

# **Cache-Oblivious Algorithms and Data Structures**

**Gerth Stølting Brodal**  
Aarhus University

# Customizing a Dell 650 (2004)



[www.dell.dk](http://www.dell.dk)



[www.intel.com](http://www.intel.com)

Processor speed	2.4 – 3.2 GHz
L3 cache size	0.5 – 2 MB
Memory	1/4 – 4 GB
Hard Disk	36 GB – 146 GB
	7.200 – 15.000 RPM
CD/DVD	8 – 48x
L2 cache size	256 – 512 KB
L2 cache line size	128 Bytes
L1 cache line size	64 Bytes
L1 cache size	16 KB

# Customizing a Dell 650 (2004)



[www.dell.dk](http://www.dell.dk)

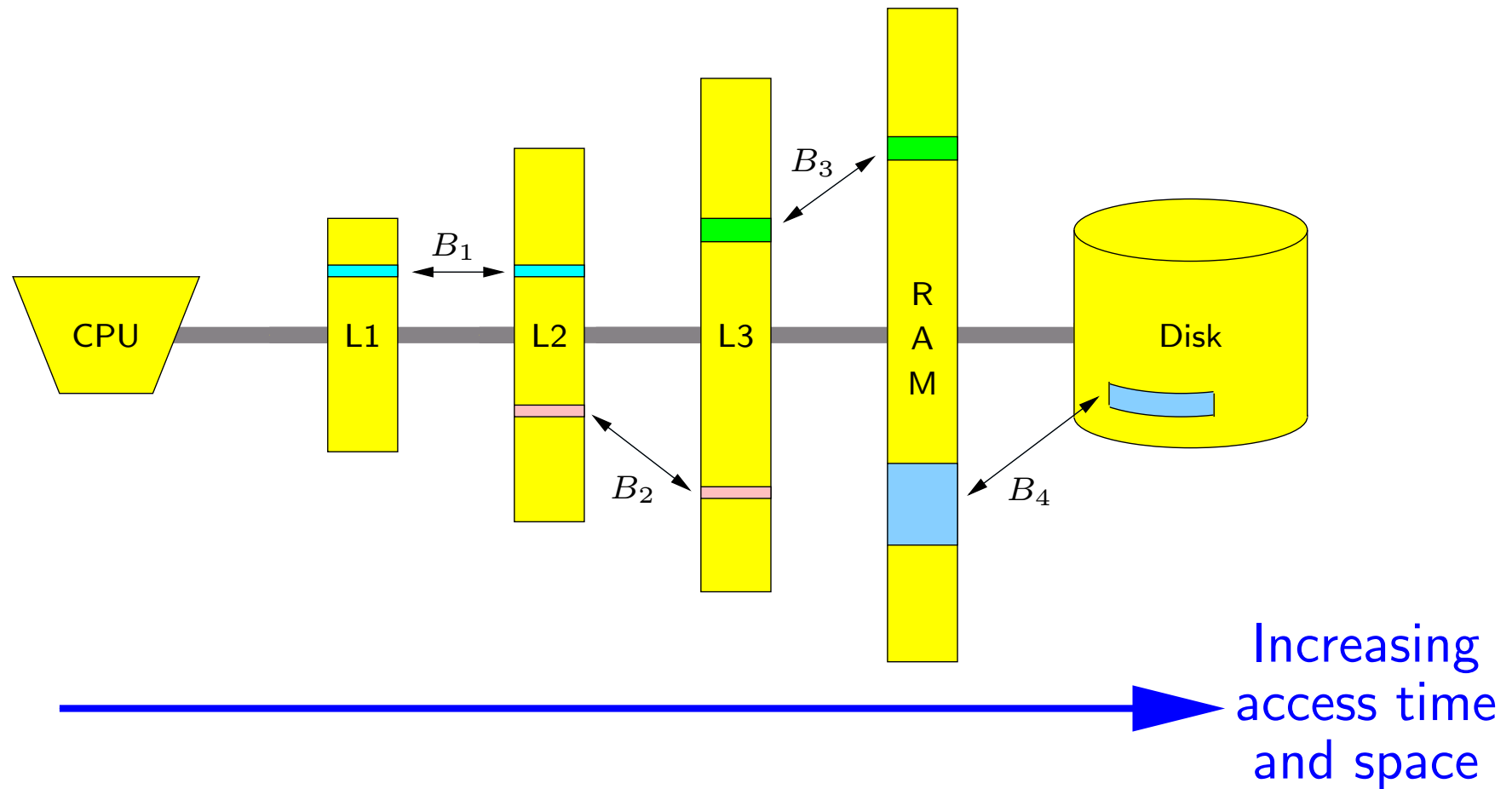


[www.intel.com](http://www.intel.com)

Processor speed	2.4 – 3.2 GHz
L3 cache size	0 – 8 MB
Memory	4 – 16 GB
Hard Disk	30 GB – 146 GB
CD/DVD	7.200 – 15.000 RPM
	8 – 48x
L2 cache size	256 – 512 KB
L2 cache line size	128 Bytes
L1 cache line size	64 Bytes
L1 cache size	16 KB

**Do we want to know?**

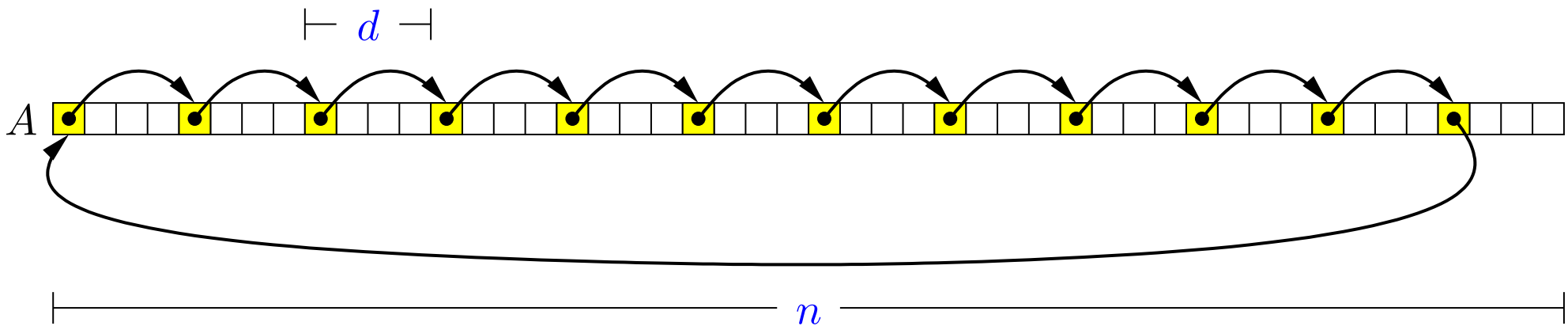
# Hierarchical Memory Basics



- Data moved between adjacent memory levels in blocks

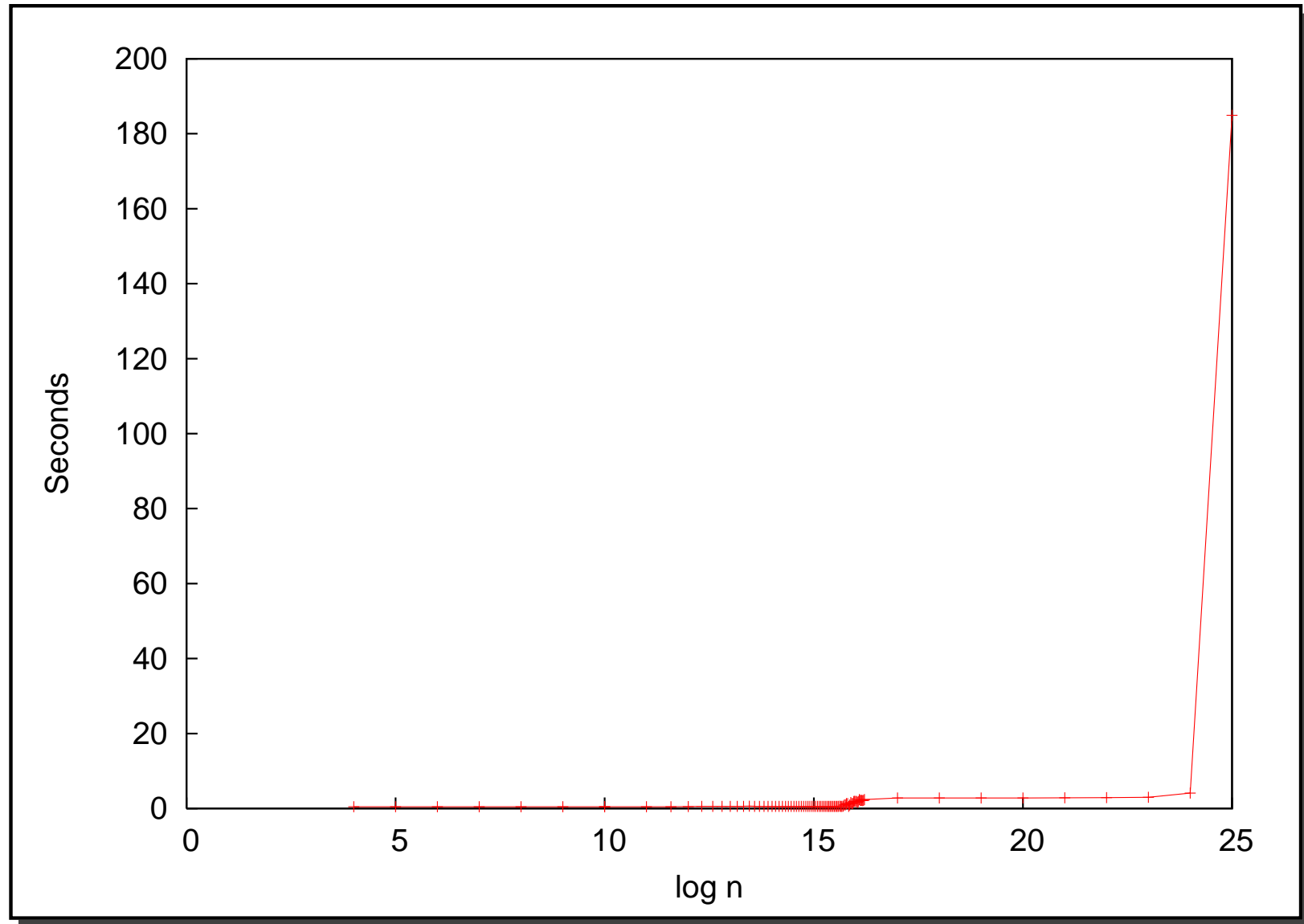
# A Trivial Program

```
for (i=0; i+d<n; i+=d) A[i]=i+d;  
A[i]=0;  
for (i=0, j=0; j<8*1024*1024; j++) i=A[i];
```



