ORACLE

MySQL

MySQL & NoSQL: The Best of Both Worlds

Mark Swarbrick
Principal System Consultant MySQL
Mark.swarbrick@oracle.com

Safe Harbour 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 decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

History of MySQL I

- 2001 MySQL 3.23 GA – our first GA release ever!
- 2005 Oracle Corporation acquired Innobase OY
- 2008 Sun acquired MySQL AB for $1 billion
- 2010 Oracle acquired Sun on 27 January
- 2010 MySQL 5.5 first Oracle release, great feedback from community!
- 2012 MySQL 5.6 “Best release ever“

History of MySQL II

- World's Most Popular Open Source Database
- Over 12 million product installations
- 65,000 downloads/day
- The "M" of the widely deployed LAMP stack
- MySQL Commercial Editions Available

World wide use

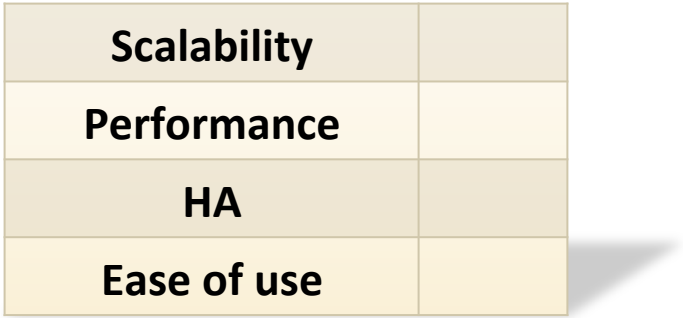
World wide use “at night”

Session Agenda

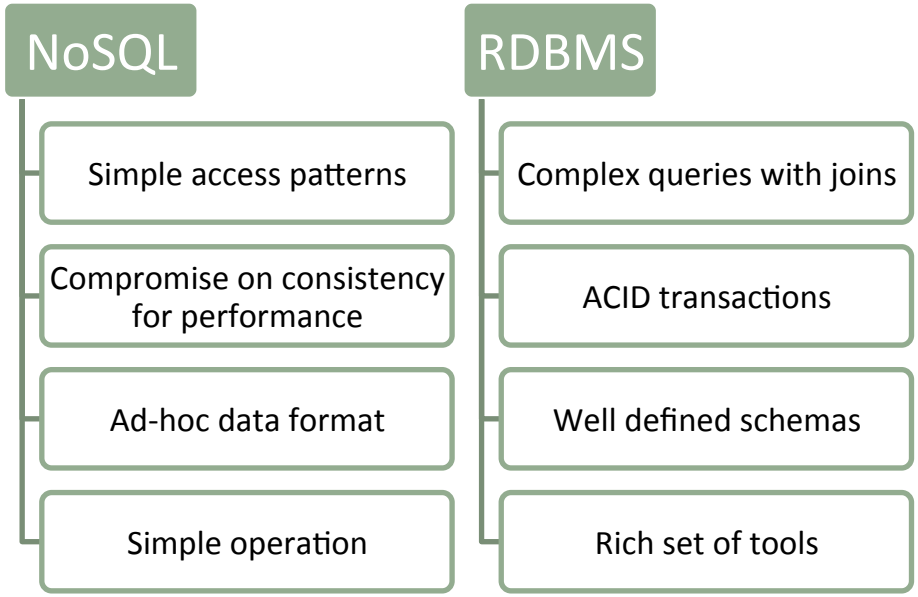
- NoSQL – What are people looking for?
- RDBMS – What advantages do they still have?
- How MySQL Delivers the Best of Both Worlds
  - MySQL Cluster
    - NoSQL attributes: Scale-out, performance, ease-of-use, schema flexibility, on-line operations
    - NoSQL APIs
  - Key-Value store access to InnoDB (Memcached)
- What is coming with future releases

What NoSQL must deliver

- Massive scalability
  - No application-level sharding
- Performance
- High Availability/Fault Tolerance
- Ease of use
  - Simple operations/administration
  - Simple APIs
  - Quickly evolve application & schema



Still a role for the RDBMS?



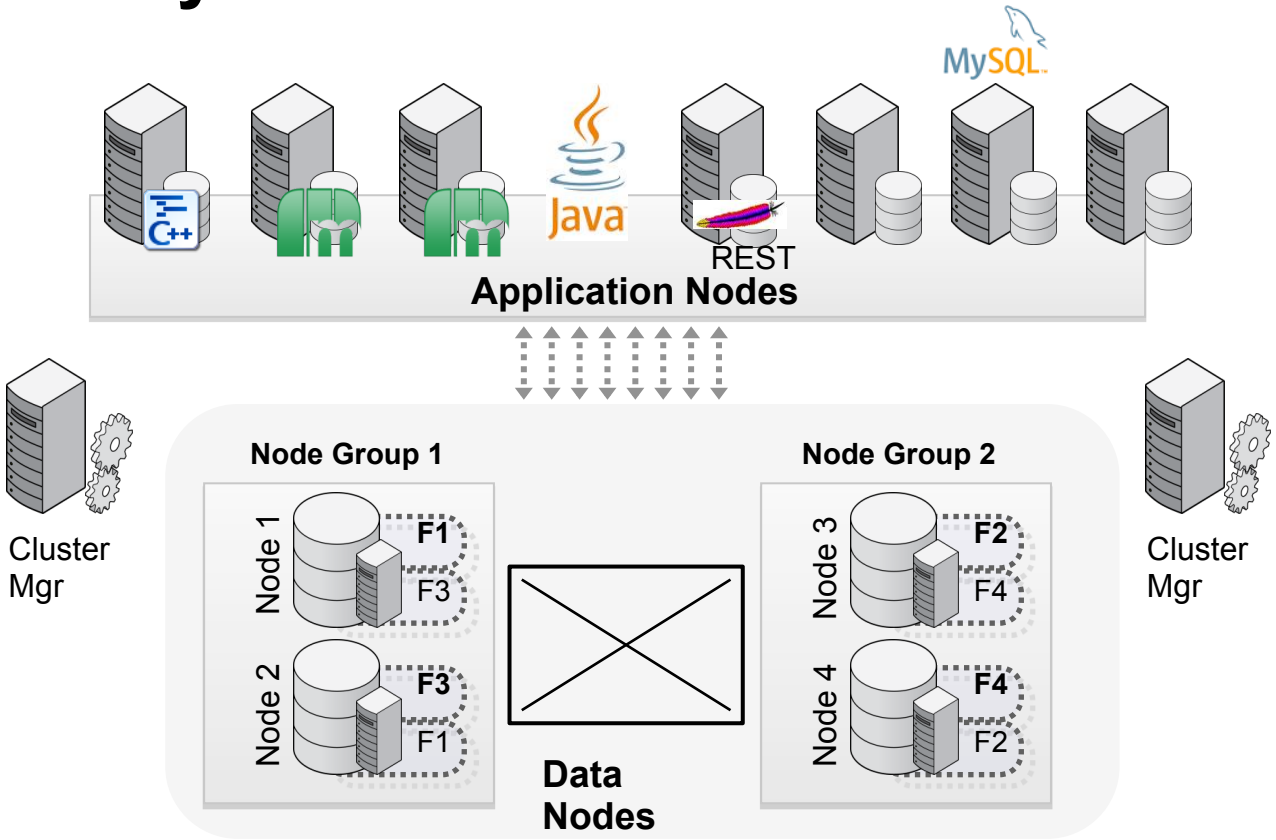
- No best single solution fits all
- Mix and match

Scalability	
Performance	
HA	
Ease of use	
SQL/Joins	
ACID Transactions	

MySQL Cluster introduction

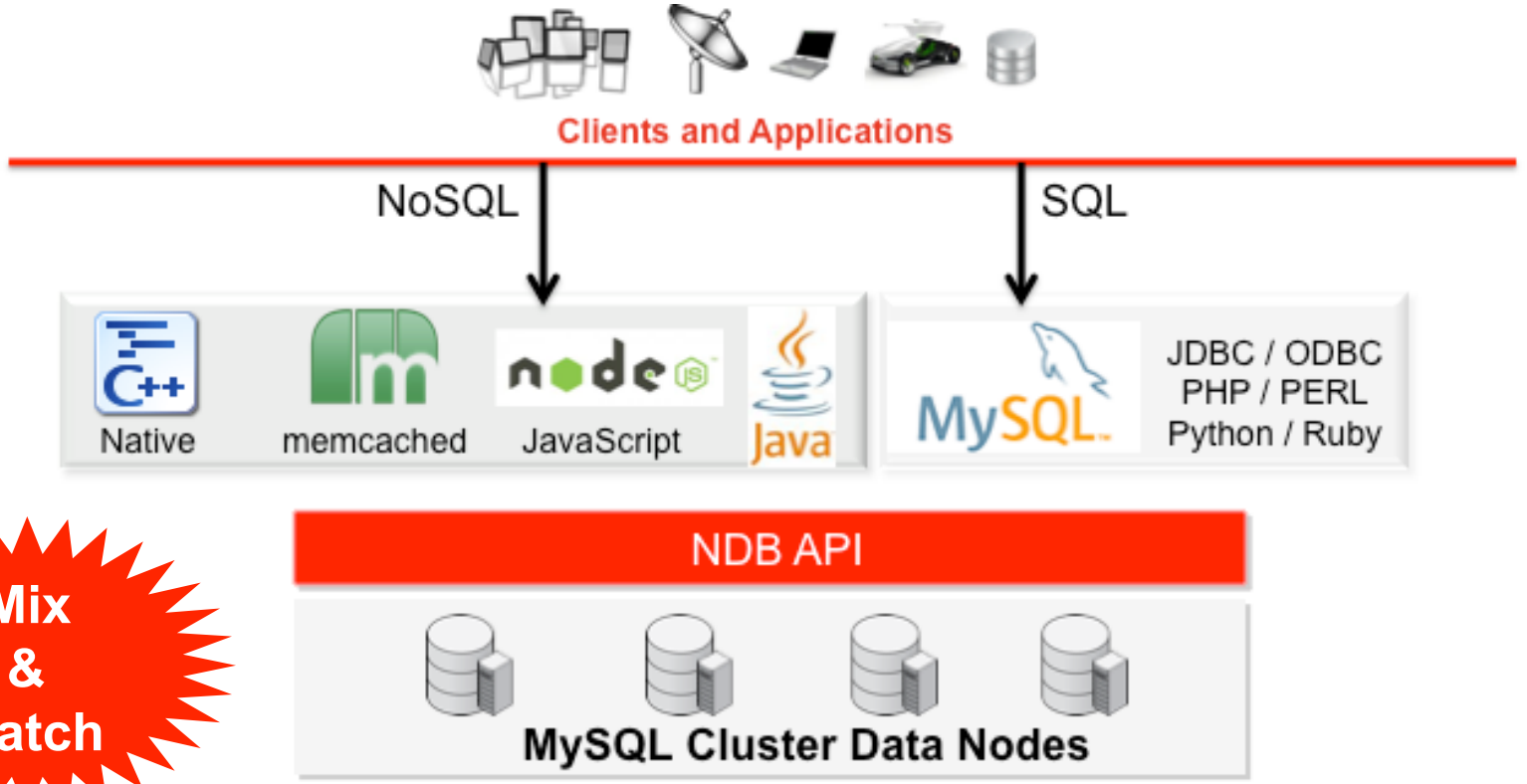
Scaling Reads & Writes	Auto-sharding + Multi-master Transactional, ACID-compliant relational database
99.999% Availability	Shared-nothing design, no Single Point of Failure On-Line operations: Scale, Upgrade Schema, etc.
Real-Time Responsiveness	High-load, real-time performance Predictable low latency, 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 Architecture



