

# Managing Large Sharded Topologies with Jetpants

Evan Elias

Percona Live New York 2012

# About Jetpants

---

- Automation toolkit for handling monstrously large MySQL database topologies
- NOT a middleware server, storage engine, or replication replacement
- Command line tool for common operational tasks (splitting shards, cloning replicas, performing master promotions, etc)
- Usable as a library to provide a fully scriptable object model for your DB topology
- Open source (Apache license) and written in Ruby

# Motivations for Jetpants

---

1. Split range-based shards
2. Automate operational tasks to reduce human error and time
3. Provide reusable library for complex data migrations

# Why range-based sharding?

---

- Didn't want to use a lookup table, since that's a SPOF and can hit its own size issues
- Didn't want to use a modulo or hashing scheme, too complex to rebalance given our growth rate
- To reduce operational complexity, we didn't want pre-allocation of thousands of tiny shards, or initially colocate shards per MySQL instance or machine
- Decided on a range-based sharding scheme, with blog ID as sharding key. Now we just needed a way to split shards efficiently based on ranges of blog IDs.

# Shard split automation prereqs

---

- Ability to clone slaves quickly
- Fast bulk import/export of portions of a data set
- Config file generation
- Automation around replication discovery, set-up, tear-down

# Reducing human error

---

- In-house system, Collins, to track hardware asset state
- Wanted MySQL topology changes (master promotions, slave cloning, etc) to be fully automated, AND immediately reflected in Collins
- Essential because we have a relatively large MySQL footprint vs small employee headcount

# Tumblr's Scale, Oct 2012

---

- **103 billion rows** on masters
- **28 terabytes** of distinct relational data
- **201 dedicated MySQL servers**, grouped into **5 global pools** and **58 shard pools**
- Grows by 3 or 4 shard pools per month

# Number of DBAs at Tumblr

---

ZERO

Just two engineers each spending ~50% of  
their time on non-maintenance MySQL work

# This means three things

---

1. We're hiring!
2. Percona consultants are a lifesaver
3. Automation must be powerful but still easy to use

# Reusable code library

---

## Scriptable and pluggable manipulation of DBs

- Custom scripts: shard an unsharded table, merge pools, defragment pools
- Pluggable: backups, monitoring, query killers, etc
- Generic enough to be useful for other companies

# Implementation

---

- Ruby objects representing hosts, MySQL instances, pools, shards, and the topology as a whole
- Methods implementing state discovery, topology changes, etc
- Command suite is a separate executable, using Jetpants as a library
  - Takes user input and just calls these library methods as needed
  - All domain knowledge is in the library, not the command suite
  - So every command suite action is also scriptable via the library

# How Jetpants clones replicas

---

- Shut down mysql daemon on a standby slave
- Much faster than a hot copy
- Copy all data files using tar | pigz | nc
- Can "chain" a serial copy to multiple destinations using tee and a fifo:  
<http://engineering.tumblr.com/post/7658008285>

# How Jetpants exports data

---

- Create a temporary mysql user with FILE privileges
- Conceptually divide sharding key range into N chunks (tens, hundreds, or thousands — depends on table size)
- Spawn up M threads in Jetpants, where  $M \leq N$
- Each thread pulls a chunk range off the queue, does a `SELECT ... INTO OUTFILE` query with `WHERE` clause for that chunk, and repeats until no chunks left

# How Jetpants imports data

---

- Same chunking algorithm as export, but uses LOAD DATA INFILE ... CHARACTER SET binary
- Jetpants disables binary logging before doing the import (major speed boost), re-enables it afterwards
- Concurrent import doesn't work yet on tables with AUTO\_INCREMENT, so set chunks=1 for these
- Jetpants also leverages mysqldump -d to obtain DROP TABLE / CREATE TABLE statements when needed

# Import/export use cases

---

- **Defragmentation:** Export all data, drop and recreate tables, import all data
- **Data set reduction in shard split:** Export range of data, drop and recreate tables, import range of data
- **Data migration / merging pools:** Export all data, send it somewhere else, import it there
- **Differential backups:** Export all data, diff it against previous export (not recommended, just a crazy idea)

# 3 magic bullets of data migration

---

Useful background for shard splits, pool merges, initial sharding, major schema changes

1. Redundant writes
2. Move reads separately, before stopping redundant writes
3. Choose a cutover ID “in the future”

# Redundant writes

---

- Make your app write data to an old source and a new source, until the migration is finished
- The writes to the new source are "dark writes" until migration completes — nothing reads from them yet
- If you're not making schema changes, you can leverage MySQL replication for this in interesting ways
- Otherwise, you can do the redundant writes at the app or service level

# Alter read pattern before writes

---

No way to instantaneously update app configuration on every app server. So if you adjust reads and writes simultaneously, config deploy lag causes inconsistency:

- Host A gets new config, sends writes only to new data source
- Host B doesn't have new config yet, is still reading and writing to old data source, doesn't see host A's writes

Solution is to use two steps:

