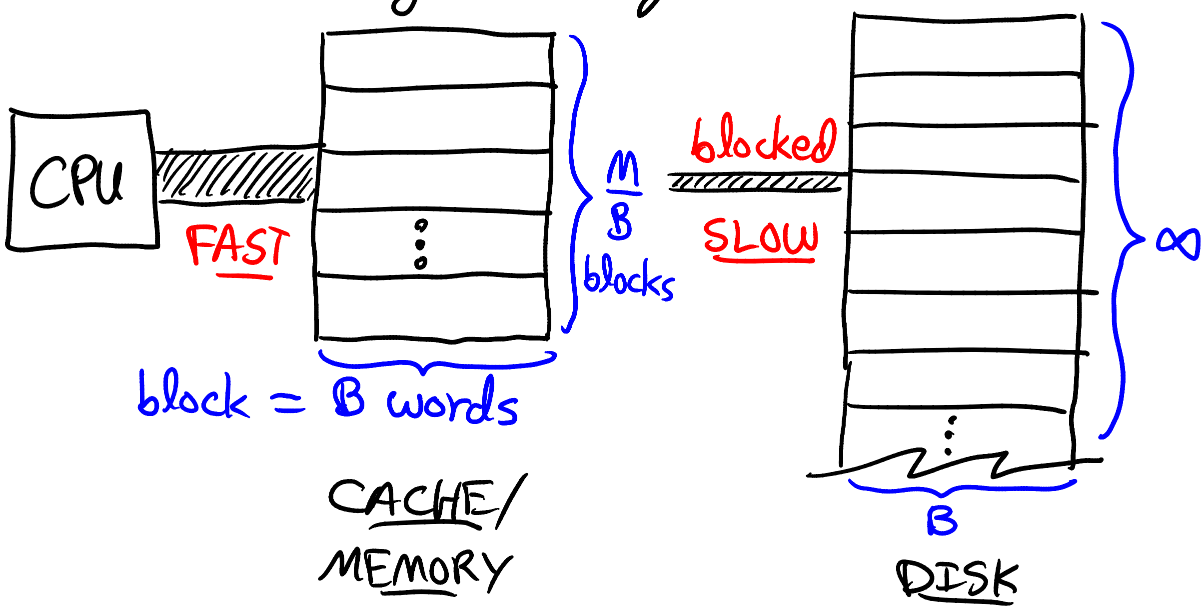
Lecture 19

External memory / I/O / Disk Access Model [Aggarwal & Vitter—CACM 1988]

- two-level memory hierarchy:



- charge mainly for memory transfers: blocks read/written between cache & disk
- any algorithm with running time $T(N)$ uses $\le T(N)$ memory transfers
- when can we use fewer? (& still $T(N)$ time)

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 in comparison model)
   - this is optimal 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 $\Rightarrow \le \lg(B+1)$ bits of information

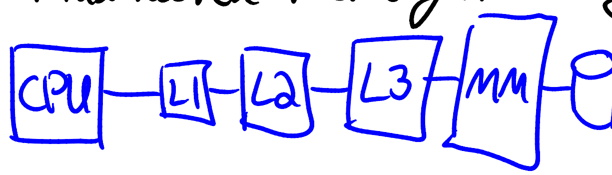
     $\Rightarrow$ need $\geq \frac{\lg(N+1)}{\lg(B+1)}$ memory transfers
   - also optimal in "block-probe model" if $B \geq w$ [Lecture 15]
2. Sorting: $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ memory transfers
   ↳ $\geq B\times$ faster than B-tree sort!
   $\Omega(\text{ditto})$ in comparison model
3. Permutation: $O(\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\})$
   physical execution
   $\Omega(\text{ditto})$ in "indivisible model" — can't pack pieces of input words into word
4. Buffer tree: $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ amortized memory transfers for delayed queries / batched updates & instant delete-min $\Rightarrow$ priority queues

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

- like external-memory model
- but algorithm doesn't know B or M (!)
- automatic block transfers triggered by word access with offline optimal block replacement
  - FIFO, LRU, or any conservative replace. strategy is 2-competitive given cache of 2× size
  - dropping $M \searrow M/2$ doesn't affect bounds like

Cool:

- clean model: algorithm just like RAM alg.
- adapts to changing B (disk tracks)
  informally...   & M (competing processes)
- adapts to all levels of multilevel memory hierarchy each with own B & M



- often possible!

