

ORACLE®



**ORACLE®**

## **The Native NDB Engine for Memcached**

John David Duncan  
[john.duncan@oracle.com](mailto:john.duncan@oracle.com)

# Program **Agenda**

- MySQL Cluster Today
- A little bit about memcached
- Design Decisions
- Configuration
- Performance
- Links
- Demo



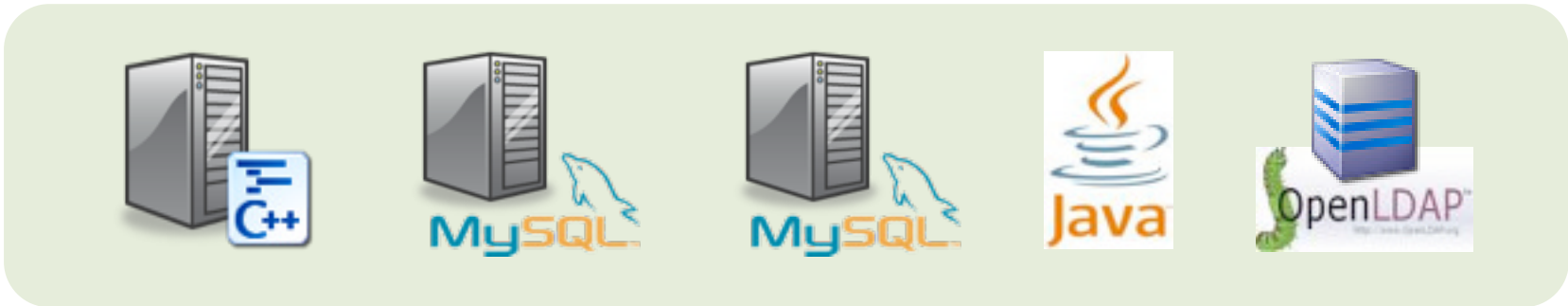
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development,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

# MySQL Cluster Today



# The Basics

## API Nodes



Node Group 1



Node Group 2

## NDB Data Nodes

## The Basics – Data Storage

- High Read and Write Performance
  - Automatic data partitioning, multi-master, parallel execution
- High Availability
- Fast Failover
  - sub-second fault detection and reconfiguration
- Scalable
  - Using commodity hardware



Node Group 1



Node Group 2

**NDB Data Nodes**

# The Basics – Data Access

## API Nodes



- High throughput
  - tens-to-hundreds of thousands of transactions per second
- Low latency
  - sub-millisecond response
- Multiple Access Methods
  - SQL and NoSQL

# Recent History

- Cluster 6.3 (2008)
  - multi-master replication, key-distribution awareness
- Cluster 7.0 (2009)
  - Multi-threaded NDB nodes
  - Ability to add nodes online to a running cluster
- Cluster 7.1 (2010)
  - ClusterJ
  - MySQL Cluster Manager
  - *ndbinfo* (and then MEM support for Cluster)
  - Windows support
- Plus
  - Extensive improvements to BLOB performance, locking behavior, node restart times, *etc.* in monthly releases

## ClusterJ Example

```
Employee findEmployee(long id) {  
    Employee employee = session.find(Employee.class, id);  
    return employee;  
}
```

- Runs faster than JDBC
- Almost as fast as the C++ NDB API
- 3 lines of code

# MySQL Cluster 7.2 Beta

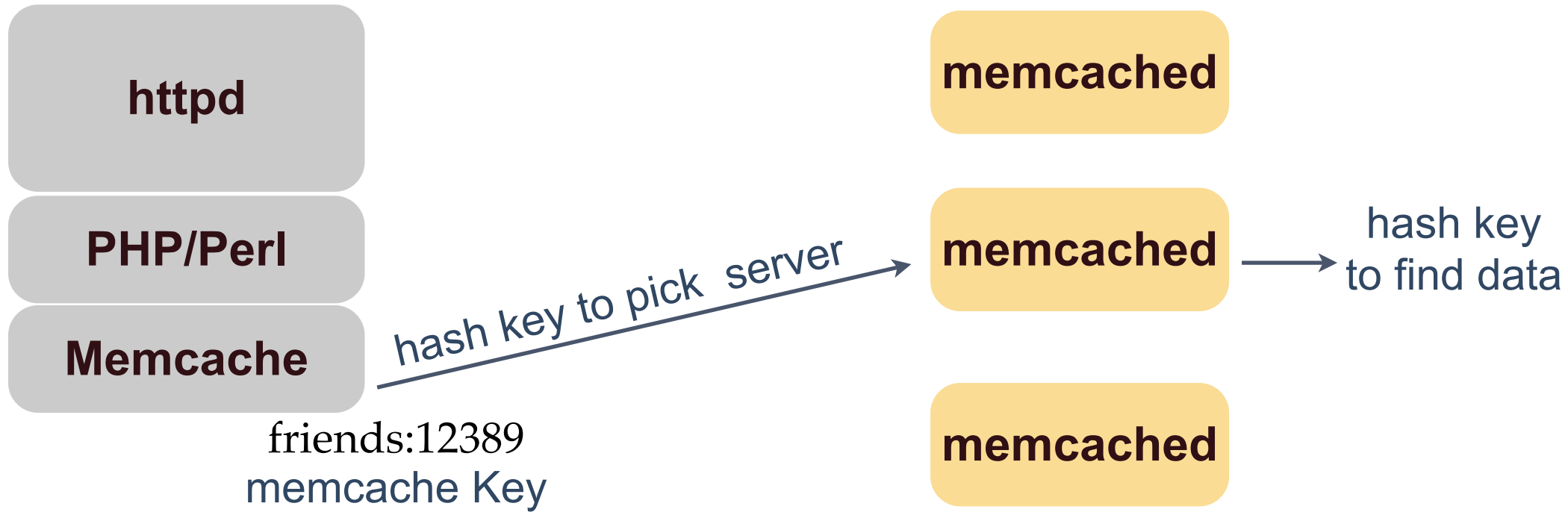
- Push-Down Joins
  - Many “SPJ” (Select/Project/Join) operations can be executed at the data nodes rather than the MySQL server
  - Long-term effort (presented at this conference last year)
  - 50x improvement for some queries
- MySQL privilege tables can be stored in cluster
- Memcache API

