Using TokuDB

A Guided Walk Through a TokuDB Implementation

Presented by Jon Tobin
jon@tokutek.com

Tokutek®

Performance Database Company

Tokutek, Inc.
57 Bedford Road, Suite 101
Lexington, MA 02420

www.tokutek.com

Tokutek Customers

Agenda

- Storage Engine Overview
- Why's TokuDB Different
- Demo Environment Overview
  - Configuration Settings
  - iiBench Overview
- Agility (Hot Schema Change)
- Compression
- Performance
- Read Free Replication

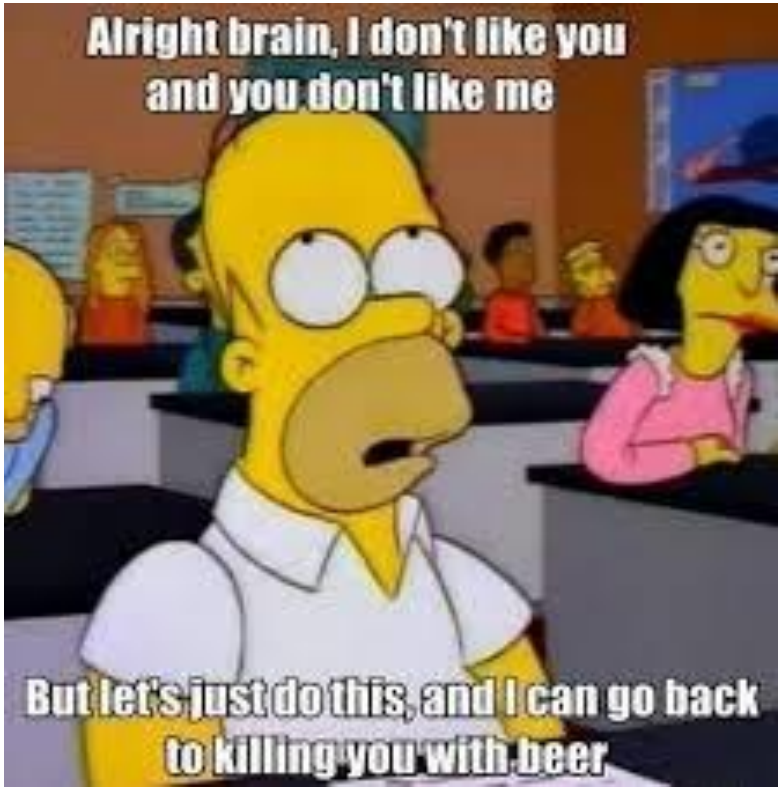
Why Change?

1. I want to increase the number of sources I ingest
2. I want to be able to follow a more agile development path
3. My data is taking up too much space
4. My current method of scaling is not sustainable
5. CapEx & OpEx for my environment are rising too quickly

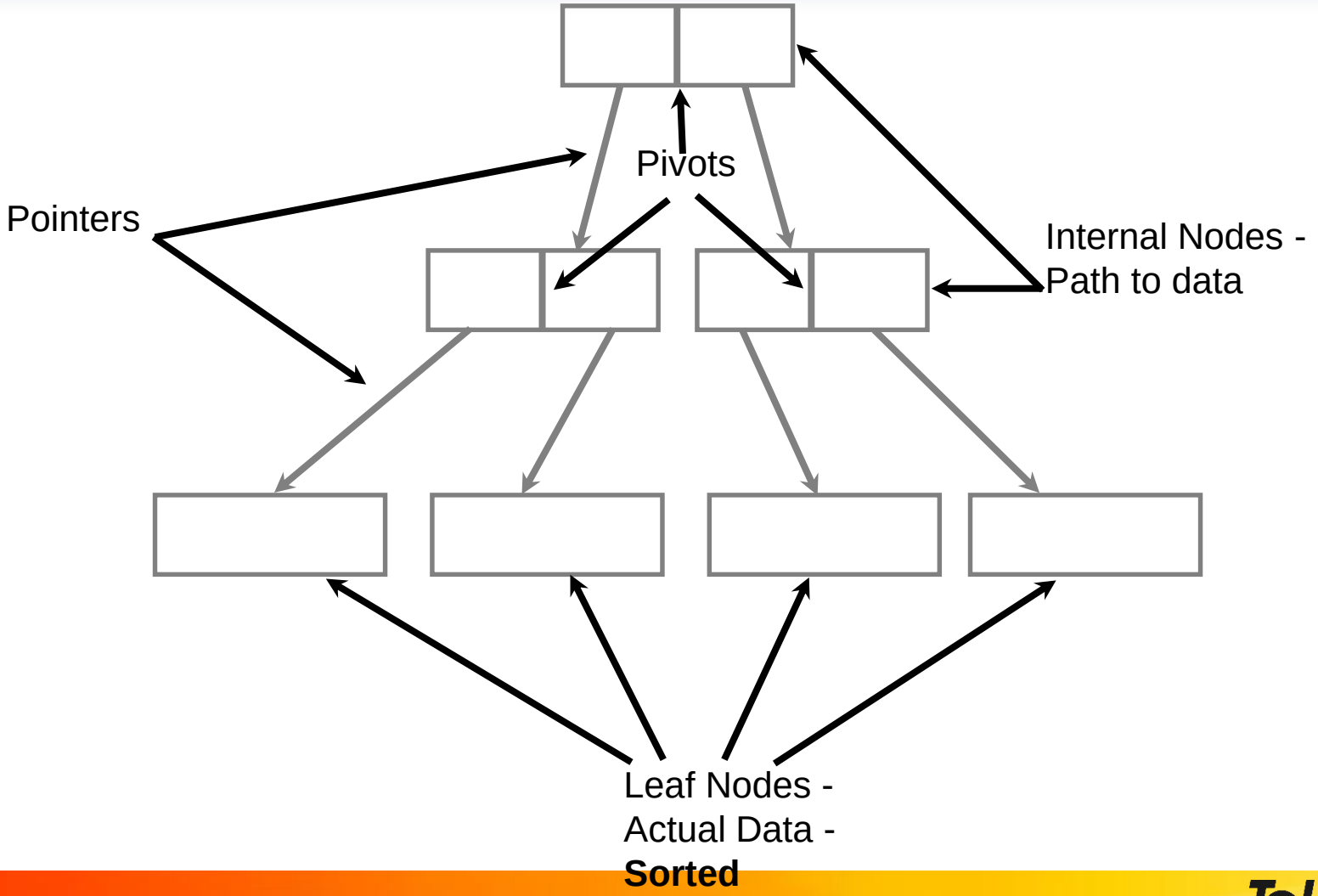
Storage Engines

	InnoDB	TokuDB
Data Structure	B-tree	Fractal Tree
Transactions	Yes	Yes
Foreign Keys	Yes	No
Compression	Yes*	Yes
Clustered Indexes	Primary Key Only	Any
Hot Schema Change	Yes*	Yes
Strength	In memory working set	> Memory performance
Weakness	Efficiency	Point Queries*

Why's TokuDB Different?



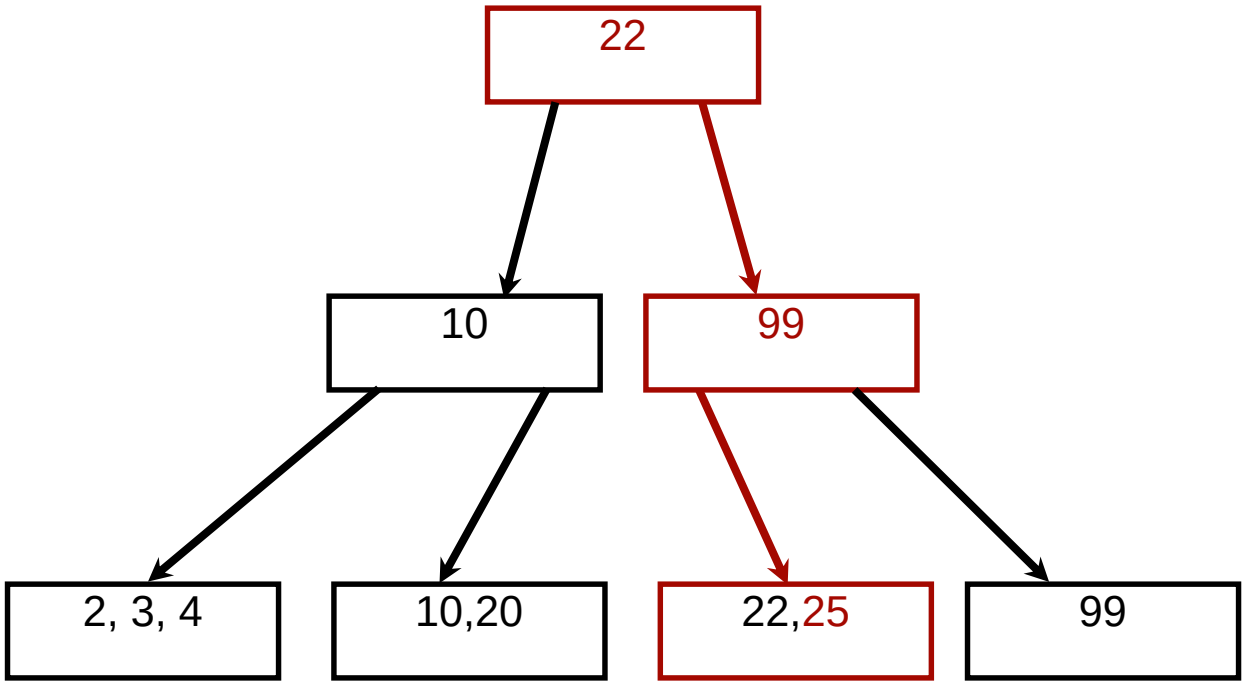
B-tree Overview - vocabulary



Tokutek

B-tree Overview - search

"Find 25"

