tumblr.
Massively Sharded MySQL

Evan Elias

Velocity Europe 2011

Tumblr's Size and Growth

	1 Year Ago	Today
Impressions	**3 Billion/month**	**15 Billion/month**
Total Posts	**1.5 Billion**	**12.5 Billion**
Total Blogs	**9 Million**	**33 Million**
Developers	**3**	**17**
Sys Admins	**1**	**5**
Total Staff (FT)	**13**	**55**

Our databases and dataset

- Machines dedicated to MySQL: over 175
  - That’s roughly how many production machines we had total a year ago
- Relational data on master databases: over 11 terabytes
- Unique rows: over 25 billion

MySQL Replication 101

- Asynchronous
- Single-threaded SQL execution on slave
- Masters can have multiple slaves
- A slave can only have one master
- Can be hierarchical, but complicates failure-handling
- Keep two standby slaves per pool: one to promote when a master fails, and the other to bring up additional slaves quickly
- Scales reads, not writes

Why Partition?

Reason 1: Write scalability

- No other way to scale writes beyond the limits of one machine
- During peak insert times, you'll likely start hitting lag on slaves before your master shows a concurrency problem

Why Partition?

Reason 2: Data size

- Working set won't fit in RAM
- SSD performance drops as disk fills up
- Risk of completely full disk
- Operational difficulties: slow backups, longer to spin up new slaves
- Fault isolation: all of your data in one place = single point of failure affecting all users

Types of Partitioning

- Divide a table
  - Horizontal Partitioning
  - Vertical Partitioning

- Divide a dataset / schema
  - Functional Partitioning

Horizontal Partitioning

Divide a table by relocating sets of rows

- Some support internally by MySQL, allowing you to divide a table into several files transparently, but with limitations
- **Sharding** is the implementation of horizontal partitioning outside of MySQL (at the application level or service level). Each partition is a separate table. They may be located in different database schemas and/or different instances of MySQL.

Vertical Partitioning

Divide a table by relocating sets of columns

- Not supported internally by MySQL, though you can do it manually by creating separate tables.
- Not recommended in most cases – if your data is already normalized, then vertical partitioning introduces unnecessary joins
- If your partitions are on different MySQL instances, then you're doing these "joins" in application code instead of in SQL

Functional Partitioning

Divide a dataset by moving one or more tables

- First eliminate all JOINs across tables in different partitions
- Move tables to new partitions (separate MySQL instances) using selective dumping, followed by replication filters
- Often just a temporary solution. If the table eventually grows too large to fit on a single machine, you'll need to shard it anyway.

When to Shard

- Sharding is very complex, so it's best not to shard until it's obvious that you will actually need to!
- Predict when you will hit write scalability issues — determine this on spare hardware
- Predict when you will hit data size issues — calculate based on your growth rate
- Functional partitioning can buy time

Sharding Decisions

- Sharding key — a core column present (or derivable) in most tables.
- Sharding scheme — how you will group and home data (ranges vs hash vs lookup table)
- How many shards to start with, or equivalently, how much data per shard
- Shard colocation — do shards coexist within a DB schema, a MySQL instance, or a physical machine?

Sharding Schemes

Determining which shard a row lives on

- **Ranges**: Easy to implement and trivial to add new shards, but requires frequent and uneven rebalancing due to user behavior differences.
- **Hash or modulus**: Apply function on the sharding key to determine which shard. Simple to implement, and distributes data evenly. Incredibly difficult to add new shards or rebalance existing ones.
- **Lookup table**: Highest flexibility, but impacts performance and adds a single point of failure. Lookup table may eventually become too large.

Application Requirements

- Sharding key must be available for all frequent look-up operations. For example, can’t efficiently look up posts by their own ID anymore, also need blog ID to know which shard to hit.
- Support for read-only and offline shards. App code needs to gracefully handle planned maintenance and unexpected failures.
- Support for reading and writing to different MySQL instances for the same shard range — not for scaling reads, but for the rebalancing process

Service Requirements

- ID generation for PK of sharded tables
- Nice-to-have: Centralized service for handling common needs
  - Querying multiple shards simultaneously
  - Persistent connections
  - Centralized failure handling
  - Parsing SQL to determine which shard(s) to send a query to

Operational Requirements

- Automation for adding and rebalancing shards, and sufficient monitoring to know when each is necessary
- Nice-to-have: Support for multiple MySQL instances per machine — makes cloning and replication setup simpler, and overhead isn't too bad

How to initially shard a table

Option 1: Transitional migration with legacy DB

- Choose a cutoff ID of the table's PK (not the sharding key) which is slightly higher than its current max ID. Once that cutoff has been reached, all new rows get written to shards instead of legacy.
- Whenever a legacy row is updated by app, move it to a shard
- Migration script slowly saves old rows (at the app level) in the background, moving them to shards, and gradually lowers cutoff ID
- Reads may need to check shards and legacy, but based on ID you can make an informed choice of which to check first

How to initially shard a table

Option 2: All at once

1. **Dark mode**: app redundantly sends all writes (inserts, updates, deletes) to legacy database as well as the appropriate shard. All reads still go to legacy database.

2. **Migration**: script reads data from legacy DB (sweeping by the sharding key) and writes it to the appropriate shard.

3. **Finalize**: move reads to shards, and then stop writing data to legacy.

Shard Automation

Tumblr's custom automation software can:

- Crawl replication topology for all shards
- Manipulate server settings or concurrently execute arbitrary UNIX commands / administrative MySQL queries, on some or all shards
- Copy large files to multiple remote destinations efficiently
- Spin up multiple new slaves simultaneously from a single source
- Import or export arbitrary portions of the dataset
- Split a shard into N new shards

Splitting shards: goals

- 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 800gb shard (hundreds of millions of rows) in two in only 5 hours
- Full read and write availability: shard-splitting process has no impact on live application performance, functionality, or data consistency

Splitting shards: assumptions

- All tables using InnoDB
- All tables have an index that begins with your sharding key, and sharding scheme is range-based. This plays nice with range queries in MySQL.
- No schema changes in process of split
- Disk is < 2/3 full, or there's a second disk with sufficient space
- Keeping two standby slaves per shard pool (or more if multiple data centers)
- Uniform MySQL config between masters and slaves: log-slave-updates, unique server-id, generic log-bin and relay-log names, replication user/grants everywhere
- No real slave lag, or already solved in app code
- Redundant rows temporarily on the wrong shard don't matter to app

