# Cache-Oblivious Algorithms

# Cache-Oblivious Model

# 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

# The Unknown Machine

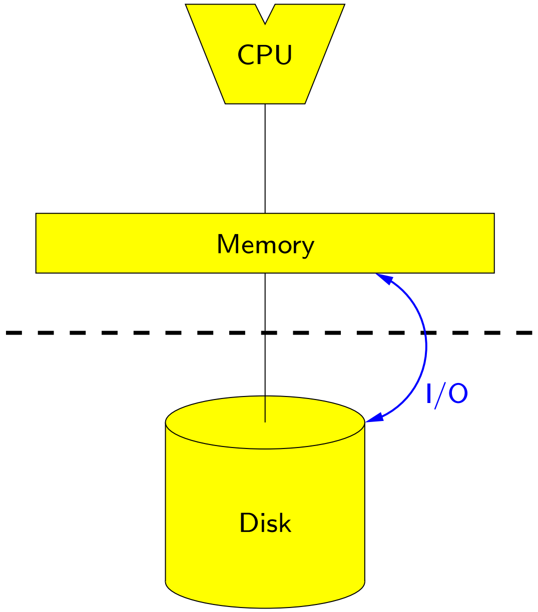
| Algorithm | Algorithm |
|---|---|
| ↓ | ↓ |
| C program | Java program |
| ↓ gcc | ↓ javac |
| Object code | Java bytecode |
| ↓ linux | ↓ java |
| Execution | Interpretation |

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

# 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 associative 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

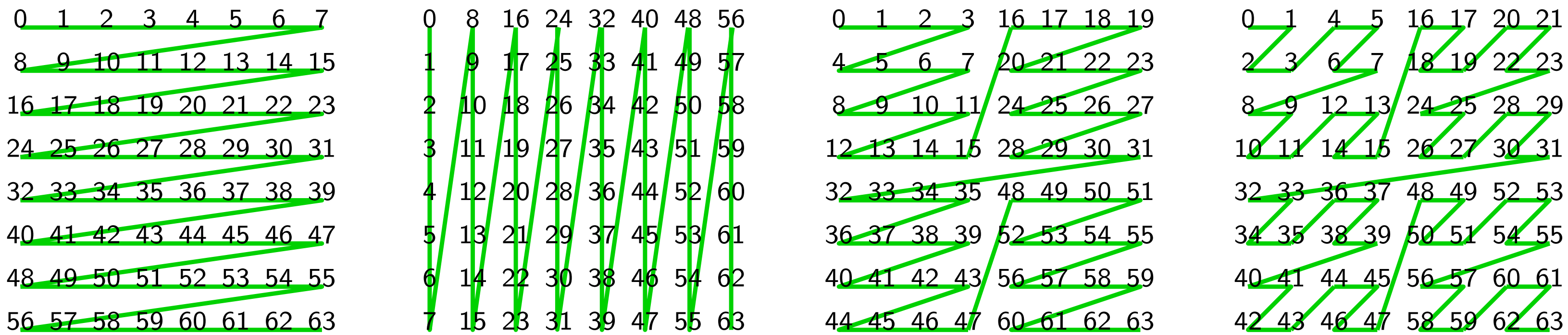
# Matrix Multiplication

# Matrix Multiplication

## Problem

$$C = A \cdot B \,, \quad c_{ij} = \sum_{k=1..N} a_{ik} \cdot b_{kj}$$

## Layout of matrices




Row major — Column major — $4 \times 4$-blocked — Bit interleaved

# Matrix Multiplication

## Algorithm 1: Nested loops

- Row major
- Reading a column of $B$ uses $N$ I/Os
- Total $O(N^3)$ I/Os

```
for i = 1 to N
    for j = 1 to N
        c_ij = 0
        for k = 1 to N
            c_ij = c_ij + a_ik · b_kj
```

# Matrix Multiplication

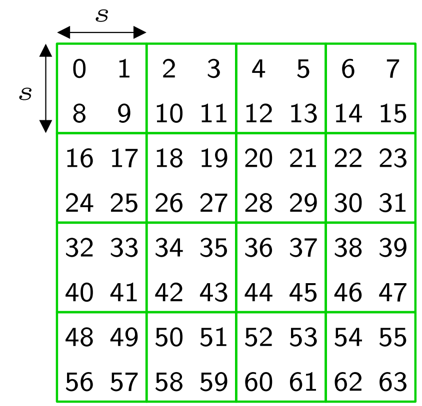
## Algorithm 1: Nested loops

- Row major
- Reading a column of $B$ uses $N$ I/Os
- Total $O(N^3)$ I/Os

for $i = 1$ to $N$
  for $j = 1$ to $N$
    $c_{ij} = 0$
    for $k = 1$ to $N$
      $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$

## Algorithm 2: Blocked algorithm (cache-aware)

- Partition $A$ and $B$ into blocks of size $s \times s$ where $s = \Theta(\sqrt{M})$
- Apply Algorithm 1 to the $\frac{N}{s} \times \frac{N}{s}$ matrices where elements are $s \times s$ matrices



# Matrix Multiplication

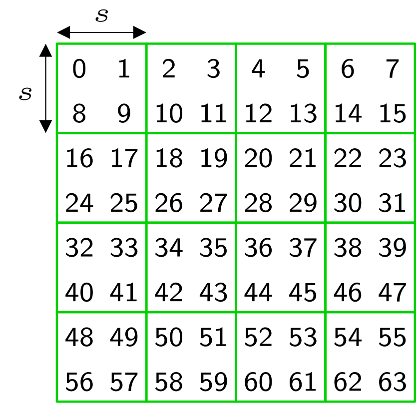
## Algorithm 1: Nested loops

- Row major
- Reading a column of $B$ uses $N$ I/Os
- Total $O(N^3)$ I/Os

for $i = 1$ to $N$<br>
    for $j = 1$ to $N$<br>
        $c_{ij} = 0$<br>
        for $k = 1$ to $N$<br>
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$

## Algorithm 2: Blocked algorithm (cache-aware)

- Partition $A$ and $B$ into blocks of size $s \times s$ where $s = \Theta(\sqrt{M})$
- Apply Algorithm 1 to the $\frac{N}{s} \times \frac{N}{s}$ matrices where elements are $s \times s$ matrices
- $s \times s$-blocked or $\left(\text{row major and } M = \Omega(B^2)\right)$




$$O\left(\left(\frac{N}{s}\right)^3 \cdot \frac{s^2}{B}\right) = O\left(\frac{N^3}{s \cdot B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right) \text{ I/Os}$$

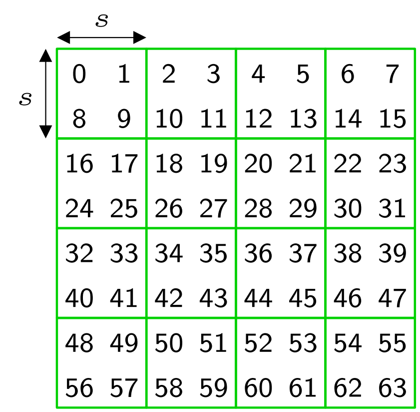
# Matrix Multiplication

## Algorithm 1: Nested loops

for $i = 1$ to $N$
  for $j = 1$ to $N$
    $c_{ij} = 0$
    for $k = 1$ to $N$
      $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$

- Row major
- Reading a column of $B$ uses $N$ I/Os
- Total $O(N^3)$ I/Os

## Algorithm 2: Blocked algorithm (cache-aware)




- Partition $A$ and $B$ into blocks of size $s \times s$ where $s = \Theta(\sqrt{M})$
- Apply Algorithm 1 to the $\frac{N}{s} \times \frac{N}{s}$ matrices where elements are $s \times s$ matrices
- $s \times s$-blocked or $\left(\text{row major and } M = \Omega(B^2)\right)$

$$O\left(\left(\frac{N}{s}\right)^3 \cdot \frac{s^2}{B}\right) = O\left(\frac{N^3}{s \cdot B}\right) = O\left(\frac{N^3}{B\sqrt{M}}\right) \text{ I/Os}$$

- Optimal

Hong & Kung, 1981

# Matrix Multiplication

## Algorithm 3: Recursive algorithm (cache-oblivious)

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

– 8 recursive $\frac{N}{2} \times \frac{N}{2}$ matrix multiplications + 4 $\frac{N}{2} \times \frac{N}{2}$ matrix sums

# Matrix Multiplication

## Algorithm 3: Recursive algorithm (cache-oblivious)

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

– 8 recursive $\frac{N}{2} \times \frac{N}{2}$ matrix multiplications + 4 $\frac{N}{2} \times \frac{N}{2}$ matrix sums

– # I/Os if bit interleaved or $\Big($row major and $M = \Omega(B^2)\Big)$

$$T(N) \leq \begin{cases} O(\frac{N^2}{B}) & \text{if } N \leq \varepsilon\sqrt{M} \\ 8 \cdot T\left(\frac{N}{2}\right) + O\left(\frac{N^2}{B}\right) & \text{otherwise} \end{cases}$$

$$T(N) \leq O\left(\frac{N^3}{B\sqrt{M}}\right)$$

# Matrix Multiplication

## Algorithm 3: Recursive algorithm (cache-oblivious)

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

– 8 recursive $\frac{N}{2} \times \frac{N}{2}$ matrix multiplications + 4 $\frac{N}{2} \times \frac{N}{2}$ matrix sums

– # I/Os if bit interleaved or $\Big($row major and $M = \Omega(B^2)\Big)$

$$T(N) \leq \begin{cases} O(\frac{N^2}{B}) & \text{if } N \leq \varepsilon\sqrt{M} \\ 8 \cdot T\left(\frac{N}{2}\right) + O\left(\frac{N^2}{B}\right) & \text{otherwise} \end{cases}$$

$$T(N) \leq O\left(\frac{N^3}{B\sqrt{M}}\right)$$

– Optimal — Hong & Kung, 1981

– Non-square matrices — Frigo et al., 1999

# Matrix Multiplication

## Algorithm 4: Strassen's algorithm (cache-oblivious)

– 7 recursive $\frac{N}{2} \times \frac{N}{2}$ matrix multiplications + $O(1)$ matrix sums

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned}
m_1 &:= (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11}) \\
m_2 &:= a_{11}b_{11} \\
m_3 &:= a_{12}b_{21} \\
m_4 &:= (a_{11} - a_{21})(b_{22} - b_{12}) \\
m_5 &:= (a_{21} + a_{22})(b_{12} - b_{11}) \\
m_6 &:= (a_{12} - a_{21} + a_{11} - a_{22})b_{22} \\
m_7 &:= a_{22}(b_{11} + b_{22} - b_{12} - b_{21})
\end{aligned}$$

$$\begin{aligned}
c_{11} &:= m_2 + m_3 \\
c_{12} &:= m_1 + m_2 + m_5 + m_6 \\
c_{21} &:= m_1 + m_2 + m_4 - m_7 \\
c_{22} &:= m_1 + m_2 + m_4 + m_5
\end{aligned}$$

# Matrix Multiplication

## Algorithm 4: Strassen's algorithm (cache-oblivious)

– 7 recursive $\frac{N}{2} \times \frac{N}{2}$ matrix multiplications + $O(1)$ matrix sums

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned}
m_1 &:= (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11}) \\
m_2 &:= a_{11} b_{11} \\
m_3 &:= a_{12} b_{21} \\
m_4 &:= (a_{11} - a_{21})(b_{22} - b_{12}) \\
m_5 &:= (a_{21} + a_{22})(b_{12} - b_{11}) \\
m_6 &:= (a_{12} - a_{21} + a_{11} - a_{22}) b_{22} \\
m_7 &:= a_{22}(b_{11} + b_{22} - b_{12} - b_{21})
\end{aligned}$$

$$\begin{aligned}
c_{11} &:= m_2 + m_3 \\
c_{12} &:= m_1 + m_2 + m_5 + m_6 \\
c_{21} &:= m_1 + m_2 + m_4 - m_7 \\
c_{22} &:= m_1 + m_2 + m_4 + m_5
\end{aligned}$$

– # I/Os if bit interleaved or ( row major and $M = \Omega(B^2)$ )

