MySQL Fabric:

Easy Management of MySQL Servers

Mats Kindahl (mats.kindahl@oracle.com)
Alfranio Correia (alfranio.correia@oracle.com)
Narayanan Venkateswaran (v.narayanan@oracle.com)

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decision. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Presentation Outline

- Introducing MySQL Fabric
- High-Availability
- Scaling
- Managing
- Connecting

MySQL Fabric

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

MySQL Fabric: What is it?

- “Farm” Management System
  - Distributed Framework
- Procedure Execution
  - Farm Management
- Extensible
  - Extensions are first-class
  - High-Availability Groups
  - “Semi-Automatic” Sharding

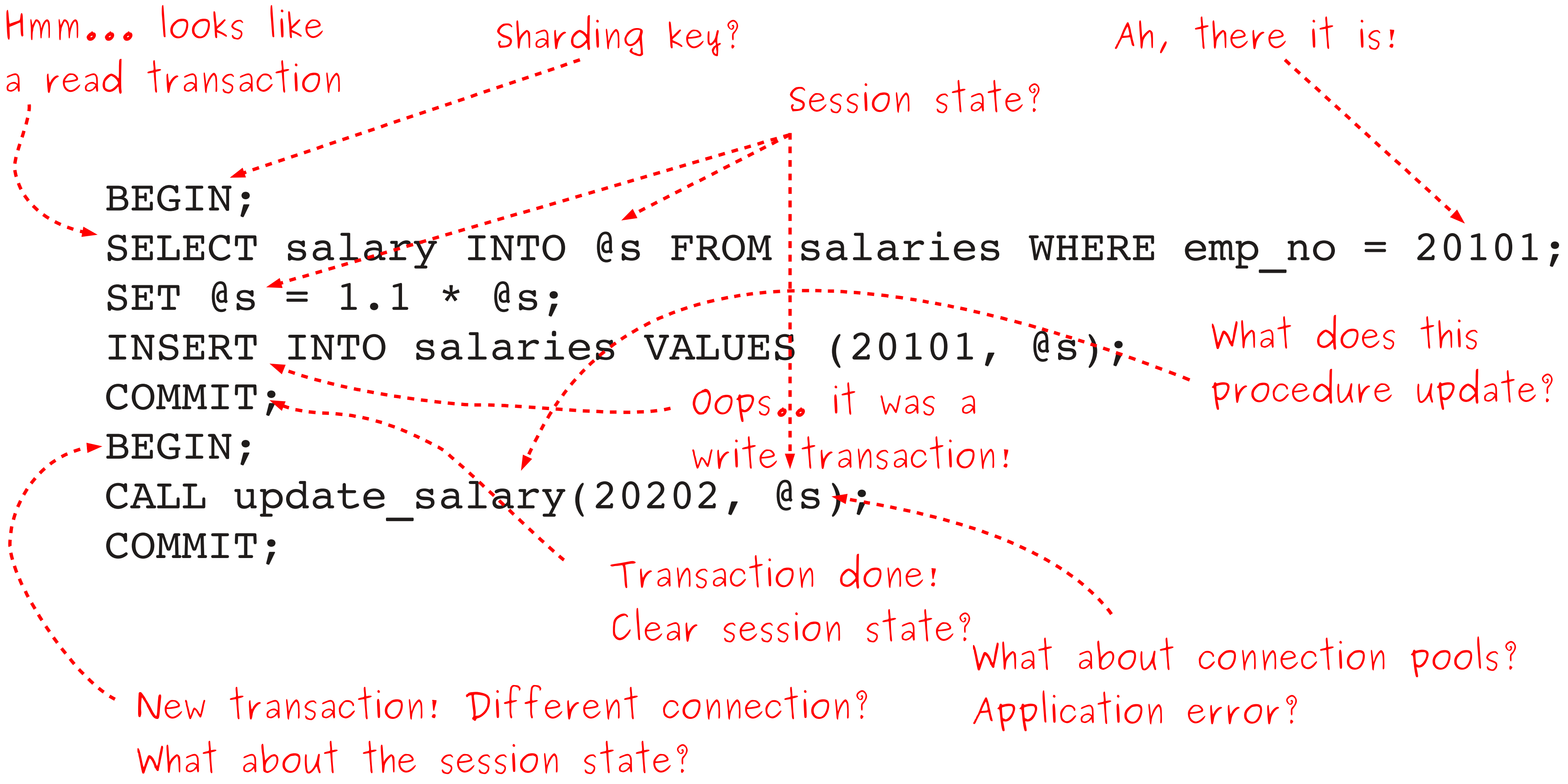
- Written in Python
- Early alpha
  - Long road ahead
- Open Source
  - You can participate
  - Suggest features
  - Report bugs
  - Contribute patches
- MySQL 5.6 is focus

MySQL Fabric: Goals & Features

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

- Shard Multiple Tables
  - Using same key
- Sharding Functions
  - Range
  - (Consistent) Hash
- Shard Operations
  - Using built-in executor
  - Shard move
  - Shard split

Transaction Handling



```
BEGIN;
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;
```

Hmm... looks like a read transaction

Sharding key?

Ah, there it is!

Session state?

What does this procedure update?

Oops.. it was a write transaction!

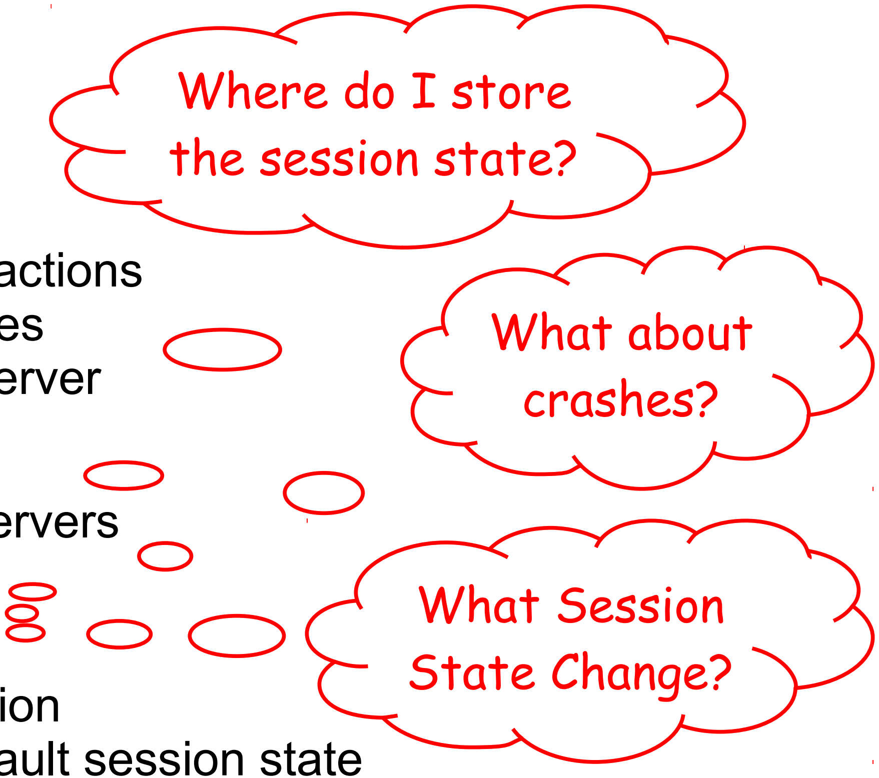
Transaction done! Clear session state?

