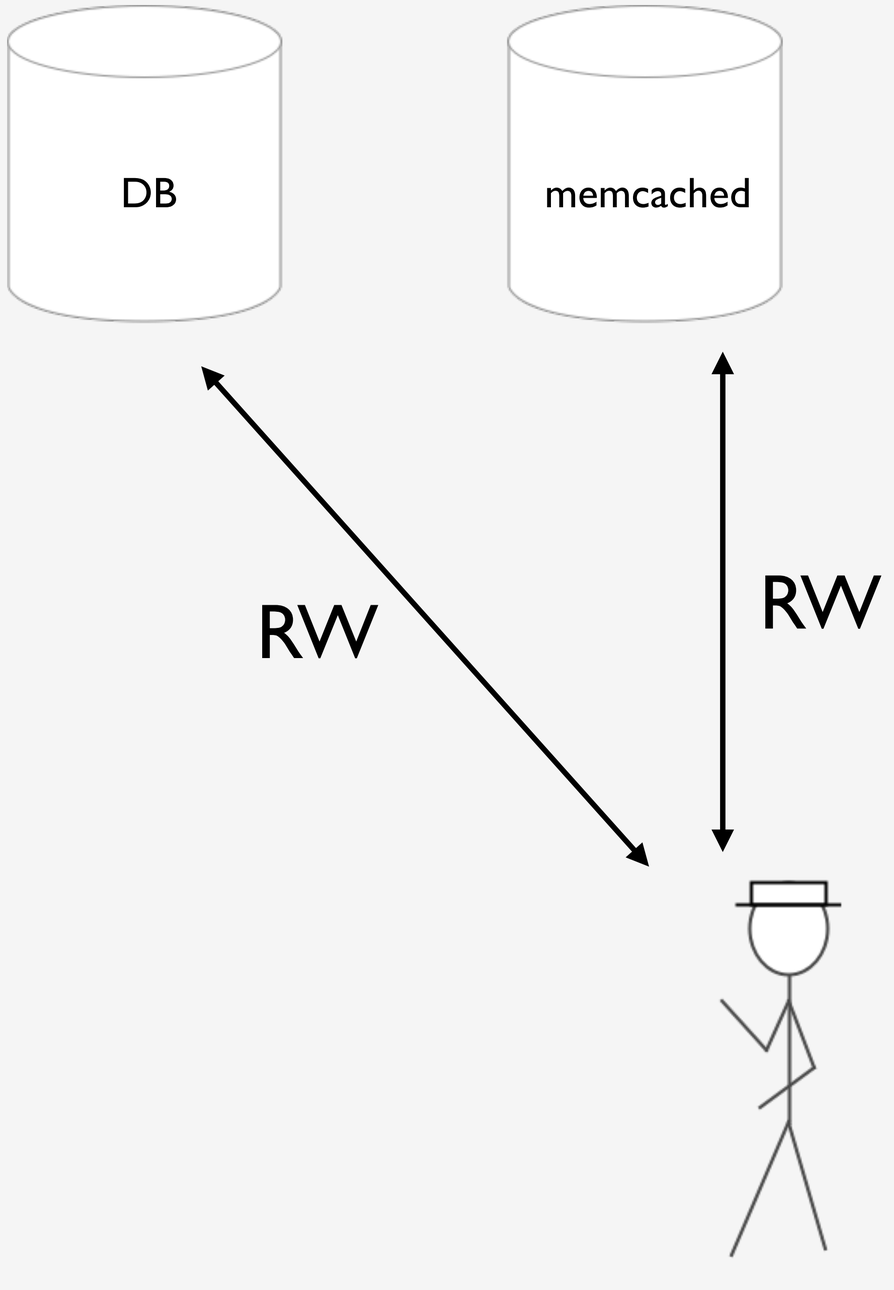
YAPC::NA 2016

What is a distributed system?

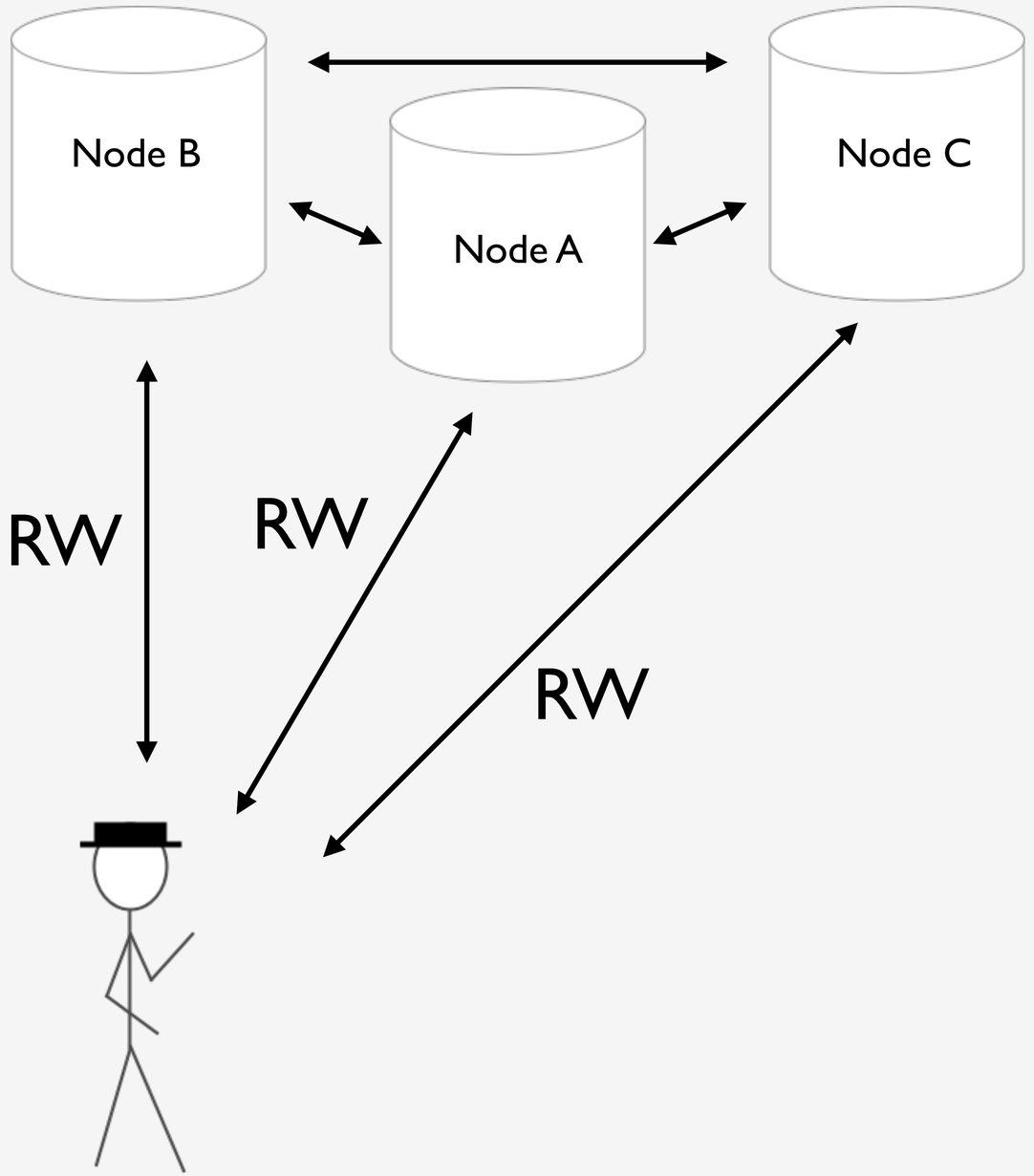
Data processing spread over time & space


mongoDB
'Replica Set'
Node B
(secondary)
Node A
(primary)
Node C
(secondary)
RO
RW

memcached



etcd



Why use a distributed system?

Scale

Scale

Performance

Scale

Performance

Redundancy

How do you break a distributed system?


ICH BIN EIN BERLINER

Crash

Crash

Packet loss

Crash

Packet loss

Garbage collection

Crash

Packet loss

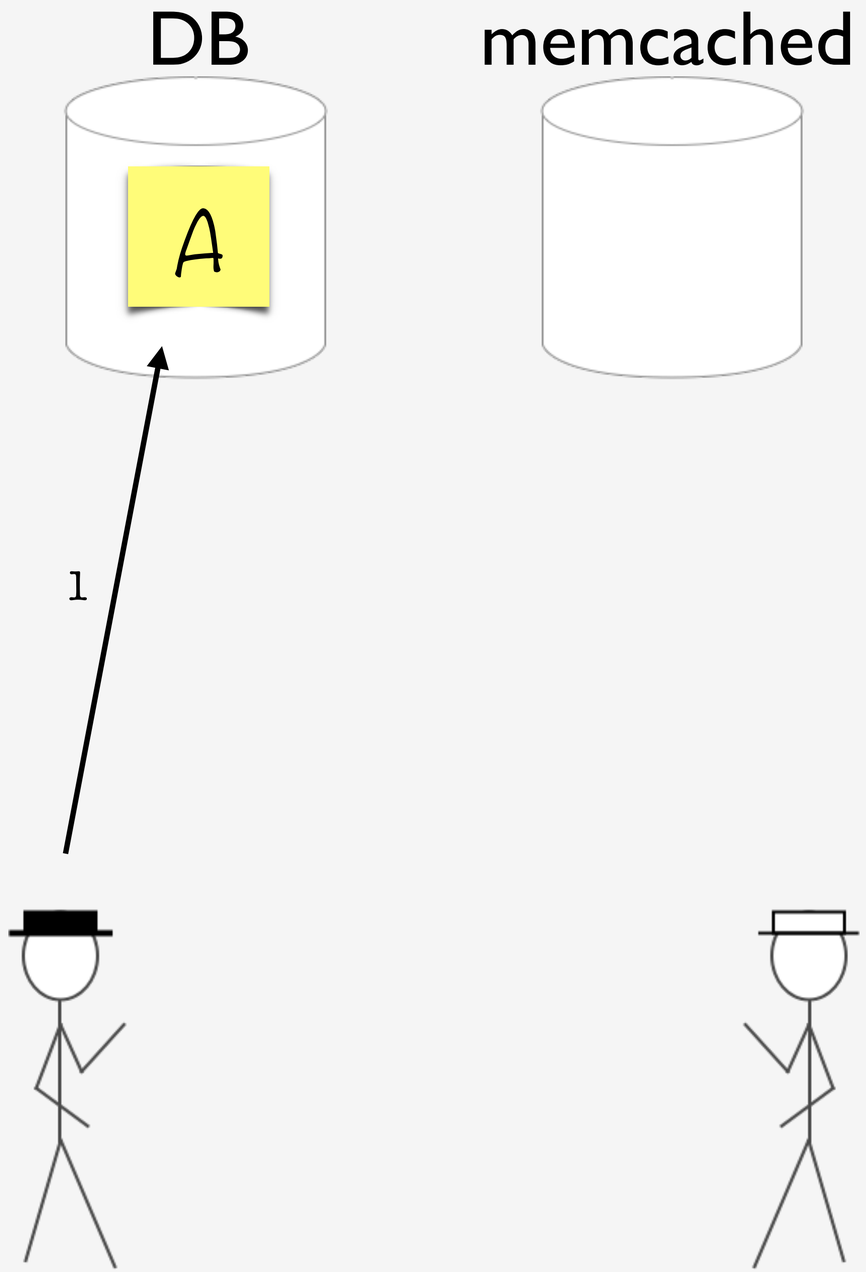
Garbage collection

Process swapped out

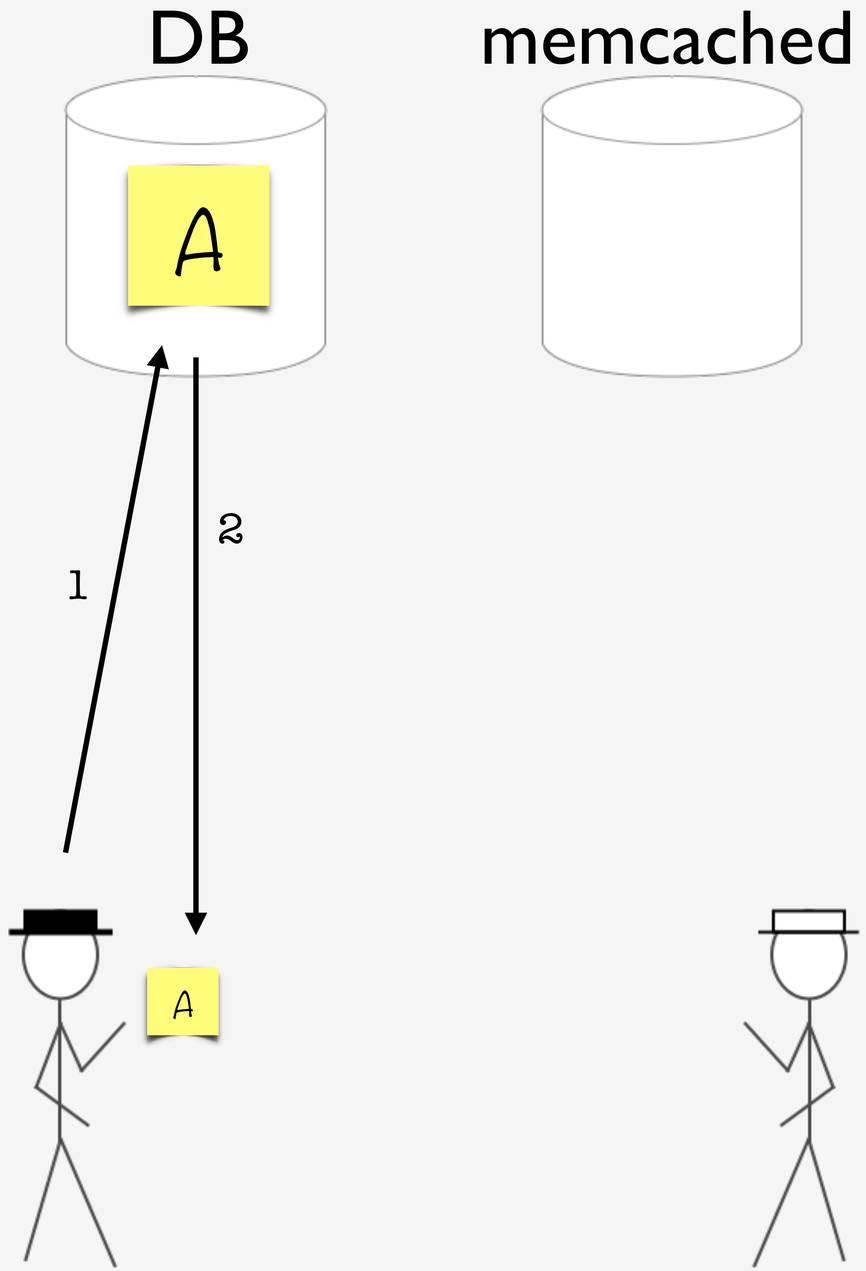
DB

memcached

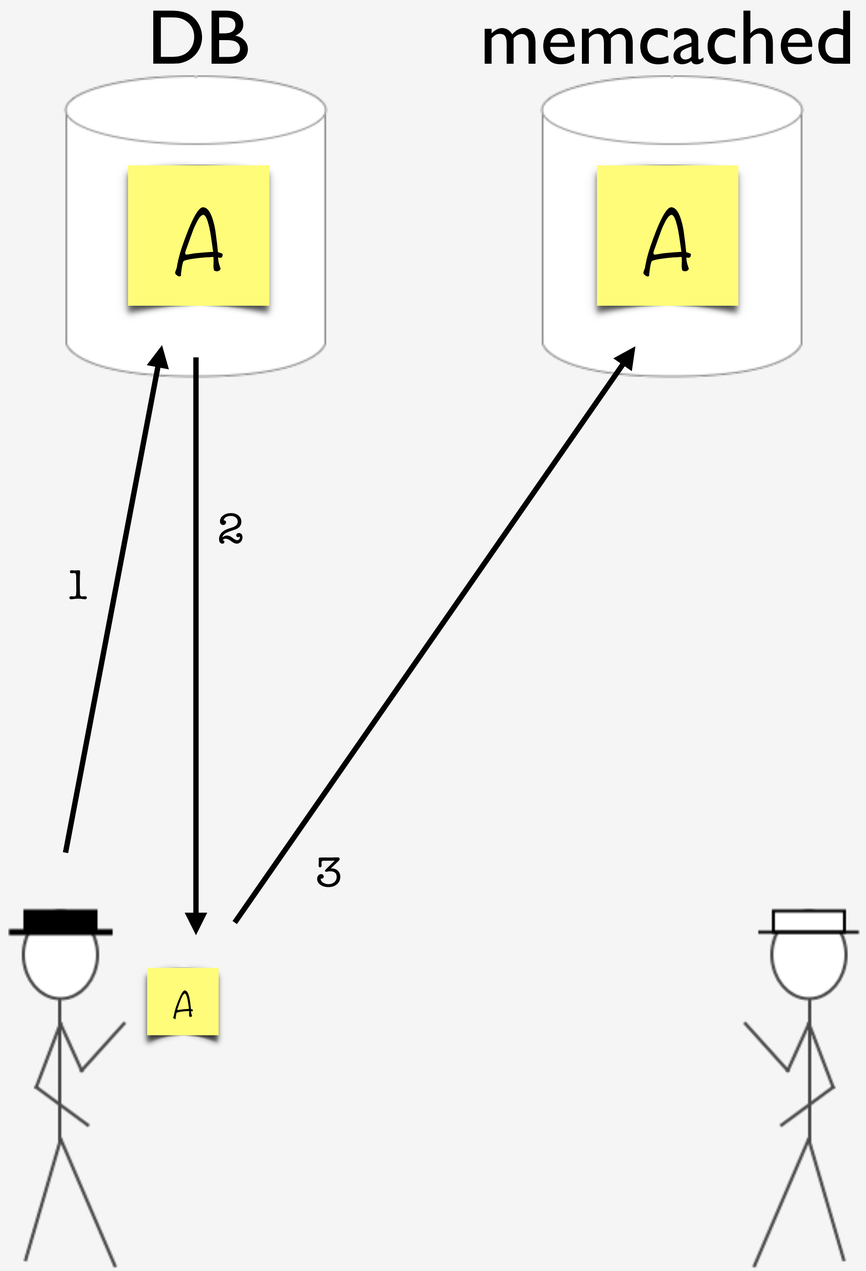
Update...



