Cache-Oblivious Search Trees

The Unknown Machine

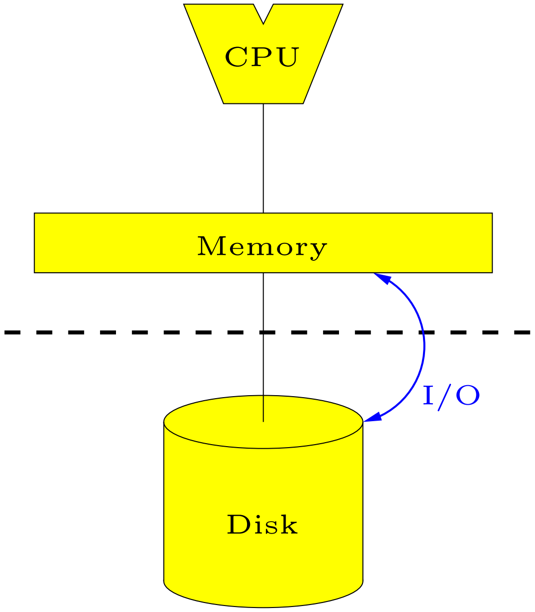
Algorithm	Algorithm
$\downarrow$	$\downarrow$
C program	Java program
$\downarrow$ gcc	$\downarrow$ javac
Object code	Java bytecode
$\downarrow$ linux	$\downarrow$ java
Execution	Interpretation

Can be executed on machines with a specific class of CPUs

Can be executed on any machine with a Java interpreter

Goal Develop algorithms that are optimized w.r.t. memory hierarchies without knowing the parameters

Cache-Oblivious Model



- I/O model
- Algorithms do not know the parameters B and M
- Optimal off-line cache replacement strategy

Frigo et al. 1999

Justification of the ideal-cache model

Optimal replacement

LRU + 2 $\times$ cache size $\Rightarrow$ at most 2 $\times$ cache misses Sleator an Tarjan, 1985

Corollary

$T_{M,B}(N) = O(T_{2M,B}(N)) \Rightarrow$ #cache misses using LRU is $O(T_{M,B}(N))$

Two memory levels

Optimal cache-oblivious algorithm satisfying $T_{M,B}(N) = O(T_{2M,B}(N))$
$\Rightarrow$ optimal #cache misses on each level of a multilevel cache using LRU

