Streaming B-Trees for File System Grand Challenges

Michael A. Bender[†] Bradley C. Kuszmaul*
Martin Farach-Colton[‡] Charles E. Leiserson*

† SUNY Stony Brook
‡ Rutgers
* MIT

Grand Challenges

- At last year's HECIWG, some file-system grand challenges were identified.
- Of interest to us, develop a file system that supports:
  - Creating 30,000 microfiles/second.
  - ls -R at near disk bandwidth speed.

Our Results

- We have developed the Streaming B-tree, which is a drop-in replacement for the B-tree at the back end of file systems.
- Streaming B-trees:
  - Make »30,000 insertions per second.
  - Do range queries at ~20-50% of disk bandwidth.
- When SB-trees are deployed in a file system, we expect to solve two grand challenges.

Streaming B-Trees: Fast Updates and Range Queries

Our data structures:

- Cache-oblivious lookahead array (COLA):
  - Over 2 orders of magnitude improvement in inserts.
- Cache-oblivious shuttle tree:
  - Asymptotically optimal point queries with fast updates.
- Both:
  - are cache oblivious (no platform dependent tuning).
  - are fast for range queries.
  - slower than B-trees for point queries.

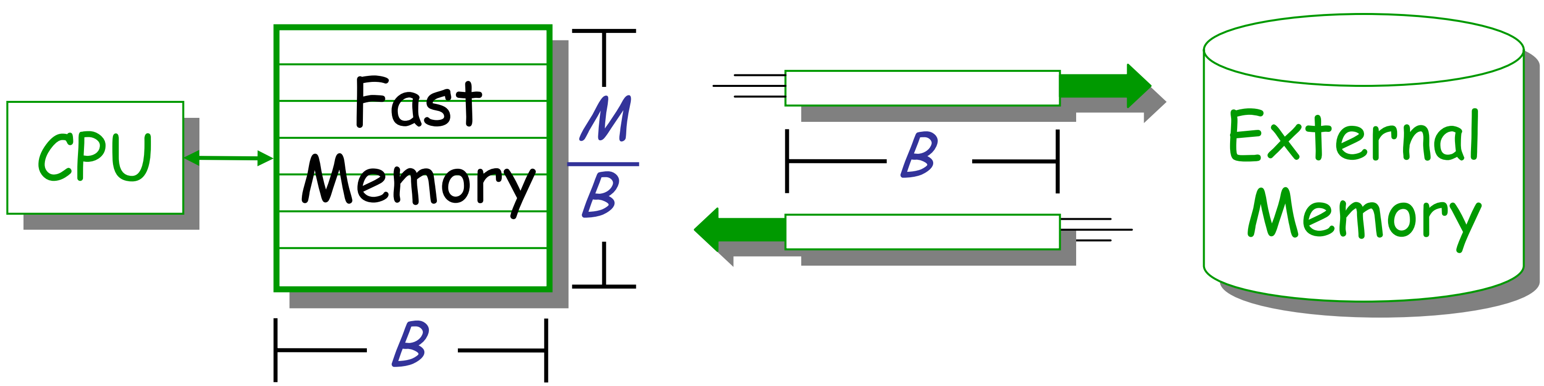
Talk outline

- Analytic introduction to the memory hierarchy.
- Description of data structures.
- Experimental results.
- More data structures.

Disk-Access-Machine (DAM) Model

[Aggarwal, Vitter 88]

- Fast memory of size M
- Data grouped in blocks of size B
- Count # of memory (block) transfers