Scalability	
Performance	
HA	
Ease of use	
SQL/Joins	✔
ACID Transactions	✔

MySQL Cluster: Extensive Choice of NoSQL APIs



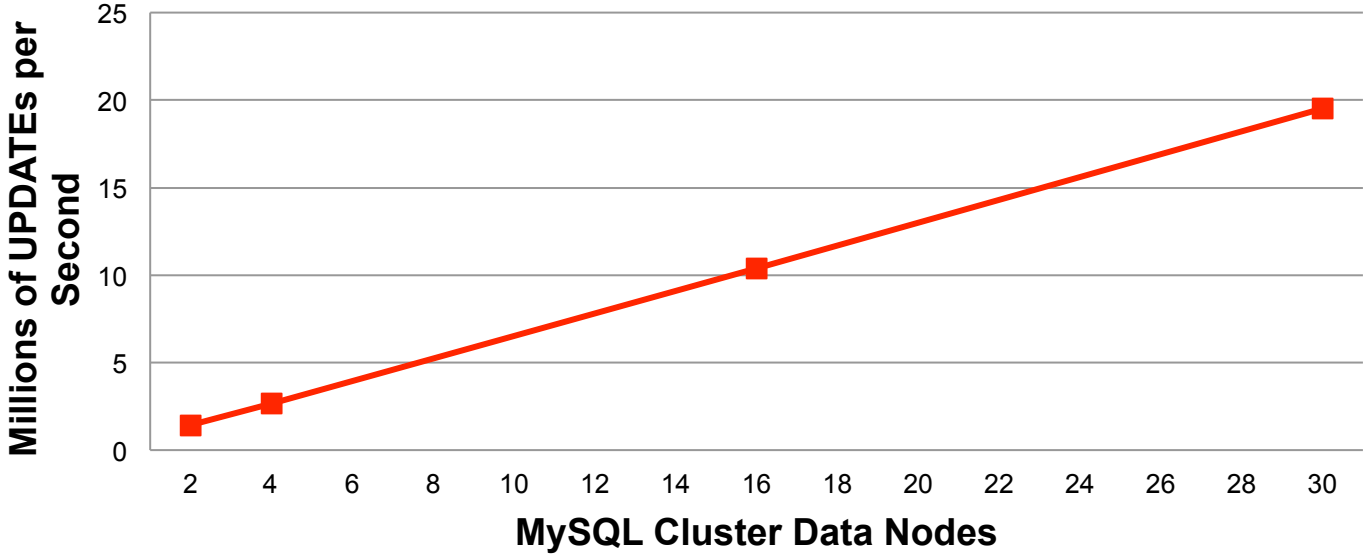
C++ example

```
NdbOperation *op = trx->getNdbOperation(myTable);

op->insertTuple();
op->equal("key", i);
op->setValue("value", &value);

trx->execute( NdbTransaction::Commit );
```

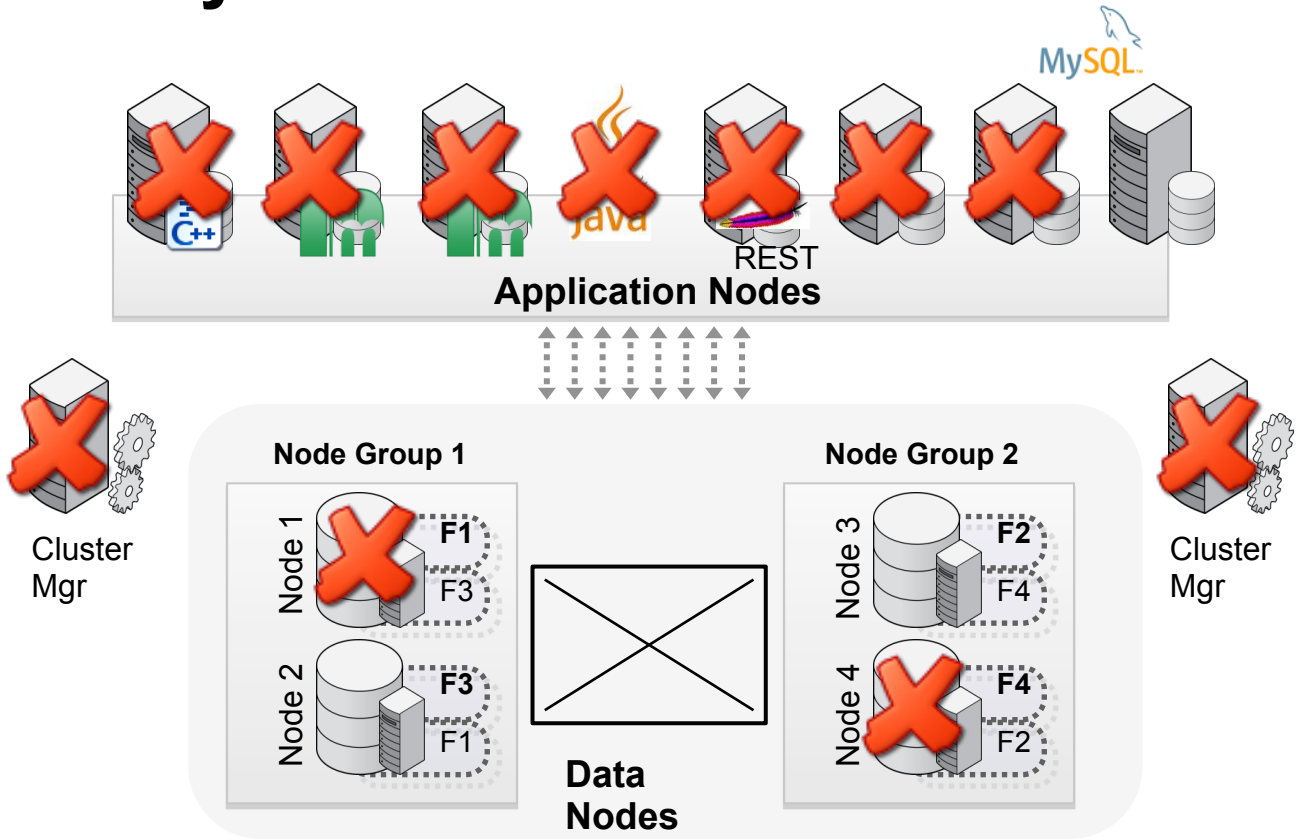
1.2 Billion UPDATEs per Minute



- NoSQL C++ API, flexaSynch benchmark
- 30 x Intel E5-2600 Intel Servers, 2 socket, 64GB
- ACID Transactions, with Synchronous Replication

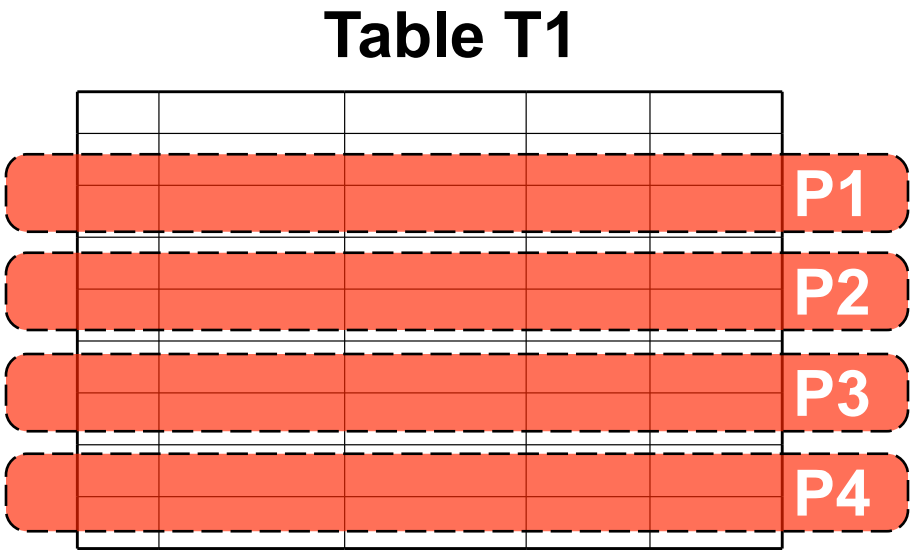
MySQL Cluster Architecture

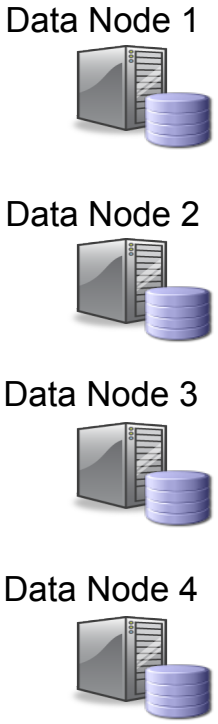
http://clusterdb.com/u/demo



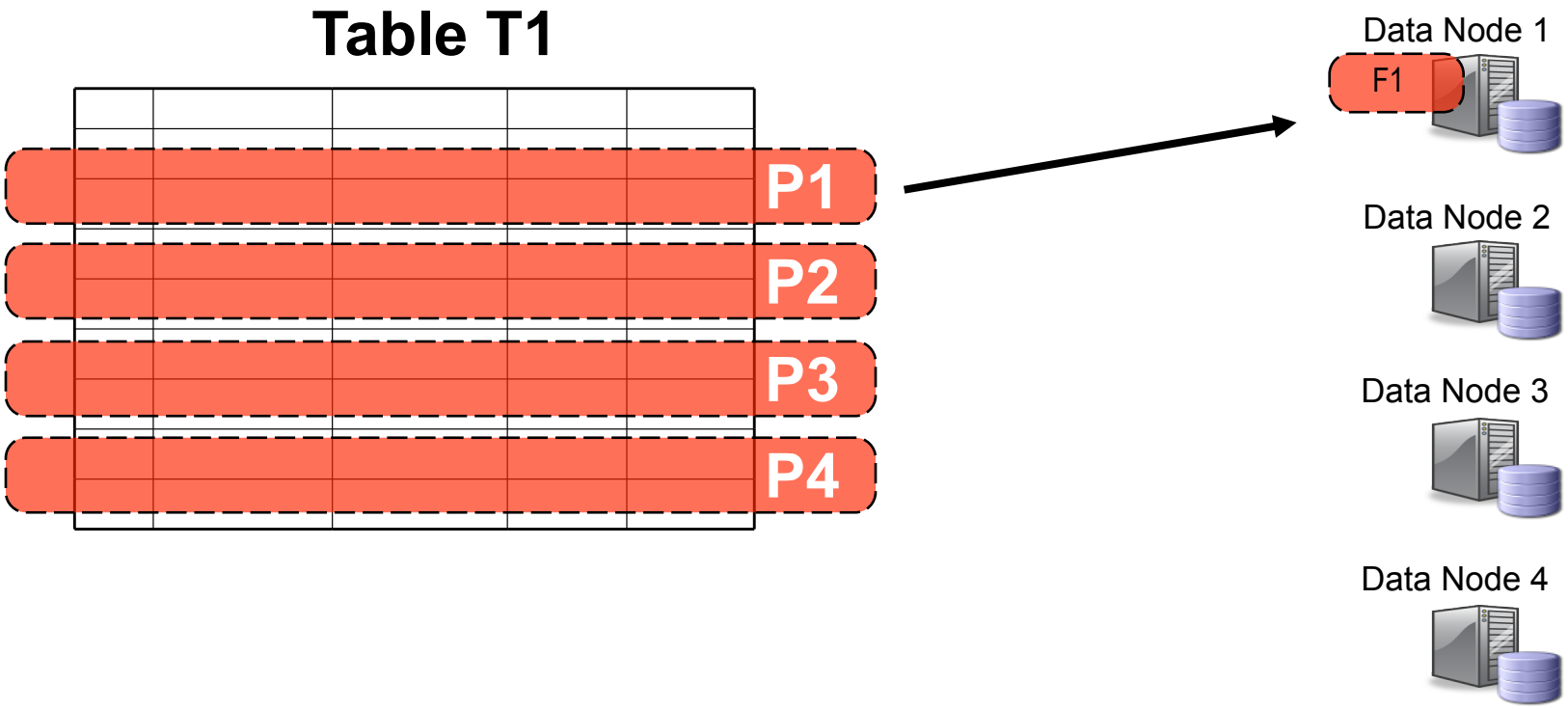
Scalability	
Performance	
HA	✔
Ease of use	
SQL/Joins	✔
ACID Transactions	✔

