Solving the Challenges of Big Databases with MySQL

Bradley C. Kuszmaul

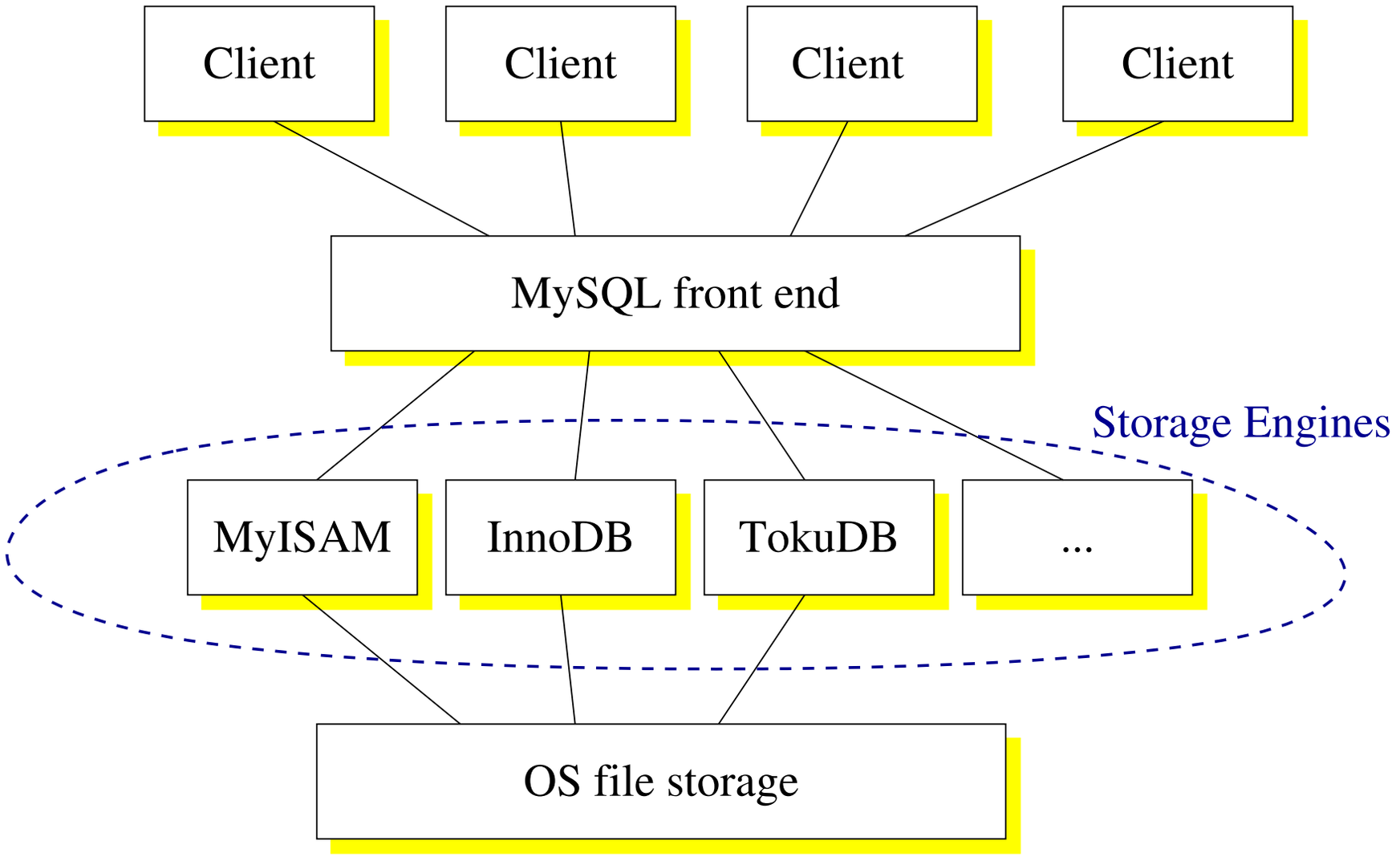
Challenging Activities

Some of the top challenges I hear:

- Loading a big database takes a long time.
- Adding rows to an existing database is slow.
- Adding or removing a column takes my table offline for a long time.
- Adding an index takes my table offline for a long time.
- Backup is difficult. (Not in this talk.)

These activities are painless for small data, but can be painful for big data.

Storage Engines Can Help MySQL



Rest of this talk is about what a storage engine can do, focusing on TokuDB because that's what I know. Tokutek sells TokuDB, a closed-source storage engine for MySQL.

Big Data

- Big data can range from a terabyte to multiple petabytes.
- The difficulties start showing up when your data does not fit in main memory.

Is a Gigabyte Big?

- No, since it fits in main memory even on a cheap server.
- No, since scanning the entire data set off of disk is on the order of seconds.

Is a Terabyte Big?

- A terabyte fits on a small $500 server with a single disk drive. But even the small end of a terabyte can contain billions of rows of data. The little server probably cannot handle much load.
- This machine has too little memory to hold the entire dataset.
- It can take hours to scan the data.

So yes, a terabyte can be big.

Is 10 Terabytes Big?

- You might spend $5,000–$20,000 on a server that can hold ten terabytes on a RAID with with a battery-backed-up controller.
- Now it's really tough to fit the data into memory...

Is a Petabyte Big?

- Yes.
- Some MySQL users have multiple petabytes of data.
- They typically shard the data up across hundreds or thousands of servers.

- Each server suffers from big data problems, and the system as a whole suffers really big data problems.
- Let's focus on the problems a single server faces. These problems show up at a terabyte or so.

Challenges

As you move from millions of rows and gigabytes of data to billions of rows and terabytes of data, these problems get harder:

Challenge I: Loading data

Challenge II: Maintaining indexes under insertions, deletions, and updates.

Challenge III: Adding or removing columns.

Challenge IV: Adding an index online.

Challenge V: Replication (slave lag).

Challenge VI: Compression.

Challenge I: Loading Data

- The first problem is loading the data.
- Two reasons for slow performance:
  1. Fetching one row at a time from a file.
  2. Storing one row at a time onto the database.
- TokuDB speeds up #2, but does not address #1.
- Rich found an 8-fold speedup loading TPCC.
- Perhaps more speedup on more cores (especially if someone addresses #1).

Where does the TokuDB Loader Find Parallelism?

1. Merge sort (especially the in-memory stages) contains a huge amount of parallelism.
2. Compression offers a huge amount of parallelism. TokuDB compresses 4MB blocks.

Challenge II: Maintaining Indexes

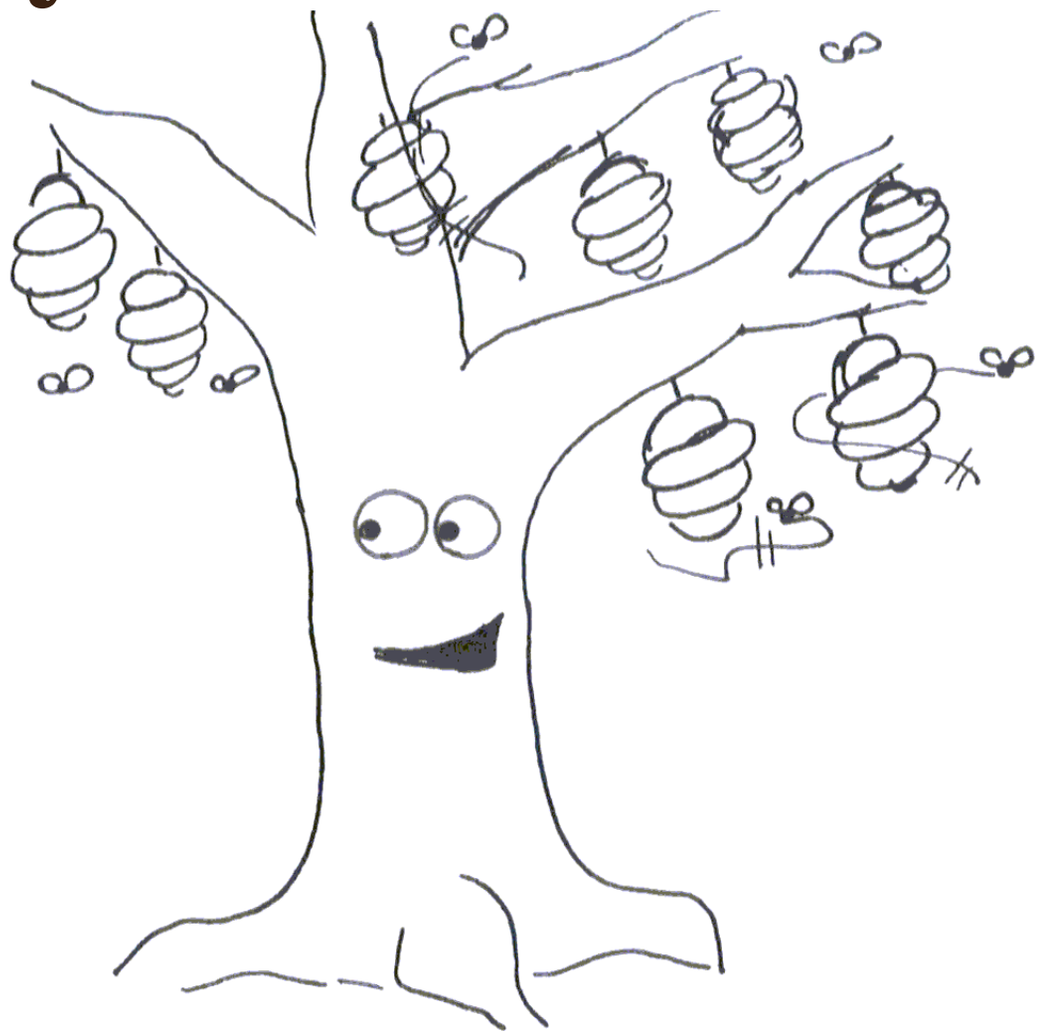
Setup: A big database with many indexes adds, removes, or updates thousands of rows per second.

