Fractal Tree® Indexes

Theory and Practice

Tim Callaghan
VP/Engineering, Tokutek
tim@tokutek.com
@tmcallaghan

Tokutek®

Who am I?

"Mark Callaghan's lesser-known but nonetheless smart brother."

[C. Monash, May 2010]

www.dbms2.com/2010/05/25/voltdb-finally-launches

Tokutek®

Please
Ask
Questions.

Tokutek®

Fractal Tree Indexes

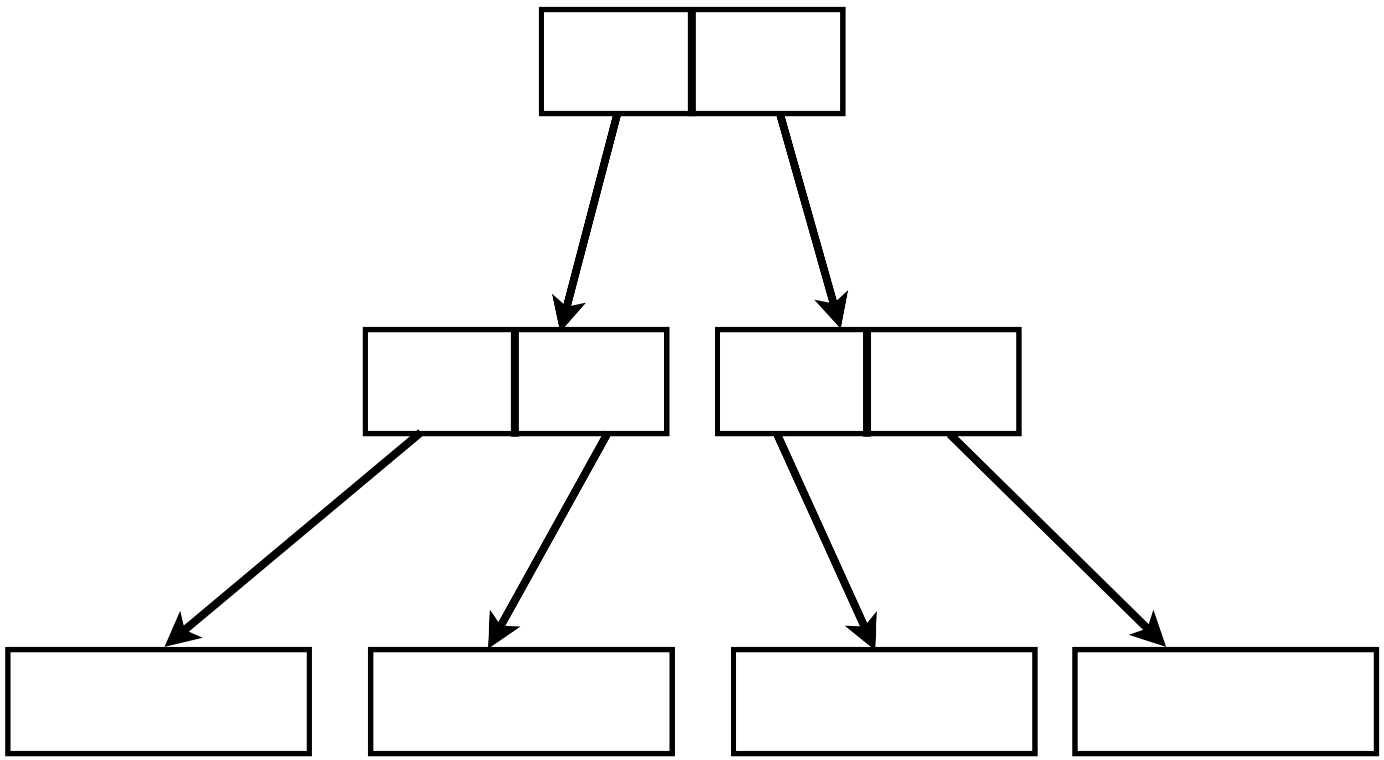
Theory

- At a high level
  - B-trees
  - InnoDB storage
  - TokuDB storage
  - MongoDB storage

I’m going to cover very simple scenarios (no splitting or merging)