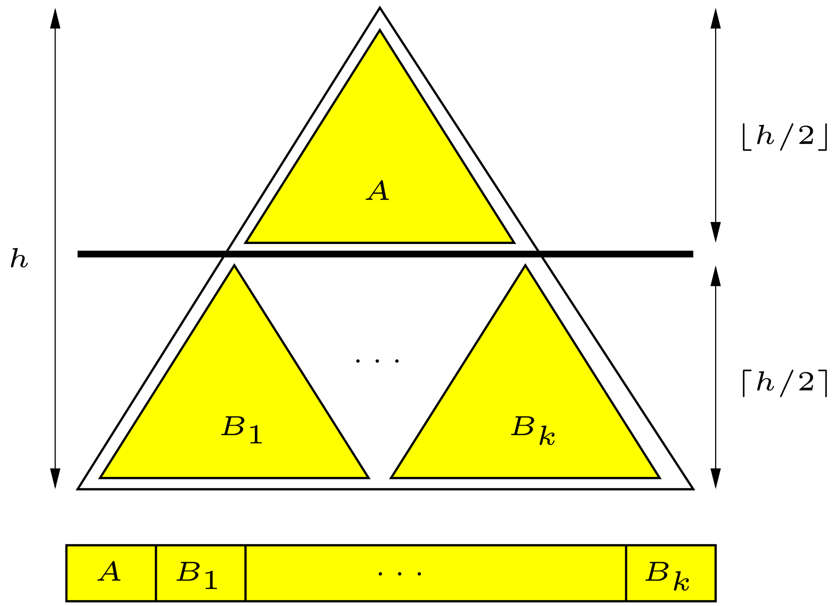
Fully associativity cache

Simulation of LRU

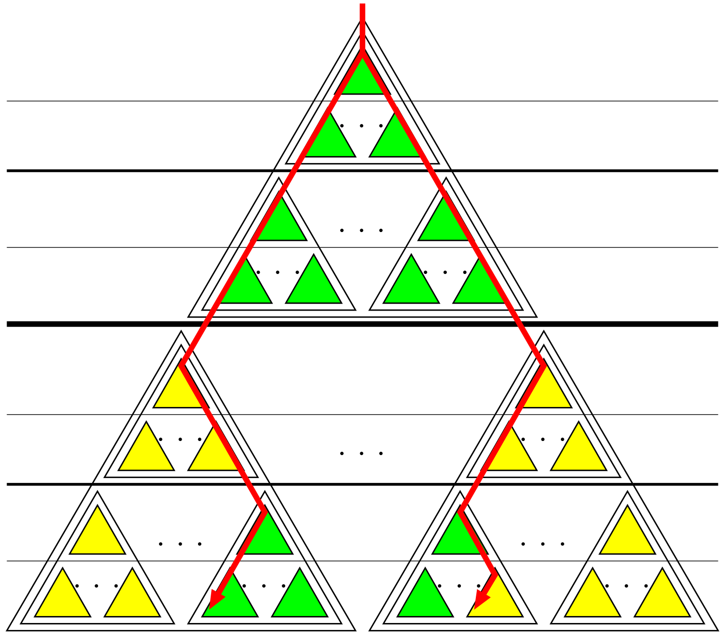
- Direct mapped cache
- Explicit memory management
- Dictionary (2-universal hash functions) of cache lines in memory
- Expected $O(1)$ access time to a cache line in memory

Static Cache-Oblivious Trees

Recursive memory layout $\equiv$ van Emde Boas layout



Degree O(1)



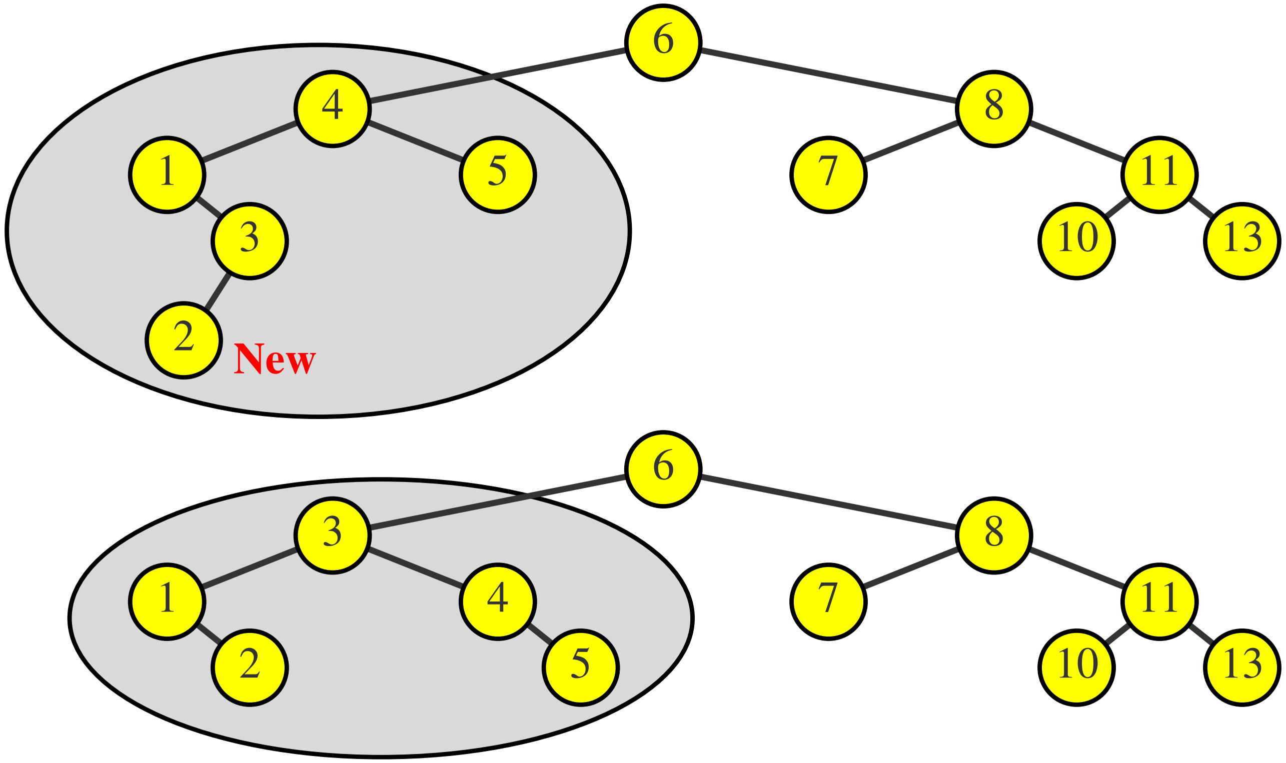
Searches use $O(\log_B N)$ **I/Os**

Range reportings use

$O\left(\log_B N + \frac{k}{B}\right)$ **I/Os**

Prokop 1999

Binary Trees of Small Height



- If an insertion causes non-small height then rebuild subtree at nearest ancestor with sufficient few descendents
- Insertions require amortized time $O(\log^2 N)$

