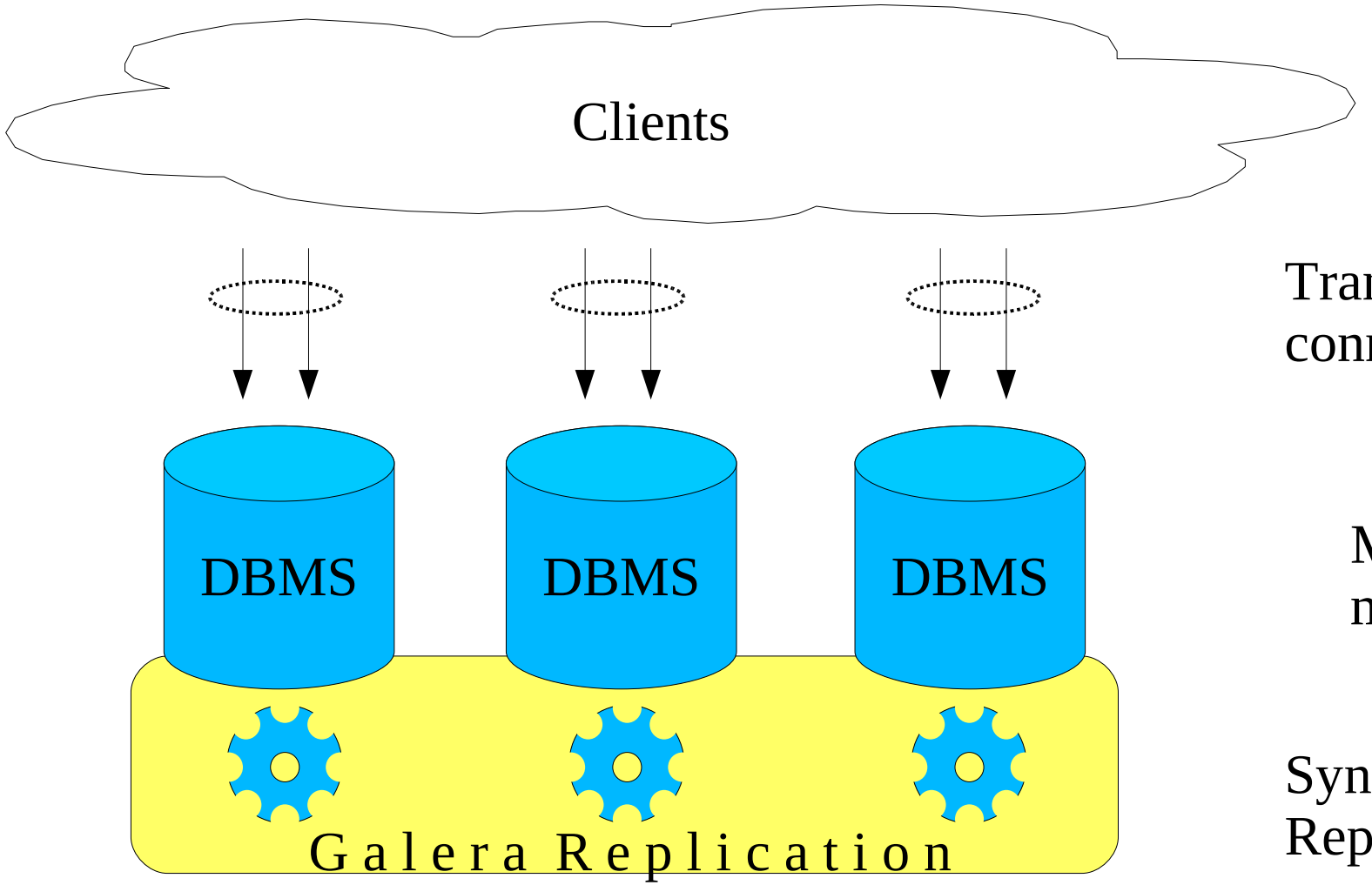
Galera Replication

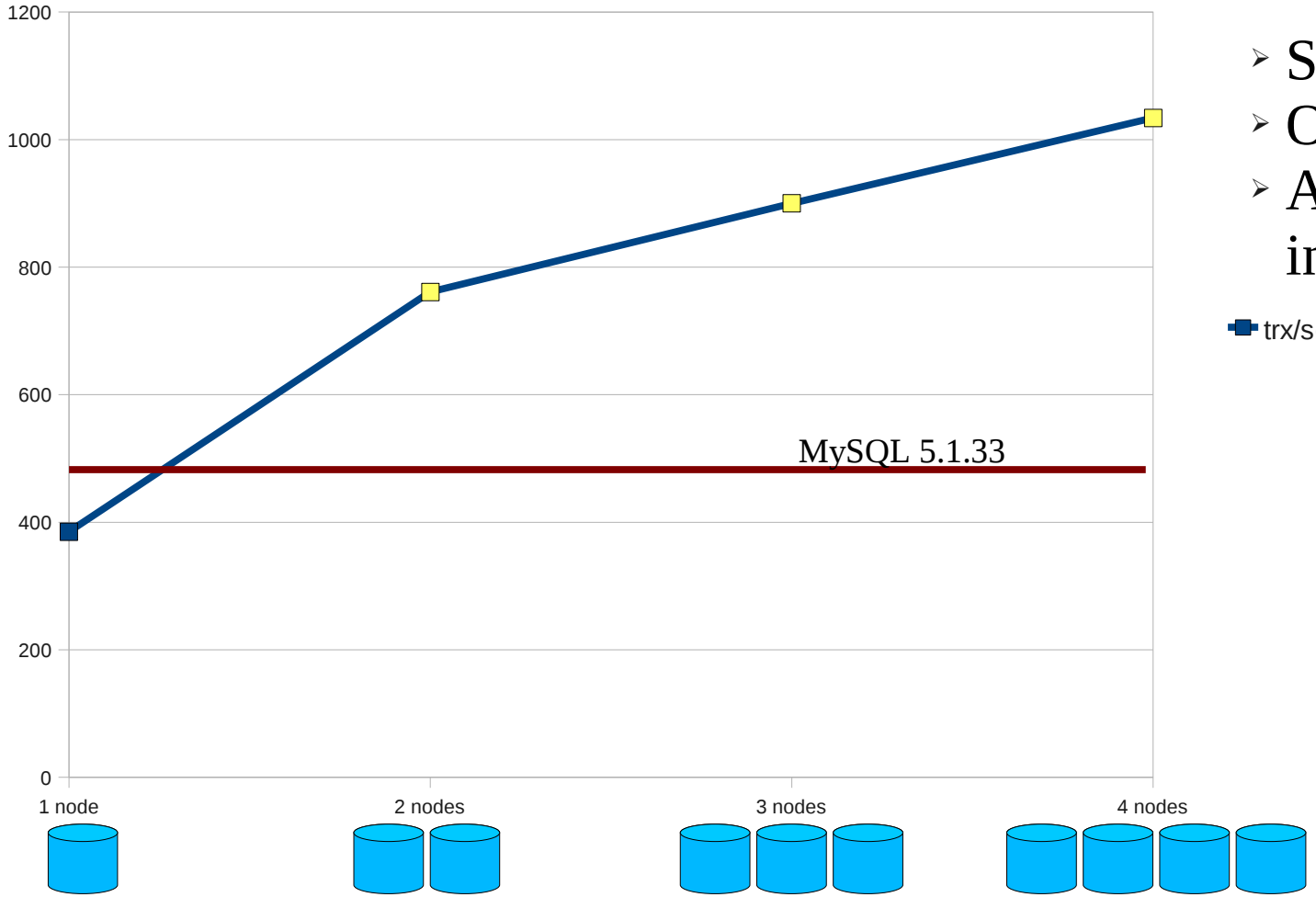


Transparent connections

Multi-master

Synchronous Replication

Pretty Good Scalability



- Sysbench
- OLTP mode
- Amazon EC2, large instances

Contents

About Me & Codership

- Me = Seppo Jaakola, CEO
- Friends = Alexey Yurchenko,
  Teemu Ollakka
  + hang around guys
- Fin-Rus community working from Finland
- Several years experience in building networking applications, distributed systems, clustering solutions, firewalls etc...
- We enjoy writing code, that's why we founded Codership
- Set Sails Oct 2007