B-trees

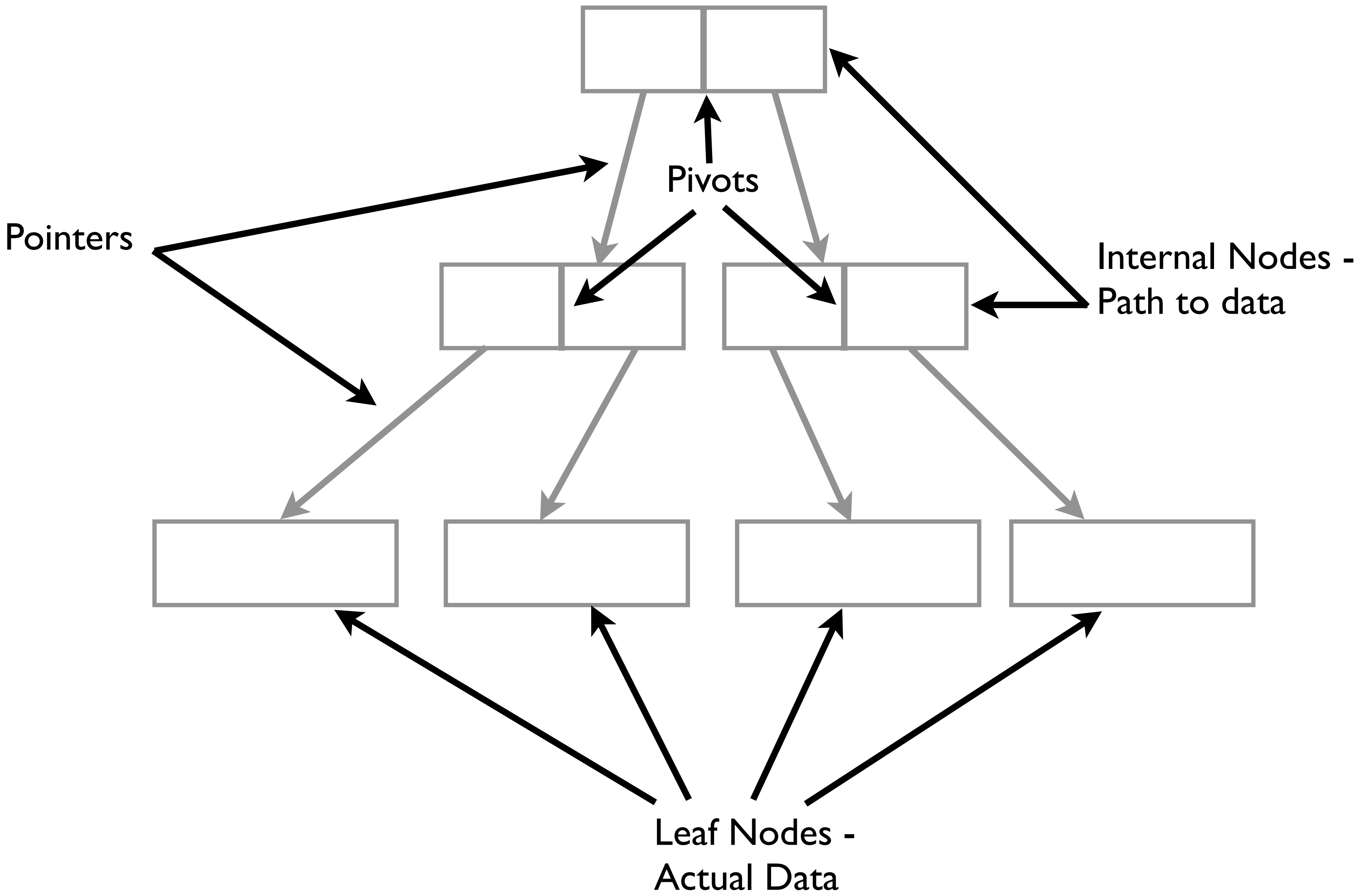
B-tree Overview



I will use a simple single-pivot example throughout this presentation

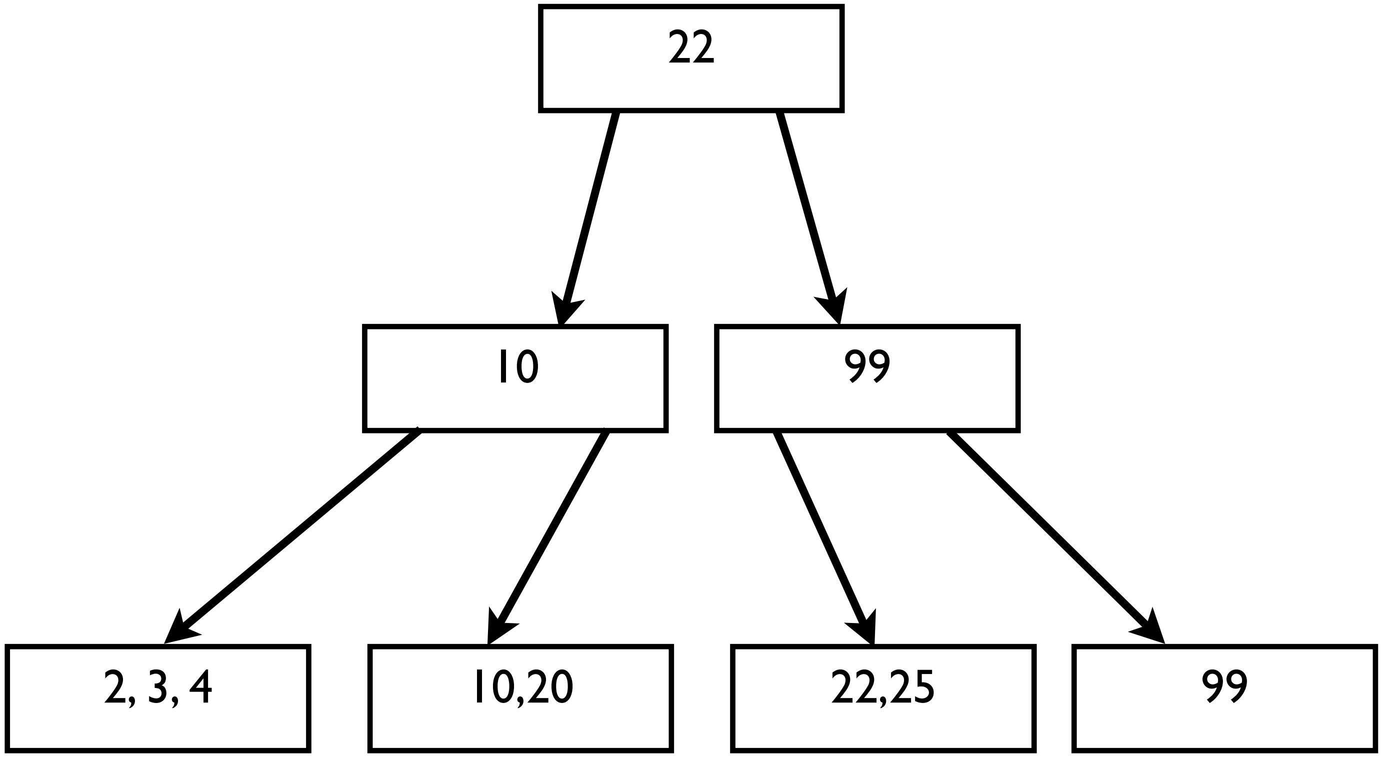
Tokutek

B-tree Overview - vocabulary



Tokutek

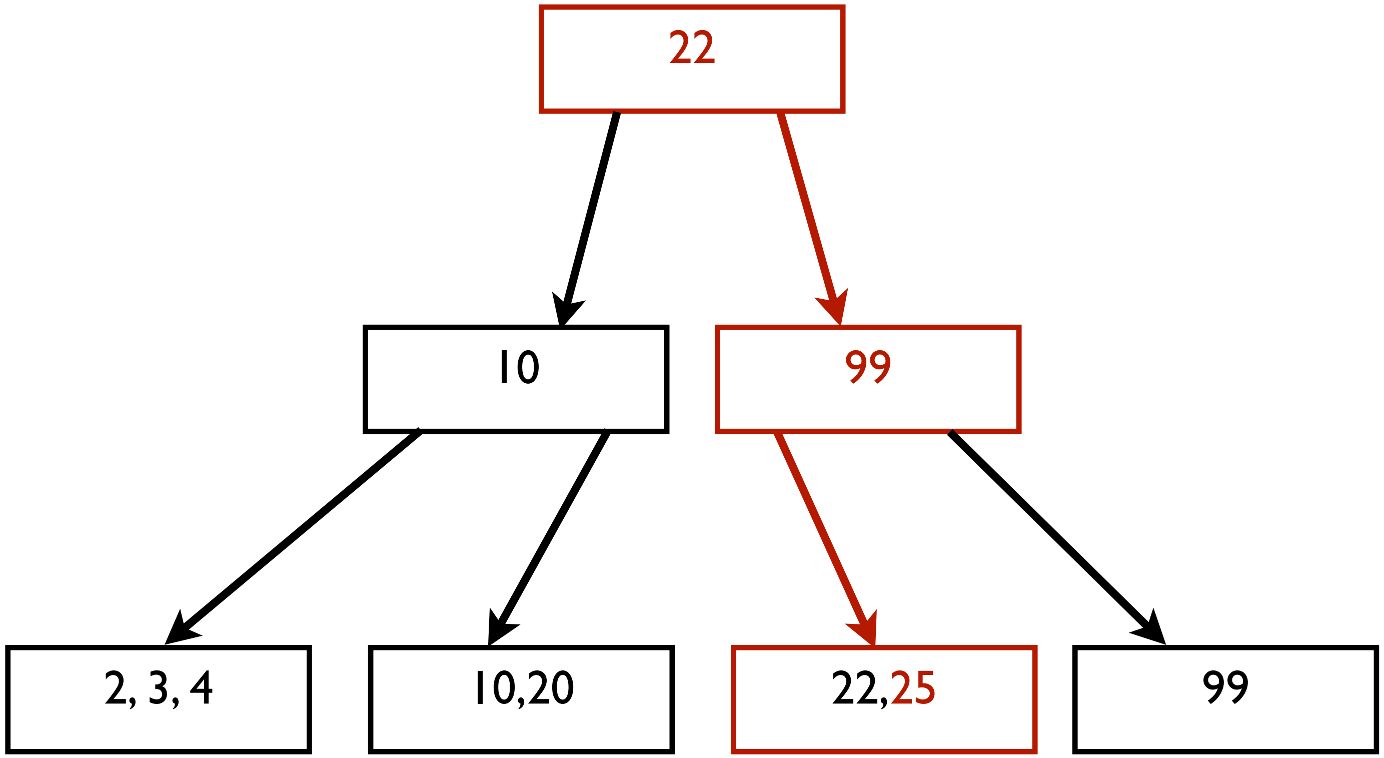
B-tree Overview - example



* Pivot Rule is >=

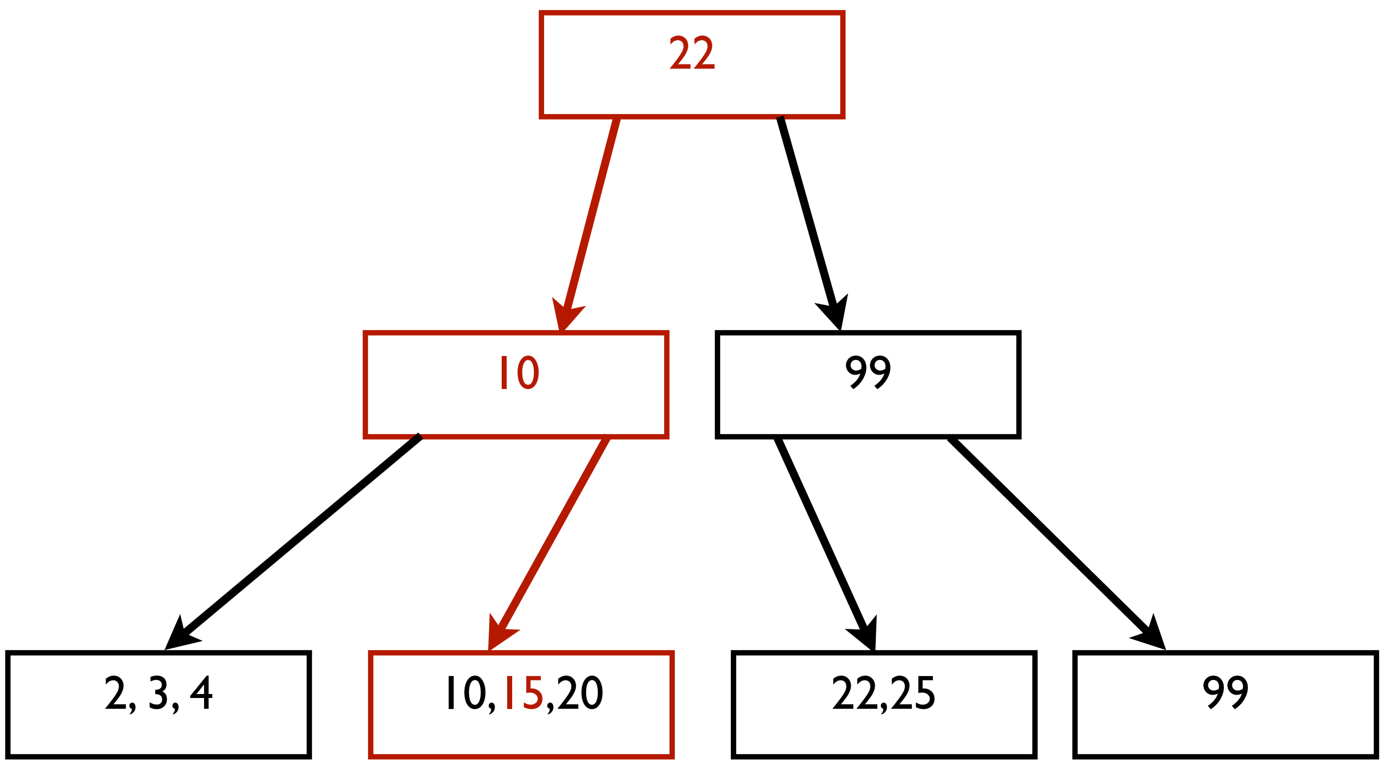
B-tree Overview - search

"Find 25"



B-tree Overview - insert

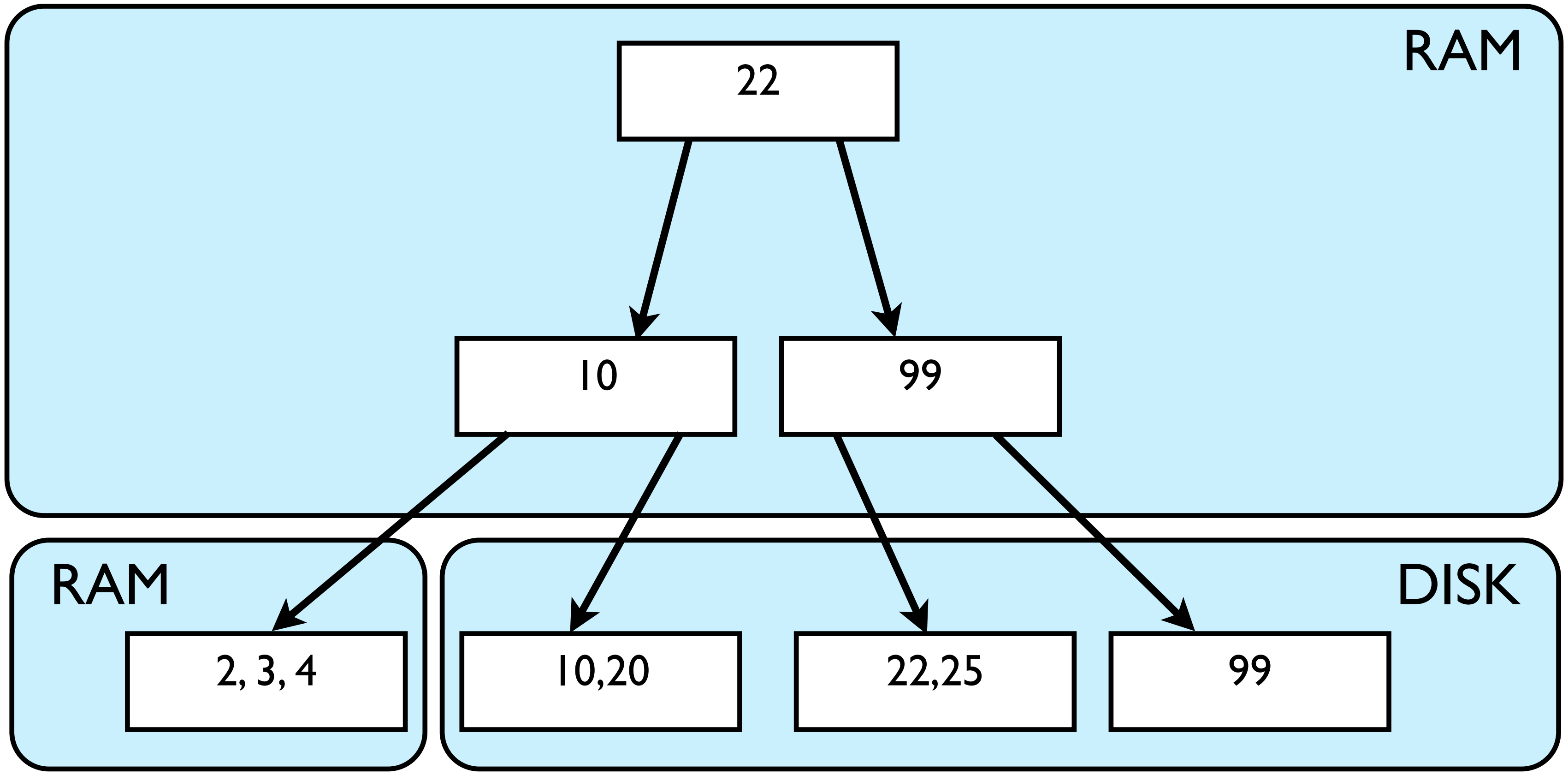
"Insert 15"



Tokutek

B-tree Overview - storage

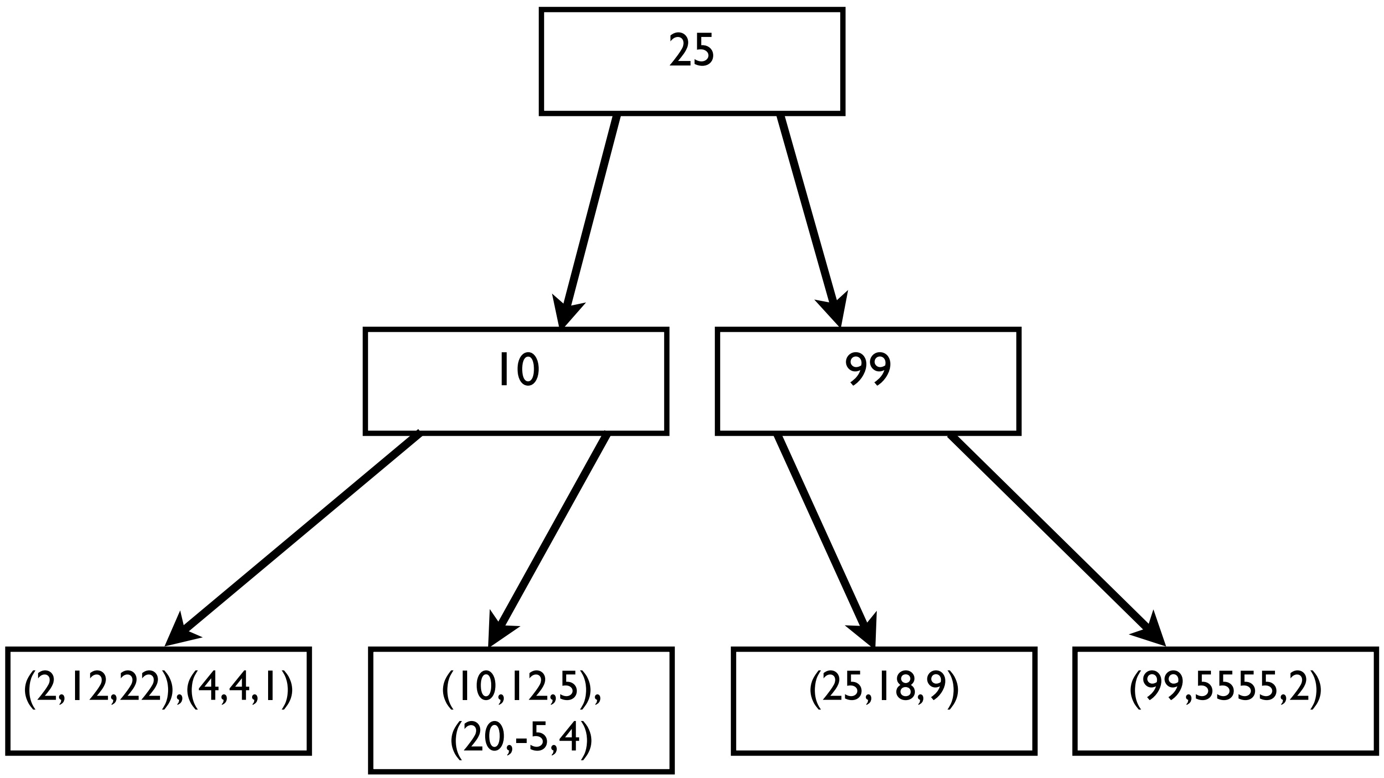
Performance is IO limited when bigger than RAM: try to fit all internal nodes and some leaf nodes



