MySQL Cluster Workshop

Ted Wennmark,
MySQL Solution Architect, MySQL, 2020

Program Agenda

1. Introduction
2. Getting started: configuration/install/start/stop
3. Administration: backup/upgrade/logs/conf/sizing
4. Monitoring, surveillance and problem solving
5. Best practices, architectures and case studies
6. NDB 7.6 and what's new in NDB 8.0
7. Geo-replication

MySQL High Availability Solutions

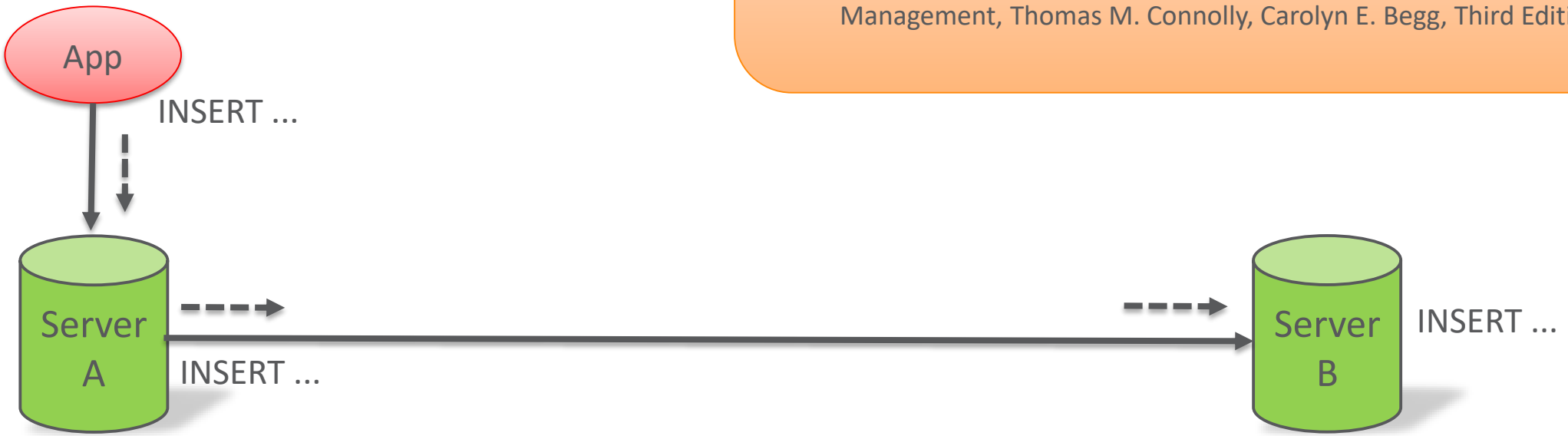
- MySQL NDB Cluster
  - NDB storage engine
  - Memory database
  - Automatic sharding of data
  - SQL Access via MySQL with cross shard join support
  - Native access via several API's
  - Read/write consistency
  - Read/write scalability
  - ACID and transactions
- MySQL InnoDB Cluster
  - Easy HA built into MySQL 5.7+ for InnoDB
  - MySQL Group Replication, Shell and Router
  - Write consistency
  - Read Scalability
  - Native CRUD API in MySQL 8
  - Synchrounous (Paxos)
- MySQL Replication
  - Core part of MySQL, used by almost everyone.
  - Can be used by any storage engine.
  - Asynchronous and semi-sync option.
  - Scale out reads.
  - ReplicaSet from 8.0.19 with integratic
    Router

Database Replication

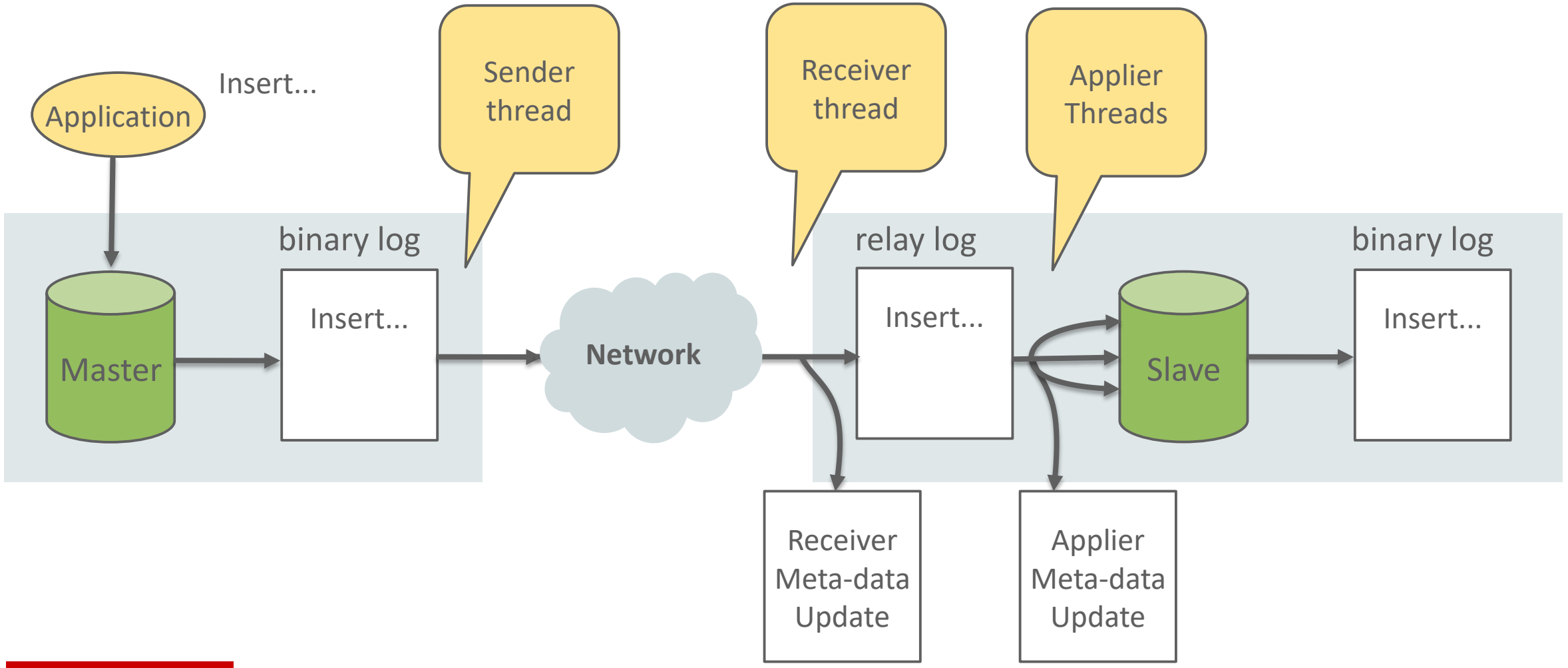
Replication

"The process of generating and reproducing multiple copies of data at one or more sites.",

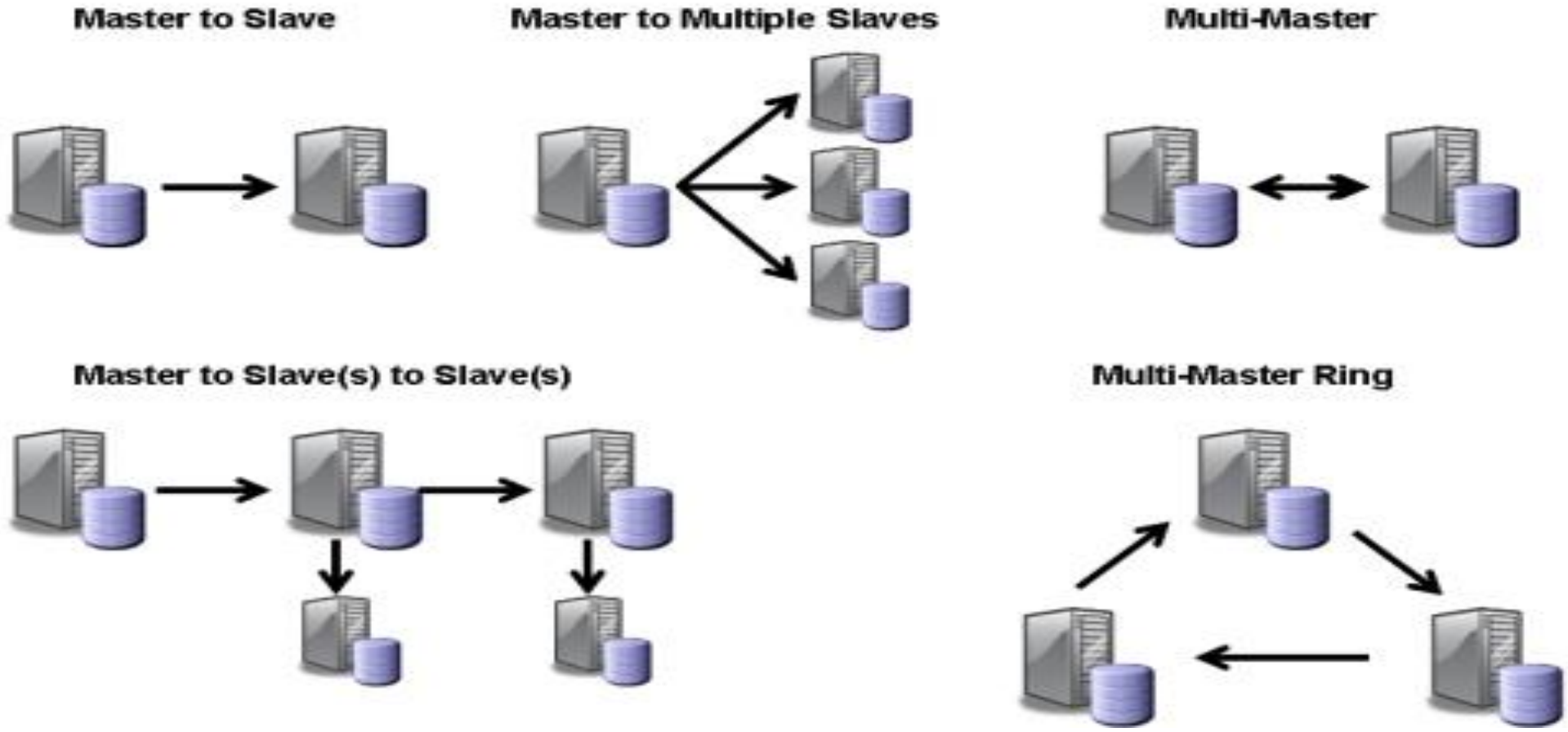
Database Systems: A Practical Approach to Design, Implementation, and Management, Thomas M. Connolly, Carolyn E. Begg, Third Edition, 2002.



Background: Replication Components



Background: Repliction Architectures



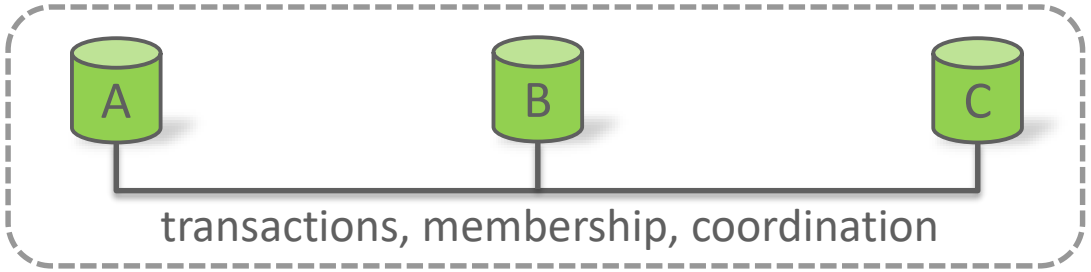
MySQL Database Replication: Consistency

Since 3.23

asynchronous (native)

Since 5.5

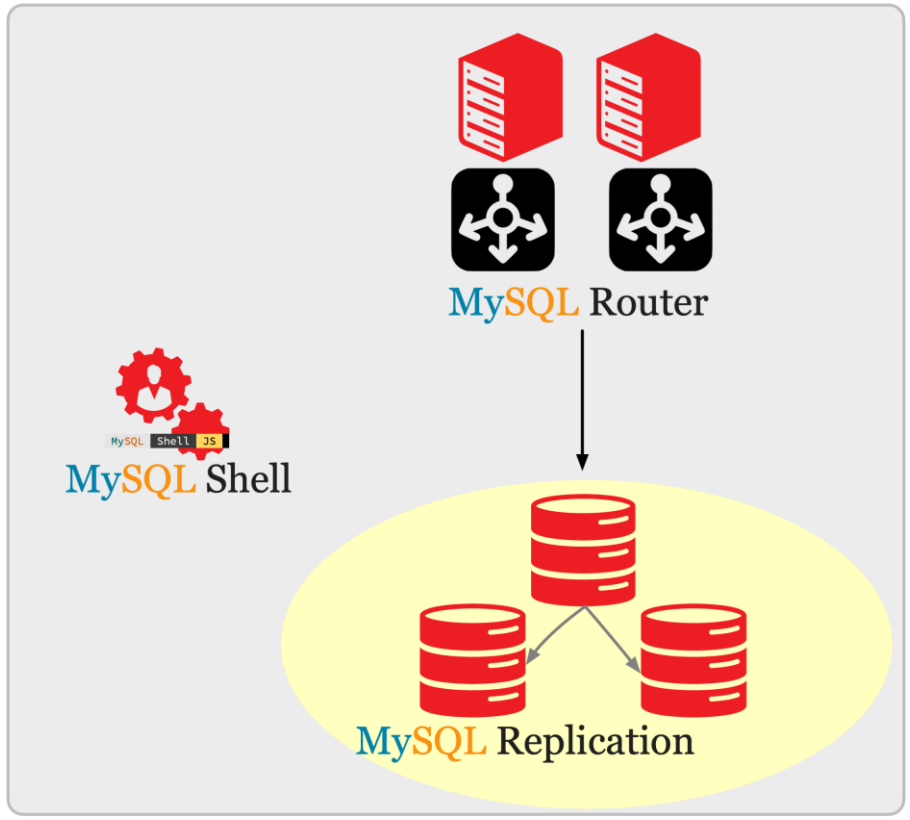
semi-synchronous (plugin)



Since 5.7.17

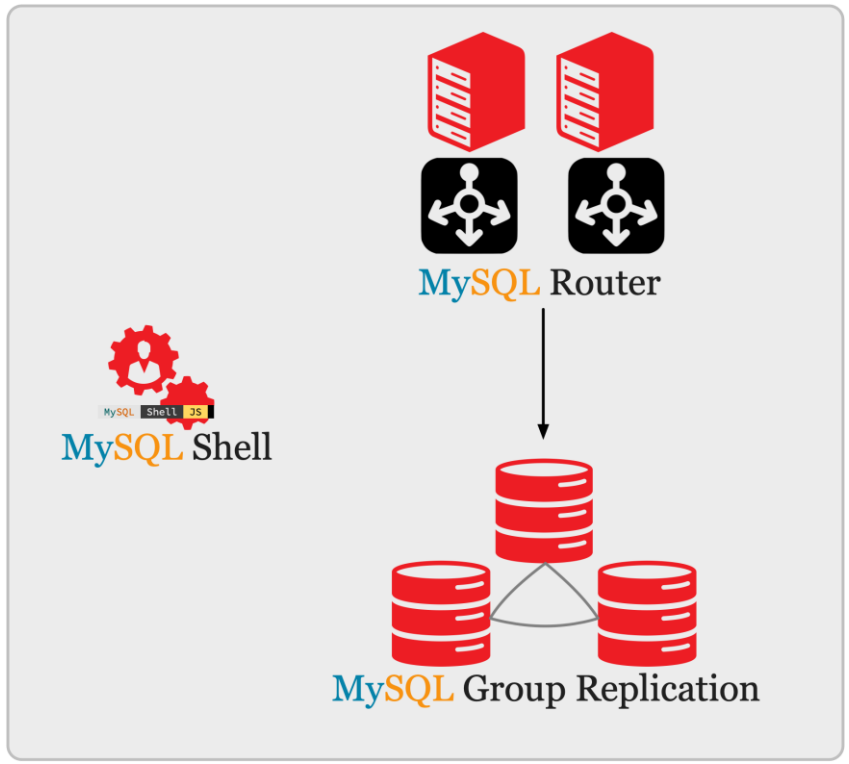
group replication (plugin)

MySQL InnoDB ReplicaSets (8.0.19)



- **Asynchronous** Replication Architecture
  - Manual Switchover & Failover
  - Asynchronous Read Scaleout
  - Simple Replication architecture
  - Automatic Node provisioning (CLONE)
- **MySQL Shell** Configuring, Adding, Removing members
- **MySQL Router** to route application traffic

MySQL InnoDB Cluster



- **Group Replication:**
  - Read/write Consistency
  - Automatic Failover
  - No data loss in case of failover
  - Conflict resolution
  - Automatic Node provisioning (CLONE)
- **MySQL Shell** to configure, add, remove Instances
- **MySQL Router** to route application traffic

MySQL Replication vs InnoDB Cluster vs NDB Cluster

	MySQL Replication	MySQL InnoDB Cluster	MySQL NDB Cluster
Storage Engine	All	InnoDB	NDBCLUSTER
Distributed Architecture	Shared	Shared nothing	Shared nothing
Clustering Mode	Master + slaves	Multi-master (possible)	Multi-master (default)
Replication mode	Asynchronous	Paxos (Synchronous)	2PC (Synchronous)
Consistency Model	Weak Consistency	Medium Consistency	Strong Consistency
Sharding	No	No	**Yes**
Arbitration	No	No	Yes
Load Balancing	No	Partly via MySQL Router	Yes
NoSQL APIs	MySQL CRUD API	MySQL CRUD API	Native NDB API
Operational Complexity	Easy	Medium	High
Administration	Standard (MySQL)	Standard (MySQL)	Custom (MySQL + NDB)

When TO consider MySQL Cluster

- You need High Availability 6-9's (and strong consistency).
- You need sharding, either due to size or write performance.
- You need linear scalabillity when adding more nodes.
- You need predictable real-time response times.
- SQL and cross shard join support.
- You want a ACID distributed in-memory database.

When NOT to consider MySQL Cluster

- 3rd party applications
- Need for spatial, GIS or full text indexes
- Large dataset (> 50TB for NDB 7.6)
- Complex access pattern at scale

Introduction to MySQL Cluster

ORACLE

History of MySQL Cluster ”NDB”

The Network DataBase NDB

- MySQL Cluster aka Network DataBase NDB
- Designed/Developed at Ericcson in late 90’s
- Original design paper: ”Design and Modeling of a Parallel Data Server for Telecom Applications” from 1997 by Michael Ronström
- Originally written in PLEX (Programming Language for EXchanges) but later converted to C++.
- MySQL AB acquired Alzato (owned by Ericsson) late 2003.

History of MySQL Cluster ”NDB”

The Network DataBase NDB

- Databases services back then:
  - SCP/SDP (Service Control/Data Point) in Intelligent Networks.
  - HLR (Home Location Register) for keeping track of mobile phones/users.
  - Databases for network management especially real-time charging information.

History of MySQL Cluster ”NDB”

The Network DataBase NDB

- NDB was designed to:
  - Reliability, the availability class of the telecom databases should be 6. This means that downtime must be less than 30 seconds per year. This means that no planned down time of the system is allowed.
  - Performance, designed for high throughput, linear scalabillity when adding more servers (data nodes) for simple access patterns (PK lookups).
  - Real-time, data is kept in memory and system is designed for memory operations.

MySQL Cluster overview

Scaling Reads & Writes	Auto-sharding + Multi-master Transactional, ACID-compliant relational database
99.999% Availability	Shared-nothing design, no Single Point of Failure On-Line operations: Scale, Upgrade Schema, etc.
Real-Time Responsiveness	High-load, real-time performance Predictable low latency, bounded access times
SQL & NoSQL APIs	Complex, relational queries + Key/Value Access MySQL, Memcached, C++, Java, JPA, HTTP / REST
Low TCO, Open platform	GPL & Commercial editions Commodity hardware, management & monitoring tools

MySQL Cluster “Components”

API

SQL Node

(Applications)

- Standard SQL interface
- Scale out for performance
- Enables Geo Replication

NDB API

(Applications)

- Real-time applications
- C++/Java APIs
- Automatic failover & load balancing

CORE

Data Node

(Data Storage)

- Data storage (Memory & Disk)
- Automatic & User defined data partitioning
- Scale out for capacity and performance

MGM Node

(Management)

- Management, Monitoring & Configuration
- Arbitrator for split brain/network partitioning
- Cluster logs

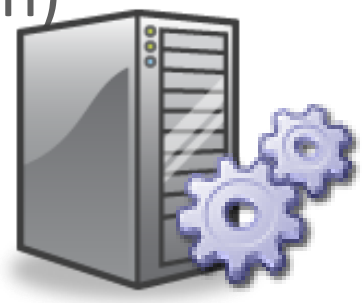
Data Nodes (ndb[mt]d)

- Stores data and indexes
  - In memory
  - Non-indexed data possible on disk
  - Contain several type of threads (blocks in the code), most important, LDM(LQH block), TUP, ACC and TC.
- Data check pointed to disk “LCP”
- Transaction coordination
- Handling fail-over
- Doing online backup

- All connect to each other
- Up to 48
  - Typically 2, 4.

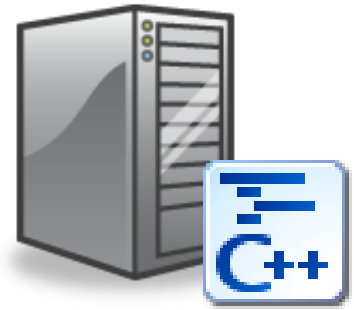
Management Nodes (ndb_mgmd)

- Responsible for configuration of cluster
  - All other processes connect initially to management node
- Holds the central cluster log file
- Act as Arbitrator
  - Prevents split-brain
- OK when not running (not critical when down)
  - Need to be running to start other processes
- 1 is minimum, 3 too many, 2 is OK



API Nodes

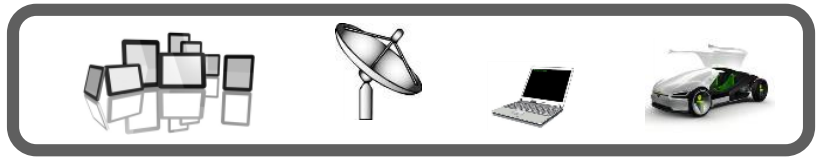
- Applications using the NDB API
  - Our connectors: C++/Java
- Fast
  - No SQL parsing
- Examples:
  - NDBCluster storage engine
  - ndb_restore
  - flexAsynch



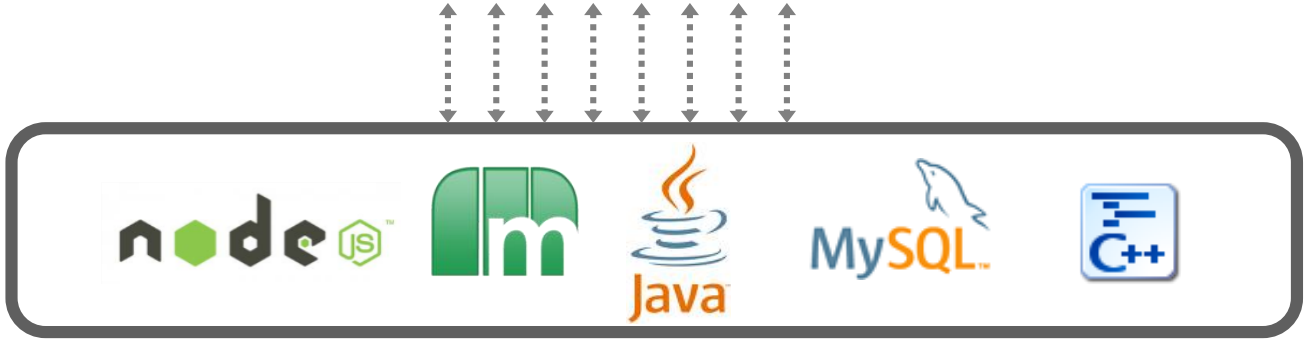
SQL Nodes

- MySQL using NDBCluster storage engine
  - Is just like any application connecting to data nodes
- Means transparent usage for most SQL applications
- Used to create tables
- Used for Geographical Replication
  - Binary logging all changes
- Can act as Arbitrator
- Connects to all Data Nodes

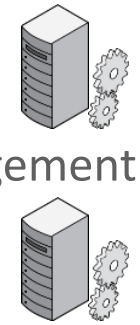
MySQL Cluster Architecture



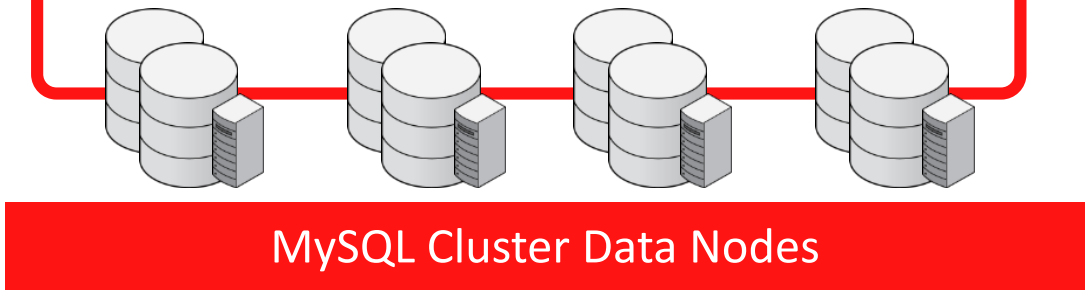
Clients



Application Layer



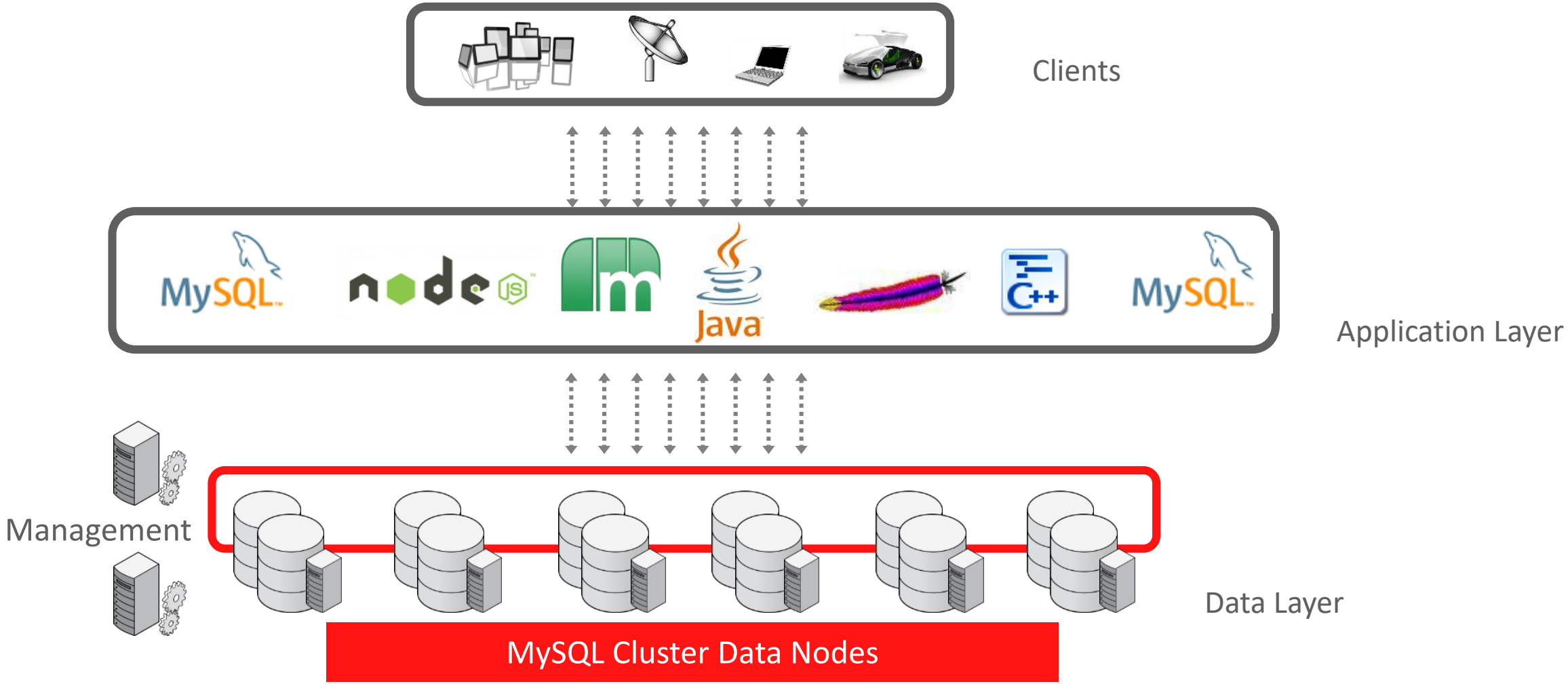
Management



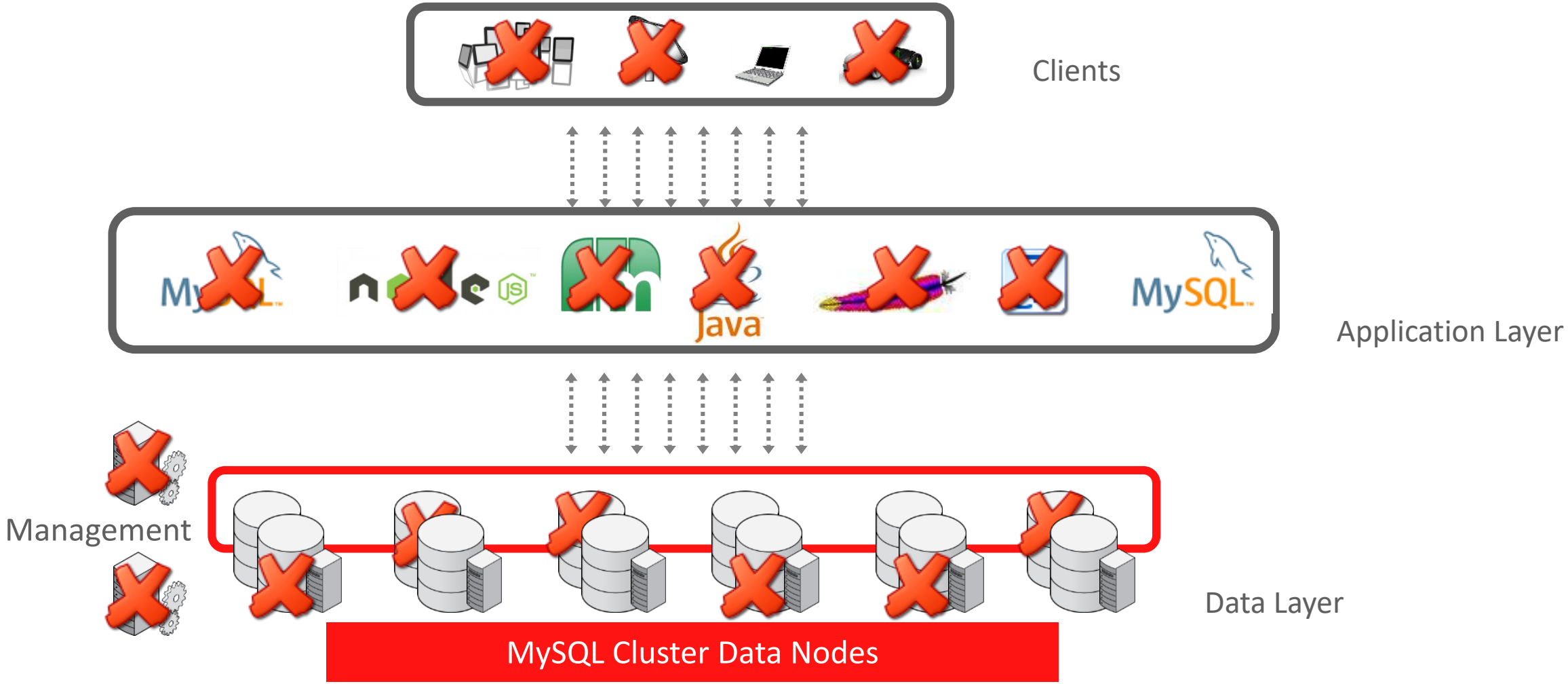
MySQL Cluster Data Nodes

Data Layer

MySQL Cluster Scaling

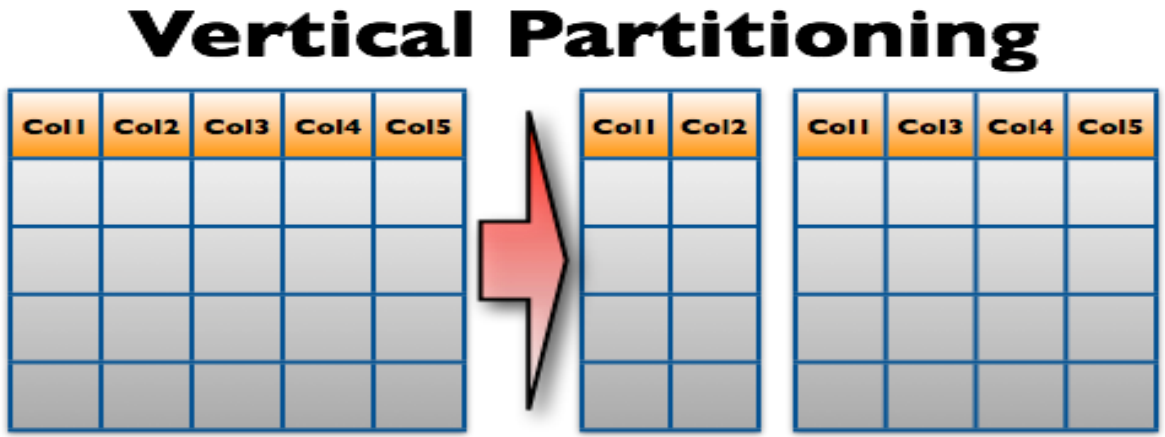


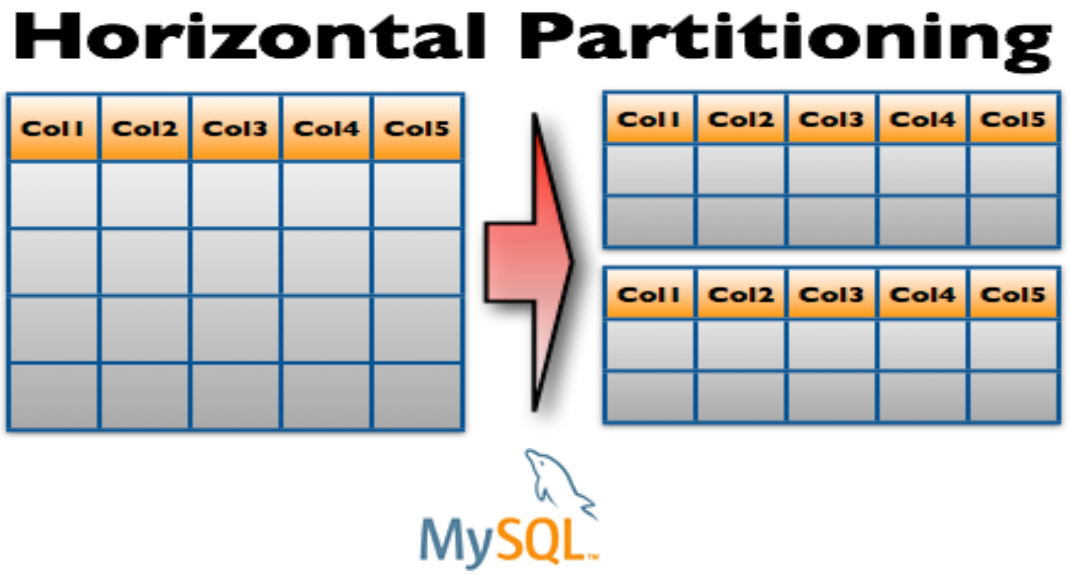
MySQL Cluster - Extreme Resilience



Partitioning I

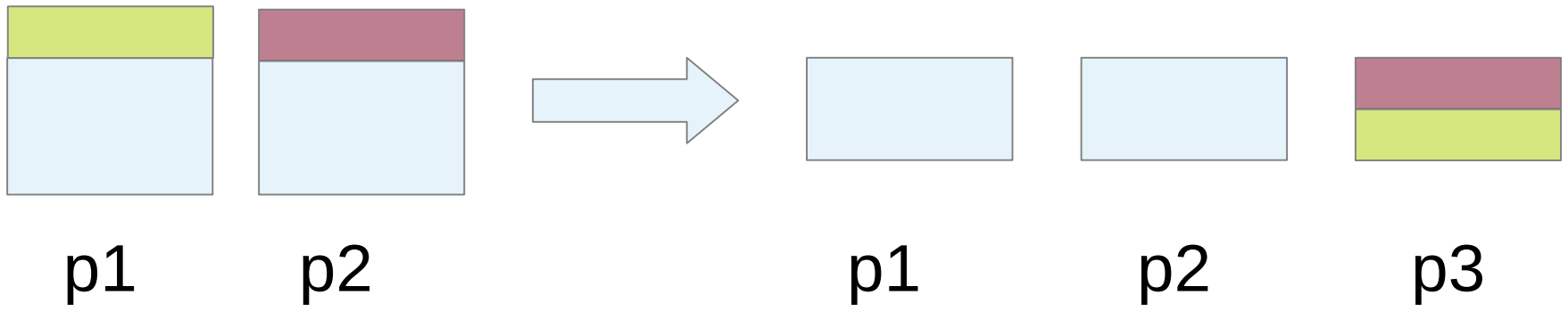
- Vertical Partitioning - 1:1 tables to reduce the size of rows, tables and indexes
- Horizontal Partitioning - 1 table split on multiple tables with different rows





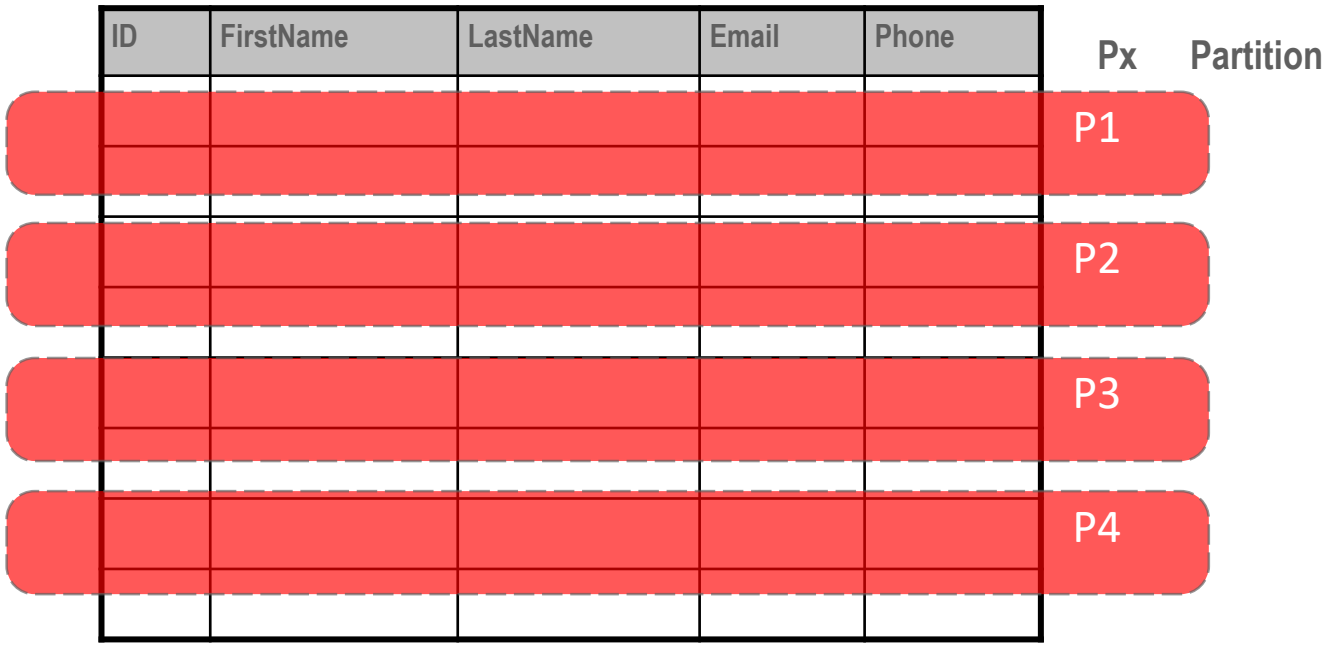
Data Partitioning II

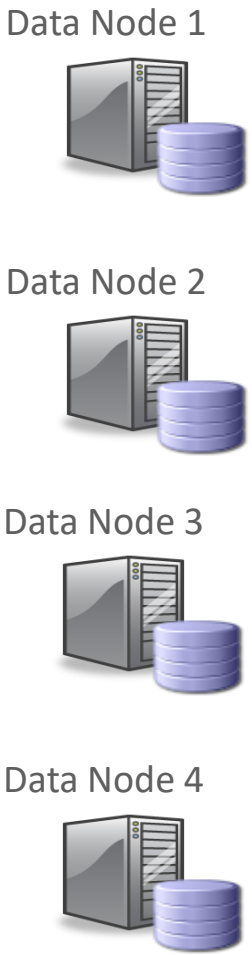
- Data is partitioned on primary key per default (P=md5(PK)%(NoOfNodes))
- HASH value of PK, only selective if you provide full PK not “left most”
- Linear hashing, data is only moved away (low impact of reorganize)



Automatic Data Partitioning

Table T1

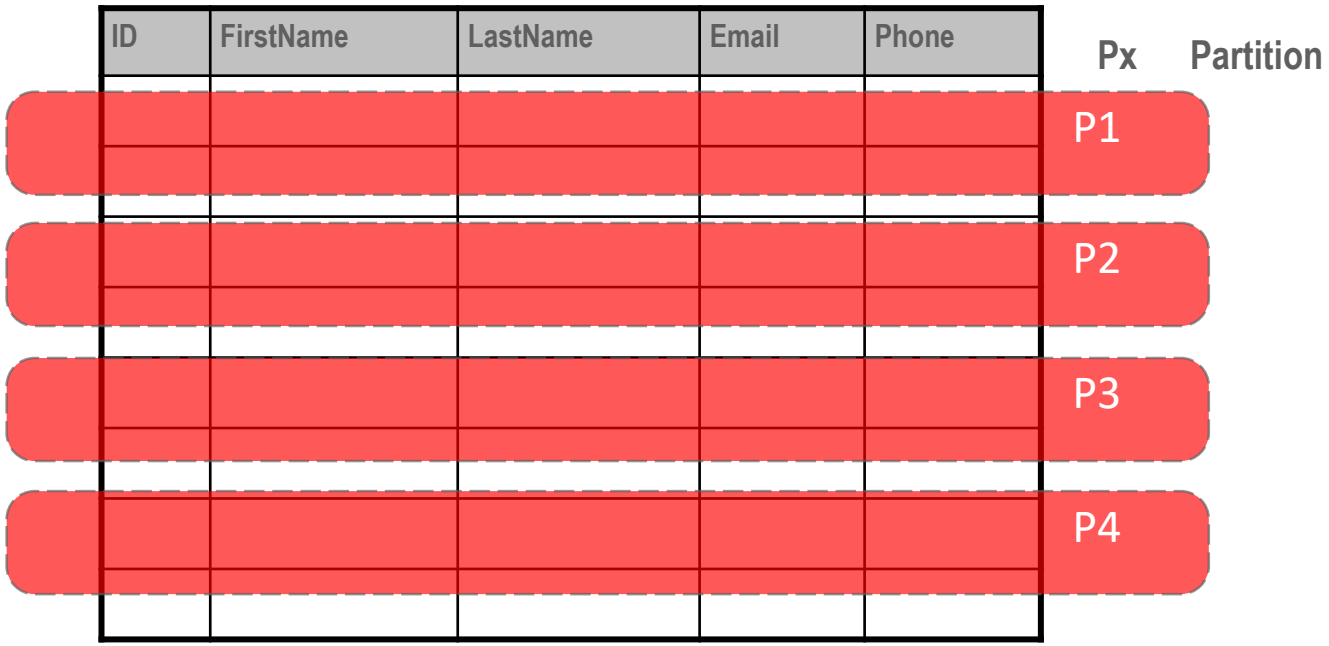




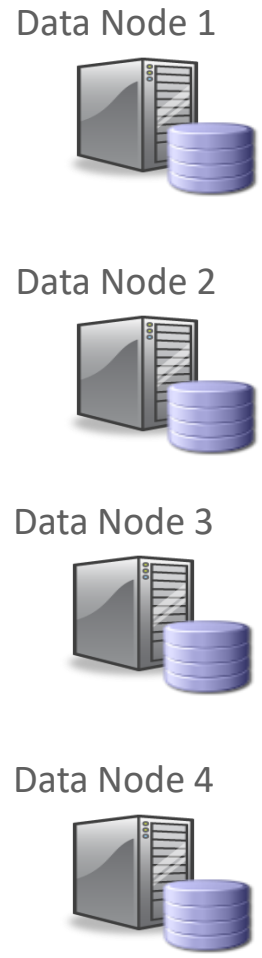
- A partition is a portion of a table (real hashmap has 3840 partitions)
- Number of partitions = number of data nodes (simplified view)
- Horizontal partitioning

