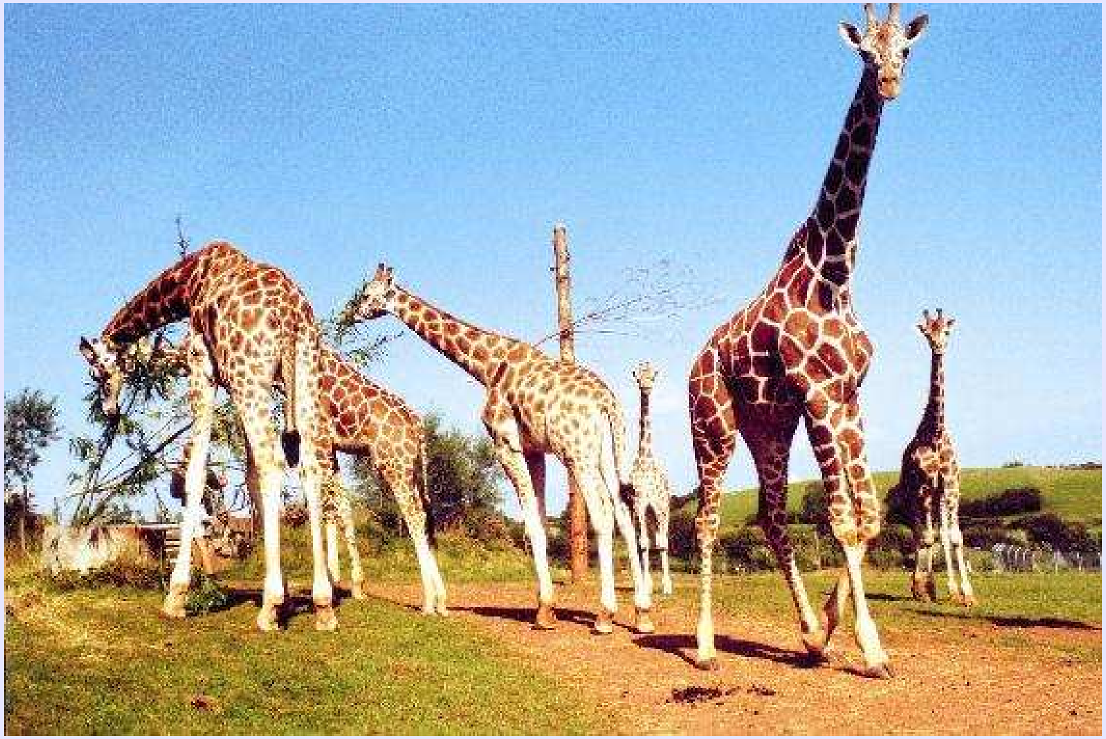
Cache-Oblivious String Dictionaries



Gerth Stølting Brodal

University of Aarhus

Joint work with Rolf Fagerberg

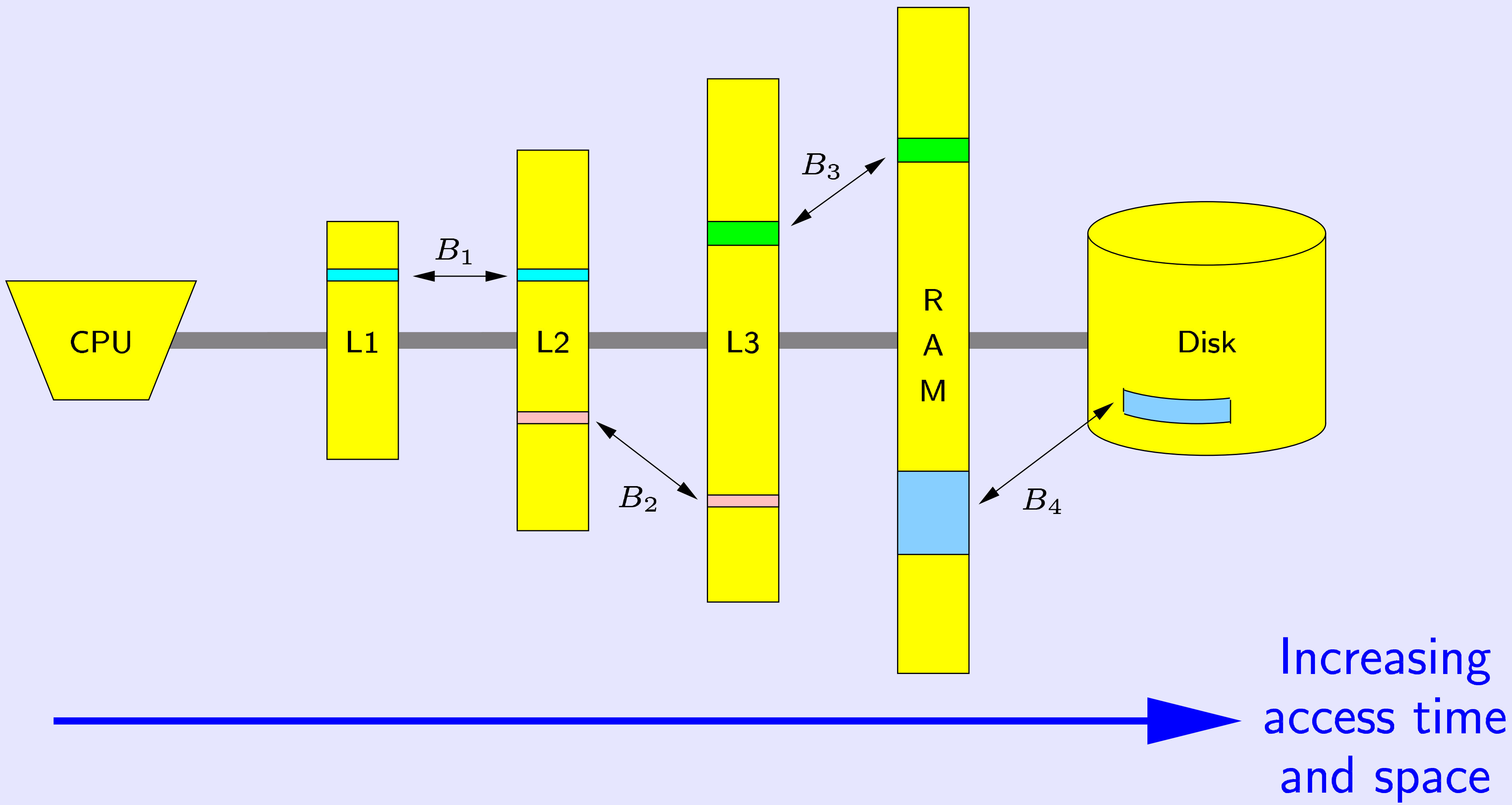
Workshop on Space-Conscious Algorithms, June 13, 2006, Bertinoro, Italy

Outline of Talk

- Cache-oblivious model
- Basic cache-oblivious techniques
- Cache-oblivious string algorithms
- Cache-oblivious string dictionaries
  - Cache-oblivious tries and blind tries

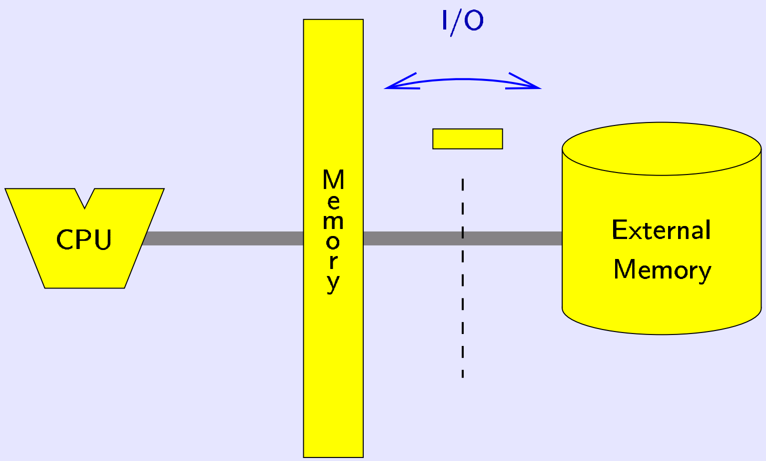
Hierarchical Memory Models

Hierarchical Memory



I/O Model

Aggarwal and Vitter 1988



N = problem size

M = memory size

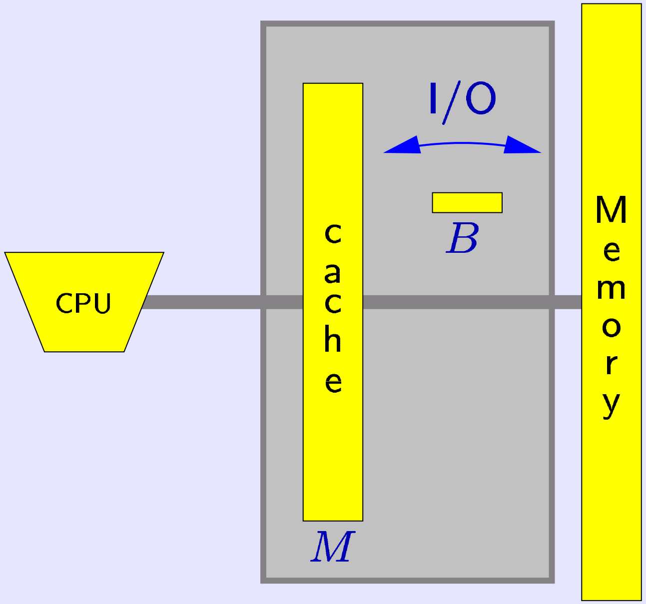
B = I/O block size

- One I/O moves B consecutive records from/to disk
- Complexity measure = number of I/Os

Ideal Cache Model — no parameters!?

Frigo, Leiserson, Prokop, Ramachandran 1999

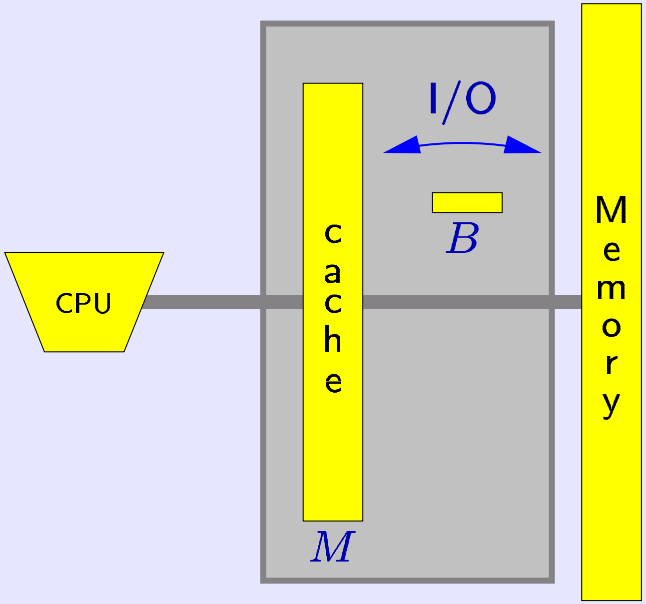
- Program with only one memory
- Analyze in the I/O model for
- Optimal off-line cache replacement strategy arbitrary B and M