Andersson and Lai 1990

Binary Trees of Small Height

- For each level i there is a threshold $\tau_i = \tau_L + i\Delta$, such that $0 < \tau_L = \tau_0 < \tau_1 < \cdots < \tau_H = \tau_U < 1$
- For a node v_i on level i define the density

$$\rho(v_i) = \frac{\text{\# nodes below } v_i}{m_i}$$

where m_i = # possible nodes below v_i with depth at most H

Insertion

- Insert new element
- If depth $> H$ then locate neirest ancestor v_i with $\rho(v_i) \leq \tau_i$ and rebuild subtree at v_i to have minimum height and elements evenly distributed between left and right subtrees

Andersson and Lai 1990

Binary Trees of Small Height

Theorem Insertions require amortized time $O(\log^2 N)$

Proof Consider two redistributions of v_i

- After the first redistribution $\rho(v_i) \leq \tau_i$
- Before second redistribution a child v_{i+1} of v_i has $\rho(v_{i+1}) > \tau_{i+1}$
- Insertions below v_i : $m(v_{i+1}) \cdot (\tau_{i+1} - \tau_i) = m(v_{i+1}) \cdot \Delta$
- Redistribution of v_i costs $m(v_i)$, i.e. per insertion below v_i

$$\frac{m(v_i)}{m(v_{i+1}) \cdot \Delta} \leq \frac{2}{\Delta}$$

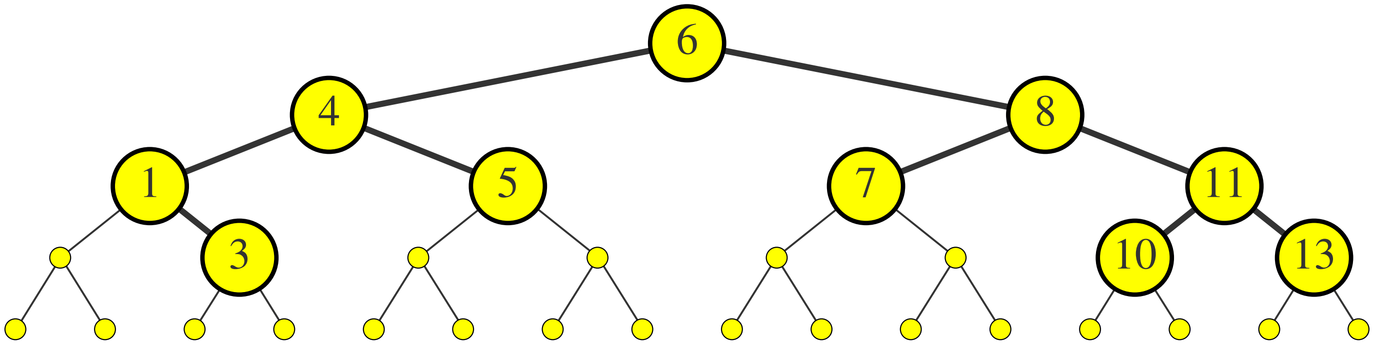
- Total insertion cost per element

$$\sum_{i=0}^{H} \frac{2}{\Delta} = O(\log^2 N) \qquad \square$$

Andersson and Lai 1990

Dynamic Cache-Oblivious Trees

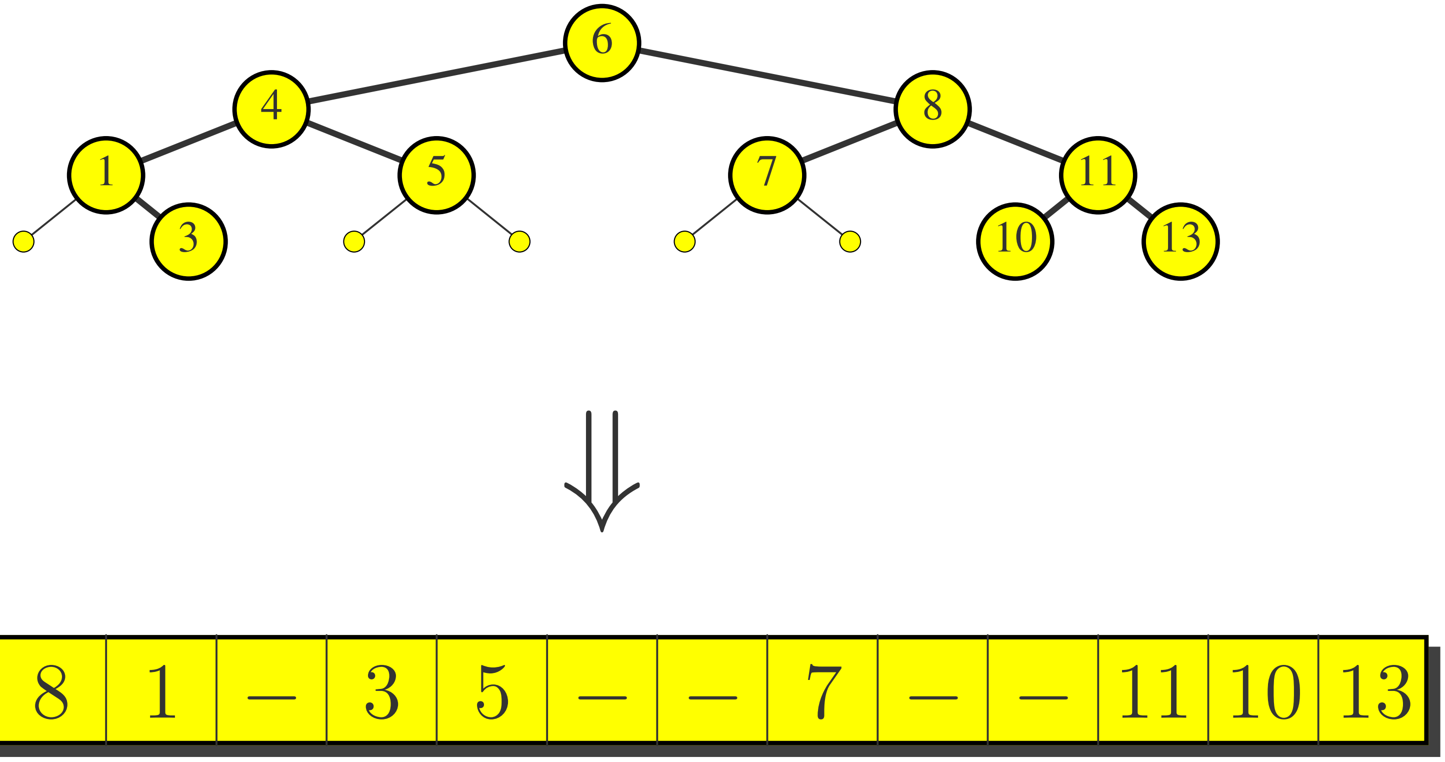
- Embed a dynamic tree of small height into a complete tree
- Static van Emde Boas layout
- Rebuild data structure whenever N doubles of halves



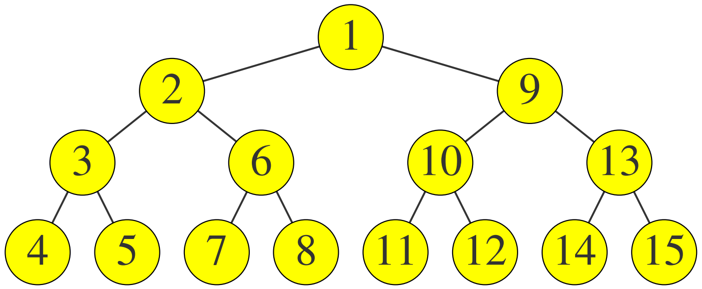
Search	$\mathrm{O}(\log_B N)$
Range Reporting	$\mathrm{O}\left(\log_B N + \frac{k}{B}\right)$
Updates	$\mathrm{O}\left(\log_B N + \frac{\log^2 N}{B}\right)$