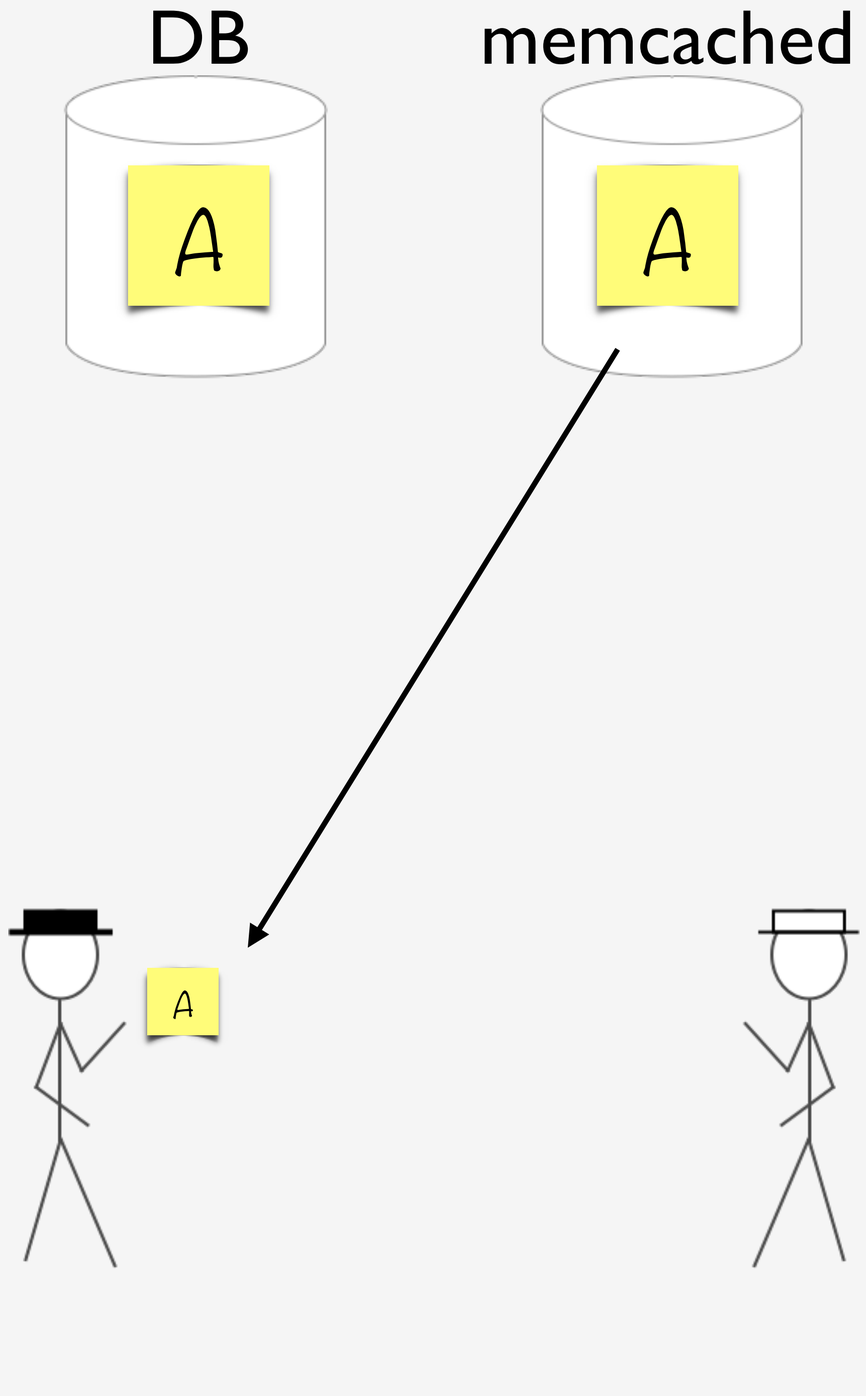
Update...



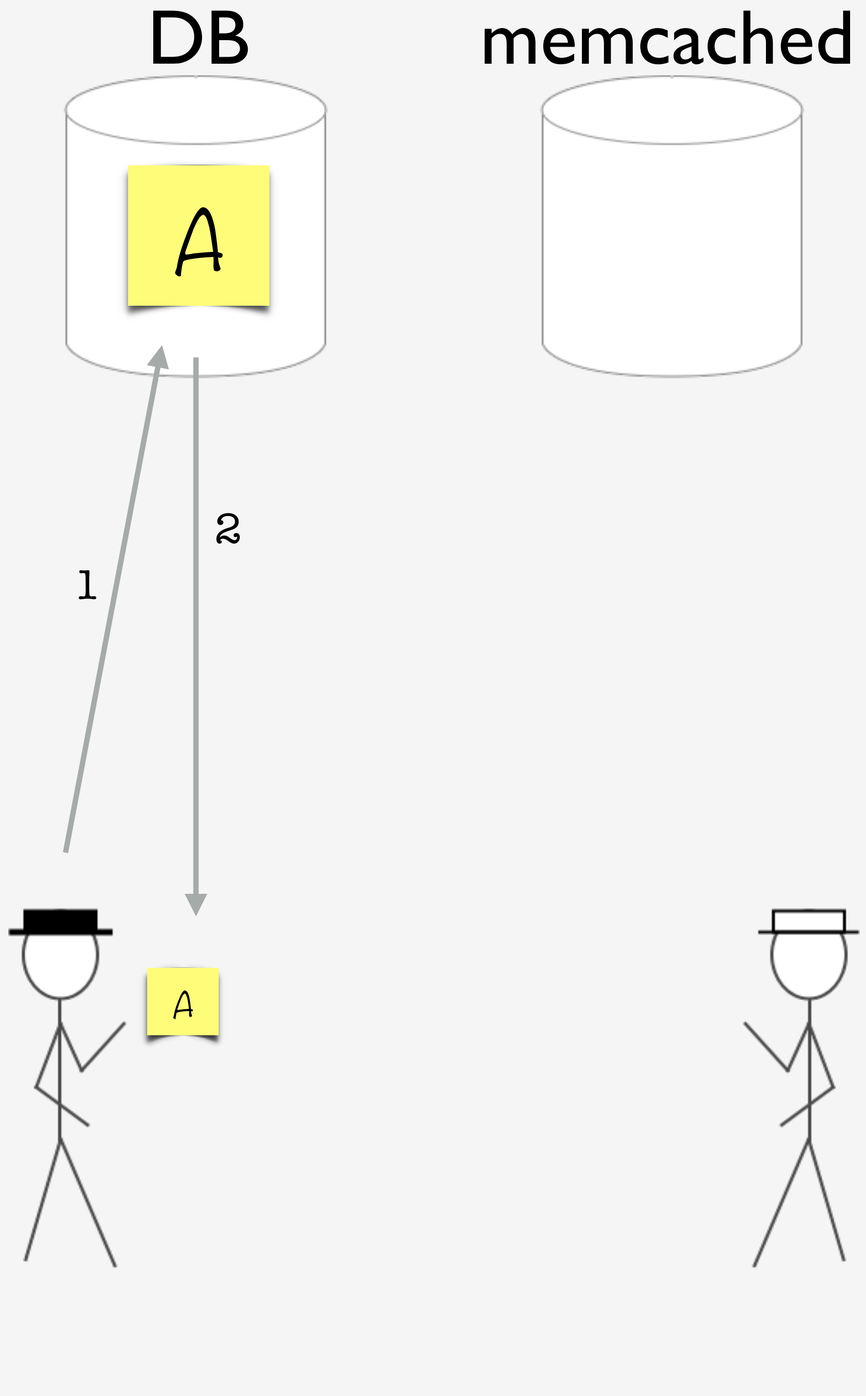
Update...

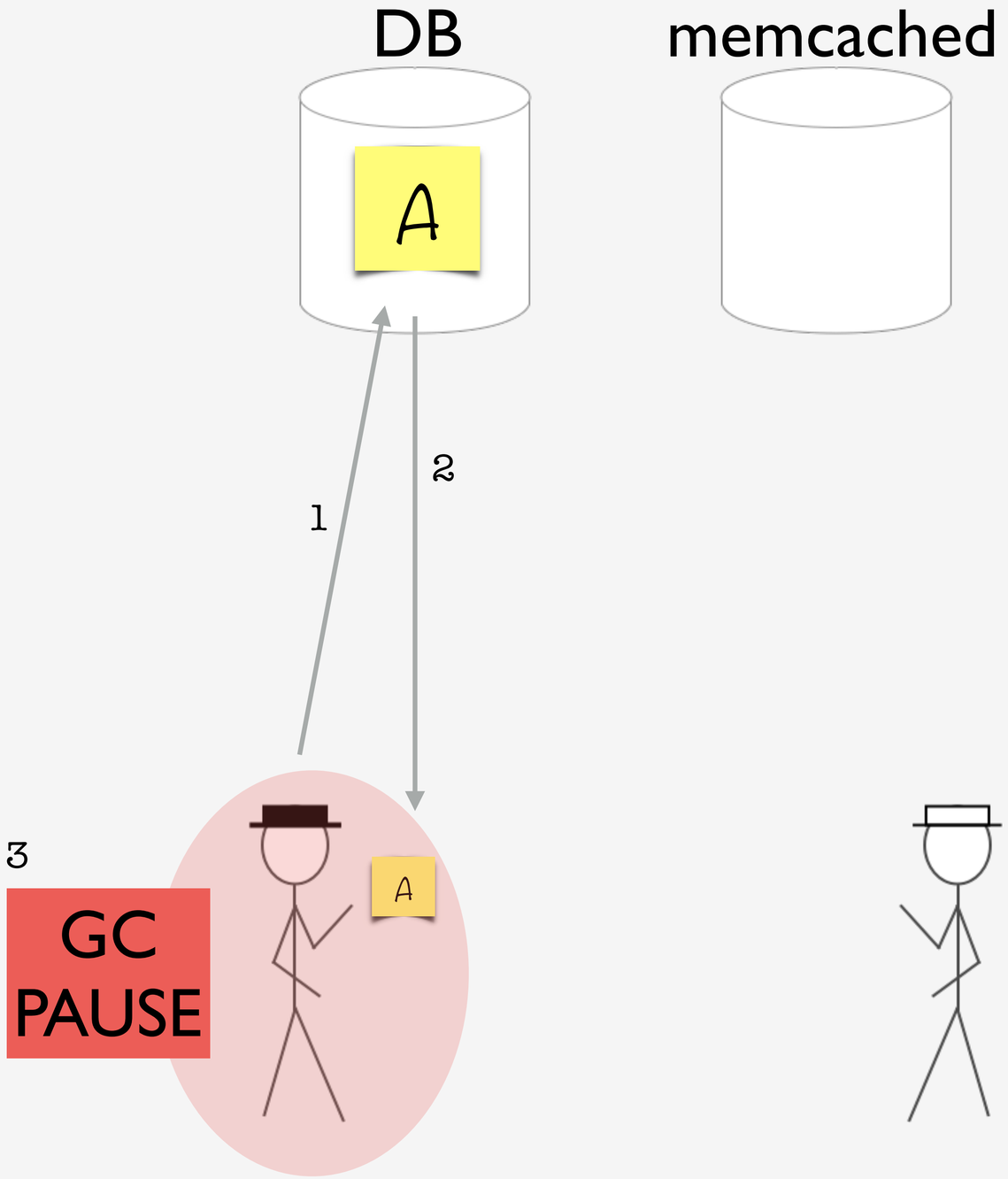


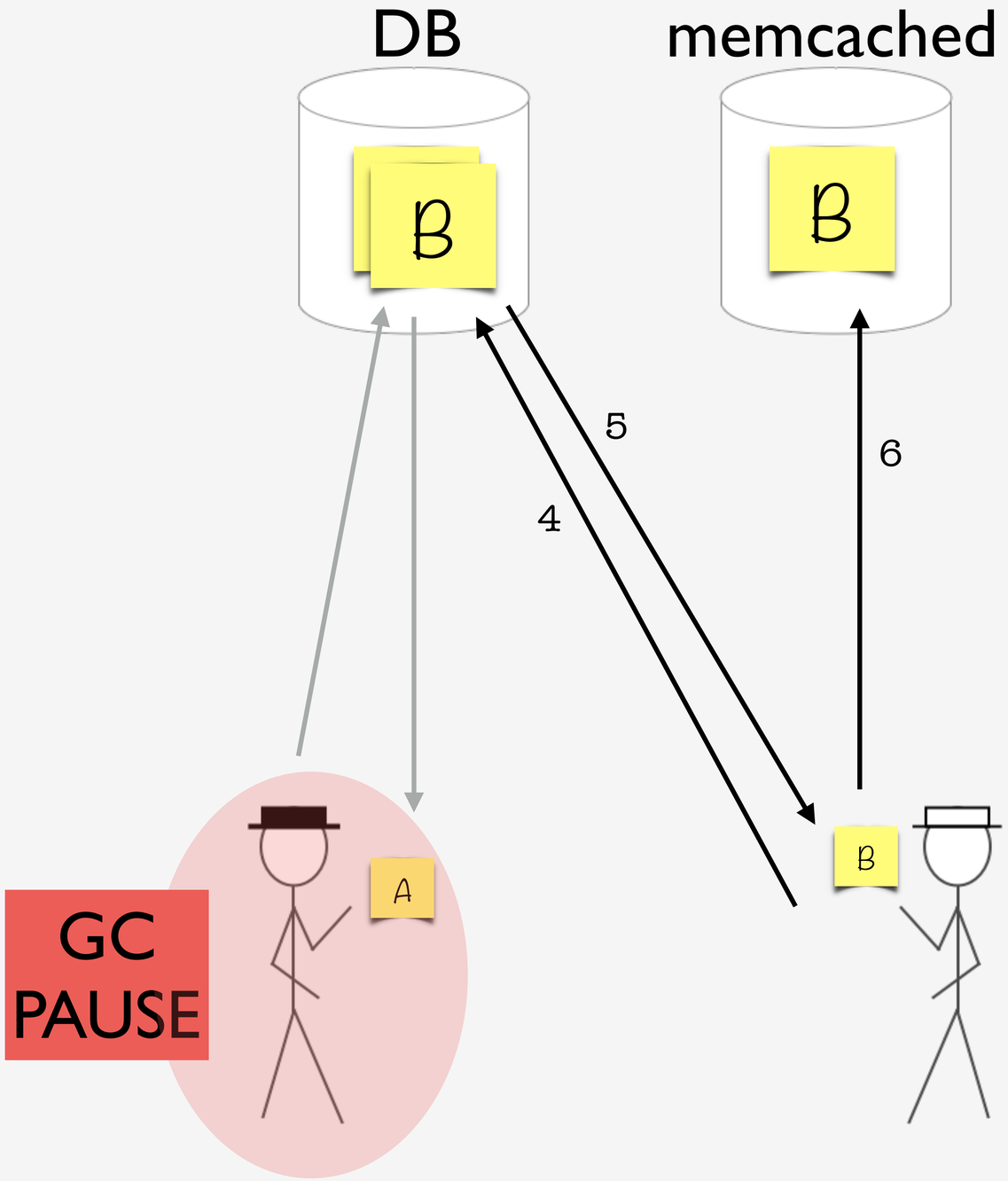
Read...