What about connection pools? Application error?

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

Transaction Handling

- Routing Transactions
  - Pre-declare properties of transactions
  - Detecting transaction boundaries
  - Push as much as possible to server
- Managing Session State
  - Move session state between servers
    - Easy to use
    - Expensive and error prone
  - Reset state after each transaction
    - Transactions start with default session state

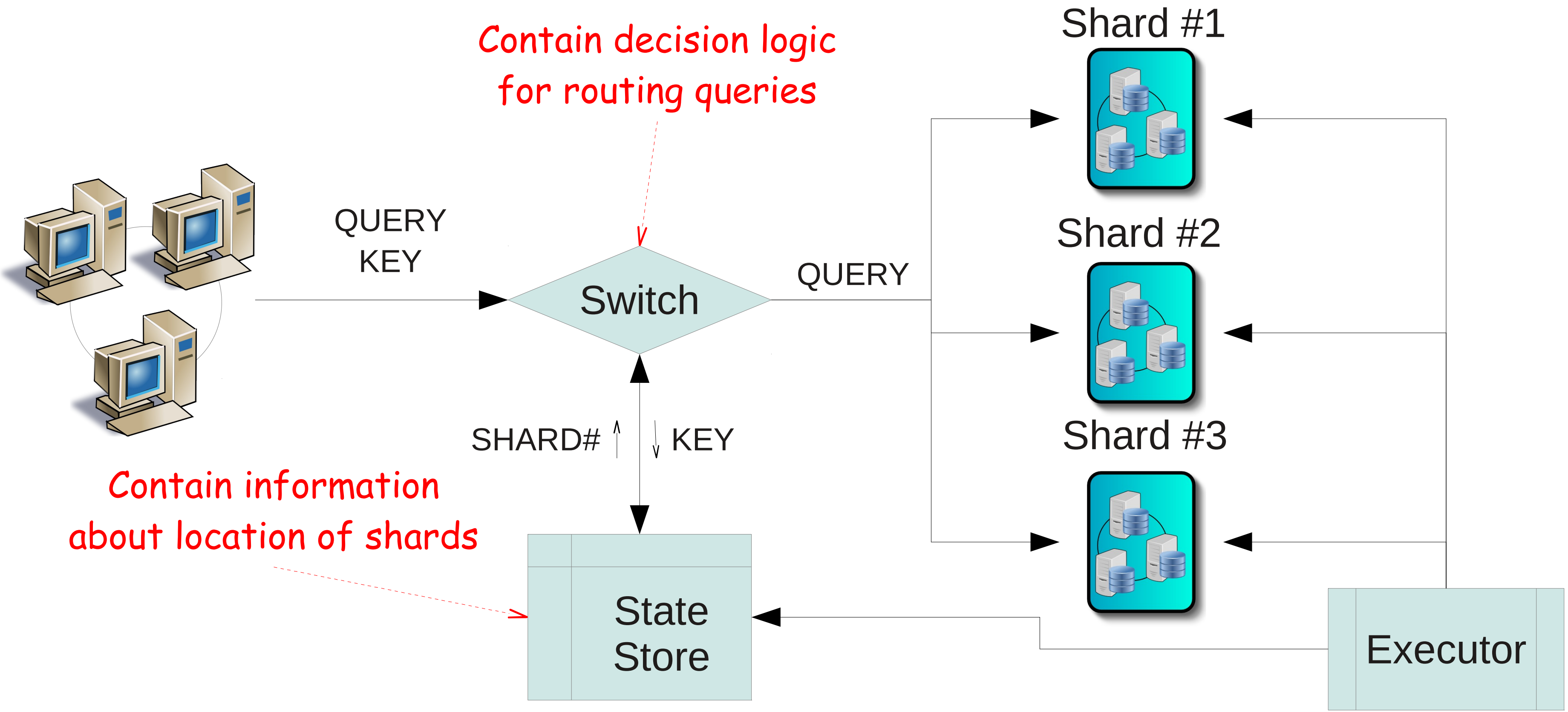


Where do I store the session state?

What about crashes?

What Session State Change?

