Cache-Oblivious Dynamic Dictionaries with Optimal Update/Query Tradeoff

Indeksstrukturer til Ekstern Hukommelse

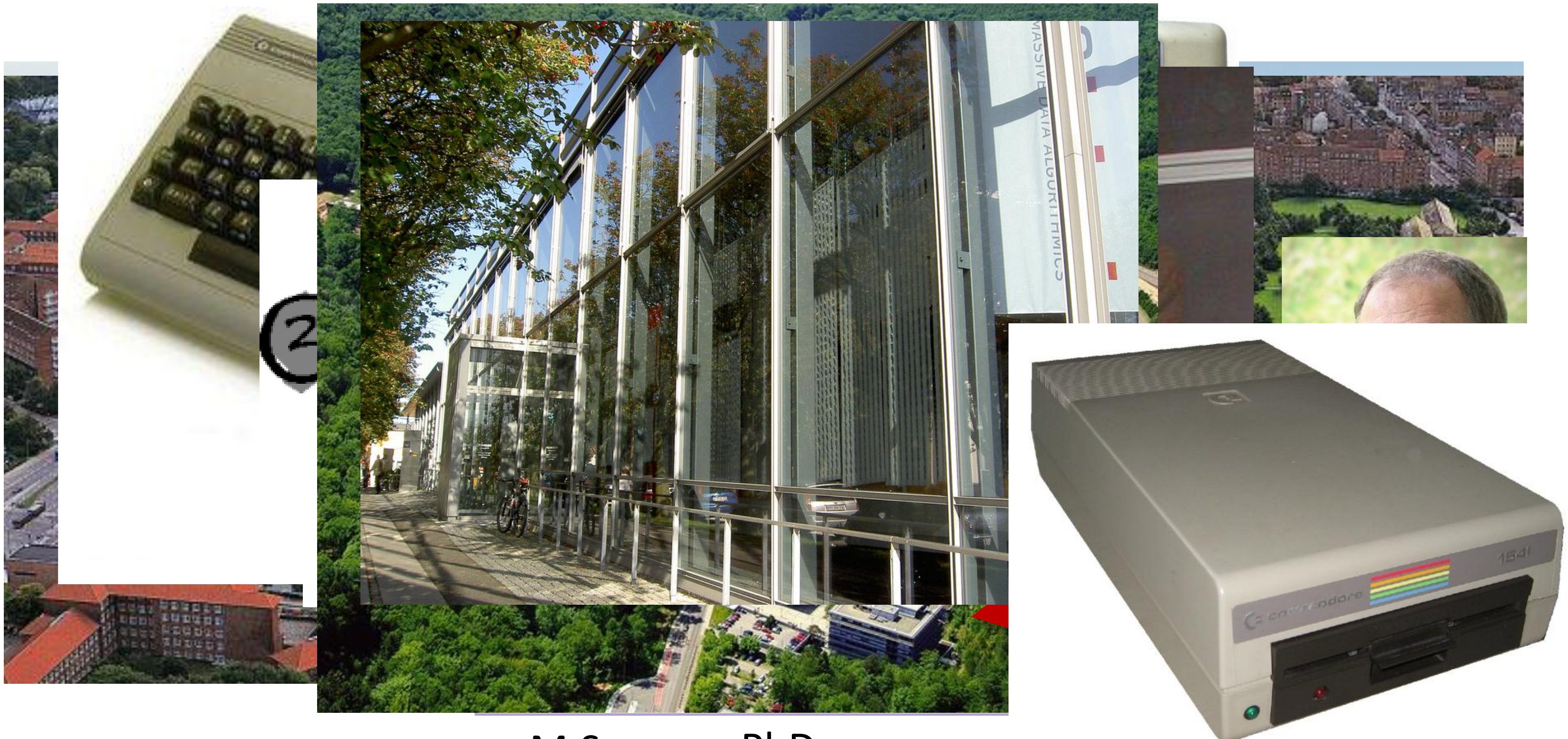
Gerth Stølting Brodal

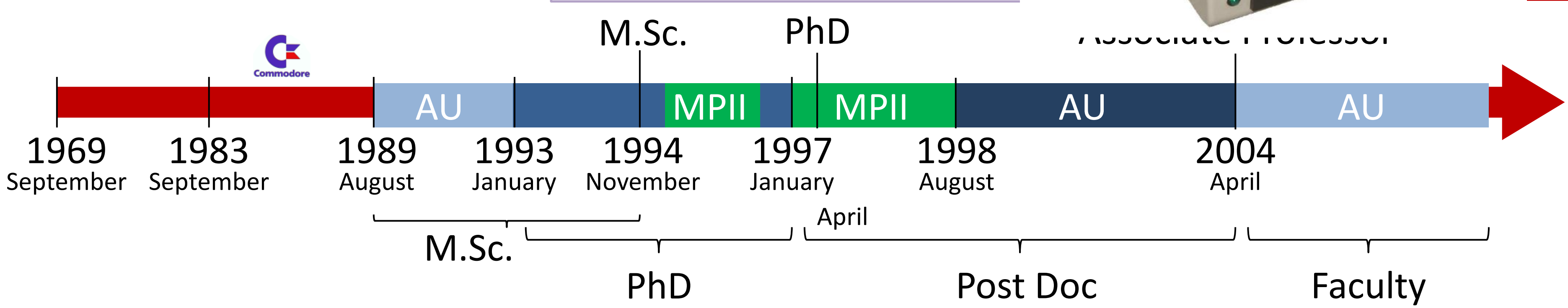
Aarhus University

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

Dansk Selskab for Datalogi, Frederiksbjerg, October 13, 2010

Gerth Stølting Brodal





madalgo

CENTER FOR MASSIVE DATA ALGORITHMICS

MADALGO

CENTER FOR MASSIVE DATA ALGORITHMICS

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

- Center of Danmarks Grundforskningsfond Danish National Research Foundation
- **Lars Arge**, Professor, Centerleader
- Gerth S. Brodal, Associate Professor
- 5 Post Docs, 10 PhD students, 4 TAP
- Total budget for 5 years ca. 60 million DKR

