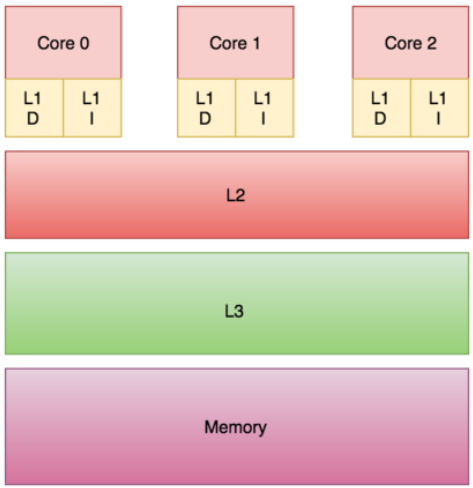
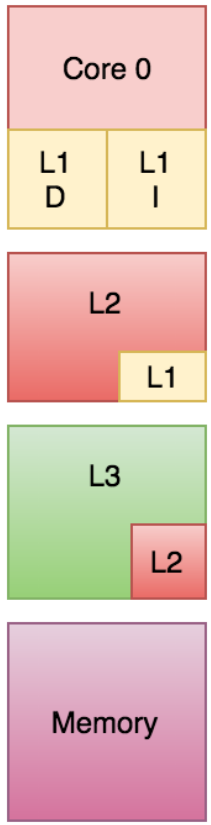
Memory Hierarchies and Models

- Up to know we have worked only with the RAM model of computation.
- This model account for communication with memory one word at a time: to read 10 words costs 10 units.
- On modern computers this is virtually never the case. Modern computers have a memory hierarchy to attempt to speed up memory operations.

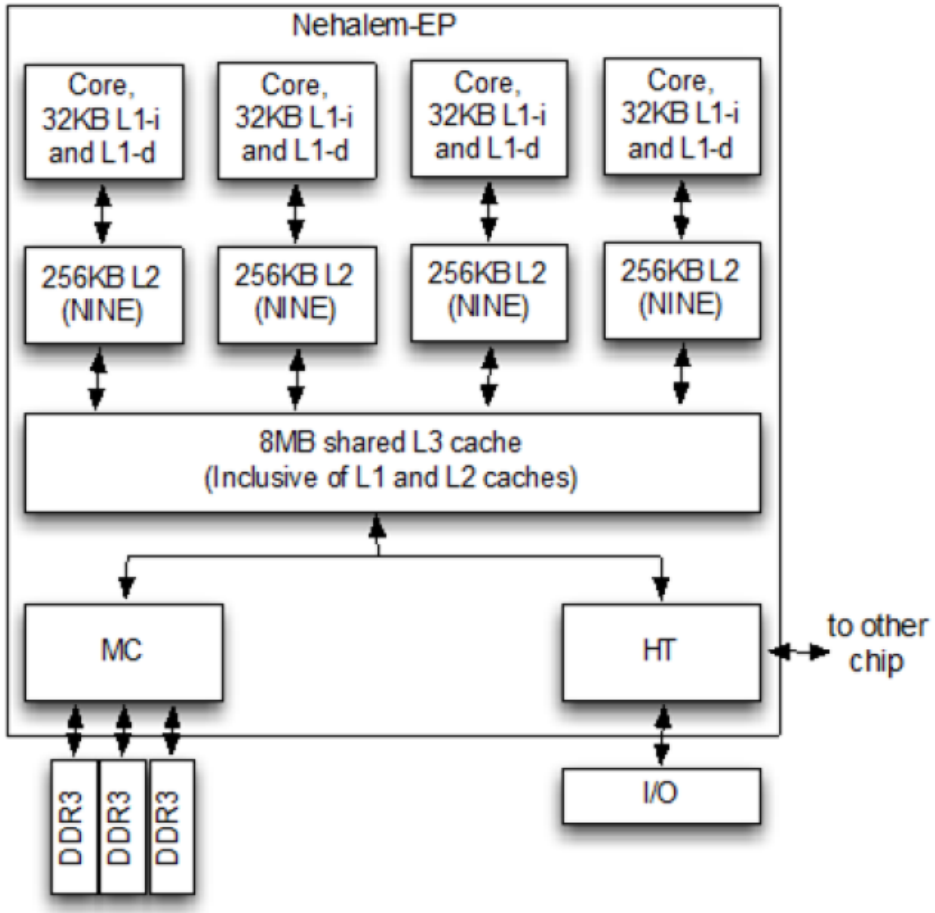
Memory Hierarchies and Models





Figures from Wikipedia.

