Zero-Downtime Rebalancing and Data Migration of a Mature Multi-Shard Platform

JUSTIN LI
@pushrax

FLORIAN WEINGARTEN
@fw1729

shopify

Zero-Downtime Rebalancing and Data Migration of a Mature Multi-Shard Platform

Zero-Downtime Rebalancing and Data Migration of a Mature Multi-Shard Platform

Zero-Downtime Rebalancing and Data Migration of a Mature Multi-Shard Platform

Zero-Downtime Rebalancing and Data Migration of a Mature Multi-Shard Platform

Zero-Downtime Rebalancing and Data Migration of a Mature Multi-Shard Platform

Zero-Downtime Rebalancing and Data Migration of a Mature Multi-Shard Platform

INTRODUCTION

SHOPIFY'S MULTI-SHARD PLATFORM

Shopify's data axioms

- Many shops can share the same database shard, but…
- **All** of a shop's data is stored in the **same** database shard
- Shop datasets are **completely independent** of each other, i.e., shop A's data never references shop B's data
- Every MySQL table has a **`shop_id`** column
- Every "unit of work" can **only access data of one shop**

Shard-aware primary key generation

id	shop_id	product_name
1	1	…
2	2	…
3	2	…
4	1	…
5	3	…
6	1	…

Shard-aware primary key generation

id	shop_id	product_name
1	1	…
2	2	…
3	2	…
4	1	…
5	3	…
6	1	…

id	shop_id	product_name

Shard-aware primary key generation

id	shop_id	product_name
1	1	…
2	2	…
3	2	…
4	1	…
5	3	…
6	1	…

id	shop_id	product_name
1	17	…
2	24	…
3	31	…

Shard-aware primary key generation

id	shop_id	product_name
1	1	…
2	2	…
3	2	…
4	1	…
5	3	…
6	1	…

id	shop_id	product_name
1	17	…
2	24	…
3	31	…

Shard-aware primary key generation

id	shop_id	product_name
2	1	…
4	2	…
6	2	…
8	1	…
10	3	…
…	…	…
2*j + 0	…	…

id	shop_id	product_name
1	17	…
3	24	…
5	31	…
7	17	…
11	17	…
…	…	…
2*j + 1	…	…

Shard-aware primary key generation

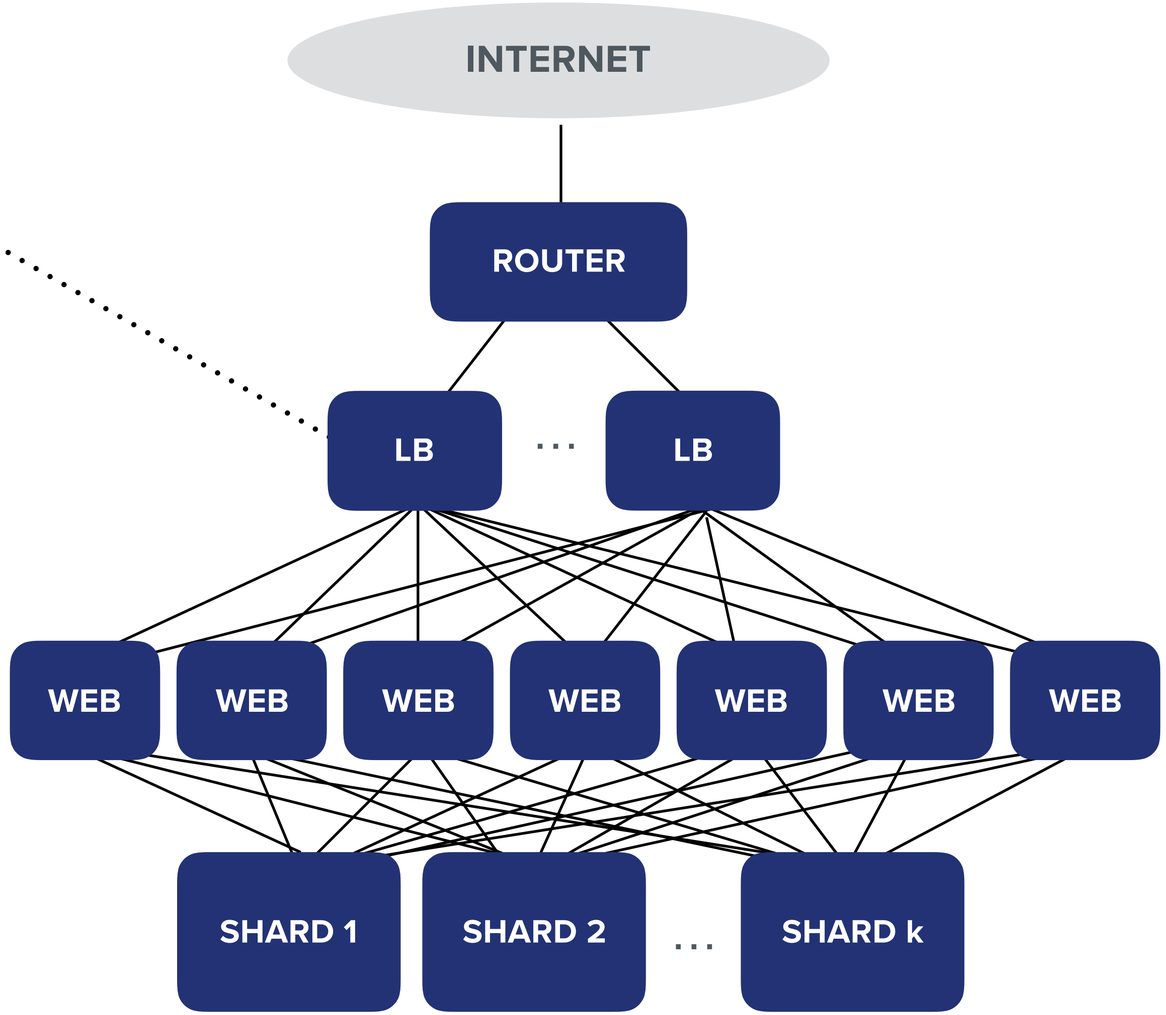
Shard 0	Shard 1	Shard 2	Shard 3
0	-	-	-
-	1	-	-
-	-	2	-
-	-	-	3
4	-	-	-
-	5	-	-
-	-	6	-
-	-	-	7
8	-	-	-
-	9	-	-
-	-	10	-
-	-	-	11