Brodal, Fagerberg, Jacob 2001

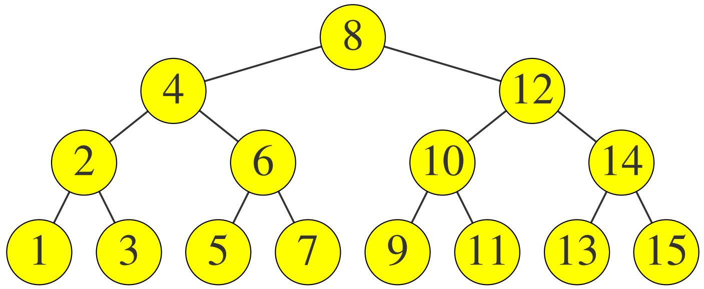
Example



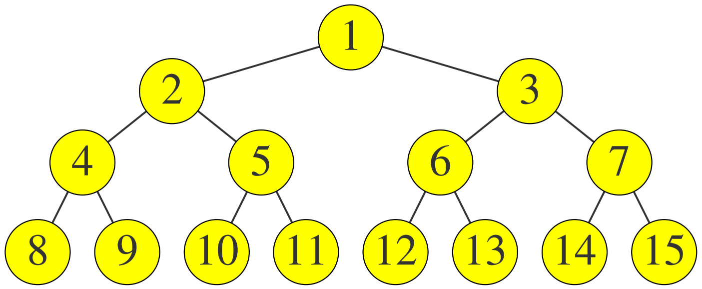
Memory Layouts of Trees