Memory Hierarchies and Models



Figures from Wikipedia.

Memory Hierarchies and Models

Figures from Wikipedia.

Memory Hierarchies and Models

Memory Level	Size	Response Time
CPU registers	$\sim$100B	$\sim$ 0.5 ns
L1 Cache	$\sim$ 64KB	$\sim$ 1 ns
L2 Cache	$\sim$ 1MB	$\sim$ 10 ns
Main Memory	$\sim$ 2 GB	$\sim$ 150 ns
Hard Disk	$\sim$ 1TB	$\sim$10 ms

External Memory Model

Also known as Input/Output (I/O) Model or Disk Access Model (DAM). Introduced by Aggarwal and Vitter in 1988. It simplifies the memory hierarchy into just 2 levels:

- The CPU is connected to a fast cache of size M.
- The cache is in turn connected to a slower disk of effectively ”infinite” size.
- Both cache and disk are divided into blocks of size B.
- Reading and writing one block from cache to disk costs 1 unit.
- Operations on blocks in RAM are free.

External Memory Model

- Clearly any algorithm in RAM model with running time $T(N)$ requires no worse than $T(N)$ memory transfers in the external memory model.
- The lower bound which is usually harder to obtain is $\frac{T(N)}{B}$, where we take perfect advantage of cache locality, i.e., each block is read/written a constant number of times.

External Memory Model

- Note that the external memory model is a good first approximation to the slowest connection in the memory hierarchy.
- For a large database ”cache” could be system RAM and ”disk” could be the hard disk.
- For a small simulation ”cache” might be the L2 and ”disk” could be system RAM.

External Memory Model: Scanning

Clearly, scanning N items costs $O\left(\left\lceil \frac{N}{B} \right\rceil\right)$ memory tranfers.

External Memory Model: Searching

- Searching is accomplished with a B-tree using a branching factor that is $\Theta(B)$.
- Insert, delete and predecessor/successor searches are then handled with $O(\log_{B+1} N)$ memory time.
- Since in the RAM model this takes $O(\log N)$ time, the bound $O(\log_{B+1} N)$ is optimal.

B-trees (Definition)

Def. (Knuth [2]) (Introduced by Bayer& McCreight in 1972). A B-tree of order m is a tree that satisfies:

- Every node has at most m children.
- Every node (except the root) hast at least $\lceil \frac{m}{2} \rceil - 1$ children.
- The root has at least 2 children if it is not a leaf node.
- A non-leaf node with k children contains $k - 1$ keys.
- All leaves appear in the same level and carry the associated information.

B-TREES (EXEMPLE)

