CassandraSF 2011

Distributed Counters in Cassandra

Sylvain Lebresne (@pcmanus)
sylvain@datastax.com

11th July, 2011

DataStax

Once upon a time. . .

- Started at the end of 2009 (roughly).
- Committed in December 2010.
- Released in 0.8.0 (June 2011).
- Initiated by peoples from Digg, then Twitter (Rainbird), . . .

Why?

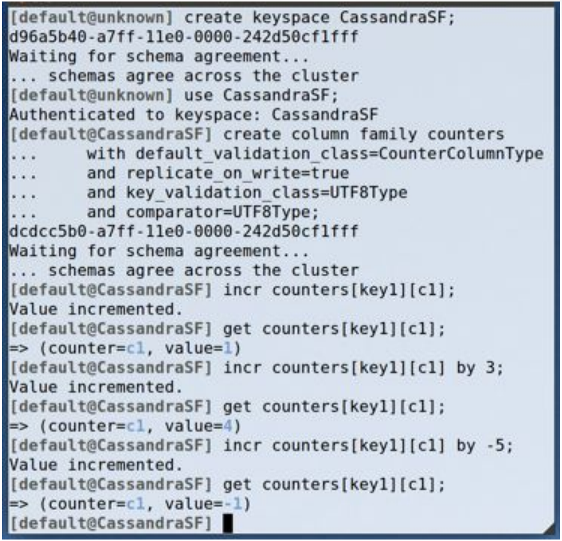
Prior to counters, solutions for counting looked like this:

- one column per increment, with a batch background job
- external synchronization (Zookeeper, through Cages)
- use another database (Redis, PostgreSQL, . . .)

Those all suffers from one or more of:

- unfriendly to use
- poor performance
- not scalable (in particular across DC)
- require additional software

What?



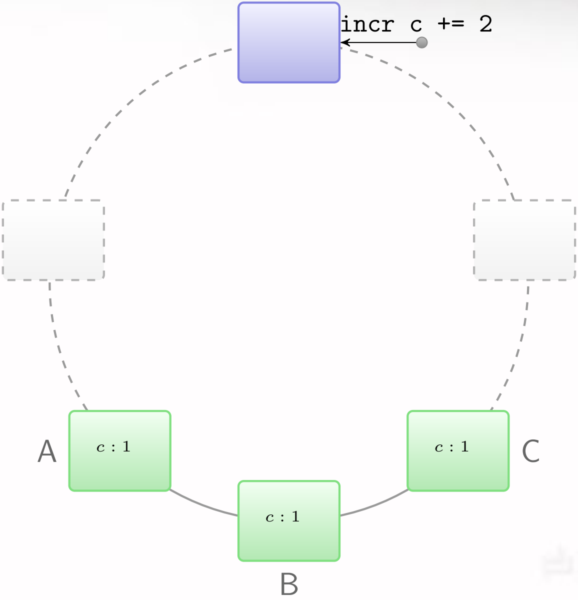
```
[default@unknown] create keyspace CassandraSF;
d96a5b40-a7ff-11e0-0000-242d50cf1fff
Waiting for schema agreement...
... schemas agree across the cluster
[default@unknown] use CassandraSF;
Authenticated to keyspace: CassandraSF
[default@CassandraSF] create column family counters
...     with default_validation_class=CounterColumnType
...     and replicate_on_write=true
...     and key_validation_class=UTF8Type
...     and comparator=UTF8Type;
dcdcc5b0-a7ff-11e0-0000-242d50cf1fff
Waiting for schema agreement...
... schemas agree across the cluster
[default@CassandraSF] incr counters[key1][c1];
Value incremented.
[default@CassandraSF] get counters[key1][c1];
=> (counter=c1, value=1)
[default@CassandraSF] incr counters[key1][c1] by 3;
Value incremented.
[default@CassandraSF] get counters[key1][c1];
=> (counter=c1, value=4)
[default@CassandraSF] incr counters[key1][c1] by -5;
Value incremented.
[default@CassandraSF] get counters[key1][c1];
=> (counter=c1, value=-1)
[default@CassandraSF] 
```

- fast reads/writes
- distributed design, no SPOF
- multi-DC support

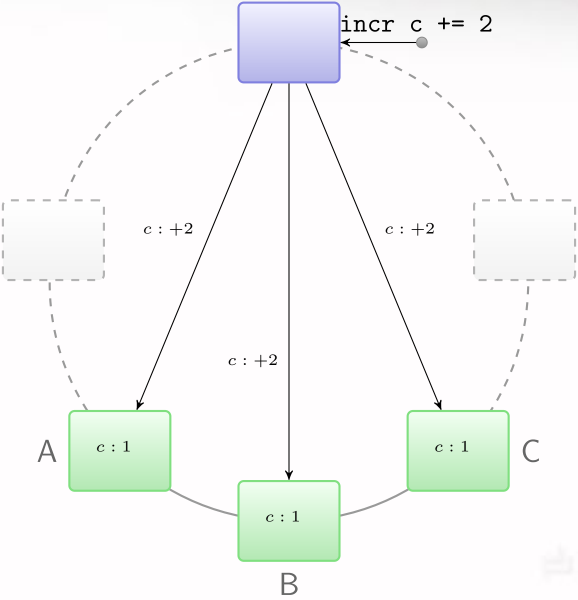
How?

Let's start by how not, and build from there.

