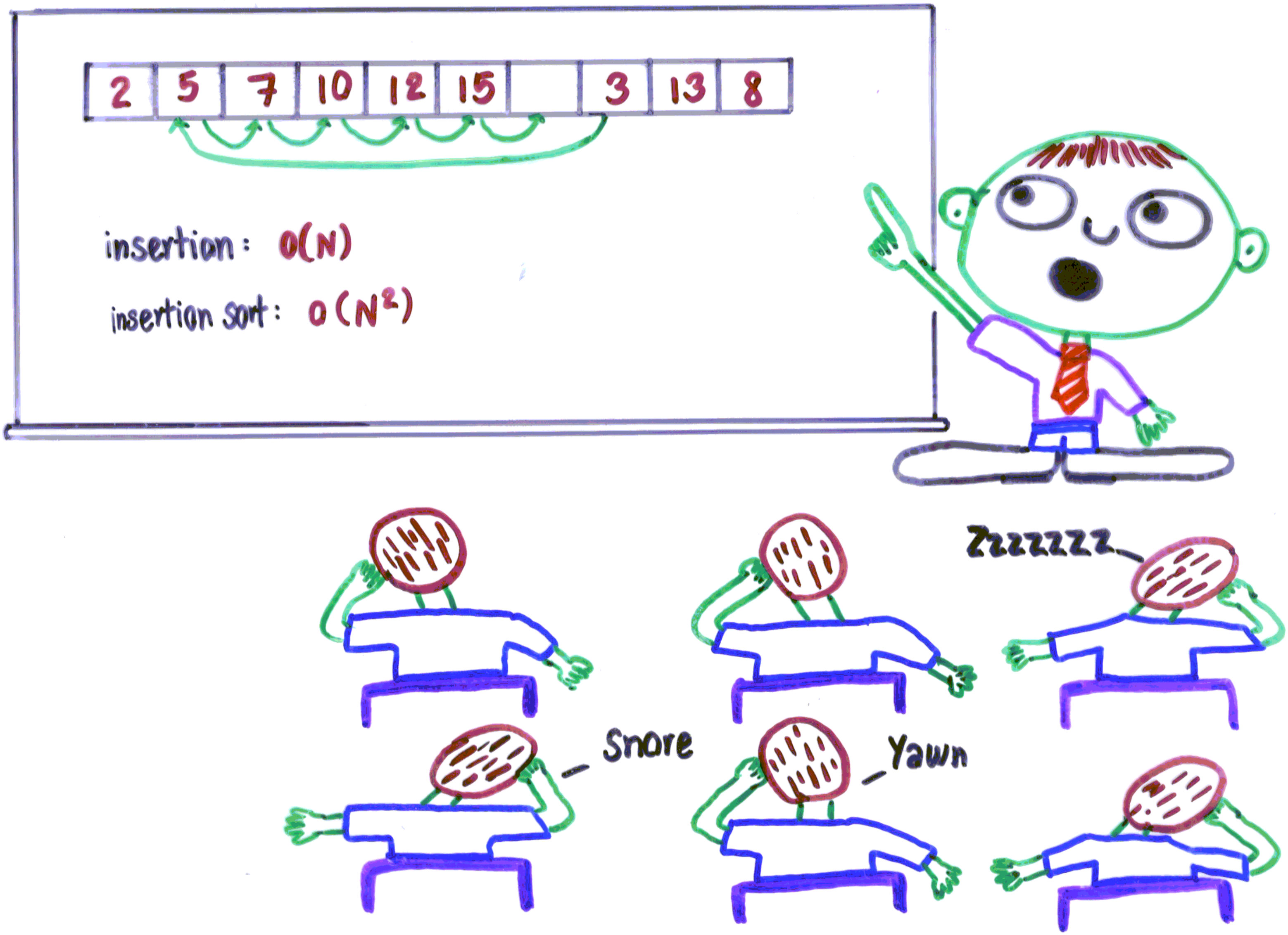
An Adaptive Packed-Memory Array

Michael A. Bender

Stony Brook and Tokutek[R], Inc

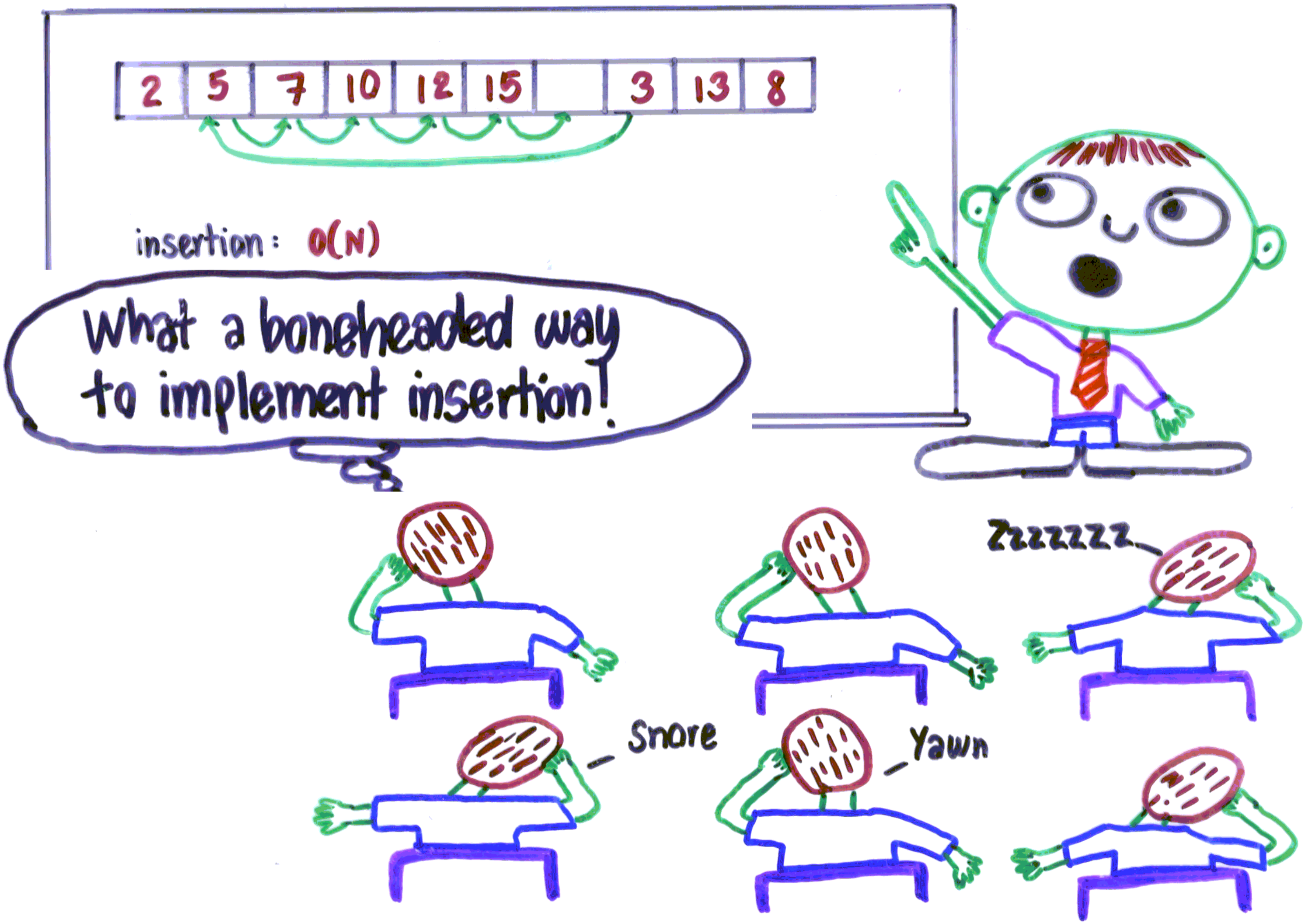
Fall 1990. I'm an undergraduate.

Insertion-Sort Lecture.