AU

Arge

Brodal

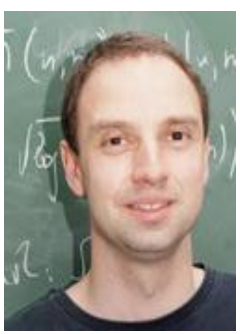
MIT

Demaine

Indyk

MPII

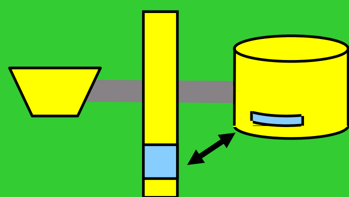
Mehlhorn

Frankfurt

Meyer

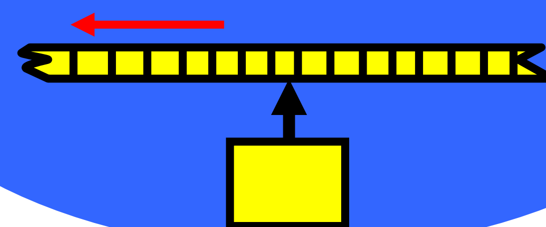
MADALGO

CENTER FOR MASSIVE DATA ALGORITHMICS

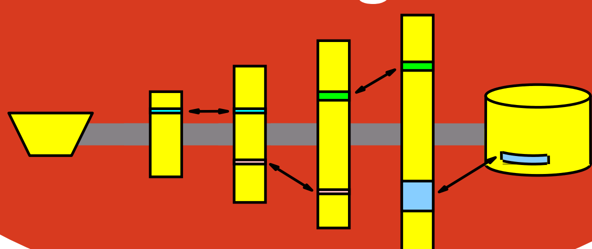
I/O Efficient Algorithms



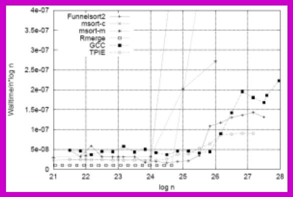
Streaming Algorithms

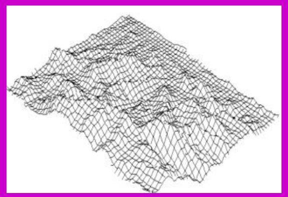


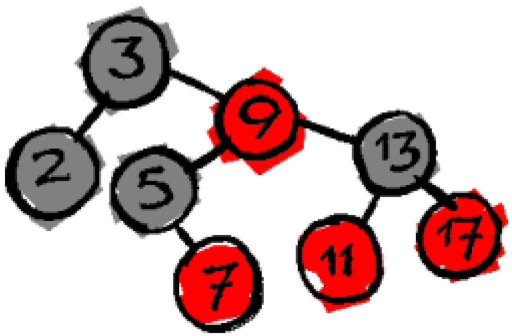
Cache Oblivious Algorithms



Algorithm Engineering

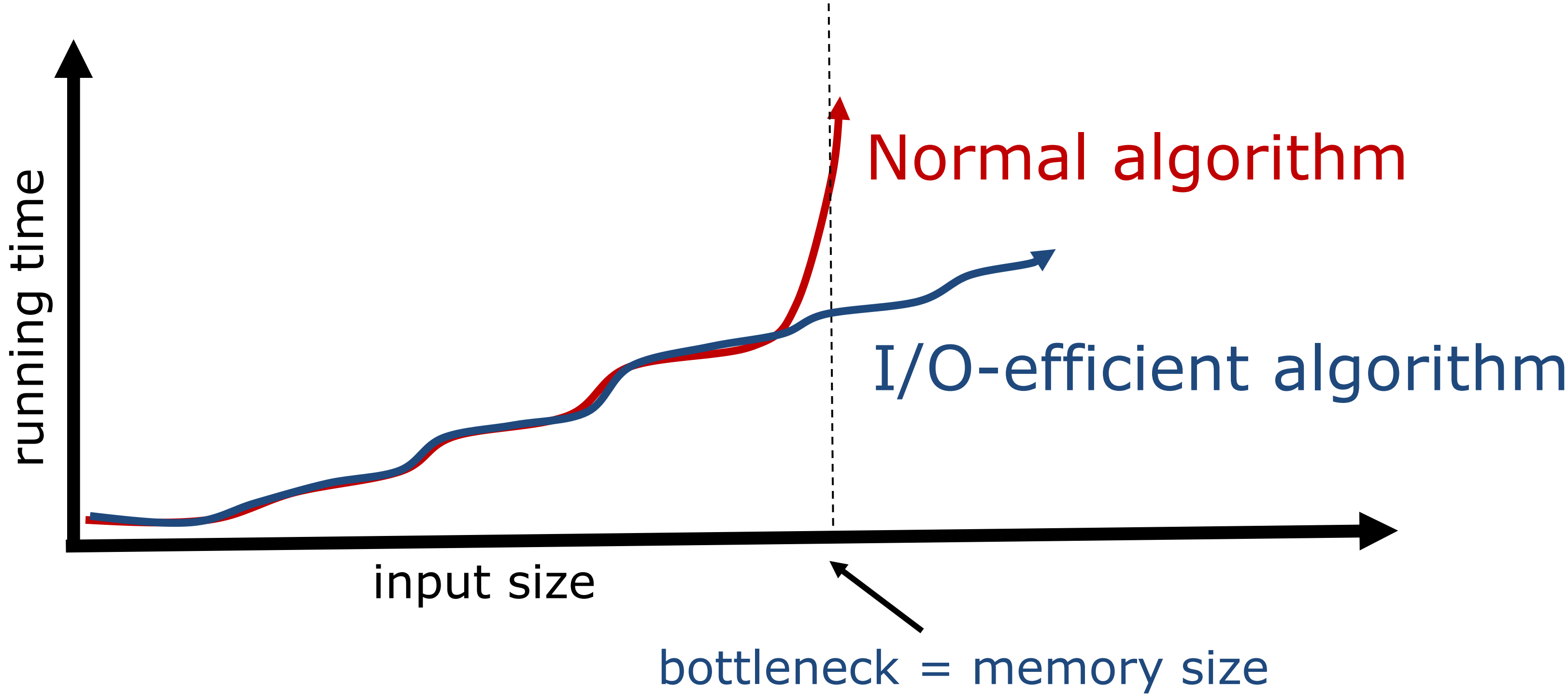




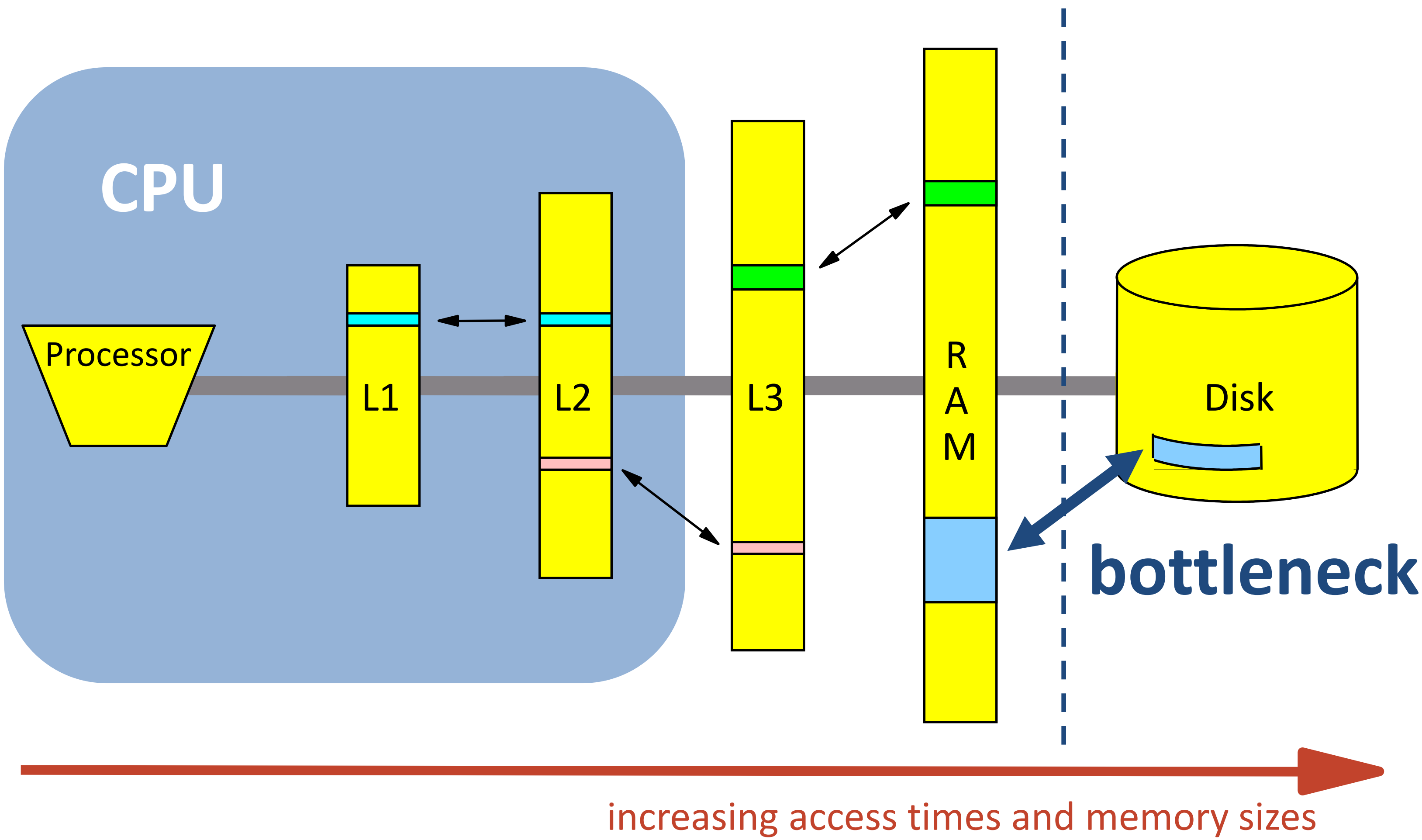

3
9
2
5
13
7
11
17

+

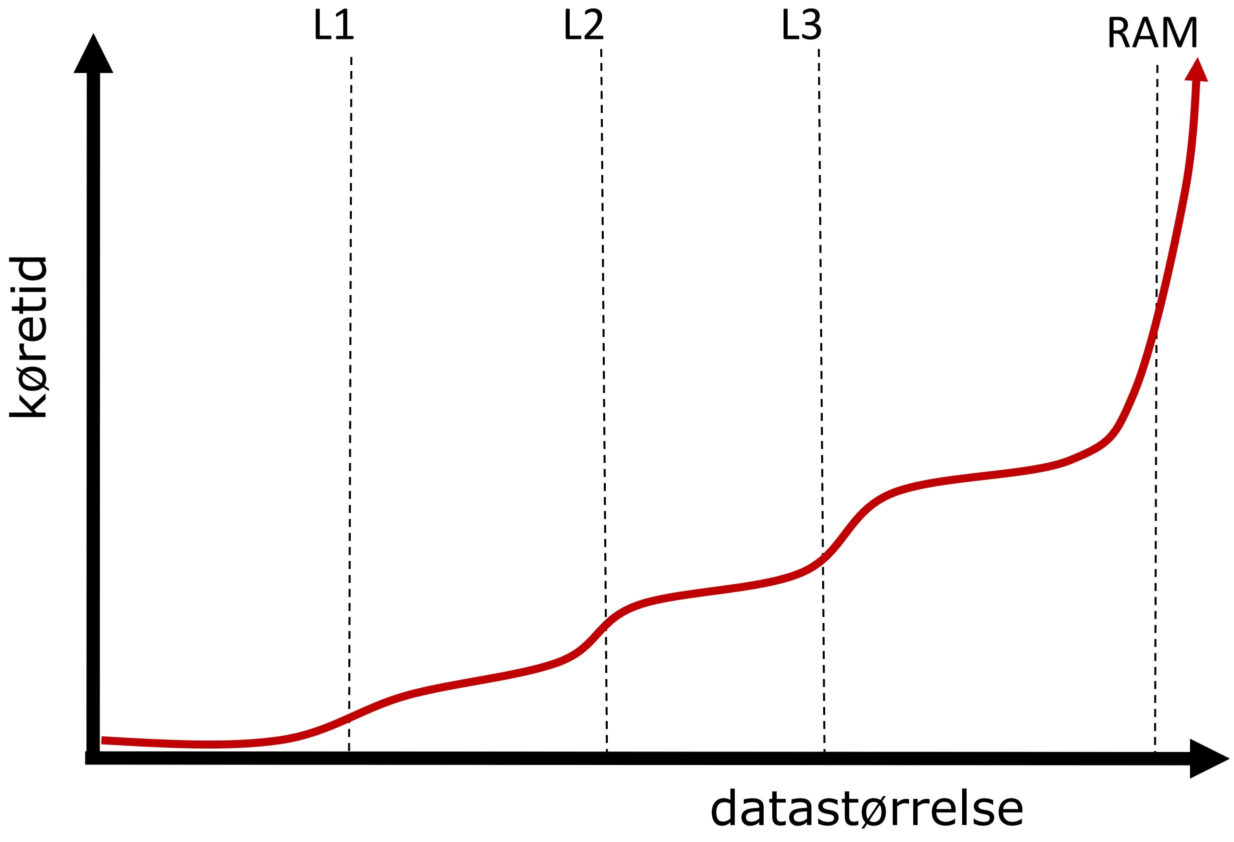
The problem...


Normal algorithm
I/O-efficient algorithm
running time
input size
bottleneck = memory size

Memory Hierarchies



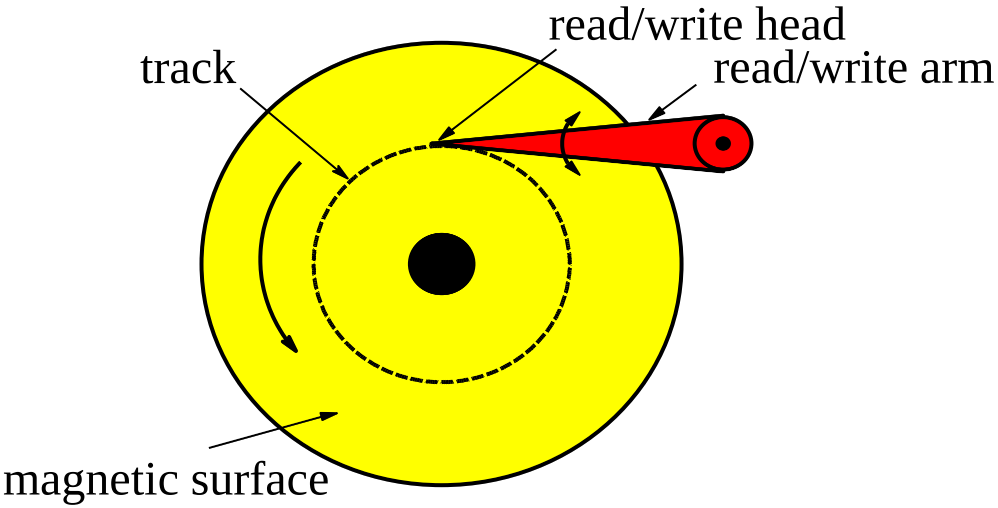
Memory Hierarkies
vs.
Running Time



Memory Access Times

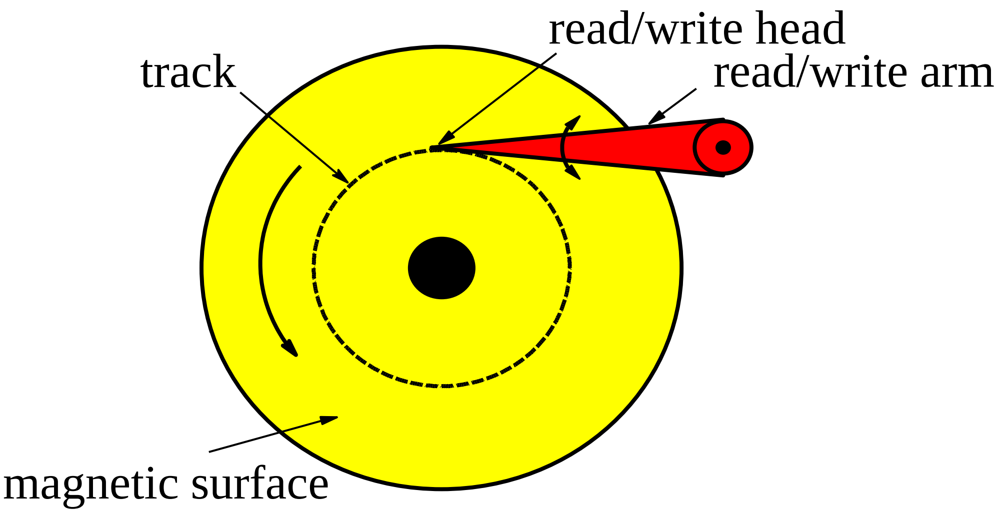
	Latency	Relative to CPU
Register	0.5 ns	1
L1 cache	0.5 ns	1-2
L2 cache	3 ns	2-7
DRAM	150 ns	80-200
TLB	500+ ns	200-2000
Disk	10 ms	10^7

