Coping with the Memory Hierarchy the Cache-Oblivious Way

Rolf Fagerberg

University of Aarhus

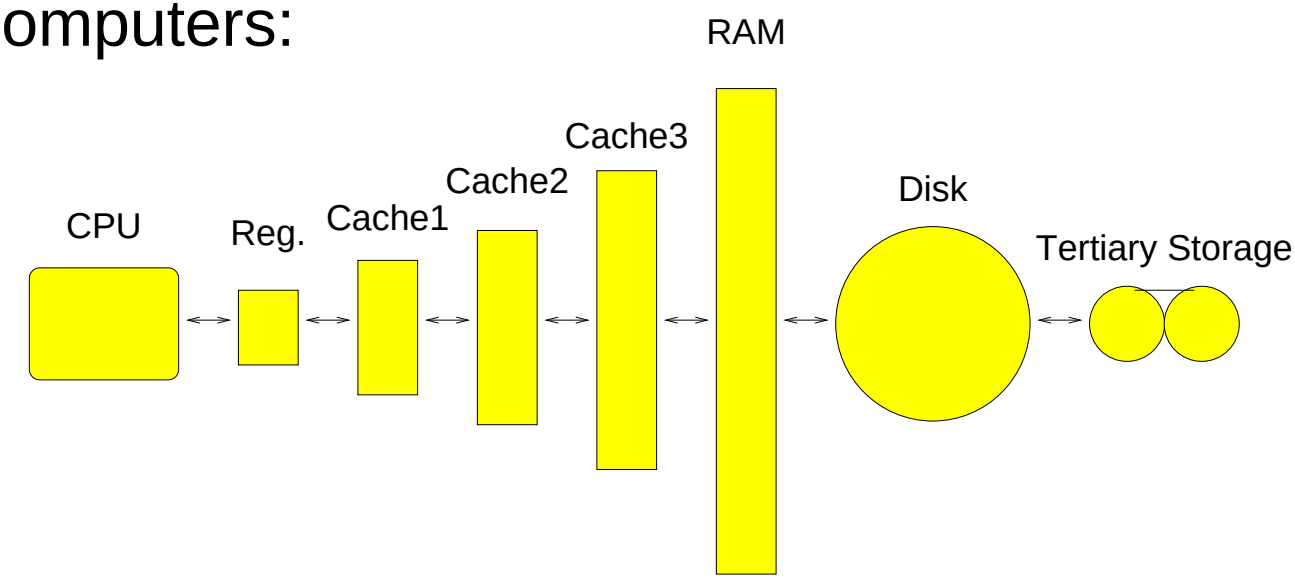
Imada, SDU, February 18, 2004

Overview

- The memory hierachy
- The I/O-model
- The cache-oblivious model
- Examples of cache-oblivious algorithms
  - Double for-loop (with applications)
  - Searching
  - Sorting
- Theoretical limits of cache-obliviousness

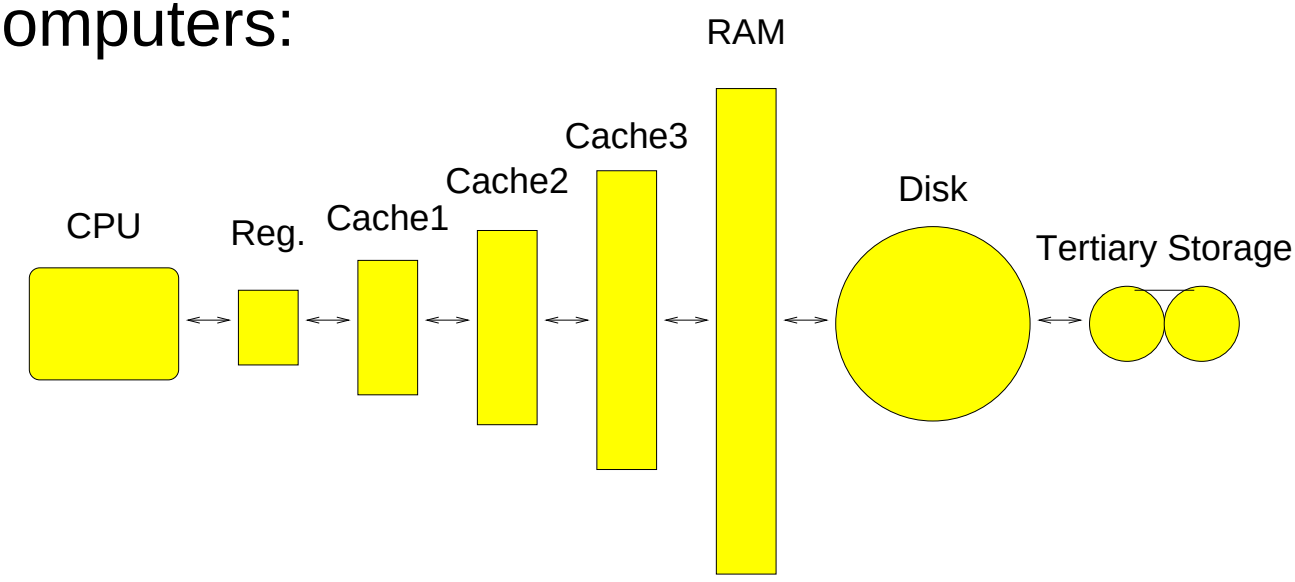
The Memory Hierarchy

Modern computers:



The Memory Hierarchy

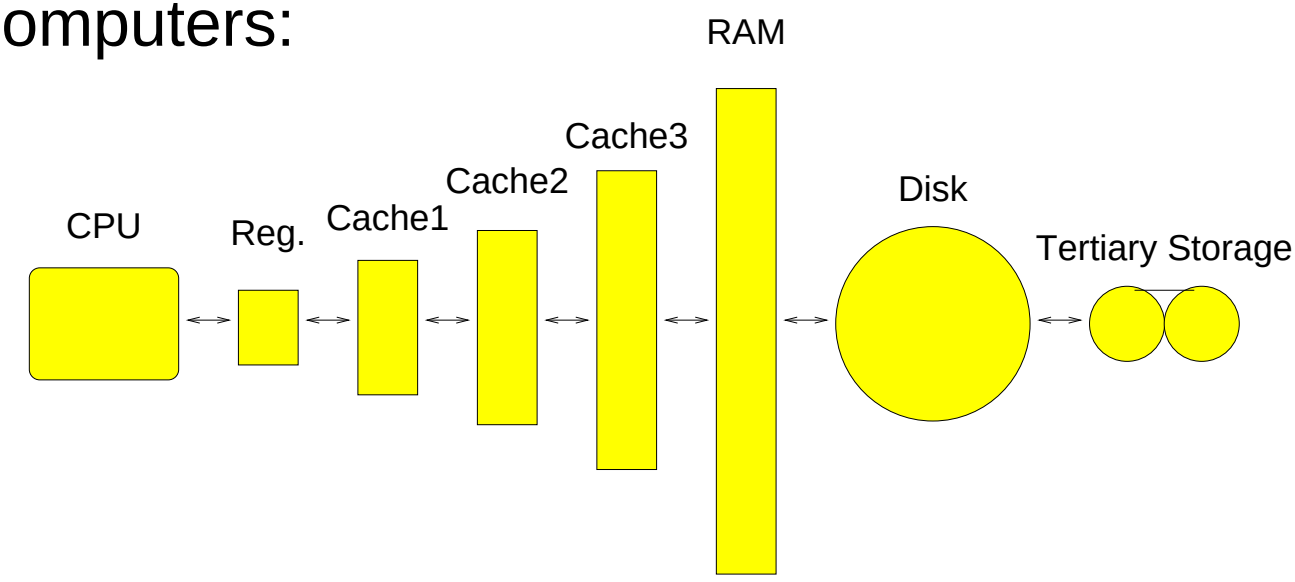
Modern computers:



	Access time	*Volume*
Registers	1 cycle	1 Kb
Cache	10 cycles	512 Kb
RAM	100 cycles	512 Mb
Disk	20,000,000 cycles	80 Gb

The Memory Hierarchy

Modern computers:



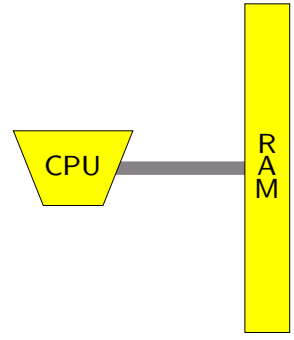
	Access time	*Volume*
Registers	1 cycle	1 Kb
Cache	10 cycles	512 Kb
RAM	100 cycles	512 Mb
Disk	20,000,000 cycles	80 Gb

Gap increases over time.

Real problems of Gigabyte, Terabyte, and even Petabyte size: Databases (finance, phone companies, banks, weather, geology, geography, astronomy), WWW, GIS systems, computer graphics.

Classic RAM Model

The RAM model:



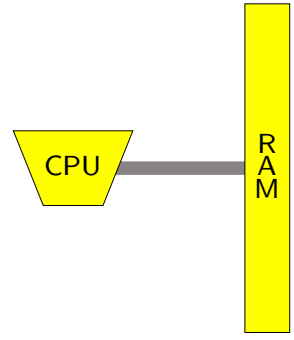
Add: $O(1)$

Branch: $O(1)$

Mem access: $O(1)$

Classic RAM Model

The RAM model:



Add: $O(1)$

Branch: $O(1)$

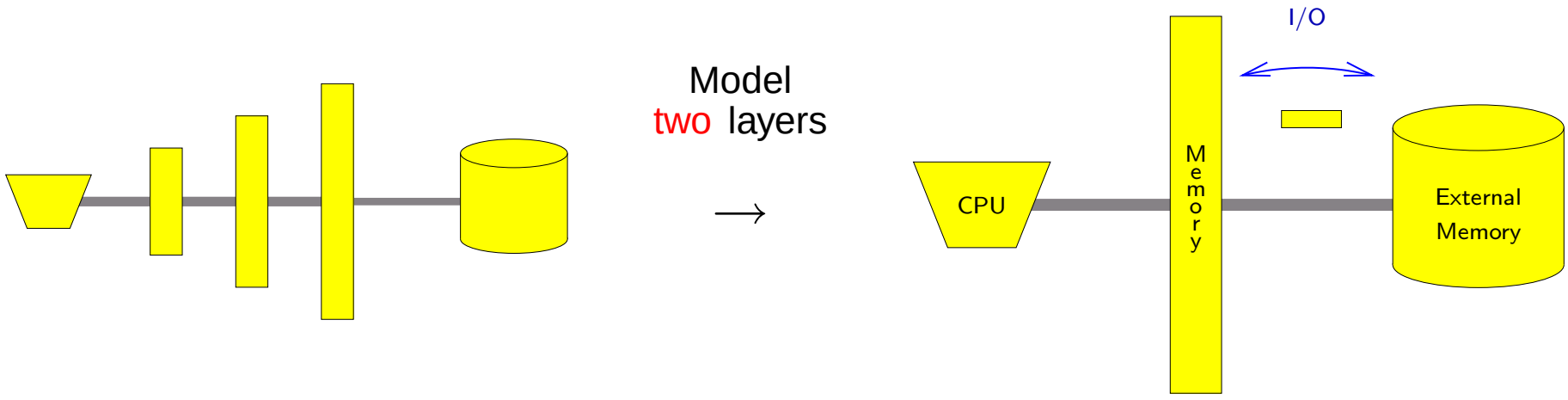
~~Mem access: $O(1)$~~

Increasingly inadequate

Overview

- ✓ The memory hierachy
- The I/O-model
- The cache-oblivious model
- Examples of cache-oblivious algorithms
  - Double for-loop (with applications)
  - Searching
  - Sorting
- Theoretical limits of cache-obliviousness

I/O Model



N = problem size

M = memory size

B = I/O block size

Aggarwal and Vitter 1988

Cost: number of I/Os.

Example

	CPU time	Inplace	Worstcase
Heapsort	$N \log N$	$\surd$	$\surd$
Quicksort	$N \log N$	$\surd$	
Mergesort	$N \log N$		$\surd$

Example

	CPU time	Inplace	Worstcase	I/O
Heapsort	$N \log N$	$\surd$	$\surd$	$N \log N$
Quicksort	$N \log N$	$\surd$		$(N \log N)/B$
Mergesort	$N \log N$		$\surd$	$(N \log N)/B$

Random memory access $\Rightarrow$ page fault at every access.

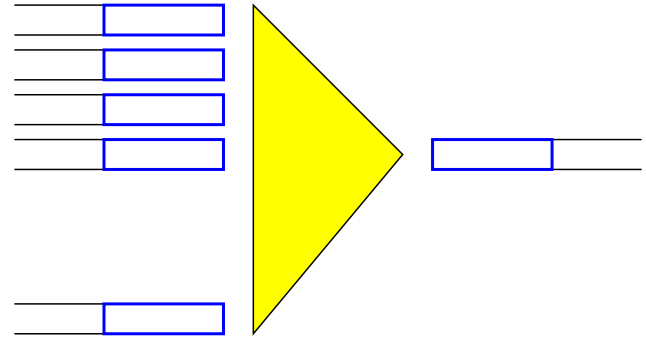
Sequential memory access $\Rightarrow$ page fault every B accesses.

Typically, $B \sim 10^3$

I/O-Optimal Sorting

Binary Mergesort: $\frac{N}{B} \log_2 N$ I/Os

Multi-Way Merging:



Maximal merge degree $\approx M/B$

Multi-Way Mergesort: $\frac{N}{B} \log_{M/B} \frac{N}{M}$ I/Os

I/O Model Facts

- Scanning: $\Theta(N/B)$ I/Os.
- Searching: $\Theta(\log_B N)$ I/Os by B-trees.
- Sorting: $\Theta\left(\frac{N}{B}\log_{M/B}\frac{N}{M}\right)$ I/Os by $\frac{M}{B}$-way merge-sort.
- Permuting: $\Theta\left(\min\{N, \frac{N}{B}\log_{M/B}\frac{N}{M}\}\right)$ by direct move or sorting

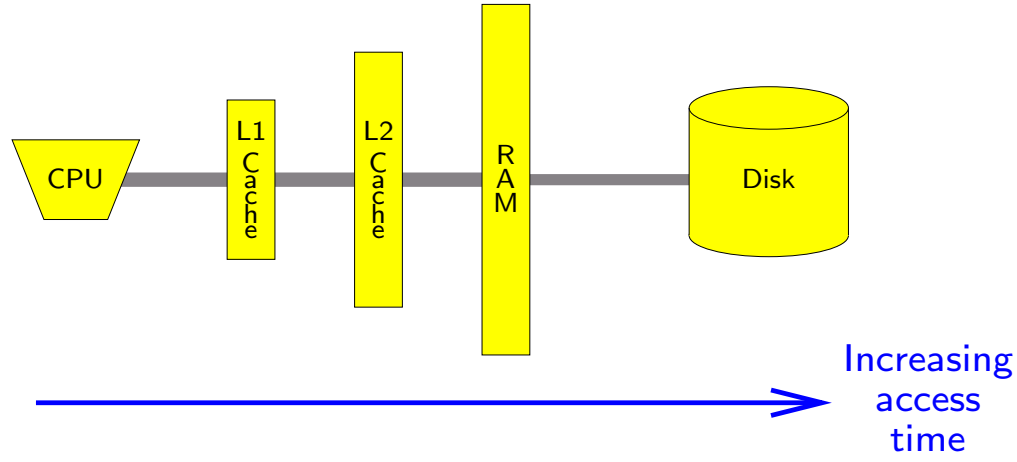
1988-2004:
Many algorithms and data structures for problems from computational geometry, graphs, strings, ...