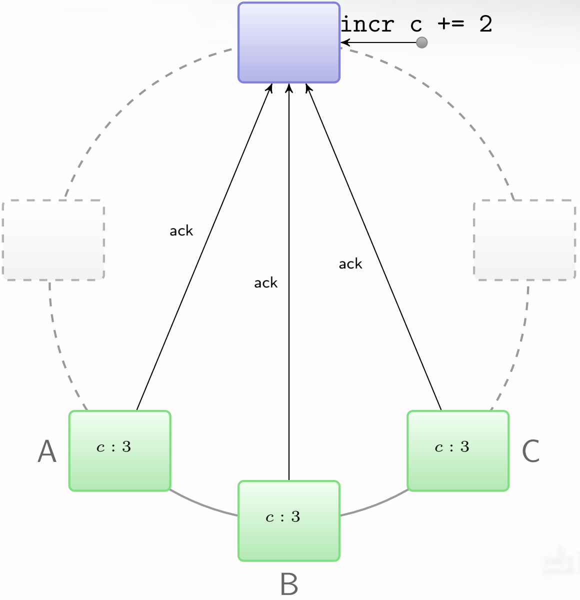
The naive approach



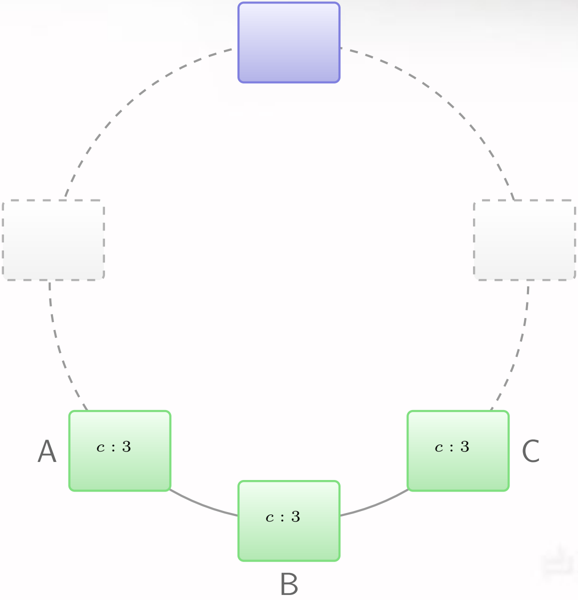
The naive approach



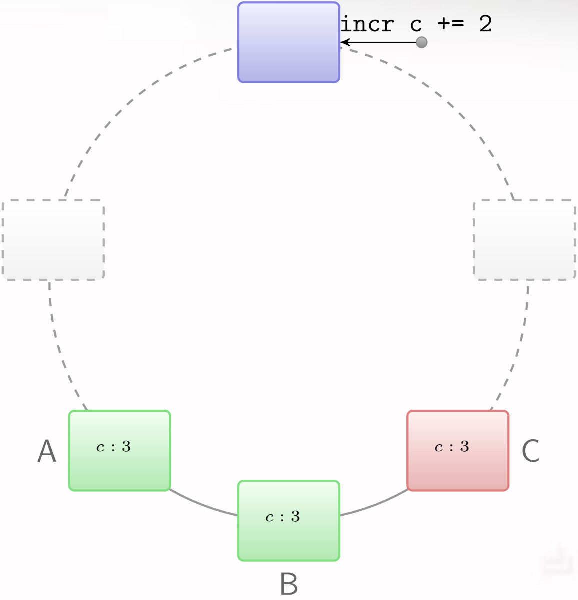
The naive approach



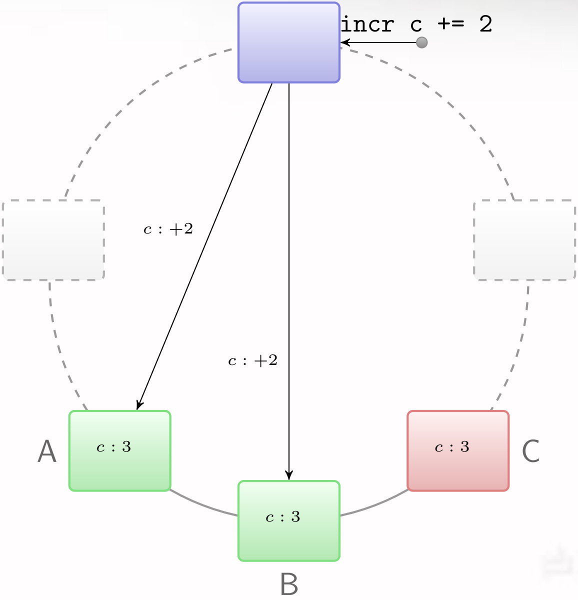
The naive approach



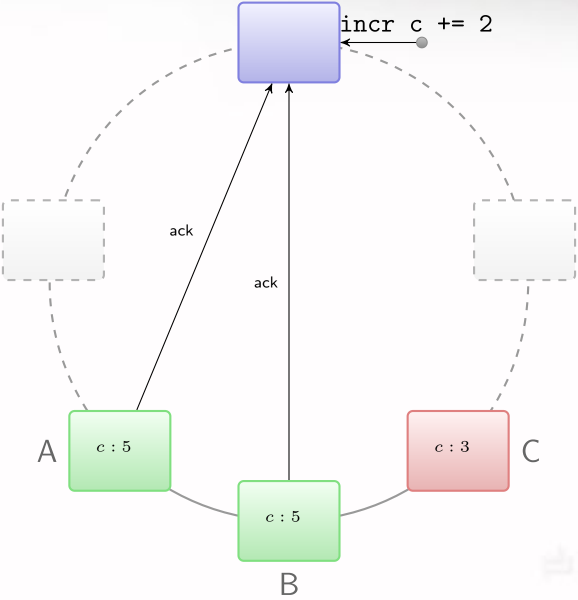
The naive approach



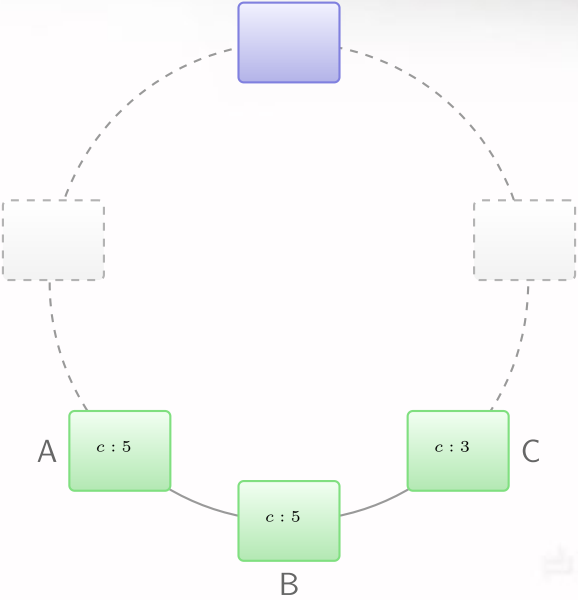
The naive approach



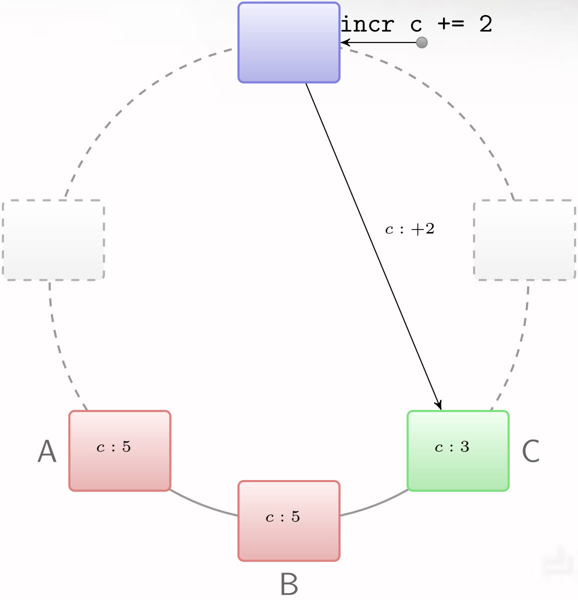
The naive approach



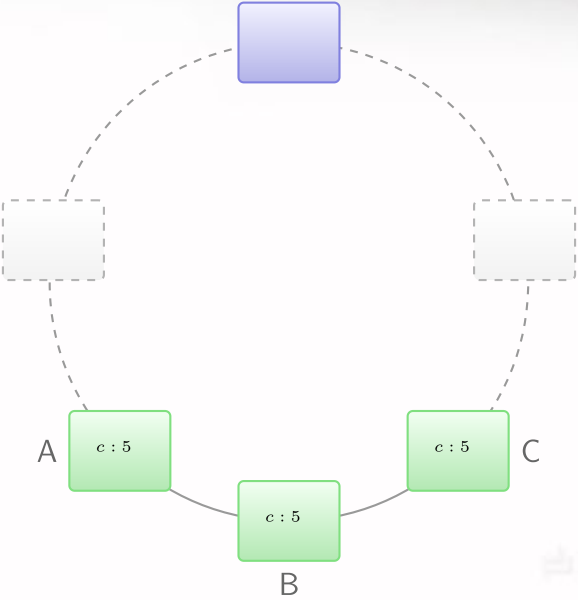
The naive approach



The naive approach



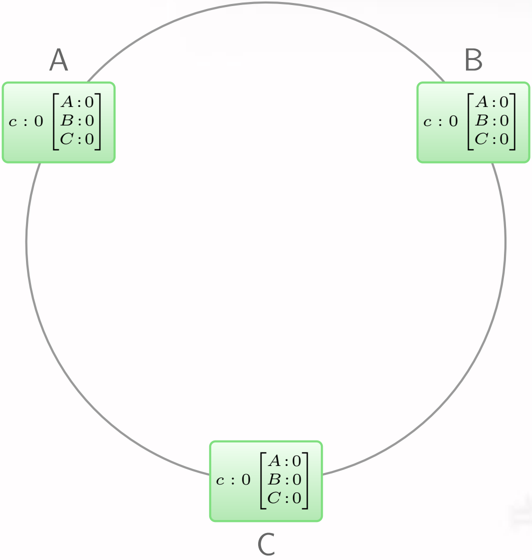
The naive approach



Vector clocks/Version vectors

> *"Vector clocks is an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations"*
>
> – *Wikipedia*