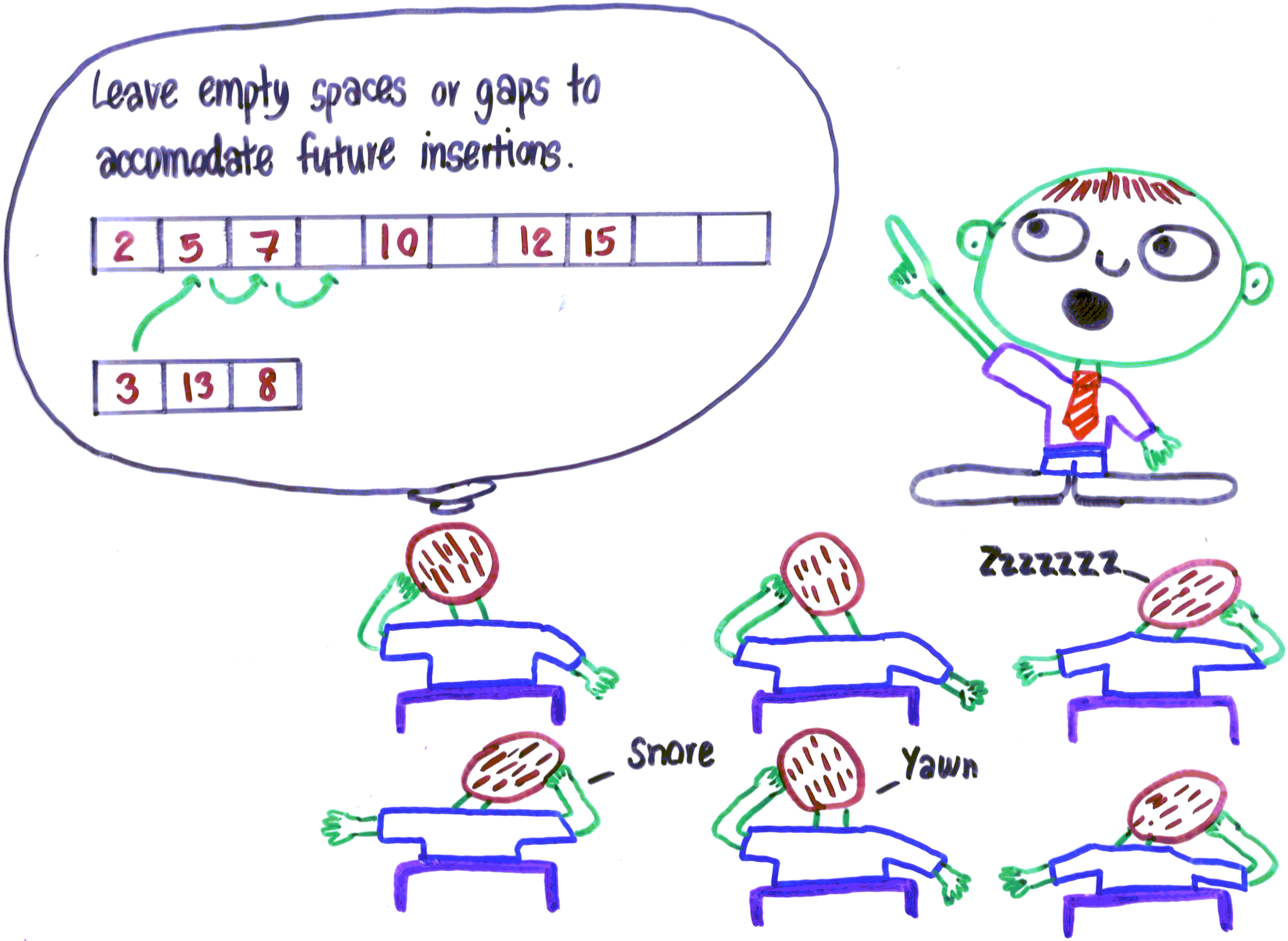
Fall 1990. I'm an undergraduate.

Insertion-Sort Lecture.

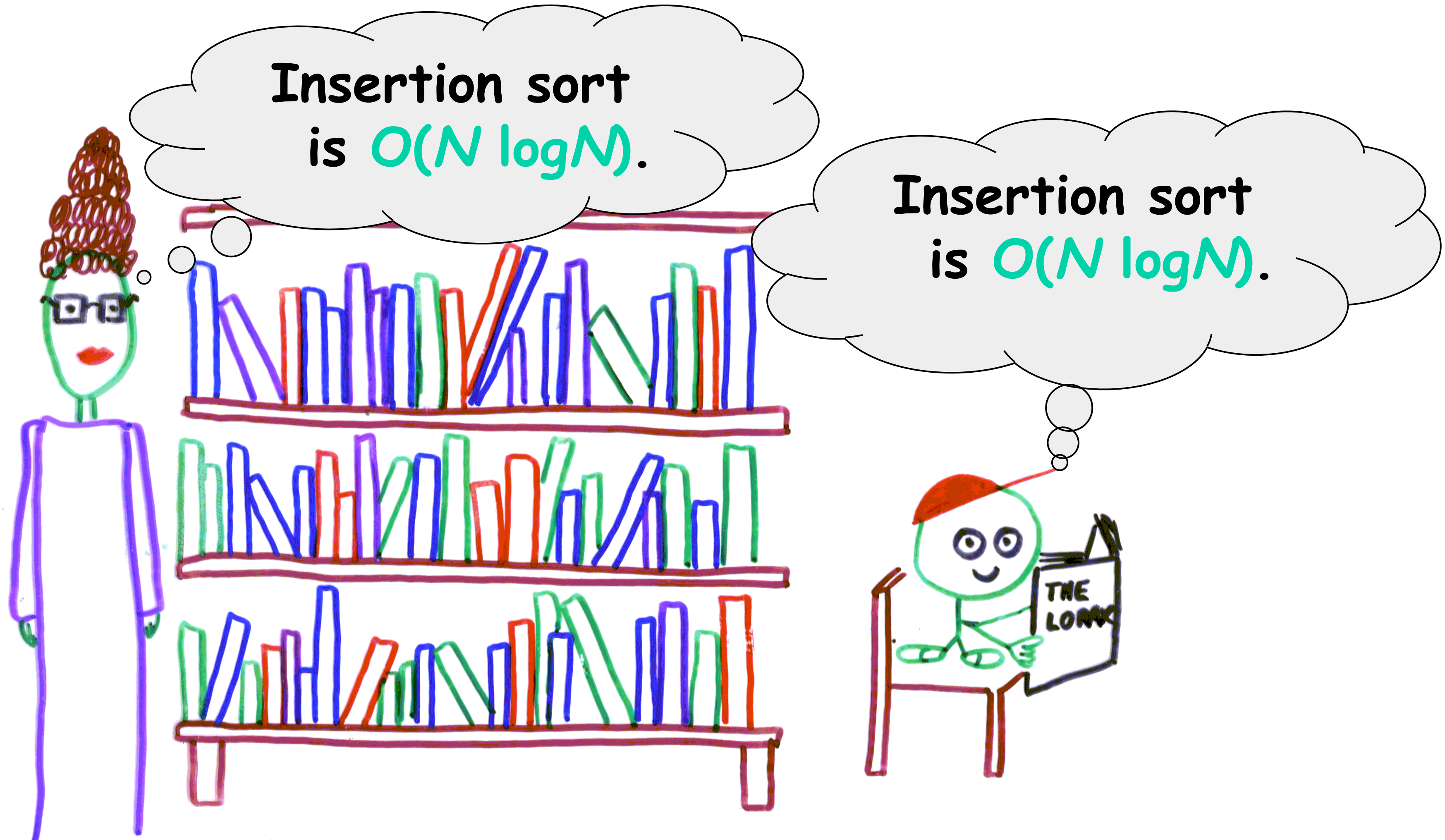


Fall 1990. I'm an undergraduate.

Insertion-Sort Lecture.



Anybody who has spent time in a library knows that insertions are cheaper than linear time.



LibrarySort [Bender, Farach-Colton, Mosteiro 04] :

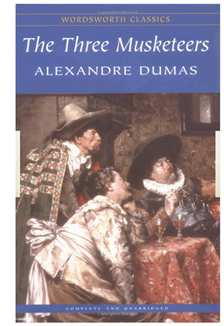
$O(N \log N)$ sorting for average-case insertions.

How is LibrarySort like a library?

- Leave gaps on shelves so shelving is fast
- Putting books *randomly* on shelves with gaps: At most $O(\log N)$ books need to be moved with high probability* to make room for a new book.

