MySQL 5.7 Fabric: High Availability and Sharding

Ulf Wendel, MySQL/Oracle

The speaker says…

MySQL is under pressure by NoSQL solutions. For OLTP workloads the pressure is on:

- Easy to use, built-in high availability
- Easy to use, elastic, horizontal scaling for the cloud
- Attractive data models and drivers for developers

MySQL 5.7 introduces a High Availability and Sharding solution called Fabric – of course, it is Open Source and freely abailable. Similarities with MongoDB cannot be overseen, but Fabric is different. However, we start with a short recap on database clusters before we dive into MySQL Fabric.

Goals of distributed databases

Availability

- Cluster as a whole unaffected by loss of nodes

Scalability

- Geographic distribution
- Scale size in terms of users and data
- Database specific: read and/or write load

Distribution Transparency

- Access, Location, Migration, Relocation (while in use)
- Replication
- Concurrency, Failure

The speaker says…

A distributed database cluster strives for maximum availability and scalability while maintaining distribution transparency. MySQL Cluster has a shared-nothing design good enough for 99,999% (five minutes downtime per year). It scales from Rasperry Pi run in a briefcase to 1.2 billion write transactions per second on a 30 data nodes cluster (if using possibly unsupported bleeding edge APIs.) It offers full distribution transparency with the exception of partition relocation to be triggered manually but performed transparently by the cluster.

On the other end of the distribution transparency scale is MySQL Replication. Why two products?

What kind of cluster?

		Where are transactions run?	
		Primary Copy	Update Anywhere
When does synchronization happen?	Eager	Not available for MySQL	MySQL Cluster 3rd party
	Lazy	MySQL Replication 3rd party	MySQL Cluster Replication

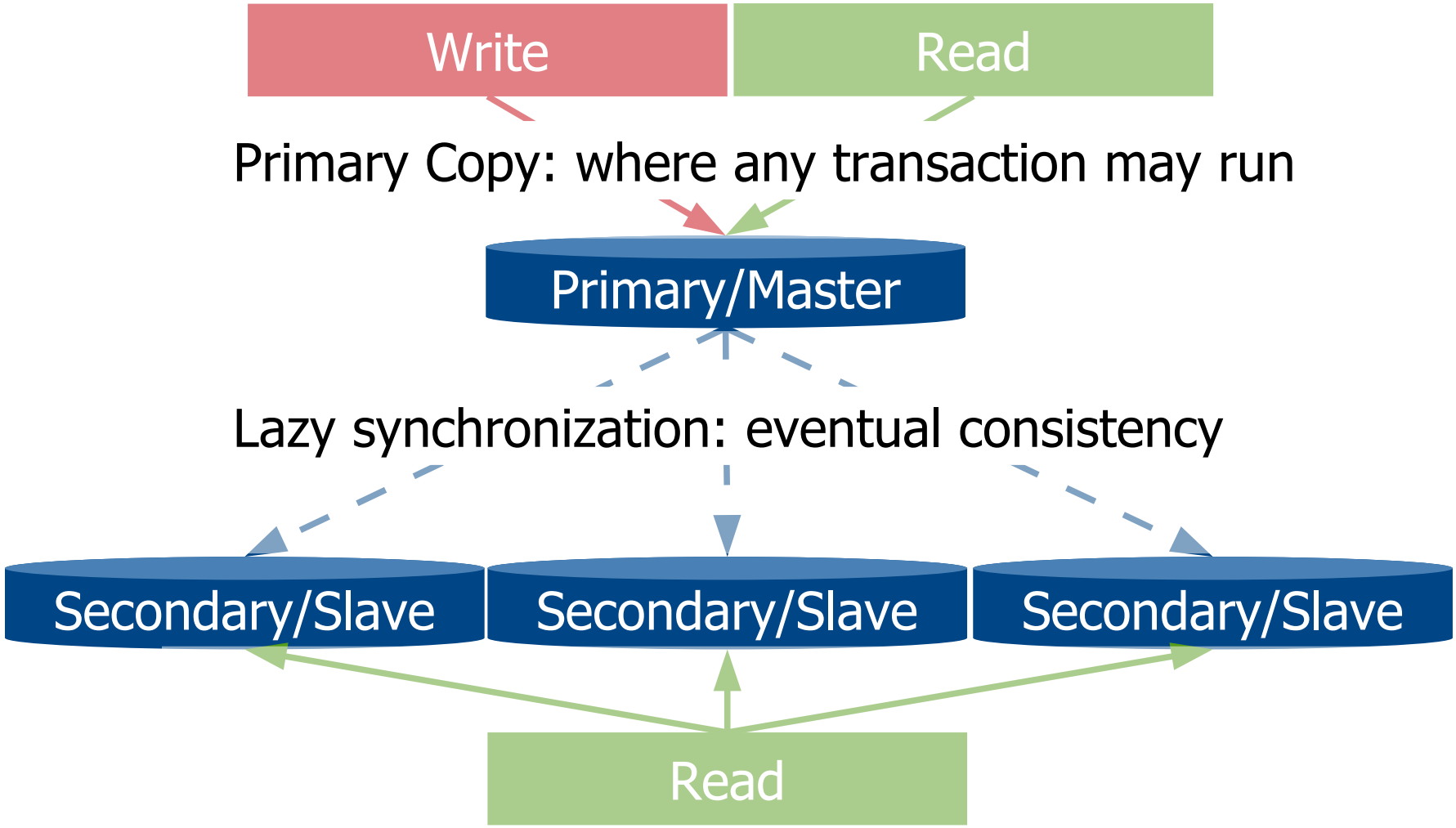
The speaker says…

A wide range of clusters can be categorized by asking **where transactions are run and when replicas synchronize** their data. Any eager solution ensures that all replicas are synchronized at any time: it offers **strong consistency**. A transaction cannot commit before synchronization is done. Please note, what it means to transaction rates:

- Single computer tx rate ~ disk/fsync rate
- Lazy cluster tx rate ~ disk/fsync rate
- Eager cluster tx rate ~ network latency (RTT)

Test: Would you deploy a synchronous (eager) cluster on a WAN, or prefer using an asynchronous (lazy) solution?

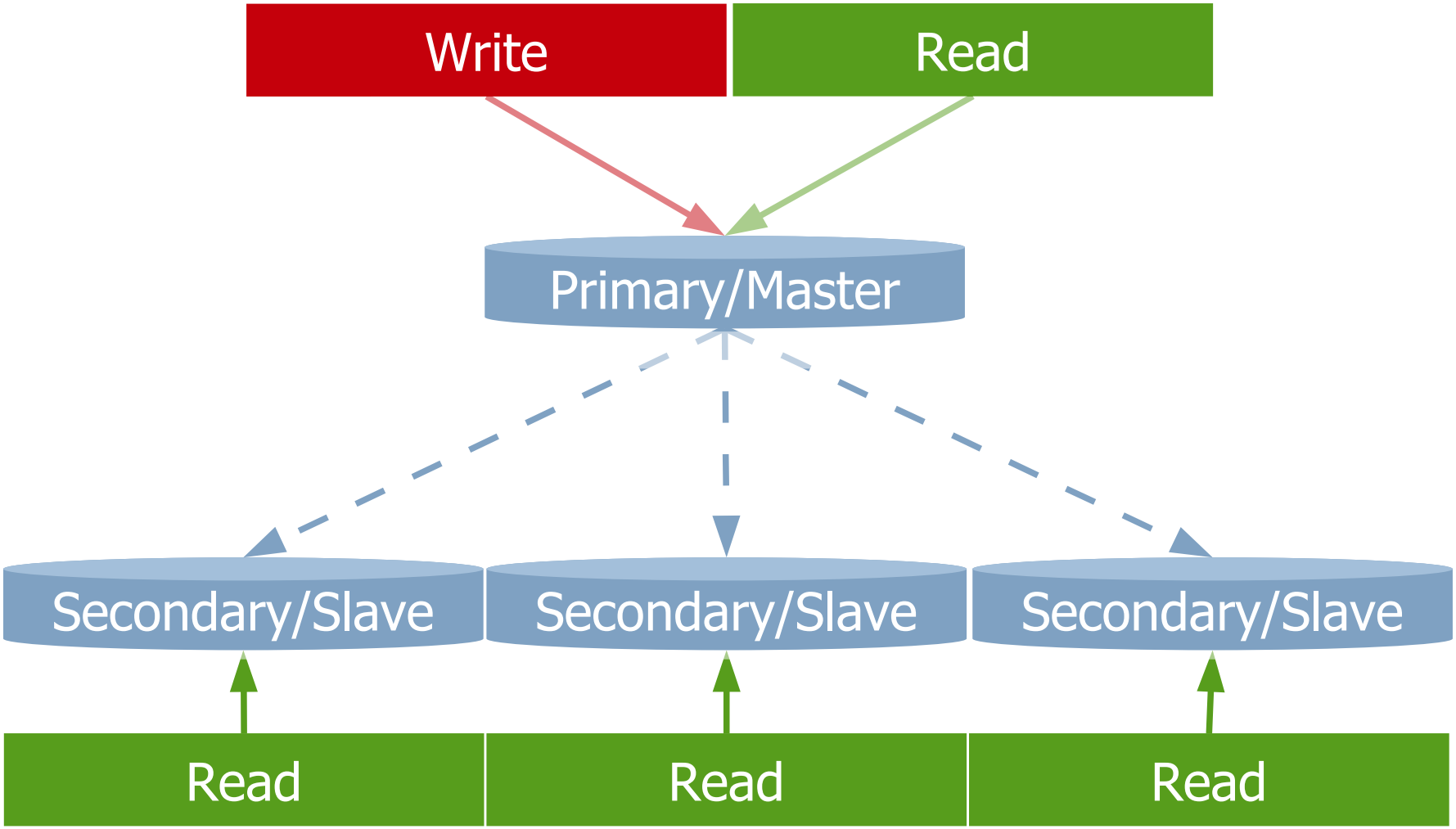
Lazy Primary Copy



The speaker says…

MySQL Replication falls into the category of lazy Primary Copy clusters. It is a rather **unflexible** solution as all updates must be sent to the primary. However, this **simplifies concurrency control** of conflicting, concurrent update transactions. Concurrency control is no different from a single database. Lazy replication **can be fast**. Transactions don't have to wait for synchronization of replicas. The price of the fast execution is the **risk of stale reads** and eventual consistency. **Transactions can be lost** when the primary crashes after commit and before any copy has been updated. (Workaround: MySQL semi-sync replication, which delays the commit until delivery to copy. Alternatively, use shared disk and standby.)

