INTRODUCTION TO CASSANDRA

Aaron Morton
@aaronmorton
www.thelastpickle.com

This is an introduction, not a reference. Technical details have been simplified or omitted.

Cassandra is a distributed, fault tolerant, scalable, column oriented data store.

The data model and on disk storage are inspired by Google Big Table.

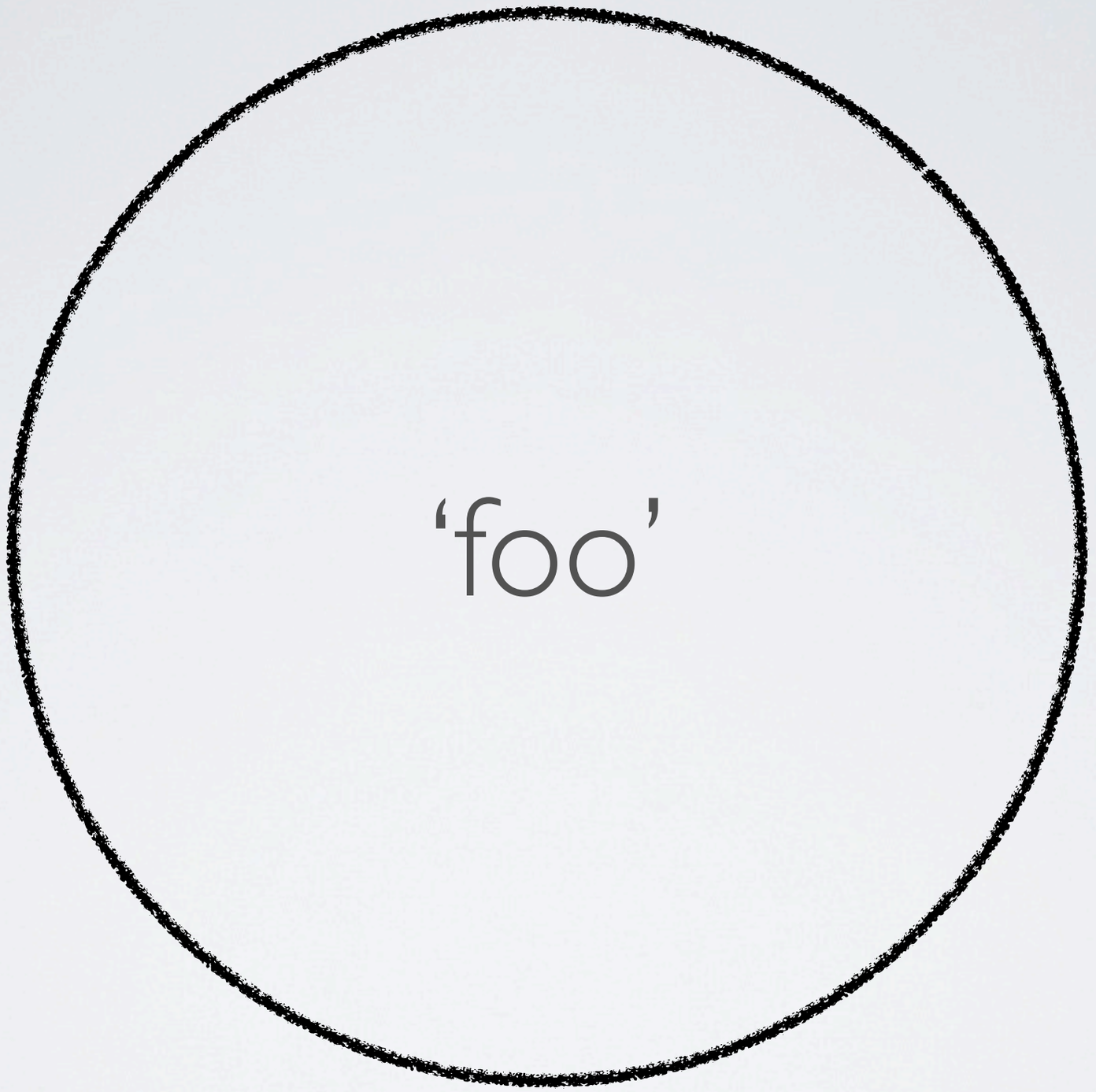
The distributed cluster is inspired by Amazon Dynamo.

First, a word about the column oriented data model.

It's different to a relational database like MySql.

For now let's say rows have a key and each row can have different columns.

Let's store the 'foo' key in our system.