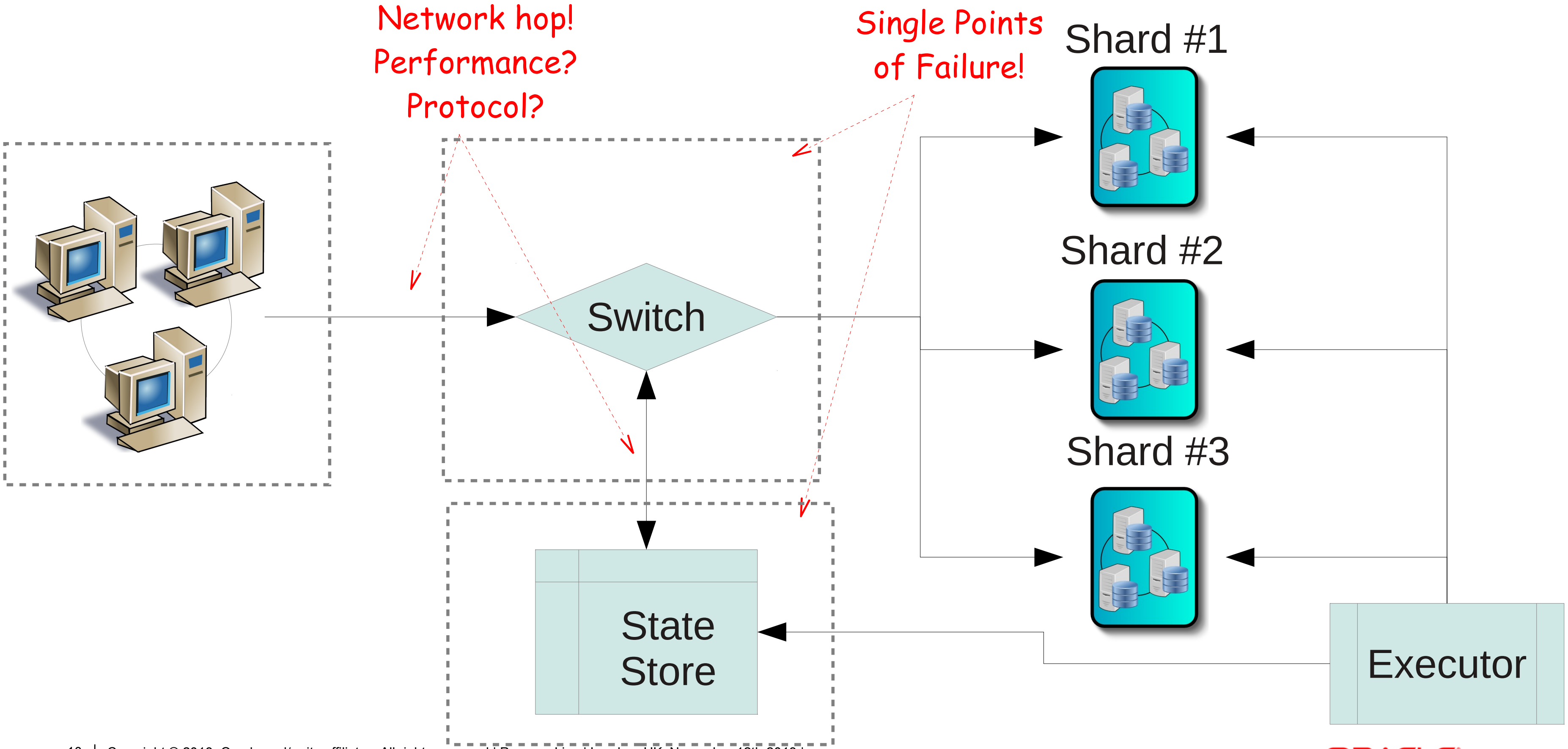
Routing Transactions



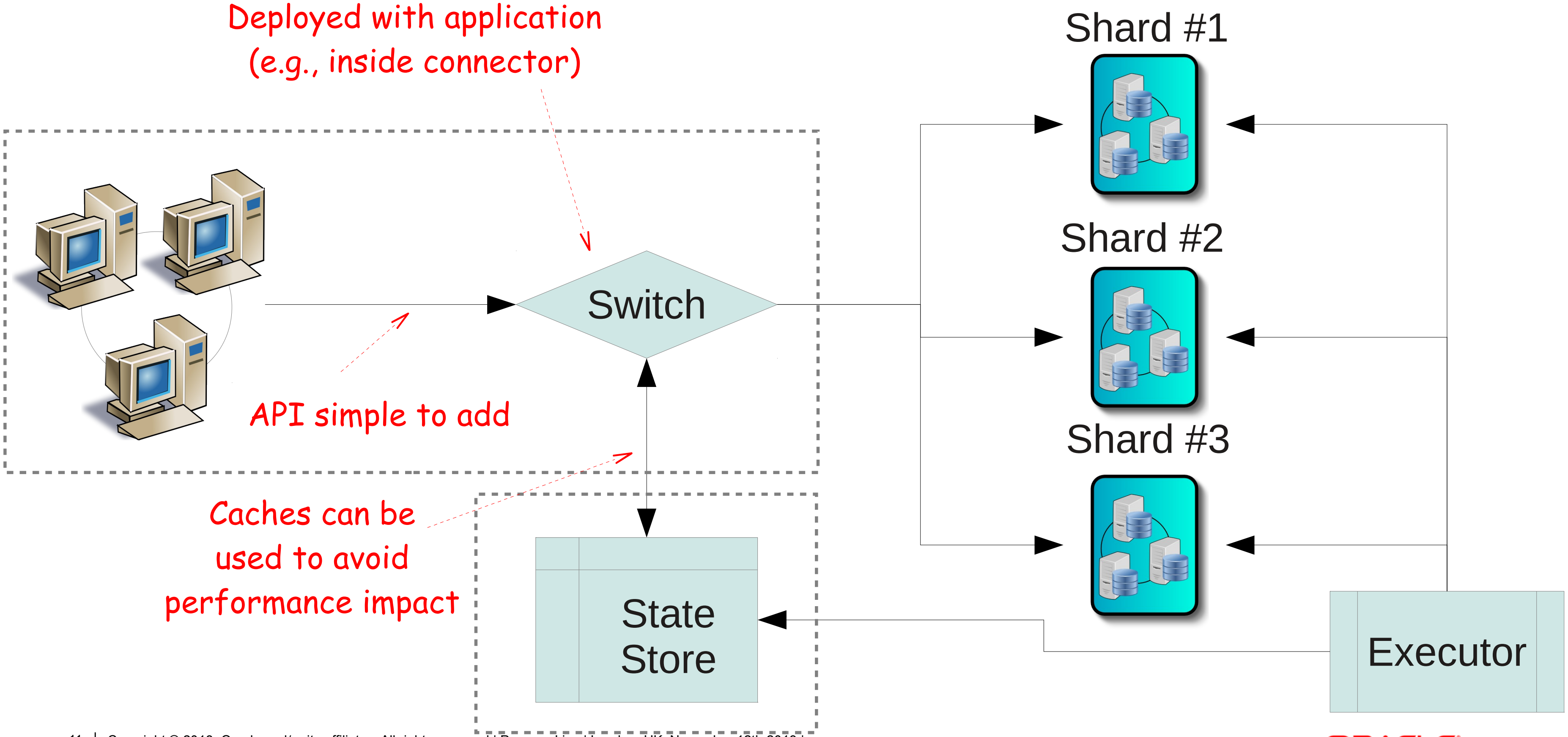
Contain decision logic for routing queries

Contain information about location of shards

Deployment for Routing Transactions



Deployment for Routing Transactions



Deployed with application (e.g., inside connector)