- shard `i` generates ids **`n*j + i`**
- id spaces are **disjoint**, no collisions
- 100% **decentralized**, no "id generator" authority required
- `auto_increment_increment` (n=4)
- `auto_increment_offset` (i)

Shard-aware request routing

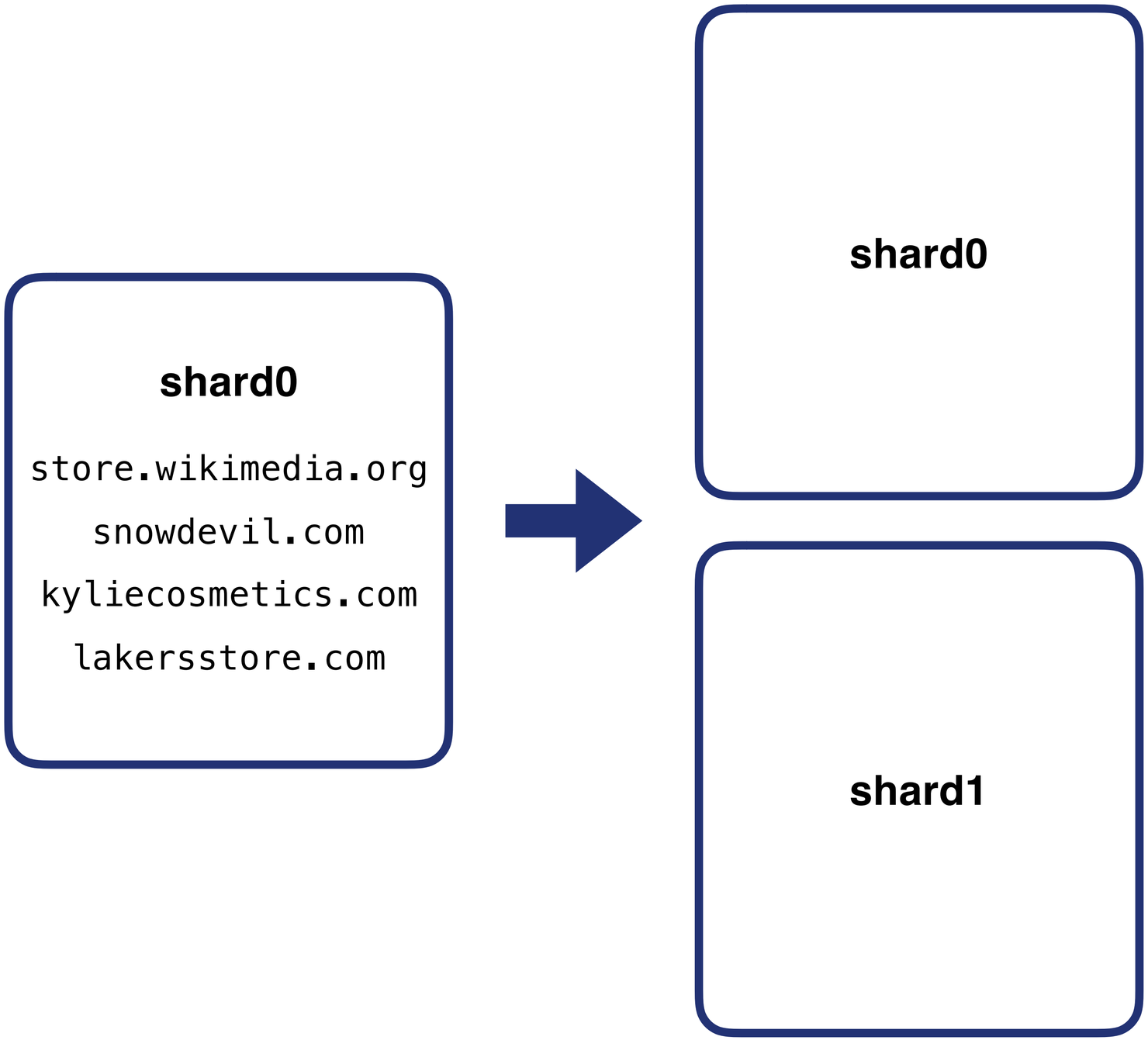
(simplified)

domain	shop_id	shard_id
foo.myshopify.com	1	1
bar.myshopify.com	2	3
fashionnova.com	3	17
startupsocks.com	4	1
snowdevil.com	5	27
kyliecosmetics.com	6	5



SHARDING IN PRODUCTION

THE PROBLEMS NOBODY TOLD YOU ABOUT


shard0
store.wikimedia.org
snowdevil.com
kyliecosmetics.com
lakersstore.com
shard0
shard1

