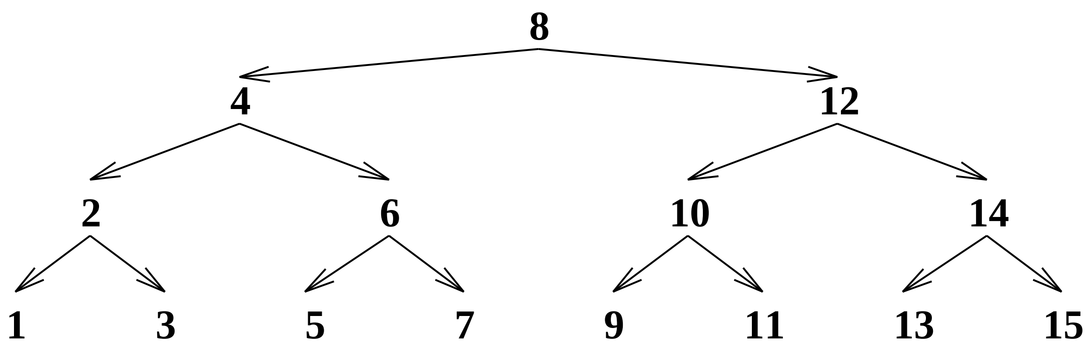
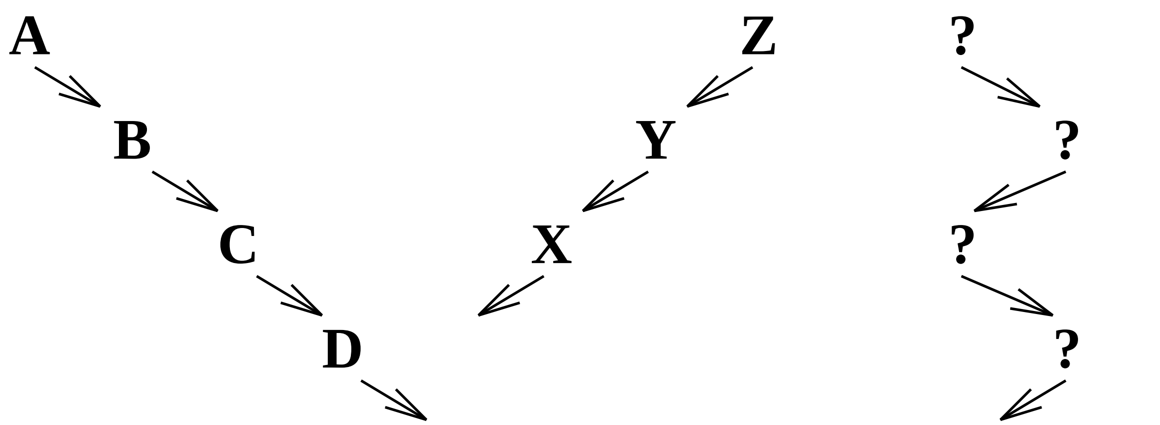
The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

Degeneracy in Binary Trees

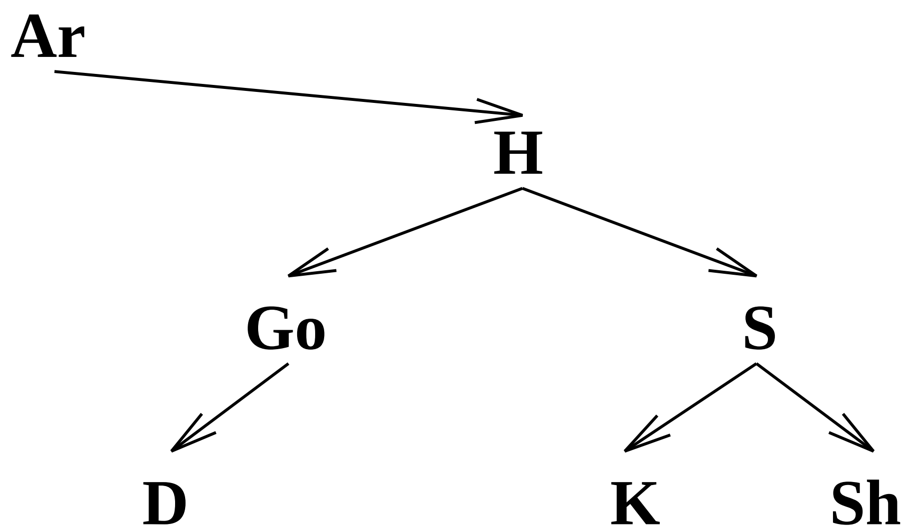
1. Ideal ($h = \log_2 N$)



