MySQL Cluster

Monty Taylor
MySQL Inc

University of São Paulo
February 2007

Who am I?

- Monty Taylor
- Senior Consultant
- Working for MySQL Inc.
- Living in Seattle, Washington
- Theatre Director, Lighting Designer and Photographer

Quick Introduction to MySQL

SQL Relational Database Management System

Aiming for full standards compliance
(SQL 2003)

Quick Introduction to MySQL

Reputation for:

- ✓Speed
- ✓Reliability
- ✓Ease of Use

Cluster appeared in 4.1

Quick Introduction to MySQL

5.0 is GA

Recommended release for Production

Many speed improvements for Cluster

Quick Introduction to MySQL

5.1 is Beta
Many features for Cluster
(also speed improvements)

What to Expect Today

What to expect today

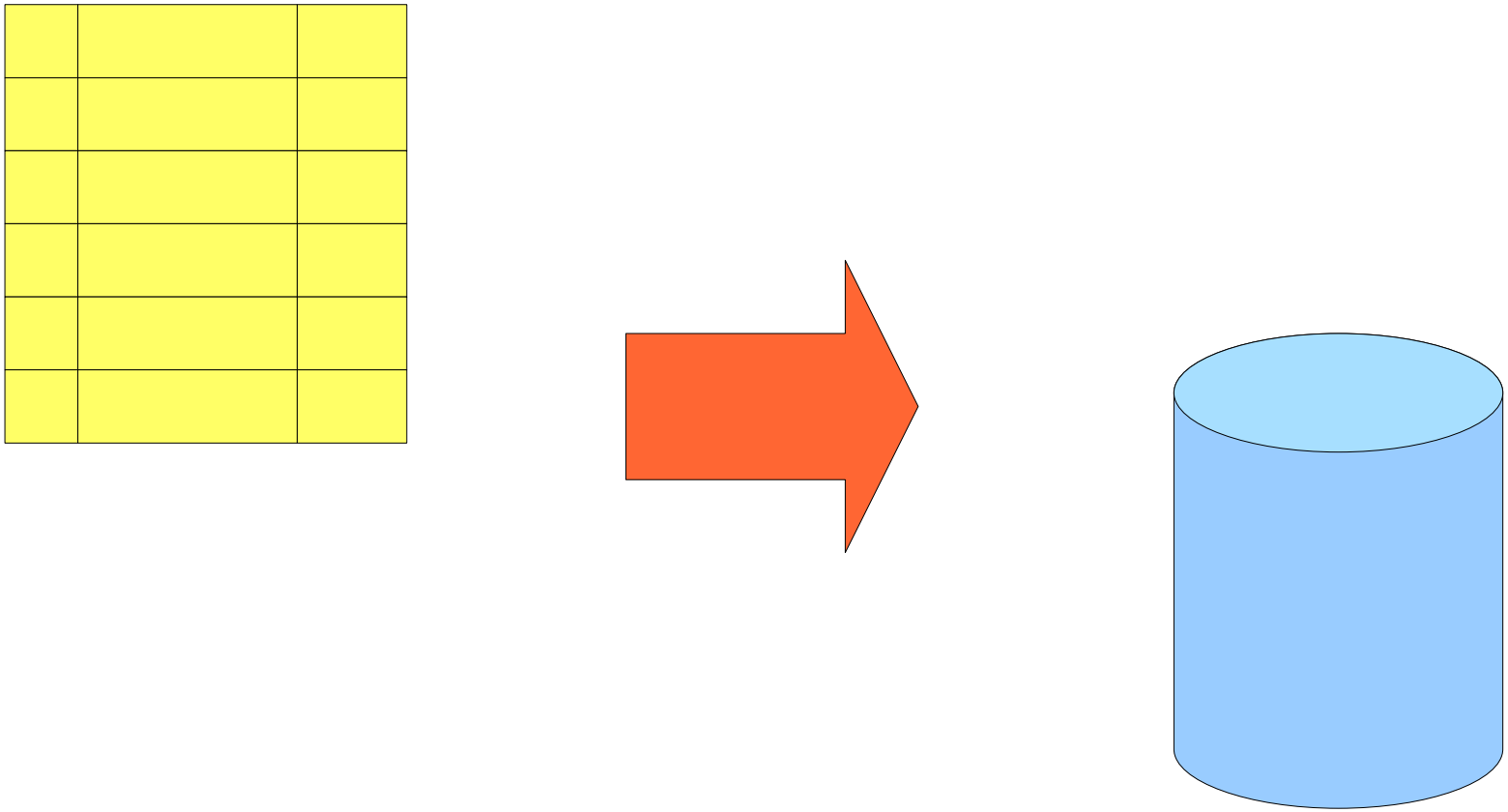
- Learn about the architecture of MySQL Cluster
- Current and future features
- What problems MySQL Cluster is currently good at solving
  - and those that it isn't
- Some discussion of internals

Storage Engines

Storage Engines

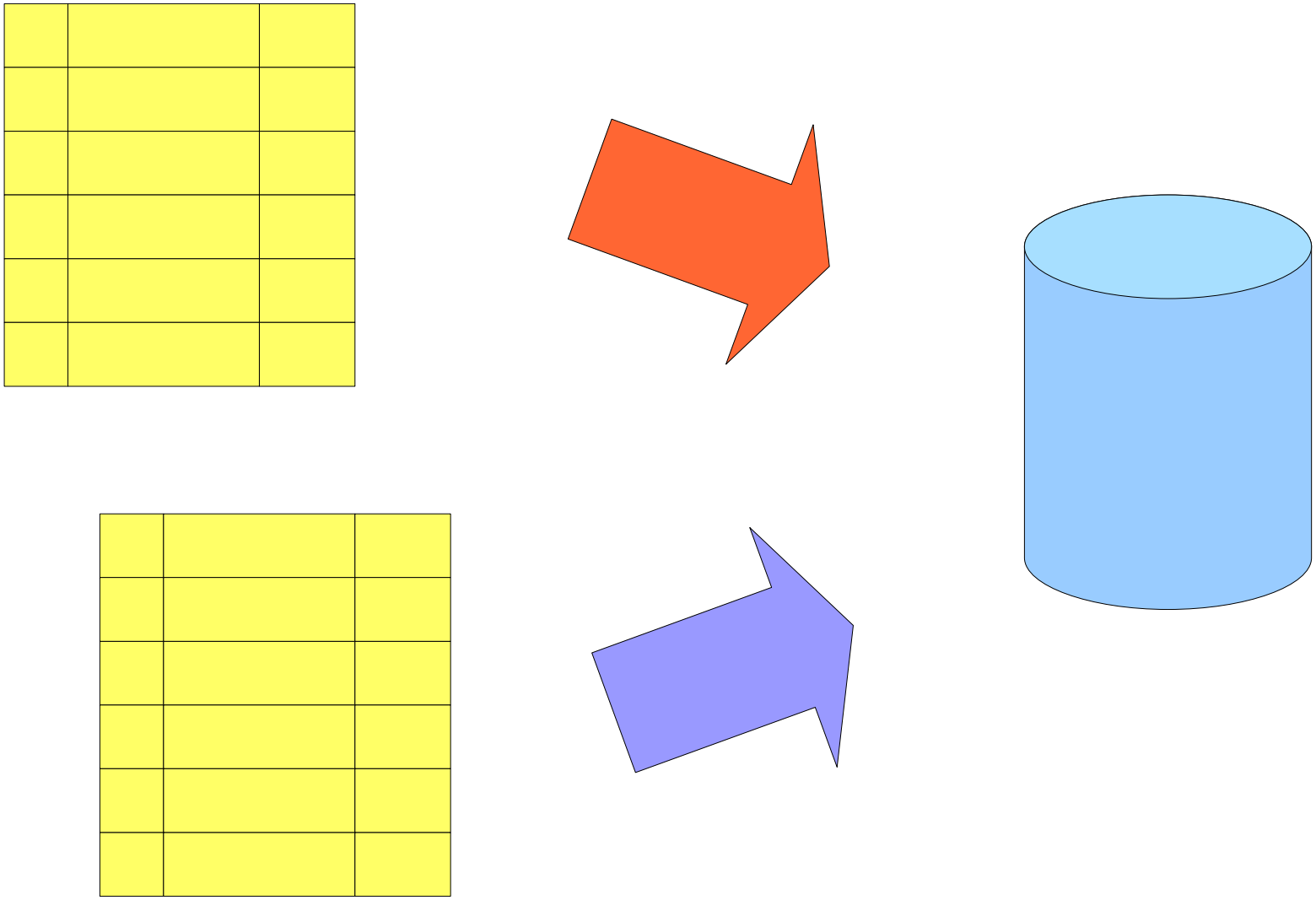
Unique feature of MySQL

No one best way to store tables

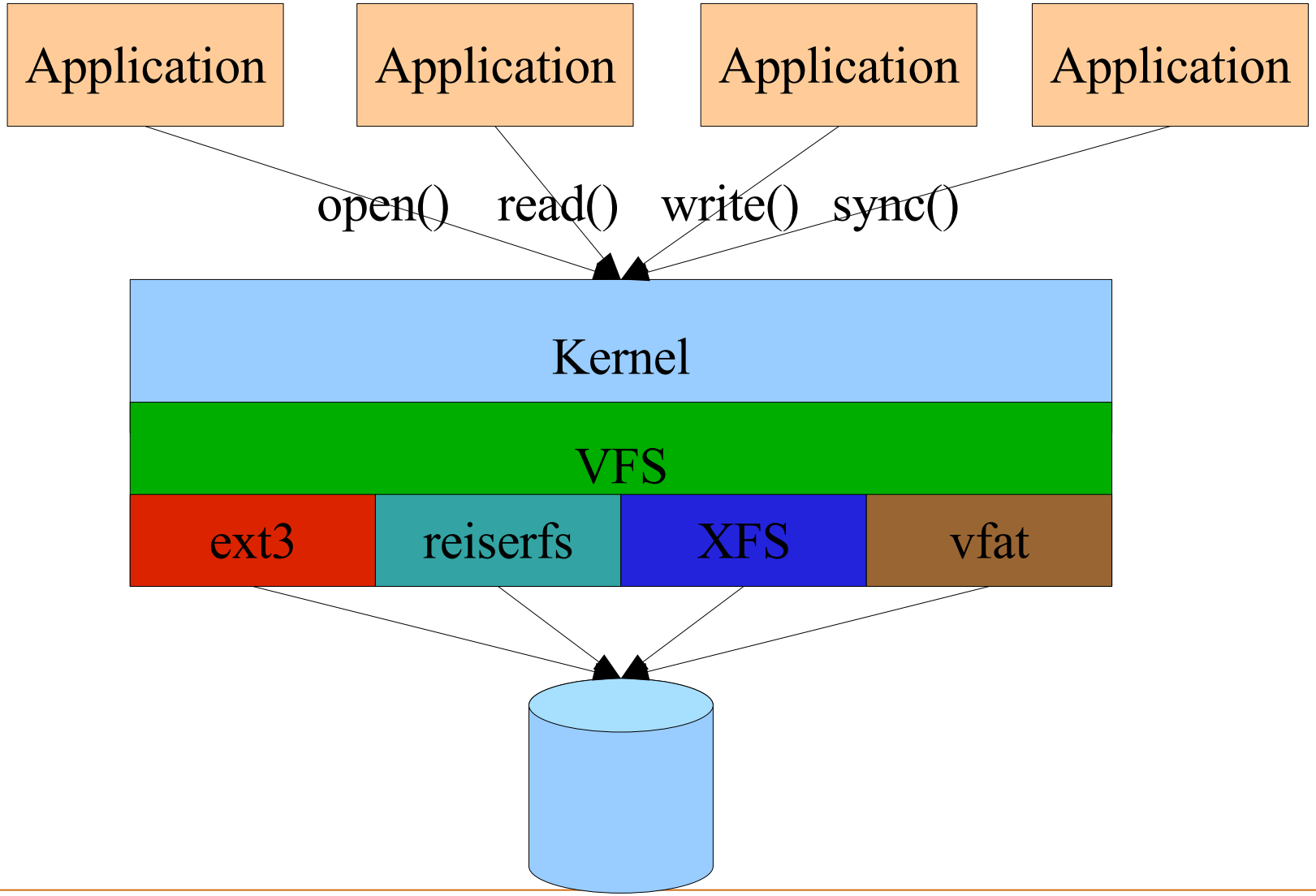


Choice of Storage Engines

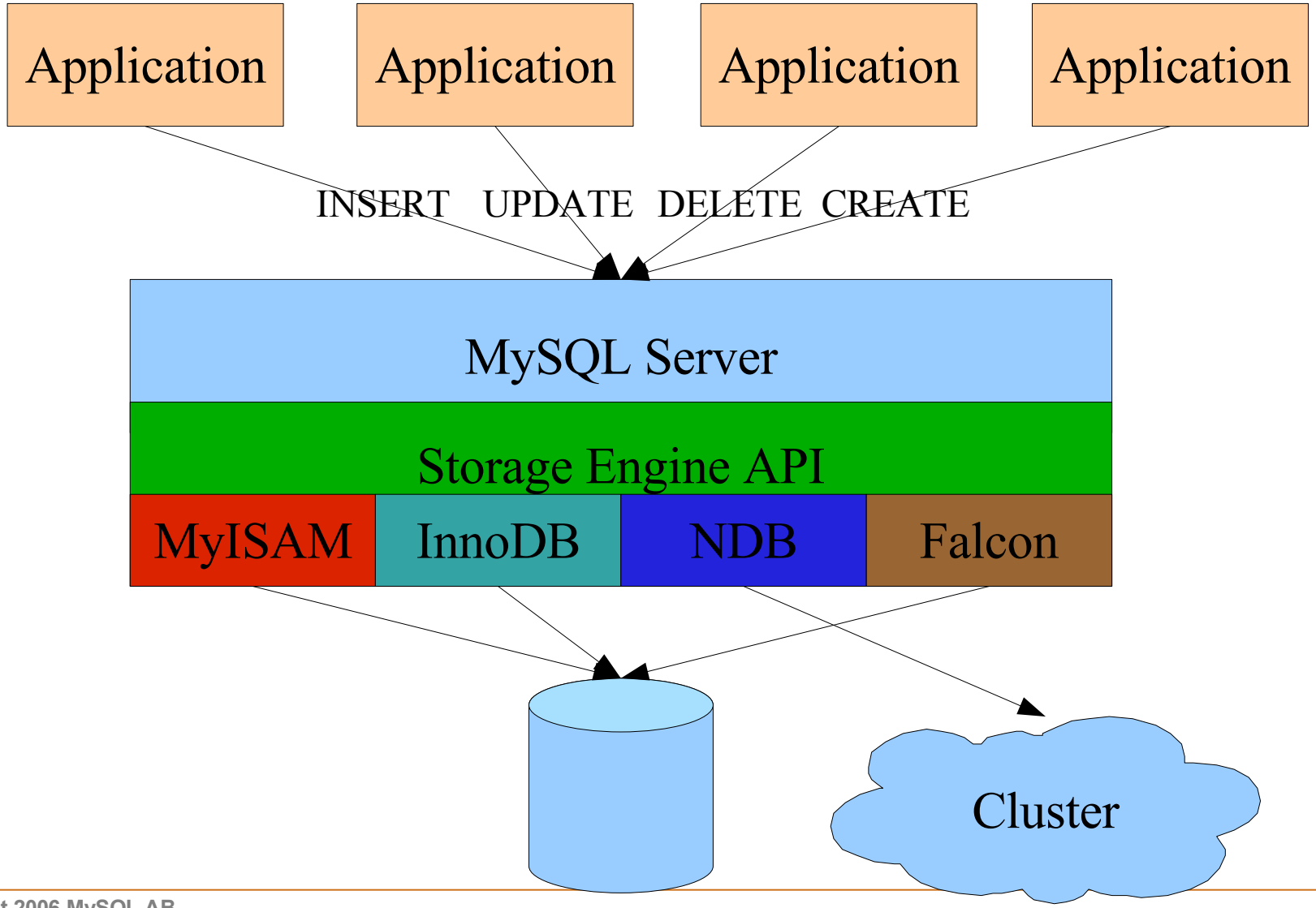
Different engine per table (if you want)



Just like a Virtual File System Layer



Just like a Virtual File System Layer



What is MySQL Cluster?

What is MySQL Cluster?

A High Availability

What is MySQL Cluster?

A High Availability
High Performance

What is MySQL Cluster?

A High Availability
High Performance
In Memory (4.1 and 5.0)

What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

Clustered

What is MySQL Cluster?

A High Availability
High Performance
In Memory (4.1 and 5.0)
Shared Nothing
Clustered
Storage Engine

What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

Clustered

Storage Engine

What is MySQL Cluster?

High Availability

High Availability

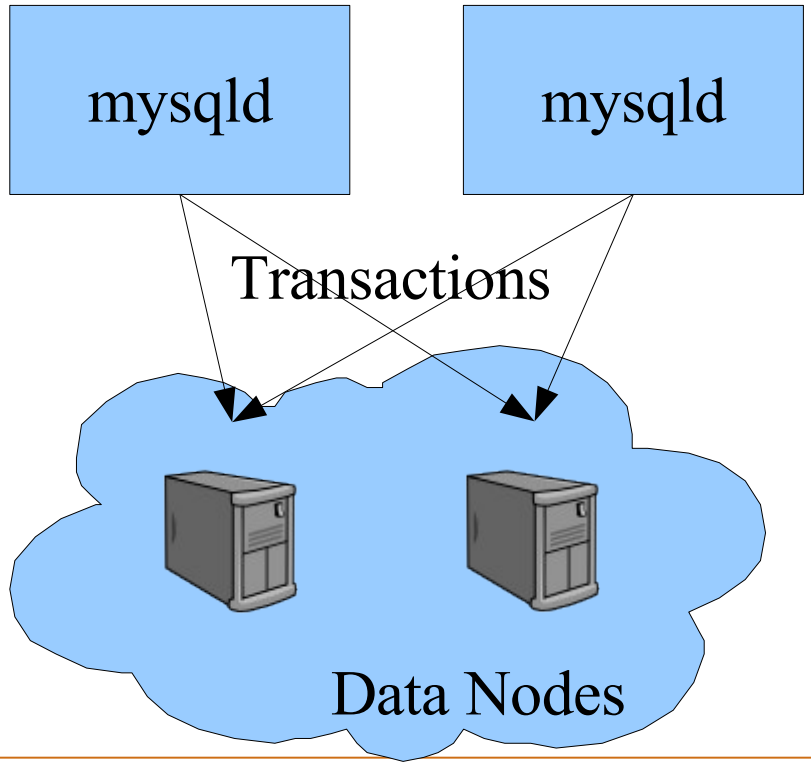
Designed for Five Nines (99.999%) Uptime

High Availability

Sub-Second Failover

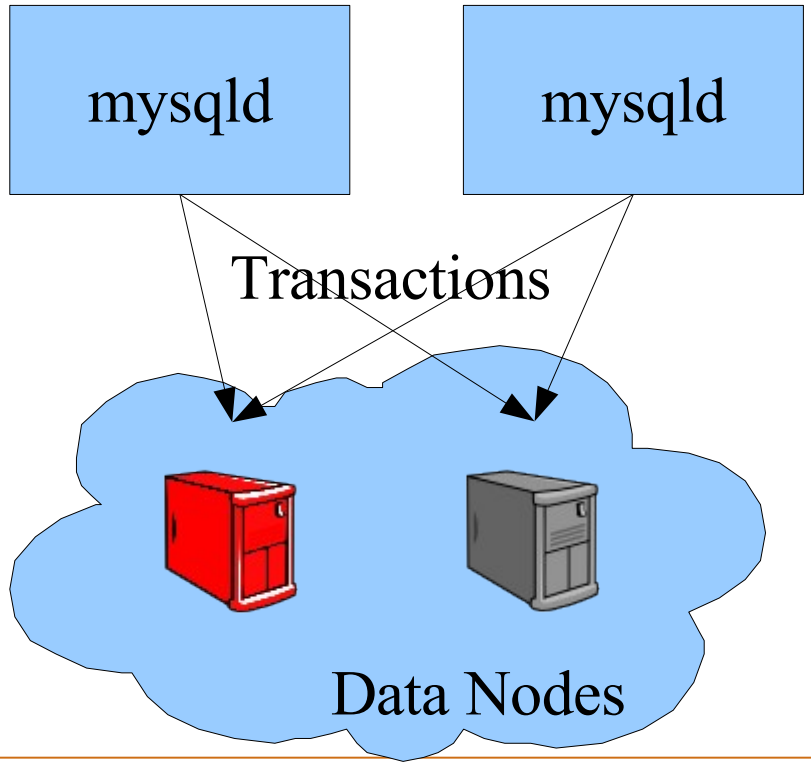
High Availability

Sub-Second Failover



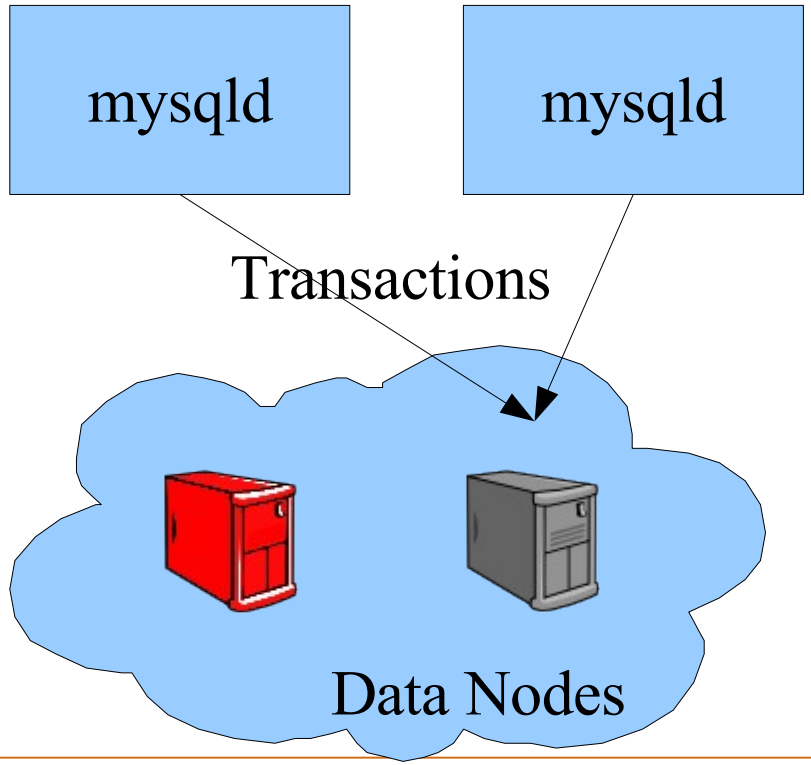
High Availability

Sub-Second Failover



High Availability

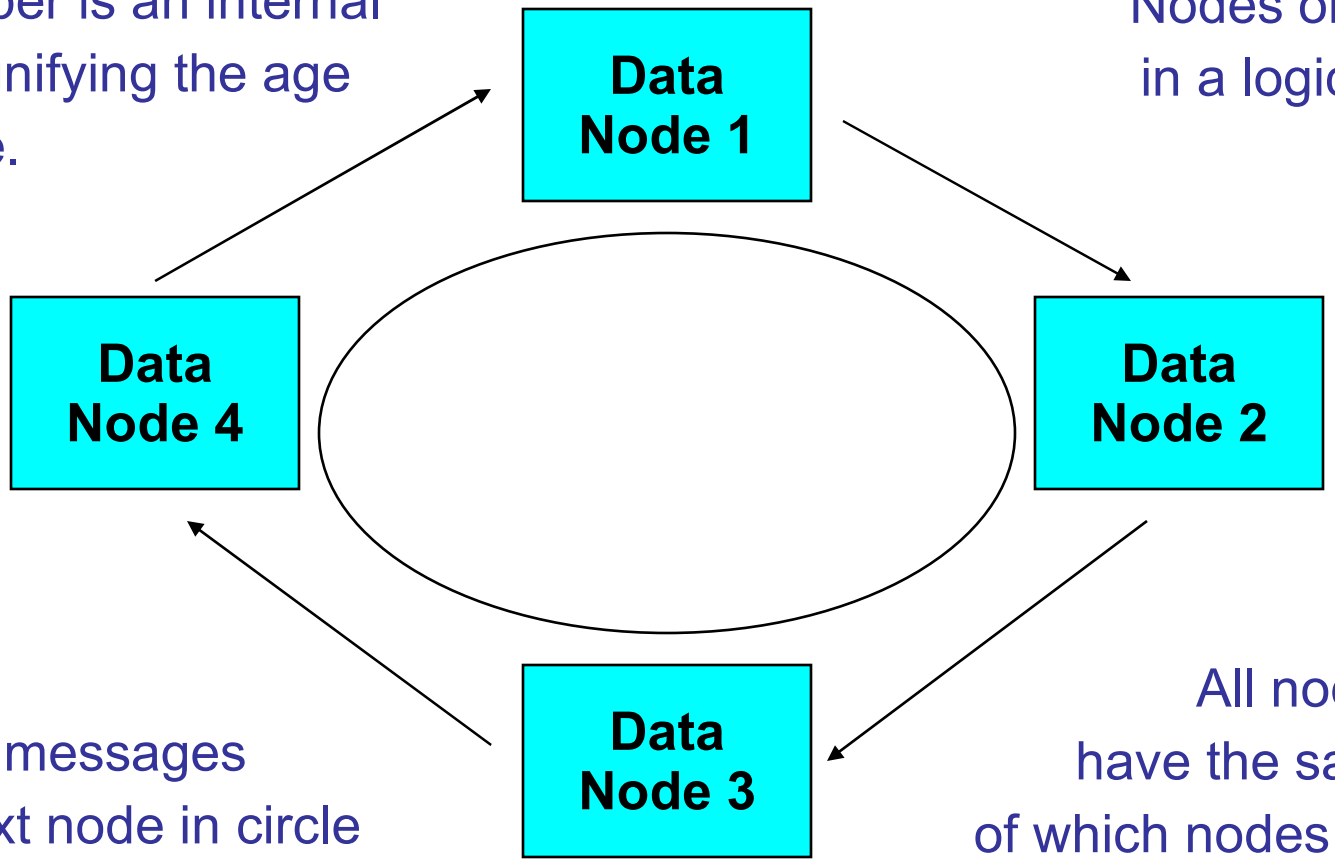
Sub-Second Failover



Failure Detection: Heartbeats, lost connections

Node number is an internal number signifying the age of the node.

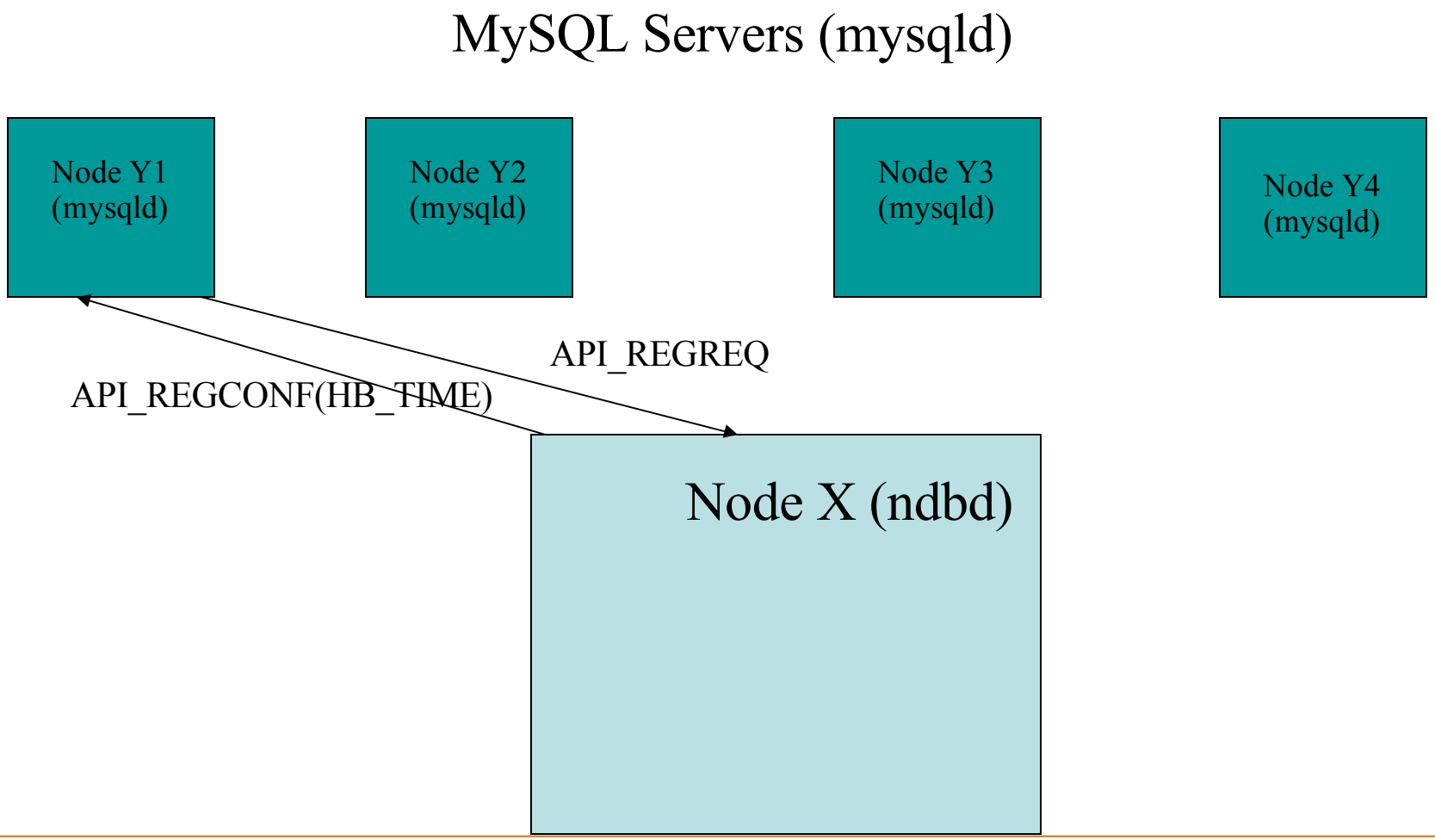
Nodes organized in a logical circle



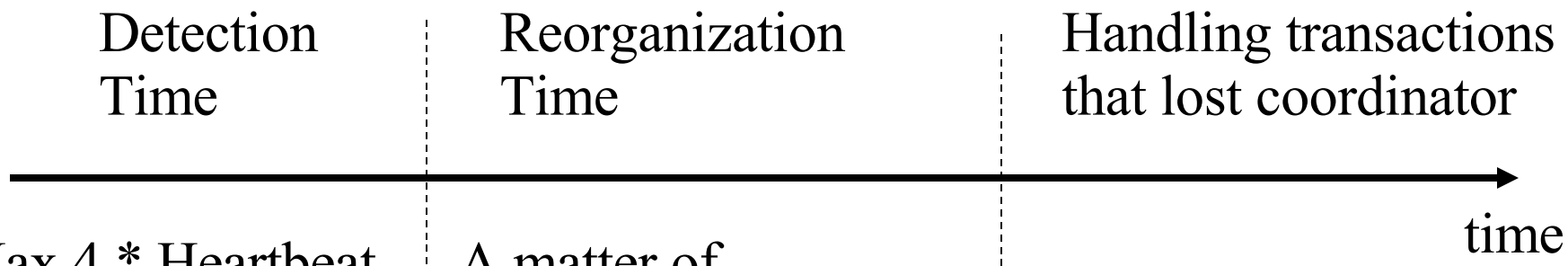
Heartbeat messages sent to next node in circle

All nodes must have the same view of which nodes are alive

Heartbeat Handling of MySQL Servers



Time to Handle Node Failure



Max 4 * Heartbeat Timer (default 1.5 Seconds)
Almost instanteous if node failure discovered by OS.

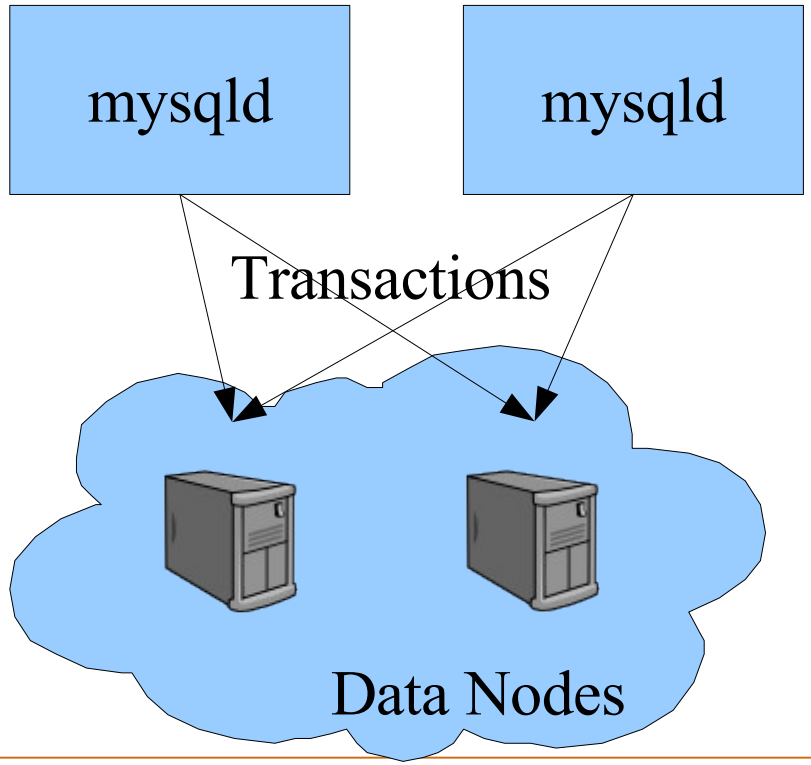
A matter of microseconds sending two rounds of broadcast messages

High Availability

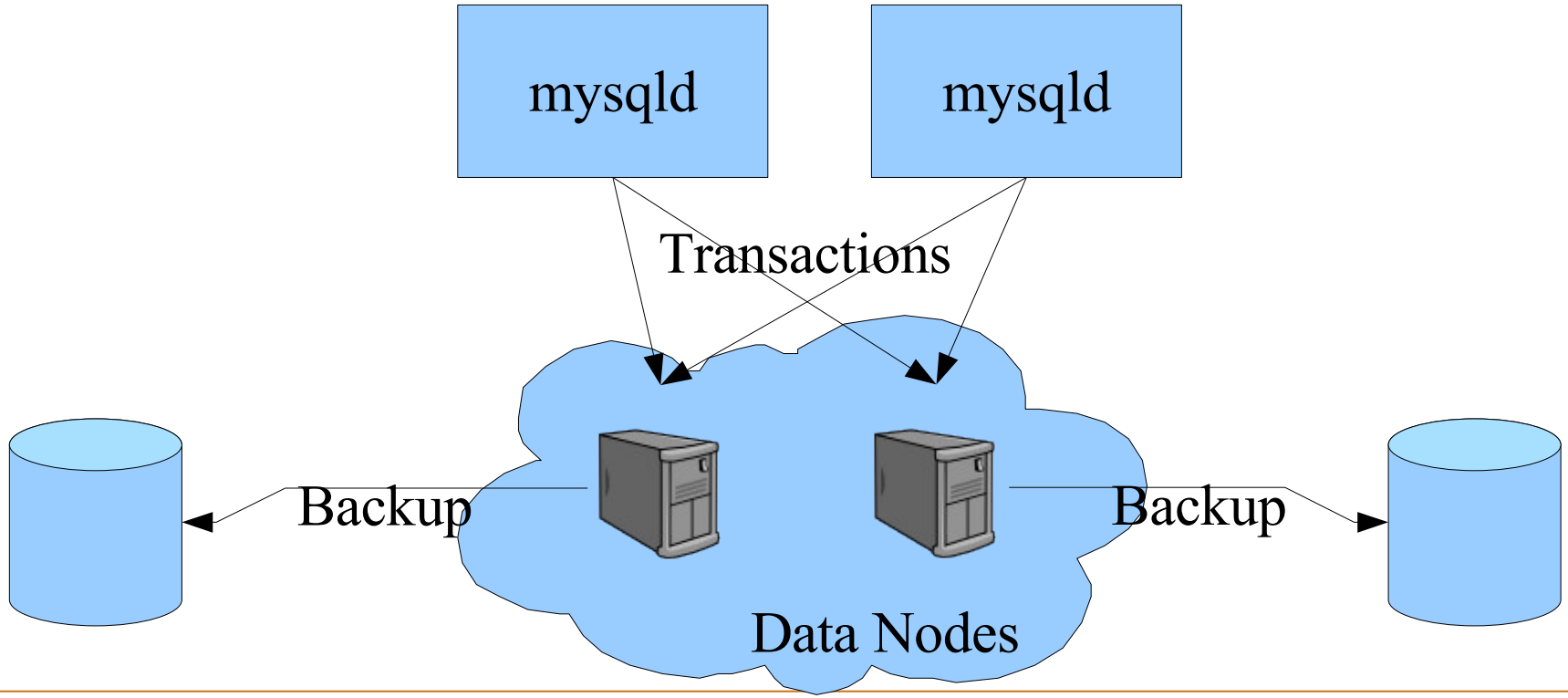
"Hot" (Online) Backup

High Availability

Hot (Online) Backup



Hot (Online) Backup



High Availability

Configurable Amount of Redundancy

High Availability

Configurable Amount of Redundancy

NoOfReplicas

High Availability

NoOfReplicas=1




D a t a

High Availability

NoOfReplicas=2




Da ta Da ta

High Availability

NoOfReplicas=3

Da Da ta Da ta ta

High Availability

NoOfReplicas=4




Data Data Data Data

What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

Clustered

Storage Engine

High Performance

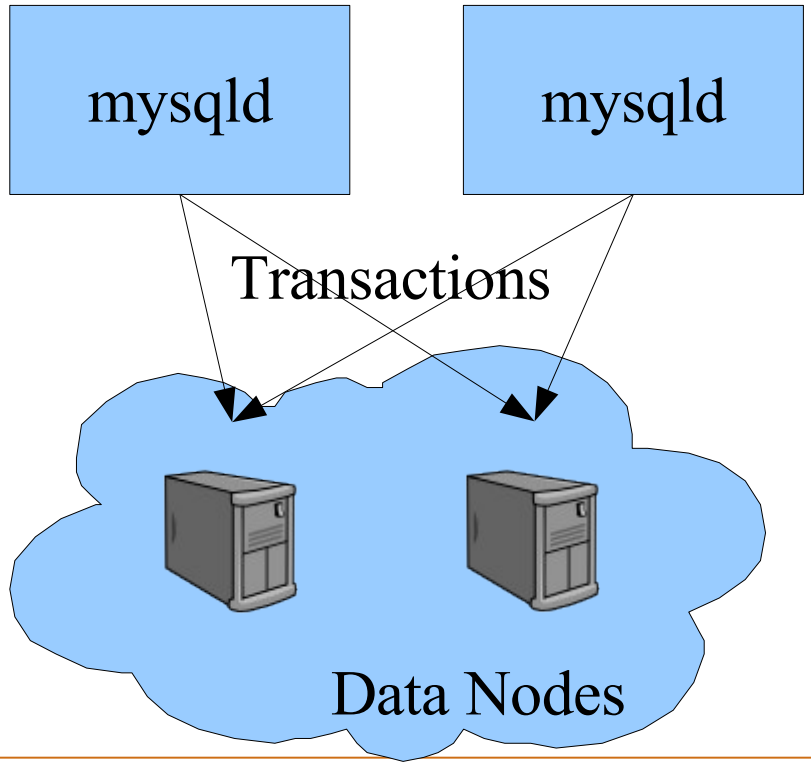
High Performance

Not BEGIN to COMMIT

Not BEGIN to COMMIT Through Parallelism

High Performance

Parallelism



What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

Clustered

Storage Engine

What is MySQL Cluster?

In Memory (4.1 and 5.0)

In Memory (4.1 and 5.0)

Data and Indexes kept in main memory

$$\text{Per Node} = \frac{\text{Size of Database} \times \text{No Of Replicas} \times 1.1}{\text{No Of Data Nodes}}$$

In Memory (4.1 and 5.0)

Check point to disk

In Memory (4.1 and 5.0)

Check point to disk

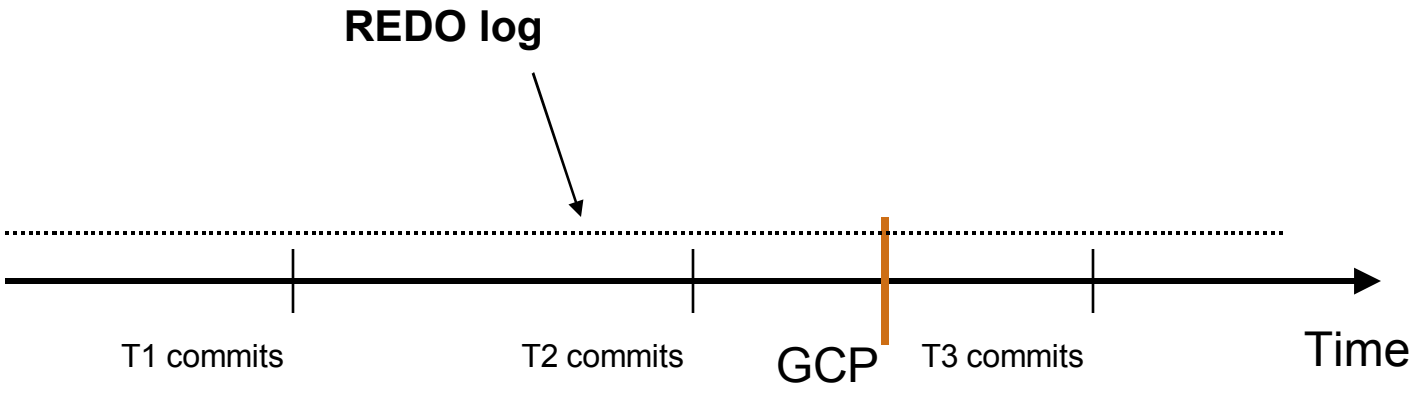
Frequent, Configurable

In Memory (4.1 and 5.0)

Check point to disk

Not complete data loss after power outage

Flushing the log to disk

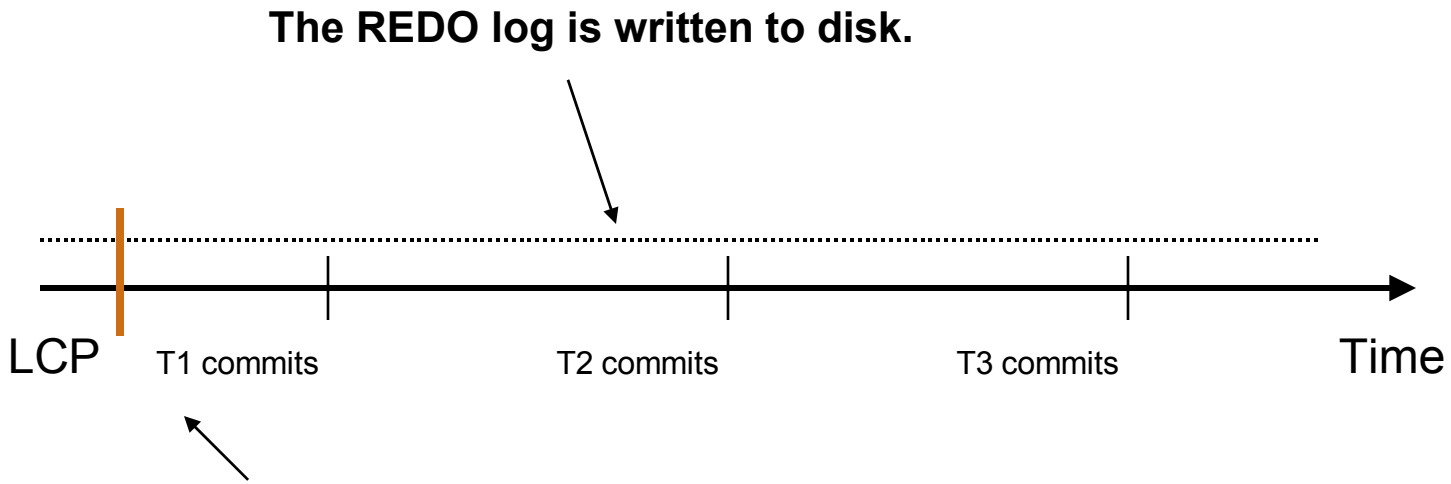


A *global checkpoint* GCP flushes REDO log to disk.

Transactions T1 and T2 become *disk persistent*.

(Also called *group commit*.)

Log and Local Checkpoints

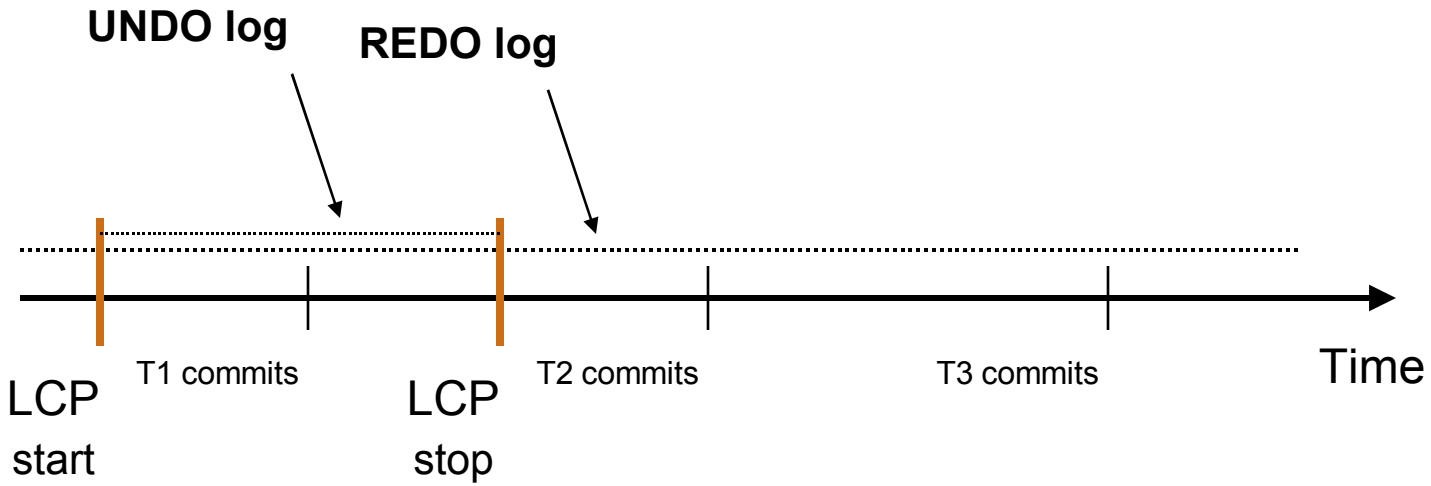


A *local checkpoint* LCP saves an image of the database on disk.
This enables cutting the tail of the REDO log.

Since MySQL Cluster replicate all data, the REDO log and LCPs are only used to recover from whole system failures.

Transactions T1, T2 and T3 are *main memory persistent*.

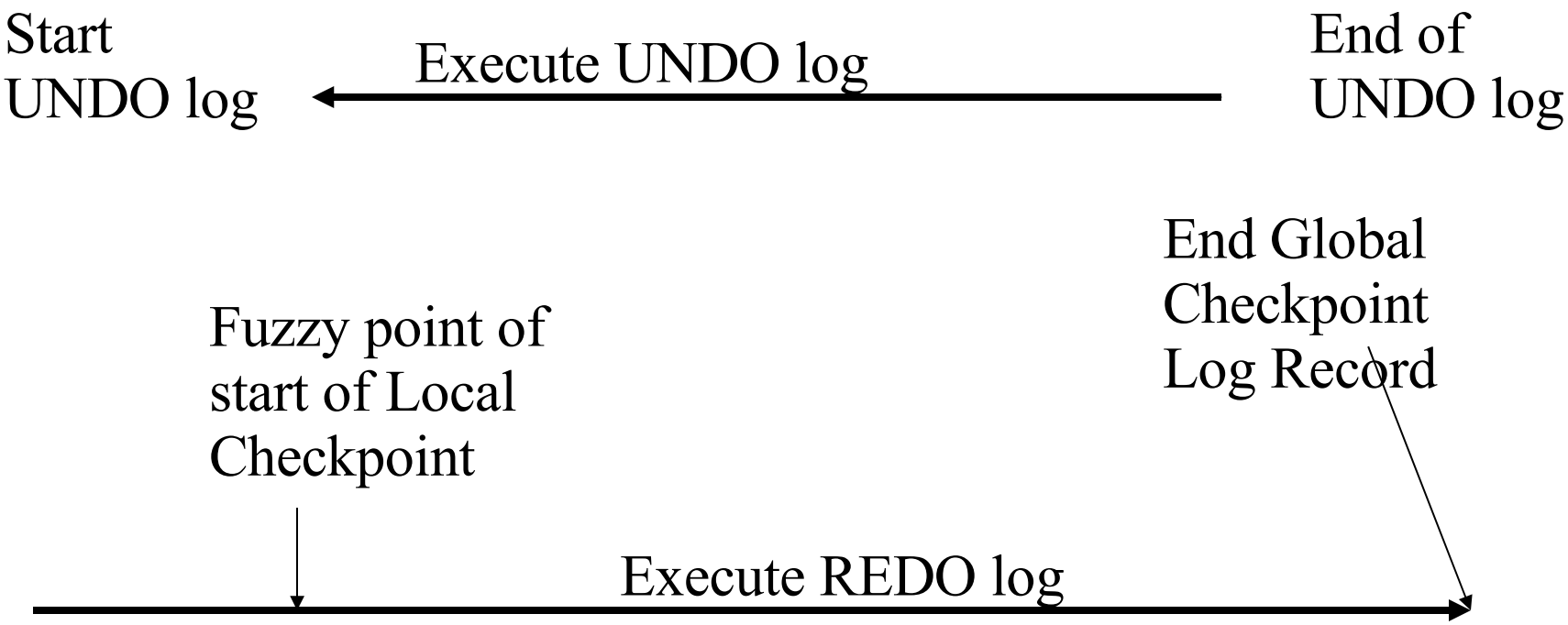
Checkpointing takes time



UNDO log makes LCP image consistent with database at LCP start time.

Restoring a Fragment at System Restart

1) Load Data Pages from local checkpoint into memory



Data on Disk in 5.1

Data on Disk in 5.1

Indexed fields and Indexes in main memory

What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

Clustered

Storage Engine

What is MySQL Cluster?

Shared Nothing

Shared Nothing

Commodity PCs

Shared Nothing

Commodity Interconnects

Shared Nothing

Commodity Interconnects

Ethernet

Shared Nothing

Commodity Interconnects

Ethernet

SCI

Shared Nothing

No Expensive Shared Disk

Shared Nothing

No Expensive Shared Disk

(so no single point of failure there)

What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

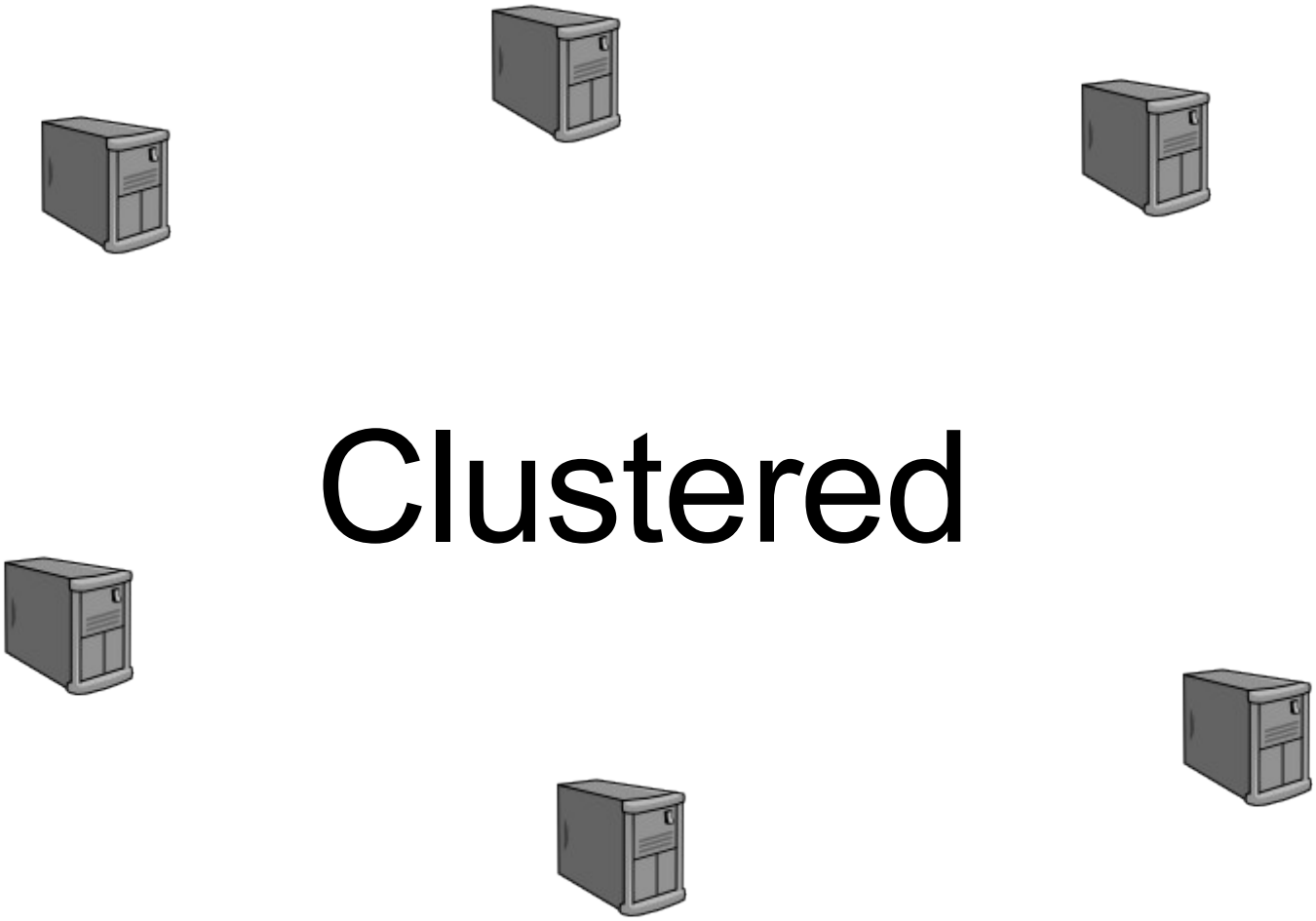
Clustered

Storage Engine

What is MySQL Cluster?

Clustered

What is MySQL Cluster?



What is MySQL Cluster?

A High Availability

High Performance

In Memory (4.1 and 5.0)

Shared Nothing

Clustered

Storage Engine

What is MySQL Cluster?

Storage Engine

What is MySQL Cluster?

ENGINE=NDBCLUSTER

What is MySQL Cluster?

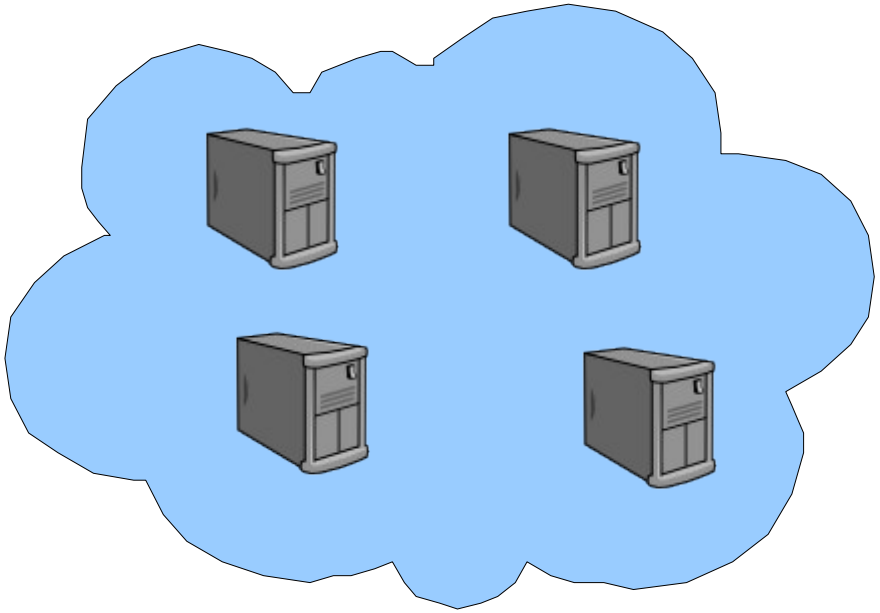
A High Availability
High Performance
In Memory (4.1 and 5.0)
Shared Nothing
Clustered
Storage Engine

Node Types

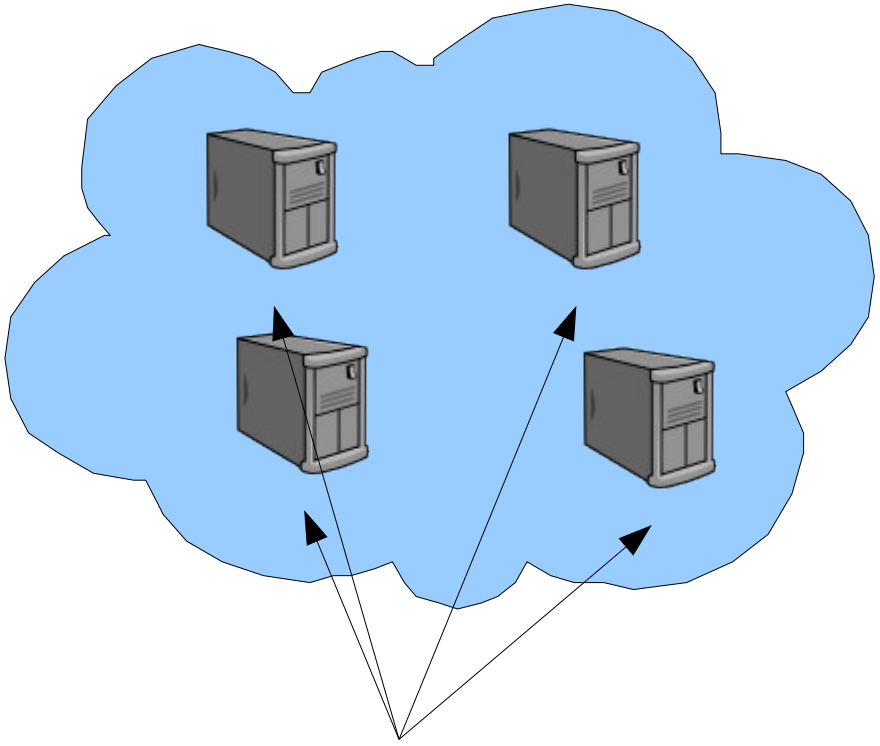
Node Types

Data Nodes (ndbd)

Data Nodes



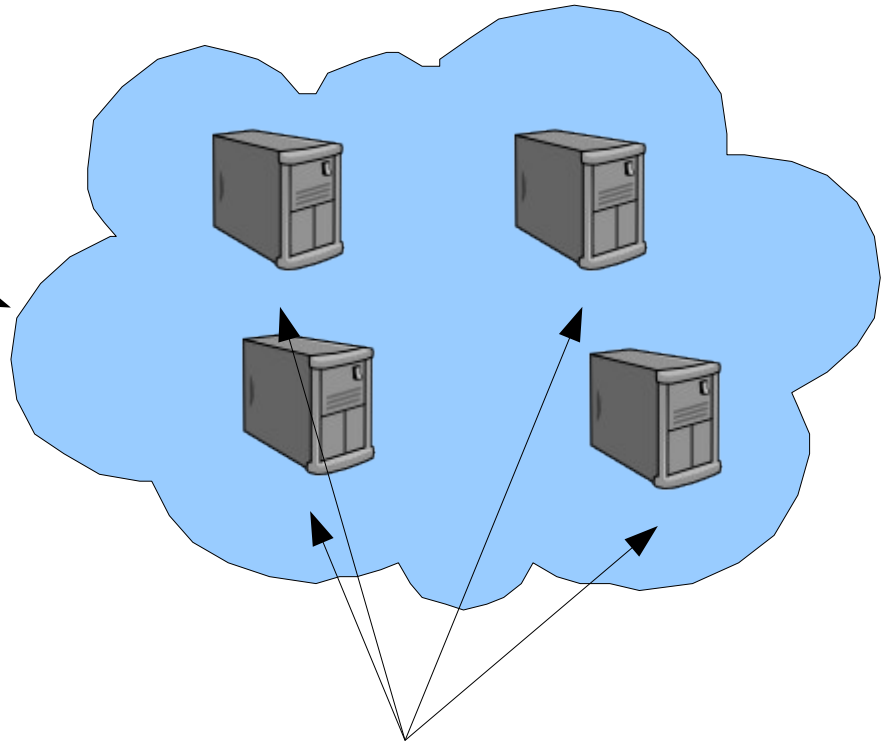
Data Nodes



Data nodes (running ndbd)

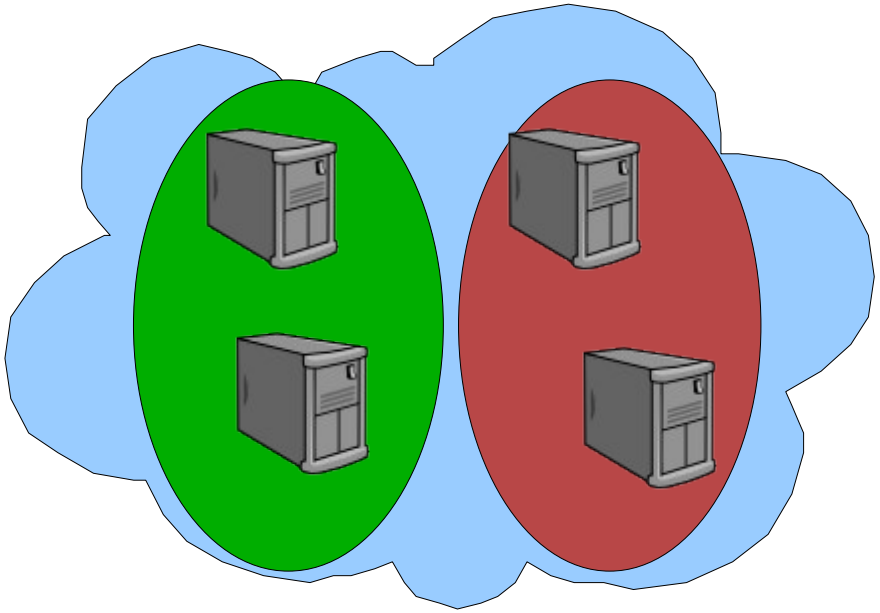
Data Nodes

This cloud means a cluster

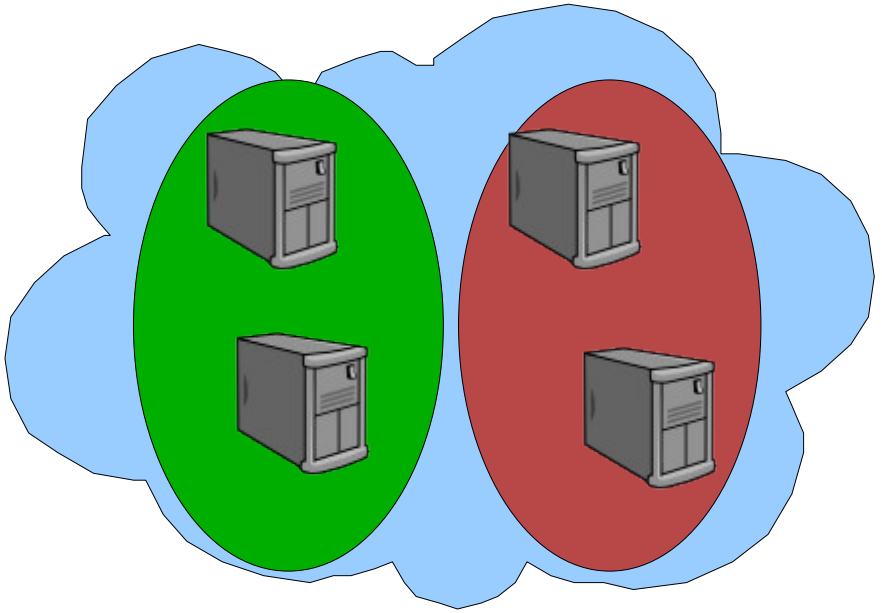


Data nodes (running ndbd)

Data Nodes grouped



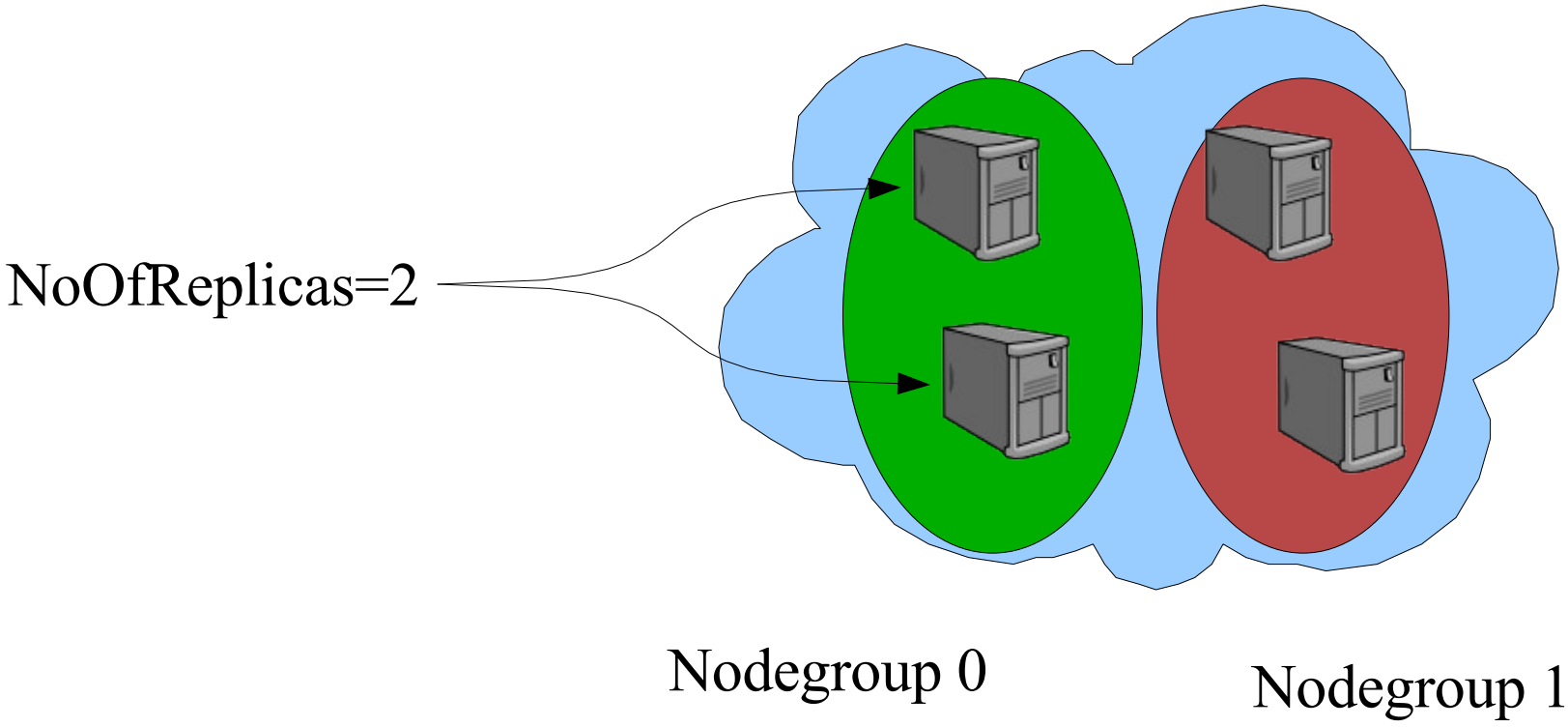
Data Nodes grouped into nodegroups



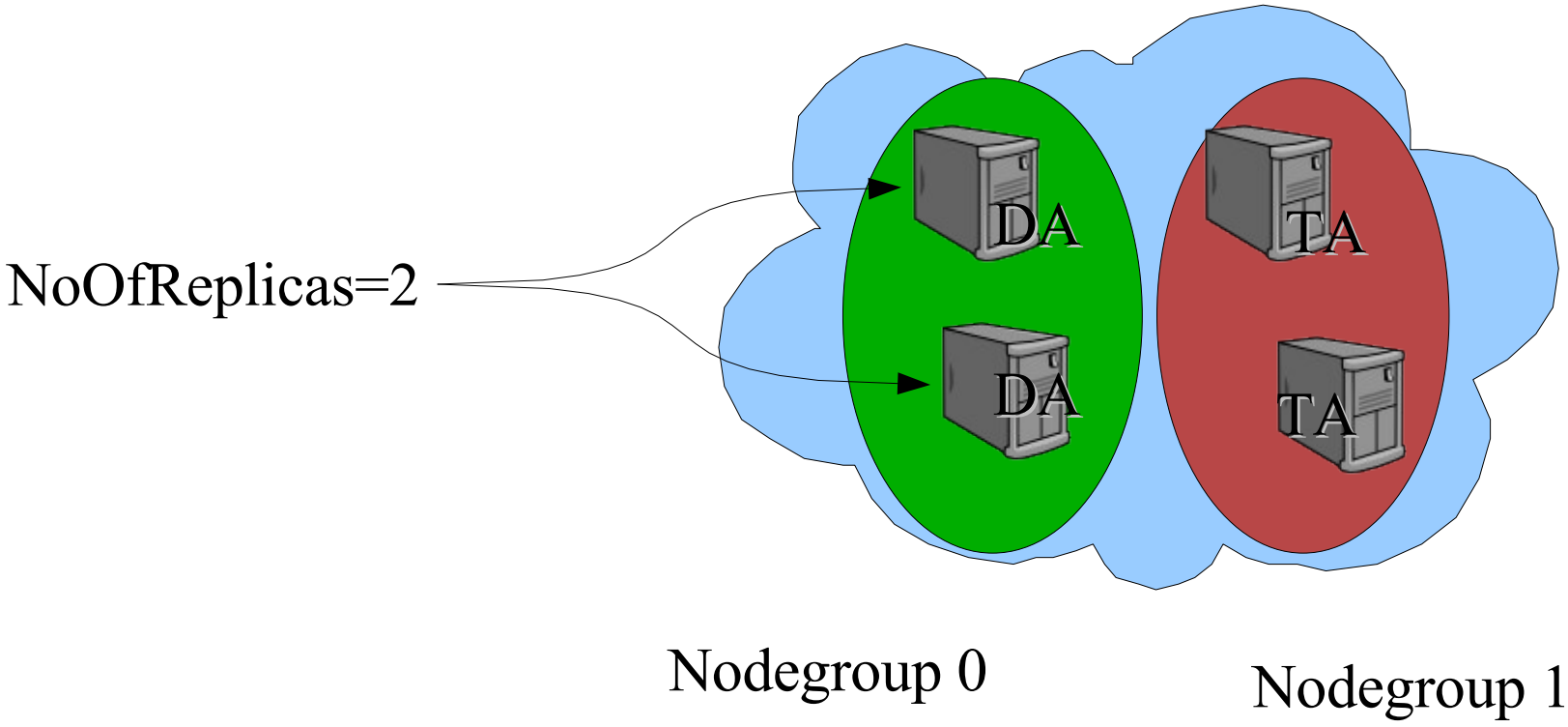
Nodegroup 0

Nodegroup 1

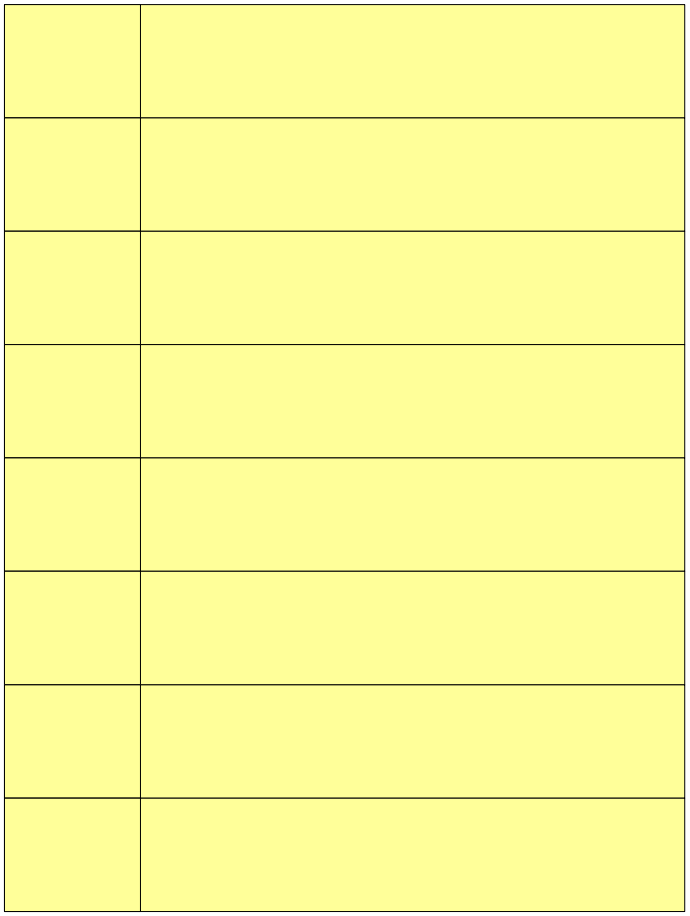
NoOfReplicas is number of nodes in node group