API simple to add

Caches can be used to avoid performance impact

Switch

State Store

Shard #1

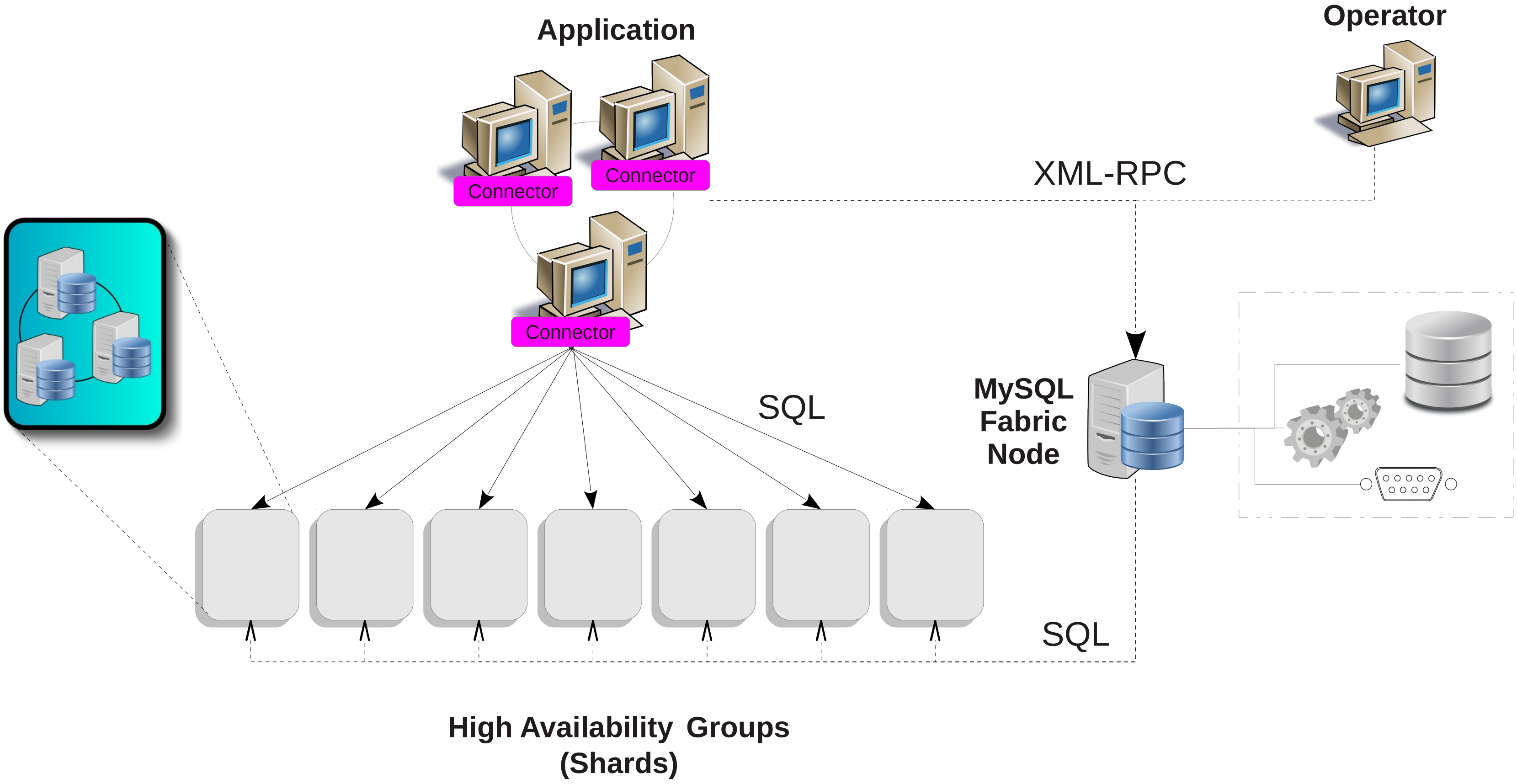
Shard #2

Shard #3

Executor

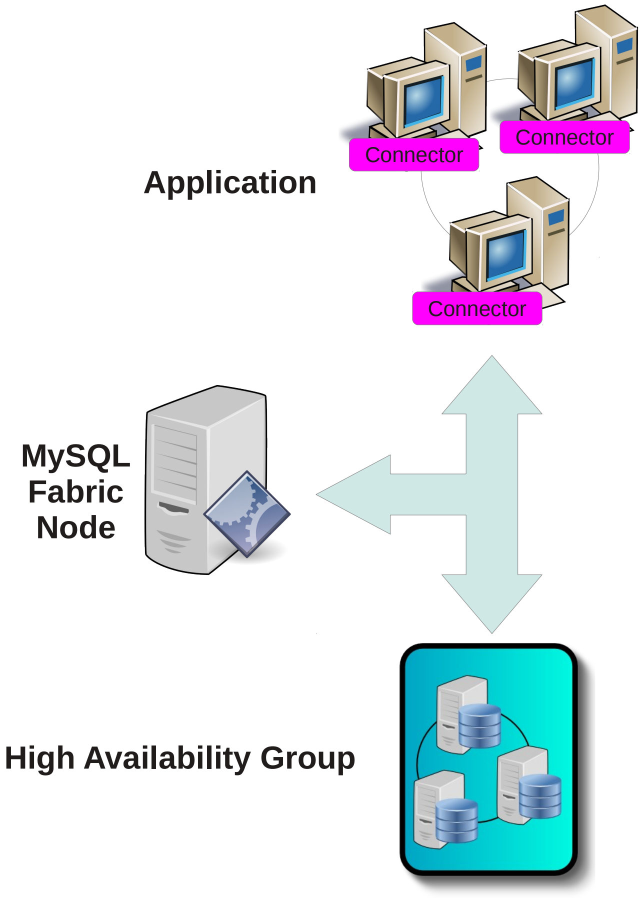
Introducing MySQL Fabric

Birds-eye View

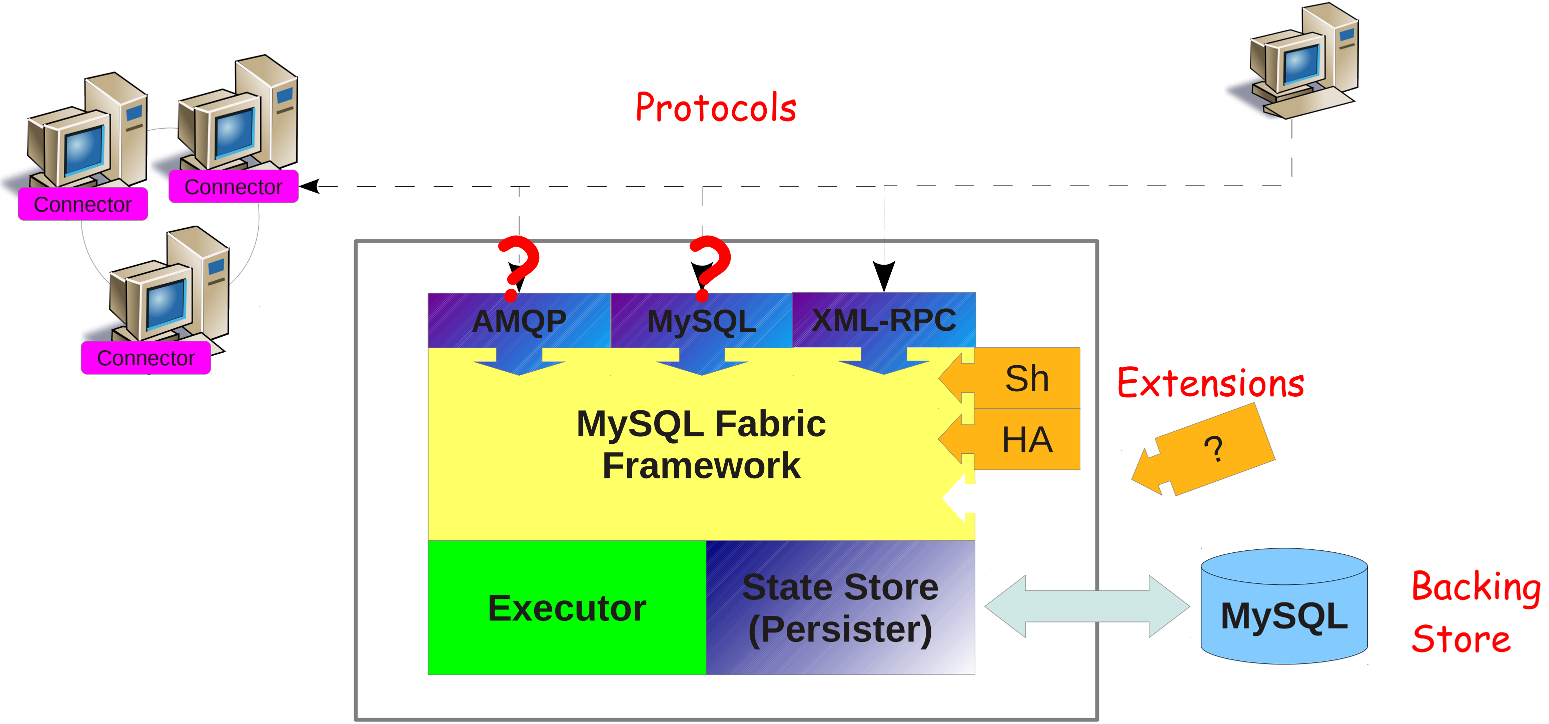


High-Level Components

- Fabric-aware Connectors
  - Python, PHP, and Java
  - Extended Connector API
- MySQL Servers
  - In High-Availability Groups
  - Managing the data
- MySQL Fabric Node
  - Maintain Meta-Information
  - Information Interfaces
  - Execute Procedures



MySQL Fabric Node Architecture



MySQL Fabric: Prerequisites

- MySQL Servers (version 5.6.10 or later)
  - Server for meta-data backing store
  - Servers being managed
- Python 2.6 or 2.7
  - No support for 3.x yet
- MySQL Utilities 1.4.0
  - Available at http://labs.mysql.com/

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
connection_timeout = 6

[protocol.xmlrpc]
address = localhost:8080
threads = 5

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