‘foo’

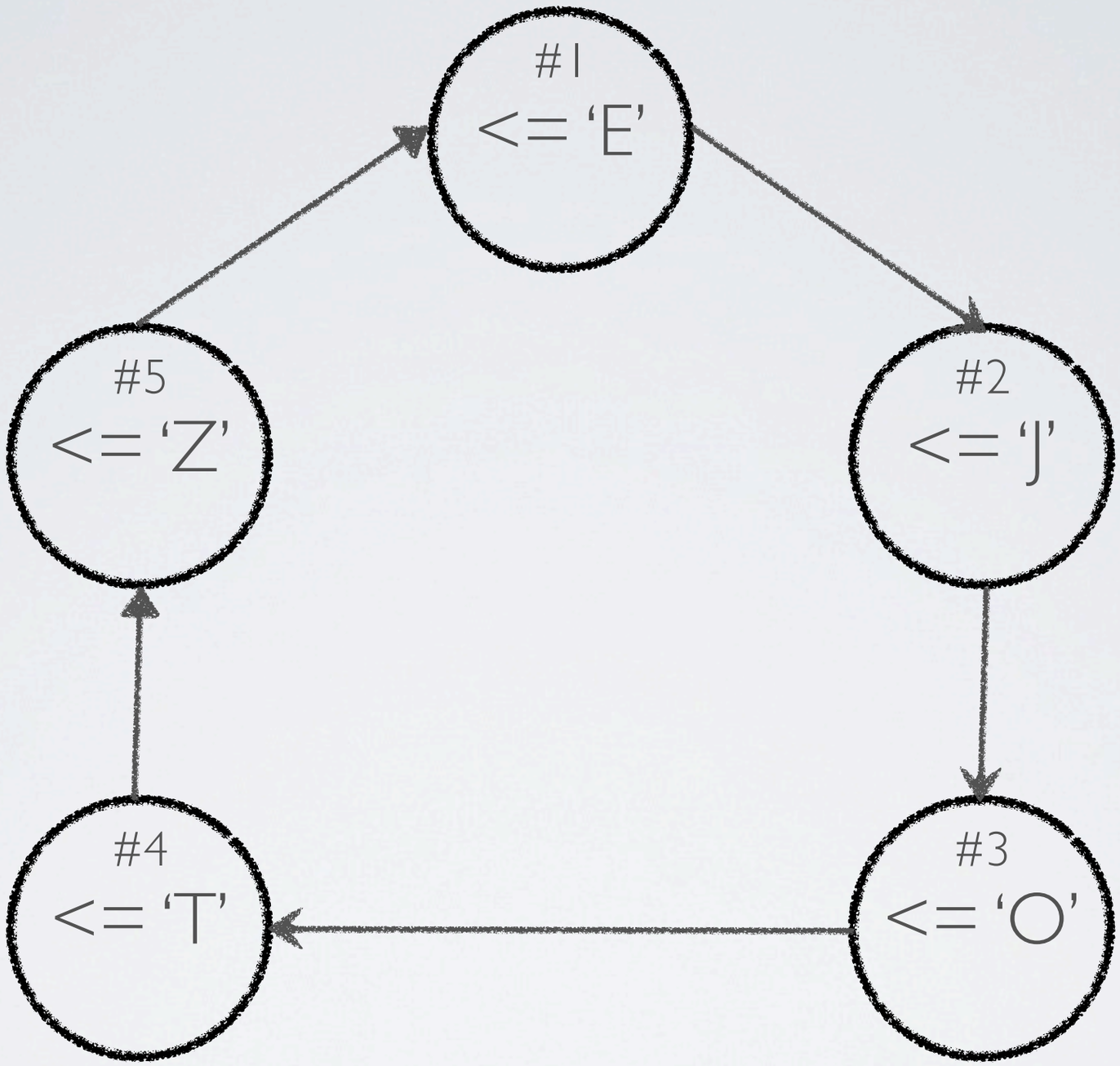
Done. But we want to be able to read it back if that one machine fails.

Let's distribute the value to 3 of the 5 nodes we have.

This is the Replication Factor, called RF or N.

We need to know which nodes the key was written to. So we know where to read it from later.

Each node has an Initial Token that identifies the upper value of the key range it is responsible for.