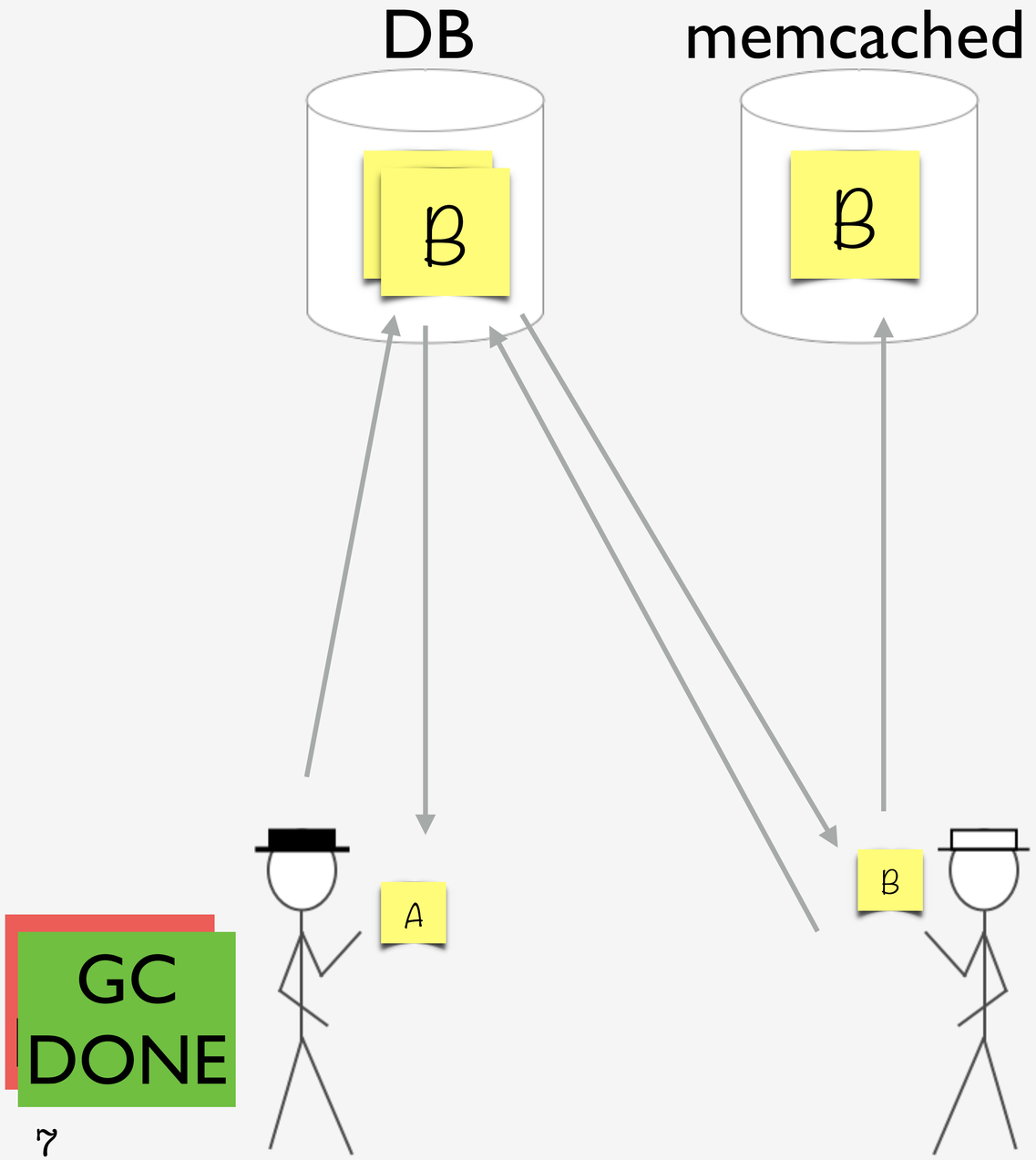


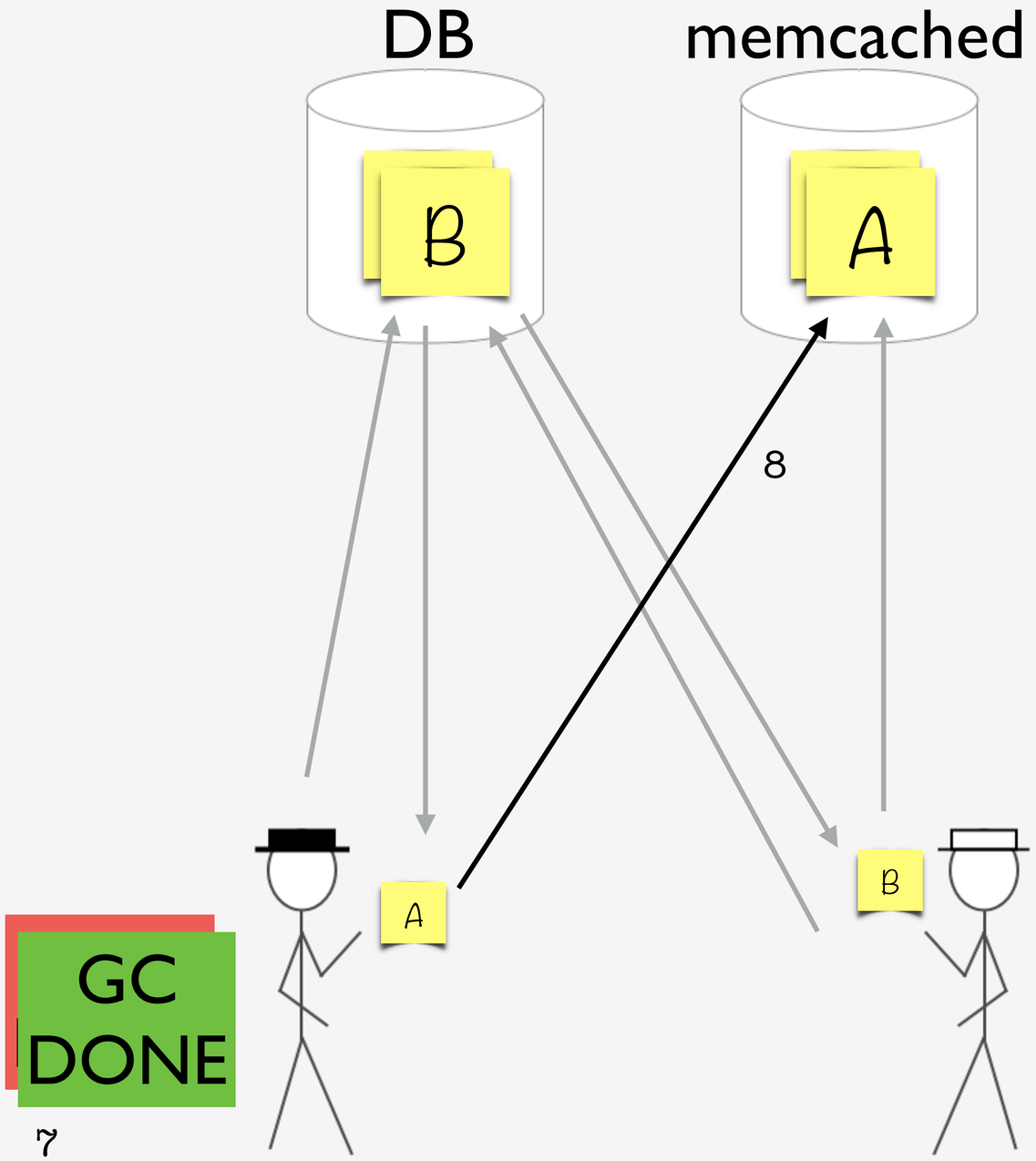
How can that go wrong?

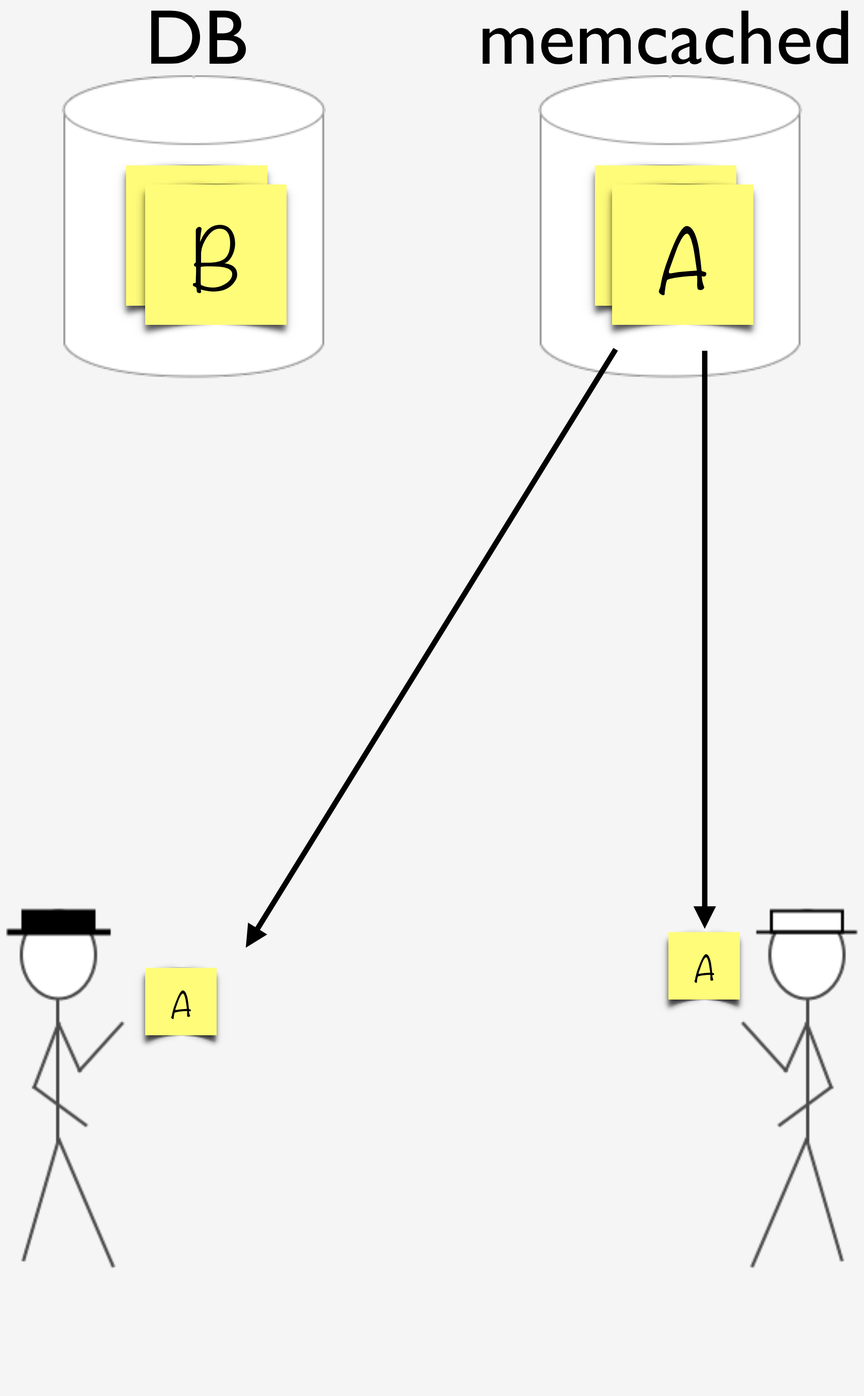

DB
memcached
A
1
2
A


DB
memcached
A
1
2
3
GC
PAUSE
A


DB
memcached
B
B
5
4
6
GC
PAUSE
A
B


DB
memcached
B
B
A
B
GC
DONE


DB
memcached
B
A
8
A
B
GC
DONE
7


DB
memcached
B
A
A
A

DB + memcached is not a ‘good’ system

What makes a good distributed system?

CAP Theorem

Consistency
Availability
Partition tolerance

Pick two!

Consistency

Appears to be a single-copy of the data to an outside observer.

Weaker models exist, e.g. ‘eventual consistency’.

Availability

Node failures don't prevent survivors from operating.

Partition tolerance

Partition: network can lose arbitrarily many messages from one node to another

Tolerant: other properties remain true

Can’t avoid partitions!

CP or AP only!

CAP → PAC/ELC

```
# pac/elc system design

if ( partition ) {
    pick(“availability”, “consistency”)
}
else {
    pick(“low latency”, “consistency”)
}
```

Still simplistic

Reads vs writes

Majority-side of a partition

Can write (appears consistent)

Can read (available)

Minority-side of a partition

Can't write (not available)

Can read (available – but stale)

‘Practical Consistency’

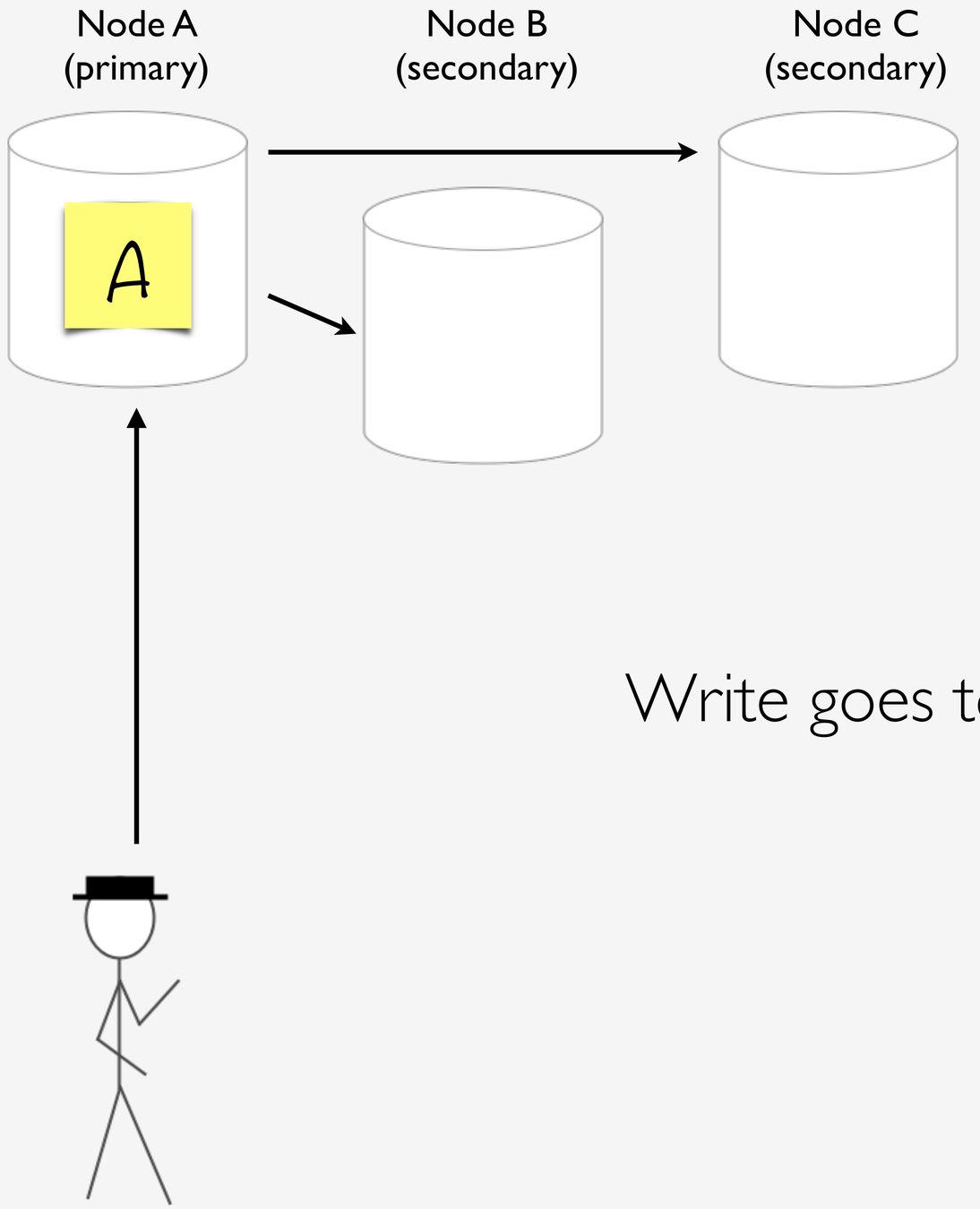
Do I know when a write is committed?

