Wavelet Trees for All

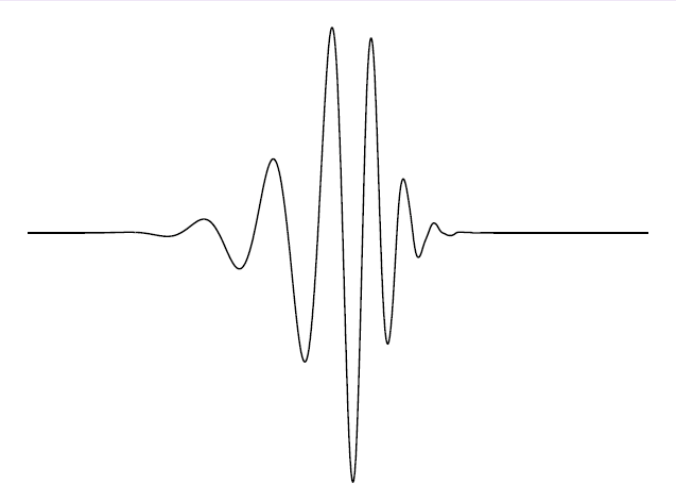
Gonzalo Navarro
www.dcc.uchile.cl/gnavarro
gnavarro@dcc.uchile.cl

Department of Computer Science
University of Chile

R. Grossi, A. Gupta, J.S. Vitter.
High-order entropy-compressed text indexes. *SODA*, Jan 2003.

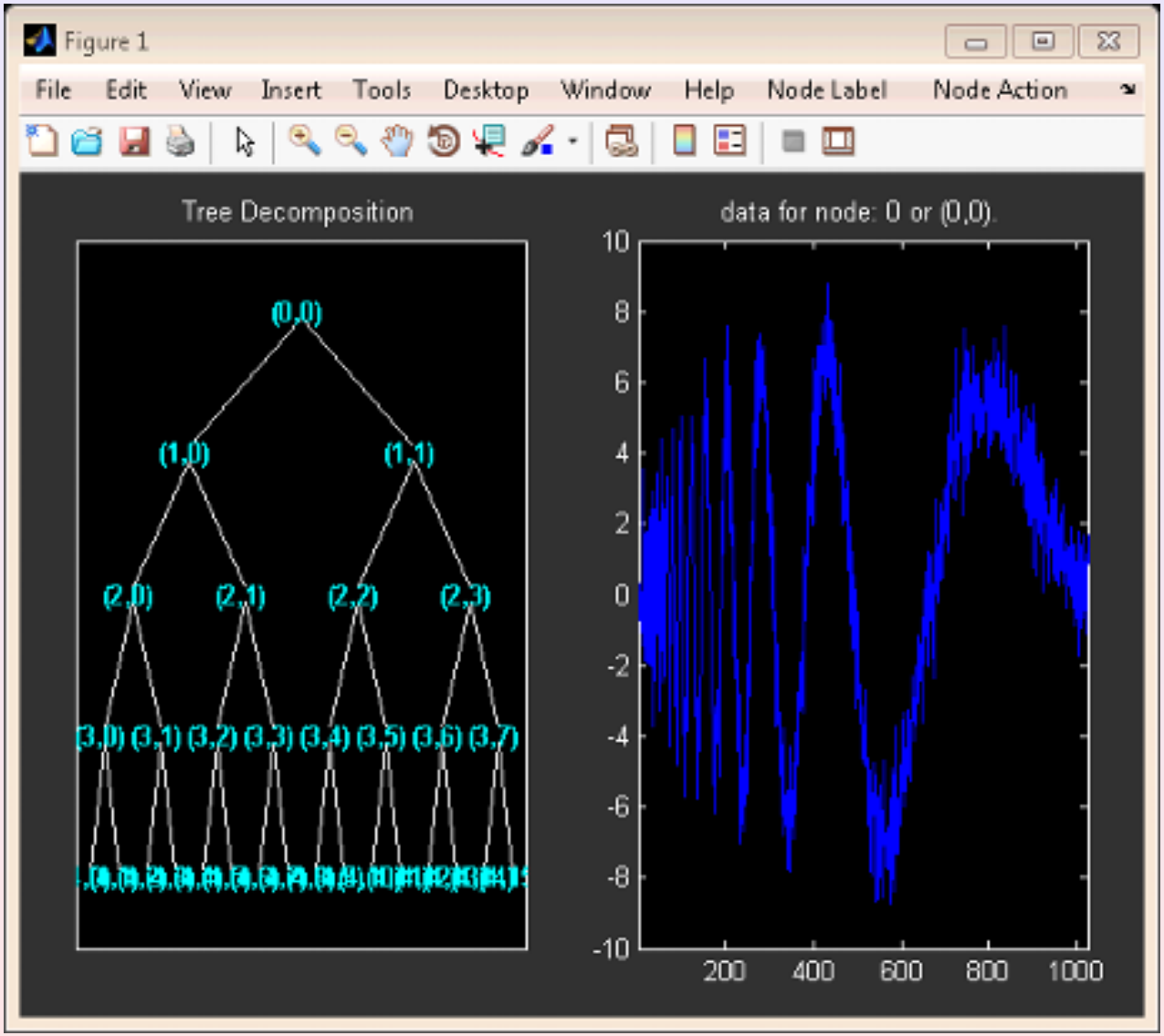
B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comp.* 1988.

Wavelet + Tree = Wavelet Tree?

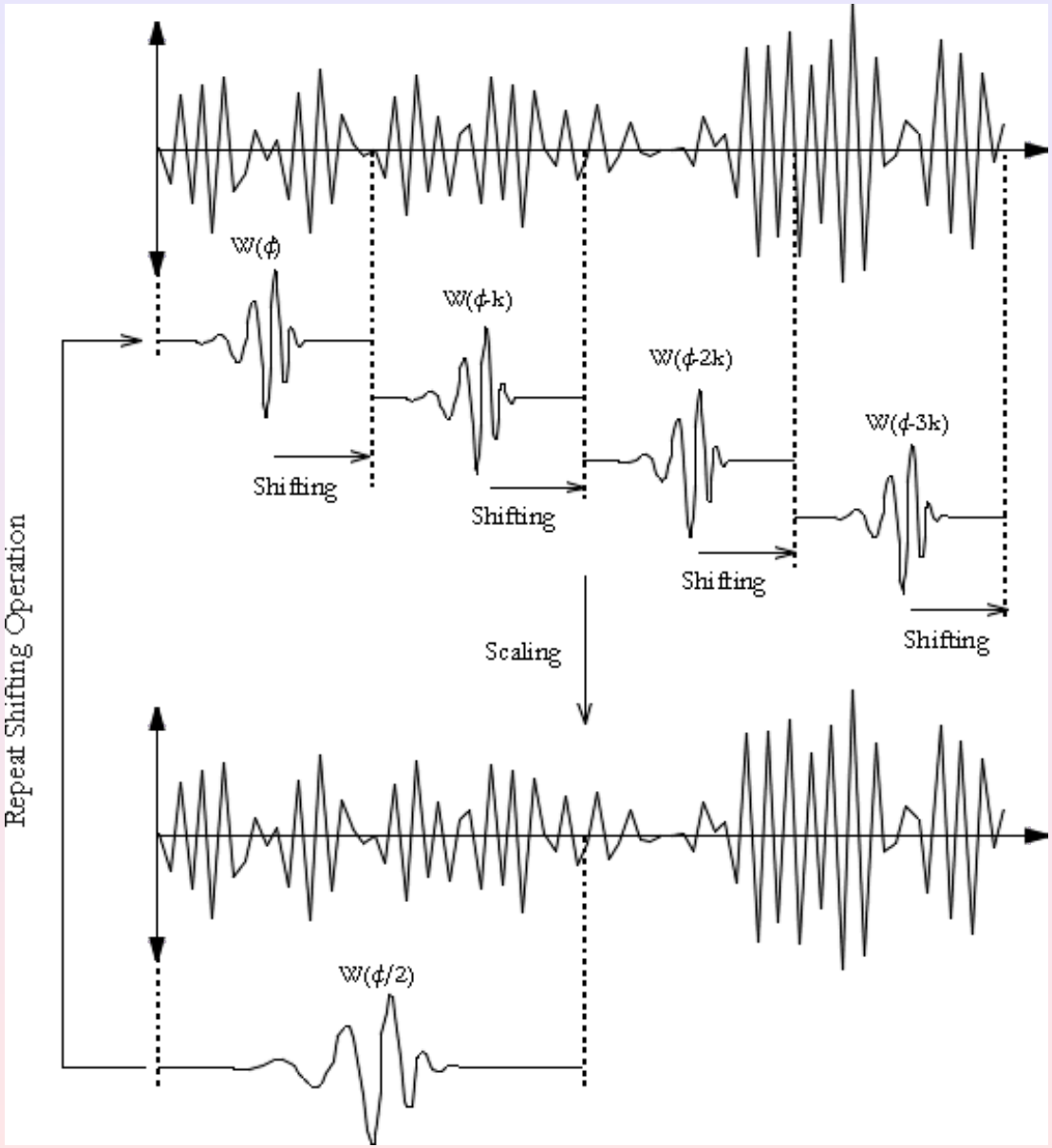


Wavelet + Tree = Wavelet Tree?