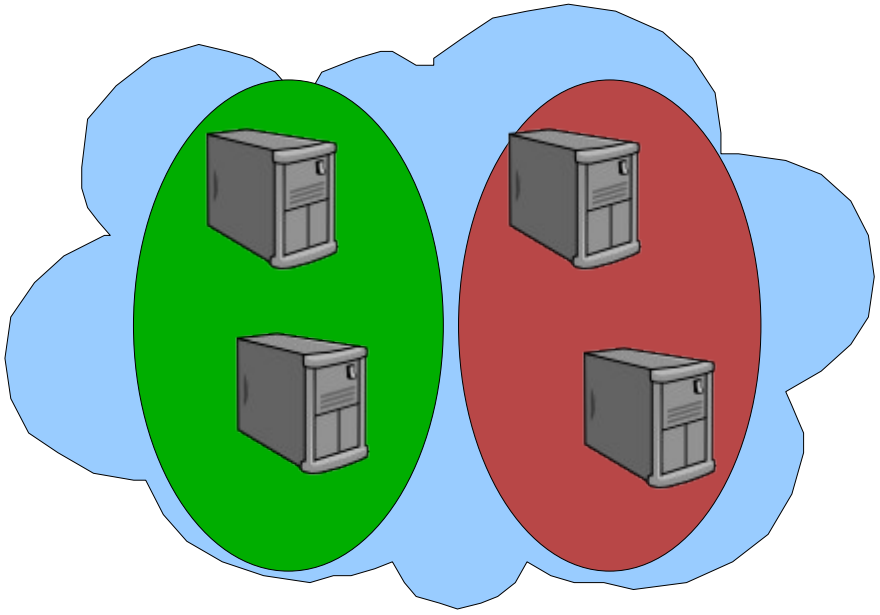


NoOfReplicas is number of nodes in node group



Data Nodes



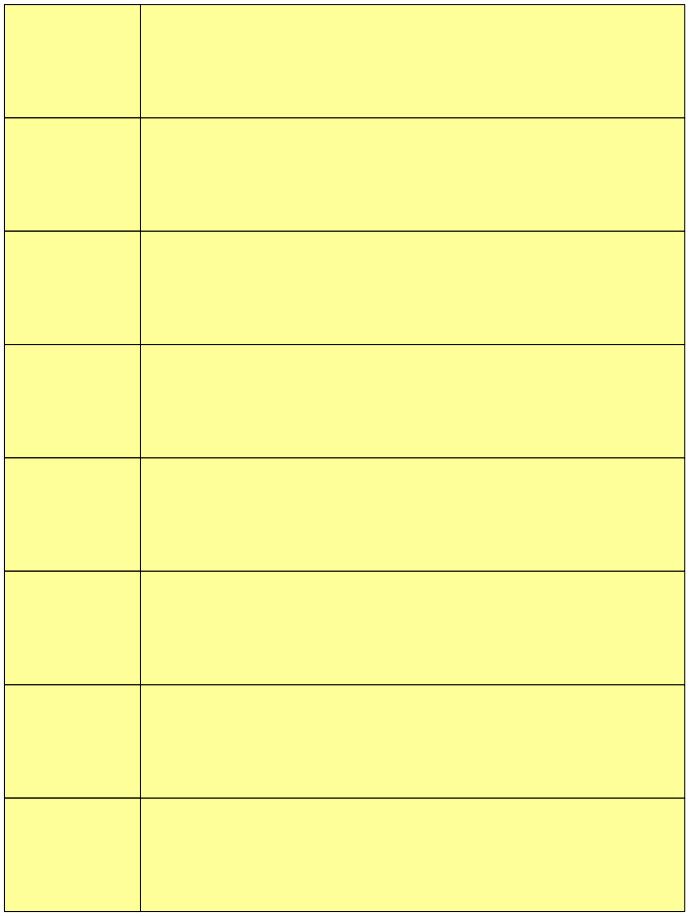


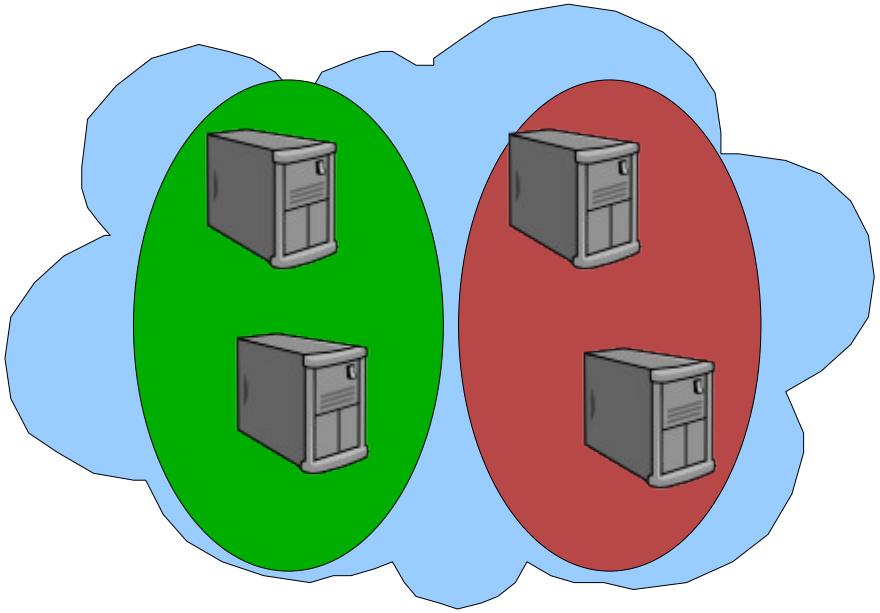
Nodegroup 0

Nodegroup 1

Data Nodes

pk



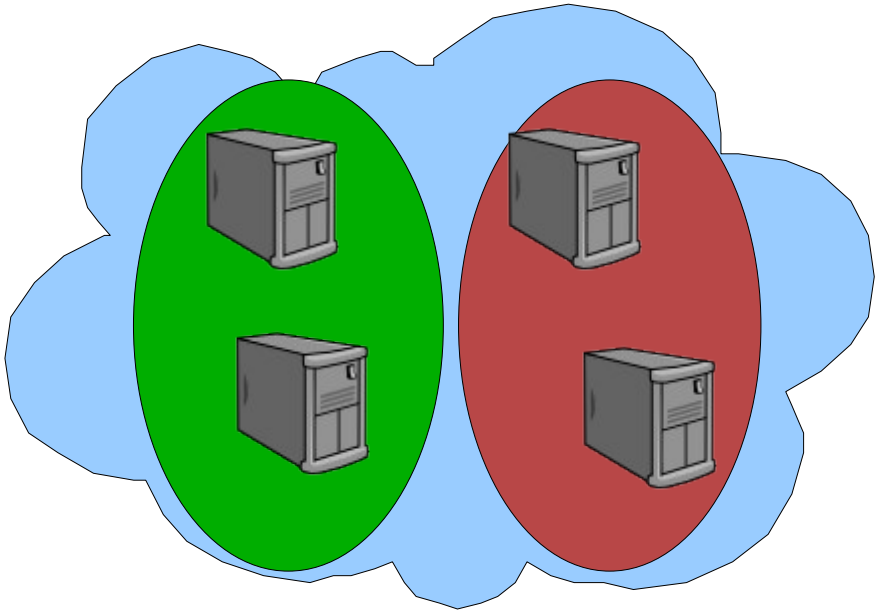


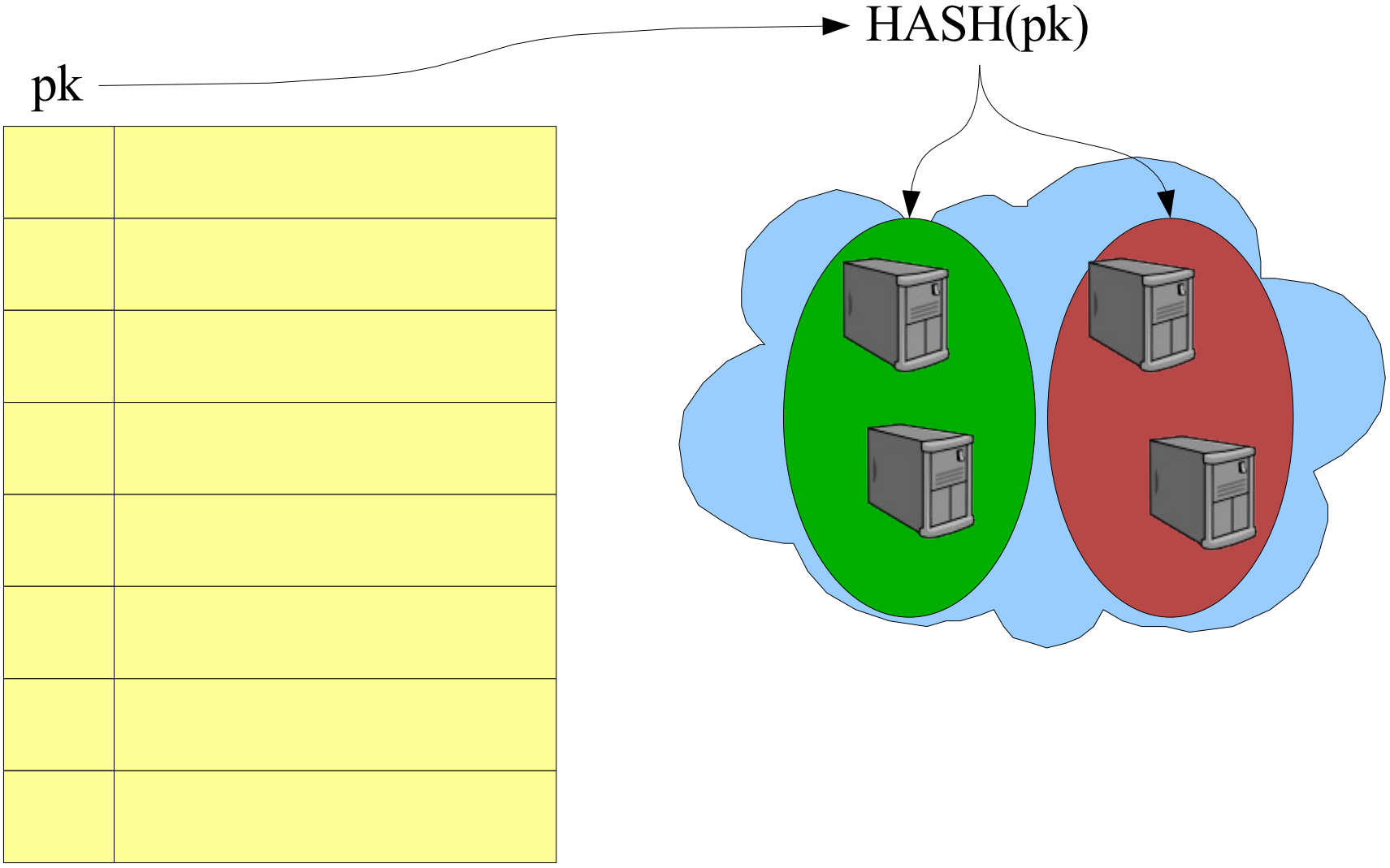
Nodegroup 0

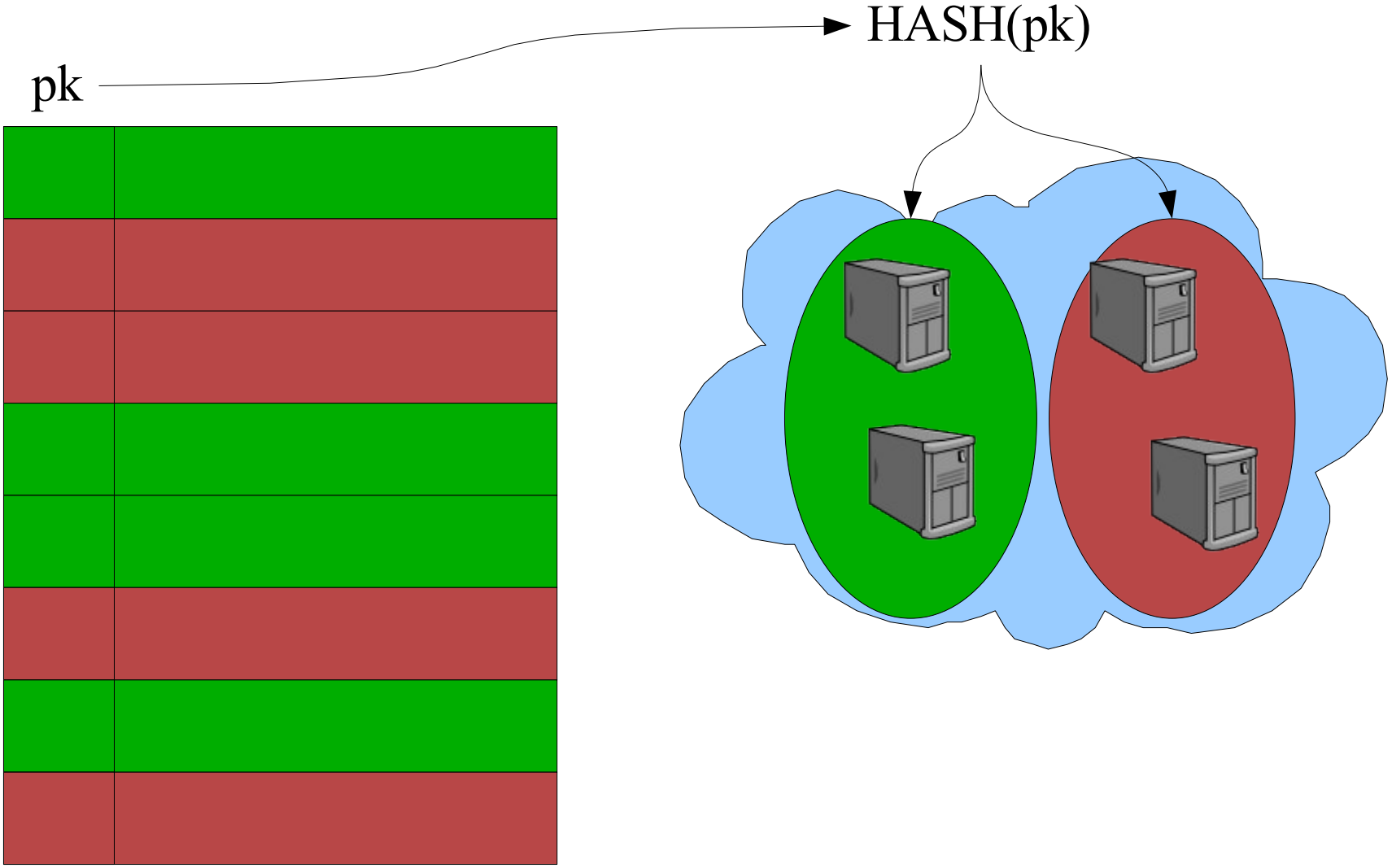
Nodegroup 1

HASH(pk)

pk

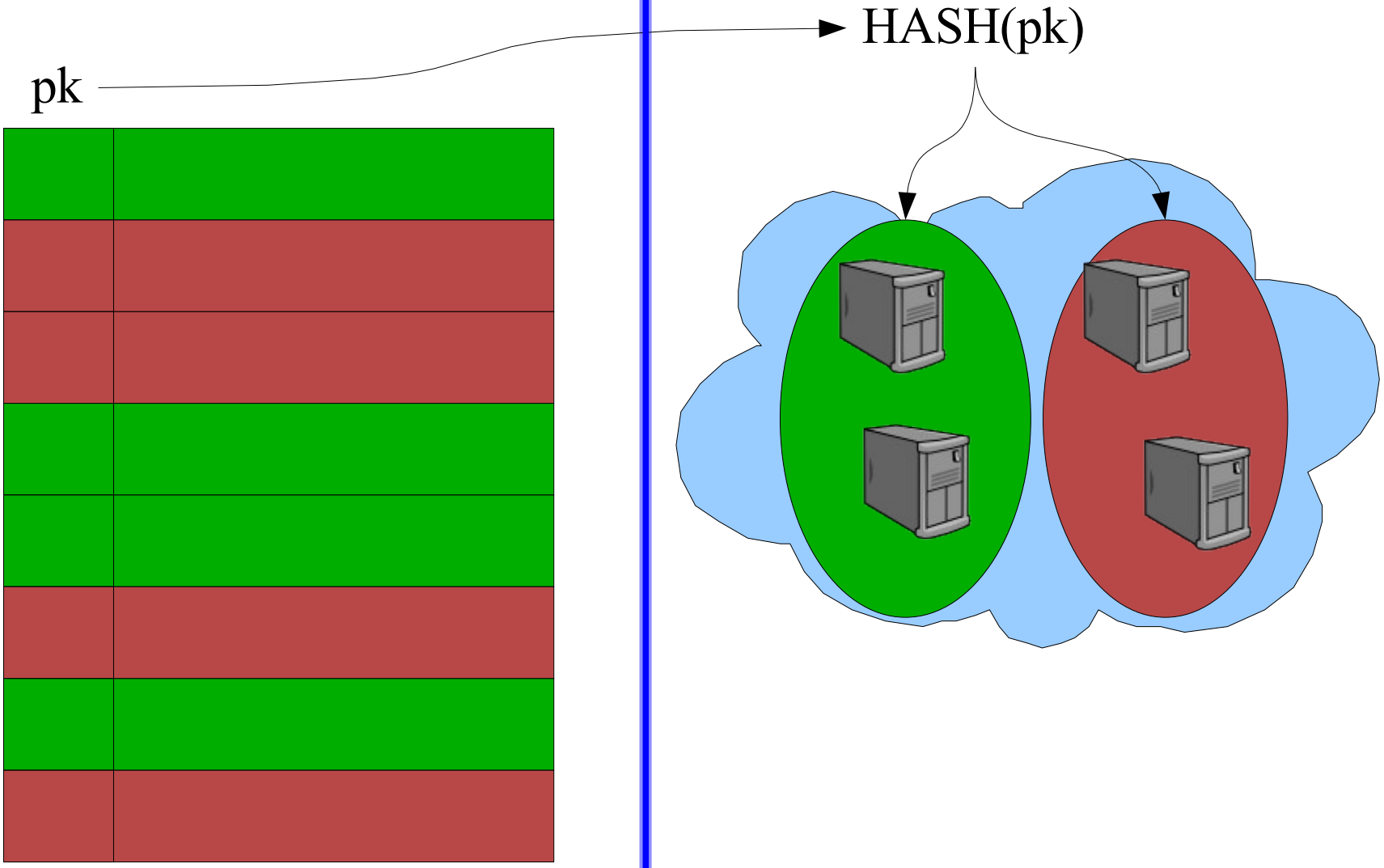


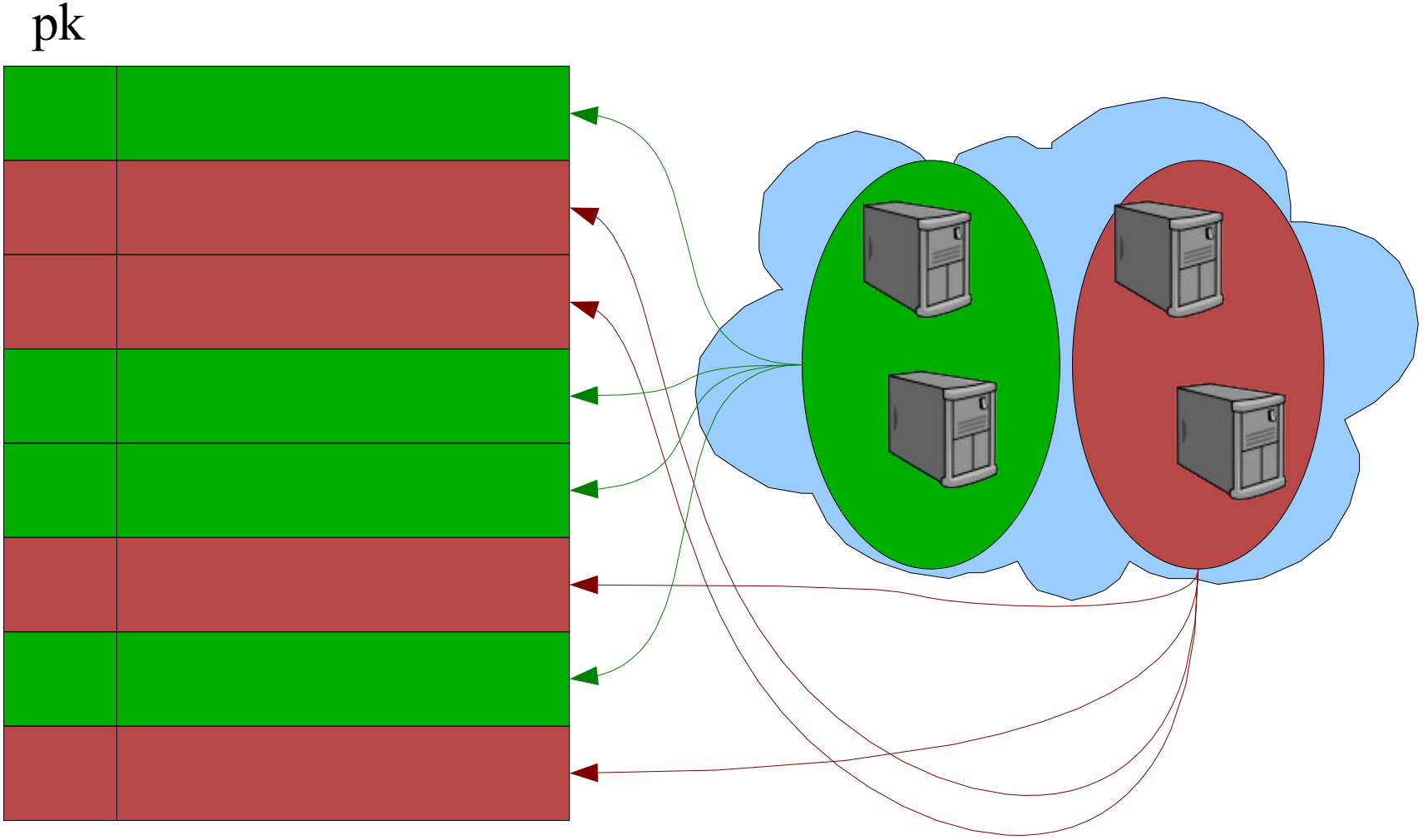




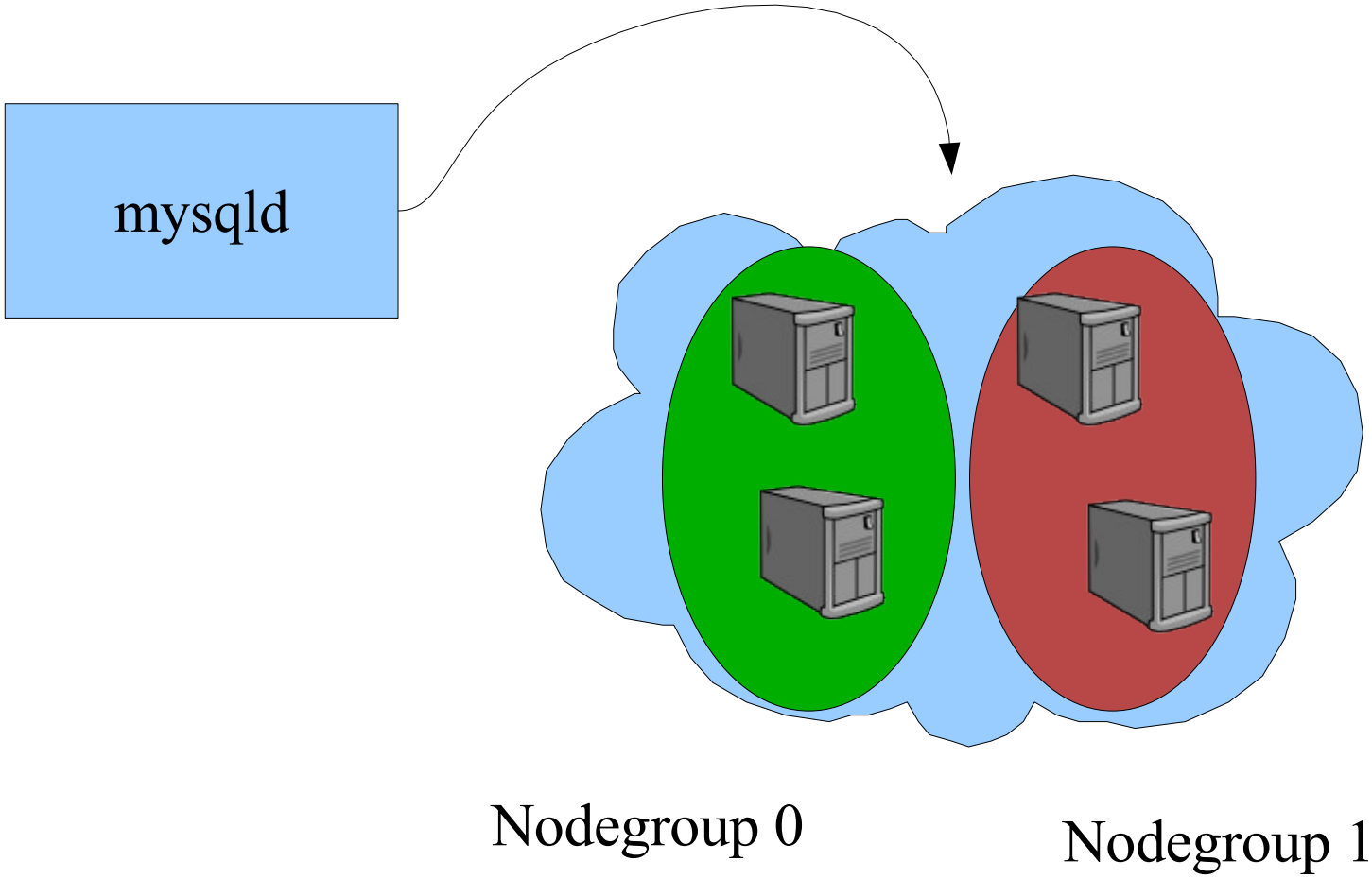
Perception

Reality

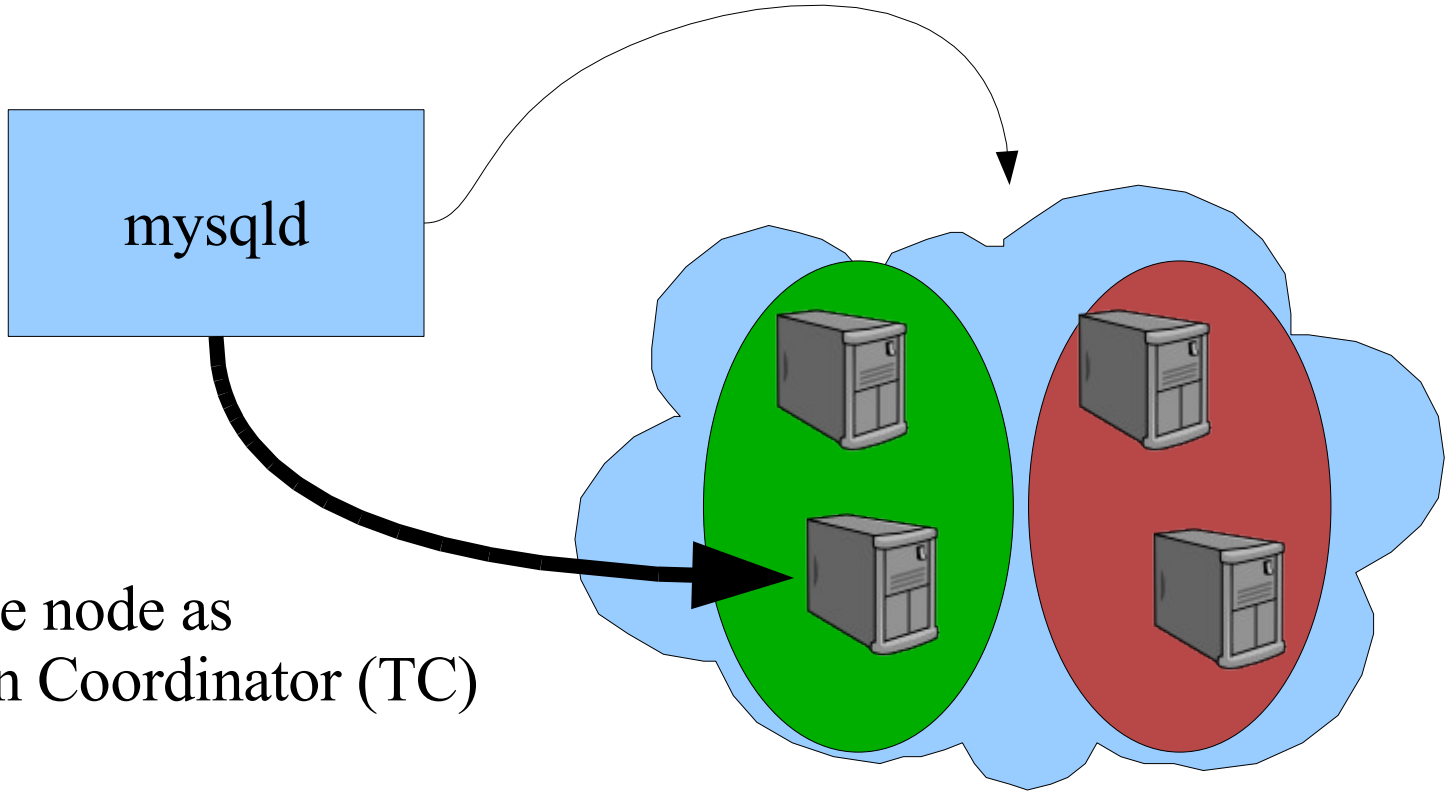




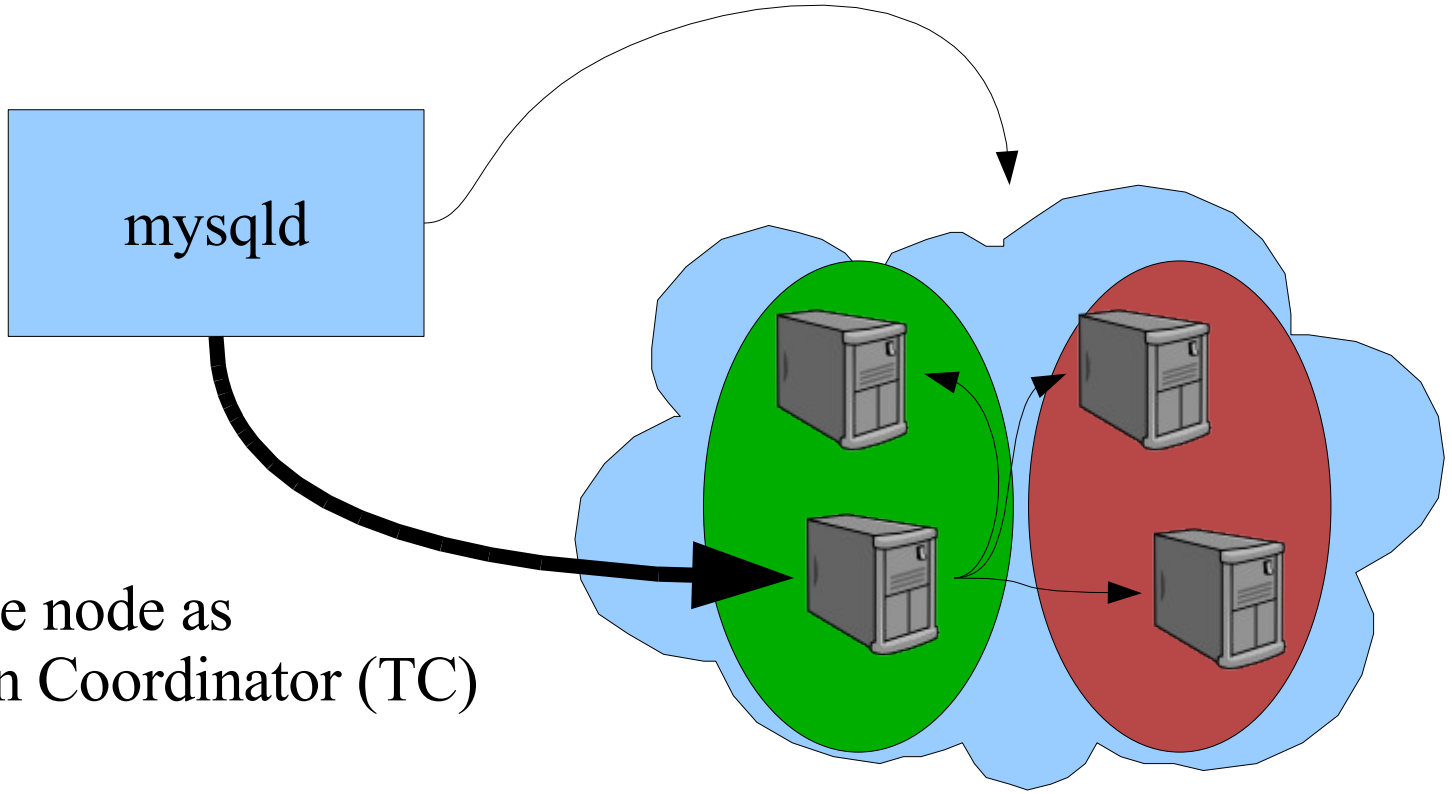
MySQL Servers talk to the Data Nodes



One used as a Transaction Coordinator

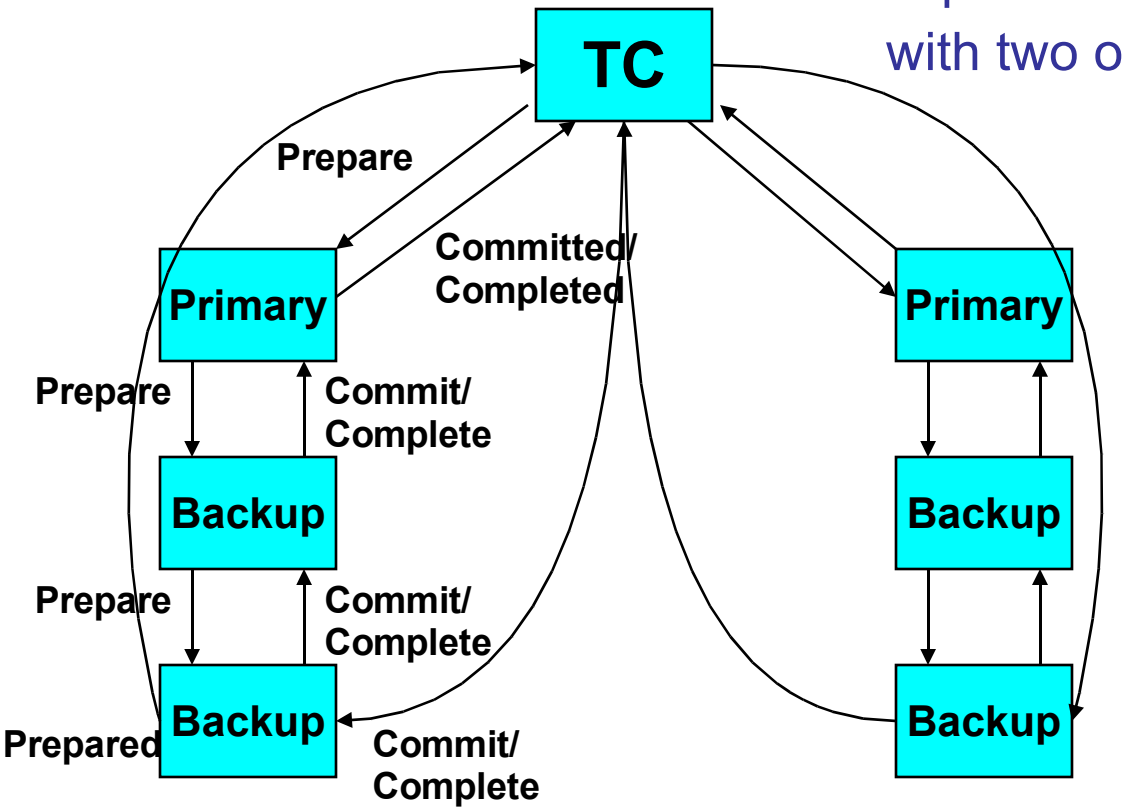


One used as a Transaction Coordinator



Variant of 2-phase commit protocol

Example showing transaction with two operations using three replicas



1. Prepare F1 2. Commit F1

3. Complete F1

1. Prepare F2 2. Commit F2

3. Complete F2

Node Types

Data Nodes (ndbd)

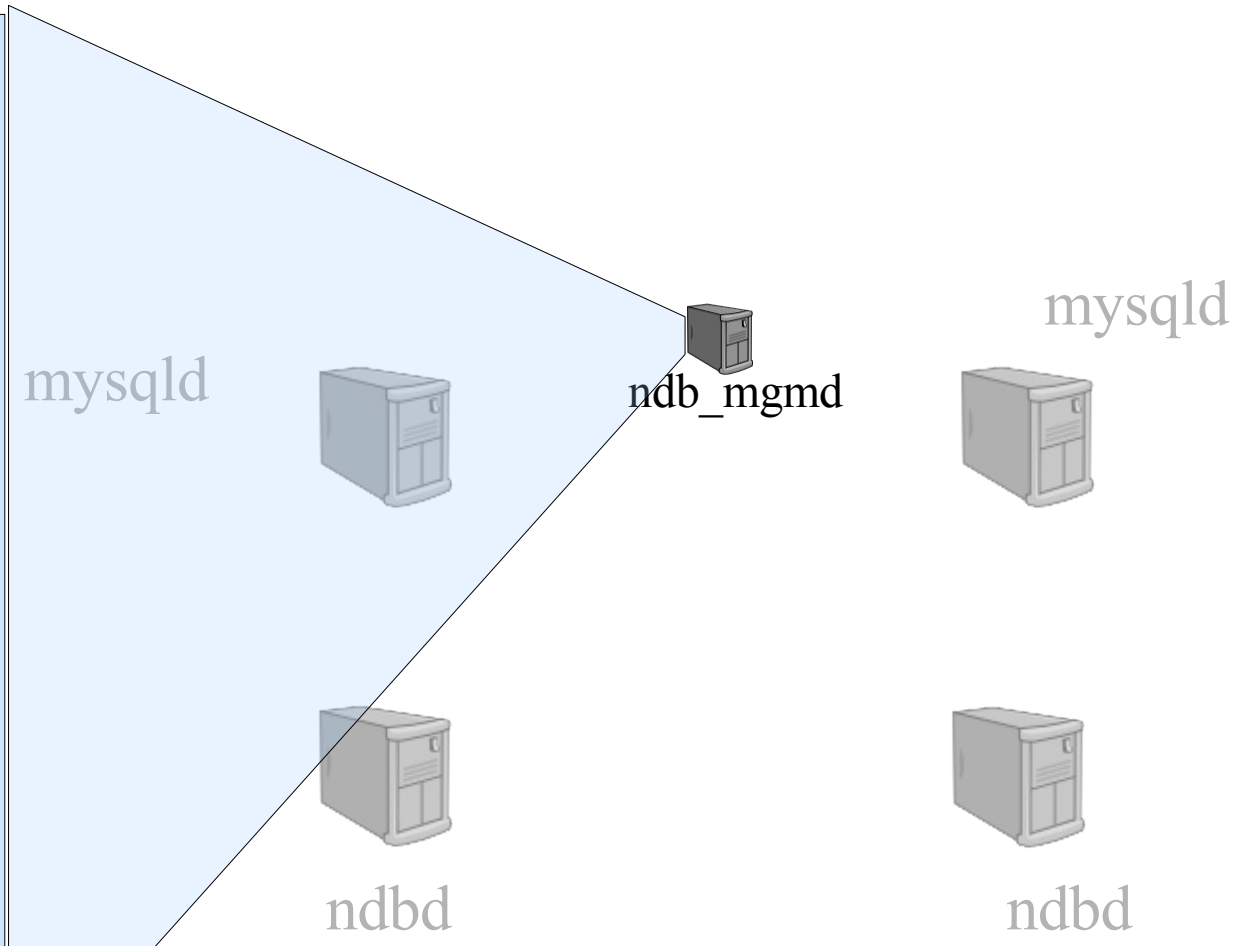
Up to 48 in one cluster

Management Server (ndb_mgmd)