# About memcache

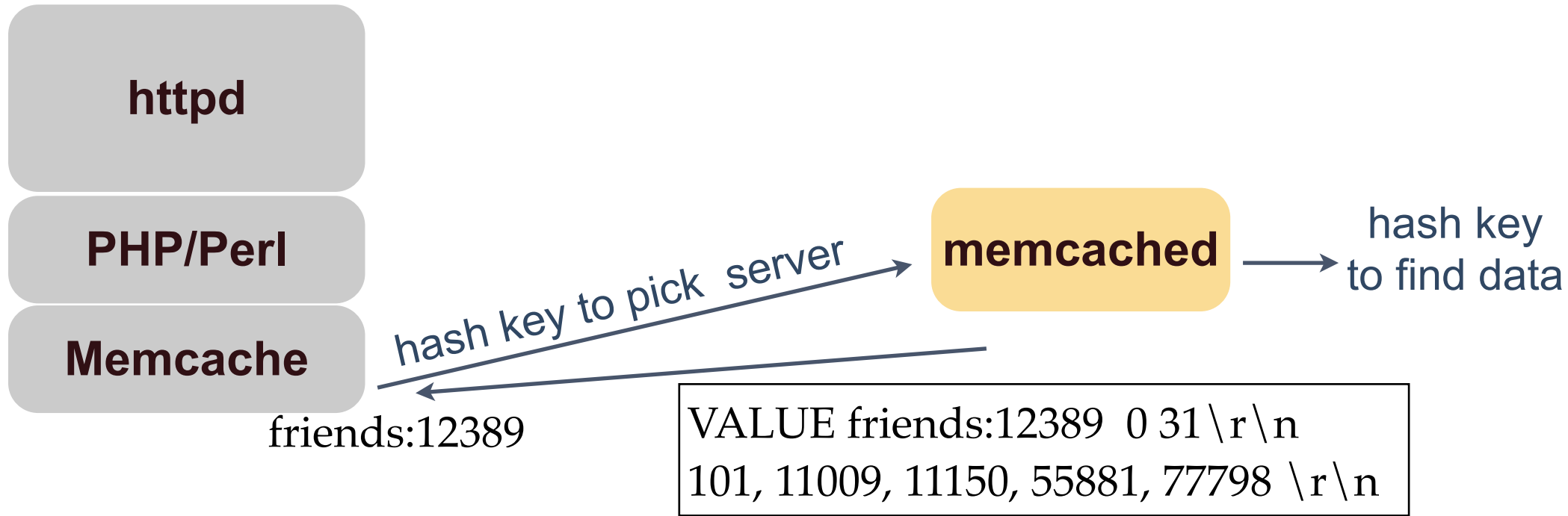


- It's a cache!
- It's not your data store!
- If it fails, you get a cache miss!