MySQL Fabric: Setup and Teardown

- Create the necessary tables in backing store

```
mysqlfabric manage setup
```

- Remove the tables from backing store

```
mysqlfabric manage teardown
```

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

MySQL Fabric: Starting and Stopping

- 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
```

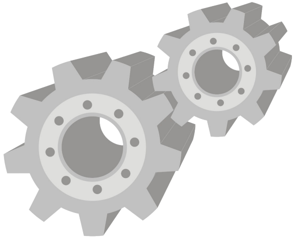
- Stop MySQL Fabric node

```
mysqlfabric manage stop
```

Architecture for High-Availability

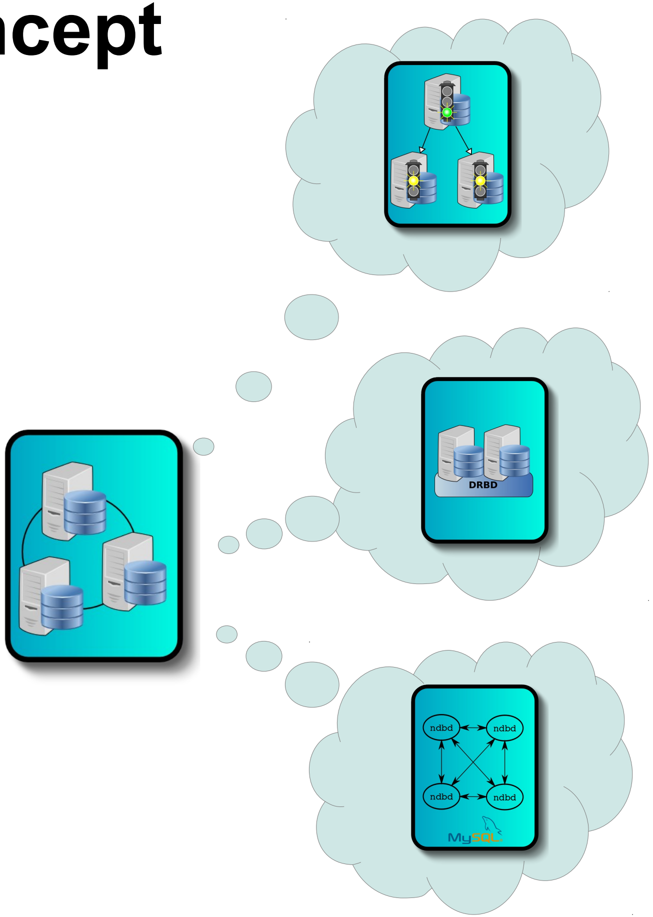
High-Availability Concepts

- Redundancy
  - Duplicate critical components
- Monitoring
  - Detecting failing components
  - Monitor load
- Procedure
  - Activate backups
  - Distribute load



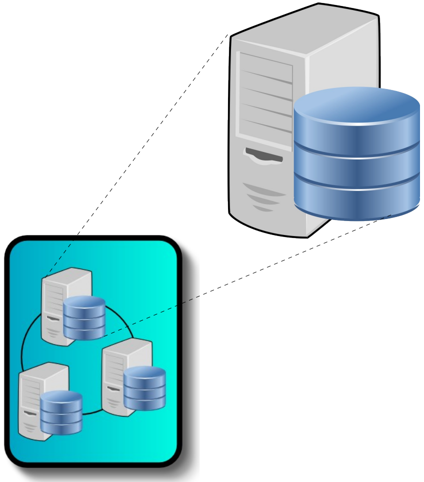
High-Availability Group Concept

- 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



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
  - **State:** state/role of the server



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

MySQL Fabric: Create Groups and add Servers

- Define a group

