Sharding and Scale-out using MySQL Fabric

Mats Kindahl (mats.kindahl@oracle.com)
Principal Senior Software Developer

Safe Harbor Statement

Presentation Outline

- Introducing MySQL Fabric
- Architecture for High-Availability
- Connecting to a MySQL Fabric Farm
- Architecture for Sharding
- Summary and Closing Remarks

MySQL Fabric

An extensible and easy-to-use framework for managing a farm of MySQL servers supporting high-availability and sharding

ORACLE

What does all that mean?

- "Farm" Management System
  - **Farm:** Collection of components
  - Distributed Framework
- Framework
  - Procedure execution
  - State store
  - Transaction Routing
- Extensible
  - Extensions are first-class
  - High-Availability Groups
  - "Semi-Automatic" Sharding
- Written in Python
- Latest Release 1.4.2
  - Release Candidate
- Open Source
  - **You** can participate
  - Suggest features
  - Report bugs
  - Contribute patches
- MySQL 5.6 is focus

MySQL Fabric: Goals & Features

- Decision logic in connector
  - Eliminate one network hop
  - Reducing network load
- Connector API Extensions
  - Support Transactions
  - Support full SQL
- Load Balancing
  - Read-Write Split
  - Round-robin

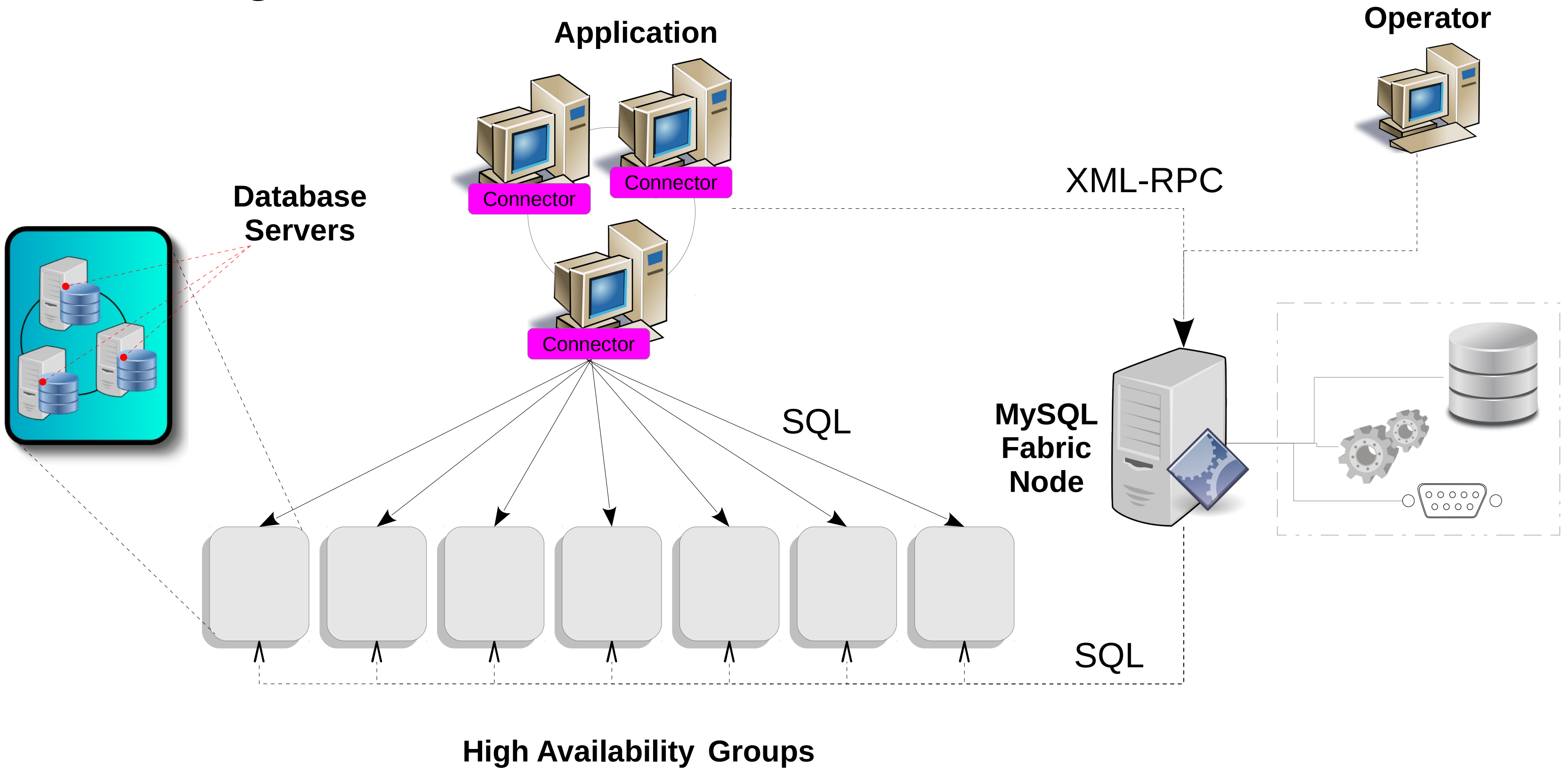
- Multi-Table Sharding
- Sharding Functions
  - Range
  - (Consistent) Hash
- Shard Operations
  - Shard move
  - Shard split
- Global Updates
  - Global tables
  - Schema updates