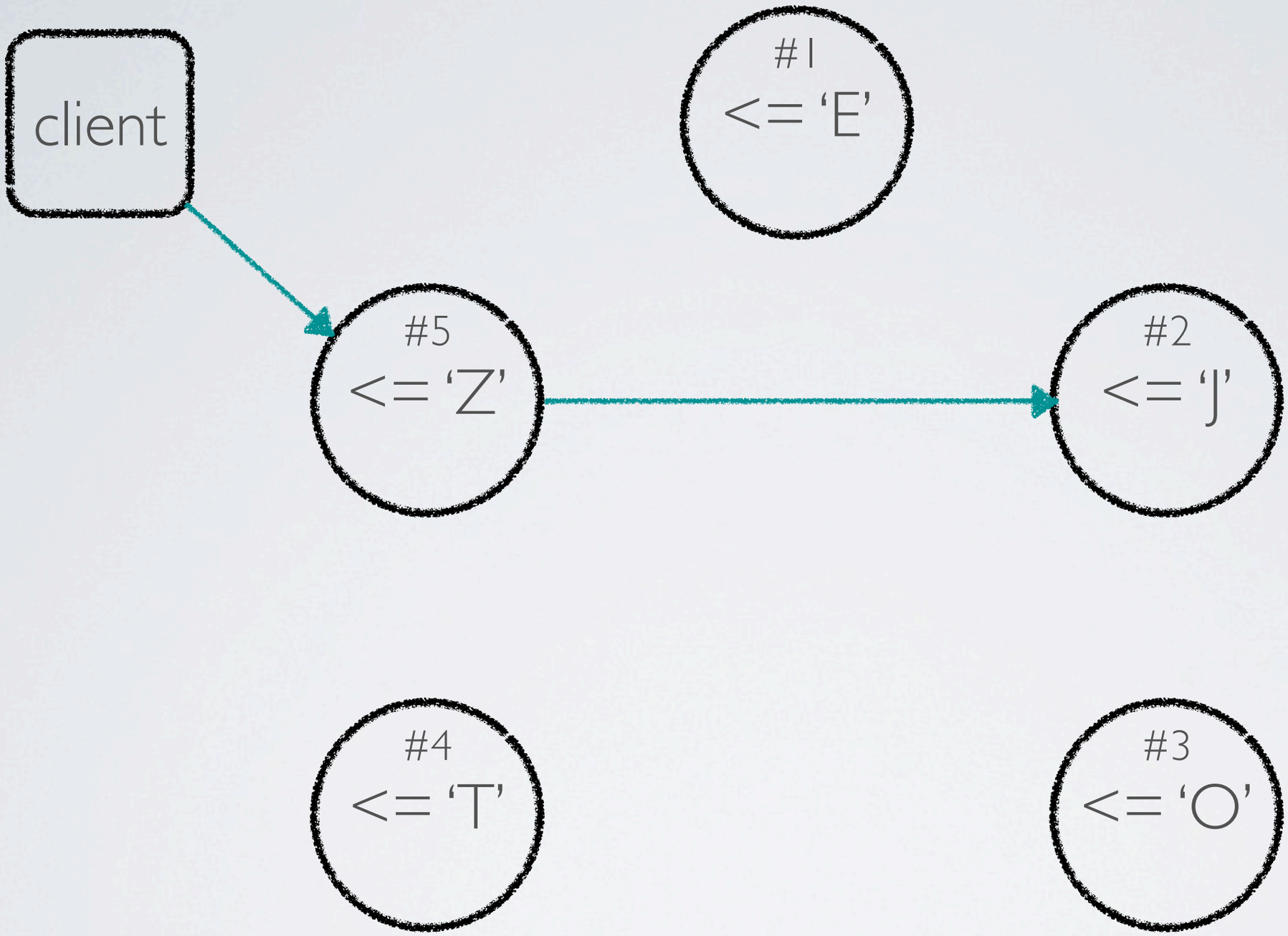

#1
<= 'E'
#5
<= 'Z'
#2
<= 'J'
#4
<= 'T'
#3
<= 'O'

A Gossip protocol is used to allow each node to know about all other nodes in the cluster. Including their initial token.

A client can connect to any node in the cluster and ask it to store the ‘foo’ key.

The Coordinator node for the client will know which node is responsible for the ‘foo’ key.

Our client connects to node 5 which knows that node 2 is responsible for the key range that contains ‘foo’.


client
#1
<= 'E'
#5
<= 'Z'
#2
<= 'J'
#4
<= 'T'
#3
<= 'O'

If we have lots of keys between ‘F’ and ‘J’, node 2 may get overloaded.

A Partitioner is used to transform the key.

Transforming the keys can result in two lexically close keys (e.g. ‘foo’ and ‘foo2’) are stored on different nodes.

Transformed keys are also used as the Initial Tokens for nodes.