How do I read only committed and/or current data?

Thinking about writes...

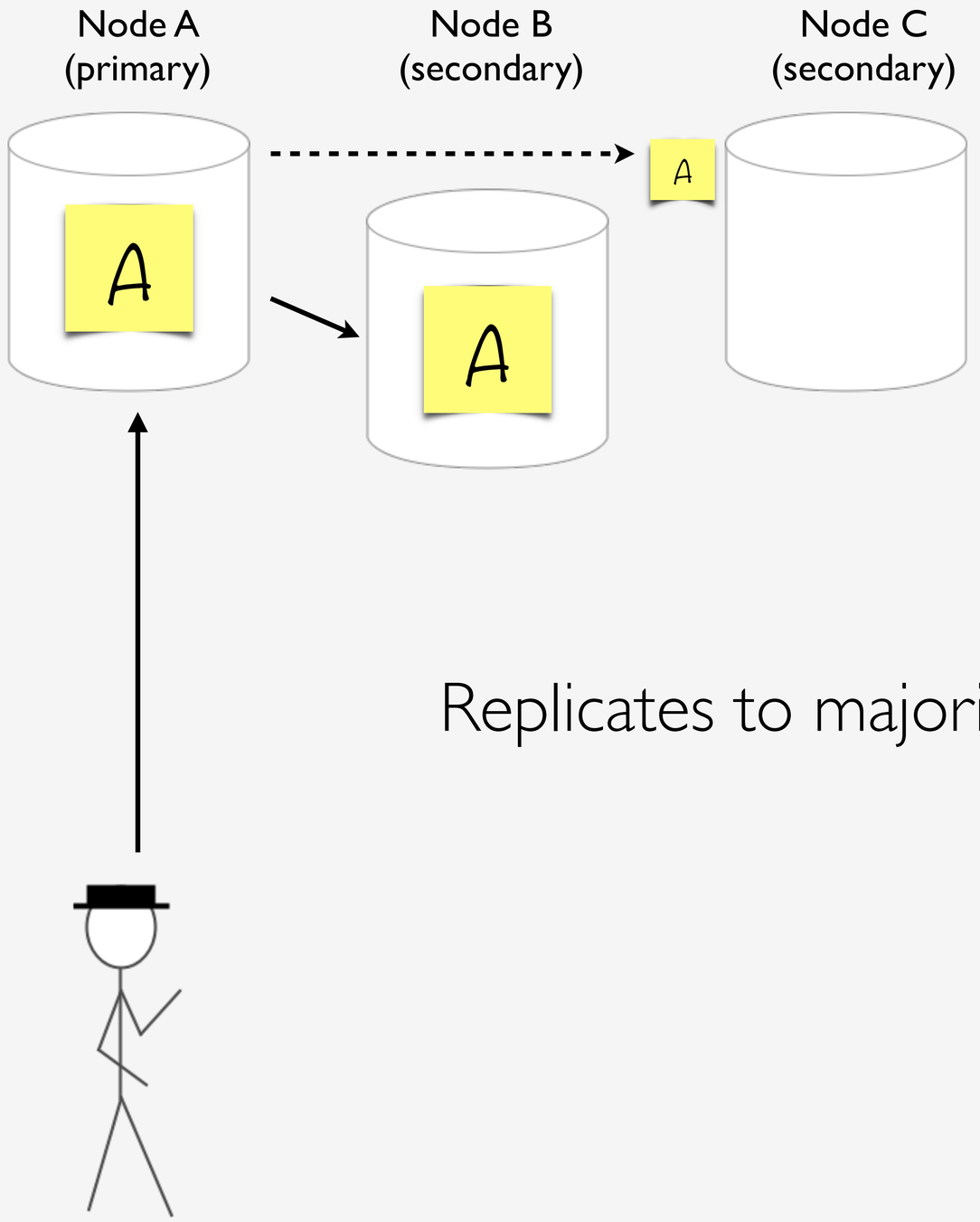
Durability
Convergence
Error recovery

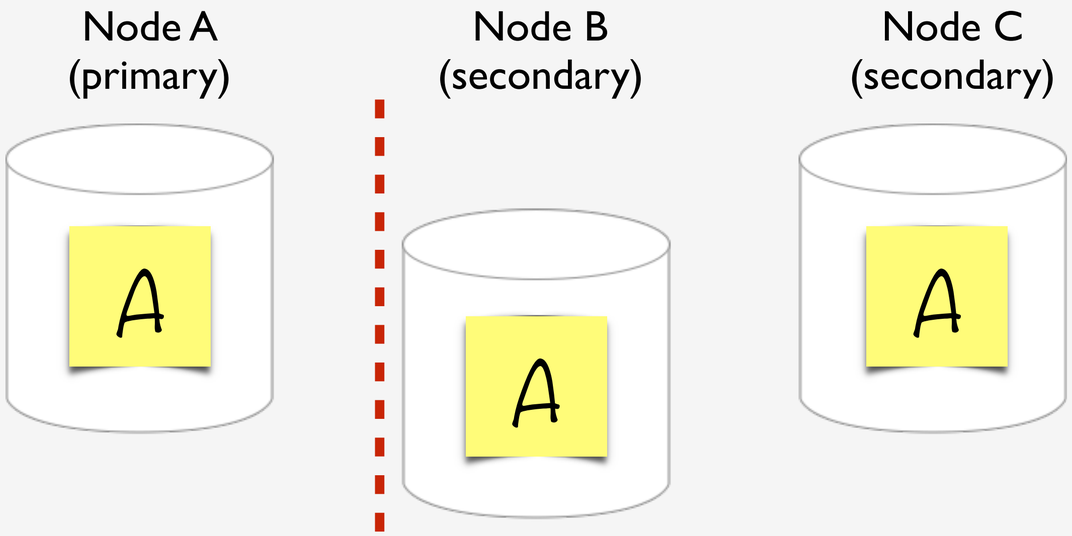
Do we know when writes are durable?



mongoDB

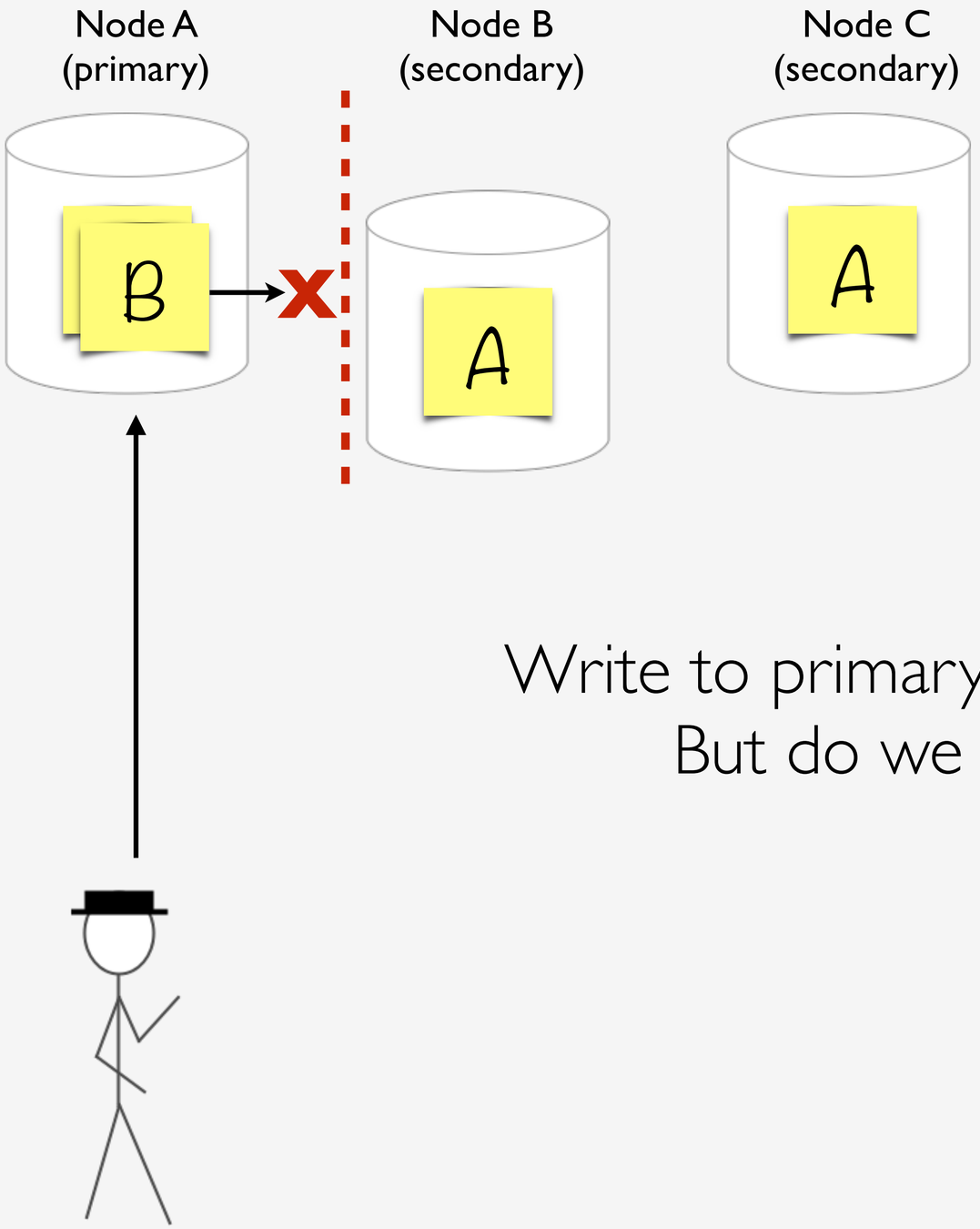
Write goes to primary


Node A
(primary)
Node B
(secondary)
Node C
(secondary)
A
A
A
Replicates to majority → committed



