Write-Optimization in B-Trees

Bradley C. Kuszmaul
MIT & Tokutek

The Setup

What and Why B-trees?

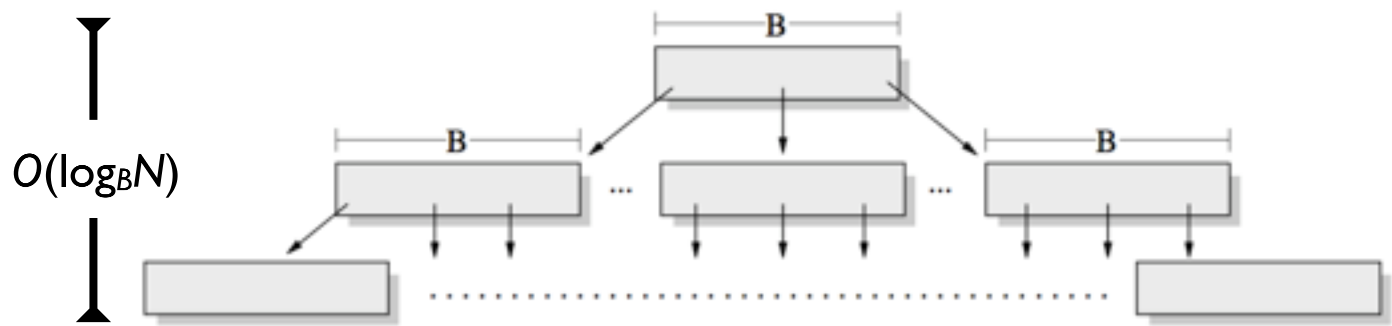
B-trees are a family of data structures.

B-trees implement an ordered map.

- Implement GET, PUT, NEXT, PREV.

B-trees are a wide-fanout tree.

- It's a **tree** so that we can implement NEXT and PREV.
- It's wide-fanout so to reduce the depth of the tree.
  - A binary tree has depth $O(\log_2 N)$.
  - A B-tree with fanout B has depth $O(\log_B N)$.
- Why does this matter?



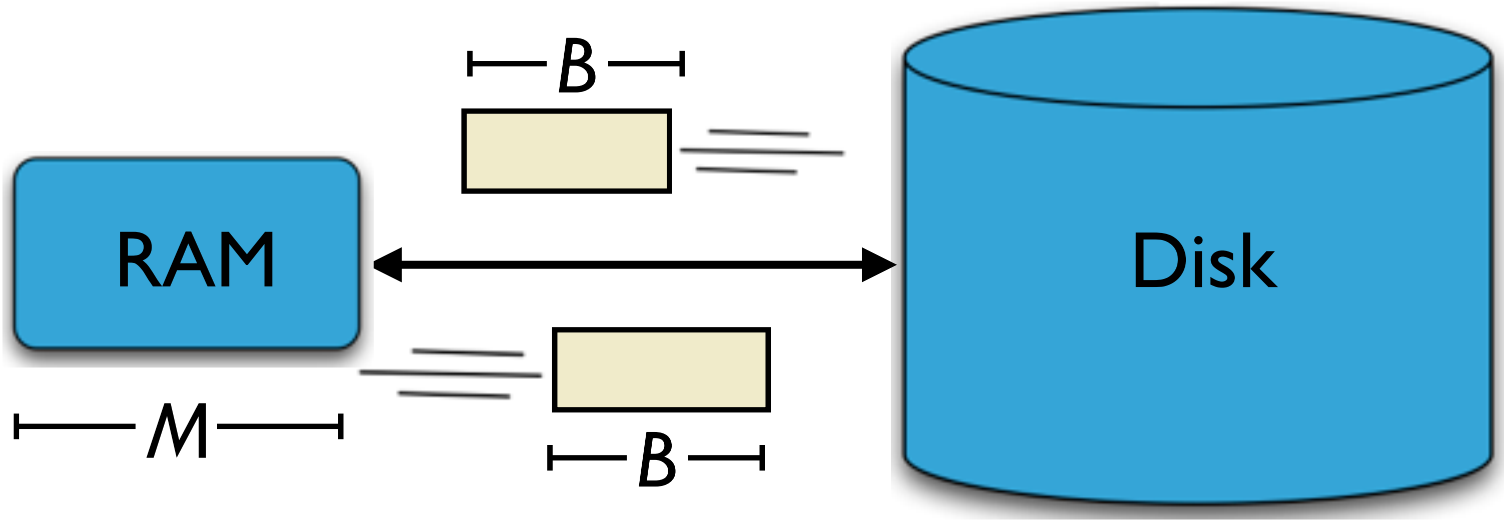
An algorithmic performance model

How computation works:

- Data is transferred in blocks between RAM and disk.
- The number of block transfers dominates the running time.

Goal: Minimize # of block transfers

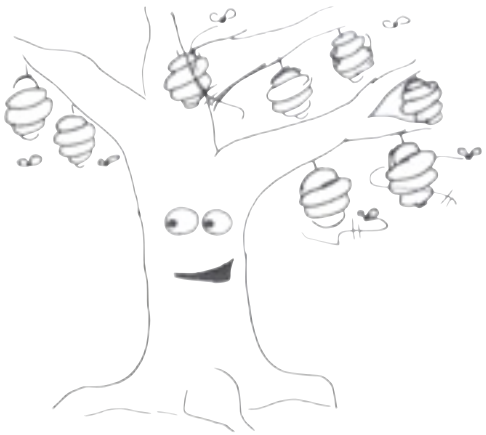
- Performance bounds are parameterized by block size ***B***, memory size ***M***, data size ***N***.

