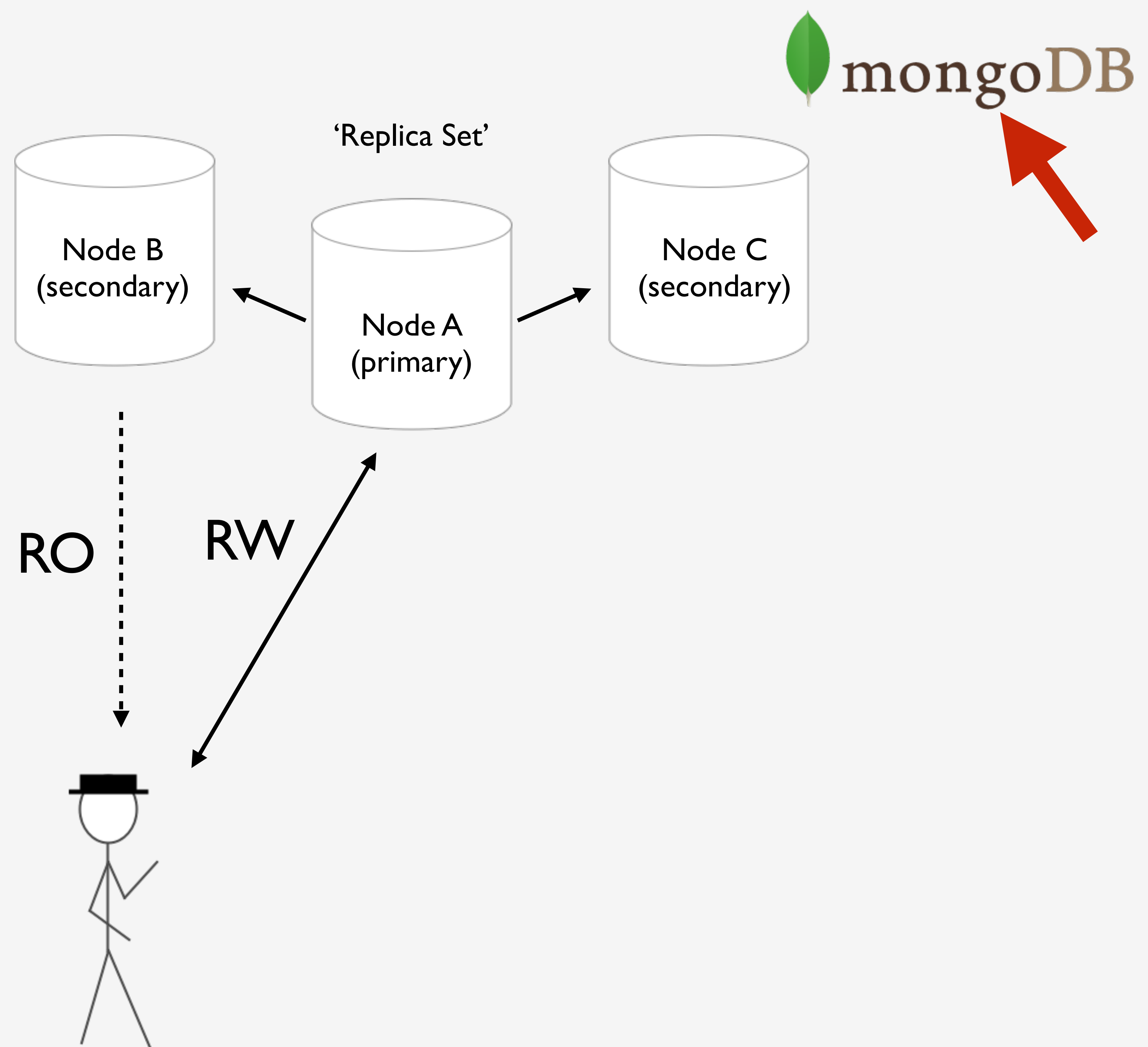


What is a  
distributed system?

Data processing spread  
over time & space



memcached

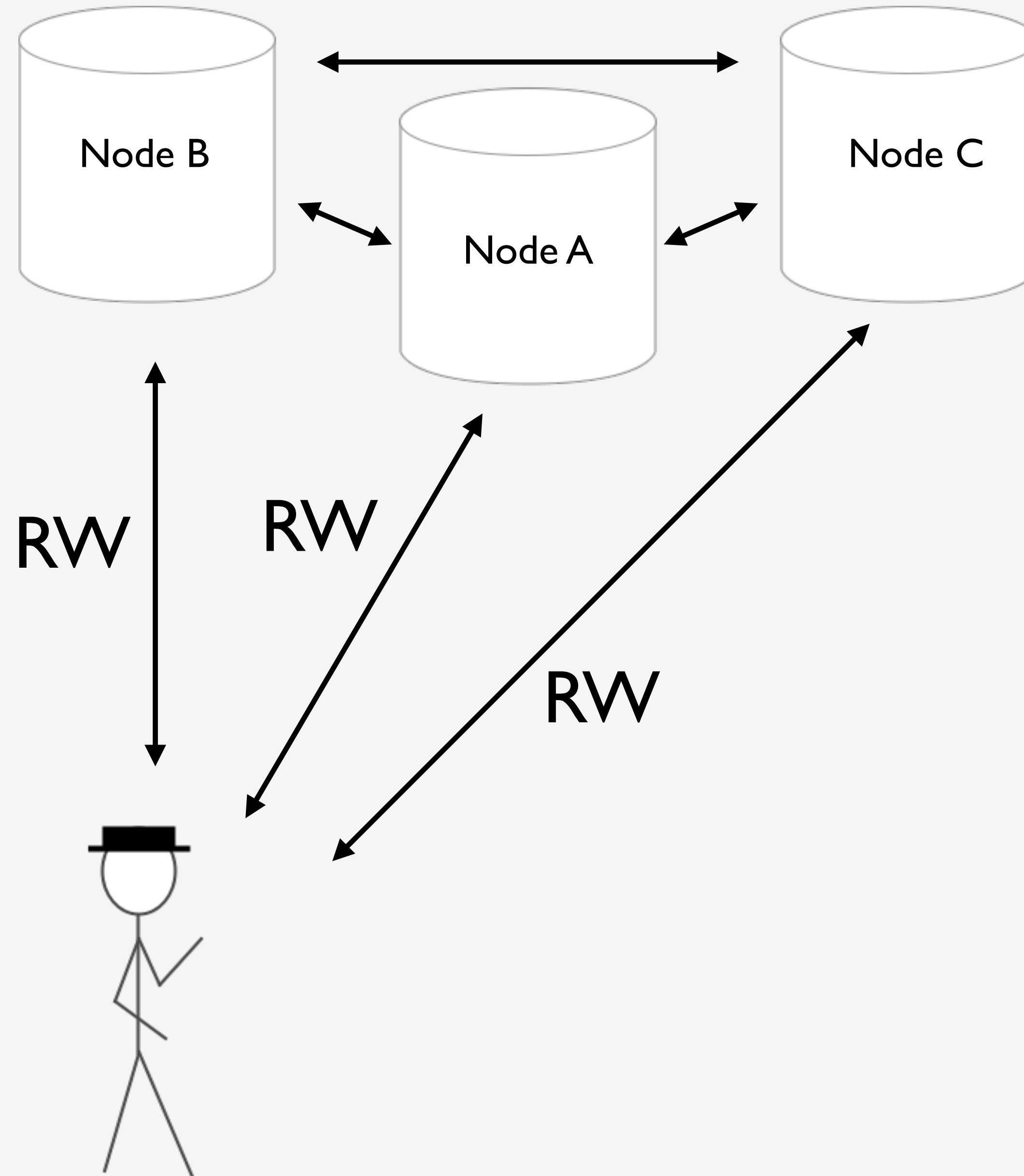


RW

RW



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



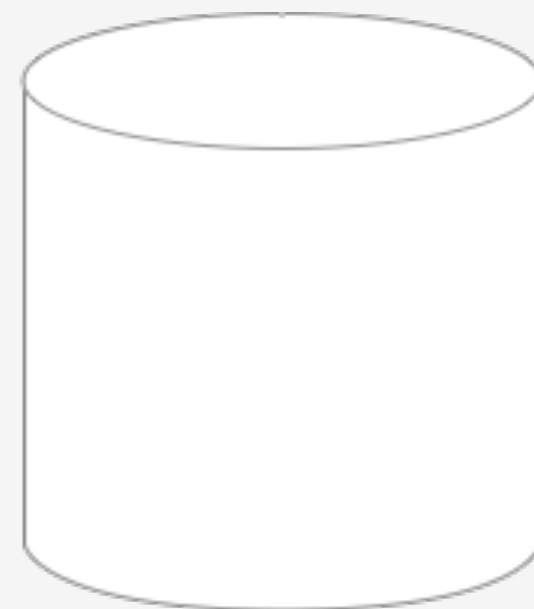
Crash

Packet loss

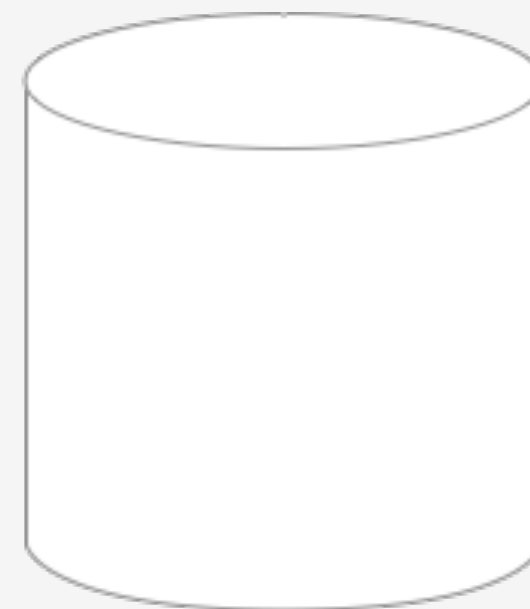
Garbage collection

Process swapped out

DB



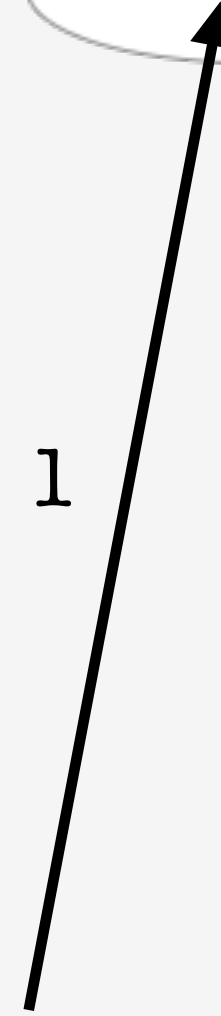
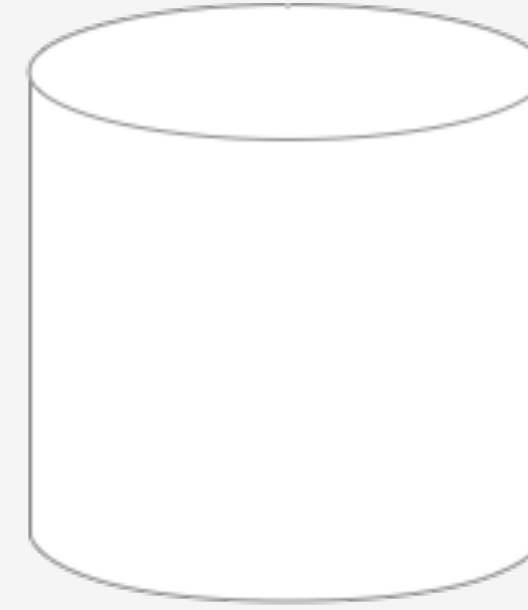
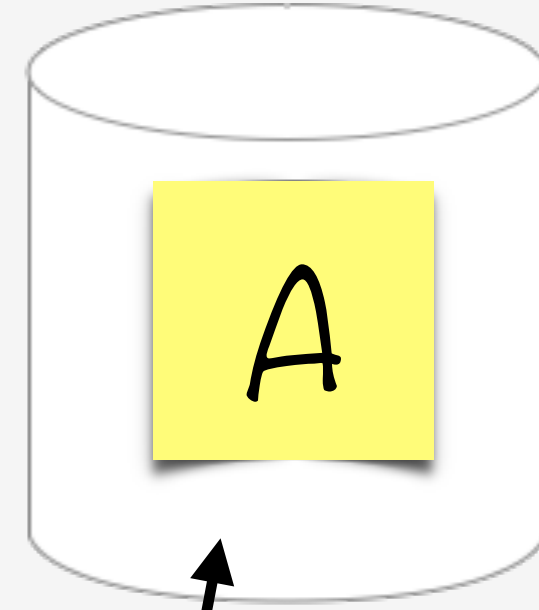
memcached