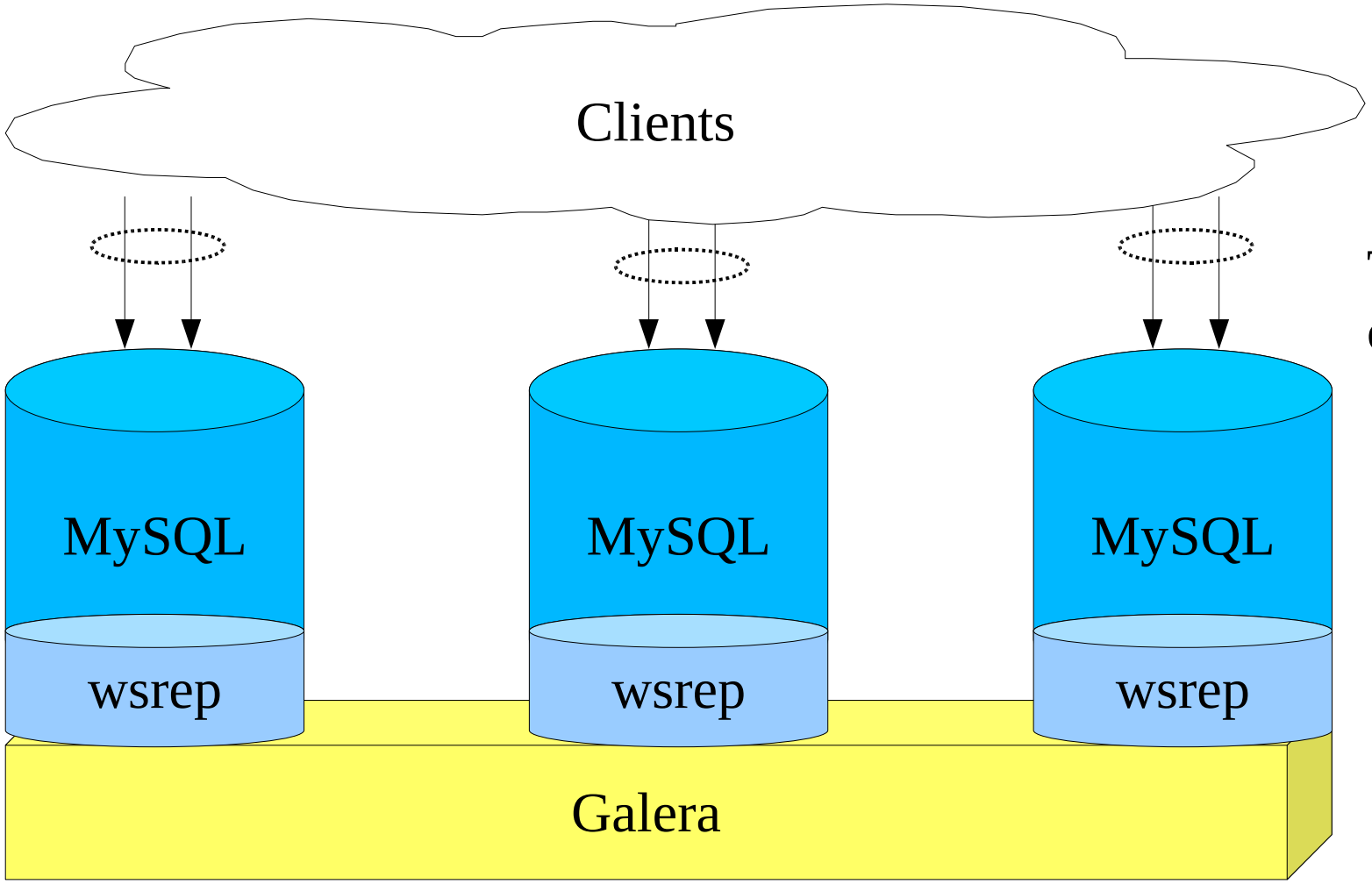
Galera Replication

- Certification based replication model (based on academic research by F. Pedone et al)
- Multi-master synchronous replication
  - ➔ High Availability
- No middle-ware, connections directly to DBMS
  - ➔ Transparency
- Row level locking
  - ➔ Write scalability

Galera Replication

- Galera is implemented as software library
- Generic replication system to make a cluster from any transactional DBMS
- First implementation MySQL/Innodb cluster
  - More to come...