ORACLE

A Brief History of MySQL Fabric

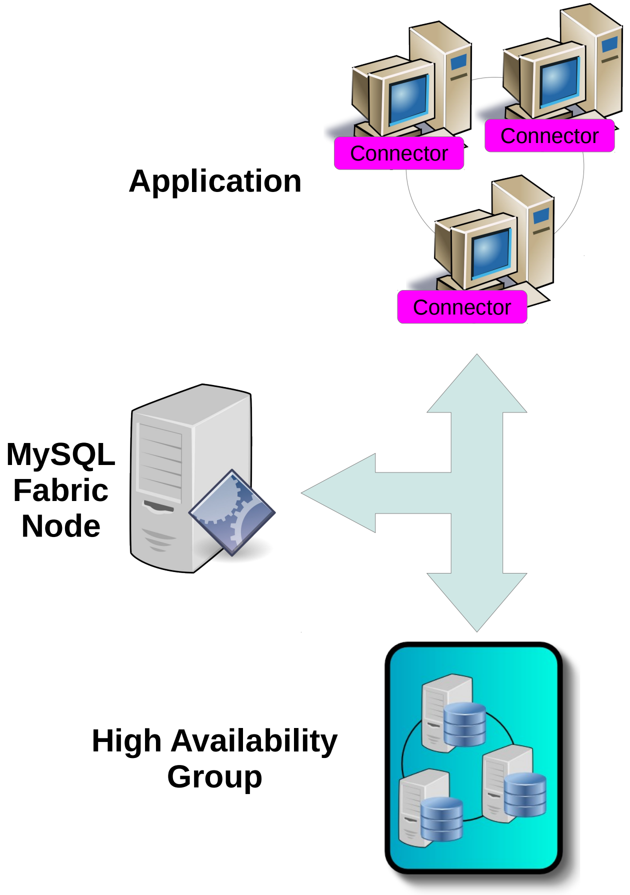
- MySQL Fabric 1.4.0
  - September, 2013
  - First public release
  - High-Availability Groups
  - Slave Promotion
  - Range and Hash Sharding
  - Shard move and split
  - Connector/Python Support
  - Connector/J Support
  - Connector/PHP Support
- MySQL Fabric 1.4.1
  - December, 2013
  - Alpha release
  - Sharding refactorings
- MySQL Fabric 1.4.2
  - April, 2014
  - Release Candidate
  - Distributed Failure Detection
  - Credentials
  - Weighted Round-Robin

Birds-eye View

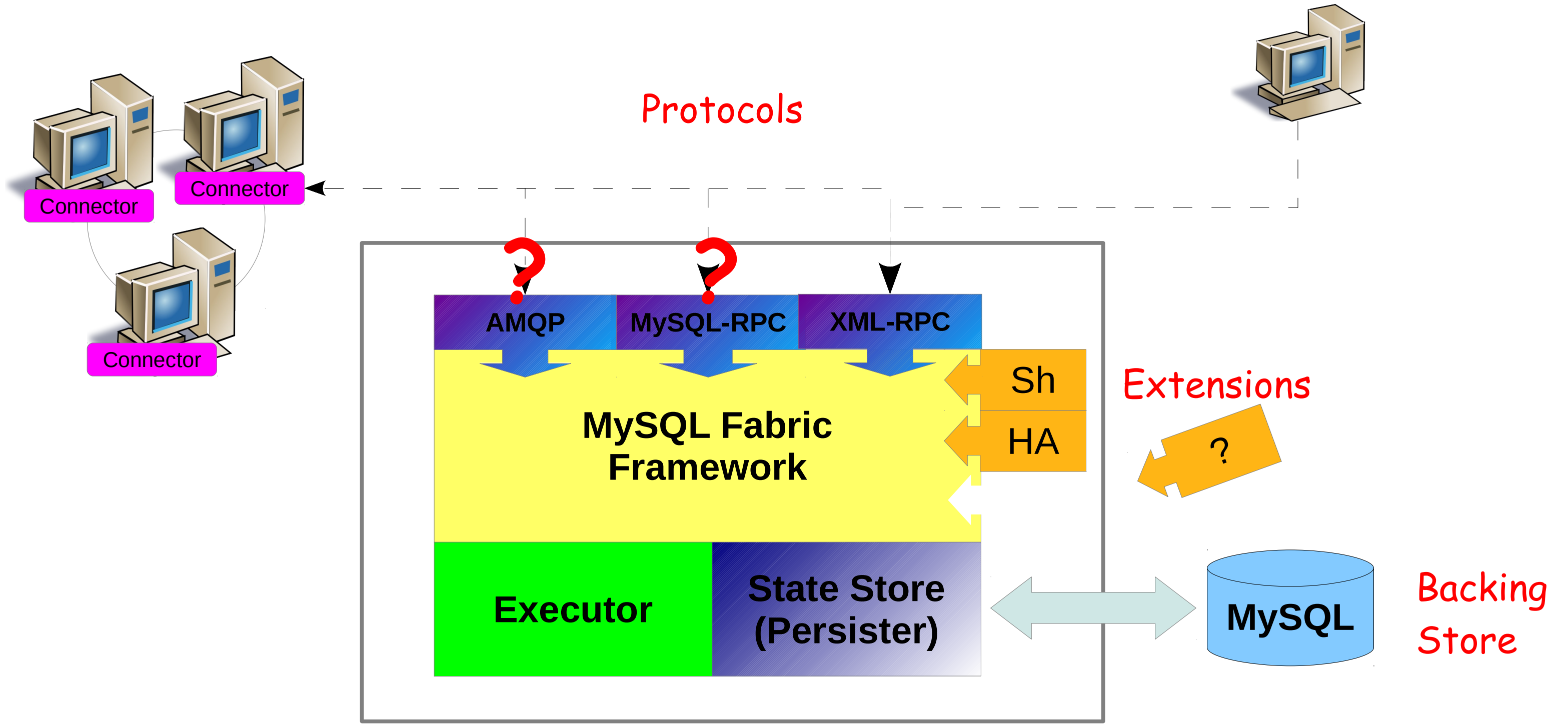


High-Level Components

- Fabric-aware Connectors
  - Python, PHP, and Java
  - Enhanced Connector API
- MySQL Fabric Node
  - Manage information about farm
  - Provide status information
  - Execute procedures
- MySQL Servers
  - Organized in High-Availability Groups
  - Handling application data



MySQL Fabric Node Architecture



MySQL Fabric: Prerequisites

- MySQL Servers (version 5.6.10 or later)
  - Backing store database server
  - Application database servers
- Python 2.6 or 2.7
  - No support for 3.x yet
- MySQL Utilities 1.4
  - Available at https://dev.mysql.com/downloads/tools/utilities
  - “Development release” tab

MySQL Fabric: Configuration

- Backing Store
  - MySQL server
  - Persistent storage for state
  - Storage engine-agnostic
- Protocol
  - Address where node will be
  - Currently only XML-RPC
- Logging
  - Chatty: **INFO** (default)
  - Moderate: **WARNING**
  - URL for rotating log

```
[storage]
address = localhost:3306
user = fabric
password =
database = fabric

[servers]
user = fabric
password =

[protocol.xmlrpc]
address = localhost:32274
threads = 5
disable_authentication = yes

[logging]
level = INFO
url = file:///var/log/fabric.log
```

MySQL Fabric: Basic Commands and Help

- Command Structure

```
mysqlfabric group command ...
```

- Getting help

```
mysqlfabric help
mysqlfabric help commands
mysqlfabric help manage
mysqlfabric help manage setup
```

- MySQL Utilities Documentation:
  - http://dev.mysql.com/doc/mysql-utilities/1.4/en/index.html
- MySQL Fabric Documentation:
  - http://dev.mysql.com/doc/mysql-utilities/1.4/en/fabric.html