*Update...*

**DB**

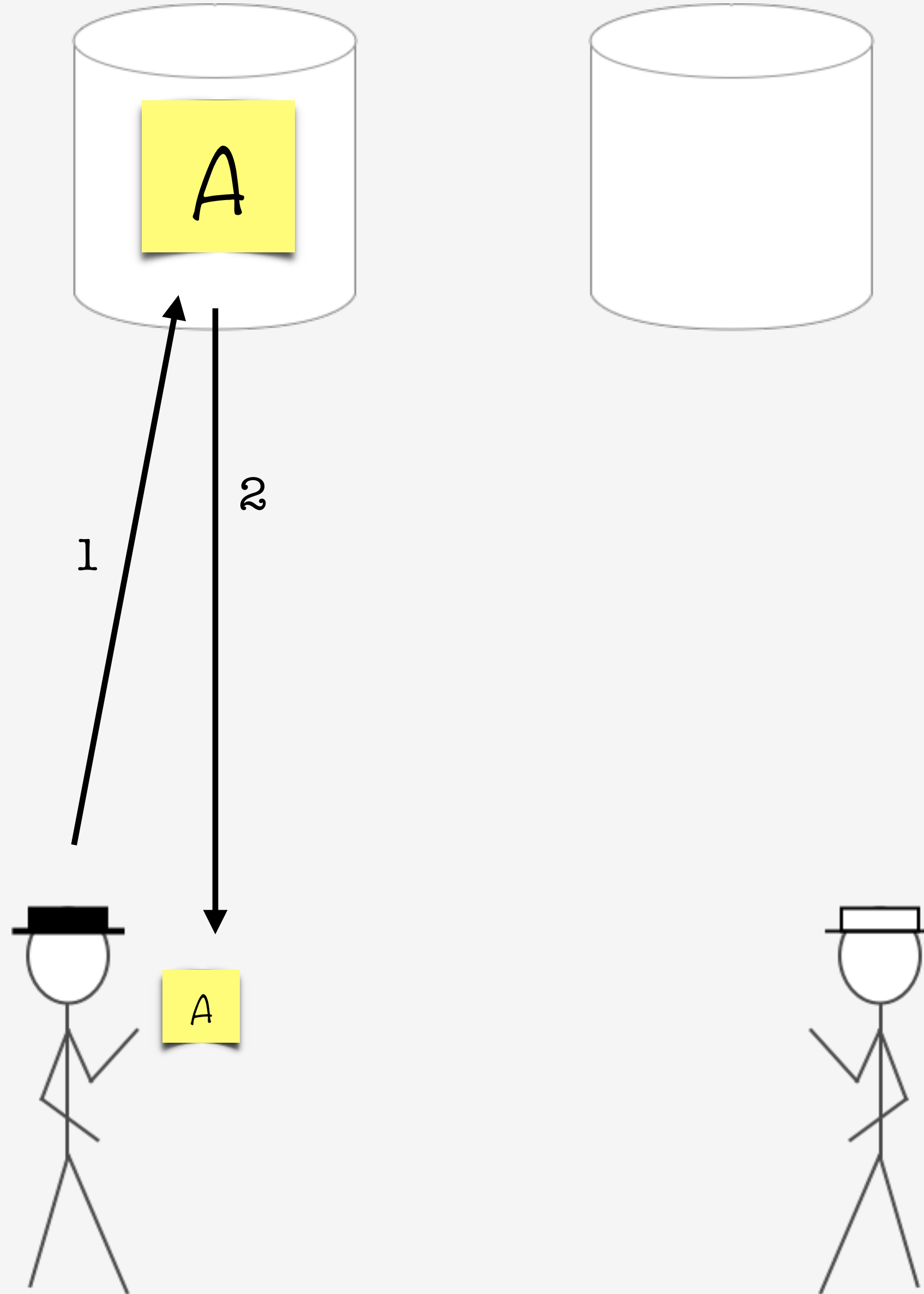
**memcached**



*Update...*

**DB**

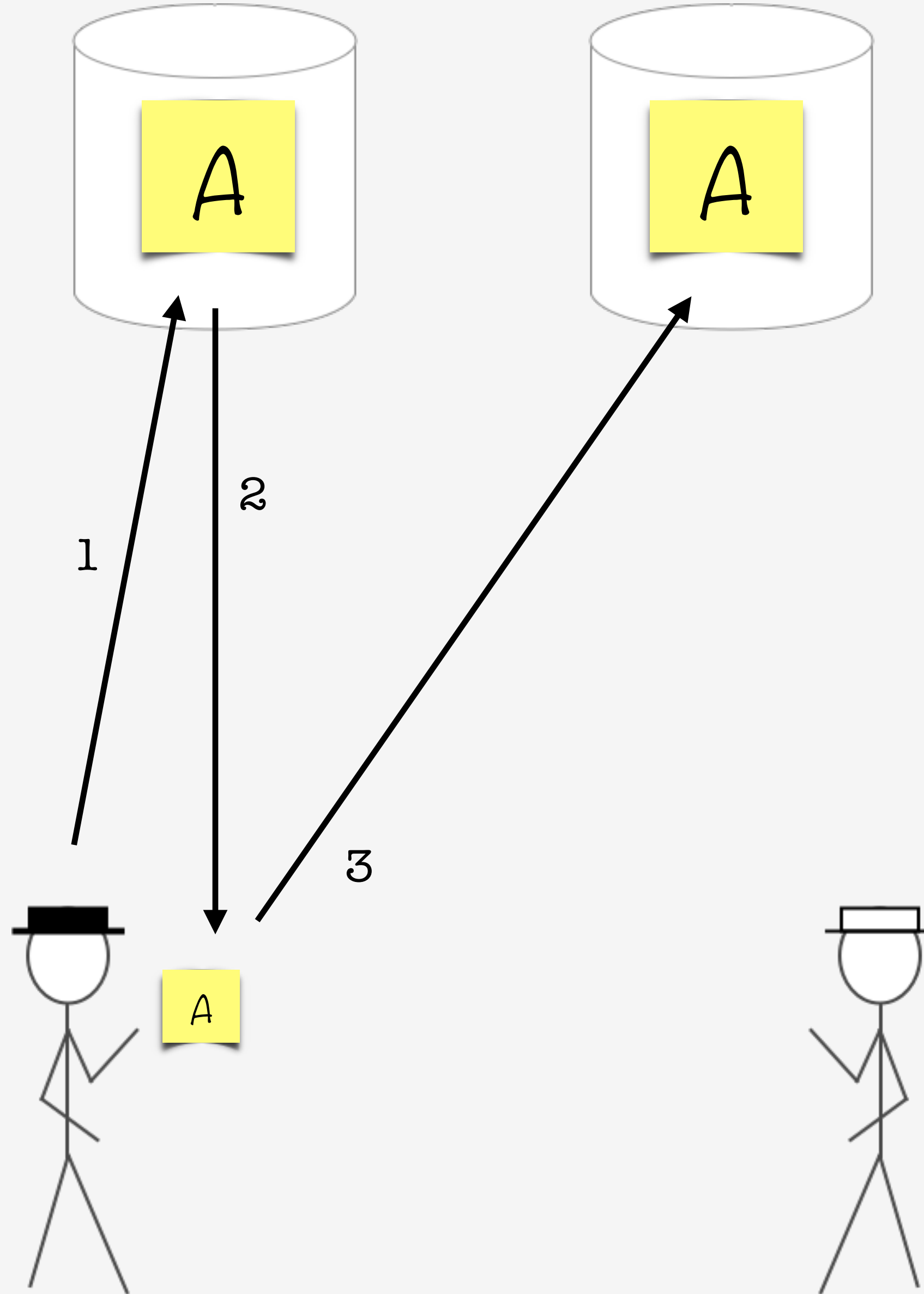
**memcached**



*Update...*

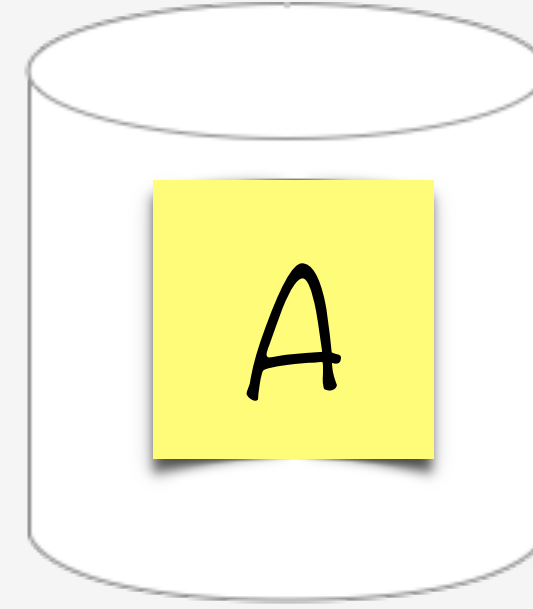
**DB**

**memcached**

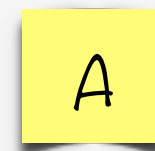
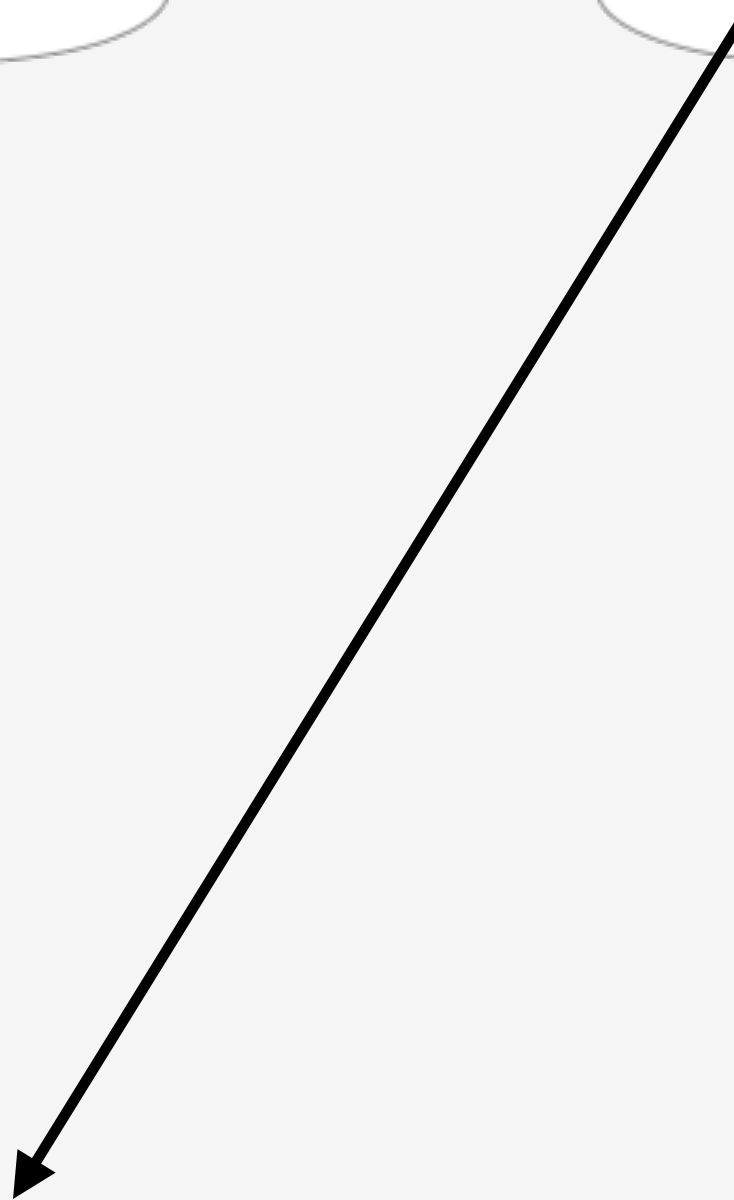
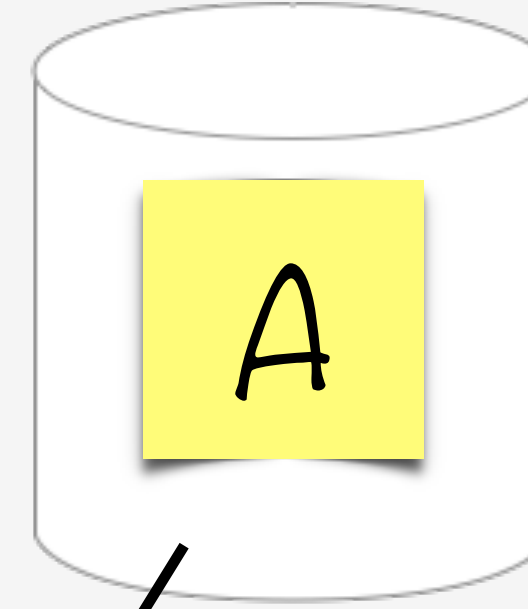


*Read...*

**DB**



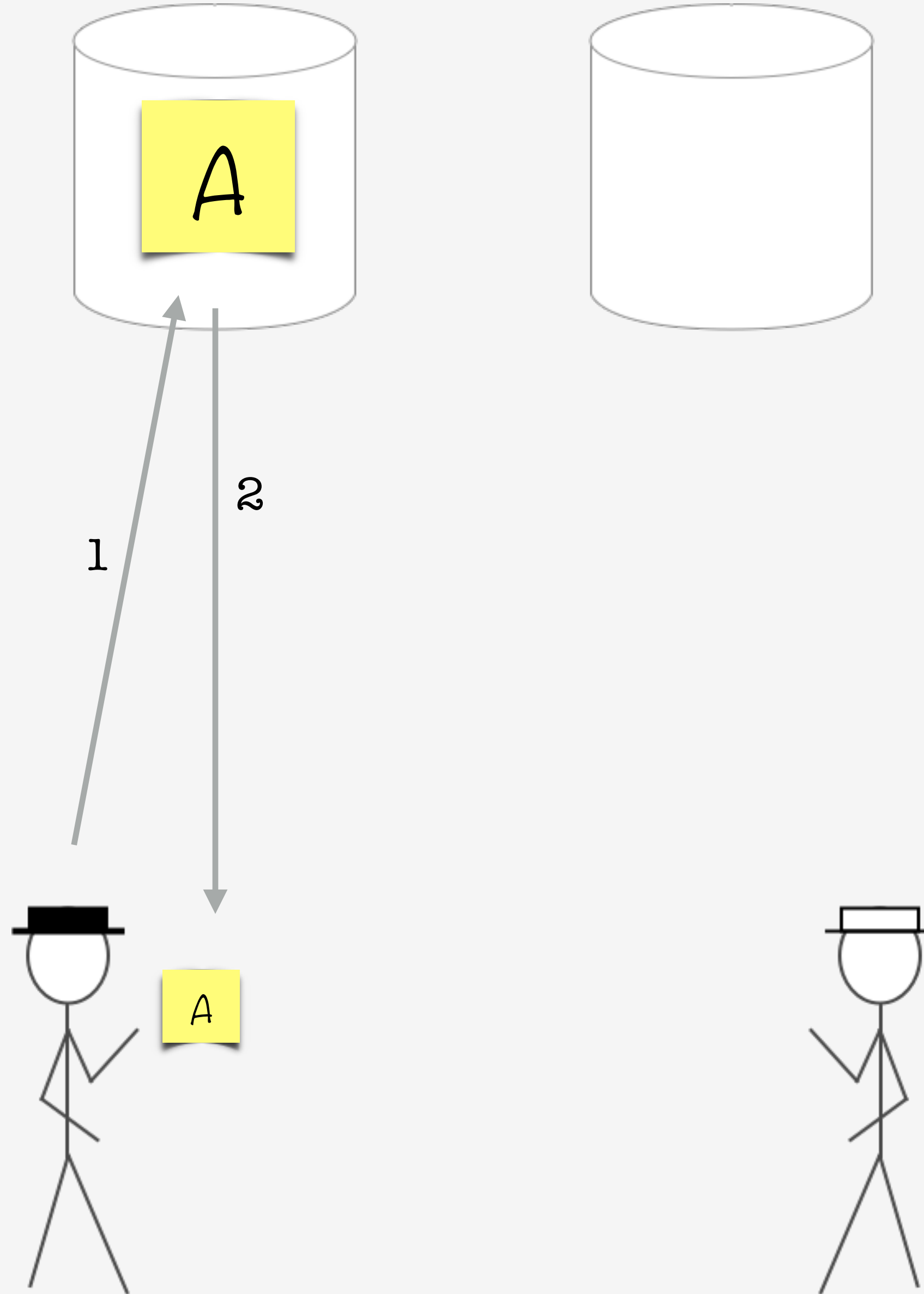
**memcached**



How can that go wrong?

DB

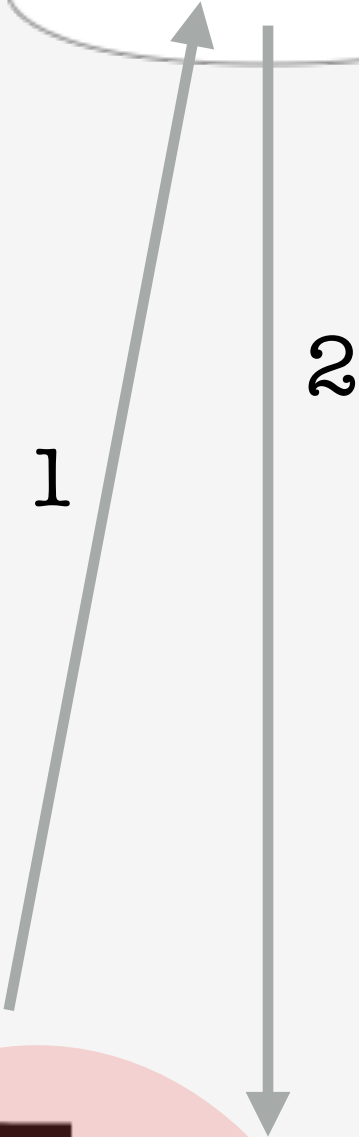
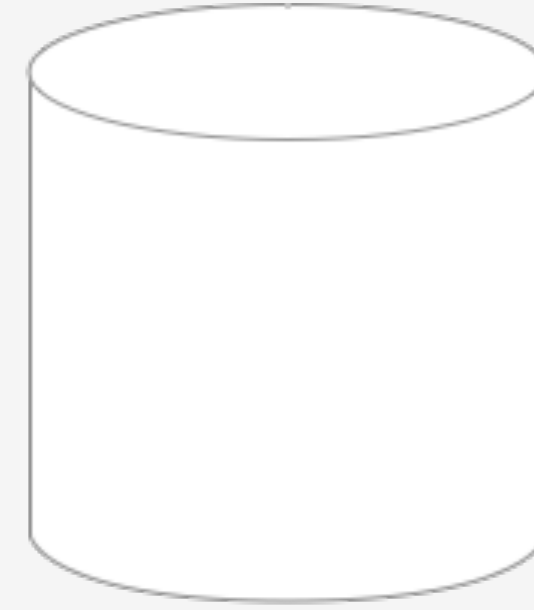
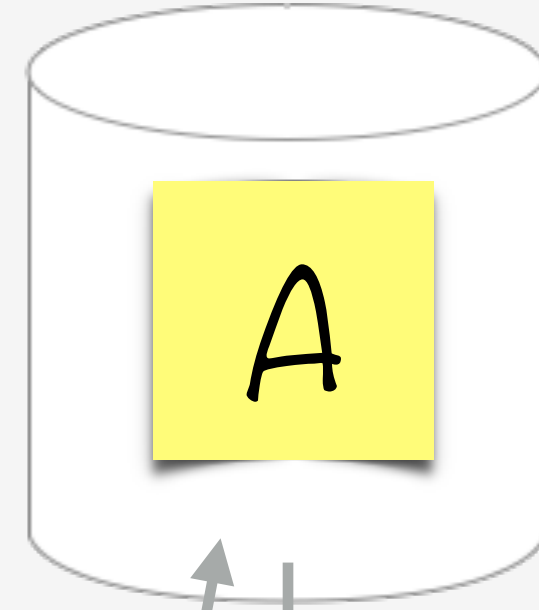
memcached