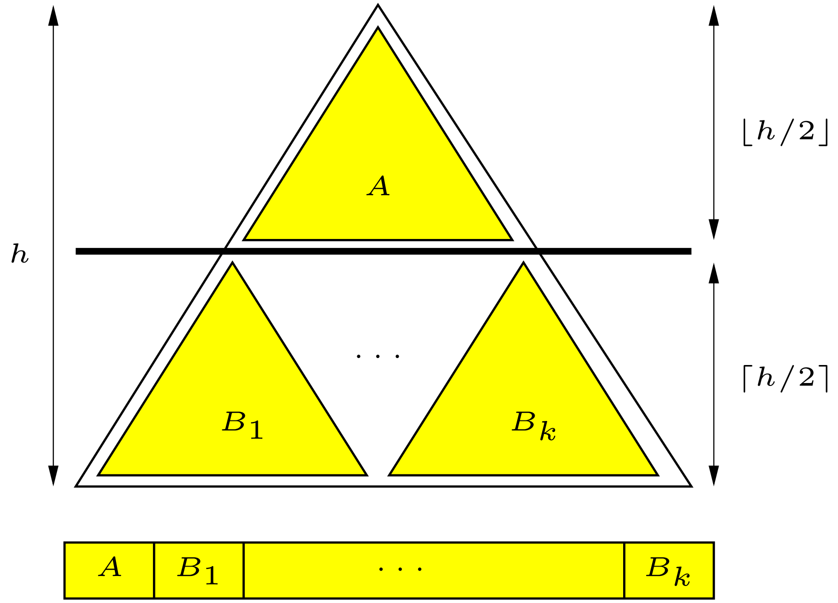
$$T(N) \leq \begin{cases} O(\frac{N^2}{B}) & \text{if } N \leq \varepsilon\sqrt{M} \\ 7 \cdot T\left(\frac{N}{2}\right) + O\left(\frac{N^2}{B}\right) & \text{otherwise} \end{cases}$$

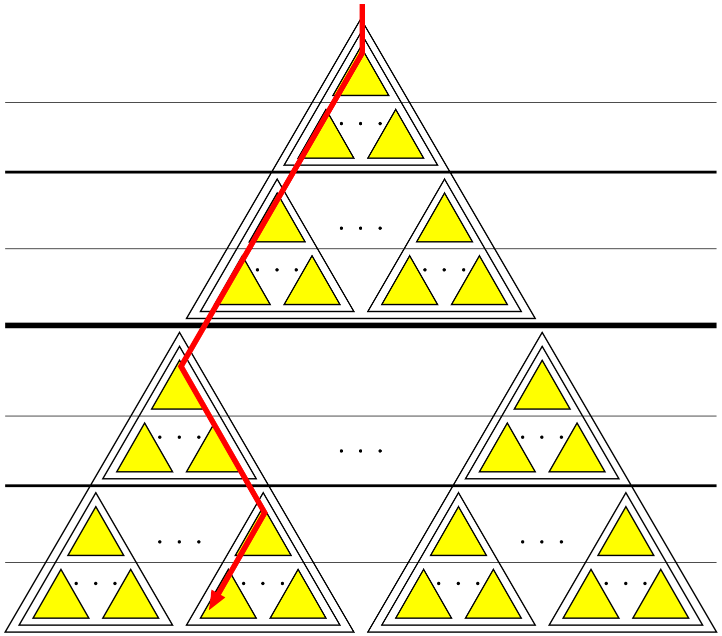
$$T(N) \leq O\left(\frac{N^{\log_2 7}}{B\sqrt{M}}\right) = O\left(\frac{N^{\log_2 7}}{B\sqrt{M}^{\log_2 7 - 2}}\right) \qquad \log_2 7 \approx 2.81$$

# Cache-Oblivious Search Trees

# Static Cache-Oblivious Trees

Recursive memory layout $\equiv$ van Emde Boas layout



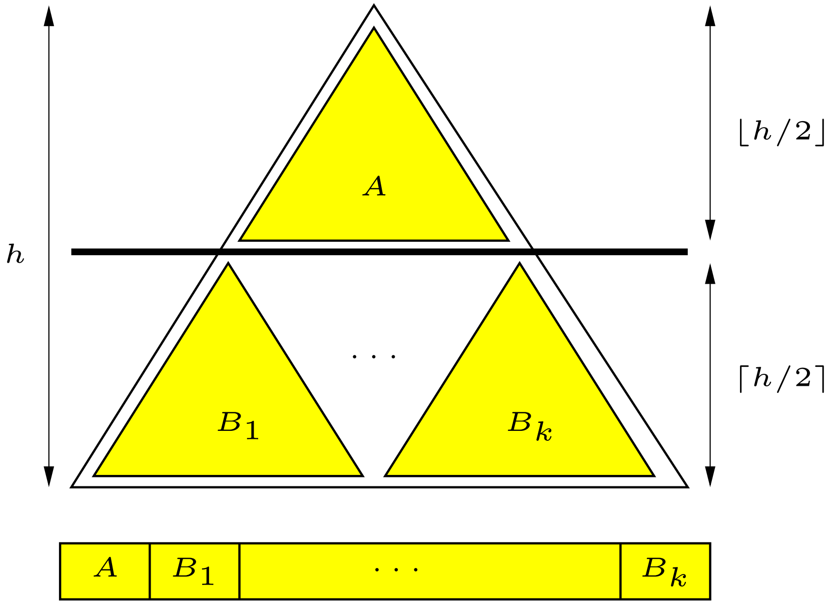



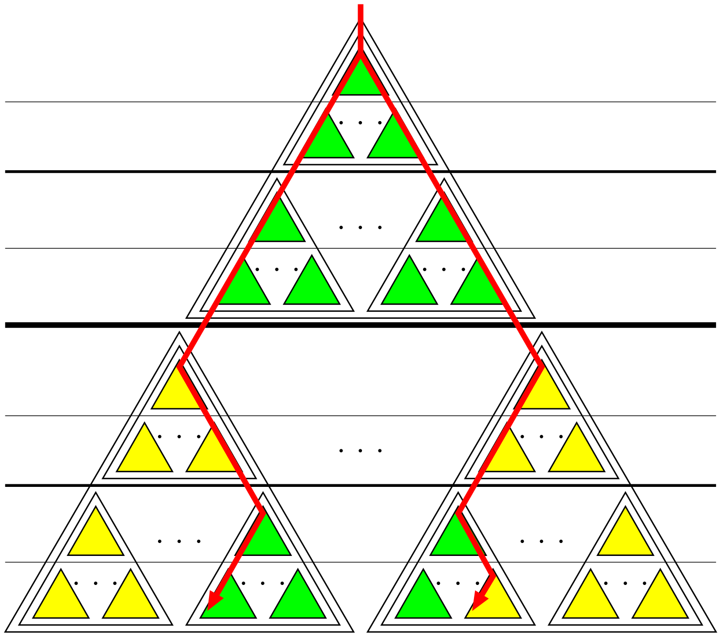
Searches use $O(\log_B N)$ I/Os

Prokop 1999

# Static Cache-Oblivious Trees

Recursive memory layout $\equiv$ van Emde Boas layout



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

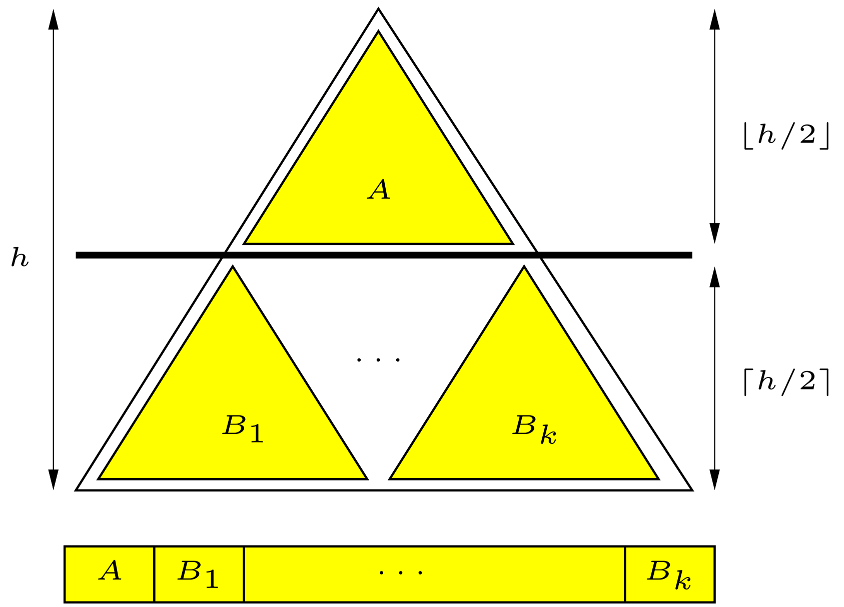
Range reportings use

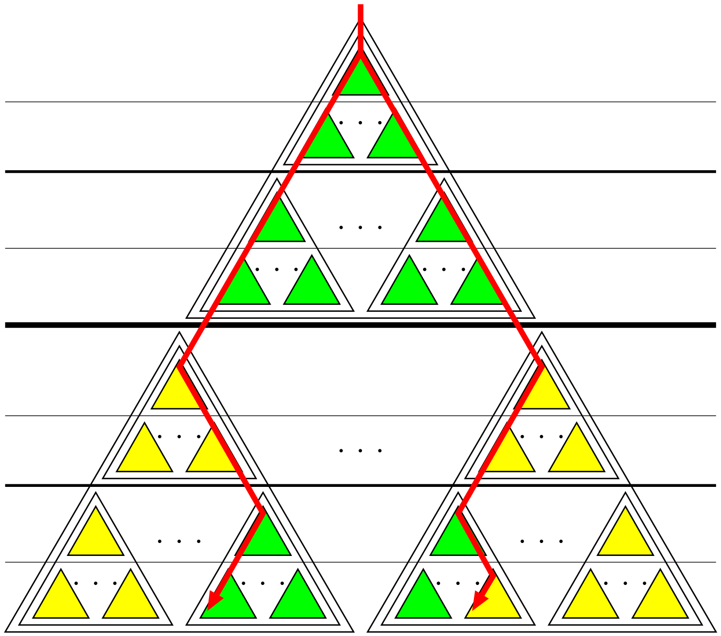
$O\left(\log_B N + \frac{k}{B}\right)$ I/Os

Prokop 1999

# Static Cache-Oblivious Trees

Recursive memory layout $\equiv$ van Emde Boas layout



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

Range reportings use

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

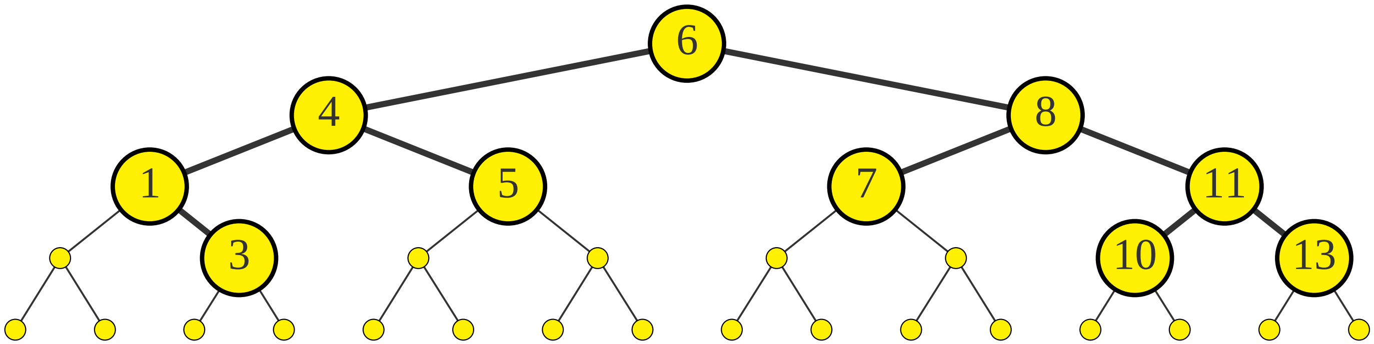
Prokop 1999

Best possible $(\log_2 e + o(1)) \log_B N$

Bender, Brodal, Fagerberg, Ge, He, Hu
Iacono, López-Ortiz 2003

# Dynamic Cache-Oblivious Trees

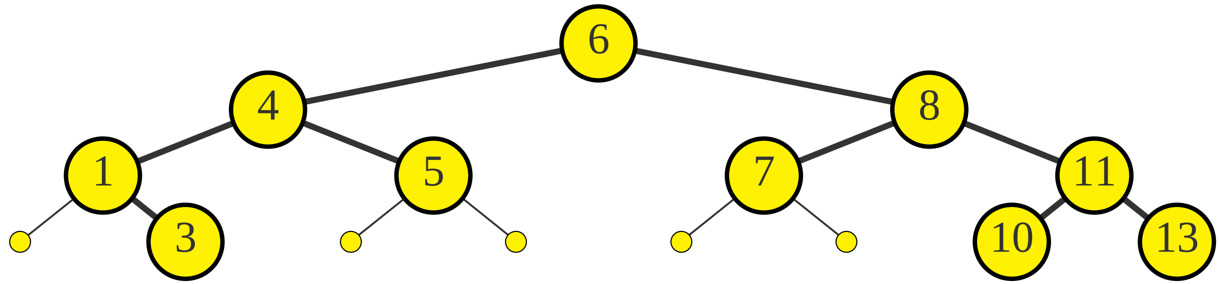
- Embed a dynamic tree of small height into a complete tree
- Static van Emde Boas layout
- Rebuild data structure whenever $N$ doubles or halves