Splitting shards: process

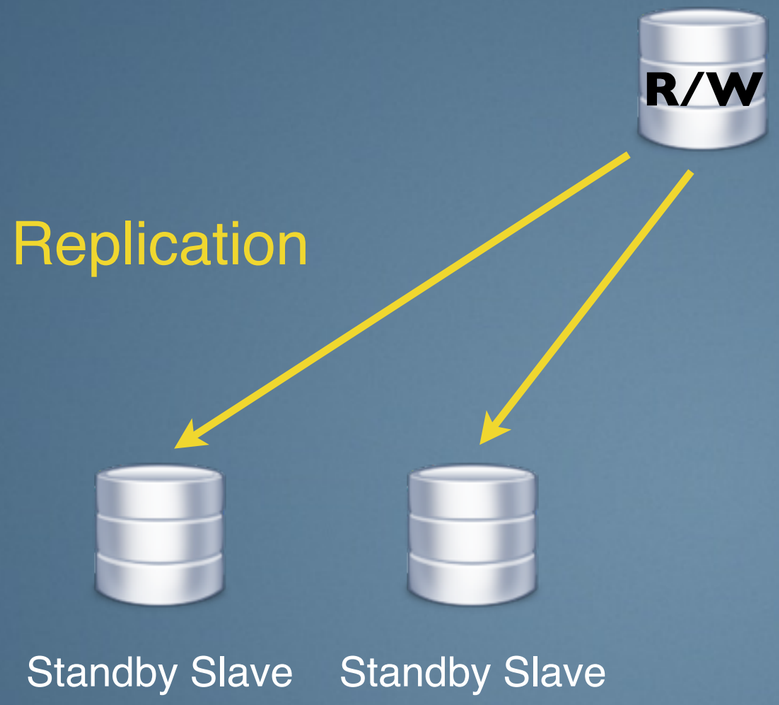
Large “parent” shard divided into N “child” shards

1. Create N new slaves in parent shard pool — these will soon become masters of their own shard pools
2. Reduce the data set on those slaves so that each contains a different subset of the data
3. Move app reads from the parent to the appropriate children
4. Move app writes from the parent to the appropriate children
5. Stop replicating writes from the parent; take the parent pool offline
6. Remove rows that replicated to the wrong child shard

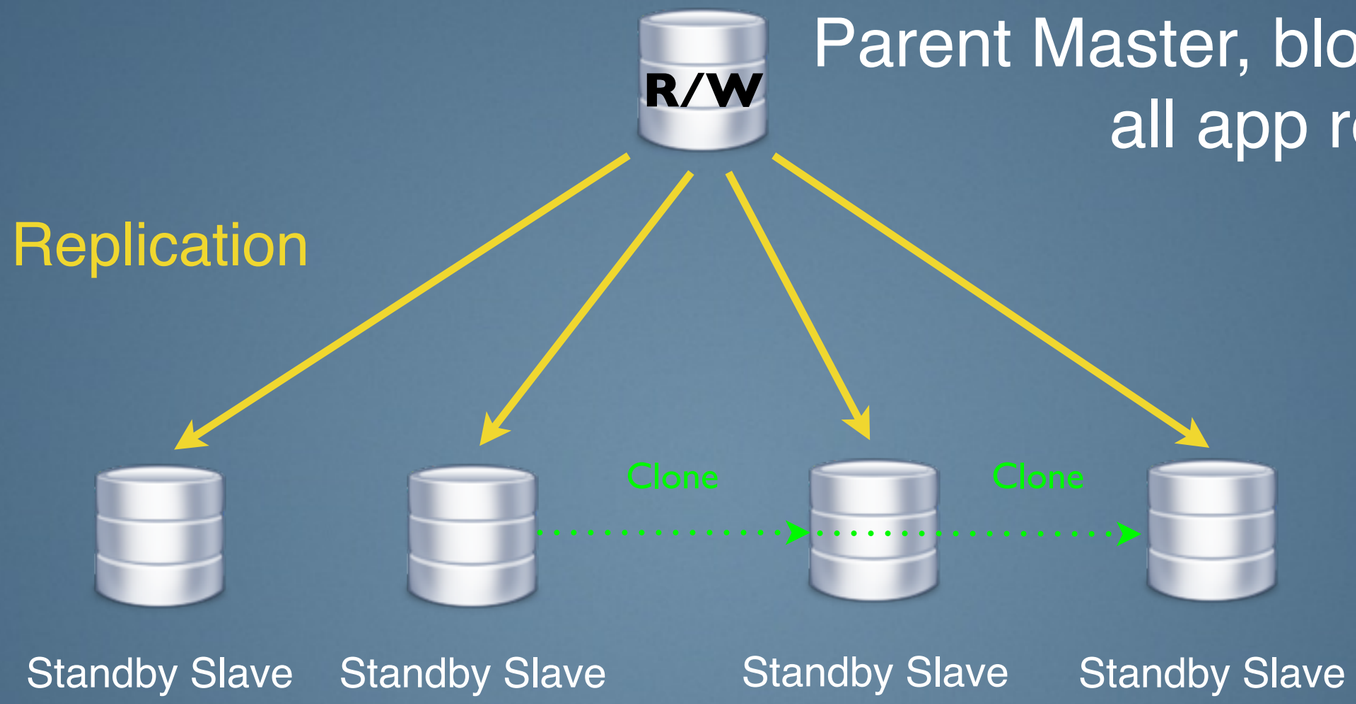
Splitting shards: process

Large “parent” shard divided into N “child” shards

1. Create N new slaves in parent shard pool — these will soon become masters of their own shard pools
2. Reduce the data set on those slaves so that each contains a different subset of the data
3. Move app reads from the parent to the appropriate children
4. Move app writes from the parent to the appropriate children
5. Stop replicating writes from the parent; take the parent pool offline
6. Remove rows that replicated to the wrong child shard



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


R/W
Parent Master, blogs 1-1000,
all app reads/writes
Replication
Clone
Clone
Standby Slave
Standby Slave
Standby Slave
Standby Slave

Aside: copying files efficiently

To a single destination

- Our shard-split process relies on creating new slaves quickly, which involves copying around very large data sets
- For compression we use pigz (parallel gzip), but there are other alternatives like qpress
- Destination box: `nc -l [port] | pigz -d | tar xvf -`
- Source box: `tar vc . | pigz | nc [destination hostname] [port]`