DB

memcached

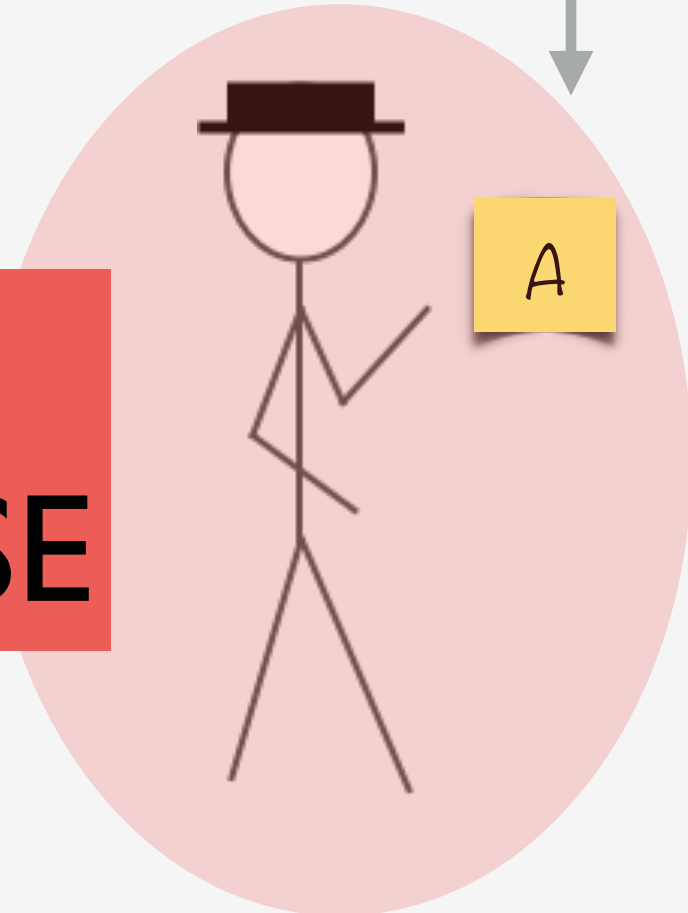


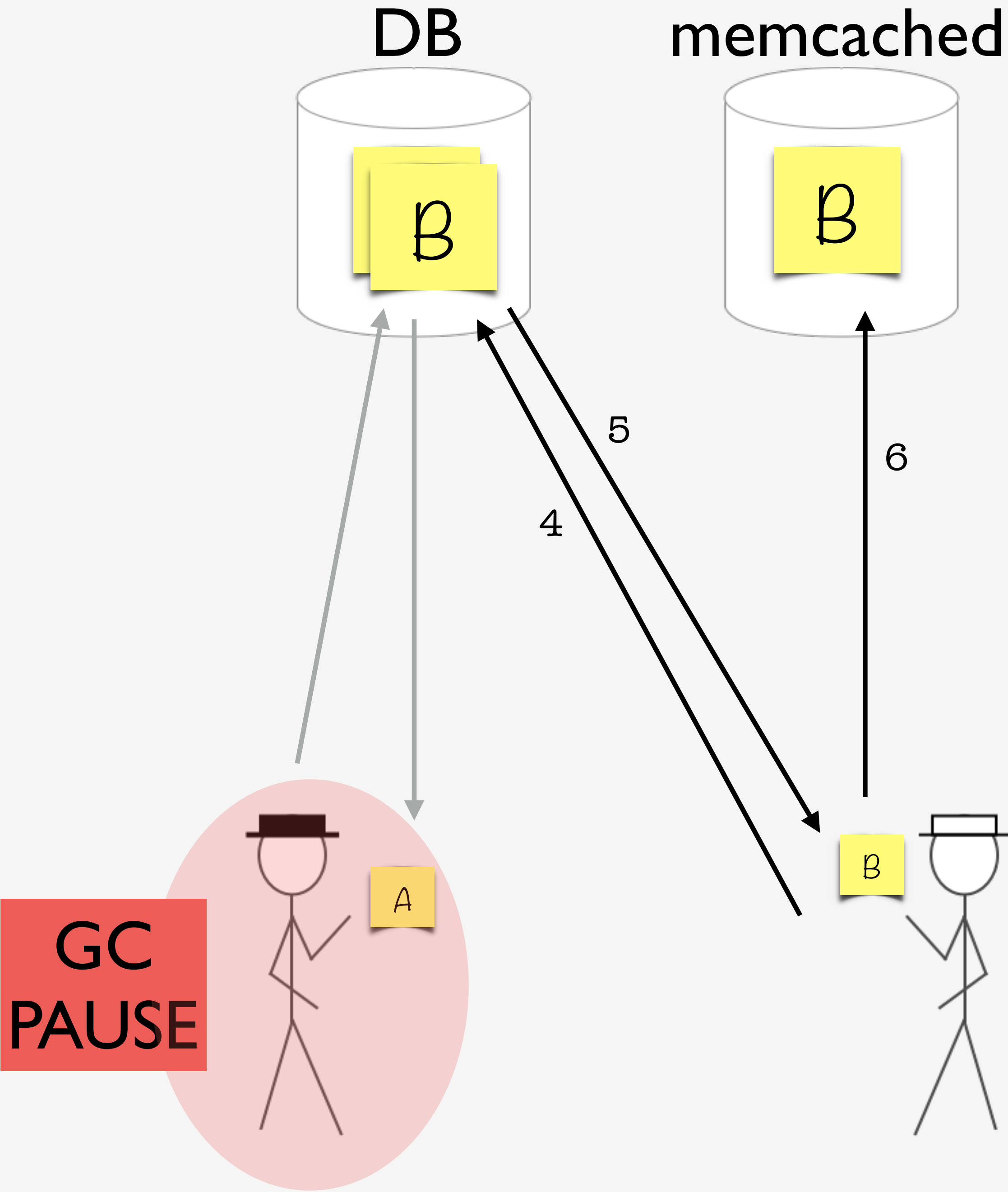
1

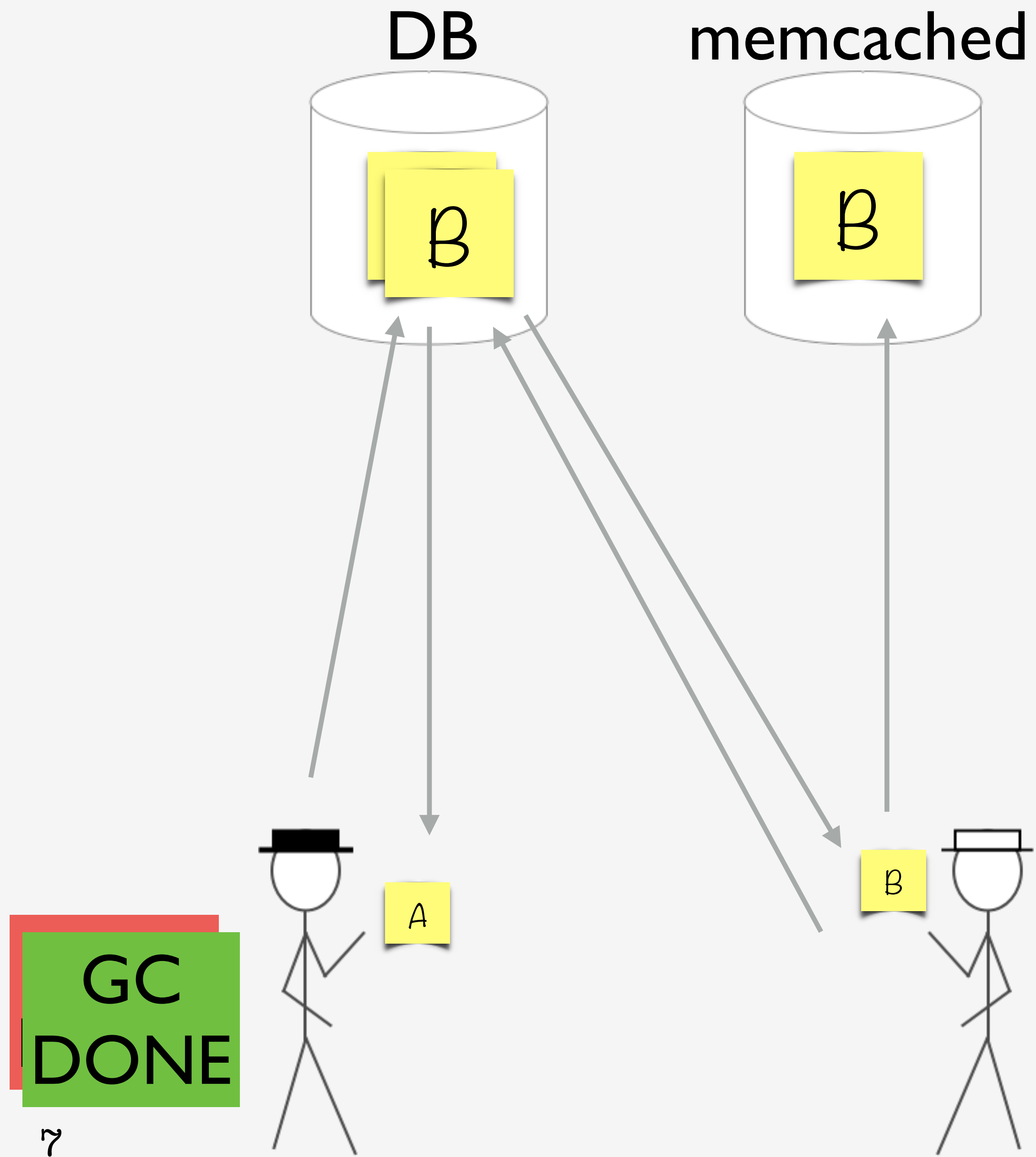
2

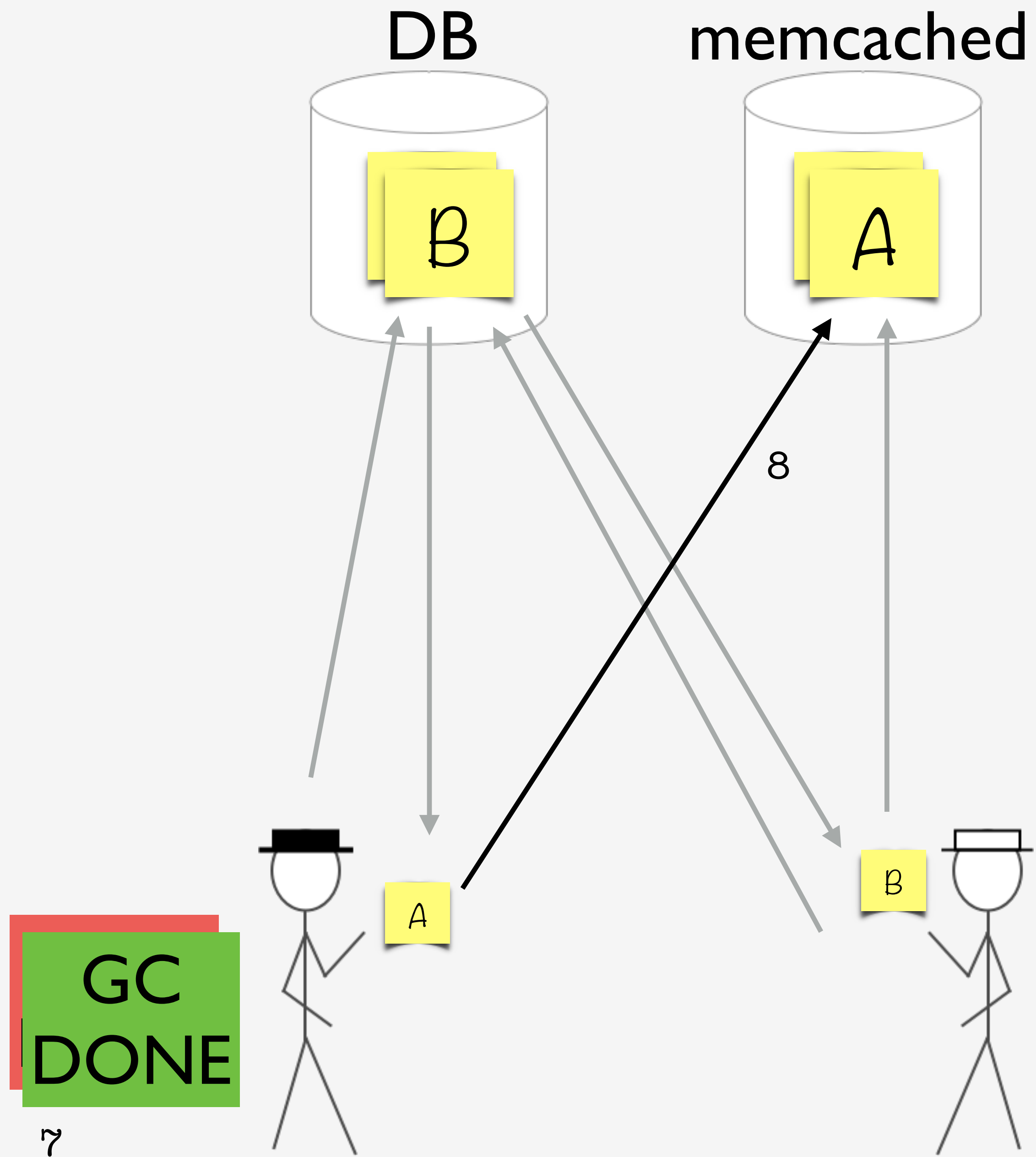
3

GC  
PAUSE

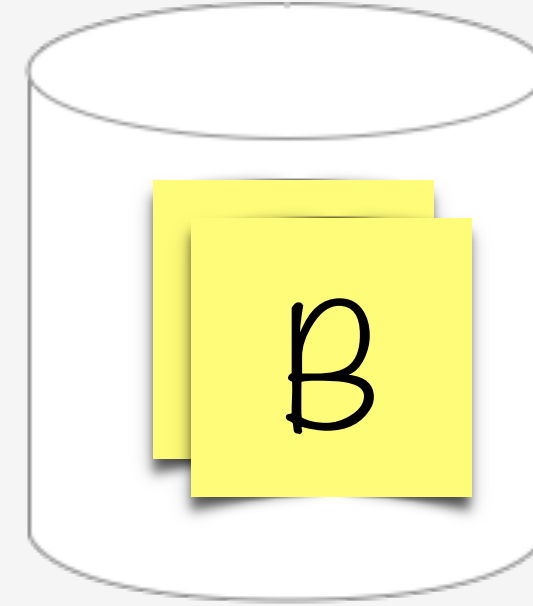




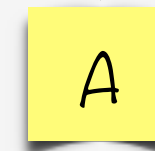
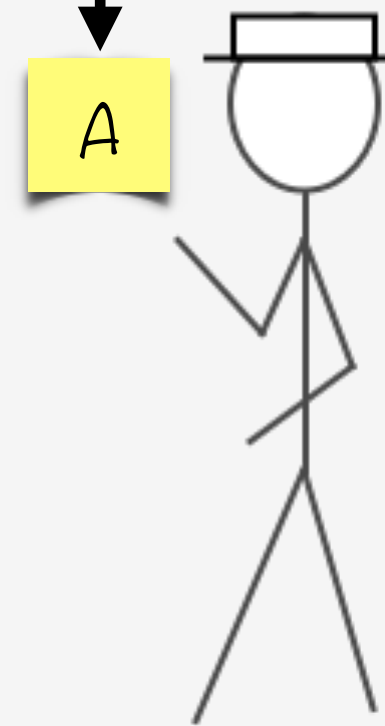
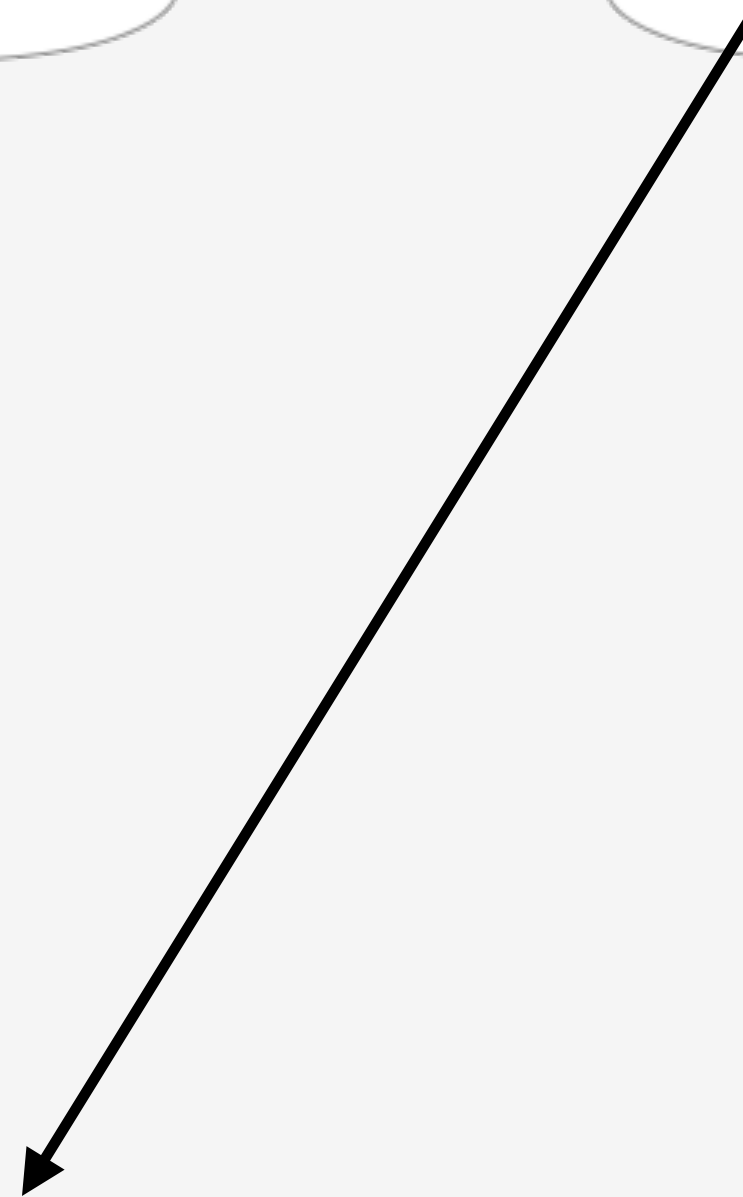
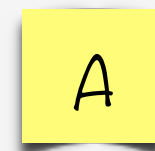
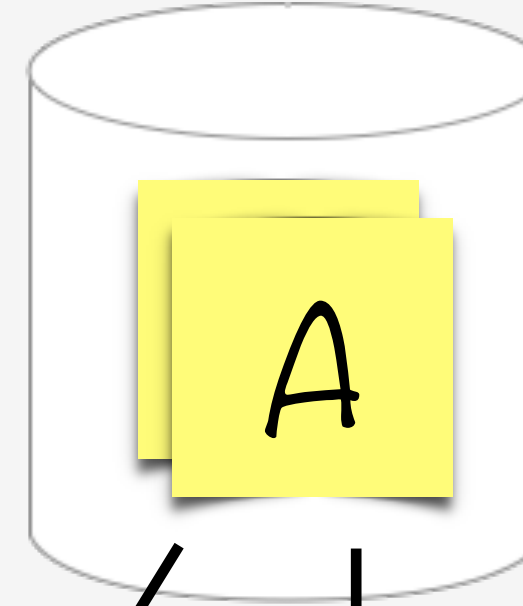




DB



memcached



DB + memcached is not  
a 'good' system

What makes a good  
distributed system?







# CAP Theorem

**C**onsistency

**A**vailability

**P**artition 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.

# Ppartition 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

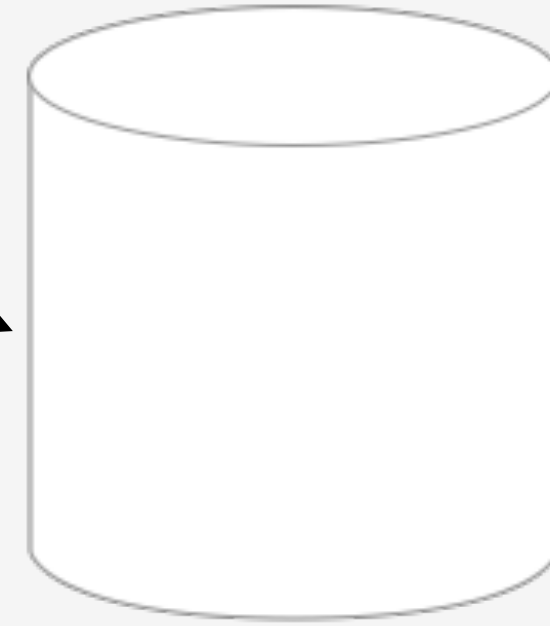
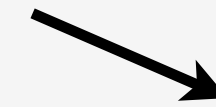
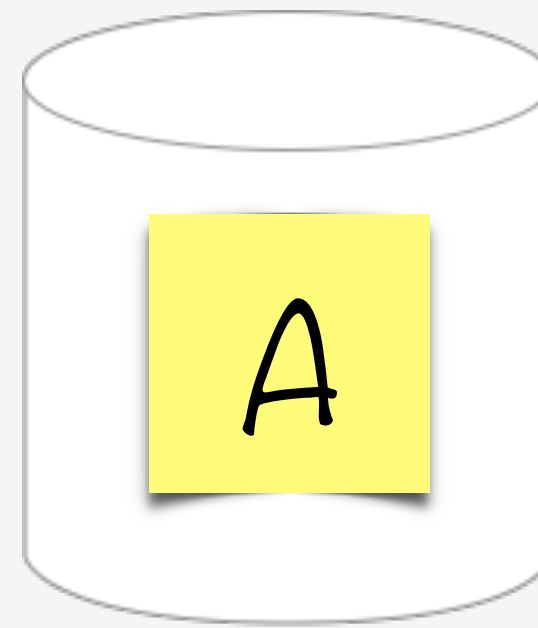


Do we know when  
writes are durable?