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

Brodal, Fagerberg, Jacob 2001

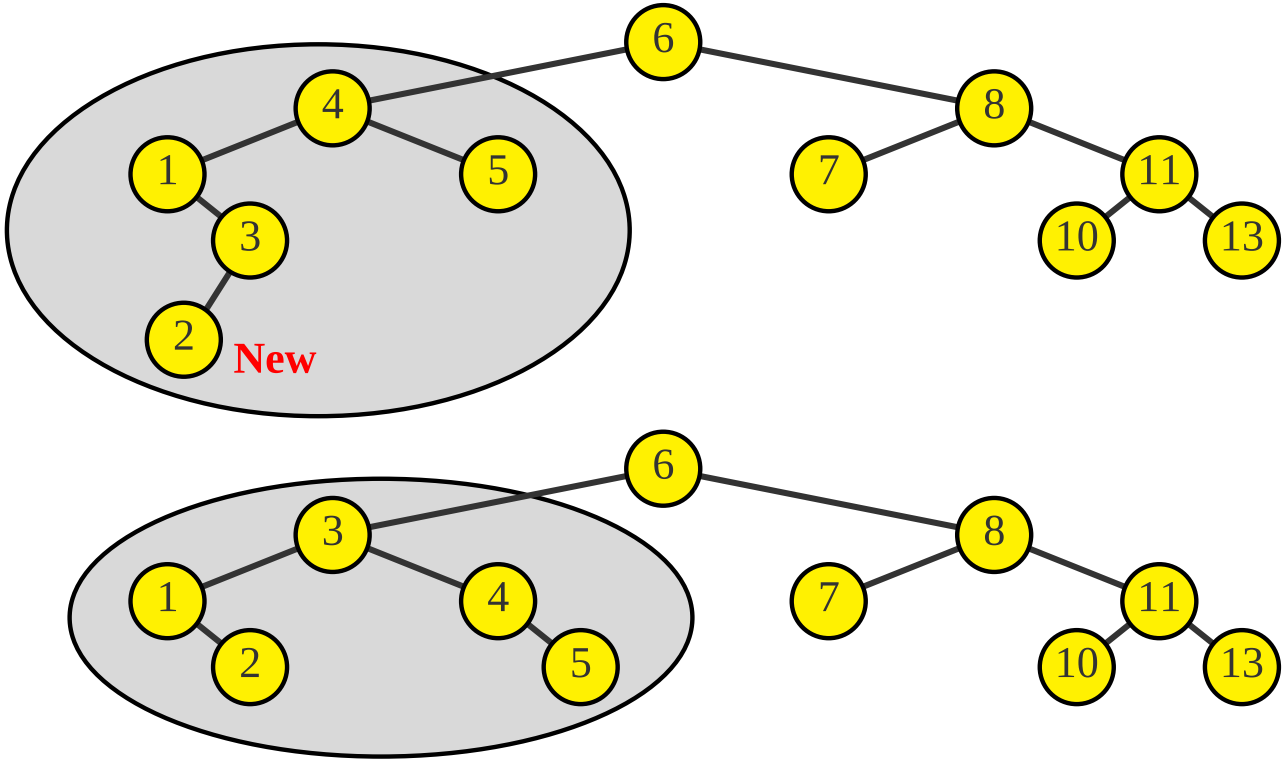
# Example




$$\Downarrow$$

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 4 | 8 | 1 | – | 3 | 5 | – | – | 7 | – | – | 11 | 10 | 13 |

# 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 nearest 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 sketch* Consider two redistributions of $v_i$

- Before second redistribution a child $v_{i+1}$ of $v_i$ has $\rho(v_{i+1}) > \tau_{i+1}$
- After the first redistribution $\rho(v_i) \leq \tau_i$, $\rho(v_{i+1}) \leq \tau_i$
- 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 $v_i$

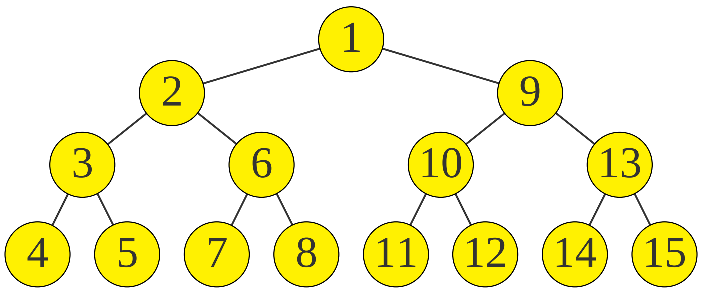
$$\frac{m(v_i)}{m(v_{i+1}) \cdot \Delta} \leq \frac{2}{\Delta}$$

- Total insertion cost per element

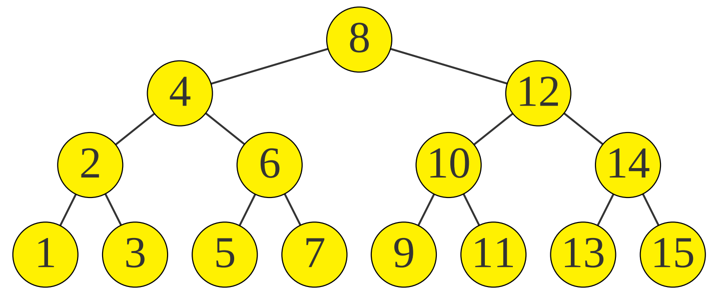
$$\sum_{i=0}^{H} \frac{2}{\Delta} = O(\log^2 N) \qquad \square$$