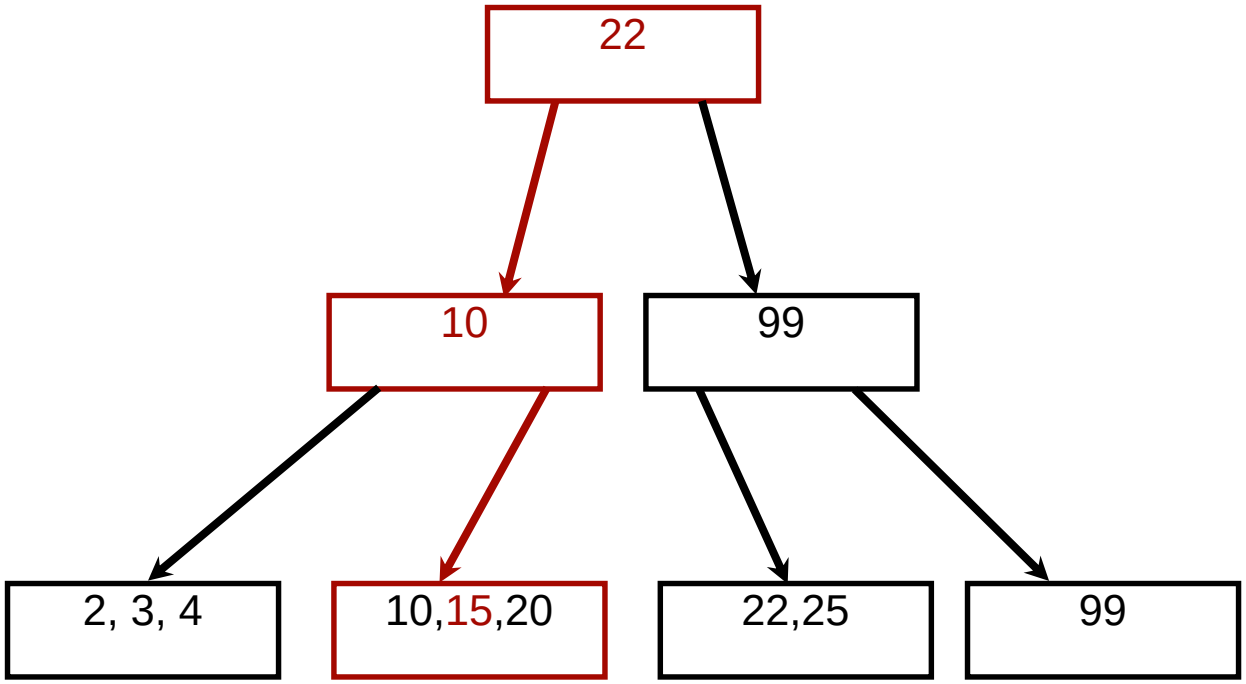


Pivot Rule >=

Tokutek

B-tree Overview - insert

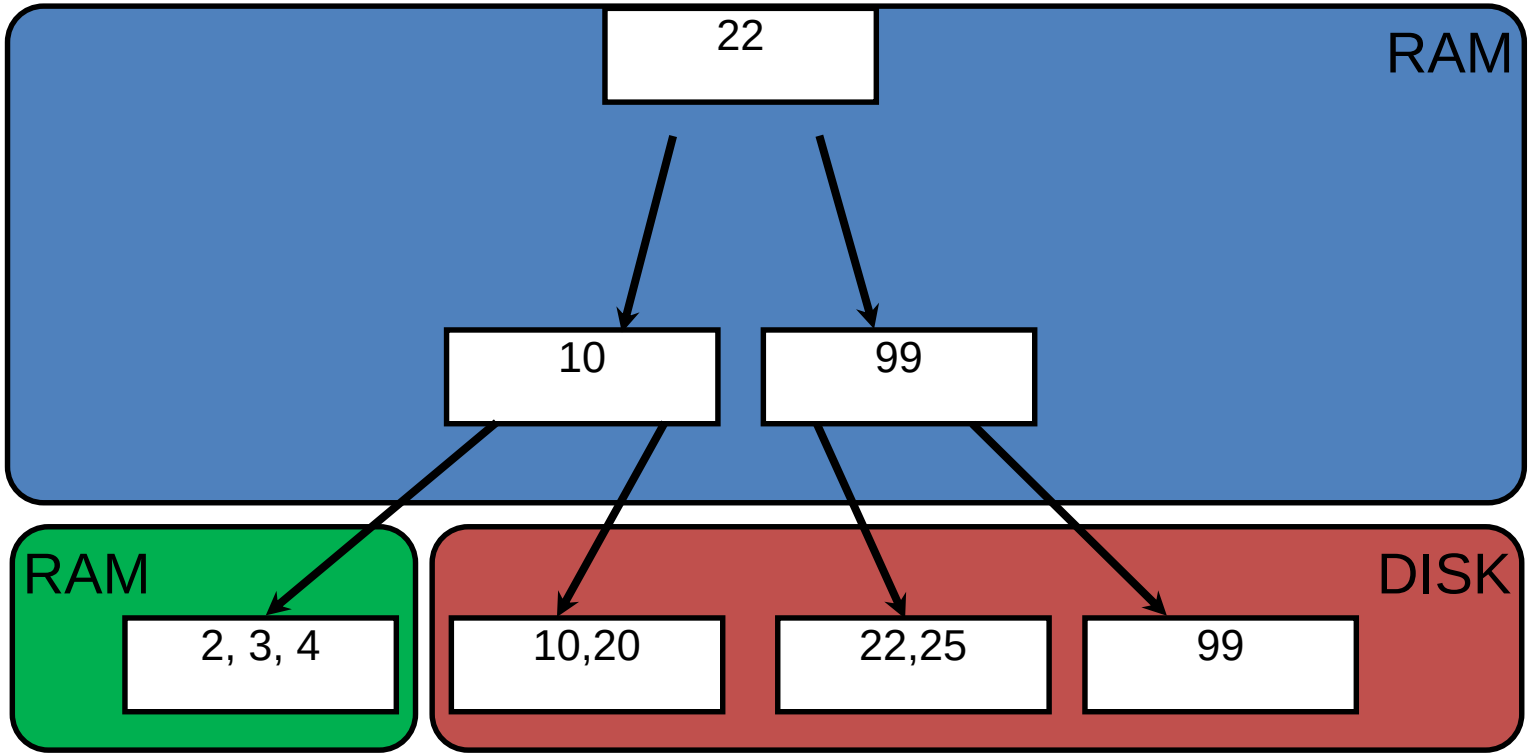
"Insert 15"



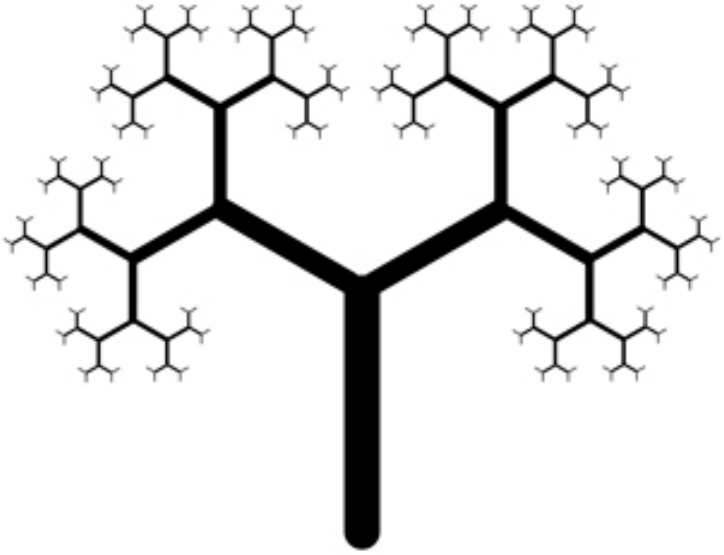
Tokutek

B-tree Overview - performance

Performance is IO limited when data > RAM, one IO is needed for each insert/update (actually it's one IO for every index on the table)