* Probability $>1-1/\mathrm{poly}(N)$, N=#books.

But what if Library buys 10 copies of….

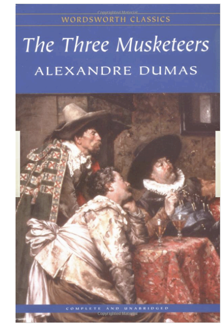


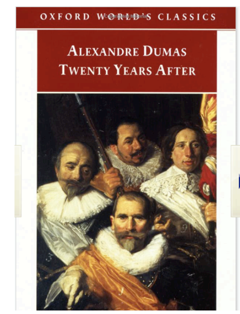
- The Three Musketeers (Dumas)

…   .

But what if Library buys 10 copies of….

- The Three Musketeers (Dumas)
- 20 Years After (Dumas)

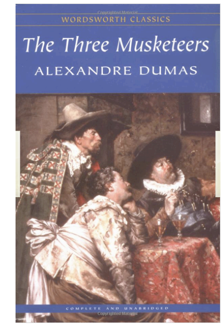


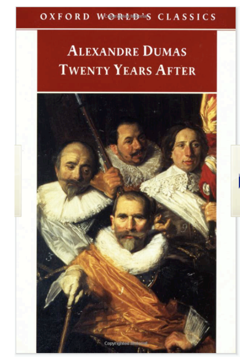


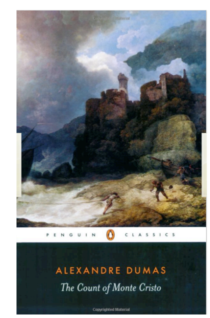
… .

But what if Library buys 10 copies of….

- The Three Musketeers (Dumas)
- 20 Years After (Dumas)
- The Count of Monte Cristo, Vol. 1,2,3 (Dumas)
- The Vicomte of Bragelonne, Vol. 1,2,3 (Dumas)



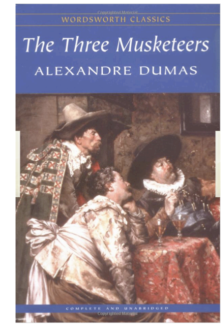


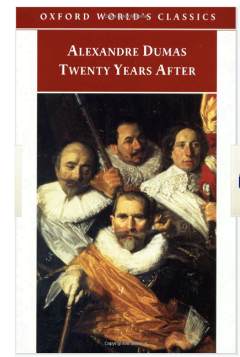


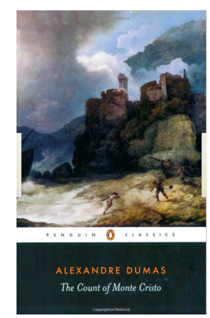
… .

But what if Library buys 10 copies of….

- The Three Musketeers (Dumas)
- 20 Years After (Dumas)
- The Count of Monte Cristo, Vol. 1,2,3 (Dumas)
- The Vicomte of Bragelonne, Vol. 1,2,3 (Dumas)
- All other books by Dumas
- Now there's a bolus of books on in one place. Can we still maintain tiny shelving costs?



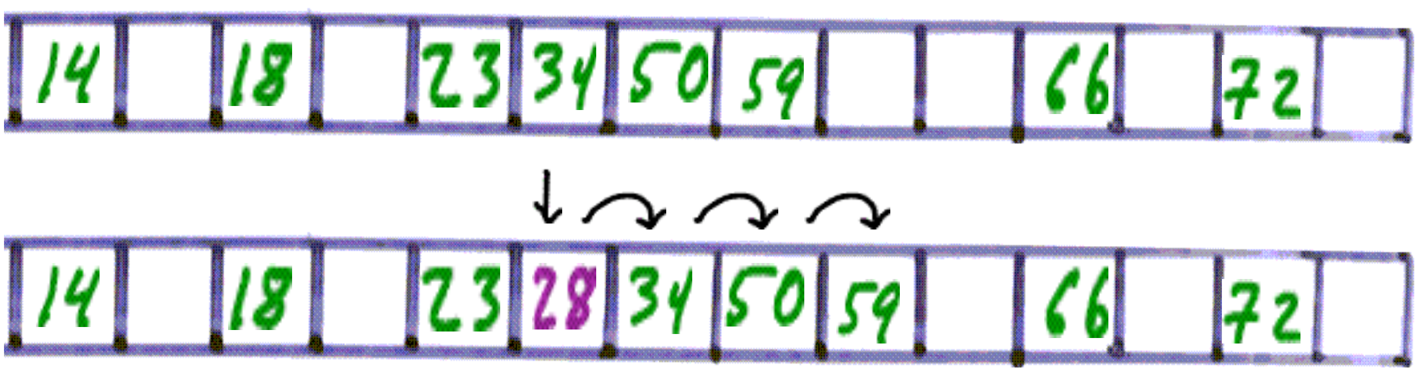




... .

Insertions into Array with Gaps

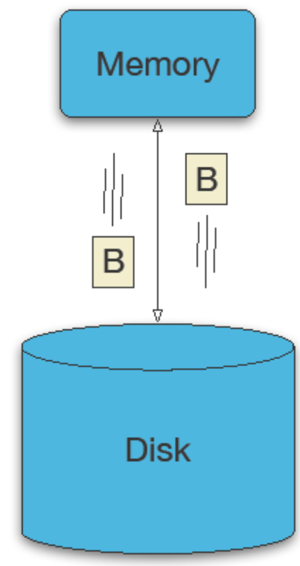
- Dynamically maintain N elements sorted in memory/on disk in a $\Theta(N)$ -sized array
  - Idea: rearrange elements & gaps to accommodate future insertions



- **Objective**: Minimize amortized (technical form of ave) # of elts moved per update.

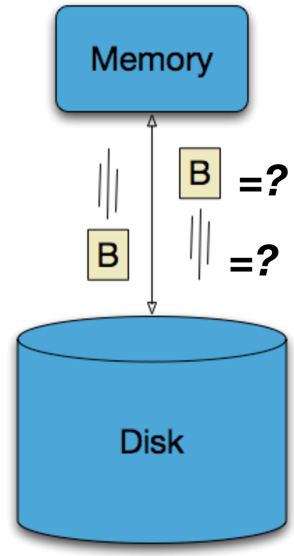
Actually Two Objectives

- Minimize # **elements moved** per insert.
- Minimize # **block transfers** per insert.
- Disk Access Model (DAM) of Computer
  - Two levels of memory
  - Two parameters:
    block size B, memory size M.



Cache-Oblivious Model [FLPR '99]

- Disk Access Model (DAM)
  - Two levels of memory
  - Two parameters:
    block size B, memory size M.



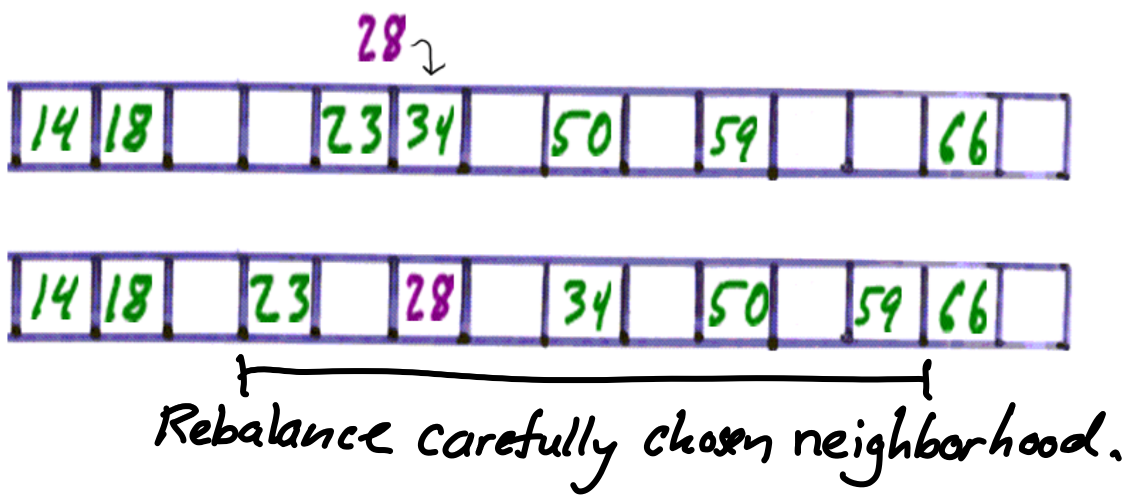
- Cache-Oblivious Model (CO) :
  - Similar to DAM, but parameters B and M are unknown to the algorithm or coder.
  - (Of course, used in proofs.)