Read but not write scale out



The speaker says…

Primary copy is **best suited for read dominated workloads**, as commonly found in web applications. Write scale out is not possible. **Eager update anywhere** solutions offer **some write scale out** because writes can execute in parallel on multiple nodes but must then be synchronized for commit (commit rate ~ network latency). **Partial replication is the only way to scale writes**. Every write adds load to the entire cluster, regardless whether concurrency control involes all replicas (ROWA), or a good number of them (Quorum). Every additional replica adds load to all others. The solution is to partition the data set and keep each partition on a subset of all replicas only. **See MySQL Cluster or NoSQL.**

MySQL (NDB) Cluster

Availability

- Shared-nothing, High Availability (99,999%)
- WAN Replication to secondary data centers

Scalability

- Read and write through partial replication (partitioning)
- Distributed queries (parallize work), real-time guarantees
- Focus In-Memory with disk storage extension
- Sophisticated thread model for multi-core CPU
- Optimized for short transaction (hundrets of operations)

Distribution Transparency

- SQL level: 100%, low-level interfaces available

The speaker says…

MySQL Cluster uses partial replication for read and write scale out. It follows a hybrid system design and offers **eager update anywhere**: each node can answer read and writes questions. Replication is synchronous. Tables are automatically partitioned. The **sharding is transparent at SQL level**. Originally developed and optimized for Telco applications, Cluster never gained the popularity of InnoDB for web applications. **NDB ain't InnoDB, and Jo Doe voted InnoDB for the web.**

With classic MySQL Replication and InnoDB hitting the write scale out limit, **MySQL Fabric was born: middleware based sharding for MySQL.**

MySQL Fabric = Replication++

Availability

- Primary Copy with external heartbeats
- Automatic MySQL 5.6.10+ GTID based failover/switchover
- Vision: zero transaction loss (semi-synch replication)

Scalability

- Read scale out through slaves/copies
- Read and write scale out using table based sharding
- (Nodes and shard administration)

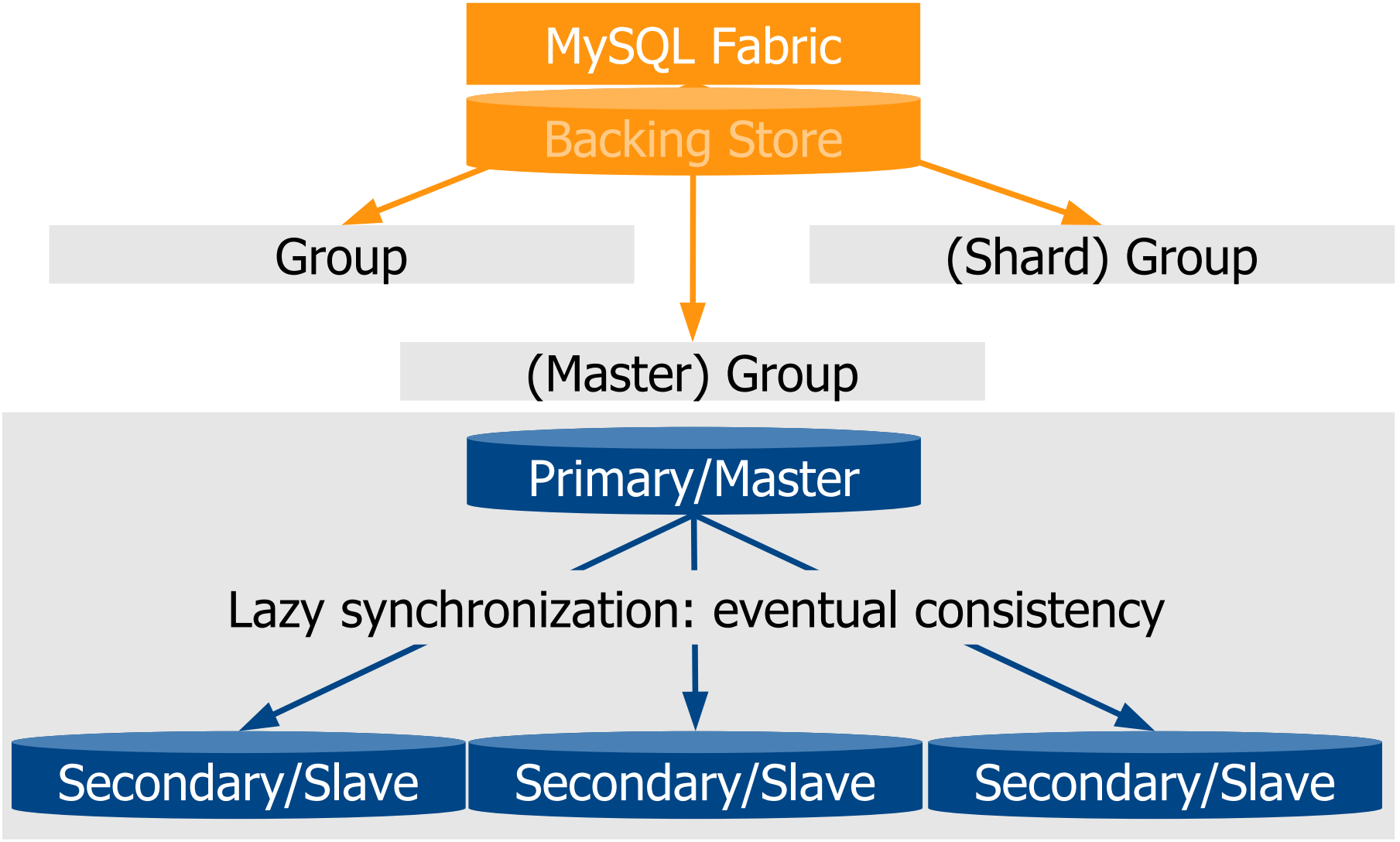
Distribution Transparency

- Low, and no area of focus
- Some Java, Python and – most limited - PHP driver support

The speaker says…

MySQL Fabric 0.3 pre-production is a super-sized management tool for MySQL Replication. Thus, it inherits all major MySQL Replication properties. Only addition: horizontal partitioning (sharding). Technically, the initial version is barely more than a set of Python scripts. **Look beyond, note the change in thinking, grasp the ideas.** Stay tuned for critique. Years after it has become mainstream, **MySQL Replication learns sharding for read and write scale out.** Welcome back, old lady Sakila (MySQL dolphin/mascot)! It is the first time ever that **MySQL ships MySQL Replication HA out of the box.** It is the first time ever that **MySQL aims to automate management of nodes**.

Middleware Replication++

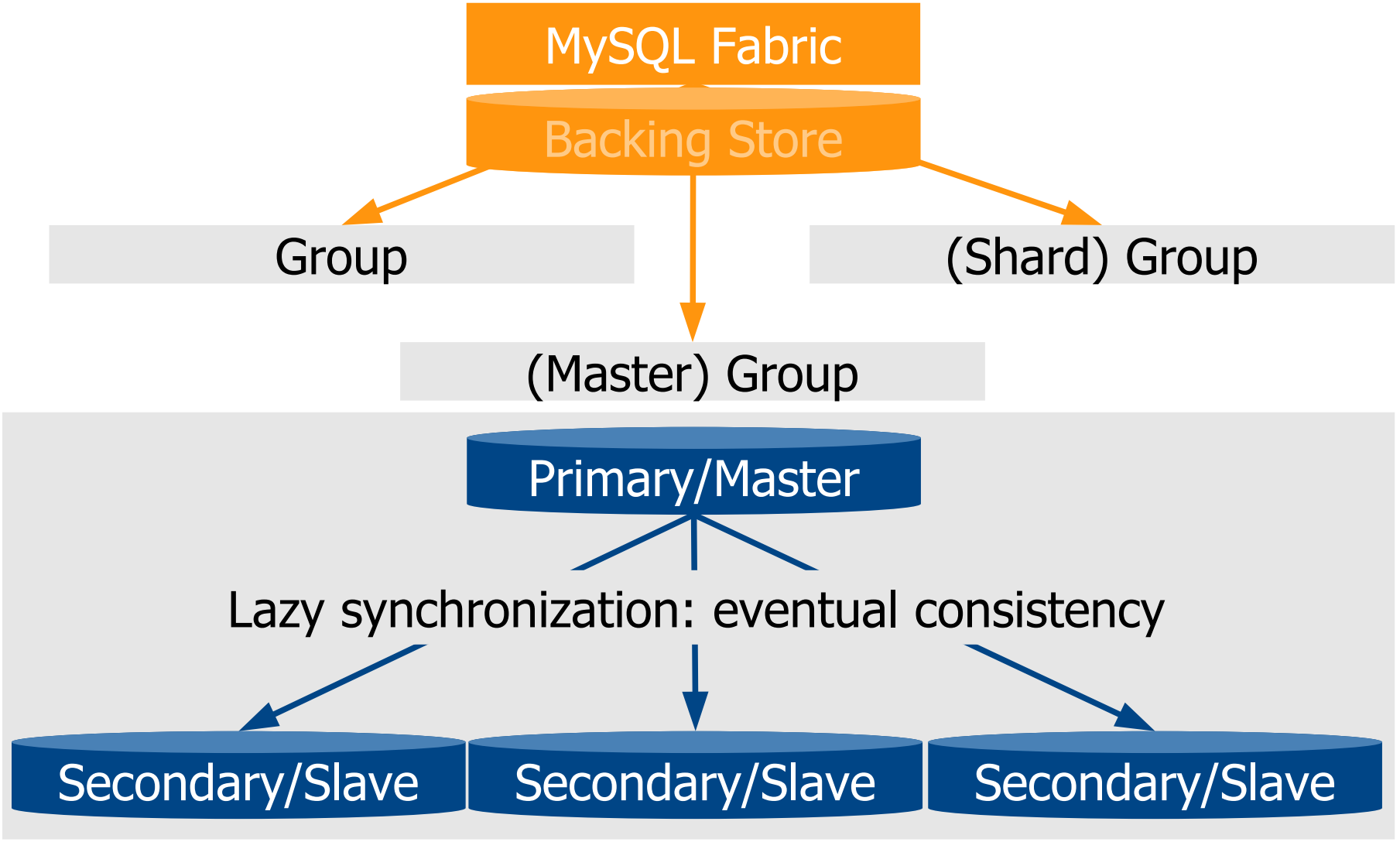


The speaker says…

Basics first. **MySQL Fabric is daemon for managing farms of MySQL servers**. Farms consist of „groups“. A group either consists of any number of individual MySQL servers, or holds a MySQL Replication cluster. A group describing a replication cluster consists of a master and any number of slaves, as ever since.

MySQL Fabric can setup, administrate and monitor groups. Once a MySQL Server has been installed, Fabric can take care of the replication setup details. DBAs might, for example, use virtual machine images to add new MySQL Servers, whenever needed. Then, Fabric is used to integrate those servers into the replication cluster. **For example, integrating a new node boils down to one line on the CLI. More later, it is getting better.**

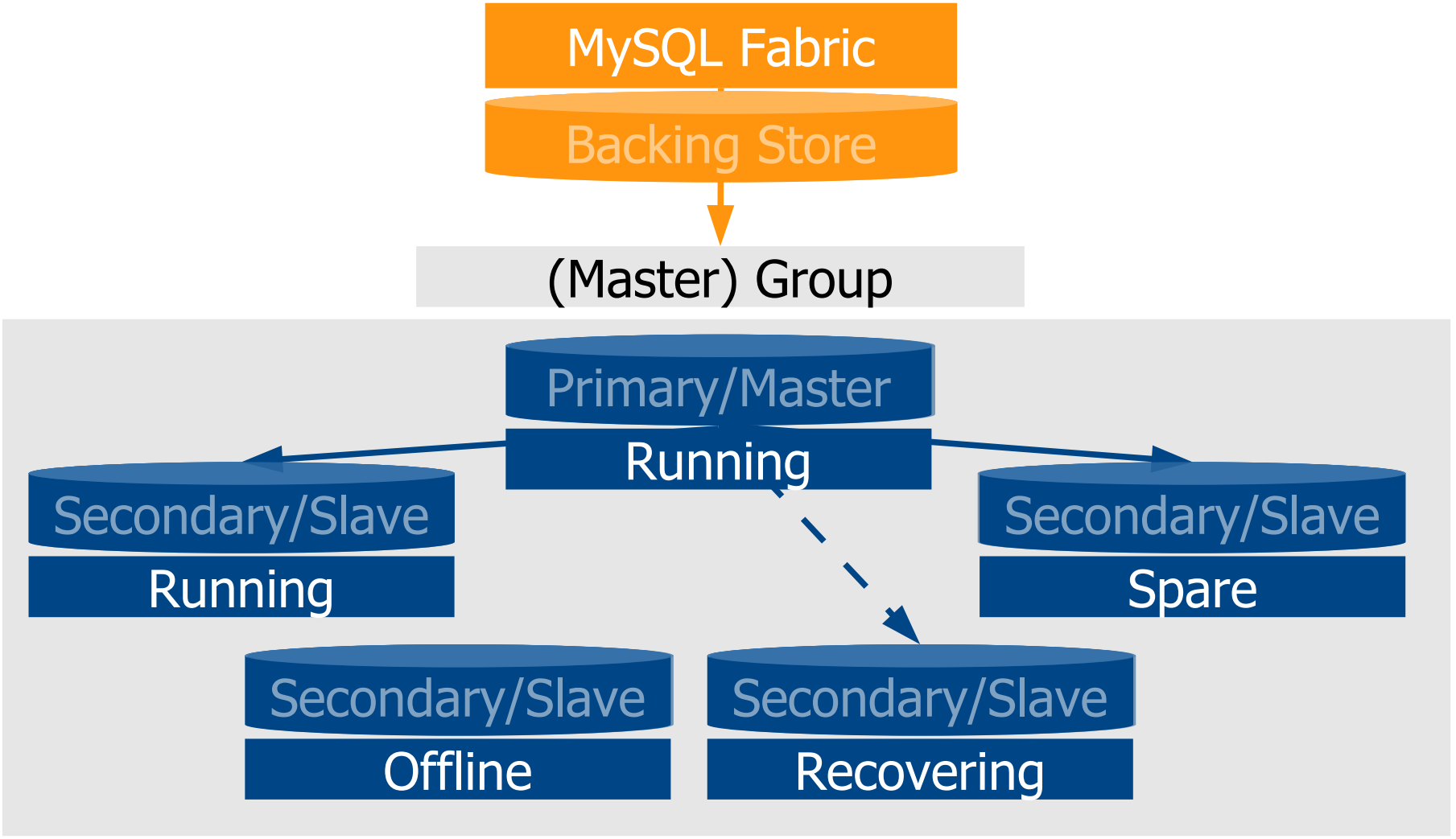
Middleware Replication++



The speaker says…

Basics first. **MySQL Fabric is daemon for managing farms of MySQL servers**. Farms consist of „groups“. A group either consists of any number of individual MySQL servers, or holds a MySQL Replication cluster. A group describing a replication cluster consists of a master and any number of slaves, as ever since. **MySQL Fabric can setup, administrate and monitor groups.** Once a MySQL Server has been installed, Fabric can take care of the replication setup details. DBAs might, for example, use virtual machine images to add new MySQL Servers, whenever needed. Then, Fabric is used to integrate those servers into the replication cluster. **For example, integrating a new node boils down to one line on the CLI.** More later…