Counting with version vectors

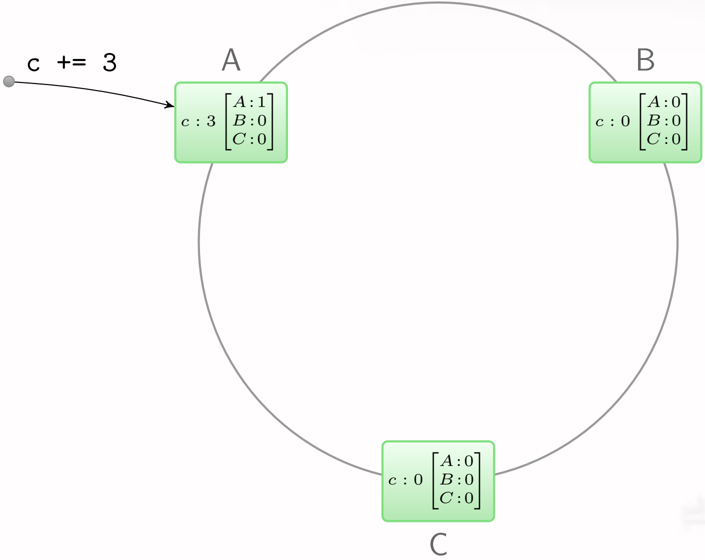


A: $c : 0 \begin{bmatrix} A:0 \\ B:0 \\ C:0 \end{bmatrix}$

B: $c : 0 \begin{bmatrix} A:0 \\ B:0 \\ C:0 \end{bmatrix}$

C: $c : 0 \begin{bmatrix} A:0 \\ B:0 \\ C:0 \end{bmatrix}$

Counting with version vectors

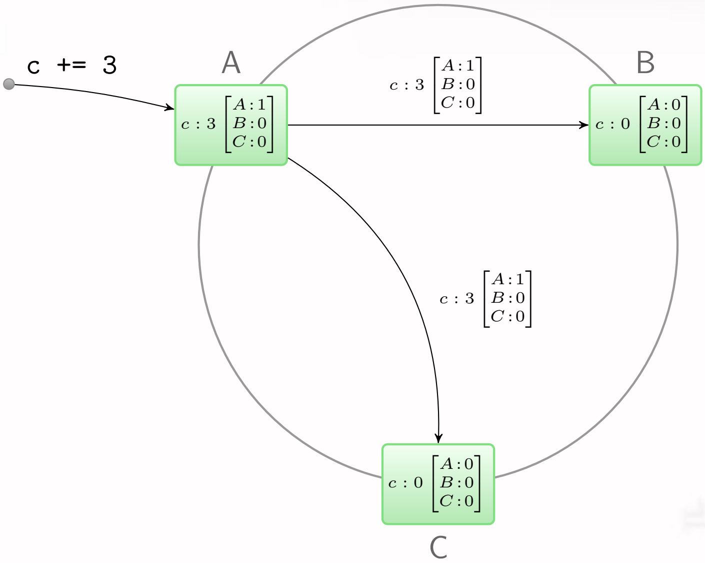


$c : 3 \begin{bmatrix} A:1 \\ B:0 \\ C:0 \end{bmatrix}$

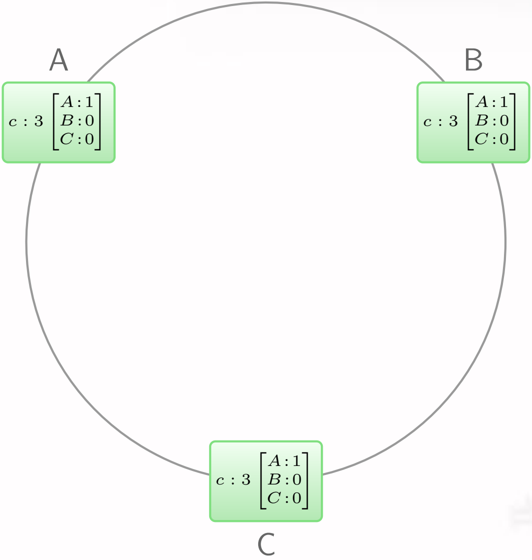
$c : 0 \begin{bmatrix} A:0 \\ B:0 \\ C:0 \end{bmatrix}$

$c : 0 \begin{bmatrix} A:0 \\ B:0 \\ C:0 \end{bmatrix}$

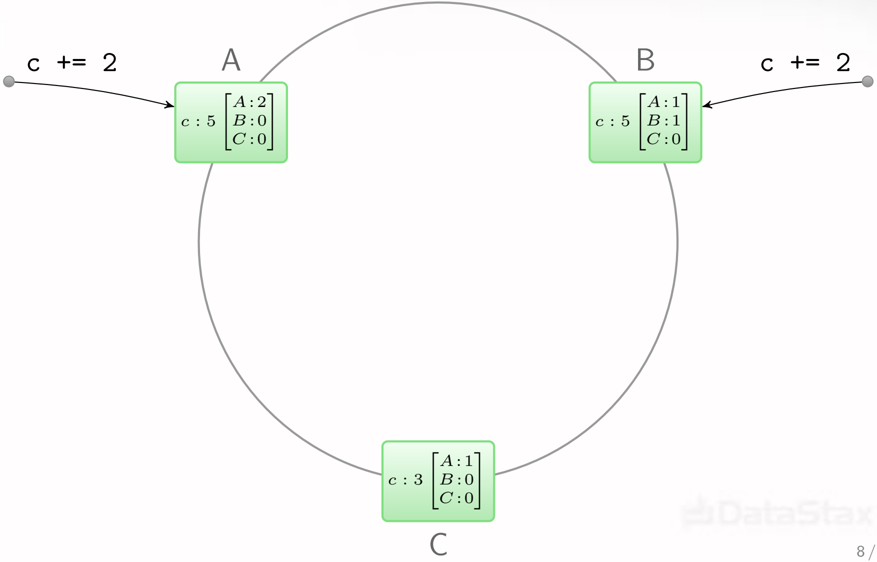
Counting with version vectors


c += 3
A
B
C
$c : 3 \begin{bmatrix} A:1 \\ B:0 \\ C:0 \end{bmatrix}$
$c : 3 \begin{bmatrix} A:1 \\ B:0 \\ C:0 \end{bmatrix}$
$c : 0 \begin{bmatrix} A:0 \\ B:0 \\ C:0 \end{bmatrix}$
$c : 3 \begin{bmatrix} A:1 \\ B:0 \\ C:0 \end{bmatrix}$
$c : 0 \begin{bmatrix} A:0 \\ B:0 \\ C:0 \end{bmatrix}$

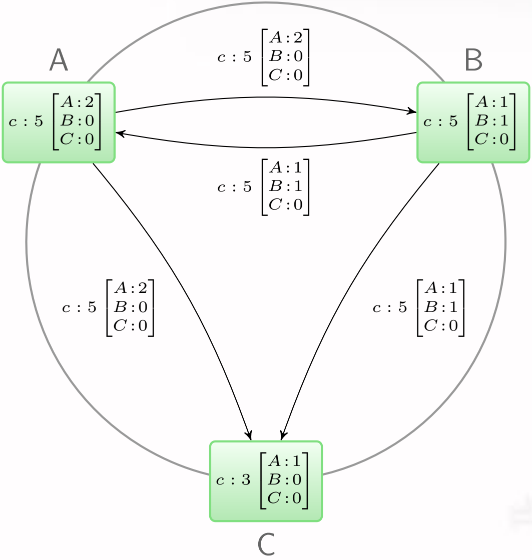
Counting with version vectors


A
$c:3\begin{bmatrix}A:1\\B:0\\C:0\end{bmatrix}$
B
$c:3\begin{bmatrix}A:1\\B:0\\C:0\end{bmatrix}$
$c:3\begin{bmatrix}A:1\\B:0\\C:0\end{bmatrix}$
C