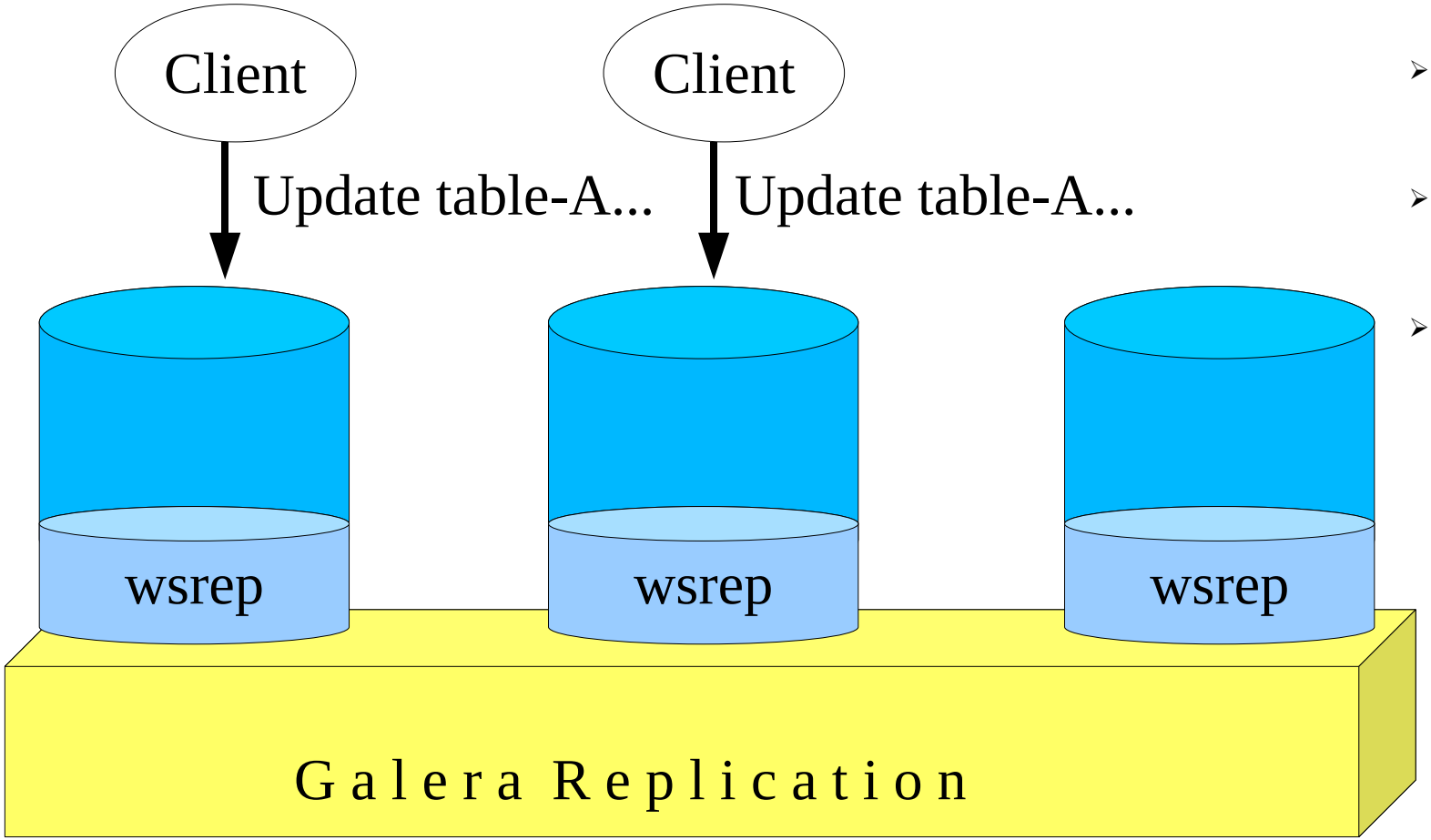
Galera Cluster



Transparent connections

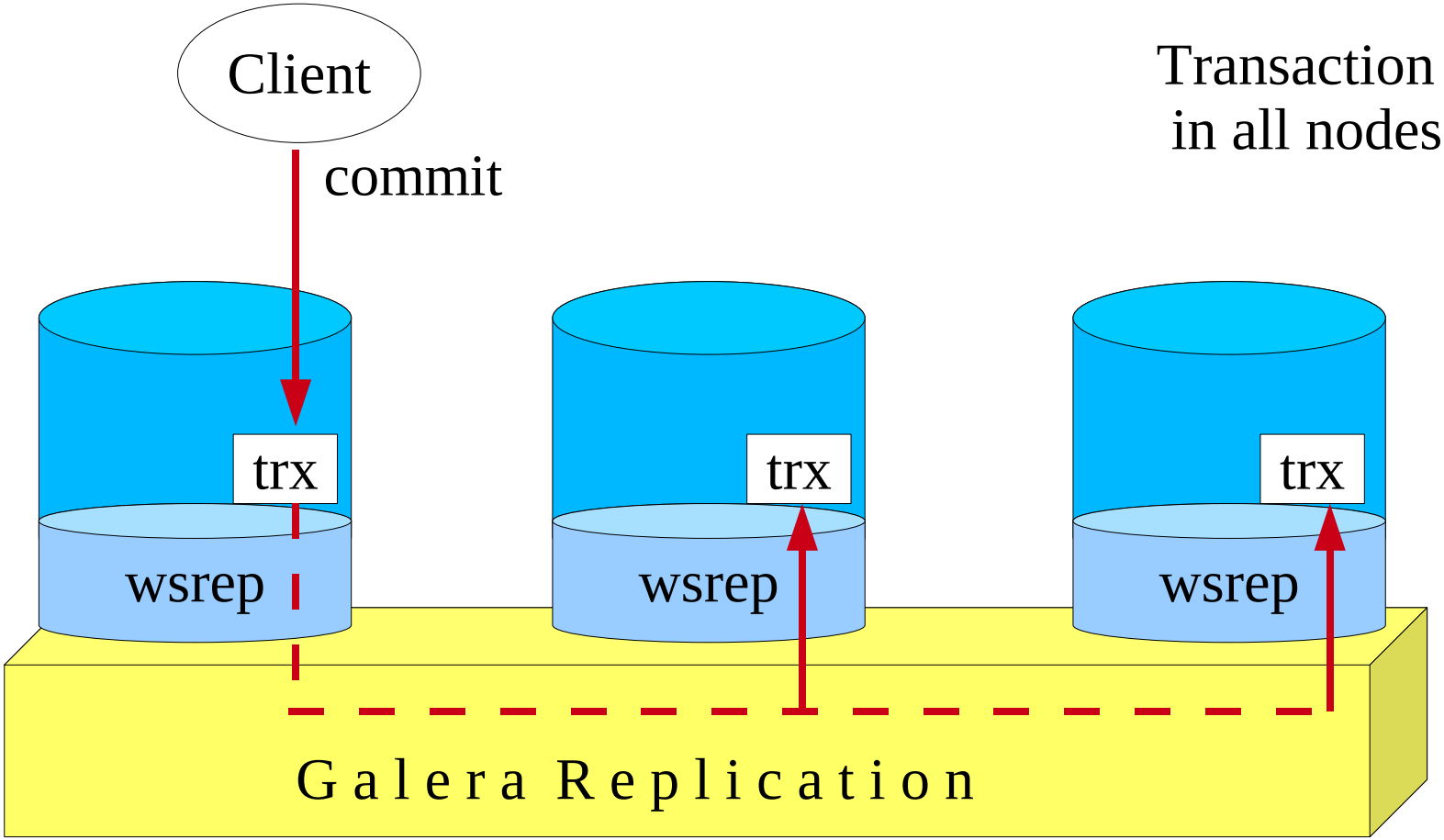
Multi-master

True Multi master



- No restrictions for write access
- Galera sorts out write conflicts
- Client sees cluster node as standard DBMS server

Synchronous Replication



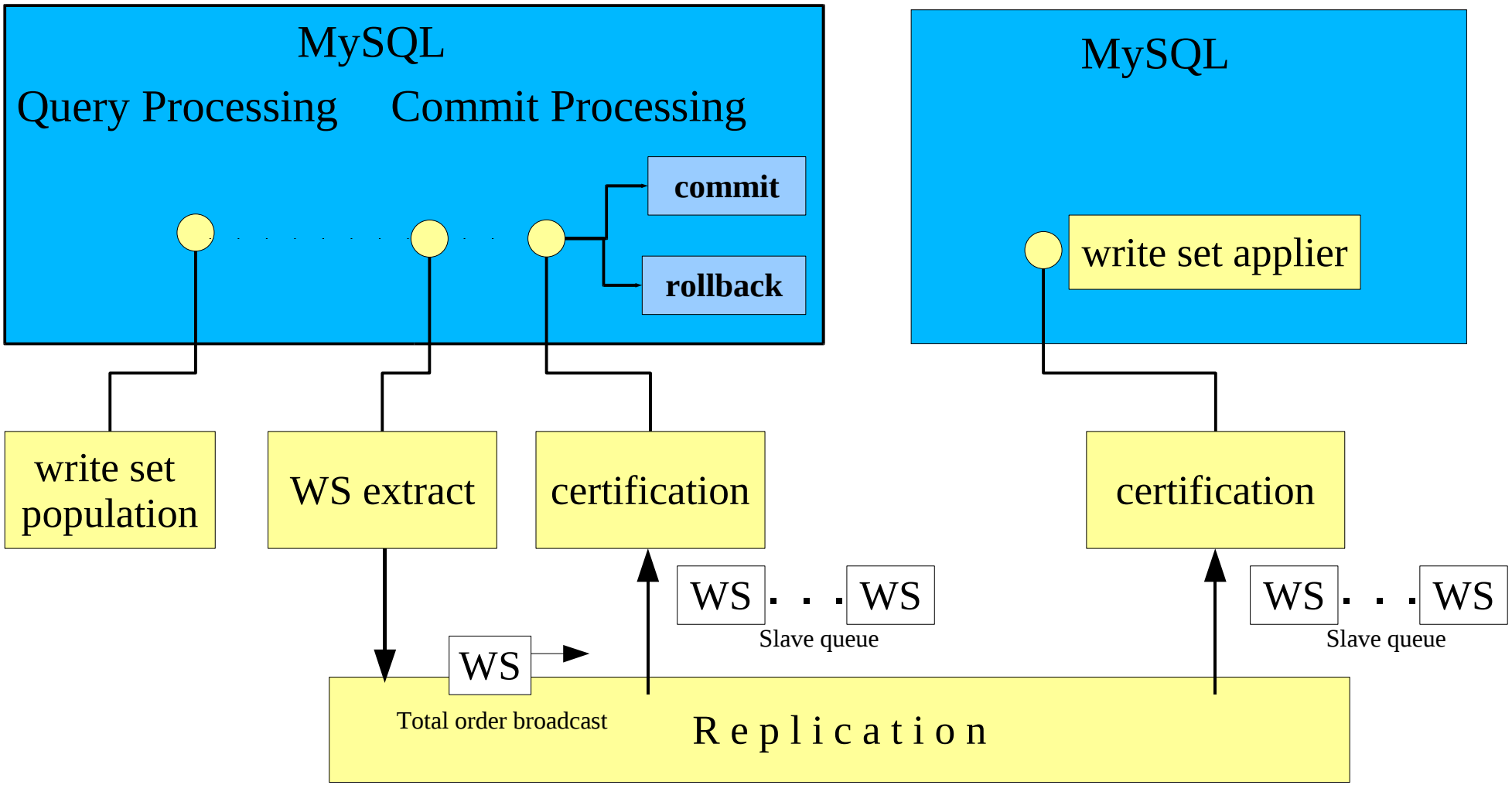
Transaction is committed in all nodes => HA

Certification Based Replication

Warning: Scientific content

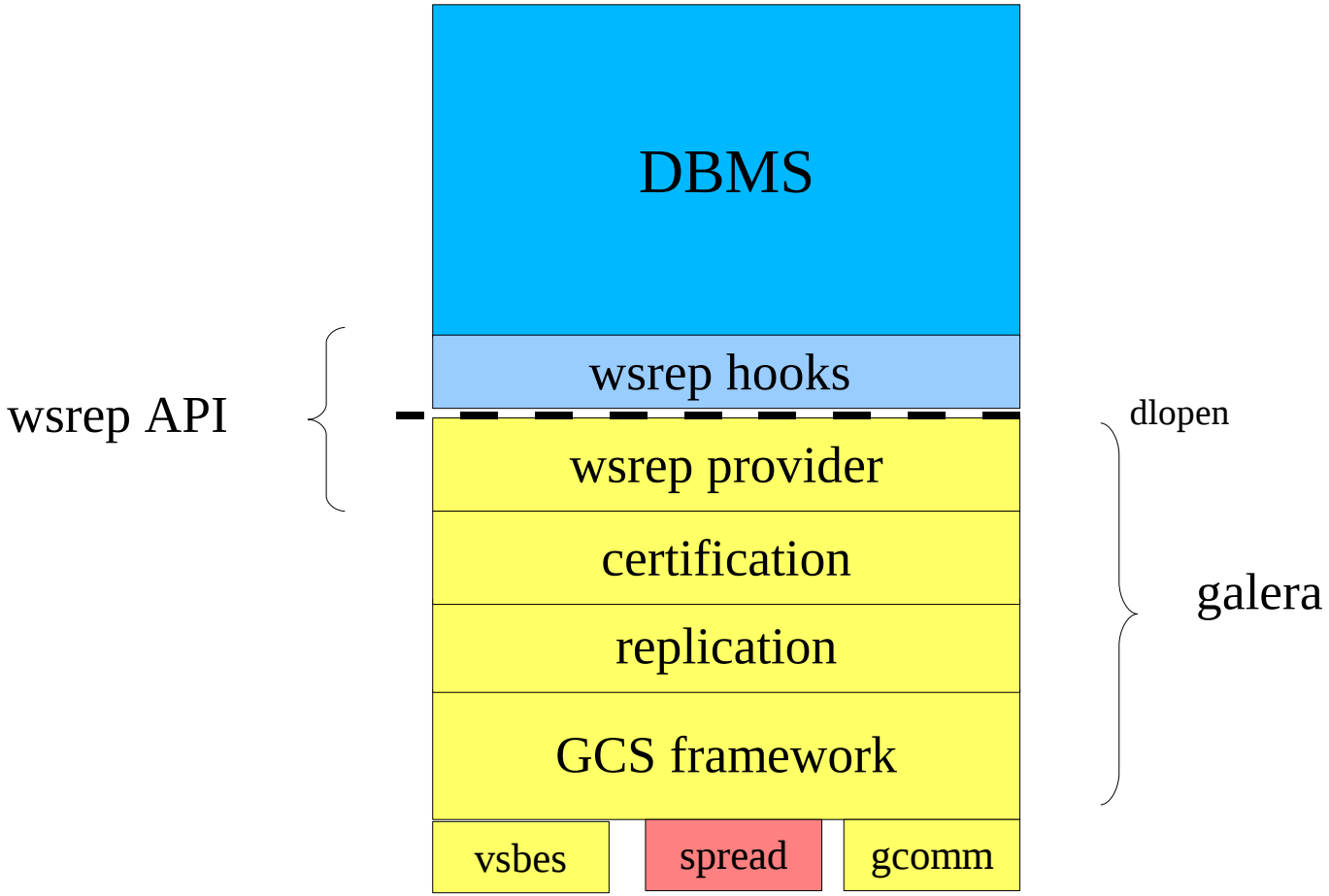
- At commit time, a write set (ws) is extracted
- GCS assigns seqno for this ws and broadcasts to the cluster
- There can be a set of write sets in replication queue, which have not yet committed
- Certification test check conflicts against this set
- Ws contains all information to run the certification test deterministically in each cluster node
- Snapshot isolation