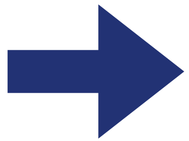
shard0

store.wikimedia.org

snowdevil.com

kyliecosmetics.com

lakersstore.com



shard0

store.wikimedia.org

snowdevil.com

kyliecosmetics.com

lakersstore.com

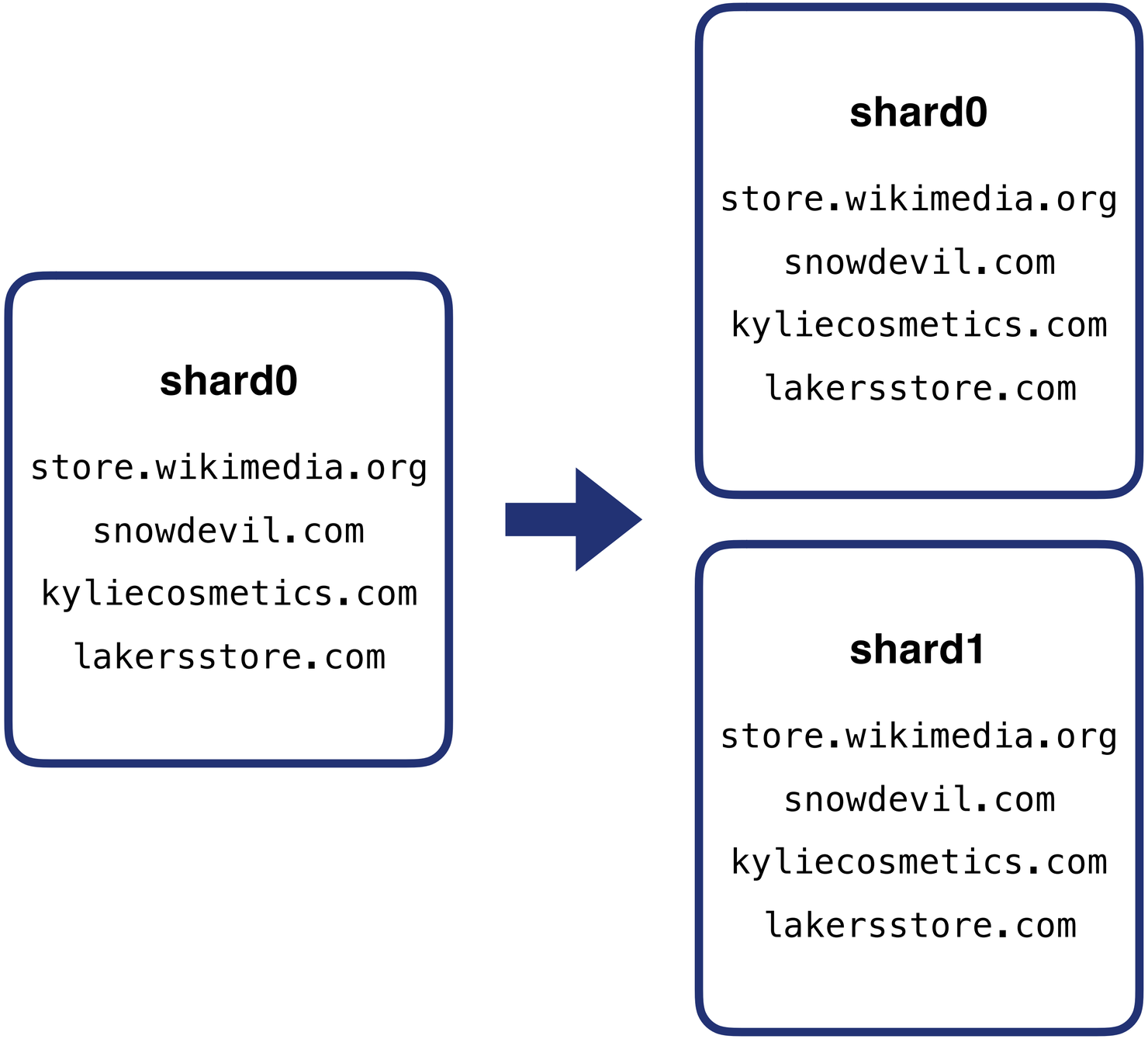
shard0-copy

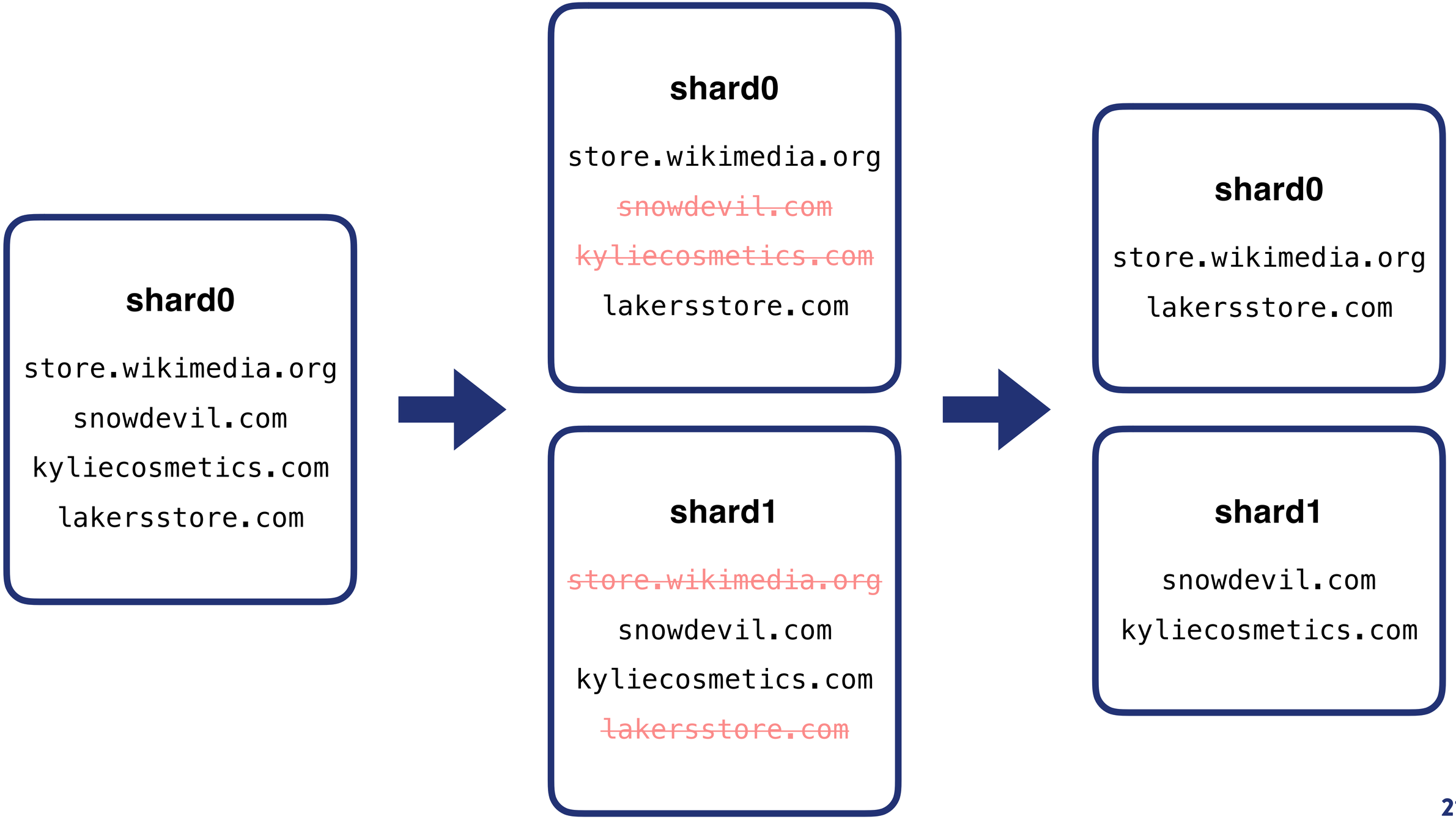
store.wikimedia.org

snowdevil.com

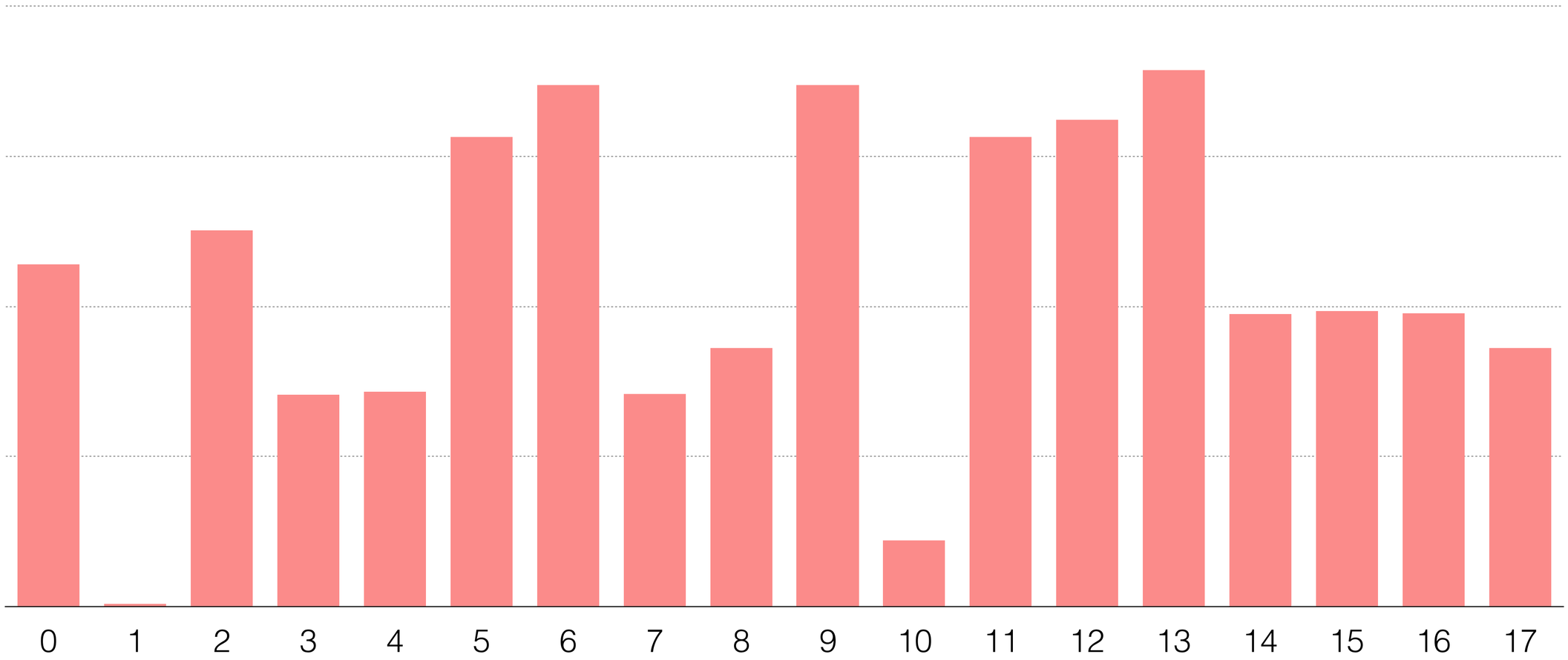
kyliecosmetics.com

lakersstore.com

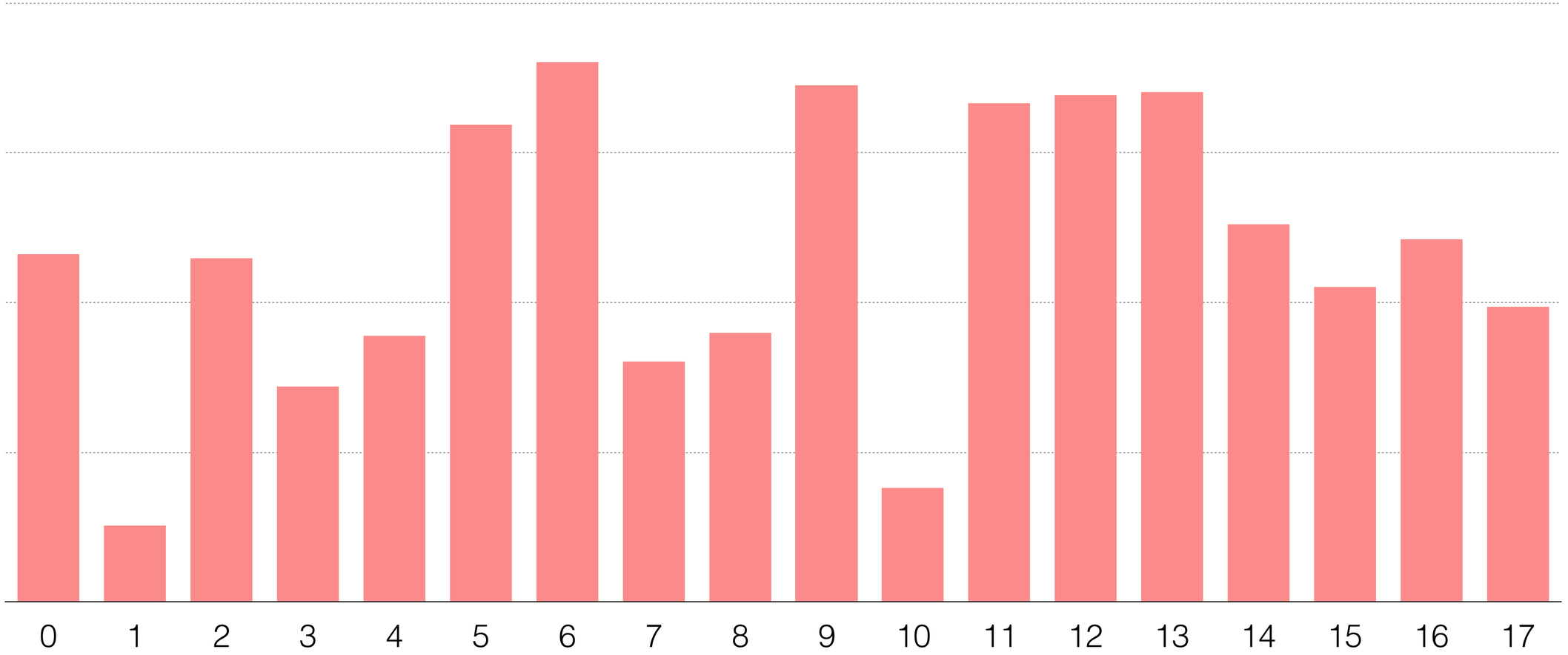

shard0
store.wikimedia.org
snowdevil.com
kyliecosmetics.com
lakersstore.com
shard0
store.wikimedia.org
snowdevil.com
kyliecosmetics.com
lakersstore.com
shard1
store.wikimedia.org
snowdevil.com
kyliecosmetics.com
lakersstore.com


shard0
store.wikimedia.org
snowdevil.com
kyliecosmetics.com
lakersstore.com
shard0
store.wikimedia.org
~~snowdevil.com~~
~~kyliecosmetics.com~~
lakersstore.com