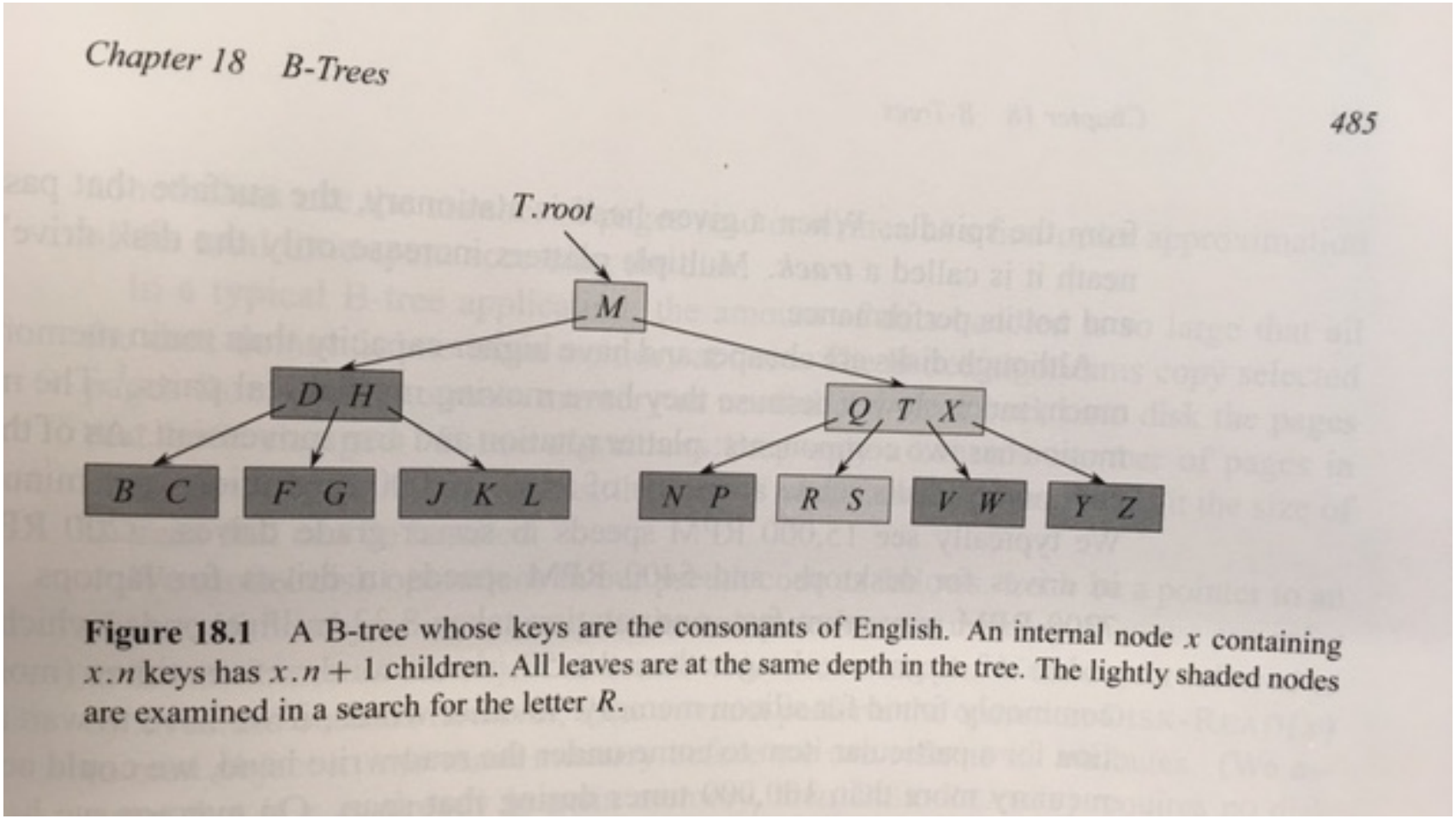


Figure 18.1 A B-tree whose keys are the consonants of English. An internal node x containing $x.n$ keys has $x.n + 1$ children. All leaves are at the same depth in the tree. The lightly shaded nodes are examined in a search for the letter R.

Figure from [1].

B-trees (Searching and Updating)

Search: Identical to Binary Search Trees.

Insertions: Locate the leaf in which the new key should be inserted.

- If it has less than m keys insert and finish.
- If it has m keys split the leaf into two leaves of size $\frac{m}{2}$ and send the median key to the parent node and so on.

Deletion from an internal node: Choose a new separator (either the greatest of the left child of the deleted node or the smallest of the right child), if underflow happens re-balance.

Deletion from a leaf: Just delete, if underflow happens re-balance.