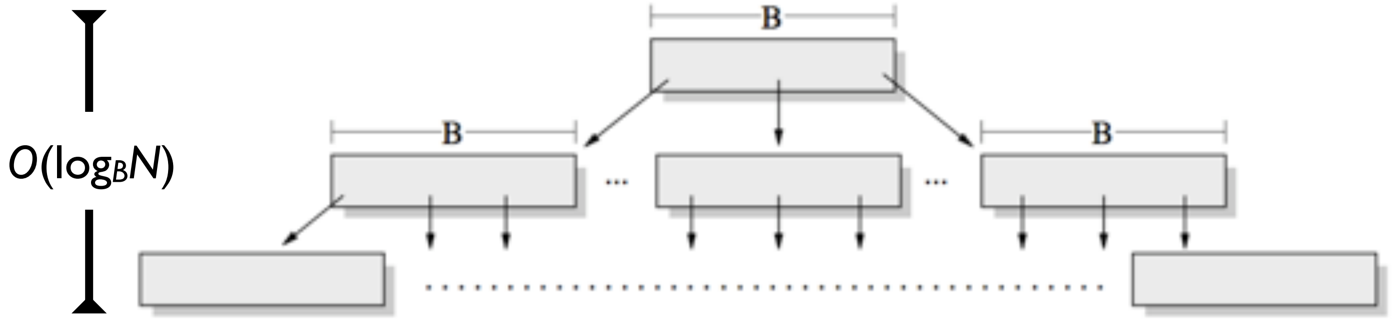


[Aggarwal+Vitter '88]

B-Trees Gain a Factor of O(log *B*)

B-tree point queries: $O(\log_B N)$ I/Os for PUT or GET.





Binary tree search: $O(\log_2 N)$ I/Os.

Slower by a factor of $O(\log B)$

Write-optimized B-Trees Can PUT faster

Data structures: [O'Neil,Cheng, Gawlick, O'Neil 96], [Buchsbaum, Goldwasser, Venkatasubramanian, Westbrook 00], [Argel 03], [Graefe 03], [Brodal, Fagerberg 03], [Bender, Farach,Fineman,Fogel, Kuszmaul, Nelson'07], [Brodal, Demaine, Fineman, Iacono, Langerman, Munro 10], [Spillane, Shetty, Zadok, Archak, Dixit 11].
Systems: BigTable, Cassandra, H-Base, LevelDB, TokuDB.

	B-tree	Some write-optimized structures
Insert/delete	$O(\log_B N) = O\left(\frac{\log N}{\log B}\right)$	$O\left(\frac{\log N}{B}\right)$

- If B=1024, then insert speedup is $B/\log B \approx 100$.
- Hardware trends mean bigger B, bigger speedup.
- Less than 1 I/O per insert.

Optimal Search-Insert Tradeoff [Brodal, Fagerberg 03]

	insert	point query
Optimal tradeoff (function of ε=0...1)	$O\left(\frac{\log_{1+B^\varepsilon} N}{B^{1-\varepsilon}}\right)$	$O\left(\log_{1+B^\varepsilon} N\right)$
B-tree (ε=1)	$O\left(\log_B N\right)$	$O\left(\log_B N\right)$
ε=1/2	$O\left(\frac{\log_B N}{\sqrt{B}}\right)$	$O\left(\log_B N\right)$
ε=0	$O\left(\frac{\log N}{B}\right)$	$O\left(\log N\right)$