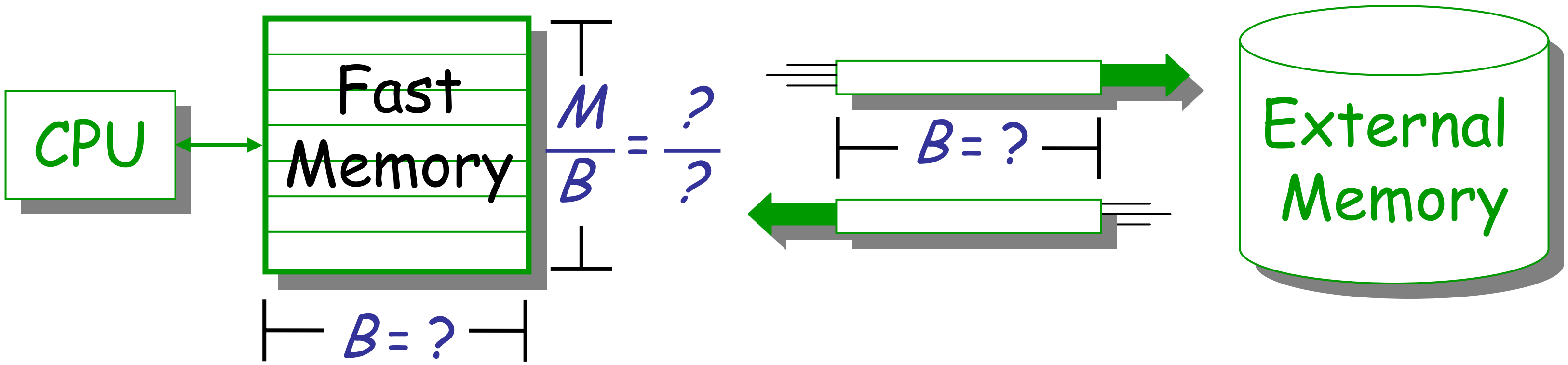


Cache-Oblivious (CO) Model

[Frigo, Leiserson, Prokop, Ramachandran 99]

Like DAM model, except B and M unknown to algo.

- Parameters B and M appear in proofs only.
- Results generalize to multilevel hierarchy.
- Platform independent.

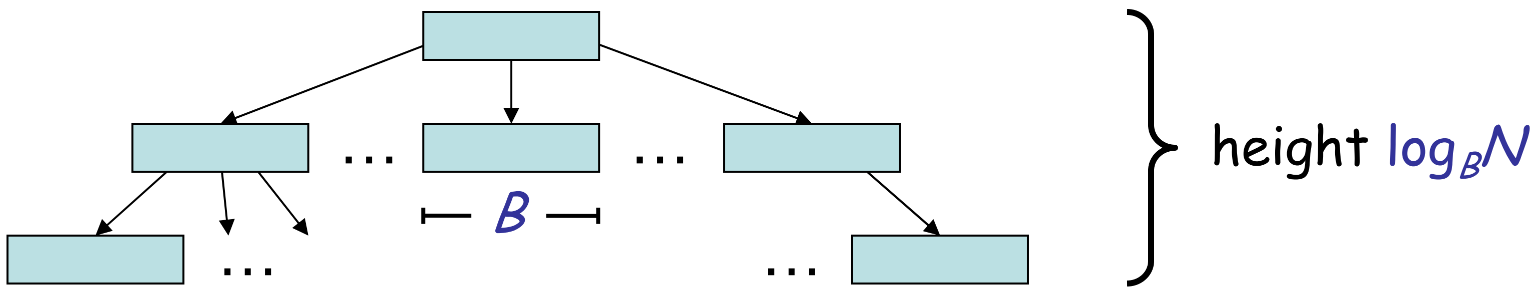


Great for disks, which have no "correct" block size.

Disk-resident CO data structures can offer speedups [Bender, Farach-Colton, Kuszmaul '06]

B-Tree Inserts Are Slow

B-tree [Bayer, McCreight 72]



$O(\log_B N)$ is suboptimal for inserts.

- Can get faster inserts with small loss to searches

Cache-Aware Data Structure	Search	Insert
B-tree [BM72]	$O(\log_B N)$	$O(\log_B N)$
B^{ε}-tree [BF03]	$O((1/\varepsilon)\log_B N)$	$O((1/\varepsilon B^{1-\varepsilon})\log_B N)$*
$B^{1/2}$-tree [BF03]	$O(2\log_B N)$	$O((1/\sqrt{B})\log_B N)$*
BRT [BGVW00]	$O(\log_2 N)$	$O((1/B)\log_2 N)$*