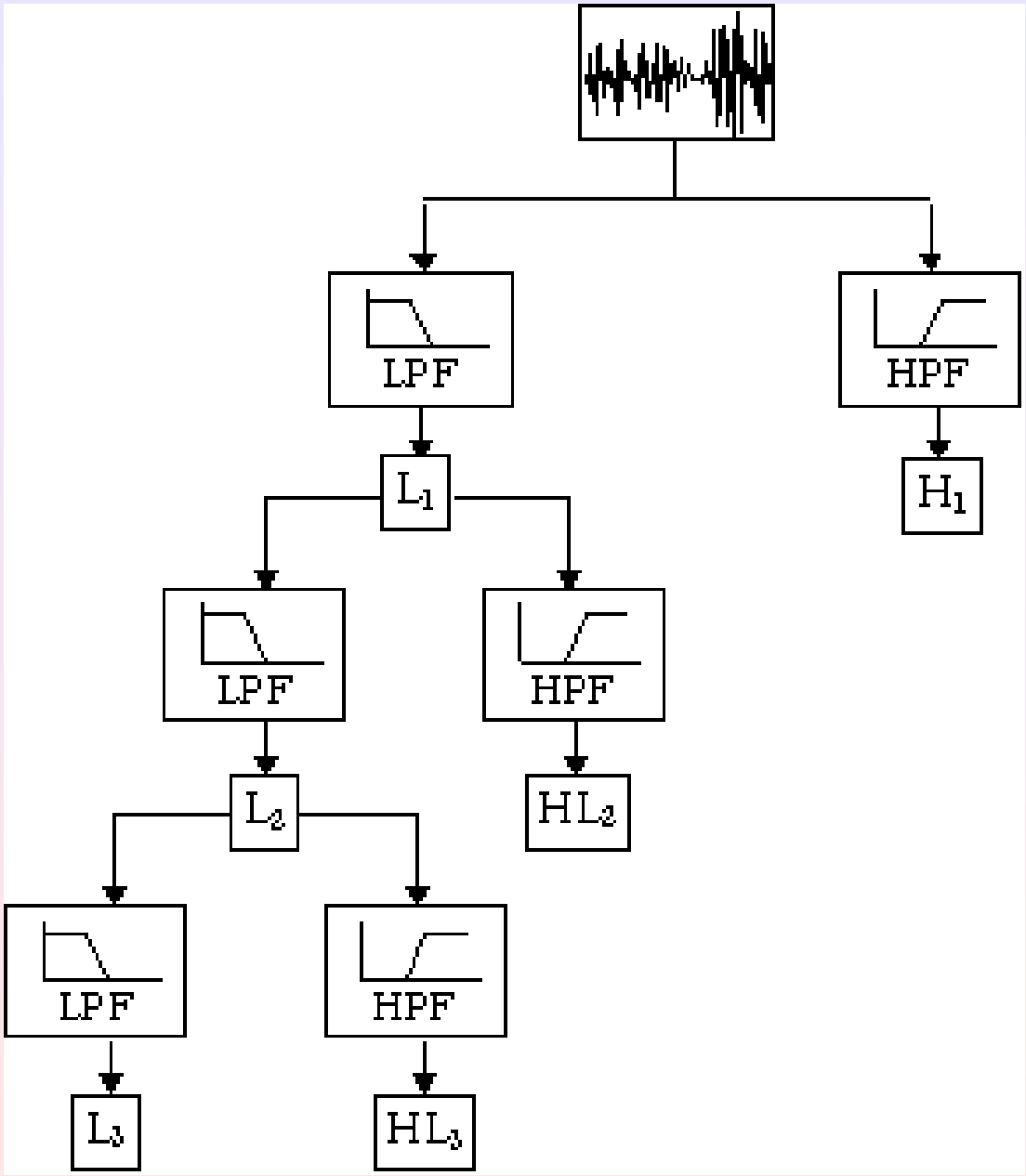
=



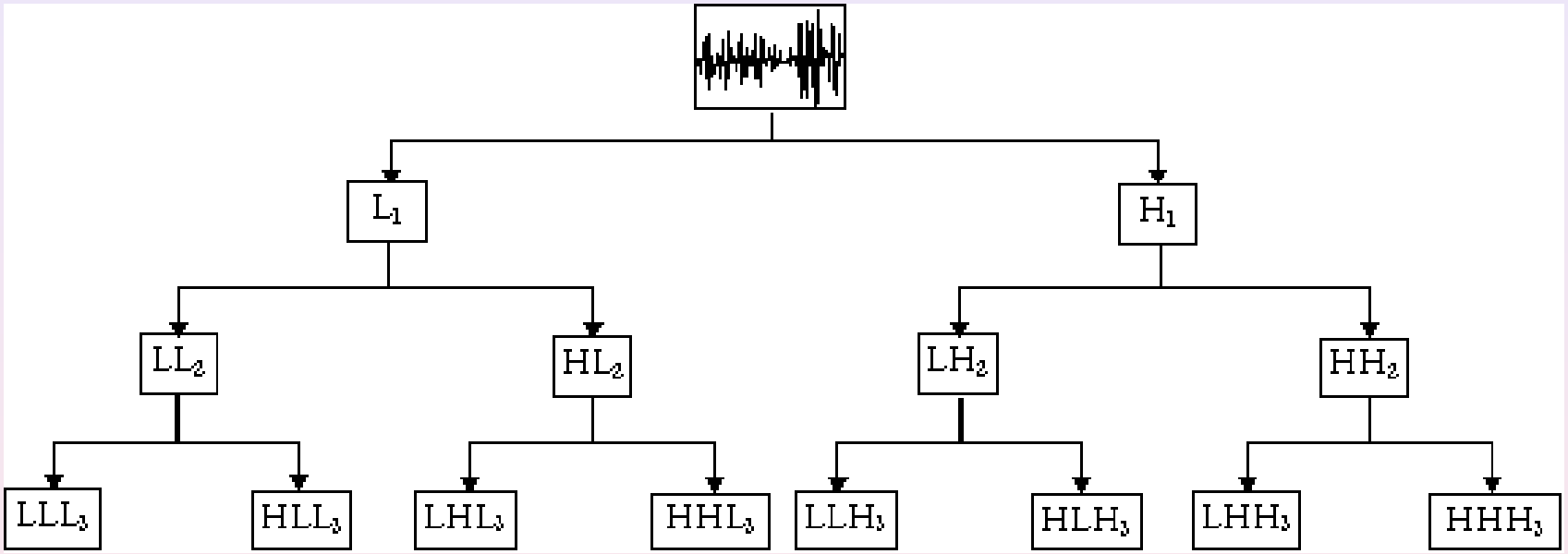
Wavelet + Tree = Wavelet Tree?



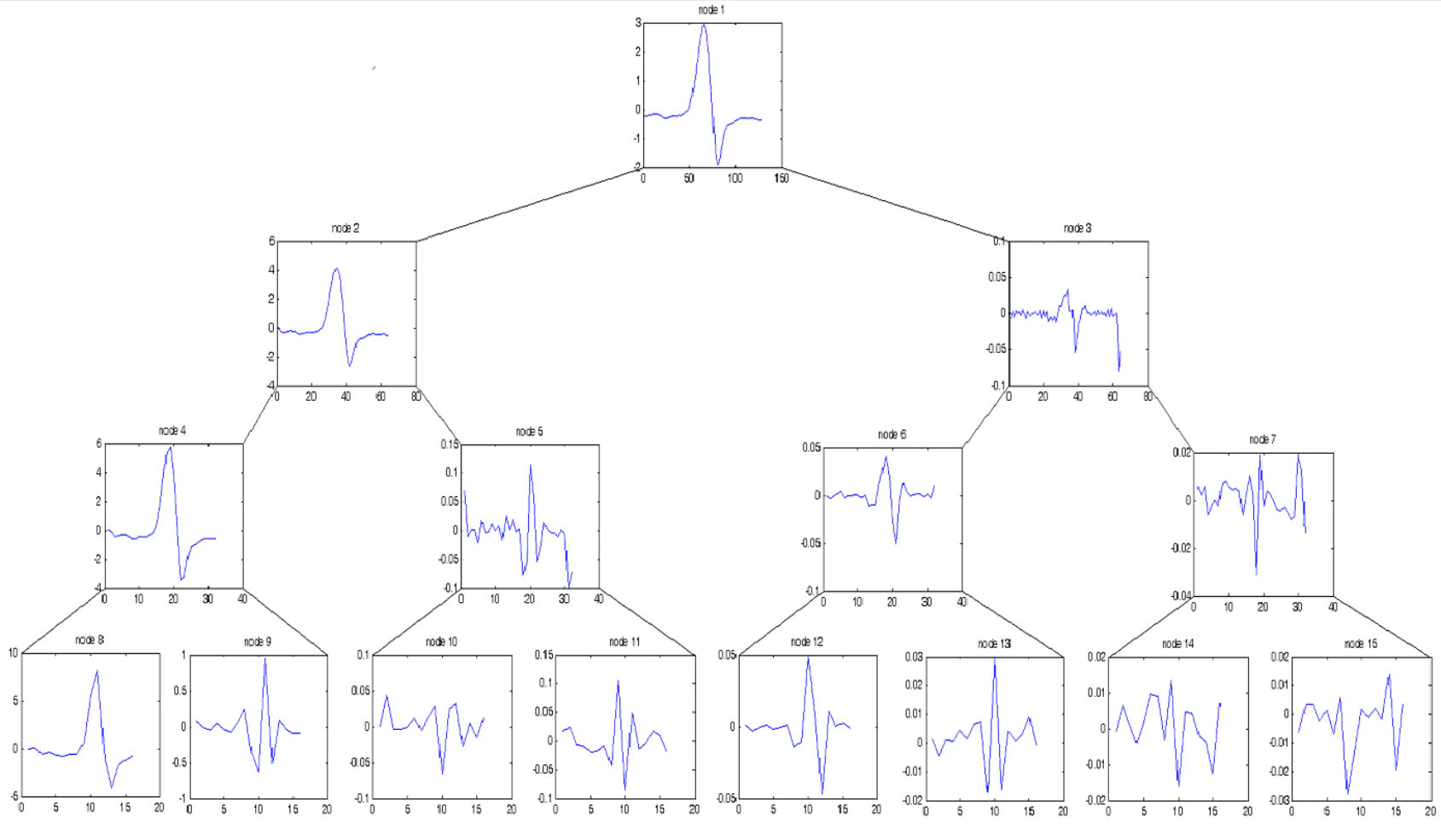
Wavelet + Tree = Wavelet Tree?



