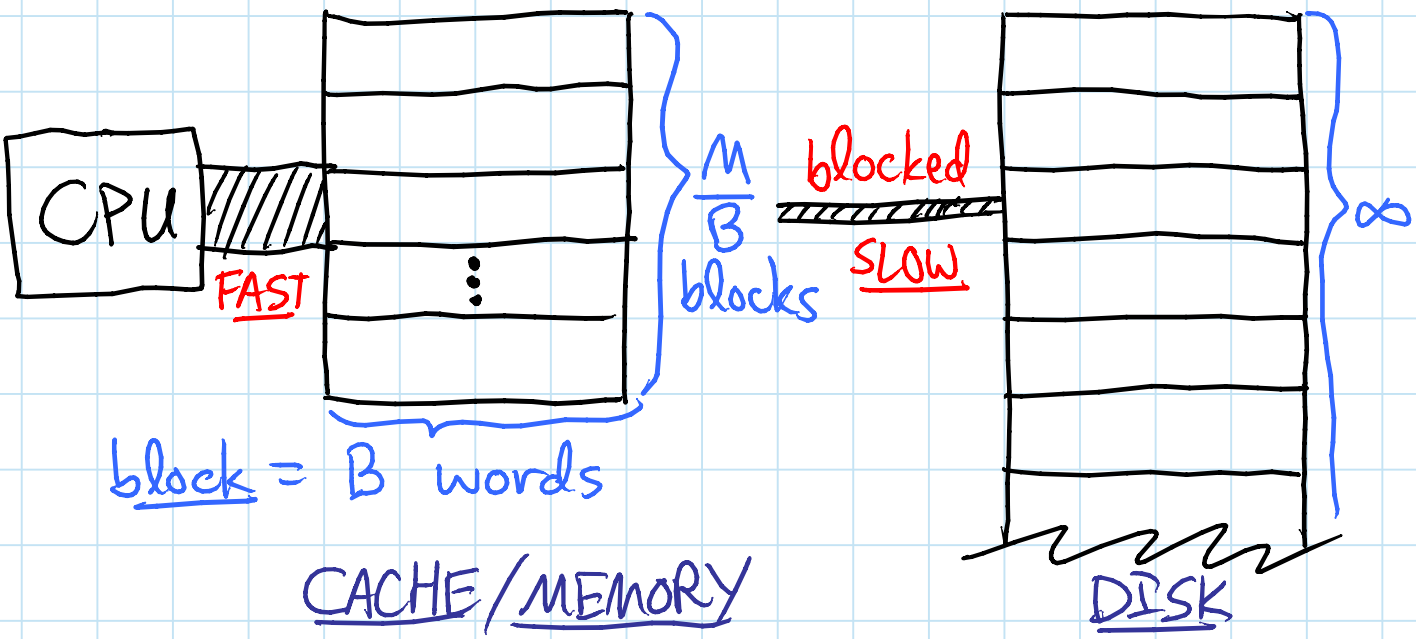
6.851	Lecture 20	Apr. 22, 2010

TODAY: memory hierarchies
- 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

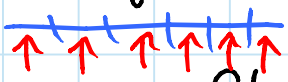


- focus on # memory transfers: blocks read/written between cache & disk
  - $\leq$ RAM running time
  - $\geq \dfrac{\text{cell-probe LB}}{B}$
- when can we save this factor of $\geq B$?

Basic results in external memory:

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

(1) Search trees:

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

(2) Sorting: $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers
↳ $B\times$ faster than B-tree sort!
Ω(ditto) in comparison model

(3) Permuting: $O(\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\})$
physical execution
Ω(ditto) in indivisible model
↳ can't pack pieces of input words in words

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

Cache-oblivious model: [Frigo, Leiserson, Prokop, Ramachandran – FOCS 1999; Prokop – MEng 1999] 6.046

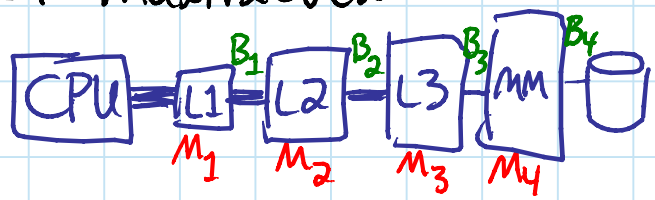
- like external-memory model
- but algorithm doesn't know B or M (!)