2. Degenerate ($h = N$)



3. Actual ($h = 2 \ln N$, expected)



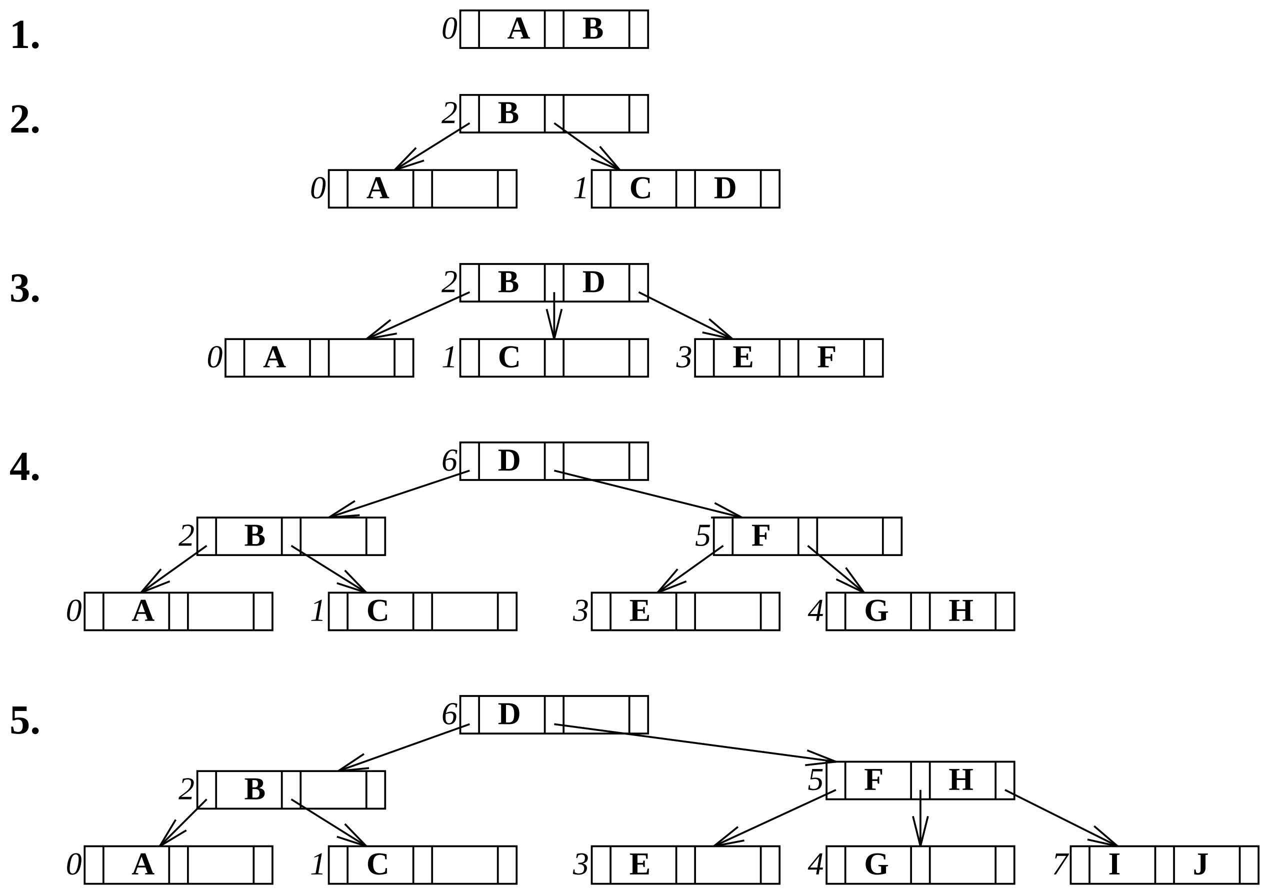
We need *balance*.

Can we satisfy these rules? Given a maximum fanout, f:

- The root has at least two subtrees (unless it is a leaf).
- Every node (apart from root or leaf) has s subtrees, $f/2 \leq s \leq f$.
- All leaves are on the same level.

B-Tree

$f = 3$

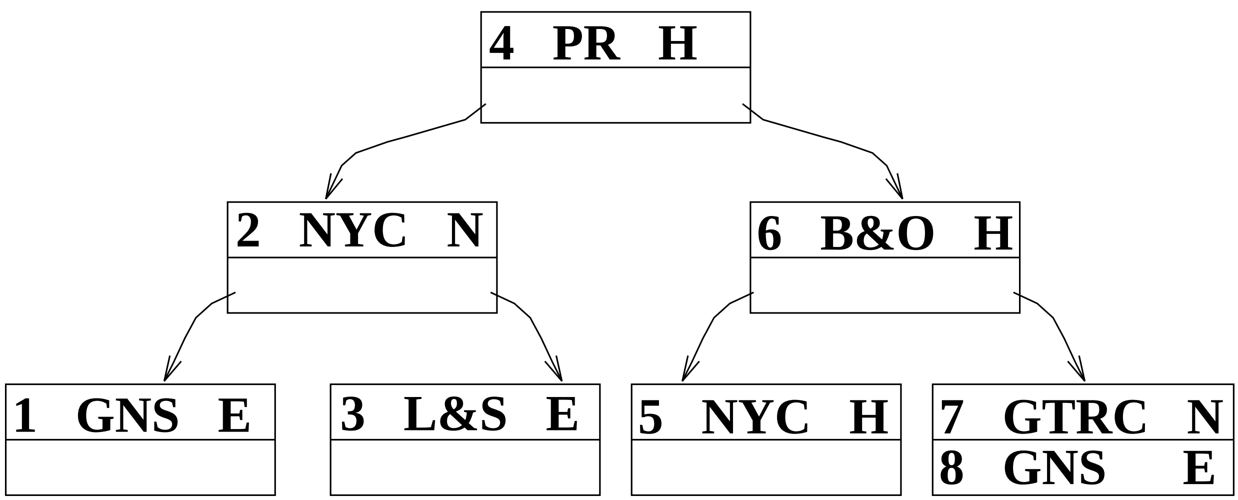


Java

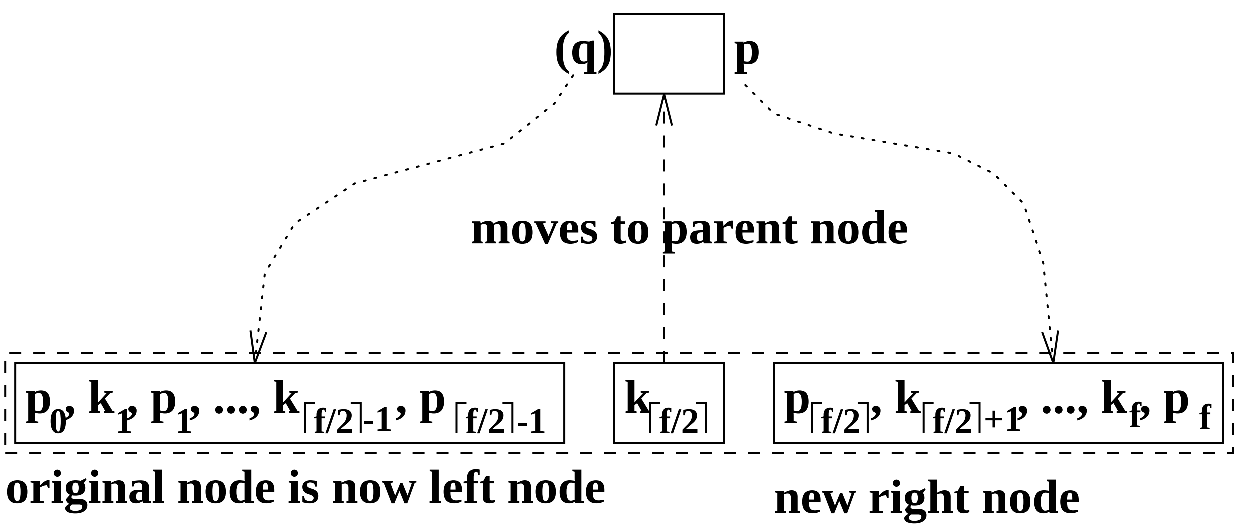
```
RandomAccessFile btreed = new RandomAccessFile("bTree","r");

btreed.seek(loc*(f*4 + (f-1)*recordSize));
```