Setting up and Tearing down MySQL Fabric

- Create and populate the necessary tables in backing store

```
mysqlfabric manage setup
```

- Remove the tables from backing store

```
mysqlfabric manage teardown
```

- Connects to the database server in “storage” section
  - Ensure that you have the necessary users and privileges

ORACLE®

Starting and Stopping MySQL Fabric

- Start MySQL Fabric node in foreground – print log to terminal

```
mysqlfabric manage start
```

- Start MySQL Fabric node in background – print log to file

```
mysqlfabric manage start --daemonize
```

- Stop MySQL Fabric node

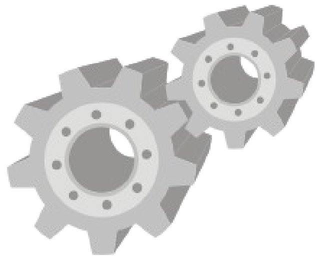
```
mysqlfabric manage stop
```

ORACLE

Architecture for High-Availability

High-Availability Concepts

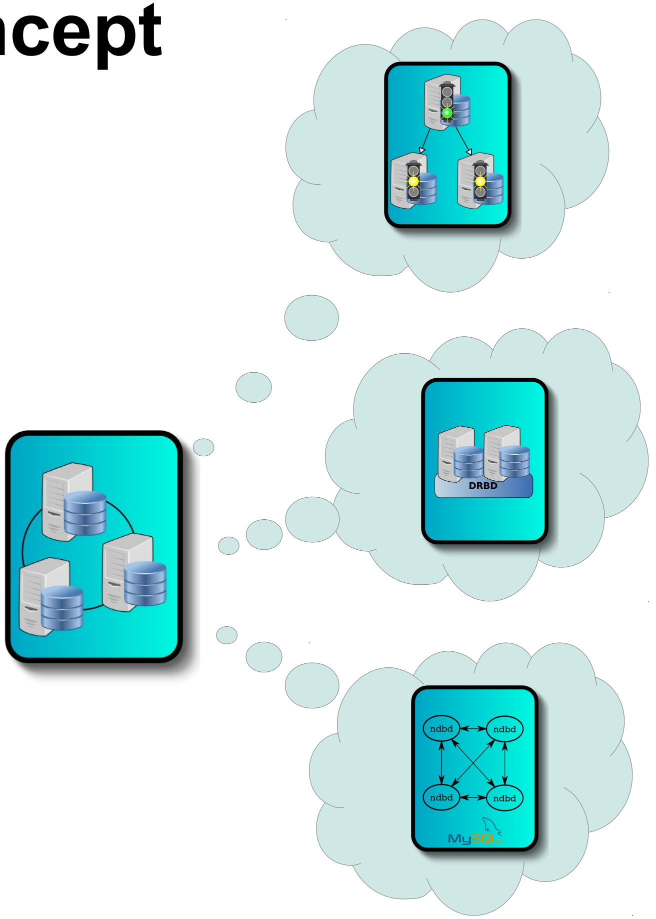
- Redundancy
  - Duplicate critical components
- Monitoring
  - Detecting failing components
  - Monitor load
- Procedures
  - Activate replacements
  - Distribute load



High-Availability Group Concept

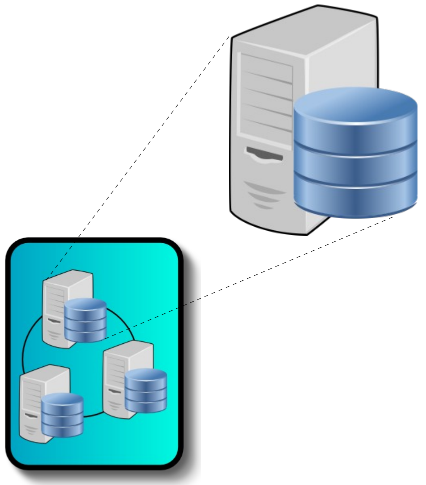
- Group of servers
  - Hardware redundancy
  - Data redundancy
- Generic Concept
  - Implementation-independent
  - Self-managed or externally managed
- Different Types
  - Primary-Backup (Master-Slave) Done!
  - Shared or Replicated Storage
  - MySQL Cluster

Examples Only
Not Implemented



High-Availability Group Concept

- Abstract Concept
  - Set of servers
  - Server attributes
- Connector Attributes
  - Connection information
  - **Mode:** read-only, read-write, ...
  - **Weight:** distribute load
- Management Attributes
  - **Status:** state/role of the server



Status: Primary
Mode: Read-Write
Host: server-1.example.com

Create Groups and add Servers

- Define a group

```
mysqlfabric group create my_group
```

- Add servers to group

```
mysqlfabric group add my_group server1.example.com
mysqlfabric group add my_group server2.example.com
```

Activate High-Availability Group

- Promote one server to be primary

```
mysqlfabric group promote my_group
```

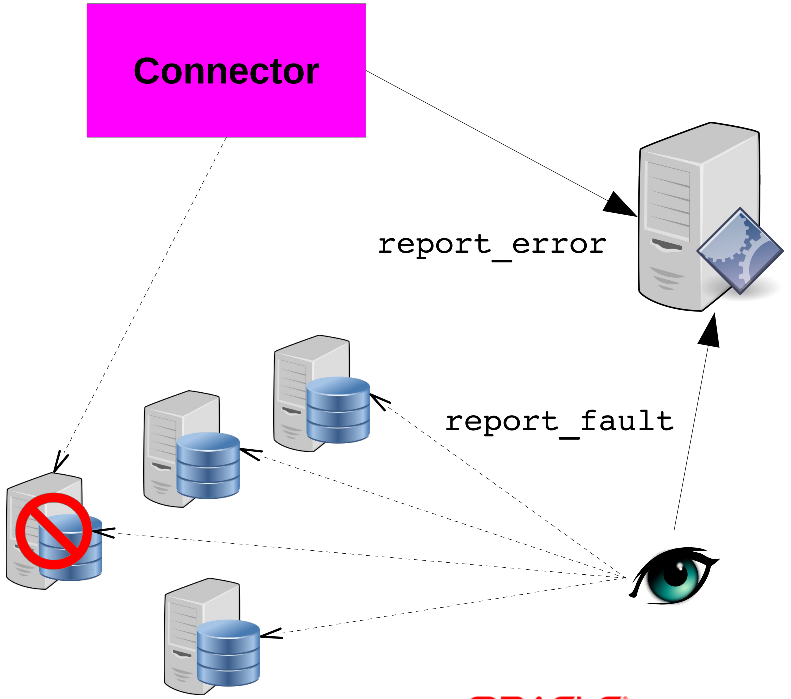
- Tell built-in failure detector to monitor group

```
mysqlfabric group activate my_group
```