Certification Based Replication



Generic Replication

First we take MySQL, then we take Postgres



wsrep API

- Defines a generic interface for DBMS and replication system
- Write set replication API for transactions
- DDL replication using TO isolation
- https://launchpad.net/wsrep

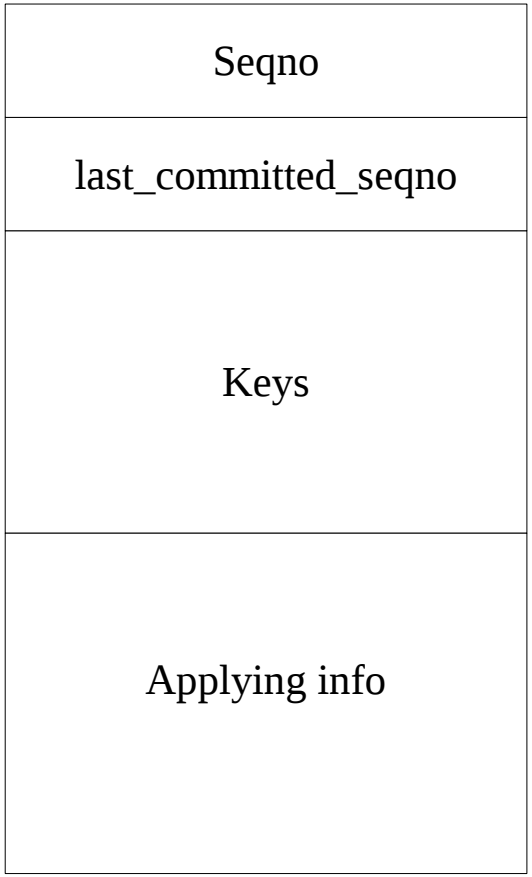
wsrep integration in MySQL

- Calls to wsrep provider:
  - ws populating,
  - replication...
- Handlers for various wsrep callbacks:
  - ws applying
  - DDL applying ...
- Changes in innodb code to provide ***prioritized transactions***
- https://launchpad.net/codership-mysql

Write Set

- Contains all information needed for certifying and applying of transactions
- Data changes can be specified in several different replication levels:
  1. **SQL statement**
  2. Lex structures (AST) from parser
  3. **RBR event**
  4. Row (as binary image)
  5. Column value

Write Set



- Sequence number of the trx
- Sequence number of the last committed trx, which affected the processing state
- All row changes are identified with keys
- SQL statements or RBR events

Write Set

- By default write sets are flushed asap
- However, key information is stored in certification index
- Write sets can also be saved for future needs:
  - wsrep_ws_persistency = ON/OFF
  - For debugging purposes
  - For incremental state transfer

Recap

- ...learnt so far:
- Galera is true multi-master
- And synchronous replication
- Certification based replication model
- Provided by generic software library
- wsrep API defines the interface
- ...and then...

Advanced Replication Features

- Optimistic concurrency control
- Flow control
- Auto increment management
- Asymmetric lock granularity issue
- Parallel applying
- Retrying of aborted autocommit trxs
- etc...

Optimistic Concurrency Control

- Transactions proceed independently in each cluster node assuming they can eventually commit
- Certification test tells if trxs committing from different nodes conflict
- Victim trx must abort in master node, and avoid committing in other nodes
- WS applying happens with high priority and this can further abort local trxs
- ER_DEADLOCK returned for cluster aborts

Optimistic Concurrency Control

- Hot spots are bad
  - .e.g. DBT2 shows pretty high conflict rate
- Long lasting transactions are vulnerable
- Rollbacks eat performance, but rollback happens only in one node, all the rest nodes just avoid applying
- With problematic SQL load, cluster could be adjusted to have a smaller number of write capable nodes

Flow Control

- Synchronous operation requires that each node will process evenly
- Without any flow control some node(s) would hijack all master activity => slave queues would grow indefinitely
- Group communication actions for managing flow control
- Limits for slave queue length, which trigger flow control

Autoincrements

- Galera can manage automatically autoinc control variables (auto_increment_increment, auto_increment_offset)
- This happens optionally: wsrep_autoinc_control = ON/OFF
- Autoincrement control is triggered by cluster membership changes:
  - Increment = number of cluster nodes
  - Offset = node id
- Autoincrement control can yield best possible insert performance

Asymmetric Lock Granularity

- Write set defines modified rows by unique key values
- When applying the write set, DBMS may need to lock more “resources” (like InnoDB gap locks)
- Applying several write sets in parallel can lead to conflicts

Maximal Insert Performance
(theoretical thought content)

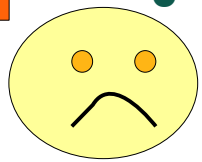
- Even faster inserts?
- Due to autoincrement management, inserts won't conflict
  - ➔ Transactions with only inserts (...and which get auto inc value) don't need to certify
  - ➔ We can/could skip certification test and process inserts even faster

Retrying

- “Cluster can abort trxs at will”
- If autocommit trx was aborted due to cluster wide deadlock, it is safe to immediately retry the trx
- Option: wsrep_retry_autocommit = ON/OFF

Parallel Applying

- It is possible to launch several appliers
  - wsrep_slaves = n, option
- Appliers run in parallel until commit time



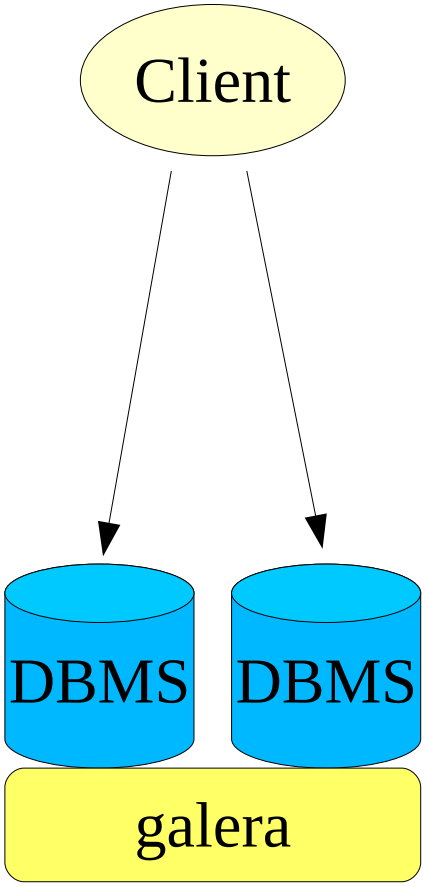
But: no performance gain has been observed

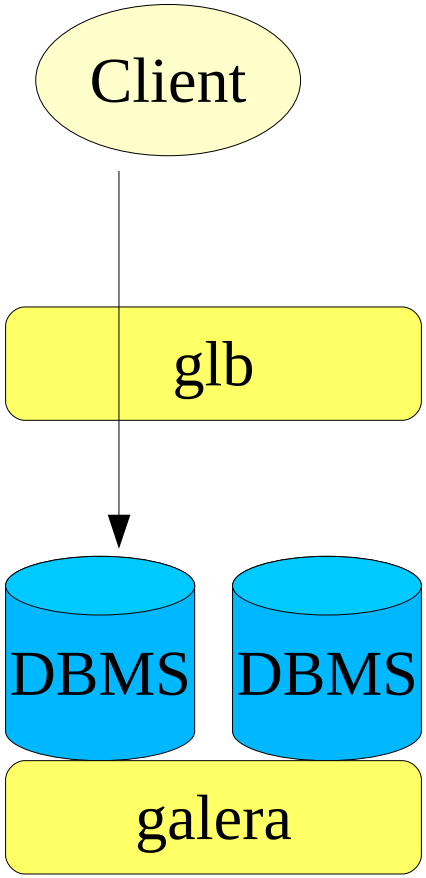
  - RBR event applying is very fast compared to SQL query processing
  - Galera cluster scales even with 100% write rate
  - Flow control cuts some of parallelism
- (Implementation suffers from ALG issue)