Overview

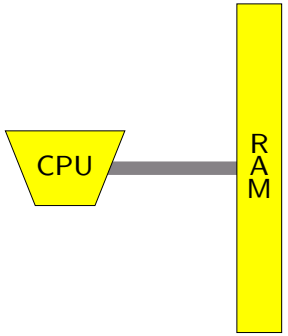
- √ The memory hierachy
- √ The I/O-model
- The cache-oblivious model
- Examples of cache-oblivious algorithms
  - Double for-loop (with applications)
  - Searching
  - Sorting
- Theoretical limits of cache-obliviousness

Computer Models

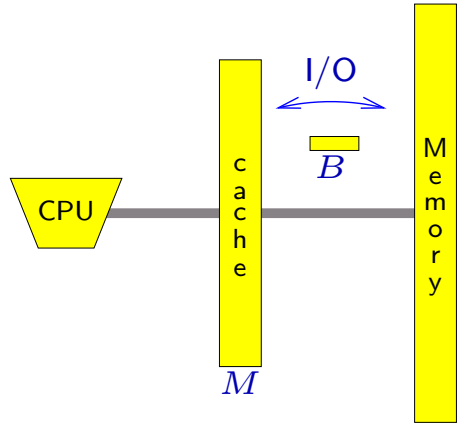
Reality:



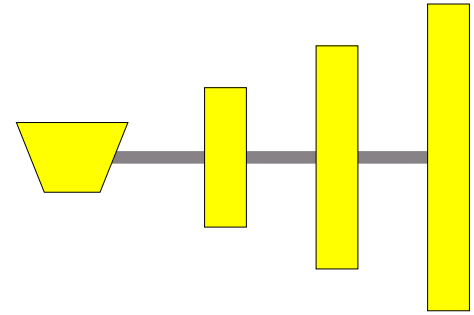
Models:



RAM model



I/O model



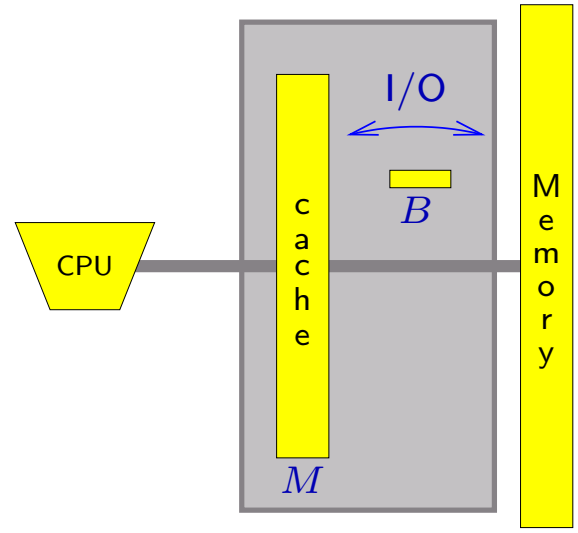
Multi-level models

Cache-Oblivious-ness

New Model

Cache-Oblivious Model

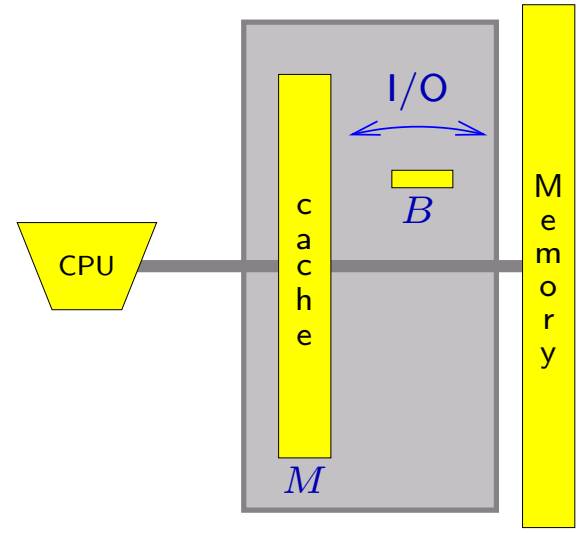
- Program in the RAM model
- Analyze in the I/O model for arbitrary B and M
- Optimal off-line cache replacement strategy



Frigo, Leiserson, Prokop, Ramachandran, FOCS'99

Cache-Oblivious Model

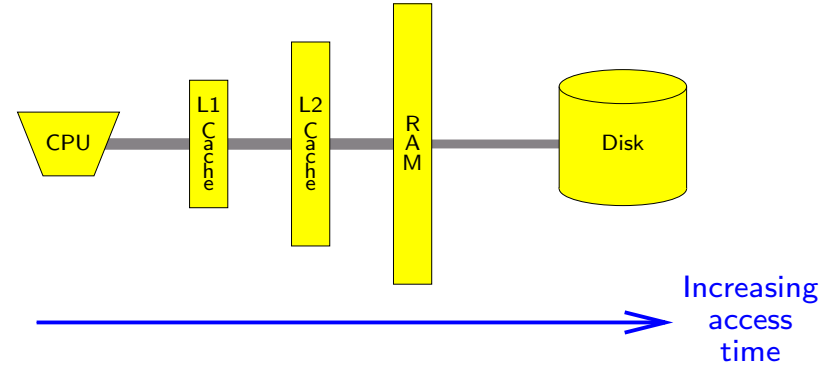
- Program in the RAM model
- Analyze in the I/O model for arbitrary B and M
- Optimal off-line cache replacement strategy



Frigo, Leiserson, Prokop, Ramachandran, FOCS'99

Advantages:

- Optimal on arbitrary level $\Rightarrow$ optimal on all levels
- Portability
- Simplicity of model.



Cache-Oblivious Results

Scanning $\Rightarrow$ stack, queue, selection,. . . .

Cache-Oblivious Results

Scanning $\Rightarrow$ stack, queue, selection,. . . .	
Matrix multiplication, FFT:	FOCS'99
Sorting:	FOCS'99, ICALP'02, ALENEX'04
Search trees:	Prokop 99, FOCS'00, WAE'01, SODA'02 $\times$ 2, ESA'02, FOCS'03
Priority queues:	STOC'02, ISAAC'02
Graph algorithms:	STOC'02, BRICS-04-2
Computational geometry:	2 $\times$ ICALP'02 , SCG'03
Scanning dynamic sets:	ESA'02
Power of cache-obliviousness:	STOC'03

Cache-Oblivious Results

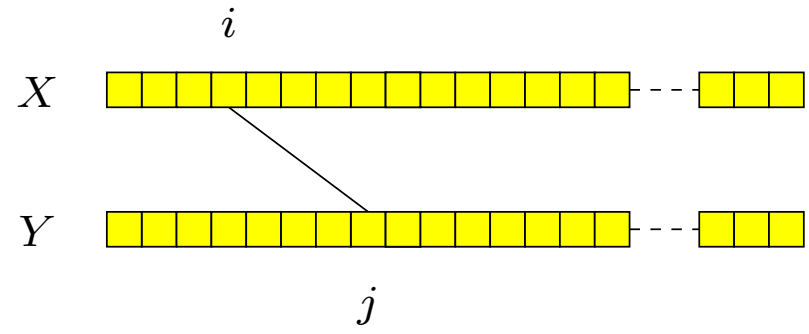
Scanning $\Rightarrow$ stack, queue, selection,. . . .	
Matrix multiplication, FFT:	FOCS'99
Sorting:	FOCS'99, ICALP'02, ALENEX'04
Search trees:	Prokop 99, FOCS'00, WAE'01, SODA'02 $\times$ 2, ESA'02, FOCS'03
Priority queues:	STOC'02, ISAAC'02
Graph algorithms:	STOC'02, BRICS-04-2
Computational geometry:	2 $\times$ ICALP'02 , SCG'03
Scanning dynamic sets:	ESA'02
Power of cache-obliviousness:	STOC'03

Overview

- √ The memory hierachy
- √ The I/O-model
- √ The cache-oblivious model
- Examples of cache-oblivious algorithms
  - Double for-loop (with applications)
  - Searching
  - Sorting
- Theoretical limits of cache-obliviousness

Double for-loop

X, Y arrays of length n:



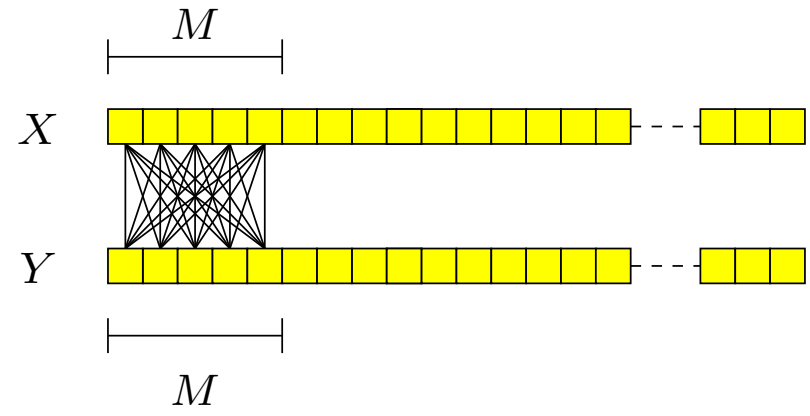

```
for(i=0; i<n; i++)
   for(j=0; j<n; j++)
      f(X[i], Y[j])
```

I/O complexity:

$$n \times \frac{n}{B} = \frac{n^2}{B}$$

Double for-loop

More efficient version
in the I/O-model:



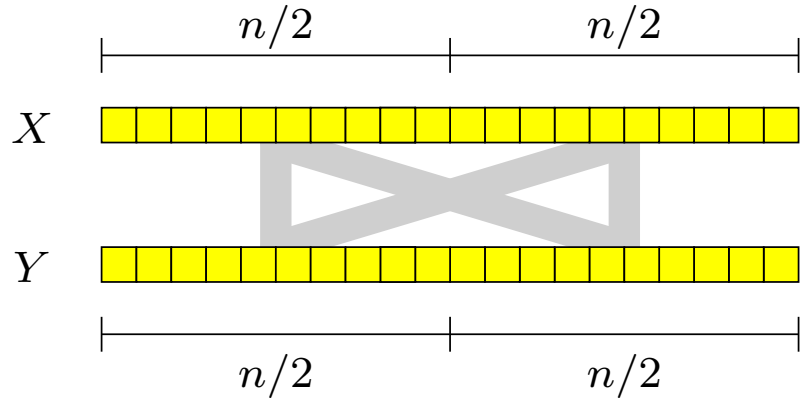
I/O complexity:

$$\frac{n}{M} \times \frac{n}{M} \times \frac{M}{B} = \frac{n^2}{MB}$$

```
for(i=0; i<n; i=i+M)
   for(j=0; j<n; j++)
      for(k=i; k<i+M; k++)
         f(X[i], Y[j])
```

Double for-loop

Cache-oblivious version:



+ recursion

I/O complexity:

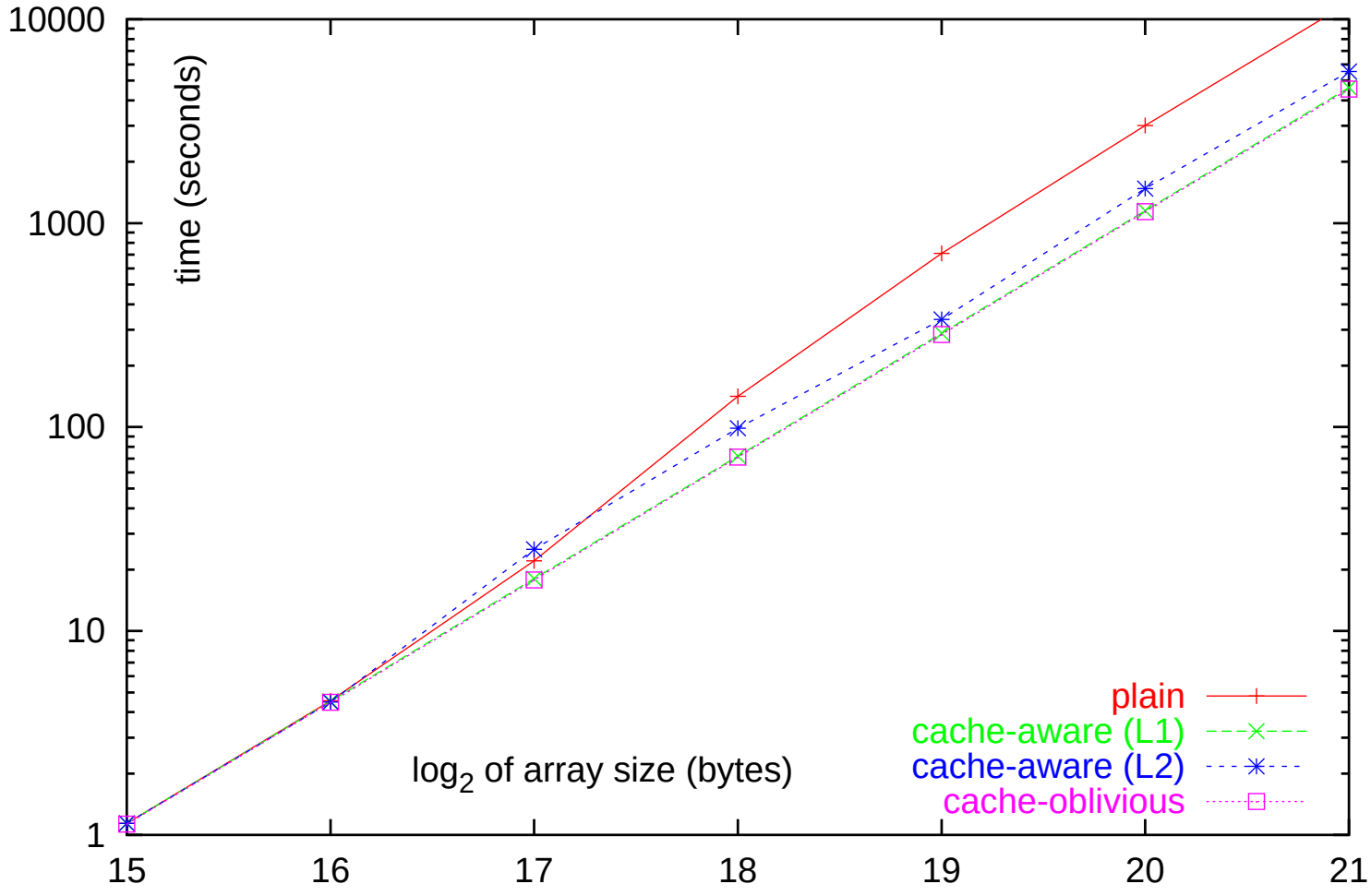
Again $\frac{n^2}{MB}$

Double for-loop

Cache-oblivious version

```
doubleLoop(i, j, length)
    if (length == 1)
        f(X[i], Y[j])
    else
        doubleLoop(i, j, length/2)
        doubleLoop(i, j+length/2, length/2)
        doubleLoop(i+length/2, j, length/2)
        doubleLoop(i+length/2, j+length/2, length/2)
```