Basic cache-oblivious results:

0. Scanning: same algorithm & bound
* 1. "B-tree": insert, delete, & search in $O(\log_{B+1} N)$ memory transfers } TODAY & L20

   [Bender, Demaine, Farach-Colton — FOCS 2000/SICOMP 2005; Bender, Duan, Iacono, Wu — SODA 2002; 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 is impossible [Brodal & Fagerberg — same]

4. Priority queue: $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ amortized mem. transfers (also uses tall-cache assumption)

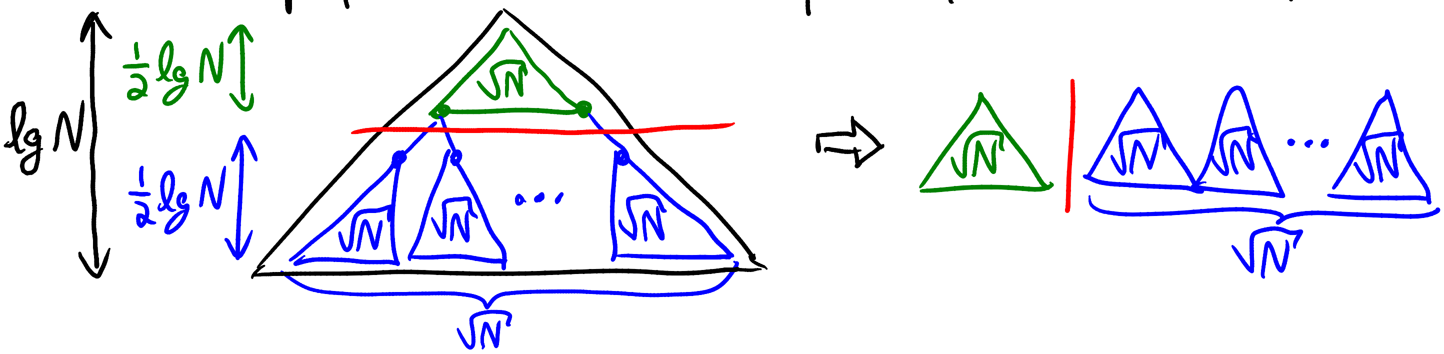
   [Arge, Bender, Demaine, Holland-Minkley, Munro — STOC 2002 / SICOMP 2007; Brodal & Fagerberg — ISAAC 2002]

Cache-oblivious binary search / static search trees: van Emde Boas layout

[Prokop – MEng 1999; Bender, Demaine, Farach-Colton – FOCS 2000]

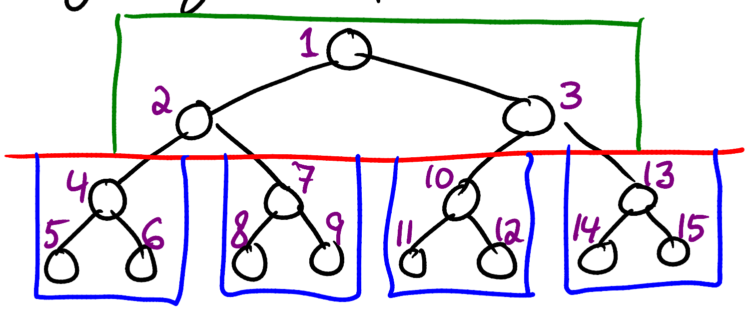
- store N elements in order in N-node complete BST
- carve tree at middle level of edges

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



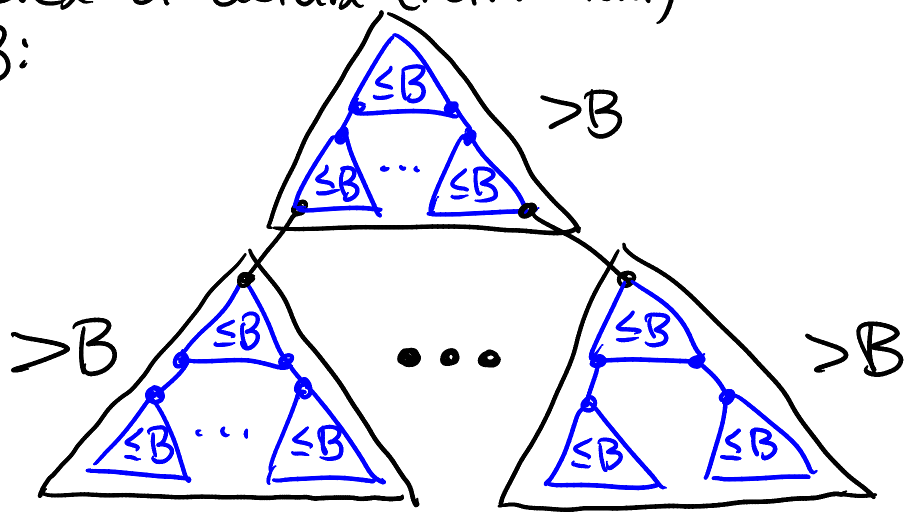
- recursively lay out pieces & concatenate

e.g:



Analysis of van Emde Boas layout:

- consider level of detail (refinement) straddling B:



- cutting height in half until $\le \lg B$ → <ins>pieces</ins>

$\Rightarrow$ <ins>pieces</ins> have height between $\frac{1}{2}\lg B$ & $\lg B$ (& size between $\sqrt{B}$ & B)

- # <ins>pieces</ins> visited on root-to-leaf path $\le \dfrac{\lg N}{\frac{1}{2}\lg B} = 2\log_B N$