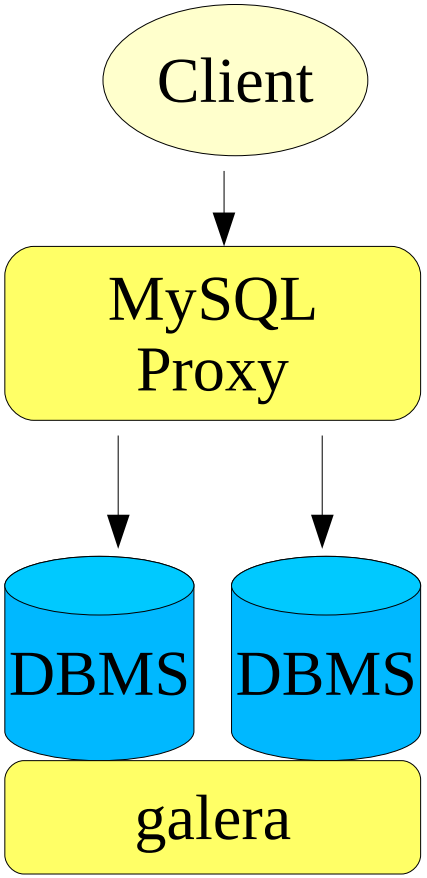
Connecting

- Galera is true multi-master, clients can directly connect to any node
- Also connection pool can know the node addresses and balance connections to the cluster
- If single connection point is needed:
  - TCP load balancer is viable option (like Galera load balancer, glb)
  - Proxy component (MySQL Proxy)

Connecting


Client
DBMS
DBMS
galera

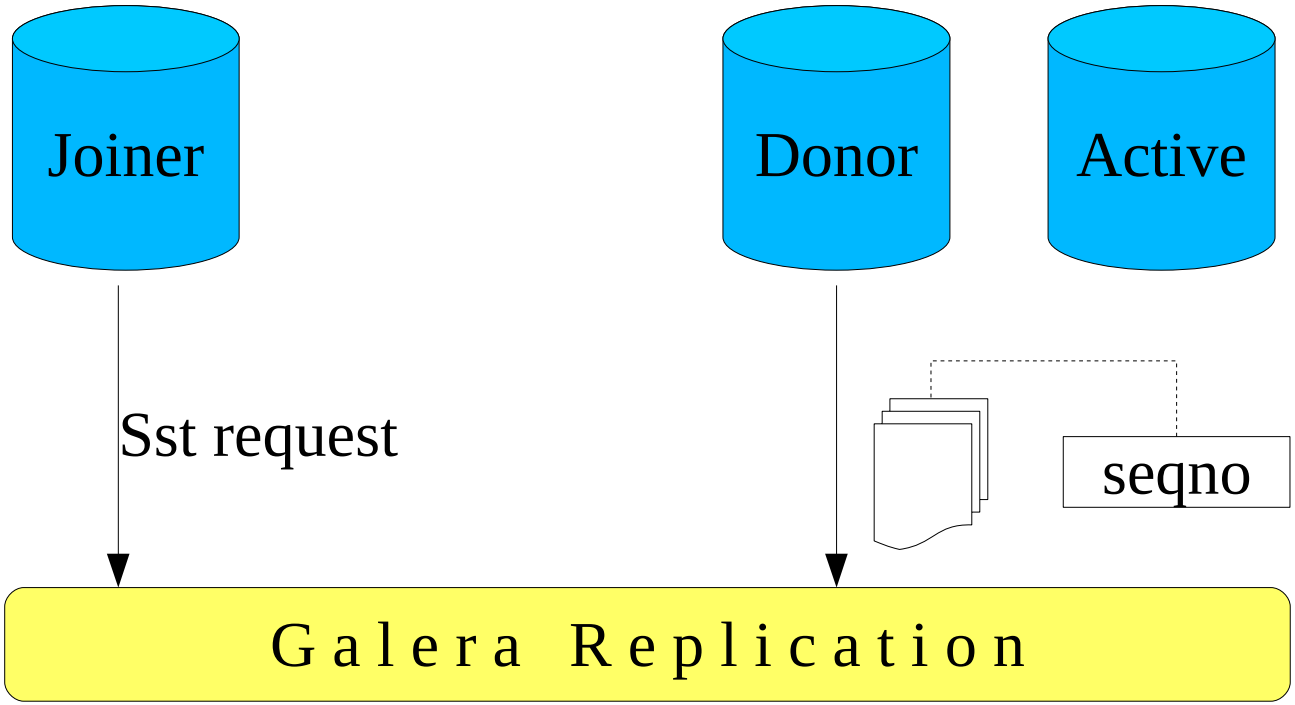

Client
glb
DBMS
DBMS
galera


Client
MySQL
Proxy
DBMS
DBMS
galera

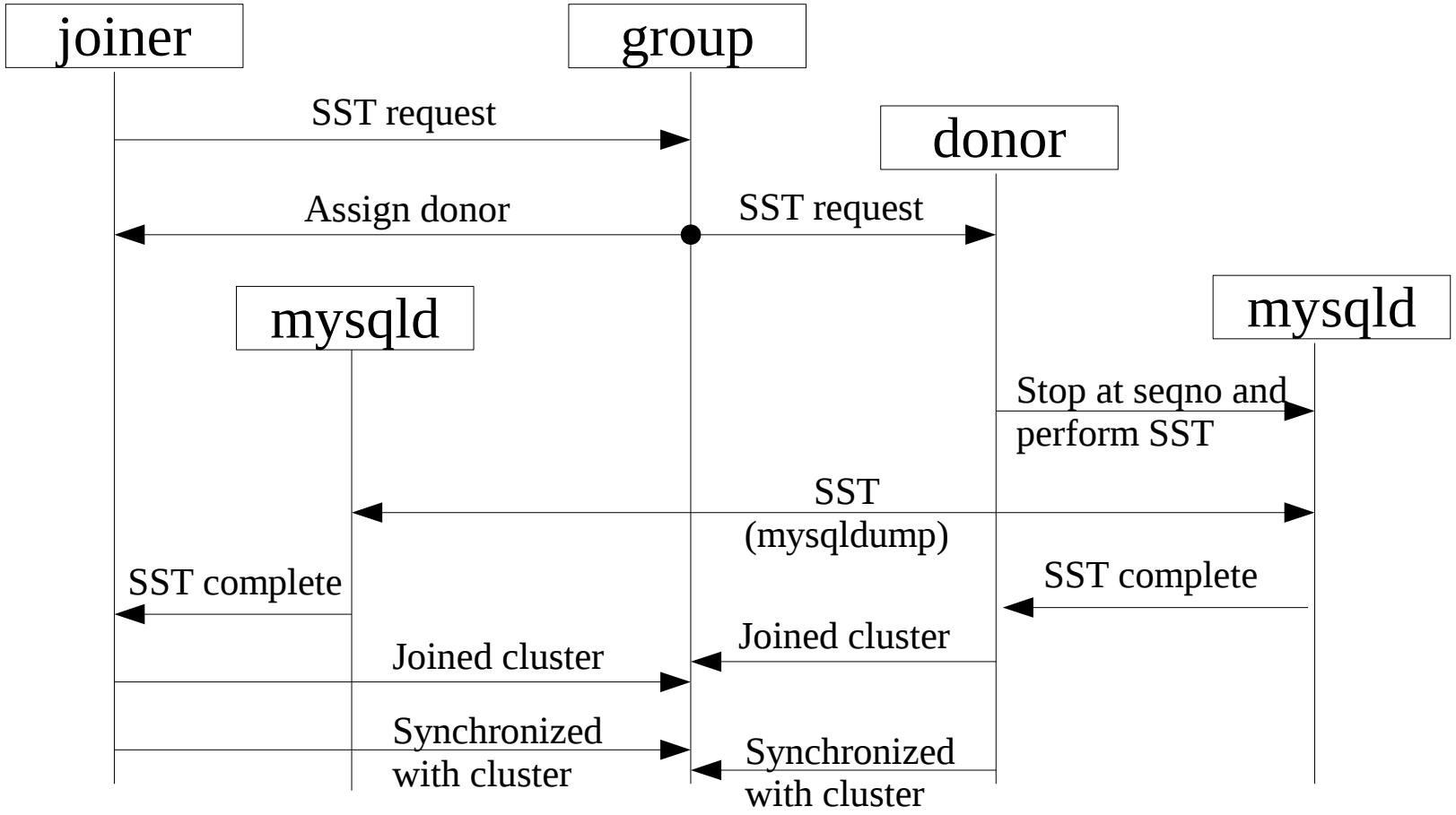
DB State Transfer

- To join new node in running cluster
- Copies DB state from an active node to the joiner
- The task is:
  - Donor must prepare a DB snapshot in known position of replication stream (seqno)
- DB State options (for innodb):
  - Mysqldump release 0.7
  - Xtrabackup release 0.8
  - LVM release 0.8
  - InnoDB HotBackup

DB State Transfer



SST in 0.7


joiner
group
donor
mysqld
mysqld
SST request
Assign donor
SST request
Stop at seqno and
perform SST
SST
(mysqldump)
SST complete
SST complete
Joined cluster
Joined cluster
Synchronized
with cluster
Synchronized
with cluster

Limitations

Twist in my Transparency

- Commit can return ER_DEADLOCK
- Implicit snapshot isolation
- SQL level limitations:
  - Locking session
  - Load data infile not supported (yet)
  - Lock functions

Lock Tables?

