File System Grand Challenges

- File System Grand Challenges from the HEC-IWG* File Systems IO workshop in '06:
  - Develop a file system that can create 30,000 microfiles/second.
  - Develop a file system that can perform "ls –R" (recursive directory listing) at near disk bandwidth.

- The right data structure (this talk) solves these grand challenges on a laptop.

* Mysteriously, this acronym stands for "High End Computing Revitalization Task Force (HEC-RTF), Inter Agency Working Group (HEC-IWG) File Systems and I/O Research Guidance Workshop"

Cache-Oblivious Streaming B-Trees

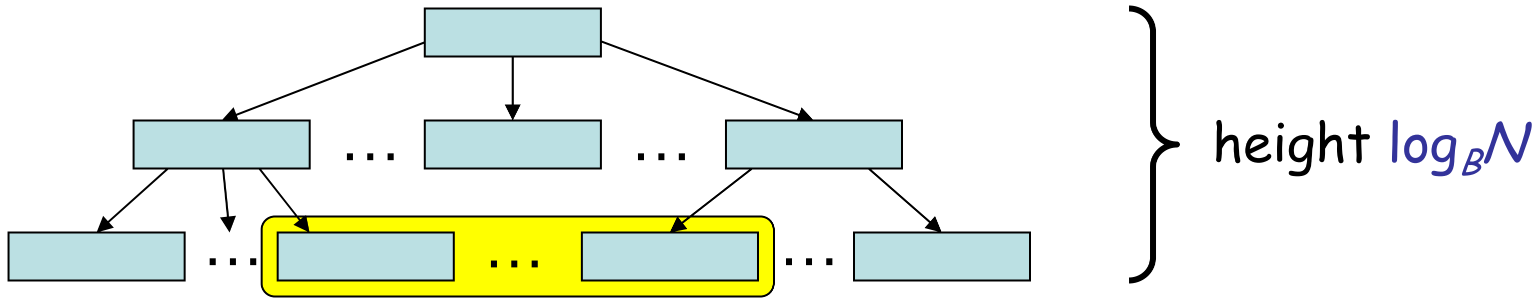
Michael A. Bender[†◊] Martin Farach-Colton[‡◊]
Jeremy T. Fineman[*] Yonatan R. Fogel[†◊]
Bradley C. Kuszmaul[*◊] Jelani Nelson[*]

† Stony Brook University
◊ Tokutek™
‡ Rutgers
* MIT

This Talk

- The **cache-oblivious streaming B-tree** is a drop-in replacement for the B-tree at the back end of file systems and databases.
- COSB-trees:
  - Make »30,000 insertions per second per disk.
  - Do range queries at ~20-50% of disk bandwidth.
- When COSB-trees are deployed in a file system, they solve these two grand challenges.

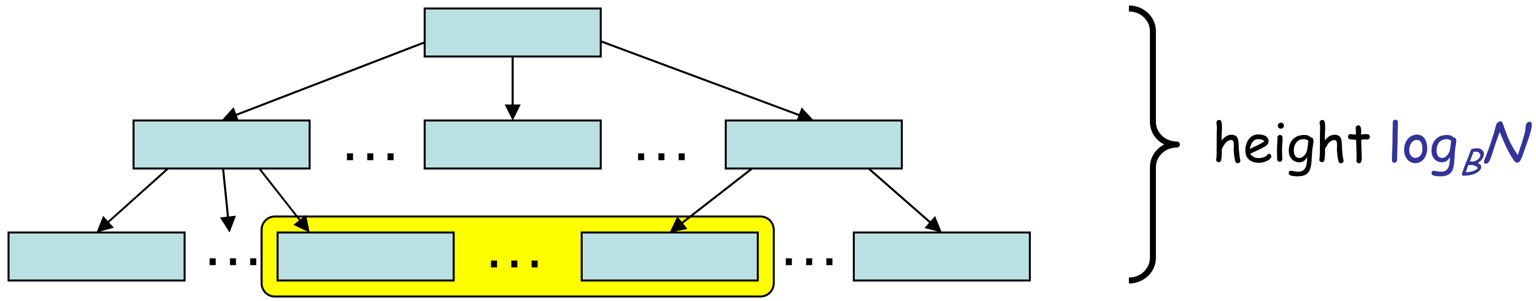
B-trees: Used by Databases & File Systems



- B-tree inserts are slow
  - Inserts require $O(\log_B N)$ memory transfers.

- B-tree range queries are slow
  (Range query: scan of elements in chosen range, e.g., “ls -R”)
  - B-tree leaves are scattered across disk.
  - Random block transfers are 1-2 orders of magnitude slower than sequential transfers.

B-trees: Used by Databases & File Systems

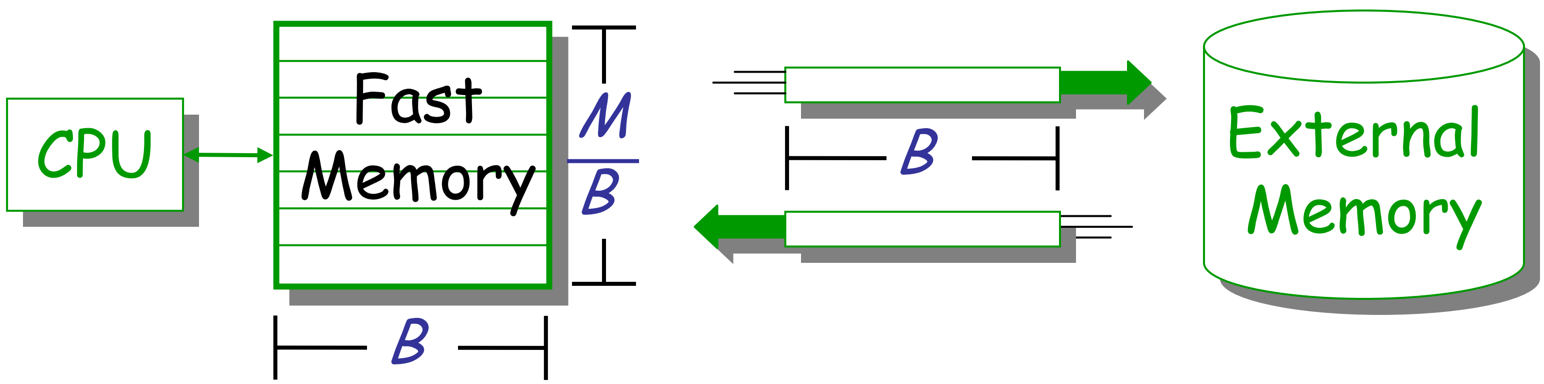


- B-tree inserts are slow
  - Inserts require $O(\log_B N)$ memory transfers.
  - But $\log_B N$ is a lower bound only for searches, not inserts.
- B-tree range queries are slow

  (Range query: scan of elements in chosen range, e.g., "ls -R")
  - B-tree leaves are scattered across disk.
  - Random block transfers are 1-2 orders of magnitude slower than sequential transfers.
  - But data structures such as the PMA keep elements ordered

Cache-Oblivious [Frigo, Leiserson, Prokop, Ramachandran 99] and DAM Models [Aggarwal & Vitter 88]

- **Disk-Access Model (DAM)** : Two-levels of memory.
- Two parameters: block-size B and memory-size M.
- Count number of memory transfers.



- **Cache-Oblivious Model (CO):** Similar to DAM, but B and M are unknown to the algorithm and coder.
- Great for disks, which have no "correct" block size.
  CO data structures can offer speedups [Bender, Farach-Colton, Kuszmaul '06].