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

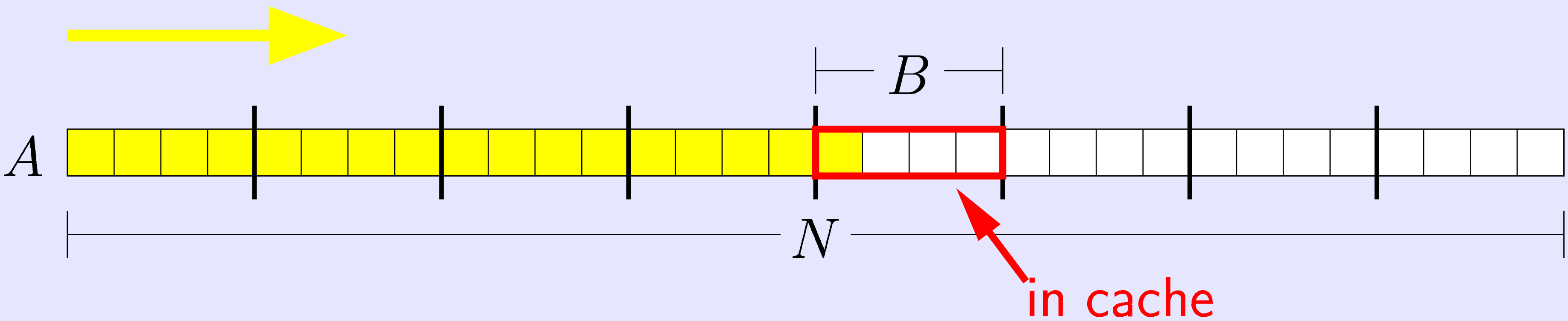


Advantages

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

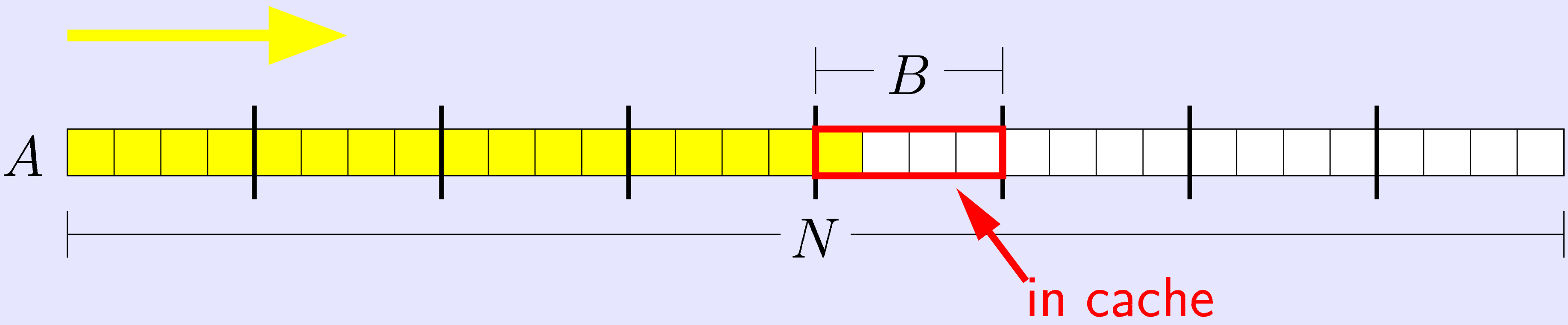
Cache-Oblivious Preliminaries

Cache-Oblivious Scanning



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

Cache-Oblivious Scanning

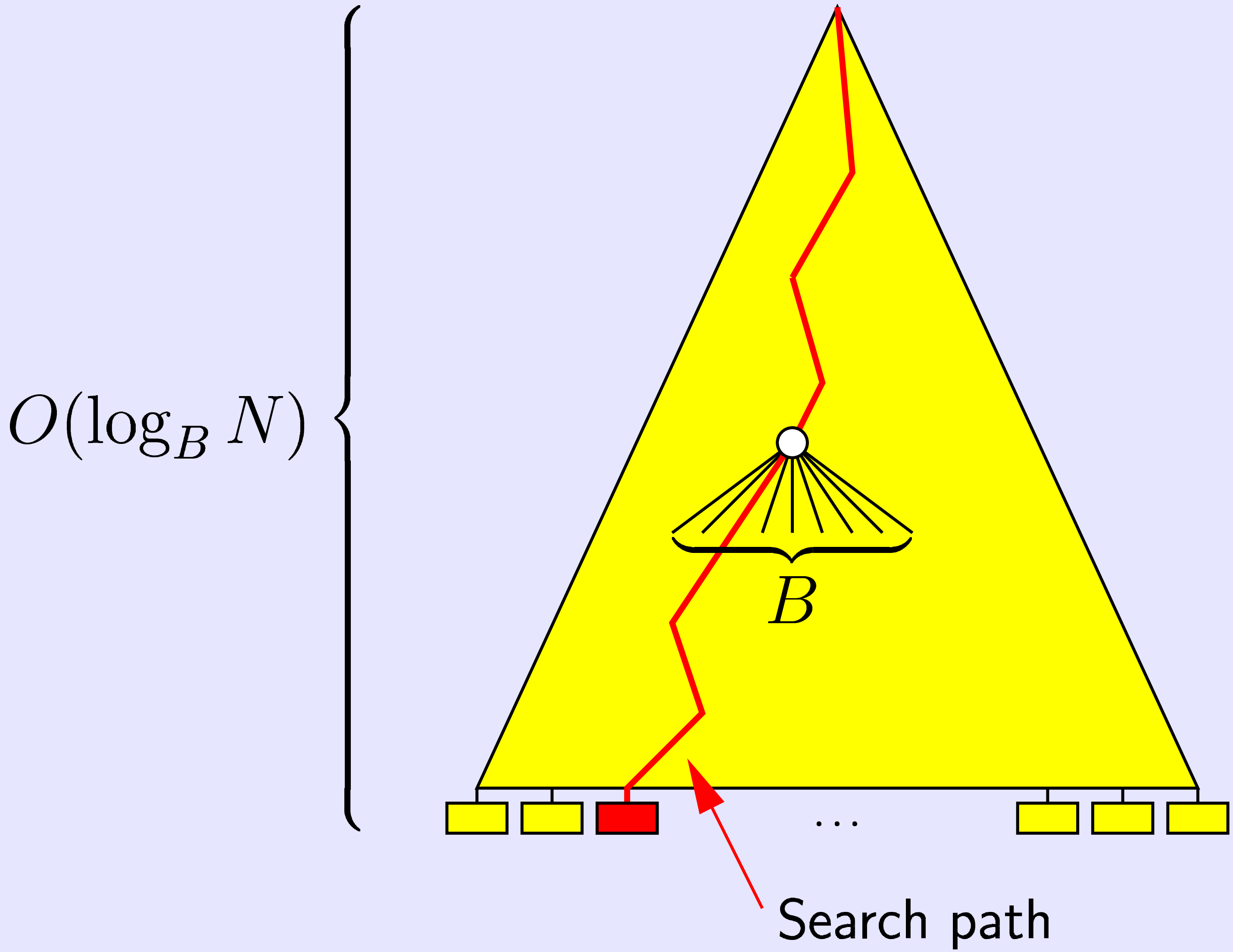


$$O\left(\frac{N}{B}\right) \text{ I/Os}$$

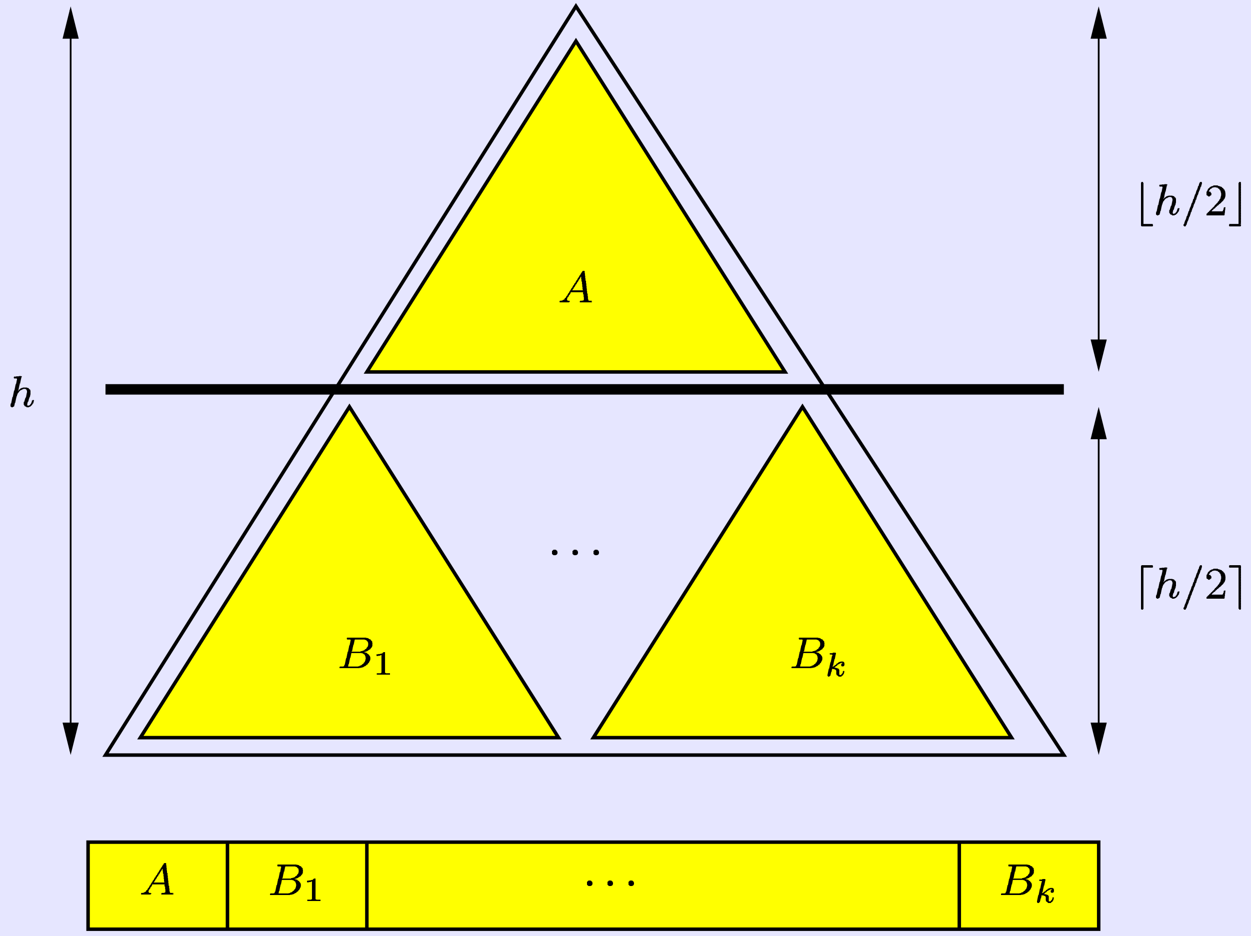
Corollary Cache-oblivious selection requires $O(N/B)$ I/Os