Aside: copying files efficiently

To multiple destinations

- Add tee and a FIFO to the mix, and you can create a chained copy to multiple destinations simultaneously
- Each box makes efficient use of CPU, memory, disk, uplink, downlink
- Performance penalty is only around 3% to 10% — much better than copying serially or from copying in parallel from a single source
- http://engineering.tumblr.com/post/7658008285/efficiently-copying-files-to-multiple-destinations

Splitting shards: process

Large “parent” shard divided into N “child” shards

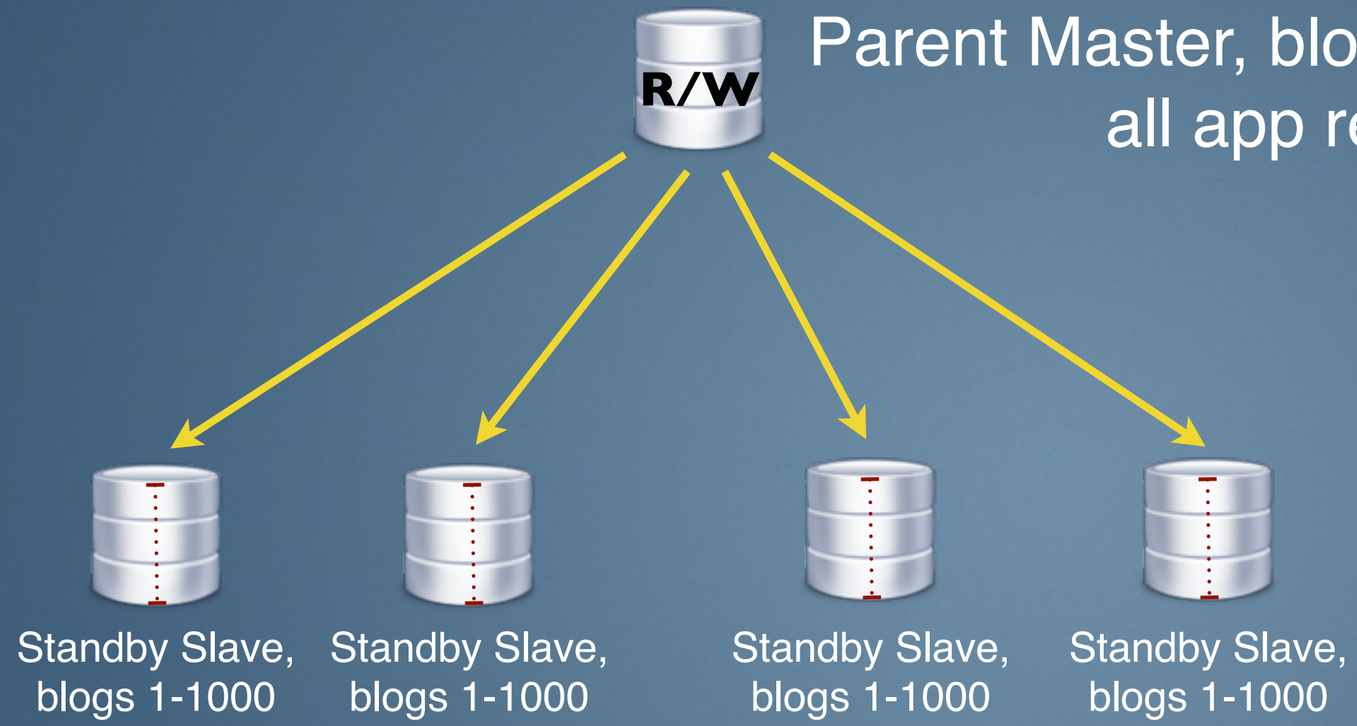
1. Create N new slaves in parent shard pool — these will soon become masters of their own shard pools
2. Reduce the data set on those slaves so that each contains a different subset of the data
3. Move app reads from the parent to the appropriate children
4. Move app writes from the parent to the appropriate children
5. Stop replicating writes from the parent; take the parent pool offline
6. Remove rows that replicated to the wrong child shard

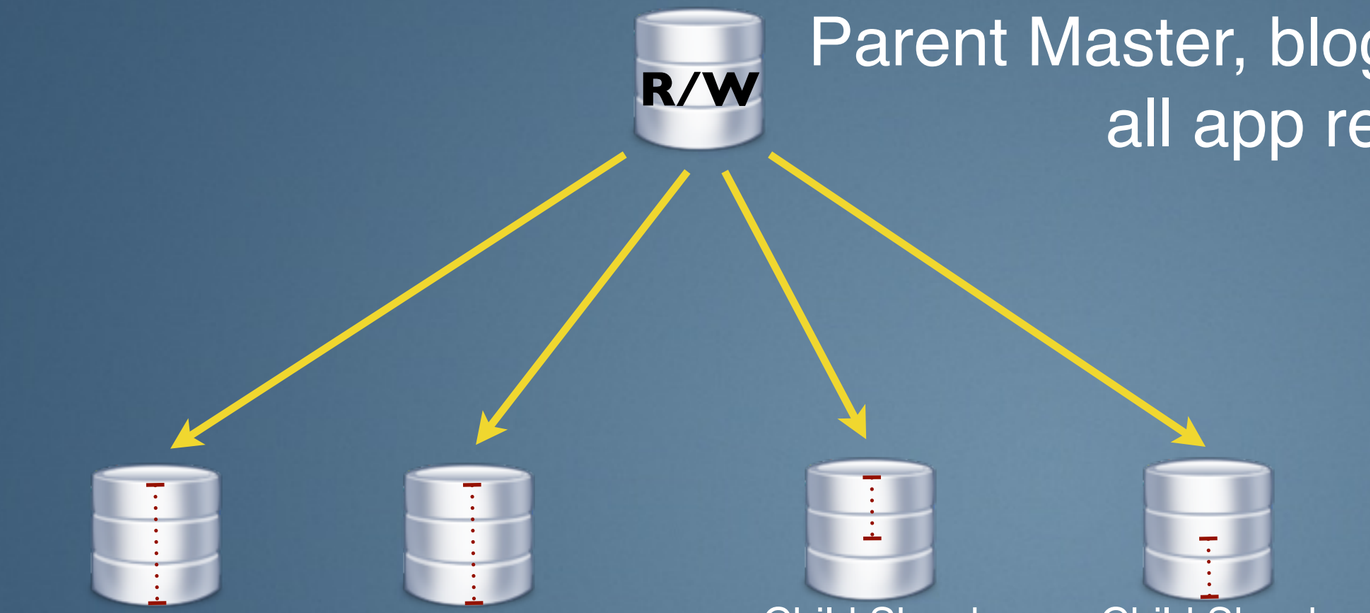
Importing/exporting in chunks

- Divide ID range into many chunks, and then execute export queries in parallel. Likewise for import.
- Export via `SELECT * FROM [table] WHERE [range clause] INTO OUTFILE [file]`
- Use mysqldump to export `DROP TABLE` / `CREATE TABLE` statements, and then run those to get clean slate
- Import via `LOAD DATA INFILE`

