

Writing Efficient Java Applications for MySQL Cluster Using NDB/J

Jess Balint <Jess.Balint@Sun.COM>
Monty Taylor <mordred@Sun.COM>
(Sun Microsystems)



APRIL 20–23, 09 SANTA CLARA, CA

Agenda

- Basics of MySQL Cluster
- Basics of NDB API
- NDB/Bindings Project
- API Details
- Converting SQL
- Other Features

Agenda

→ Basics of MySQL Cluster

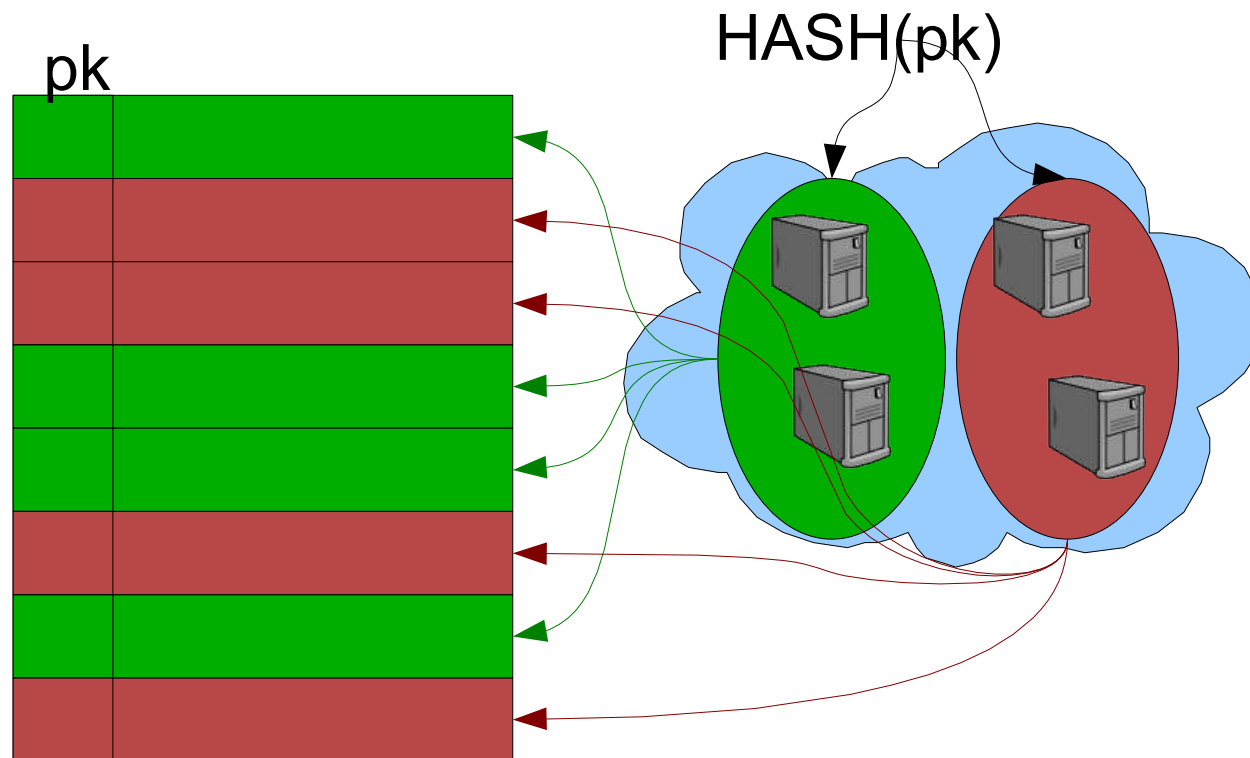
- Basics of NDB API
- NDB/Bindings Project
- API Details
- Converting SQL
- Other Features

What is MySQL Cluster?

