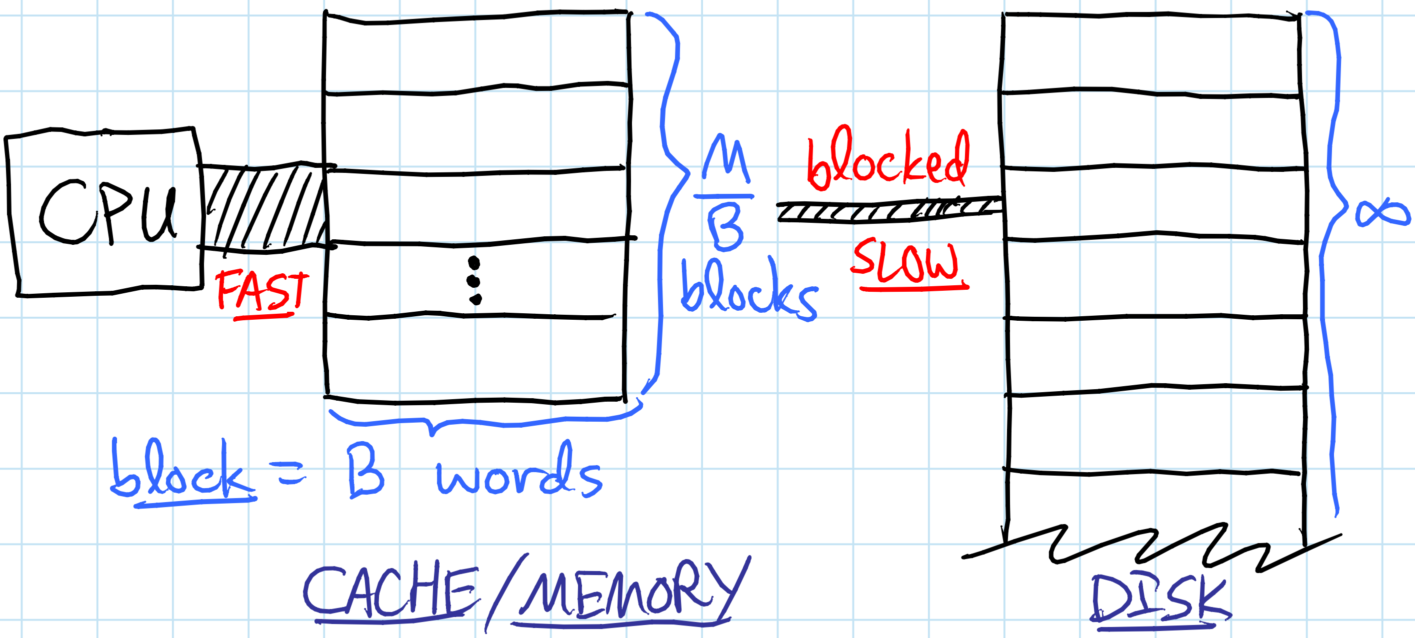
6.851	Lecture 7	Mar. 8, 2012

TODAY: Memory Hierarchies I (of 3)
- external-memory model
- cache-oblivious model
- cache-oblivious B-trees

External memory / I/O / Disk Access Model:

[Aggarwal & Vitter - CACM 1988]

two-level memory hierarchy

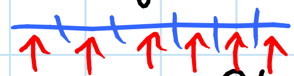


- focus on # memory transfers: blocks read/written between cache & disk
  - ≤ RAM running time
  - ≥ $\frac{\text{cell-probe LB}}{B}$
  - when can we save this factor of ≥ B?

Basic results in external memory:

(0) <u>Scanning</u>: $O(\lceil \frac{N}{B} \rceil)$ to read/write N words in order

(1) <u>Search trees</u>:

- B-trees with branching factor $\Theta(B)$ support insert, delete, predecessor search in $O(\log_{B+1} N)$ memory transfers (& $O(\lg N)$ time, with care, in comparison model)
- $\Omega(\log_{B+1} N)$ for search in comparison model:
  - where query fits among N items requires $\lg(N+1)$ bits of information 
  - each block read reveals where query fits among B items $\Rightarrow \leq \lg(B+1)$ bits of info.