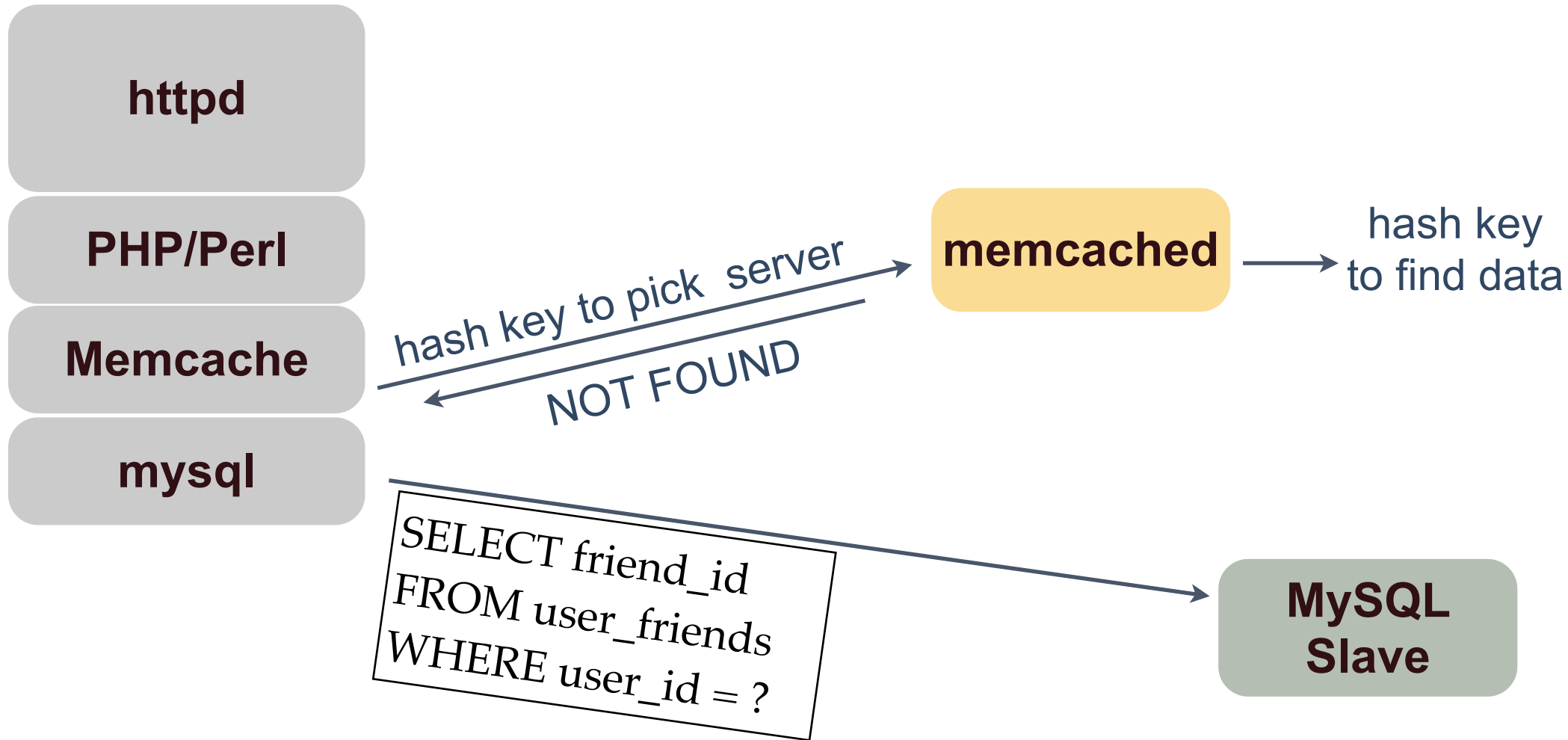
# Two levels of hashing



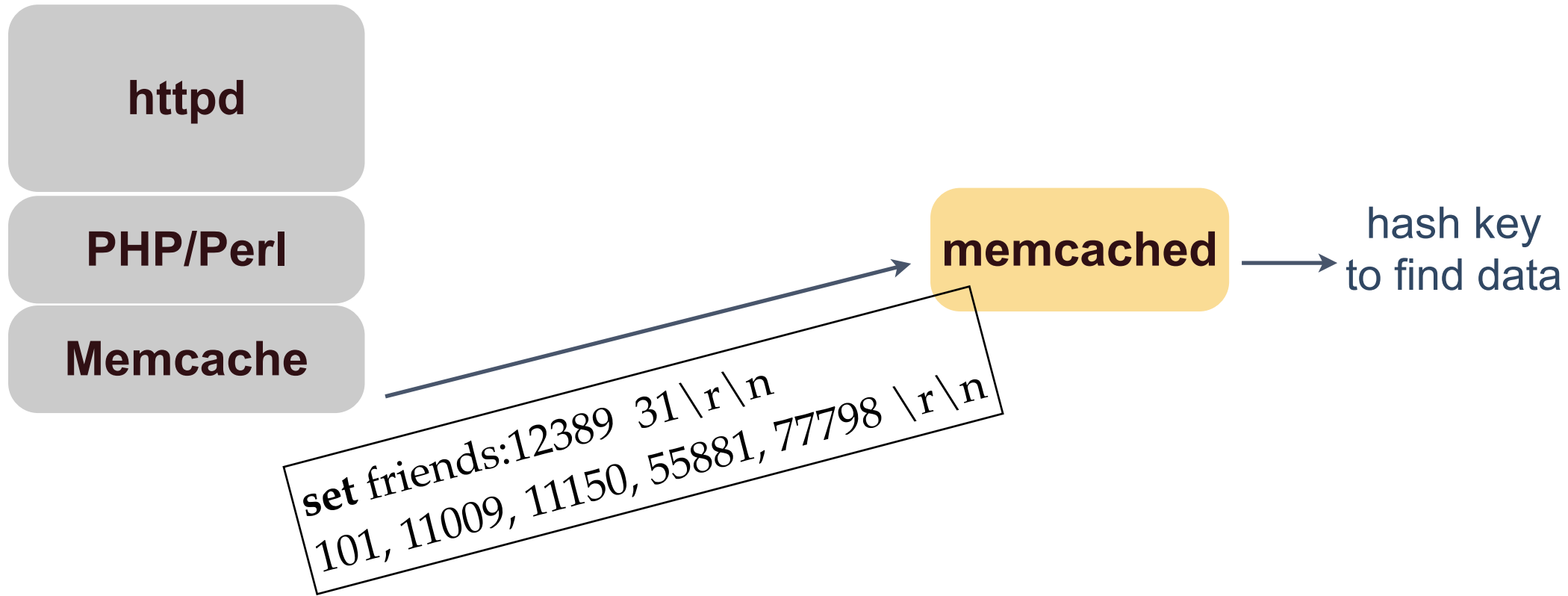
# Cache hit



# Cache miss (1): fetch from DB



# Cache miss (2): manage cache



# Data change (1): Write to DB

httpd