B-TREES (SPLIT EXEMPLE)

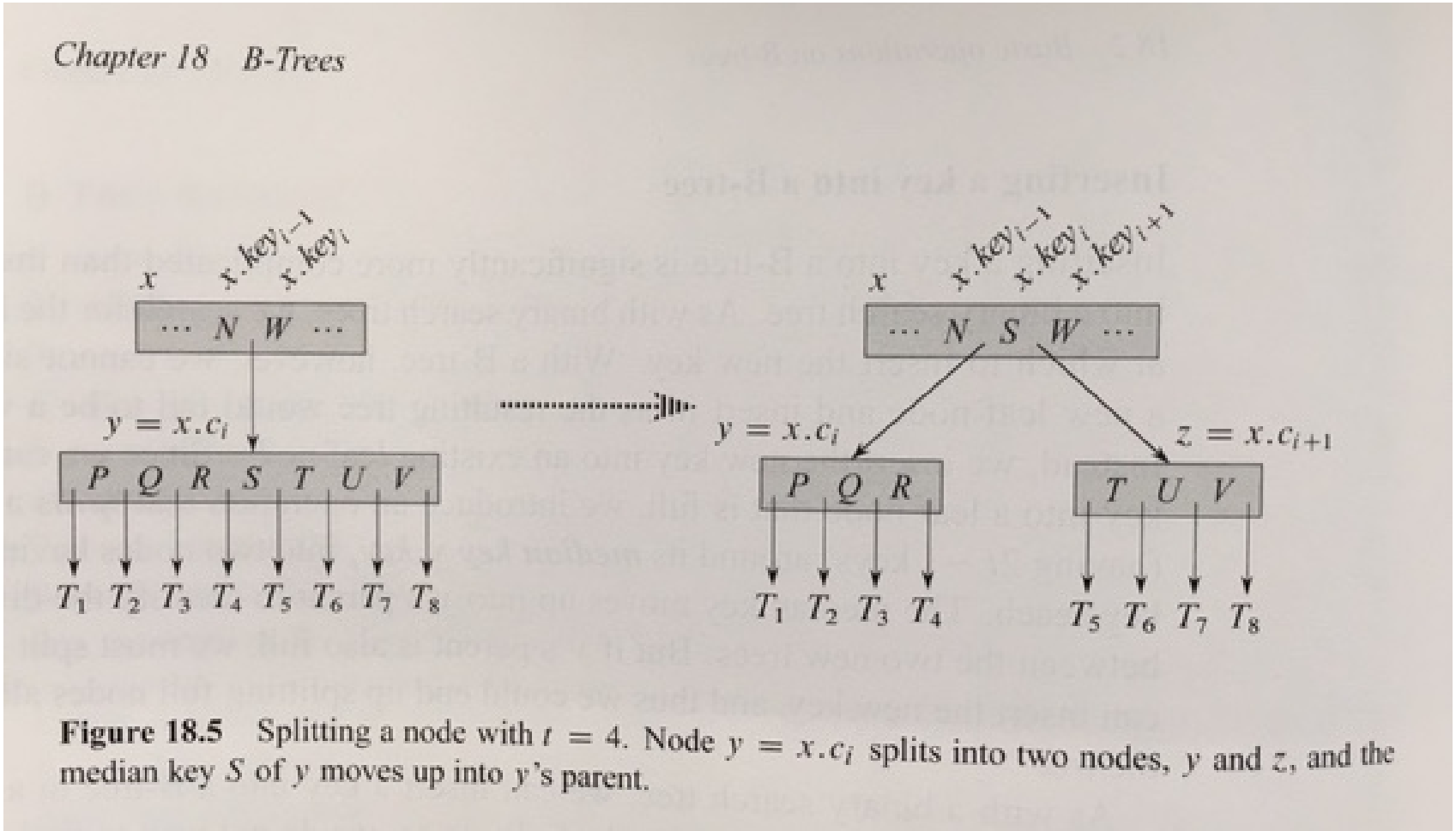


Figure 18.5 Splitting a node with $t = 4$. Node $y = x.c_i$ splits into two nodes, y and z, and the median key S of y moves up into y's parent.

Figure from [1].

B-TREES (INSERTION EXEMPLE)

(From Knuth's definition $m = 6$.)