I'll talk about:

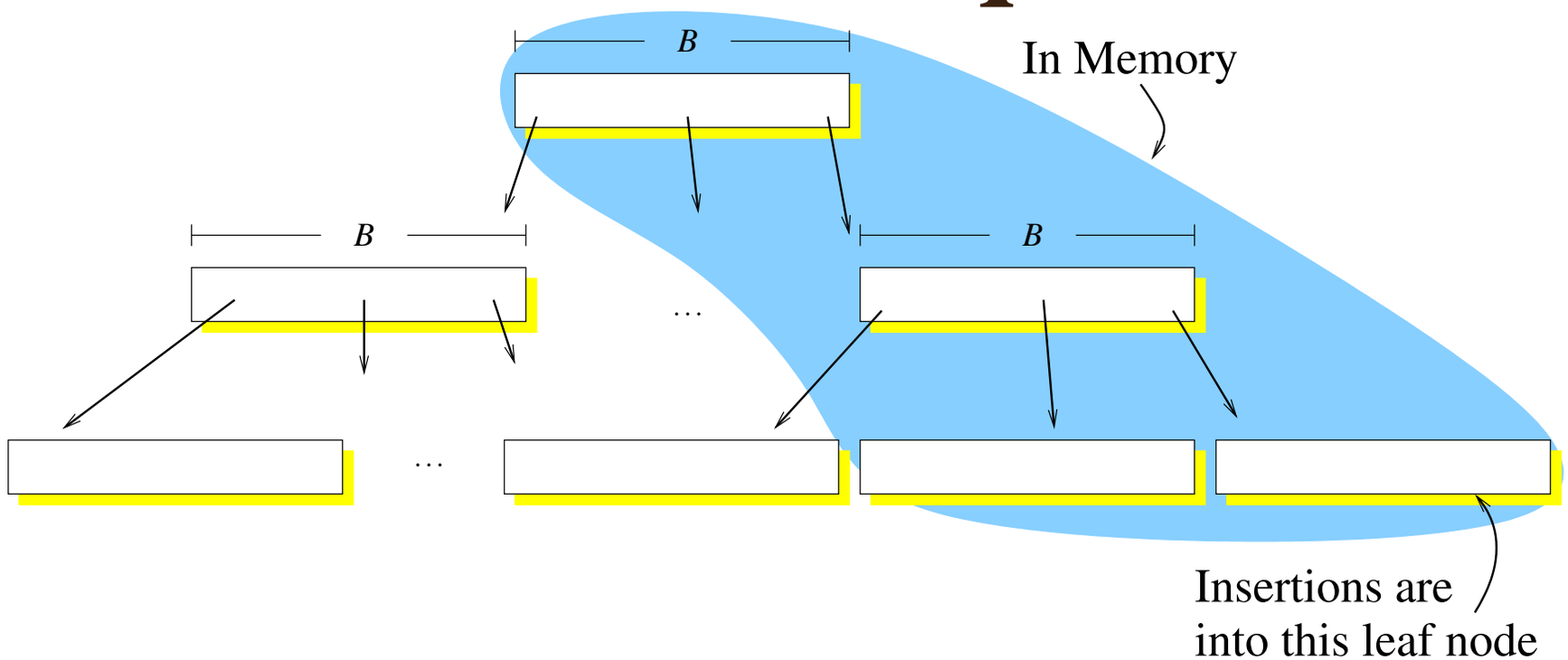
- Why B-trees bottleneck insertion rates.
  InnoDB and MyISAM use B-trees.
- How fractal trees improve speed.
  TokuDB uses fractal trees.

B-Trees are Everywhere

B-Trees show up in database indexes (such as MyISAM and InnoDB), file systems (such as XFS), and many other storage systems.

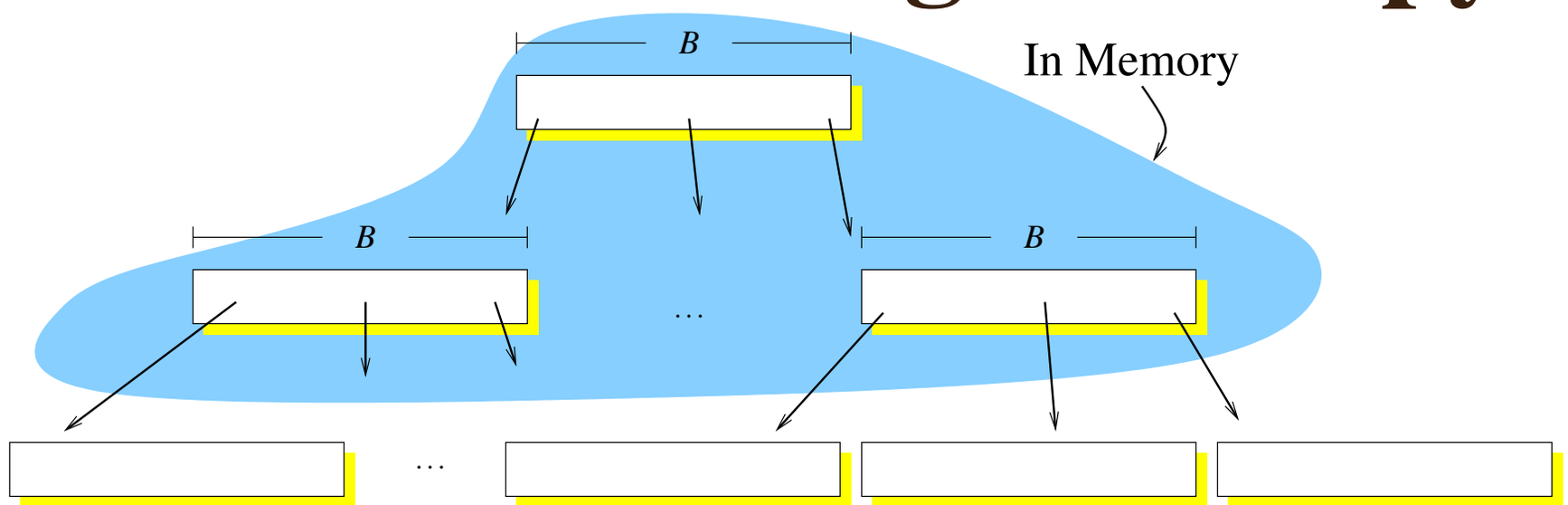


B-Trees are Fast at Sequential Inserts



- One disk I/O per leaf (which contains many rows).
- Sequential disk I/O.
- Performance is limited by *disk bandwidth*.

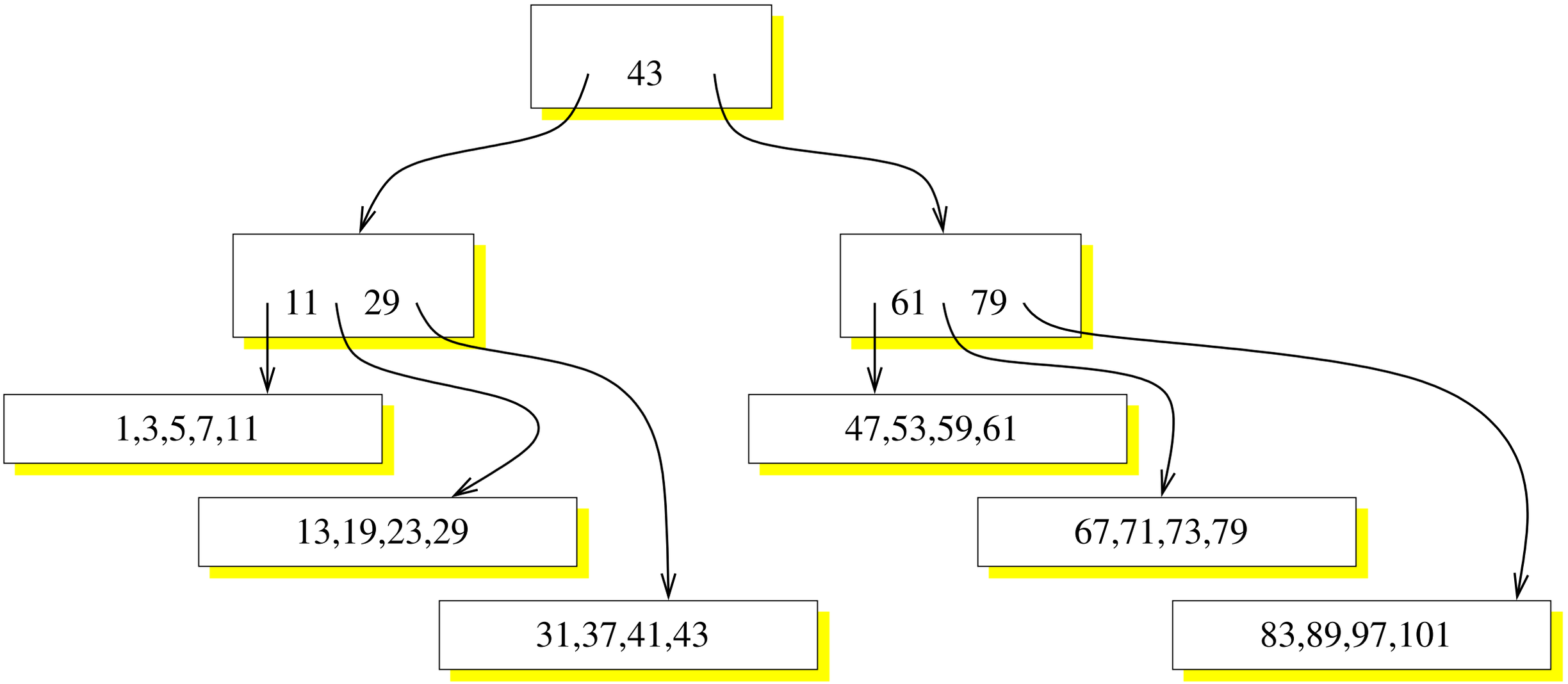
B-Trees are Slow for High-Entropy Inserts



- Most nodes are not in main memory.
- Most insertions require a random disk I/O.
- Performance is limited by *disk head movement*.
- Only 100's of inserts/s/disk ($\leq$ 0.2% of disk bandwidth).