Andersson and Lai 1990

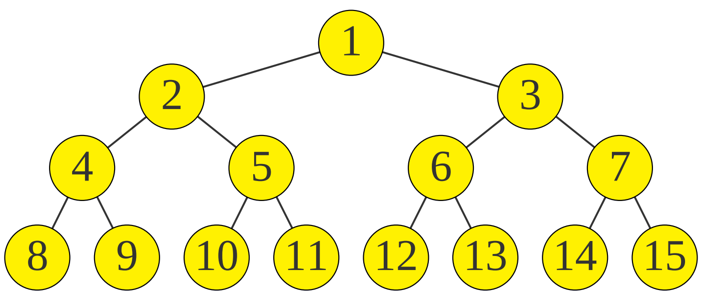
# Memory Layouts of Trees



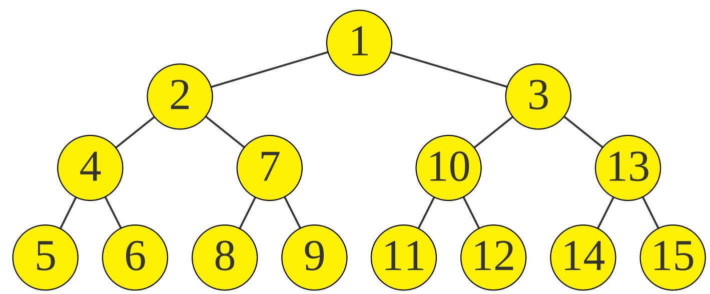

DFS




inorder



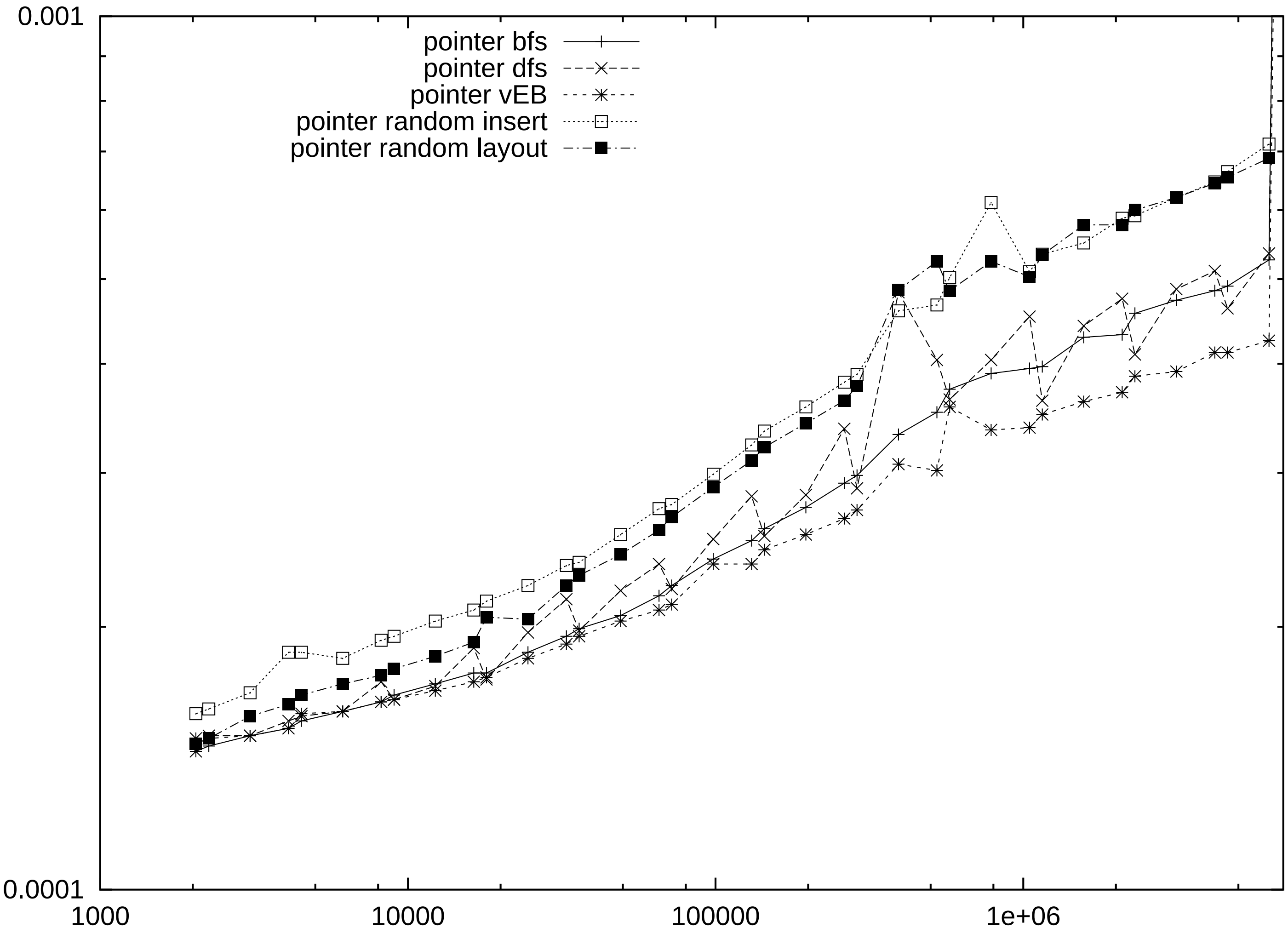

BFS




van Emde Boas

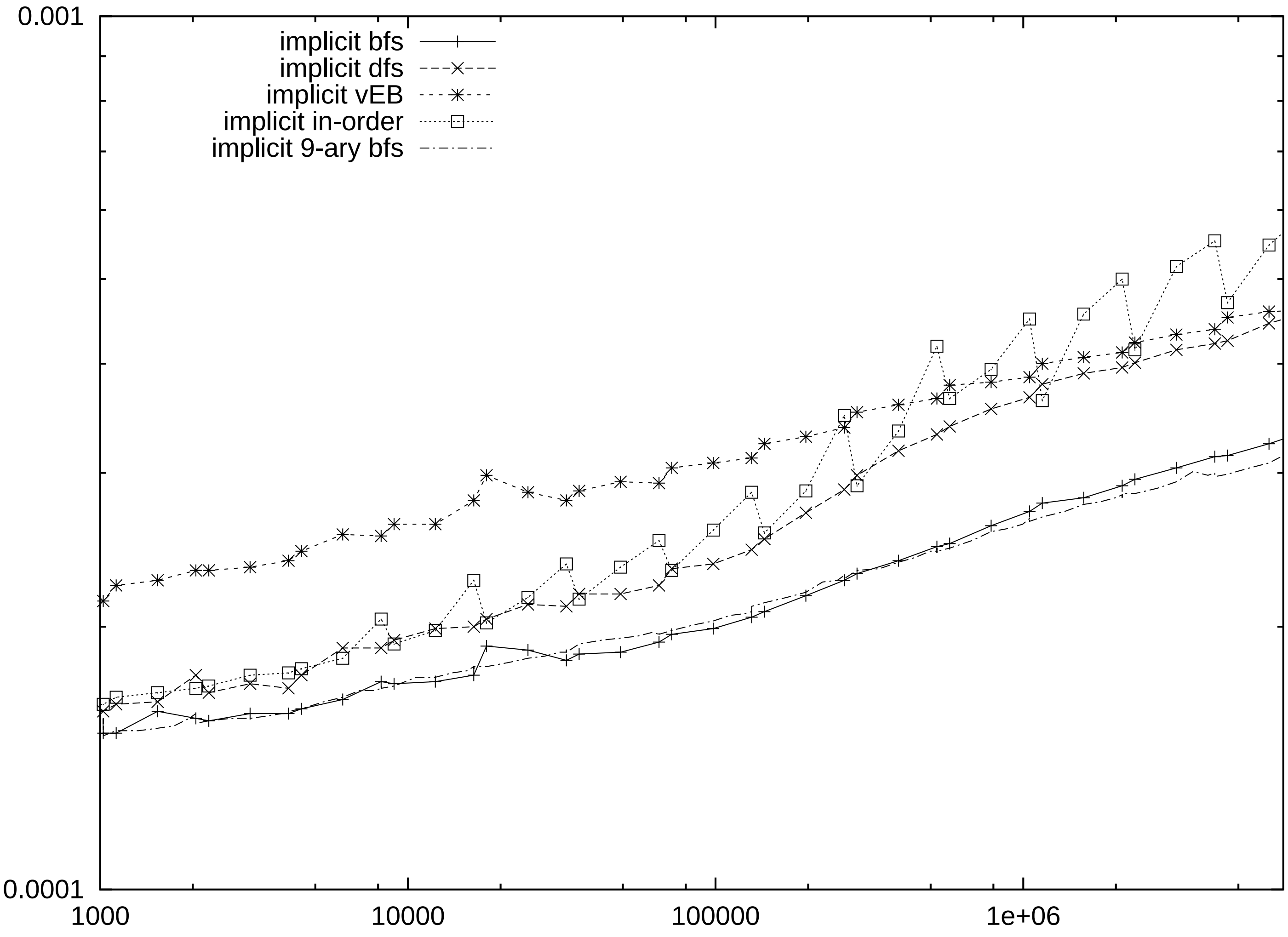
(in theory best)

# Searches in Pointer Based Layouts



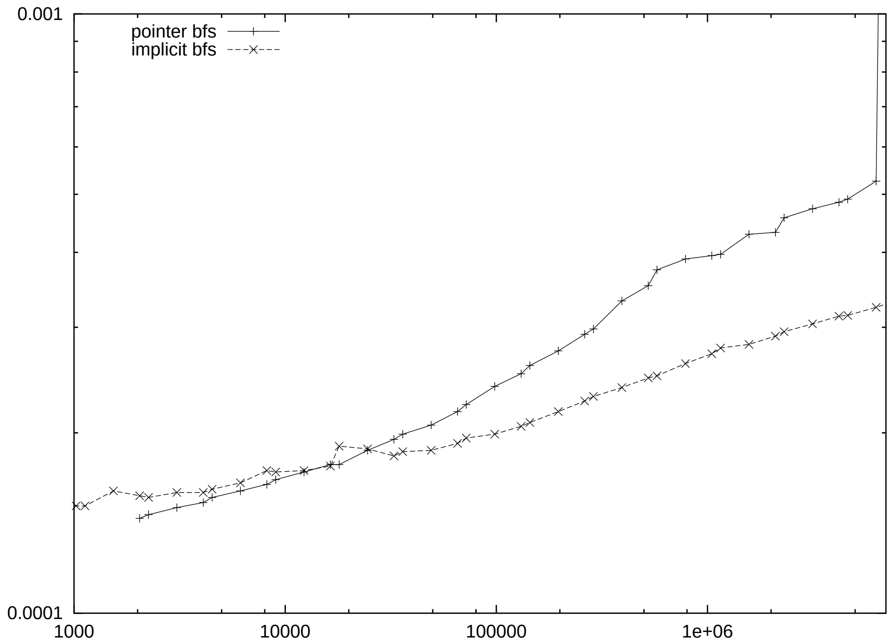

- van Emde Boas layout wins, followed by the BFS layout

# Searches with Implicit Layouts

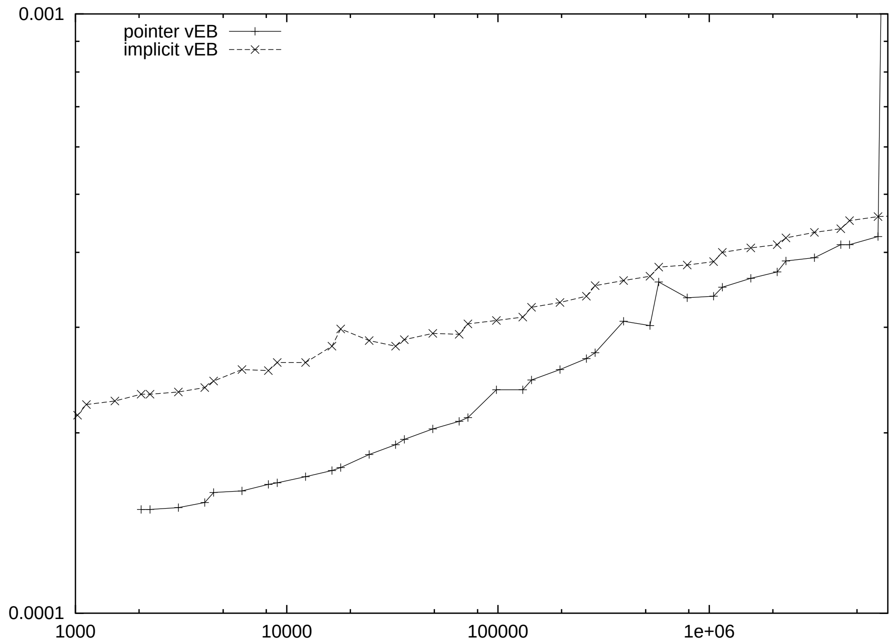



- BFS layout wins due to simplicity and caching of topmost levels
- van Emde Boas layout requires quite complex index computations