Counting with version vectors


c += 2
A
$c : 5 \begin{bmatrix} A:2 \\ B:0 \\ C:0 \end{bmatrix}$
B
c += 2
$c : 5 \begin{bmatrix} A:1 \\ B:1 \\ C:0 \end{bmatrix}$
$c : 3 \begin{bmatrix} A:1 \\ B:0 \\ C:0 \end{bmatrix}$
C

Counting with version vectors


A
B
C
$c : 5 \begin{bmatrix} A:2 \\ B:0 \\ C:0 \end{bmatrix}$
$c : 5 \begin{bmatrix} A:2 \\ B:0 \\ C:0 \end{bmatrix}$
$c : 5 \begin{bmatrix} A:1 \\ B:1 \\ C:0 \end{bmatrix}$
$c : 5 \begin{bmatrix} A:1 \\ B:1 \\ C:0 \end{bmatrix}$
$c : 5 \begin{bmatrix} A:2 \\ B:0 \\ C:0 \end{bmatrix}$
$c : 5 \begin{bmatrix} A:1 \\ B:1 \\ C:0 \end{bmatrix}$
$c : 3 \begin{bmatrix} A:1 \\ B:0 \\ C:0 \end{bmatrix}$

- Vector clocks/version vectors are about detecting conflicting updates ...
- ... but says nothing about how to resolve those conflicts.

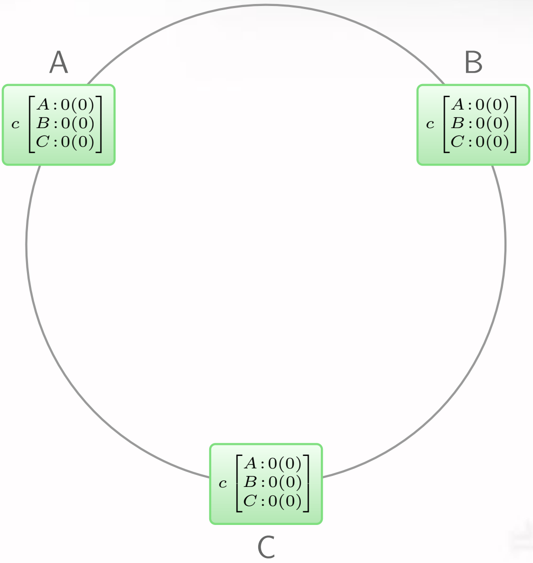
Partitioned counters

A counter is partitioned into one sub-count by replica of the counter, it is a vector of tuple (host id, sub-count, version).

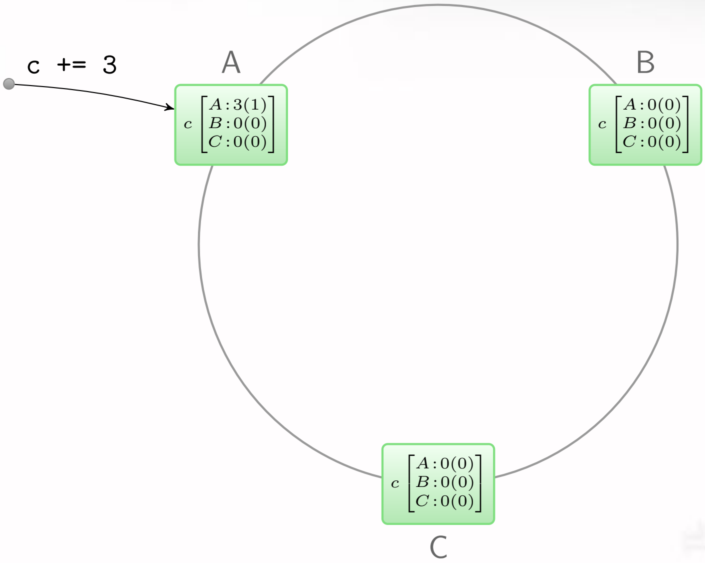
The actual value of the counter is the sum of all the sub-counts. It is only resolved on reads before answering the client.

$$c : \begin{bmatrix} A\!:\!24\ (2) \\ B\!:\!42\ (3) \\ C\!:\!17\ (1) \end{bmatrix} \iff c = 83$$

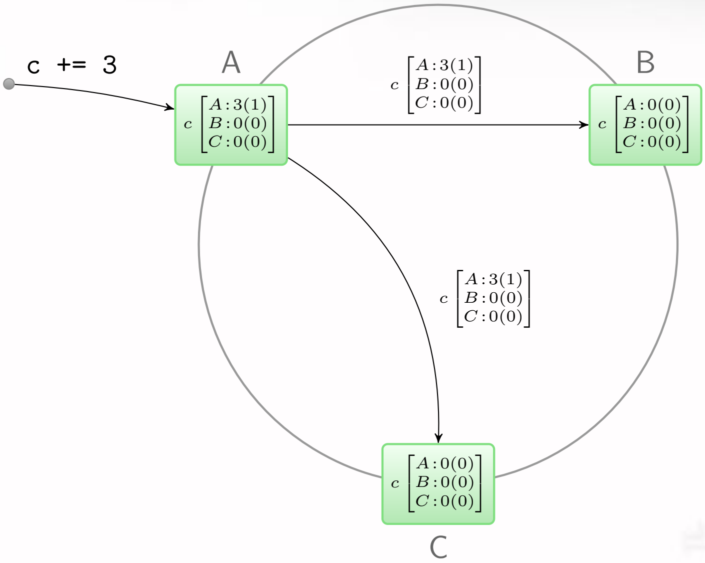
The Cassandra way


A
$c\begin{bmatrix}A:0(0)\\B:0(0)\\C:0(0)\end{bmatrix}$
B
$c\begin{bmatrix}A:0(0)\\B:0(0)\\C:0(0)\end{bmatrix}$
$c\begin{bmatrix}A:0(0)\\B:0(0)\\C:0(0)\end{bmatrix}$
C

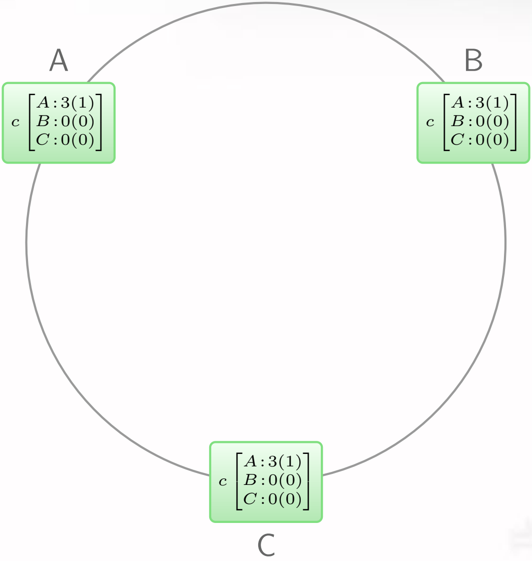
The Cassandra way


c += 3
A
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
$c \begin{bmatrix} A:0(0) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:0(0) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C