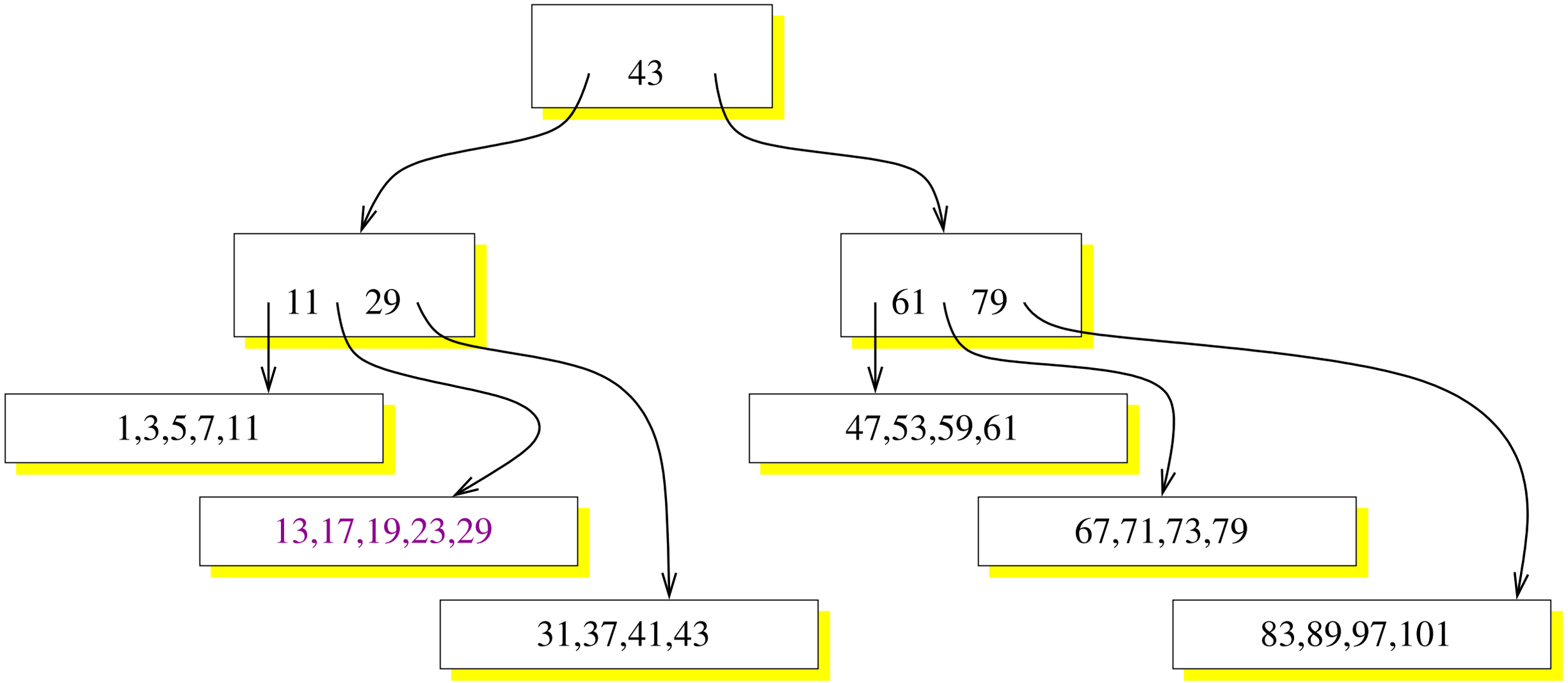
Recipe for a fractal tree

Start with a B-tree:



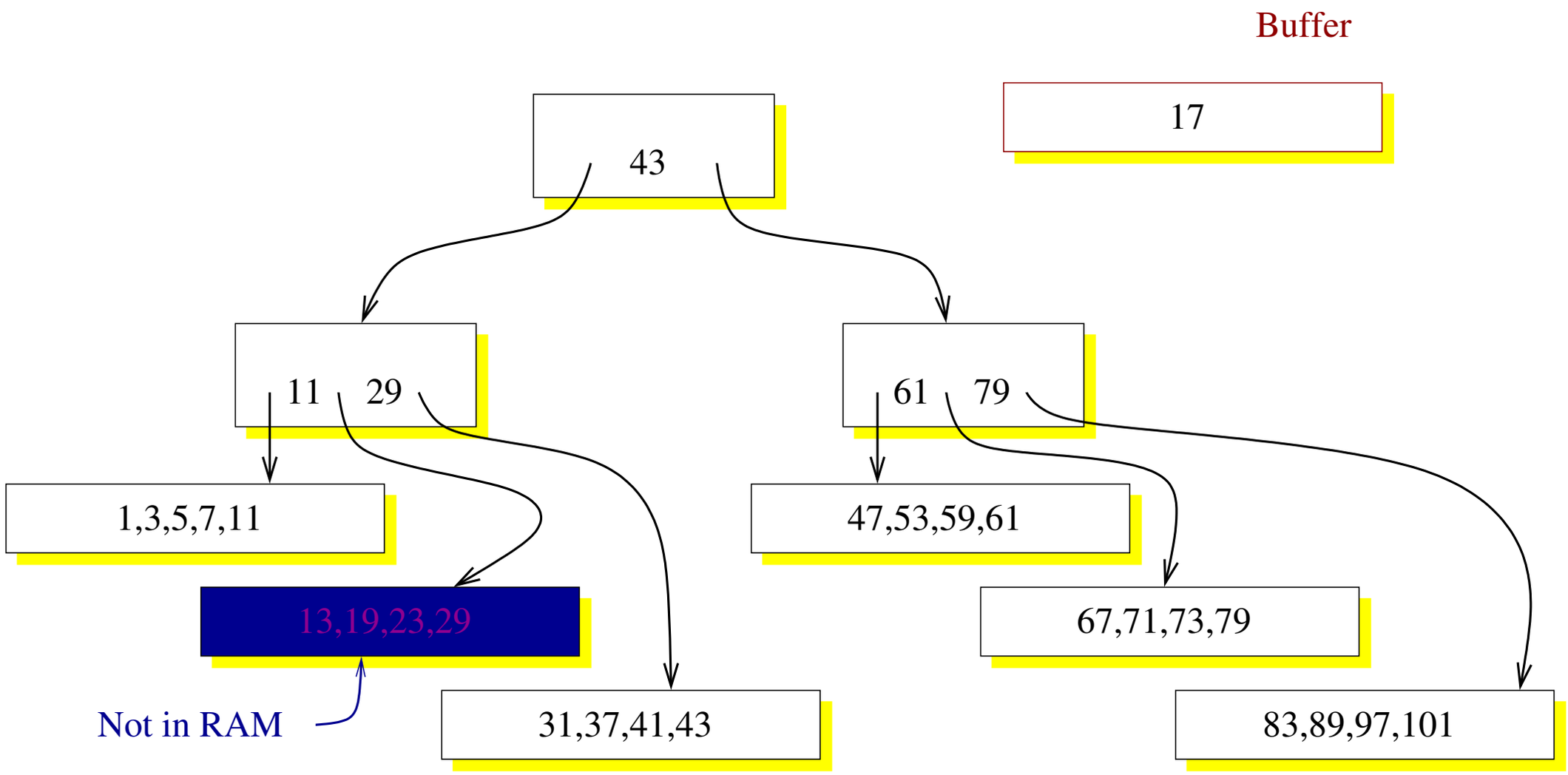
Oops, I forgot 17.

Added 17



Maybe needed several disk I/Os to bring blocks in to store 17.

InnoDB Adds a Buffer

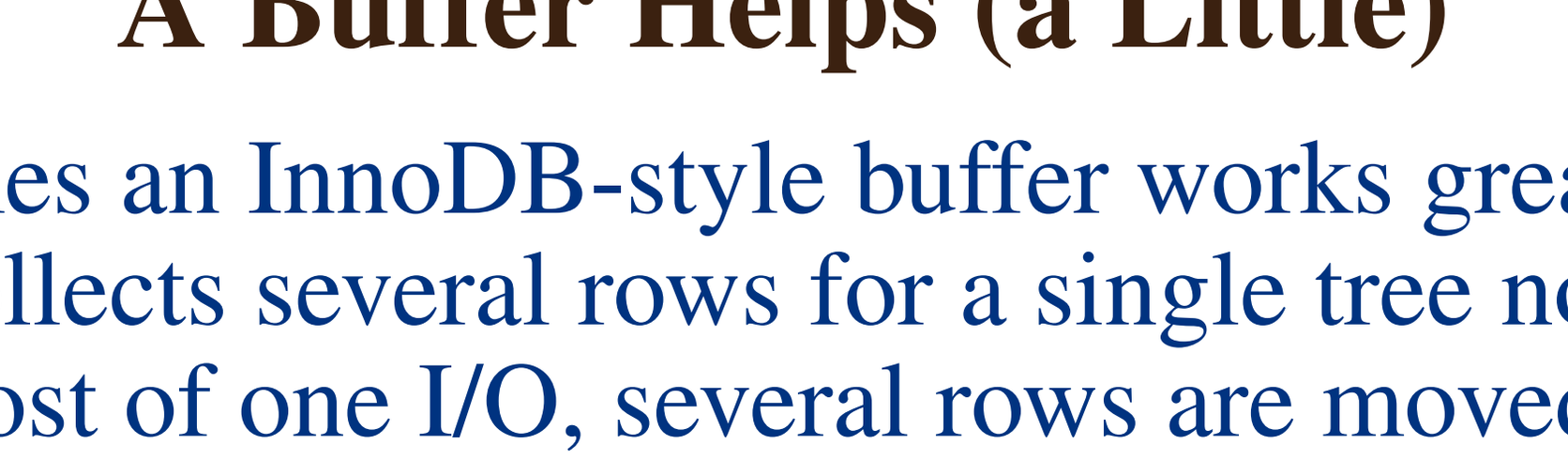


- If a block is on disk (not in RAM) then keep the row in a buffer.
- Later, move rows from the buffer to the tree nodes.

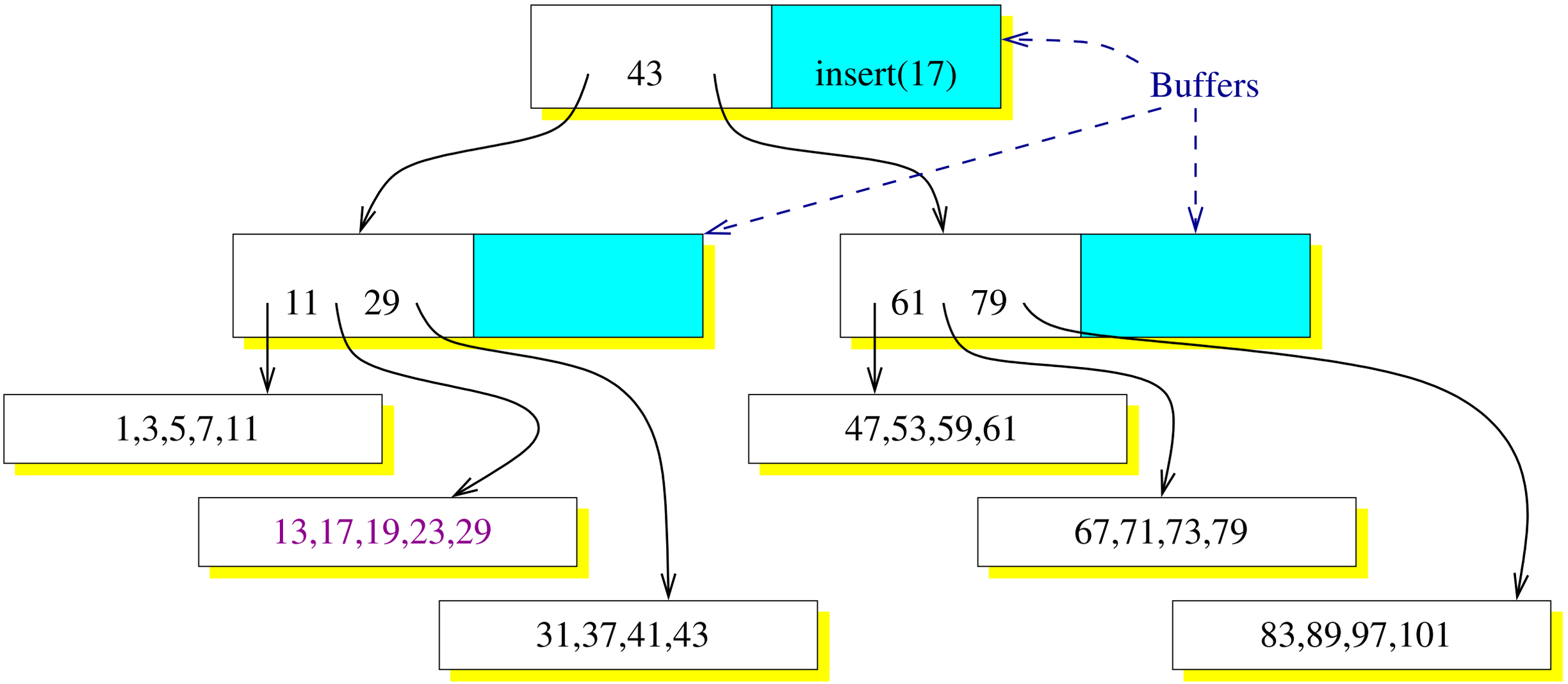
A Buffer Helps (a Little)