1. Push out config moving reads to new data source
2. Push out config stopping redundant writes to old data source

# Future Cutover

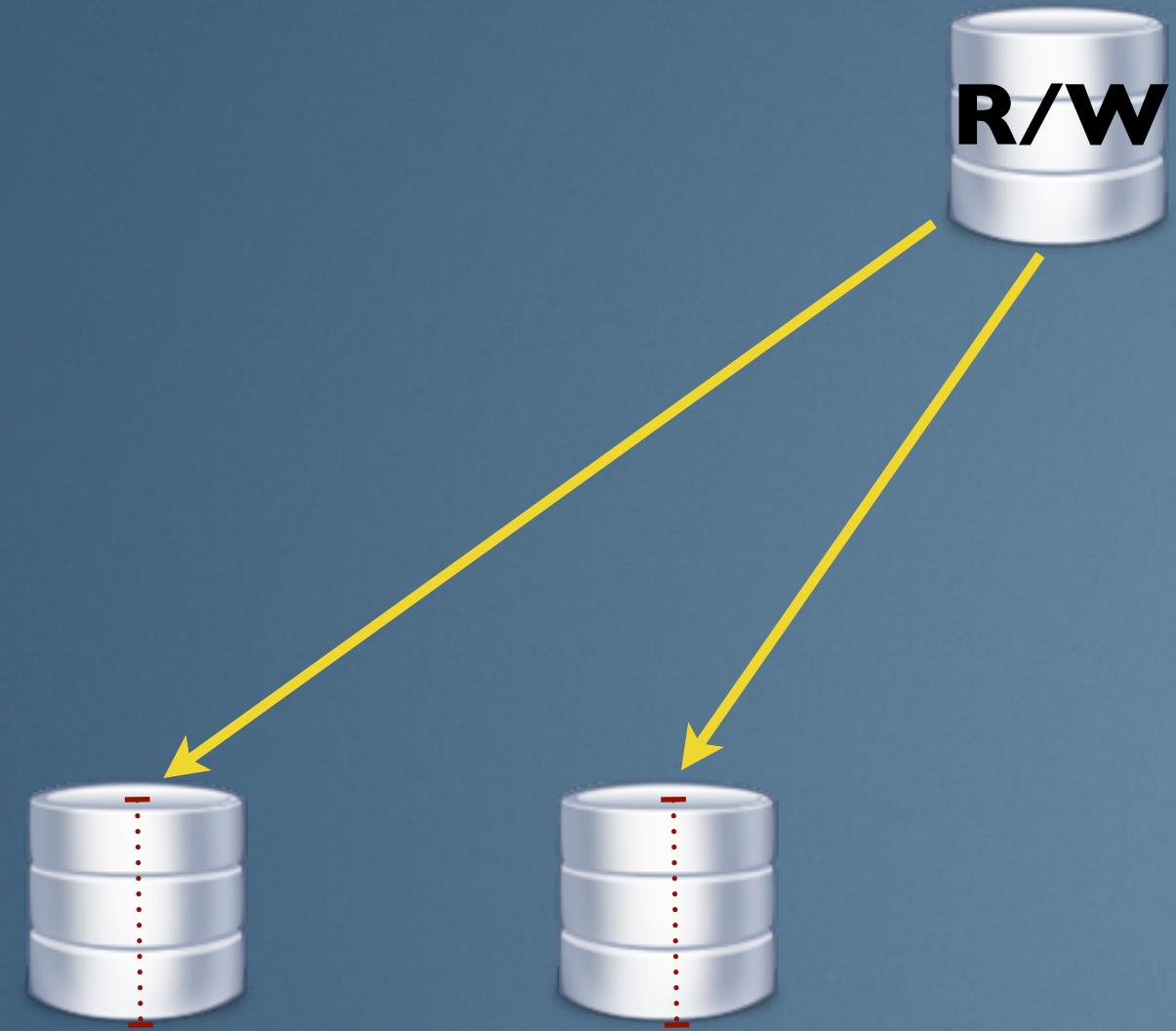
---

- Always deploy cutover logic in advance of it having any impact
- Say your last shard has range  $[5001, \infty)$ , and currently  $\text{MAX(id)} = 6800$
- Range-wise, a lot of your data set is on this one shard. You should add a new shard after it.
- Choose cutover logic to begin on ID 7001: this shard now handles  $[5001, 7000]$  and new last shard handles  $[7001, \infty)$
- Deploy this everywhere prior to ID 7001 existing. No need to move data and no data inconsistencies from configuration deploy lag!