PHP/Perl

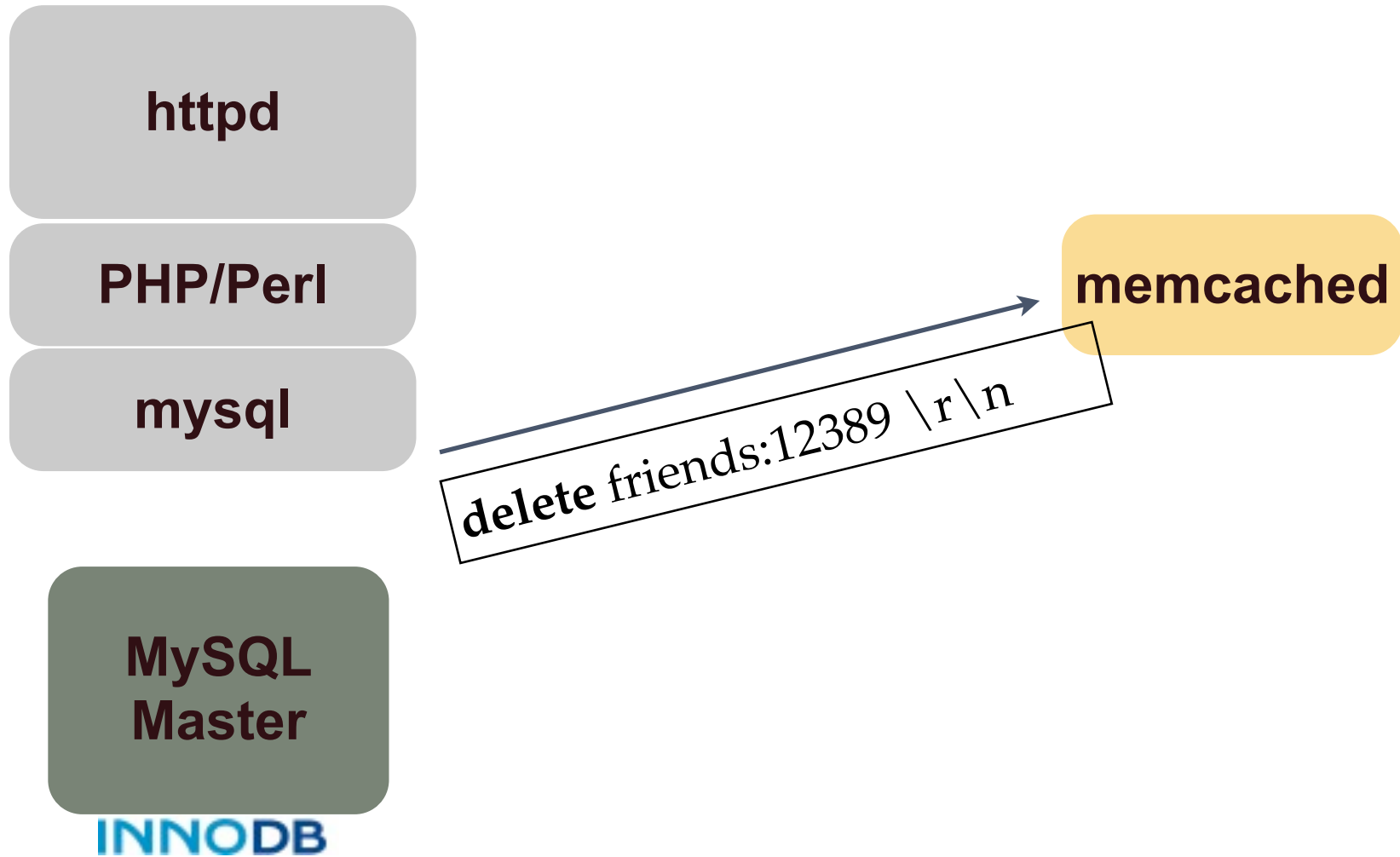
mysql

MySQL  
Master

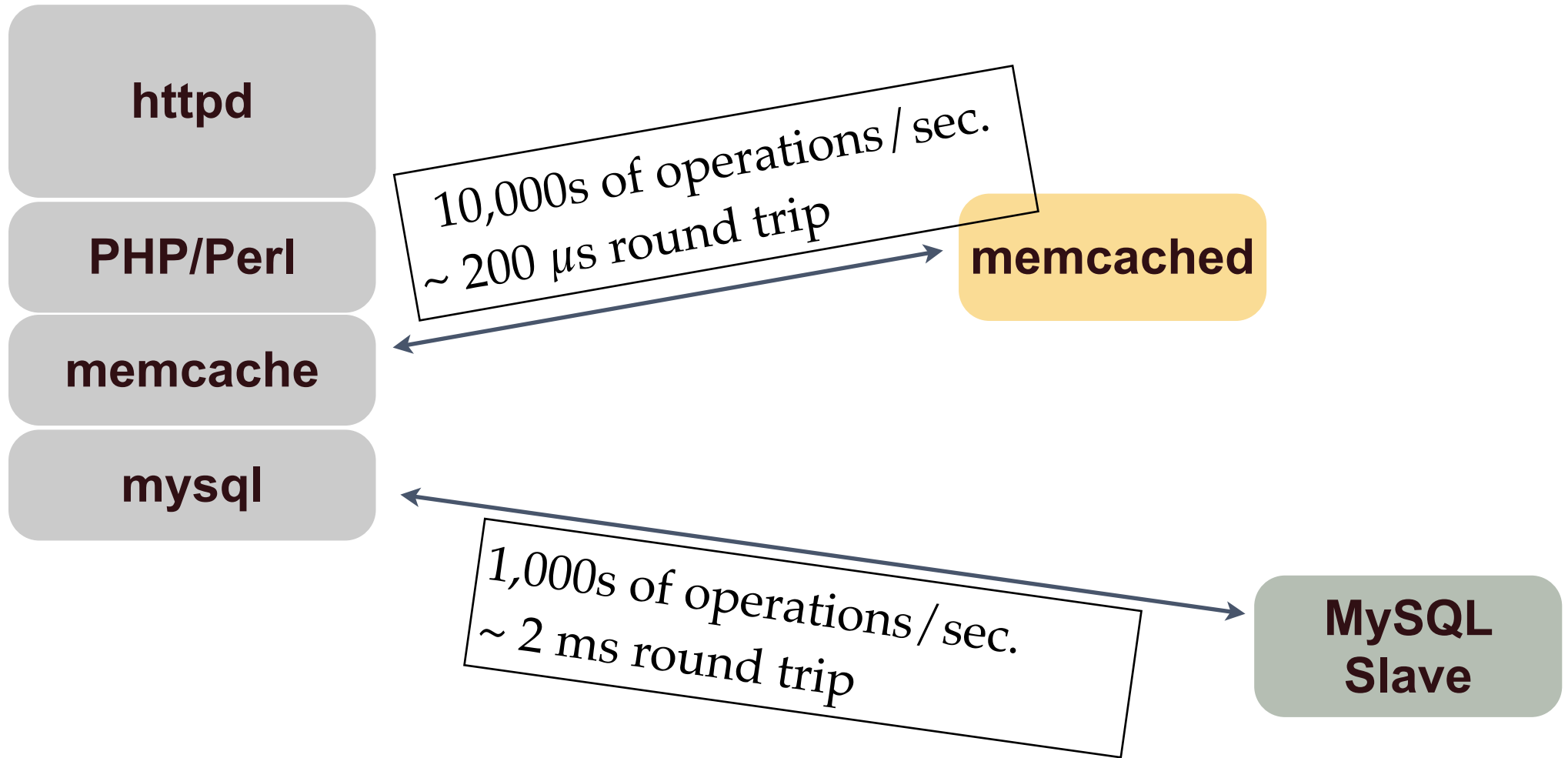
INNODB

```
INSERT INTO user_friends  
(user_id, friend_id)  
VALUES ( 12389, 999101);
```