InnoDB Storage

Tokutek®

InnoDB B-trees - primary key



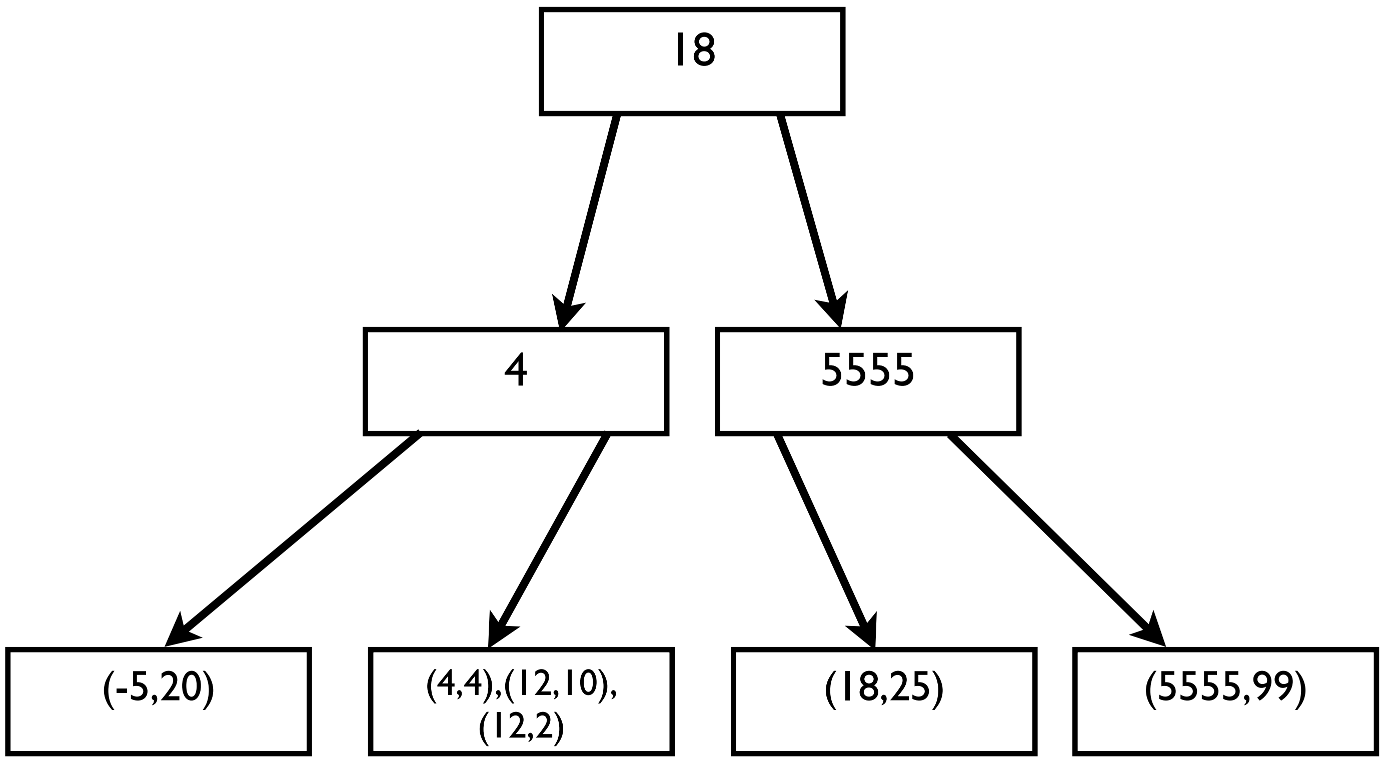
store full rows in leaf nodes

use primary key for ordering

example: table t (a int primary key, b int, c int)

Tokutek®

InnoDB B-trees - secondary keys



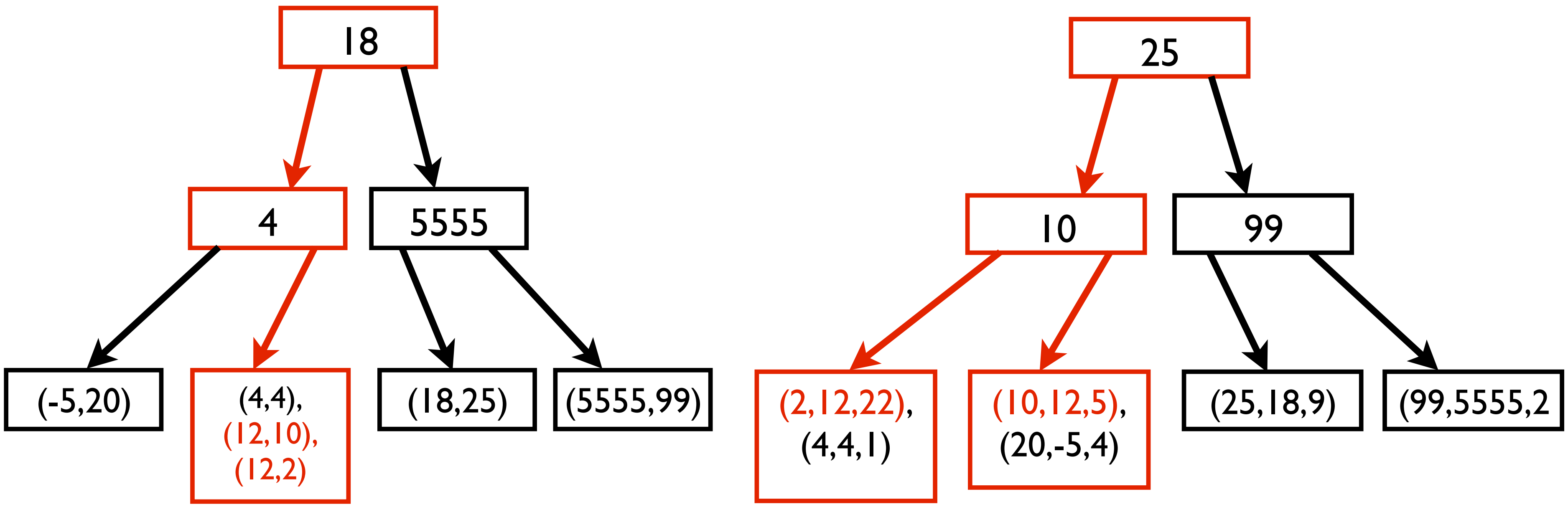
each secondary key creates another B-tree
use secondary key for ordering
PK is stored as well

Tokutek®

InnoDB B-trees - search

select * from t where b=12;

(a) secondary index - find rows for b=12 (get PKs) (b) PK index - get * for PK=10 and 2



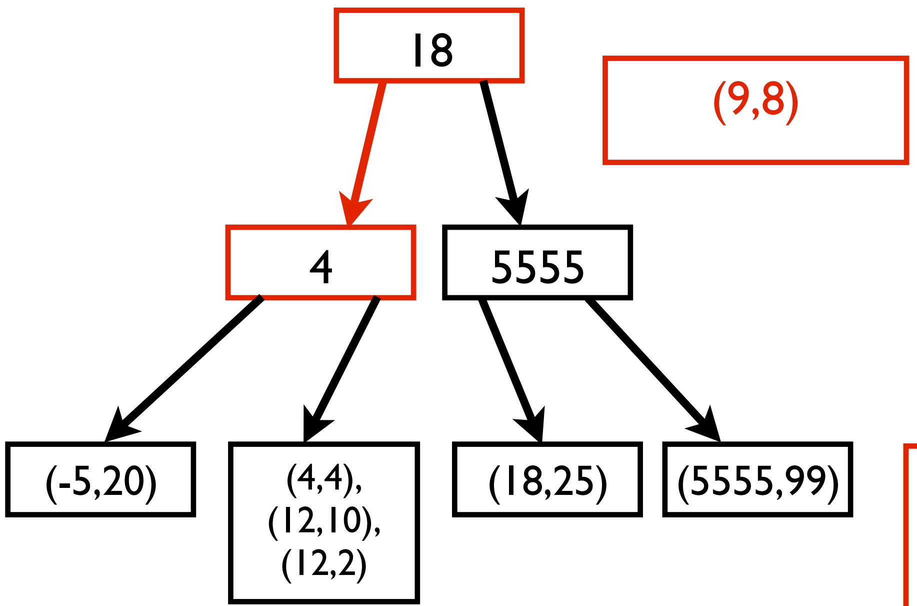
Tokutek

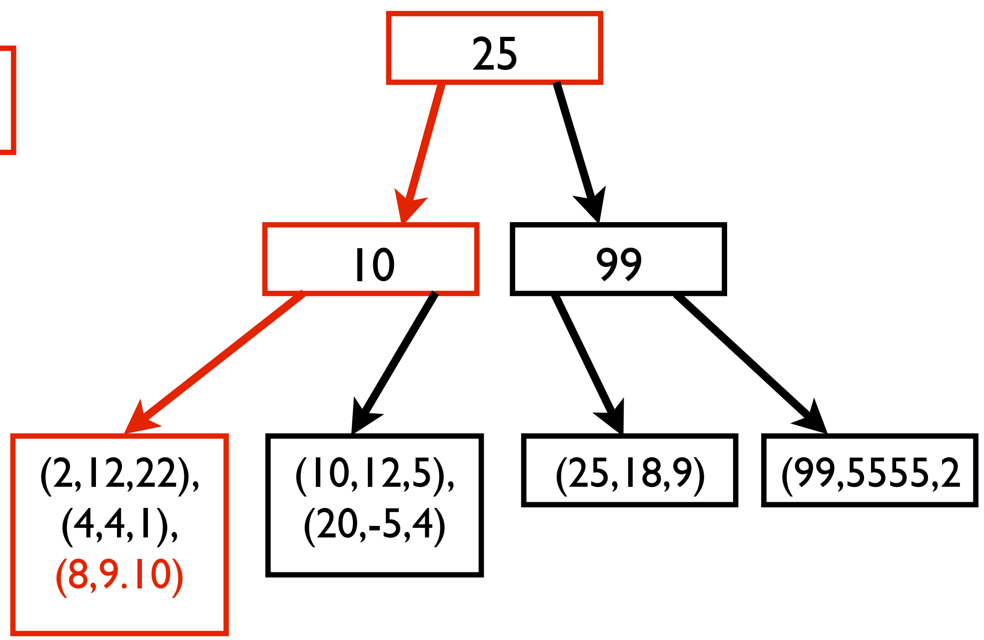
InnoDB B-trees - insert

insert into t values (8,9,10);