Importing/exporting gotchas

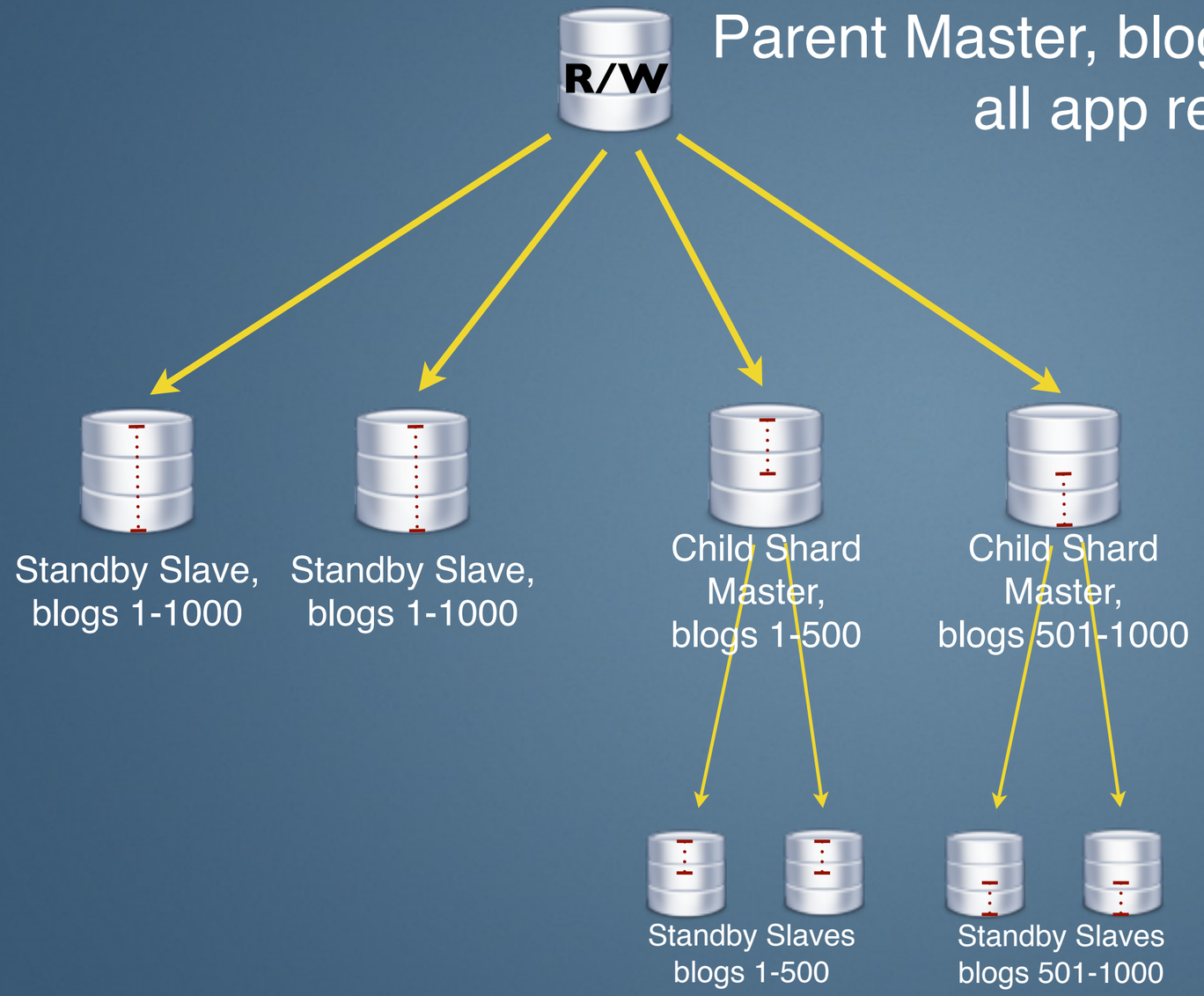
- Disable binary logging before import, to speed it up
- Disable any query-killer scripts, or filter out `SELECT ... INTO OUTFILE` and `LOAD DATA INFILE`
- Benchmark to figure out how many concurrent import/export queries can be run on your hardware
- Be careful with disk I/O scheduler choices if using multiple disks with different speeds


R/W
Parent Master, blogs 1-1000,
all app reads/writes
Standby Slave,
blogs 1-1000
Standby Slave,
blogs 1-1000
Standby Slave,
blogs 1-1000
Standby Slave,
blogs 1-1000