Hoare 1961 / Blum et al. 1973

Cache-Aware B-trees

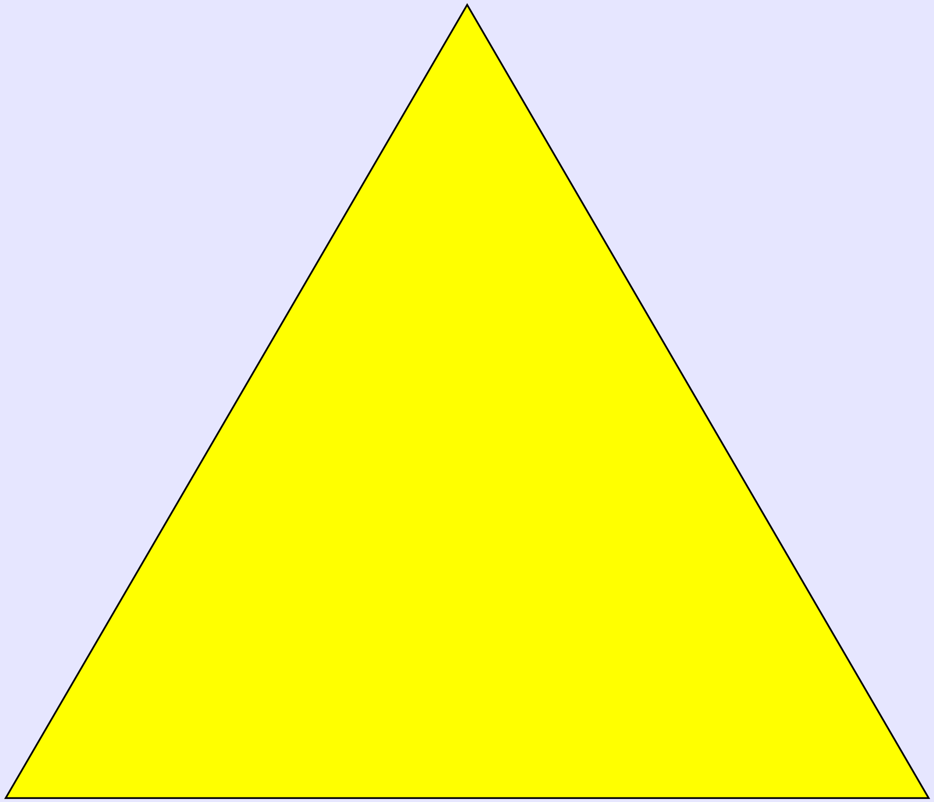


Static Cache-Oblivious B-Tree

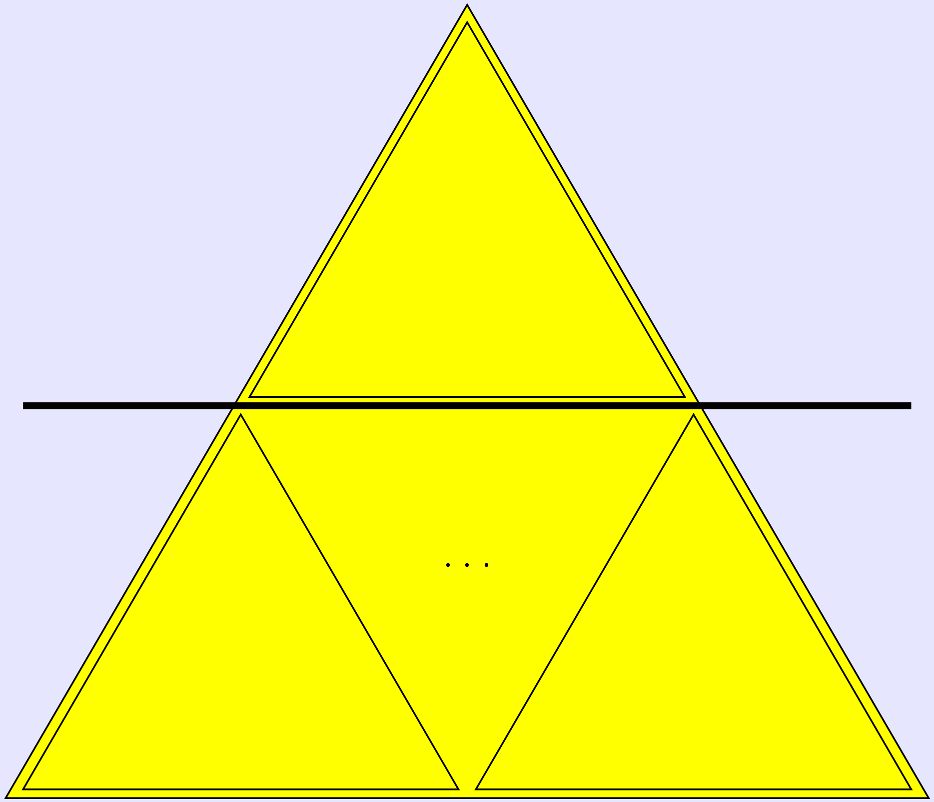


Recursive layout of binary tree $\equiv$ van Emde Boas layout

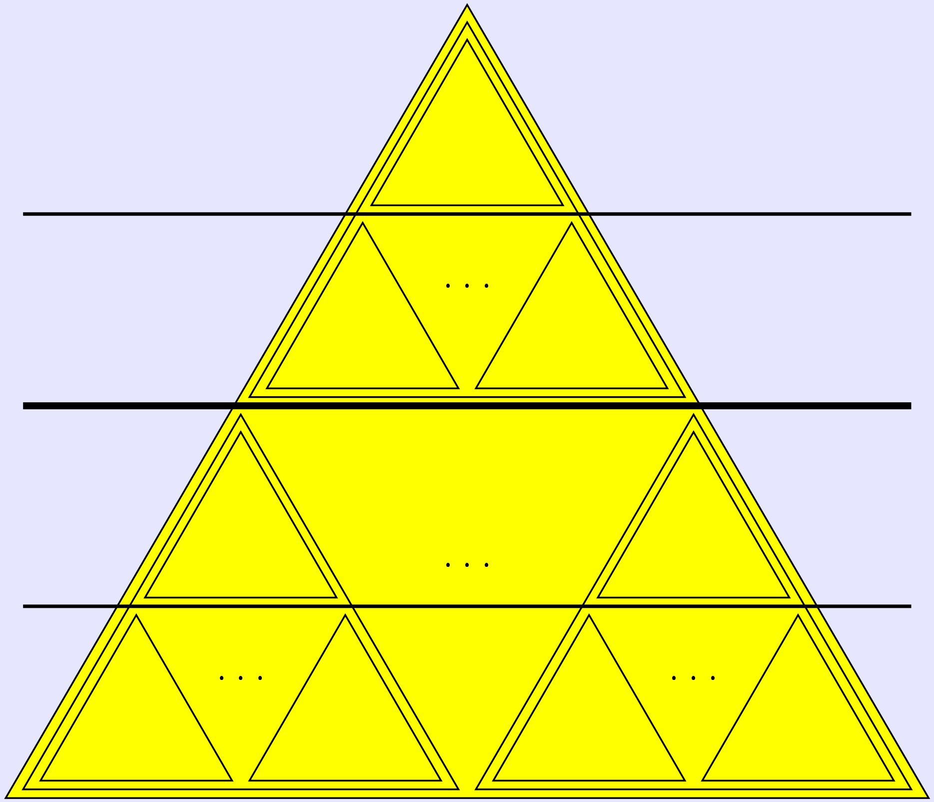
Static Cache-Oblivious B-Tree



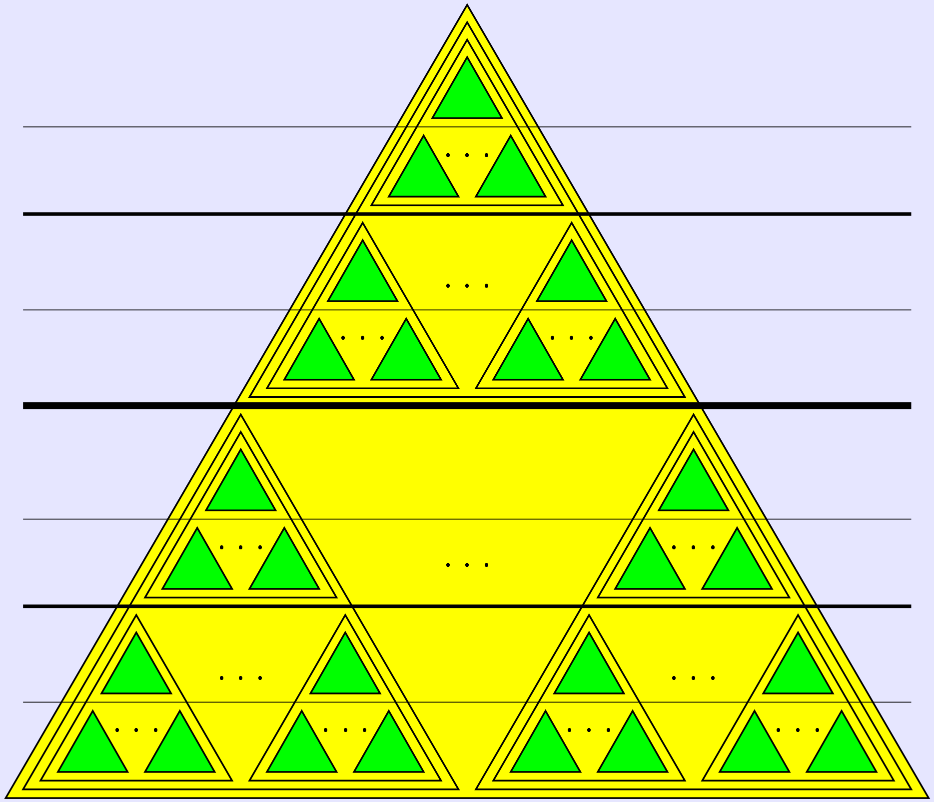
Static Cache-Oblivious B-Tree



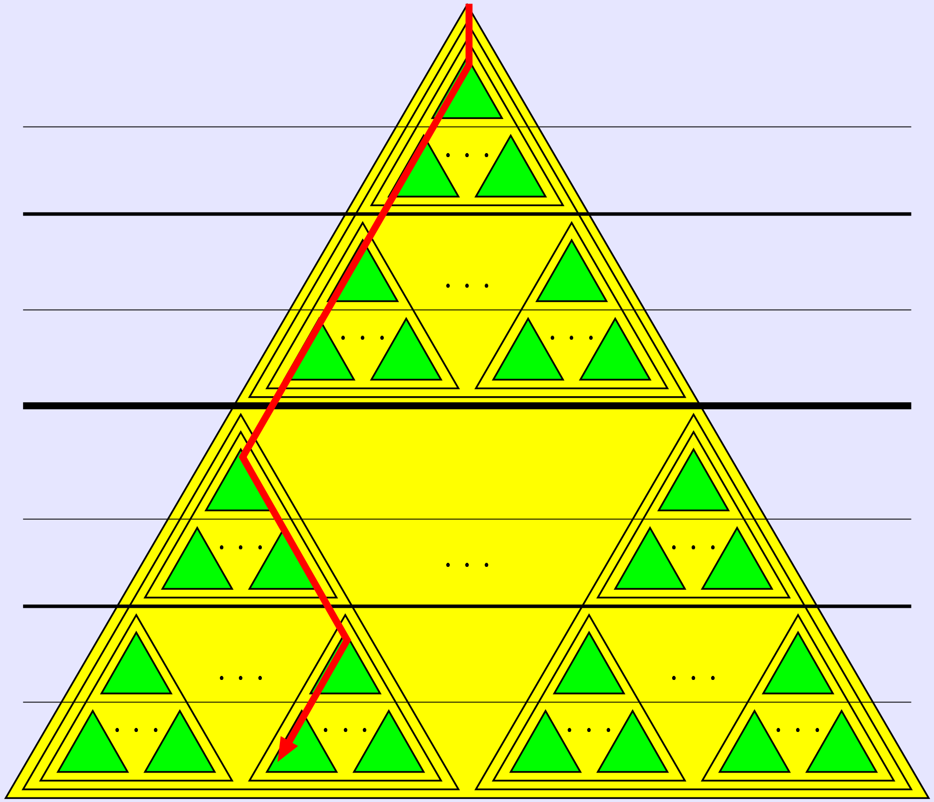
Static Cache-Oblivious B-Tree



Static Cache-Oblivious B-Tree



Static Cache-Oblivious B-Tree



- Each green tree has height between $(\log_2 B)/2$ and $\log_2 B$
- Searches visit between $\log_B N$ and $2\log_B N$ green trees, i.e. perform at most $4\log_B N$ I/Os (misalignment)

