Introduction to Sharding with Jetpants

Evan Elias

Percona Live MySQL Conference 2013

What We'll Cover

Jetpants intro
What is it, and what can it do?

Sharding fundamentals
Range-based sharding pros/cons

Practicalities
Jetpants installation, configuration, integration, and usage

Jetpants internals
What is the tool actually doing?

About Jetpants

MySQL management suite

- Command-line toolkit
- Ruby automation library
- Interact with many database instances at once
- Focus on moving lots of data quickly

Developed by Tumblr in NYC

Open sourced in June 2012
http://github.com/tumblr/jetpants

Why did we build this?

A lot of data. Not a lot of engineers.

Tumblr's MySQL footprint, April 2013:

- 41 TB relational data on masters
- 179 billion distinct rows
- 213 dedicated database servers running MySQL
- 4 engineers with solid MySQL expertise (out of 75 total eng)

In the past year, tripled our amount of relational data

No net increase in database server count

Using Jetpants for Partitioning

Built-in support for two partitioning schemes:

- Functional partitioning
- Dynamic range-based sharding

Other approaches not supported by Jetpants:

- Hash / modulo based sharding
- Lookup table sharding

Functional Partitioning

- Each pool (1 master + N slaves) contains a different set of unsharded tables
- Good for global access patterns / unshardable data
- Scale reads by adding more read slaves
- Scale writes by moving more tables to separate partitions
- Use sparingly!
  - Usually creates SPOFs
  - Read slaves = higher redundancy = higher cost

Sharding

- Partitioning a data set horizontally in the application layer
- Each shard has the same set of tables, but a different portion of the data set
- Motivations: scaling writes, capacity limitations, eliminating single points of failure

Range-based sharding

- Each shard contains data for a different [min_id, max_id] range of the sharding key
- Ranges may be of even or uneven sizes
- Easy to express the shard mapping in a language-neutral fashion

Dynamic range-based sharding

- Shard ranges can change = data can move
- Shards may be split into N new shards, each covering a different portion of the original shard's range. May be divided evenly or unevenly.
- Avoid doing splits in-place!
  - Create N new shards that eventually replace the old shard
  - Permits fast, non-disruptive splits via bulk data loading and replication trickery

Benefits of this approach

- No pre-allocation of thousands of small shards — instead just create new shards on-demand as needed

- Lower total shard count = much smaller range-to-shard mapping

- Shard split process defragments the data set

Drawbacks

- Potential for uneven distribution of data
- Lower average usage of disk capacity, unless putting multiple shards on a machine
- More moving parts

Sharding Decisions

- Sharding key: often user_id or similar
- Sharding scheme: range vs hash vs lookup table
- Growth strategy: what to do when hitting capacity or I/O limitations
- Colocation: multiple shards per instance? multiple instances per machine?

Sharding Decisions with Jetpants

- ☑ Sharding key: configurable
- ☑ Sharding scheme: range-based
- ☑ Growth strategy: split shards on-demand
  Can also “cap” last shard and add a new one after it
- ☑ Colocation: not yet supported
  1 shard = 1 server = 1 mysqld instance = 1 database
  Results in “fat” shards, but less complexity

Sharding Checklist

- [ ] ID generation
- [ ] Query routing in application or proxy
- [ ] Failure handling in application
- [ ] General-purpose management / tooling / automation
- [ ] Growth solution

Sharding Checklist with Jetpants

- ☐ ID generation
- ☐ Query routing in application or proxy
- ☐ Failure handling in application
- ☑ General-purpose management / tooling / automation
- ☑ Growth solution

Jetpants installation

- Requires Ruby 1.9.2+
- Easiest installation: `gem install jetpants`
- Gives you the command-line tool plus Ruby library, and all Ruby dependencies
- May need to install some system packages first (especially mysql-devel or equivalent)

Configuration

- Create /etc/jetpants.yaml or ~/.jetpants.yaml
- See docs and example config file in repo

evan — ssh — 80×27

```
export_location: /export
max_concurrency: 12
mysql_grant_privs:
    - SELECT
    - INSERT
    - UPDATE
    - DELETE
    - CREATE TEMPORARY TABLES

compress_with: qpress -ioT4K32768 dummyfilename
decompress_with: qpress -dioT4

sharded_tables:
    posts:
        sharding_key: tumblelog_id
        chunks: 500
    tracked_tags:
        sharding_key: tumblelog_id
        chunks: 10
    likes:
        sharding_key: from_tumblelog_id
        chunks: 200
    notes:
        sharding_key: root_tumblelog_id
        chunks: 300
    custom_themes:
```

Integration

- Jetpants intentionally does not maintain state persistently
- Integrate it with your existing asset tracker
- Takes some effort, but low risk / not too scary. (Jetpants is not a proxy / middleware server, storage engine, replication replacement, etc...)
- Plugin system permits further integration with your existing monitoring, trending, configuration management, app framework config generation

MySQL assumptions

- MySQL / Percona Server 5.1 or 5.5. Soon 5.6.
- InnoDB / Percona XtraDB — clustered index is vital
- First col of PK of each sharded table is the sharding key
- No auto_increment on shards
- One mysqld per server, running on 3306
- One application database per mysqld
- No master-master, filtered slaves, rings, other insanity

Using the command suite

- Run `jetpants` with no params to see help / list of commands
- Over a dozen commands in base
- Plugins can add additional commands
- Most commands take args, but will prompt you interactively if not supplied

evan — ssh — 119×25

```
[evan@dev-evan-22fec837:~]$ jetpants
Tasks:
  jetpants activate_slave           # turn a standby slave into an active slave
  jetpants clone_slave              # clone a standby slave
  jetpants console                  # Jetpants interactive console
  jetpants defrag_slave             # export and re-import data set on a standby slave
  jetpants destroy_slave            # remove a standby slave from its pool
  jetpants help [TASK]              # Describe available tasks or one specific task
  jetpants pools                    # display a full summary of every pool in the topology
  jetpants pools_compact            # display a compact summary (master, name, and size) of every pool in the topology
  jetpants promotion                # perform a master promotion, changing which node is the master of a pool
  jetpants pull_slave               # turn an active slave into a standby slave
  jetpants regen_config             # regenerate the application configuration
  jetpants shard_cutover            # truncate the current last shard range, and add a new shard after it
  jetpants shard_offline            # mark a shard as offline (not readable or writable)
  jetpants shard_online             # mark a shard as fully online (readable and writable)
  jetpants shard_read_only          # mark a shard as read-only
  jetpants shard_split              # shard split step 1 of 4: spin up child pools with different portions of data set
  jetpants shard_split_child_reads  # shard split step 2 of 4: move reads to child shards
  jetpants shard_split_child_writes # shard split step 3 of 4: move writes to child shards
  jetpants shard_split_cleanup      # shard split step 4 of 4: clean up data that replicated to wrong shard
  jetpants summary                  # display information about a node in the context of its pool
  jetpants weigh_slave              # change the weight of an active slave

[evan@dev-evan-22fec837:~]$
```