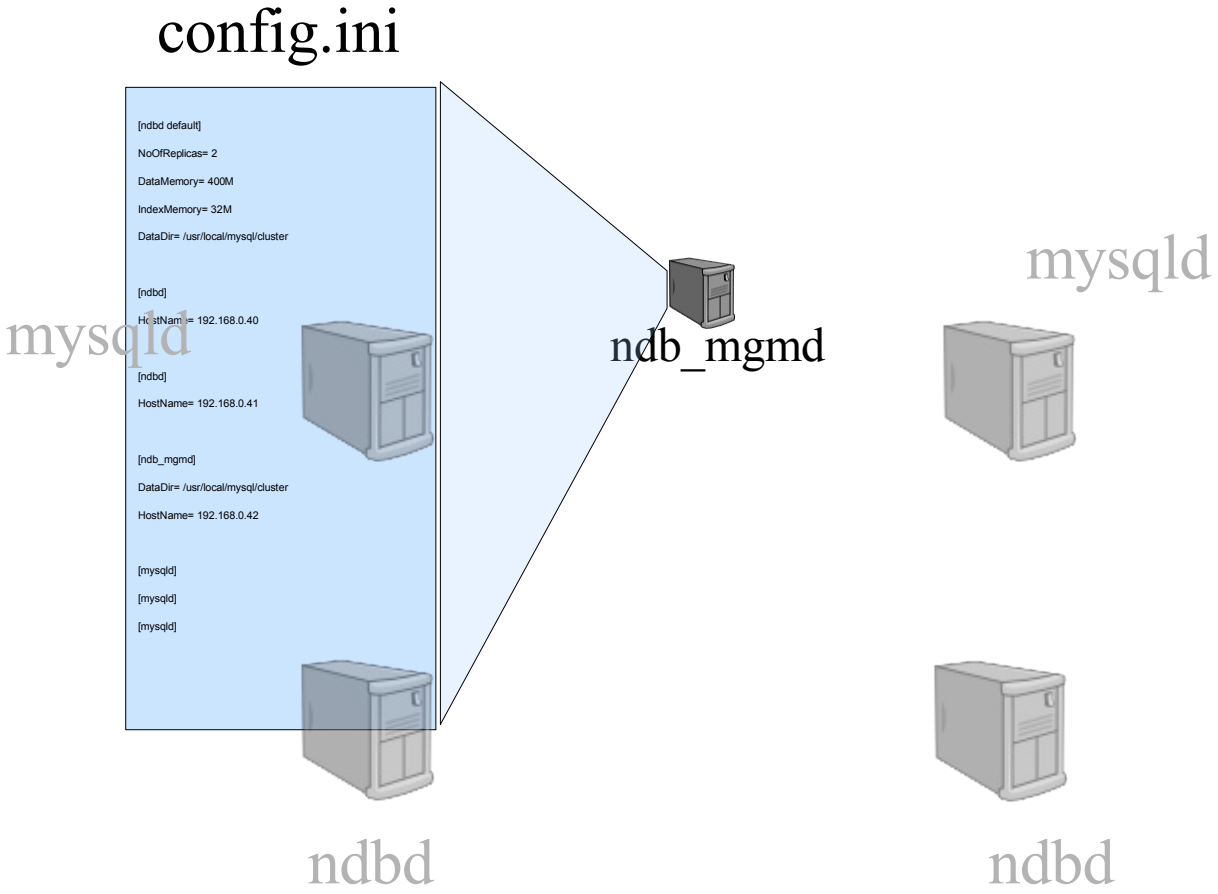
Management Server

config.ini

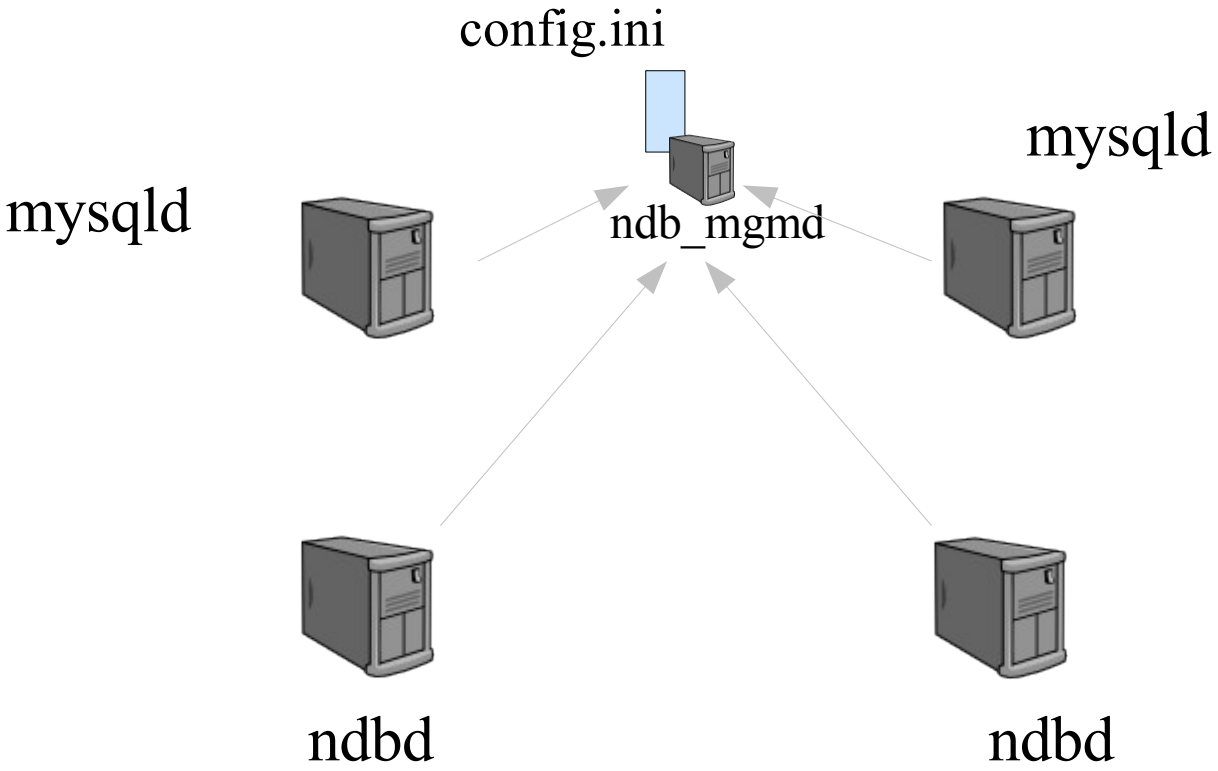
```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```



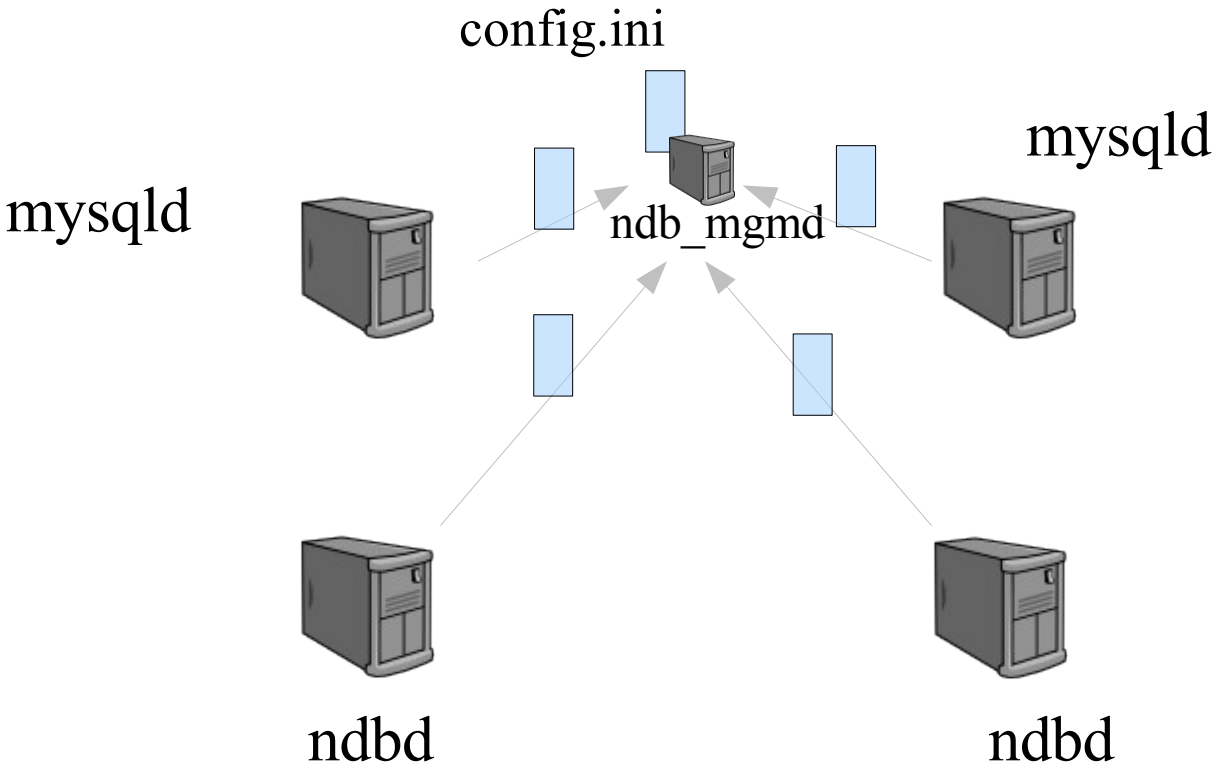
Management Server



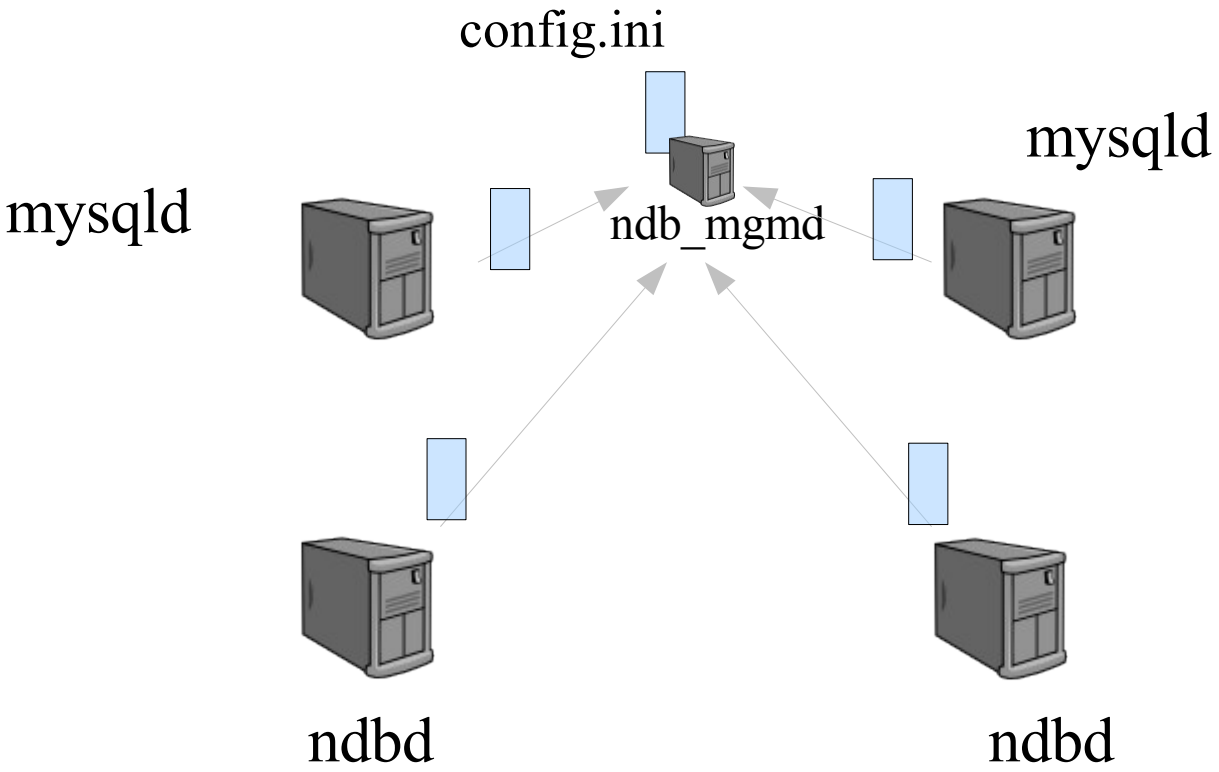
Management Server



Management Server



Management Server



Node Types

Management Server (ndb_mgmd)

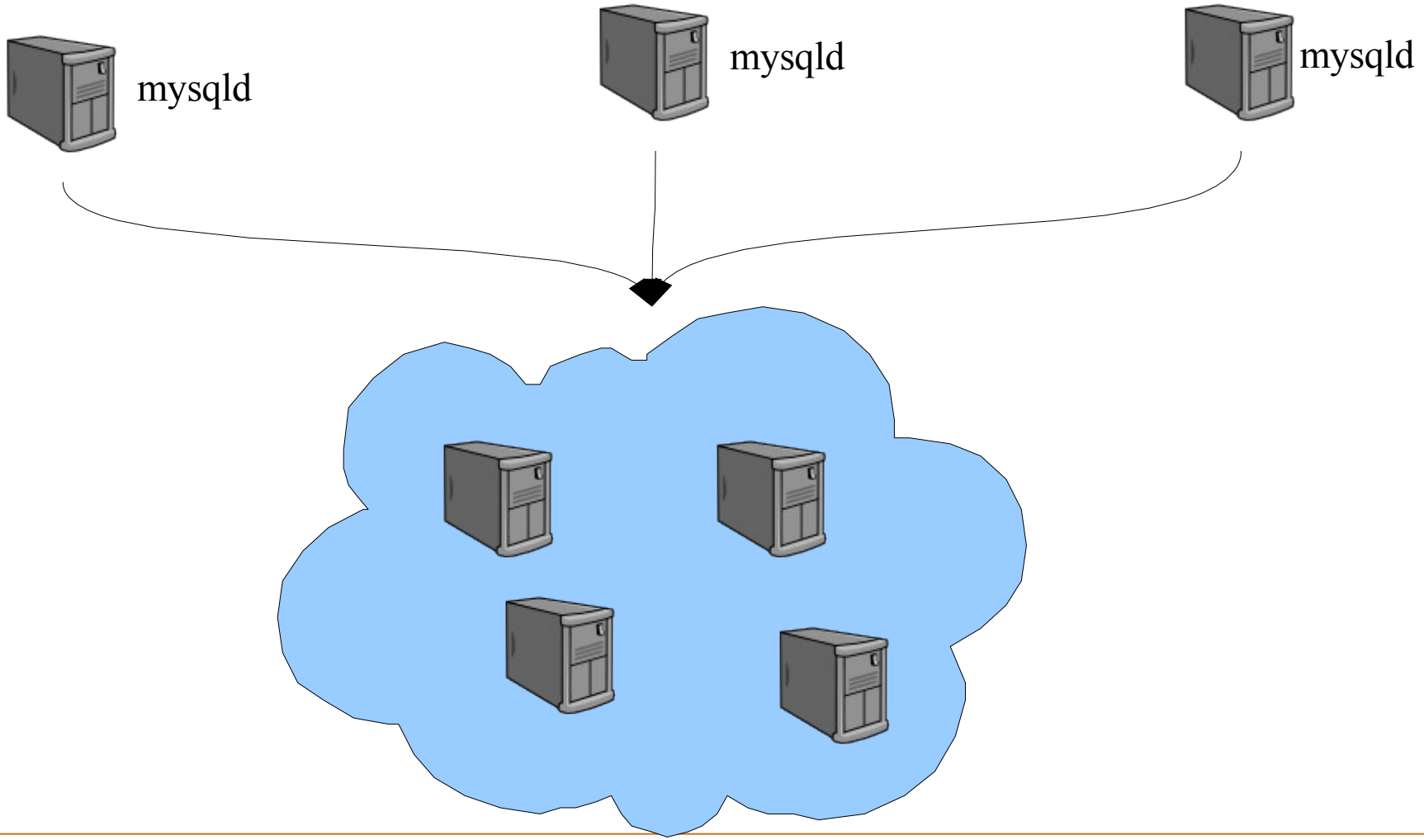
also involved in arbitration, starting backups, issuing commands to nodes (start, stop, restart)

Node Types

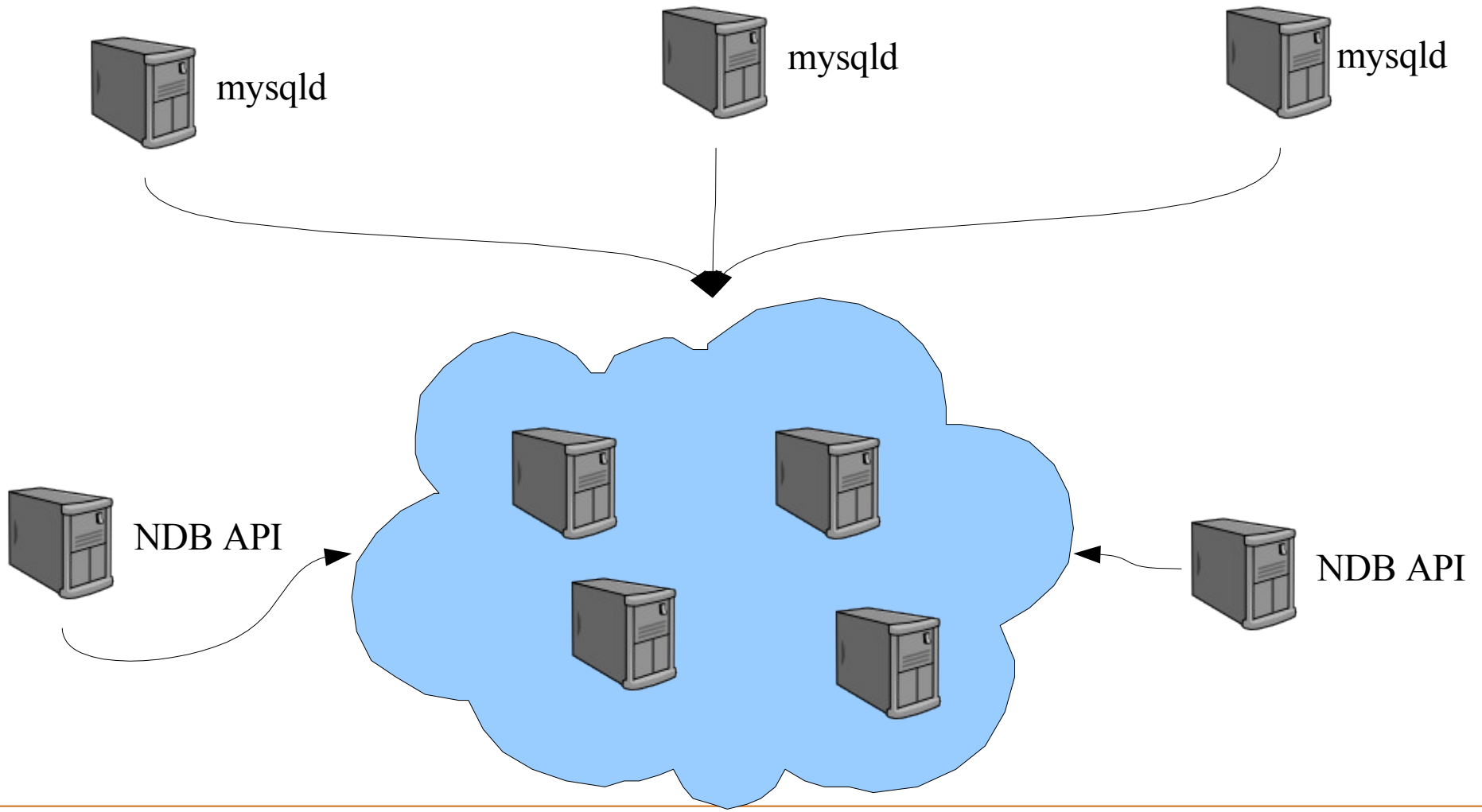
SQL Nodes (mysqld)

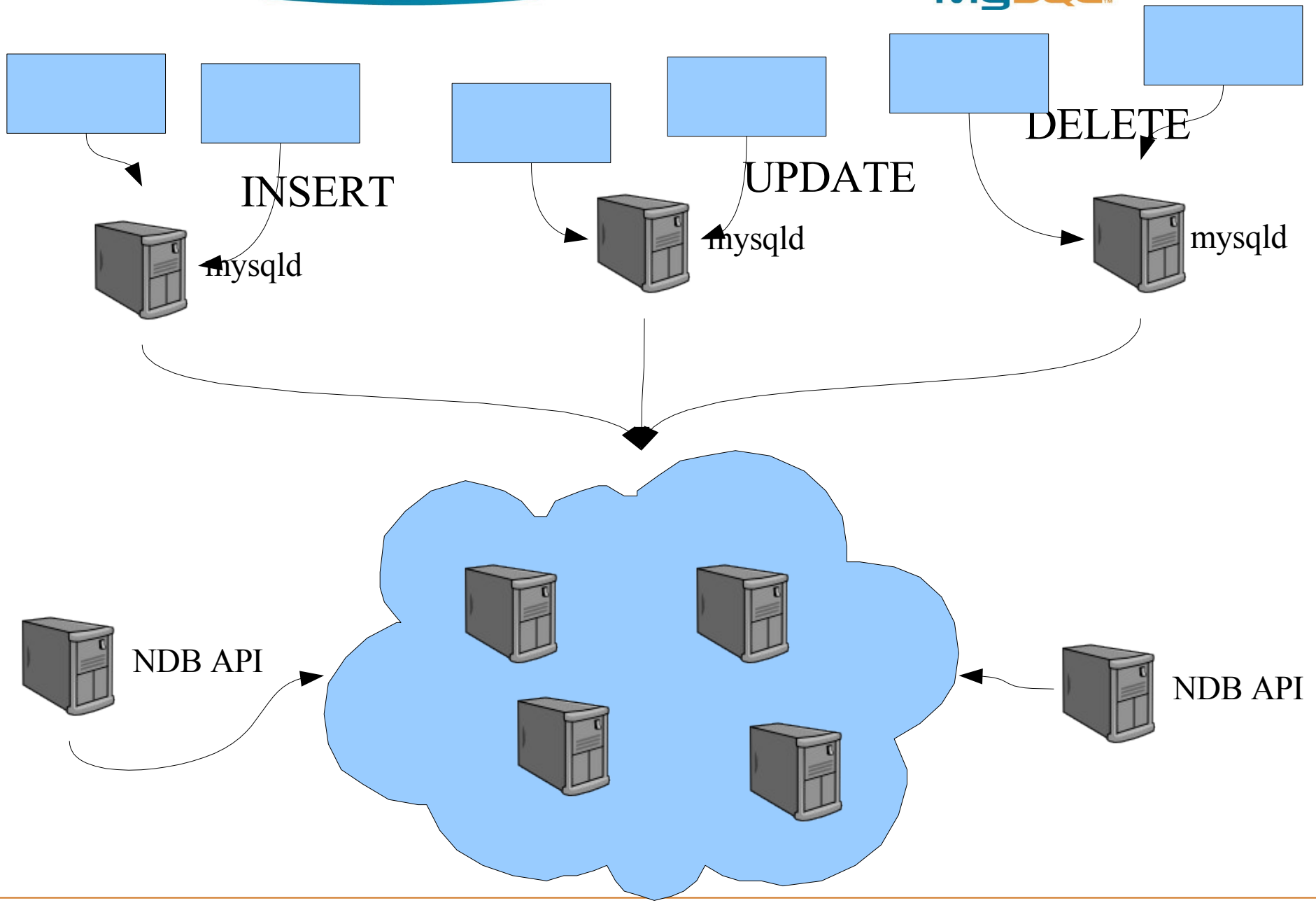
sometimes called API nodes

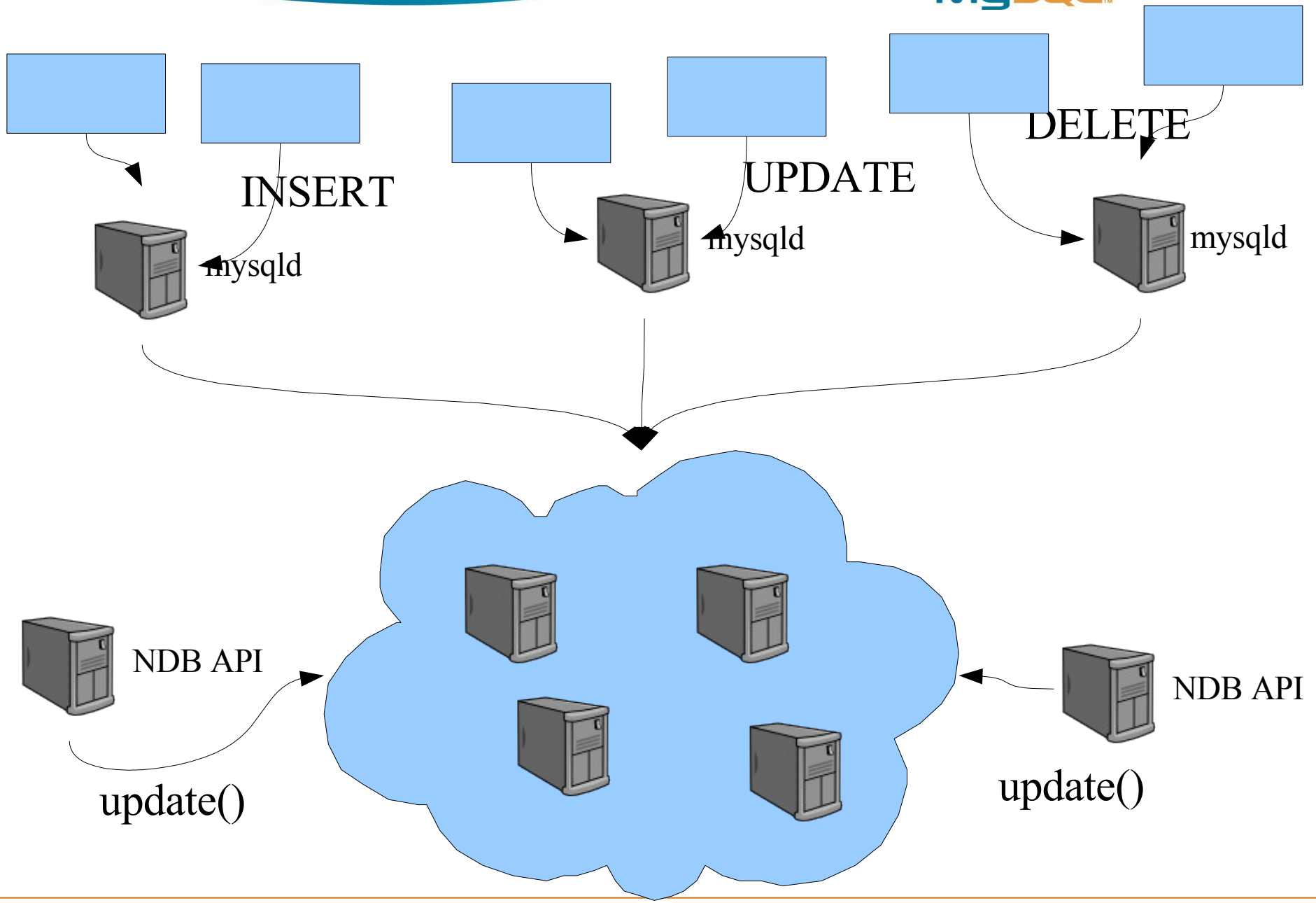
SQL Nodes



SQL and API Nodes







Node Types

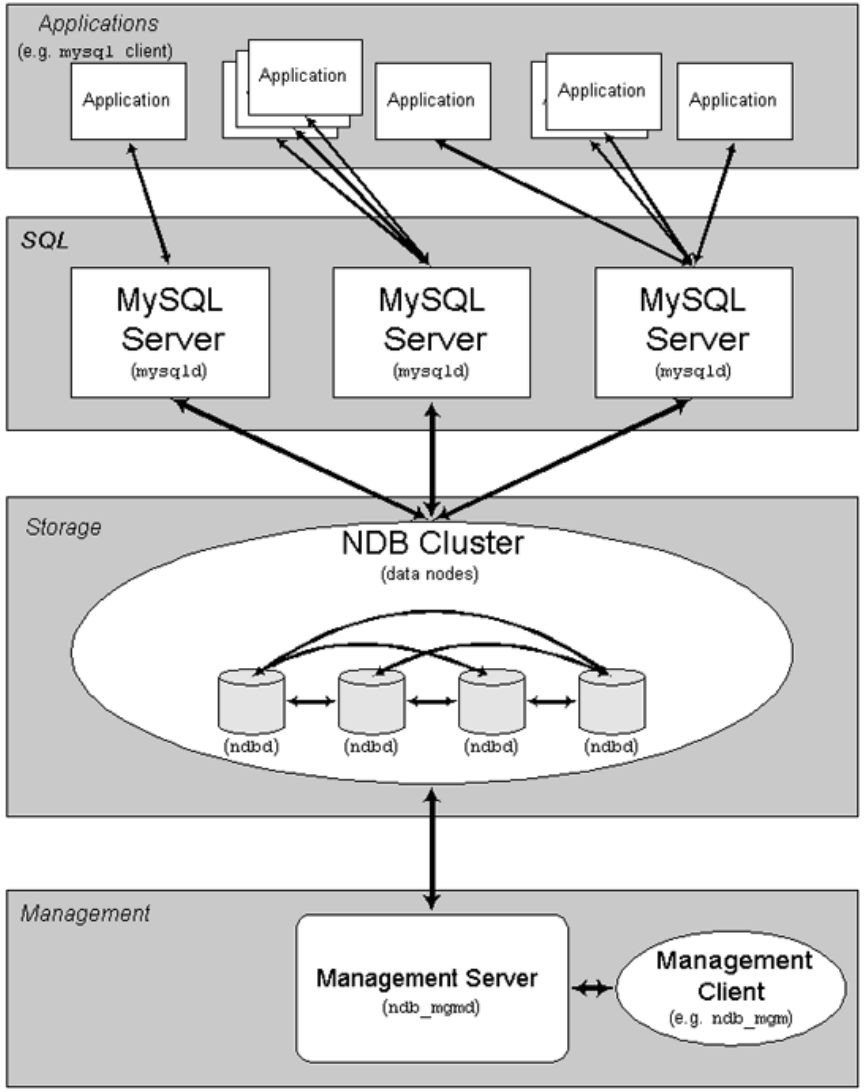
SQL Nodes (mysqld)

Accessed like any other MySQL Server

API Nodes

Talk NDB API directly to the Data Nodes

The full stack



Physical Requirements

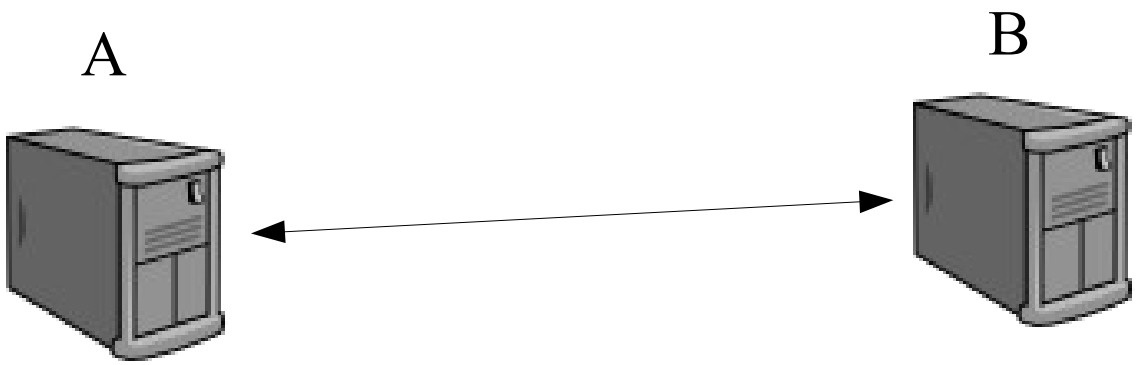
Physical Requirements

A node is a process,
not a computer

Physical Requirements

At least three physical machines
for High Availability

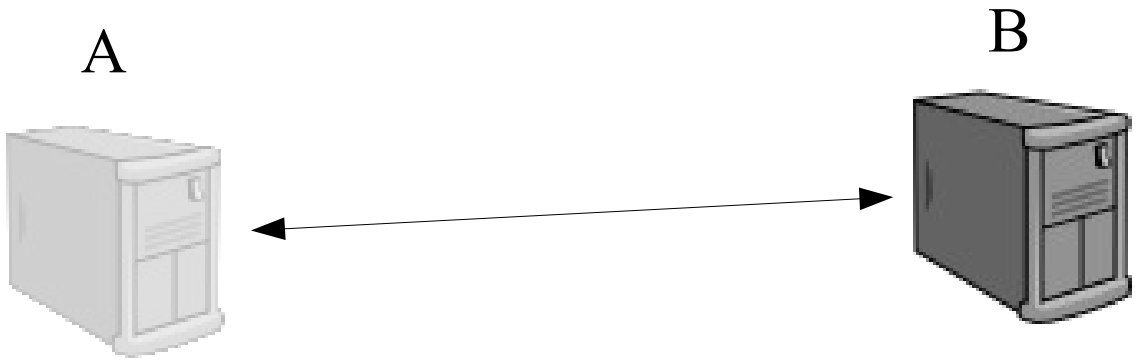
Three machines minimum for HA



Three machines minimum for HA

Can no longer see A

Three machines minimum for HA



Did A Die?

Can no longer see A

Three machines minimum for HA

Or did the network link between A and B die?

Can no longer see A

Three machines minimum for HA

Can no longer see B

Or did the network link between A and B die?

Can no longer see A

Three machines minimum for HA

Who is in charge now?

Can no longer see B

Or did the network link between A and B die?

Can no longer see A

Three machines minimum for HA

Split Brain = Badness

Can no longer see B

Or did the network link between A and B die?

Can no longer see A

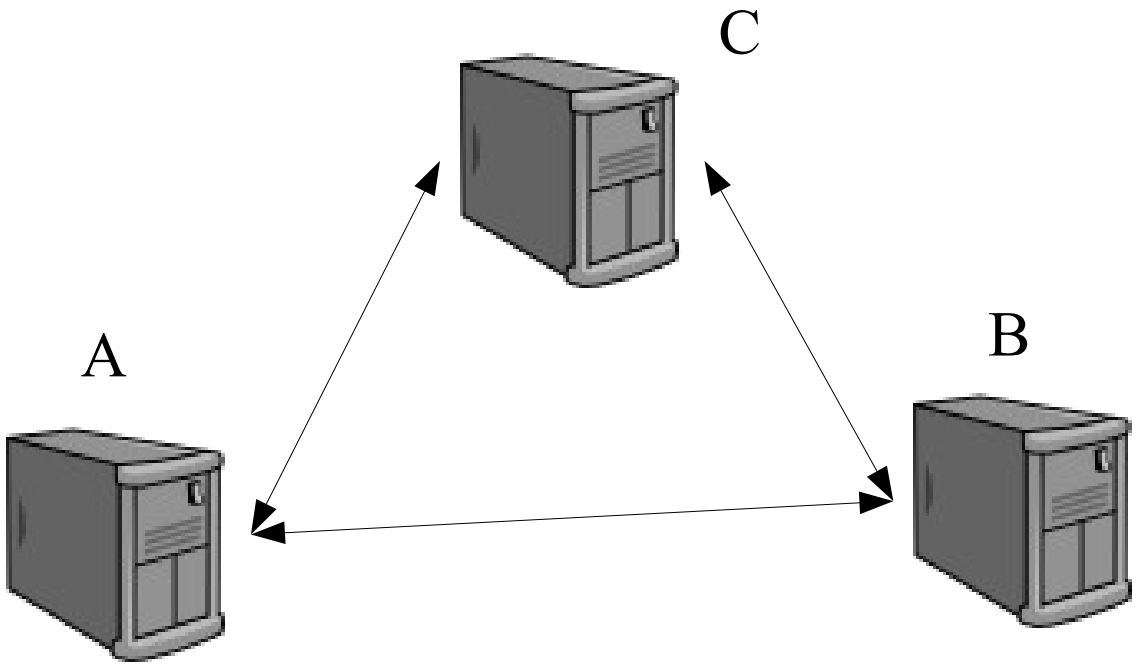
Three machines minimum for HA

Split Brain = Badness

We detect possible
split brain scenarios.

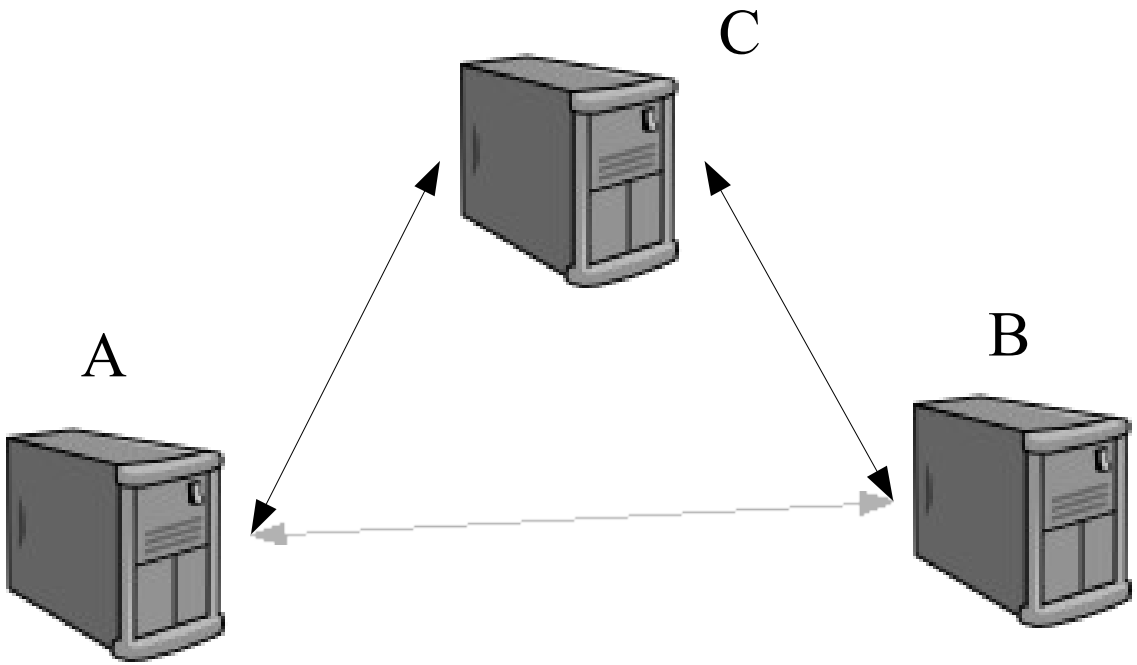
Nodes will shut down instead

Three machines minimum for HA



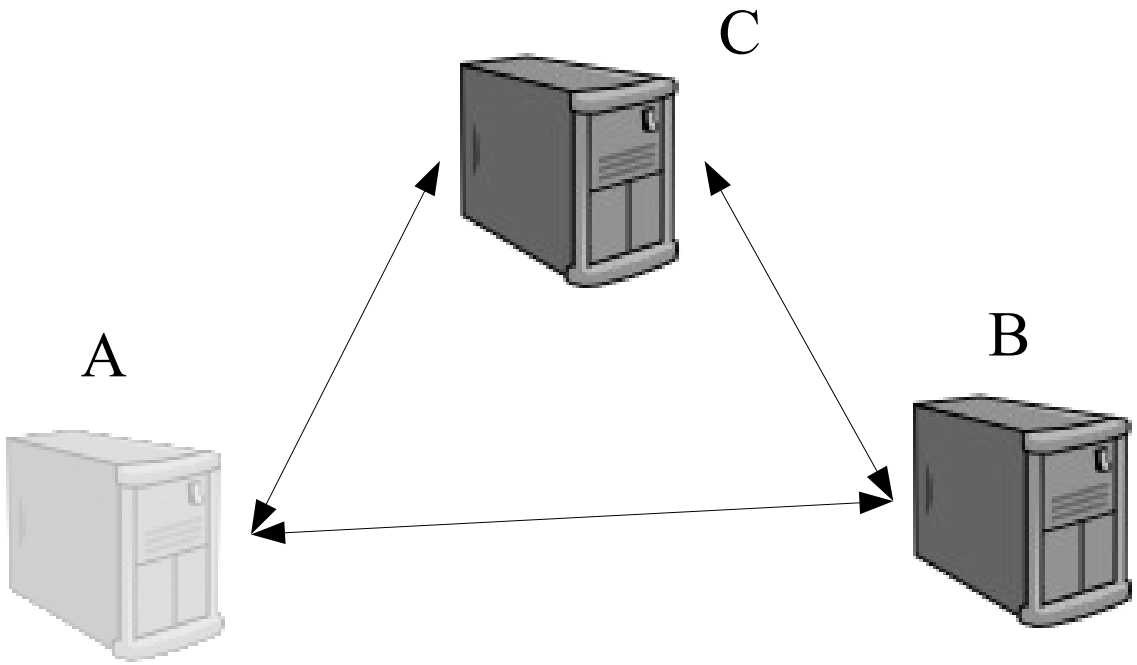
Management server on 3rd machine

Three machines minimum for HA



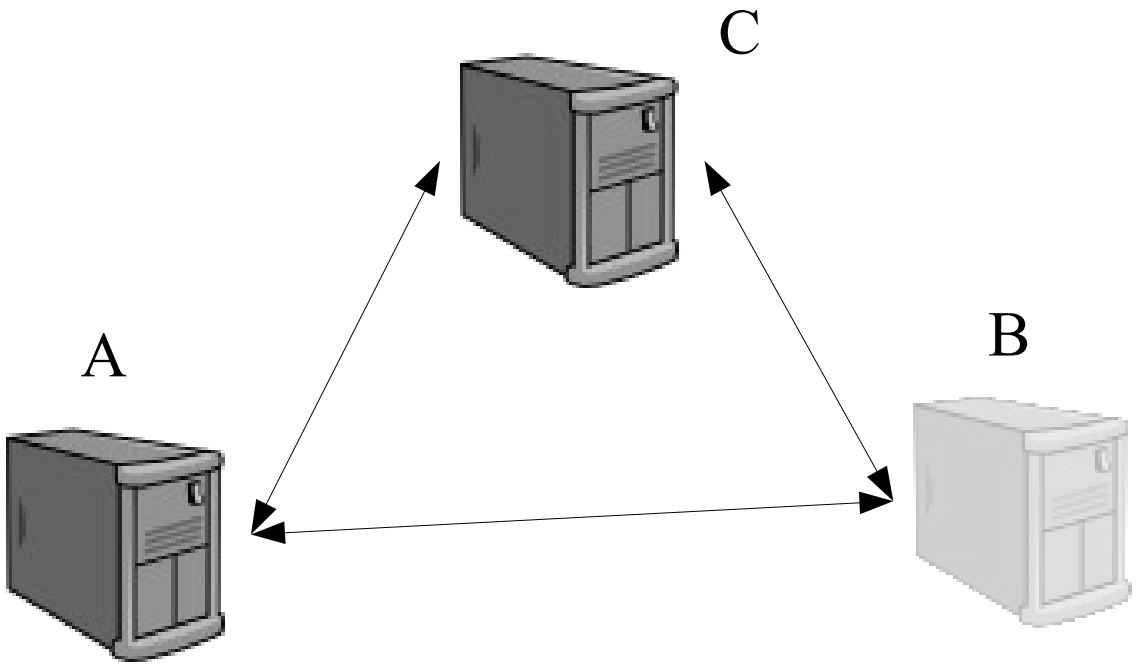
Management server on 3rd machine
Is **The Decider**

Three machines minimum for HA



Management server on 3rd machine
Is **The Decider**

Three machines minimum for HA



Management server on 3rd machine
Is **The Decider**

Physical Requirements

Management Server:
Not CPU intensive

Physical Requirements

- Data Nodes
  - Lots of Memory
  - Disk I/O Can be calculated
  - (generally) not CPU bound
  - ndbd is single threaded
    - multiple nodes for SMP

Physical Requirements

SQL Node:

Many API/SQL nodes are needed to load Storage Nodes

Physical Requirements

SQL Node:

MySQL is multi-threaded – takes advantage of SMP transparently

A Configuration

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```

A Configuration

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```

ndb_mgmd

A Configuration

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```

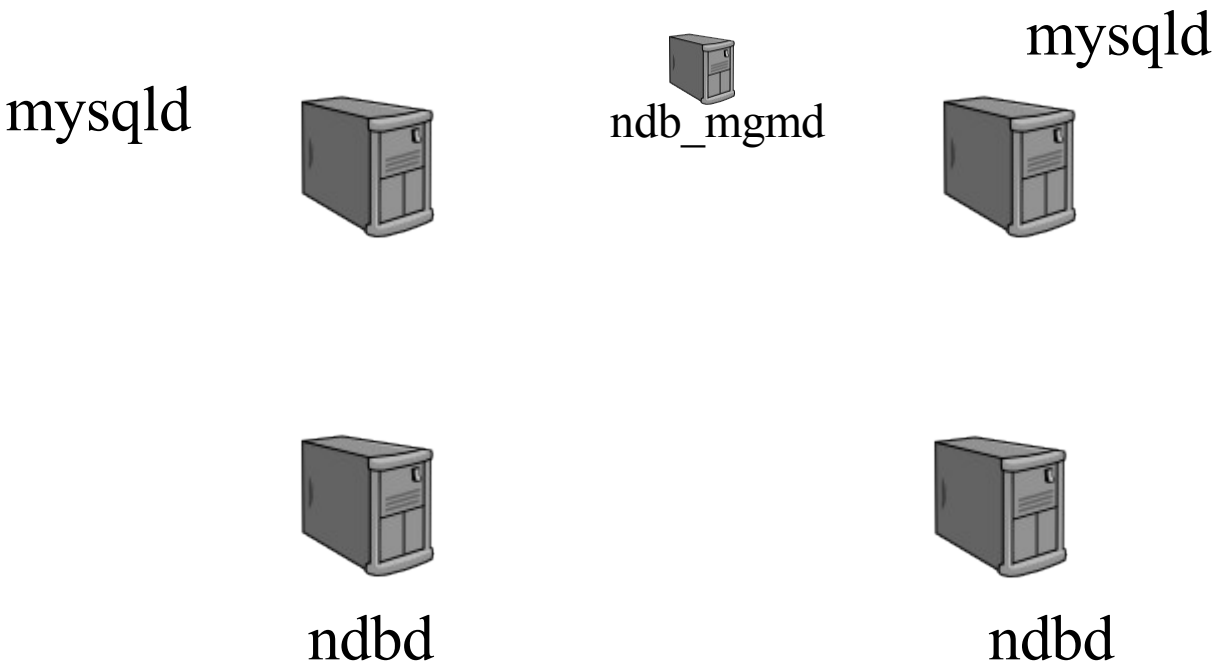
ndb_mgmd

ndbd

ndbd

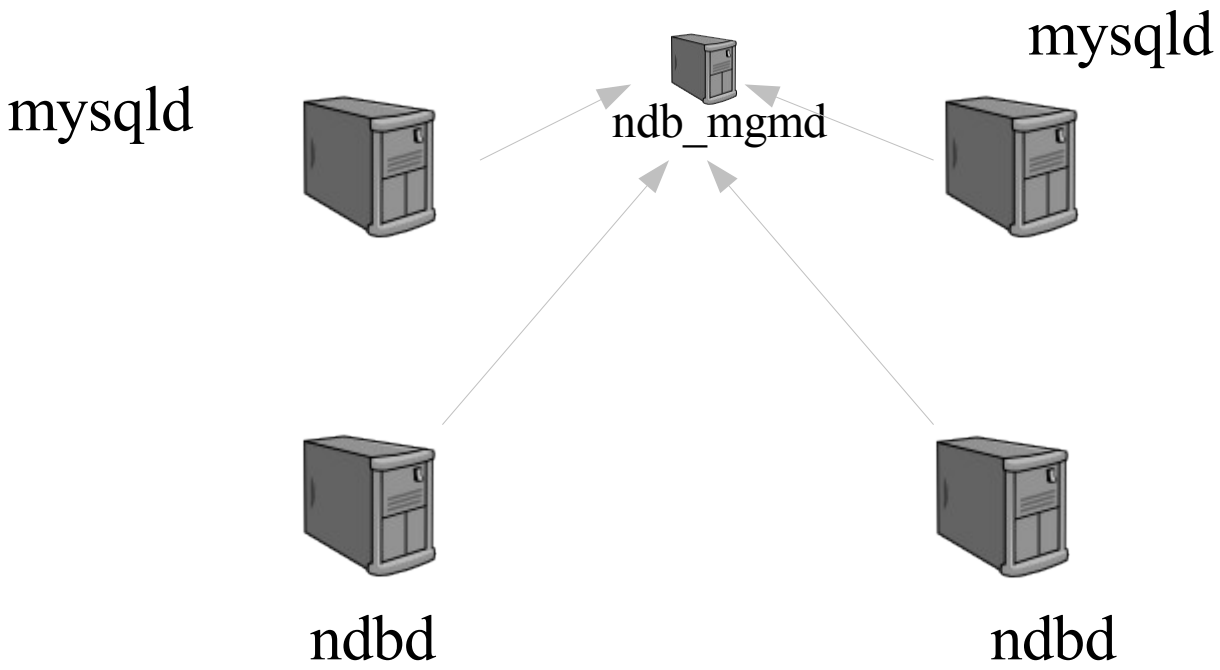
A Configuration

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```



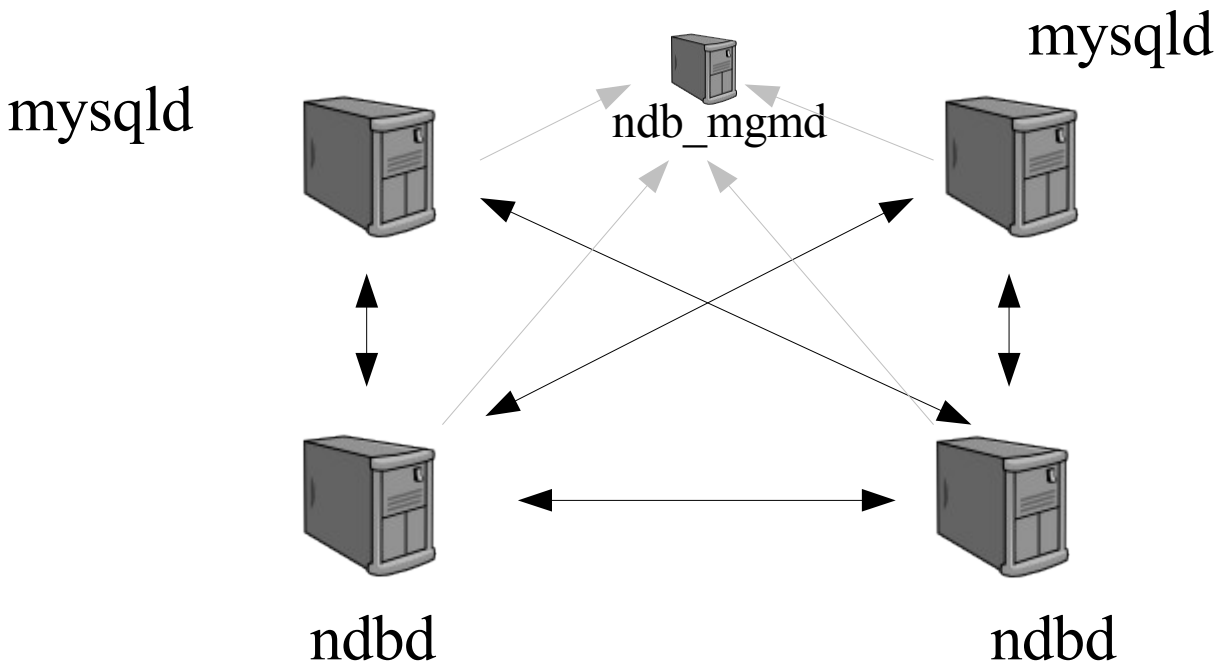
A Configuration

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```



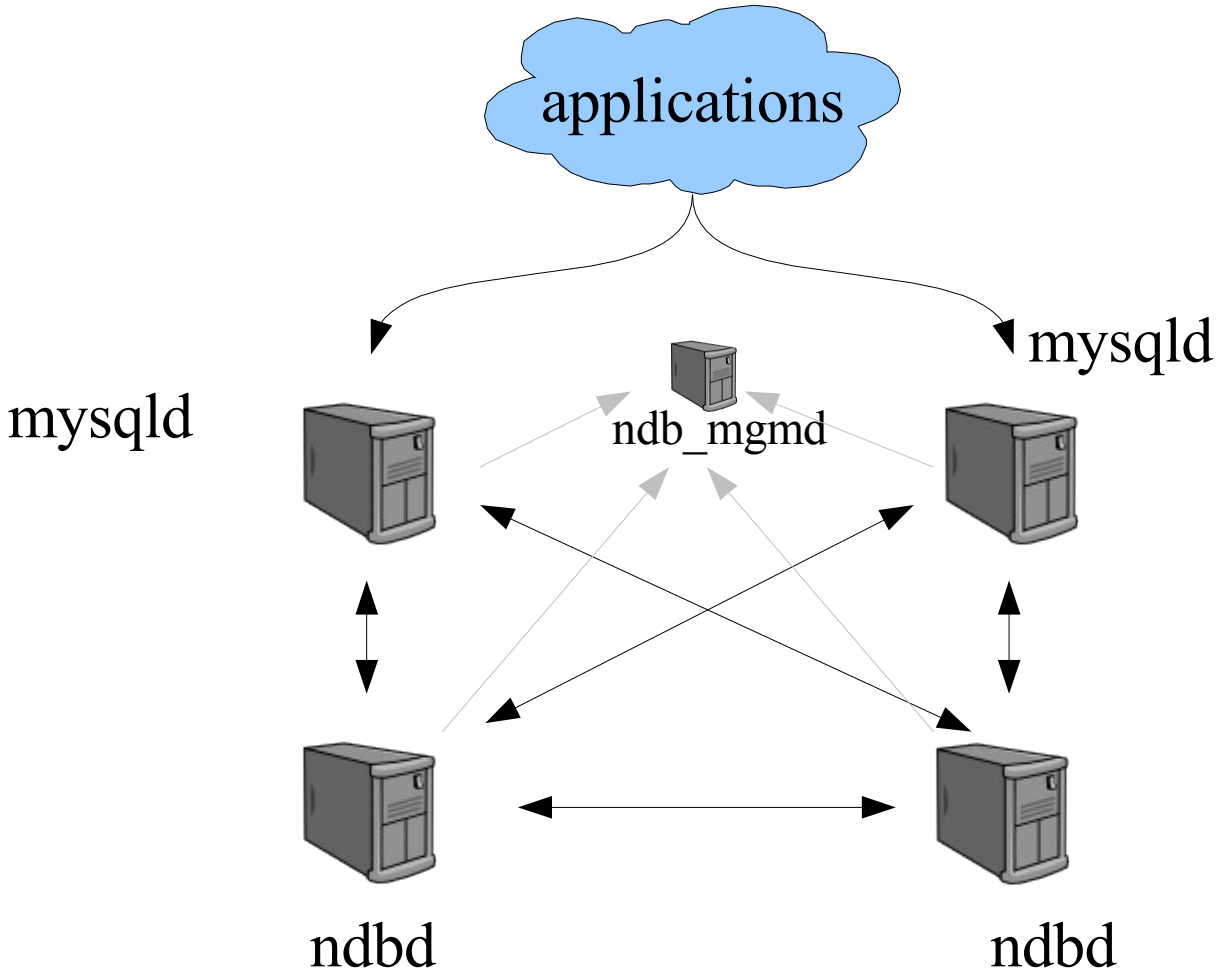
A Configuration

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```



A Configuration

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```



Using the Cluster

- MySQL Cluster is a storage engine
  - tables can be stored in the cluster simply by ENGINE=NDBCLUSTER
- No Foreign Key constraints
- Supports 5.0 features:
  - Views
  - Stored Procedures
  - Triggers
  - standard permissions
  - these need to be set up on each SQL node
    - as the mysql.* tables aren't stored as NDB tables

Failure Scenarios

- MySQL Server Node
  - Can be restarted and reconnected to the cluster.
  - Applications can connect to other MySQL Server nodes
- Data Node
  - All other storage nodes are informed about failure
  - Data is replicated, so there is another storage node to service transaction requests
  - Currently, transactions using a failed node are aborted, and have to be restarted by the application
- Management Server Node
  - Continued operation not dependent on management server
  - may be restarted, or have redundant nodes

Schema Considerations

- All tables have a primary key
  - if no primary key is explicitly set, NDBCLUSTER creates a hidden field
- Three types of indexes
  - Primary Hash Index
  - Unique Hash Index
  - Ordered Tree Index
- Creating a UNIQUE index from SQL will also create an ordered index unless USING HASH is specified
  - requires knowing how you application will query your DB
- Primary key is primary hash index and ordered index
  - unless USING HASH is specified

Optimizations

- An improvement in 5.0 is the engine_condition_pushdown option
  - SELECT A from T where unindexed_field = 42;
  - The condition is evaluated in the data nodes, not in the SQL node
  - saving a full table scan
    - which can be rather expensive on large tables
  - Common to see 5-10x speed improvement
  - details in EXPLAIN output
- In 5.0, we integrate with the query cache
- Batched lookups
  - SELECT * from t1 where primary_key IN (1,2,3,4,5,6,7,8,9,10);
  - all 10 key lookups are sent in one batch rather than one at a time
  - 2-3 times performance gain

Condition Pushdown

- Conditions on non-key/index attributes are pushed down to data nodes whenever possible.

```
select * from t2,t3 where t2.attr2 = t3.attr2
  and t2.attr1 < 1 and t3.attr1 < 5;
```

  Join query is executed as two nested scans on table t2 and t3.The scan on t2 will push the condition `t2.attr1 < 1` and the scan on t3 will push the condition `t3.attr1 < 5`

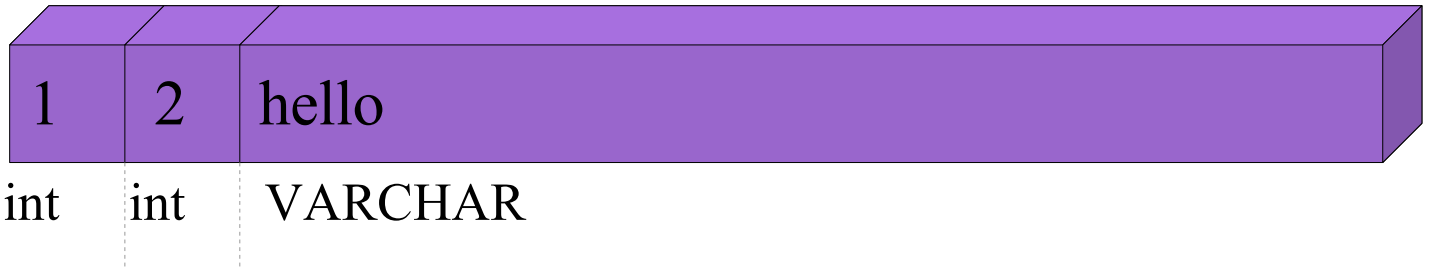
- Can be a arbitrary nesting of AND/OR/NOT and support the following operators=,=!,<,<=,>,>=,IS NULL, IS NOT NULL, LIKE, and NOT LIKE.
- Is enabled|disabled by

```
set engine_condition_pushdown=on|off;
```

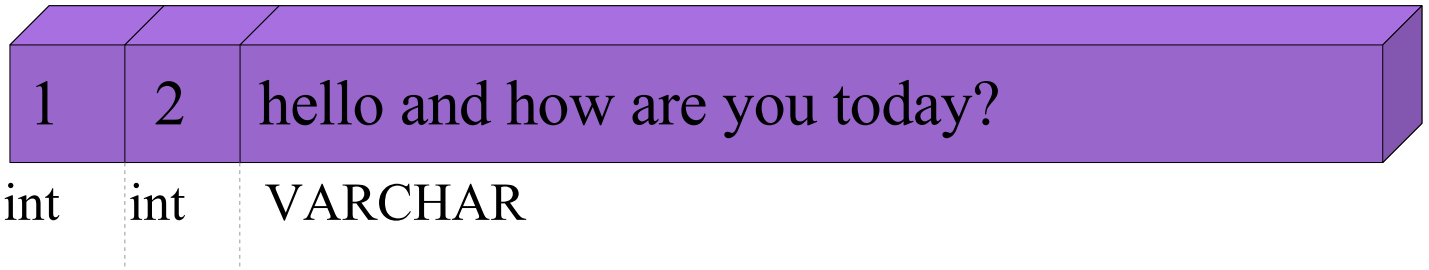
New in 5.1

Variable Sized Rows

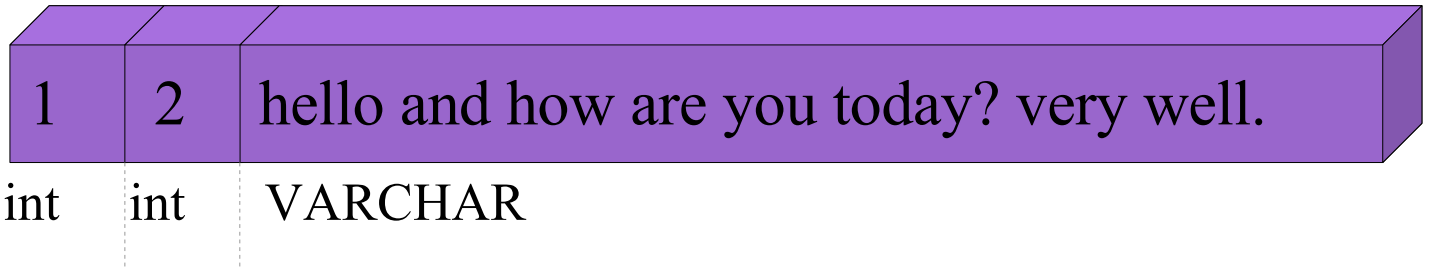
4.1 and 5.0



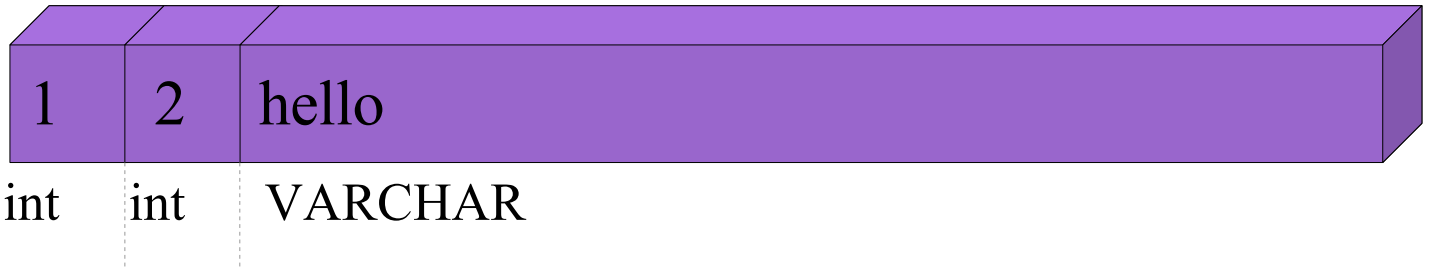
4.1 and 5.0



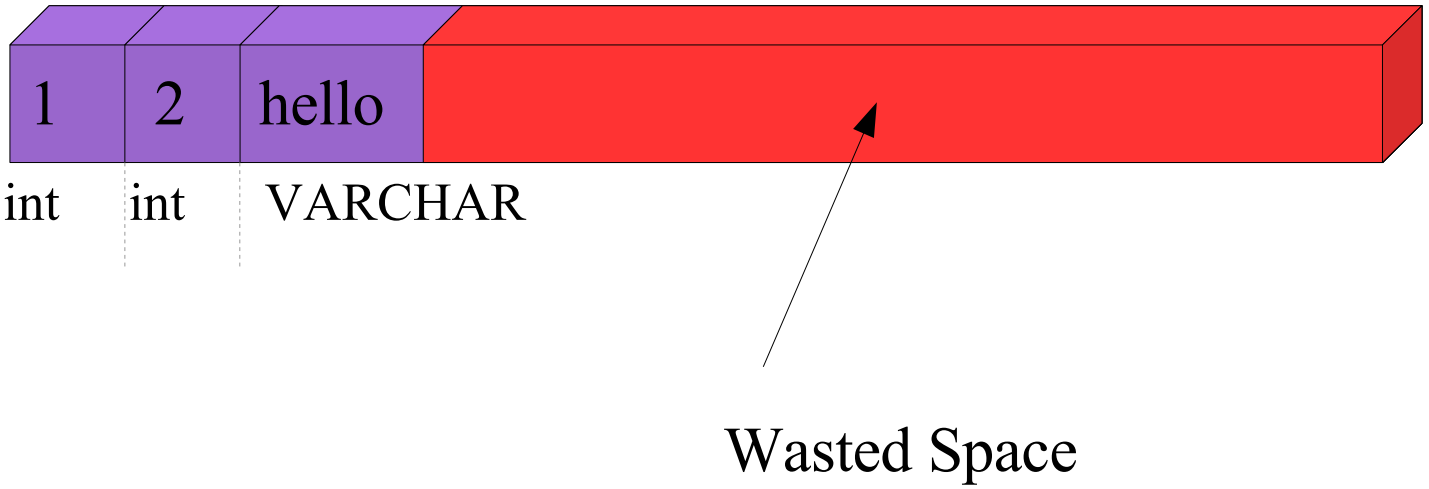
4.1 and 5.0



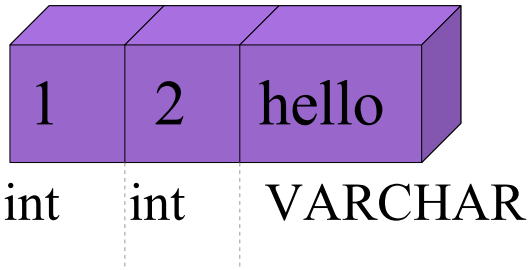
4.1 and 5.0



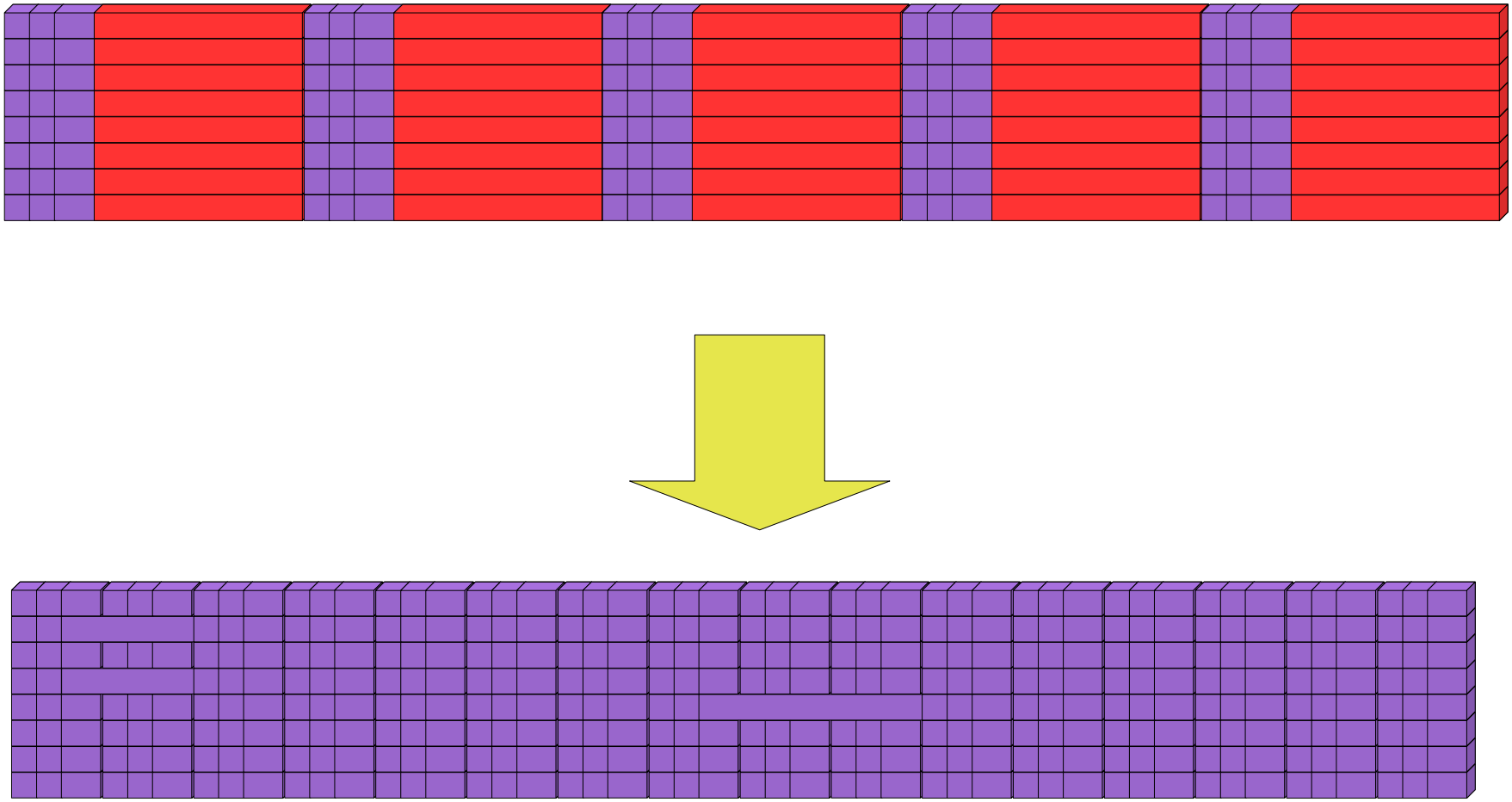
Wasted Space



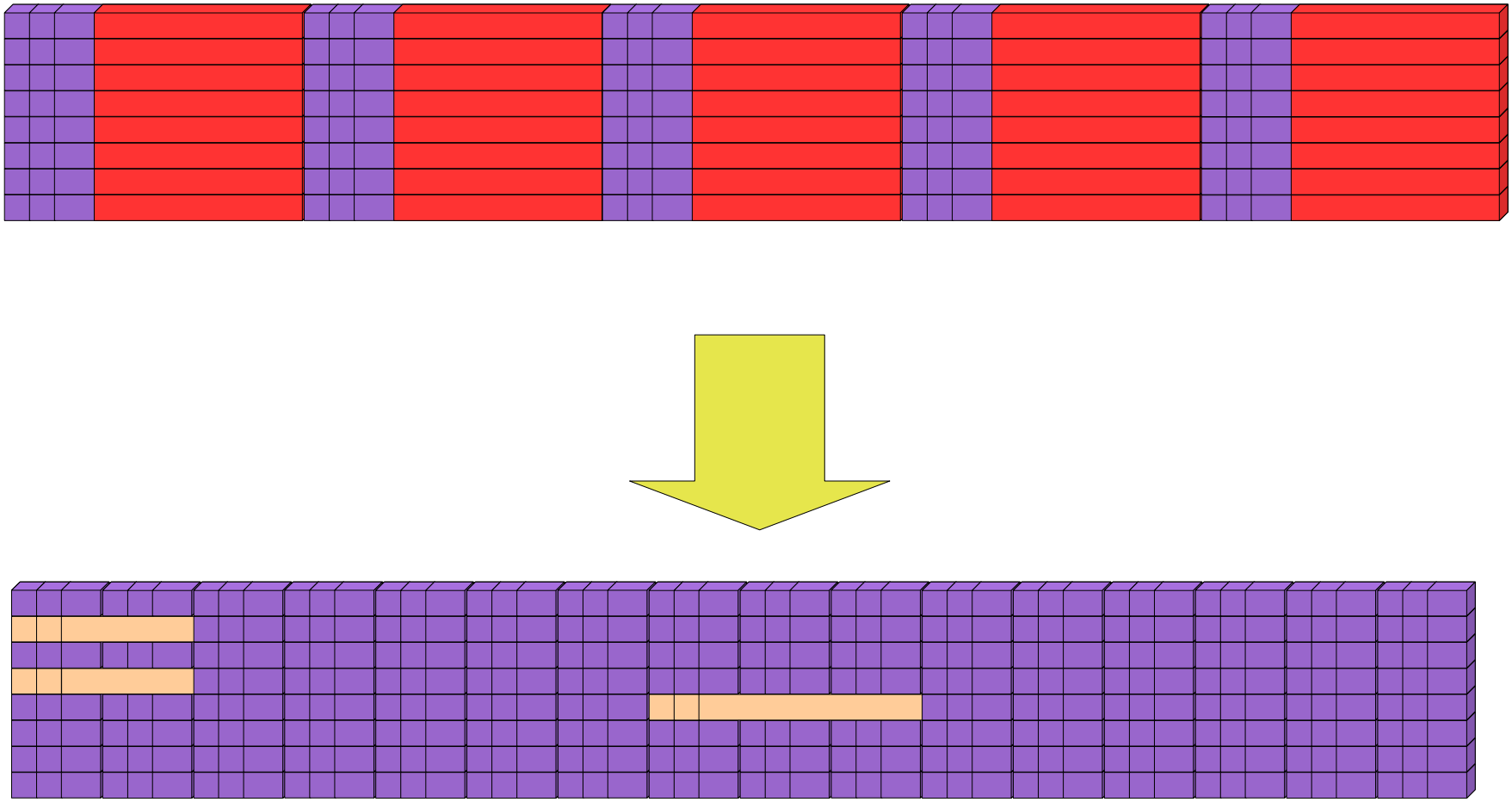
5.1



The Saving



The Saving



Variable Sized Rows

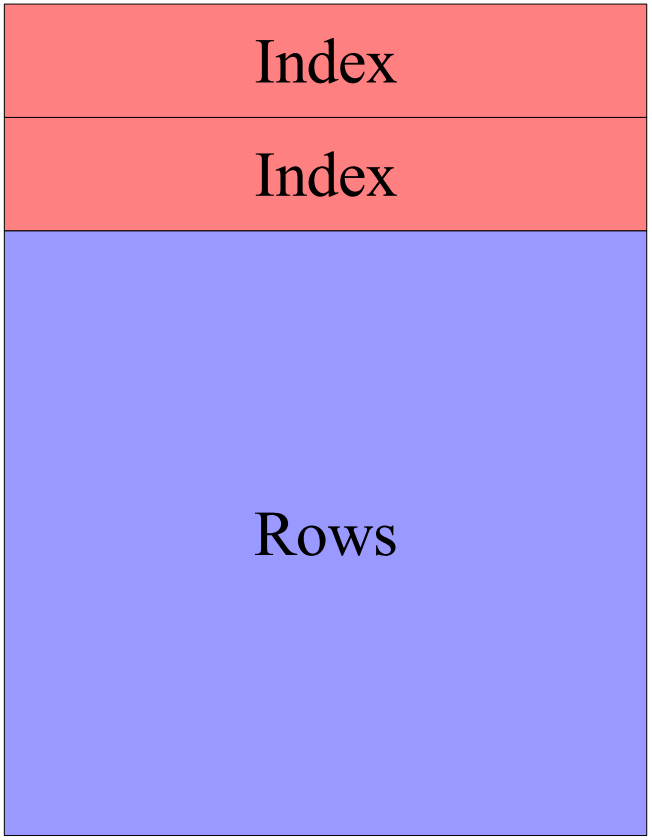
More rows per gigabyte

Larger datasets in Cluster

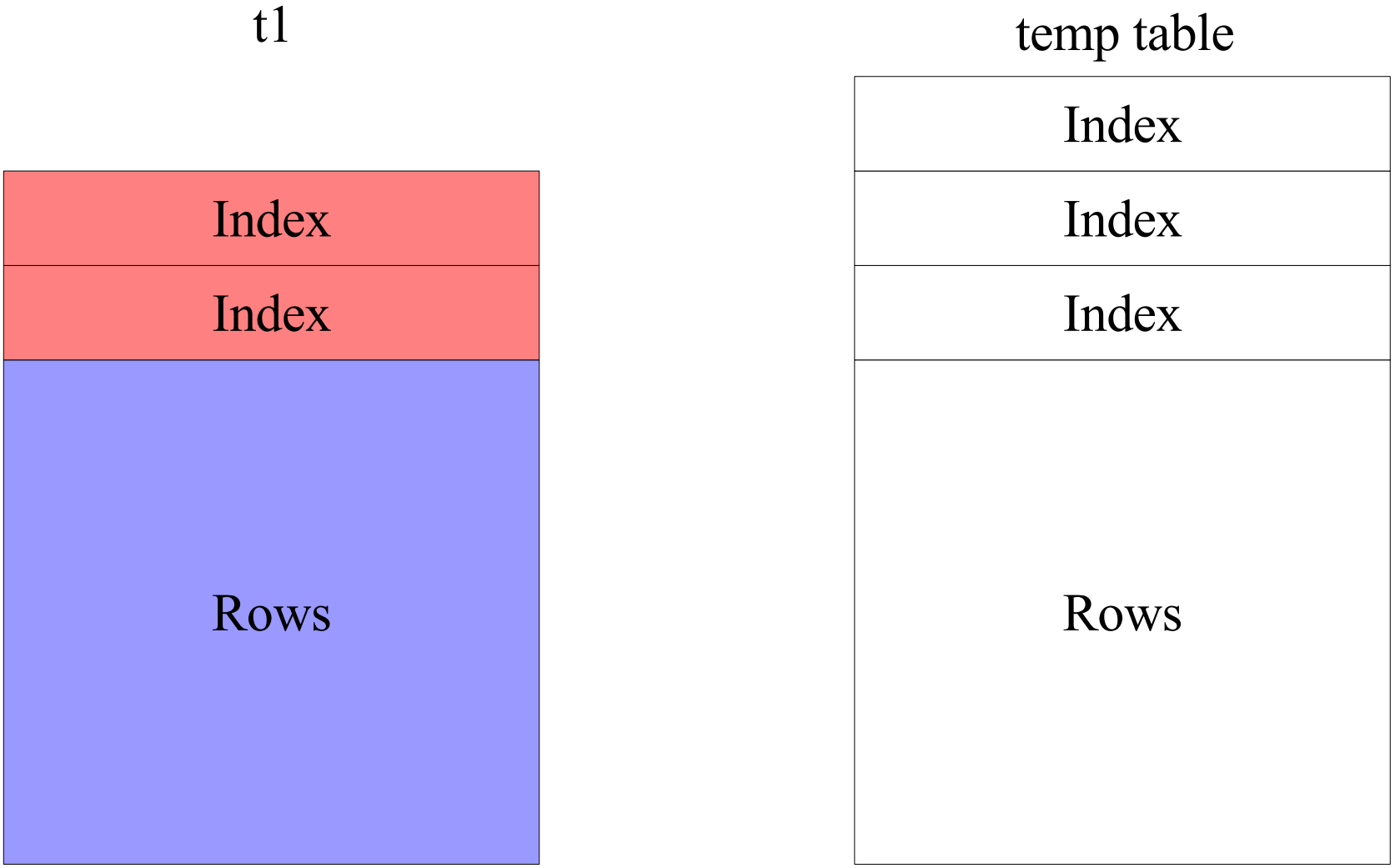
On line Add/Drop Index

ADD INDEX (4.1, 5.0)

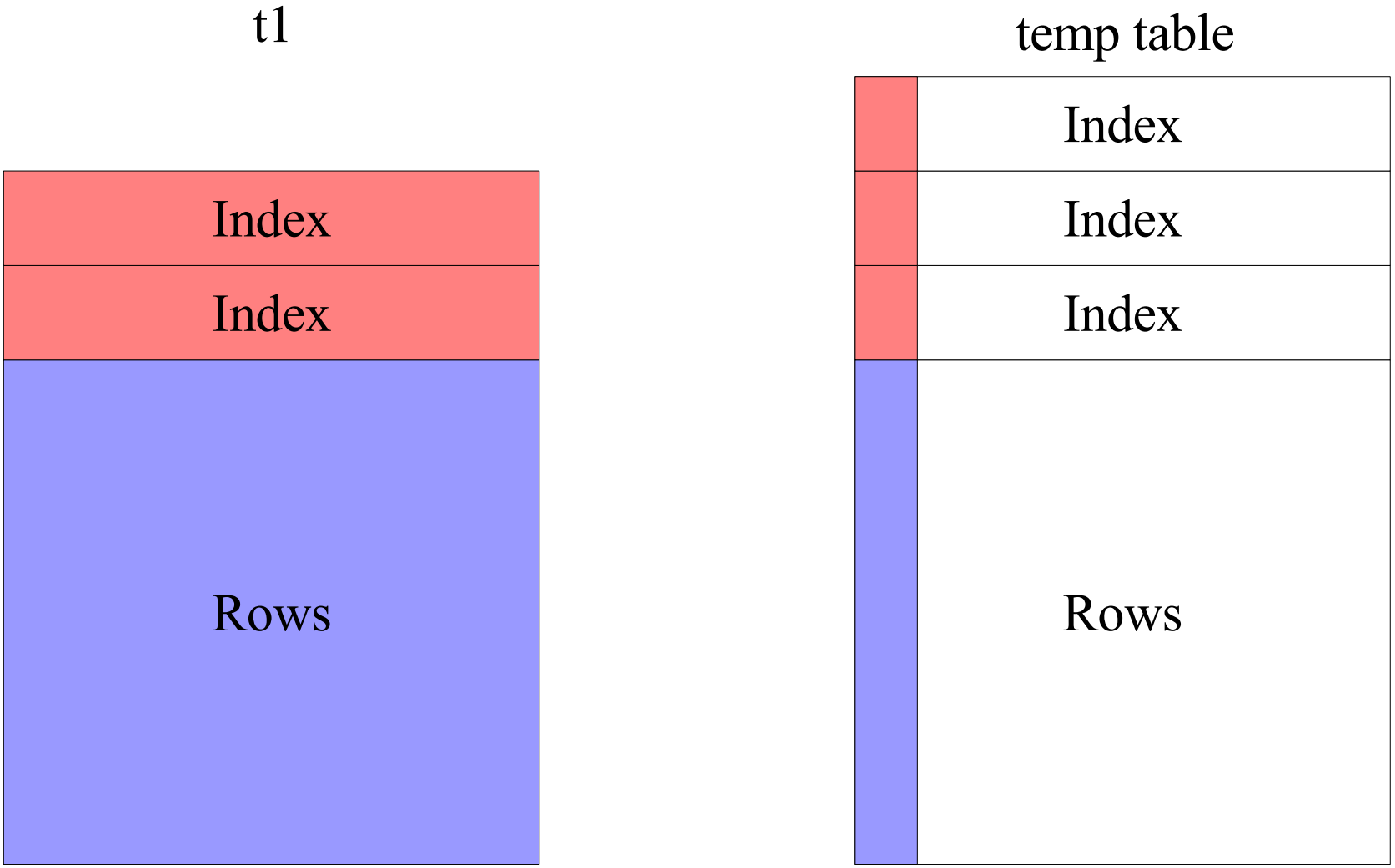
t1



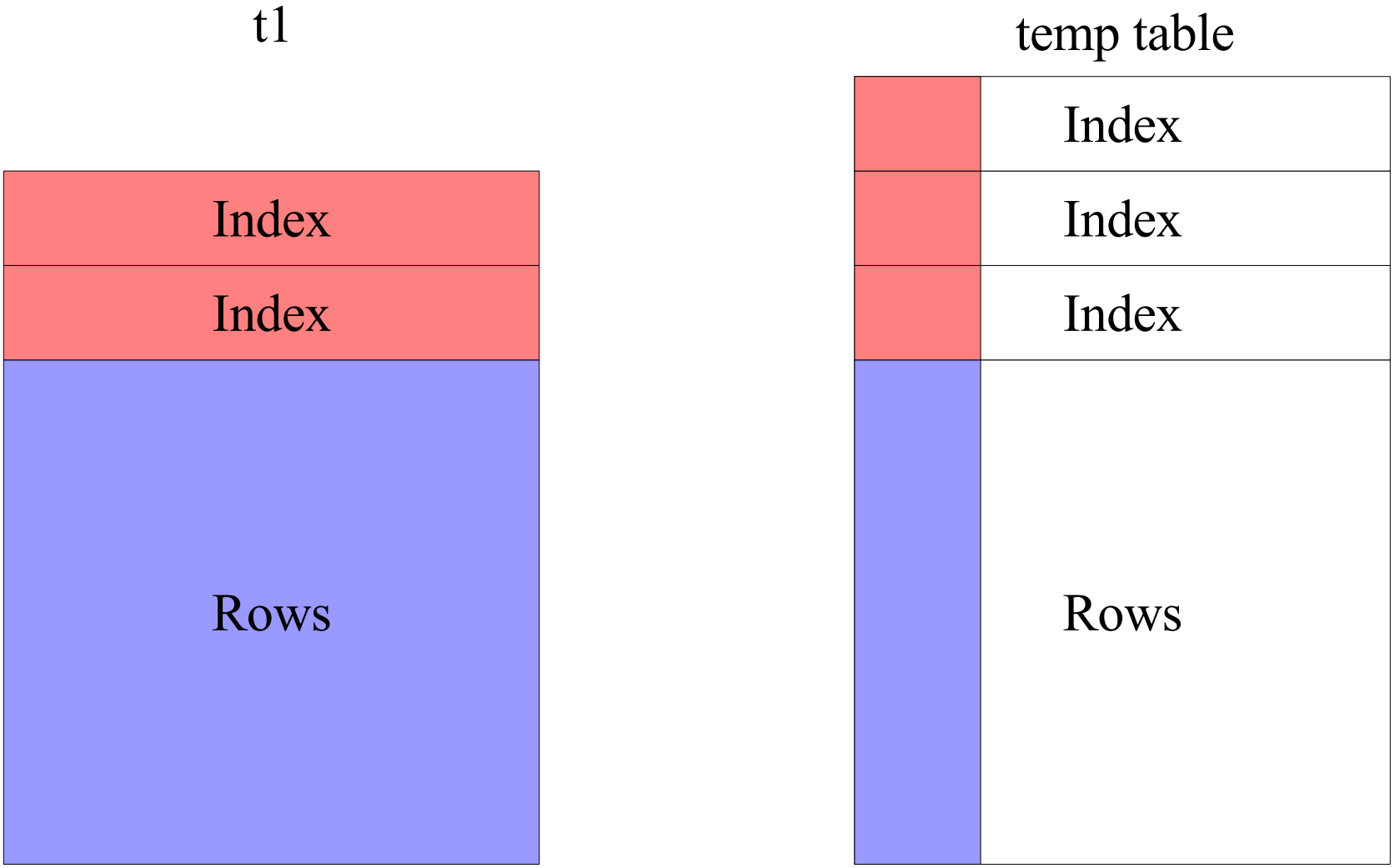
ADD INDEX (4.1, 5.0)



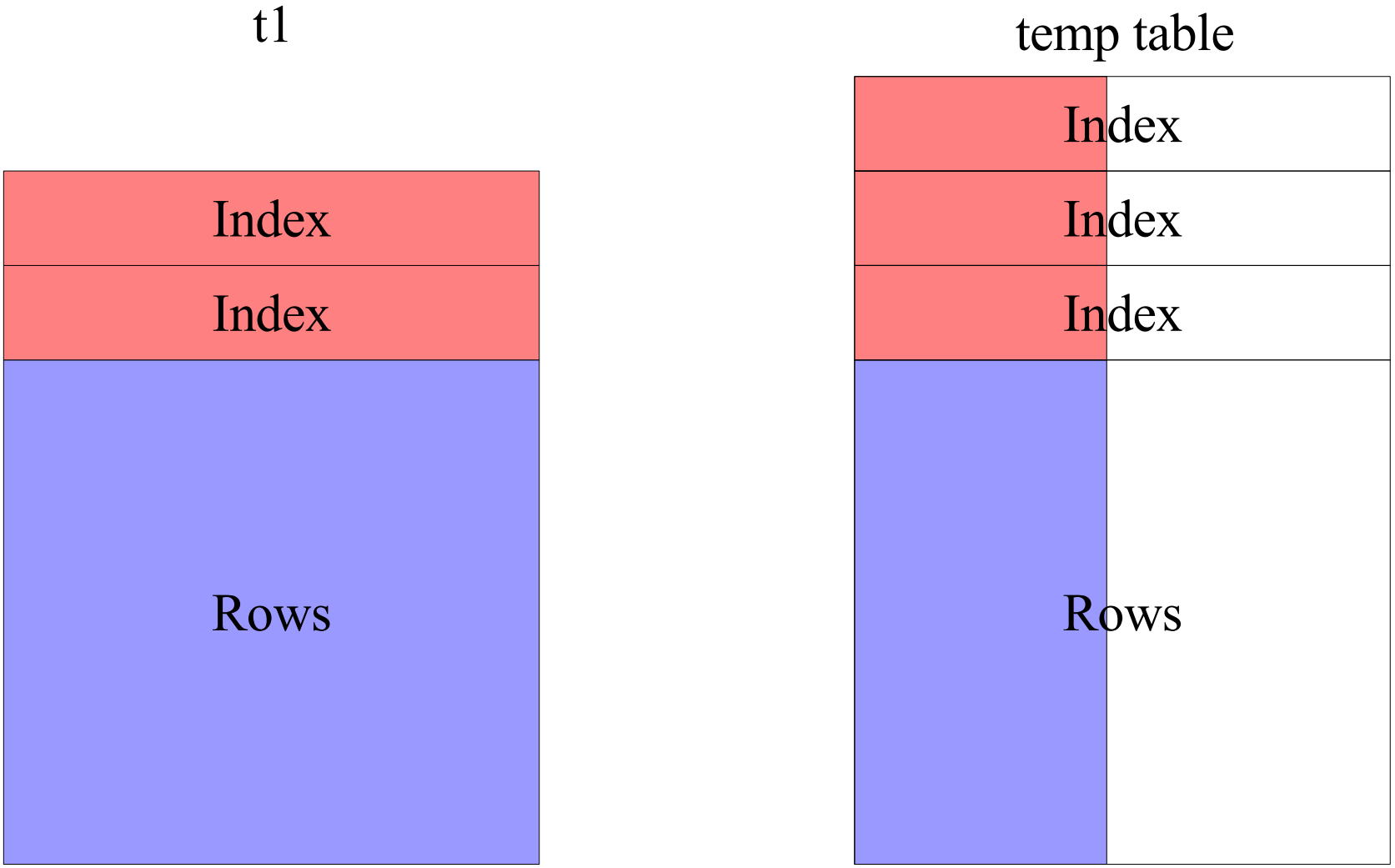
ADD INDEX (4.1, 5.0)



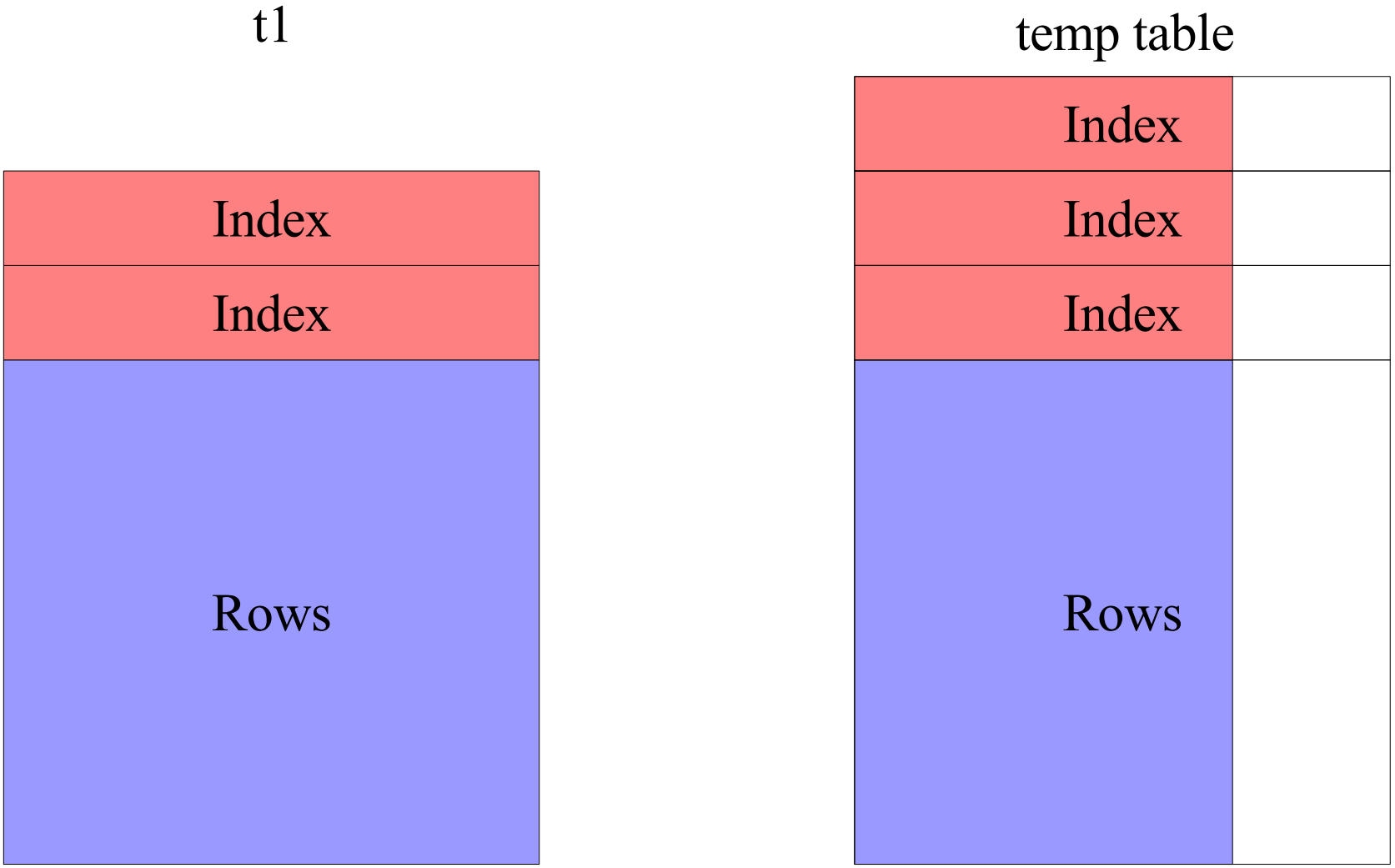
ADD INDEX (4.1, 5.0)



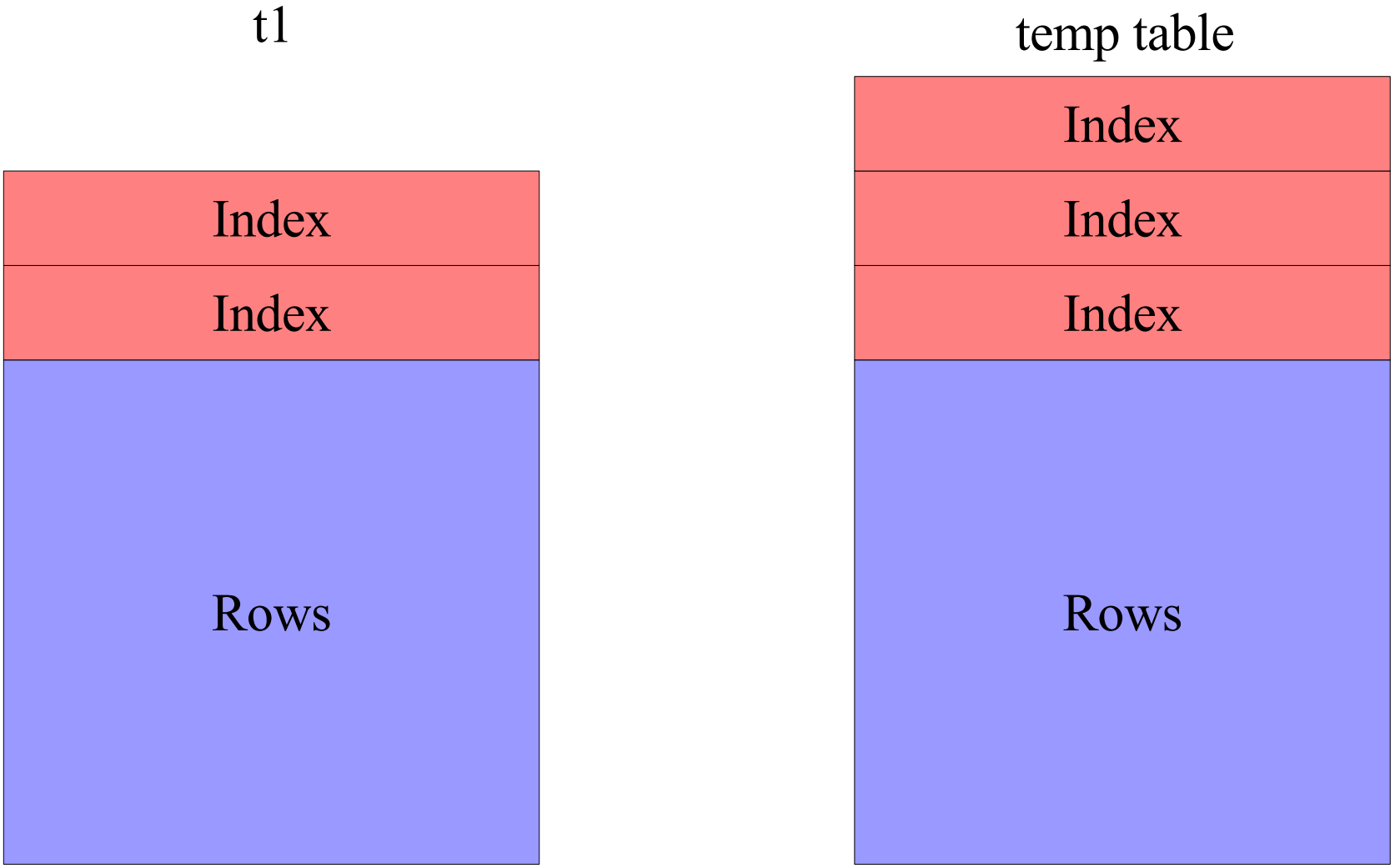
ADD INDEX (4.1, 5.0)



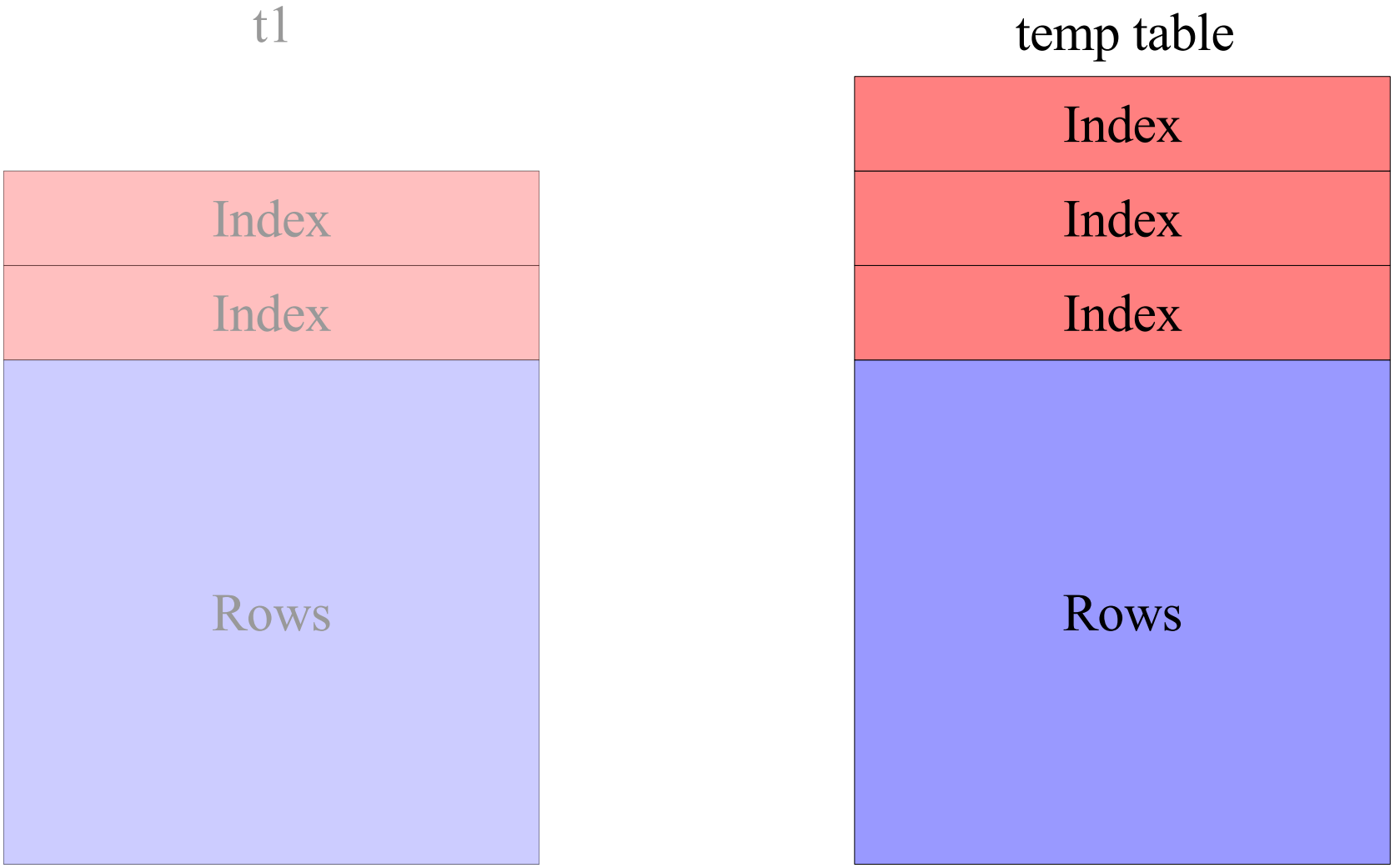
ADD INDEX (4.1, 5.0)



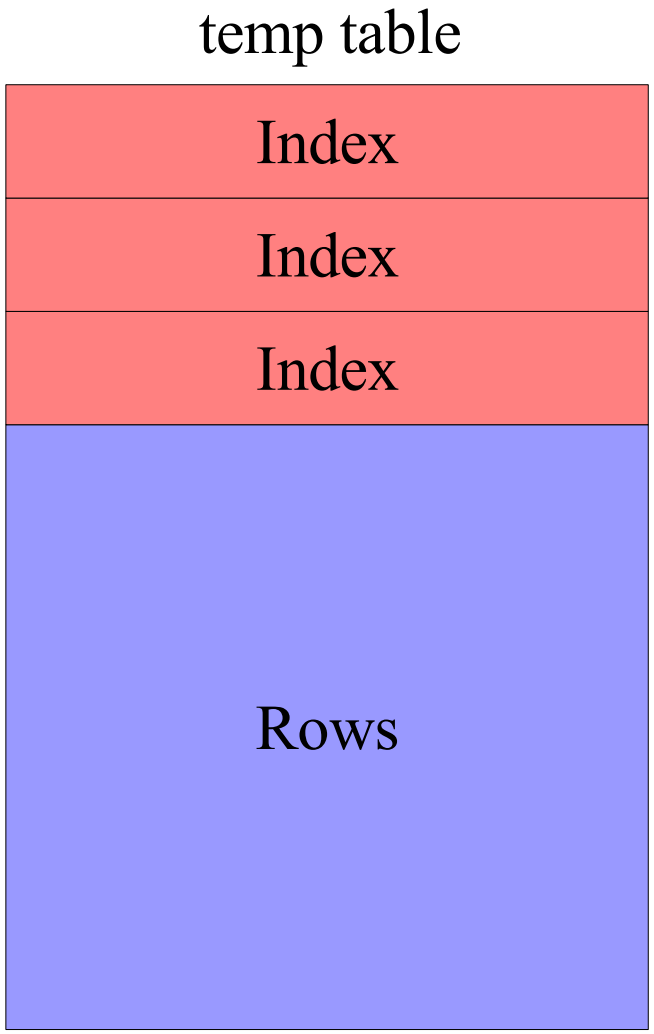
ADD INDEX (4.1, 5.0)



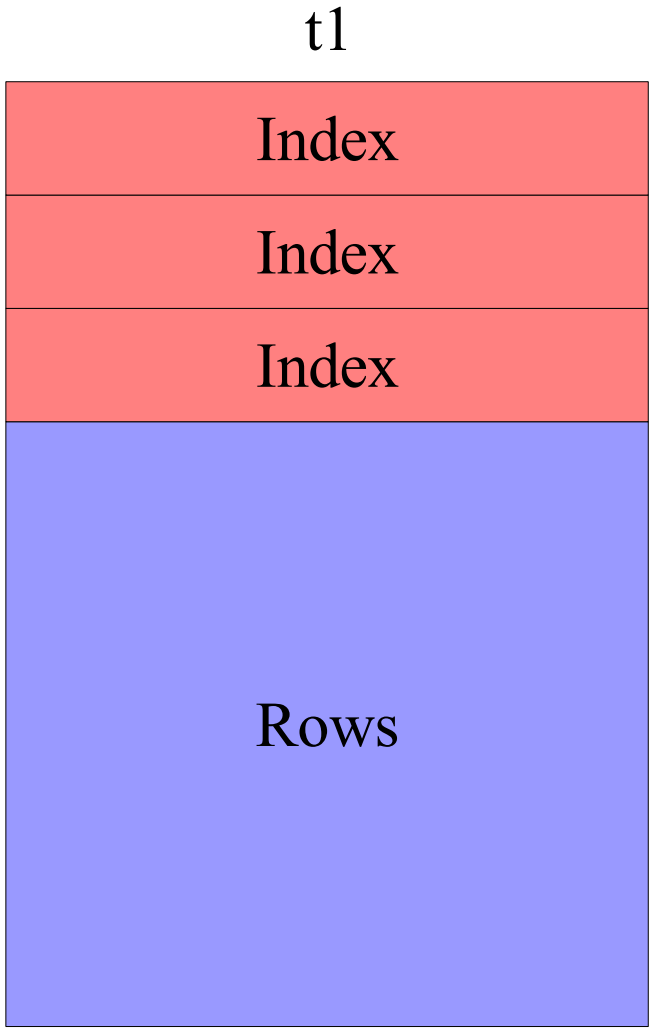
ADD INDEX (4.1, 5.0)



ADD INDEX (4.1, 5.0)

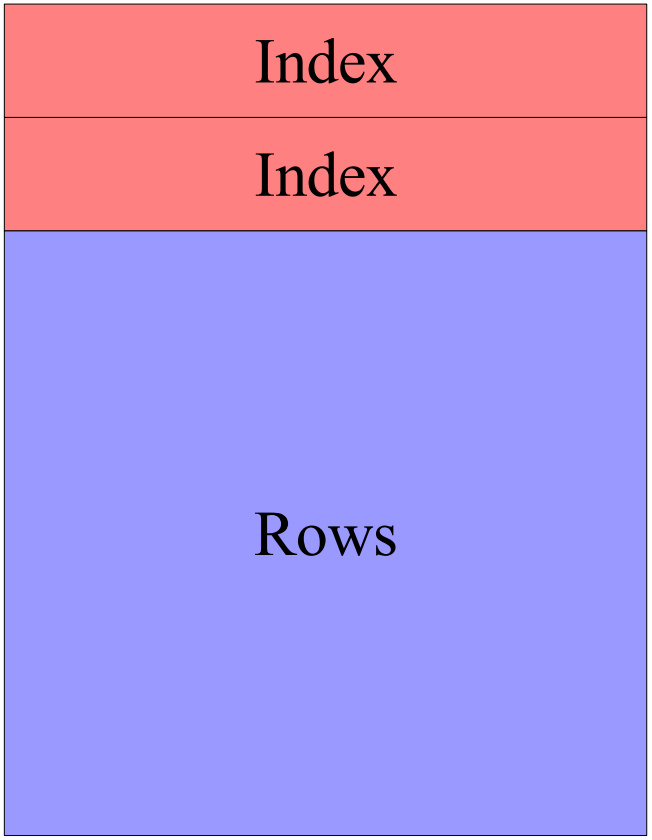


ADD INDEX (4.1, 5.0)

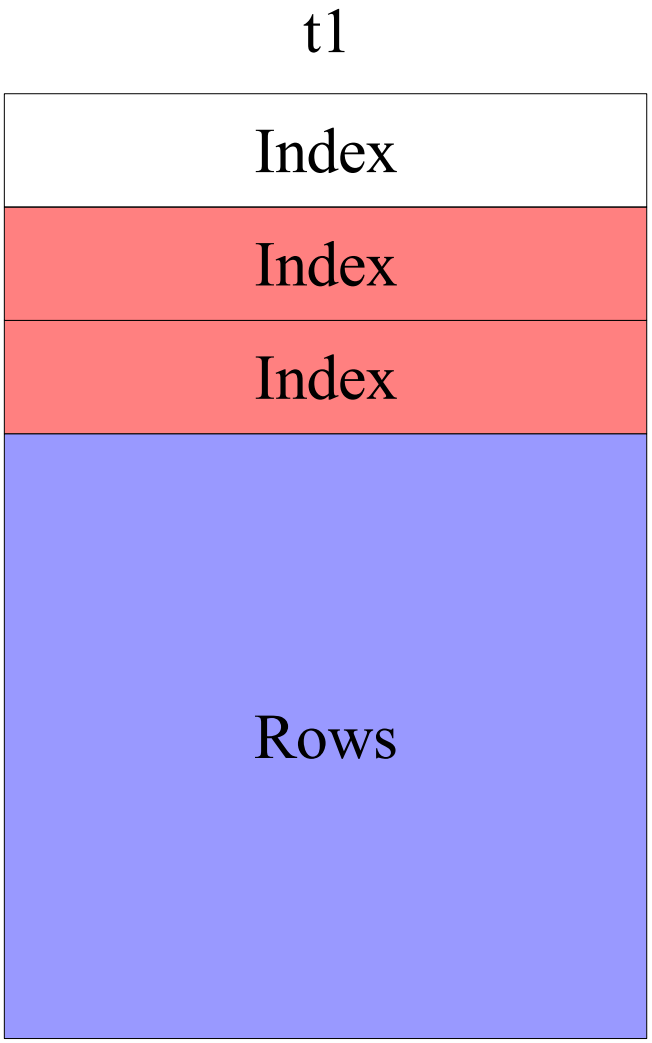


ADD INDEX in 5.1

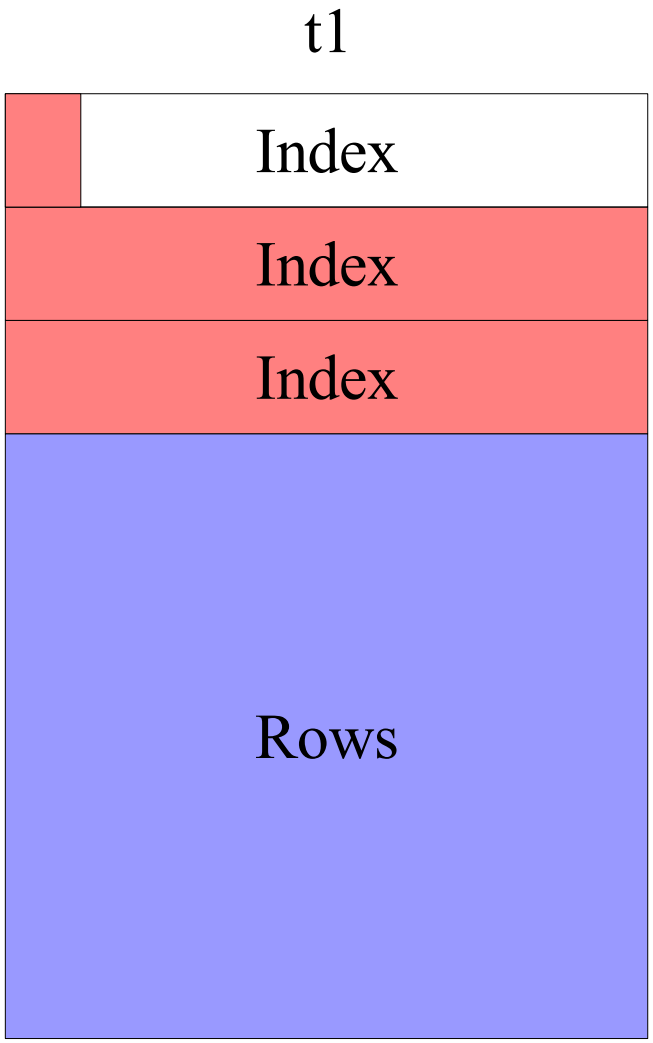
t1



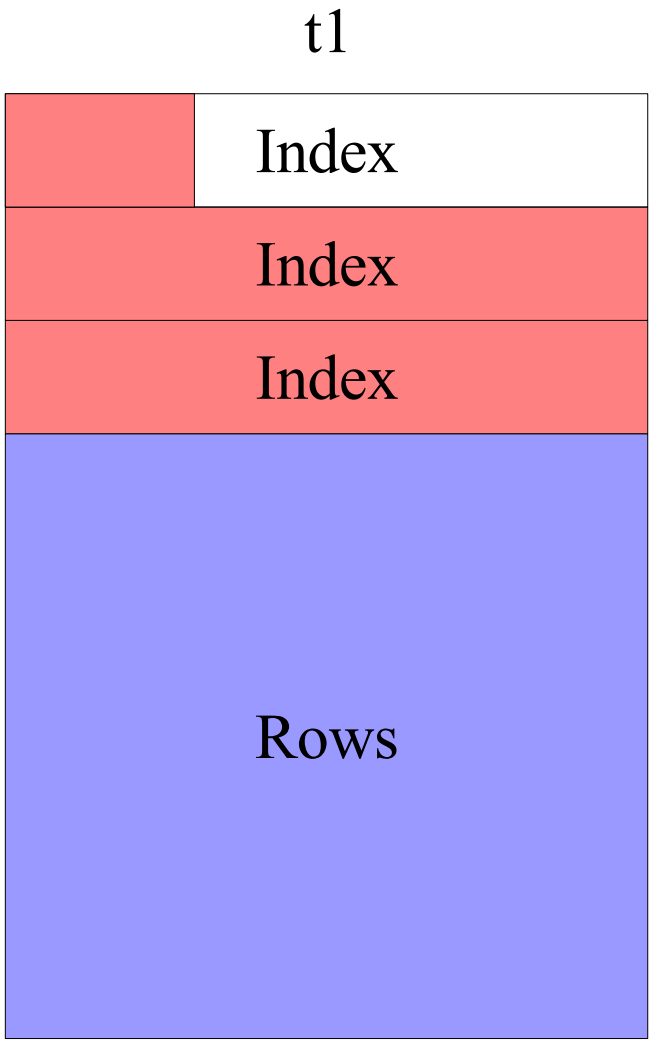
ADD INDEX in 5.1



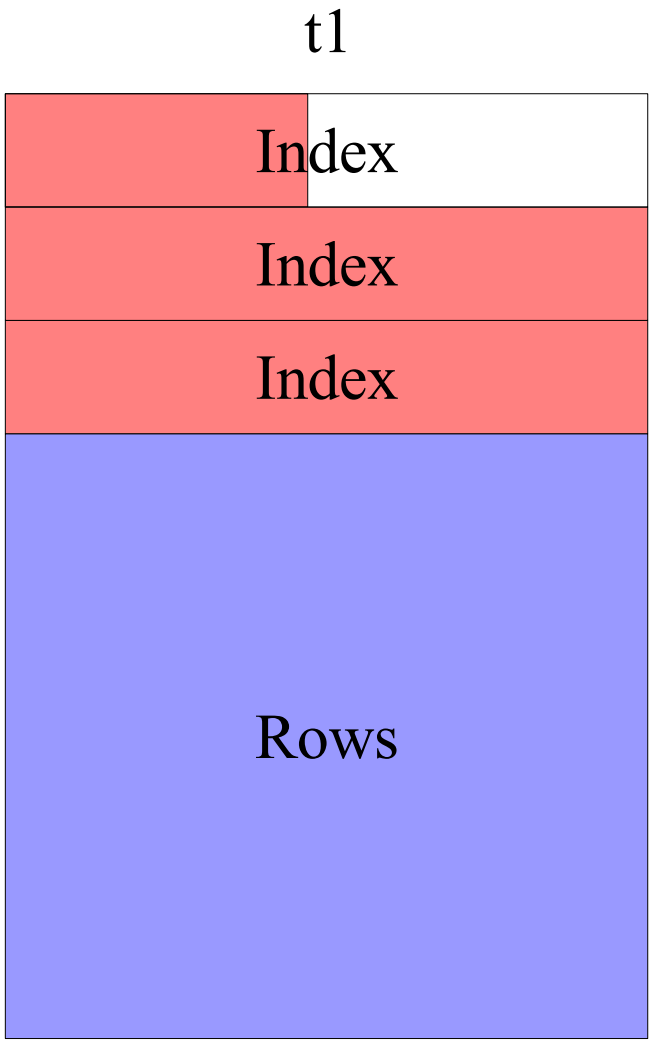
ADD INDEX in 5.1



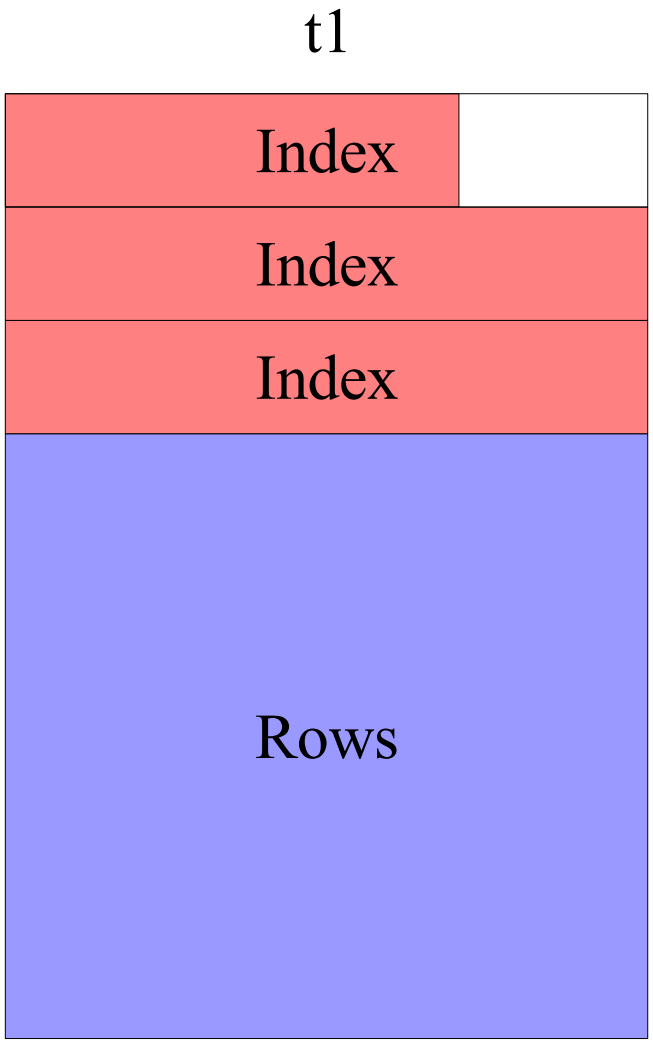
ADD INDEX in 5.1



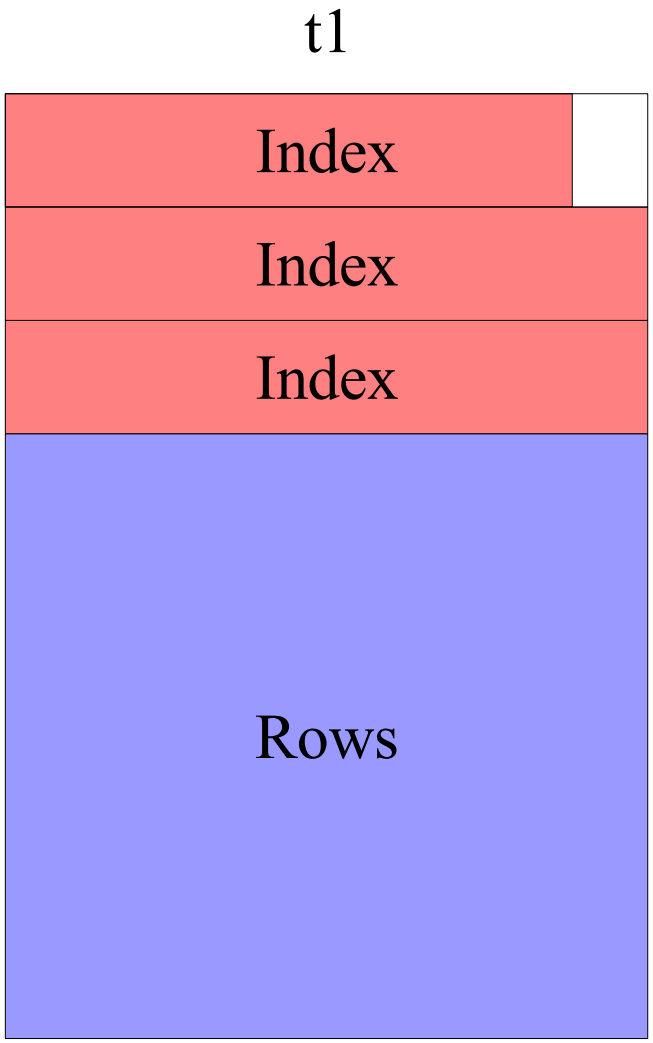
ADD INDEX in 5.1



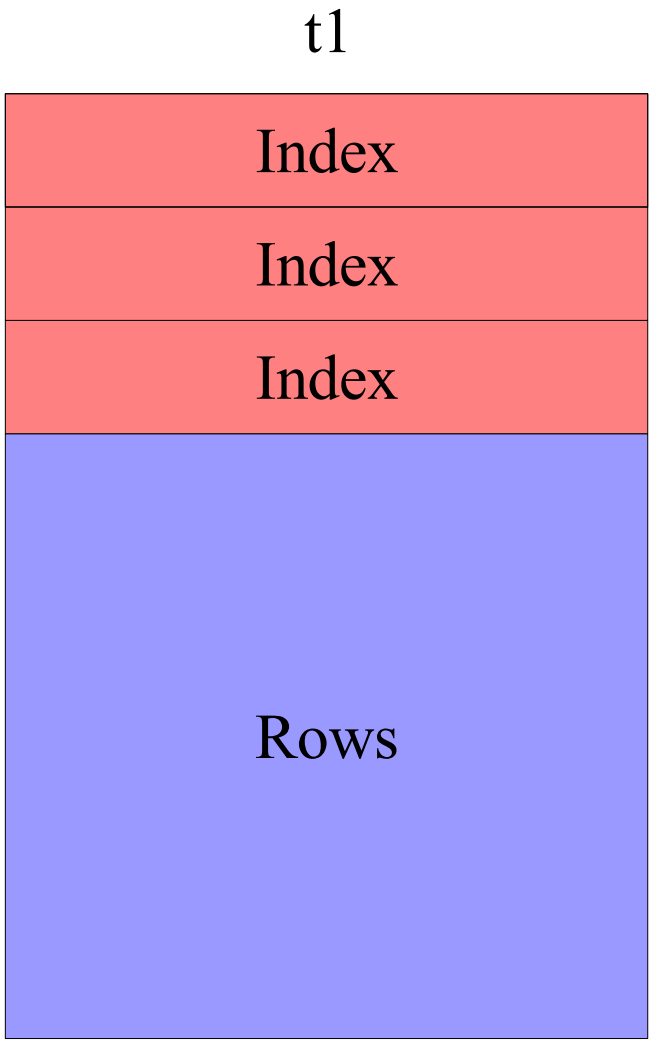
ADD INDEX in 5.1



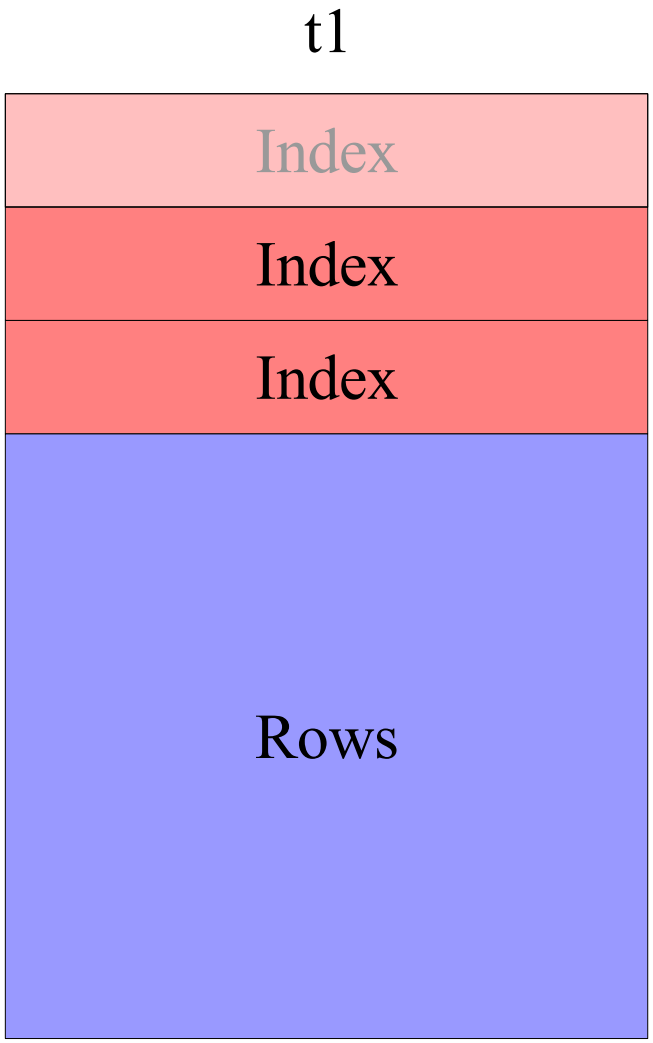
ADD INDEX in 5.1



ADD INDEX in 5.1

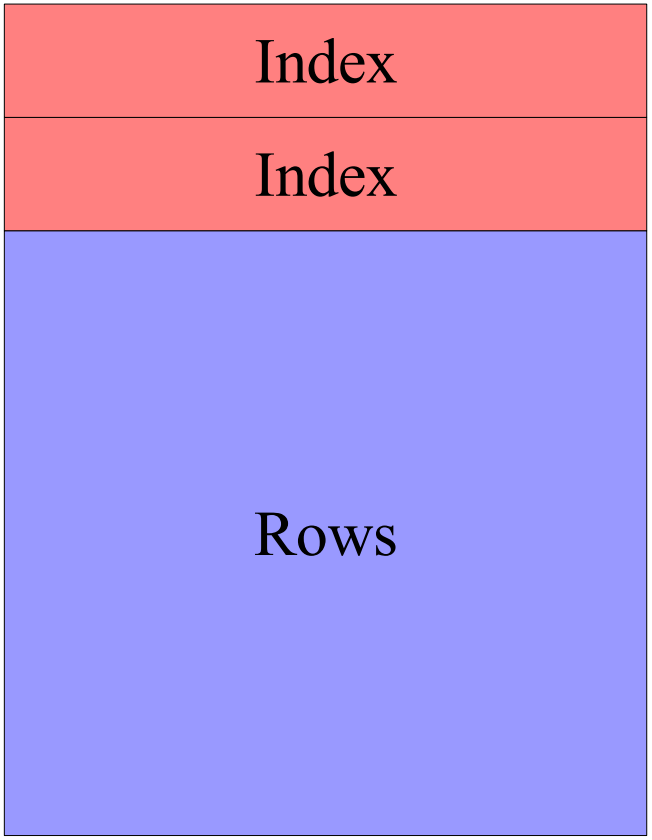


DROP INDEX in 5.1



DROP INDEX in 5.1

t1



How much faster?

Online Add Index

- Before (copy the table):

