facebook

InnoDB to MyRocks migration in main MySQL database at Facebook

Yoshinori Matsunobu
Production Engineer, Facebook
May/2017

Who am I

- Was a MySQL consultant at MySQL (Sun, Oracle) for 4 years
- Joined Facebook in March 2012
  - MySQL 5.1 -> 5.6 upgrade
  - Fast master failover without losing data
  - Partly joined HBase Production Engineering team in 2014.H1
  - Started a research project to integrate RocksDB and MySQL from 2014.H2, with Database Engineering Team
  - Started MyRocks deployment since 2016.H2

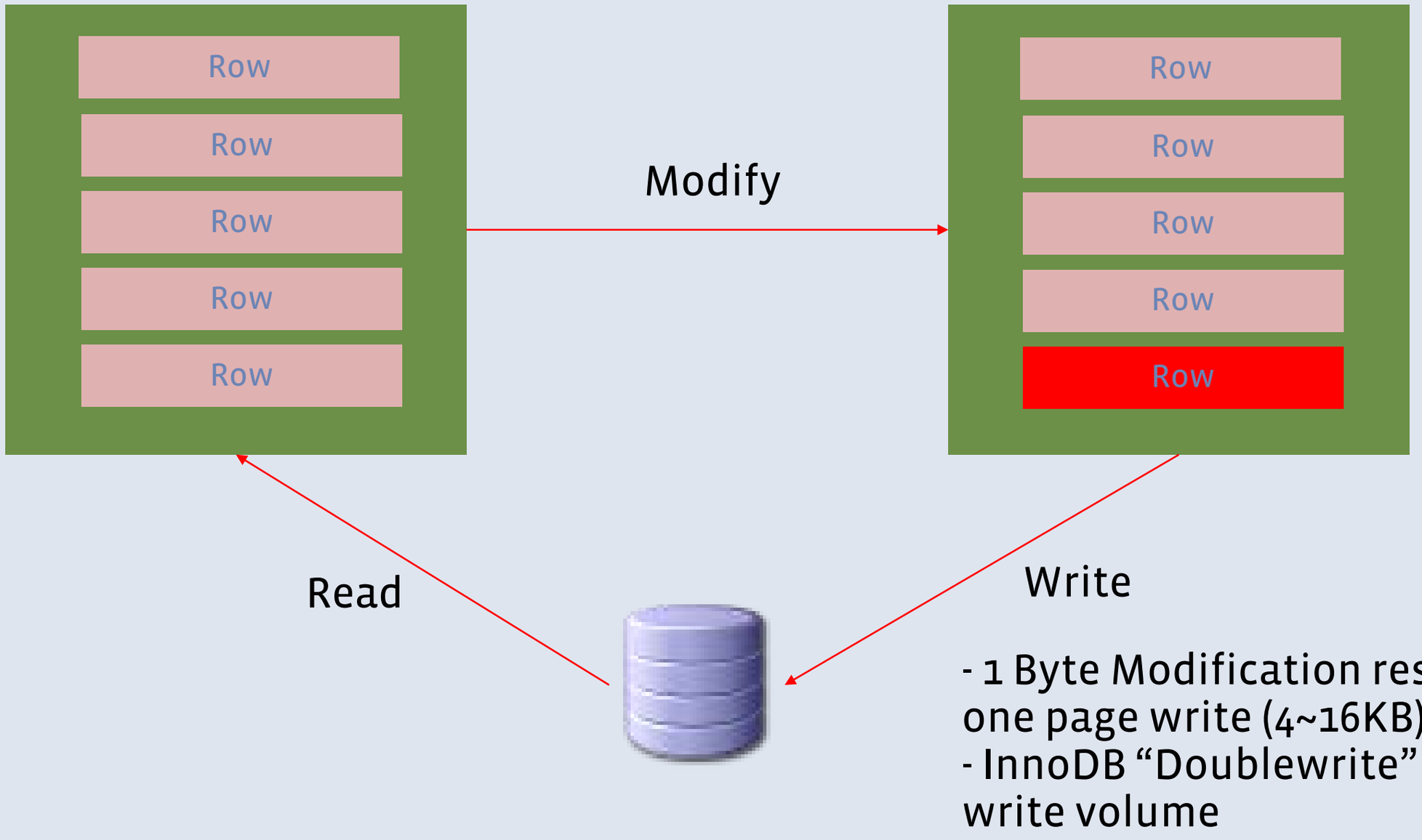
Agenda

- MySQL at Facebook
- Issues in InnoDB
- RocksDB and MyRocks overview
- Production Deployment

“Main MySQL Database” at Facebook

- Storing Social Graph
- Massively Sharded
- Petabytes scale
- Low latency
- Automated Operations
- Pure Flash Storage (Constrained by space, not by CPU/IOPS)

InnoDB Issue (1) -- Write Amplification

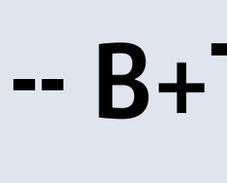


- 1 Byte Modification results in one page write (4~16KB)
- InnoDB “Doublewrite” doubles write volume

InnoDB Issue (2) -- B+Tree Fragmentation

```
INSERT INTO message_table (user_id) VALUES (31)
```