# Implicit vs Pointer Based Layouts



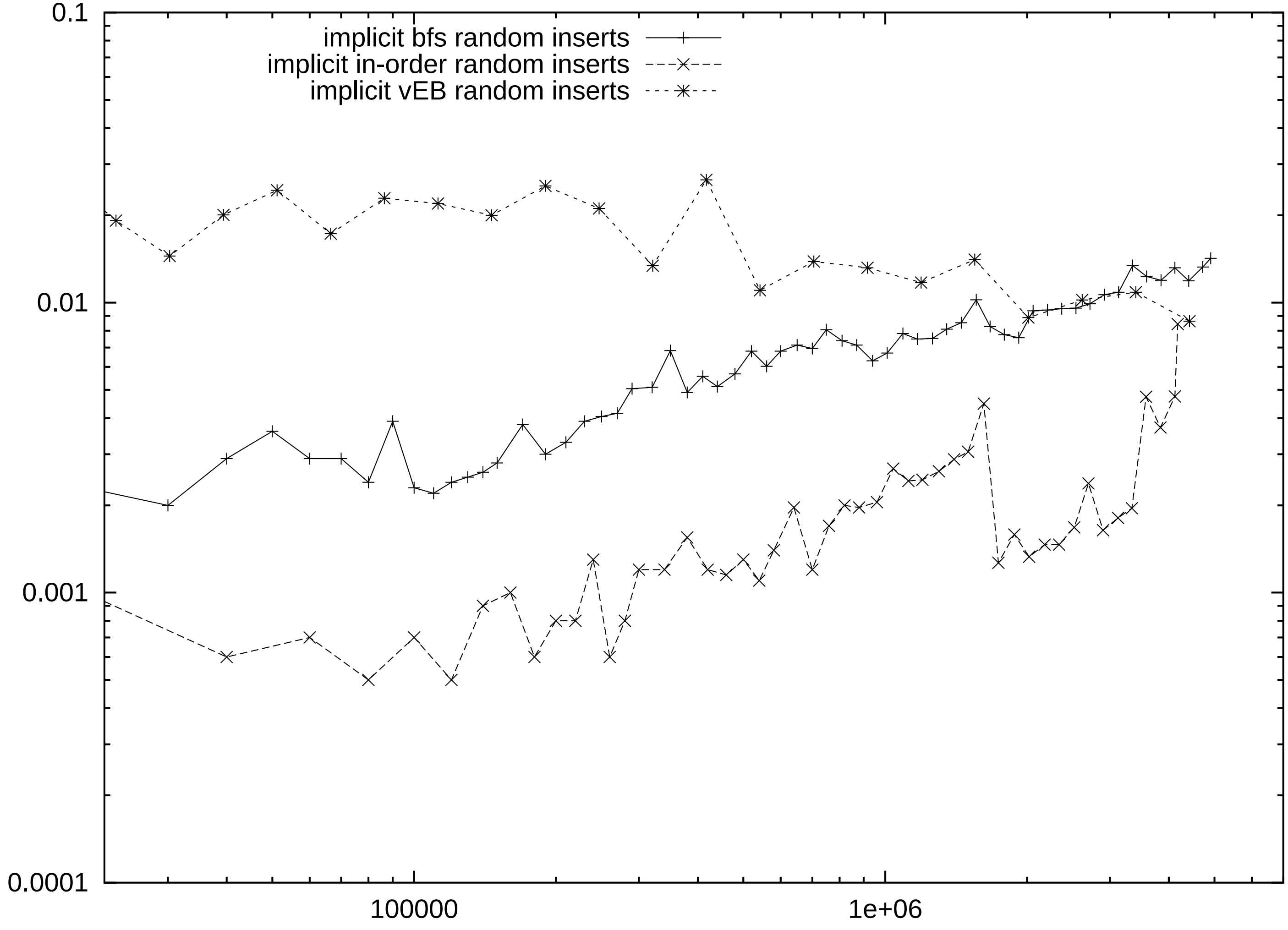

BFS layout




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

- Dynamic 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 |

# Cache-Oblivious Sorting

# Sorting Problem

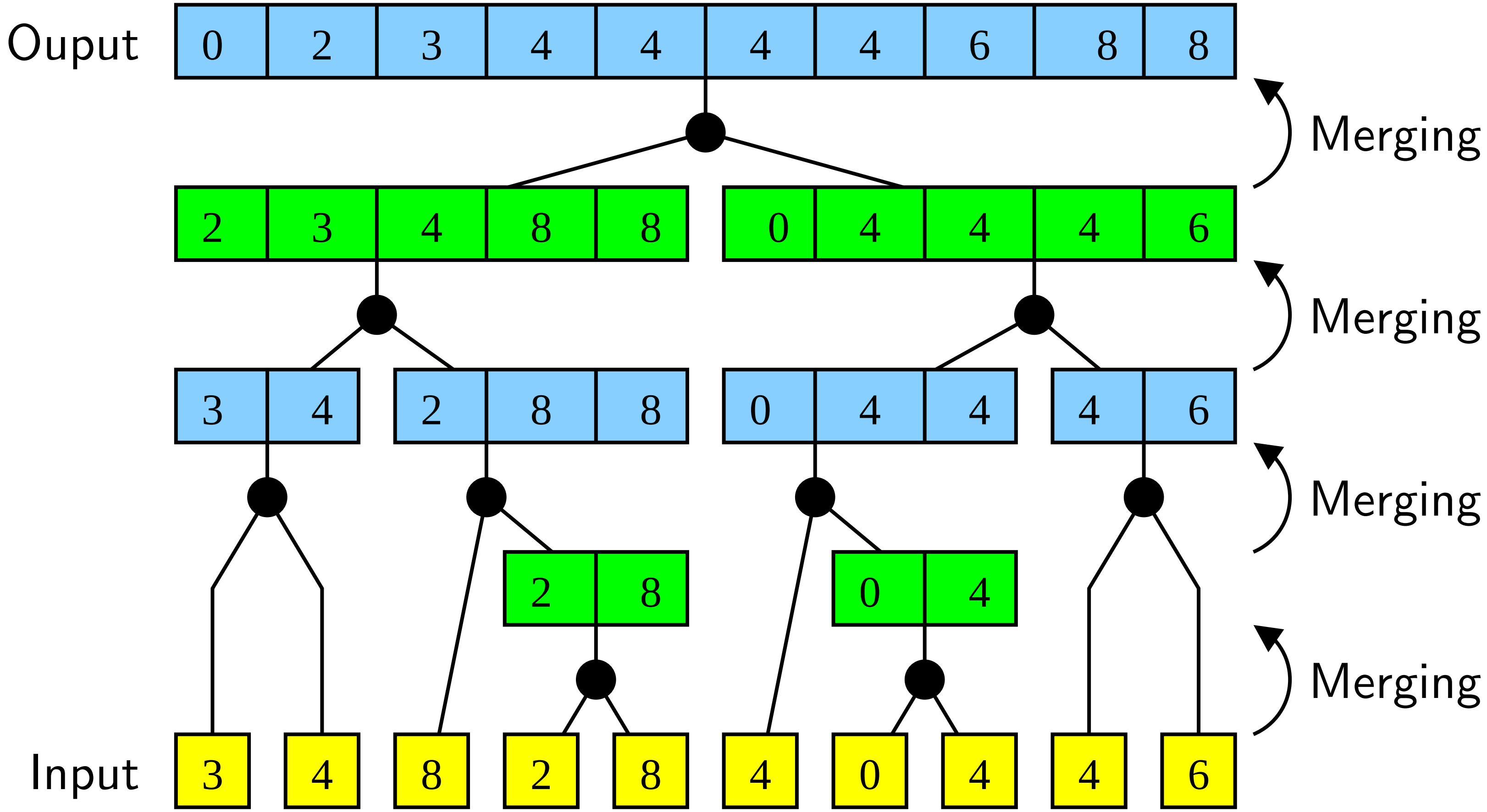
- Input : array containing $x_1, \ldots, x_N$
- Output : array with $x_1, \ldots, x_N$ in sorted order
- Elements can be compared and copied

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| D | E | M | E | T | R | E | S | C | U |

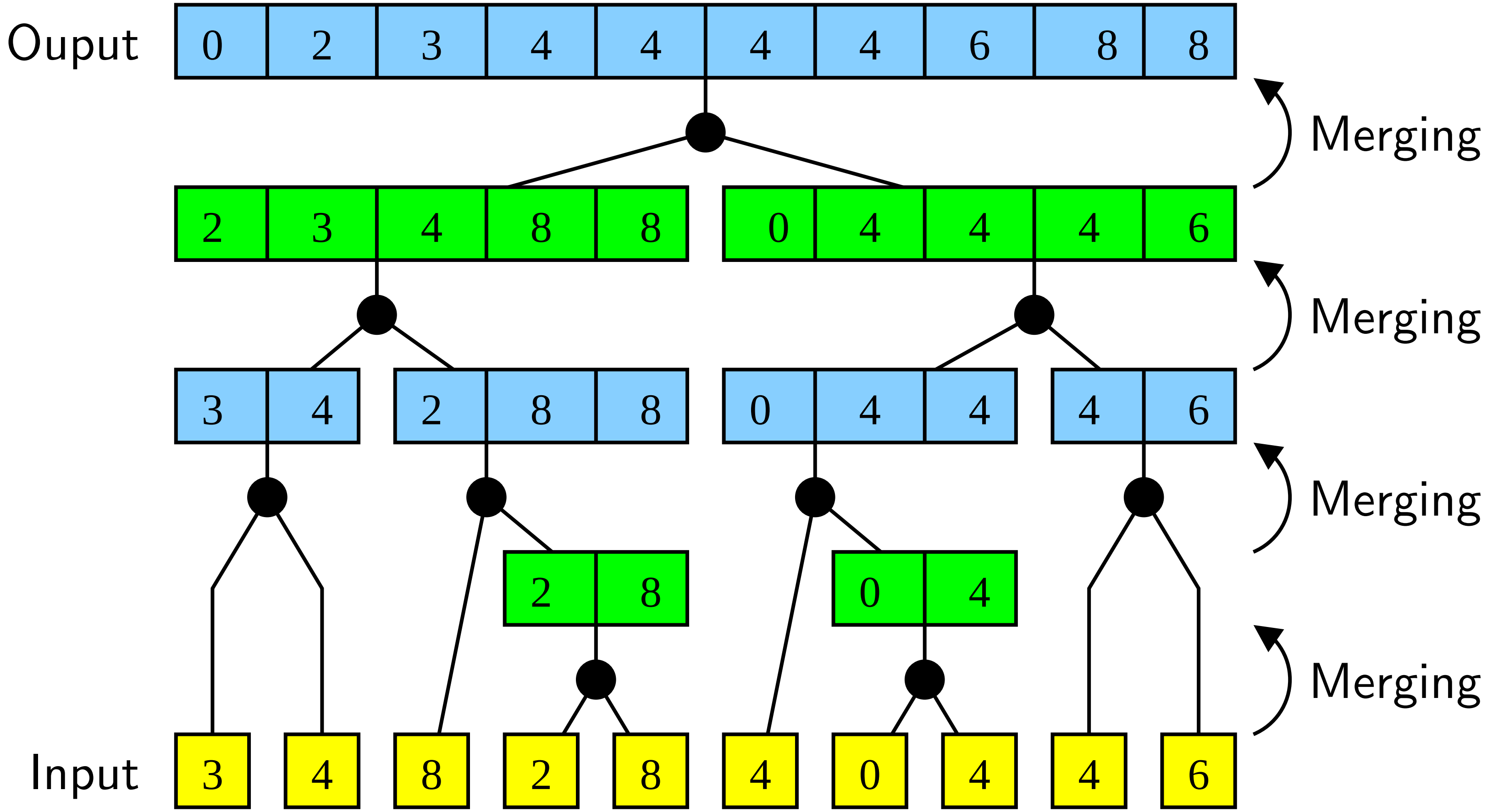
$$\Downarrow$$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| C | D | E | E | E | M | R | S | T | U |

# Binary Merge-Sort



# Binary Merge-Sort




- Recursive; two arrays; size $O(M)$ internally in cache
- $O(N \log N)$ comparisons
- $O\left(\frac{N}{B} \log_2 \frac{N}{M}\right)$ I/Os

# Merge-Sort

| Degree | I/O |
|---|---|
| $2$ | $O\left(\frac{N}{B}\log_2\frac{N}{M}\right)$ |
| $d$ <br> $(d \leq \frac{M}{B}-1)$ | $O\left(\frac{N}{B}\log_d\frac{N}{M}\right)$ |
| $\Theta\left(\frac{M}{B}\right)$ | $O\left(\frac{N}{B}\log_{M/B}\frac{N}{M}\right) = O(\mathrm{Sort}_{M,B}(N))$ |

Aggarwal and Vitter 1988

# Funnel-Sort

| | |
|---|---|
| $2$ <br> $(M \geq B^{1+\varepsilon})$ | $O(\frac{1}{\varepsilon}\,\mathrm{Sort}_{M,B}(N))$ |

Frigo, Leiserson, Prokop and Ramachandran 1999

Brodal and Fagerberg 2002

# Lower Bound

Brodal and Fagerberg 2003

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

One algorithm, two machines, $B_1 \leq B_2$

Trade-off

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

# Lower Bound

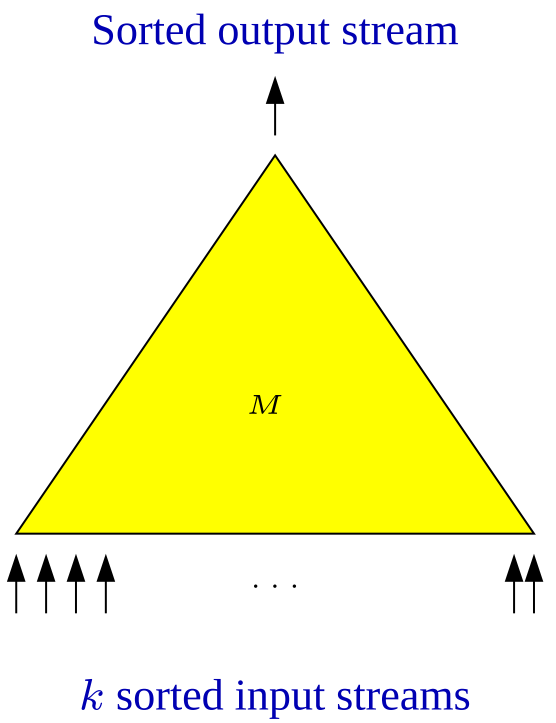
| | Assumption | I/Os |
|---|---|---|
| Lazy Funnel-sort | $B \leq M^{1-\varepsilon}$ | $(a)\ B_2 = M^{1-\varepsilon}: \quad \mathrm{Sort}_{B_2,M}(N)$ |
| | | $(b)\ B_1 = 1: \quad \mathrm{Sort}_{B_1,M}(N) \cdot \frac{1}{\varepsilon}$ |
| Binary Merge-sort | $B \leq M/2$ | $(a)\ B_2 = M/2: \quad \mathrm{Sort}_{B_2,M}(N)$ |
| | | $(b)\ B_1 = 1: \quad \mathrm{Sort}_{B_1,M}(N) \cdot \log M$ |