- A shared nothing clustered database
- Supported in MySQL by a storage engine component
- In-memory database, with support for non-indexed columns on disk
- Transactional with synchronous commit across cluster

Data Distribution

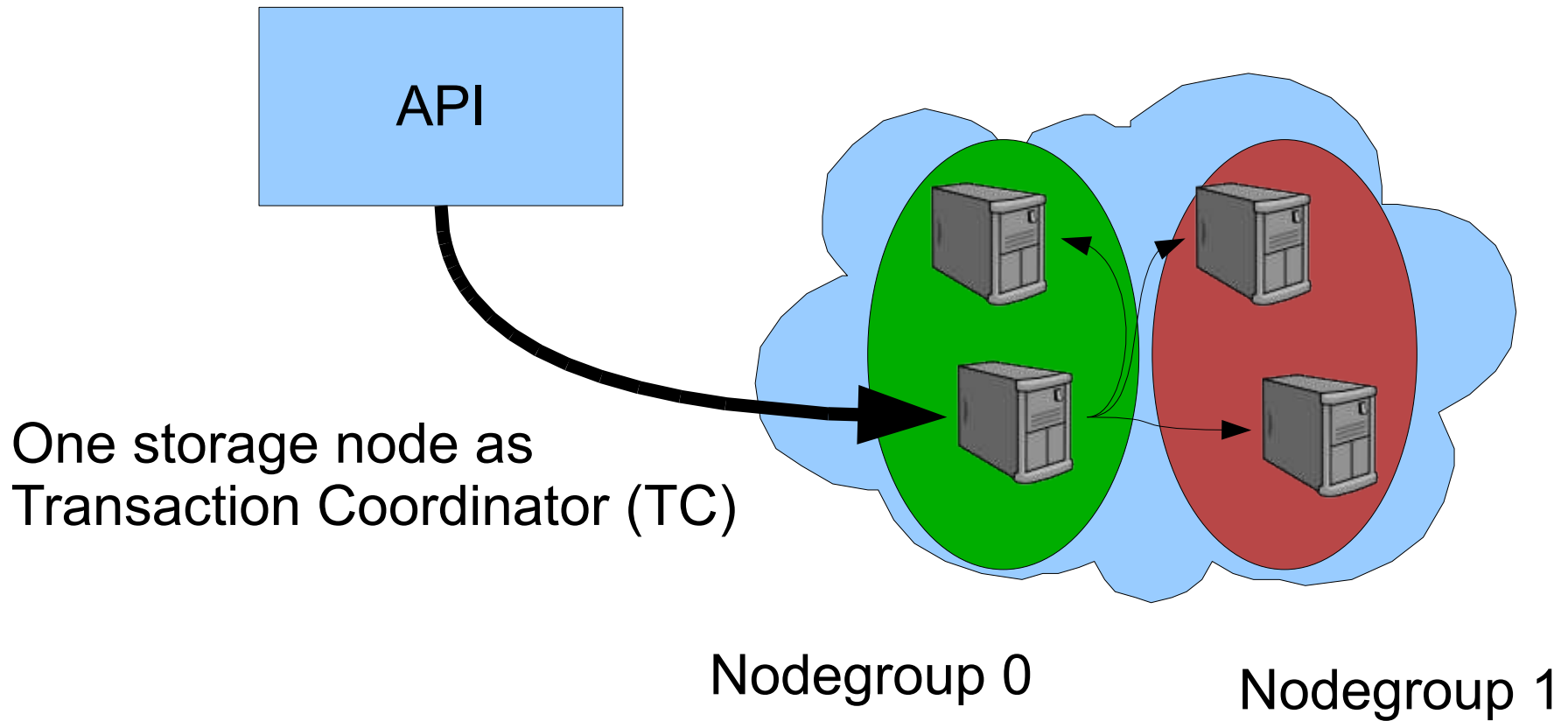
- Horizontal partitioning
- Data is replicated across all data nodes in a node group and partitioned across node groups