(a) insert into leaf if in memory, buffer if not

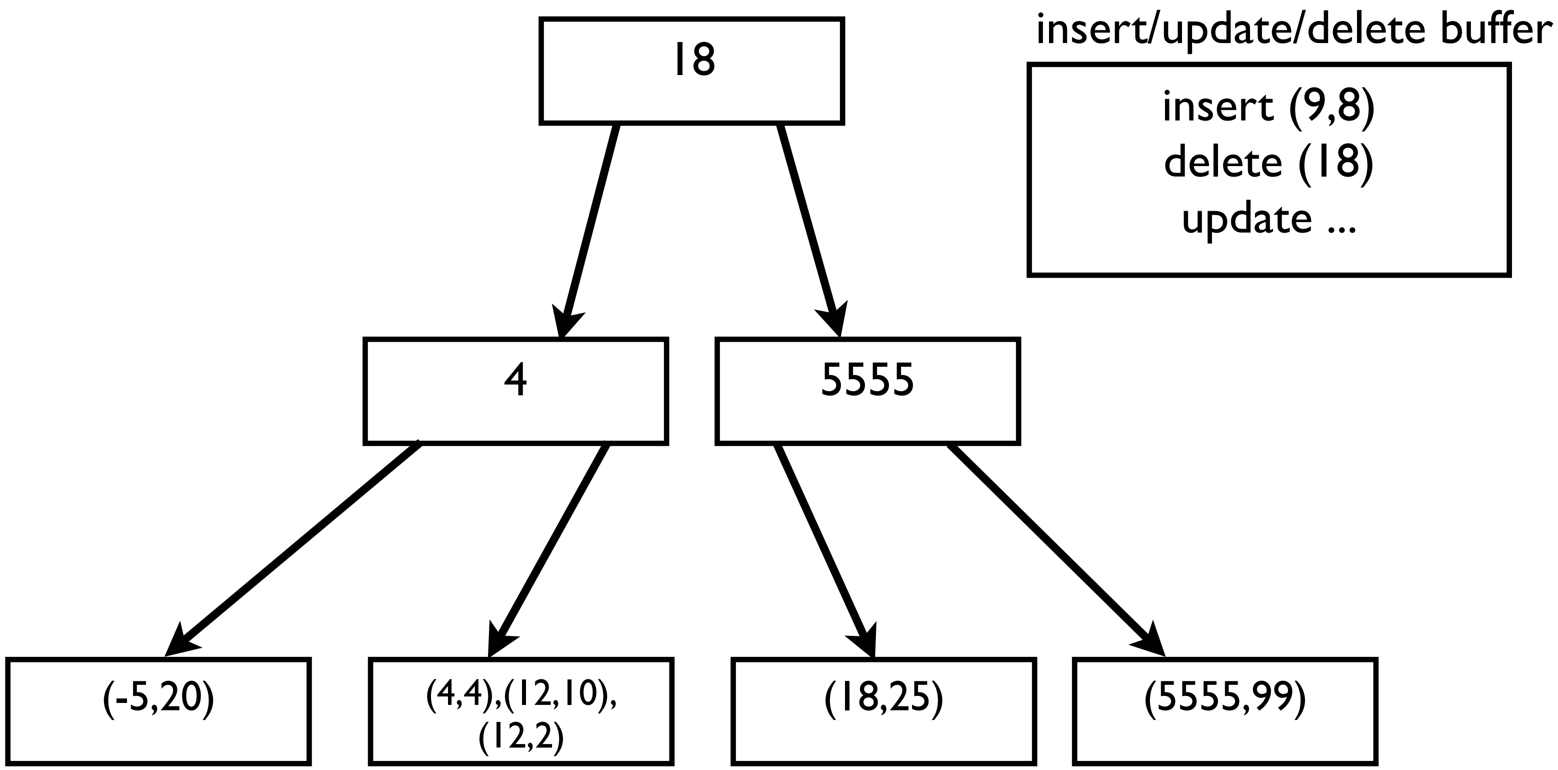
(b) insert into leaf node in PK index





Tokutek

InnoDB B-trees - secondary key buffer



- Buffer inserts, deletes, and updates if the needed leaf node is not in memory
- Flushing occurs when the buffer is full or the leaf node comes into memory

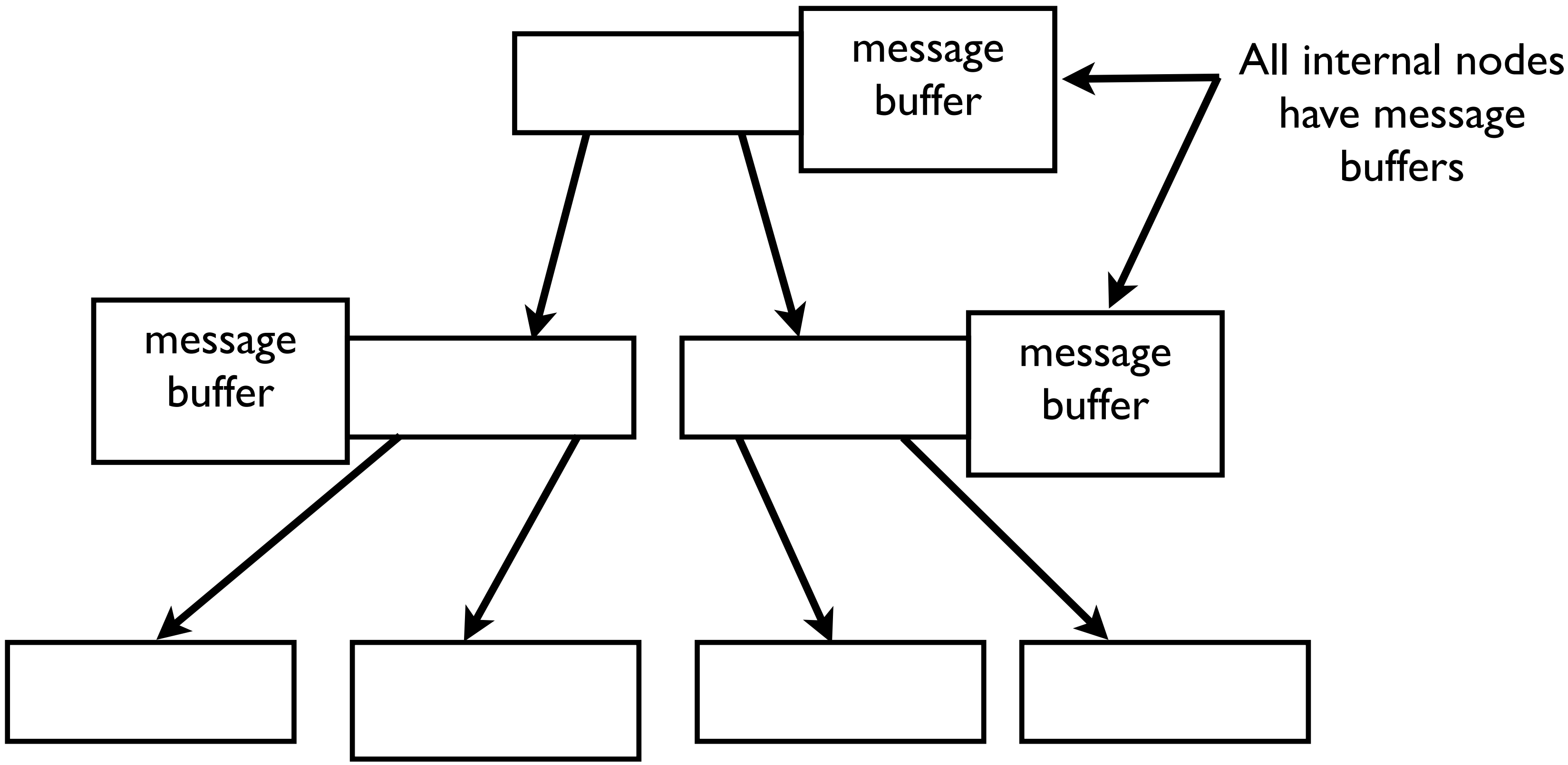
Tokutek

Fractal Tree® Indexes

TokuDB Storage

Tokutek®

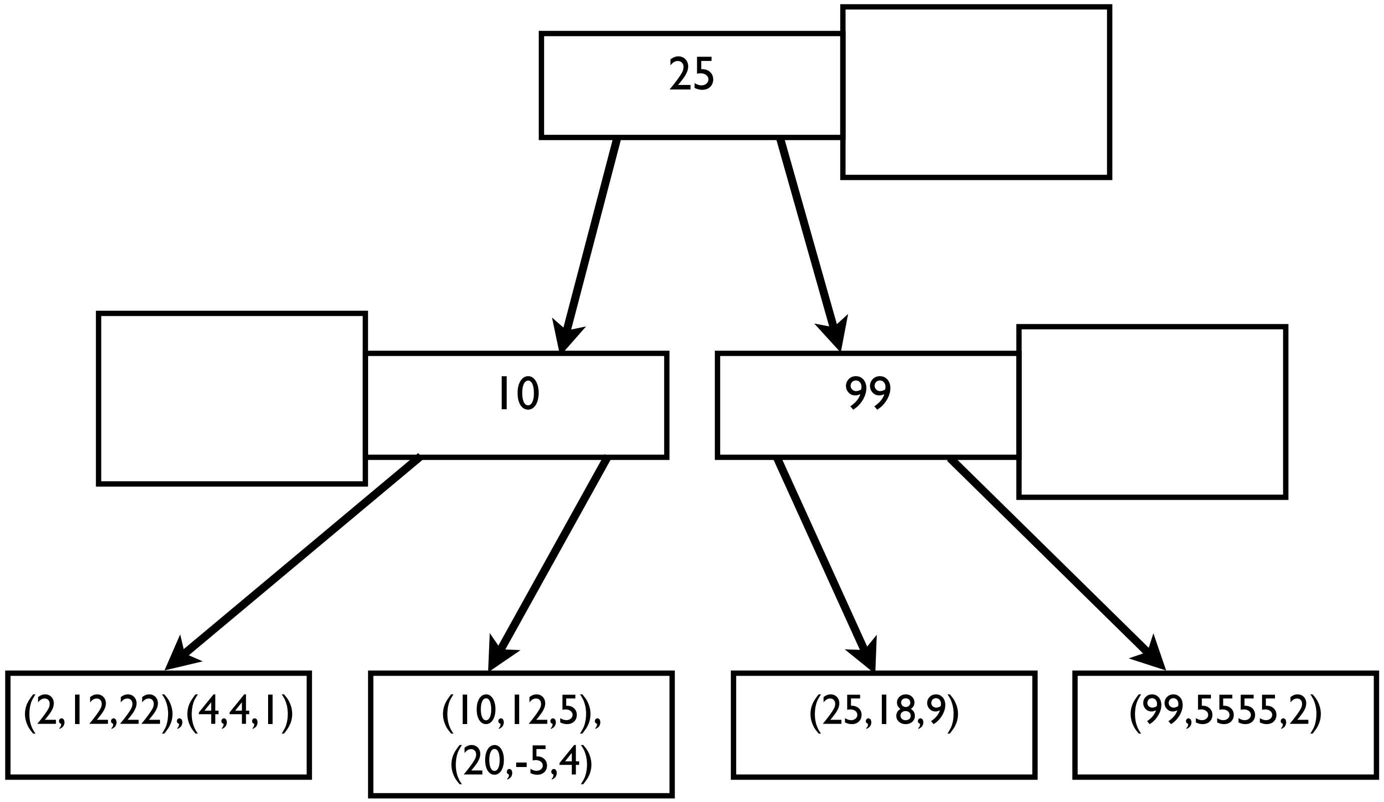
Fractal Tree® Indexes



similar: store data in leaf nodes
similar: use PK for ordering
different: message buffer within all internal nodes
different: much larger nodes (4MB vs. 16KB)

