



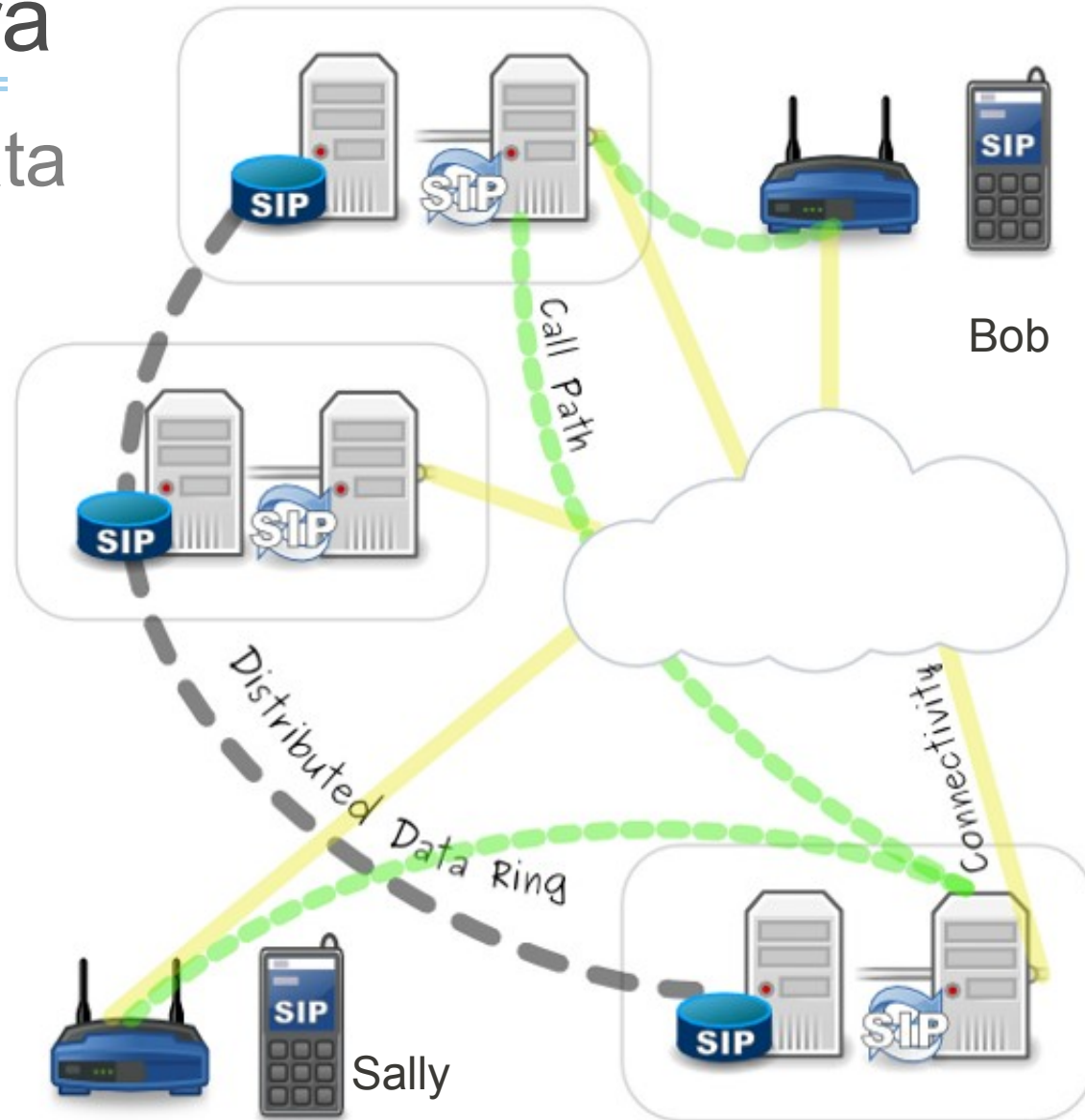
Distributed Data with Cassandra Network Topology Strategy

Outline

- Use case for multi data center Cassandra deployment
 - Use Case: Sourcing and Locality
 - OnSIP and Cassandra
- Cassandra 101
 - Tokens
 - SimpleStrategy Algorithm
- Snitches and Strategies
 - Locality
 - Algorithm
- Write placement internals
 - CalculateNaturalEndpoints()
 - Collections.binarysearch()
 - ringIterator()
 - firstTokenIndex()
- NetworkTopologyStrategy
 - Algorithm overview
 - Core differences from other strategies
 - Token Selection: Logical rings and offset mirror token selection

OnSIP and Cassandra

- Sourcing distributed data
 - Historical call tracing
- Providing data locality
 - Distributed registration