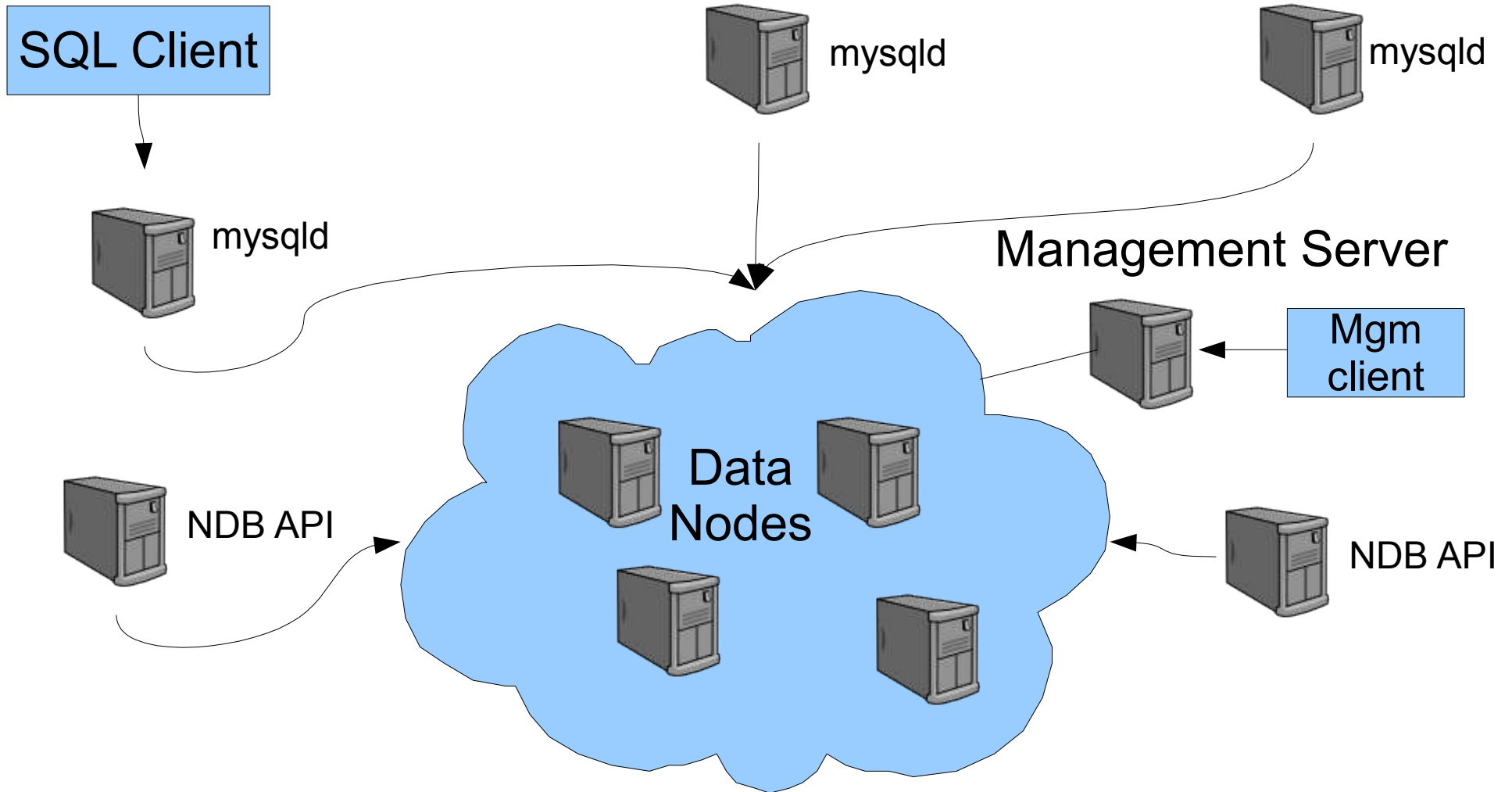
Data Access

- One data node selected as transaction coordinator (TC)
 - TC node can be chosen with TC hints
- API node sends all commands to TC
 - TC relays commands to other data nodes as necessary
- Any node can send data back to the API node

Data Access & TC



System Architecture



Agenda

- Basics of MySQL Cluster
- **Basics of NDB API**
- NDB/Bindings Project
- API Details
- Converting SQL
- Other Features

Basics

- Low-level, object oriented C++ API based on storage operations (all examples shown in Java)
- Main NDB API Classes
 - NdbClusterConnection
 - Ndb
 - NdbTransaction
 - NdbOperation (and subclasses)
 - NdbRecAttr (or NdbResultSet in NDB/Binding's Java wrapper)
- Ndb is a database handle, typically one per thread.