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

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



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

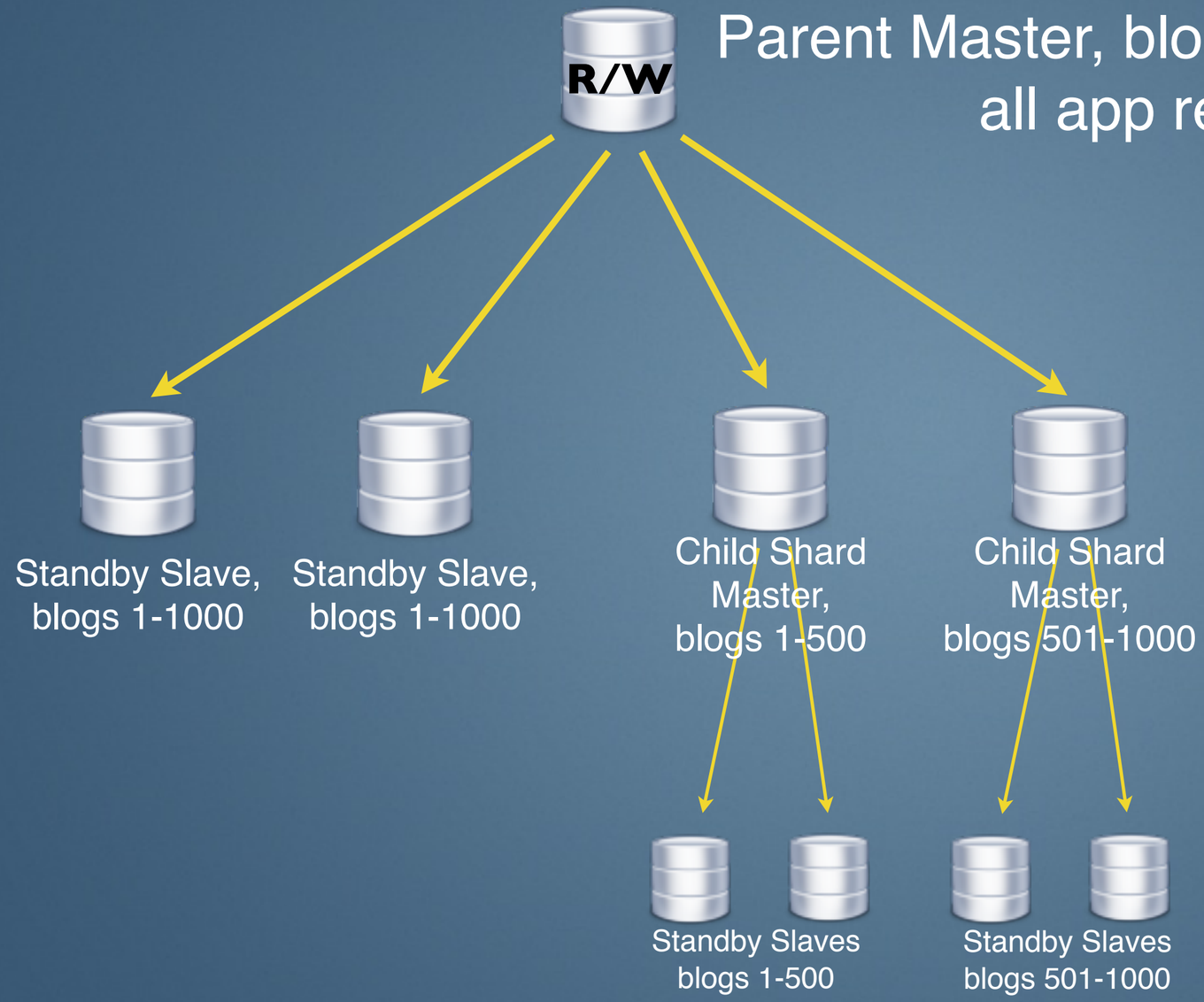
Splitting shards: process

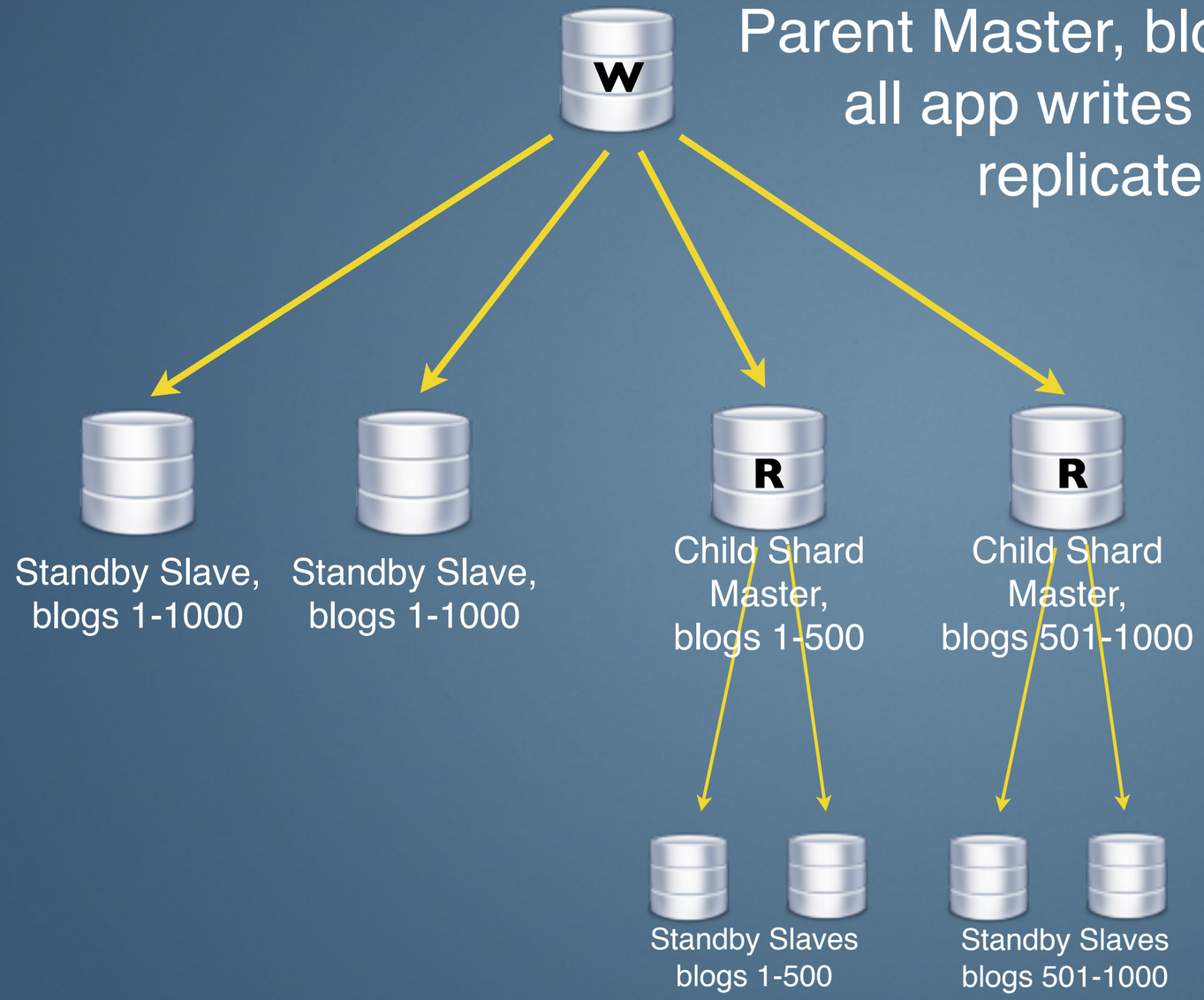
Large “parent” shard divided into N “child” shards

