State of the Art for MySQL Multi-Master Replication

Robert Hodges, CEO

A Short and Depressing Introduction to Distributed Systems

Why Do We Care About Multi-Master?

The Dream: Multiple, active DBMS servers with exactly the same data

1. Simple high availability model
2. Operate systems over multiple sites
3. Access geographically “close” data
4. Enable communication between applications

There's Just One Problem...

It Doesn't Work

More Precisely: Multi-Master Systems Cannot Provide One Copy Serialization

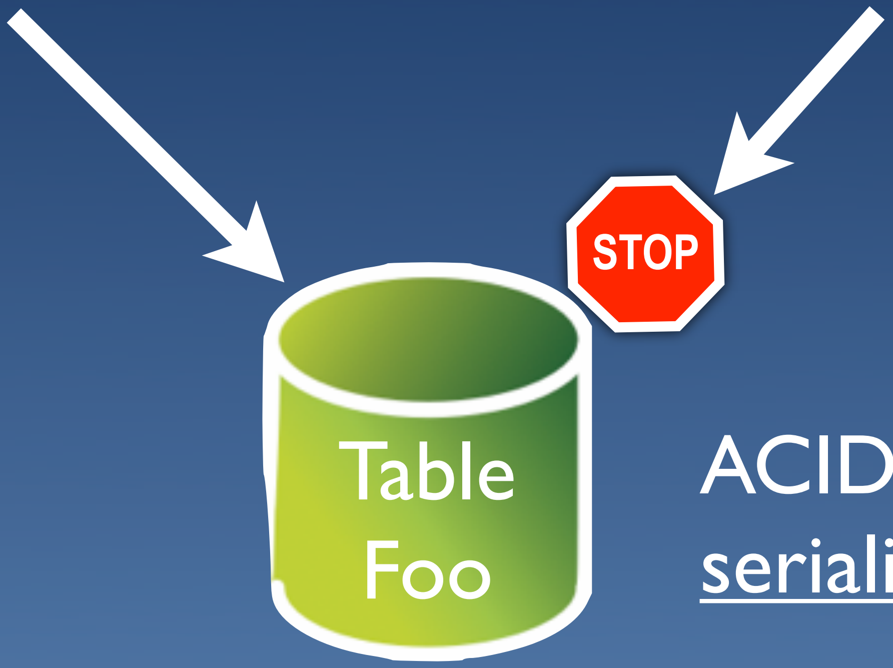
Consistency In a Single DBMS

Transaction #1

```
update foo
set follower=23
where id=32
```

Transaction #2

```
update foo
set follower=976
where id=32
```



ACID transactions serialize updates

```
follower=976
```

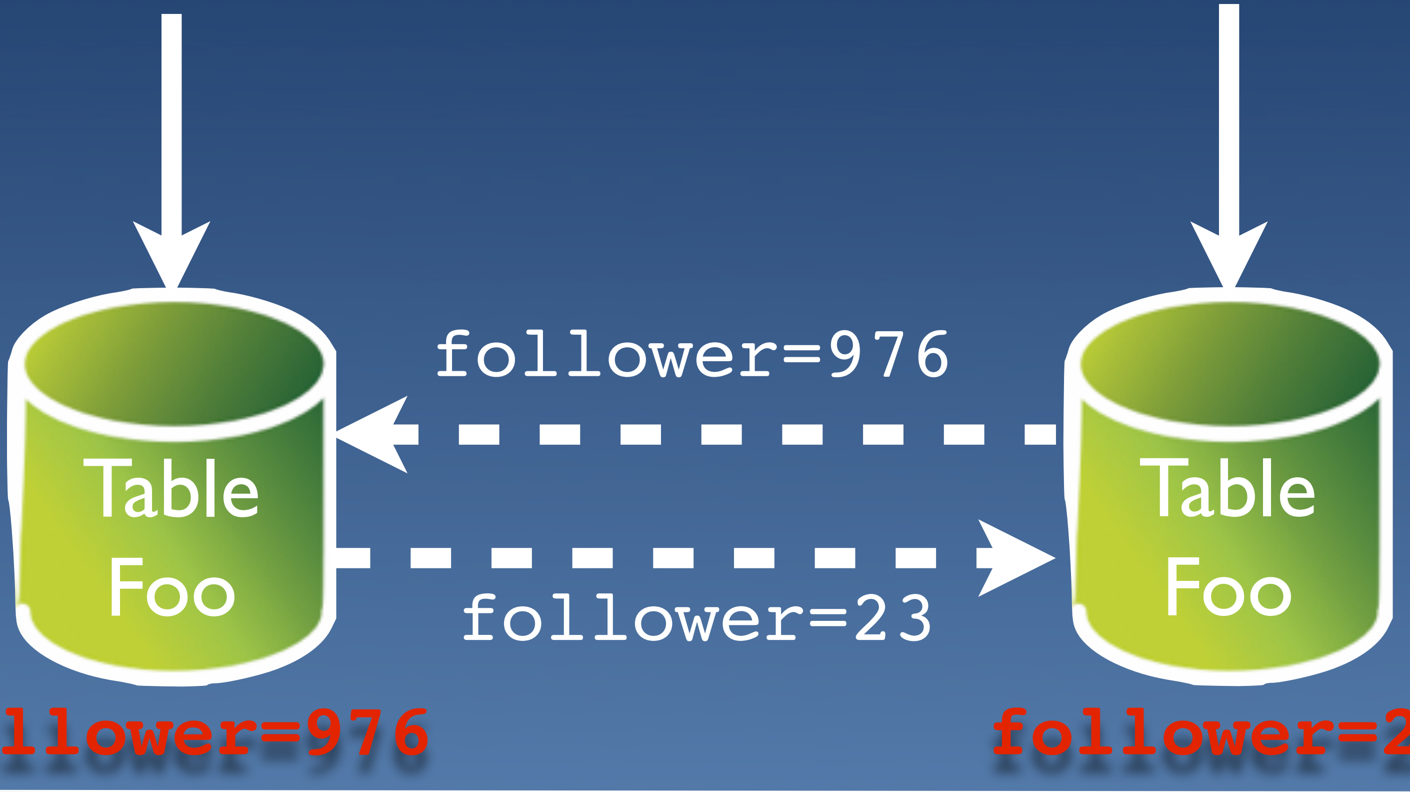
Consistency in a Distributed DBMS

Transaction #1

```
update foo
set follower=23
where id=32
```

Transaction #2

```
update foo
set follower=976
where id=32
```



follower=976

follower=23

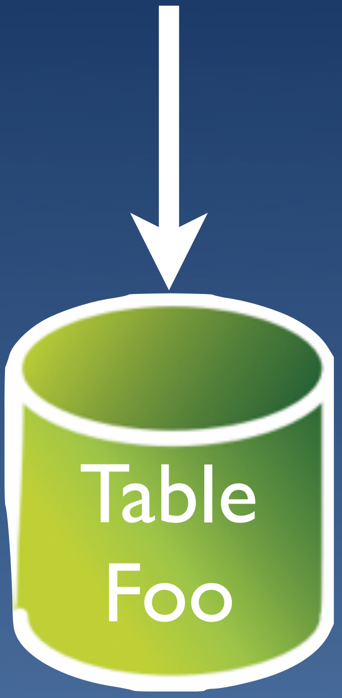
Ensuring Distributed Consistency

Transaction #1

`follower=23`

Transaction #2

`follower=976`



Pessimistic Locking

(Wait your turn, pal!)

Optimistic Locking

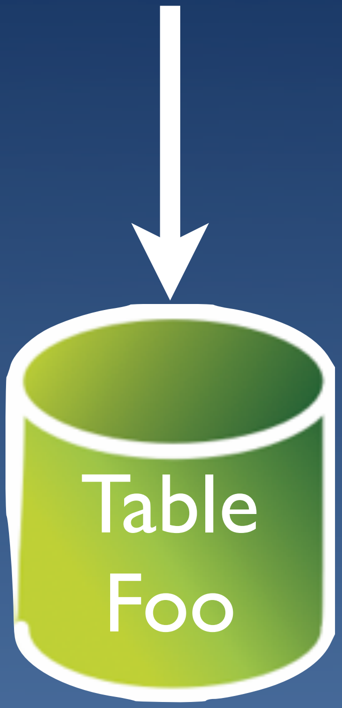
(Early bird gets the worm)

Conflict Resolution

(Your mother cleans up later)

Conflict Avoidance

(Solve the problem by not having it)