# How Jetpants splits a shard

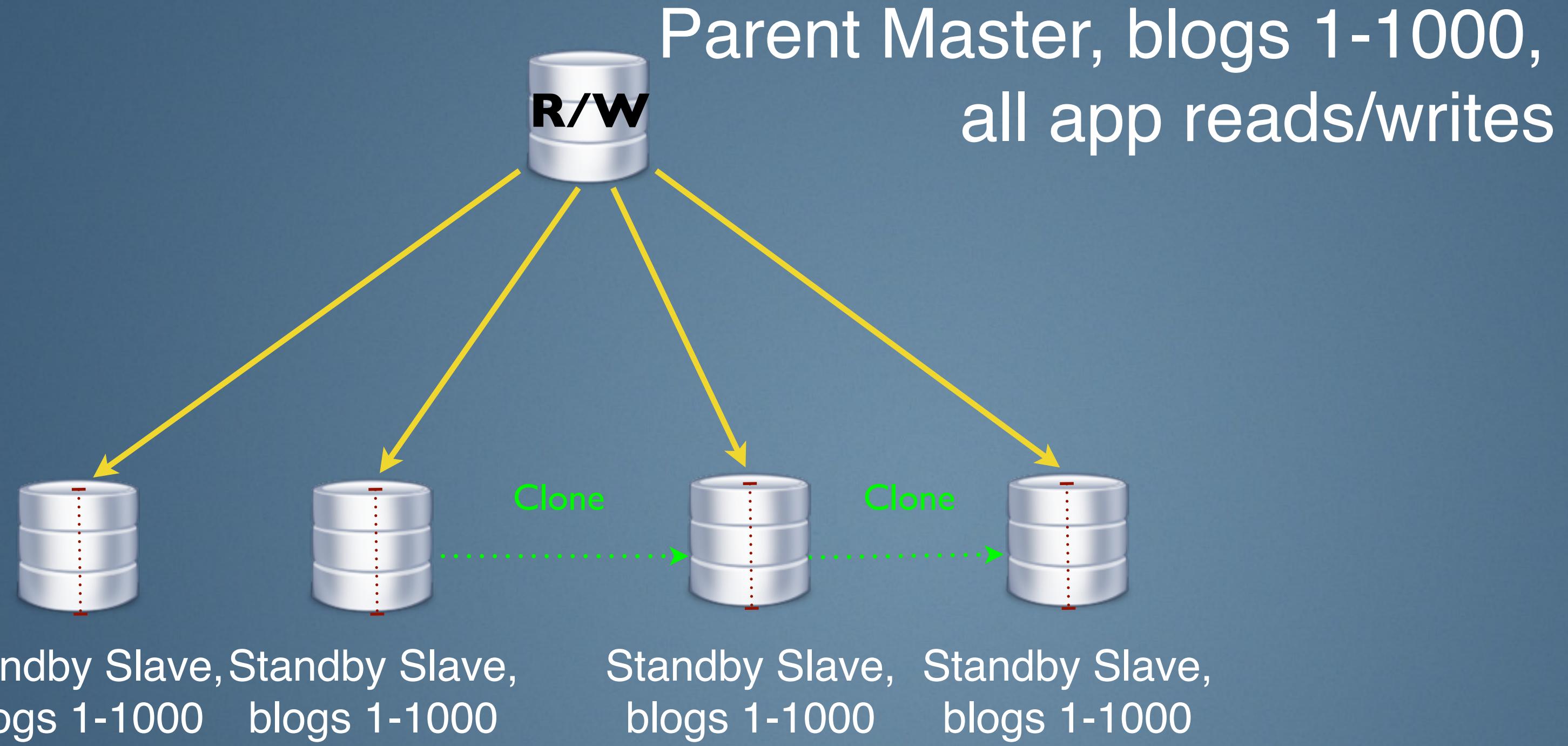
---

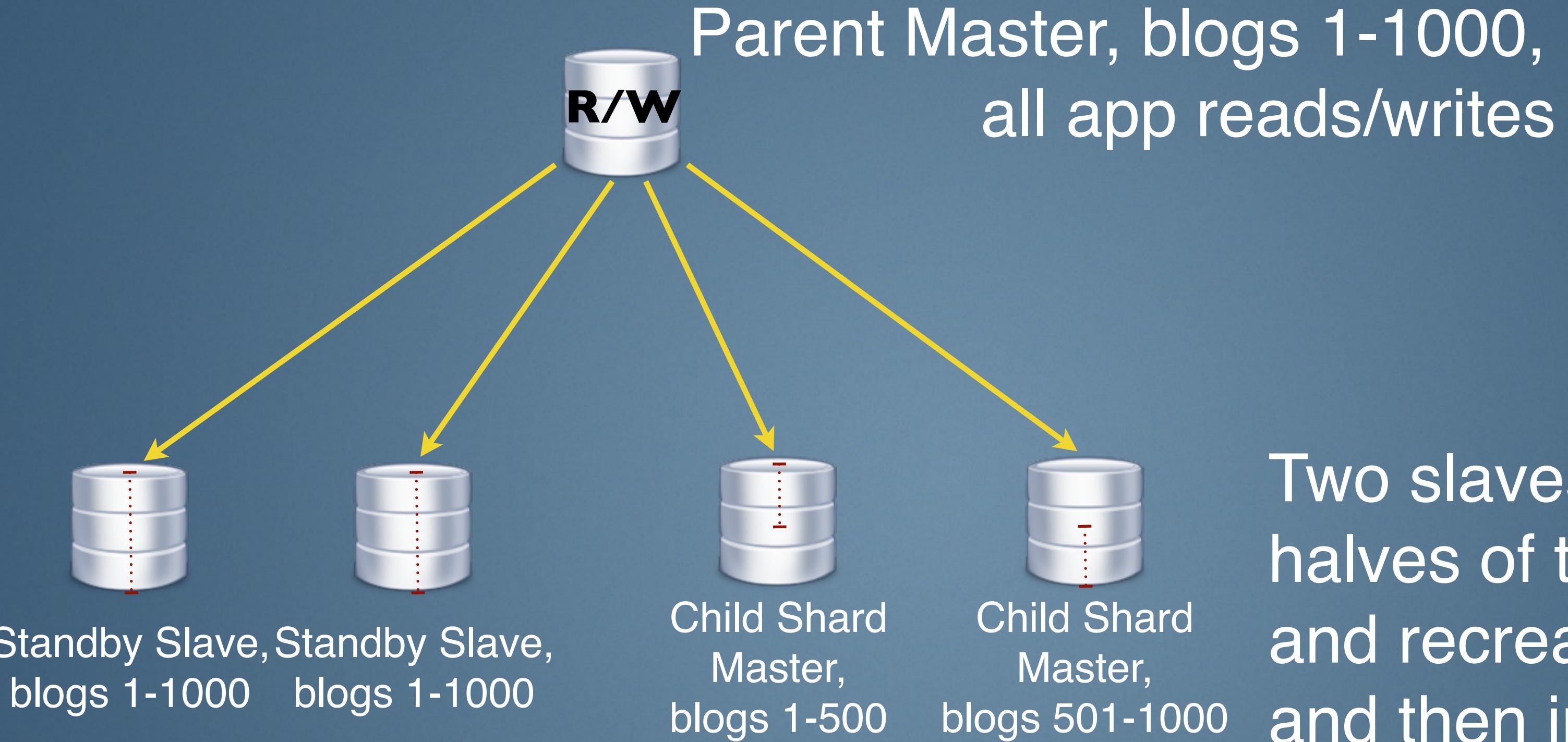
- Rebalance an overly-large shard by dividing it into N new shards, of even or uneven size
- Speed
- No locks
- No application logic
- Divide a billion-row shard in only a few hours
- Full read and write availability: shard-splitting process has no impact on live application performance, functionality, or data consistency, since it isn't performed “in place”