Corollary $(a) \Rightarrow (b)$

# Funnel-Sort

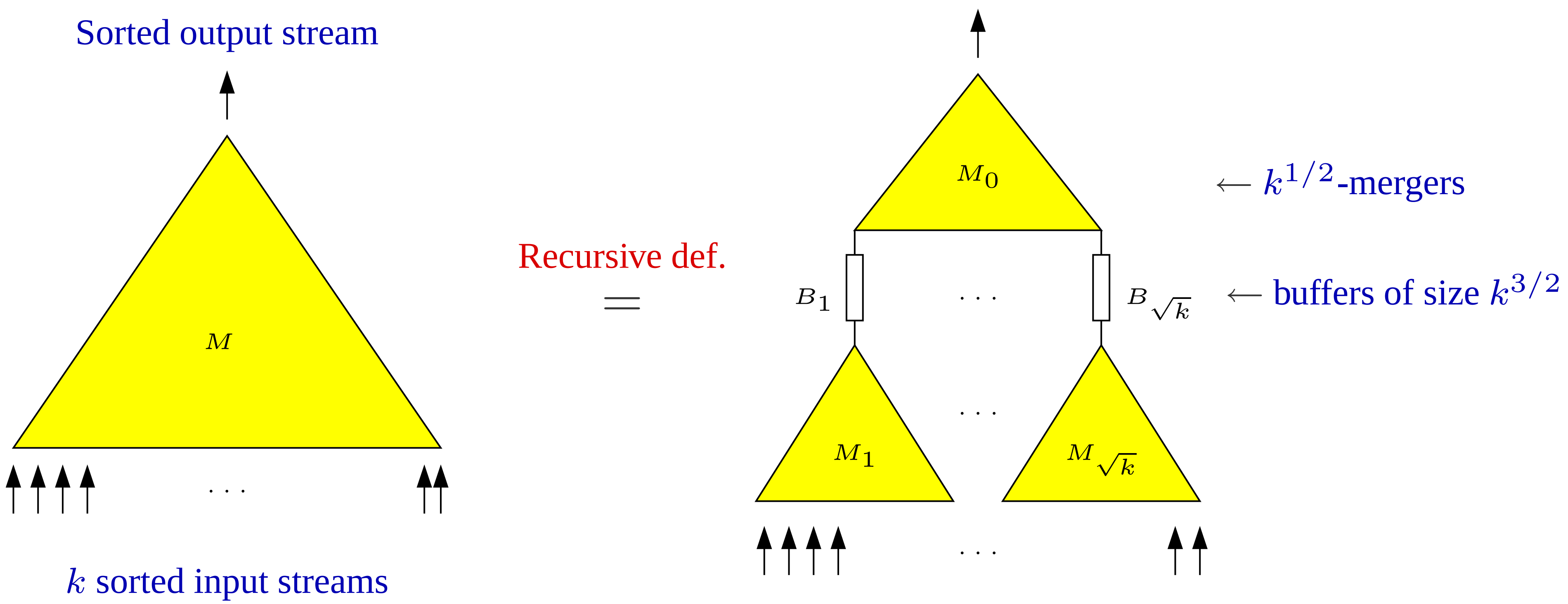
# $k$-merger

Frigo et al., FOCS'99



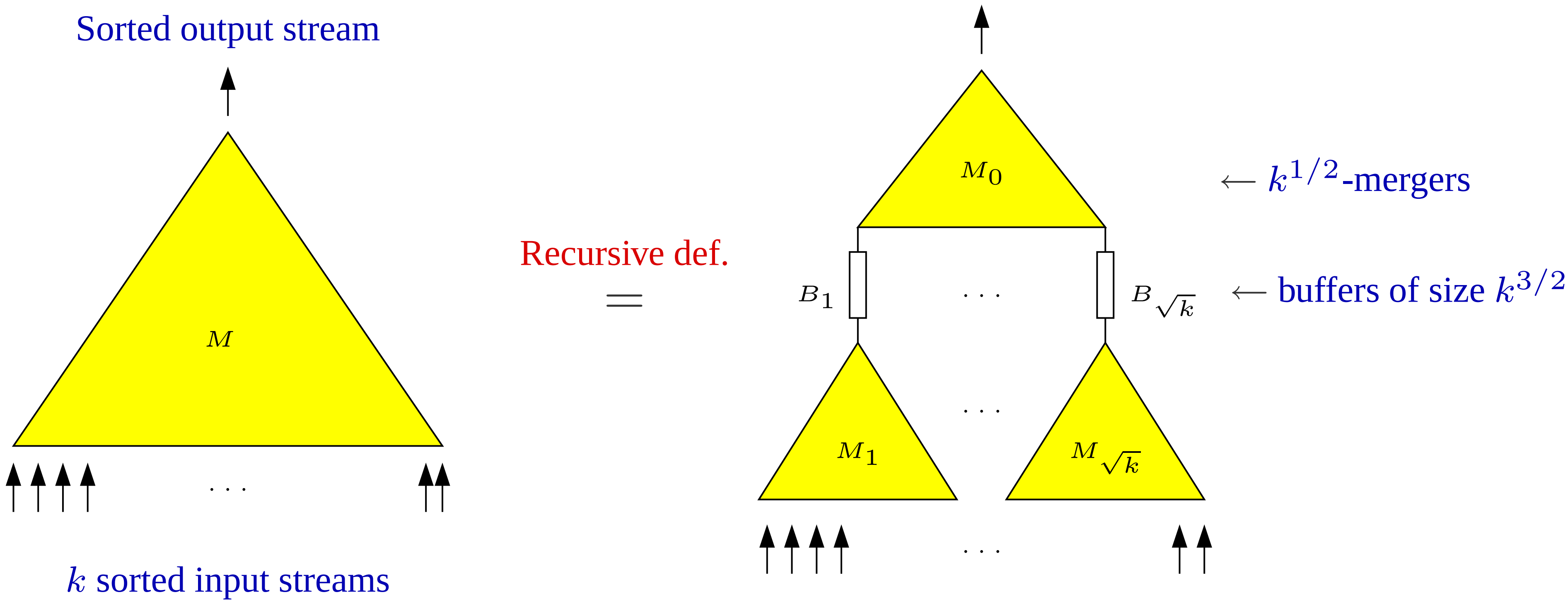
# $k$-merger

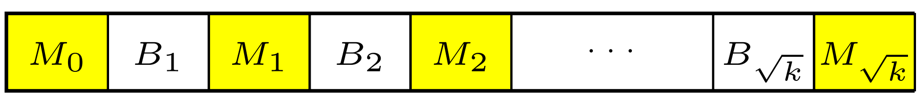
Frigo et al., FOCS'99



# $k$-merger

Frigo et al., FOCS'99

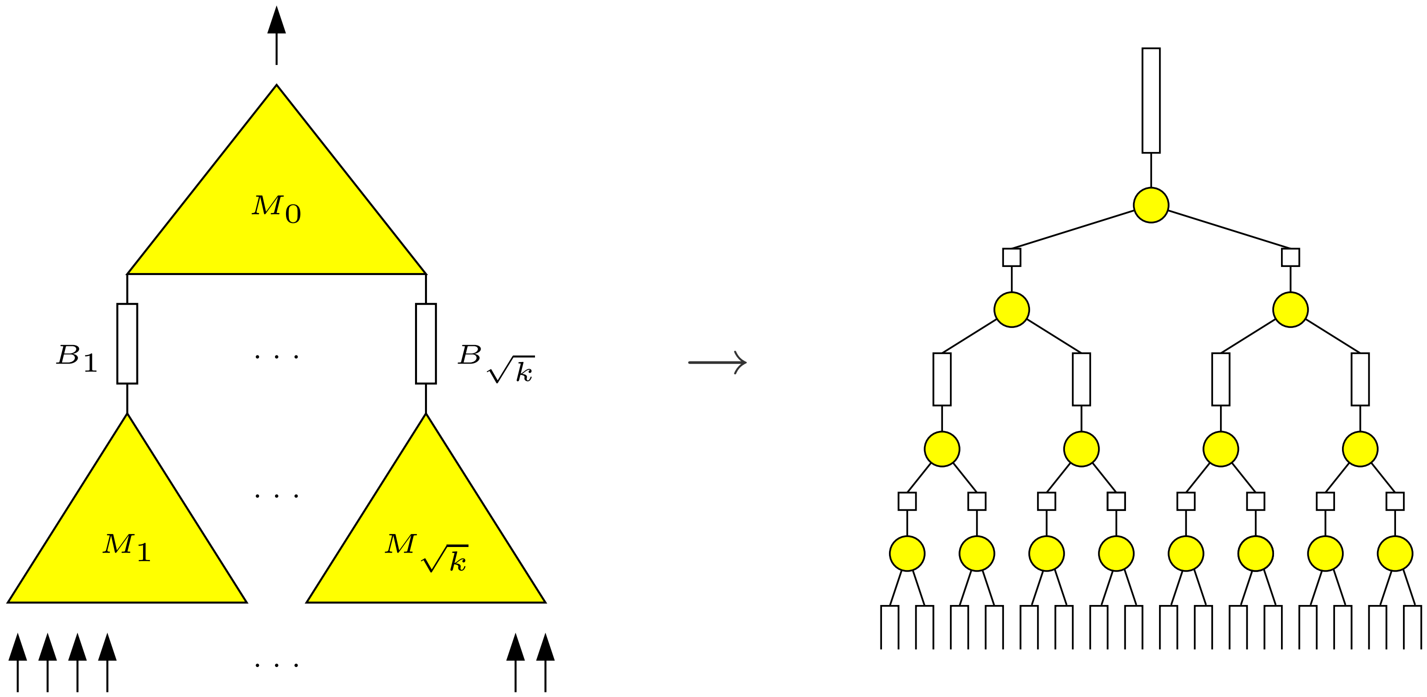



Recursive Layout

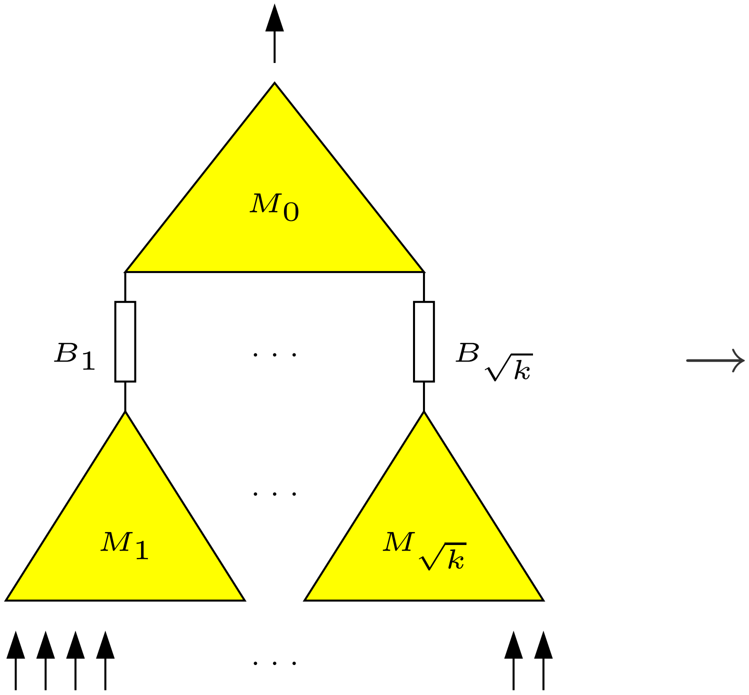
# Lazy $k$-merger

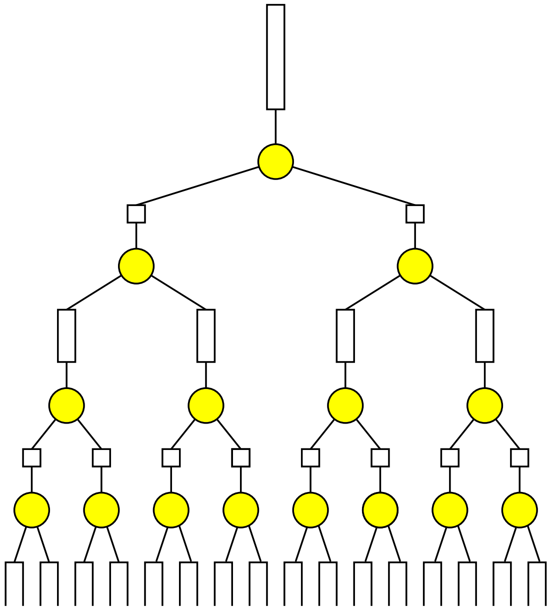
Brodal and Fagerberg 2002



# Lazy $k$-merger

Brodal and Fagerberg 2002

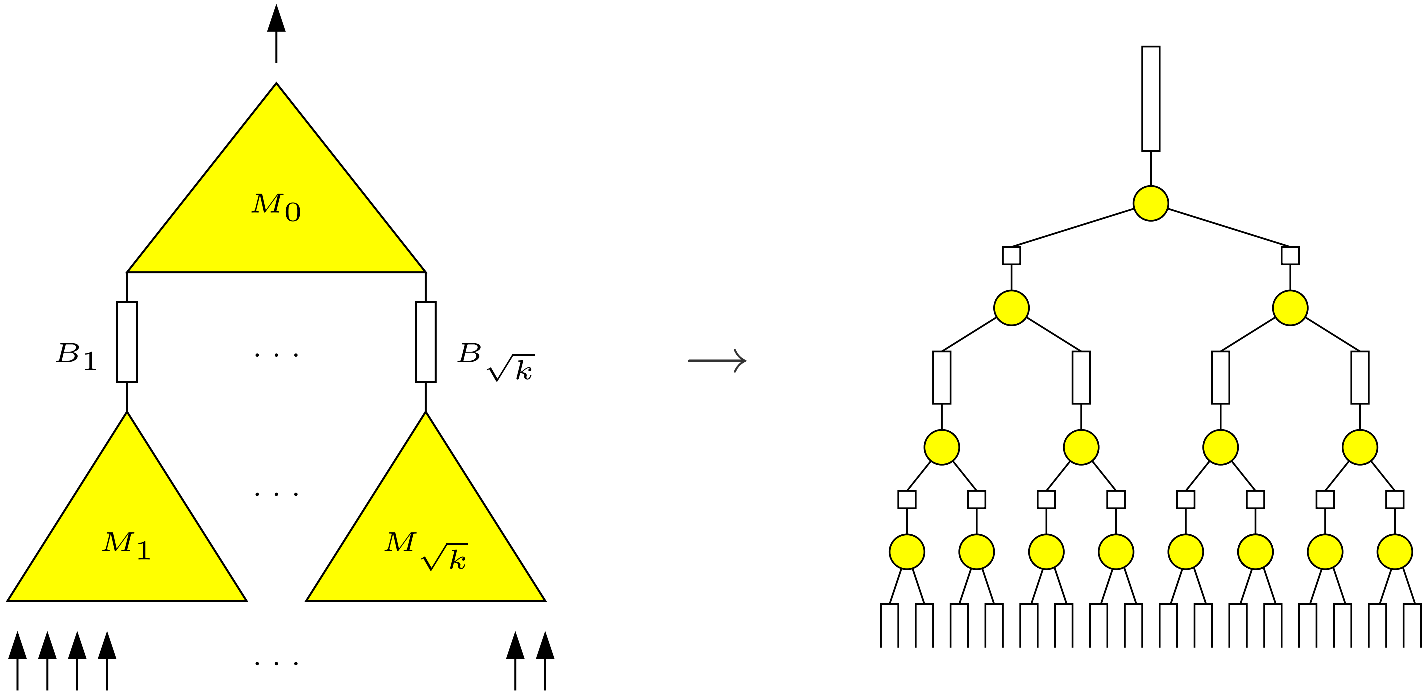


Procedure **Fill**($v$)
- **while** out-buffer not full
  - **if** left in-buffer empty
    - **Fill**(left child)
  - **if** right in-buffer empty
    - **Fill**(right child)
  - perform one merge step

# Lazy $k$-merger

Brodal and Fagerberg 2002




Procedure **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

## Lemma

If $M \geq B^2$ and output buffer has size $k^3$ then $O(\frac{k^3}{B}\log_M(k^3) + k)$ I/Os are done during an invocation of **Fill**(root)