Node A  
(primary)

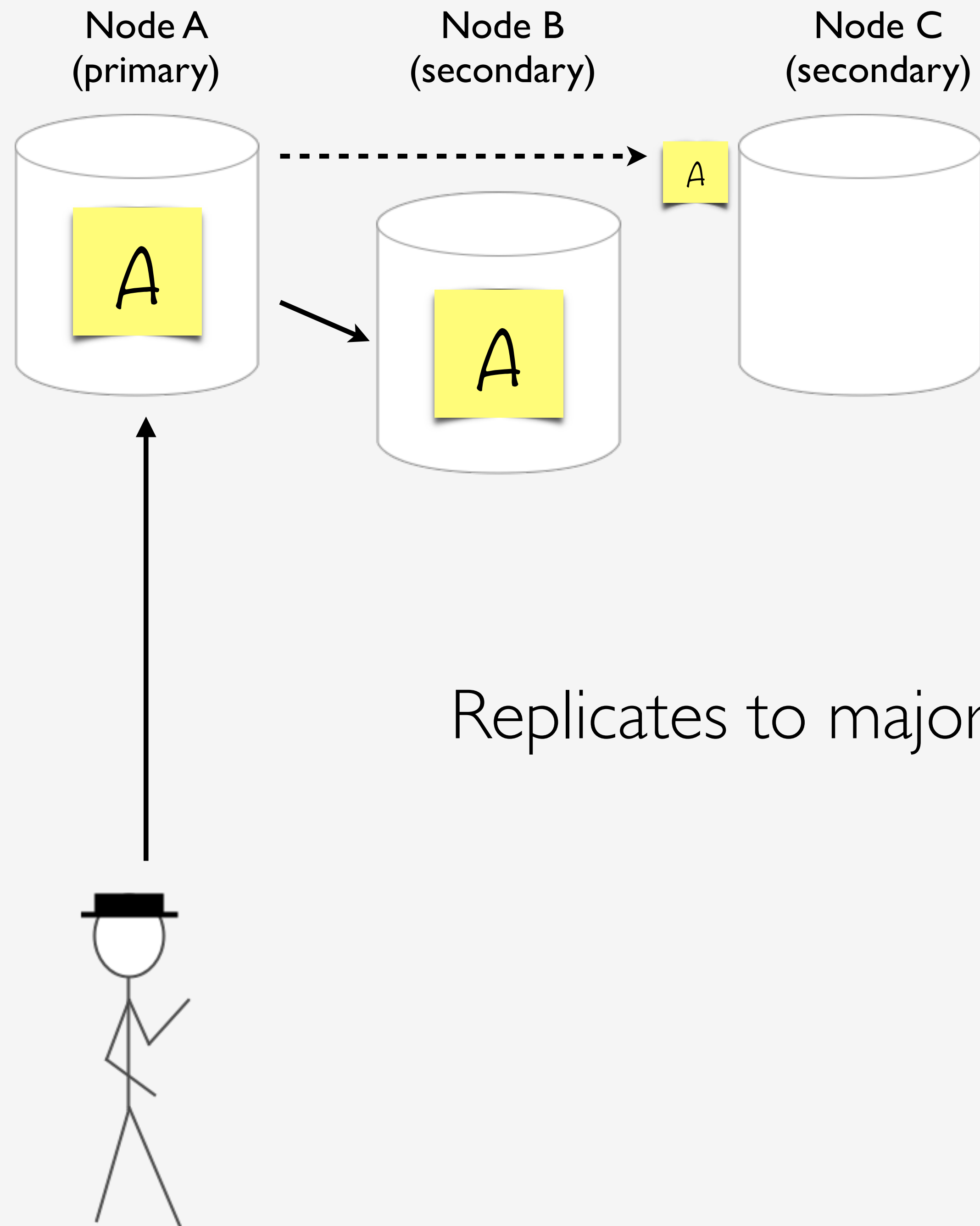
Node B  
(secondary)

Node C  
(secondary)

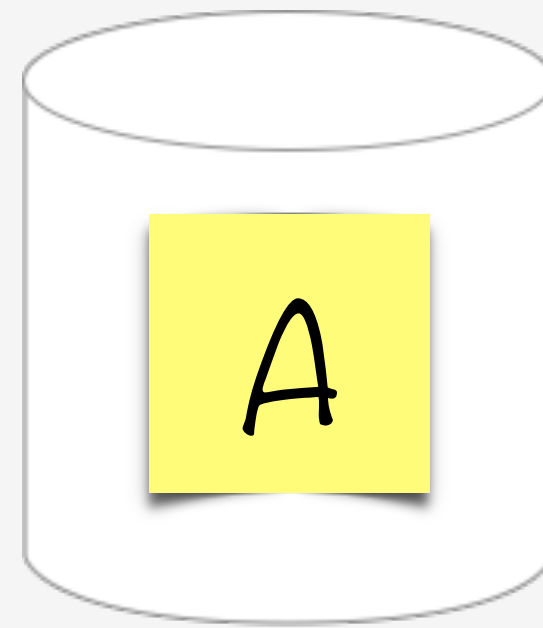


Write goes to primary

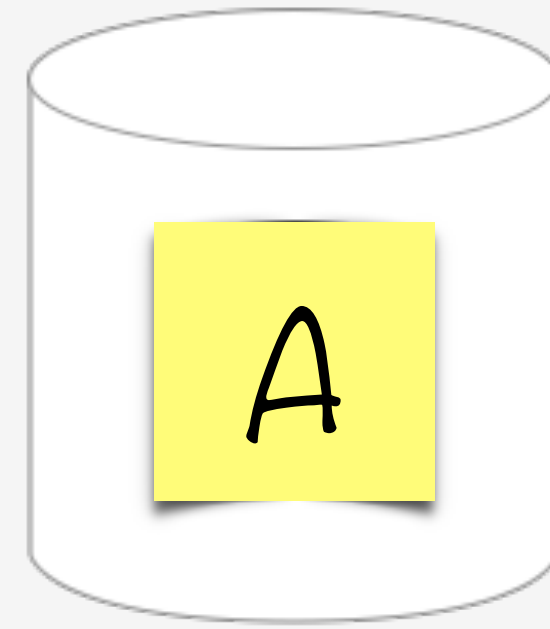




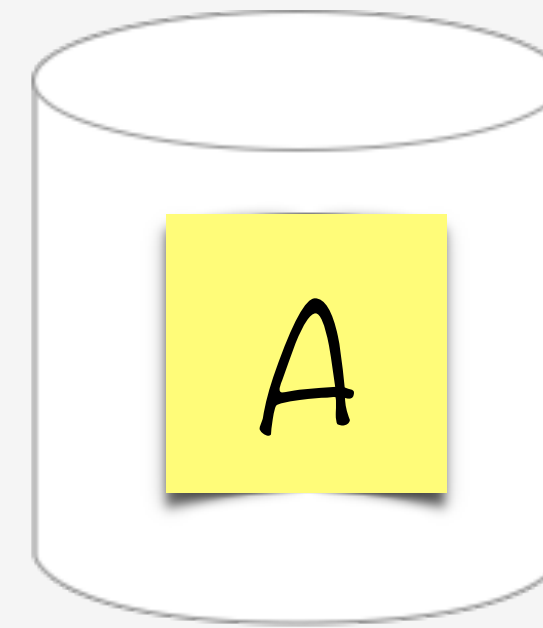
Node A  
(primary)



Node B  
(secondary)

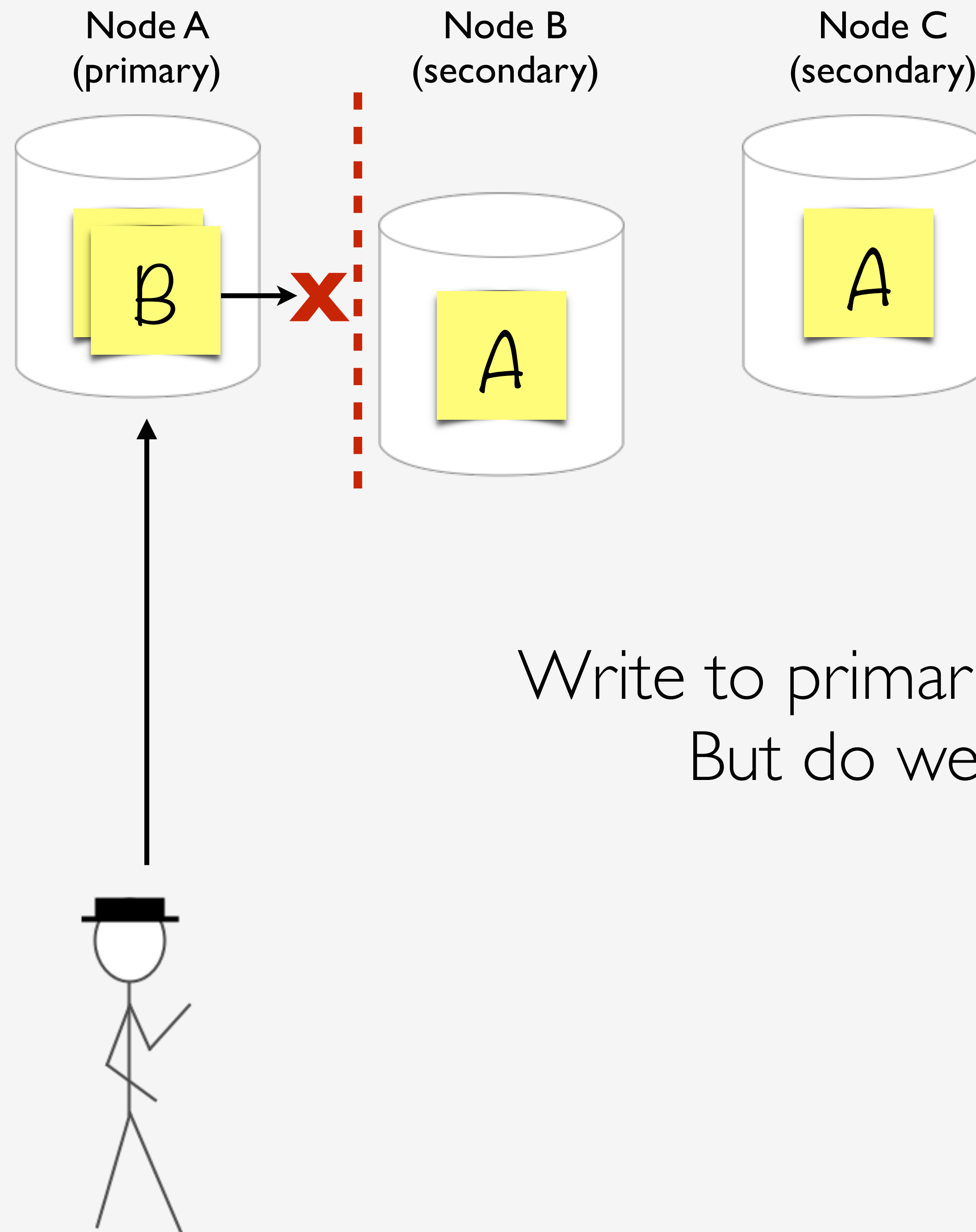


Node C  
(secondary)



Partition separates primary





```
# write concern
```

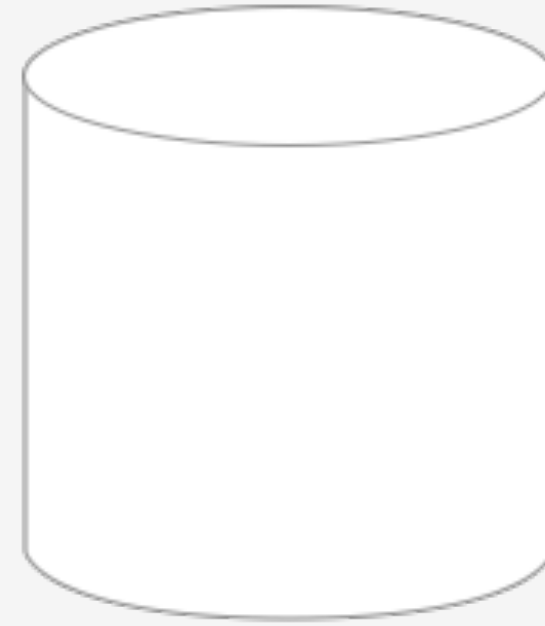
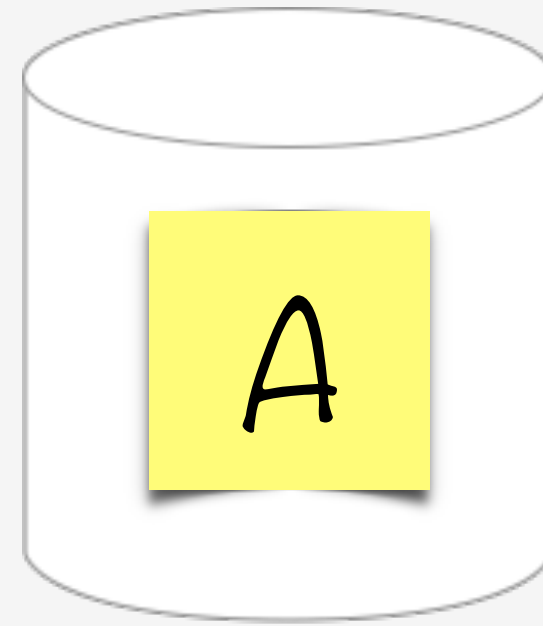
```
MongoDB->connect( $url,  
                  { w => 1 }  
);
```

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

Node A  
(primary)

Node B  
(secondary)

Node C  
(secondary)

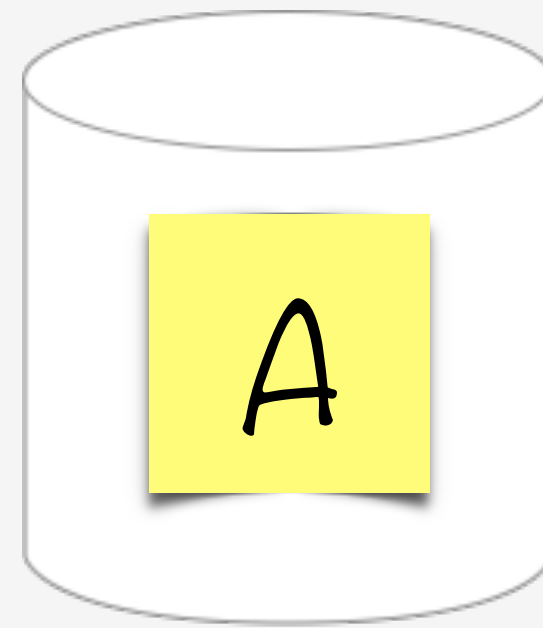


{w => 1}

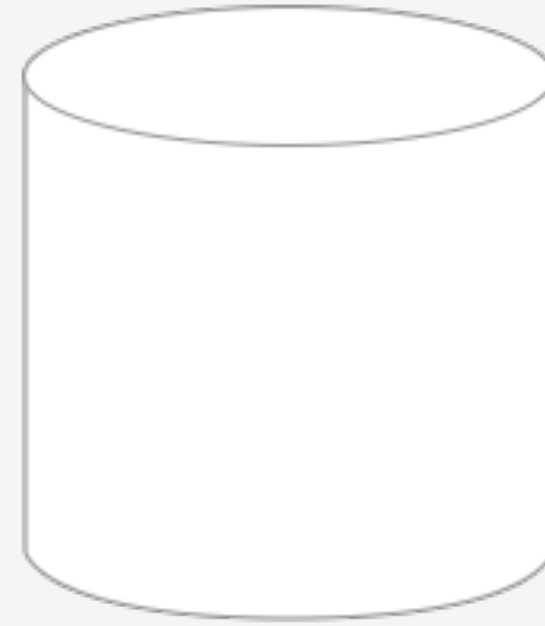


OK!