shard1
~~store.wikimedia.org~~
snowdevil.com
kyliecosmetics.com
~~lakersstore.com~~
shard0
store.wikimedia.org
lakersstore.com
shard1
snowdevil.com
kyliecosmetics.com

Tenants by shard



Stored data by shard



shard0

id	shop_id	data
1	42	socks
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard1

id	shop_id	data
2	58	umbrella
4	58	vase

shard0

id	shop_id	data
1	42	socks
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard1

id	shop_id	data
2	58	umbrella
4	58	vase

shard0

id	shop_id	data
1	42	socks
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard1

id	shop_id	data
1	42	socks
2	58	umbrella
4	58	vase
5	42	hat
7	42	shirt

shard0

id	shop_id	data
3	77	laptop
9	77	watch

shard1

id	shop_id	data
1	42	socks
2	58	umbrella
4	58	vase
5	42	hat
7	42	shirt

data integrity > availability > throughput

PROBLEM 1:

AVOIDING LOST WRITES

Avoiding lost writes

- **Problem:** Can’t allow shop to get modified during move
- **Ideas:**
  - Set database to readonly?
  - Kill all ongoing work for the shop?
  - Mark the shop as locked and don’t allow new work for it to start?

Avoiding lost writes

- **Problem:** Can’t allow shop to get modified during move
- **Solution:** “Readers-writers problem”

Shared/exclusive shop locking

- **Data structure:**
  - *Any number* of processes can acquire a **shared lock** concurrently, but only if no process is holding an exclusive lock.
  - *Only one* process can acquire an **exclusive lock**, but only if no process holds a shared lock.
- **Idea:**
  - Regular work that accesses shop data needs to "register" itself by acquiring a shared lock.
  - Shop mover acquires exclusive lock before moving the shop.