Cache-Oblivious Streaming B-trees:

- **Cache-Oblivious Streaming B-trees:**
  - trade off insert and search performance,
  - perform efficient range queries,
  - are cache-oblivious.

- We give two such structures:
  - **Cache-oblivious lookahead array (COLA):**
    - Over 2 orders of magnitude improvement in inserts.
  - **Cache-oblivious shuttle tree:**
    - Asymptotically optimal searches with faster inserts.

Key Search/Insert Tradeoff for DAM (not CO) Data Structures

DAM DS	Search	Insert
B^{ε}-tree [Brodal, Fagerberg 03]	$O((1/\varepsilon)\log_B N)$	$O((1/\varepsilon B^{1-\varepsilon})\log_B N)$*

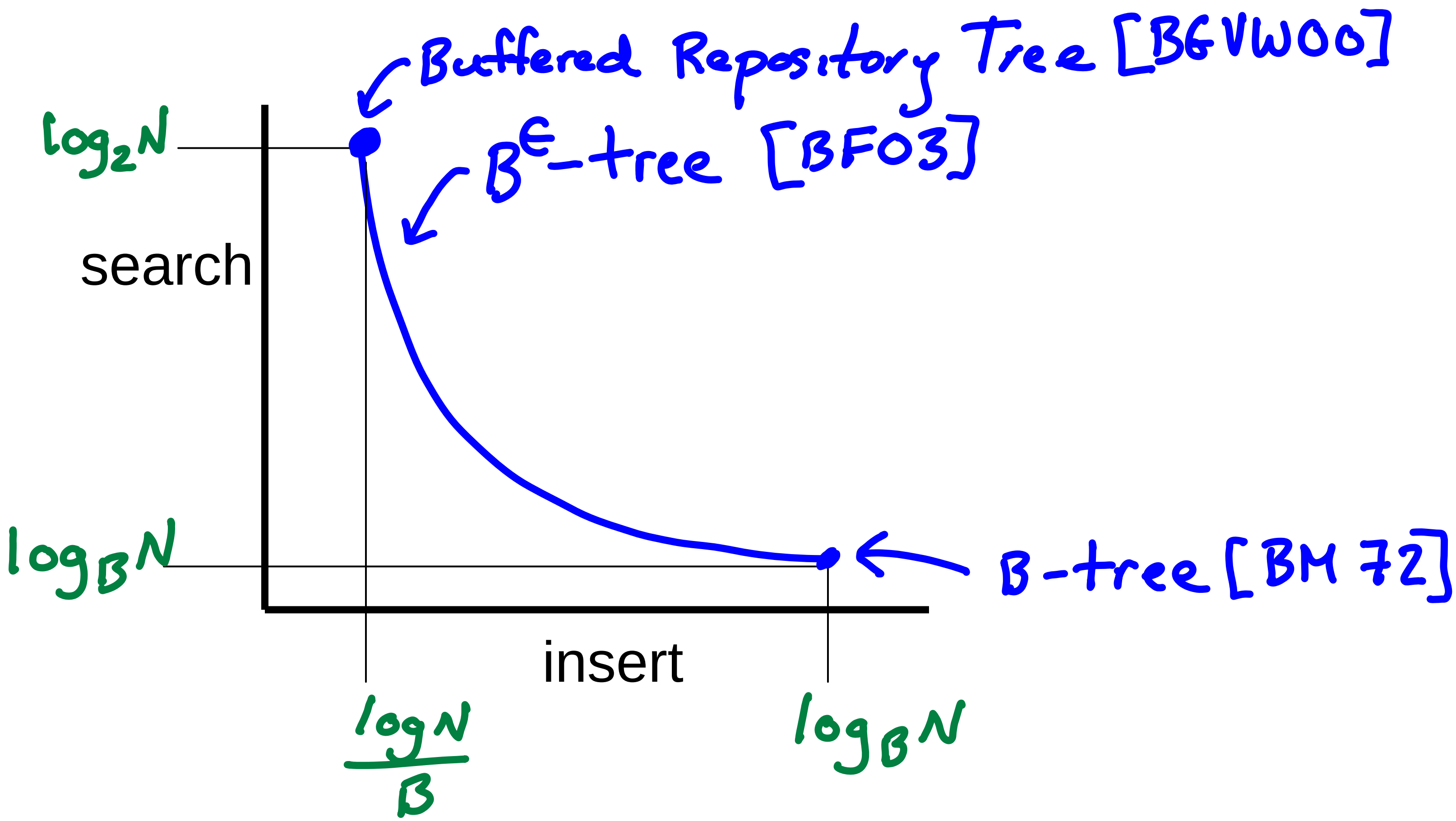
ε ranges from 1 down to 0.

(When ε hits 0, low-order terms that I didn't write kick in.)

DAM DS	Search	Insert
B-tree [Bayer, McCreight 72]	$O(\log_B N)$	$O(\log_B N)$
$B^{1/2}$-tree [Brodal, Fagerberg 03]	$O(2\log_B N)$	$O((1/\sqrt{B})\log_B N)$*
BRT [Buchsbaum, Goldwasser, Venkatasubramanian, Westbrook 00]	$O(\log_2 N)$	$O((1/B)\log_2 N)$*