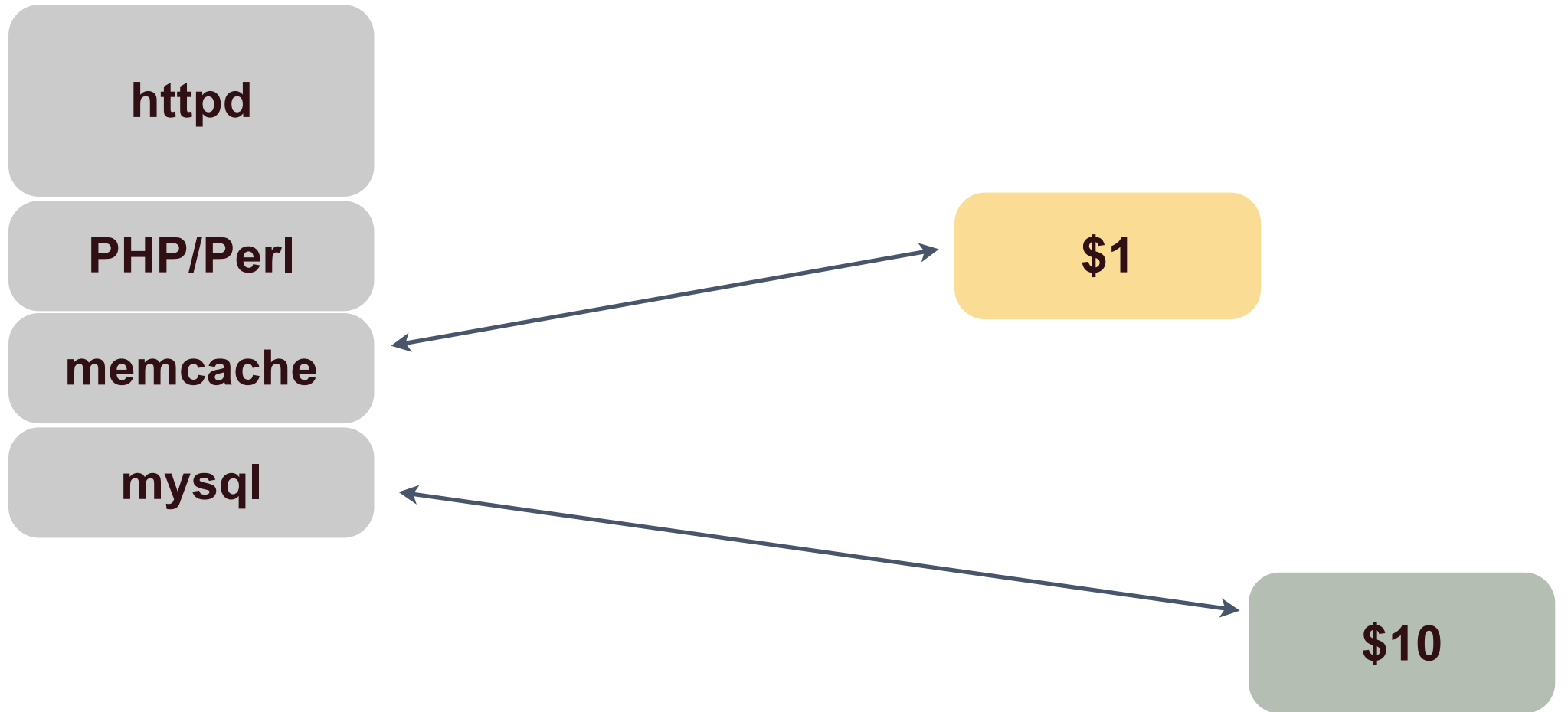
# Data change (2): manage cache



# Expected Latency & Throughput



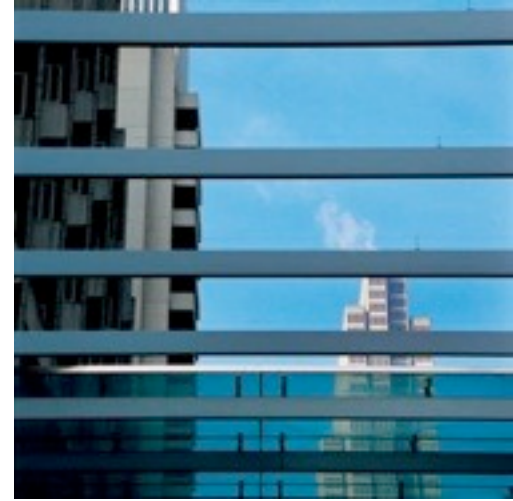
# Cost per *n* active users



## A little bit more history

- Memcached 1.2 (2007)
  - The 2000-line Facebook patch
    - UDP support
    - Vector I/O and other “details”
      - adding up to 25% improved CPU efficiency
    - Multithreaded (for a small number of libevent threads)
    - Compare-And-Set operation (CAS)
- Memcached 1.4 (2009):
  - Binary Protocol
  - SASL Authentication
- Memcached 1.6 (upcoming)
  - Storage Engines
  - Logging modules
  - Windows platform support

# Design Decisions



# Goals

- Access NDB data from memcache clients
  - *Memcached perspective:*
    - NDB is a reliable, write-scalable, replicated data store
  - *MySQL Cluster perspective:*
    - memcache is an easy-to-use high performance API

# Goals

- Support existing schemas and all MySQL data types
- Cache NDB data inside memcached
  - with automatic cache management
  - and flexibility to fine-tune (or disable) the cache policies
- Support the whole memcache protocol
- Achieve superior performance
  - *latency* as expected from memcached
  - *throughput* as expected from memcached