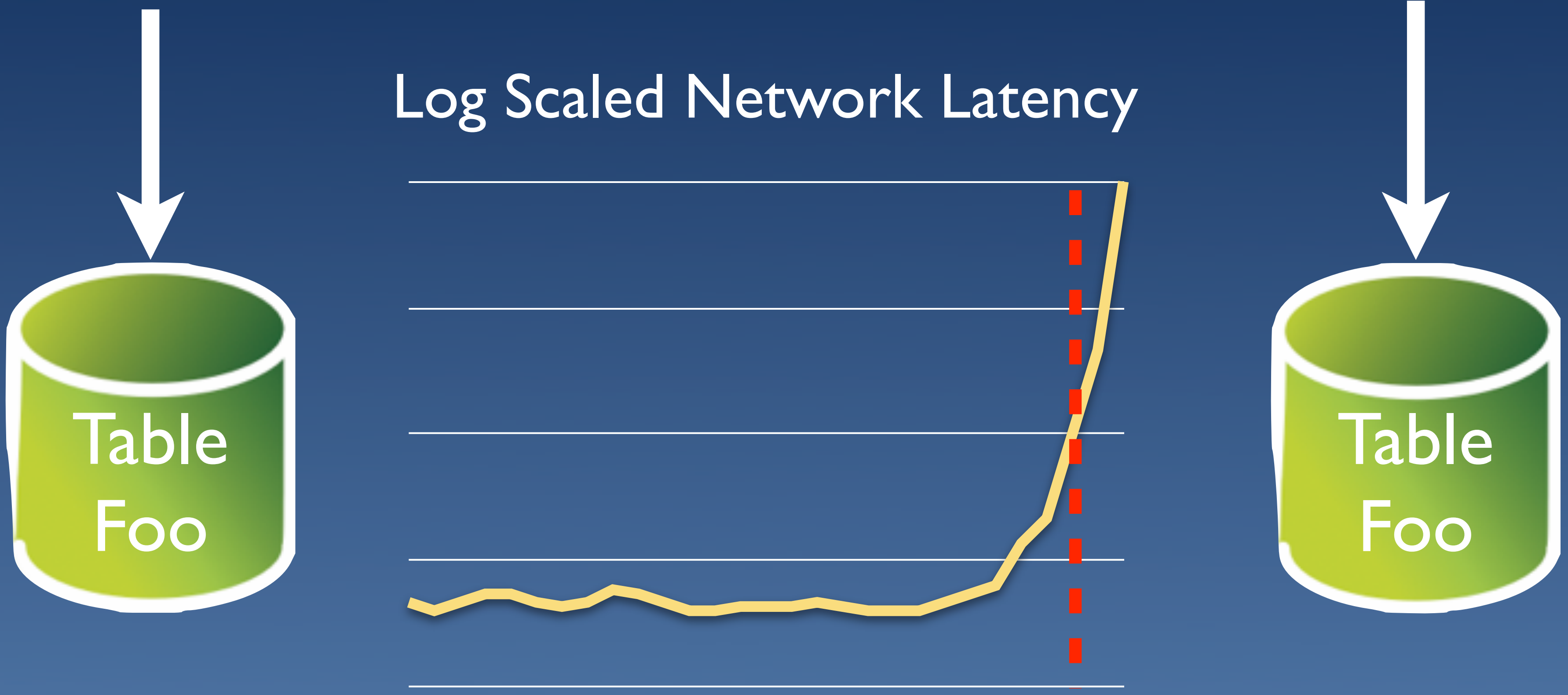
Communication Implies Latency

Transaction #1

`follower=23`

Transaction #2

`follower=976`

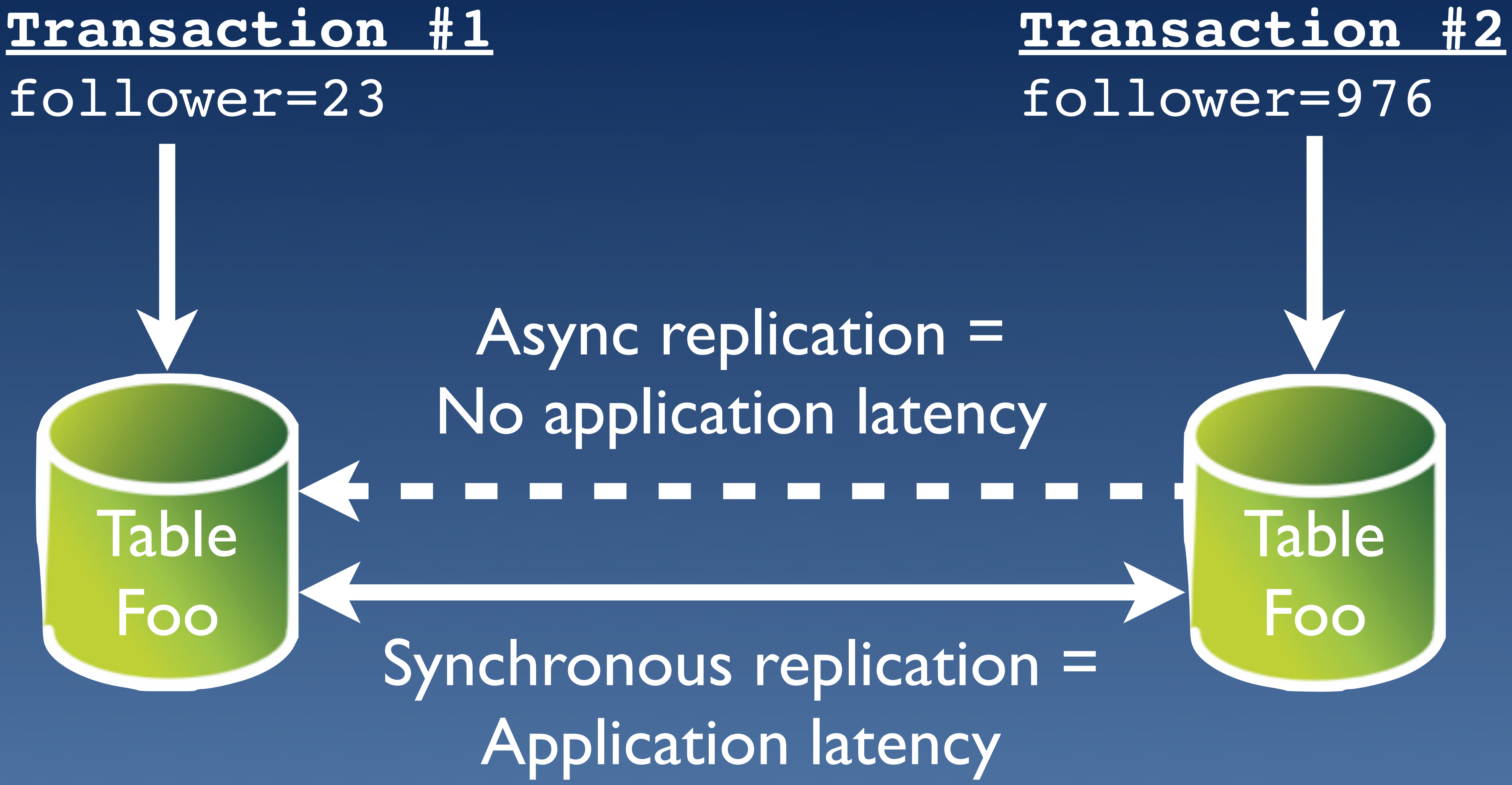


Log Scaled Network Latency

Table Foo

Table Foo

Communication Implies Latency



Transaction #1

`follower=23`

Transaction #2

`follower=976`

Async replication =
No application latency

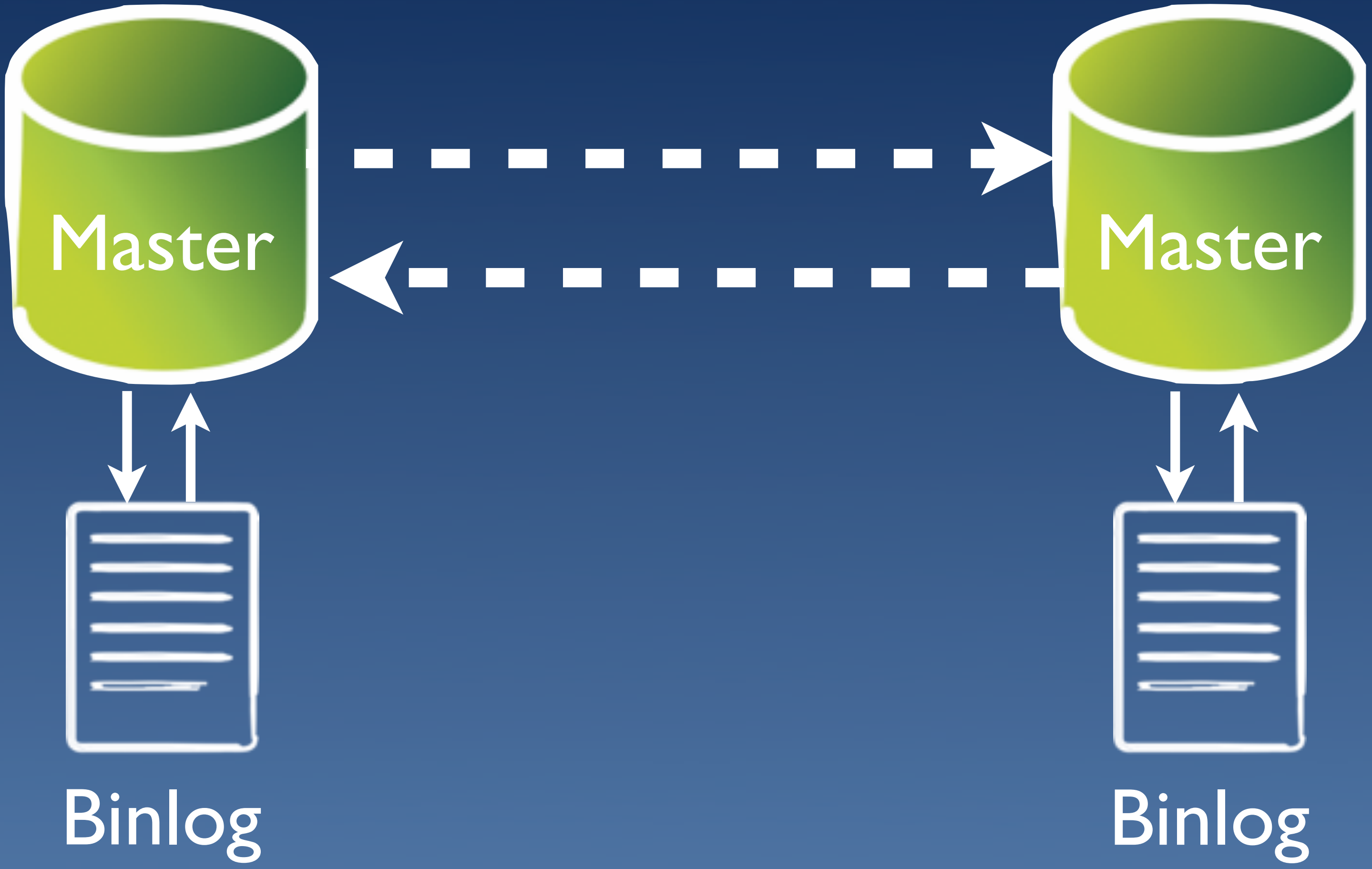
Synchronous replication =
Application latency

So Can We Build Useful Applications?

Absolutely.

MySQL Native Replication (Aka the basics)

How Does It Work?