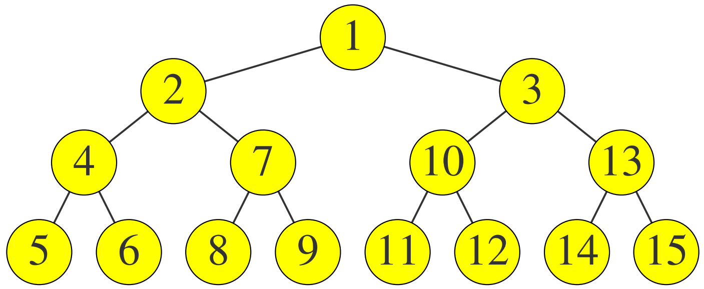
DFS



inorder

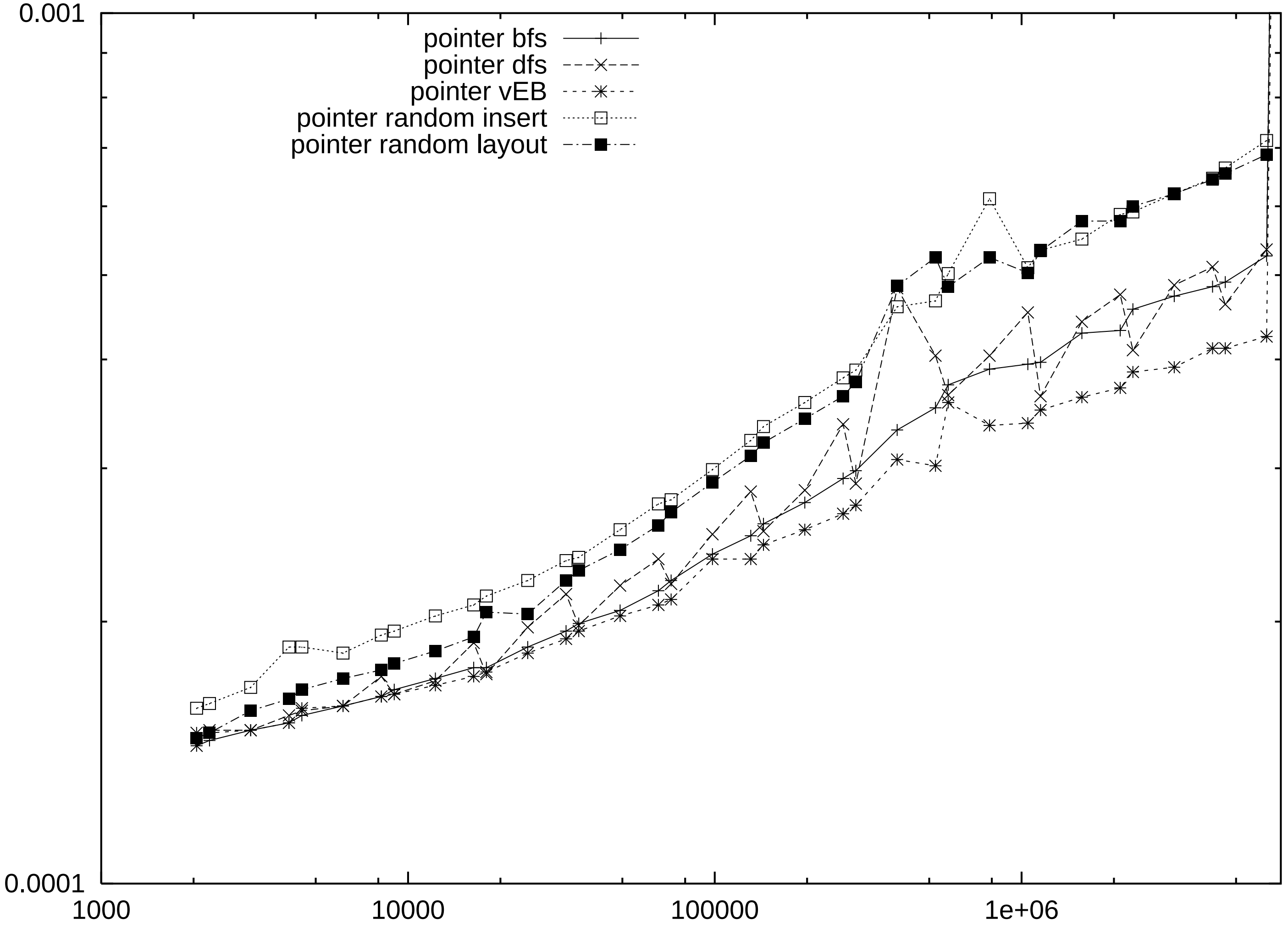


BFS



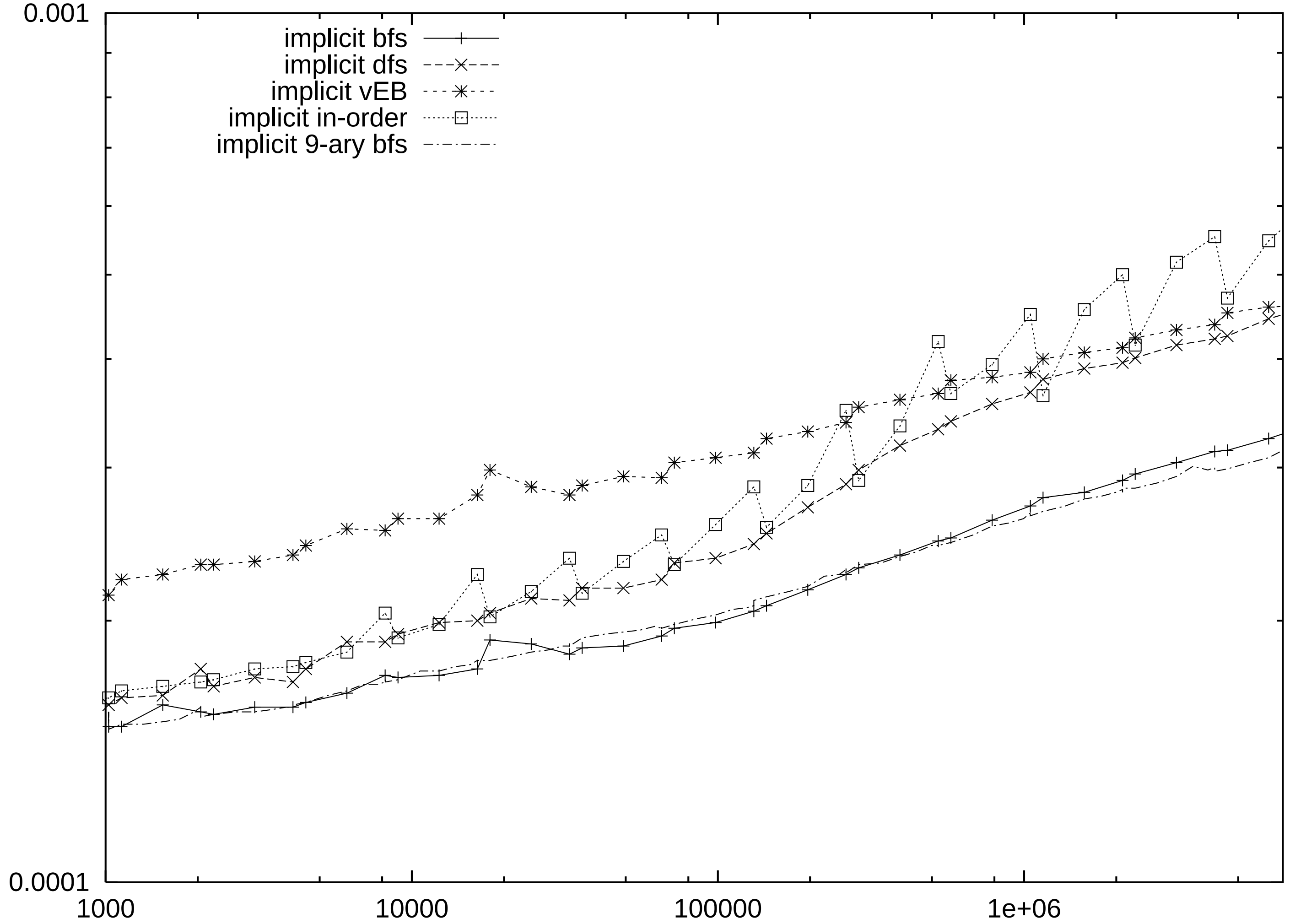
van Emde Boas
(in theory best)