  $\Rightarrow$ need $\geq \frac{\lg(N+1)}{\lg(B+1)}$ memory transfers
- also optimal in "block-probe model" if $B \geq w$ [Pătraşcu & Thorup — see L11]

(2) <u>Sorting</u>: $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers

↳ $B\times$ faster than B-tree sort!

Ω(ditto) in comparison model

(3) <u>Permuting</u>: $O(\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\})$

physical execution

Ω(ditto) in <u>indivisible model</u>

↳ can't pack pieces of input words in words

(4) <u>Buffer tree</u>: $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ amortized mem. transf. for delayed queries & batched updates & $O(0)$ find-min ($\Rightarrow$ priority queues)

Cache-oblivious model: [Frigo, Leiserson, Prokop, Ramachandran — FOCS 1999; Prokop — MEng 1999]

6.046

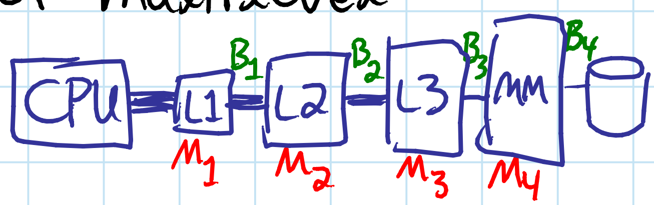
- like external-memory model
- but algorithm doesn't know B or M (!)

⇒ must work for <u>all</u> B & M

- automatic block transfers triggered by word access with offline optimal block replacement
  - FIFO, LRU, or any conservative replacement is 2-competitive given cache of 2× size (<u>resource augmentation</u>)
  - dropping M ↘ M/2 doesn't affect typical bounds e.g. sorting bound

Cool:

- clean model: algorithm just like RAM
- adapts to changing B (disk tracks & cache) & M (competing processes)
  - OPEN: formalize this for B
  - changing M formalized in [Bender, Ebrahimi, Fineman, Ghasemiesfeh, Johnson, McCauley — SODA 2014; Bender, Demaine, Ebrahimi, Fineman, Johnson, Lincoln, Lynch, McCauley — SPAA 2016]
- adapts to all levels of multilevel memory hierarchy:



- often possible!

Basic cache-oblivious results:

Ø Scanning: same (algorithm & bound)

* (1) Search trees: insert, delete, & search in $O(\log_{B+1} N)$ memory transfers } TODAY & L8

[Bender, Demaine, Farach-Colton — FOCS 2000/SICOMP 2005]
[Bender, Duan, Iacono, Wu — SODA 2002/J.Alg. 2004]
[Brodal, Fagerberg, Jacob — SODA 2002]

— best constant is lg e, not 1
[Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz — FOCS 2003]

(2) Sorting: $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers
[Frigo et al. 1999; Brodal & Fagerberg — ICALP 2002]

— uses tall-cache assumption: $M = \Omega(B^{1+\varepsilon})$

— impossible otherwise [Brodal & Fagerberg — STOC 2003]

(3) Permuting: min impossible [Brodal & Fagerberg — same]

* (4) Priority queue: $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ amortized mem. transf.

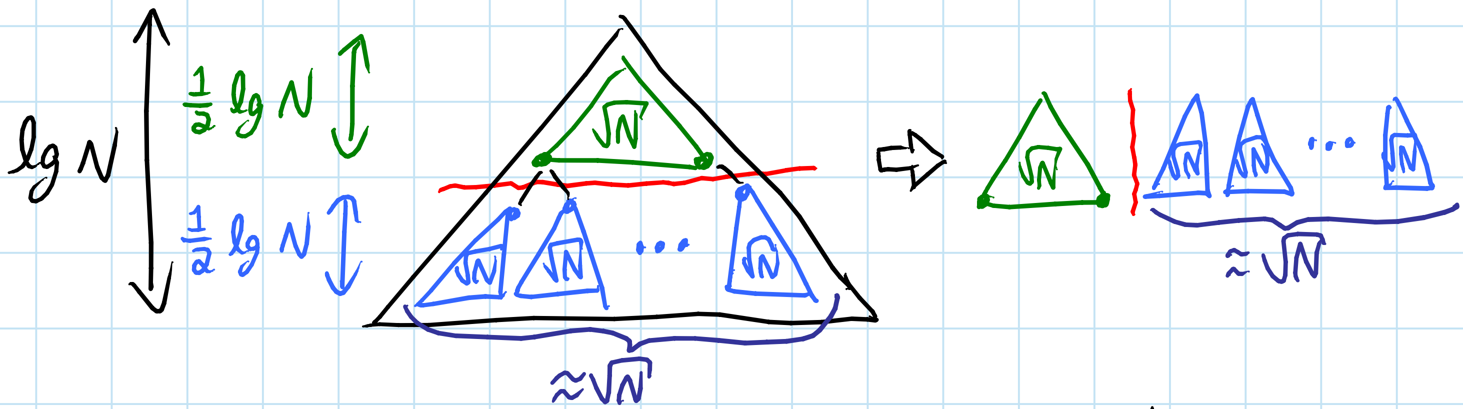
— uses tall-cache assumption (L8)