Connecting

- Create a connection object
 - NdbClusterConnection **conn** =
NdbClusterConnection.*create*("localhost");
- Connect to the management server
 - **conn.connect**(/* retries -> */ 5, /* delay -> */ 3,
/* verbose -> */ true);
- Wait until connection to data node(s) is established
 - **conn.waitUntilReady**(
/* timeout for first ndbd connection */ 30,
/* timeout for connection to all ndbds */ 0);

Database Handle

- Ndb object is a handle for a specific database
 - Ndb **ndb** = **conn.createNdb**(
 /* database */ "test",
 /* max # of tx on this handle */1024);

Transactions

- Every operation must be performed inside a transaction
- Any locks acquired are not released until the transaction completes
- Transaction is created from Ndb handle and used to create operations
 - `NdbTransaction trans = ndb.startTransaction();`
 - `NdbOperation op = trans.getInsertOperation("xy");`
 - `trans.execute(ExecType.Commit,
AbortOption.AbortOnError, true);`

Four Types of Operations

- Single row operations – read/write
 - NdbOperation - based on it's primary key value
 - NdbIndexOperation - based on it's key value in a given index
- Multiple row operations – read/write (no insert)
 - These operations can be limited with NdbScanFilter
 - NdbScanOperation - scan of the table
 - NdbIndexScanOperation – scan of an index
 - Can also be limited by specifying bounds on the index

Filtering in Operations

- Acquire the appropriate type from the transaction (see NdbTransaction)
 - `getUpdateOperation()`, `getSelectScanOperation()`, etc
- Single row operations
 - Specify key value with `equal*()` method (`equalInt`, `equalString`, etc)
- Multiple row operations
 - `setBound()` for index scan
 - `NdbScanFilter`

Accessing Data with Operations

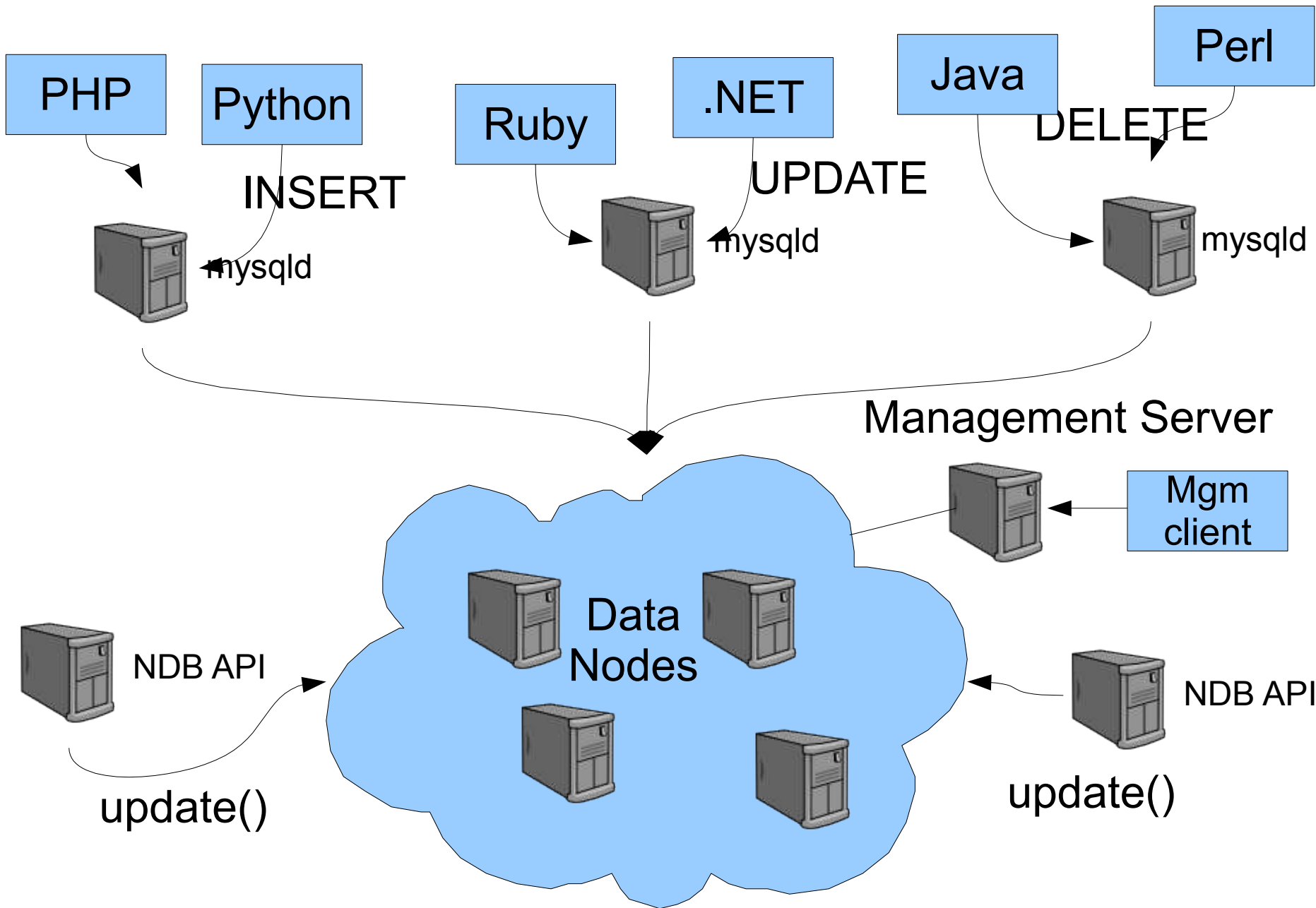
- Read data
 - readTuple(), readTuples()
- Write data
 - insertTuple() (NdbOperation only), updateTuple(), deleteTuple()
 - **NdbScanOperation**: lockCurrentTuple(), updateCurrentTuple(), deleteCurrentTuple()
- Most of these calls are handled automatically by acquiring the correct operation from NdbTransaction