# A Trivial Program (cont.)

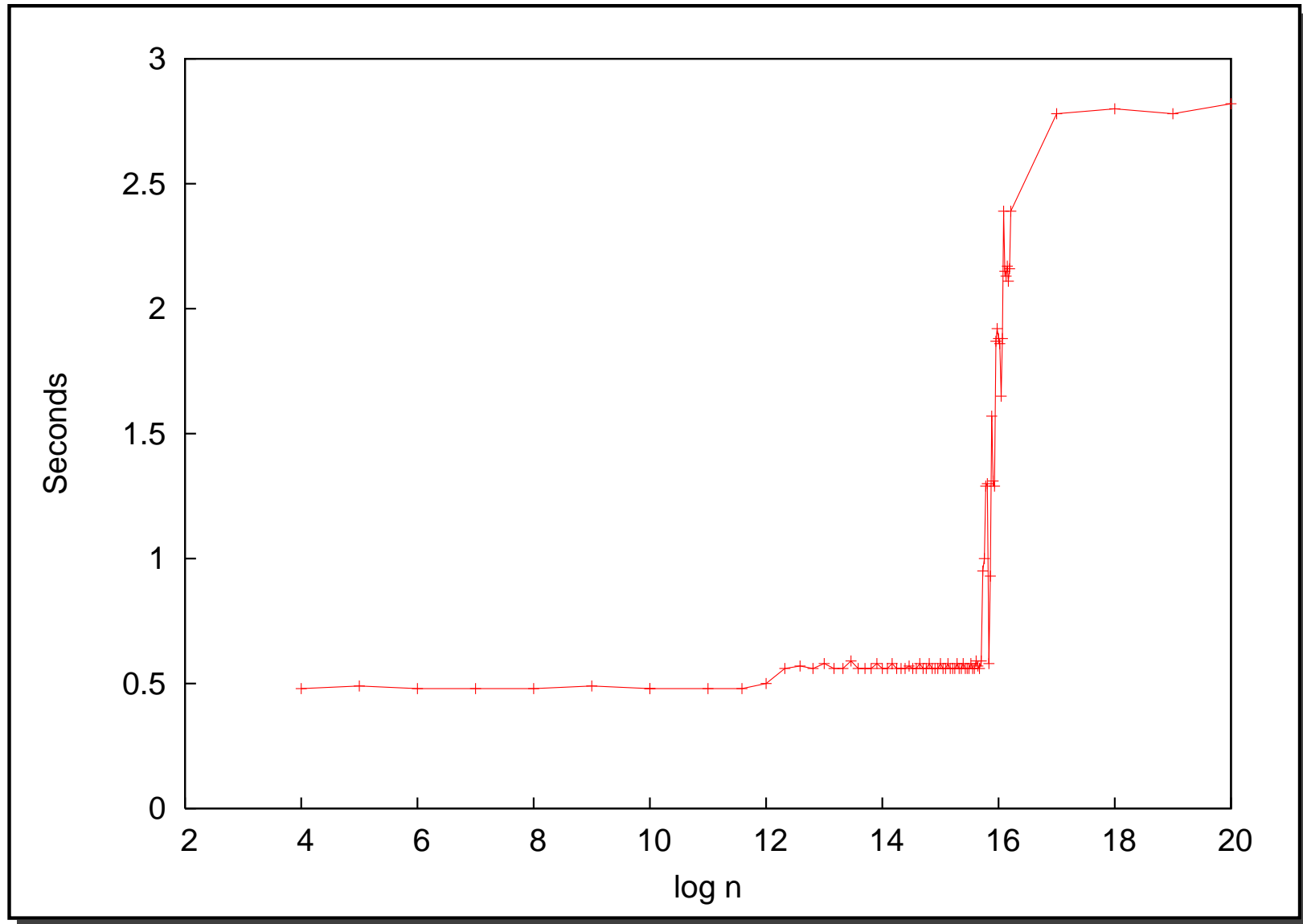
$$d = 1$$



RAM :  $n \approx 2^{25} \equiv 128 \text{ MB}$

# A Trivial Program (cont.)

$d = 1$

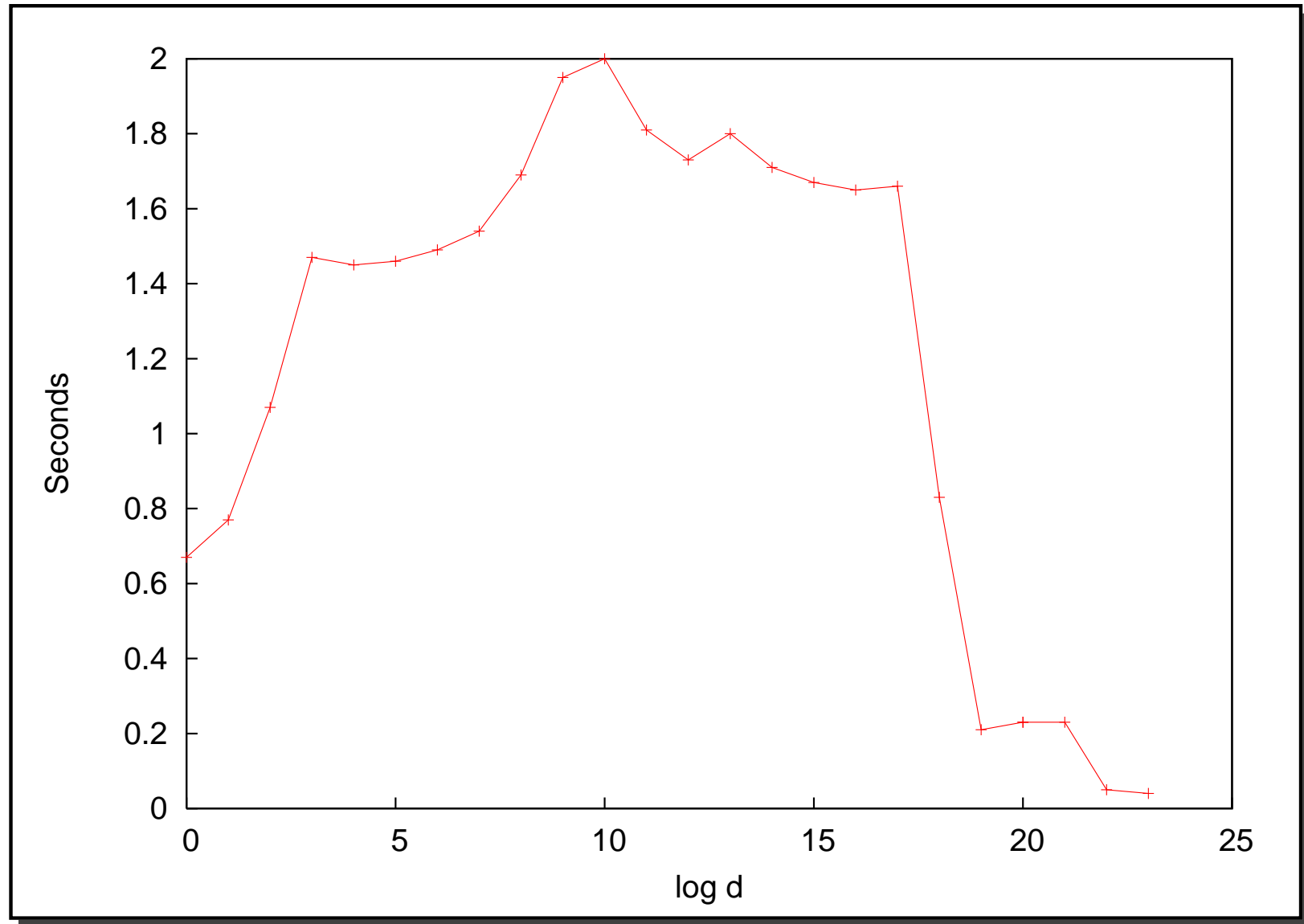


**L1** :  $n \approx 2^{12} \equiv 16 \text{ KB}$

**L2** :  $n \approx 2^{16} \equiv 256 \text{ KB}$

# A Trivial Program (cont.)

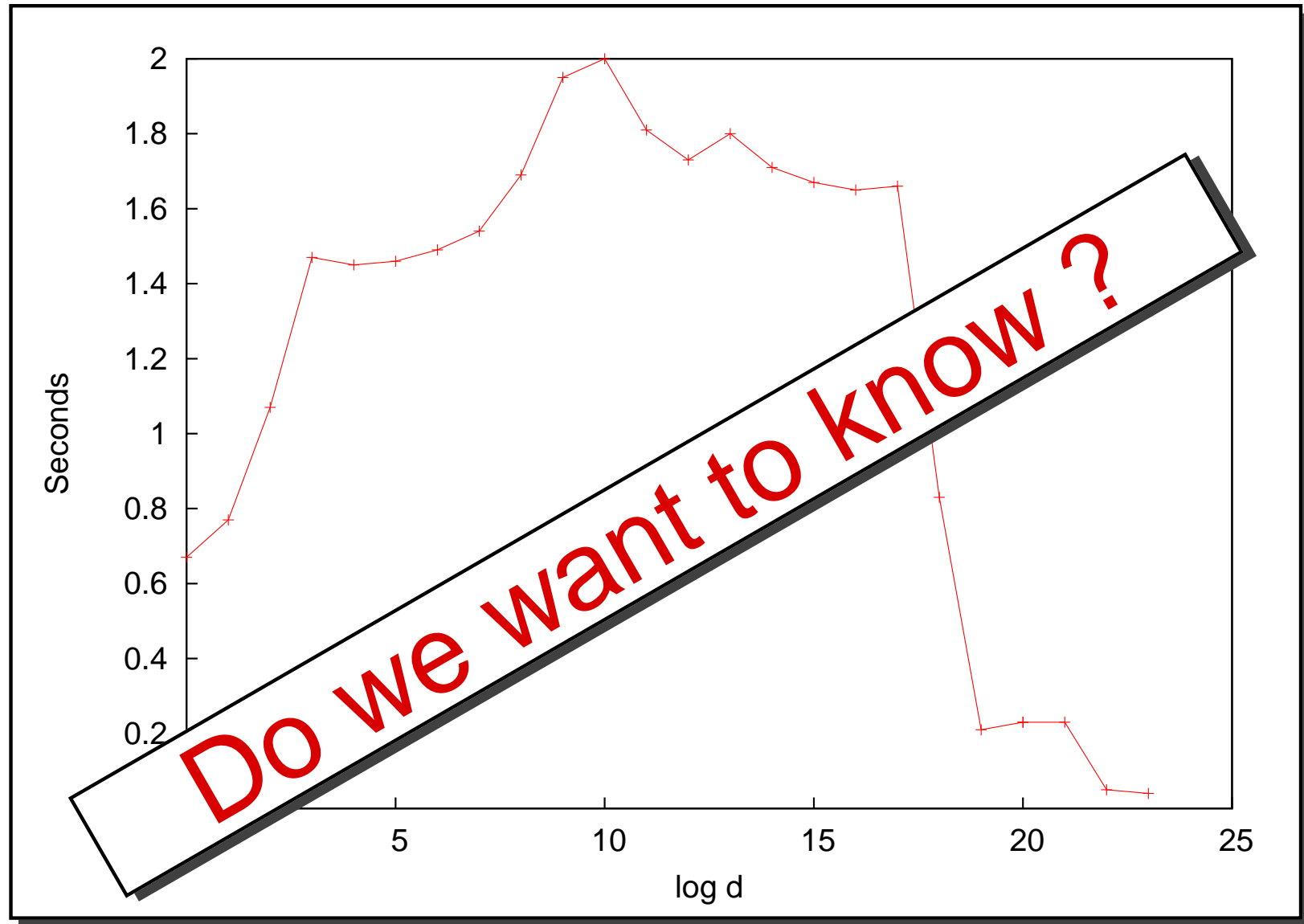
$$n = 2^{24}$$



Cache line  $d = 2^3 \equiv 32$  Bytes

# A Trivial Program (cont.)

$$n = 2^{24}$$



Cache line  $d = 2^3 \equiv 32$  Bytes

# A Trivial Program (cont.)

— If you want to know...

Experiments were performed on a DELL 8000, Pentium III, 850 MHz, 128 MB RAM, running Linux 2.4.2, and using gcc version 2.96 with optimization -O3

L1 instruction and data caches

- 4-way set associative, 32-byte line size
- 16 KB instruction cache and 16 KB write-back data cache

L2 level cache

- 8-way set associative, 32-byte line size
- 256 KB

[www.Intel.com](http://www.Intel.com)

# Algorithmic Problem

- Memory hierarchy has become a fact of life
- Accessing non-local storage may take a very long time
- Good locality is important for achieving high performance

	Latency	Relative to CPU
Register	0.5 ns	1
L1 cache	0.5 ns	1-2
L2 cache	3 ns	2-7
DRAM	150 ns	80-200
TLB	500+ ns	200-2000
Disk	10 ms	$10^7$

**← Increasing**

# Algorithmic Problem

- Modern hardware is not uniform — *many* different parameters
  - Number of memory levels
  - Cache sizes
  - Cache line/disk block sizes
  - Cache associativity
  - Cache replacement strategy
  - CPU/BUS/memory speed

# Algorithmic Problem

- Modern hardware is not uniform — *many* different parameters
  - Number of memory levels
  - Cache sizes
  - Cache line/disk block sizes
  - Cache associativity
  - Cache replacement strategy
  - CPU/BUS/memory speed
- Programs should ideally run for many different parameters

# Algorithmic Problem


- Modern hardware is not uniform — *many* different parameters
  - Number of memory levels
  - Cache sizes
  - Cache line/disk block sizes
  - Cache associativity
  - Cache replacement strategy
  - CPU/BUS/memory speed
- Programs should ideally run for many different parameters
  - by knowing many of the parameters at runtime
  - by knowing few essential parameters
  - ignoring the memory hierarchies

# Algorithmic Problem

- Modern hardware is not uniform — *many* different parameters
  - Number of memory levels
  - Cache sizes
  - Cache line/disk block sizes
  - Cache associativity
  - Cache replacement strategy
  - CPU/BUS/memory speed
- Programs should ideally run for many different parameters
  - by knowing many of the parameters at runtime
  - by knowing few essential parameters
  - ignoring the memory hierarchies



# Algorithmic Problem

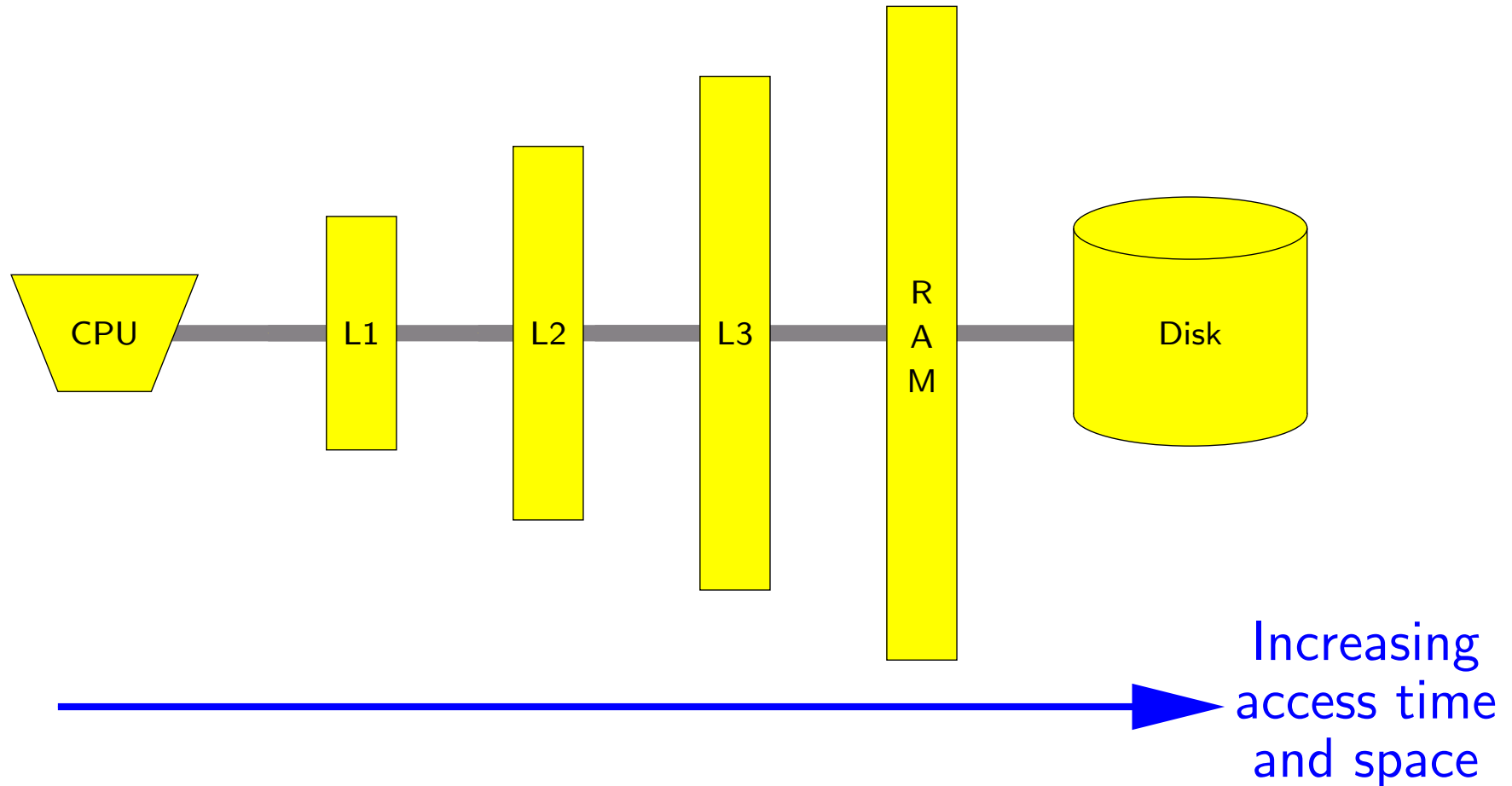
- Modern hardware is not uniform — *many* different parameters
  - Number of memory levels
  - Cache sizes
  - Cache line/disk block sizes
  - Cache associativity
  - Cache replacement strategy
  - CPU/BUS/memory speed
- Programs should ideally run for many different parameters
  - by knowing many of the parameters at runtime
  - by knowing few essential parameters
  - ignoring the memory hierarchies  **practice**
- Programs are executed on unpredictable configurations
  - Generic portable and scalable software libraries
  - Code downloaded from the Internet, e.g. Java applets
  - Dynamic environments, e.g. multiple processes

