Cache-Oblivious Search Trees via Trees of Small Height

Gerth Stølting Brodal

BRICS

University of Aarhus

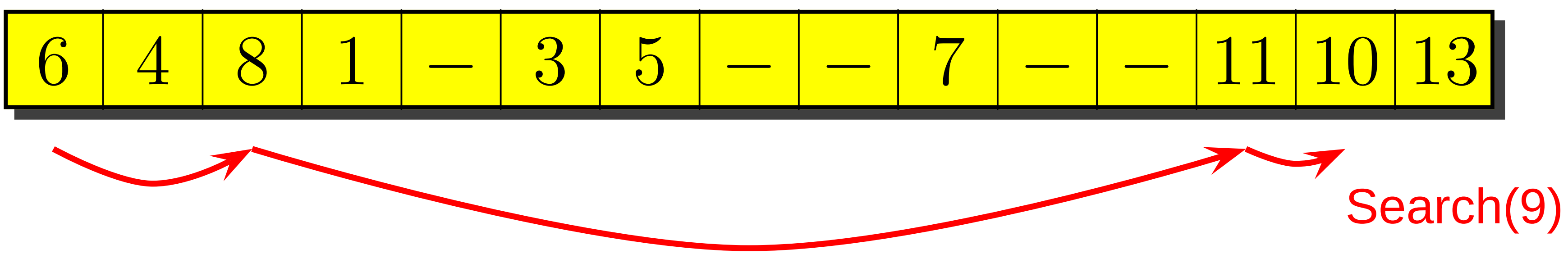
Joint work with Rolf Fagerberg and Riko Jacob

To be presented at the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002

New Search Tree

$$\{\ 1,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 10,\ 11,\ 13\ \}$$

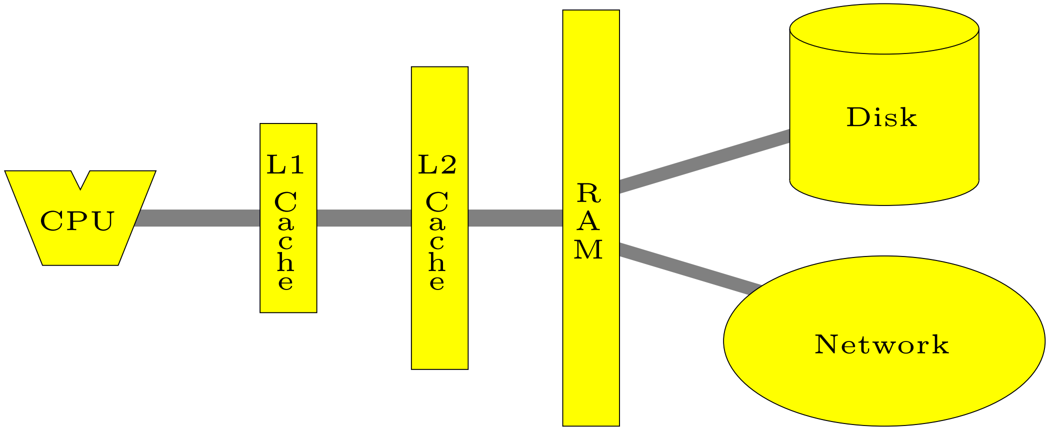
$$\Downarrow$$



Outline

- Trends in Implementation Technology
- Models of Computation
  - I/O Model
  - Cache-Oblivious Model
- Cache-Oblivious Search Trees
  - Static
  - Dynamic
- Experiments
  - Memory Layouts of Trees
- Summary

Trends in Implementation Technology



Integrated Circuit Logic Technology

- Transistor count increases $\approx$ 60-80% per year

DRAM

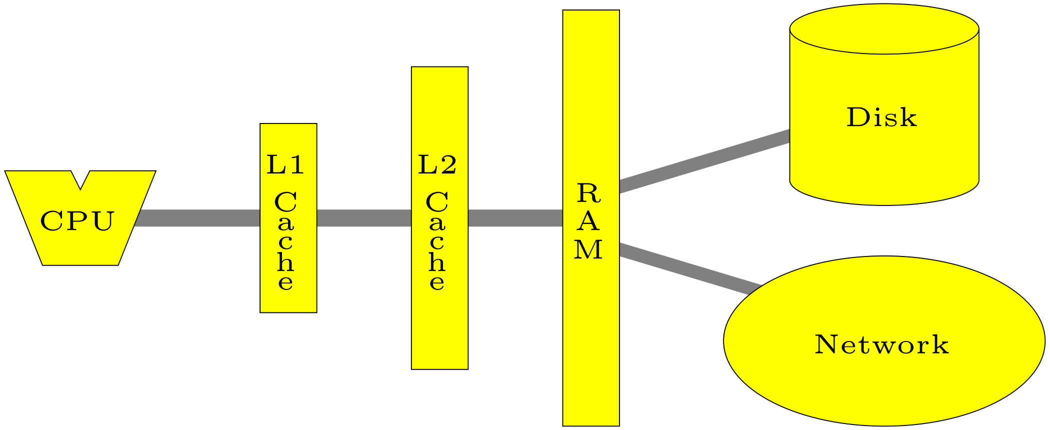
- Density improves $\approx$ 60% per year
- Cycle time improves $\approx$ 35% per 10 years

Magnetic Disk

- Density improves $\approx$ 50% per year
- Access time improves $\approx$ 35% per 10 years

Source: *Computer Architecture – A Quantitative Approach*, Hennessy & Patterson, 2nd. Ed. 1996

Trends in Implementation Technology



	L1 Cache	L2 Cache	Virtual memory
Block size	4 – 32 bytes	32 – 256 bytes	4 – 16 KB
Hit time (cycles)	1 – 2	6 – 15	10 – 100
Miss penalty (cycles)	8 – 66	30 – 200	700.000 – 6.000.000
Size	1 – 128 KB	256 KB – 16 MB	16 – 8192 MB

Source: *Computer Architecture – A Quantitative Approach*, Hennessy & Patterson, 2nd. Ed. 1996

The Unknown Machine

Algorithm

↓

C program

↓ gcc

Object code

↓ linux

Execution

Can be executed on machines with a specific class of CPUs

Algorithm

↓

Java program

↓ javac

Java bytecode

↓ java

Interpretation

Can be executed on any machine with a Java interpreter

The Unknown Machine

Algorithm	Algorithm
↓	↓
C program	Java program
↓ gcc	↓ javac
Object code	Java bytecode
↓ linux	↓ java
Execution	Interpretation