A classic setup

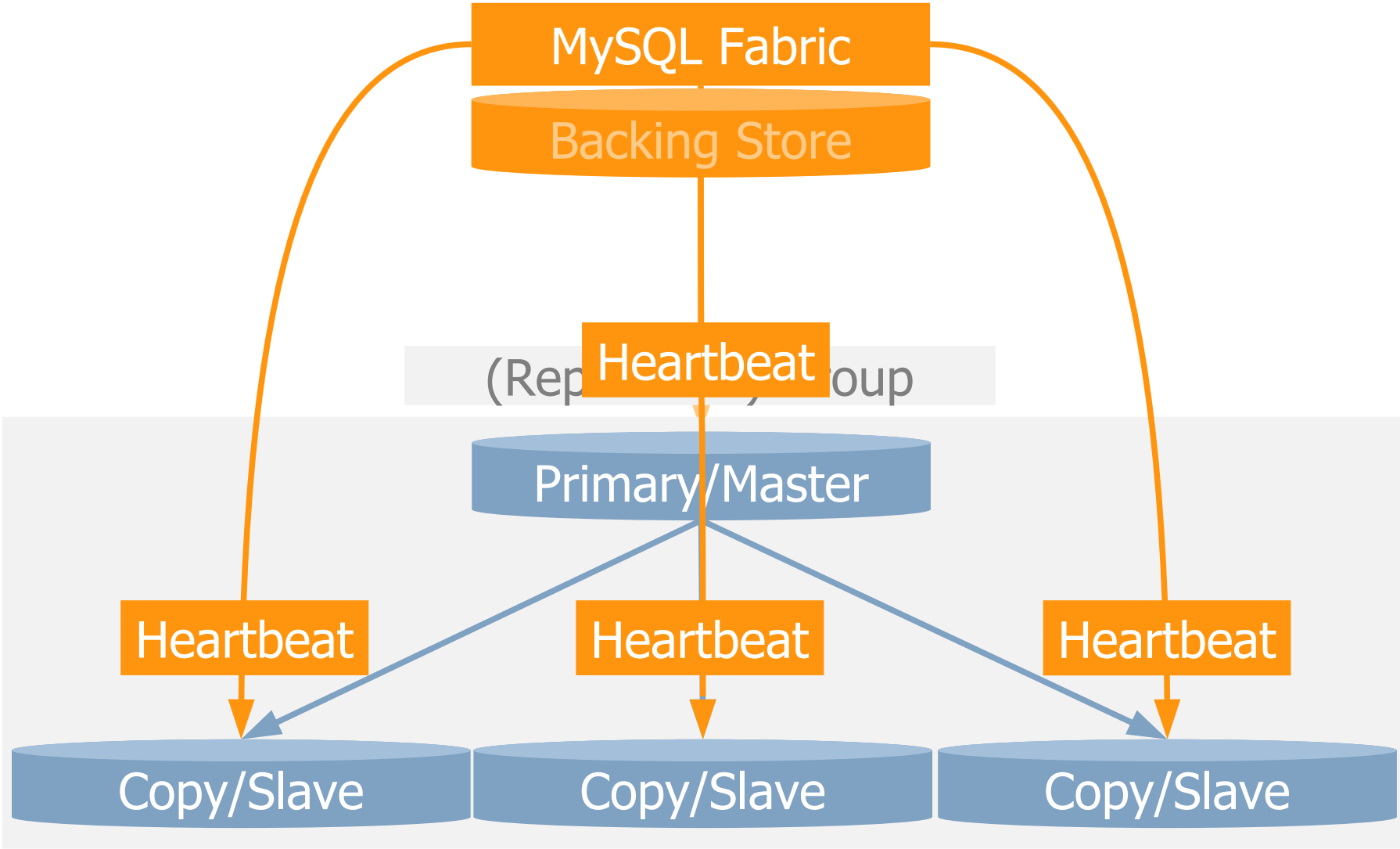


The speaker says…

A most basic farm managed by Fabric might contain only one master group. **A master group is a logical name for a number of servers belonging to a standard MySQL Replication cluster.**

The servers in the master group are in one of five operational states: **running, spare, offline, faulty and recovering.** Fabric aware drivers send questions to running servers only. Spare servers pariticipate in replication but do not handle queries until turned online. Spare servers shall help to quickly grow groups, for example, during periods of peak loads.

Availability: automatic failover

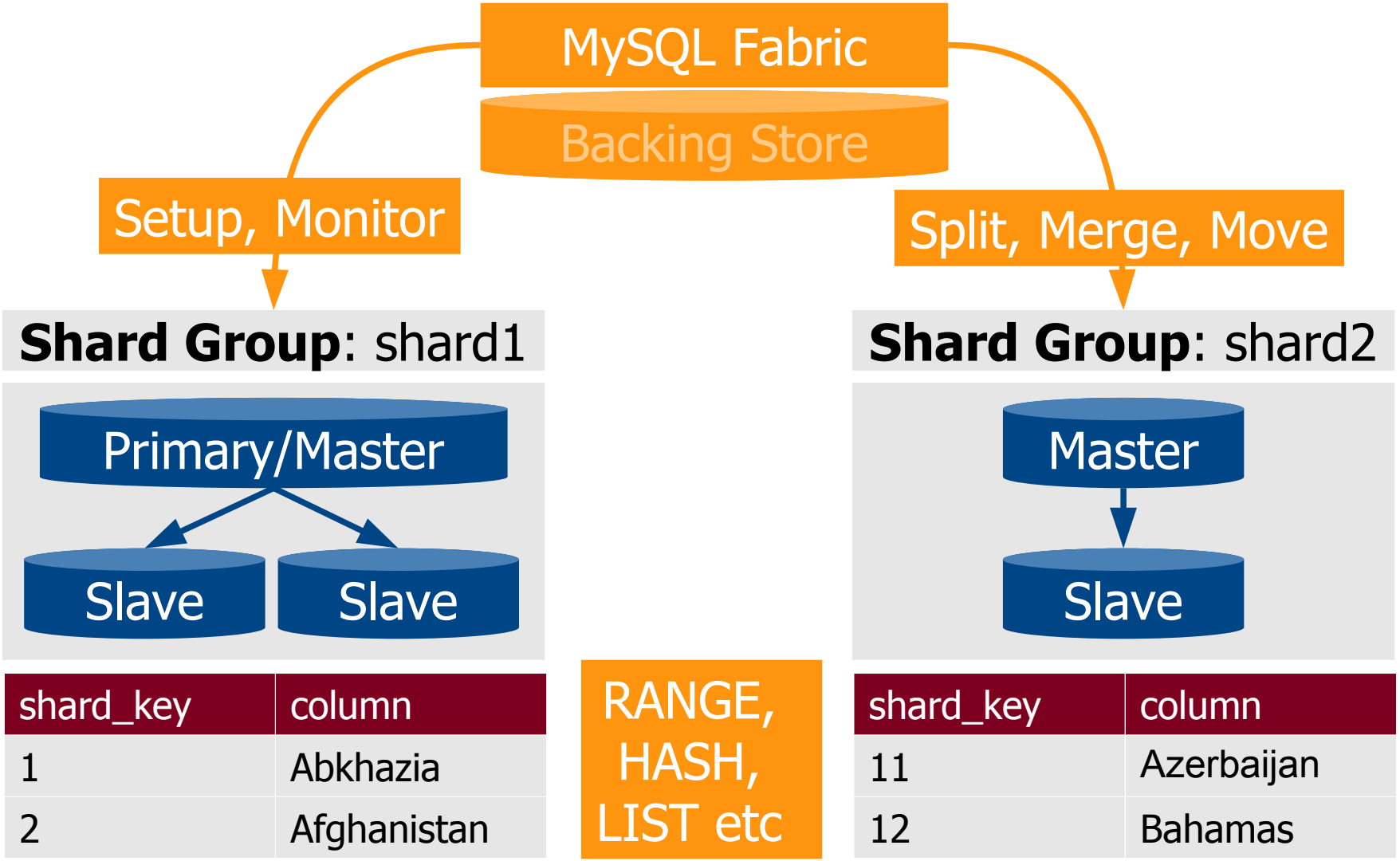


The speaker says…

Failure detectors can be used to detect node failures. The built-in failure detector uses heartbeats. If a master fails, automatic failover can be performed. The system will search for the most up-to date slave, promote it to the new master and reconfigure all remaining nodes to start replicating from the new master. Failover is based on MySQL 5.6+ GTID logic, which means you have to use MySQL 5.6.10 or newer.

Clients can ask MySQL Fabric for a list of nodes. Hence, they **can deploy themselves** and automatically explore nodes. A **long lasting** Connectors team dream to improve distribution transparency comes true. More below.