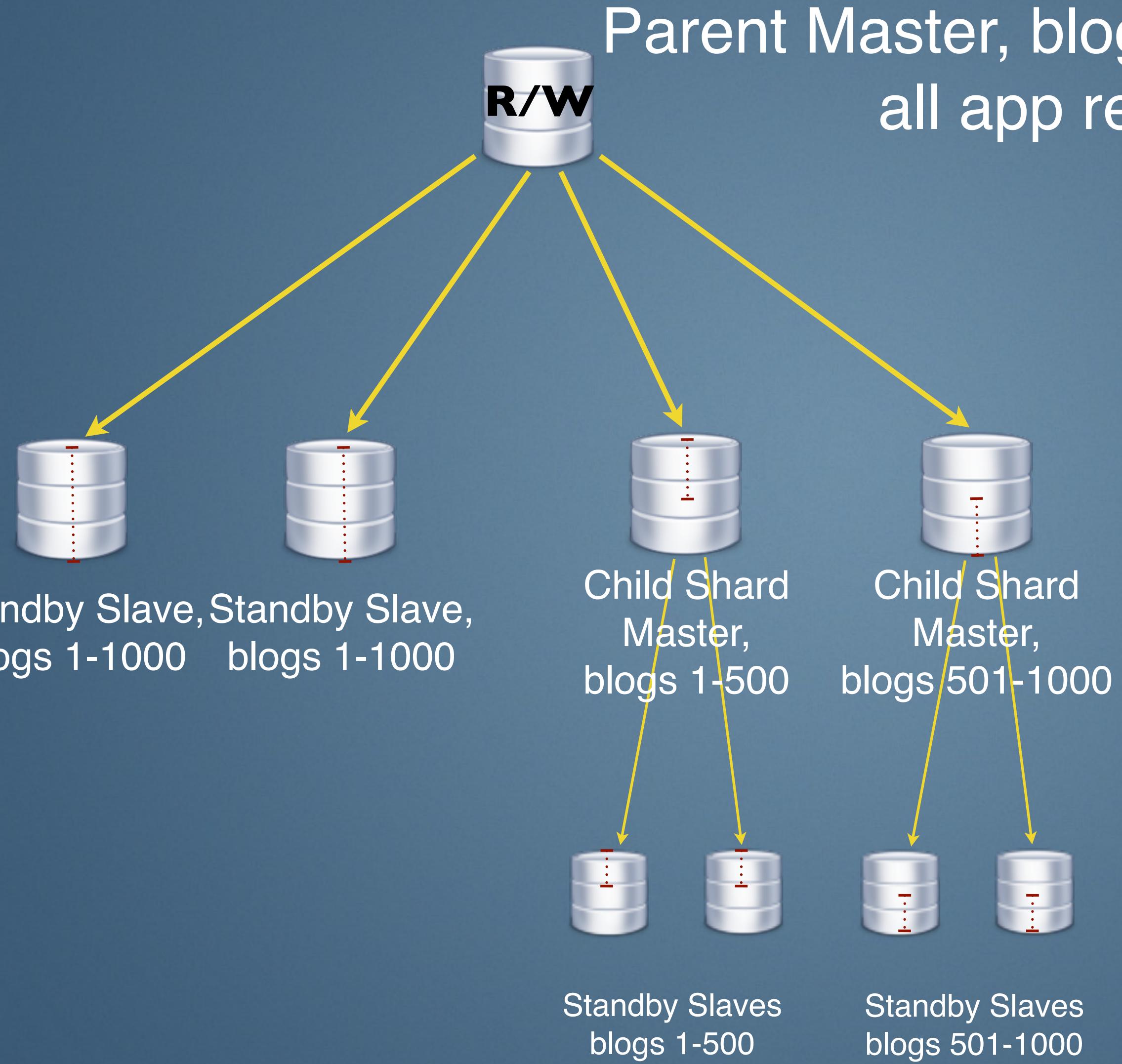
Parent Master, blogs 1-1000,  
all app reads/writes