Most useful commands

- `jetpants pools`
- `jetpants clone_slave`
- `jetpants promotion`
- `jetpants shard_split`
- `jetpants console`

Internals

- Jetpants core is about 4k lines of Ruby
- Bundled plugins are about 1k more
- What's going on under the hood?

Design Principles

- Pluggability
- Concurrency
- Composability

Pluggability

- Allow plugins to override any method
- Hook into call chain before/after any method
- Plugin system made open source effort feasible
- Override core functionality if desired: replace master promotion logic with MHA, replace slave-cloning method with XtraBackup, etc

Concurrency

- Perform actions concurrently whenever possible
- Speed benefits, especially in import/export on SSDs
- Many commands interact with several nodes simultaneously
- Or manually query multiple DBs at once in jetpants console

Composability

- Implement reusable low-level components:
  - Replication discovery / manipulation
  - Fast file copy
  - State probing
  - Parallelized bulk data import/export
- Combine them to build higher-level functionality:
  - Clone slaves
  - Defragment nodes
  - Split shards

How Jetpants clones replicas

- Stop mysqld and perform physical copy — much faster than a hot copy, but requires standby slave
- Uses `tar`, `nc`, and optionally a compression program of your choice (we recommend `qpress` or `pigz`)
- Can automatically "chain" to multiple destinations by using `tee` and a fifo on intermediate nodes

How Jetpants exports data

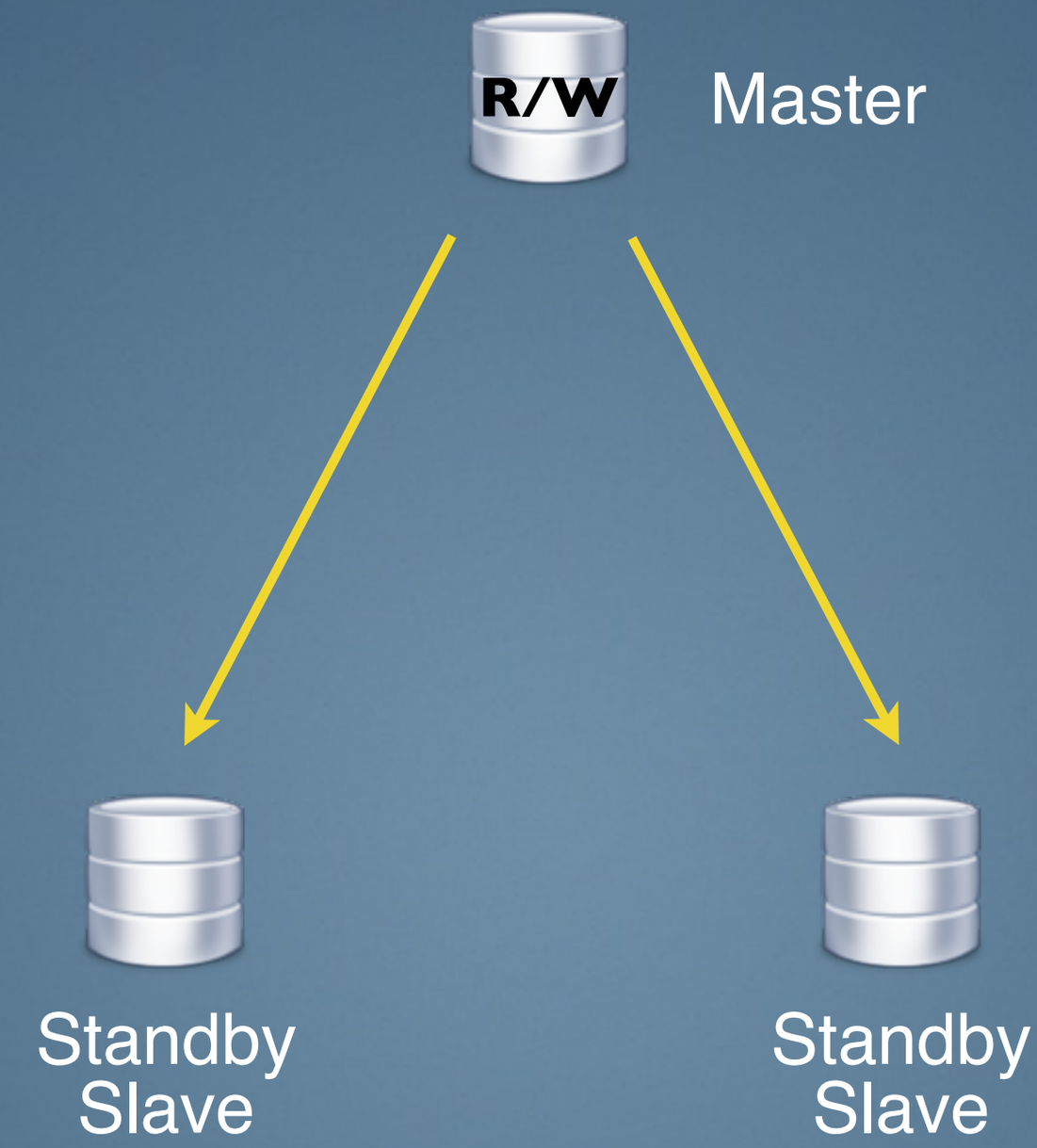
- Create a temporary mysql user with FILE privileges
- Conceptually divide primary key range into N chunks (tens, hundreds, or thousands — depends on table size)
- Spawn up M threads in Jetpants, where M ≤ N
- Each thread pulls a chunk range off the queue, does a SELECT ... INTO OUTFILE query with WHERE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