Scale-Out: Auto-Partitioning

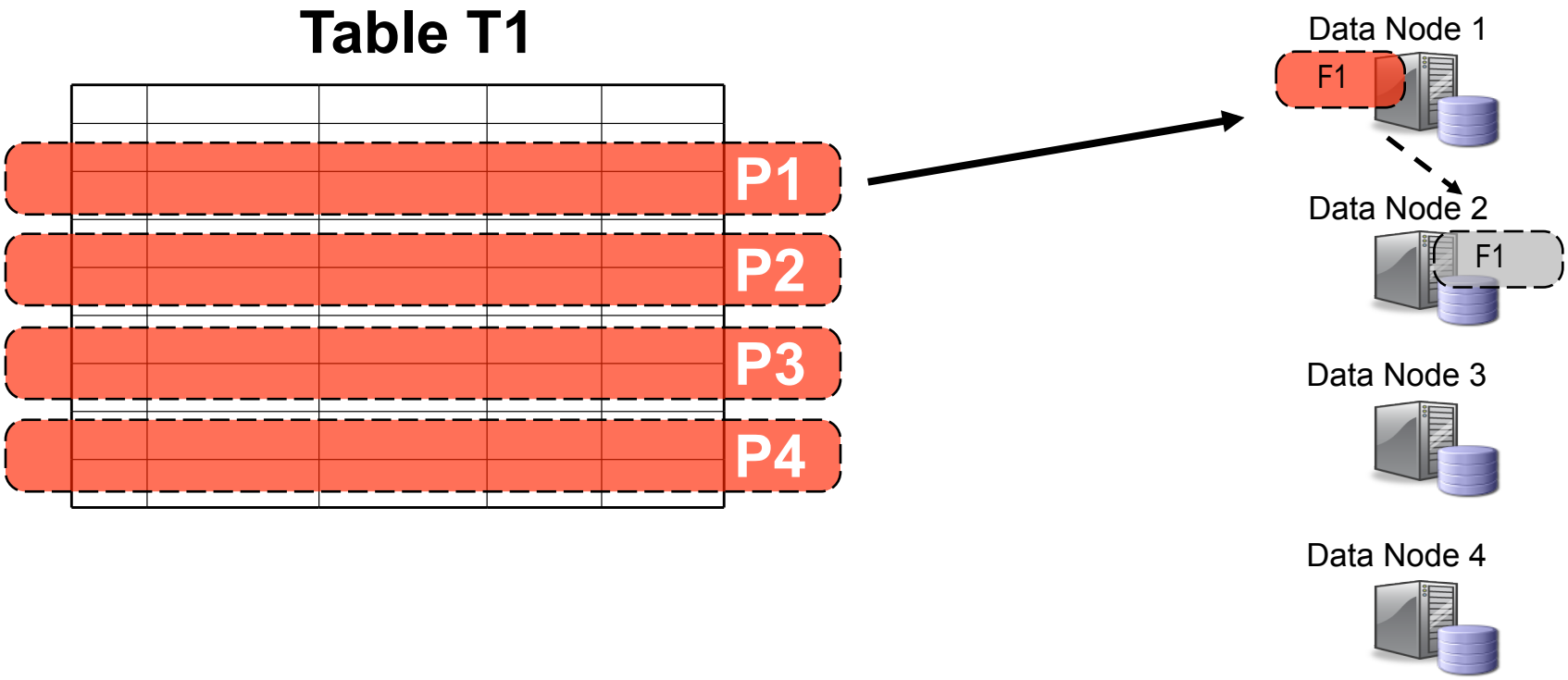




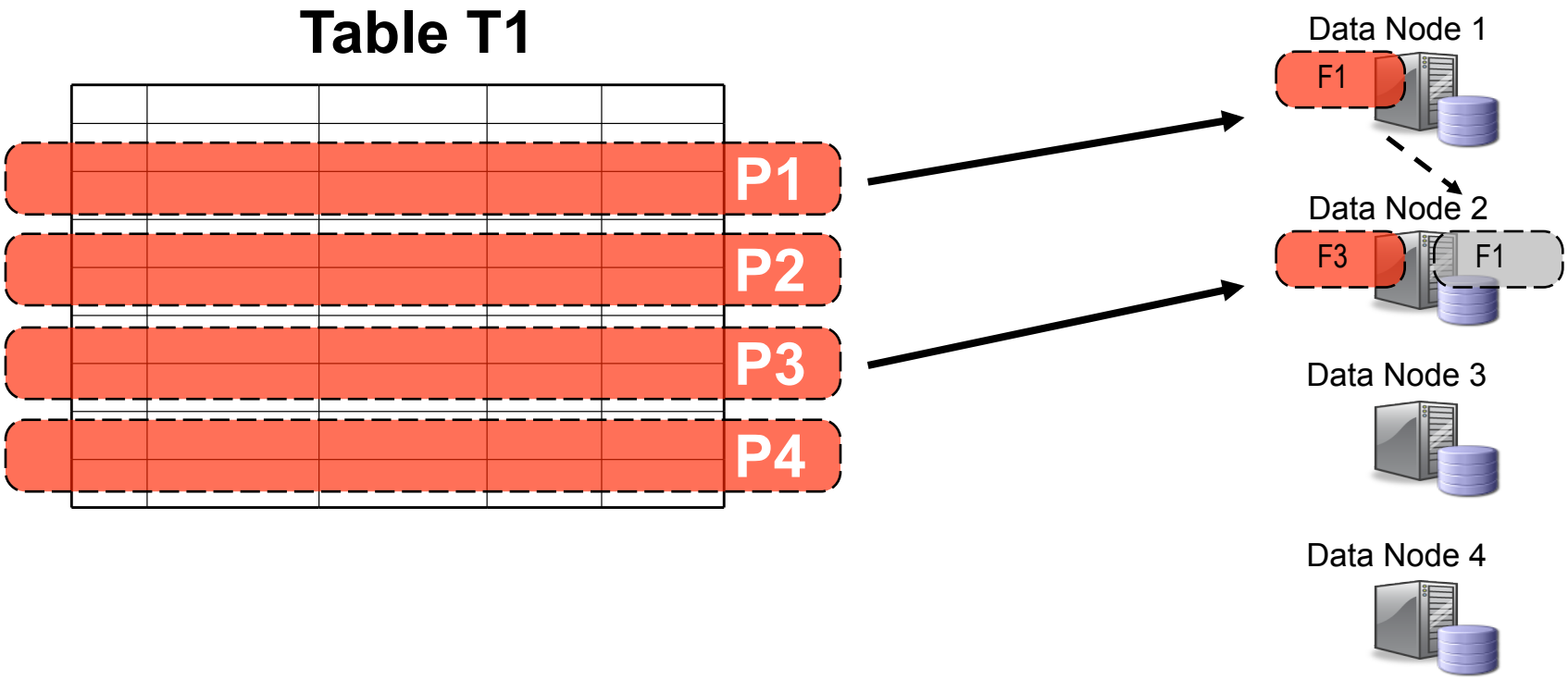
Scale-Out: Auto-Partitioning



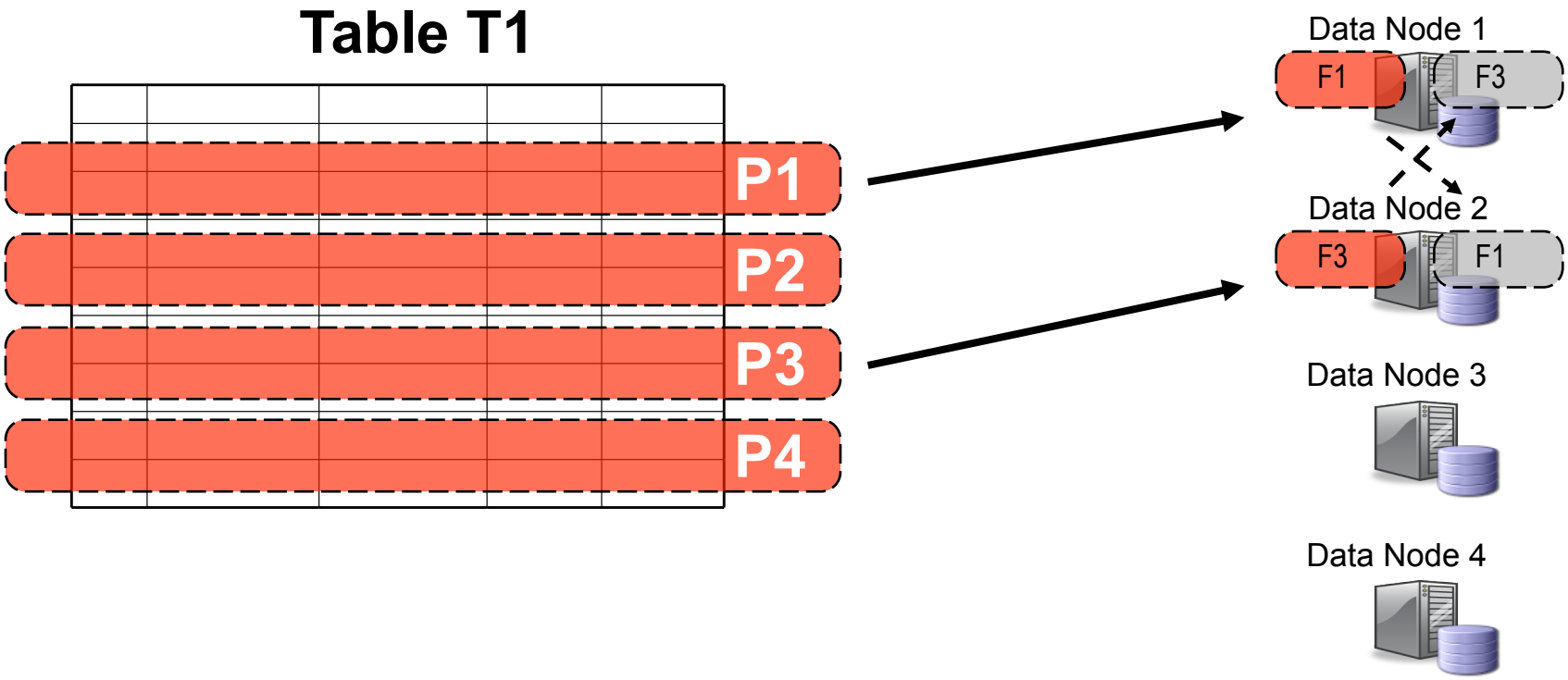
Scale-Out: Auto-Partitioning



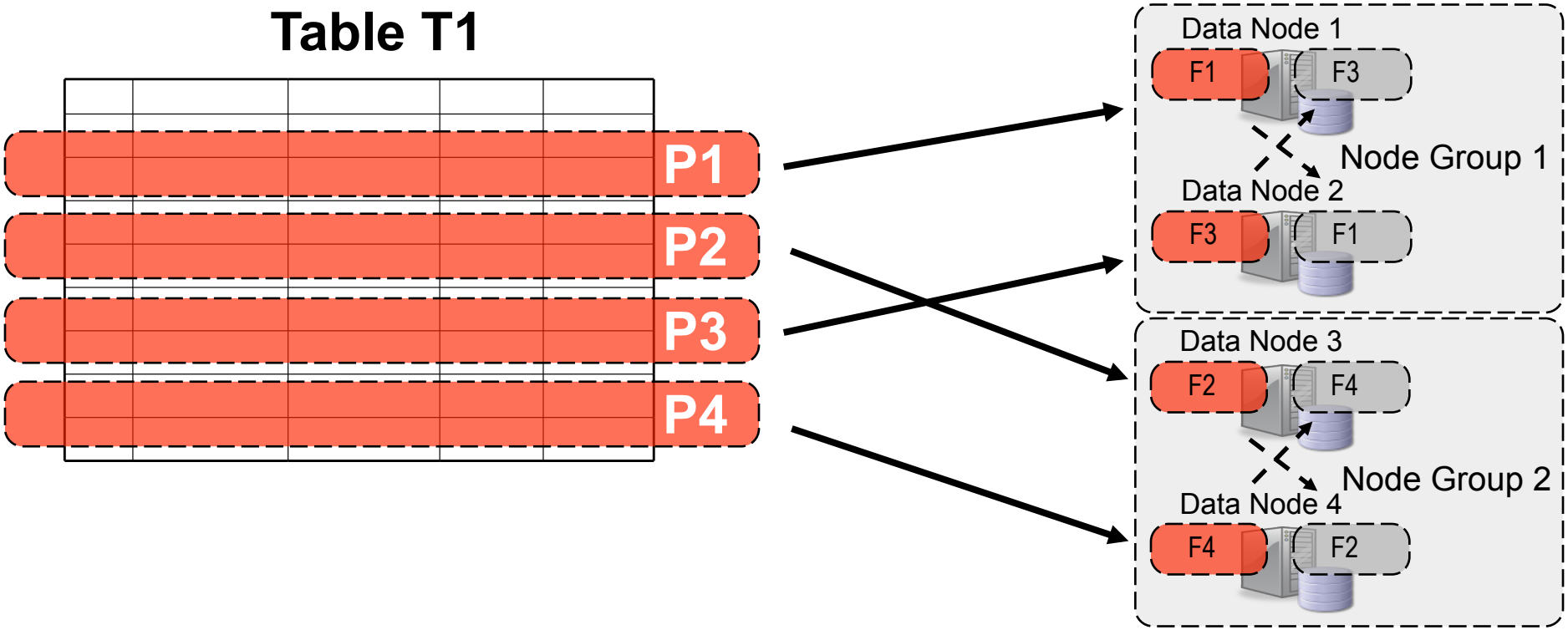
Scale-Out: Auto-Partitioning



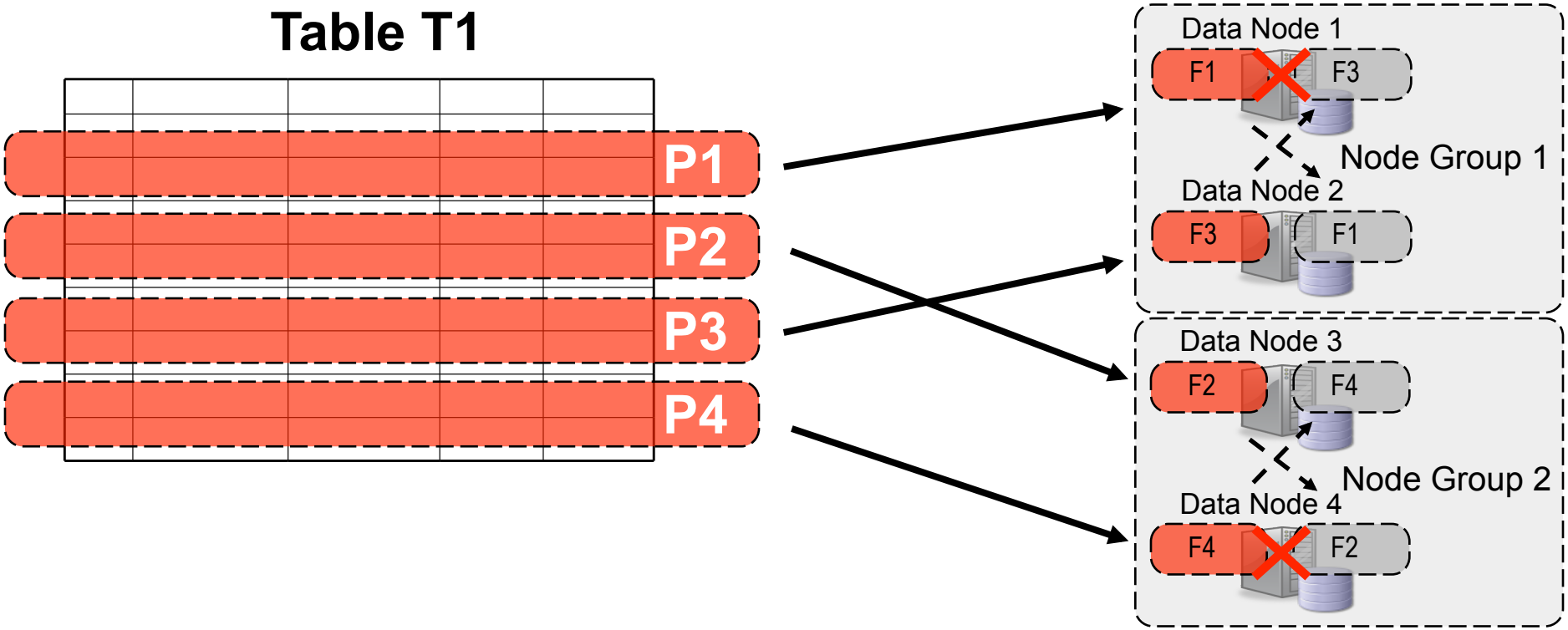
Scale-Out: Auto-Partitioning



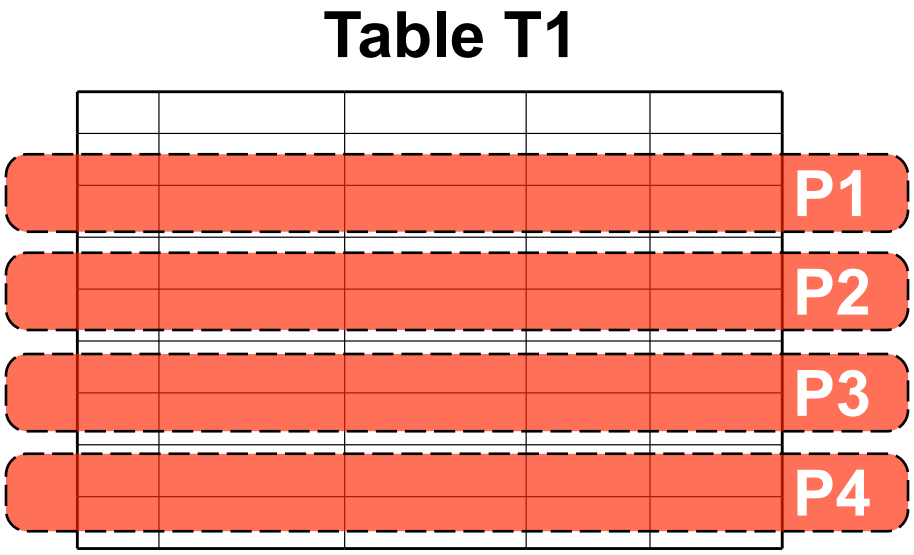
Scale-Out: Auto-Partitioning



Scale-Out: Auto-Partitioning

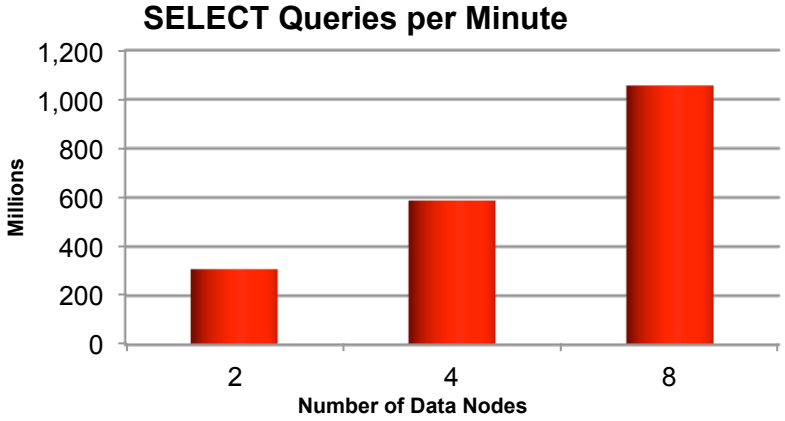


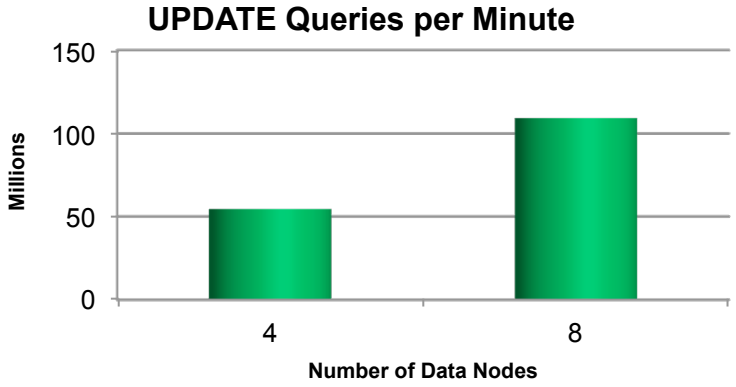
Scale-Out: Auto-Partitioning



Scalability	✔
Performance	
HA	✔
Ease of use	
SQL/Joins	✔
ACID Transactions	✔