Standby Slave, Standby Slave,  
blogs 1-1000    blogs 1-1000



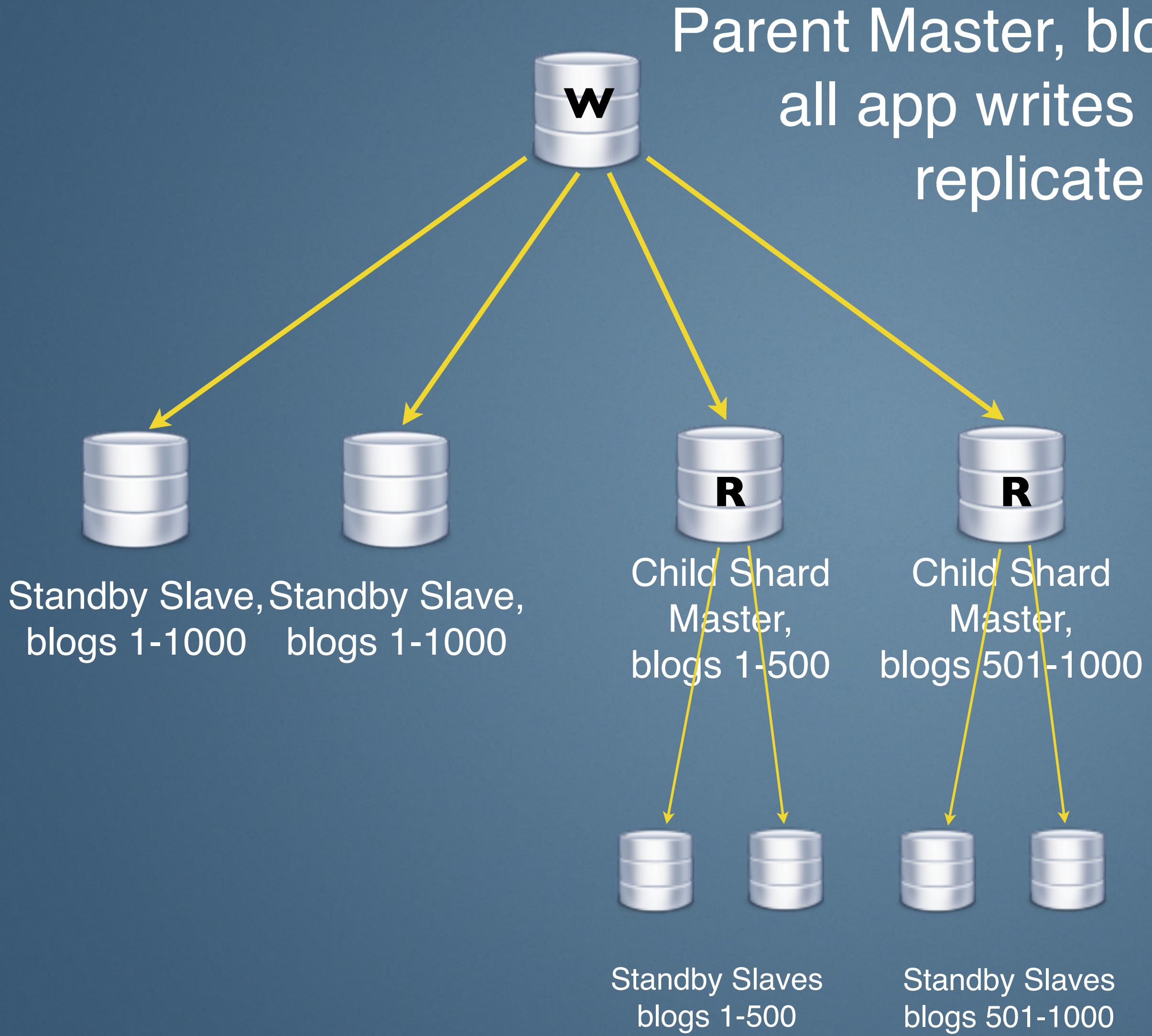


Two slaves export different  
halves of the dataset, drop  
and recreate their tables,  
and then import the data