Node A  
(primary)



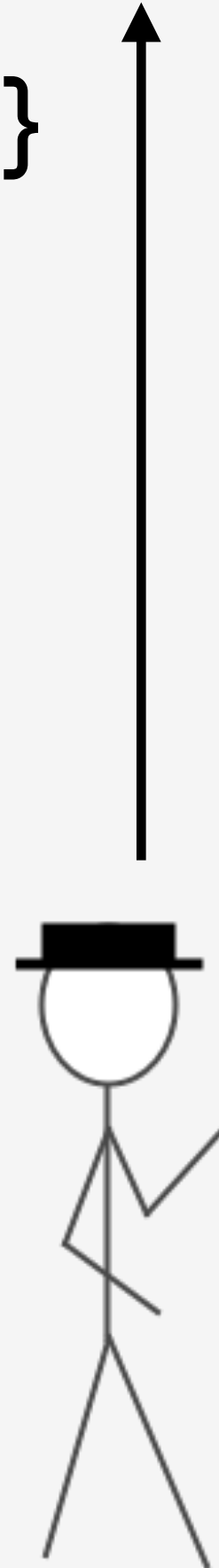
Node B  
(secondary)



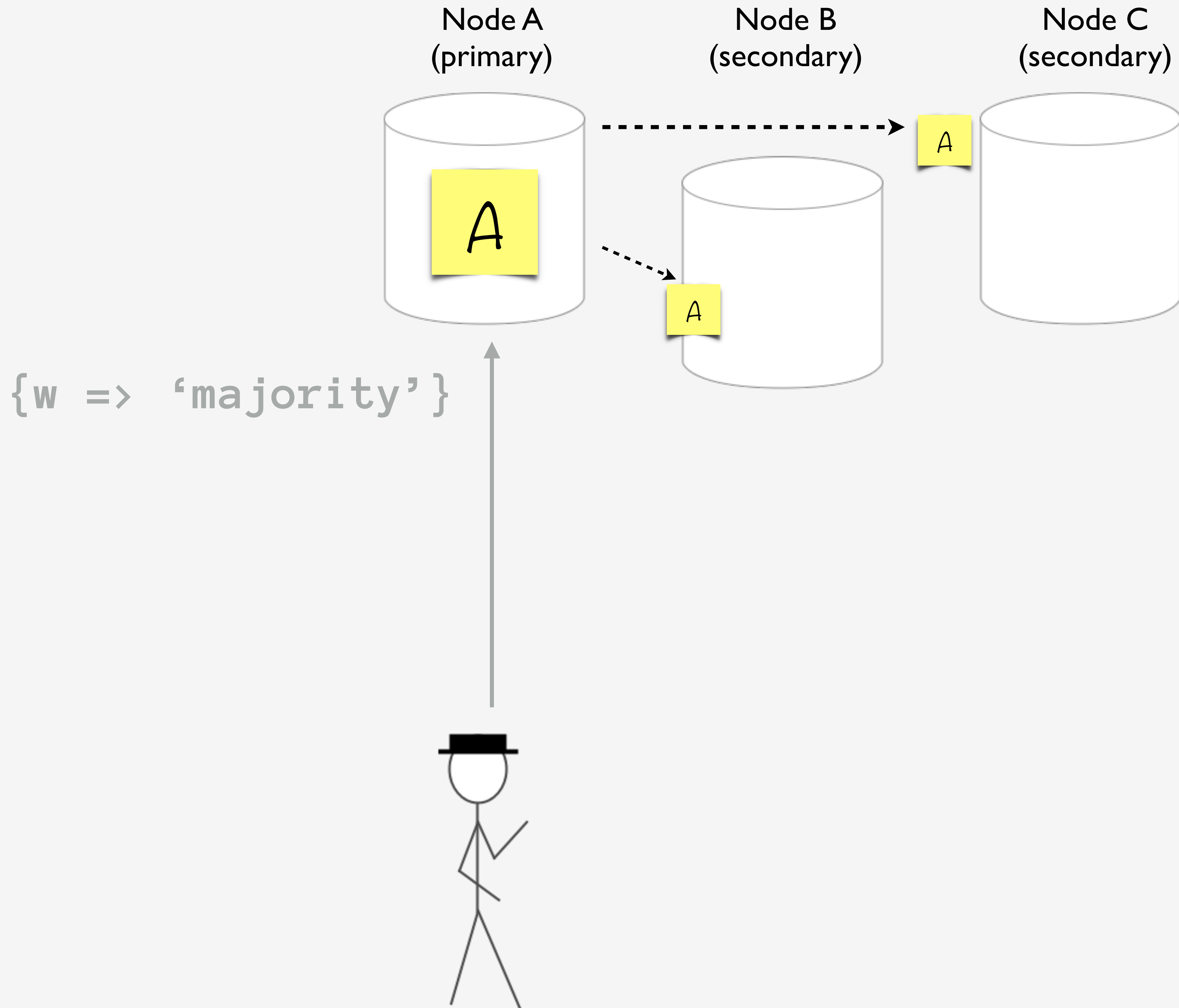
Node C  
(secondary)

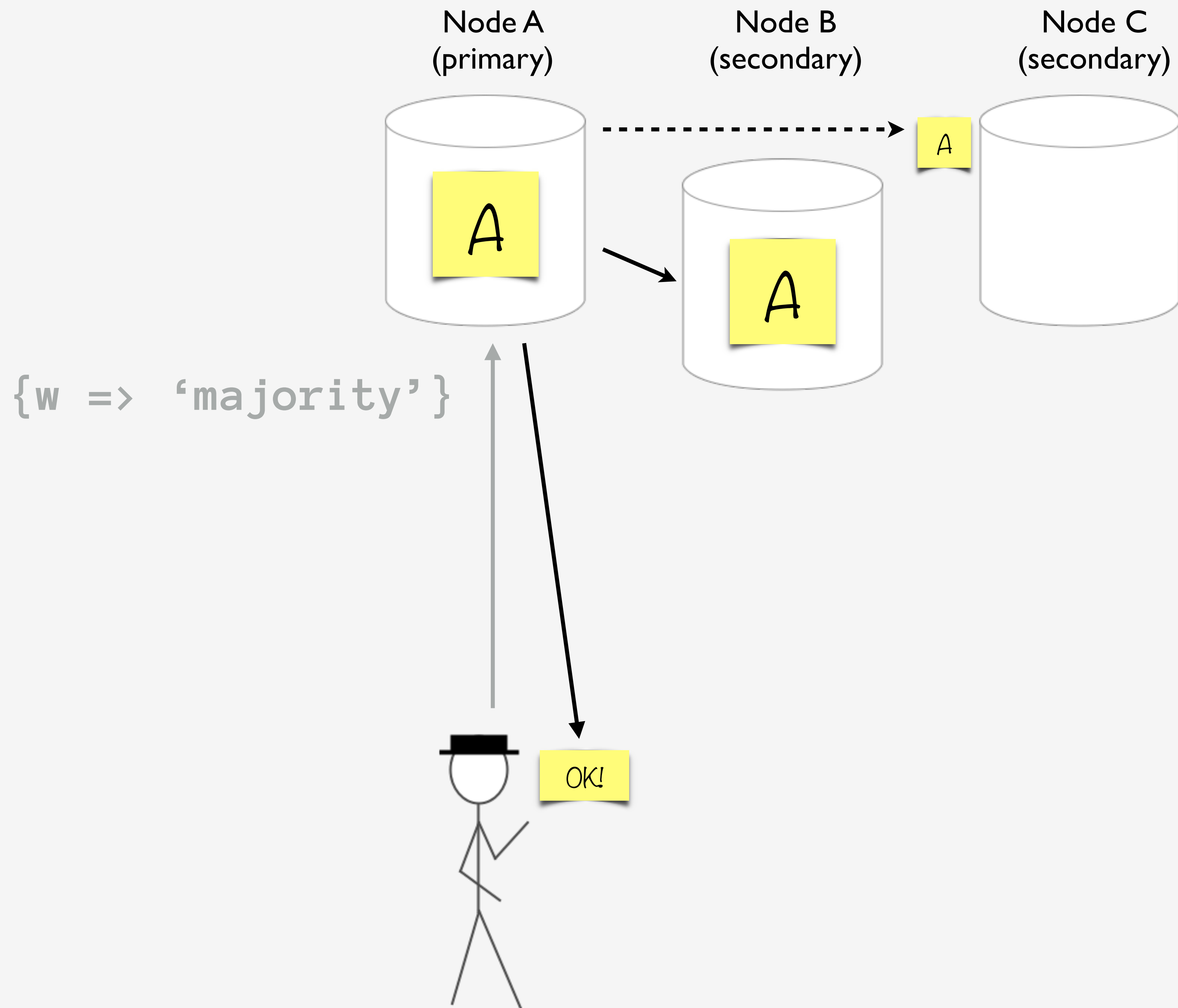


`{w => 'majority'}`

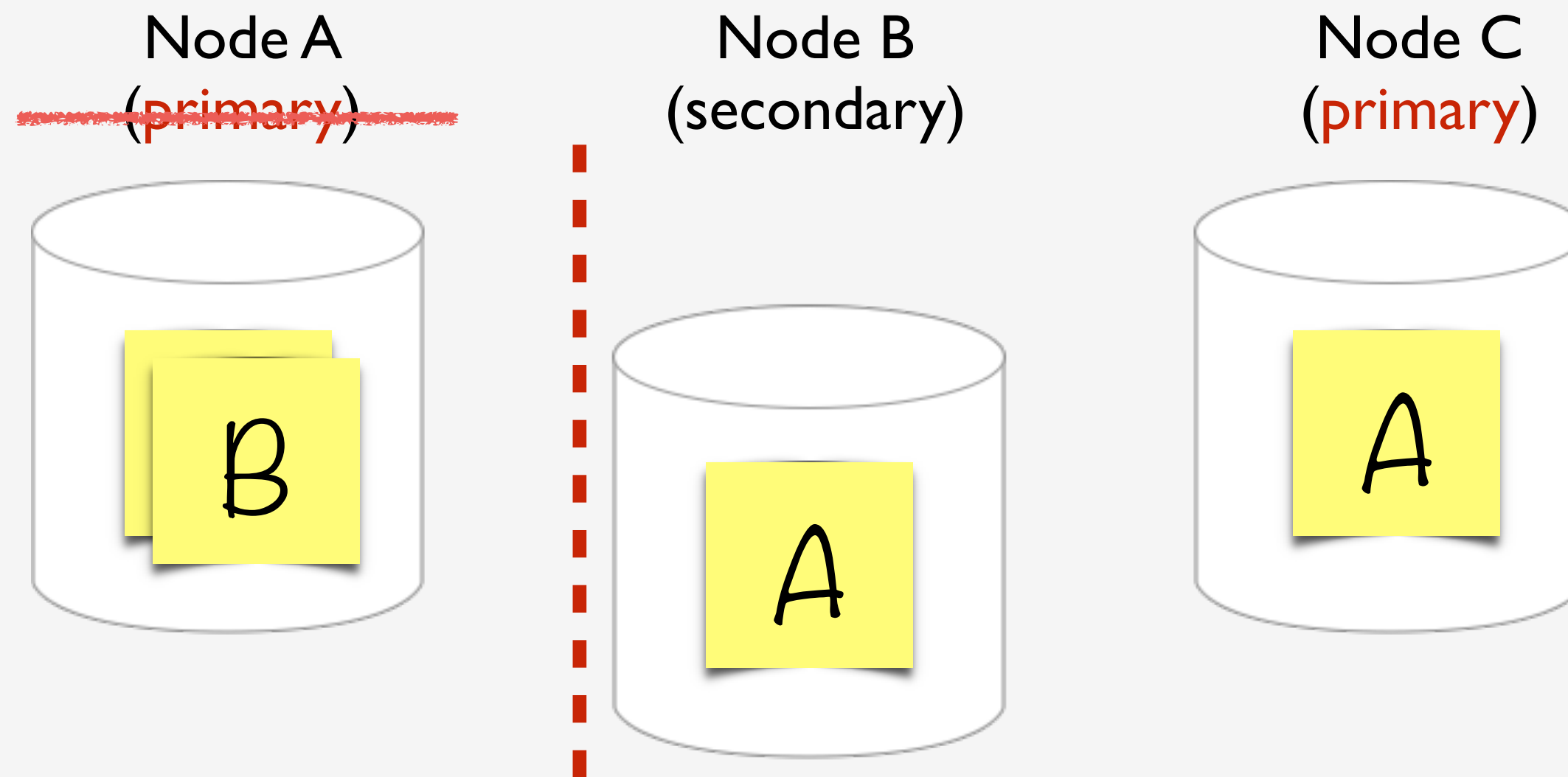








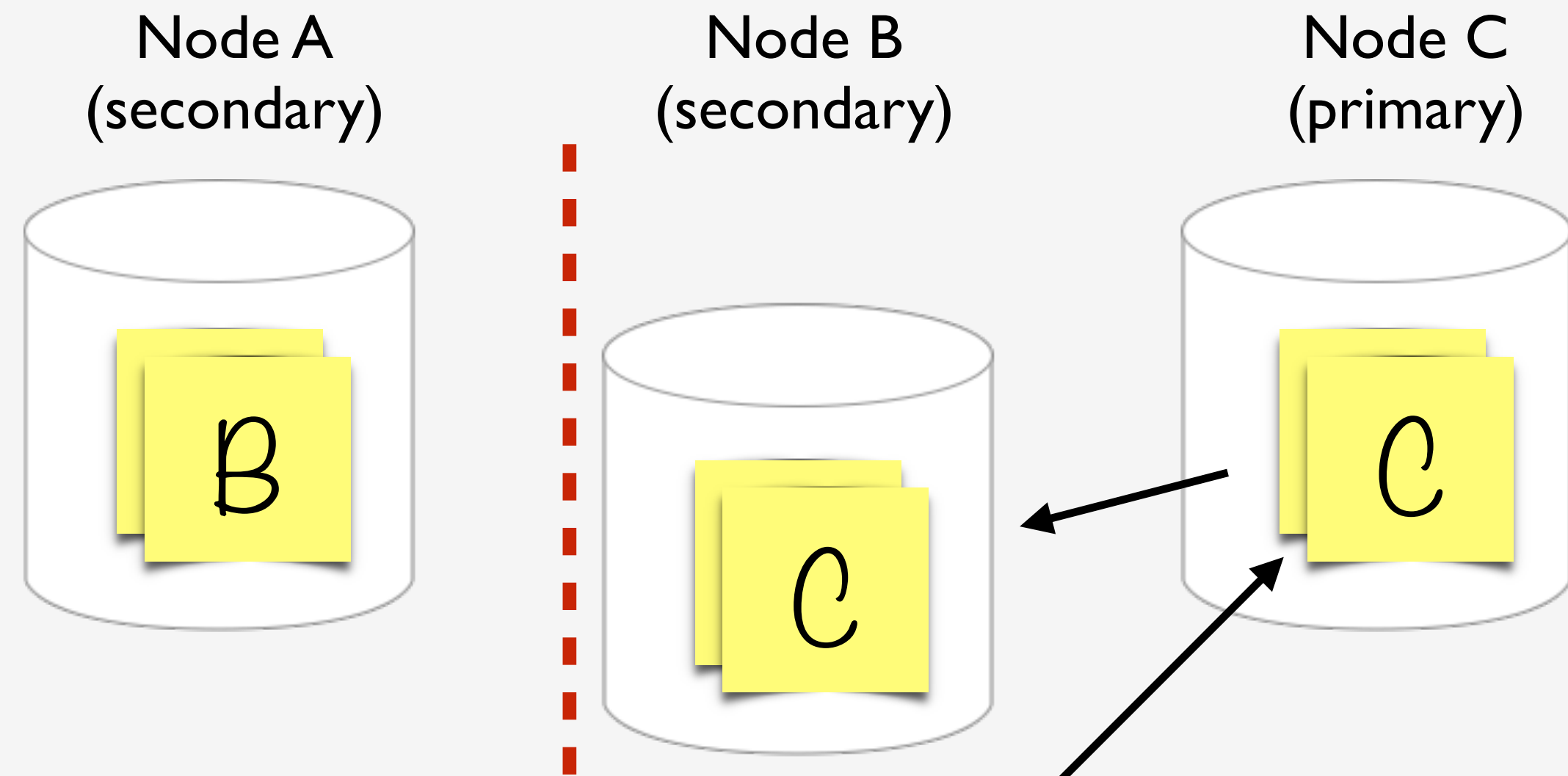
How will the system  
converge on recovery?