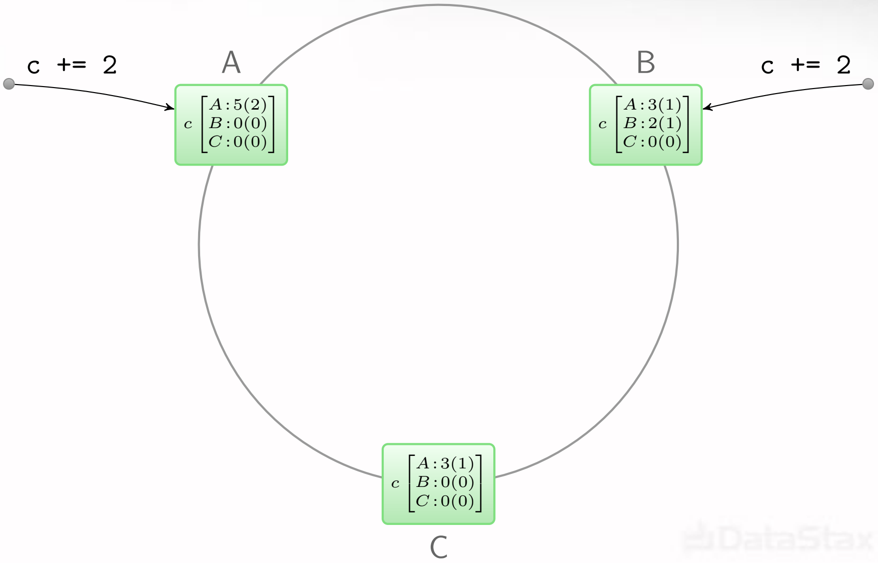
The Cassandra way


c += 3
A
B
C
$c\begin{bmatrix}A:3(1)\\B:0(0)\\C:0(0)\end{bmatrix}$
$c\begin{bmatrix}A:3(1)\\B:0(0)\\C:0(0)\end{bmatrix}$
$c\begin{bmatrix}A:0(0)\\B:0(0)\\C:0(0)\end{bmatrix}$
$c\begin{bmatrix}A:3(1)\\B:0(0)\\C:0(0)\end{bmatrix}$
$c\begin{bmatrix}A:0(0)\\B:0(0)\\C:0(0)\end{bmatrix}$

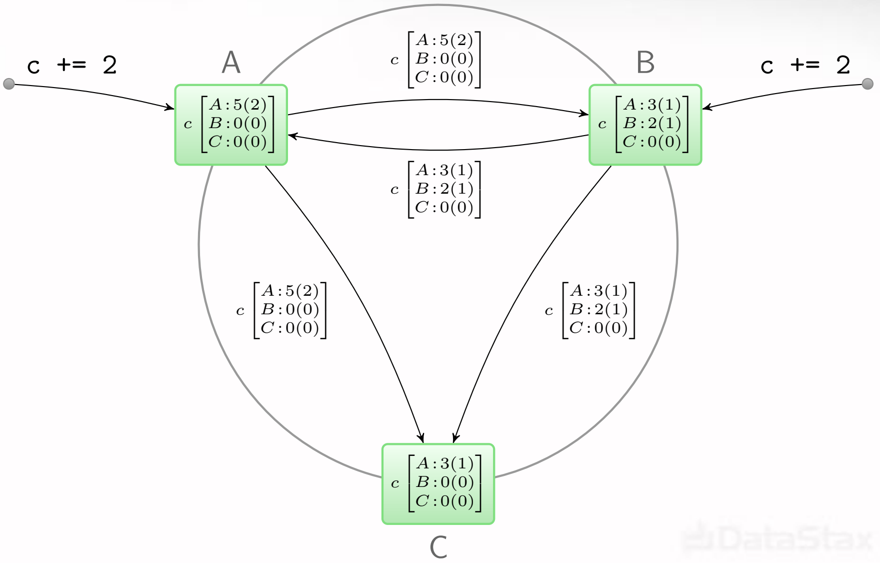
The Cassandra way


A
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C

The Cassandra way

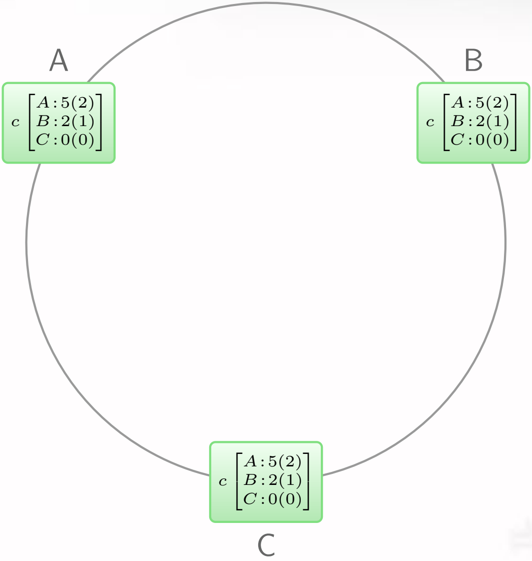

c += 2
A
$c \begin{bmatrix} A:5(2) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
c += 2
$c \begin{bmatrix} A:3(1) \\ B:2(1) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C

The Cassandra way

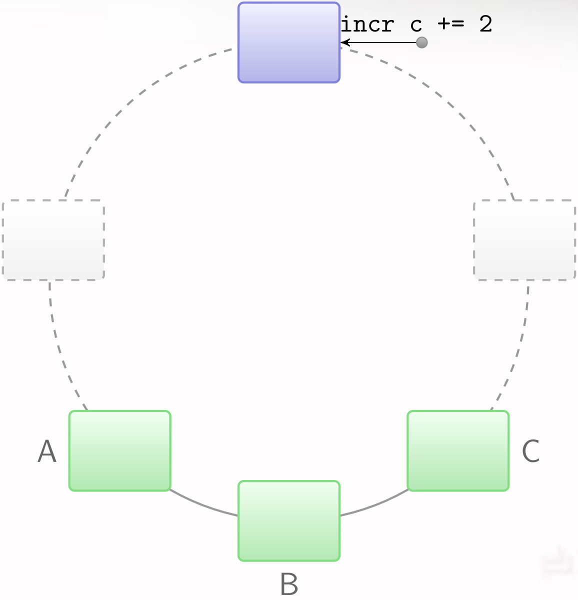


c += 2
A
B
c += 2
$c \begin{bmatrix} A:5(2) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:5(2) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:3(1) \\ B:2(1) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:3(1) \\ B:2(1) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:5(2) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:3(1) \\ B:2(1) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C

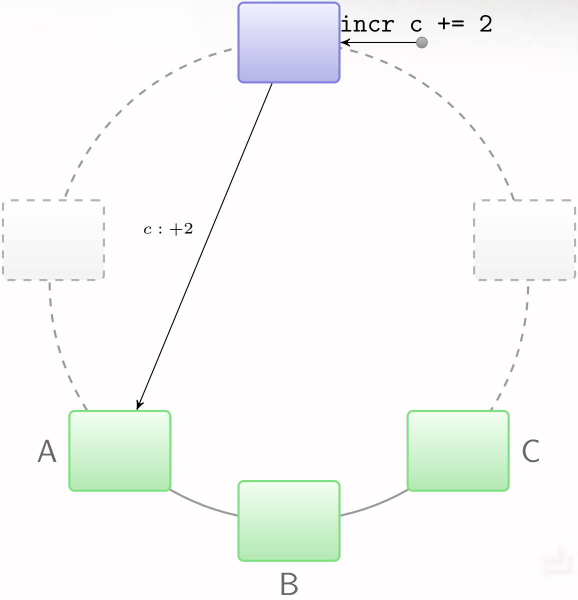
The Cassandra way


A
$c \begin{bmatrix} A:5(2) \\ B:2(1) \\ C:0(0) \end{bmatrix}$
B
$c \begin{bmatrix} A:5(2) \\ B:2(1) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:5(2) \\ B:2(1) \\ C:0(0) \end{bmatrix}$
C