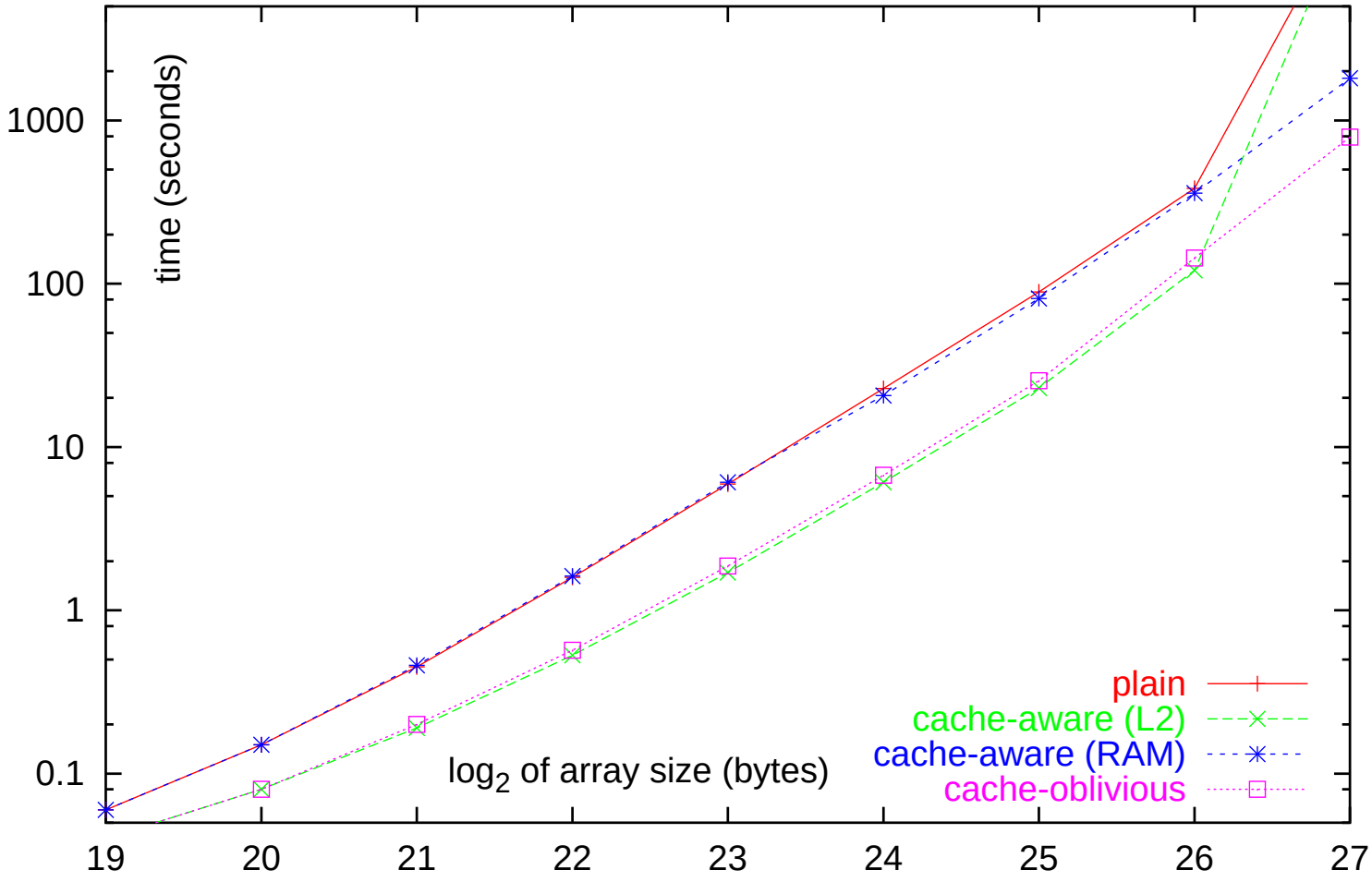
Experiments



Sizes within RAM (element size 4 bytes)

366 MHz Pentium II, 128 MB RAM, 256 KB Cache, gcc -O3, Linux

Experiments



Sizes exceeding RAM (element size 1 KB)

366 MHz Pentium II, 128 MB RAM, 256 KB Cache, gcc -O3, Linux

For-loop Applications

Join in databases

Dynamic programming
(bioinformatics)

Matrix multiplication
(scientific computing)

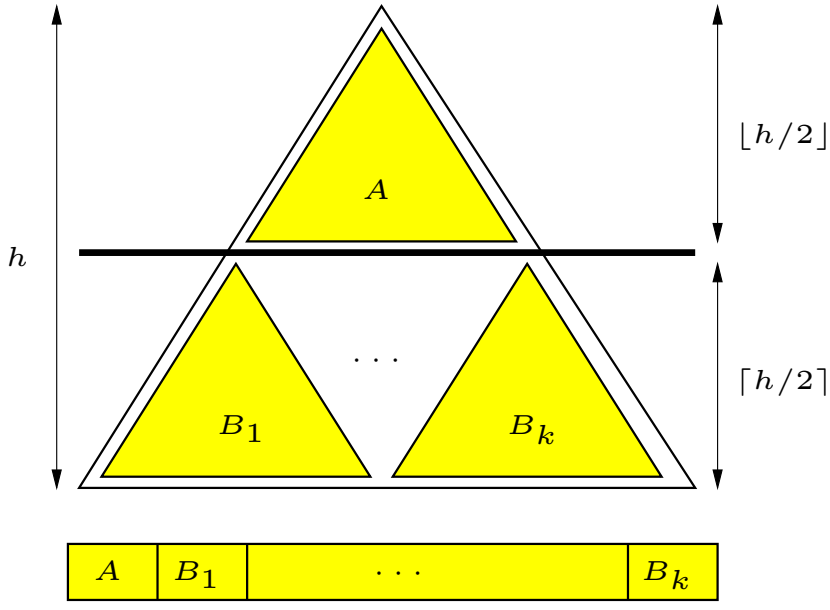
Overview

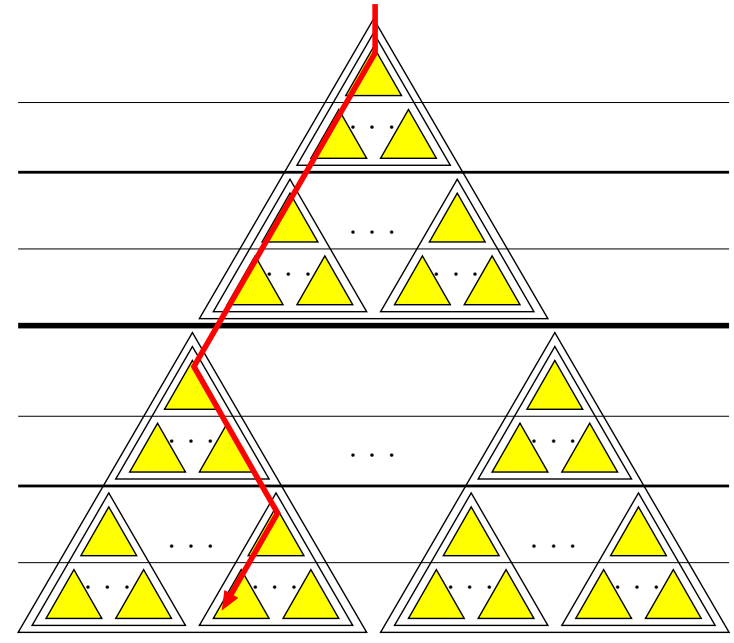
- √ The memory hierachy
- √ The I/O-model
- √ The cache-oblivious model
- Examples of cache-oblivious algorithms
  - √ Double for-loop (with applications)
  - Searching
  - Sorting
- Theoretical limits of cache-obliviousness

Static Cache-Oblivious Trees

Recursive memory layout (van Emde Boas layout)

Prokop 1999





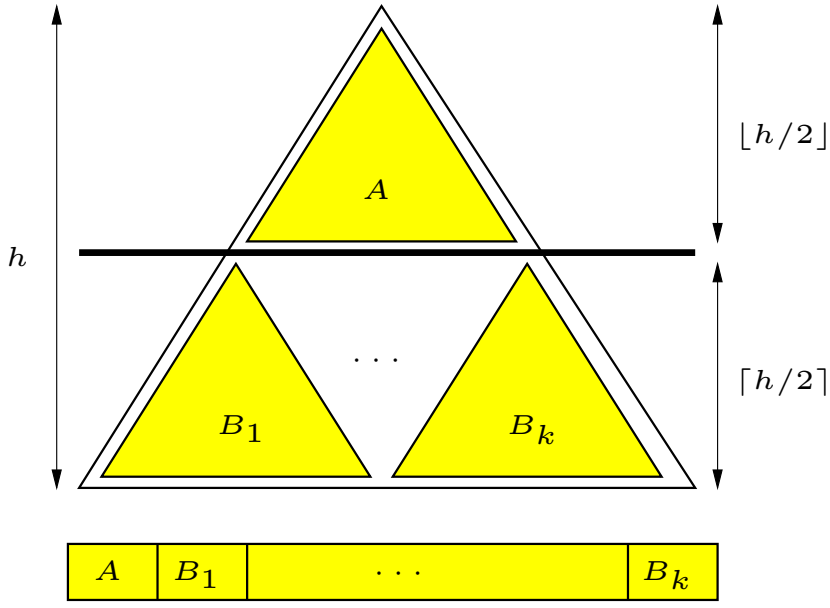
Binary tree

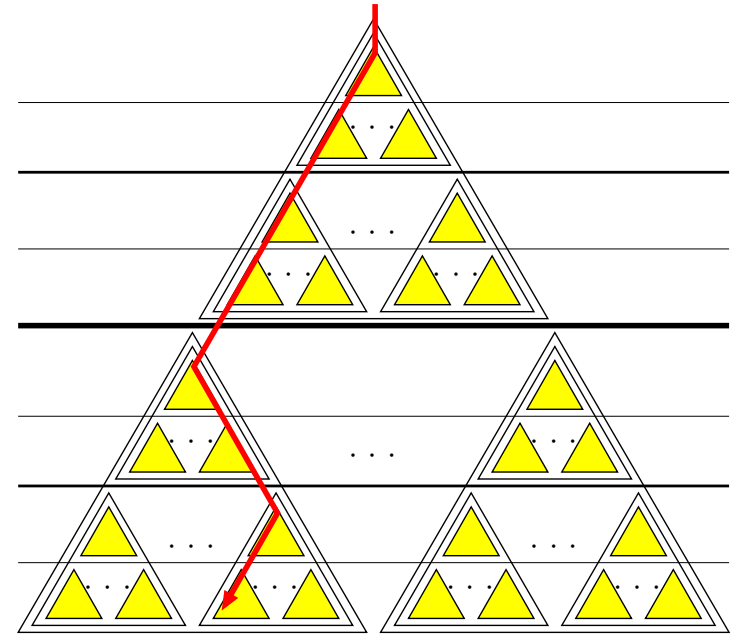
Searches use $O(\log_B N)$ I/Os

Static Cache-Oblivious Trees

Recursive memory layout (van Emde Boas layout)

Prokop 1999





Binary tree

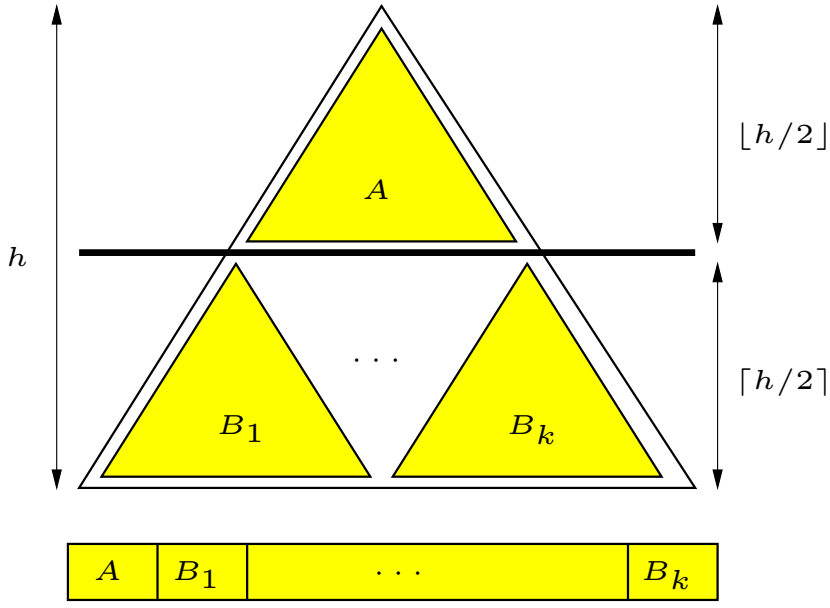
Searches use $O(\log_B N)$ I/Os

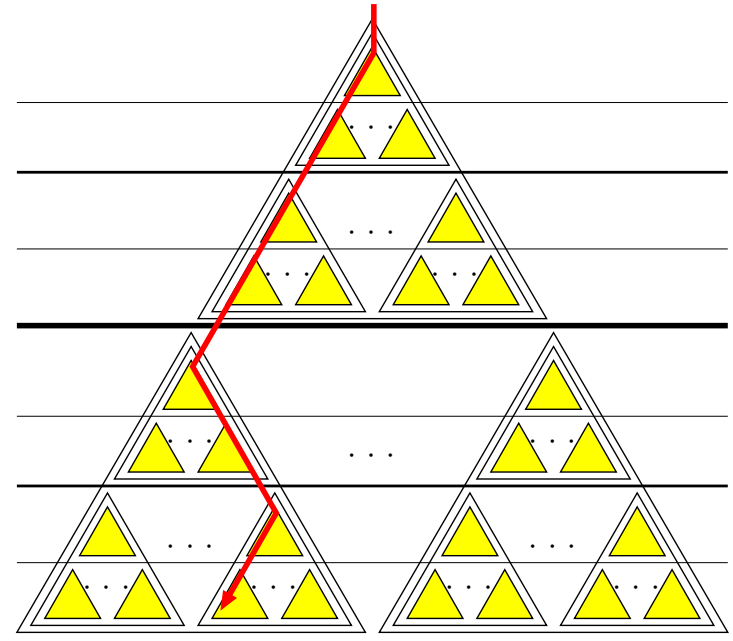
Dynamization?