1. Create N new slaves in parent shard pool — these will soon become masters of their own shard pools
2. Reduce the data set on those slaves so that each contains a different subset of the data
3. Move app reads from the parent to the appropriate children
4. Move app writes from the parent to the appropriate children
5. Stop replicating writes from the parent; take the parent pool offline
6. Remove rows that replicated to the wrong child shard

Moving reads and writes separately

- If configuration updates do not simultaneously reach all of your web/app servers, this would create consistency issues if reads/writes moved at same time
  - Web A gets new config, writes post for blog 200 to 1st child shard
  - Web B is still on old config, renders blog 200, reads from parent shard, doesn't find the new post
- Instead only move reads first; let writes keep replicating from parent shard
- After first config update for reads, do a second one moving writes
- Then wait for parent master binlog to stop moving before proceeding


R/W
Parent Master, blogs 1-1000,
all app reads/writes
Standby Slave,
blogs 1-1000
Standby Slave,
blogs 1-1000
Child Shard
Master,
blogs 1-500
Child Shard
Master,
blogs 501-1000
Standby Slaves
blogs 1-500
Standby Slaves
blogs 501-1000



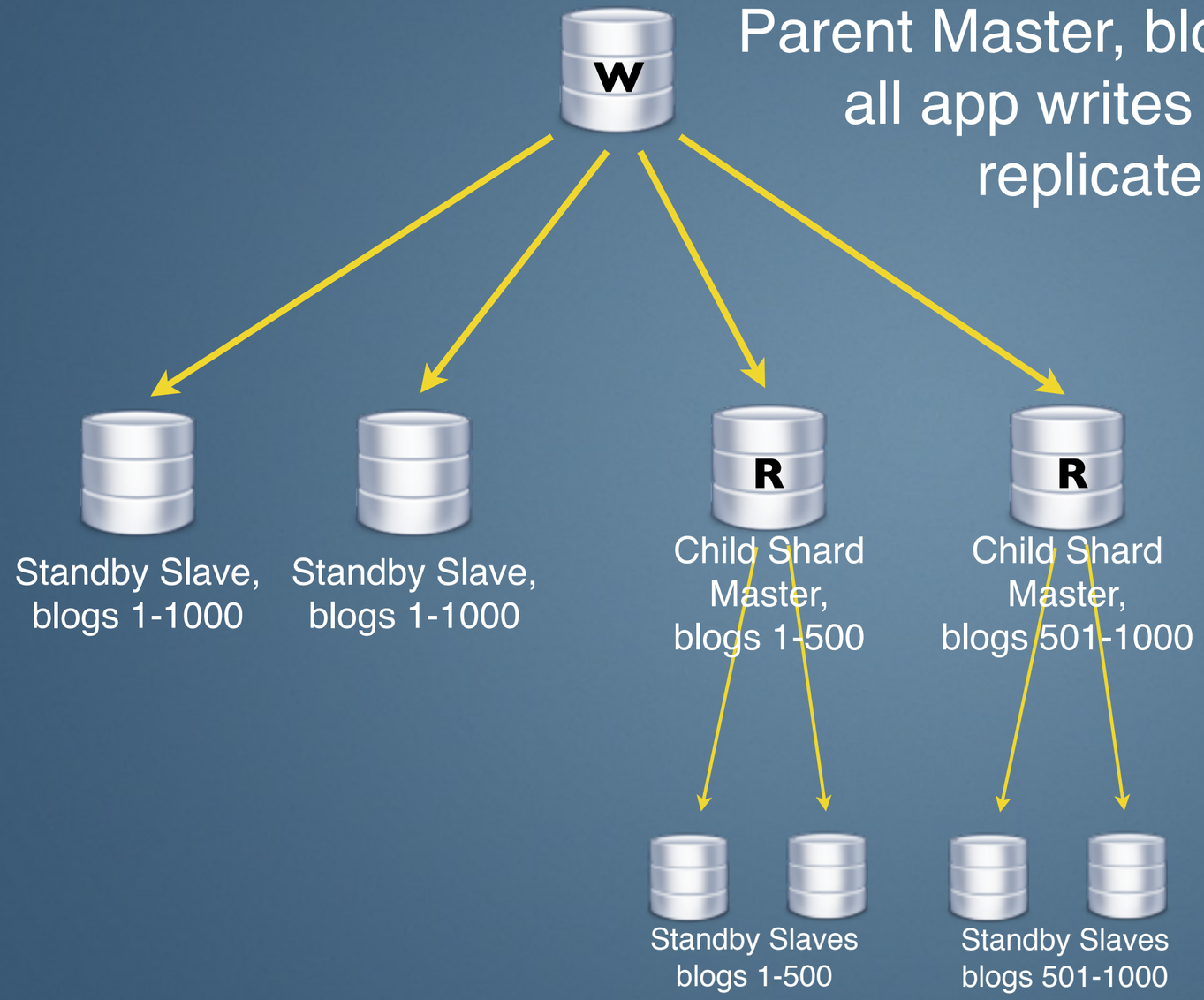
Parent Master, blogs 1-1000, all app writes (which then replicate to children)