[Arge, Bender, Demaine, Holland-Minkley, Munro — STOC 2002/SICOMP 2007; Brodal & Fagerberg — ISAAC 2002]

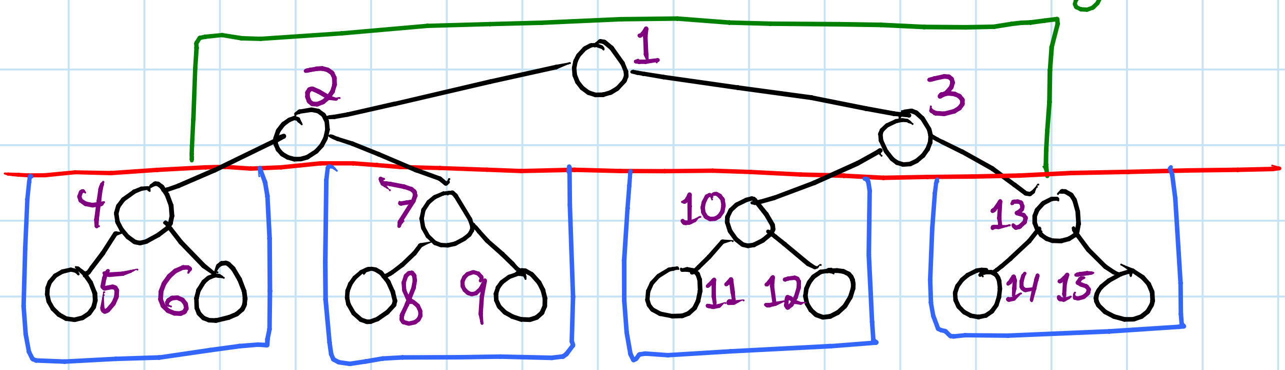
Cache-oblivious static search trees:
(binary search) [Prokop - MEng 1999]

- store N elements in N-node complete BST
- carve tree at middle level of edges

$\Rightarrow$ one top piece, $\approx \sqrt{N}$ bottom pieces, each size $\approx \sqrt{N}$



- recursively lay out pieces & concatenate: (in any order)



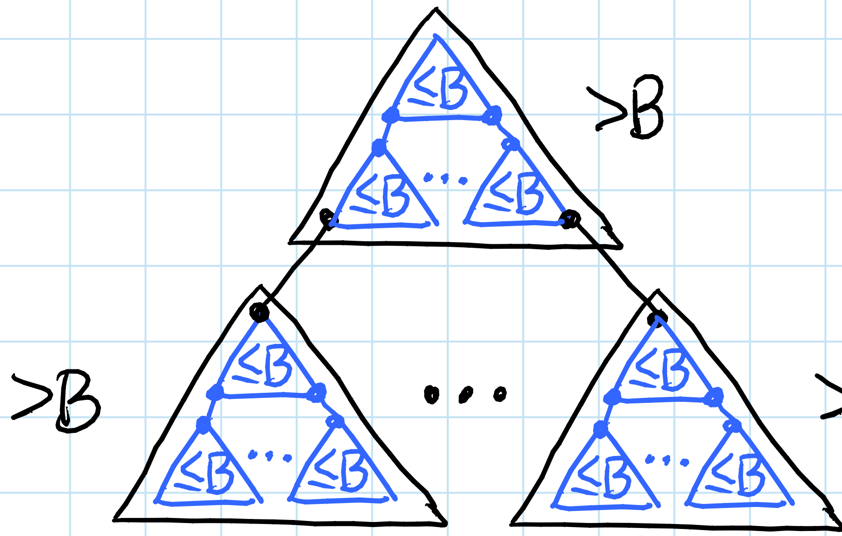
$\Rightarrow$ order to store nodes

"van Emde Boas layout"