Wavelet + Tree = Wavelet Tree?



Wavelet + Tree = Wavelet Tree?



Part I: The Wavelet Tree Structure

Part I: The Wavelet Tree Structure

Part I: The Wavelet Tree Structure

Basic Structure

Basic Functionality

Compression

Changing Shape

Part I: The Wavelet Tree Structure

Basic Structure

Basic Functionality

Compression

Changing Shape

Part II: Applications

Part II: Applications

Part II: Applications

Part II: Applications

Three Views

Applications as Sequences

Applications as Reorderings

Applications as Grids

Part I: The Wavelet Tree Structure

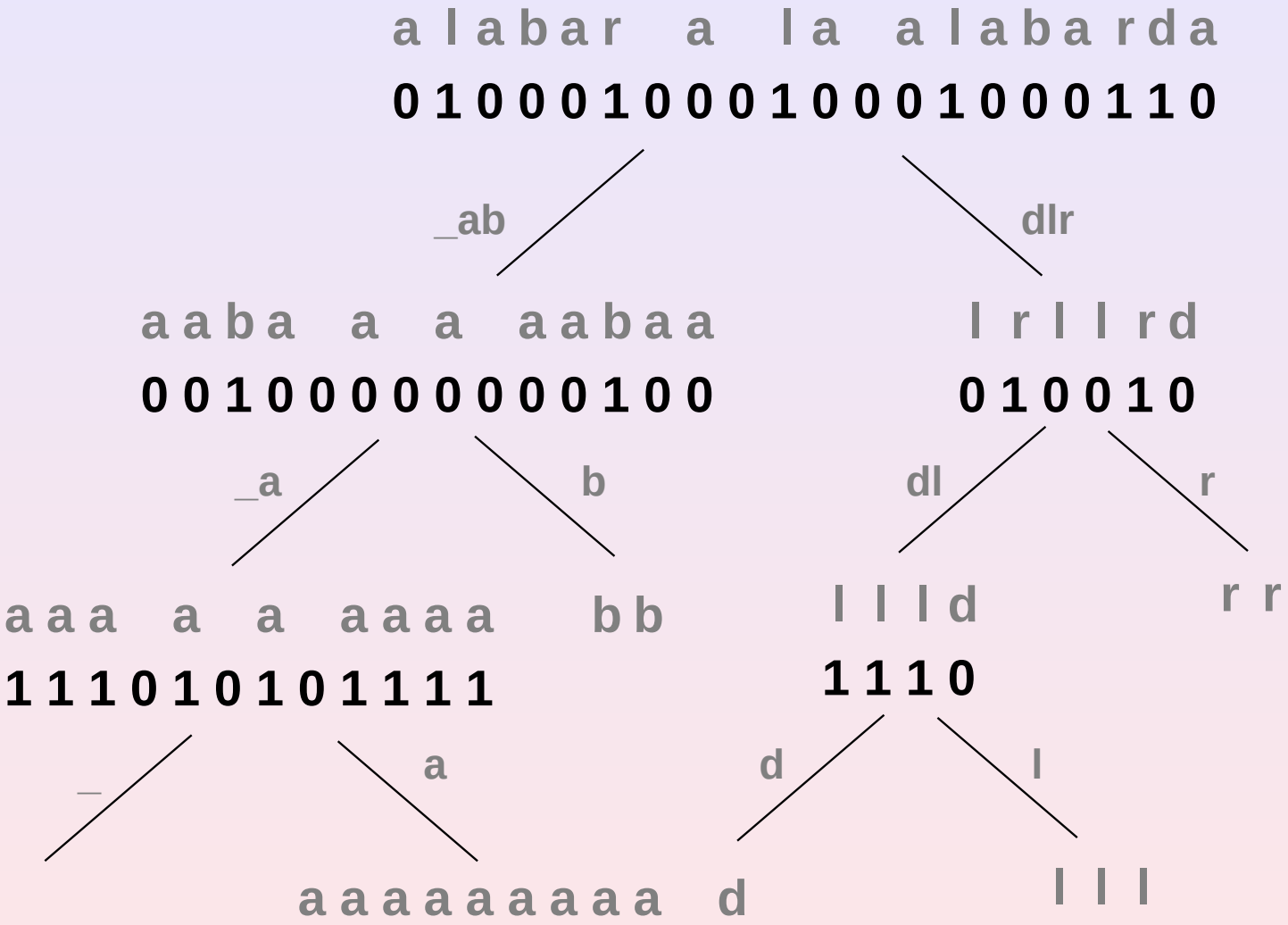
Basic Wavelet Tree Structure

- Built on a sequence $S[1, n]$ over alphabet $[1, \sigma]$.
- A binary and perfectly balanced tree.
- Each node represents a range $[a, b]$ of $[1, \sigma]$.
- The root represents the whole $[1, \sigma]$.
- The children of a node v split its alphabet range by half.
- Each leaf represents some $c \in [1, \sigma]$.

Basic Wavelet Tree Structure

- Associate with each node v a sequence $S_v[1, n_v]$ (not stored).
- Let $[a, b] \subseteq [1, \sigma]$ the alphabet range of v.
- Then S_v is the subsequence of S formed by the characters in $[a, b]$.
- At each node v we store a bitmap $B_v[1, n_v]$.
- $B_v[i] = 0$ iff $S_v[i]$ belongs to the alphabet of the left child.

Basic Wavelet Tree Structure



Basic Space Analysis

- The tree is balanced, so there are $h = \lceil \lg \sigma \rceil$ levels.
- We store at most n bits per level.
- Thus there are at most $n\lceil \lg \sigma \rceil$ bits.
- Plus $O(\sigma \lg n)$ bits for the tree pointers.
- Total: $n \lg \sigma + O(n) + O(\sigma \lg n)$.
- This is close to the space needed to represent $S[1, n]$ in plain form!

Part I: The Wavelet Tree Structure

Basic Functionality: Tracking Symbols

- Start at a position i at the root bitmap B_{root}.
- Where has it gone, left or right?
  - Depends on whether $B_{root}[i] = 0$ or 1.
- And at which position has it been mapped?
  - If it went left, to $i_0 =$ number of 0s up to i in B_{root}.
  - If it went right, to $i_1 =$ number of 1s up to i in B_{root}.
- Continue recursively downwards.
- When we arrive at the leaf of symbol c, it turns out that $S[i] = c$.

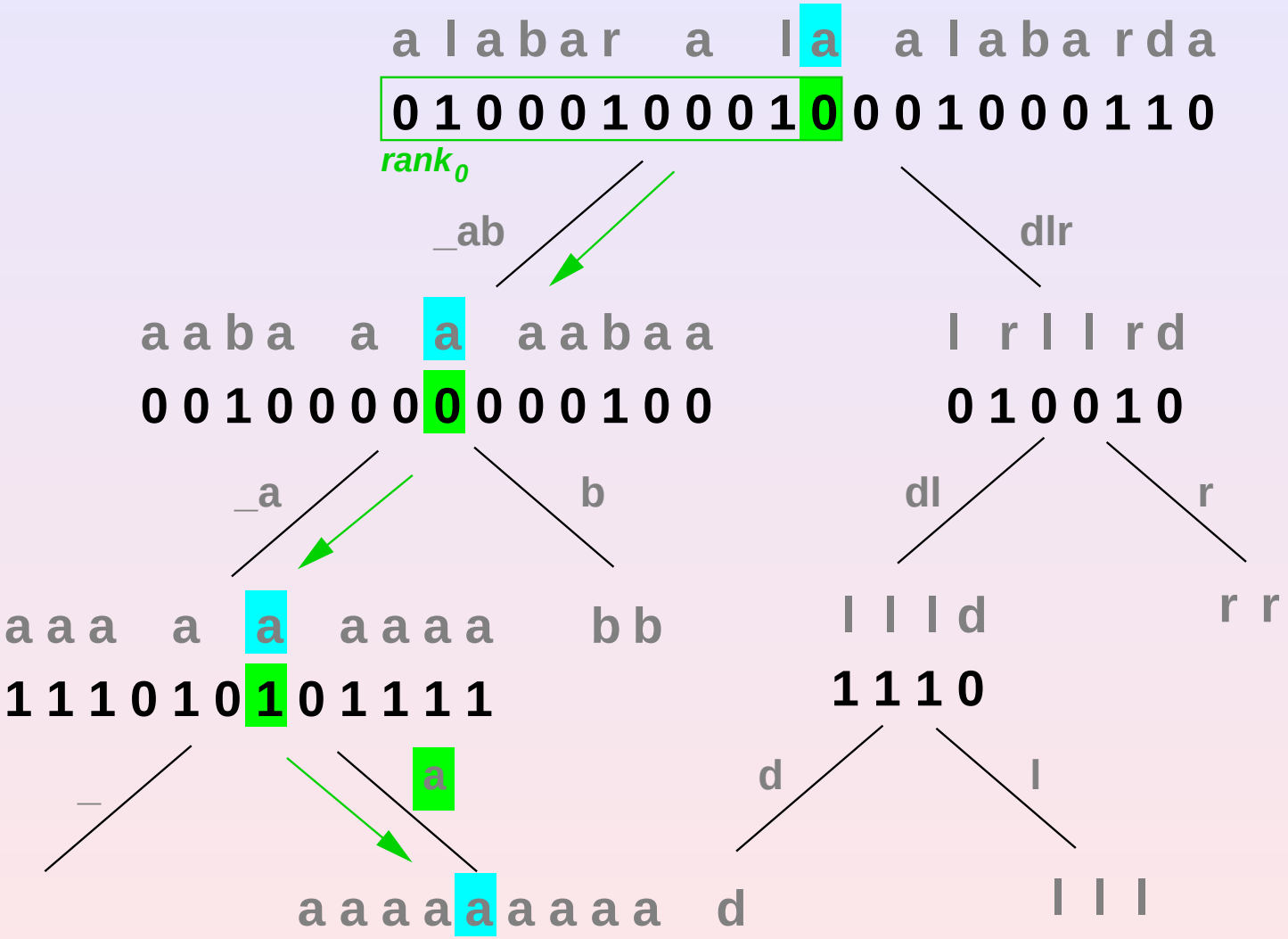
Basic Functionality: Tracking Symbols

- Values i_0 and i_1 can be defined via operation

$$rank_b(B, i) \quad = \quad \text{number of occurrences of bit } b \text{ in } B[1, i]$$

- Operation *rank* can be computed in constant time...
- ... by preprocessing B and storing $o(|B|)$ further bits.
- Thus we obtain any $S[i]$ in time $O(\lg \sigma)$.
- The space raises to $n \lg \sigma + o(n \lg \sigma) + O(n) + O(\sigma \lg n)$
- Thus the wavelet tree replaces S, more or less within the same space.