Scalability: sharding



shard_key	column
1	Abkhazia
2	Afghanistan

RANGE, HASH, LIST etc

shard_key	column
11	Azerbaijan
12	Bahamas

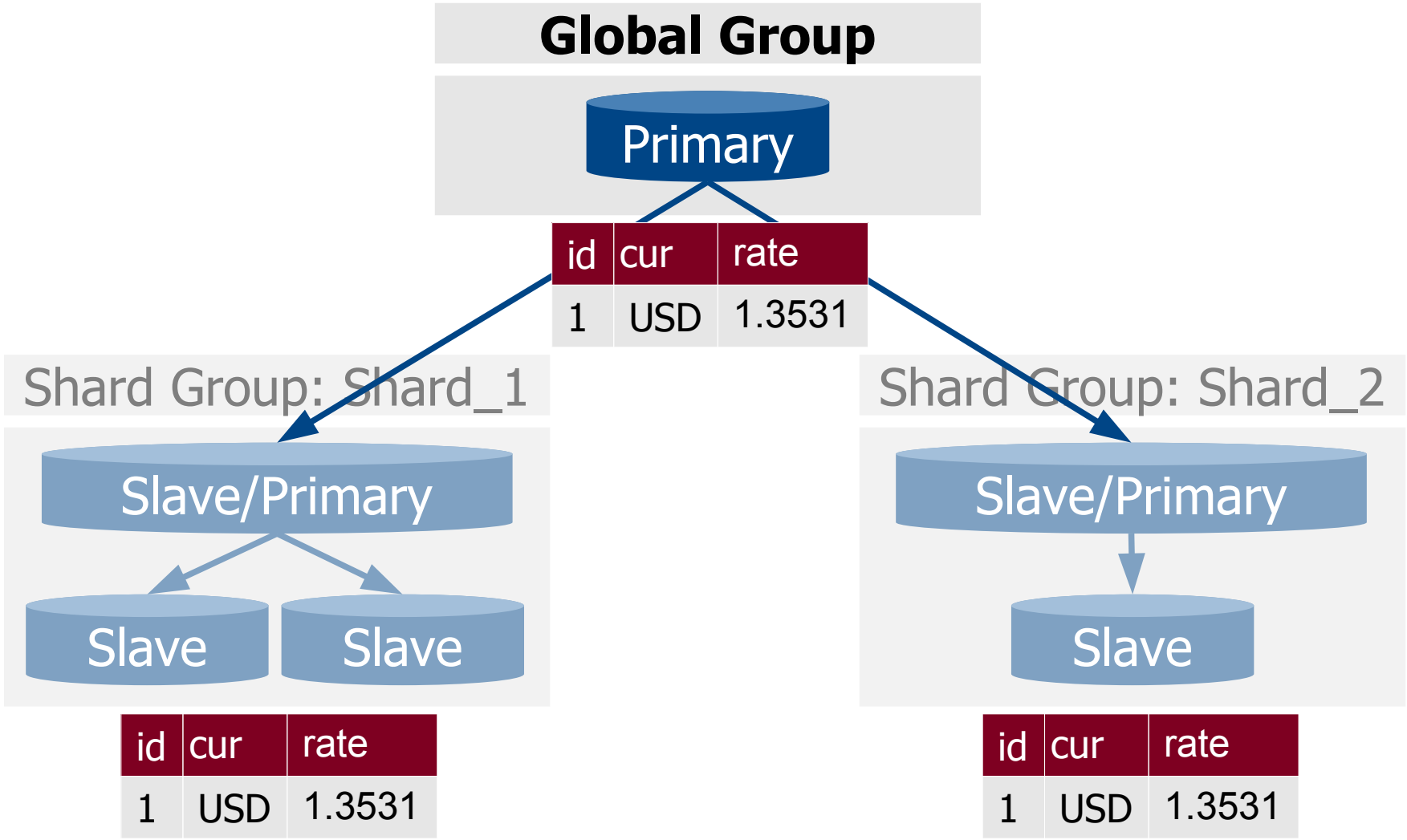
The speaker says…

Fabric supports range, hash or list based partitioning of tables using one column as a shard key. Each partition is assigned to a logical shard group, short: shard. Recall, a group consists of an individual server, or forms a replication cluster in itself.

There are Fabric commands for defining the sharding rules (shard mappings), for assigning nodes to shards, for populating shards from unsharded database servers, for splitting shards, for merging shards, and for moving shards.

Clients can ask MySQL Fabric for a list of nodes and sharding rules. Given a shard key, clients can route requests to the appropriate servers.

Schema updates, global tables



The speaker says…

A global group can be defined to replicate global tables to all shards and to manage schema changes to partitioned tables. Updates to global tables and DDL operations on partitioned tables are performed on the global group. Then, all shards replicate from the global group to copy the changes.

Clients ask Fabric where to send global updates and route their requests to the appropriate servers.

The DBAs view on Fabric

New mysqlfabric command line tool

- Central administration tool
- Easy for you to integrate in your favourite deployment tool
- Easy for us to integrate into our admin/management GUIs

Extensible HTTP XML RPC interface

- No SSH access required for remote deployment
- Power users may add custom commands long term

Self-deploying clients

- Use „fabric aware" drivers, or improve your clients

The speaker says…

A major goal was to create an extensible, flexible base which integrates smoothly in existing deployments. We currently do not offer integration in our own free GUI administration tool MySQL Workbench and our own commercial GUI management tool MySQL Enterprise Monitor. However, look at the architecture and draw your own conclusions.

As I expect mostly developers not DBAs reading this, and the **pre-production** release tends to use basic methods of performing actions (3rd-party: Call for Patches and Branches is open ;-)) , I skip further details.

Replication with auto failover

```
> # Install MySQL servers, edit Fabric config, setup MySQL backing server for Fabric
> mysqlfabric manage start
> # Create group to manage master/slave replication
> mysqlfabric group create my_master_group
> # Assign servers to master group
> mysqlfabric group add my_master_group mysql_host mysql_user mysql_password
> …
> # Choose primary, start replication
> mysqlfabric group promote my_master_group
> # Add heartbeating for automatic failover
> mysqlfabric group activate my_master_group
```

The speaker says…

The slide shows an example of setting up a standard MySQL Replication with heartbeating and automatic failover. Compared with MongoDB replica set deployment:

	MongoDB	MySQL
1)	Setup mongod's	Setup mysqld's
2)	On any node, create replica set	Using mysqlfabric, create master group
3)	On primary, add secondaries	Using mysqlfabric, add nodes, choose primary
4)	Built-in failure detector automatically activated	Choose failure detector, activate. E.g., use built-in.

Server setup for sharding

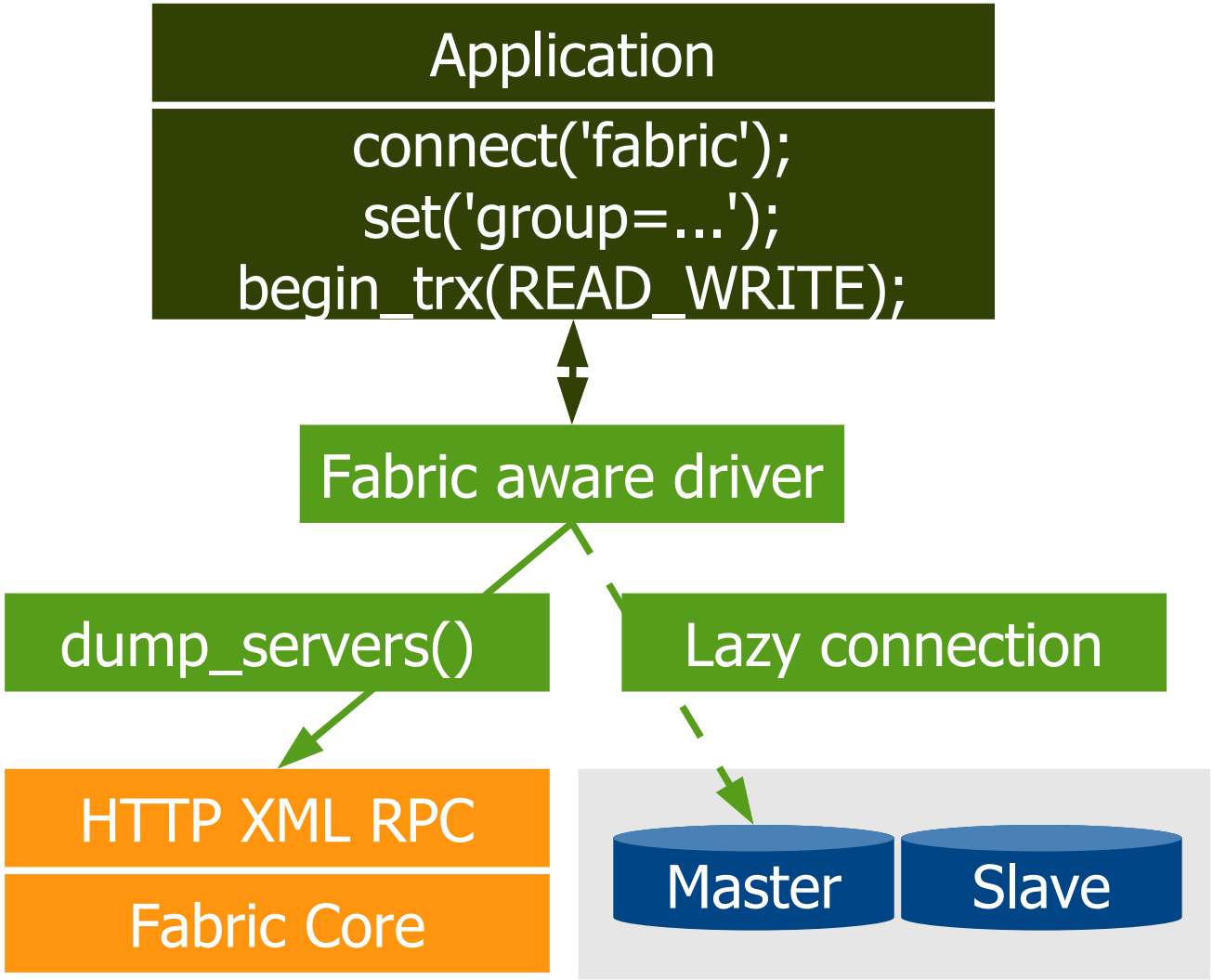
```
> # Create one shard group per partition, add servers
> mysqlfabric group create shard1
> mysqlfabric group add shard1 host user password
> …
> # Define sharding rules/mapping
> mysqlfabric sharding define RANGE s1_tmp
> # Which table to shard and by what column
> mysqlfabric sharding add_mapping 1 db.table column
> # Connect shard group with mapping rules
> # Group shard1 shall be used to hold db.table
> # with shard key column and values 1...100
> mysqlfabric sharding add_shard 1 1 100 shard1
enable
```

The speaker says…

Setting up sharing takes a bit longer as you have to grasp the relation between a global group, shard mapping and shards. Compared with MongoDB shard deployment:

	MongoDB	MySQL
1)	Setup mongod's	Setup mysqld's
2)	Create shards: single mongod or replica set	Create shard group: single mysqld or replication
3)	If sharding for collection: define key for db.collection	Create mapping: define key for db.table
4)	Add shards to cluster, set distribution details	Add shards to cluster, set distribution details