Can be executed on machines with a specific class of CPUs

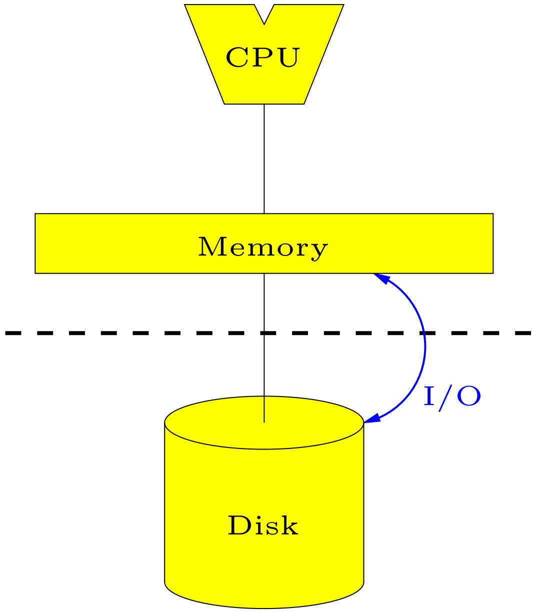
Can be executed on any machine with a Java interpreter

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

Outline

- Trends in Implementation Technology
- Models of Computation
  - I/O Model
  - Cache-Oblivious Model
- Cache-Oblivious Search Trees
  - Static
  - Dynamic
- Experiments
  - Memory Layouts of Trees
- Summary

I/O Model



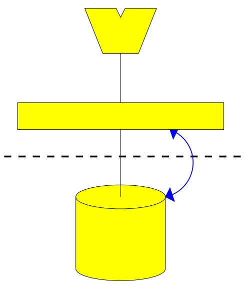
N = problem size
M = memory size
B = I/O block size

Aggarwal and Vitter 1988

- Bottleneck $\equiv$ I/Os between the two highest memory levels
- B-trees support searches and updates in $O(\log_B N)$ I/Os
- $\Theta\left(\frac{M}{B}\right)$-way merge-sort achieves optimal $\Theta\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os

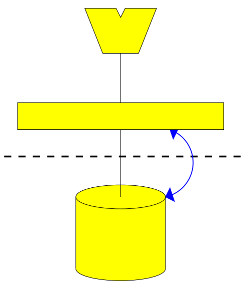
Cache-Oblivious Model

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



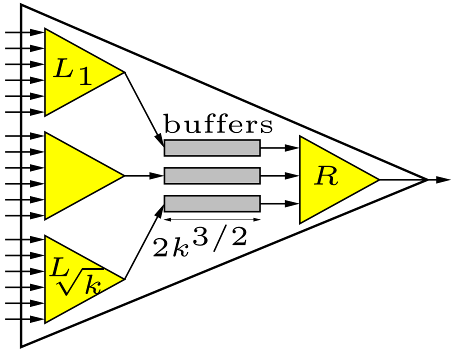
Cache-Oblivious Model

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



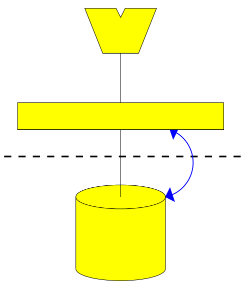
Examples

- Scanning, Linear time selection
- Matrix-transposition, FFT, Funnel-sorting



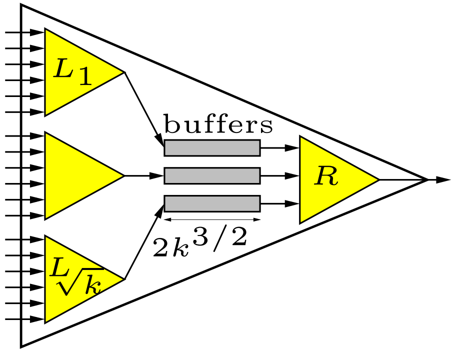
Cache-Oblivious Model

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



Examples

- Scanning, Linear time selection
- Matrix-transposition, FFT, Funnel-sorting



Lemma

Optimal cache-oblivious algorithm implies optimal algorithm on each level of a fully associative multi-level cache using LRU

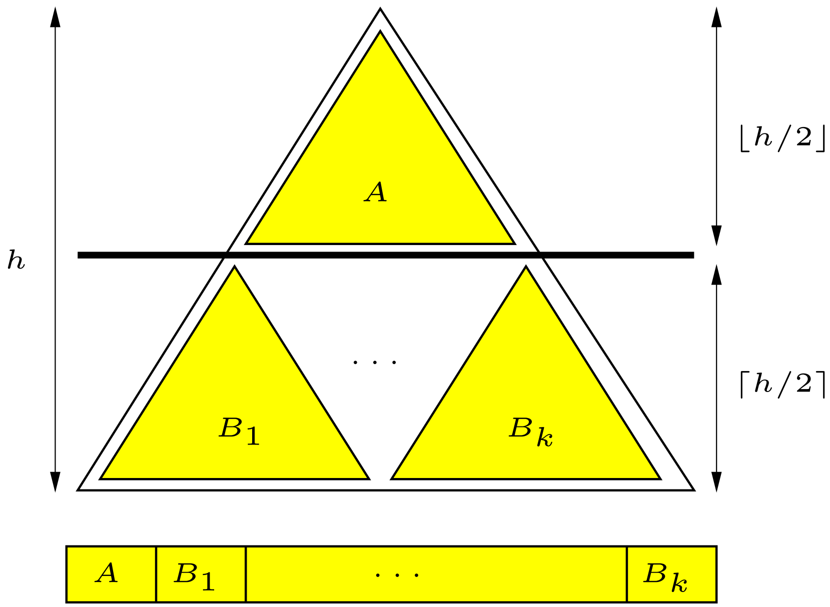
Frigo et al. 1999

Outline

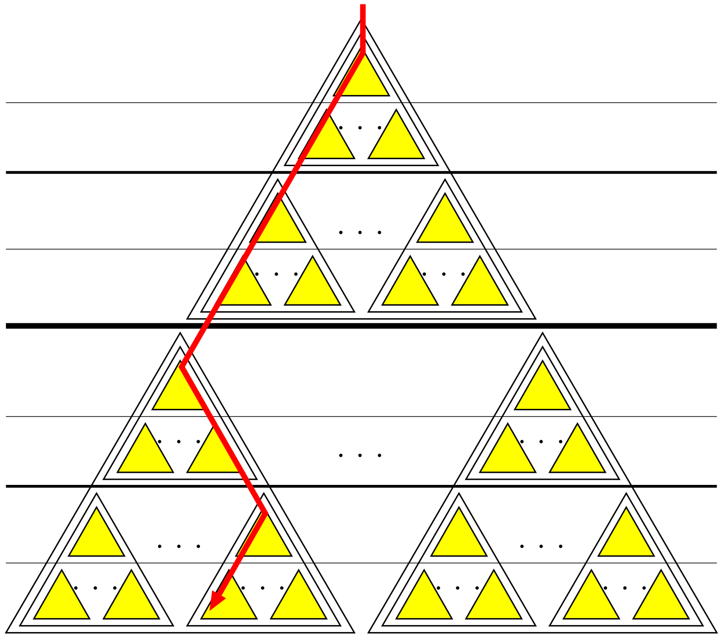
- Trends in Implementation Technology
- Models of Computation
  - I/O Model
  - Cache-Oblivious Model
- ▶ Cache-Oblivious Search Trees
  - Static
  - Dynamic
- Experiments
  - Memory Layouts of Trees
- Summary