Row vs. Statement Replication

- Statement replication = send client SQL
- Row replication = send changed rows
- Use row replication for multi-master

Server IDs

- Distinguish between different MySQL servers
- Prevent replication loops in multi-master

```
[my.cnf]
server-id=1
...

[my.cnf]
server-id=2
...

[my.cnf]
server-id=3
...
```

Auto-Increment Key Offsets

- Set different keys on each server to avoid primary key collisions

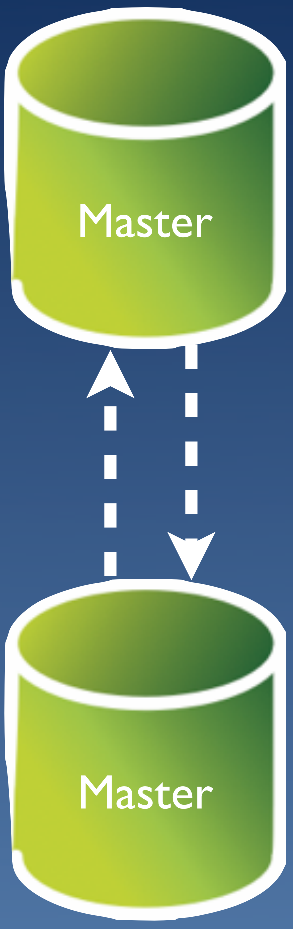
```
[my.cnf]
server-id=1
auto-increment-offset = 1
auto-increment-increment = 4
...

[my.cnf]
server-id=2
auto-increment-offset = 2
auto-increment-increment = 4
...
```

Global Transaction IDs (GTID)

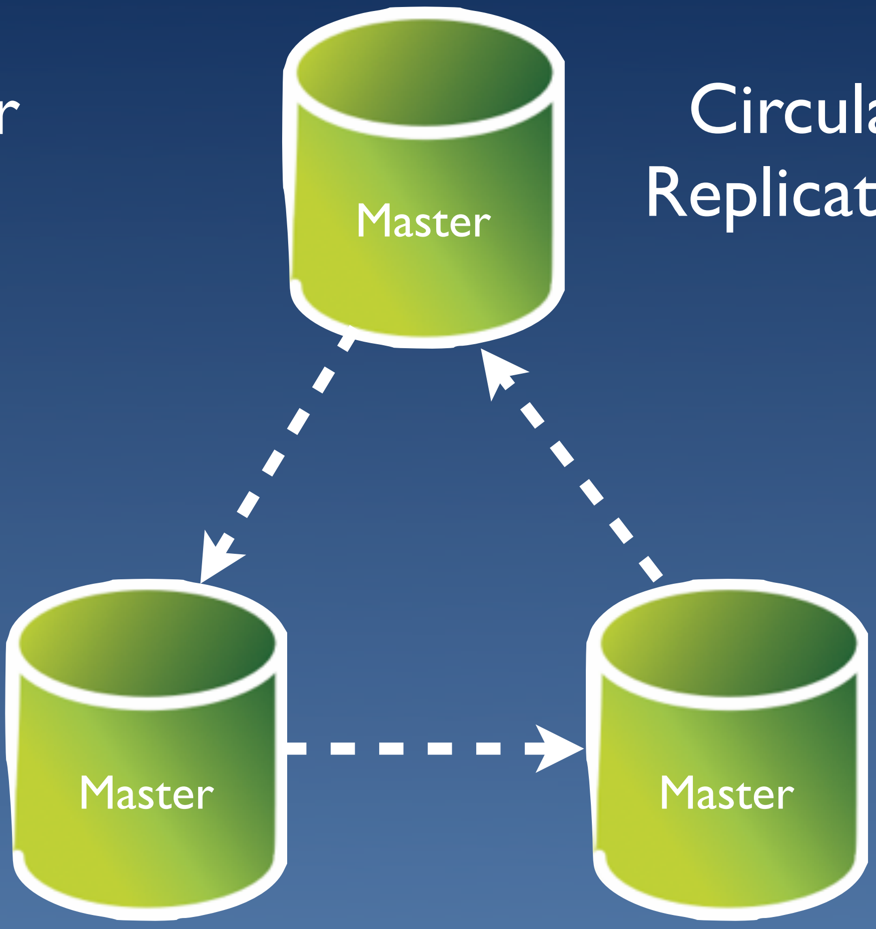
- New feature in MySQL 5.6 to provide globally unique IDs for transactions
- Consist of UUID plus sequence number
- Designed to capture location of original update as well as the sequence number of the transaction