Tokutek®

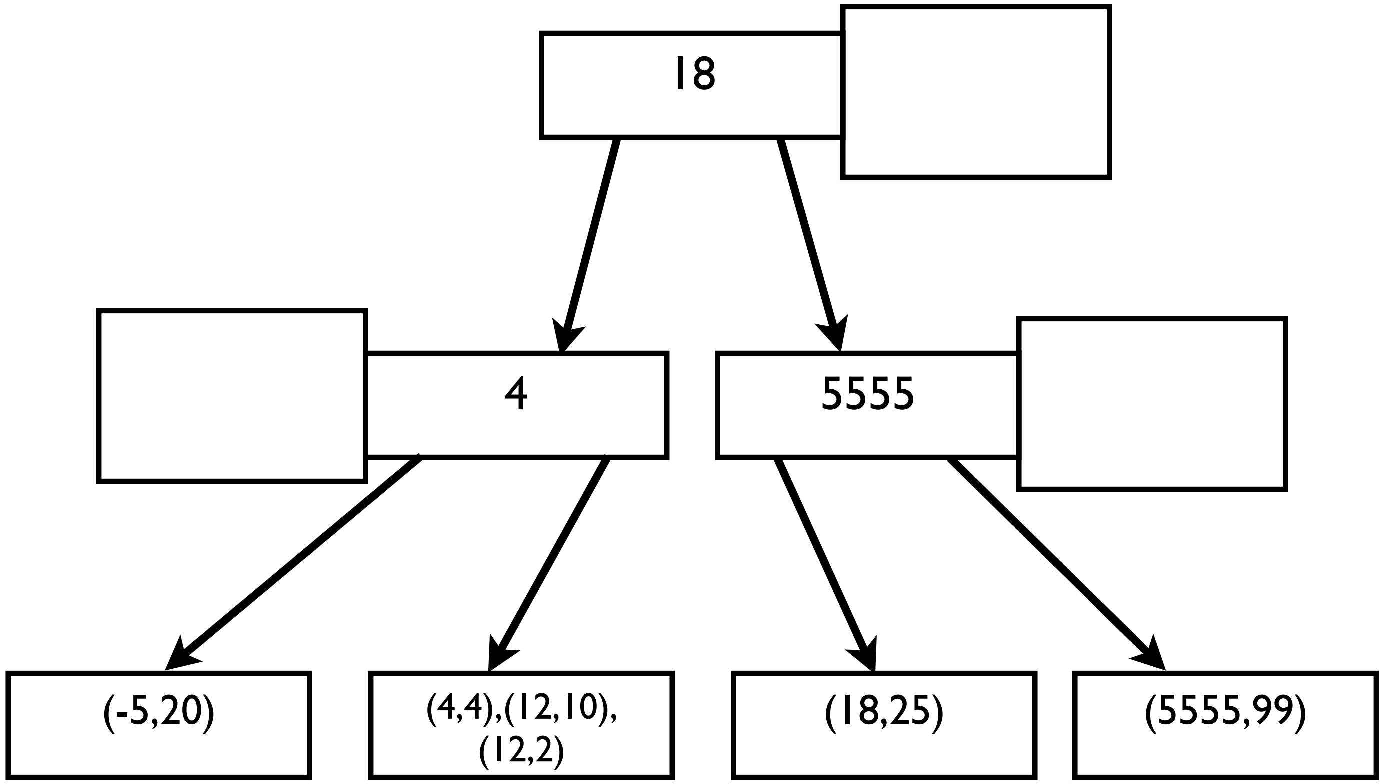
Fractal Tree® Indexes - primary key



example: table t (a int primary key, b int, c int)

Tokutek®

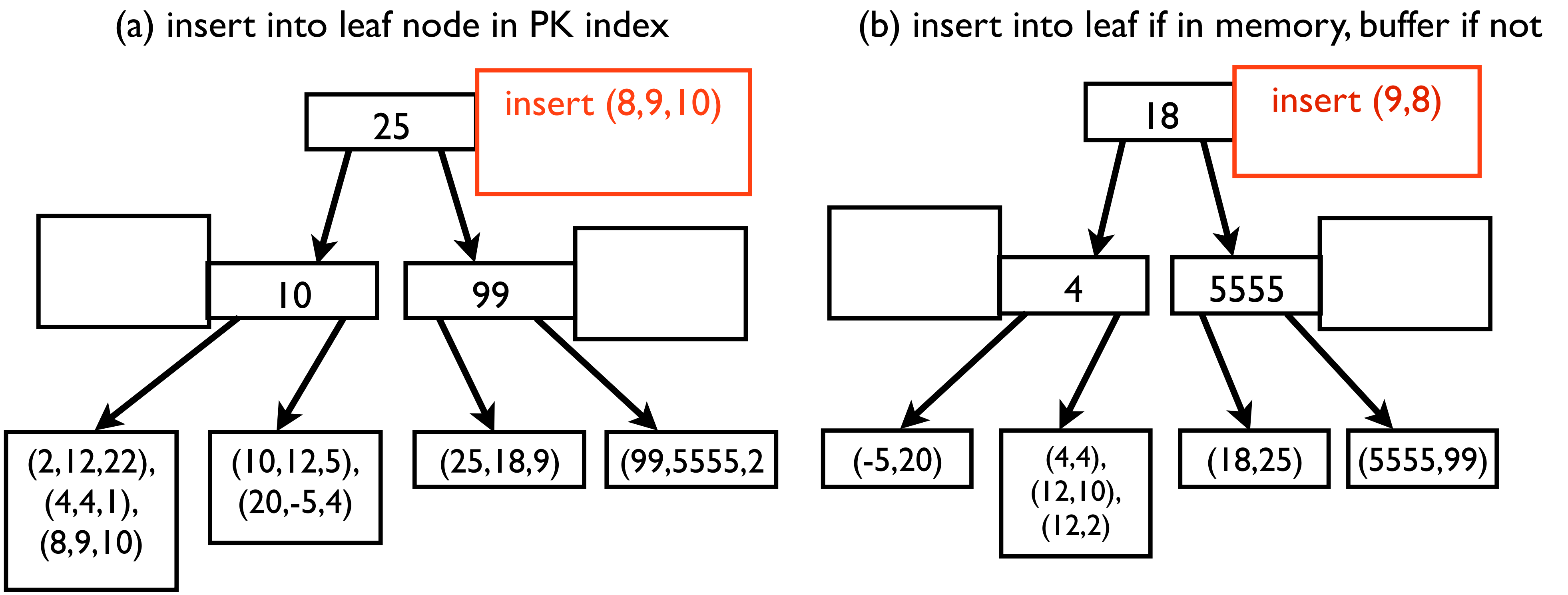
Fractal Tree® Indexes - secondary keys



Tokutek®

Fractal Tree® Indexes - insert

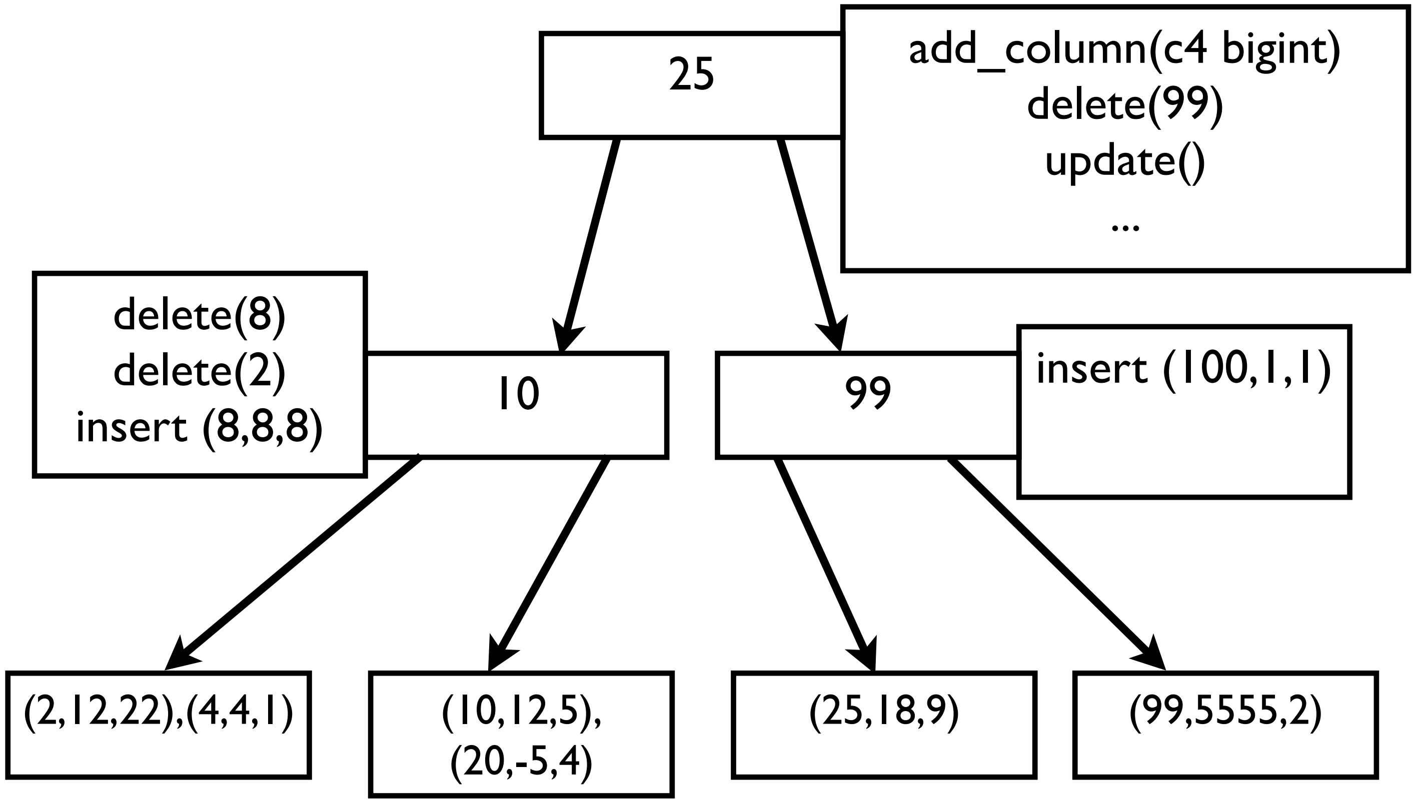
insert into t values (8,9,10);



- messages cascade down the tree as buffers fill up
- they are eventually applied to the leaf nodes

Tokutek

Fractal Tree® Indexes - other operations



Lots of operations can be messages!

Tokutek®

MongoDB Storage

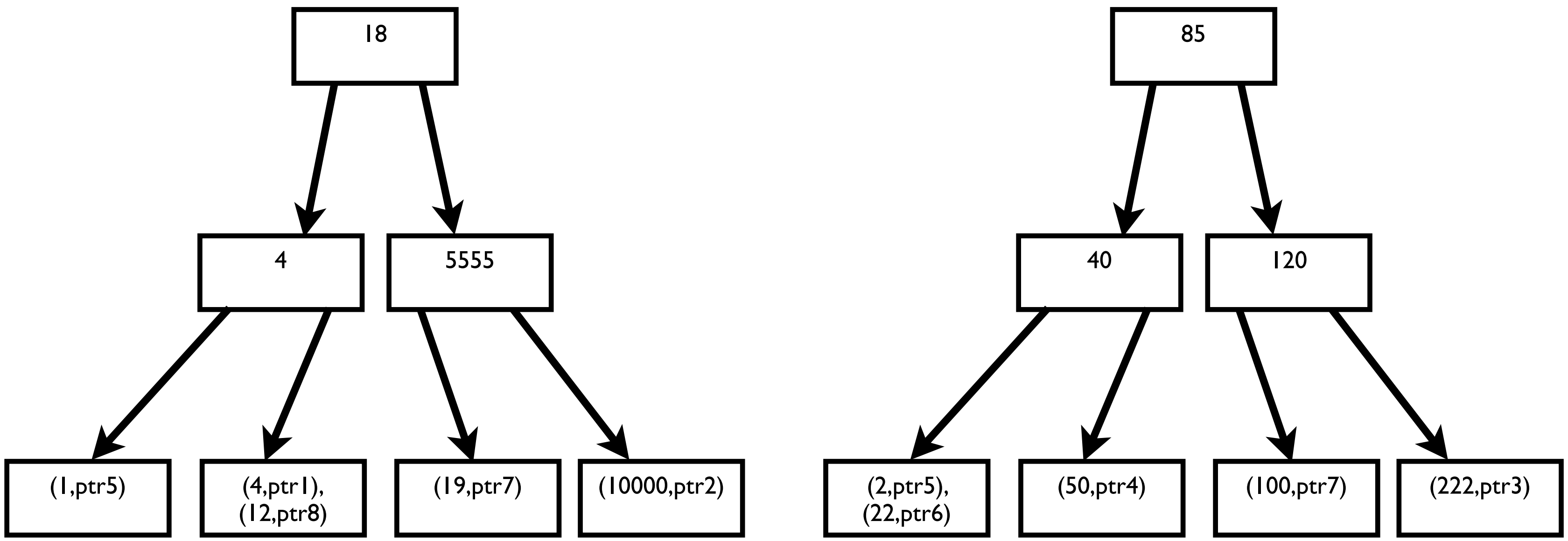
MongoDB Storage

db.test.insert({foo:55})
db.test.ensureIndex({foo:1})

memory mapped heap

PK index (_id + pointer)

Secondary index (foo + pointer)



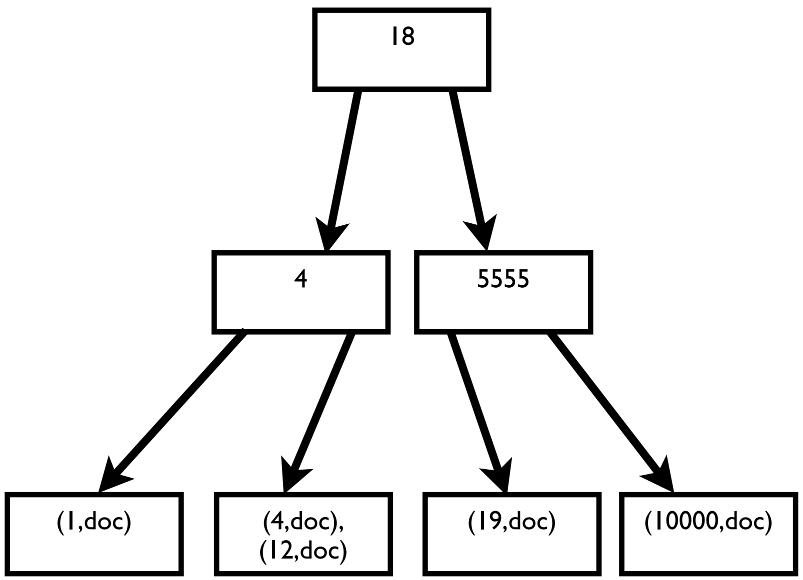
The "pointer" tells MongoDB where to look in the heap for the document.