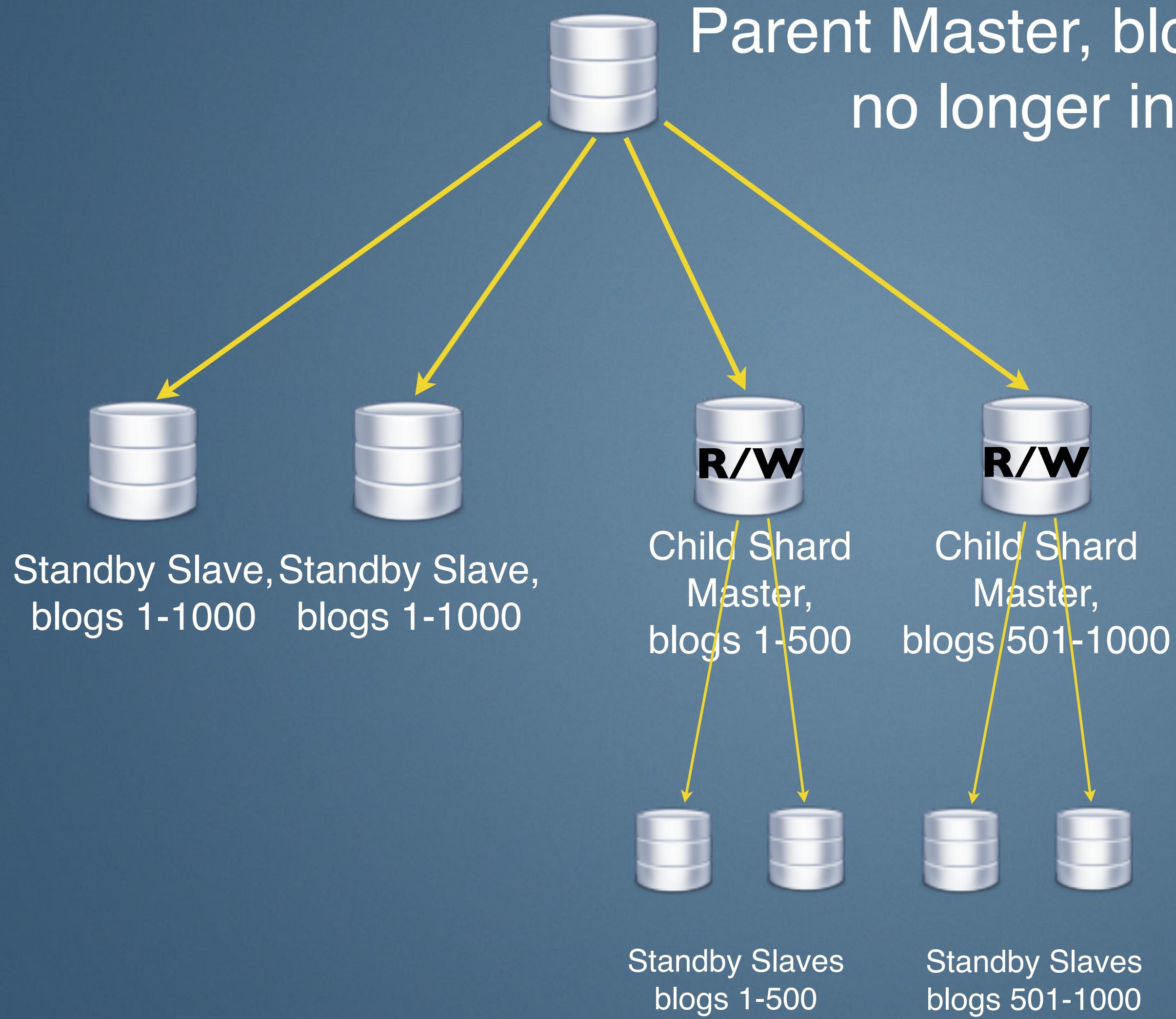
Parent Master, blogs 1-1000,  
all app reads/writes

Every pool needs two  
standby slaves, so spin  
them up for the child  
shard pools