```
mysql> create index b on t1(b);
Query OK, 1356 rows affected (2.20 sec)
Records: 1356  Duplicates: 0  Warnings: 0

mysql> drop index b on t1;
Query OK, 1356 rows affected (2.03 sec)
Records: 1356  Duplicates: 0  Warnings: 0
```

Online Add Index

- Before (copy the table):

```
mysql> create index b on t1(b);
Query OK, 1356 rows affected (2.20 sec)
Records: 1356  Duplicates: 0  Warnings: 0

mysql> drop index b on t1;
Query OK, 1356 rows affected (2.03 sec)
Records: 1356  Duplicates: 0  Warnings: 0
```

- Now (just add/drop an index):

```
mysql> create index b on t1(b);
Query OK, 0 rows affected (0.58 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> drop index b on t1;
Query OK, 0 rows affected (0.46 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

On line Add/Drop Index

Faster Add/Drop Index

Less memory requirements

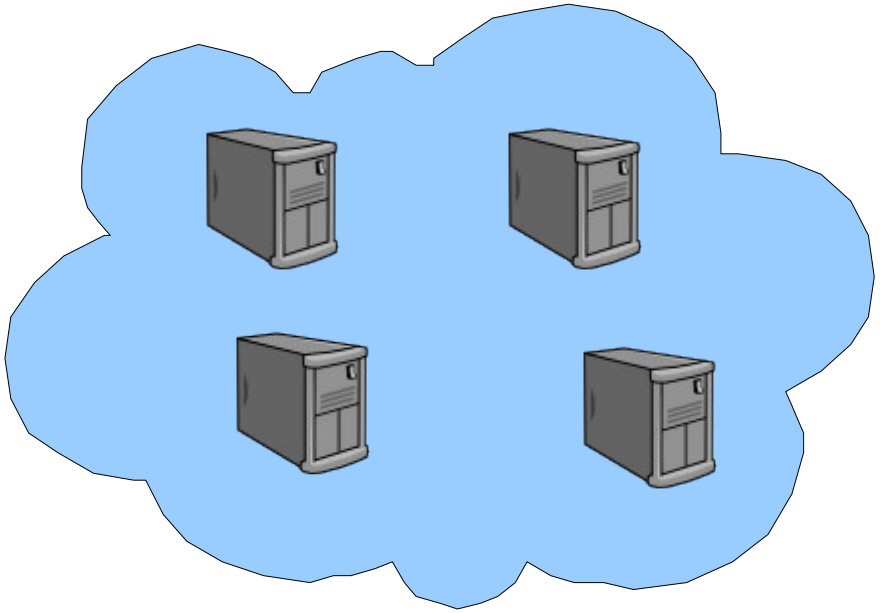
A more adaptive Cluster

User Defined Partitioning

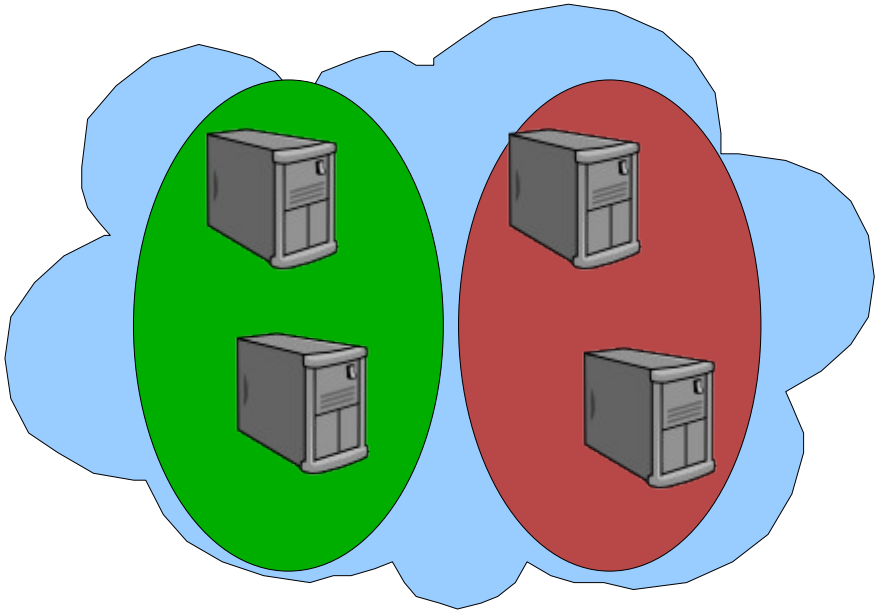
Since the dawn of time...

pk

pk

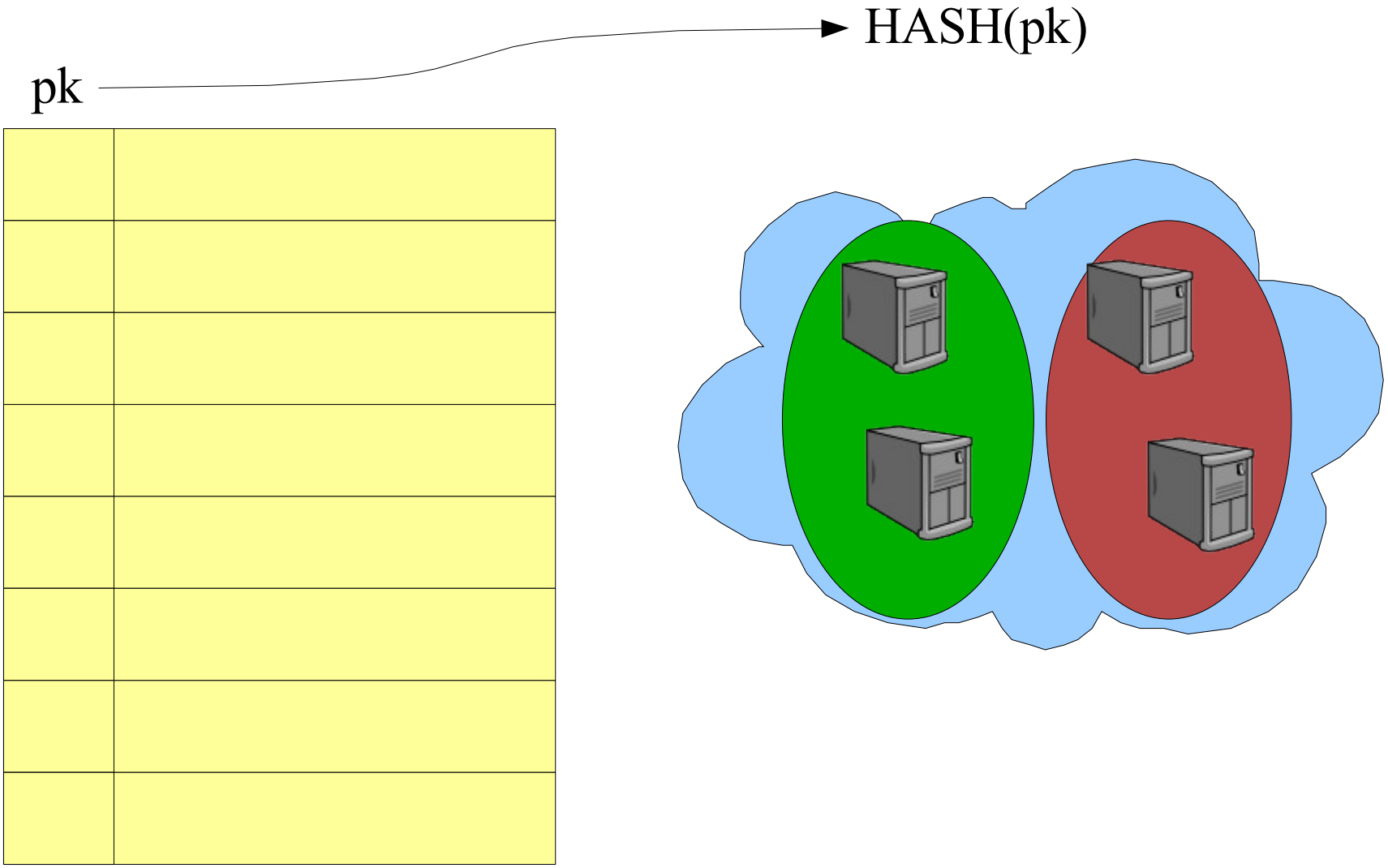


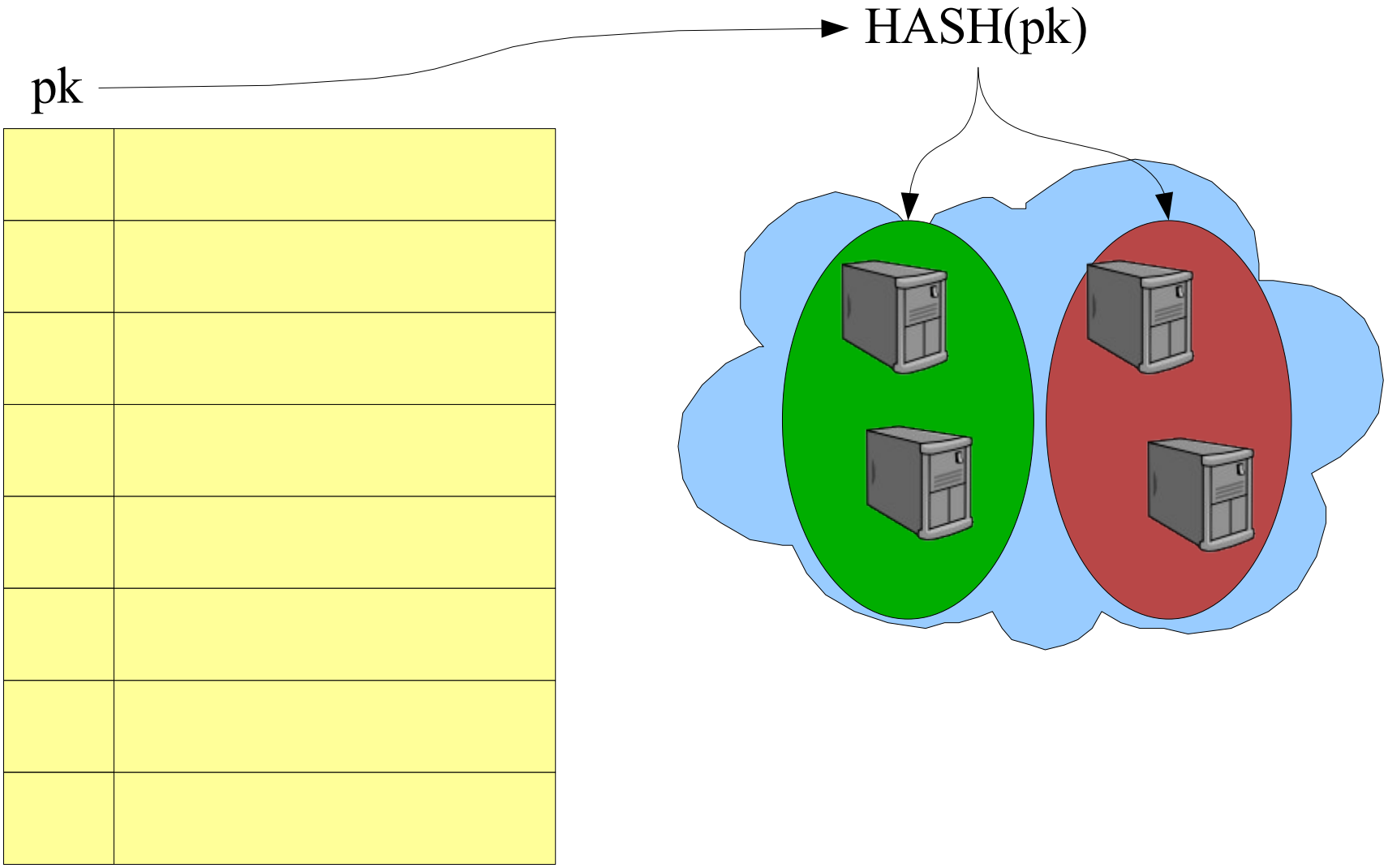
pk

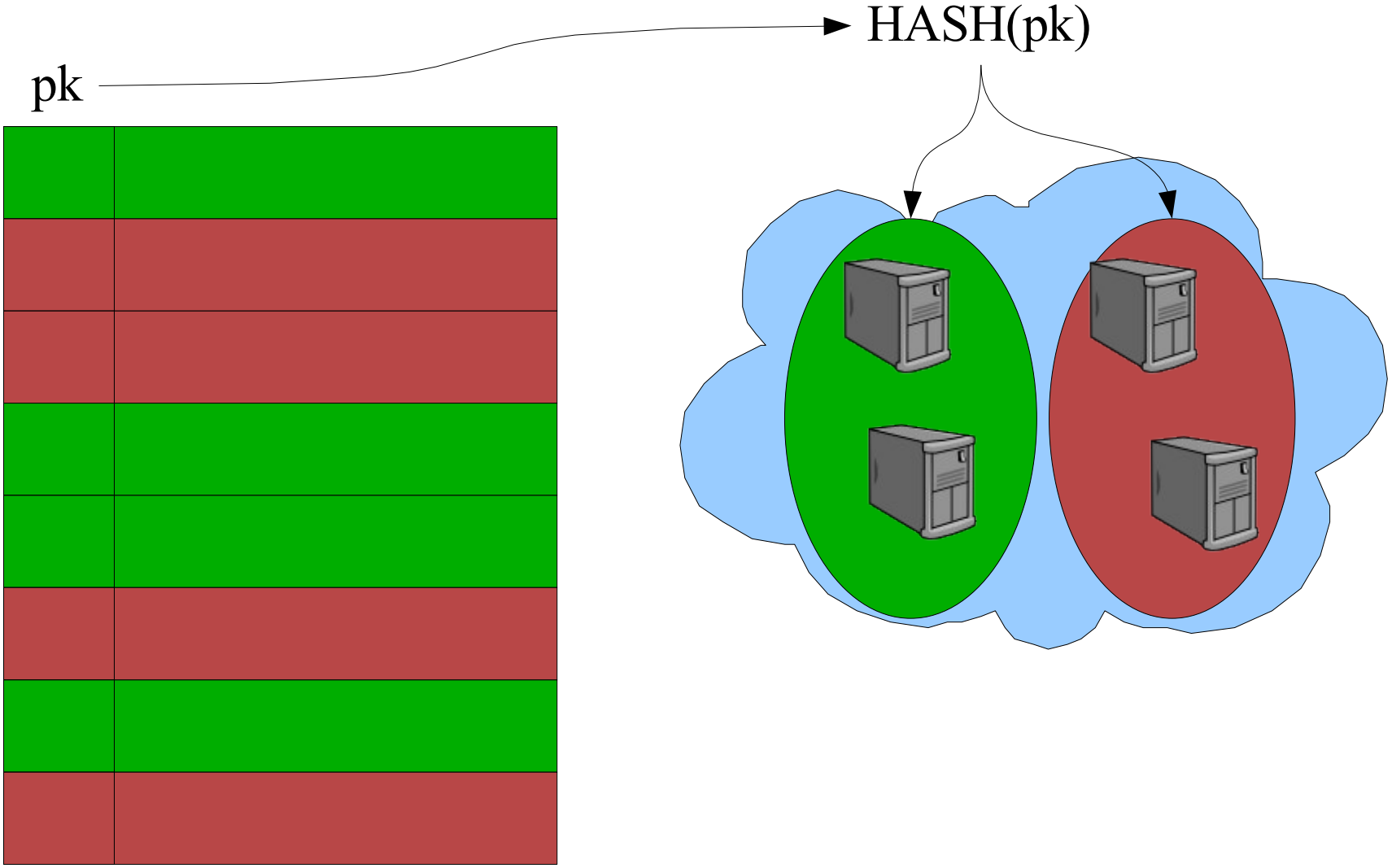


Nodegroup 0

Nodegroup 1

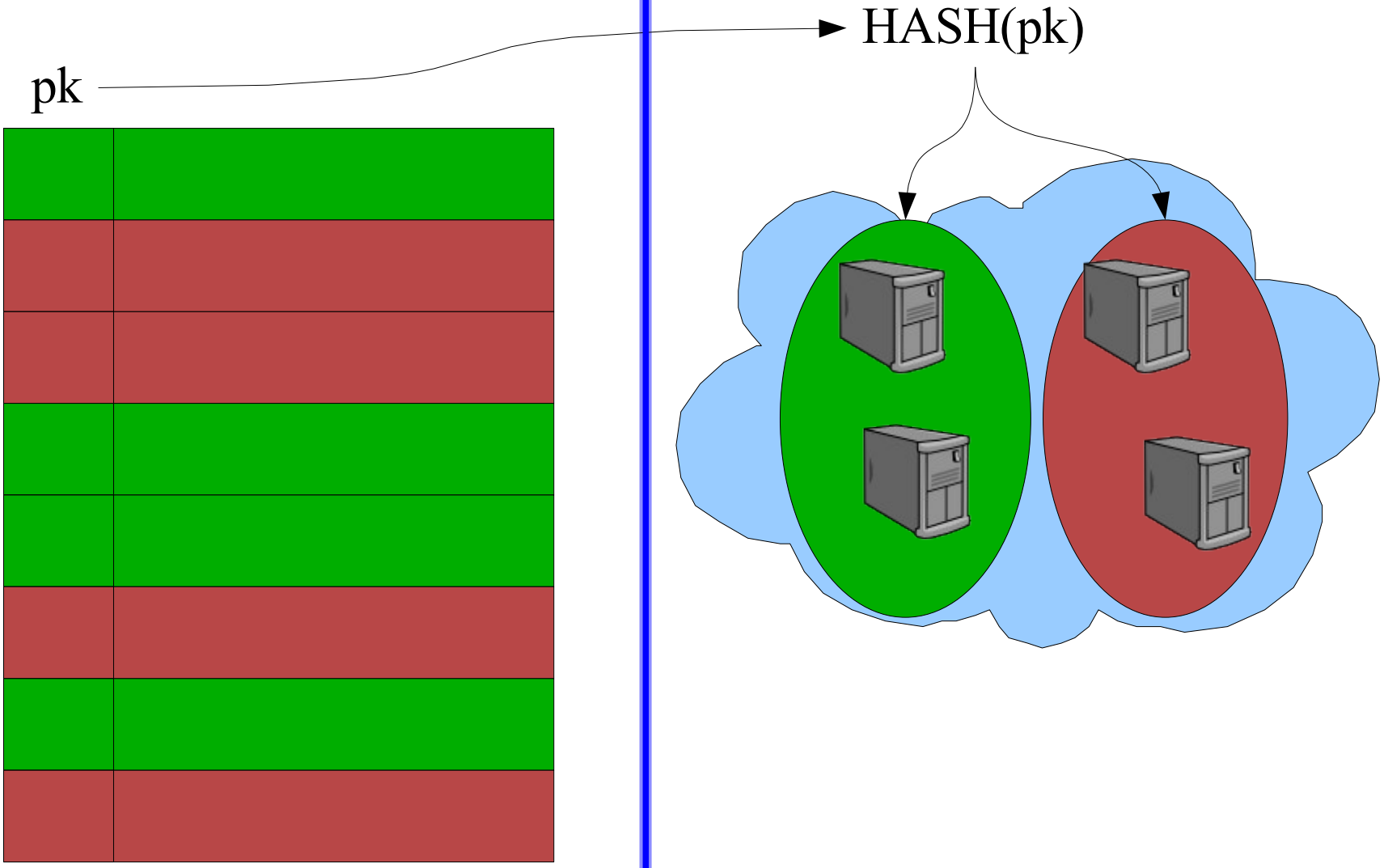


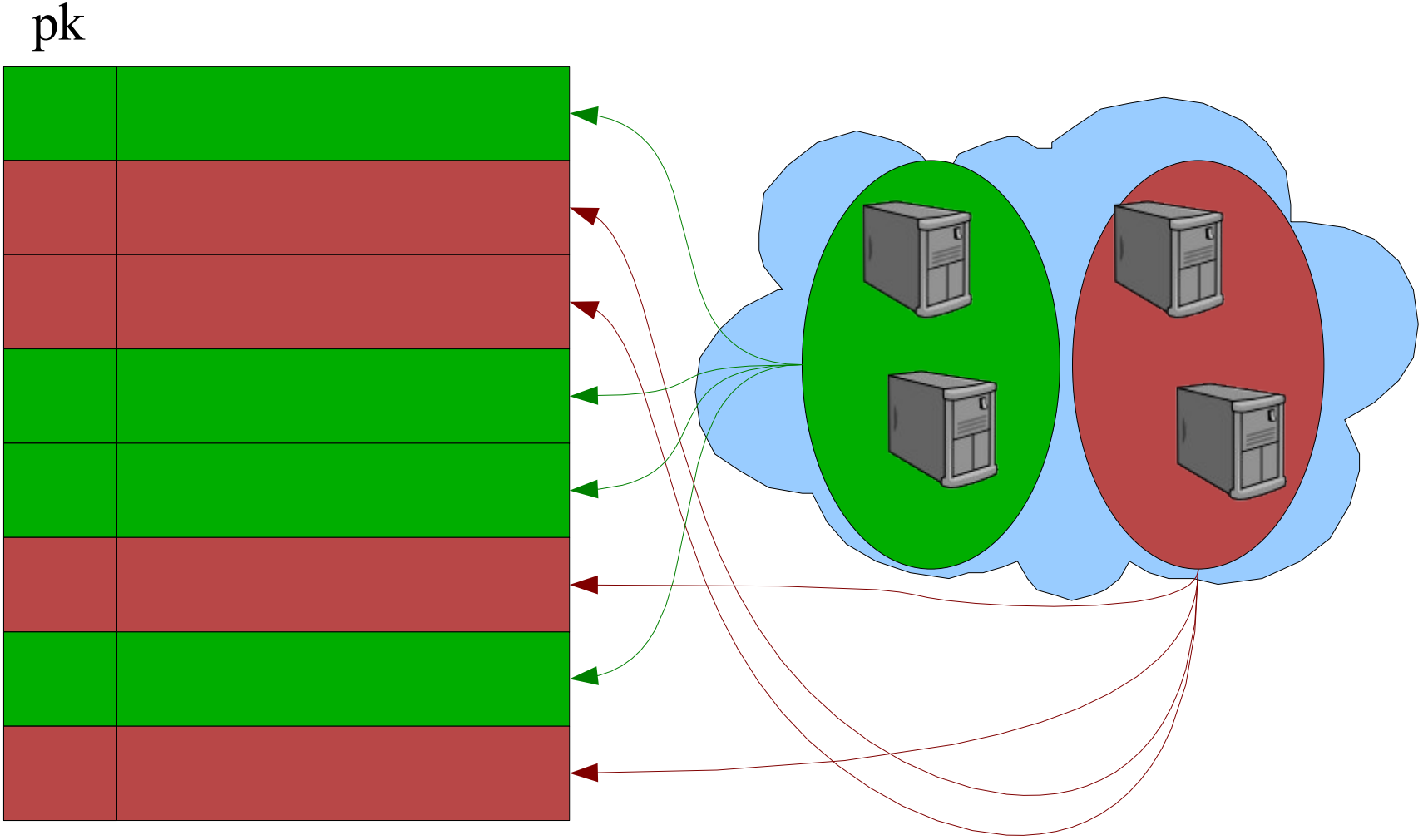




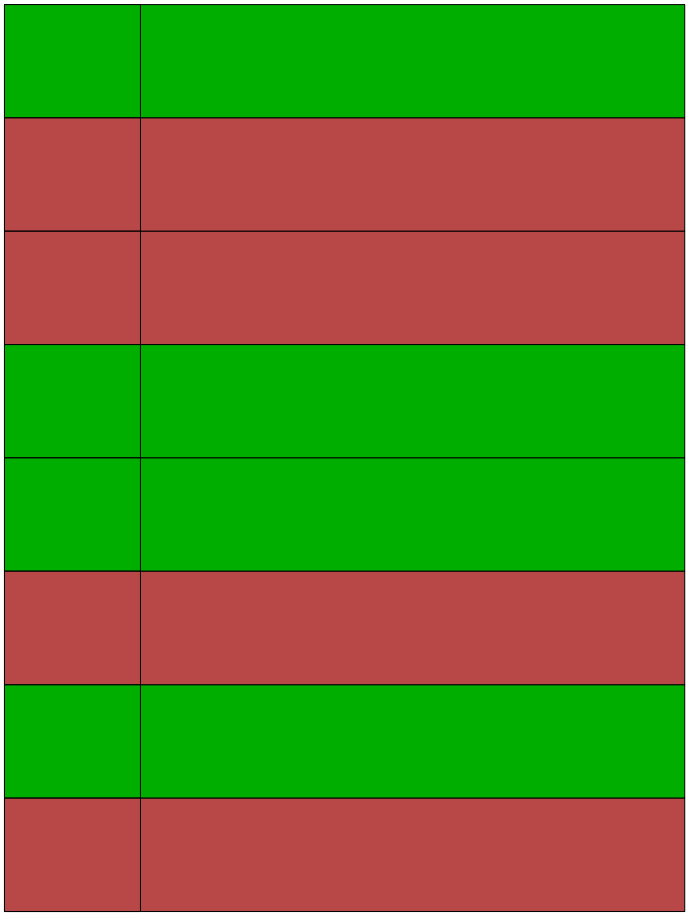
Perception

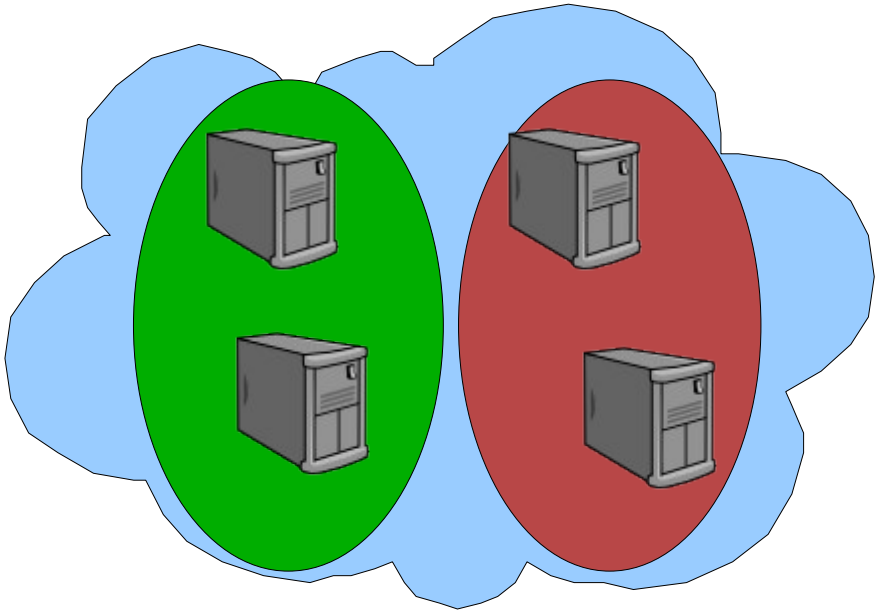
Reality





pk





Two Partitions

How the default looks (in 5.1 SHOW CREATE TABLE)

```
CREATE TABLE account(number int unsigned,
                     location varchar,
                     amount int)
PRIMARY KEY (number)
[PARTITION BY KEY ()]
[(PARTITION P0 NODEGROUP 0, PARTITION P1 NODEGROUP 1,
  …)]
ENGINE=NDBCLUSTER;
```

How the default looks (in 5.1 SHOW CREATE TABLE)

```
CREATE TABLE account(number int unsigned,
                     location varchar,
                     amount int)
PRIMARY KEY (number)
[PARTITION BY KEY ()]
[(PARTITION P0 NODEGROUP 0, PARTITION P1 NODEGROUP 1,
  …)]
ENGINE=NDBCLUSTER;
```

How the default looks (in 5.1 SHOW CREATE TABLE)

```
CREATE TABLE account(number int unsigned,
                     location varchar,
                     amount int)
PRIMARY KEY (number)
[PARTITION BY KEY ()]
[(PARTITION P0 NODEGROUP 0, PARTITION P1 NODEGROUP 1,
  …)]
ENGINE=NDBCLUSTER;
```

How the default looks (in 5.1 SHOW CREATE TABLE)

```
CREATE TABLE account(number int unsigned,
                     location varchar,
                     amount int)
PRIMARY KEY (number)
[PARTITION BY KEY ()]
[(PARTITION P0 NODEGROUP 0, PARTITION P1 NODEGROUP 1,
  …)]
ENGINE=NDBCLUSTER;
```

Now, In 5.1

User Defined Partitioning

By Key

Partition by Key

```
CREATE TABLE account(number int unsigned,
                     location varchar,
                     amount int)
PRIMARY KEY (number)
[PARTITION BY KEY ()]
[(PARTITION P0 NODEGROUP 0, PARTITION P1 NODEGROUP 1,
  …)]
ENGINE=NDBCLUSTER;
```

By Range

Partition by Range

```
CREATE TABLE account(number int unsigned,
                     location varchar,
                     amount int)
  PRIMARY KEY (number)
  PARTITION BY RANGE (amount)
  PARTITIONS 2
  (PARTITION P0 VALUES LESS THAN 10000
   NODEGROUP 0,
   PARTITION P1 VALUES LESS THAN 100000
   NODEGROUP 1)
  ENGINE=NDBCLUSTER;
```

or by List

Partition by List

```
CREATE TABLE account(number int unsigned,
                     location varchar,
                     amount int)
  PRIMARY KEY (number)
  PARTITION BY LIST (ORD(location))
  PARTITIONS 2
  (PARTITION P0 VALUES IN (66, 76) -- Berlin, London
   NODEGROUP 0,
   PARTITION P1 VALUES IN (78, 84) -- New York, Tokyo
   NODEGROUP 1)
  ENGINE=NDBCLUSTER;
```

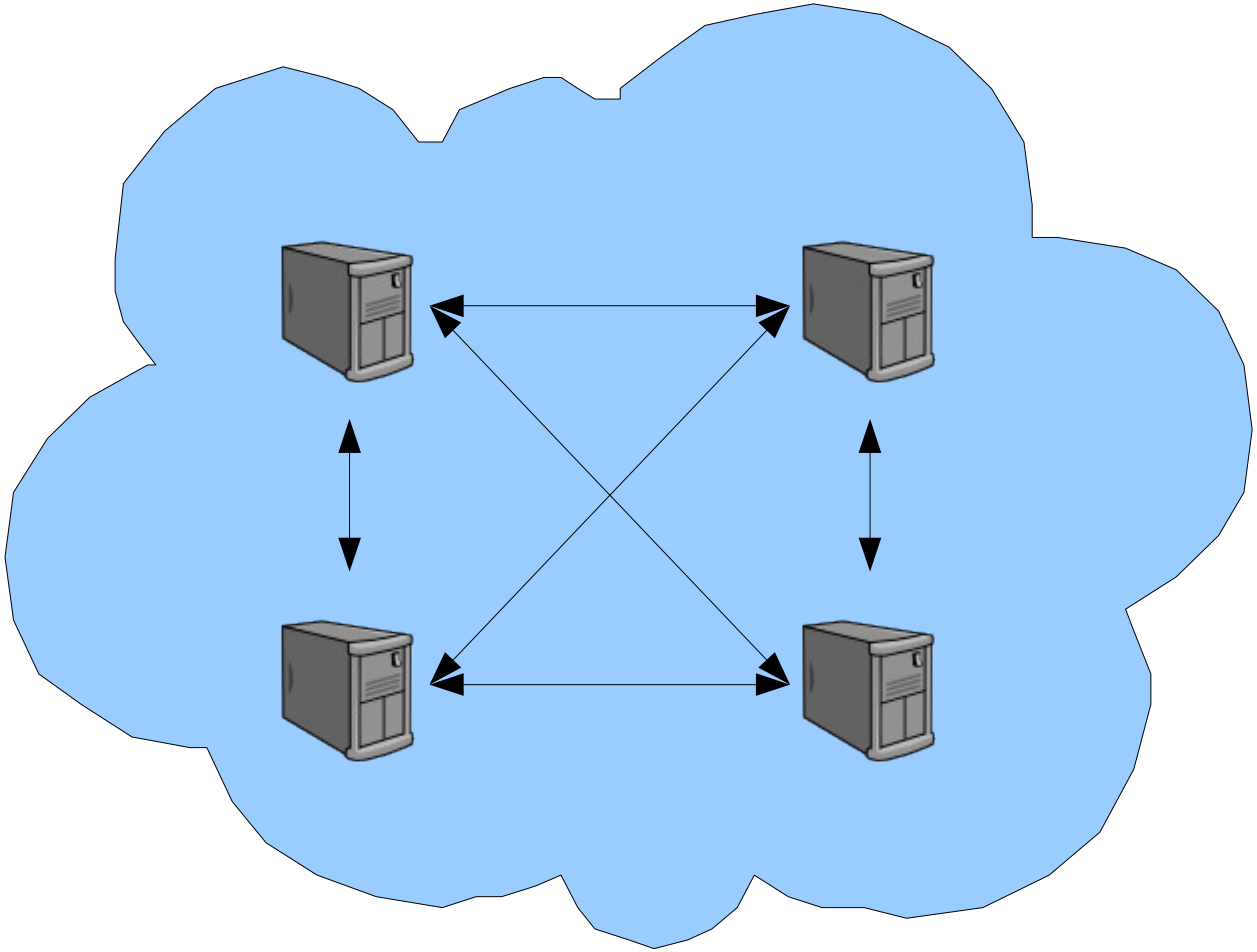
User Defined Partitioning

Gives the User control

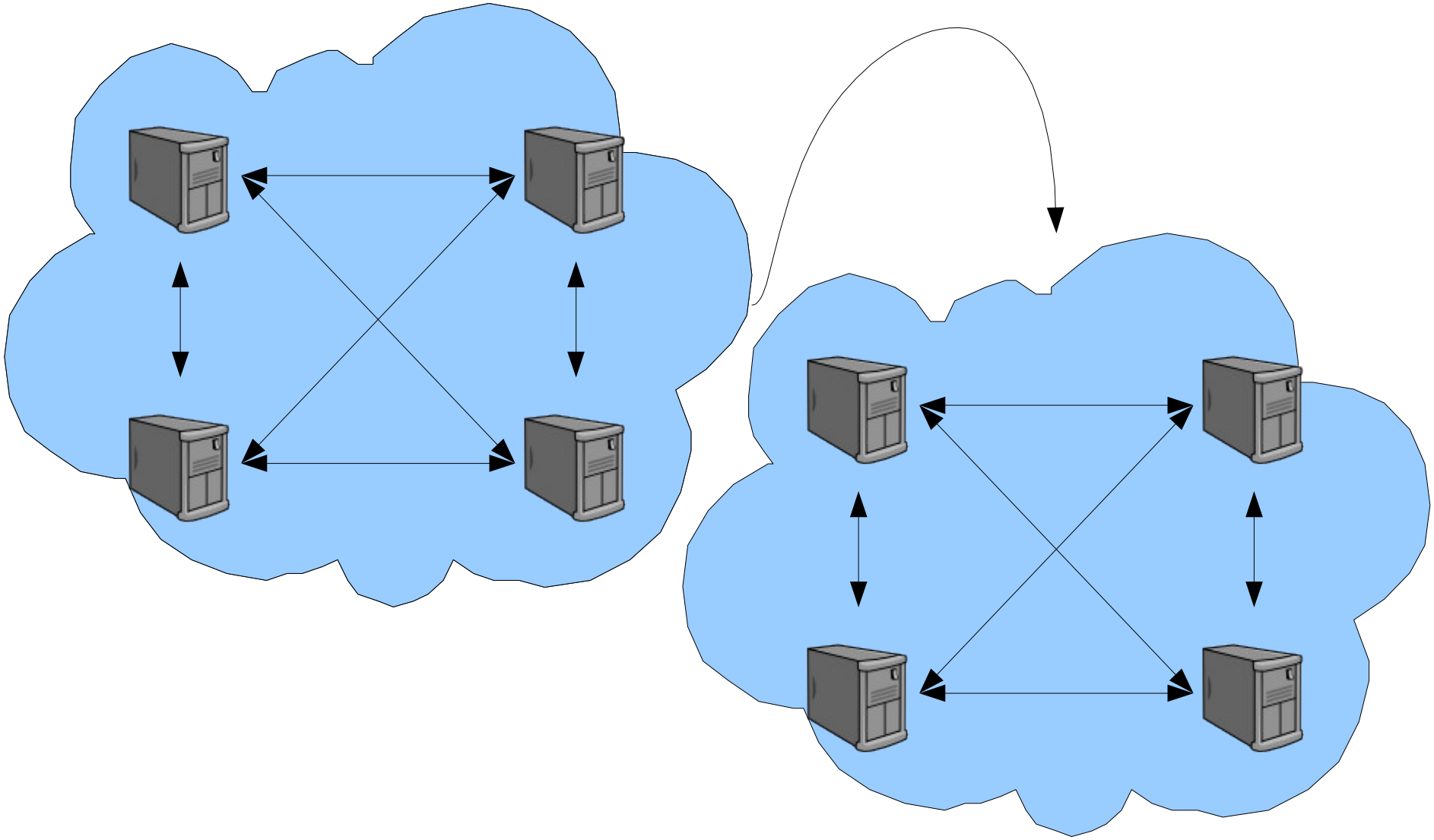
Gives the User control over physical distribution of data

MySQL Cluster Replication

Not Internal Mirroring between nodes



Replication from one Cluster to Another Cluster



Why?

the usual reasons

Why replicate between Clusters?

- Geographical Redundancy

Why replicate between Clusters?

- Geographical Redundancy
- Split the processing load

Why replicate between Clusters?

- Geographical Redundancy
- Split the processing load
  - e.g. for monthly reports

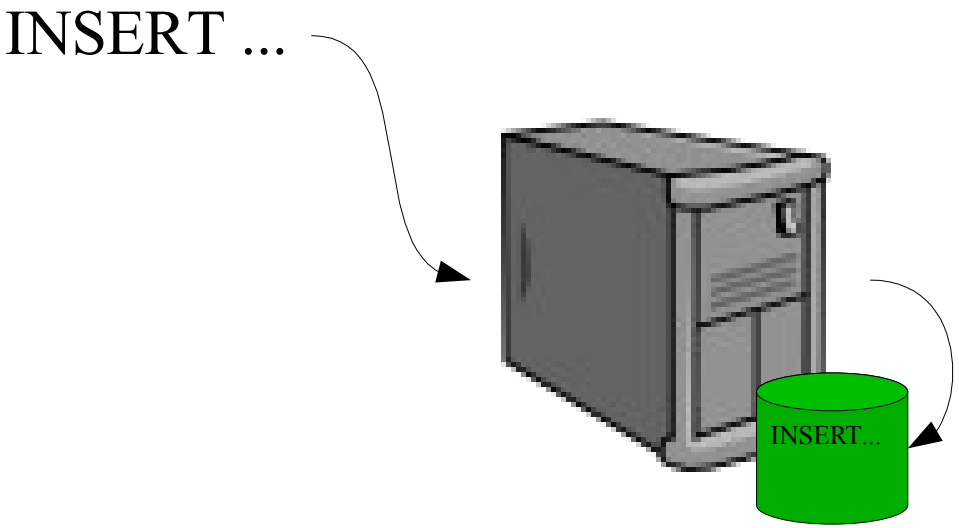
Quick Overview of MySQL Replication

MySQL Replication

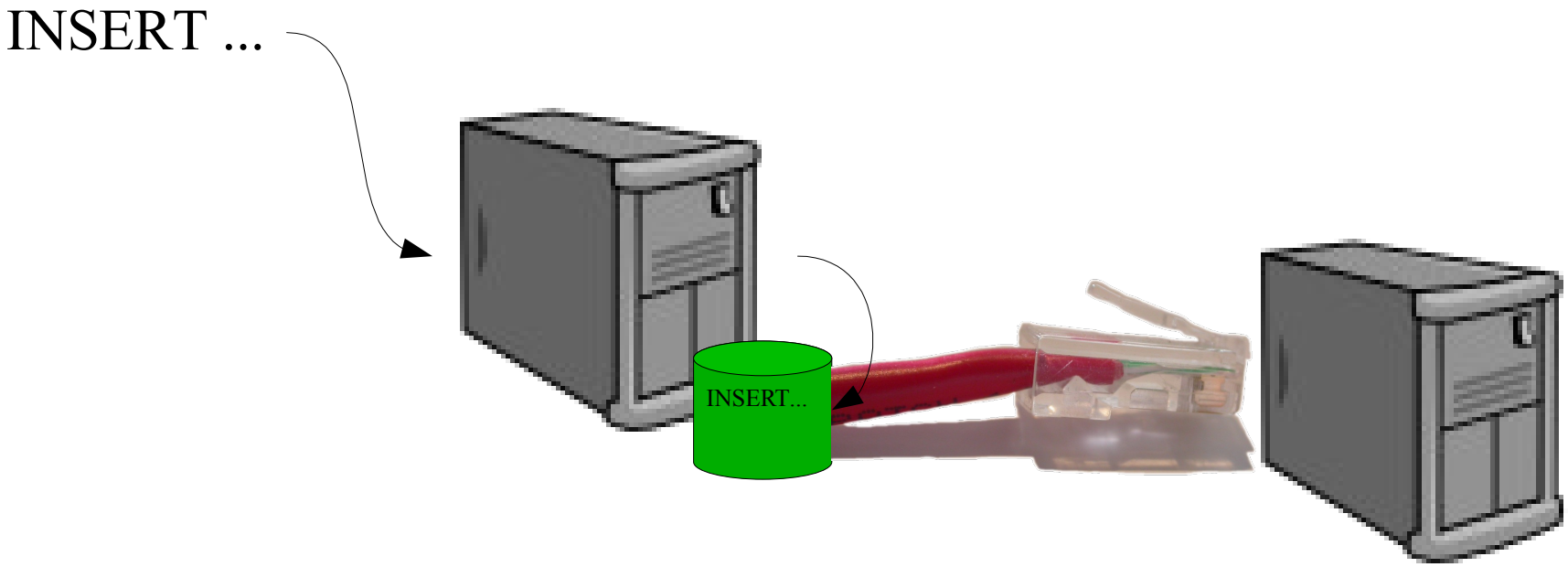
MySQL Replication

INSERT ...