Distributed Failure Detector

New in MySQL Fabric 1.4.2

- Connectors report errors
  - Report that an error was noticed
  - Failover based on statistics
  - **report_error(***server*, *source*, *error***)**
- Report failure
  - A server is known to have failed
  - Failover occurs immediately
  - **report_fault(***server*, *source*, *error***)**



Update Only Operations

New in MySQL Fabric 1.4.2

- **Situation:**
  - Server promotion is done elsewhere
  - Real situation does not match content of state-store
- **Problem:**
  - Need to update state store to match real situation
  - Should not touch application servers
- **Solution:** Use update-only option

```
mysqlfabric group promote my_group --update_only \
--slave_uuid=29bf3b2d-b5ac-11e3-a383-58946b051f64
```

Credentials in MySQL Fabric

New in MySQL Fabric 1.4.2

- Credentials
  - Digest Authentication
  - SSL connection
- Digest Authentication
  - RFC 2617
  - **Server:** Realm
  - **Client:** User + Password
- SSL connection
  - Setup Like MySQL Server

```
[storage]
address = localhost:3306
user = fabric
password = xyzzy
database = fabric

[protocol.xmlrpc]
address = localhost:32274
threads = 5
disable_authentication = no
realm = MySQL Fabric
user = admin
password = xyzzy
ssl_ca = /etc/mysql/fabric_ca.pem
ssl_key = /etc/mysql/fabric_key.pem
ssl_cert = /etc/mysql/fabric_cert.pem
```

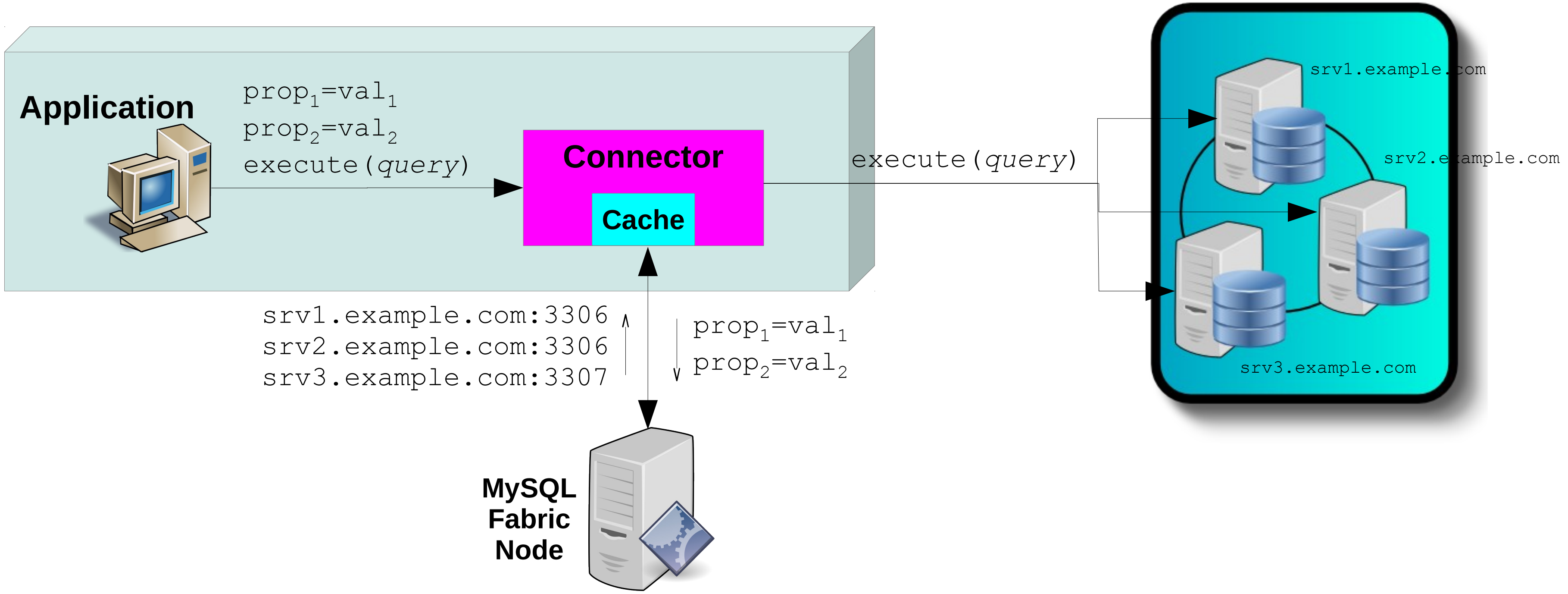
Connecting to a MySQL Fabric Farm

Fabric-aware Connector

- Fabric-aware Connectors
  - Connector/J
  - Connector/Python
  - Connector/PHP
- Fabric-aware Frameworks
  - Doctrine
  - Hibernate
- In this presentation:
  - Connector/Python

- Connector API Extensions
  - Support Transactions
  - Support full SQL
- Decision logic in connector
  - Reducing network load
- Load Balancing
  - Read-Write Split
  - Distribute transactions

Routing Transactions



Fabric-aware Connector API

- Establish a “virtual” connection
  - Real server connection established lazily
- Provide connection information for the *Fabric node*
  - Connector will fetch information about servers

```
import mysql.connector

conn = mysql.connector.connect(
    fabric={"host": "fabric.example.com"},
    user='mats', password='xyzzy', database="employees"
)
```

Enable Connector/Python Error Reporting

New in Connector/Python 1.2.1

- Connectors can report errors to Fabric node
  - Enable using `report_error`
  - Defaults to **False**
  - Require MySQL Fabric 1.4.2

```
import mysql.connector

conn = mysql.connector.connect(
    fabric={"host": "fabric.example.com"},
    user='mats', password='xyzzy', database="employees",
    report_error=True,
)
```

Connector API: Executing a Transaction

- Provide group name
  - **Property:** `group`
  - Fabric will compute candidate servers
- Provide transaction mode
  - **Property:** `mode`
  - Fabric will pick server in right mode

```
conn.set_property(group='my_group', mode=MODE_READWRITE)
cur = conn.cursor()
cur.execute("INSERT INTO employees VALUES (%s,%s,%s)",
            (emp_no, first_name, last_name))
cur.execute("INSERT INTO titles(emp_no,title,from_date)"
            " VALUES (%s,%s,CURDATE())",
            (emp_no, 'Intern'));
conn.commit()
```

Same as before

ORACLE

Executing a Transaction

Hmm... looks like
a read transaction

Where's the sharding key?