# Outline

- Motivation
  - A typical workstation
  - A trivial program
- ▶ Memory models
  - I/O model
  - **Ideal cache model**
- Basic cache-oblivious algorithms
  - Matrix multiplication
  - Search trees
  - Sorting
- Some experimental results
- Conclusion

# Hierarchical Memory Models

— many parameters

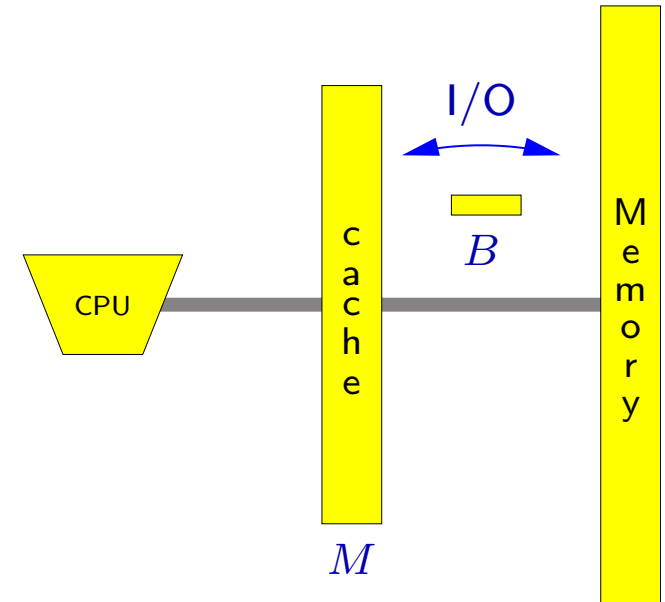


- Limited success since model **to complicated**

# I/O Model — two parameters

Aggarwal and Vitter 1988

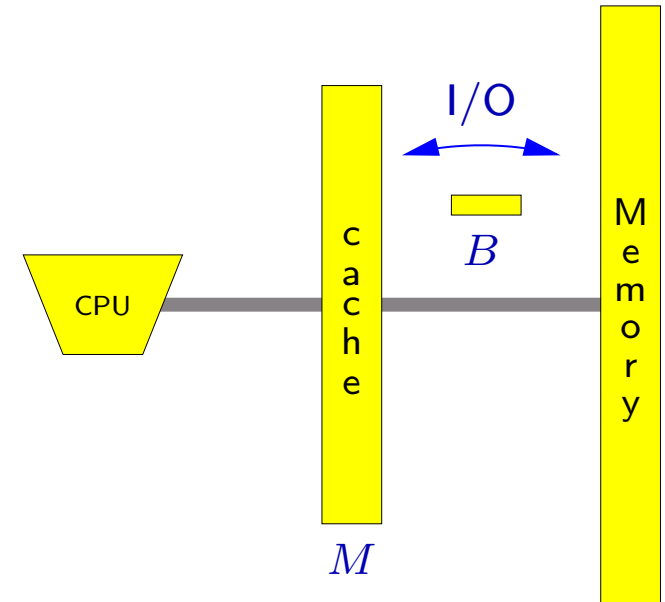
- Measure number of block transfers between two memory levels
- Bottleneck in many computations
- Very successful (simplicity)



# I/O Model — two parameters

Aggarwal and Vitter 1988

- Measure number of block transfers between two memory levels
- Bottleneck in many computations
- Very successful (simplicity)



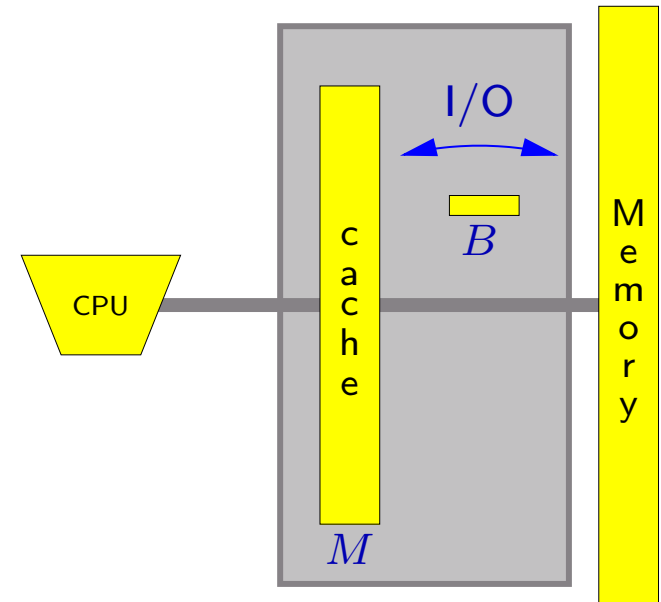
## Limitations

- Parameters  $B$  and  $M$  must be known
- Does not handle multiple memory levels
- Does not handle dynamic  $M$

# Ideal Cache Model — no parameters!?

Frigo, Leiserson, Prokop, Ramachandran 1999

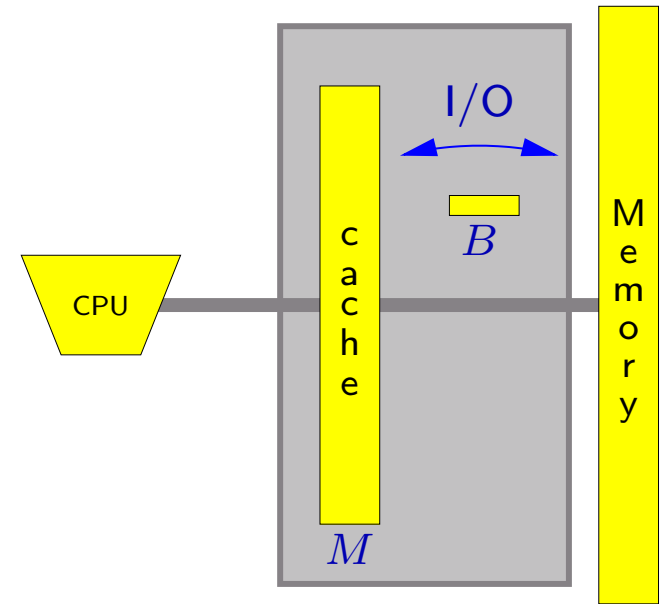
- Program with only one memory
- Analyze in the I/O model for
- Optimal off-line cache replacement strategy arbitrary  $B$  and  $M$



# Ideal Cache Model — no parameters!?

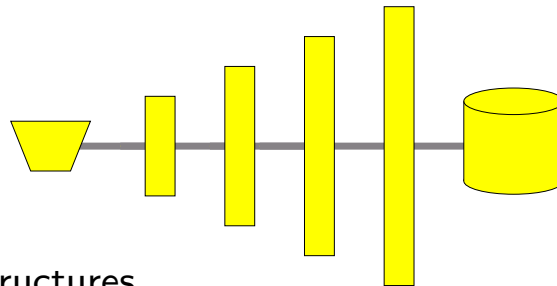
Frigo, Leiserson, Prokop, Ramachandran 1999

- Program with only one memory
- Analyze in the I/O model for
- Optimal off-line cache replacement strategy arbitrary  $B$  and  $M$



## Advantages

- Optimal on arbitrary level  $\Rightarrow$  optimal on **all levels**
- Portability,  $B$  and  $M$  not hard-wired into algorithm
- Dynamic changing parameters



# Justification of the Ideal-Cache Model

Frigo, Leiserson, Prokop, Ramachandran 1999

**Optimal replacement** LRU +  $2 \times$  cache size  $\Rightarrow$  at most  $2 \times$  cache misses

Sleator and 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** LRU cache

## 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

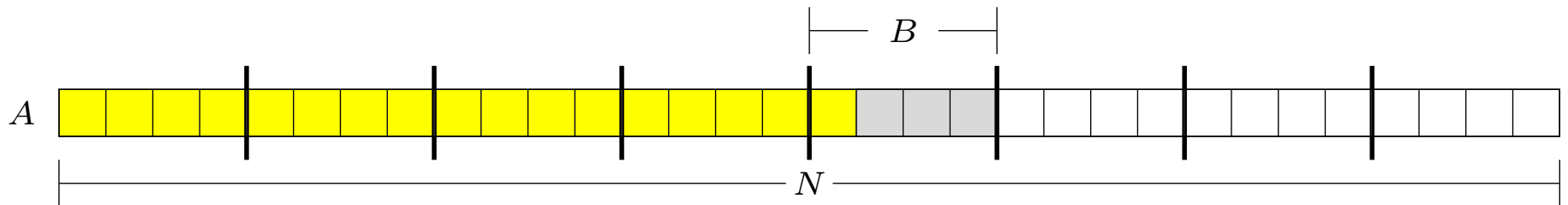
# Outline

- Motivation
  - A typical workstation
  - A trivial program
- Memory models
  - I/O model
  - **Ideal cache model**
- ▶ Basic cache-oblivious algorithms
  - Matrix multiplication
  - Search trees
  - Sorting
- Some experimental results
- Conclusion

# Warm-up : Scanning

```
sum = 0  
for i = 1 to N do sum = sum + A[i]
```

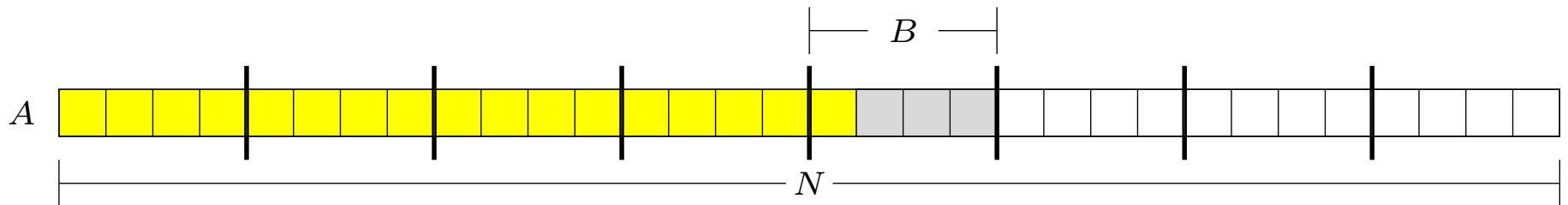
$O\left(\frac{N}{B}\right)$  I/Os



# Warm-up : Scanning

```
sum = 0  
for i = 1 to N do sum = sum + A[i]
```

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



**Corollary** Cache-oblivious selection requires  $O(N/B)$  I/Os

Hoare 1961 / Blum et al. 1973

# Cache-Oblivious Matrix Multiplication

# Matrix Multiplication

## Problem

$$Z = X \cdot Y, \quad z_{ij} = \sum_{k=1}^N x_{ik} \cdot y_{kj}$$

## Layout of matrices

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Row major

0	8	16	24	32	40	48	56
1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63

Column major

0	1	2	3	16	17	18	19
4	5	6	7	20	21	22	23
8	9	10	11	24	25	26	27
12	13	14	15	28	29	30	31
32	33	34	35	48	49	50	51
36	37	38	39	52	53	54	55
40	41	42	43	56	57	58	59
44	45	46	47	60	61	62	63

4 × 4-blocked

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

Bit interleaved

# Matrix Multiplication

## Algorithm 1: Nested loops

- Row major
- Reading a **column of  $Y$**   $\Rightarrow N$  I/Os
- Total  $O(N^3)$  I/Os

```
for  $i = 1$  to  $N$ 
  for  $j = 1$  to  $N$ 
     $z_{ij} = 0$ 
    for  $k = 1$  to  $N$ 
       $z_{ij} = z_{ij} + x_{ik} \cdot y_{kj}$ 
```