B-Tree with Full Records



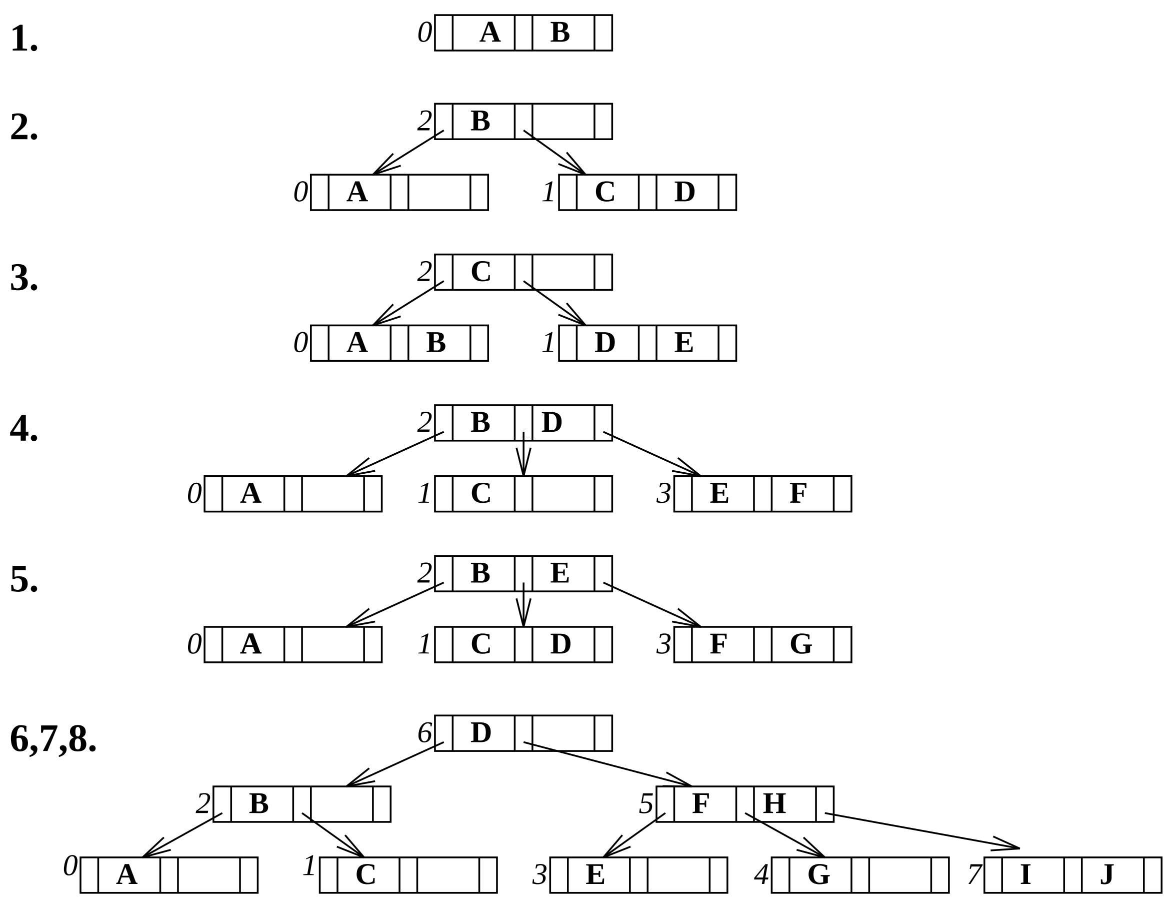
Splitting a Node of a B-tree



B-Tree Improvements

1. B*-Tree (2→3 splits: at least ~ 2/3 full)

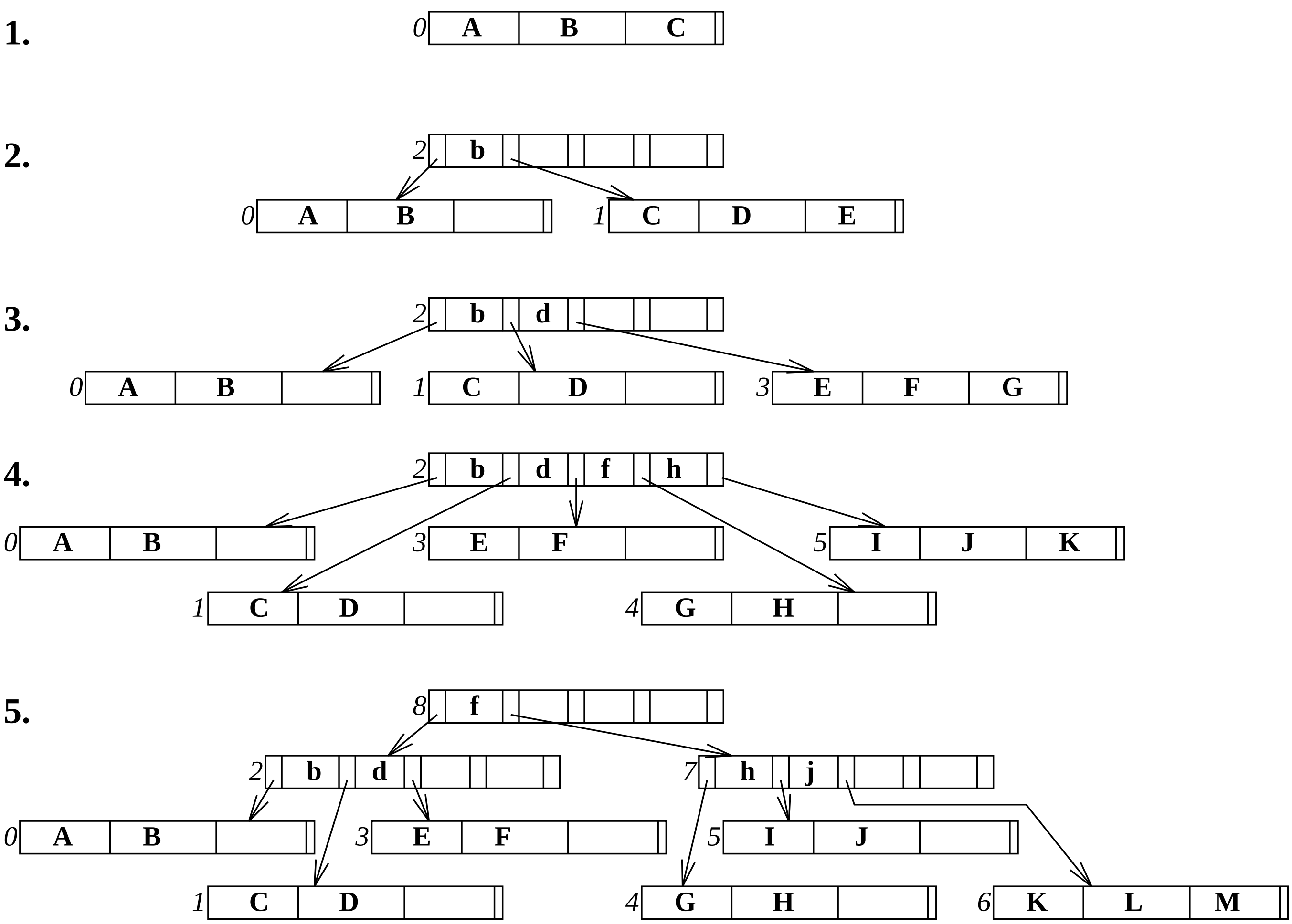
$2f$ keys in 2 nodes and parent; put in 3 nodes, moving up keys $\lceil 2f/3 \rceil$ and $\lceil 4f/3 \rceil$ $\qquad f = 3$