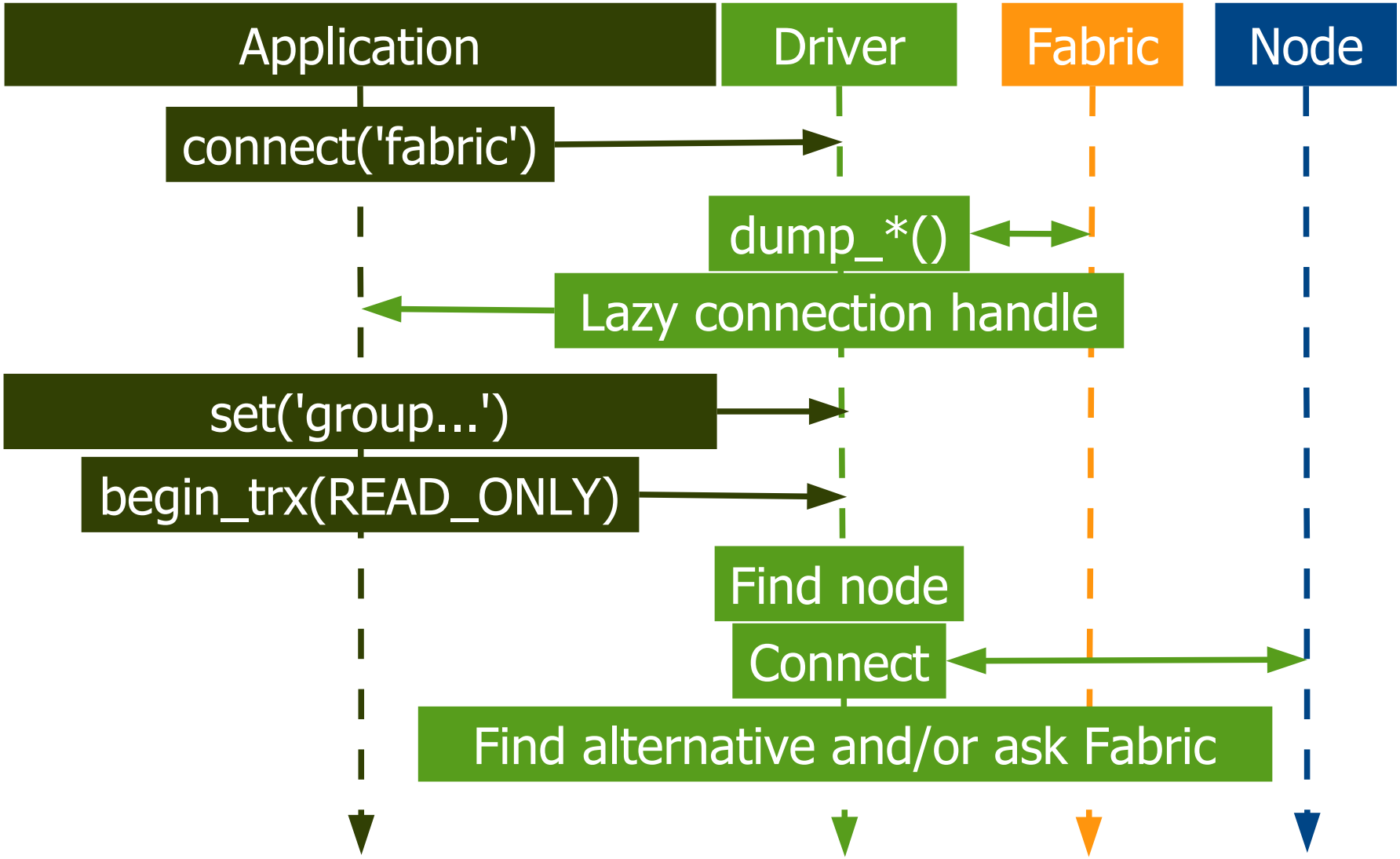
How fabric aware clients tick…



The speaker says…

Fabric aware drivers communicate with Fabric to learn about master groups, shard groups, global groups and their nodes. **Application developers think in terms of groups instead of individual servers.**
If, for example, an application requests use of a master group, **the driver asks Fabric for a list of all nodes using the dump_servers() XML RPC call**. Then, it returns a connection handle to the application. At this point, no connection to any MySQL server has been established yet. The driver uses **lazy connections** to delay the actual connect until it knows about the transaction/query to choose an appropriate server. In the example, the master would be used to run a read-write transaction.

Client failover – in theory..



The speaker says…

Assuming a Fabric aware driver recognizes a node failure when not in the middle of a transaction, the **driver can automatically pick an alternative** node without raising an application error. The driver knows about possible alternatives. **Additionally, the driver can hint Fabric that a node has failed.** If no alternative is available, the **driver might want to ask Fabric for an updated list of nodes to deploy itself**. Maybe, in the case of a master failure, Fabric fixed the problem already and has promoted a slave to be the new master. Then, failover and reconfiguration would be transparent from an applications point of view. **At the time of writing, only Python supports this – in parts!**

Client failover – today

Virtual IP

- Clients use virtual IP, virtual IP moved during failover
- Often used for masters only

Automated client configuration deployment

- Server monitoring tool deploys client configuration
- Example with PHP and 3rd party MHA (Master High Availability Manager tool): http://blog.ulf-wendel.de/2013/peclmysqlnd_ms-let-mha-update-your-client-configs/

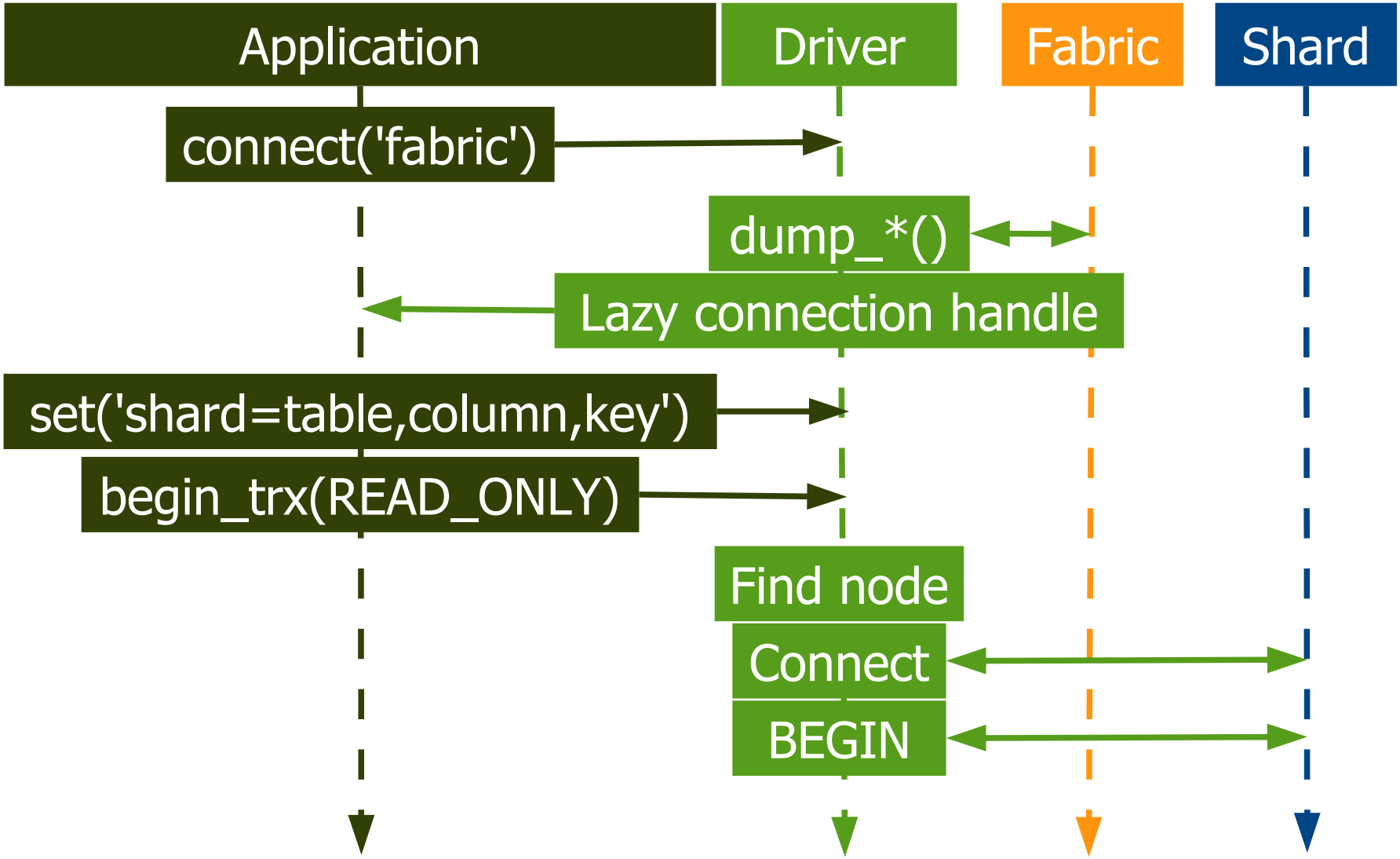
The speaker says…

MySQL Failover with zero, manual client deployment is possible already. It is, however, usually restricted to master failover, and there is **no out-of-the box solution**.

A true classic is using a virtual IP. Clients connect to the MySQL master through a virtual IP. In case of a MySQL master failover – or, planned switchover -, the DBA moves the virutal IP from one server to another. The change is transparent to the clients.

A pattern similar to MySQL Fabric is also possible using a MySQL monitor tool to deploy client configuration files.

Ex-bound shard key

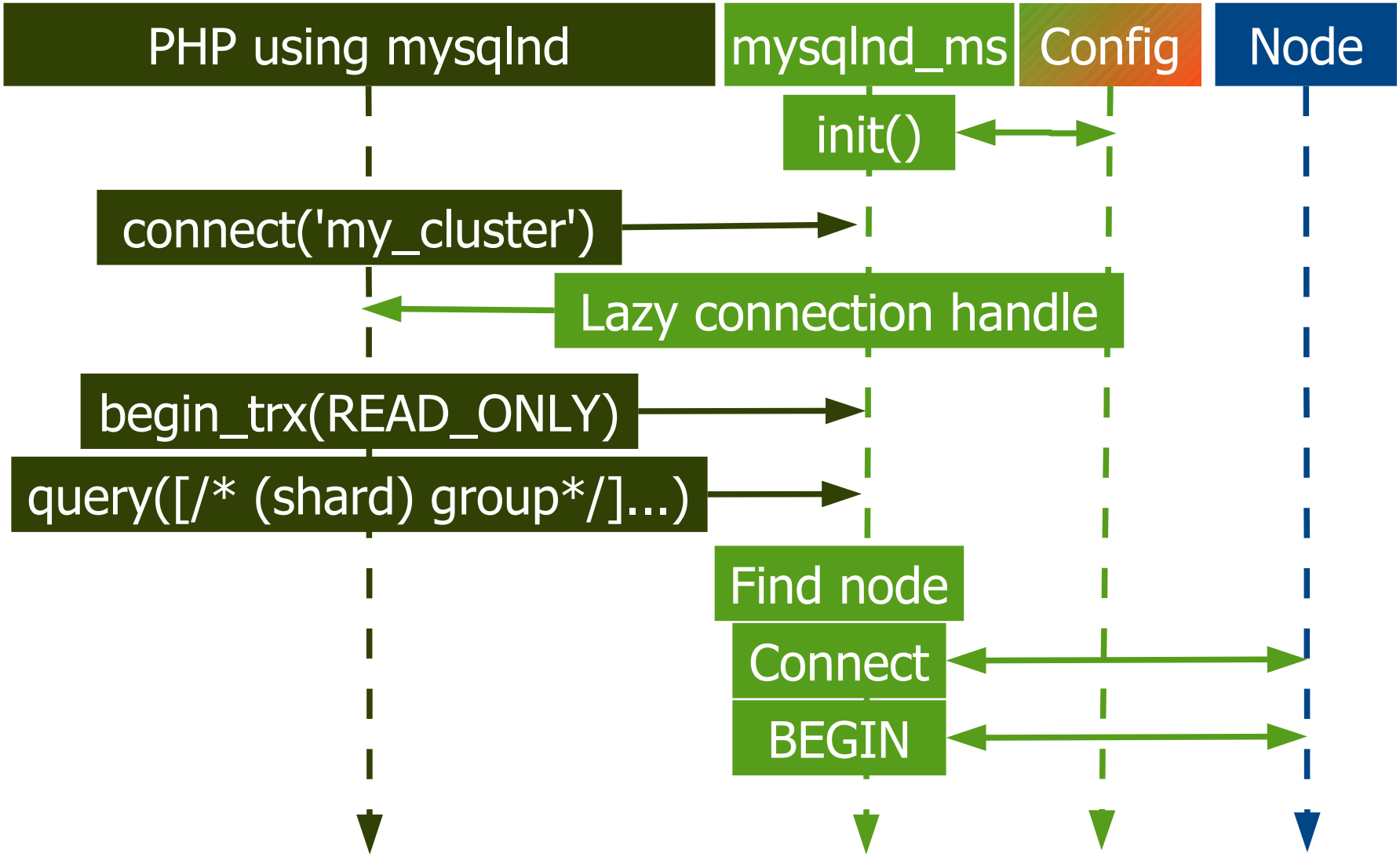


The speaker says…

Sharding works very similar.

The application connects to Fabric instead of connecting to an individual server, and the driver learns about the nodes in the server farm. Then, the application hints the driver to choose a shard (node). At this point the driver establishes a connection to the node – if none exists – and routes the requests to it.

MySQL Manual Myth Buster



The speaker says…

After studying the MySQL manual you might believe, the **client part is an all new, sensational invention. Myth!** Think **PECL/mysqlnd_ms (04/2011)** with a config stored on an network server instead on the local file system. The current, **pre-production version of Fabric lacks**:

- state alignment for lazy connections (09/2011)
- quality of service concept: consistency abstraction, maximum age and result caching, ... (11/2011)
- weigthed load balancing, locality support (07/2012)

We still lack server support for these. Will we get it ;-) ?

MMM Buster!

To utilize Fabric in your application, you must have a Fabric aware connector *installed on the system where the application is run […] Fabric aware connector:*

- *Connector/Python 1.1.1a2 or later*
- *Connector/J 5.1.26 or later*

True or false?