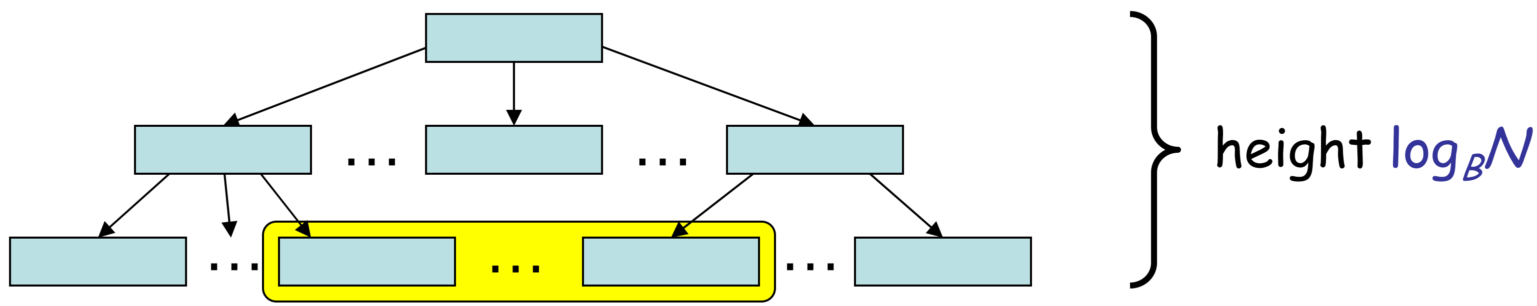
* amortized

B-Tree Range Queries Are Slow

Range query: scan of elements in chosen range.

- *e.g.*, "ls -R"

- B-tree (and B$^{\varepsilon}$-) leaves are scattered across disk.
- Random block transfers are 1-2 orders of magnitude slower than sequential transfers.



CO trees keeps keys (nearly) in order on disk

$\Rightarrow$ fast range queries.

CO Streaming B-Trees: Results

There exists cache-aware search/insert tradeoff.

Cache-Aware DS	Search	Insert
B^{ε}-tree [BF03]	$O((1/\varepsilon)\log_B N)$	$O((1/\varepsilon B^{1-\varepsilon})\log_B N)$*

This work: two points in tradeoff, cache obliviously.

CO Data Structure	Search	Insert
CO B-tree [BDF-C00,BDIW04,BFJ02]	$O(\log_B N)$	$O(\log_B N + (\log^2 N)/B)$*
CO Lookahead Array (COLA) [this talk]	$O(\log_2 N)$	$O((1/B)\log_2 N)$*
CO Shuttle Tree [this talk]	$O(\log_B N)$	$O((1/B^{\Omega(1/(\log\log B)^2)})\log_B N + (\log^2 N)/B)$*

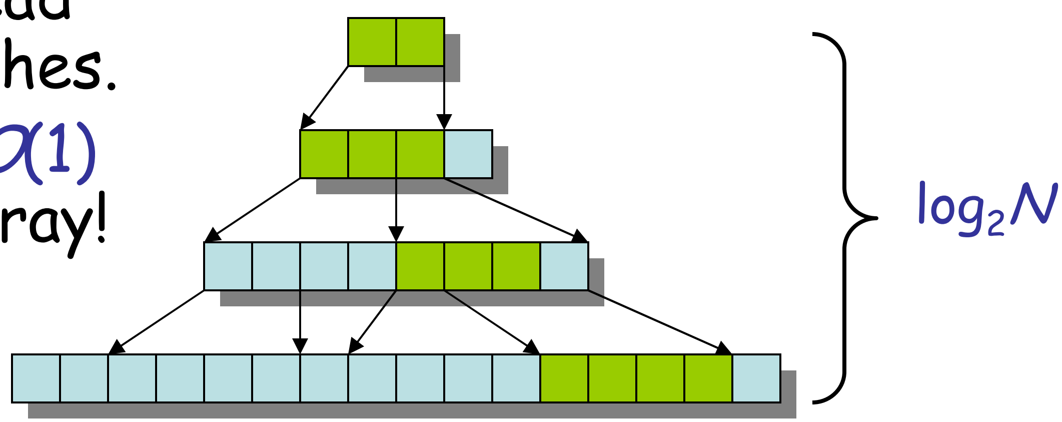
* amortized

Talk outline

- Analytic introduction to the memory hierarchy.
- Description of COLA.
- Experimental results.
- More data structures.

Cache-Oblivious Lookahead Array

- Search: $O(\log_2 N)$ block transfers.
- Insert: $O((1/B)\log_2 N)$ amortized and $O(\log_2 N)$ worst-case block transfers.
- Consists of $\lceil \log N \rceil$ arrays where the ith array stores 2^i elements.
  - Each array is sorted and full (2^i elements) or "empty" (0 elements).
  - Redundant "lookahead pointers" aid searches.
  - Search scans only $O(1)$ elements in each array!



Talk outline

- Analytic introduction to the memory hierarchy.
- Description of data structures.
- Experimental results.
- More data structures.