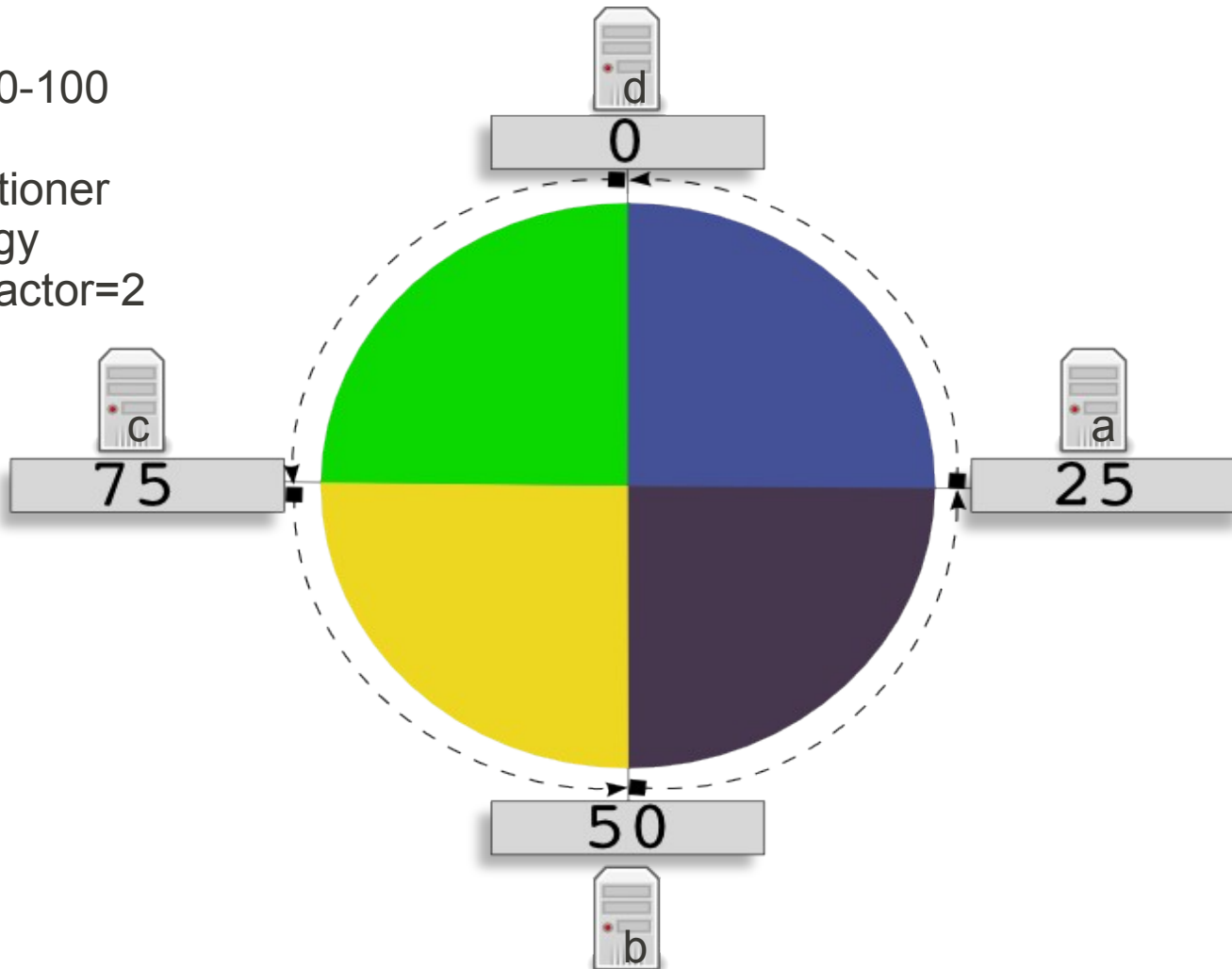


Cassandra 101

- Tokens
 - Data to be stored is consistently hashed to a token value
 - Token Range of $0 - 2^{127}$
 - Every node gets assigned an Initial Token which defines the end of the token range it is responsible for
- SimpleStrategy Algorithm
 - Write to the first node whose token is larger than the data's token. If no token exists that is larger, write to the node with the lowest token.

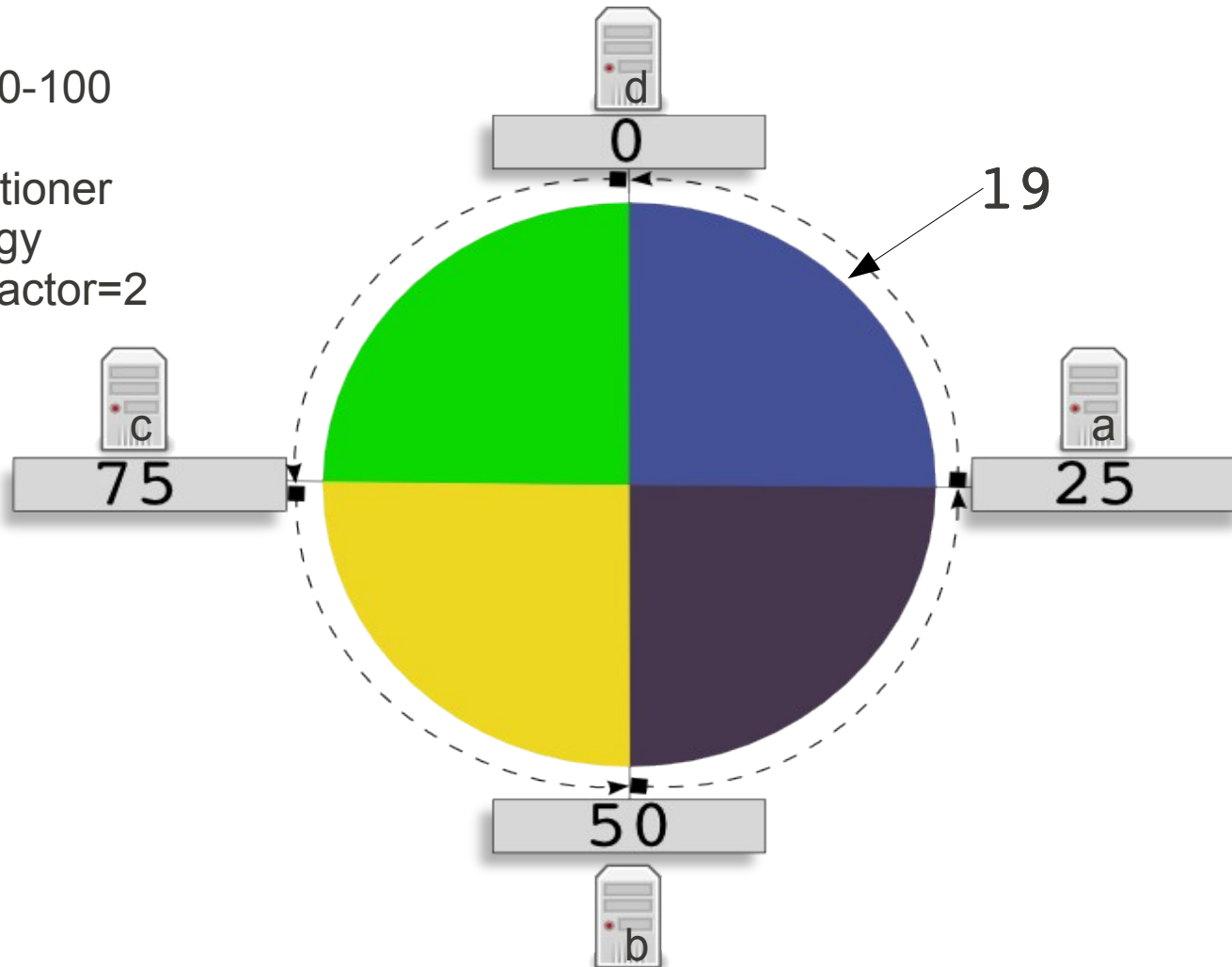
Cassandra 101

Token range 0-100
 4 Nodes
 RandomPartitioner
 SimpleStrategy
 Replication Factor=2