(sloppy on B+1 issue)

- each piece stores $\le B$ elements consecutively

$\Rightarrow$ occupies ≤ 2 blocks (depending on alignment)

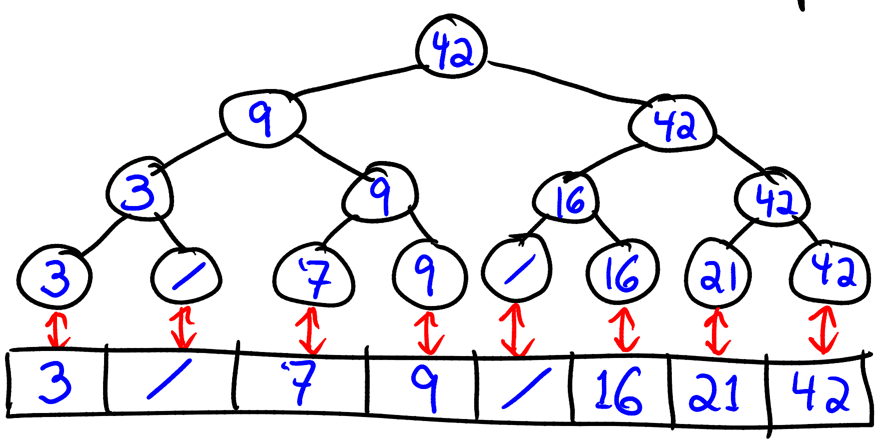
$\Rightarrow$ # memory transfers $\le 4\log_B N$ (assuming $M \ge 2B$)

Generalizations:

- height not a power of 2
- node degrees ≥ 2 & $O(1)$

Cache-oblivious B-trees, as in [Bender, Duan, Iacono, Wu — SODA 2002]

1. ordered file maintenance:
   - store N elements in specified order in array of size $O(N)$ (allow gaps)
   - updates: insert element between two specified / delete element by moving elements in array interval of $O(\lg^2 N)$ amortized
   - black box to be filled [Lecture 20]
2. build static search tree on top:



static search tree (van Emde Boas layout), key = max(subtree)

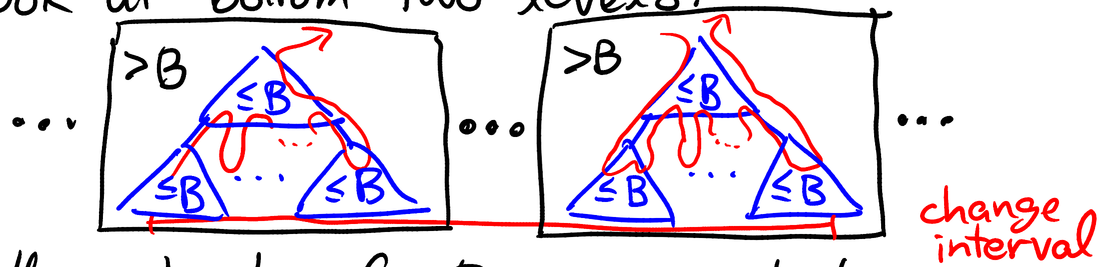
← cross-pointers

← ordered file maintenance on all keys

3. operations:
   - search looks at left child's key to decide direction
   - insert(x):
     - search(x) finds 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 $O(\frac{K}{B} + \log_B N)$ mem. transf.

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



- within chunk of >B, jumping between ≤2 subchunks of ≤B ~ assume $M \geq 2B$

$\Rightarrow O(\text{chunk}/B)$ memory transfers per chunk

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

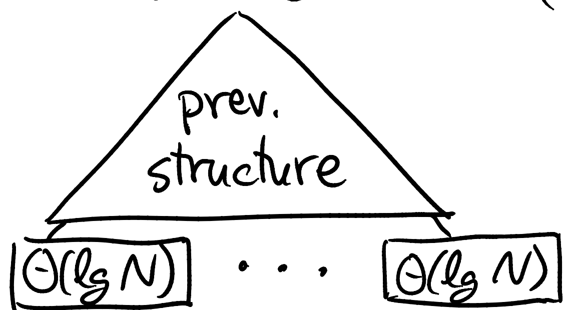
- # nodes above these two levels $\leq \underbrace{\frac{K}{B}}_{\text{ancestors to lca cost} \leq 1 \text{ each}} + \underbrace{\lg N}_{\text{path to root } \log_B N \text{ cost as before}}$

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

<u>So far</u>: search in $O(\log_{B+1} N)$
update in $O(\log_{B+1} N + \underbrace{\frac{\lg^2 N}{B}}_{\text{suboptimal if } B = o(\lg N \lg\lg N)})$ amortized

⑤ indirection:

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



- update cluster by complete rewrite ~ $O(\frac{\lg N}{B})$
- keep cluster between 25% & 100% full
- split / merge & split when necessary

$\Omega(\lg N)$ updates to charge to

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

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

Conclusion: $O(\log_{B+1} N)$ insert, delete, search