Platform independent, i.e., memory-hierarchy universal.

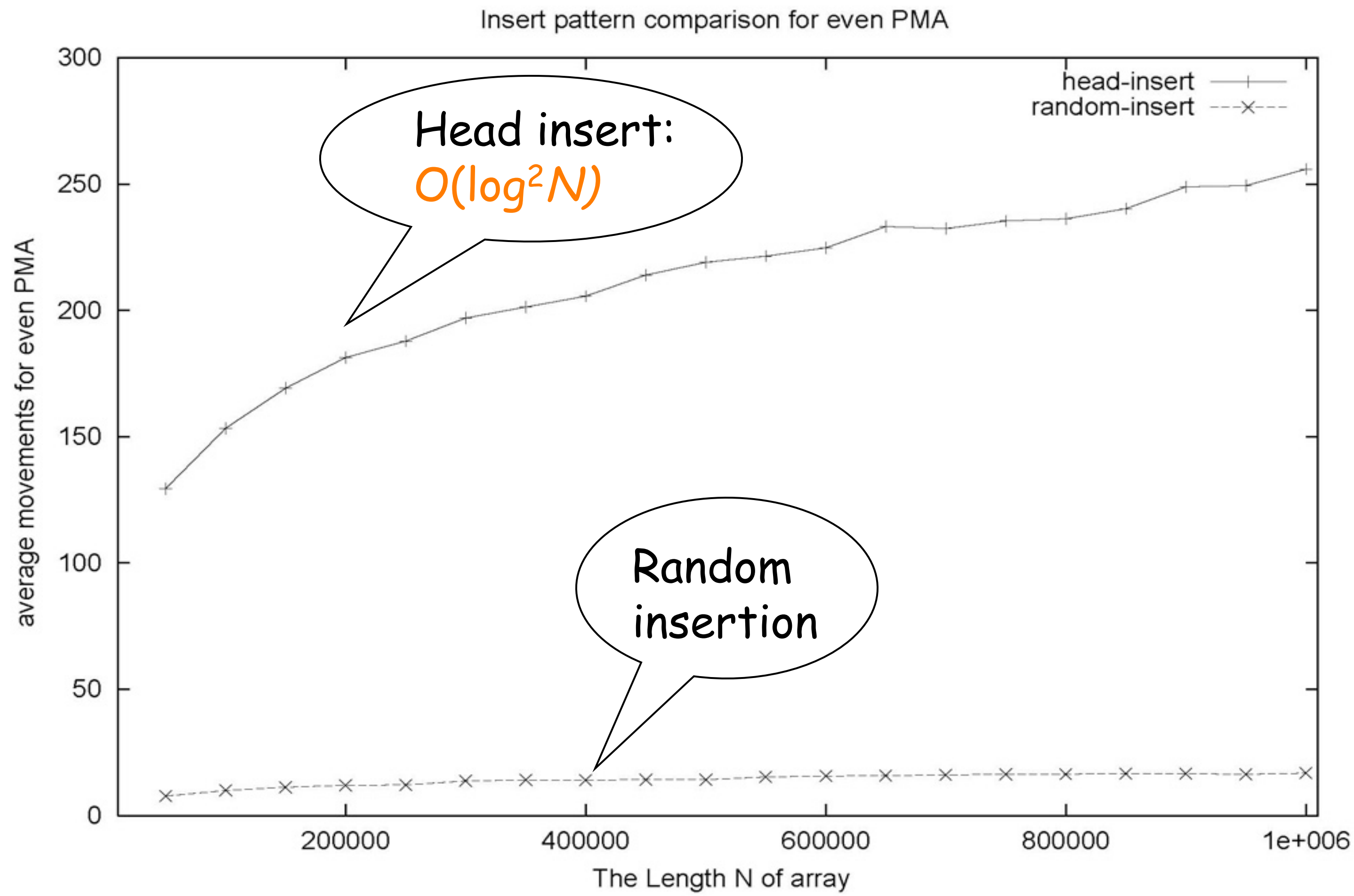
Packed-Memory Array (PMA)

[Bender, Demaine, Farach-Colton 00,05]

- **(Worst-case) Inserts/Deletes:**
  - $O(\log^2 N)$ amortized element moves
  - $O(1+(\log^2 N)/B)$ amortized memory transfers
- **Scans of *k* elements after given element:**
  - $O(1+k/B)$ memory transfers



PMA is good, but…



Problem: a worst case for PMA is sequential inserts, but this is a common case for databases. Industrial data structures (Oracle, TokuDB) are optimized for sequential inserts.

An Adaptive PMA

[Bender, Hu 2007]

- **Same** guarantees as PMA:

  $O(\log^2 N)$ element moves per insert/delete

  $O(1+(\log^2 N)/B)$ memory transfers

- Optimized for common insertion patterns:

  insert-at-head (sequential inserts)

  random inserts

  bulk inserts (repeatedly insert $O(N^b)$ elements in random position, $0 \le b \le 1$)

  Guarantees:

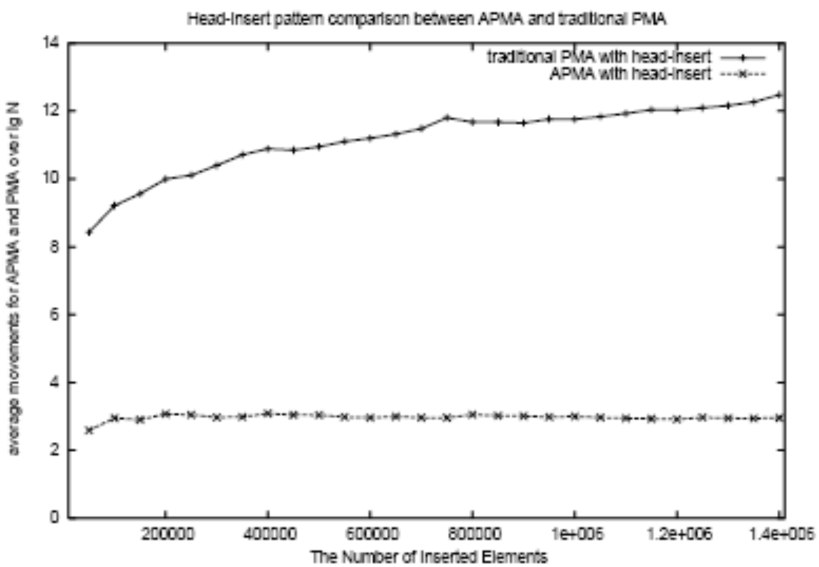
  $O(\log N)$ element moves

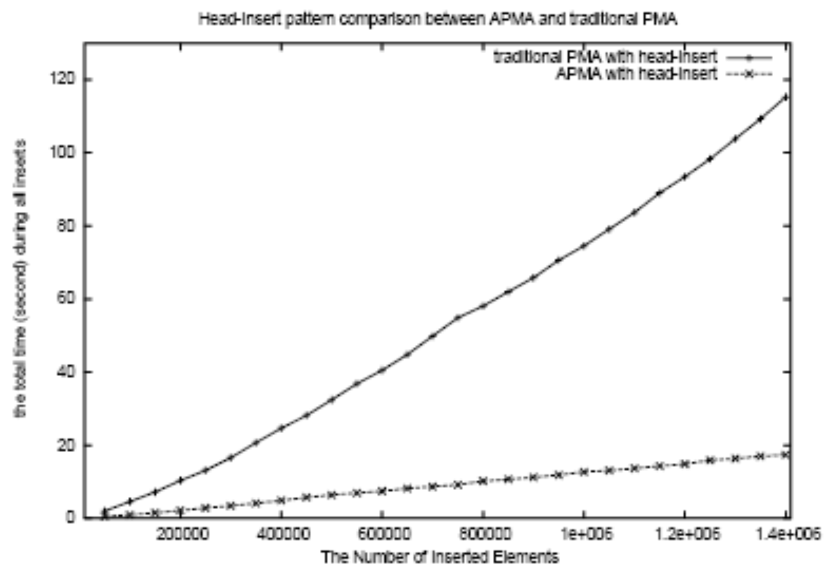
  $O(1+(\log N)/B)$ mem transfers

log N factor improvement.