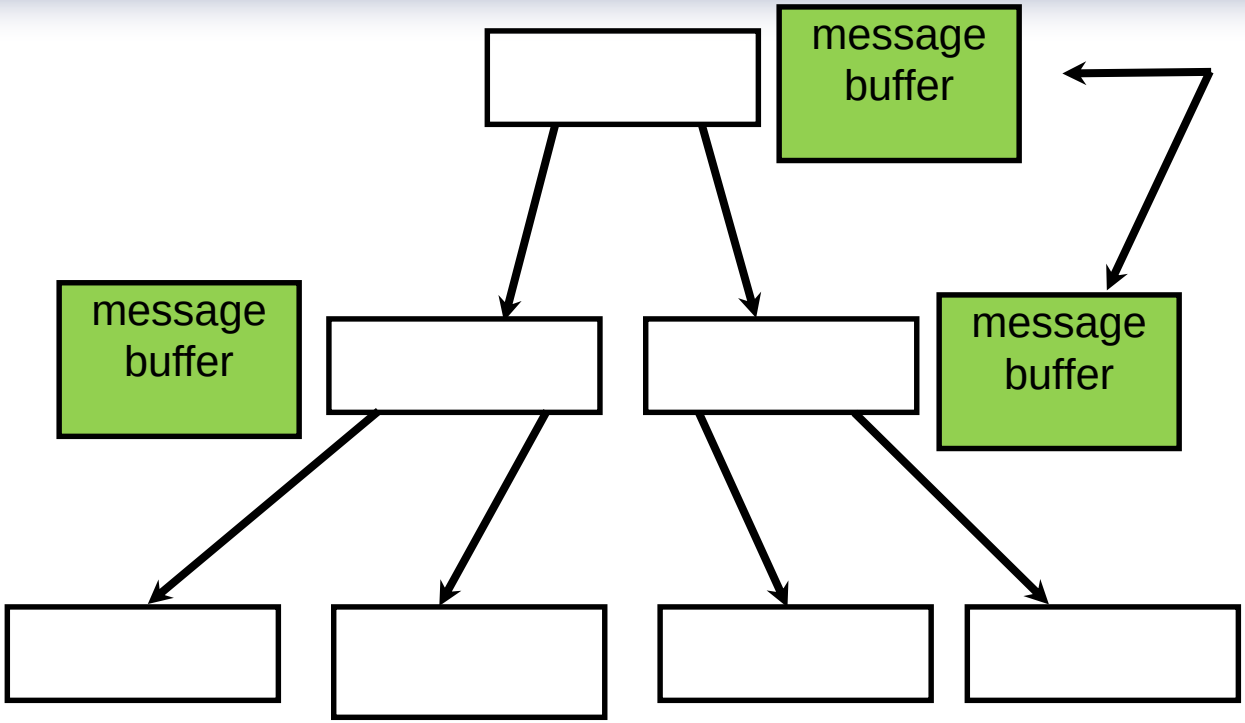


Fractal Tree Indexes



Tokutek®

Fractal Tree Indexes



All internal nodes have message buffers

As buffers overflow, they cascade down the tree

Messages are eventually applied to leaf nodes

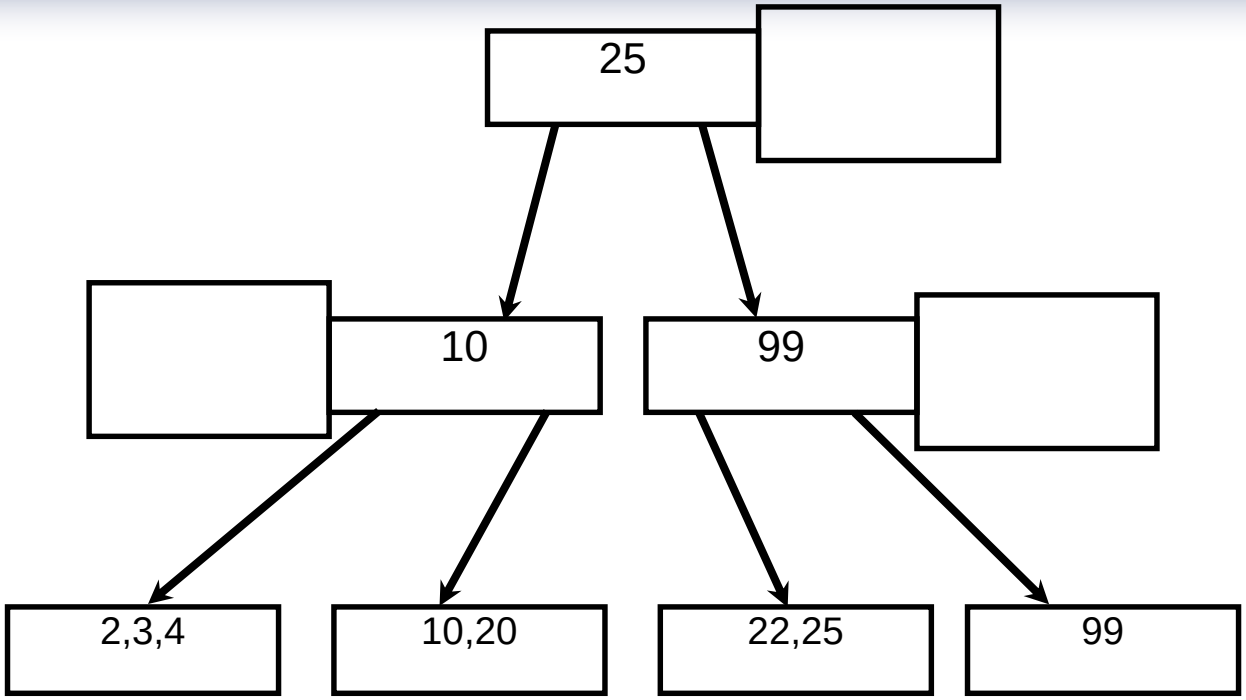
similar to B-trees

- store data in leaf nodes
- use index key for ordering

different than B-trees

- message buffers
- big nodes (4MB vs. ~16KB)

Fractal Tree Indexes - sample data

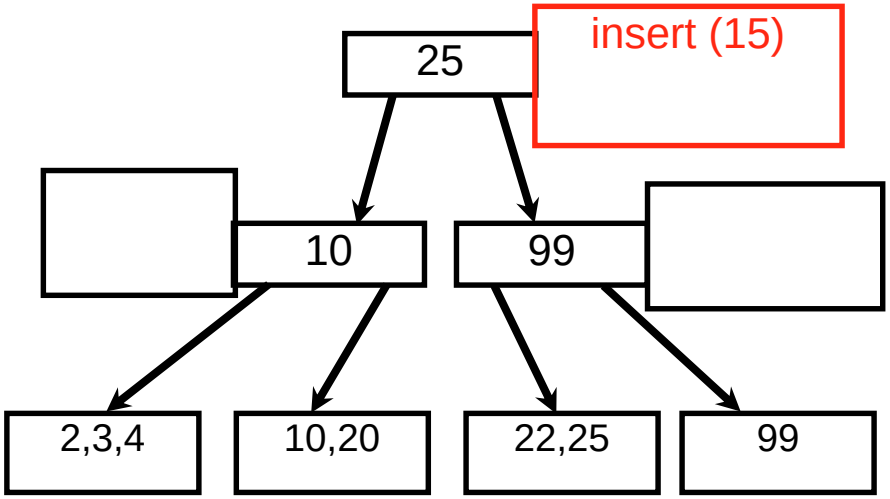


Looks a lot like a b-tree!