MongoDB Storage with Fractal Tree Indexes

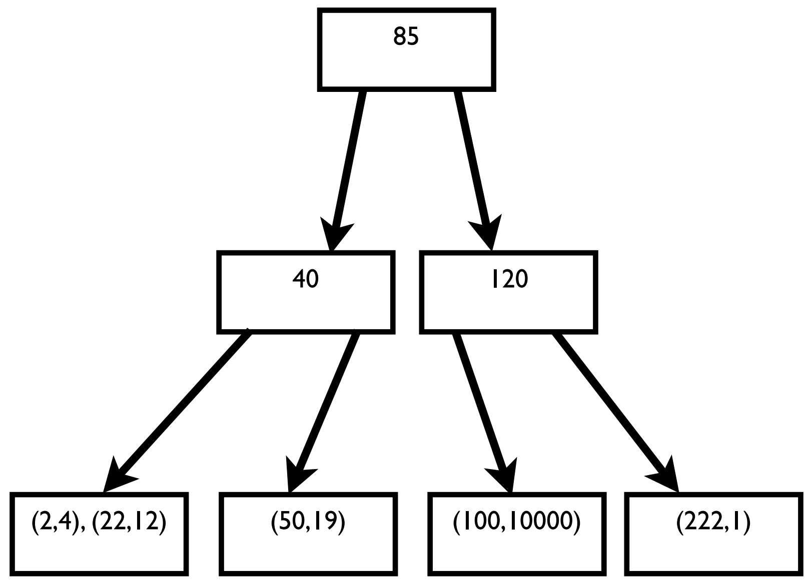
db.test.insert({foo:55})
db.test.ensureIndex({foo:1})

memory mapped heap

PK index (_id + document)



Secondary index (foo + _id) - can be clustered



Similar to TokuDB implementation

Tokutek

Fractal Tree Indexes

Practice

TokuDB

Fractal Tree Indexes
for MySQL and MariaDB

Tokutek®

What is TokuDB?

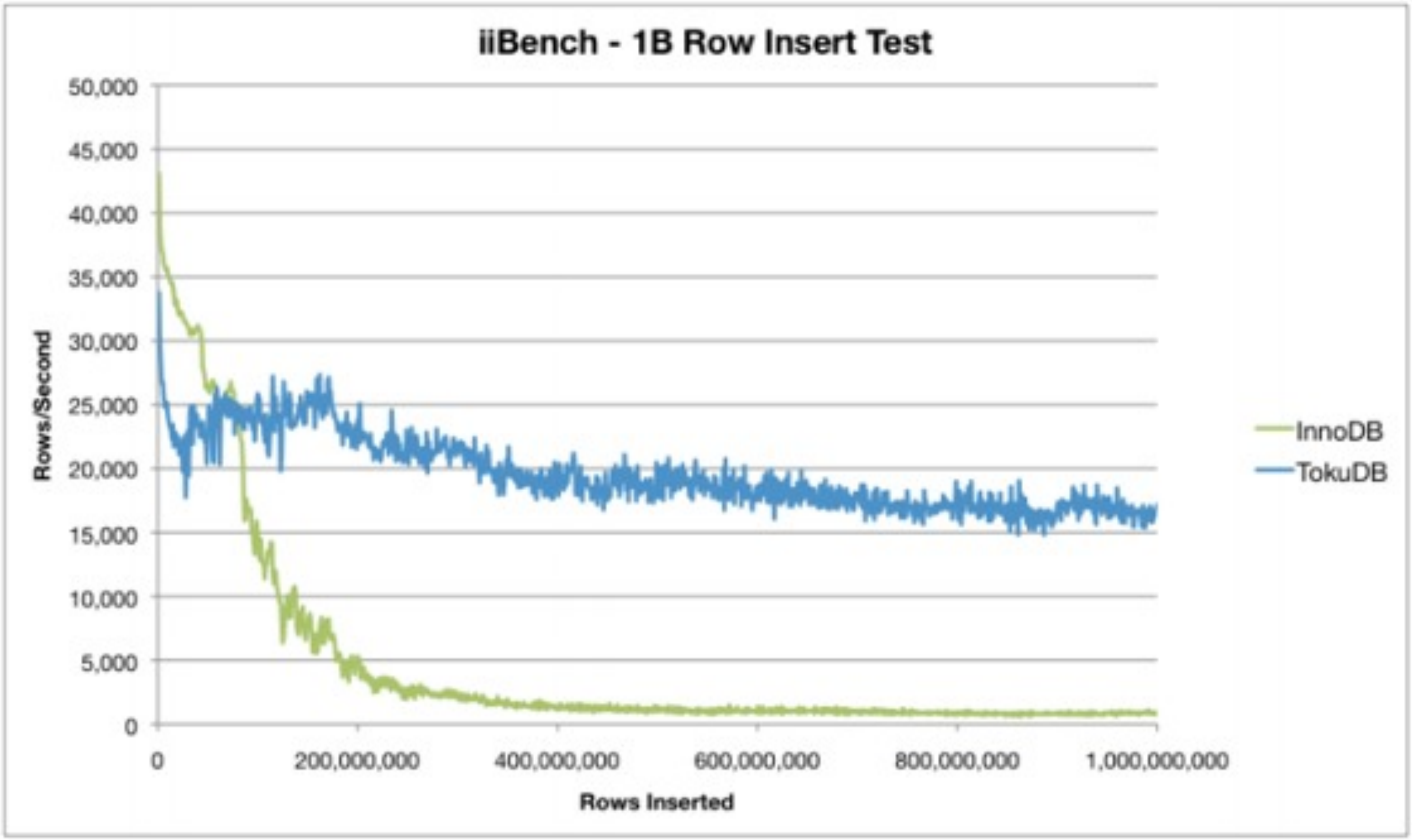
- Transactional MySQL Storage Engine - think InnoDB
- Available for MySQL 5.5 and MariaDB 5.5
- ACID and MVCC
- 64-bit Linux
- Advantages: Performance, Compression, Agility
- Free/OSS Community Edition
- Enterprise Edition = commercial support + hot backup

Tokutek®

Warning - Benchmarks Ahead!

Indexed Insertion Performance

- High-performance insert/update/delete for large databases (> RAM) while maintaining indexes



** old numbers, now > 25K/sec*

Tokutek

Managing Indexes in the Real World

- Evaluation : huge table (> 25 billion rows)
- Schema = int a, int b, varchar c
- Unacceptable insertion performance if 1 secondary index
- Some queries require column a, others require column b
- InnoDB solution = 2 masters (each with 2 slaves)
  - Master 1, pk = (a,b)
  - Master 2, pk = (b,a)
  - All inserts to both, queries directed by index need
- TokuDB solution = 1 master (2 slaves)
  - pk = (a,b), secondary clustered index (b,a)
  - 1/2 the servers, 1/5 the storage

Tokutek®

Compression: TokuDB vs. InnoDb

- InnoDB starts with smaller block size, 16KB vs. 64KB (or more)
- InnoDB compression requires fixed "on-disk" size
  - 1KB, 2KB, 4KB, 8KB
- Limits compression achieved
  - 16KB stored as 4KB = 4x compression
- Compression misses force node splits, which greatly reduces performance
- TokuDB on-disk size is variable
  - Data is compressed and stored
- Multiple compression algorithms supported
  - lzma, quicklz, zlib
  - easy to add more (lz4, lz4hc)

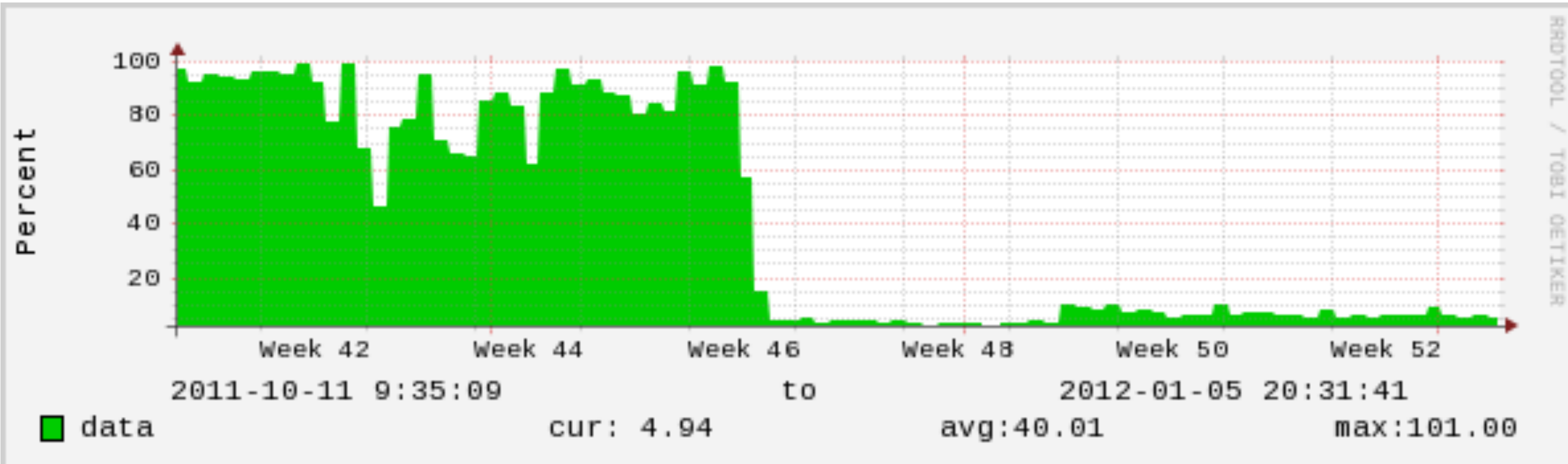
Tokutek®

Compression: Disk vs. Flash

- Both
  - Bigger, less frequent writes
    - Have measured 200x less IO on benchmarks
- Disk
  - Compressed reads and writes overcome IO limitations
- Flash
  - Buy less
    - 250G device w/ 5x compression behaves as 1.2TB
  - Large/less frequent writes are flash friendly