Automatic Data Partitioning

Table T1



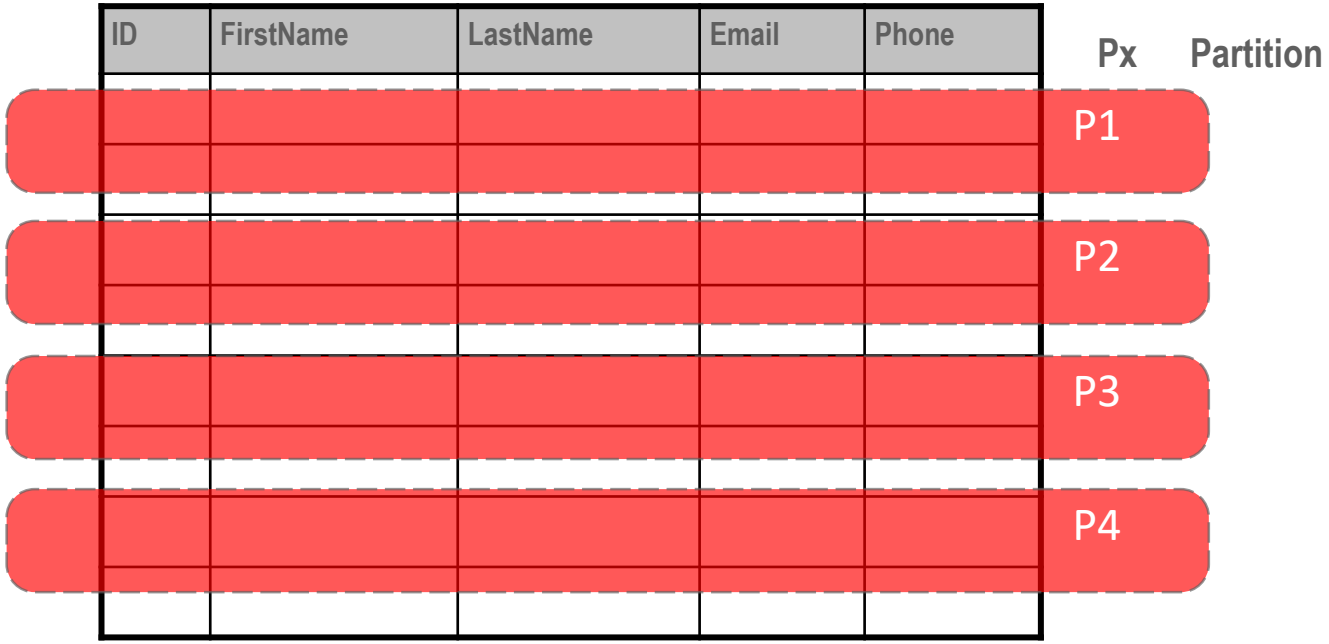
A fragment is a partition of your data on data nodes
Number of fragments = # of partitions * # of replicas



Automatic Data Partitioning

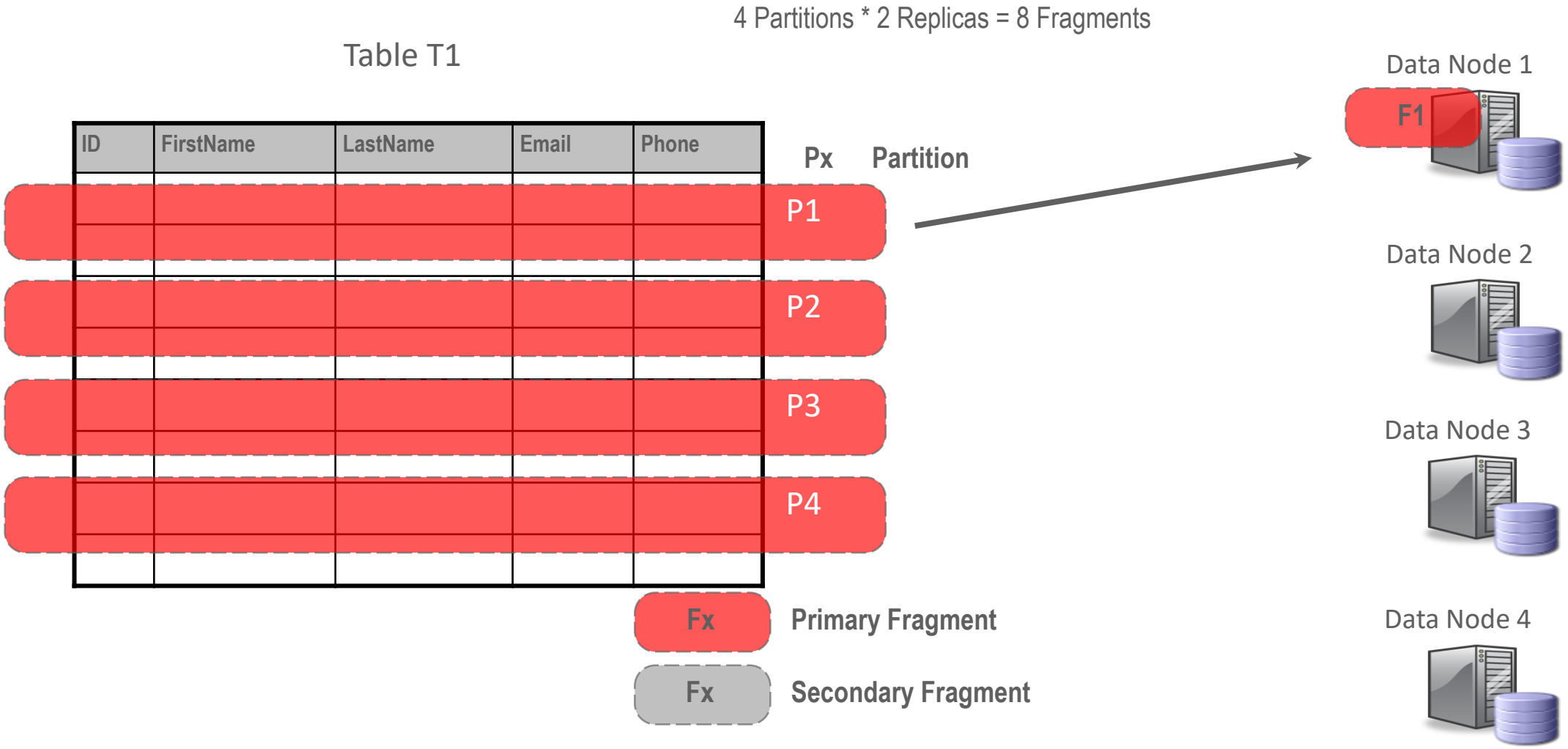
4 Partitions * 2 Replicas = 8 Fragments

Table T1

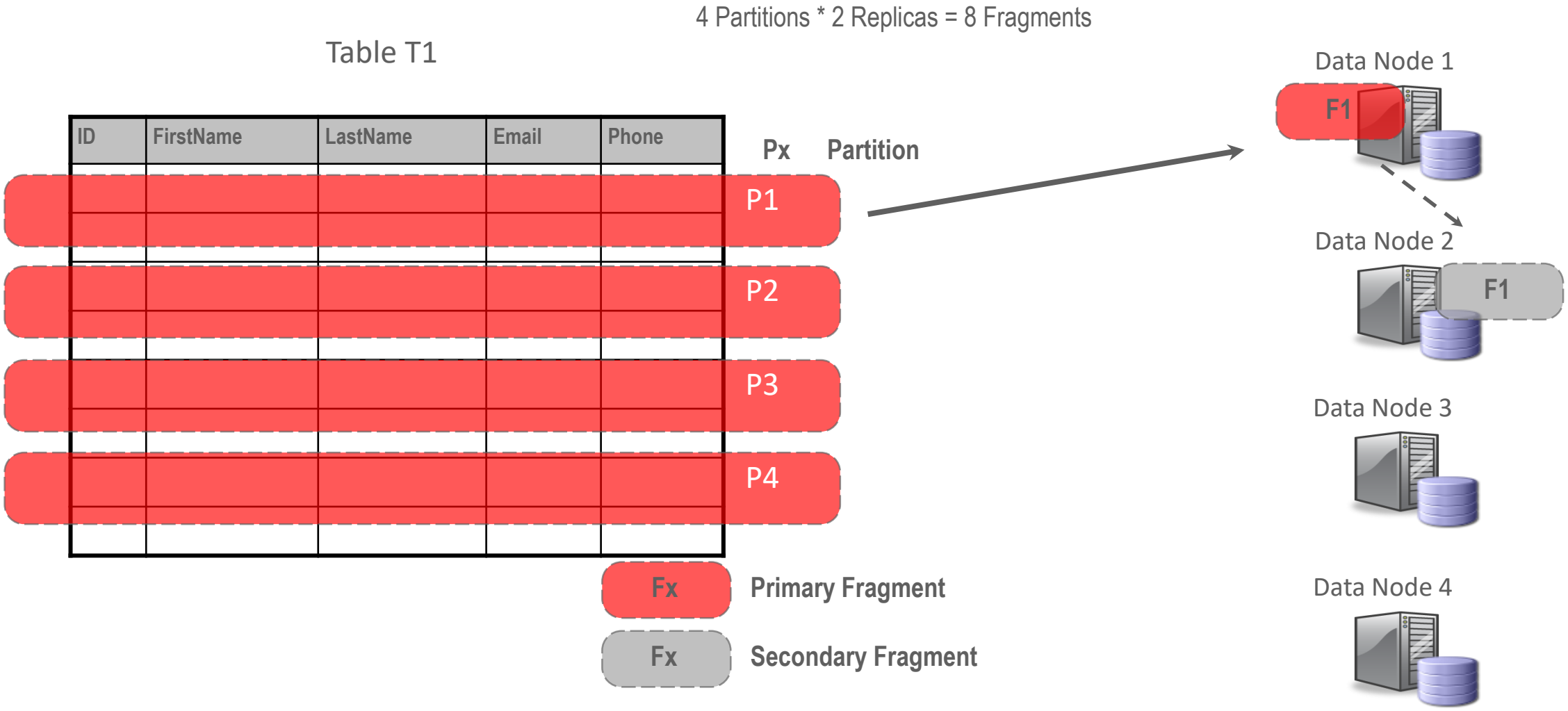


A fragment can be primary or secondary/backup
Number of fragments = # of partitions * # of replicas

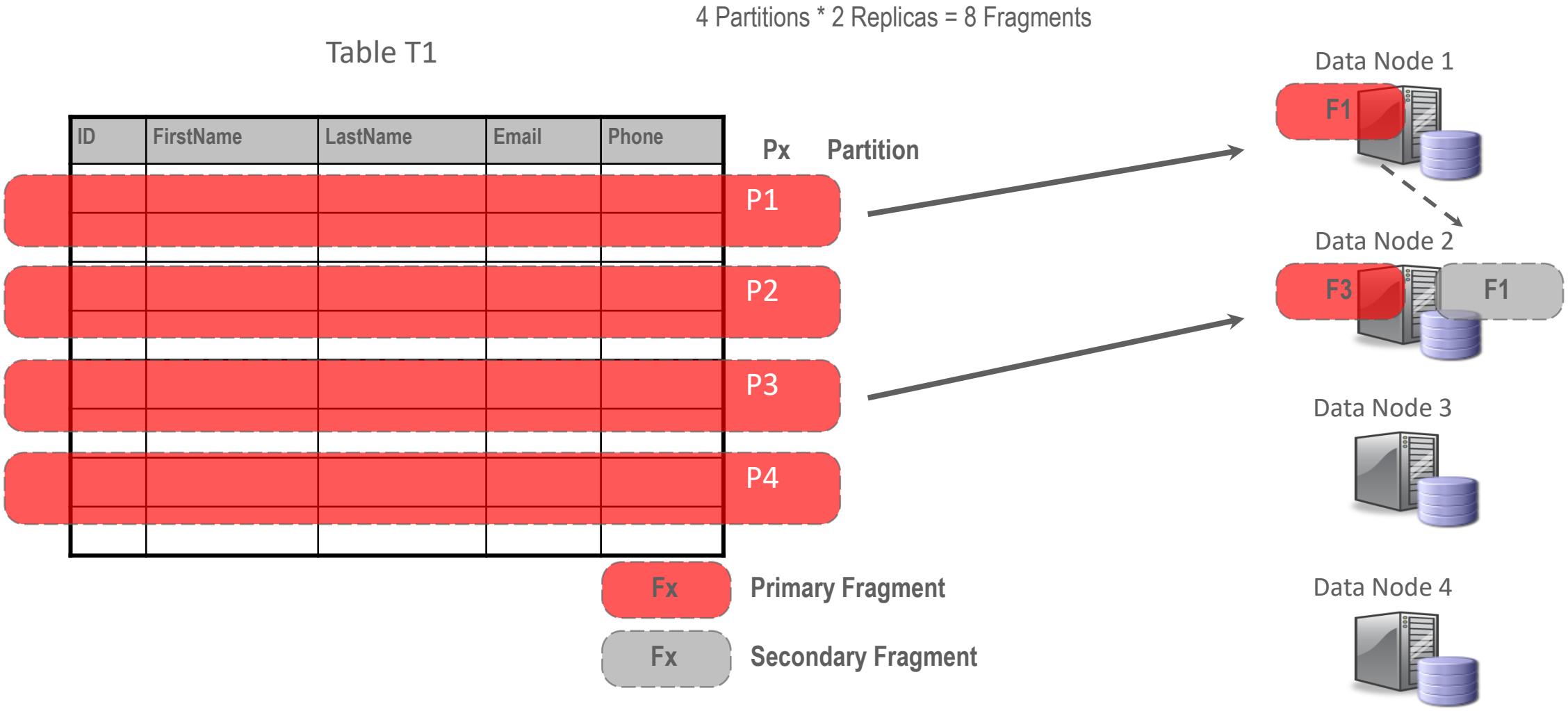
Automatic Data Partitioning



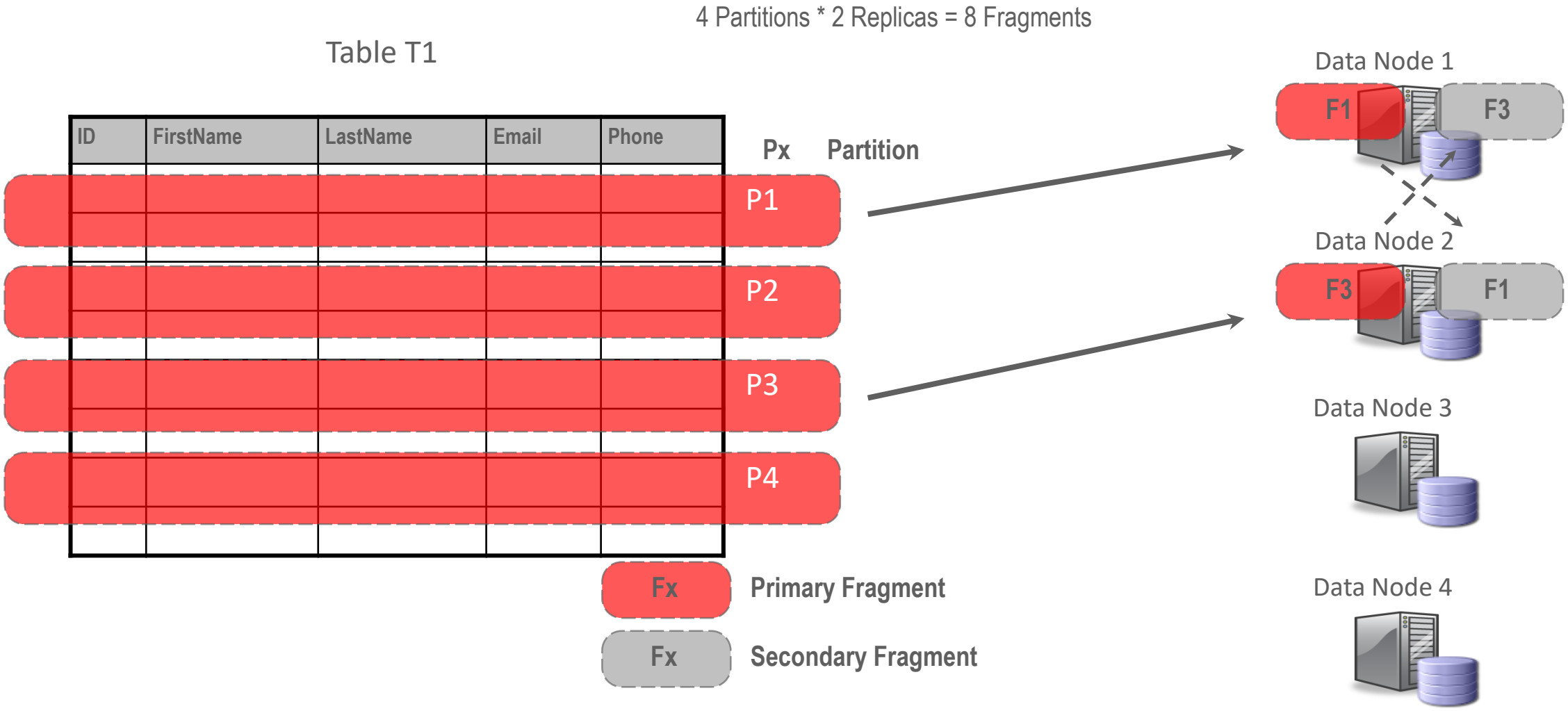
Automatic Data Partitioning



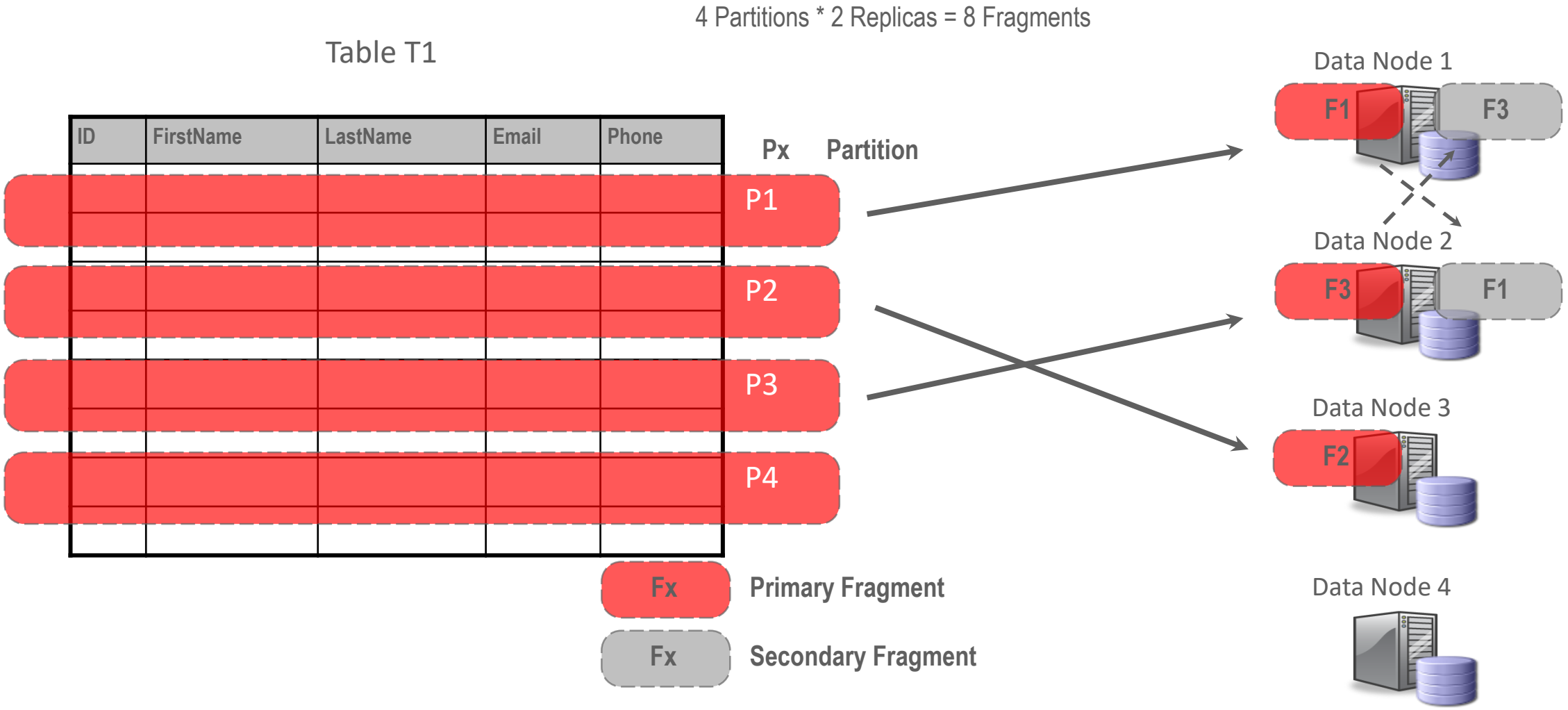
Automatic Data Partitioning



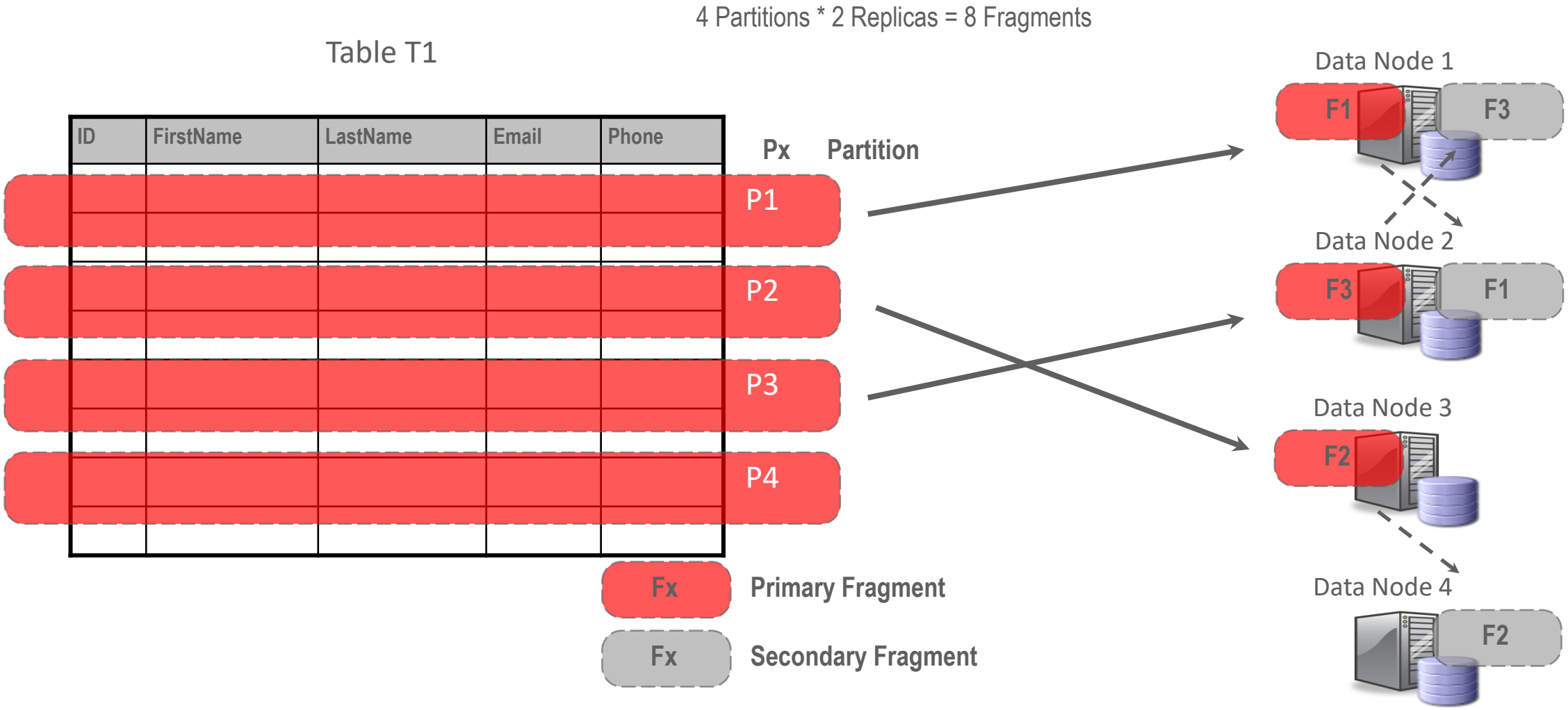
Automatic Data Partitioning



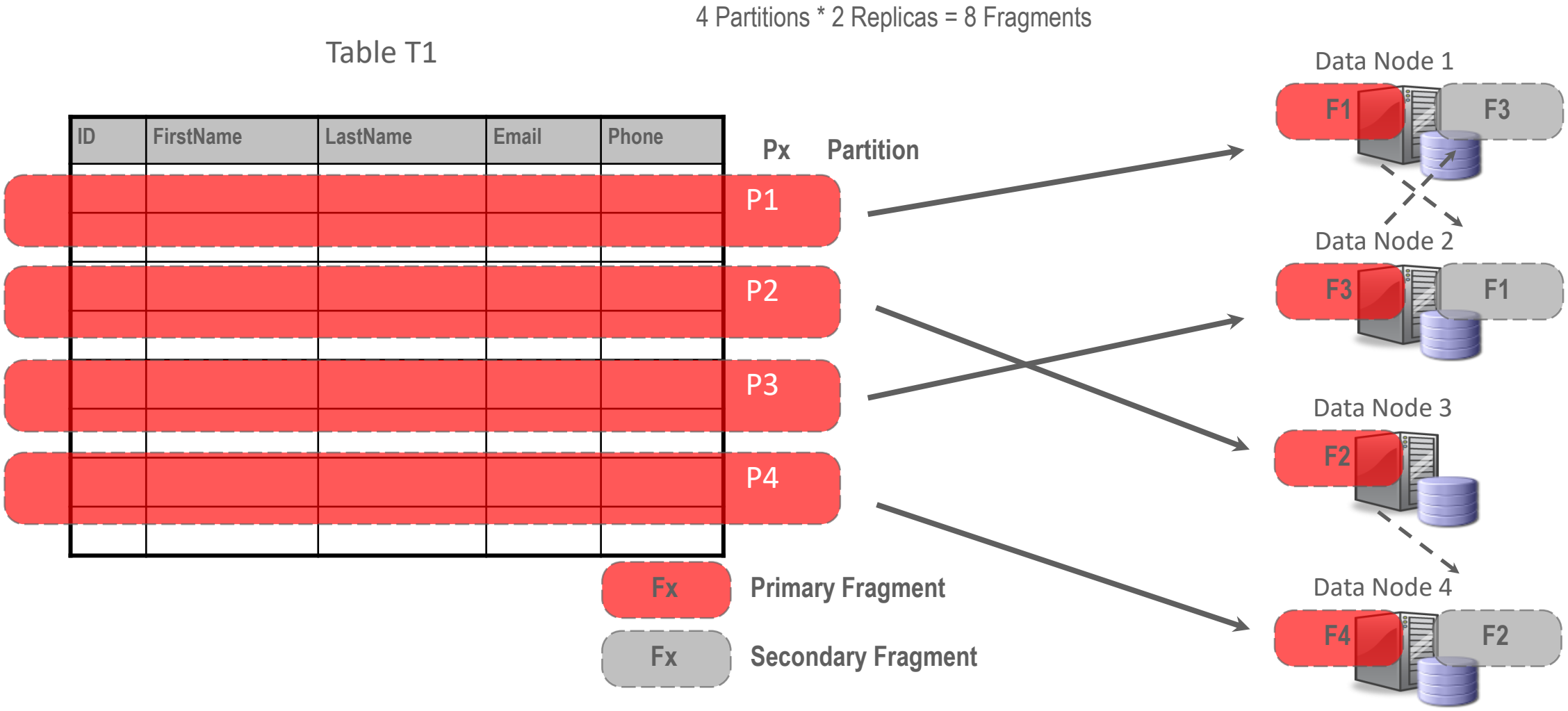
Automatic Data Partitioning



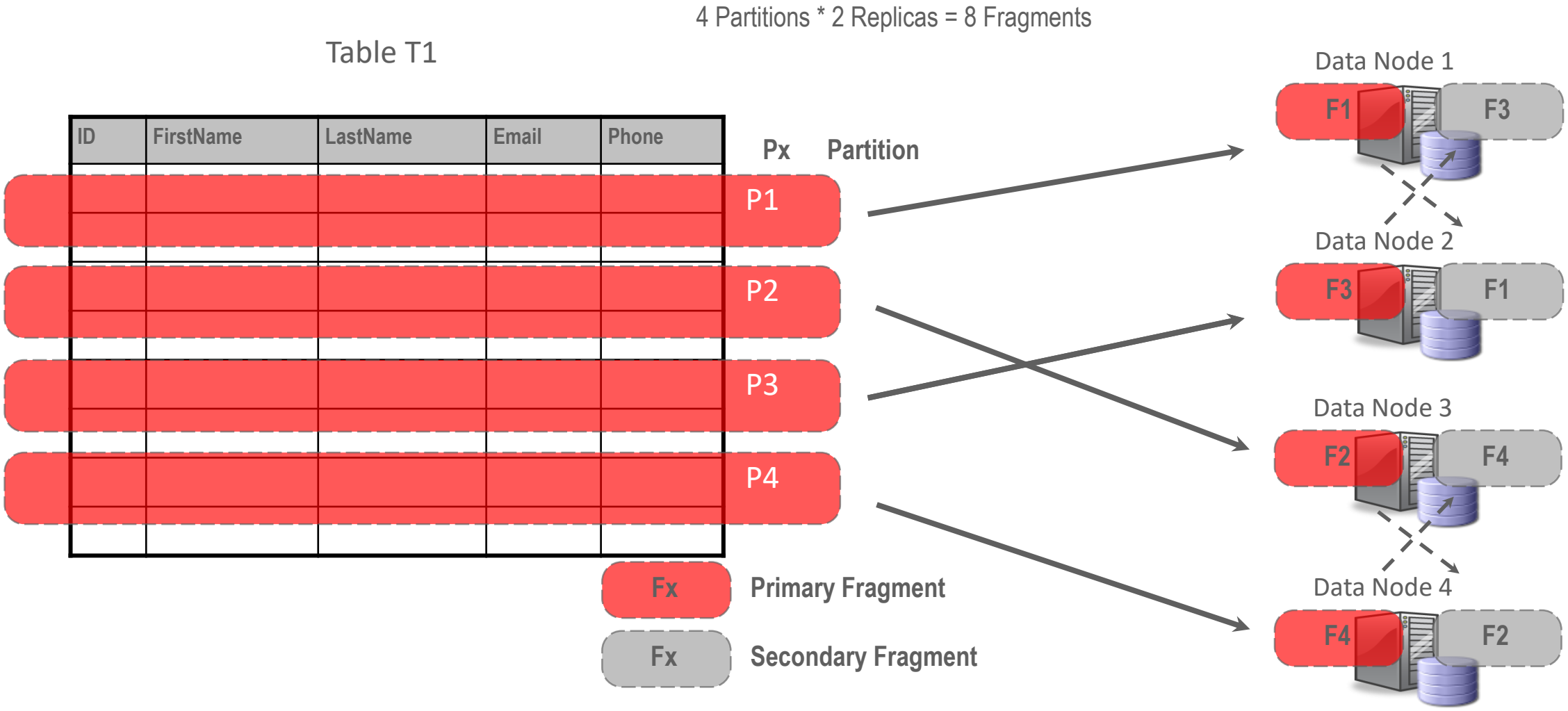
Automatic Data Partitioning



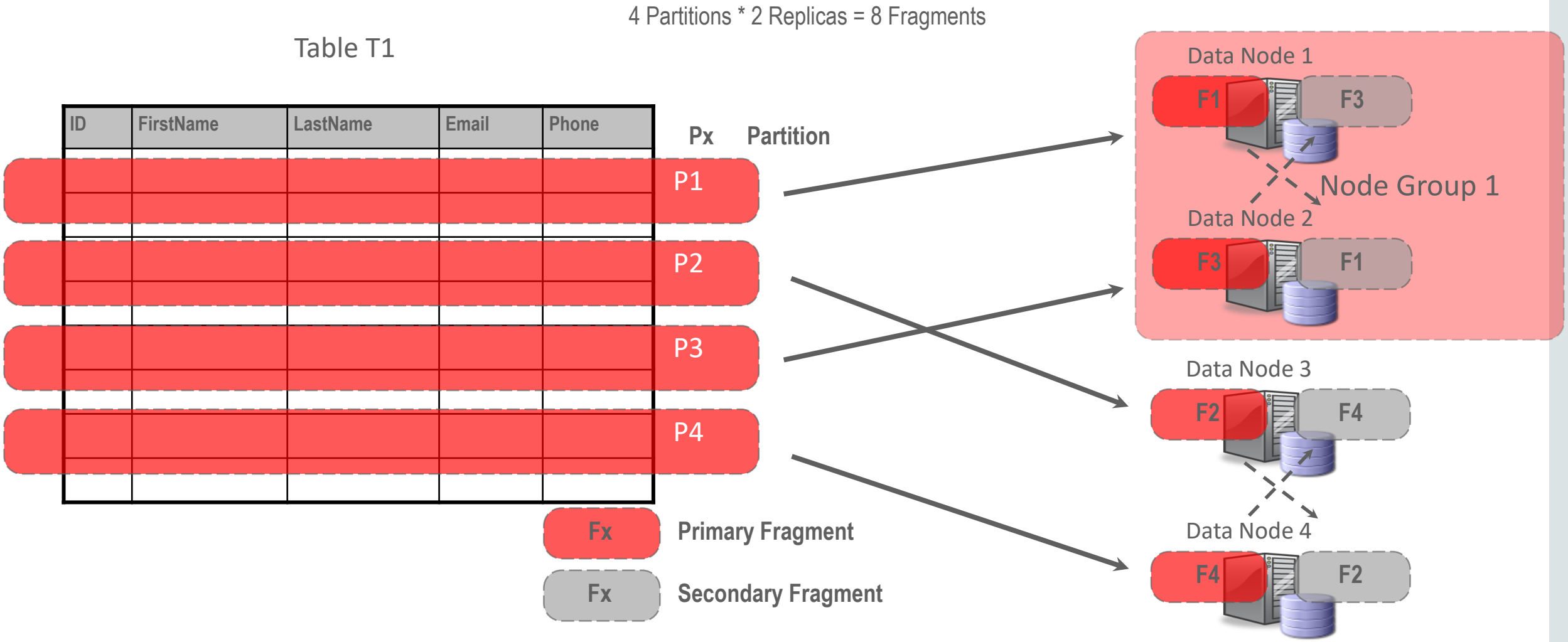
Automatic Data Partitioning



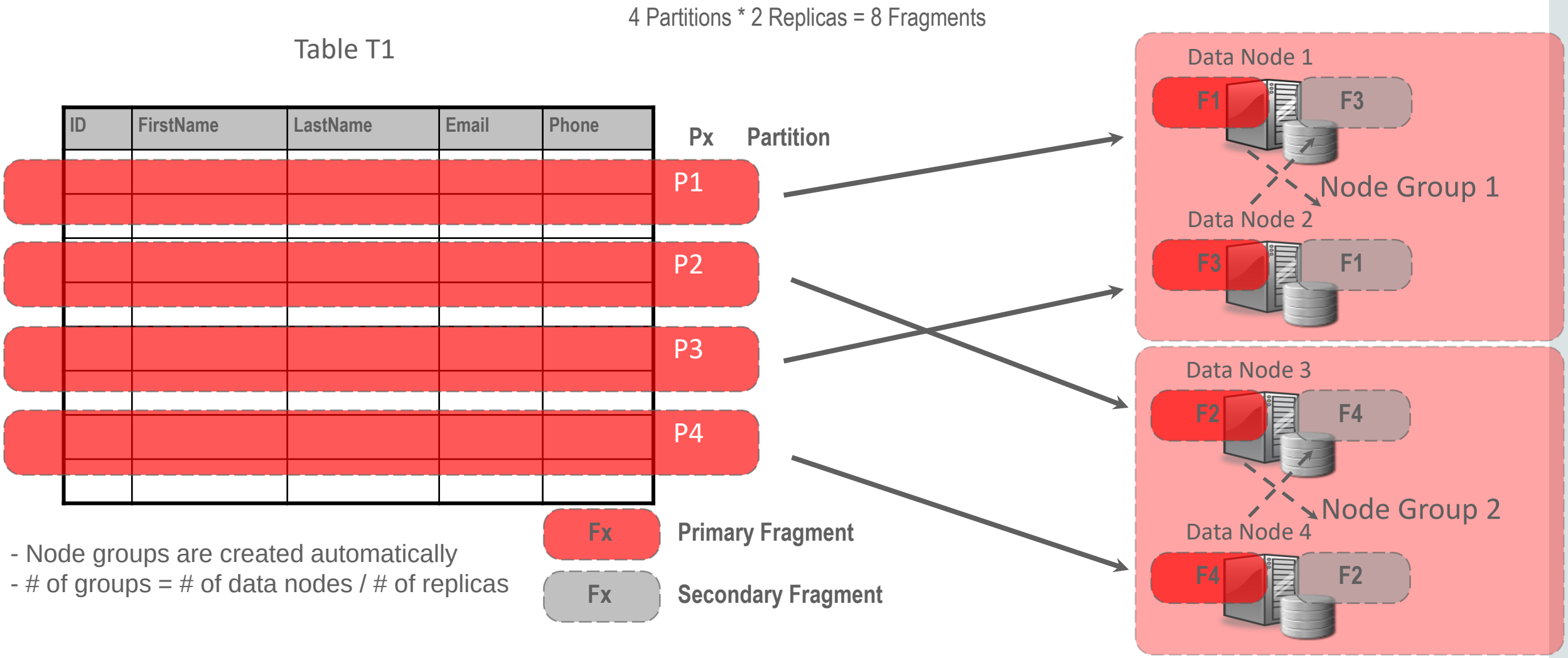
Automatic Data Partitioning



Automatic Data Partitioning



Automatic Data Partitioning

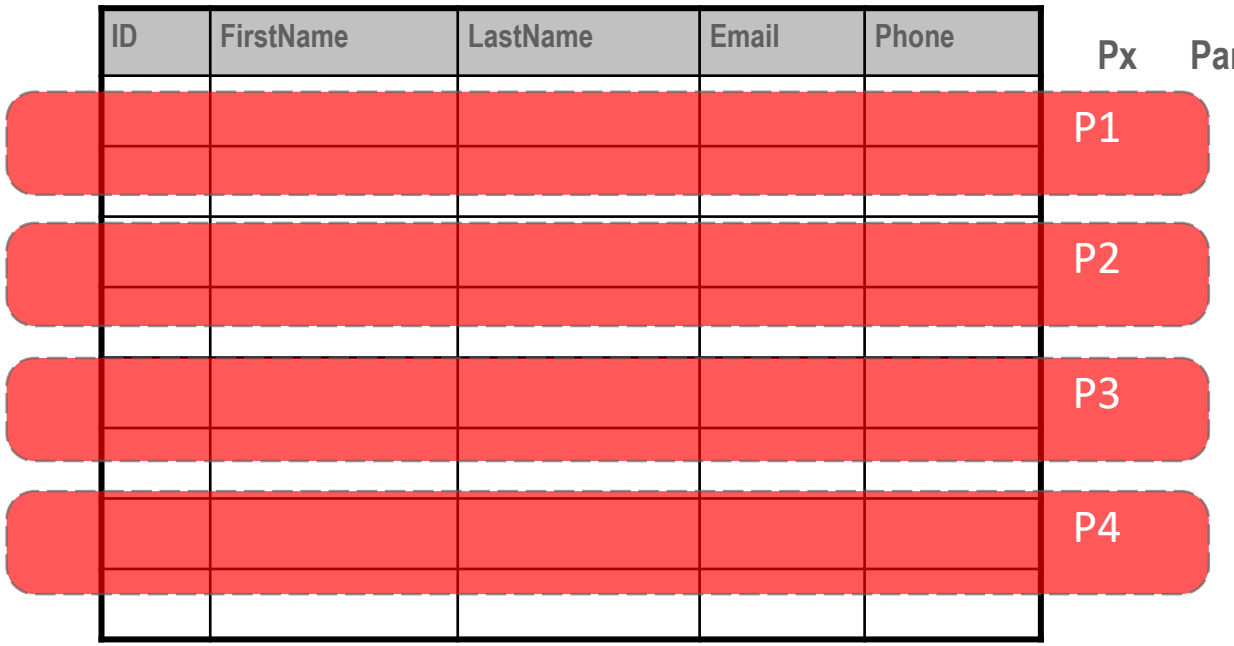


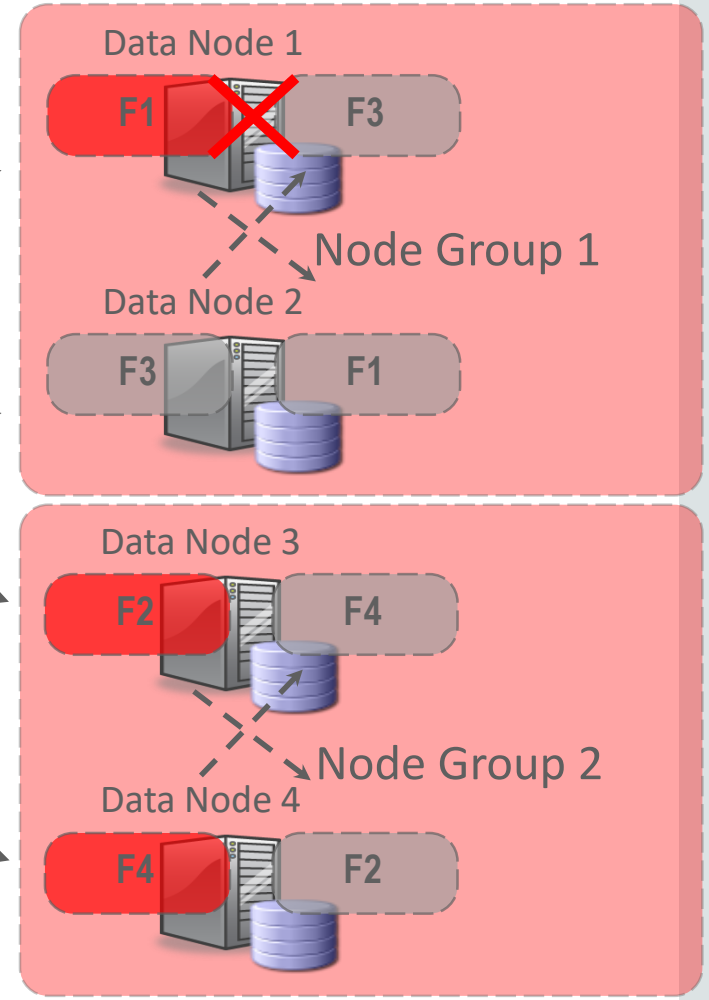
- Node groups are created automatically
- # of groups = # of data nodes / # of replicas