COLA vs. B-Tree*: Experimental Results

Random inserts are 1300 times faster in COLA

- B-tree: 14 days to insert (1.5×mem)-size dataset.
- COLA: 14 minutes to insert the same dataset.

COLA inserts are consistently fast

- Random only 10% slower than presorted inserts.
- Presorted inserts are 3.1× slower than B-tree, but COLA does not (yet) optimize for this case.

Tradeoff:

- Point searches are 3.5× slower than B-tree.

* Our B-tree's performance is comparable to Berkeley DB [Bender, Farach-Colton, Kuszmaul 06].

COLA Test Specs

Machine:

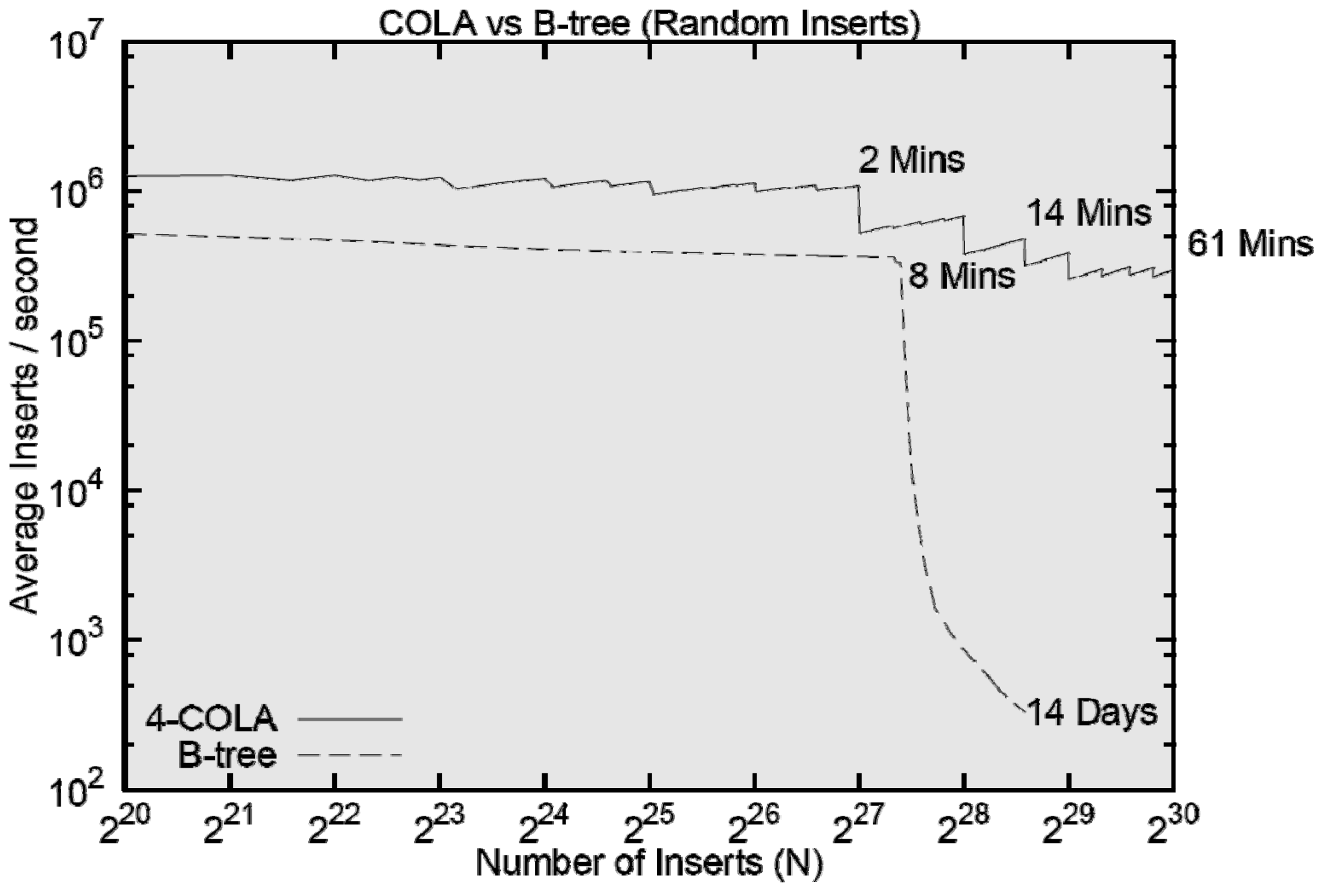
- Dual Xeon 3.2GHz with 2MiB of L2 Cache.
- 4GiB RAM.
- Two 250GB Maxtor 7L250S0 SATA drives.
  - Software RAID-0 with 64KiB stripe width.
- Linux 2.6.12-10-amd64-xeon in 64-bit mode.