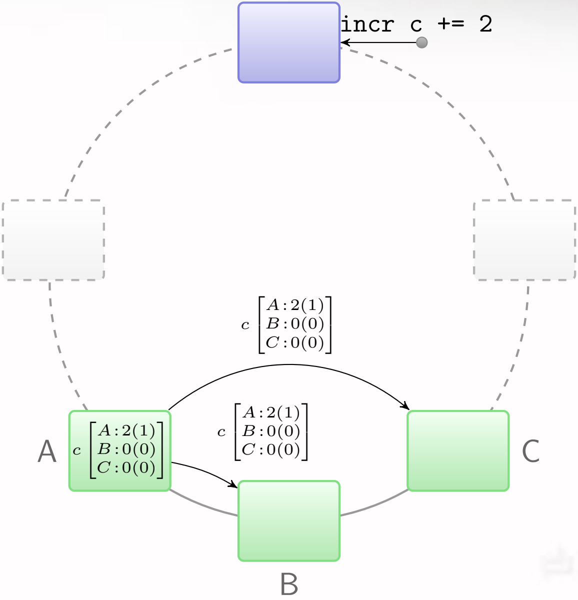
Counter write/read protocol



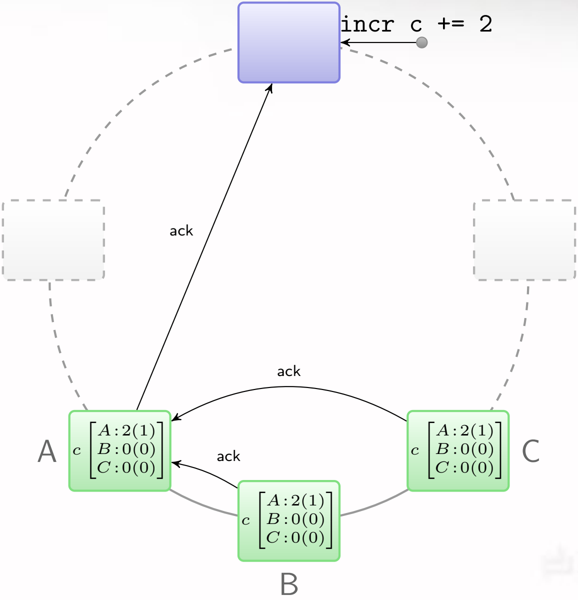
Counter write/read protocol



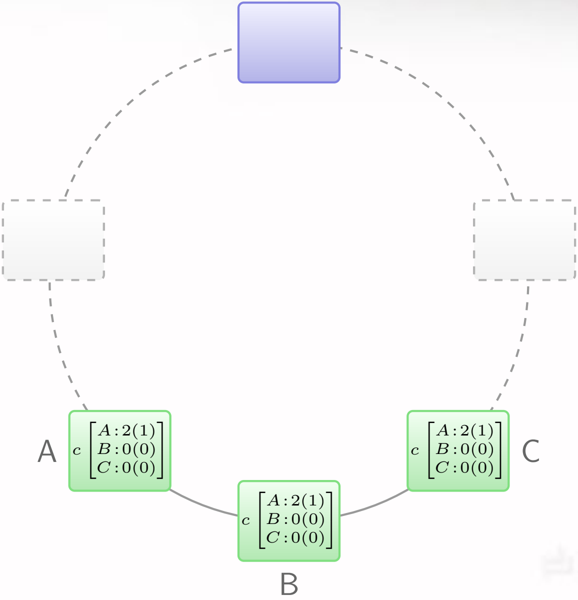
Counter write/read protocol


incr c += 2
A
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C
B

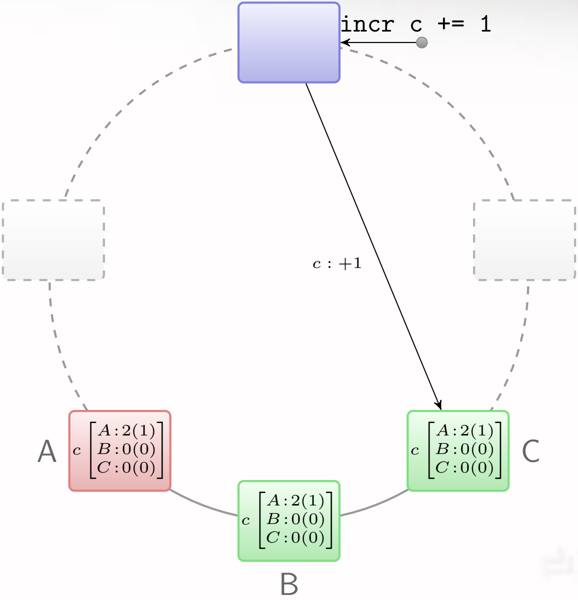
Counter write/read protocol


incr c += 2
ack
ack
ack
A
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$

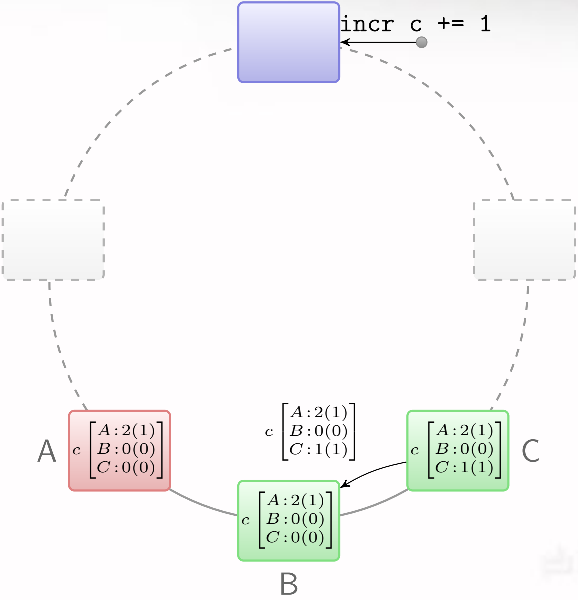
Counter write/read protocol


A
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$

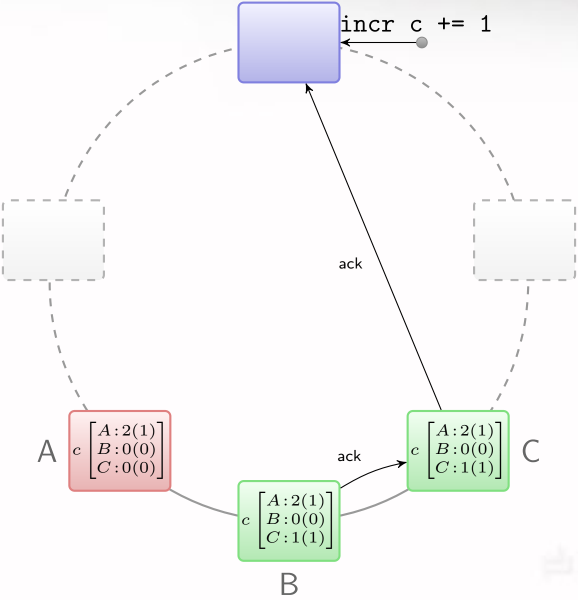
Counter write/read protocol


incr c += 1
$c : +1$
A
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
C
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$