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:5660
```

MySQL Multi-Master Topologies



Master-master Replication



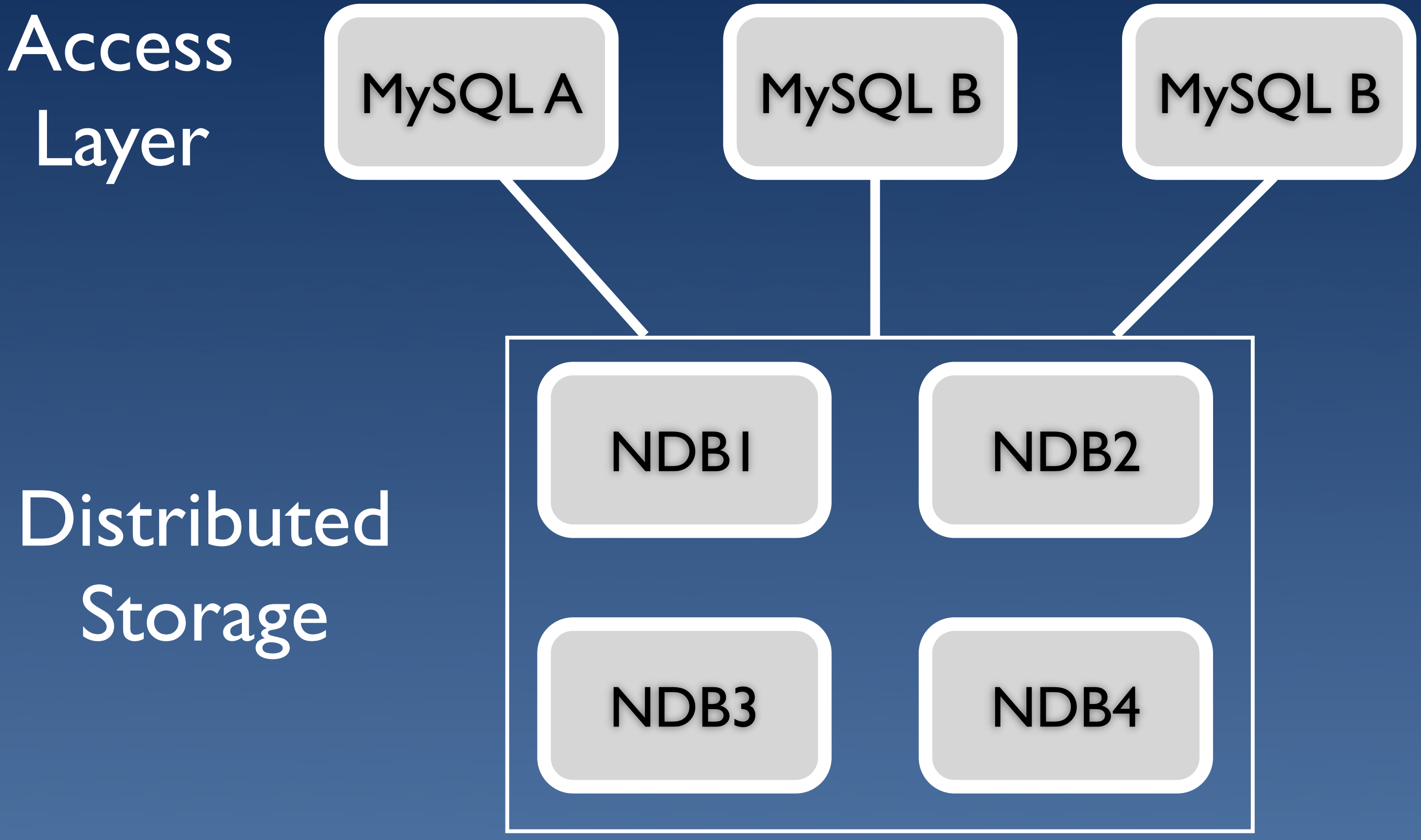
Circular Replication

Native Replication Summary

- Built in with well-known capabilities
- Very limited topology support
- Very limited conflict avoidance
- Not a good choice for multi-master if there are writes to more than 1 master
- GTIDs, multi-source (MariaDB) replication promise for the future

MySQL Cluster

How Does It Work?



Access Layer

MySQL A

MySQL B

MySQL B

Distributed Storage

NDB1

NDB2

NDB3

NDB4

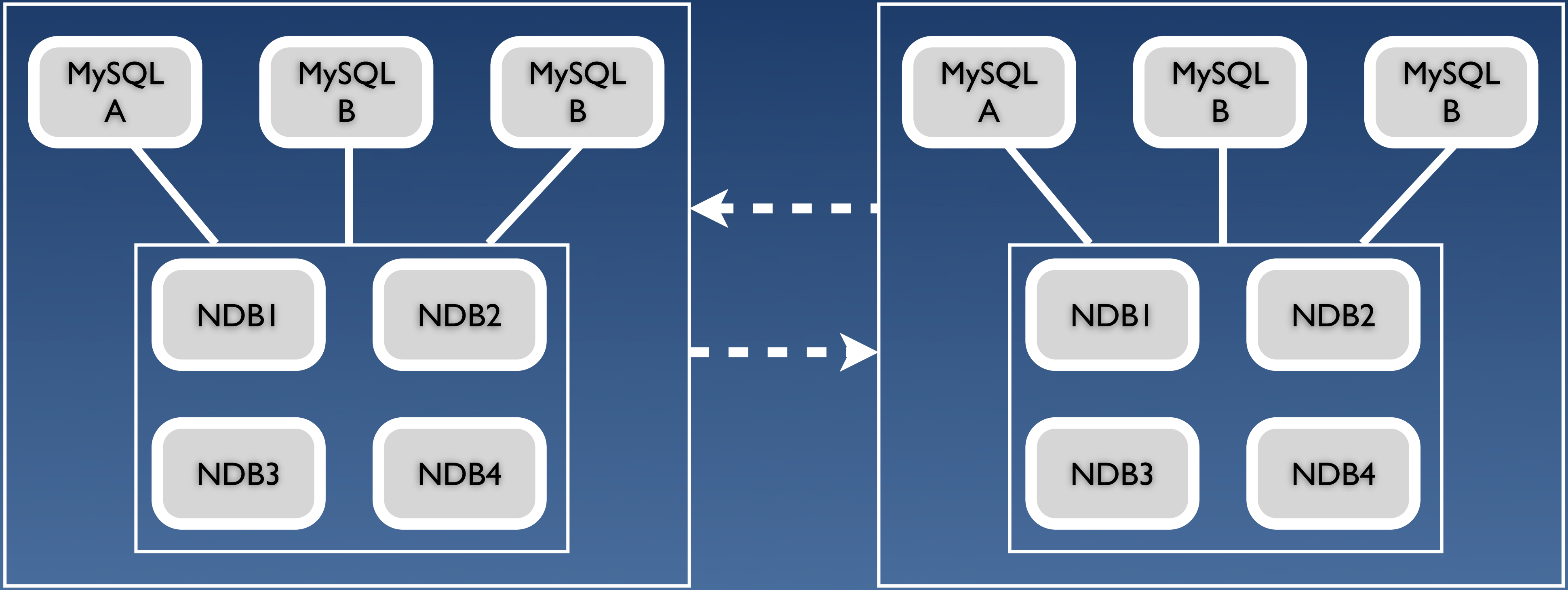
MySQL Cluster Cross-Site Topology

Transaction #1

`follower=23`

Transaction #2

`follower=976`



Primary Replica

Backup Replica

Eventual Consistency Algoritm

- NDB has built-in cross-cluster conflict detection based on epochs and primary keys
- Updates to primary always succeed
- Update to backup may be rolled back if primary has a conflicting update
- MySQL Cluster resends updates from the primary to “cure” conflicts on the slave

Caveat: I am not a MySQL Cluster expert

MySQL Cluster Summary

- Allows active/active operation
- Innovative eventual consistency algorithm
- Covers failure of individual MySQL nodes
- Detects conflicts automatically on rows
- Limited by lack of NDB usage in general

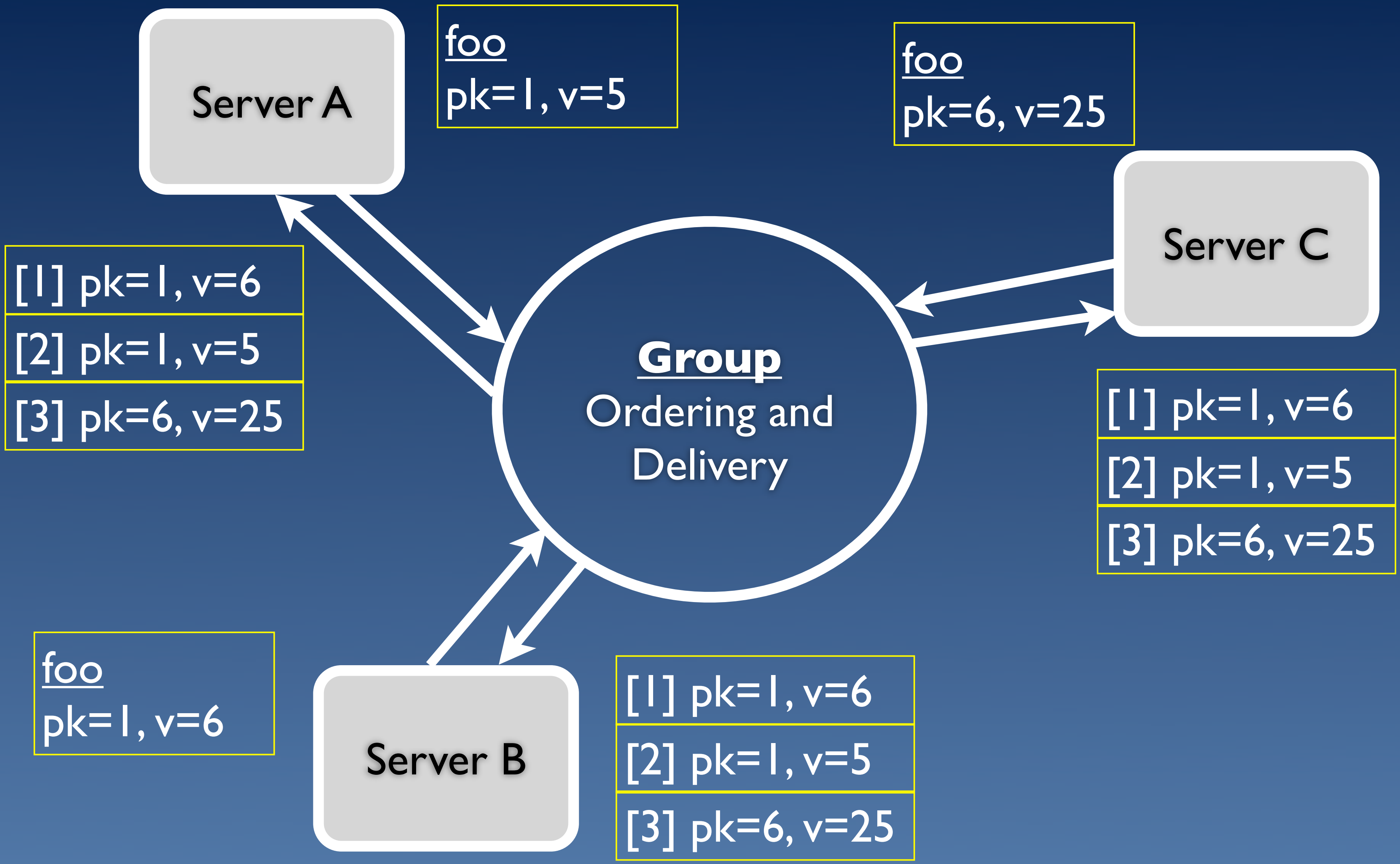
Check with MySQL Cluster folks for more

See also Brendan Frazer's posts at

http://messagepassing.blogspot.com

Galera

How It Works: Group Communication



Server A

foo
pk=1, v=5

foo
pk=6, v=25

Server C

[1] pk=1, v=6
[2] pk=1, v=5
[3] pk=6, v=25

Group
Ordering and
Delivery

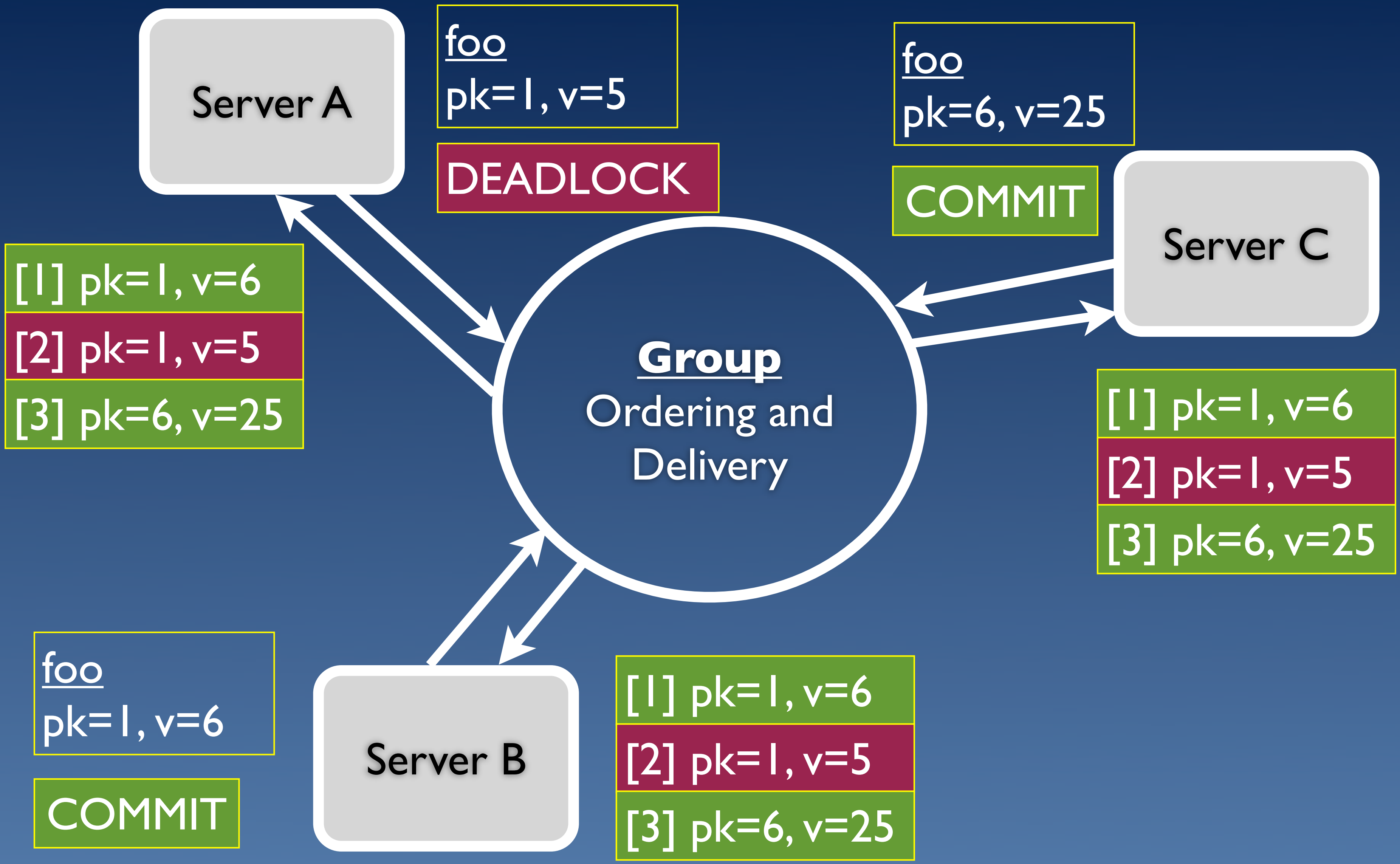
[1] pk=1, v=6
[2] pk=1, v=5
[3] pk=6, v=25

foo
pk=1, v=6

Server B

[1] pk=1, v=6
[2] pk=1, v=5
[3] pk=6, v=25

How It Works: Certification



Server A

foo
pk=1, v=5

DEADLOCK

[1] pk=1, v=6
[2] pk=1, v=5
[3] pk=6, v=25

Group
Ordering and
Delivery

Server C

foo
pk=6, v=25

COMMIT

[1] pk=1, v=6
[2] pk=1, v=5
[3] pk=6, v=25

Server B

foo
pk=1, v=6

COMMIT

[1] pk=1, v=6
[2] pk=1, v=5
[3] pk=6, v=25

New Node Start-Up and SST

```
(Initializing a new node)

# vi /etc/my.cnf  <== set node name
# mysql_install_db --user=mysql --datadir=/data/
galera
# mysqld_safe &
```

1. Assess node state
2. Join the cluster
3. Request SST (= “State Snapshot Transfer”)
4. Recover DBMS

Connect to Any Node for Writes