Input:

- 64-bit keys and values.

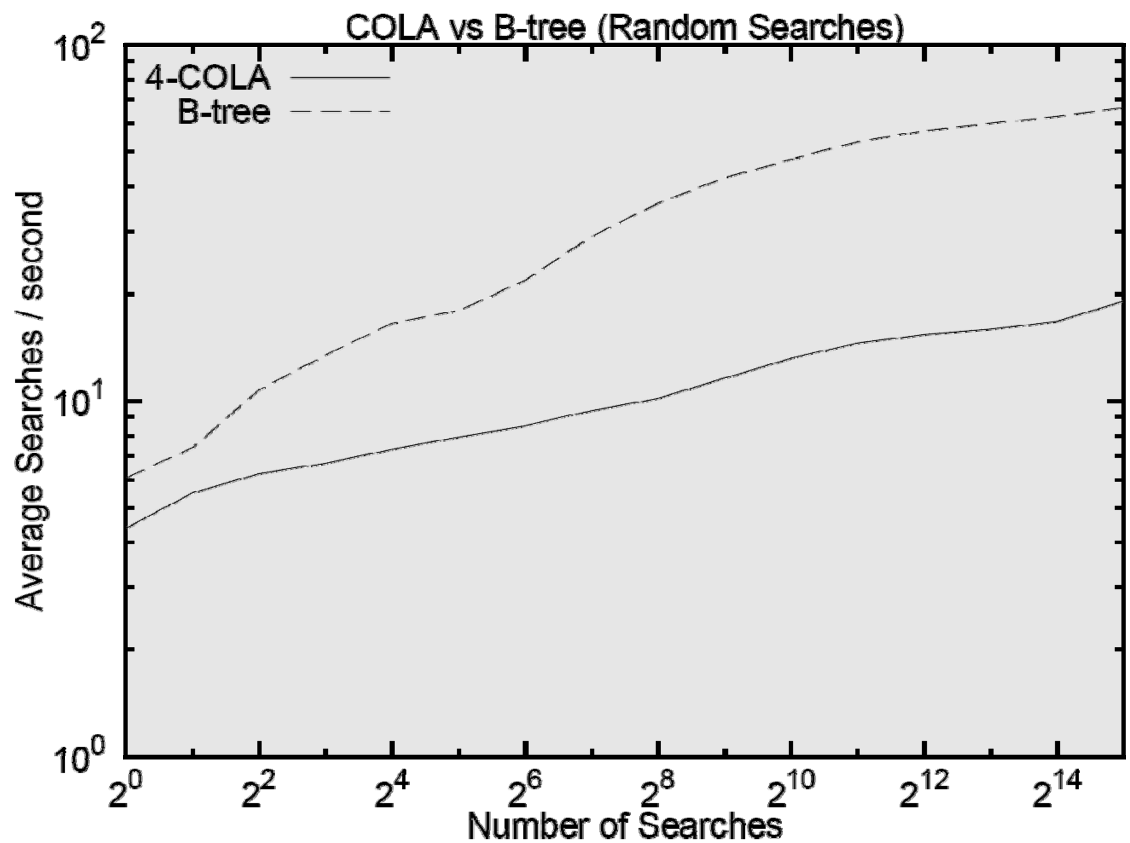
COLA vs. B-Tree: Random Inserts

- The COLA is 1300 times faster than the B-tree
  - Expect the B-tree to level off at ~3 orders of magnitude slower than the COLA.



COLA vs. B-Tree: Searches

- The COLA is 3.5 times slower for searches
  - $N = 2^{30} - 1$
  - Keys were inserted in order for the B-tree



Comparison

- B-tree gives ~100/insertions/second/disk.
- COLA gives ~150,000 insertions/second/disk.
  - But point queries are 3.5x slower than B-trees.
- We have a new implementation that:
  - handles 20K-30K.
  - Point queries are 40% slower than B-trees.
  - handles variable-length keys.