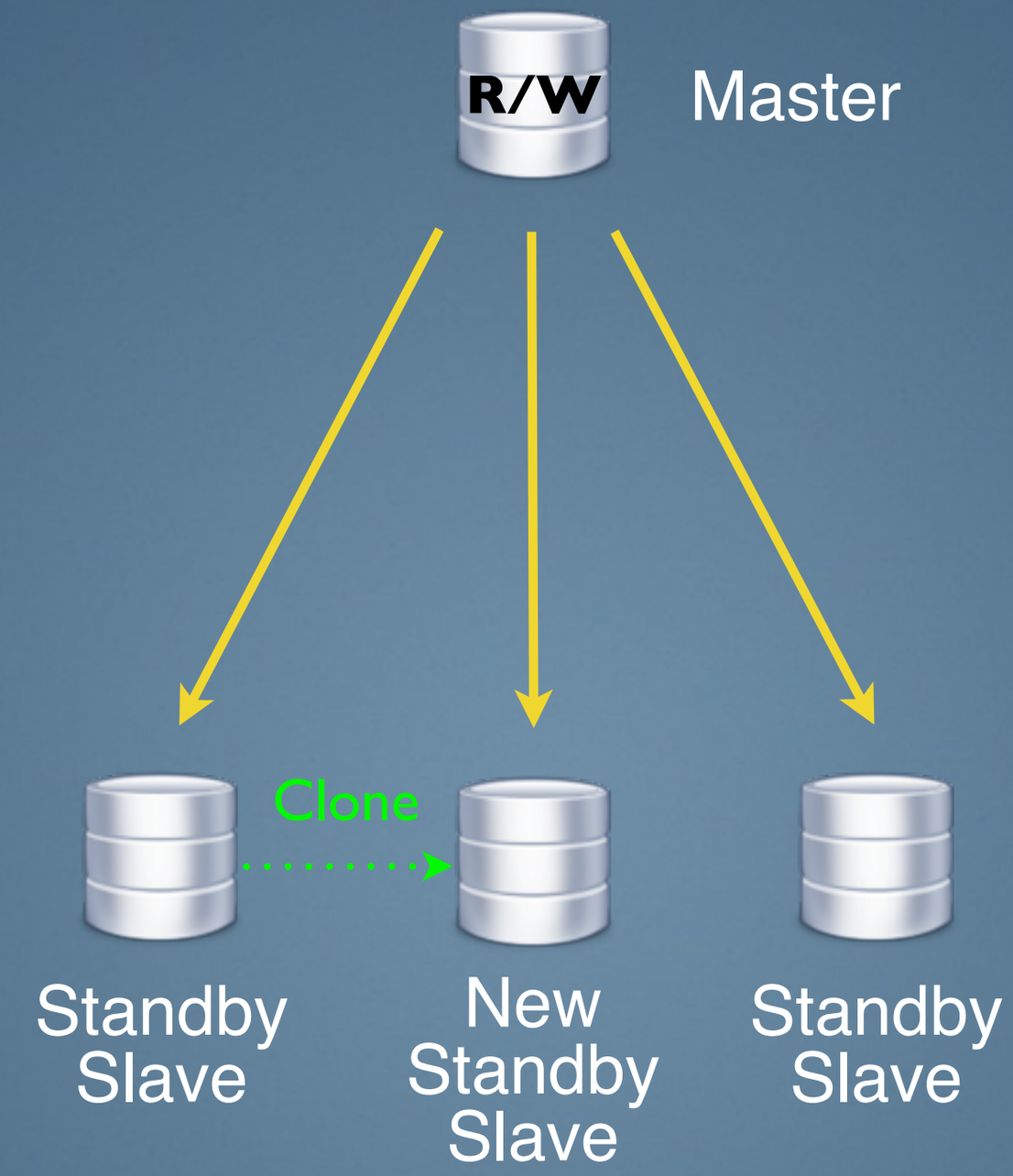
How Jetpants imports data

- 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
- Also temporarily sets innodb-autoinc-lock-mode = 2, permits import of auto_increment data (unsharded only)
- Upon connect, each thread runs SET unique_checks = 0
- To do: defer secondary index creation until after import

Ejecting a shard master

- Do a master promotion in multiple steps, without a read_only or lock period
- Key insight: first make all app servers start reading from the new master. Once complete, now make writes go to the new master. Avoids inconsistency issues.
- Assumptions
  - No auto_increment
  - Take care with unique keys
  - Very little slave lag between old master and new master