10x-100x faster inserts

Illustration of Optimal Tradeoff

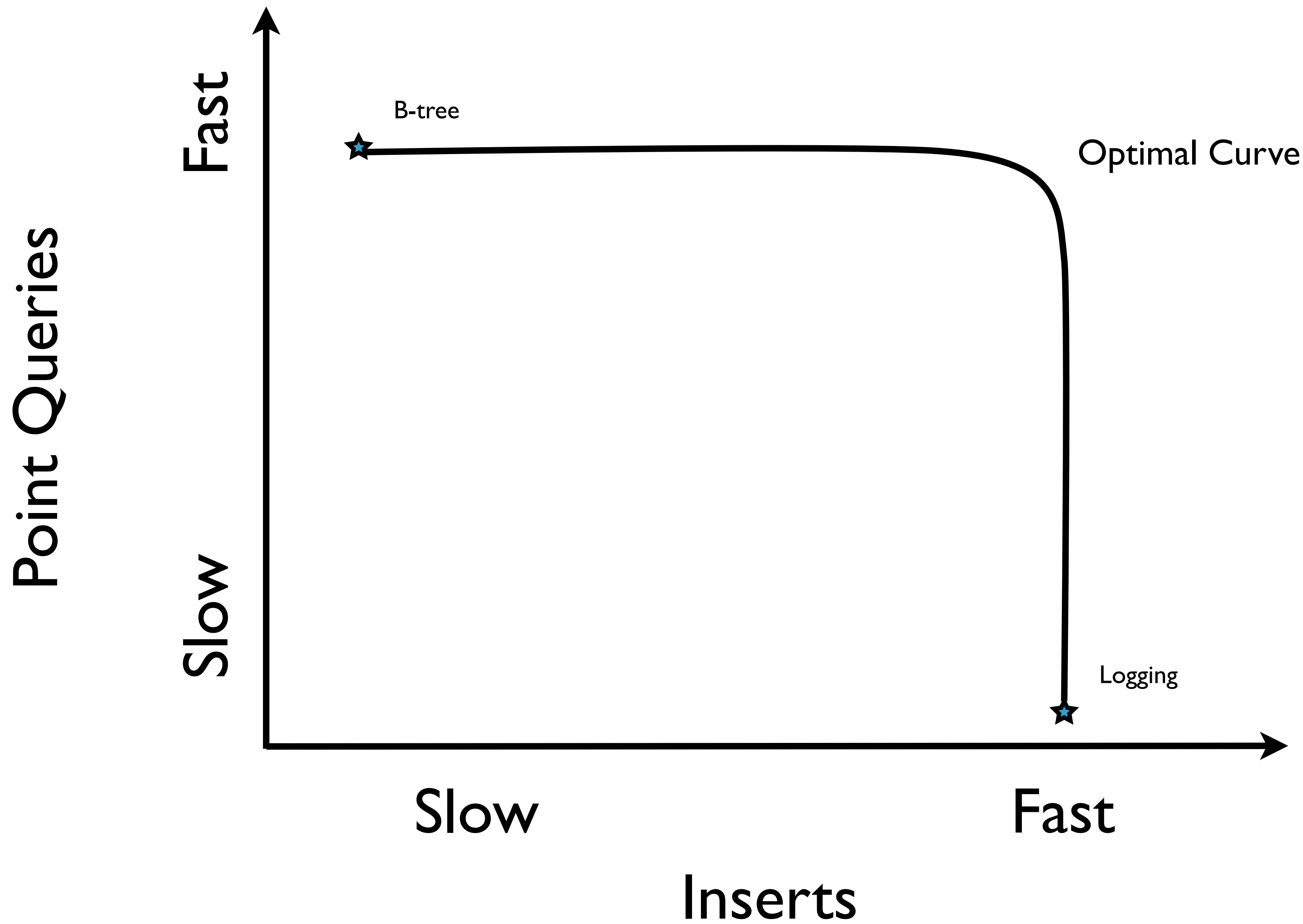
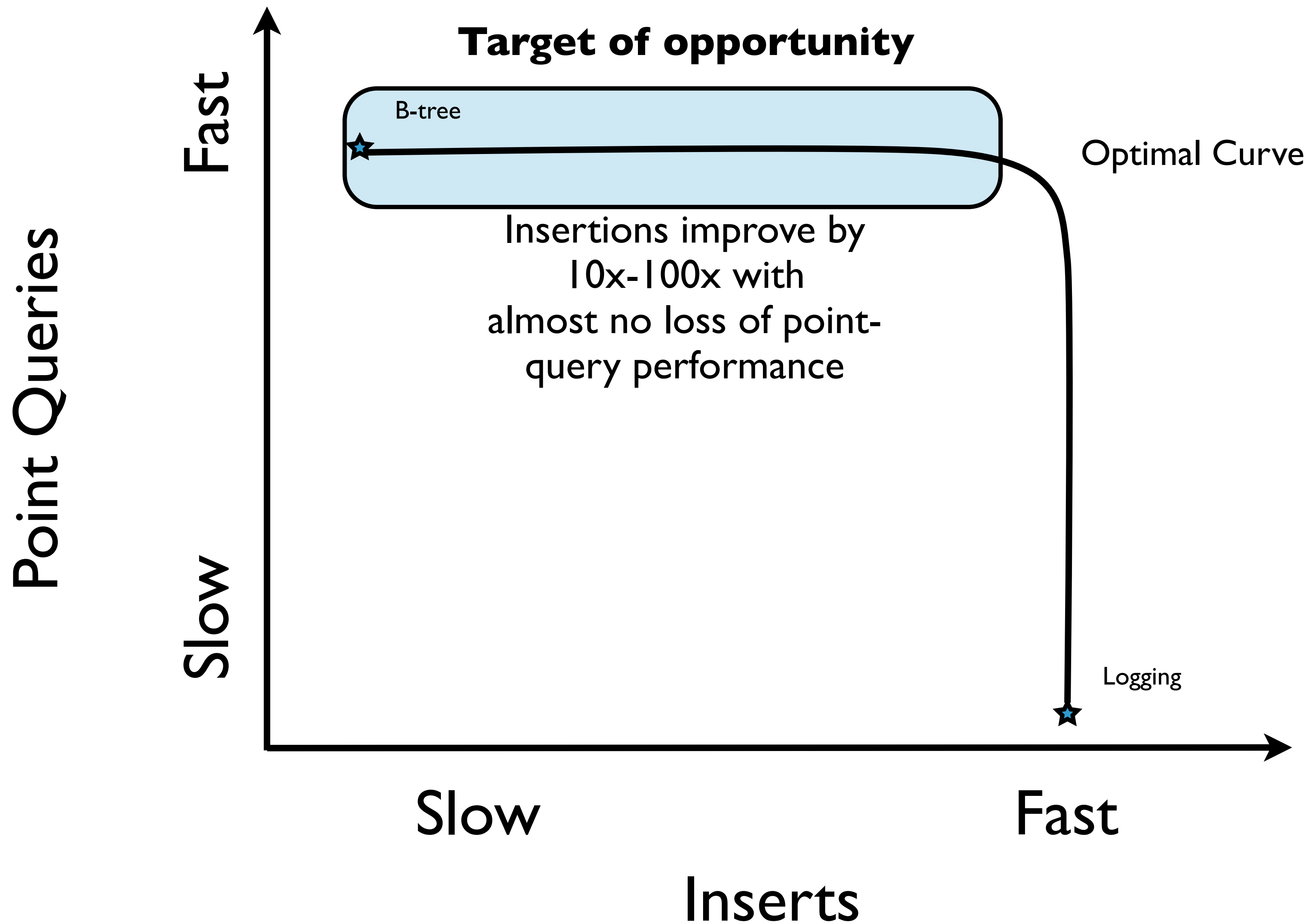