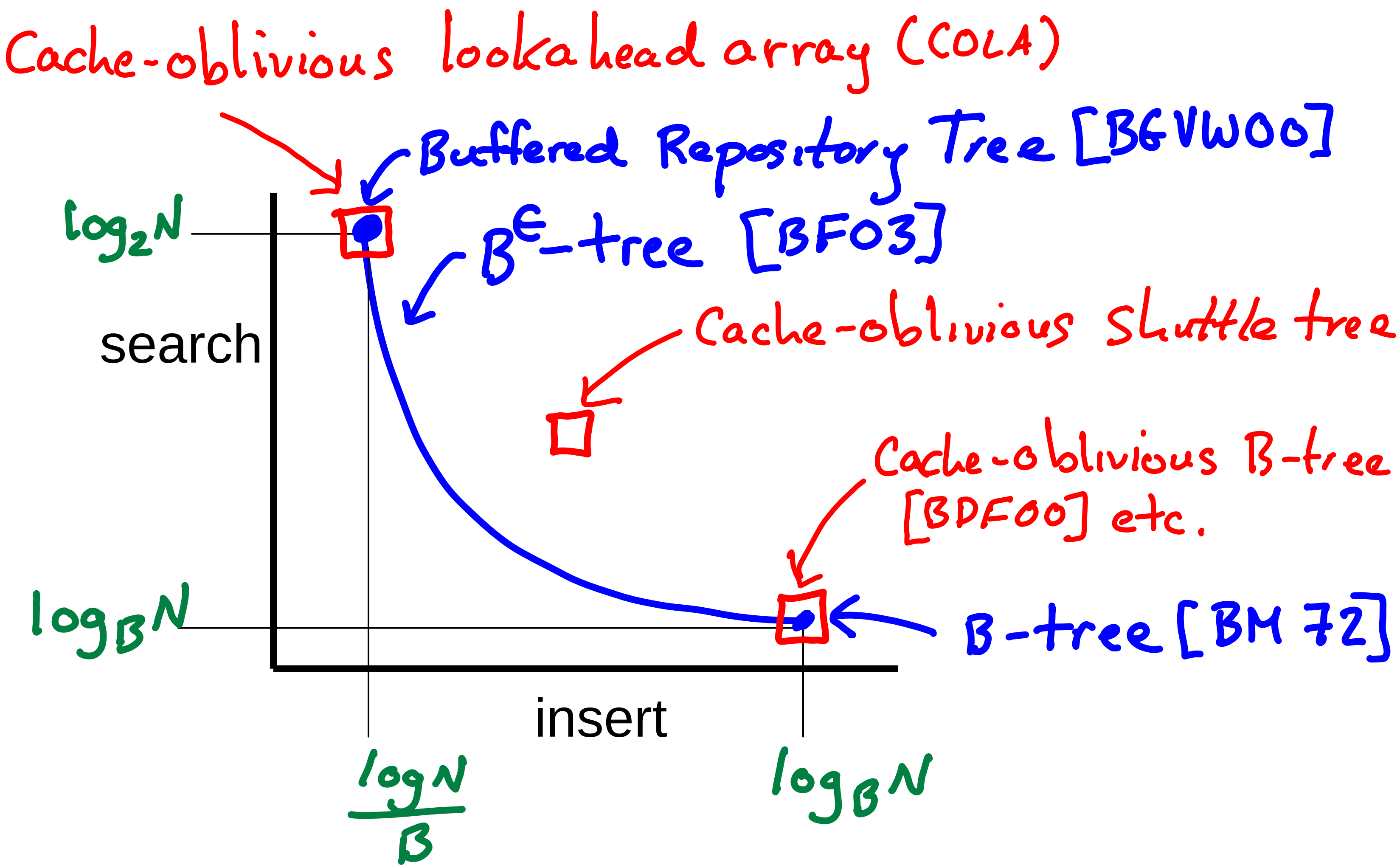
* amortized

Picture Showing the DAM (not CO) Search/Insert Tradeoff



Picture Showing Our Results

We explore several points in the tradeoff for cache-oblivious data structures.



CO Streaming B-Trees: Results

This work: two points in tradeoff, cache obliviously.

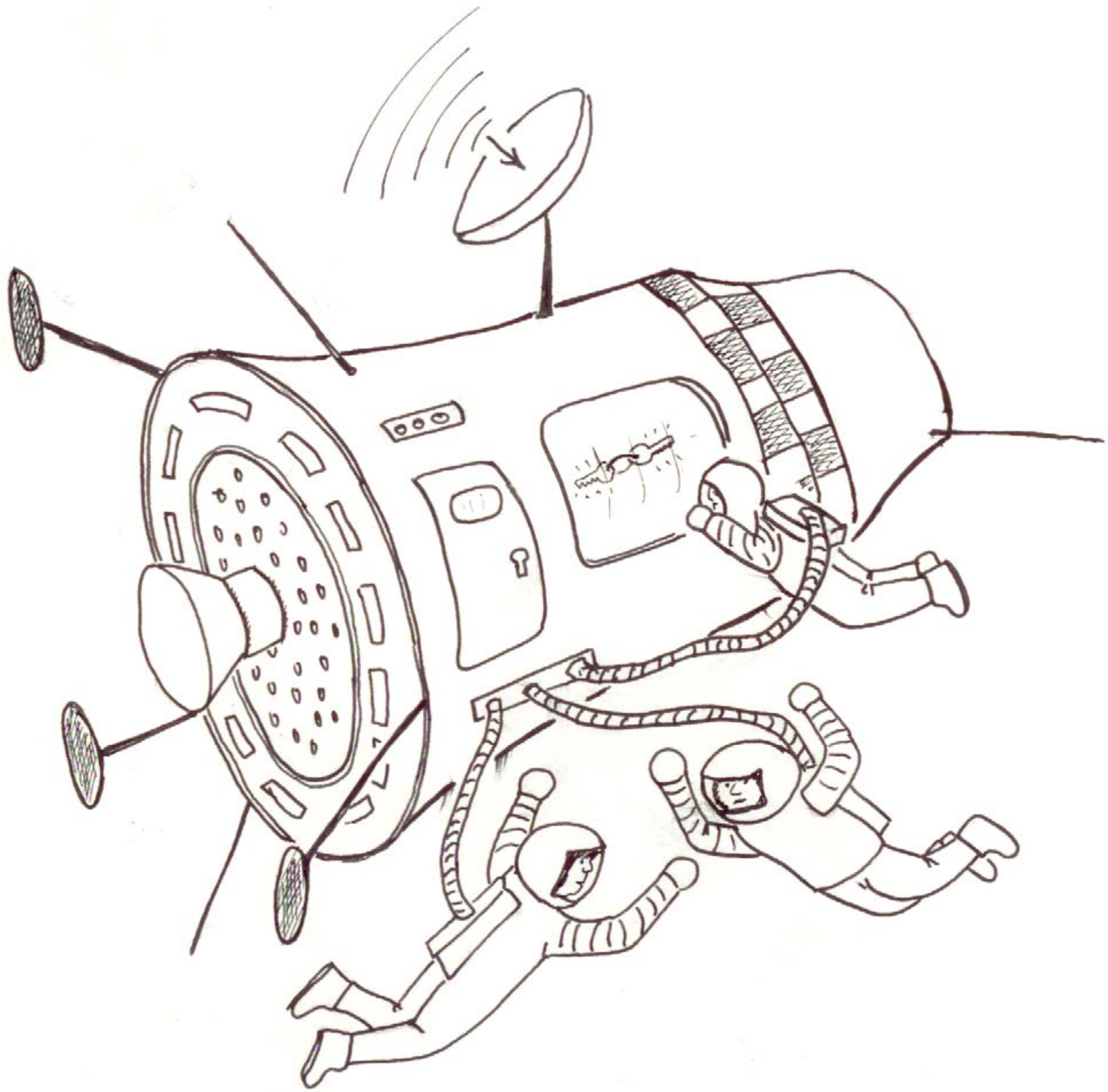
CO Data Structure	Search	Insert
CO B-tree [BDF00,BDIW04,BFJ02]	$O(\log_B N)$	$O(\log_B N + (\log^2 N)/B)$*
CO Lookahead Array (COLA) [this talk]	$O(\log_2 N)$	$O((1/B)\log_2 N)$*
CO Shuttle Tree [this talk]	$O(\log_B N)$	$O((1/B^{\Omega(1/(\log\log B)^2)})\log_B N + (\log^2 N)/B)$*

Here's the DAM search/insert tradeoff again:

Cache-Aware DS	Search	Insert
B^ε-tree [Brodal, Fagerberg 03]	$O((1/\varepsilon)\log_B N)$	$O((1/\varepsilon B^{1-\varepsilon})\log_B N)$*