Tokutek

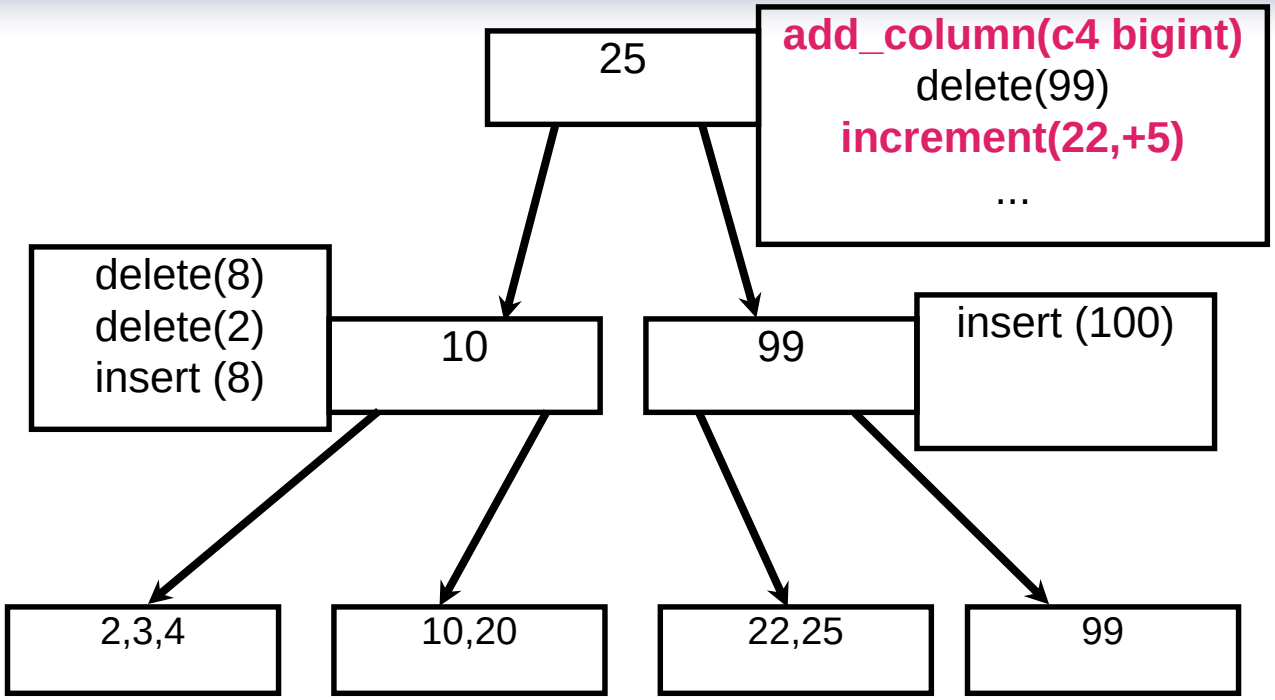
Fractal Tree Indexes - insert

insert 15;



- search operations must consider messages along the way
- messages cascade down the tree as buffers fill up
- they are eventually applied to the leaf nodes, hundreds or thousands of operations for a single IO
- CPU and cache are conserved as important data is not ejected

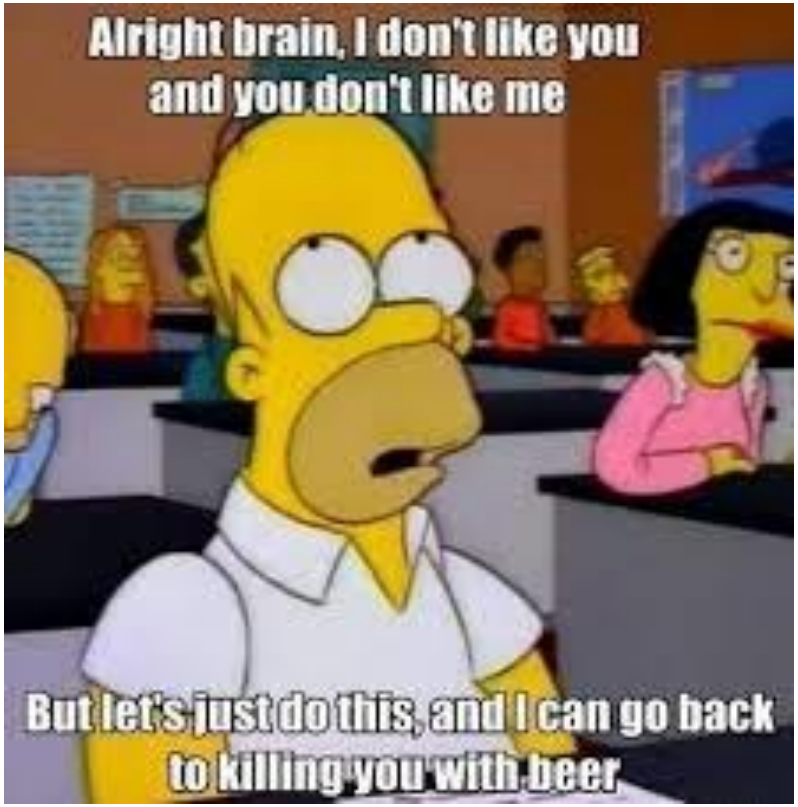
Fractal Tree Indexes - other operations



Lots of operations can be messages!

Tokutek

Demo Environment



Our Configuration

InnoDB

- Cache 64 (innodb_buffer_pool_size=64M)
- Direct IO (innodb_flush_method=0_DIRECT)
- 1 File/table (inno_files_per_table=1)
- Barracuda format (innodb_file_format)

TokuDB

- Cache 64M (tokudb_cache_size)
- Direct IO (tokudb_directio=1)

iibench Overview

- Python based benchmarking app
- Created by Tokutek
- Maintained by Mark Callaghan (facebook)
- Simulates Point of Sale environment
- Highly customizable – just pass parameters

- Schema
  - **transactionid** - int(11) NOT NULL AUTO_INCREMENT,
  - **dateandtime** - datetime DEFAULT NULL,
  - **cashregisterid** - int(11) NOT NULL,
  - **customerid** - int(11) NOT NULL,
  - **productid** - int(11) NOT NULL,
  - **Price -** float NOT NULL,
  - **data** - varchar(4000) DEFAULT NULL,

Why is iiBench Interesting?

- "ii" = indexed inserts
  - TokuDB's good at index maintenance
  - Incrementing PK
  - 3 indexes
- Batch inserts
- Batch queries
  - Amortizes transactional overhead over many operations
- Lets look "below the covers"

Agility

1. MySQL 5.6 & PT offers online schema change
   - `create index` & `drop index` are least expensive operations
   - Most other operations require a table copy
   - Resource intensive (CPU, RAM, I/O)
2. Master – Slave Switching
3. TokuDB
   - Hot column addition, deletion, expansion
     - Expand char, varchar, varbinary & integer
     - No table copy
   - Background index creation
   - HCADER can do one operation at a time

Online Index Creation

- Don't use `ALTER TABLE` for hot operation
- Turn on 'online index creation'
  - `SET tokudb_create_index_online=ON`
- Create the index
  - `CREATE INDEX index_name ON table (field_name(s)`
- `SHOW PROCESSLIST` will display progress
  - Will likely be slower that offline index creation

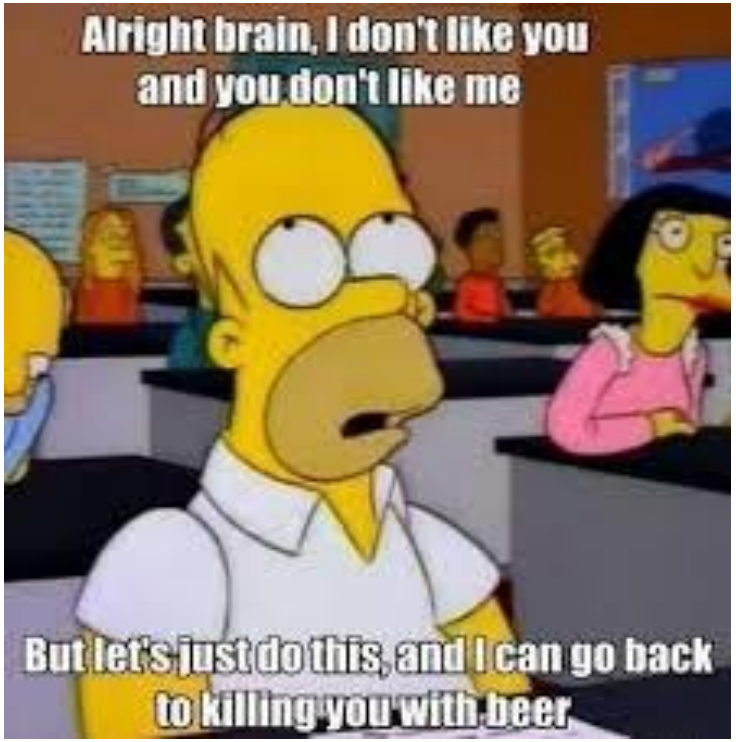
Good to Know

- HCADER takes 1 operation at a time
  - More than one will result in SLOW operation (resource utilization)
- You can disable slow alters
  - `tokudb_disable_slow_alter=ON`
  - Will pass an error back to MySQL if slow operation is passed