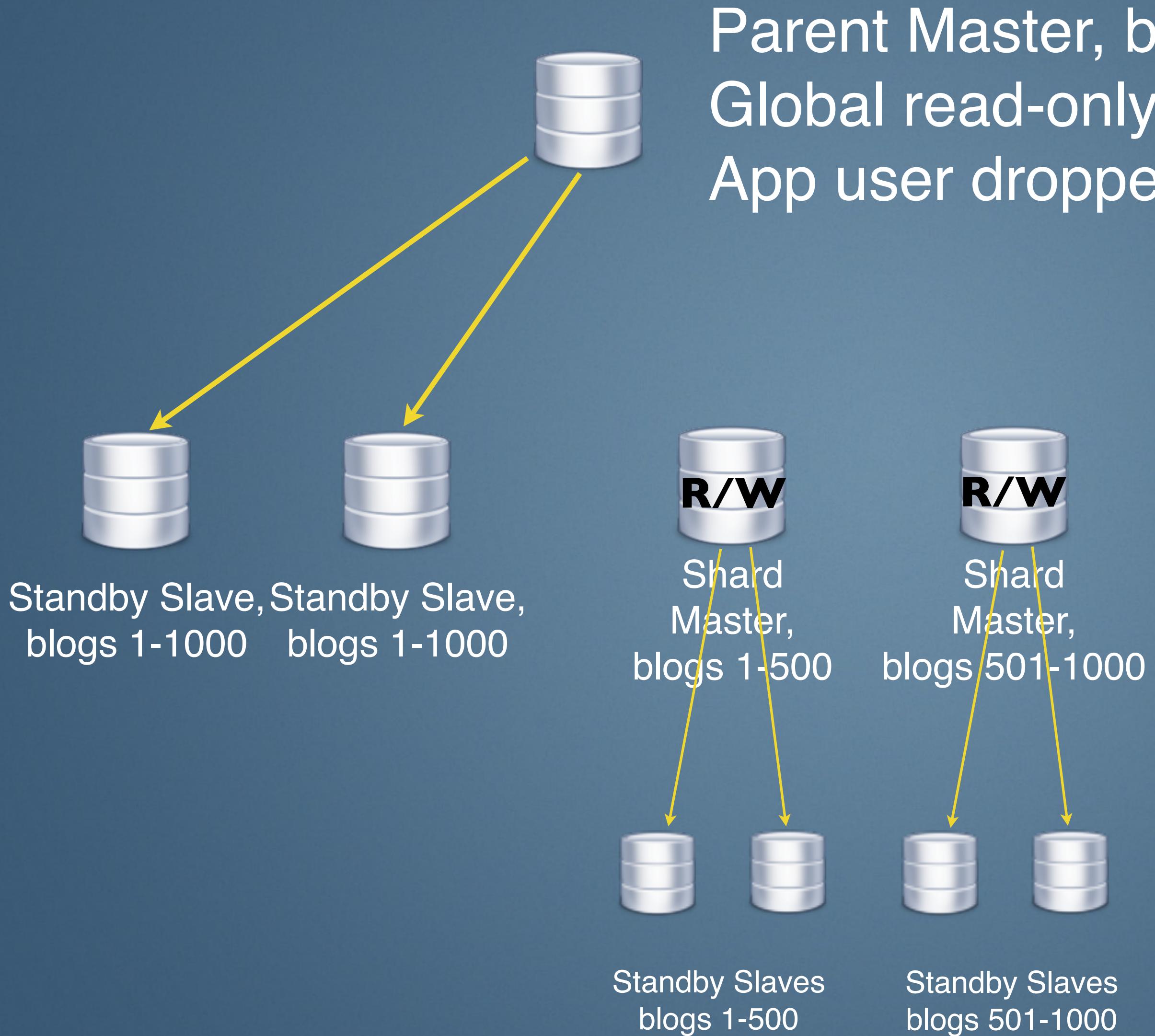
Parent Master, blogs 1-1000,  
all app writes (which then  
replicate to children)

Reads now go to children  
for appropriate queries to  
these ranges



Parent Master, blogs 1-1000,  
no longer in use by app

Reads and writes now go to  
children for appropriate  
queries to these ranges



Parent Master, blogs 1-1000  
Global read-only set  
App user dropped

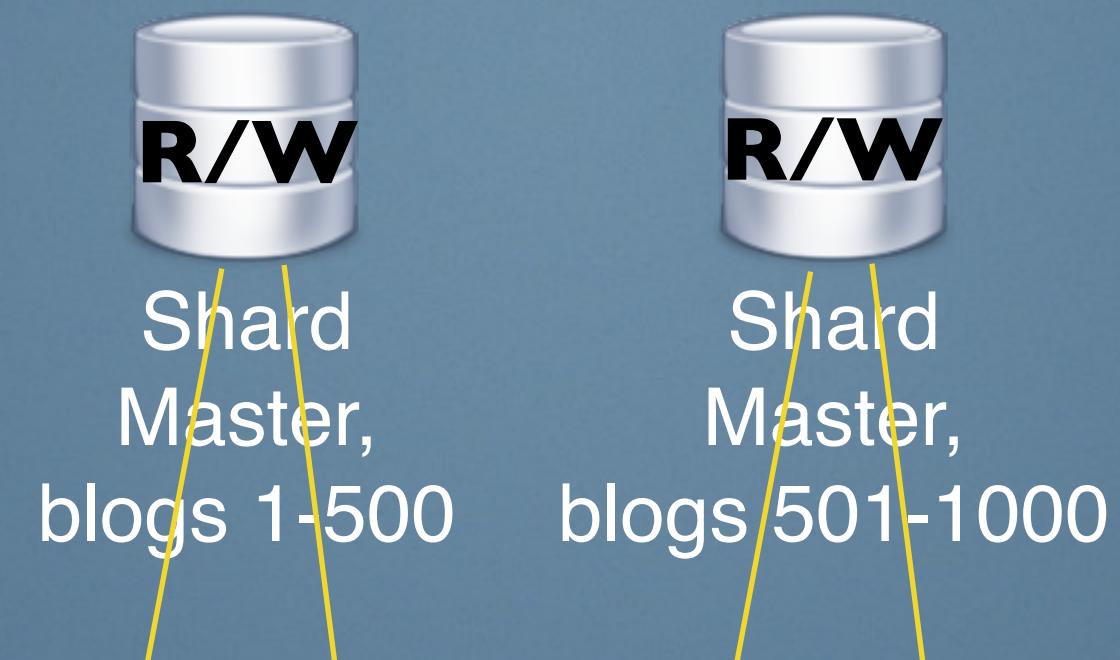
These are now complete  
shards of their own  
(but with some surplus data  
that needs to be removed)