* amortized

Picture Showing a Key Search



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

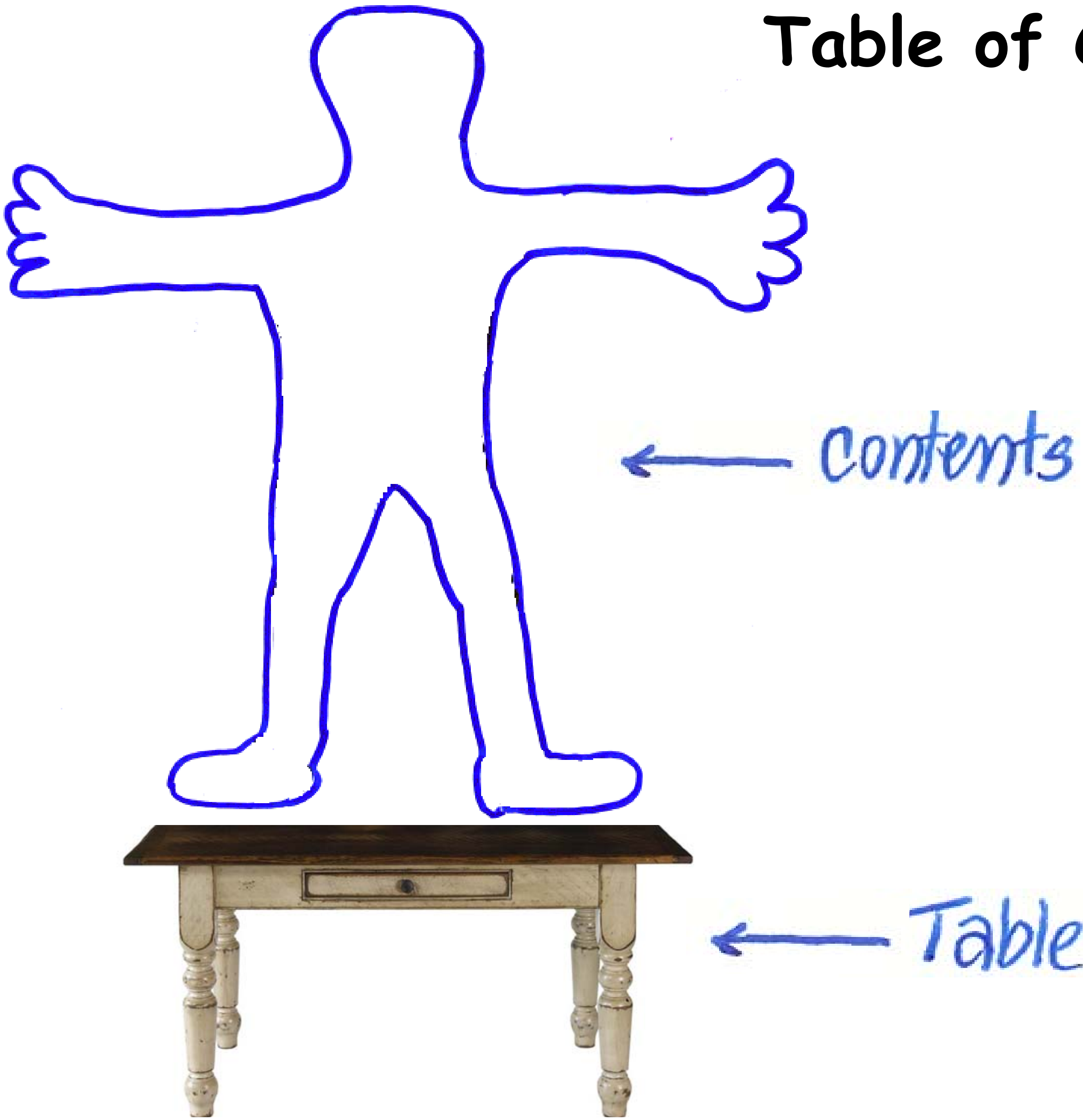
- Random only 10% slower than presorted inserts.
- Presorted inserts are 3.1× slower than B-tree, but COLA does not (yet) optimize for this case.

Tradeoff:

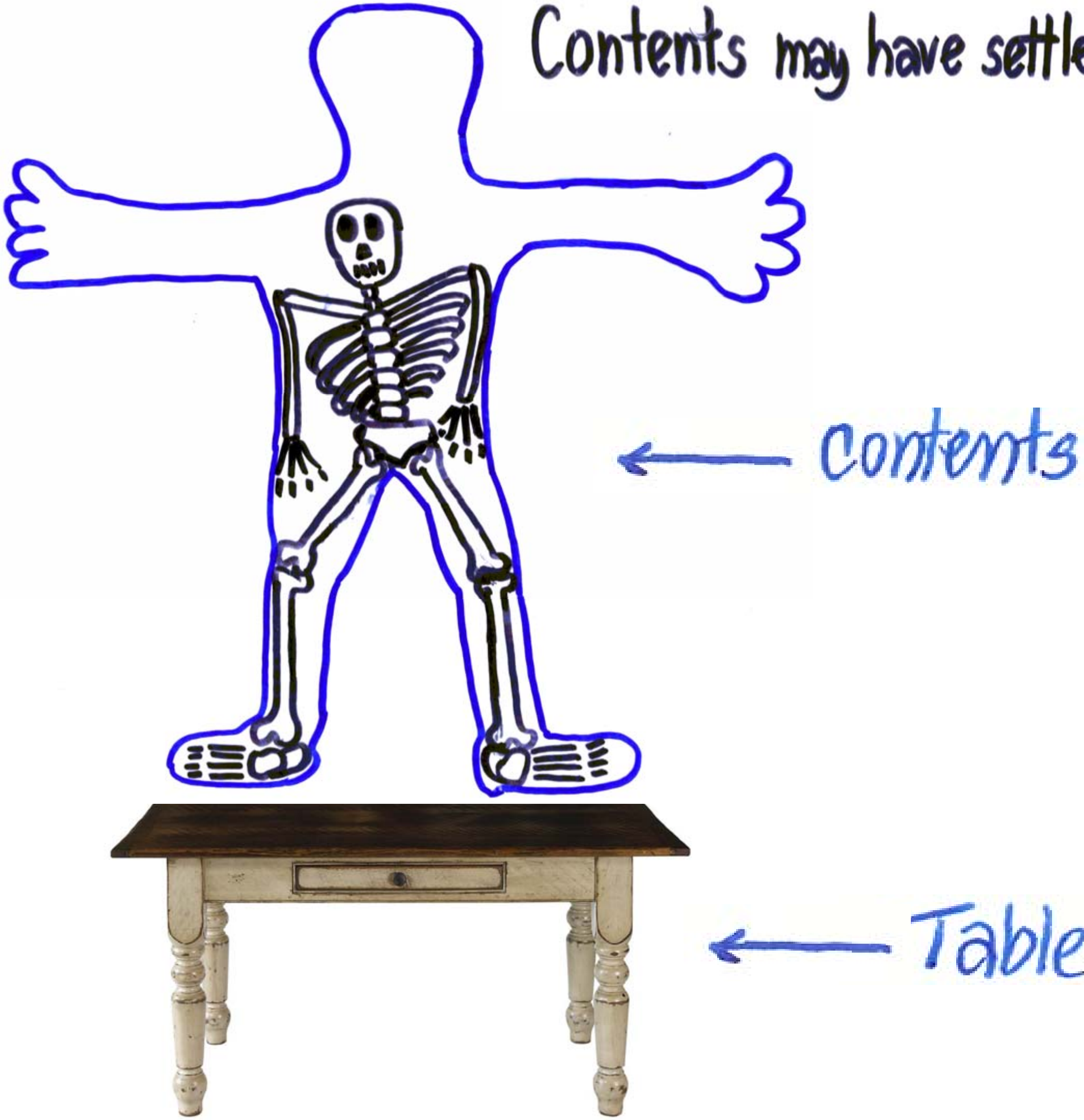
- Key searches are 3.5× slower than B-tree.

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

Table of Contents



Contents may have settled during shipping.