Static Cache-Oblivious Trees

Recursive memory layout (van Emde Boas layout)

Prokop 1999





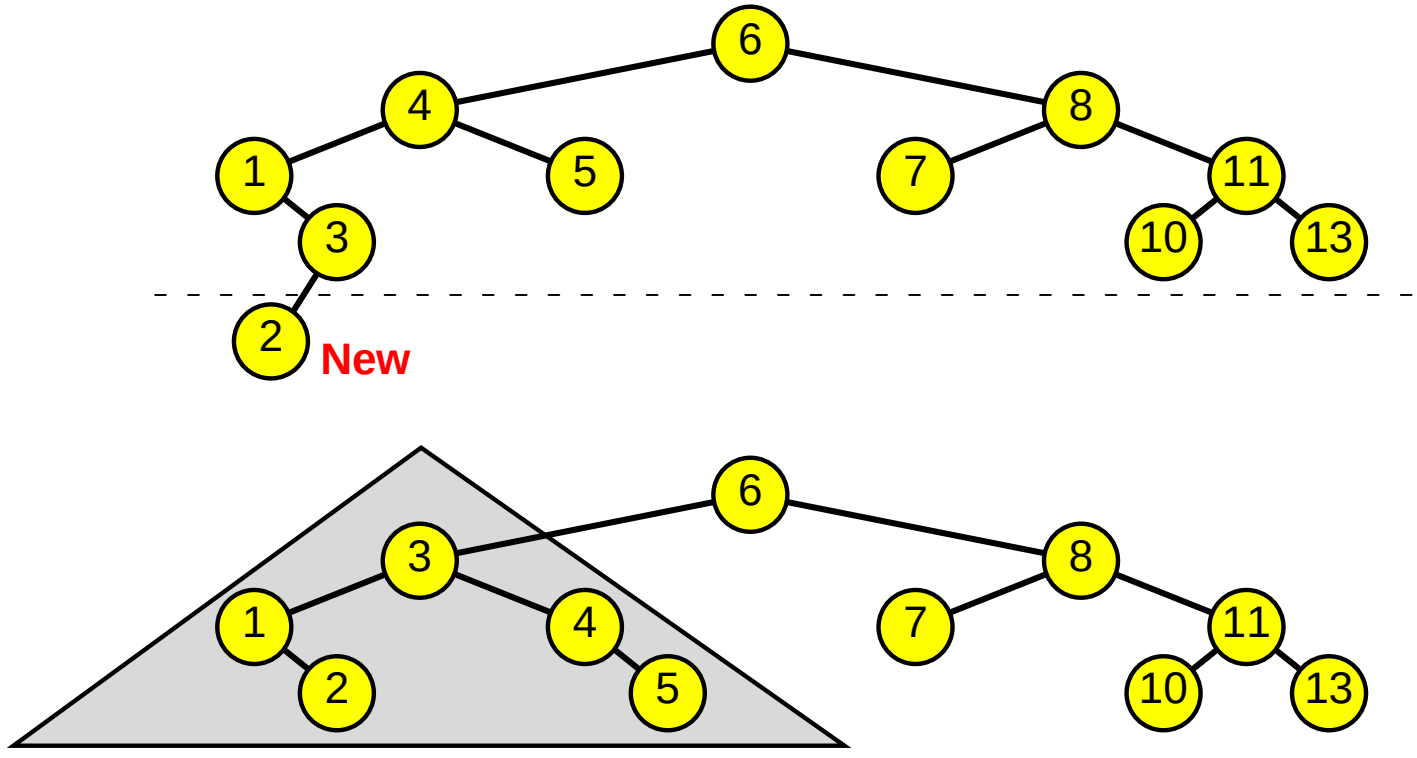
Binary tree

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

Dynamization?

Bender, Demaine, Farach-Colton, FOCS'00
Rahman, Cole, Raman, WAE'01
Bender, Duan, Iacono, Wu, SODA 02
Brodal, Fagerberg, Jacob, SODA'02

Binary Trees of Height $\log_2(n) + \mathrm{O}(1)$

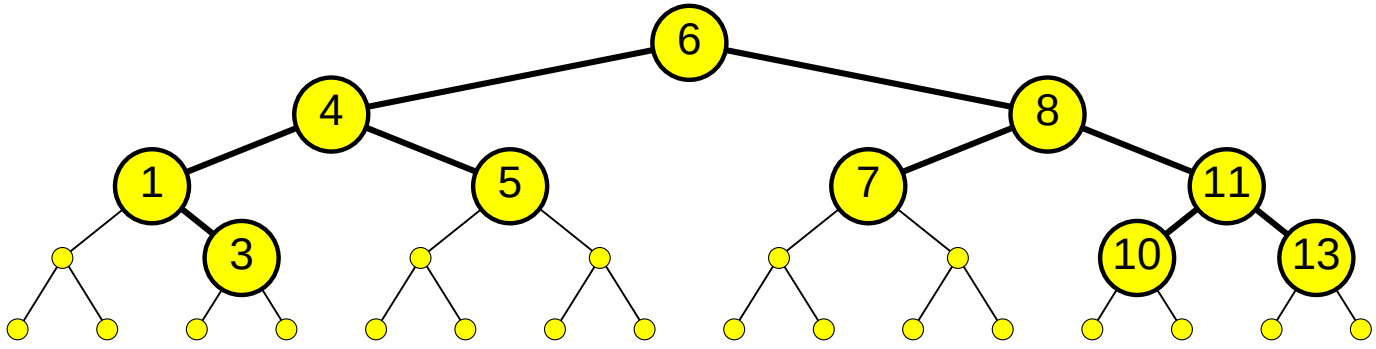


- If an insertion causes non-small height then rebuild subtree at nearest ancestor with sufficient few descendents
- Insertions require amortized time $\mathrm{O}(\log^2 N)$

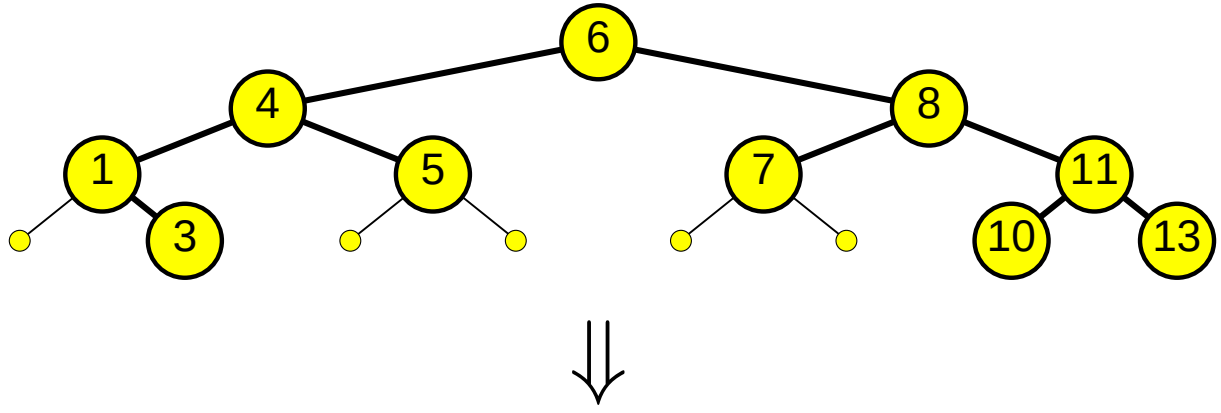
Itai, Konheim, Rodeh, 1981
Andersson, Lai, 1990

Simple Dynamic Cache-Oblivious Trees

- Embed a dynamic tree of height $\log_2(n) + \mathrm{O}(1)$ into a complete tree
- Static van Emde Boas layout of the complete tree in array.



Example



6	4	8	1	–	3	5	–	–	7	–	–	11	10	13

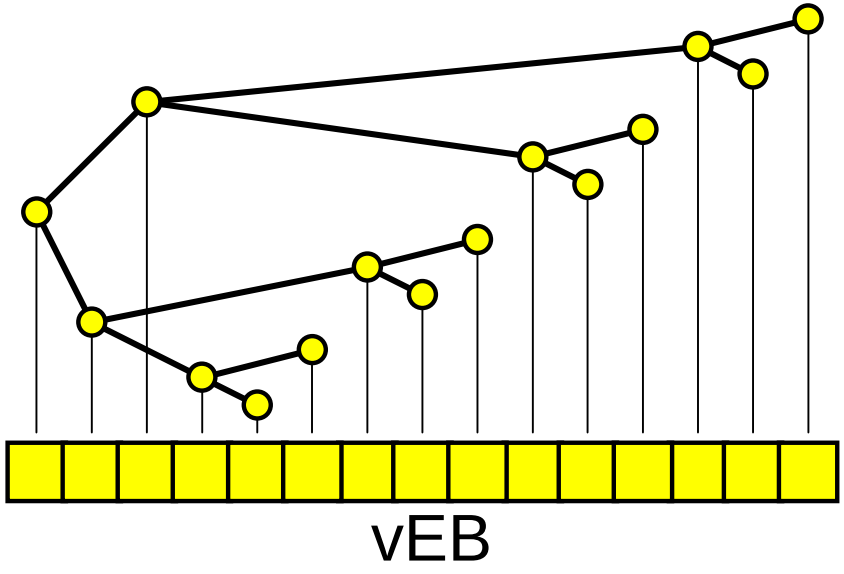
Search	$\mathrm{O}(\log_B N)$
Range Reporting	$\mathrm{O}\left(\log_B N + \frac{k}{B}\right)$
Updates	$\mathrm{O}\left(\log_B N + \frac{\log^2 N}{B}\right)$

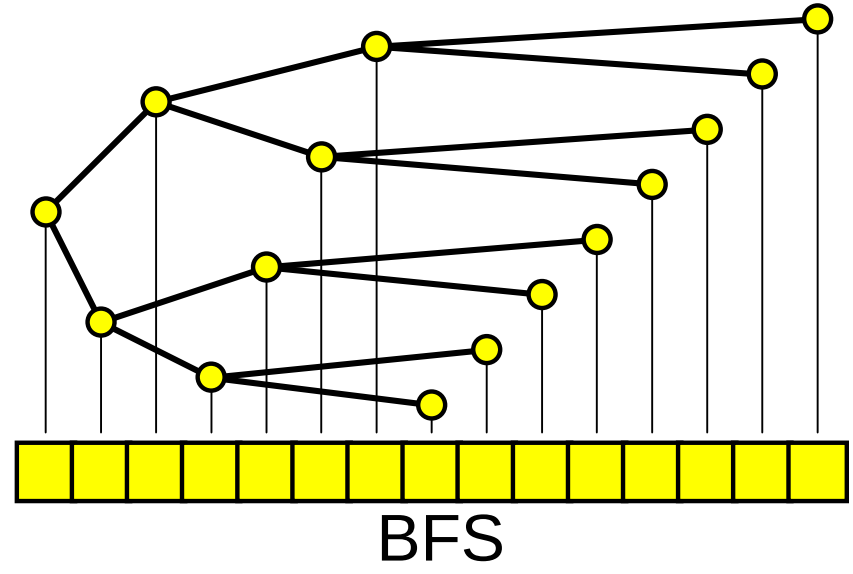
Experiments

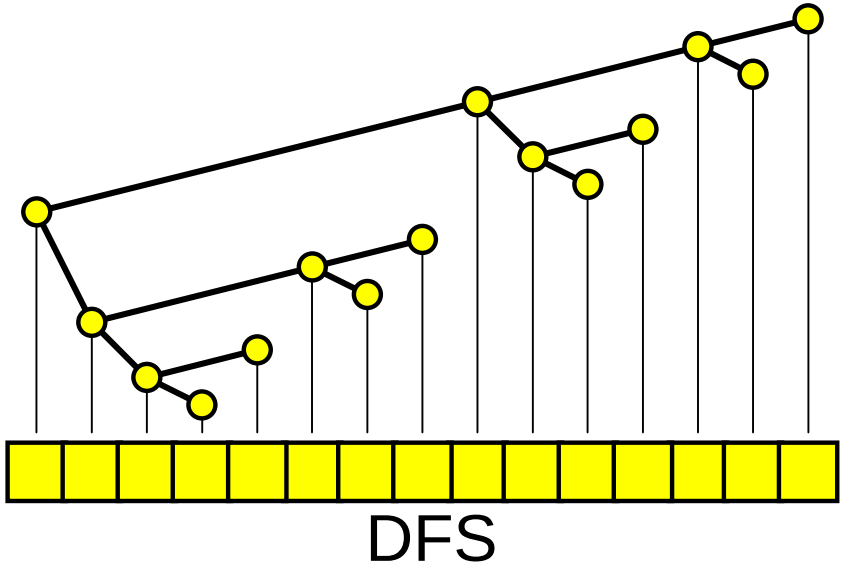
Brodal, Fagerberg, Jacob, SODA'02

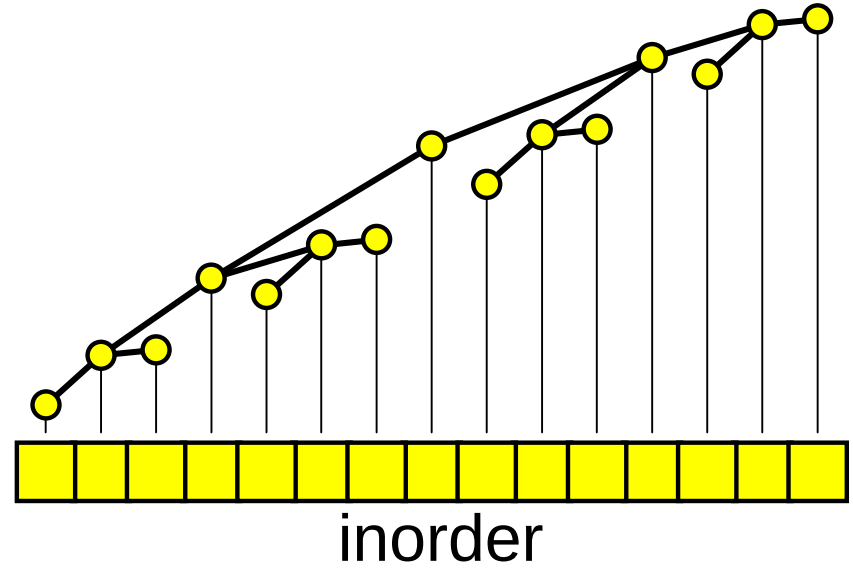
1. Study search time in static tree layouts.
   - Classic layouts: BFS, DFS, inorder, randomly built trees
   - Cache-aware multi-way trees
   - Cache-oblivious vEB layout
2. Study pointer-based vs. implicit representation.