Disk Mechanics



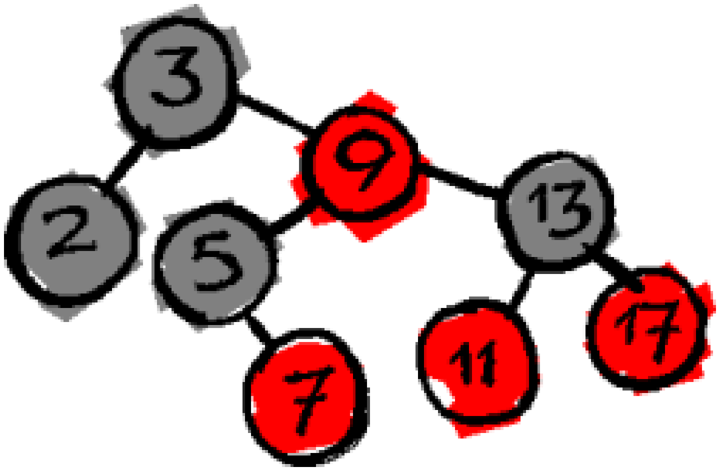
"The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on one's desk or by taking an airplane to the other side of the world and using a sharpener on someone else's desk." (D. Comer)

Disk Mechanics



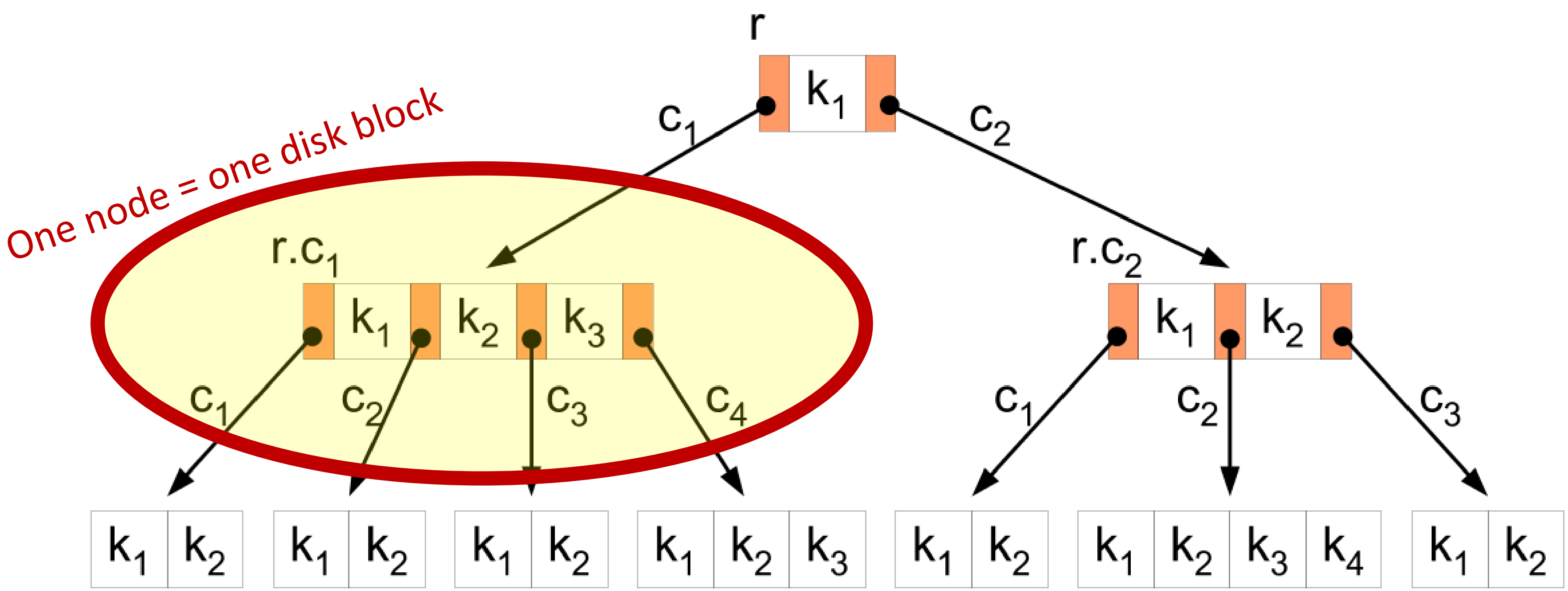
- I/O is often bottleneck when handling massive datasets
- Disk access is 10^7 times slower than main memory access!
- Disk systems try to amortize large access time transferring large contiguous blocks of data
- Need to store and access data to take advantage of blocks !

Internal Memory Index



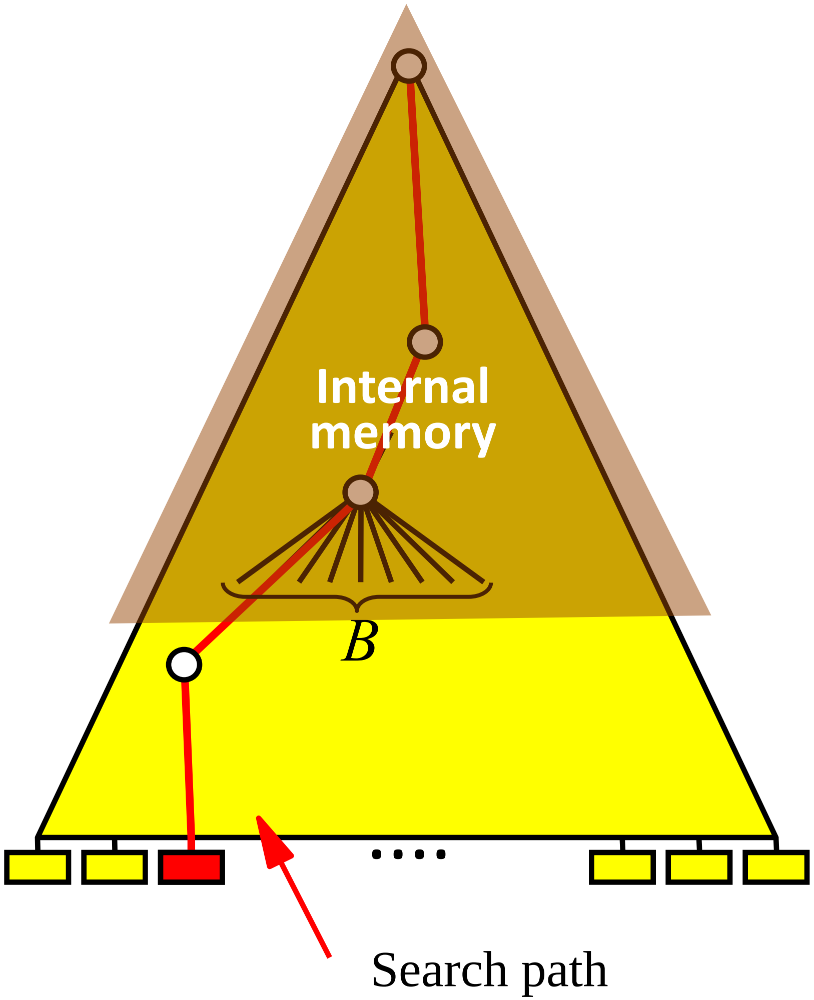
Bayer og McCreight (1971)

B(oing)-Trees



Height $O(\log_B N)$

B-trees -
The Basic Searching Structure



- **Searches**
  Practice: 4-5 I/Os

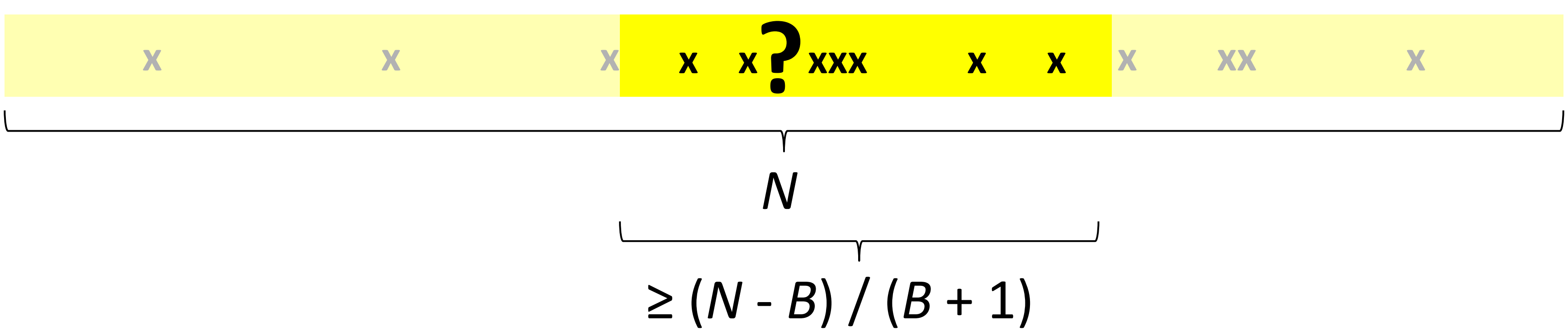
- **Repeated searching**
  Practice: 1-2 I/Os

The Bad News...