Outline

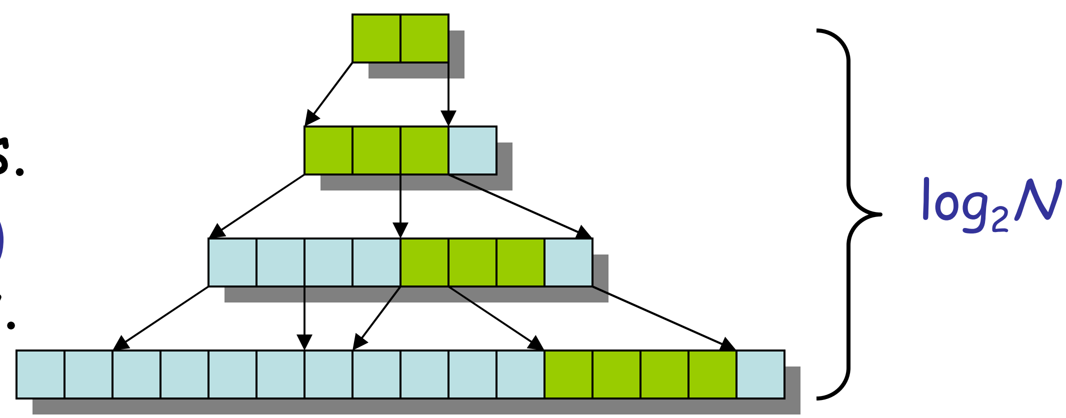
- Earlier stuff
- Cache-oblivious lookahead array (COLA)
- Cache-oblivious shuttle tree

Cache-Oblivious Lookahead Array

[fractional cascading + static-to-dynamic transformations + deamortization + cook until done]

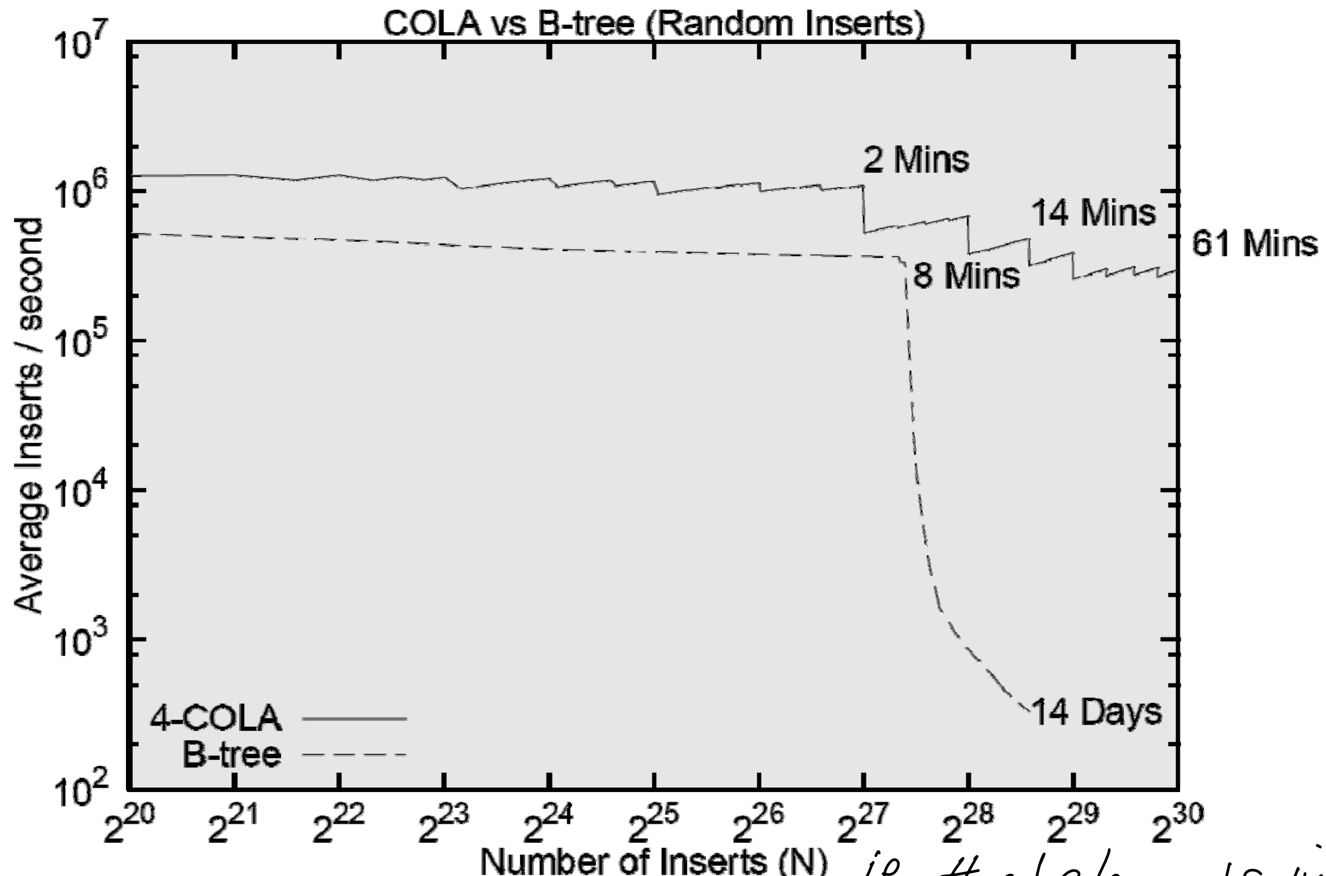
- Search: $\mathcal{O}(\log_2 N)$ block transfers.
- Insert: $\mathcal{O}((1/B)\log_2 N)$ amortized and $\mathcal{O}(\log_2 N)$ worst-case block transfers.
- Data is stored in $\log N$ arrays of sizes 2, 4, 8, 16,…
- Elements get inserted into the smallest array.
  When an array gets too full, the elements spill over into the next larger arrays.

- Redundant "lookahead pointers" aid searches.
- Search scans only $\mathcal{O}(1)$ elements in each array.



