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
  \( \leq \) RAM running time
  \( \geq \) cell-probe LB

- when can we save this factor of \( \geq B \)?
Basic results in external memory:

1. **Scanning**: $O(\frac{N}{B})$ to read/write $N$ words in order
   - B-trees with branching factor $O(B)$
     - Support insert, delete, predecessor search in $O(\log_{B+1} N)$ memory transfers
     - $\Omega(\log_{B+1} N)$ 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 $B \geq w$ 
       [Patrascu & Thorup — see L11]

2. **Sorting**: $O(\frac{N}{B} \log_{\log B} \frac{N}{B})$ memory transfers
   \(\Rightarrow\) B× faster than B-tree sort!
   - $\Omega$(ditto) in comparison model

3. **Permuting**: $O(\min \{N, \frac{N}{B} \log_{\log B} \frac{N}{B}\})$
   - $\Omega$(ditto) in indivisible model
   - physical execution \(\leq\) can’t pack pieces of input words in words

4. **Buffer tree**: $O(\frac{1}{B} \log_{\log B} \frac{N}{B})$ amortized mem. transf.
   for delayed queries & batched updates
   & $O(\phi)$ find-min \(\Rightarrow\) priority queues
Cache-oblivious model: [Frigo, Leiserson, 6.046]
Prokop, Ramachandran - FOCS 1999; Prokop - MEng 1999
- 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
   \([\text{Bender, Demaine, Farach-Colton} - \text{FOCS 2000/SICOMP 2005}]\)
   \([\text{Bender, Duan, Iacono, Wu} - \text{SODA 2002/JAlg. 2004}]\)
   \([\text{Brodal, Fagerberg, Jacob} - \text{SODA 2002}]\)
   - best constant is \(\log e\), not 1
   \([\text{Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz} - \text{FOCS 2003}]\)

2. Sorting: \(O(\frac{N}{B} \log_{b+1} \frac{N}{B})\) memory transfers
   \([\text{Frigo et al. 1999; Brodal & Fagerberg - ICALP 2002}]\)
   - uses tall-cache assumption: \(M = \Omega(B^{1+\varepsilon})\)
   - impossible otherwise \([\text{Brodal & Fagerberg - STOC 2003}]\)

3. Permuting: \(\min\) impossible \([\text{Brodal & Fagerberg - same}]\)

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

(binary search) \cite{Prokop-MEng1999}

- Store \( N \) elements in \( N \)-node complete BST
- Carve tree at middle level of edges

\[ \Rightarrow \text{one top piece, } \approx \sqrt{N} \text{ bottom pieces, each size } \approx \sqrt{N} \]

\[ \lg N \xrightarrow{\frac{1}{2} \lg N} \frac{1}{2} \lg N \xrightarrow{\frac{1}{2} \lg N} \]

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

\[ \Rightarrow \text{order to store nodes} \]

"Van Emde Boas layout"

- Generalizes to \cite{BenderDemaineFotakis2000}
  - Height not a power of 2
  - Node degrees \( \geq 2 \) & \( O(1) \)
Analysis:
- level of detail (refinement) straddling B:

- cutting height in half until piece size ≤ B
  ⇒ height of piece between \( \frac{1}{2} \log B \) & \( \log B \) (slppy)
  (⇒ size between \( \sqrt{B} \) & \( B \))
  ⇒ # pieces along root-to-leaf path ≤ \( \frac{\log N}{\frac{1}{2} \log B} \)
- each piece stores ≤ B elements consecutively
  ⇒ occupies ≤ 2 blocks (depending on alignment)
  ⇒ # memory transfers ≤ 4 \( \log B \) \( N \) (assuming \( M ≥ DB \))

(Really should be \( B+1 \))

Improvements:

[BBFGHHLIL 2003]

① randomize starting location (w.r.t. block)
  ⇒ expected cost ≤ \( (2 + \frac{3}{18}) \log B \) \( N \)

② split height into \( \frac{1}{2} - \varepsilon : \frac{1}{2} + \varepsilon \) ratio
  ⇒ expected cost ≤ \( (\log e + o(1)) \log B \) \( N \)
  = \( O(\log \log B / \log B) \)
Cache-oblivious B-trees as in [Bender, Duan, Iacono, Wu]

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

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

3. \( \{ \text{VEB layout} \} \)

3. operations:
   - binary search via left child's key
   - \text{insert}(x) \text{ finds predecessor & successor, inserts there in ordered file, & updates leaves & max's up tree via postorder traversal}
   - delete similar
4. Update analysis:
   - If \( K \) cells change in ordered file, then update tree in \( O\left(\frac{K}{B} + \log_B N\right) \) mem.
   - Look at level of detail straddling \( B \)
   - Look at bottom two levels:

   ![Diagram of update process]

   - Within chunk of \( >B \), jumping between \( \leq 2 \) pieces of \( \leq B \) (assume \( M \geq 4B \))
   \[ \Rightarrow O(\text{chunk}/B) \text{ memory transfers in chunk} \]
   - Portion in update interval +3 maybe (first, last, & root)
   \[ \Rightarrow O(\frac{K}{B}) \text{ memory transfers in bottom 2 levels} \]
   - 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
   \[ \Rightarrow O\left(\frac{K}{B} + \log_B N\right) \text{ total memory transfers} \]

So far: search in \( O(\log_B N) \) update in \( O(\log_B N + \frac{\log^2 N}{B}) \) amortized
5 Indirection:
- Cluster elements into $\Theta\left(\frac{N}{\lg N}\right)$ groups, each of size $\Theta\left(\lg N\right)$.
- Use previous structure on min's of clusters

\[
\Theta\left(\frac{N}{\lg N}\right)
\]

\[
\Theta\left(\lg N\right) \quad \Theta\left(\lg N\right) \quad \ldots \quad \Theta\left(\lg N\right)
\]

- Update cluster by complete rewrite.
  \[\Rightarrow \Theta\left(\frac{\lg N}{b}\right)\] memory transfers.
- Split/merge clusters as necessary to keep between 25% & 100% full.
  \[\Rightarrow 2\left(\lg N\right)\] updates to charge to.
  \[\Rightarrow \Theta\left(\frac{\lg^2 N}{b}\right)\] update cost in top structure.
  Only "every" $2\left(\lg N\right)$ actual updates.
  \[\Rightarrow \text{amortized update cost} \; \Theta\left(\frac{\lg N}{b}\right)\]
  (plus search cost).

Finally: $\Theta\left(\log_b N\right)$ insert, delete, predecessor, successor.
Just like B-trees in external mem. (Known $B$)