

Replication Techniques

MySQL Snapshots and Replication from Oracle

- theo schlossnagle
omniti

theo
schlossnagle

CTO
omniti
"scalability junkies"



what is replication

— [replication is the “Holy Grail” of database technologies

— as such, no one really does it well

— [replication **is** N-way, multi-master replication without:

— the need for conflict resolution

— changing your understanding of database semantics

— pain, suffering and performance degradation

but (insert vendor) replicates!

- [today's multi-master replication uses N-way 2-phase commit

- i actually can't conceive of a more inefficient approach

- [today's master-slave replication...

- actually works pretty well...

- just call it "single-master" or "master-slave" replication

cross-vendor replication (1 of 2)

— [master-master replication requires that the internals of database transactions be exposed to the replication protocol to allow for parallel transactions to proceed efficiently

— this is more or less impossible to do between two database vendors

— this is not the point of this talk

cross-vendor replication (2 of 2)

— [master-slave replication is typically a database-native feature

— Oracle re-applies archive redo logs

— MySQL pipes binlogs to slaves.

— [this makes it infeasible to reuse the “native” feature for cross-vendor master-slave replication

— [**this** is the point of this talk

Vernacular (1 of 2)

— [**OLTP: Online Transaction Processing**

small, fast queries that require immediate satisfaction

— [**ODS: Operational Datastore**

all types of queries including gigantic 5-day monsters

— [**DDL: Data Definition Language**

— [**DML: Data Modification Language**

— [**Queries: non-modifying questions to the database**

Vernacular (2 of 2)

— [**Inexpensive, small affect queries**
cheap execution, small underlying data change

— [**Expensive, small affect queries**
expensive execution, small underlying data change

— [**Inexpensive, large affect queries**
cheap execution, large underlying data change

— [**Expensive, large affect queries**
expensive execution, large underlying data change