2. Inhomogeneous B-Tree

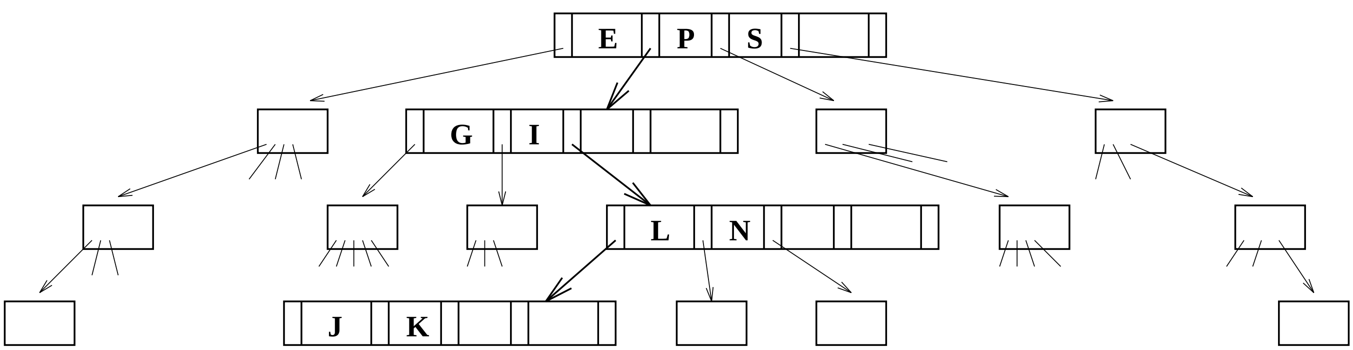
Decrease height (hence cost) by increasing fanout (but without changing bytes/block), for a given number of records. $f = 5$

a (3 bytes) is the key for record **A** (10 bytes) etc.



B-Trees:
Activity, Volatility, and Symmetry

	lo	**hi**
Volatility	$\surd$	$\surd$
Symmetry	$\surd$	$\times$
Activity	$\surd$	$\surd$:



If $|\text{RAM}| = \mathcal{O}(\log N)$ then retrieve each needed page only once for any level of activity. E.g., above, 4 buffers can fit into RAM.

Cost Analysis of B-Trees

	Best Case		Worst Case	
Level	No. Nodes	No. Records	No. Nodes	No. Records
0	1	$f-1$	1	1
1	f	$f-1$	2	$\lceil\frac{f}{2}\rceil-1$
2	f^2	$f-1$	$2\lceil\frac{f}{2}\rceil$	$\lceil\frac{f}{2}\rceil-1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$h-1$	f^{h-1}	$f-1$	$2\lceil\frac{f}{2}\rceil^{h-2}$	$\lceil\frac{f}{2}\rceil-1$

(best case)

$$N=(f-1)(1+f+f^2+..+f^{h-1})=f^h-1$$

(worst case)

$$N=1+2(f_2-1)(1+f_2+..+f_2^{h-2})=2*f_2^{h-1}-1$$

where $f_2=\lceil\frac{f}{2}\rceil$.

$$\log_f N+1\leq h\leq 1+\log_{\lceil\frac{f}{2}\rceil}(N+1)/2$$

Cost Analysis of ϕ-ary Trees

Level	No. Nodes	No. Accesses
0	1	1
1	ϕ	2ϕ
2	ϕ^2	$3\phi^2$
:	:	:
$h-1$	ϕ^{h-1}	$h\phi^{h-1}$

total nodes $= 1 + \phi + \phi^2 + .. + \phi^{h-1} = \frac{\phi^h - 1}{\phi - 1}$

total accesses $= 1 + 2\phi + 3\phi^2 + .. + h\phi^{h-1}$

Note that $(k+1)\phi^k = \frac{\mathrm{d}}{\mathrm{d}\phi}\phi^{k+1}$

So total accesses $= \frac{\mathrm{d}}{\mathrm{d}\phi}\phi\frac{\phi^h - 1}{\phi - 1}$

$$= \frac{h\phi^{h+1} - (h+1)\phi^h + 1}{(\phi - 1)^2}$$

average accesses = total accesses / total nodes

Special cases:

- binary tree ($\phi = 2$)

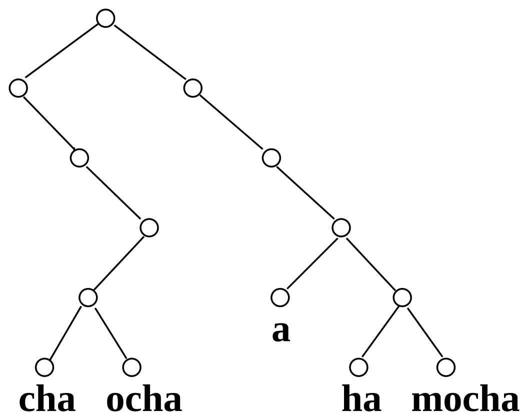
  average accesses $= \frac{(h-1)2^h + 1}{2^h - 1} \approx h - 1$

- large ϕ

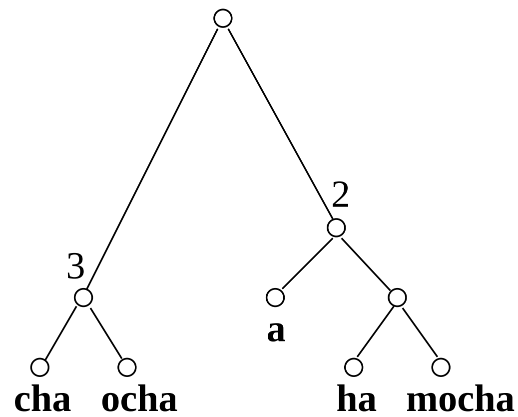
  average accesses $\approx \frac{h\phi^{h-1}}{\phi^{h-1}} = h$

So expected depth of search is $h - 1$ for binary tree; and h, effectively, for anything else.