Basic Functionality: Tracking Symbols



Basic Functionality: Tracking Upwards

- Similarly, we can start at a position j of the leaf of a symbol c and go upwards.
- Where is the current position i in the parent node v?
  - If my node is a left child of v, at $j_0 =$ position where the ith 0 occurs in B_v.
  - If my node is a right child of v, at $j_1 =$ position where the ith 1 occurs in B_v.
- Continue recursively up to the root.
- At the root, we have found the position in S of the jth c.

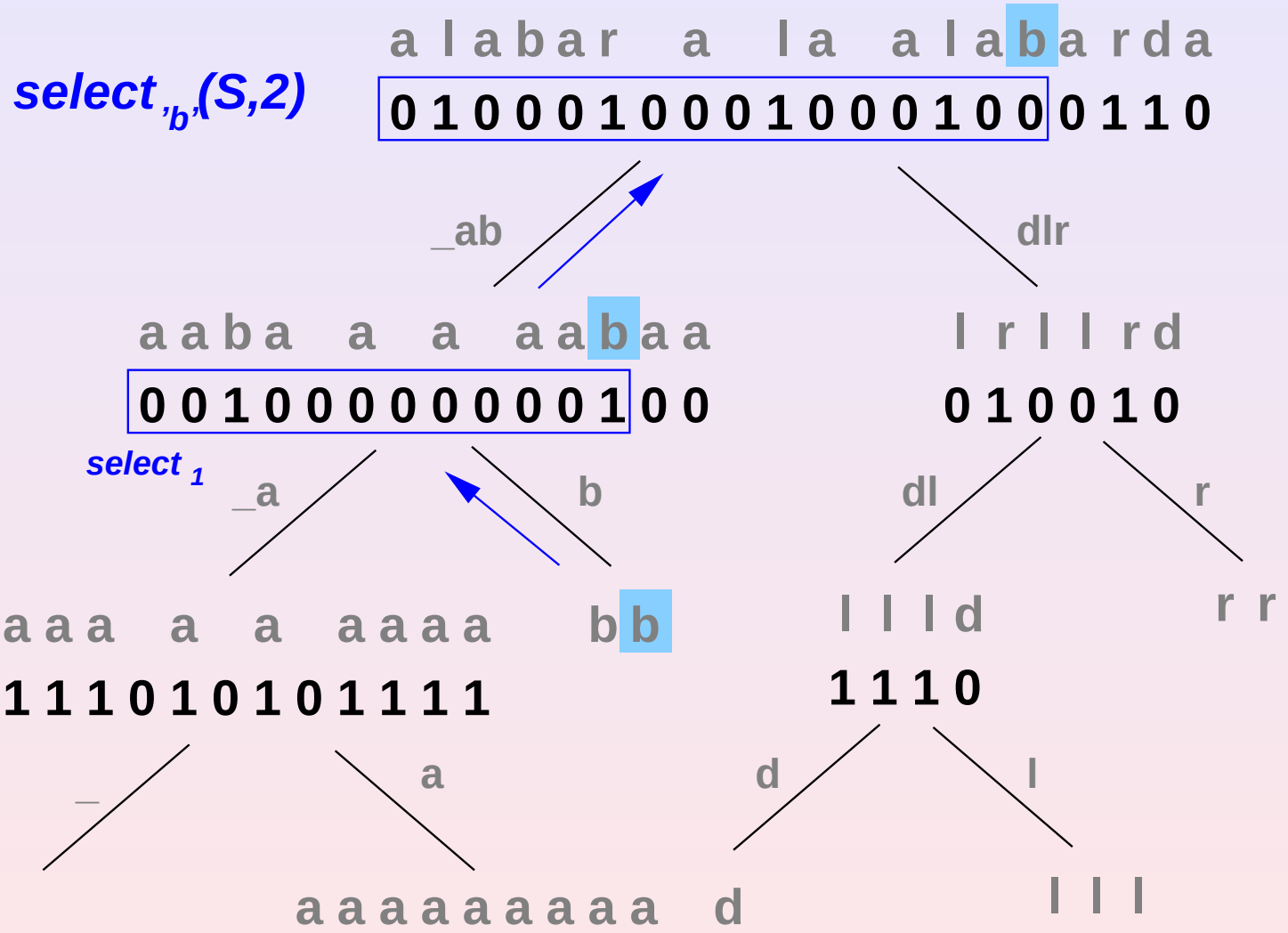
Basic Functionality: Tracking Upwards

- Values j_0 and j_1 can be defined via operation

$$\mathit{select}_b(B, j) \quad = \quad \text{position of the } j\text{th } b \text{ in } B$$

- Operation *select* can be also computed in constant time...
- ... by preprocessing B and storing $o(|B|)$ further bits.
- Thus we track upwards also in time $O(\lg \sigma)$.
- Indeed, we solve on S the generalization of select_b to sequences.

Basic Functionality: Tracking Upwards



Some Technical Improvements

- The tree pointers can be eliminated by concatenating all the bitmaps of the same level: $O(\sigma \lg n)$ disappears from the space.
- The total space can be reduced to $n \lg \sigma + o(n)$ by using recent low-redudnancy bitmap representations.
- The downward traversal can be sped up to $O(\lg \sigma / \lg w)$, in a machine of w bits, by using multiary trees.
- The upward traversal can be sped up to $O(\lg^{\epsilon} \sigma)$, at the price of $O((1/\epsilon) n \lg \sigma)$ bits of space, by using long upward pointers.

Part I: The Wavelet Tree Structure

Entropy Compression

- When bitmaps have many more 0s that 1s, or viceversa, they can be compressed.
- If $B[1, n]$ has n_0 0s and n_1 1s, then

$$H_0(B) = \frac{n_0}{n} \lg \frac{n}{n_0} + \frac{n_1}{n} \lg \frac{n}{n_1}$$

- This is lower when $n_0 << n_1$ or viceversa, and maximum ($H_0(B) = 1$) if $n_0 = n_1$.
- This notion can be extended to sequences $S[1, n]$:

$$H_0(S) = \sum_{c \in [1,\sigma]} \frac{n_c}{n} \lg \frac{n}{n_c}$$

  where n_c is the frequency of c in S.
- $H_0(S)$ is lower when some symbols are more frequent than others, and it always holds $H_0(S) \leq \lg \sigma$.

Entropy Compression

- There exist constant-time rank/select capable bitmap representations that require $nH_0(B) + o(n)$ bits.
- What happens if we use them on the bitmaps B_v of a wavelet tree?
- Say $B_{root}[1, n]$ has n_0 0s and n_1 1s, then the first level uses

$$n_0 \lg \frac{n}{n_0} + n_1 \lg \frac{n}{n_1} \quad \text{bits.}$$

Entropy Compression

- Say v_0 is the left child of the root and v_1 its right child.
- Says $B_{v_0}[1, n_0]$ has n_{00} 0s and n_{01} 1s.
- Then the bitmap B_{v_0} uses

$$n_{00} \lg \frac{n_0}{n_{00}} + n_{01} \lg \frac{n_0}{n_{01}} \quad \text{bits.}$$

- Similarly, B_{v_1} uses

$$n_{10} \lg \frac{n_1}{n_{10}} + n_{11} \lg \frac{n_1}{n_{11}} \quad \text{bits.}$$

- Adding up the three bitmaps, we have that the first two levels use

$$n_{00} \lg \frac{n}{n_{00}} + n_{01} \lg \frac{n}{n_{01}} + n_{10} \lg \frac{n}{n_{10}} + n_{11} \lg \frac{n}{n_{11}} \quad \text{bits.}$$

- This would be $nH_0(S)$ if $\sigma = 4$!