Summary Cache-Oblivious Tools

Scanning :	$O(N/B)$
B-tree searching :	$O(\log_B N)$
Sorting* :	$O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$

* requires a tall cache assumption $M \geq B^{1+\varepsilon}$

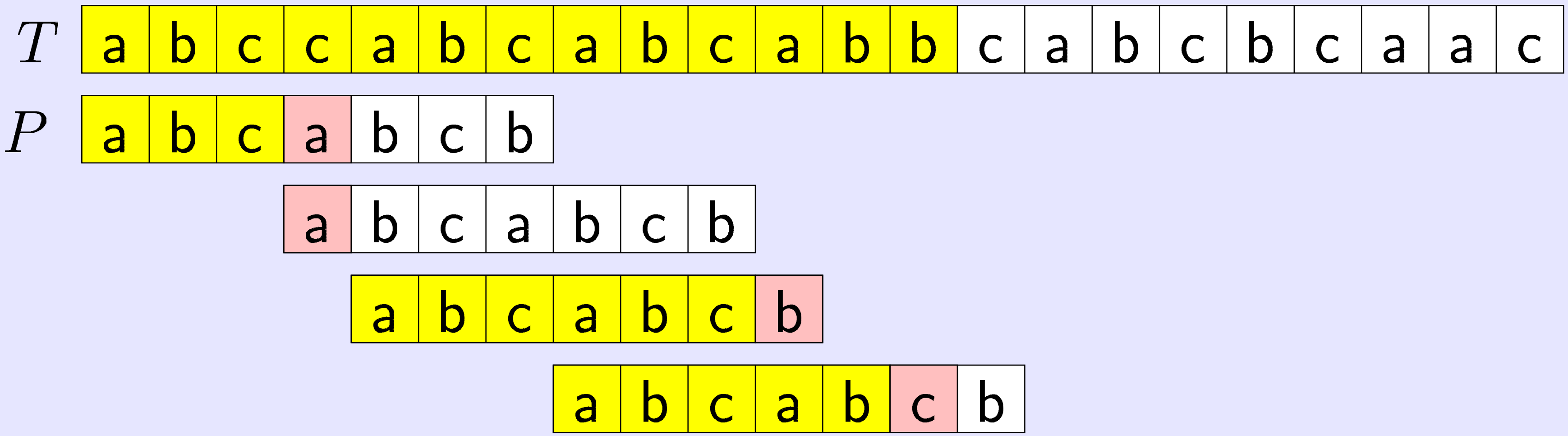
Frigo, Leiserson, Prokop, Ramachandran 1999

Brodal and Fagerberg 2002, 2003

Cache-Oblivious String Algorithms

Knuth-Morris-Pratt String Matching

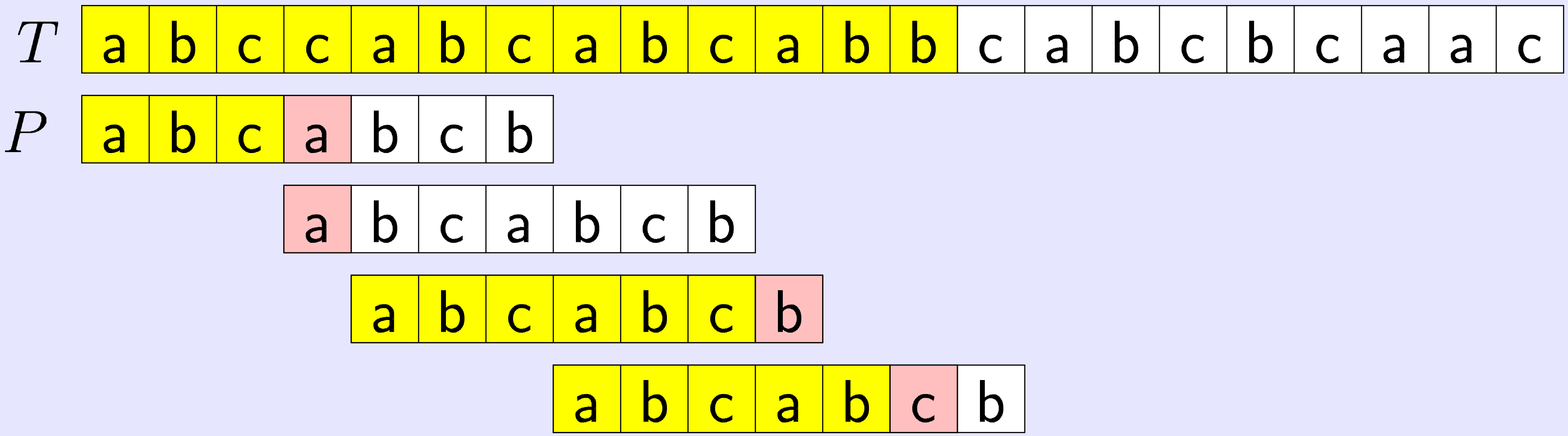
Knuth, Morris, Pratt 1977



- Time $O(|T|)$
- Scans text left-to-right
- Accesses the pattern (and failure function) like a stack

Knuth-Morris-Pratt String Matching

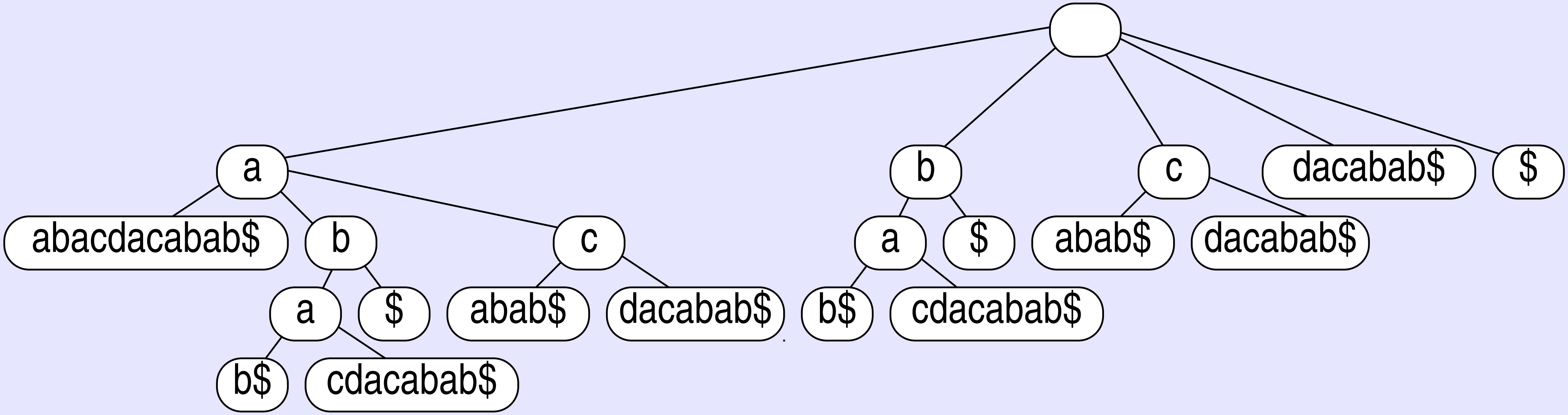
Knuth, Morris, Pratt 1977



- Time $O(|T|)$
- Scans text left-to-right
- Accesses the pattern (and failure function) like a stack
- KMP is cache-oblivious and uses $O(|T|/B)$ I/Os

Suffix Tree/Suffix Array Construction

Farach et al. 2000

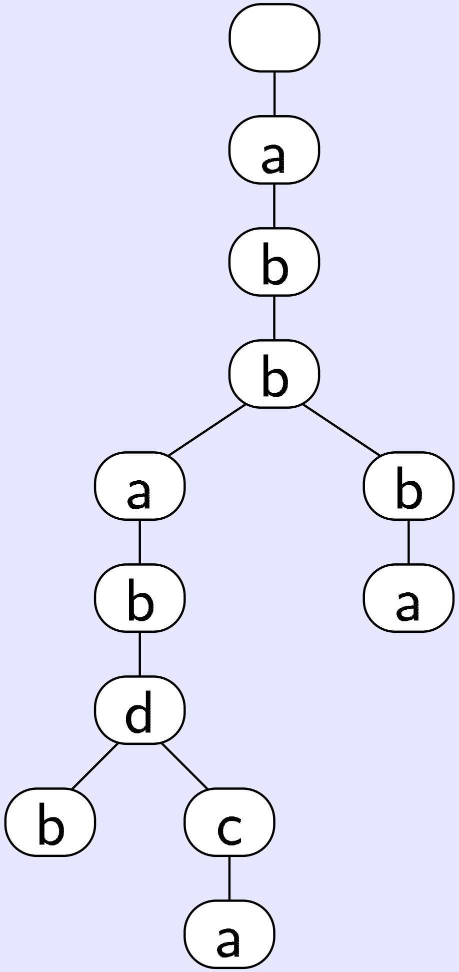


aabacdacabab$

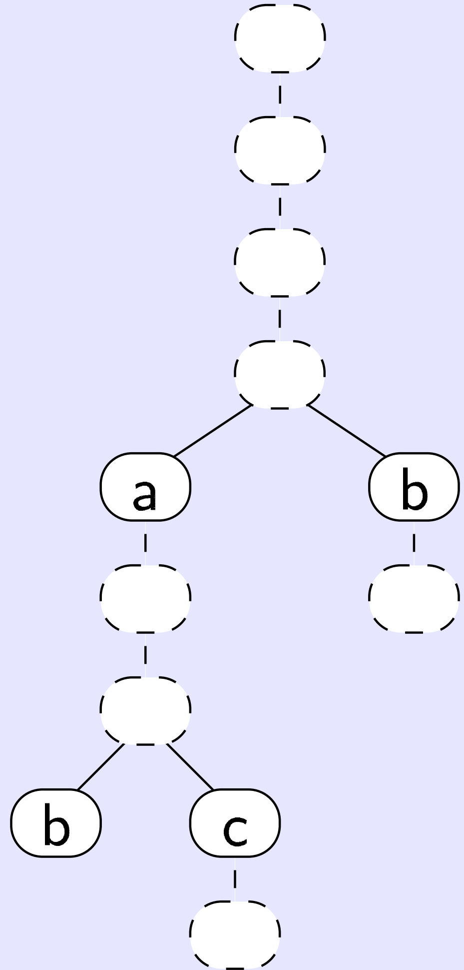
- Reduces to sorting, i.e. $\mathrm{Sort}(N)$ I/Os

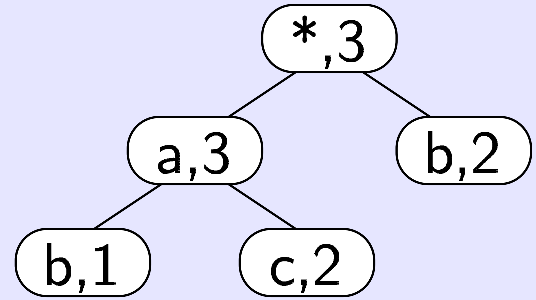
String Dictionaries

Tries vs Blind Tries



Trie





Blind trie

Searches take $O(|P|)$ time in internal memory for constant sized alphabets and $O(\log n + |P|)$ time for comparison based alphabets