Different Memory Layouts

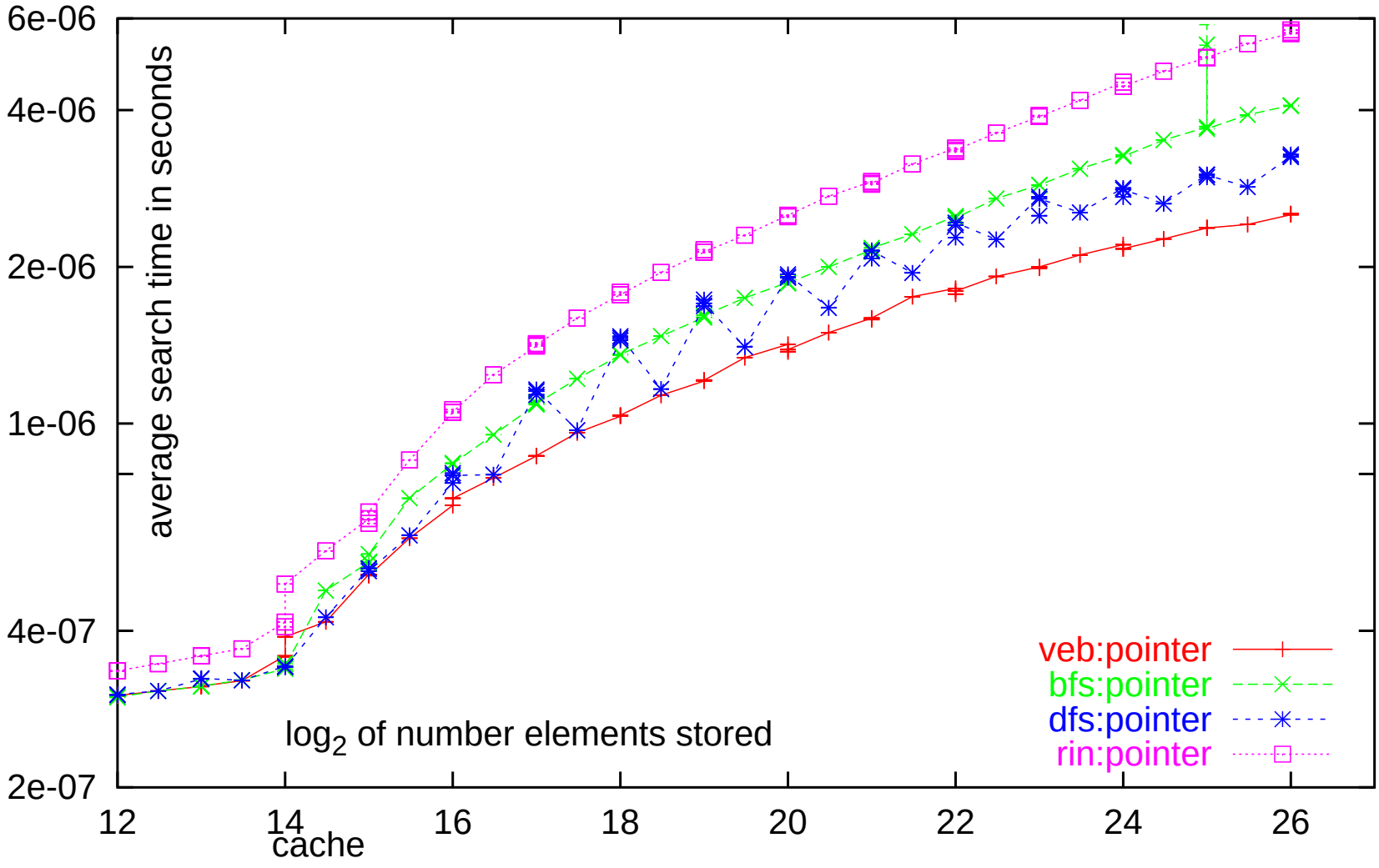

vEB


BFS


DFS


inorder

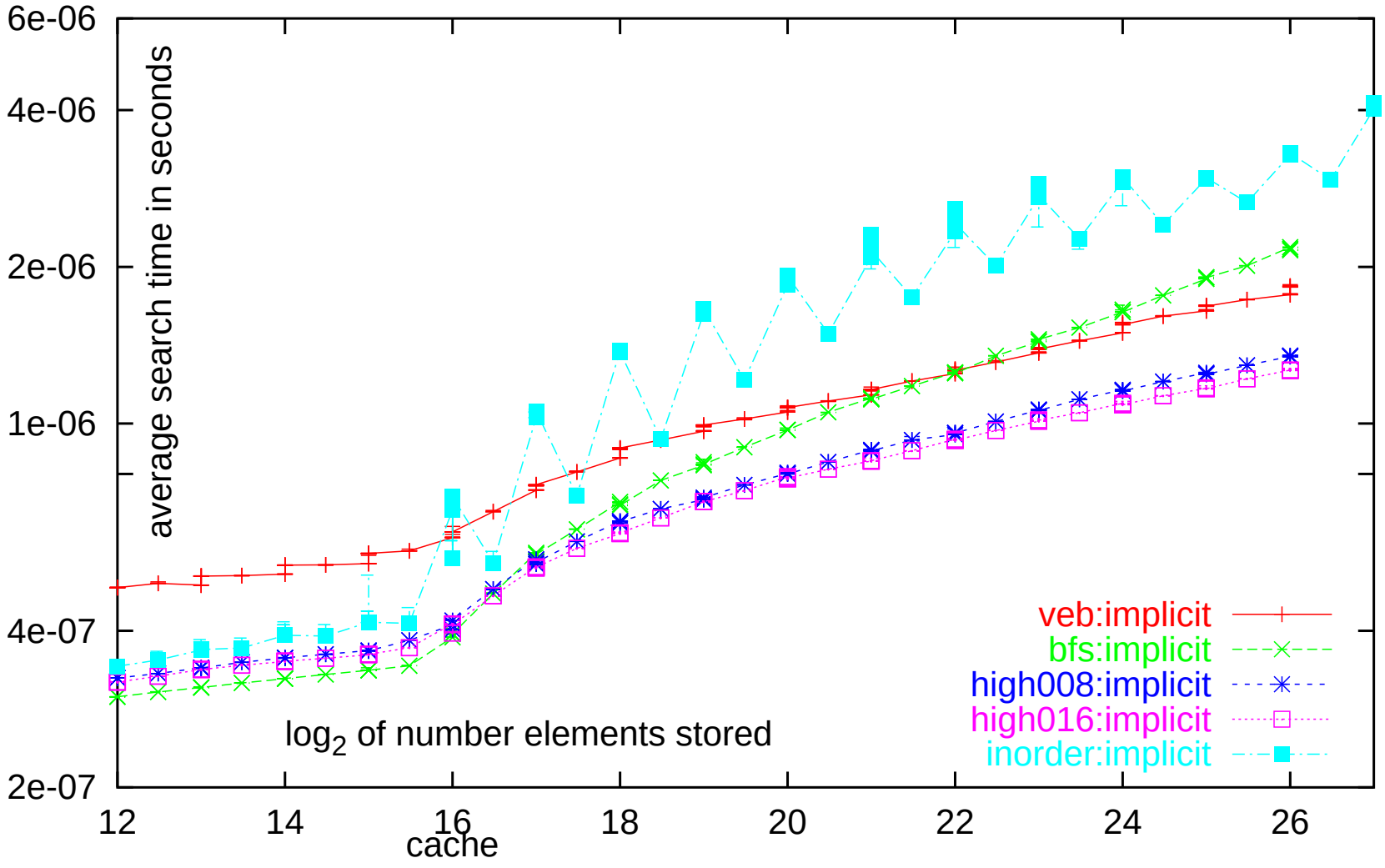
Search in Pointer Based Tree



Pointer based vEB, BFS, DFS, and randomly built.

1 GHz Pentium III, 1GB RAM, 256 KB Cache, gcc -O3 / linux

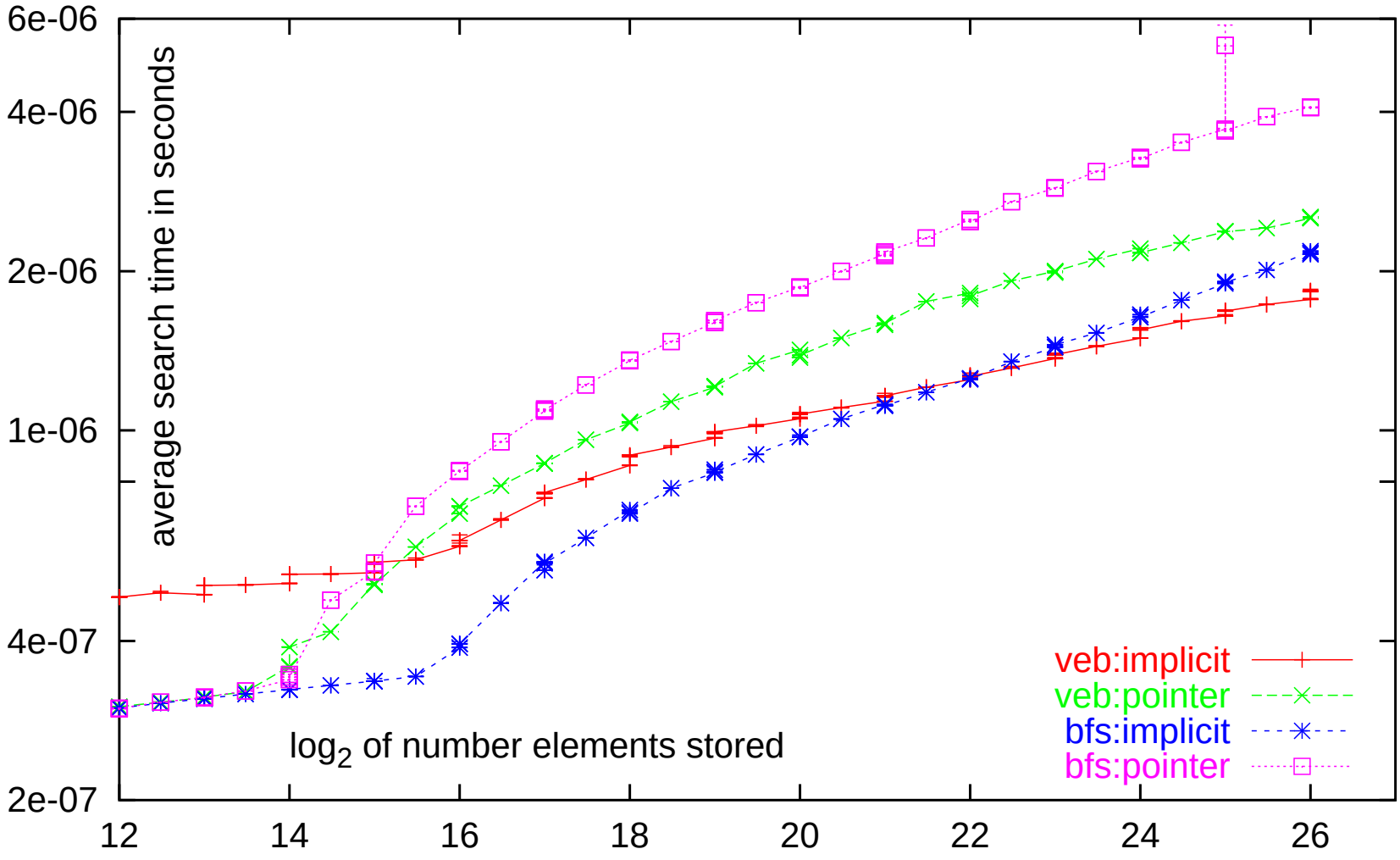
Search in Implicit Tree



Implicit vEB, BFS, inorder, multi-way.

1 GHz Pentium III, 1GB RAM, 256 KB Cache, gcc -O3 / linux

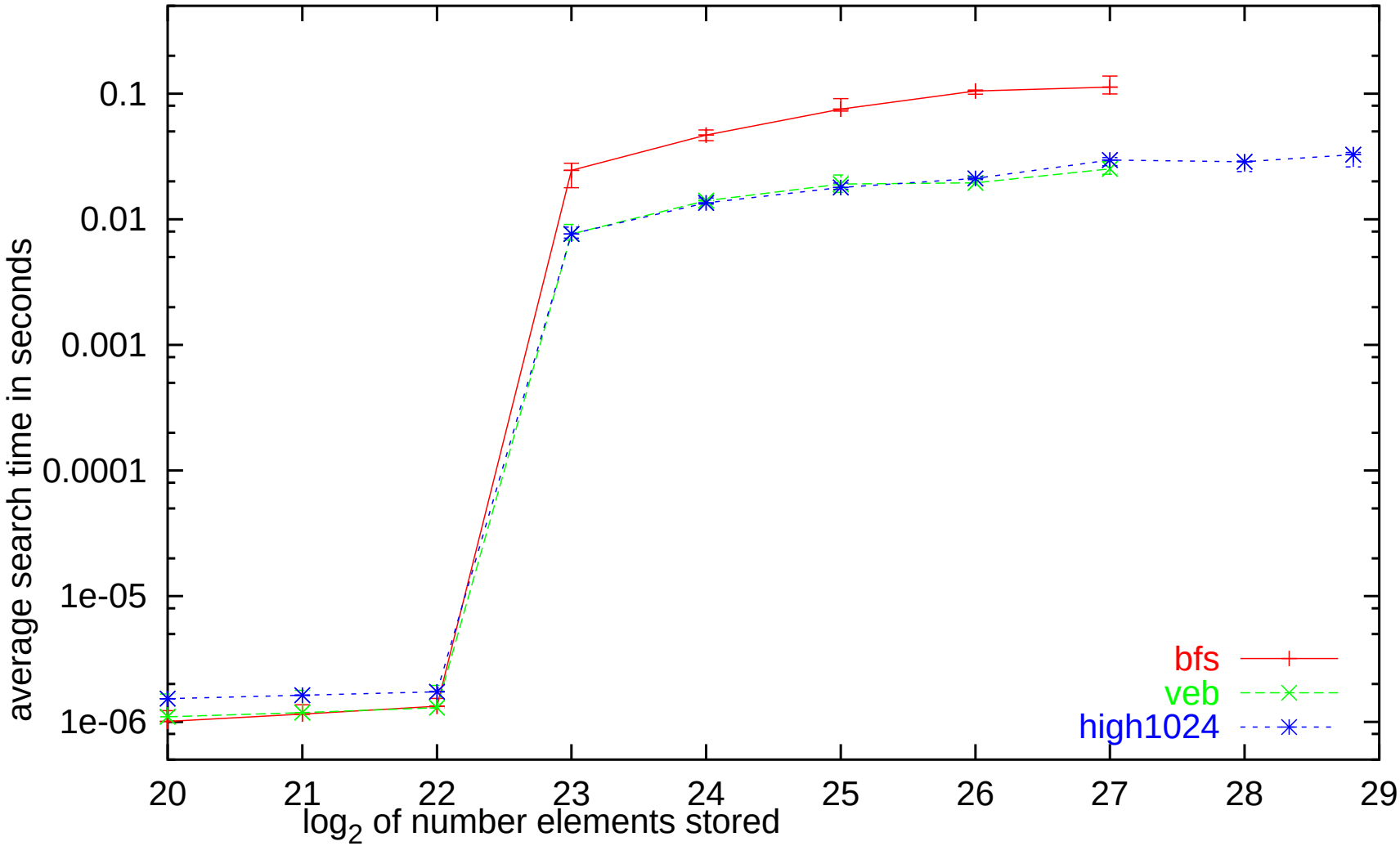
Pointer versus Implicit



static van Emde Boas layout and bfs-layout

1 GHz Pentium III, 1GB RAM, 256 KB Cache, gcc -O3 / linux

Beyond Main Memory



Multiway-tree, BFS, vEB implicit layout

1 GHz Pentium III, 32MB RAM, 256 KB Cache, gcc -O3 / linux

Overview

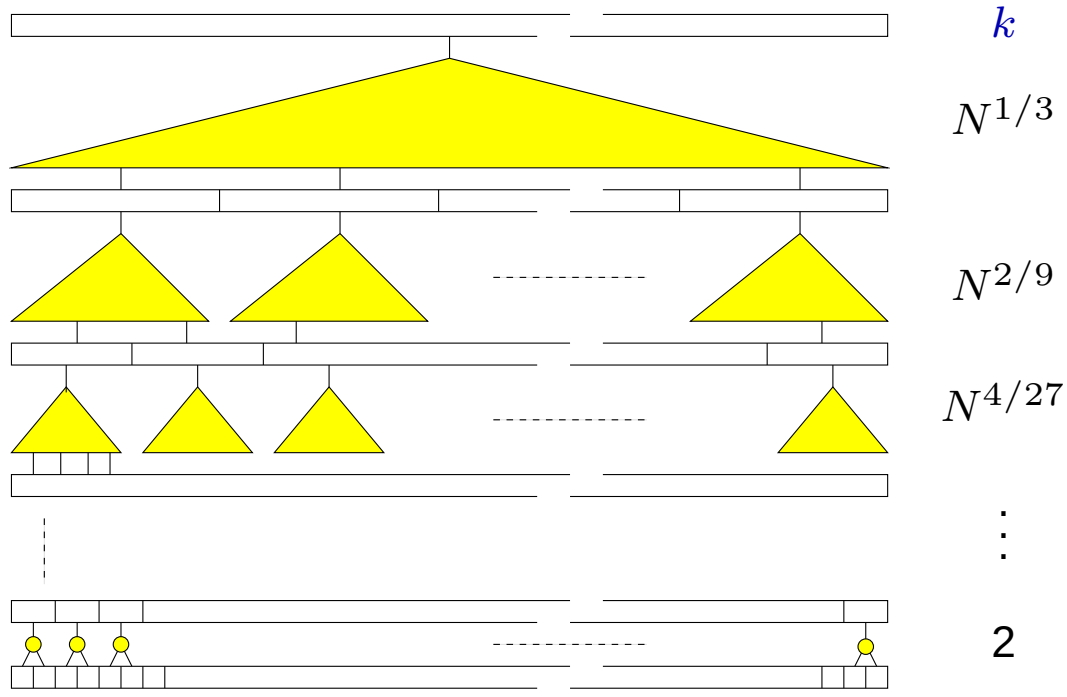
- √ The memory hierachy
- √ The I/O-model
- √ The cache-oblivious model
- Examples of cache-oblivious algorithms
  - √ Double for-loop (with applications)
  - √ Searching
  - Sorting
- Theoretical limits of cache-obliviousness