Old primary steps down  
New primary elected

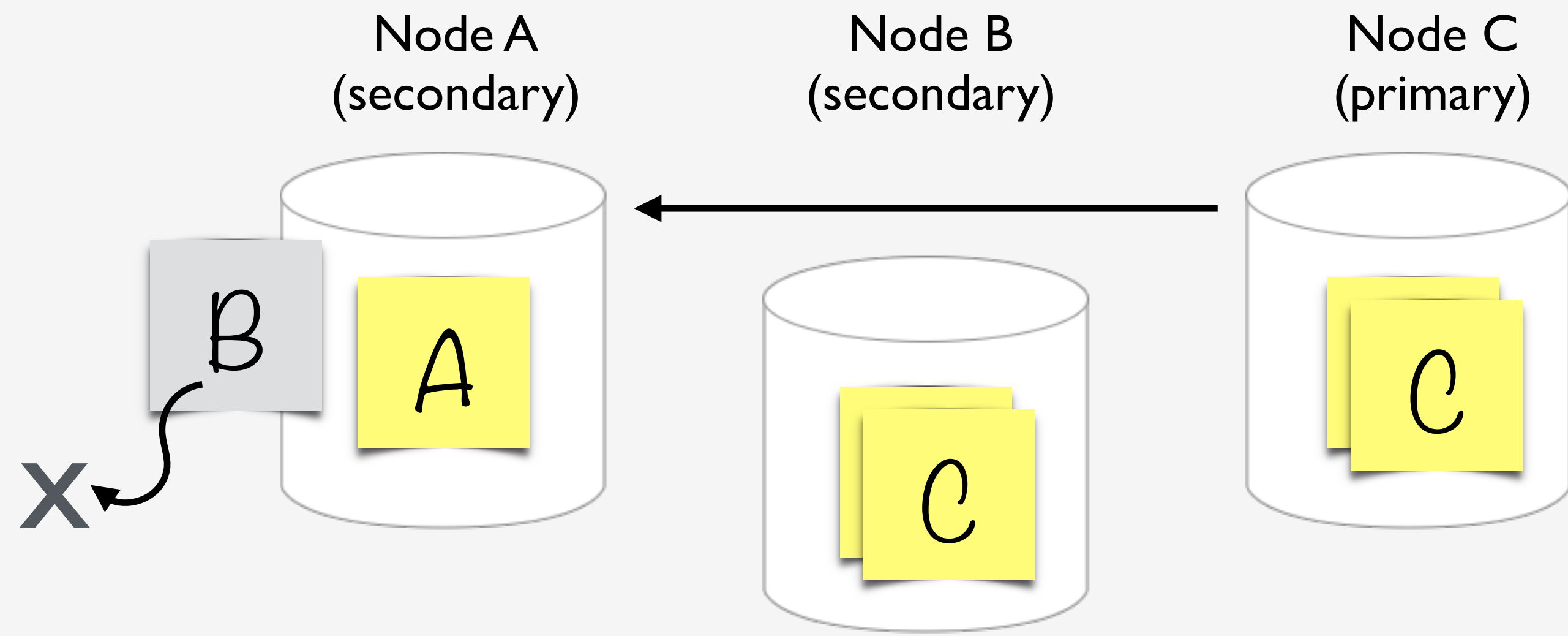






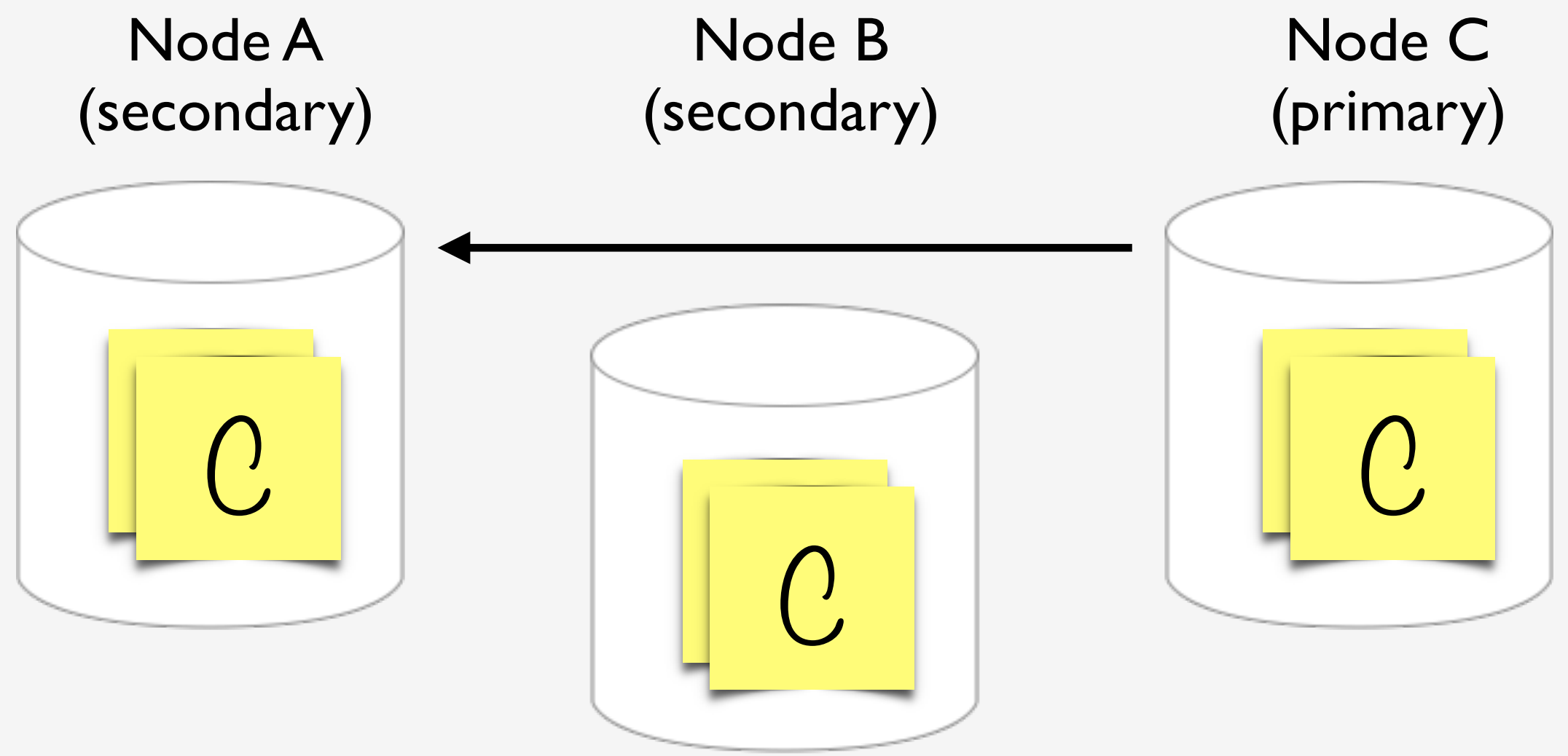
New writes occur and replicate





Partition heals  
Returning node rolls back history





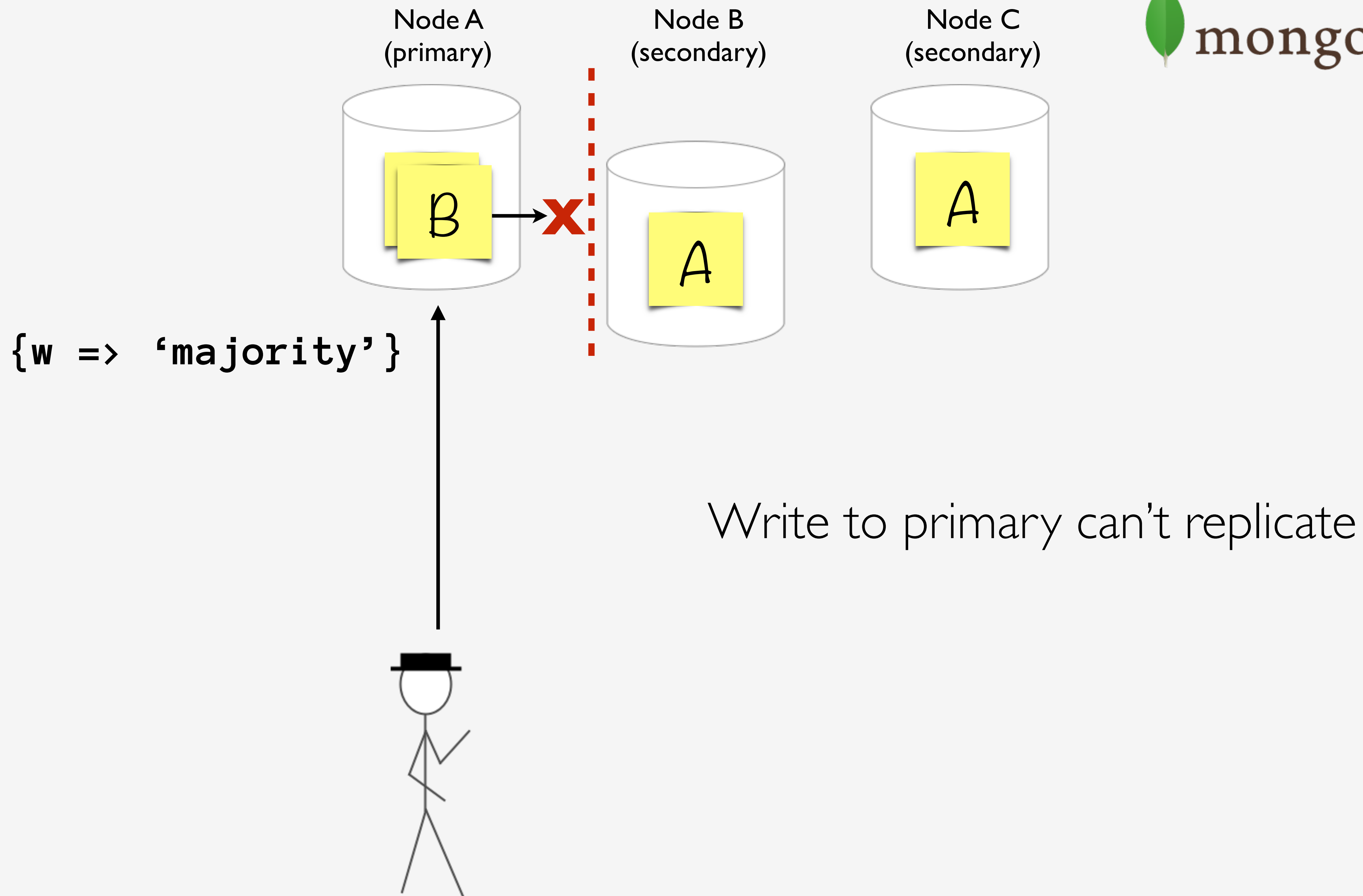
Returning node catches up with primary

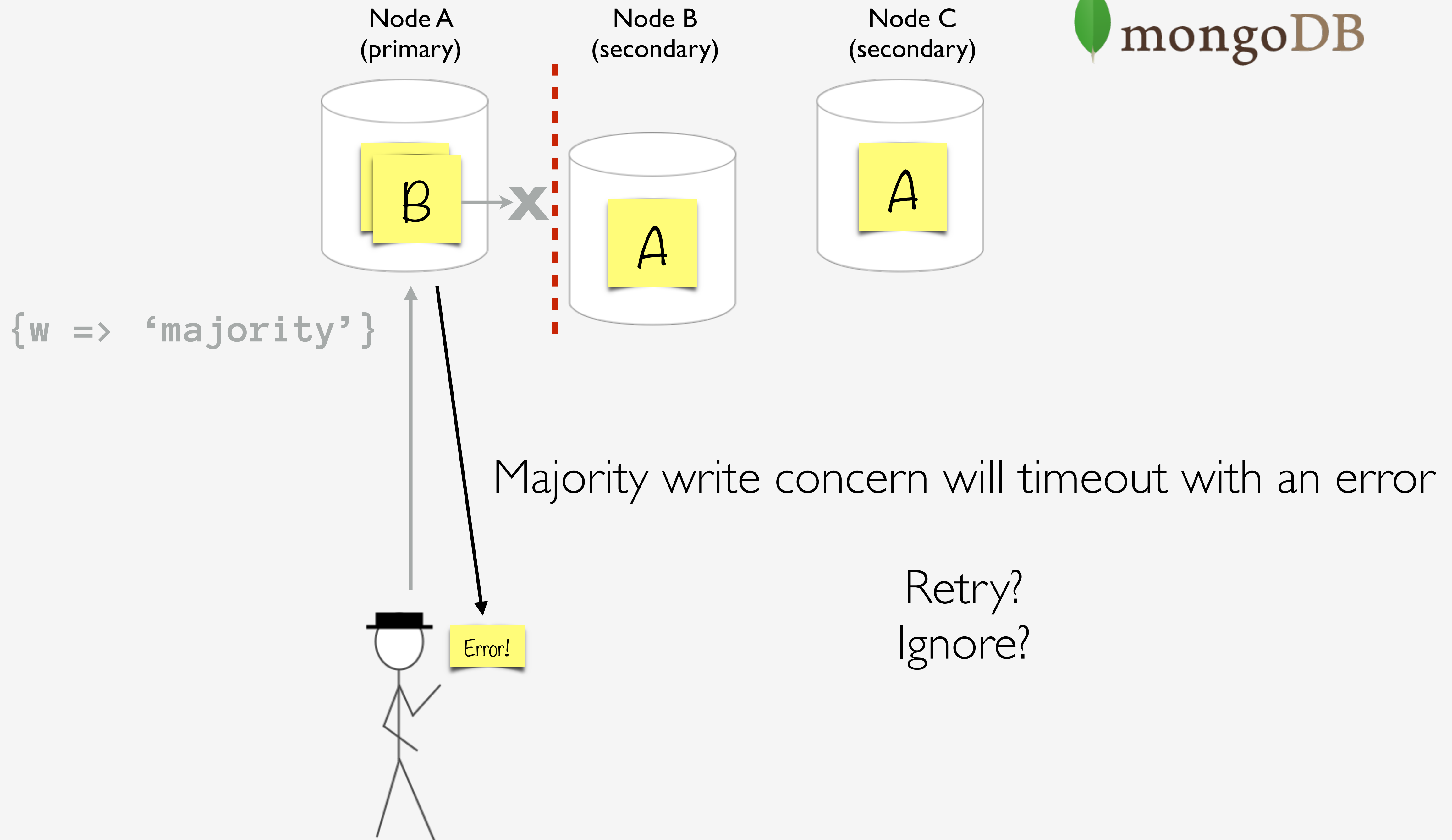


- Rollback
- Conflict records
- Conflict-free replicated data type (CRDT)  
(e.g. “add to set”)



What do we do with a  
write error?





Answers are specific to  
your application!



Thinking about reads...

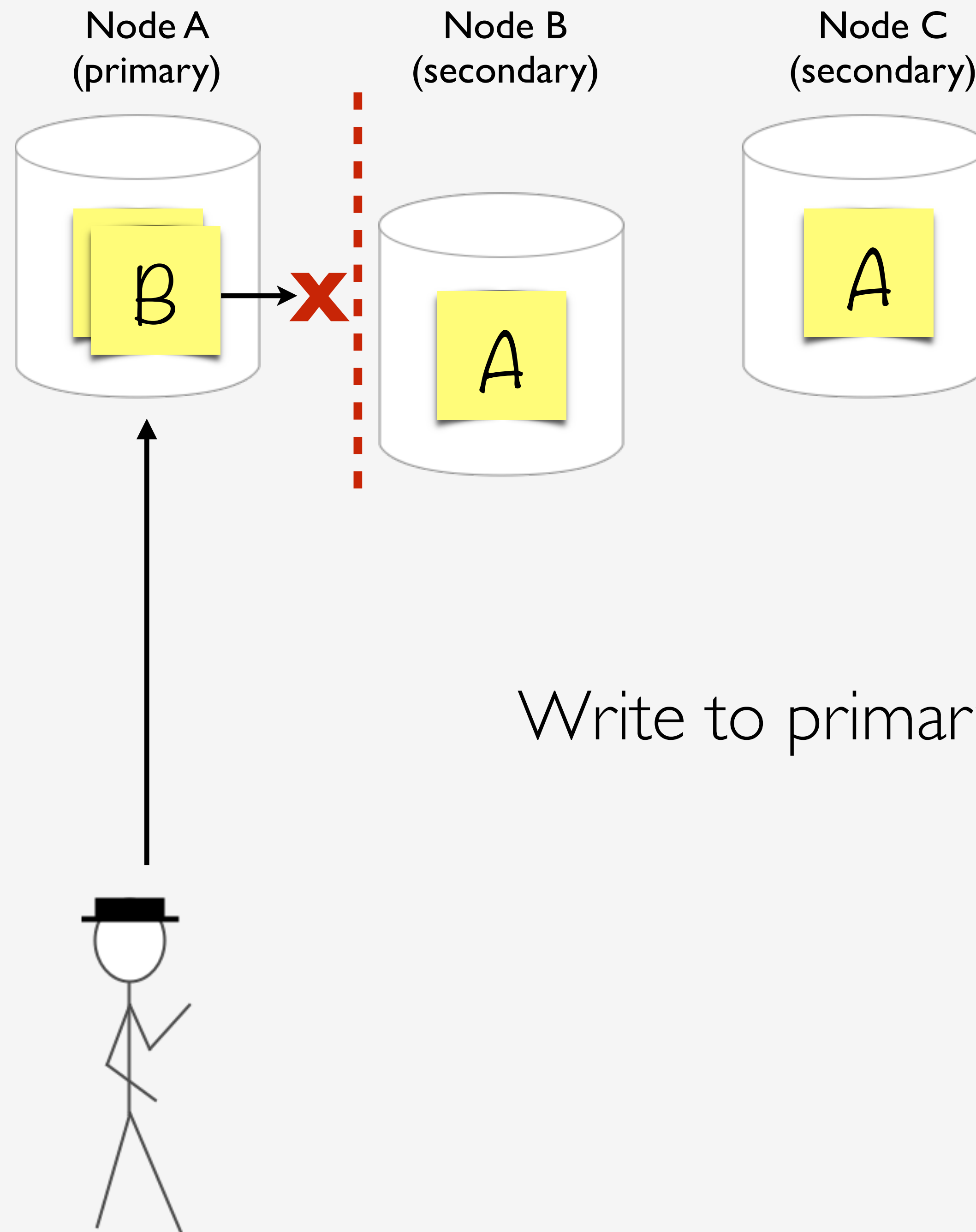
Recency  
Durability  
Latency

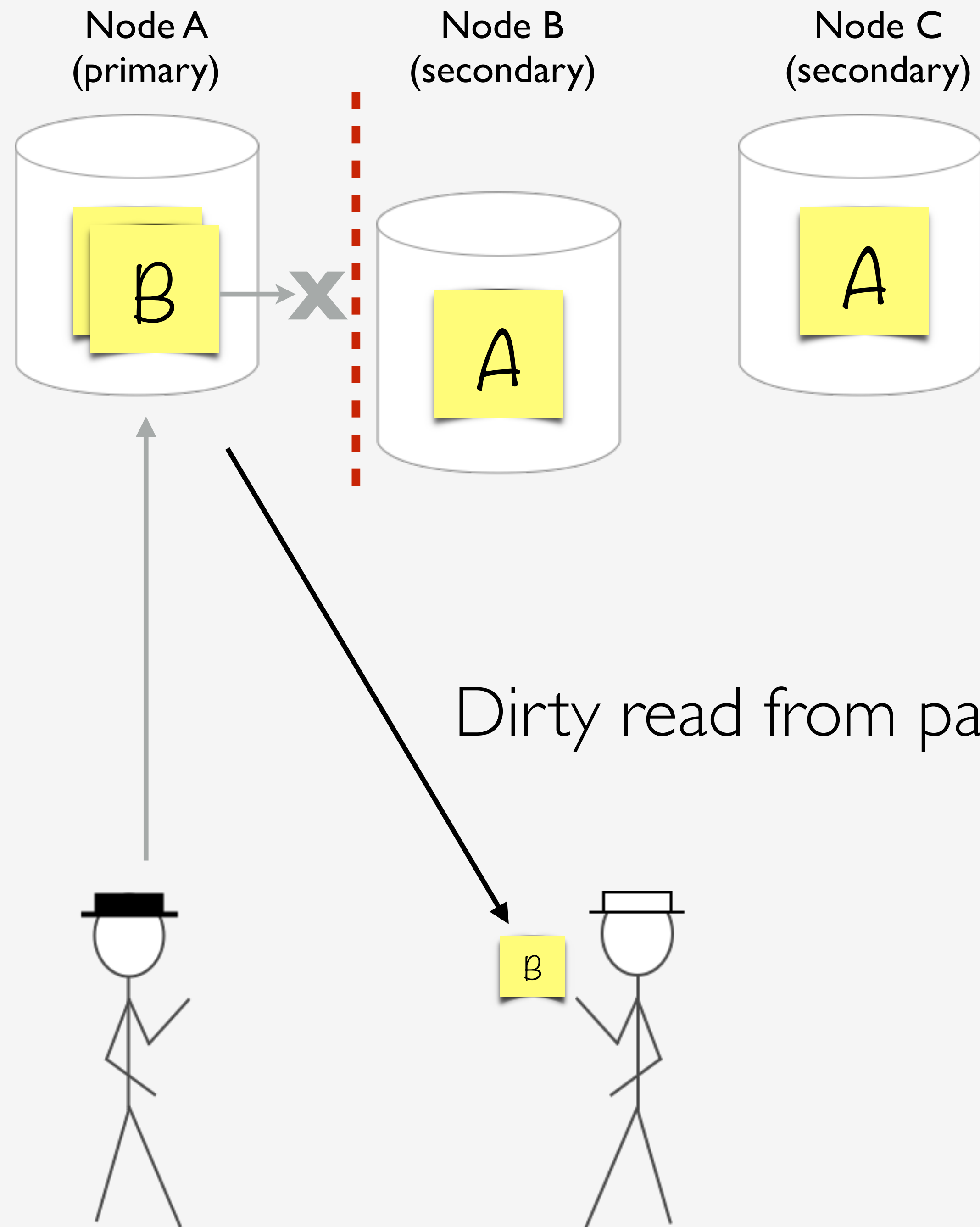
Do we care if we read  
the latest write?