Text and PATRICIA



1) Truncated Trie



2) PATRICIA Trie

Sample "text":

`mocha :` 1110110101101111011000111110100011100001

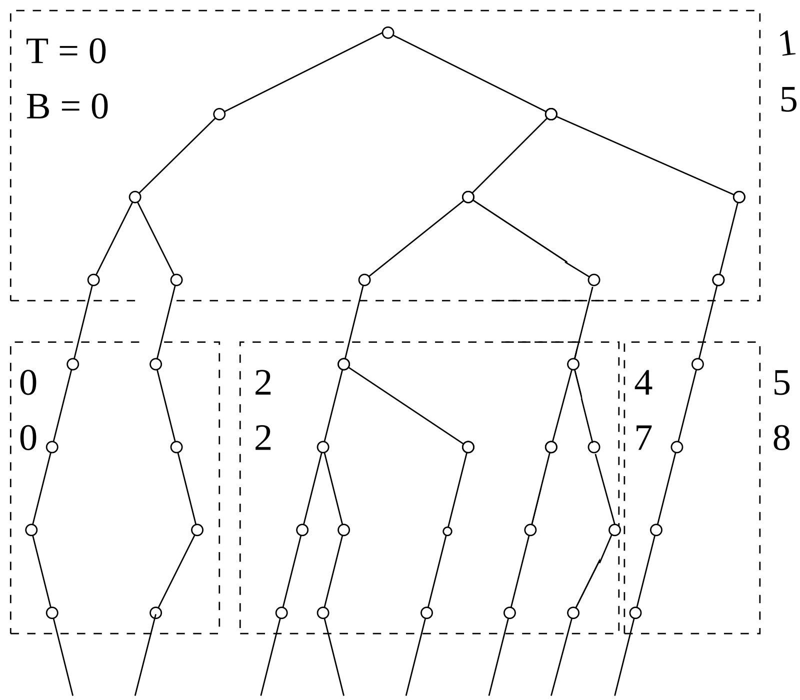
with "starts" every eight bits.

Note that pointers alone, to each byte of a 100 Mb text, would occupy 4 times the size of the text: text indexes are big.

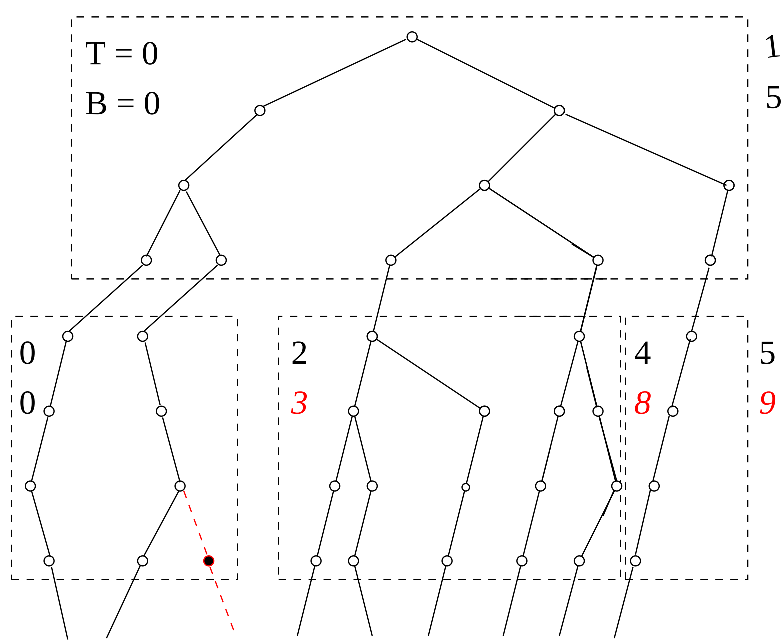
Using PATRICIA tries, stored 1 bit per node, and simple tricks, a byte-by-byte text index can be reduced to 2.7 times the text size. (Word-by-word: 0.7 times.)

Not only any substring but also arbitrary regular expressions can be found.