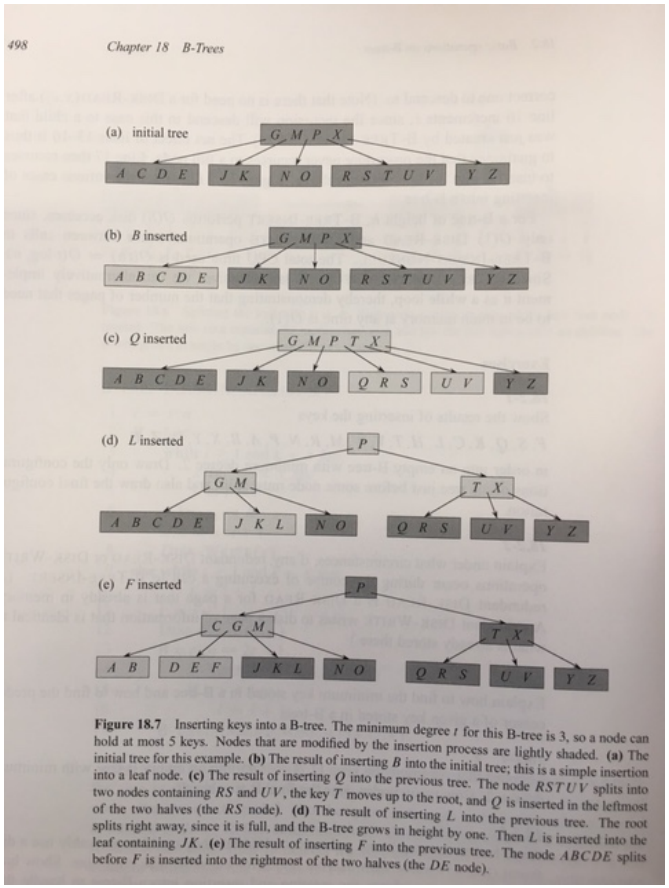


Figure 18.7 Inserting keys into a B-tree. The minimum degree t for this B-tree is 3, so a node can hold at most 5 keys. Nodes that are modified by the insertion process are lightly shaded. **(a)** The initial tree for this example. **(b)** The result of inserting B into the initial tree; this is a simple insertion into a leaf node. **(c)** The result of inserting Q into the previous tree. The node $RSTUV$ splits into two nodes containing RS and UV, the key T moves up to the root, and Q is inserted in the leftmost of the two halves (the RS node). **(d)** The result of inserting L into the previous tree. The root splits right away, since it is full, and the B-tree grows in height by one. Then L is inserted into the leaf containing JK. **(e)** The result of inserting F into the previous tree. The node $ABCDE$ splits before F is inserted into the rightmost of the two halves (the DE node).

Figure from [1].

B-TREES (DELETION EXEMPLE)