Automatic Data Partitioning

4 Partitions * 2 Replicas = 8 Fragments

Table T1



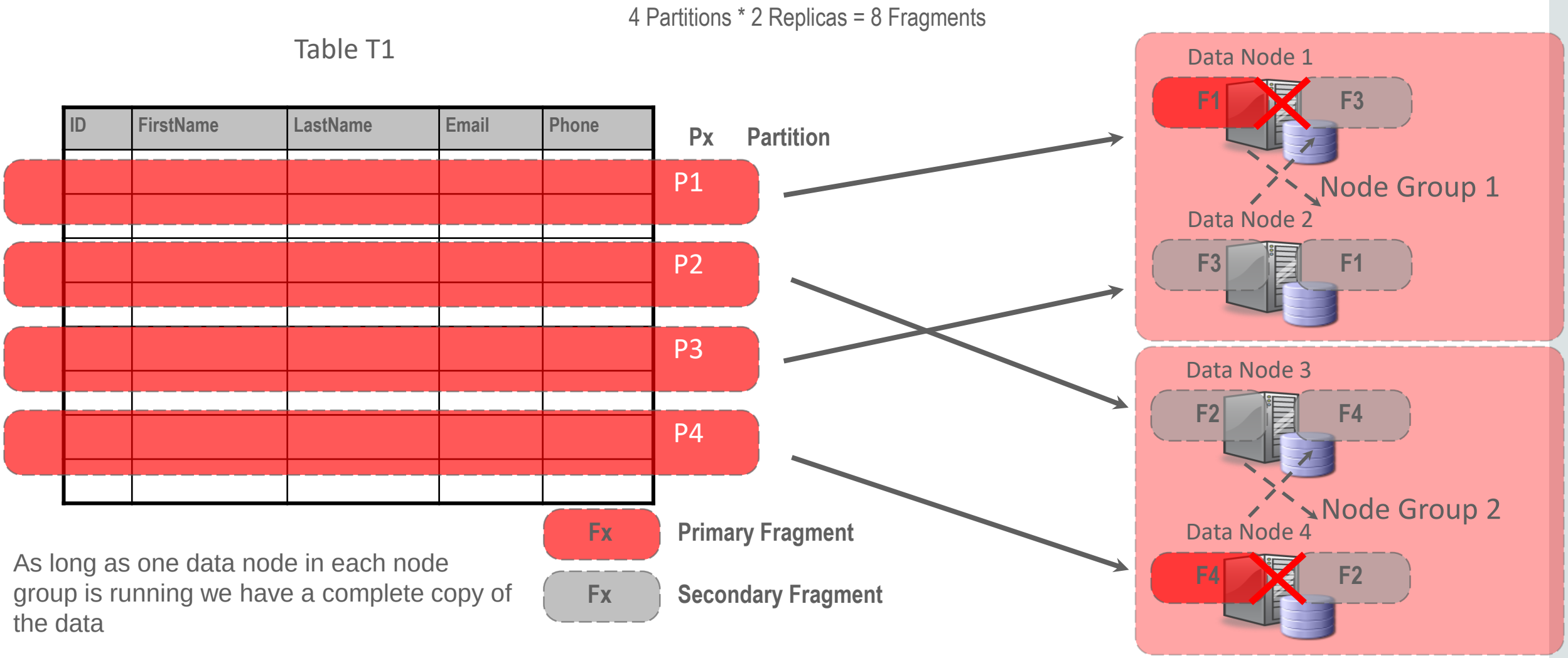


Fx Primary Fragment

Fx Secondary Fragment

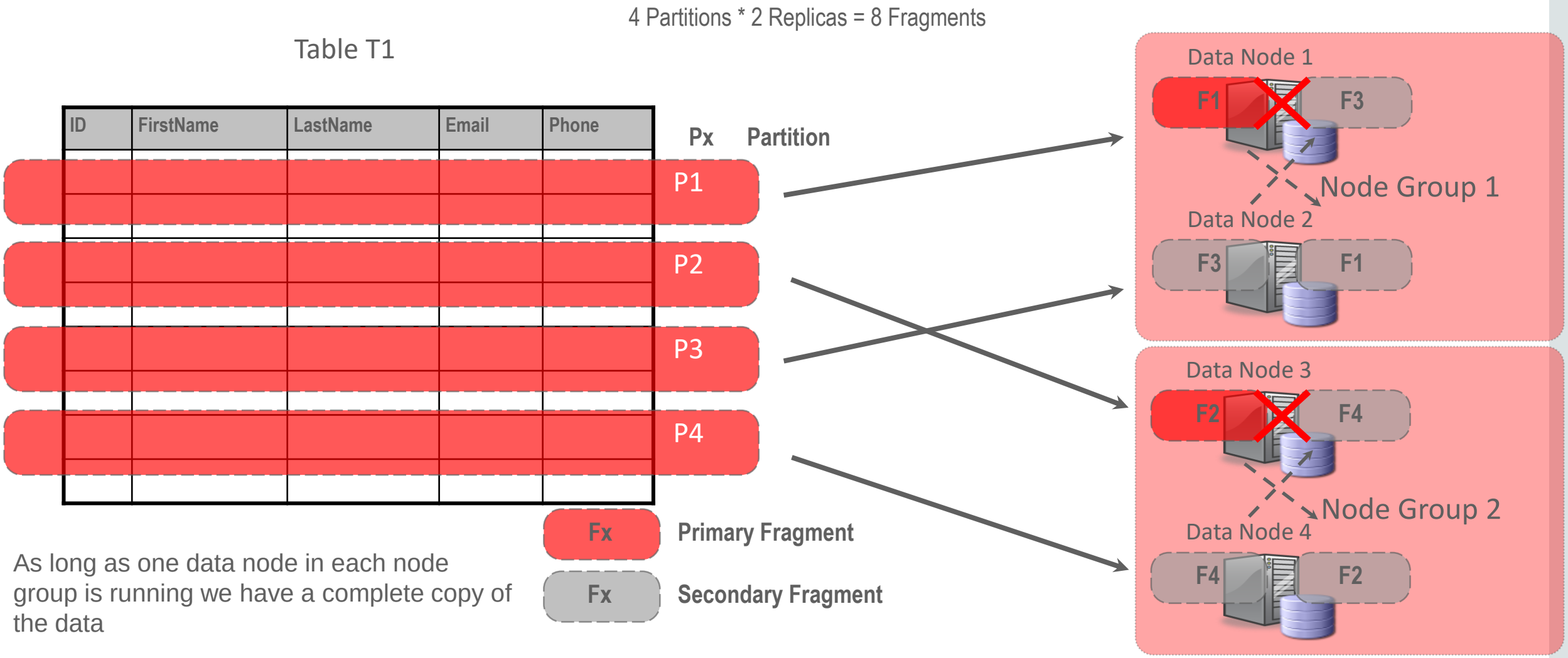
As long as one data node in each node group is running we have a complete copy of the data

Automatic Data Partitioning



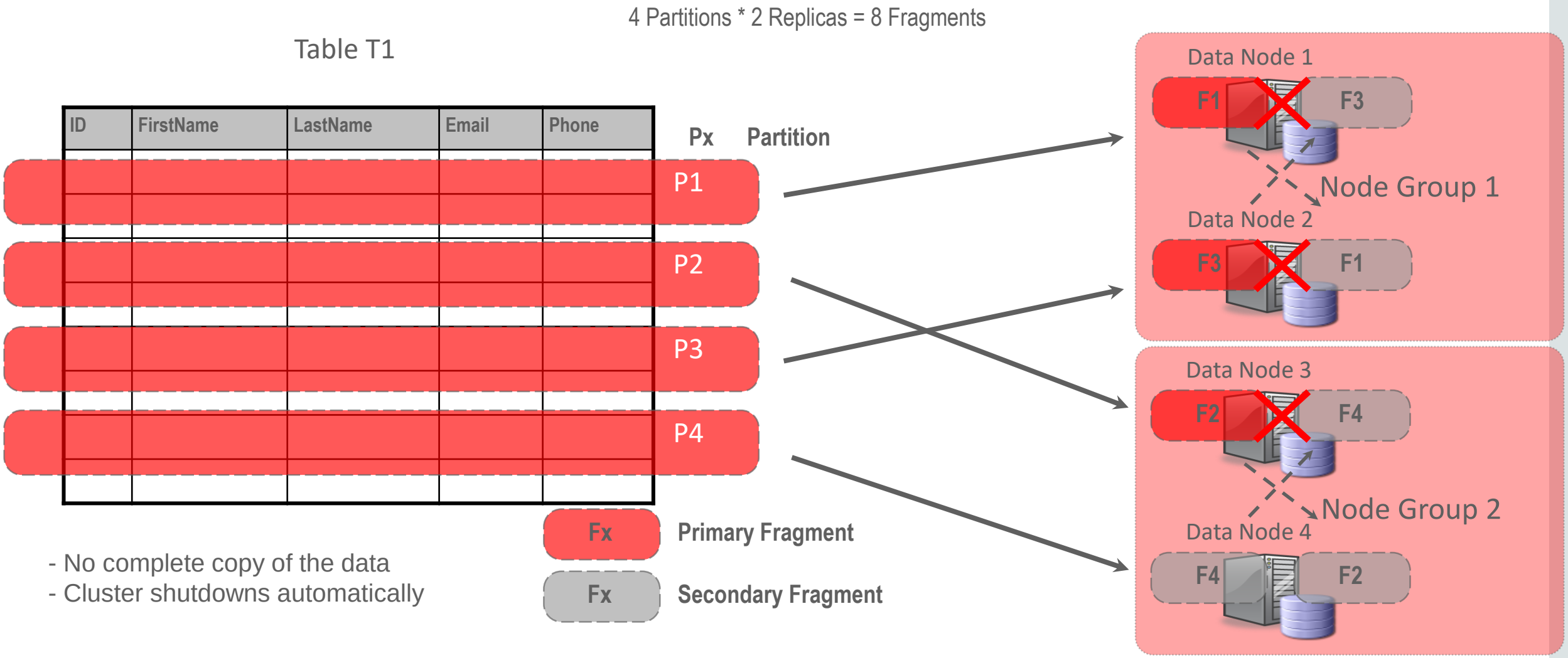
As long as one data node in each node group is running we have a complete copy of the data

Automatic Data Partitioning



As long as one data node in each node group is running we have a complete copy of the data

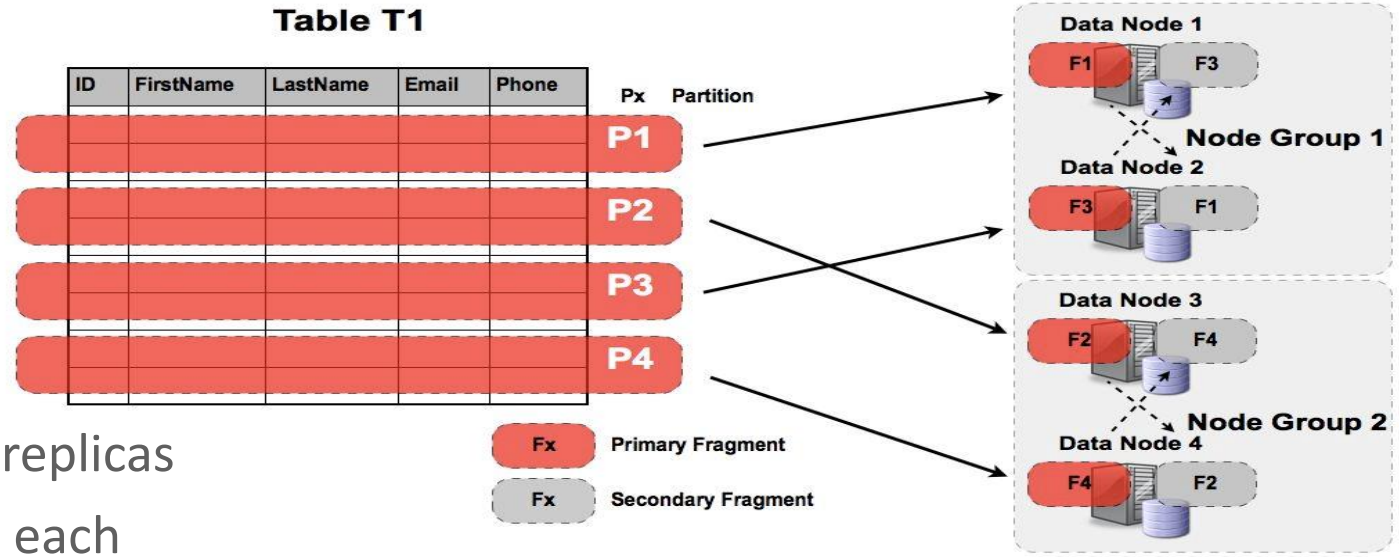
Automatic Data Partitioning



- No complete copy of the data
- Cluster shutdowns automatically

Data Partitioning III

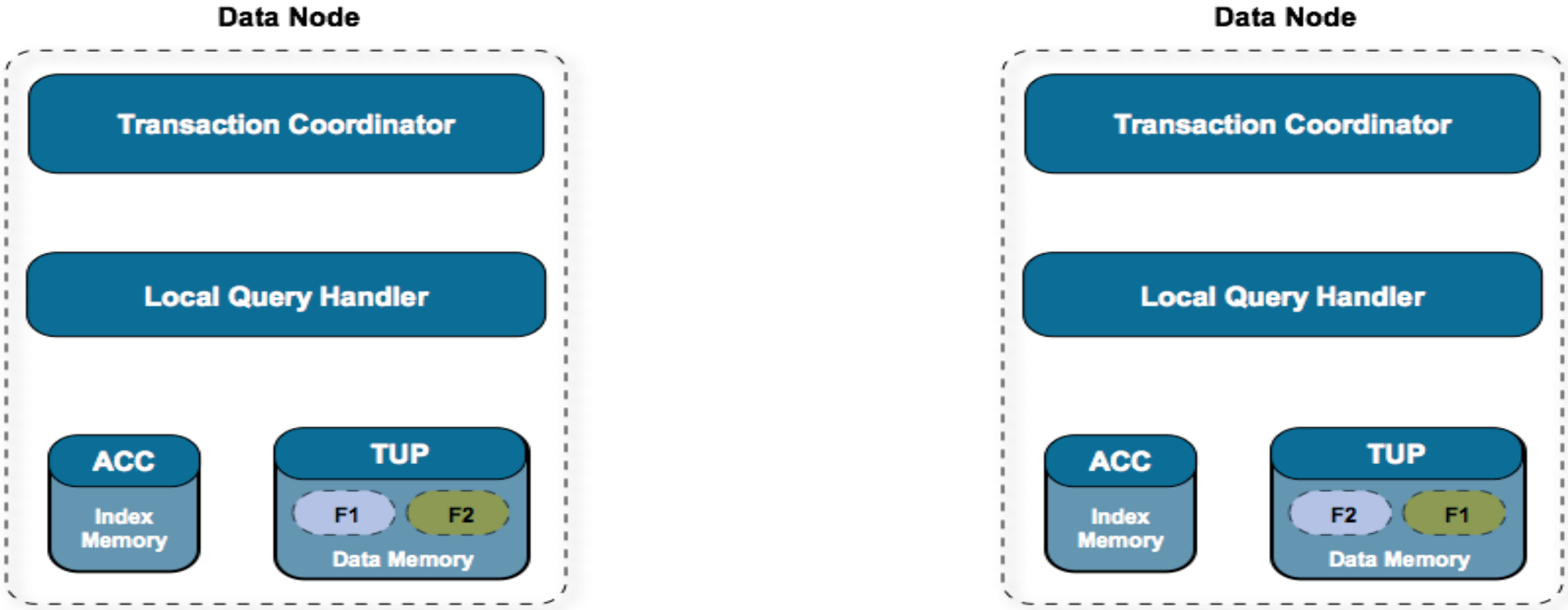
- Partition
  - Horizontal partitioning
  - A portion of a table, each partition contains a set of rows
- Replica
  - A complete copy of the data
- Node Group
  - Created automatically
  - # of groups = # of data nodes / # of replicas
  - As long as there is one data node in each node group we have a complete copy of the data



Internal Replication “2-Phase Commit”

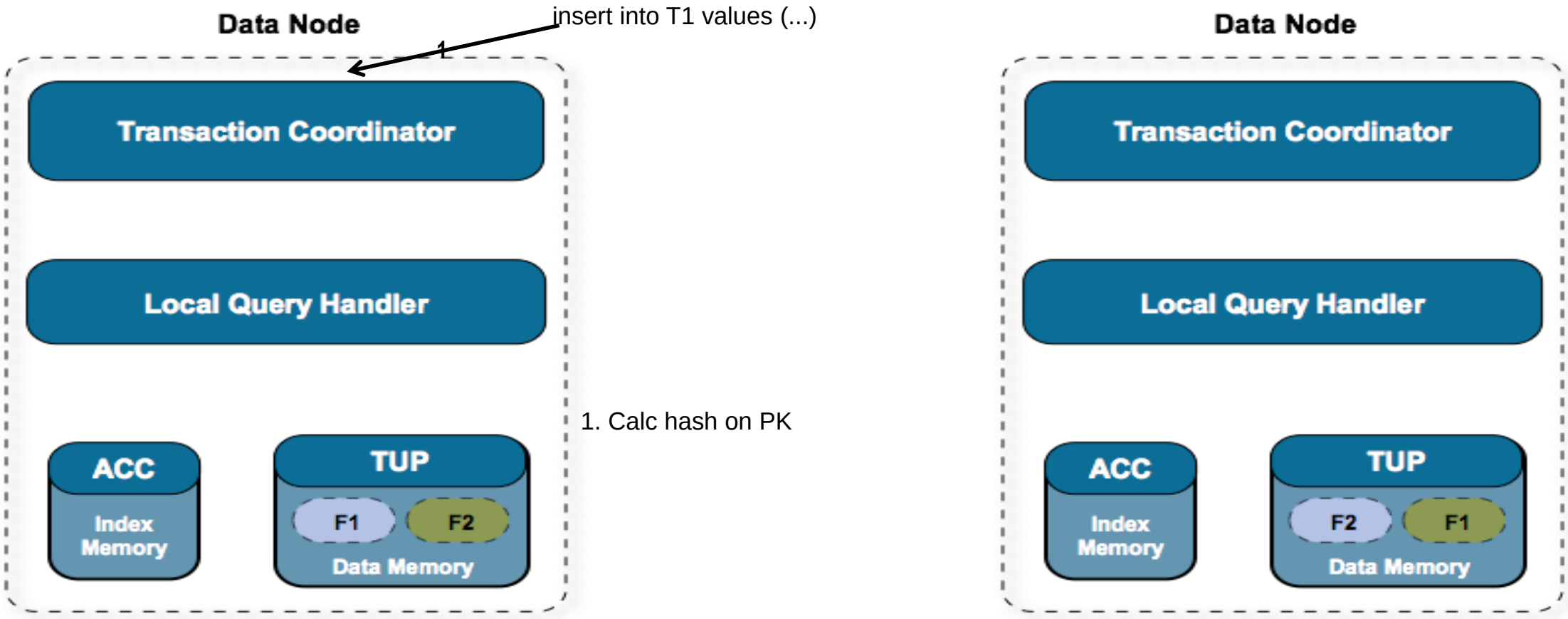
Internal Replication “2-Phase Commit”

Simplistic view of two Data Nodes



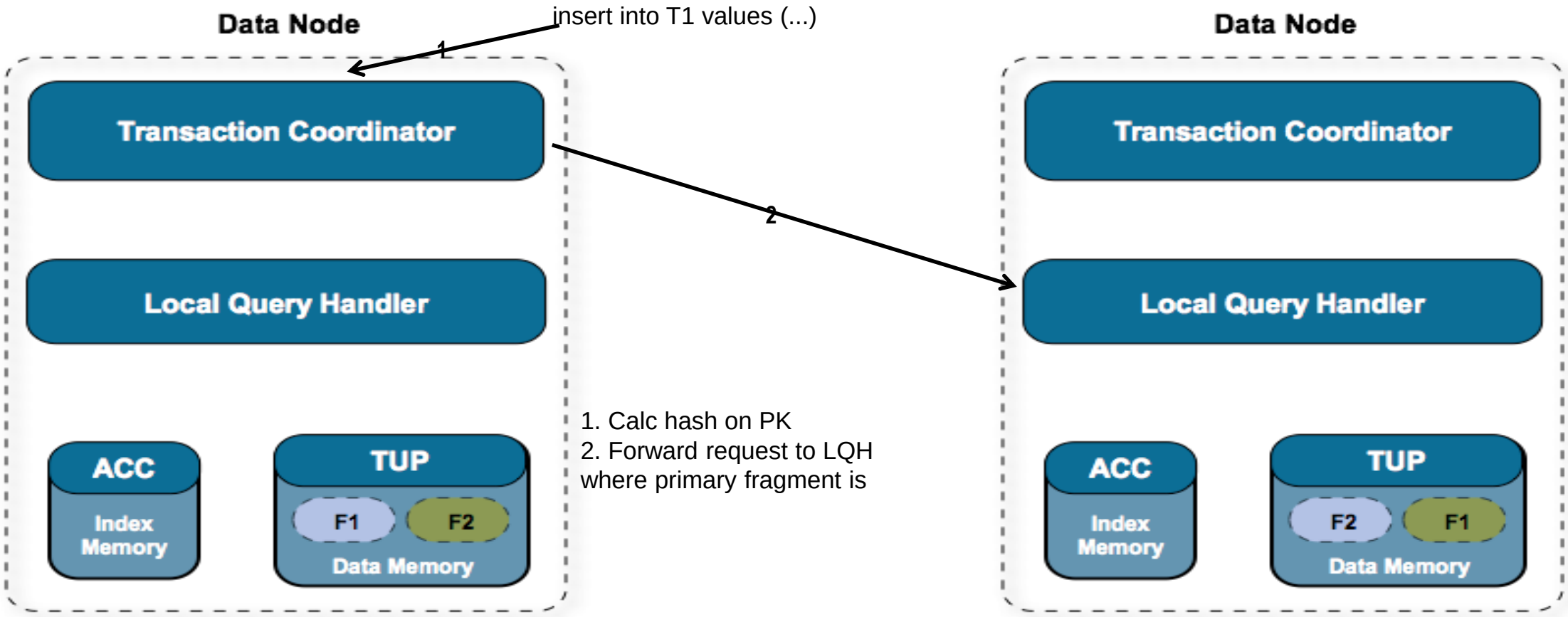
Internal Replication “2-Phase Commit”

Prepare Phase



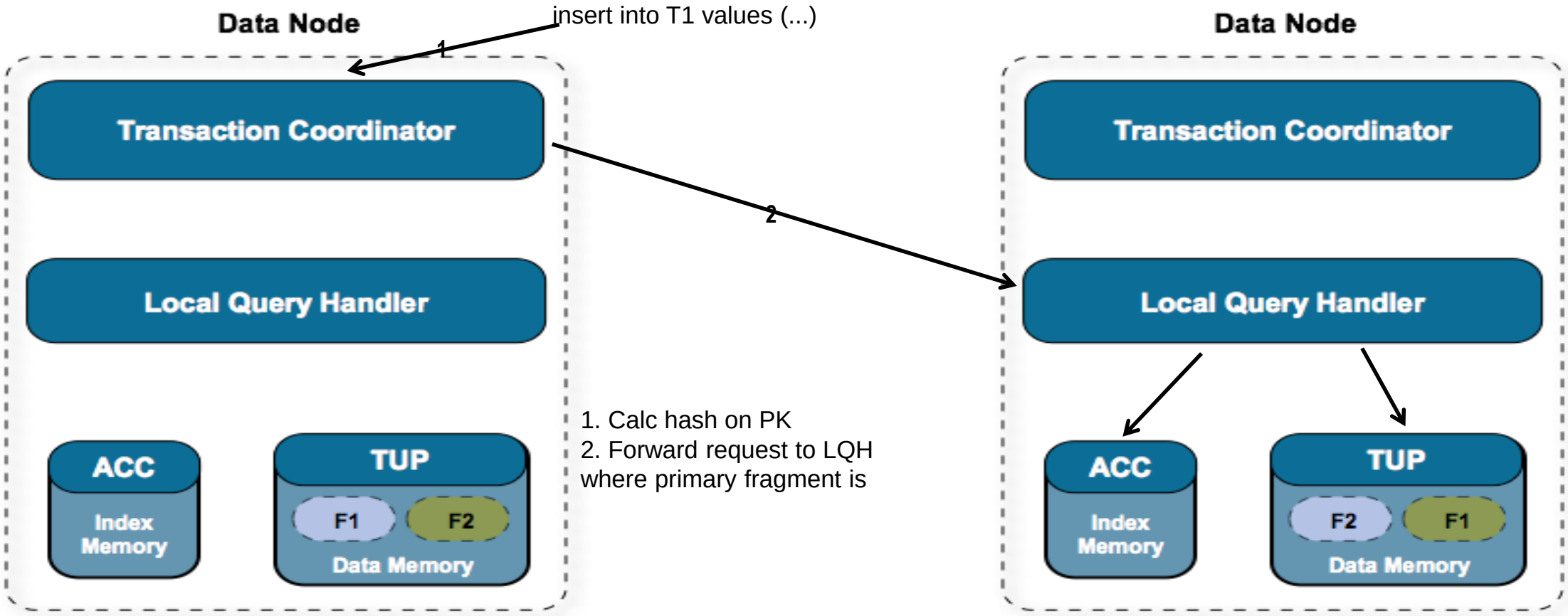
Internal Replication “2-Phase Commit“

Prepare Phase



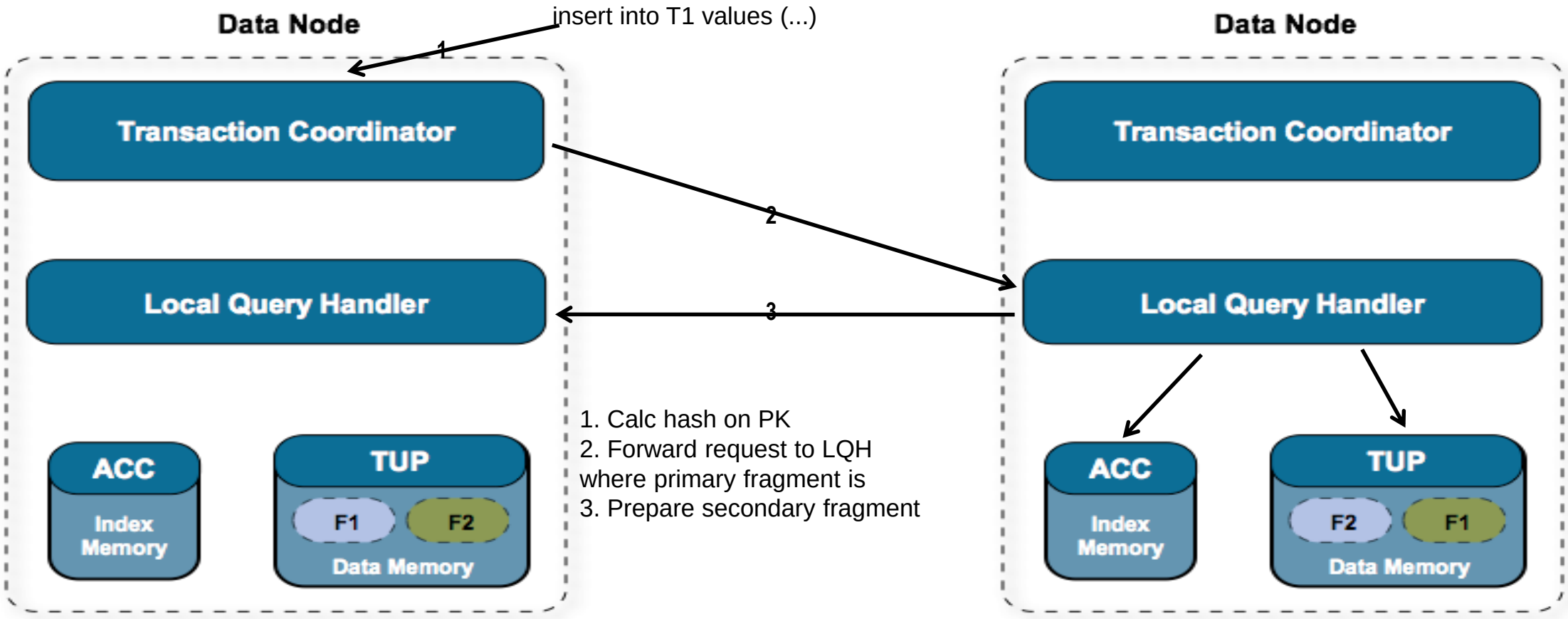
Internal Replication “2-Phase Commit”

Prepare Phase



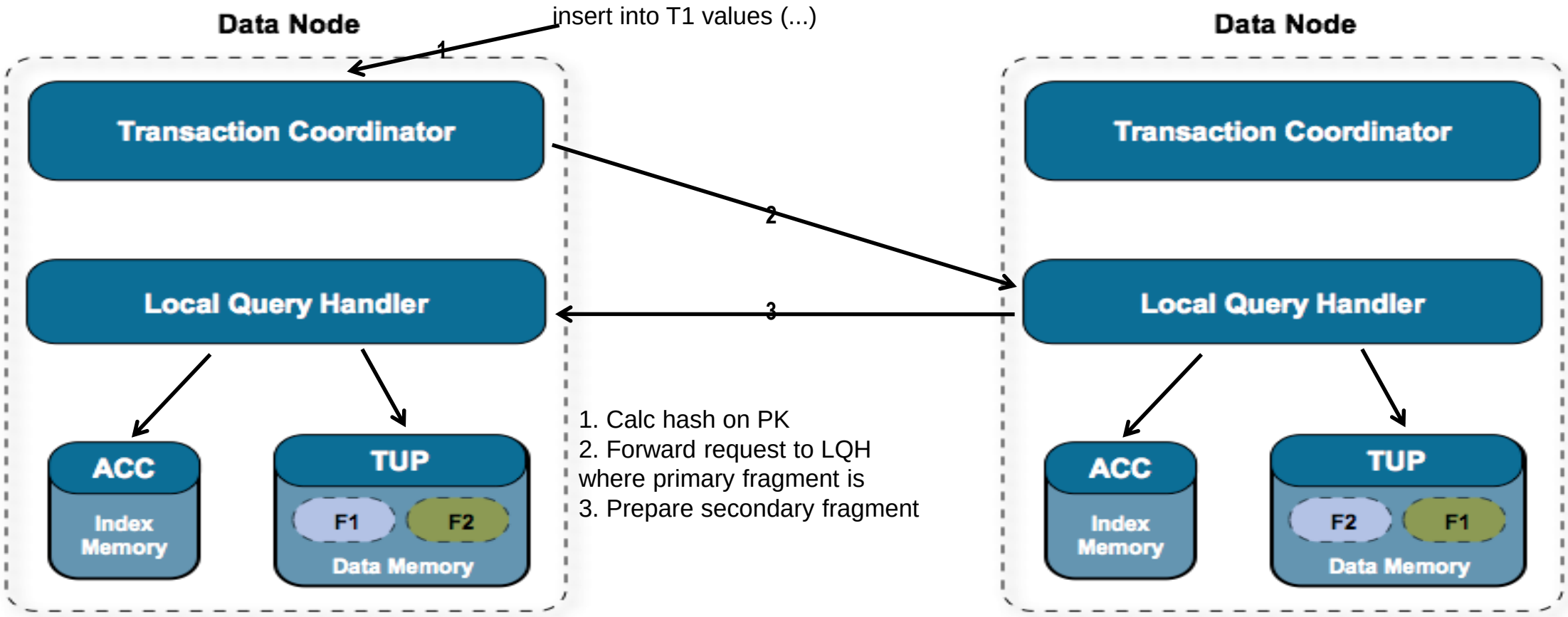
Internal Replication “2-Phase Commit“

Prepare Phase



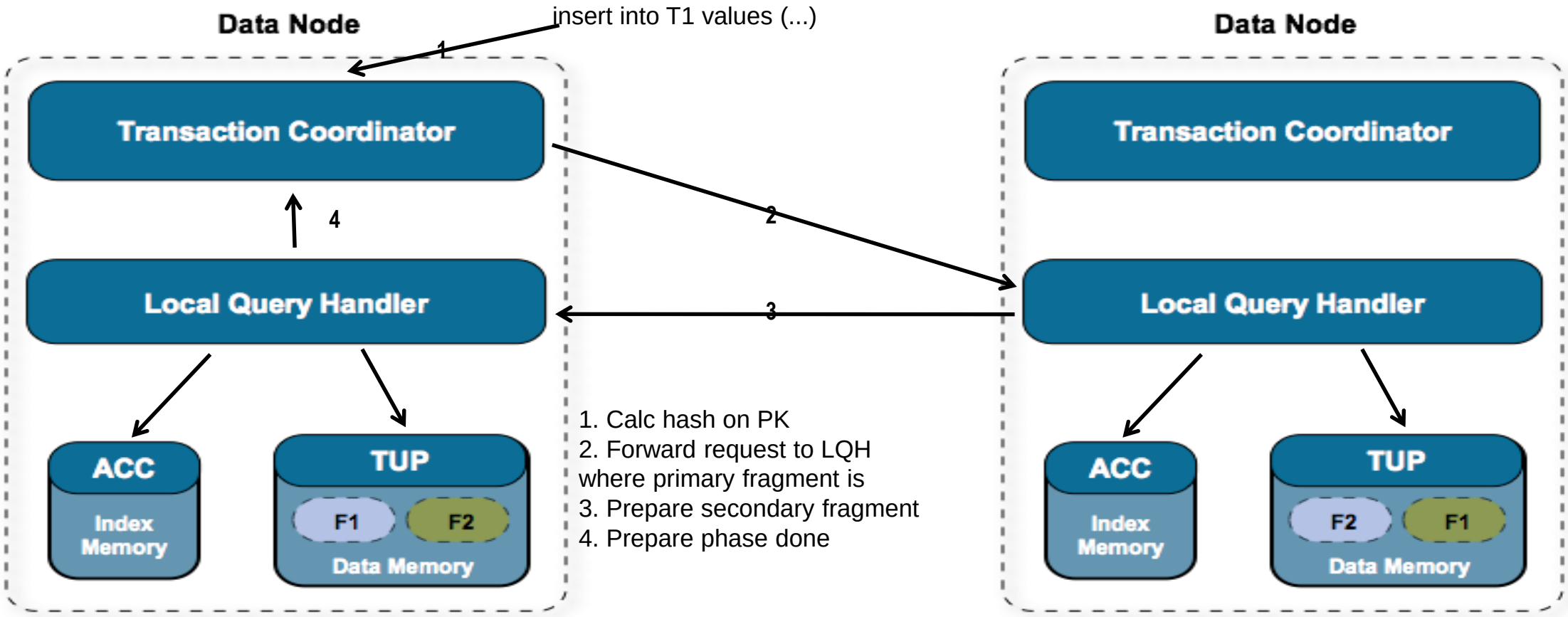
Internal Replication “2-Phase Commit”

Prepare Phase



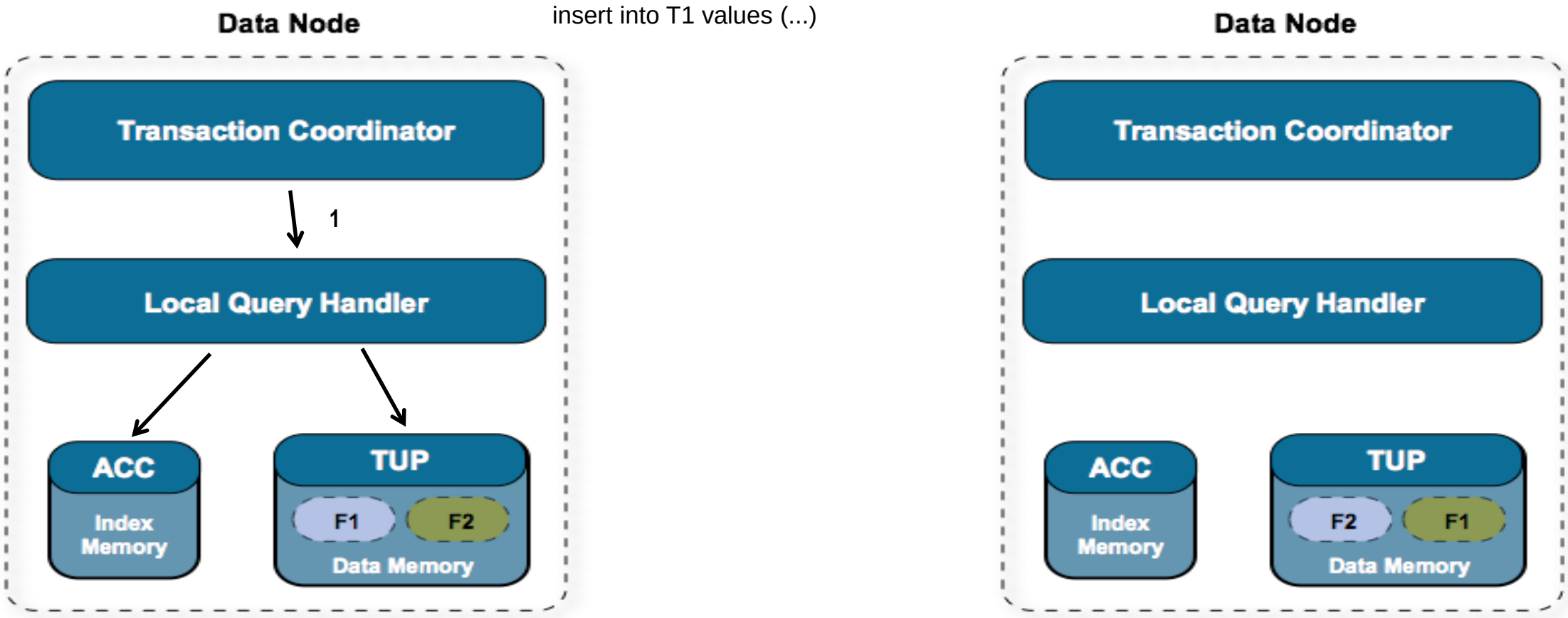
Internal Replication “2-Phase Commit”

Prepare Phase



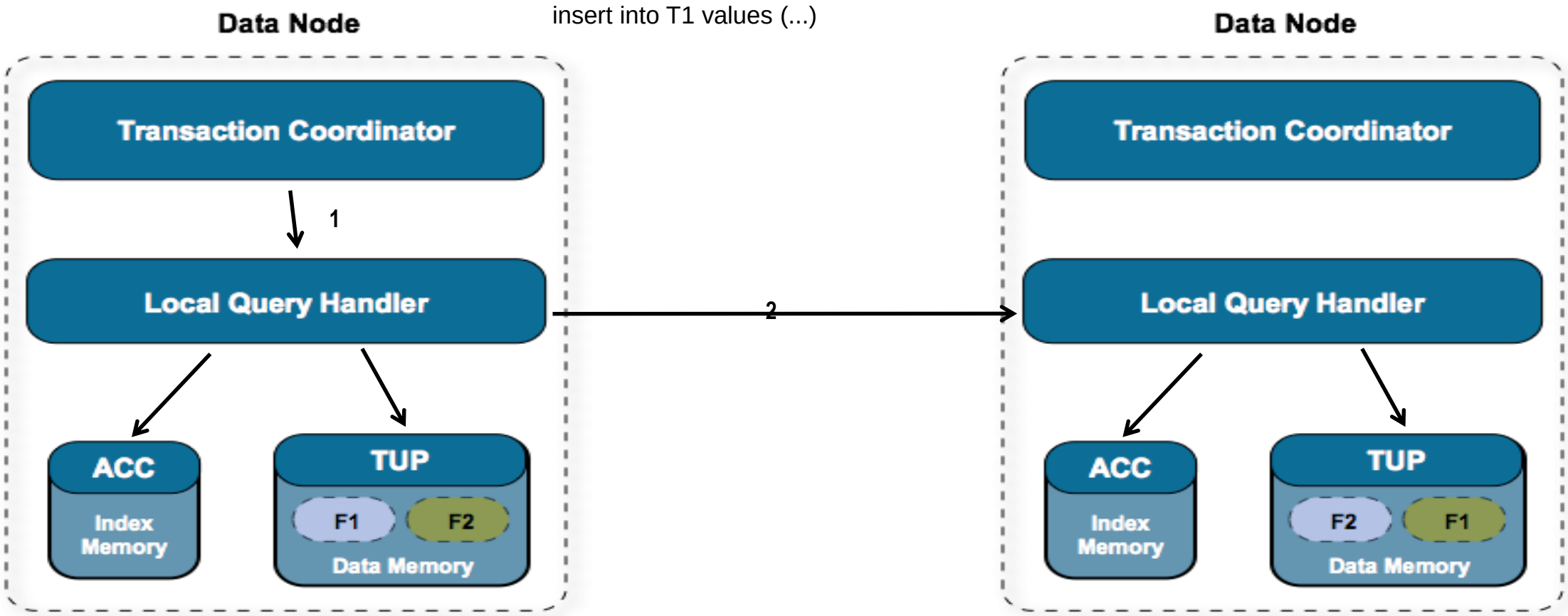
Internal Replication “2-Phase Commit”

Commit Phase



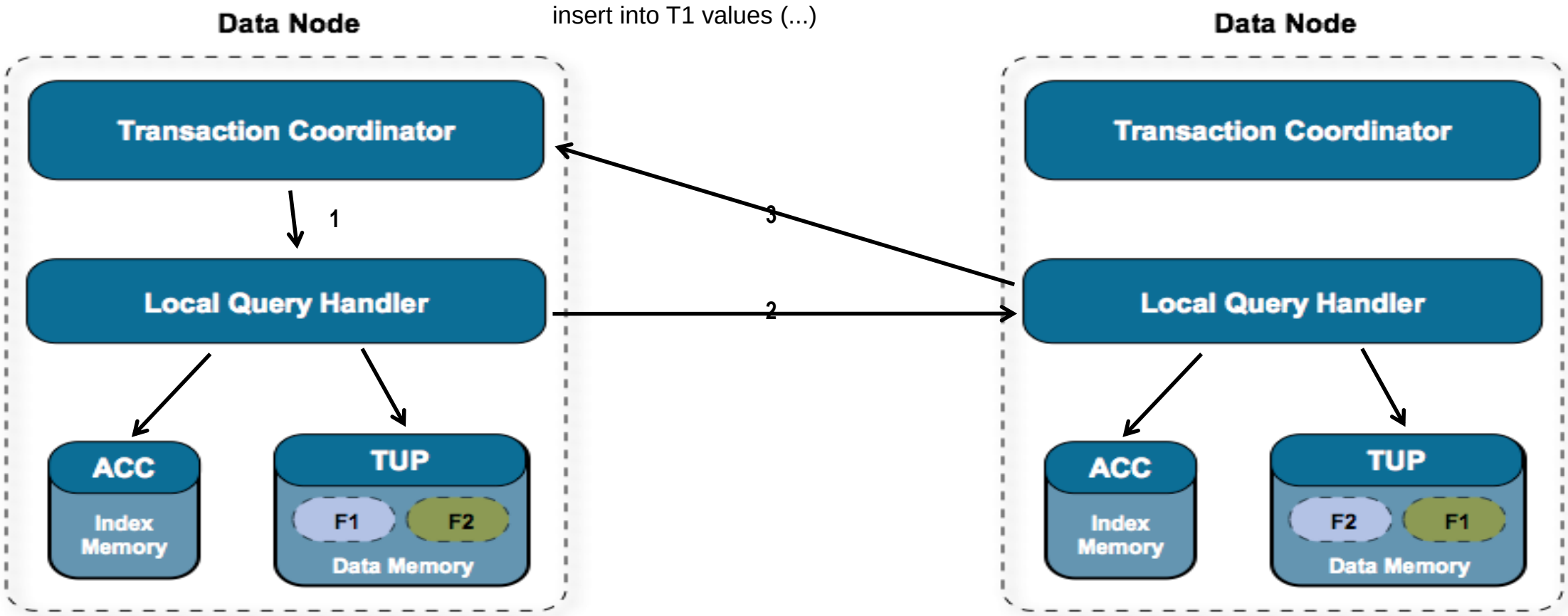
Internal Replication “2-Phase Commit”

Commit Phase



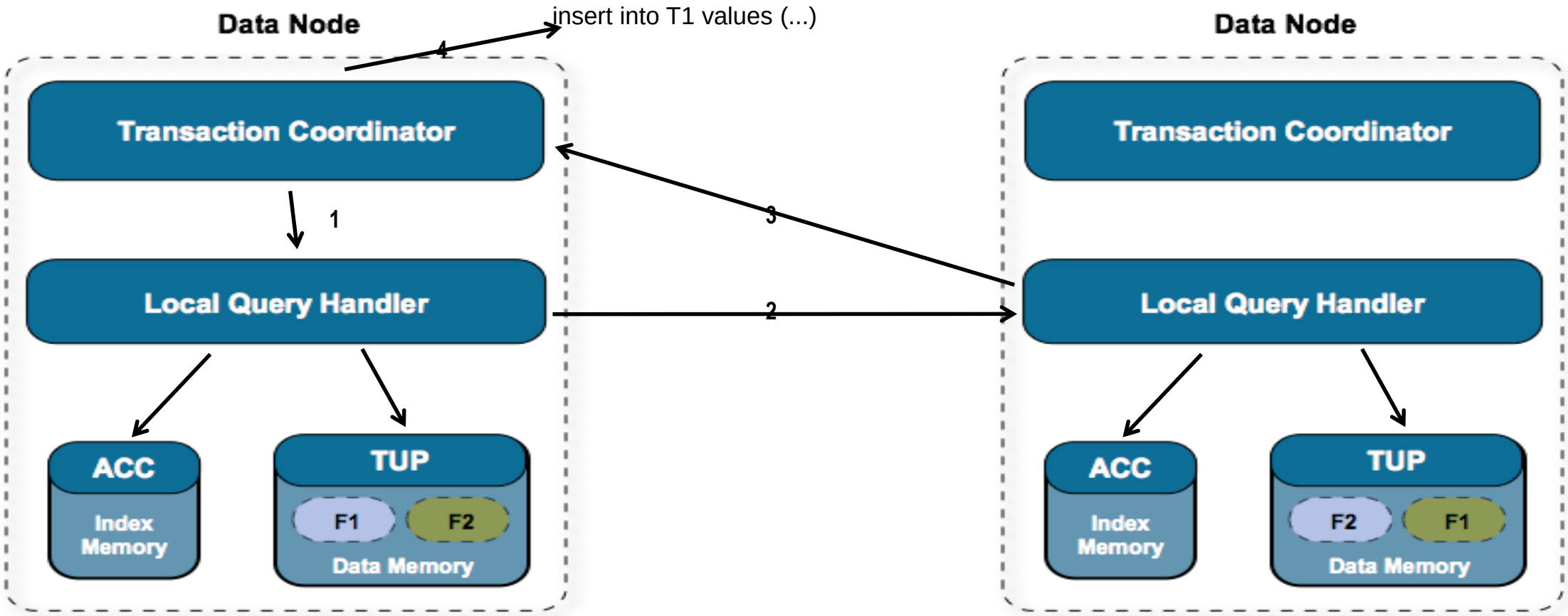
Internal Replication “2-Phase Commit”

Commit Phase



Internal Replication “2-Phase Commit”

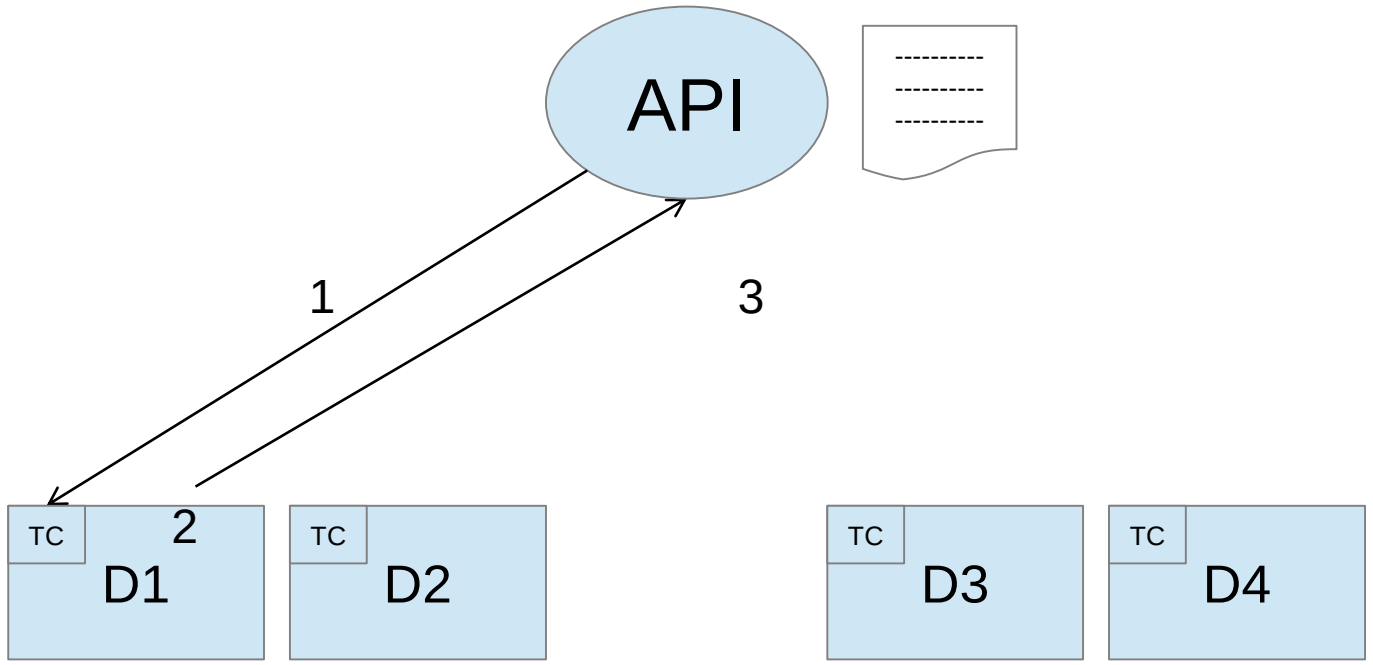
Commit Phase



Accessing data

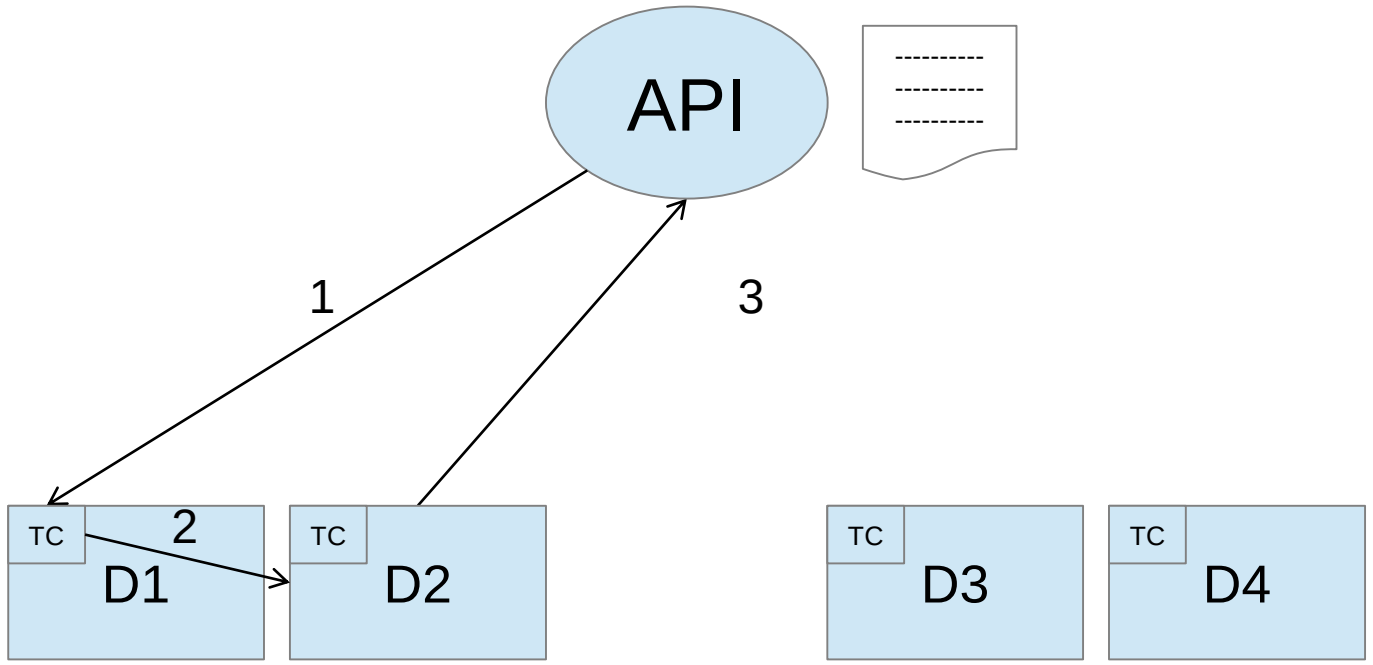
- Four operation types, each accessing a single table or index:
  - **Primary key operation**. Hash key to determine node and 'bucket' in node. O(1) in rows and nodes. Batching gives intra-query parallelism.
  - **Unique key operation**. Two primary key operations back to back. O(1) in rows and nodes
  - **Ordered index scan operation**. In-memory tree traversal on one or all table fragments. Fragments can be scanned in parallel. O(log N) in rows, O(n) in nodes, unless pruned.
  - **Table scan operation**. In memory hash/page traversal on all table fragments. Fragments can be scanned in parallel. O(n) in rows, O(n) in nodes.