Scale-Out Reads & Writes on Commodity Hardware





- 8 x Commodity Intel Servers
  - 2 x 6-core processors 2.93GHz
  - x5670 processors (24 threads)
  - 48GB RAM
- Infiniband networking
- flexAsynch benchmark (NDB API)

Scalability	✔
Performance	✔
HA	✔
Ease of use	
SQL/Joins	✔
ACID Transactions	✔

On-line Schema changes

On-Line Operations

- Scale the cluster (add & remove nodes on-line)
- Repartition tables
- Upgrade / patch servers & OS
- Upgrade / patch MySQL Cluster
- Back-Up
- Evolve the schema on-line, in real-time

MySQL Cluster 7.3

Auto-Sharding, Extreme Performance, Global Replication

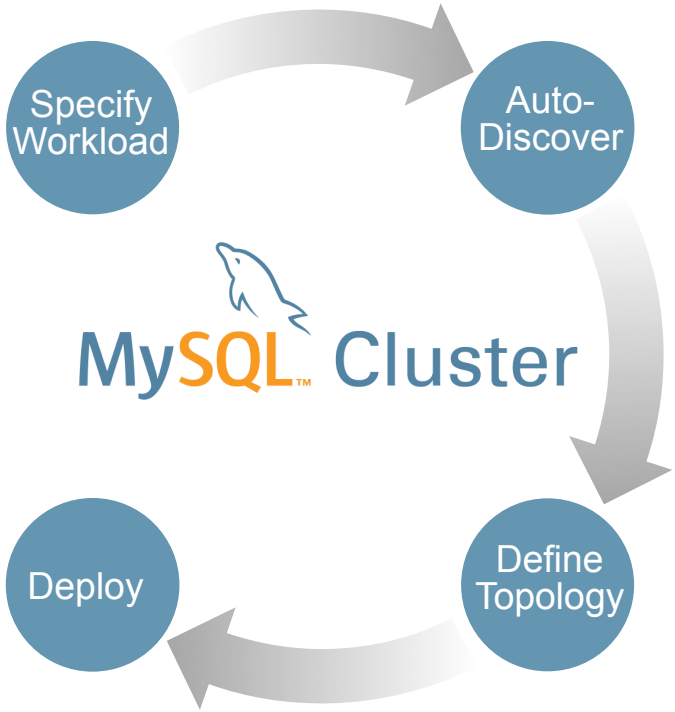
GA Now!