Illustration of Optimal Tradeoff [Brodal, Fagerberg 03]

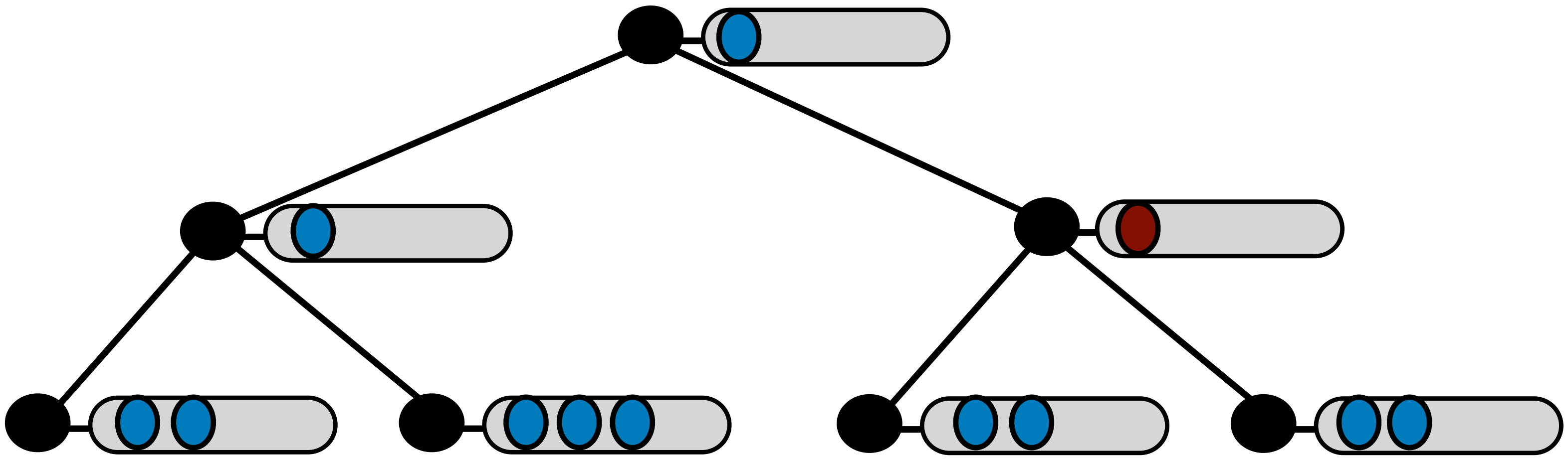


One way to Build Write-Optimized Structures

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



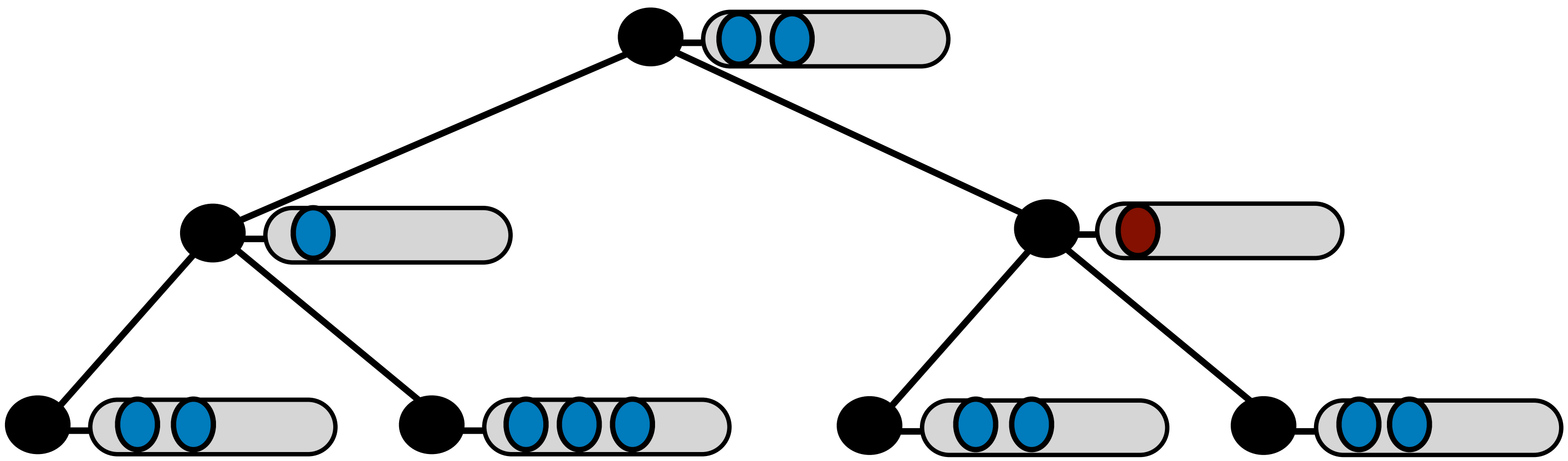
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



