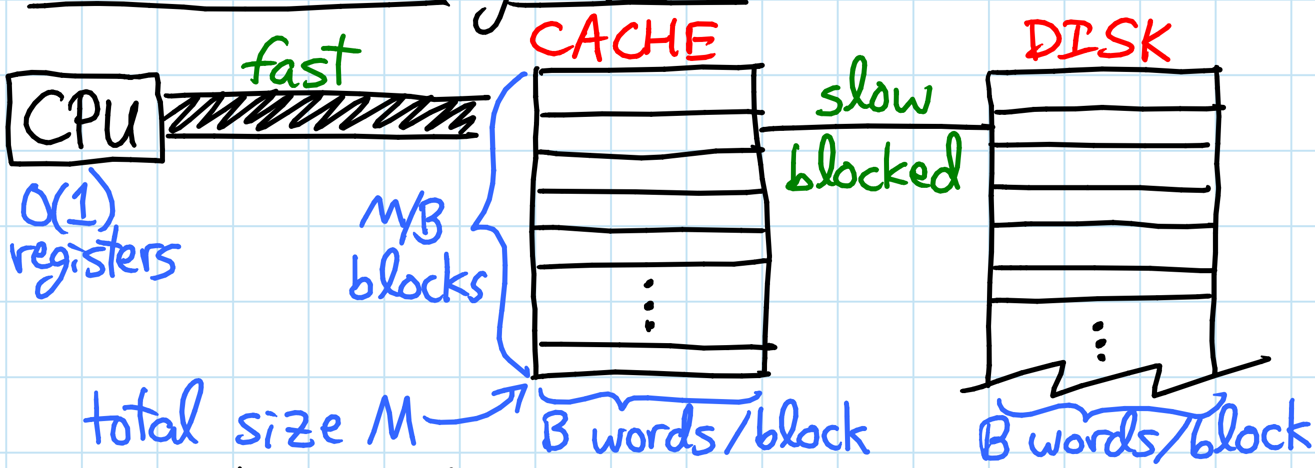
6.046	Lecture 24	May 14, 2015

TODAY: Cache-oblivious algorithms II
- search: binary
  B-ary
  cache-oblivious
- sorting: mergesorts
  cache-oblivious
- follow-on classes

Recall:
- external-memory model:



- count # (block) memory transfers MT(N)

- cache-oblivious model:
  - algorithm doesn't know B or M
  - automatic block loads & eviction of Least Recently Used (LRU) block

Why LRU block replacement strategy?

$LRU_M \le 2 \cdot OPT_{M/2}$ — RESOURCE AUGMENTATION

[Sleator & Tarjan 1985] — (changing M)

Proof:

- partition block access sequence into maximal phases of M/B distinct blocks
- LRU spends $\le M/B$ memory transfers/phase
- OPT must spend $\ge \frac{M}{2}/B$ memory transfers per phase: at best, starts phase with entire $M/2$ cache with needed items, but there are M/B blocks during phase, so $\le$ half free

ONLINE ALGORITHMS — comparing regular "online" algorithm (can't see the future) against offline/prescient optimal algorithm

- changing M by factor of 2 doesn't affect bounds like $O\left(\frac{N^2}{B\sqrt{M}}\right)$

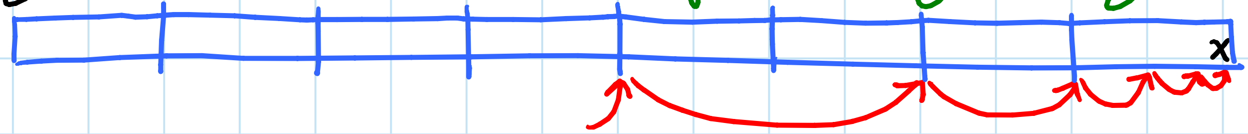
<u>Search</u>: preprocess n elements in comparison model
to support predecessor search for x

① <u>B-trees</u> support predecessor (& insert & delete)
in $O(\log_{\underbrace{B+1}_{\text{won't }>1\text{ even if }B=1}} N)$ memory transfers ~ but will ignore

- each node occupies $\Theta(1)$ blocks
- height $= \Theta(\log_B N)$
- need to know B

Cache oblivious?

② Binary search: divide & conquer is good, right?