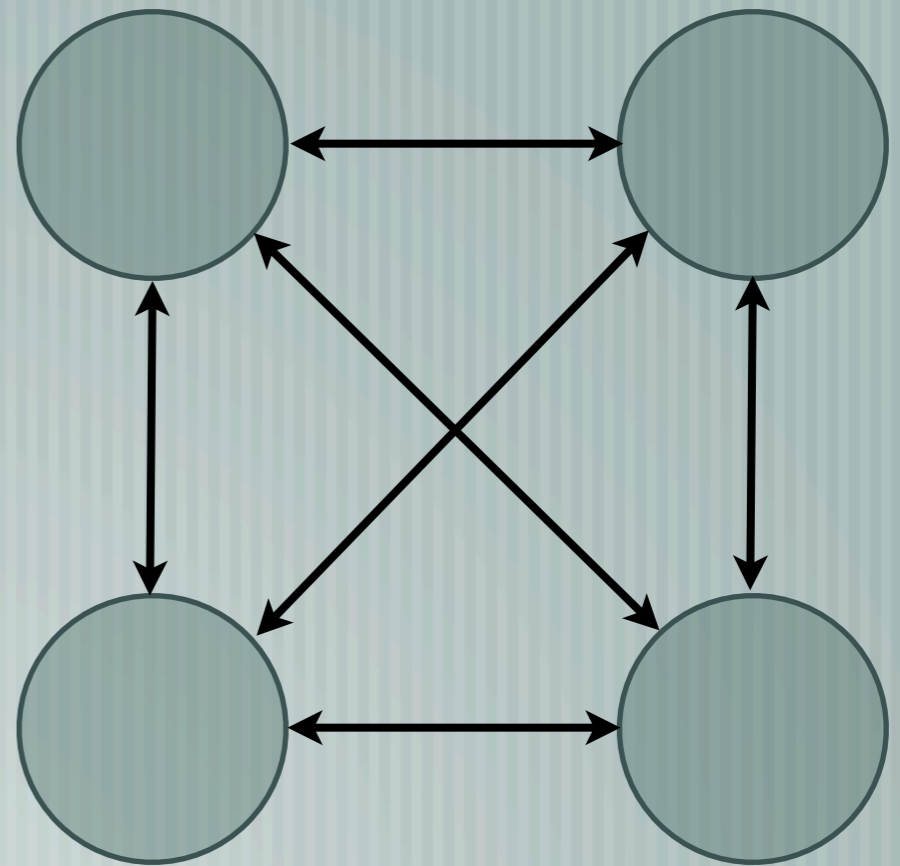
defining a problem scope

Required: 4-way multi-master replication with all the bells and whistles

we trust Oracle to do this

existing infrastructure

behaves as advertised

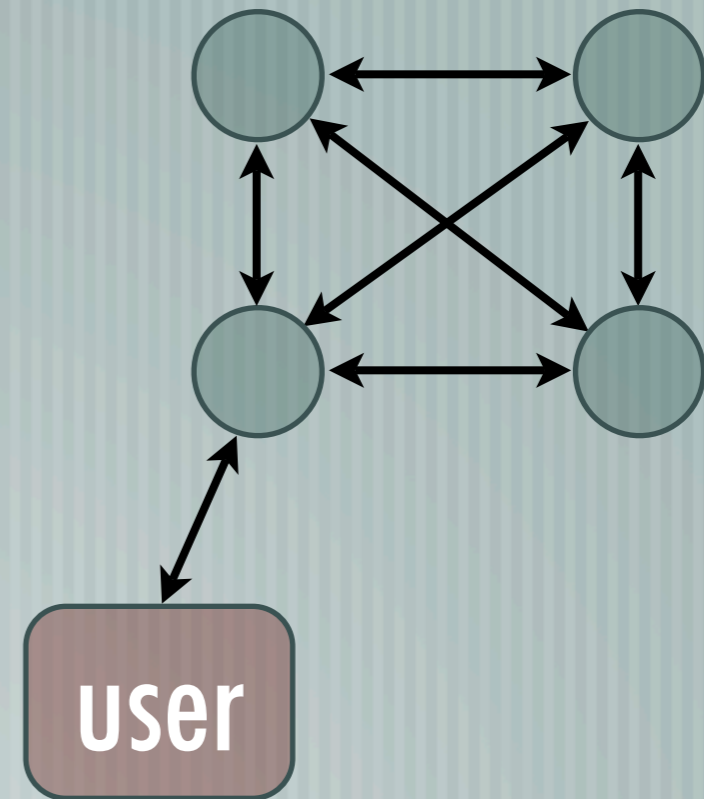


refining the problem scope

10 to 100 updates per second

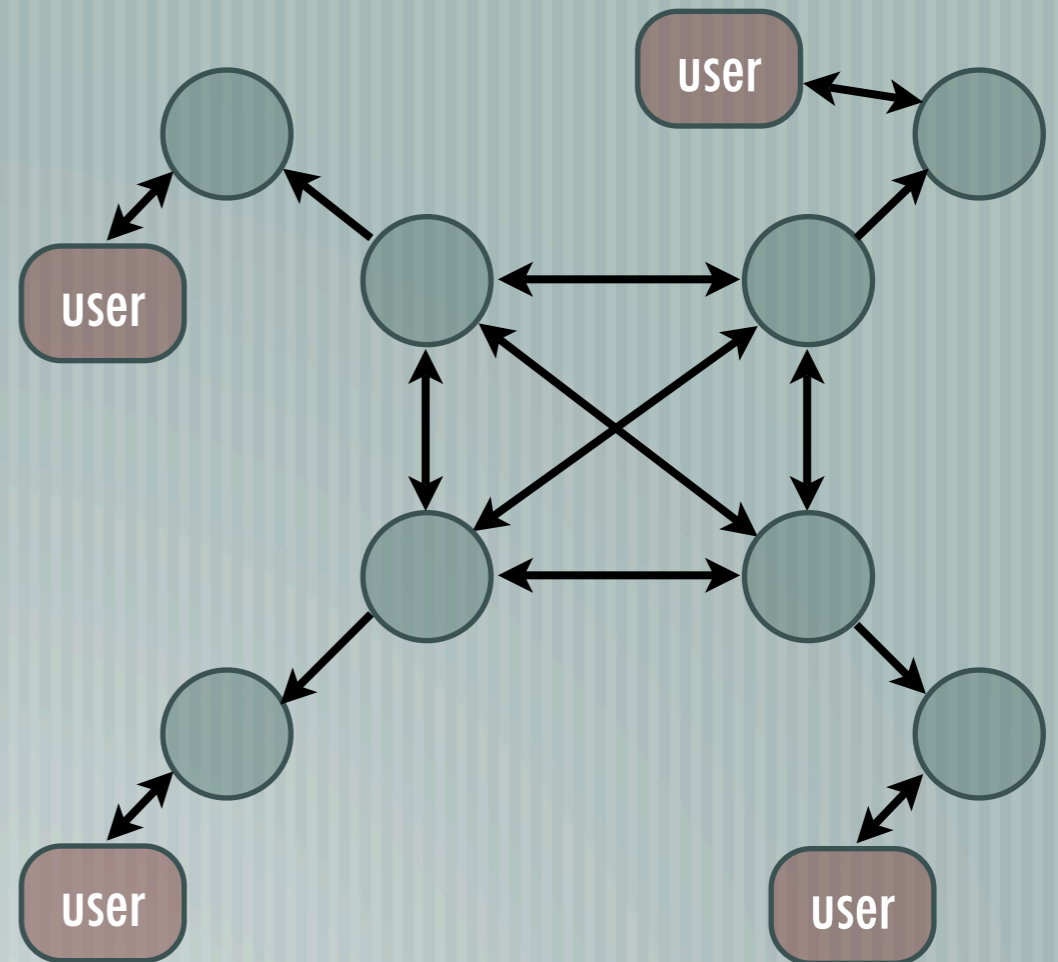
200 to 500 queries per second

... safe so far

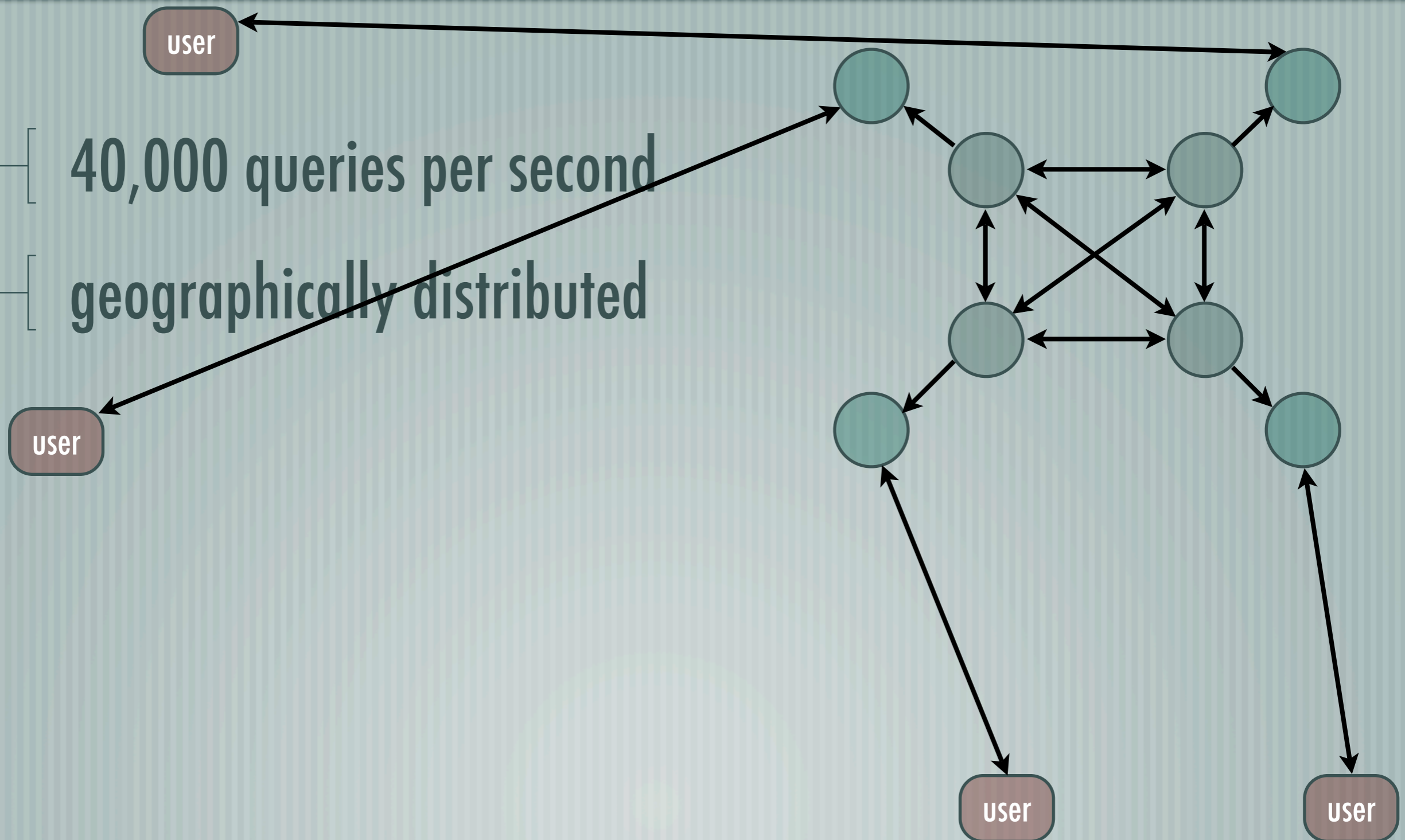


redefining the problem scope

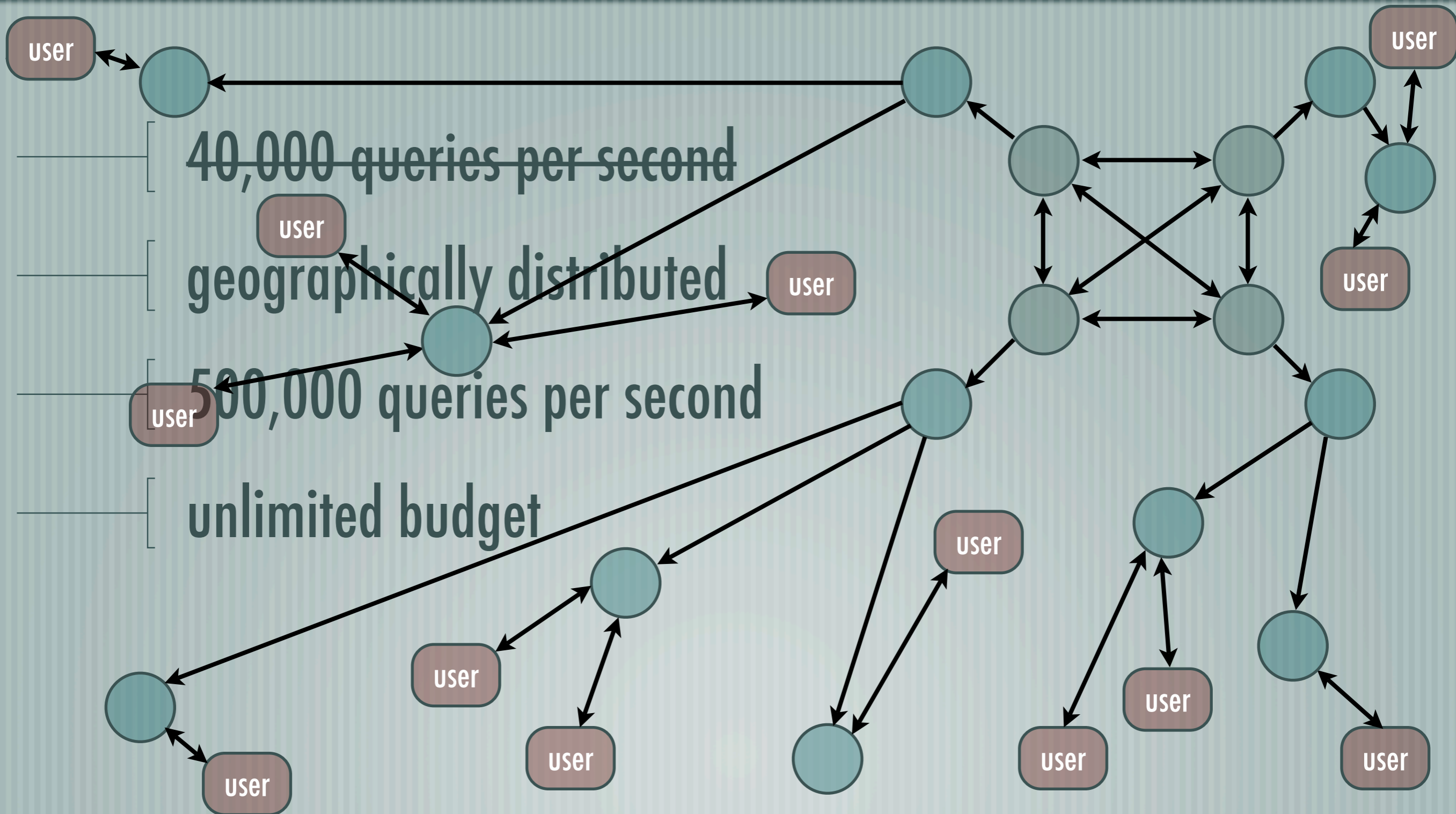
40,000 queries per second



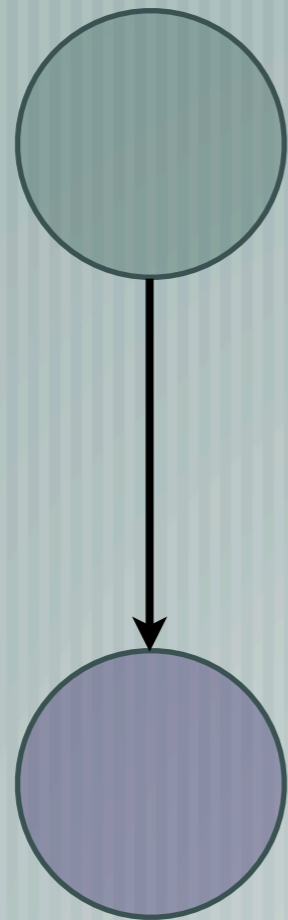
redefining the problem scope



redefining the problem scope



simply green to purple



— [cost-effective

— [one-way
(master-slave)

— [application level

— [manually configured
(you choose which tables)

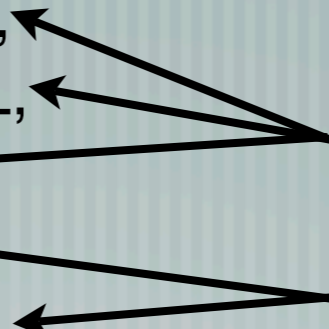
Tackling a table