- Of course, you could do the XML RPC yourself
- Sometimes, there is no Fabric aware connector
- The XML RPC interface is rather simple to use

The speaker says…

The reasoning behind this statement is crystal clear. Distributed database designs with low distribution transparency, can improve the developers experience using intelligent load balancers. For example, using a driver integrated load balancer. There are valid reasons for low distribution transparency.
PECL/mysqlnd_ms came with two messages:

- using any MySQL cluser becomes easy, and the documentation shows usage and algorithm
- all default decisions can be overruled

Following educate and enable users: Fabric can be used – less comfy – without special driver from MySQL.

MMM Buster!

A master group is a group of servers that together work to provide a high-availability master using reundancy. In the master group, one of the servers is the primary, while the others are secondaries. As long as the ***primary*** *is alive and running, it* ***will handle all the queries, while the secondaries replicate everything from the primary to keep up to date****.*

True or false?

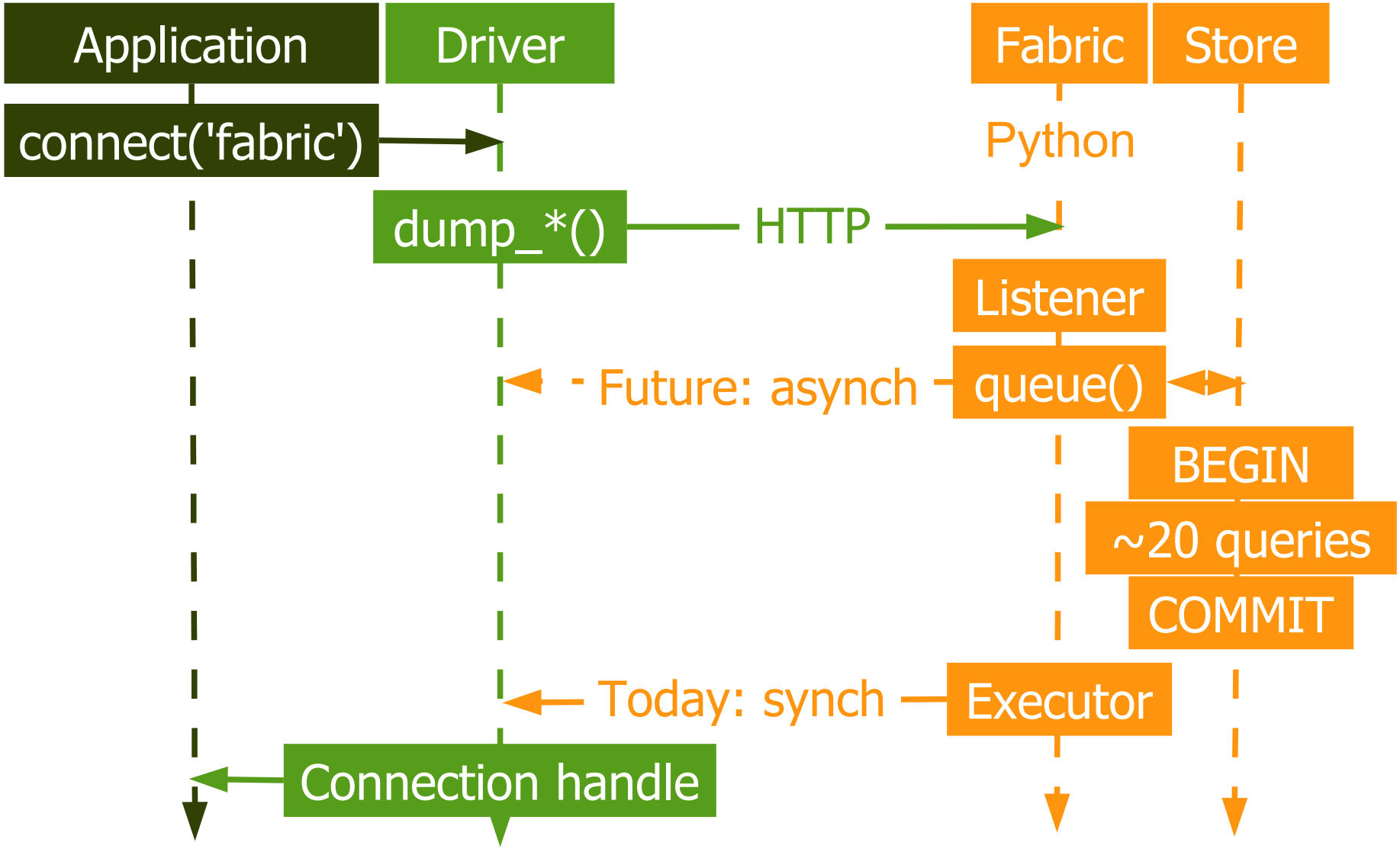
- Sometimes people say: query (write), question (read)
- No, Fabric is not about scale in
- Load shall be spread over all runing nodes of any kind

The speaker says…

I am no native english speaker. Is it me only who has to smile when reading secondaries (slaves) do not participate in handling queries? Research literature makes a distinction between queries (write) and questions (read). **Of course, client load shall distribute over all running nodes of any kind.** In general, I find the manual a bit hard to understand as it sometimes uses – correctly – different terms for basically the same thing. Same for my presentation, which sticks to Fabric wording. Compare with:

Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves).

Performance considerations

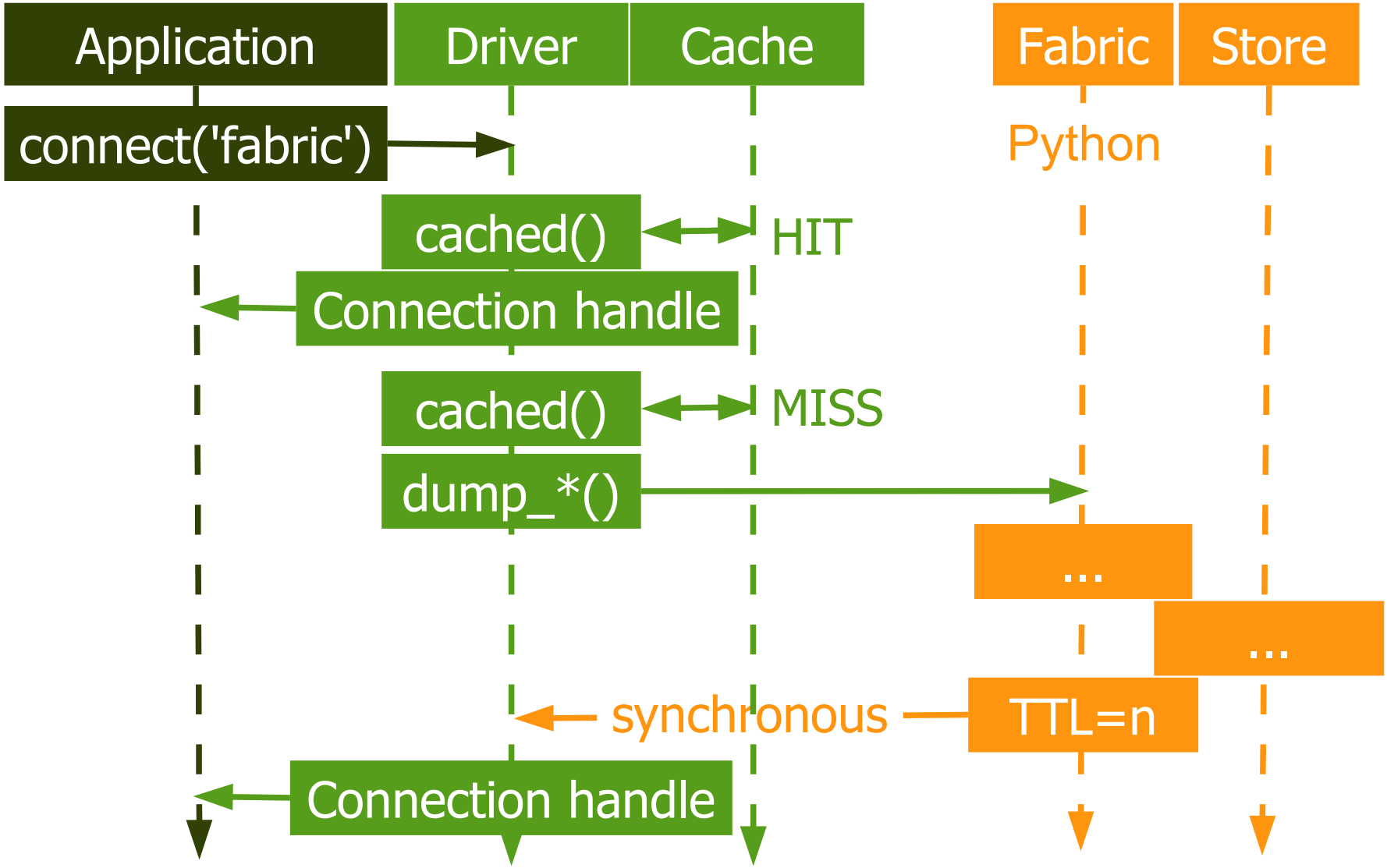


The speaker says…

Fabric HTTP XML RPC is expensive. Remote procedure calls add latency to applications using Fabric. HTTP and XML are verbose protocols not optimized for performance. Multiple HTTP listeners and executors can be configured to scale Fabric on multi-core systems. However, Fabric is written in Python, and Python is not the fastest programming language.

A MySQL database is used by Fabric as a backing store to persist state. Don't nail me on when exactly Fabric persists state. If you set the Fabric log level to Info, you can see the queries run. Given this architecture, the Fabric RPC rate is less or equal to the maximum transaction rate of the backing store.

Clients must use caches



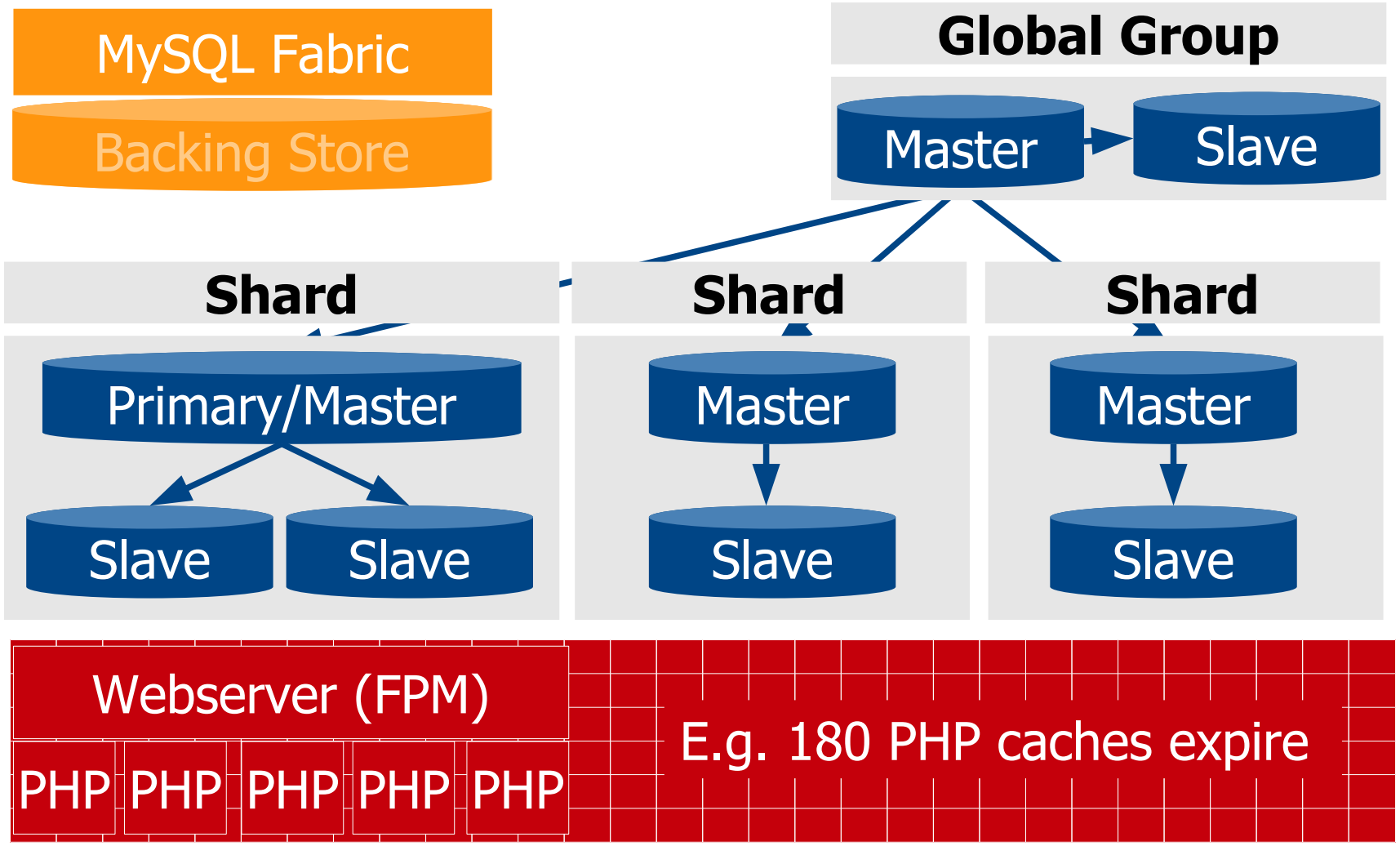
The speaker says…

To reduce latency clients should use pooled connections, and must cache XML RPC responses.