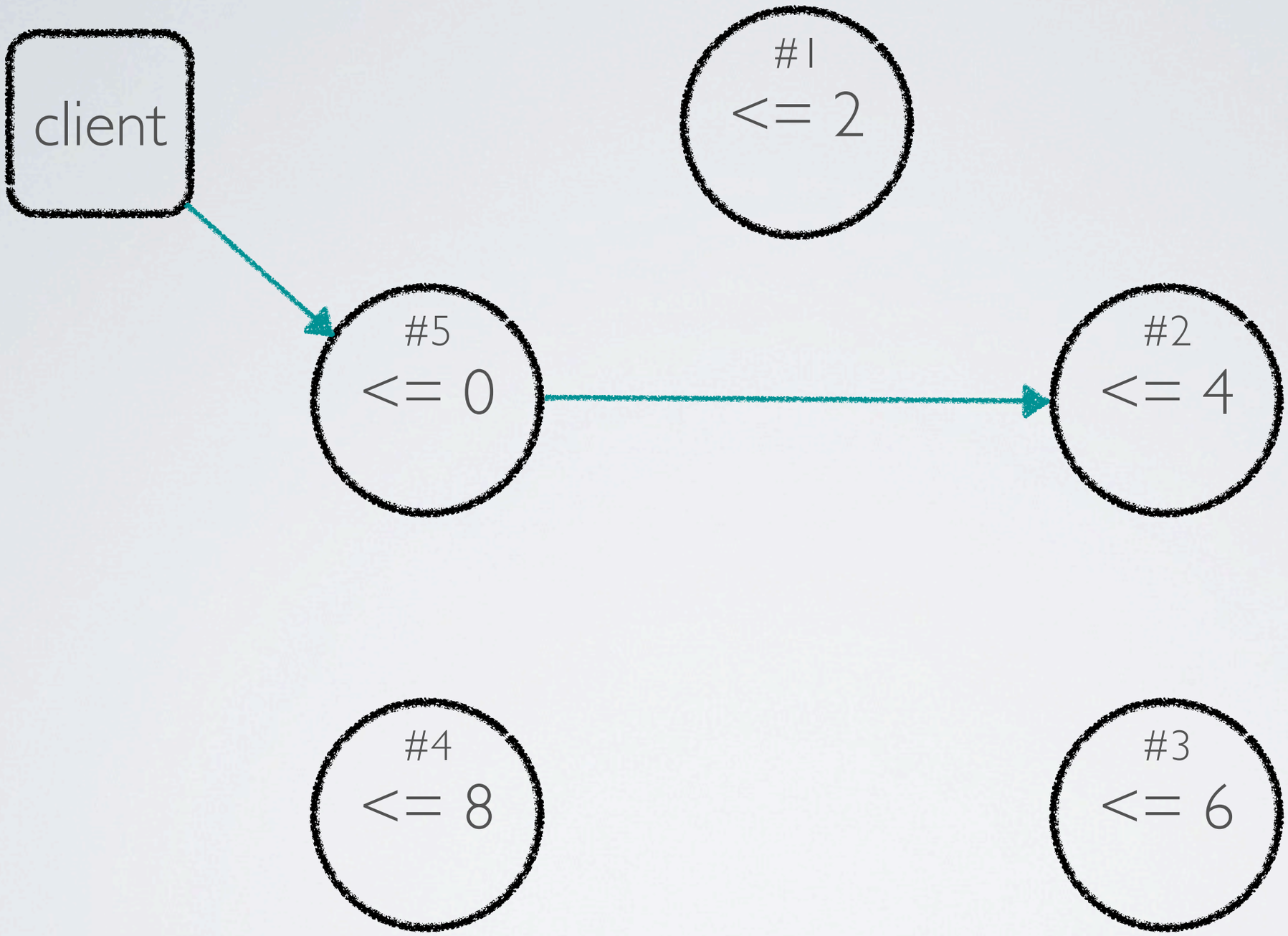
The Random Partitioner applies a MD5 transform to the key. The range of possible keys is transformed to a 128 bit integer.

There is also a Byte Order Partitioner which orders keys according to their byte values.

Start with the Random Partitioner.

In this example all our keys are transformed to an integer between 0 and 9, and our ‘foo’ key transforms to 3.

Our client connects to node 5, which transforms the key and determines node 2 is responsible for that range of keys.


client
#1
<= 2
#5
<= 0
#2
<= 4
#4
<= 8
#3
<= 6

But where are the 3 replicas?