Volatility—Updating Tries

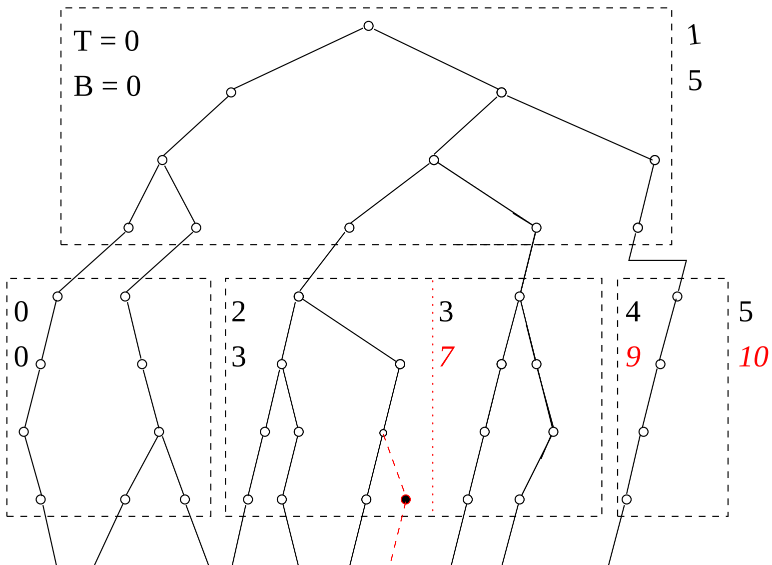


Adding 00101111



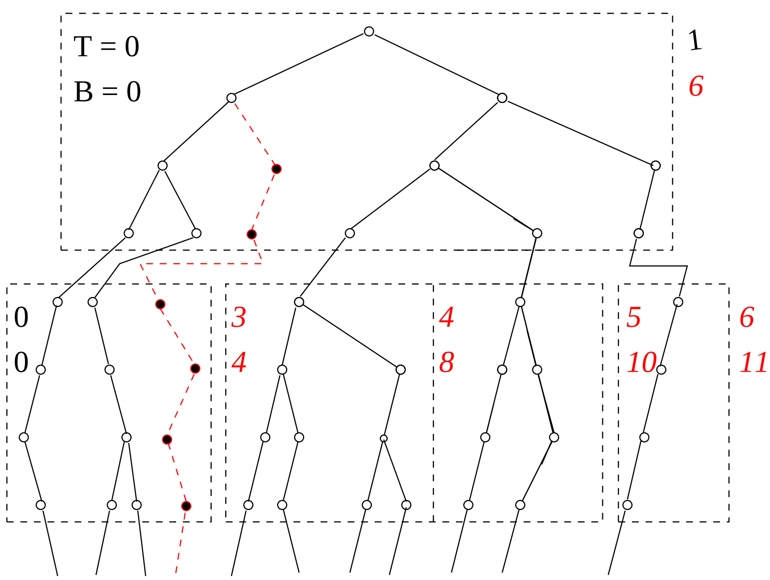
Updating Tries

Adding 10001010



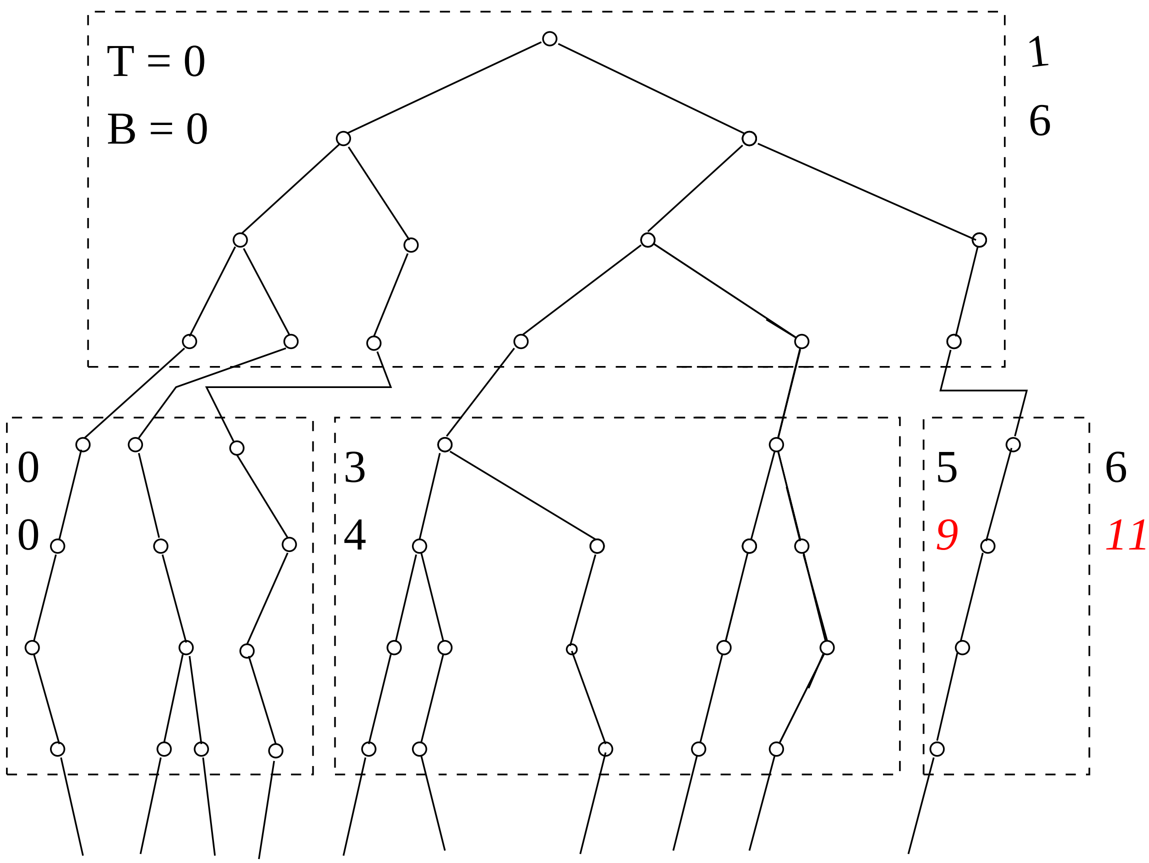
N.B. t node levels per page level; $\geq 2^t - 1$ nodes per page