Searches in Pointer Based Layouts



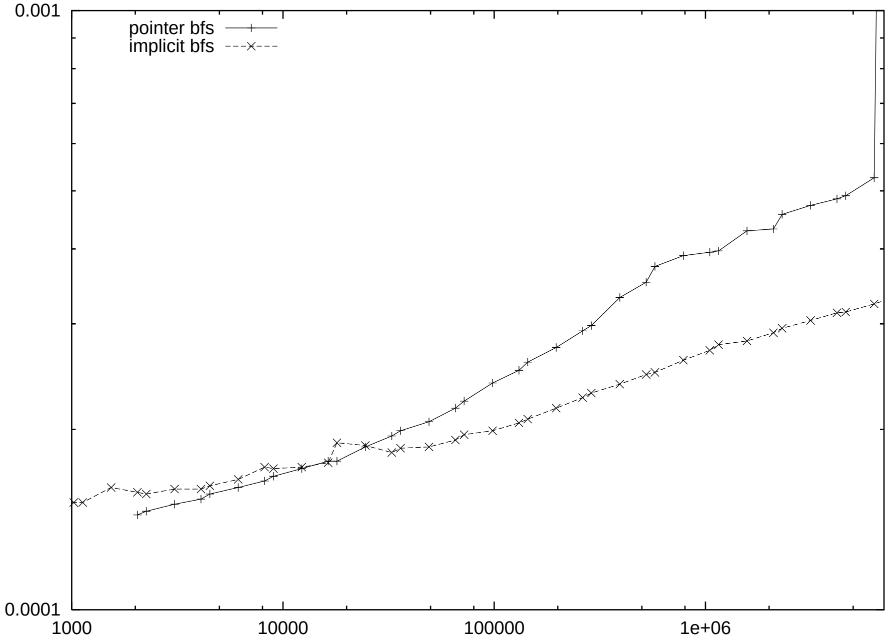
- van Emde Boas layout wins, followed by the BFS layout

Searches with Implicit Layouts

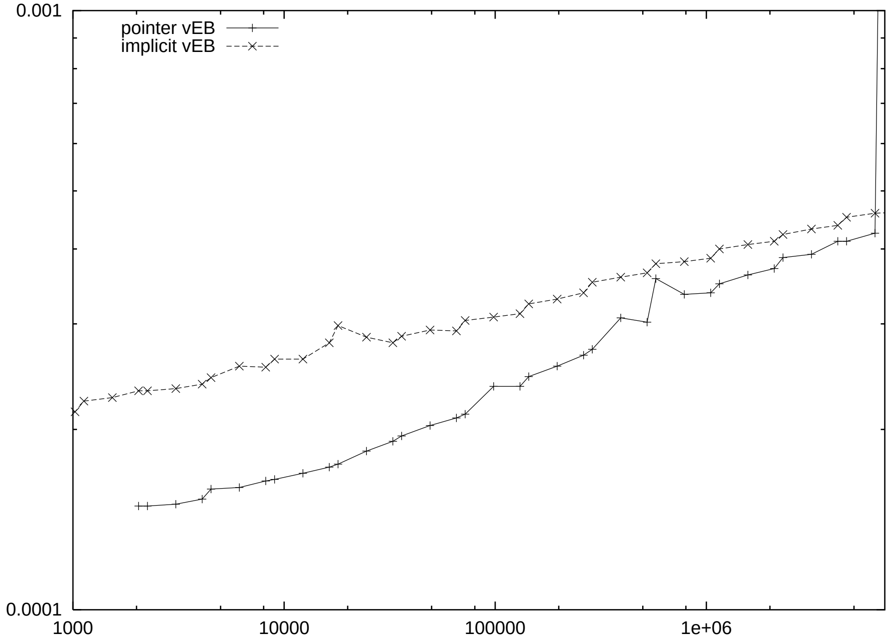


- BFS layout wins due to simplicity and caching of topmost levels
- van Emde Boas layout requires quite complex index computations