Your Turn

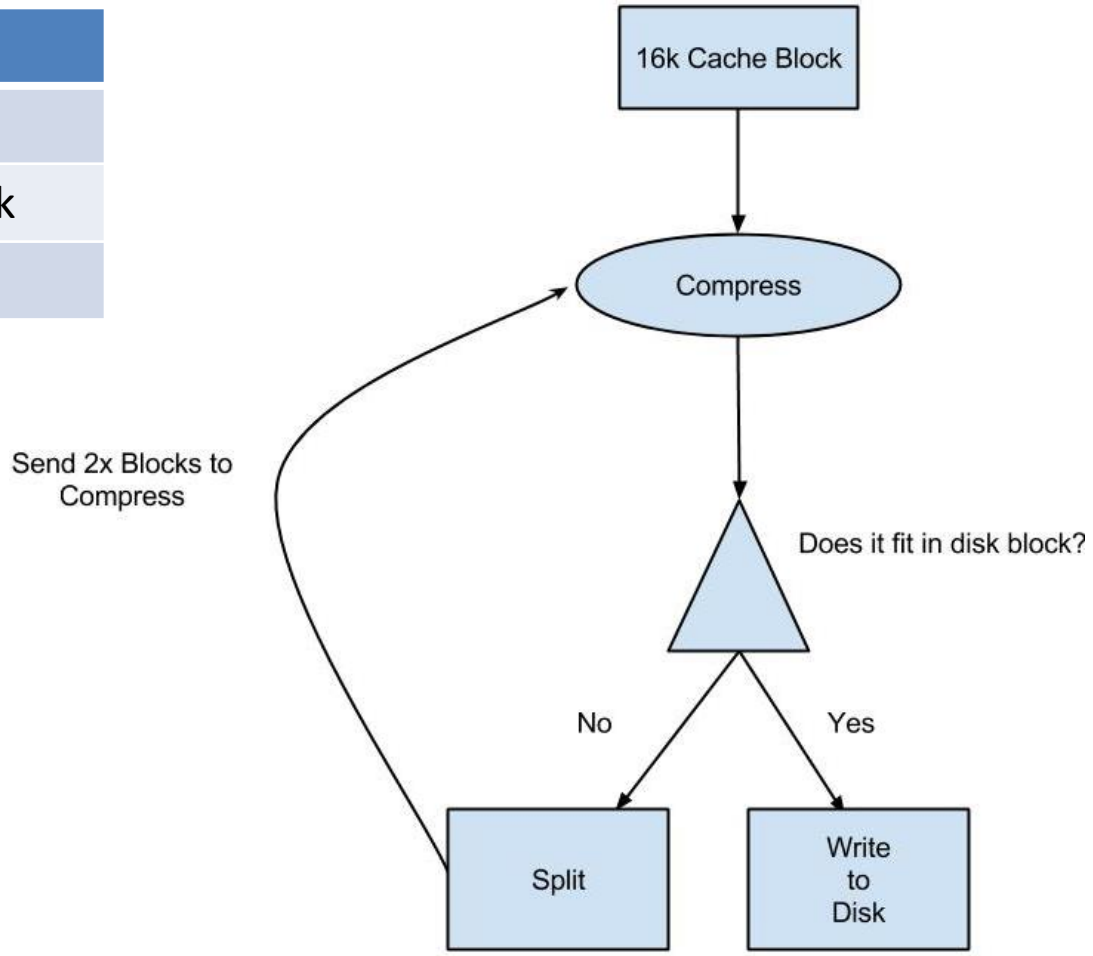
Run the Agility Lab now

Compression



Inno Compression

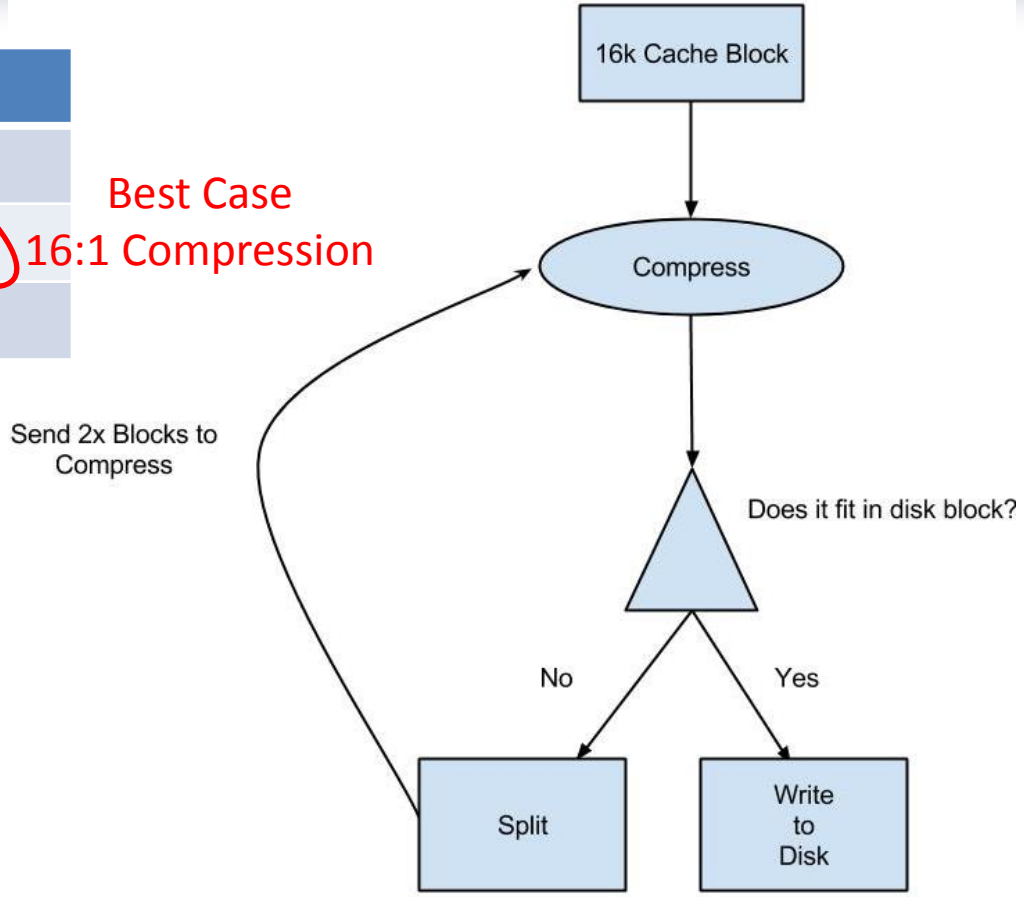
Inno	
Cache block	16k
Disk block	8k, 4k, 2k, 1k
Algorithm	zlib



Inno Compression

Inno	
Cache block	16k
Disk block	8k, 4k, 2k, 1k
Algorithm	zlib

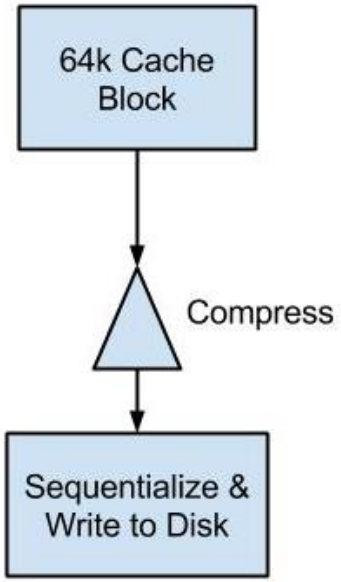
Best Case
16:1 Compression



*5.6 new features – adaptive padding, compression level

TokuDB Compression

- Fractal Tree was made to compress
- More data = better compression
- No split/recompress = better performance
- Decompression may lead to addt'l query latency



TokuDB	
Cache block	64k (tuneable)
Disk block	4MB (default)
Algorithm	Quicklz/zlib(d)/lzma

Compression Thoughts

- Good
  - Space savings is a big win for most use cases
  - When compression saves and IO it's well worth tradeoffs
- Bad
  - Compression increases latency on the way down (not a big deal) and up (query latency)
  - Taxes CPU
- Ugly
  - It's all about managing the quid-pro-quo
  - Unpredictability can have bad side effects

Things to think about

- `tokudb_read_block_size`
  - Will affect compression
  - May also speed up reads that fetch a small amount of data
- `Tokudb_row_format`
  - Will change compression for newly written data
  - `OPTIMIZE TABLE` needs to be run to change entire index
- CPU Utilization
  - If CPU utilization gets too high, try changing the compression algorithm to something more light weight
  - If workload is read heavy, reducing the `read_block_size` may also help
- Tokudb_directio
  - On = usually means more consistent performance esp write heavy
  - Off = uses OS buffers to store COMPRESSED data. For majority read workloads with high compression ratios, **MAY** yield serious performance gains