Deprecated Parent Master, blogs 1-1000  
Recycle soon



Standby Slave, Standby Slave,  
blogs 1-1000      blogs 1-1000  
Recycle soon      Recycle soon



Standby Slaves  
blogs 1-500

Standby Slaves  
blogs 501-1000

Clean up data that  
replicated to the wrong  
shard

# Why this matters

---

*"Resharding - I don't recommend it. You don't move users between shards ... From experience I will claim that for a large and busy site, [automating] this is a one-year project."* — Mark Callaghan, Facebook

- Jetpants is open source, so you won't have to spend a year reimplementing this automation
- Instead just spend a few days writing custom plugins to tie into your asset tracker and the rest of your environment
- We will happily merge your changes and patches upstream

# Merging pools

---

## Why?

- Can fix overzealous partitioning
- Hardware has scaled up, SSDs higher capacity now

# Merging pools

---

## How to merge pool A into pool B

1. Export all data on a standby slave of pool A
2. Copy data to a standby slave of pool B, and import there
3. Clone that slave over top all other nodes in pool B (one at a time, swapping roles and promoting as needed)
4. Make the master of B slave from the master of A
5. Move app reads for pool A to pool B
6. Move app writes for pool A to pool B, tear down replication, kill A

# Remote command execution

---

Object model includes hosts, and the ability to run arbitrary commands via SSH

```
# Scan spare pool nodes for failed drives

selector = {operation: 'and', details: true, type:
'SERVER_NODE', primary_role: 'DATABASE', status:
'Provisioned'}

nodes = Plugin::JetCollins.find(selector).map &:to_db

nodes.concurrent_each {|n| n.output n.ssh_cmd '/usr/
StorMan/arccconf getconfig 1 | grep -i degraded'}
```

# Map/Reduce in MySQL?

---

Not exactly, but you can do distributed computation for the “map” step

```
# Calculate a global max
Jetpants.topology.shards.concurrent_map { |s|
  s.query_return_first_value 'SELECT MAX(id) from
  foo' }.max
```

```
# Distributed counts (naive implementation)
Jetpants.topology.shards.concurrent_map { |s|
  s.query_return_first_value 'SELECT COUNT(*) from
  foo' }.reduce(0) { |total, shard_count| total +
  shard_count}
```

# Auto-sharding in MySQL?!?

---

- Hypothetical, and not necessarily recommended
- Would probably be < 500 lines of Ruby code in a cloud environment
- Develop a daemon process using Jetpants as a library
- Monitor shard sizes and peak I/O utilization; use shard#split method on worst offender if over a certain size or load
- Would need ability to automatically deploy app configuration changes
- Would need ability to monitor spare host list size, and spin up new nodes when needed

# Plugin system

---

- Built using Ruby metaprogramming
- Hooks: Plugins can easily insert code before or after any method in any Jetpants class. They "stack", allowing multiple plugins to hook into the same place, with support for specifying priority numbers to resolve conflicts.
- Overrides: Plugins may also completely override any method in any Jetpants class
- Config: Plugins can accept arbitrary configuration data in Jetpants YAML config file
- Plugins should be stand-alone Ruby gems

# Plugins: monitoring

---

- Jetpants DB class has methods `enable_monitoring` and `disable_monitoring`; stock implementation is a no-op
- Called whenever appropriate (MySQL instance stopped, replication stopped, etc)
- Plugin should override these, or use stackable hooks like `DB#before_enable_monitoring` and `DB#before_disable_monitoring`

# Plugins: asset tracker

---

- Jetpants intentionally does not store persistent state on its own
- Most large sites presumably already have a hardware asset tracking system: something that has a list of all hosts, along with their type, role, pool, etc
- This makes more sense as the single point of truth
- So just write a plugin to make Jetpants query this
  - At start-up: discover list of pools, pre-populate Jetpants topology object
  - Upon any topology change from Jetpants: inform the asset tracker appropriately

# Plugins: MHA for promotion logic

---

- MHA is a master promotion tool that solves slave consistency issues by diff'ing the binlogs
- Manual failover: Jetpants plugin could override the Pool#master\_promotion method to figure out which slave has highest binlog coordinates, shell out to masterha\_master\_switch with --new\_master\_host param specifying that slave, and then update topology on Jetpants side
- Automatic failover: Not recommended in general, but could run MHA manager as usual. Write scripts that notify Jetpants about topology changes via MHA's master\_ip\_failover\_script and master\_ip\_online\_change\_script config params

# Jetpants Roadmap

---

- Nonstandard ports and multi-instance-per-host support
- Support more environments (MySQL versions, Linux distros, JRuby)
- Collins asset tracker integration plugin
- Support master-master and other nonstandard replication configurations
- Functional tests
- Configurable/pluggable compression and file transfer

# Questions?

Email: [me@evanelias.com](mailto:me@evanelias.com)