COLA vs. B-Tree: Random Inserts

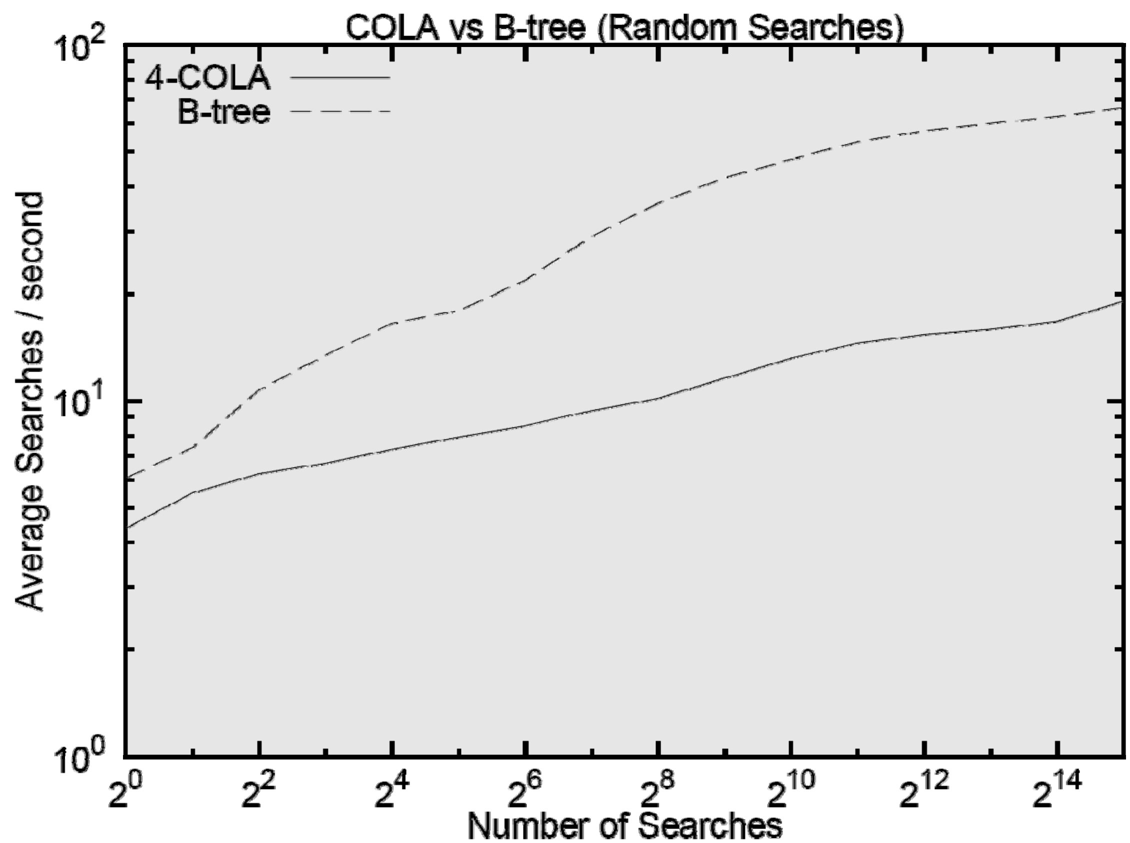
- The COLA is 1300 times faster than the B-tree
  - Expect the B-tree to level off at ~3 orders of magnitude slower than the COLA.



i.e. # of elements inserted so far

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



Outline

- Introduction
- COLA
- Shuttle tree (as static tree)
  - Buffer for fast inserts
  - Shuttle-tree layout

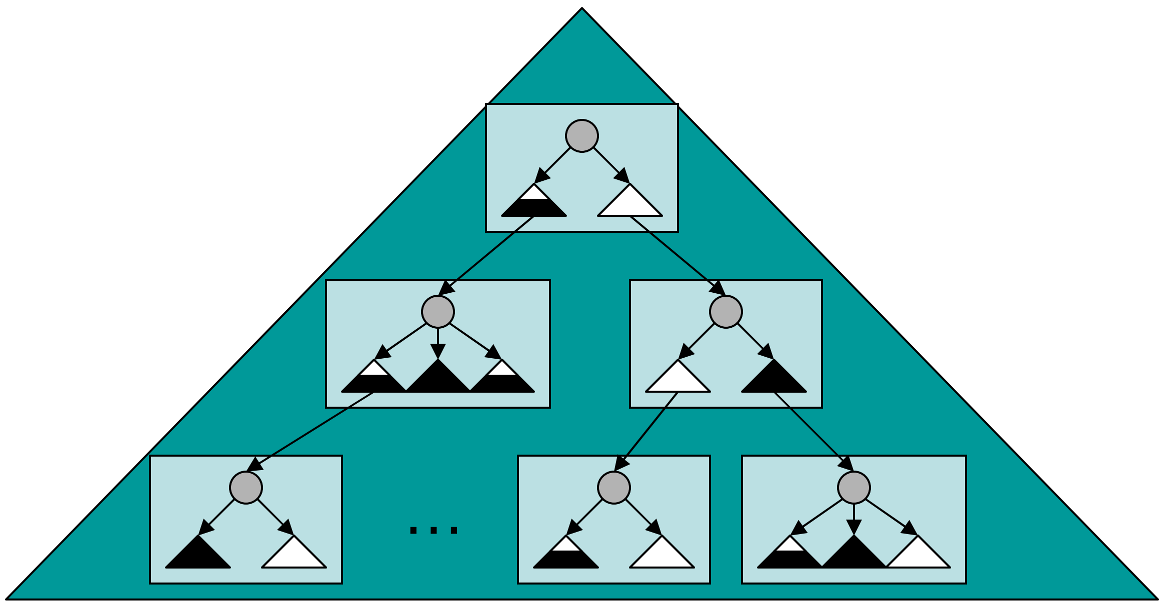
Shuttle-Tree Overview

- 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.



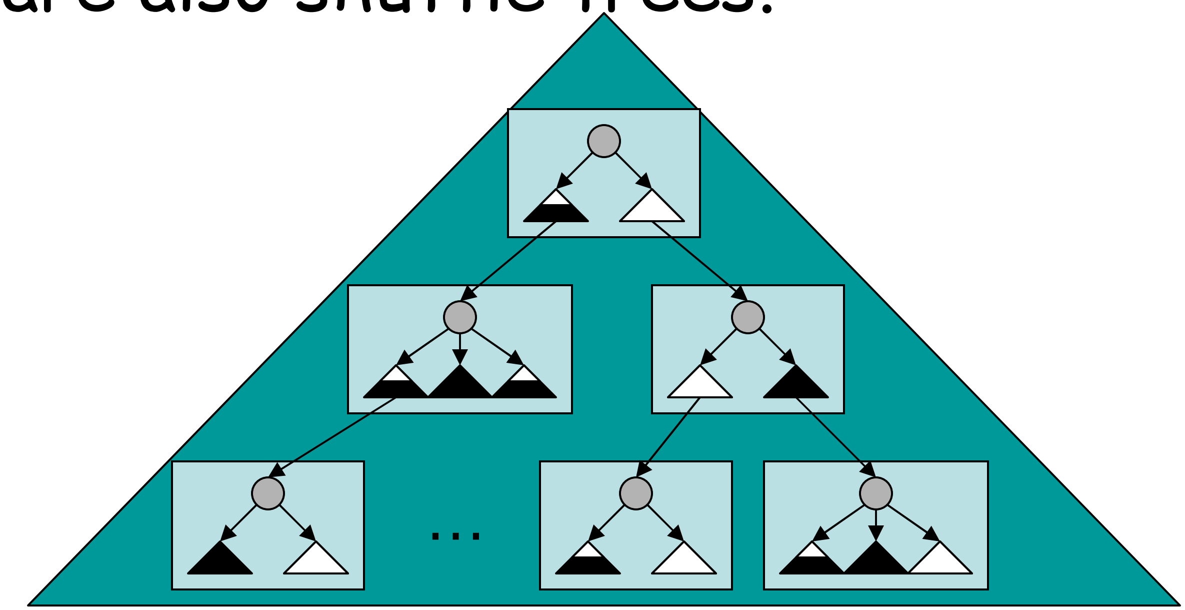
Search:

- 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.