Funnelsort

Divide input in $N^{1/3}$ segments of size $N^{2/3}$

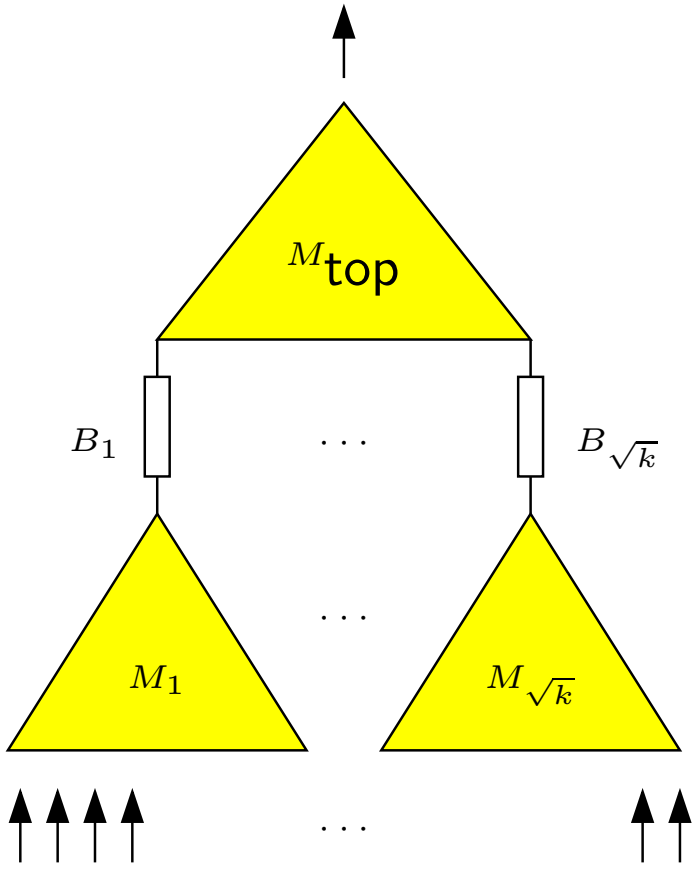
Recursively **Funnelsort** each segment

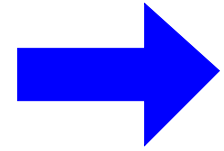
Merge sorted segments by an $N^{1/3}$**-merger**

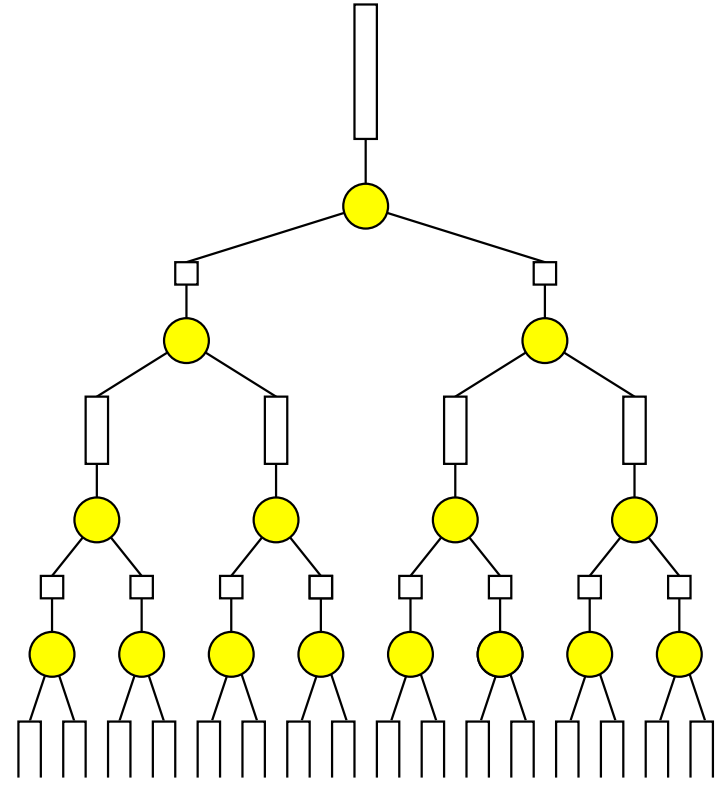


Frigo, Leiserson, Prokop, Ramachandran, 1999

k-merger



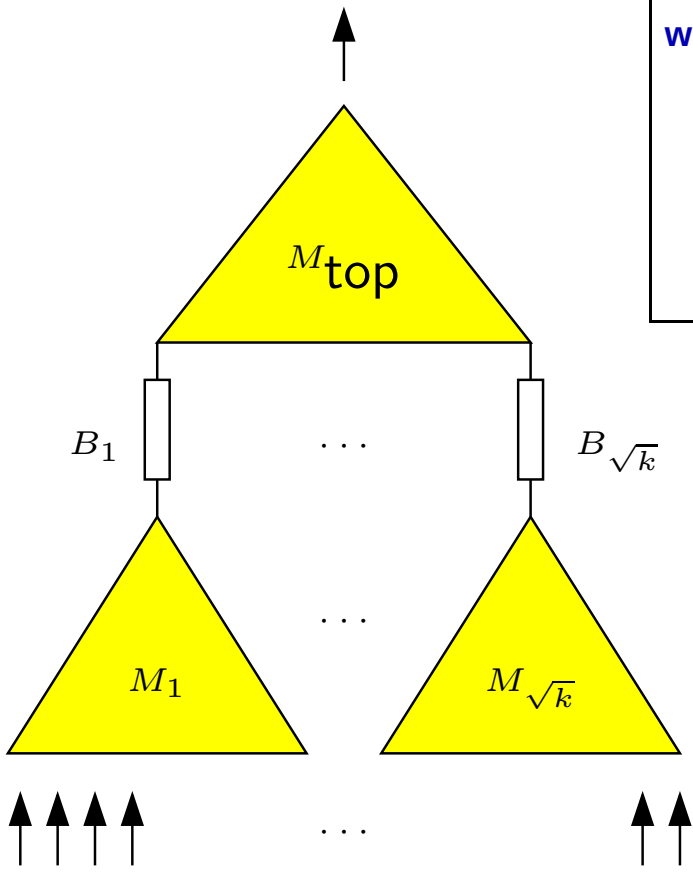




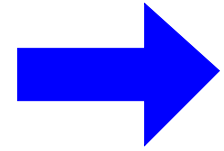
Buffer size $\alpha \cdot \sqrt{k}^{\,d}$

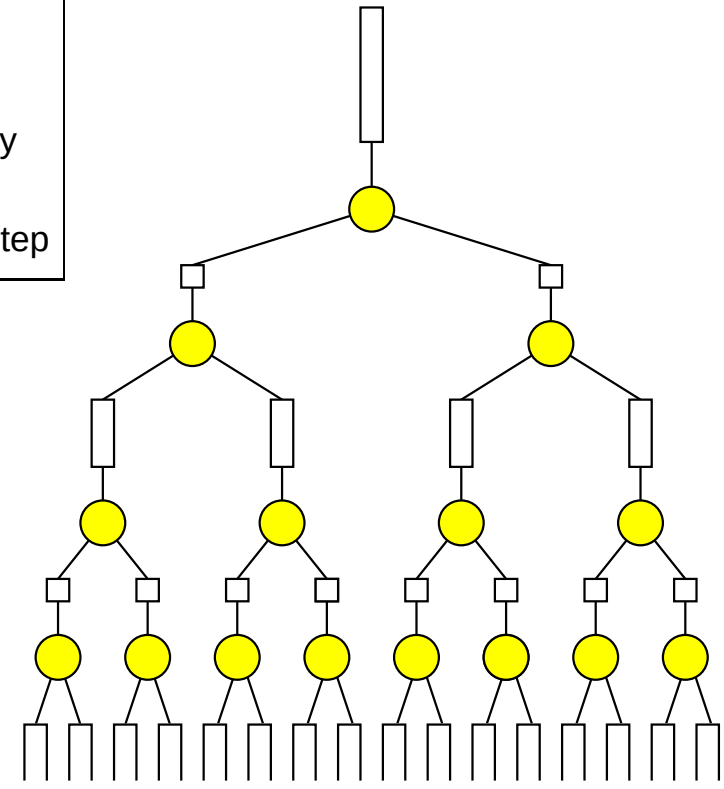
Brodal, Fagerberg 2002

k-merger



Fill(v):
while out-buffer not full
  if left in-buffer empty
    Fill(left child)
  if right in-buffer empty
    Fill(right child)
  perform one merge step





Buffer size $\alpha \cdot \sqrt{k}^{\,d}$

Brodal, Fagerberg 2002

Experiments

Engineer: Try a large number of possibilities for parameter choices, design choices, code optimizations (human vs. compiler+CPU), memory layouts,. . .

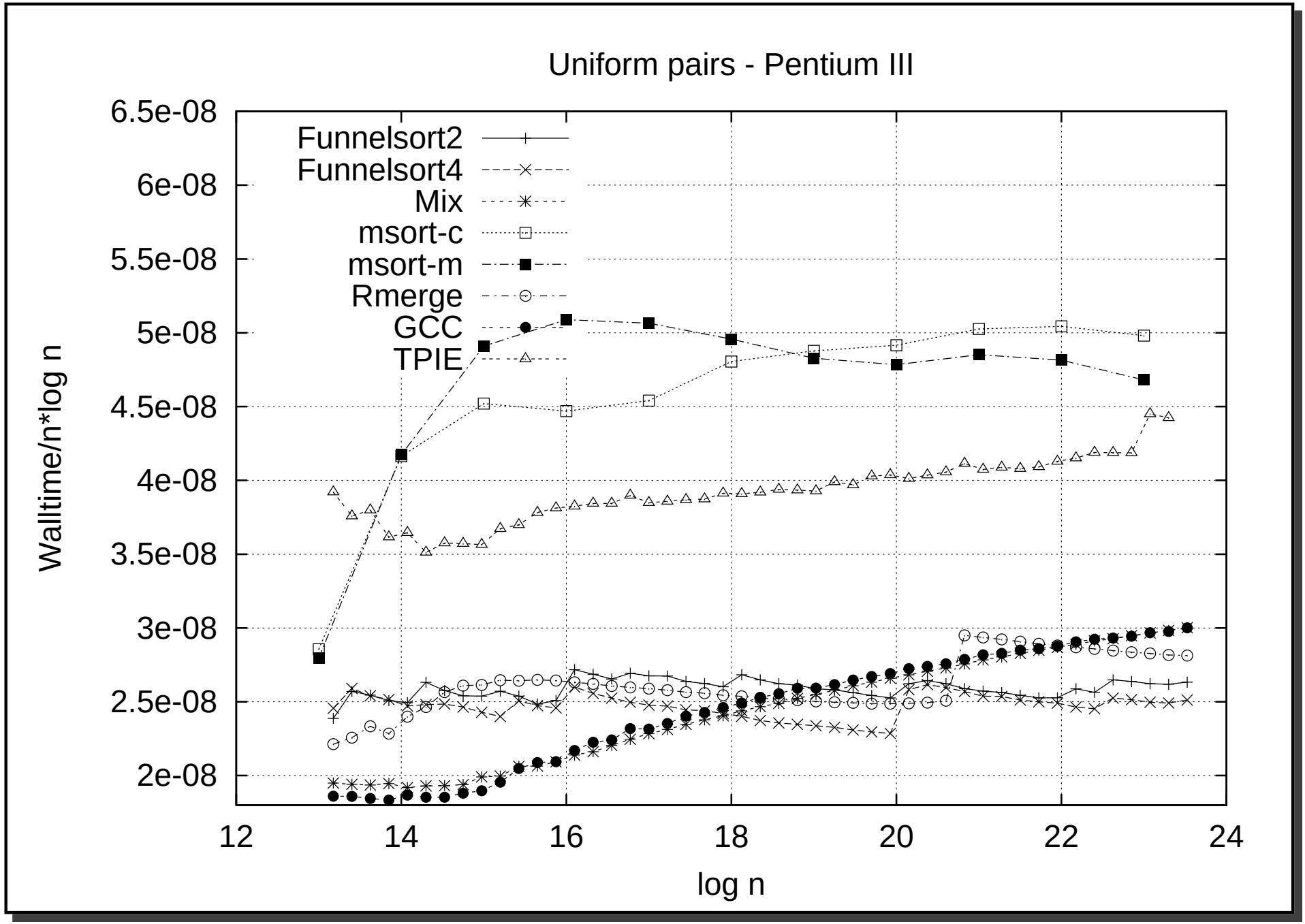
Choose a hard pack of competitors:

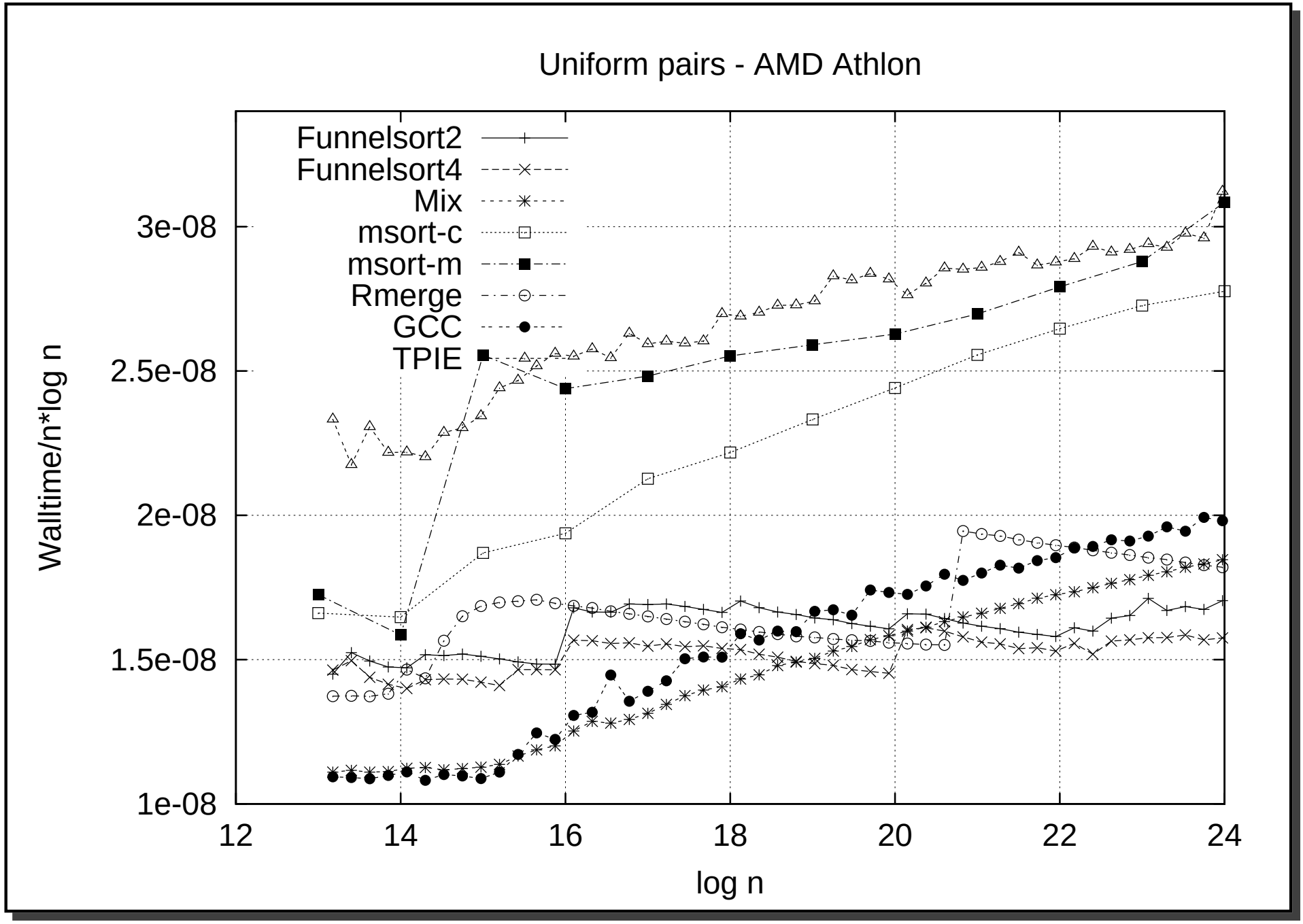
- 2-way and 4-way Funnelsort
- Best library Quicksort we can find
- Recent cache-aware proposals (tuned for RAM or for disk)