- No!
- Locking sessions require managing long term session to each node
- ➔ Complicated and error prone

Load Data ?

- “Load data infile” requires that each server has identical file in place
  - ➔ Not very practical in cluster
- “Load data local”, possible to implement but:
  - ➔ is not yet supported

Myisam support?

- Can be supported, if myisam writes will go through one selected node
  - ➔ Myisam support will fall back to master slave replication (synchronous)
- Galera can reject myisam writes in any other node

Large Write Sets

TODO list

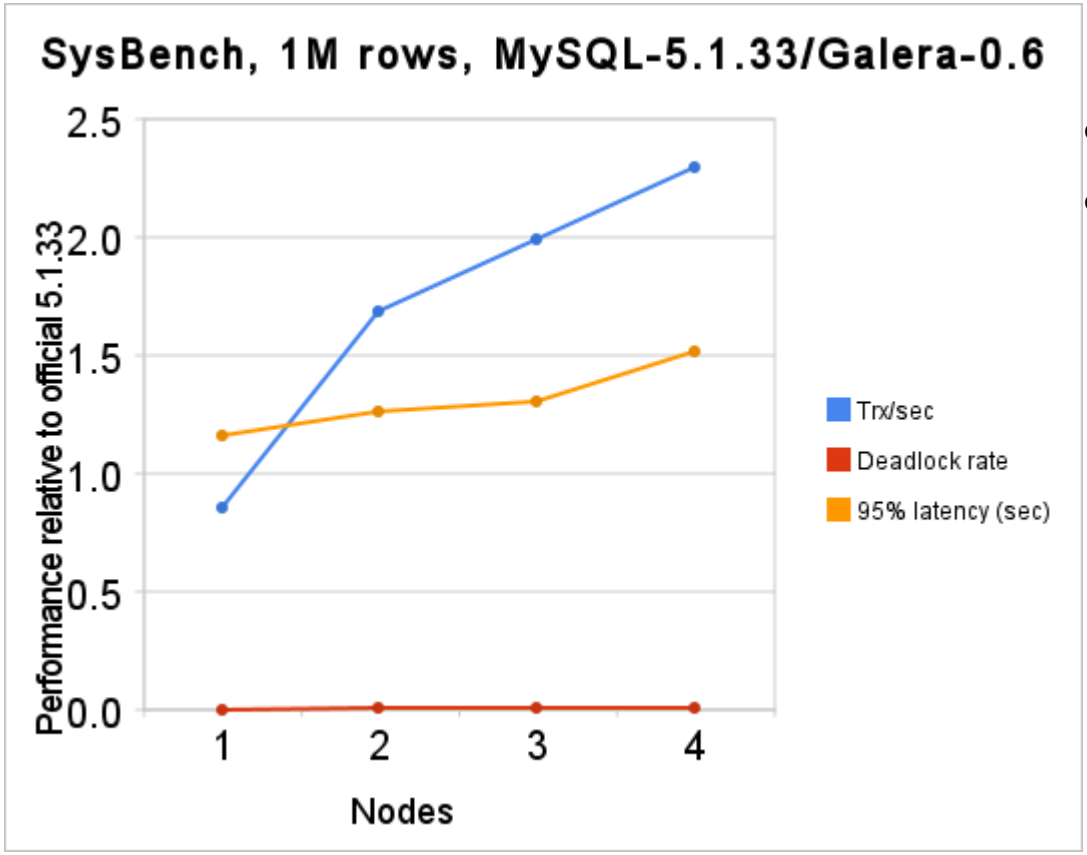
- If transaction modifies a lot of rows, the write set size can grow too large to handle
  - ➔ Escalate to page/table level locking
  - ➔ Escalate to SQL replication
- Long running transactions
  - ➔ Limit the number of write capable nodes

Benchmarking

Benchmarking

- Tested with several benchmarks
  - Sysbench, dbt2, DOTS, osdb, jmeter, sqlgen...
- Benchmarks testing with 'physical hardware' and with Amazon EC2 small and large instances
- Currently tests only up to 5 cluster nodes
- In general, shows good scalability even with write intensive work loads

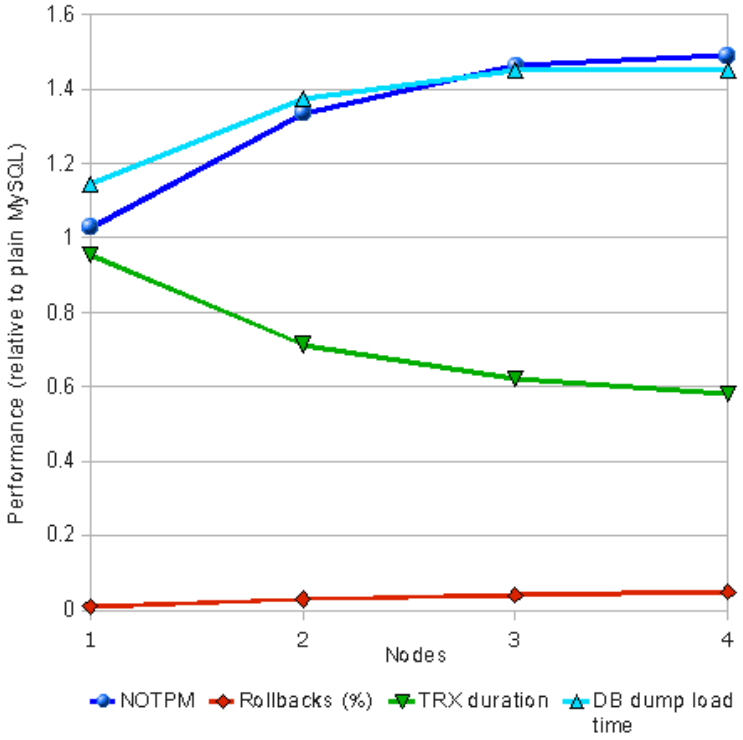
Sysbench Benchmarks



- Sysbench oltp mode test
- EC2 Large instances

nodes	users	trx/s	deadlks	95%lat
1	18	385	0	0.092
2	36	761	2.54	0.100
3	45	900	3.42	0.103
4	60	1034	4.54	0.120
official 5.1.33 binary:				
1	18	451	0	0.079