```
CREATE TABLE important_table
(
  server VARCHAR(128) NOT NULL,
  daemon VARCHAR(32) NOT NULL,
  item VARCHAR(32) NOT NULL,
  value VARCHAR(32) NOT NULL,
  modified DATE DEFAULT(sysdate),
  CONSTRAINT important_table_pk
    PRIMARY KEY (server,daemon,item)
);
CREATE INDEX important_table_server
  ON important_table ( server );
CREATE INDEX important_table_value
  ON important_table ( value );
```

Define vocabulary:

these are "pks"

these are "npks"



Building a DML log

```
CREATE TABLE log_important_table
(
  seq_id number,
  timestamp date,
  dmlmode CHAR(1) NOT NULL,

  old_server VARCHAR(128),
  old_daemon VARCHAR(32),
  old_item VARCHAR(32),

  server VARCHAR(128),
  daemon VARCHAR(32),
  item VARCHAR(32),
  value VARCHAR(32),
  modified DATE,
  CONSTRAINT log_important_table_pk PRIMARY KEY(seq_id)
);
CREATE INDEX log_important_table_idx on log_important_table(timestamp);

CREATE SEQUENCE seq_log_important_table
  start with 1 increment by 1 cache 100 order;
```

these are reference "pks"

these are new "pks"

these are new "npks"

Populating the DML log

```
CREATE TRIGGER important_table_trig
AFTER INSERT OR UPDATE OR DELETE ON important_table
FOR EACH ROW
DECLARE
    v_dmlmode CHAR(1);
    v_r_server VARCHAR(128);
    v_r_daemon VARCHAR(32);
    v_r_item VARCHAR(32);
    v_server VARCHAR(128);
    v_daemon VARCHAR(32);
    v_item VARCHAR(32);
    v_value VARCHAR(32);
    v_modified DATE;

BEGIN
IF INSERTING THEN
    v_dmlmode := 'I';
    v_server :=:new.server;
    v_daemon :=:new.daemon;
    v_item :=:new.item;
    v_value :=:new.value;
    v_modified :=:new.modified;
END IF;
IF UPDATING THEN
    v_dmlmode := 'U';
    v_r_server := :old.server;
    v_r_daemon := :old.daemon;
    v_r_item := :old.item;
    v_server :=nvl(:new.server,:old.server);
    v_daemon :=nvl(:new.daemon,:old.daemon);
    v_item :=nvl(:new.item,:old.item);
    v_value :=nvl(:new.value,:old.value);
    v_modified :=nvl(:new.modified,:old.modified);
END IF;
```

```
IF DELETING THEN
    v_dmlmode := 'D';
    v_server :=:old.server;
    v_daemon :=:old.daemon;
    v_item :=:old.item;
    v_value :=:old.value;
    v_modified :=:old.modified;
END IF;

INSERT INTO log_important_table(
    seq_id,
    timestamp,
    dmlmode,
    r_server,
    r_daemon,
    r_item,
    server,
    daemon,
    item,
    value,
    modified
)
values(
    seq_log_important_table.nextval,
    sysdate,
    v_dmlmode,
    v_r_server,
    v_r_daemon,
    v_r_item,
    v_server,
    v_daemon,
    v_item,
    v_value,
    v_modified
);

END;
```