# Which codebase?

	memcached.org	libmemcached	build our own
Text Protocol	✓		
Binary Protocol	✓	✓	
TCP	✓	✓	
UDP	✓		
Authentication	✓		
	✓		

# Server Architecture

- **Memcached separate from ndbd**
  - *Definitely!*
  - “m:n” ratio of memcache servers to data nodes

memcached

memcached



- **Memcached inside ndbd**
  - *Maybe!*
  - we know how to do it
    - “embedded” API-to-TC channel
  - lower latency
    - *network round trip + tens of microseconds*
  - Not as flexible

memcached



# Decisions

- Use memcached 1.6 tree
- Isolate the NDB-specific code into an *ndb\_engine*
  - which currently exists outside the memcached source tree
  - and runs in an unmodified memcached server
- Build a standalone server binary
- Possibly also build an *ndbmtd-embedded* server
- Share code with InnoDB Memcache team
  - on basic configuration and user-visible concepts
  - and on changes that would allow a memcache server module to be loaded into either ndbmtd or mysqld

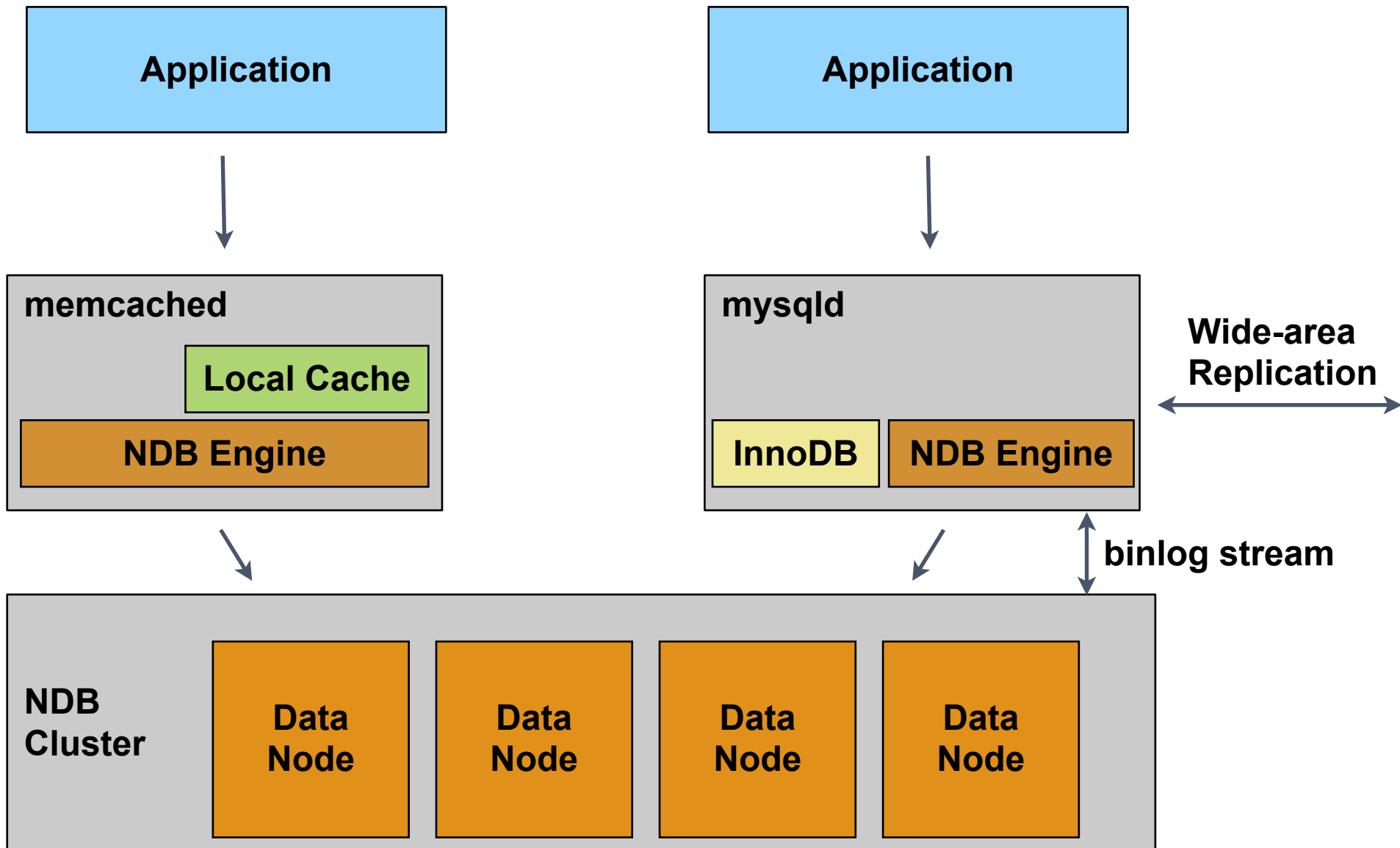
## Other decisions

- Memcache INCR & DECR
  - Fully supported
  - Atomic operations
  - Performed at the data node
- Memcache CAS (version id check)
  - Fully Supported
    - either at local cache or at database
  - CAS check is pushed down to the data node
  - Ideally an update that comes from a non-memcache API node should invalidate the CAS
    - *we still need to decide the best design for this*

# Limitations

- The size of stored values is limited to the NDB row size
  - currently just less than 8 KB
  - vs. 1 MB typical limit for memcached
  - due to the lack of BLOB support in the async NDB API

# Architecture Overview



# Configuring **It**



# What's Configurable

- Does it use local cache?
- Does it use NDB?
- What columns hold the keys?
- What columns hold the data?
- Are memcache commands like DELETE and FLUSH allowed to delete records from the database?
- *You decide all of this ...*
- *on a “per-key-prefix” basis.*