mongoDB

Partition separates primary

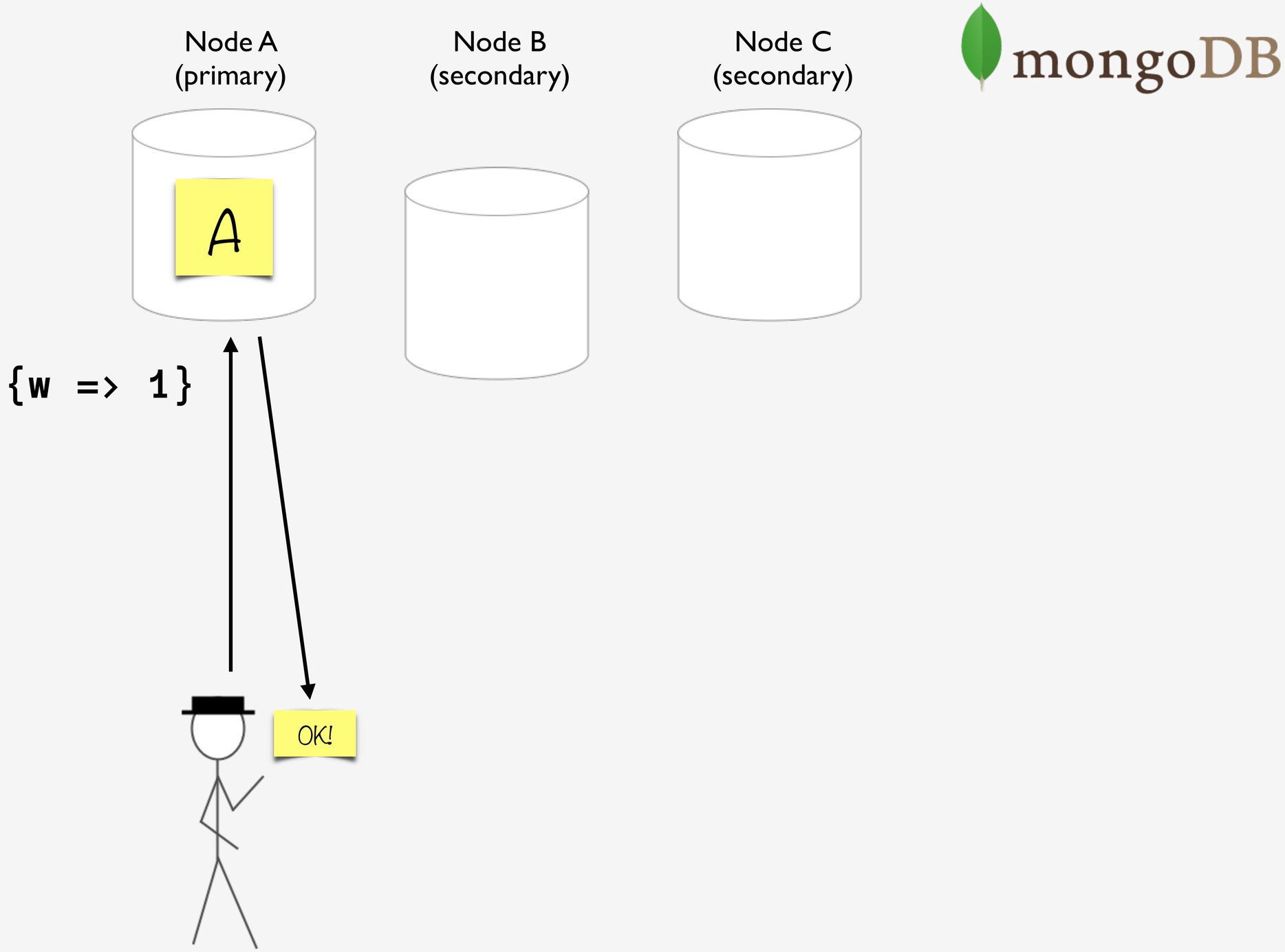


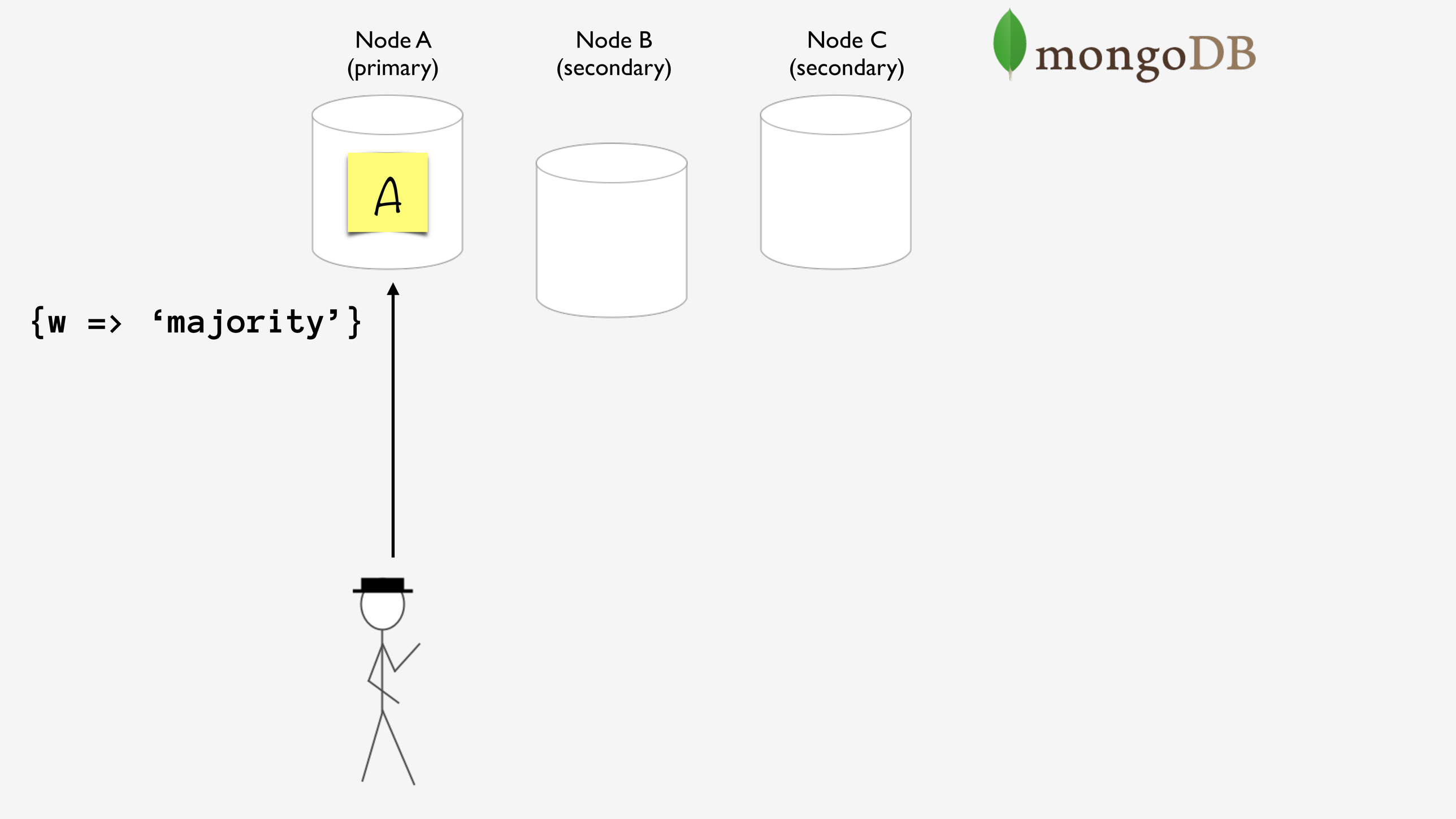
mongoDB

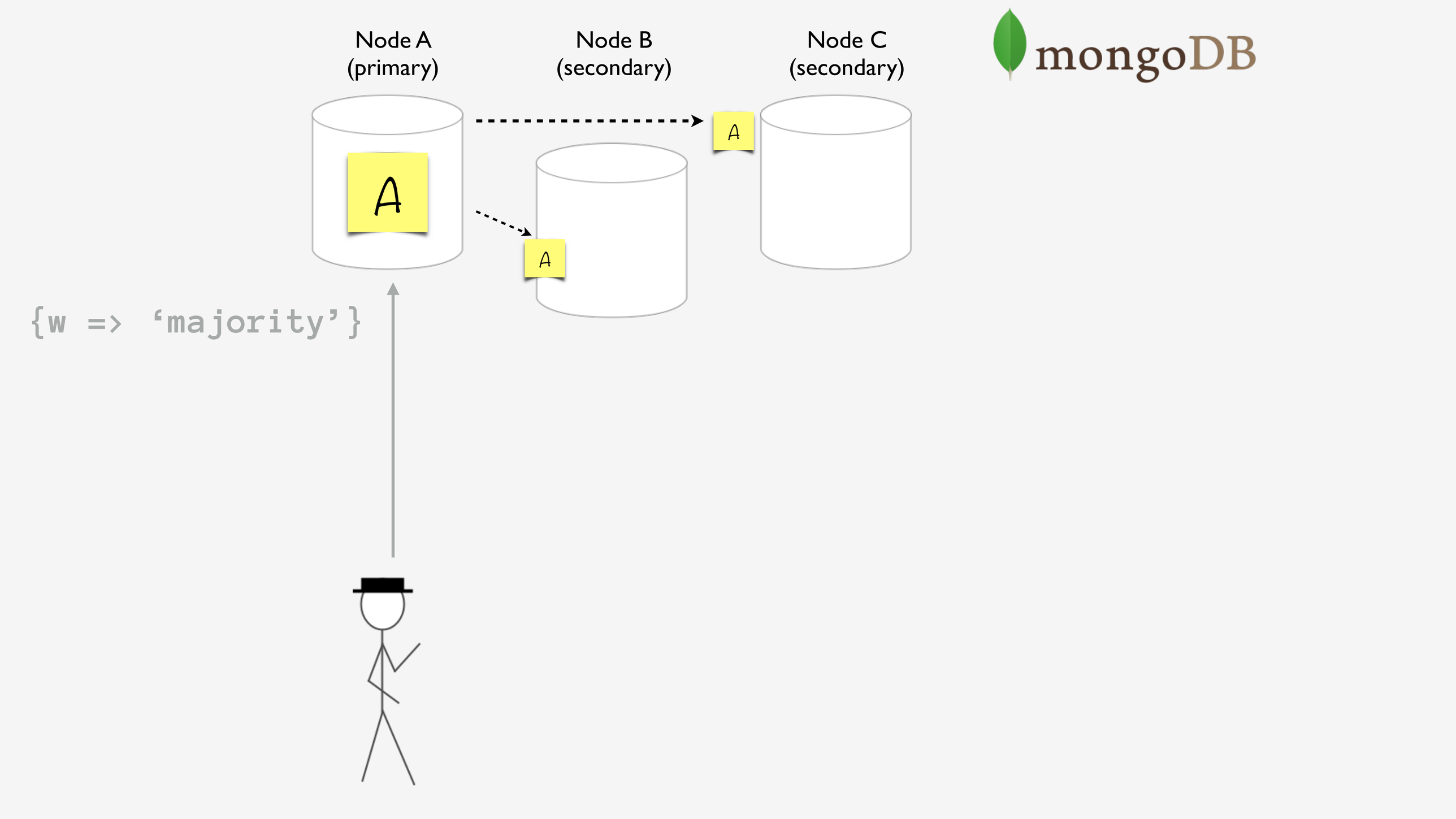
Write to primary can't replicate
But do we find out?

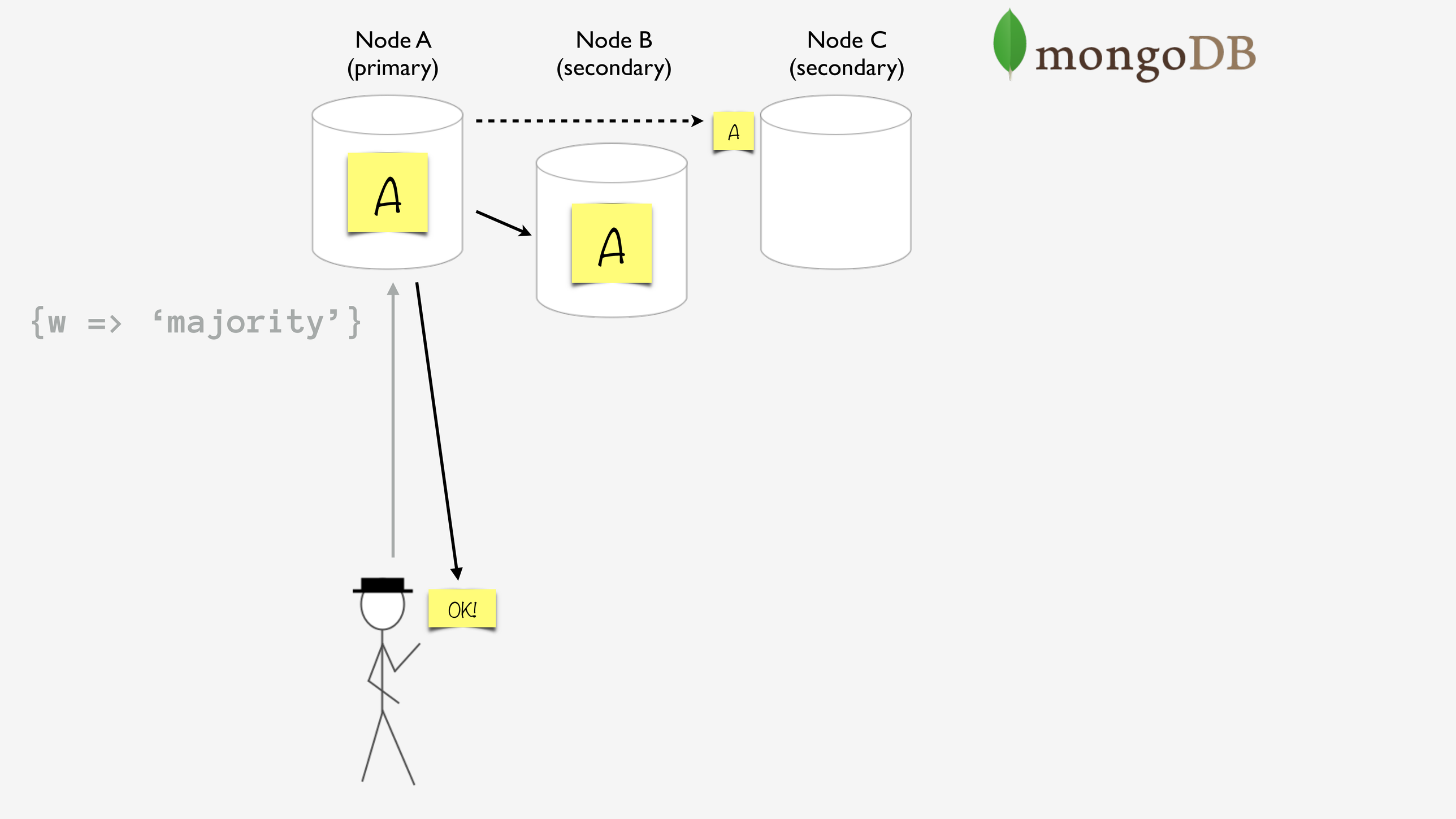
mongoDB

```
# write concern

MongoDB->connect( $url,
    { w => 1 }
);

MongoDB->connect( $url,
    { w => ‘majority’ }
);
```

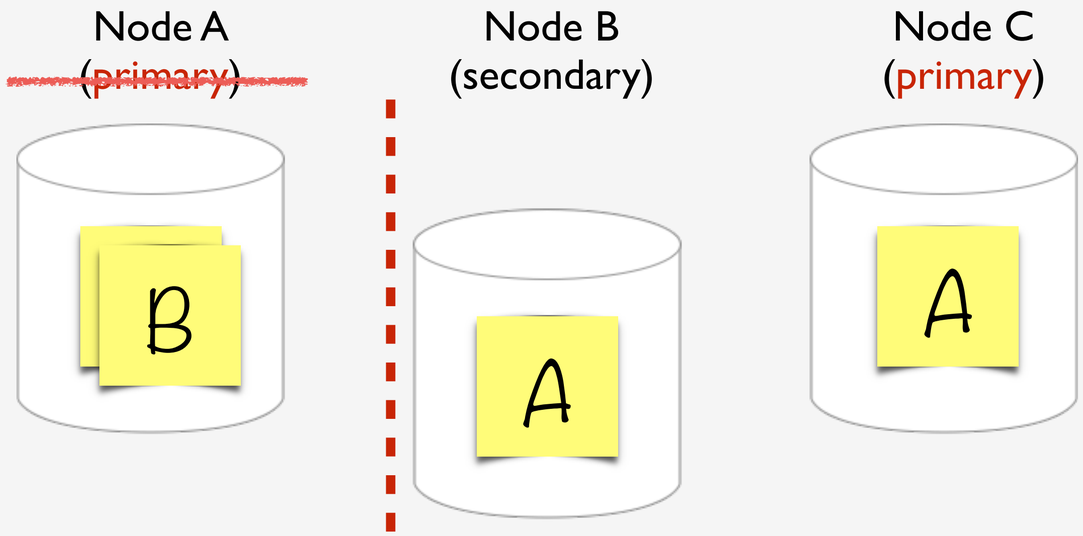

Node A
(primary)
Node B
(secondary)
Node C
(secondary)
A
{w => 1}
OK!


Node A
(primary)
Node B
(secondary)
Node C
(secondary)
mongoDB
A
{w => ‘majority’}


Node A
(primary)
Node B
(secondary)
Node C
(secondary)
A
A
A
{w => 'majority'}

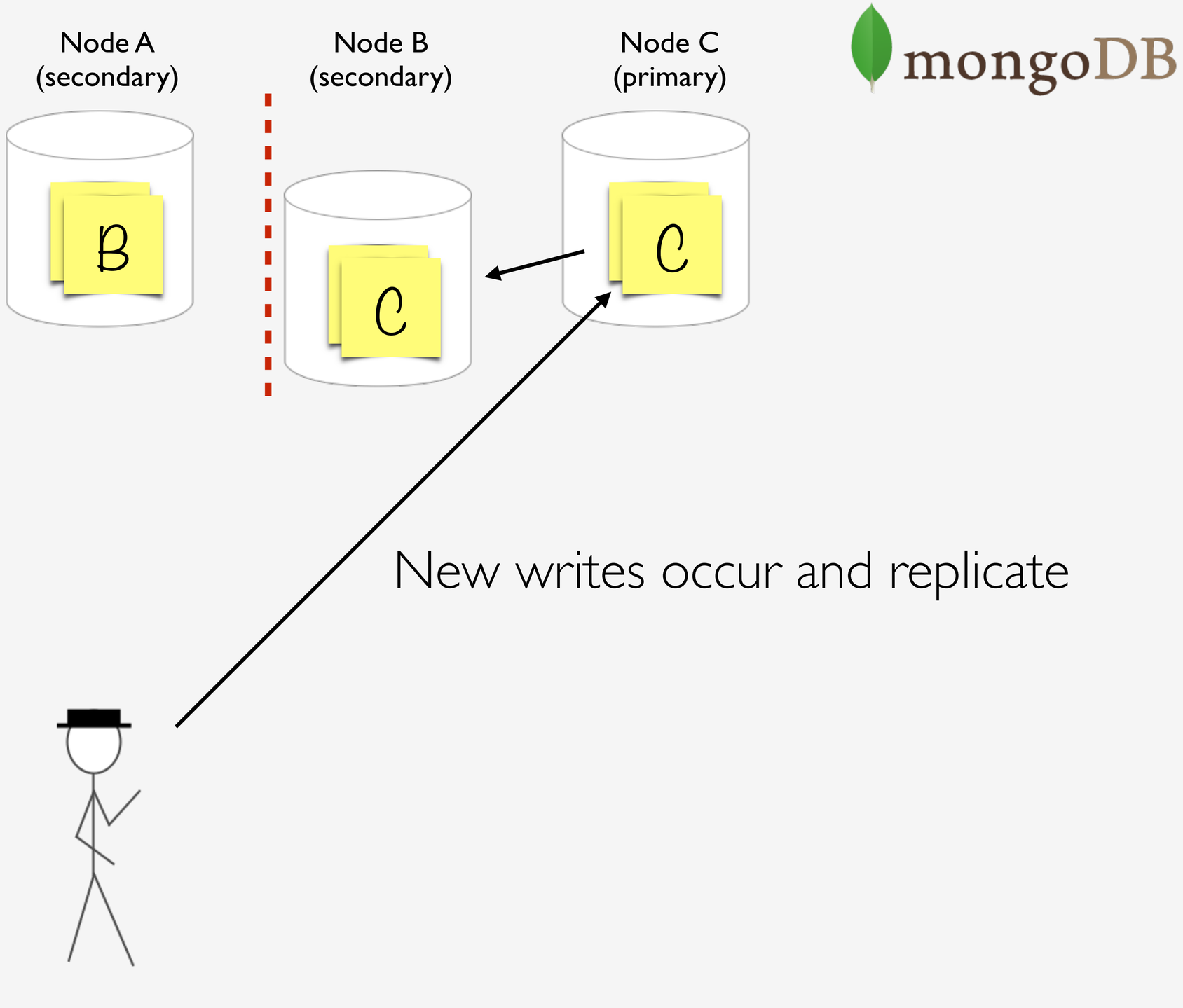

Node A
(primary)
Node B
(secondary)
Node C
(secondary)
A
A
A
{w => 'majority'}
OK!