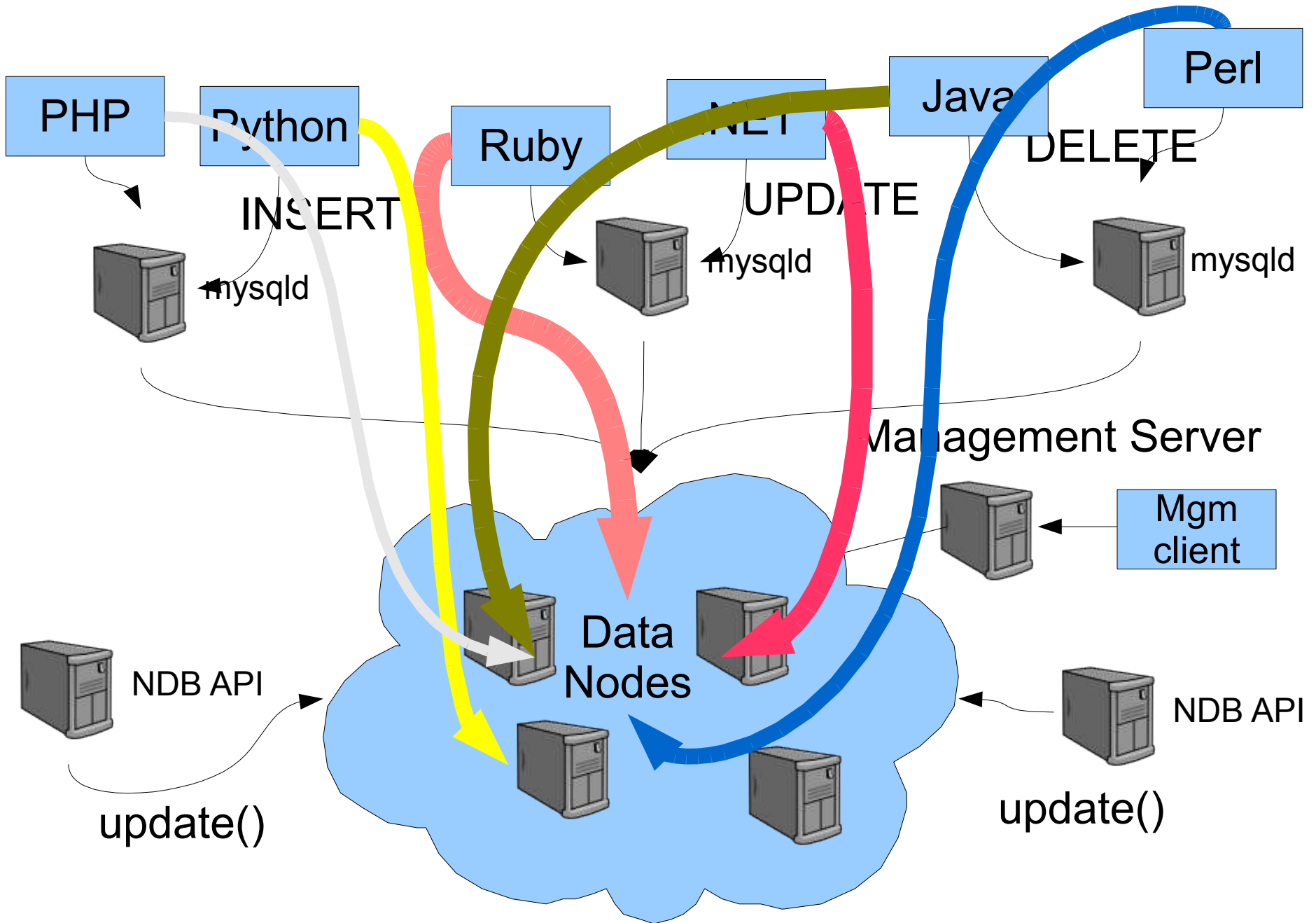
Agenda

- Basics of MySQL Cluster
- Basics of NDB API
- **NDB/Bindings Project**
- API Details
- Converting SQL
- Other Features

What is NDB/Bindings?

- Language specific wrappers of C++ NDB API
- Support for Java, Python, C#, Lua, Perl, PHP, Ruby, XML
 - Generated with SWIG and many language-specific customizations
- Currently only runs on UNIX/Linux
- Java wrapper referred to as NDB/J
 - Includes NdbResultSet class with wide range of data types





Complete Example

- Many examples in
`/java/src/java/com/mysql/cluster/ndbj/examples`

```
NdbClusterConnection conn =  
    NdbClusterConnection.create("localhost");  
conn.connect(5, 3, true);  
conn.waitForReady(30, 0);  
Ndb ndb = conn.createNdb("test", 1024);  
NdbTransaction trans = ndb.startTransaction();
```

Complete Example Cont'd

```
NdbOperation op =  
    trans.insertOperation("xy");  
op.equalInt("id", 1); /* declare PK value */  
op.setString("col1", "value A");  
op.setString("col2", "value B");  
trans.execute(ExecType.Commit,  
                AbortOption.AbortOnError, true);  
trans.close();
```

Reading Results

- C++ and other wrappers use NdbRecAttr
- In Java, we have NdbResultSet
- Similar usage to JDBC ResultSet
- One NdbResultSet per operation
 - Obtained by calling op.resultData()
- Not valid until transaction is executed
- Not compatible with lock transfers on scans