Learn More »

- Foreign Key Support
- Connection Thread Scalability
- MySQL 5.6
- Auto-Installer
- NoSQL JavaScript for node.js

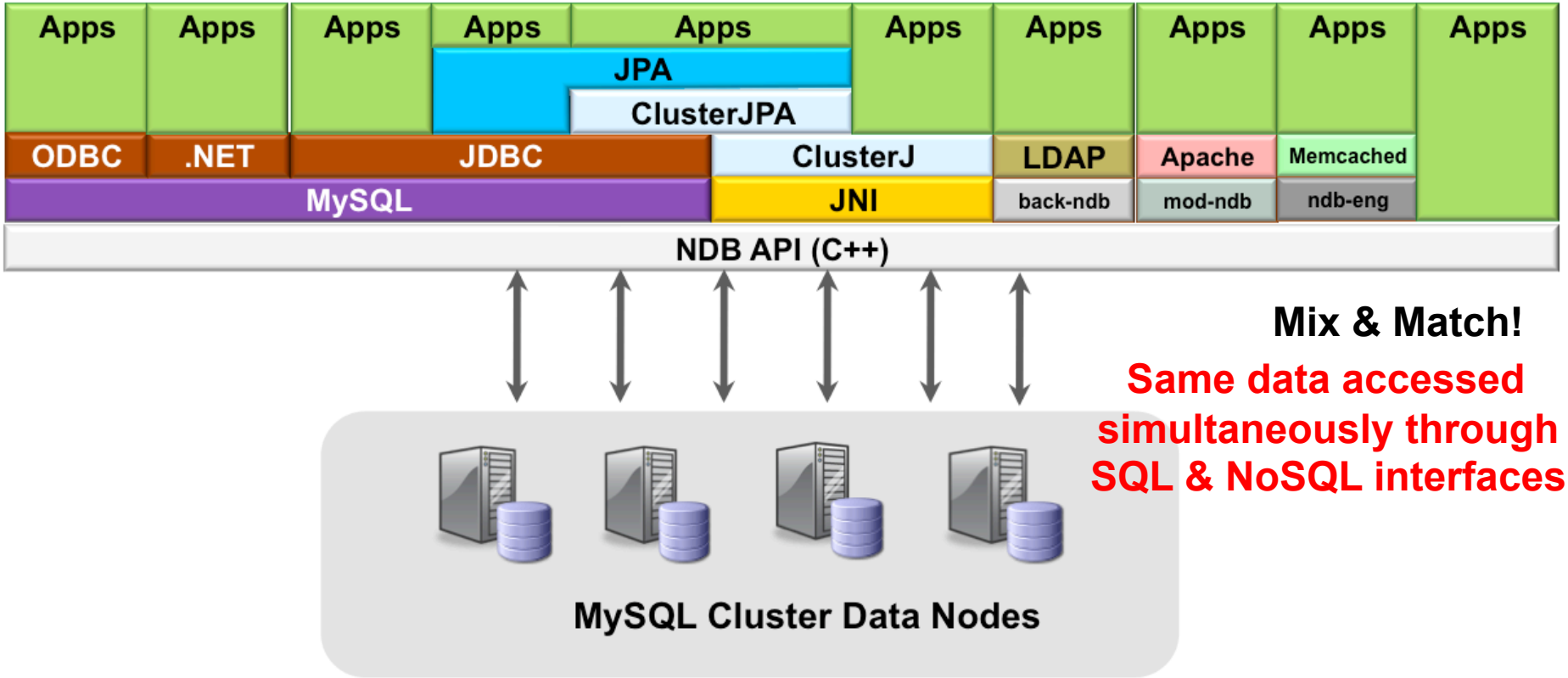
MySQL Cluster 7.3: Auto-Installer

- Fast configuration
- Auto-discovery
- Workload optimized
- Repeatable best practices
- For MySQL Cluster 7.2 + 7.3



Scalability	✔
Performance	✔
HA	✔
Ease of use	✔
SQL/Joins	✔
ACID Transactions	✔

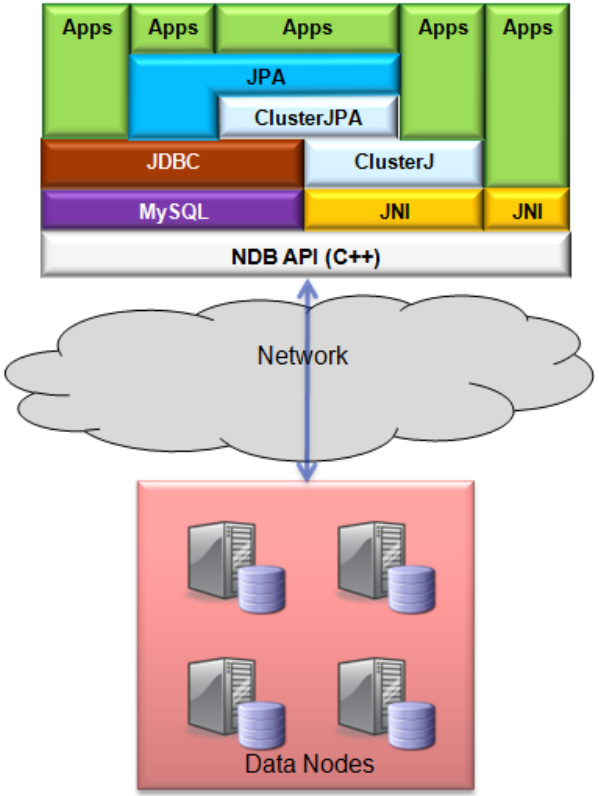
NoSQL Access to MySQL Cluster data



Mix & Match!

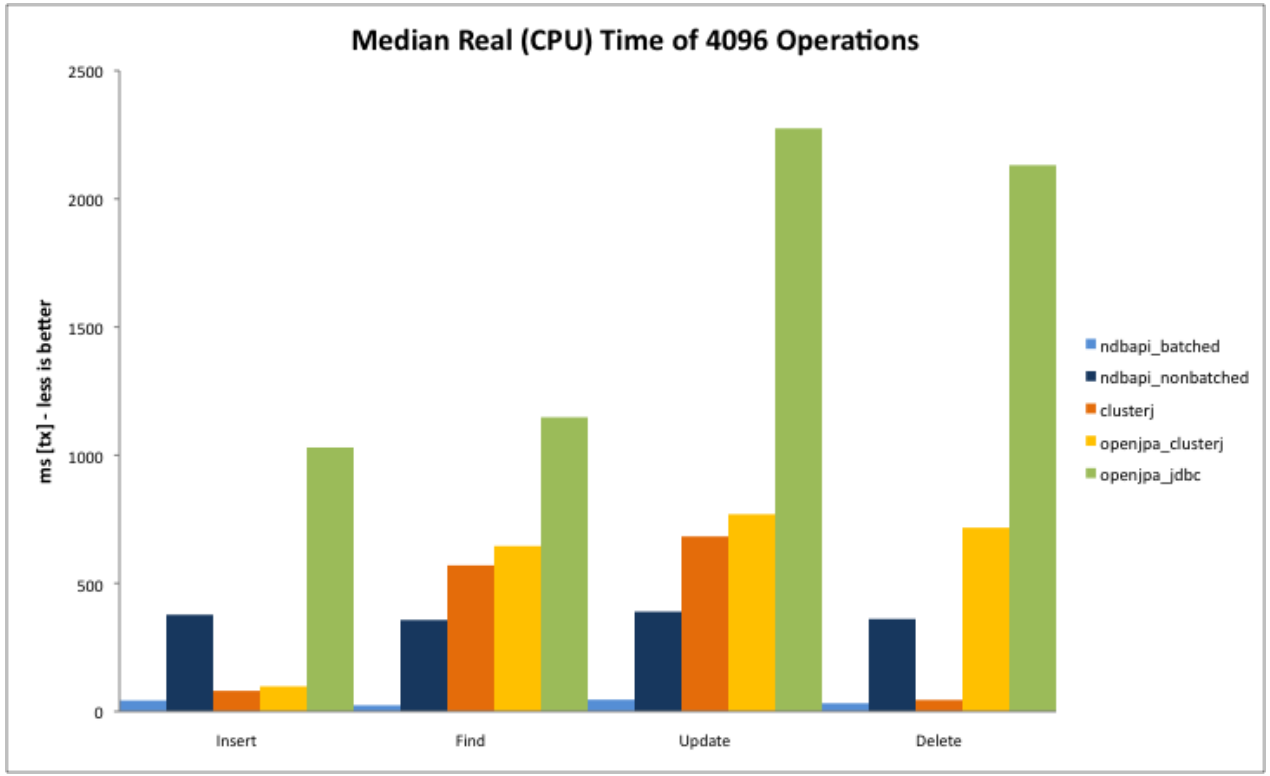
Same data accessed simultaneously through SQL & NoSQL interfaces

MySQL Cluster 7.1: ClusterJ/JPA



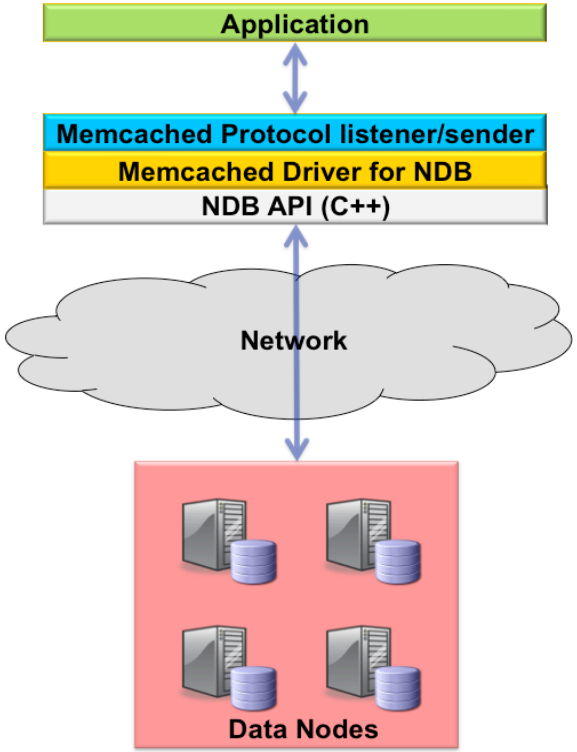
- New Domain Object Model Persistence API (ClusterJ) :
  - Java API
  - High performance, low latency
  - Feature rich
- JPA interface built upon this new Java layer:
  - Java Persistence API compliant
    - Implemented as an OpenJPA plugin
  - Uses ClusterJ where possible, reverts to JDBC for some operations
  - Higher performance than JDBC
  - More natural for most Java designers
  - Easier Cluster adoption for web applications

Java Access Performance



http://www.mysql.com/why-mysql/white-papers/mysql_wp_cluster_connector_for_java.php

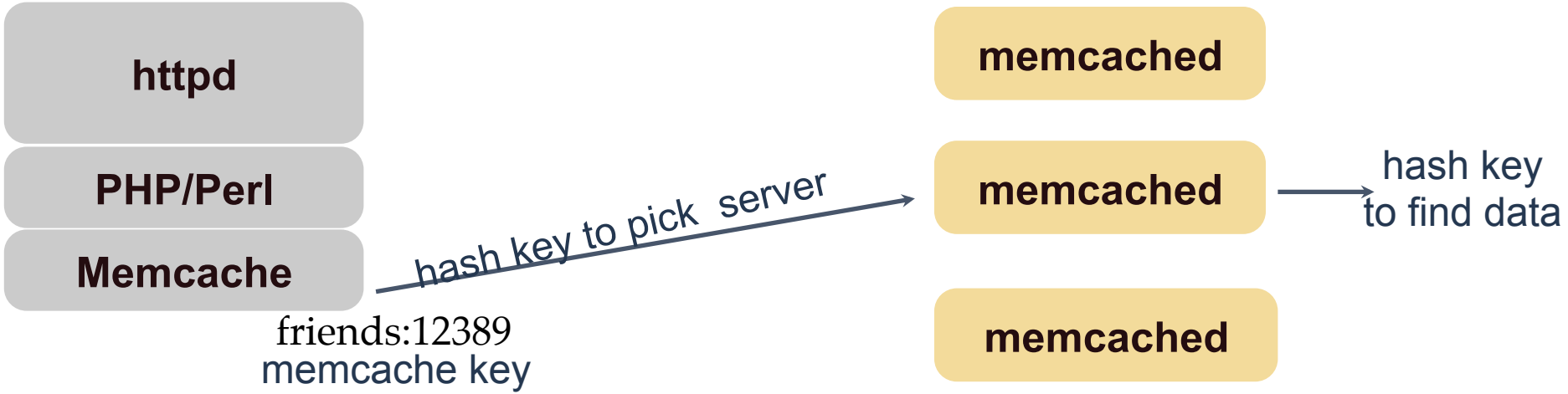
NoSQL with Memcached (MySQL Cluster 7.2)