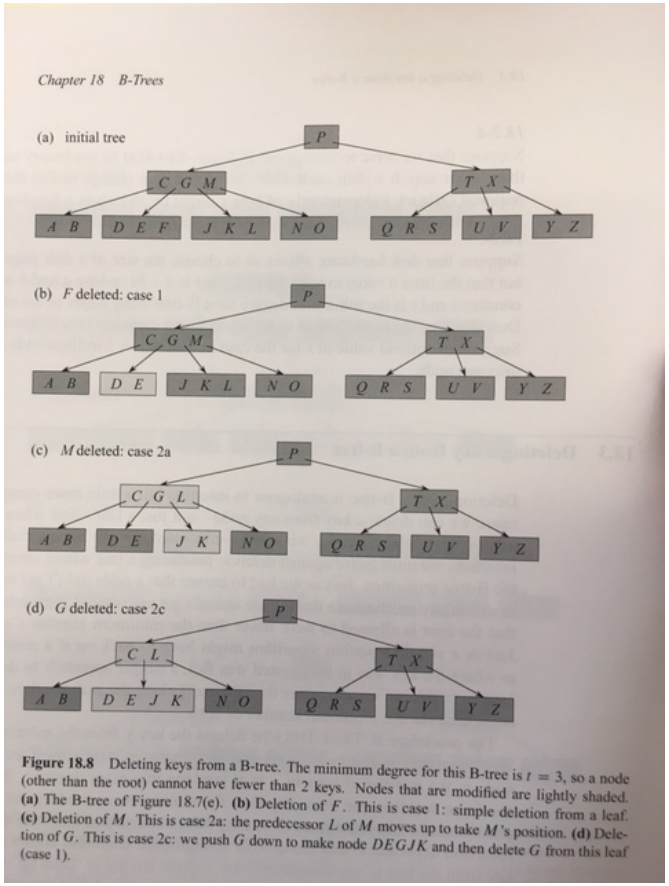


Figure 18.8 Deleting keys from a B-tree. The minimum degree for this B-tree is $t = 3$, so a node (other than the root) cannot have fewer than 2 keys. Nodes that are modified are lightly shaded. **(a)** The B-tree of Figure 18.7(e). **(b)** Deletion of F. This is case 1: simple deletion from a leaf. **(c)** Deletion of M. This is case 2a: the predecessor L of M moves up to take M's position. **(d)** Deletion of G. This is case 2c: we push G down to make node $DEGJK$ and then delete G from this leaf (case 1).

Figure from [1].

External Memory Model: Sorting

In the RAM model a B-tree can sort in optimal time: insert all elements and successively delete the minimal element. the same technique yields $O\left(\frac{N}{B}\log_{B+1}\frac{N}{B}\right)$ (amortized) memory transfers in the external memory model, which is not optimal. An optimal algorithm is a $\frac{M}{B}$-way version of merge-sort. It obtains optimal performance by solving subproblems that fit in cache, leading to a total of $O\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$ memory tranfers.

External Memory Model: Permutation

The permutation problem is: Given N elements in some order and a new ordering, re-arrange the elements to appear in the new order.
Naively, this takes $O(N)$ operations: just swap each element into its new place.
It may be fastest to make the key equal to the permutation ordering and then apply the above optimal sort. This gives a bound of $O\left(\min\{N, \frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\}\right)$ amortized.

External Memory Model: Buffer Trees

Buffer trees are the external memory priority queue.
They also achieve the sorting bound if all elements are inserted an then the minimum is deleted sequentially.
Buffer trees achieve $O\left(\frac{1}{B}\log_{\frac{M}{B}}\frac{N}{B}\right)$ (amortized) memory transfers per operation.

Cache-Oblivious Model

Similar to the External Memory Model except that the values of M and B are unknown. One has to work with the assumption that bounds have to be reached with any combination of M, B.
Assumptions:

- Tall cache assumption: $M > \Omega(B^2)$.
- Memory transfers are performed by an off-line optimal replacement strategy.

Cache-Oblivious Fundamental Primitives: Scanning

Scanning an array with N elements incurs $\Theta(\frac{N}{B})$ memory transfers for any value B, thus, algorithms such as finding the median and data structures such as stacks and queues that only rely on scanning are automatically cache-oblivious.

Cache-Oblivious Fundamental Primitives: Van Emde-Boas Layout

Apart from algorithms and data structures that only use scanning, most cache-oblivious results use recursion to obtain efficiency, in almost cases, the sizes of the recursive problem decreases double-exponentially.
For simplicity we only consider complete binary trees.
The basic idea in the Van Emde-Boas Layout of a complete binary tree T with N leaves is to divide T at the middle level and layout the pieces recursively.

Cache-Oblivious Fundamental Primitives: Van Emde-Boas Layout

If T has only one node, it is simply laid out as a single node in memory.
Otherwise let $h = \log(N)$ be the height of T. We define the top tree T_0 to be the subtree consisting of the nodes in the topmost $\lfloor \frac{h}{2} \rfloor$ levels of T and the bottom trees $T_1, T_2, \ldots, T_k$ be the $O(\sqrt{N})$ subtrees rooted in the nodes of level $\lceil \frac{h}{2} \rceil$ of T.

Note that all the subtrees have size $\Theta(\sqrt{N})$.

The Van Emde-Boas layout of T consists of the Van Emde-Boas layout of T_0 followed by the Van Emde-Boas layout of $T_1, T_2, \ldots, T_k$.
Subdivide until trees are of size B.