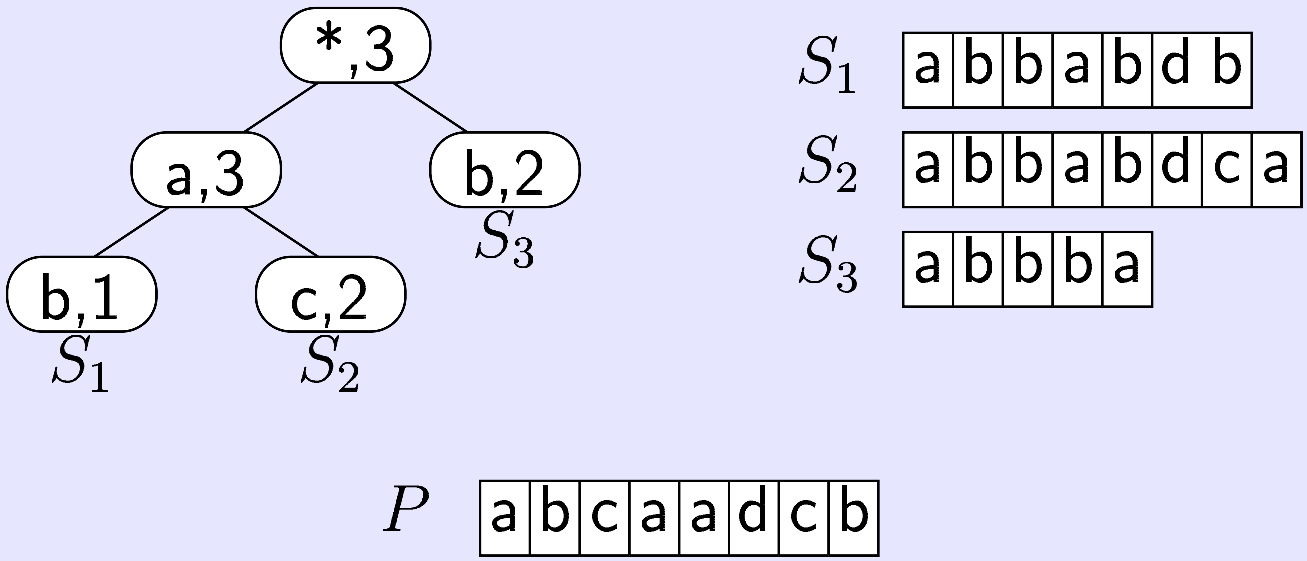
The Trouble Starts...

– Tries cannot be stored cache-aware to support top-down searches in $O(\log_B N + |P|/B)$ I/Os Demaine et al 2004

– Can construct suffix trees cache-obliviously using $O(\mathrm{Sort}(N))$ I/Os, but cannot search in it efficiently...

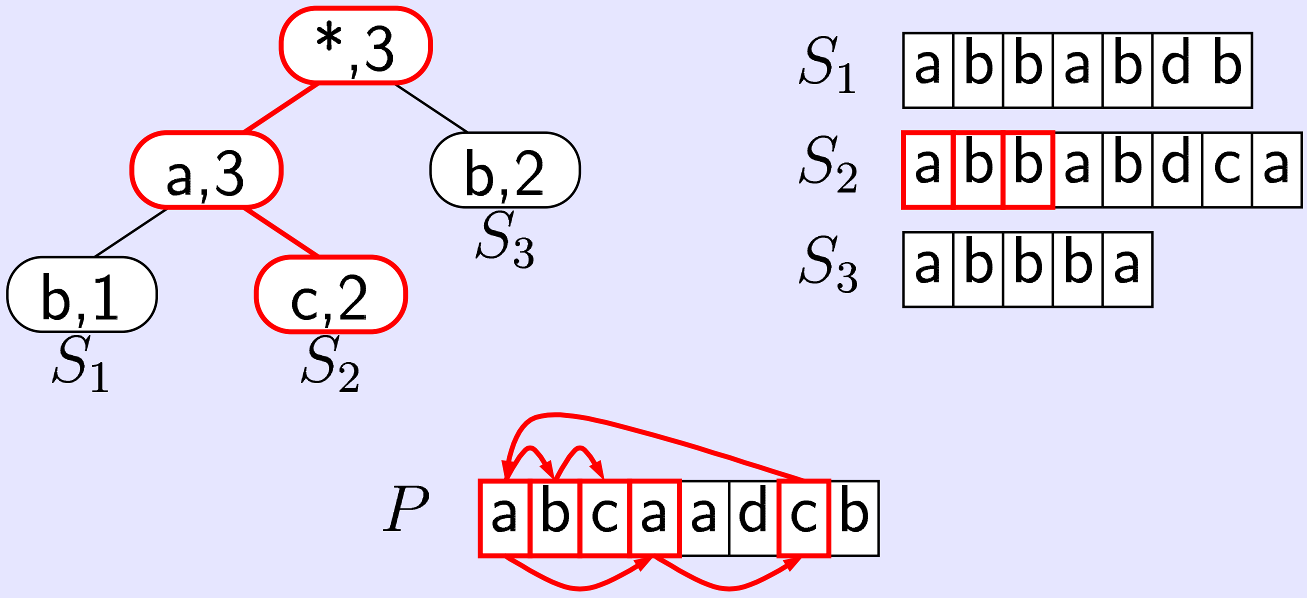
+ Cache-aware string B trees support searches in a set of strings in $O(\log_B n + |P|/B)$ I/Os Ferragina and Grossi 1999

String Dictionary



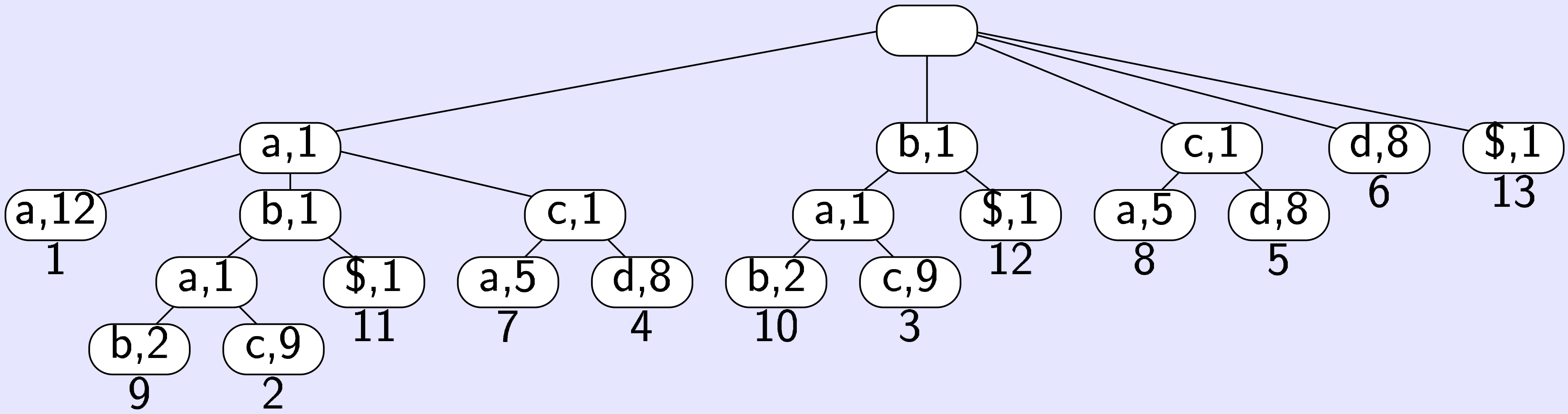
Queries: Search blind trie + Verify one string

String Dictionary



Queries: Search blind trie + Verify one string

Suffix Tree

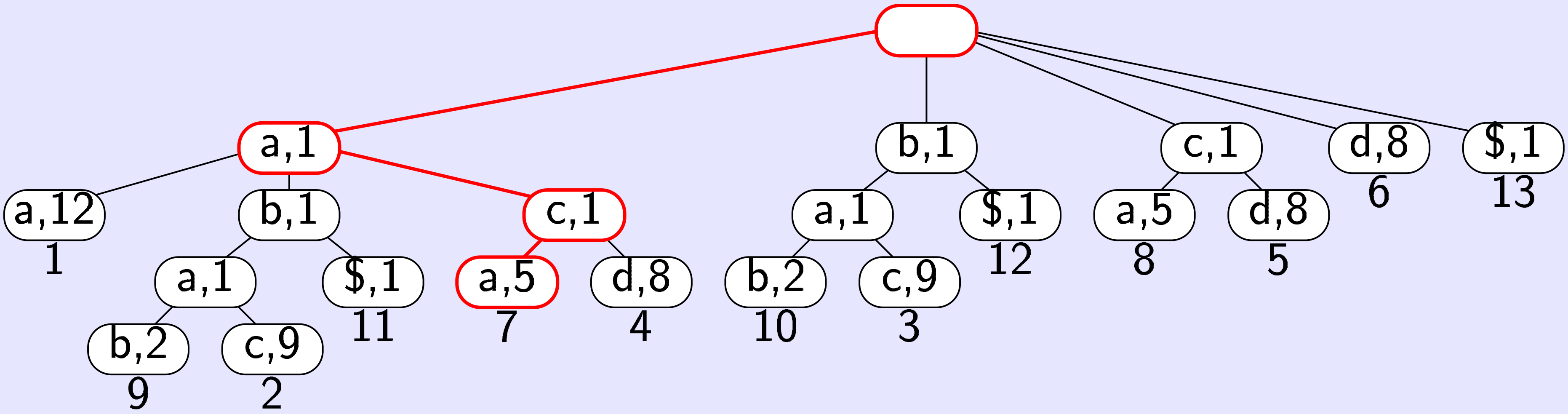


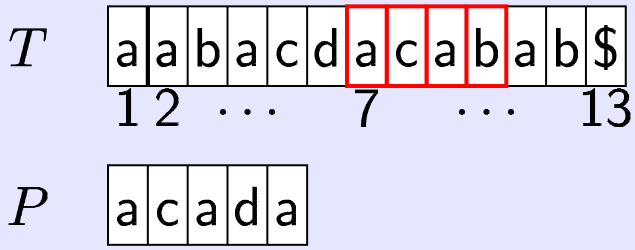
T	a	a	b	a	c	d	a	c	a	b	a	b	$
	1	2	⋯				7		⋯				13