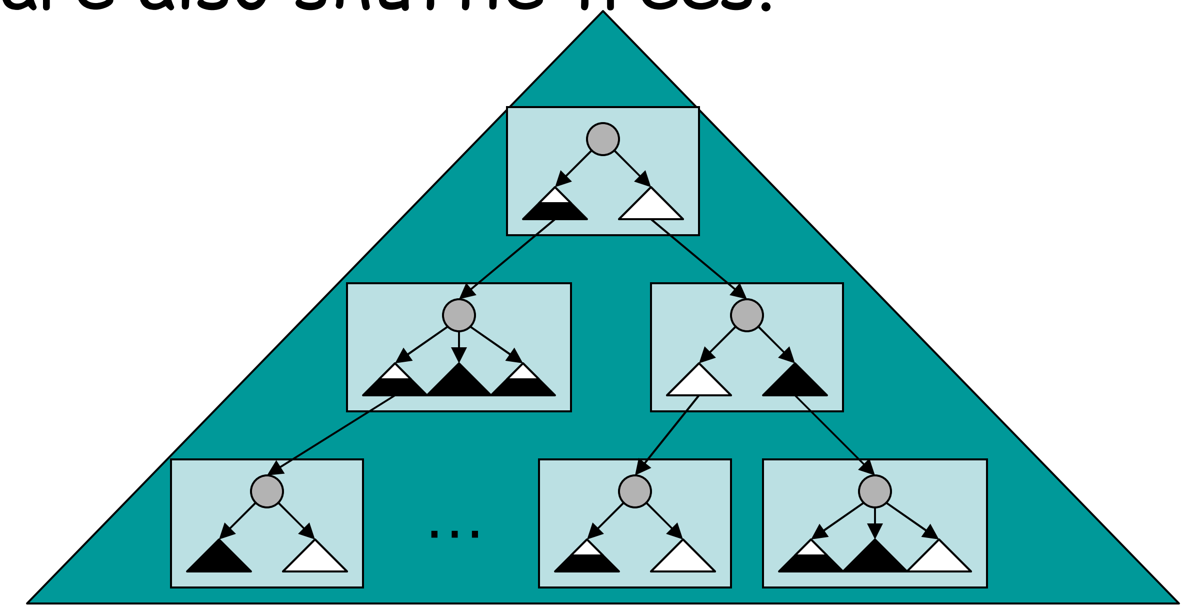
Insert:

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

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.



Difficulty: How big should the buffers be?

- Big buffers slow down searches.
- Small buffers don't improve inserts.

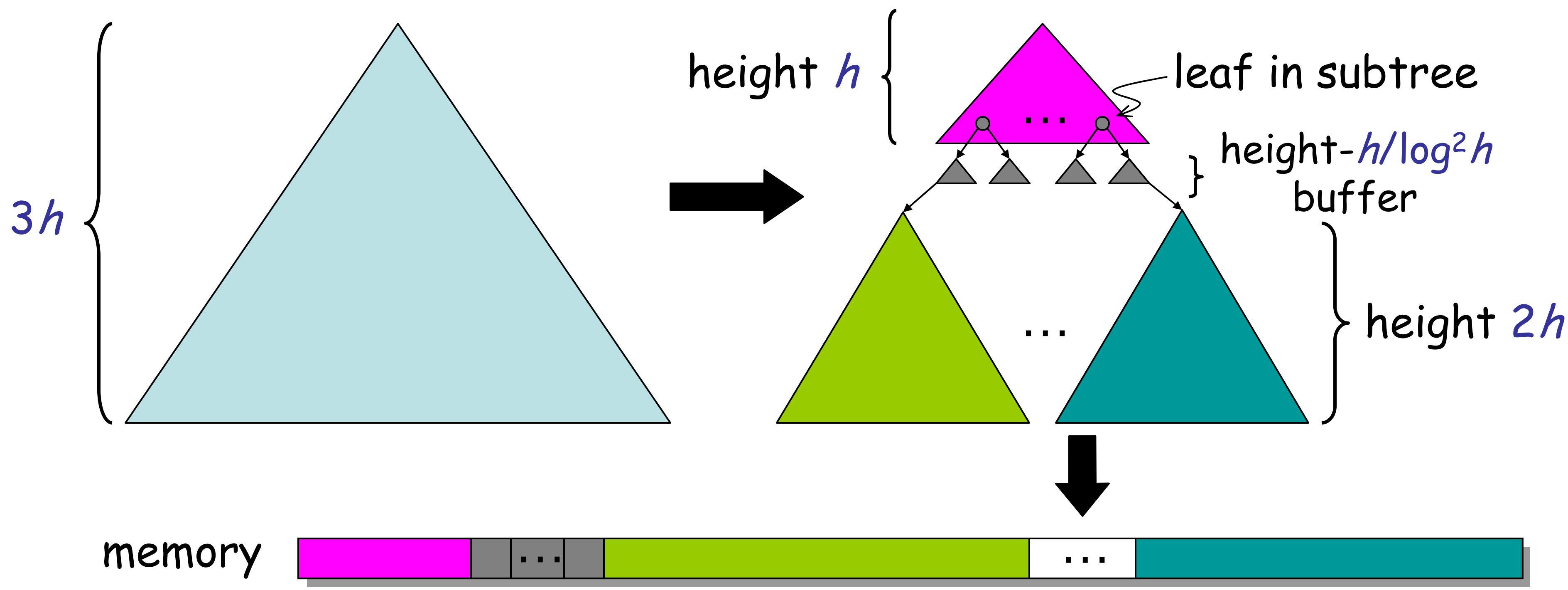
Outline

- Introduction
- COLA
- Shuttle tree (as static tree)
  - Buffer for fast inserts
  - Shuttle-tree layout

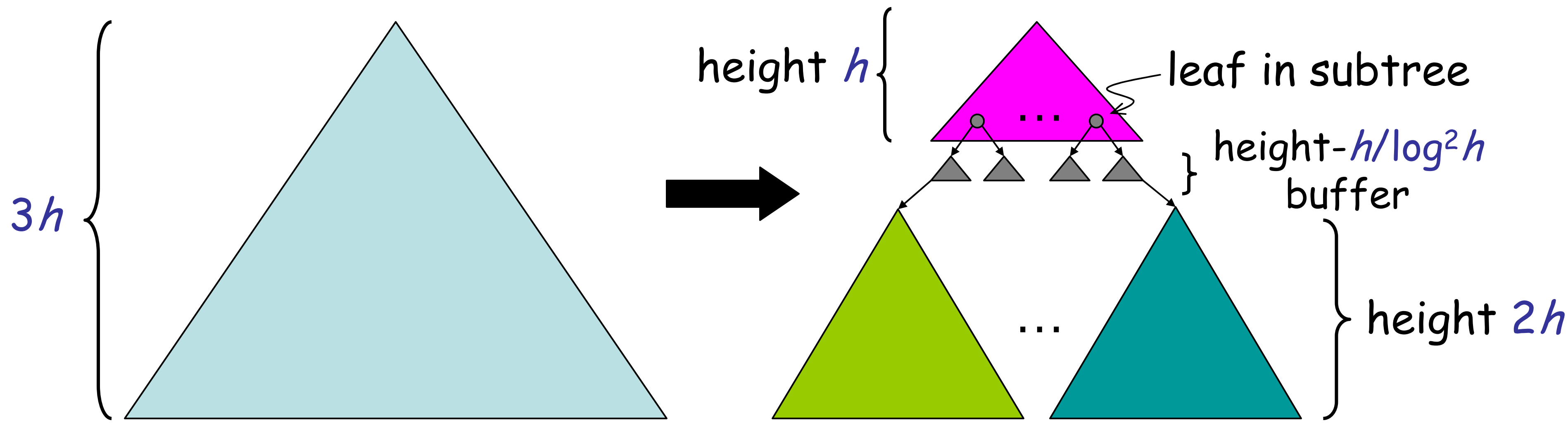
Shuttle-Tree Layout Idea

Based on recursive vEB layout [Prokop 99].

- Split tree recursively at 1/3 height.
- A leaf in a height-h recursive subtree has a height-$O(h/\log^2 h)$ shuttle-tree buffer.