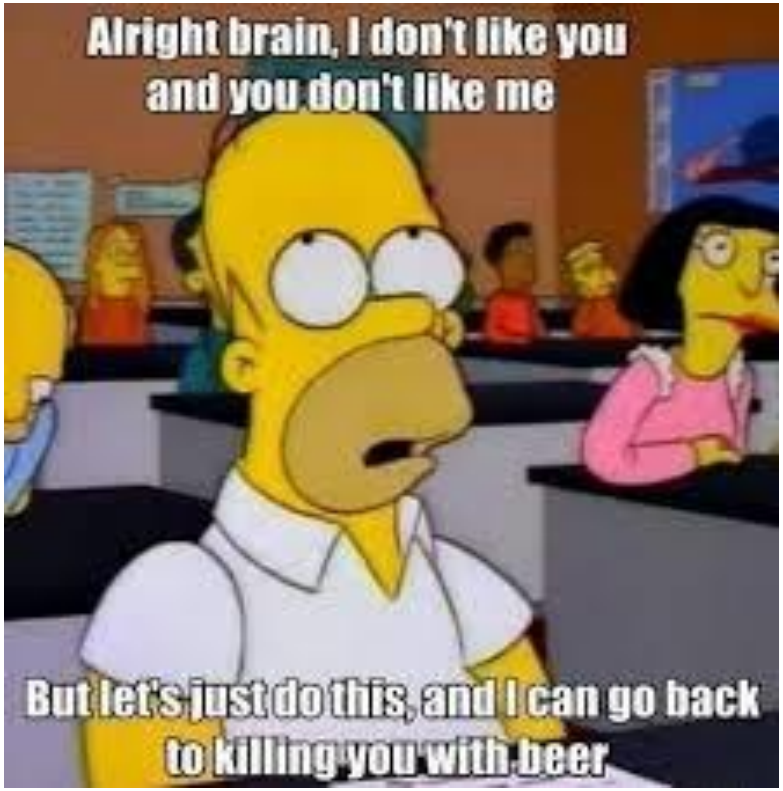
Compress EngStatus

- Tokudb_LEAF_COMPRESSION_TO_MEMORY_SECONDS
- Tokudb_LEAF_SERIALIZATION_TO_MEMORY_SECONDS
- Tokudb_LEAF_DECOMPRESSION_TO_MEMORY_SECONDS
- Tokudb_LEAF_DESERIALIZATION_TO_MEMORY_SECONDS
- Tokudb_NONLEAF_COMPRESSION_TO_MEMORY_SECONDS
- Tokudb_NONLEAF_SERIALIZATION_TO_MEMORY_SECONDS
- Tokudb_NONLEAF_DECOMPRESSION_TO_MEMORY_SECONDS
- Tokudb_NONLEAF_DESERIALIZATION_TO_MEMORY_SECONDS

Your Turn

Compression Lab

Performance



Performance

InnoDB

Pro

- In memory performance is exceptional
- Point query performance is good

Con

- B-trees rely on storage for performance when working set >memory
- Performance decreases as tables (keys) fragment

TokuDB

Pro

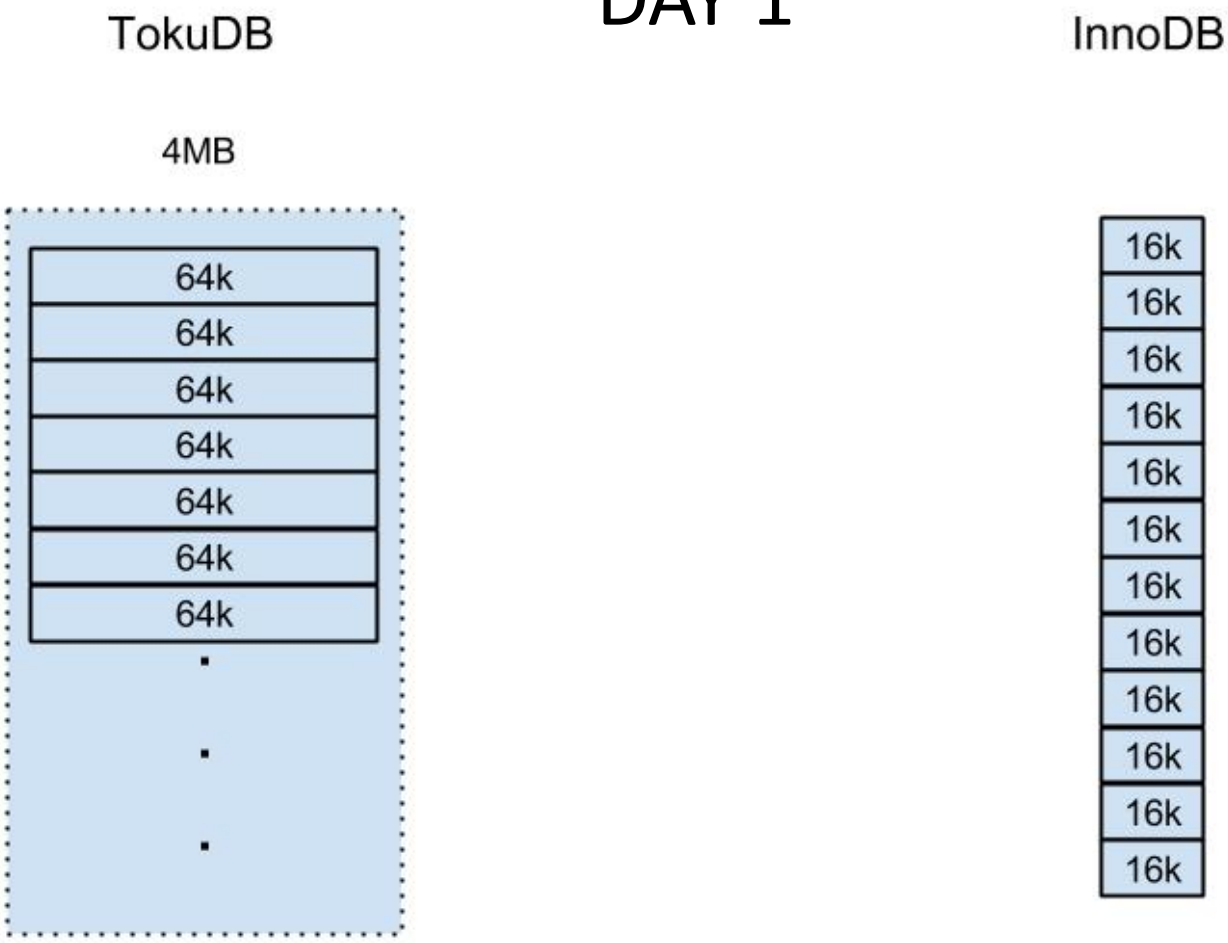
- Performance is consistently very good
- "No fragmentation"
- Range query performance is great

Con

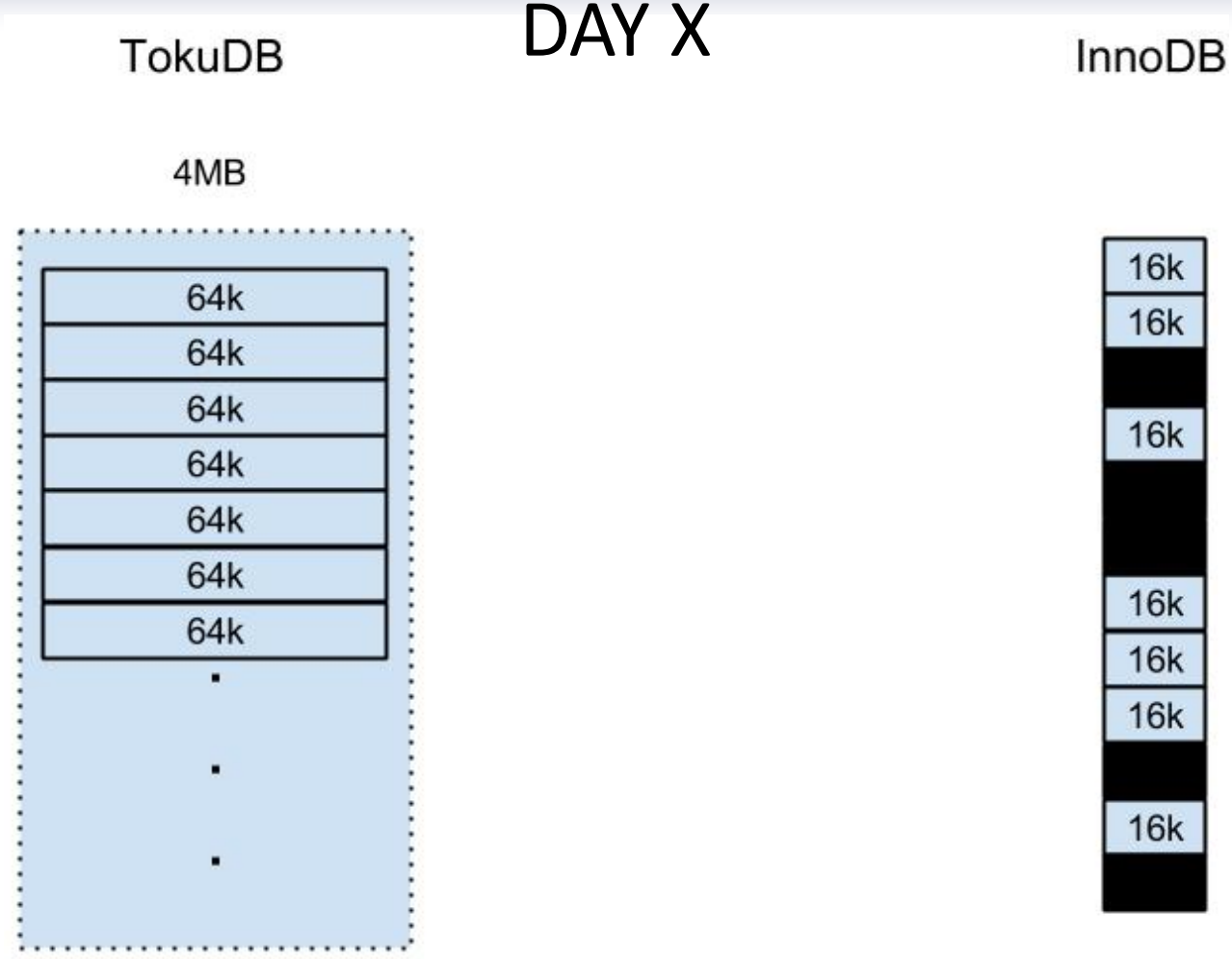
- Additional query latency due to data decompression (μs)
- Point queries can be expensive (tuneable)