Reads now go to children for appropriate queries to these ranges

Splitting shards: process

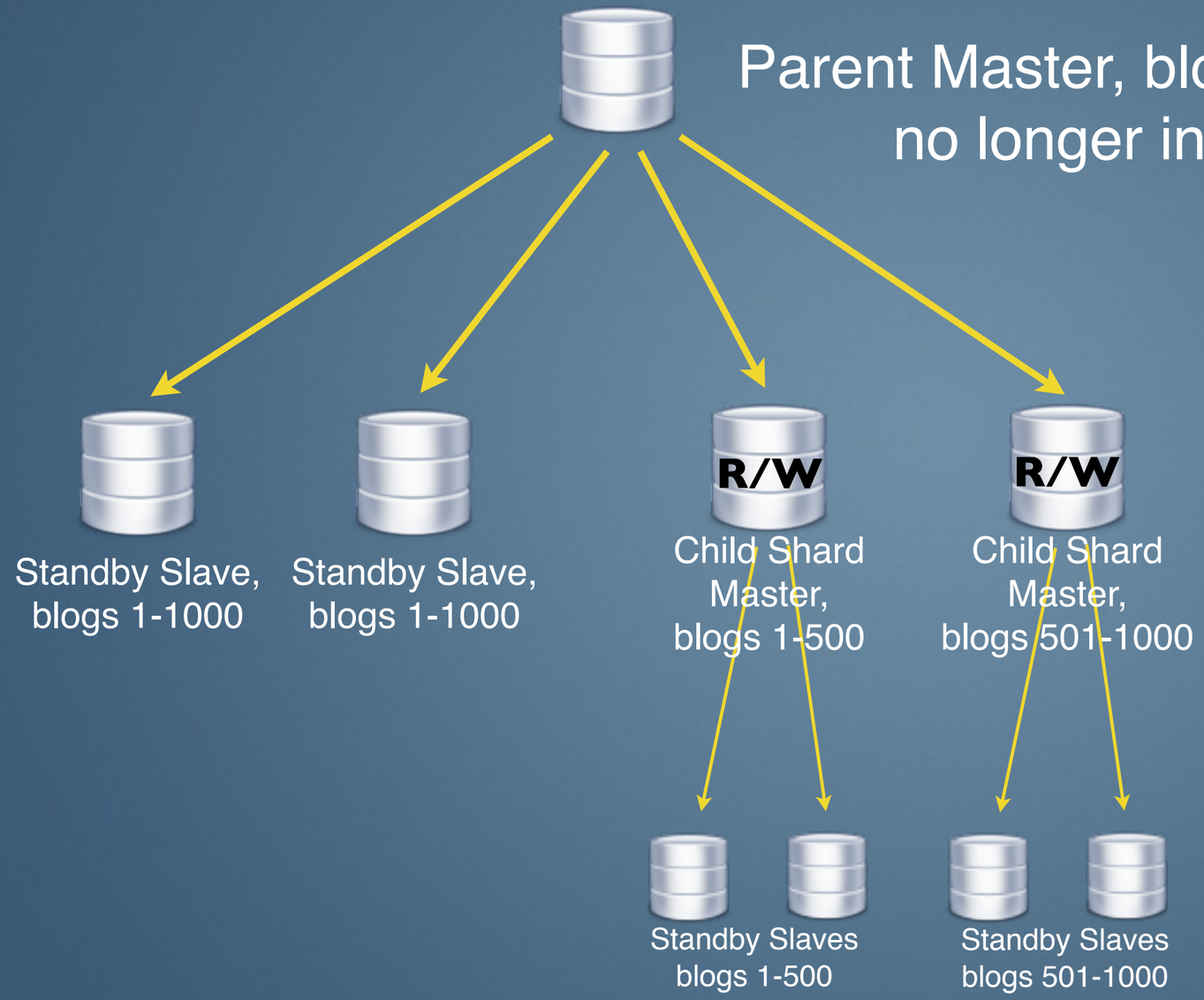
Large “parent” shard divided into N “child” shards

1. Create N new slaves in parent shard pool — these will soon become masters of their own shard pools
2. Reduce the data set on those slaves so that each contains a different subset of the data
3. Move app reads from the parent to the appropriate children
4. Move app writes from the parent to the appropriate children
5. Stop replicating writes from the parent; take the parent pool offline
6. Remove rows that replicated to the wrong child shard