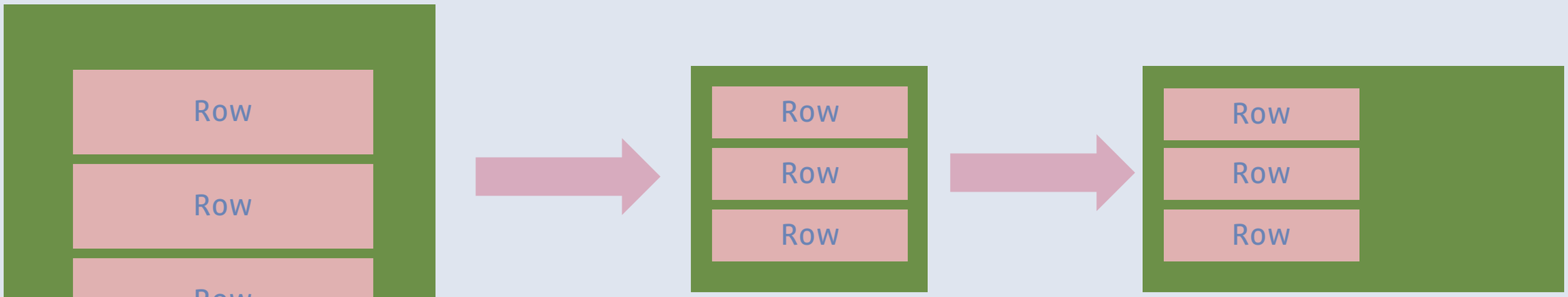
Leaf Block 1	
user_id	RowID
1	10000
2	5
3	15321
…	
60	431



Leaf Block 1	
user_id	RowID
1	10000
…	
30	333
Empty	

Leaf Block 2	
user_id	RowID
31	345
…	
60	431
Empty	

InnoDB Issue (3) -- Compression



Uncompressed
16KB page

Compressed
to 5KB

Using 8KB space
on storage

0~4KB => 4KB
4~8KB => 8KB
8~16KB => 16KB

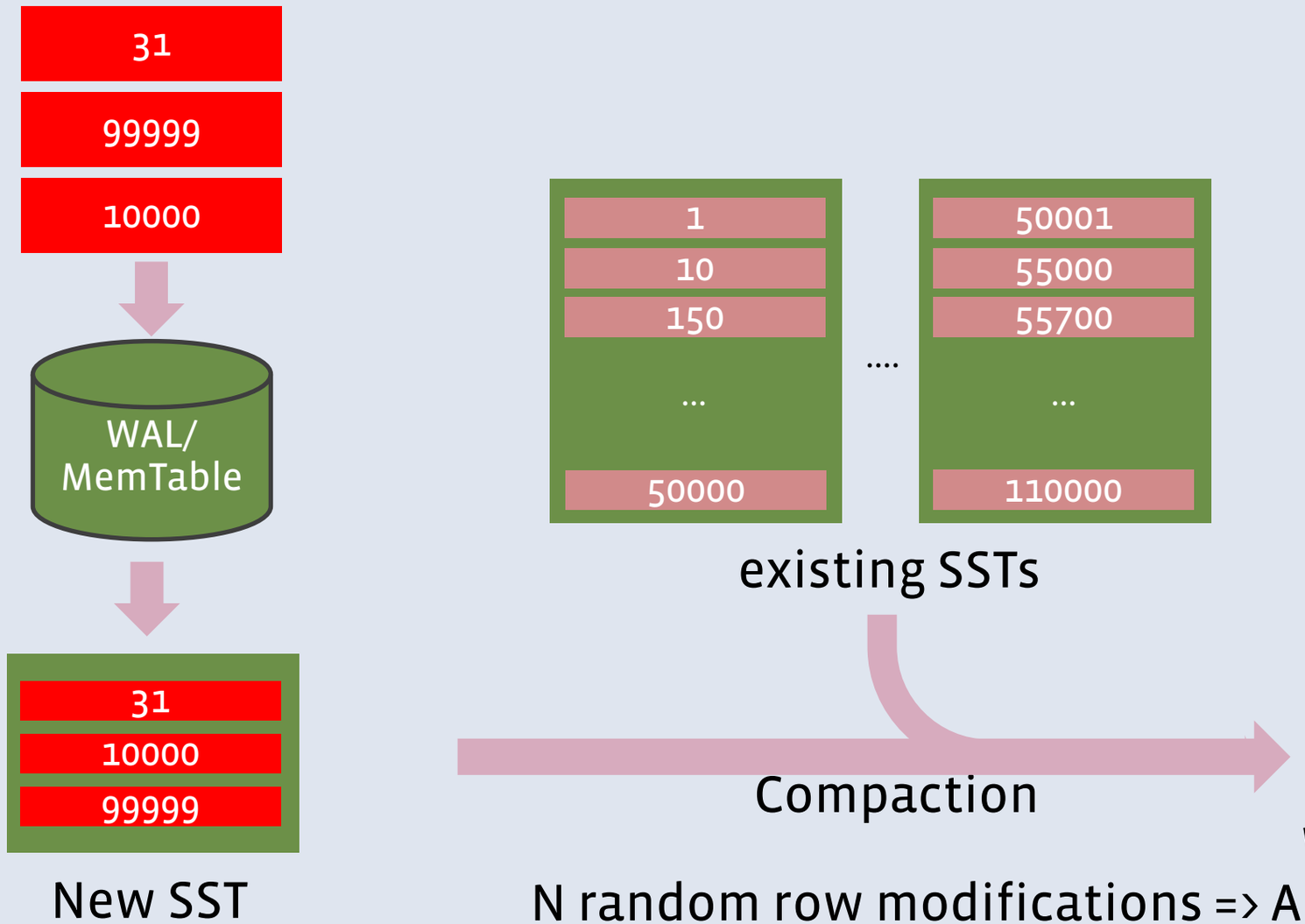
RocksDB

- http://rocksdb.org/
- Forked from LevelDB
  - Key-Value LSM (Log Structured Merge) persistent store
  - Embedded
  - Data stored locally
  - Optimized for fast storage