Tokutek®

Compression + IO Reduction

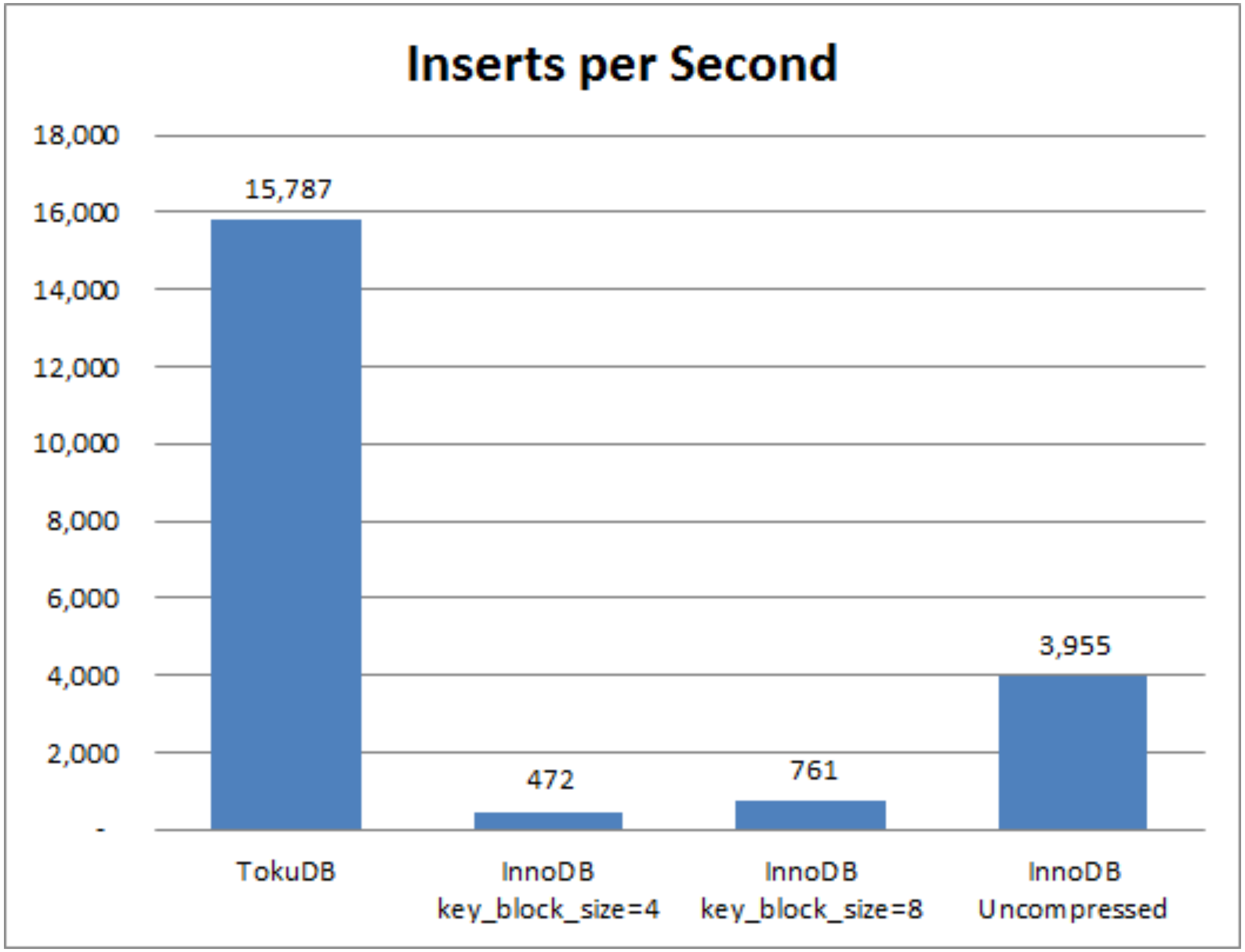


- Server was at 90% IO utilization with InnoDB, 10% IO utilization with TokuDB

Tokutek®

Compression Performance

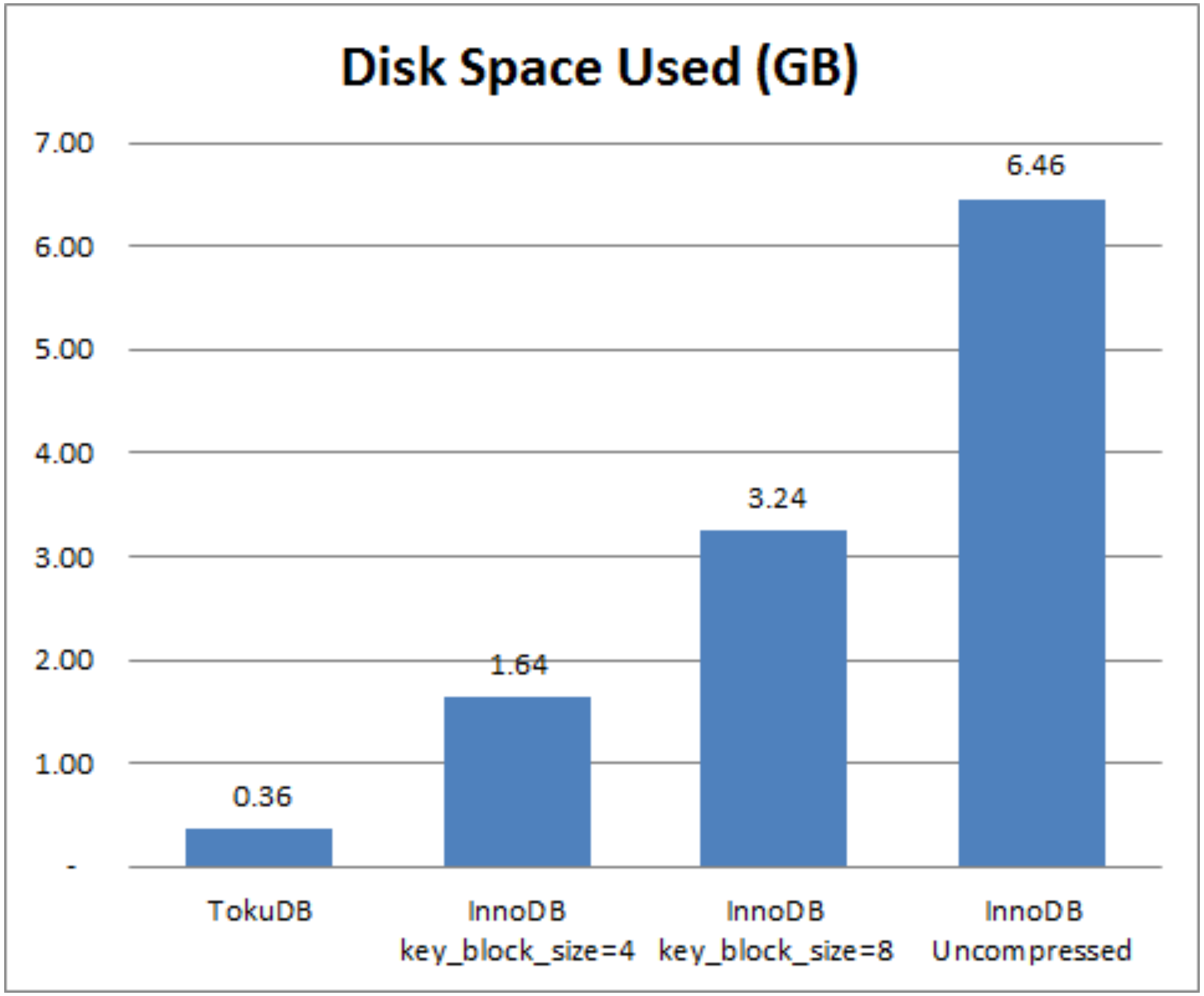
- iiBench benchmark



Tokutek

Compression Achieved

- log data load



Tokutek

TokuDB: Performance Advantages

- Efficient secondary index maintenance
  - Much better than InnoDB’s secondary index buffer
- Clustered secondary indexes
  - No additional IO to get row data from primary key index
  - Think better covering index (all non-indexed columns)
  - Compression eliminates size concerns

Tokutek

TokuDB: Performance Advantages

- Big blocks = sequential IO for range scans
  - Basement nodes are always co-located
- Compressed reads
  - With 10x compression a 16K read is actually 160K
- High-performance bulk loader
  - Creates Fractal Tree Indexes directly

Tokutek®

TokuDB: SQL Optimizations

- Message based architecture allows for certain optimizations
- SQL statements that do not require read-before-write
  - replace into
  - insert ignore
  - update hit_counter set hits=hits+1 where site=‘www.tokutek.com’;
  - insert ... on duplicate key update;
- Message is inserted
- Client does not get # of affected rows

Tokutek®

TokuDB: Online operations

- Common schema changes can take hours in MySQL
  - Adding, dropping, or expanding a column
  - Adding an index
- Also, the table is unavailable to write operations during the process
- As a workaround, people generally
  - Use a replication slave, then swap with master
  - Use helper tools: Percona OSC, MySQL 5.6
    - These have IO, CPU, RAM consequences
    - New column unavailable until the end
- Many are considering NoSQL (schema-less) technologies to overcome these limitations

Tokutek®

TokuDB: Hot column addition/deletion

- "alter table t1 add column c4 bigint;"
- InnoDB
  - Locks the table and performs a "select into ..." to the new table structure
  - Indexes get rebuilt as well
  - No access to the table allowed during the process
- TokuDB
  - Creates an addcolumn() message and returns
  - Over time, the column is physically added to the actual rows

Tokutek®

TokuDB: Hot indexing

- "create index c4_idx on t1(c4);"
- InnoDB
  - Locks the table and creates the index
  - Read only access to the table
  - Can take hours (or days) on large tables
- TokuDB
  - Begins creating the index in the background
  - Index is available to MySQL when finished
  - Accurate progress via "show processlist;"

Tokutek®

Management/other

- Frequent checkpointing, fast recovery
  - Defaults to 60 seconds, 1 knob
- Online backup
  - Coming soon, enterprise feature
- Simple server configuration, sensible defaults
  - Few knobs, I generally set 3 for benchmarking
- Production is 64-bit Linux only
  - Mac binary for dev/test, Windows use VM

Fractal Tree Indexes for MongoDB

Tokutek®

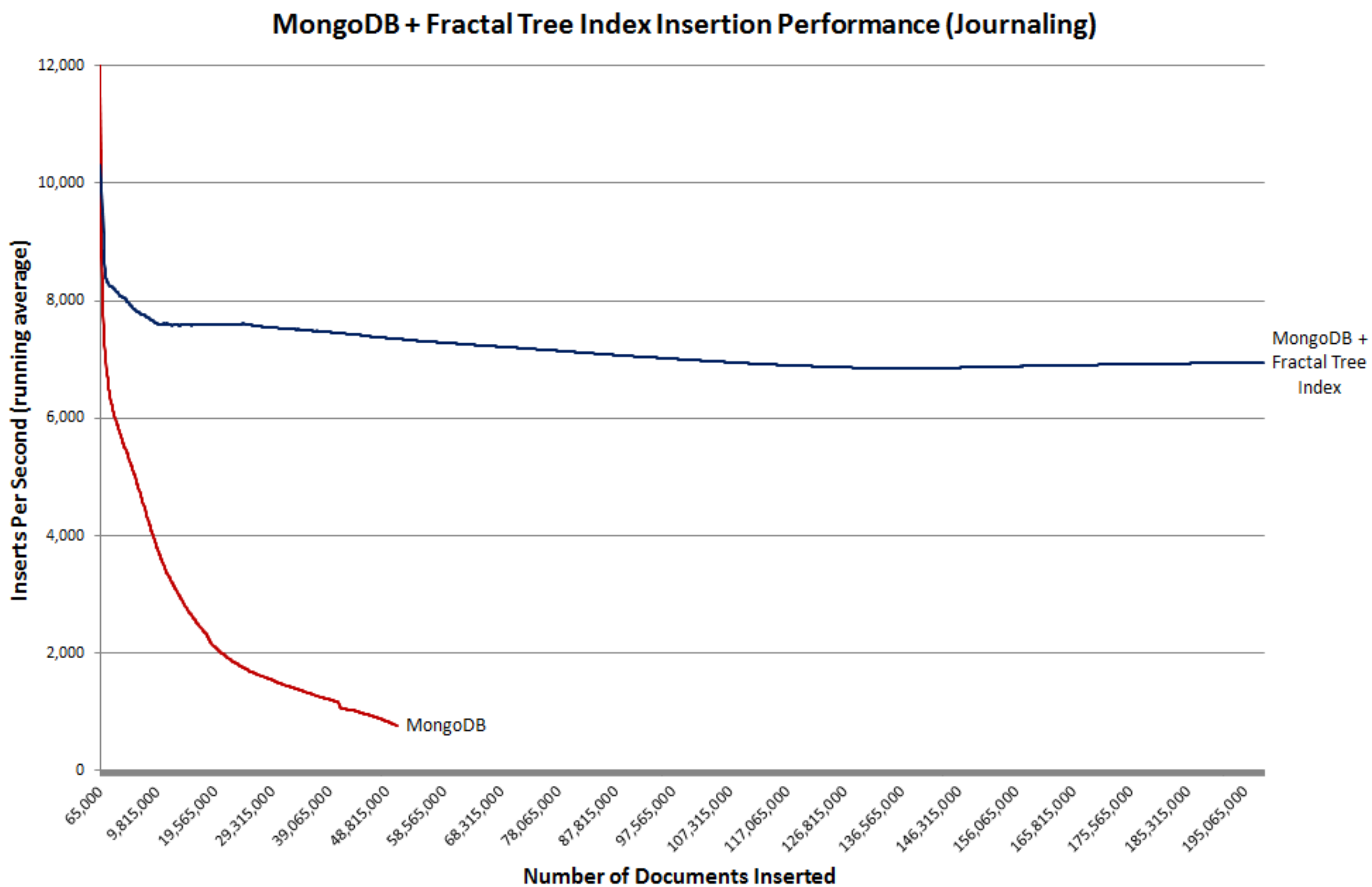
What is it?