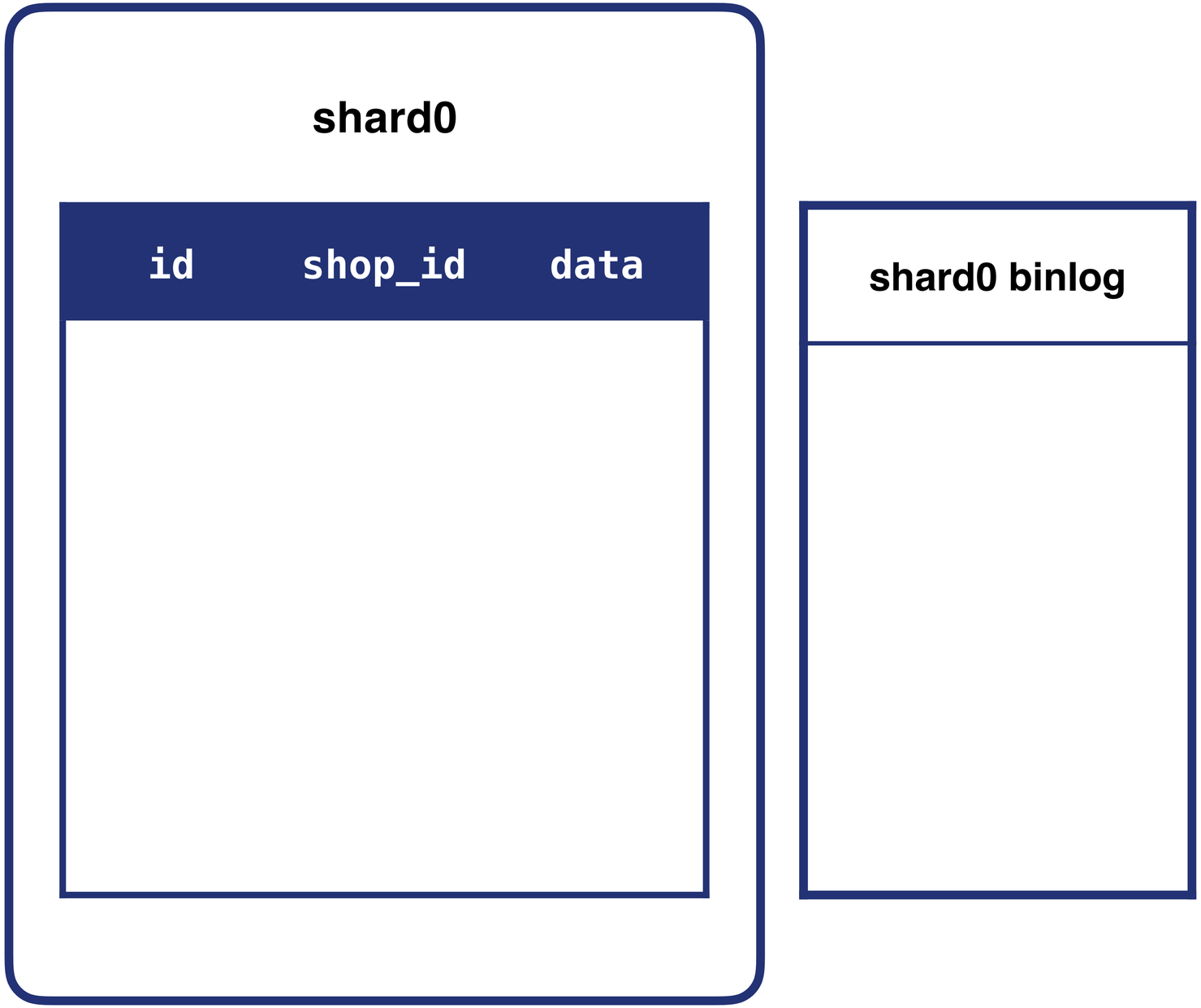
Shared/exclusive shop locking

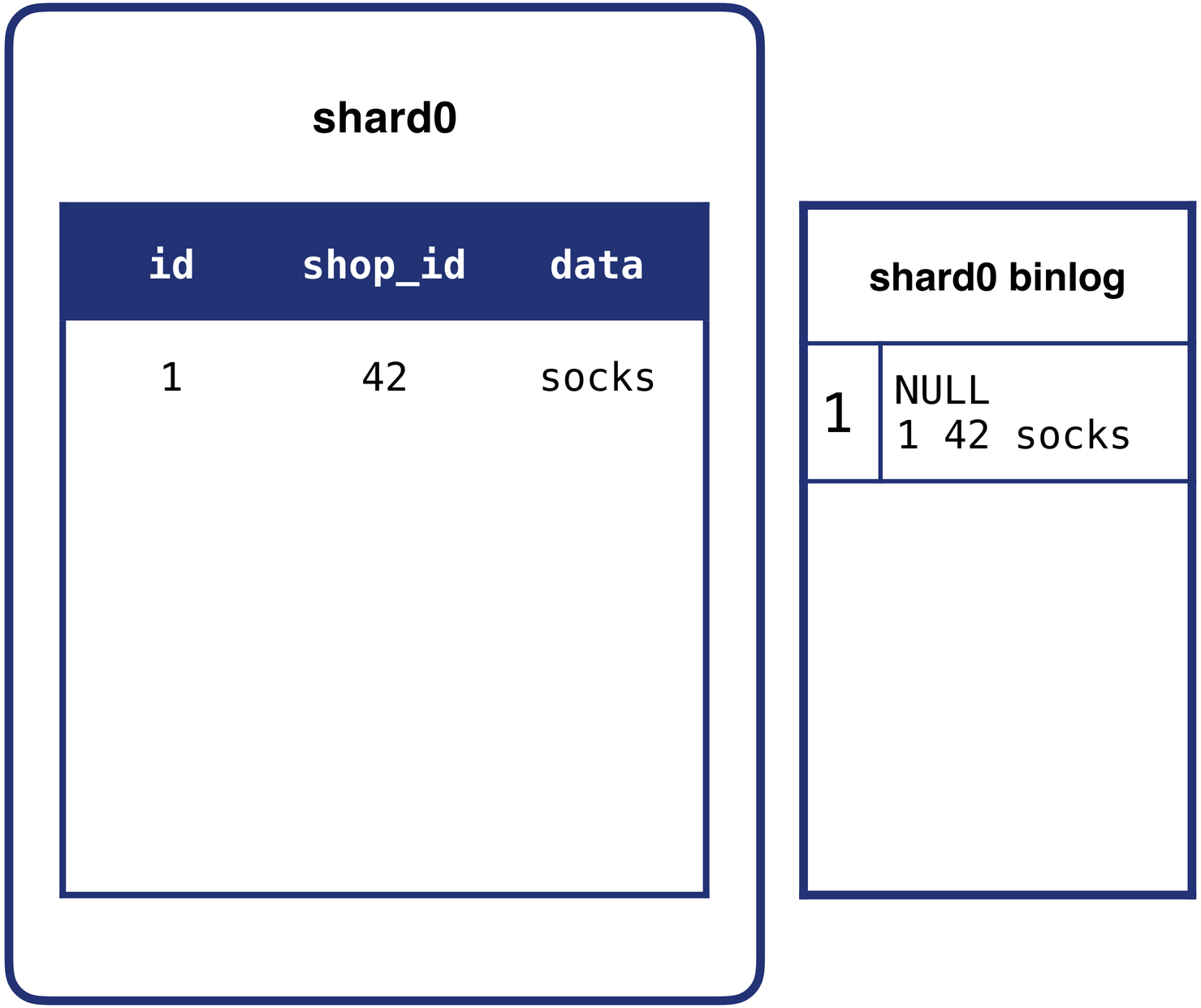
- **Idea:**
  - Regular work that accesses shop data needs to "register" itself by acquiring a shared lock.
- **Some advice:**
  - Safety depends on assumption that all work does this correctly
  - Make it **hard** for developers to circumvent this
  - Make this **invisible** and baked into your framework
  - Don’t underestimate required refactoring effort in legacy codebase

PROBLEM 2:

MINIMIZING DOWNTIME

- copy data in batches
- replicate writes from the source


shard0
id
shop_id
data
shard0 binlog


shard0
id
shop_id
data
1
42
socks

shard0 binlog
1
NULL
1 42 socks

shard0

id	shop_id	data
1	42	gloves

shard0 binlog	
1	NULL 1 42 socks
2	1 42 socks 1 42 gloves

shard0

id	shop_id	data
1	42	gloves
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard0 binlog

6	NULL 9 77 watch

shard1

id	shop_id	data
2	58	umbrella
4	58	vase

shard0

id	shop_id	data
1	42	gloves
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard0 binlog

6	NULL 9 77 watch

shard1

id	shop_id	data
2	58	umbrella
4	58	vase

shard0

id	shop_id	data
1	42	gloves
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard0 binlog

6	NULL 9 77 watch

shard1

id	shop_id	data
1	42	gloves
2	58	umbrella
4	58	vase

shard0

id	shop_id	data