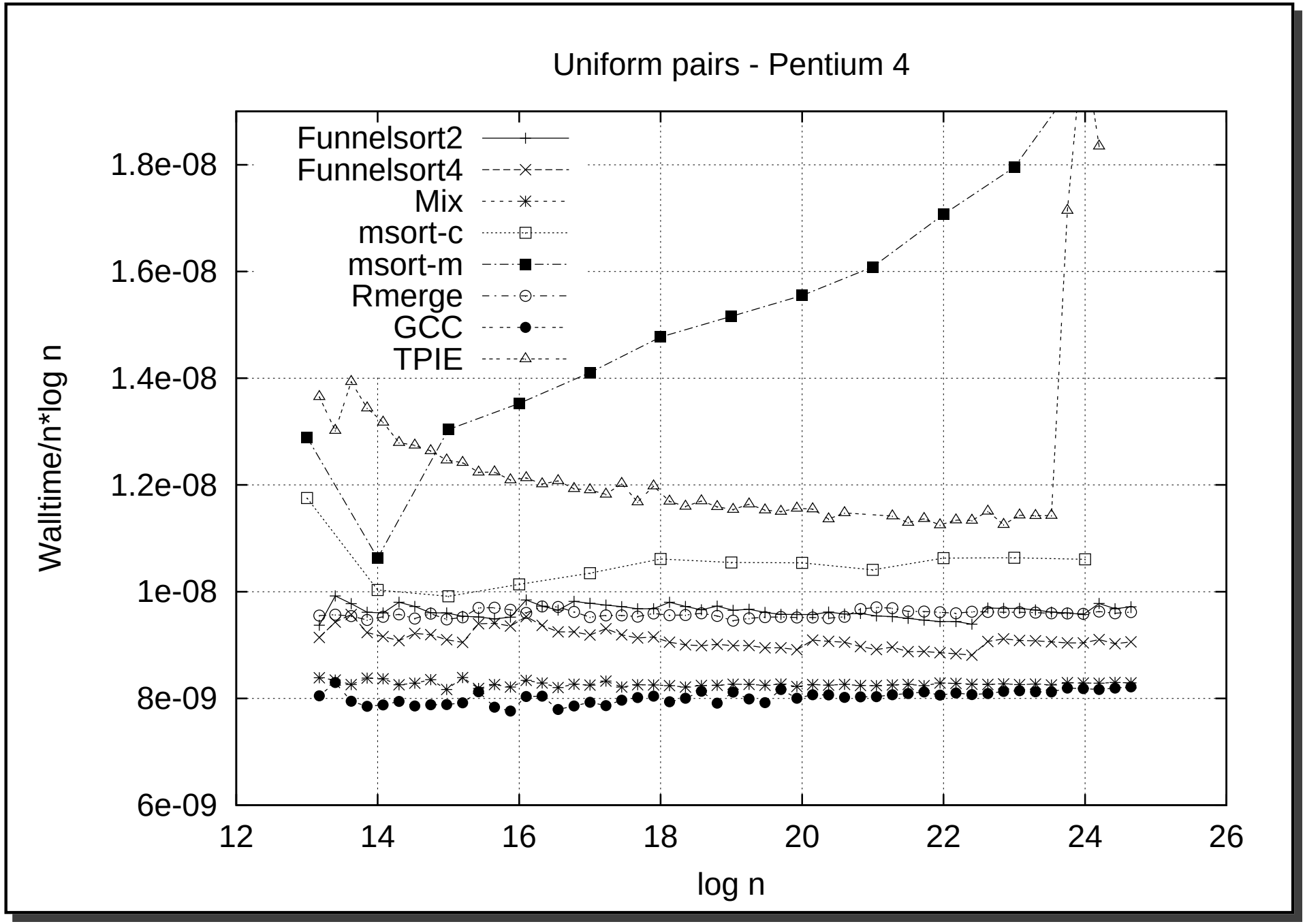
Run on a number of different machines: Pentium 4, Pentium III, MIPS 10000, AMD Athlon, Itanium 2.

Brodal, Fagerberg, Vinther, ALENEX'04

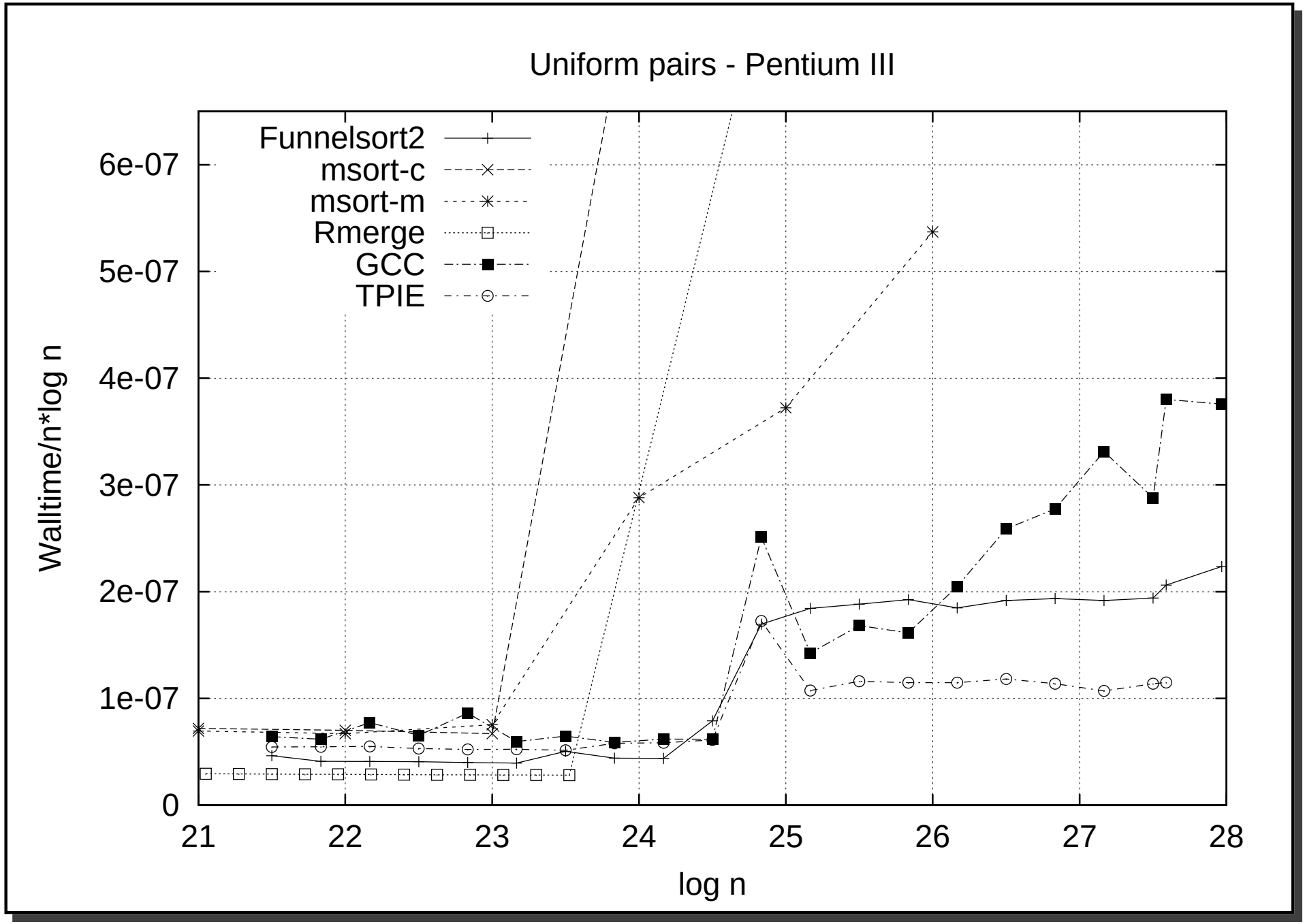
Results for Inputs in RAM

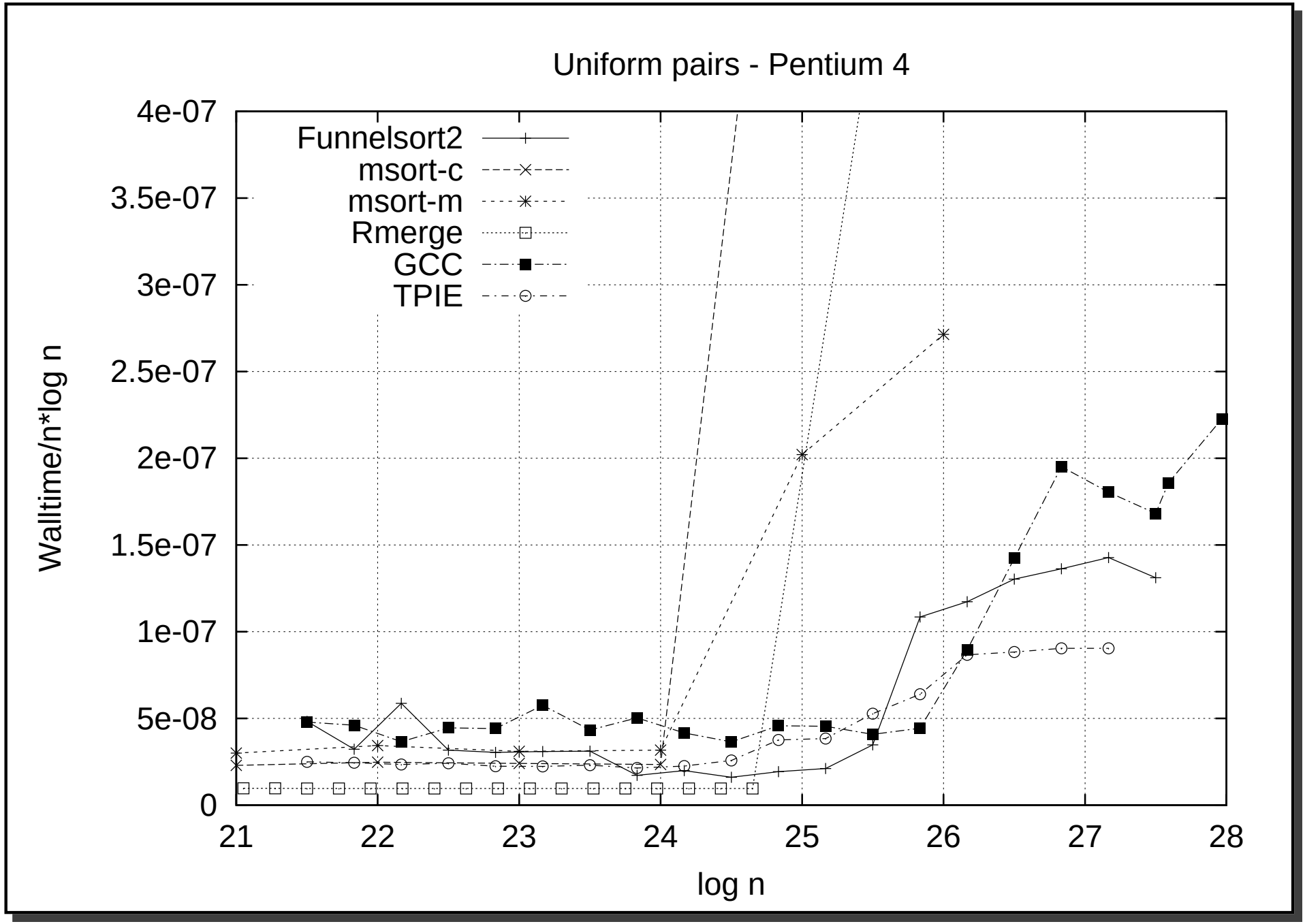

Uniform pairs - Pentium III
Funnelsort2
Funnelsort4
Mix
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
log n
6.5e-08
6e-08
5.5e-08
5e-08
4.5e-08
4e-08
3.5e-08
3e-08
2.5e-08
2e-08
12
14
16
18
20
22
24


Uniform pairs - AMD Athlon
Funnelsort2
Funnelsort4
Mix
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
3e-08
2.5e-08
2e-08
1.5e-08
1e-08
12
14
16
18
20
22
24
log n


Uniform pairs - Pentium 4
Funnelsort2
Funnelsort4
Mix
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
1.8e-08
1.6e-08
1.4e-08
1.2e-08
1e-08
8e-09
6e-09
12
14
16
18
20
22
24
26
log n

Results for Inputs on Disk


Uniform pairs - Pentium III
Funnelsort2
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
0
1e-07
2e-07
3e-07
4e-07
5e-07
6e-07
21
22
23
24
25
26
27
28
log n


Uniform pairs - Pentium 4
Funnelsort2
msort-c
msort-m
Rmerge
GCC
TPIE
Walltime/n*log n
4e-07
3.5e-07
3e-07
2.5e-07
2e-07
1.5e-07
1e-07
5e-08
0
21
22
23
24
25
26
27
28
log n

Practical Conclusion

For a number of basic algorithmic problems there exist solutions which

- Are theoretically I/0-efficient
- Are simple
- Are robust - adapt automatically to the specifics of the memory hierarchy
- Compete well with explicit (cache-aware) I/O algorithms.

Practical Conclusion

For a number of basic algorithmic problems there exist solutions which

- Are theoretically I/0-efficient
- Are simple
- Are robust - adapt automatically to the specifics of the memory hierarchy
- Compete well with explicit (cache-aware) I/O algorithms.

Consider putting cache-obliviousness in your toolbox of algorithmic techniques.

Overview

- √ The memory hierachy
- √ The I/O-model
- √ The cache-oblivious model
- Examples of cache-oblivious algorithms
  - √ Double for-loop (with applications)
  - √ Searching
  - √ Sorting
- Theoretical limits of cache-obliviousness

Basic Question

In terms of power:

Cache-oblivious algorithms = I/O-algorithms?