Cache-Oblivious Fundamental Primitives: Van Emde-Boas Layout

Search:
To analyze the number of memory transfers needed to perform a search in T, that is, traverse a root-leaf path, we consider the first recursive level of the Van Emde-Boas layout where the subtrees are smaller than B.
At this level T is divided into a set of base trees of size between $\Theta(\sqrt{B})$ and $\Theta(B)$, that is, of height $\Omega(\log(B))$.
By definition of the layout, each base tree is stored in $O(B)$ contiguous memory locations and thus can be accessed in $O(1)$ memory transfers.
The search is performed in $O(\log_B N)$ memory transfers follows since the search path traverses $O\left(\frac{\log N}{\log B}\right)$ different base trees.

vEB-TREES

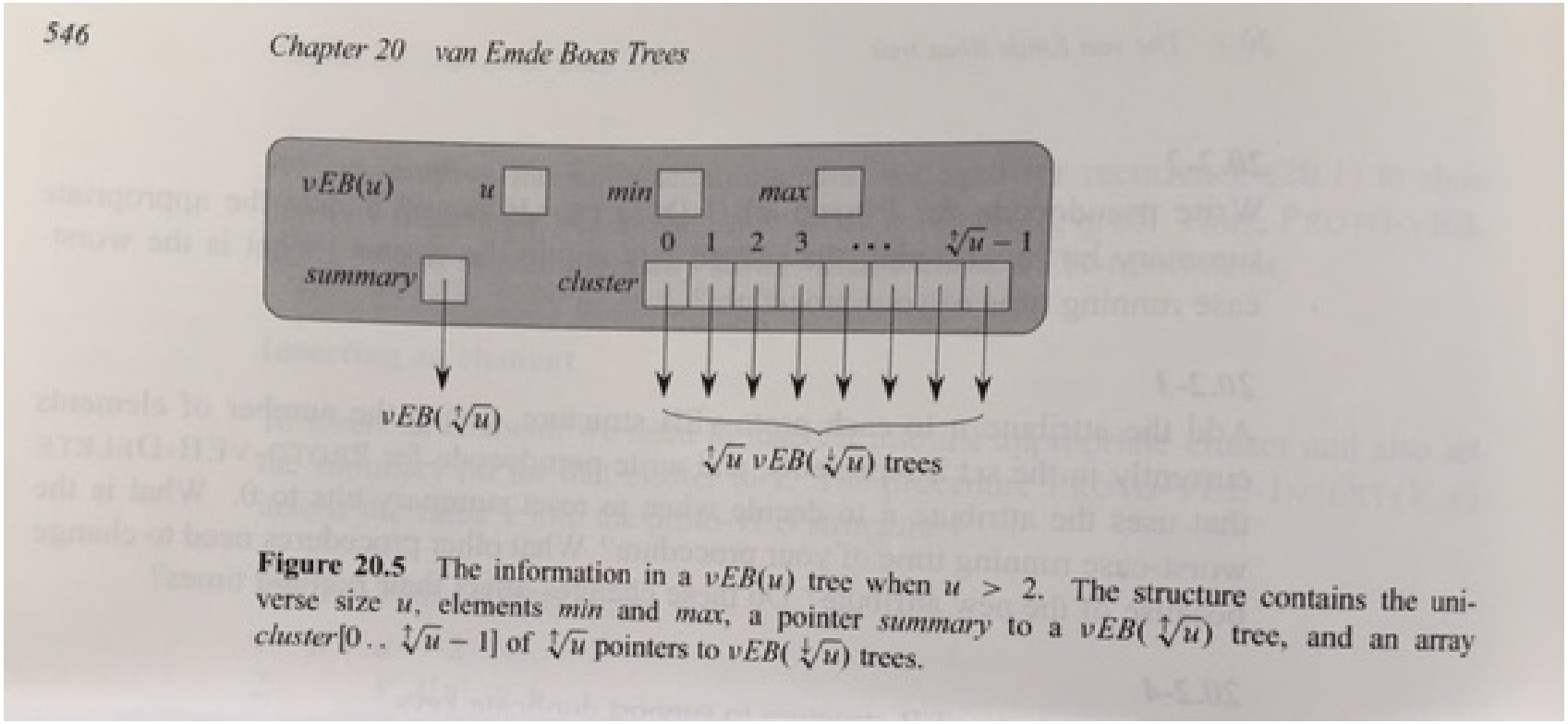