Accessing data: PK key lookup



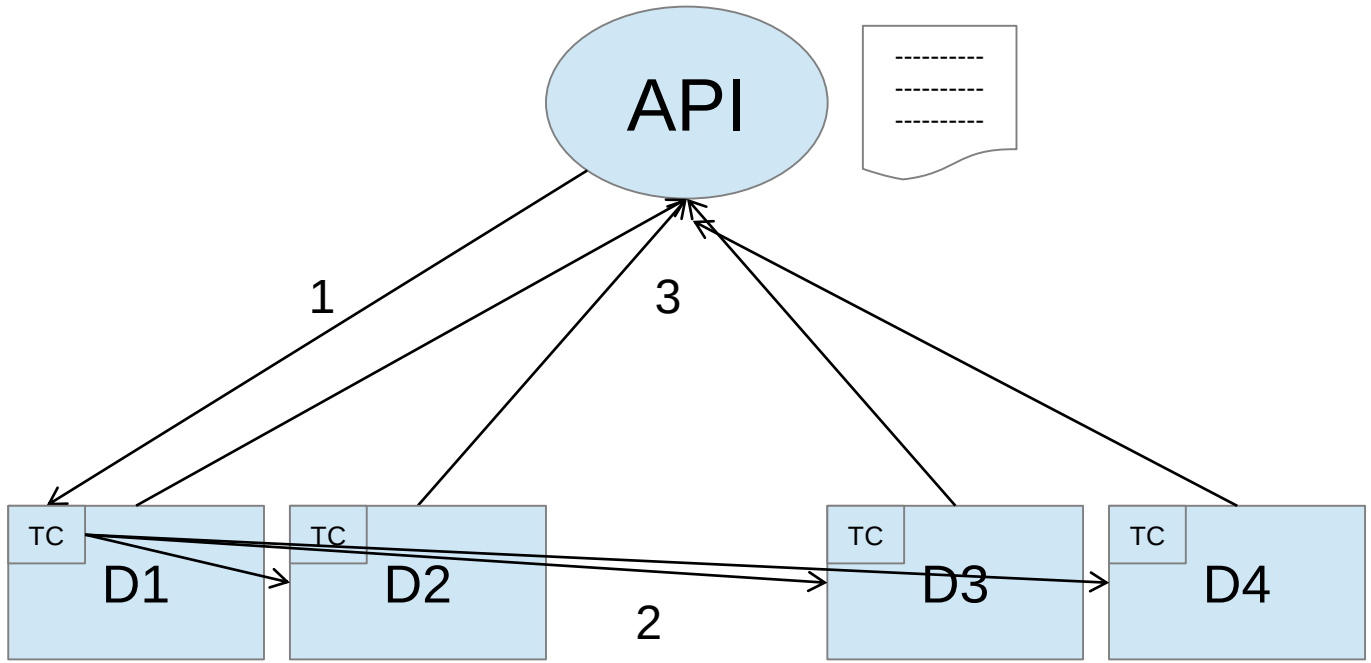
- First Statement decides TC
  - You keep this TC for the whole transaction!
- Keep transactions short!

Accessing data: Unique key lookup



- Secondary keys are implemented as “hidden” tables.
- These tables have the secondary key as PK and base table PK as value.
- Secondary key data may reside on same node or not.

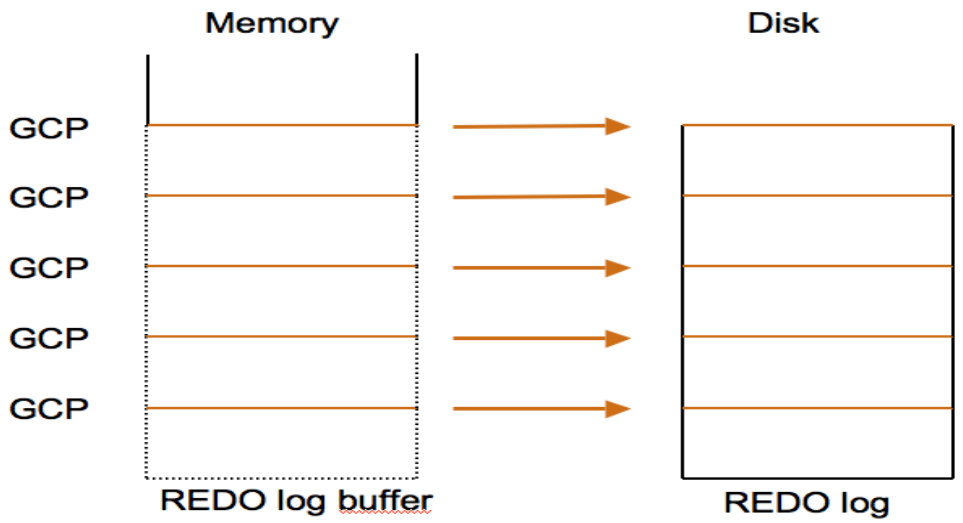
Accessing data: Table scan



- TC is chosen using Round Robin
- Data nodes send data directly to API Node
- Flow:
  - Pick one TC
  - Send request to all LDM
  - Send back data to API node

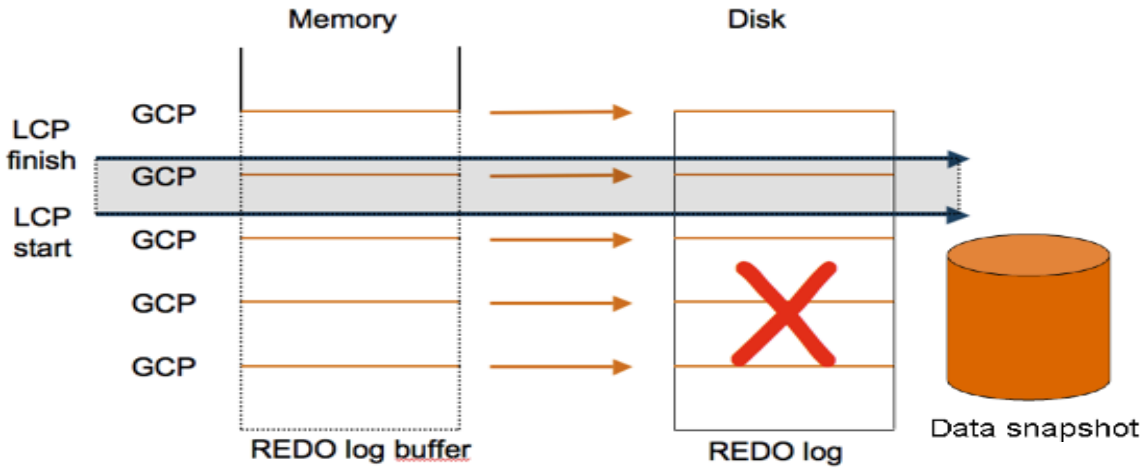
Checkpoints and Redo Logging
Global

- Global Checkpoint Protocol/Group Commit - GCP
  - Transactions (+increase GCI) synchronized cross all data nodes.
  - Persists transactions that have been recorded in the REDO log buffer to disk.
  - Frequency controlled by TimebetweenGlobalCheckpoints setting
    - Default is 2000ms
  - Size of the Redo log is defined by: (NumOfFragmentLogFiles x FragmentLogFileSize) x NoOfFragmentLogParts



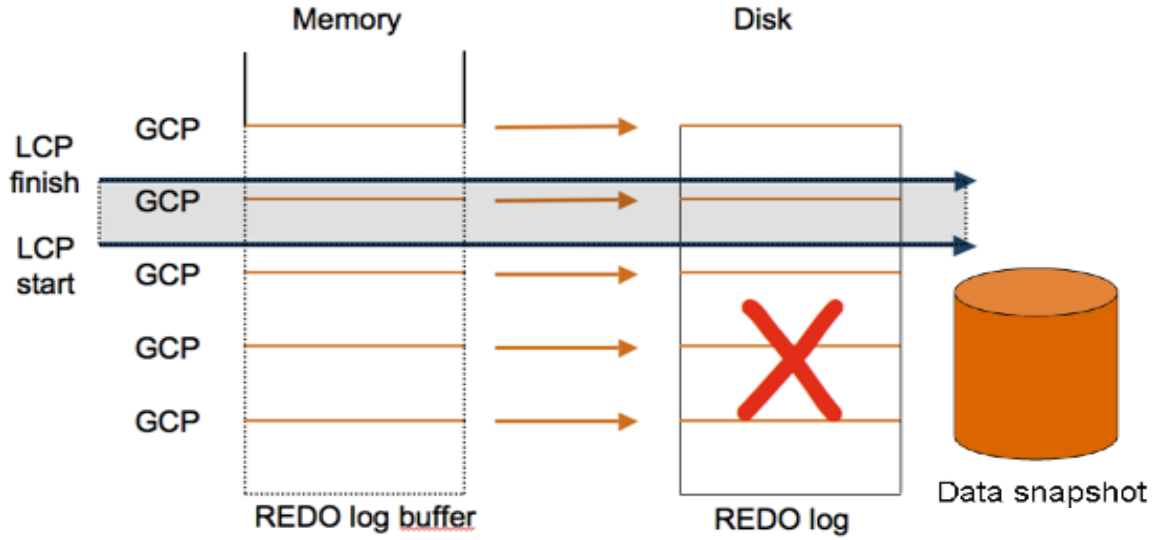
Checkpoints and Logging
Local

- Local Checkpoint Protocol - LCP
  - Persist data in memory to disk. After LCP the REDO log can be re-used.
  - GCP/GCI is important part of LCP
  - Frequency controlled by TimebetweenLocalCheckpoints setting
    - Specifies the amount of data that can change before flushing to disk
    - Not a time! Base-2 logarithm of the number of 4-byte words
    - Ex: Default value of 20 means 4*2^20 = 4MB of data changes, value of 21 = 8MB



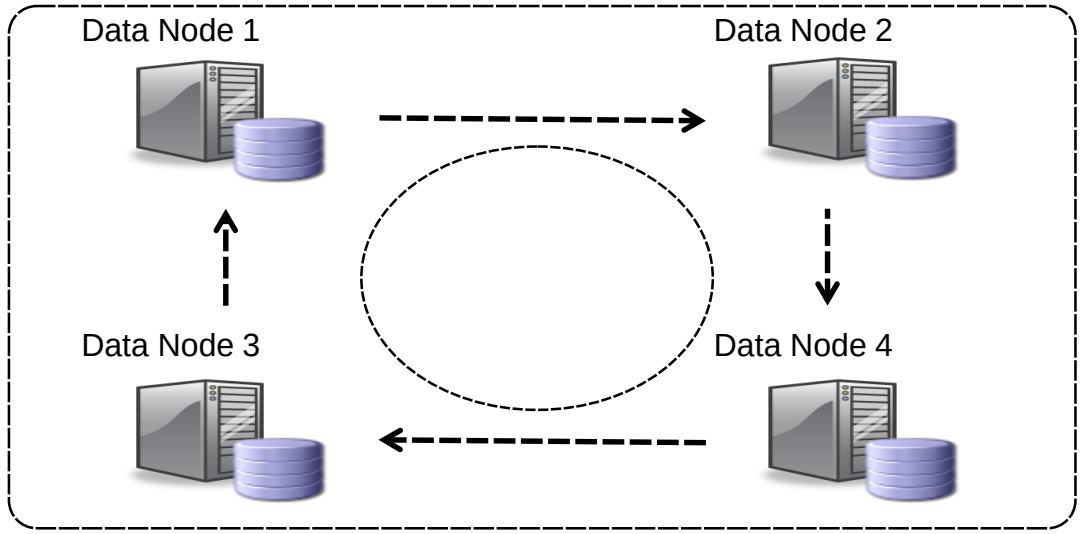
Checkpoints and Logging
Local & Redo

- LCP and REDO log are used to recover cluster to a consistent state.
  - System failure or restart of cluster
  - 1st Data Nodes are restored using the latest LCP
  - 2nd the REDO logs are applied from GCI recorded in LCP



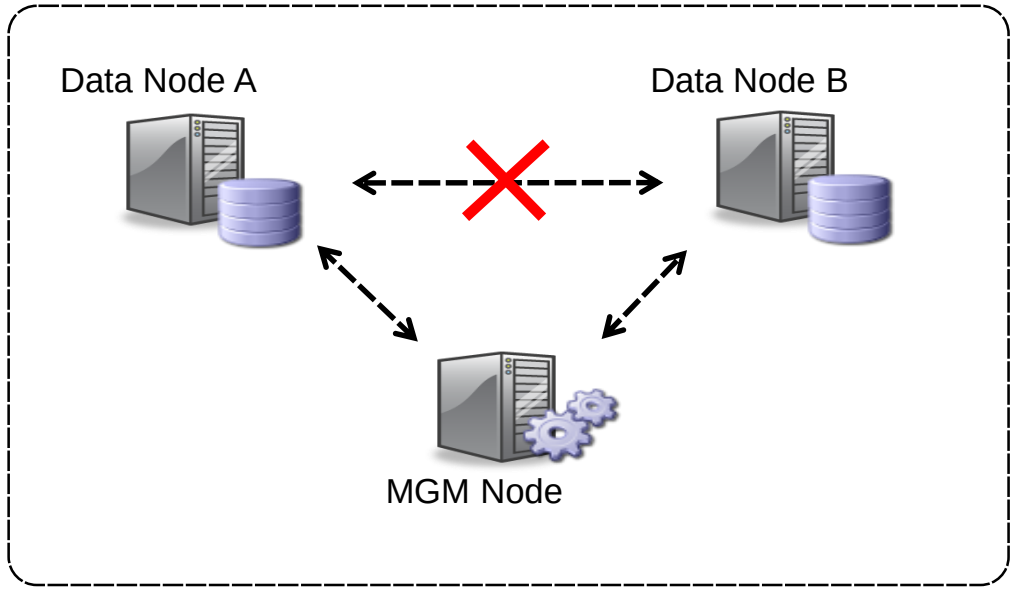
Failure Detection

- Node Failure
  - Heartbeat
    - Each Data Node is responsible for performing periodic heartbeat checks of other nodes
  - Requests/Response
  - After 3 failures, node is considered dead
- Failed heartbeat/response
  - The Node detecting the failed Node reports the failure to the rest of the cluster
- Hearbeat order can be set in configuration.



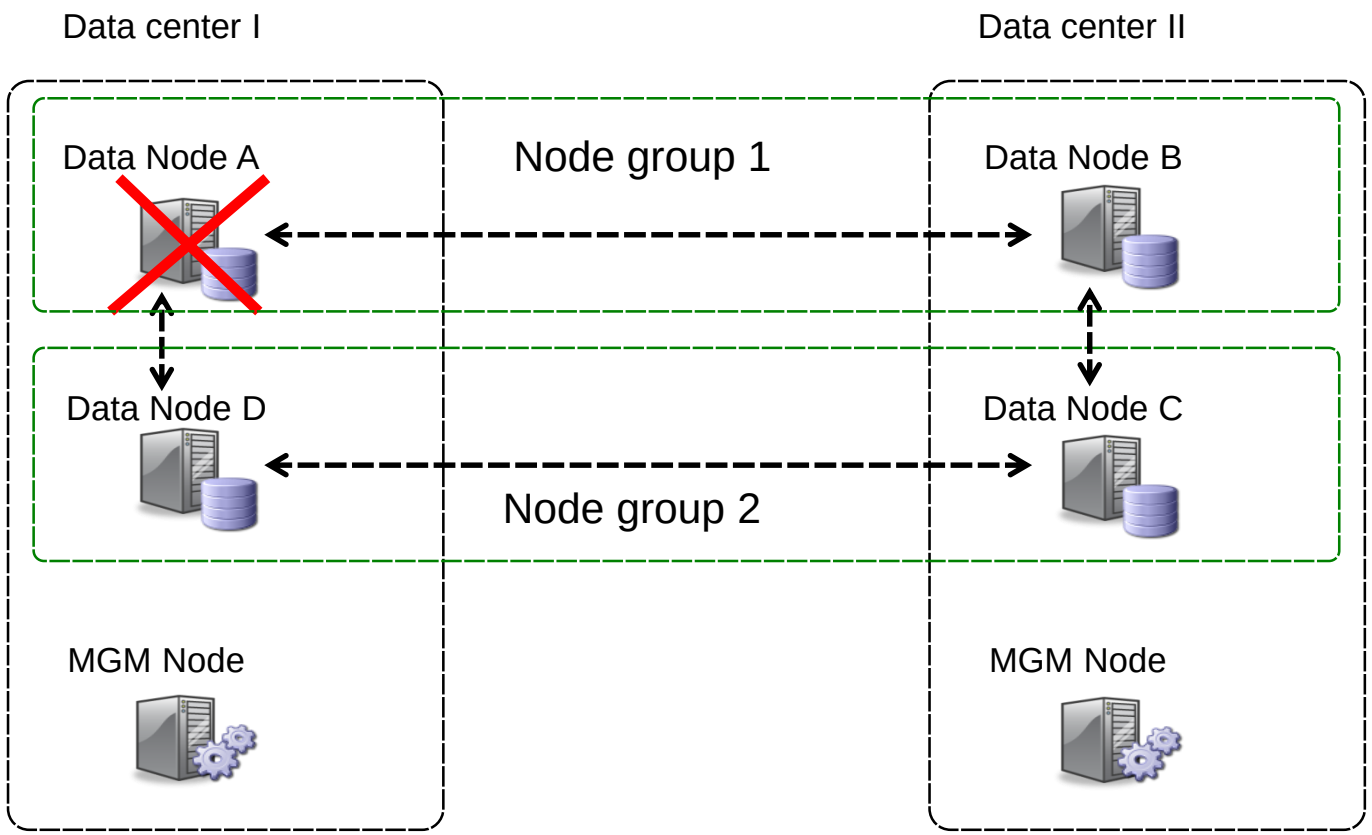
Date Nodes are organized in a logical circle
Heartbeat messages are sent to the next Data Node in the circle

Arbitration I



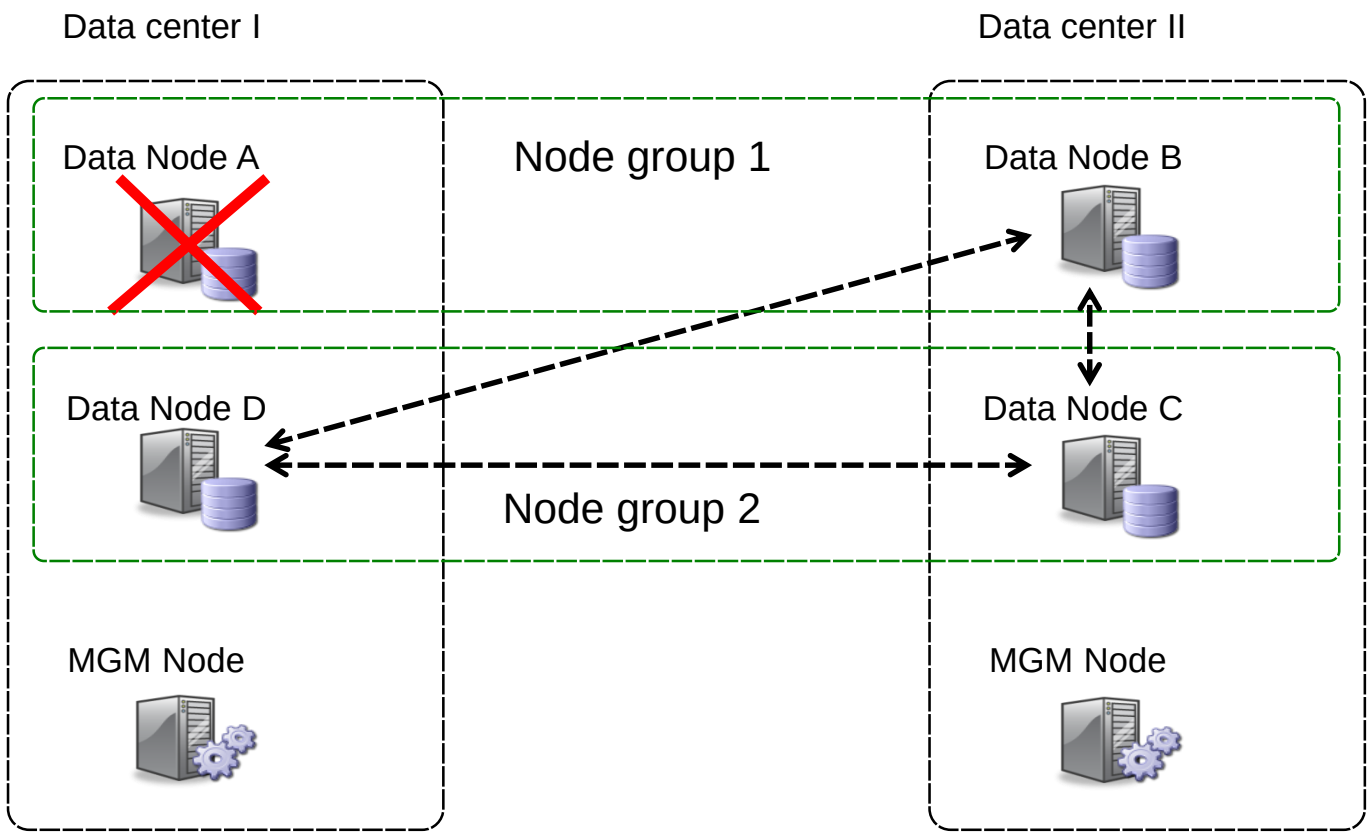
- What will happen:
  - NoOfReplicas==2?
  - NoOfReplicas==1?

Arbitration II



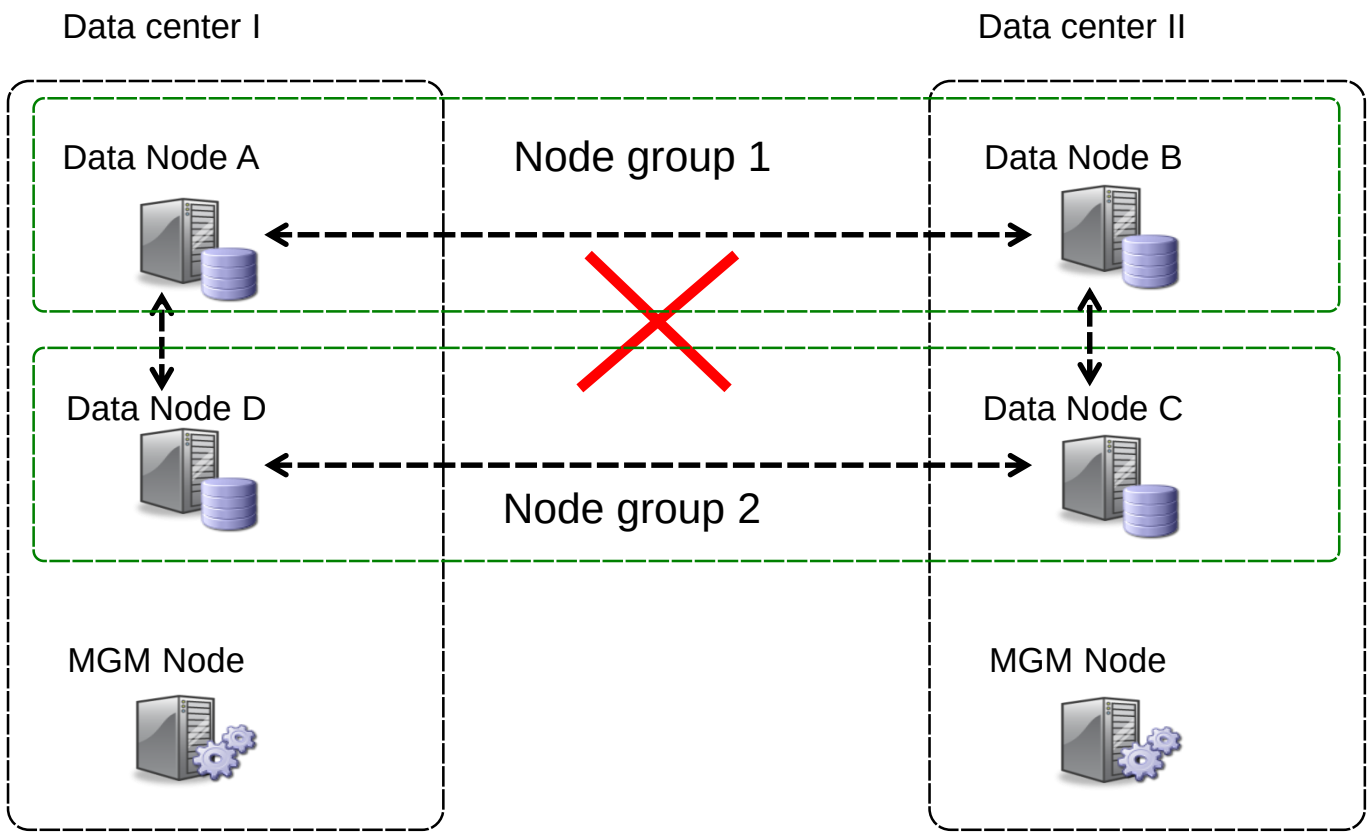
- What will happen:
  - Which side will survive?
  - And why?

Arbitration II



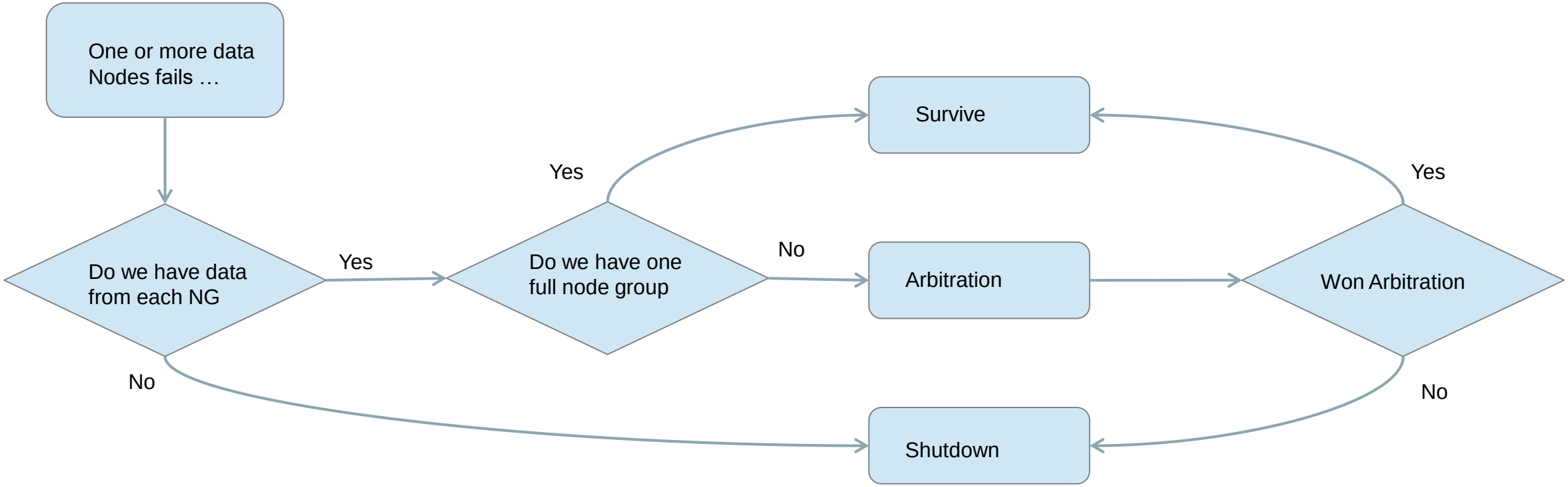
- What will happen:
  - New cluster with 3 nodes will continue!

Arbitration III

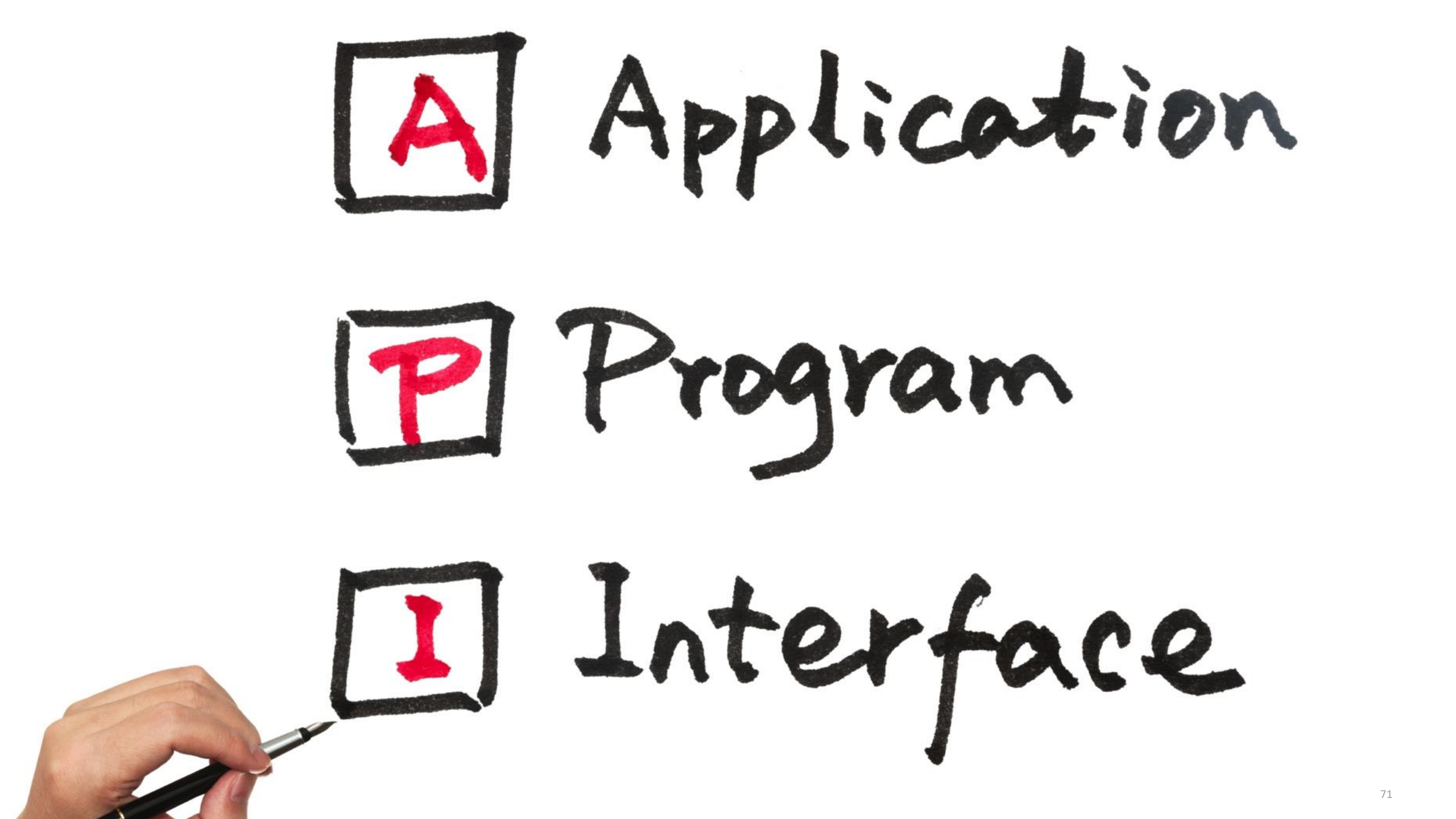


- What will happen:
  - Which side will survive?
  - And why?

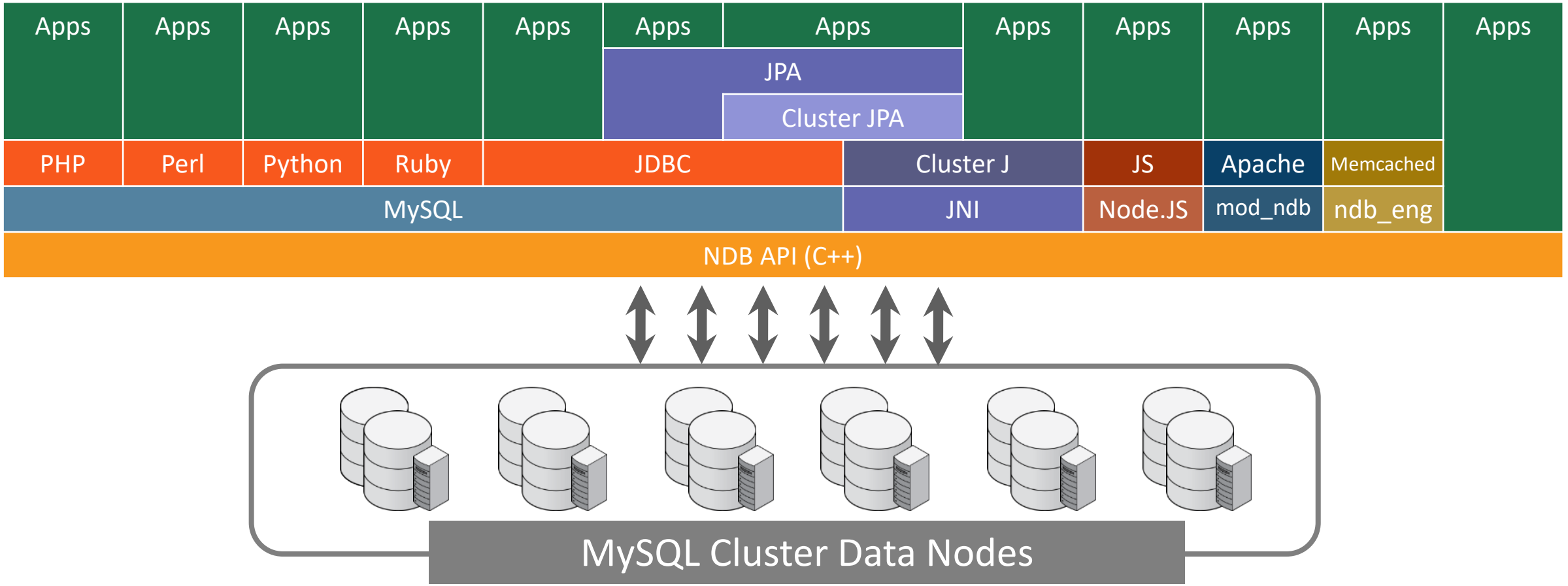
Arbitration flow chart



1. Check whether a data node from each node group is present. If that is not the case, the data nodes will have to shutdown.
2. Are all data nodes from one of the node groups present? If so it is guaranteed that this fragment is the only one that can survive. If no, continue to 3.
3. Contact the arbitrator.
4. If arbitration was won, continue. Otherwise shutdown.


A Application
P Program
I Interface

NoSQL Access to MySQL Cluster data



Choosing the right application API

SQL

- Industry standard
- Joins & Complex queries
- Relational model

Memcached

- Simple to use API
- Key/value
- Drivers for many languages

Mod-ndb

- REST
- Html
- Plugin for Apache

ClusterJ

- Simple to Use Java API
- Web & telco
- Object Relational Mapping
- Native & fast access to data

ClusterJPA

- OpenJPA plugin
- Standards defined ORM
- Cross table Joins

JavaScript/Node.js

- Native JavaScript: client to DB
- Blazing fast asynchronous throughput

Program Agenda

1. Introduction
2. Getting started: configuration/install/start/stop
3. Administration: backup/upgrade/logs/conf/sizing
4. Monitoring, surveillance and problem solving
5. Best practices, architectures and case studies
6. NDB 7.6 and what's new in NDB 8.0
7. Geo-replication

MySQL Cluster Installation options!

- Community / GPL
  - http://dev.mysql.com/downloads/cluster
- Carrier Grade Edition
  - http://edelivery.oracle.com
  - http://support.oracle.com
- Tarball
  - Manual “hands-on” install.
- RPM, DEB, PKG, DMG, MSI
  - O.S. specific install packages.
  - Specific paths, packages (server, client, etc.)
- Source code
  - Compile your own version.
    - `cmake .`
    - `make`
    - `make install`

MySQL Cluster Configuration

- All nodes are defined in config.ini
  - Divided into sections using “[…]”, sections for;
    - data nodes ndb{mt}d
    - Management nodes [ndb_mgmd]
    - MySQL servers [mysqld]
    - API nodes [ndbapi]
    - If keyword default is added to section values are applied to all
  - Management node is started with a config.ini. Should be same for all management nodes in one cluster.
- MySQL API nodes are configured both in config.ini and have their dedicated configuration file “my.cnf”

Configuration “config.ini”
Typical parameters

- `IndexMemory` and `dataMemory`.
- Set `MaxNoOfExecutionThreads` <= #cores
  - Otherwise contention will occur → unexpected behaviour.
- `RedoBuffer=32-64M`
  - If you need to set much it higher → your disks are probably too slow
- `FragmentLogFileSize=256M`
- `NoOfFragmentLogFiles`= 6 x DataMemory (in MB) / (4x 256MB)
  - Most common issue – customers never configure large enough redo log
- Data and index memory to cope with size of database/indexes
- `LockPagesInMainMemory=1`
- `MaxNoOf*` -parameters

Configuration
Disk-based tables

- Use Disk Data tables for
  - Simple accesses (read/write on PK)
  - Same for InnoDB – you can easily get IO BOUND (iostat)
- Set `DiskPageBufferMemory=3072M`
  - is a good start if you rely a lot on disk data – like the Innodb_Buffer_Pool, but set it as high as you can!
- Increased chance that a page will be cached
  - `SharedGlobalMemory=384M-1024M`
- `UNDO_BUFFER=64M` to `128M` (if you write a lot)
  - You cannot change this BUFFER later!