Ah, there it is!

Session state?

```
START TRANSACTION;
SELECT salary INTO @s FROM salaries WHERE emp_no = 20101;
SET @s = 1.1 * @s;
INSERT INTO salaries VALUES (20101, @s);
COMMIT;
BEGIN;
CALL update_salary(20202, @s);
COMMIT;
```

What does this
procedure update?

Oops.. it was a
write transaction!

Transaction done!
Clear session state?

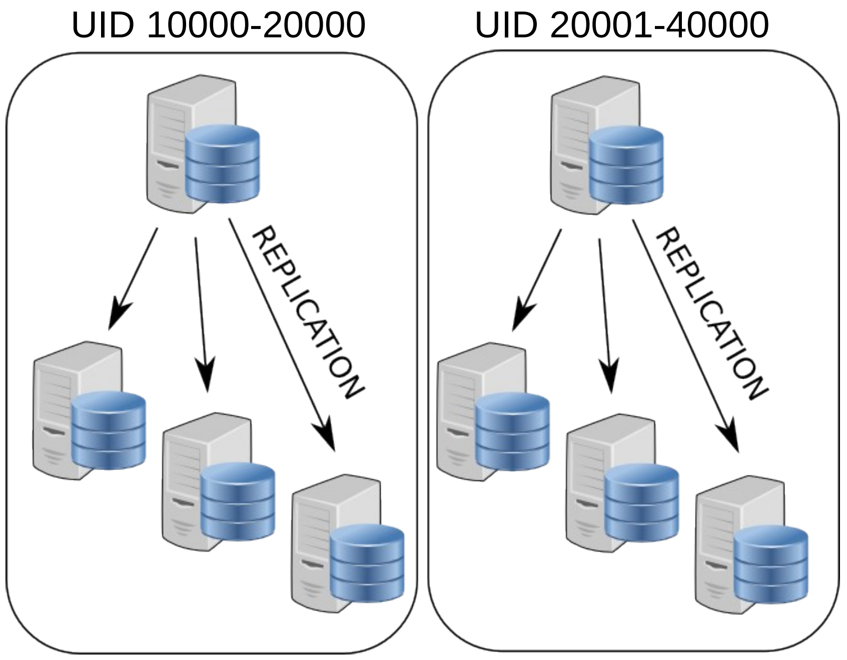
What about connection pools?
Application error?

New transaction! Different connection?
What about the session state?

Architecture for Sharding

Benefits of Sharding

- Write scalability
  - Can handle more writes
- Large data set
  - Database too large
  - Does not fit on single server
- Improved performance
  - Smaller index size
  - Smaller working set



ORACLE

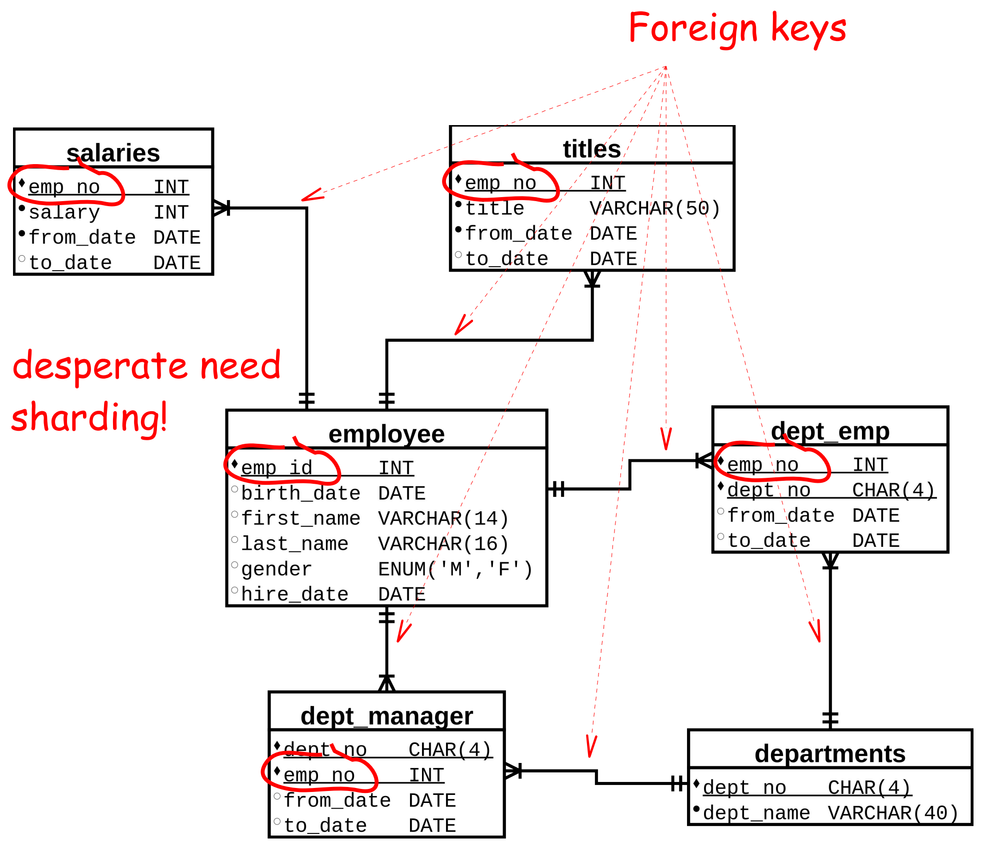
MySQL Fabric: Sharding Goals & Features

- Connector API Extensions
  - Support Transactions
  - Support full SQL
- Decision logic in connector
  - Reducing network load
- Shard Multiple Tables
  - Using same key
- Global Updates
  - Global tables
  - Schema updates
- Sharding Functions
  - Range
  - (Consistent) Hash
- Shard Operations
  - Using built-in executor
  - Shard move
  - Shard split

Sharded Tables

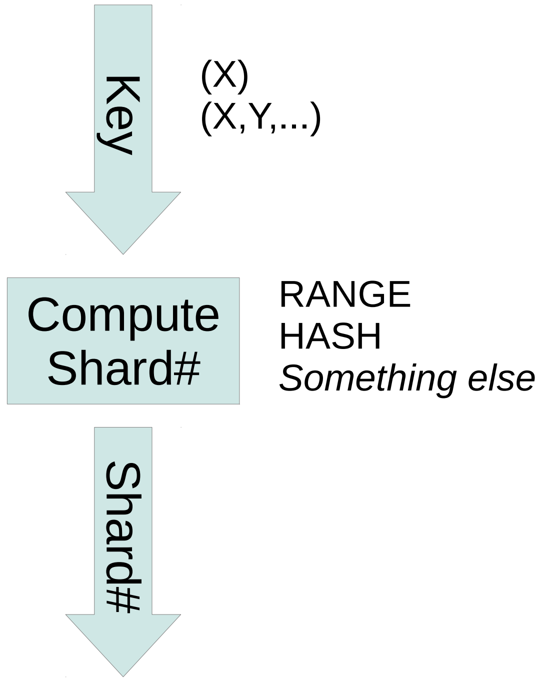
Table	Rows
salaries	284 404 700
titles	44 330 800
employees	30 002 400
dept_emp	33 160 300
dept_manager	2 400
departments	900

In desperate need of sharding!