Static Cache-Oblivious Trees

Recursive memory layout $\equiv$ van Emde Boas layout



Binary tree

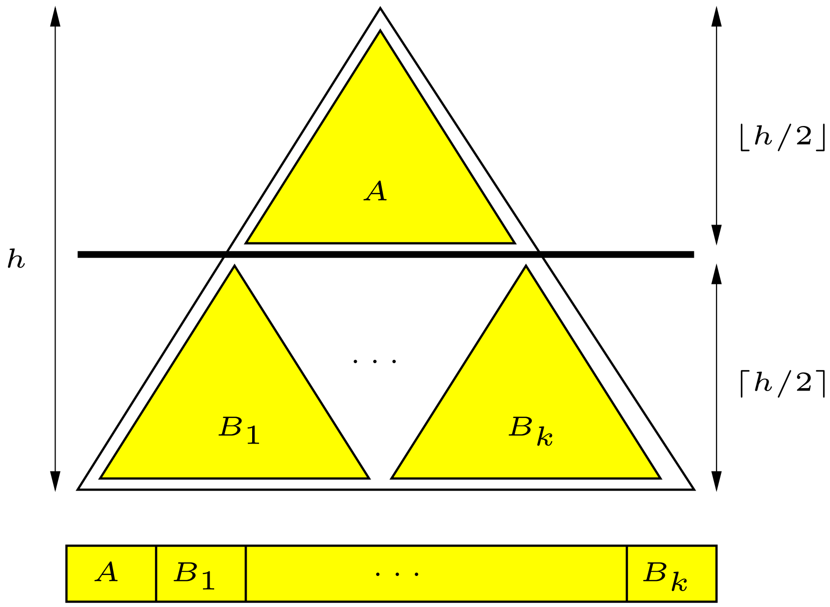


Searches use $O(\log_B N)$ I/Os

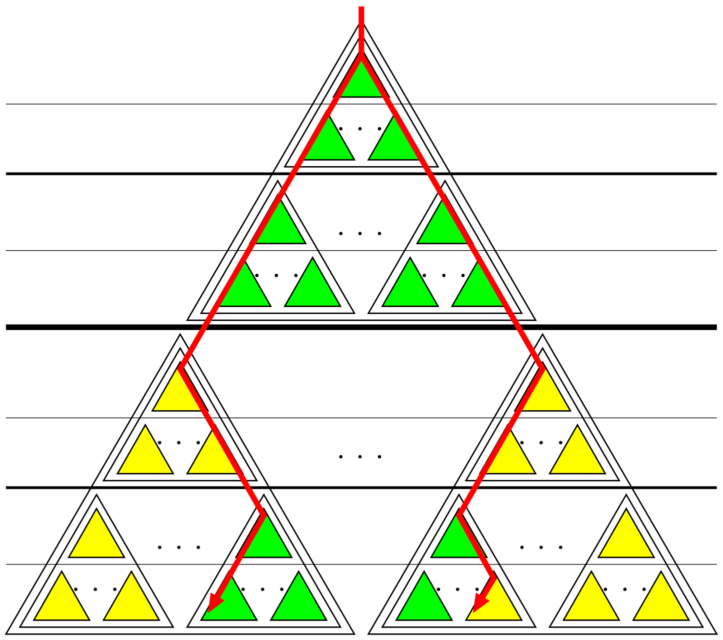
Prokop 1999

Static Cache-Oblivious Trees

Recursive memory layout $\equiv$ van Emde Boas layout



Binary tree



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

Range reportings use

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

Prokop 1999

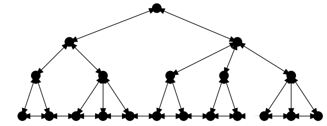
Dynamic Cache-Oblivious Trees

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

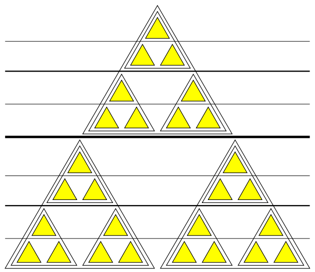
Dynamic Cache-Oblivious Trees

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

- Pointer Based Strongly Weight Balanced B-trees
- Dynamic van Emde Boas Layout
- Packed Memory Management
- Two Levels of Indirection

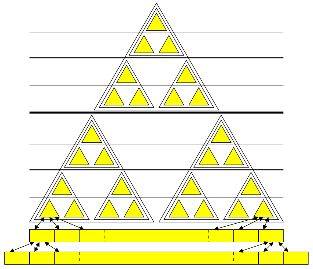


Arge and Vitter 1996



Prokop 1999

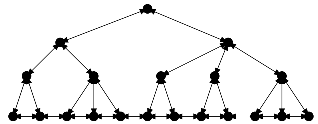
Itai et al. 1981



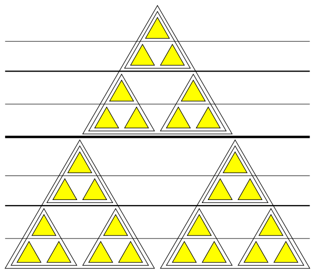
Bender, Demain, Farach-Colton 2000

Dynamic Cache-Oblivious Trees

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

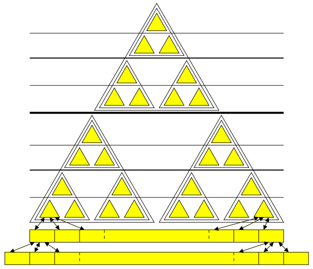


Arge and Vitter 1996



Prokop 1999

Itai et al. 1981

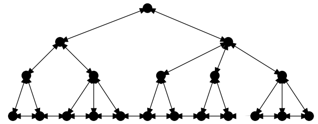


- Pointer Based ~~Strongly Weight Balanced B-trees~~
- Dynamic van Emde Boas Layout
- Packed Memory Management
- Two Levels of Indirection

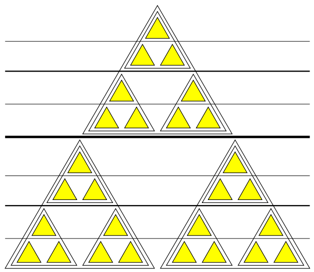
Bender, Demain, Farach-Colton 2000

Dynamic Cache-Oblivious Trees

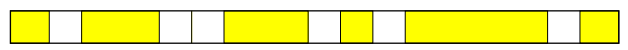
Search	$O(\log_B N)$
Range Reporting	$O\left(\log_B N + \frac{k}{B}\right)$
Updates	$O\left(\log_B N + \frac{\log^2 N}{B}\right)$



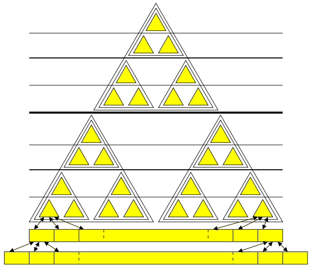
Arge and Vitter 1996



Prokop 1999



Itai et al. 1981

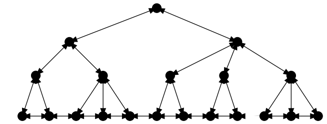


- Pointer Based ~~Strongly Weight Balanced B-trees~~
- ~~Dynamic~~ van Emde Boas Layout
- Packed Memory Management
- Two Levels of Indirection