Where are we now?

— [All data modifications on important_table are tracked

— [Logged into a SQL accessible log table that can be

— replayed against another data source

— managed via SQL (expunged, analyzed, archived)

Replaying the DML log

```
connect to master;
connect to slave;
while(1):
    $tables := select table_name from slave.checkpoints;
    foreach $table in $tables:

        $chkpt := select last_chkpt from slave.checkpoints where table_name=$table;

        $cur := select * from master.log_$table where seq_id > $chkpt
                order by seq_id;
        foreach $row in $cur:
            $last_id := $row.seq_id
            case $row.dml_type in
                'D') delete from slave.$table where pks = $row.r_pks;
                'I') insert into slave.$table (pks, npks) VALUES($row.pks, $row.npks);
                'M') update slave.$table set pks = $row.pks, npks = $row.npks
                    where pks = $row.r_pks;
            update slave.checkpoints set last_chkpt = $last_id where table_name=$table;
            slave.commit;
    sleep 30;
```

What's wrong with this?

The race condition

Process 1:

```
INSERT INTO important_table VALUES(some stuff);  
  trigger: INSERT INTO log_important_table  
           select log_important_table_seq.nextval
```

COMMIT;

Process 2:

```
INSERT INTO important_table VALUES(some other stuff);  
  trigger: INSERT INTO log_important_table  
           select log_important_table_seq.nextval
```

COMMIT;

```
1: INSERT INTO important_table VALUES(some stuff);  
1:   trigger: INSERT INTO log_important_table  
1:     select log_important_table_seq.nextval  (4)  
2: INSERT INTO important_table VALUES(some other stuff);  
2:   trigger: INSERT INTO log_important_table  
2:     select log_important_table_seq.nextval  (5)  
2: COMMIT;  
R: SELECT * FROM log WHERE seq_id > :last_seen (3)  
R: APPLY  (5... 4 isn't there, 1: didn't commit yet)  
R: UPDATE local checkpoint to 5.  
1: COMMIT; (and lost forever)
```

Replication Process:

```
SELECT * FROM log WHERE seq_id > :last_seen  
APPLY  
Update local checkpoint
```

Solution 1

```
$lag = 5/60; # 5 minutes
connect to master;
connect to slave;
while(1):
    $tables := select table_name from slave.checkpoints;
    foreach $table in $tables:

        $chkpt := select last_chkpt from slave.checkpoints where table_name=$table;

        $cur := select * from master.log_$table where seq_id > $chkpt
                and timestamp > sysdate - $lag
                order by seq_id;
        foreach $row in $cur:
            $last_id := $row.seq_id
            case $row.dml_type in
                'D') delete from slave.$table where pks = $row.r_pks;
                'I') insert into slave.$table (pks, npks) VALUES($row.pks, $row.npks);
                'M') update slave.$table set pks = $row.pks, npks = $row.npks
                    where pks = $row.r_pks;
            update slave.checkpoints set last_chkpt = $last_id where table_name=$table;
            slave.commit;
        sleep 30;
```

Requires intimate knowledge of how commits are done against your data model
– still unsafe