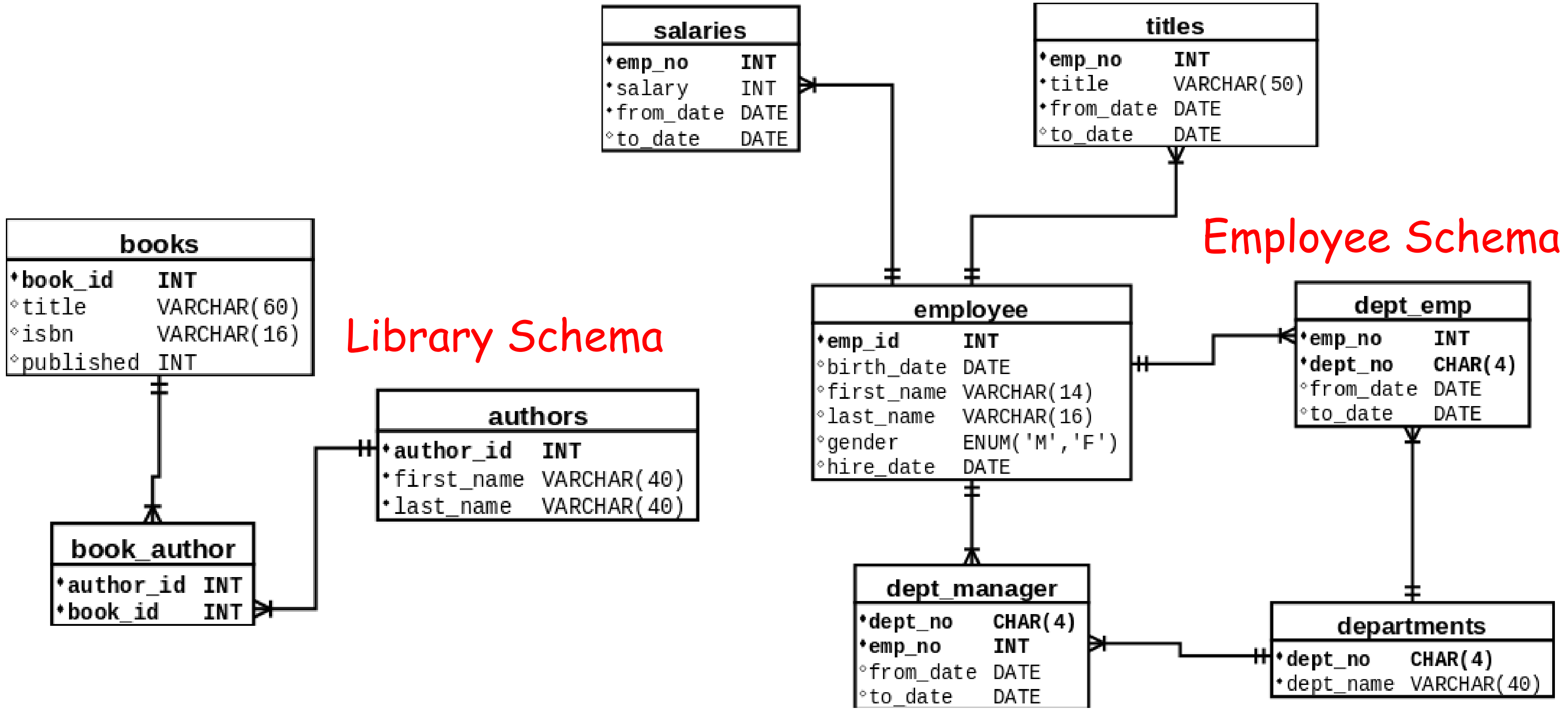


Mapping the Sharding Key

- What is a sharding key?
  - Single column
  - Multi column
    - Same table?
    - Different tables?
- How is the key transformed?
  - Hash
  - Range
  - User-defined

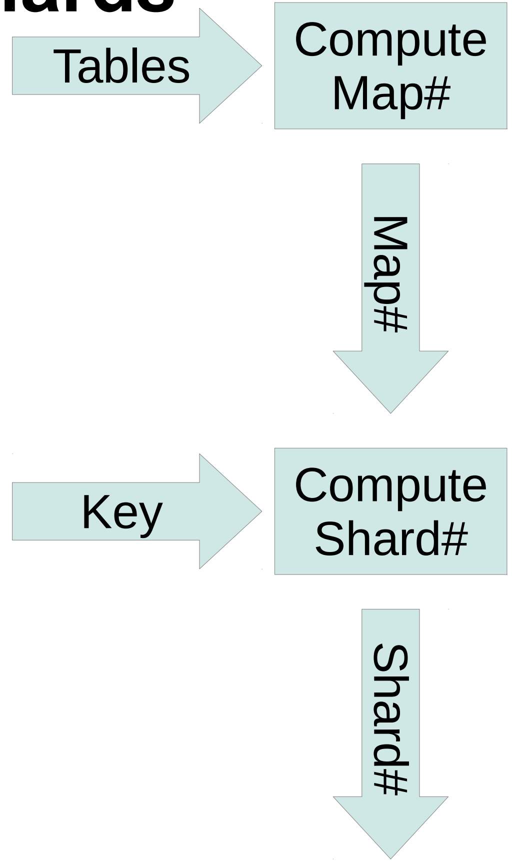


Sharded Tables: Multiple Mappings



Digression: Computing Shards

- Multiple Mappings
  - Which mapping to use?
  - Application don't care
    … but know tables in transaction
  - Currently only one mapping
- Computing shard requires
  - Tables + sharding key
  - Map# + sharding key
- Enhanced Connector API



