Temperature rising: Hot Standby In PostgreSQL 9.0

Open Source Days, March 2010
Copenhagen, Denmark

Magnus Hagander
Redpill Linpro AB

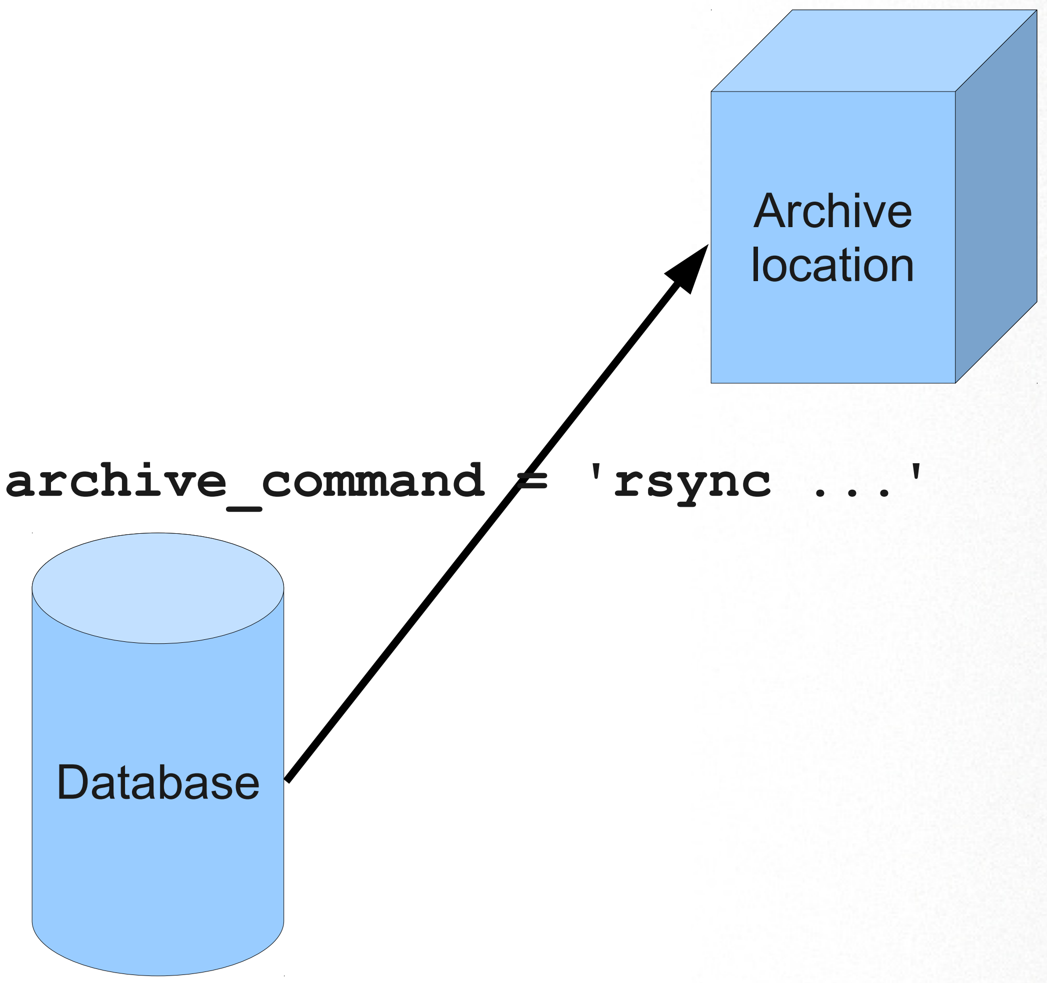
It's not just Hot Standby

- Builds on Warm Standby
- Becomes powerful with Streaming Replication
- 1+1 = 3 (or more!)
- So let's discuss them all

Warm Standby

- Introduced in PostgreSQL 8.2
  - Actually existed before, but not included in core, and with many caveats
- Based on transaction log
- Same as Point In Time Recovery
- Runs normal crash recovery code
  - Just never finishes