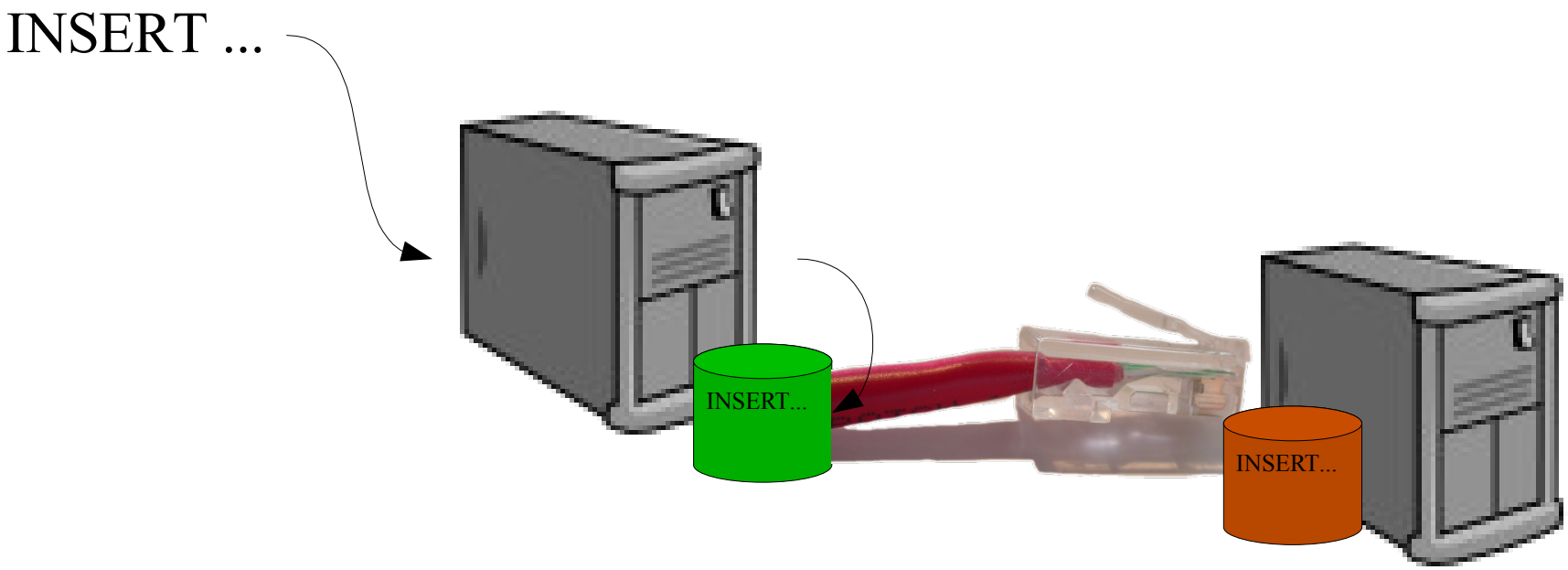
MySQL Replication



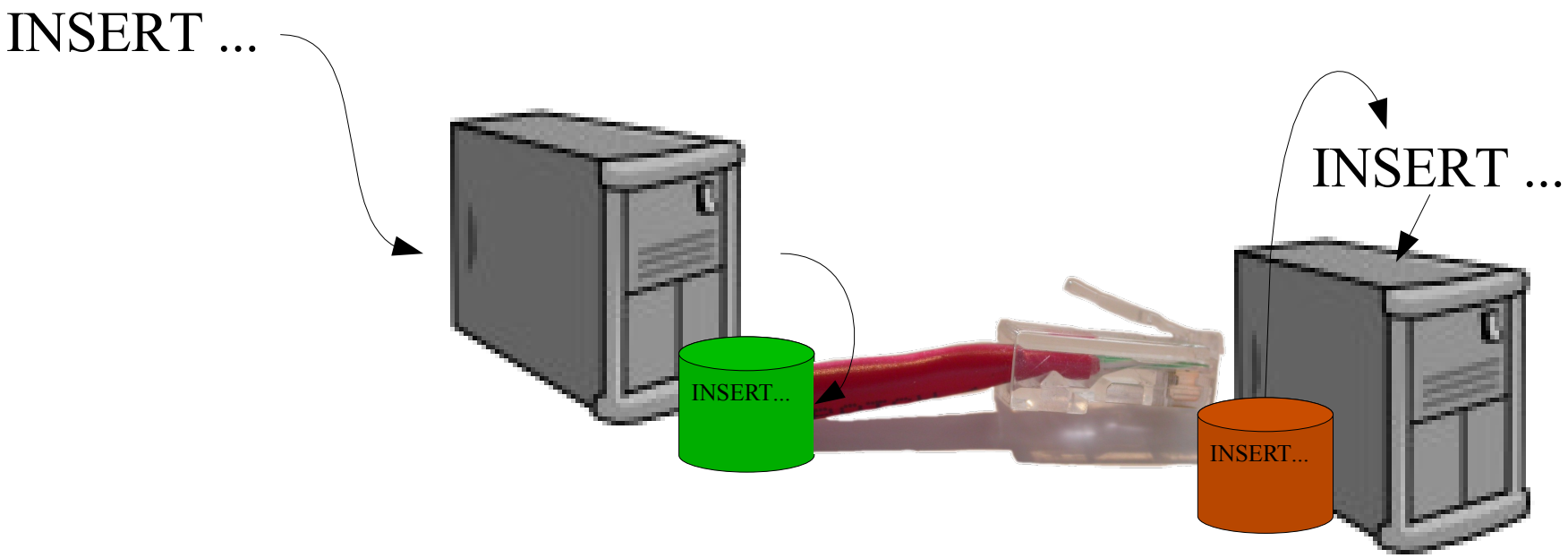
MySQL Replication



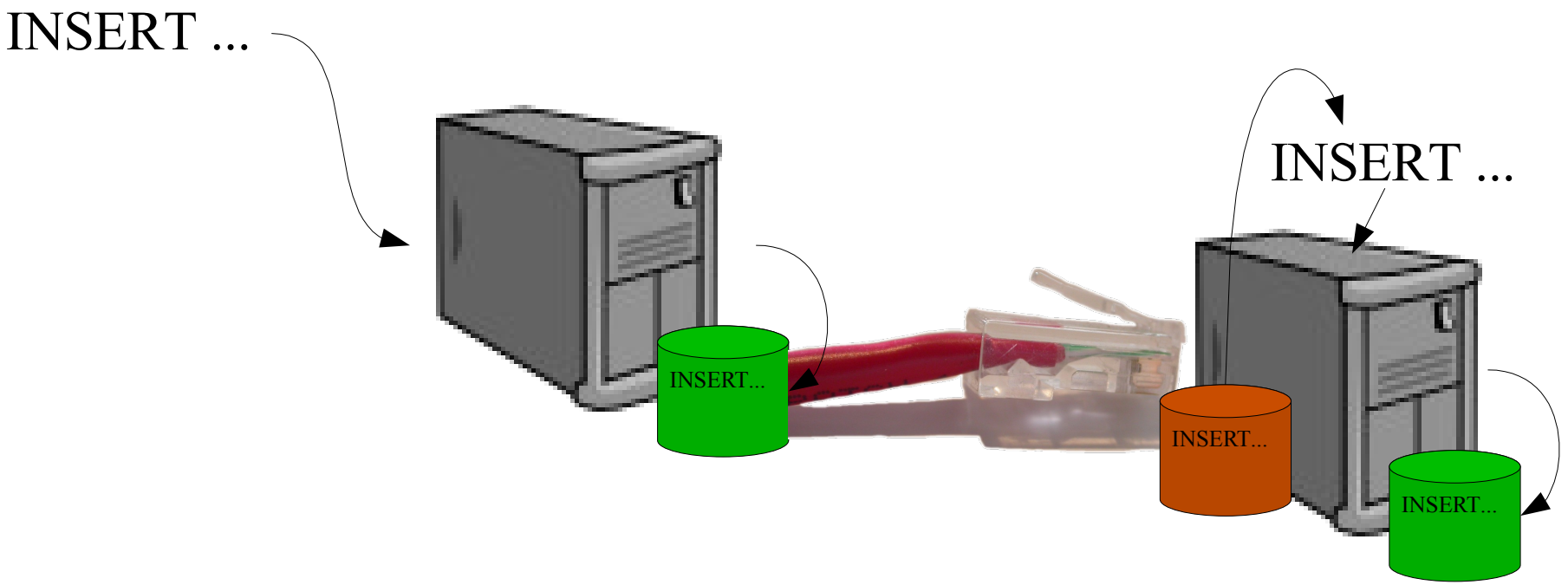
MySQL Replication



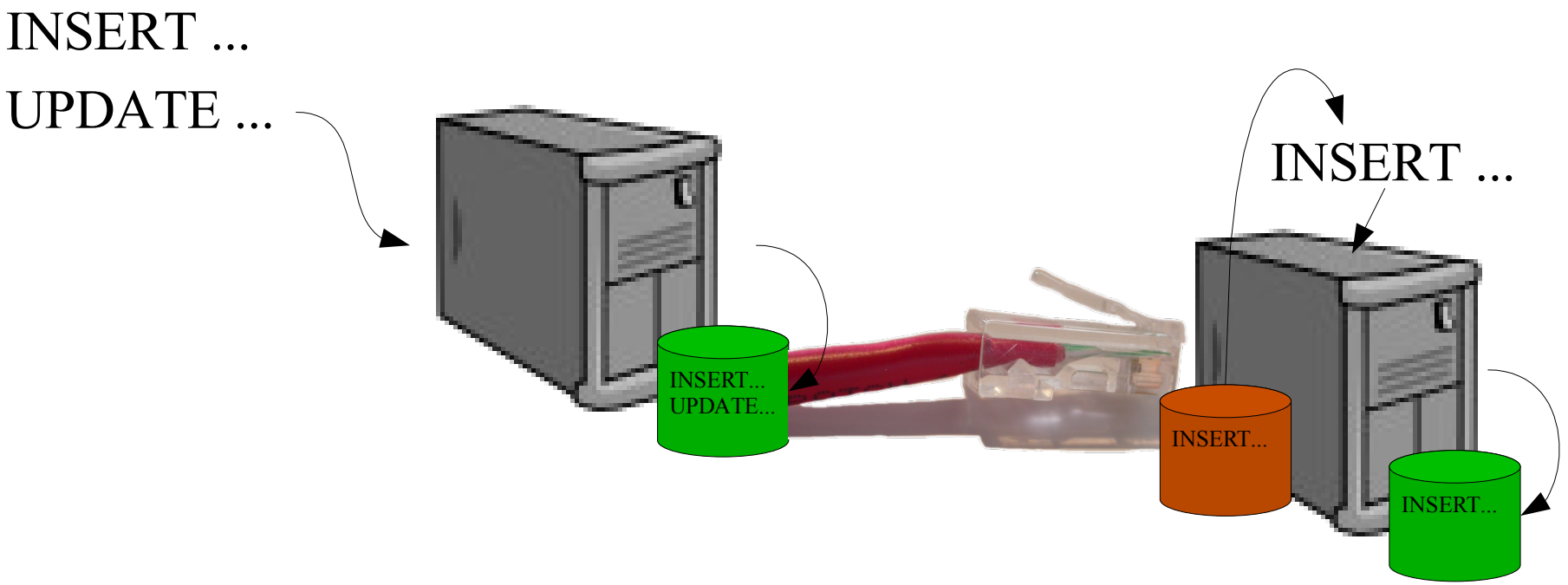
MySQL Replication



MySQL Replication

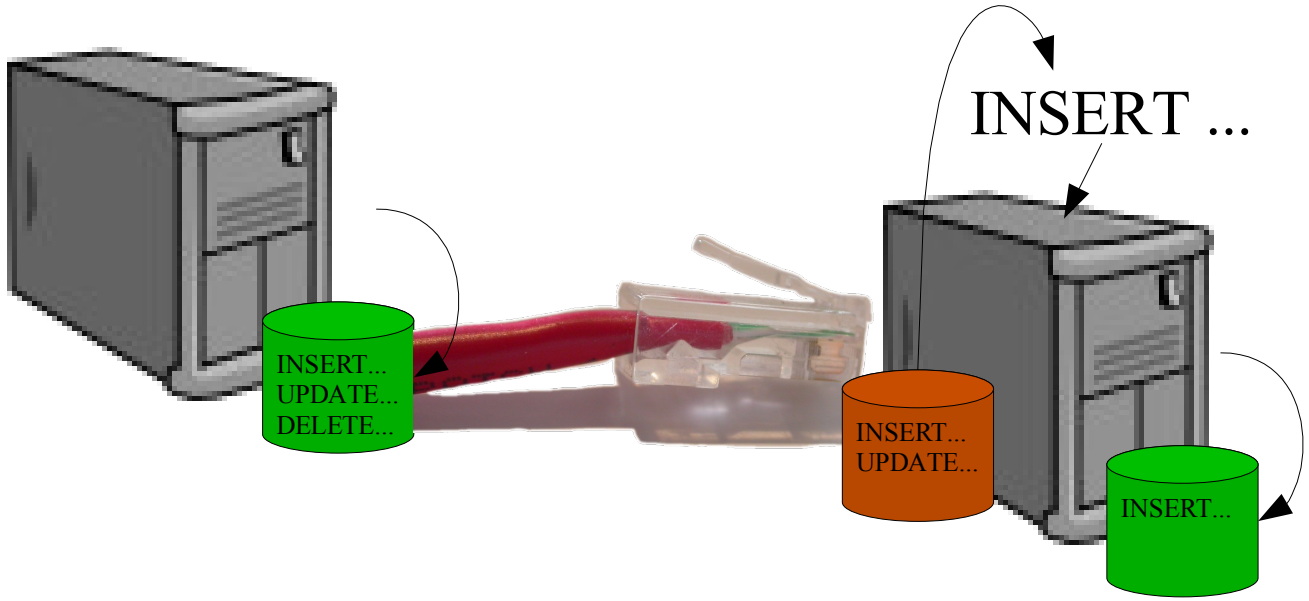


MySQL Replication



MySQL Replication

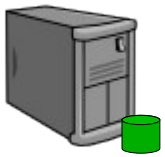
INSERT ...
UPDATE ...
DELETE ...