- Same Fractal Tree index technology that supports TokuDB for MySQL
  - compression
  - checkpointing
  - memory management
    - control your MongoDB server's memory usage
  - ACID + MVCC
  - multi-statement transactions
  - document level locking
    - MongoDB offers database level locking

Tokutek®

MongoDB : Indexed Insertion

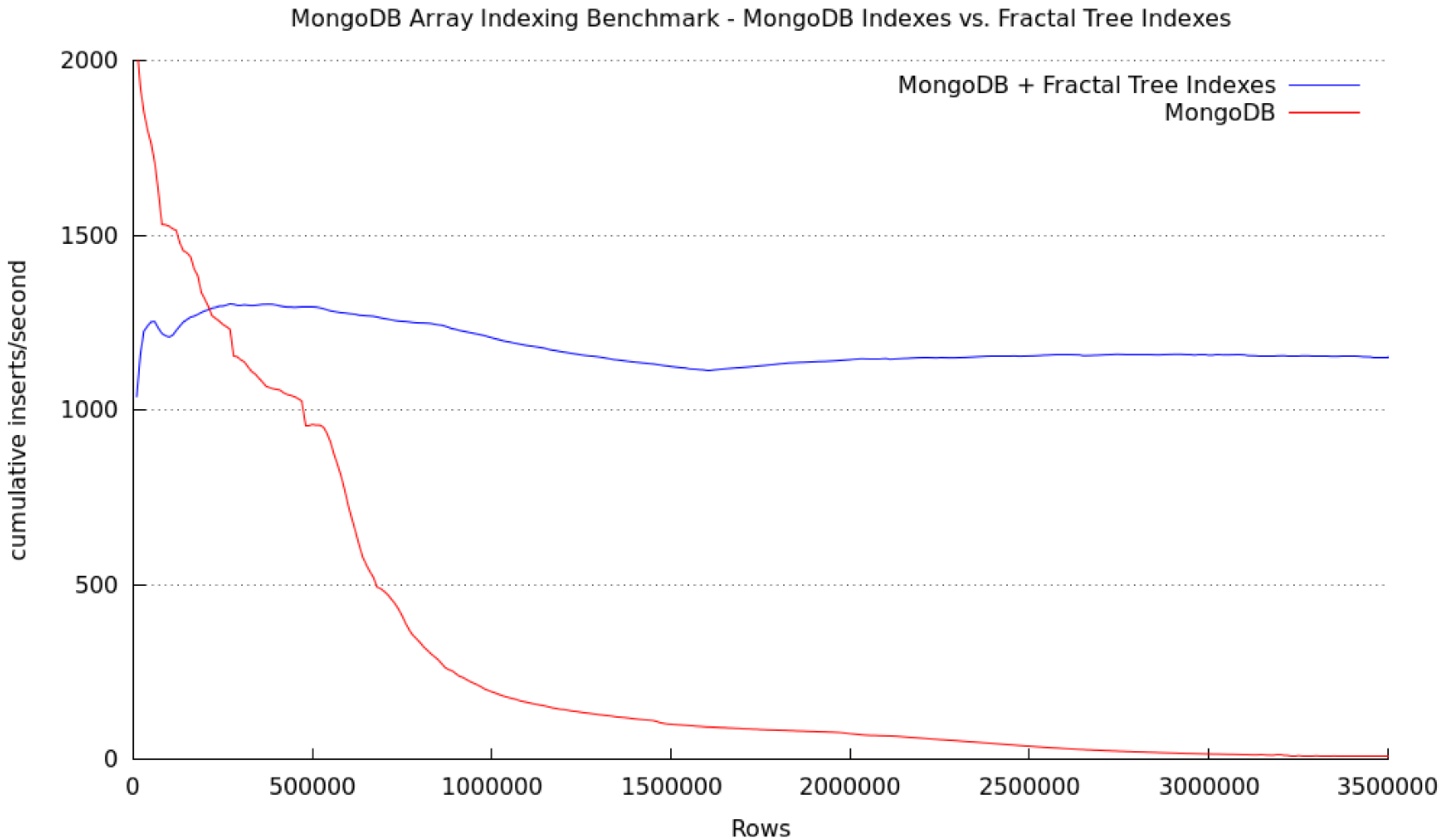
- 3 secondary indexes, similar to TokuDB vs. InnoDB



Tokutek®

MongoDB : Indexed Insertion - Arrays

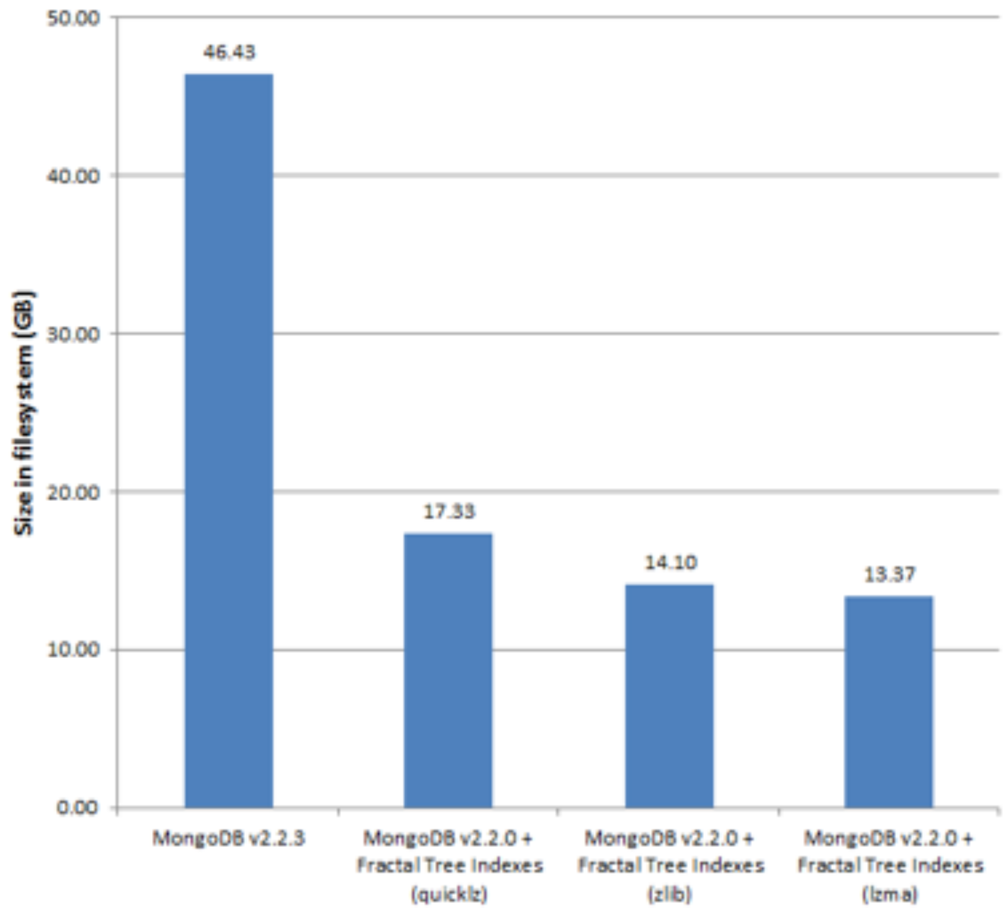
- Indexed Insertion : Multikey (100 inserts per doc)



Tokutek

MongoDB : Compression

- MongoDB has no compression option
- We offer the same 3 compression algorithms as TokuDB
- Chart shows space used for 51 million semi-compressible documents



Tokutek®

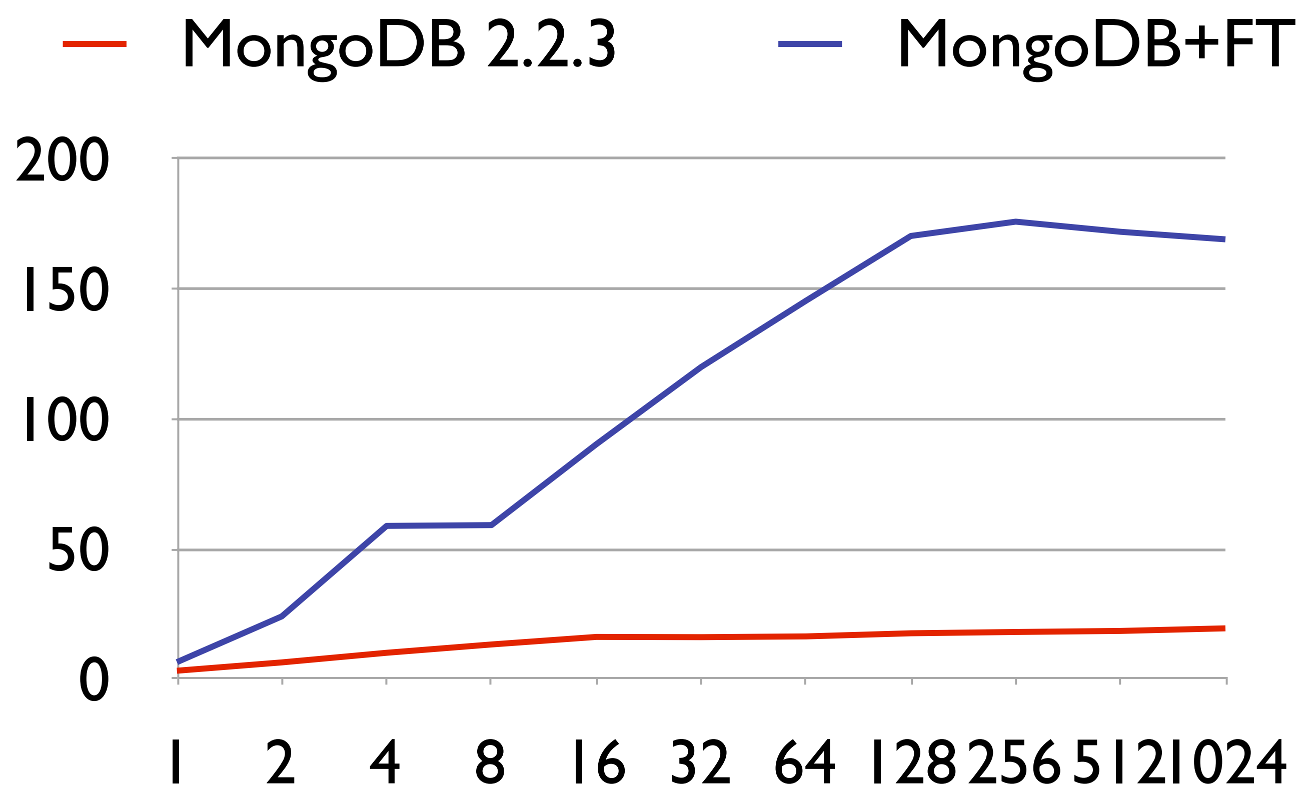
MongoDB : Clustered Secondary Indexes

- Like MySQL, MongoDB supports covering indexes
- As with TokuDB, we support clustering secondary indexes
  - cover all non-indexed attributes
  - no need to drop/create when another attribute is needed
- Compression eliminates size concerns

Tokutek®

MongoDB : > RAM performance

- Sysbench > RAM performance
  - 16G RAM, 16 x 10mm document collections
  - TPS achieved at various client levels



Tokutek

MongoDB : ACID + MVCC

- ACID
  - In MongoDB, multi-insertion operations allow for partial success
    - Asked to store 5 documents, 3 succeeded
  - We offer "all or nothing" behavior
- MVCC
  - In MongoDB, queries can be interrupted by writers.
    - The effect of these writers are visible to the reader
  - We offer MVCC
    - Reads are consistent as of the operation start

Tokutek®

MongoDB : Multi-statement Transactions

- Bringing the following to MongoDB
  - beginTransaction
  - ... do 1 or more operations
  - rollbackTransaction | commitTransaction
- Zardosht has some great blogs
  - http://www.tokutek.com/2013/04/mongodb-transactions-yes/
  - http://www.tokutek.com/2013/04/mongodb-multi-statement-transactions-yes-we-can/

Tokutek®

We’re Hiring!

Looking for Test/Support Ninjas!

Tokutek®

Questions?

Tim Callaghan
VP/Engineering, Tokutek
tim@tokutek.com
@tmcallaghan

Tokutek®