$\Rightarrow$ 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 $M \searrow M/2$ 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)
  - $\boxed{\text{OPEN}}$: formalize this
- 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 & L21

[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: $O(\frac{1}{B} \log_{M/B} \frac{1}{B})$ amortized mem. transf.

— uses tall-cache assumption L21

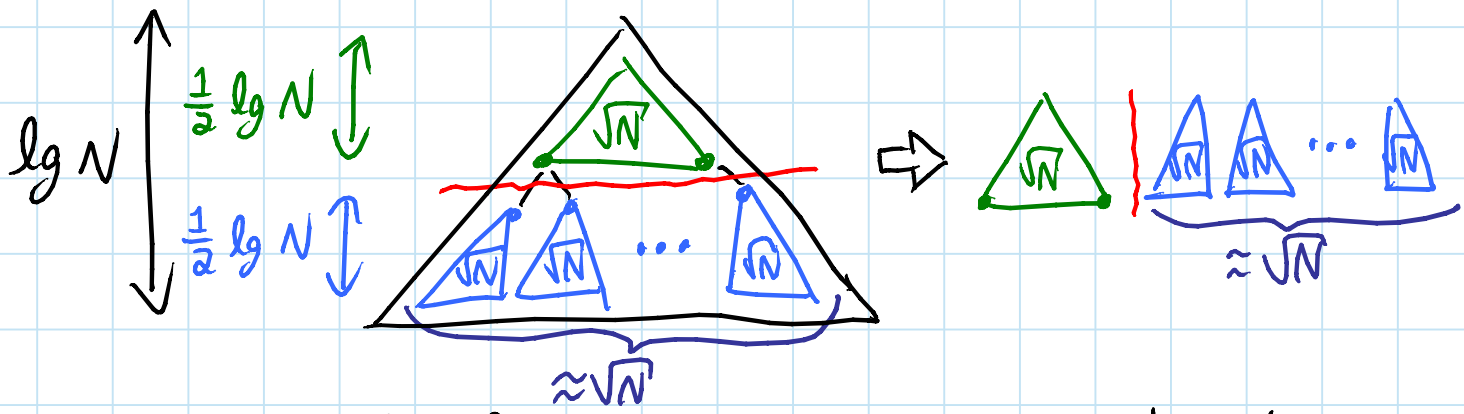
[Arge, Bender, Demaine, Holland-Minkley, Munro — STOC 2002/SICOMP 2007; Brodal & Fagerberg — ISAAC 2002]

Cache-oblivious static search trees:

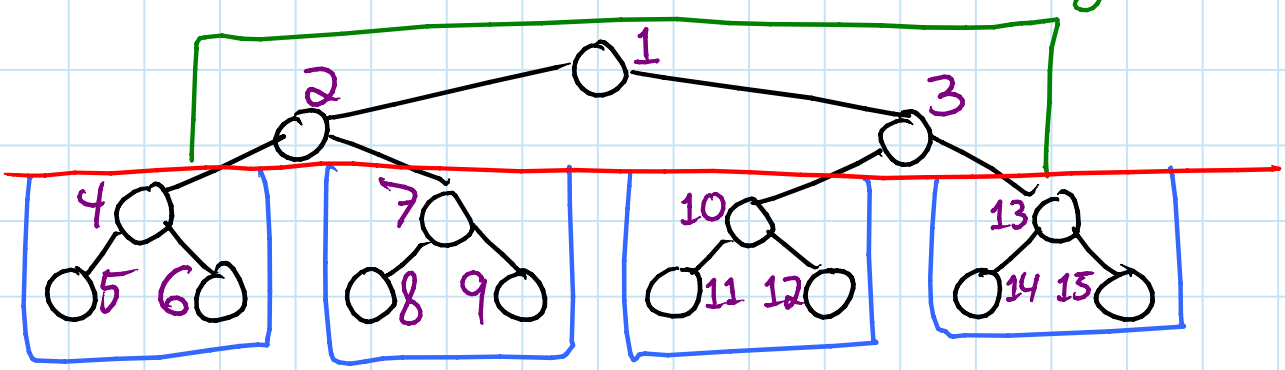
(binary search) [Prokop–MEng 1999]