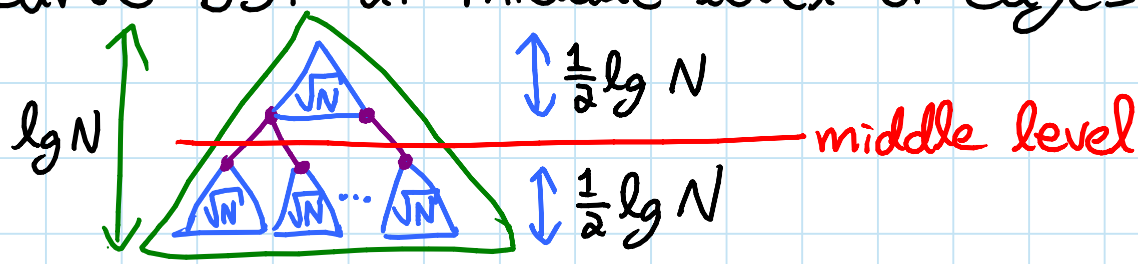


- different block ≈ until in x's block

$\Rightarrow MT(N) = \Theta(\lg N - \lg B) = \Theta(\lg N/B)$ SLOW

③ Van Emde Boas layout: [Prokop 1999]

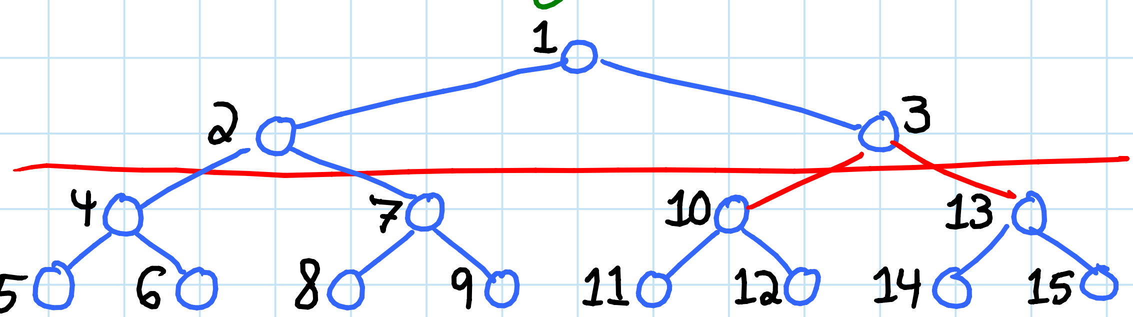
- store N elements in complete BST
- carve BST at middle level of edges:



- recursively lay out the pieces & concatenate:

- like block matrix multiplication, order of pieces doesn't matter; just need each piece to be stored consecutively

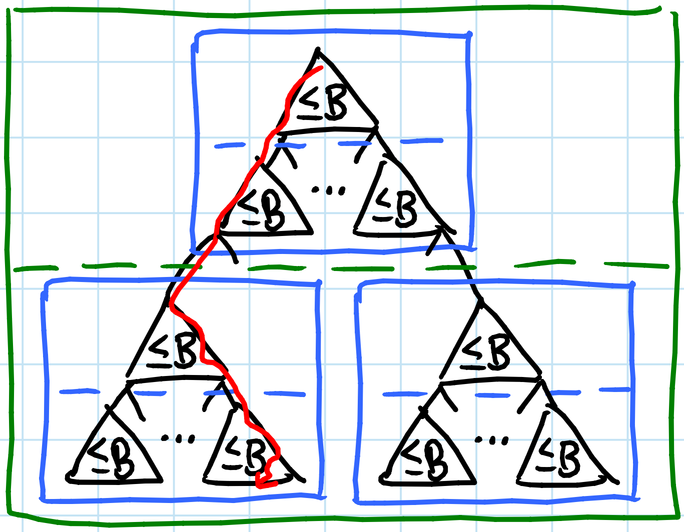
Example:



↳ order in memory

Analysis of BST search in vEB layout:

- consider recursive level of refinement at which $\triangle$ has $\leq B$ nodes:



- $\triangle_{\leq B}$ height is between $\frac{1}{2}\lg B$ & $\lg B$

(binary searching on height)

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

$\Rightarrow$ any root-to-node path (search path) visits $\leq \frac{\lg N}{\frac{1}{2}\lg B} = 2\log_B N$ $\triangle_{\leq B}$'s