TokuDB & InnoDB On Disk

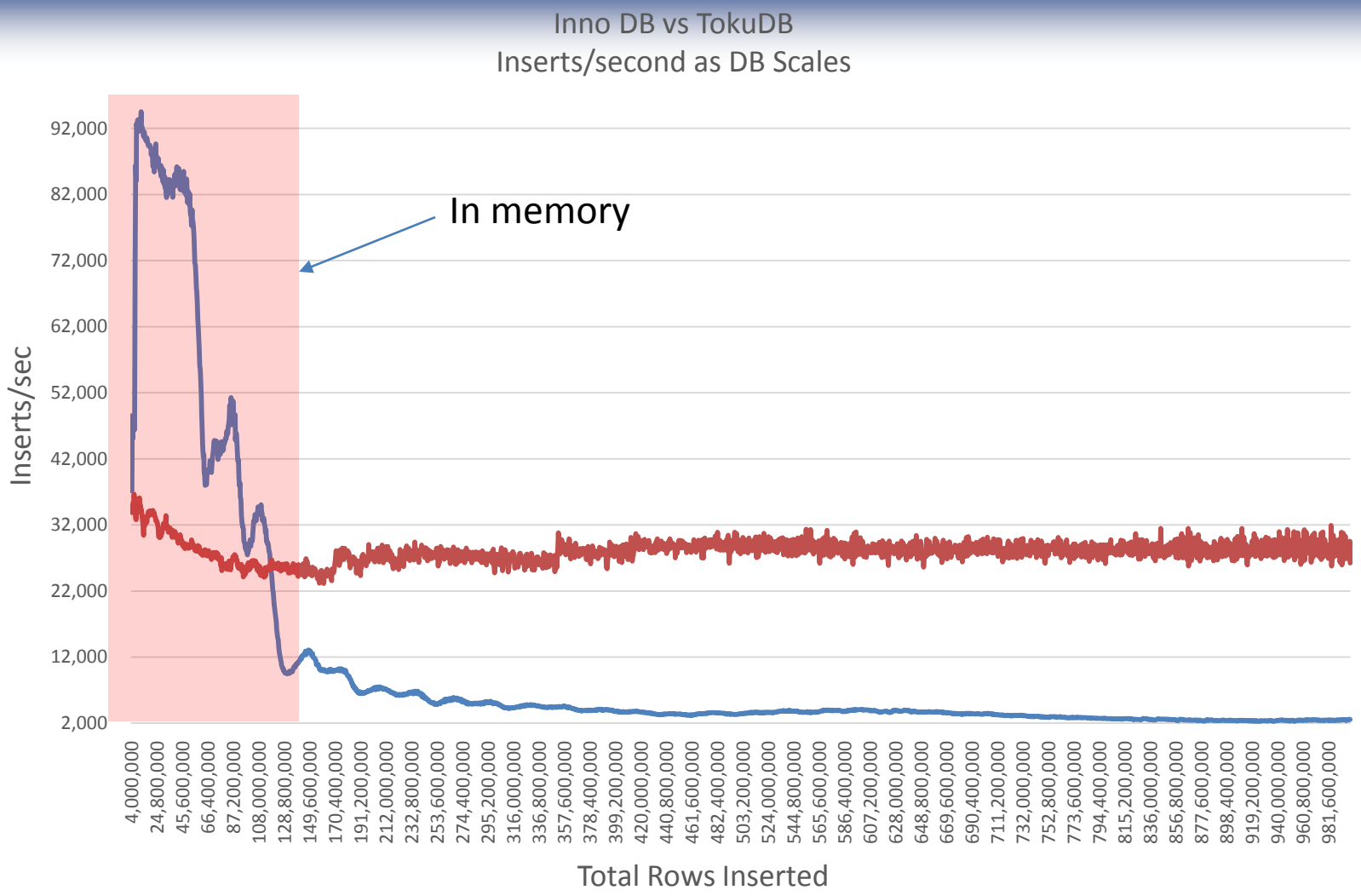
DAY 1



TokuDB & InnoDB On Disk



Performance



Performance EngStatus

- Tokudb_CHECKPOINT_PERIOD
- Tokudb_CHECKPOINT_LAST_BEGAN
- Tokudb_CHECKPOINT_LAST_COMPLETE_BEGAN
- Tokudb_CHECKPOINT_LAST_COMPLETE_ENDED
- Tokudb_CHECKPOINT_TAKEN
- Tokudb_CHECKPOINT_FAILED
- Tokudb_CHECKPOINT_BEGIN_TIME
- Tokudb_CHECKPOINT_LONG_BEGIN_TIME
- Tokudb_CHECKPOINT_LONG_BEGIN_COUNT

Your Turn

Performance Lab

Read Free Replication

- Our newest feature in v7.5

WHY

- Master = multithreaded
- Replication = single threaded (5.5)
  multi threaded (5.6)
- Master can process more concurrent threads
  - Master can do more work/unit of time
- Slave can be bottlenecked by two things working together
  - Time to process a single 'operation'
  - Replication "bandwidth"
- Slave application overhead is high
  - Constantly trying to confirm it's "before image" is consistent with the master (reading before writing)

Read Free Replication

"Beer, the cause and solution to all of life's problems"

-Homer Simpson

Tokutek to the Rescue

- Only changes (writes) are replicated
- Fractal Tree writes are cheap & fast
- Remove the read and we can empty the replication stream faster

Goal: Reduce replication related I/O on slave to nada in between checkpoints while reducing (hopefully eliminating) lag

More bandwidth for read scaling

Make Sense?

How do we use it?

Master:

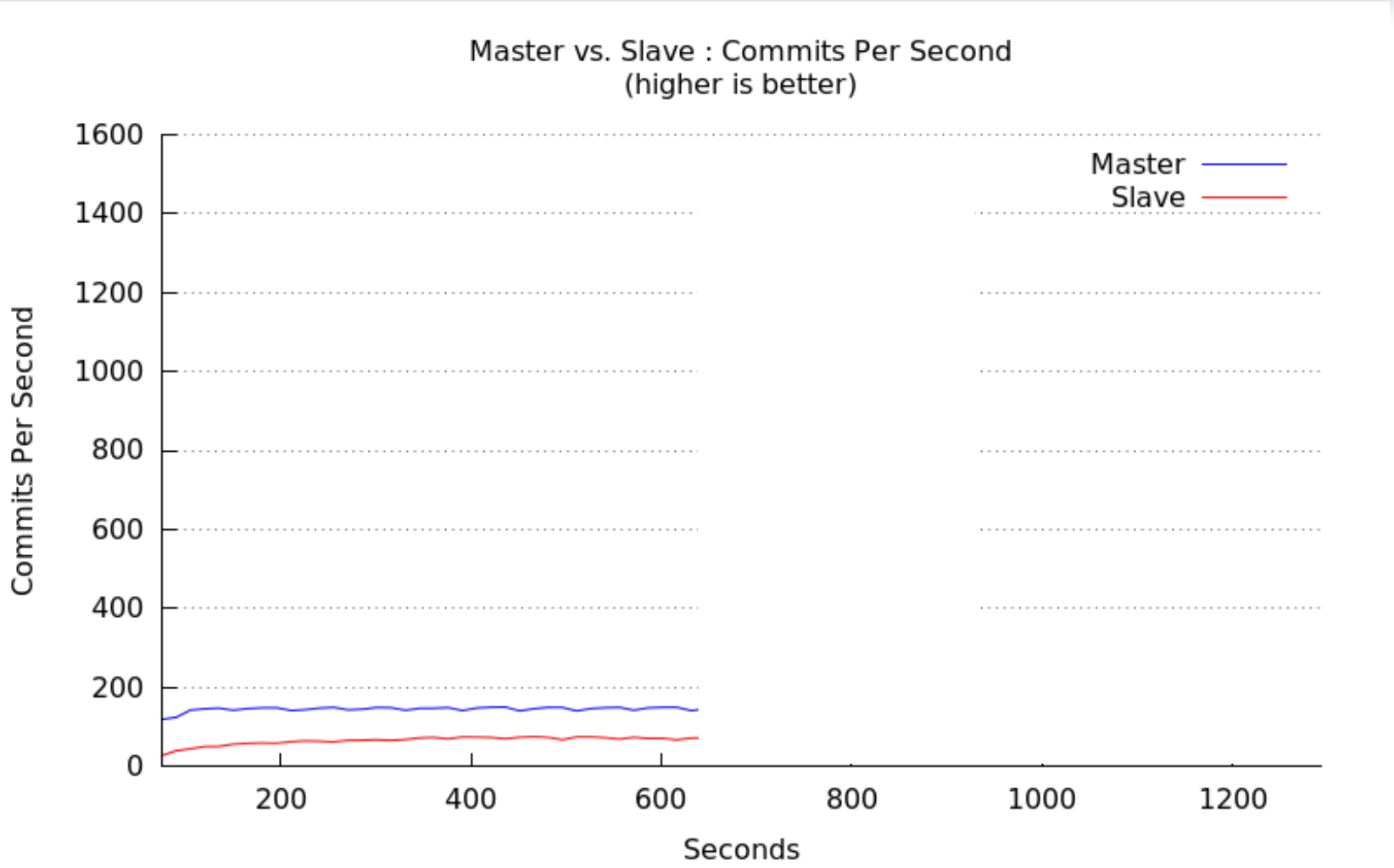
- Row based replication

Slave:

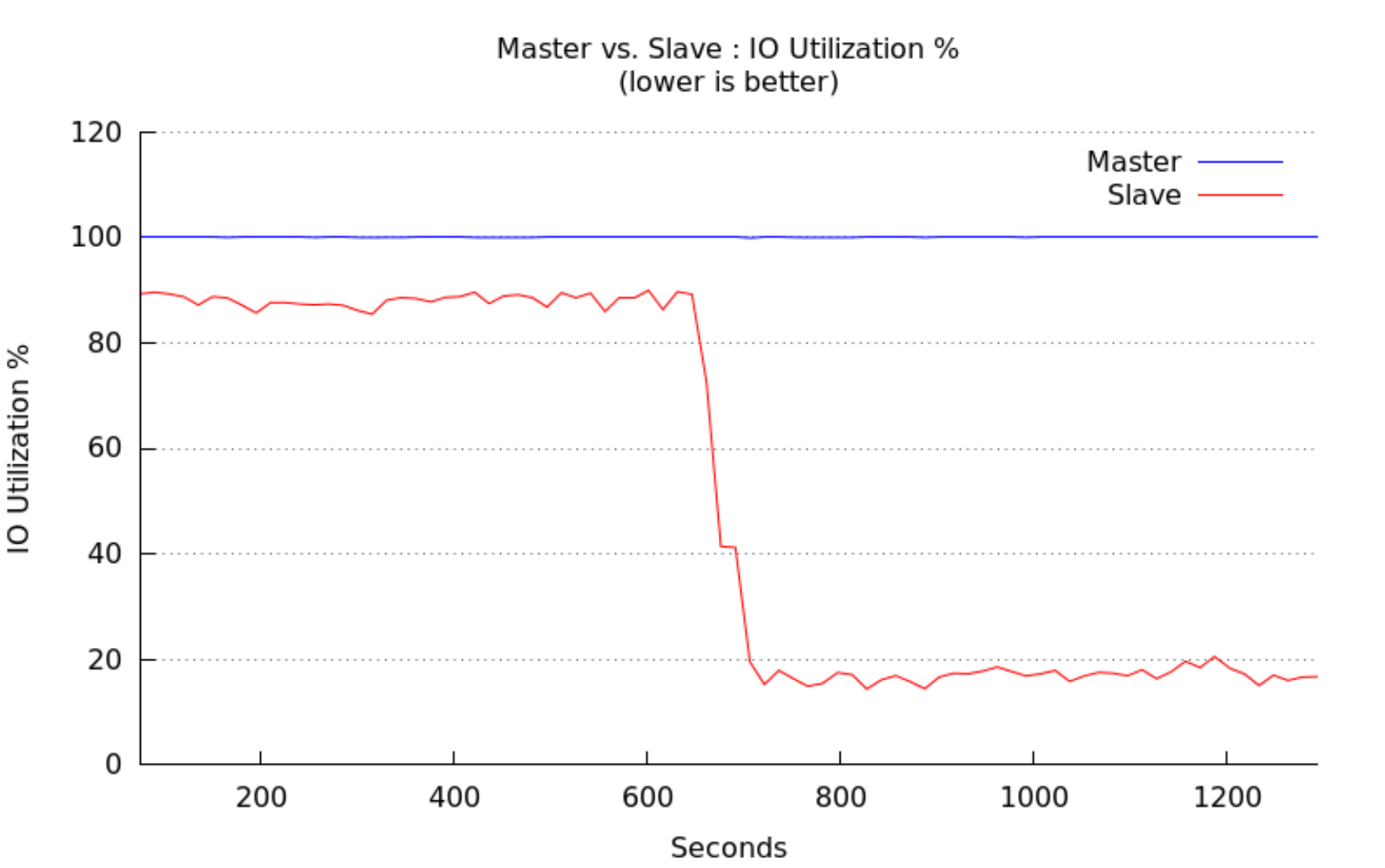
- Must be in “read only mode”
- Can disable unique checks (not necessary)
  - `tokudb_rpl_unique_checks=0`
- Can disable lookups
  - `tokudb_rpl_lookup_rows=0`

NOTE: mysql & maria 5.5 pk needs to be defined for the optimization to work

Does it work?


Master vs. Slave : Commits Per Second
(higher is better)
Commits Per Second
Seconds
Master
Slave
0
200
400
600
800
1000
1200
1400
1600
200
400
600
800
1000
1200

What about I/O?



Customer Testimonial

"Finally got to bounce the master server [...] and wow, talk about an improvement!... This release made a huge difference for our slave lag. Thank you! We went from 10-15 minute periods of slave lag with peaks about 15 minute of slave lag every hour to 5 minutes with under 1 minute of slave lag"

-Joe Piscatella, Limelight Networks

There's Still Work To do

- 5.5 still only has one thread
  - Percona Server 5.6 has RFR with multi threads!!!!
- Binlog still doesn't have great concurrency in 5.5
  - Bottlenecks fsyncing log (very expensive)
  - 5.6 has a much better "group commit algorithm"
    - TokuDB v7.5.4 for Percona takes advantage of this

InnoDB My.cnf Parameters

InnoDB

- **innodb_flush_method**
  - Default is fdatasync
  - How InnoDB performs IO
- **innodb_file_per_table=true**
  - Put each table in it's own file (instead of putting all tables into a single file), TokuDB always creates 1 file per index.
- **innodb_buffer_pool_size**
  - Default is 128M (Yikes!)
  - Amount of RAM to allocate for cache
- **innodb_flush_log_at_trx_commit**
  - Default is 1.
  - Controls what InnoDB does at transaction commit

TokuDB My.cnf Parameters

TokuDB

- **tokudb_commit_sync**
  - Default "on"
  - If on, log file is fsync()'d when transaction is committed.
- **tokudb_fsync_log_period**
  - Default is 0 (milliseconds)
  - Allows control of how frequently fsync() operations on the log occur, only valid if tokudb_commit_sync is on.
- **tokudb_read_block_size**
  - Default is 64K
  - Smallest unit of row data (think point queries).
- **tokudb_cache_size**
  - Default is 50% RAM
  - Amount of RAM to allocate for cache (i.e., 4G)
- **tokudb_row_format**
  - Default is tokudb_zlib
  - Valid values are tokudb_uncompressed, tokudb_quicklz, tokudb_zlib, tokudb_lzma
- **tokudb_directio**
  - Default is 0
  - Set to 1 to use directIO, not bufferedIO

TokuDB My.cnf Parameters

- **tokudb_prelock_empty**
  - default is on
  - set to off to disable bulk loading
  - the bulk loader is a great way to load large tables
    - only works if table is empty
    - only advantageous if you are loading a good deal of data (>500,000 rows).
    - overhead isn't worth it for small loads
  - be careful, bulk loading is triggered by the first insert into an empty table if this is set to on
    - LOAD DATA [INFILE] ...
    - INSERT INTO foo SELECT * FROM bar
    - CREATE TABLE foo SELECT * FROM bar
    - INSERT INTO foo VALUES (1,1,1);

Tips for a Successful Eval

1. Define your success criteria before eval
2. Use data representative of your expected workload
3. Use like for like (hardware & parameters)
4. Bottlenecks may be different
   - Inno = Disk bound
   - Toku = CPU bound
5. Try “slaving” in a Toku machine to see relative difference
6. Evaluate your ‘unique’ indexes

Links

- iiBench
  - https://code.launchpad.net/~mdcallag/mysql-patch/mytools
- Andy Pavlo's Datasets
  - http://www.cs.cmu.edu/~./pavlo/datasets/index.html
- TokuDB 7.5.5 Download
  - http://www.tokutek.com/products/downloads/

Any Questions?

THANK YOU!