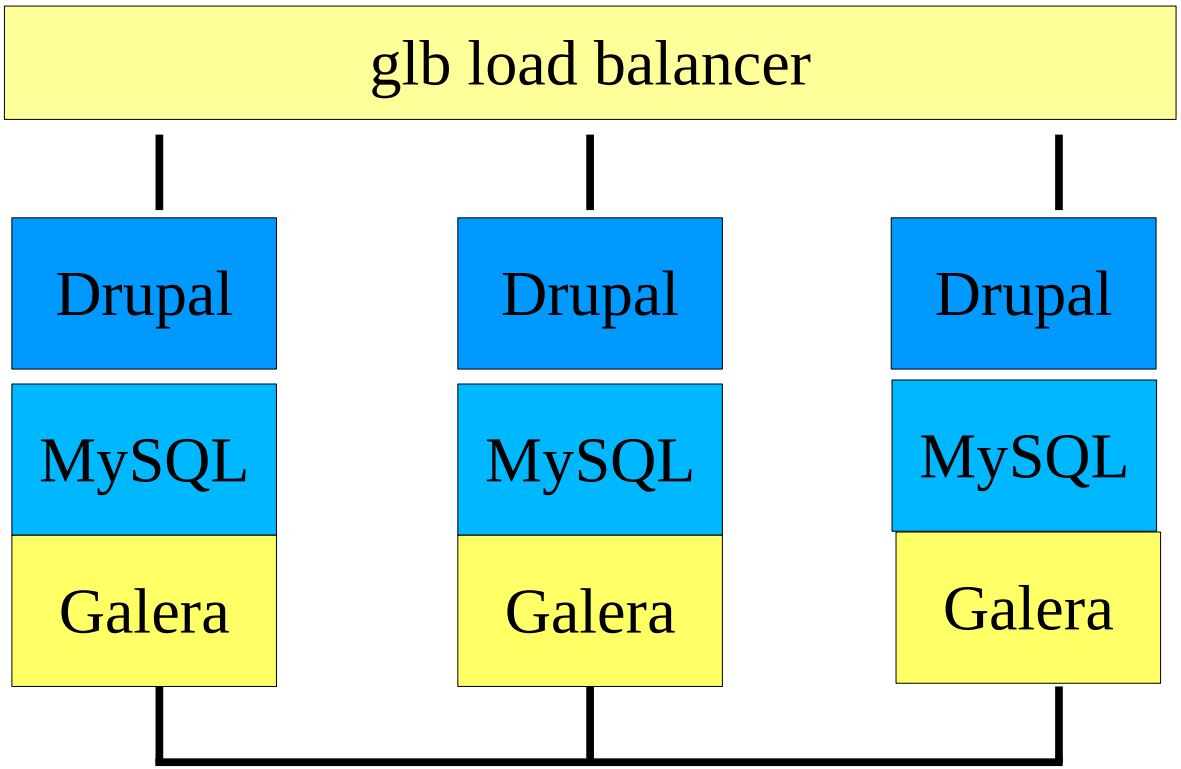
Dbt2 Benchmark



- EC2 large instances
- Dbt2 benchmark
- 60 warehouses

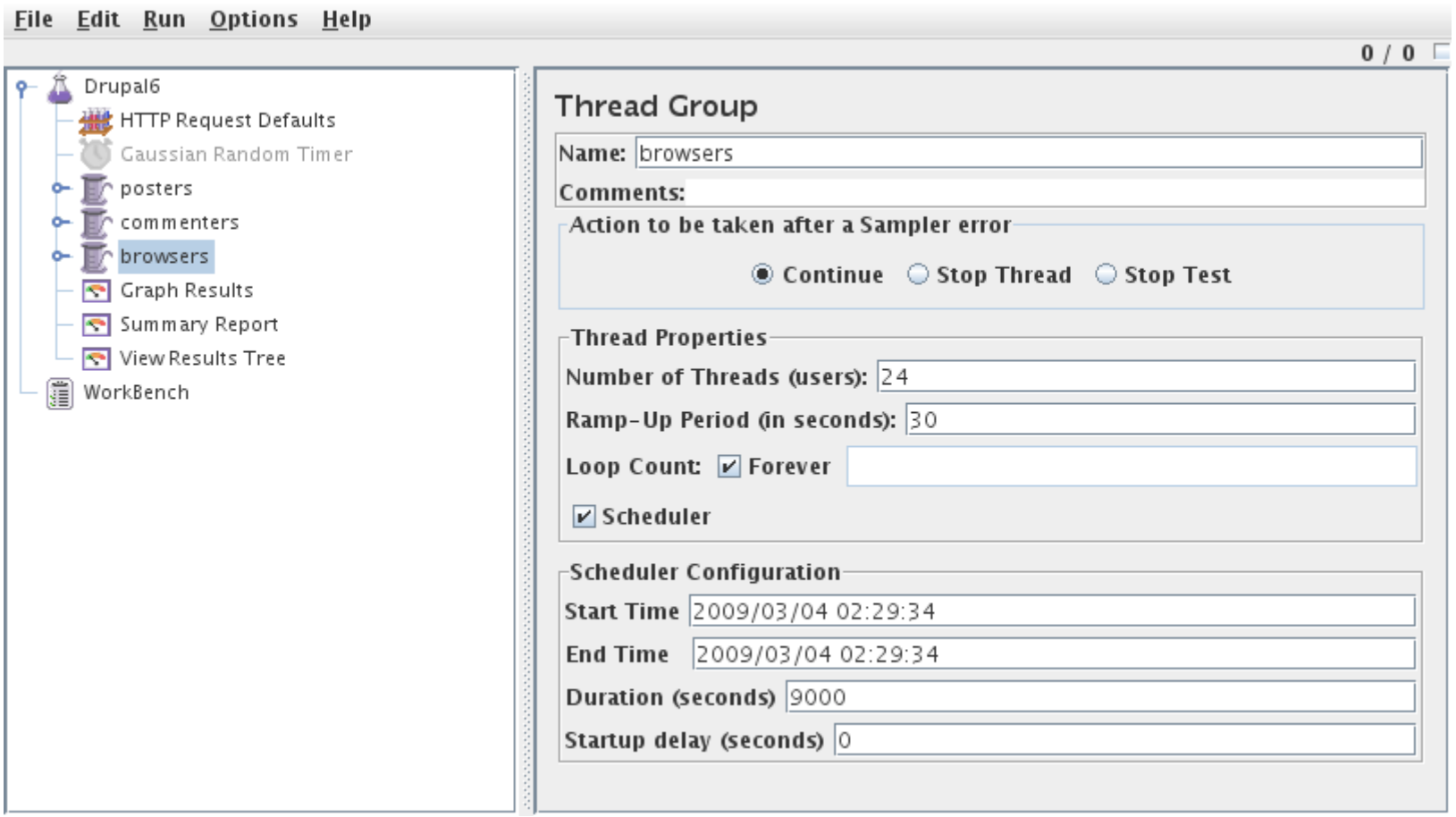
	Conns	NOTPM	Rollbacks(%)	TRX duration(sec)	Dump load(min)
Plain 5.1.30:	20	~7220	1	2.27	26
1 node :	12	~7420	1	2.17	30
2 nodes :	24	~9630	3	1.63	36
3 nodes :	36	~10555	4	1.41	38
4 nodes :	48	~10753	5	1.32	38

Drupal Scale-Out



- Proof of concept
- Each drupal node has local MySQL
- all nodes identical
- ~10% of CPU for MySQL
- glb load balancer

The Test


File Edit Run Options Help
0 / 0
Drupal6
HTTP Request Defaults
Gaussian Random Timer
posters
commenters
browsers
Graph Results
Summary Report
View Results Tree
WorkBench
Thread Group
Name: browsers
Comments:
Action to be taken after a Sampler error
Continue
Stop Thread
Stop Test
Thread Properties
Number of Threads (users): 24
Ramp-Up Period (in seconds): 30
Loop Count: Forever
Scheduler
Scheduler Configuration
Start Time 2009/03/04 02:29:34
End Time 2009/03/04 02:29:34
Duration (seconds) 9000
Startup delay (seconds) 0

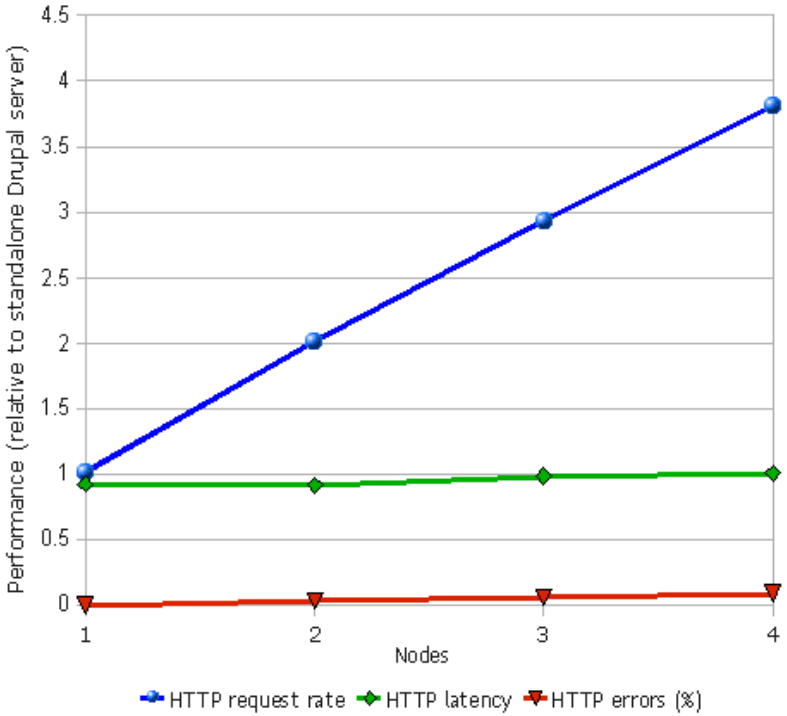
The Test

- Jmeter thread groups:
  - Posters write new pages
  - Commenters read pages and add comments
  - Browsers read pages and comments
- Threads created in proportion: 4/12/24
- Write intensive work load

Issues

- Drupal Cache must be off
- 'files' directory contents
  - ➔ use network file system
  - ➔ store all files in DB
- Concurrent login failures
- Autoinc insert bug: http://drupal.org/node/282555
  - Sorted out by a workaround to retry failed autoinc insert