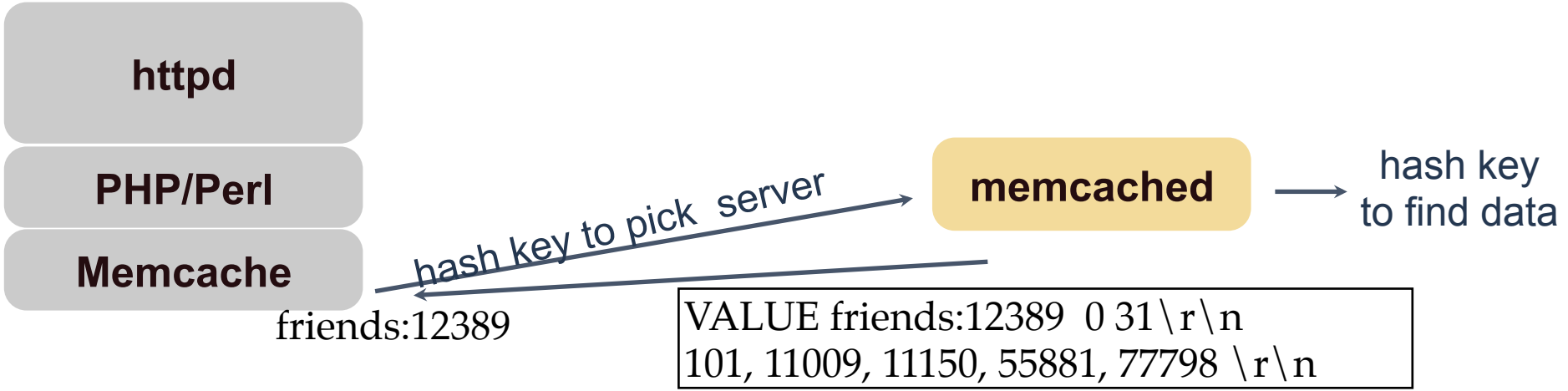
- Memcached is a distributed memory based hash-key/ value store with no persistence to disk
- NoSQL, simple API, popular with developers
- MySQL Cluster already provides scalable, in-memory performance with NoSQL (hashed) access as well as persistence
  - Provide the Memcached API but map to NDB API calls
- Writes-in-place, so no need to invalidate cache
- Simplifies architecture as caching & database integrated into 1 tier
- Access data from existing relational tables