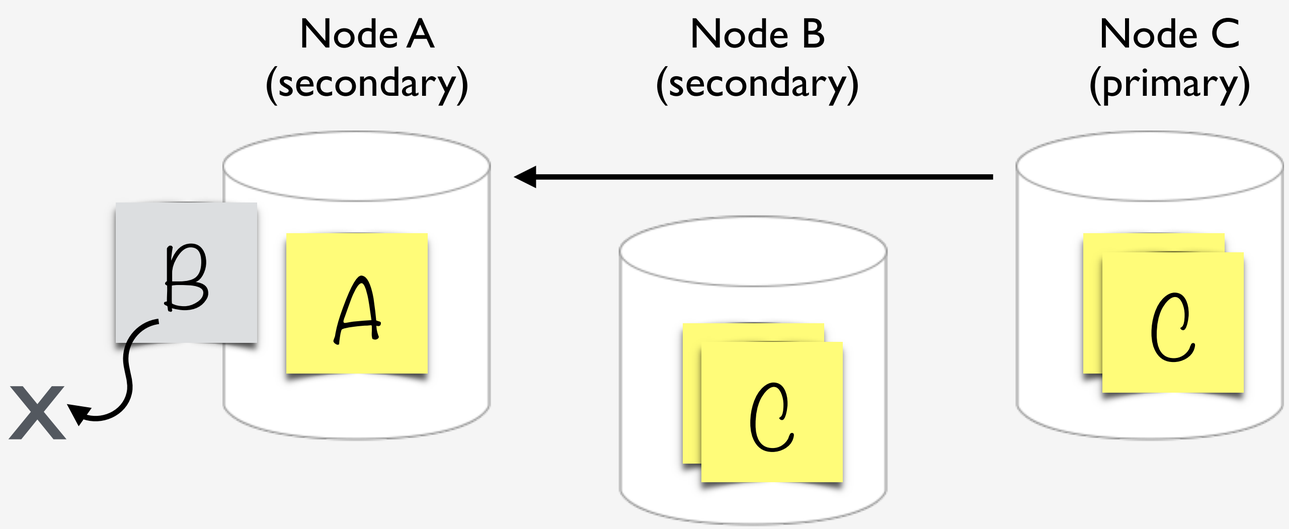
How will the system converge on recovery?



mongoDB

Old primary steps down
New primary elected

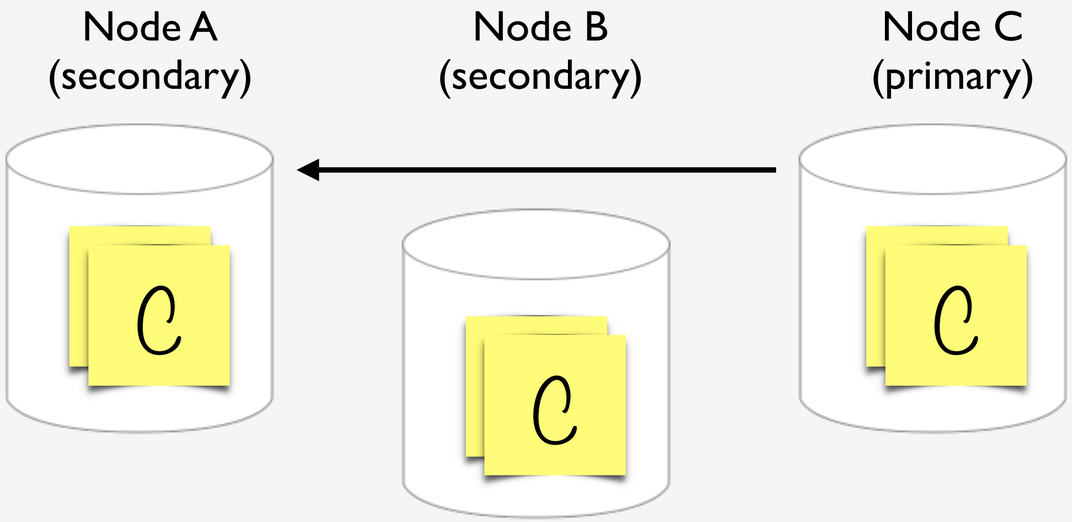

Node A
(secondary)
Node B
(secondary)
Node C
(primary)
B
C
C
New writes occur and replicate



mongoDB

Partition heals

Returning node rolls back history

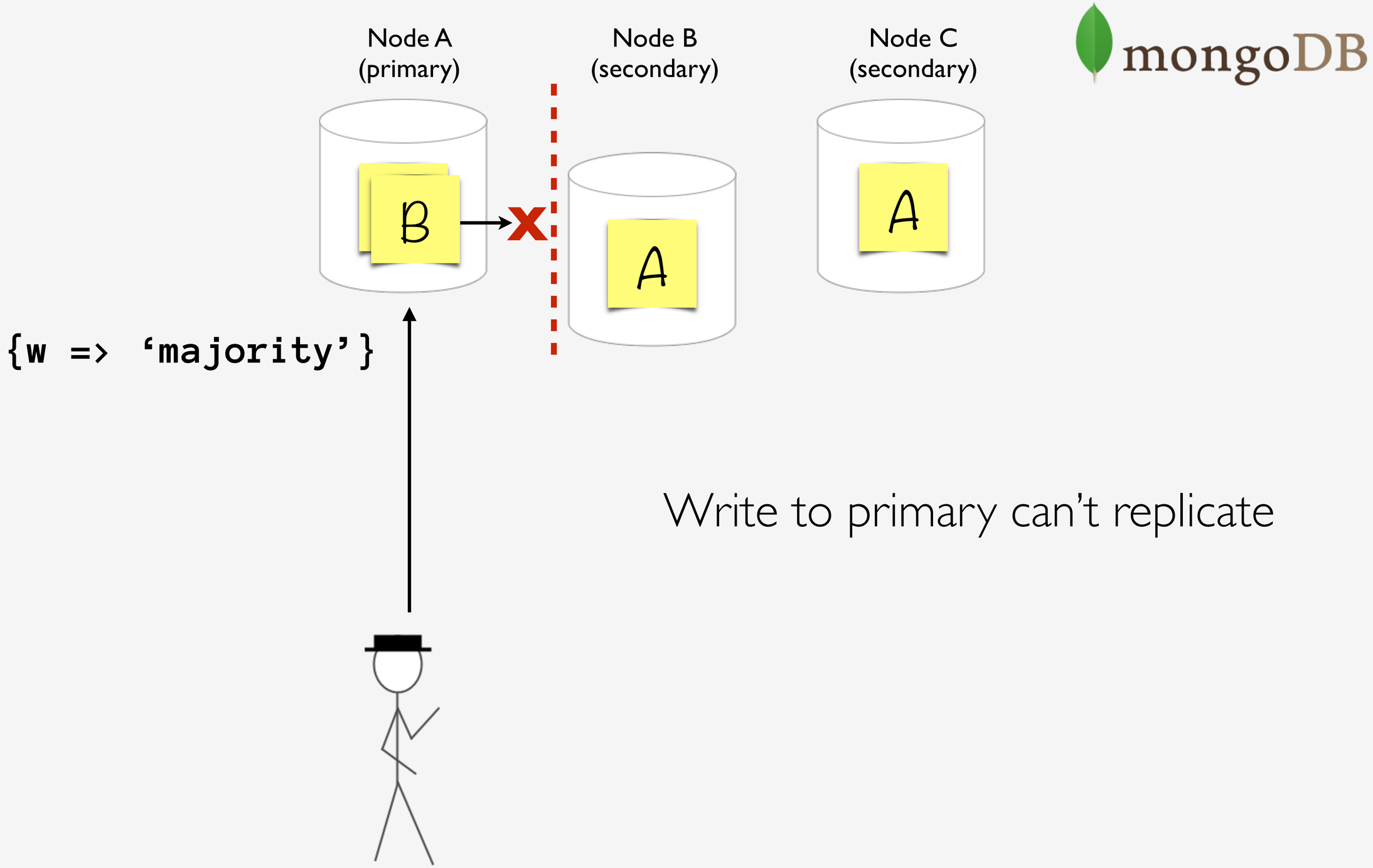


mongoDB

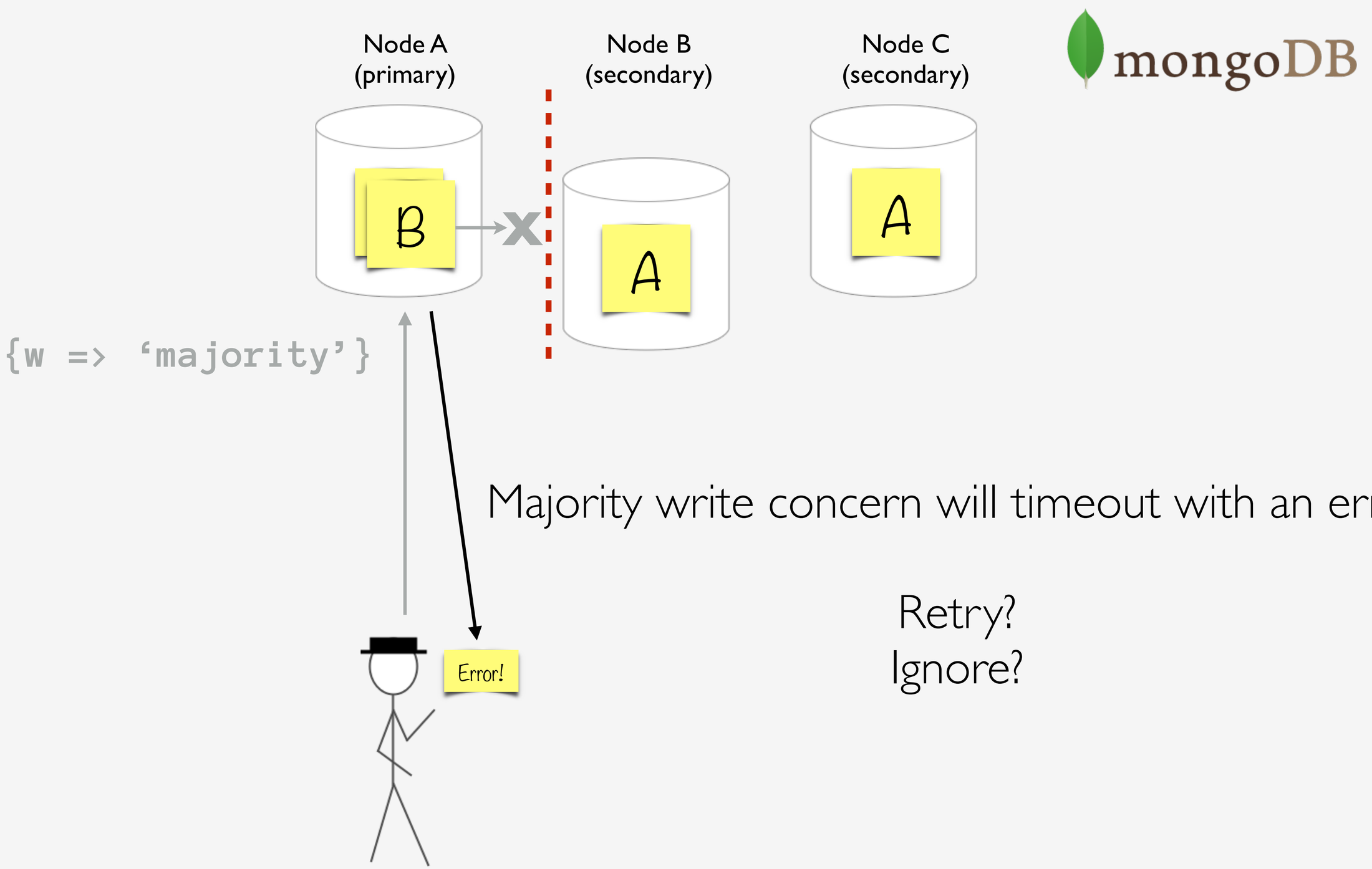
Returning node catches up with primary

- Rollback
- Conflict records
- Conflict-free replicated data type (CRDT) (e.g. “add to set”)

What do we do with a write error?



Write to primary can't replicate



Majority write concern will timeout with an error

Retry?
Ignore?

Answers are specific to your application!

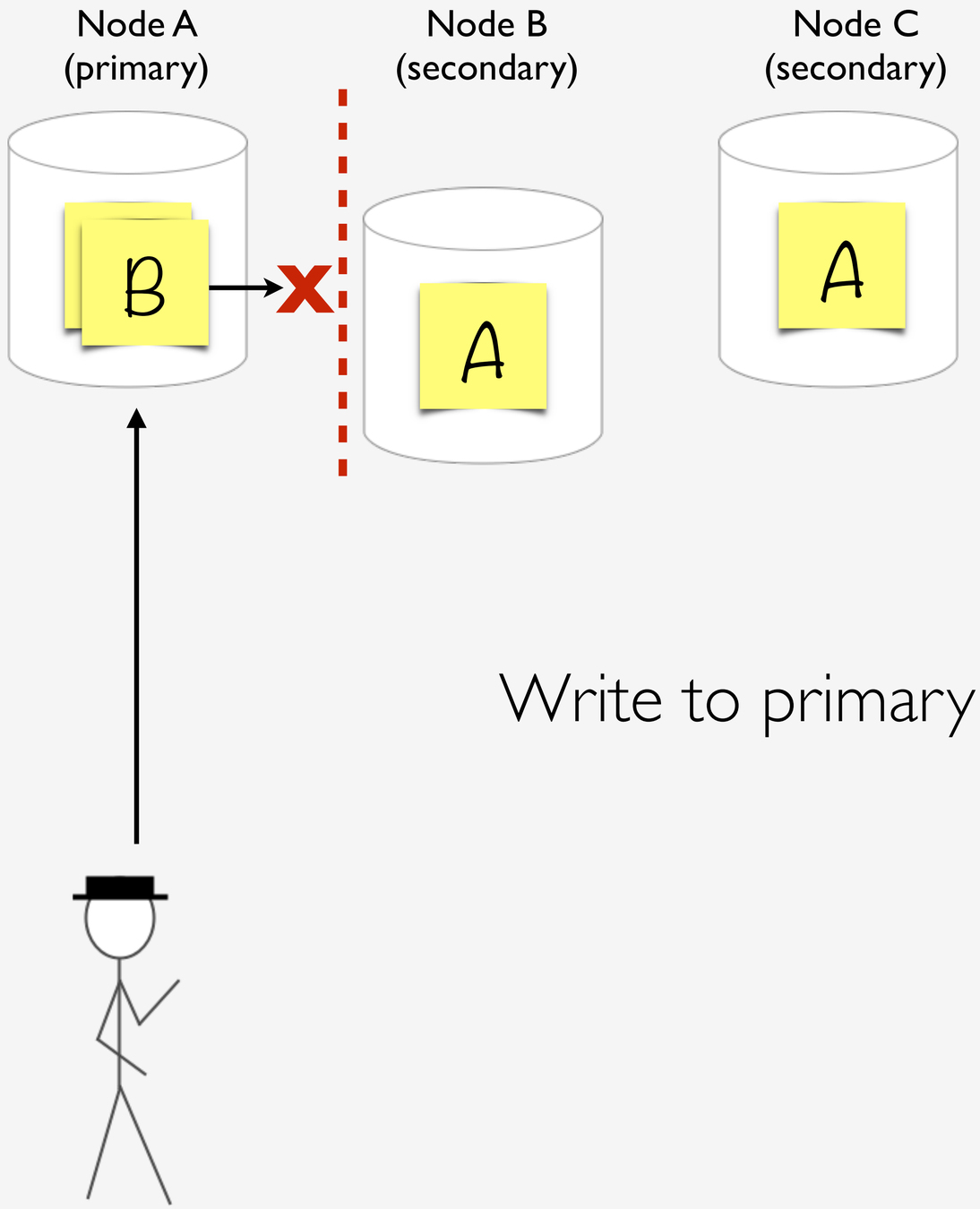
Thinking about reads...

Recency
Durability
Latency

Do we care if we read the latest write?

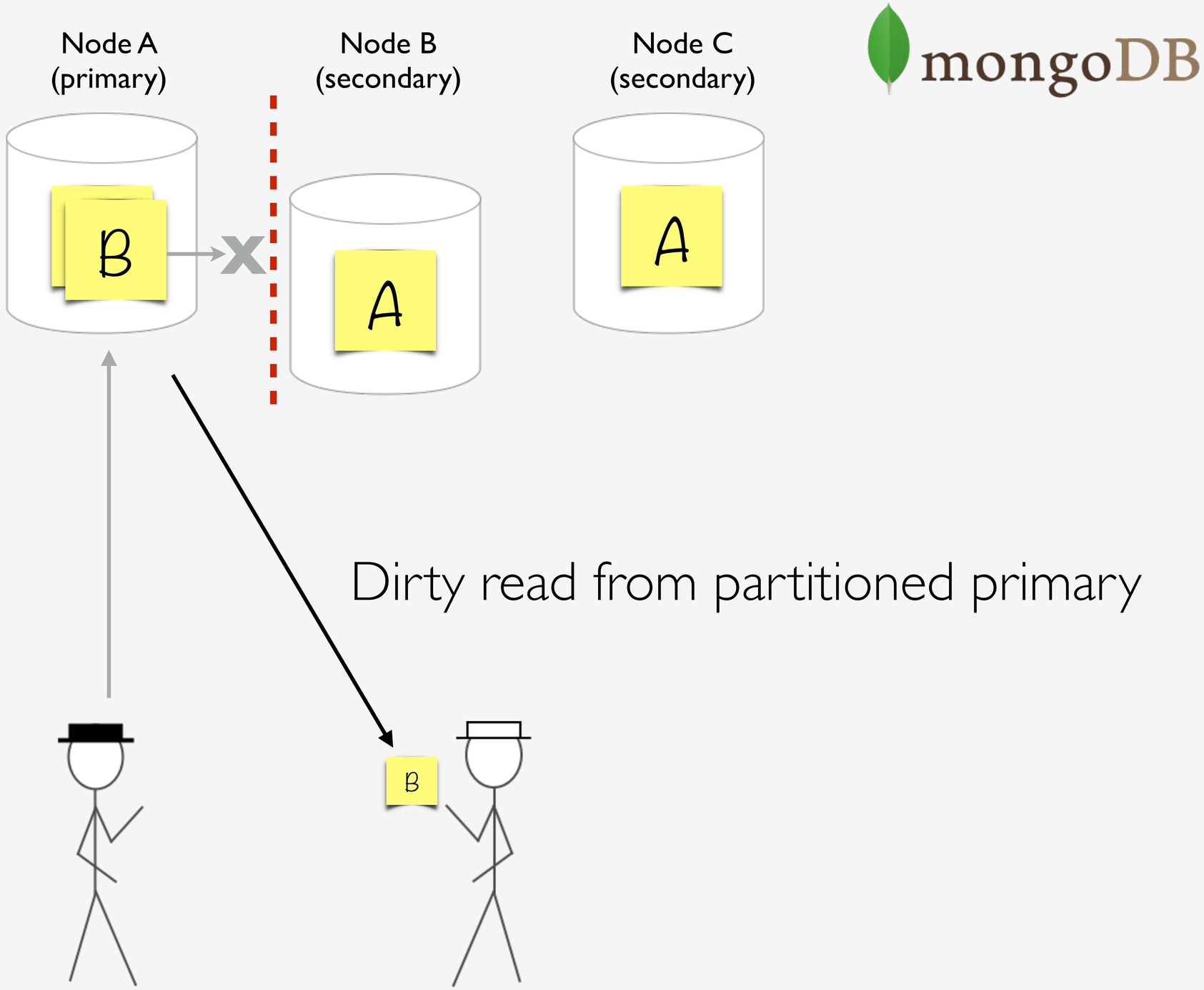
Do we care if data we read rolls back?