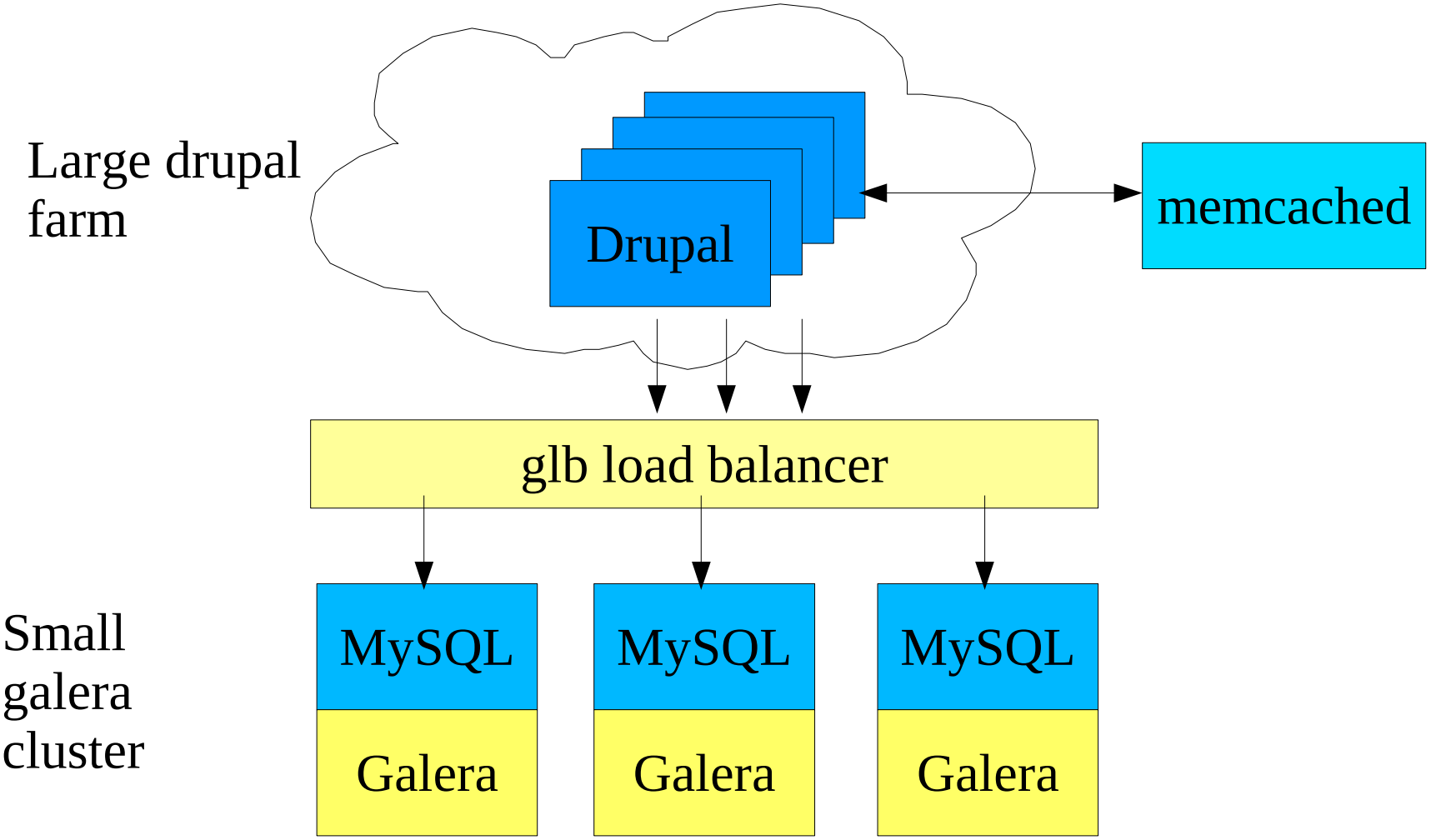
Results



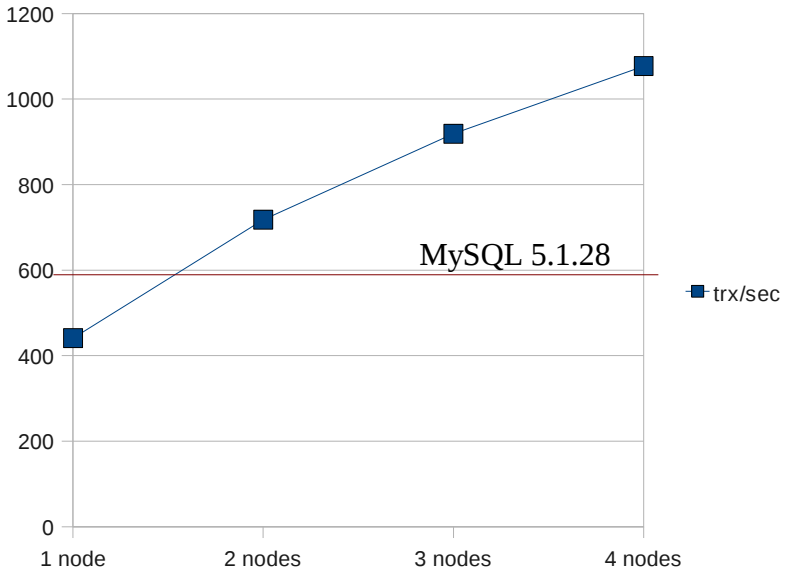
Nodes	Users	Throughput (req/min)	Latency (ms, median)	Latency (ms, average)	Errors (%)
1	180	724	1203	1827	0.00
2	360	1436	1190	1829	0.03
3	540	2091	1280	2150	0.06
4	720	2717	1214	2330	0.12

- http://www.codership.com/content/scaling-drupal-stack-galera-part-2-mystery-failed-login

The Way to Do It



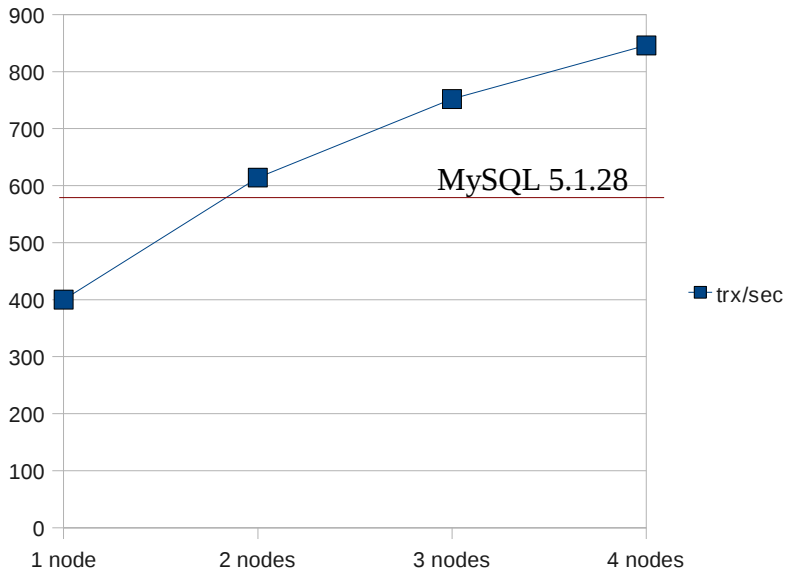
Sqlgen r/w: 50/50



50 tables / 1000 rows

	R/W	SELECTs	UPDATEs	TRANSACT	abort%	CONNECTIONS
plain	1.0	1458.06	1461.39	583.75	0.0	38.98
1 node	1.0	1104.51	1104.43	441.24	0.0	29.52
2 nodes	1.0	1796.16	1796.39	718.60	0.0	47.90
3 nodes	1.0	2297.58	2298.43	919.54	0.1	61.25
4 nodes	1.0	2698.50	2696.27	1077.75	0.1	71.95

Sqlgen r/w: 0/100



	R/W	SELECTs	UPDATEs	TRANSACT	abort%	CONNECTIONS
plain	0.0	0.00	2841.50	568.09	0.0	37.94
1 node	0.0	0.00	2002.69	400.27	0.0	26.64
2 nodes	0.0	0.00	3077.22	614.61	0.1	41.08
3 nodes	0.0	0.00	3773.47	752.58	0.3	50.14
4 nodes	0.0	0.00	4253.30	846.76	0.5	56.51

Project Status

State of the Code

- Public releases since Jan 2009
- Current release 0.6.2
  - Good for benchmarking & evaluating
  - Lacks “node join” feature
  - Only vsbes group communication
- Planned releases 0.7, 0.8, 09 and 1.0
- Release 0.7 is fully open source and is targeted for Sep 2009

Roadmap

- Release 0.7 - Stability Milestone
  - Node join capability
  - Open Source
  - Fault tolerant GCS
- Release 0.8 – Optimization Milestone
  - Incremental State Transfer
  - Xtrabackup, LVM
- Release 0.9 - Security Milestone
  - TLS tunneling
- Release 1.0 - Management Milestone
  - Management tools

Reaching Out

- wsrep integration code should be maintained by DBMS provider
- MariaDB integration to happen in near future
- Drizzle replication framework is shaping up
- PostgreSQL support
  - We need a partner for this
  - Postgres-R by Markus Wanner
- Any other transactional engine?
  - Check out wsrep API
  - Snapshot isolation
  - Prioritized transactions

Summary

- Certification based replication turns out effective
  - High Availability
  - Transparency
  - Good scalability even with high write rates
- wsrep API is “not too hard” to implement
- Any (transactional) DBMS can leverage this replication possibility

Get in Touch!

codership

- Downloads available: http://www.codership.com
- R&D consulting services
- Evaluation support
- info@codership.com