- LevelDB was created by Google
- Facebook forked and developed RocksDB
- Used at many backend services at Facebook, and many external large services
- Needs to write C++ or Java code to access RocksDB

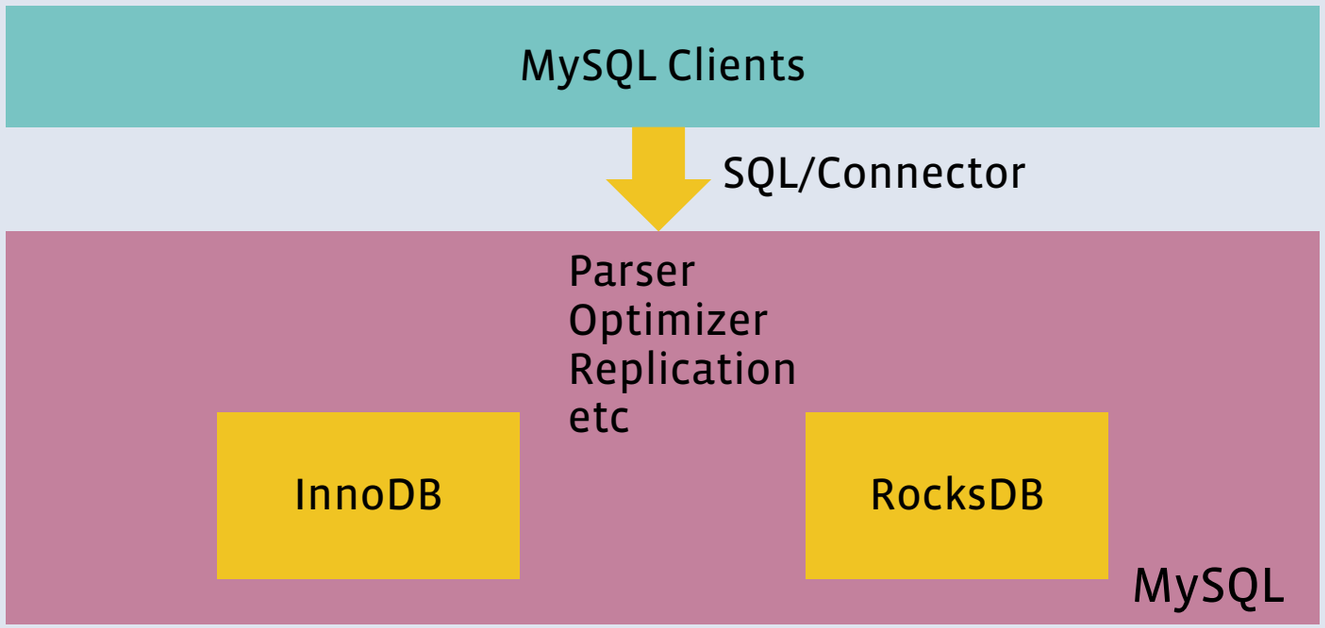
How LSM/RocksDB works

```
INSERT INTO message (user_id) VALUES (31);
INSERT INTO message (user_id) VALUES (99999);
INSERT INTO message (user_id) VALUES (10000);
```