Solution 2

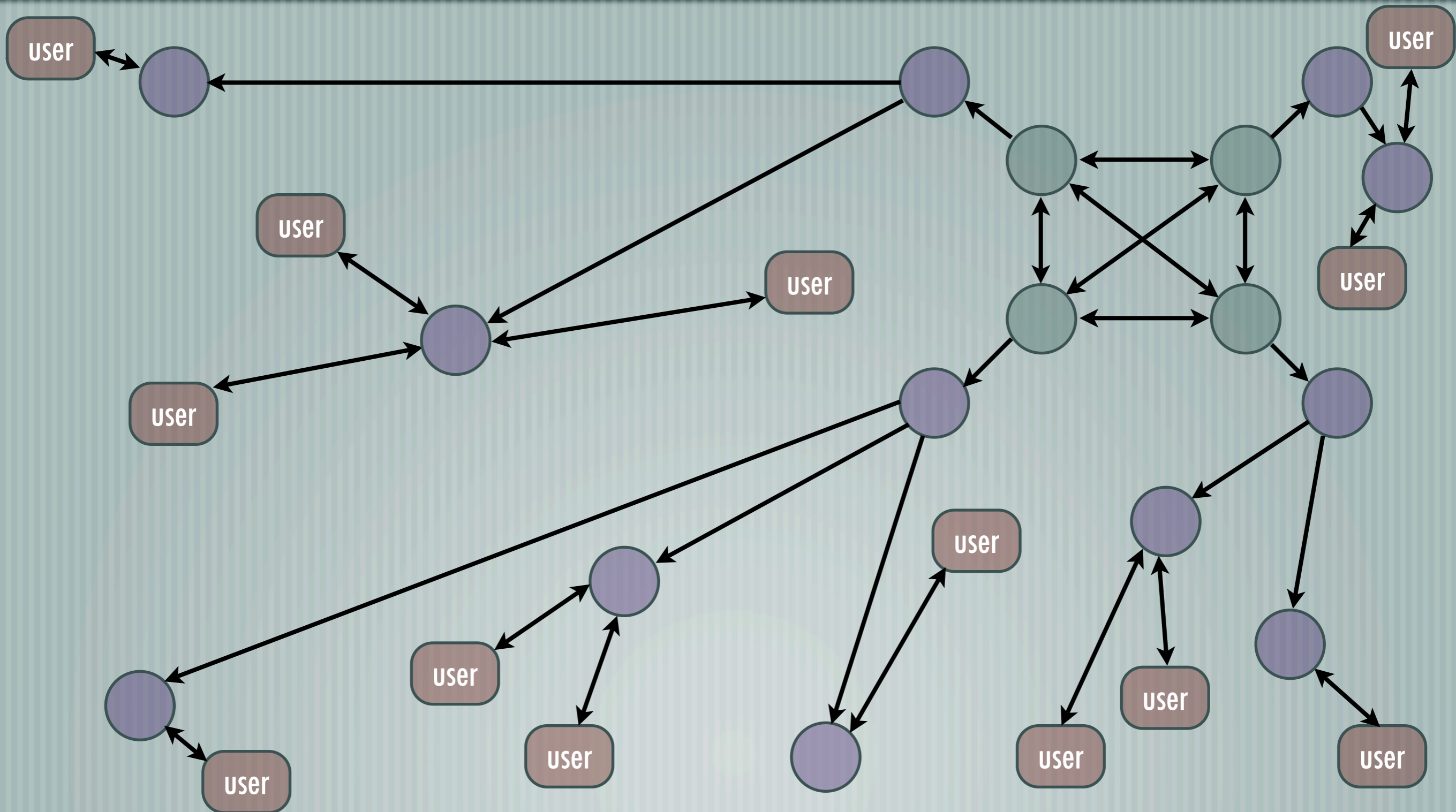
```
$slavename = `hostname`;
$lag = 5/60; # 5 minutes
connect to master;
connect to slave;
while(1):
    $tables := select table_name from checkpoints;
    foreach $table in $tables:

        $cur := select t.* from master.log_$table t, master.chkpt_$table c
                where t.seq_id = c.seq_id(+)
                and $slavename = c.slave_name(+)
                and c.seq_id is NULL;

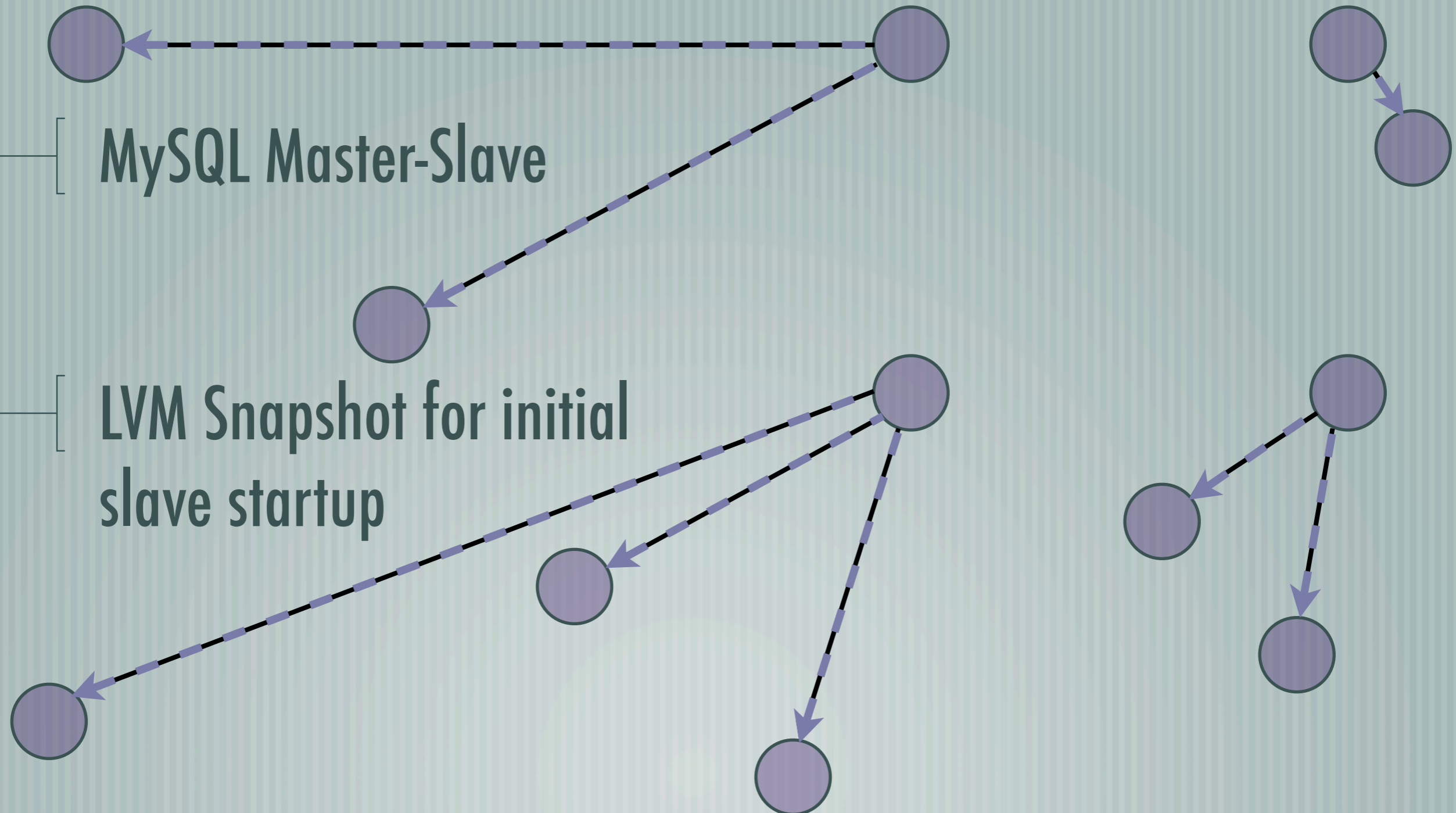
        foreach $row in $cur:
            case $row.dml_type in
                'D') delete from $table where pks = $row.pks;
                'I') insert into $table (pks, npks) VALUES($row.pks, $row.npks);
                'M') update $table set pks = $row.pks and npks = $row.npks
                    where pks = $row.pks;
            insert into master.chkpt_$table VALUES($slavename, $row.seq_id);
        slave.commit;
        master.commit;
    sleep 30;
```

Correct (not XA), but the master is taxed for every slave.

revisiting the problem scope



revisiting the problem scope



Does it work?

— [Currently serving...

— The entire Internet

— UltraDNS hosting $> 10^7$ records,
feel free to test it with "dig"

Conclusion

— [Old technology and ideas... but damn useful

— [It works and can save millions of dollars scaling read-intensive operational data stores

— [Many thanks to UltraDNS for allowing a reference

— [Any Questions?