PITR – Archive Logging

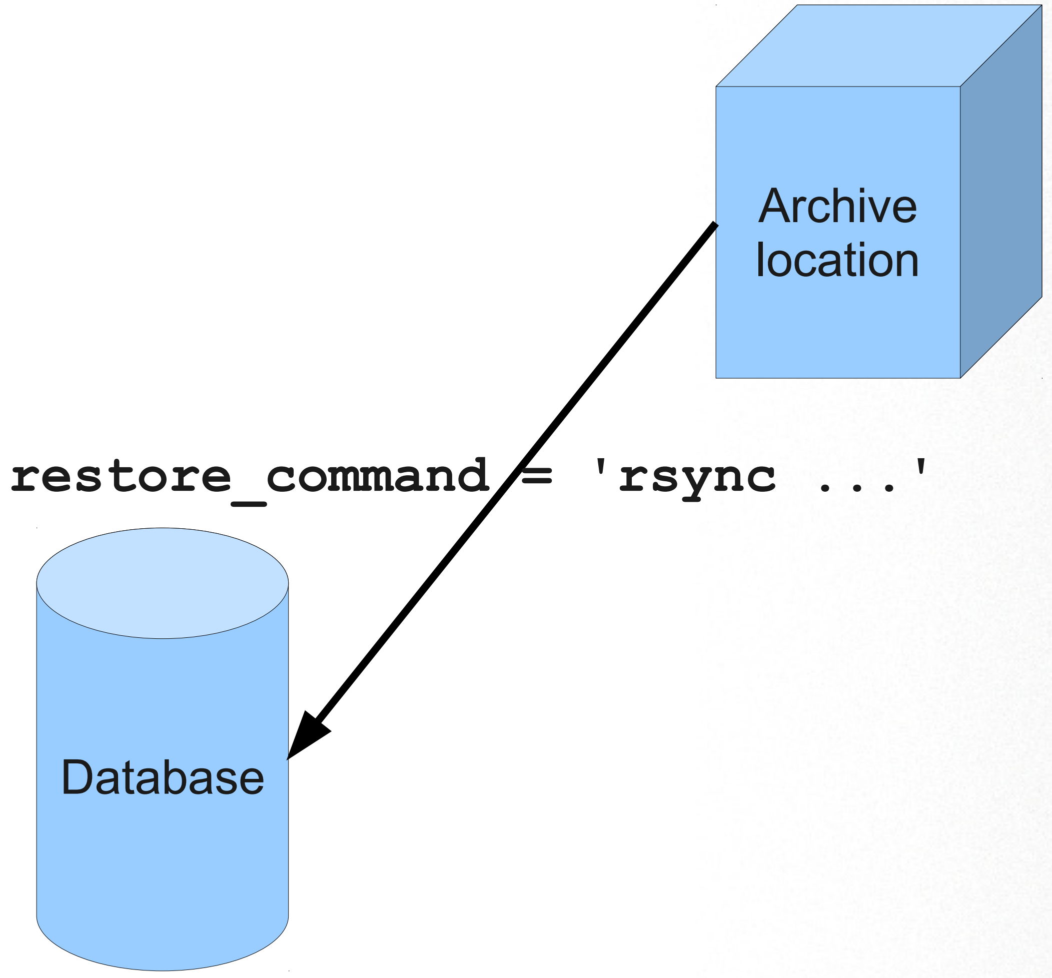


```
archive_command = 'rsync ...'
```

Point In Time Recovery

- Each log file (16MB) shipped when filled with data
- Or when *archive_timeout* has expired
- Leaves dataloss window at max *archive_timeout*

PITR – Recovery



PITR - Recovery

- Reads all transaction log files
- Until there are no more, or until recovery time has been reached
- Re-applies all changes sequentially