Sometimes an InnoDB-style buffer works great: The buffer collects several rows for a single tree node, and for the cost of one I/O, several rows are moved to disk.

Sometimes the buffer fills up, and the system slows down. Even in these situations, an insertion buffer helps (but not by much).

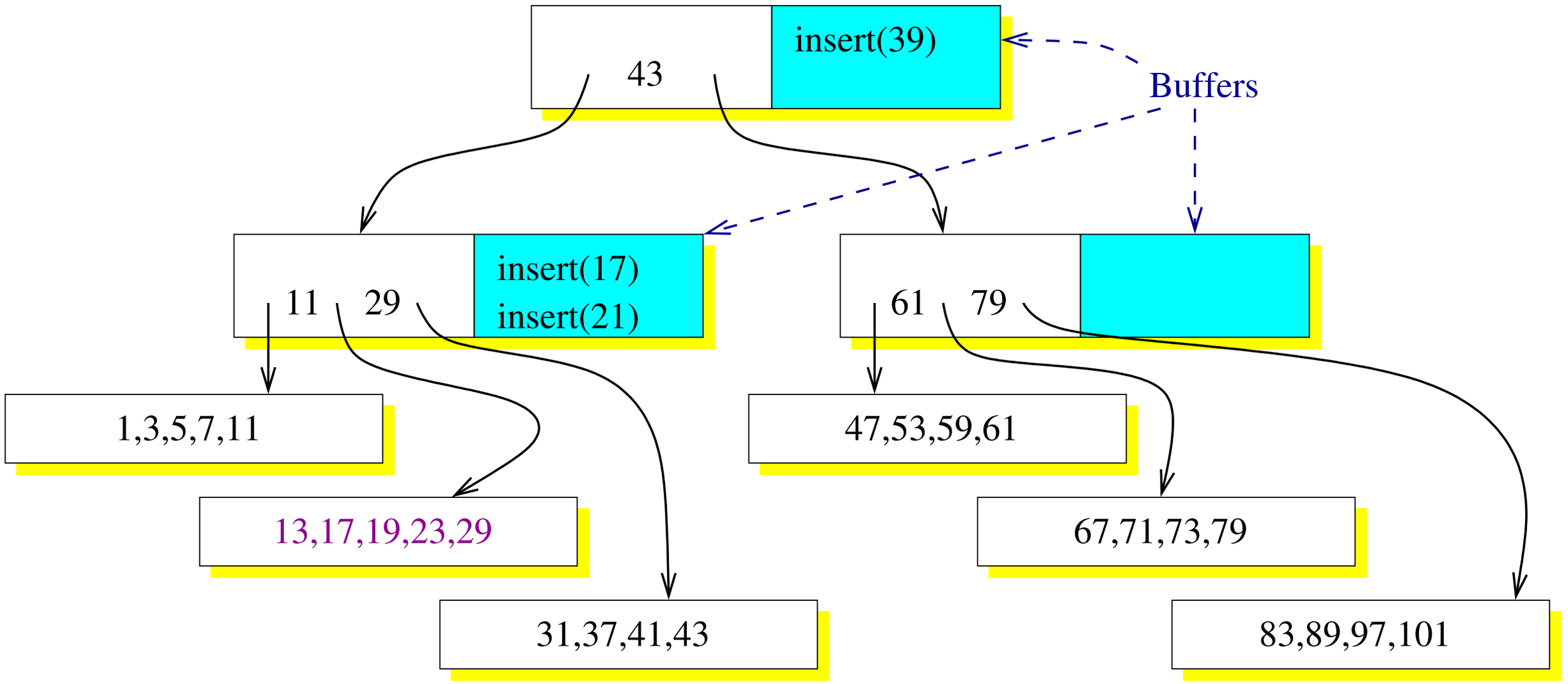


Fractal Trees Indexes Add Many Buffers



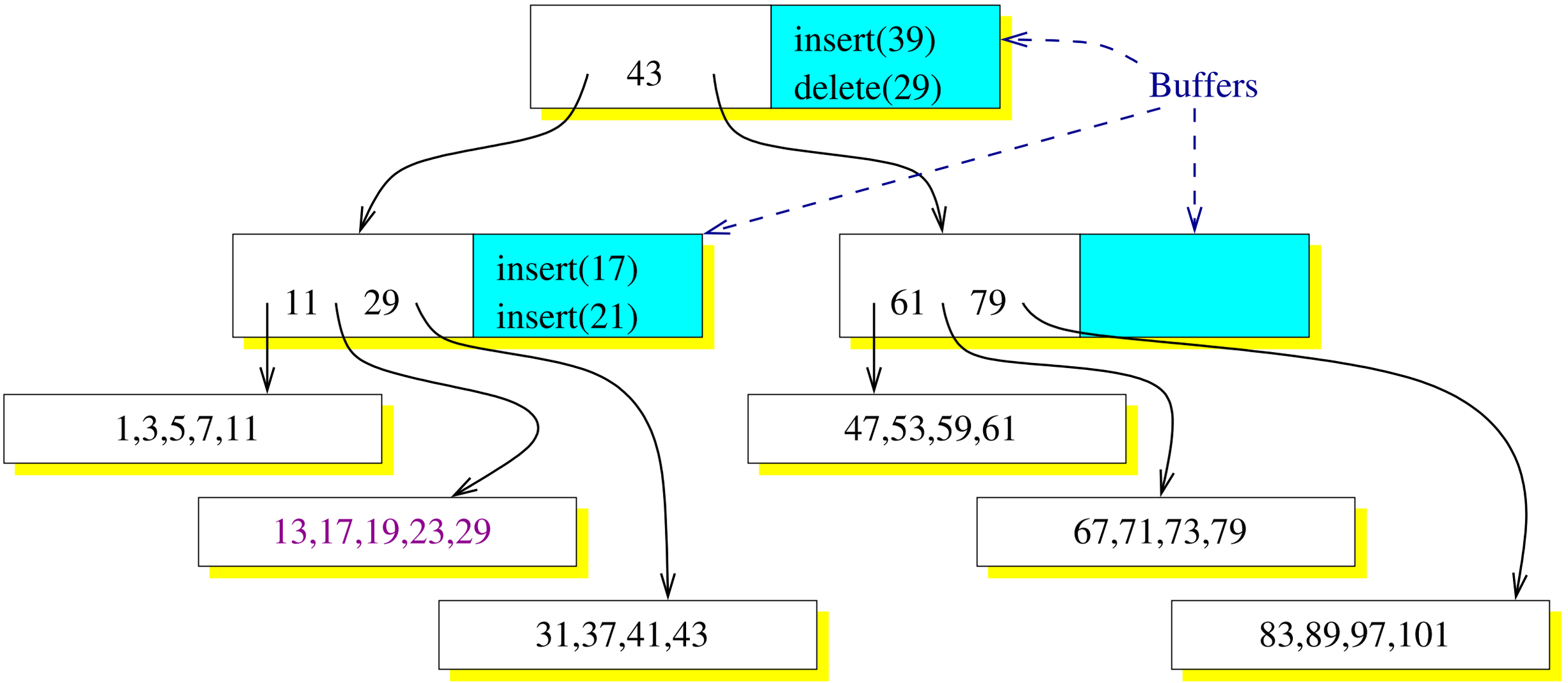
- All the internal nodes of the tree include buffers.
- To insert a row, put a message in the root's buffer.

When a Buffer Overflows



- Move messages down the tree when buffers fill.
- When a message arrives at a leaves, apply it.