# Matrix Multiplication

## Algorithm 1: Nested loops

- Row major
- Reading a **column of  $Y$**   $\Rightarrow N$  I/Os
- Total  $O(N^3)$  I/Os

```
for  $i = 1$  to  $N$ 
  for  $j = 1$  to  $N$ 
     $z_{ij} = 0$ 
    for  $k = 1$  to  $N$ 
       $z_{ij} = z_{ij} + x_{ik} \cdot y_{kj}$ 
```

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

- Partition  $X$  and  $Y$  into blocks of size  $s \times s$ ,

$$s = \Theta(\sqrt{M})$$

- Apply Algorithm 1 to the  $\frac{N}{s} \times \frac{N}{s}$  matrices where elements are  $s \times s$  matrices

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

# Matrix Multiplication

## Algorithm 1: Nested loops

- Row major
- Reading a **column of  $Y$**   $\Rightarrow N$  I/Os
- Total  $O(N^3)$  I/Os

```
for  $i = 1$  to  $N$ 
  for  $j = 1$  to  $N$ 
     $z_{ij} = 0$ 
    for  $k = 1$  to  $N$ 
       $z_{ij} = z_{ij} + x_{ik} \cdot y_{kj}$ 
```

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

- Partition  $X$  and  $Y$  into blocks of size  $s \times s$ ,

$$s = \Theta(\sqrt{M})$$

- 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 (row major and  $M = \Omega(B^2)$ )

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

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

# Matrix Multiplication

## Algorithm 1: Nested loops

- Row major
- Reading a **column of  $Y$**   $\Rightarrow N$  I/Os
- Total  $O(N^3)$  I/Os

```
for  $i = 1$  to  $N$ 
  for  $j = 1$  to  $N$ 
     $z_{ij} = 0$ 
    for  $k = 1$  to  $N$ 
       $z_{ij} = z_{ij} + x_{ik} \cdot y_{kj}$ 
```

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

- Partition  $X$  and  $Y$  into blocks of size  $s \times s$ ,

$$s = \Theta(\sqrt{M})$$

- 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 (row major and  $M = \Omega(B^2)$ )

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

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

- Optimal

Hong & Kung, 1981

# Matrix Multiplication

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

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{pmatrix}$$

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

# Matrix Multiplication

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

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{pmatrix}$$

- 8 recursive  $\frac{N}{2} \times \frac{N}{2}$  multiplications + 4  $\frac{N}{2} \times \frac{N}{2}$  matrix sums
- # I/Os if bit interleaved or ( row major and  $M = \Omega(B^2)$  )

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

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

# Matrix Multiplication

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

$$\begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} \begin{pmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{pmatrix} = \begin{pmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{pmatrix}$$

- 8 recursive  $\frac{N}{2} \times \frac{N}{2}$  multiplications + 4  $\frac{N}{2} \times \frac{N}{2}$  matrix sums
- # I/Os if bit interleaved or ( row major and  $M = \Omega(B^2)$  )

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

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

- Optimal
- Non-square matrices

Hong & Kung, 1981

Frigo et al., 1999

# Cache-Oblivious Matrix Multiplication

I/O bound

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

## Techniques applied

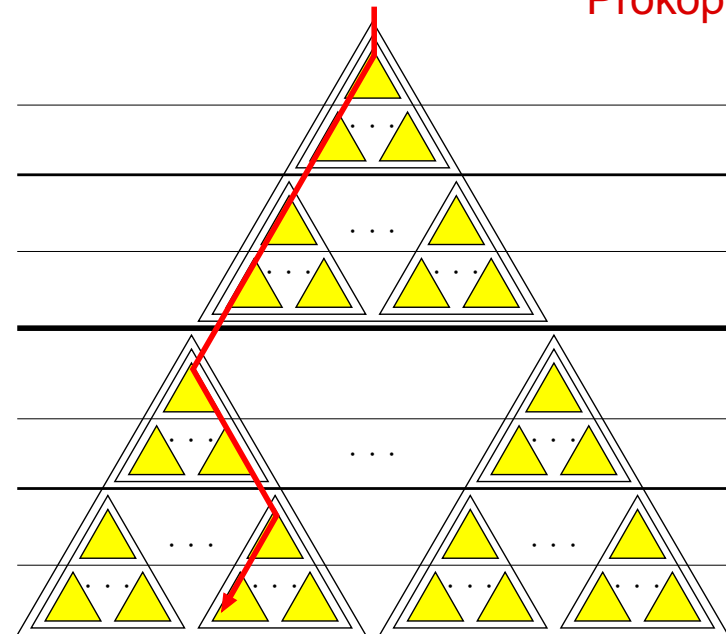
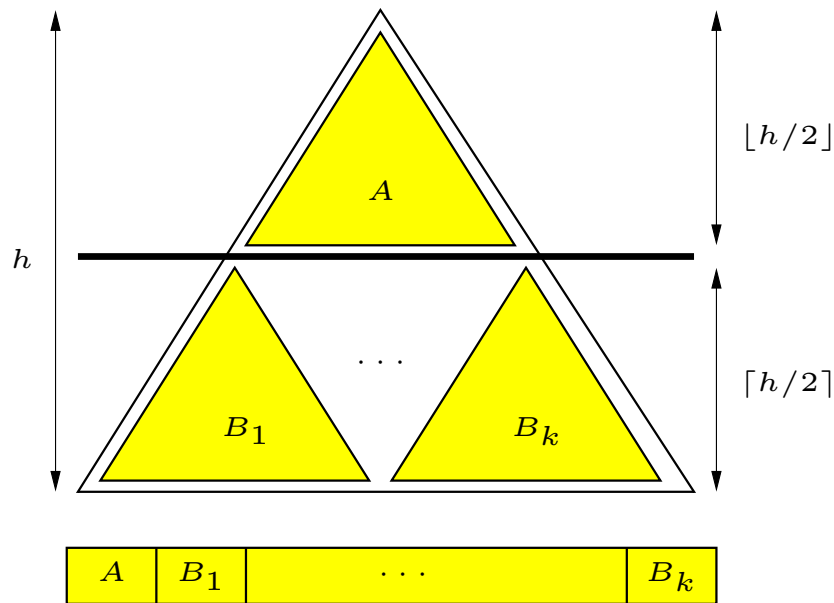
- Recursion / divide-and-conquer
- Recursive memory layout
- Scanning

# Cache-Oblivious Search Trees

# Static Search Trees

Recursive memory layout  $\equiv$  van Emde Boas layout

Prokop 1999

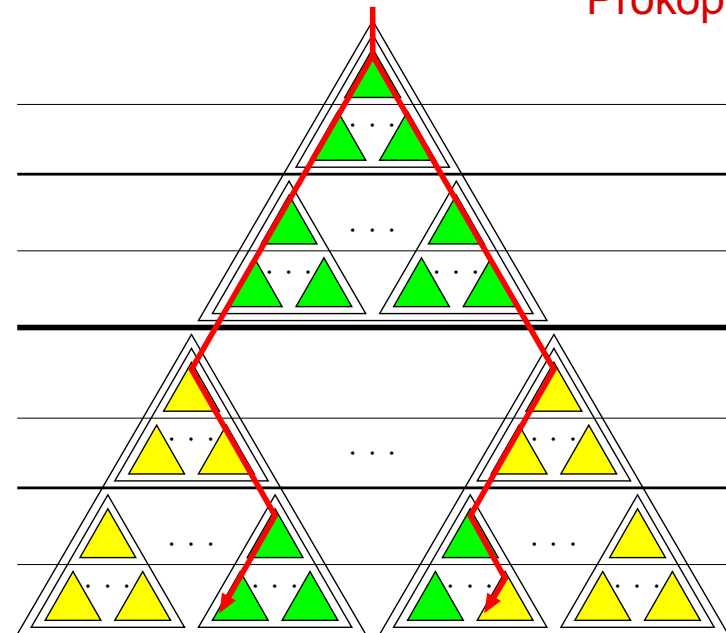
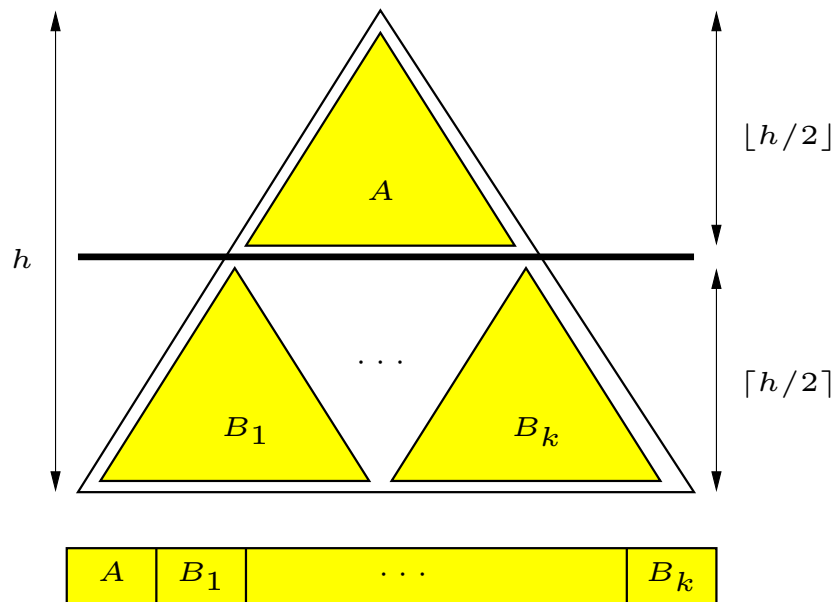


Searches use  $O(\log_B N)$  I/Os

# Static Search Trees

Recursive memory layout  $\equiv$  van Emde Boas layout

Prokop 1999



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

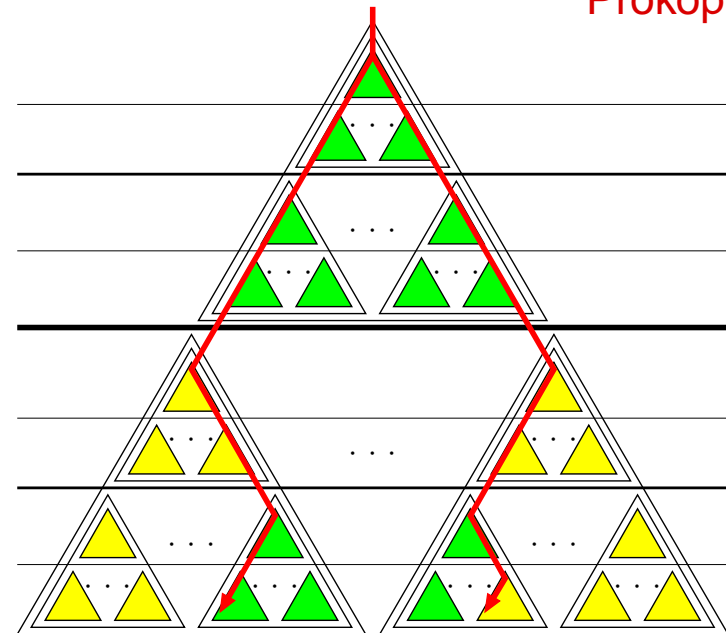
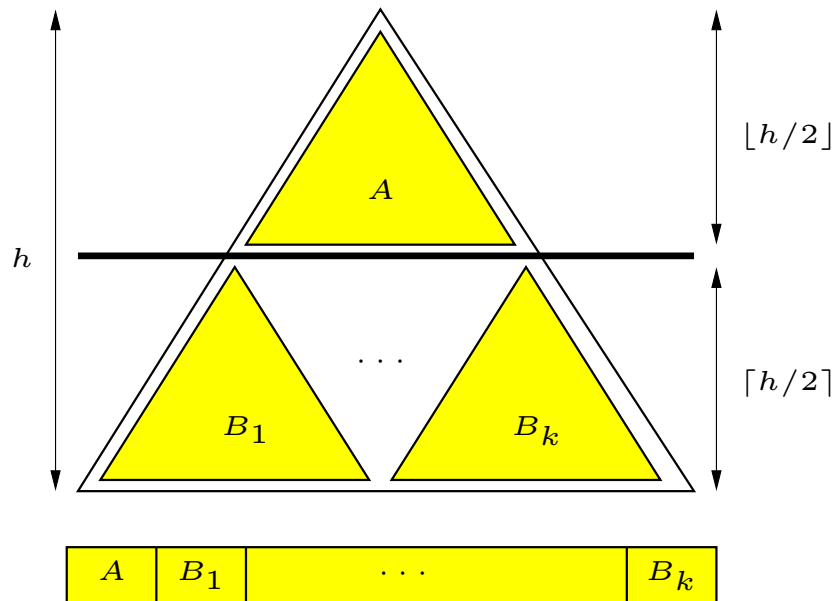
Range reportings use