# Funnel-Sort

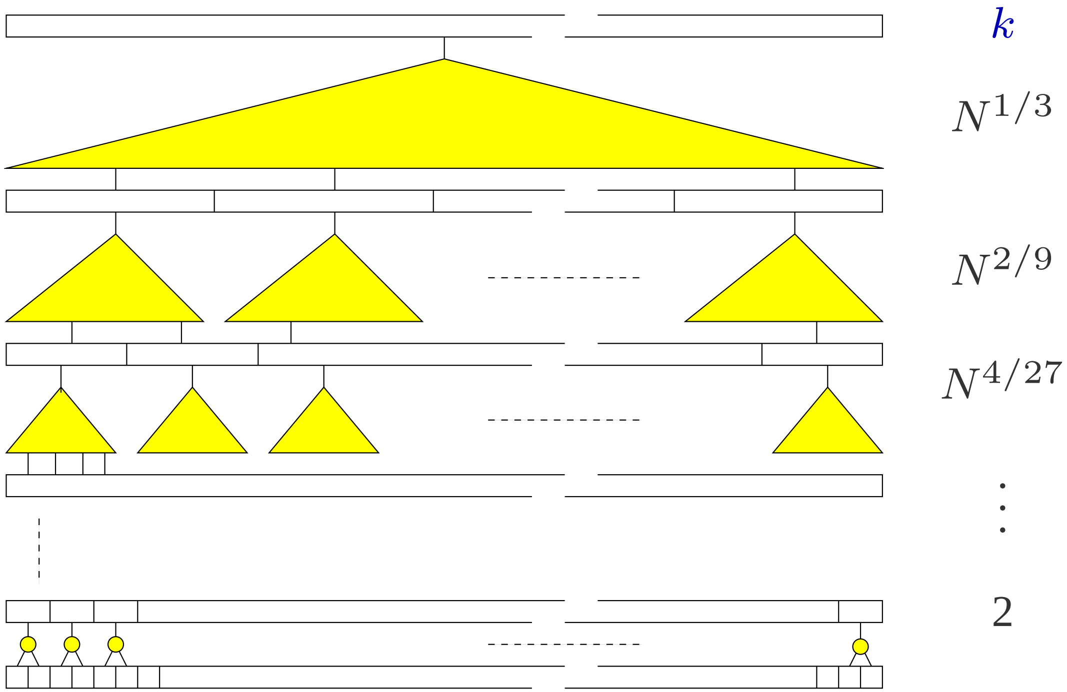
Brodal and Fagerberg 2002

Frigo, Leiserson, Prokop and Ramachandran 1999

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

Recursively **Funnel-Sort** each segment

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



# Funnel-Sort

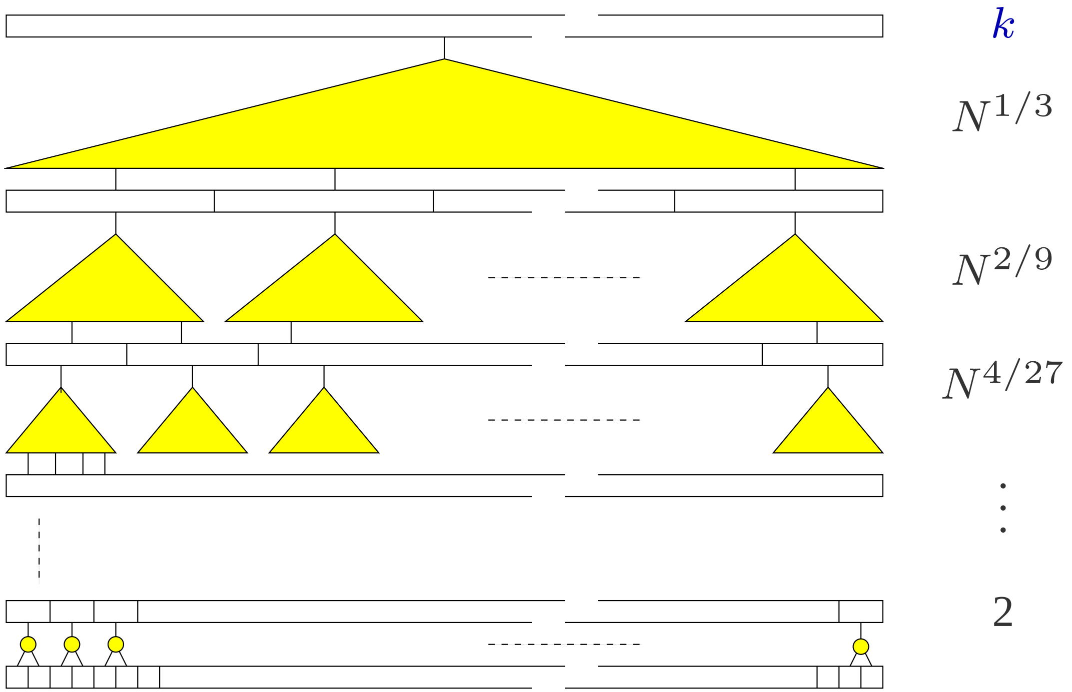
Brodal and Fagerberg 2002

Frigo, Leiserson, Prokop and Ramachandran 1999

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

Recursively **Funnel-Sort** each segment

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

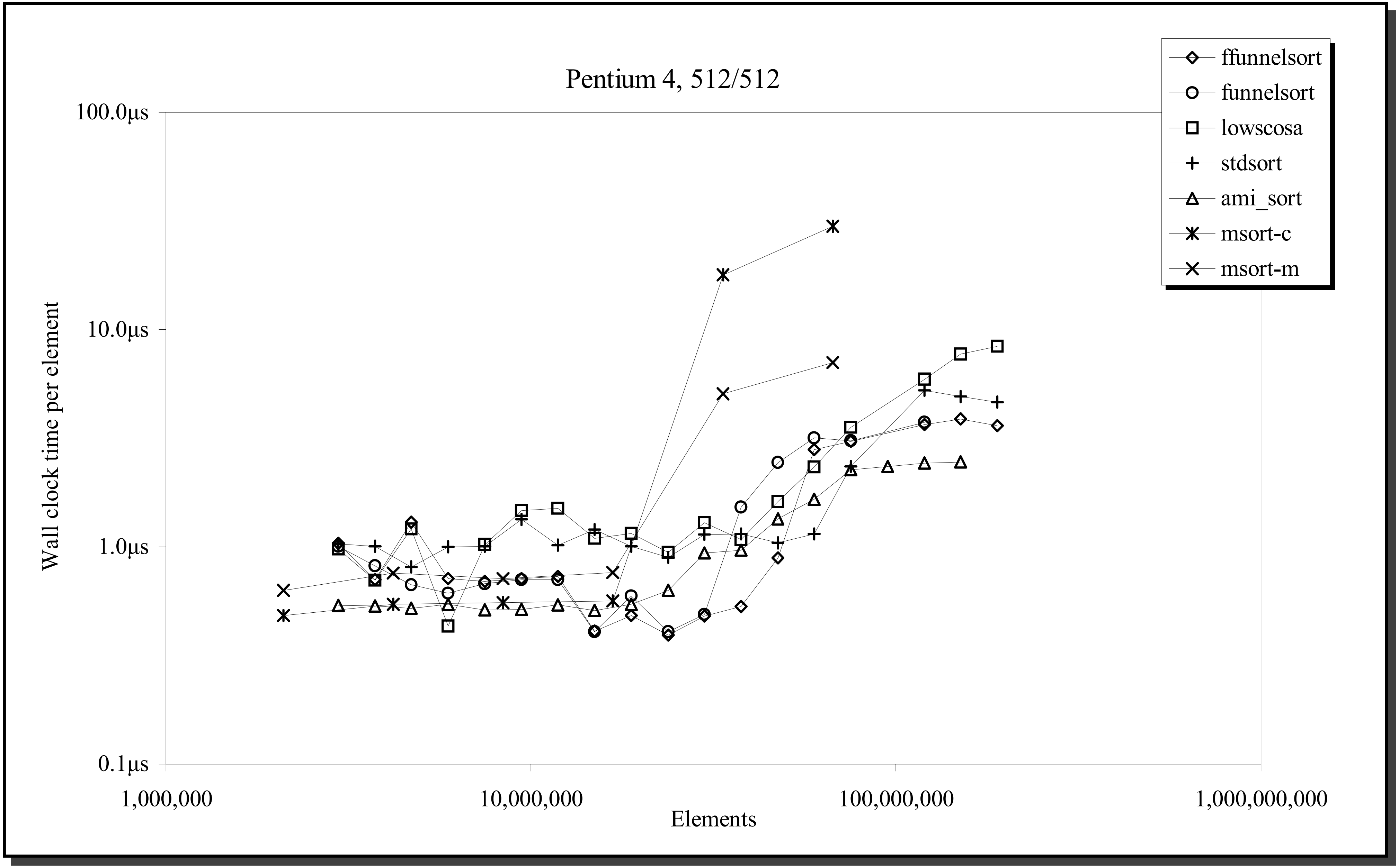



**Theorem** Funnel-Sort performs $O(\mathrm{Sort}_{M,B}(N))$ I/Os for $M \geq B^2$

# Hardware

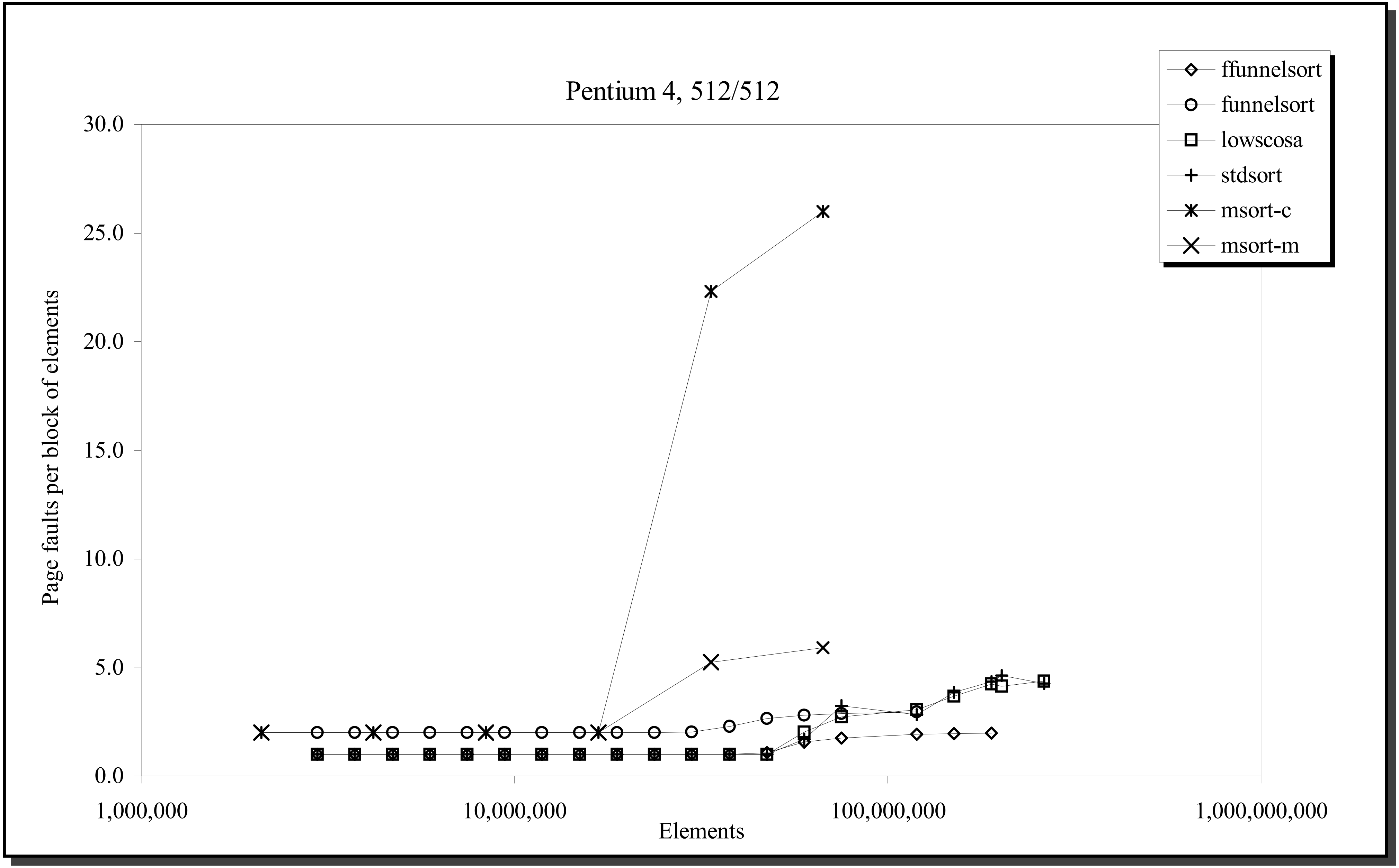
| Processor type | Pentium 4 | Pentium 3 | MIPS 10000 |
|---|---|---|---|
| Workstation | Dell PC | Delta PC | SGI Octane |
| Operating system | GNU/Linux Kernel version 2.4.18 | GNU/Linux Kernel version 2.4.18 | IRIX version 6.5 |
| Clock rate | 2400 MHz | 800 MHz | 175 MHz |
| Address space | 32 bit | 32 bit | 64 bit |