```
(galera1)
mysql> create table test
(id int primary key
auto_increment, data
varchar(30));

mysql> insert into
test(data) values('g1');

mysql> select * from test;
+----+-------+
| id | data  |
+----+-------+
|  3 | g2    |
|  4 | g1    |
+----+-------+
```

Auto_increment keys handled by Galera

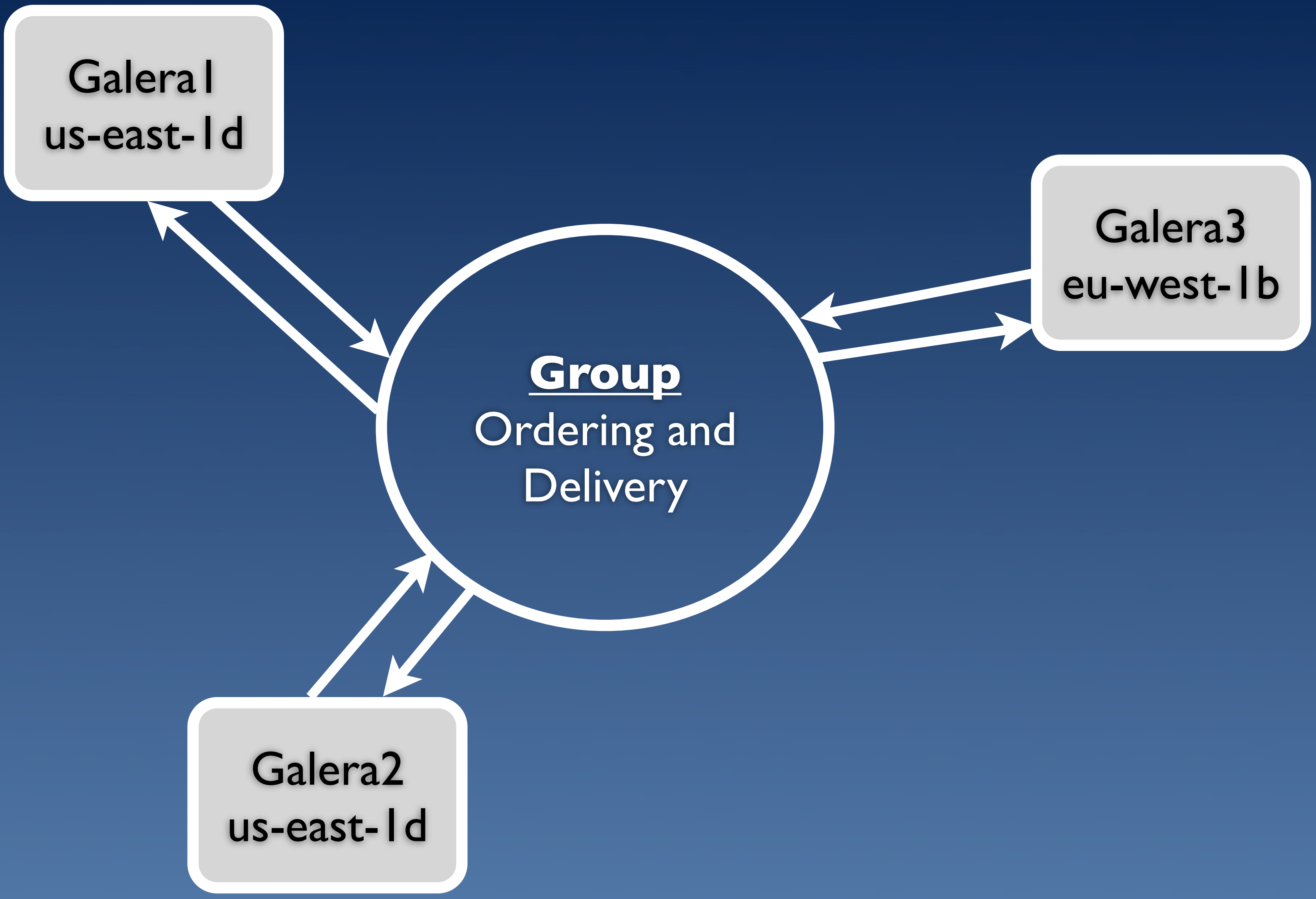
```
(galera2)

mysql> insert into
test(data) values('g2');

mysql> select * from test;
+----+-------+
| id | data  |
+----+-------+
|  3 | g2    |
|  4 | g1    |
+----+-------+
```

Optimistic Locking in Action

```
(galera1)
mysql> begin;

mysql> update test set
data='from g1' where id=3;

mysql> commit;

mysql> select * from test;
+----+----------+
| id | data     |
+----+----------+
|  3 | from g1  |
|  4 | g1       |
+----+----------+
```

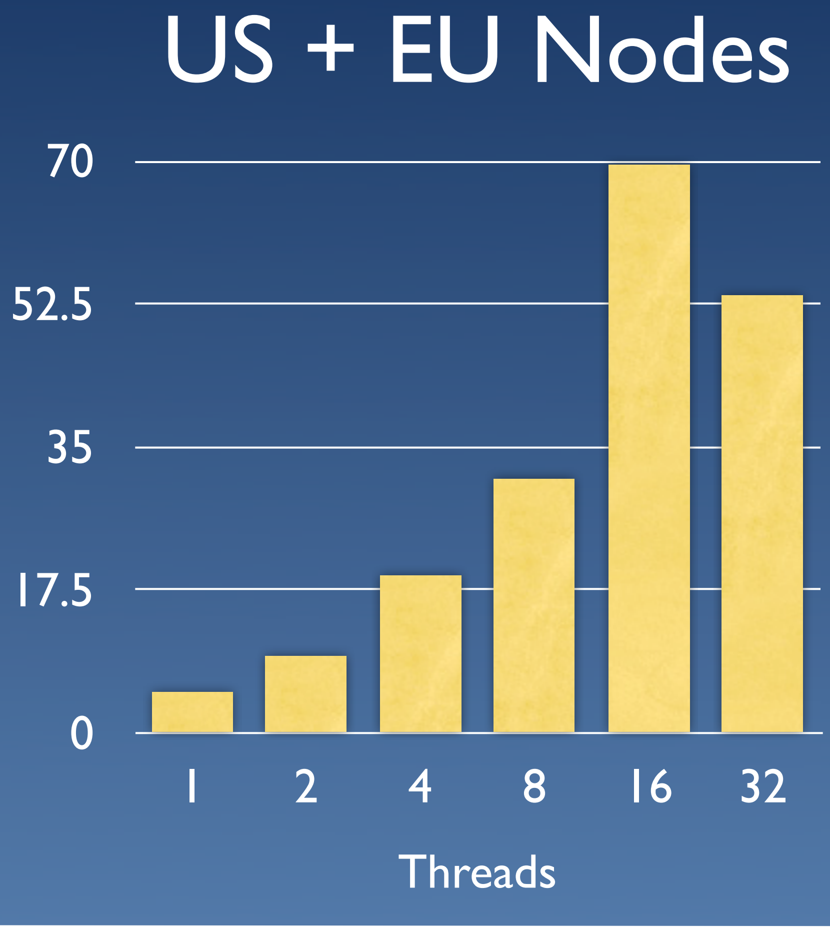
```
(galera2)
mysql> begin;

mysql> update test set
data='from g2' where id=3;

mysql> commit;
ERROR 1213 (40001): Deadlock
found when trying to get lock;
try restarting transaction

mysql> select * from test;
+----+----------+
| id | data     |
+----+----------+
|  3 | from g1  |
|  4 | g1       |
+----+----------+
```

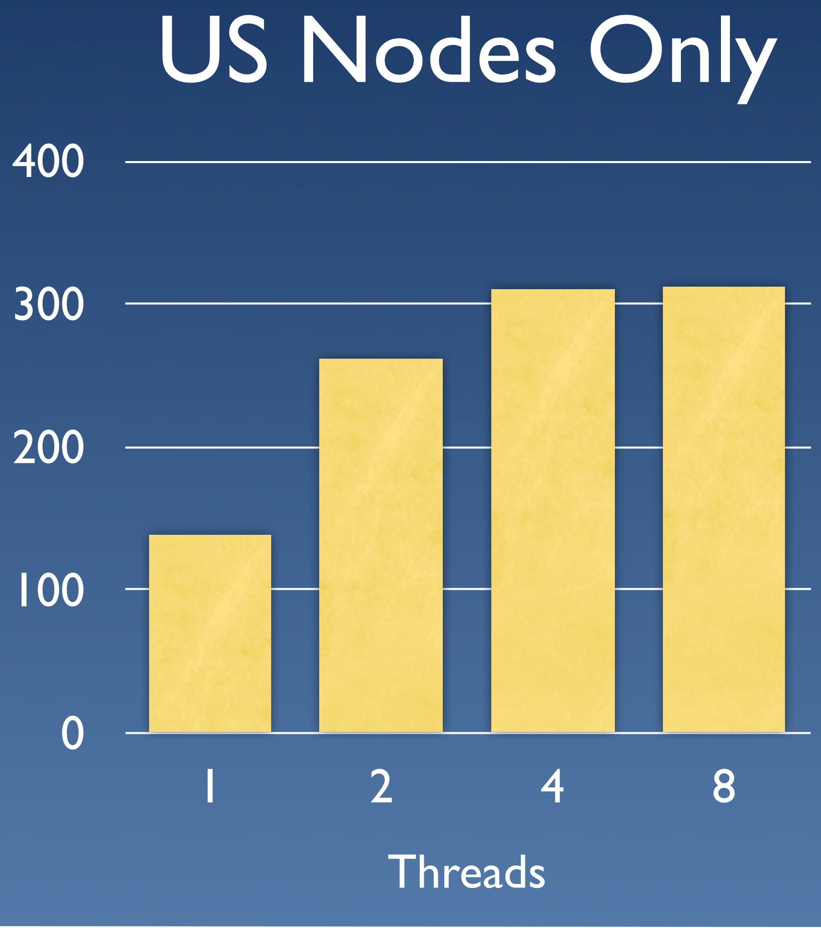
Cross-Site Replication



Galera Replication Performance

Sysbench Transactions per Second





Failure Handling

- Crashed servers drop out of the cluster
- IST (= incremental state transfer) can repair nodes that are just out of date
- Quorum determined automatically; On loss of quorum all nodes stop
- Loss of quorum can stop entire sites from working if you operate cross-site

Fixing Broken Nodes

- IST (= incremental state transfer) can repair nodes that are just out of date
- Inconsistent nodes require full state transfer

```
130422 16:28:09 [ERROR] WSREP: Local state seqno
(88056) is greater than group seqno (88054): states
diverged. Aborting to avoid potential data loss. Remove '/
data/galera//grastate.dat' file and restart if you wish to
continue. (FATAL)
```

- SST is time-consuming for large data sets
- Enabling binlogs may help determine current state

Replicating To/From Galera Clusters

- Enable binlogs on all clusters & restart

```
# Same server ID on all nodes
server-id=13
log-slave-updates=1
```

- Try to connect with Tungsten!