Deletes are Also Messages



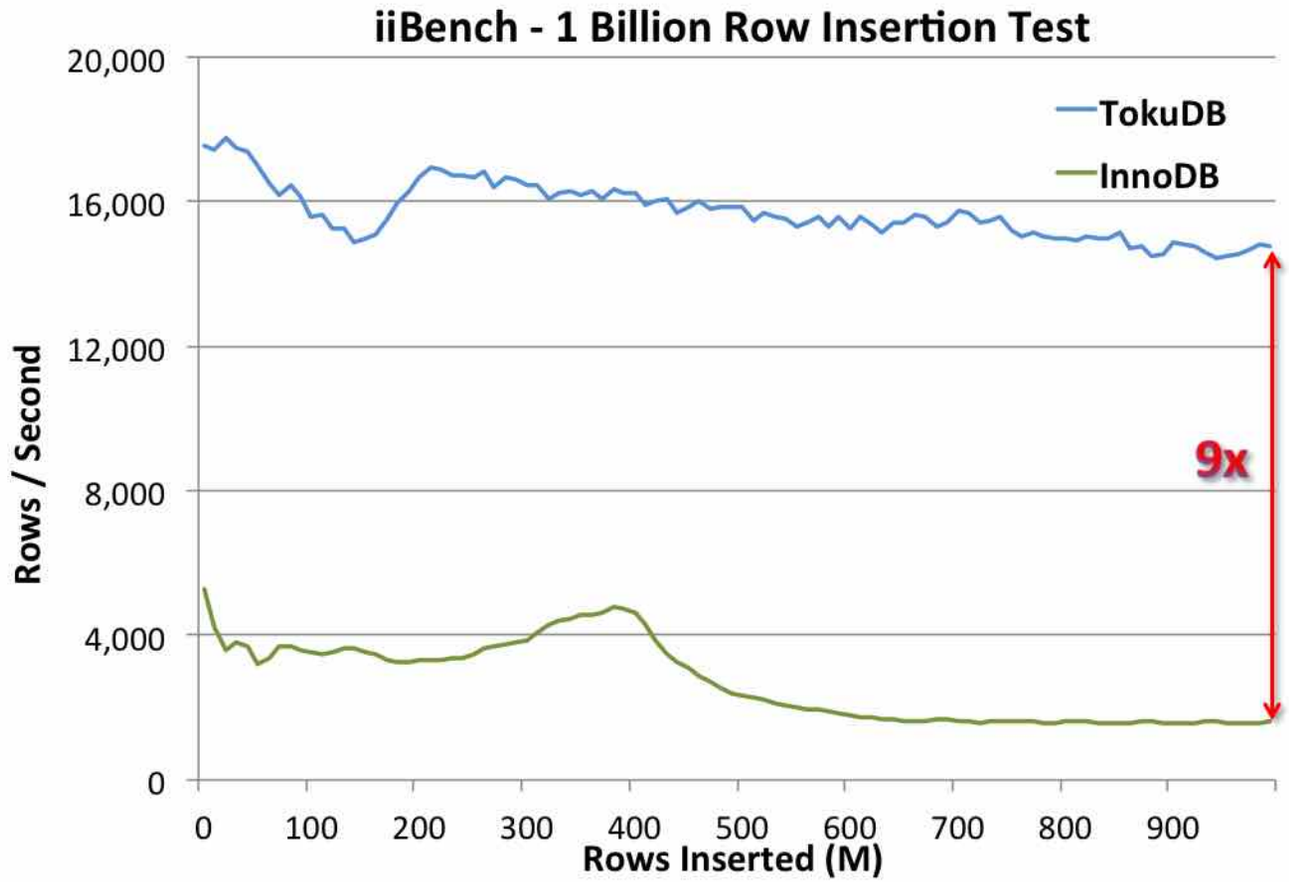
Comparing Data Structures

Disk I/Os are the important metric. How many does it take to do an insertion or a query?

Data structure	Insertion cost	Query cost (in practice)
B-tree	Many I/Os	*One I/O*
LSM tree	*Fraction of an I/O*	Many I/Os
Fractal Tree	*Fraction of an I/O*	*One I/O*

- Fractal tree indexes reduce the cost of index maintenance by two to three orders of magnitude.
- So you can ingest data faster and maintain more indexes.

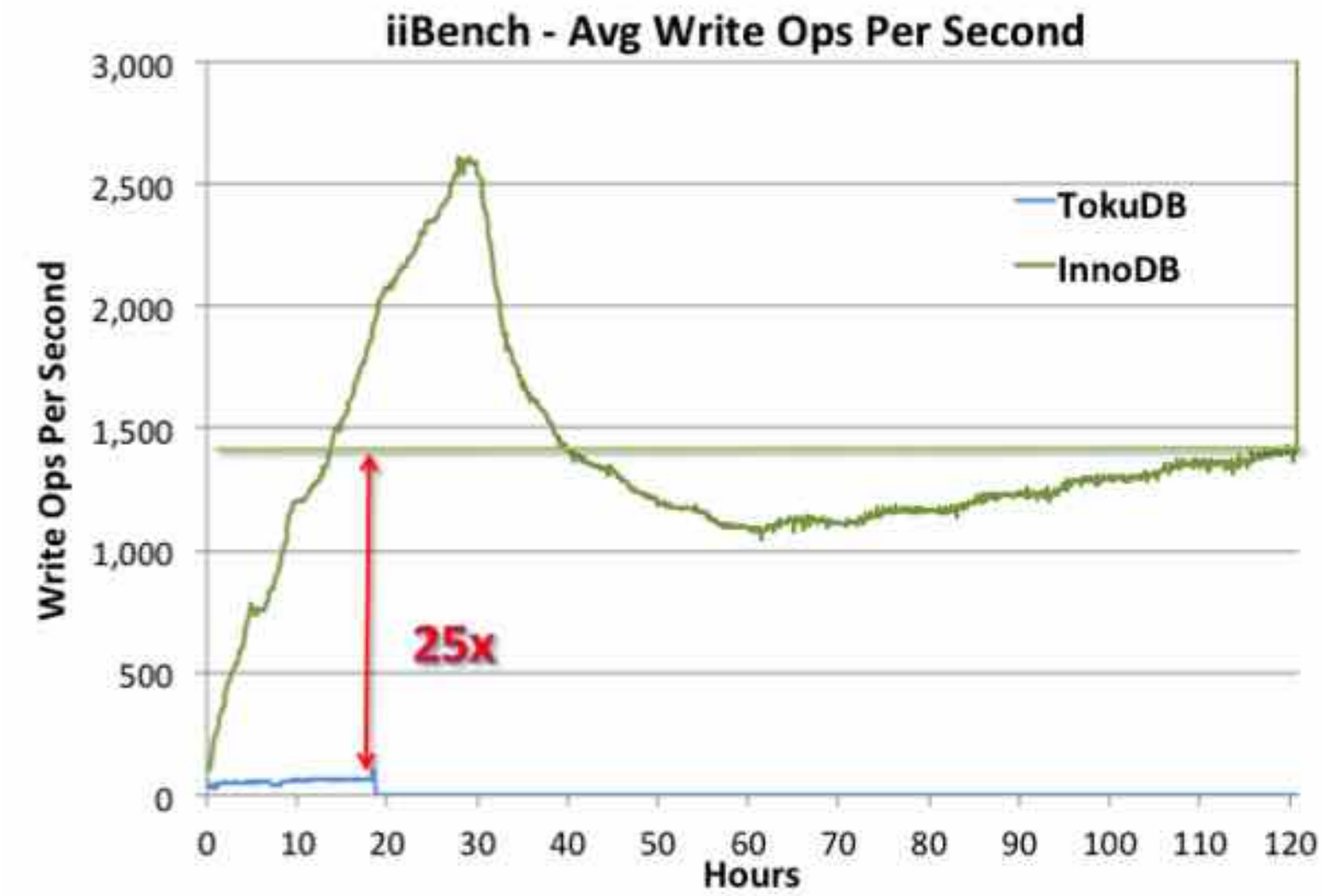
Maintaining Indexes on SSD



Inserting one billion rows, maintaining three secondary indexes. TokuDB's terminal insertion rate was 14,532 inserts/s. InnoDB's was 1,607 inserts/s.

Platform: TokuDB v6.5, Centos 5.6; 2x Xeon L5520; 72GB RAM; LSI MegaRaid 9285; 2x 256GB Samsung 830 in RAID0.

Reducing Wear on SSD



- 161 times fewer write operations.
- 17 times fewer bytes written.

SSDs Wear Out

- Modern SSDs wear out after about 1000 overwrites.
- sysbench achieves 100MB/s on random writes on a 256GB Samsung 830, overwriting in 7 hours, and wearing out in 300 days.
- In this scenario, the FTL amplifies writes by another 4x, so it will last only 75 days.
- InnoDB is a little slower than sysbench, and would wear it out in about 246 days.
- Under the same workload, TokuDB would last 11 years.

Covering Indexes

Arranging for your queries to be index-only can make huge speedups. For example see this guest blog posting from Baron Schwartz of Percona:

http://www.tokutek.com/2011/09/are-you-forcing-mysql-to-do-twice-as-many-joins-as-necessary/

Challenge III: Adding a Column

Many DBAS have had the following experience:

- Load a bunch of data into a table and set up the associated indexes.
- Realize that adding a column would be helpful.

This becomes painful for big data.

A Painful Way to Add a Column

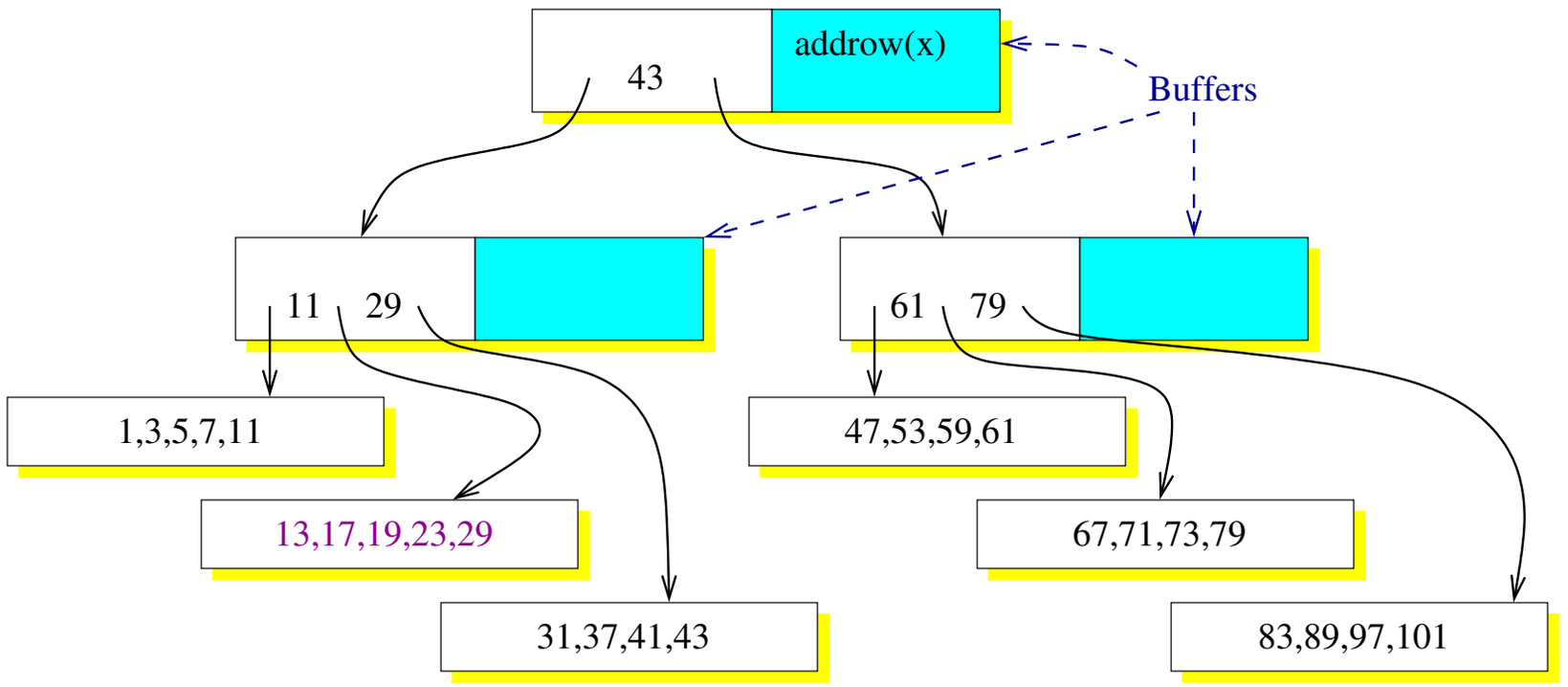
The MyISAM / InnoDB way:

- Lock the table.
- Scan through the table, changing every row to the new format (adding the column).
- Unlock the table.