Traditional Memcached Architecture

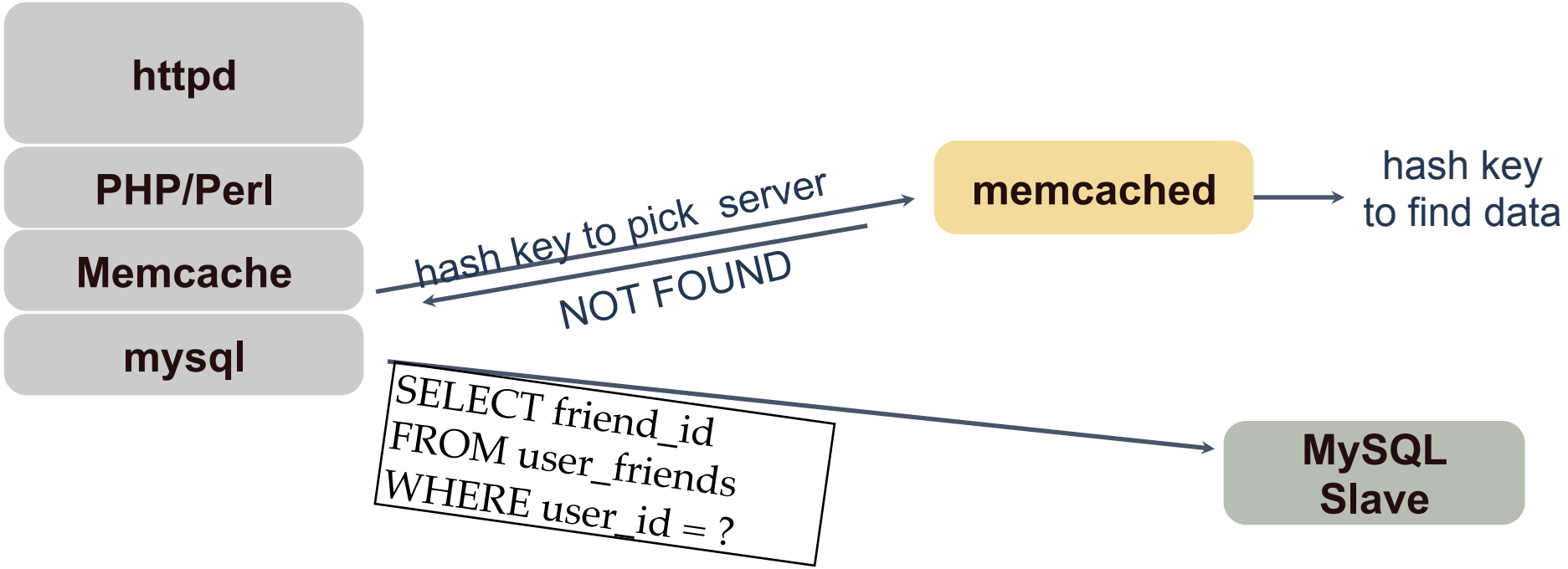
- Two levels of hashing



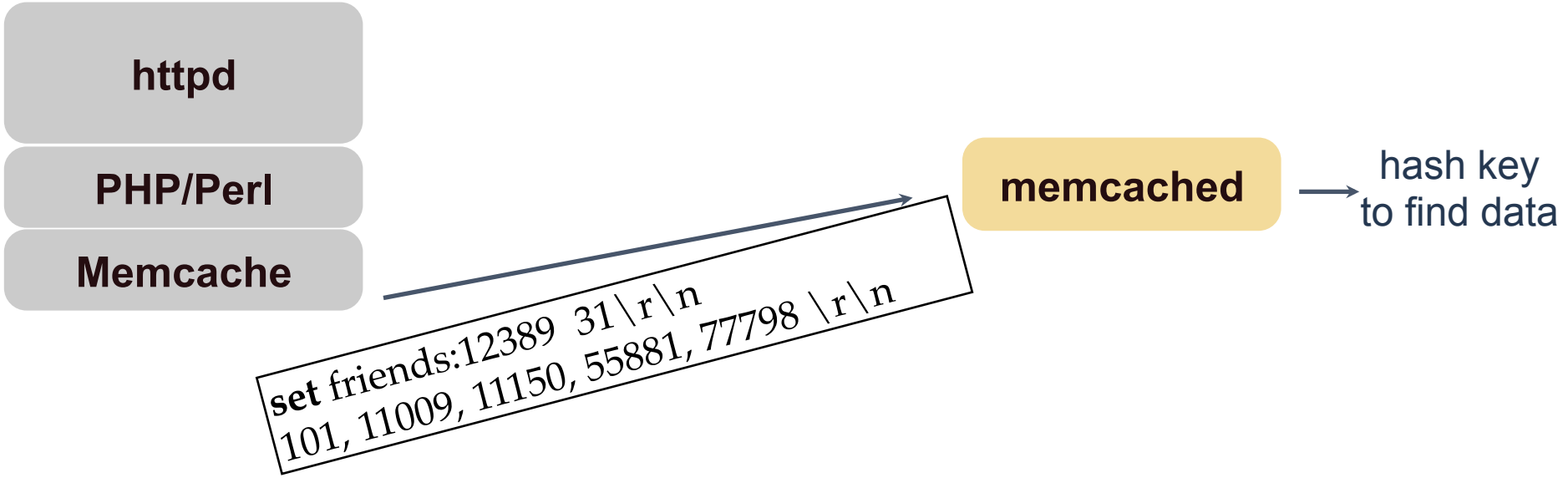
Cache hit



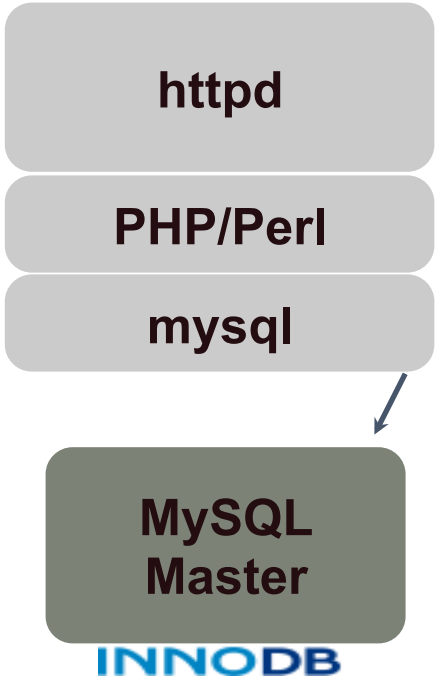
Cache miss (1): fetch from DB



Cache miss (2): manage cache

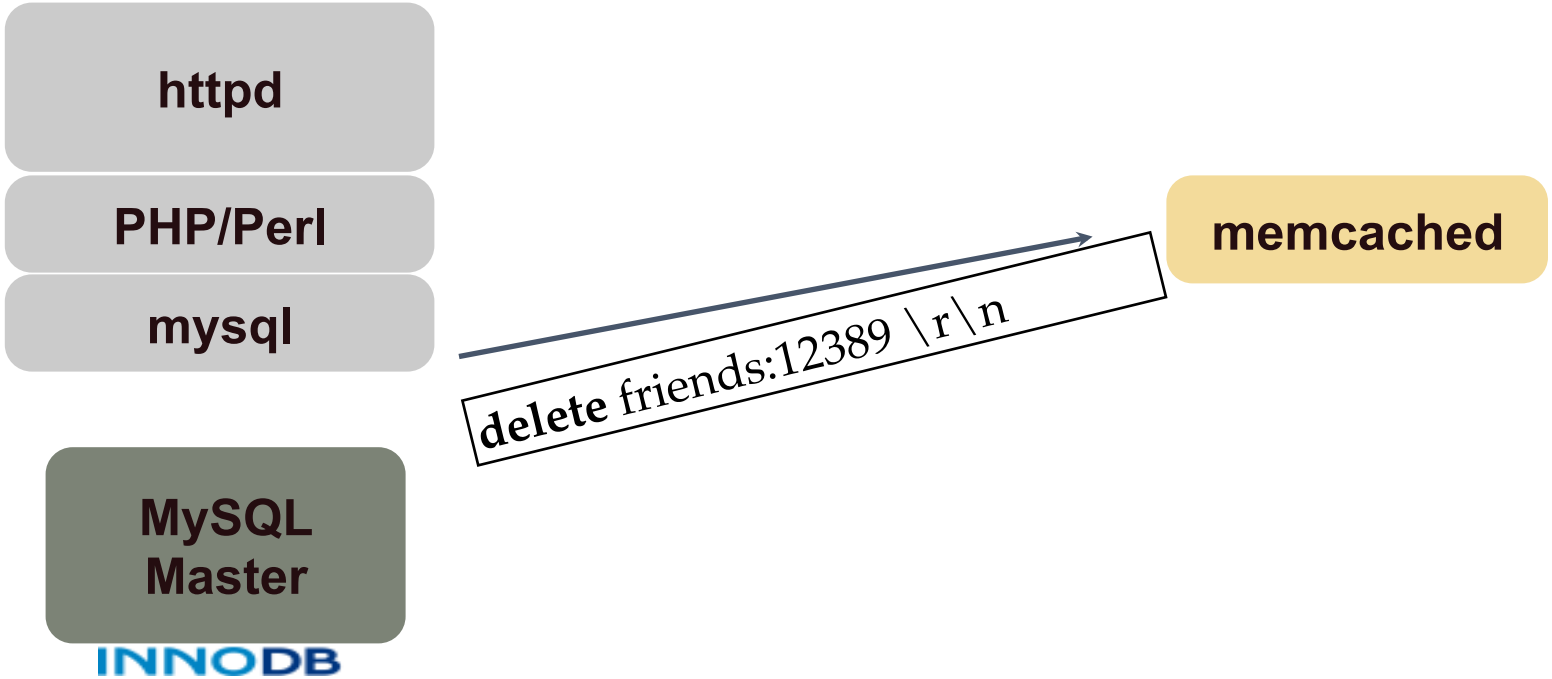


Data change (1): Write to DB

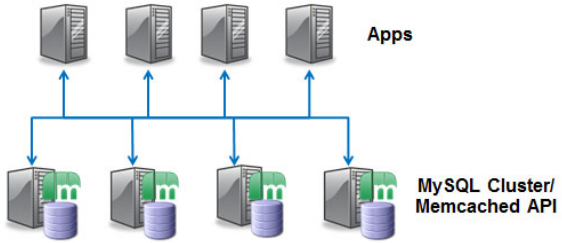


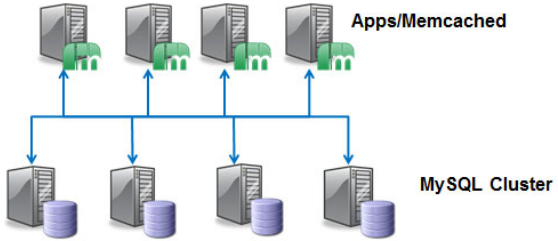

```
DELETE FROM user_friends
VALUES (12389, 999101);
```

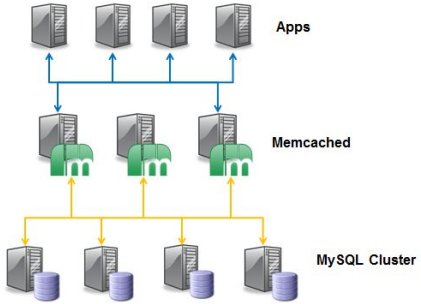
Data change (2): manage cache



NoSQL with Memcached







- Flexible:
  - Deployment options
  - Multiple Clusters
  - Simultaneous SQL Access
  - Can still cache in Memcached server
  - Flat key-value store or map to multiple tables/ columns

```
set maidenhead 0 0 3
SL6
    STORED

get maidenhead
    VALUE maidenhead 0 3
    SL6
    END
```

Cluster & Memcached – Schema-Free

Application view

key value

<town:maidenhead,SL6>

SQL view

key value

<town:maidenhead,SL6>

Key	Value
town:maidenhead	SL6

generic table

Cluster & Memcached - Configured Schema

Application view

key value

<town:maidenhead,SL6>

SQL view

prefix key value

<town:maidenhead,SL6>

Prefix	Table	Key-col	Val-col	policy
town:	map.zip	town	code	cluster
pop:	map.zip	town	popul	cluster

Config tables

town	...	code	popul
maidenhead	...	SL6	...

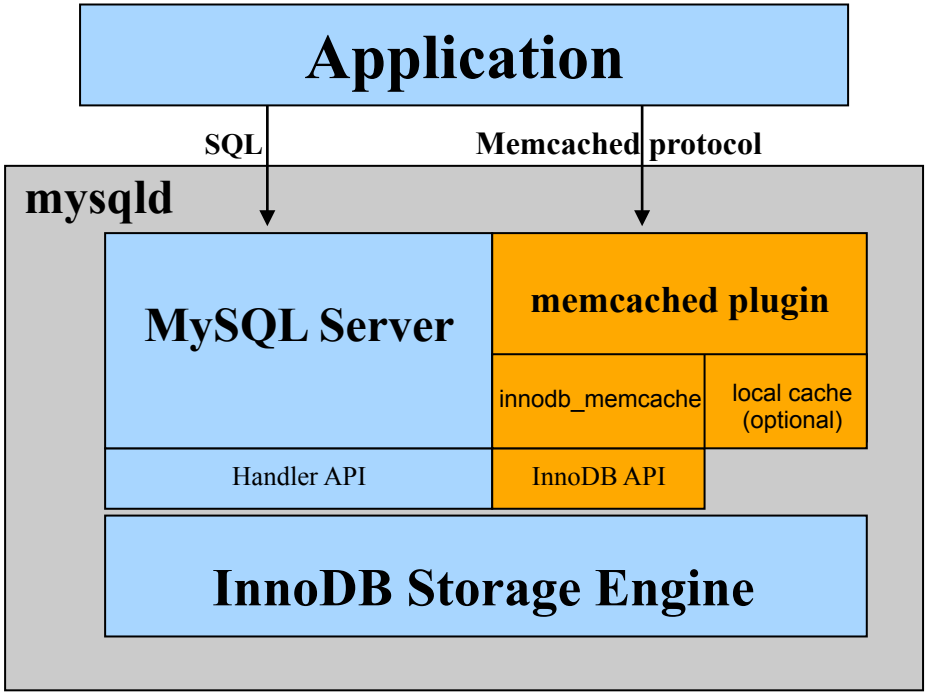
map.zip

Memcached with MySQL Cluster

Try it out

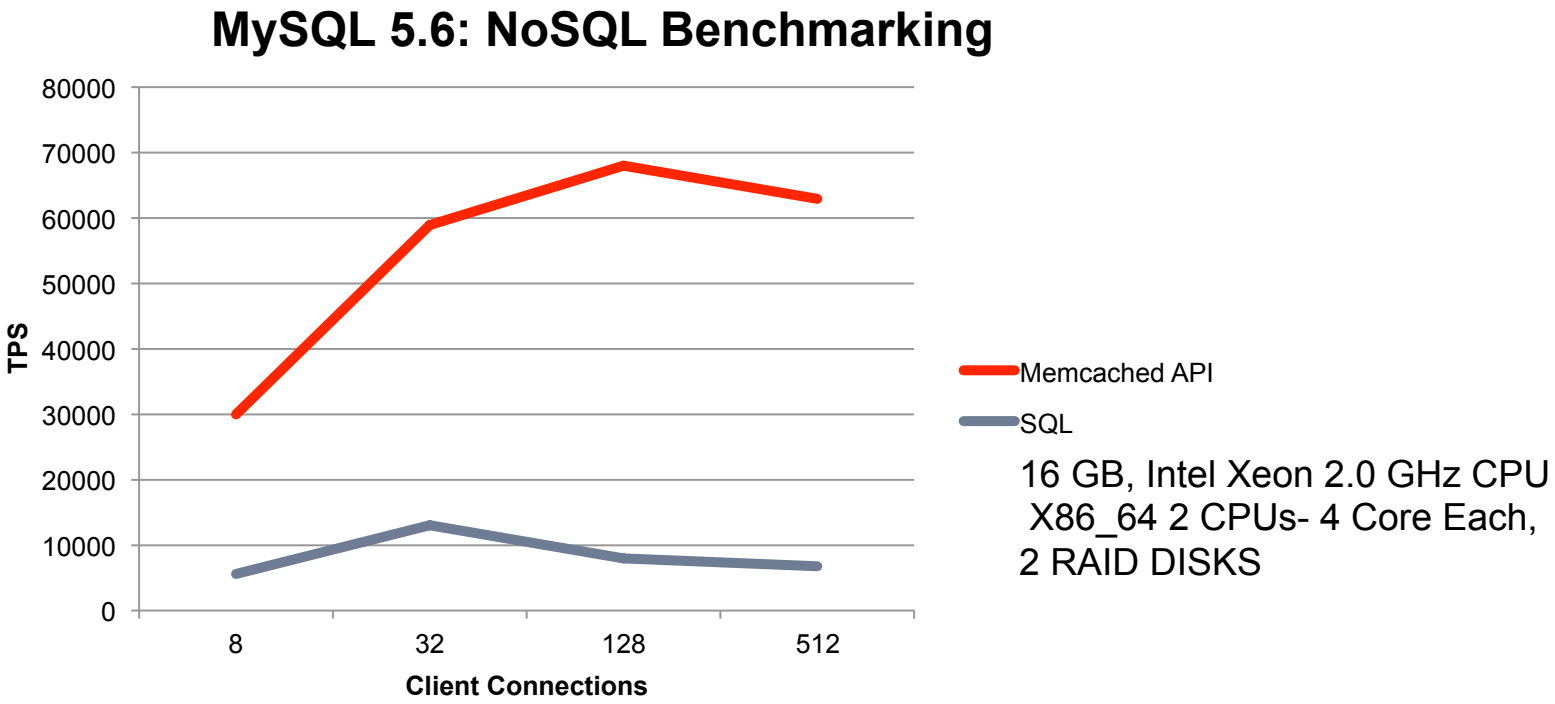
http://clusterdb.com/u/memcached

Memcached NoSQL Access with InnoDB



- Memcached as a plugin of MySQL Server; same process space, with very low latency access to data
- Memcapable: supports both memcached ascii protocol and binary protocol
- Support multiple columns: users can map multiple columns into “value”
- Optional local caching: “innodb-only”, “cache-only”, and “caching”
- Batch operations for performance
- Available from in MySQL 5.6

Performance



Up to 9x Higher "SET / INSERT" Throughput

InnoDB & Memcached - Configured Schema

Application view

key value

<@@town:maidenhead,SL6:12000>

SQL view

prefix key value

<town:maidenhead,SL6:12000>

name	schema	table	keycol	valcols	flags
town	map	city	town	code,popul	
country	world	state	code	name,capital	

Config tables

town	...	code	popul
maidenhead	...	SL6	12000

map.city

Which API to use?