MySQL Replication

Master

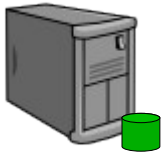
MySQL Replication



Master

MySQL Replication

Slave

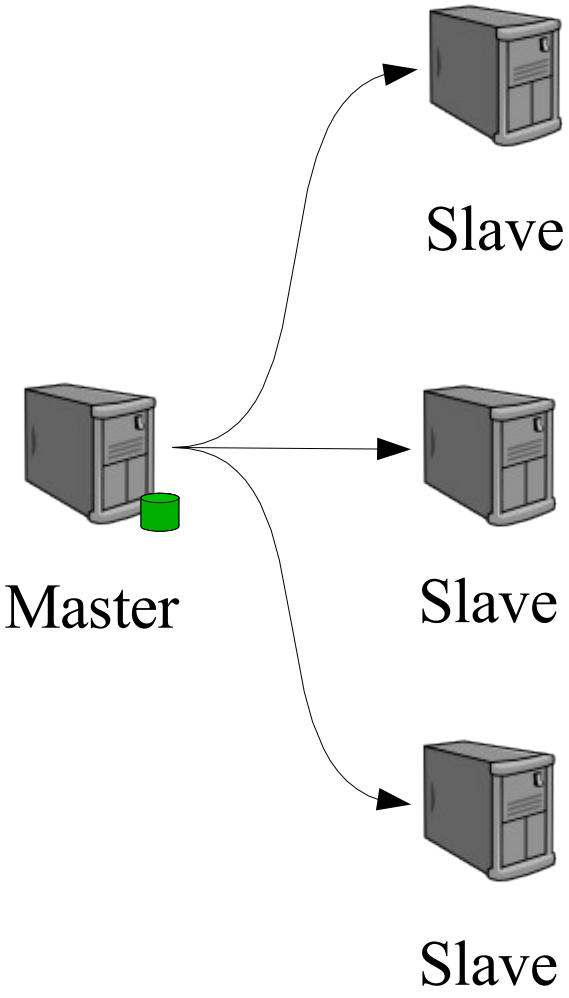


Master

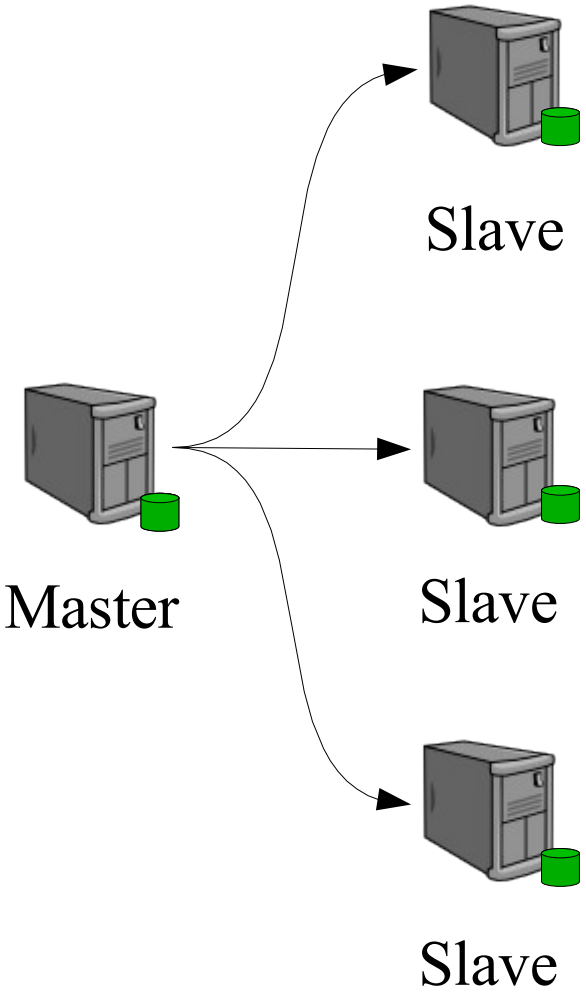
Slave

Slave

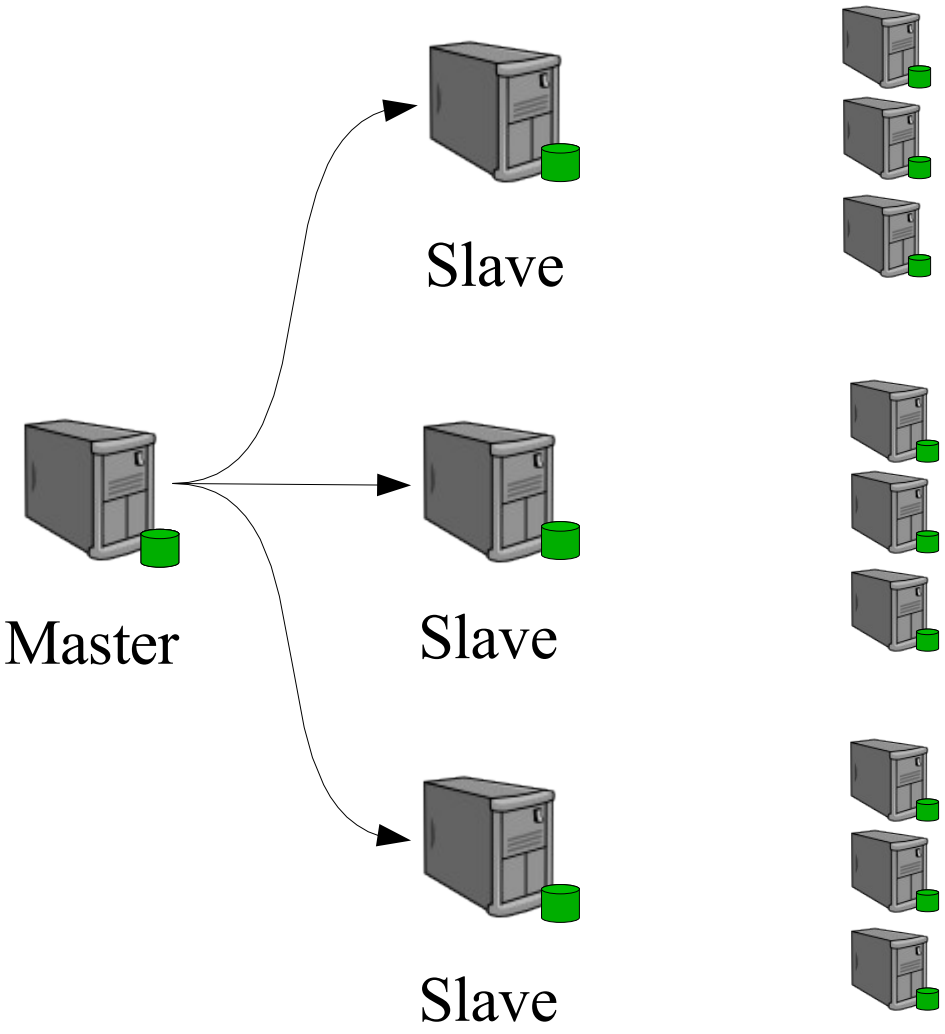
MySQL Replication



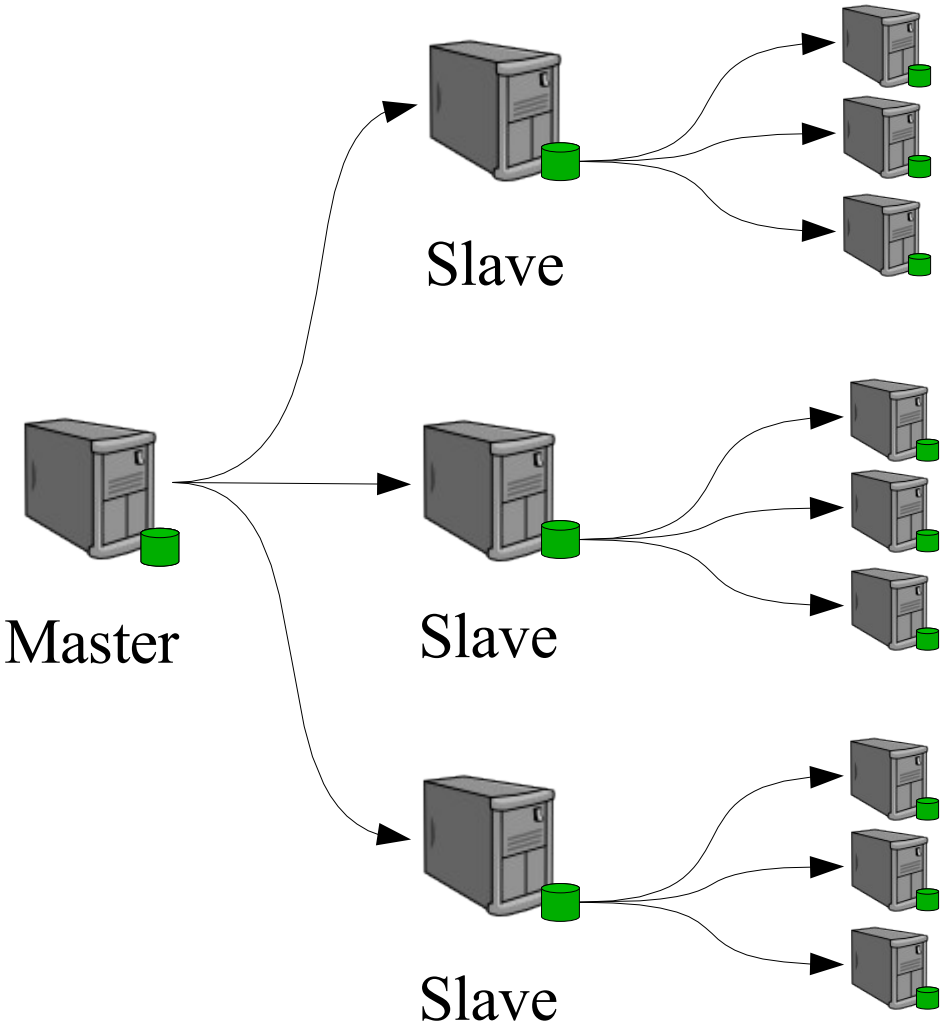
MySQL Replication



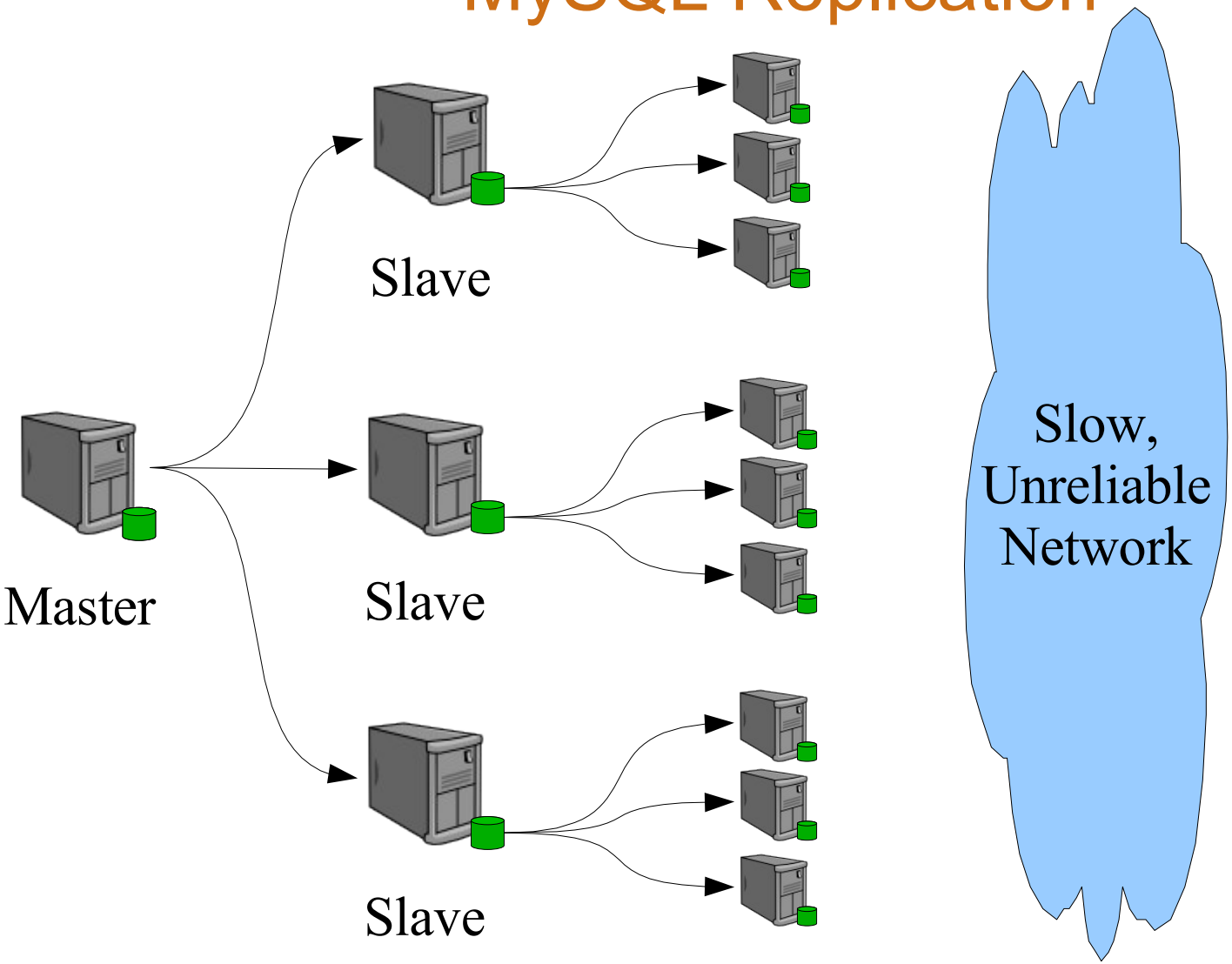
MySQL Replication



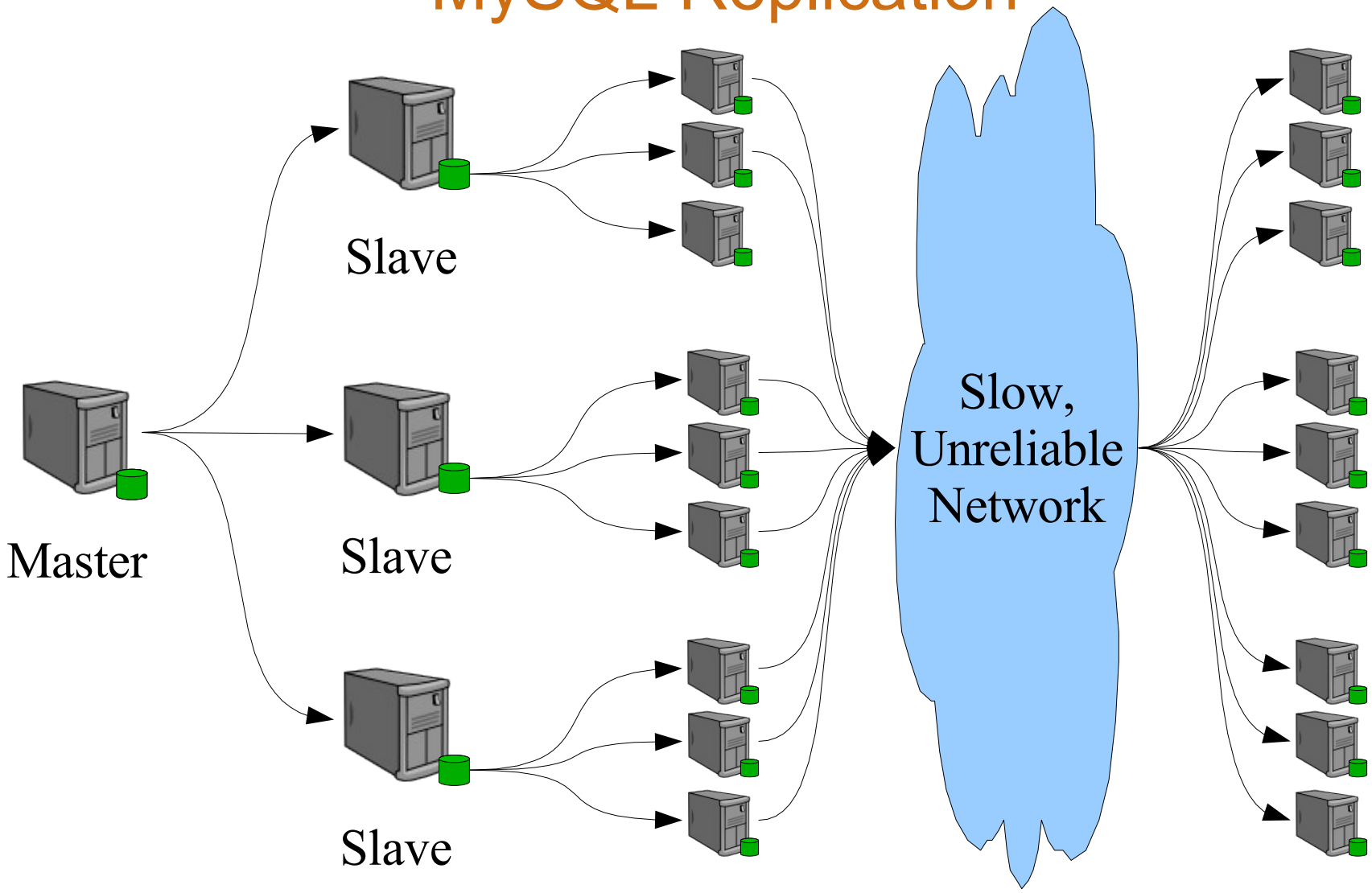
MySQL Replication



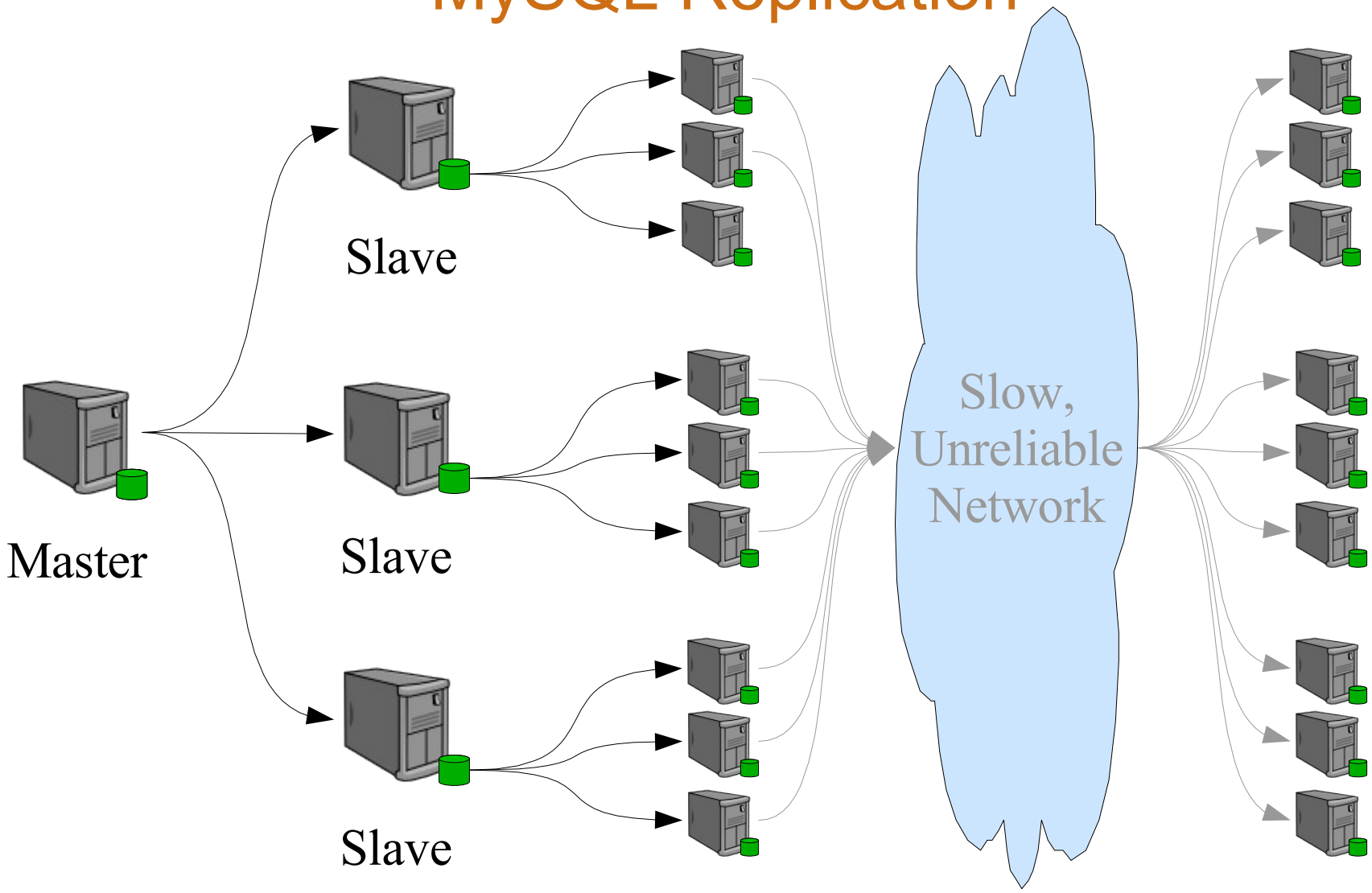
MySQL Replication



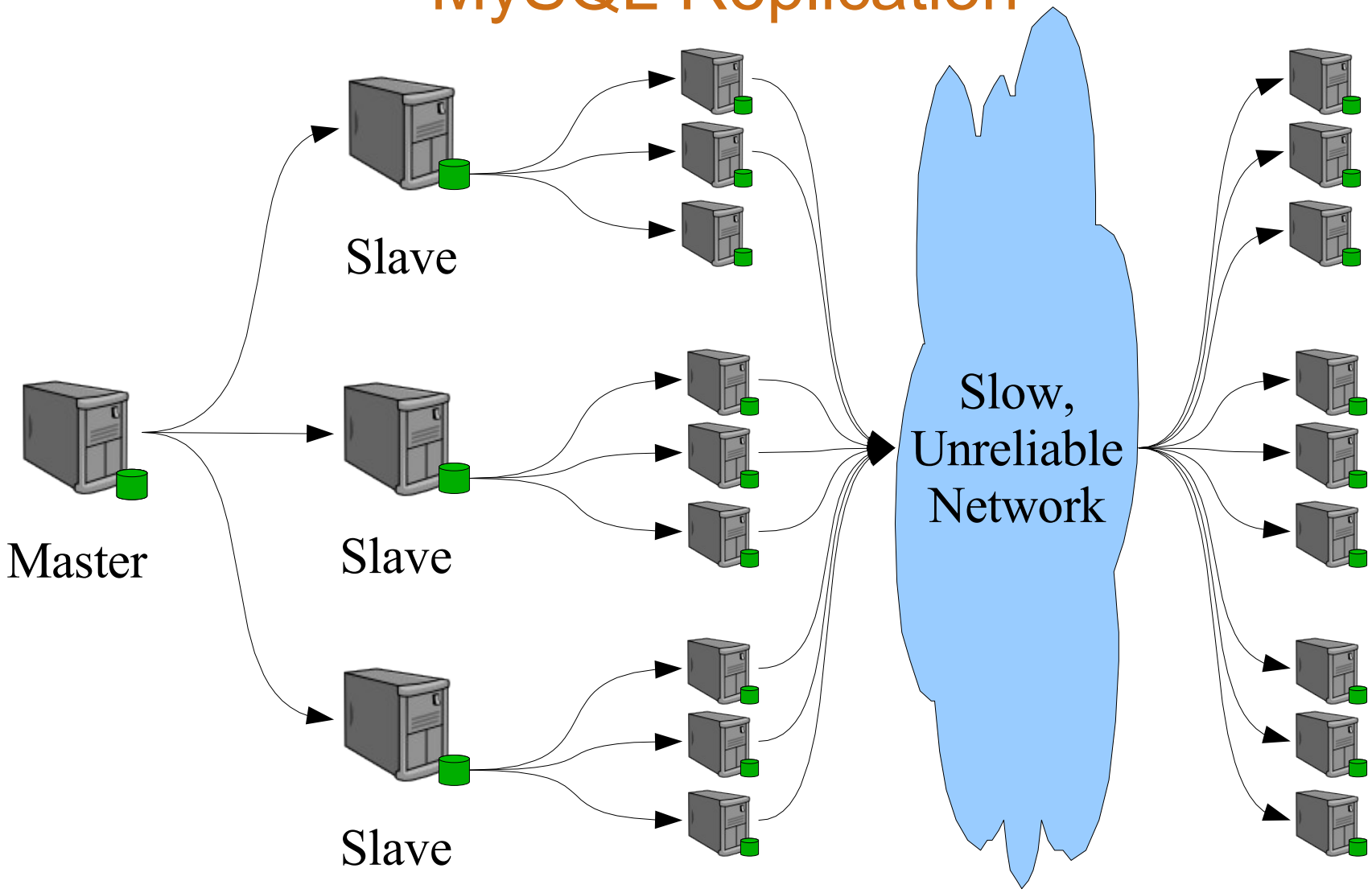
MySQL Replication



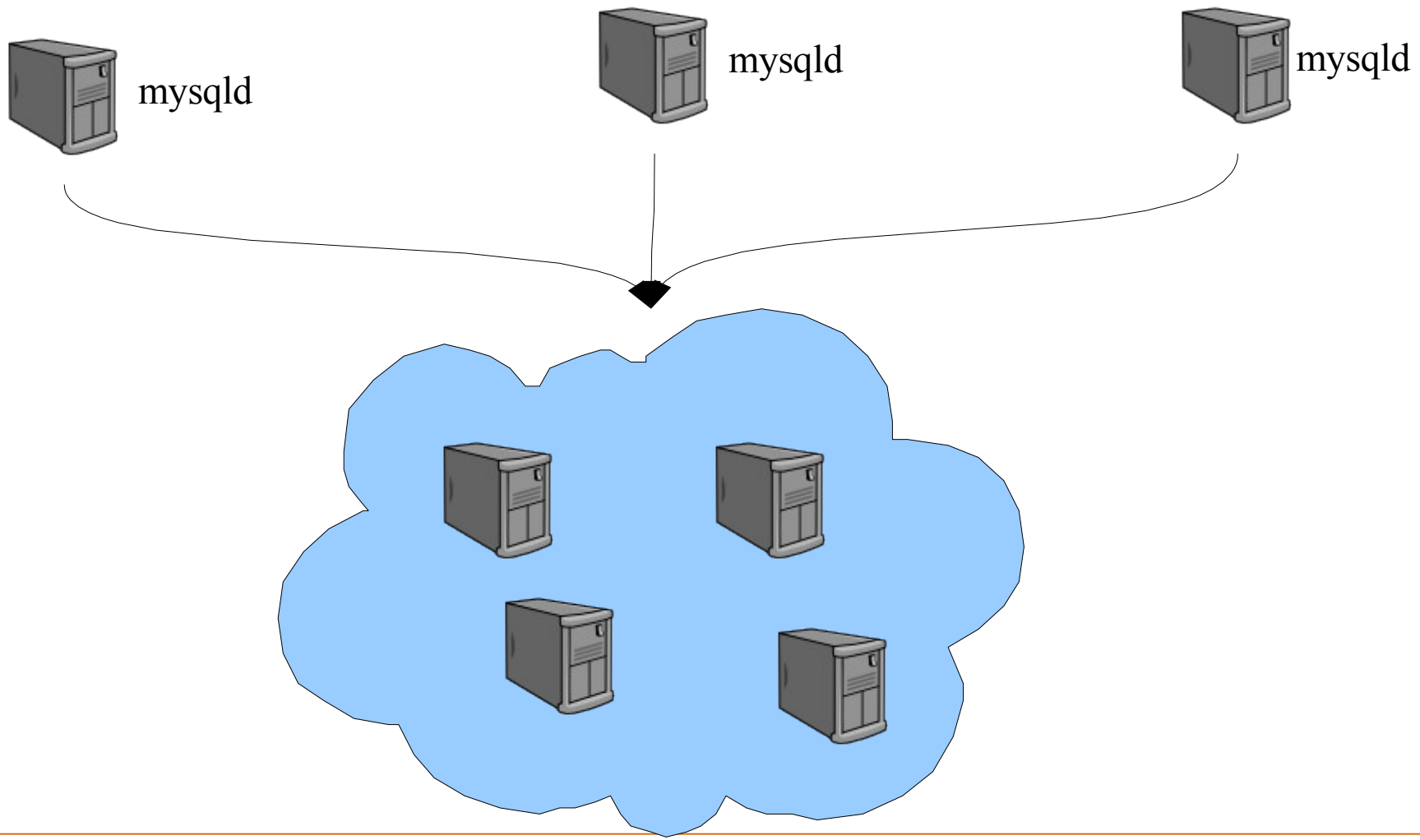
MySQL Replication

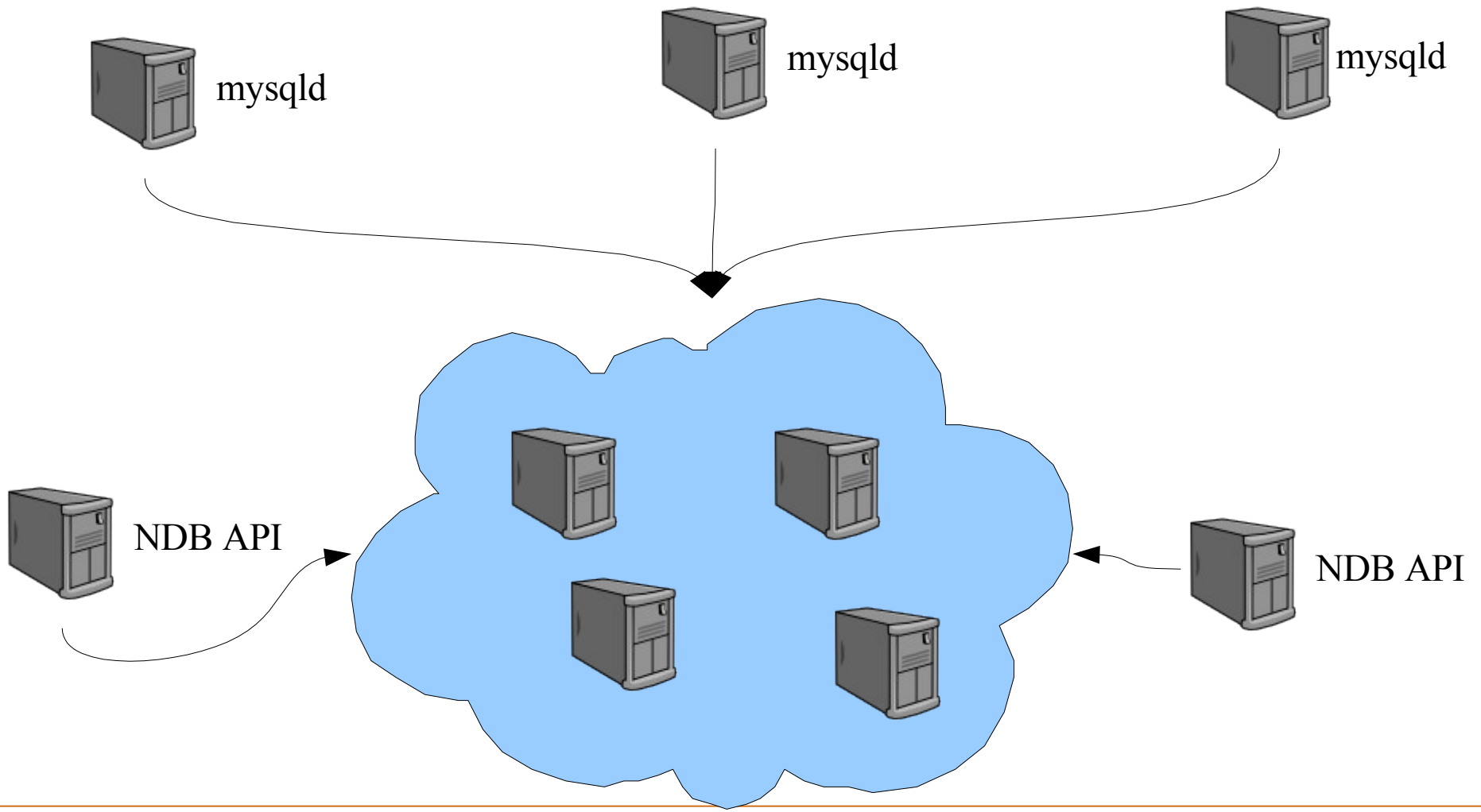


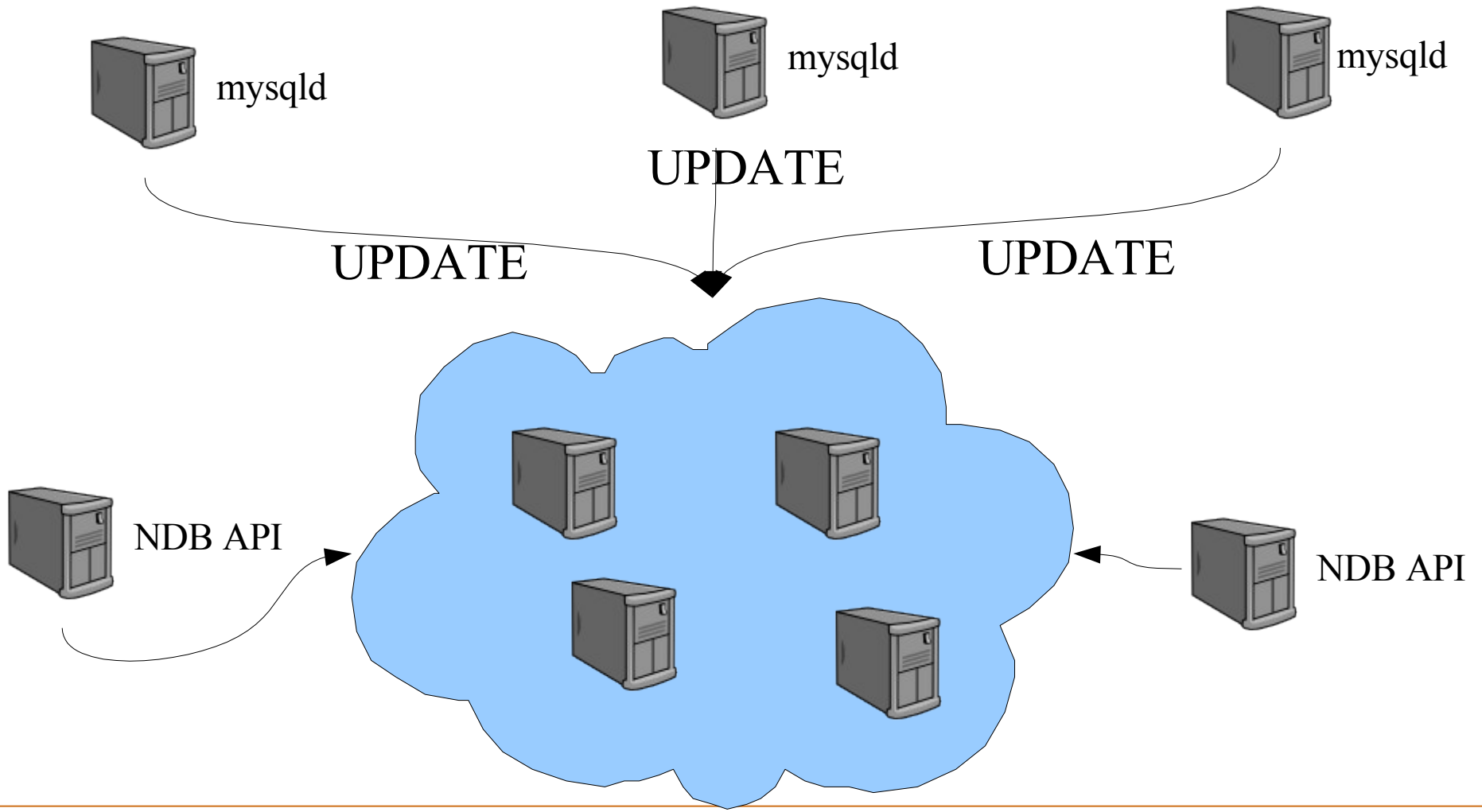
MySQL Replication

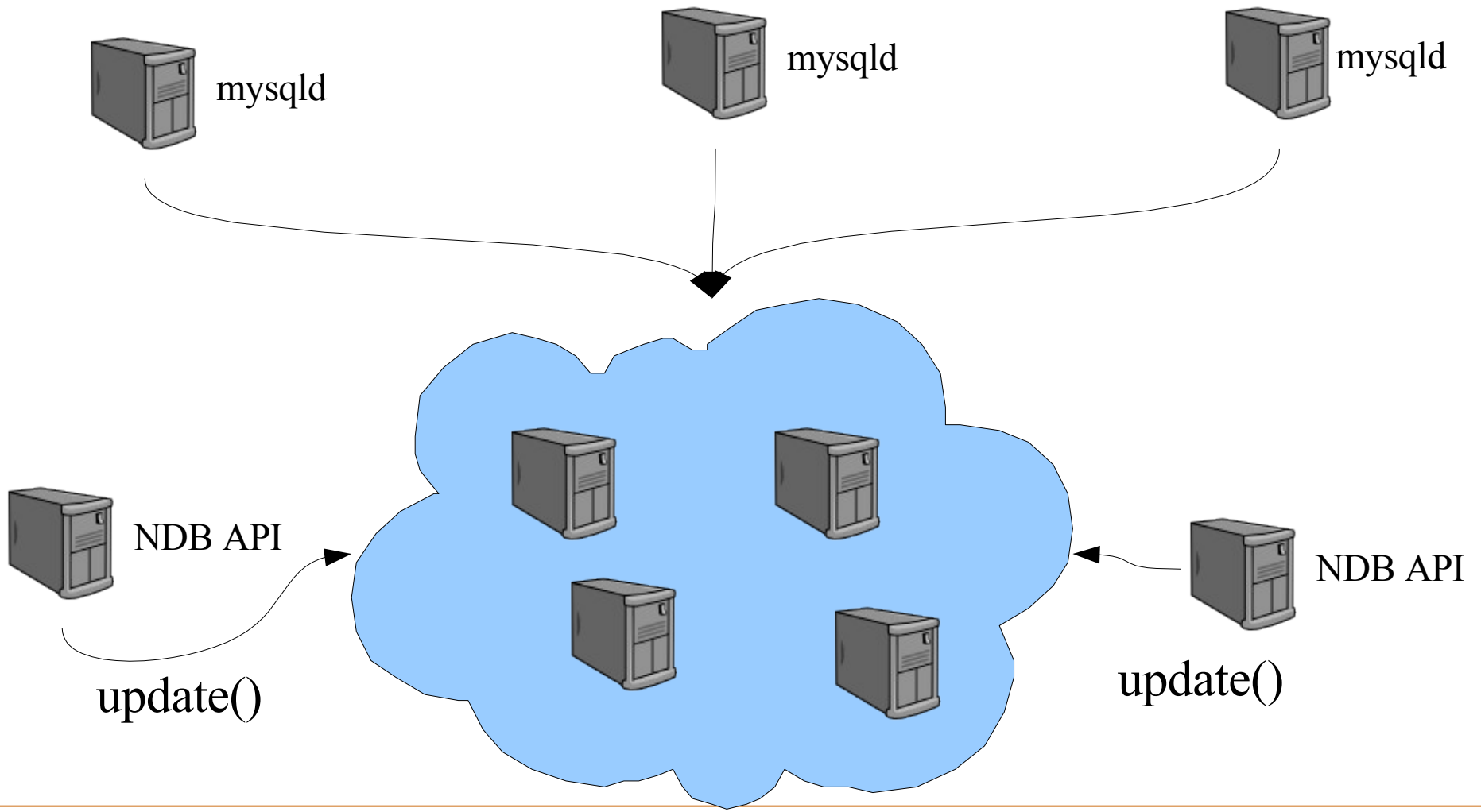


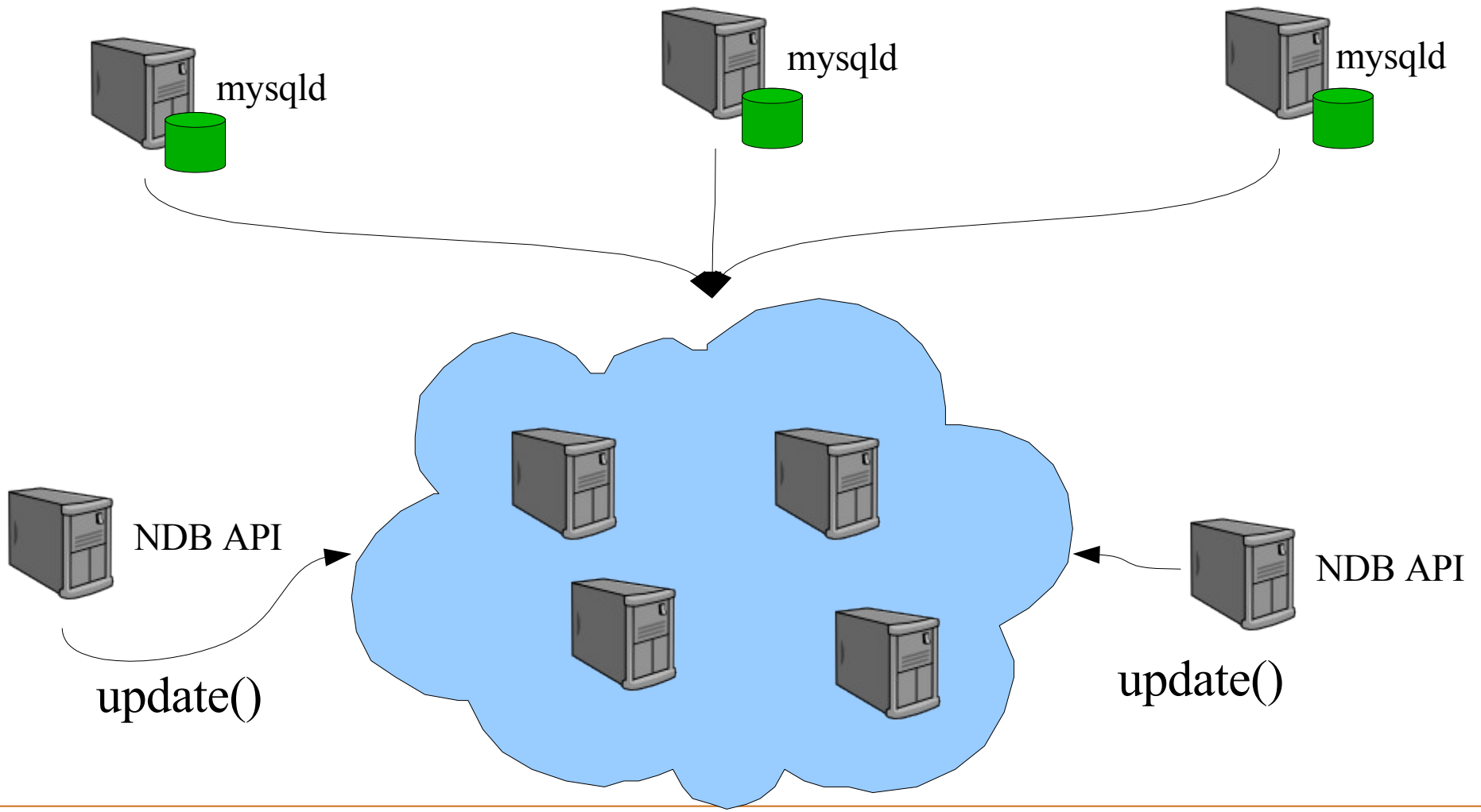
Back at Cluster

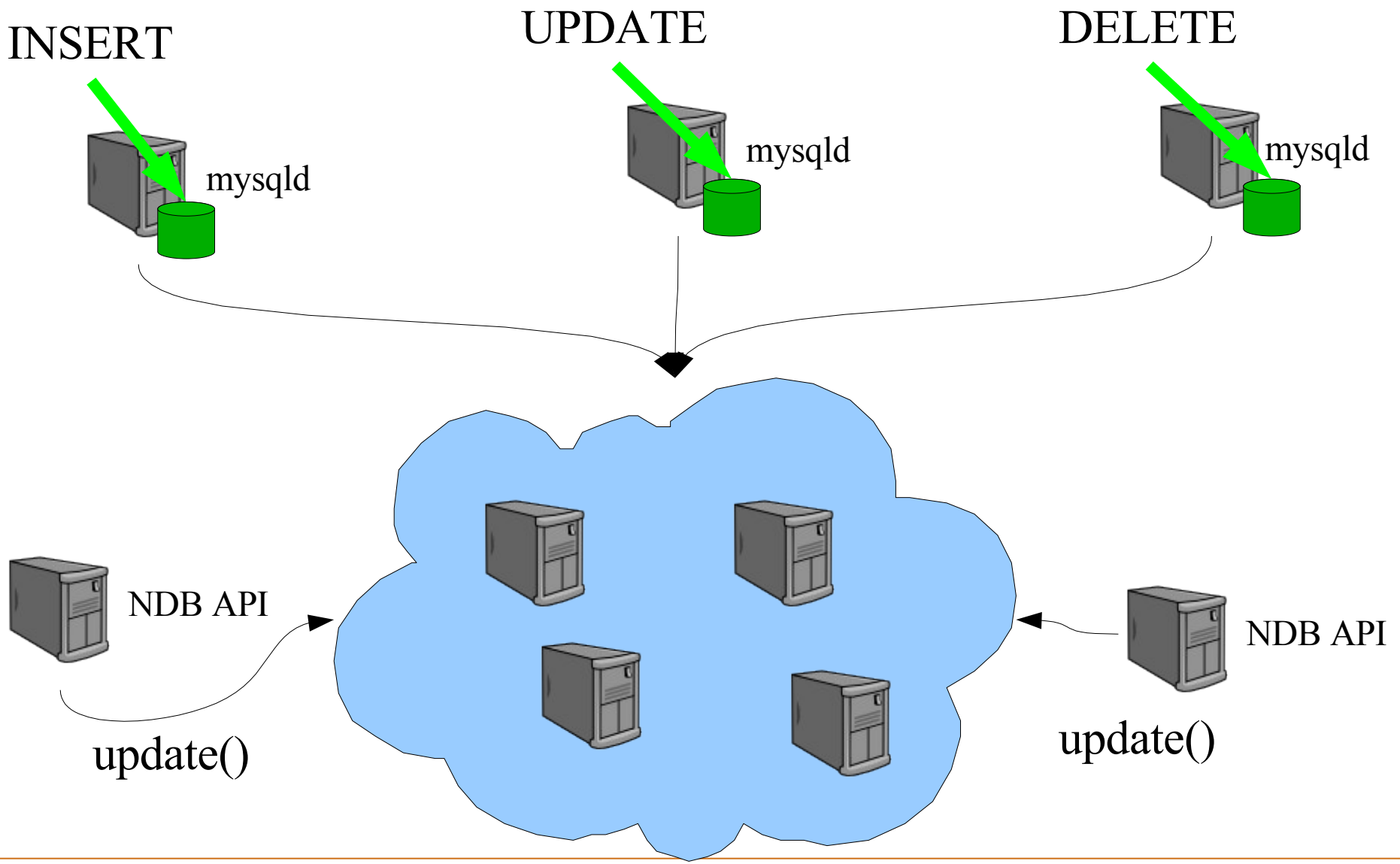


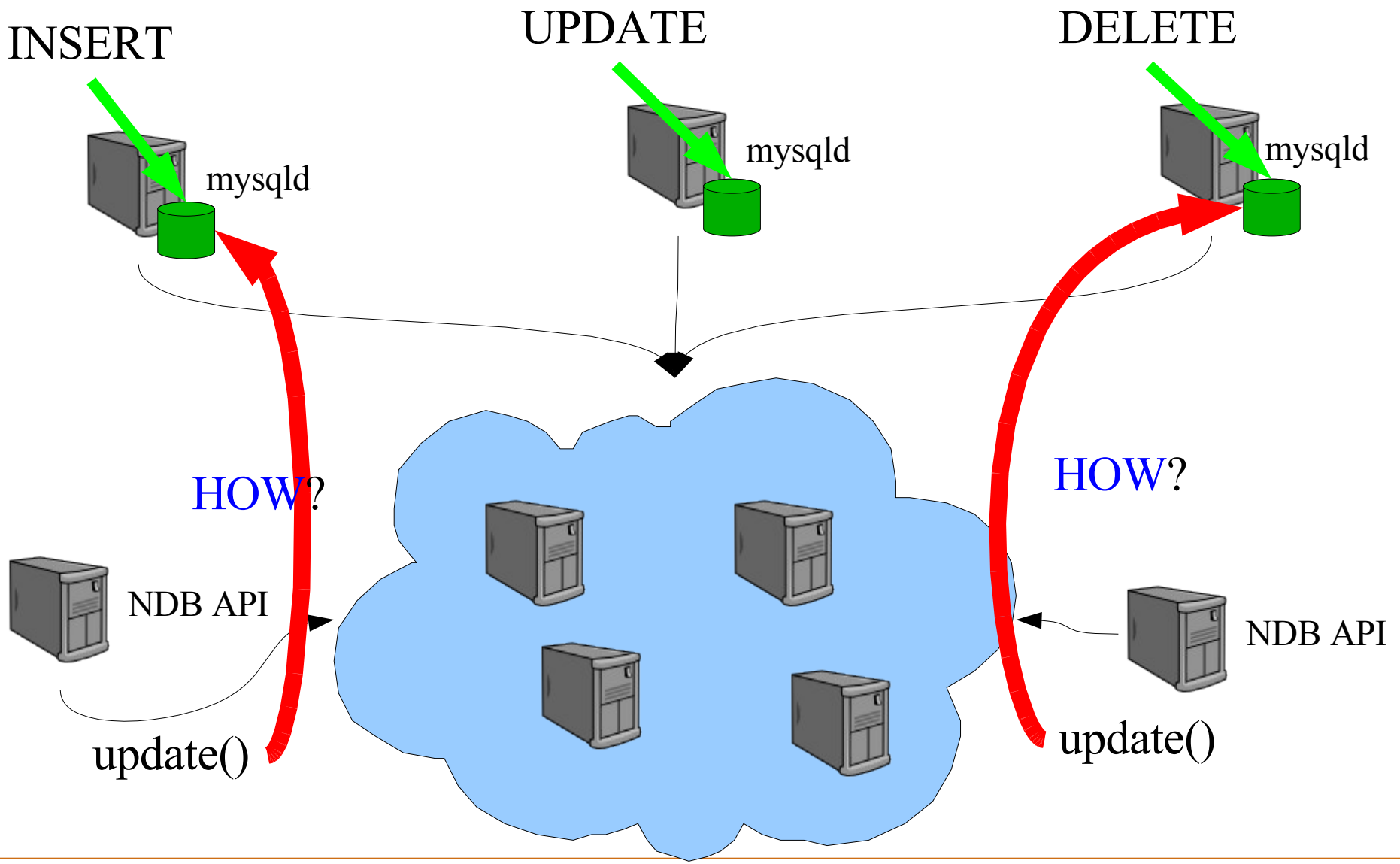


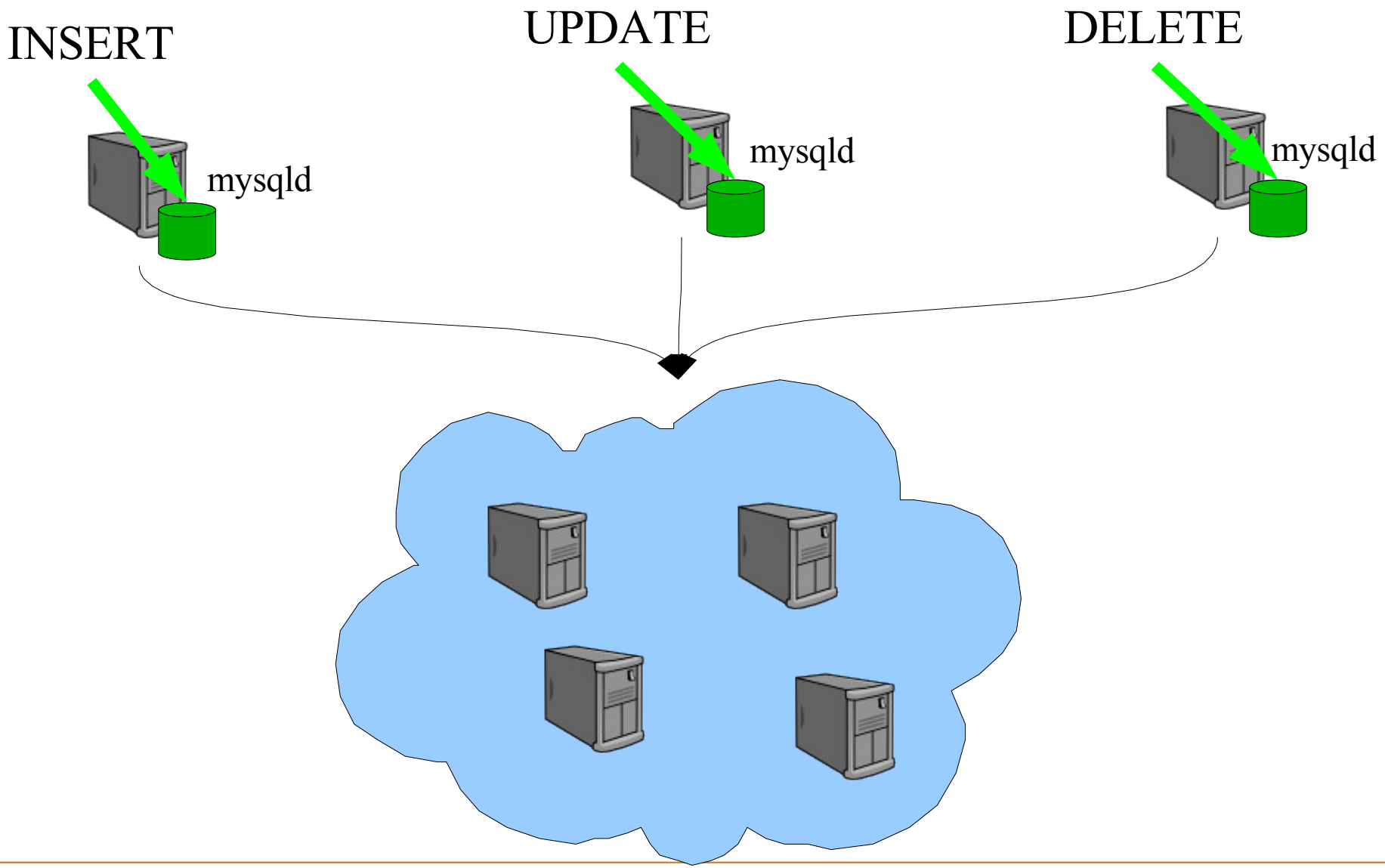


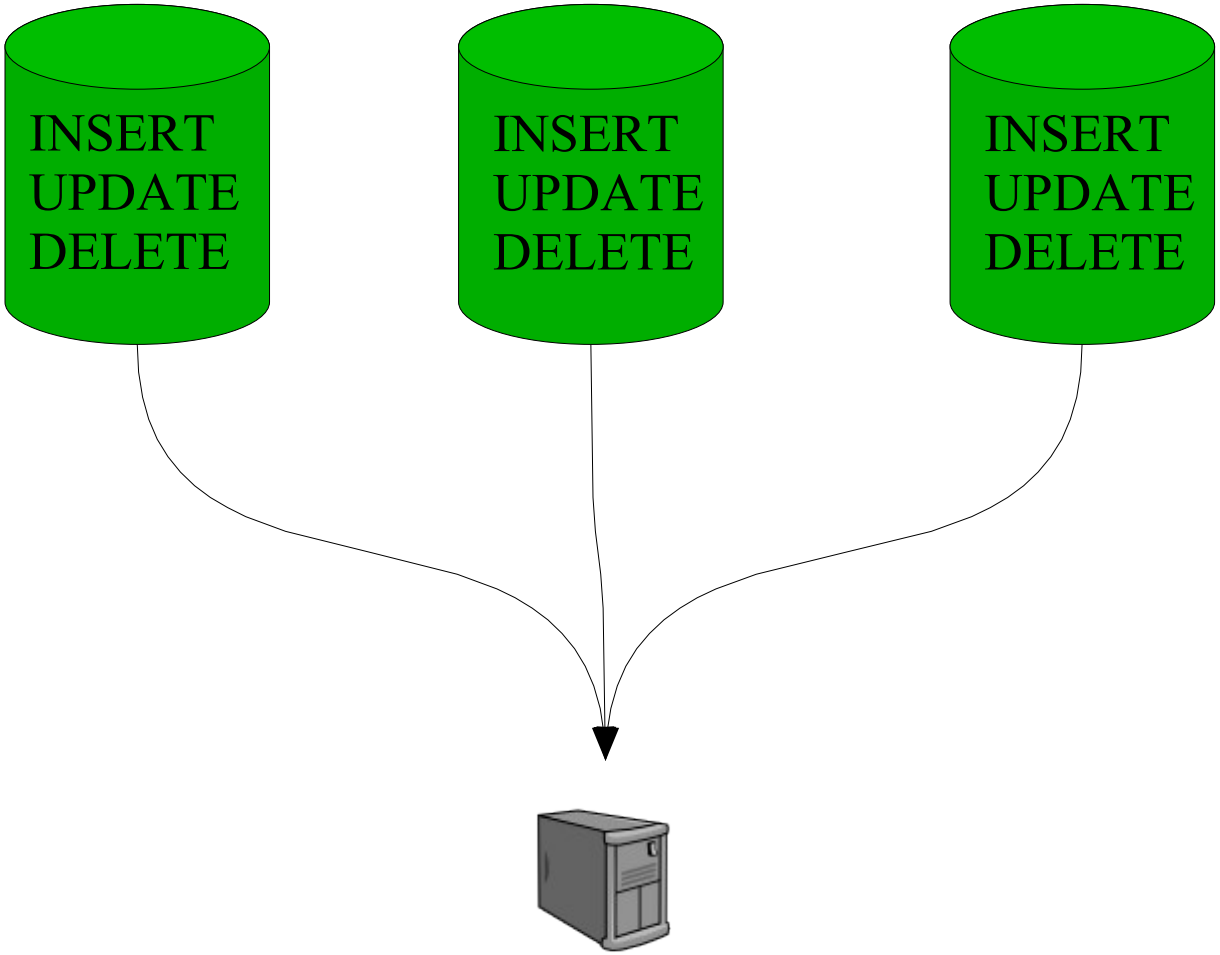


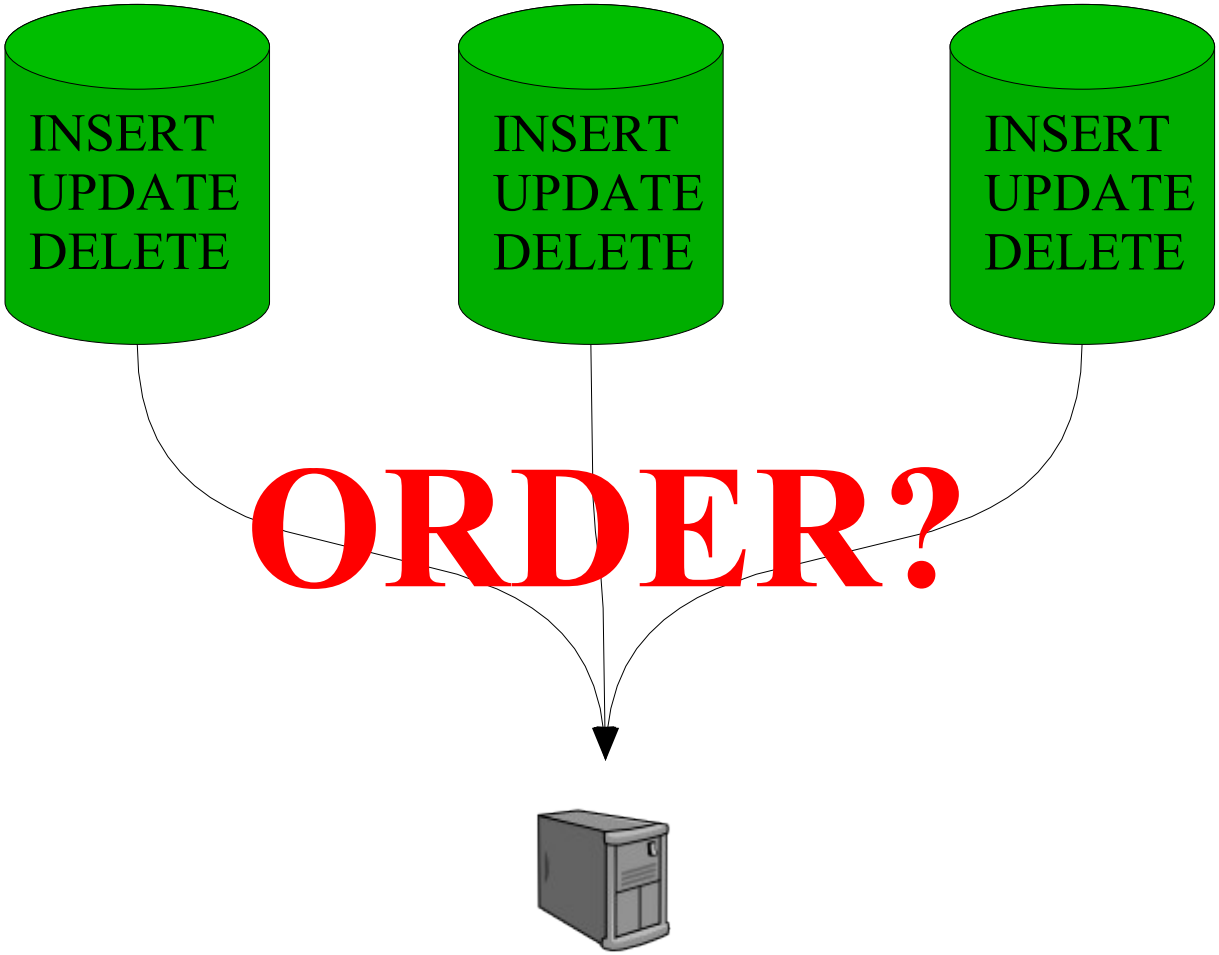




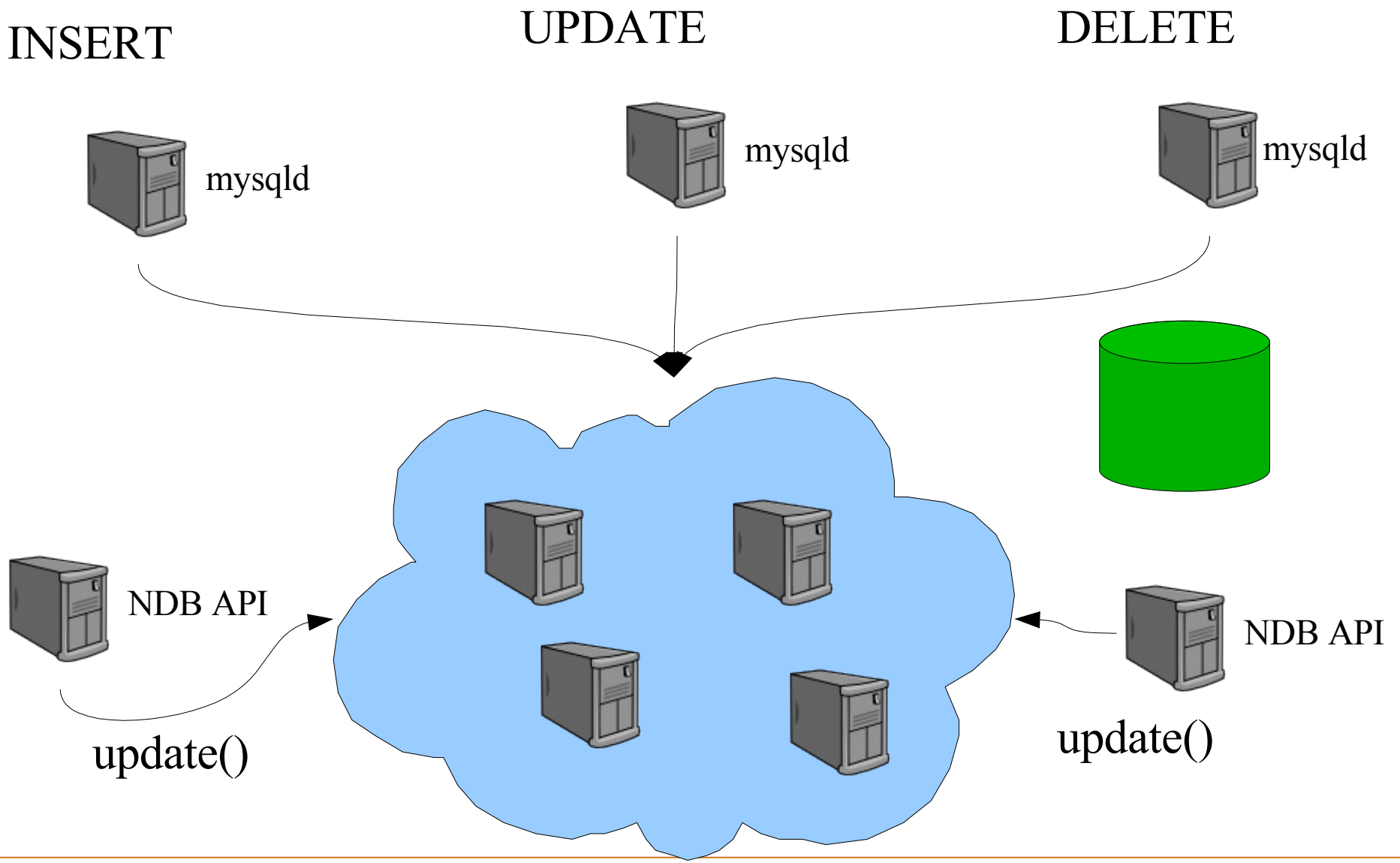


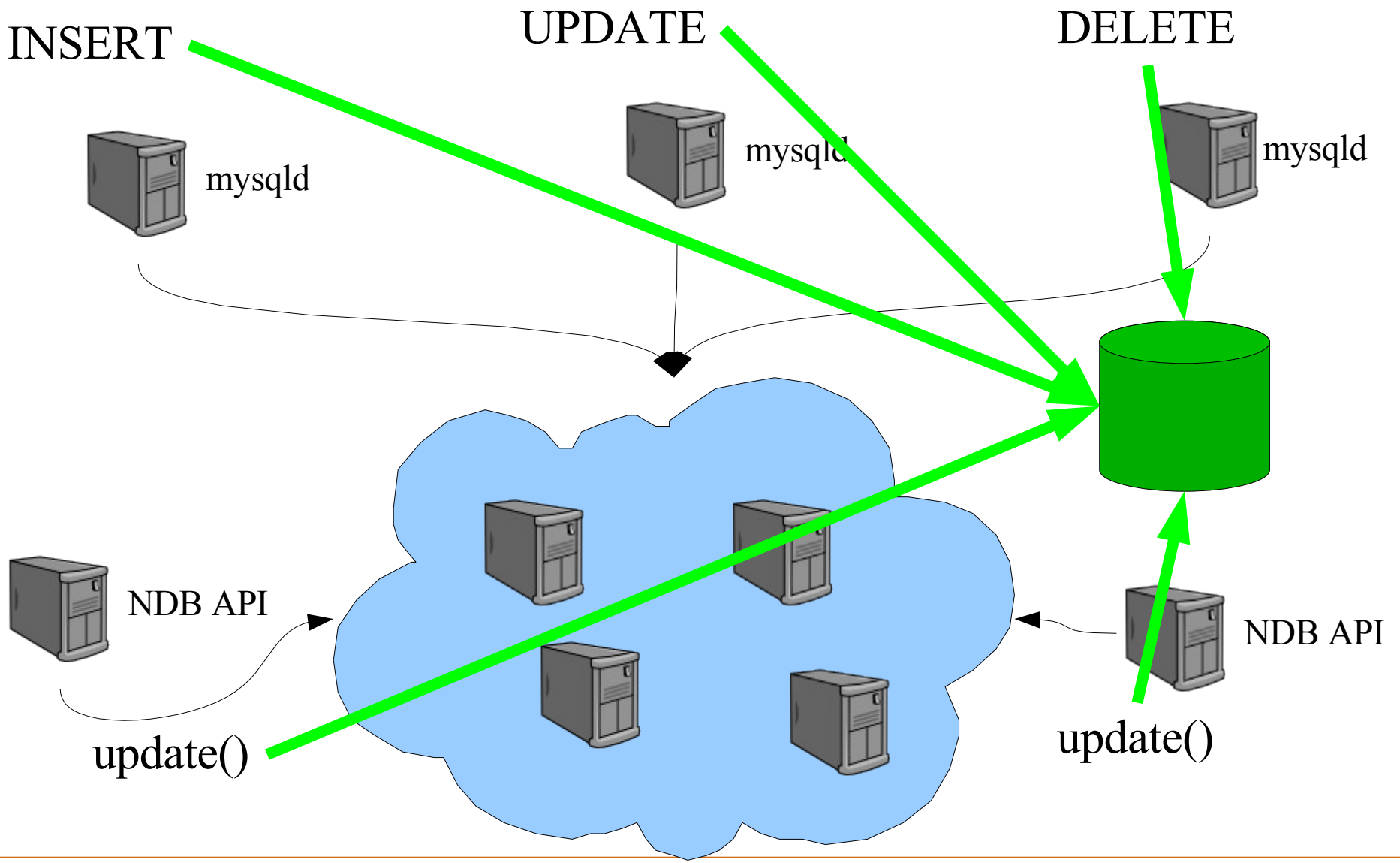


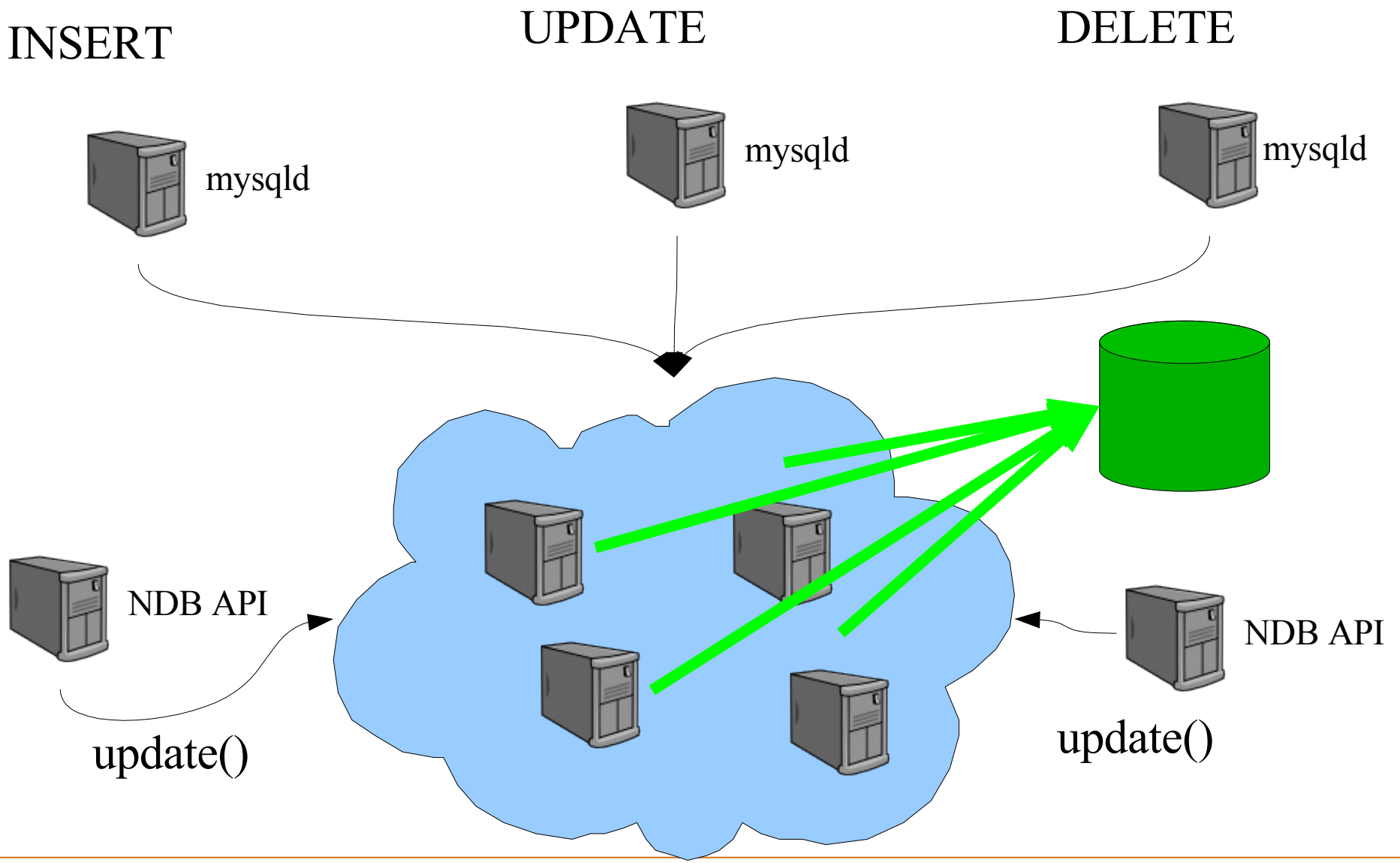




Serialization is in the storage nodes

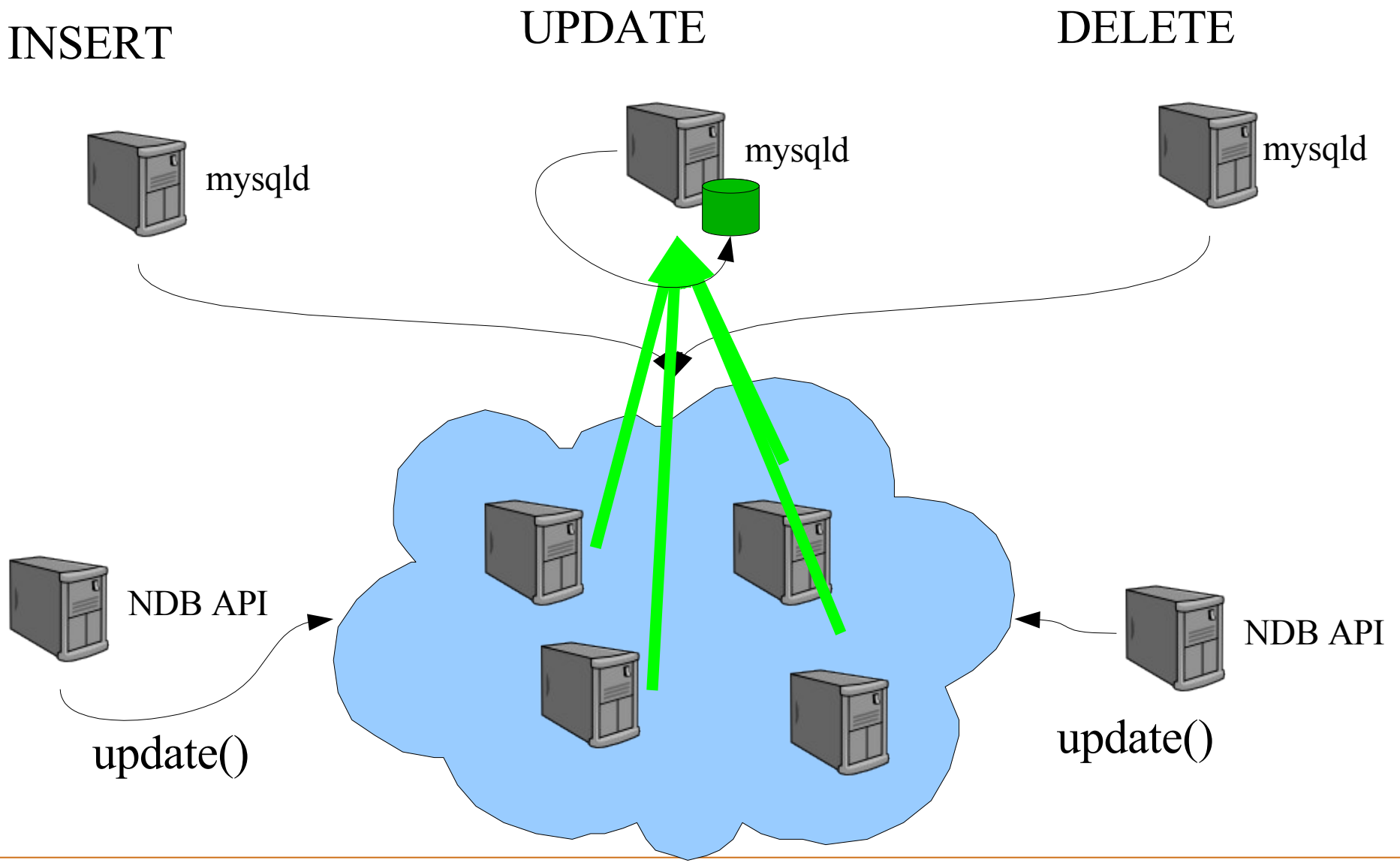






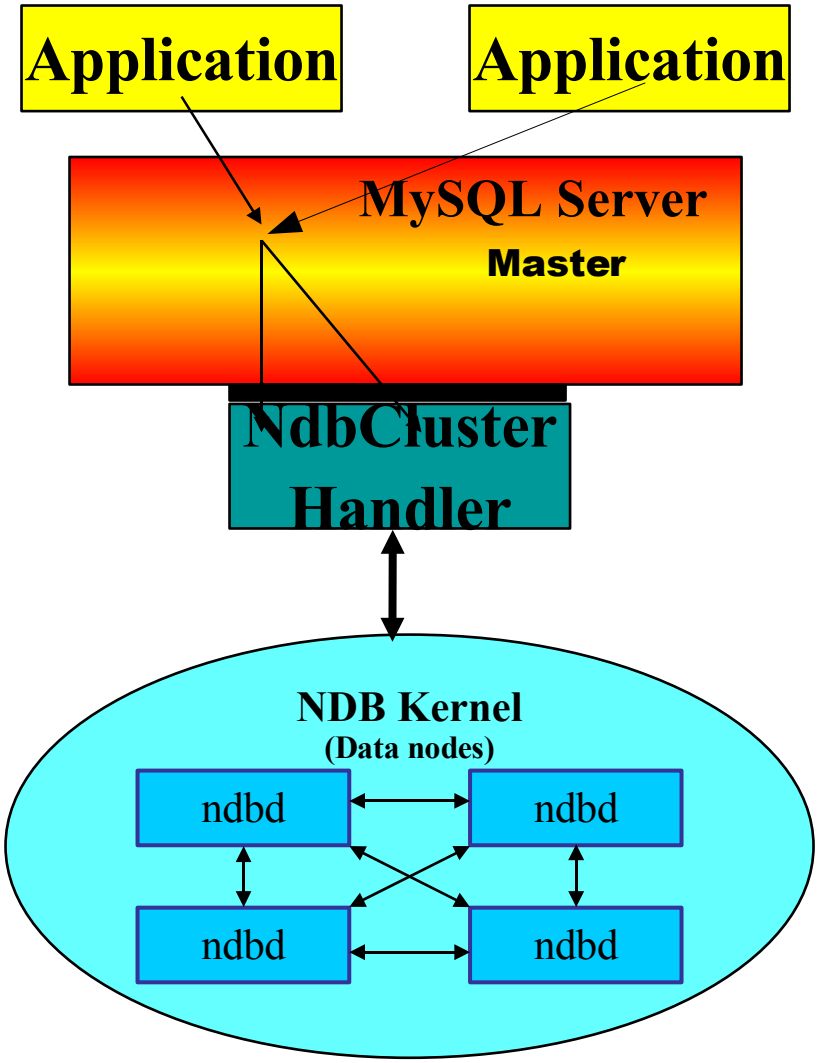
NDB Injector Thread

- A thread inside the MySQL Server
- Subscribes to events in NDB
  - The event of “Row was committed”
- Injects the rows into the binlog
- Producing a single, canonical binlog of your cluster
  - not just one mysql server
  - it contains EVERYTHING done on ALL ndbapi programs (including mysqld) connected to the cluster

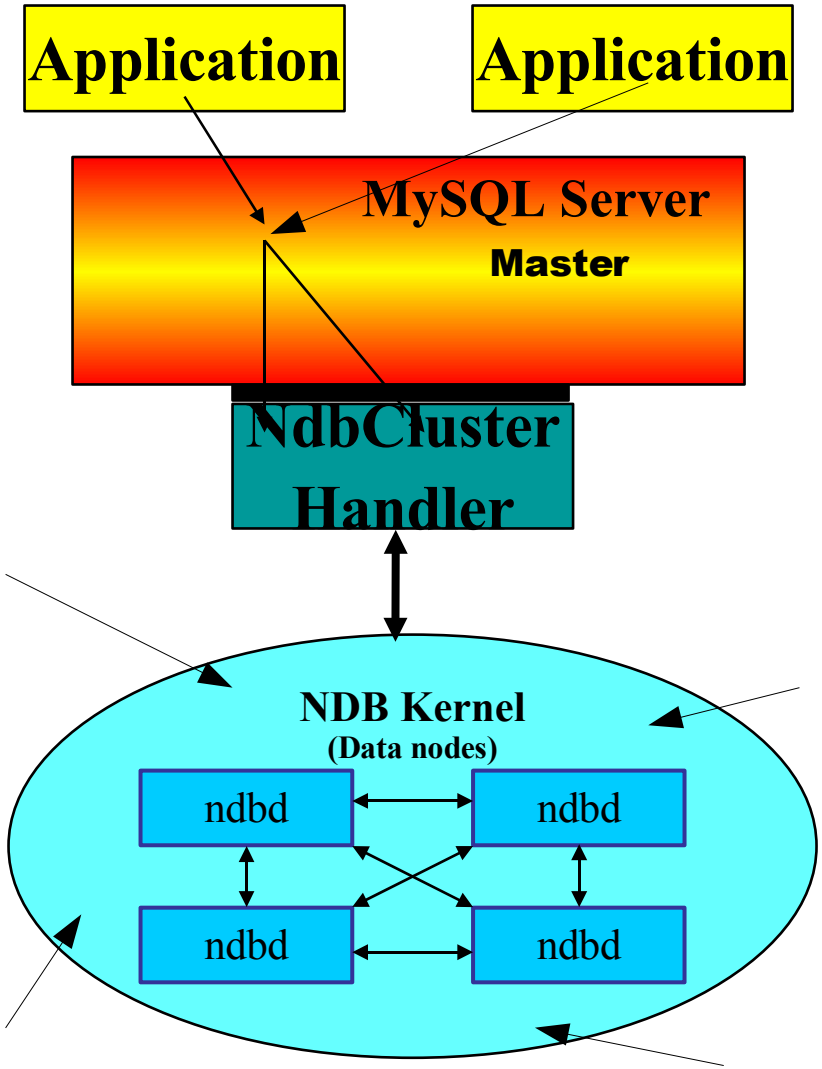


A Closer Look...

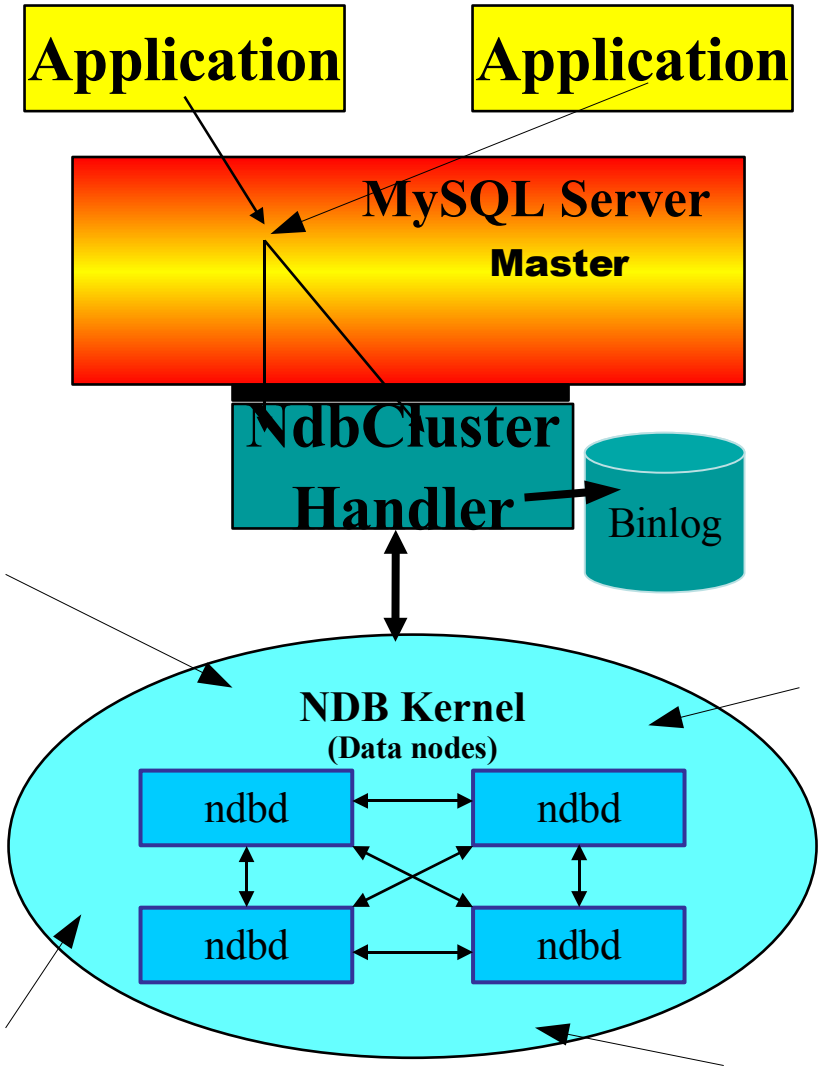
MySQL Replication between Clusters



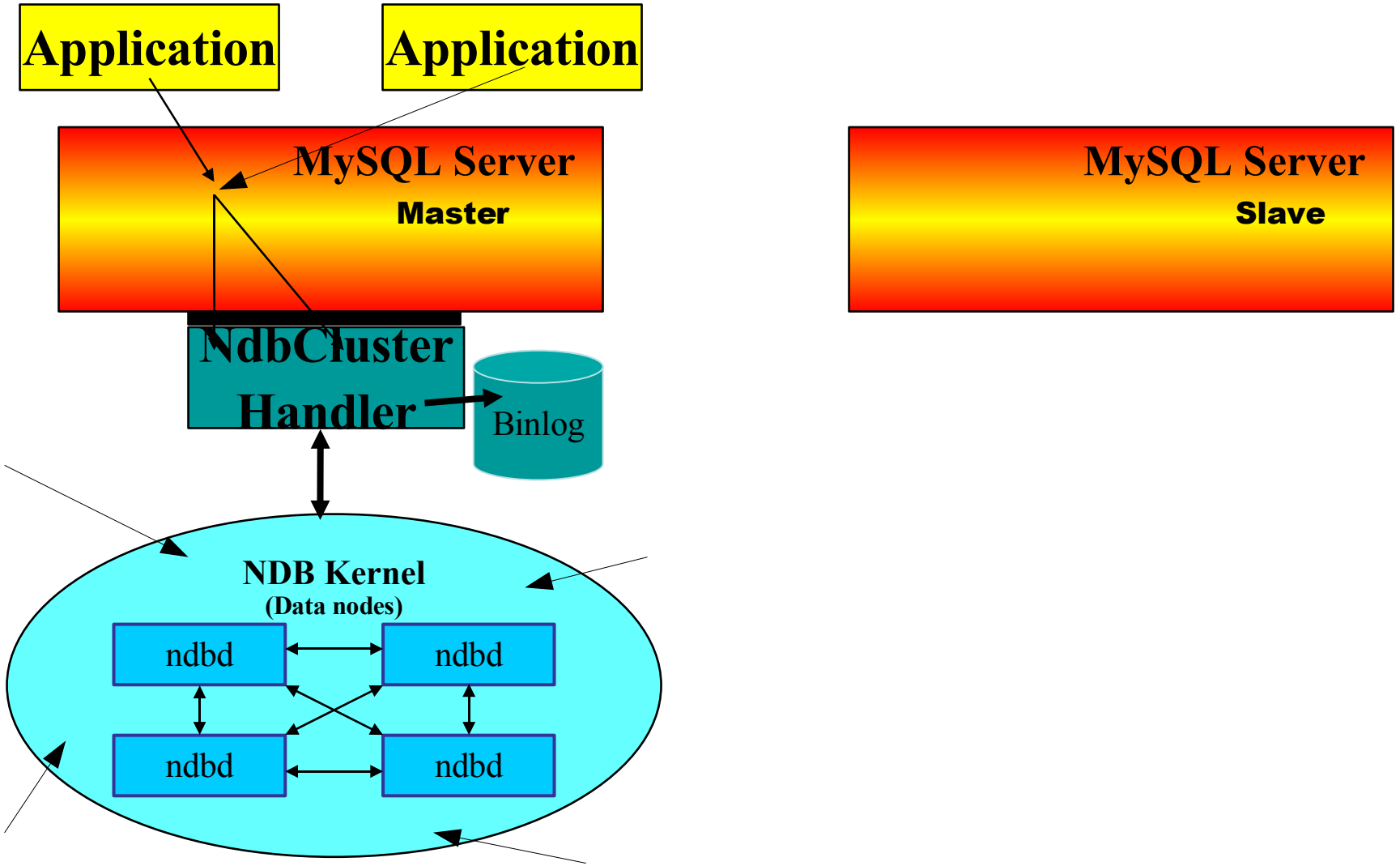
MySQL Replication between Clusters



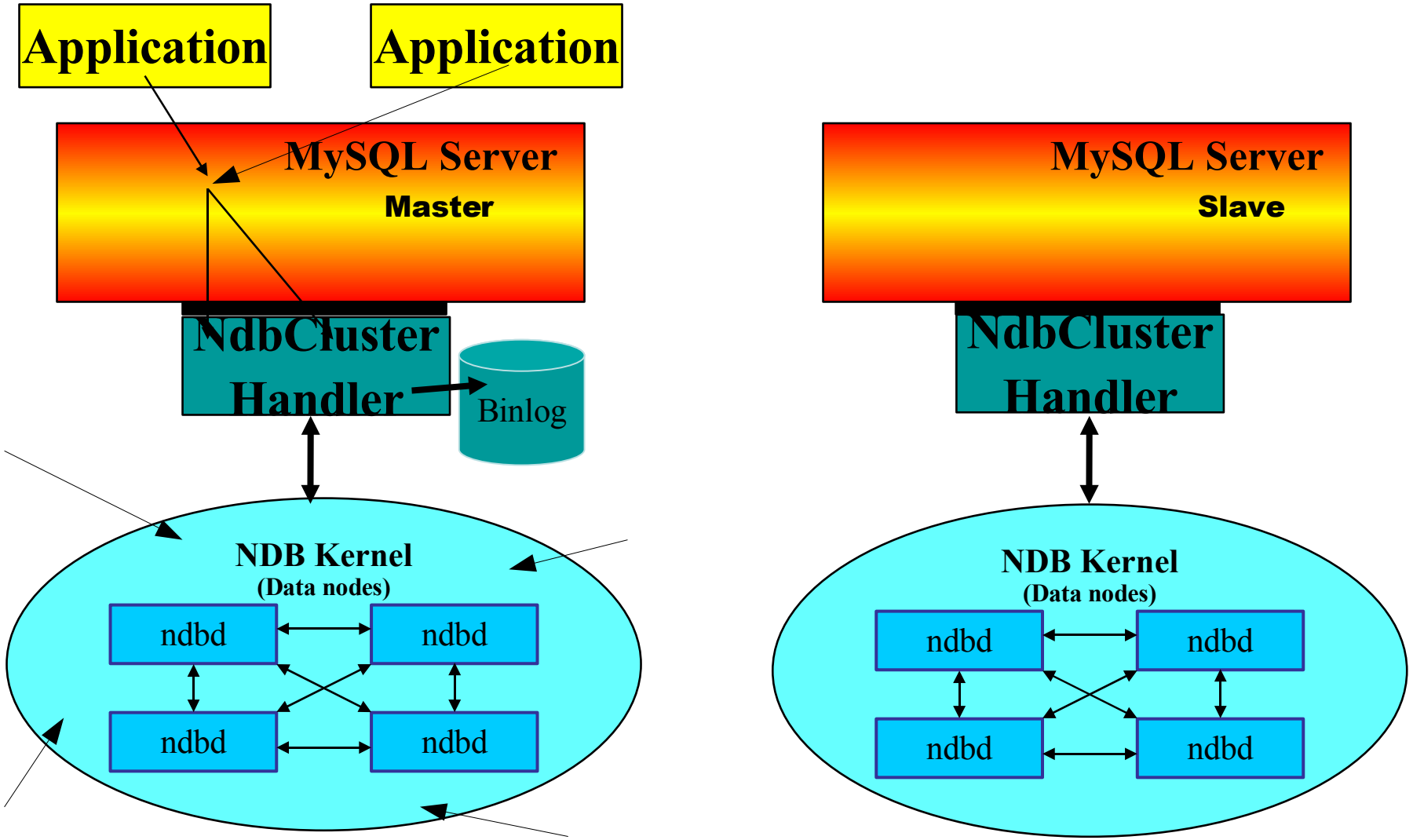
MySQL Replication between Clusters



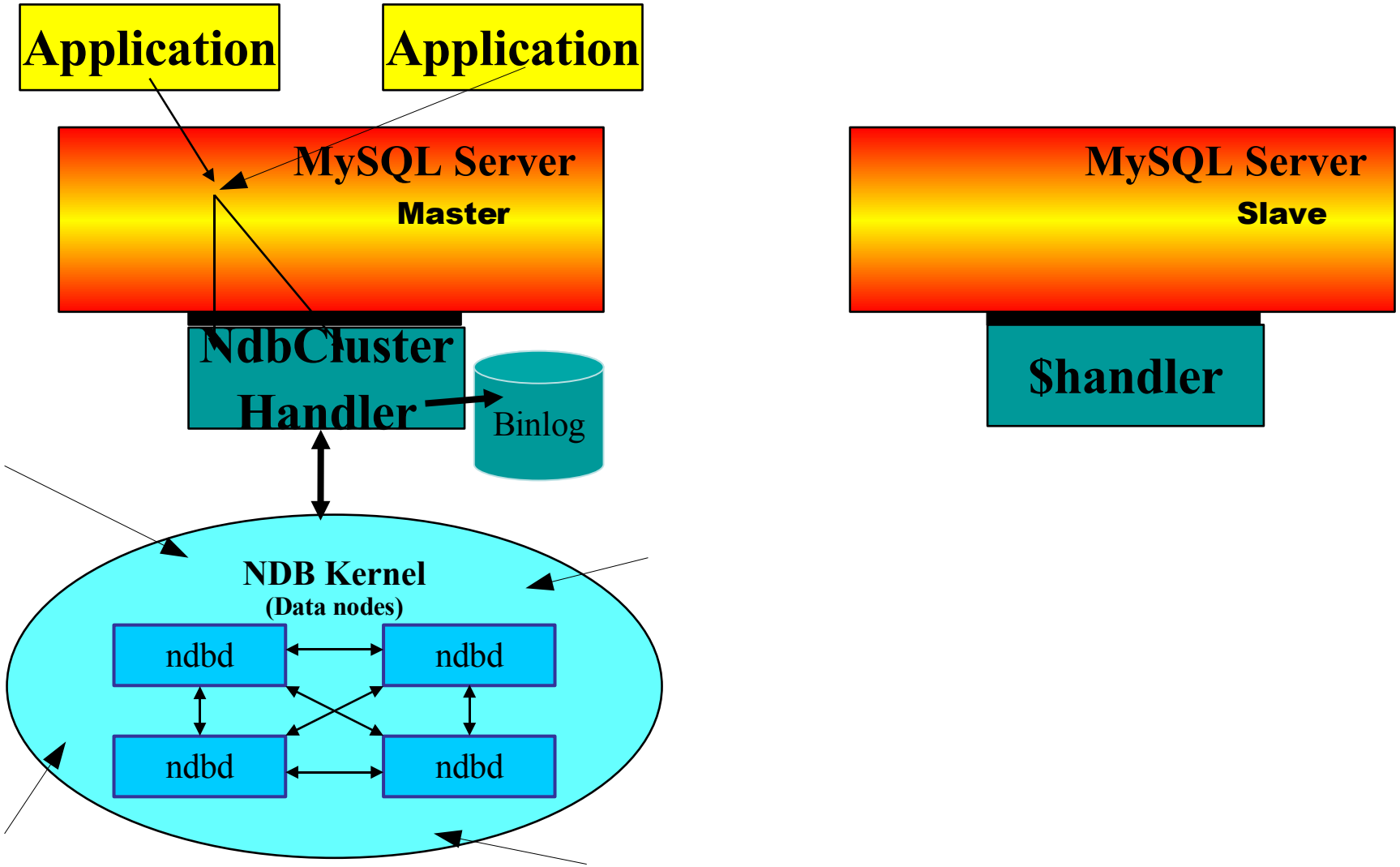
MySQL Replication between Clusters



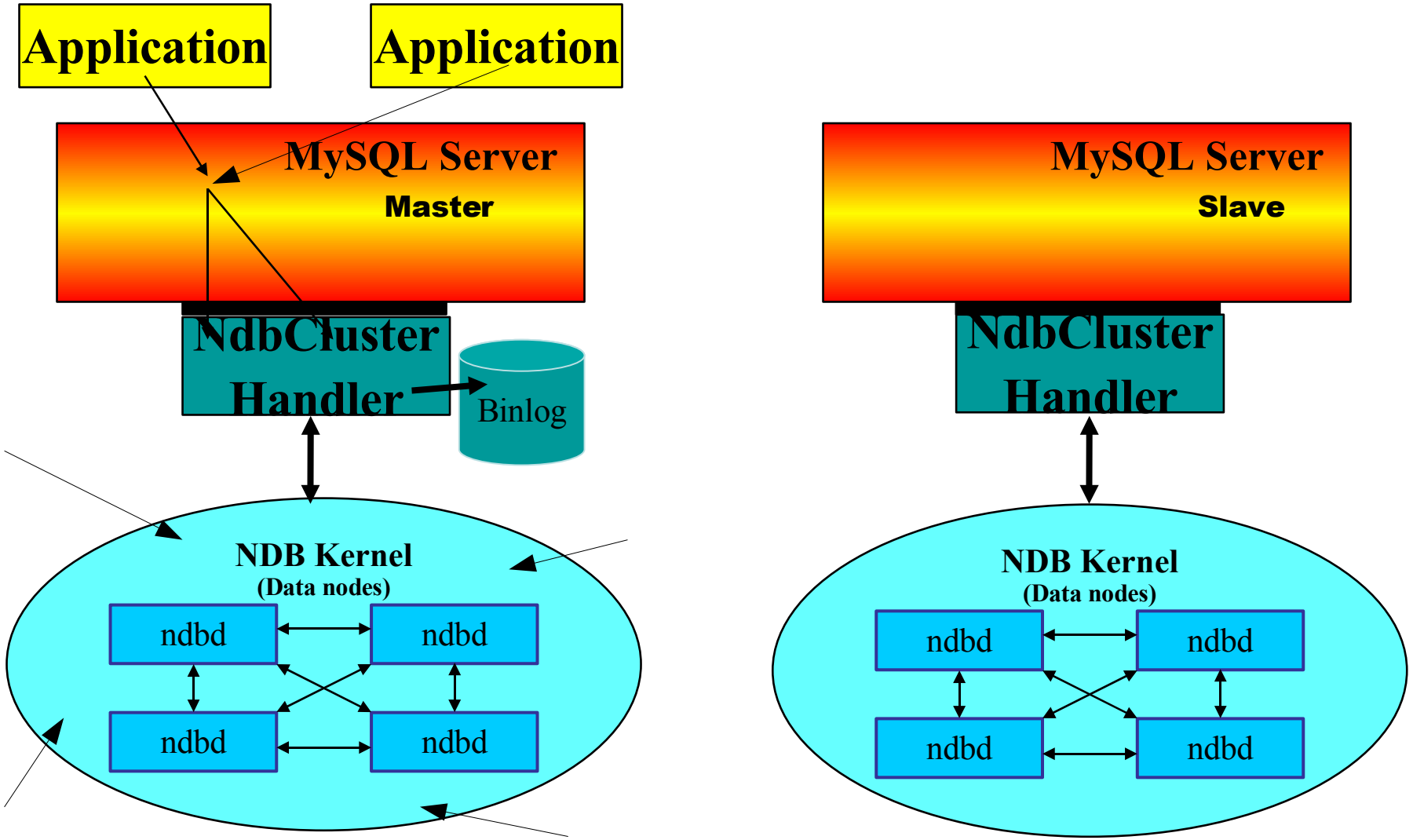
MySQL Replication between Clusters



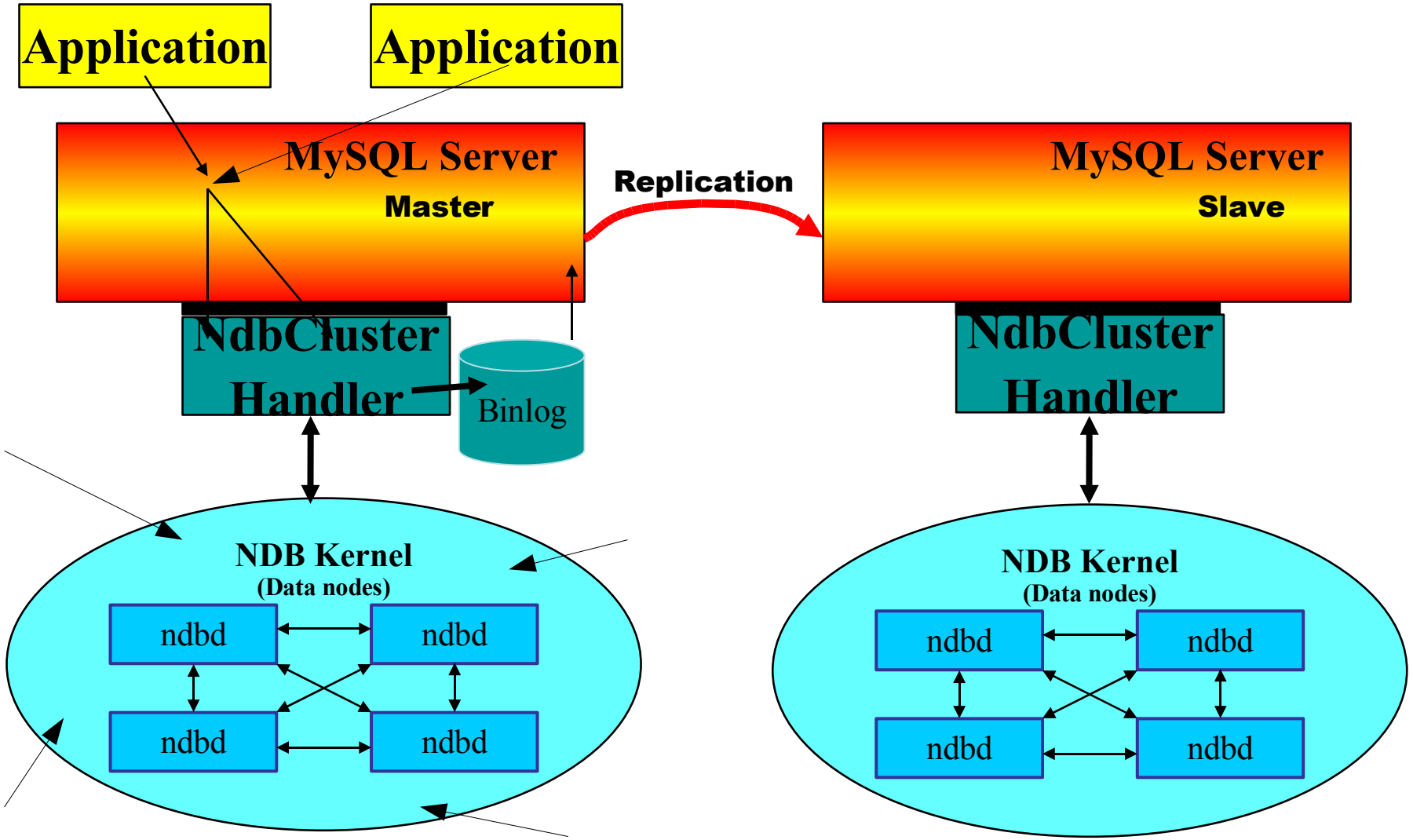
MySQL Replication between Clusters



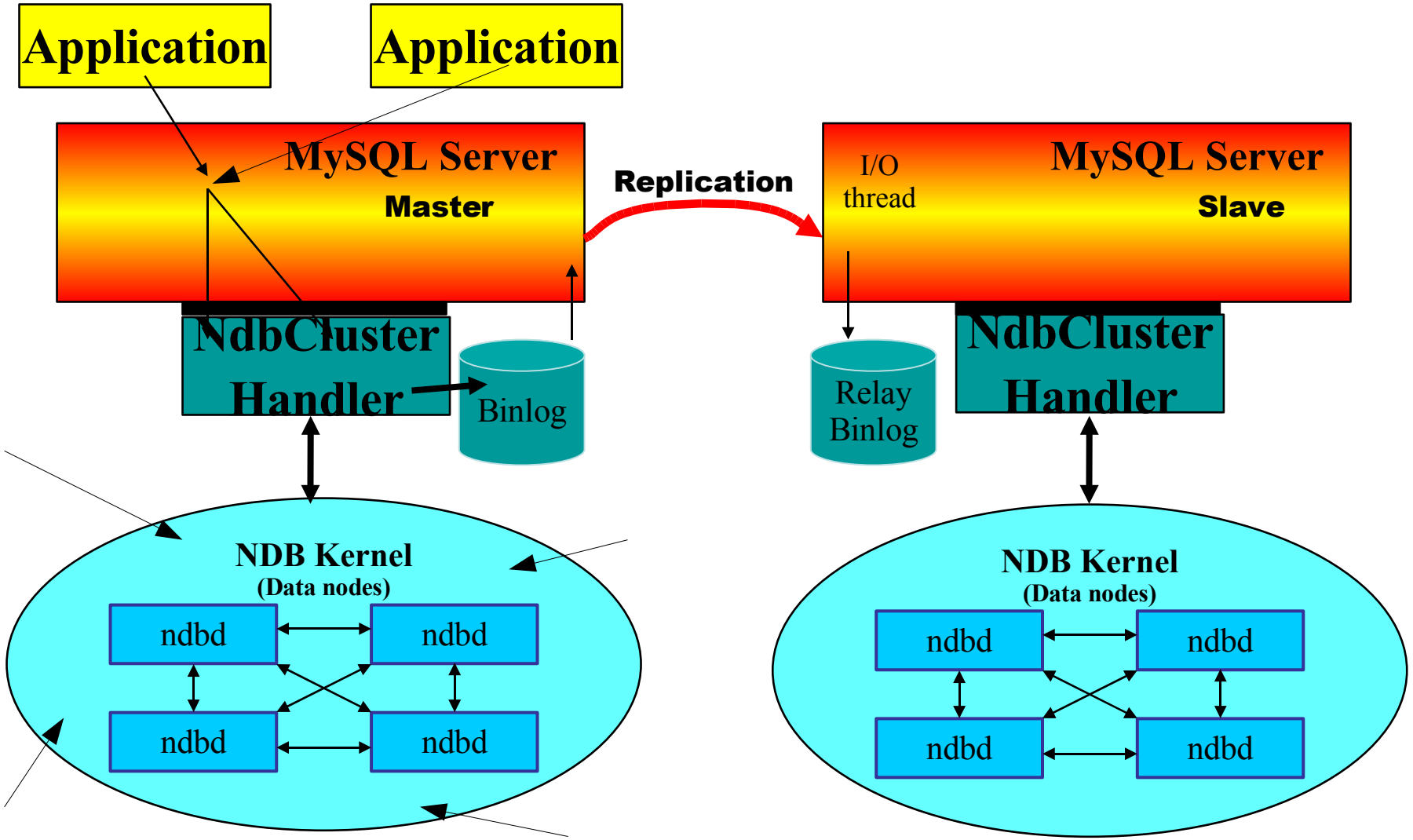
MySQL Replication between Clusters



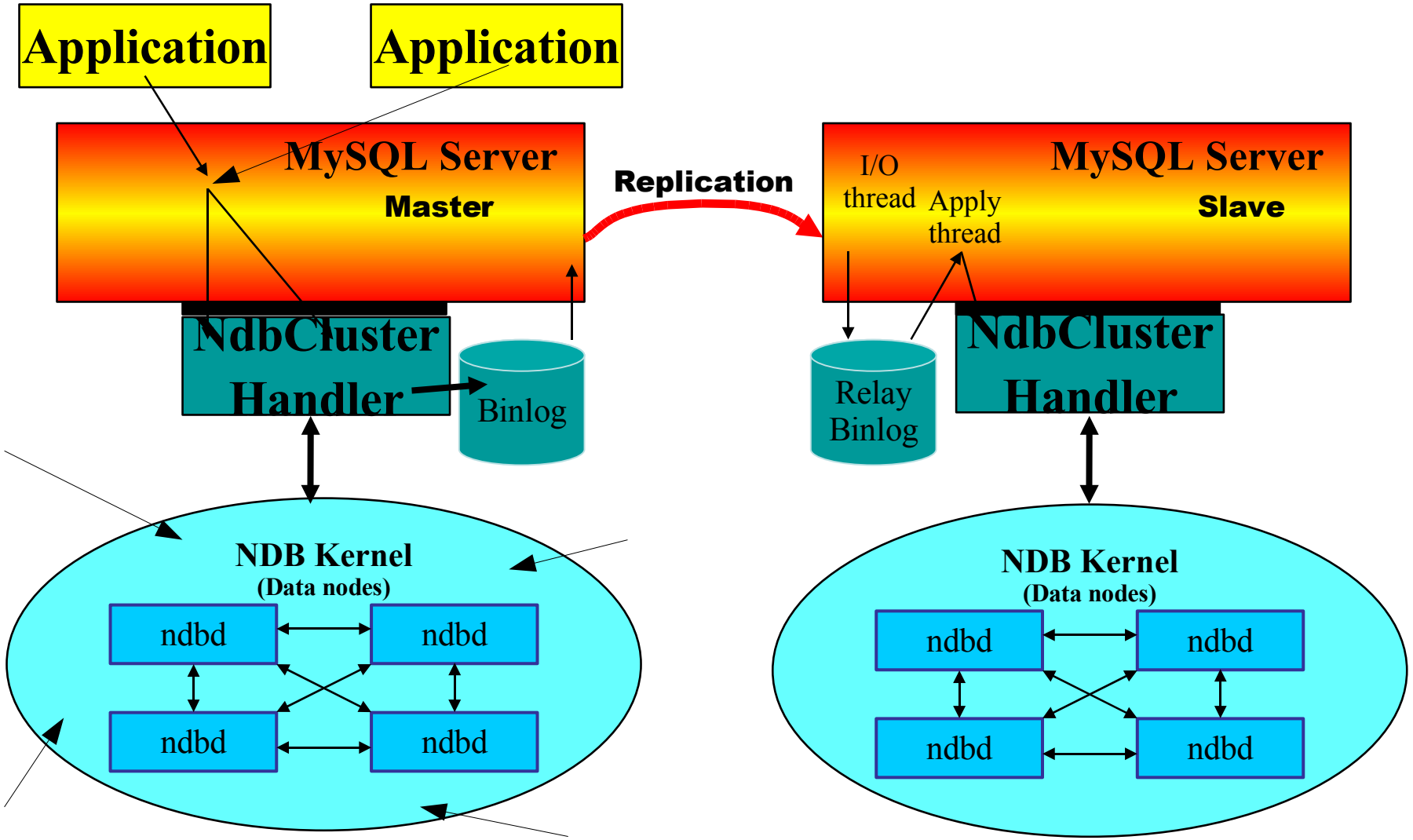
MySQL Replication between Clusters



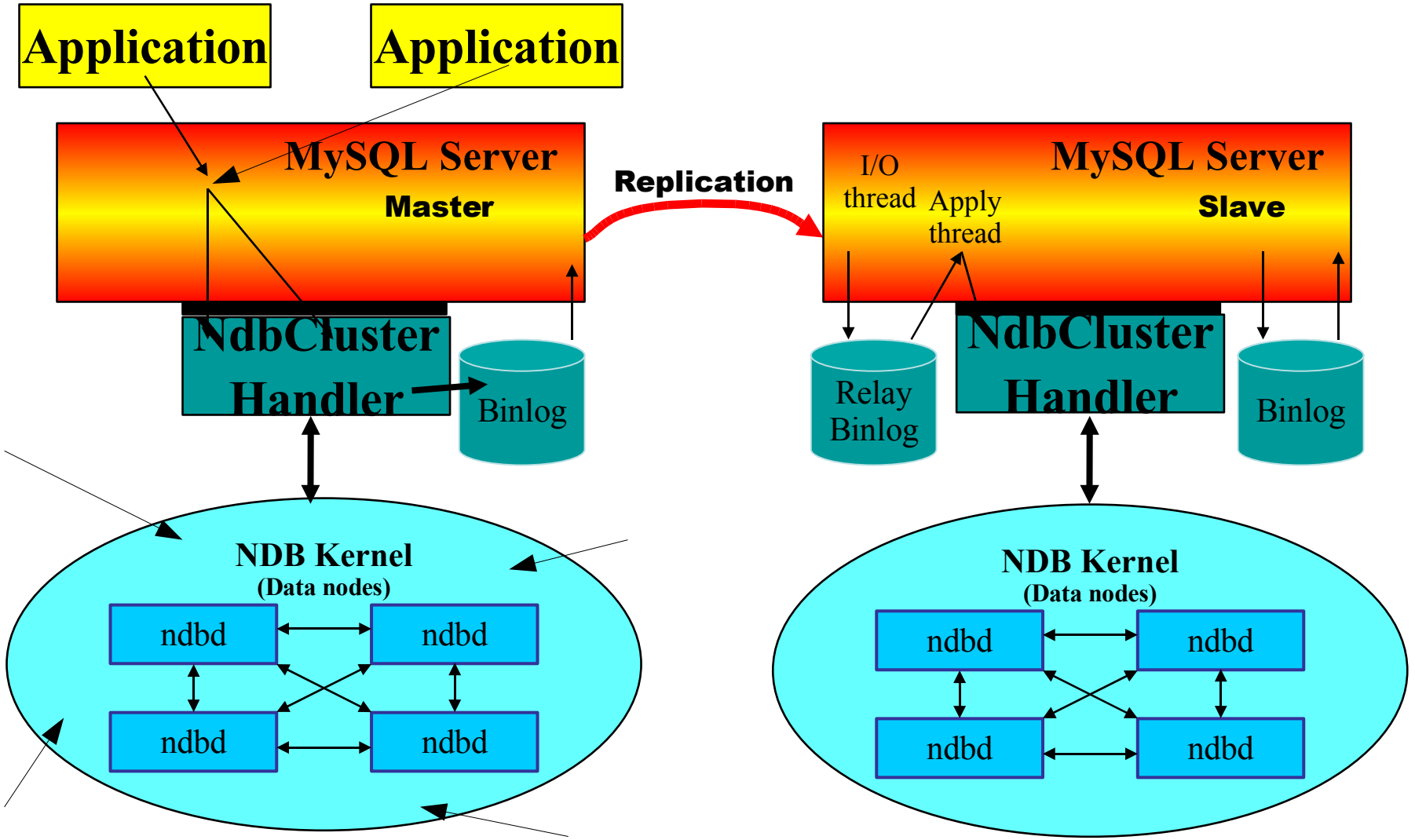
MySQL Replication between Clusters



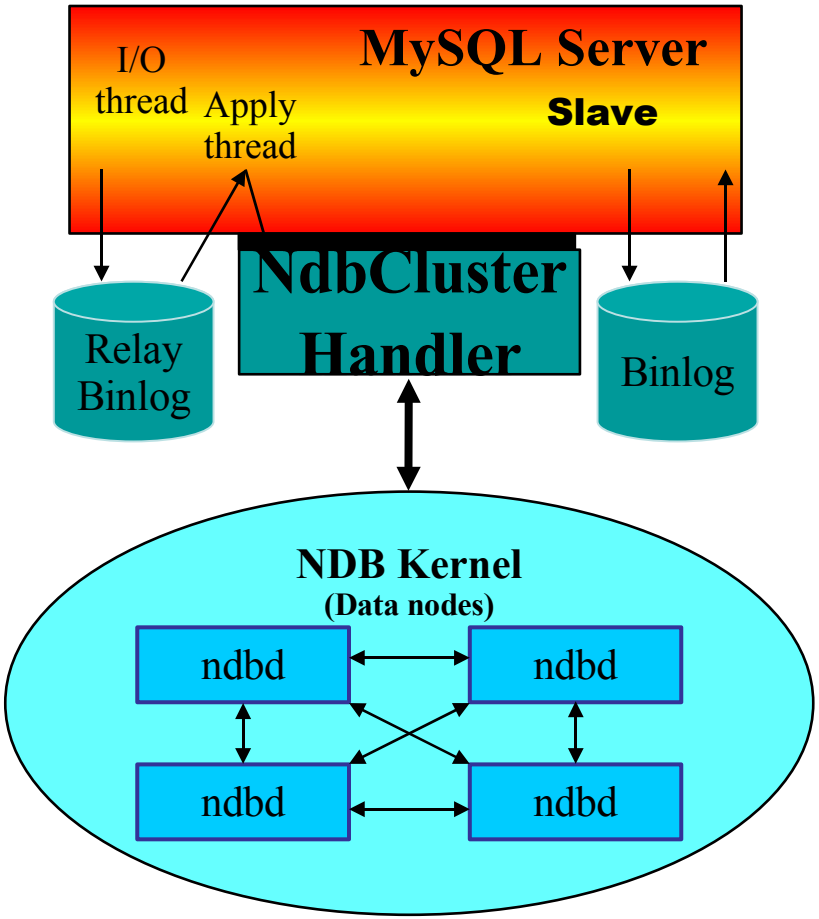
MySQL Replication between Clusters



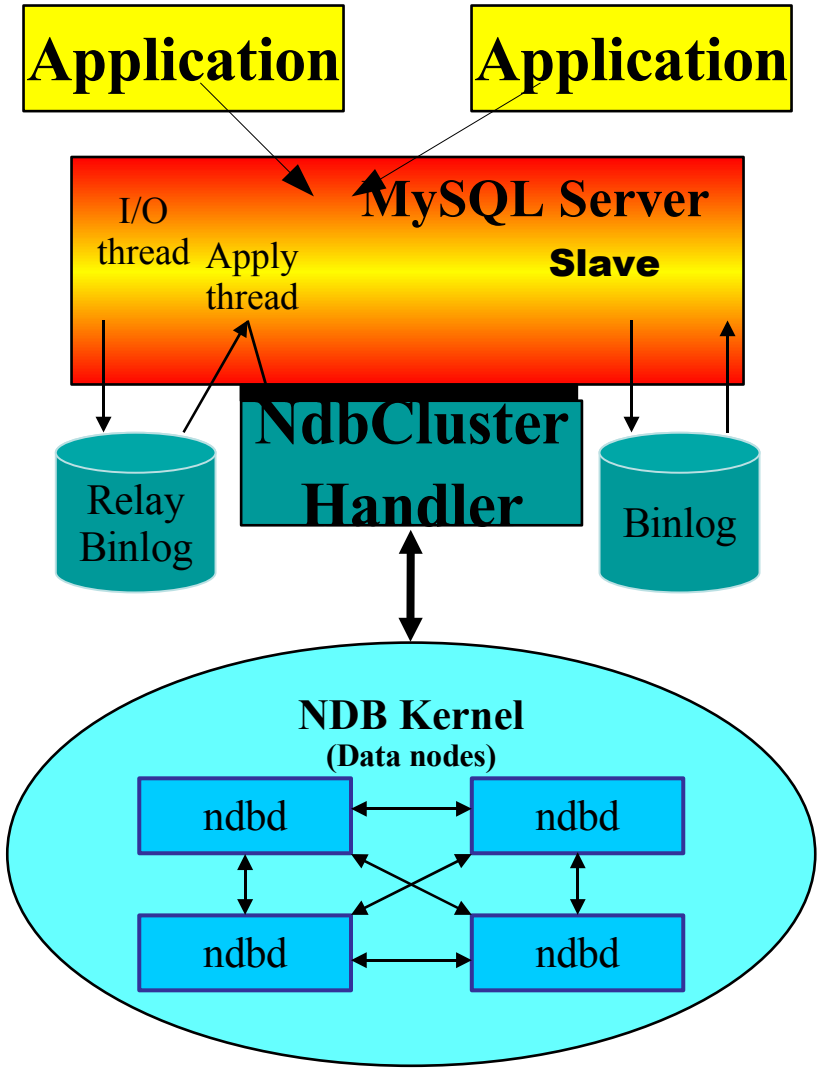
MySQL Replication between Clusters



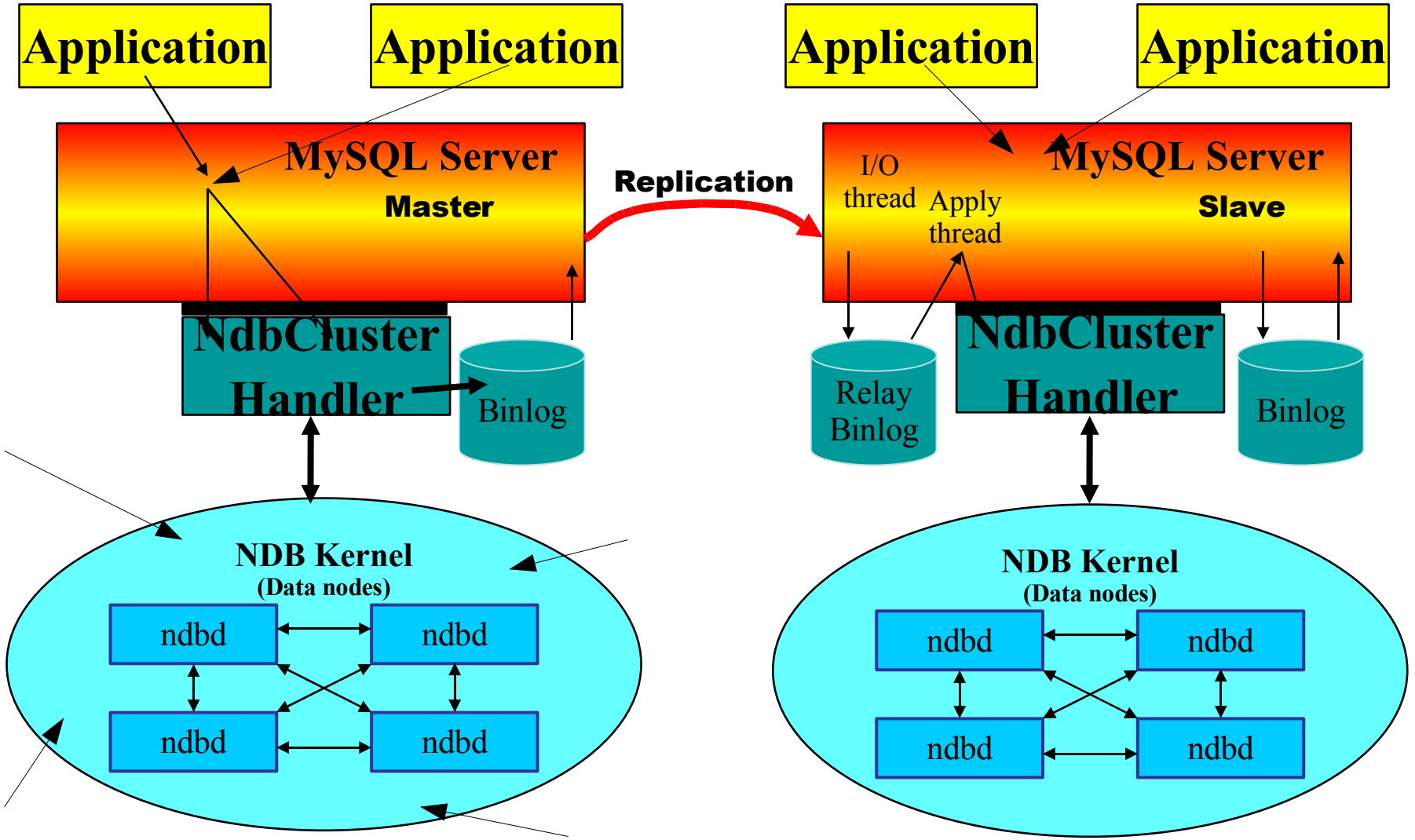
MySQL Replication between Clusters



MySQL Replication between Clusters



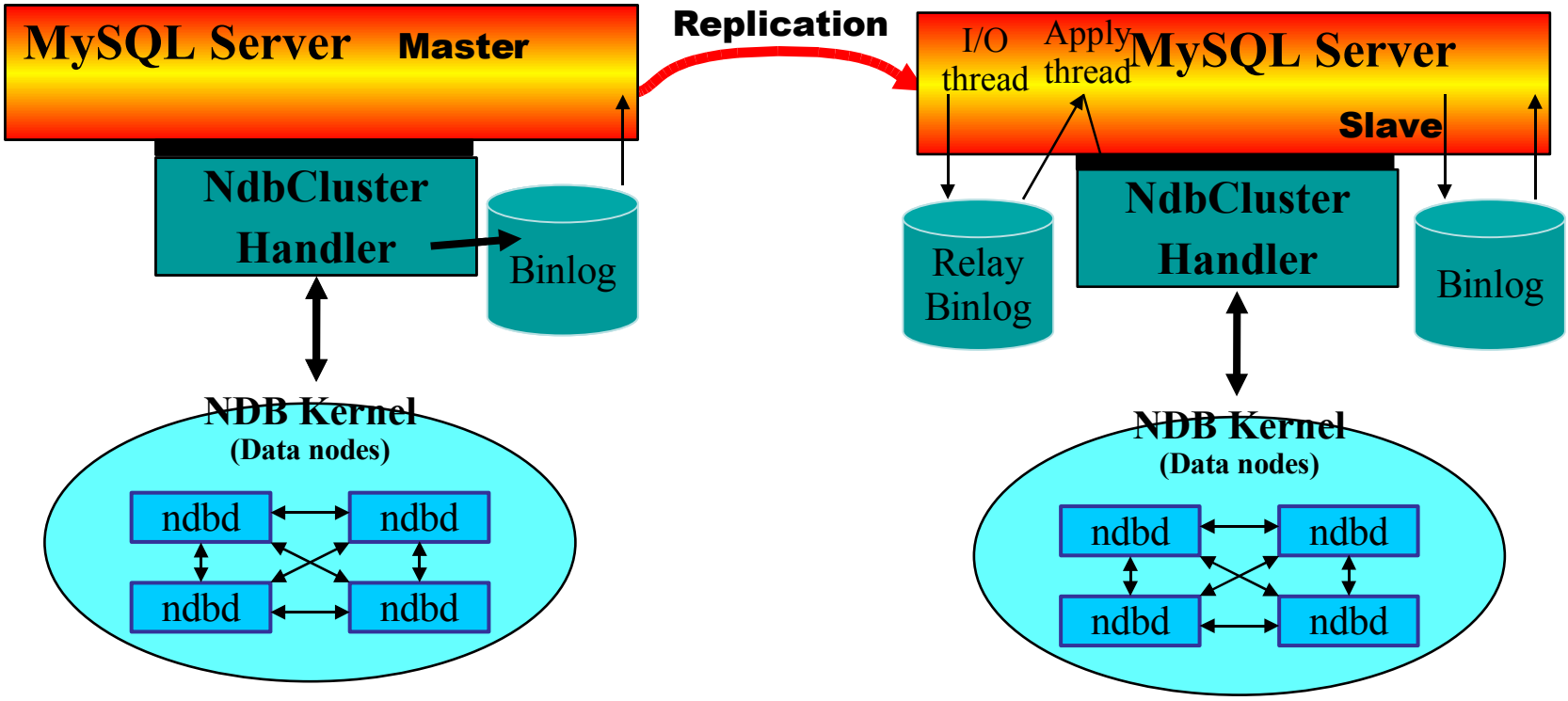
MySQL Replication between Clusters



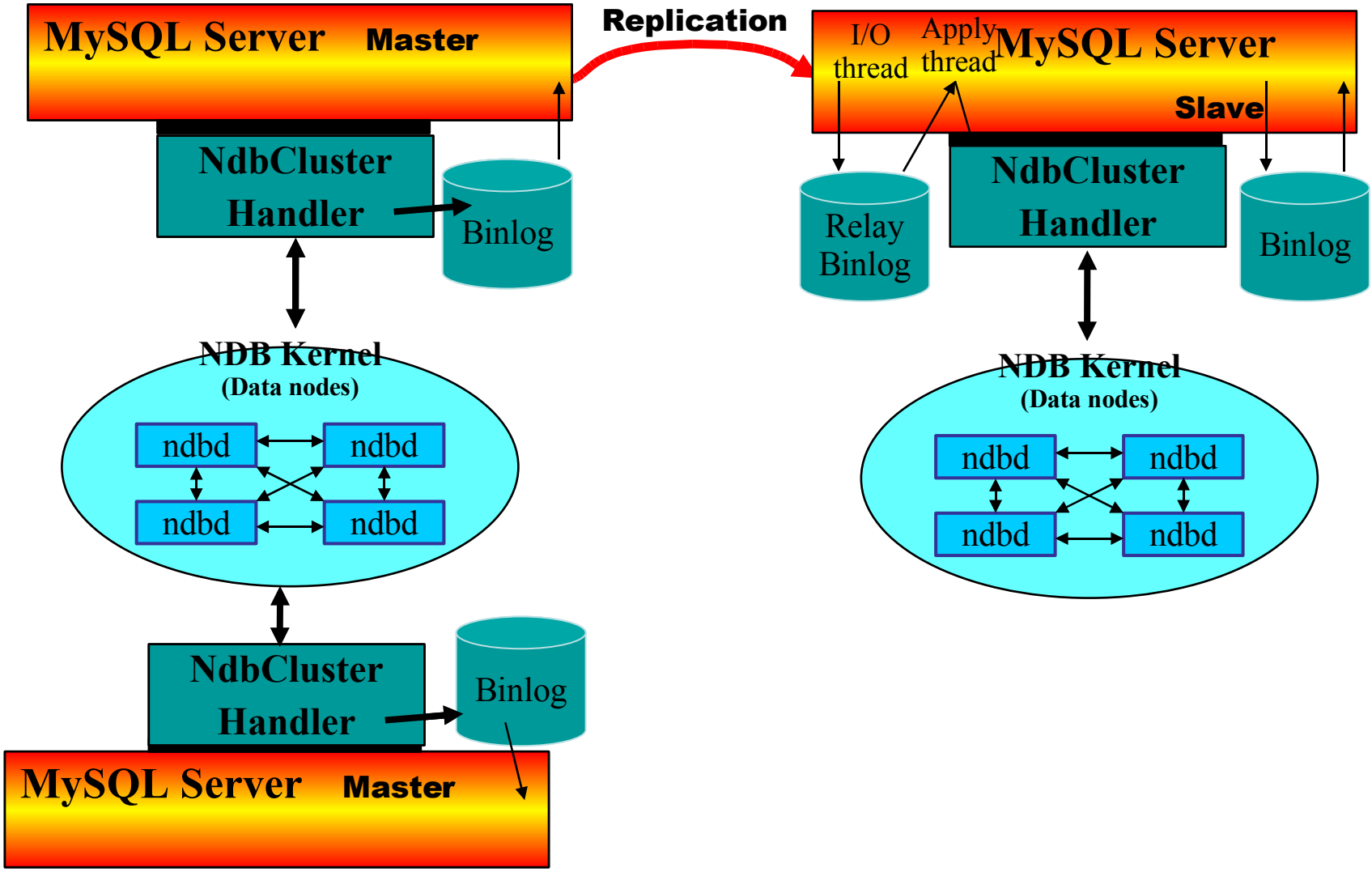
One thing...