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

# Static Search Trees

Recursive memory layout  $\equiv$  van Emde Boas layout

Prokop 1999



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

Range reportings use

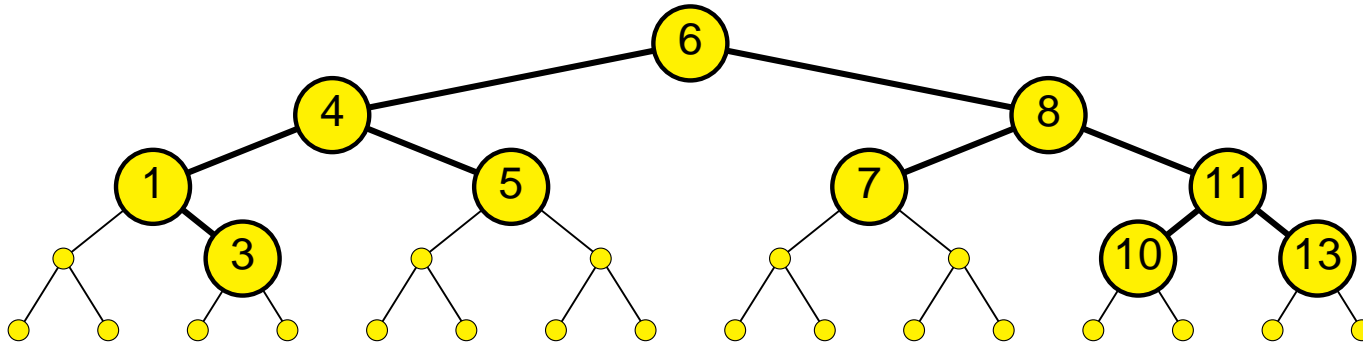
$O\left(\log_B N + \frac{K}{B}\right)$  I/Os

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

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

# Dynamic Search 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

$$O(\log_B N)$$

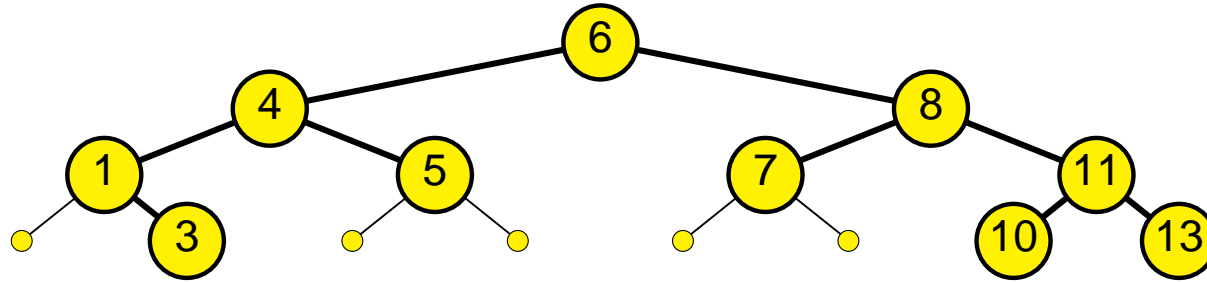
Range Reporting

$$O\left(\log_B N + \frac{K}{B}\right)$$

Updates

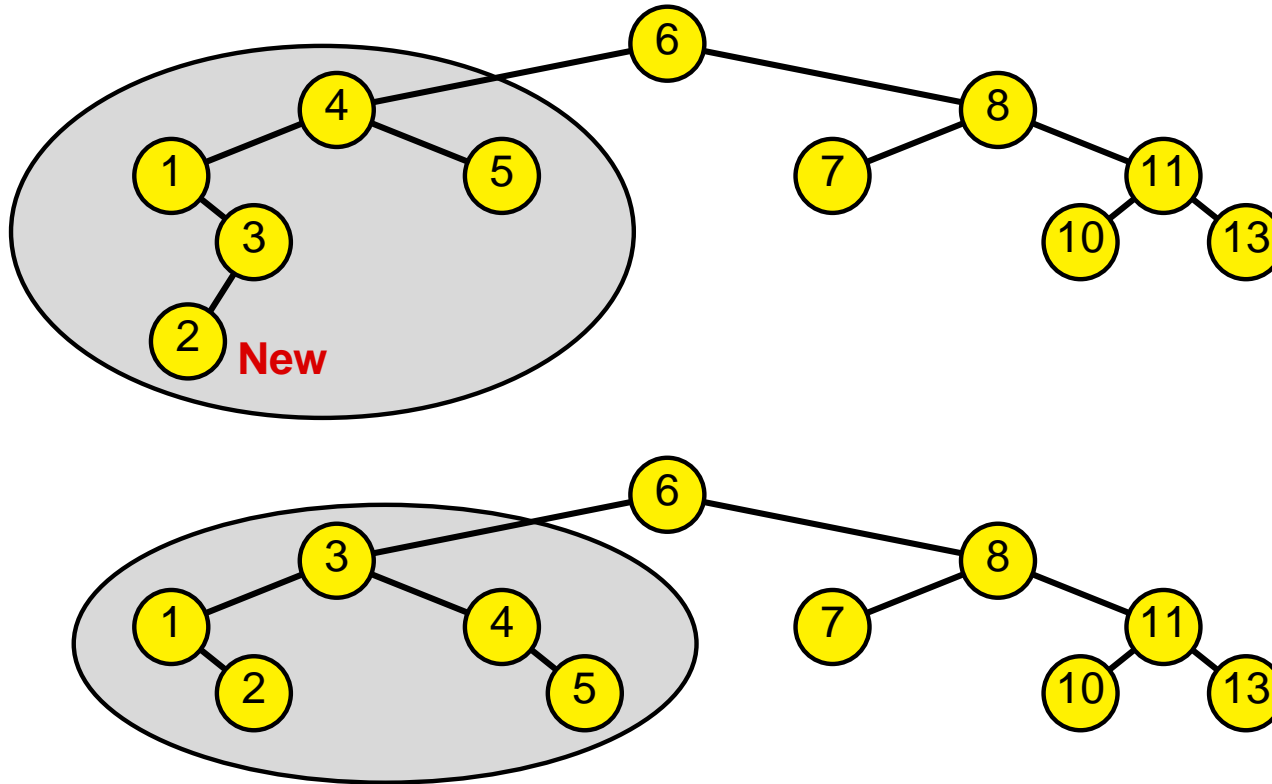
$$O\left(\log_B N + \frac{\log^2 N}{B}\right)$$

# Example



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

Andersson and Lai 1990

# Cache-Oblivious Search Trees

## I/O bounds

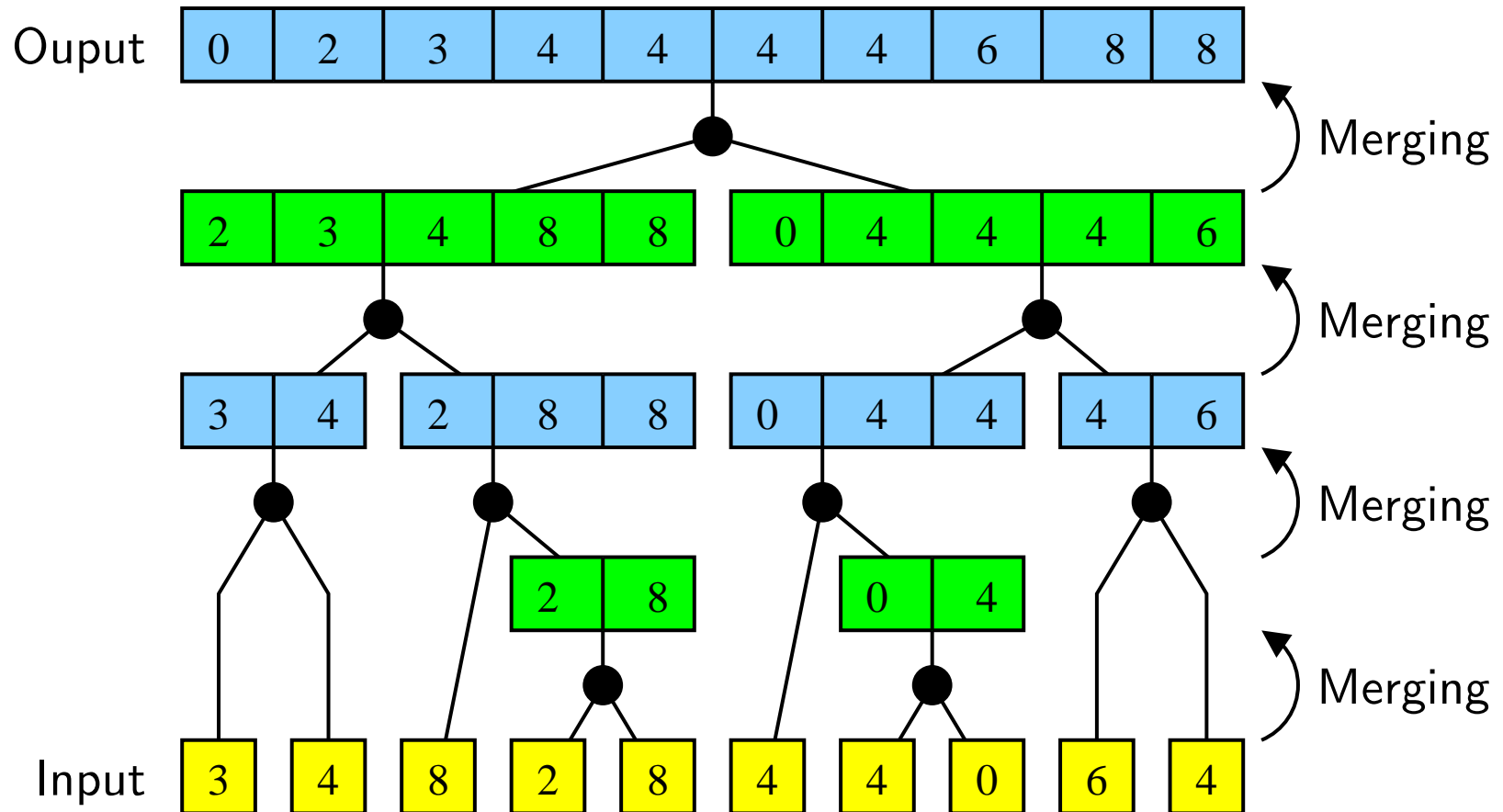
Search	$O(\log_B N)$	
Range Reporting	$O\left(\log_B N + \frac{K}{B}\right)$	
Updates	$\begin{cases} O\left(\log_B N + \frac{\log^2 N}{B}\right) \\ O(\log_B N) \end{cases}$	(no range reporting)

## Techniques applied

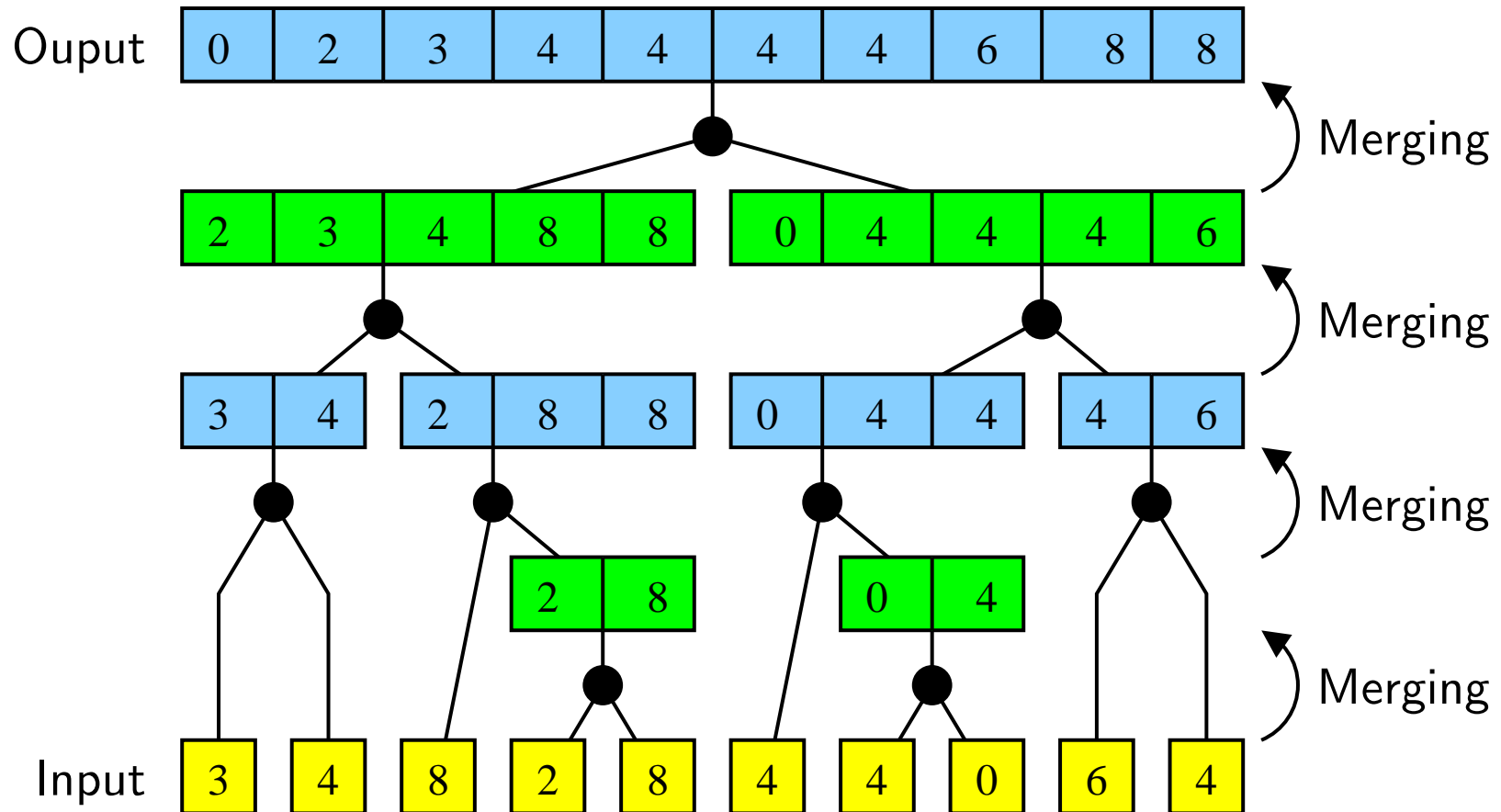
- Recursion
- Recursive memory layout
- Scanning (updates, range reporting)