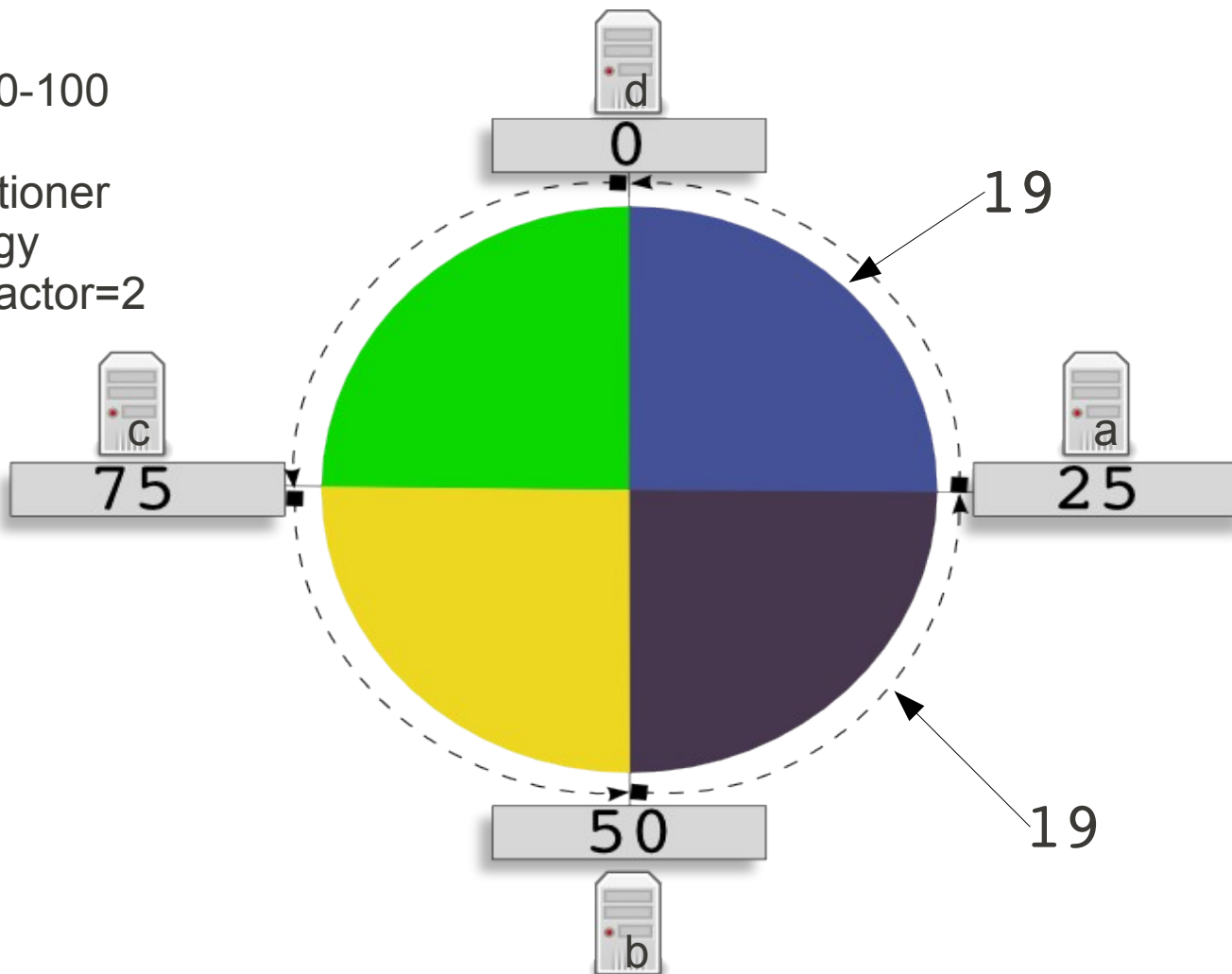
Cassandra 101

Token range 0-100
 4 Nodes
 RandomPartitioner
 SimpleStrategy
 Replication Factor=2



Cassandra 101

Token range 0-100
 4 Nodes
 RandomPartitioner
 SimpleStrategy
 Replication Factor=2



Snitches and Strategies

- Snitches
 - Provide information about nodes relative location
- Strategies
 - Use information from a Snitch to implement an algorithm for gathering endpoints
- `src/java/org/apache/cassandra/locator`
 - `SimpleSnitch.java`
 - `SimpleStrategy.java`

PropertyFileSnitch

```
# Cassandra Node IP=Data Center:Rack
127.0.0.1=DC0:RAC1
127.0.0.2=DC0:RAC2
127.0.0.3=DC1:RAC1
127.0.0.4=DC1:RAC2

# default for unknown nodes
default=DC0:RAC1
```



Strategies

- Implement CalculateNaturalEndpoints()
 - Generally implementation
 - gather a list of available endpoints from the snitch
 - Create a ringIterator from the list and the token to insert
 - Gather endpoints from the iterator until replication factor is met

Write placement internals

- `Collections.binarysearch(endpoints,token)`
- `TokenMetadata.java`
 - `ringIterator(endpoints,token)`
 - `firstTokenIndex(endpoints,token)`