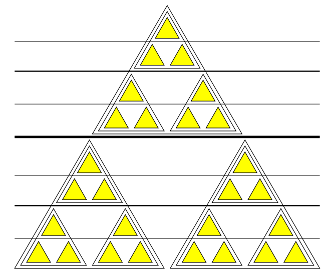
Bender, Demain, Farach-Colton 2000

Dynamic Cache-Oblivious Trees

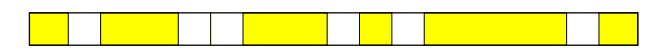
Search	$O(\log_B N)$
Range Reporting	$O\left(\log_B N + \frac{k}{B}\right)$
Updates	$O\left(\log_B N + \frac{\log^2 N}{B}\right)$



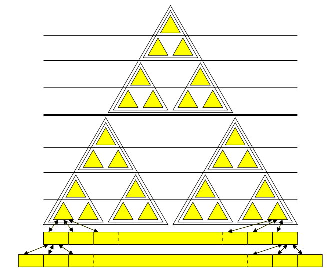
Arge and Vitter 1996



Prokop 1999



Itai et al. 1981

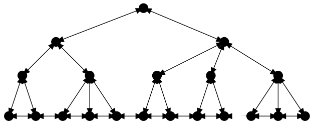


- Pointer Based Strongly Weight Balanced B-trees
- Dynamic van Emde Boas Layout
- Packed Memory Management
- Two Levels of Indirection

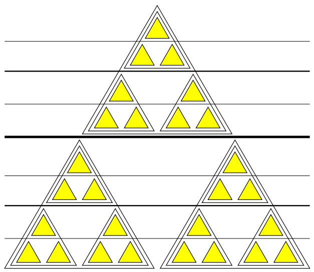
Bender, Demain, Farach-Colton 2000

Dynamic Cache-Oblivious Trees

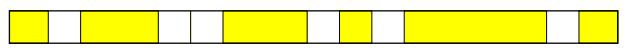
Search	$O(\log_B N)$
Range Reporting	$O\left(\log_B N + \frac{k}{B}\right)$
Updates	$O\left(\log_B N + \frac{\log^2 N}{B}\right)$



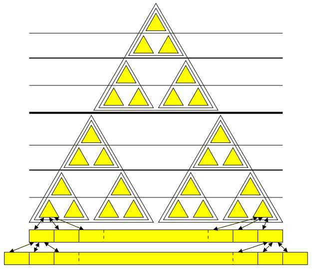
Arge and Vitter 1996



Prokop 1999



Itai et al. 1981

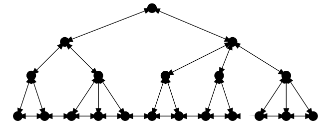


- ~~Pointer Based~~ ~~Strongly Weight Balanced B-trees~~
- ~~Dynamic~~ van Emde Boas Layout
- Packed Memory Management
- ~~Two Levels of Indirection~~

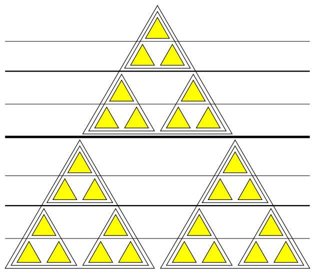
Bender, Demain, Farach-Colton 2000

Dynamic Cache-Oblivious Trees

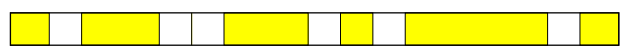
Search	$O(\log_B N)$
Range Reporting	$O\left(\log_B N + \frac{k}{B}\right)$
Updates	$O\left(\log_B N + \frac{\log^2 N}{B}\right)$



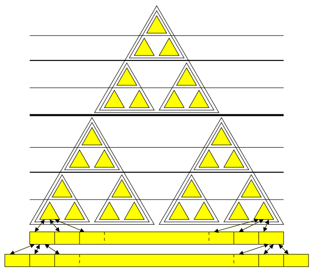
Arge and Vitter 1996



Prokop 1999



Itai et al. 1981

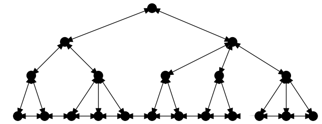


- ~~Pointer Based~~ ~~Strongly Weight Balanced B-trees~~
- ~~Dynamic~~ van Emde Boas Layout
- ~~Packed Memory Management~~
- ~~Two Levels of Indirection~~

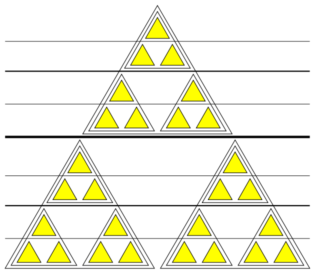
Bender, Demain, Farach-Colton 2000

Dynamic Cache-Oblivious Trees

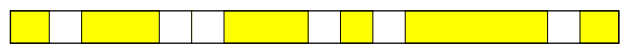
Search	$O(\log_B N)$
Range Reporting	$O\left(\log_B N + \frac{k}{B}\right)$
Updates	$O\left(\log_B N + \frac{\log^2 N}{B}\right)$