# Cache-Oblivious Sorting

# 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	$O\left(\frac{N}{B} \log_2 \frac{N}{M}\right)$
$d$ $(d \leq \frac{M}{B} - 1)$	$O\left(\frac{N}{B} \log_d \frac{N}{M}\right)$
$\Theta\left(\frac{M}{B}\right)$	$O\left(\frac{N}{B} \log_{M/B} \frac{N}{M}\right) = O(\text{Sort}_{M,B}(N))$ Aggarwal and Vitter 1988

## Funnel-Sort

2 $(M \geq B^{1+\varepsilon})$	$O\left(\frac{1}{\varepsilon} \text{Sort}_{M,B}(N)\right)$ 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	$B_2$	$M$	$t_2$

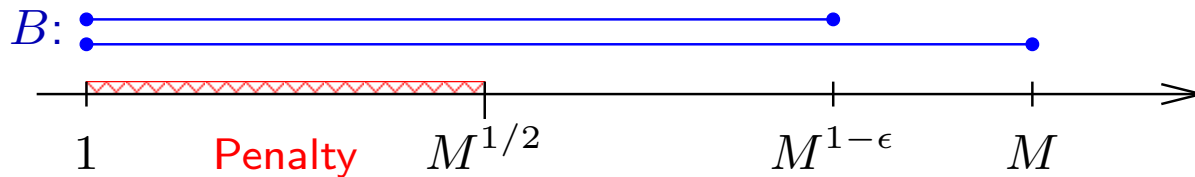
One algorithm, two machines,  $B_1 \leq B_2$

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	$B \leq M^{1-\epsilon}$	(a) $B_2 = M^{1-\epsilon} : \text{Sort}_{B_2, M}(N)$ (b) $B_1 = 1 : \text{Sort}_{B_1, M}(N) \cdot \frac{1}{\epsilon}$
Binary Merge-sort	$B \leq M/2$	(a) $B_2 = M/2 : \text{Sort}_{B_2, M}(N)$ (b) $B_1 = 1 : \text{Sort}_{B_1, M}(N) \cdot \log M$



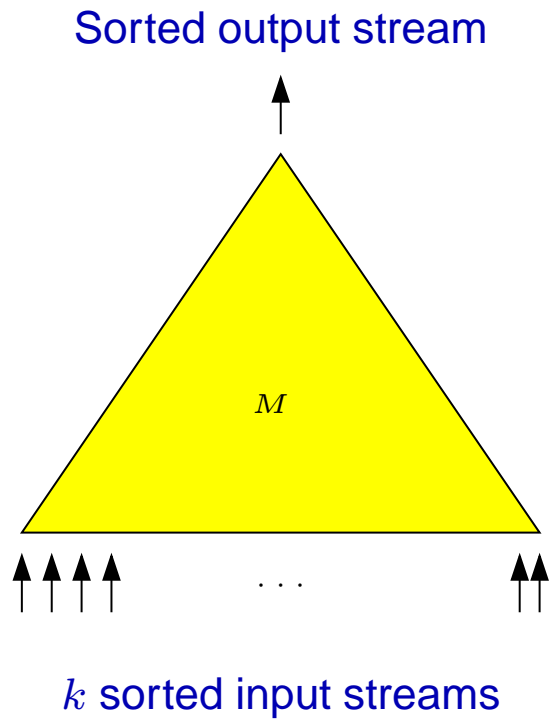
**Theorem** This is tight. For any cache-oblivious comparison based sorting algorithm: (a)  $\Rightarrow$  (b)

# Funnel-Sort



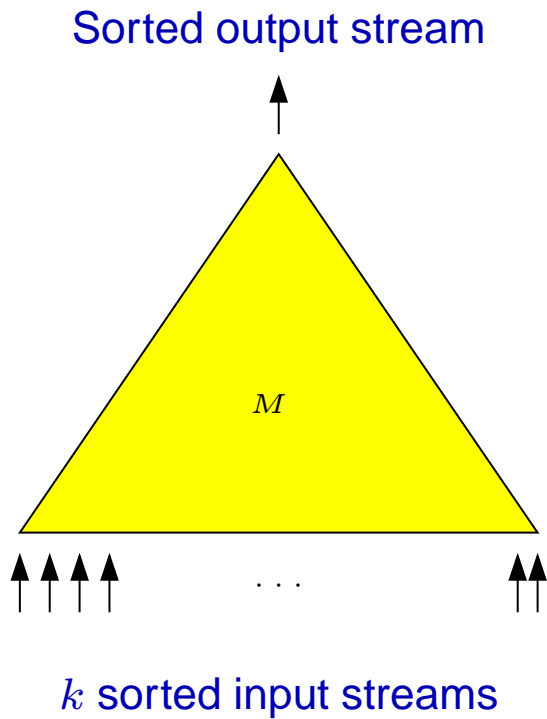
# $k$ -merger

Frigo et al., FOCS'99



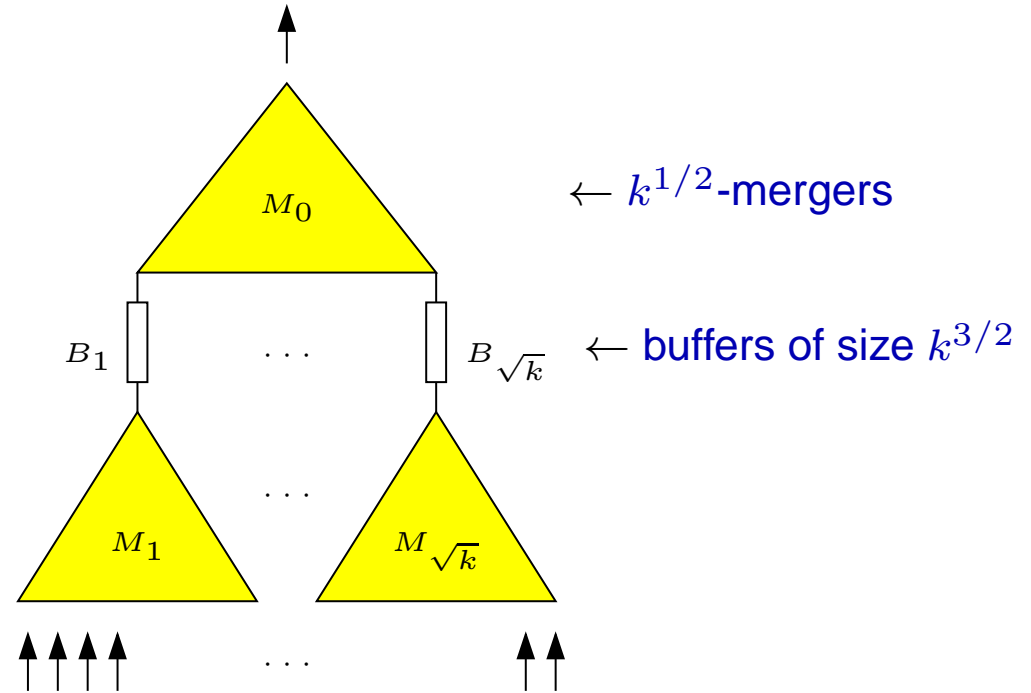
# $k$ -merger

Frigo et al., FOCS'99



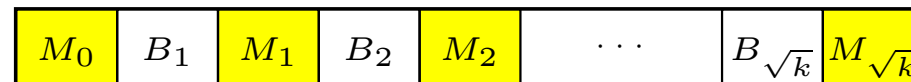
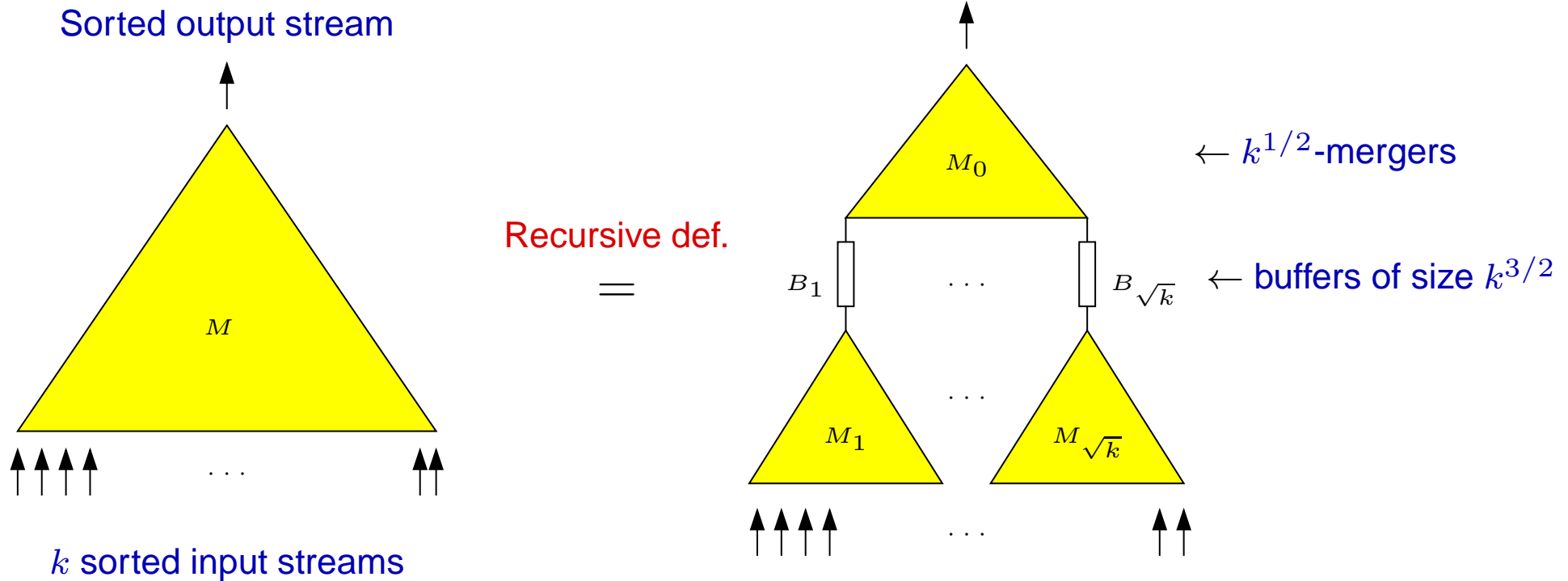
Recursive def.