Implicit vs Pointer Based Layouts



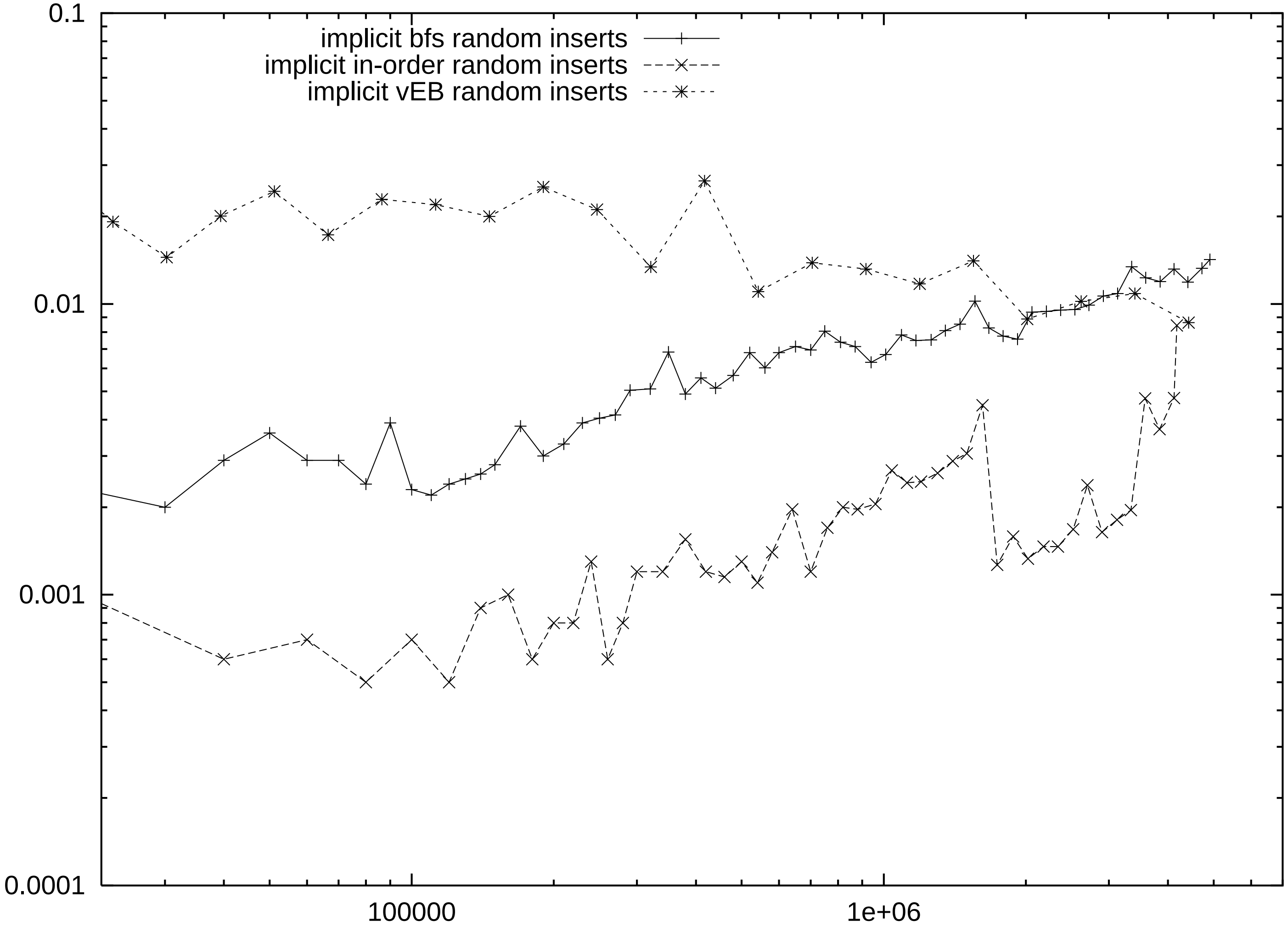
BFS layout



van Emde Boas layout

- Implicit layouts become competitive as n grows

Insertions in Implicit Layouts



- Insertions are rather slow (factor 10-100 over searches)

Summary

- Cache-oblivious search trees

Search	$O(\log_B N)$
Range Reporting	$O\left(\log_B N + \frac{k}{B}\right)$
Updates	$O\left(\log_B N + \frac{\log^2 N}{B}\right)$

- Update time $O(\log_B N)$ by one level of indirection (implies sub-optimal range reporting)
- Importance of memory layouts
- van Emde Boas layout gives good cache performance
- Computation time is important when considering caches

6	4	8	1	–	3	5	–	–	7	–	–	11	10	13