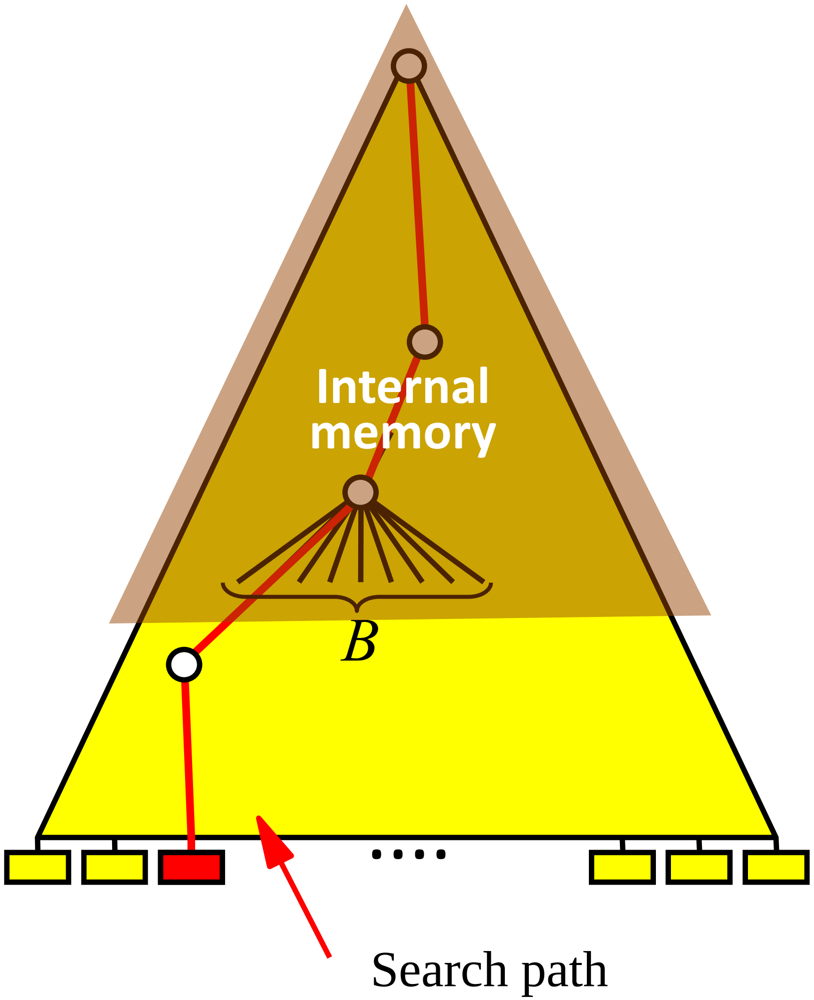
Searching any external memory dictionary (incl. B-trees) requires worst-case

$$\Omega(\log_B N) \quad \text{I/Os}$$

Proof idea:



B-trees

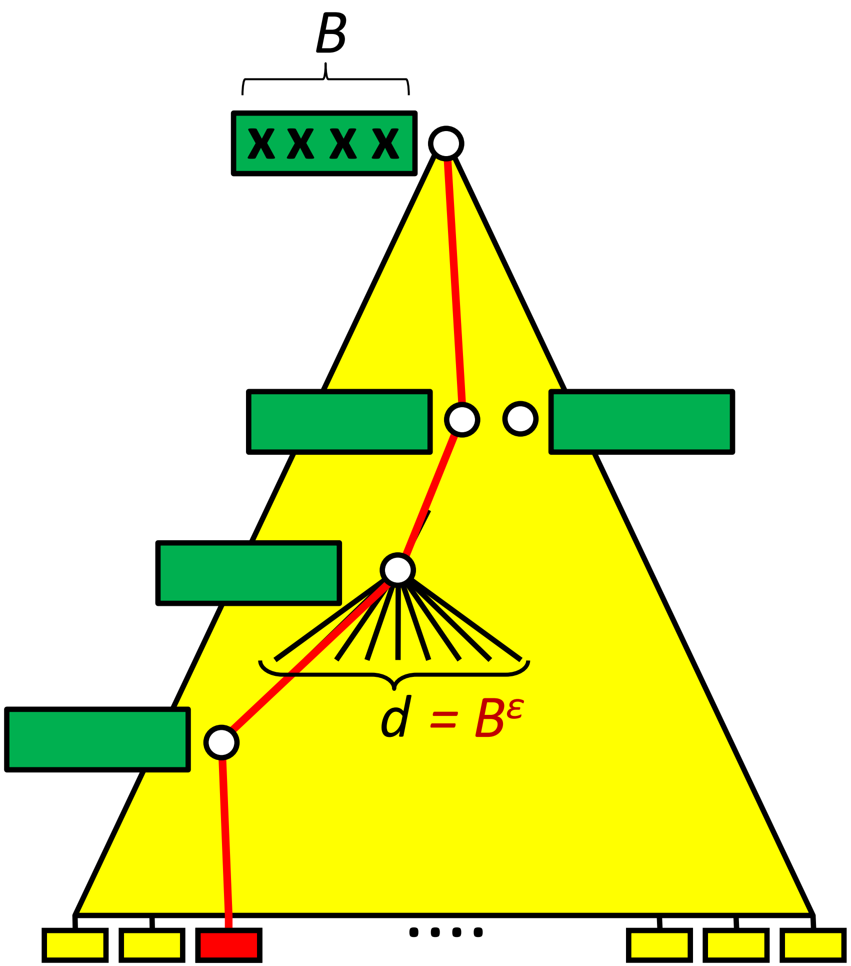


Best possible

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

?

B-trees with Buffered Updates



- **Searches cost**

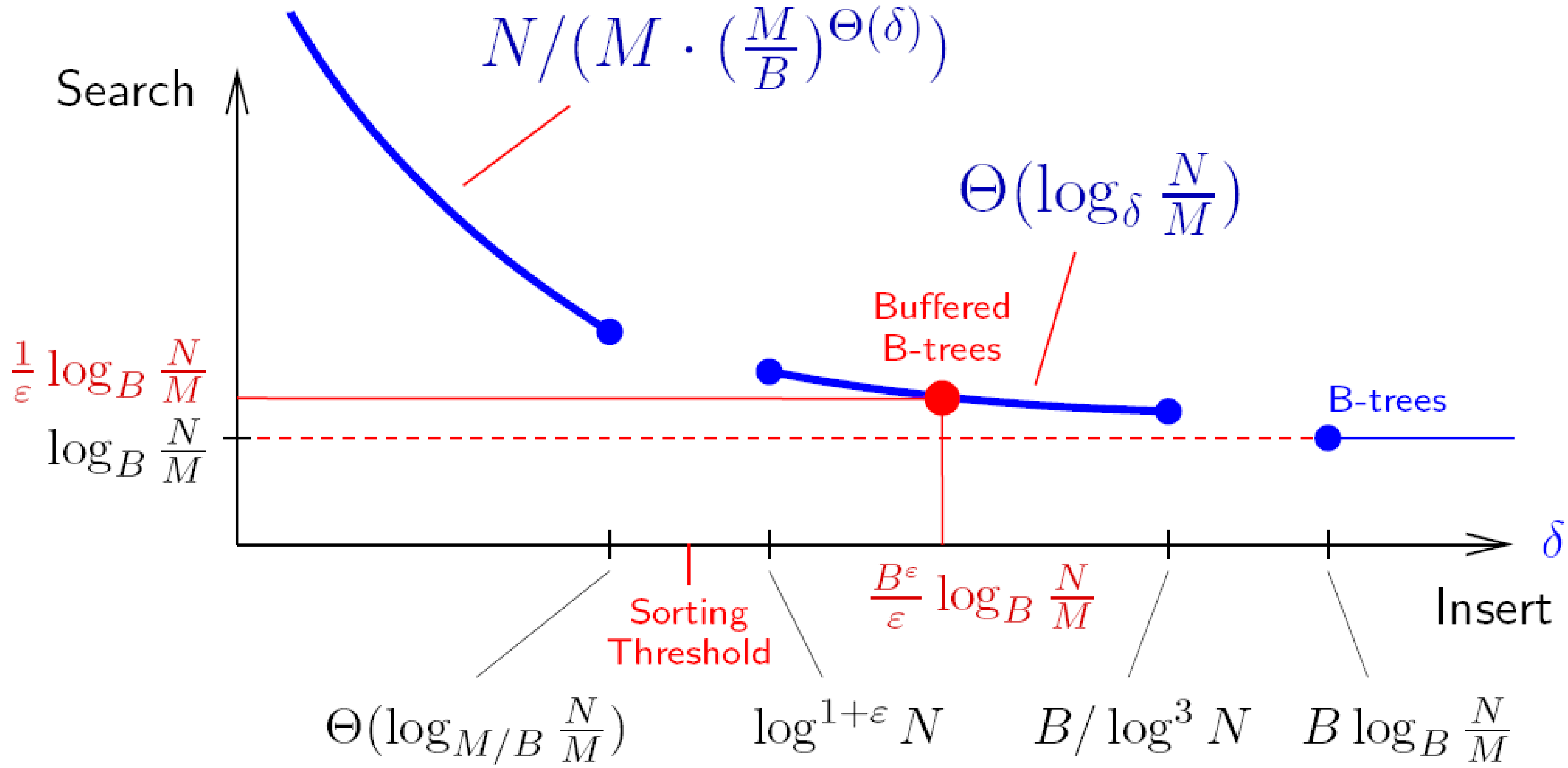
$$O(\log_d N) \quad \text{I/Os}$$
$$= O(\log_B N \cdot \underbrace{1/\varepsilon}_{2}) \quad \text{I/Os}$$

- ***N* updates cost**

$$O(N \cdot \log_d N \cdot d / B) \quad \text{I/Os}$$
$$= O(N \cdot \log_B N \cdot \underbrace{1/\varepsilon B^{1-\varepsilon}}_{2/\sqrt{B}}) \quad \text{I/Os}$$