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

$\Rightarrow \approx\sqrt{N}+1$ pieces of size $\approx\sqrt{N}$



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



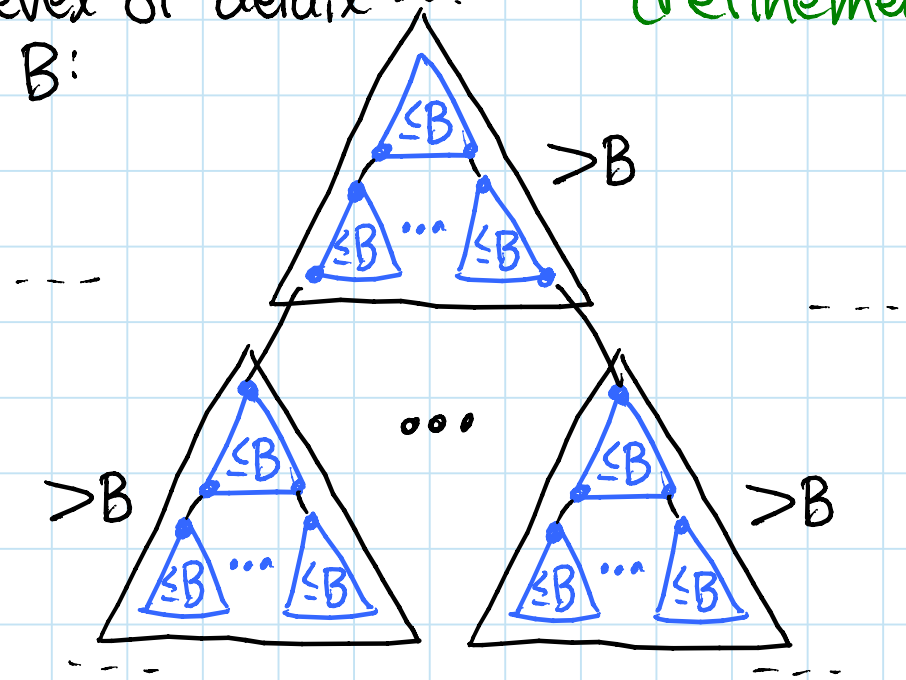
$\Rightarrow$ order to store nodes

"van Emde Boas layout"

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

Analysis of van Emde Boas layout:

- consider level of detail ... straddling B: (refinement)



- cutting height in half until $\leq \lg B$ → chunks

⇒ chunks have height between $\frac{1}{2}\lg B$ & $\lg B$

(⇒ size between $\sqrt{B}$ & B)

⇒ # chunks visited on root-to-leaf path $\leq \dfrac{\lg N}{\frac{1}{2}\lg B} = 2\log_B N$

(assume $B \geq 2 \Rightarrow \lg B \geq 1$)

- each chunk stores $\leq B$ words consecutively

⇒ occupies ≤ 2 blocks (depending on alignment)