Talk outline

- Analytic introduction to the memory hierarchy.
- Description of data structures.
- Experimental results.
- More data structures.

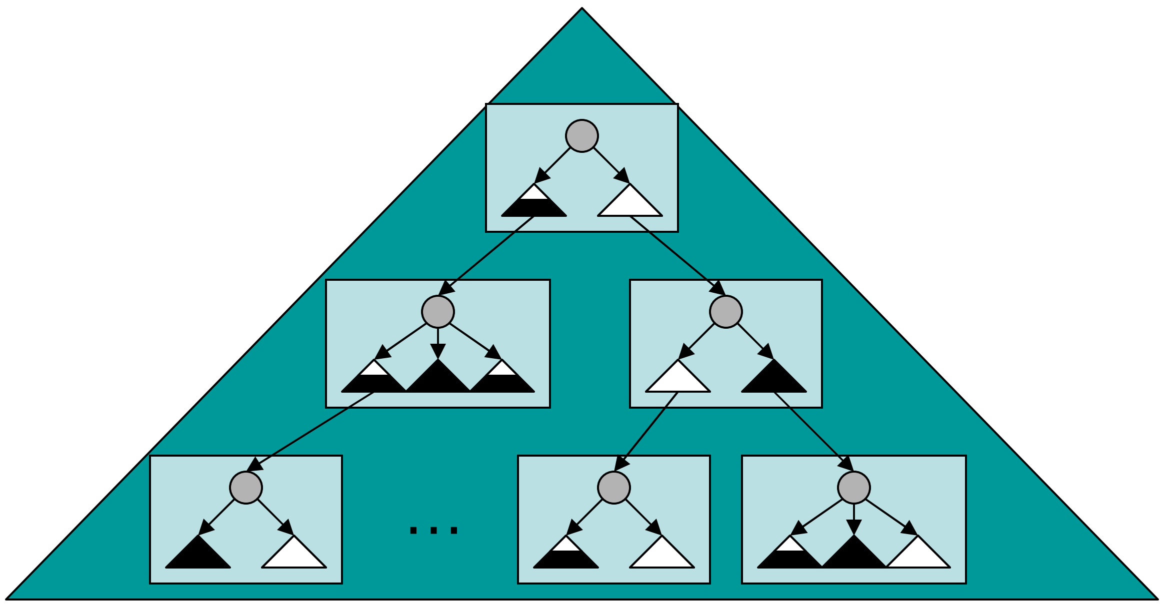
Shuttle-Tree Overview

- Cache oblivious.
- Fast inserts ($O((1/B^{\Omega(1/(\log\log B)^2)})\log_B N + (\log^2 N)/B)$)
  - using buffers that are (recursively) shuttle trees.
- Searches asymptotically match B-trees at $O(\log_B N)$. (COLA searches are only $O(\log_2 N)$.)
  - using recursive cache-oblivious layout.
- Fast range queries.
  - Layout keeps elements (nearly) in order.
- Uses PMA [Bender, Demaine, Farach-Colton 00] to keep layout dynamically.

Shuttle Tree Uses Buffers For Fast Inserts

The ***Shuttle Tree*** is a CO tree with degree- $\Theta(1)$ nodes, where each node has buffers.

- Buffers are also shuttle trees.