=



# $k$ -merger

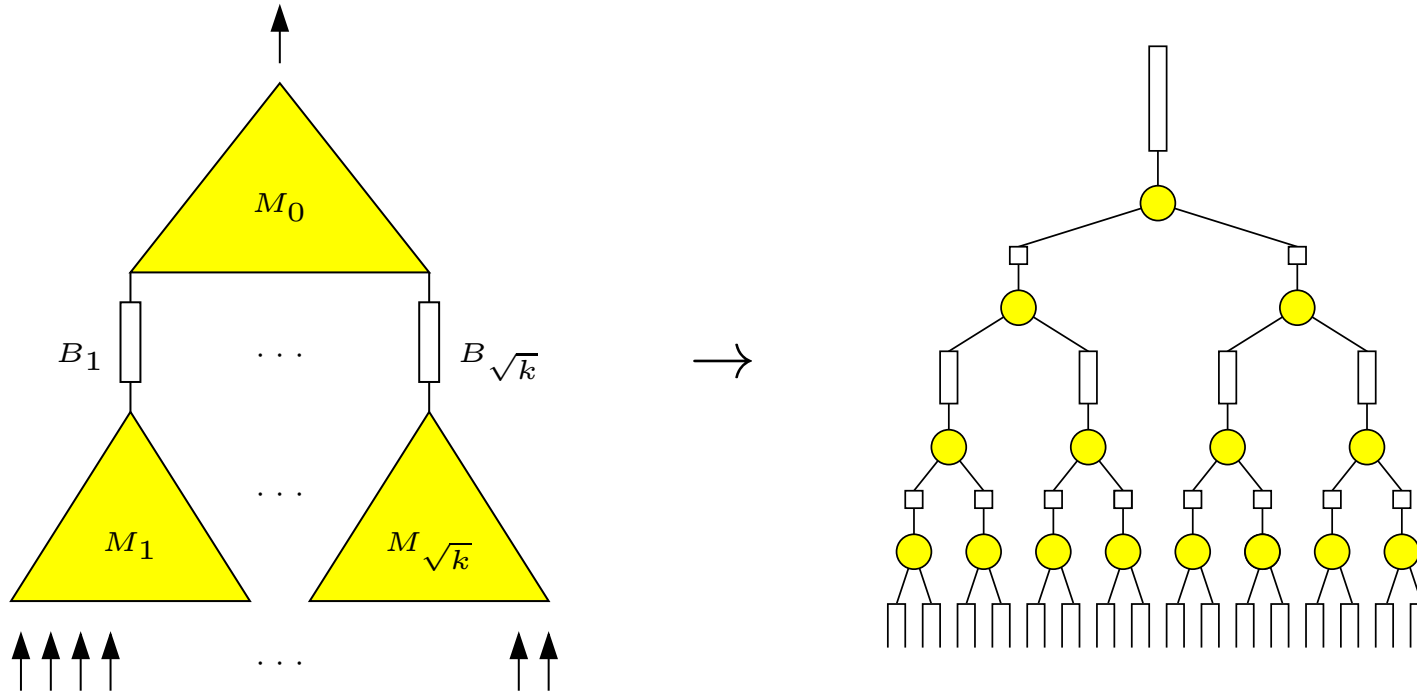
Frigo et al., FOCS'99



Recursive Layout

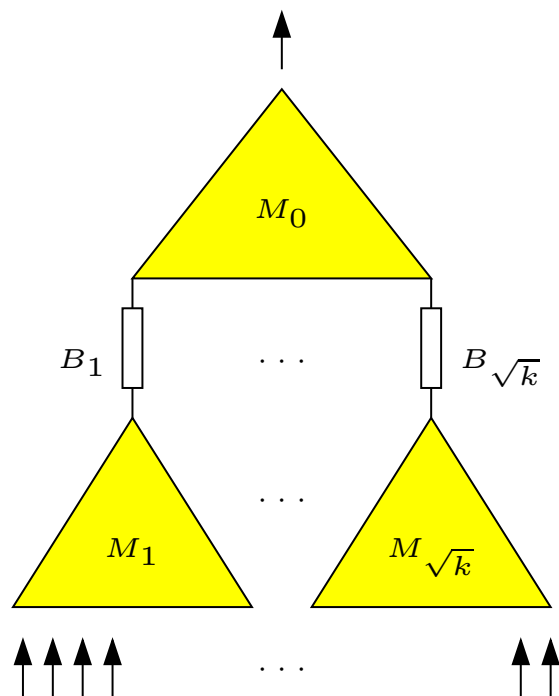
# Lazy $k$ -merger

Brodal and Fagerberg 2002

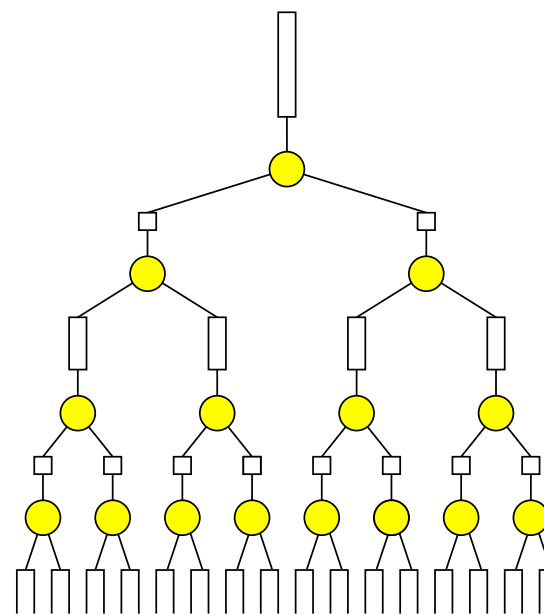


# Lazy $k$ -merger

Brodal and Fagerberg 2002



→



Procedure **Fill**( $v$ )

**while** out-buffer not full

**if** left in-buffer empty

**Fill**(left child)

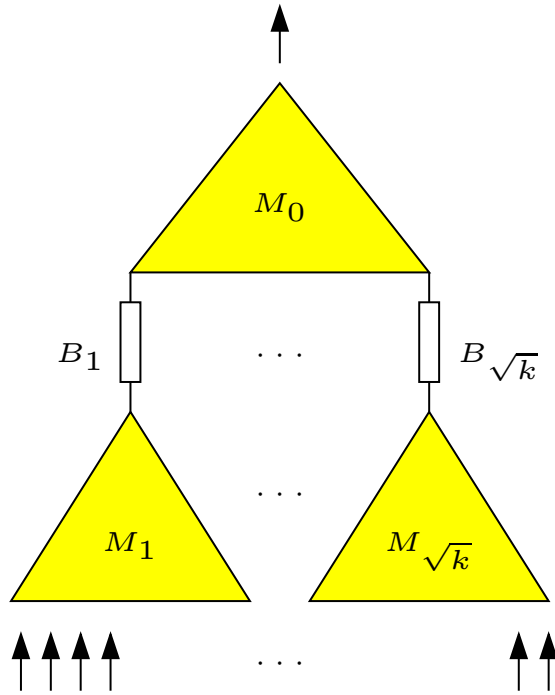
**if** right in-buffer empty

**Fill**(right child)

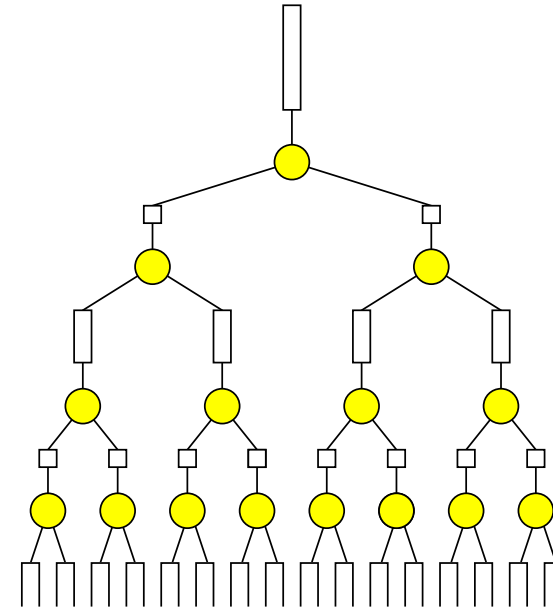
  perform one merge step

# 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

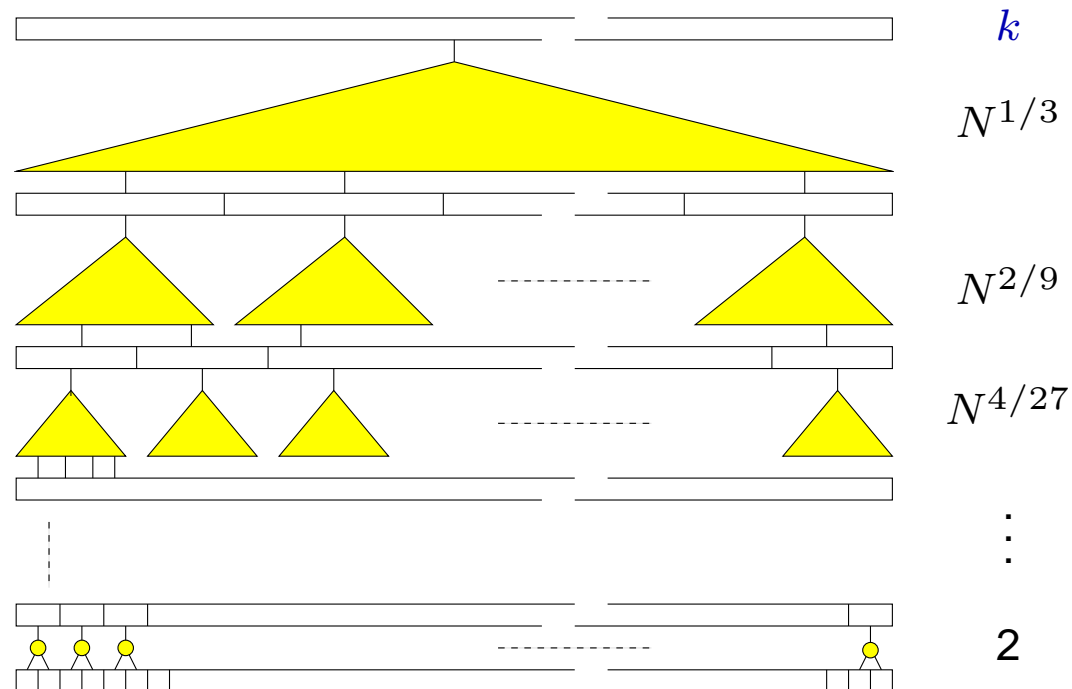
Brodal and Fagerberg 2002

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





# Cache-Oblivious Sorting

## I/O bounds

$$O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right) \text{ assuming a tall cache } M = \Omega\left(B^{1+\varepsilon}\right)$$

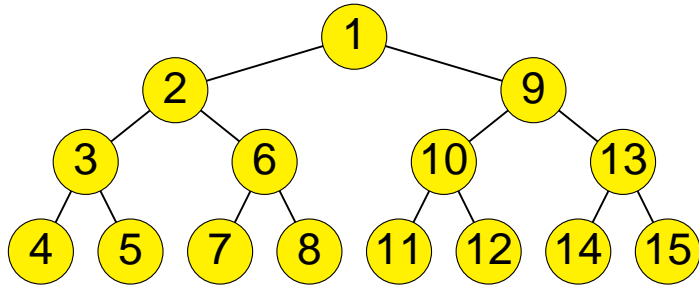
## Techniques applied

- Divide-and-conquer
- Recursive memory layout ( $k$ -merger)
- Scanning (buffers)
- Buffers of double-exponential size

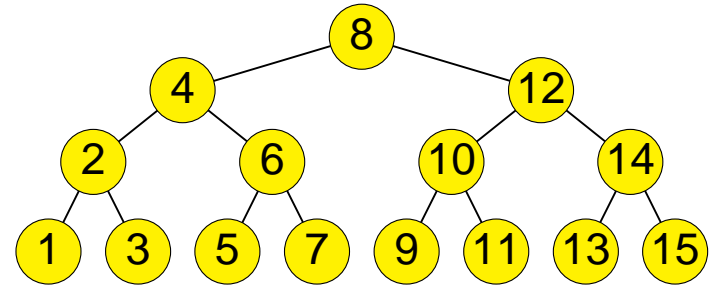
# Outline

- Motivation
  - A typical workstation
  - A trivial program
- Memory models
  - I/O model
  - **Ideal cache model**
- Basic cache-oblivious algorithms
  - Matrix multiplication
  - Search trees
  - Sorting
- ▶ Some experimental results
- Conclusion

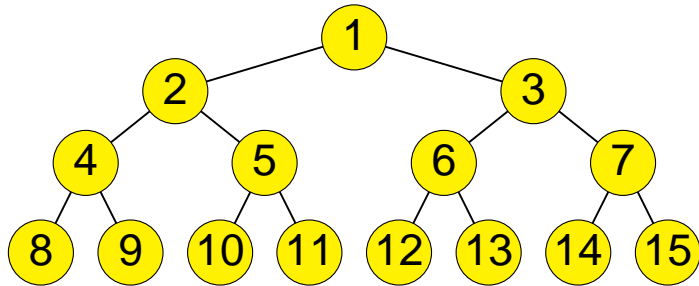
# Memory Layouts of Trees



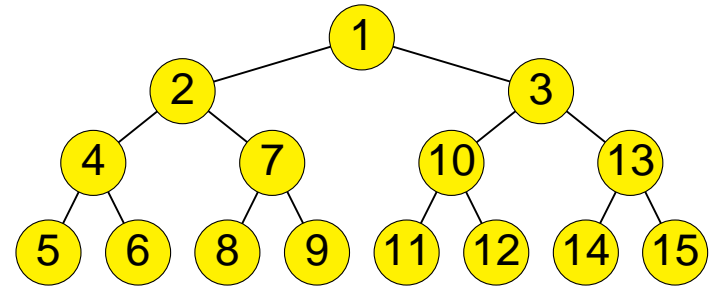
DFS



inorder



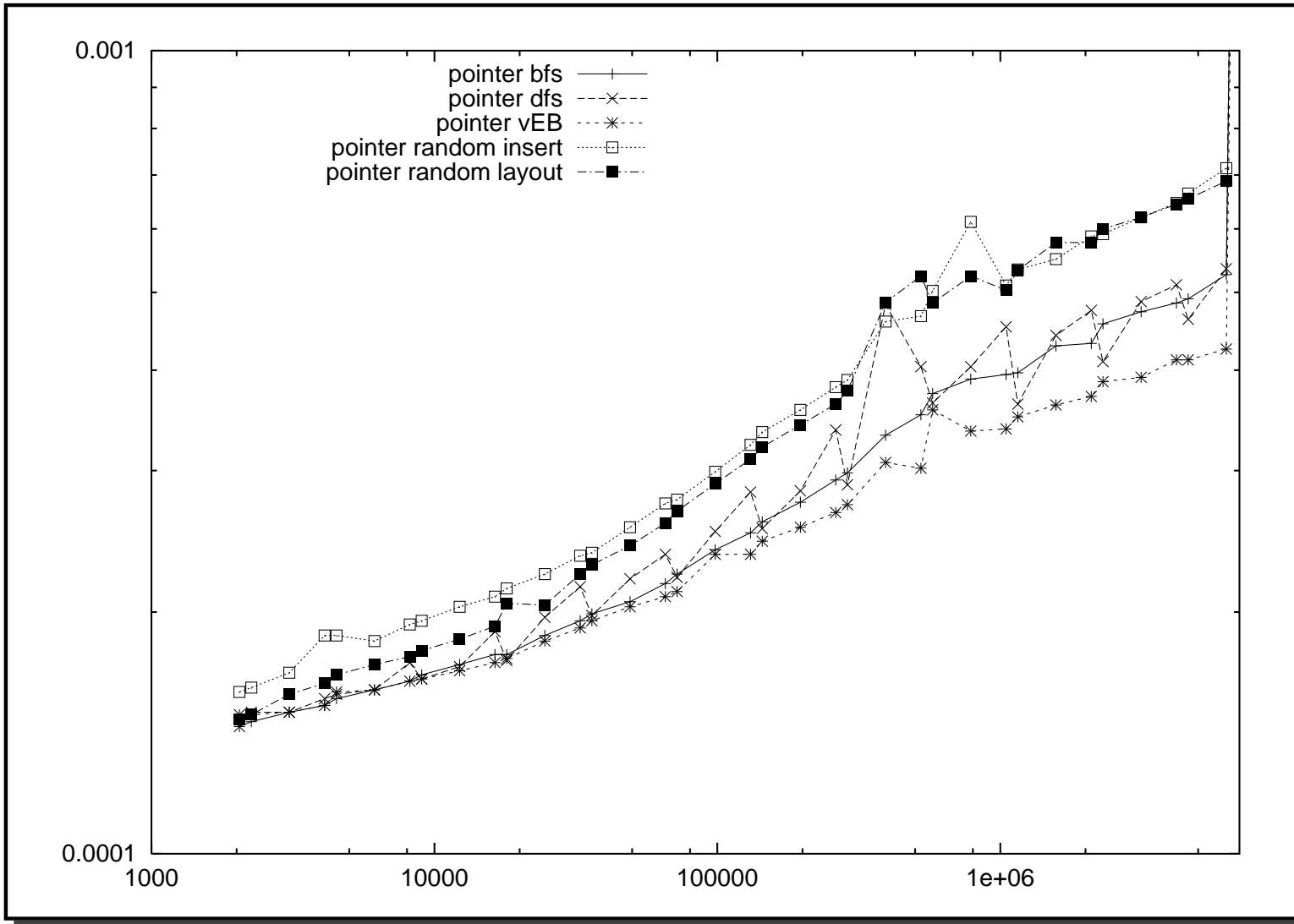
BFS



van Emde Boas  
(in theory best)