SQL

- Industry standard
- Joins & complex queries
- Relational model

mod_ndb

- REST/JSON
- HTML
- using Apache

Memcached

- simple to use API
- key/value
- driver for many languages
- ideal as e.g. PHP proxy

ClusterJ

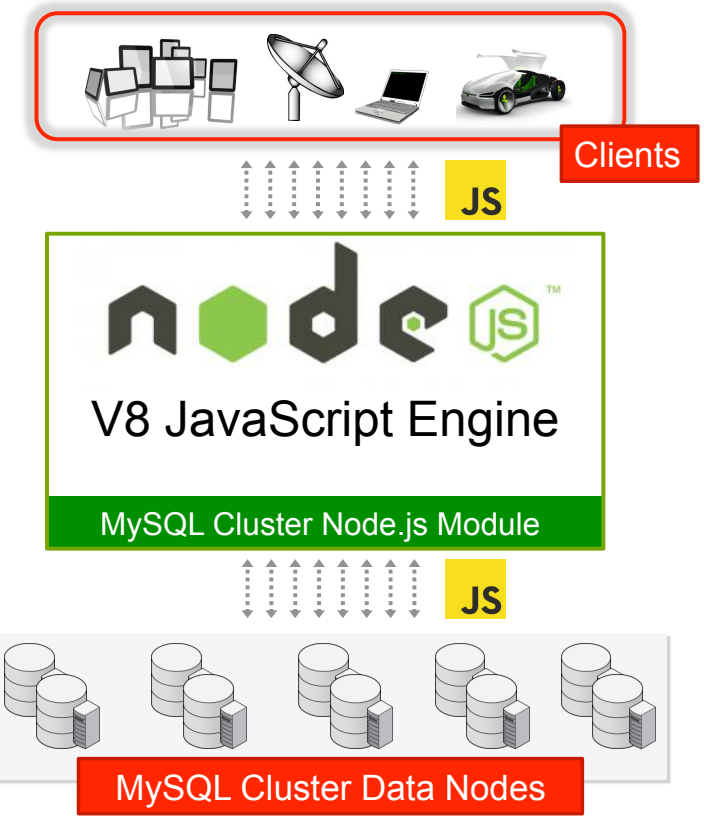
- simple to use Java API
- Web & telco
- Object Relational Mapping
- native & fast access to MySQL Cluster

C++

- experienced developer
- ultra low latency / real-time

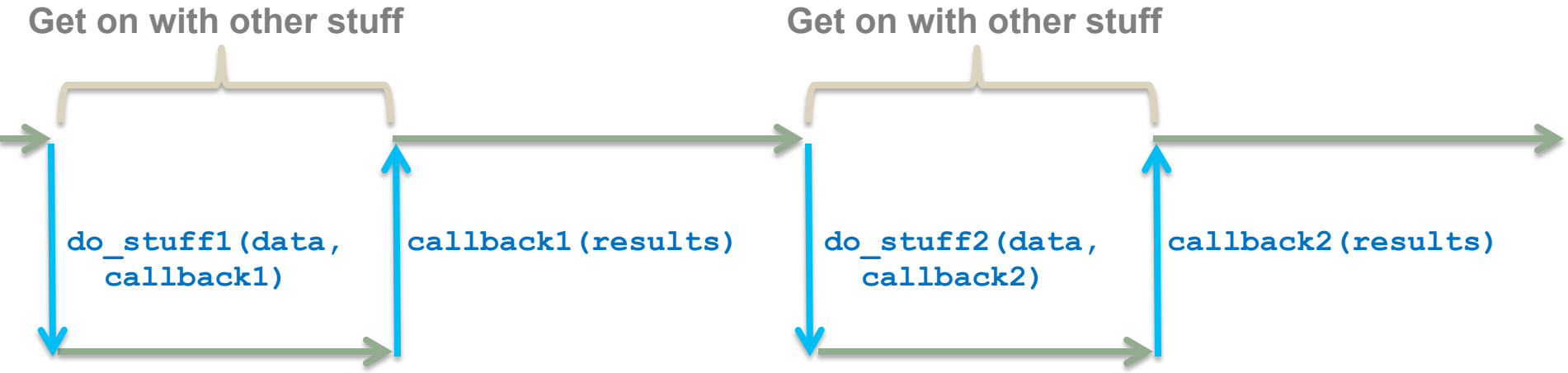
Scalability	✔
Performance	✔
HA	✔
Ease of use	✔
SQL/Joins	✔
ACID Transactions	✔

MySQL Cluster 7.3: Node.js NoSQL API



- Native JavaScript access to MySQL Cluster
  - End-to-End JavaScript: browser to the app and database
  - Storing and retrieving JavaScript objects directly in MySQL Cluster
  - Eliminate SQL transformation
- Implemented as a module for node.js
  - Integrates full Cluster API library within the web app
- Couple high performance, distributed apps, with high performance distributed database

What does an asynchronous API mean?



MySQL Cluster NoSQL API for Node.js

```
var nosql = require('mysql-js');
var annotations = new
 nosql.Annotations();
annotations.mapClass(lib.Tweet,
 {'table' : 'tweet'});
var dbProperties =
 nosql.ConnectionProperties('ndb');
nosql.openSession(dbProperties,
 annotations, onSession);
```

- Modular connector with various back-end adapters:
  - **ndb:** low-level native access to MySQL Cluster
  - **mysql:** access to any MySQL server (translates operations to SQL statements)

MySQL Cluster NoSQL API for Node.js

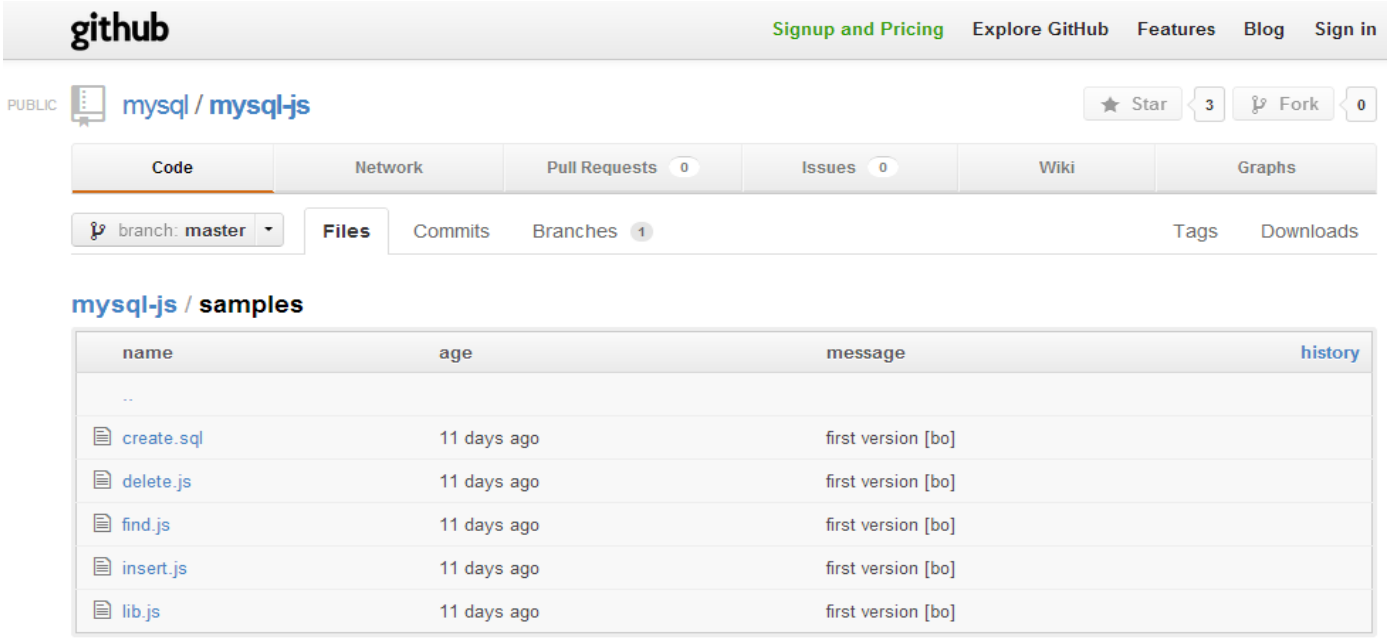
```
var onSession = function(err,
 session) {
  if (err) {…} else {
    var data = new
      lib.Tweet(user_args[0],
      user_args[1]);
    session.persist(data,
      onInsert, data);
  }
};
```

MySQL Cluster NoSQL API for Node.js

```
var onInsert = function(err,
 object) {
  console.log('onInsert.');
  if (err) {…} else {
    console.log('Inserted: ' +
     JSON.stringify(object));
  }
};
```

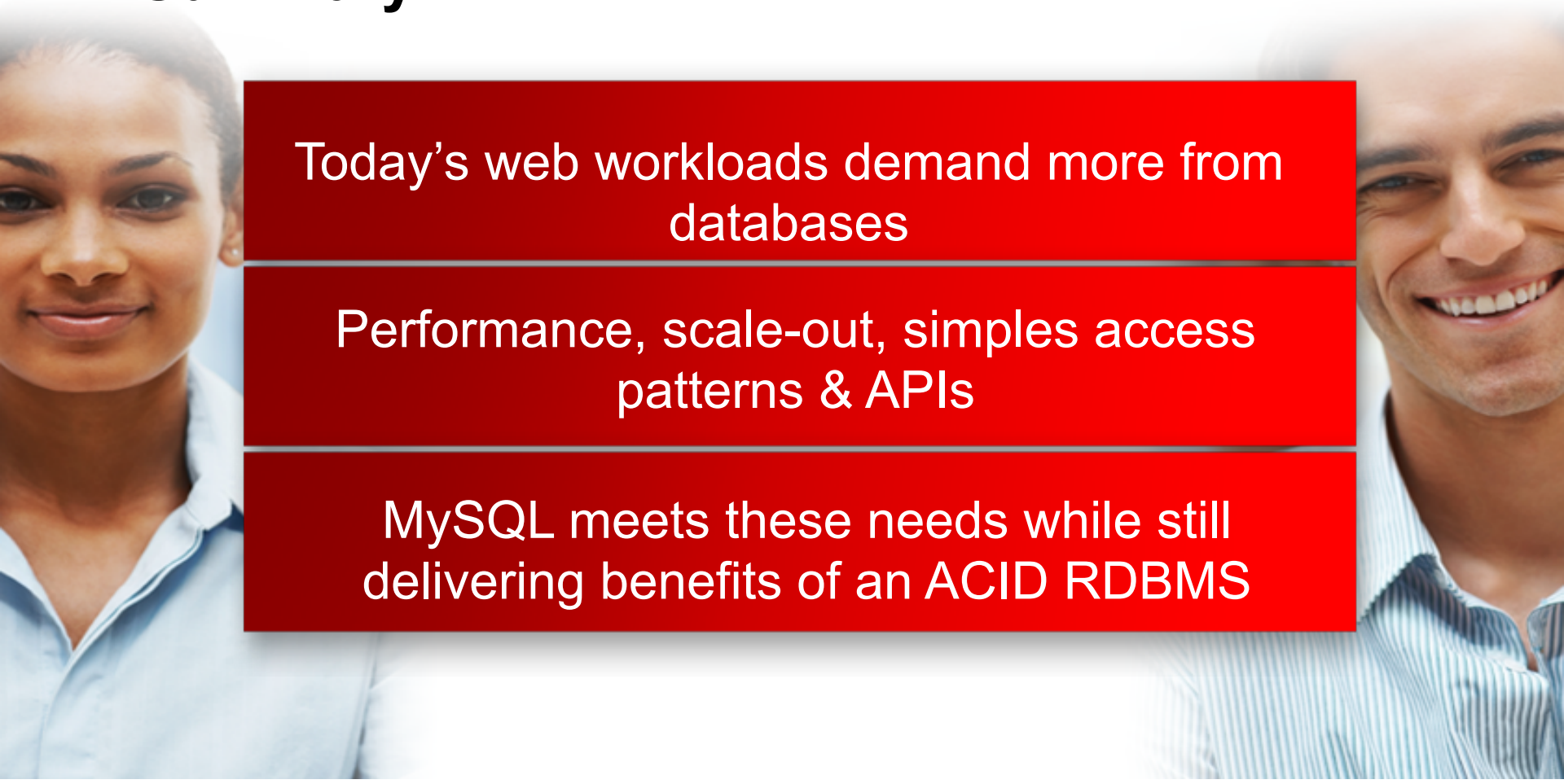
Try Node.js example for yourself

- https://github.com/mysql/mysql-js/tree/master/samples



Who's Using MySQL Cluster?

Summary



Today's web workloads demand more from databases

Performance, scale-out, simples access patterns & APIs

MySQL meets these needs while still delivering benefits of an ACID RDBMS

Next Steps

- Guide to MySQL and NoSQL - Delivering the Best of Both Worlds
  - http://mysql.com/why-mysql/white-papers/mysql-wp-guide-to-nosql.php
- Evaluate MySQL Cluster 7.3
  - http://www.mysql.com/downloads/cluster/
- Bootstrap a Cluster
  - https://edelivery.oracle.com/
- Try Memcached API for InnoDB in 5.6
- http://www.mysql.com/downloads/

Thank you!