MySQL Cluster Starting / Stopping
config.ini

```
[ndb_mgmd default]
ArbitrationRank                 =1
DataDir                         =/opt/mysql/735/mgmd_data

[ndb_mgmd]
hostname                        =khollman-es
NodeId                          =1

[ndbd default]
noofreplicas                    =2
DataDir                         =/opt/mysql/735/ndbd_data
DataMemory                      =20M
IndexMemory                     =10M
DiskPageBufferMemory            =4M
StringMemory                    =5
MaxNoOfConcurrentOperations     =2K
MaxNoOfConcurrentTransactions   =2K
SharedGlobalMemory              =500K
LongMessageBuffer               =512K
MaxParallelScansPerFragment     =16
MaxNoOfAttributes               =1000
MaxNoOfTables                   =20
MaxNoOfOrderedIndexes           =20
ODirect                         =TRUE
HeartbeatIntervalDbDb           =500
HeartbeatIntervalDbApi          =500
StopOnError                     =1
TransactionInactiveTimeout      =500
TransactionDeadlockDetectionTimeout = 1200
LockPagesInMainMemory           =2

[ndbd]
hostname                        =khollman-es
datadir                         =/opt/mysql/735/ndbd_data
nodeid                          =3

[mysqld default]

[mysqld]
NodeId                          =10

[mysqld]
NodeId                          =11

[NDBAPI]
NodeId                          =12

[NDBAPI]
NodeId                          =13
```

MySQL Cluster Starting / Stopping

my.cnf

```
[client]
socket                          =/tmp/mysql_7351.sock

[mysql]
prompt                          =\R:\m \d>\_
no-beep

[mysqld]
ndbcluster
datadir                         =/opt/mysql/735/data
ndb-connectstring               =khollman-es:1186
user                            =mysql
port                            =7351
socket                          =/tmp/mysql_7351.sock
general-log                     =1
log-output                      =FILE
log-error                       =khollman-es_7351.err
slow-query-log                  =1
max_connections                 =20
innodb_log_buffer_size          =8M
innodb_buffer_pool_size         =64M
innodb_log_file_size            =16M
innodb_flush_log_at_trx_commit  =2
innodb_file_per_table           =1
innodb_data_home_dir           =/opt/mysql/735/data
innodb_data_file_path
  =ibdata1:50M;ibdata2:50M:autoextend

[mysql_cluster]
ndb-connectstring               =khollman-es:1186
```

MySQL Cluster Starting / Stopping

- First time & when configuration changes are needed: **`--INITIAL`**
  - When the config.ini changes, no need to do a complete shutdown. Restart ndb_mgmd with `--INITIAL` to clean the cached config information. And then just restart the data nodes (without --initial)

- Starting

```
# ndb_mgmd -f config.ini --config-dir=/usr/local/mysql-cluster/conf --INITIAL
# ndbd --INITIAL  | ndbmtd -c localhost:1186 --INITIAL  |  ndbd –n (nostart)
# scripts/mysql_install_db --defaults-file=my.cnf --user=mysql
# mysqld_safe --defaults-file=my.cnf --user=mysql
```

MySQL Cluster Starting / Stopping

- Starting
  - `ndb_mgmd -f config.ini --config-dir=/usr/local/mysql-cluster/conf`
  - `ndbd | ndbmtd -c localhost:1186`
  - `mysqld_safe --defaults-file=my.cnf`
- Stopping
  - `ndb_mgm -e shutdown`
  - `ndb_mgm -e` [mgmt node & datanode] `1 | 3 | 4 stop`
  - `ndb_mgm -e` [mgmt node & datanode] `1 | 3 | 4 restart`
  - `mysqladmin --defaults-file=my.cnf -uroot shutdown`

MySQL Cluster Starting / Stopping
nostart

```
ndbmtd --ndb-nodeid=3 -n

ndbmtd --ndb-nodeid=4 -n

ndb_mgm -e show
Cluster Configuration
---------------------
[ndbd(NDB)]   2 node(s)
id=3   @127.0.0.1  (mysql-5.6.17 ndb-7.3.5, not started)
id=4   @127.0.0.1  (mysql-5.6.17 ndb-7.3.5, not started)

ndb_mgm -e "all start"

ndb_mgm -e show
Cluster Configuration
---------------------
[ndbd(NDB)]   2 node(s)
id=3   @127.0.0.1  (mysql-5.6.17 ndb-7.3.5, starting, Nodegroup: 0, *)
id=4   @127.0.0.1  (mysql-5.6.17 ndb-7.3.5, starting, Nodegroup: 0)
```

MCM Hands On Lab
1 Installing MySQL Cluster Manager

- Lab-1
  - https://github.com/wwwted/ndb-cluster-workshop
- Environment for HOL

```
/home/<user>/MCM_LAB/
                cluster-758/
                mcm1.4.4/
                mcm_data/
                mcmd.ini
```

MCM Hands On Lab
2 Configure and start MySQL Cluster

- Lab-2
  - https://github.com/wwwted/ndb-cluster-workshop

```
mcm> show status -r mycluster;
+--------+----------+-----------+---------+-----------+------------+
| NodeId | Process  | Host      | Status  | Nodegroup | Package    |
+--------+----------+-----------+---------+-----------+------------+
| 49     | ndb_mgmd | 127.0.0.1 | running |           | cluster735 |
| 1      | ndbmtd   | 127.0.0.1 | running | 0         | cluster735 |
| 2      | ndbmtd   | 127.0.0.1 | running | 0         | cluster735 |
| 50     | mysqld   | 127.0.0.1 | running |           | cluster735 |
| 51     | mysqld   | 127.0.0.1 | running |           | cluster735 |
| 52     | ndbapi   | *127.0.0.1 | added  |           |            |
| 53     | ndbapi   | *127.0.0.1 | added  |           |            |
| 54     | ndbapi   | *127.0.0.1 | added  |           |            |
| 55     | ndbapi   | *127.0.0.1 | added  |           |            |
+--------+----------+-----------+---------+-----------+------------+
9 rows in set (0,07 sec)
```

Program Agenda

1. Introduction
2. Getting started: configuration/install/start/stop
3. Administration: backup/upgrade/logs/conf/sizing
4. Monitoring, surveillance and problem solving
5. Best practices, architectures and case studies
6. NDB 7.6 and what's new in NDB 8.0
7. Geo-replication

Backup & Restore

Backup

- Backup of NDB tables
  - Online – can have ongoing transactions
  - Consistent – only committed data and changes are backed up
- ndb_mgm -e “START BACKUP”
  - Copy backup files from data nodes to safe location
  - Non-NDB tables must be backed up separately
  - MySQL system tables are stored only in MYISAM.

- You want to backup data in MySQL
  - Users in mysql database
  - Triggers, routines, events ...
- MySQL Users:
  - `mysqlpump --exclude-databases=% --users > users.sql`
- MySQL DB logic:
  - `mysqldump --no-data --no-create-info --skip-opt --routines --events --triggers --databases db1 db2 > dblogic.sql`
- Copy `my.cnf` & `config.ini` files

Backup

- `ndb_mgm> START BACKUP`
- Generates files holding the following information on each node, under the BackupDataDir, or datanode datadir BACKUP/BACKUP-backup-id :
  - **Metadata**: saved in controlfile
    - Holds table definitions and control information
    - `BACKUP-backup-id.nodeid.ctl`: eg. `BACKUP-1.6.ctl`
  - **Data**: saved in a file ".Data"
    - Holds per-fragment records
    - `BACKUP-1-0.6.Data`
  - **Transaction log**: saved file ".log"
    - Stores transactions happening during backup, used to recover to a consistent state
    - `BACKUP-1.6.log`

Backup
What happens whilst backing up?

- Each data node scans its tables and adds them to the backup data buffer
  - `BackupDataBufferSize` (default 16Mb). 0 – 4Gb, min +188Kb > `BackupWriteSize` (min = 190Kb).
- Backup flushes data to disk determined by:
  - `BackupWriteSize` (default 256Kb) 2Kb – 4Gb.
- Maximum size of disk writes controlled by:
  - `BackupMaxWriteSize` (default 1Mb) 2Kb – 4Gb.
- If the buffer fills up, the scan pauses until enough data has been written to disk before continuing.

Backup
And ongoing transactions?

- Backup process logs all data modifications during backup in:
  - `BackupLogBufferSize` (default 16Mb)
- Log buffer flushes changes to disk same way as it does with the Data Buffer
  - `BackupWriteSize` & `BackupMaxWriteSize`.

- Depending on system activity, buffer to disk flushes might require moving BackupDataDir to separate disks than redo, disk data, etc.
- If the log buffer fills up during backup, the backup aborts.

Backup Config

- Status reporting:
  - `BackupReportFrequency` (default 0 max 4Gb).
  - Number of seconds between status updates in management console or cluster log.

Backup
management console

- For large backups, to regain control of management console:
  - `START BACKUP NOWAIT`
    - Immediate return
  - `START BACKUP WAIT STARTED`
    - Once it starts, return control.
  - `START BACKUP` = `START BACKUP WAIT COMPLETED`.

Backup

- By default the backup represents the state of the database at the end of the backup process.
- You can take a backup to reflect the state at the beginning:
  - `ndb_mgm> START BACKUP SNAPSHOTSTART`
  - If the dataset is large and you have high rate of changes, it may take some time to undo all changes that happended during the backup.

Backup
Monitoring

- Whilst a backup is running we can check the status with:

```
ndb_mgm> 3 REPORT backupstatus
ndb_mgm> ALL REPORT backupstatus
```

```
Node 3: Local backup status: backup 3 started from node 1
 #Records: 0 #LogRecords: 0
 Data: 0 bytes Log: 0 bytes
Node 4: Local backup status: backup 3 started from node 1
 #Records: 0 #LogRecords: 0
 Data: 0 bytes Log: 0 bytes
Node 5: Local backup status: backup 3 started from node 1
 #Records: 0 #LogRecords: 0
 Data: 0 bytes Log: 0 bytes
Node 6: Local backup status: backup 3 started from node 1
 #Records: 0 #LogRecords: 0
 Data: 0 bytes Log: 0 bytes
Node 5: Backup 3 started from node 1
Node 5: Backup 3 started from node 1 completed
 StartGCP: 174775 StopGCP: 174778
 #Records: 14148 #LogRecords: 0
 Data: 1018516 bytes Log: 0 bytes
```

Backup
Troubleshooting

- To abort the backup just run:

```
ndb_mgm> abort backup 3
```

- Typical errors:
  - Disk full: Change `BackupDataDir` to a new path.
    - Set `CompressedBackup=1` in the config.ini.
  - Disk slow: Choose a faster IO subsystem or increase `BackupDataBufferSize` & `BackupLogBufferSize`.
  - Backup aborted: increase `BackupLogBufferSize` to handle more ongoing transactions.

Backup
Restore

- Use the **`ndb_restore`** CLI.
  - Enter single user mode on API node to avoid conflicts with other API nodes.
    - During restore data should not be accessed as there are no consistency guarantees until restore is finalized.
  - Restore Metadata separately from Data:
    - Metadata restore to a new or initialized Cluster.
    - Restore data only for the same cluster.
  - Ndb_restore is executed once for each backed up data node.
    - Nodeid is specific in all statements.

Backup
Restore

- 2 ways to launch restore:

```
ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 2 -r –b 4
ndb_restore --backup-path=/backups/BACKUP/BACKUP-4 --nodeid=2 --restore_data --
  backupid=4
```

- So if there are 4 datanodes in the cluster:

```
ndb_mgm> ENTER SINGLE USER MODE 2
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 2 -r –b 4
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 3 -r –b 4
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 –n 4 -r –b 4
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 5 -r –b 4
ndb_mgm> EXIT SINGLE USER MODE
```

Backup
Restore cont.

- If it's a new cluster, i.e. not restoring on the same system, make sure the metadata is restored first:

```
ndb_mgm> ENTER SINGLE USER MODE 2
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 2 –b 4 --restore_meta
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 2 -r –b 4
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 3 -r –b 4
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 –n 4 -r –b 4
# ndb_restore --backup_path=/backups/BACKUP/BACKUP-4 -n 5 -r –b 4
ndb_mgm> EXIT SINGLE USER MODE
```

Backup
Being selective

- Restore only specific databases:

```
--include-databases=db1,db2
```

- Restore all except specific databases:

```
--exclude-databases=db3
```

- Restore selected tables:

```
--include-tables=db3.tbl1
```

- Restore all but selected tables:

```
--exclude-tables=db1.tbl2
```

Administration - Upgrades

Hot Database Upgrades
Alternatives

- In order to keep it Hot, a “Rolling Restart” is the best and only alternative:
  1. Stop all Cluster management nodes, replace ndb_mgmd binary with new version & restart them.
  2. Stop, replace and restart each data node, each in turn, one-by-one.
  3. Stop, replace and restart each sql node, in turn.

  - “Replace” being either redirecting to the new software install on the node (ln –s, or using direct path).
  - “software” means the new Cluster distribution package, in “`mysql-cluster-gpl-7.3.6-linux-glibc2.5-i686`” format.

Hot Database Upgrades
Alternatives cont.

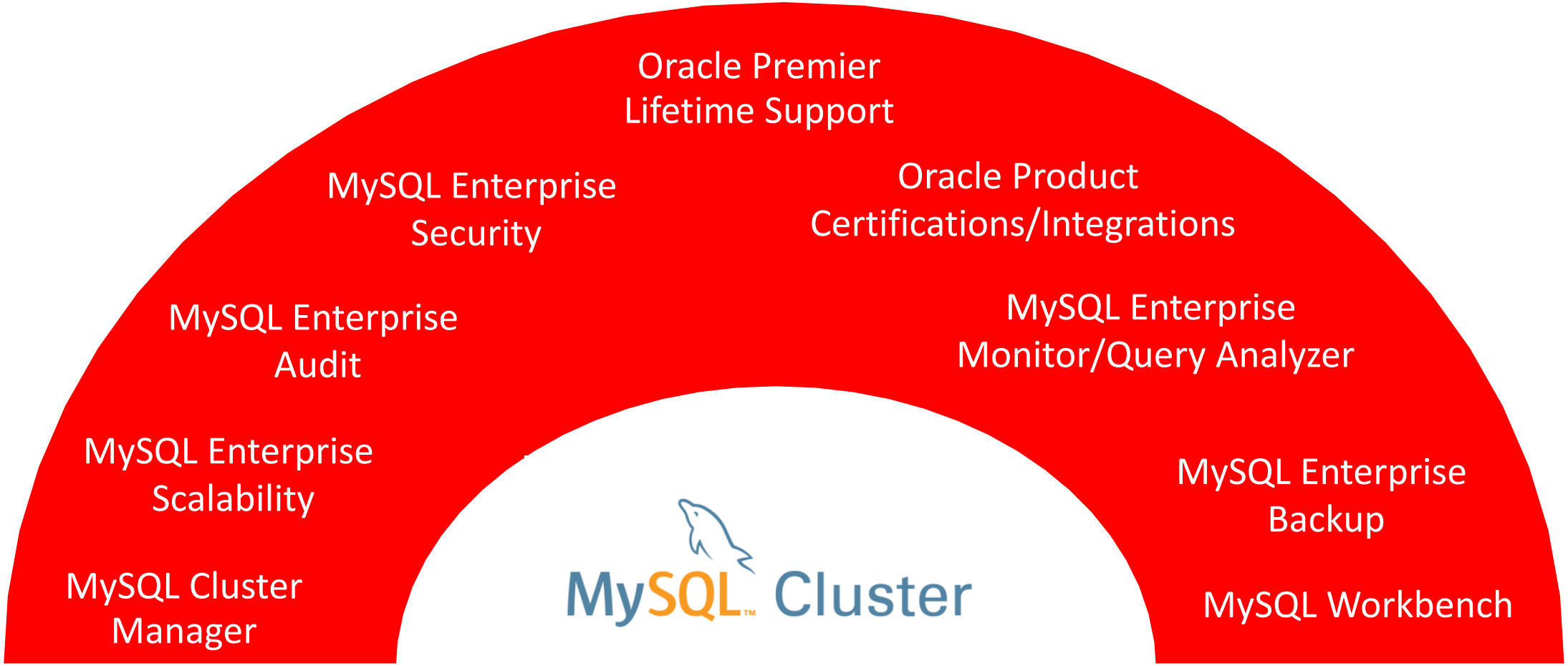
- Use MySQL Replication
  - Slave's can have a superior version compared to Master.

- If you want to use MySQL Cluster Manager:

```
mcm> add package --basedir=/opt/MCM_LAB/cluster-736 cluster736;
mcm> upgrade cluster --package=cluster736 mycluster;
```

MySQL Cluster CGE

Highest Levels of Performance, Security & Availability



How Does MySQL Cluster Manager Help?
Initiating upgrade from MySQL Cluster 7.0 to 7.3

MySQL Cluster Manager

Before MySQL Cluster Manager

- 1 x preliminary check of cluster state
- 8 x ssh commands per server
- 8 x per-process stop commands
- 4 x scp of configuration files (2 x mgmd & 2 x mysqld)
- 8 x per-process start commands
- 8 x checks for started and re-joined processes
- 8 x process completion verifications
- 1 x verify completion of the whole cluster.
- Excludes manual editing of each configuration file.

Total: 46 commands -
2.5 hours of attended operation

With MySQL Cluster Manager

upgrade cluster --package=7.3 mycluster;

Total: 1 Command -
Unattended Operation

ORACLE

Enhancing DevOps Agility, Reducing Downtime

MySQL Cluster Manager

Automated Management

- Start / Stop node or whole cluster
- On-Line Scaling
- On-Line Reconfiguration
- On-Line Upgrades
- On-Line Backup & Restore
- Import Running Cluster

Self-Healing

- Node monitoring
- Auto-recovery extended to SQL + mgmt nodes

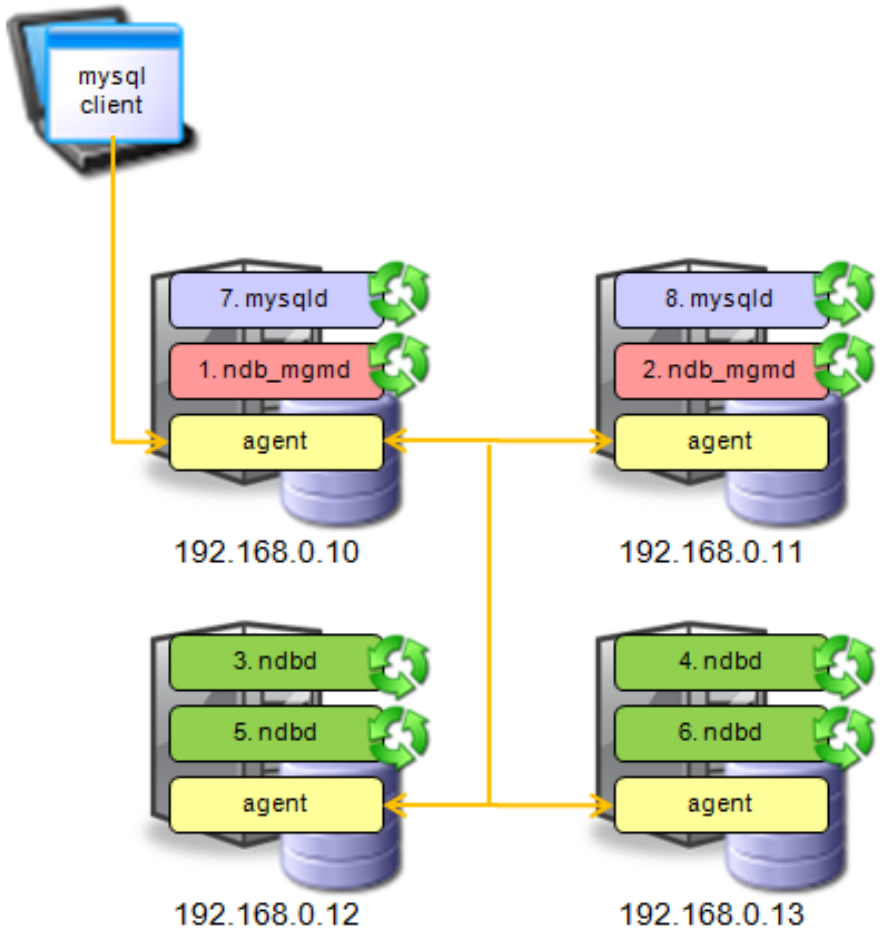
HA Operations

- Cluster-wide configuration consistency
- Persistent configurations
- HA Agents

ORACLE

MCM: Upgrade Cluster

MySQL Cluster Manager




```
mcm> upgrade cluster --package=cluster736
mycluster;
```

Administration - Logs

Log files

- Logs
  - Management
    - Datadir: `/opt/mysql/735/mgmd_data/` **`ndb_1_cluster.log`** / `ndb_1_out.log`
  - Datanodes
    - Datadir: `/opt/mysql/735/ndbd_data/` `ndb_3_out.log` / `ndb_3_out.err`
  - SQLNodes
    - Datadir `/opt/mysql/735/data/` `hostname.err`
- Errors
  - Cluster log file `ndb_1_cluster.log` best starting point.

Log file:
configuring

- To write the cluster log in 1 or more destinations use `LogDestination:`
  - `CONSOLE` writes to same console where ndb_mgmd was launched.
  - `FILE` (default) can be relative or absolute: `filename=cluster_log` or `filename=/var/log/mysql-cluster`
    - `maxsize`= number of bytes at which it rotates. Default 1000000.
    - `maxfiles`= max number of log files. Default 6.
    - `LogDestination=FILE:filename=/var/log/mysql-cluster,maxsize=500000,maxfiles=15`
  - `SYSLOG` Can be used in conjunction with available facilities:
    - `Sends the log to a syslog facility, possible values auth, authpriv, cron, daemon, ftp, kern, news, syslog, user, local0, local1 … etc`
- Sample: `CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd`

Troubleshooting
cluster log file

- By default on, as mentioned before. There are other options:

```
CLUSTERLOG ON [<severity>] ...          Enable Cluster logging
CLUSTERLOG OFF [<severity>] ...         Disable Cluster logging
CLUSTERLOG TOGGLE [<severity>] ...      Toggle severity filter on/off
CLUSTERLOG INFO                         Print cluster log information
```

- Changing this will only affect the management node you're connected to. Two different management nodes in the same cluster can have different settings.
- Use `clusterlog TOOGLE` to enable & disable different severities.

```
severity = ALERT | CRITICAL | ERROR | WARNING | INFO | DEBUG
```

(from most severe to least severe)

Troubleshooting
Event filtering

- Every event is in a Category, with a specific Severity and a Priority level.

```
category = STARTUP | SHUTDOWN | STATISTICS | CHECKPOINT | NODERESTART |
           CONNECTION | INFO | ERROR | CONGESTION | DEBUG | BACKUP |
           SCHEMA

level    = 0 - 15 (0 only prints the most imporant messages,
                   15 prints all messages)
```

- Each category has a threshold, events with priority <= threshold will be recorded.
  - ERROR default threshold = 15 (records all events by default)
  - All other categories have default threshold = 7 (only events with priority level <= 7 will be reported)

Troubleshooting
Adjusting thresholds

- For example, to remove all reporting for category STARTUP on nodeid 3:

```
ndb_mgm> 3 CLUSTERLOG STARTUP=0
```

- To include all events in the STARTUP category for all nodes:

```
ndb_mgm> ALL CLUSTERLOG STARTUP=15
```

- List of events, ordered by category & severity:

http://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-log-events.html

Logfiles:
ndb_1_cluster.log

```
2014-05-22 10:35:14 [MgmtSrvr] INFO     -- Got initial configuration from '/usr/local/mysql-cluster/conf/config.ini', will try to set it when all ndb_mgmd(s) started
2014-05-22 10:35:15 [MgmtSrvr] INFO     -- Id: 1, Command port: *:1186
2014-05-22 10:35:15 [MgmtSrvr] INFO     -- Node 1: Node 1 Connected
2014-05-22 10:35:15 [MgmtSrvr] INFO     -- MySQL Cluster Management Server mysql-5.6.17 ndb-7.3.5 started
2014-05-22 10:35:15 [MgmtSrvr] INFO     -- Node 1 connected
2014-05-22 10:35:15 [MgmtSrvr] INFO     -- Starting initial configuration change
2014-05-22 10:35:15 [MgmtSrvr] INFO     -- Configuration 1 commited
2014-05-22 10:35:15 [MgmtSrvr] INFO     -- Config change completed! New generation: 1
2014-05-22 10:35:58 [MgmtSrvr] INFO     -- Nodeid 3 allocated for NDB at 127.0.0.1
2014-05-22 10:35:59 [MgmtSrvr] INFO     -- Node 1: Node 3 Connected
2014-05-22 10:36:01 [MgmtSrvr] INFO     -- Nodeid 4 allocated for NDB at 127.0.0.1
2014-05-22 10:36:02 [MgmtSrvr] INFO     -- Node 1: Node 4 Connected
2014-05-22 10:36:02 [MgmtSrvr] INFO     -- Node 3: Initial start, waiting for 4 to connect,  nodes [ all: 3 and 4 connected: 3 no-wait:  ]
2014-05-22 10:36:02 [MgmtSrvr] INFO     -- Node 4: Node 3 Connected
2014-05-22 10:36:02 [MgmtSrvr] INFO     -- Node 3: Node 4 Connected
2014-05-22 10:36:02 [MgmtSrvr] INFO     -- Node 3: Initial start with nodes 3 and 4 [ missing:  no-wait:  ]
…
2014-10-20 10:48:49 [MgmtSrvr] INFO     -- Node 5: LQH: index 17 rebuild done
2014-10-20 10:48:49 [MgmtSrvr] INFO     -- Node 5: LQH: Rebuild ordered indexes complete
…
2014-10-20 10:48:49 [MgmtSrvr] WARNING  -- Failed to allocate nodeid for API at 127.0.0.1. Returned error: 'No free node id found for mysqld(API).'
2014-10-20 10:48:49 [MgmtSrvr] WARNING  -- Failed to allocate nodeid for API at 127.0.0.1. Returned error: 'No free node id found for mysqld(API).' - Repeated 3 times
2014-10-20 10:48:50 [MgmtSrvr] INFO     -- Node 3: Local checkpoint 28 started. Keep GCI = 29779 oldest restorable GCI = 29397
2014-10-20 10:48:50 [MgmtSrvr] WARNING  -- Failed to allocate nodeid for API at 127.0.0.1. Returned error: 'No free node id found for mysqld(API).'
2014-10-20 10:48:52 [MgmtSrvr] WARNING  -- Failed to allocate nodeid for API at 127.0.0.1. Returned error: 'No free node id found for mysqld(API).' - Repeated 3 times
2014-10-20 10:48:53 [MgmtSrvr] INFO     -- Node 3: Local checkpoint 28 completed
…
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: ParticipatingLQH = 0000000000000000
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LCP_COMPLETE_REP_Counter_DIH = [SignalCounter: m_count=0 0000000000000000]
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LCP_COMPLETE_REP_Counter_LQH = [SignalCounter: m_count=0 0000000000000000]
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LAST_LCP_FRAG_ORD = [SignalCounter: m_count=0 0000000000000000]
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LCP_COMPLETE_REP_From_Master_Received = 1
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: LCP Take over completed (state = 4)
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: ParticipatingDIH = 0000000000000000
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: ParticipatingLQH = 0000000000000000
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LCP_COMPLETE_REP_Counter_DIH = [SignalCounter: m_count=0 0000000000000000]
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LCP_COMPLETE_REP_Counter_LQH = [SignalCounter: m_count=0 0000000000000000]
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LAST_LCP_FRAG_ORD = [SignalCounter: m_count=0 0000000000000000]
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: m_LCP_COMPLETE_REP_From_Master_Received = 1
2014-10-20 14:37:44 [MgmtSrvr] ALERT    -- Node 4: Node 3 Disconnected
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: GCP Monitor: unlimited lags allowed
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: GCP Take over completed
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 5: kk: 35115/10 0 0
2014-10-20 14:37:44 [MgmtSrvr] ALERT    -- Node 5: Node 3 Disconnected
2014-10-20 14:37:44 [MgmtSrvr] INFO     -- Node 3: Node shutdown completed.
2014-10-20 14:37:47 [MgmtSrvr] INFO     -- Node 6: Communication to Node 3 opened
2014-10-20 14:37:48 [MgmtSrvr] INFO     -- Node 5: Communication to Node 3 opened
2014-10-20 14:37:48 [MgmtSrvr] INFO     -- Node 4: Communication to Node 3 opened
```

For baseline understanding:
- WARNING & INFO is normal cluster activity.

For error investigation:
- Isolate time period.
- Focus on ERROR, CRITICAL & ALERT events.
- Check time differences between events.

Logfiles:

ndb_3_out.log

```
2014-05-22 10:35:59 [ndbd] INFO     -- Angel pid: 12371 started child: 12372
2014-05-22 10:35:59 [ndbd] INFO     -- Configuration fetched from 'localhost:1186', generation: 1
NDBMT: non-mt
2014-05-22 10:35:59 [ndbd] INFO     -- NDB Cluster -- DB node 3
2014-05-22 10:35:59 [ndbd] INFO     -- mysql-5.6.17 ndb-7.3.5 --
2014-05-22 10:35:59 [ndbd] INFO     -- numa_set_interleave_mask(numa_all_nodes) : OK
2014-05-22 10:35:59 [ndbd] INFO     -- Ndbd_mem_manager::init(1) min: 78Mb initial: 78Mb
Adding 36Mb to ZONE_LO (1,1151)
Instantiating DBSPJ instanceNo=0
Started thread, index = 0, id = 12375, type = SocketClientThread
Started thread, index = 1, id = 12374, type = WatchDogThread
Started thread, index = 2, id = 12376, type = SocketServerThread
2014-05-22 10:35:59 [ndbd] INFO     -- Start initiated (mysql-5.6.17 ndb-7.3.5)
NDBFS/AsyncFile: Allocating 310392 for In/Deflate buffer
Started thread, index = 4, id = 12377, type = NdbfsThread
..
RESTORE table: 15 3021 rows applied
RESTORE table: 15 3032 rows applied
RedoPageCache: avoided 0 (0/0) page-reads
RedoOpenFileCache: Avoided 0 file-open/close closed: 0
Dbspj::execSTTOR() inst:0 phase=4
2014-10-20 10:48:49 [ndbd] INFO     -- Start phase 4 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 5 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 6 completed
2014-10-20 10:48:53 [ndbd] INFO     -- President restarts arbitration thread [state=1]
m_active_buckets.set(0)
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 7 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 8 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 9 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 100 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 101 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Node started
2014-10-20 10:48:54 [ndbd] INFO     -- Started arbitrator node 1 [ticket=6bcd00010ef206a4]
alloc_chunk(1492 16) -
2014-10-20 13:41:58 [ndbd] WARNING  -- timerHandlingLab, expected 10ms sleep, not scheduled for:
    2395 (ms)
2014-10-20 13:41:58 [ndbd] WARNING  -- Time moved forward with 2386 ms
2014-10-20 13:41:58 [ndbd] INFO     -- Watchdog: User time: 3543  System time: 5435
2014-10-20 13:41:58 [ndbd] WARNING  -- Watchdog: Warning overslept 2475 ms, expected 100 ms.
..
2014-10-20 14:37:37 [ndbd] INFO     -- Suma: handover to node 4 gci: 35115 buckets: 00000001 (2)
35115/0 (35114/4294967295) switchover complete bucket 0 state: 100shutdown handover
2014-10-20 14:37:44 [ndbd] INFO     -- Shutdown initiated
2014-10-20 14:37:44 [ndbd] INFO     -- Shutdown completed - exiting
2014-10-20 14:37:44 [ndbd] INFO     -- Angel shutting down
2014-10-20 14:37:44 [ndbd] INFO     -- Node 3: Node shutdown completed.
```

ndb_4_out.log

```
2014-05-22 10:36:01 [ndbd] INFO     -- Angel pid: 12381 started child: 12382
2014-05-22 10:36:02 [ndbd] INFO     -- Configuration fetched from 'localhost:1186', generation: 1
NDBMT: non-mt
2014-05-22 10:36:02 [ndbd] INFO     -- NDB Cluster -- DB node 4
2014-05-22 10:36:02 [ndbd] INFO     -- mysql-5.6.17 ndb-7.3.5 --
2014-05-22 10:36:02 [ndbd] INFO     -- numa_set_interleave_mask(numa_all_nodes) : OK
2014-05-22 10:36:02 [ndbd] INFO     -- Ndbd_mem_manager::init(1) min: 78Mb initial: 78Mb
Adding 36Mb to ZONE_LO (1,1151)
Instantiating DBSPJ instanceNo=0
Started thread, index = 0, id = 12385, type = SocketClientThread
Started thread, index = 1, id = 12384, type = WatchDogThread
Started thread, index = 2, id = 12386, type = SocketServerThread
2014-05-22 10:36:02 [ndbd] INFO     -- Start initiated (mysql-5.6.17 ndb-7.3.5)
NDBFS/AsyncFile: Allocating 310392 for In/Deflate buffer
Started thread, index = 4, id = 12387, type = NdbfsThread
..
RESTORE table: 10 0 rows applied
RESTORE table: 15 3021 rows applied
RESTORE table: 15 3032 rows applied
RedoPageCache: avoided 0 (0/0) page-reads
RedoOpenFileCache: Avoided 0 file-open/close closed: 0
Dbspj::execSTTOR() inst:0 phase=4
2014-10-20 10:48:49 [ndbd] INFO     -- Start phase 4 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 5 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 6 completed
m_active_buckets.set(1)
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 7 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 8 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 9 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 100 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Start phase 101 completed
2014-10-20 10:48:53 [ndbd] INFO     -- Node started
2014-10-20 10:48:54 [ndbd] INFO     -- Prepare arbitrator node 1 [ticket=6bcd00010ef206a4]
alloc_chunk(1492 16) -
2014-10-20 13:41:58 [ndbd] WARNING  -- Time moved forward with 2387 ms
2014-10-20 13:41:58 [ndbd] WARNING  -- timerHandlingLab, expected 10ms sleep, not scheduled for:
    2389 (ms)
2014-10-20 13:41:58 [ndbd] INFO     -- Watchdog: User time: 2908  System time: 3960
2014-10-20 13:41:58 [ndbd] WARNING  -- Watchdog: Warning overslept 2482 ms, expected 100 ms.
prepare to takeover bucket: 0
35115/0 (35114/4294967295) switchover complete bucket 0 state: 200shutdown takover
2014-10-20 14:37:44 [ndbd] INFO     -- findNeighbours from: 4882 old (left: 5 right: 6) new (5 6)
Finished with handling node-failure
```

Administration

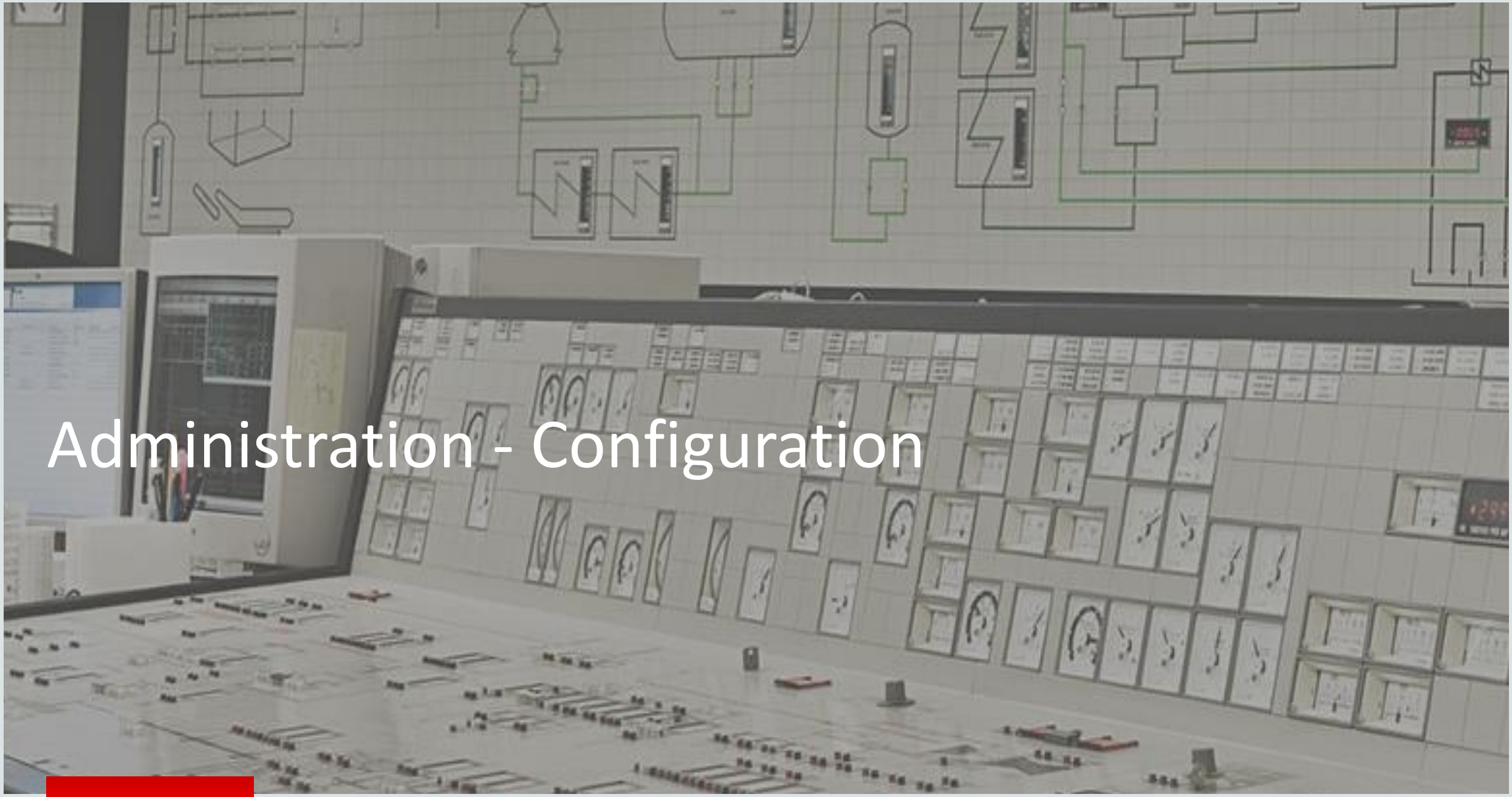
- Data nodes – designed for zero maintenance.
  - Logs
    - Writes error logs and trace files in its data directory.
    - Configurable how many error messages/trace files that should be saved
  - Memory Fragmentation
    - Free pages are reclaimed and can be reused
    - If you do a lot of insert/delete on VAR* attributes (of different sizes) you can get fragmentation
    - `OPTIMIZE TABLE` / Rolling restart of data nodes can help reduce fragmentation
- Management servers
  - Writes cluster log (rotating, size configurable) in its data directory
  - Cluster logs can be sent to Syslog if desired

Administration

- MySQL Servers
  - Binary logs - (if enabled) normally removed manually, can be done with `--expire_logs_days` but are you sure they have all been applied on the slave?
  - General log / error log / slow log - does not rotate automatically. A script called `mysql_log_rotate` can help.
    - Or move/cp log manually (or scripted) and do `FLUSH LOGS`
  - For MySQL Cluster it is also good to have a MySQL Server for administration purposes.
    - Perform offline `ALTER TABLE` (like change data type etc)
    - Export all non-NDB objects & procedures.

Administration
debugging

- `ndbmtd --nodaemon` / `--foreground` so it doesn't become a bg process, displaying output to the console.
- `ndb_`*`node_id`*`_trace.log.`*`trace_id`* is the trace file generated after an error.
- `ndb_mgm> clusterlog on debug`



Administration - Configuration

Configuration
Users & permissions

- NDB (7.6 and earlier) Ensure all sqlnodes / mysqld's have the same user grant tables:
  - `share/ndb_dist_priv.sql`
    - Run from 1 sqlnode
    - Propagates to all other sqlnodes, the same `mysql.user` contents.
    - Converts `user` table from MYISAM to NDBCLUSTER.
  - Standard procedures convert existing users to every other sqlnode.
  - Changes storage engine for mysql.user table to NDBCLUSTER.
- New model in NBD 8.0 where you specify distributed privileges for each user as part of the GRANT statement, use keyword "NDB_STORED_USER".

Configuration
Network

- `[tcp]` section for direct connections between nodes.

```
[tcp]
NodeId1=3
Hostname=10.0.0.3
NodeId2=4
Hostname=10.0.0.4
```

- Other options `[sci]` & `[shm]`.
- Use `ndbmtd` **`--bind-address=192.168.56.18`** to bind to a specific network interface.

Datanodes

- NDB - single threaded data node for TELCO platforms
- NDBMTD - multi-threaded data nodes for general use.
- Data nodes are made up several components:
  - LDM (Local Data Manager)
    - LQH, local query handler, local transaction management, local data operations.
    - ACC, Access control, lock manager
    - TUP, tuple manager, manages physical storage of data (read, insert, update, delete, and monitoring changes of tuples)
  - TC (Transaction Coordinator)
    - Keeps track of user transactions
    - Handle native NDB transactions cross the cluster

NDB Threads I

- LDM threads
  - Number of LDM threads should be same in all nodes.
  - Changing number of LDM threads is hard and requires rebuild of tables.
  - Up to 32 LDM threads is supported
- NoOfFragmentLogParts:
  - Number of REDO log parts for NDB.
  - Each LDM need to have at least one dedicated REDO log part.

NDB Threads II

- TC (Transaction Coordinator) threads
  - Normally 25% of LDM threads but up to 100% for write intense applications
- Send threads
  - Handles sending distributed signals, 2-3 is good enough.
- Receive threads
  - Handles reception of distributed signals, 2-3 is good enough.
- IO threads, handles disk IO, depending on disks.
- Main and replication threads, one of each.

NDB Threads III

- Before NDB 7.2 only way to configure multiple threads was to use `MaxNoOfExecutionThreads`
  - MAX value 8 means 1 TC, 1 send/recs, 1 SUMA and 4 LQH thread.
  - With NDB 7.5 and above we support ranges up to max 72 threads
- As of NDB 7.2 and onwards you can use `ThreadConfig = main={count=1}, tc={count=2}, ldm={count=4}, recv={count=1}, io={count=1}, send={count=1} ,rep={count=1}`
  - Using ThreadConfig you can also pin processes to specific cores by specifying options cpubind like ldm={count=4, cpubind=1-4}

Security

- Basics:
  - Make sure all [api] / [mysqld] entries in config.ini have a hostname assigned.
  - Locally installed firewall rules only allow access from the specific networks & IP's ranges. Port 1186 only accessible from specific nodes.

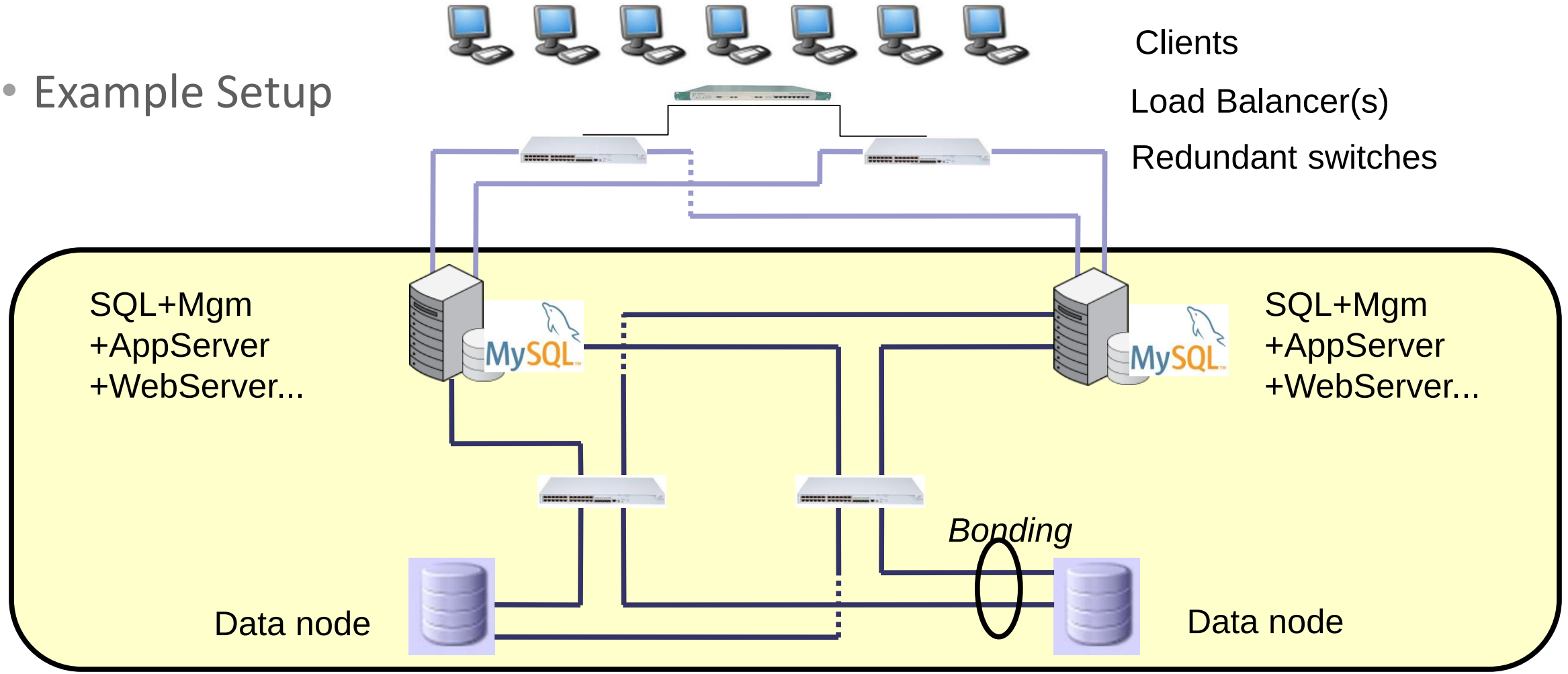
- Patching Policy
  - Will depend heavily on Development Teams Roadmap & Go-To-Market.
  - Understand Oracle MySQL's policy on Support and Release frequency.

Networking

- NDB API & NDB_MGM don’t support IPv6.
  - MUST have all connections originating in IPv4, between ndbd, ndb_mgmd & ndb_mgm plus all API’s.
  - MySQL Servers (mysqld) do / can use it to connect to other sqlnodes.
- Transporters
  - Traditional Ethernet or Infiniband (IPoIB)
- Number of NIC’s per data node (High Availability):
  - 2 with bonding
- Dedicated Subnet for Datanodes:
  - dual Switches.
- Ports open / default values
  - ServerPort between sqlnodes & datanodes? FW?