Trade-off between search and update times – optimal *!*

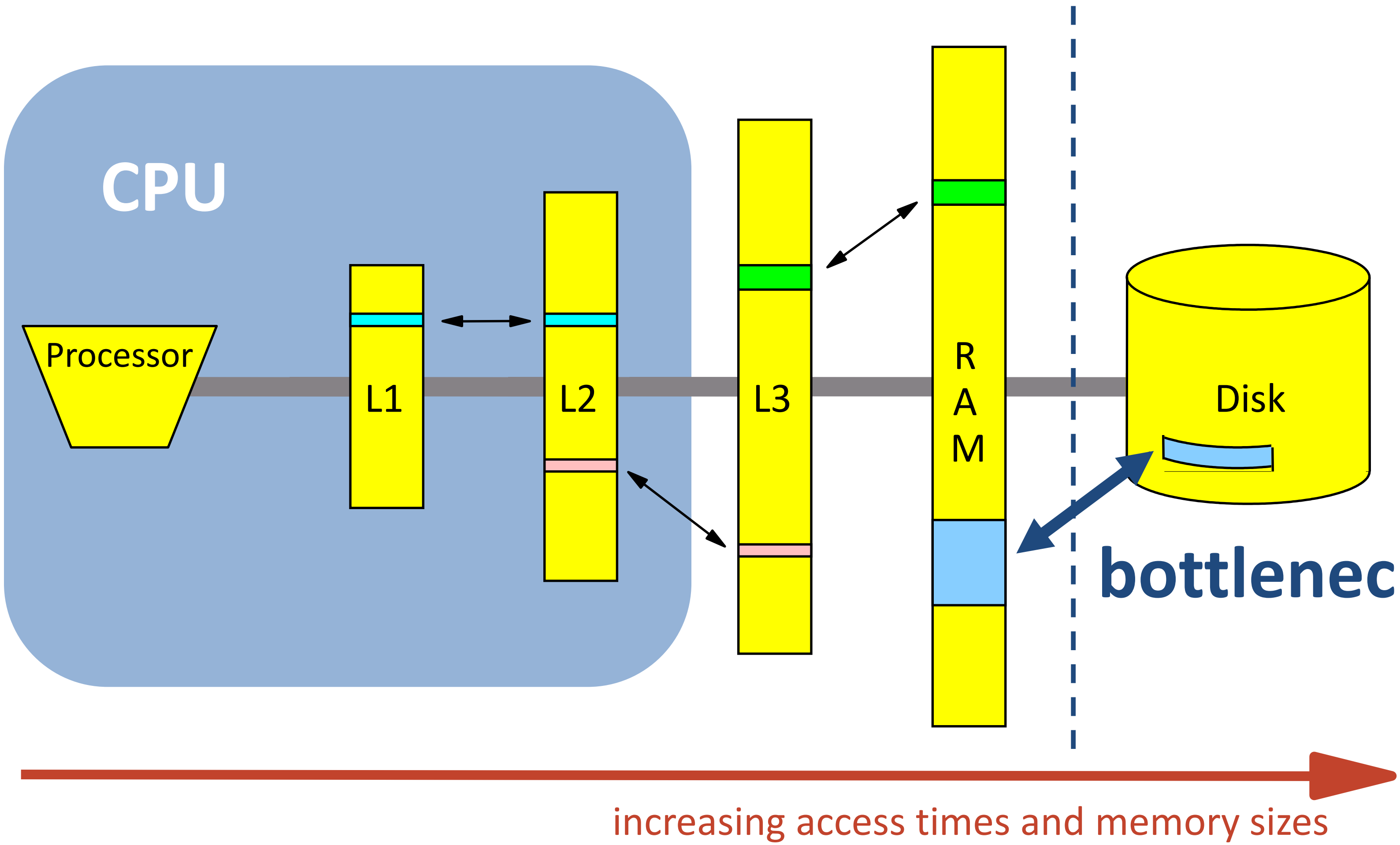
B-trees with Buffered Updates



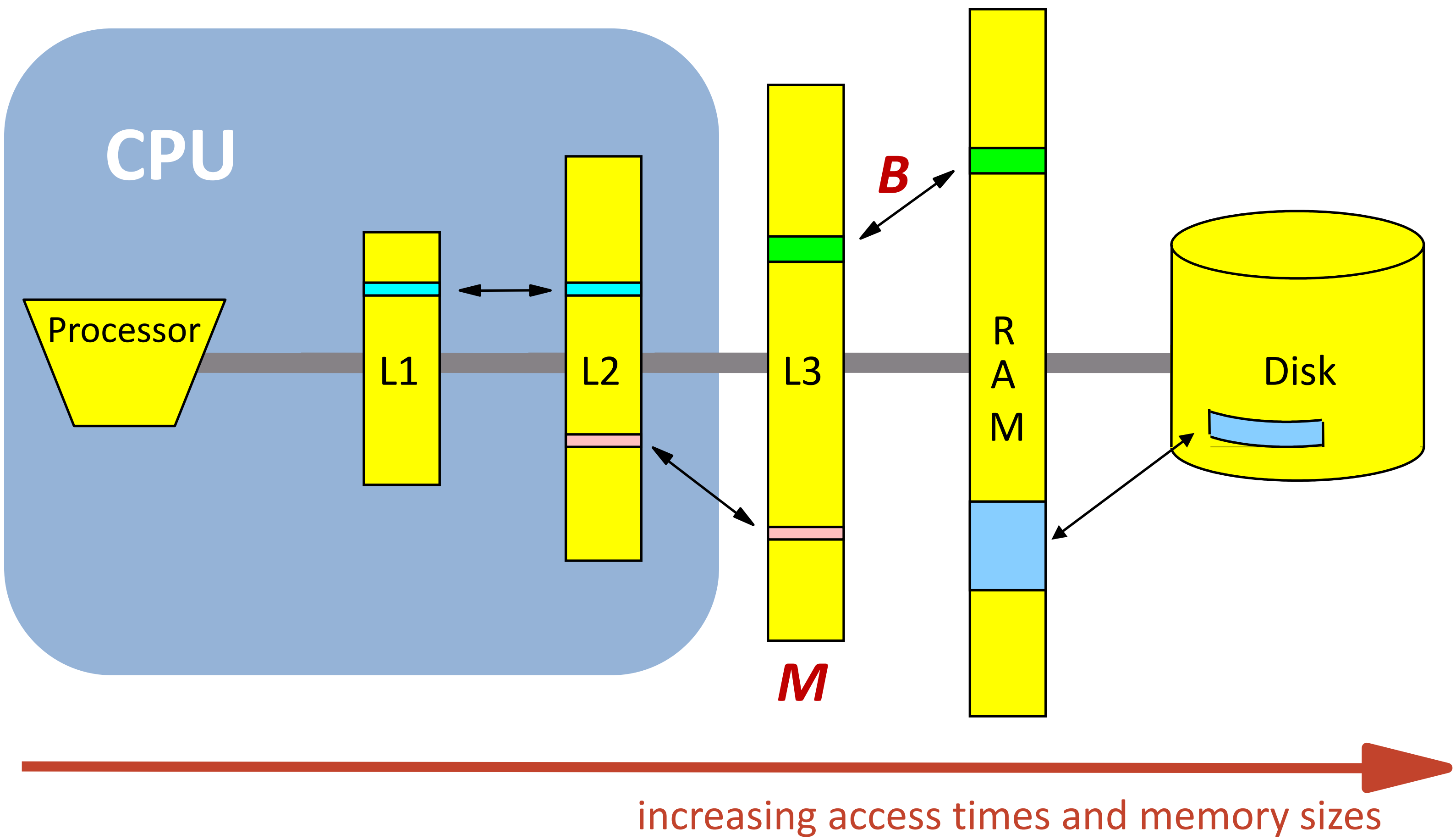
B-trees with Buffered Updates Experimental Study

- 100.000.000 elements
- Search time basically unchanged with buffers
- 100 times faster updates

Assumptions until now : *B* is known + one level is the bottleneck



Cache-Obliviousnes Model

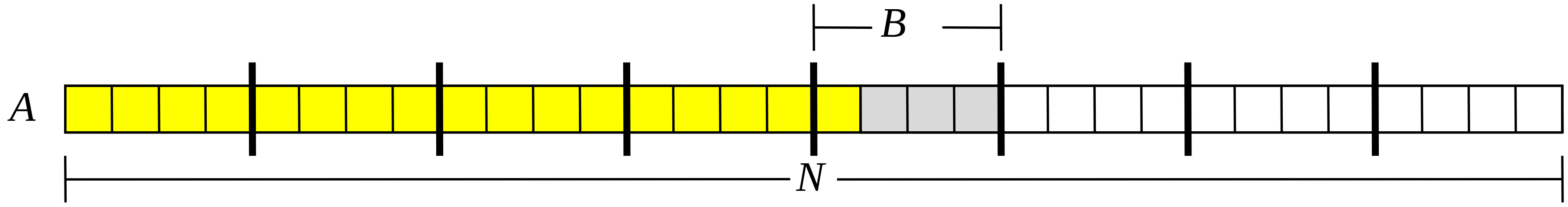


Algorithm **does not know** *B* and *M*

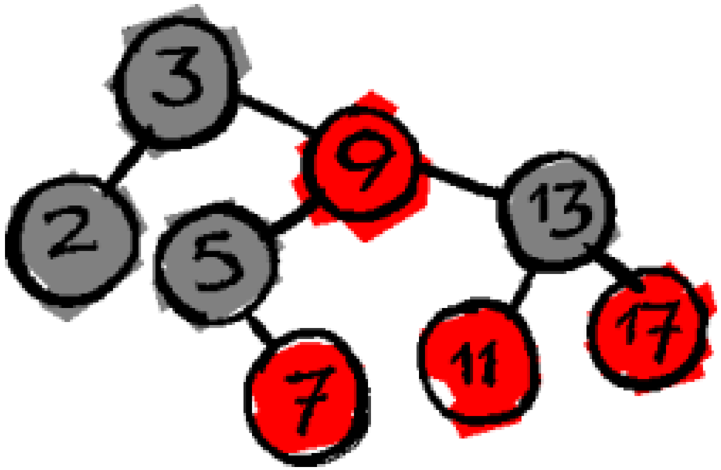
(assume optimal offline cache-replacement strategy)

I/O Efficient Scanning

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



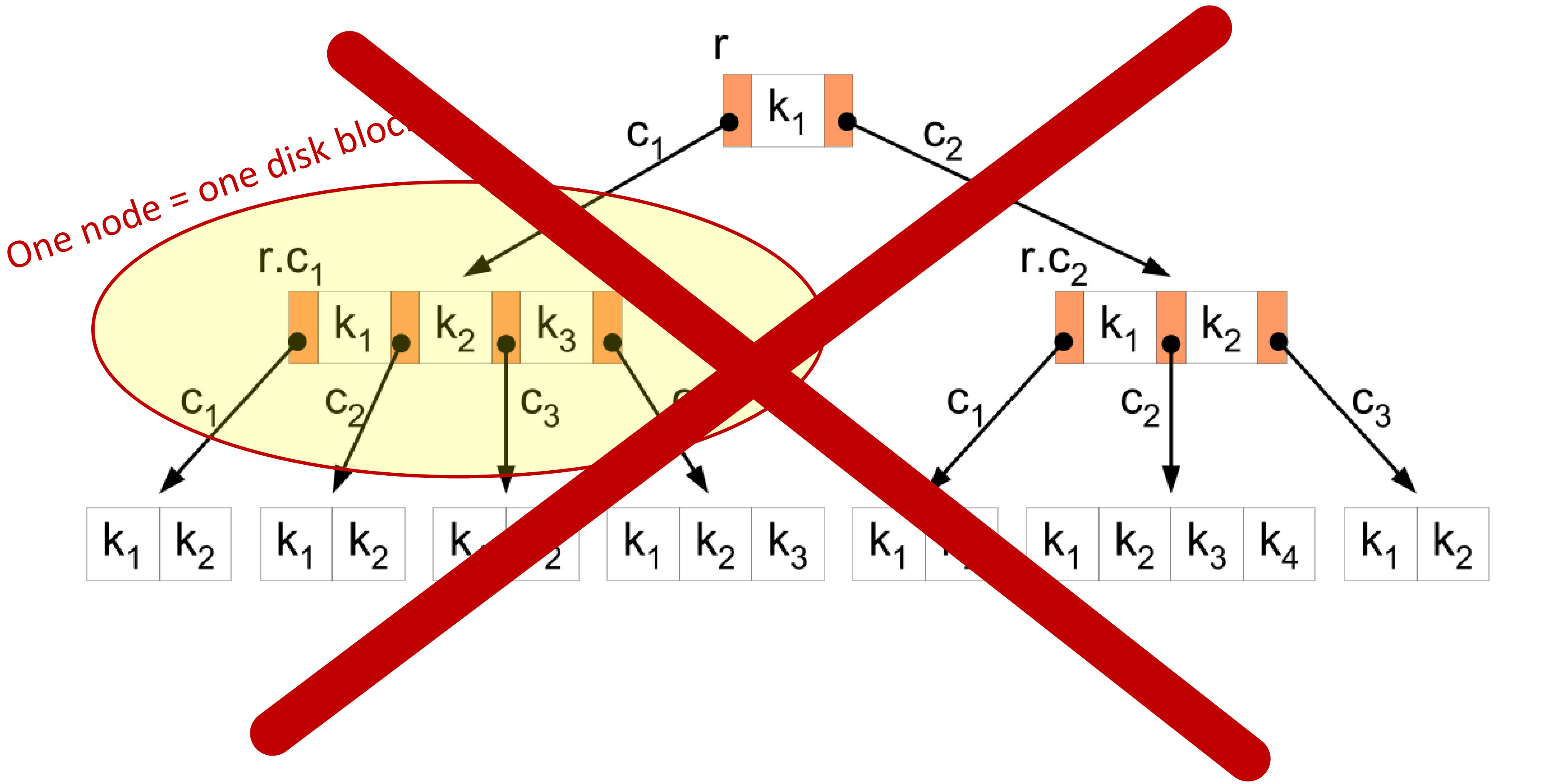
$O(N/B)$ I/Os



3
9
2
5
13
7
11
17

Bayer og McCreight (1971)