Who spotted the single point of failure?

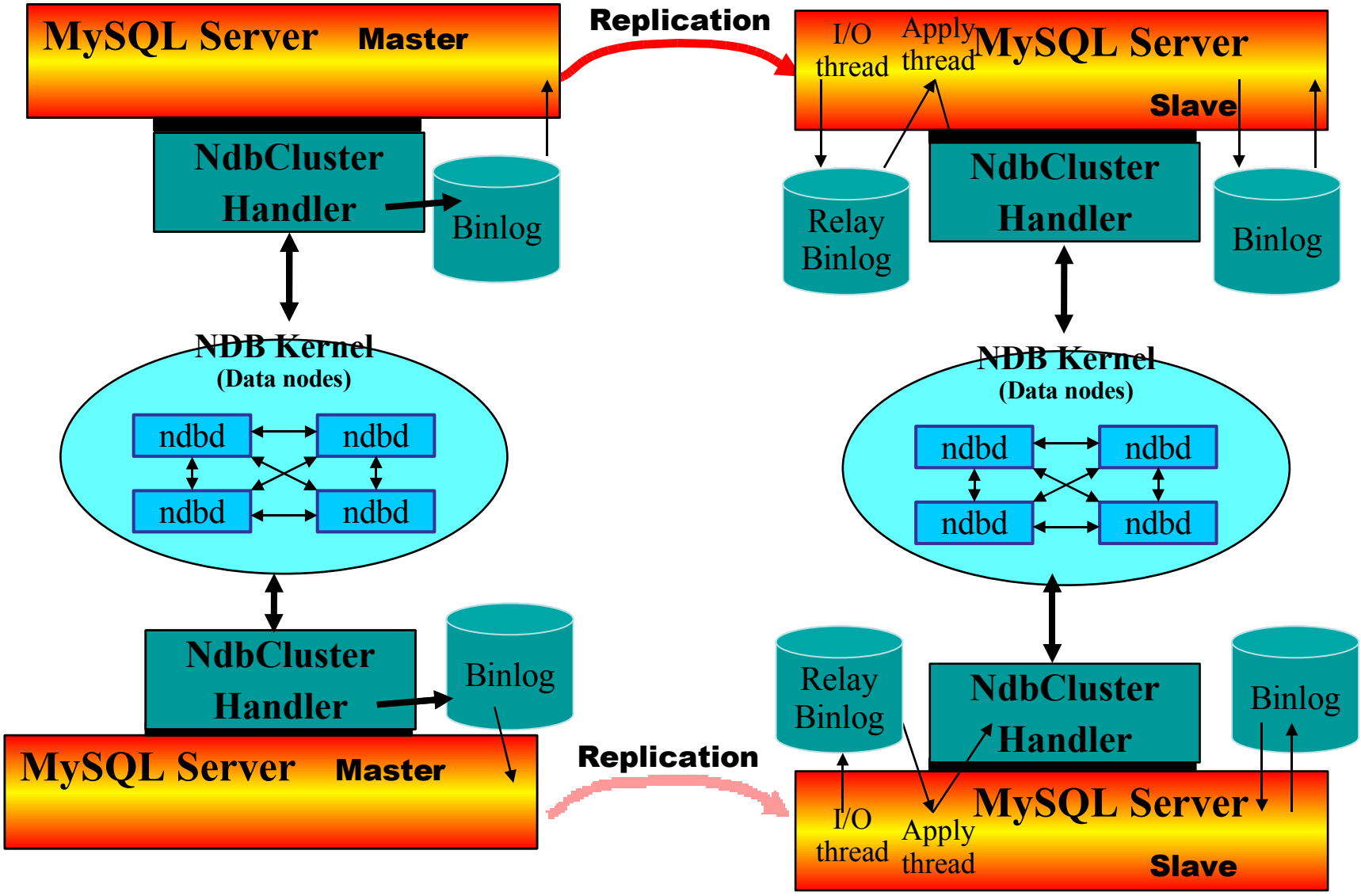
Redundant Replication Channels



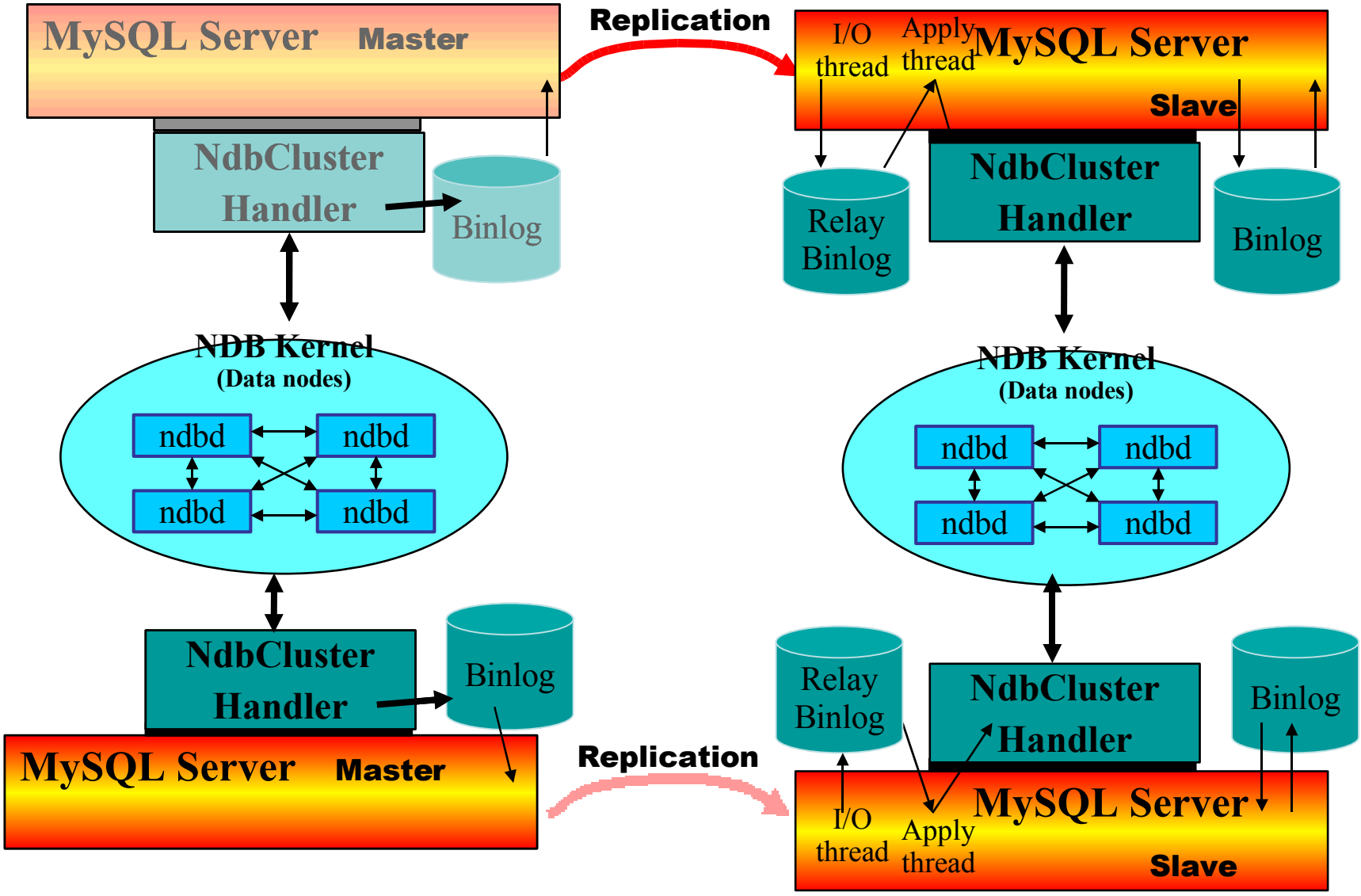
Redundant Replication Channels



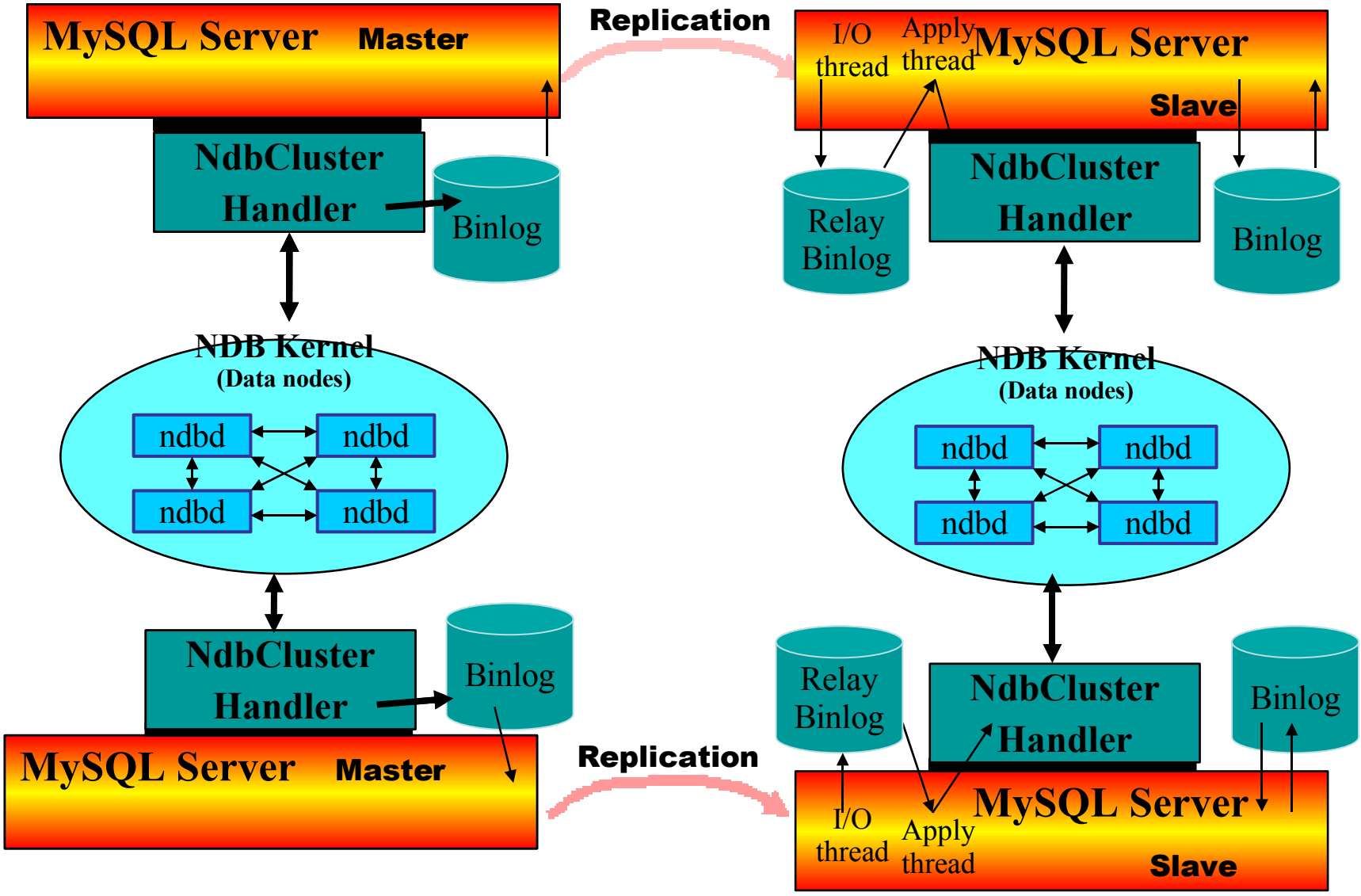
Redundant Replication Channels



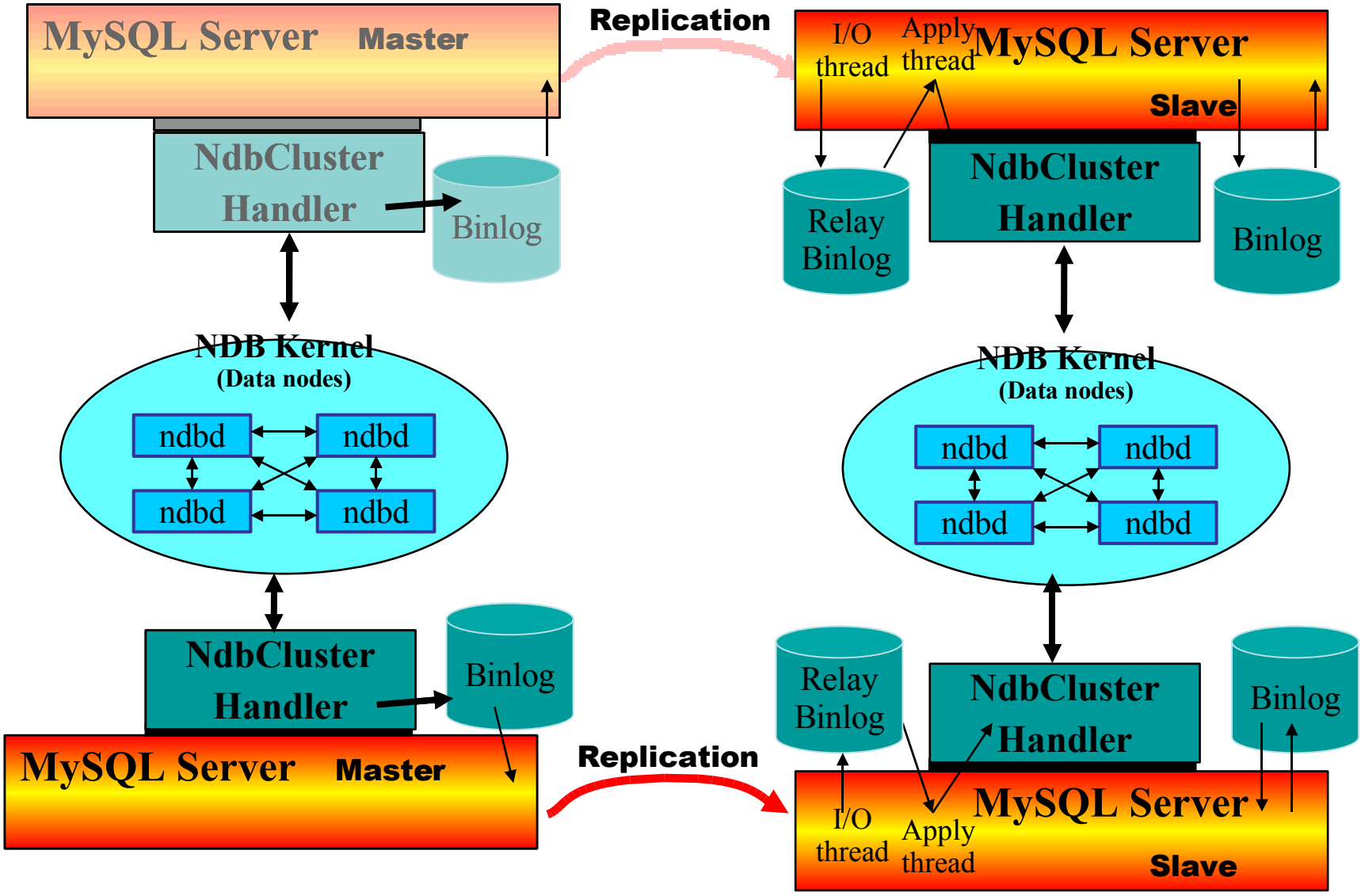
Redundant Replication Channels



Redundant Replication Channels



Redundant Replication Channels



So,

THAT'S COOL!

Not just Cool,

It's SHINY.

How do I make fail over happen?

But first...

Epoch

- A point of synchronization in the cluster
- Everybody agrees on what transactions are disk persistent
- In case of system crash, this is where we'll recover to.

Okay, but fail over?

Currently manual,

Currently manual,
but only four simple steps

STEP 1

- Find out where the Slave is up to
- In the binary log produced by the injector
  - each Global Check Point (epoch) is a transaction
  - Where we are up to is recorded on the slave
- The `cluster` database tracks these things
  - here, the `apply_status` table.
    - Which has two columns: `server_id` and `epoch` (both integers)
    - and is `ENGINE=ndb`
      - so is available everywhere!
- So, we
  - `mysqlS`> SELECT @latest:=MAX(epoch) from cluster.apply_status;`
  - possibly with a WHERE clause for server_id

STEP 2

- Find the binlog position for this epoch
- The `cluster.binlog_index` table will help us here
  - It maps binlog position to GCI
  - and tells us the number of INSERT, UPDATES, DELETES and SCHEMAOPS per GCI
  - Is MyISAM and is per-master
- So, we run the query (@latest from slave)
- **mysqlM`>**

```
SELECT @file:=SUBSTRING_INDEX(File, '/', -1),
       @pos:=Position
FROM cluster.binlog_index
WHERE epoch > @latest
ORDER BY epoch ASC LIMIT 1;
```

STEP 3

- Synchronize the second channel
  - i.e. change the master
- Run the query (using @file and @pos from last query)
  -
```
mysqlS`> CHANGE MASTER TO
         MASTER_LOG_FILE='@file',
         MASTER_LOG_POS=@pos;
```

STEP 4

- `mysqlS`> START SLAVE;
- No, really, that's it.

Limitations

- Fail over of replication channels is manual
  - can be scripted
- Since all updates are through one injector thread, there is a limit
  - this limit is much less than what you can pump through a good cluster
  - We are working to overcome this

You can now...

- Have a 99.999% uptime cluster
  - with great performance
  - and a cool name

You can now...

- Have a 99.999% uptime cluster
  - with great performance
  - and a cool name
- Have replication between it and another Cluster (or single server)
  - Load balancing
  - Geographical redundancy

You can now...

- Have a 99.999% uptime cluster
  - with great performance
  - and a cool name
- Have replication between it and another Cluster (or single server)
  - Load balancing
  - Geographical redundancy
- Redundant replication channels between these setups
- Redundancy up the wazoo!

Disk Data

Options for Adding Disk Data to NDB

1. Put Existing disk engine into NDB kernel

Options for Adding Disk Data to NDB

1. Put Existing disk engine into NDB kernel
2. Careful re-engineering of NDB kernel to add support for disk based attributes

Options for Adding Disk Data to NDB

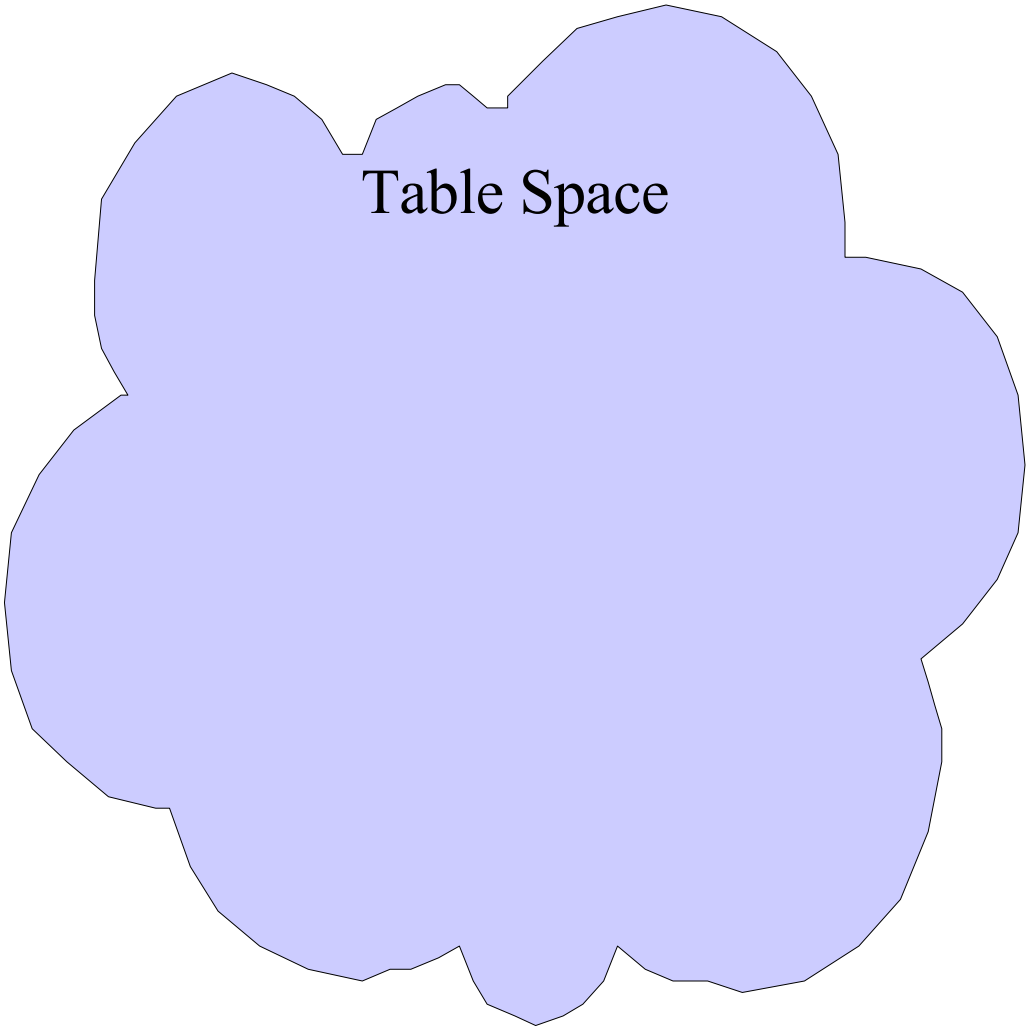
1. Put Existing disk engine into NDB kernel
2. Careful re-engineering of NDB kernel to add support for disk based attributes

Two Phase Implementation

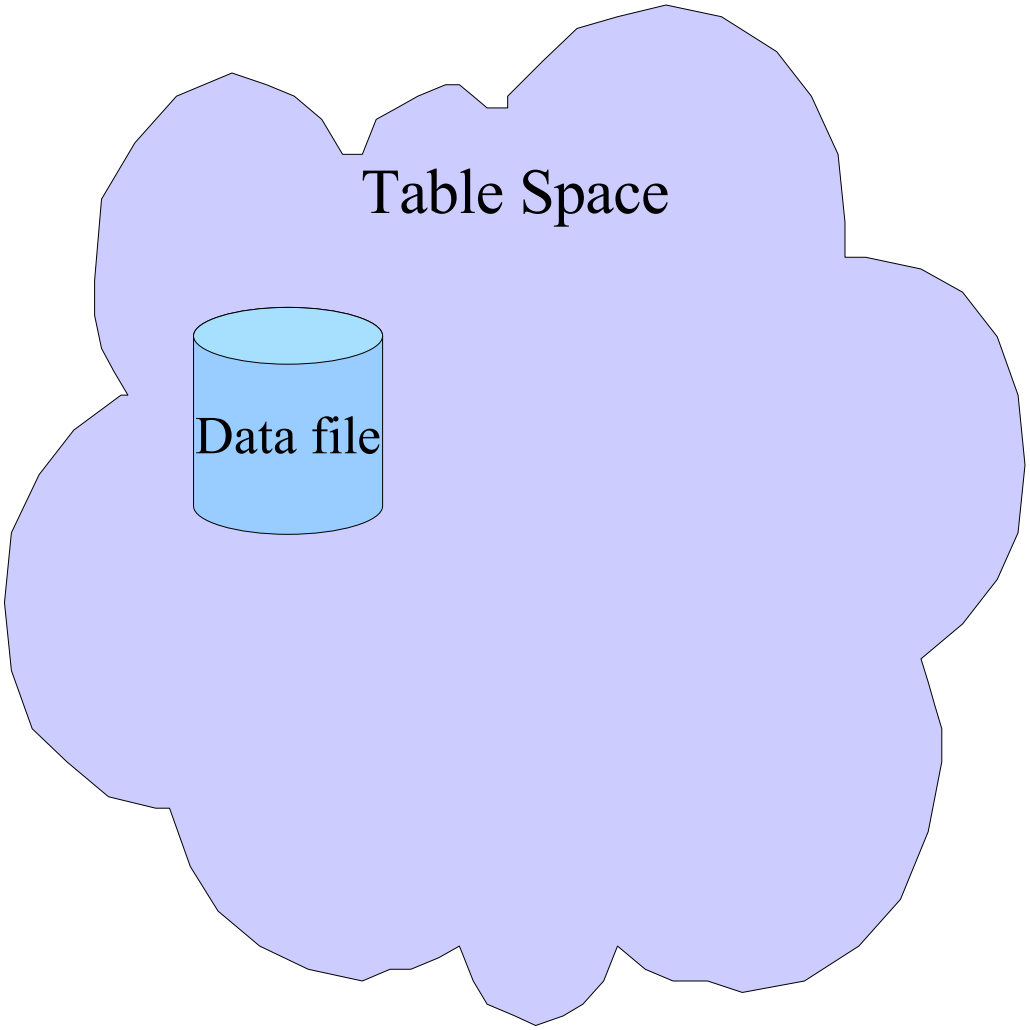
1. Data on disk (5.1)
2. Indexes on disk (future)

A few concepts...

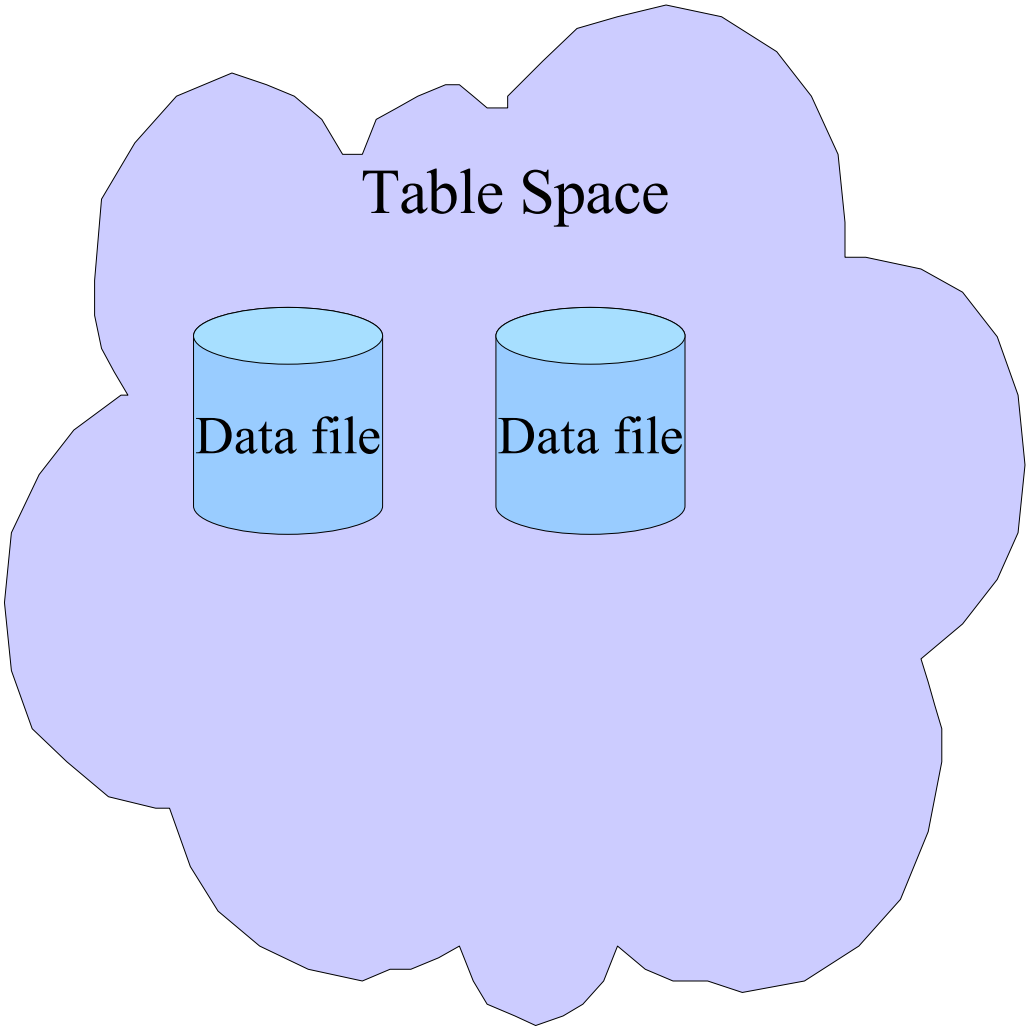
Where we store things



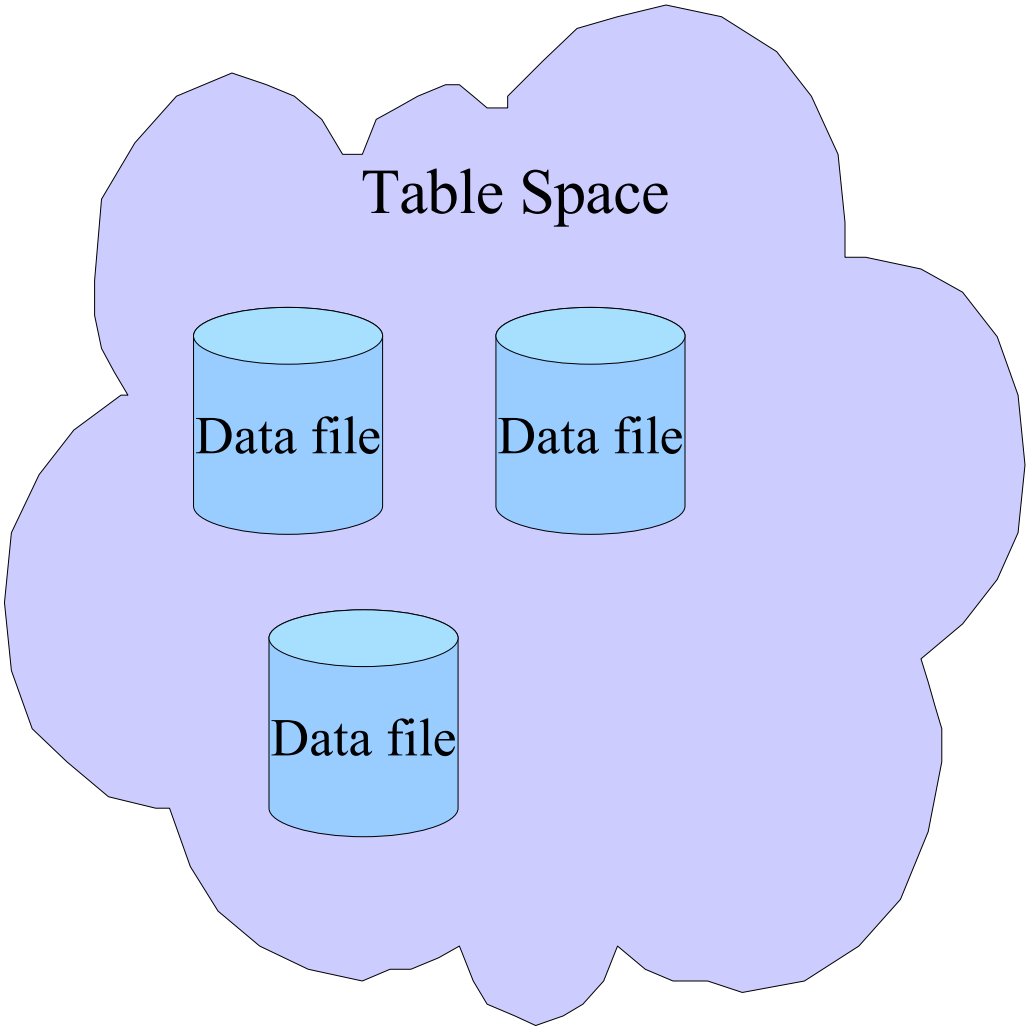
Where we store things



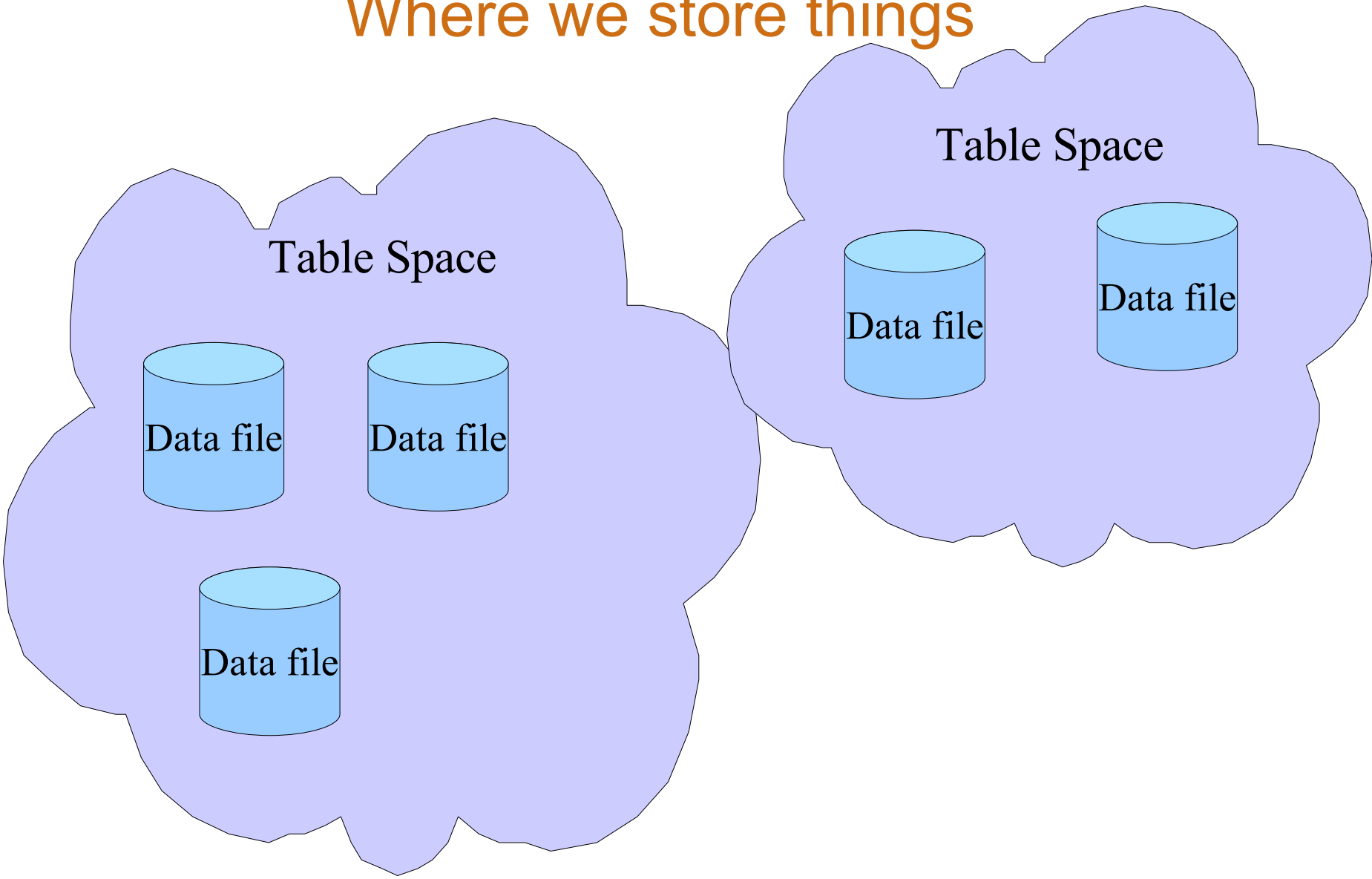
Where we store things



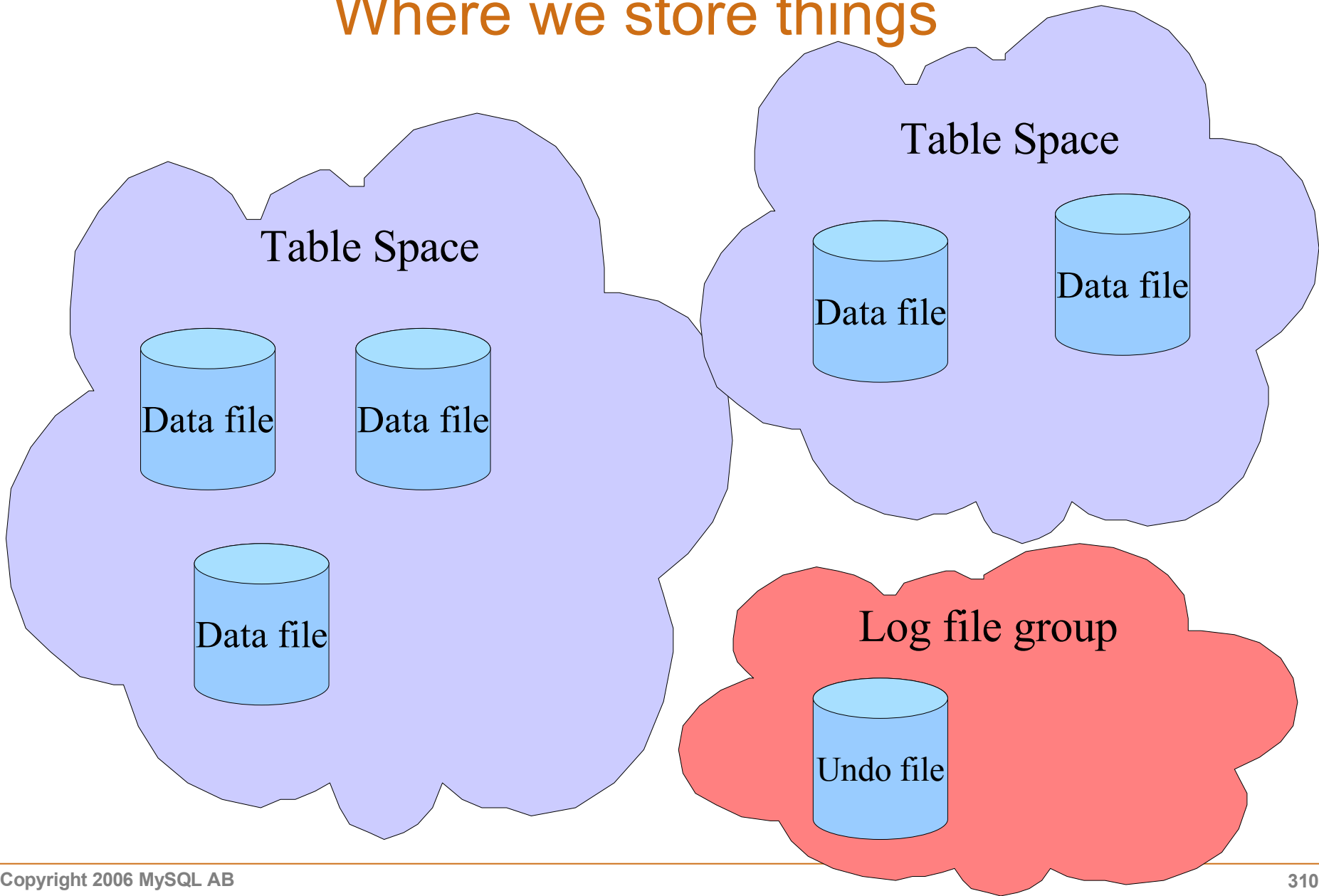
Where we store things



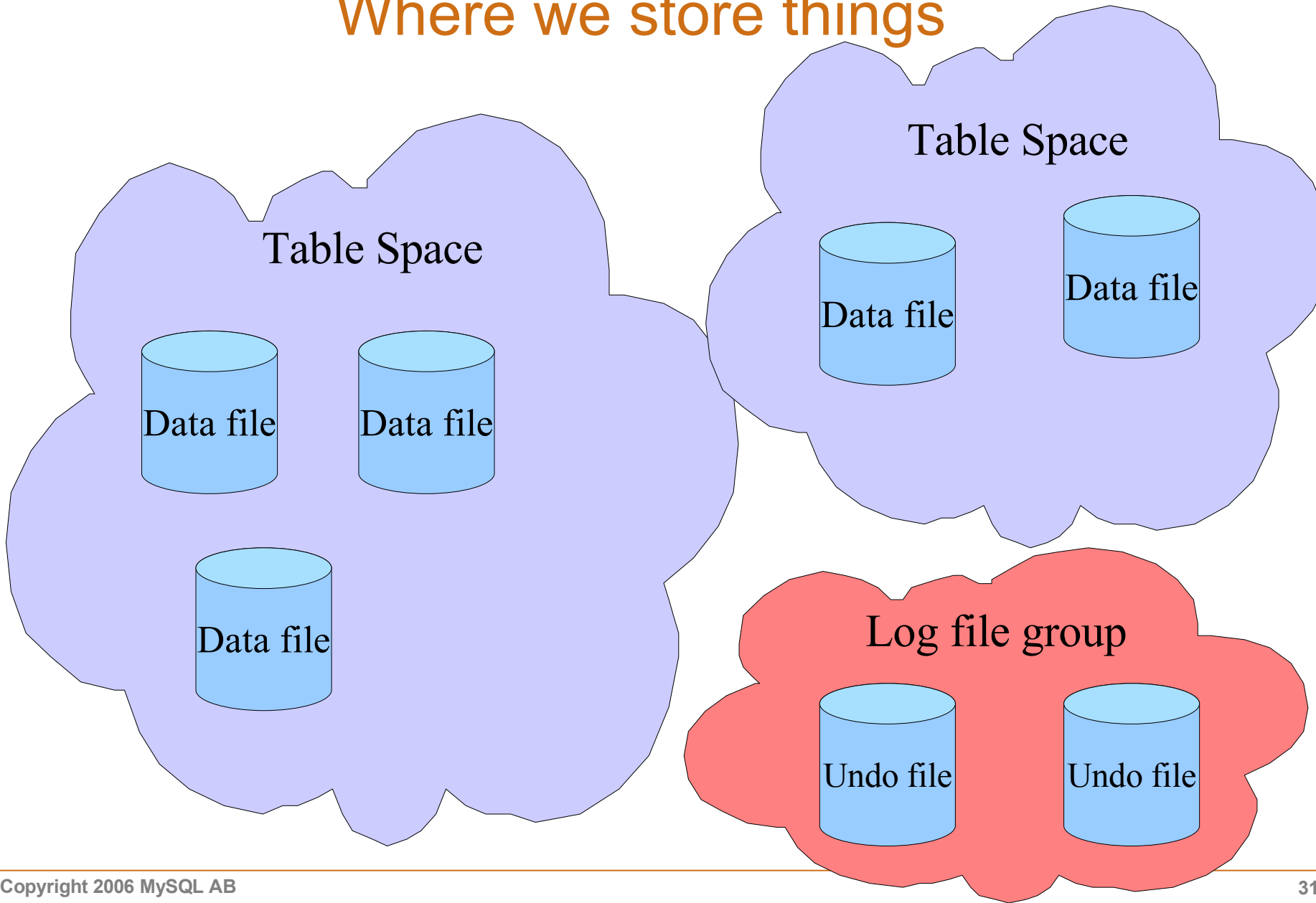
Where we store things



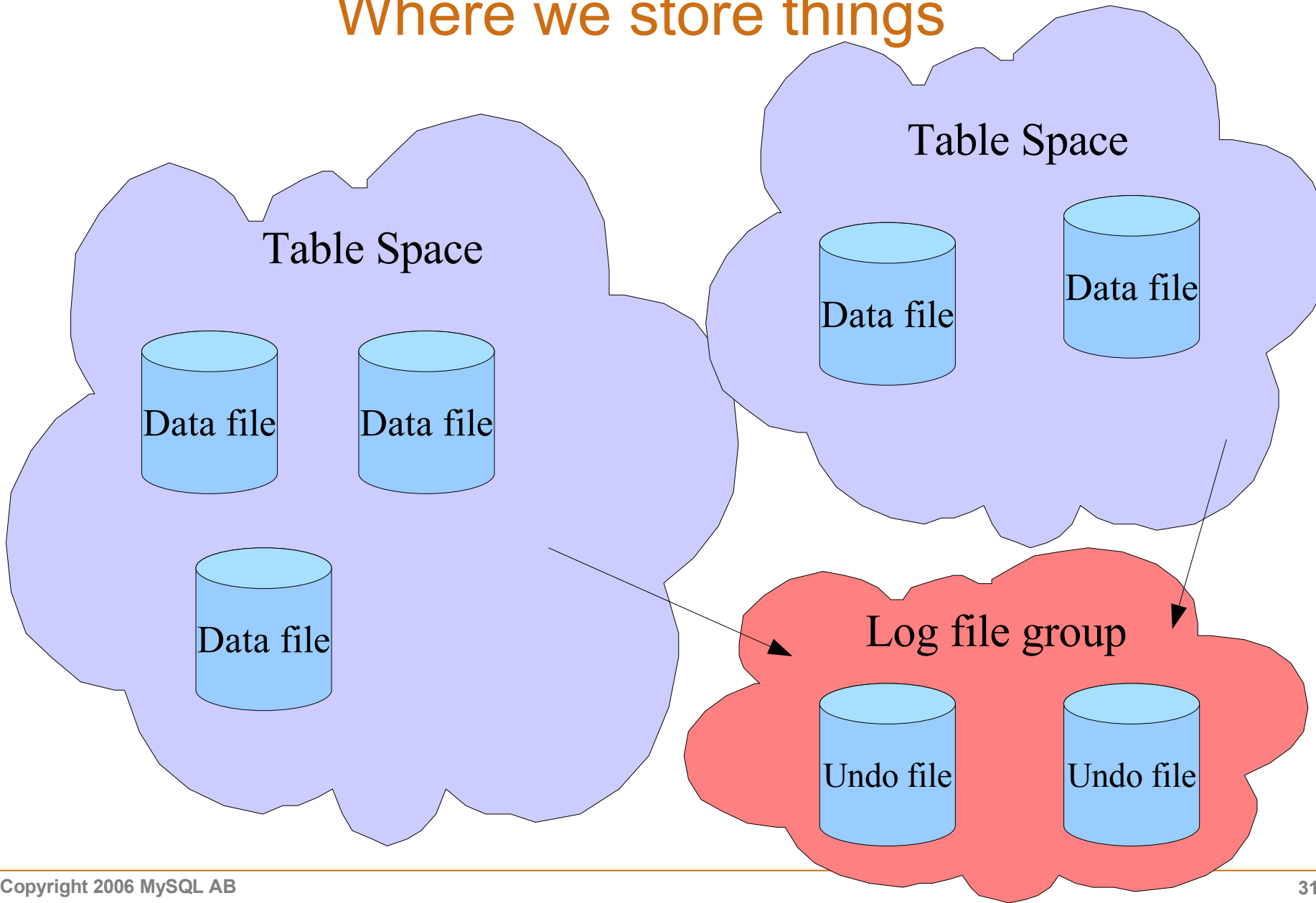
Where we store things



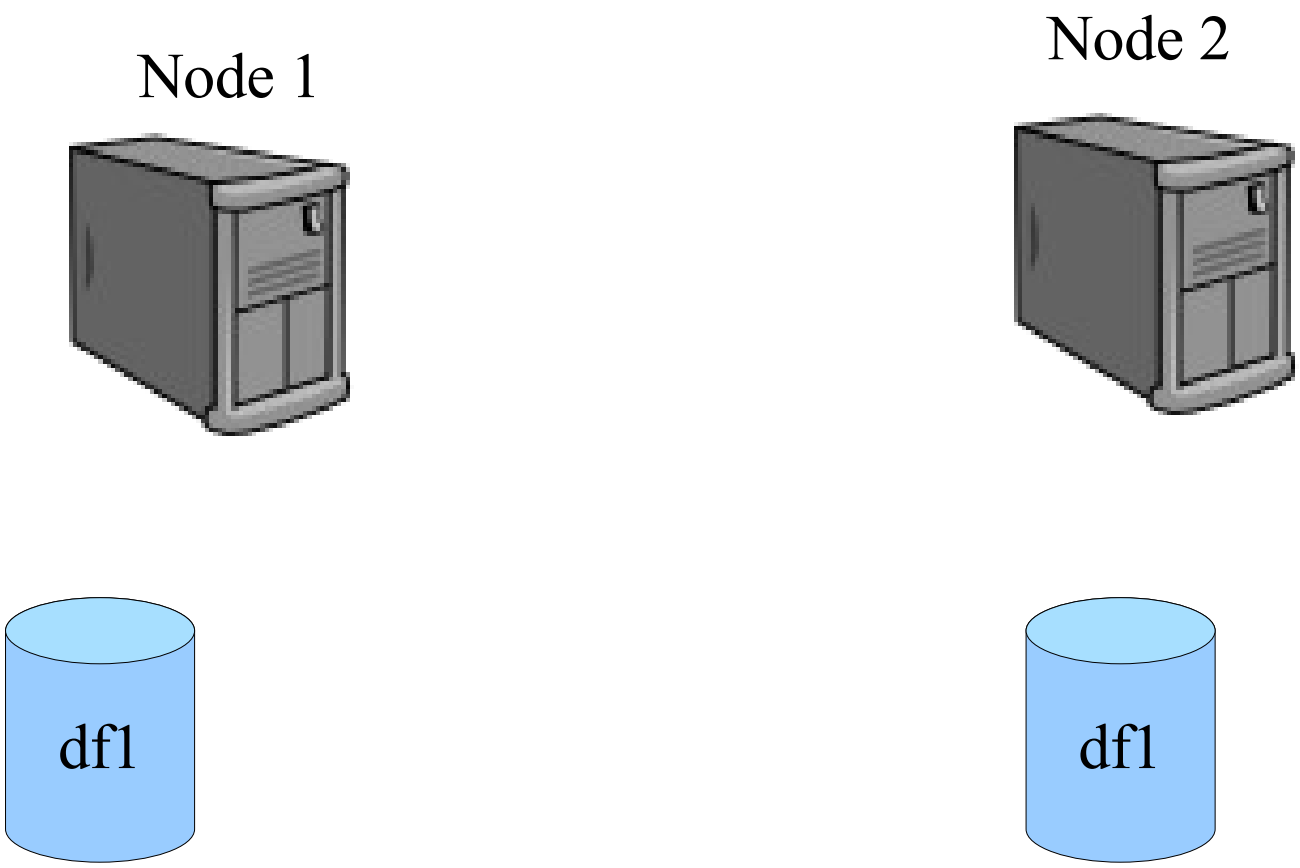
Where we store things



Where we store things



Files are per node



Let's look at the SQL

CREATE LOGFILE GROUP

- CREATE LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo1'
  INITIAL_SIZE 16M
  UNDO_BUFFER_SIZE 2M
  ENGINE=NDB;
- We can add another undo file
- ALTER LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo2'
  INITIAL_SIZE 12M
  ENGINE=NDB;
- We currently don't auto-extend files
  - For more space, add files

CREATE TABLESPACE

- CREATE TABLESPACE ts1
  ADD DATAFILE 'datafile1'
  USE LOGFILE GROUP lg1
  INITIAL_SIZE 32M
  ENGINE=NDB;
- We can add another datafile too
- ALTER TABLESPACE ts1
  ADD DATAFILE 'datafile2'
  INITIAL_SIZE 48M
  ENGINE=NDB;
- We currently don't auto-extend
  - So just add another file

CREATE TABLE

- CREATE TABLE t1 (
  pk1 INT NOT NULL PRIMARY KEY,
  b INT NOT NULL,
  c INT NOT NULL)
  TABLESPACE ts1 STORAGE DISK
  ENGINE=NDB;
- b and c will be stored on disk
- pk1 in memory (as it's indexed)

ALTER TABLE (add index)

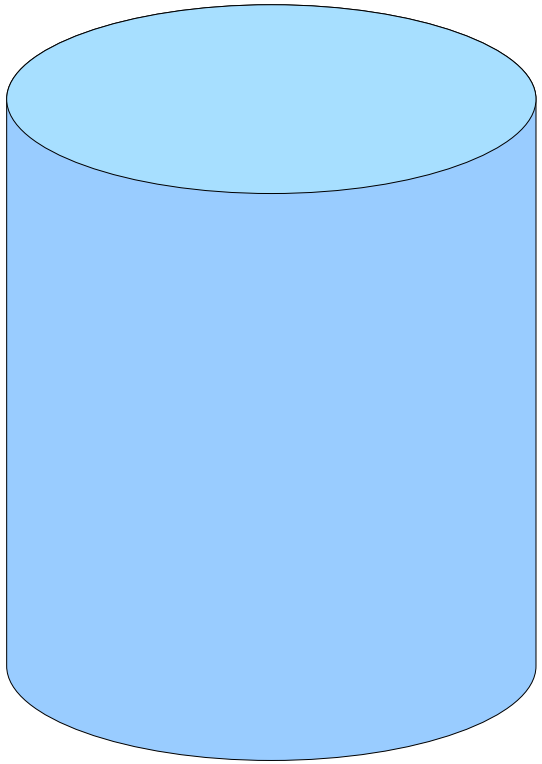
- If you add an index, fields moved to MM

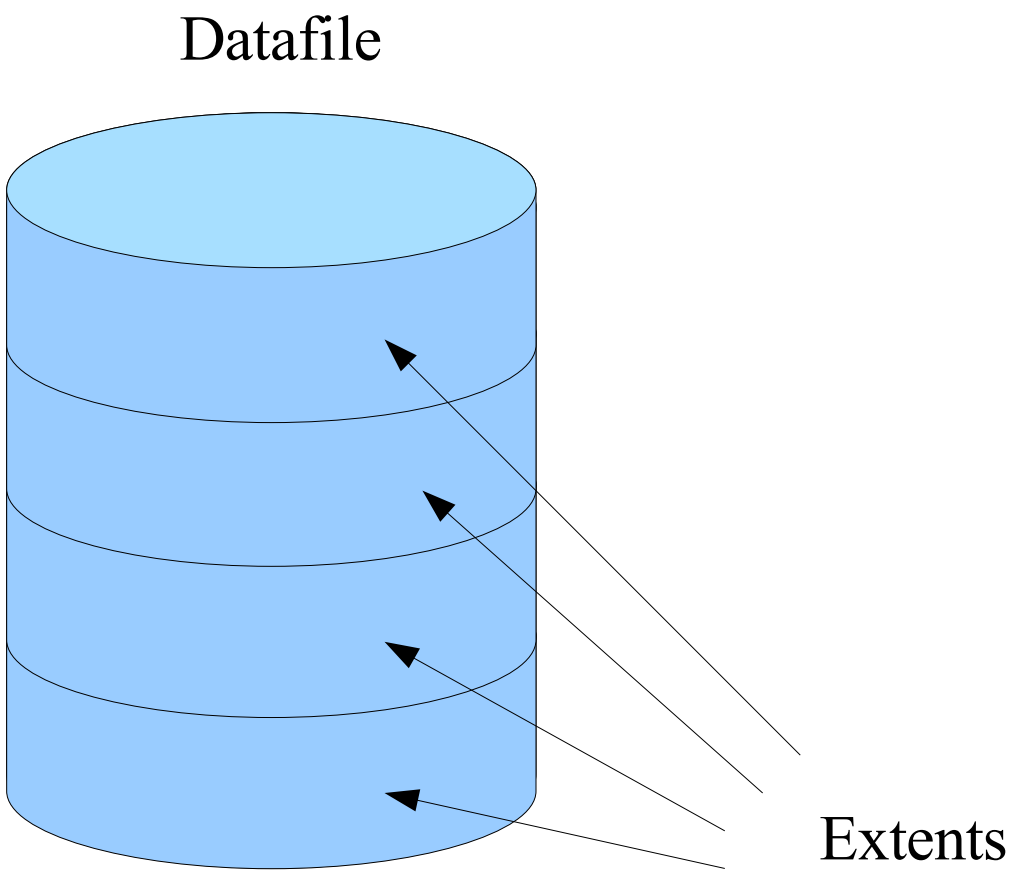
Checking free space

- INFORMATION_SCHEMA.FILES
  - Ongoing debate on if this is going to stay in INFORMATION_SCHEMA or move to performance schema.
  - The actual schema of the table is pretty decided though
  - Currently NDB only
    - Pressure your storage engine developers to add code for theirs!
- Does require some knowledge about how space is allocated to tables