Networking

- Example Setup



Administration - Sizing

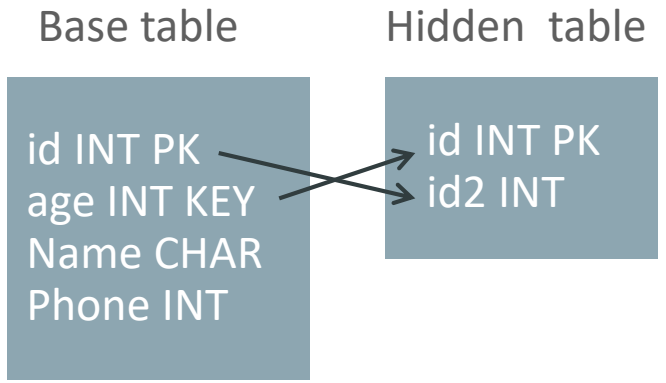
Best Practices - Sizing Tips
Data set

Data Type	Size
VARCHAR(n) and VARBINARY(n)	n + 2 rounded up to next 4 byte boundary
CHAR(n) and BINARY(n)	n rounded up to the next 4 byte boundary
BLOB/TEXT(n)	n < 256 then n, otherwise: 256 + (n-256) rounded up to the next multiple of 2000. For each BLOB column an extra table is used to store the BLOB overflow data in 2000B chunks.
DATE	4
INT	4
BIGINT	8

http://dev.mysql.com/doc/refman/5.6/en/storage-requirements.html

Best Practices - Sizing Tips
Data set

- Fixed overhead per row for data memory is 16 bytes.
  - 4 extra bytes if you have any nullable columns in your tables
  - 8 extra bytes for each BLOB column
- Every MySQL cluster table must have a primary key. Every PK creates a hash index (HI) which has a size of 20 bytes
- Every Ordered Index has a size of 16 bytes (resides inside DM)
- Every UK creates a new “hidden” cluster table with the UK column as PK and the originating tables PK as an value column.



Best Practices - Sizing Tips
Data set “example”

```
CREATE TABLE example (
          IDX INT PRIMARY KEY,
           Col1 VARCHAR(13),
           Col2 INT,
           UNIQUE INDEX(Col2)
            /* USING HASH */
) ENGINE=ndbcluster;
```

Column	Size	Size in NDB
IDX	4	4
Col1	13	16
Col2	4	4
Total		24

- PK 20 bytes per row
- Unique key (own table 4+4+20) per row
- Ordered index on Col2 16 bytes (can be avoided)
- DM = 24 + 8 + 16 = 48 bytes
- IM = 20 + 20 = 40 bytes
- Total row size 24 + 20 + 28 + 16 = 88 bytes

Best Practices - Sizing Tips
Memory

- Calculate data set in un-clustered setup, divide by number of data nodes.
  - `DataMemory` x3 data set (guestimate)
  - `(before NDB 7.6) IndexMemory` x 0.25 of data set (guestimate)
- `ndb_mgm> all report memoryusage` <- And monitor.
- `mysql> select * from ndbinfo.memoryusage;`

Best Practices - Sizing Tips
Disk

- Datanode `datadir`: local checkpoints, trace files, log files, PID files &error log.
  - LCP: 2 x sizeof(used DataMemory)
- `FileSystemPath`: metadata files, REDO & UNDO logs & Data files. By default, subdirectory in datadir “ndb_*NodeID*_fs”.
  - NDB 7.6 with partial LCP set REDO size to 1.5 of DataMemory (LCP)
  - NDB 7.5 ore older set REDO: [4-6] x DataMemory
    - More (6x) REDO log for write intensive
    - Don’t have a too short REDO (e.g 2x or 3x)
- `BackupDir`: default = `FileSystemPath`, where backups are placed taken from ndb_mgm console.
  - Backups: sizeof(used DataMemory)
  - TableSpace (if disk data tables): Must fit dataset.

MCM Hands On Lab
3 Backup and Restore

- Lab-3
  - https://github.com/wwwted/ndb-cluster-workshop

Program Agenda

1. Introduction
2. Getting started: configuration/install/start/stop
3. Administration: backup/upgrade/logs/conf/sizing
4. Monitoring, surveillance and problem solving
5. Best practices, architectures and case studies
6. NDB 7.6 and what's new in NDB 8.0
7. Geo-replication

Monitoring Tools

- **ndb_mgm**
  - `all status | all report memoryusage | 3 status | 4 report memory`
- `ndbinfo` schema / `ndbinfo_select_all`
- **ndb_desc**
  - View types of indexes created, i.e. if `using hash` was included with PK.
  - View partitioning information on specific tables
- `ndb_config`
  - `--config-info`
  - `-q NodeId,HostName,NoOfReplicas,DataMemory --type=ndbd`
- `ndb_mgmd –print-full-config`

Monitoring Tools
continued.

- `ndb_show_tables`
- `ndb_desc -p -d<db> <table>`
- `ndb_select_count  ndb_delete_all  ndb_select_all`
  - `ndb_select_count -d test city`
- `ndb_index_stat` to view & modify index stats
- MySQL Workbench
- MySQL Enterprise Monitor

Monitoring Tools
ndbinfo / ndbinfo_select_all

```
mysql> show engines;
mysql> show tables from ndbinfo;
mysql> SELECT * from ndbinfo.memoryusage;
mysql> SELECT * from ndbinfo.cluster_transactions;
mysql> SELECT * from ndbinfo.cluster_operations;
```

- Mapping above to a mysql connnection:

```
mysql> SELECT * from performance_schema.threads where
TYPE="FOREGROUND";
mysql> SELECT * from ndbinfo.server_operations;
```

- Low level view of what is happening:

```
mysql> SELECT * from ndbinfo.counters;
```

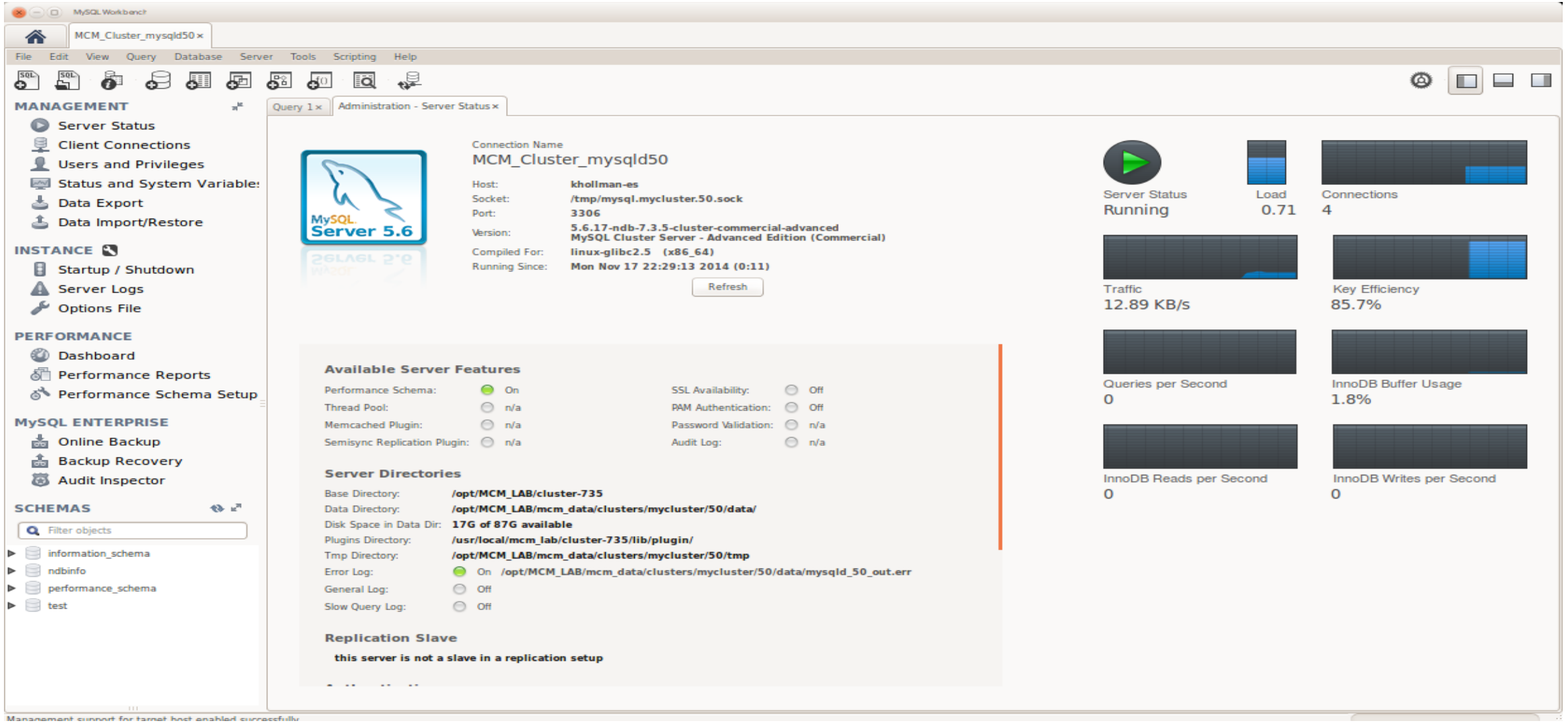
Monitoring Tools
ndb_desc

```
$ ndb_desc -d test City -p
-- City --
Version: 16777218
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 5
Number of primary keys: 1
Length of frm data: 369
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 4
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
HashMap: DEFAULT-HASHMAP-3840-4
-- Attributes --
ID Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
Name Char(35;latin1_swedish_ci) NULL AT=FIXED ST=MEMORY DEFAULT ""
CountryCode Char(3;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY DEFAULT ""
District Char(20;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY DEFAULT ""
Population Int NOT NULL AT=FIXED ST=MEMORY DEFAULT 0
-- Indexes --
PRIMARY KEY(ID) - UniqueHashIndex
CountryCode(CountryCode) - OrderedIndex
PRIMARY(ID) - OrderedIndex
-- Per partition info --
Partition   Row count   Commit count           Frag fixed memory      Frag varsized memory    Extent_space           Free extent_space
0           3021        3021                   360448                 0                       0                      0
2           3032        3032                   360448                 0                       0                      0
1           3027        3027                   360448                 0                       0                      0
3           3005        3005                   360448                 0                       0                      0
```

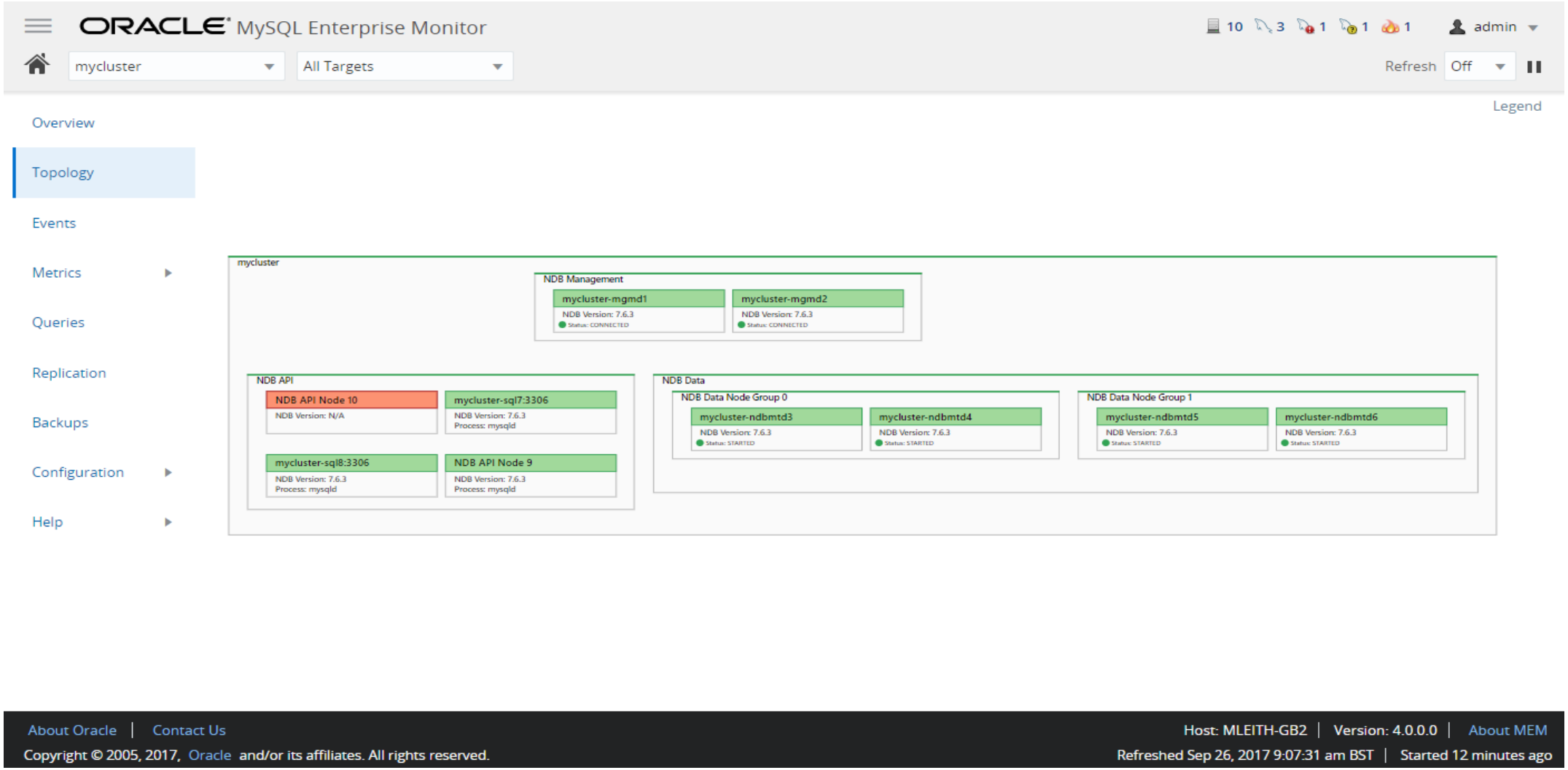
Monitoring Tools
ndb_mgmd

```
$ ndb_mgmd -f /usr/local/mysql-cluster/conf/config.ini --
   config-dir=/usr/local/mysql-cluster --print-full-config

MySQL Cluster Management Server mysql-5.6.17 ndb-7.3.5
[SYSTEM]
Name=MC_20141020173233
PrimaryMGMNode=1
ConfigGenerationNumber=0
[ndb_mgmd(MGM)]
HostName=khollman-es
DataDir=/opt/mysql/735/mgmd_data
NodeId=1
PortNumber=1186
ArbitrationRank=1
ArbitrationDelay=0
ExtraSendBufferMemory=0
TotalSendBufferMemory=0
HeartbeatIntervalMgmdMgmd=1500
[mysqld(API)]
HostName=
NodeId=10
ArbitrationRank=0
ArbitrationDelay=0
MaxScanBatchSize=262144
BatchByteSize=16384
BatchSize=256
ExtraSendBufferMemory=0
TotalSendBufferMemory=0
AutoReconnect=1
DefaultOperationRedoProblemAction=1
DefaultHashmapSize=0
[mysqld(API)]
HostName=
NodeId=11
ArbitrationRank=0
ArbitrationDelay=0
MaxScanBatchSize=262144
BatchByteSize=16384
BatchSize=256
ExtraSendBufferMemory=0
TotalSendBufferMemory=0
AutoReconnect=1
DefaultOperationRedoProblemAction=1
DefaultHashmapSize=0
[mysqld(API)]
HostName=
NodeId=12
ArbitrationRank=0
ArbitrationDelay=0
MaxScanBatchSize=262144
BatchByteSize=16384
BatchSize=256
ExtraSendBufferMemory=0
TotalSendBufferMemory=0
AutoReconnect=1
DefaultOperationRedoProblemAction=1
DefaultHashmapSize=0
……
```

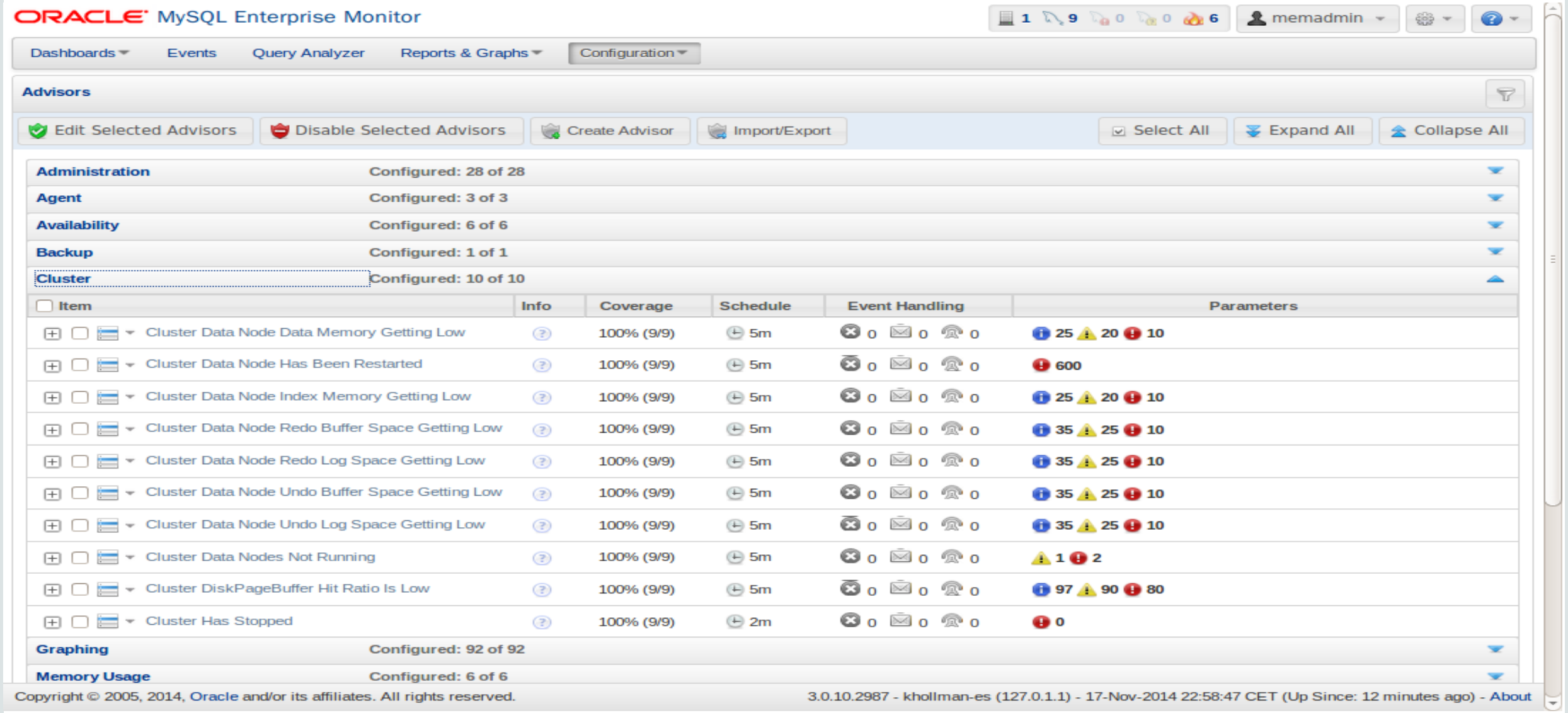
Workbench



MySQL Enterprise Monitor



Enterprise Monitor cont.



MCM Hands On Lab
4 Monitoring - NDBINFO

- Lab-4
  - https://github.com/wwwted/ndb-cluster-workshop

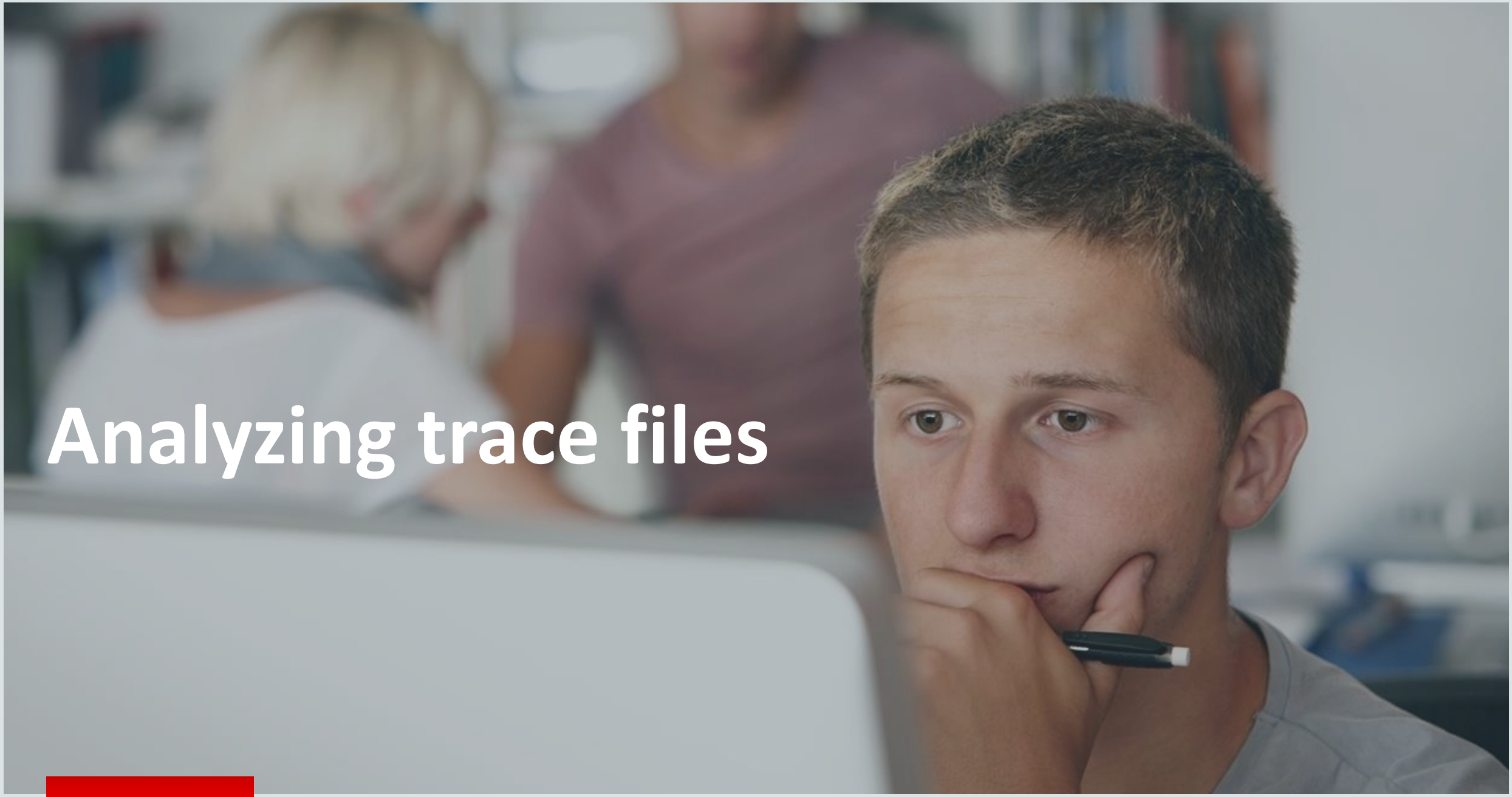
```
mysql> SELECT * FROM memoryusage;
+---------+--------------------+---------+------------+----------+-------------+
| node_id | memory_type        | used    | used_pages | total    | total_pages |
+---------+--------------------+---------+------------+----------+-------------+
|       1 | Data memory        | 2228224 |         68 | 83886080 |        2560 |
|       1 | Index memory       |  458752 |         56 | 19398656 |        2368 |
|       1 | Long message buffer |  393216 |       1536 | 67108864 |      262144 |
|       2 | Data memory        | 2228224 |         68 | 83886080 |        2560 |
|       2 | Index memory       |  458752 |         56 | 19398656 |        2368 |
|       2 | Long message buffer |  393216 |       1536 | 67108864 |      262144 |
+---------+--------------------+---------+------------+----------+-------------+
```

MCM Hands On Lab
5 Monitoring - Benchmark

- Lab-5
  - https://github.com/wwwted/ndb-cluster-workshop

```
mysqlslap -h127.0.0.1 -P3310 -uroot -proot
          --auto-generate-sql
          --auto-generate-sql-guid-primary
          --auto-generate-sql-secondary-indexes=2
          --auto-generate-sql-load-type=read
          --auto-generate-sql-write-number=200000
          --auto-generate-sql-execute-number=10
          --concurrency=6 --engine=ndbcluster
```

Analyzing trace files

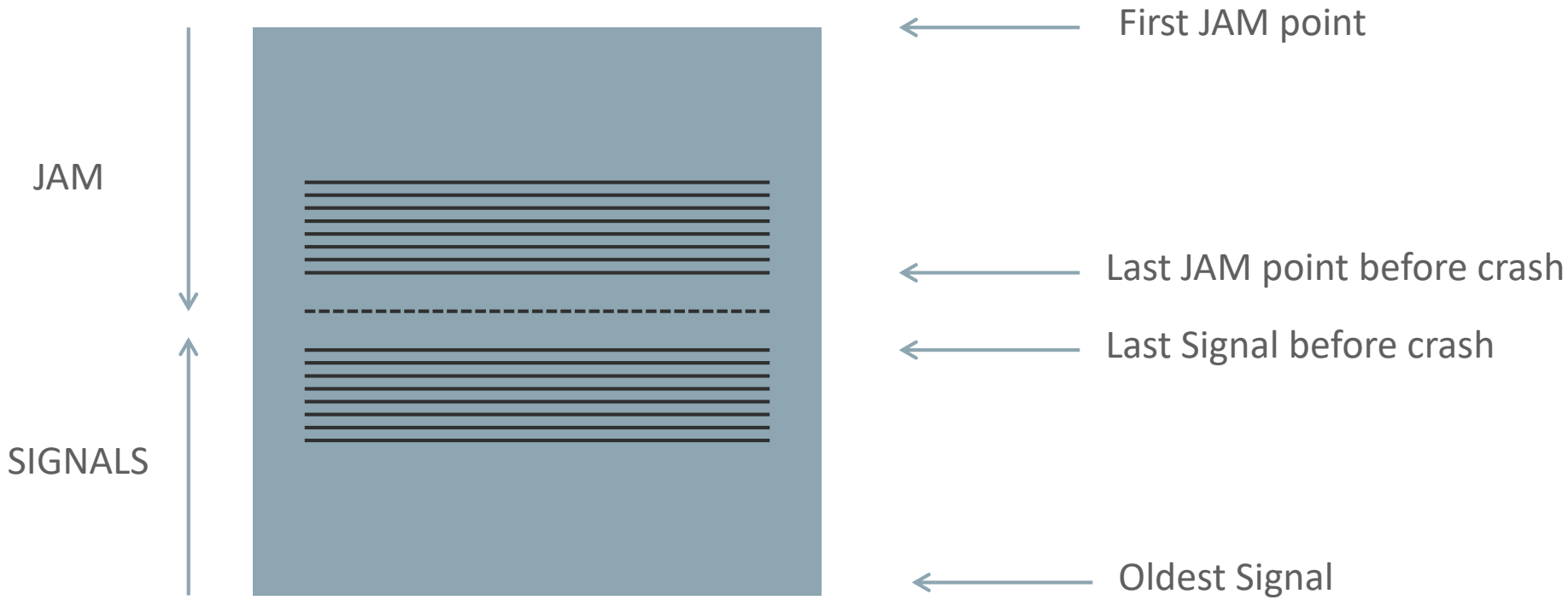


Crash Investigations and Trace Files
files to Analyze

- During a crash investigation the files collected by the `ndb_error_reporter` script is your best friend.
- If you have problem and cluster created trace files, create a support ticket and attach the trace files.
- If you want to investigate problems yourself;
  - Start om central Cluster log file located on management node
  - Then look in local logs on data nodes (.err and .out)
  - If problem is related to MySQL API nodes, look in error logs for MySQL Server
  - If Trace files have been created by data nodes, they will show your what caused data node to stop.

Crash Investigations and Trace Files
analyzing trace files

- The trace files are the ones containing the most detailed information about the crash. The files is divided into two parts, JAM and Signals.



Crash Investigations and Trace Files
analyzing trace files

- The JAM points are generated by the following two macros in the source code:
  - jam(), inserts the source code line number into the buffer.
  - jamEntry(), additionally inserts a new line and the name of the block before the line number
- MySQL Cluster uses two kinds of signals:
  - asynchronous signals that are added to a queue and processed according to priority and the order they are inserted into the queue
  - synchronous signals that block until they have been processed

Crash Investigations and Trace Files
analyzing trace files JAM()

```
QMGR  004152 004189 004201 004207 004209 004247 004252
QMGR  004273 004287 004287 004287 004287 004287 004287 004287
      004287 004320 005250 006018 006018 006018 006018 006021
      006018 006018 006018 006018 006018 006018 006018 006018
      006018 006018 006018 006018 006018 006018 006018 006018
      006018 006018 006018 006018 005259
DBDIH 016151 016159 016163 016169 016169 016172 016192
QMGR  005266 005277 005315 005324 005340 005353 005420 005961
      005963 005975
```

So the first place to look is the QMGR kernel block. The source code for the kernel blocks is located in storage/ndb/src/kernel/blocks and every block is implemented as a C++ class. Most blocks has its own subdirectory. The file the line number refers to is the <Block>Main.cpp where <Block> should be replaced by the name of the block, e.g. for the QMGR block, the filename is QmgrMain.cpp. Looking up line 5975 in QmgrMain.cpp

```
void
Qmgr::stateArbitCrash(Signal* signal)
{
        jam();
I       f (arbitRec.newstate) {
                jam();
                CRASH_INSERTION((Uint32)910 + arbitRec.state);
                arbitRec.setTimestamp();
                ..............
```

Crash Investigations and Trace Files
analyzing trace files Signals

- r.bn – reciever block
- r.proc – node ID of reciever
- gsn – Global Signal Name
- s.bn – sender block
- s.proc – Node ID of sender

```
---------- Signal ----------
r.bn: 252 "QMGR", r.proc: 4, r.sigId: 2437605 gsn: 343 "PREP_FAILCONF" prio: 0
s.bn: 252 "QMGR", s.proc: 4, s.sigId: 2437604 length: 2 trace: 8 #sec: 0 fragInf: 0
H'00000004 H'00000002
---------- Signal ----------
r.bn: 252 "QMGR", r.proc: 4, r.sigId: 2437604 gsn: 126 "CLOSE_COMCONF" prio: 0
s.bn: 266 "TRPMAN", s.proc: 4, s.sigId: 2437603 length: 19 trace: 8 #sec: 0 fragInf: 0
xxxBlockRef = (252, 4) failNo = 2 noOfNodes = 1
Nodes: 3
---------- Signal ----------
r.bn: 266 "TRPMAN", r.proc: 4, r.sigId: 2437603 gsn: 126 "CLOSE_COMCONF" prio: 0
s.bn: 266 "TRPMAN", s.proc: 4, s.sigId: 2437602 length: 19 trace: 8 #sec: 0 fragInf: 0
xxxBlockRef = (252, 4) failNo = 2 noOfNodes = 1
Nodes: 3
.....
```

When analyzing the signal list for a multi-threaded data node, in general all but one thread will have the STOP_FOR_CRASH signal as the last one. **The thread not having this signal is the interesting one as it is responsible for the crash**. The STOP_FOR_CRASH signal is sent to the remaining threads to stop them as well

Crash Investigations and Trace Files
introduction to the Source Code

The most important directories and files are:

- `storage/ndb/include`
  - The shared header files for MySQL Cluster. One directory that may be of particular interest is kernel/signaldata which contains all of the classes implementing the signals.
- `storage/ndb/src`
  - The source code for the core cluster parts (for example the data nodes and the management server/client, but not high level interaction such as the NDBCluster storage engine, ClusterJ)
- `storage/ndb/src/kernel`
  - The data nodes.
- `storage/ndb/src/kernel/vm`
  - Code that is common for the kernel blocks.
- `storage/ndb/src/kernel/blocks`
  - The code for the actual kernel blocks.

Administration
Adding data nodes online

- Add new data nodes to config.ini.
- Rolling restart of all components (management node, data nodes & sql nodes).
- Start **new** data nodes with `--initial`.
- Create new node group via management console.
- `alter online table xxx reorganize partition;` <- 1 online DDL per table.
- `optimize table xxx;`

https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-online-add-node-example.html

MCM Hands On Lab
6 Add more capacity

- Lab-6
  - https://github.com/wwwted/ndb-cluster-workshop

```
mcm> show status -r mycluster;
+--------+----------+-----------+----------+-----------+------------+
| NodeId | Process  | Host      | Status   | Nodegroup | Package    |
+--------+----------+-----------+----------+-----------+------------+
| 49     | ndb_mgmd | 127.0.0.1 | running  |           | cluster758 |
| 1      | ndbmtd   | 127.0.0.1 | running  | 0         | cluster758 |
| 2      | ndbmtd   | 127.0.0.1 | running  | 0         | cluster758 |
| 3      | ndbmtd   | 127.0.0.1 | added    | n/a       | cluster758 |
| 4      | ndbmtd   | 127.0.0.1 | added    | n/a       | cluster758 |
| 50     | mysqld   | 127.0.0.1 | running  |           | cluster758 |
| 51     | mysqld   | 127.0.0.1 | running  |           | cluster758 |
| 52     | ndbapi   | *127.0.0.1 | added   |           |            |
| 53     | ndbapi   | *127.0.0.1 | added   |           |            |
| 54     | ndbapi   | *127.0.0.1 | connected |         |            |
| 55     | ndbapi   | *127.0.0.1 | connected |         |            |
+--------+----------+-----------+----------+-----------+------------+
```

Program Agenda

1. Introduction
2. Getting started: configuration/install/start/stop
3. Administration: backup/upgrade/logs/conf/sizing
4. Monitoring, surveillance and problem solving
5. Best practices, architectures and case studies
6. NDB 7.6 and what's new in NDB 8.0
7. Geo-replication

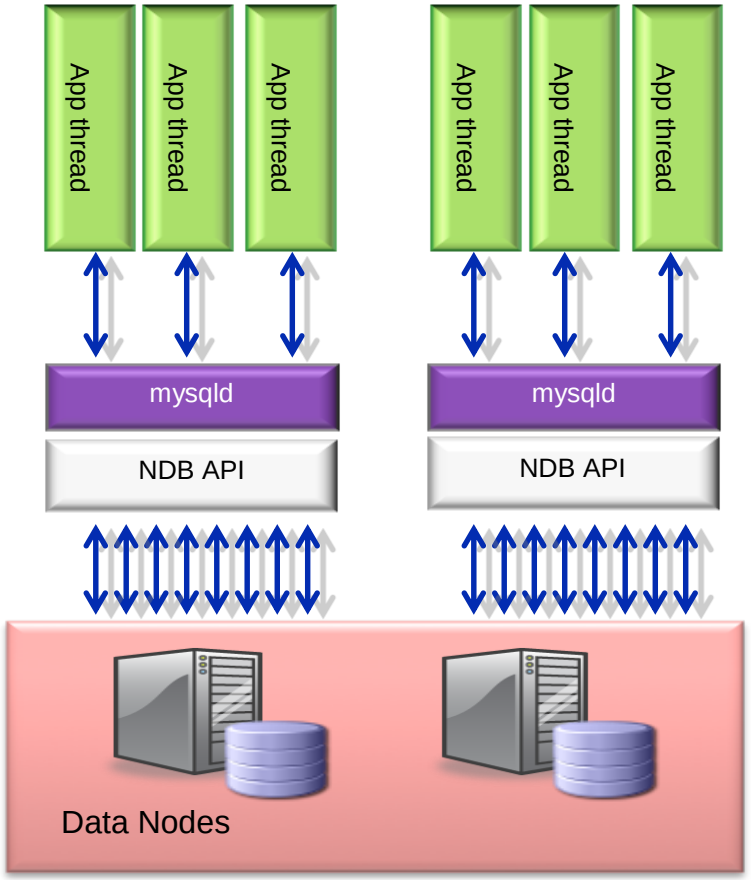
Best Practices
General Design Considerations

- MySQL Cluster is designed for
  - Short transactions
  - Many parallel transactions
- Utilize Simple access patterns to fetch data
  - Use efficient scans and batching interfaces
- Analyze what your most typical use cases are
  - optimize for those

Overall design goal

Minimize network roundtrips for your most important requests!

Best Practices
Parallel operations for throughput



- Network hops increase latency (e.g. Compared with InnoDB read of cached data)
- Increase throughput by sending in lots of parallel operations
- Multiple client connections (sessions) to each MySQL Server
- Multiple MySQL Servers
- Connection pooling between MySQL Servers and data nodes
- Set `ndb-cluster-connection-pool` > 1 in my.cnf
- Ensure enough `[api]` sections in config.ini

Best Practices
User defined partitioning

- User defined partitioning is “holy grail” of NDB performance!

- You can choose to partition on any part of PK
  - `CREATE TABLE … PARTITON BY (X)`
  - Partition on the correct columns
  - You can only partition on whole or part of PK
  - Try to collocate data that is joined
  - You have the same TC during whole transaction

Best Practices
User defined partitioning II

- PK (a)
  - Data partitioned by hash(a)
  - `select * from ptable where a=2`
- PK (a,b)
  - Data partitioned by hash(a,b)
  - `select * form ptable where b=2`?
  - `select * from ptable where a=2`?
  - `select * from ptable where a=2 and b=4`?
- `create table ptable … PK(a,b) PARTITION by (b)`
  - Data partitioned by hash(b) ….

Best Practices: Distribution Aware Apps

```
SELECT SUM(population) FROM towns WHERE
country="UK";
```

Partition Key		
Primary Key		
town	country	population
Maidenhead	UK	78000
Paris	France	2193031
Boston	UK	58124
Boston	USA	617594

```
SELECT SUM(population) FROM towns WHERE
town="Boston";
```

Partition Key		
Primary Key		
town	country	population
Maidenhead	UK	78000
Paris	France	2193031
Boston	UK	58124
Boston	USA	617594

- Partition selected using hash on Partition Key
  - Primary Key by default
  - User can override in table definition
- MySQL Server (or NDB API) will attempt to send transaction to the correct data node
  - If all data for the transaction are in the same partition, less messaging -> faster
- Aim to have all rows for high-running queries in same partition

Best Practice: Distribution Aware – Multiple Tables

Partition Key

Primary Key

sub_id	age	gender
19724	25	male
84539	43	female
19724	16	female
74574	21	female

Partition Key

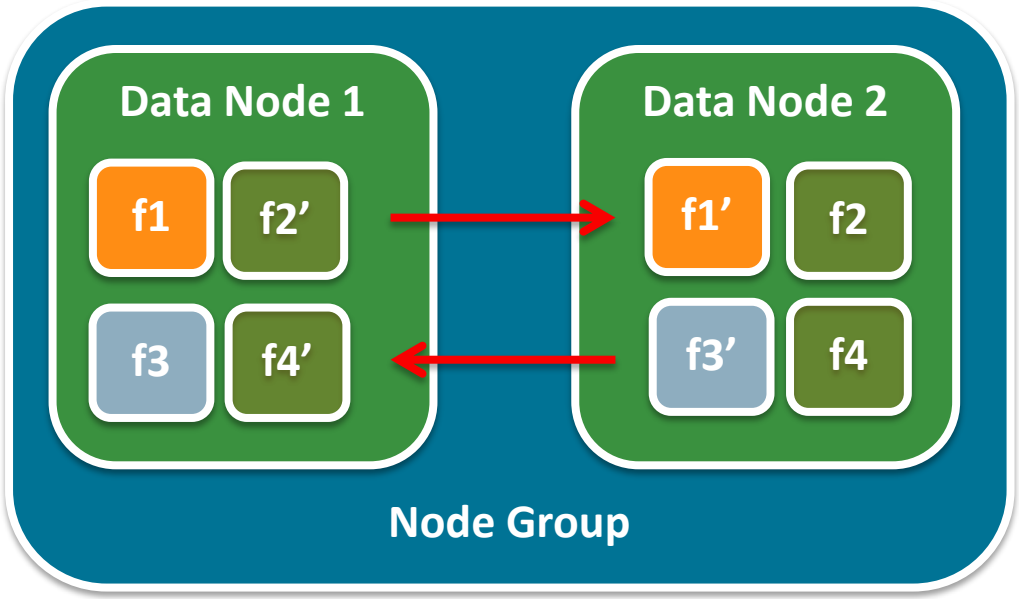
Primary Key

service	sub_id	svc_id
twitter	19724	76325732
twitter	84539	67324782
facebook	19724	83753984
facebook	73642	87324793

- Extend partition awareness over multiple tables
- Same rule – aim to have all data for instance of high running transactions in the same partition

```
ALTER TABLE service_ids PARTITION
  BY KEY(sub_id);
```

Best Practices
Improving read/join access – READ_BACKUP (NDB 7.5)

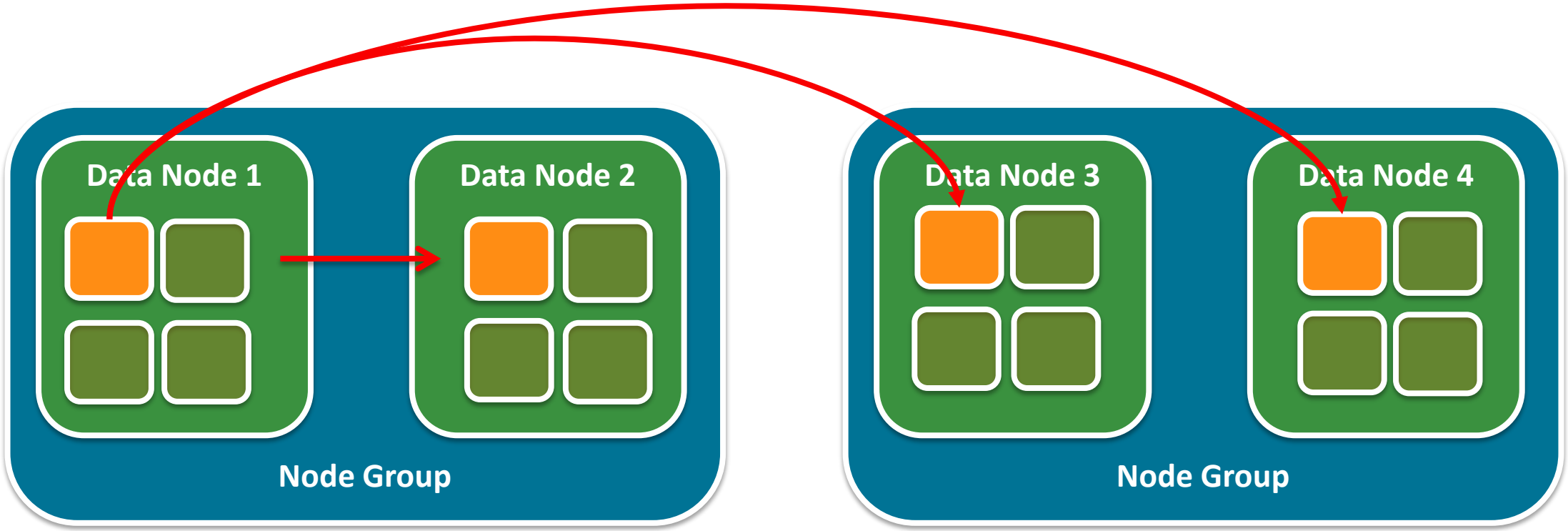


- Reading from backup allows to read from any copy (f1 or f1‘ (backup))
- Previously all reads were directed towards the primary fragment only

Best Practices
Improving read/join access – READ_BACKUP (NDB 7.5)

- Per default we only read the primary fragment of data, the backup fragment that might be ”closer” is not used.
- Read backup can help to avoid network traffic by reading as much data as possible locally, read backup will be most useful in a 2 data node setup where MySQL API nodes are co-located with data nodes.
  - Can be configured when creating table using:
    CREATE TABLE .... ENGINE=NDB COMMENT="NDB_TABLE=READ_BACKUP=1”;
    or by running ALTER TABLE <table-name> COMMENT="NDB_TABLE=READ_BACKUP=1”;
  - There is also a variable ndb_read_backup that can be set in MySQL configuration file that will ensure read backup for all tables created after variable set set.
  - It’s important to remember to also set ndb_data_node_neighbour if we have co-located MySQL/Data ndoes to ensure we read from co-located data node.

Best Practices
Improving read/join access – FULLY_REPLICATED (NDB 7.5)



- Fully replicated allows a table to be read/written locally on any node
- Ideal for static data, faster join performance

Best Practices
Improving read/join access – FULLY_REPLICATED (NDB 7.5)

- All tables in MySQL Cluster are distributed over all data nodes be default, under some circumstances this might not be optimal. One such case is when you want to join data between tables, this will cause lots of network traffic and slow down your queries.
- Fully replicated can help to avoid network traffic by reading as much data as possible locally, this should be considered primary for static tables that are used in join operations with your larger and more frequently updated tables.
  - Fully replicated can be configured when creating table using:
    CREATE TABLE .... ENGINE=NDB COMMENT="NDB_TABLE=FULLY_REPLICATED=1”;
    or by running ALTER TABLE <table-name> COMMENT="NDB_TABLE=FULLY_REPLICATED=1”;
  - There is also a configuration variable named ndb_fully_replicated that can be set in MySQL configuration file that will ensure fully_replicated for all tables created after variable is set.
  - It’s important to remember to also set ndb_data_node_neighbour if we have co-located MySQL/Data nodes to ensure we read from co-located data node.