⇒ # memory transfers $\leq 4 \log_B N$

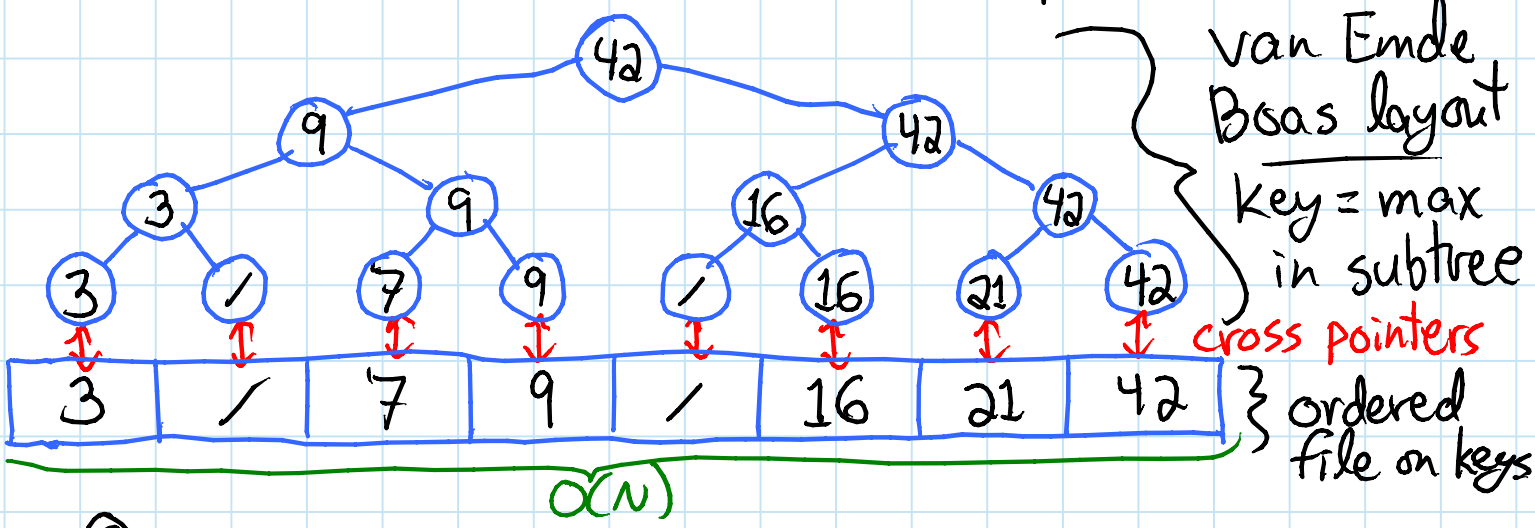
(assuming $M \geq 2B$)

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

① ordered file maintenance: (to do in L21)

- store N elements in specified order in an array of size $O(N)$ ~ allow gaps
- updates: delete element;
  insert element between two specified elts.
  by moving elements in array interval of $O(\lg^2 N)$ amortized via $O(1)$ interleaved scans

② build static search tree on top:

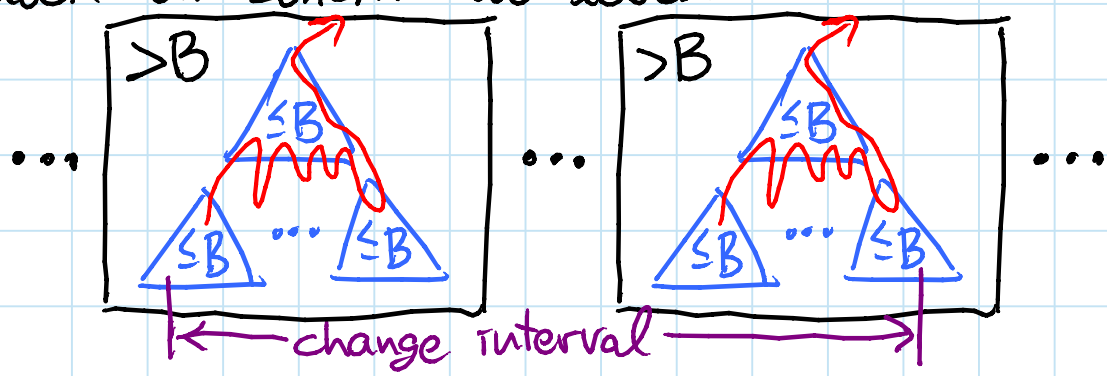


③ ops:

- search looks at left child's key to decide L/R
  - still $O(\log_{B+1} N)$ ← at most double
- insert(x):
  - search(x) to find predecessor & successor
  - insert x in between in ordered file
  - update values in leaves corresp. to changed cells & propagate changes up tree in postorder traversal
- delete(x) similar

④ <u>update analysis</u>: if K cells change in ordered file then $O(\frac{K}{B} + \log_{B+1} N)$ memory transfers

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



- within <u>superchunk</u> of $>B$, jumping between ≤ 2 chunks of $\leq B$ ~assume $M \geq 2B$

$\Rightarrow O(\lceil \frac{\text{superchunk}}{B} \rceil)$ memory transfers per superchunk

↳ actually visited portion

↳ unnecessary because superchunk $>B$, except first & last

$\Rightarrow O(\frac{K}{B} + 1)$ memory transfers in bottom two levels

- # ancestors above these two levels

$$\leq \underbrace{\underbrace{\frac{K}{B}}_{\text{ancestors to lca}}}_{\text{cost} \leq 1 \text{ each}} + \underbrace{\underbrace{\lg N}_{\text{path to root}}}_{\log_{B+1} N \text{ cost as in search}}$$

$\Rightarrow O(\frac{K}{B} + \log_{B+1} N)$ total memory transfers

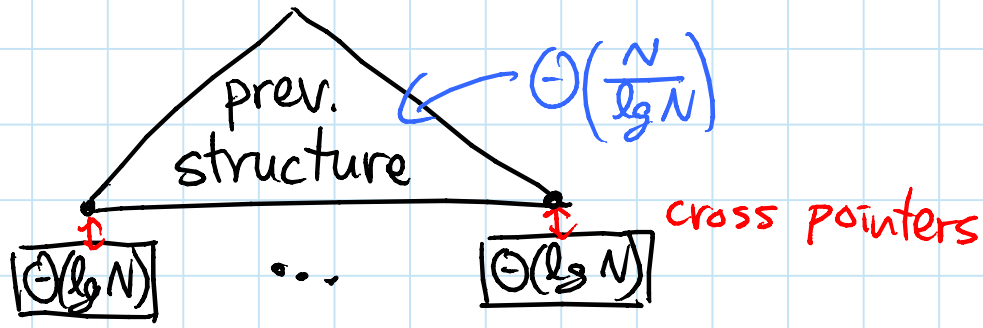
<u>So far</u>: search in $O(\log_{B+1} N)$

update in $O(\log_{B+1} N + \underbrace{\frac{\lg^2 N}{B}})$ amortized

suboptimal if $B = o(\lg N \lg\lg N)$

⑤ <u>indirection:</u>

- divide elements into clusters of $\Theta(\lg N)$
- use previous structure on min(each cluster)



- search: vEB top, scan bottom
  $\Rightarrow O\left(\log_{B+1} N + \frac{\lg N}{B}\right) = O(\log_{B+1} N)$
- update cluster by complete rewrite $\Rightarrow O\left(\frac{\lg N}{B}\right)$
- keep clusters between 25% & 100% full
- split / merge & split <u>when necessary</u> (like B-tree)
  $\Rightarrow \Omega(\lg N)$ updates to change to

$\Rightarrow O(1)$ updates in top structure
only every $\Omega(\lg N)$ updates

$$\Rightarrow \text{amortized update cost} = O\left(\frac{\log_B N + \frac{\lg^2 N}{B}}{\lg N}\right)$$

$$= O\left(\frac{\lg N}{B}\right)$$

$$\text{plus search cost} = O(\log_B N)$$

<u>So</u>: $O(\log_{B+1} N)$ insert, delete, search