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
  \[ \leq \text{RAM running time} \]
  \[ \geq \frac{\text{cell-probe LB}}{B} \]
- when can we save this factor of \( \geq B \)?
Basic results in external memory:

0. Scanning: \(O(N/B)\) to read/write \(N\) words in order

1. Search trees:
   - B-trees with branching factor \(O(B)\)
     - Support insert, delete, predecessor search
     - \(O(\log_{B+1} N)\) memory transfers
     - \(\Omega(\log_{B+1} N)\) 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 info.
   - \(\Rightarrow\) Need \(\geq \frac{\log (N+1)}{\log (B+1)}\) memory transfers

   - Also optimal in “block-probe model” if \(B \geq w\)
     - [Pătraşcu & Thorup — see L.11]

2. Sorting: \(O(N/B \log w_B N/B)\) memory transfers
   - \(\Rightarrow B \times\) faster than B-tree sort!
   - \(\Omega(\text{ditto})\) in comparison model

3. Permuting: \(O(\min \{ \frac{N}{B}, \frac{N}{B \log w_B N/B} \})\)
   - \(\Omega(\text{ditto})\) in indivisible model

4. Buffer tree: \(O(\frac{1}{B} \log w_B N/B)\) amortized mem. transf.
   - For delayed queries & batched updates
   & \(O(\delta)\) delete-min (\(\Rightarrow\) priority queues)
- like external-memory model
- but algorithm doesn't know B or M ( ! )
⇒ must work for all 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 2x size (resource augmentation)
  - 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
- adapts to all levels of multilevel memory hierarchy:

- often possible!
Basic cache-oblivious results:

1. **Scanning**: same algorithm & bound
   - in $O(\log_{B+1} N)$ memory transfers
     - best constant is $\log e$, not 1
     - [Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz - FOCS 2003]

2. **Sorting**: $O\left(\frac{N}{B} \log M_B \frac{N}{B}\right)$ memory transfers
   - uses tall-cache assumption: $M = \Omega(\sqrt{B^{1+\varepsilon}})$
   - impossible otherwise [Brodal & Fagerberg - STOC 2003]

3. **Permuting**: min impossible [Brodal & Fagerberg - same]

4. **Priority queue**: $O\left(\frac{1}{B} \log M_B \frac{1}{B}\right)$ amortized mem. transf.
   - uses tall-cache assumption
     - [Arge, Bender, Demaine, Holland-Minkley, Munro - STOC 2002/SICOMP 2007; Brodal & Fagerberg - ISAAC 2002]
Cache-oblivious static search trees:

- Store $N$ elements in $N$-node complete BST
- Carve tree at middle level of edges
  $\Rightarrow \sim \sqrt{N}+1$ pieces of size $\sim \sqrt{N}$

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

$\Rightarrow$ Order to store nodes

- Generalizes to [Bender, Demaine, Farach-Colton 2000]
  - Height not a power of 2
  - Node degrees $\geq 2$ & $O(1)$
Analysis of van Emde Boas layout:

- Consider level of detail

Straddling B:

- Cutting height in half until $\leq \log B \Rightarrow$ chunks
- Chunks have height between $\frac{1}{2} \log B$ & $\log B$ ($\Rightarrow$ size between $\sqrt{B}$ & $B$)
- #chunks visited on root-to-leaf path $\leq \frac{\log N}{\frac{1}{2} \log B} = 2 \log_B N$ (assume $B \geq 2 \Rightarrow \log B \geq 1$)
- Each chunk stores $\leq B$ words consecutively
- Occupies $\leq 2$ blocks (depending on alignment)
- #memory transfers $\leq 4 \log_B N$ (assuming $M \geq 2B$)
Cache-oblivious B-trees as in [Bender, Duan, Tamano, Wu]

1. ordered file maintenance: (to do in L21)
   - store $N$ elements in specified order in an array of size $O(N)$
   - updates: delete element; insert element between two specified elts.
     by moving elements in array interval of $O(\log^2 N)$ amortized via $O(1)$ interleaved scans

2. build static search tree on top:

   Van Emde Boas layout
   - key = max in subtree
   - cross pointers
   - ordered file on keys

3. ops:
   - `search` looks at left child’s key to decide L/R
     - still $O(\log_b+1 N)$
   - `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) update analysis: 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:

   ![Diagram showing levels of detail with arrows indicating change intervals]

   - within superchunk of $B$, jumping between
     $\leq 2$ chunks of $\leq B$
   $\Rightarrow O(\frac{1}{B} \text{superchunk})$ memory transfers per superchunk
     - assume $M > 2B$
     - 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
     $\leq \frac{K}{B} + \log N$
     ancestors to LCA path to root
     cost $\leq 1$ each
   $\Rightarrow O(\frac{K}{B} + \log_{B+1} N)$ 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:
- divide elements into clusters of $\Theta(lg N)$
- use previous structure on $\text{min}(\text{each cluster})$

prev. structure
\[ \Theta(lg N) \]
\[ \Theta(lg N) \]

search: vEB top, scan bottom
\[ \Rightarrow O(lg_{B+1} N + \frac{lg N}{B}) = O(lg_{B+1} N) \]

update cluster by complete rewrite \[ \Rightarrow O(lg N) \]

keep clusters between 25% & 100% full

split / merge & split when necessary (like B-tree)
\[ \Rightarrow \Omega(lg N) \text{ updates to charge to} \]

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

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

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

plus search cost $= O(lg_{B} N)$

So: $O(lg_{B+1} N)$ insert, delete, search