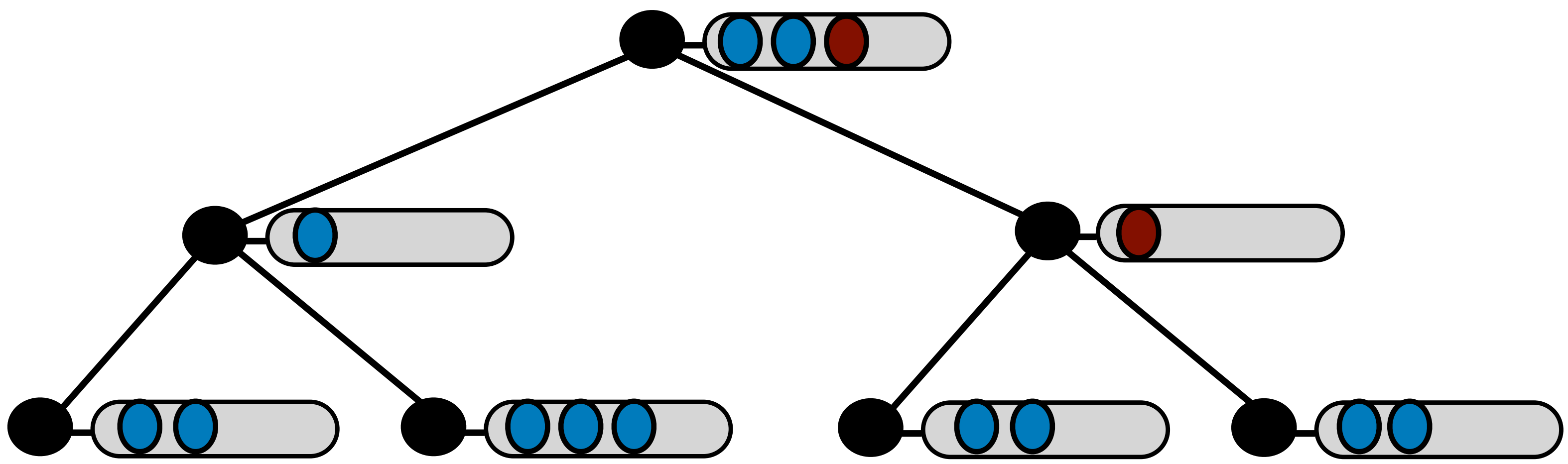
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



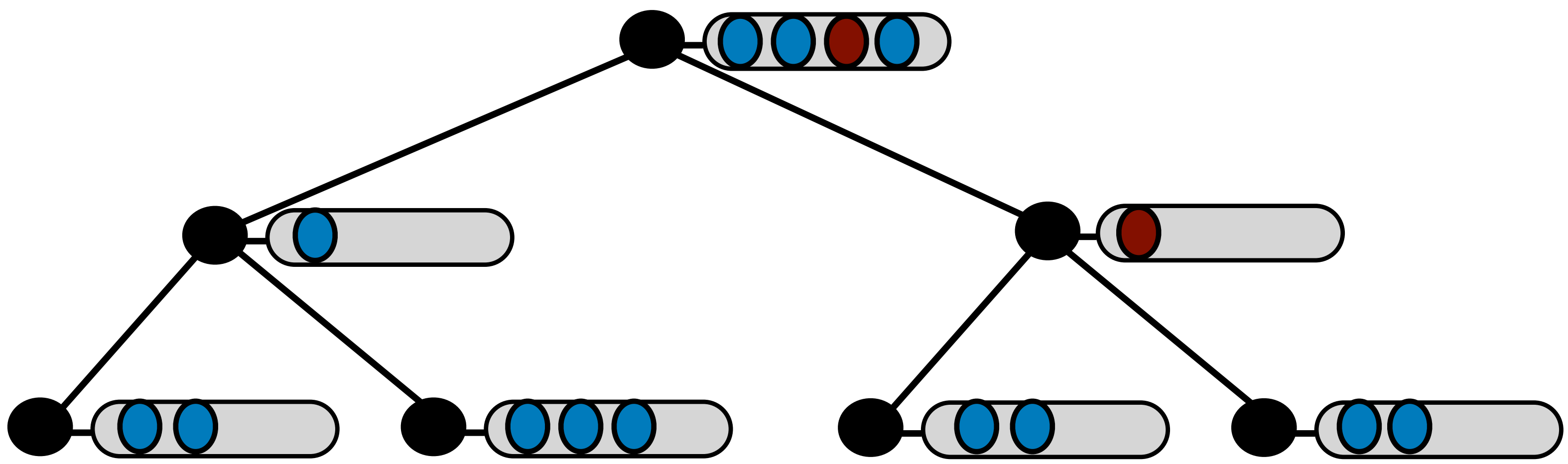
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