Entropy Compression

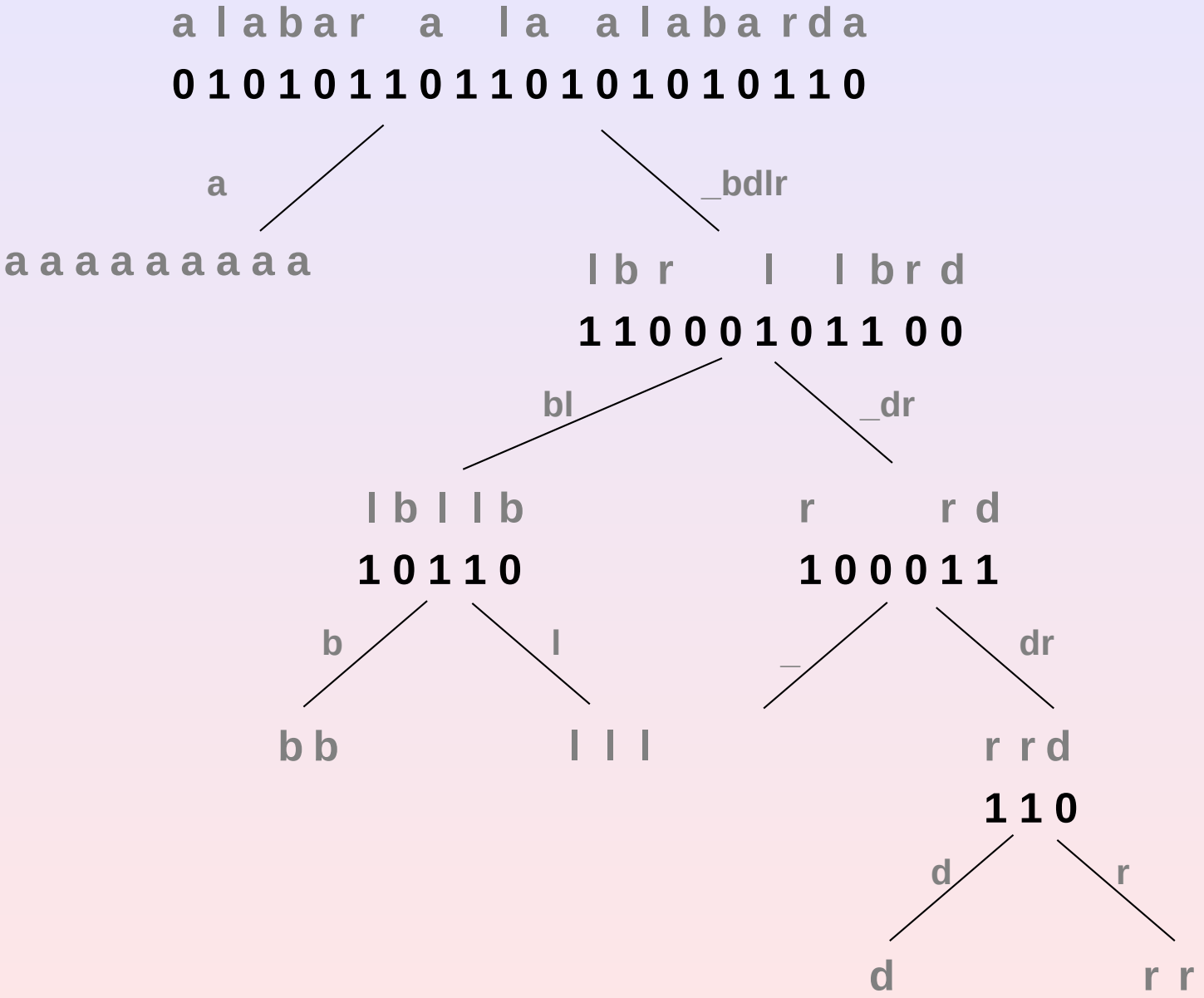
- It is not hard to see that this holds in general: the sum of all $n_v H(B_v)$ is $nH_0(S)$.
- The total space can be as good as $nH_0(S) + o(n)$ bits.
- Previous time complexities are maintained.
- Thus the wavelet tree represents S in compressed form.

Part I: The Wavelet Tree Structure

Changing Shape

- Zero-order entropy can also be achieved with a radically different technique.
- Instead of a balanced shape, give the wavelet tree a Huffman shape.
- Use the frequencies of the symbols in S.
- One can use plain (and faster) bitmap representations.
- The total number of bits represented is less than $n(H_0(S) + 1)$.

Changing Shape



Changing Shape

- Average times are better than $O(\lg \sigma)$: they can be $O(1 + H_0(S))$ under some conditions.
- Worst case times can worsen, but they can be brought back to $O(\lg \sigma)$ at no asymptotic price, by rebalancing deep subtrees.
- Huffman shape gives a more practical way to reduce the redundancy, from $o(n \lg \sigma)$ to $o(n(H_0(S) + 1)$, even if in theory $o(n)$ can be obtained.
- It requires $O(\sigma \lg n)$ bits of space again.

Changing Shape

- Note that this can be used on any variable-length encoding, not only Huffman.
- For example, a sequence of δ-codes.
- The Wavelet Tree would require roughly the same number of bits...
- ... yet it would allow extracting the code of the ith symbol in time proportional to its length.

Part II: Applications

Three Views

A wavelet tree on a sequence of symbols can be regarded in three ways:

- As representing a sequence.
- As representing a reordering.
- As representing a grid of points.

As a Sequence

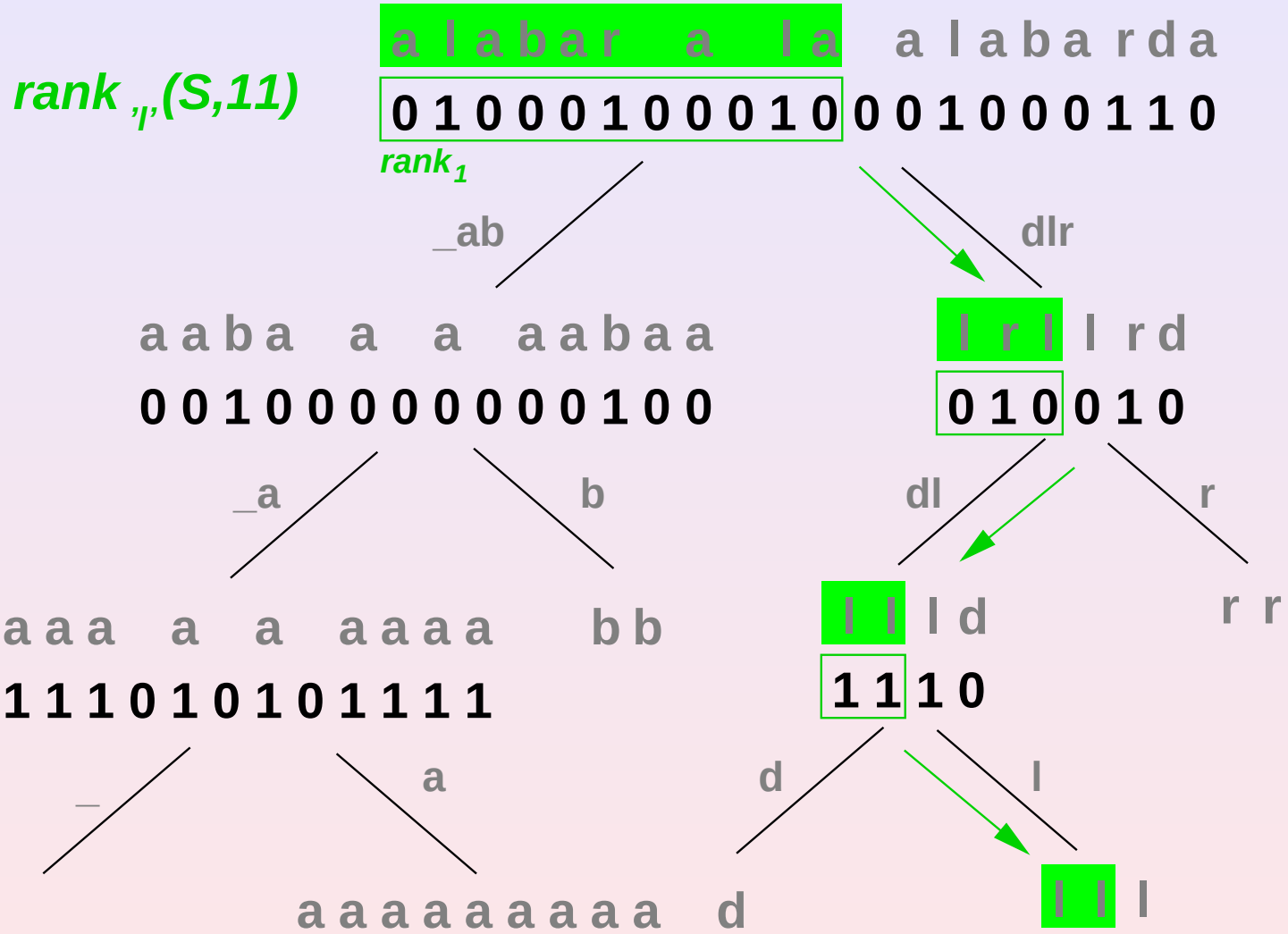
- As we have presented it till now.
- It represents $S[1, n]$ within compressed space.
- It supports operations
  - *access*, downward tracking.
  - *select*, upward tracking.
  - *rank*, defined as

$$rank_c(S, i) \quad = \quad \text{number of occurrences of symbol } c \text{ in } S[1, i]$$

As a Sequence: Rank Operation

- Start at a position i at the root bitmap B_{root}.
- Where do cs go, left or right?
  - If left, continue to left child at position $i \leftarrow i_0$.
  - If right, continue to right child at position $i \leftarrow i_1$.
- When we arrive at the leaf of symbol c, i is the answer.