P	a	c	a	d	a

Queries: Search blind trie + Verify one suffix

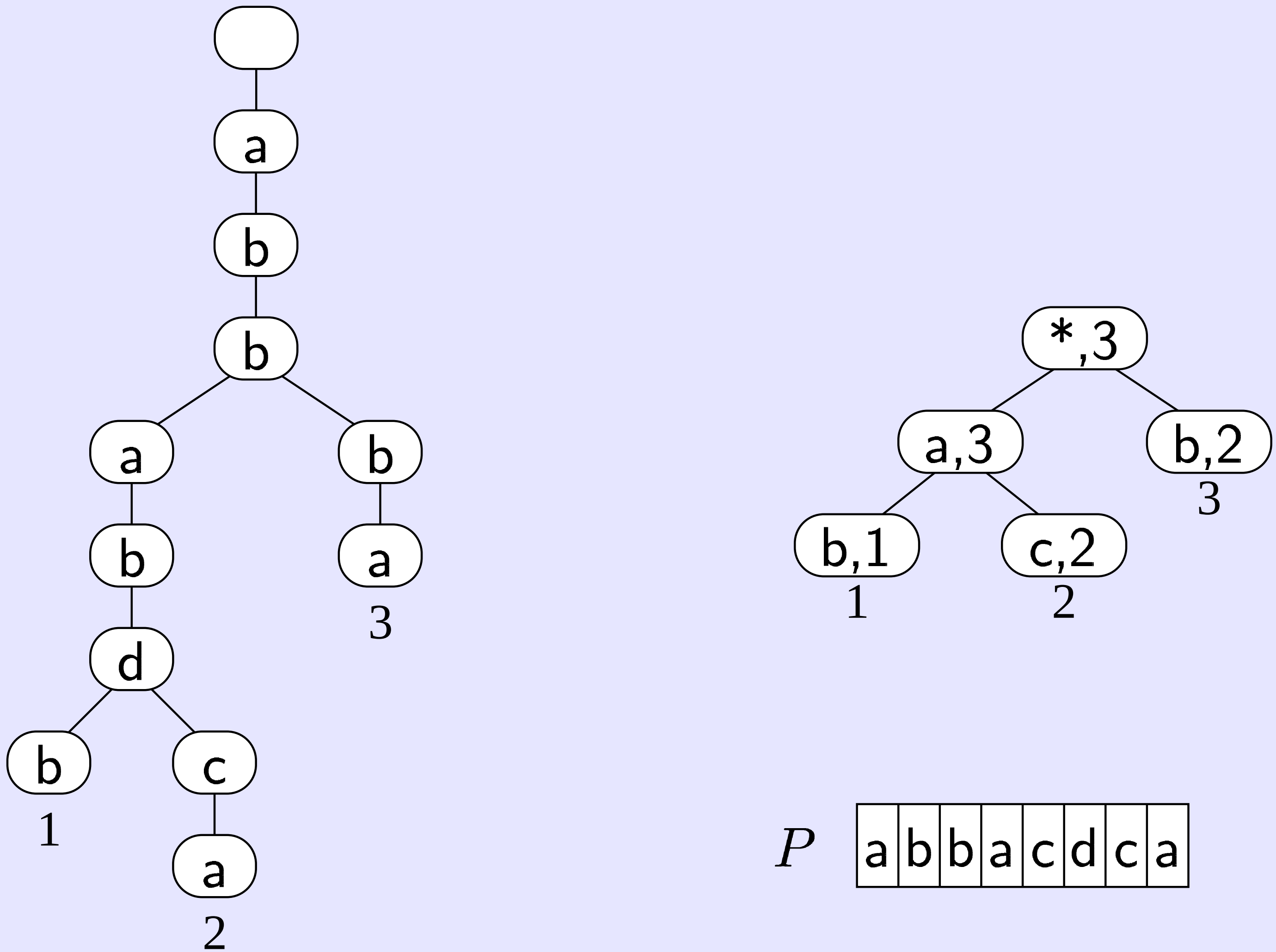
Suffix Tree





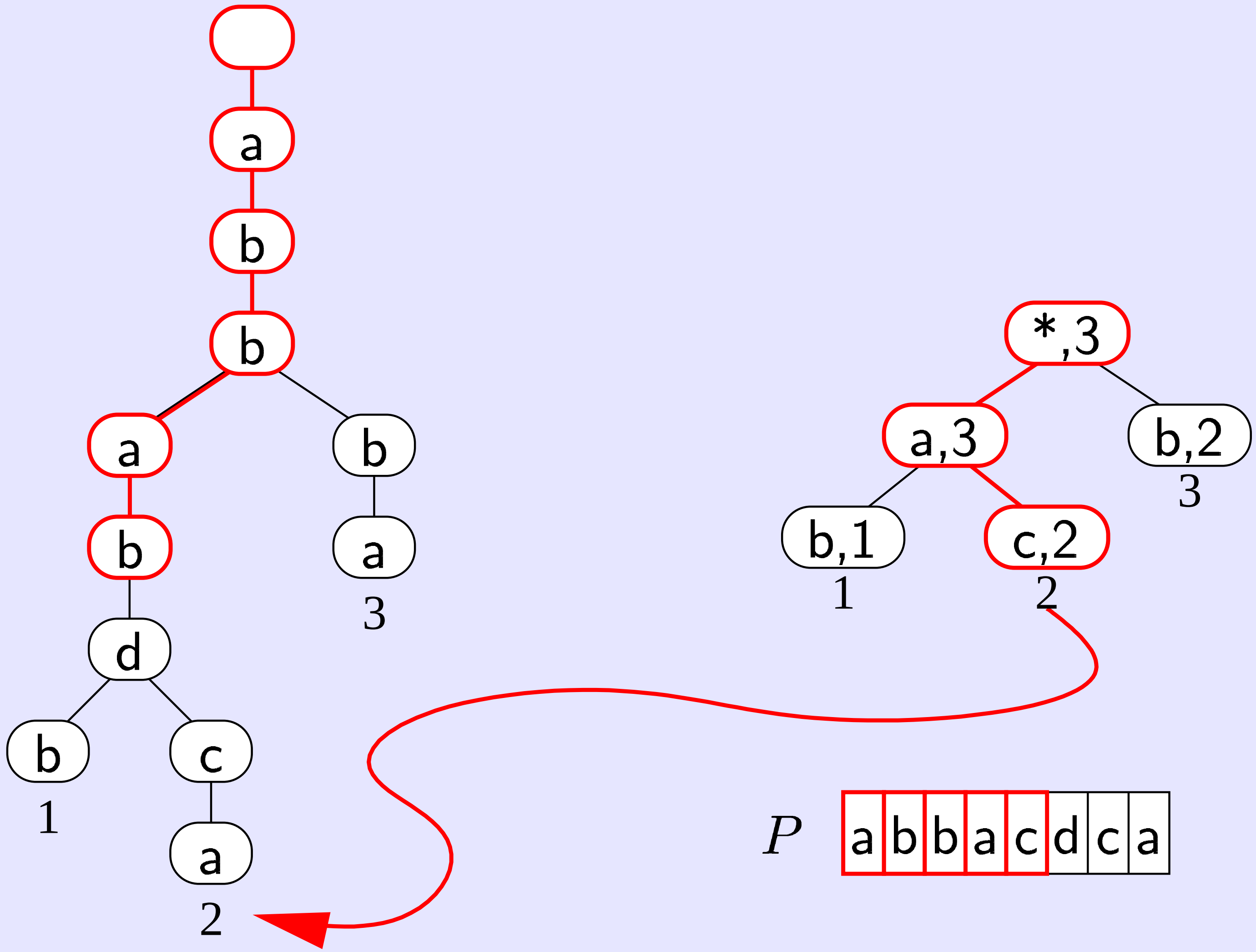
Queries: Search blind trie + Verify one suffix

Tries



Queries: Search blind trie + Verify prefix of one path

Tries



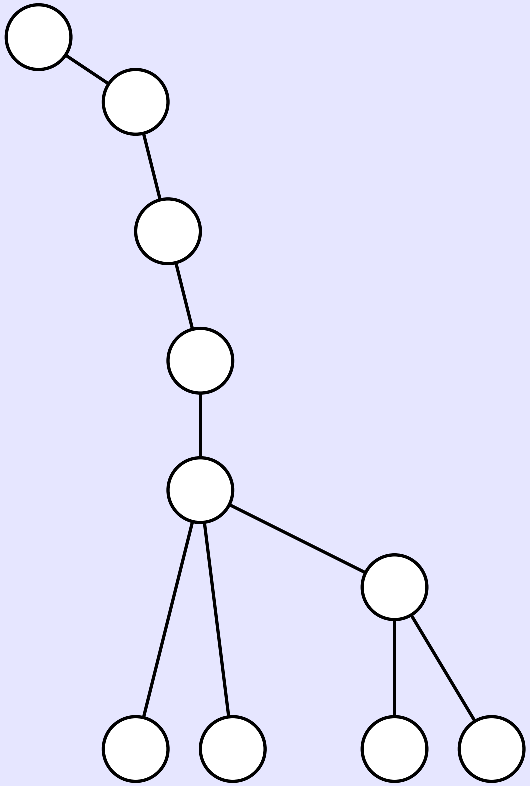
Queries: Search blind trie + Verify prefix of one path

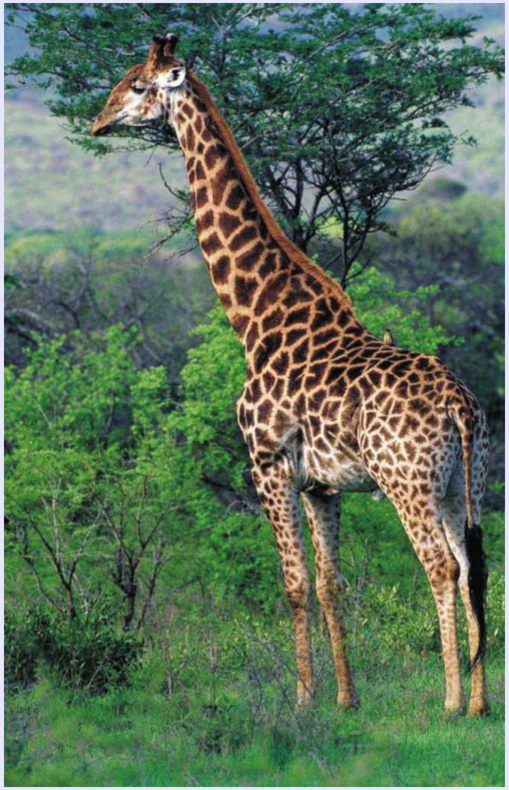
Verifying a Prefix of a Path in a Tree

Verifying Paths in Giraffe Trees is Easy

Definition

A tree is a giraffe tree if all root-to-leaf paths share at least half of the nodes of the tree (long neck)