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

Reads now go to children for appropriate queries to these ranges

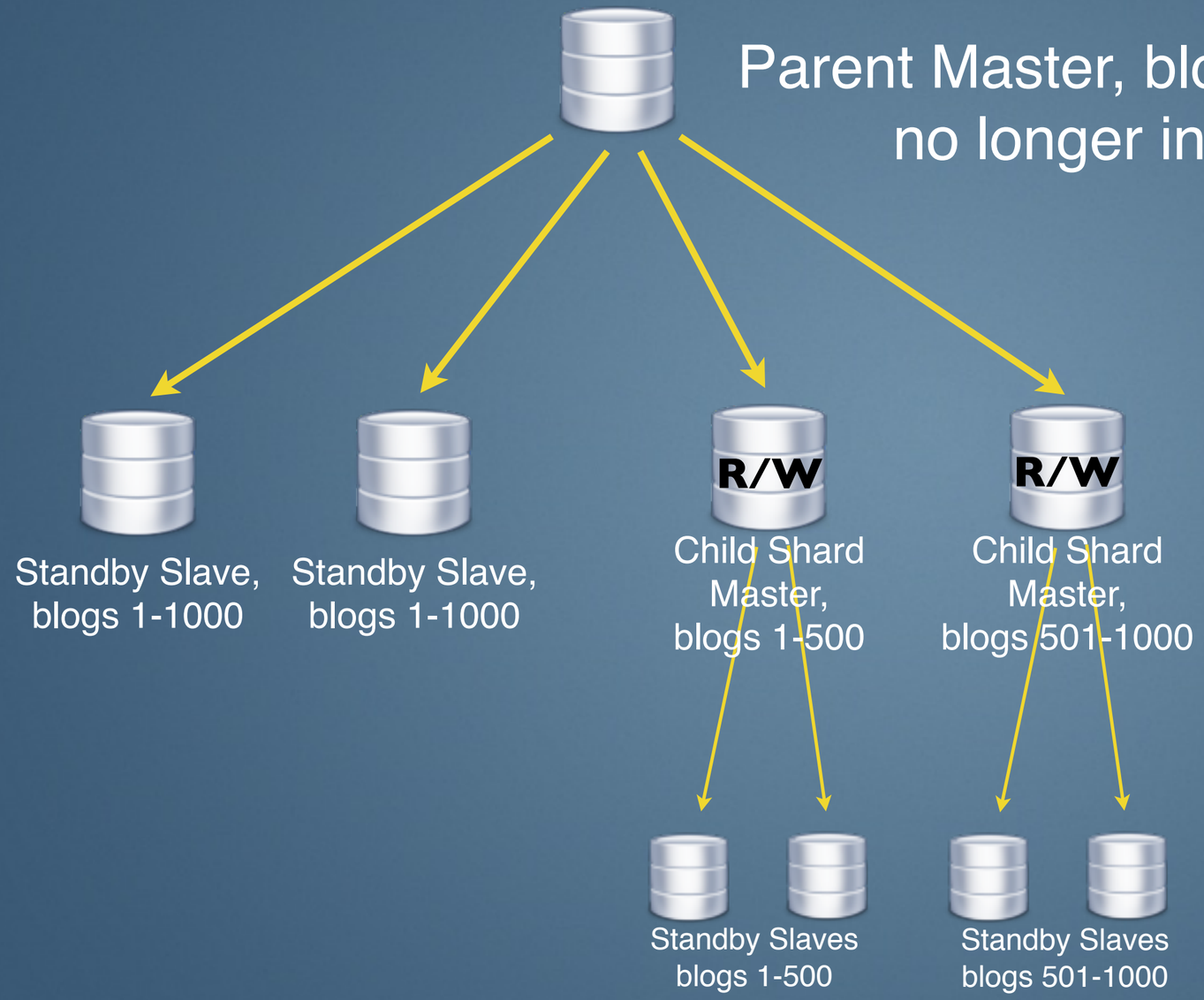


Reads and writes now go to children for appropriate queries to these ranges

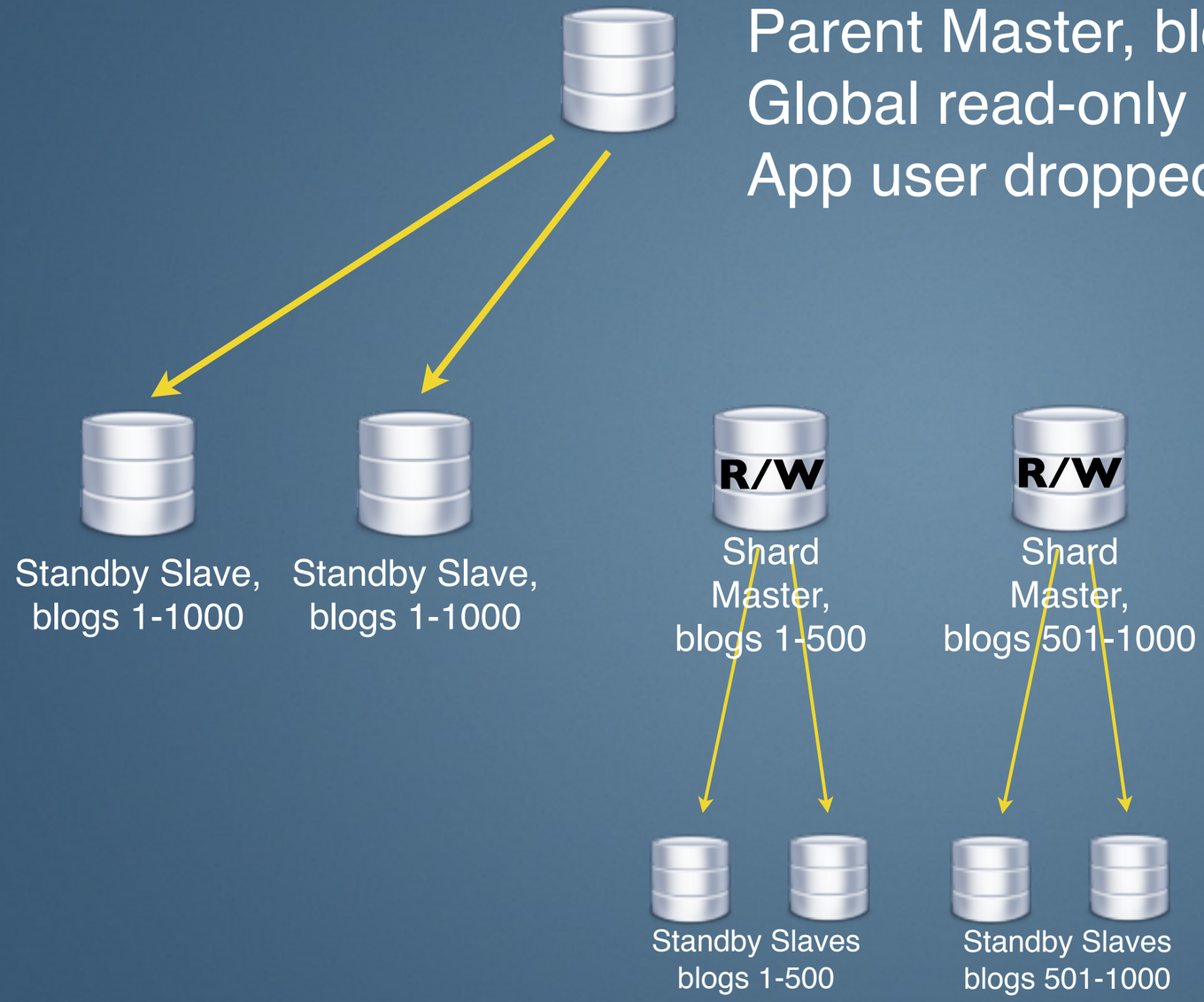
Splitting shards: process

Large “parent” shard divided into N “child” shards

1. Create N new slaves in parent shard pool — these will soon become masters of their own shard pools
2. Reduce the data set on those slaves so that each contains a different subset of the data
3. Move app reads from the parent to the appropriate children
4. Move app writes from the parent to the appropriate children
5. Stop replicating writes from the parent; take the parent pool offline
6. Remove rows that replicated to the wrong child shard



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



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,
blogs 1-1000
Recycle soon

Standby Slave,
blogs 1-1000
Recycle soon

Shard
Master,
blogs 1-500

Shard
Master,
blogs 501-1000

Standby Slaves
blogs 1-500

Standby Slaves
blogs 501-1000

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

Splitting shards: process

Large “parent” shard divided into N “child” shards

1. Create N new slaves in parent shard pool — these will soon become masters of their own shard pools
2. Reduce the data set on those slaves so that each contains a different subset of the data
3. Move app reads from the parent to the appropriate children
4. Move app writes from the parent to the appropriate children
5. Stop replicating writes from the parent; take the parent pool offline
6. Remove rows that replicated to the wrong child shard

Splitting shards: cleanup

- Until writes are moved to the new shard masters, writes to the parent shard replicate to all its child shards
- Purge this data after shard split process is finished
- Avoid huge single DELETE statements — they cause slave lag, among other issues (huge long transactions are generally bad)
- Data is sparse, so it's efficient to repeatedly do a SELECT (find next sharding key value to clean) followed by a DELETE.

General sharding gotchas

- **Sizing shards properly**: as small as you can operationally handle
- **Choosing ranges**: try to keep them even, and better if they end in 000's than 999's or, worse yet, random ugly numbers. This matters once shard naming is based solely on the ranges.
- **Cutover**: regularly create new shards to handle future data. Take the max value of your sharding key and add a cushion to it.
  - Before: highest blog ID 31.8 million, last shard handles range [30m, ∞)
  - After: that shard now handles [30m, 32m) and new last shard handles [32m, ∞)

Questions?

Email: evan -at- tumblr.com