Collections.binarysearch(list,token)

Searches the list for token index, or insertion point

- If token is found, its index is returned
- If token is not found
 - Returns $-(\text{insertionPoint})-1$
 - insertionPoint is the index of the first token in the list that is larger than the supplied token
 - If all tokens in the list are smaller than the supplied token, insertionPoint is `list.size()`

Collections.binarysearch(list,token)

Example1: Collections.binarysearch([0,25,50,75] , 19)

- Token not found
 - InsertionPoint index=1
 - Return $-(1)-1 = -2$

Example2: Collections.binarysearch([0,25,50,75] , 98)

- Token not found
 - All tokens in list are smaller than provided token
 - Return $-(list.size())-1 = -5$

firstTokenIndex(list,token)

```
i = Collections.binarysearch(list,token)

// Processes return value
i = (i+1)*-1
if i >= list.size()
    return 0
else
    return i
```

ringIterator(list, token)

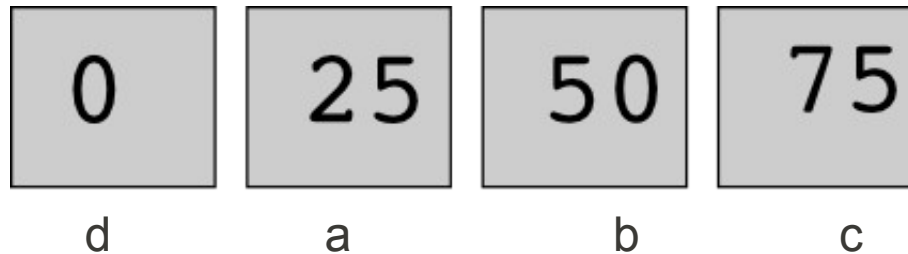
```
j=firstTokenIndex(list, token);
try
{
    return ring.get(j);
}
finally
{
    j++;
    if (j == ring.size())
        j = insertMin ? -1 : 0;
    if (j == startIndex)
        // end iteration
        j = -2;
}
```

NetworkTopologyStrategy Algorithm

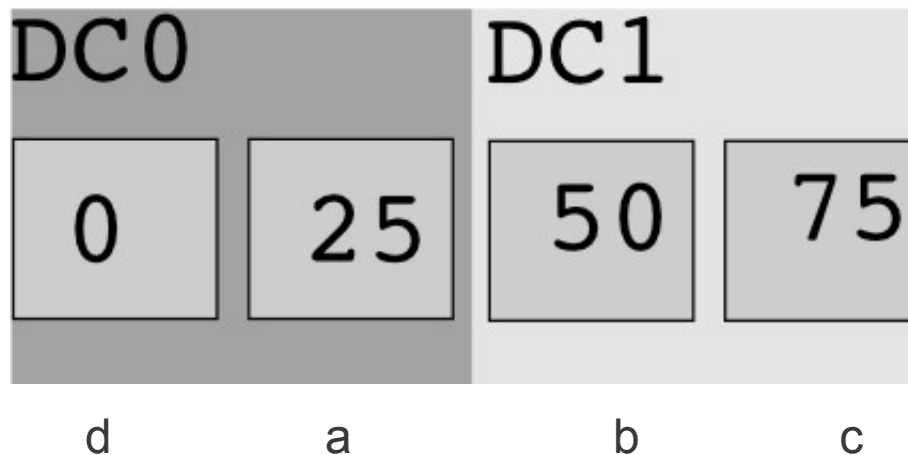
- Get Datacenters from strategy options: {DC0:1,DC1:1}
- For each data center entry
 - Get replication factor
 - Get a list of all endpoints for this datacenter from the snitch
 - Create a ringiterator from the datacenter endpoints list and Collect endpoints to write to – only select an endpoint from the list for any given rack once (distribute across racks)
 - If replication factor has not been met, continue to collect endpoints from the list, allowing racks that already contain an endpoint in the write list
- If our replication factor is not equal to our list of endpoints, throw an error because there are not enough nodes in the data center to meet the replication factor