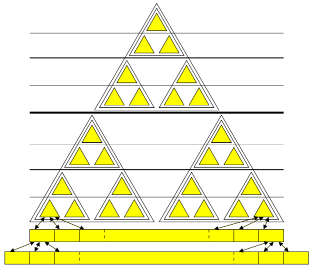
Arge and Vitter 1996



Prokop 1999



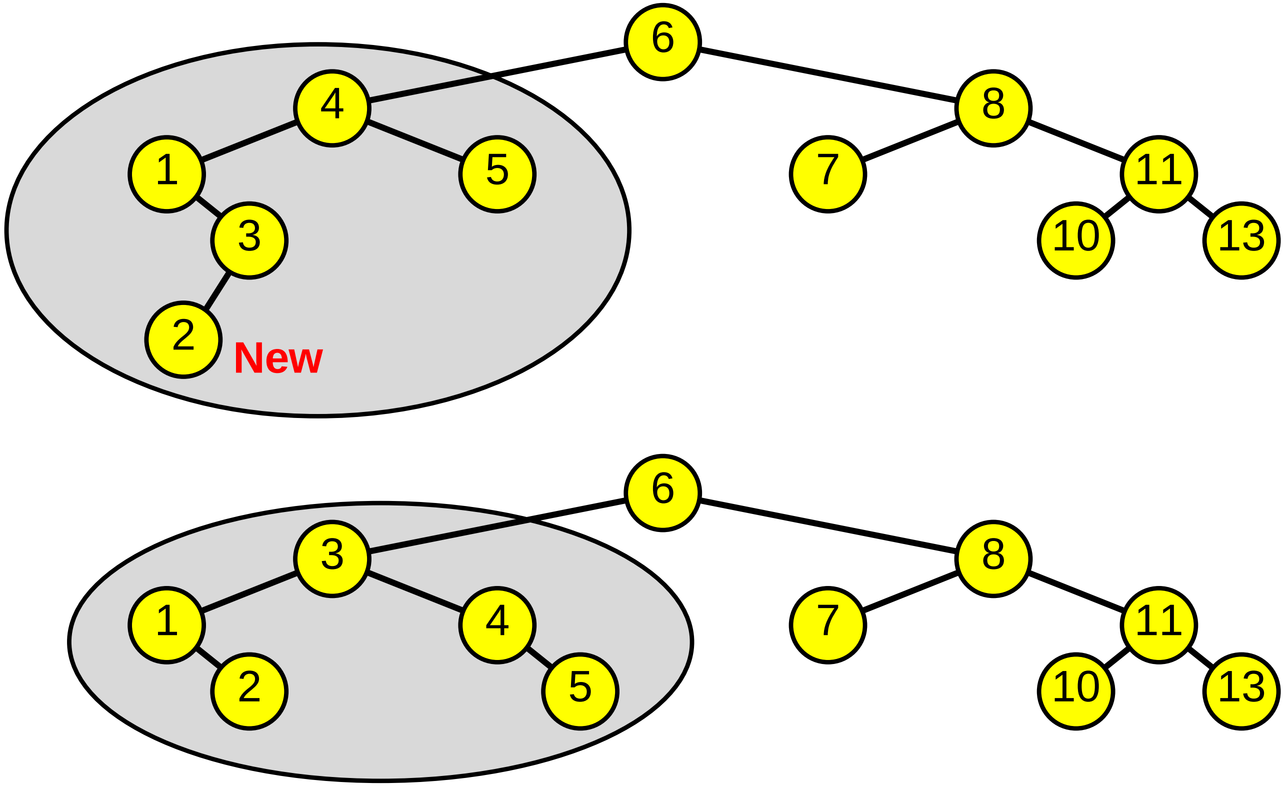
Itai et al. 1981



- Pointer Based Strongly Weight Balanced B-trees
- Dynamic van Emde Boas Layout
- Packed Memory Management Trees of Small Height
- Two Levels of Indirection

Bender, Demain, Farach-Colton 2000

Binary Trees of Small Height

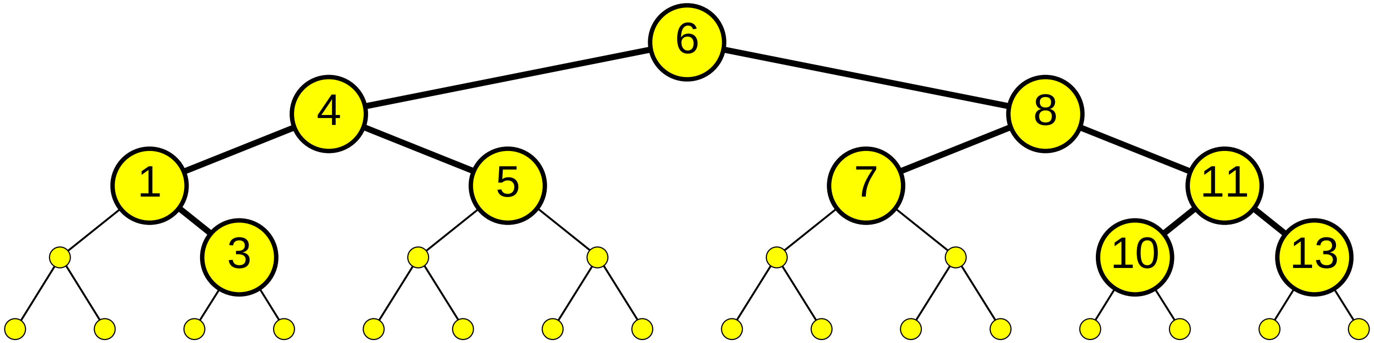


- If an insertion causes non-small height then rebuild subtree at nearest ancestor with sufficient few descendents
- Insertions require amortized time $\mathrm{O}(\log^2 N)$

Andersson and Lai 1990

Dynamic Cache-Oblivious Trees

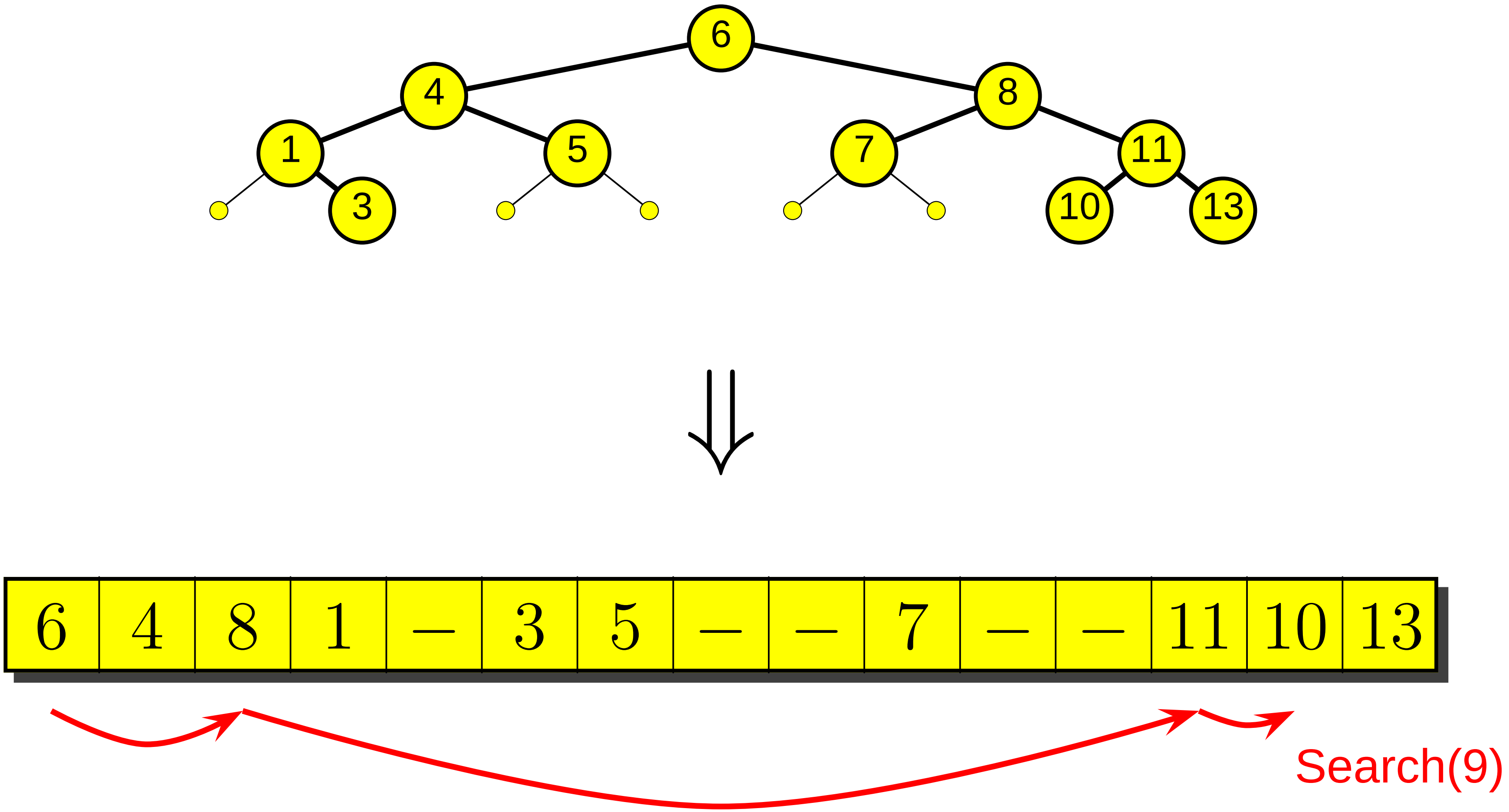
- Embed a dynamic tree of small height into a complete tree
- Static van Emde Boas layout