```
130422 19:21:07 [ERROR] Slave SQL: Could not
execute Update_rows event on table
tungsten_g1.heartbeat; Can't find record in
'heartbeat', Error_code: 1032; handler error
HA_ERR_KEY_NOT_FOUND; the event's master log
FIRST, end_log_pos 153, Error_code: 1032
```

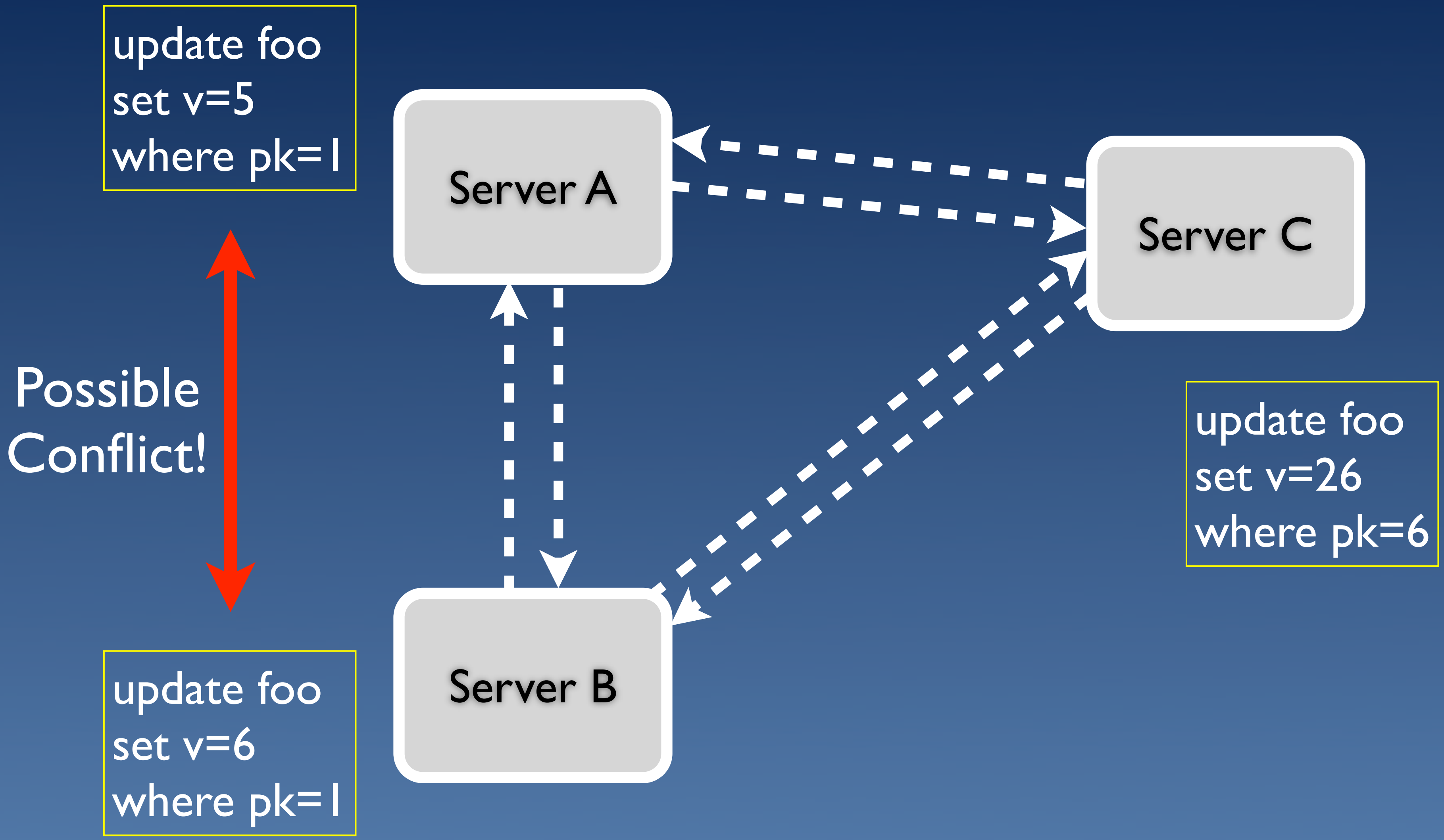
A new Galera bug is born :(

Galera Summary

- Big plus #1: simplicity
- Big plus #2: synchronous replication
- Performance looks promising
- Optimistic locking not good for all workloads
- SST is problematic for large datasets
- Unstable for some use cases but improving fast

Tungsten

How It Works: Eventual Consistency



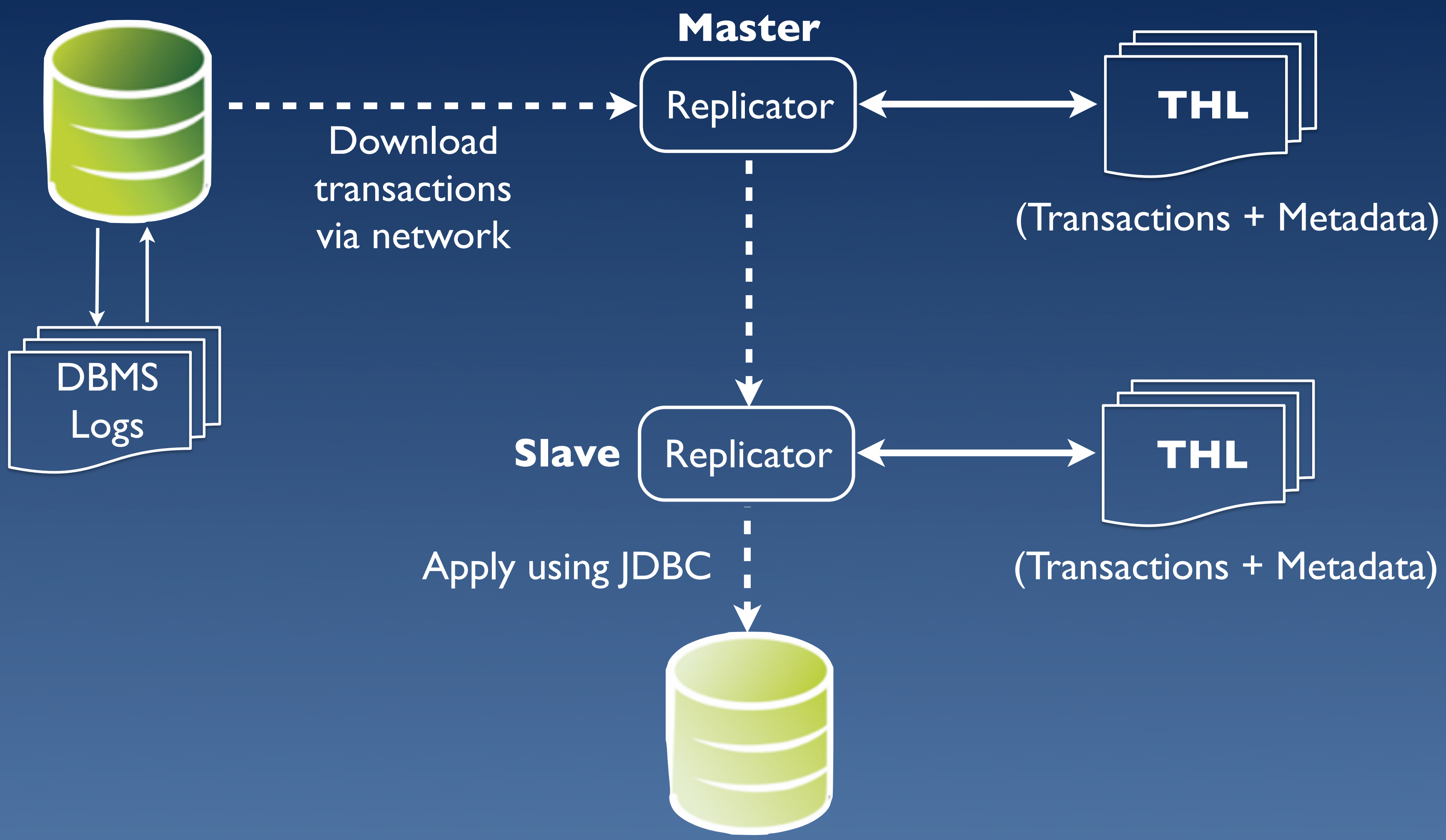
update foo
set v=5
where pk=1

Possible Conflict!

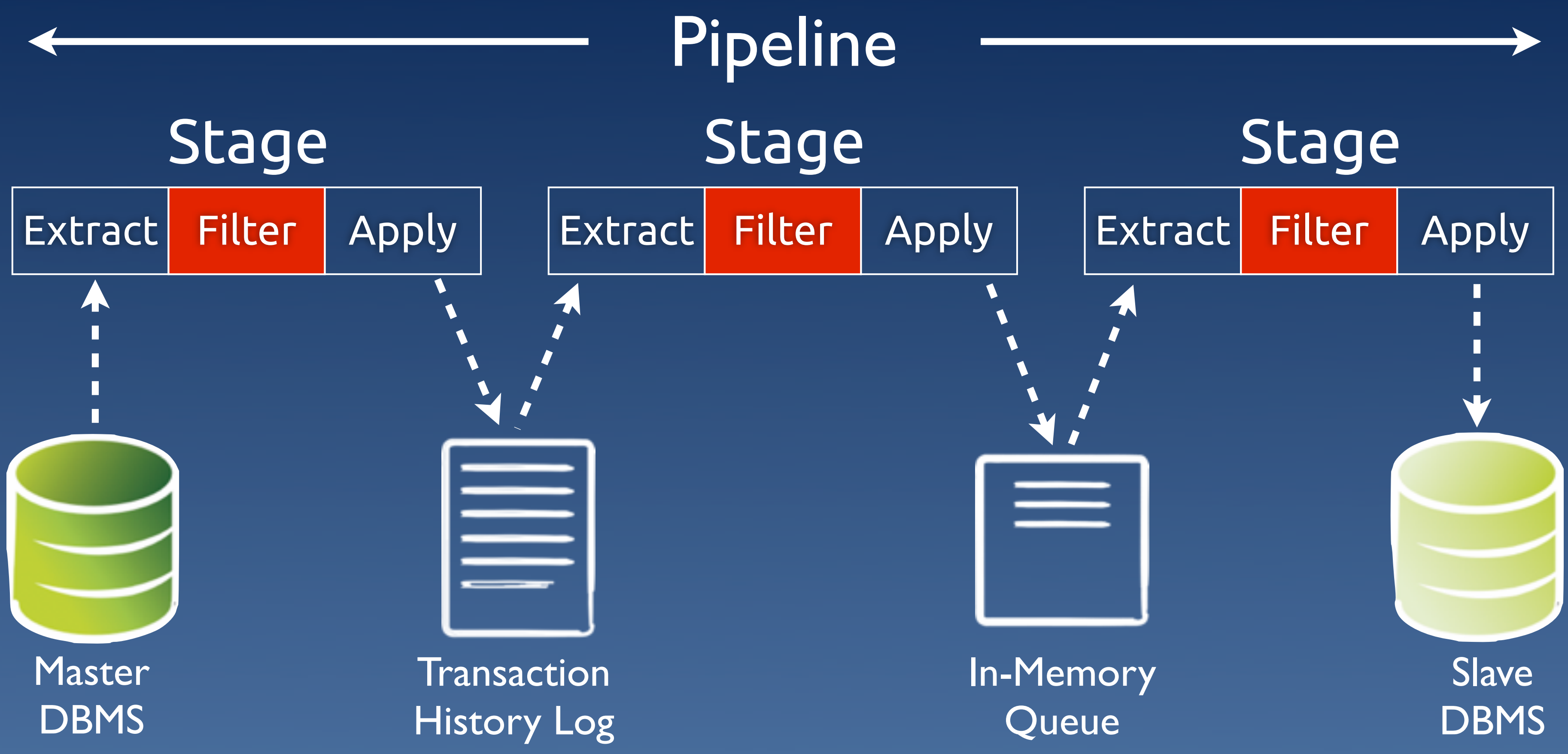
update foo
set v=6
where pk=1

update foo
set v=26
where pk=6

Tungsten Replicator Overview


Master
Replicator
THL
Download transactions via network
(Transactions + Metadata)
DBMS Logs
Slave
Replicator
THL
Apply using JDBC
(Transactions + Metadata)

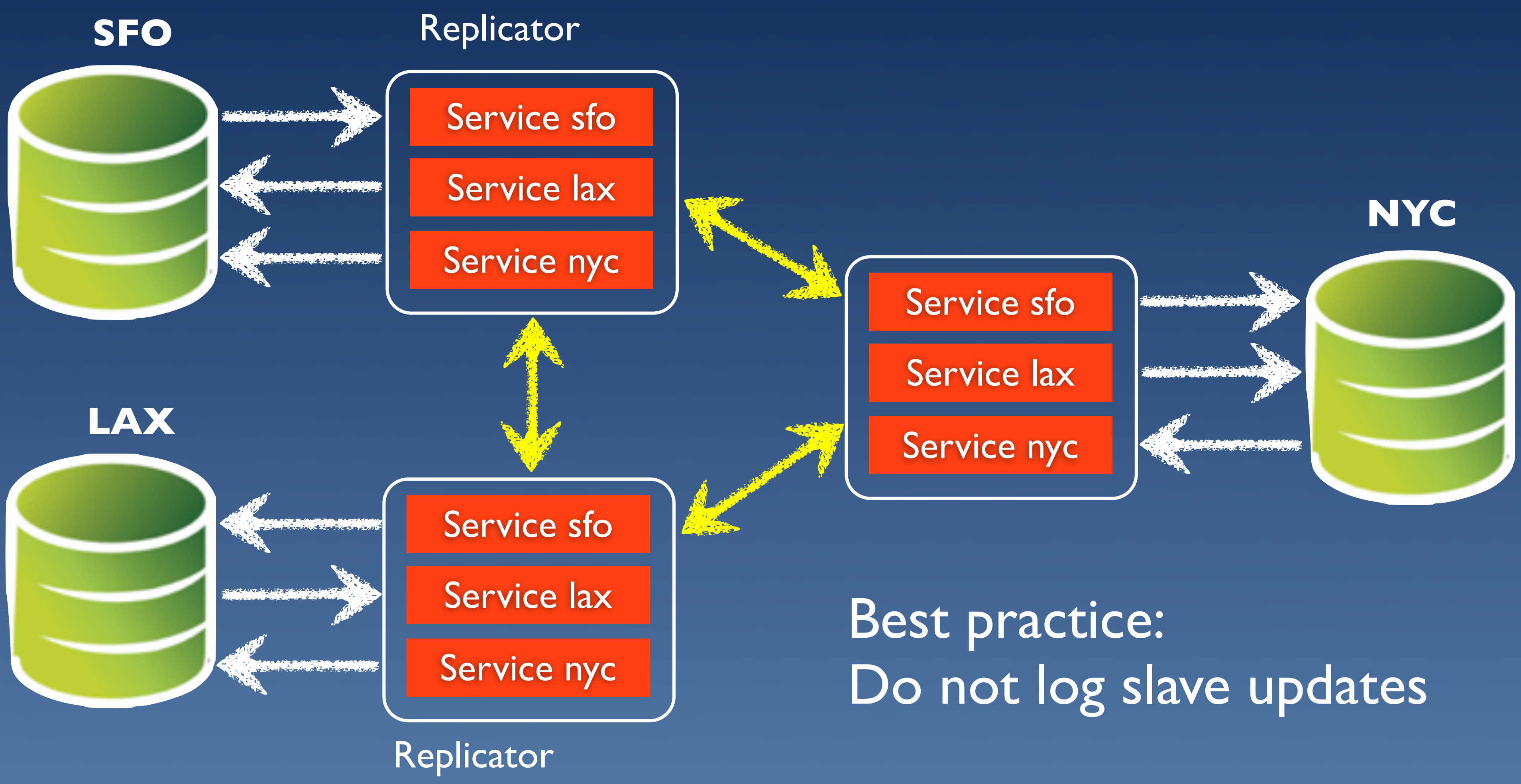
Tungsten Replication Service



Eventually Consistent Design

- MySQL servers are completely independent
- Replication provides multi-way links
- Transfer is asynchronous
- Links can be down for days or weeks if required
- It is the application's responsibility to ensure there are no conflicts

Asynchronous Multi-Master in Action



Best practice:
Do not log slave updates

Connect to Any Node for Writes