How NTS is different

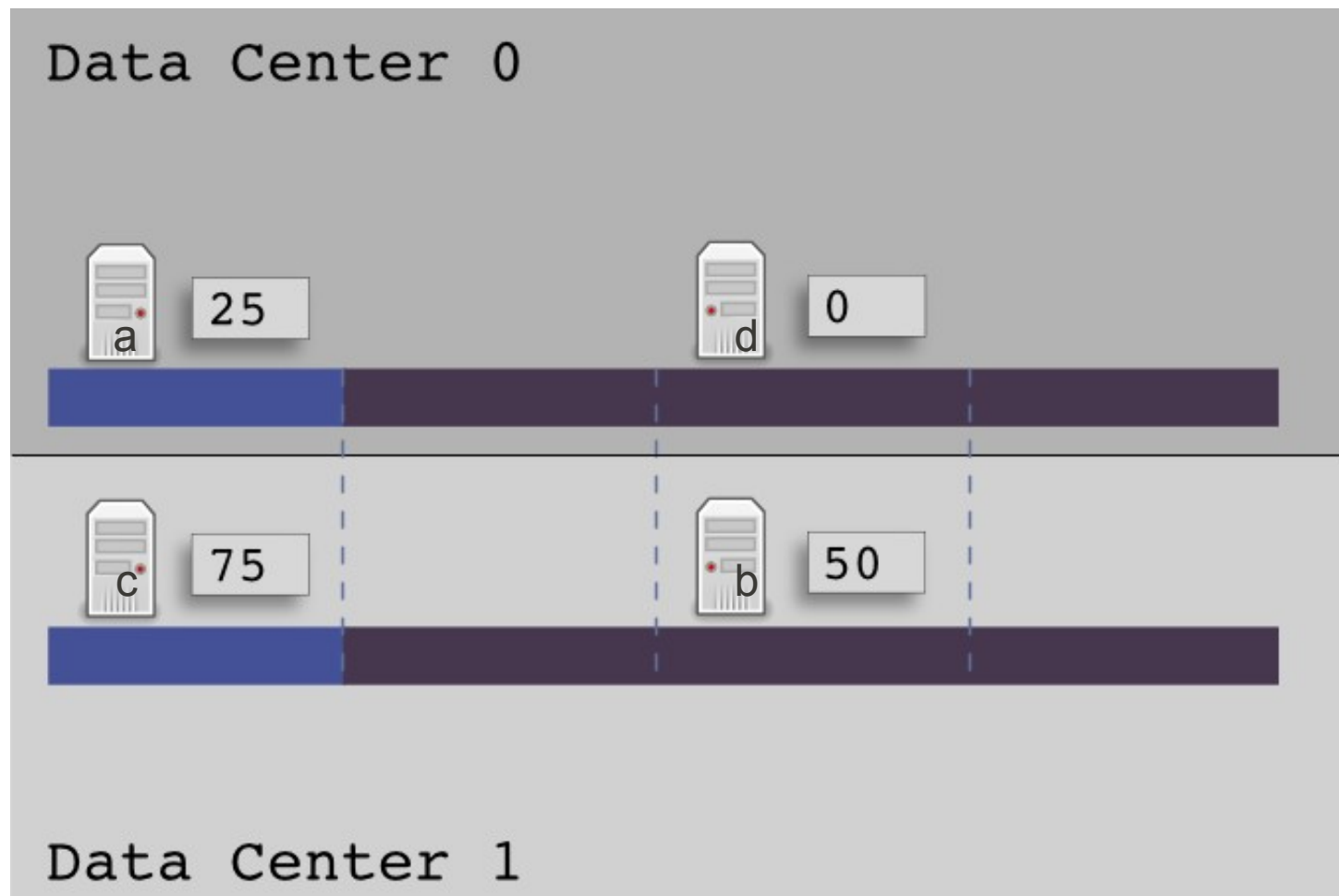
SimpleStrategy



NetworkTopologyStrategy



How NTS is different





NTS Initial Token Selection

- Offset mirroring
 - Use logical rings per datacenter and offset conflicting tokens
 - Calculate the tokens for each datacenter as if it were it's own cluster
 - For the second data center add 1 to each token
 - For the third data center add 2 to each token etc.
 - For data centers with different numbers of nodes, Calculate tokens for each data center individually and offset any collisions.