| Integer pipeline stages | 20 | 12 | 6 |
| L1 data cache size | 8 KB | 16 KB | 32 KB |
| L1 line size | 128 Bytes | 32 Bytes | 32 Bytes |
| L1 associativity | 4 way | 4 way | 2 way |
| L2 cache size | 512 KB | 256 KB | 1024 KB |
| L2 line size | 128 Bytes | 32 Bytes | 32 Bytes |
| L2 associativity | 8 way | 4 way | 2 way |
| TLB entries | 128 | 64 | 64 |
| TLB associativity | Full | 4 way | 64 way |
| TLB miss handler | Hardware | Hardware | Software |
| Main memory | 512 MB | 256 MB | 128 MB |

# Wall Clock



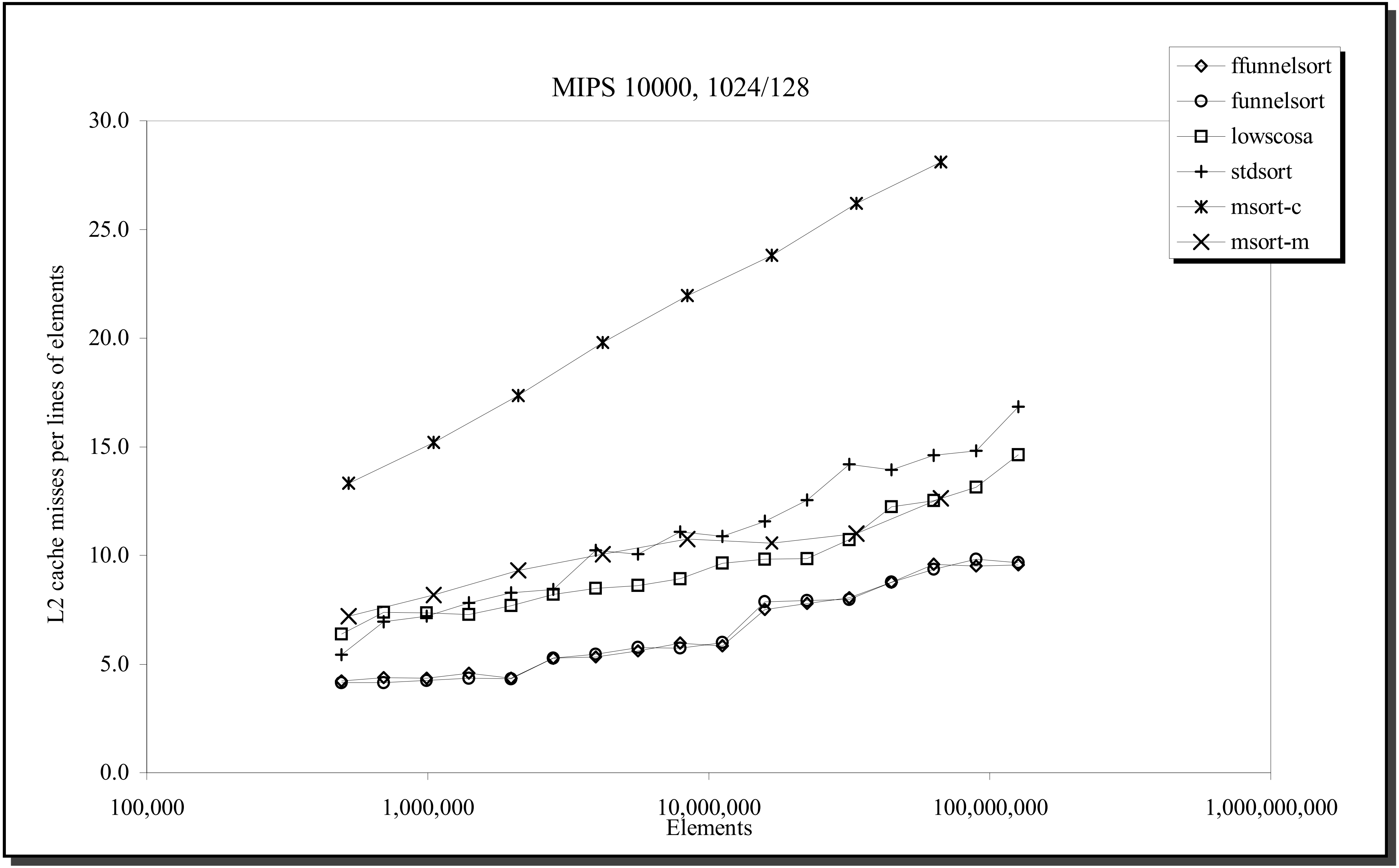

Kristoffer Vinther 2003

# Page Faults



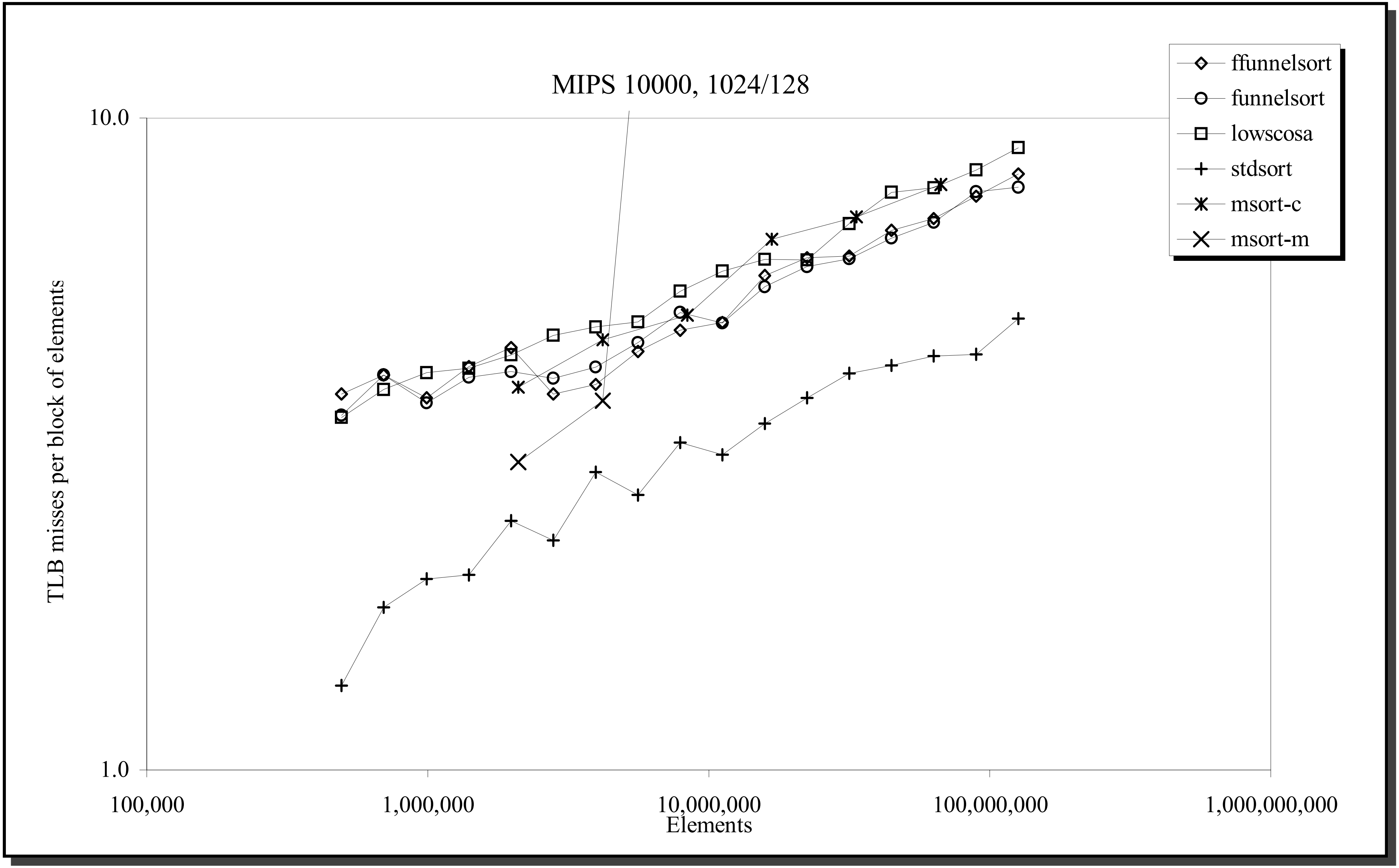

Kristoffer Vinther 2003

# Cache Misses




Kristoffer Vinther 2003

# TLB Misses




Kristoffer Vinther 2003

# Conclusions

Cache oblivious sorting

- is possible
- requires a tall cache assumption $M \geq B^{1+\varepsilon}$
- comparable performance with cache aware algorithms

Future work

- more experimental justification for the cache oblivious model
- limitations of the model — time space trade-offs
- tool-box for cache oblivious algorithms