Trade recency for durability

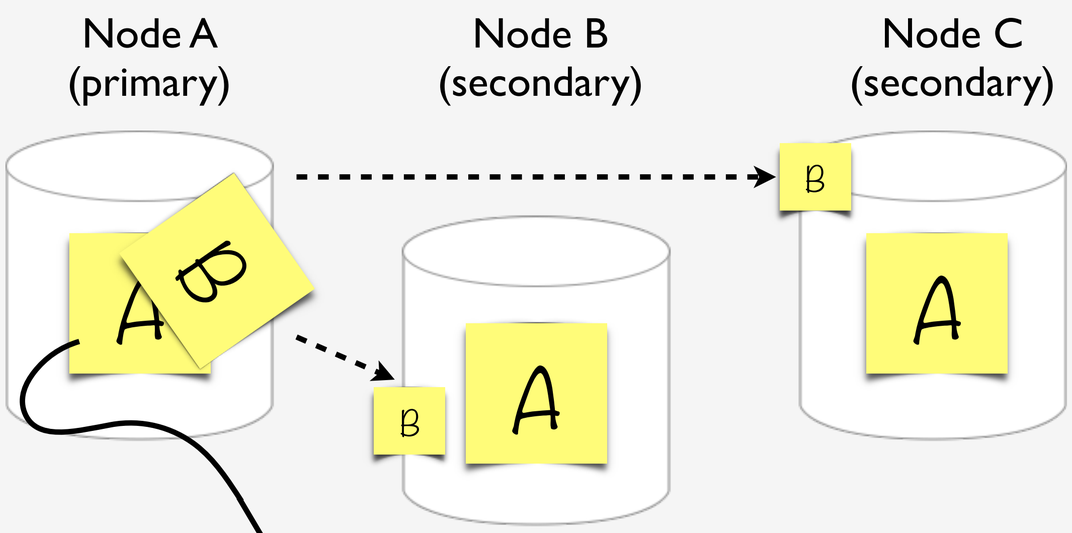


mongoDB

Write to primary can't replicate

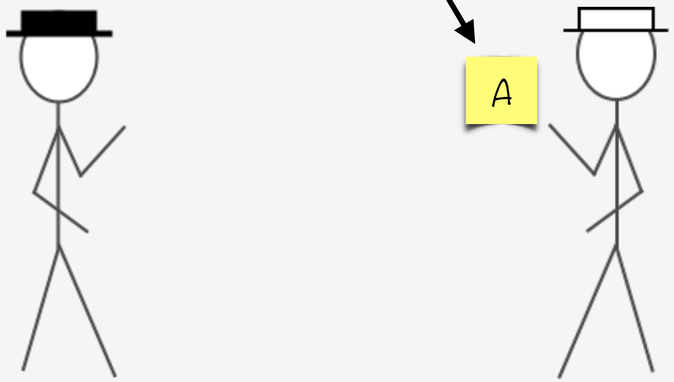

Node A
(primary)
Node B
(secondary)
Node C
(secondary)
B
A
A
B
Dirty read from partitioned primary


```
# read concern (3.2+)

MongoDB->connect( $url,
    { read_concern_level => ‘local’ }
);

MongoDB->connect( $url,
    { read_concern_level => ‘majority’ }
);
```



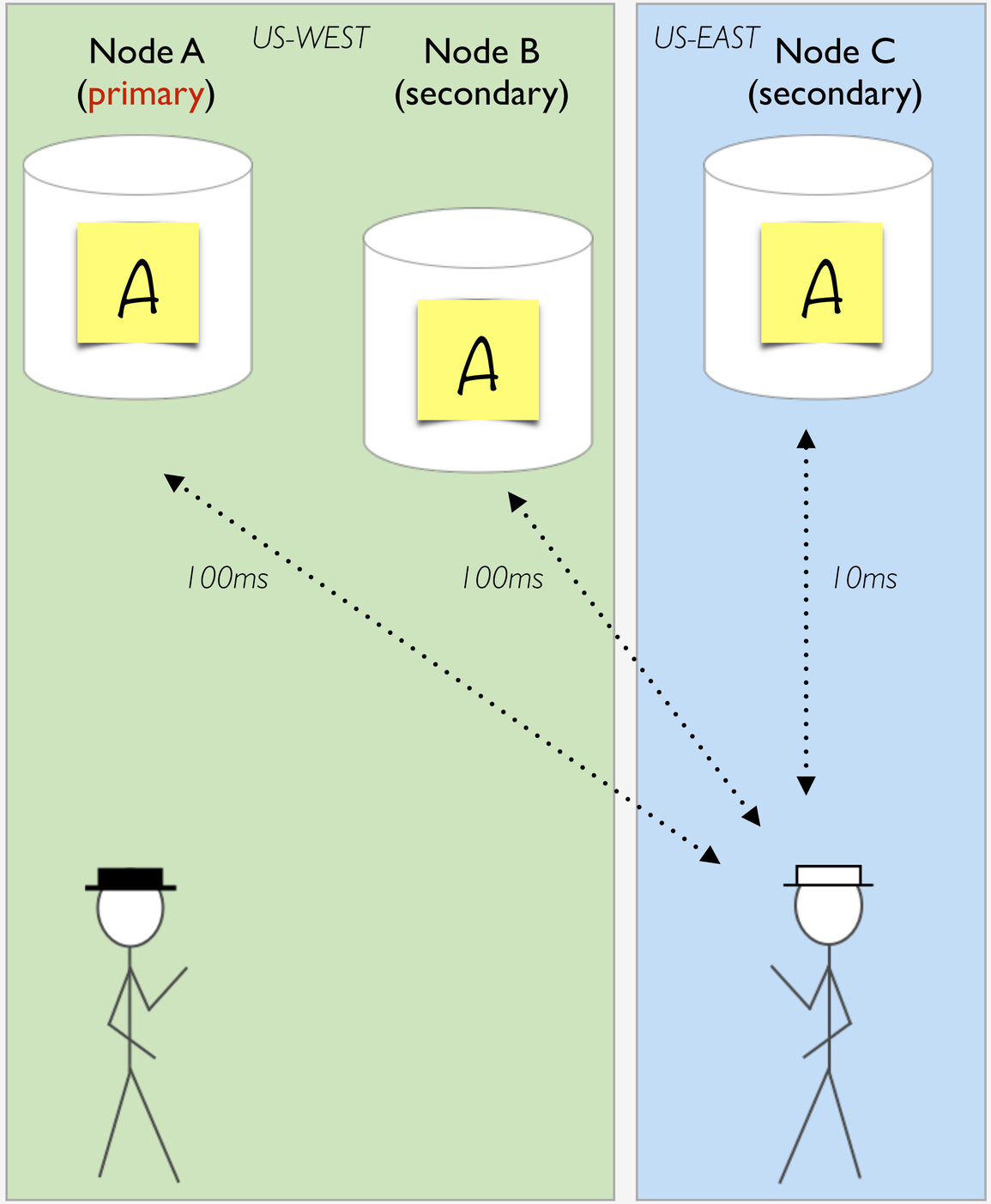

```
{read_concern_level => 'majority'}
```

Without partition, majority read concern lags replication



Trade recency for latency

Round-trip time

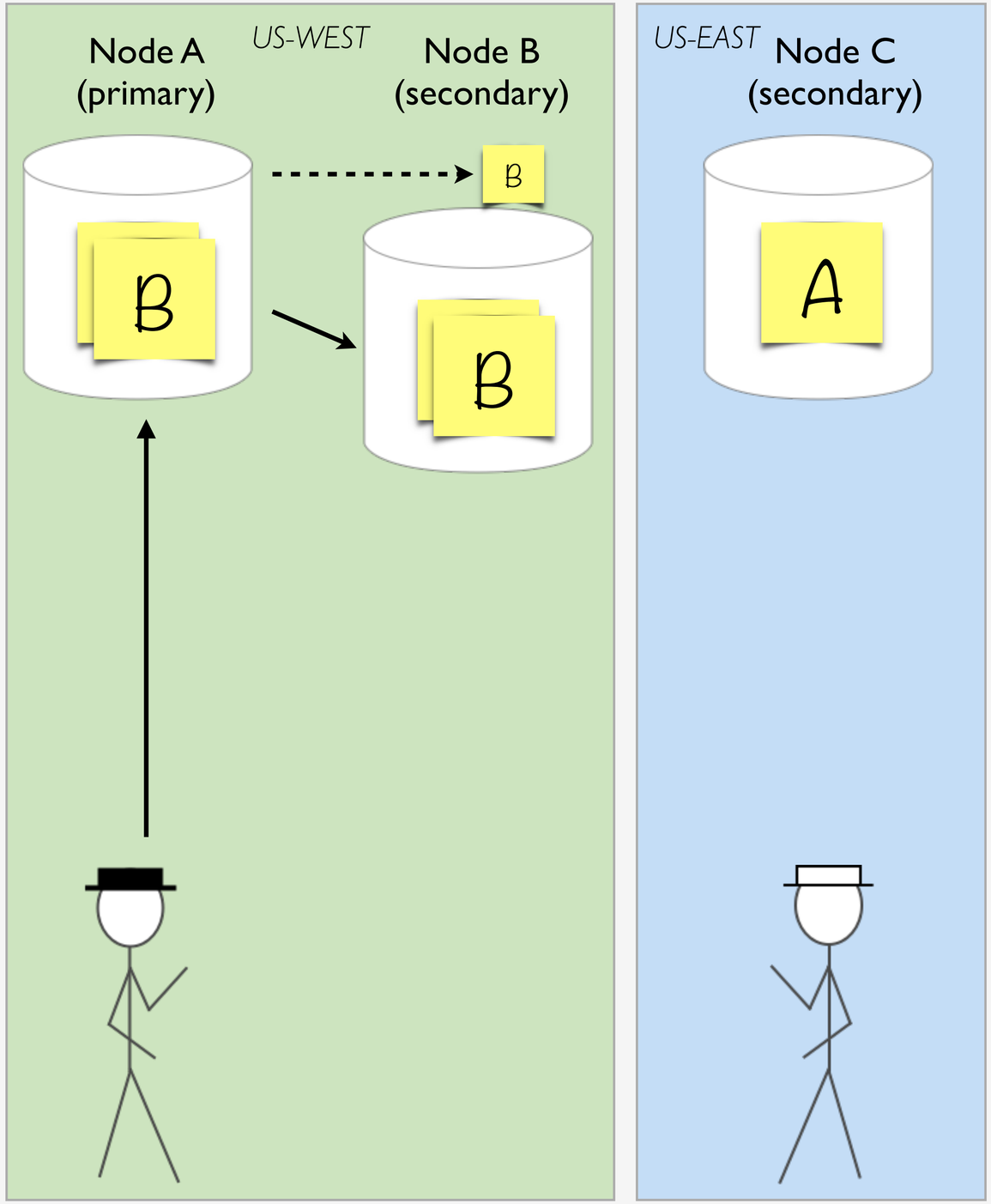


mongoDB

RTT for each data center

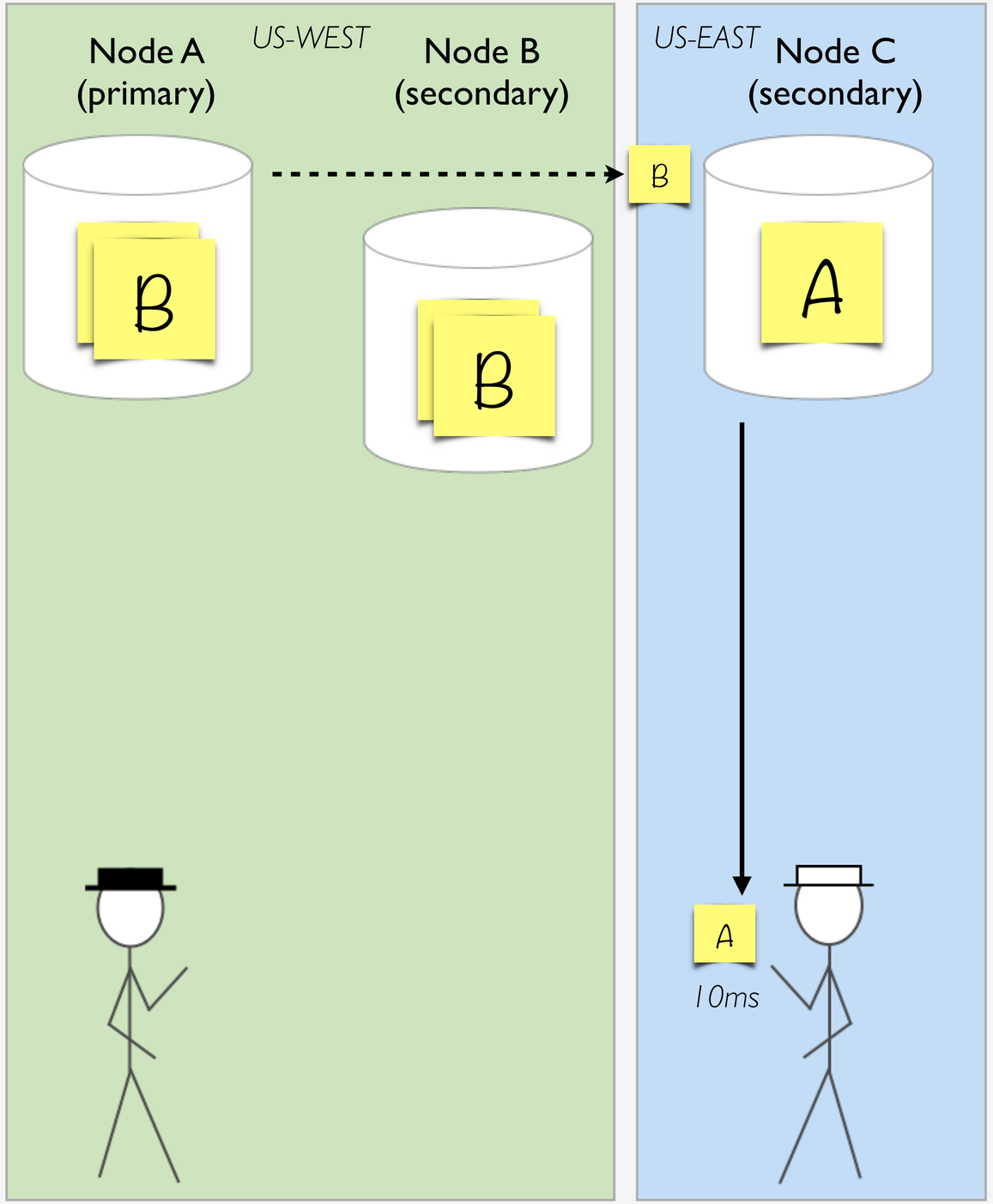
mongoDB

```
# read preference

MongoDB->connect( $url,
    { read_pref_mode => ‘primary’ }
);

MongoDB->connect( $url,
    { read_pref_mode => ‘nearest’ }
);
```