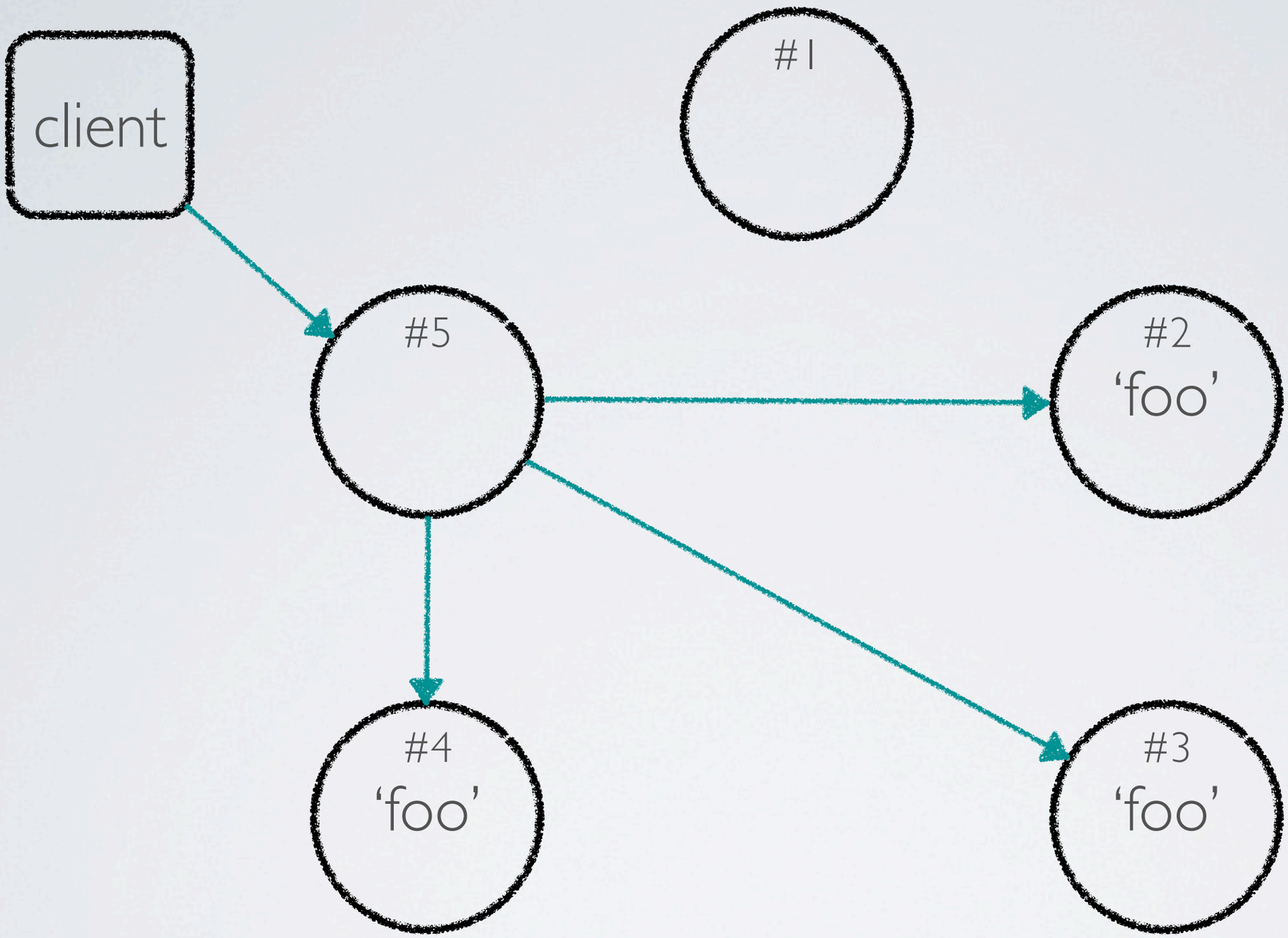
A Replica Placement Strategy determines which nodes should contain replicas.

The Simple Strategy orders the nodes by their initial token and places the replicas clockwise around the ring of nodes.

The Network Topology Strategy can place the replicas in different Data Centres and on different Racks.