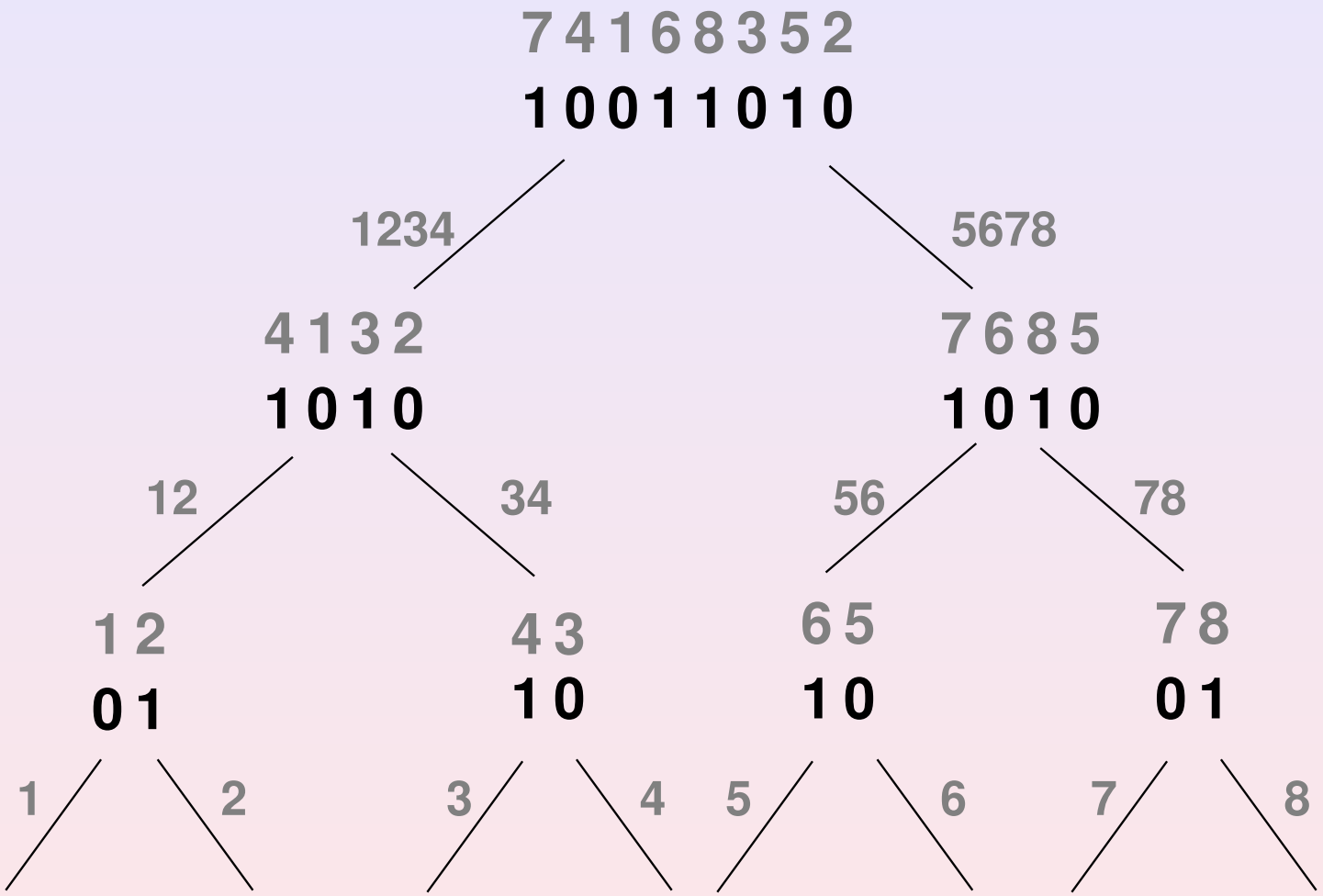
As a Sequence: Rank Operation



As a Reordering

- We have a sequence on the top...
- ... and a stable sorting of its elements at the bottom.
- The wavelet tree describes the sorting process.
- For example, it can represent a permutation π of $[1, n]$ using $n \lg n + o(n)$ bits.
- It supports both $\pi(i)$ and $\pi^{-1}(i)$ in time $O(\lg n / \lg \lg n)$.

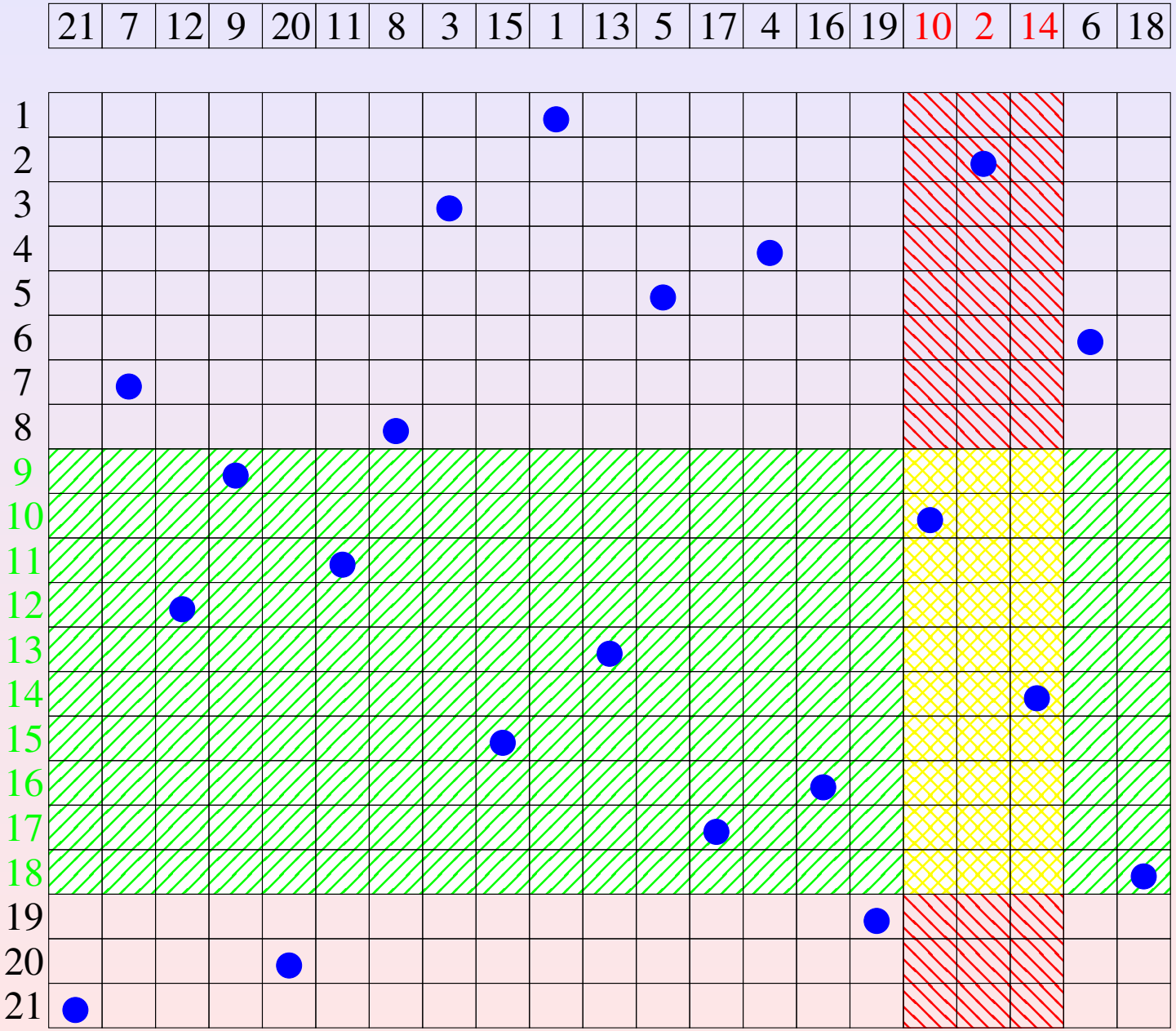
As a Reordering: Permutations



As a Grid of Points

- Simplest case: $n \times n$ grid with n points.
- Exactly one point per row and one per column.
- More general cases require (easy) mappings from real coordinates.
- The main operation in these applications: count/report the points in a rectangle.

As a Grid of Points



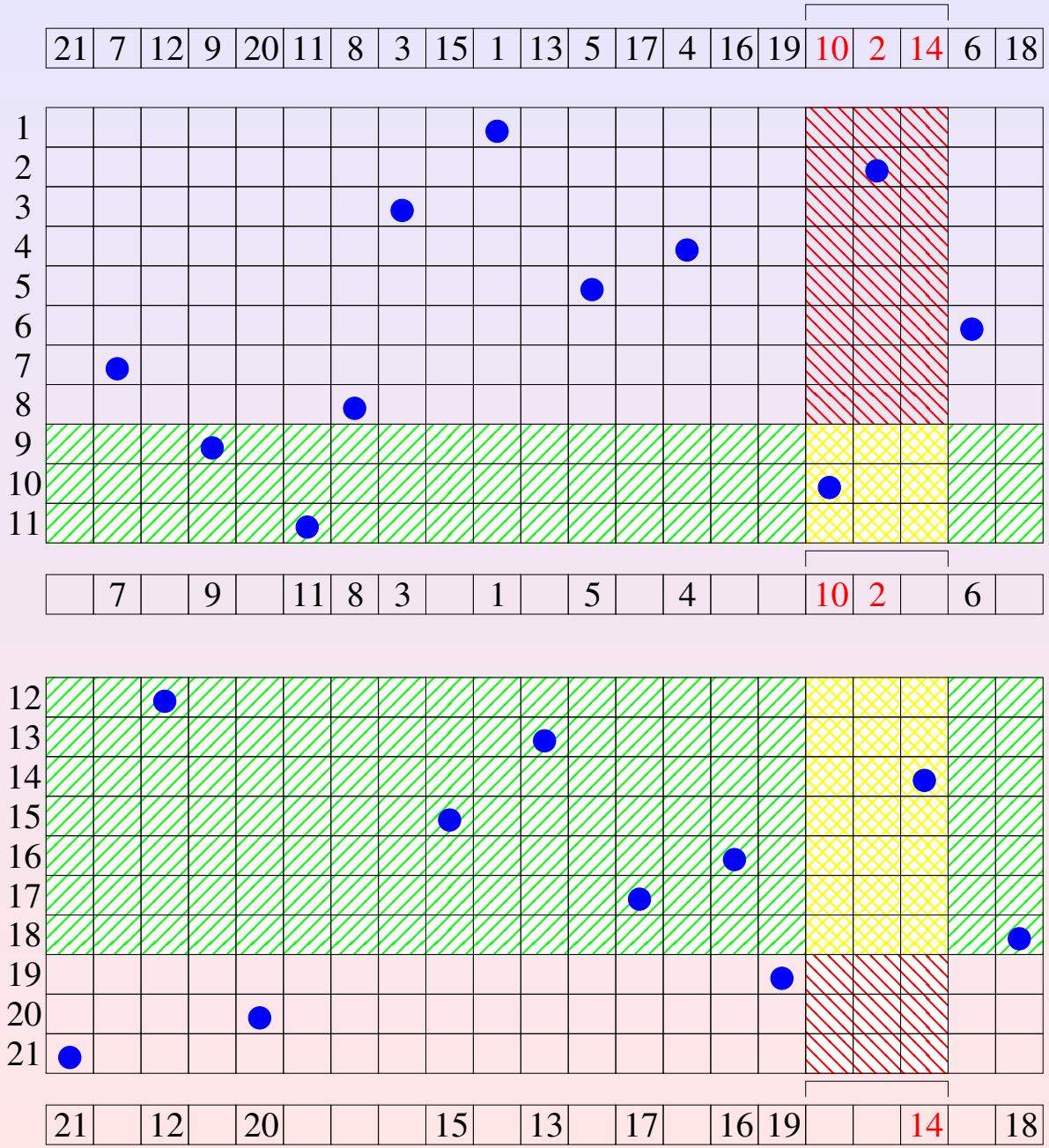
As a Grid of Points

- Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be the points sorted by x-coordinate.
- Consider the wavelet tree of string $S = y_1 y_2 \ldots y_n$.
- Then the ith point in x-order is $(i, S[i])$.
- And the ith point in y-order is $(select_i(S), i)$.