- each $\triangle_{\leq B}$ occupies ≤ 2 memory blocks

$\Rightarrow \leq 4\log_B N = O(\log_B N)$ memory transfers

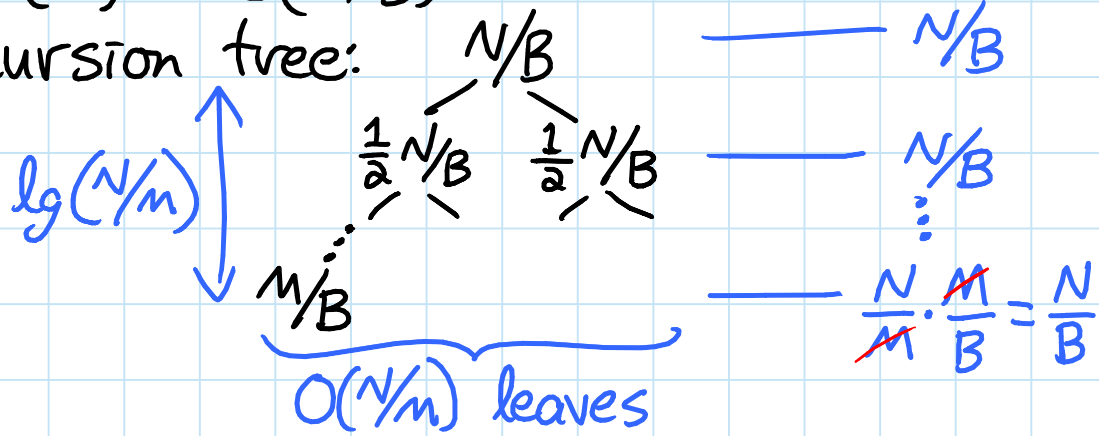
- generalizes to height not a power of 2, B-trees of constant branching factor, & dynamic B-trees: $O(\log_B N)$ insert/del. [Bender, Demaine, Farach-Colton 2000]

(see 6.851: Advanced DSs)

Sorting:

① N inserts into (cache-oblivious) B-tree
$\Rightarrow MT(N) = \Theta(N \log_B N)$ — NOT OPTIMAL
— by contrast, BST sort is optimal $O(N \lg N)$

② (binary) mergesort is cache-oblivious
— merge is 3 parallel scans
$\Rightarrow MT(N) = 2 \cdot MT(N/2) + O(N/B + 1)$
$MT(M) = O(M/B)$
— recursion tree:



$\Rightarrow MT(N) = \frac{N}{B} \lg \frac{N}{M}$ ← $\frac{B}{\lg B}$ faster than ①!

③ **M/B-way mergesort:** (vs. binary mergesort)
— split array into M/B equal subarrays (contiguous)
— recursively sort each
— merge via M/B parallel scans
(keeping one "current" block per list)

$\Rightarrow MT(N) = \frac{M}{B} \cdot MT\left(\frac{N}{M/B}\right) + O(N/B + 1)$

$MT(M) = O(M/B)$

$\Rightarrow$ height becomes $\log_{M/B} N/M + 1$

$= \log_{M/B} \frac{N}{B}\frac{B}{M} + 1$

$= \log_{M/B} N/B \cancel{- \log_{M/B} M/B} + \cancel{1}$

$= \log_{M/B} N/B$

$\Rightarrow MT(N) = O\left(\frac{N}{B} \log_{M/B} N/B\right) \leftarrow$ asymptotically optimal

(in comparison model)

④ <u>cache-oblivious sorting</u> requires <u>tall-cache assumption</u>:

$M = \Omega(B^{1+\varepsilon})$ for some fixed $\varepsilon > 0$

e.g. $M = \Omega(B^2)$ i.e. $\underbrace{M/B}_{\#\text{blocks}} = \Omega(\underbrace{B}_{\text{size of block}})$

- then $\approx N^{\varepsilon}$-way mergesort with recursive ("funnel") merge works

⑤ priority queues: $O(\frac{1}{B} \log_{M/B} N/B)$ per insert or delete-min

$\Rightarrow$ generalizes sorting

- external memory & cache oblivious!
- see 6.851

Algorithms classes at MIT: (post-6.046)

- 6.047: Computational Biology
  (genomes, phylogeny, etc.)
- 6.854: Advanced Algorithms
  (intense survey of whole field)
- 6.850: Geometric Computing
  (working with points, lines, polygons, meshes, ...)
- 6.849: Geometric Folding Algorithms ← Demaine
  (origami, robot arms, protein folding, ...)
- 6.851: Advanced Data Structures ← Demaine
  (sublogarithmic performance)
- 6.852: Distributed Algorithms ← Lynch
  (reaching consensus in a network with faults)
- 6.853: Algorithmic Game Theory
  (Nash equilibria, auction mechanism design, ...)
- 6.855: Network Optimization
  (optimization in graph: beyond shortest paths)
- 6.856: Randomized Algorithms
  (how randomness makes algs. simpler & faster)
- 6.857: Network and Computer Security
  (applied cryptography)
- 6.875: Cryptography and Cryptanalysis
  (theoretical cryptography)
- 6.816: Multicore Programming

Other theory classes:

- 6.045: Automata, Computability, & Complexity
- 6.840: Theory of Computing
- 6.841: Advanced Complexity Theory
- 6.842: Randomness & Computation
- 6.845: Quantum Complexity theory
- 6.440: Essential Coding Theory
- 6.441: Information Theory

— Frisbee Competition —