(more on this soon)

Agenda

- Basics of MySQL Cluster
- Basics of NDB API
- NDB/Bindings Project
- **API Details**
- Converting SQL
- Other Features

Error Handling

- NDB/J throws an exception if an NDB API call returns an error
- Everything derives from the checked NdbApiException class
 - Immediate subclasses include permanent and temporary errors
 - Concrete classes include
 - Permanent: constraint violations, internal error, schema error, schema object exists, no data found
 - Temporary: insufficient space, node recovery error, node shut down, timeout expired

Using Scan Filters

- Can be used on NdbScanOperation and NdbIndexScanOperation
- Example: SQL equivalent of $a = 1$ AND $b > 200$
 - NdbScanFilter **sf** = **scanOp.getNdbScanFilter()**;
 - **sf.begin**(Group.AND);
 - **sf.eq**("a", 1);
 - **sf.gt**("b", 200);
 - **sf.end**();

NdbResultSet

- `op.getValue()` must be called for each field retrieved (or `getBlob()` for BLOBs)
- `get*()` calls on `NdbResultSet` must match type of field (same as `NdbRecAttr`)

Executing and Committing Transactions

- Shown earlier when inserting a tuple:
 - `trans.execute(ExecType.Commit, AbortOption.AbortOnError, true);`
- We can also execute the transaction without committing by using `ExecType.NoCommit`
 - Used when reading data
- Arbitrary sets of operations can be included in a transaction