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

New

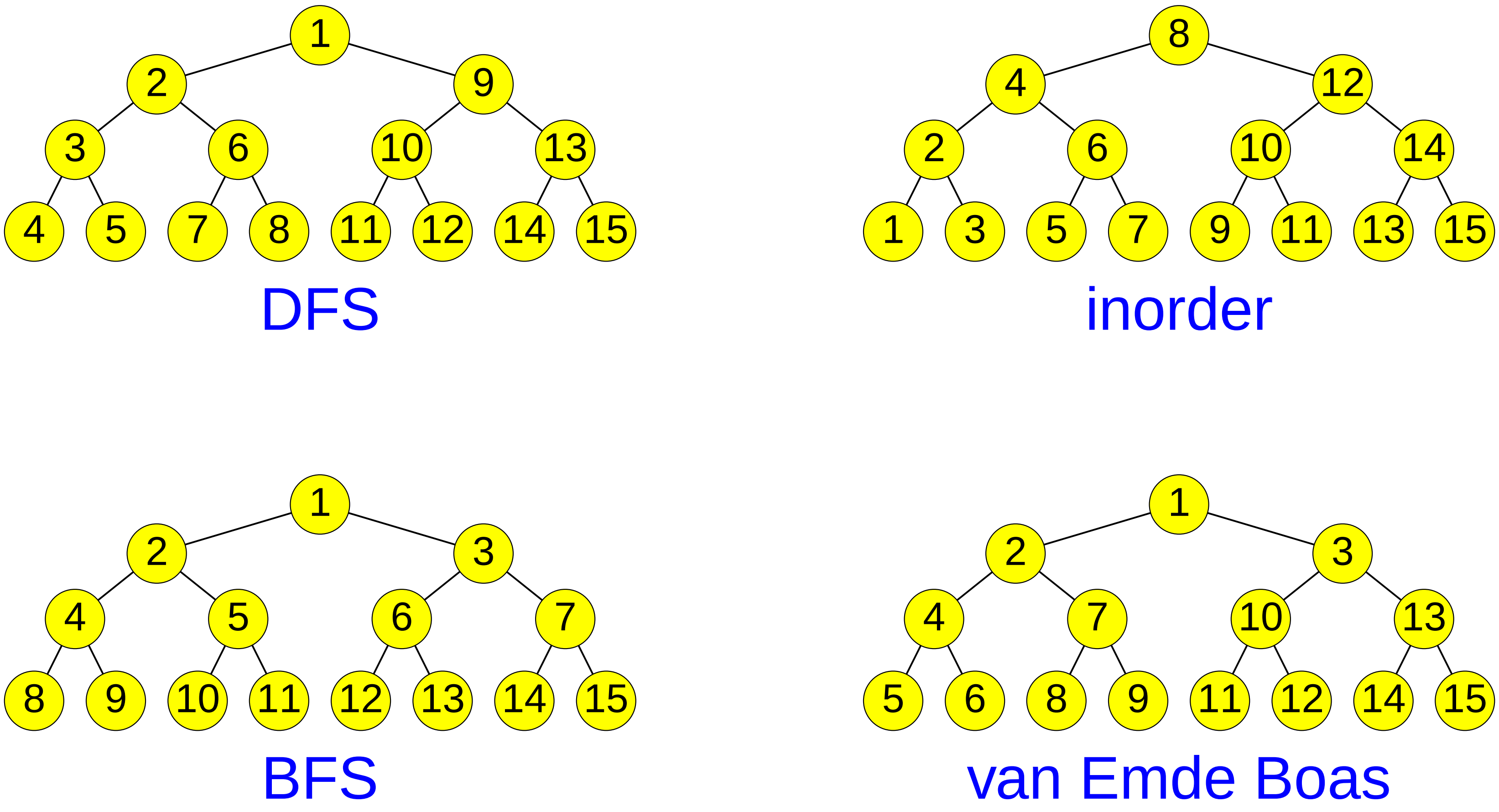
Example



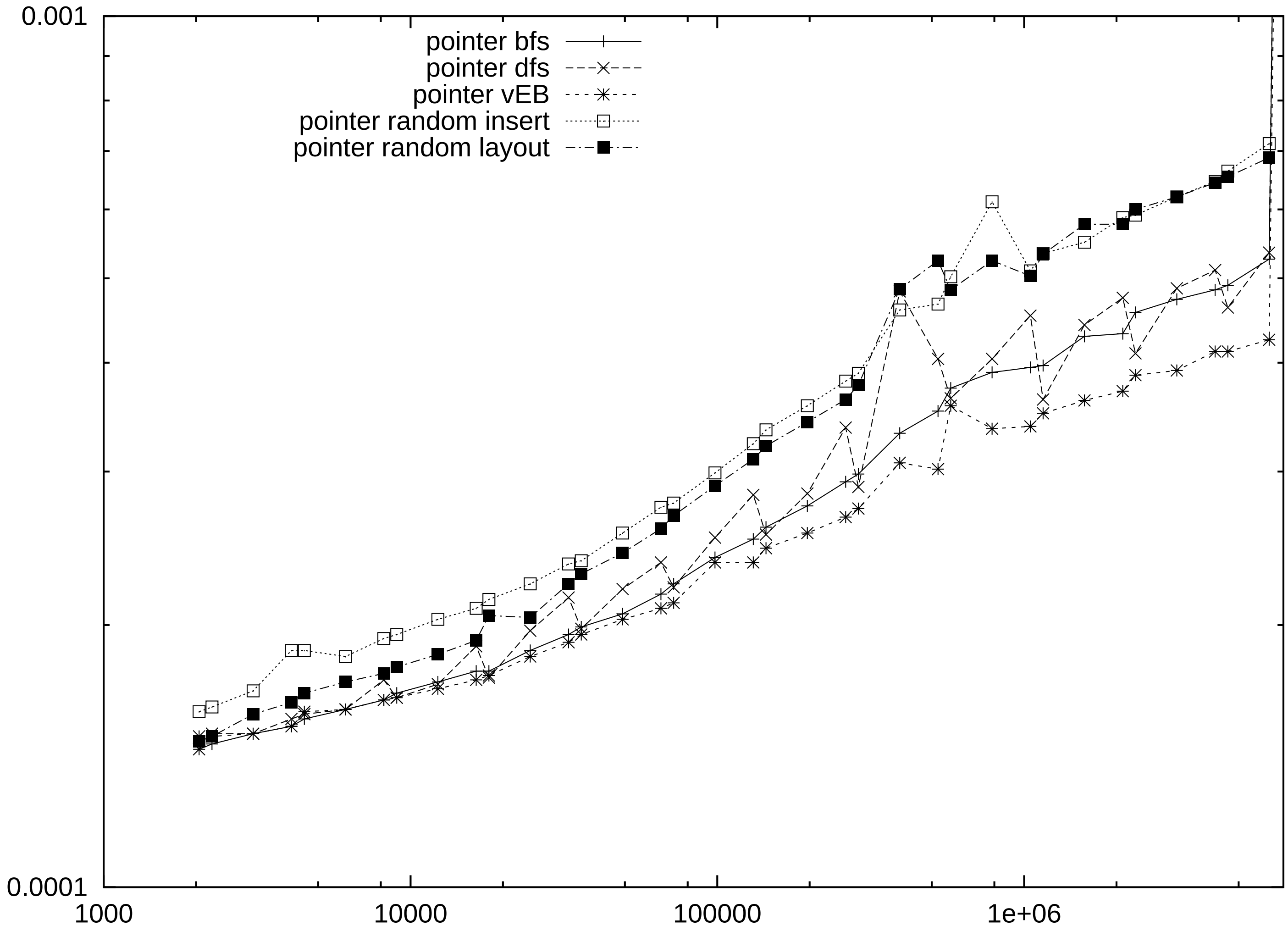
Outline

- Trends in Implementation Technology
- Models of Computation
  - I/O Model
  - Cache-Oblivious Model
- Cache-Oblivious Search Trees