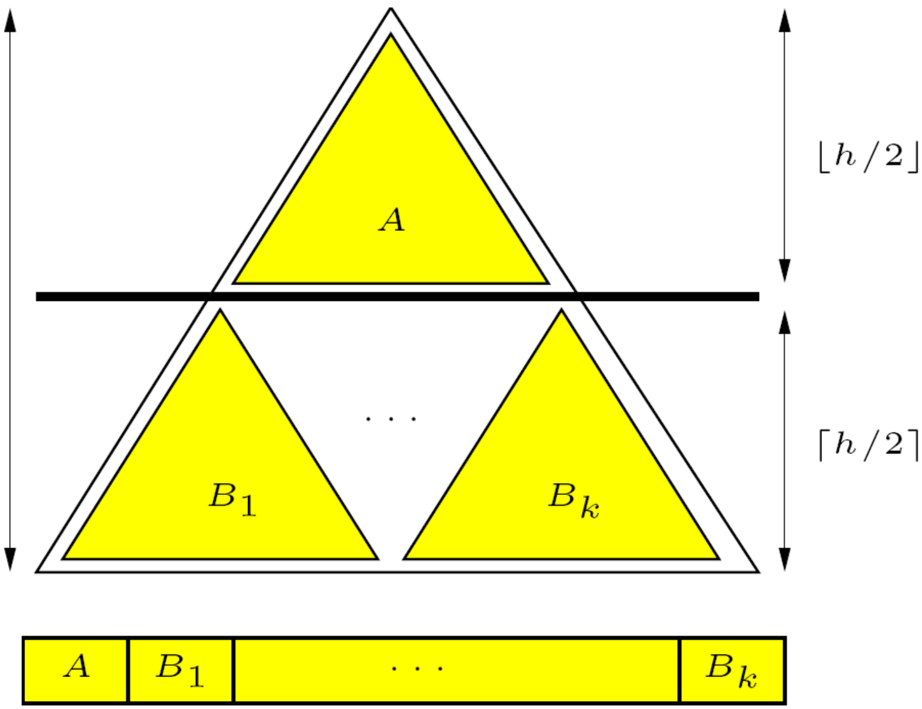
B(oing)-Trees

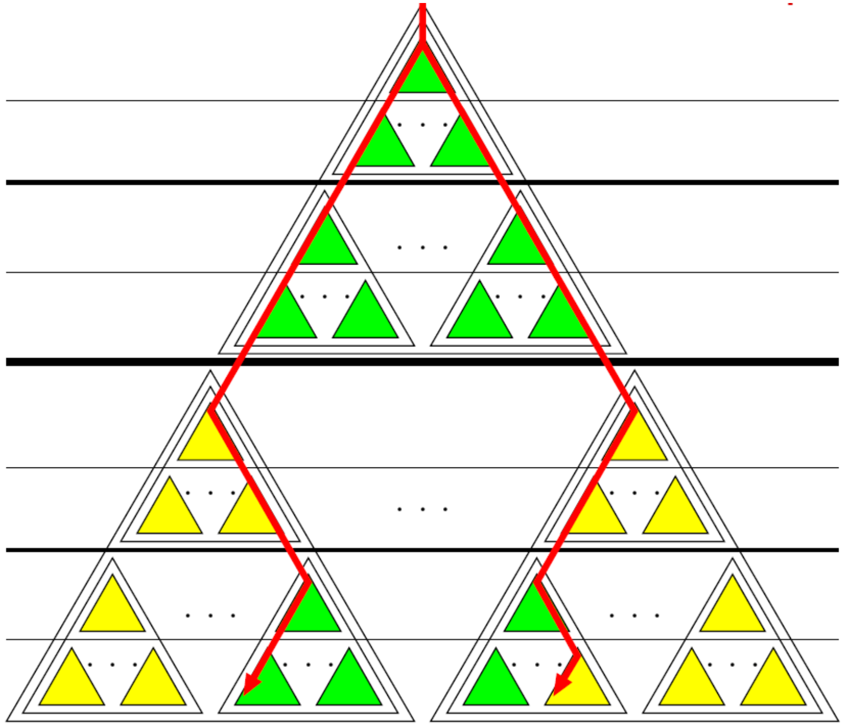


Height $O(\log_B N)$

Frigo (1999)

Recursive Search Tree Layout (Cache-Obliviousness)

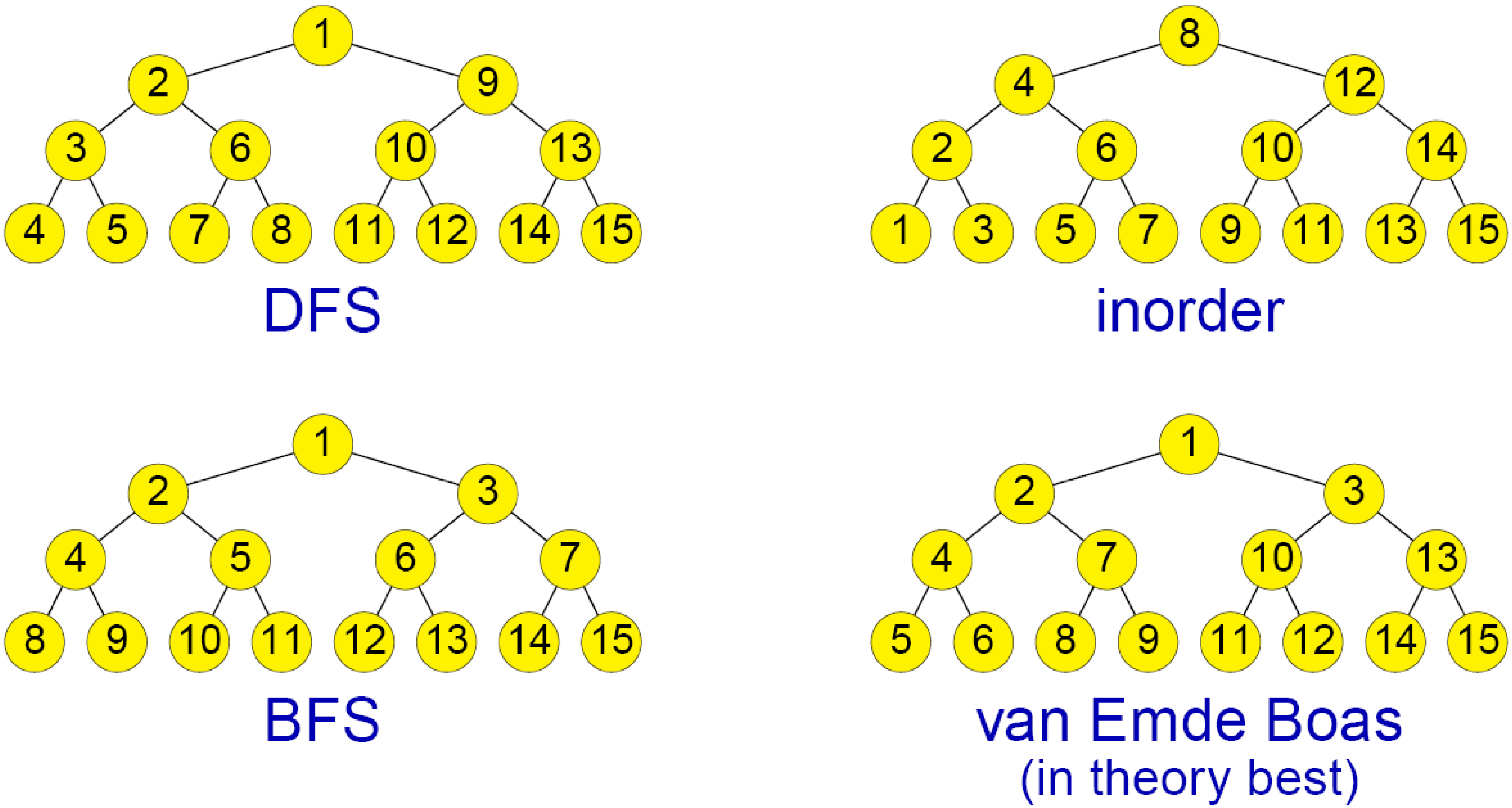




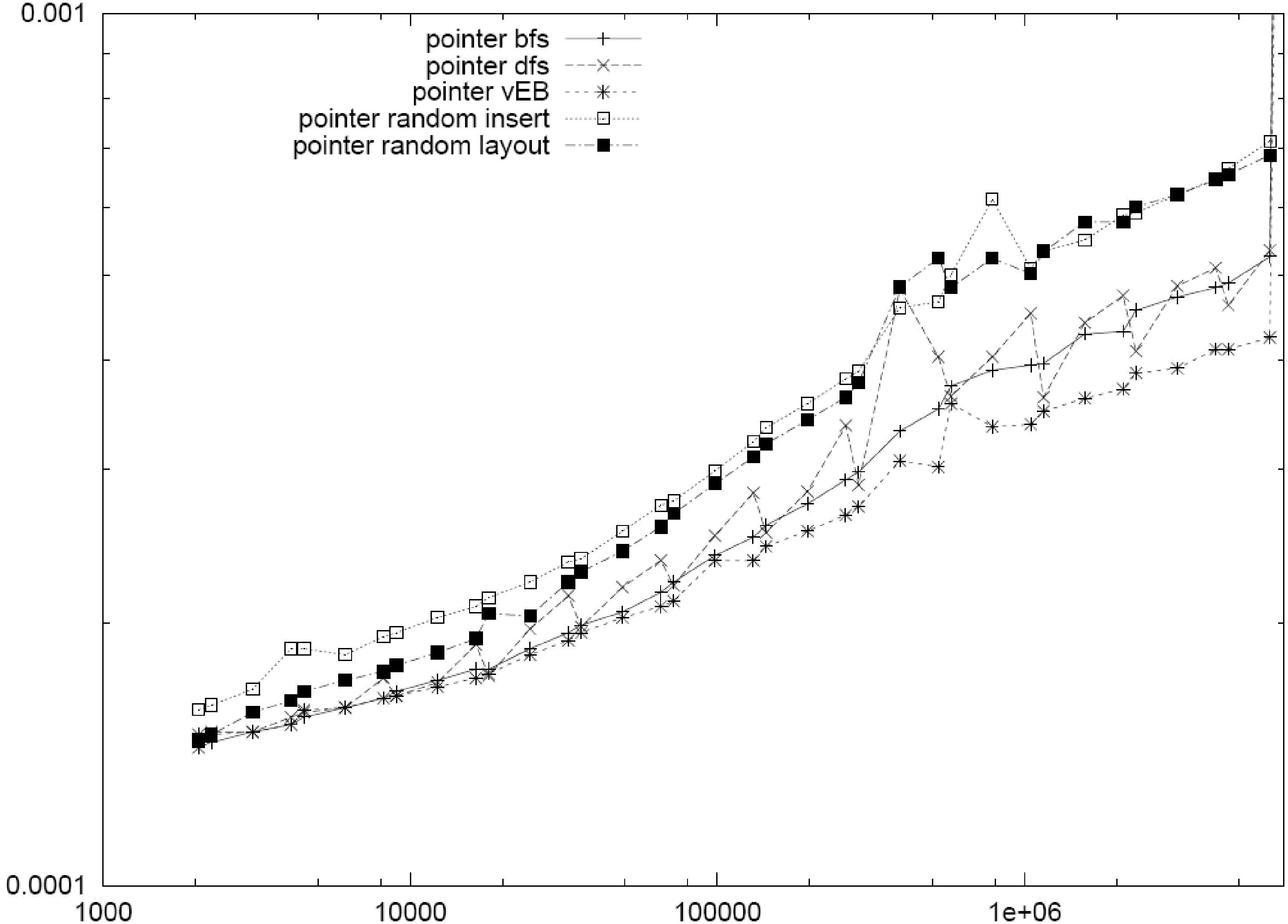
Searches require $O(\log_B N)$ I/Os

(small subtrees have $\sqrt{B} \leq$ size $\leq B$, a path traverses at most $2 \cdot \log_B N$ trees)