Adding 01011010



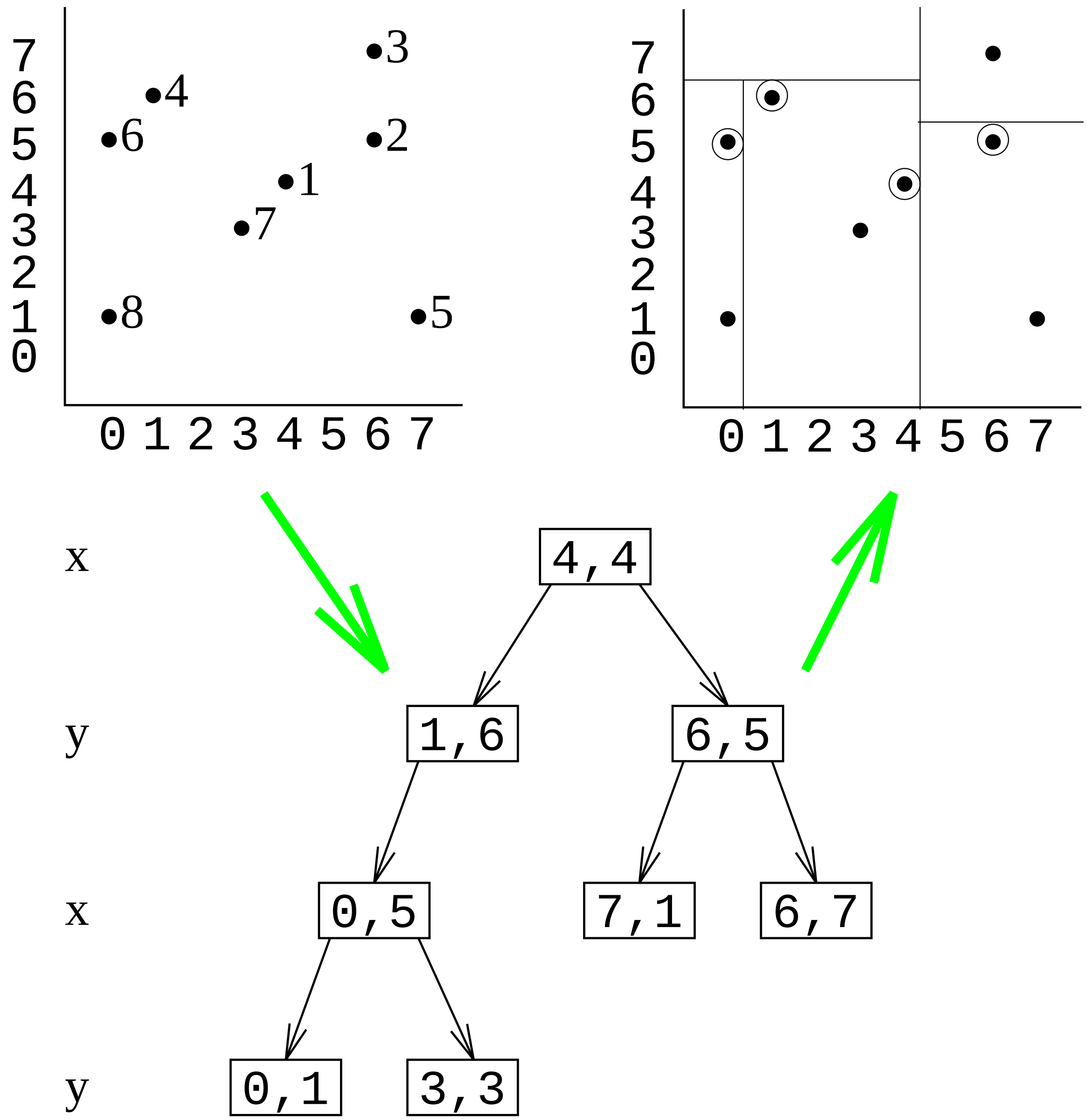
Updating Tries

Deleting 10001000

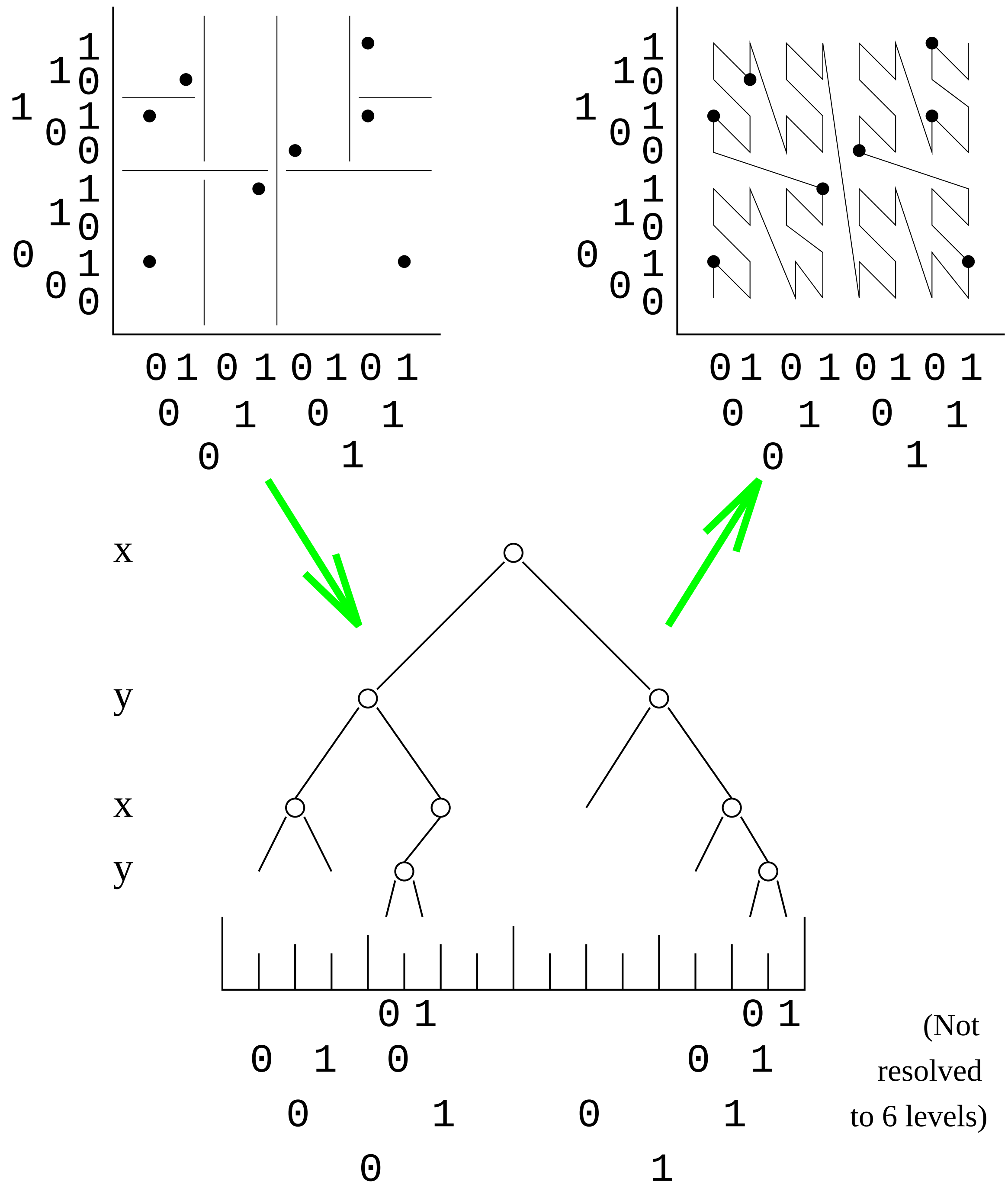


Symmetry

kd-Trees (2d-Tree)

