External memory I/O Disk Access Model — two-level memory hierarchy:

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

[Aggarwal & Vitter—CACM 1988]
Basic results in external memory:

1. Scanning: $O(\frac{N}{B})$ to read/write $N$ words in order

2. Search trees:
   - $B$-trees with branching factor $O(B)$
     - support insert, delete, (predecessor) search
     - $O(\log_{B+1} N)$ memory transfers
     - $O(\log N)$ time in comparison model
   - this is optimal for search in comparison model:
     - where query fits among $N$ items requires
       \[ \log(N + 1) \] bits of information
     - each block read reveals where query fits among $B$ items \( \Rightarrow \leq \log(B+1) \) bits of information
     - \( \Rightarrow \text{need} \geq \frac{\log(N+1)}{\log(B+1)} \) memory transfers
   - also optimal in "block-probe model" if $B \geq w$ [Lecture 15]

3. Sorting: $O(\frac{N}{B} \log_{MB} \frac{N}{B})$ memory transfers
   \( \Rightarrow \geq B \times \) faster than $B$-tree sort!
   - $\Omega(\text{ditto})$ in comparison model

4. Permutation: $O(\min(\frac{N}{B} \log_{MB} \frac{N}{B}, \frac{N}{3}))$
   - physical execution $\Omega(\text{ditto})$ in "indivisible model" —
     - can't pack pieces of input words into word

4. Buffer tree: $O(\frac{1}{B} \log_{MB} \frac{N}{B})$ 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 2x size
  - dropping M ≥ 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:

1. Scanning: same algorithm & bound
   ① "B-tree": insert, delete, & search
   in $O(\log_{B+1} N)$ memory transfers
   [Bender, Demaine, Farach-Colton – FOCS 2000/STOC 2001;
    Bender, Duan, Iacono, Wu – SODA 2002;
    Brodal, Fagerberg, Jacob – SODA 2002]
   – best constant is $\leq e$, not 1
   [Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz – FOCS 2003]

2. Sorting: $O\left(\frac{N}{B} \log_{MB} \frac{N}{B}\right)$ memory transfers
   – 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: $O\left(\frac{1}{B} \log_{MB} \frac{N}{B}\right)$ amortized mem. transfers
   (also uses tall-cache assumption)
   [Arge, Bender, Demaine, Holland-Minkley, Munro – STOC/SICOMP 2002;
    Brodal & Fagerberg – ISAAC 2002]
Cache-oblivious binary search/static search trees:
van Emde Boas layout

- 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.

\[ \begin{align*}
\log N & \quad \frac{\log N}{2} \\
\frac{\log N}{2} & \quad \frac{\log N}{2}
\end{align*} \]
Analysis of van Emde Boas layout:
- consider level of detail (refinement)

straddling B:

- cutting height in half until $\leq \log B \rightarrow$ pieces

$\Rightarrow$ pieces have height between $\frac{1}{2} \log B$ & $\log B$

($\&$ size between $\sqrt{B}$ & $B$)

- # pieces visited on root-to-leaf path $\leq \frac{\log N}{\frac{1}{2} \log B}$

(sloppy on $B+1$ issue)

- each piece stores $\leq B$ elements consecutively

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

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

(assuming $M \geq 2B$)

Generalizations:
- height not a power of 2
- node degrees $\geq 2$ & $O(1)$
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 $O(\log^2 N)$ amortized
   - black box to be filled [Lecture 20]

2. build static search tree on top:

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. **update analysis**: if $K$ cells change in ordered file then $O\left(\frac{K}{B} + \log_B N\right)$ mem. transf.
   - look at level of detail straddling $B$
   - look at bottom two levels:
     
     \[ \begin{array}{c}
     \begin{array}{c}
     \begin{array}{c}
     >B \\
     \downarrow \\
     \leq B \\
     \downarrow \\
     \leq B
     \end{array}
     \end{array}
     \begin{array}{c}
     \begin{array}{c}
     >B \\
     \downarrow \\
     \leq B \\
     \downarrow \\
     \leq B
     \end{array}
     \end{array}
     \end{array} \]

     - within chunk of $>B$, jumping between $\leq 2$ subchunks of $\leq B$  \( \Rightarrow \) assume $M \geq 2B$
     \( \Rightarrow O(\text{chunk}/B) \) memory transfers per chunk
     \( \Rightarrow O\left(\frac{K}{B}\right) \) memory transfers in bottom two levels
     - # nodes above these two levels $\leq \frac{K}{B} + \log_B N$

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

**So far**: search in $O(\log_{B+1} N)$
update in $O(\log_{B+1} N + \frac{\log^2 N}{B})$ amortized

suboptimal if $B=O(\log N \log \log N)$
indirection:
- cluster elements into $\Theta\left(\frac{N}{\log N}\right)$ groups of $\Theta(\log N)$
- use previous structure on $\min$(each cluster)

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

\[\Omega(\log N)\] updates to charge to

\[\Rightarrow\] update in top structure
only every \[\Omega(\log N)\] updates

\[\Rightarrow\] amortized update cost = $O\left(\frac{\log N}{B}\right)$
plus search cost

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