Basic Question

In terms of power:

Cache-oblivious algorithms = I/O-algorithms?

Or is there a separation: A problem for which no cache-oblivious algorithm can match the best I/O-algorithm?

Frigo, Leiserson, Prokop, Ramachandran, FOCS'99

Recent Answers

Brodal, Fagerberg, STOC'03

Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz, FOCS'03

Sorting: $\boxed{\mathrm{Sort}_{B,M}(N) \text{ I/Os}}$?

Permuting: $\boxed{\mathrm{Perm}_{B,M}(N) \text{ I/Os}}$?

Searching: $\boxed{c \cdot \log_B \frac{N}{M} \text{ I/Os}}$?

Recent Answers

Brodal, Fagerberg, STOC'03

Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz, FOCS'03

Sorting: $\boxed{\mathrm{Sort}_{B,M}(N) \text{ I/Os}}$?

Not possible without tall cache assumption

Permuting: $\boxed{\mathrm{Perm}_{B,M}(N) \text{ I/Os}}$?

Not possible even with tall cache assumption

Searching: $\boxed{c \cdot \log_B \frac{N}{M} \text{ I/Os}}$?

$c \geq \log(e) \approx 1.443$

Recent Answers

Brodal, Fagerberg, STOC'03

Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz, FOCS'03

Sorting: $\boxed{\mathrm{Sort}_{B,M}(N) \text{ I/Os}}$?

Not possible without tall cache assumption

Permuting: $\boxed{\mathrm{Perm}_{B,M}(N) \text{ I/Os}}$?

Not possible even with tall cache assumption

Searching: $\boxed{c \cdot \log_B \frac{N}{M} \text{ I/Os}}$?

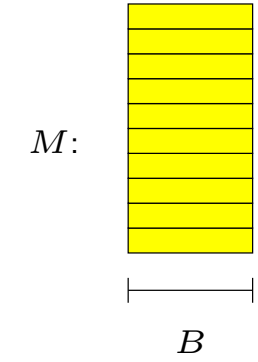
$c \geq \log(e) \approx 1.443$

Matching upper bounds for sorting and searching

Cache-Oblivious Sorting

Previous cache-oblivious sorting results use a

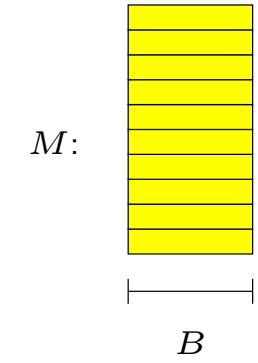
Tall Cache Assumption $B \leq M^{1/2}$



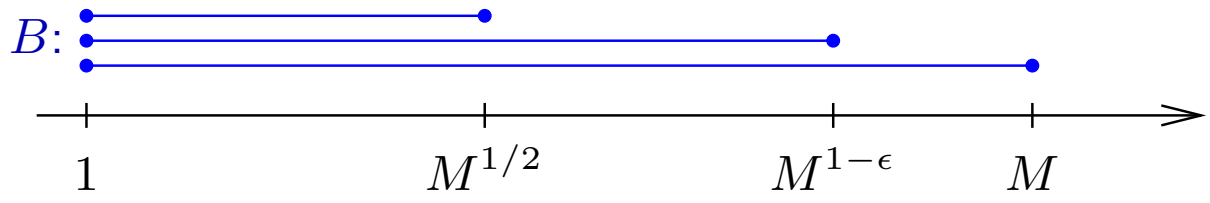
Cache-Oblivious Sorting

Previous cache-oblivious sorting results use a

Tall Cache Assumption $B \leq M^{1/2}$

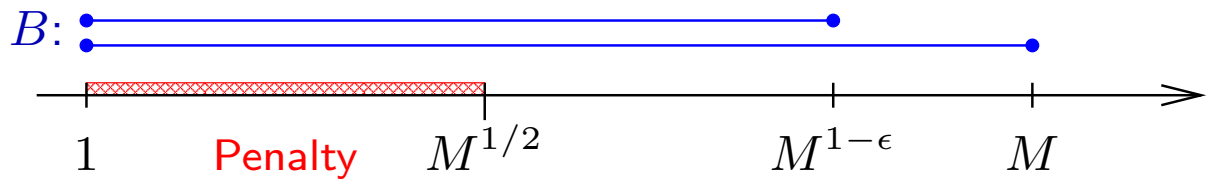


Algorithm	*Assumption*	*I/O bound*
Funnelsort [FLPR99]	$B \leq M^{1/2}$	$\mathrm{Sort}_{B,M}(N)$
Lazy Funnelsort [BF02]	$B \leq M^{1-\epsilon}$	$\frac{N}{\epsilon B} \log_M \frac{N}{M}$
Binary Mergesort	$(B \leq M)$	$\frac{N}{B} \log_2 \frac{N}{M}$



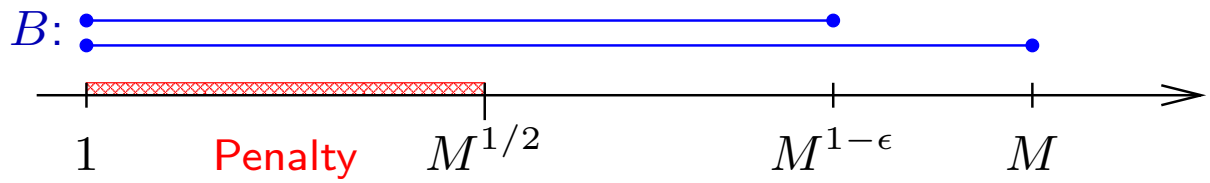
Result for Sorting

	Assumption	*I/Os*
Lazy Funnelsort	$B \leq M^{1-\epsilon}$	$B = M^{1-\epsilon}$: $\mathrm{Sort}_{B,M}(N)$ $B \leq M^{1/2}$: $\mathrm{Sort}_{B,M}(N) \cdot \boxed{\frac{1}{\epsilon}}$
Binary Mergesort	$(B \leq M)$	$B = M$: $\mathrm{Sort}_{B,M}(N)$ $B \leq M^{1/2}$: $\mathrm{Sort}_{B,M}(N) \cdot \boxed{\log M}$



Result for Sorting

	Assumption	*I/Os*
Lazy Funnelsort	$B \leq M^{1-\epsilon}$	$(a)\ B = M^{1-\epsilon}: \quad \mathrm{Sort}_{B,M}(N)$ $(b)\ B \leq M^{1/2}: \quad \mathrm{Sort}_{B,M}(N) \cdot \boxed{\frac{1}{\epsilon}}$
Binary Mergesort	$(B \leq M)$	$(a)\ B = M \quad : \quad \mathrm{Sort}_{B,M}(N)$ $(b)\ B \leq M^{1/2}: \quad \mathrm{Sort}_{B,M}(N) \cdot \boxed{\log M}$



Theorem This is tight. For any cache-oblivious comparison based sorting algorithm:

$$(a) \Rightarrow (b)$$

The end

Proof

One algorithm, two machines ($B_1 \leq B_2$):

	Block Size	Memory	I/Os
Machine 1	B_1	M	t_1
Machine 2	B_2	M	t_2

Main result:

$$8t_1B_1 + 3t_1B_1 \log \frac{8Mt_2}{t_1B_1} \geq N \log \frac{N}{M} - 1.45N \qquad (*)$$

Theorem 1:

Inserting (a) in $(*)$ leads to (b)

Fake Proof

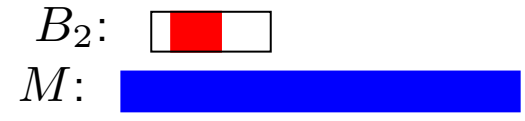
Goal:

$$8t_1B_1 + 3t_1B_1 \log \frac{8Mt_2}{t_1B_1} \geq N \log \frac{N}{M} - 1.45N \qquad (*)$$

Merging sorted lists X and Y takes $|X| \log \frac{|Y|}{|X|}$ comparisons.

In total t_1B_1 elements touched $\Rightarrow$ t_1B_1/t_2 elements touched on average per B_2-I/O $\Rightarrow$ effective B_2 is t_1B_1/t_2.

Comparisons gained per B_2-I/O:



$$t_1B_1/t_2 \cdot \log \frac{M}{t_1B_1/t_2} \ .$$

Hence:

$$t_1B_1 \cdot \log \frac{Mt_2}{t_1B_1} \geq N \log N - 1.45N \ .$$

□

Fake Proof

Goal:

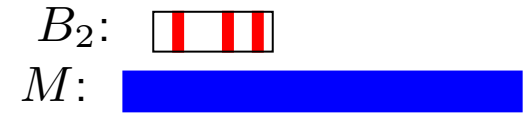
$$8t_1B_1 + 3t_1B_1 \log \frac{8Mt_2}{t_1B_1} \geq N \log \frac{N}{M} - 1.45N \qquad (*)$$

Merging sorted lists X and Y takes $|X| \log \frac{|Y|}{|X|}$ comparisons.

In total t_1B_1 elements touched $\Rightarrow$ t_1B_1/t_2 elements touched on average per B_2-I/O $\Rightarrow$ effective B_2 is t_1B_1/t_2.

Comparisons gained per B_2-I/O:

$$t_1B_1/t_2 \cdot \log \frac{M}{t_1B_1/t_2} \ .$$



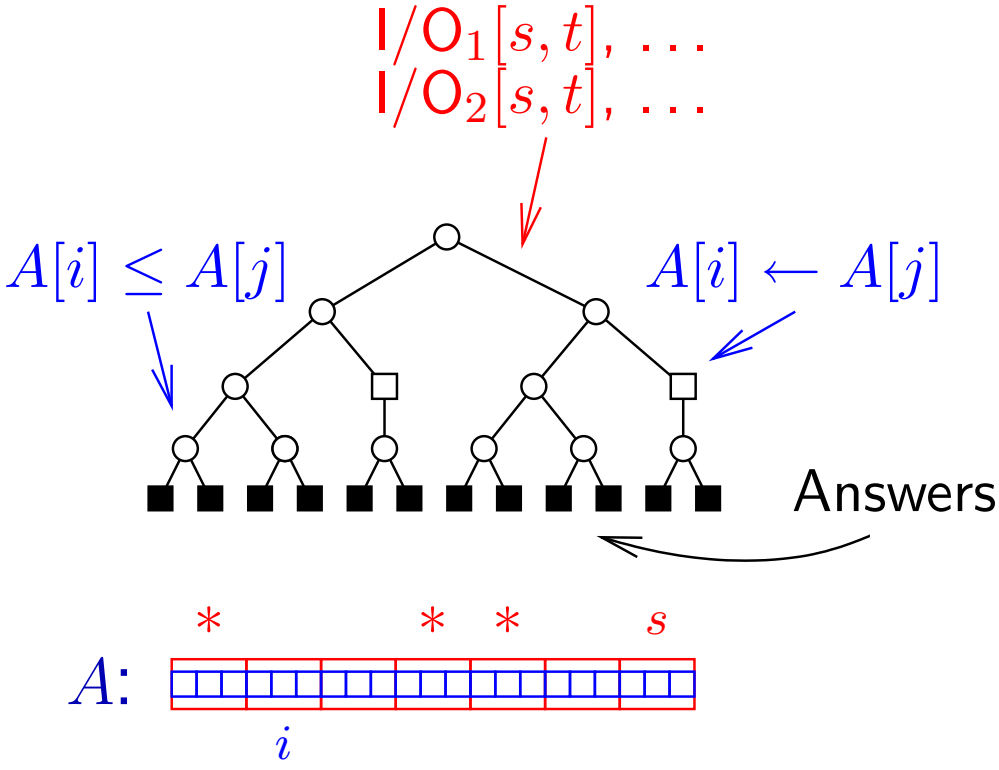
One problem:
Online choice

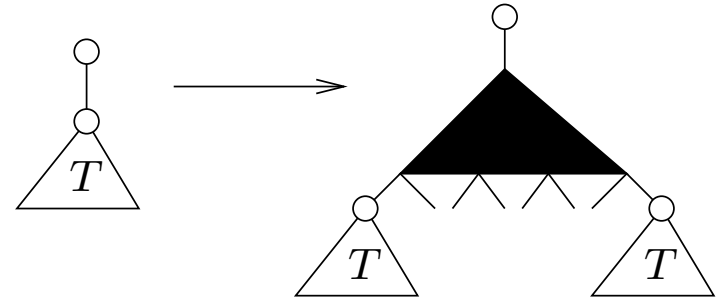
Hence:

$$t_1B_1 \cdot \log \frac{Mt_2}{t_1B_1} \geq N \log N - 1.45N \ .$$

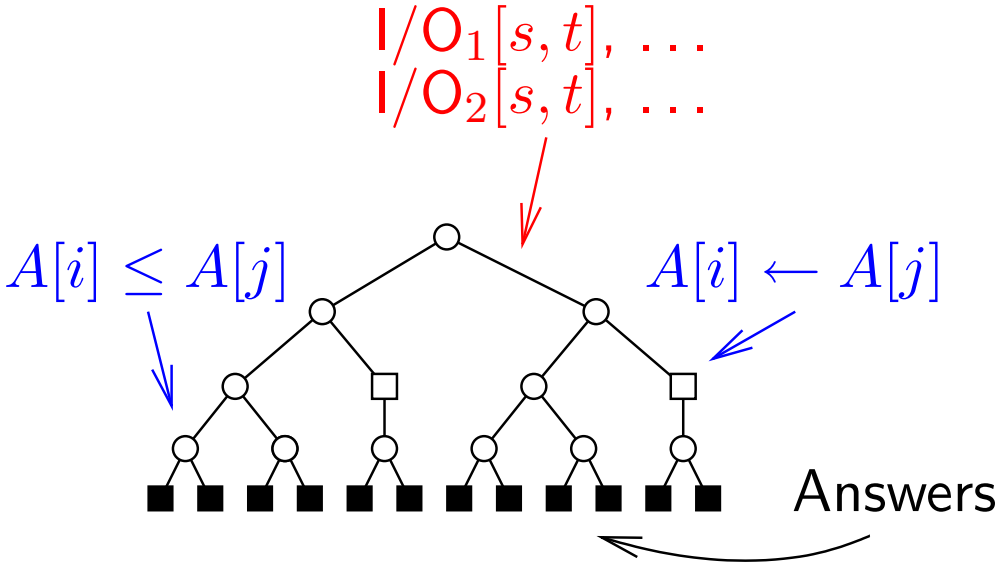
□

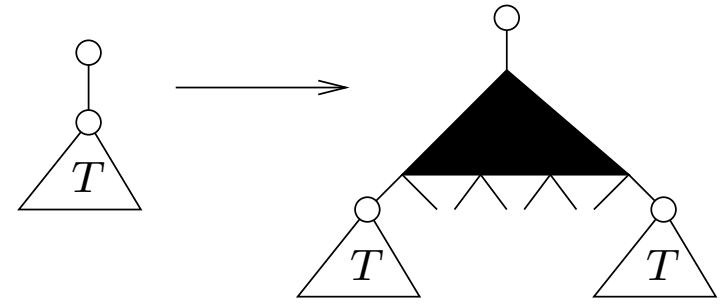
Ideas from Real Proof





Ideas from Real Proof





$$8t_1B_1 + 3t_1B_1 \log \frac{8Mt_2}{B_1t_1} \geq \text{height} \geq N \log \frac{N}{M} - 1.45N \ . \qquad (*)$$

The end