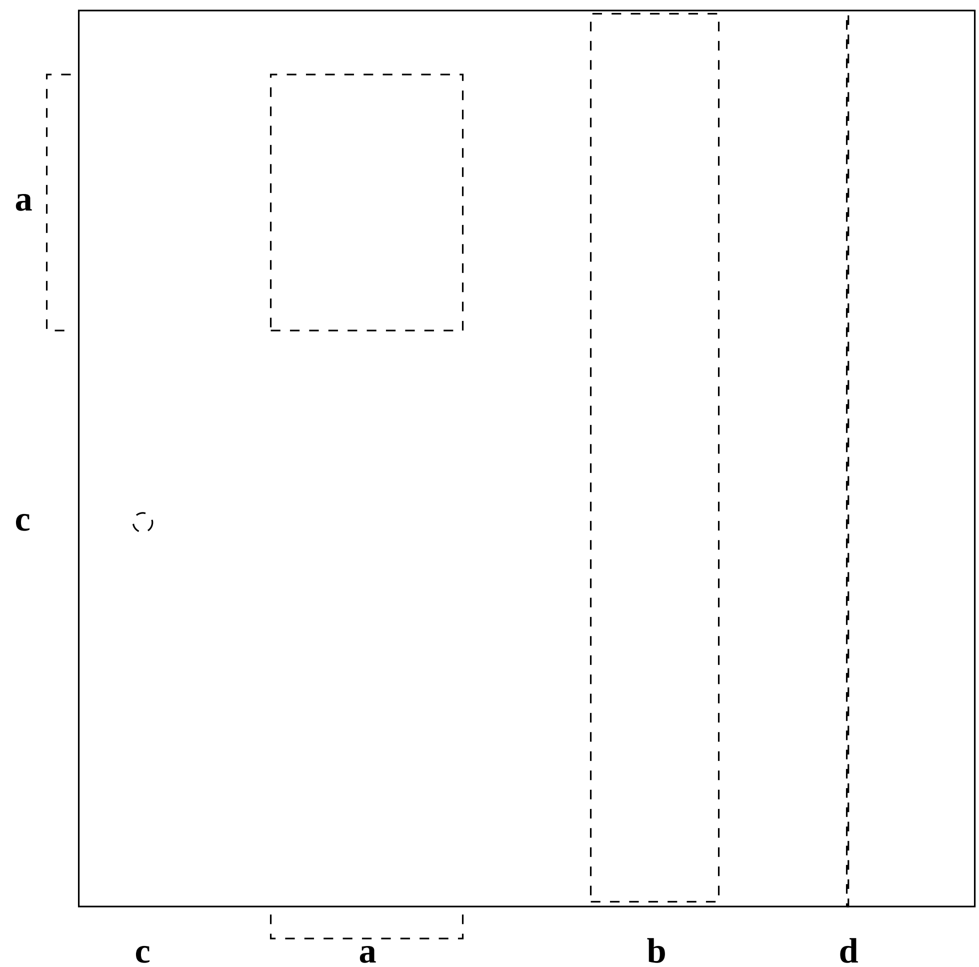


kd-Tries (2d-Trie)



Bit interleaving, e.g.: (6,5) = (110,101) ⇒ 111001

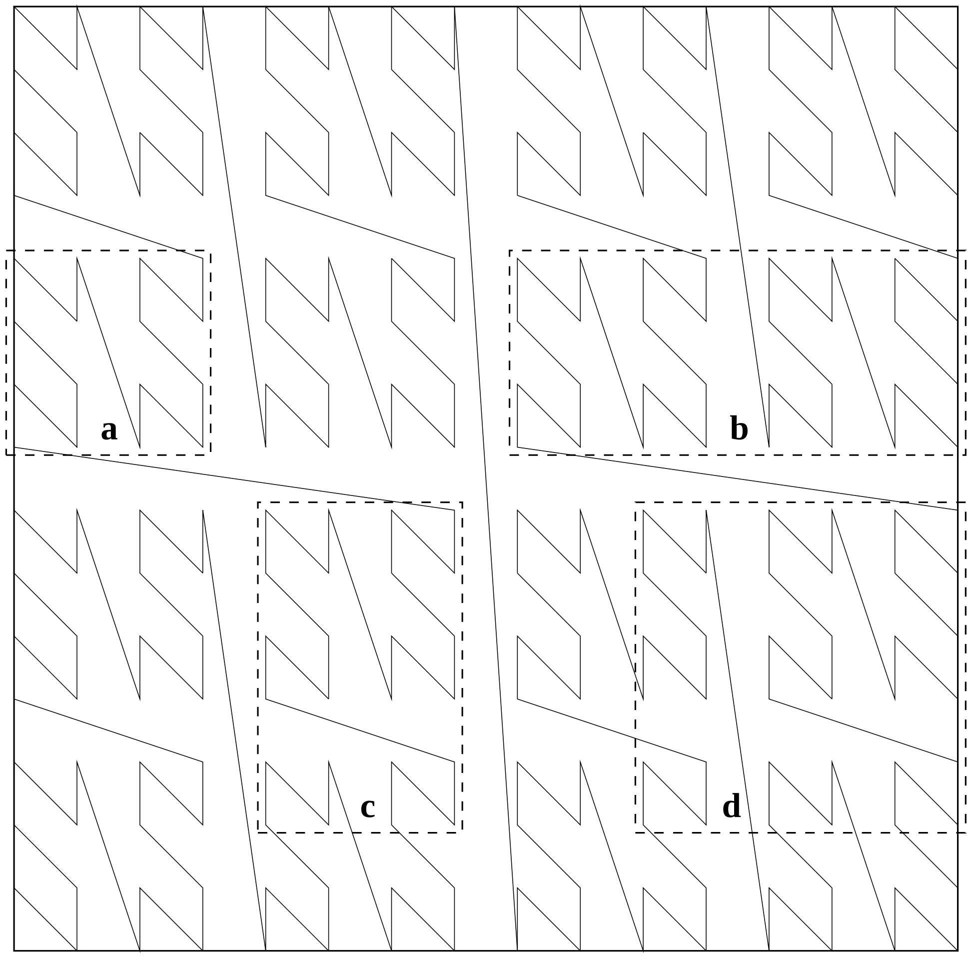
Orthogonal Range Queries



b) partial range; c) exact match; d) partial match

ORQ with Z-Order

(also applies to kd-tries)



a) 1 probe, 1 scan

b,c) 2 probes, 2 scans;

or 1 probe, 1 scan with extraneous reads

d) 4 probes, 4 scans; or ..