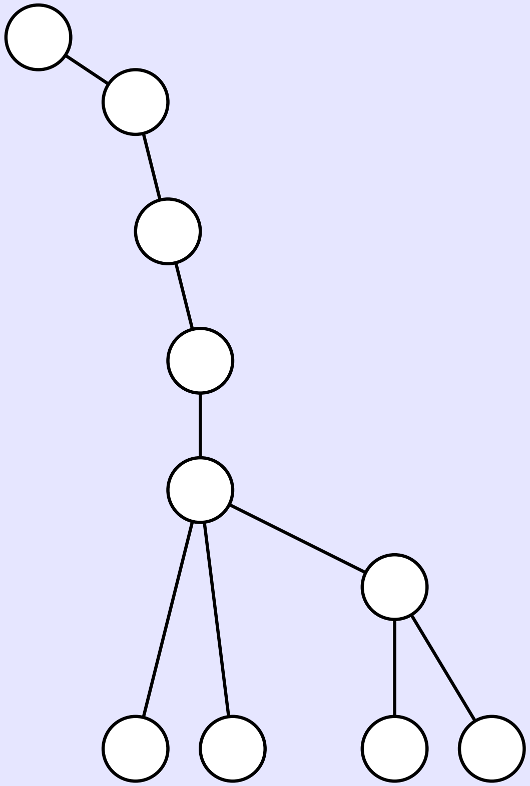


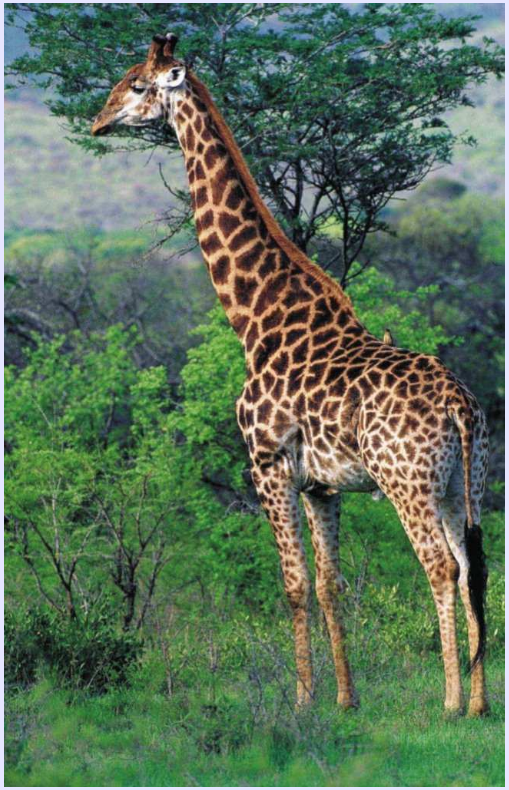


Verifying Paths in Giraffe Trees is Easy

Definition

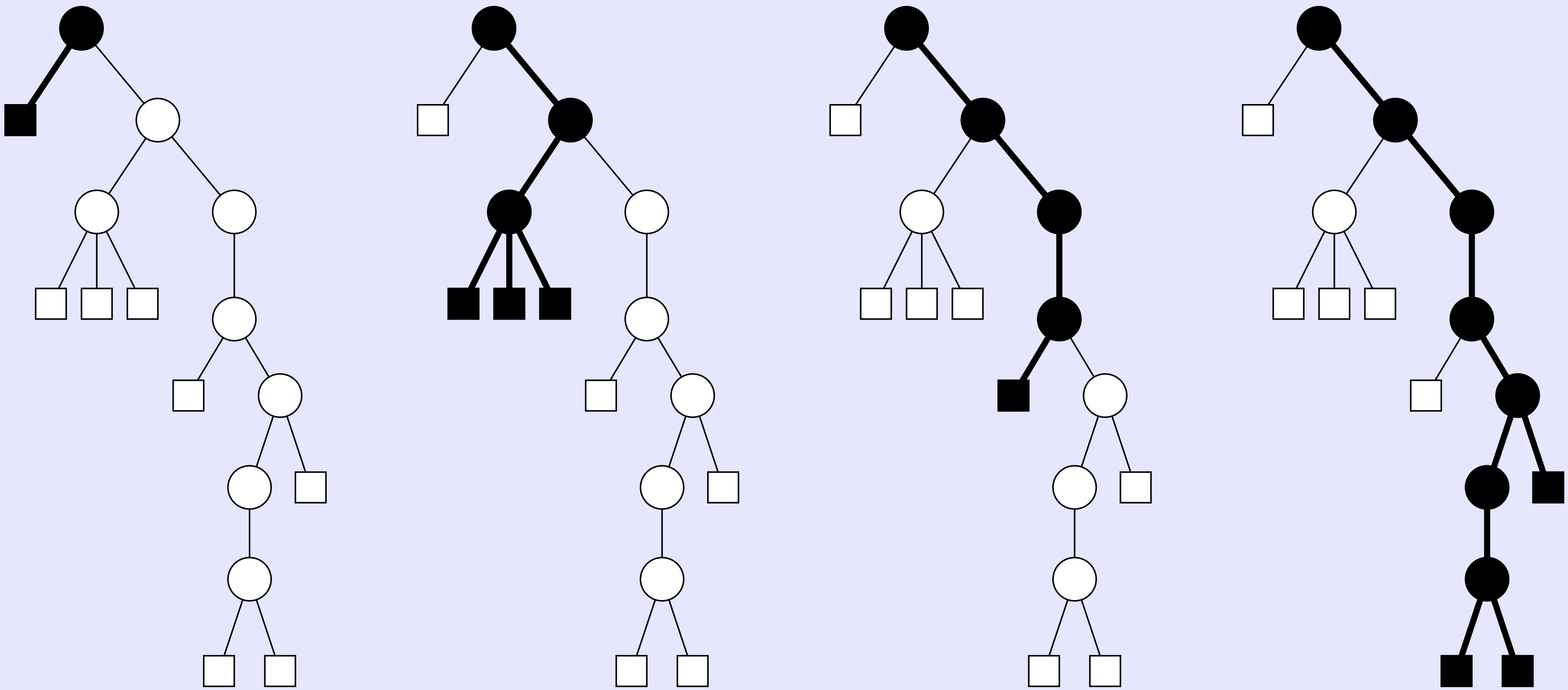
A tree is a giraffe tree if all root-to-leaf paths share at least half of the nodes of the tree (long neck)



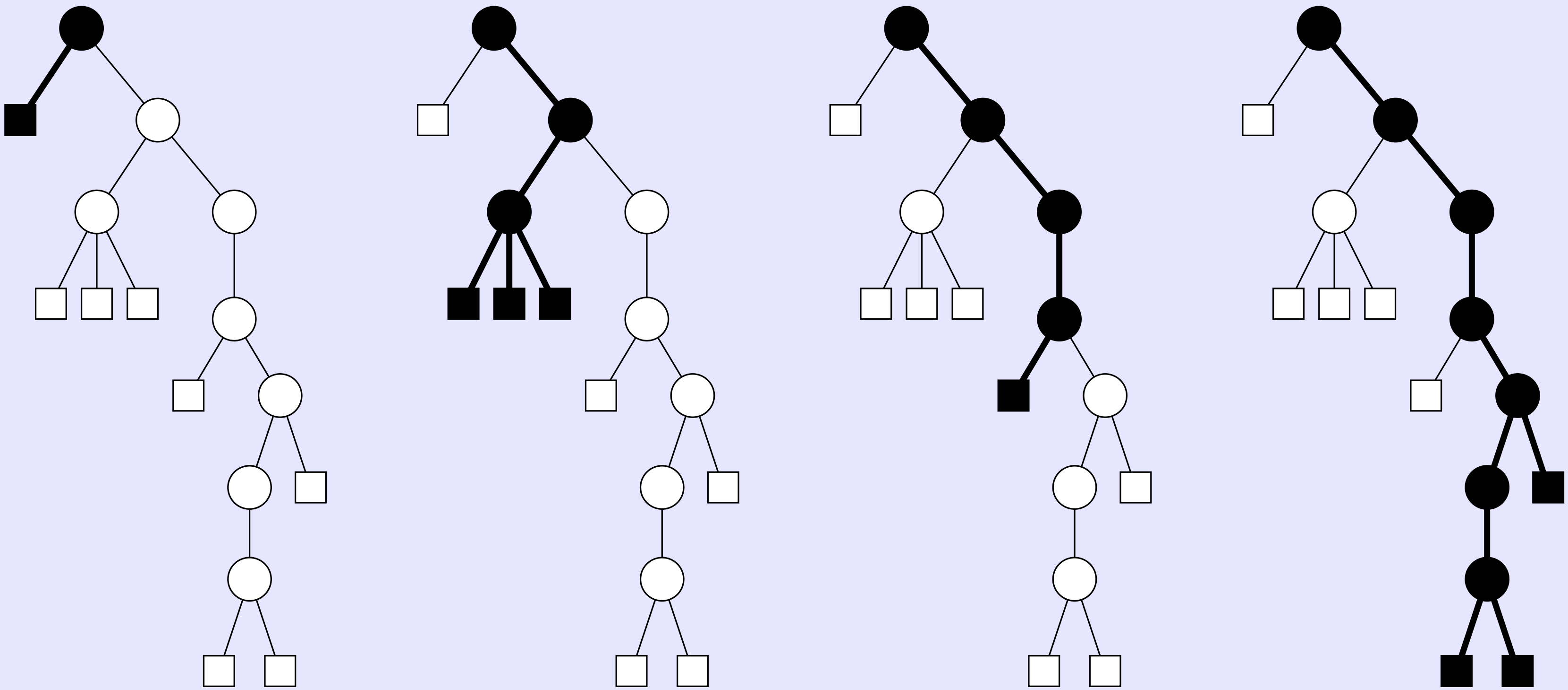


- A prefix of length p of a path in a giraffe tree using a BFS layout can be traversed in $O(p/B)$ I/Os

Giraffe Cover of a Tree



Giraffe Cover of a Tree



- Uses space $O(N)$ and can be constructed greedily from left-to-right using $O(N/B)$ I/Os by an Euler traversal of T
- BFS layout of each giraffe
- A prefix of length p of a path in a known giraffe can be traversed in $O(p/B)$ I/Os

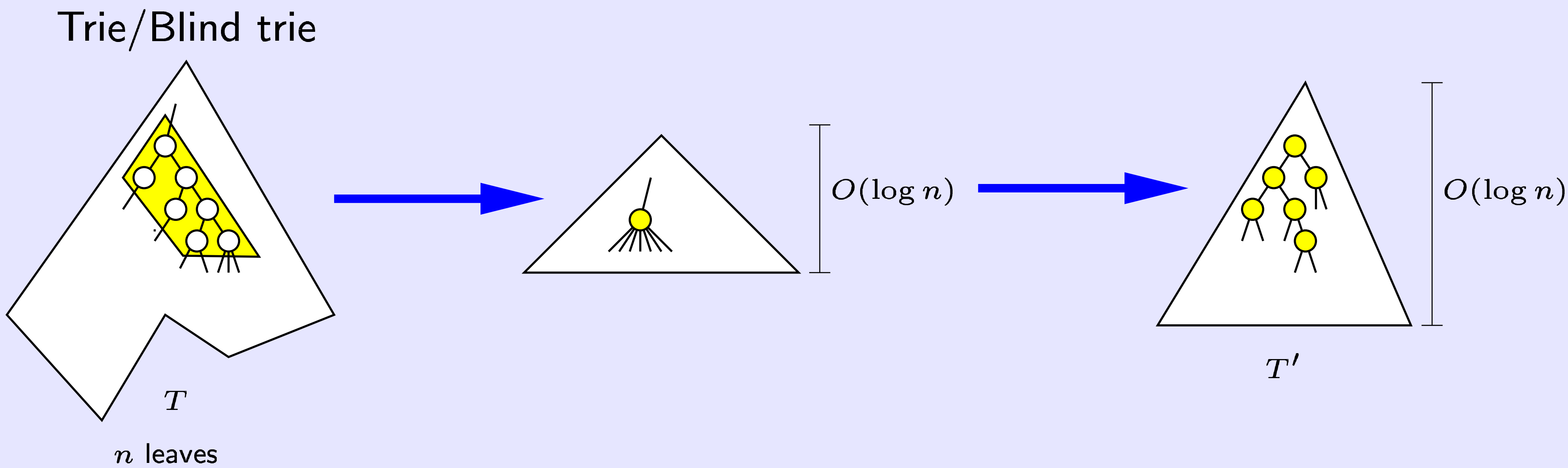
Summary so far...

$$\left.\begin{array}{l}\text{String dictionary search} \\ \text{Suffix tree search} \\ \text{Trie search}\end{array}\right\} \text{ reduce to blind trie search}$$

Query : Blind trie search + $O\left(1 + \frac{|P|}{B}\right)$ I/Os

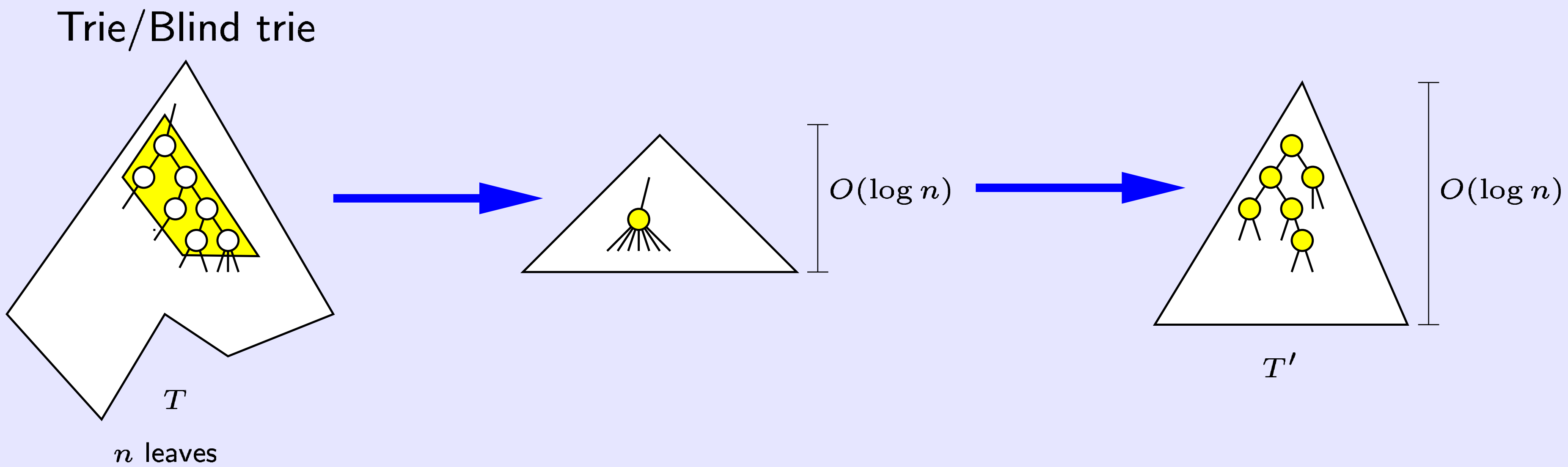
Cache-Oblivious (Blind) Tries

Cache-Oblivious (Blind) Tries



- Partition input trie T into components (generalization of heavy paths)
- T' = collapse components in T into high degree nodes and replace by weight balanced trees
- Apply van Emde Boas layout out to T'