This process can take many hours during which time the system permits no modifications to the table.

The bigger the data the longer the downtime.

A Fast Way to Add a Column



Send a "broadcast" message into the fractal tree index.

- As the message descends tree the message is replicated.
- When it arrives at a leaf, update the leaf's rows.

TokuDB can add or delete a column with essentially no additional I/O.

Q: How Fast is Hot Column Addition?

Zardosht wrote about a (typical) example:

```
alter table ontime add column totalTime int default 0;
```

Hot column addition reduced the schema change time from 17 hours to 3 seconds:

http://www.tokutek.com/2011/03/hot-column-addition-and-deletion-part-i-performance

A: It's fast

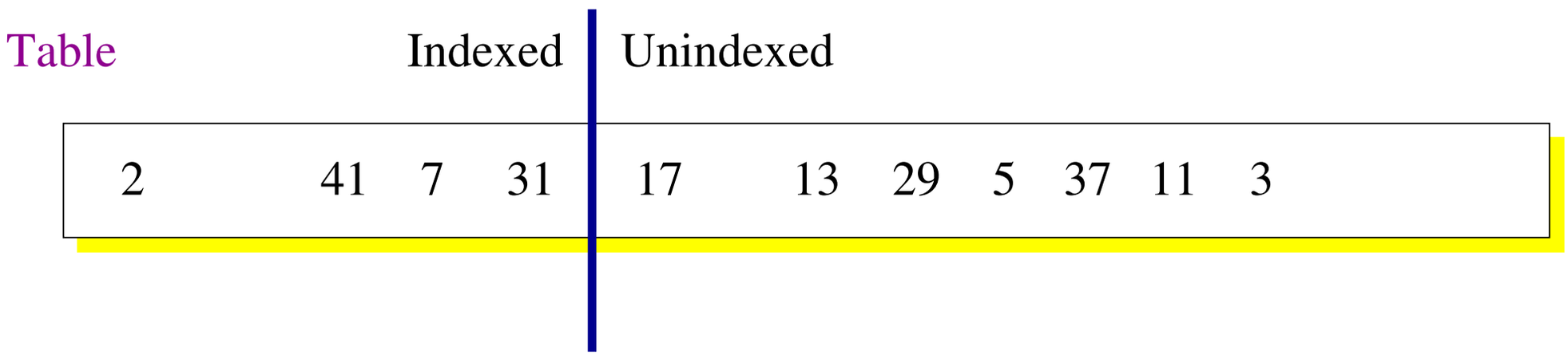
Challenge IV: Adding an Index

Adding an index to a big database can be painful.
The problem: What if someone adds a row while we are building the index? The row might not make it into the index, so the index is corrupted.

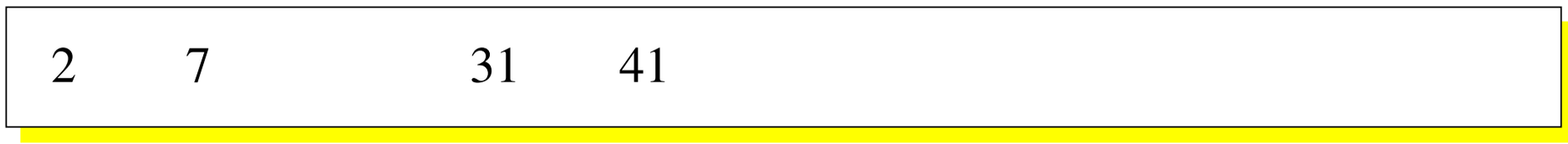
Many storage engines lock the table when adding an index.

A Fast Way to Add an Index

Scan through the table creating the index.

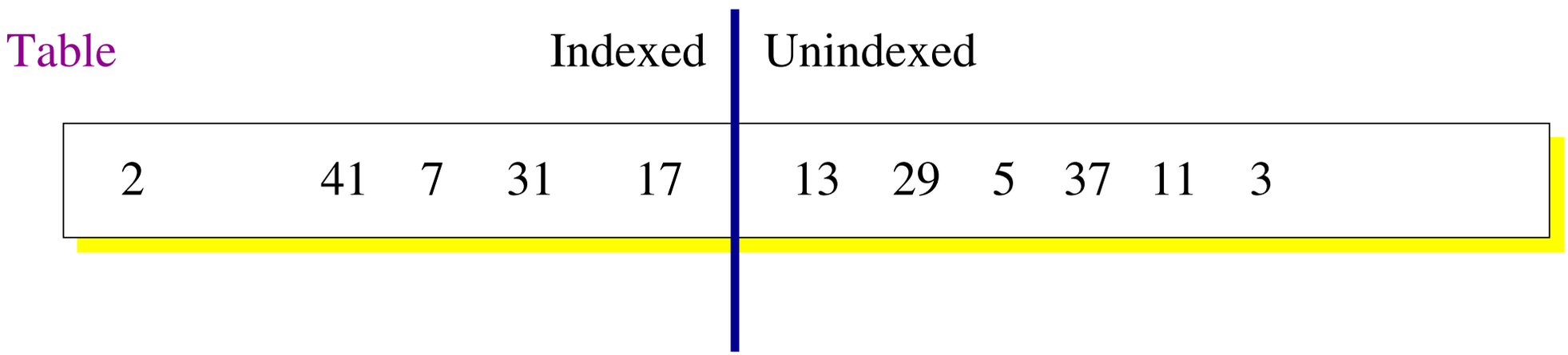


Partially constructed index

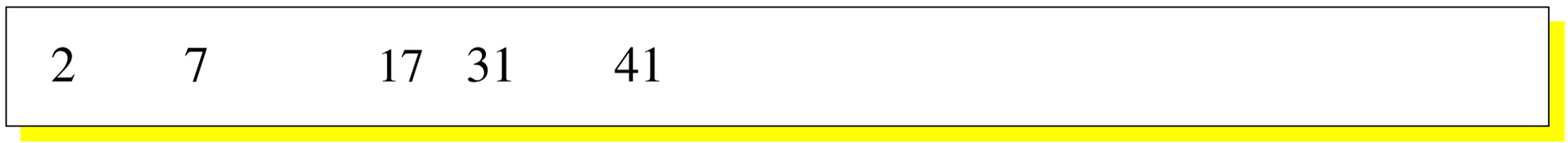


A Fast Way to Add an Index

Scan through the table creating the index.

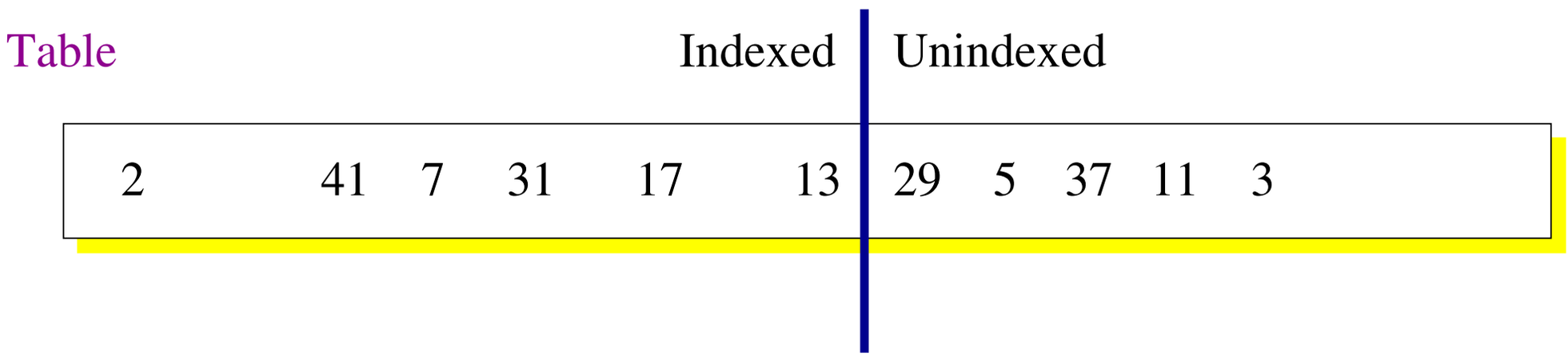


Partially constructed index

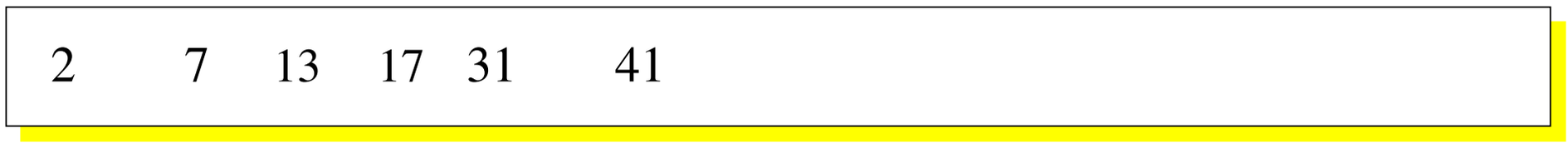


A Fast Way to Add an Index

Scan through the table creating the index.