# A Key Prefix



# Fundamentals

- memcached command line specifies a connectstring for a *primary cluster*
- primary = “where the config is stored”
- The NDB Engine reads the configuration tables from the *ndbmemcache* database on the primary cluster
- You, the administrator, manage the config via SQL
- An NDB Memcache server can operate on data in the primary cluster or in other clusters
- Different NDB Memcache servers can fetch different configurations

# Standard Tables in *ndbmemcache*

- meta
  - stores configuration schema version (for upgrade compatibility); consider it to be read-only
- ndb\_clusters
- containers
  - where data is stored
- cache\_policies
  - how it can be accessed
- key\_prefixes
- memcache\_server\_roles
- last\_memcached\_signon
- demo\_table

# ndb\_clusters

- **cluster\_id**
  - int, referenced by key\_prefixes
- **ndb\_connectstring**
  - varchar(128) – how to reach this cluster
- **microsec\_rtt**
  - default 250; used for internal performance tuning

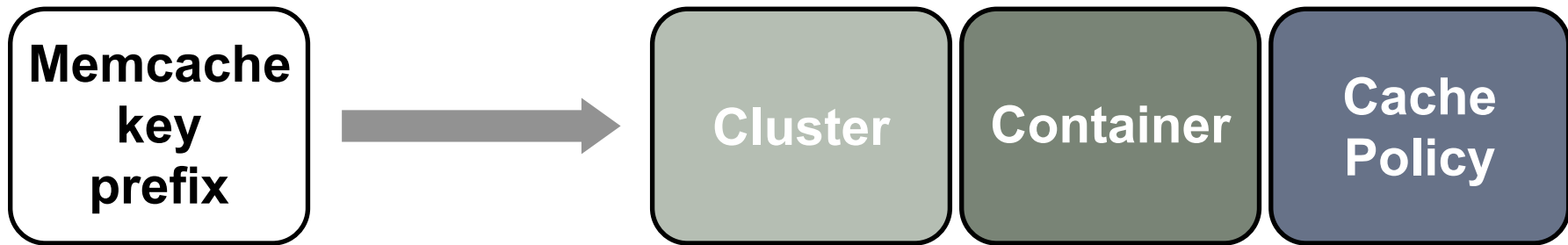
# containers

- **name**
- **schema\_name**
- **table\_name** (*the existing table where your data lives*)
- **key\_columns** (*comma-separated*)
- **value\_columns** (*comma-separated*)
- **flags** (*either a constant number, or a column name*)
- **increment\_column** (*optional, for INCR/DECR*)
- **cas\_column** (*optional*)
- **expire\_time\_column** (*optional*)

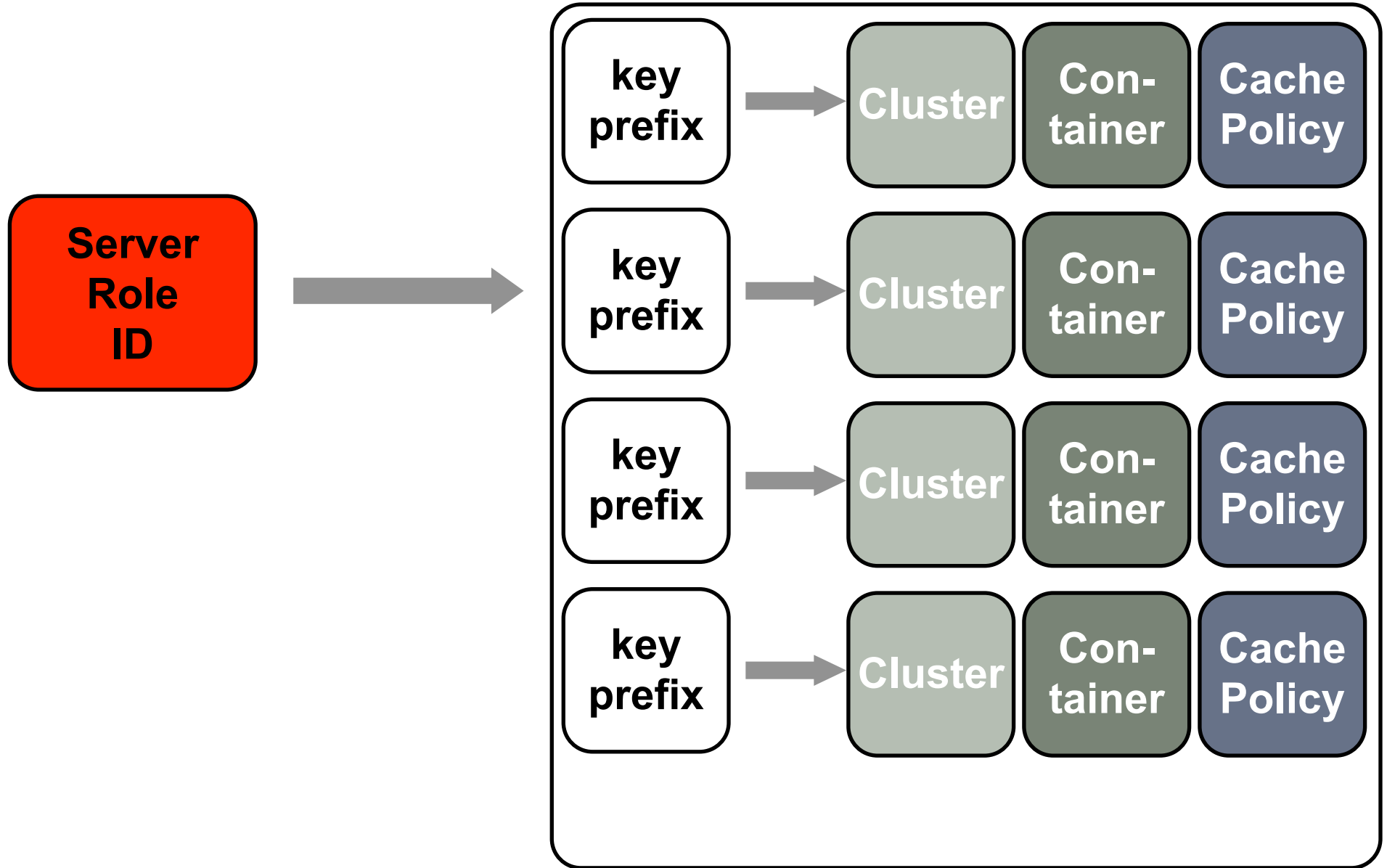
# cache\_policies

- **policy\_name**
- **get\_policy**
  - enum (cache\_only, ndb\_only, caching, disabled)
- **set\_policy**
  - enum (cache\_only, ndb\_only, caching, disabled)
- **delete\_policy**
  - enum (cache\_only, ndb\_only, caching, disabled)
- **flush\_from\_db**
  - enum(false, true)