Experiments : Binary Tree Layout

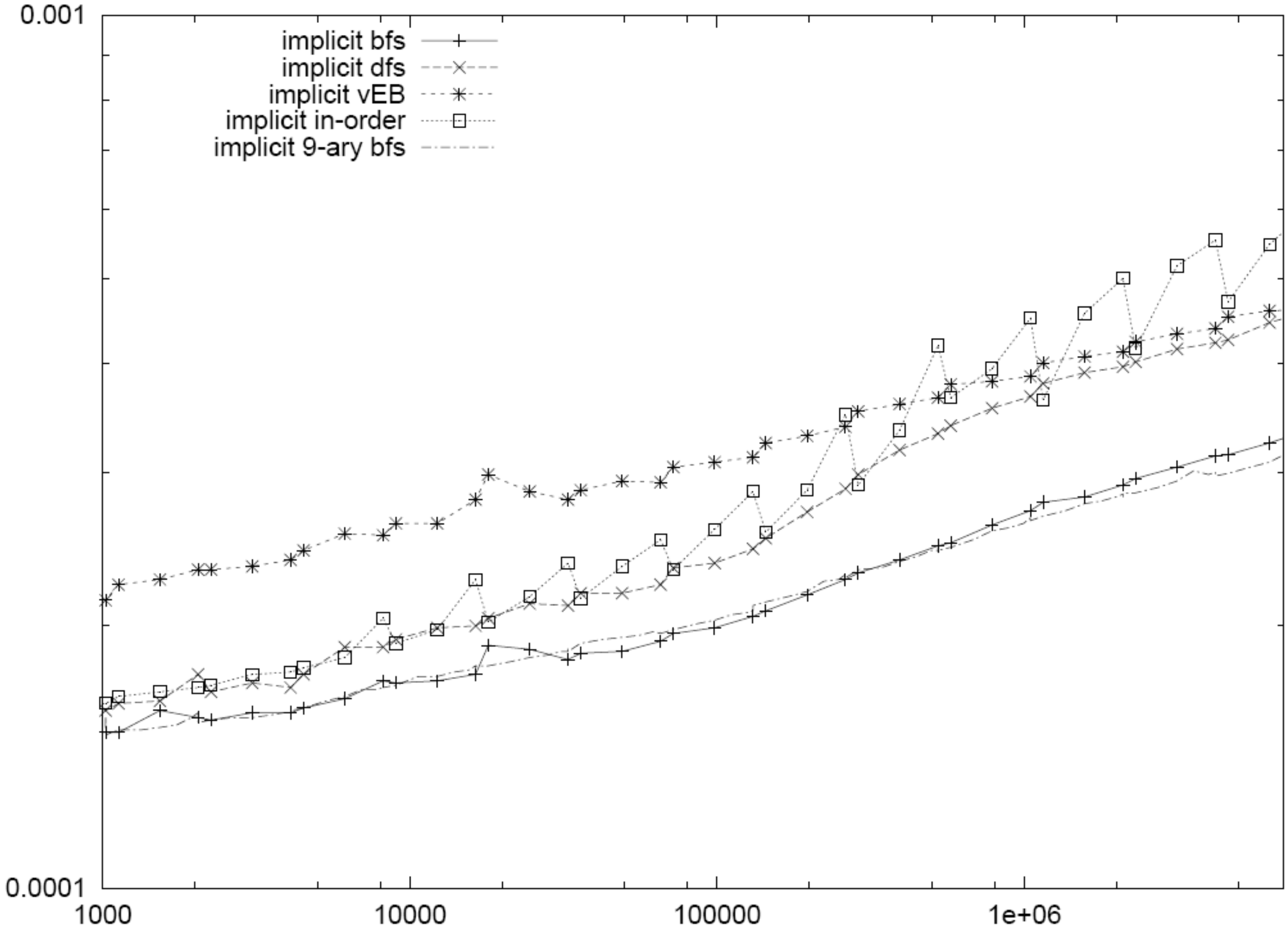


Searches with Pointer Layout



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

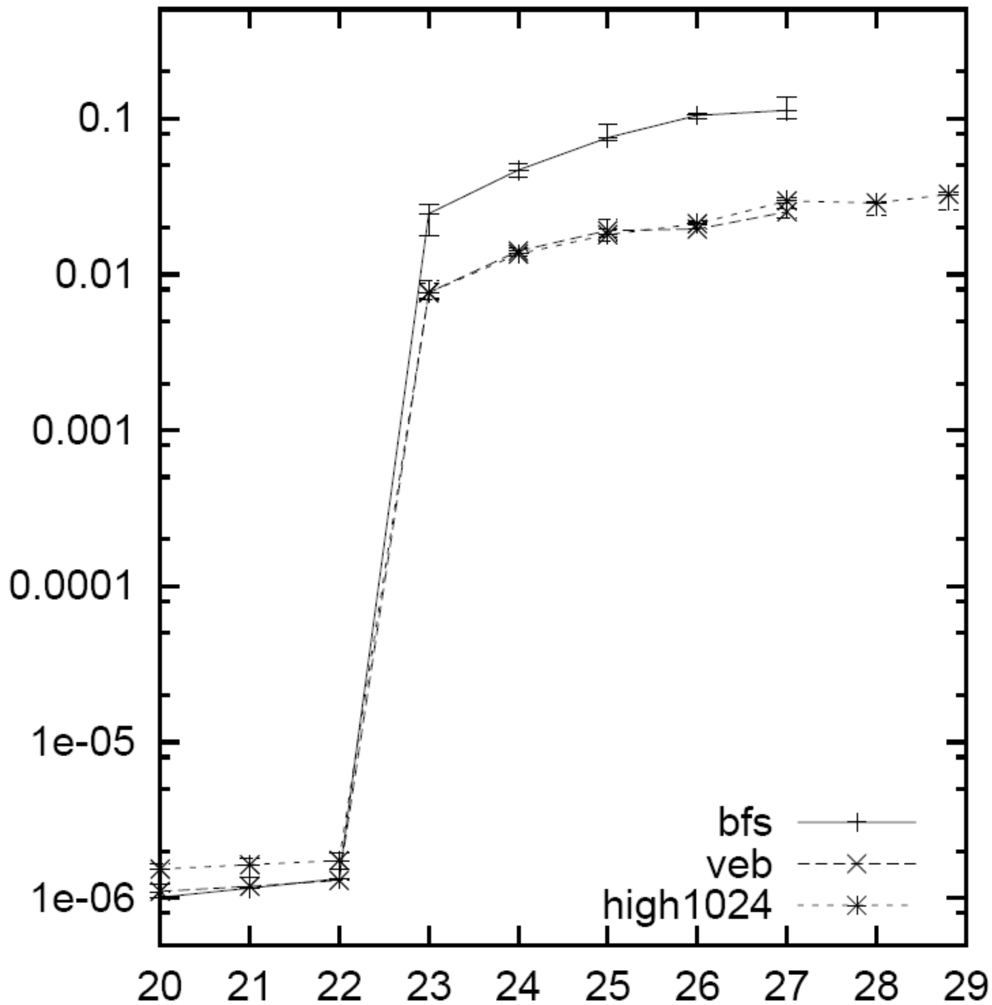
Searches with Implicit Layout



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

Brodal, Fagerberg, Jacob (2002)

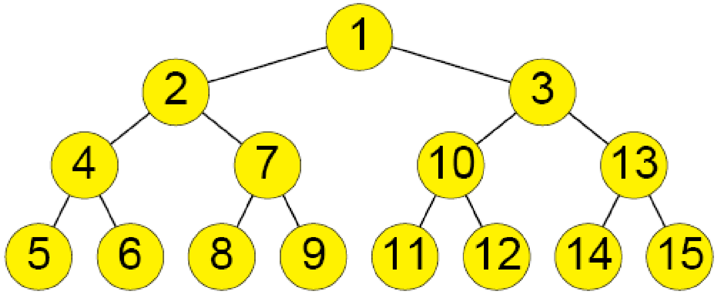
Searches with Pointer Layout - Beyond Main Memory



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

Optimal Cache-Oblivious Static Index

= van Emde Boas Layout of Complete Binary Tree

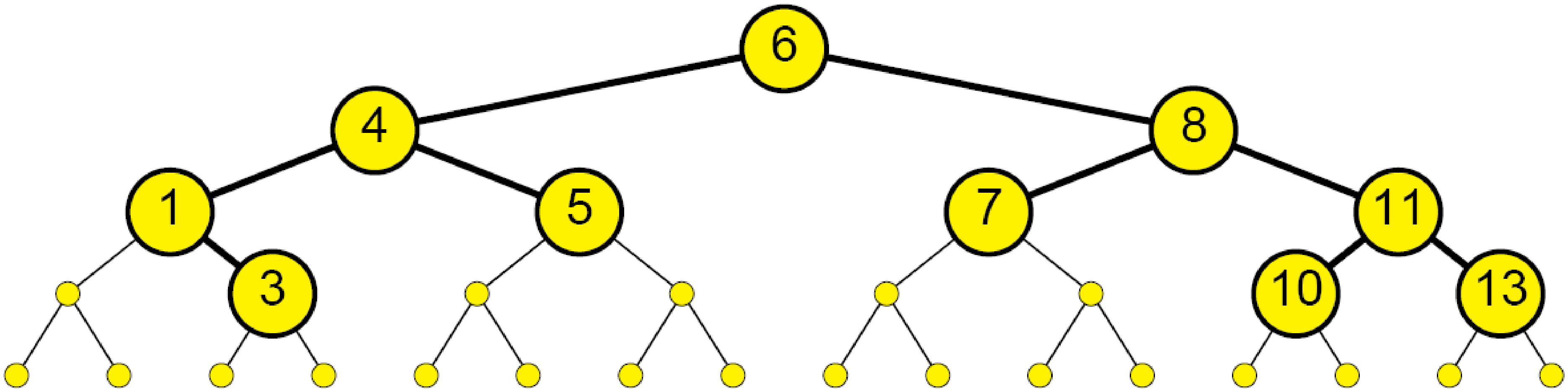


van Emde Boas

Cache-Oblivious <u>Dynamic</u> Index ?