1	42	mitts
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard0 binlog

6	NULL 9 77 watch
7	1 42 gloves 1 42 mitts

shard1

id	shop_id	data
1	42	gloves
2	58	umbrella
4	58	vase

shard0

id	shop_id	data
1	42	mitts
3	77	laptop
5	42	hat
7	42	shirt
9	77	watch

shard0 binlog

6	NULL 9 77 watch
7	1 42 gloves 1 42 mitts

shard1

id	shop_id	data
1	42	mitts
2	58	umbrella
4	58	vase

shard0

id	shop_id	data
1	42	mitts
3	77	laptop
5	42	hat
7	42	pants
9	77	watch

shard0 binlog

6	NULL 9 77 watch
7	1 42 gloves 1 42 mitts
8	7 42 shirt 7 42 pants

shard1

id	shop_id	data
1	42	mitts
2	58	umbrella
4	58	vase
5	42	hat
7	42	shirt

shard0

id	shop_id	data
1	42	mitts
3	77	laptop
5	42	hat
7	42	pants
9	77	watch

db0 binlog	
6	NULL 9 77 watch
7	1 42 gloves 1 42 mitts
8	7 42 shirt 7 42 pants
shop locked, no new writes	

shard1

id	shop_id	data
1	42	mitts
2	58	umbrella
4	58	vase
5	42	hat
7	42	pants

PROBLEM 3:

DATA INTEGRITY VERIFICATION

Data integrity verification

id	shop_id	email	name
1	1	simon@shopify.com	Simon
2	2	flo@shopify.com	Flo
3	1	bob@shopify.com	Bob
4	2	daniella@shopify.com	Daniella
5	3	camilio@camil.io	Camilio
6	2	tobi@jadedpixel.com	Tobi

id	shop_id	email	name
8	5	hormoz@waterloo.ca	Hormoz
2	2	flo@shopify.com	Flo
9	6	foo@bar.ca	Footer
4	2	daniella@shopify.com	Daniella
7	8	bla@bla.com	Dunnolol
6	2	tobi@jadedpixel.com	Tobi