XML RPC response caches must use time-to-life expiration policy. The TTL is given by Fabric. The TTL is in the response of some calls, for example, all dump_*() calls.

Caching also helps to prevent Fabric server overload.

Will it scale?

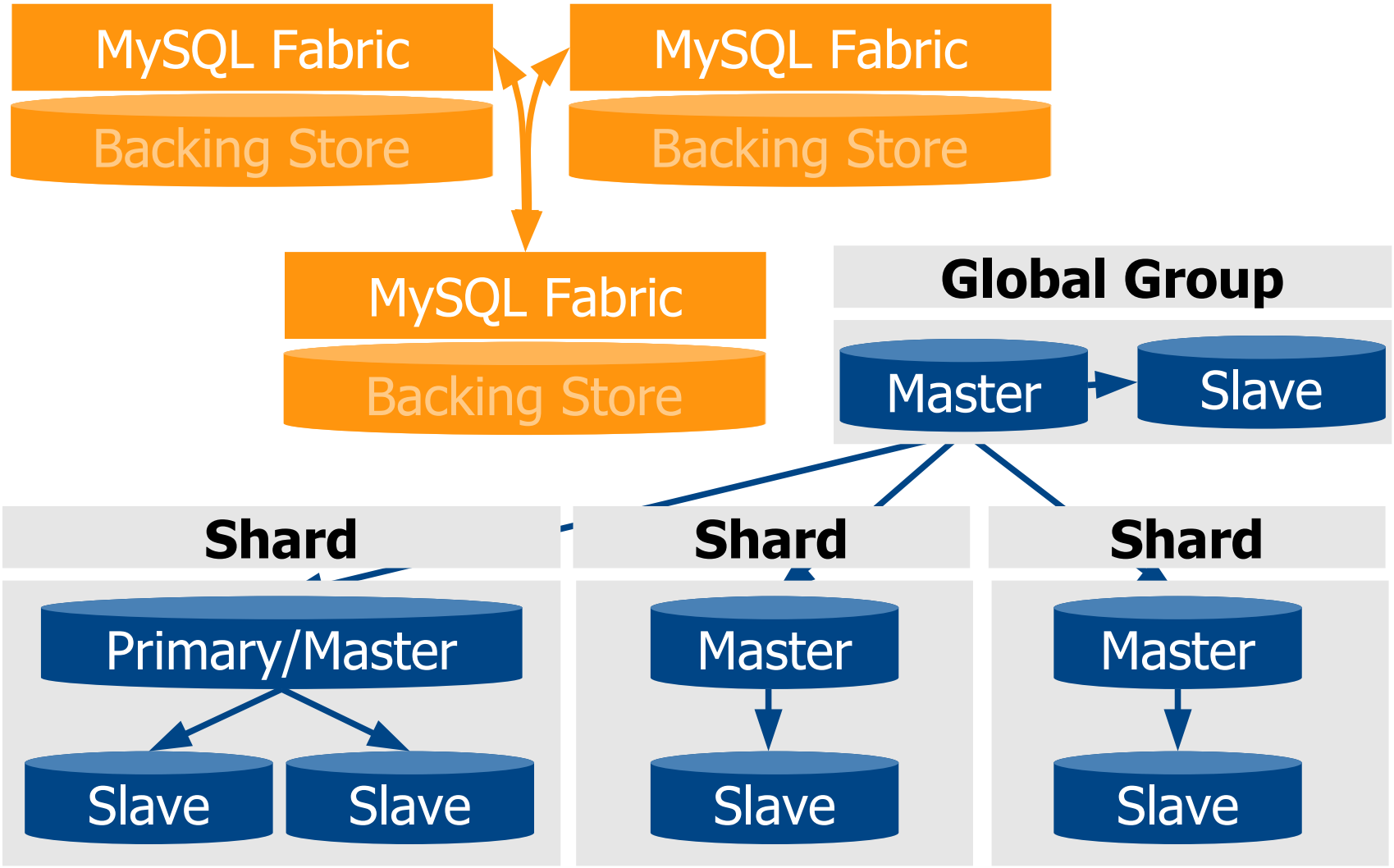


The speaker says…

There is a possibility that clients overload - the **initial** version of – Fabric. Assume you have a sharding setup with three shards and one global group, say 9 MySQL servers in total. Say4 webservers running each 5 PHP FPM processes per MySQL server. This is no figure picked out of the air but a ratio seen at a small web agency mostly deploying standard PHP solutions. This gives you 180 PHP processes. The best built-in cache store for a PHP driver is process memory. Means, up to 180 PHP caches could expire concurrently causing 180 concurrent XML RPC requests.

Assuming such a cluster consisting of machines like my notebook, Fabric became too slow. Note, I'm extrapolating, I make a guess and **its pre-production**…

It will scale – in the future

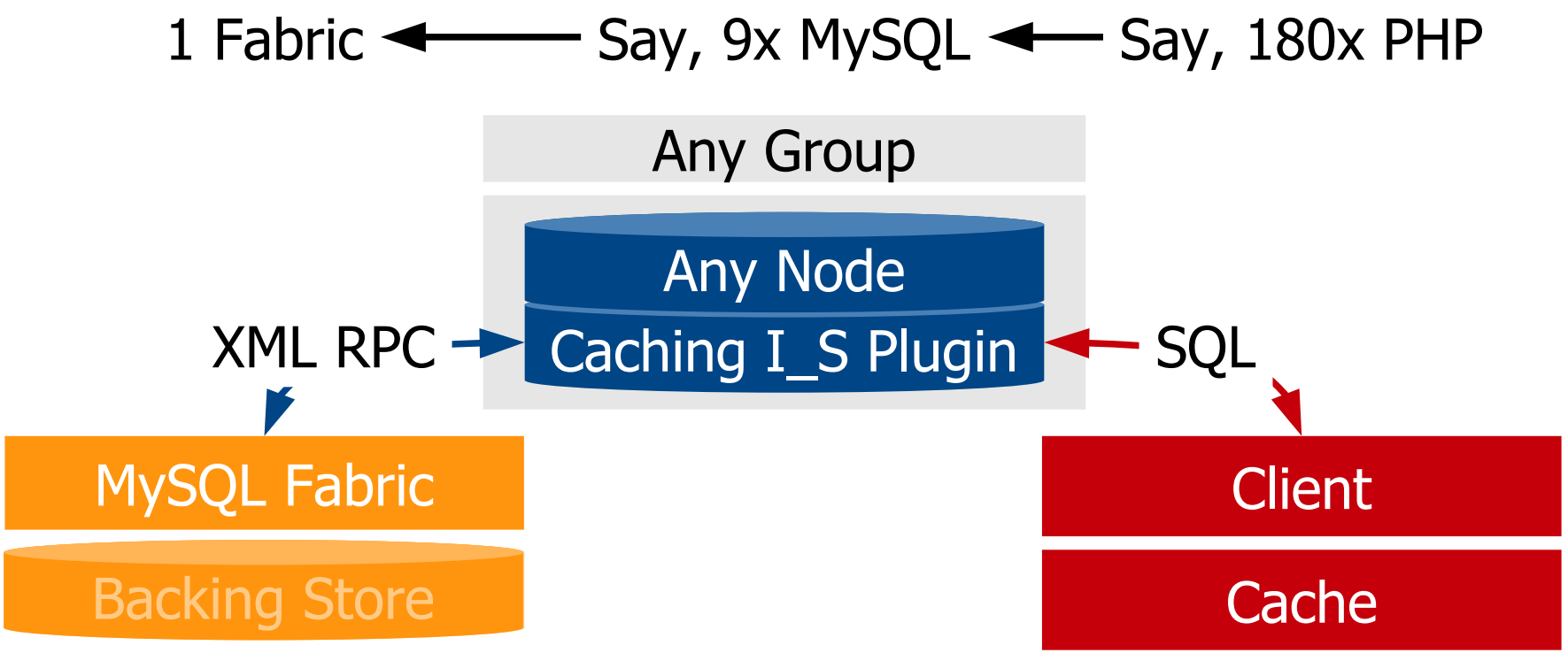


The speaker says…

MySQL was brave enough to release an initial version of a high availability solution where a Single Point of Failure (Fabric) monitors another SPOF (Primary/Master) to achieve high availability. Ha? Yes... Of course, SPOF cannot cure a SPOF!

The Fabric design supports running it as a replicated state machine (RSM). Replicated state machine is a chewy term that opens up for many options ranging from using a group communication system, raw Paxos or a synchronous, replicated database like MySQL Cluster as a backing store.

My vote: hierachical cache

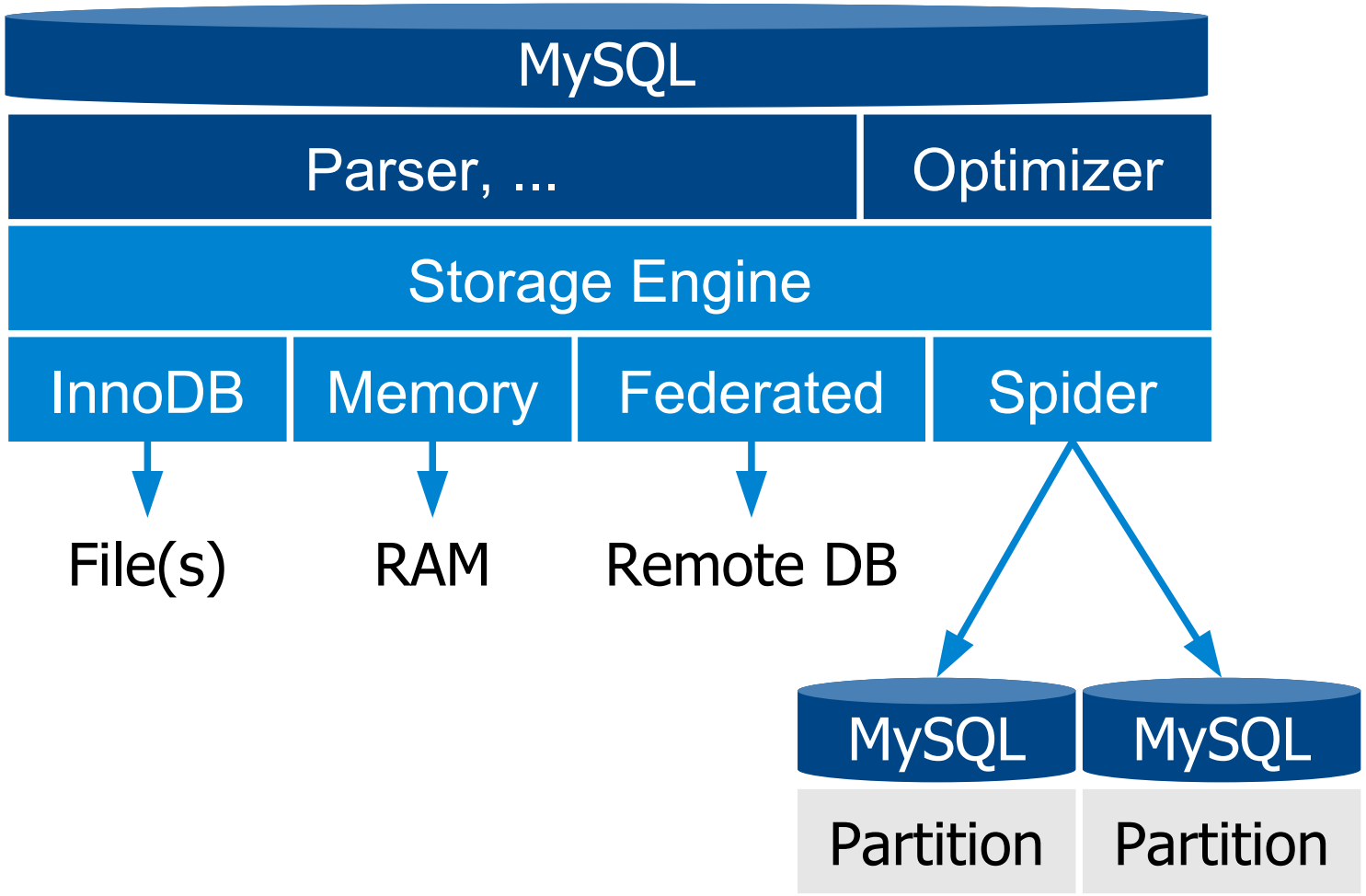



```
SELECT * FROM STORE_DUMP_SERVERS
WHERE pattern = '' AND version = 0
```