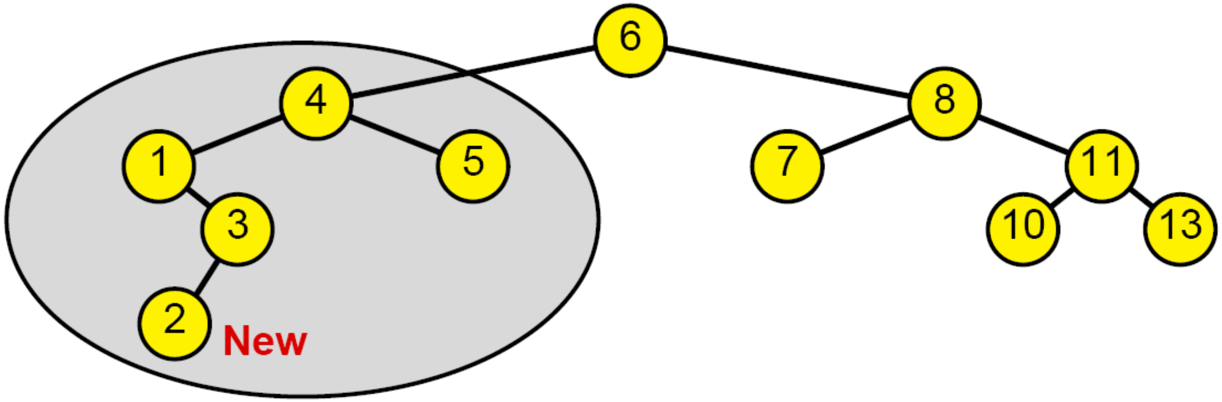
Brodal, Fagerberg, Jacob (2002)

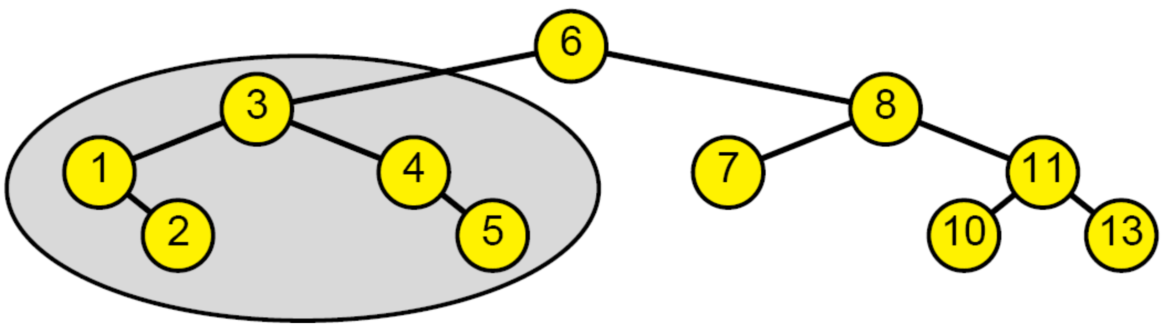
Binary Search Trees
Dynamic



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

Rebalancing Subtree during Insertion

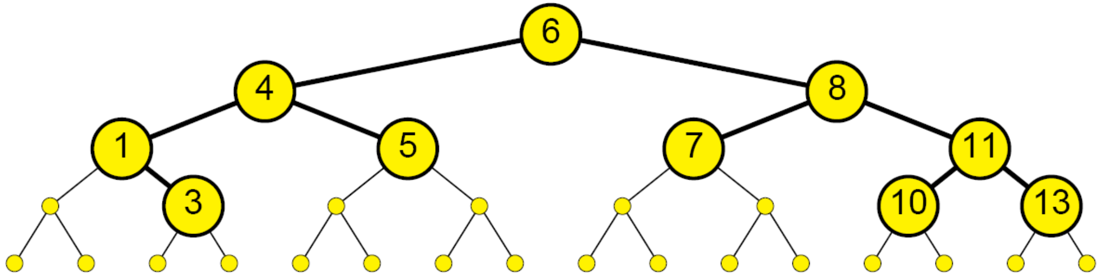




- If an insertion causes non-small height then **rebuild** subtree at nearest ancestor with sufficient few descendants
- Insertions require amortized time $O(\log^2 N)$

Optimal Cache-Oblivious Dynamic Index

- Search $O(\log_B N)$ I/Os (optimal)
- Updates $O(\log_B N + (\log^2 N) / B)$ I/Os



Cache-Oblivious Index
with Query-Update Trade-off ?

Brodal, Demaine, Fineman, Iacono, Langerman, Munro (2010)

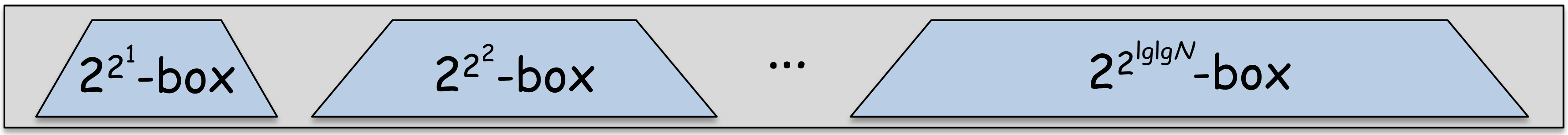
Cache-Oblivious Dynamic Dictionaries with Query-Update Trade-off

- Solution matching the I/O complexity of Buffered B-trees

- Searches $O(\log_B N \cdot 1/\varepsilon)$ I/Os

- N updates $O(N \cdot \log_B N \cdot 1/\varepsilon B^{1-\varepsilon})$ I/Os

xDict



Insert

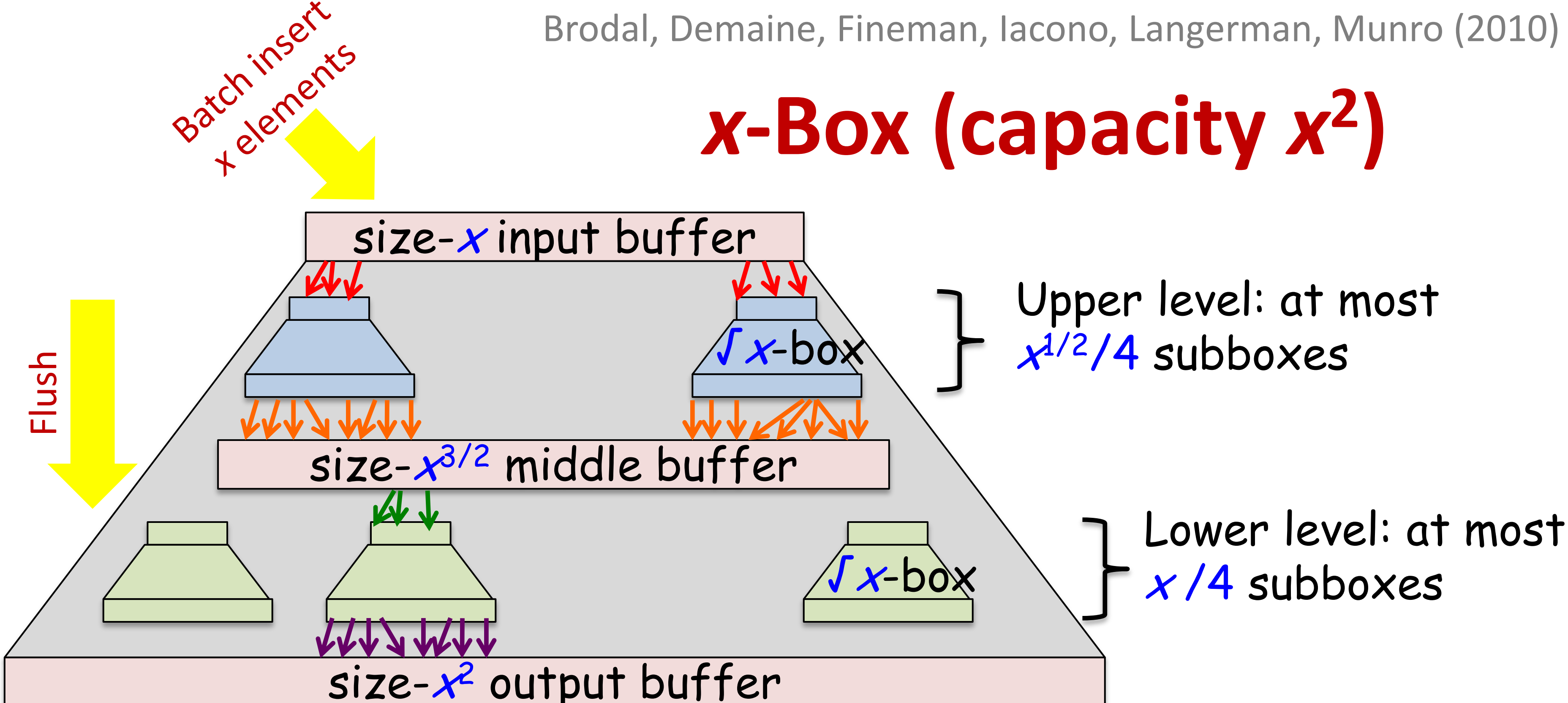
- Insert into smallest box
- When a box reaches capacity, **Flush** it and **Batch-Insert** into the next box

Search

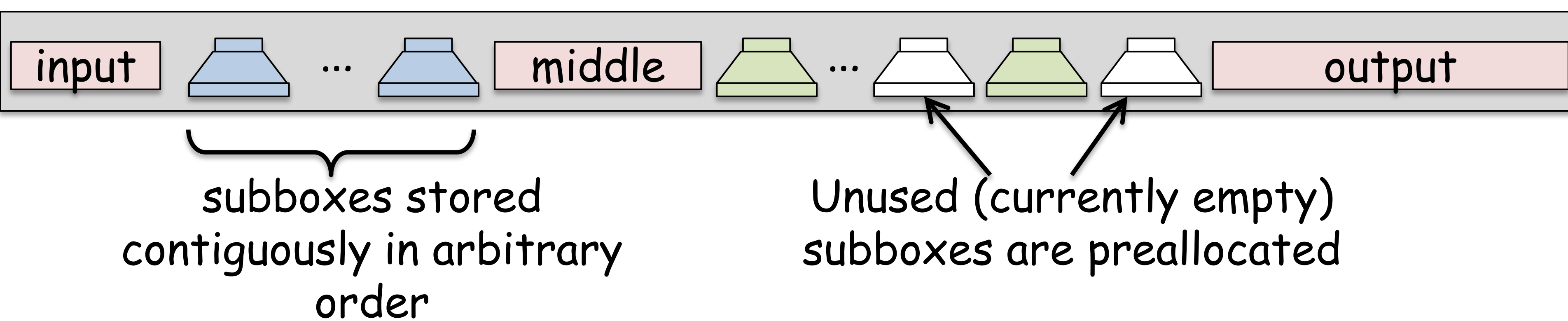
- Search in each x-box
- $O(\log_B x)$ cost is dominated by largest box $O(\log_B N)$

Brodal, Demaine, Fineman, Iacono, Langerman, Munro (2010)

x-Box (capacity x^2)



Memory Layout



External Memory Index

- Searches $O(\log_B N \cdot 1/\varepsilon)$ I/Os
- N updates $O(N \cdot \log_B N \cdot 1/\varepsilon B^{1-\varepsilon})$ I/Os