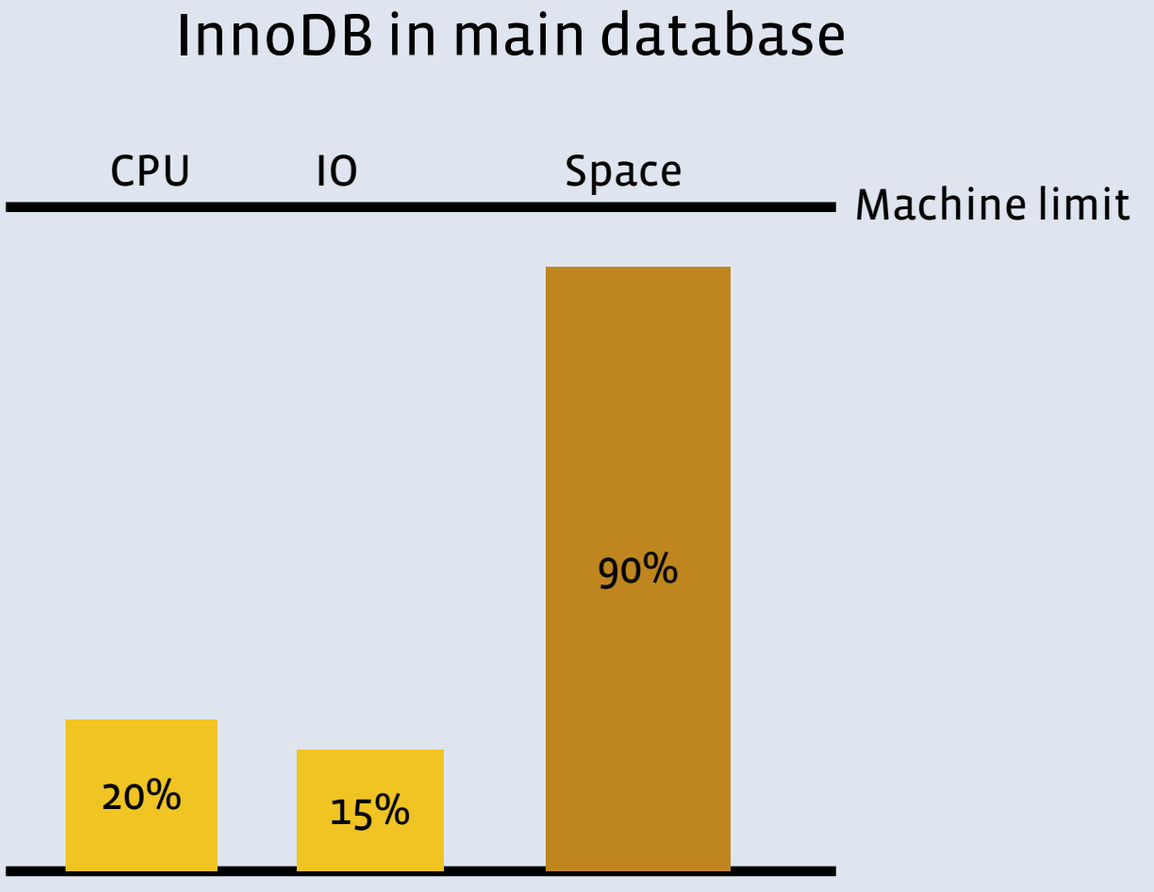
What is MyRocks

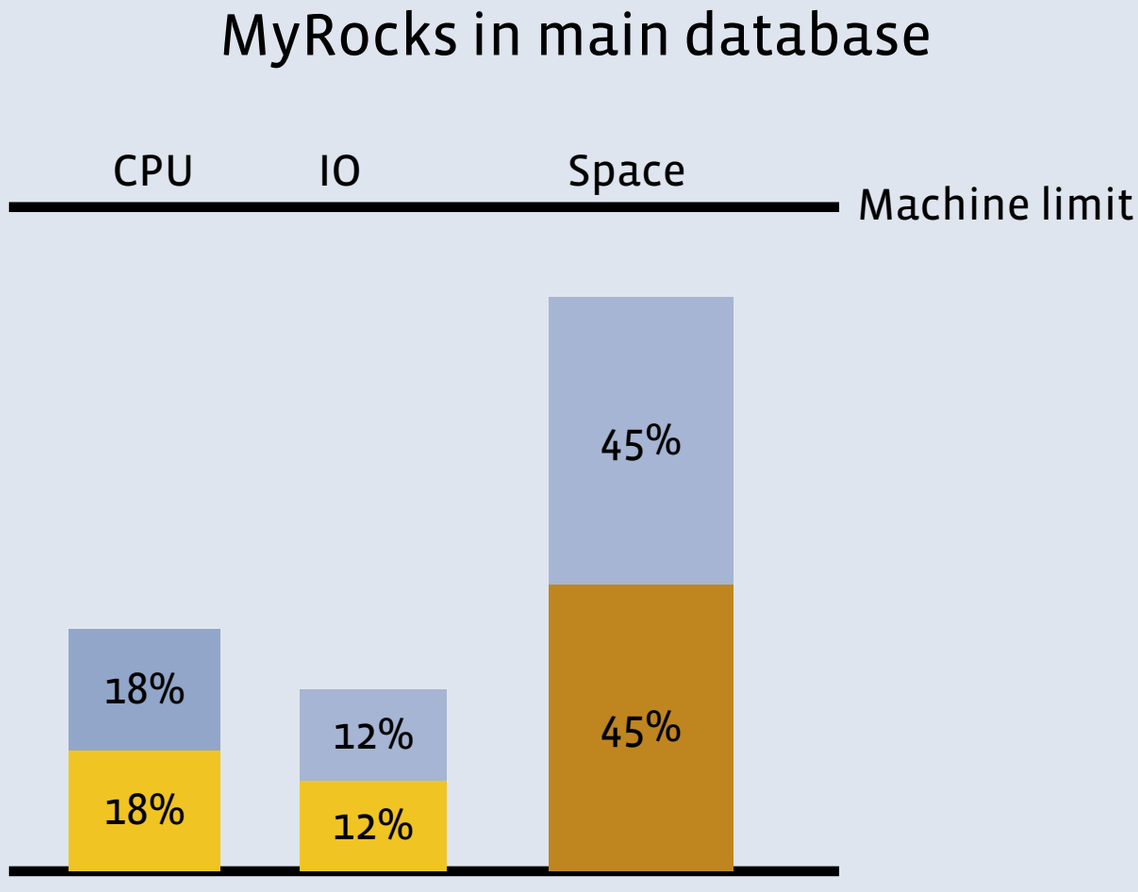
- MySQL on top of RocksDB (RocksDB storage engine)
- Open Source, distributed from MariaDB and Percona as well



http://myrocks.io/

MyRocks Goals





MyRocks goals (more details)

- Smaller space usage
  - 50% compared to compressed InnoDB at FB
- Better write amplification
  - Can use more affordable flash storage
- Fast, and small enough CPU usage with general purpose workloads
  - Large enough data that don’t fit in RAM
  - Point lookup, range scan, full index/table scan
  - Insert, update, delete by point/range, and occasional bulk inserts
  - Same or even smaller CPU usage compared to InnoDB at FB
  - Make it possible to consolidate 2x more instances per machine

MyRocks features

- Clustered Index (same as InnoDB)
- Bloom Filter and Column Family
- Transactions, including consistency between binlog and RocksDB
- Faster data loading, deletes and replication
- Dynamic Options
- TTL
- Online logical and binary backup

Facebook MySQL related teams

- Production Engineering (SRE)
  - MySQL Production Engineering
  - Data Performance
- Software Engineering
  - RocksDB Engineering
  - MySQL Engineering
    - MyRocks
    - Others (Replication, Client, InnoDB, etc)