Scanning and Locks

- Shared locks are upgraded to exclusive locks when updating/deleting a tuple
- The transaction must be committed before reading the next set of tuples from the operation (when `nextResult()` returns 1/2, not 0)
- Locks can be transferred between transactions
 - `NdbOperation.lockCurrentTuple(NdbTransaction)`
 - The transaction owning the operation will takeover the lock from the transaction passed

Agenda

- Basics of MySQL Cluster
- Basics of NDB API
- NDB/Bindings Project
- API Details
- **Converting SQL**
- Other Features

Single Table Queries

- Using operations as shown before
- `SELECT A, B FROM TBL1 WHERE A = 1`
 - If A is a unique key, use `NdbOperation` or `NdbIndexOperation`
 - If A is not unique, `NdbIndexScanOperation`, or if unindexed, `NdbScanOperation`
- `WHERE A = 1 and B = 1`
 - If either column is indexed, use `NdbIndexScanOperation` with `NdbScanFilter`
 - Otherwise, add both to `NdbScanFilter`

Multiple Table Queries

- Join options
 - Nested loop join
 - Parallel scans on ordered indexes

Nested Loop Join

Outer Loop

Scan over first table

PK if available

Once for each row:

Inner/Nested Loop

Index scan for tuples
matching current key

Nested Loop Example, Pt 1

```
/* setup scan for outer loop, set bounds/filter if necessary */
```

```
NdbIndexScanOperation op1 =  
    trans.getSelectIndexScanOperation(  
        "PRIMARY", tablename, LockMode.LM_Exclusive,  
        ScanFlag.ORDER_BY, 10, 10);  
  
NdbResultSet rs1 = op1.resultData();  
  
rs1.getValue("key");  
  
rs1.getValue("other_field");  
  
trans.execute(ExecType.NoCommit, ....);
```

Nested Loop Example, Pt 2

```
while (rs1.next()) { /* begin outer loop */  
    keyValue = rs1.getInt("key");  
    NdbIndexScanOperation op2 =  
        trans.selectIndexScanOperation(index, tbl);  
    op2.setBoundInt("key", BoundType.BoundEQ,  
                    keyValue);  
    NdbResultSet rs2 = op2.resultData();  
    rs2.getValue("joined_field");  
    trans.execute(ExecType.NoCommit, ....);
```

Nested Loop Example, Pt 3

```
while (rs2.next()) { /* begin inner loop */  
    String joinedVal = rs2.getString("joined_field");  
    /* process tuple using key, other_field, and  
       joined field */  
} /* end inner loop */  
} /* end outer loop */  
trans.execute(ExecType.NoCommit, ....);  
trans.close();
```

Guidelines

- You can do arbitrary query filtering in the application, but try to push as much as possible to data nodes
- Using batching where possible to reduce record requests
- Aggregations can be done intuitively based on needs, eg. sum via accumulator

Agenda

- Basics of MySQL Cluster
- Basics of NDB API
- NDB/Bindings Project
- API Details
- Converting SQL
- **Other Features**

Atomic Increment

- Increment a field without fetching the tuple

```
trans = ndb.startTransaction();
```

```
NdbAtomicOperation aOp =  
trans.getAtomicUpdateOperation(tablename);
```

```
aOp.equalInt("id", 1);
```

```
aOp.increment("age", 2);
```

```
trans.executeCommit();
```

```
trans.close();
```

Asynchronous Transactions

- Create a transaction and operations as usual
- Instantiate custom callback class
- Call `trans.executeAsynchPrepare()` with callback handler
- Call `ndb.sendPollNdb()` to execute transaction asynchronously

BLOB Handling

- Blobs access via blob handle class NdbBlob
- Write blobs with `setNull()` or `setValue(byte [])`
- Read data
 - *getData()* - full blob value as byte array
 - *readData(byte [], long)* - piecewise retrieval into existing byte array
 - Also *getLength()* and *getNull()*

Events

- Notify application when data is updated
- Use NdbEventOperation
- Ndb.createEventOperation()
- Get NdbResultSet from operation
- Execute operation
- Ndb.pollEvents()
- Ndb.nextEvent()
- Get results from NdbResultSet

Next Steps

- Read through the API for classes you are using to become familiar with them
- Look at some other material
 - Johan Andersson, MySQL Conf 2007
 - Developing High Performance Applications using NDB API
 - Developing High Performance Applications using NDB/J
- Explore Other Features
 - TC hint, MGM API