R/W
Master
Standby
Slave
Standby
Slave


R/W
Master
Clone
Standby
Slave
New
Standby
Slave
Standby
Slave

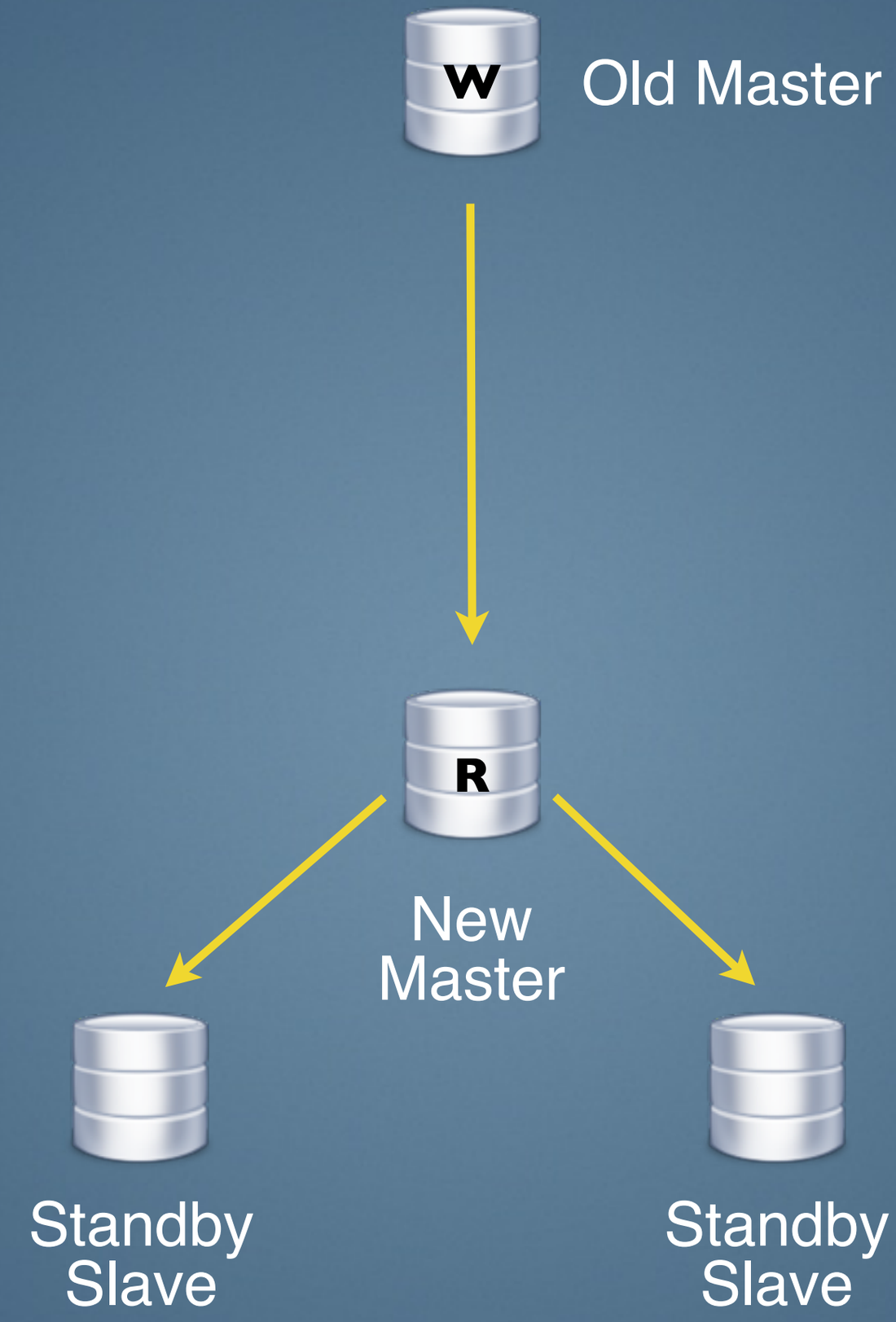
Old Master

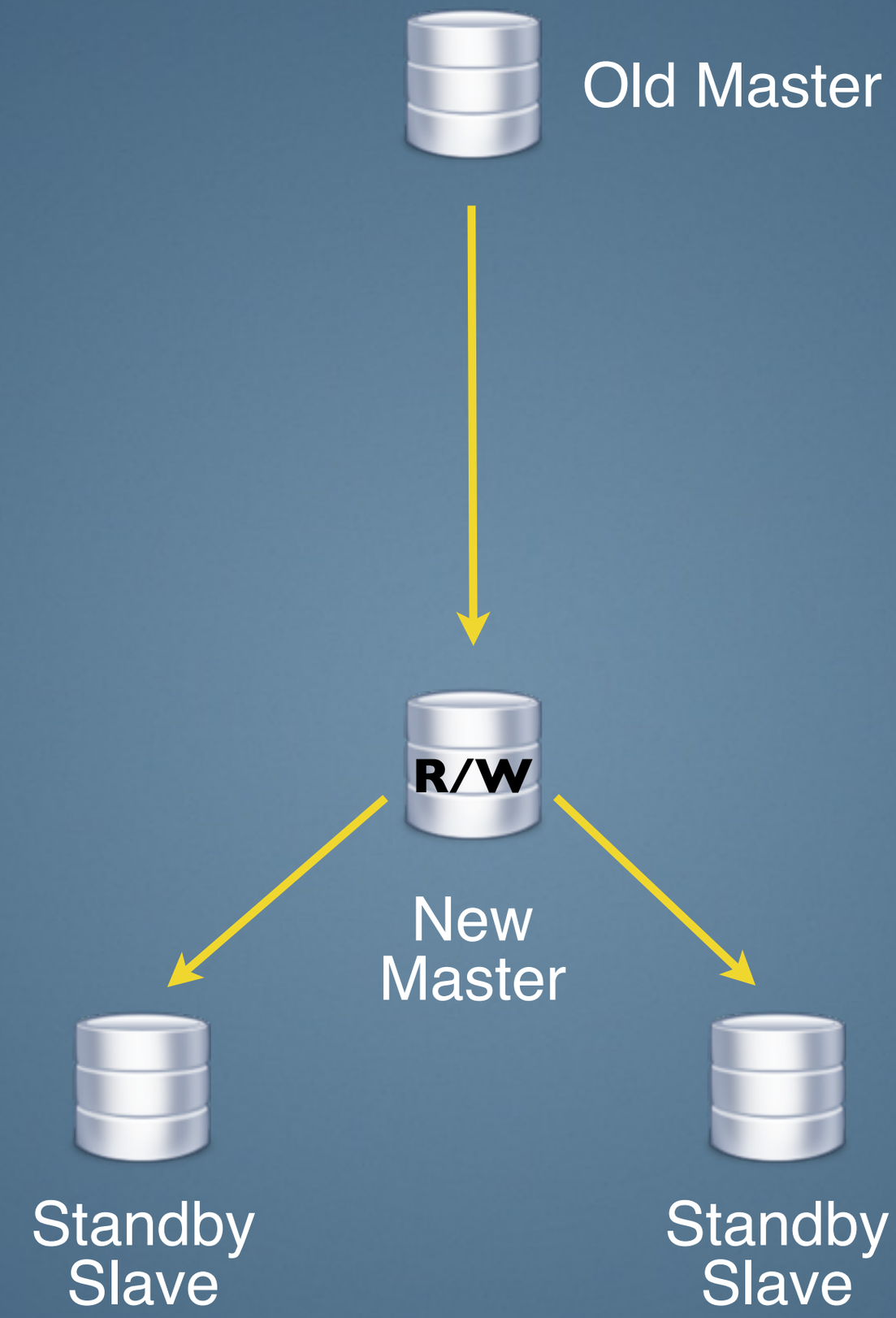
Move old slaves to new master (first use START SLAVE UNTIL to get in same position)