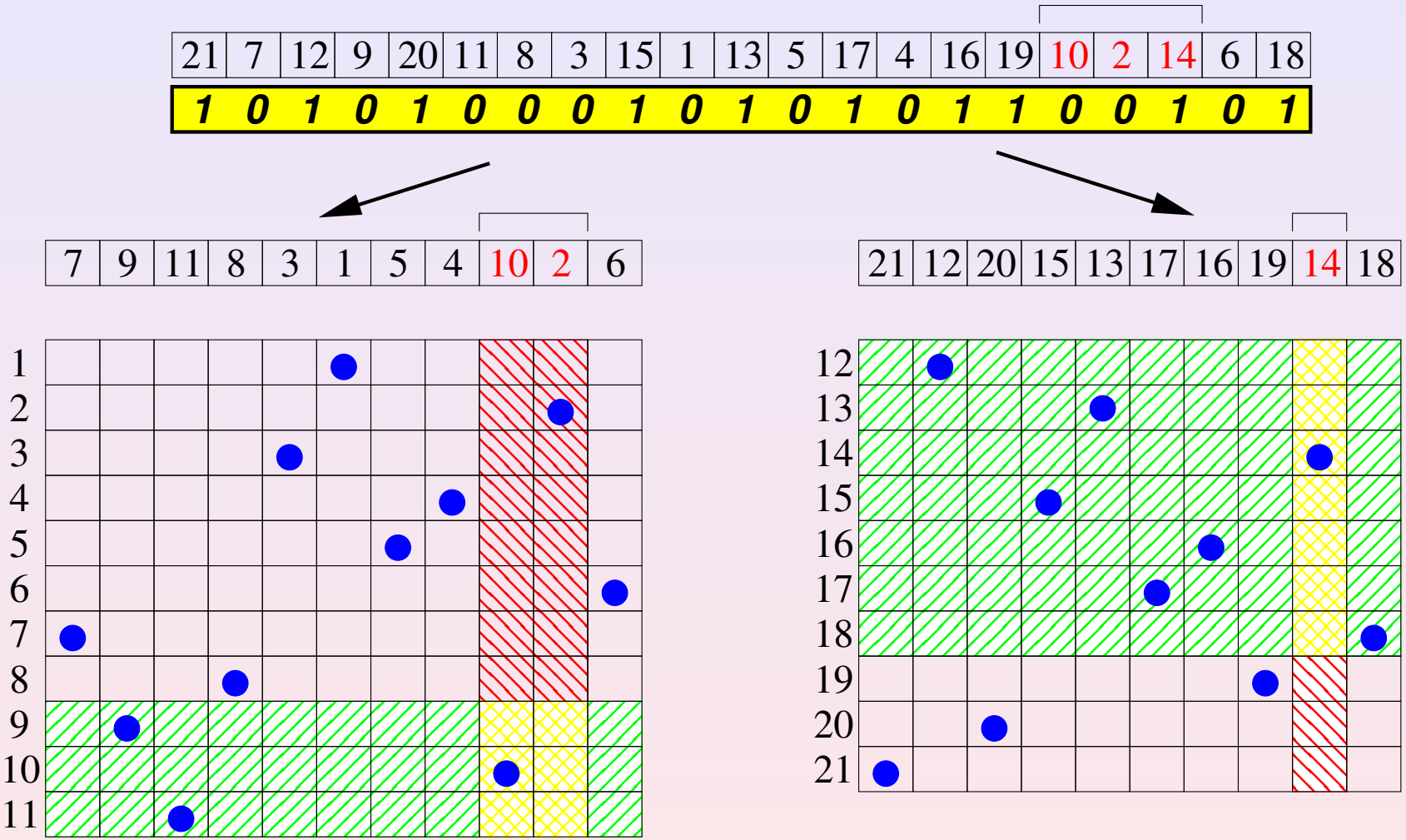
As a Grid of Points: Range Queries

- To count or report the points within $[x_1, x_2] \times [y_1, y_2]$.
- Start at the root with interval $[x_1, x_2]$.
- Project it into the left and into the right children.
- Stop when
  - The current interval $[x_1, x_2]$ is empty.
  - The current "alphabet" interval does not intersect $[y_1, y_2]$.
  - The current alphabet interval is contained in $[y_1, y_2]$.
- In the latter case, count or report all the elements in $[x_1, x_2]$.
- Counting time: $O(\lg n / \lg \lg n)$.
- Reporting time $O((k+1) \lg^{\epsilon} n)$.