Relationship between MySQL PE and SWE

- SWE does MySQL server upgrade
  - Upgrading MySQL revision with several FB patches
  - Hot fixes
- SWE participates in PE oncall
  - Investigating issues that might have been caused by MySQL server codebase
  - Core files analysis
- Can submit diffs each other
- “Hackaweek” to work on shorter term tasks for a week

MyRocks migration -- Technical Challenges

- Initial Migration
  - Creating MyRocks instances without downtime
  - Loading into MyRocks tables within reasonable time
  - Verifying data consistency between InnoDB and MyRocks

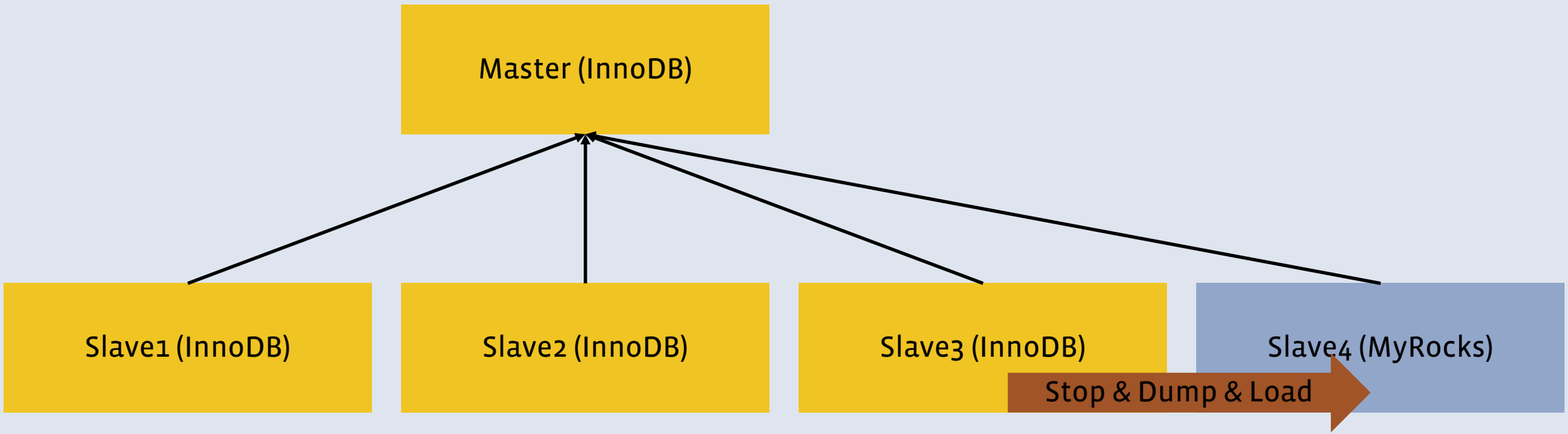
- Continuous Monitoring
  - Resource Usage like space, iops, cpu and memory
  - Query plan outliers
  - Stalls and crashes

MyRocks migration -- Technical Challenges (2)

- When running MyRocks on master
  - RBR (Row based binary logging)
  - Removing queries relying on InnoDB Gap Lock
  - Robust XA support (binlog and RocksDB)

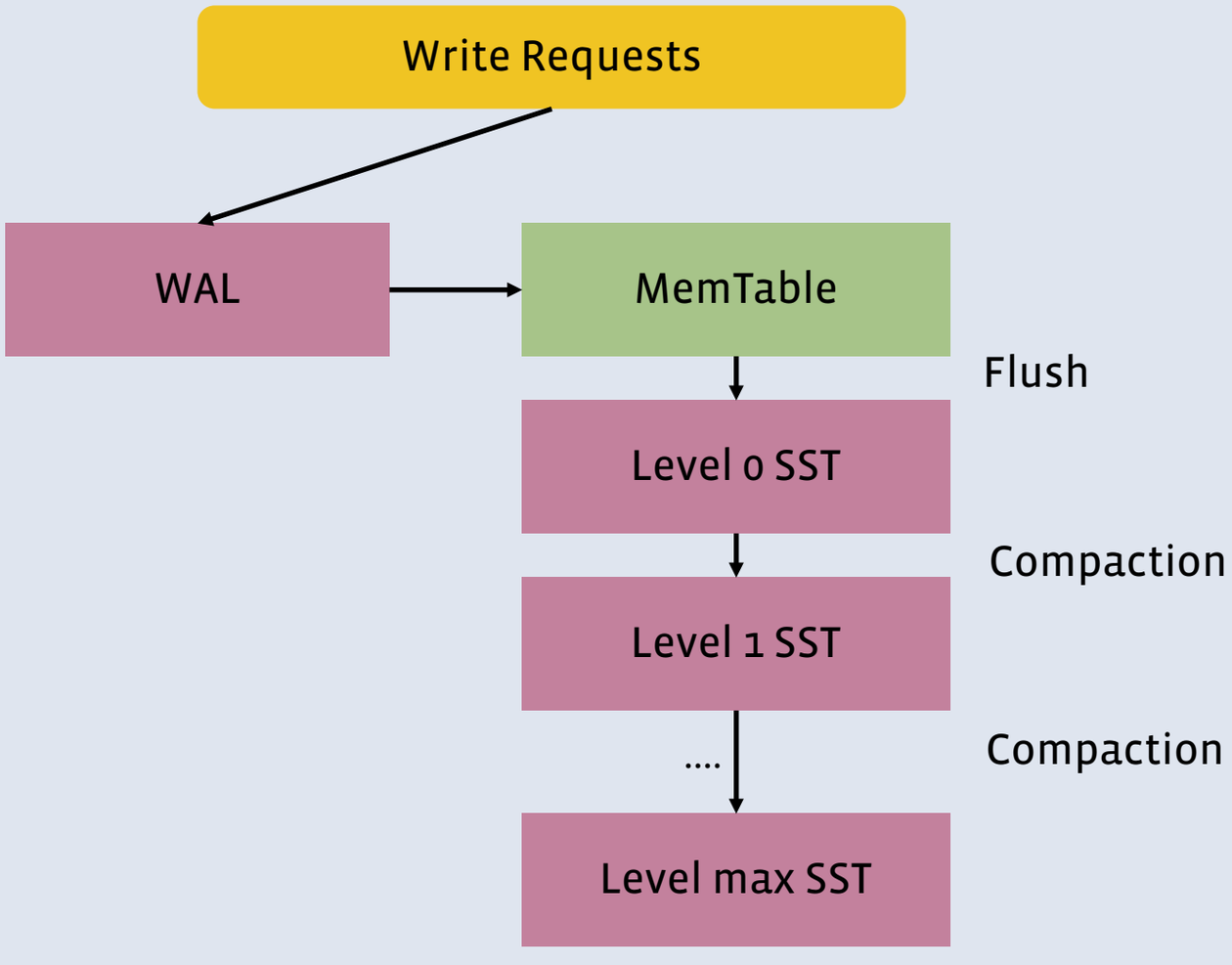
Creating first MyRocks instance without downtime