Sequential Inserts

Inserts "hammer" on one part of the array.



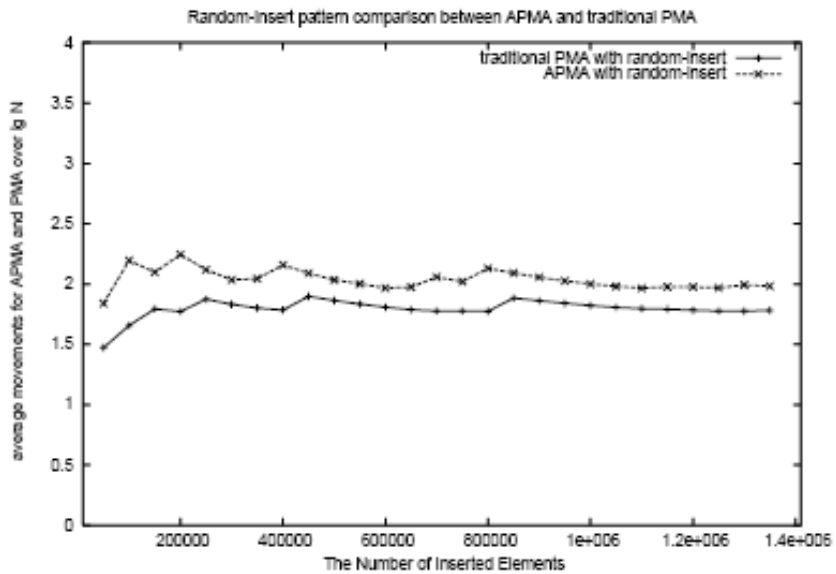


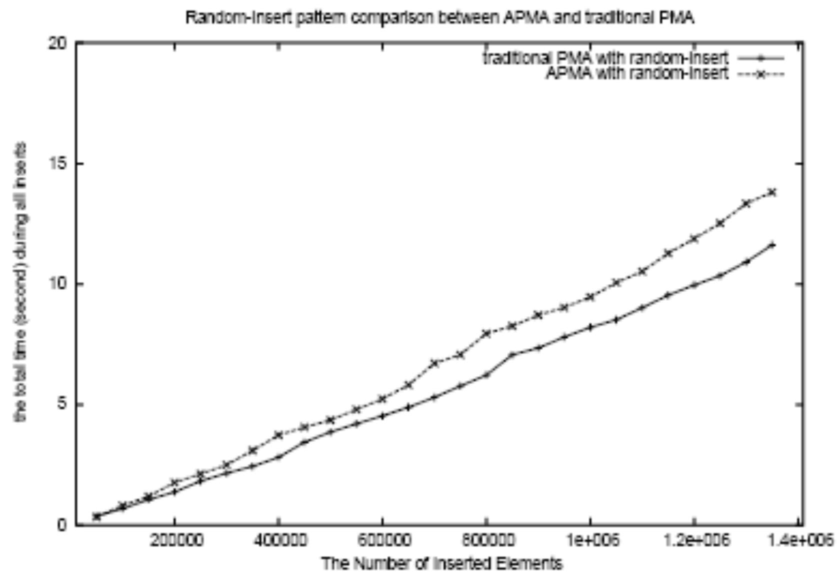
Amortized moves over log N

running time

Random Inserts

Insertions are after random elements.



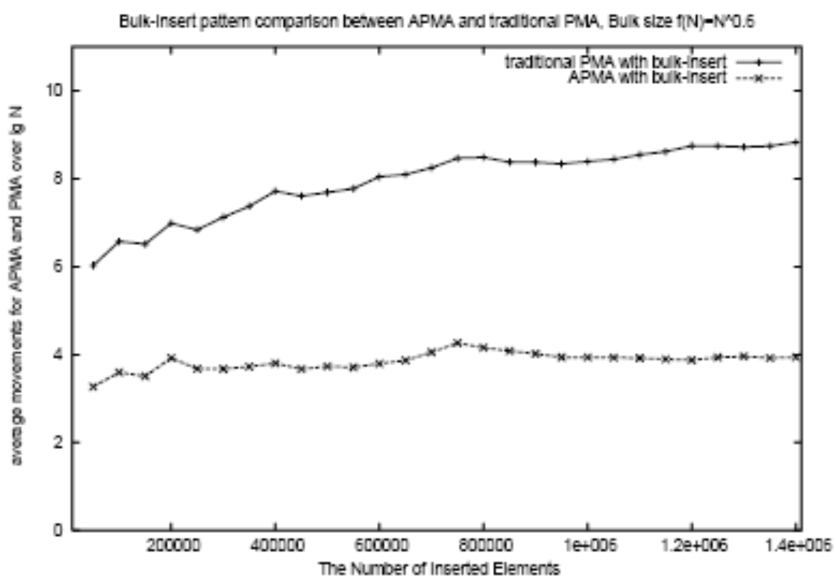


Amortized moves over log N

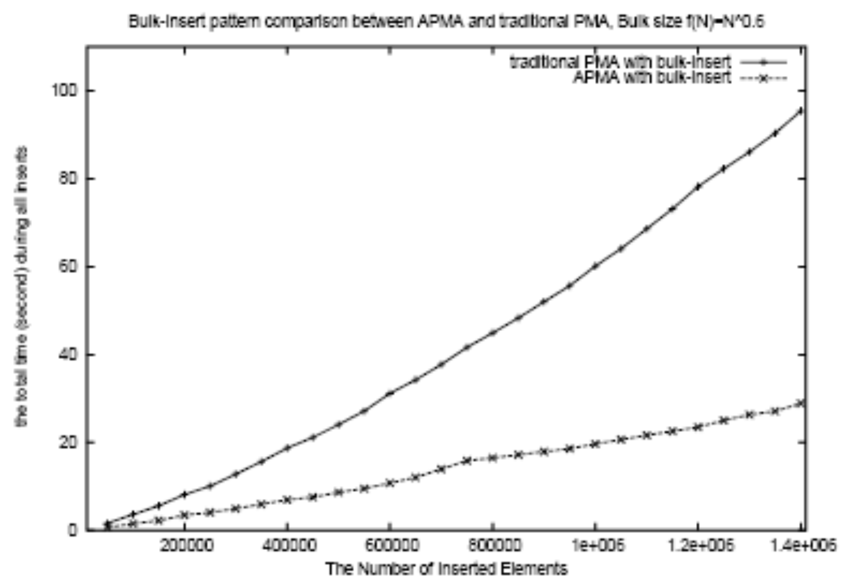
running time

Bulk Inserts

Repeatedly insert $O(N^b)$ elements after a random element ($0 \le b \le 1$).