Figure 20.5 The information in a $vEB(u)$ tree when $u > 2$. The structure contains the universe size u, elements *min* and *max*, a pointer *summary* to a $vEB(\sqrt[2]{u})$ tree, and an array *cluster*$[0..\sqrt[2]{u}-1]$ of $\sqrt[2]{u}$ pointers to $vEB(\sqrt[2]{u})$ trees.

Figure from [1].

vEB-trees

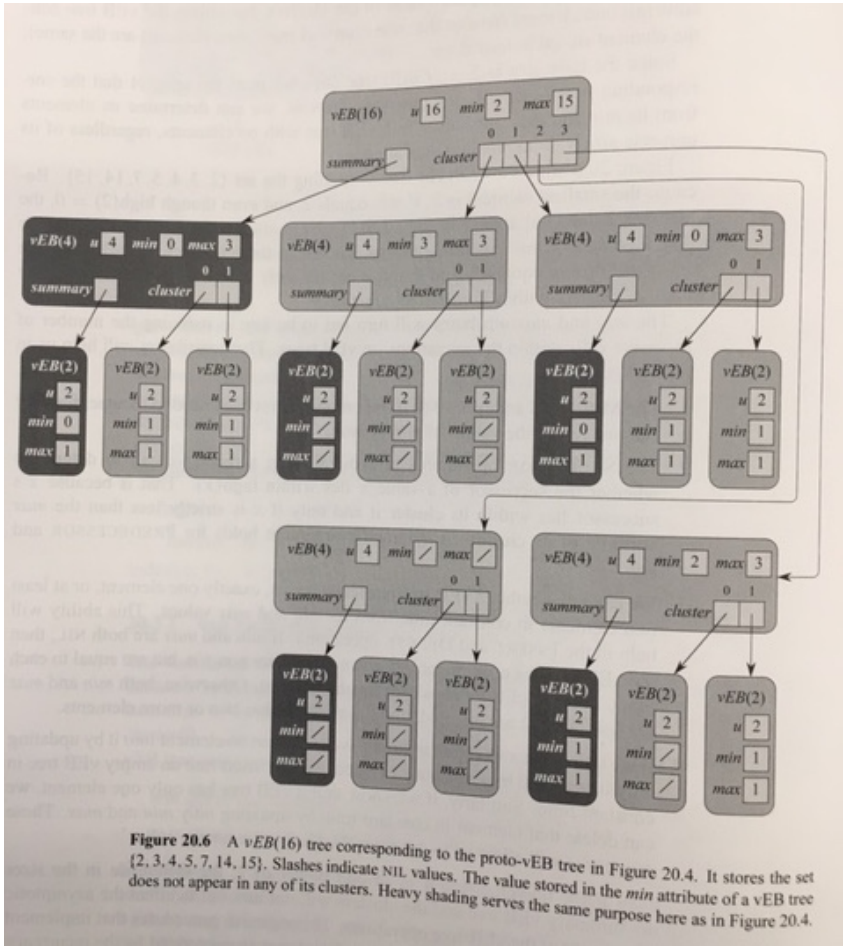


Figure 20.6 A $vEB(16)$ tree corresponding to the proto-vEB tree in Figure 20.4. It stores the set $\{2, 3, 4, 5, 7, 14, 15\}$. Slashes indicate NIL values. The value stored in the *min* attribute of a vEB tree does not appear in any of its clusters. Heavy shading serves the same purpose here as in Figure 20.4.

Figure from [1].


T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein.
Introduction to Algorithms, Third Edition.
The MIT Press, 2009.


D. E. Knuth.
The Art of Computer Programming: Sorting and Searching, volume 3.
Addison–Wesley, 2nd edition, 1998.