# Searches in Pointer Based Layouts

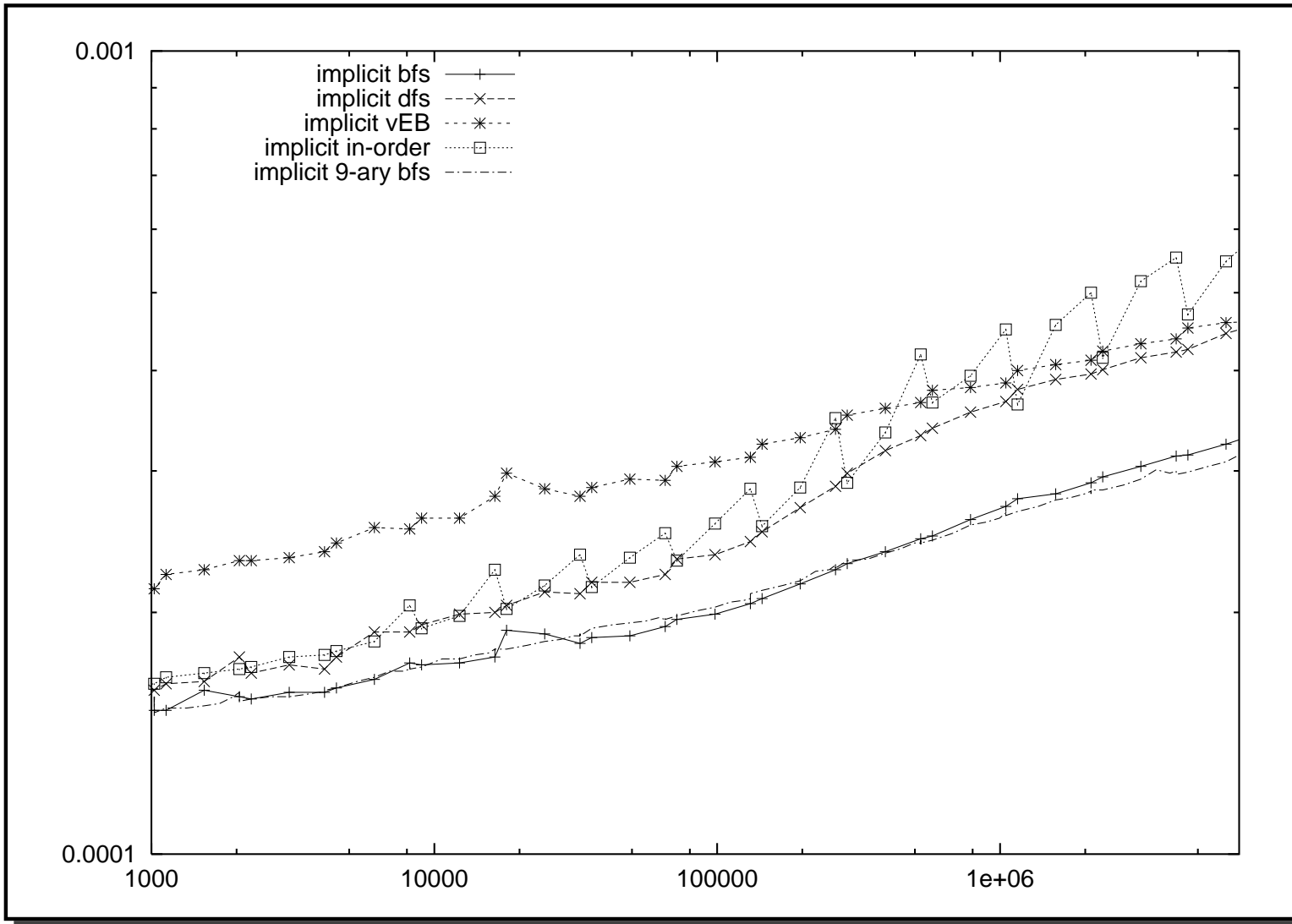
Brodal, Fagerberg, Jacob 2002



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

# Searches with Implicit Layouts

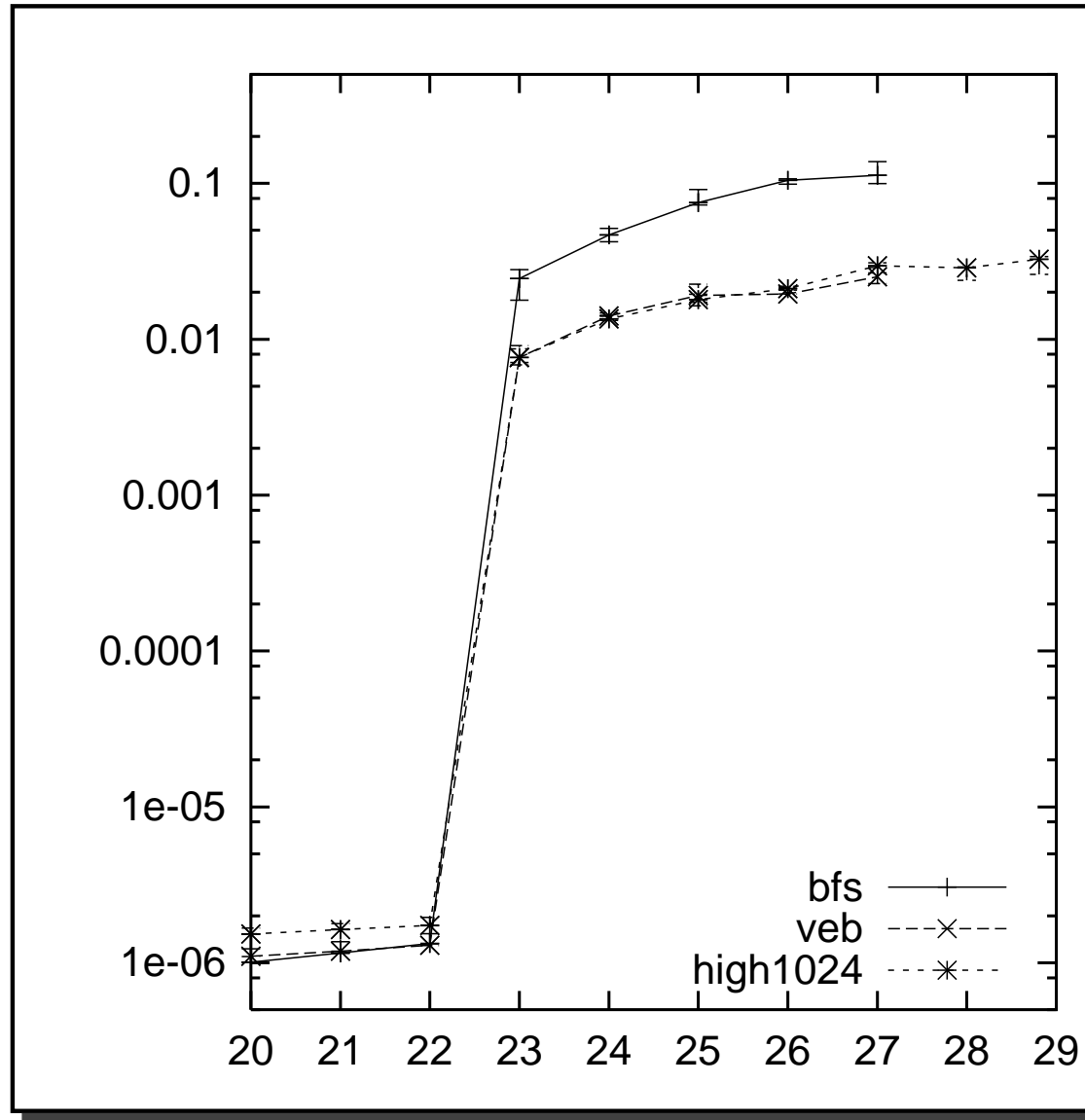
Brodal, Fagerberg, Jacob 2002



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

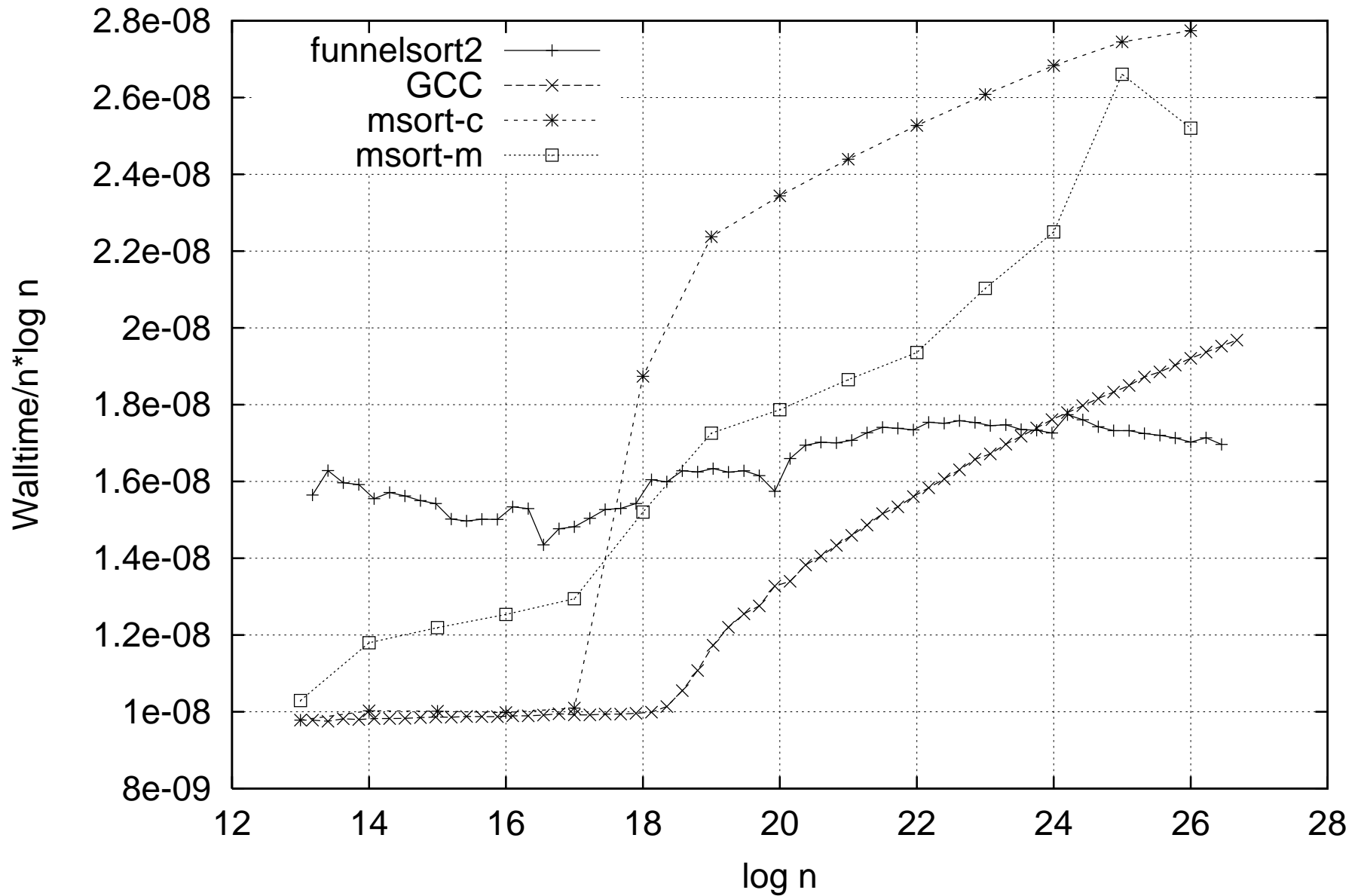
# Searches with Implicit Layouts

Brodal, Fagerberg, Jacob 2002



- van Emde Boas is competitive with a B-tree beyond main memory

### Uniform pairs - Itanium 2



*Engineering a Cache-Oblivious Sorting Algorithm*, Brodal, Fagerberg, Vinther, 2004

# Outline

- Motivation
  - A typical workstation
  - A trivial program
- Memory models
  - I/O model
  - **Ideal cache model**
- Basic cache-oblivious algorithms
  - Matrix multiplication
  - Search trees
  - Sorting
- Some experimental results
- ▶ Conclusion

# Conclusion

- The ideal cache model helps designing competitive and robust algorithms
- Basic techniques: Scanning, recursion/divide-and-conquer, recursive memory layout, sorting
- Many other problems studied: Permuting, FFT, Matrix transposition, Priority queues, Graph algorithms, Computational geometry...

Overhead involved in being cache-oblivious can be small enough for the nice theoretical properties to transfer into practical advantages