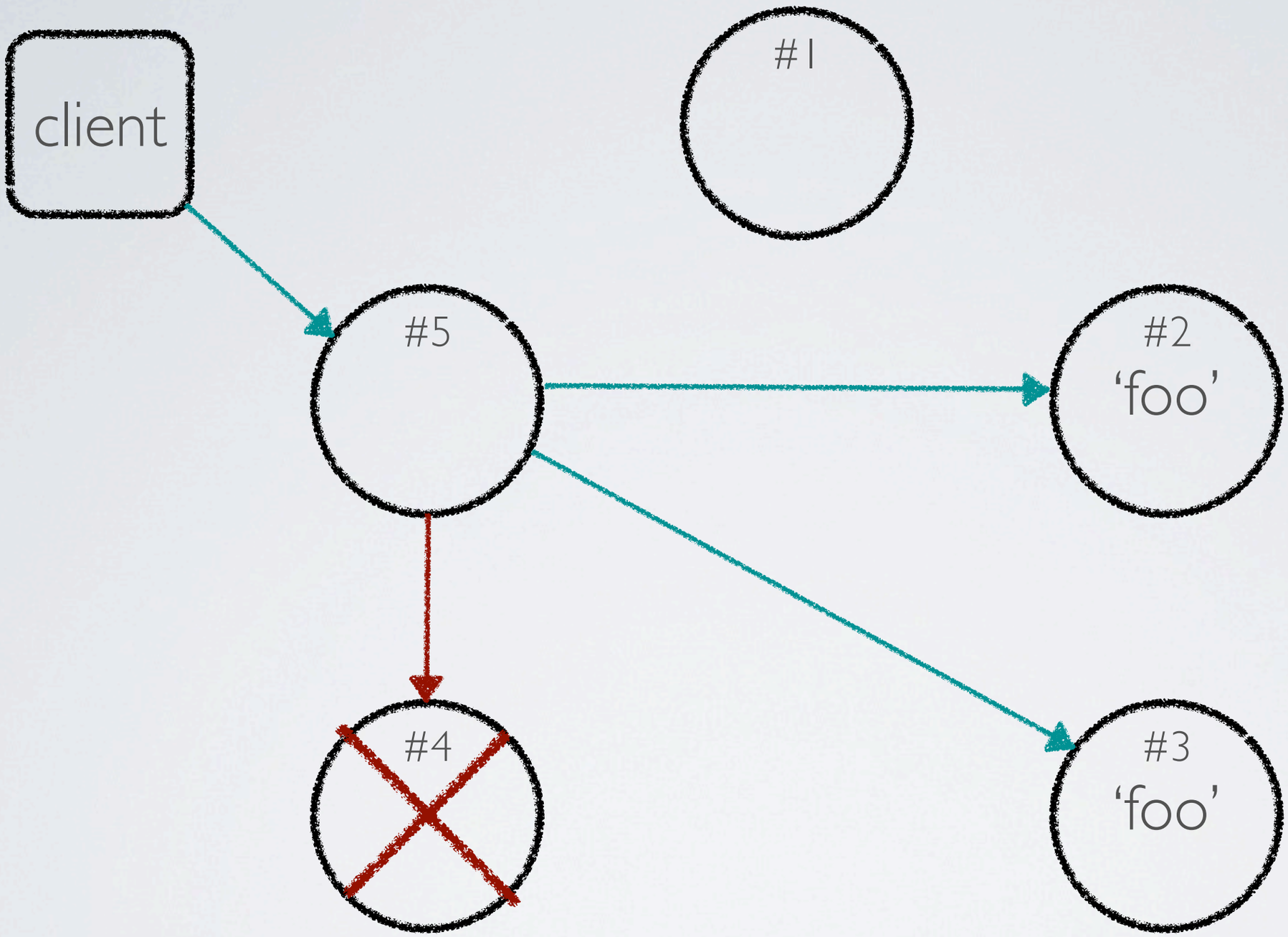
Start with the Simple Strategy.

Using the Replica Placement Strategy our coordinator will send the ‘foo’ key to 3 nodes in parallel.


client
#1
#5
#2
'foo'
#4
'foo'
#3
'foo'

When the 3 nodes tell the coordinator they have completed, it will tell the client the write has completed.

What about fault tolerance?
What if node 4 is down?


client
#1
#5
#2
'foo'
#4
#3
'foo'

The Consistency Level (CL) supplied by the client specifies how many nodes must agree for an operation to be successful.

For Reads this is number is known as R.

For Writes it is known as W.

The common Consistency Levels are One, Quorum and All.

Quorum is (N/2) +1.

If the operation succeeds it means that the required Consistency Level has been achieved.

No matter what the Consistency Level, the cluster will work to **Eventually** make the on disk data for all replicas **Consistent**.