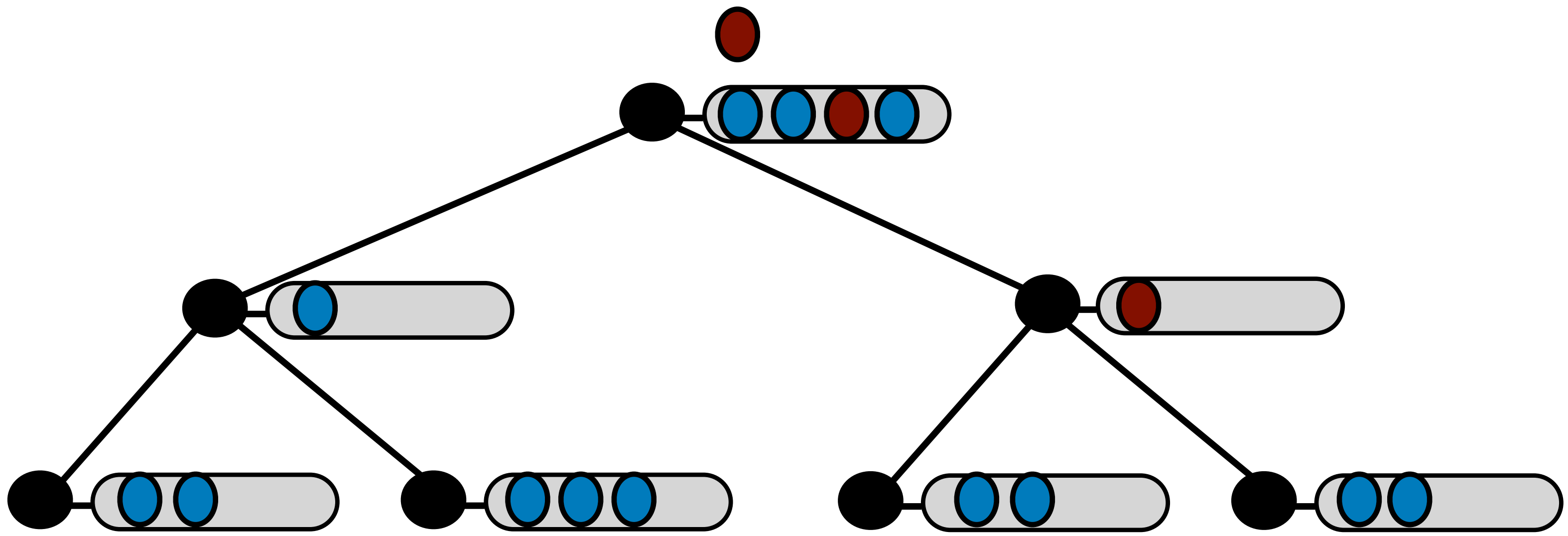
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



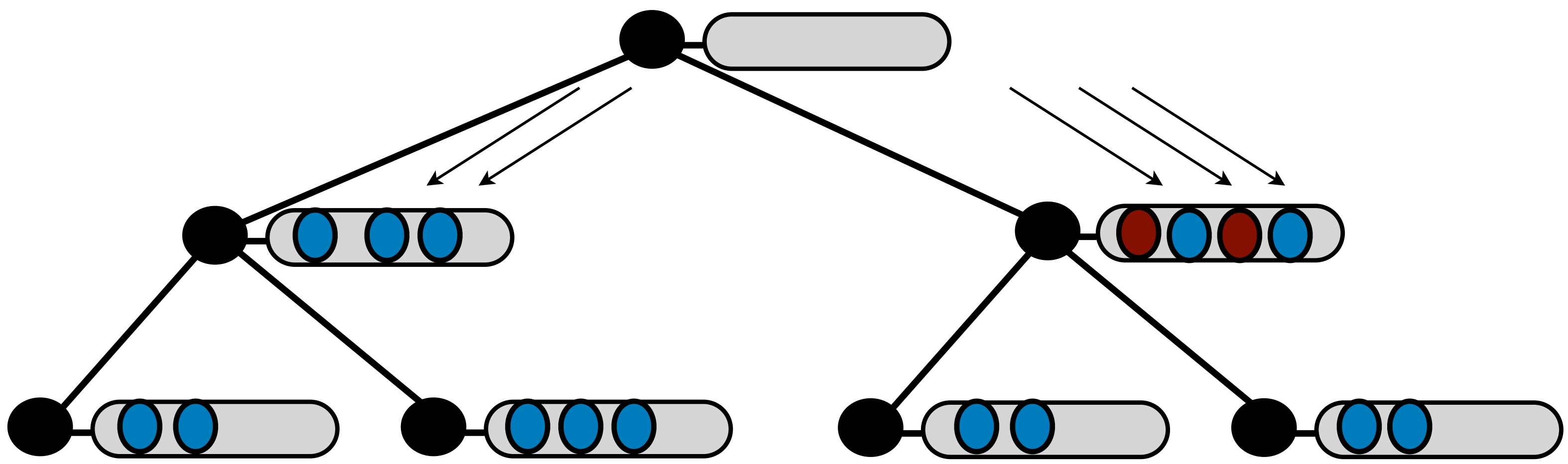
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

A simple write-optimized structure

O(log *N*) queries and O((log *N*)/*B*) inserts:

- A balanced binary tree with buffers of size *B*



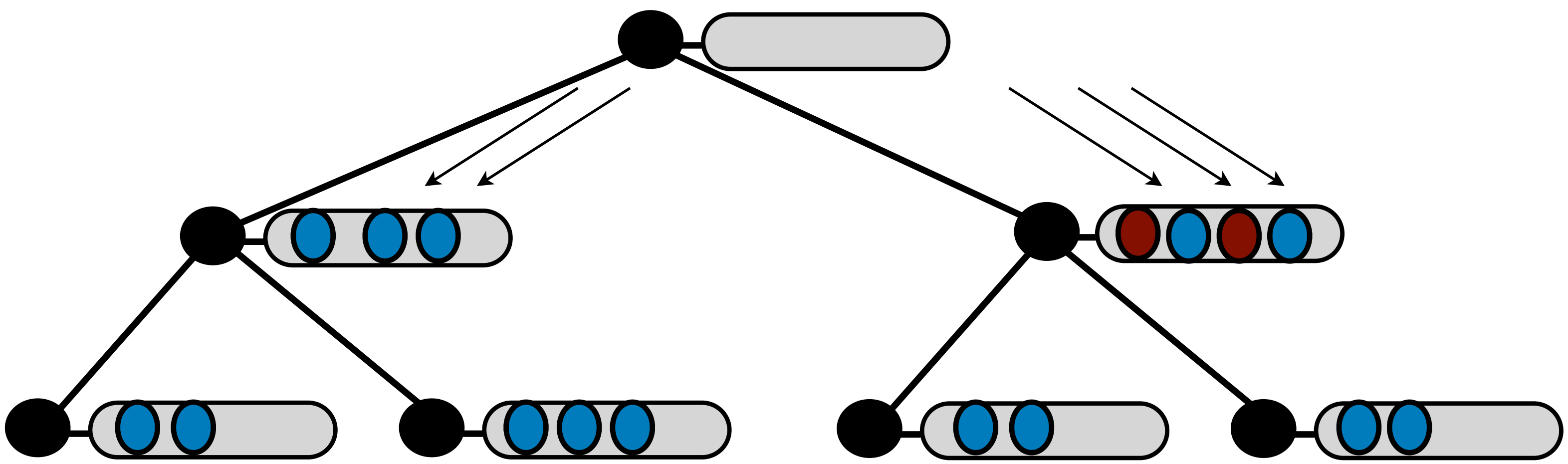
Inserts + deletes:

- Send insert/delete messages down from the root and store them in buffers.
- When a buffer fills up, flush.

Analysis of writes

An insert/delete costs amortized O((log *N*)/B) per insert or delete

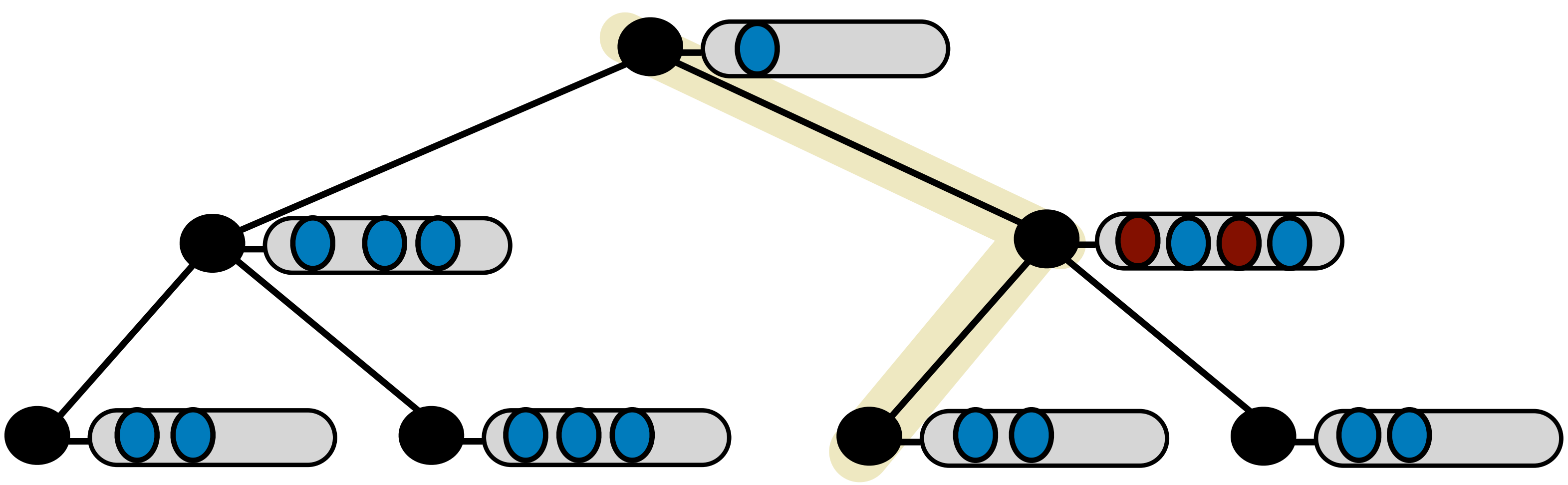
- A buffer flush costs O(1) & sends *B* elements down one level.
- It costs O(1/*B*) to send element down one level of the tree.
- There are O(log *N*) levels in a tree.