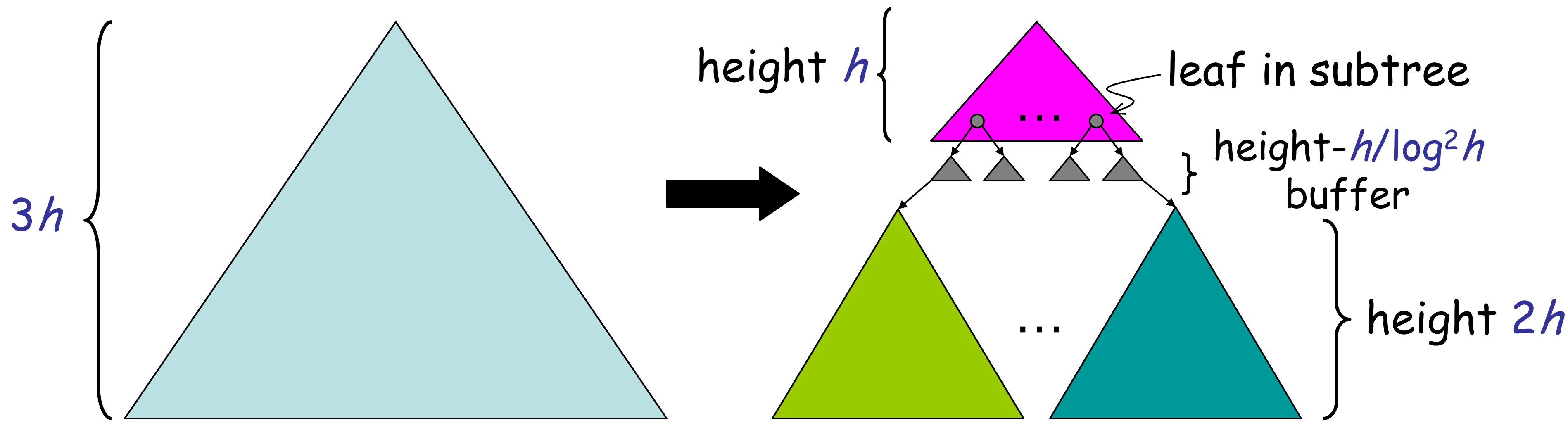
Shuttle Tree Has Good Searches



Search cost recurrence:

- $T(3h) = T(h) + T(2h) + T(h/\log^2 h)$
- with base case $T(\log B) = O(1)$,
- gives $T(\log N) = O(\log_B N)$, which is optimal.

Shuttle Tree Has Good Searches



Search cost recurrence:

- $T(3h) = T(h) + T(2h) + T(h/\log^2 h)$
- with base case $T(\log B) = O(1)$,
- gives $T(\log N) = O(\log_B N)$, which is optimal.

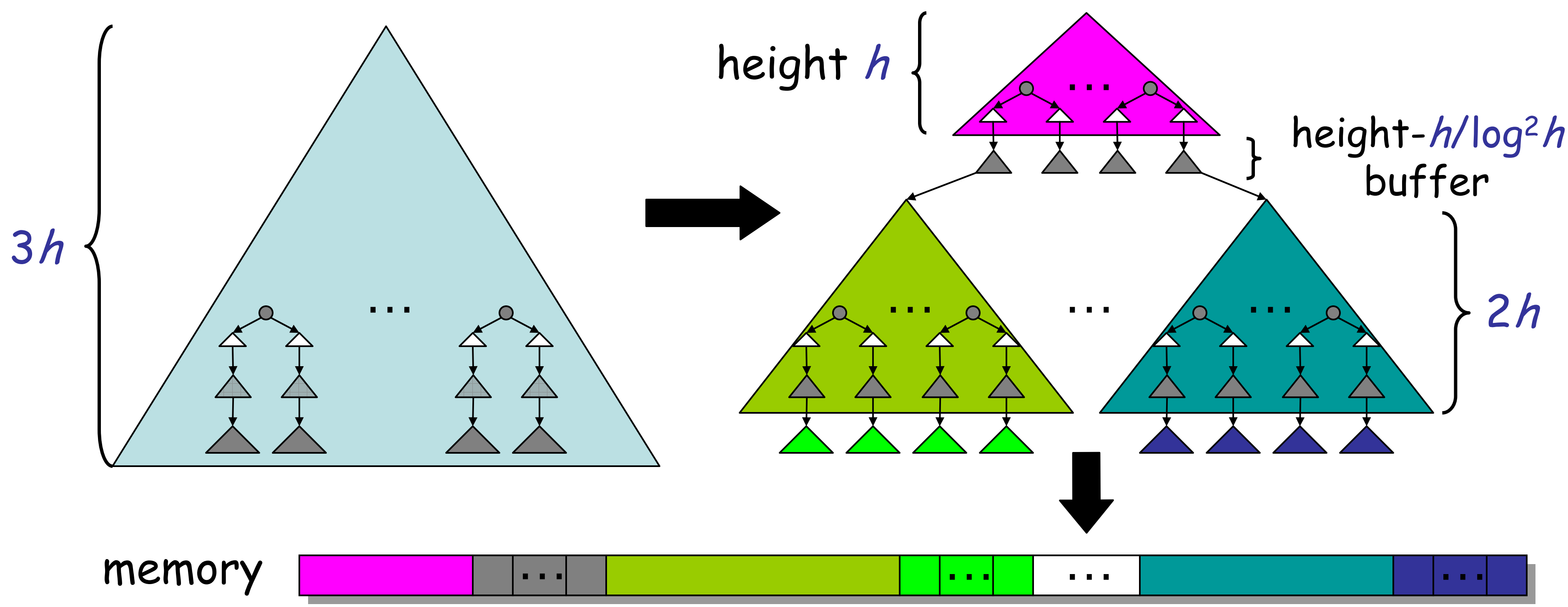
Inserts: Buffer size for $\log B$-height tree is

$\sim 2^{\log B/(\log\log B)^2} = B^{1/(\log\log B)^2}$

Shuttle-Tree Layout Expanded

- A leaf in a height-h recursive subtree has a height-$O(h/\log^2 h)$ shuttle-tree buffer.

$\Rightarrow$ Leaves have lists of buffers (multiple subtrees).

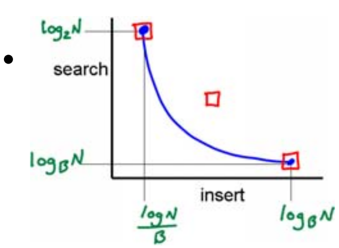


Results Summary

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]	$O(\log_2 N)$	$O((1/B)\log_2 N)$* (and $O(\log_2 N)$ w.c.)
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

- COLA random inserts beat B-tree by 1300×.
- COLA is *much* simpler than shuttle tree.



Open: Is there anything between COLA and shuttle tree? (*i.e.*, CO B^{ε}-tree)

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.
- All tests used 4-COLAs.

Tokutek™

- Founded by Michael Bender, Martin Farach-Colton, Bradley Kuszmaul, Dave Wells.
- We build storage engines.
- We have a beta storage engine for MySQL. (Other products are coming down the pike.)
- We tested against MySQL's best storage engine, InnoDB.
  - MySQL gave us a test suite.
  - TokuDB finished in about two hours.
  - We pulled the plug on InnoDB after two weeks, and it wasn't done yet.

- Visit our website http://tokutek.com
- Read Martin's blog.
- We have offices in Boston and NYC.
- We are hiring.

Tokutek™

- Visit our website http://tokutek.com
- Read Martin's blog.
- We have offices in Boston and NYC.
- We are hiring.