To get consistent behaviour for all operations ensure that

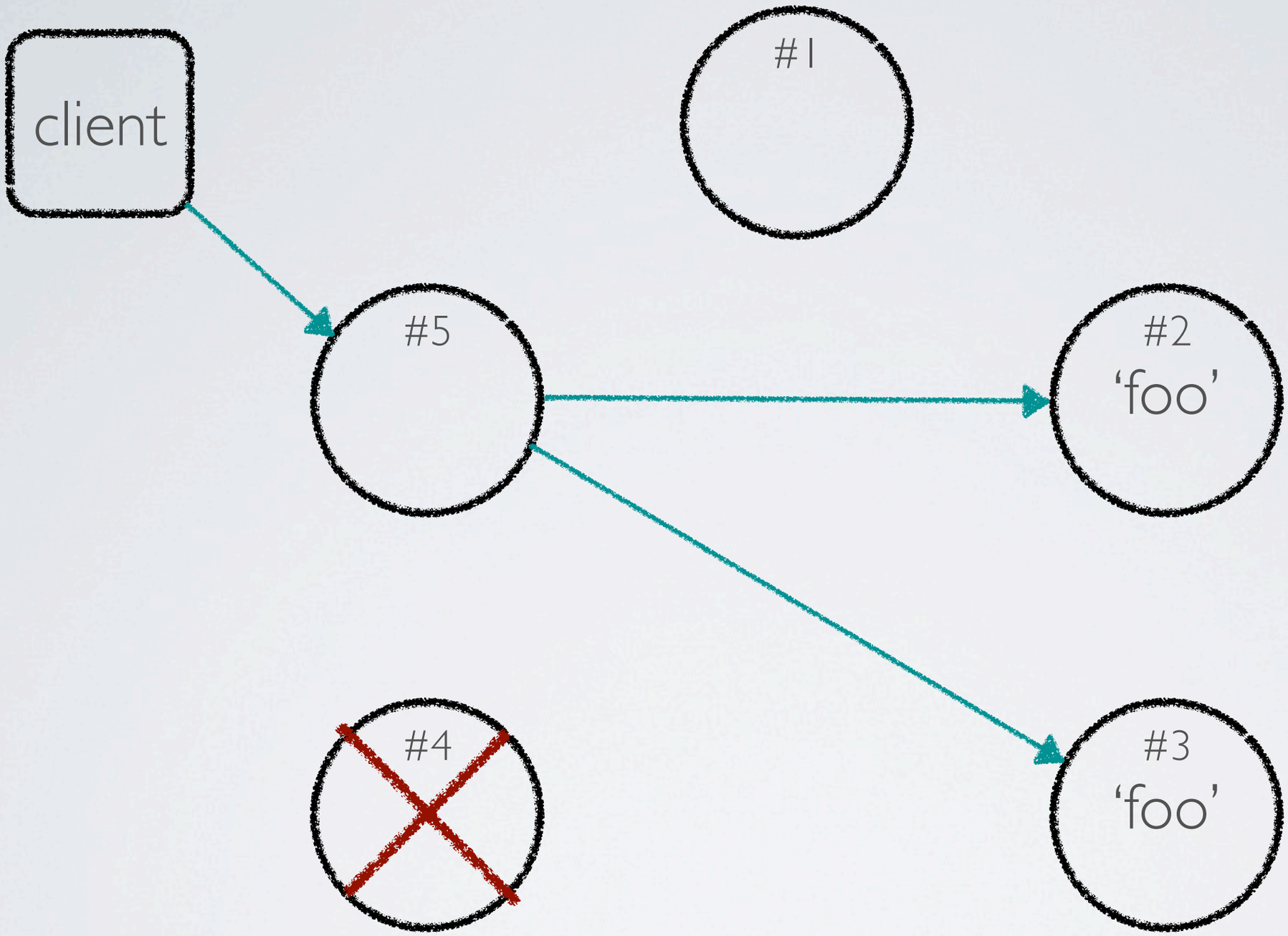
$$R + W > N$$

Using Quorum CL for Writes and Reads will give consistent operations.

Using All CL for Writes and One for Reads will give consistent operations.

Using One CL for Writes and All for Reads will give consistent operations.

If we use Quorum CL for our write, the coordinator will wait for 2 nodes to complete.


client
#1
#5
#2
'foo'
#4
#3
'foo'

If a node is offline the coordinator will include a Hinted Handoff when it sends the write to one of the online nodes.

The Hinted Handoff tells a node that it should forward the write to the offline node if it comes back online.

In our example node 5 gave node 2 a Hinted Handoff for node 4.