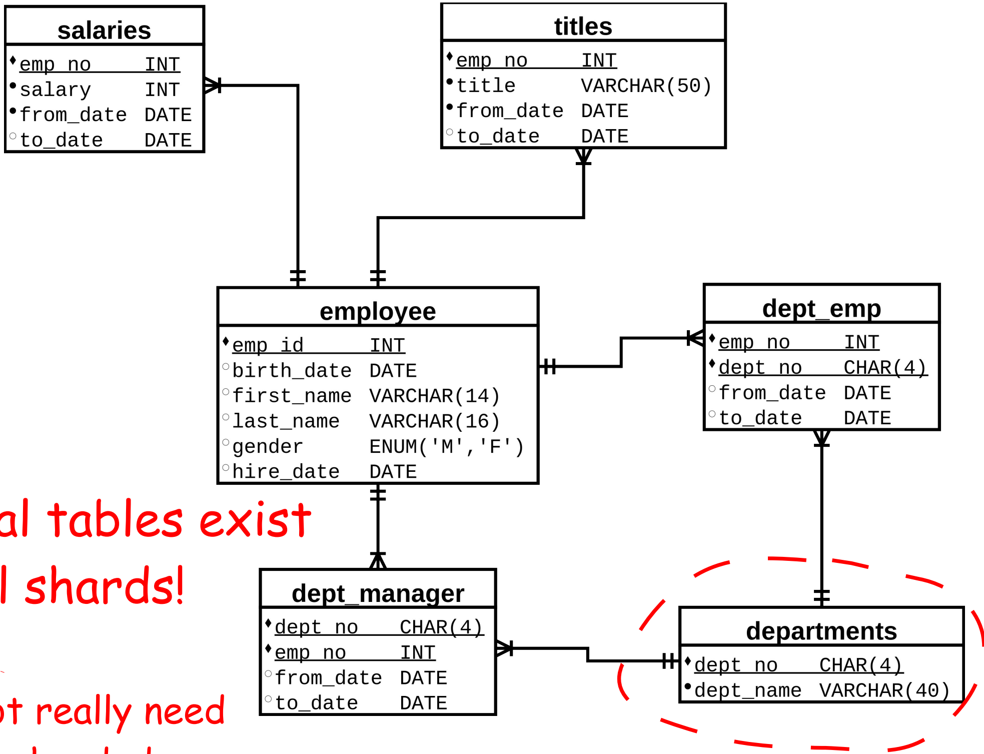
Multi-table Query with Sharded Tables

```
SELECT first_name, last_name, salary
FROM salaries JOIN employees USING (emp_no)
WHERE emp_no = 21012
    AND CURRENT_DATE BETWEEN from_date AND to_date;
```

- Referential Integrity Constraint
  - Example query joining salaries and employees
  - Same key, same shard: co-locate rows for same user
- JOIN normally based on equality
  - Using non-equality defeats purpose of foreign key

Global Tables

Table	Rows
salaries	284 404 700
titles	44 330 800
employees	30 002 400
dept_emp	33 160 300
dept_manager	2 400
departments	900



Global tables exist on all shards!

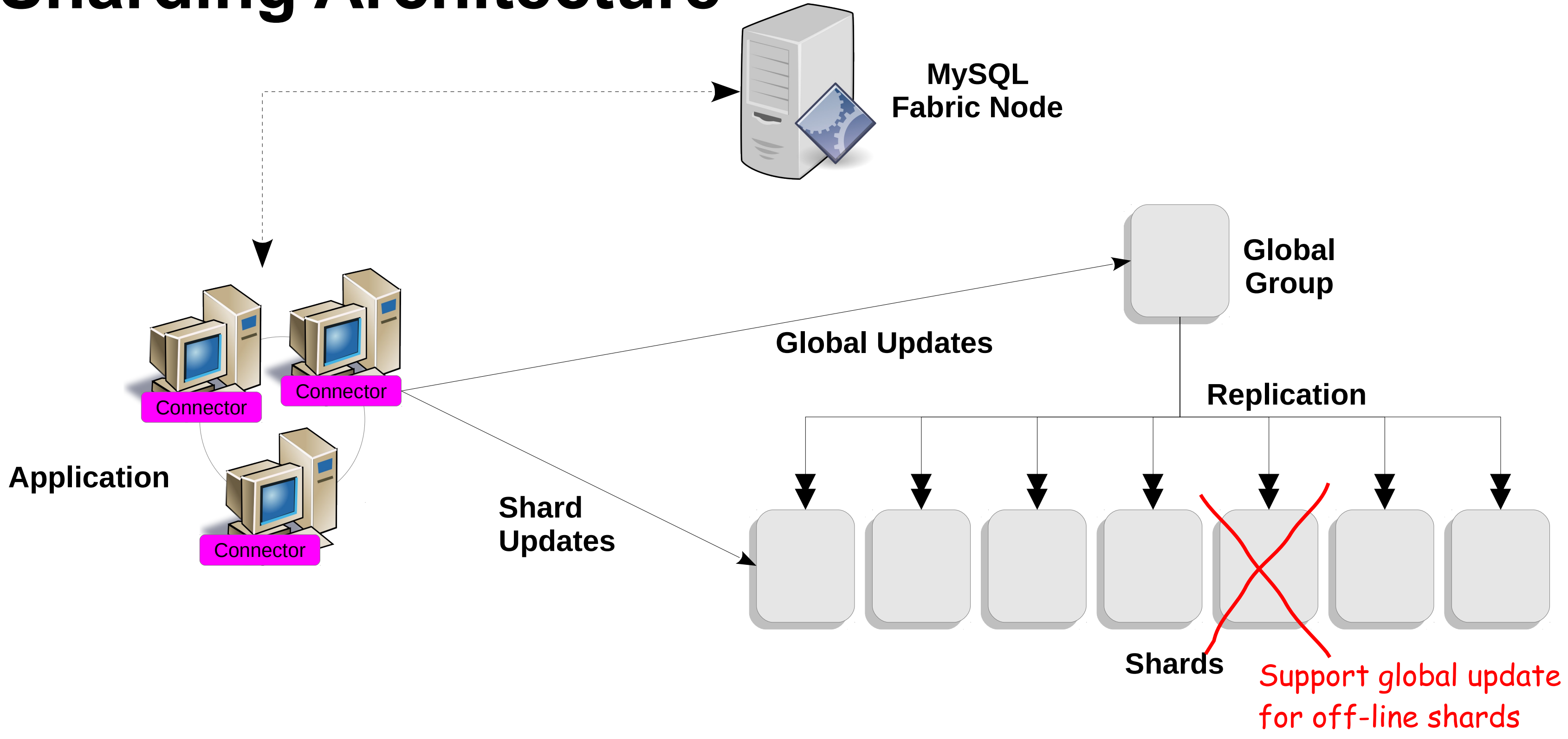
Do not really need to be sharded

Multi-table Query with Global Tables

```
SELECT first_name, last_name, GROUP_CONCAT(dept_name)
  FROM employees JOIN dept_emp USING (emp_no)
                 JOIN departments USING (dept_no)
WHERE emp_no = 21012 GROUP BY emp_no;
```

- JOIN with **departments** table
  - Has no employee number, hence no sharding key
  - Table need to be present on all shards
- But... how do we update global tables?