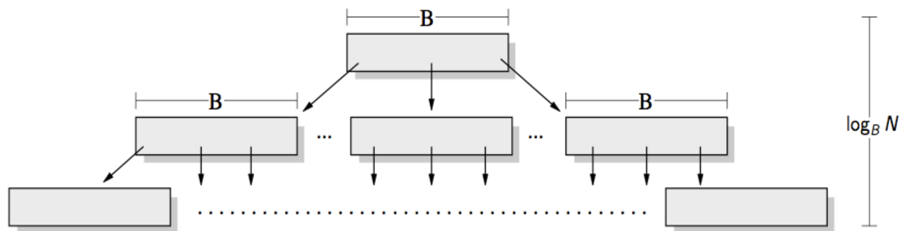


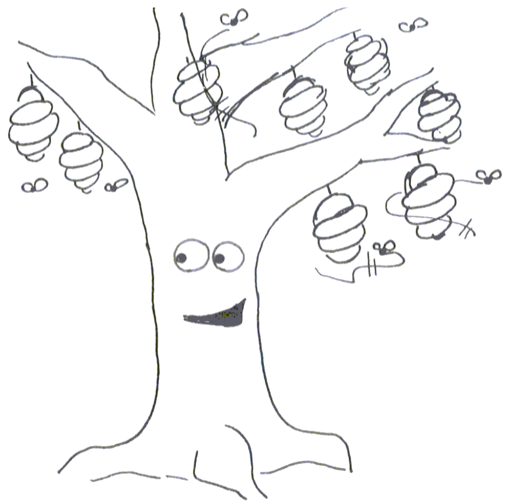
Amortized moves over $\log N$



running time

Sample Applications

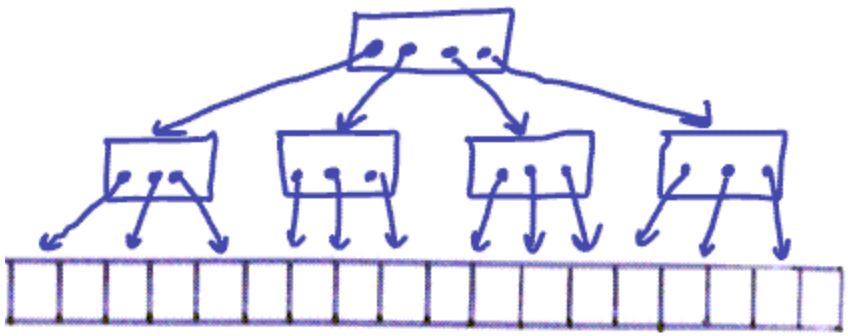




Maintain data physically in order on disk

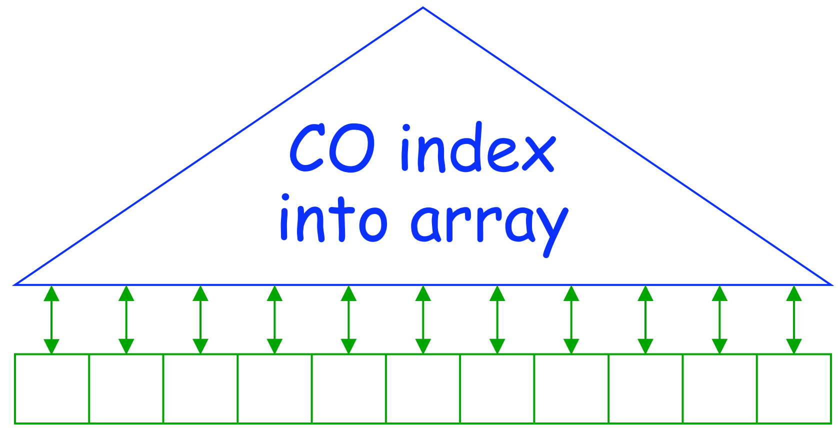
- Traditional and "cache-oblivious" B-trees
  - Core of all databases and file systems
- My startup Tokutek
- Even an online dating website

Sorted Arrays with Gaps are Used in Several External Memory Dictionaries



Locality-preserving B-tree

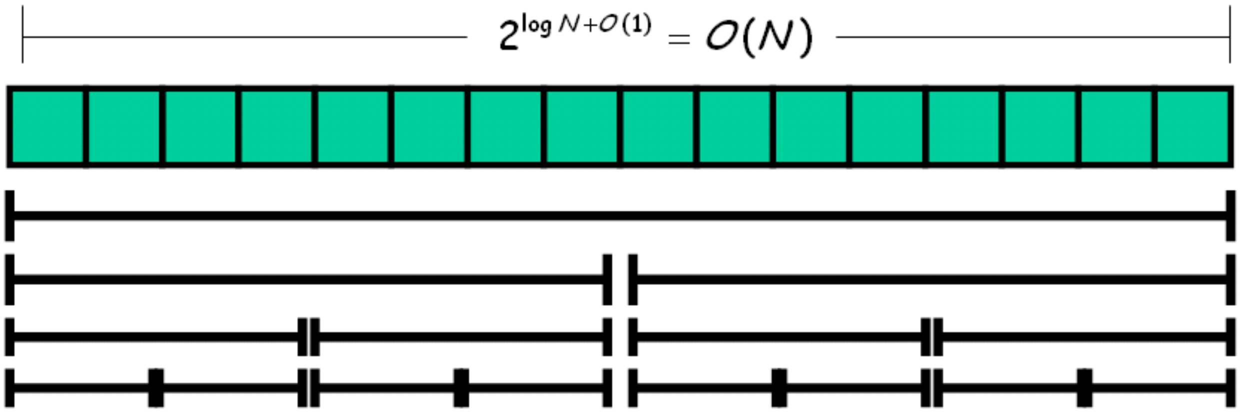
[Raman 99]



Cache-oblivious B-tree

[Bender, Demaine, Farach-Colton 00]
[Rahman, Cole, Raman 01]
[Brodal, Fagerberg, Jacob 02]
[Bender, Duan, Iacono, Wu 02,04]
[Bender,Farach-Colton, Kuszmaul, 06]

Imaginary Intervals in PMA

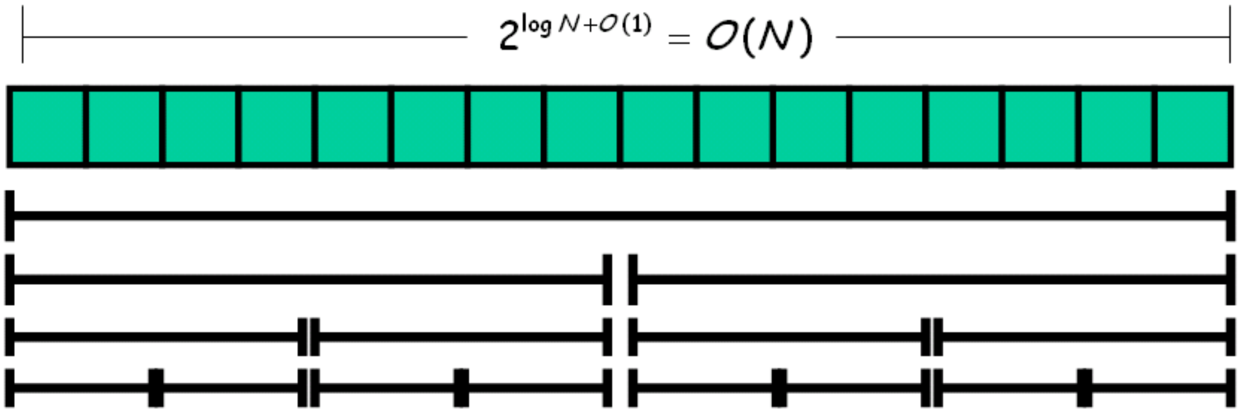


Density Thresholds

Upper	Lower
$\tau_3 = .7$	$\rho_3 = .3$
$\tau_2 = .8$	$\rho_2 = .25$
$\tau_1 = .9$	$\rho_1 = .2$
$\tau_0 = 1.0$	$\rho_0 = .15$

$$\tau_i - \tau_{i+1} = \Theta(\rho_{i+1} - \rho_i) = \Theta(1/\log N).$$

Imaginary Intervals in PMA



Density Thresholds

Upper	Lower
$\tau_3 = .7$	$\rho_3 = .3$
$\tau_2 = .8$	$\rho_2 = .25$
$\tau_1 = .9$	$\rho_1 = .2$
$\tau_0 = 1.0$	$\rho_0 = .15$