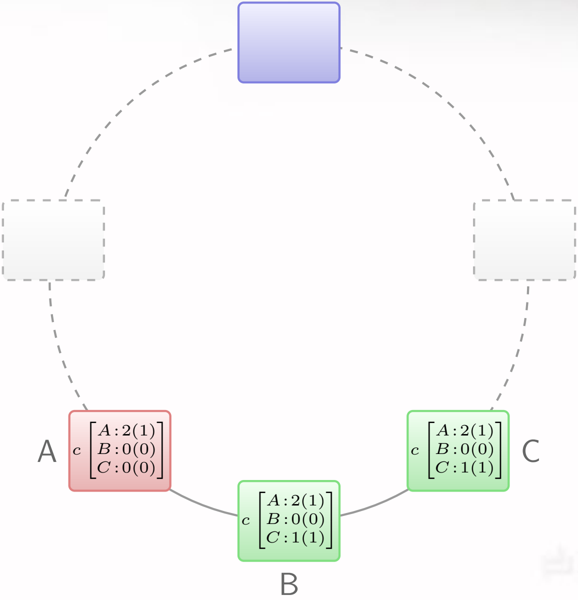
Counter write/read protocol


incr c += 1
A
$c\begin{bmatrix}A:2(1)\\B:0(0)\\C:0(0)\end{bmatrix}$
$c\begin{bmatrix}A:2(1)\\B:0(0)\\C:1(1)\end{bmatrix}$
$c\begin{bmatrix}A:2(1)\\B:0(0)\\C:1(1)\end{bmatrix}$
C
$c\begin{bmatrix}A:2(1)\\B:0(0)\\C:0(0)\end{bmatrix}$
B

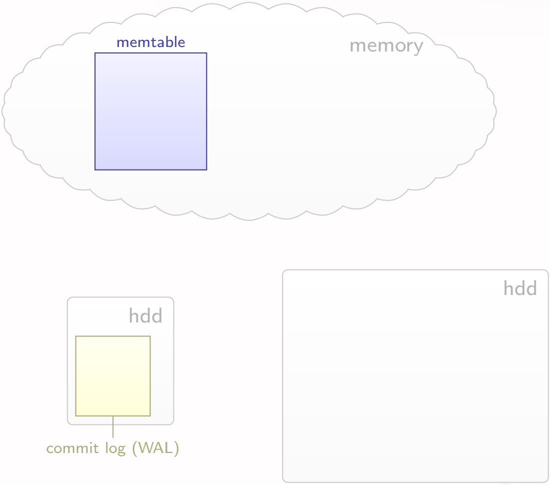
Counter write/read protocol


incr c += 1
ack
ack
A
c $\begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
c $\begin{bmatrix} A:2(1) \\ B:0(0) \\ C:1(1) \end{bmatrix}$
C
c $\begin{bmatrix} A:2(1) \\ B:0(0) \\ C:1(1) \end{bmatrix}$

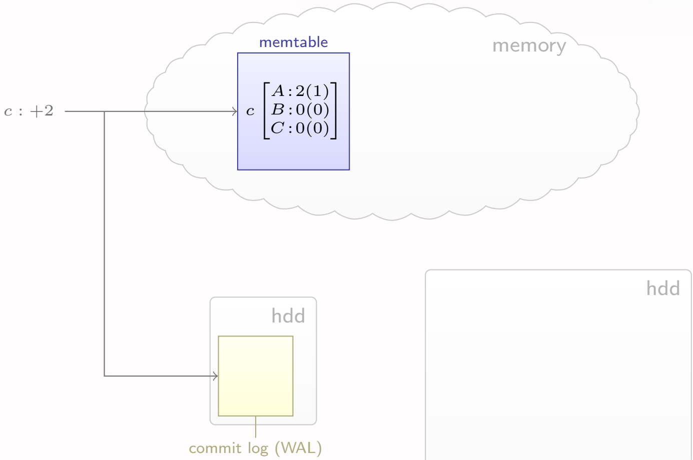
Counter write/read protocol


A
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
B
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:1(1) \end{bmatrix}$
C
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:1(1) \end{bmatrix}$

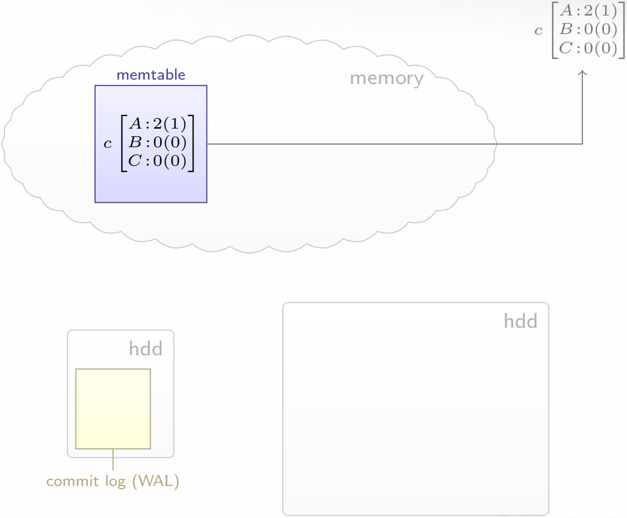
Node write/read path



Node write/read path

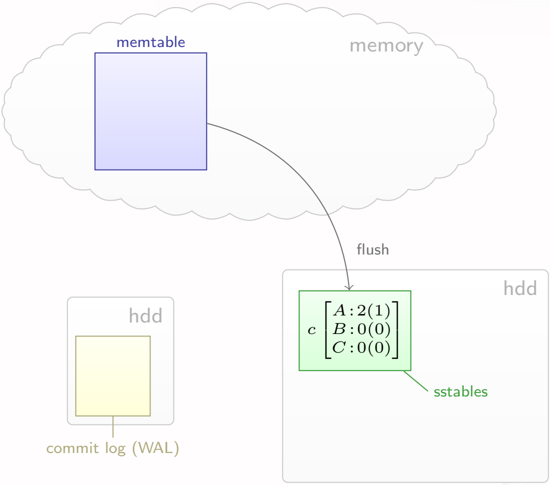

memtable
memory
$c : +2$
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
hdd
hdd
commit log (WAL)

Node write/read path

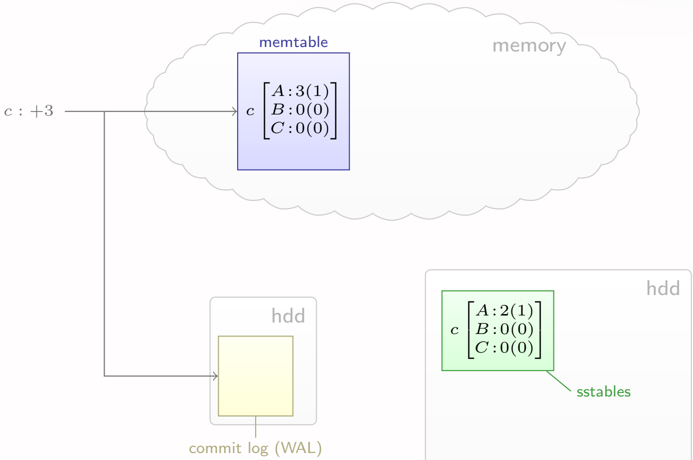


$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
memtable
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
memory
hdd
commit log (WAL)
hdd

Node write/read path

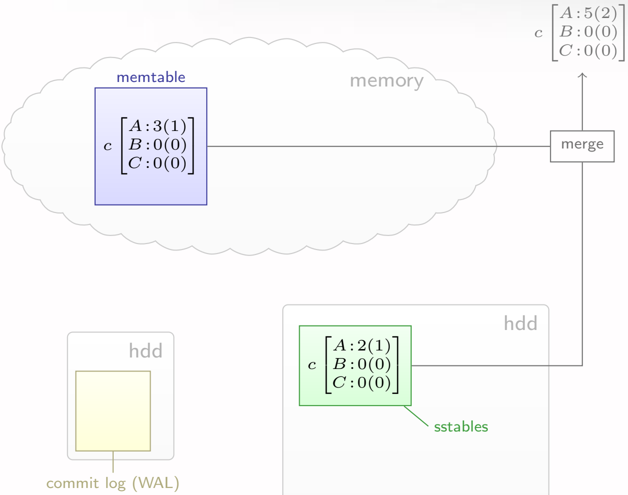

memtable
memory
flush
hdd
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
sstables
hdd
commit log (WAL)