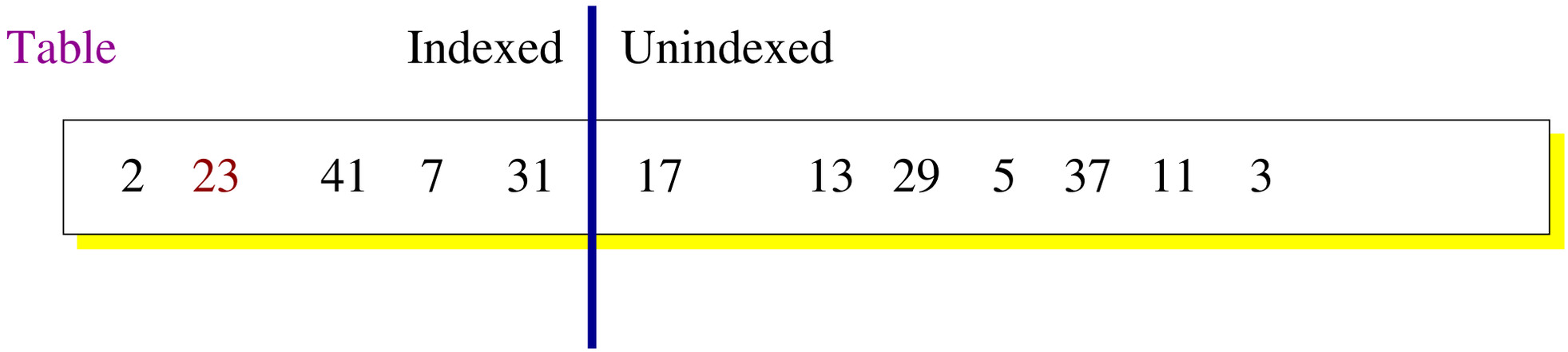


Partially constructed index

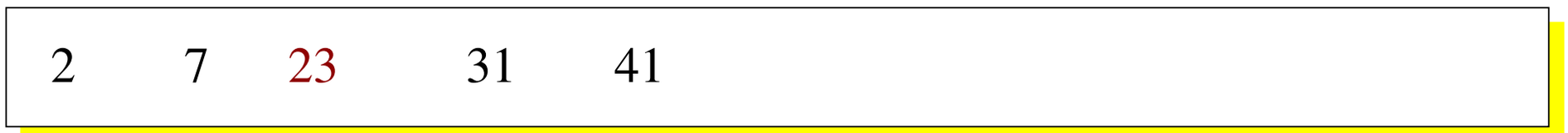


Handling Updates I

If another thread updates something that's been scanned, then update the index

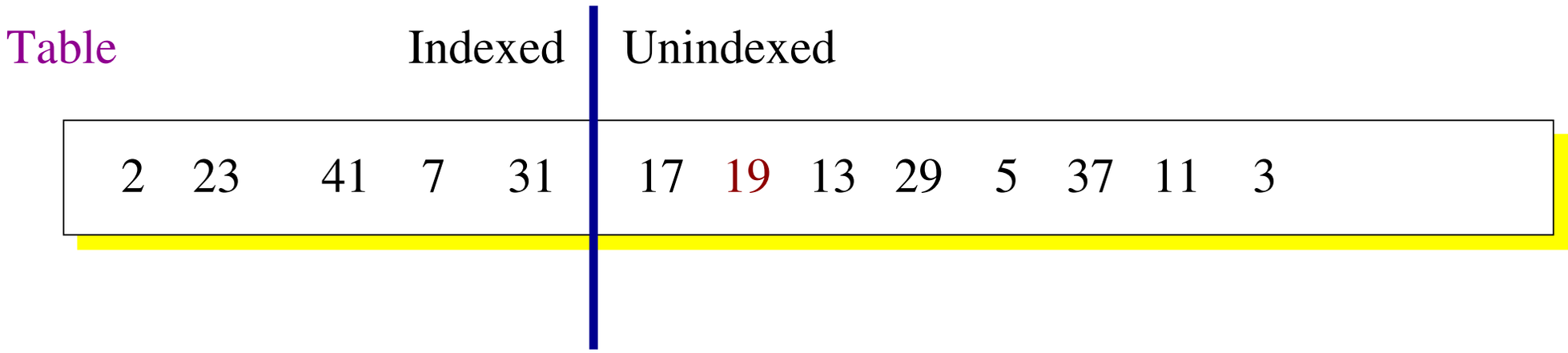


Partially constructed index

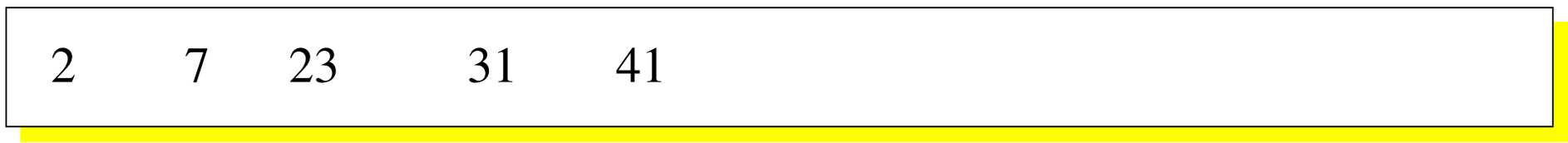


Handling Updates II

If another thread updates something that has not been scanned, then do not update the index, since the indexer will pick up the value.



Partially constructed index



Q: How Fast is Hot Index Addition?

Zardosht wrote about a (typical) example:

- InnoDB took 31m 34s.
- TokuDB took 9m 30s.
- But the table was locked for less than 2s.

`http://www.tokutek.com/2011/04/hot-indexing-part-i-new-feature`

A: It's fast

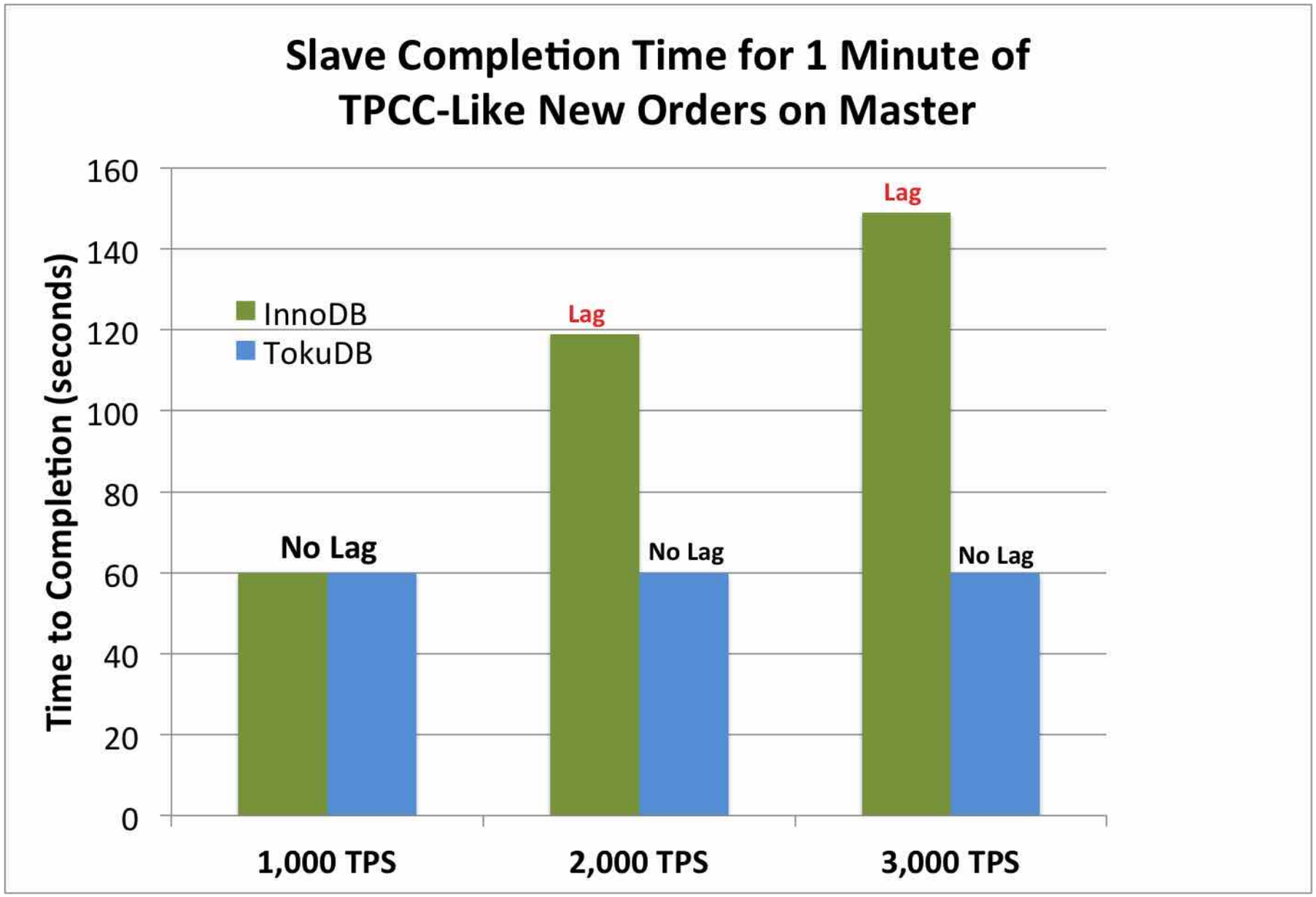
Challenge V: Replication (Slave Lag)

Many systems employ MySQL's master-slave replication

- to distribute read workloads
- to build backups, and
- to implement geographically distributed disaster recovery.

But MySQL slaves often fall behind, since the replication is single-threaded (even the latest multithreaded slaves are single threaded in each database).