Sharding Architecture



MySQL Fabric: Sharding Setup

- Set up some groups
  - `my_global` – for global updates
  - `my_group.*` – for the shards
  - Add servers to the groups
- Create a shard mapping
  - A “distributed database”
  - Give information on what tables are sharded
- Add shards
  - Mapping keys to shards

MySQL Fabric: Set up Shard Mapping

Will return a
shard map identifier

- Define shard mapping

```
mysqlfabric sharding \
  create_definition hash my_global
```

Shard map identifier

- Add tables that should be sharded

```
mysqlfabric sharding add_table 1 \
  employees.employees emp_no
mysqlfabric sharding add_table 1 \
  employees.salaries emp_no
```

- *Tables not added are considered global*

ORACLE

MySQL Fabric: Add Shards

Shard map identifier

- Add shards to shard mapping

```
mysqlfabric sharding add_shard 1 \
  “my_group.1,...,my_group.N” --state=ENABLED
```

ORACLE

MySQL Fabric: Moving and Splitting Shards

Shard ID

- Moving a shard from one group to another

```
mysqlfabric sharding move 5 my_group.5
```

- Splitting a shard into two pieces (hash)

```
mysqlfabric sharding split 5 my_group.6
```

Connector API: Shard Specific Query

- Provide tables in query
  - **Property:** `tables`
  - Fabric will compute map
- Provide sharding key
  - **Property:** `key`
  - Fabric will compute shard

```
conn.set_property(tables=['employees.employees', 'employees.titles'],
                  key=emp_no)
cur = conn.cursor()
cur.execute("INSERT INTO employees VALUES (%s,%s,%s)",
            (emp_no, first_name, last_name))
cur.execute("INSERT INTO titles(emp_no, title, from_date)"
            " VALUES (%s, %s, CURDATE())",
            (emp_no, 'Intern'));
conn.commit()
```

Connector API: Shard Specific Query

- Provide tables in query
  - **Property:** `tables`
  - Fabric will compute map

- Provide sharding key
  - **Property:** `key`
  - Fabric will compute shard

```
conn.set_property(tables=['employees.employees', 'employees.titles'],
                  key=emp_no)
cur = conn.cursor()
cur.execute(
    "SELECT first_name, last_name, title"
    "  FROM employees JOIN titles USING (emp_no)"
    " WHERE emp_no = %d", (emp_no,))
for row in cur:
    print row[0], row[1], “,“, row[2]
```

Join queries are sent to correct shard and executed there

Connector API: Global Update

- Provide tables in query
  - **Property:** `tables`
  - Fabric will compute map
  - (Not necessary)
- Set global scope
  - **Property:** `scope`
  - Query goes to global group

```
conn.set_property(tables=['employees.titles'], scope='GLOBAL')
cur = conn.cursor()
cur.execute("ALTER TABLE employees.titles ADD nickname VARCHAR(64)")
```

Closing Remarks

What do we have now?

- MySQL Farm Management
  - High-Availability
  - Sharding
- High-Availability
  - Group Concept
  - Slave promotion
- Sharding
  - Range and hash sharding
  - Shard move and shard split
- Connector APIs
  - Transaction properties
  - “Virtual” connections

- Enhanced Connectors
  - Connector/Python
  - Connector/PHP
  - Connector/J
- Command-line Interface
- XML-RPC Interfaces
- Distributed failure detector
  - Connectors report failures
  - Custom failure detectors
- Credentials
  - RFC 2617
  - SSL support

Thoughts for the Future

- Connector multi-cast
  - Scatter-gather
  - UNION of result sets
  - More complex operations?
- Extension interfaces
  - Improve extension support
  - Improve procedures support
- Command-line interface
  - Improving usability
  - Focus on ease-of-use
- More protocols
  - MySQL-RPC Protocol?
- More frameworks?
- More connectors?
  - C/C++?
  - Fabric-unaware connectors?
- More HA group types
  - DRBD
  - MySQL Cluster

ORACLE

Thoughts for the Future

- "Transparent" Sharding
  - Single-query transactions?
  - Speculative execution?
  - Cross-shard join?
- Multiple shard mappings
  - Independent tables
- Multi-way shard split
  - Efficient initial sharding
  - Better use of resources
- High-availability executor
  - Node failure stop execution
  - Replicated State Machine
    - Paxos?
    - Raft?
  - Continue execution on other Fabric node
- Session Consistency
  - We have a distributed database
  - It should look like a single database

Reading for the Interested

- MySQL Forum: *Fabric, Sharding, HA, Utilities*
  http://forums.mysql.com/list.php?144
- MySQL Fabric Documentation
  http://dev.mysql.com/doc/mysql-utilities/1.4/en/fabric.html
- Migrating From an Unsharded to a Sharded Setup
  http://vnwrites.blogspot.com/2013/09/mysqlfabric-sharding-migration.html
- Configuring and running MySQL Fabric
  http://alfranio-distributed.blogspot.com/2014/03/mysqlfabric-installation.html

Want to contribute?

- Check it

  … and send us use-case and feature suggestions

- Test it

  … and send comments to the forum

- Break it

  … and send in bugs to http://bugs.mysql.com

Keeping in Touch

Mats Kindahl

Twitter: @mkindahl

http://mysqlmusings.blogspot.com

Alfranio Correia

Twitter: @alfranio

http://alfranio-distributed.blogspot.com

Narayanan Venkateswaran

Twitter: @vn_tweets

http://vnwrites.blogspot.com

Geert Vanderkelen

Twitter: @geertjanvdk

http://geert.vanderkelen.org

ORACLE

MySQL Connect @ Oracle OpenWorld Sept 29 – Oct 2

Call for Proposals Open Until April 15!

- Keynotes
- Conferences Sessions
- Birds-of-a-feather sessions
- Tutorials
- Hands-on Labs
- Demos
- Receptions
- OpenWorld Extensive Content

Special Offer:
Register Before May 2 and Save an Extra US$100!
Code: DPER

Thank you!