Do we care if data we  
read rolls back?



Trade recency for  
durability

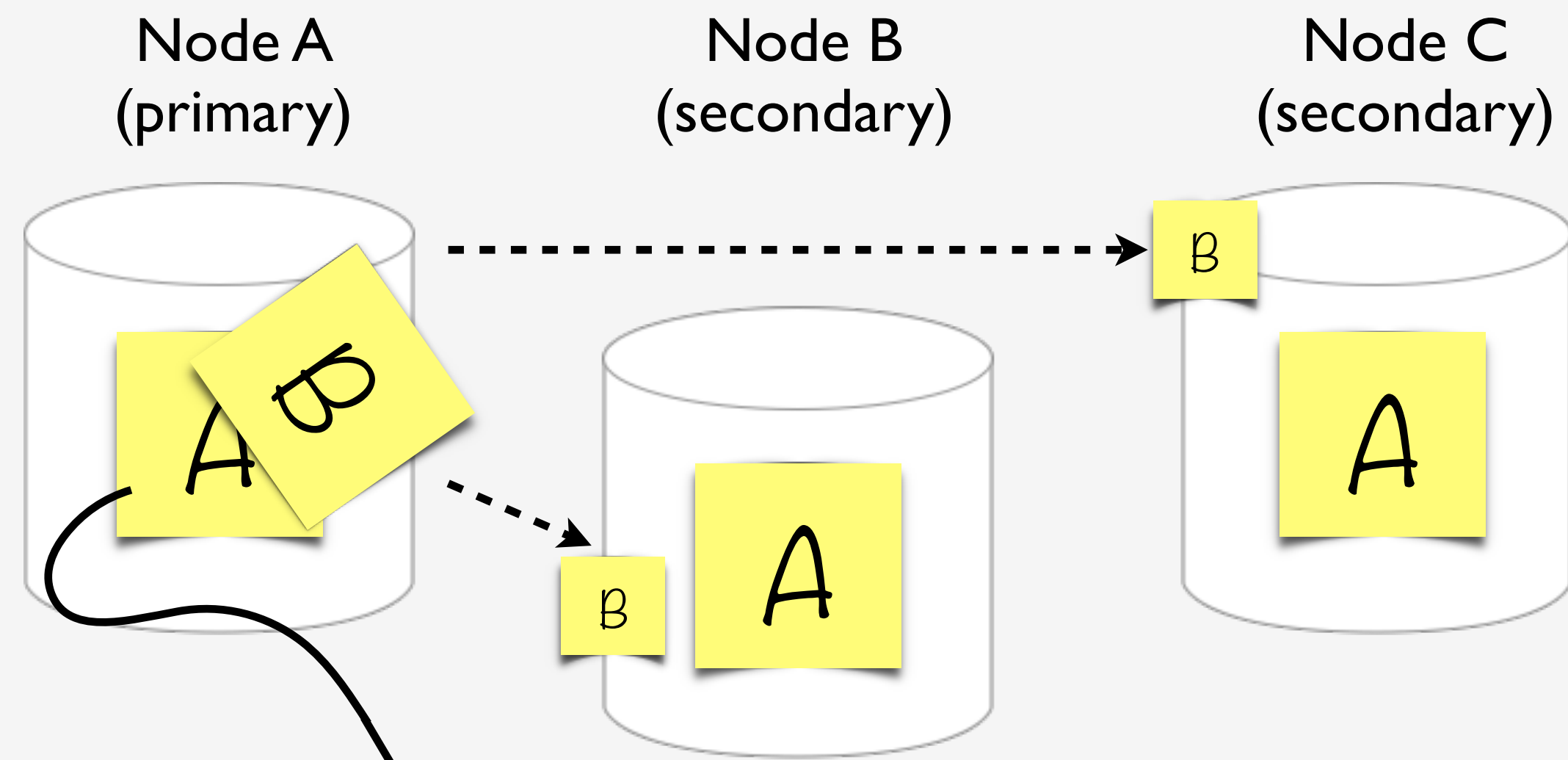




```
# read concern (3.2+)
```

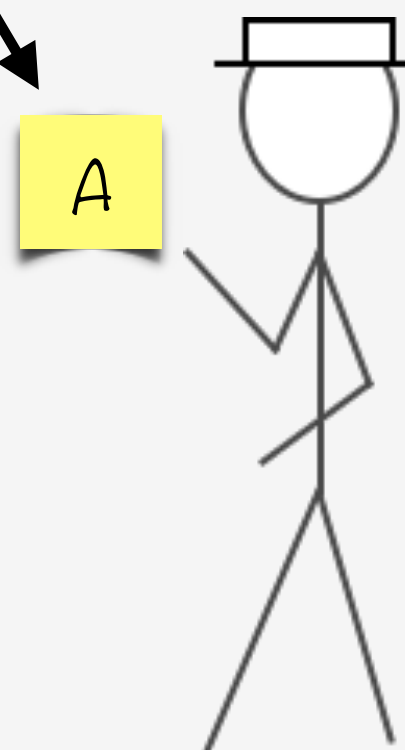
```
MongoDB->connect( $url,  
  { read_concern_level => 'local' }  
);
```

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



`{read_concern_level => 'majority'}`

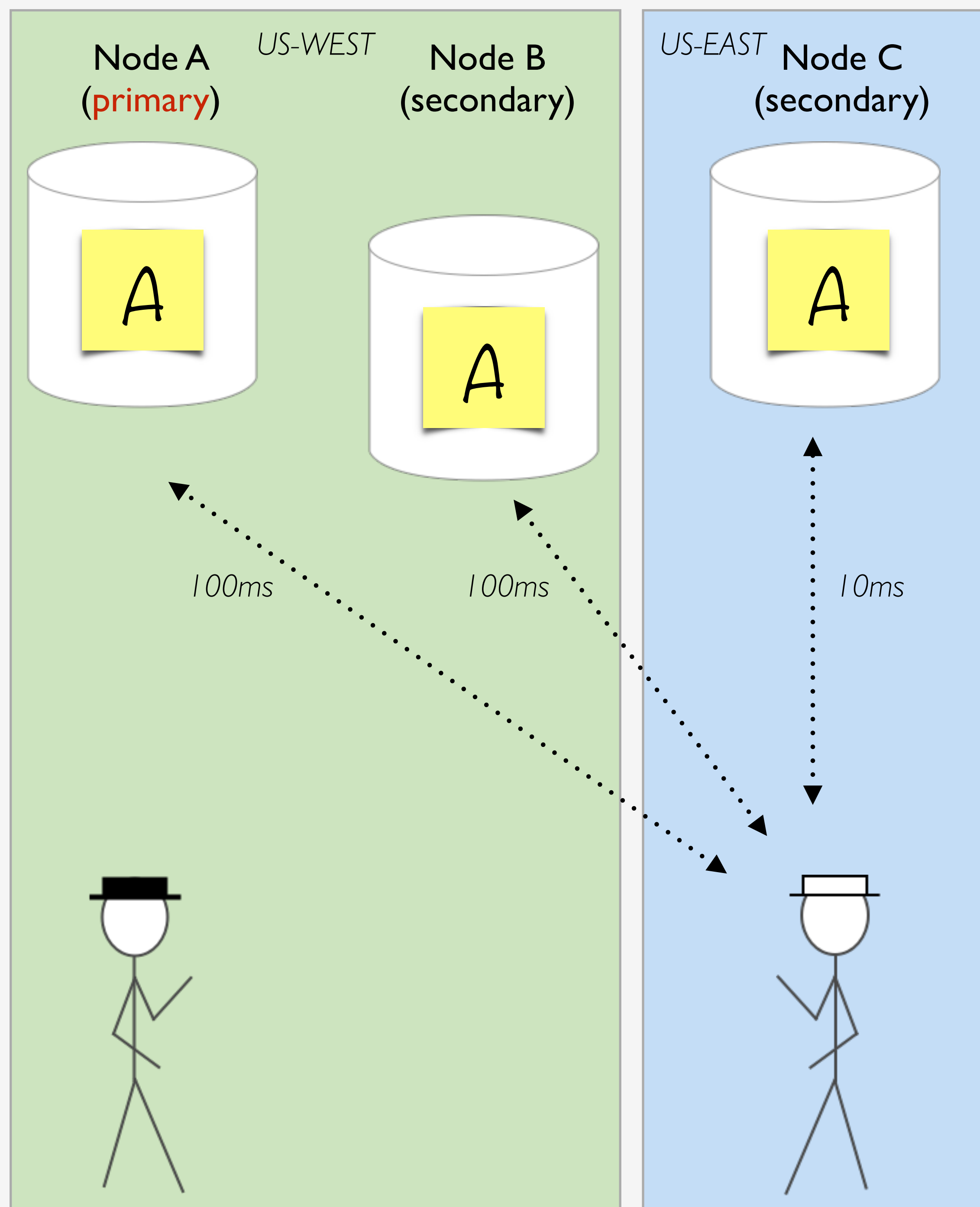
Without partition, majority read  
concern lags replication





Trade recency for  
latency

Round-trip time

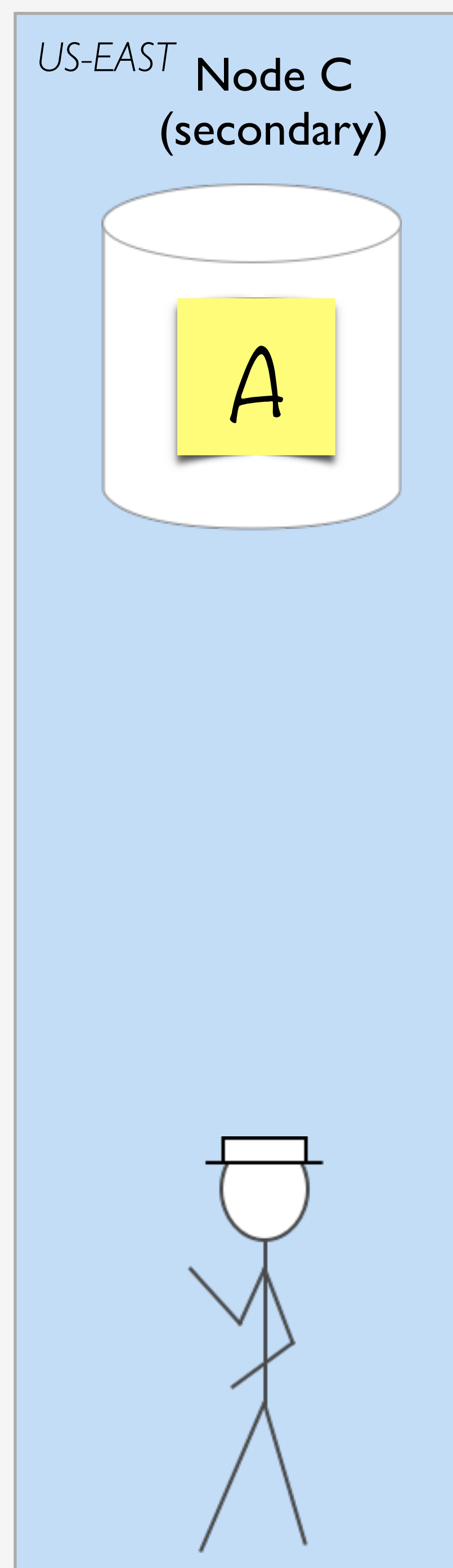
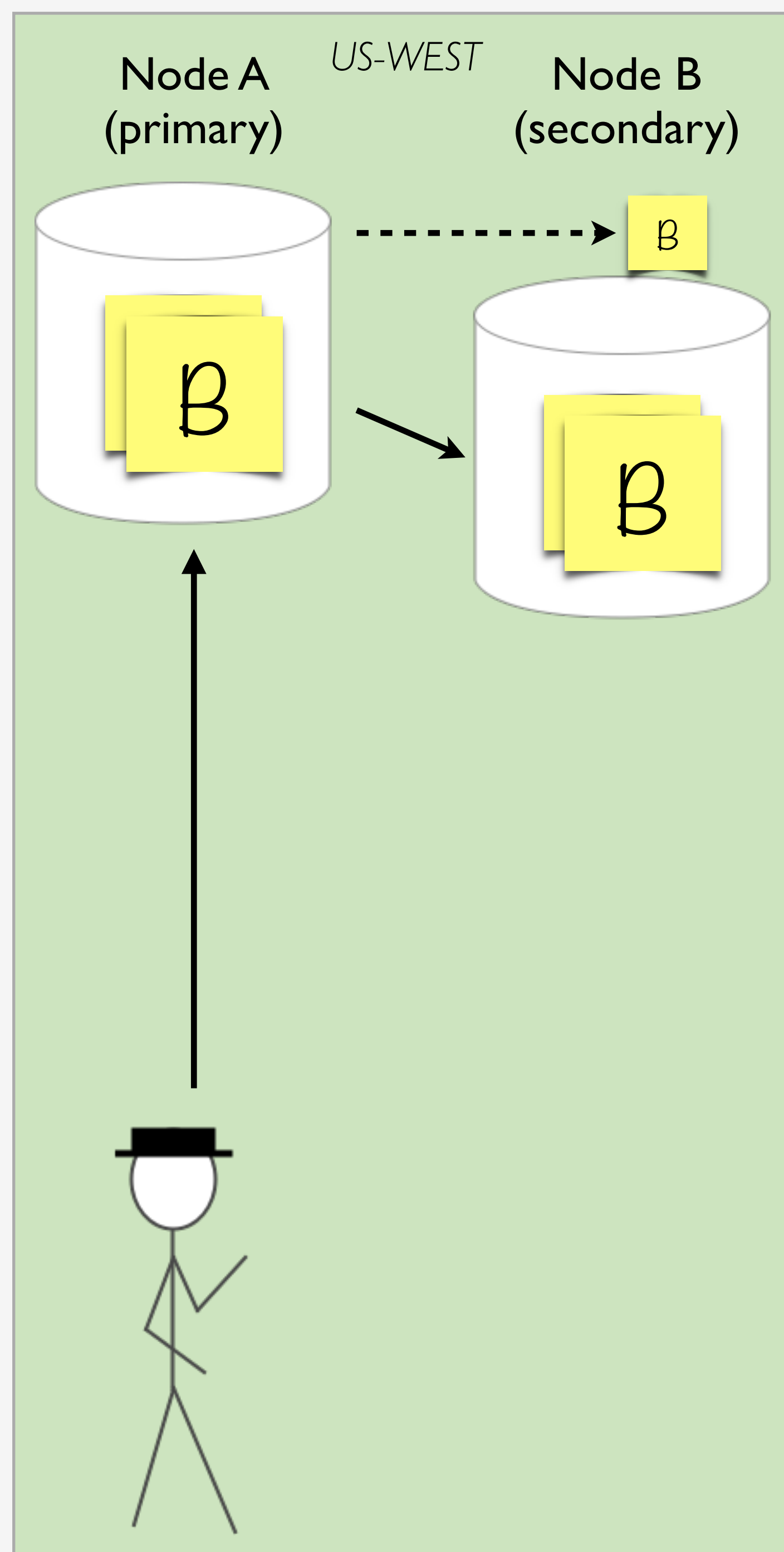