New Master

Standby Slave

Standby Slave


W
Old Master
R
New
Master
Standby
Slave
Standby
Slave


Old Master
R/W
New
Master
Standby
Slave
Standby
Slave

Recycle / Cancel

Master

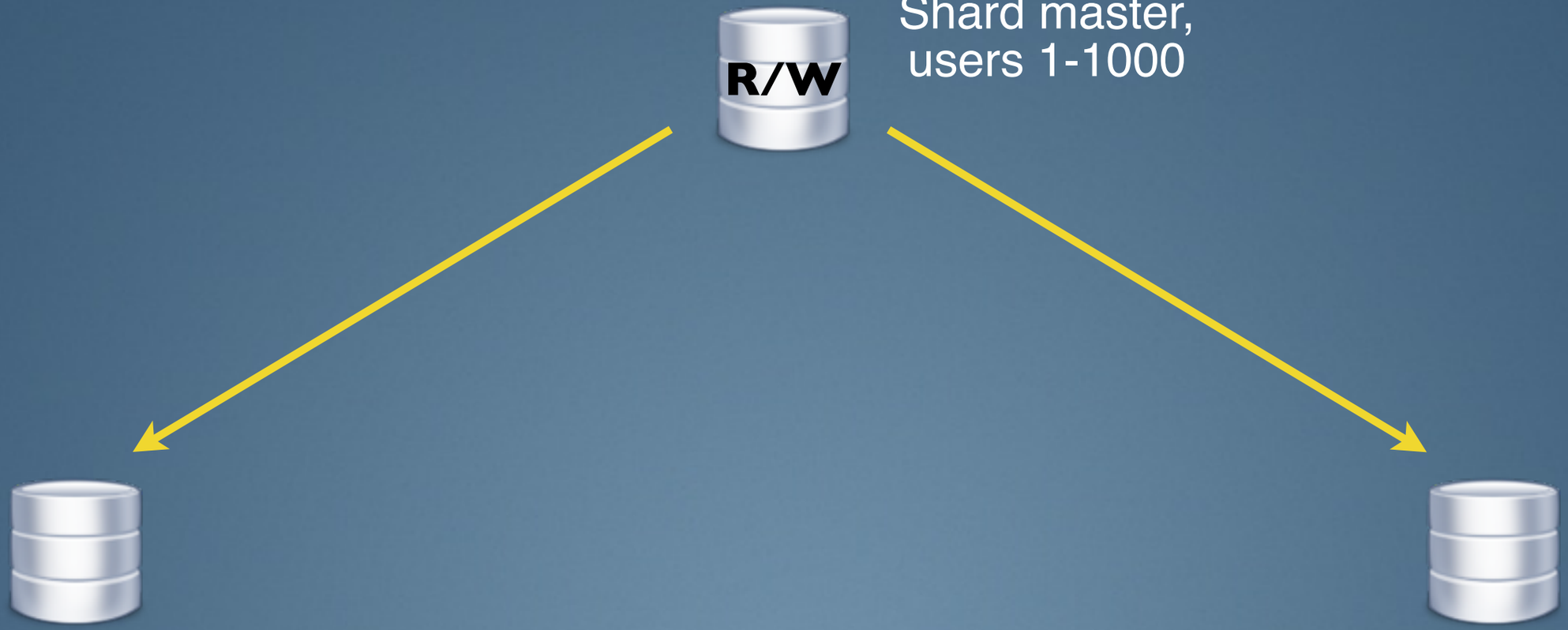
Standby
Slave

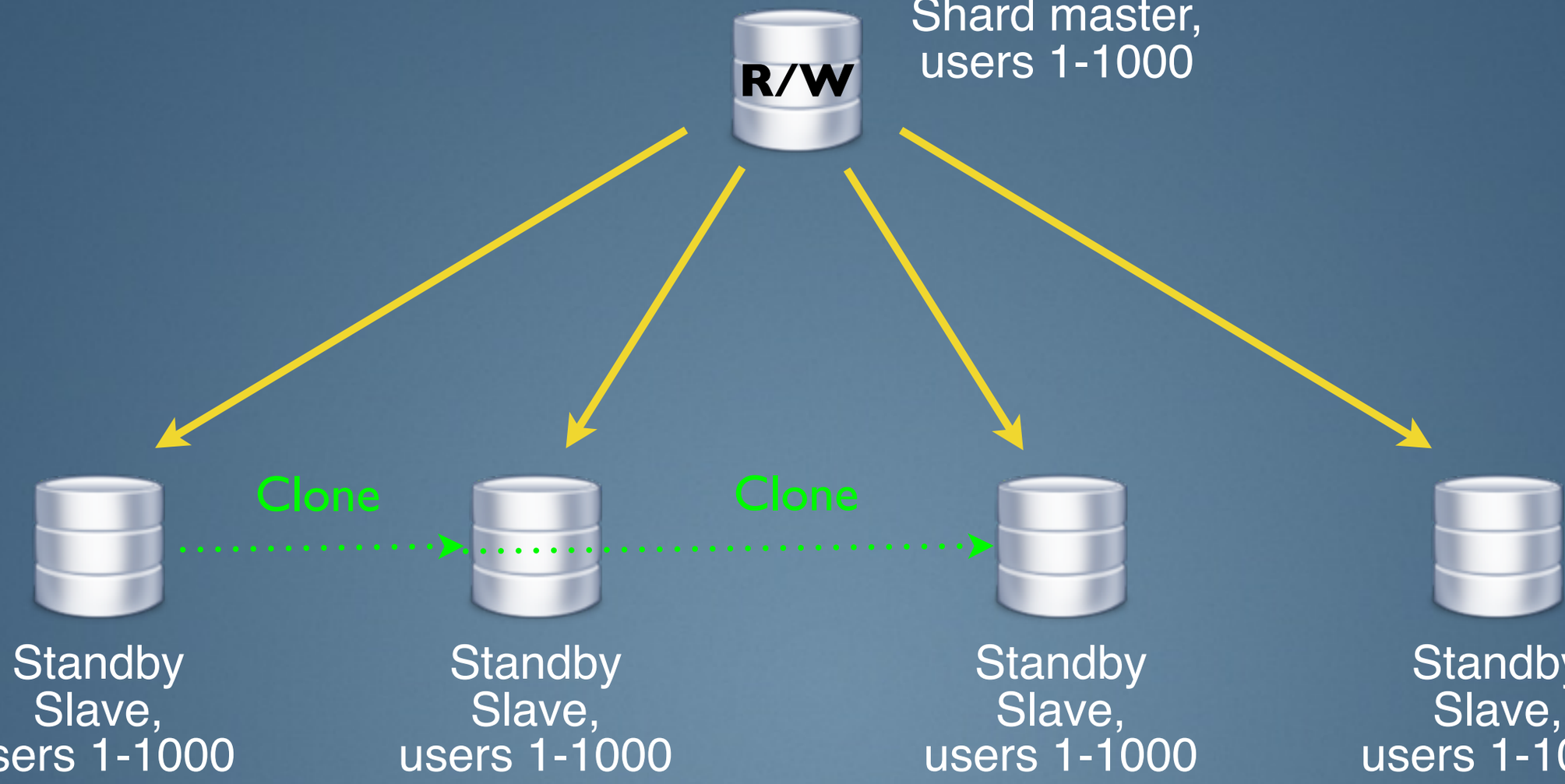
Standby
Slave

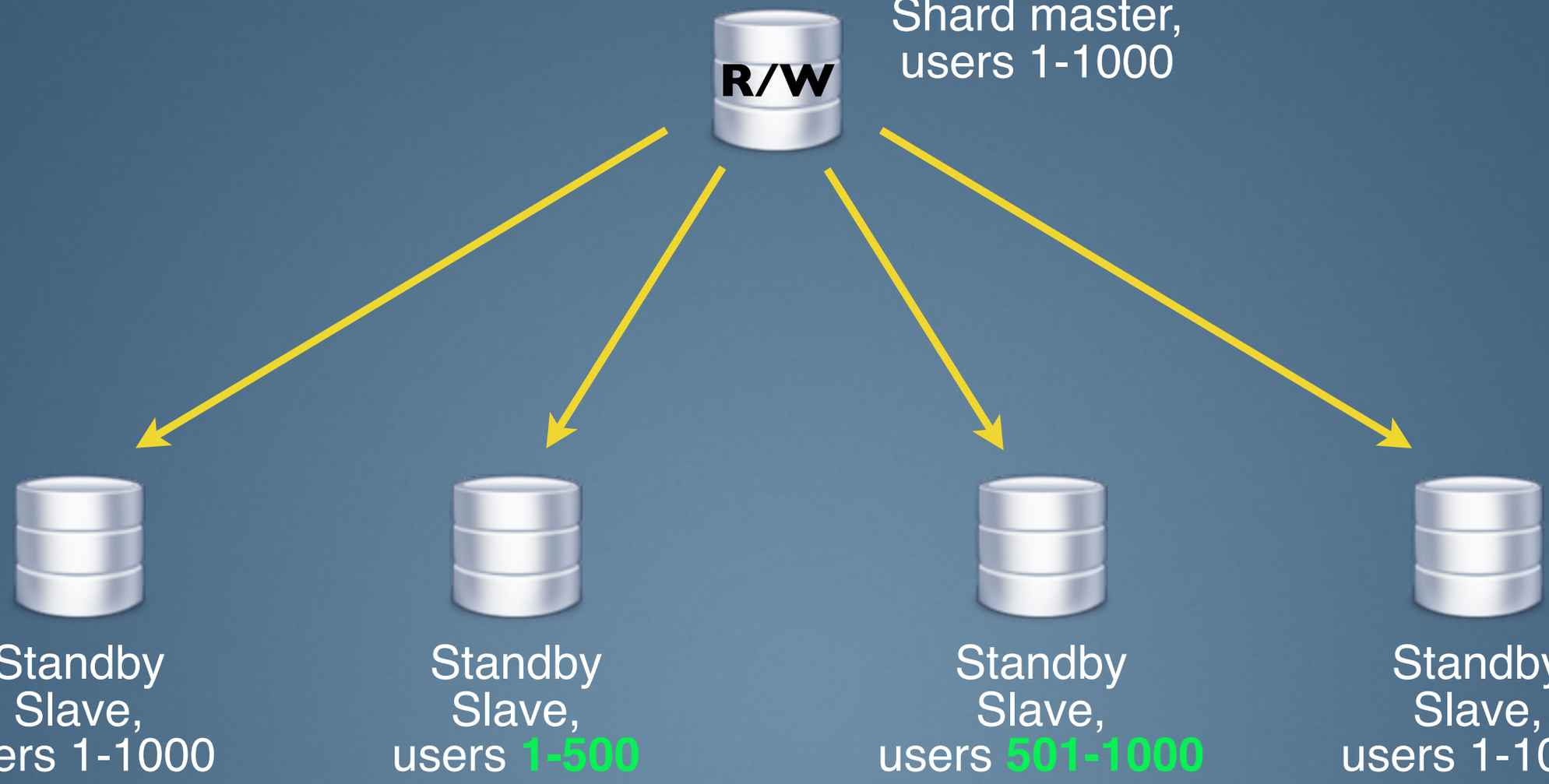
How Jetpants splits shards

Putting all these components together

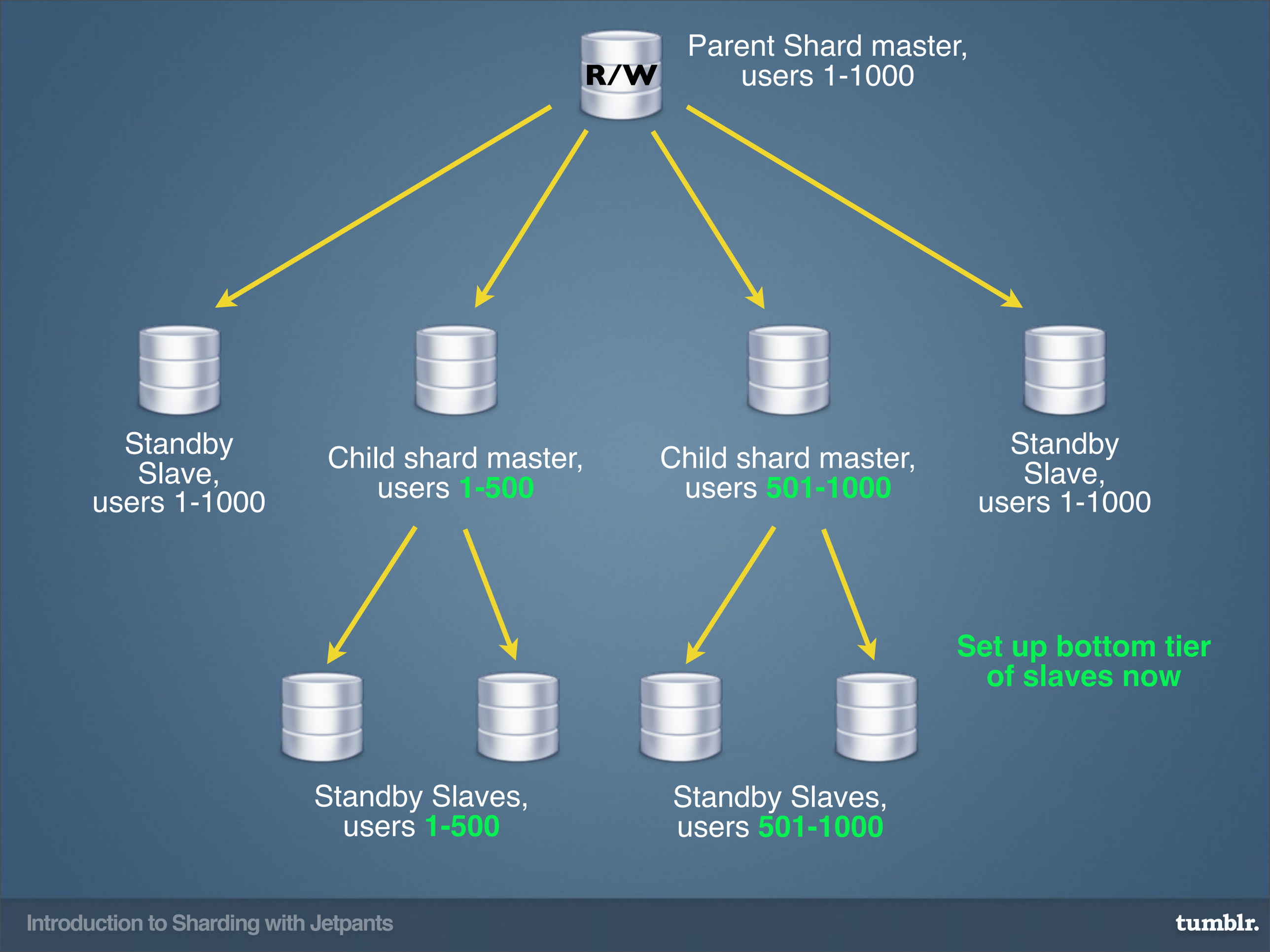
1. Clone a standby slave to N new nodes
2. Prune data set quickly: New slaves export different ranges of data set, drop tables, recreate tables, import data. Also inherently defragments!
3. Eject old shard: move reads to newly-created shards, then move writes, then tear down replication between old and new
4. Clean up recent data that replicated to wrong shard


R/W
Shard master,
users 1-1000
Standby
Slave,
users 1-1000
Standby
Slave,
users 1-1000


R/W
Shard master,
users 1-1000
Clone
Clone
Standby
Slave,
users 1-1000
Standby
Slave,
users 1-1000
Standby
Slave,
users 1-1000
Standby
Slave,
users 1-1000



Prune data set on the two new slaves via partial export then re-import


R/W
Parent Shard master,
users 1-1000
Standby
Slave,
users 1-1000
Child shard master,
users 1-500
Child shard master,
users 501-1000
Standby
Slave,
users 1-1000
Standby Slaves,
users 1-500
Standby Slaves,
users 501-1000
Set up bottom tier
of slaves now