Mirrored Offset Tokens

Token range 0-100

4 Nodes

2 Datacenters

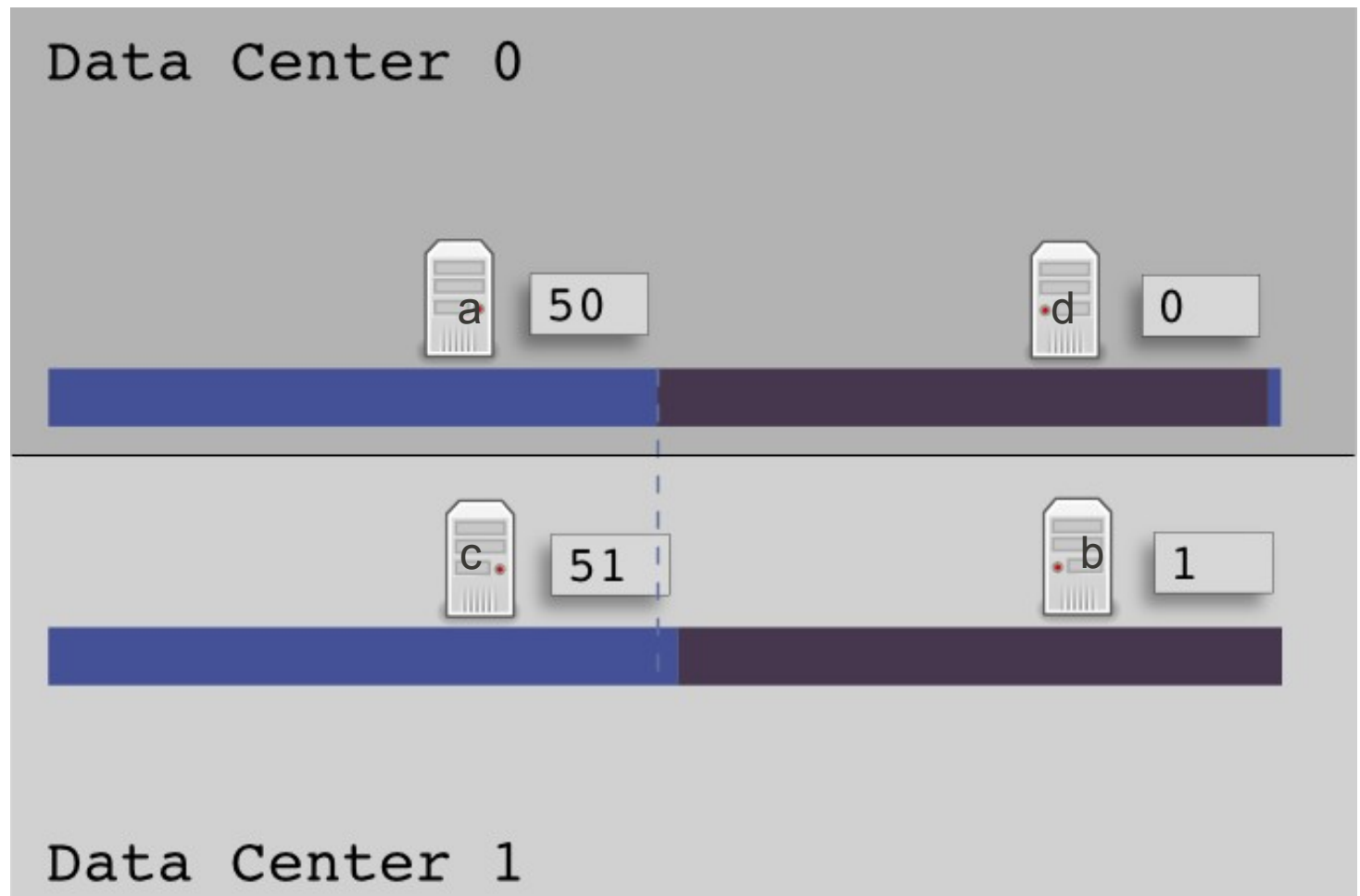
2 Racks per DC

1 Node per rack

RandomPartitioner

NTS

RF=DC0:1,DC1:1



Mirrored Offset Tokens

Token range 0-100

4 Nodes

2 Datacenters

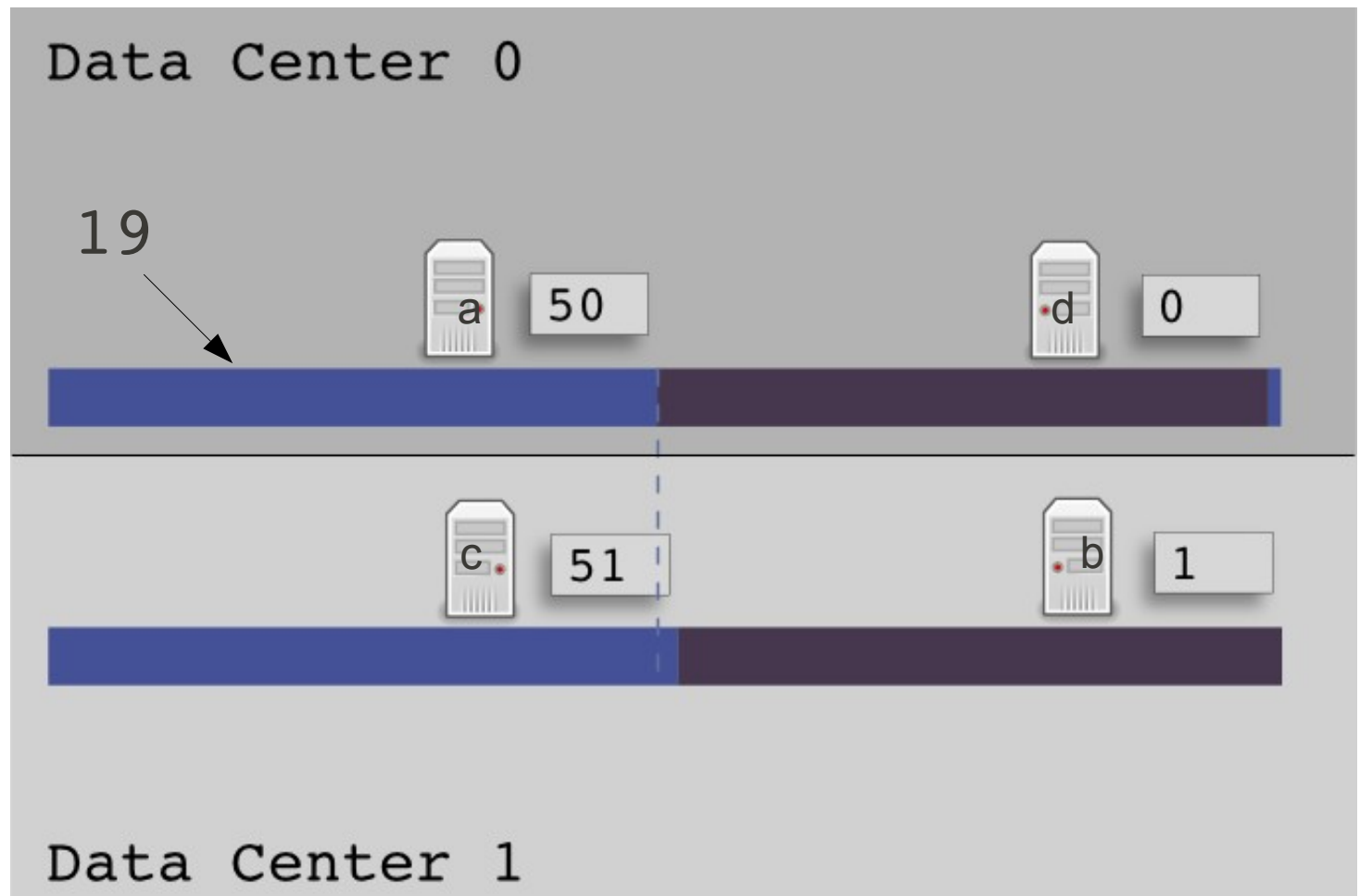
2 Racks per DC

1 Node per rack

RandomPartitioner

NTS

RF=DC0:1,DC1:1



Mirrored Offset Tokens

Token range 0-100

4 Nodes

2 Datacenters

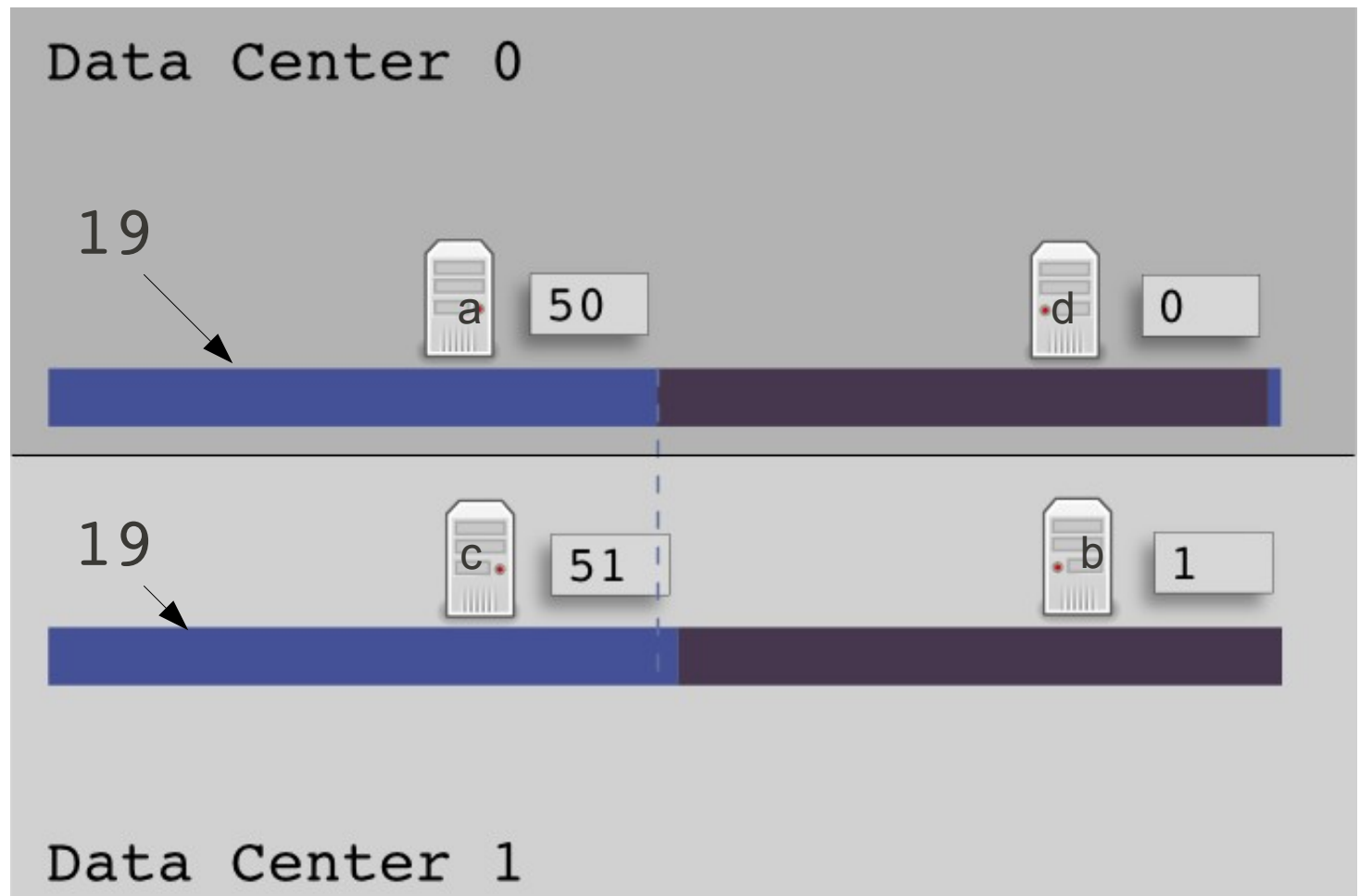
2 Racks per DC

1 Node per rack

RandomPartitioner

NTS

RF=DC0:1,DC1:1



Mirrored Offset Tokens