Hardware

MySQL Cluster Hardware Selection:
Disk Subsystem for Checkpointing

Entry-Level

LCP
REDOLOG

SATA/SAS Drives
- For read-mostly
- No redundancy

(but other data node is the mirror)

Mid-Range

LCP
REDOLOG

SAS/SSD's
- Heavy duty (many MB/s)
- No redundancy
  (but other data node is the mirror)

High-End

LCP / REDOLOG

SSDs or NVMe
- Heavy duty (many MB/s)
- Disk redundancy (RAID1+0), hot swap

- REDO, LCP, BACKUP – written sequentially in small chunks (256KB)
- If possible, use ODIRECT = 1

MySQL Cluster Hardware Selection:
Disk Data Storage

Recommended Minimum

LCP
REDOLOG
UNDOLOG
TABLESPACE

2 x SAS 10K RPM
or 2 x SSD

High-End Recommendation

UNDOLOG
(REDO LOG)

TABLESPACE 1

TABLESPACE 2

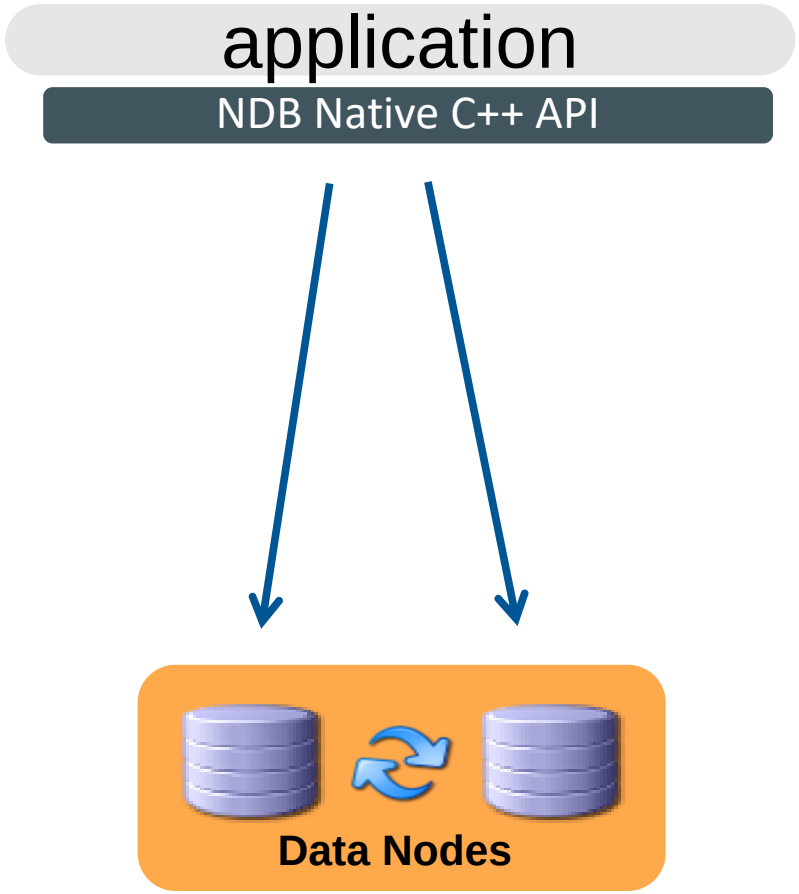
(REDO LOG / UNDO LOG)
LCP

4 x SAS 10-15K RPM or SSD

- Use High-End Recommendation for heavy read / write workloads
  - (1000's of 10KB records per sec) of data (i.e. Content Delivery platforms)
- Having TABLESPACE on separate disk is good for read performance
- Enable WRITE_CACHE on devices

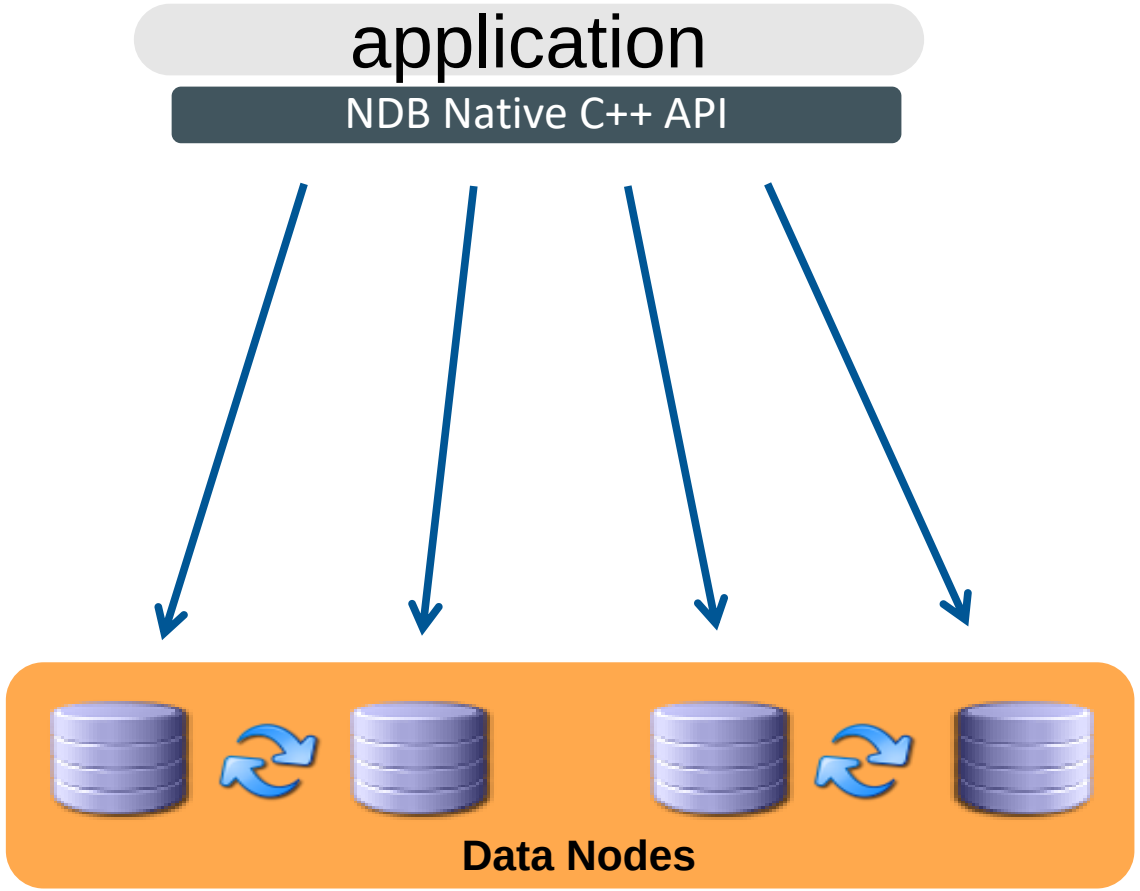
Architecture

Architectures
Small applications



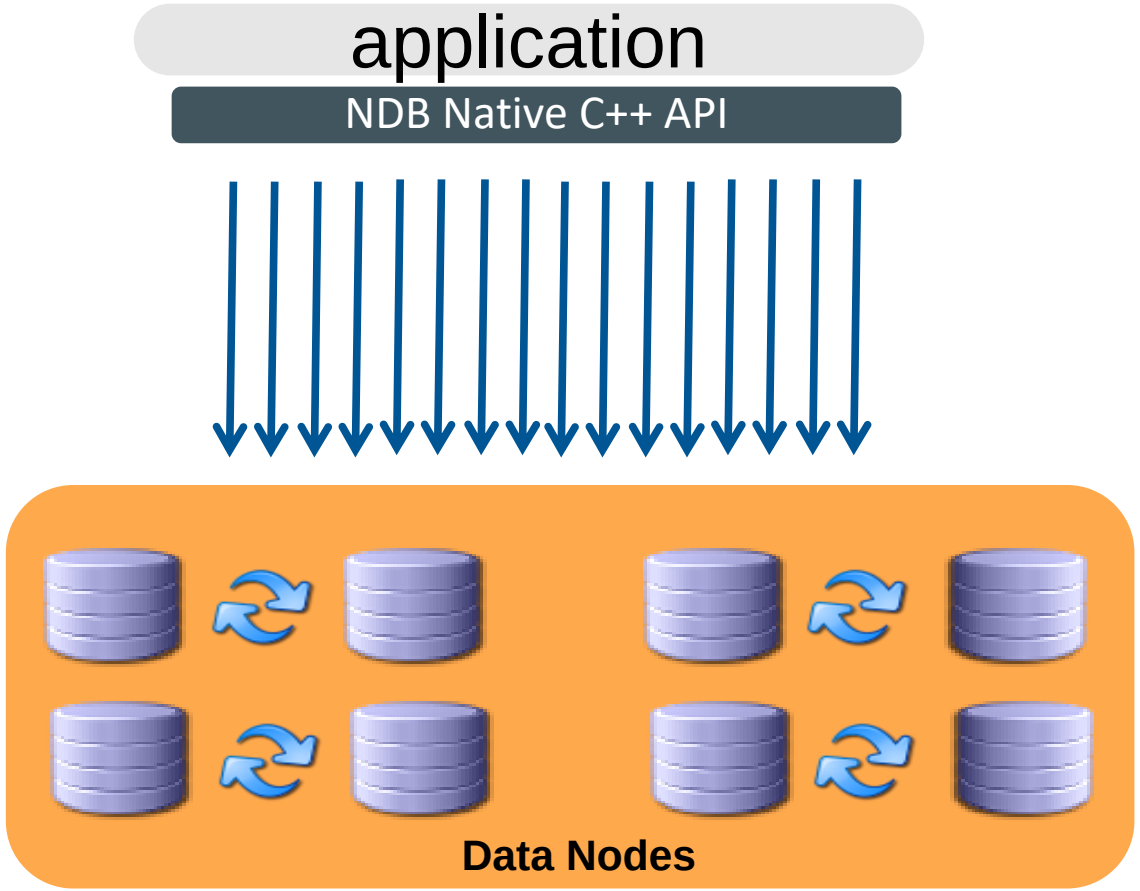
- Start small and grow.
- Smaller operators, DNS/DHCP/EMUM services.
- Minimal installation on 3 physical servers due to arbitration.
- Max 200M QPM and 20M UPM.
- Size of database limited by RAM of smallest server.

Architectures
Medium applications



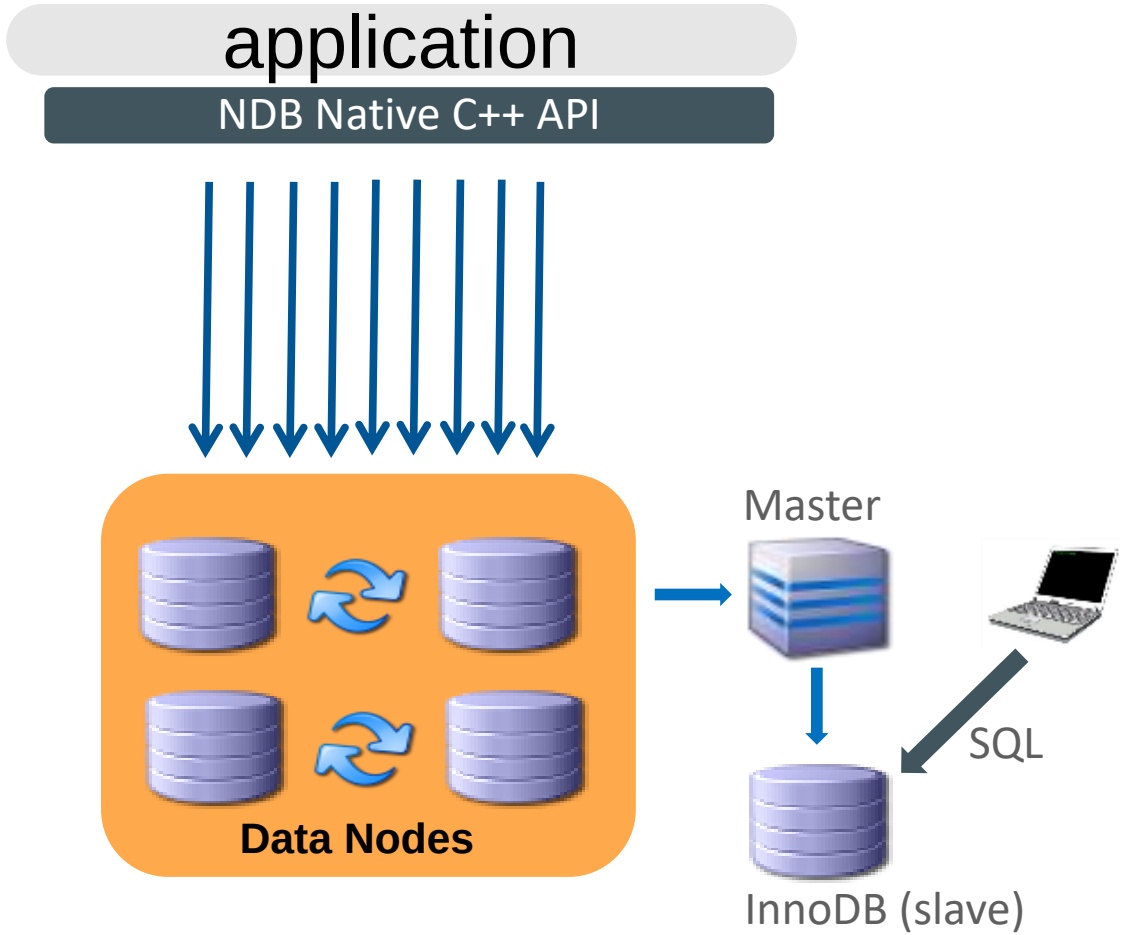
- Most common setup.
- 4 server setup, management nodes can be co-located with data nodes.
- Max 600M QPM and 55M UPM.
- Size of DB limited by 2xRAM (RAM of smallest server).

Architectures
Big applications



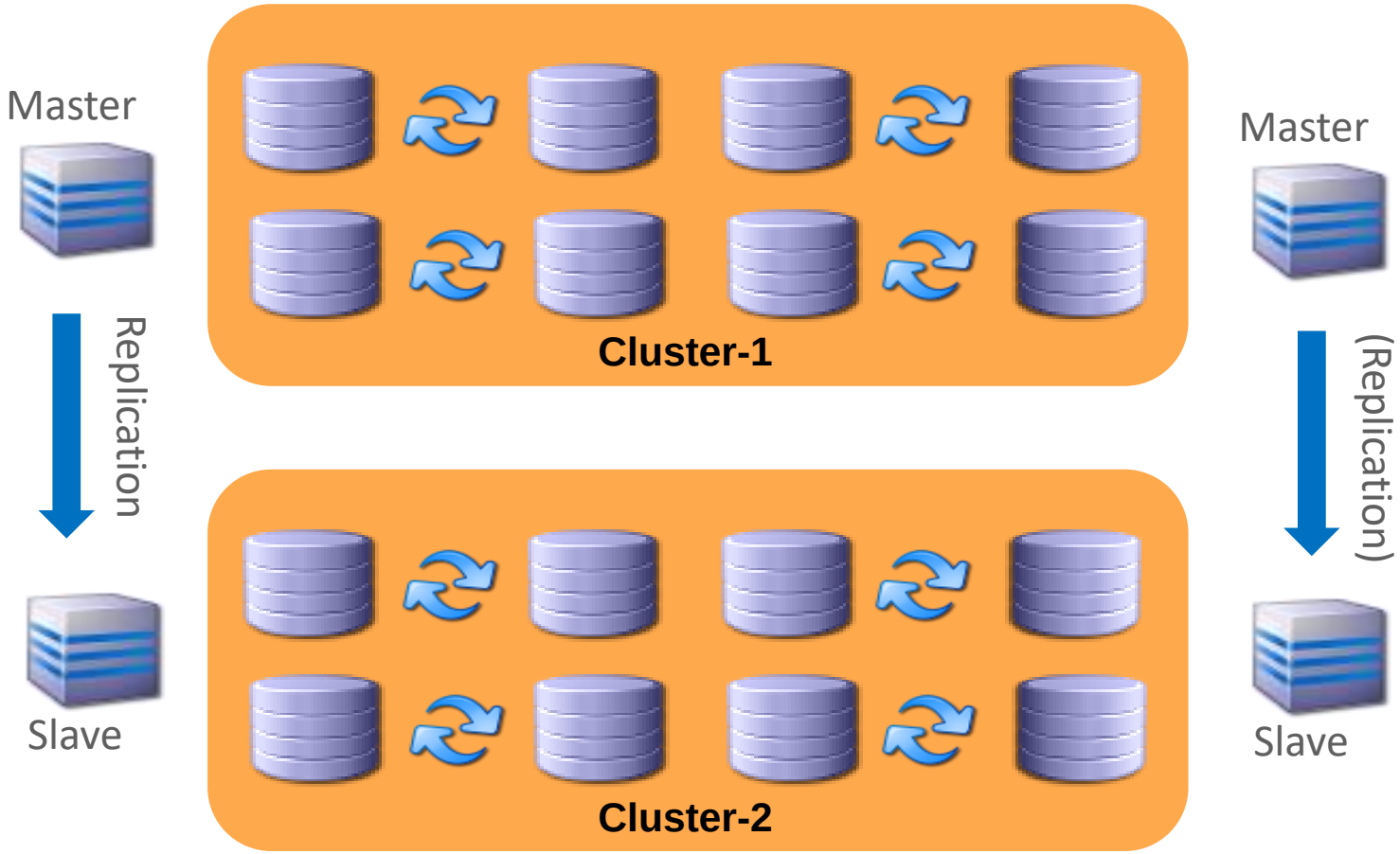
- High performance systems.
- 8 server setup
- Max number of data nodes is 48.
- Max 1100M QPM and 110M UPM.
- Size of DB limited by 4xRAM (RAM of smallest server).

Architectures
Storing Hot & Cold data



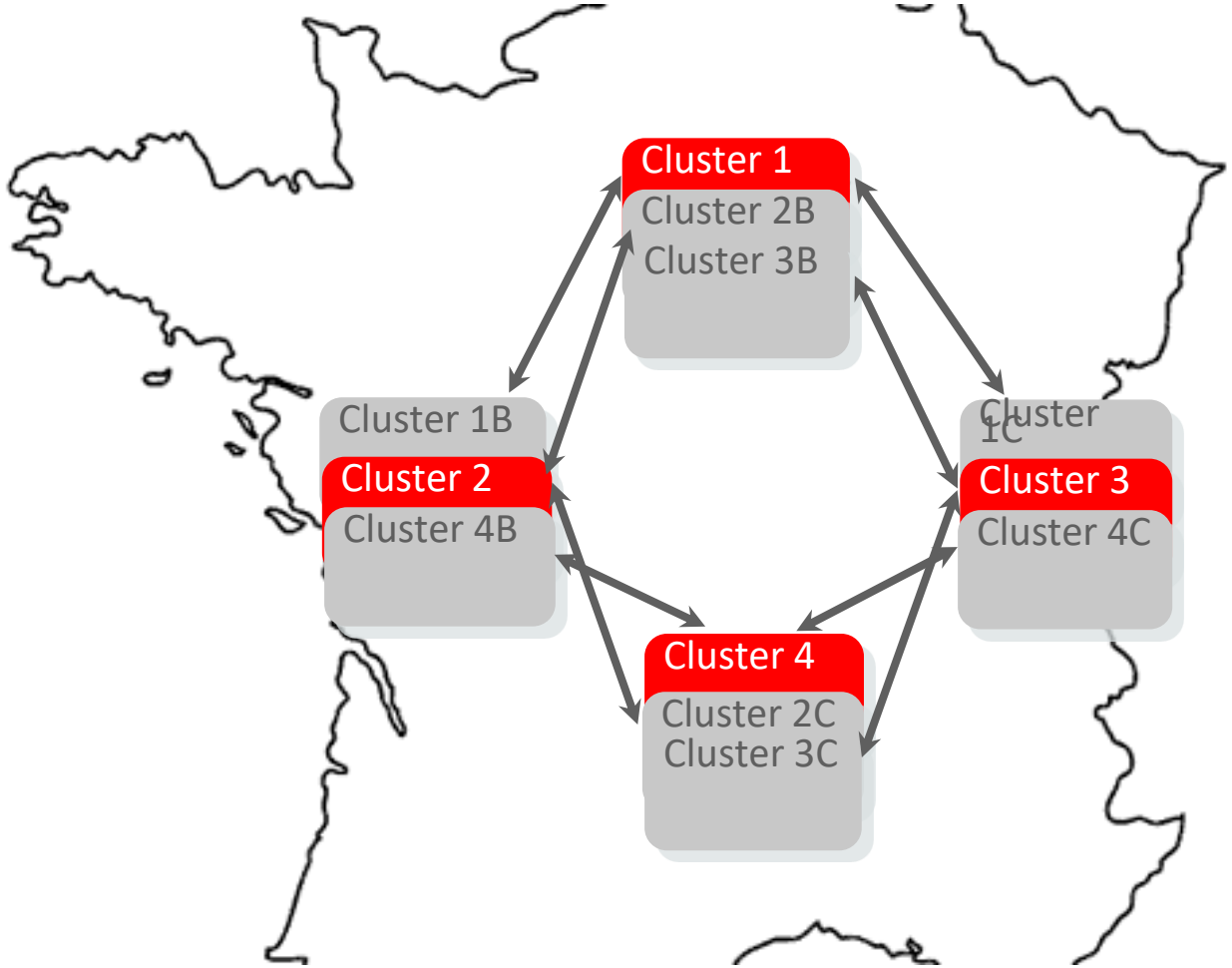
- Replicate to InnoDB for cold data.
- Full dataset in InnoDB (slave).
- Only “HOT” data in cluster low latency operations.
- Run reports/statistics on disk based database InnoDB.
- Cost effective!

Architectures
Geo redundancy



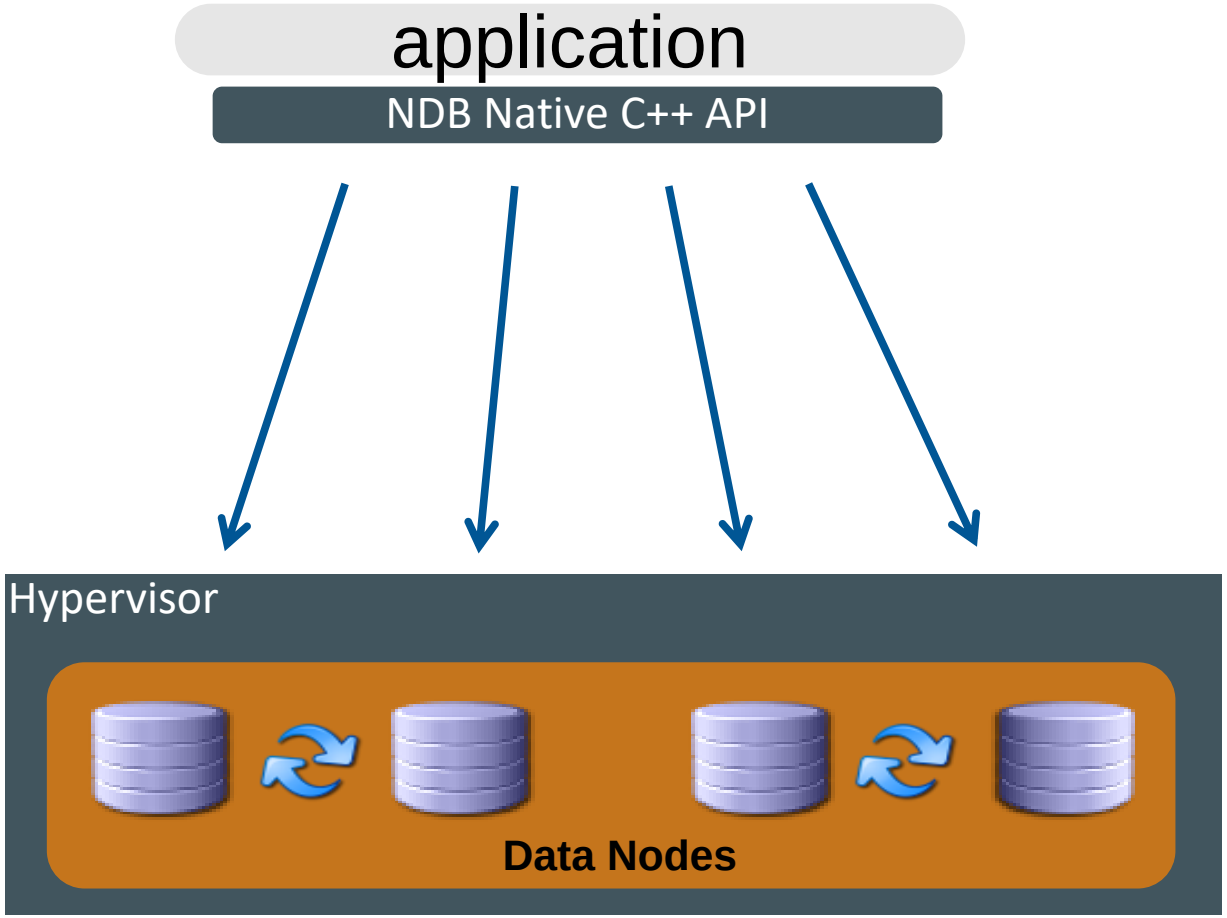
- Mission critical high performance applications.
- Geographic redundancy between data centers or continents.
- Active/passive or Active/Active

Architectures
Partition your data for low latency



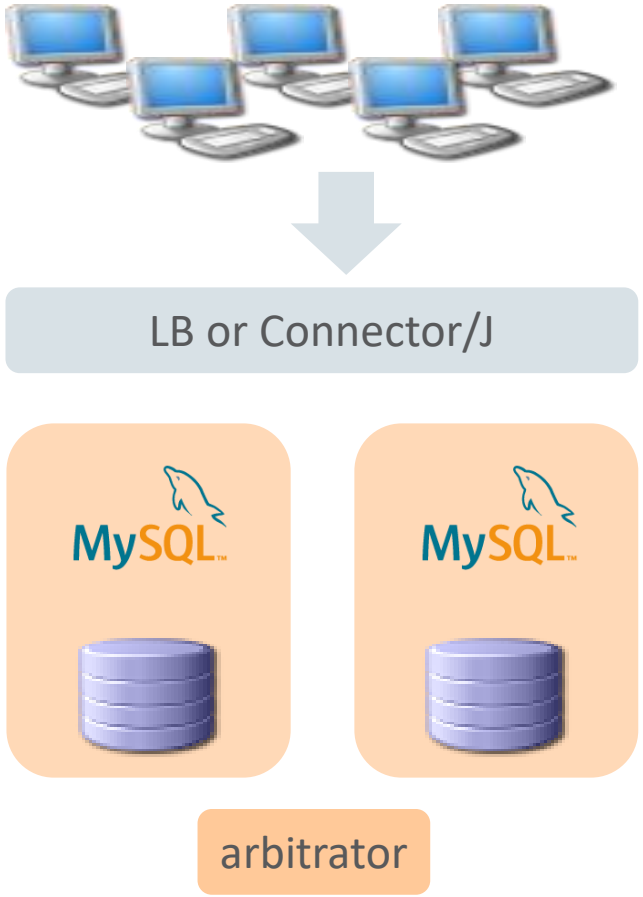
- Partition the subscribers across multiple clusters, distributed by country/region to optimize low latency access.
- Each sub-cluster is replicated for High Availability.
- Active/passive or Active/Active.

Architectures
Virtual environments



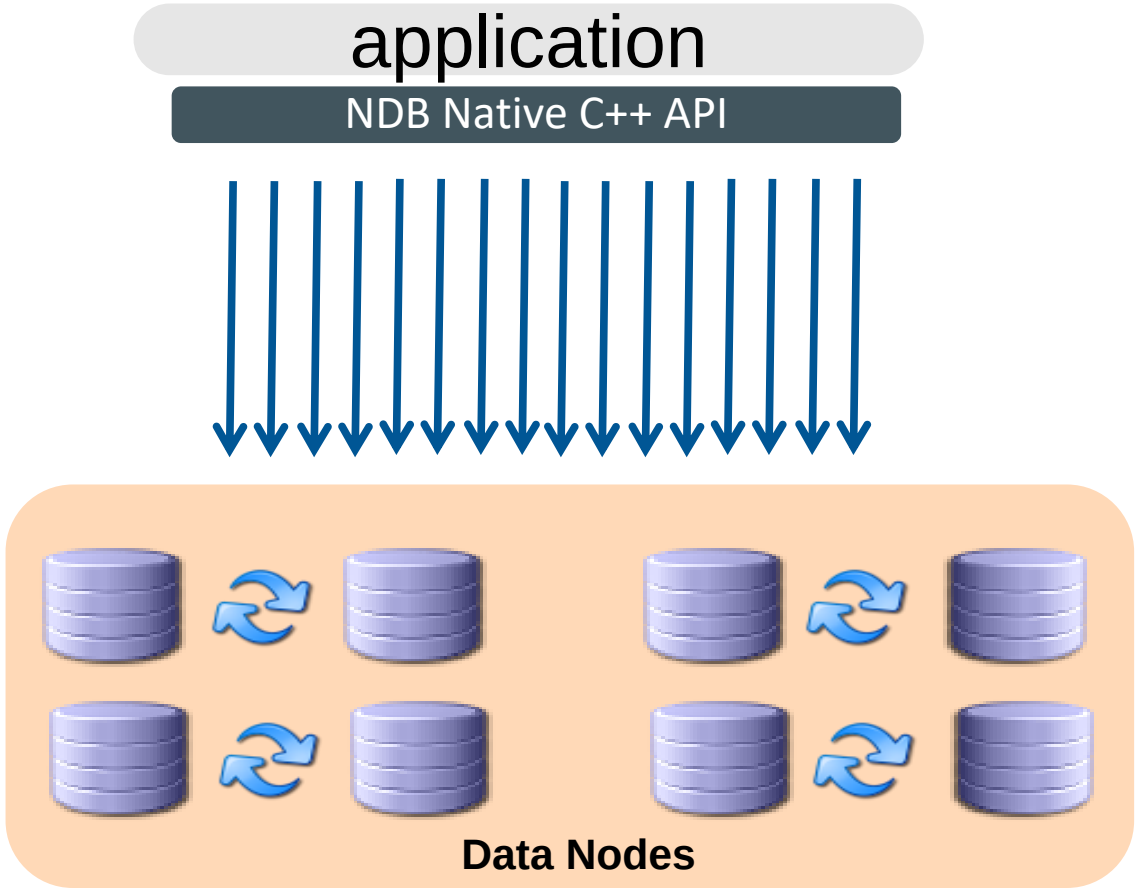
- Supported as of MySQL Cluster 7.2
- Use dedicated resources for data nodes in same node groups.
- Be aware of SPOFS.
- Mostly smaller operators using virtual environments today.

MySQL Cluster: SQL - Read Optimized HA setup



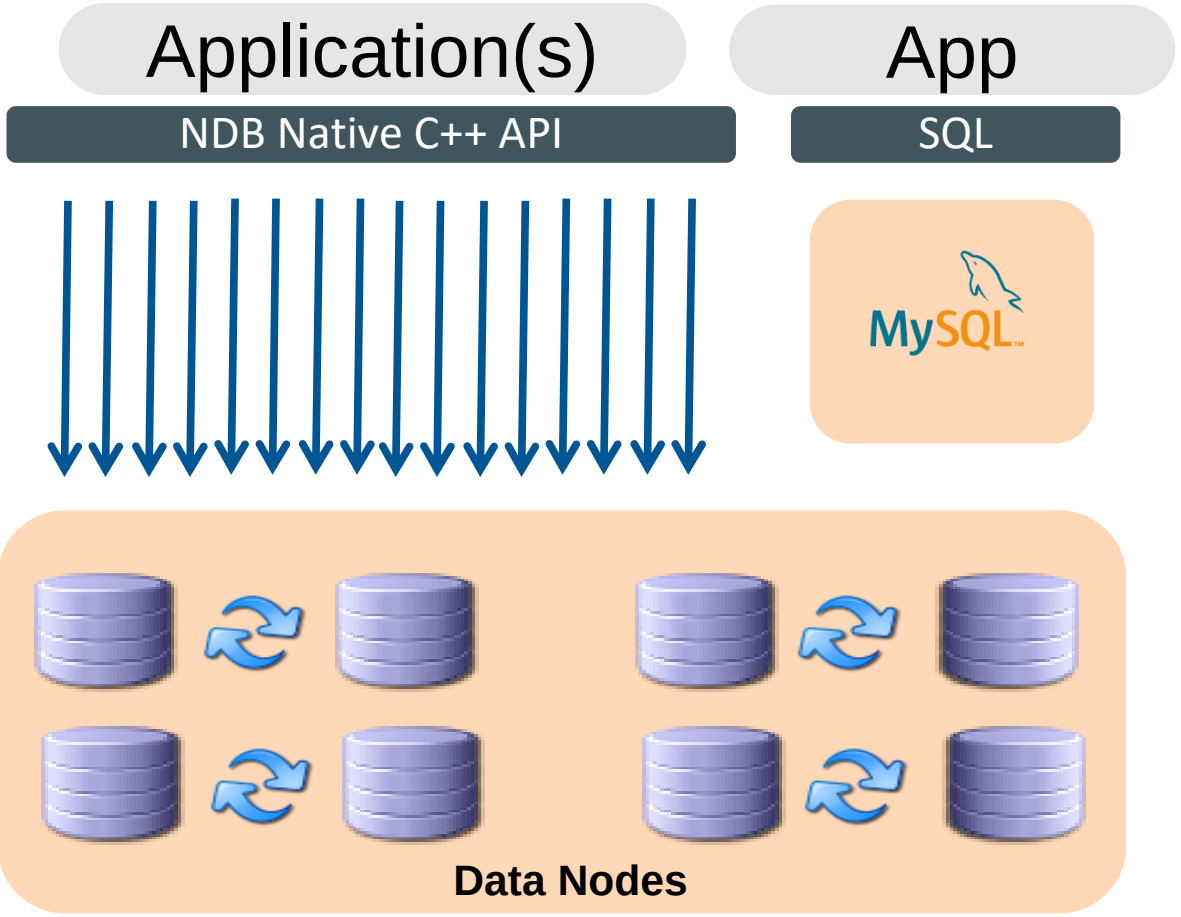
- Optimized 2 server HA setup, 3rd node with management node is needed for arbitration only.
- With ReadBackup all data is read/joined locally.
- MySQL API nodes use shared memory transporter (UseShm).
- MySQL nodes know who is local data node using configuration parameter ndb_data_node_neighbor.

MySQL Cluster: Key-Value store



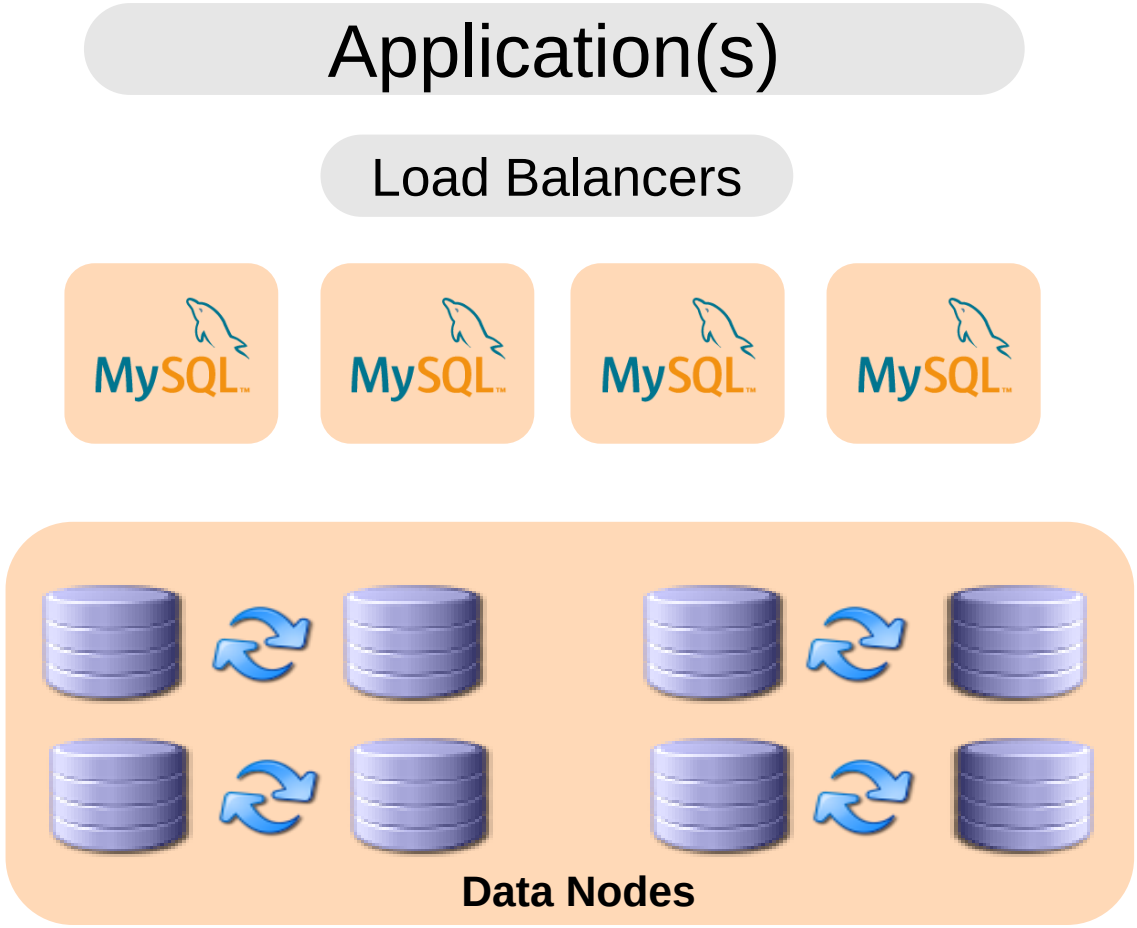
- High volume OLTP system.
- Linear scalability.
- Real-time response times.
- +50TB systems
- Use cases:
  - ✓ IoT
  - ✓ Financial data
  - ✓ Telco core network data

MySQL Cluster: Hybrid "New SQL"



- High volume OLTP system.
- Linear scalability.
- Real-time response times.
- +50TB systems
- Access via SQL:
  - ✓ Analytics
  - ✓ BI
  - ✓ Fraud

MySQL Cluster: New SQL



- High volume OLTP system.
- +50TB systems
- All access via SQL!
- Use cases:
  - ✓ IoT
  - ✓ On-line gaming
  - ✓ Trading or other financial data
  - ✓ Scalable SQL database

MySQL Cluster – Users & Applications

Telecoms

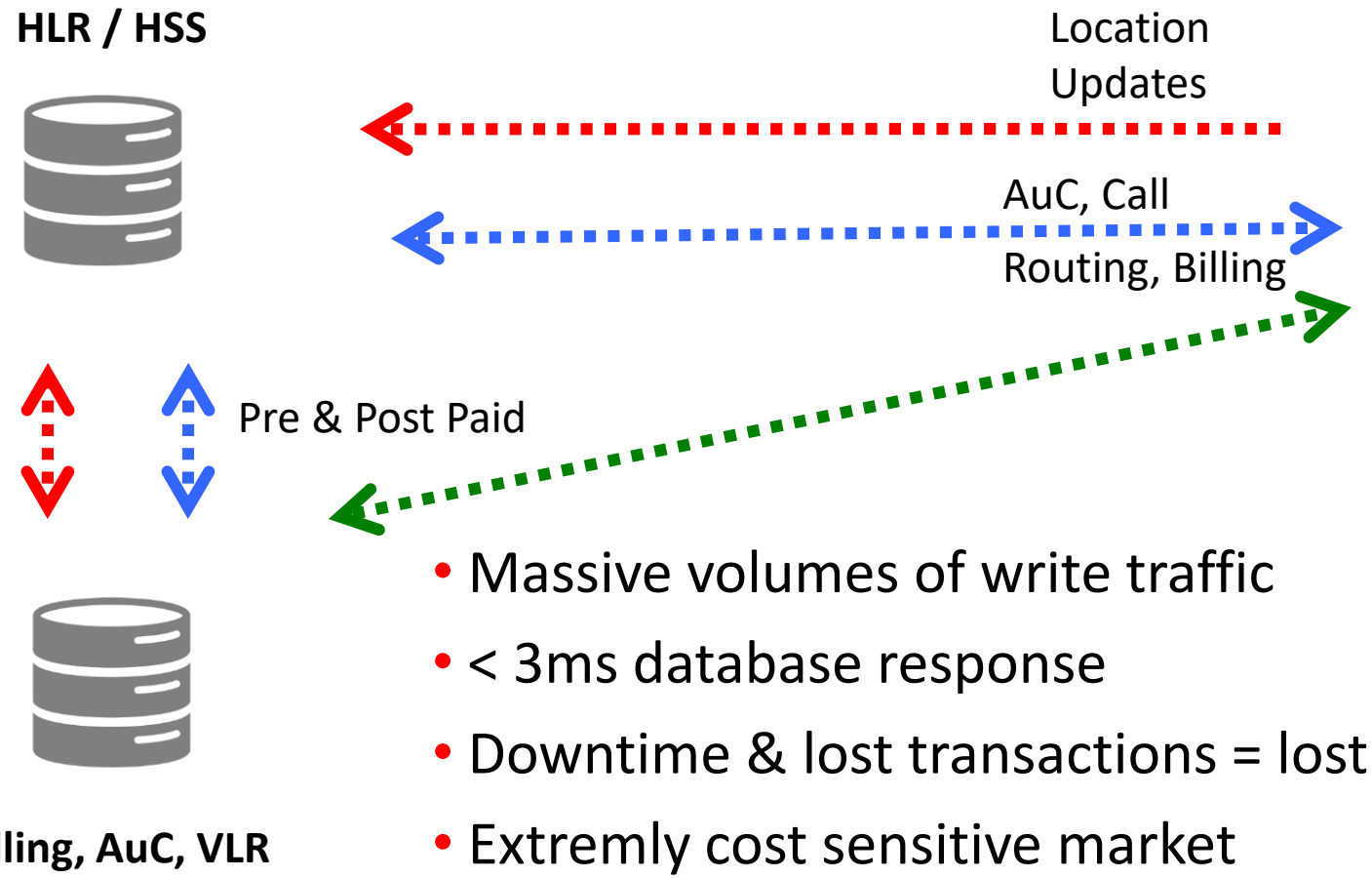
- Subscriber Databases (HLR / HSS)
- Service Delivery Platforms
- VAS: VoIP, IPTV & VoD
- Mobile Content Delivery
- Mobile Payments
- LTE Access

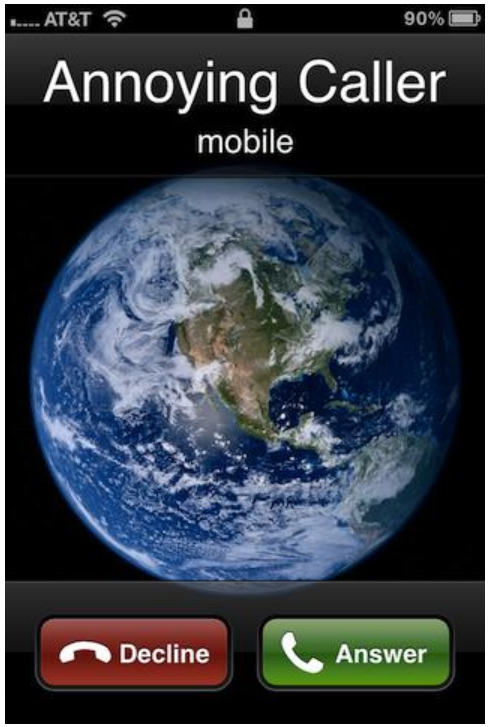
Web & Enterprise

- High volume OLTP
- eCommerce
- User Profile Management
- Session Management & Caching
- Content Management
- On-Line Gaming

http://www.mysql.com/customers/cluster/

No Trade-Offs: Cellular Network

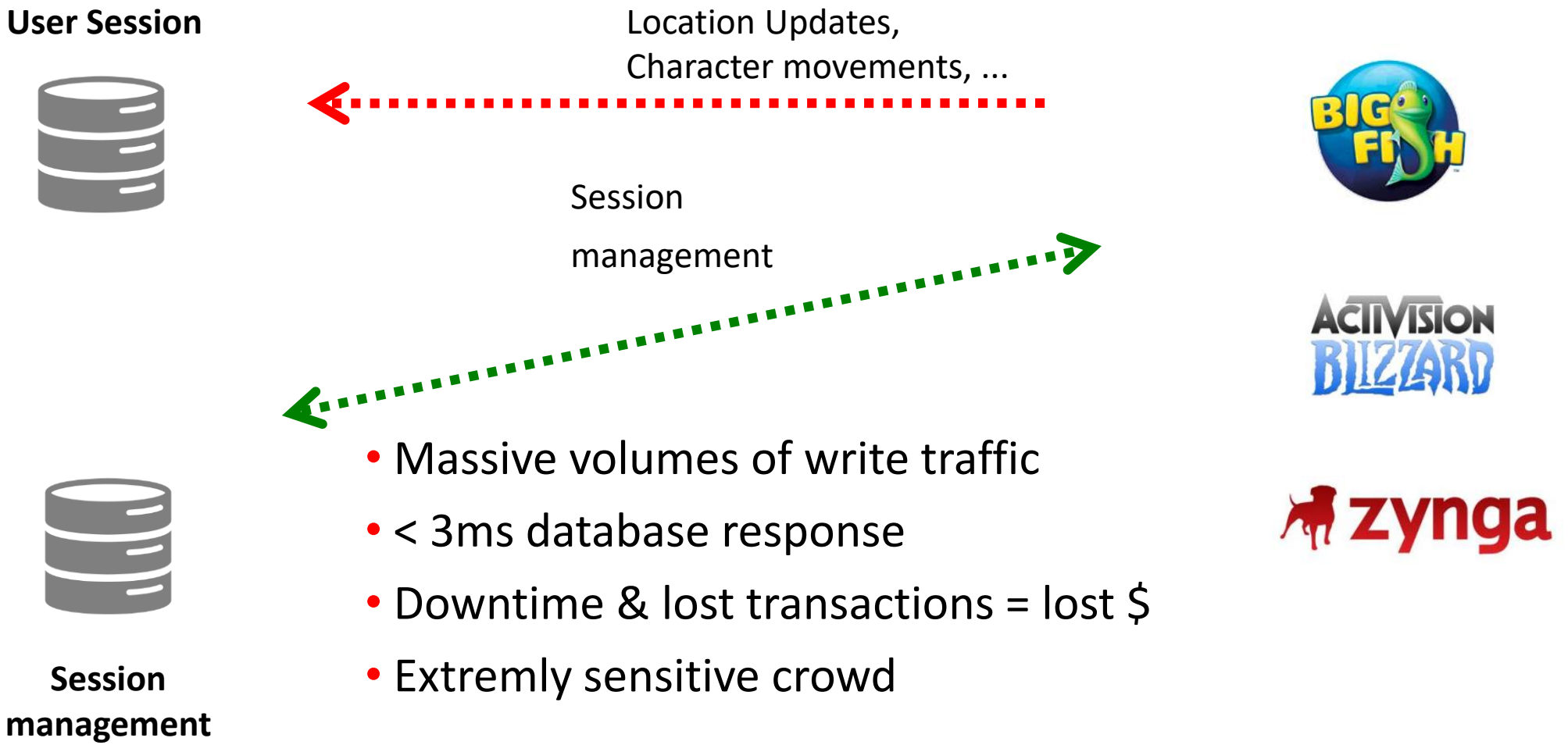




- Massive volumes of write traffic
- < 3ms database response
- Downtime & lost transactions = lost $
- Extremly cost sensitive market

MySQL Cluster in Action: http://bit.ly/oRI5tF

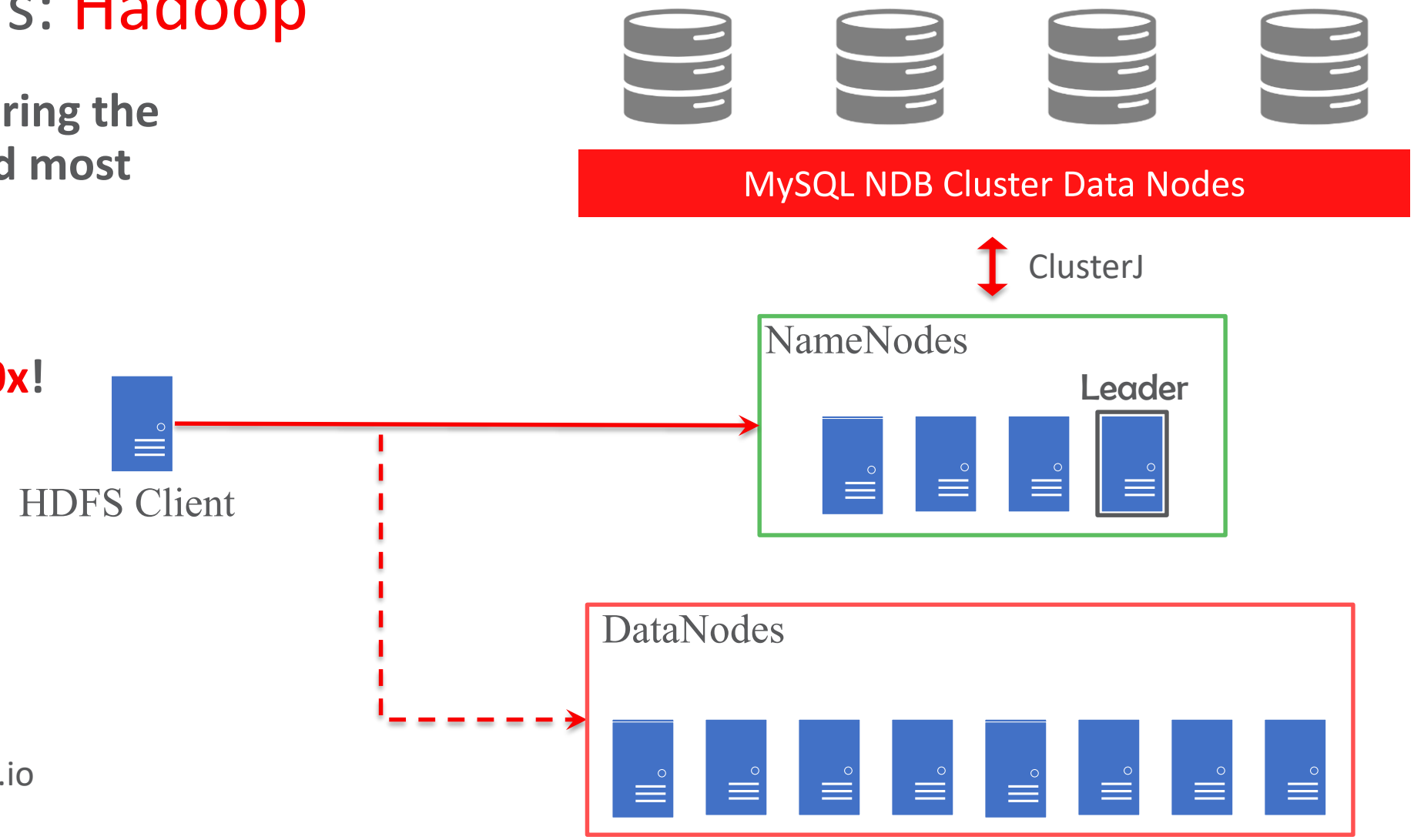
No Trade-Offs: Massive Parallel Online Games



No Trade-offs: Hadoop

NDB Cluster powering the world's fastest and most scalable Hadoop filesystem.