mongoDB

Primary write starts replicating

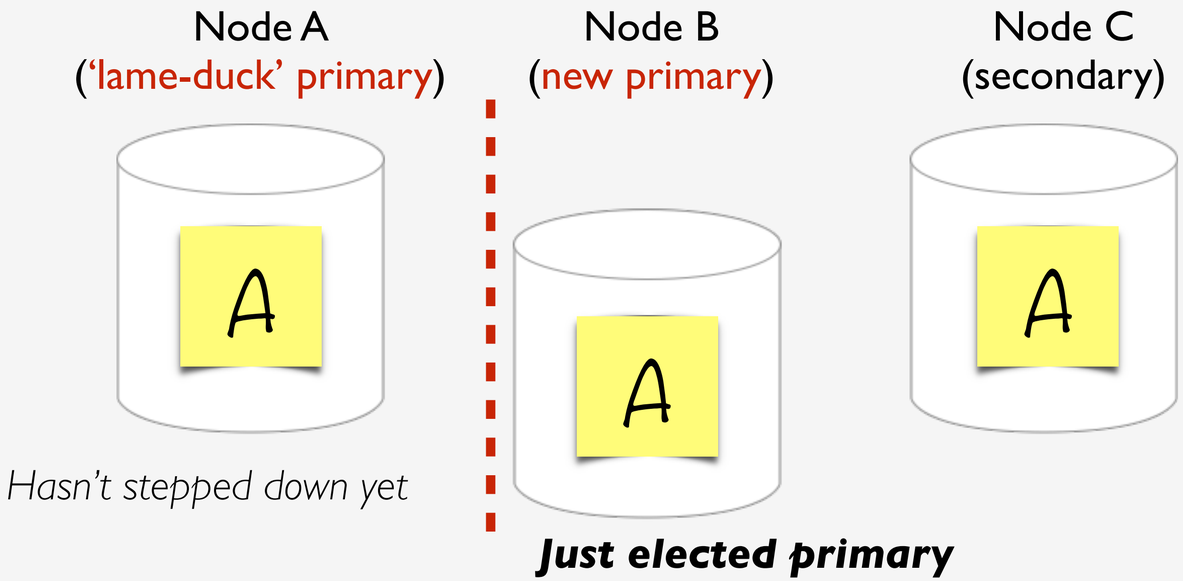


mongoDB

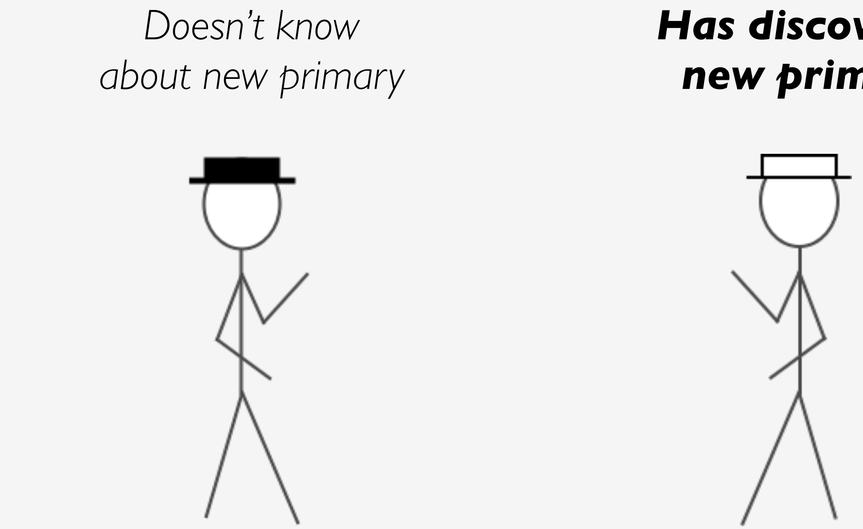
Meanwhile,
read from
nearest node

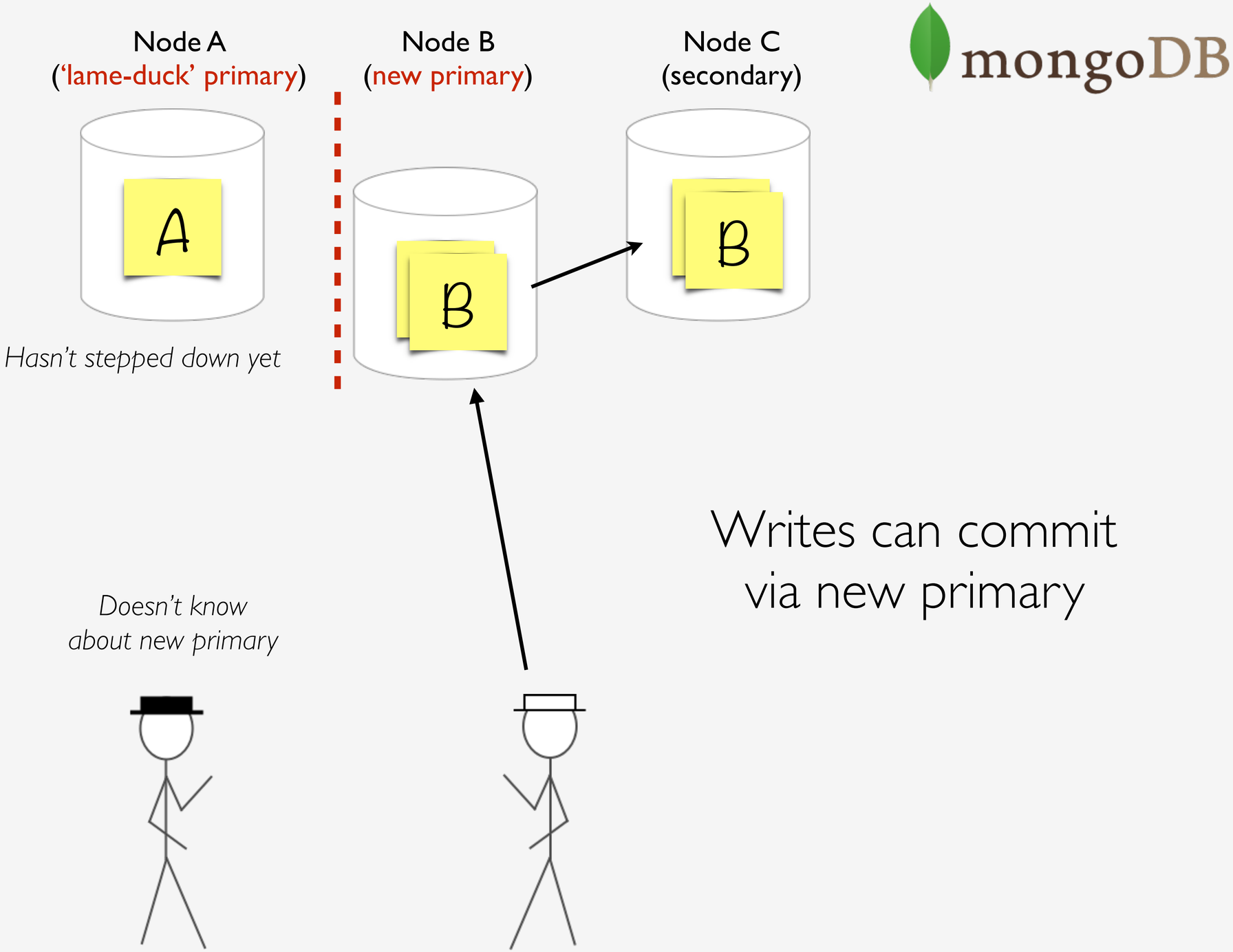
Still another ‘gotcha’...

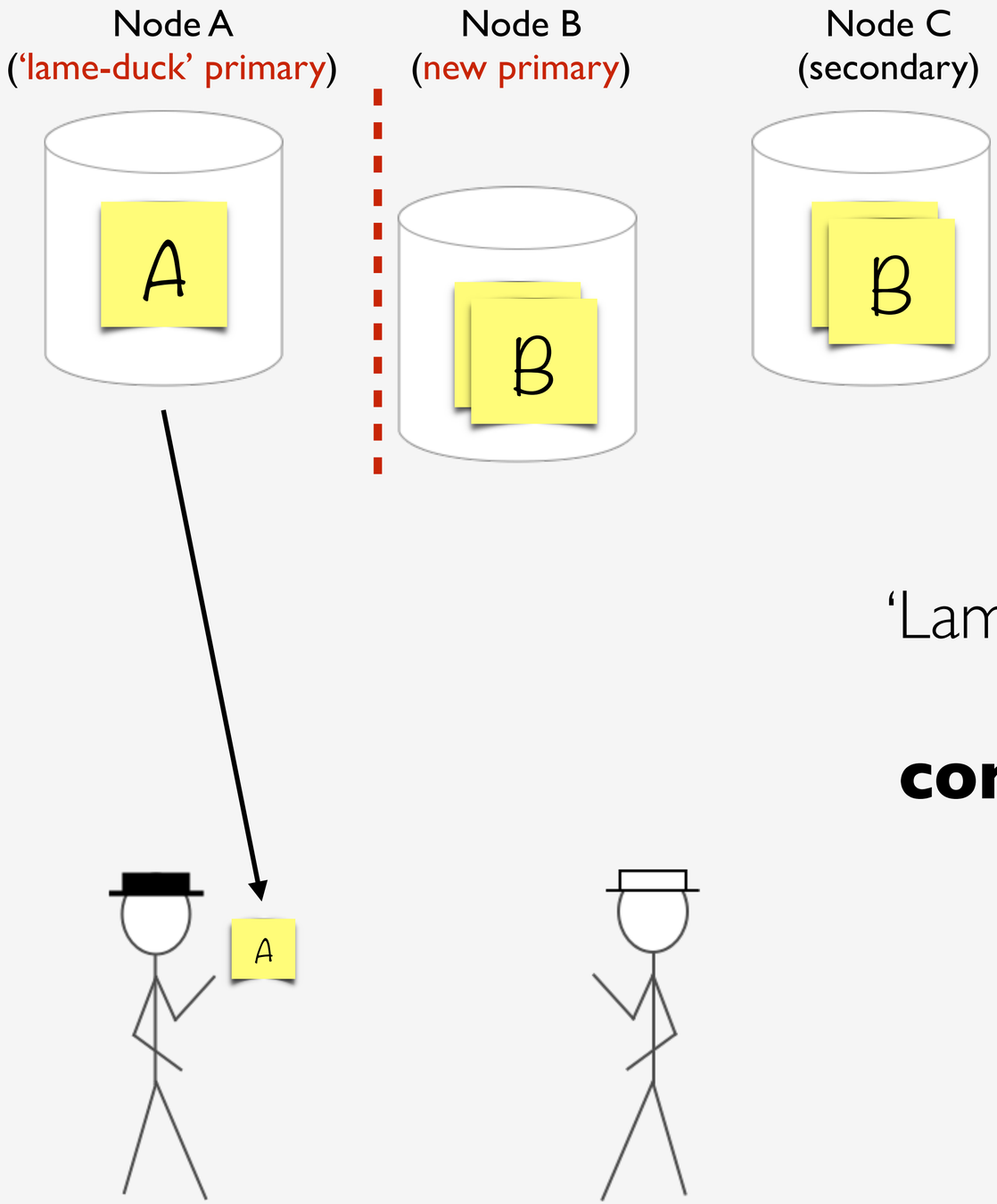
Partition detection race!


Node A
('lame-duck' primary)
Node B
(new primary)
Node C
(secondary)
A
A
A
Hasn't stepped down yet
Just elected primary

mongoDB

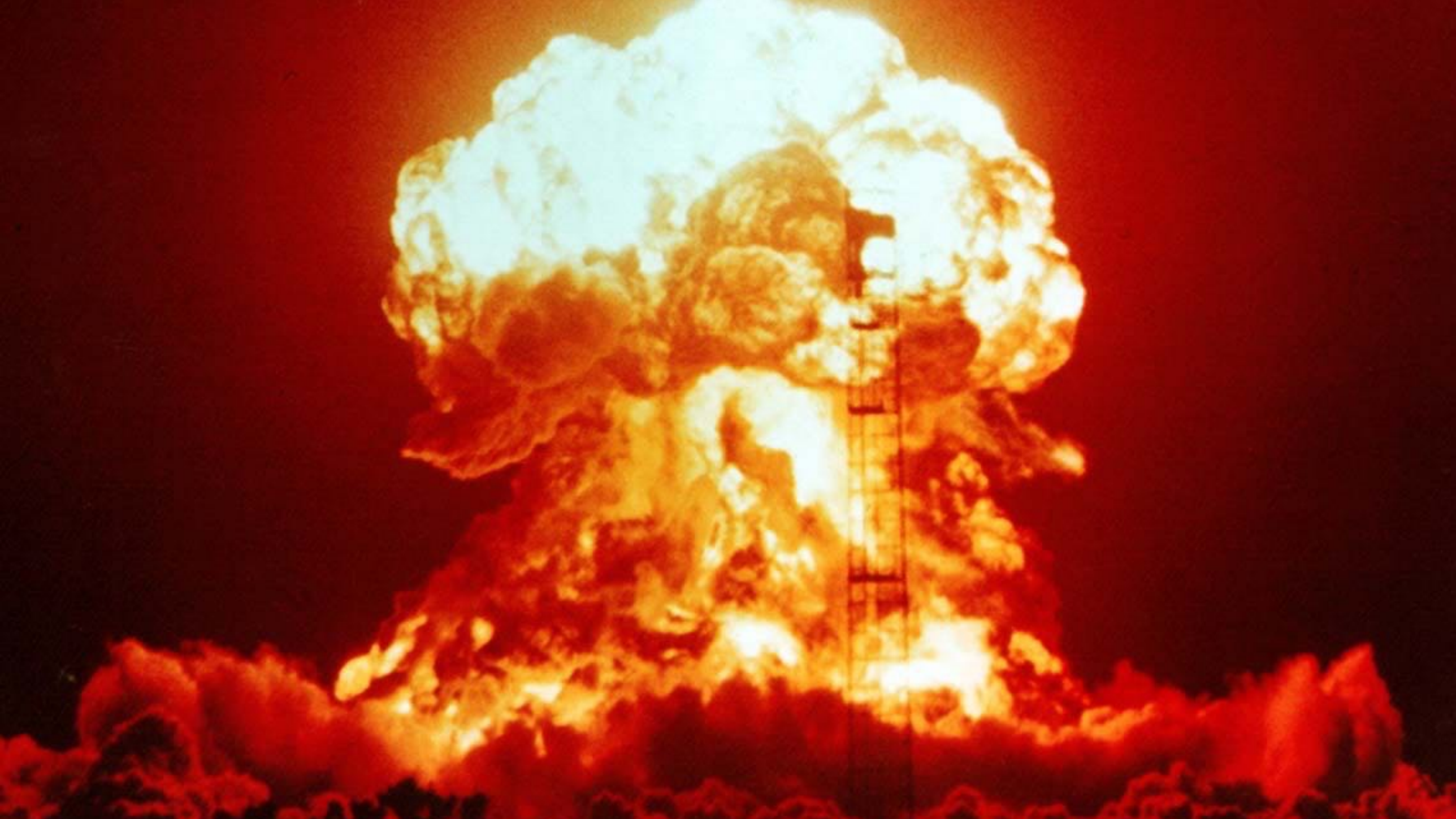

Doesn't know
about new primary
Has discovered
new primary


Node A
('lame-duck' primary)
Node B
(new primary)
Node C
(secondary)
A
B
B
Hasn't stepped down yet
Doesn't know
about new primary
Writes can commit
via new primary



'Lame-duck' primary returns old **committed** data

What if this isn’t OK?



‘Quorum read’

Read-via-write


```
# find_one_and_update (CAS*)

$mc = MongoDB->connect( $url,
    { w => ‘majority’ }
);


...

$doc = $coll->find_one_and_update(
    { _id => $id },
    { ‘$inc’ => { _dummy => 1 } },
);
```

Take aways...

CAP is simplistic

Reality is complex

Needs are
application specific

When writing, consider...

Durability
Convergence
Error recovery

When reading, consider...

Recency vs durability
Recency vs latency
Nuclear option

Questions?

Email: david@mongodb.com
Twitter/IRC: @xdg

Photo credits:

- 9:00 AM: Mitch Martinez https://www.youtube.com/watch?v=ScDYZXIcihc
- Camel race: Jason Mrachina https://www.flickr.com/photos/w4nd3rl0st/5944487713 by-nc-nd
- Partition: Marc Venezia - Own work Picture taken during a personal trip in Middle-East, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1146955l
- Eric Brewer: By Vera de Kok - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=39817816
- Daniel Abadi: @daniel_abadi profile picture https://pbs.twimg.com/profile_images/254073578/head2.jpg
- Fireball: By Photo courtesy of National Nuclear Security Administration / Nevada Site Office - This image is available from the National Nuclear Security Administration Nevada Site Office Photo Library under number XX-34. Public Domain, https://commons.wikimedia.org/w/index.php?curid=190949