# A key-prefix mapping



# A memcache server role



# key\_prefixes

- **server\_role\_id**
- **key\_prefix**
- cluster\_id
- policy
- container

## demo\_table

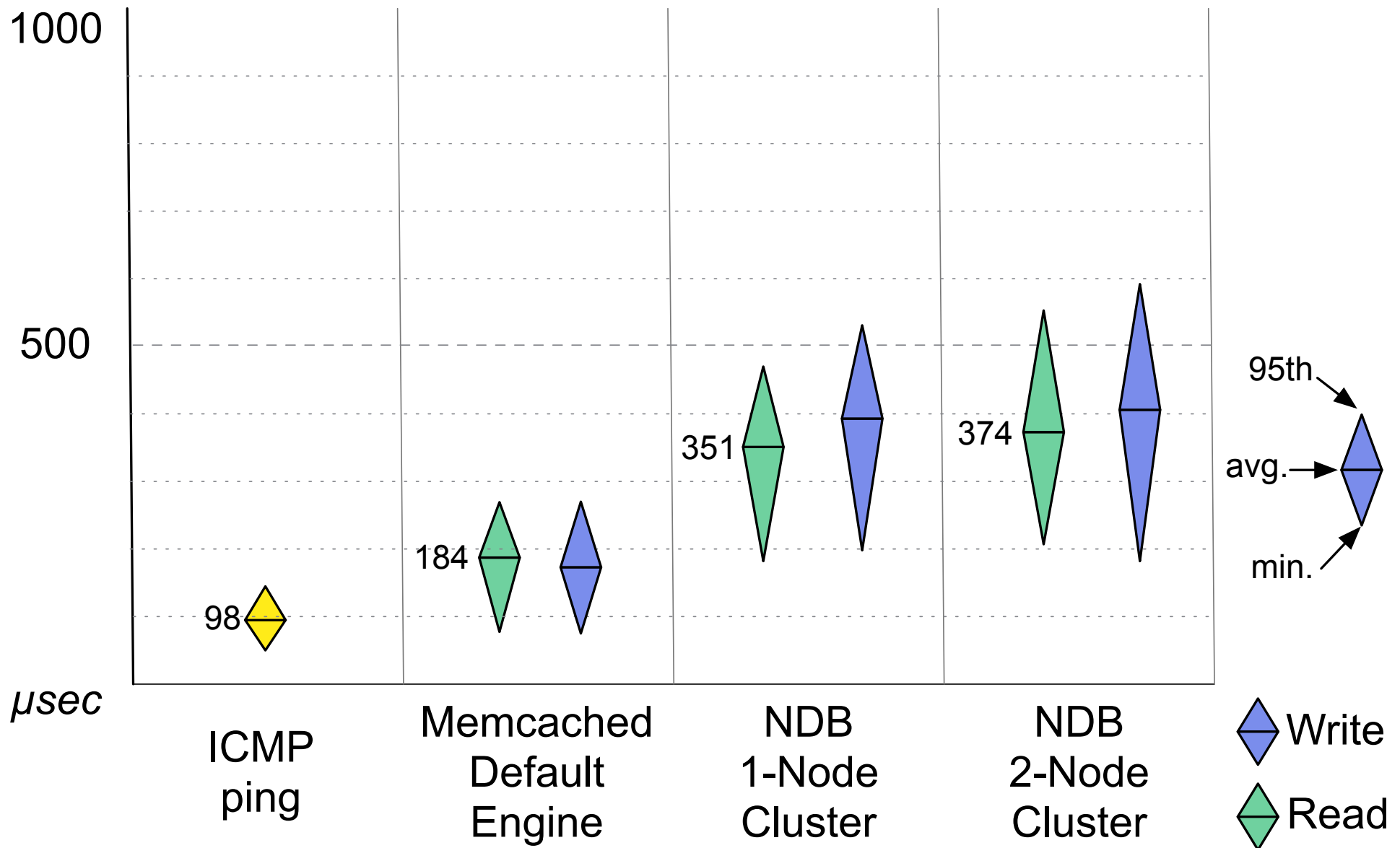
- *Not really part of the configuration ...*
- mkey
  - varchar(250) NOT NULL PRIMARY KEY
- math\_value
  - bigint unsigned
- cas\_value
  - bigint unsigned
- string\_value
  - varchar(7500)

# Performance



# Measured Latency

memcachetest -t 2 -M 7000 -c 25000



# Links

- This Presentation
  - <http://en.oreilly.com/mysql2011/>
- Memcached 1.6 development tree
  - `git clone git://github.com/memcached/memcached`
  - `cd memcached`
  - `git checkout -t origin/engine-pu`
- MySQL Cluster 7.2 with Memcached Engine
  - `bzr clone lp:~mysql/mysql-server/mysql-cluster-7.2-labs-memcached`
  - or look for it at ***labs.mysql.com***
- Mailing List
  - `cluster@lists.mysql.com`

# Demo

- What we'll see:
  - Memcache Server Window
    - start memcached “sandbox”
      - default role (NDB-only)
      - mc-only role (plain memcached)
      - ndb-caching role (NDB with local cache)
  - Client Window
    - run memcapable
  - MyQSL Window
    - examine data and configuration

ORACLE®