- Picking one of the InnoDB slave instances, then starting logical dump and restore
  - Stopping one slave does not affect services

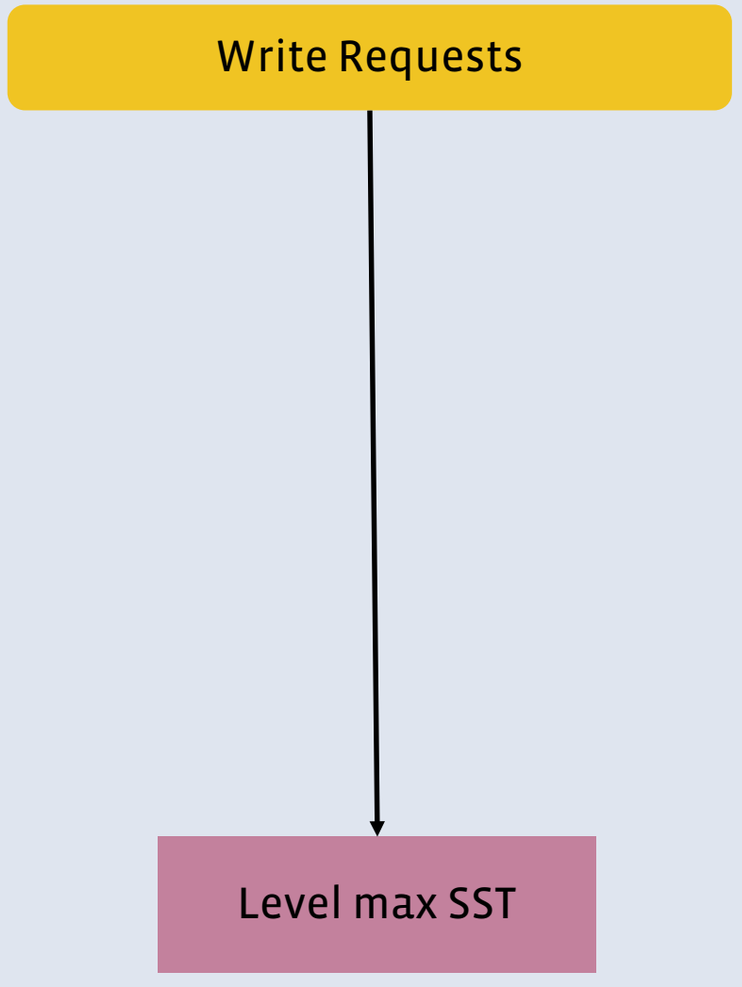


Faster Data Loading

Normal Write Path in MyRocks/RocksDB



Faster Write Path



"SET SESSION rocksdb_bulk_load=1;"
Original data must be sorted by primary key

Data migration steps

- Dst) Create table … ENGINE=ROCKSDB; (creating MyRocks tables with proper column families)
- Dst) ALTER TABLE DROP INDEX; (dropping secondary keys)
- Src) STOP SLAVE;
- mysqldump –host=innodb-host --order-by-primary | mysql –host=myrocks-host –init-command=“set sql_log_bin=0; set rocksdb_bulk_load=1”
- Dst) ALTER TABLE ADD INDEX; (adding secondary keys)
- Src, Dst) START SLAVE;

Data Verification

- MyRocks/RocksDB is relatively new database technology
- Might have more bugs than robust InnoDB
- Ensuring data consistency helps avoid showing conflicting results