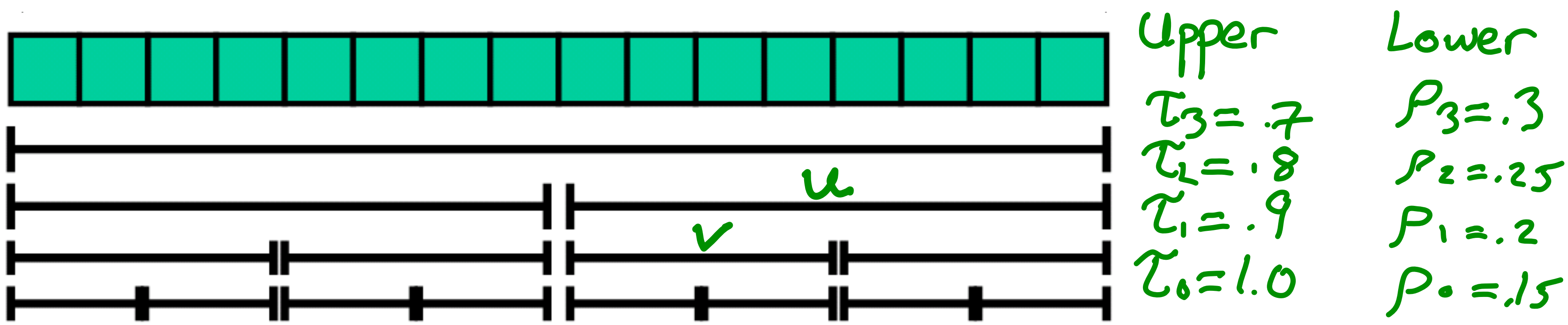
$$\tau_i - \tau_{i+1} = \Theta(\rho_{i+1} - \rho_i) = \Theta(1/\log N).$$

To insert:

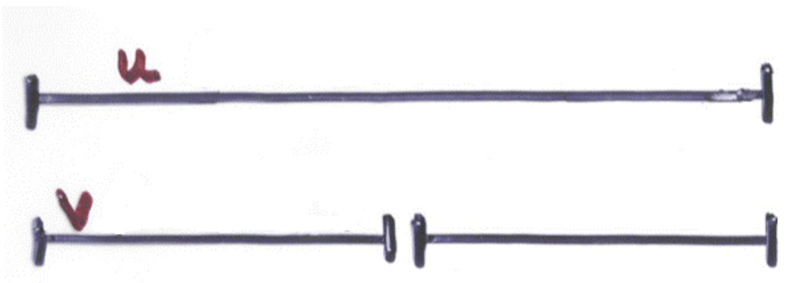
- Try to insert in leaf interval.
- If interval full, ***rebalance*** smallest enclosing interval within thresholds.

Analysis Idea: $O(\log^2 N)$ amortized element moves per insert

- $O(\log N)$ amort. moves to insert into interval
  - Amortized analysis: Charge rebalance of interval u to inserts into child interval v
- Insert in $O(\log N)$ intervals for insert in PMA



Analysis of $O(\log^2 N)$ Moves/Insert



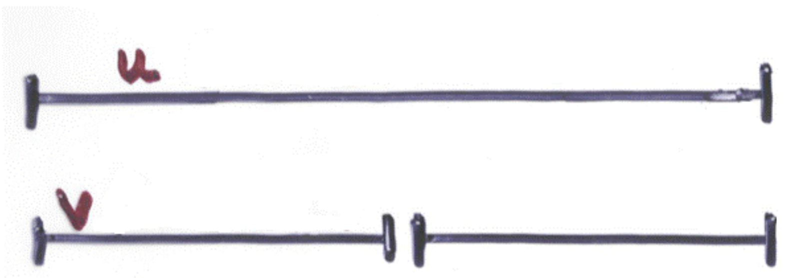
density thresholds $\tau_{\ell+1}$, $\rho_{\ell+1}$

density thresholds τ_{ℓ} , ρ_{ℓ}

Before rebalance: $\text{density}(v) > \tau_\ell$ or $\text{density}(v) < \rho_\ell$

After rebalance: $\rho_{\ell+1} < \text{density}(v) < \tau_{\ell+1}$

Analysis of $O(\log^2 N)$ Moves/Insert



density thresholds $\tau_{\ell+1}$, $\rho_{\ell+1}$
density thresholds τ_{ℓ}, ρ_{ℓ}

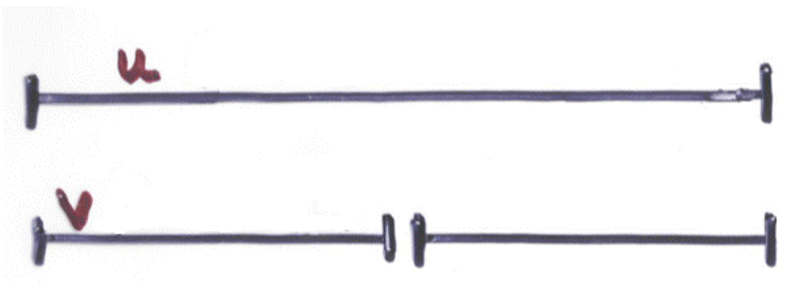
Before rebalance: $\text{density}(v) > \tau_\ell$ or $\text{density}(v) < \rho_\ell$

After rebalance: $\rho_{\ell+1} < \text{density}(v) < \tau_{\ell+1}$

Amortized cost of rebalancing u $= \dfrac{\text{cost of rebalance}}{\text{inserts betw. rebalance}} = \dfrac{\text{size}(u)}{\text{size}(v)} \text{Max}\left\{\dfrac{1}{\tau_\ell - \tau_{\ell+1}}, \dfrac{1}{\rho_{\ell+1} - \rho_\ell}\right\}$

$= O(\log N)$

Analysis Summary

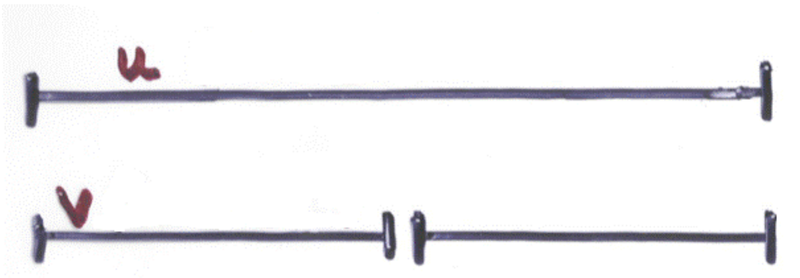


density thresholds $\tau_{\ell+1}, \rho_{\ell+1}$

density thresholds τ_{ℓ}, ρ_{ℓ}

- Charge rebalance cost of u to inserts into v
  - After rebalance v within threshold of parent u
- Amortized cost of $O(\log N)$ to insert into u

Analysis Summary



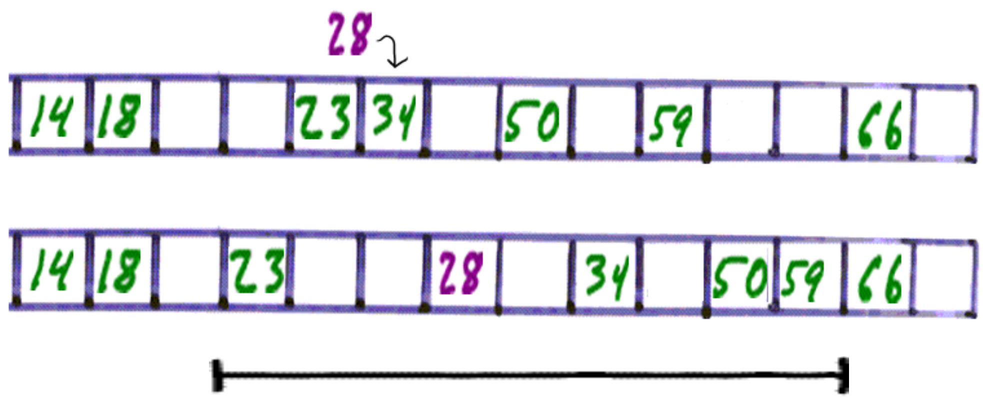
density thresholds $\tau_{\ell+1}$, $\rho_{\ell+1}$
density thresholds τ_{ℓ} , ρ_{ℓ}

- Charge rebalance cost of u to inserts into v
  - After rebalance v within threshold of parent u
- Amortized cost of $O(\log N)$ to insert into u
- But each insert is into $O(\log N)$ intervals

- Total: $O(\log^2 N)$ amortized moves

Idea of Adaptive PMA

- Adaptively remember elements that have many recent inserts nearby.
- Rebalance ***unevenly***.
  Add extra space near these volatile elements.