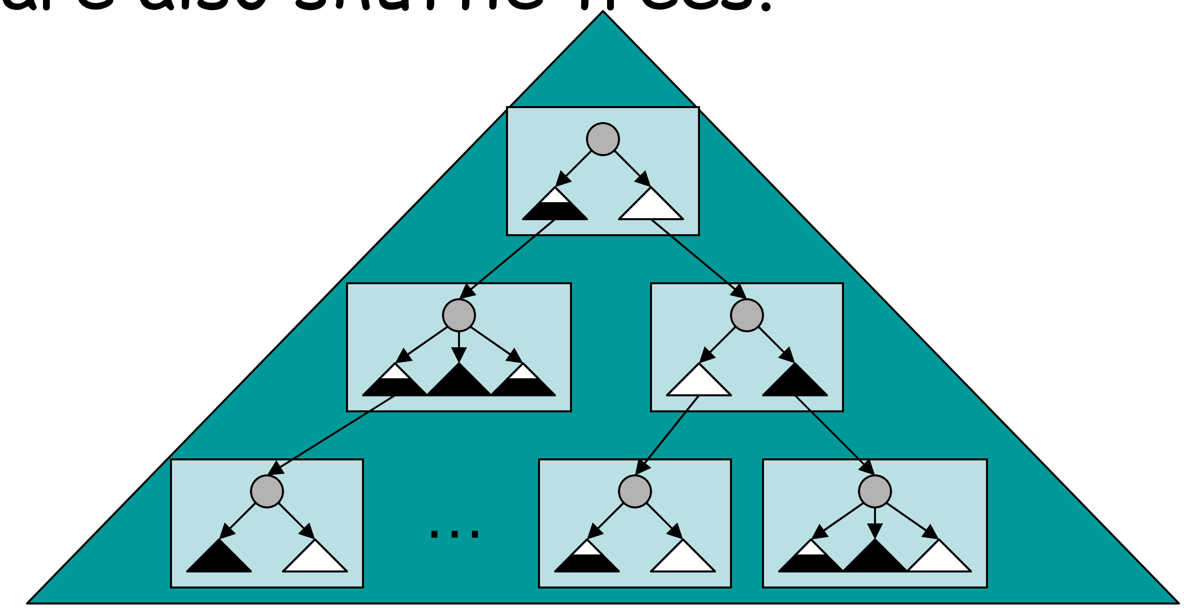
Search:

- Walk down tree, looking in buffers.
- Cost is O((buffer searches) + (root-to-leaf path)).

Shuttle Tree Uses Buffers For Fast Inserts

The ***Shuttle Tree*** is a CO tree with degree-$\Theta(1)$ nodes, where each node has buffers.

- Buffers are also shuttle trees.



Insert:

- Fill buffer before moving down tree.
- Push buffer size keys down at a time.
- Amortize moving down tree against buffer size.

Publications

- Cache-Oblivious Streaming B-Trees (SPAA 07)
  - Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, Jelani Nelson
- Cache-Oblivious String B-trees (PODS 06)
  - Michael A. Bender, Martin Farach-Colton, Bradley C. Kuszmaul

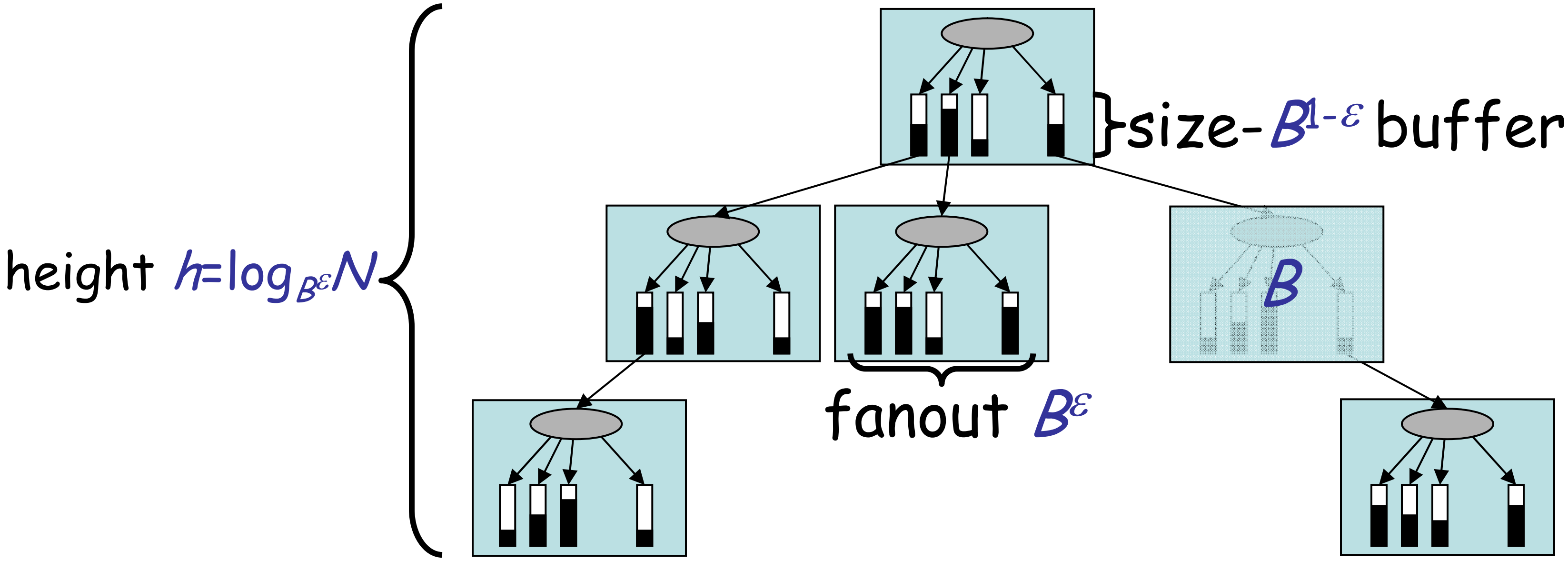
What next? Tokutek

- We are commercializing this technology through a startup called Tokutek.
- We are looking for insert-intensive applications.
- We are looking for engineers.