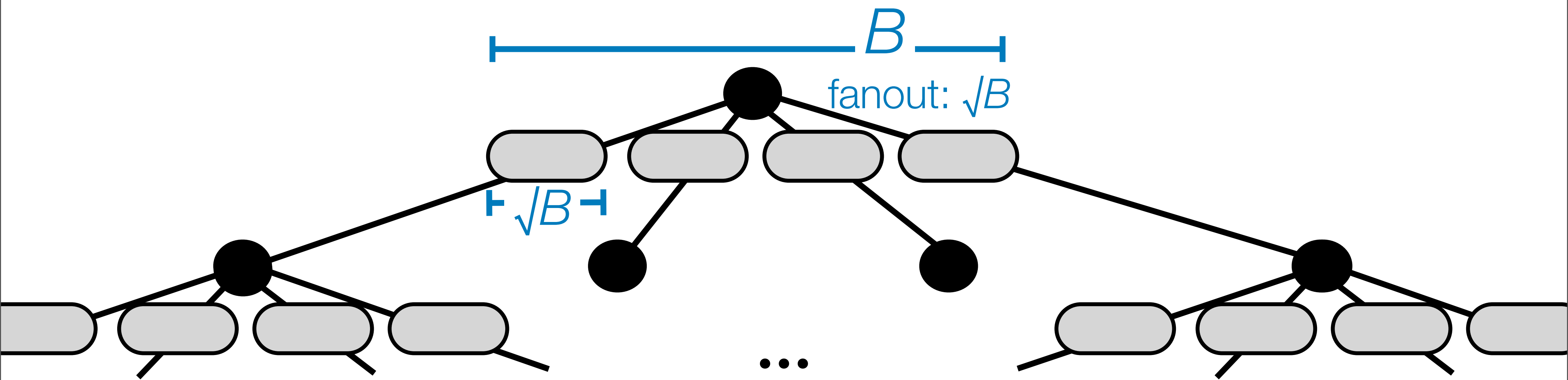
Analysis of point queries

To search:

- examine each buffer along a single root-to-leaf path.
- This costs O(log *N*).



Obtaining optimal point queries + very fast inserts



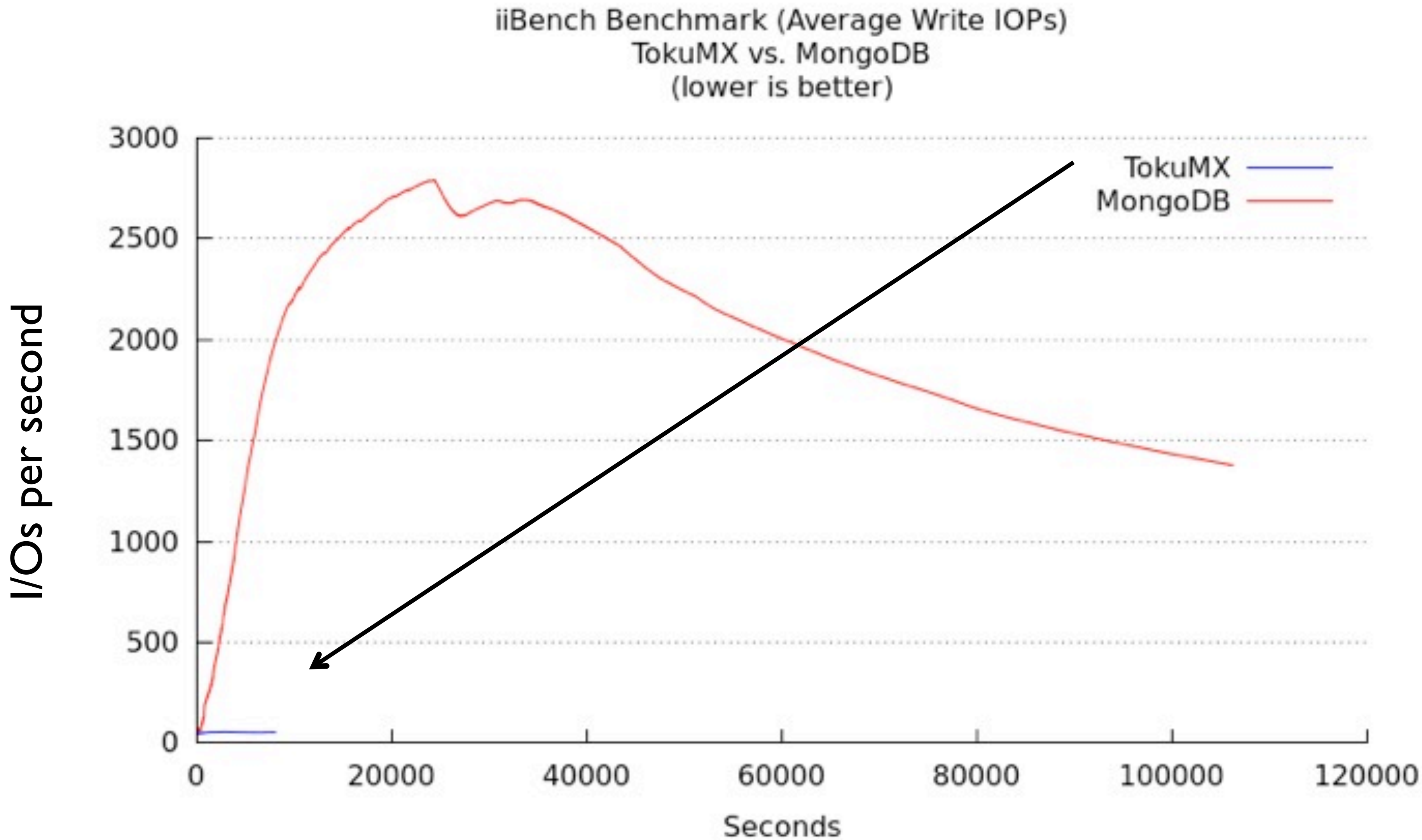
Point queries cost $O(\log_{\sqrt{B}} N) = O(\log_B N)$

- This is the tree height.

Inserts cost $O((\log_B N)/\sqrt{B})$

- Each flush cost O(1) I/Os and flushes $\sqrt{B}$ elements.

TokuMX runs fast because it uses less I/O



100M inserts into a collection with 3 secondary indexes

Applicability

Write-optimized B-Trees win big for disk-resident data.

You can maintain many more indexes on your data, which can speed queries.

Are there write-optimized data structures for geo data or string searches?