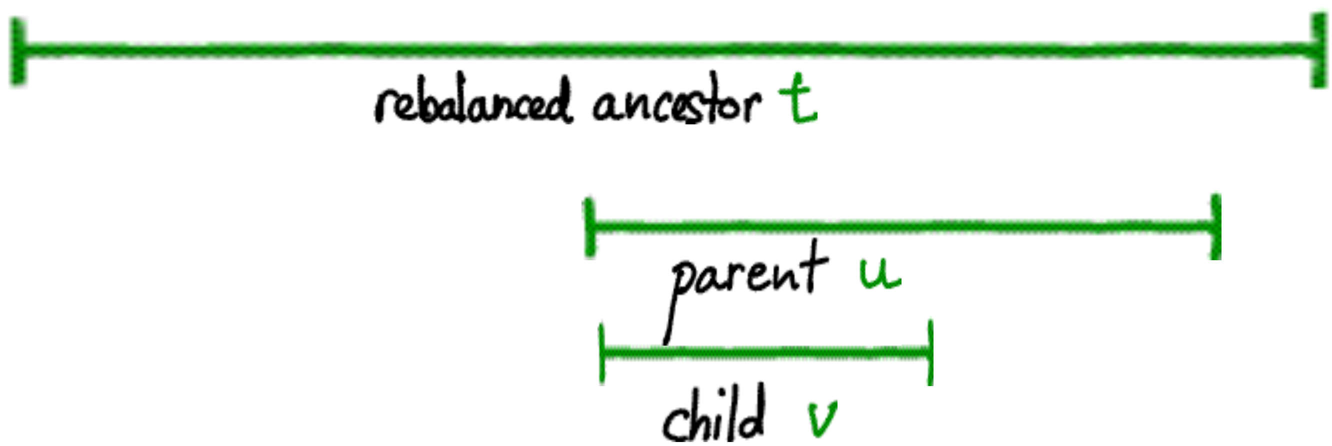


- This strategy overcomes a $\Omega(\log^2 N)$ lower bound [Dietz, Sieferas, Zhang 94] for "smooth" rebalances

Why $O(\log^2 N)$ Can Be Improved in the Common Case.

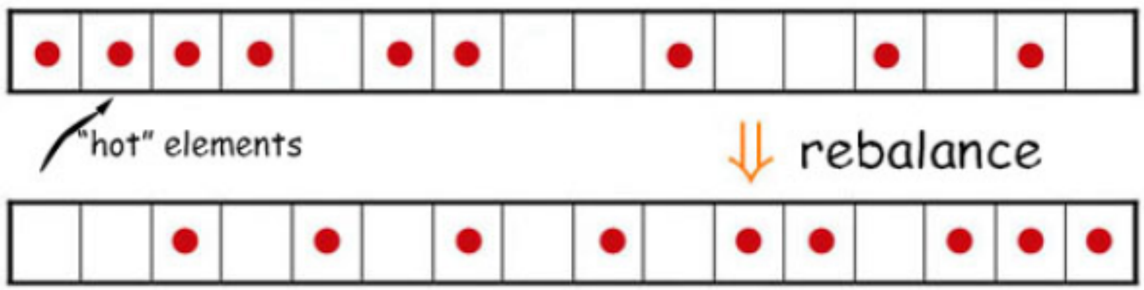
To guarantee $O(\log^2 N)$, we only need....

Rebalance Property: After a rebalance involving v, v is within parent u's density threshold.



Summary: As long as v is within u's threshold, it can be sparser or denser than t's density thresholds.

Sequential Insert: O(logN) Amortized moves



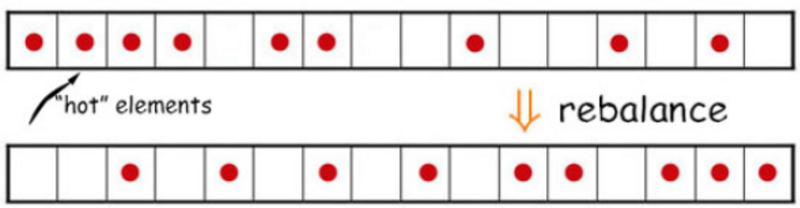
- Rebalance ***unevenly***, but maintain rebalance property.
- If hot elements are in front of array, push elements to end as far right as allowed.

Only large rebalances have lots of slop.
(Surprising to me that APMA works, since most rebalances are small.)

How to Remember Hot Elements Adaptively

- Maintain an $O(\log N)$-sized **predictor**, which keeps track of PMA regions with recent inserts
  - $O(\log N)$ counters, each up to $O(\log N)$.
  - Remembers up to $O(\log N)$ hotspot elements.
  - Tolerates "random noise" in inputs.

- (Generalization of how to find majority element in an array with a single counter.)

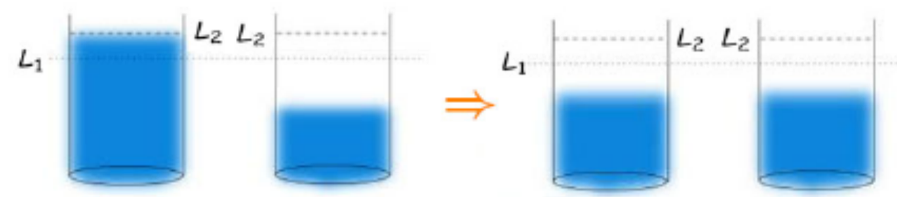


- Rebalance to even out weight of counters, while maintaining rebalance property.

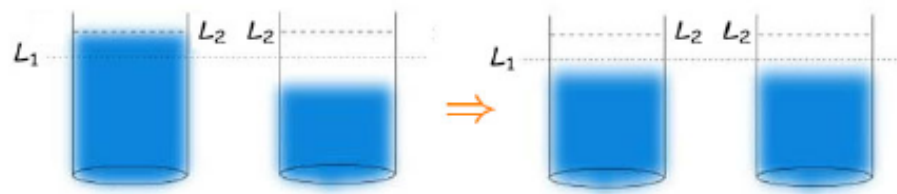
O(log log N) Even Rebalances Trigger a Larger Even Rebalance

1st rebalance,
cups at $L_2/2$:

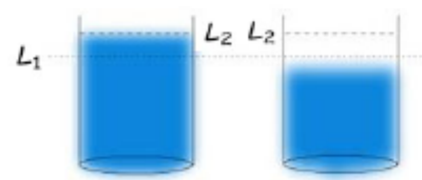
2nd rebalance,
cups at $3L_2/4$:



3rd rebalance,
cups at $7L_2/8$:



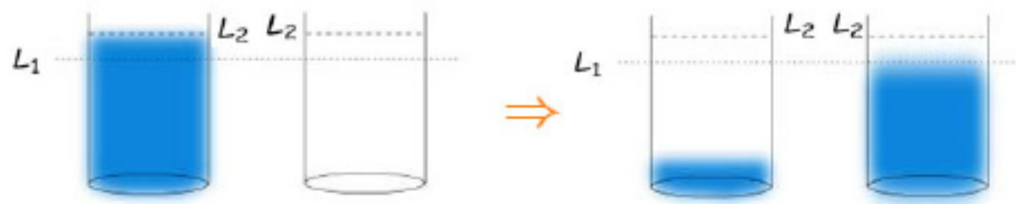
kth rebalance,
cups at $(2^k - 1)L_2/2^k$:



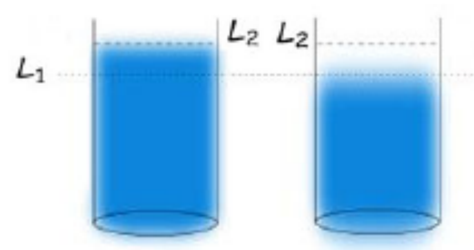
Solve $(2^k - 1)L_2/2^k \geq L_1$, where $L_2 - L_1 = \Theta(1/\log N)$.

$O(1)$ Uneven Rebalances Trigger a Larger Uneven Rebalance

1st rebalance,
cups at $L_2 - L_1$ and L_1 :



2nd rebalance,
cups at L_1 and L_2 (done):



Summary

- Insertion sort with gaps
  - LibrarySort [Bender,Farach,Colton,Mosteiro '04] (+ Wikipedia entry)
- Worst-possible inserts
  - PMA [Bender,Demaine,Farach-Colton '00,'05]
  - Cache-oblivious B-trees and other data structures
- Adapt to common distributions
  - APMA [Bender,Demaine,Farach-Colton '00,'05]
- Implementation of cache-oblivious data structures
  - Tokutek

- Is it practical to keep data physically in order in memory/on disk?