```
mysqlfabric group create my_group
```

User + Password
(Likely to go away)

- Add servers to group

```
mysqlfabric group add my_group server1.example.com \
  mats xyzzy
mysqlfabric group add my_group server2.example.com \
  mats xyzzy
```

MySQL Fabric: Create Groups and add Servers

- Promote one server to be primary

```
mysqlfabric group promote my_group
```

- Tell failure detector to monitor group

```
mysqlfabric group activate my_group
```

Connecting to a MySQL Fabric Farm

Fabric-aware Connector API

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

- Fabric-aware Connectors
  - Connector/J
  - Connector/Python
  - Connector/PHP
- Fabric-aware Frameworks
  - Doctrine
  - Hibernate
- Focus on Connector/Python

Fabric-aware Connector API

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

```
import mysql.connector.fabric as connector

conn = connector.MySQLFabricConnection(
    fabric={"host": "fabric.example.com", "port" : 8080},
    user='mats', database="employees")
```

Connector API: Executing a Transaction

- Provide group name
  - **Property:** `group`
  - Fabric will compute candidate servers

- Provide transaction type
  - **Property:** `type`
  - Fabric will pick server in right mode

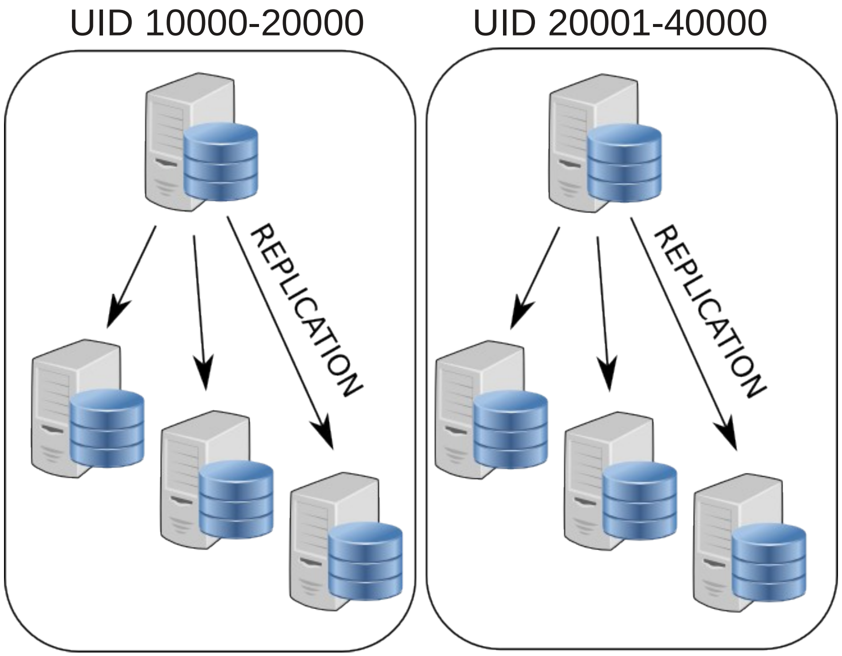
```
conn.set_property(group='my_group', type=TYPE_READWRITE)
cur = conn.cursor()
cur.execute("INSERT INTO employees VALUES (%s,%s,%s)",
            (emp_no, first_name, last_name))
cur.execute("INSERT INTO titles(emp_no,title,from_date)"
            “ VALUES (%s,%s,CURDATE())",
            (emp_no, 'Intern'));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

Transactions work fine!