Node write/read path

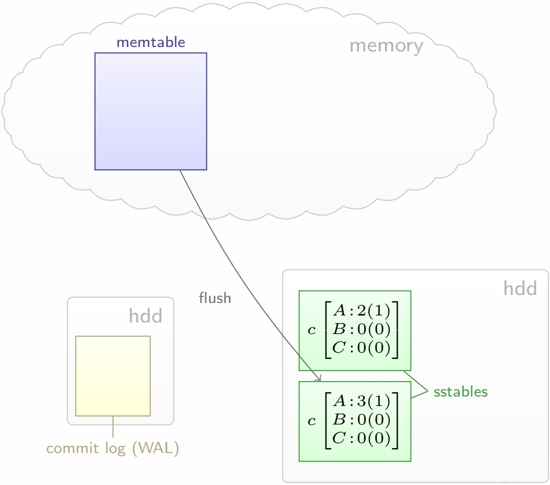


memtable
memory
$c : +3$
$c \begin{bmatrix} A : 3(1) \\ B : 0(0) \\ C : 0(0) \end{bmatrix}$
hdd
hdd
$c \begin{bmatrix} A : 2(1) \\ B : 0(0) \\ C : 0(0) \end{bmatrix}$
sstables
commit log (WAL)

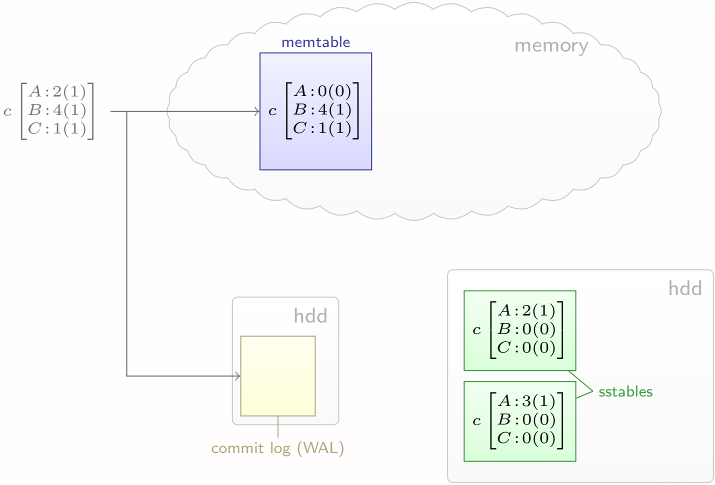
Node write/read path


$c \begin{bmatrix} A:5(2) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
memtable
memory
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
merge
hdd
hdd
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
sstables
commit log (WAL)

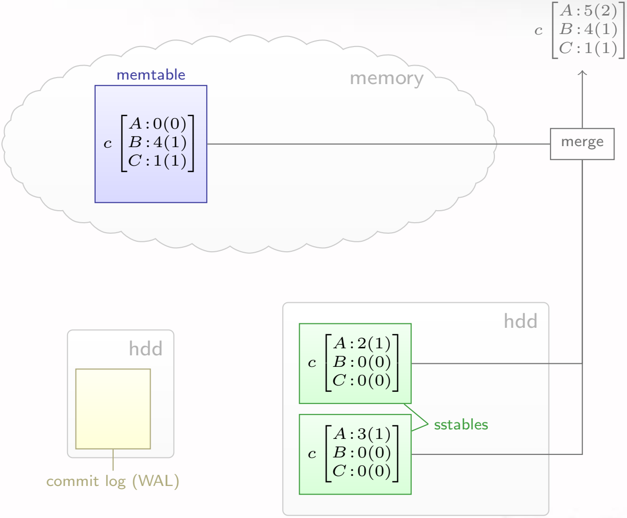
Node write/read path


memtable
memory
flush
hdd
hdd
commit log (WAL)
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
sstables

Node write/read path

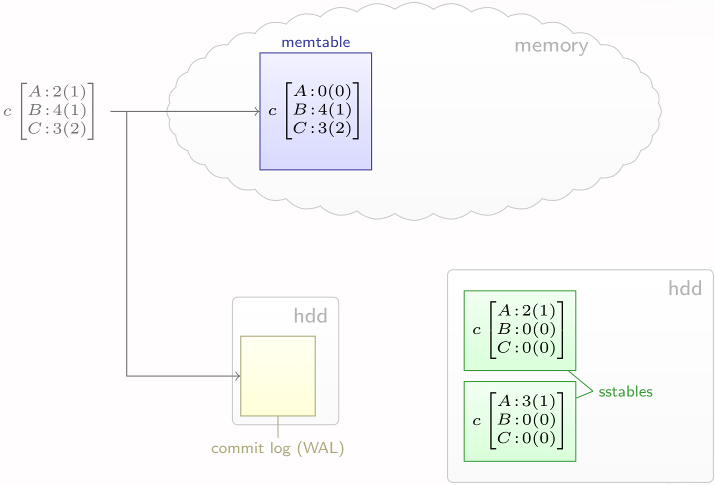

$c\begin{bmatrix}A:2(1)\\B:4(1)\\C:1(1)\end{bmatrix}$
memtable
$c\begin{bmatrix}A:0(0)\\B:4(1)\\C:1(1)\end{bmatrix}$
memory
hdd
commit log (WAL)
hdd
$c\begin{bmatrix}A:2(1)\\B:0(0)\\C:0(0)\end{bmatrix}$
sstables
$c\begin{bmatrix}A:3(1)\\B:0(0)\\C:0(0)\end{bmatrix}$

Node write/read path



$c \begin{bmatrix} A:5(2) \\ B:4(1) \\ C:1(1) \end{bmatrix}$
memtable
memory
$c \begin{bmatrix} A:0(0) \\ B:4(1) \\ C:1(1) \end{bmatrix}$
merge
hdd
hdd
$c \begin{bmatrix} A:2(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
sstables
$c \begin{bmatrix} A:3(1) \\ B:0(0) \\ C:0(0) \end{bmatrix}$
commit log (WAL)

Node write/read path



$$c\begin{bmatrix}A:2(1)\\B:4(1)\\C:3(2)\end{bmatrix}$$

$$c\begin{bmatrix}A:0(0)\\B:4(1)\\C:3(2)\end{bmatrix}$$

$$c\begin{bmatrix}A:2(1)\\B:0(0)\\C:0(0)\end{bmatrix}$$

$$c\begin{bmatrix}A:3(1)\\B:0(0)\\C:0(0)\end{bmatrix}$$

Replication

- Involves one read on the first replica (independently of RF or CL).
- No synchronization ("lock free").
- At CL.ONE, the read is not accounted in the latency of the increment operation.

Limitations

- In Cassandra, if a write fails (TimeoutException), the client does not know if it was persisted. This holds for counters, but replaying an increment has a risks for over-counts.
- Counter removal is limited. You can remove a counter, but you should not increment it afterwards (the behavior if you do is undefined). To reset a counter, insert *-value*.
- Dealing with the loss of a SSTable is less convenient: you will need to remove all the SSTables for the column family (and then repair).
- No TTL support for counter columns.
- No secondary indexes for counter columns.
- Only full column family of counters so far.
- No CL.ANY.

Last words

If you have lot to count, very quickly, you'll love counters.

Last words

If you have lot to count, very quickly, you'll love counters.

Questions?