Token range 0-100

4 Nodes

2 Datacenters

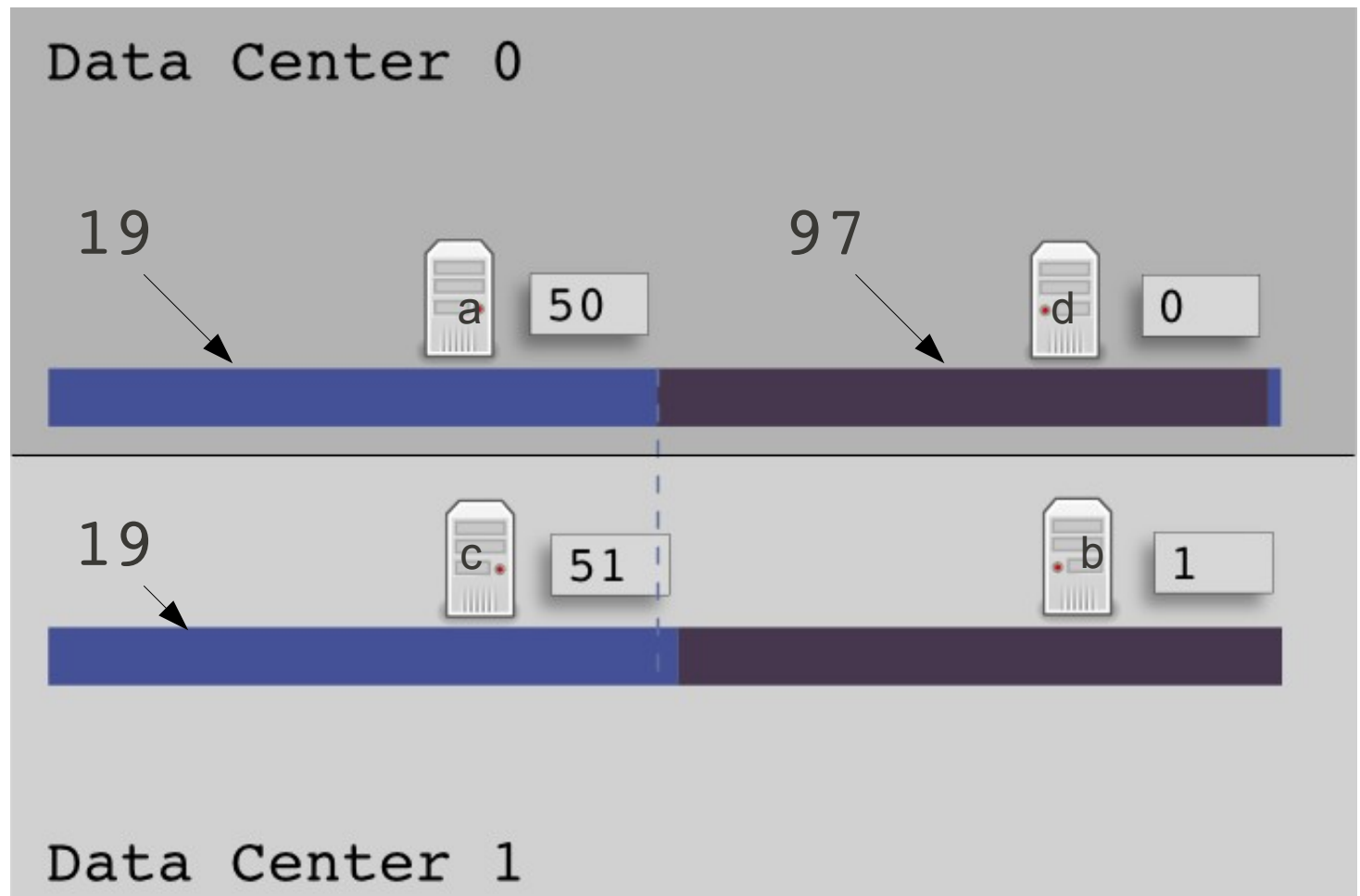
2 Racks per DC

1 Node per rack

RandomPartitioner

NTS

RF=DC0:1,DC1:1



Mirrored Offset Tokens

Token range 0-100

4 Nodes

2 Datacenters

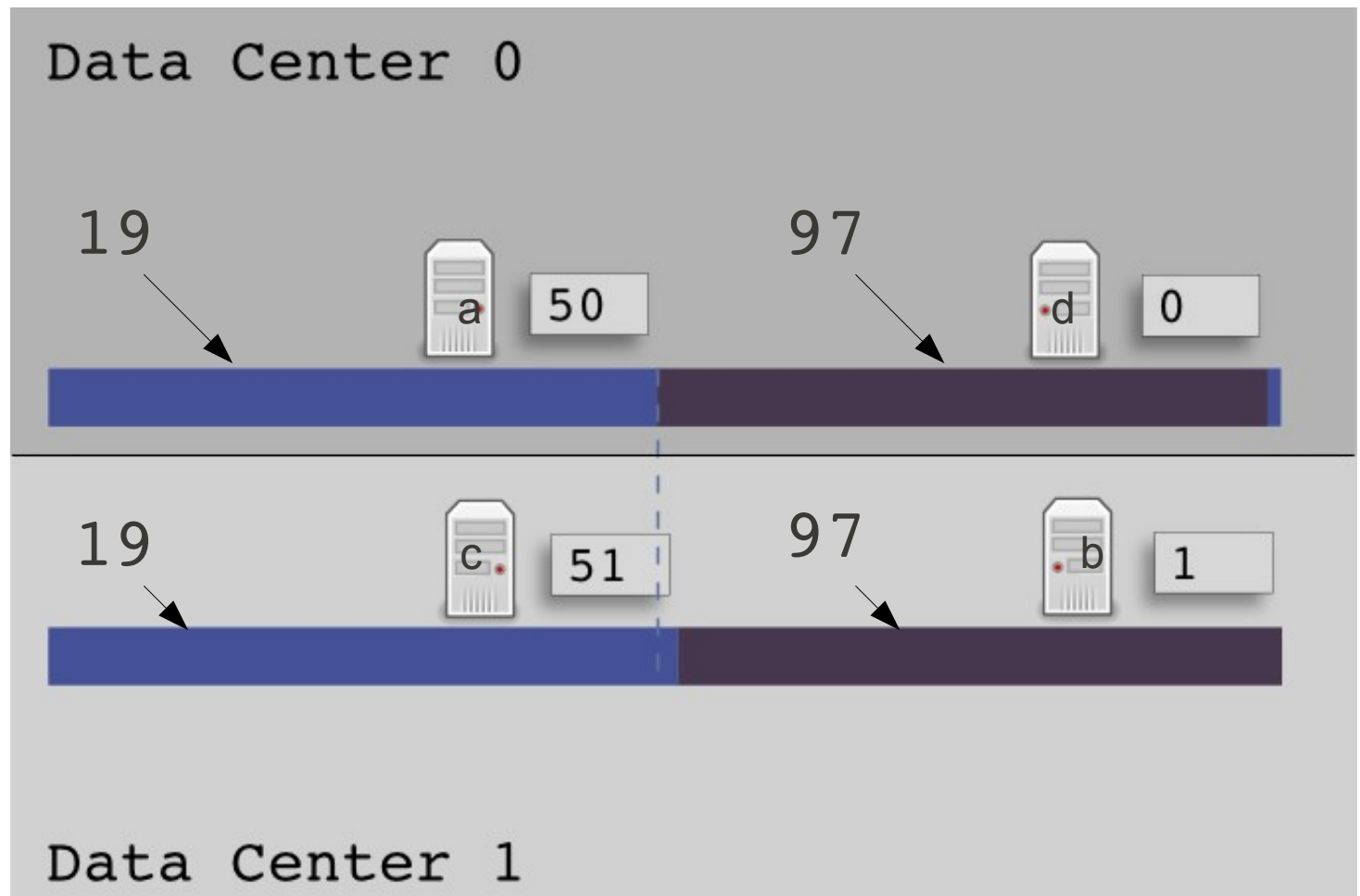
2 Racks per DC

1 Node per rack

RandomPartitioner

NTS

RF=DC0:1,DC1:1



Demo Example

PropertyFileSnitch

cassandra-topology.properties

```
# Cassandra Node IP=Data Center:Rack
```

```
127.0.0.1=DC1:RAC1
```

```
127.0.0.2=DC1:RAC2
```

```
127.0.0.3=DC2:RAC1
```

```
127.0.0.4=DC2:RAC2
```

```
# default for unknown nodes
```

```
default=DC1:RAC1
```



Demo Example

Initial Token Selection

DC1R1: 0

DC1R2: 85070591730234615865843651857942052864

DC2R1: 1

DC2R2: 85070591730234615865843651857942052865

Demo Example

Keyspace and ColumnFamily Creation

```
create keyspace SipTrace
with strategy_options = [{DC1:1,DC2:1}]
and
placement_strategy='org.apache.cassandra.locator.NetworkTopologyStrategy';

use SipTrace;

create column family SIP with column_type='Standard';

assume SIP keys as ascii;
assume SIP validator as ascii;
exit;
```