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
  - Improve performance

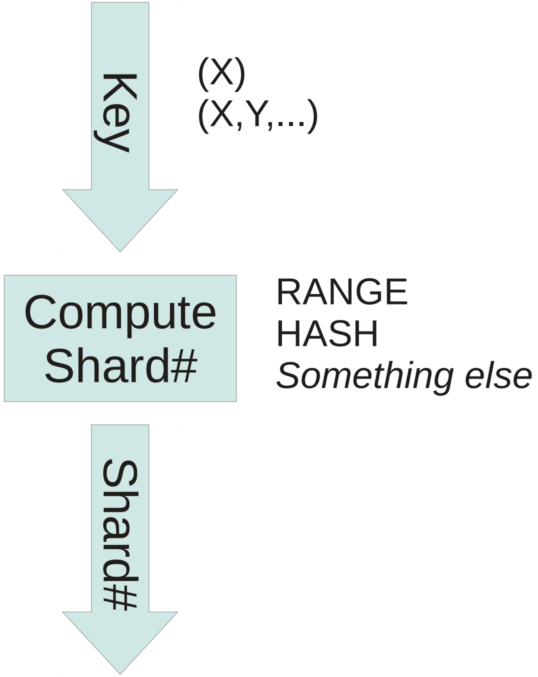


MySQL Fabric: Sharding Goals & Features

- 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

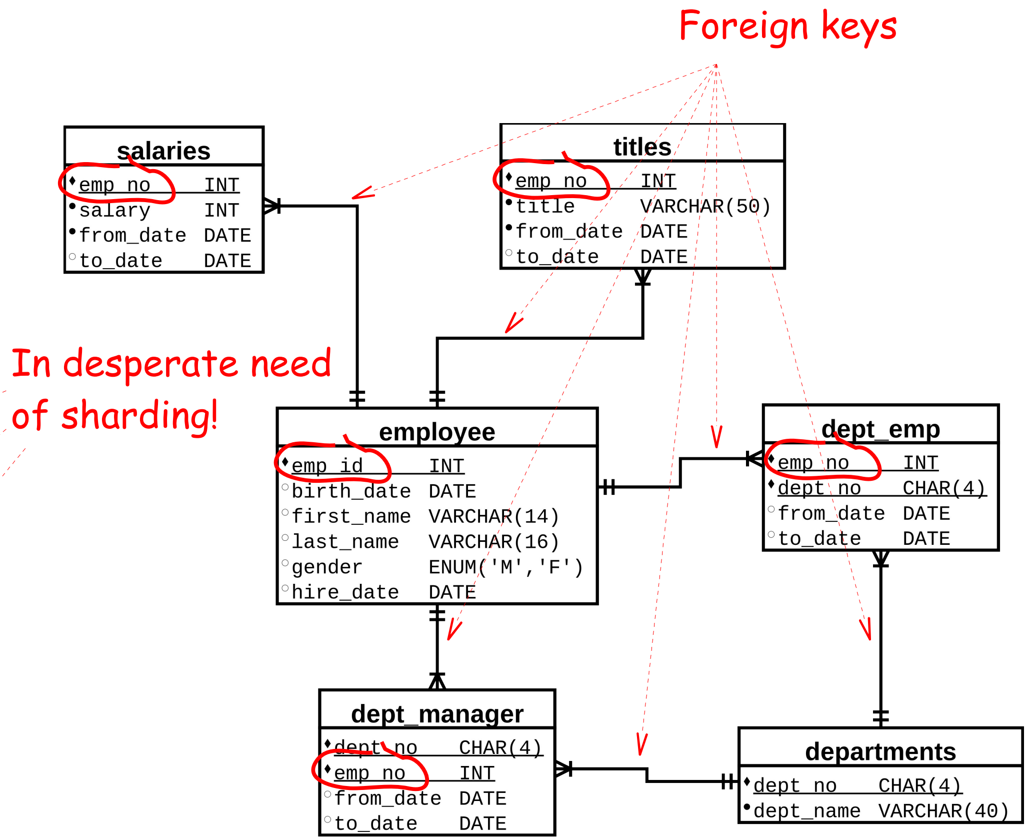
Mapping the Sharding Key

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



Sharded Tables

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



Foreign keys

In desperate need of sharding!

salaries

♦emp_no	INT
•salary	INT
•from_date	DATE
○to_date	DATE

titles

♦emp_no	INT
•title	VARCHAR(50)
•from_date	DATE
○to_date	DATE

employee

♦emp_id	INT
○birth_date	DATE
○first_name	VARCHAR(14)
○last_name	VARCHAR(16)
○gender	ENUM('M','F')
○hire_date	DATE

dept_emp

♦emp_no	INT
♦dept_no	CHAR(4)
○from_date	DATE
○to_date	DATE

dept_manager

♦dept_no	CHAR(4)
♦emp_no	INT
○from_date	DATE
○to_date	DATE

departments

♦dept_no	CHAR(4)
•dept_name	VARCHAR(40)

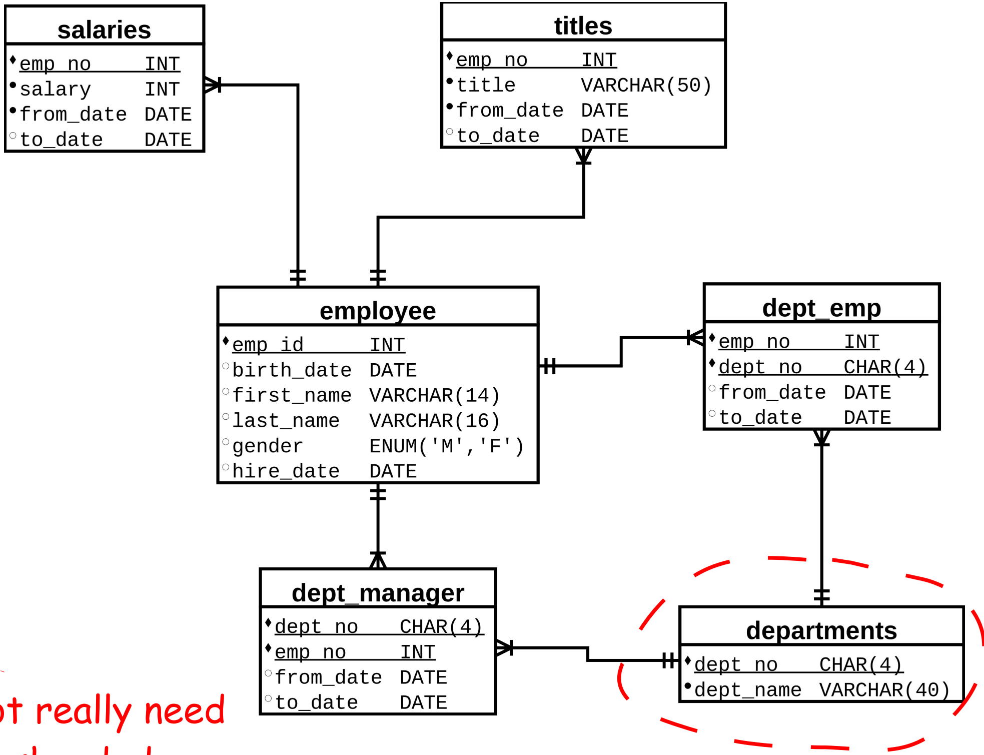
Multi-table Query with Sharded Tables

```
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



salaries	
emp_no	INT
salary	INT
from_date	DATE
to_date	DATE

titles	
emp_no	INT
title	VARCHAR(50)
from_date	DATE
to_date	DATE

employee	
emp_id	INT
birth_date	DATE
first_name	VARCHAR(14)
last_name	VARCHAR(16)
gender	ENUM('M','F')
hire_date	DATE

dept_emp	
emp_no	INT
dept_no	CHAR(4)
from_date	DATE
to_date	DATE

dept_manager	
dept_no	CHAR(4)
emp_no	INT
from_date	DATE
to_date	DATE

departments	
dept_no	CHAR(4)
dept_name	VARCHAR(40)

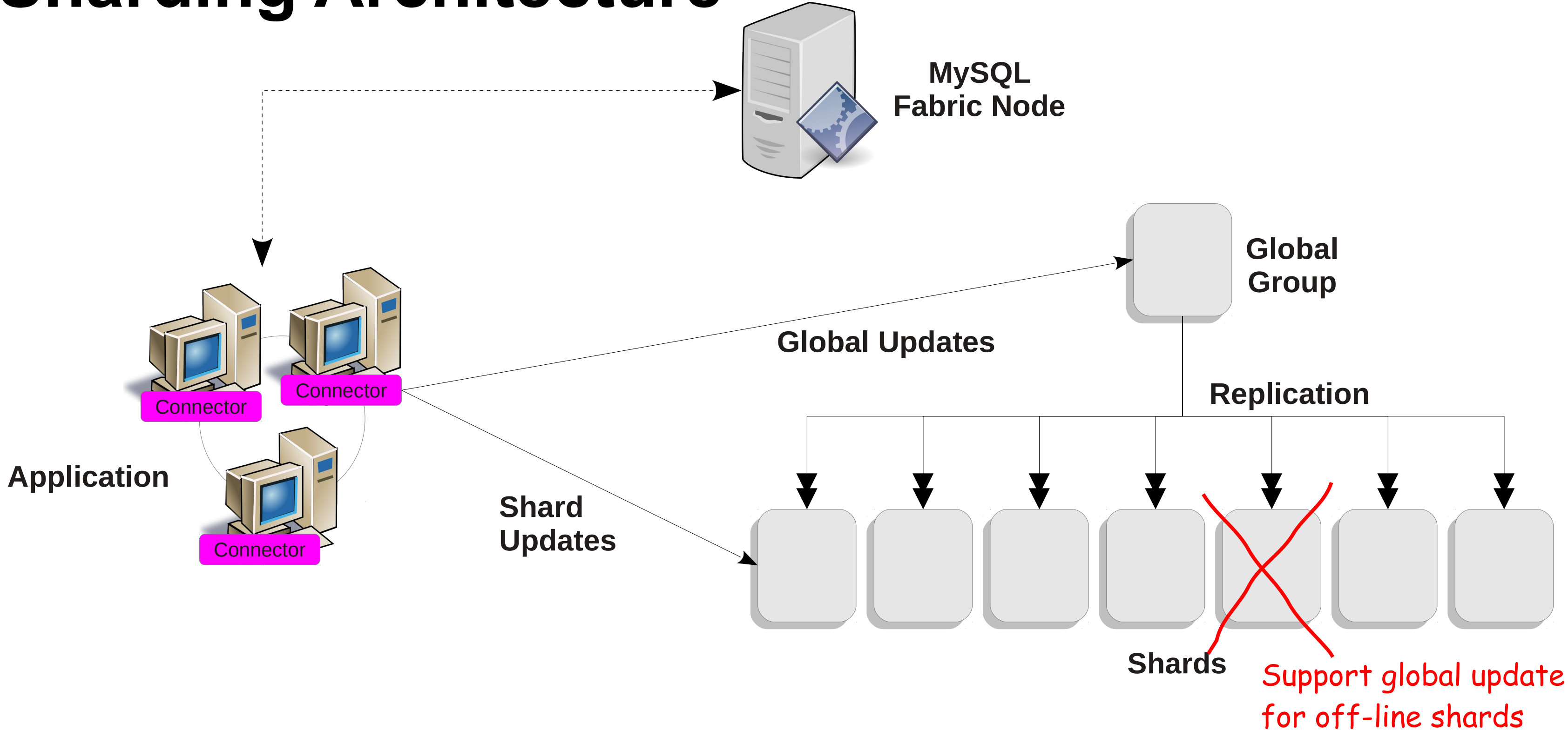
Do not really need
to be sharded

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
- How do we update global tables?

Sharding Architecture



Support global update
for off-line shards

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”
  - Mapping keys to shards
  - Give information on what tables are sharded
- Add shards

MySQL Fabric: Set up Shard Mapping

Per-mapping Global Group
(Likely to go away)

Will show the shard map identifier (a number)

- Define shard mapping