- generalizes to [Bender, Demaine, Farach-Colton 2000]
  - height not a power of 2
  - node degrees ≥ 2 & $O(1)$

Analysis:

- level of detail (refinement) straddling B:



- cutting height in half until piece size ≤B
⇒ height of piece between $\frac{1}{2}\lg B$ & $\lg B$ (sloppy)
(⇒ size between $\sqrt{B}$ & B)
⇒ # pieces along root-to-leaf path $\leq \frac{\lg N}{\frac{1}{2}\lg B} = 2\log_B N$
- each piece stores ≤B elements consecutively
⇒ occupies ≤2 blocks (depending on alignment)
⇒ # memory transfers $\leq 4\log_B N$ (assuming $M \geq 2B$)
(really should be B+1)

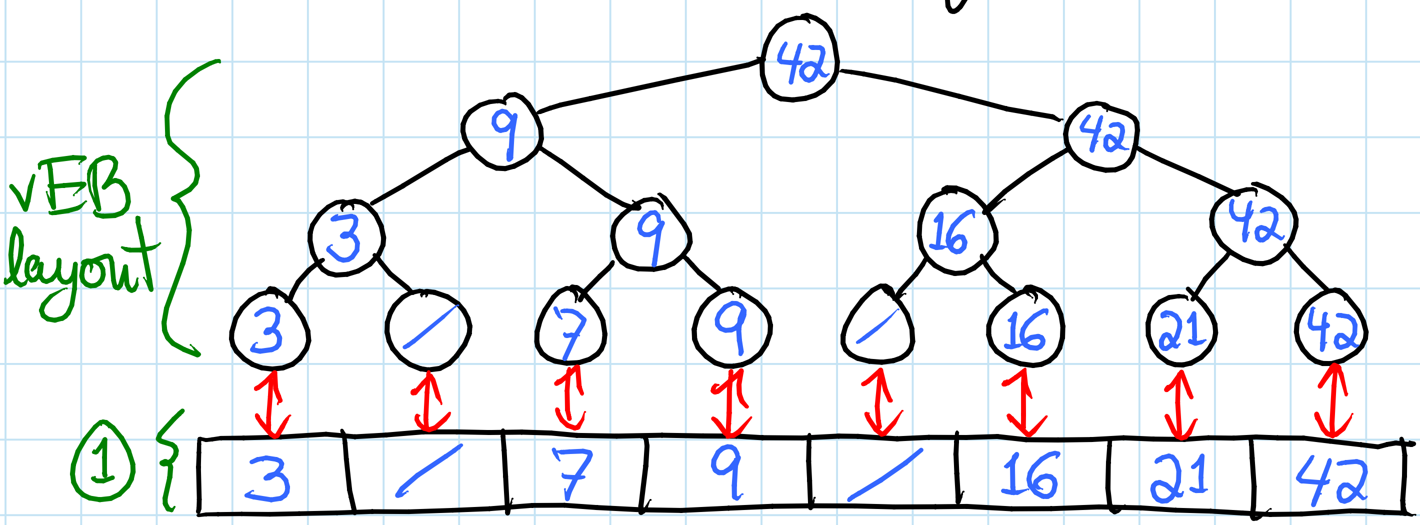
Improvements: [BBFGHHIL 2003]

① randomize starting location (w.r.t. block)
⇒ expected cost $\leq (2 + \frac{3}{\sqrt{B}})\log_B N$
② split height into $\frac{1}{2}-\varepsilon : \frac{1}{2}+\varepsilon$ ratio
⇒ expected cost $\leq (\lg e + o(1))\log_B N$
$\approx O(\lg\lg B / \lg B)$

Cache-oblivious B-trees as in [Bender, Duan, Iacono, Wu]

① <u>ordered file maintenance</u>: (to do in L8)
store N elements in specified order
in an array of size $O(N)$ with $O(1)$ gaps
— <u>updates</u>: insert element between two given
delete element
by re-arranging array interval of $O(\lg^2 N)$ am.