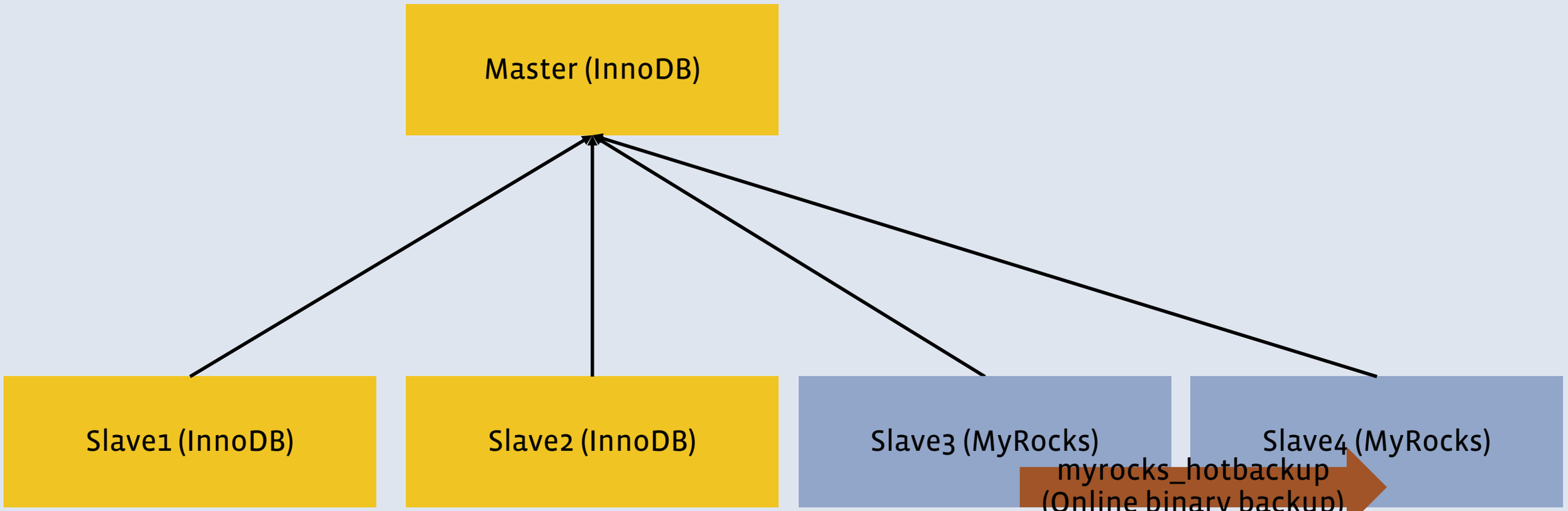
Verification tests

- Index count check between primary key and secondary keys
  - If any index is broken, it can be detected
  - SELECT ‘PRIMARY’, COUNT(*) FROM t FORCE INDEX (PRIMARY)
    UNION SELECT ‘idx1’, COUNT(*) FROM t FORCE INDEX (idx1)
  - Can’t be used if there is no secondary key
- Index stats check
  - Checking if “rows” show SHOW TABLE STATUS is not far different from actual row count
- Checksum tests w/ InnoDB
  - Comparing between InnoDB instance and MyRocks instance
  - Creating a transaction consistent snapshot at the same GTID position, scan, then compare checksum
- Shadow correctness check
  - Capturing read traffics

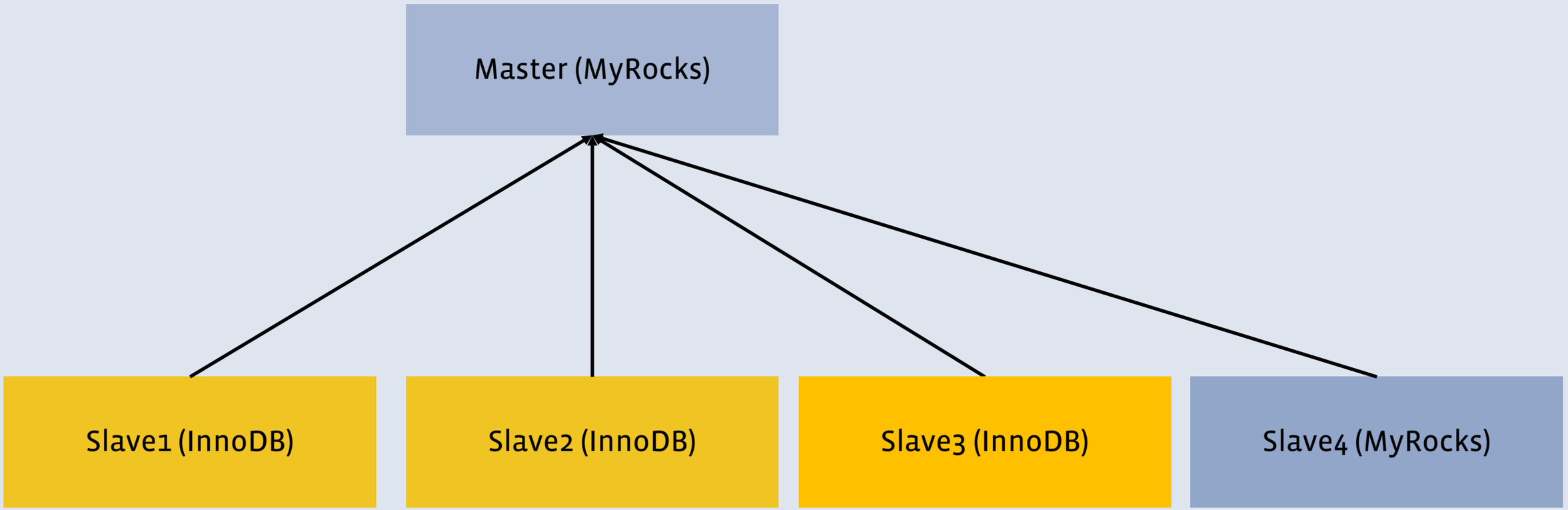
Shadow traffic tests

- We have a shadow test framework
  - MySQL audit plugin to capture read/write queries from production instances
  - Replaying them into shadow master instances
- Shadow master tests
  - Client errors
  - Rewriting queries relying on Gap Lock
    - gap_lock_raise_error=1, gap_lock_write_log=1