```
mysqlfabric sharding define hash my_global
```

Shard map identifier

- Add tables that should be sharded

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

- *Tables not added are global*

MySQL Fabric: Add Shards

Shard map identifier

- Add shards to shard mapping

```
mysqlfabric sharding add_shard 1 my_group.1 enabled
       .
       .
       .
mysqlfabric sharding add_shard 1 my_group.N enabled
```

MySQL Fabric: Moving and Splitting Shards

Shard ID

- Moving a shard from one group to another

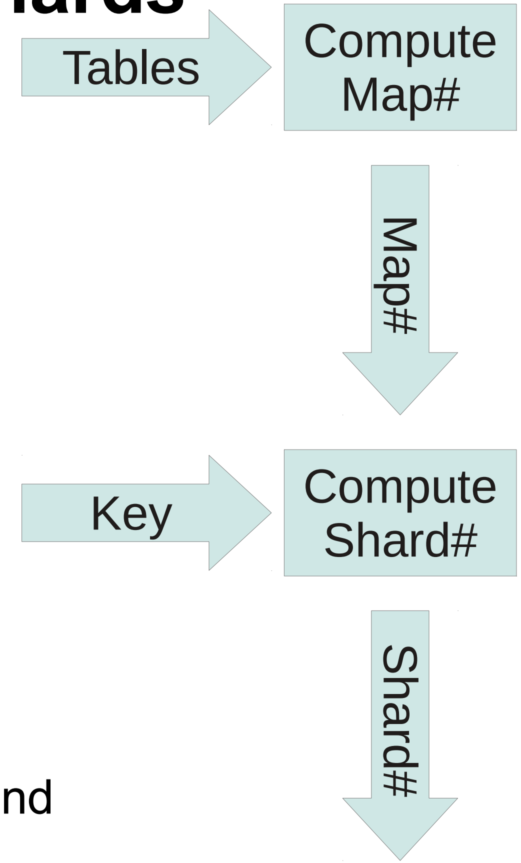
```
mysqlfabric sharding move 5 my_group.5
```

- Splitting a shard into two pieces (hash)

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

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
- Extended Connector API
  - Extra properties passed out-of-band



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()
```

Transactions work fine!

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
  - (Likely to not be needed)

- 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
  - Simple failure detector
  - Slave promotion
- Connector APIs
  - Transaction properties
  - “Logical” connection management

- Enhanced Connectors
  - Connector/Python
  - Connector/PHP
  - Connector/J
- Sharding
  - Range and hash sharding
  - Shard move and shard split
  - Global tables and updates
- Command-line Interface
  - Easy setup and management
  - XML-RPC Interfaces

Thoughts for the Future

- Connector multi-cast
  - Scatter-gather
  - UNION of result sets
  - More complex operations?
- Internal interfaces
  - Improve extension support
  - Improve procedures support
- Command-line interface
  - Improving usability
  - Focus on ease-of-use

- More protocols
  - MySQL-RPC Protocol?
  - AMQP?
- More frameworks?
- More HA group types
  - DRBD
  - MySQL Cluster
- Fabric-unaware connectors?

Thoughts for the Future

- “More transparent” sharding
  - Single-query transactions
  - Cross-shard join is a problem
- 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
  - Fail over to other Fabric node
- Distributed failure detector
  - Connectors report failures
  - Custom failure detectors

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

Reading for the Interested

- MySQL Forum: *Fabric, Sharding, HA, Utilities*
  http://forums.mysql.com/list.php?144
- A Brief Introduction to MySQL Fabric
  http://mysqlmusings.blogspot.com/2013/09/brief-introduction-to-mysql-fabric.html
- MySQL Fabric – Sharding – Introduction
  http://vnwrites.blogspot.com/2013/09/mysqlfabric-sharding-introduction.html
- Writing a Fault-tolerant Database Application using MySQL Fabric
  http://alfranio-distributed.blogspot.se/2013/09/writing-fault-tolerant-database.html

Keeping in Touch

Mats Kindahl

Twitter: @mkindahl

Blog: http://mysqlmusings.blogspot.com

Alfranio Correia

Twitter: @alfranio

Blog: http://alfranio-distributed.blogspot.com

Narayanan Venkateswaran

Twitter: @vn_tweets

Blog: http://vnwrites.blogspot.com

Thank you!