RTT for each  
data center

```
# read preference
```

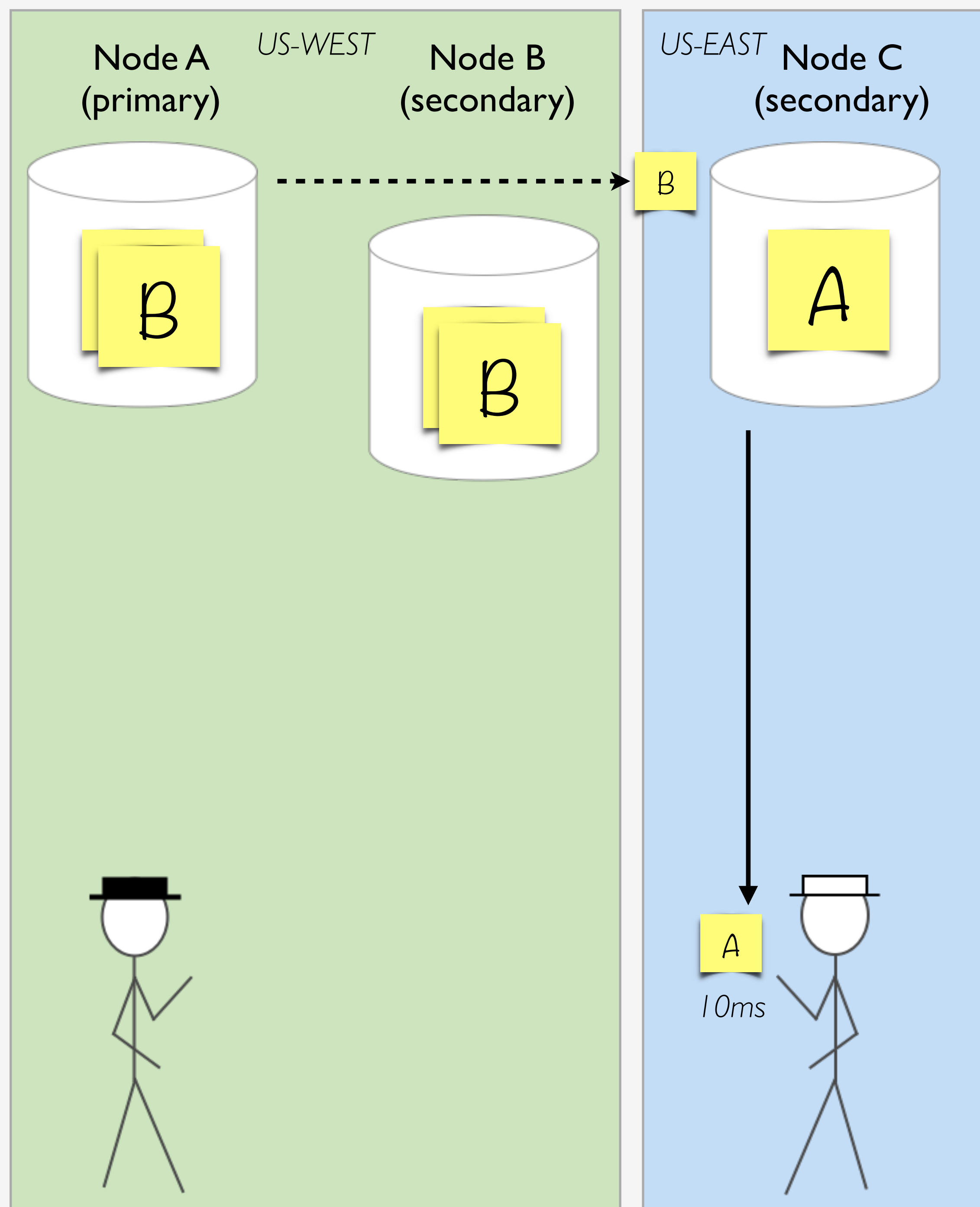
```
MongoDB->connect( $url,  
                  { read_pref_mode => 'primary' }  
);
```

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



Primary write  
starts replicating



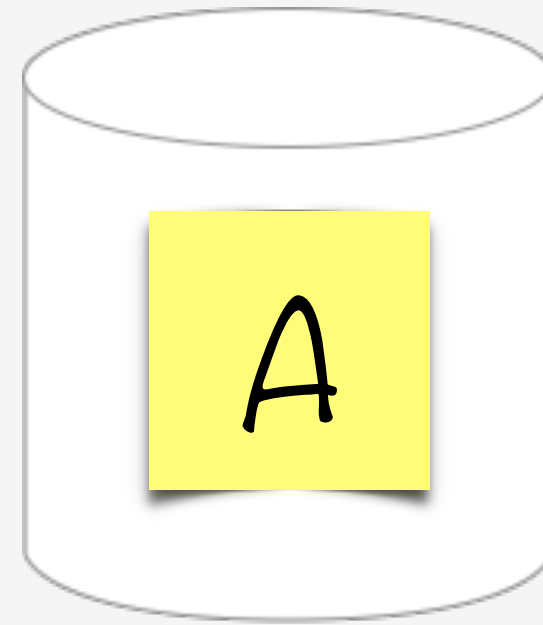


Meanwhile,  
read from  
nearest node

Still another 'gotcha'...

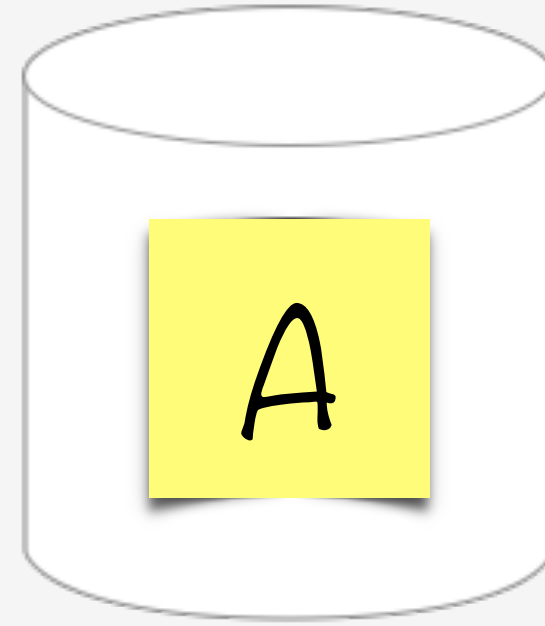
Partition detection race!

Node A  
(**'lame-duck' primary**)



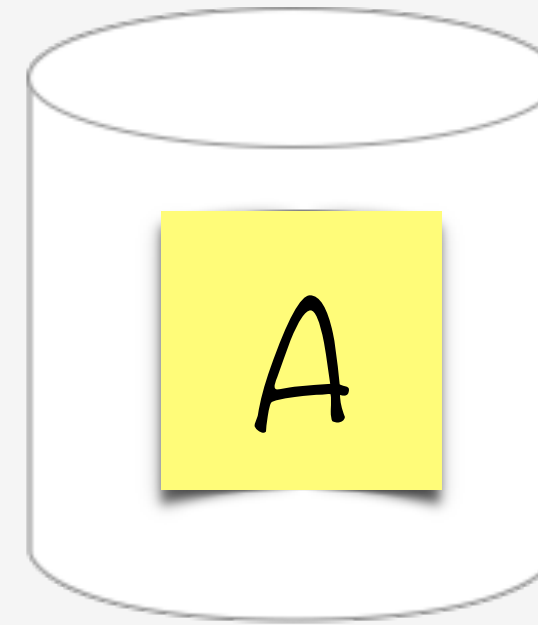
*Hasn't stepped down yet*

Node B  
(**new primary**)



*Just elected primary*

Node C  
(secondary)

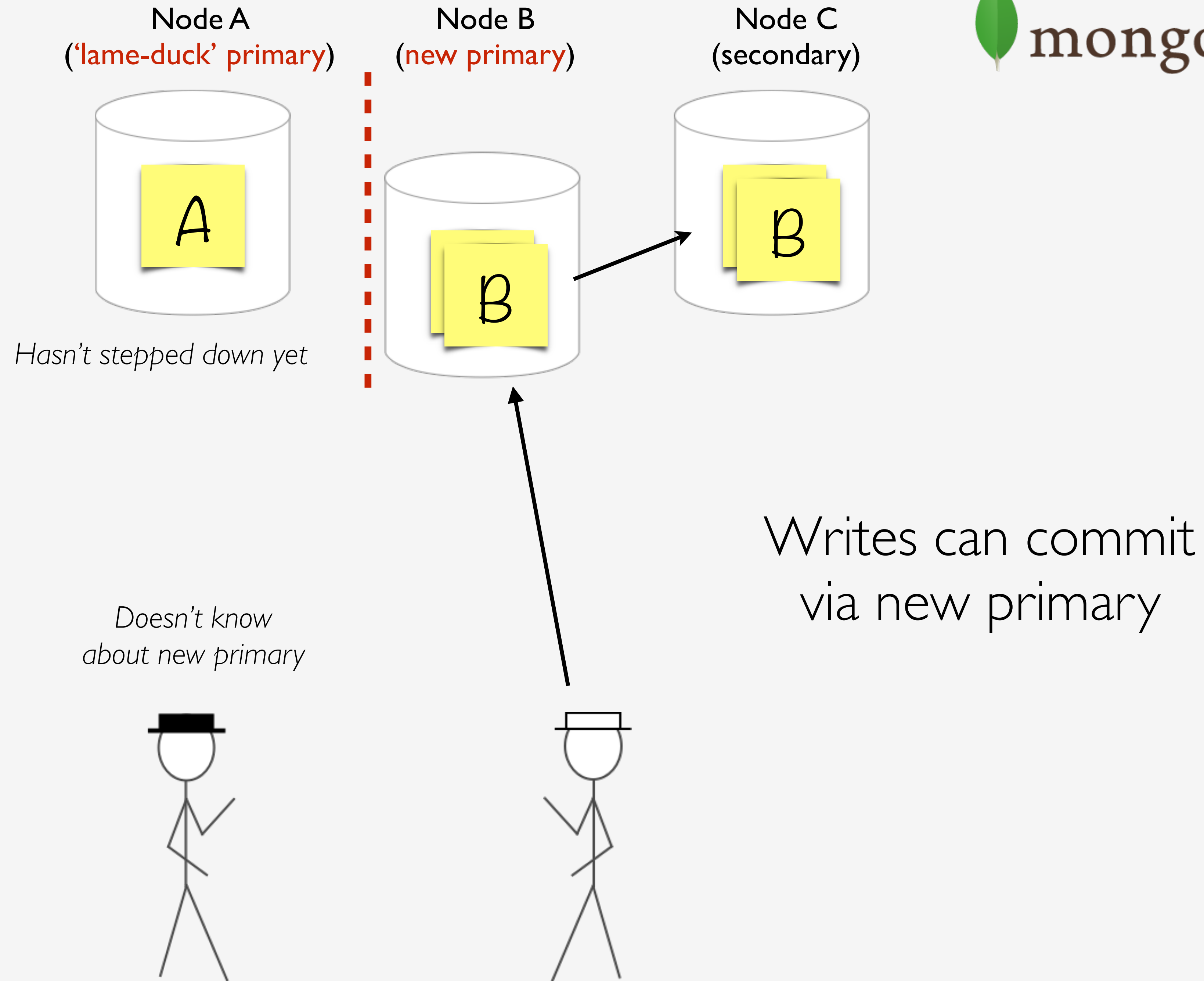


*Doesn't know  
about new primary*



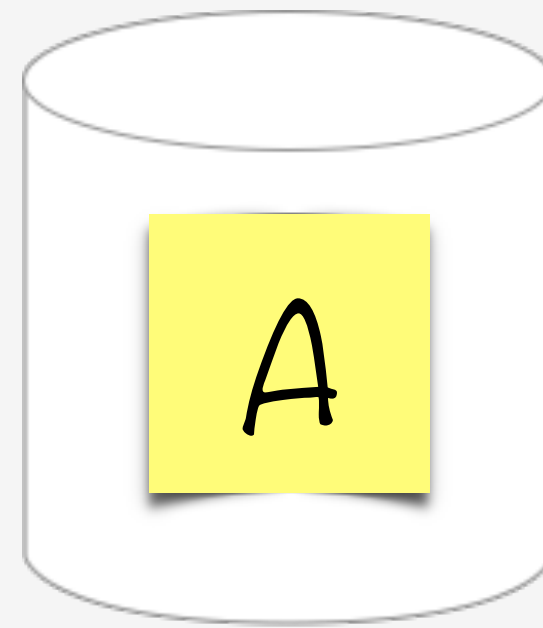
***Has discovered  
new primary***



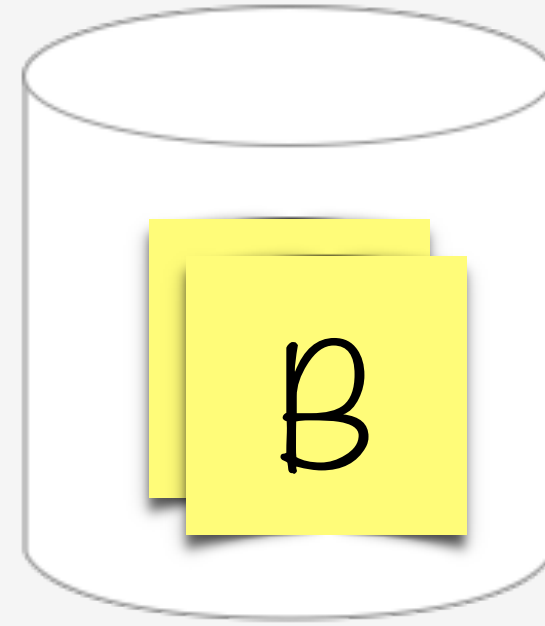




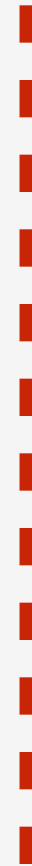
Node A  
(**'lame-duck' primary**)



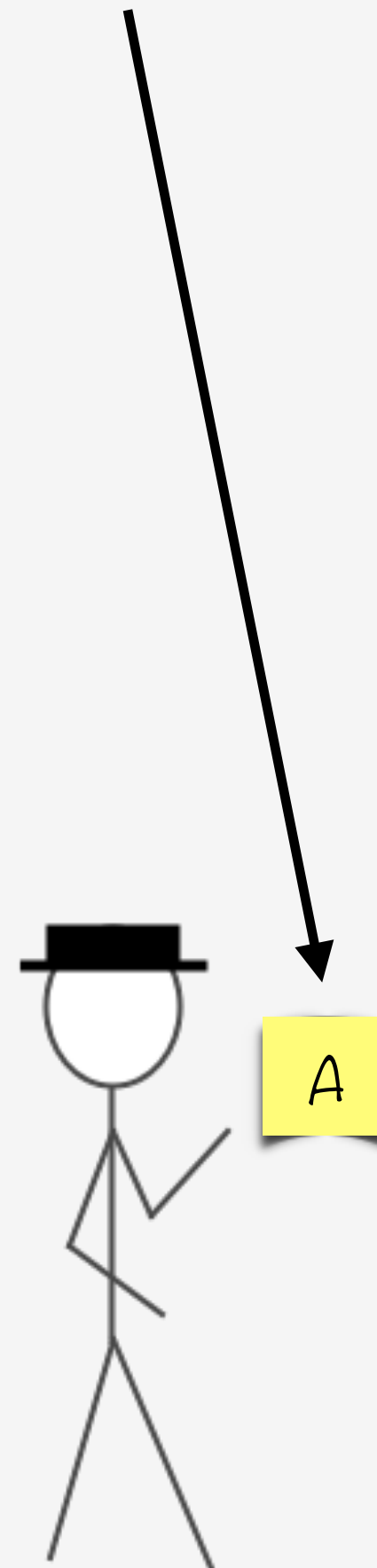
Node B  
(**new primary**)



Node C  
(secondary)

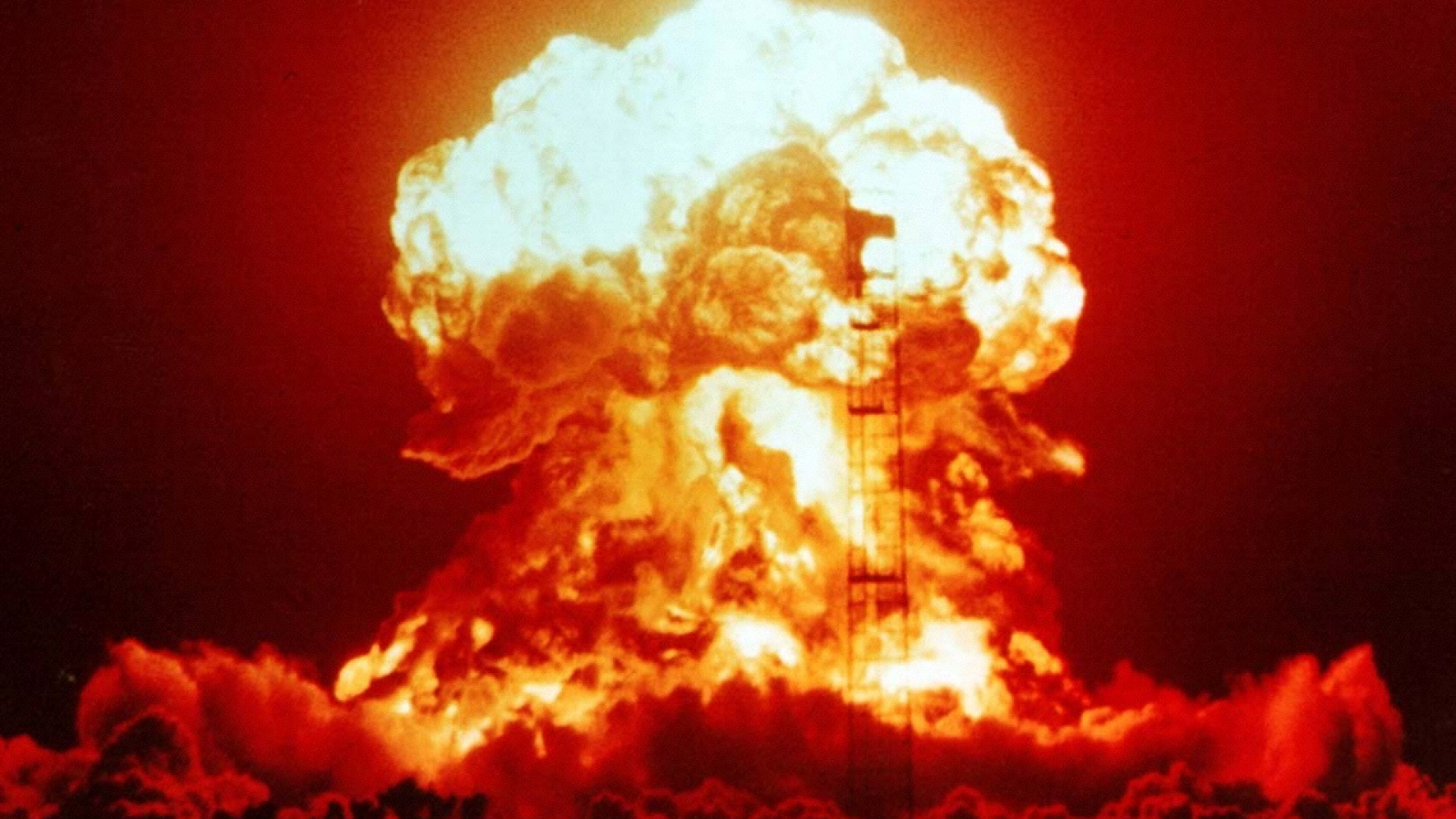


'Lame-duck' primary  
returns old  
**committed** data



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](mailto:david@mongodb.com)  
Twitter/IRC: [@xdg](#)



Photo credits:

- 9:00 AM: Mitch Martinez <https://www.youtube.com/watch?v=ScDYZXlcihc>
- 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=11469551>
- 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](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>