Writes still go to parent shard, replicate down to children

Parent Shard master, users 1-1000

Move reads to the appropriate child shard

Standby Slave, users 1-1000

Child shard master, users 1-500

Child shard master, users 501-1000

Standby Slave, users 1-1000

Standby Slaves, users 1-500

Standby Slaves, users 501-1000

Now no queries go to parent shard

Parent Shard master, users 1-1000

Move writes to the appropriate child shard

Standby Slave, users 1-1000

Child shard master, users 1-500

Child shard master, users 501-1000

Standby Slave, users 1-1000

Standby Slaves, users 1-500

Standby Slaves, users 501-1000

Set read_only and drop app user privs

Recycle / Cancel

Tear down replication from parent shard

Recycle / Cancel

R/W

Child shard master, users 1-500

R/W

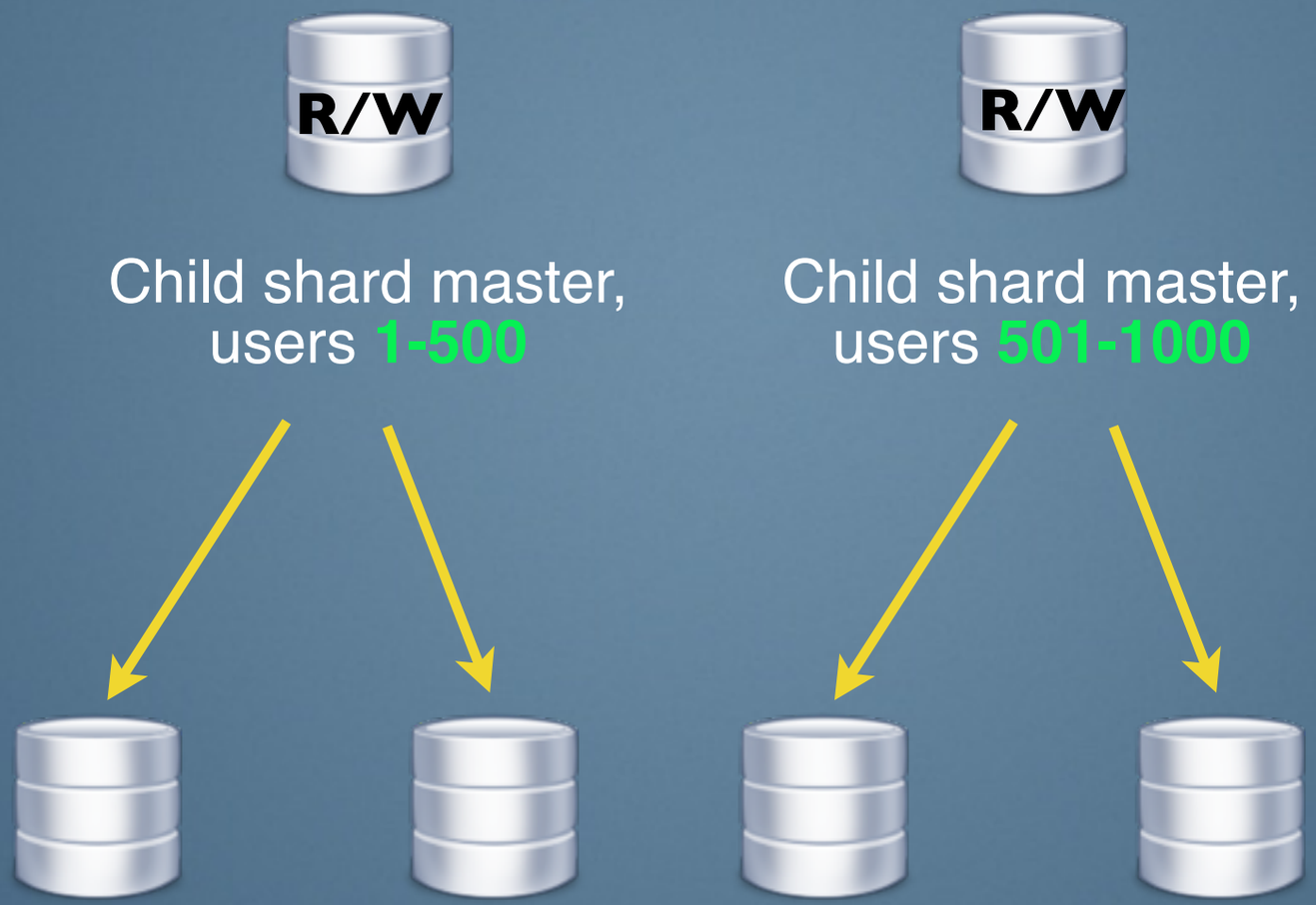
Child shard master, users 501-1000

Recycle / Cancel

Standby Slaves, users 1-500

Standby Slaves, users 501-1000

Final step: clean up data that replicated to wrong shard (iteratively SELECT then DELETE rows outside of shard range)



Coming soon

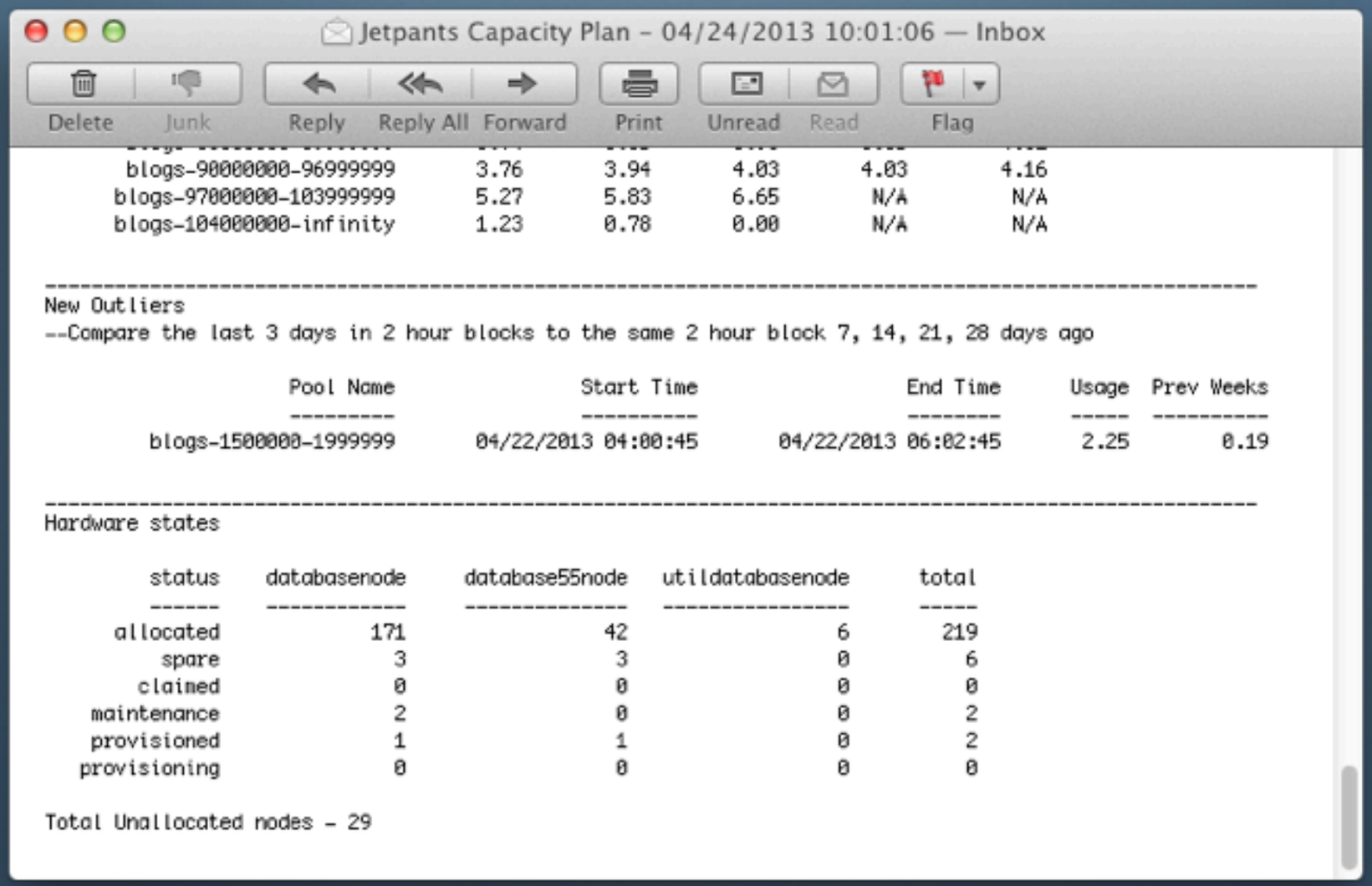
Online schema change plugin: wraps Percona's pt-osc to easily manage a schema change on all shards

Upgrade helper plugin: commands to upgrade a slave, verify data (wraps pt-table-checksum) and performance (wraps pt-upgrade), also perform lockless master promotions on shards

Schema detection for sharded tables: no configuration needed

Coming soon

Capacity planning plugin: calculates growth rates, finds outliers, provides hardware accounting, generates reports



Questions?

Email: me@evanelias.com