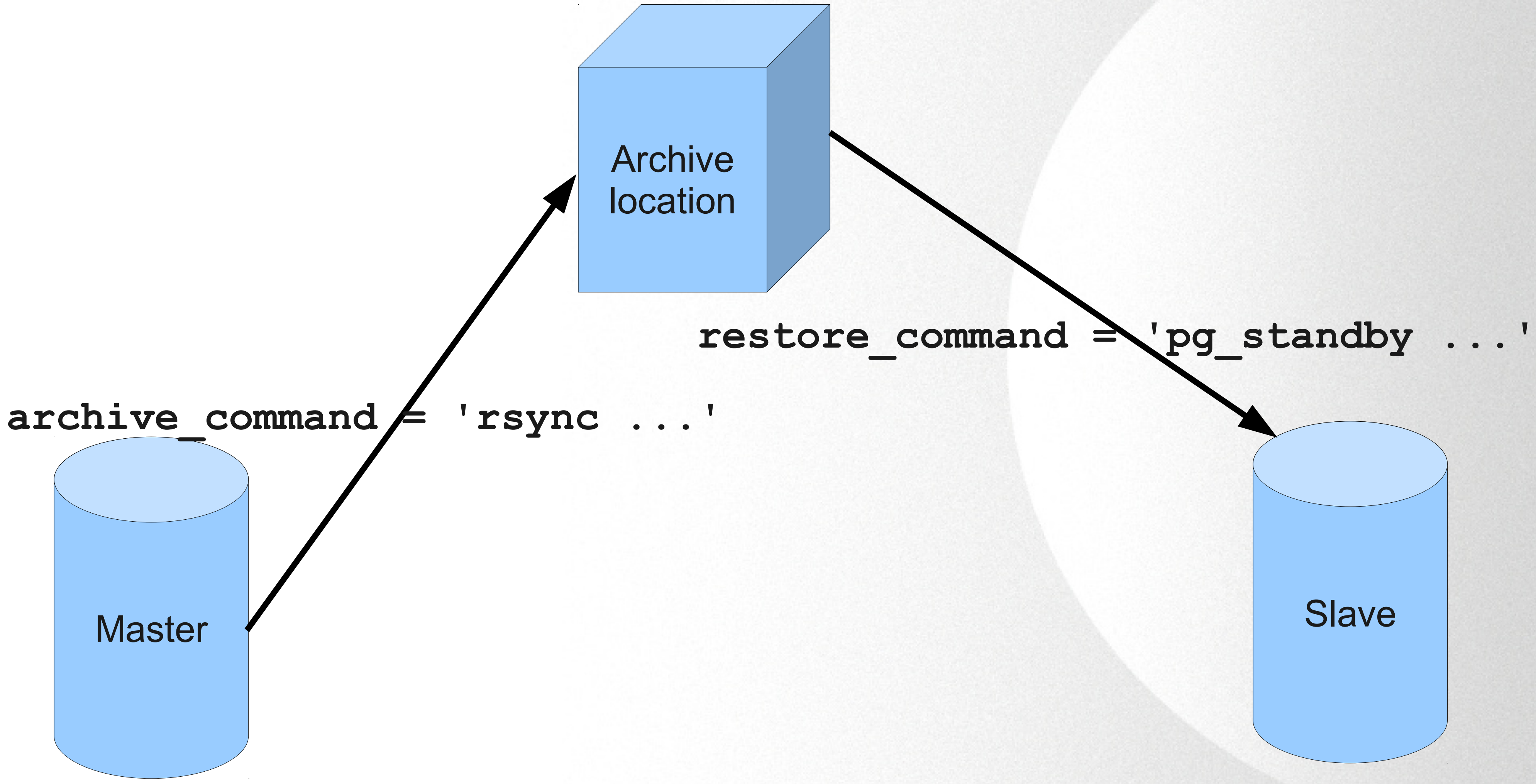
DEMO

- (that's never going to work)

Warm Standby

- Combine log archiving and PITR recovery
- Just never finish recovery
- Reference implementation: pg_standby in contrib
- Polls for new logs until trigger

Warm Standby



```
restore_command = 'pg_standby ...'
```

```
archive_command = 'rsync ...'
```