Scaling Hadoop 20x!



Program Agenda

1. Introduction
2. Getting started: configuration/install/start/stop
3. Administration: backup/upgrade/logs/conf/sizing
4. Monitoring, surveillance and problem solving
5. Best practices, architectures and case studies
6. NDB 7.6 and what's new in NDB 8.0
7. Geo-replication

MySQL Cluster 7.6

GA

Capacity & Scaleout

- Re-design for Terabyte clusters
- Faster disk data recovery
- Faster start-up
- Balanced and faster backups

Improved SQL

- Faster Joins
- Parallel Join Execution

Management

- Parallel CSV import
- MySQL Enterprise Monitor integration
- Cluster Manager and Configurator Update

ORACLE

Cluster 7.6: New! parallel ndb import

- Native CSV loading
- Highly parallel
- Load regulation
- Resume function

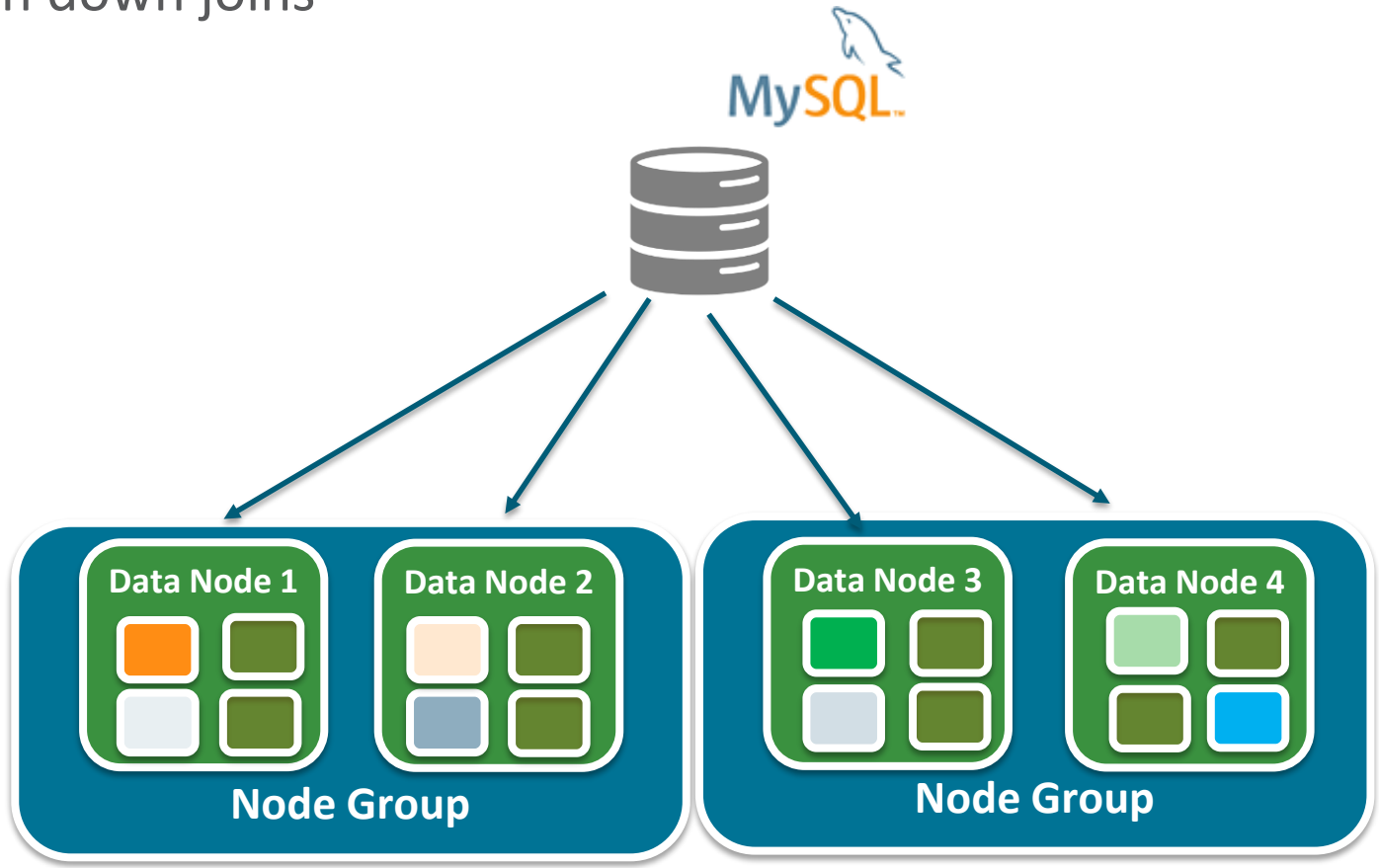
```
bo$ bin/ndb_import --rejects=3 mymusic genres.txt
                   cover.txt media.txt artists.txt

job-1 import mymusic.genres from genres.txt
job-1 [running] import test.t2 from genres.txt
job-1 [success] import test.t2 from genres.txt
job-1 imported 116660 rows in 0h0m1s at 83032 rows/s
job-1 rejected 2 rows (limit 3), see genres.rej
…
job-4 import mymusic.artists from artists.txt
job-4 [running] import test.artists from artists.txt
job-4 [success] import test.artists from artists.txt
job-4 imported 237862 rows in 0h0m3s at 78925 rows/s
job-4 rejected 0 rows (limit 3), see artists.rej


jobs summary: defined: 4 run: 4 with success: 4 with failure: 1
```

Cluster 7.6: New! Up to 50% faster JOIN performance

```
SELECT
        affiliates.uniquekey, affiliates_json, artists.name, artists.id, genres.name, genres.id,
        genres.priorityid, subgenres.id, subgenres.name, meta.name, meta.id, meta.json,
    formats.priority + formatclasses.priority + formattypes.priority,
        media.path, media.id, cover.id, cover.name, coverclasses.id, coverclasses.name,
        covertypes.id, covertypes.name
FROM
        affiliates, meta, media, cover, coverclasses, covertypes, artistsmap, artists,
        subgenresmap, subgenres, genres
WHERE
        artiststmap.id = meta.id AND artiststmap.id = artists.id AND subgenremap.id = meta.id AND
        subgenremap.id = subgenres.id AND subgenres.id = genres.id AND media.id = meta.id AND
        media.id = formats.id AND cover.coverclassid = coverclasses.id AND
        cover.canaddtocapability =  'Y' AND coverclasses.covertypeid = covertypes.id AND
        covertypes.covertypeid IN (2) AND (media.id IN (31, 8, 76)) AND
        affiliates.metaid = meta.id AND affiliates.affiliateid = '2';
```

MySQL Cluster - Parallel distributed joins

- Push down joins



- Joins are pushed down to data nodes
- Parallel cross-shard execution in the data nodes
- Result consolidation in MySQL Server

Cluster 7.6: New! Partial Local Checkpoint

- Complete local checkpoint re-design for Terabyte clusters
- Partial CPs only save a portion of the data memory and changes
- Reduced disk space (REDO / CP)
- Checkpoints deleted more promptly
- Reduce CP write rate for infrequent data changes
- Up to 6x faster Checkpointing

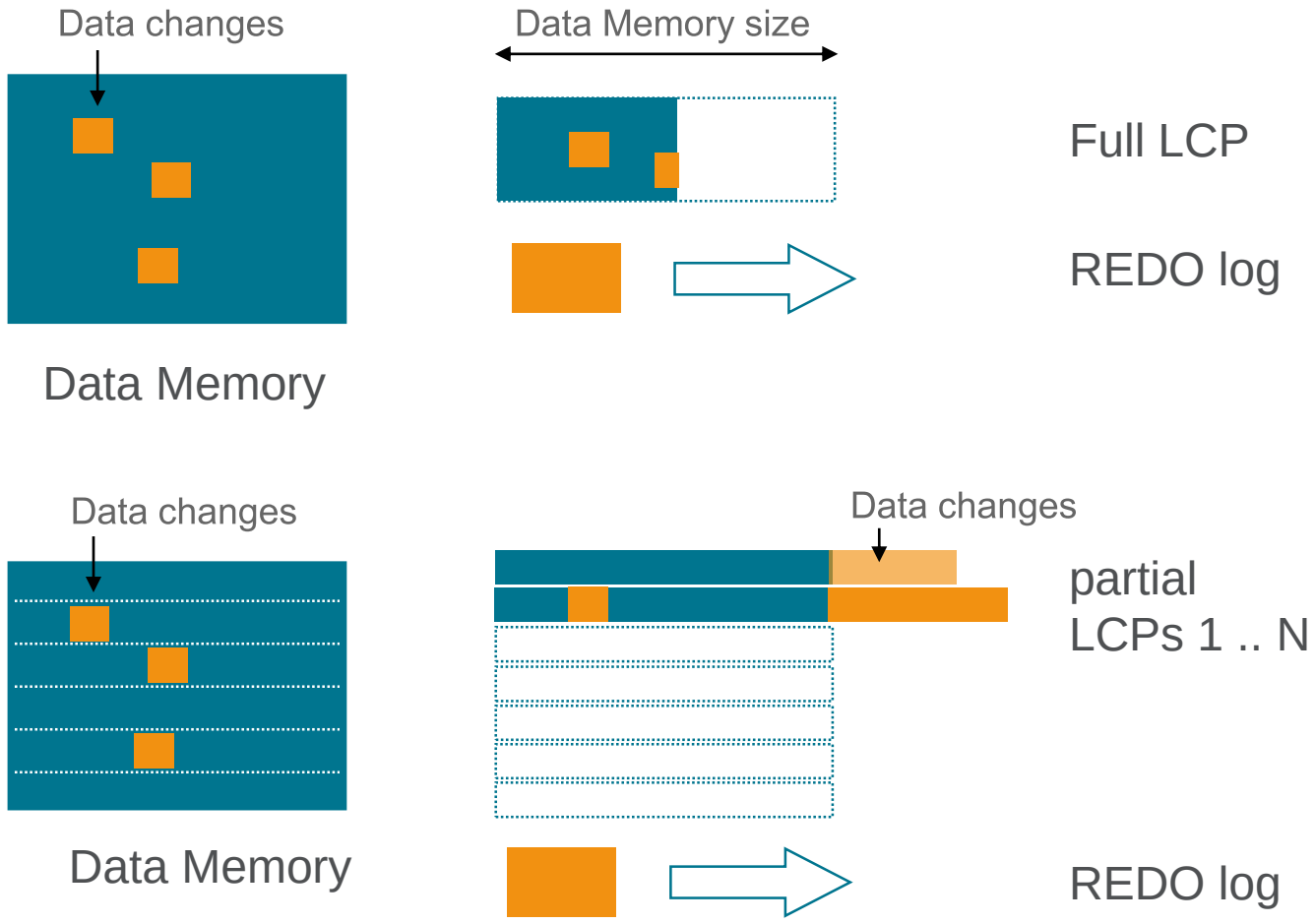
- Requires initial node restart when upgrading to it!

Cluster 7.6: New! Partial Local Checkpoint

Existing Local Checkpoints:

- Were full Checkpoints
- created large REDO logs

Cluster 7.6: New! Partial Local Checkpoint (LCP)



Cluster 7.6: New! Parallel UNDO log

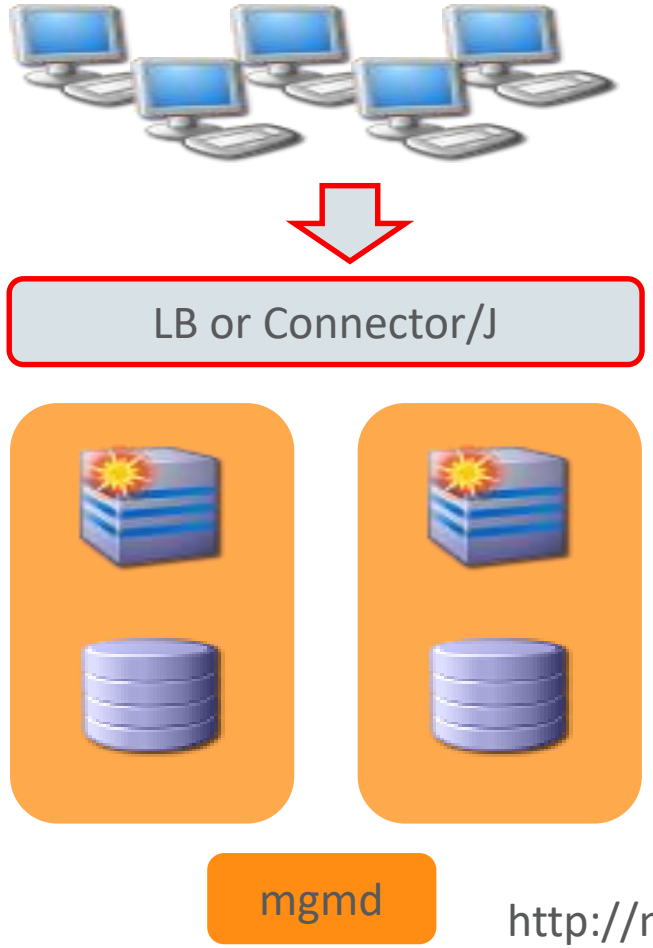
Up to 5x faster disk data recovery at start-up

- Disk data uses UNDO log during recovery
- UNDO log records are processed in parallel now
- Processed in multiple local data-manager instances

Cluster 7.6: New! Further improvements

- Equi-join improvements
- New improved disk data format
- Allow primary key schema changes in ndb_restore
- Shared memory connections
- Merging of IndexMemory and DataMemory

Cluster 7.6: Read Optimized HA setup



- Optimized 2 server HA setup, 3rd node with management node is needed for arbitration only.
- With ReadBackup all data is read/joined locally.
- MySQL API nodes use shared memory transporter (UseShm).
- MySQL nodes know who is local data node using ndb_data_node_neighbour.

http://mikaelronstrom.blogspot.com/2016/10/read-any-replica-in-mysql-cluster-75.html

MySQL Cluster 8.0 GA

GA

Capacity & Scaleout

- Larger clusters, up to 144 data nodes
- Balanced and faster backups

Improved SQL

- Based on MySQL 8
- Atomic DDL
- And all new other enhancements in MySQL 8

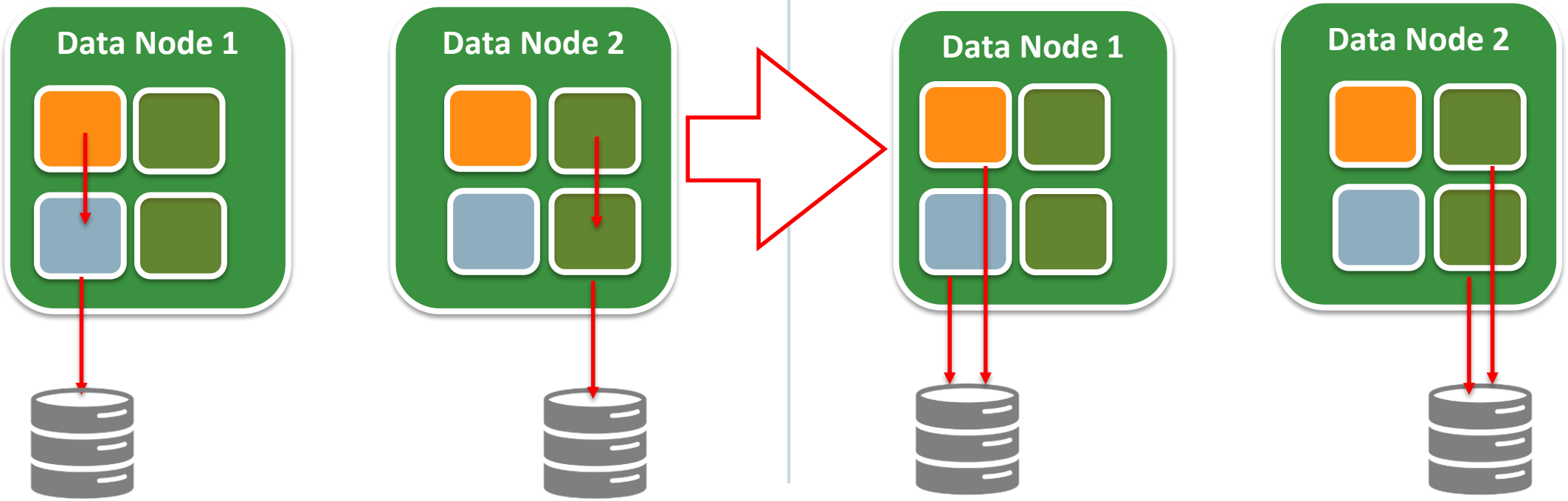
Management

- Poolification of memory for MaxNoOf-params

Cluster 8.0: Multi-threaded backup

Each data node doing own backup - one data manager handling all writing

Now each data manager handling own writing



Cluster 8.0: Dynamic resource allocation

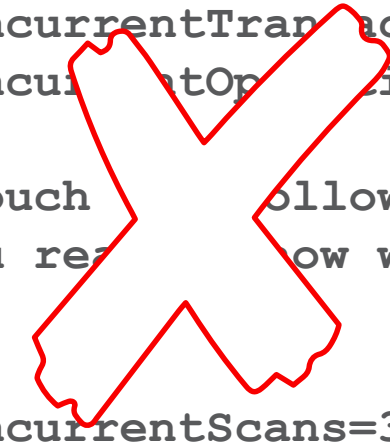
- config.ini today

```
config.ini

MaxNoOfConcurrentTransactions=70000
MaxNoOfConcurrentOperations=359500

# Don't touch the following parameter
unless you really know what you're
doing.

MaxNoOfConcurrentScans=200
MaxNoOfLocalScans=9000
```

Cluster 8.0: Dynamic resource allocation

- Transactional memory dynamically allocated from pool
- No more `MaxNoOfTransactions, MaxNoOfOperations, MaxNoOf…`
- Still can use old static allocation for highest level of performance
- More resources types to follow

```
config.ini

MaxNoOfConcurrentTransactions=90000
MaxNoOfConcurrentOperations=259200

# Don't touch following parameter
unless you realy know what you're
doing.

MaxNoOfConcurrentScans=300
MaxNoOfLocalScans=10000
```



Cluster 8.0: Dynamic resource allocation

- High-level gains
  - Lower config complexity
  - Fewer operational issues due to 'resource X exhausted’
  - Potential savings on memory due to over-configured resources.
  - Avoids hand-crafted config.ini for every new setup variant

Cluster 8.0: More

- Larger row sizes (14k -> 30k)
- Support for 4 replicas (max 2 today) of data
- Larger cluster:
  - Up to 128 data nodes
  - Total number of nodes 8192

Program Agenda

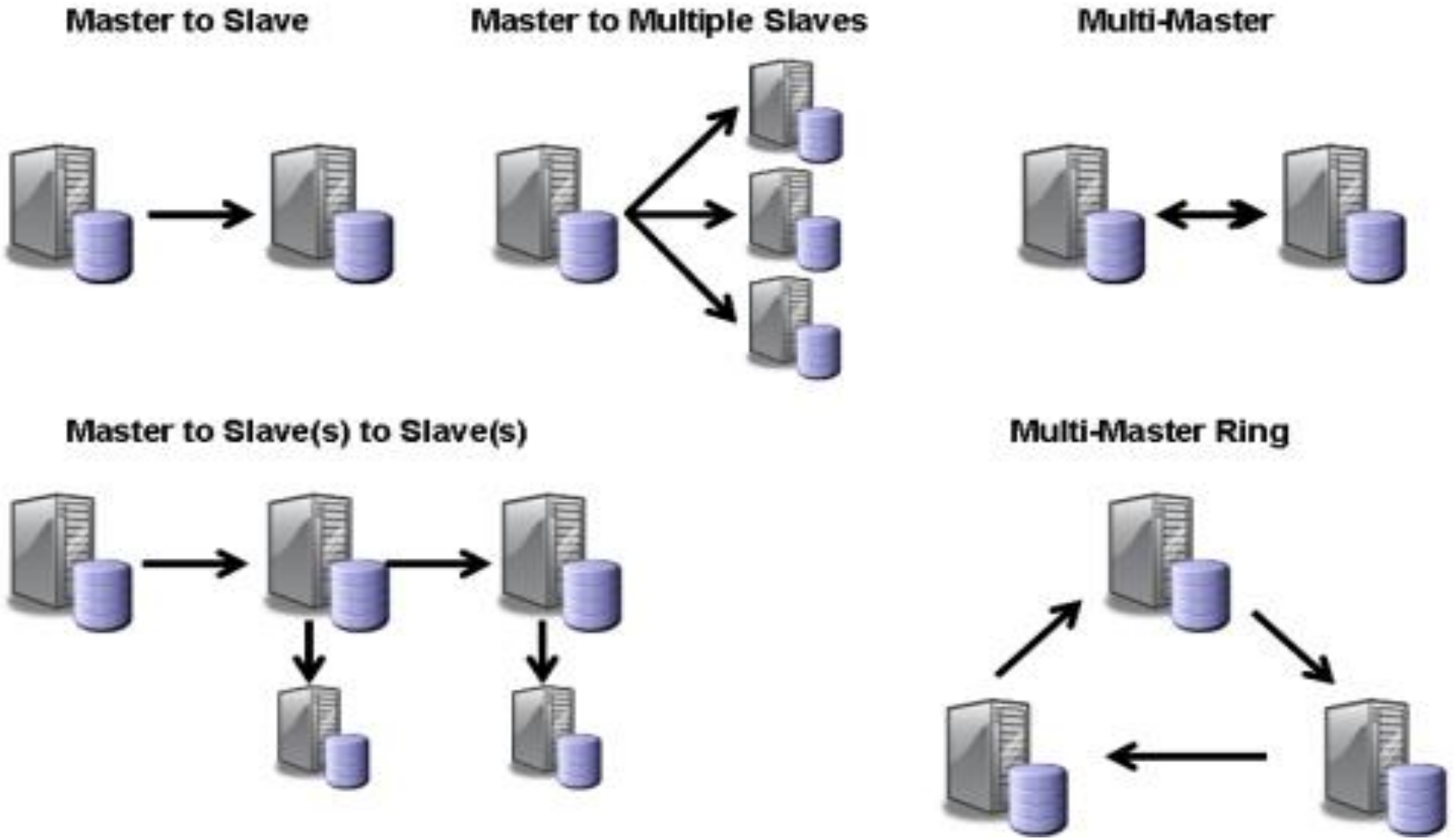
1. Introduction
2. Getting started: configuration/install/start/stop
3. Administration: backup/upgrade/logs/conf/sizing
4. Monitoring, surveillance and problem solving
5. Best practices, architectures and case studies
6. NDB 7.6 and what's new in NDB 8.0
7. Geo-replication

Geo-Replication

MySQL Cluster Replication
Basics

- Asynchronous replication
- External, not Internal.
- Geographical Replication
- MySQL Server / SQL Node – NDB Binlog Injector Thread
- Requirements:
  - 2 SQL nodes per Replication channel.
  - Unique server-id for all SQL nodes, Master & Slave
  - binlog-format=**ROW** or binlog-format=**MIXED**, depends on version of MySQL

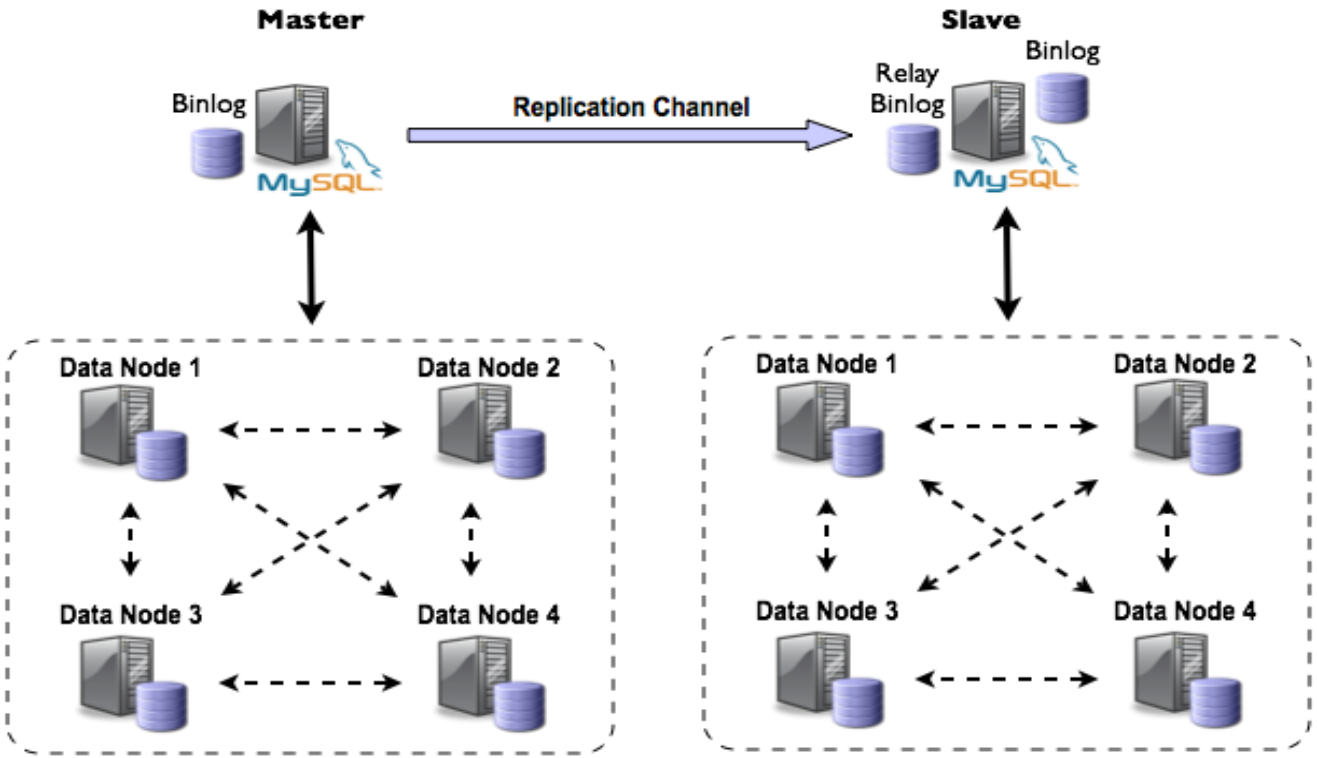
MySQL Cluster Replication
Classic Replication Scenarios



ORACLE

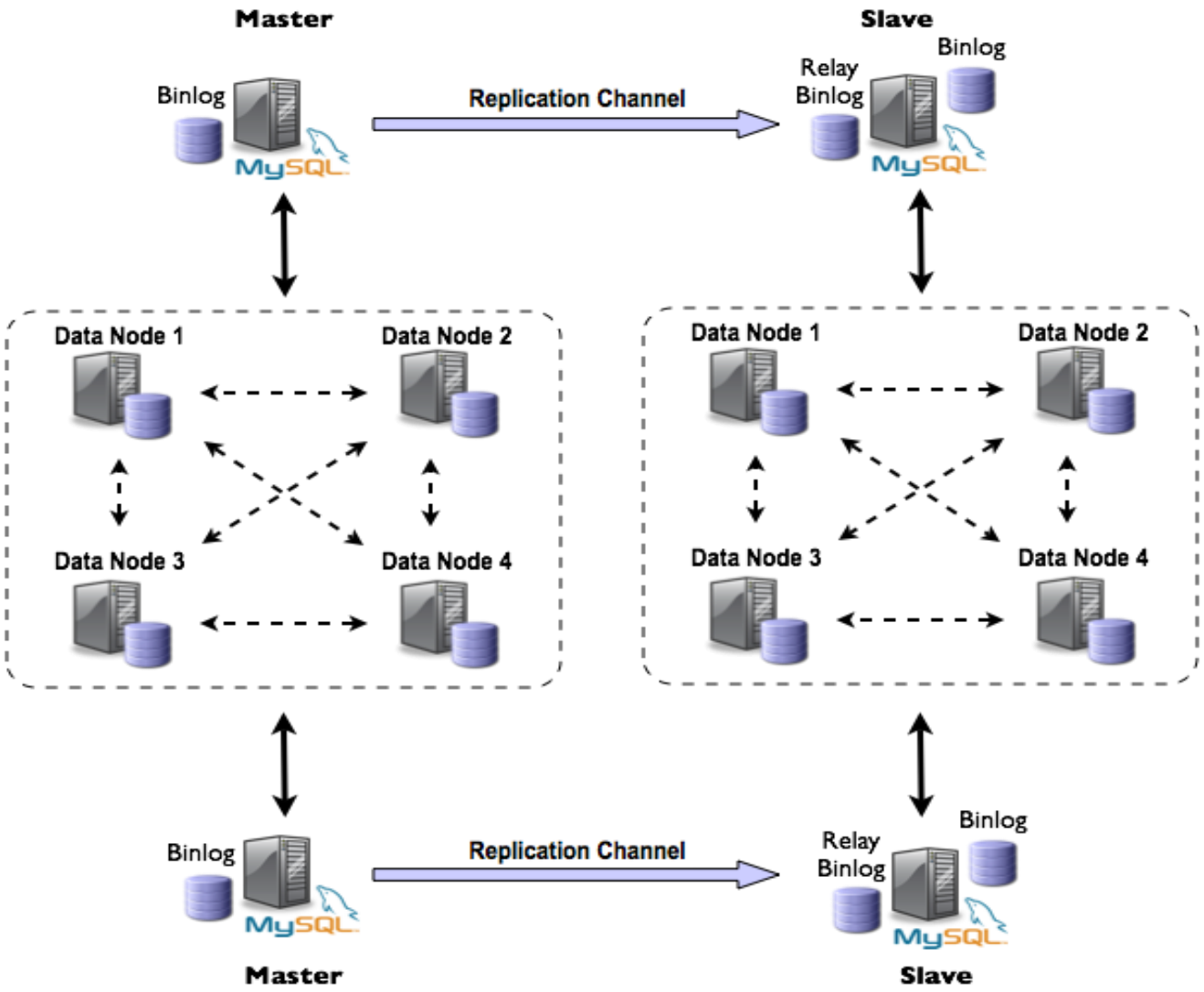
MySQL Cluster Replication

- Geographical Redundancy
  - Across data centers
  - Bring data closer to customers
- Load Balancing across clusters
  - Master cluster for writes
  - Slave cluster for reads
- Asynchronous Replication
  - Epocs
  - Slave batching
- Various Topologies
  - Master - Master
  - Master - Slave
  - Circular, ring, hub etc
- Conflict resolution



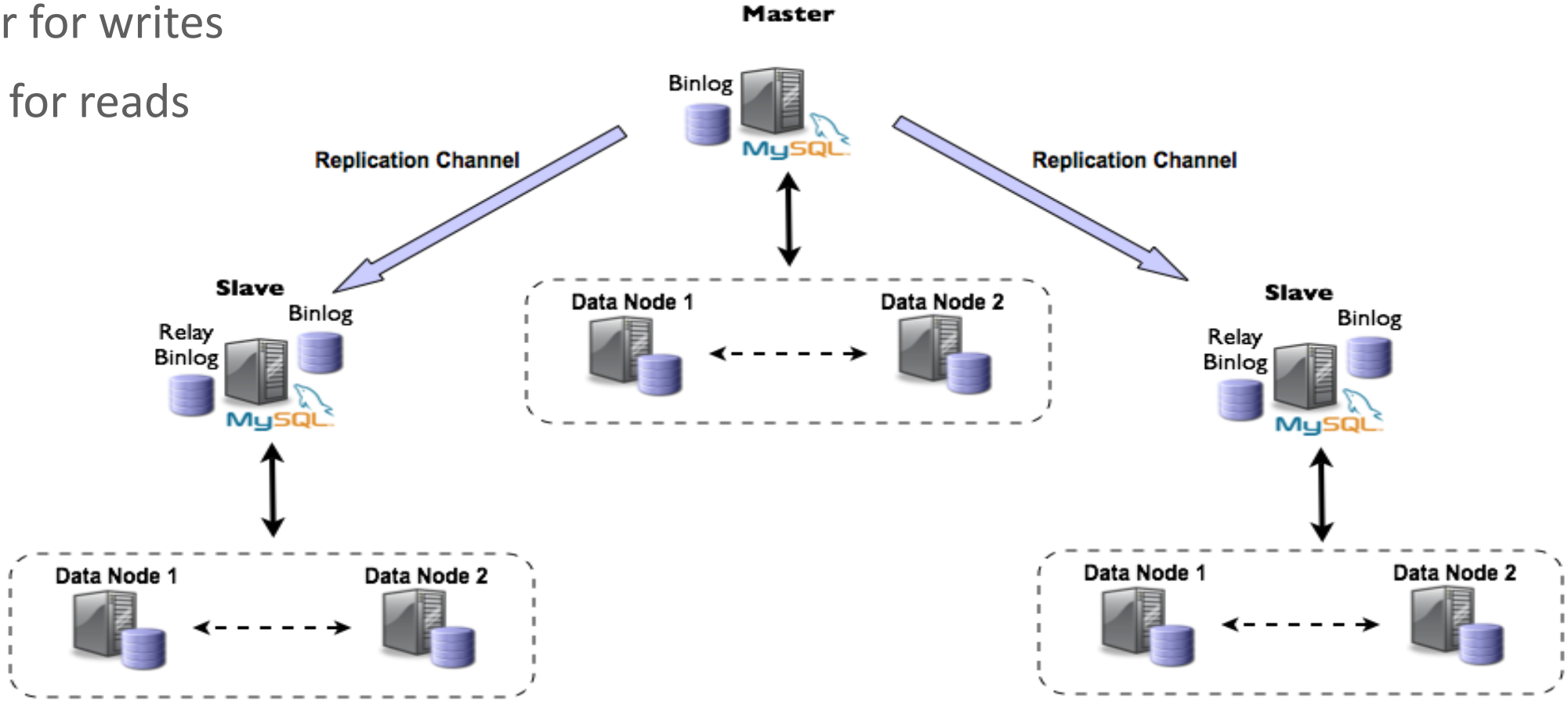
MySQL Cluster Replication
Master - Slave

- Two replication channels
  - No single point of failure



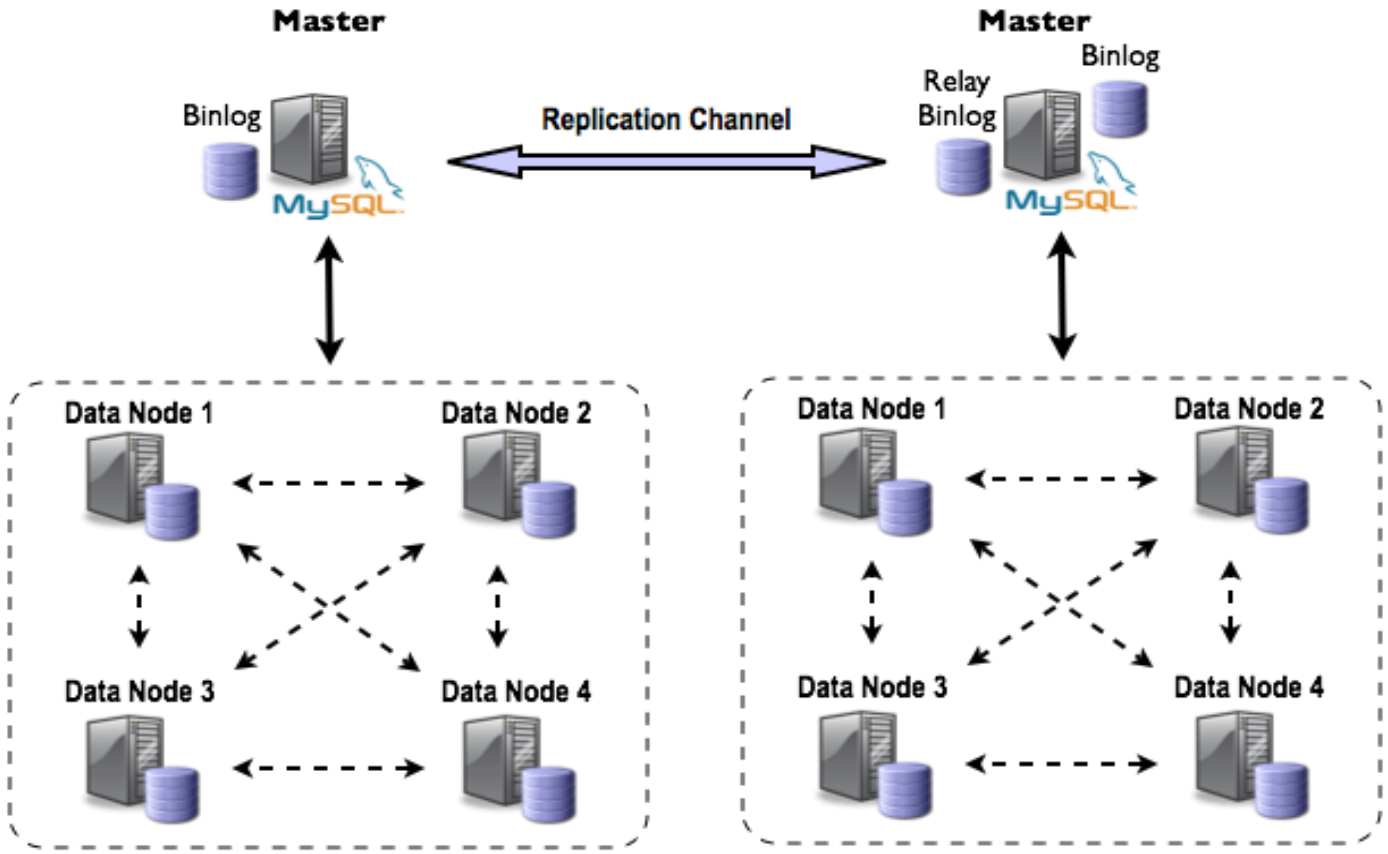
MySQL Cluster Replication
Examples

- Master for writes
- Slaves for reads



MySQL Cluster Replication
Multi-Master

- Multi-Master
- Conflict resolution



Conflict Resolution

2 types

- A user-defined resolution is permitted.
  - Types supported:
    - NDB$OLD(), NDB$MAX() & NDB$MAX_DELETE_WIN(), as a ‘timestamp’ column.
      - Row-by-row.
      - The Application holds the responsibility to make sure that the resolution column is correctly populated.
- Transaction based, comparing relative order of replication epochs:
  - NDB$EPOCH() & NDB$EPOCH_TRANS().
  - Does not make use of timestamps.

Replication hints

- Batch Updates:
  - `slave_allow_batching = ON|1` (normally, updates are applied as received, but this batches updates into 32Kb packets)
- Always have a user defined PK on all tables.

Questions?

Next Steps

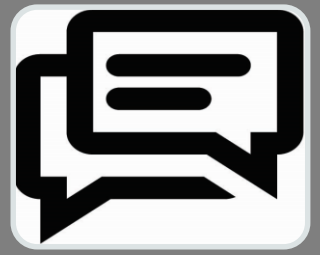
MySQL Cluster

Learn More

- www.mysql.com/cluster
- MySQL Curriculum: http://oracle.com/education/mysql

Try it Out

- dev.mysql.com/downloads/cluster/



Let us know what you think

- forums.mysql.com/list.php?25
- ted.wennmark@oracle.com
- keith.hollman@oracle.com

ORACLE

Thanks for attending!

ORACLE