  - Static
  - Dynamic
- Experiments
  - Memory Layouts of Trees
- Summary

Memory Layouts of Trees

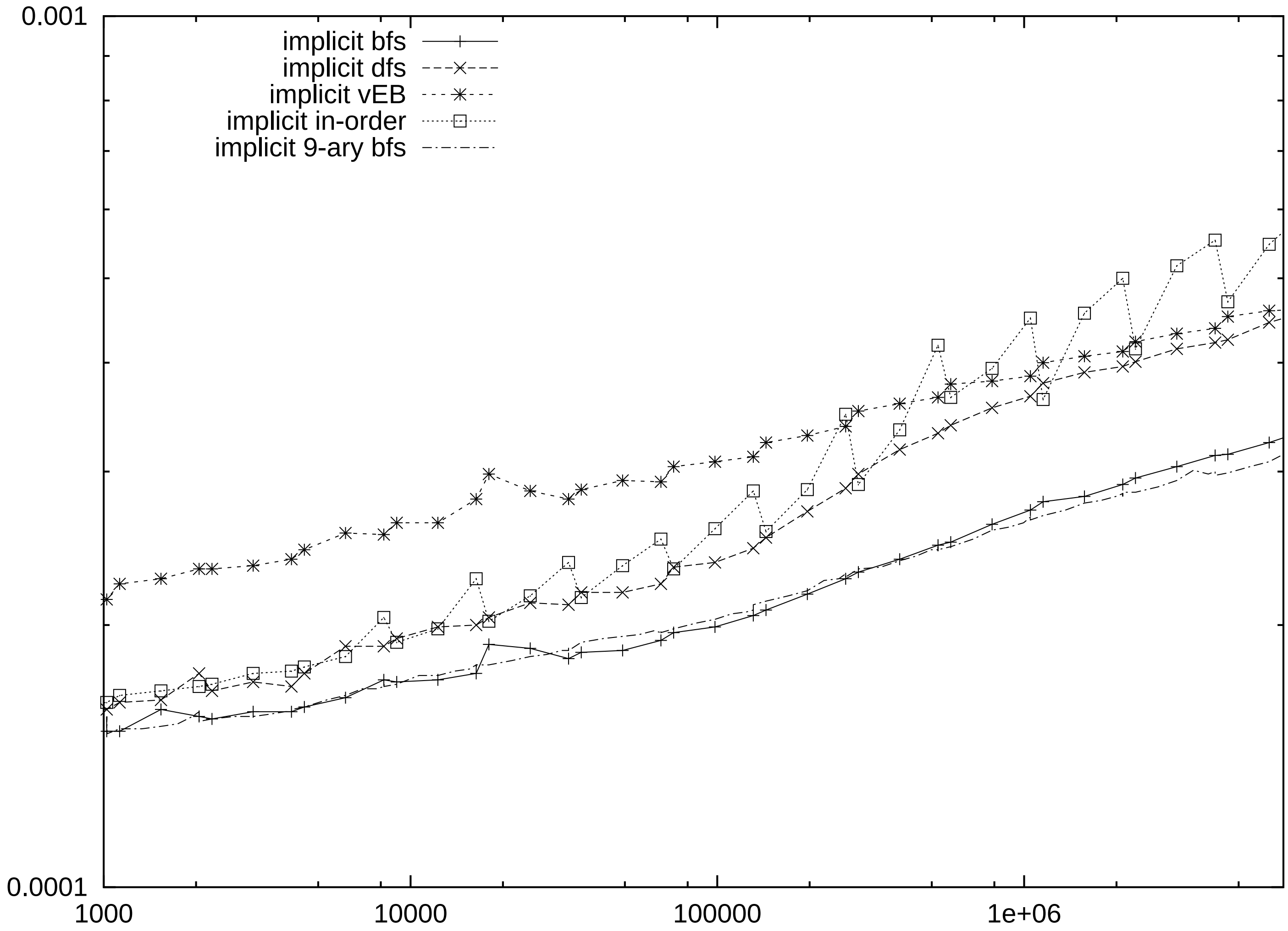


Searches in Pointer Based Layouts



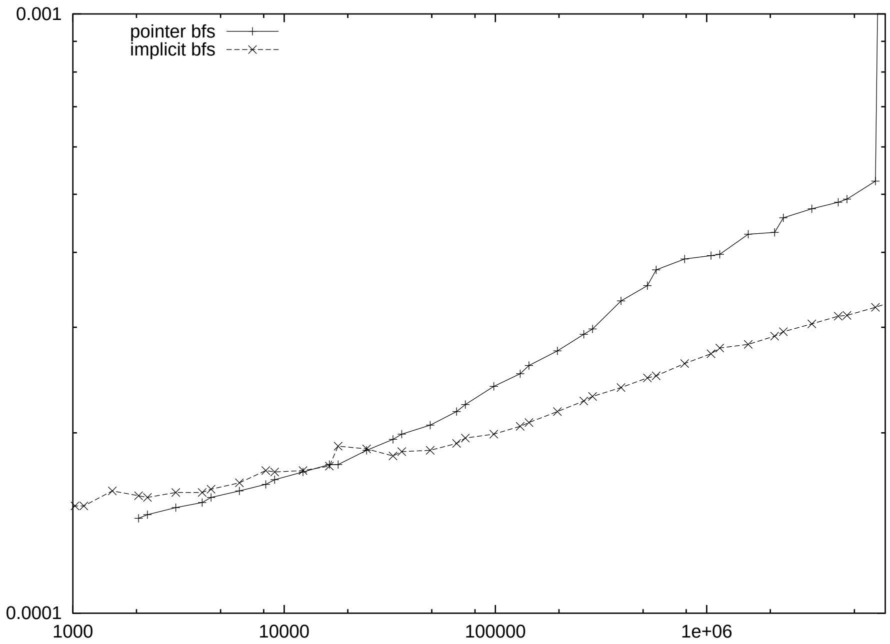
- van Emde Boas layout wins, followed by the BFS layout