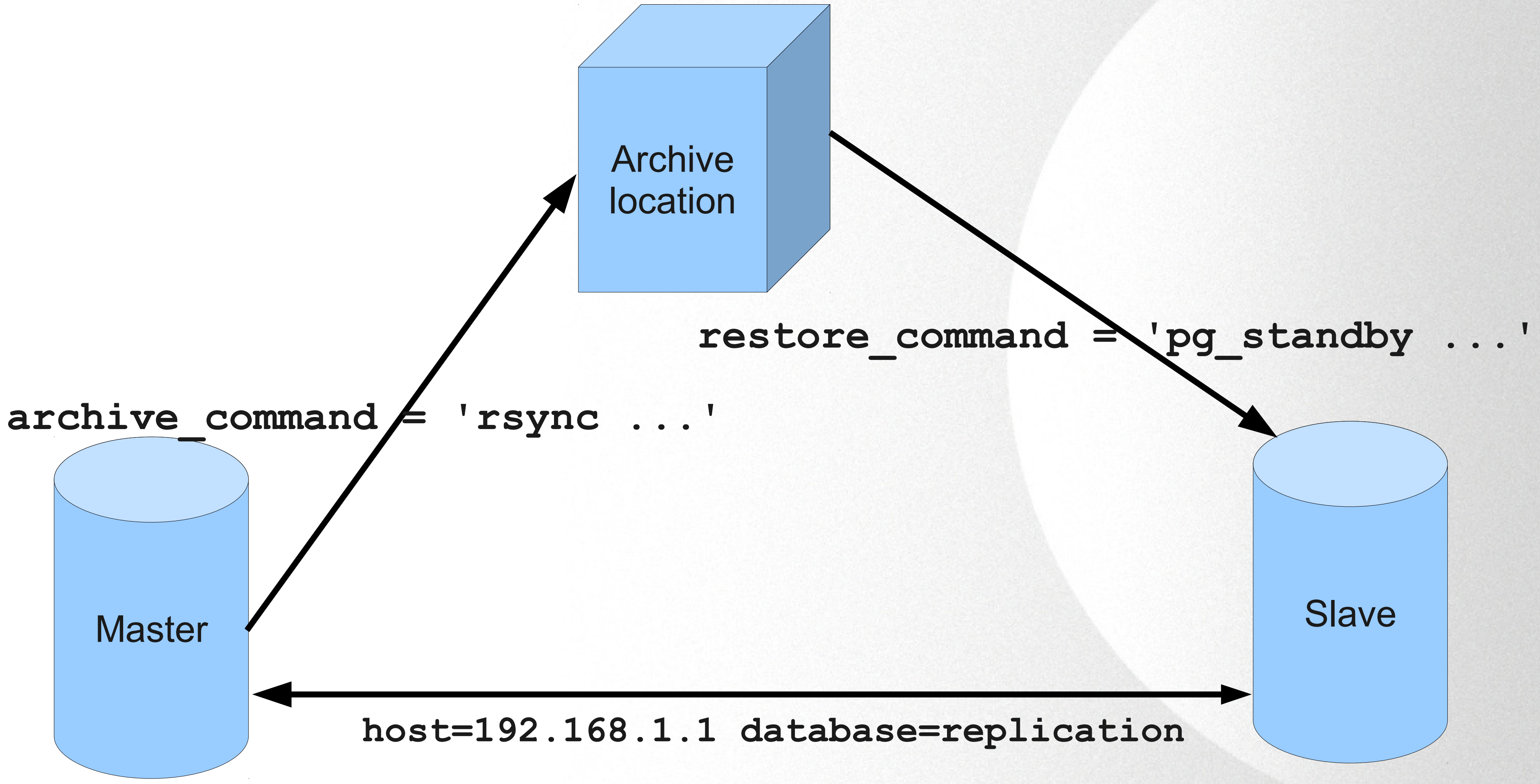
DEMO

- (he's clearly insane)

Streaming Replication

- Problem: high archive_timeout leads to high latency
- Problem: low archive_timeout leads to excessive disk and I/O
- Solution: Streaming Replication

Streaming Replication


Archive location
restore_command = 'pg_standby ...'
archive_command = 'rsync ...'
Master
Slave
host=192.168.1.1 database=replication

Streaming Replication

- First uses regular full backup to get a baseline
- Second uses regular restore_command to catch up
- Third, enables streaming mode

DEMO

- (another demo? It'll break!)

Streaming Replication

- Replicated data is sent in near real-time
  - Default wal_sender_delay = 200ms
- Terminated by trigger
  - If no trigger, never stops

Hot Standby

- Works in combination with Streaming Replication
- *Or* with pg_standby
- Or, actually, with regular recovery

Hot Standby

- Warm standby isn't even read-only
- You can't do anything until it's «opened»
- Once «opened», has to restart to catch up
- Set *recovery_connections=On*...

DEMO

- (whatever worked so far, must be pure luck)

Hot Standby

- Slave becomes *read only*
- No DDL, no DML, no share locks, no exclusive locks, no two-phase commit, no sequence changes
- Not even temporary tables!

Hot Standby

- Transaction isolation *works*
- Between master and slave
- MVCC snapshots preserved

Query Conflicts

- Master changes *will* conflict with slave, when long-running queries
  - Access Exclusive locks
  - Dropping tablespaces
  - Dropping databases
  - «Early cleanup» in btree, HOT
- Yes, we've implemented «snapshot too old»

Query Conflicts

- *max_standby_delay*
  - Controls how long we wait to apply log
  - When there is an active query on the slave
  - Then we just kill it
  - A tradeoff between availability and «reporting»

Query Conflicts

- *vacuum_defer_cleanup_age*
  - On the *master*
  - Delays how long it takes before VACUUM attempts to clean up
  - Increases bloat on master!

Summary

- There are obvious tradeoffs
  - Particularly in Hot Standby
- We want to know how it works in *your* environment!
- Download 9.0alpha4 and test, test, test and test!

Oh, and did I mention?

- Please test this for us!

Thank You!

Questions?

magnus@hagander.net
http://blog.hagander.net/
Twitter: magnushagander
FreeNode: #postgresql:magnush