```
(sfo)
mysql> create table test
(id int primary key
auto_increment, data
varchar(30));

mysql> insert into
test(data) values('sfo');

mysql> select * from test;
+----+------+
| id | data |
+----+------+
|  3 | lax  |
|  4 | sfo  |
+----+------+
```

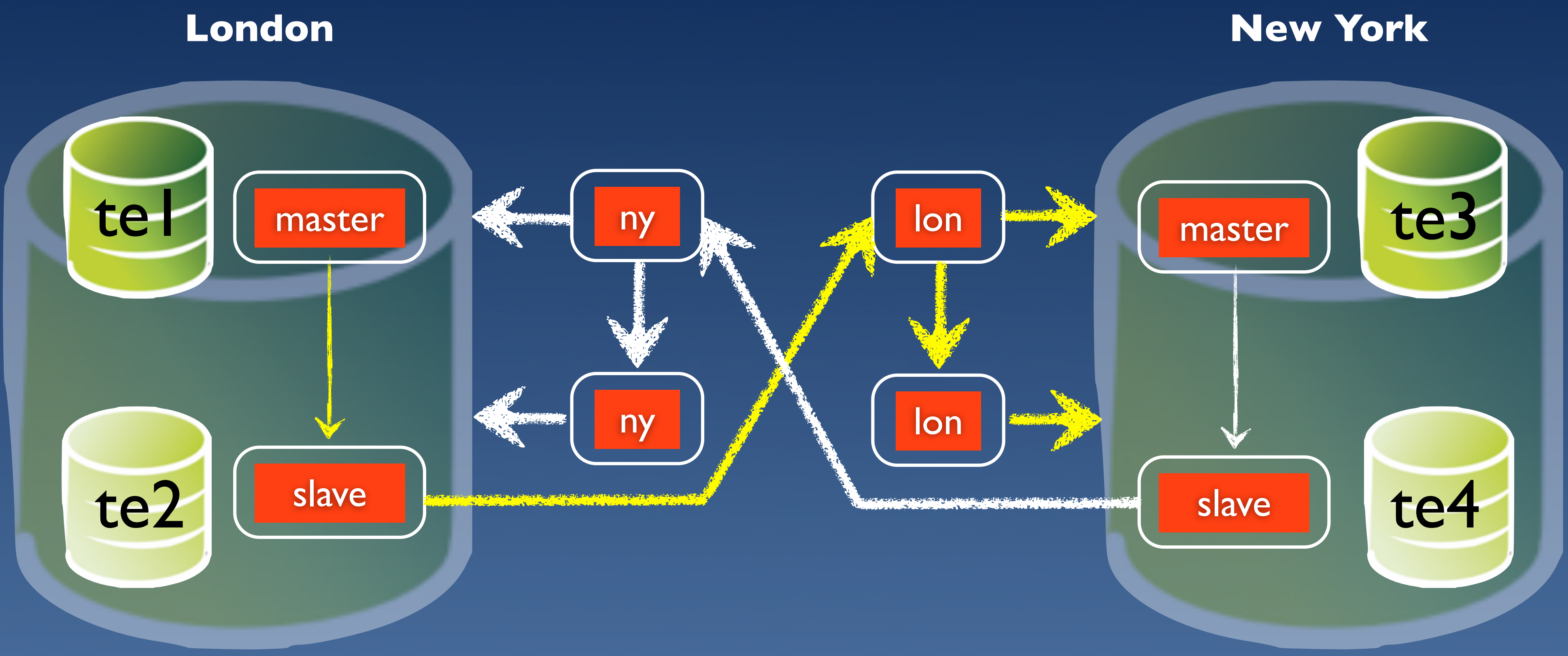
```
(lax)

mysql> insert into
test(data) values('lax');