② build static search tree on top:
each node stores max key in subtree (if any)



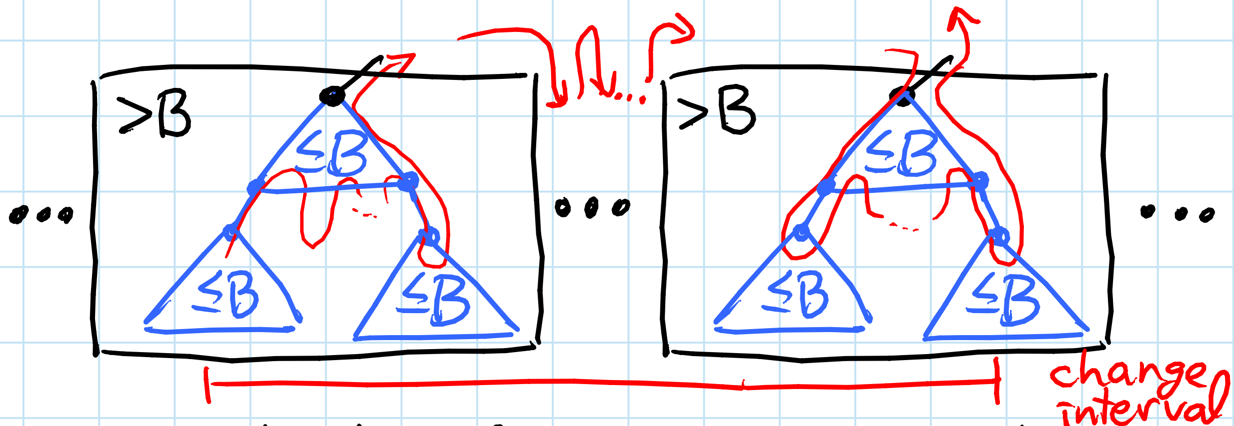
③ <u>operations</u>:
at most double →

— binary search via left child's key
— insert(x) finds predecessor & successor,
inserts there in ordered file,
& updates leaves & max's up tree
via postorder traversal
— delete similar

④ <u>update analysis</u>:

if K cells change in ordered file then update tree in $O(\frac{K}{B} + \log_B N)$ mem. tr.

- look at level of detail straddling B
- look at bottom two levels:



- within <u>chunk</u> of >B, jumping between ≤2 <u>pieces</u> of ≤B (assume $M \geq 4B$)

⇒ O(chunk/B) memory transfers in chunk
↳ portion in update interval +3 maybe (first, last, & root)

⇒ $O(\frac{K}{B})$ memory transfers in bottom 2 levels

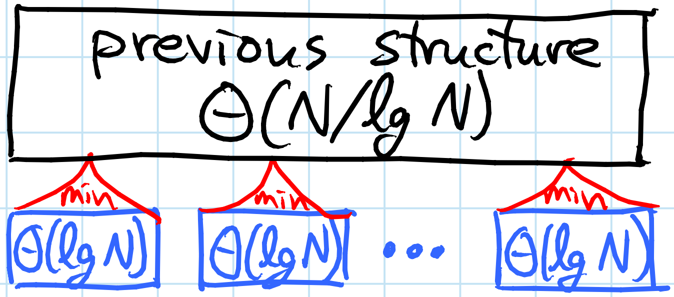
- updated nodes above these two levels:
  - subtree of $\leq \frac{K}{B}$ chunk roots up to their LCA: costs $O(\frac{K}{B})$
  - path from LCA to root of tree: costs $O(\log_B N)$ as above

⇒ $O(\frac{K}{B} + \log_B N)$ total memory transfers

<u>So far</u>: search in $O(\log_B N)$
update in $O(\log_B N + \frac{\lg^2 N}{B})$ amortized ← bad if $B = o(\lg N \lg\lg N)$

(5) indirection:

- cluster elements into $\Theta(\frac{N}{\lg N})$ groups, each of size $\Theta(\lg N)$
- use previous structure on min's of clusters



- update cluster by complete rewrite $\Rightarrow O(\frac{\lg N}{B})$ memory transfers
- split/merge clusters as necessary to keep between 25% & 100% full $\Rightarrow \Omega(\lg N)$ updates to charge to

$\Rightarrow O(\frac{\lg^2 N}{B})$ update cost in top structure only "every" $\Omega(\lg N)$ actual updates

$\Rightarrow$ amortized update cost $O(\frac{\lg N}{B})$ (plus search cost)

Finally: $O(\log_B N)$ insert, delete, predecessor, successor just like B-trees in external mem. (known B)