TokuDB Keeps Up



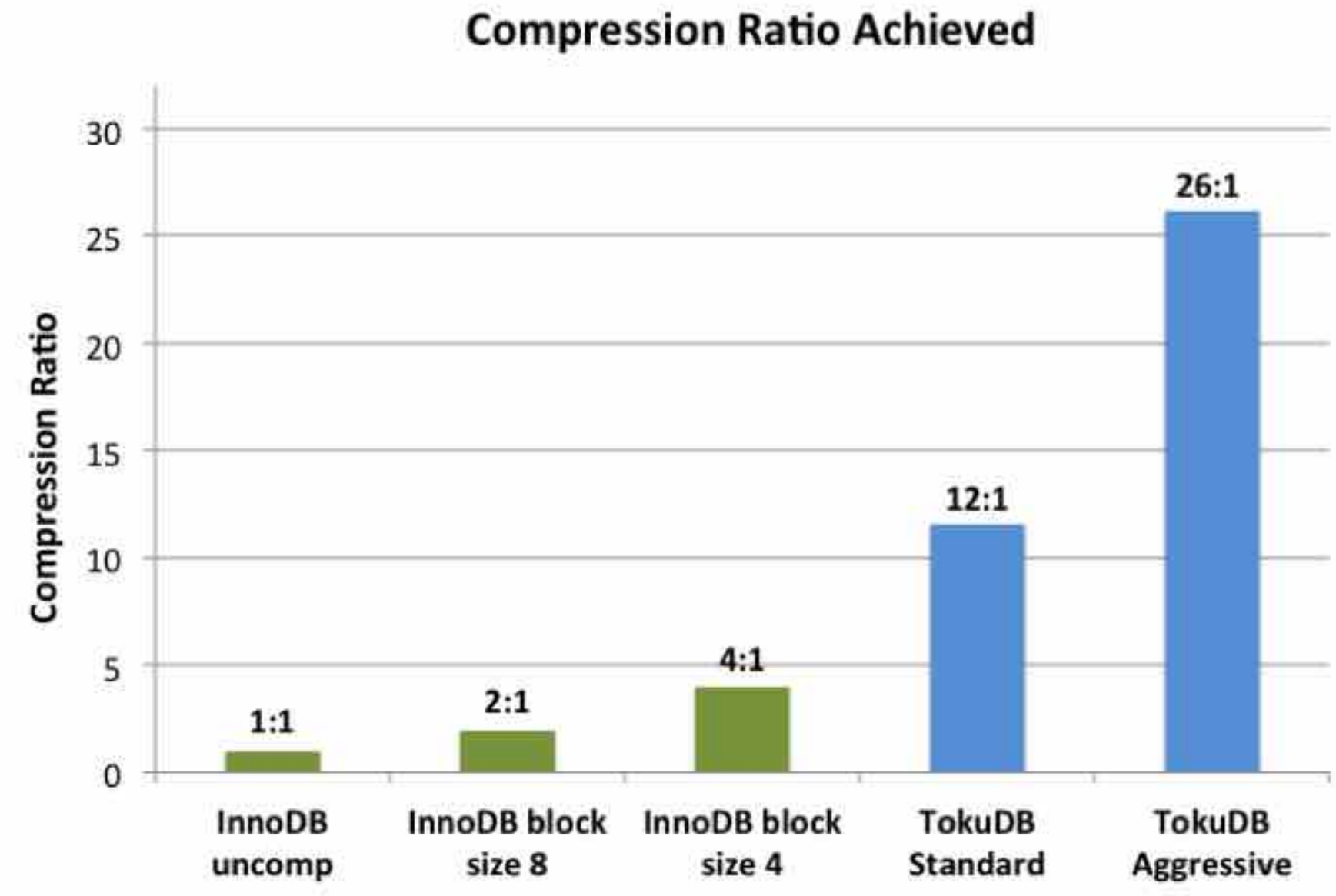
After 60s at 3000TPS, InnoDB was 140s behind, but TokuDB is still keeping up.

Challenge VI: Compression

Compression can reduce the cost of maintaining your database.

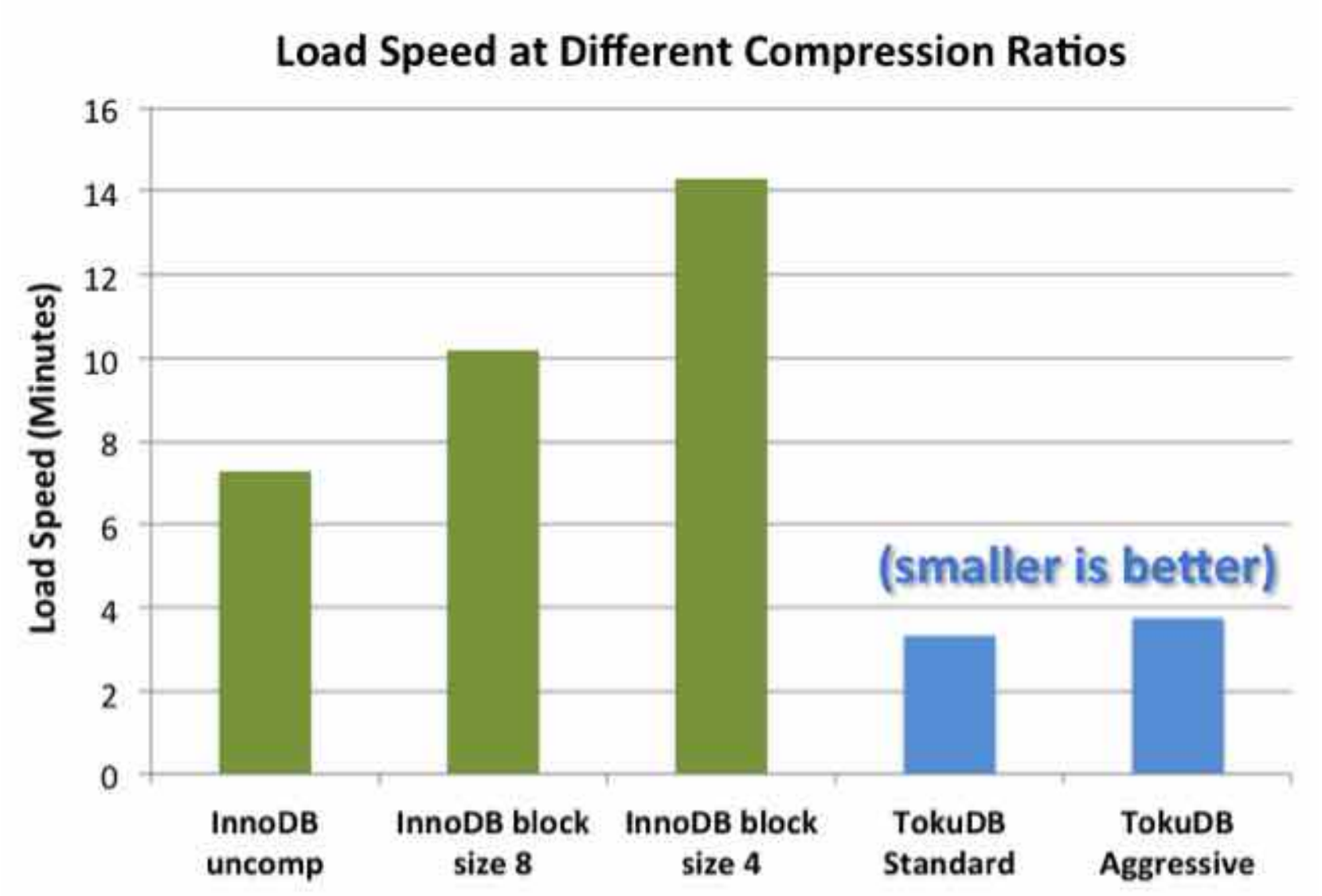
- For SSD, space costs money, so compression saves capital costs.
- For rotating disks, compression improves performance.

Aggressive Compression

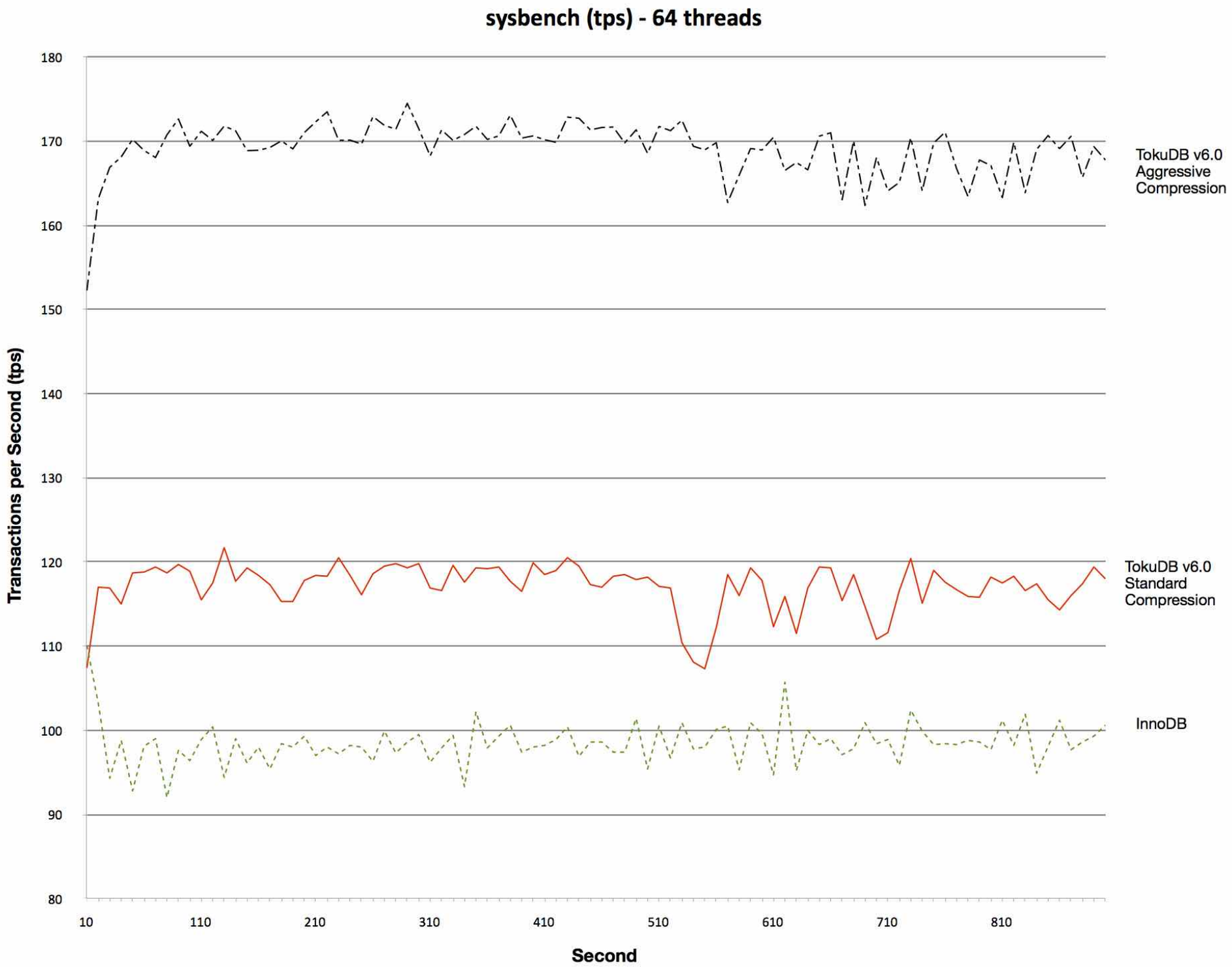


Log-style data with stored procedure names, database instance names, begin and ending execution timestamps, duration row counts, and parameter values.

Compression Speeds Up Loading



Compression Speeds Up The DB



Why not Always Use Aggressive Compression?

- If you have less than six cores in your server, the extra CPU work from the compression workload compression work might slow you down: Use standard compression.
- For modern servers, there are plenty of CPU cycles to do the compression: Use aggressive compression.

Big Data Challenges

If you have big data, you will find these challenges (and others):

- Loading data.
- Inserting, updating, and deleting rows while maintaining indexes.
- Schema changes.
- Replication (slave lag)
- Compression.

TokuDB 6.5 was announced this week and is available now.

More Information

You can download this talk and others at

http://tokutek.com/technology