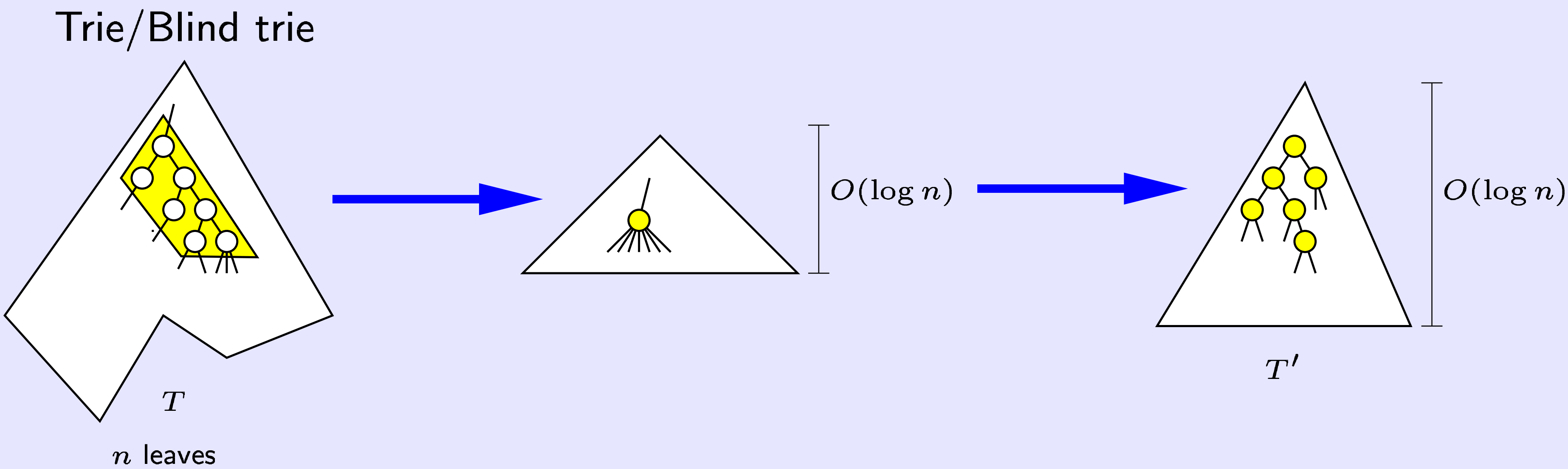
Cache-Oblivious (Blind) Tries



- Partition input trie T into components (generalization of heavy paths)
- T' = collapse components in T into high degree nodes and replace by weight balanced trees
- Apply van Emde Boas layout out to T'

Search: $O(\log_B n)$ I/O

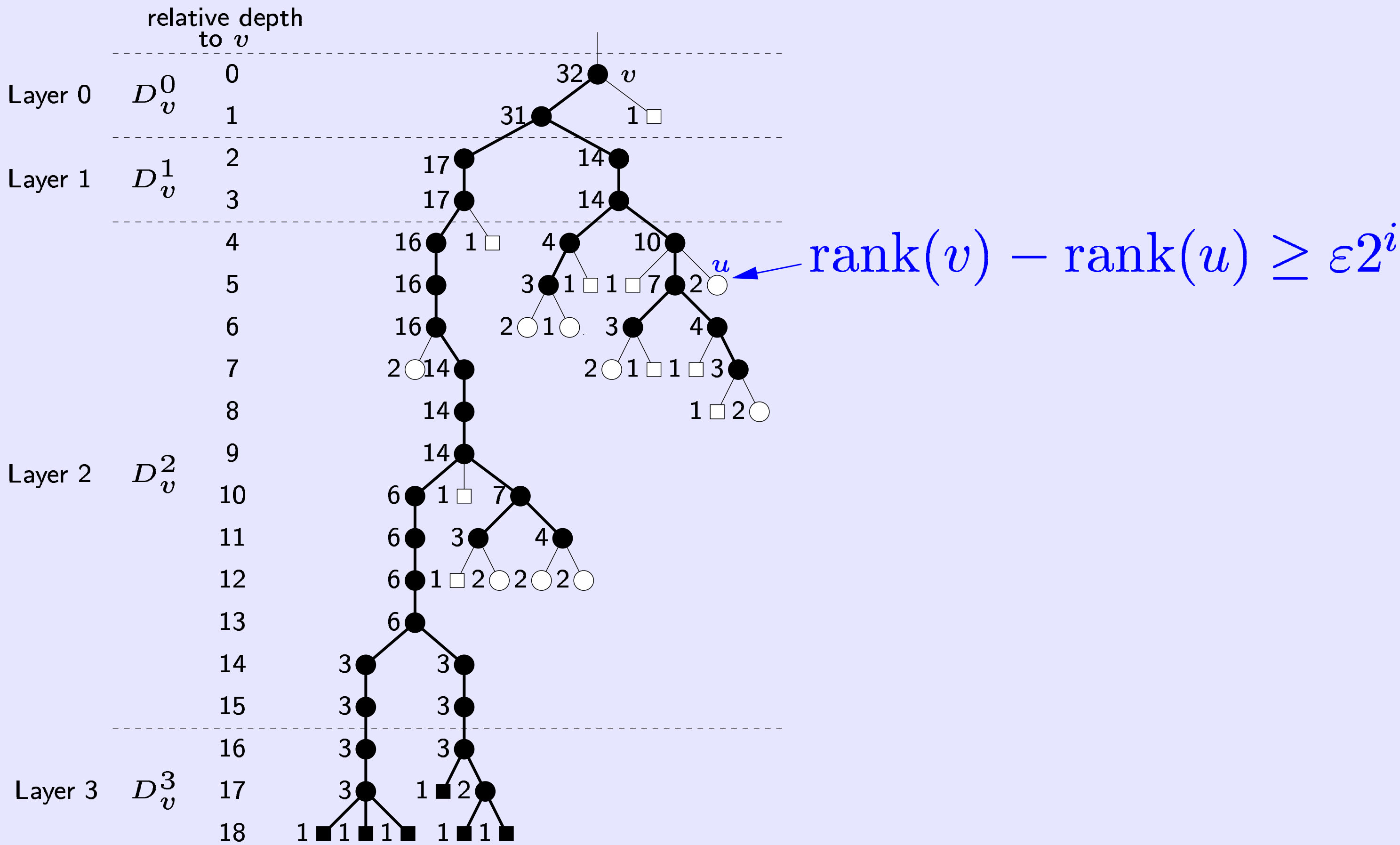
Cache-Oblivious (Blind) Tries



- Partition input trie T into components (generalization of heavy paths)
- T' = collapse components in T into high degree nodes and replace by weight balanced trees
- Apply van Emde Boas layout out to T'

Search: $O(\log_B n)$ I/O — ignoring searching inside components

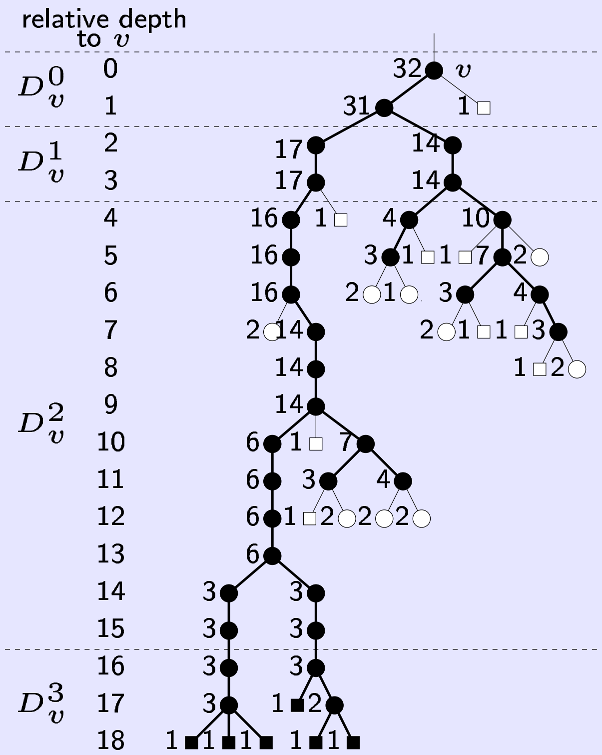
Decomposition into Components



$$
\begin{aligned}
D_v^0 &= \{u \in T_v \mid \mathrm{rank}(u) = \mathrm{rank}(v) \ \wedge\ \mathrm{depth}(u) - \mathrm{depth}(v) < 2^{2^0}\} \\
D_v^i &= \{u \in T_v \mid \mathrm{rank}(v) - \mathrm{rank}(u) < \varepsilon 2^i \\
&\qquad \wedge\ 2^{2^{i-1}} \leq \mathrm{depth}(u) - \mathrm{depth}(v) < 2^{2^i}\}
\end{aligned}
$$

Storing and Searching Components

- Store each layer D_v^i separately
- Make a giraf-decompostition of D_v^i
- For D_v^i have a blind trie of size $O(2^{\varepsilon 2^i})$ (using BFS layout) to select the right giraffe-tree
- Search: D_v^i search the blind trie + search in one giraffe-tree
- Distribute $D_v^0, D_v^1, D_v^2, \ldots$ in the van Emde Boas layout of T'



- Analysis:
  - Search in blind trie for D_v^{i+1} dominated by the matched characters in D_v^i
  - Space in van Emde Boas layout for a subtree of size k becomes $O(k^3)$

Cache-Oblivious Tries

There exists a cache-oblivious trie supporting prefix queries in

$$O(\log_B |n| + |P|/B) \text{ I/Os,}$$

where P is the query string, and n is the number of leaves in the trie.

It can be constructed in $O(\mathrm{Sort}(N))$ time, where N is the total number of characters in the input.

The space required is $O(N)$.

The structure assumes $M \geq B^{2+\delta}$.

Conclusion

- A string dictionary (trie data structure) was presented that supports queries in $O(\log_B n + |P|/B)$ I/Os. The data structure uses $O(N)$ space and can be constructed using $O(\mathrm{Sort}(N))$ I/Os.

- Lookahead in the query string is crucial (both cache-aware and cache-oblivious)

- A giraffe cover is a simple construction allowing topdown path traversals in a tree using $O(|P|/B)$ I/Os

Open problems

- Prove a lower bound trade-off between the number of I/Os required for a query and the lookahead used

- Implementation: compare with string B-trees, tries, ternary trees, different trie layouts, ...

The End