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

0. Scanning: $O(N^{1/3})$ 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
       (and $O(lg N)$ 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 $\Rightarrow \leq log (B+1)$ 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: $O(\frac{N}{B} log_{MB} \frac{N}{B})$ memory transfers
   $\Rightarrow B \times$ faster than B-tree sort!
   $\Omega$(ditto) in comparison model

3. Permuting: $O(min \frac{N}{B} log_{MB} \frac{N}{B})$
   $\Omega$(ditto) in indivisible model

4. Buffer tree: $O(\frac{1}{B} log_{MB} \frac{N}{B})$ amortized mem. transf
   for delayed queries & batched updates
   & $O(\phi)$ delete-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

[Bender, Demaine, Farach-Colton — FOCS 2000/SICOMP 2005]
[Bender, Duan, Iacono, Wu — SODA 2002/JAlg 2004]
[Brodal, Fagerberg, Jacob — SODA 2002]
— best constant is lge, not 1
[Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz — FOCS 2003]

② Sorting: $O(\frac{N}{B} \log_B N / B)$ memory transfers

[Frigo et al. 1999; Brodal & Fagerberg — ICALP 2002]
— uses tall-cache assumption: $M = \Omega(B^{1+\epsilon})$
— impossible otherwise [Brodal & Fagerberg — STOC 2003]

③ Permuting: min impossible [Brodal & Fagerberg — same]

④ Priority queue: $O(\frac{N}{B} \log_B N / B)$ 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
  \[ \Rightarrow \text{one top piece, } \approx \sqrt{N} \text{ bottom pieces, each size } \approx \sqrt{N} \]

- Recursively lay out pieces & concatenate:
  \[ \text{(in any order)} \]

\[ \frac{1}{2} \log N \]

\[ \frac{1}{2} \log N \]

\[ \approx \sqrt{N} \]

\[ \approx \sqrt{N} \]

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

- Generalizes to \[ [\text{Bender, Demaine, Farach-Colton 2000}] \]
  - Height not a power of 2
  - Node degrees $\geq 2 \& O(1)$

"van Emde Boas layout"
Analysis:
- level of detail (refinement) straddling B:

- cutting height in half until piece size ≤ B
  ⇒ height of piece between $\frac{1}{2} \lg B$ & $\lg B$ (slppy)
  ⇒ size between $\sqrt{B}$ & $B$
  ⇒ # pieces along root-to-leaf path ≤ $\frac{\lg N}{\frac{1}{2} \lg B} = 2 \log_B N$

- each piece stores ≤ B elements consecutively
  ⇒ occupies ≤ 2 blocks (depending on alignment)
  ⇒ # memory transfers ≤ 4 $\log_B N$ (assuming $M \geq 2B$)
    (really should be B+1)

Improvements: [BBFGHHIL 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 ≤ $(\lg e + o(1)) \log_B N$
     = $O(\log_B \log B)$
Cache-oblivious B-trees as in [Bender, Duan, Iacono, Wj]

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. operations:
   - binary search via left child’s key
   - $\text{insert}(x)$ finds predecessor & successor, inserts there in ordered file, & updates leaves & max’s up tree via postorder traversal
   - delete similar
update analysis:
if K cells change in ordered file
then update tree in $O(K + \log_B N)$ mem.

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

- within chunk of $> B$, jumping between $\leq 2$ pieces of $\leq B$ (assume $M \geq 2B$)
  $\Rightarrow O(\text{chunk}/B)$ memory transfers in chunk
  portion in update interval +3 maybe
  (first, last, & root)

$\Rightarrow O(K/B)$ memory transfers in bottom 2 levels
- updated nodes above these two levels:
  - subtree of $\leq K/B$ chunk roots
    up to their LCA: costs $O(K/B)$
  - path from LCA to root of tree:
    costs $O(\log_B N)$ as above
  $\Rightarrow O(K/B + \log_B N)$ 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 $\Theta\left(\frac{\lg N}{\lg N}\right)$
- use previous structure on mins of clusters

$\begin{array}{c}
\Theta(\lg N) \\
\Theta(\lg N) \\
\vdots \\
\Theta(\lg N)
\end{array}$

- update cluster by complete rewrite
  $\Rightarrow O\left(\frac{\lg N}{B}\right)$ memory transfers
- split/merge clusters as necessary to keep between 25% & 100% full
  $\Rightarrow \sum (\lg N)$ updates to charge to
  $\Rightarrow O\left(\frac{\lg^2 N}{B}\right)$ update cost in top structure
- only "every" $\sum (\lg N)$ actual updates
  $\Rightarrow$ amortized update cost $O\left(\frac{\lg N}{B}\right)$ (plus search cost)

Finally: $O(\log_B N)$ insert, delete, predecessor, successor just like B-trees in external mem. (known $B$)