The speaker says…

The potential PHP Fabric overload scenario has much to do with the potential default cache medium of a Fabric aware PHP driver. The best, default cache medium is process memory. Reuse of cache entries is low, so is sharing. As a result, there are many XML RPC requests. A hierachical cache would reduce the number of XML RPC requests. Every MySQL server could run an Information Schema plugin which performs the XML RPC when a client issues a corresponding SQL query and cache the results. Futhermore, clients would not have us learn a new protocol. They would run simple SQL statements. BTW, PoC synchronous I_S tables with node lists: http://blog.ulf-wendel.de/2013/mini-poc-using-a-group-communication-system-for-mysql-ha/

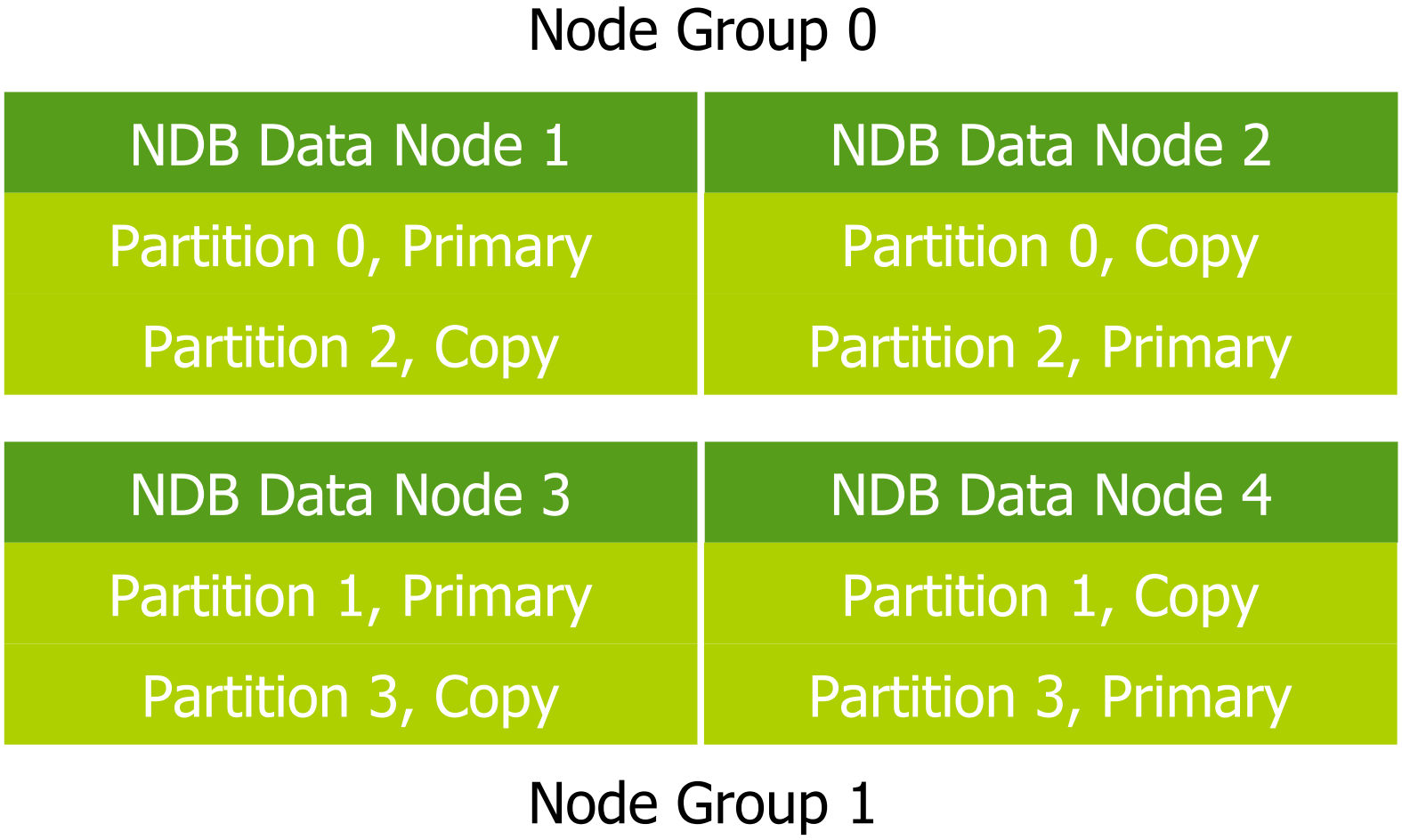
Compared with: Spider



The speaker says…

Spider is a 3rd party storage engine for MySQL. Recall, that any database table in MySQL can be created using a variety of storage engines optimized for different purposes. Different storagen engines store tables on different media. InnoDB uses files, Memory uses RAM, Federated links to a remote database. Spider stores tables on remote MySQL servers and allows partitioning (sharding) of over multiple remote MySQL servers. Distribution transparency is better with Spider than with Fabric: clients use a standard SQL table, MySQL takes care of the distribution over multiple servers. Queries may spawn multiple shards, so may transactions (XA). Good parallelization of work too. However, no HA and no cluster management tool.

Compared with: MySQL Cluster

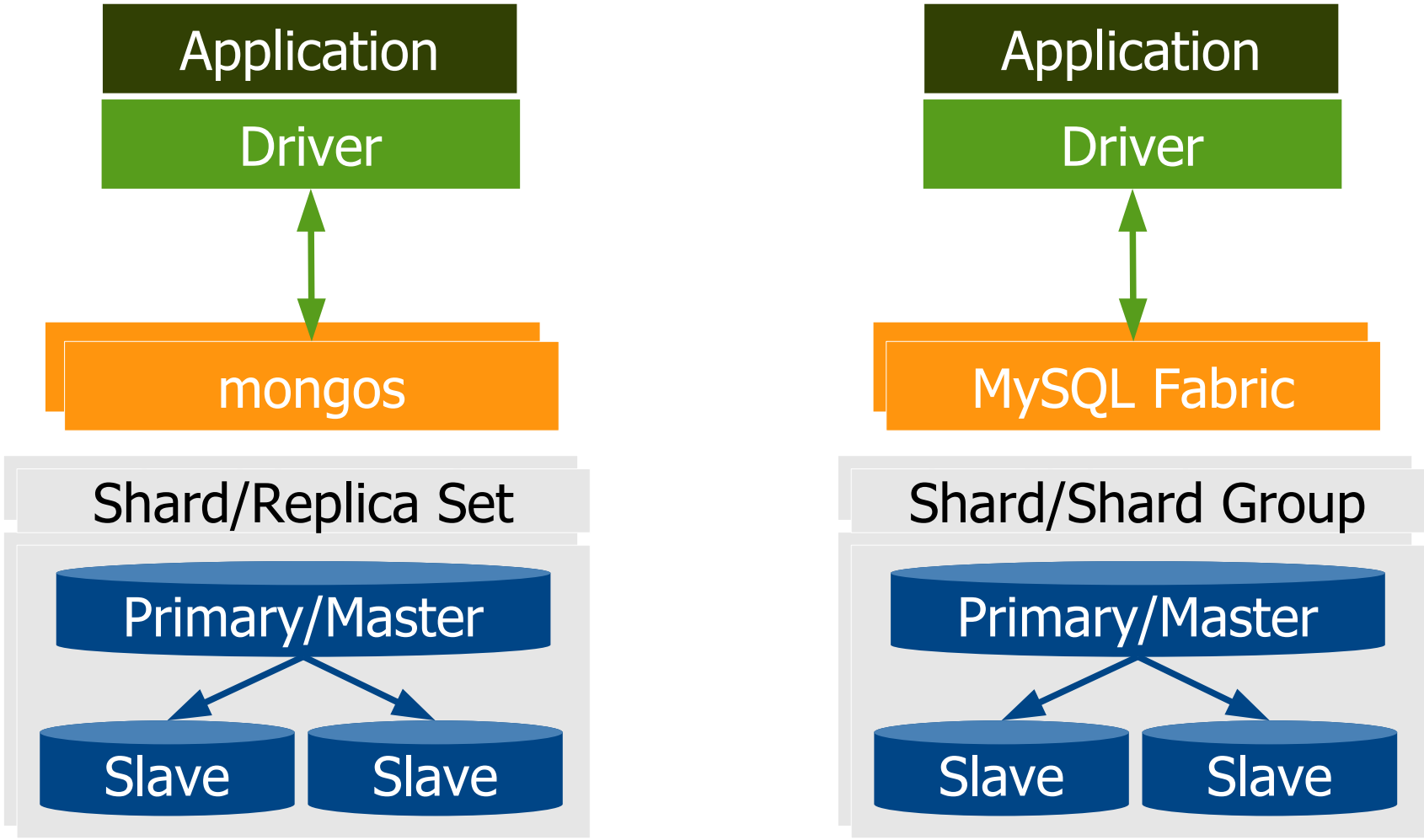


The speaker says…

MySQL Cluster uses synchronous replication (eager update anywhere) and partitioning to. Distribution transparency is high, so is availability and scalability.

The partitioning/sharding is completely hidden from the SQL user. Queries run in parallel on multiple nodes. Transactions can and will spawn over multiple nodes. However, eager systems are not well suited for WAN connections – MySQL Cluster can be replicated among multiple data centers with semi-automatic conflict resultion.
But, NDB != InnoDB. Details are worth a dedicated presentation. Fabric and Cluster co-exsist peacefully.

Compared with: MongoDB



The speaker says…

Need I say more? Yes, I do. There is way more to say. Meet me at the PHP Summit in December!

The data models are different. MySQL is using a strictly typed relational data model. Sharding is possible in such a model, but if not done transparently, it requires developers to plan their queries carefully. Think queries or transactions spawning multiple shards. Whereas plain Key/Value or Key/Document is perfect for partitioning. In an ideal world, documents are self-contained and there is no need for joins or distributed transactions. As usual, this is not free of charge. In computer science you always trade in one advantage for another. My trade at 1:39am: some sleep in favour of better slides. See you in Berlin, PHP Summit!

This presentation describes the very first pre-production lab release of MySQL Fabric published in 09/2013.

Please, keep this in mind with regards to all limits discussed. Competition will highlight the weak points. Ask youself whether this or that limit is by design, or you should check the documentation for improvements.

Initial release...

THE END

Contact: ulf.wendel@oracle.com

The speaker says…

Thank you for your attendance!

Upcoming shows:

Talk&Show! - YourPlace, any time

DOAG SIG Development/Tools,
Eschborn, 25.09.2013

PHP Summit
Berlin, 02.-04.12.2013