client

#1

#5

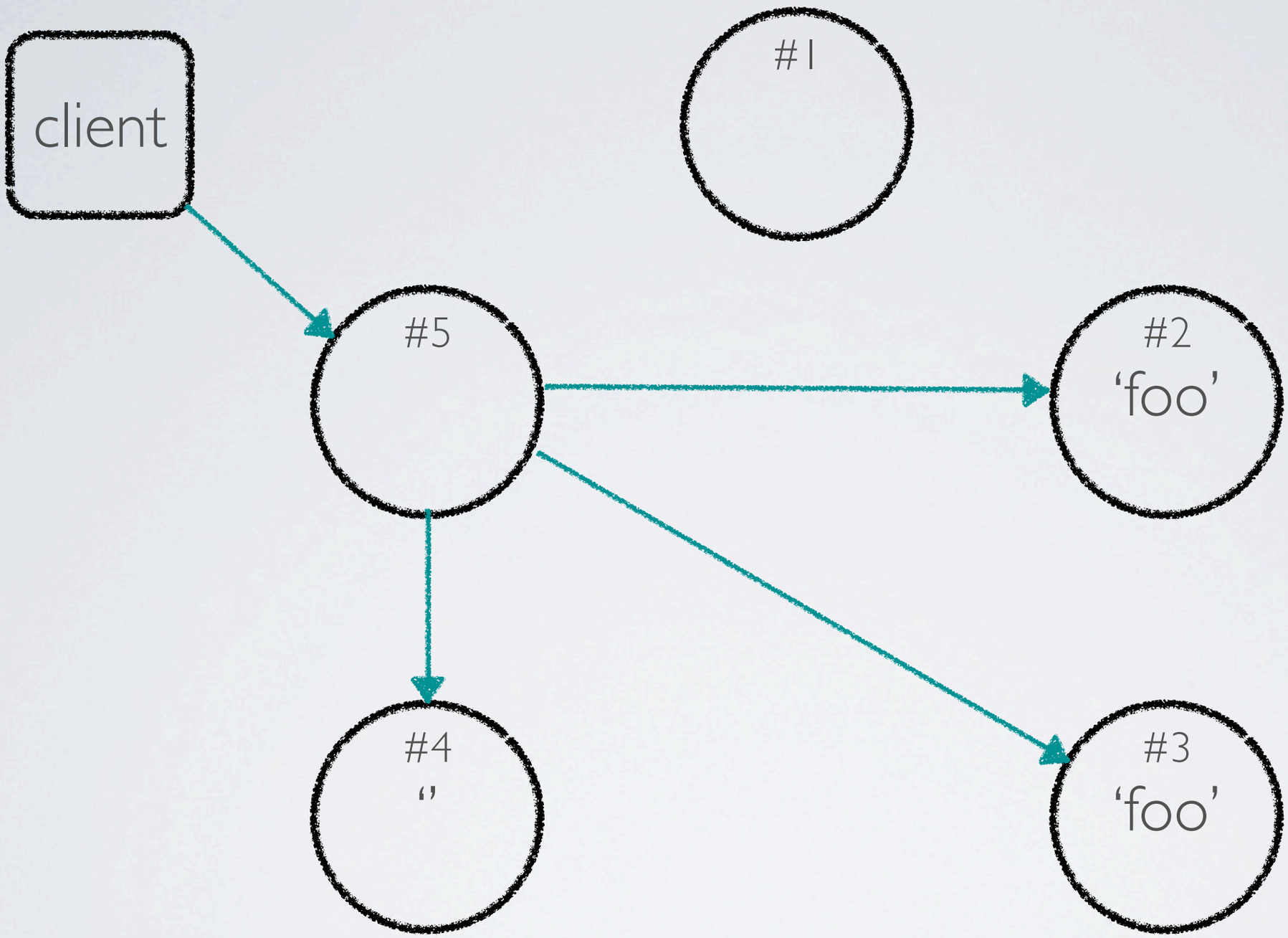
#2
'foo'

#4
'foo'

#3
'foo'

What if the ‘foo’ key is read before the Hinted Handoff is delivered?

Or if node 4 died during the write and the Hinted Handoff was not sent?


client
#1
#5
#2
'foo'
#4
''
#3
'foo'

If using Quorum CL, the coordinator asks all replicas to perform the read and waits for the Quorum of replicas to return.

If the results received do not match a Read Repair process is performed before returning to the client.

Read Repair uses the time stamp provided by the client during a write to determine the latest value.

The current value is pushed to out of date nodes and consistency is achieved before the coordinator returns to the client.

At lower Consistency Levels Read Repair happens in the background and can be assigned a probability of running.

Hinted Handoff and Read Repair are optimisations for achieving on disk consistency.

The Anti Entrophy Service (AES) is the main feature for achieving consistency.

AES is normally run as a regular weekly maintenance. Or after node downtime.

AES detects differences by generating hash trees (known as Merkle Trees) that describe a node's content.

Ranges of out of sync data are then exchanged between the nodes.

How does it scale out?

Nodes can be added to a cluster while it is online. Known as Bootstrapping.

Adding nodes increases the disk and memory available to the cluster, and decreases the load per node.

Adding nodes increases the disk and memory available to the cluster, and decreases the load per node.

Apache Cassandra
http://cassandra.apache.org/

Google Big Table
http://labs.google.com/papers/bigtable.html

Amazon Dynamo
http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

Aaron Morton
@aaronmorton
www.thelastpickle.com