Buffer for Fast Inserts:

The Cache-Aware B^ε-Tree [Brodal, Fagerberg 03]

- Nodes have fanout B^ε and total buffer size B.



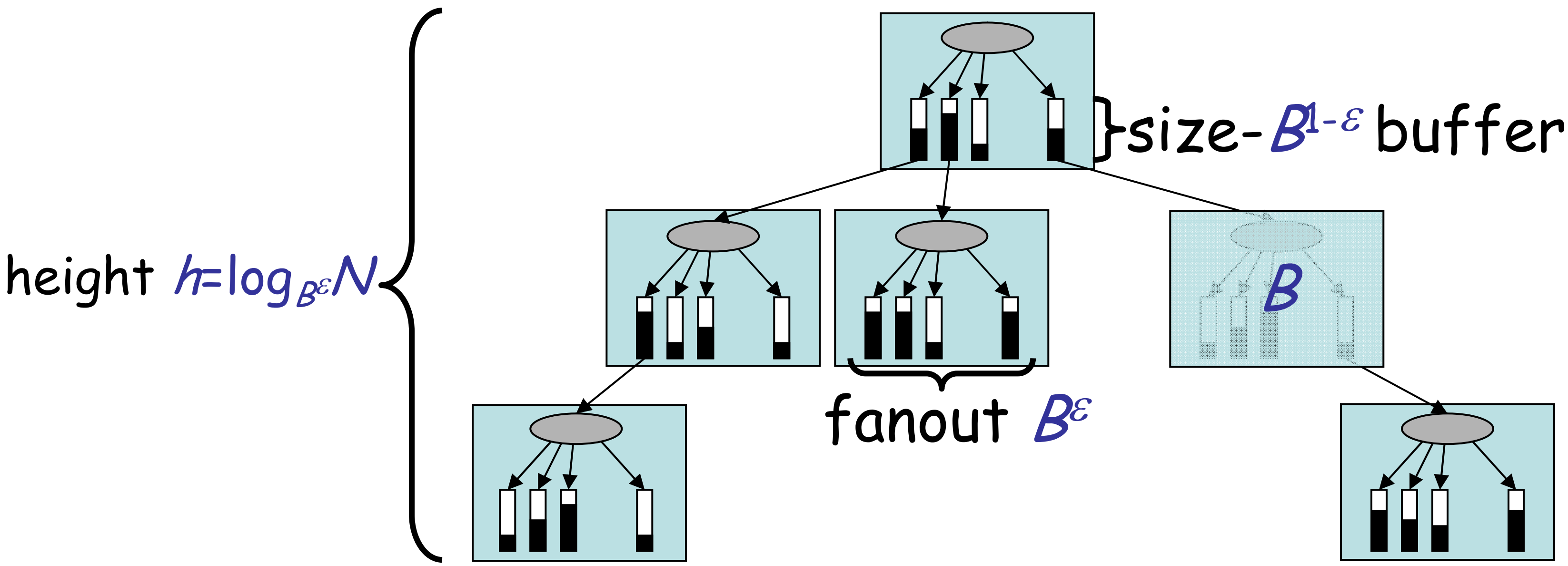
Search:

- Walk down tree, looking in buffers.
- Cost is $O(\text{(buffer search)}h) = O((1/\varepsilon)\log_B N)$

Buffer for Fast Inserts:

The Cache-Aware B^{ε}-Tree [Brodal, Fagerberg 03]

- Nodes have fanout B^{ε} and total buffer size B.



Inserts:

- Fill buffer before moving down tree.
- Push buffer size = $B^{1-\varepsilon}$ keys down at a time.
- Cost is $O(h/(\text{buffer size})) = O((1/\varepsilon B^{1-\varepsilon})\log_B N)$