Data integrity verification

id	shop_id	email	name
1	1	simon@shopify.com	Simon
2	**2**	flo@shopify.com	Flo
3	1	bob@shopify.com	Bob
4	**2**	daniella@shopify.com	Daniella
5	3	camilio@camil.io	Camilio
6	**2**	tobi@jadedpixel.com	Tobi

id	shop_id	email	name
8	5	hormoz@waterloo.ca	Hormoz
2	**2**	flo@shopify.com	Flo
9	6	foo@bar.ca	Footer
4	**2**	daniella@shopify.com	Daniella
7	8	bla@bla.com	Dunnolol
6	**2**	tobi@jadedpixel.com	Tobi

Naive approach

- SELECT all of the shop's data from the source shard
- SELECT all of the shop's data from the destination shard
- Compare all rows client-side
- 😴

Idea: Fingerprinting verification

- Digest shop's dataset to a fingerprint
- Such that A = B if (and only if) fingerprint(A) = fingerprint(B)
- Do this computation server-side (in MySQL)

Fingerprinting verification

id	shop_id	email	name
1	1	simon@shopify.com	Simon
2	**2**	flo@shopify.com	Flo
3	1	bob@shopify.com	Bob
4	**2**	daniella@shopify.com	Daniella
5	3	camilio@camil.io	Camilio
6	**2**	tobi@jadedpixel.com	Tobi

Fingerprinting verification

id	shop_id	email	name
2	**2**	flo@shopify.com	Flo
4	**2**	daniella@shopify.com	Daniella
6	**2**	tobi@jadedpixel.com	Tobi

Fingerprinting verification

id	shop_id	email	name
2	2	flo@shopify.com	Flo
4	2	daniella@shopify.com	Daniella
6	2	tobi@jadedpixel.com	Tobi

Fingerprinting verification

MD5(id)	MD5(shop_id)	MD5(email)	MD5(name)
2	2	flo@shopify.com	Flo
4	2	daniella@shopify.com	Daniella
6	2	tobi@jadedpixel.com	Tobi

Fingerprinting verification

MD5(id)	MD5(shop_id)	MD5(email)	MD5(name)
c81e7	c81e72	1feb9	300bb
a87ff	c81e7	f776f	8f216
616790	c81e7	43bc2	97d15

Fingerprinting verification

GROUP_CONCAT MD5(id)	GROUP_CONCAT MD5(shop_id)	GROUP_CONCAT MD5(email)	GROUP_CONCAT MD5(name)
c81e7 a87ff 616790	c81e72 c81e72 c81e72	1feb9 f776f 43bc2	300bb 8f216 97d15

Fingerprinting verification

MD5(GROUP_CONCAT MD5(id))	MD5(GROUP_CONCAT MD5(shop_id))	MD5(GROUP_CONCAT MD5(email))	MD5(GROUP_CONCAT MD5(name))
MD5(c81e7 a87ff 616790)	MD5(c81e72 c81e72 c81e72)	MD5(1feb9 f776f 43bc2)	MD5(300bb 8f216 97d15)

Fingerprinting verification

MD5(GROUP_CONCAT MD5(id))	MD5(GROUP_CONCAT MD5(shop_id))	MD5(GROUP_CONCAT MD5(email))	MD5(GROUP_CONCAT MD5(name))
19d25	3fea8	0a63d	5d38e

Fingerprinting verification

```
CONCAT(
  MD5(GROUP_CONCAT(MD5(id)),
MD5(GROUP_CONCAT(MD5(shop_id)),
 MD5(GROUP_CONCAT(MD5(email)),
 MD5(GROUP_CONCAT(MD5(name))
)
```

```
19d25 3fea8 0a63d 5d38e
```

Fingerprinting verification

```
MD5(
CONCAT(
MD5(GROUP_CONCAT(MD5(id)),
MD5(GROUP_CONCAT(MD5(shop_id)),
MD5(GROUP_CONCAT(MD5(email)),
MD5(GROUP_CONCAT(MD5(name))
))
```

MD5(19d25 3fea8 0a63d 5d38e)

Fingerprinting verification

```
MD5(...) AS fingerprint
```

5218e

Fingerprinting verification

```sql
SELECT
  MD5(
    CONCAT(
      MD5(GROUP_CONCAT(UNHEX(MD5(COALESCE(`first_name`, 'NULL'))) ORDER BY id SEPARATOR '')
      MD5(GROUP_CONCAT(UNHEX(MD5(COALESCE(`email`,      'NULL'))) ORDER BY id SEPARATOR '')
      -- ...
      -- same for all other relevant columns of that table
      -- ...
    )
  )
  AS fingerprint
FROM users
WHERE id IN (id1, id2, id3, id4, ..., id100)
```

Fingerprinting verification

- Heavy lifting happens in the database
- Light on the network
- O(M*N) hashing operations (M columns, N rows)
- 5x speedup vs. naive idea (same database load)
- Iterative re-verification as binlog changes come in

WHAT ELSE?

- Queueing moves
- Orchestrating mover processes
- Interrupting and transferring background jobs
- Replicating other datastores (sessions, etc.)
- Dealing with schema migrations

TL;DR

SUMMARY AND KEY TAKEAWAYS

Sharding functions

ID generation

Data migrations

Shared/exclusive locking for safety

Replication log
provides minimal downtime

Fingerprinting verification

You probably don't need to build this

github.com/shopify/ghostferry

Thanks! Questions?

Learn more:

Ghostferry: A Data Migration Tool for Incompatible Cloud Platforms
Shuhao Wu, Percona Live 2018

Shopify's Move from the Data Centre to the Cloud
Scott Francis, SREcon Asia 2018

Scaling Shopify's multi-tenant architecture across multiple data centers
Florian Weingarten, O'Reilly Velocity NY 2016

Scripting NGINX for Overload Protection
Justin Li, nginx.conf 2016

JUSTIN LI
@pushrax

FLORIAN WEINGARTEN
@fw1729