Searches with Implicit Layouts

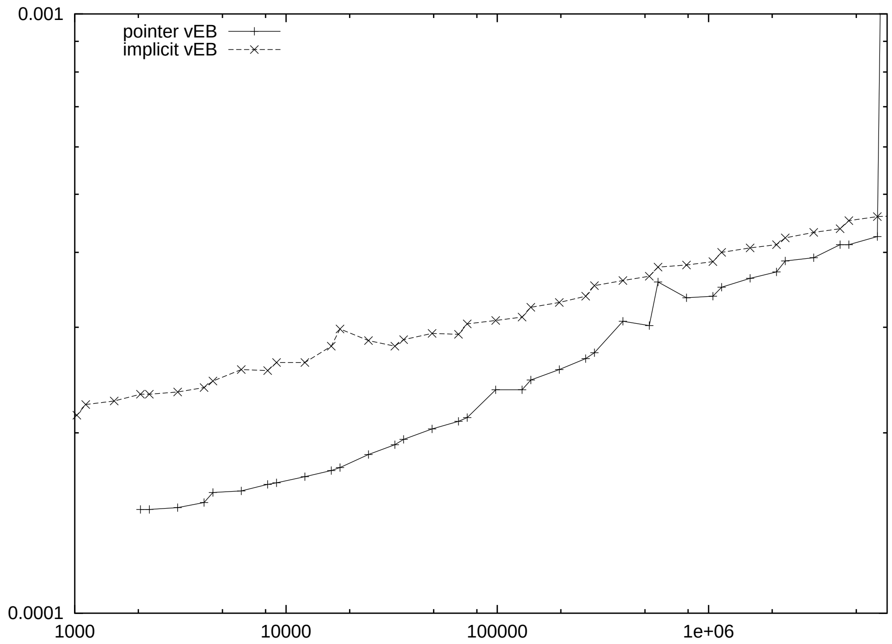


- BFS layout wins due to simplicity and caching of topmost levels
- van Emde Boas layout requires quite complex index computations

Implicit vs Pointer Based Layouts



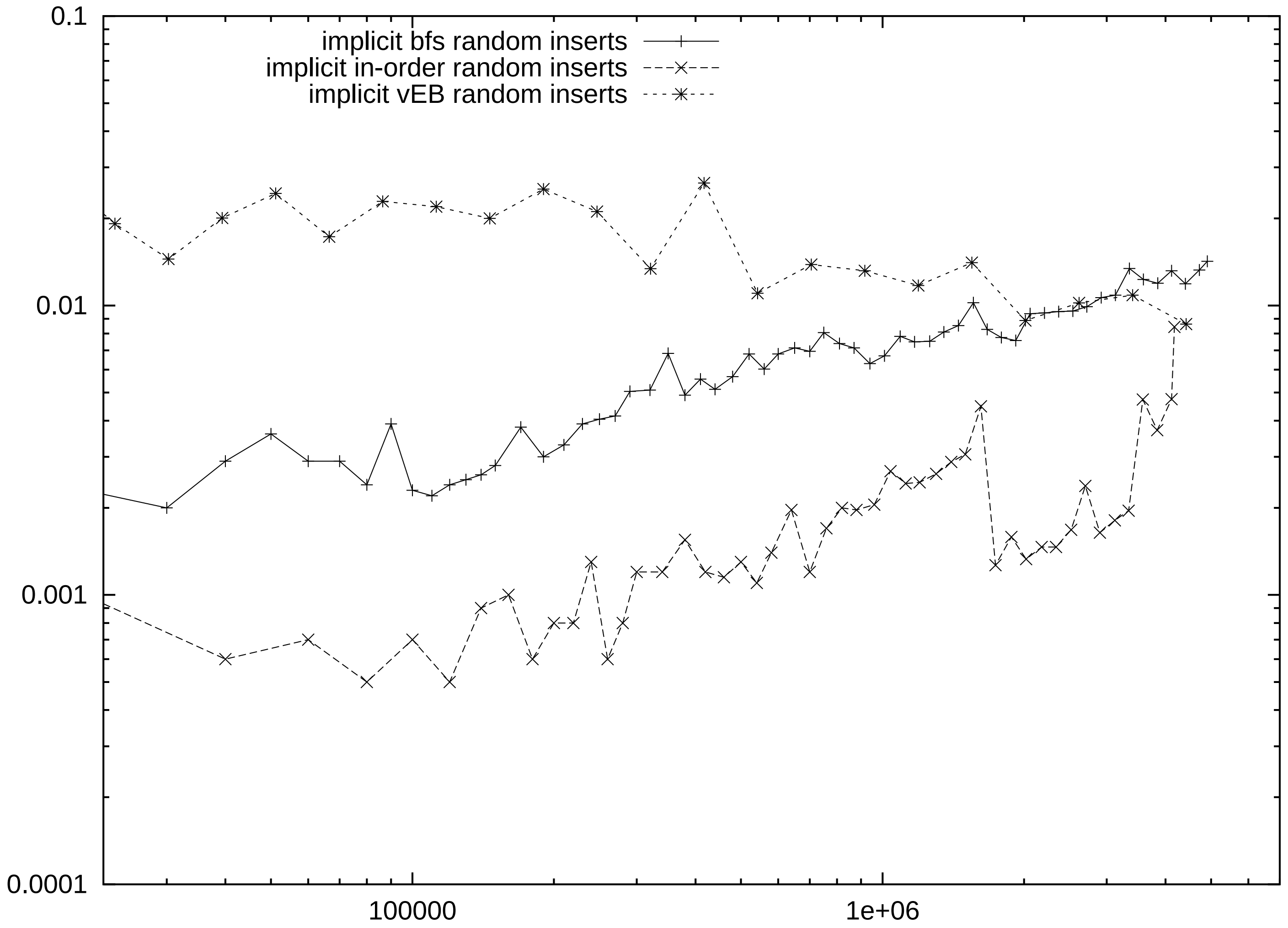
BFS layout



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

- Simple cache-oblivious search trees

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

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