Creating second MyRocks instance without downtime



Promoting MyRocks as a master



Crash Safety

- Crash Safety makes operations much easier
  - Just restarting failed instances can restart replication
  - No need to rebuild entire instances, as long as data is there
- Crash Safe Slave
  - Crash Safe Master
  - 2PC (binlog and WAL)

Master crash safety settings

	sync-binlog	rocksdb-flush-log-at-trx-commit	rocksdb-enable-2pc	rocksdb-wal-recovery-mode
No data loss on unplanned machine reboot	1 (default)	1 (default)	1 (default)	1 (default)
No data loss on mysqld crash & recovery	0	2	1	2
No data loss if always failover	0	2	0	2

Notable issues fixed during migration

- Lots of “Snapshot Conflict” errors
  - Because of implementation differences in MyRocks (PostgreSQL Style snapshot isolation)
  - Setting tx-isolation=READ-COMMITTED solved the problem
- Slaves stopped with I/O errors on reads
  - We switched to make MyRocks abort on I/O errors for both reads and writes
- Index statistics bugs
  - Inaccurate row counts/length on bulk loading -> fixed
  - Cardinality was not updated on fast index creation -> fixed
- Crash safety bugs and some crash bugs in MyRocks/RocksDB

Preventing stalls

- Heavy writes cause lots of compactions, which may cause stalls

- Typical write stall cases
  - Online schema change
  - Massive data migration jobs that write lots of data

- Use fast data loading whenever possible
  - InnoDB to MyRocks migration can utilize this technique
  - Can be harder for data migration jobs that write into existing tables

When write stalls happen

- Estimated number of pending compaction bytes exceeded X bytes
  - soft|hard_pending_compaction_bytes, default 64GB
- Number of L0 files exceeded level0_slowdown|stop_writes_trigger (default 10)
- Number of unflushed number of MemTables exceeded max_write_buffer_number (default 4)

- All of these incidents are written to LOG as WARN level
- All of these options apply to each column family

What happens on write stalls

- Soft stalls
  - COMMIT takes longer time than usual
  - Total estimated written bytes to MemTable is capped to rocksdb_delayed_write_rate, until slowdown conditions are cleared
    - Default is 16MB/s (previously 2MB/s)
- Hard stalls
  - All writes are blocked at COMMIT, until stop conditions are cleared

Mitigating Write Stalls

- Speed up compactions
  - Use faster compression algorithm (LZ4 for higher levels, ZSTD in the bottommost)
  - Increase rocksdb_max_background_compactions
- Reduce total bytes written
  - But avoid using too strong compression algorithm on upper levels
  - Use more write efficient compaction algorithm
    - compaction_pri=kMinOverlappingRatio
- Delete files slowly on Flash
  - Deleting too many large files cause TRIM stalls on Flash
  - MyRocks throttles sst file deletes by 64MB/s by default
  - Binlog file deletions should be slowed down as well

Monitoring

- MyRocks files
- SHOW ENGINE ROCKSDB STATUS
- SHOW ENGINE ROCKSDB TRANSACTION STATUS
- LOG files
- information_schema tables
- sst_dump tool
- ldb tool

SHOW ENGINE ROCKSDB STATUS

- Column Family Statistics, including size, read and write amp per level
- Memory usage

```
*************************** 7. row ***************************
  Type: CF_COMPACTION
  Name: default
Status:
** Compaction Stats [default] **
Level    Files   Size(MB) Score Read(GB)  Rn(GB) Rnp1(GB) Write(GB) Wnew(GB) Moved(GB) W-Amp Rd(MB/s) Wr(MB/s) Comp(sec) Comp(cnt) Avg(sec) KeyIn KeyDrop
----------------------------------------------------------------------------------------------------------------------------------------------------------
  L0      2/0      51.58   0.5      0.0     0.0      0.0       0.3      0.3       0.0   0.0      0.0     40.3          7         10    0.669       0      0
  L3      6/0     109.36   0.9      0.7     0.7      0.0       0.6      0.6       0.0   0.9     43.8     40.7         16          3    5.172   7494K   297K
  L4     61/0    1247.31   1.0      2.0     0.3      1.7       2.0      0.2       0.0   6.9     49.7     48.5         41          9    4.593     15M   176K
  L5    989/0   12592.86   1.0      2.0     0.3      1.8       1.9      0.1       0.0   7.4      8.1      7.4        258          8   32.209     17M   726K
  L6   4271/0  127363.51   0.0      0.0     0.0      0.0       0.0      0.0       0.0   0.0      0.0      0.0          0          0    0.000       0      0
 Sum   5329/0  141364.62   0.0      4.7     1.2      3.5       4.7      1.2       0.0  17.9     15.0     15.0        321         30   10.707     41M  1200K
```

SHOW GLOBAL STATUS

```
mysql> show global status like 'rocksdb%';
+---------------------------------------+-------------+
| Variable_name                         | Value       |
+---------------------------------------+-------------+
| rocksdb_rows_deleted                  | 216223      |
| rocksdb_rows_inserted                 | 1318158     |
| rocksdb_rows_read                     | 7102838     |
| rocksdb_rows_updated                  | 1997116     |
....
| rocksdb_bloom_filter_prefix_checked   | 773124      |
| rocksdb_bloom_filter_prefix_useful    | 308445      |
| rocksdb_bloom_filter_useful           | 10108448    |
....
```

Summary

- We started migrating from InnoDB to MyRocks in our main database
  - We run both master and slaves in production
- Major motivation was to save space
- Online data correctness check tool helped to find lots of data integrity bugs and prevented from deploying inconsistent instances in production
- Bulk loading greatly reduced compaction stalls
- Transactions and crash safety are supported

facebook