mysql> select * from test;
+----+------+
| id | data |
+----+------+
|  3 | lax  |
|  4 | sfo  |
+----+------+
```

Auto_increment keys must be manually configured

Failure Handling

- Replication stops and resumes automatically when network link goes down
- Replication stops on replicator or DBMS failure and recovers after operator restart
- Conflicts can break replication
- Reconciliation is manual and potentially very difficult
- Use binlogs to trace inconsistencies

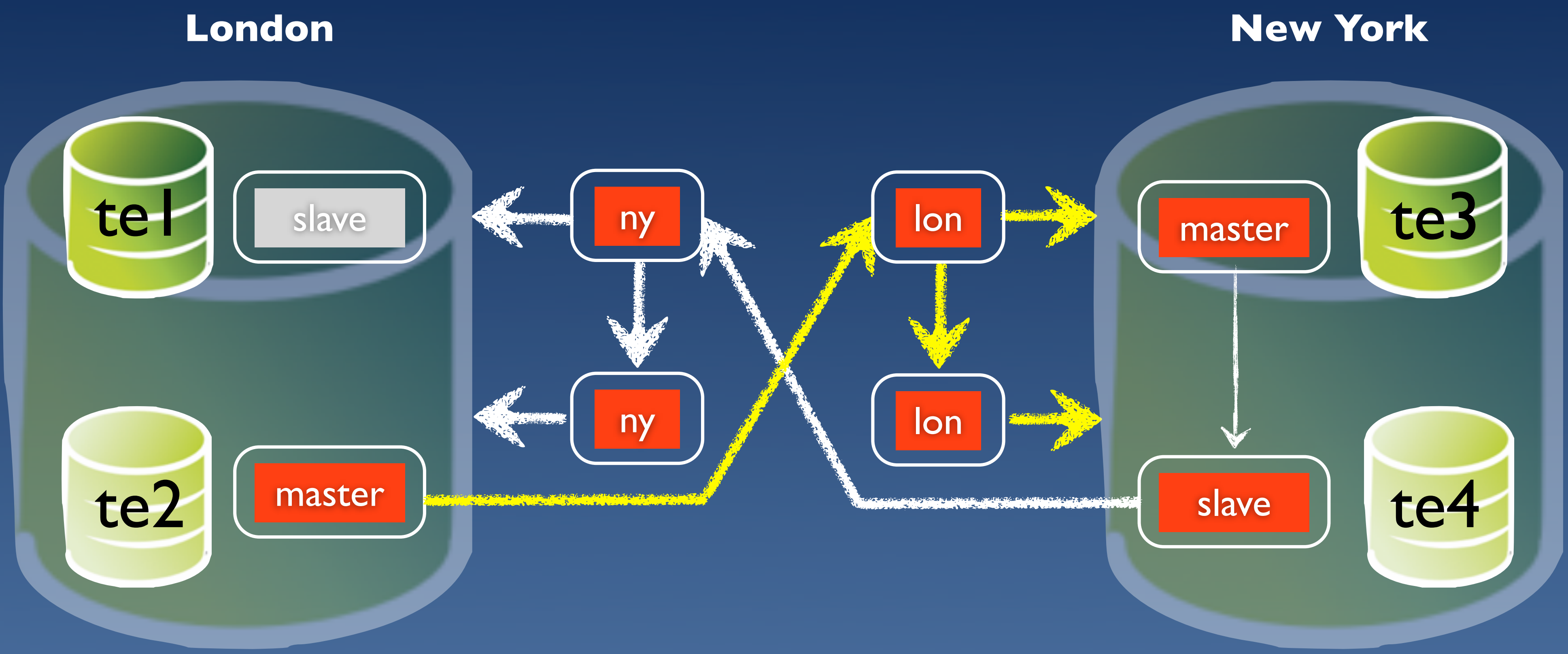
Tungsten Multi-Master Clusters



Cross-site replicators read from slaves

Cross-site updates are unlogged

Connect to Master when Slave Fails



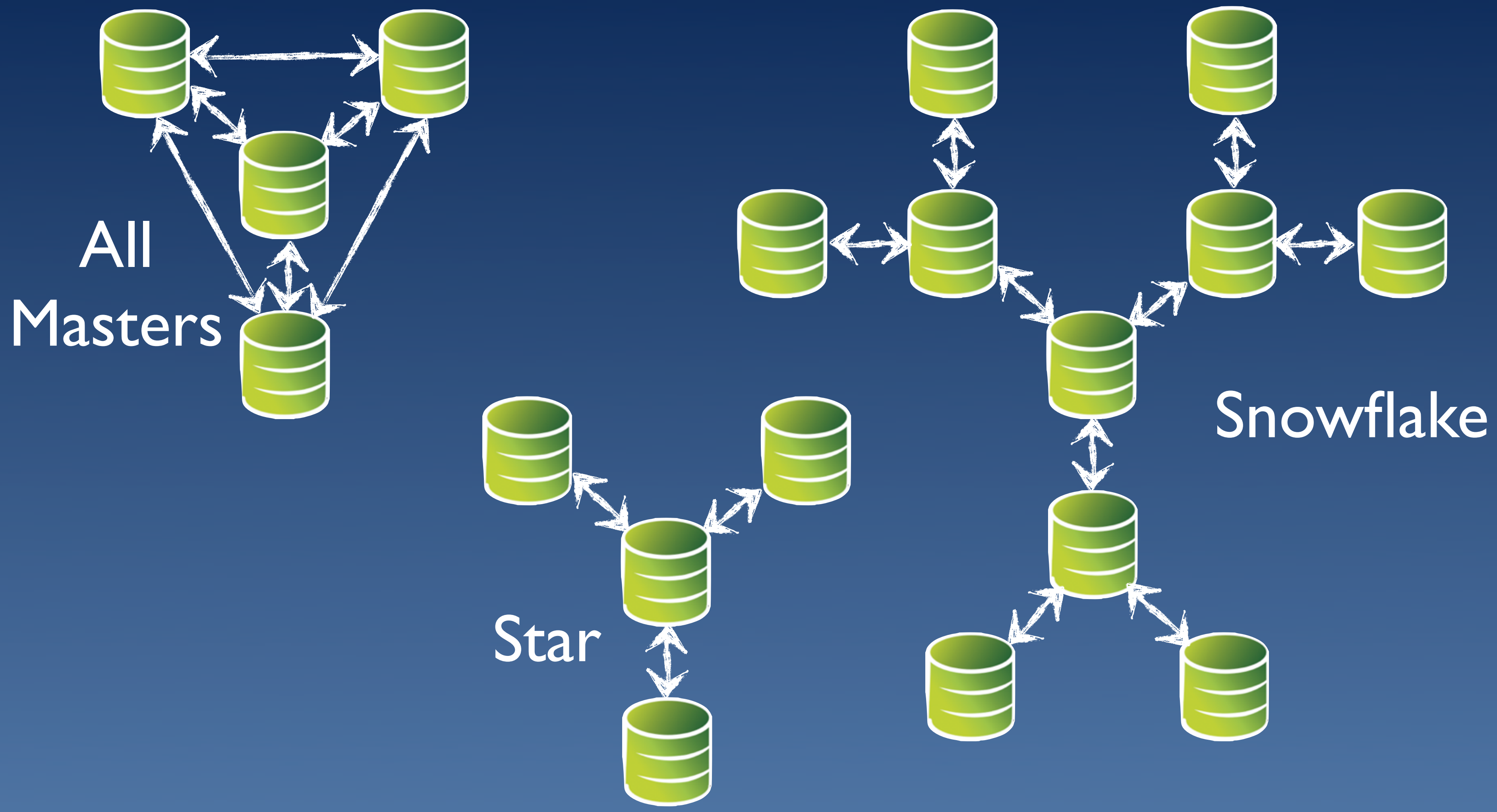
(Slave failure/maintenance)

Use Filters to Detect/Avoid Conflicts

Name	Purpose
ignore_server	Drop transactions from server-id
rename	Rename schemas/tables/columns
replicate	Control schemas/table replication
shardfilter	Control shard replication

You can also write your own filters!

Wide Range of Topologies



Tungsten Summary

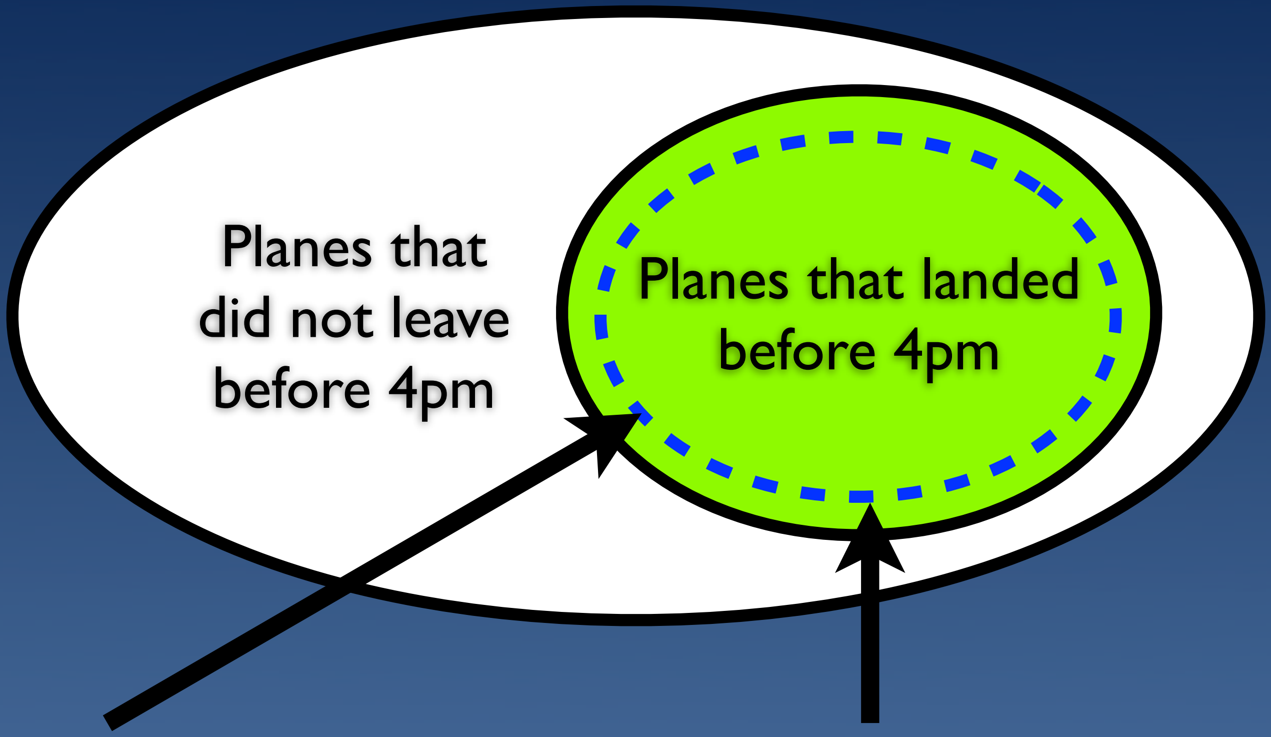
- Big plus #1: flexibility
- Big plus #2: handles mixed workloads/dirty operating conditions well
- Replication is in large-scale production use
- Can link between local clusters across sites
- Lacks conflict resolution
- Filters are powerful but a pain to write

What Will Be in Next Year’s Talk?

Improvements to Replication

- Oracle: Stable GTIDs and a host of replication improvements
- MariaDB: Multi-source replication, GTIDs, more improvements
- Galera: Cover more use cases, GTIDs
- Tungsten: Conflict resolution and better filters

Better Programming Models



Planes that did not leave before 4pm

Planes that landed before 4pm

SELECT flights that did not land before 4pm

Select flights that landed before 4pm

More Data on Multi-Site Operation

- Public cloud infrastructure is improving
- Networks are more stable
- Synchronous replication is back in favor
- We’ll see how it works out!

560 S. Winchester Blvd., Suite 500
San Jose, CA 95128
Tel +1 (866) 998-3642
Fax +1 (408) 668-1009
e-mail: sales@continuent.com

Our Blogs:
http://scale-out-blog.blogspot.com
http://datacharmer.org/blog
http://www.continuent.com/news/blogs

Continuent Web Page:
http://www.continuent.com

Tungsten Replicator 2.0:
http://code.google.com/p/tungsten-replicator