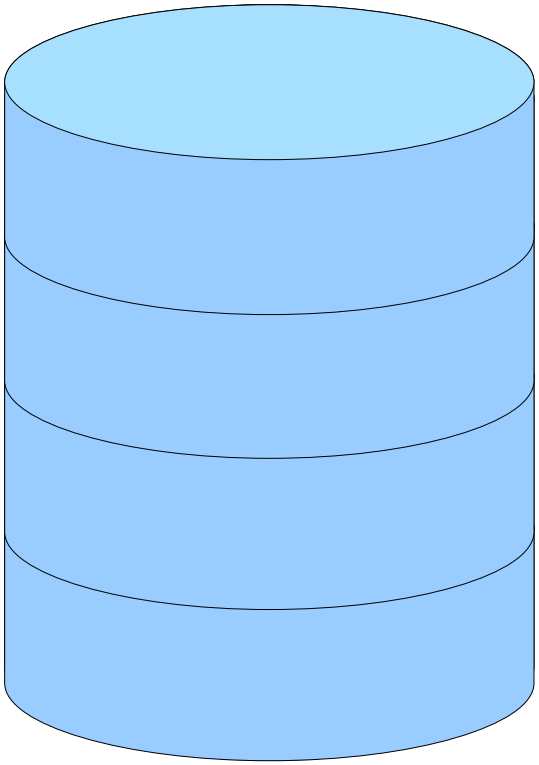
How we allocate disk space

Datafile

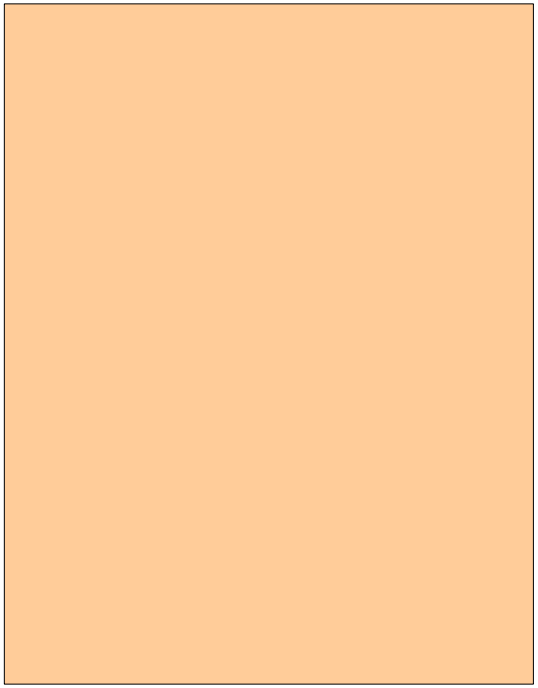




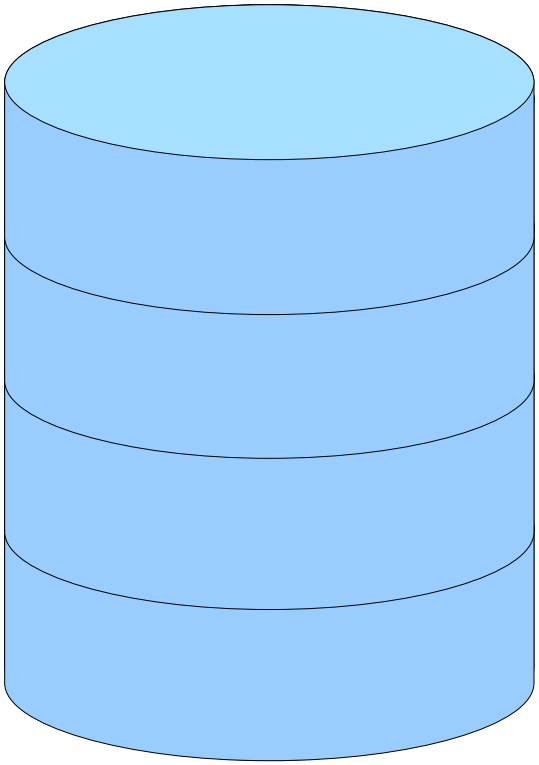
Datafile



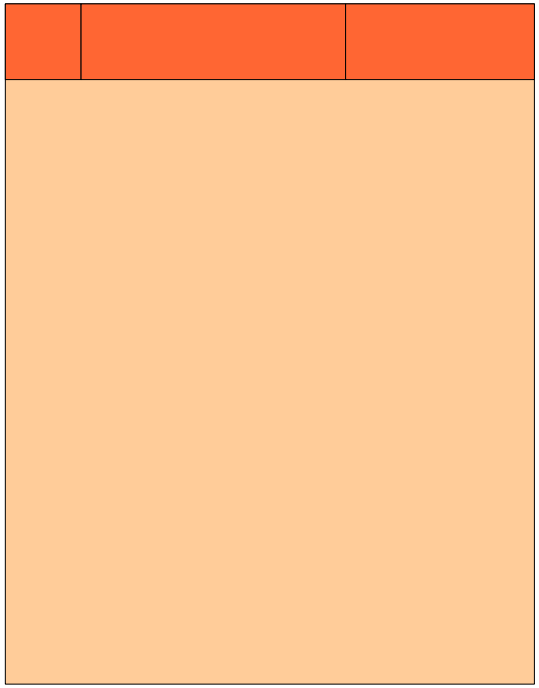
t1

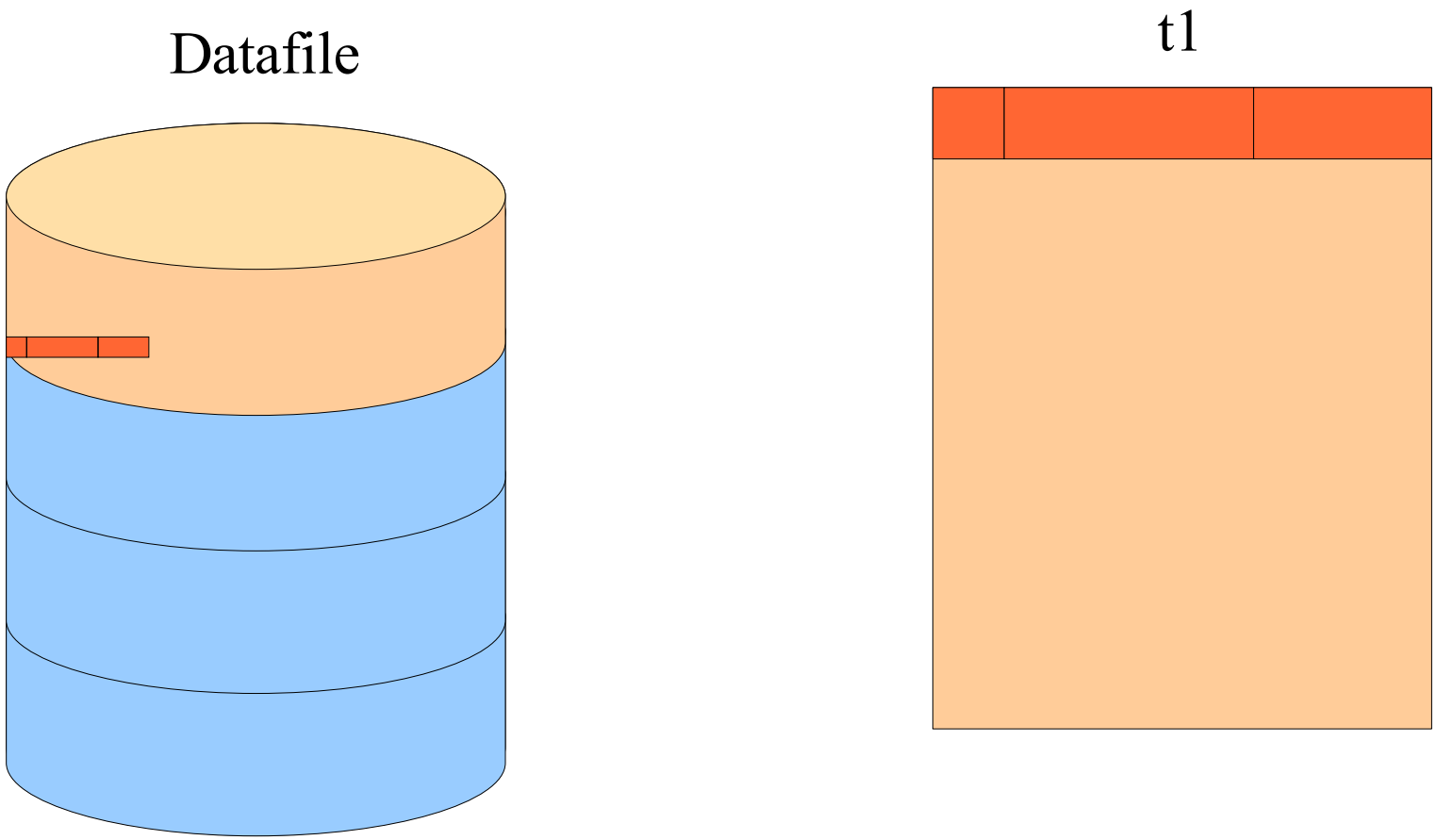


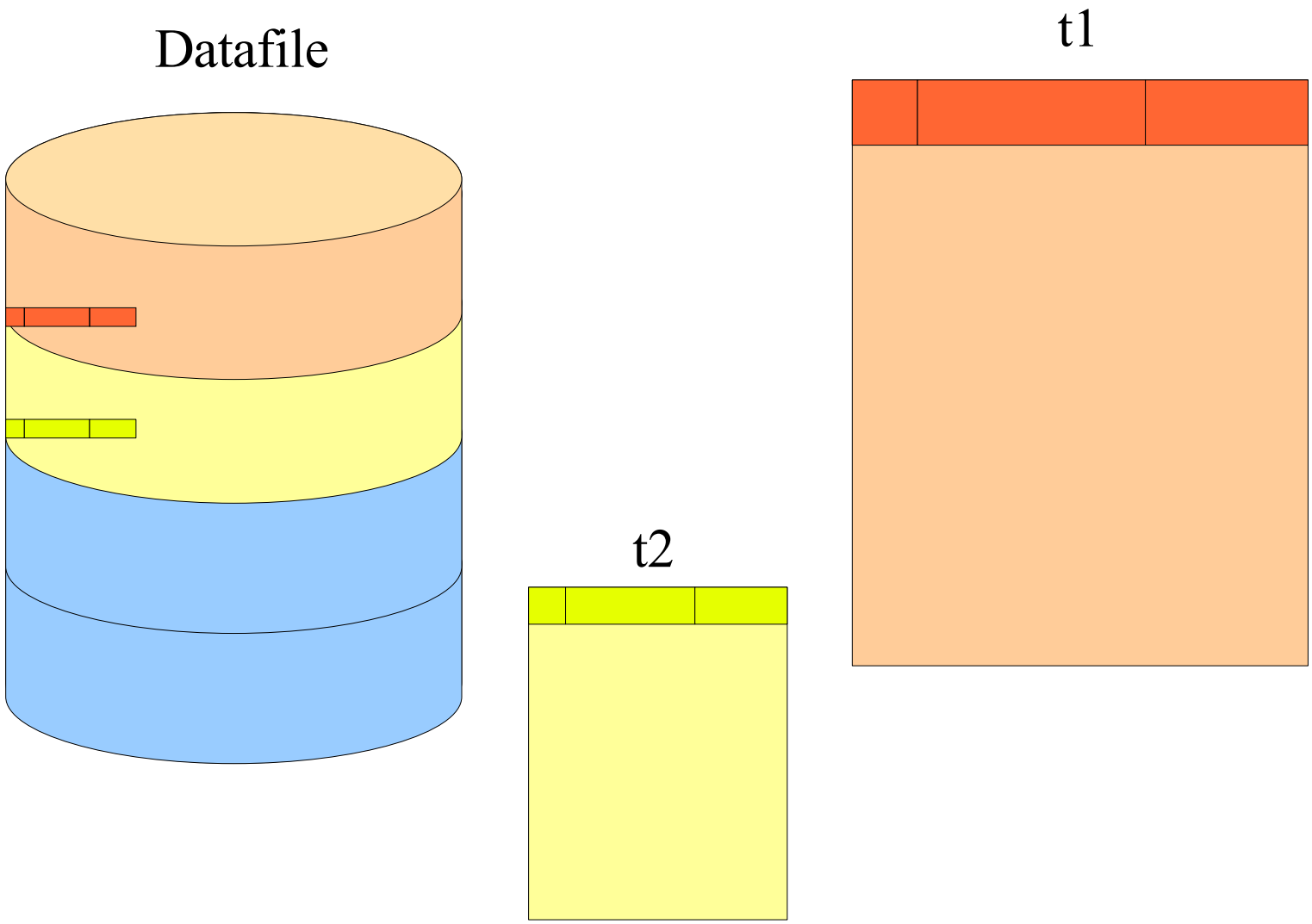
Datafile

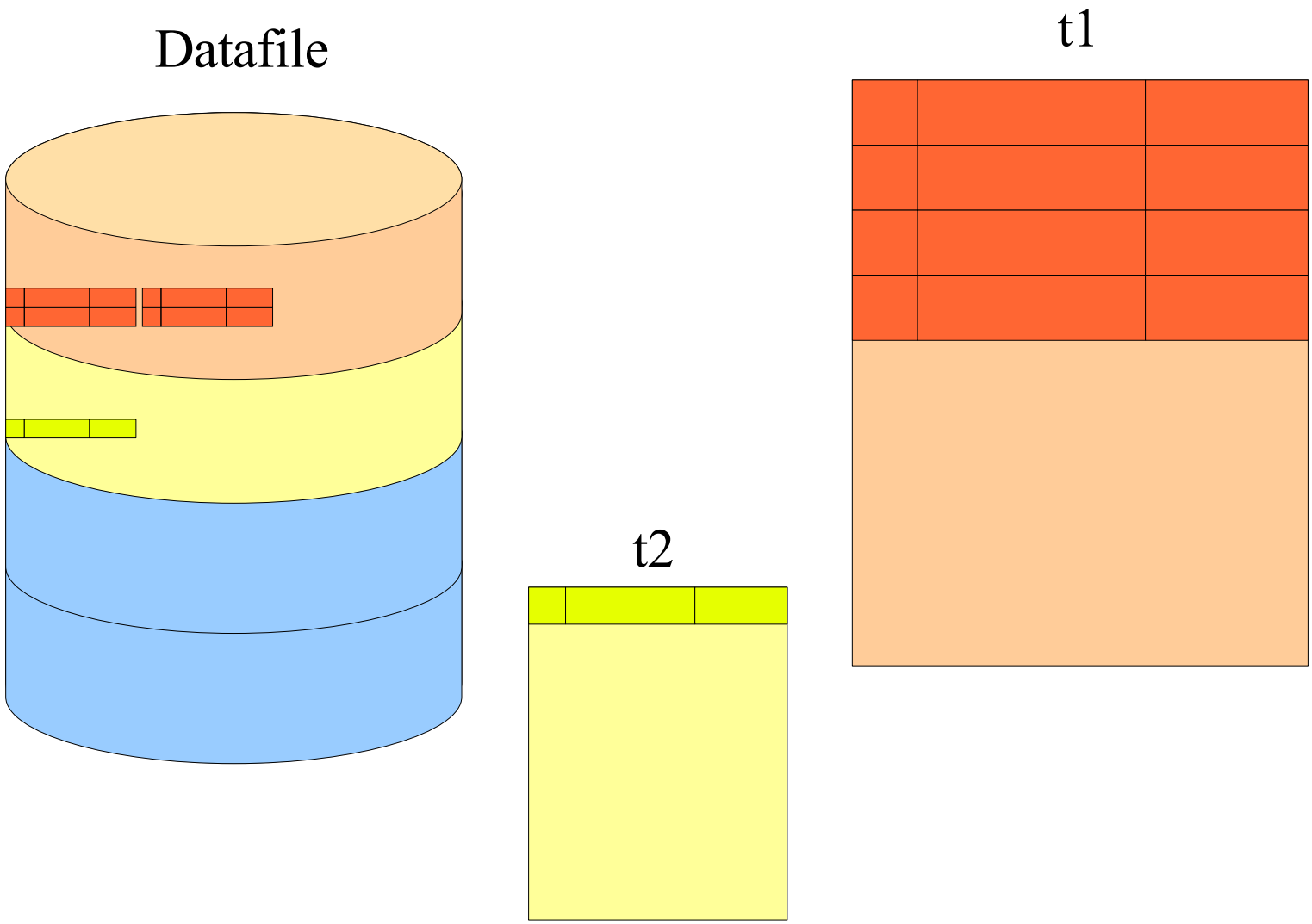


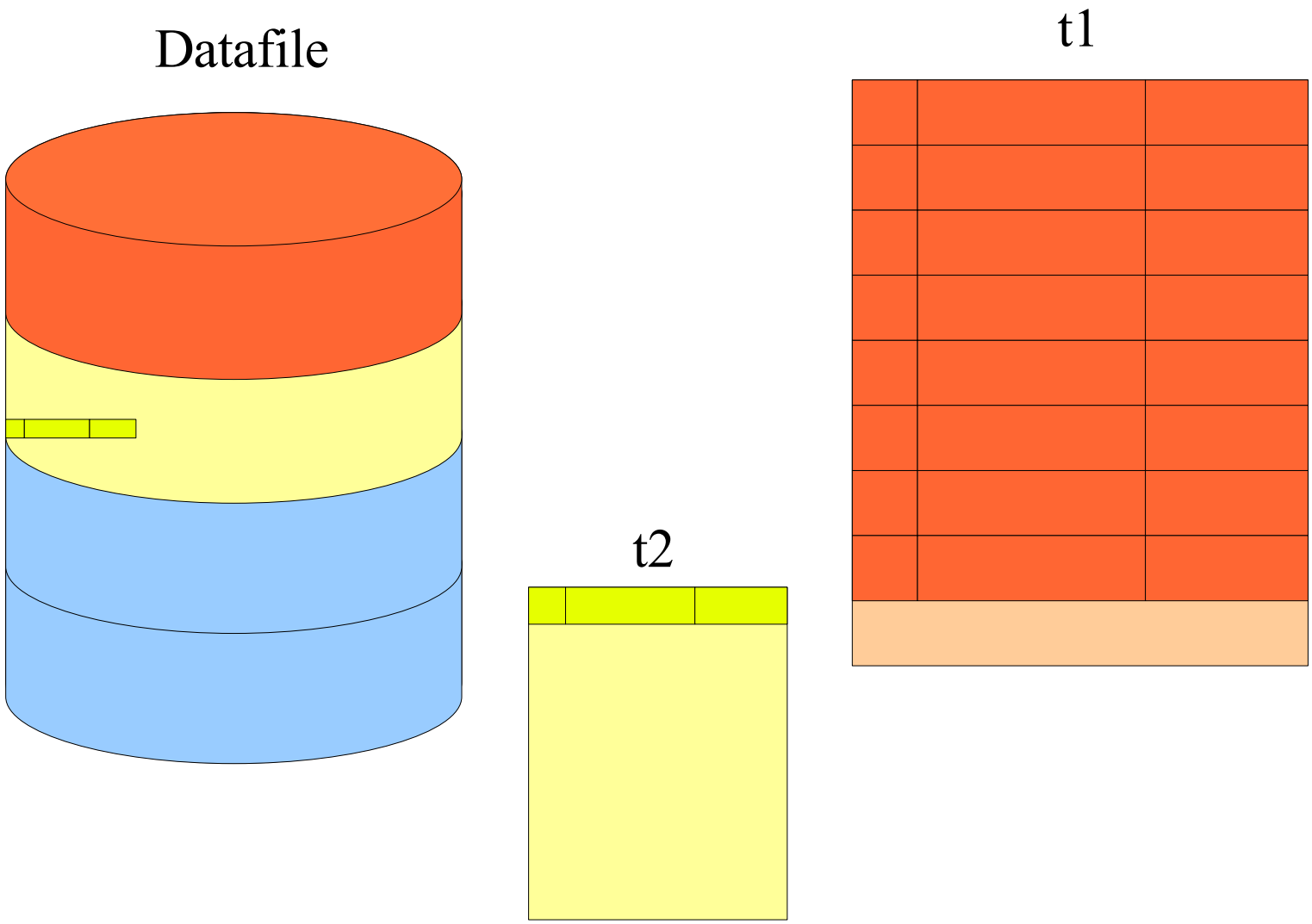
t1

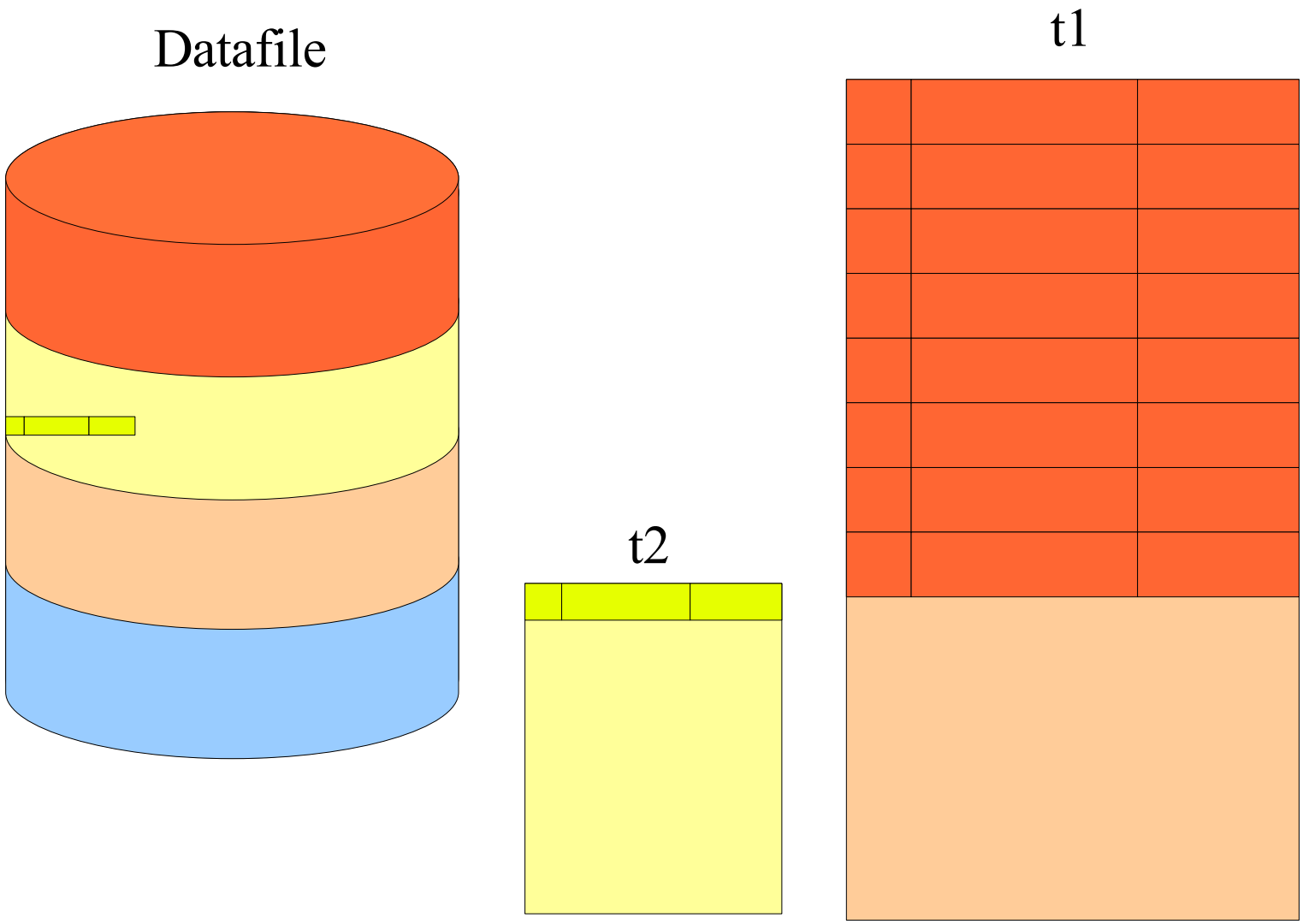


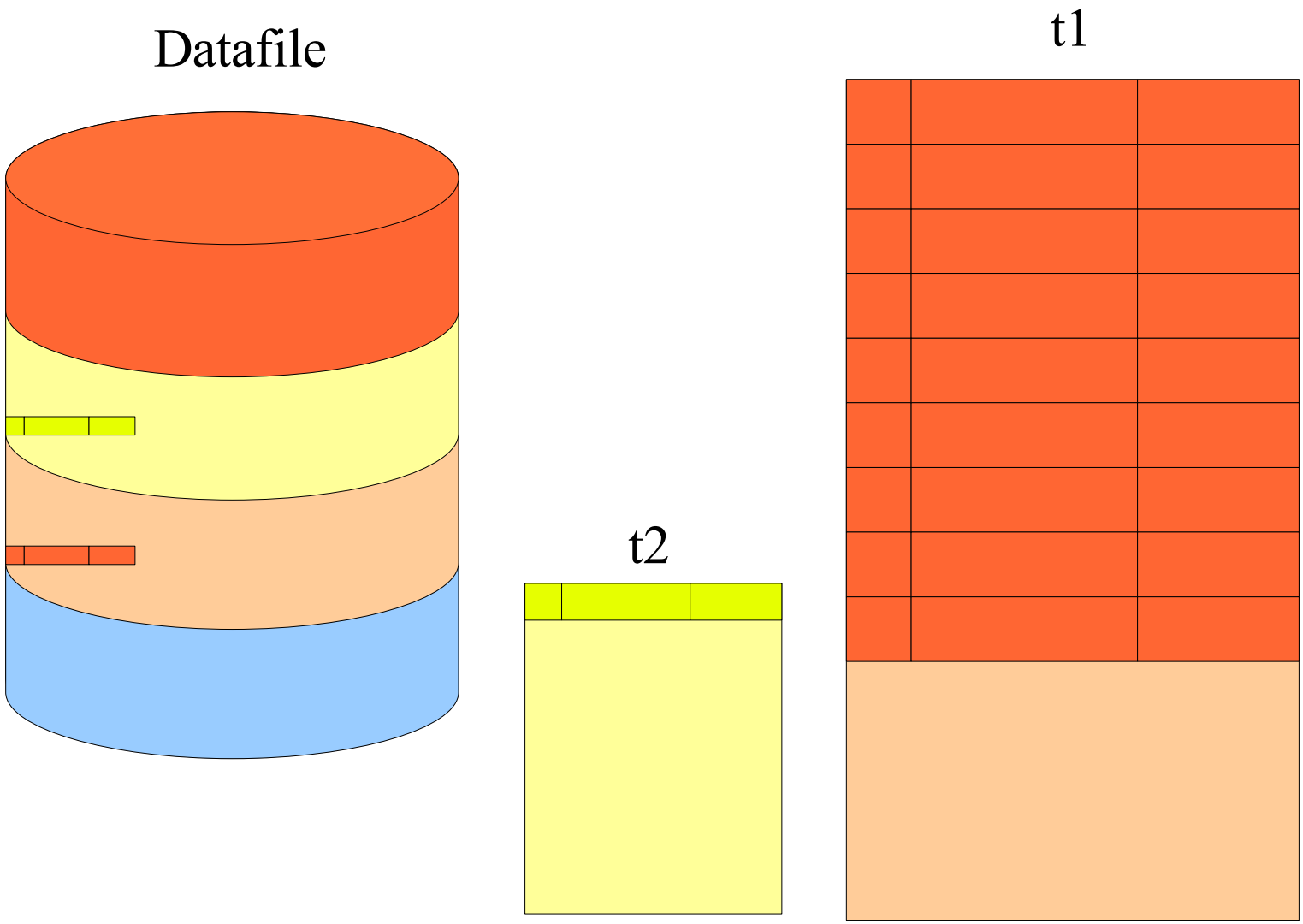


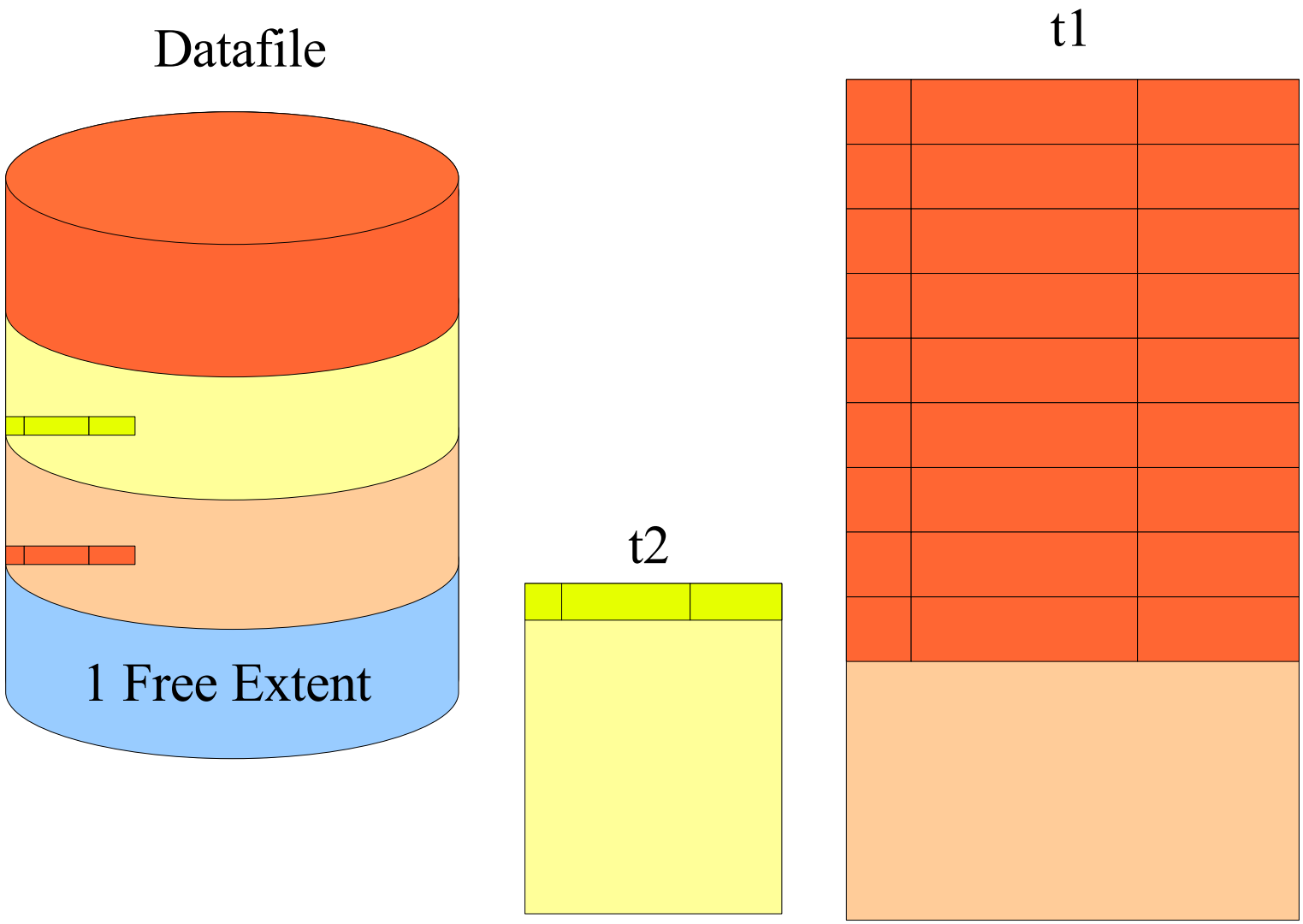












I_S.FILES for Data files

- Datafile
  - what ts it belongs to
- Extent Size (bytes)
- Number of extents in file
- Number of free extents
- So, Free extents multiplied by extent size = free bytes that can be allocated to tables

A useful VIEW

```
CREATE VIEW isf AS
    SELECT FILE_NAME,
    (TOTAL_EXTENTS * EXTENT_SIZE) AS 'Total',
    (FREE_EXTENTS * EXTENT_SIZE) AS 'Free',
    (     ((FREE_EXTENTS * EXTENT_SIZE)*100)
    /     (TOTAL_EXTENTS * EXTENT_SIZE))
          AS '% Free'
    FROM
    INFORMATION_SCHEMA.FILES
WHERE ENGINE="ndbcluster"
and FILE_TYPE = 'DATAFILE';
```

A useful VIEW

```
mysql> select * from isf;
Empty set (0.01 sec)
```

A useful VIEW

```
mysql> select * from isf;
Empty set (0.01 sec)

mysql> CREATE TABLESPACE ts1 ADD DATAFILE
'datafile.dat' USE LOGFILE GROUP lg1
INITIAL_SIZE 12M ENGINE NDB;
 Query OK, 0 rows affected (2.55 sec)
```

A useful VIEW

```
mysql> select * from isf;
+--------------+--------------------+----------------------+--------------+
| FILE_NAME    | Total Extent Size | Total Free In Bytes | % Free Space |
+--------------+--------------------+----------------------+--------------+
| datafile.dat |          12582912 |             12582912 |     100.0000 |
| datafile.dat |          12582912 |             12582912 |     100.0000 |
+--------------+--------------------+----------------------+--------------+
2 rows in set (0.00 sec)

mysql>
```

A useful VIEW

```
mysql> select * from isf;
+--------------+--------------------+----------------------+--------------+
| FILE_NAME    | Total Extent Size | Total Free In Bytes | % Free Space |
+--------------+--------------------+----------------------+--------------+
| datafile.dat |          12582912 |             12582912 |     100.0000 |
| datafile.dat |          12582912 |             12582912 |     100.0000 |
+--------------+--------------------+----------------------+--------------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE t1 (pk1 int not null
primary key, b int not null, c int not null)
tablespace ts1 storage disk engine ndb;
Query OK, 0 rows affected (0.68 sec)

mysql> insert into t1 () values ();
Query OK, 1 row affected, 3 warnings (0.00
sec)
```

A useful VIEW

```
mysql> select * from isf;
+--------------+--------------------+----------------------+--------------+
| FILE_NAME    | Total Extent Size | Total Free In Bytes | % Free Space |
+--------------+--------------------+----------------------+--------------+
| datafile.dat |           12582912 |             11534336 |      91.6667 |
| datafile.dat |           12582912 |             11534336 |      91.6667 |
+--------------+--------------------+----------------------+--------------+
2 rows in set (0.01 sec)

mysql>
```

I_S.FILES for UNDO files

- Free log space
- If running out, maybe need to add more

The future of I_S.FILES

- Perhaps reporting of (approximate or exact) free space inside extents allocated to tables
  - This means there'll be the generic row for the data file as well as rows for each table
  - Potentially very long I_S.FILES table
- Other storage engines
  - Developers need to add fill_files_table to their handlerton
  - Potentially some code cleanups to make it easier to extend the table

Some internal details....

Buffer Manager

- Main memory database has no buffer manager
  - no need, everything is in memory
  - simplifies things a lot :)
- Good Buffer Manager = Good Performance
- There will be tuning and enhancements here in future releases

NO-STEAL

- We use no-steal algorithm
  - no uncommitted data written to disk
  - Saves some disk writes
  - Limited transaction size (configurable – same as always)

Optimized NR

- Traditional NR
  - copy everything over the wire
  - PRO: easy to implement correctly
  - PRO: not too bad for a few gigs of data
  - CON: **very *very*** bad for disk data
    - think 2TB going over the wire... **ouch**!
- Details on optimized node recovery for NDB
  - in s1108-ronstrom.pdf
  - recovery from checkpoint, so don't have to copy everything

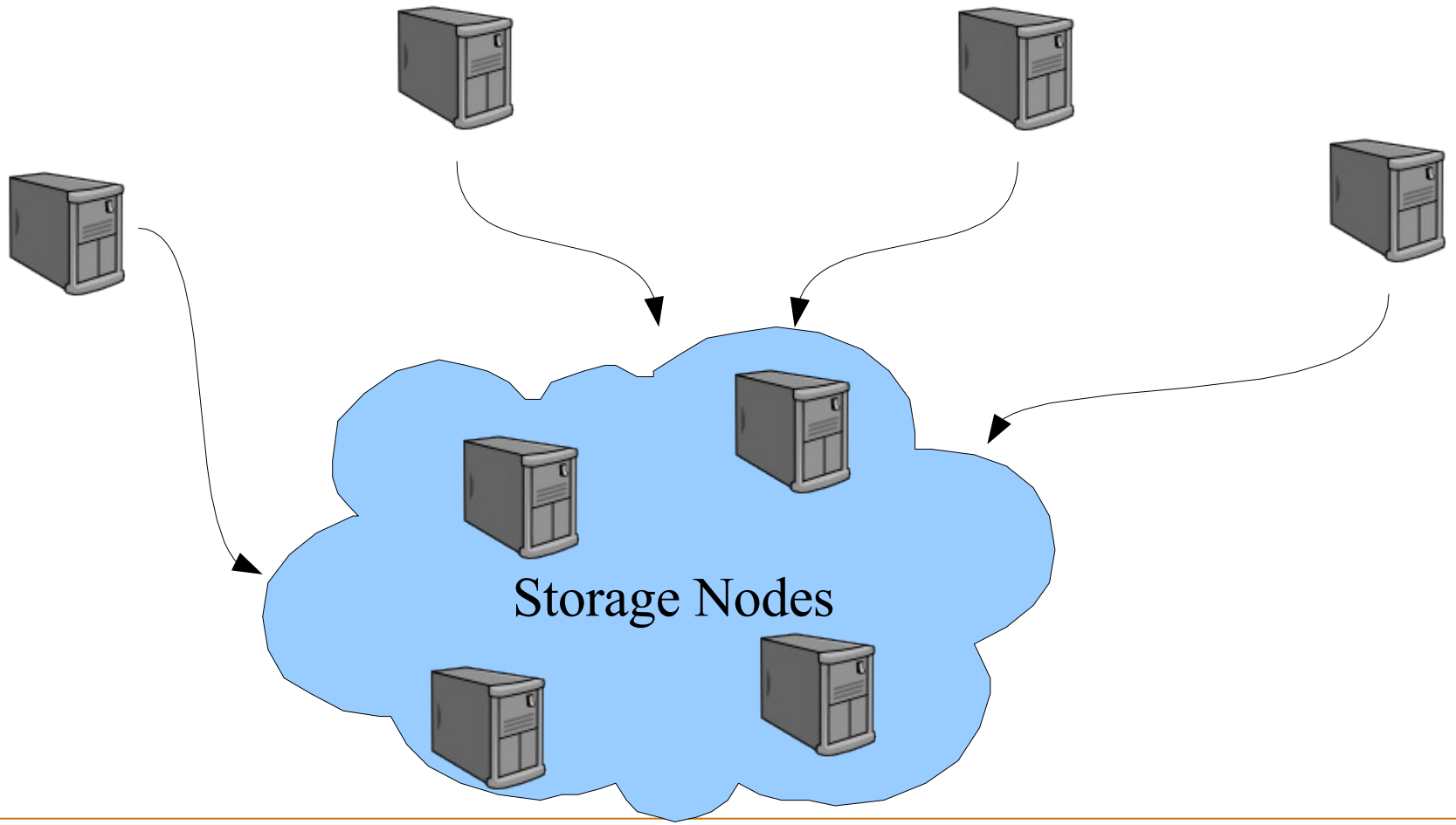
Disk Data

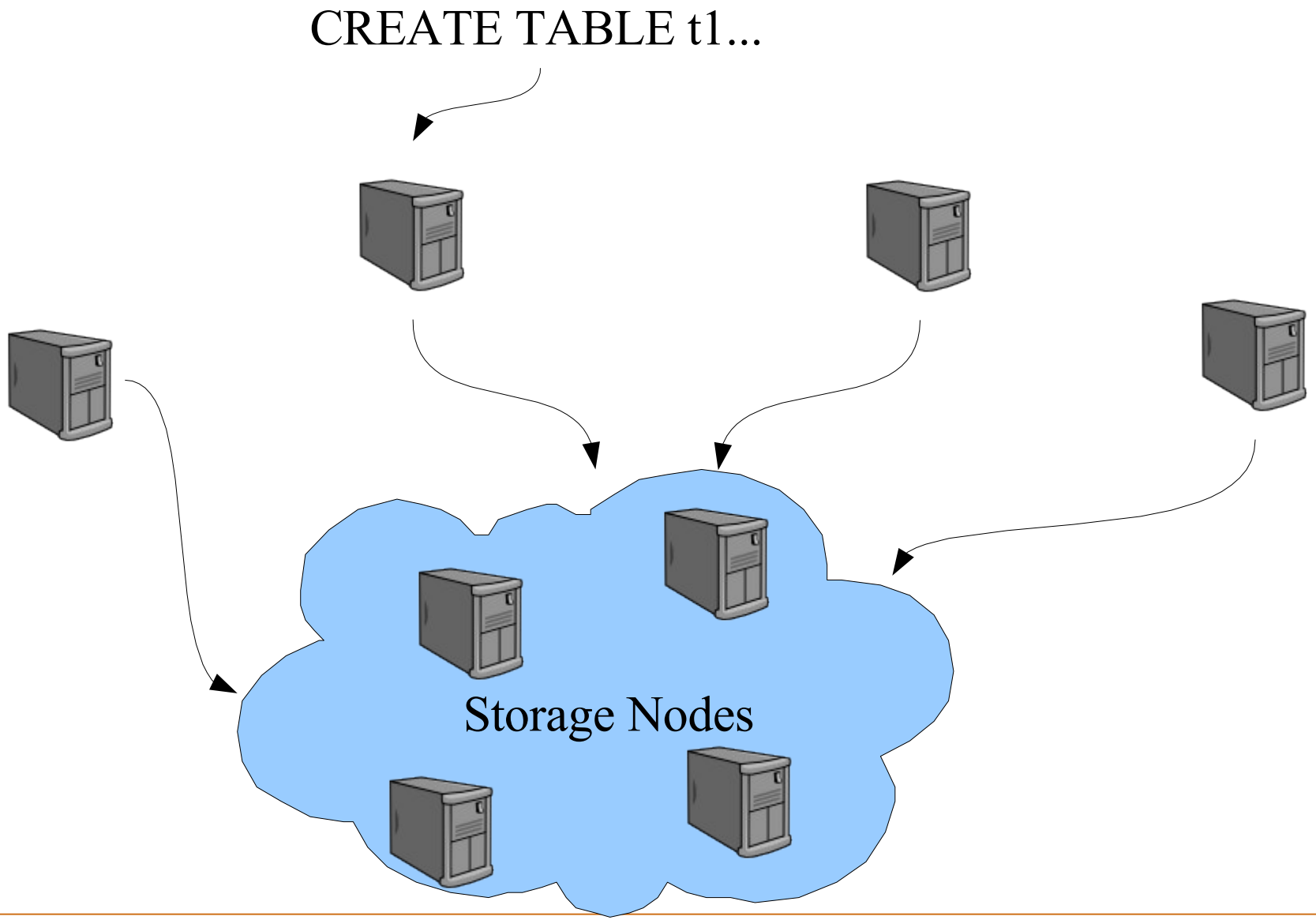
More data in Cluster

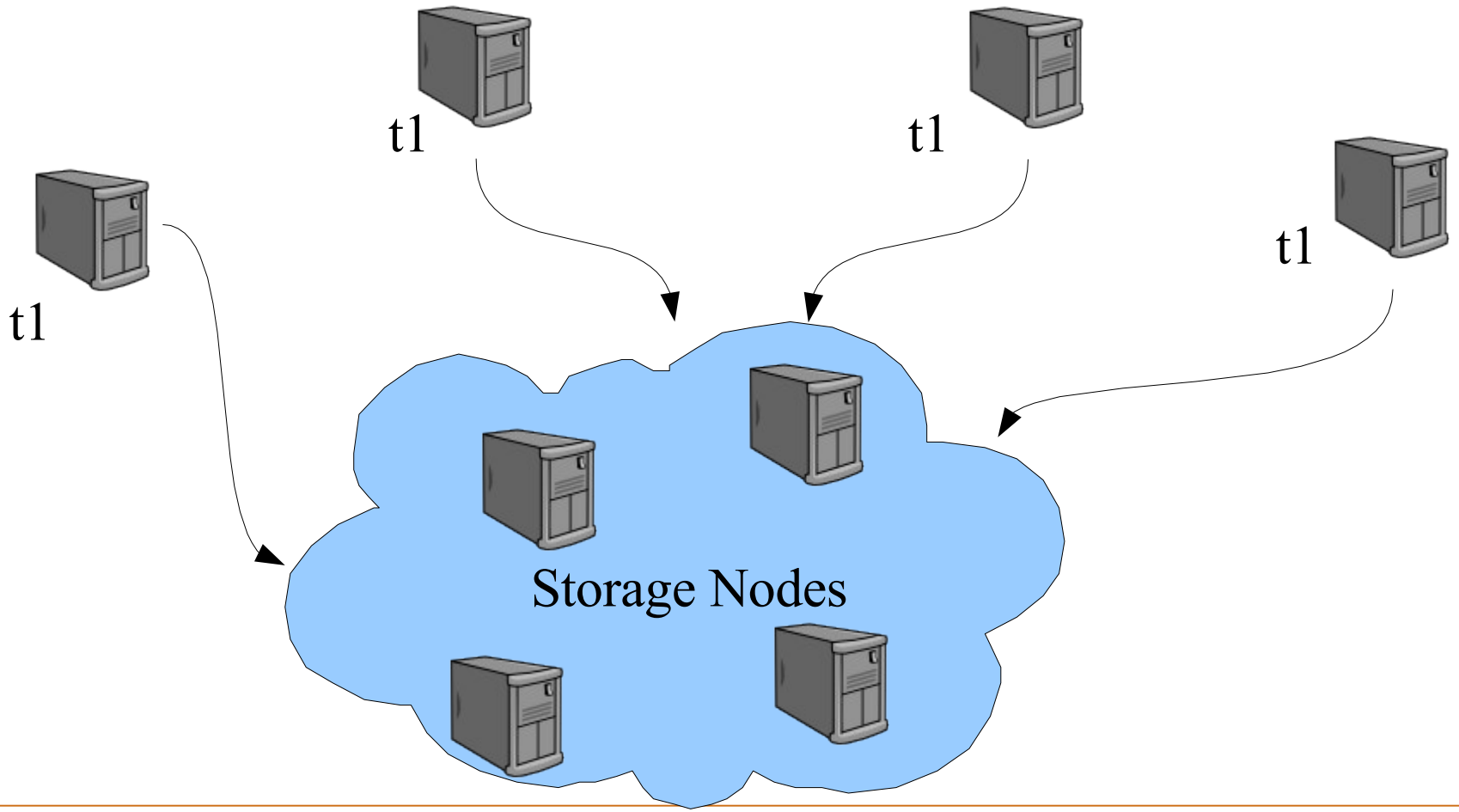
While keeping Main Memory performance when needed

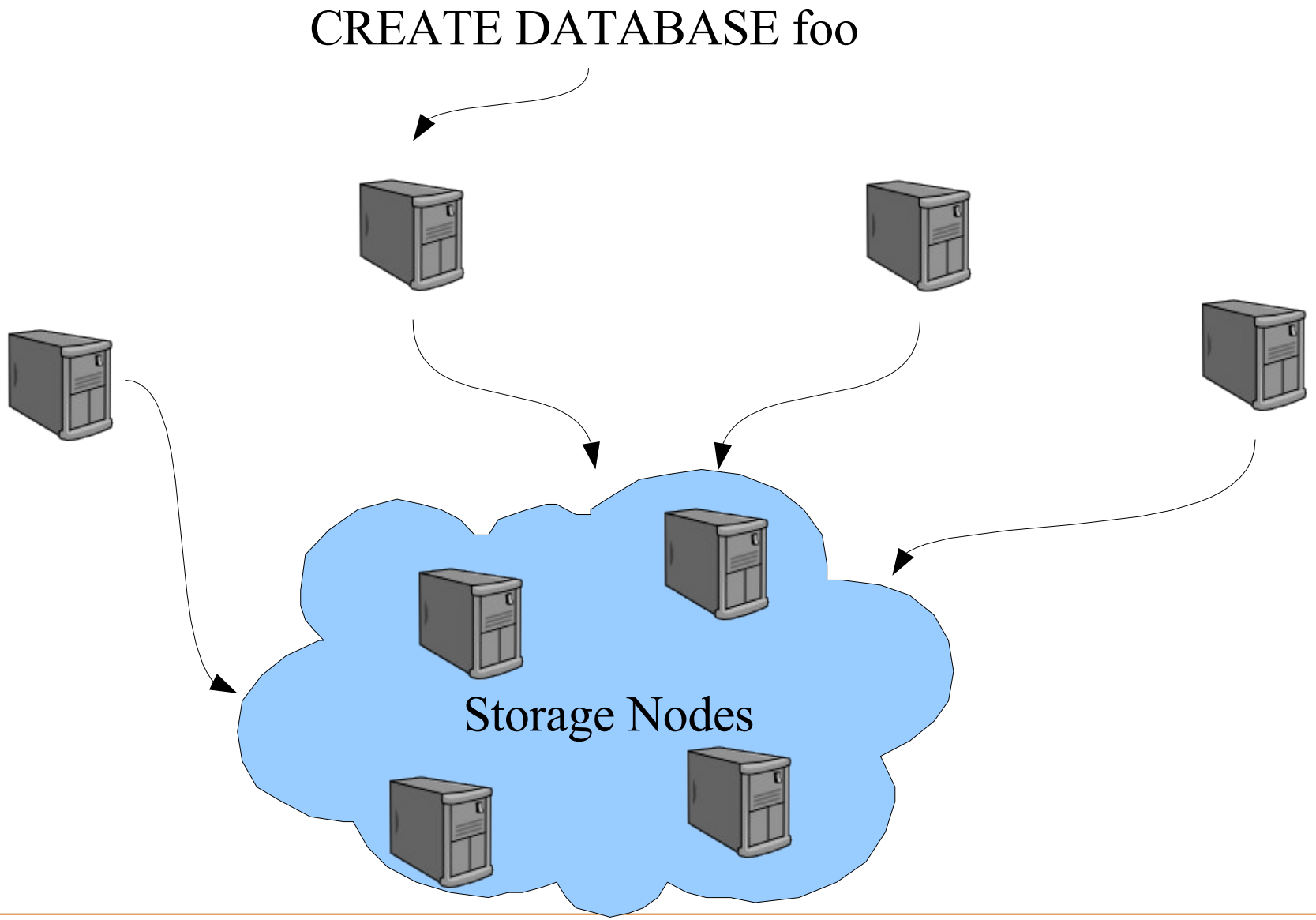
Questions?

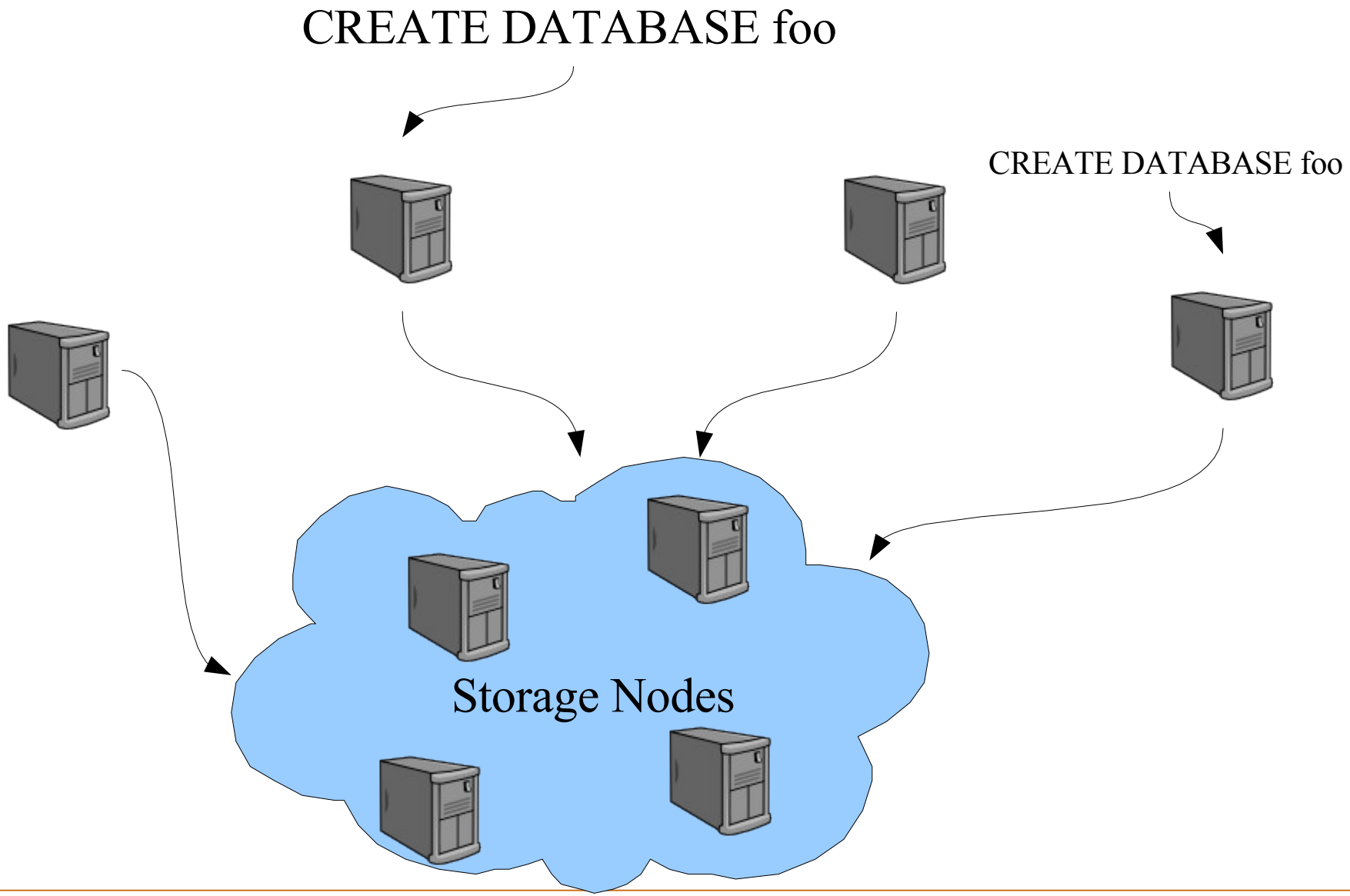
In 4.1 and 5.0...

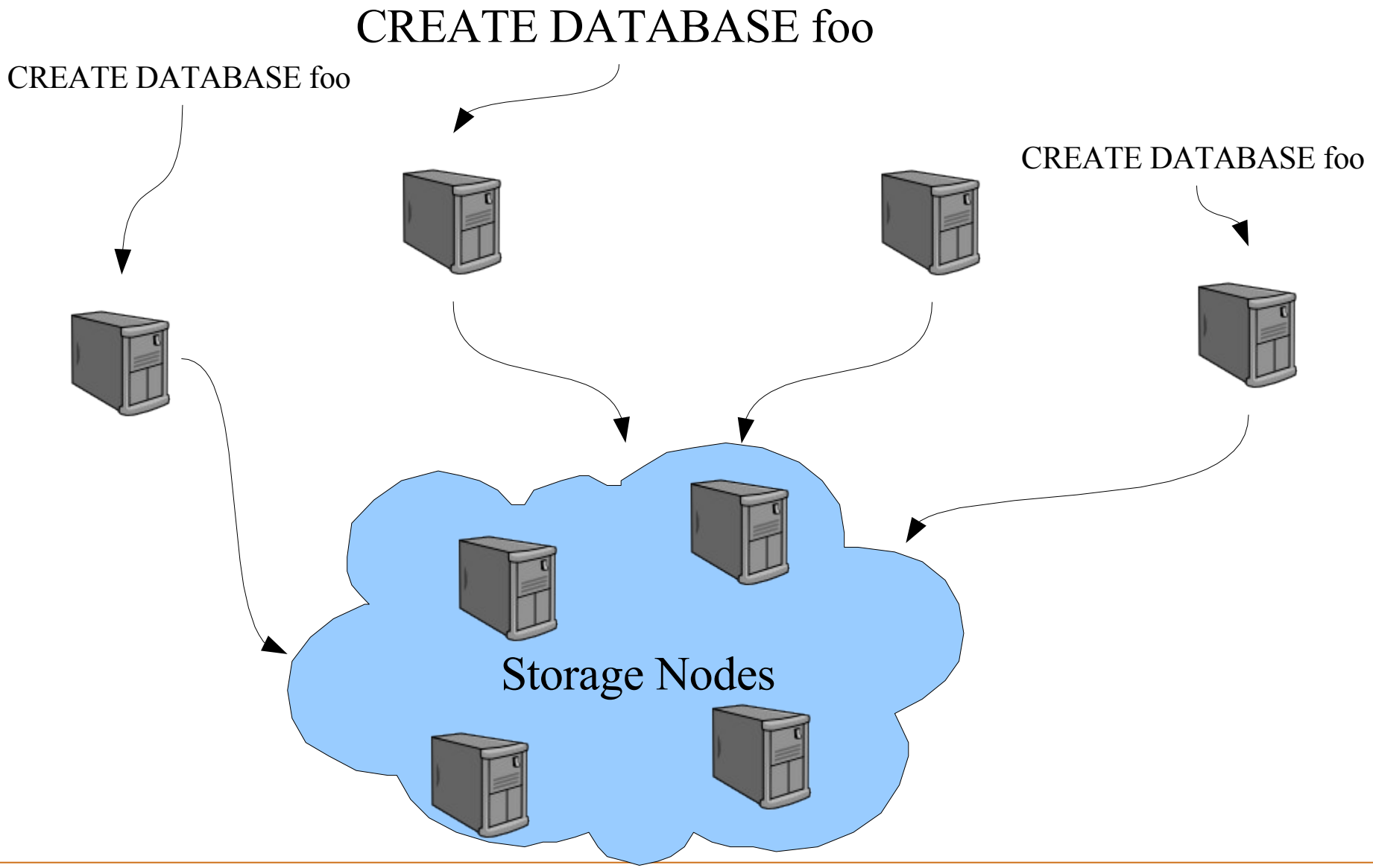


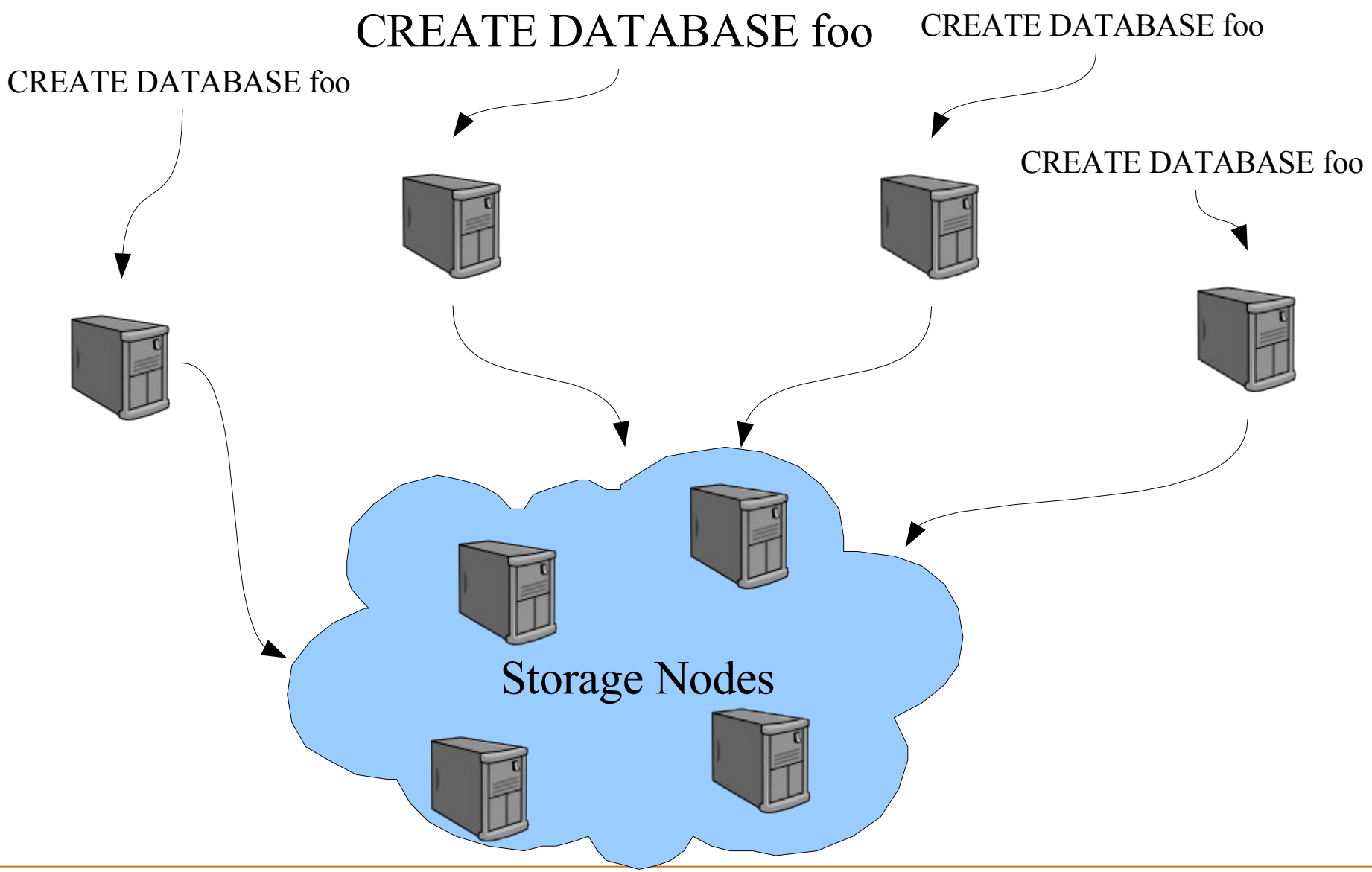






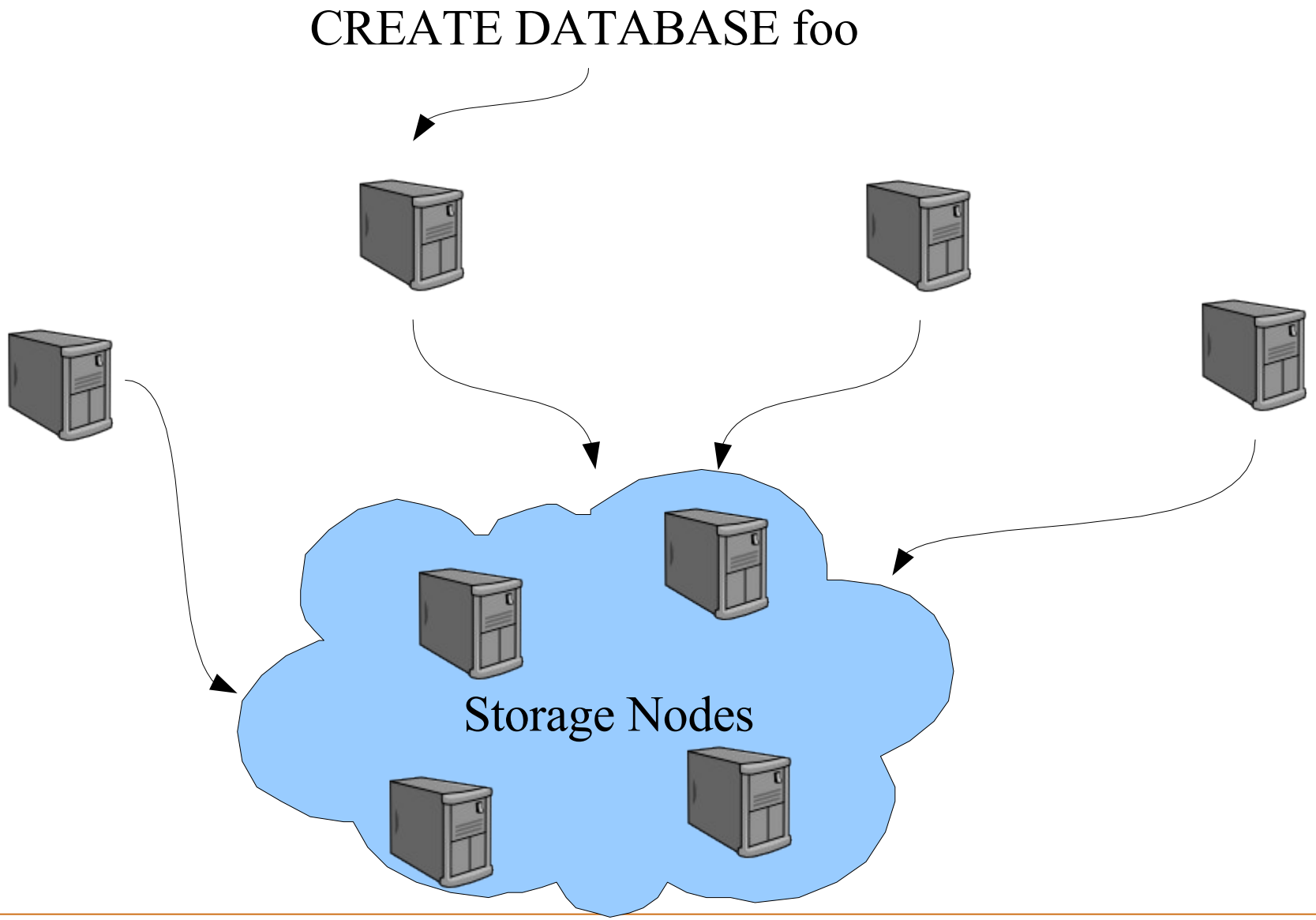


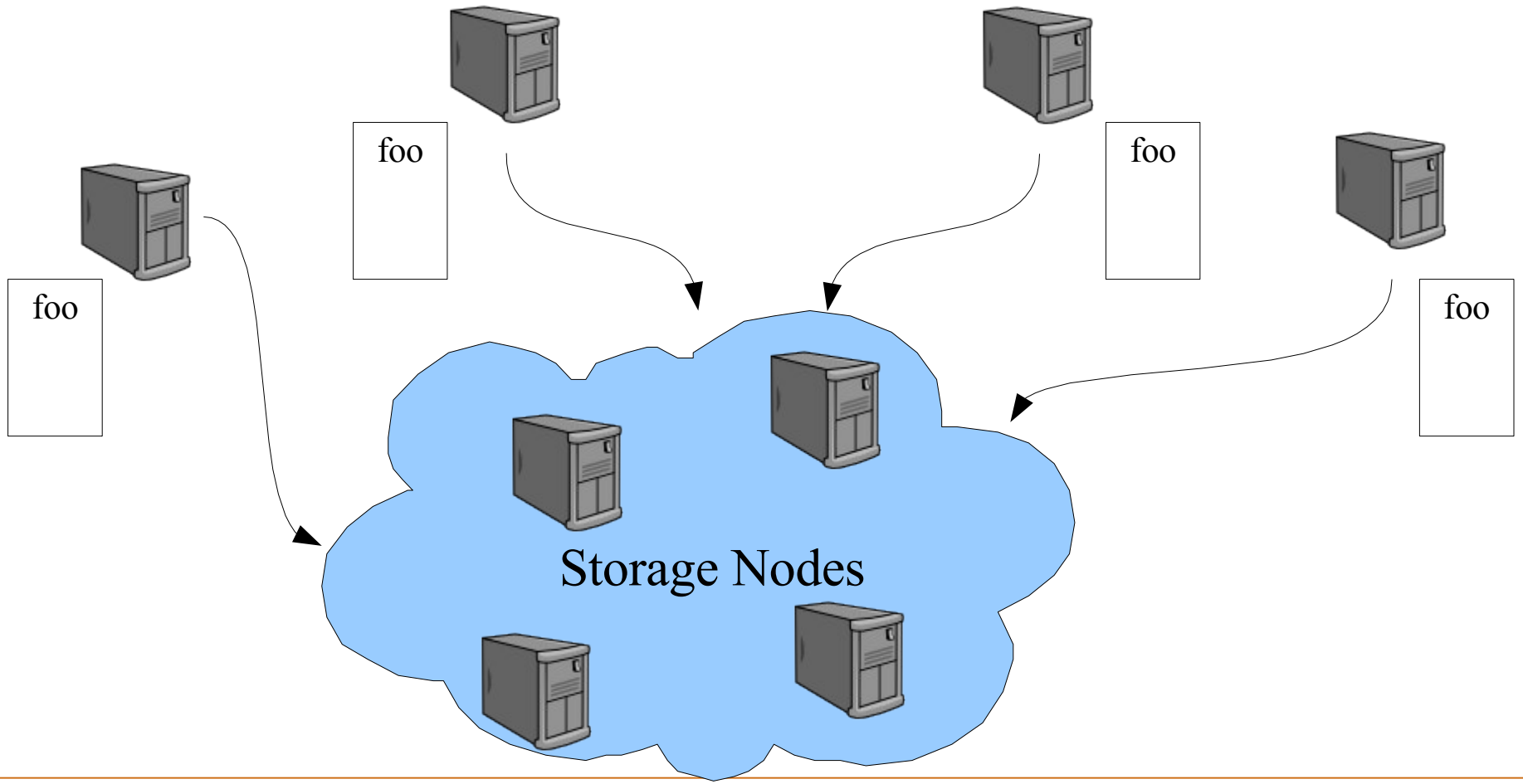




But now...

in 5.1





in 4.1 and 5.0

Table Auto Discovery

in 5.1

Database and Table auto discovery

Disk Data

in 5.1

Database and Table auto discovery

Disk Data

Integration with Replication

in 5.1

Database and Table auto discovery

Disk Data

Integration with Replication

in 5.1

Database and Table auto discovery

User Defined Partitioning

Disk Data

Integration with Replication

Variable Sized Attributes

in 5.1

Database and Table auto discovery

User Defined Partitioning

Disk Data

Integration with Replication

Variable Sized Attributes

in 5.1

Database and Table auto discovery

On line Add/Drop Index

User Defined Partitioning

Disk Data

Better error reporting

Integration with Replication

Variable Sized Attributes

in 5.1

Database and Table auto discovery

On line Add/Drop Index

User Defined Partitioning

Disk Data

Better error reporting

Integration with Replication

Variable Sized Attributes

in 5.1

Database and Table auto discovery

On line Add/Drop Index

User Defined Partitioning

More internal QA

Disk Data

Better error reporting

Integration with Replication

Variable Sized Attributes

in 5.1

Database and Table auto discovery

Improvements in Cluster management

On line Add/Drop Index

User Defined Partitioning

More internal QA

Disk Data

Better error reporting

Improving Documentation

Integration with Replication

Variable Sized Attributes

in 5.1

Database and Table auto discovery

Improvements in Cluster management

On line Add/Drop Index

User Defined Partitioning

More internal QA

Disk Data

Better error reporting

Improving Documentation

Integration with Replication

Variable Sized Attributes

Improving configuration

in 5.1

Database and Table auto discovery

Improvements in Cluster management

On line Add/Drop Index

User Defined Partitioning

More internal QA

Find out More!

- MySQL Online Documentation
  - http://dev.mysql.com/doc/refman/5.0/en/index.html
- Cluster Forum
  - http://forums.mysql.com/list.php?25
- Cluster Mailing List
  - http://lists.mysql.com/cluster
- Contact Sales
  - sales@mysql.com
- Contact Me
  - Monty Taylor
  - mtaylor@mysql.com

Quick intro to MySQL

- SQL RDBMS
  - Aiming for full standards compliance
- Reputation for speed, reliability and ease of use
- MySQL 5.0 is GA, 5.1 is Beta
  - many Cluster improvements over 4.1
- 5.0 Supports (among other things)
  - SQL 2003 Stored procedures
  - Triggers
  - Updateable Views
  - Cursors
  - Precision math
  - data dictionary (INFORMATION_SCHEMA database)
  - distributed transactions (XA)

Cluster across the versions...

- Cluster in 4.1
  - Focused on High Availability
- Cluster in 5.0
  - Make it faster!
- Cluster in 5.1
  - Added new features
- Online upgrade between minor versions (e.g. 5.0.18 to 5.0.19)

Storage Engines

- Unique feature of MySQL
  - Theory is: no one way of storing a table is ideal for all situations
  - So we let you choose!
  - on a per table basis
  - it's a VFS for your database tables!
- Several to choose from
- Can create your own
- Well known existing ones:
  - MyISAM, InnoDB, HEAP, CSV, BDB, Archive, federated
- Cluster is a storage engine for MySQL
  - ENGINE=NDB or ENGINE=ndbcluster

What is MySQL Cluster?

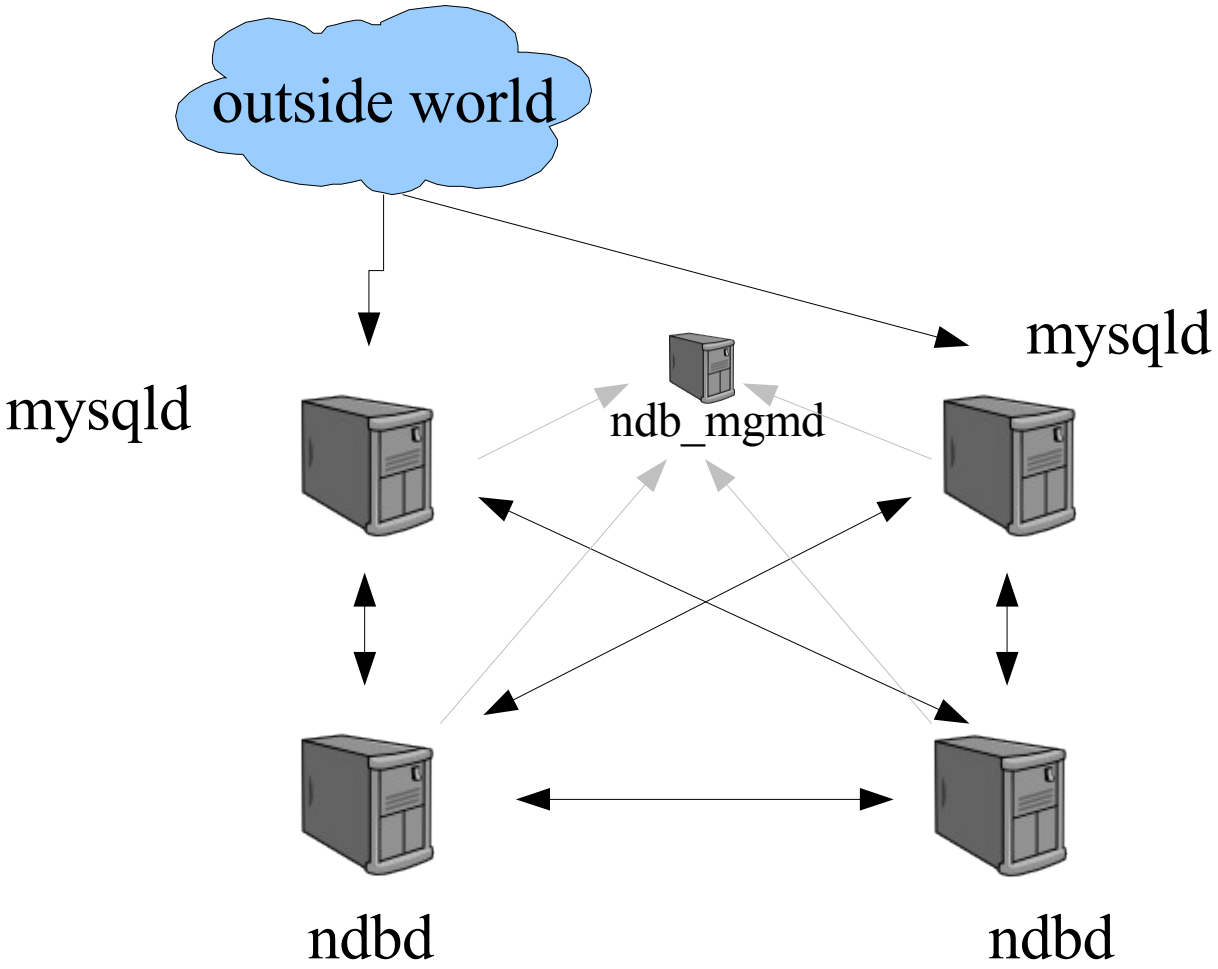
- An in-memory clustered storage engine for MySQL
  - Data and Indexes kept in main memory
  - Check-pointed to disk
  - Disk Data 5.1
- Highly Available
  - Designed for five 9s availability (99.999% uptime)
  - sub-second fail over
  - Hot (Online) Backup (no locks taken during backup)
  - Amount of redundancy is configurable: NoOfReplicas
- Shared-nothing architecture
  - A cluster of commodity hardware
  - Choice for interconnects (TCP and SCI are the main ones)
  - No need for expensive SANs

$$\text{Per Node} = \frac{\text{Size of Database} \times \text{No Of Replicas} \times 1.1}{\text{No Of Data Nodes}}$$

Management Server

config.ini

```
[ndbd default]
NoOfReplicas= 2
DataMemory= 400M
IndexMemory= 32M
DataDir= /usr/local/mysql/cluster

[ndbd]
HostName= 192.168.0.40

[ndbd]
HostName= 192.168.0.41

[ndb_mgmd]
DataDir= /usr/local/mysql/cluster
HostName= 192.168.0.42

[mysqld]
[mysqld]
[mysqld]
```



Nodes

- Storage nodes (ndbd)
  - Grouped into nodegroups
  - Each nodegroup holds a partition of the database
  - Number of nodes in a nodegroup configurable (NoOfReplicas)
  - Up to 48 in one cluster
  - One acts as a transaction coordinator
- Management Node
  - Distributes configuration to other nodes
- MySQL nodes
  - Sometimes referred to as API nodes
  - MySQL servers
  - Access just like you would any other mysql server

Physical Requirements

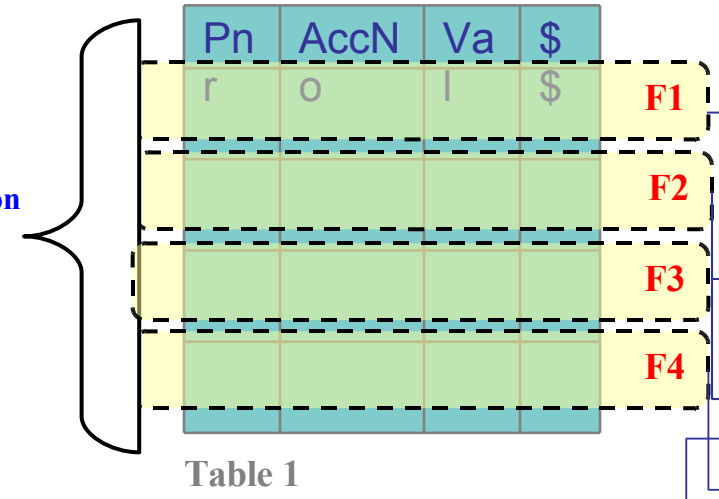
- A node is a process, not a computer
- Need at least three machines in a cluster
  - This avoids the split brain problem
- Management server doesn't need a powerful machine
- Data nodes
  - generally have a lot of memory
  - Disk IO requirements can be calculated from configuration parameters
  - not CPU bound
  - ndbd is single threaded
- SQL nodes need the most CPU
  - have more of these than data nodes
  - mysqld is multithreaded

CREATE TABLE

- ```
  CREATE TABLE fbhighscores (
    level int,
    piclevel int,
    time float,
    name varchar(50)
  ) engine=ndb;
  ```
- ```
  CREATE TABLE fblevelset (
    name varchar(20),
    `num` int auto_increment,
    `level` text,
    PRIMARY KEY  (name,num)
  ) ENGINE=ndbcluster;
  ```

Data Distribution on 4 Nodes

- Horizontal fragmentation of Table 1 (4 fragments)
- Fragments distributed on nodes



2 copies of data

Fx – primary replica

Fx – secondary replica