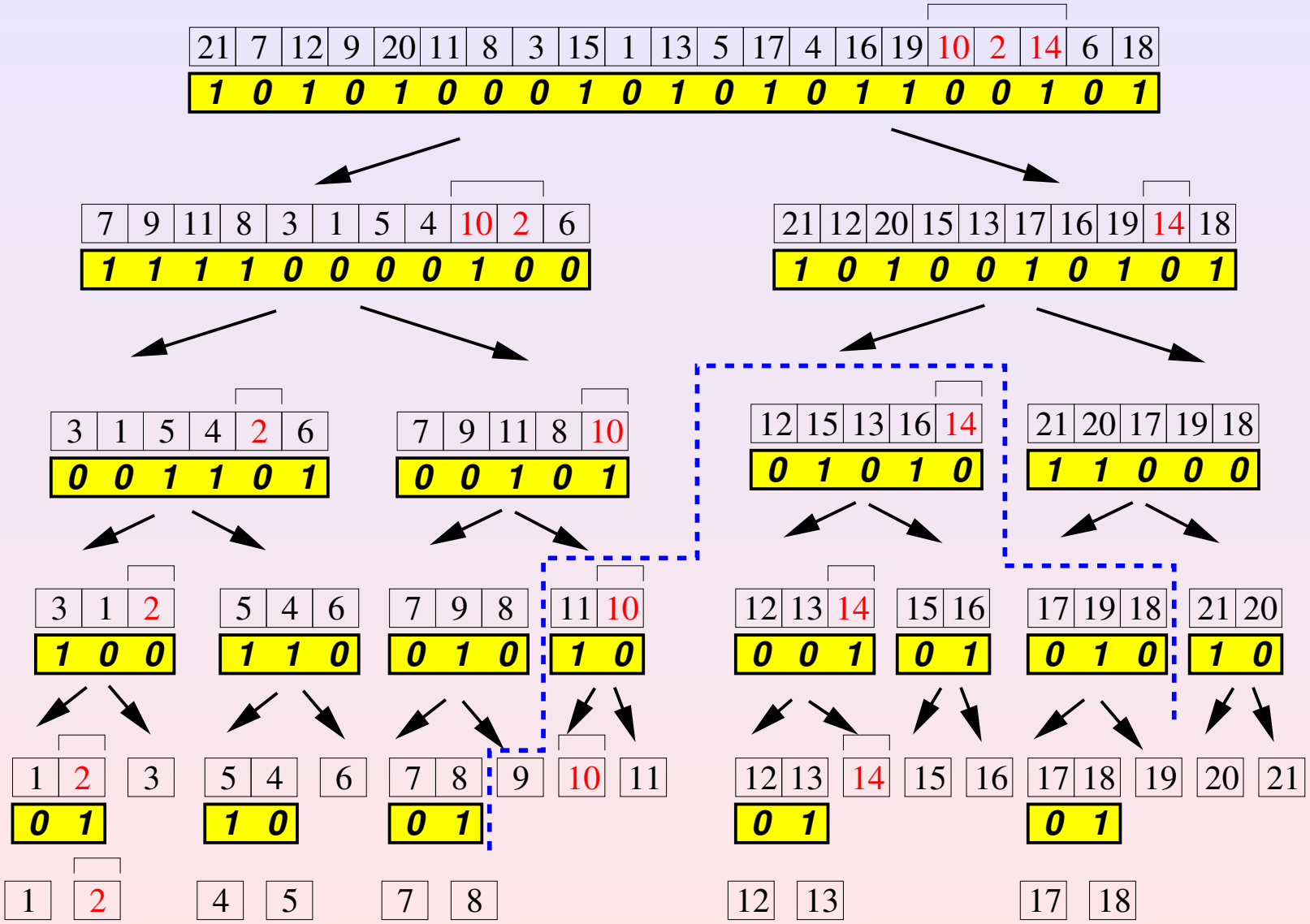
As a Grid of Points: Range Queries



As a Grid of Points: Range Queries



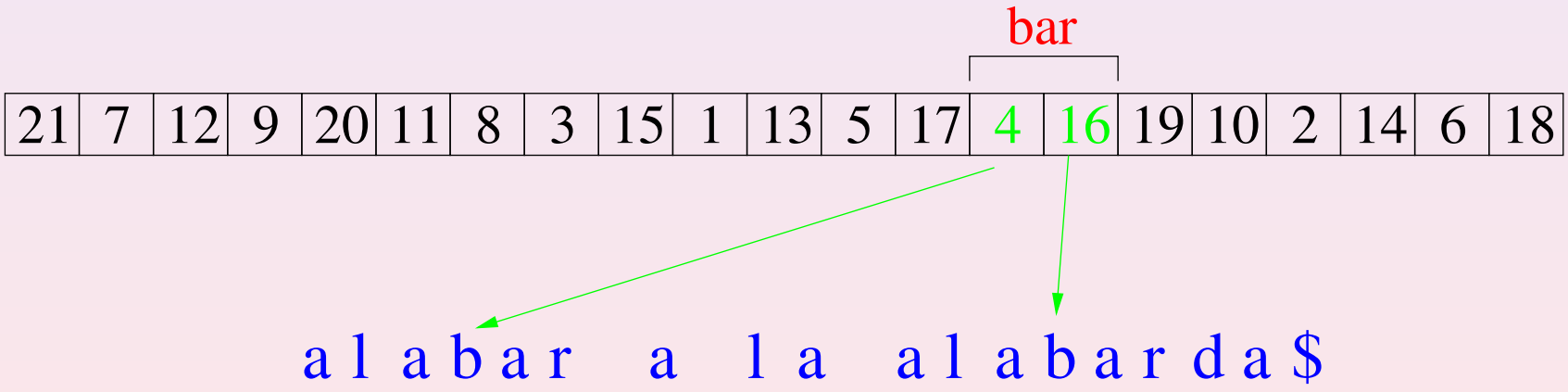
As a Grid of Points: Range Queries



Part II: Applications

Full-text Indexes

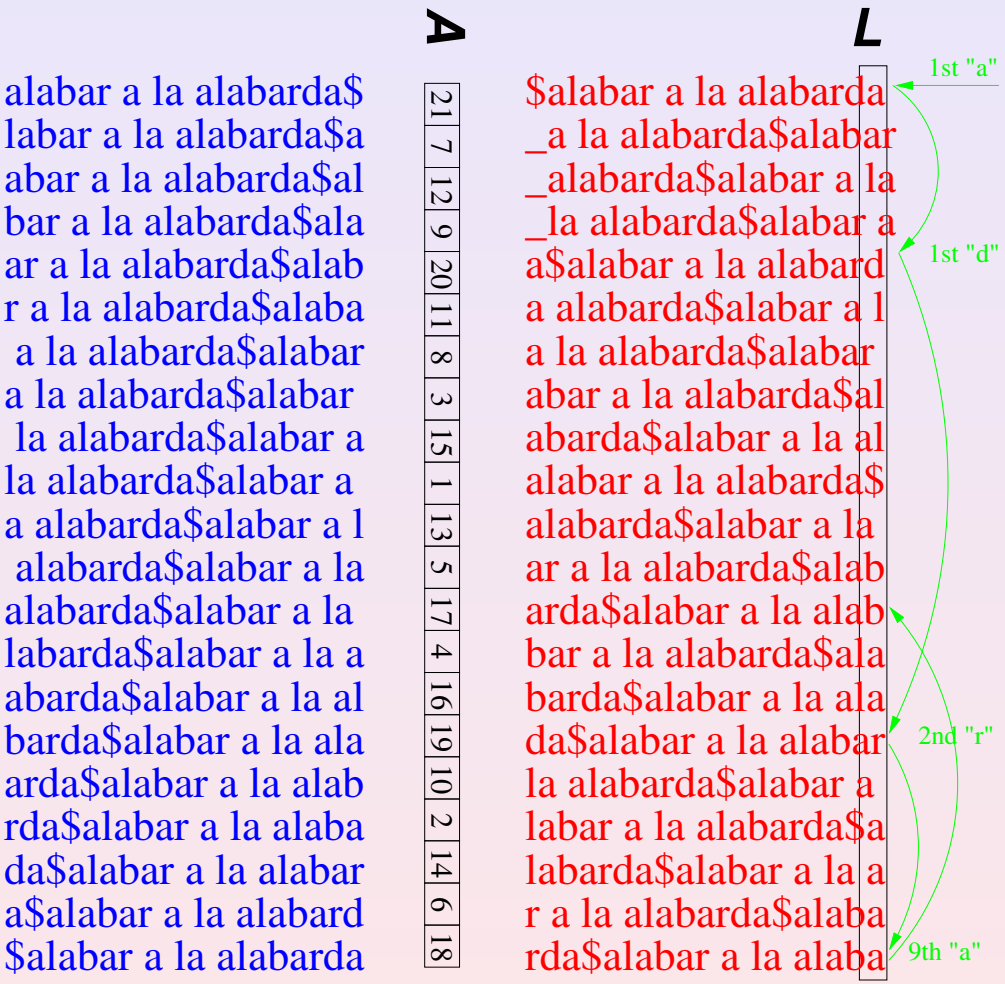
- A data structure over a text $T[1, n]$ supporting pattern searches.
- Most classical: suffix trees and suffix arrays.
- The latter are simpler and less space-consuming.
- (Still they do not support all the functionality of suffix trees.)



Full-text Indexes

- A way to simulate the suffix array $A[1, n]$ in little space uses the Burrows-Wheeler Transform (BWT) of T, T^{bwt}.
- It concatenates the symbols preceding each suffix of A.
- It is a reversible permutation of the string T.
- To reverse it, one needs a limited version of *rank* on T^{bwt}.

Full-text Indexes



T alabar a la alabarda$

T^{bwt} araadl ll$ bbaar aaaa

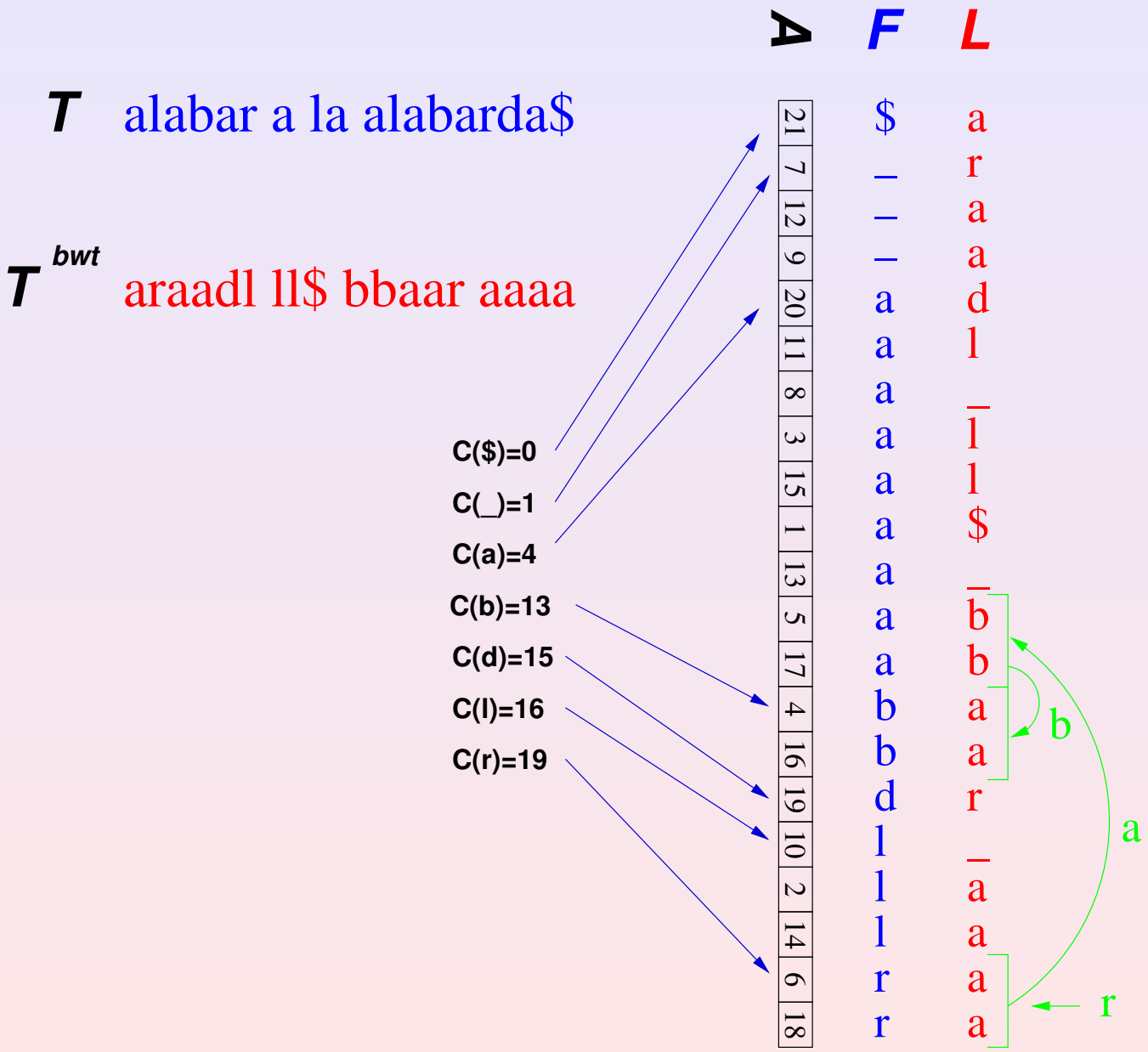
$c = L[i]$

$LF(i) = C[c]+rank_c(L,i)$

Full-text Indexes

- The search on the suffix array can be simulated via the so-called backward search.
- It uses (the general version of) *rank* queries on T^{bwt}.
- The suffix array interval of pattern P is identified in at most $2|P|$ applications of *rank*.

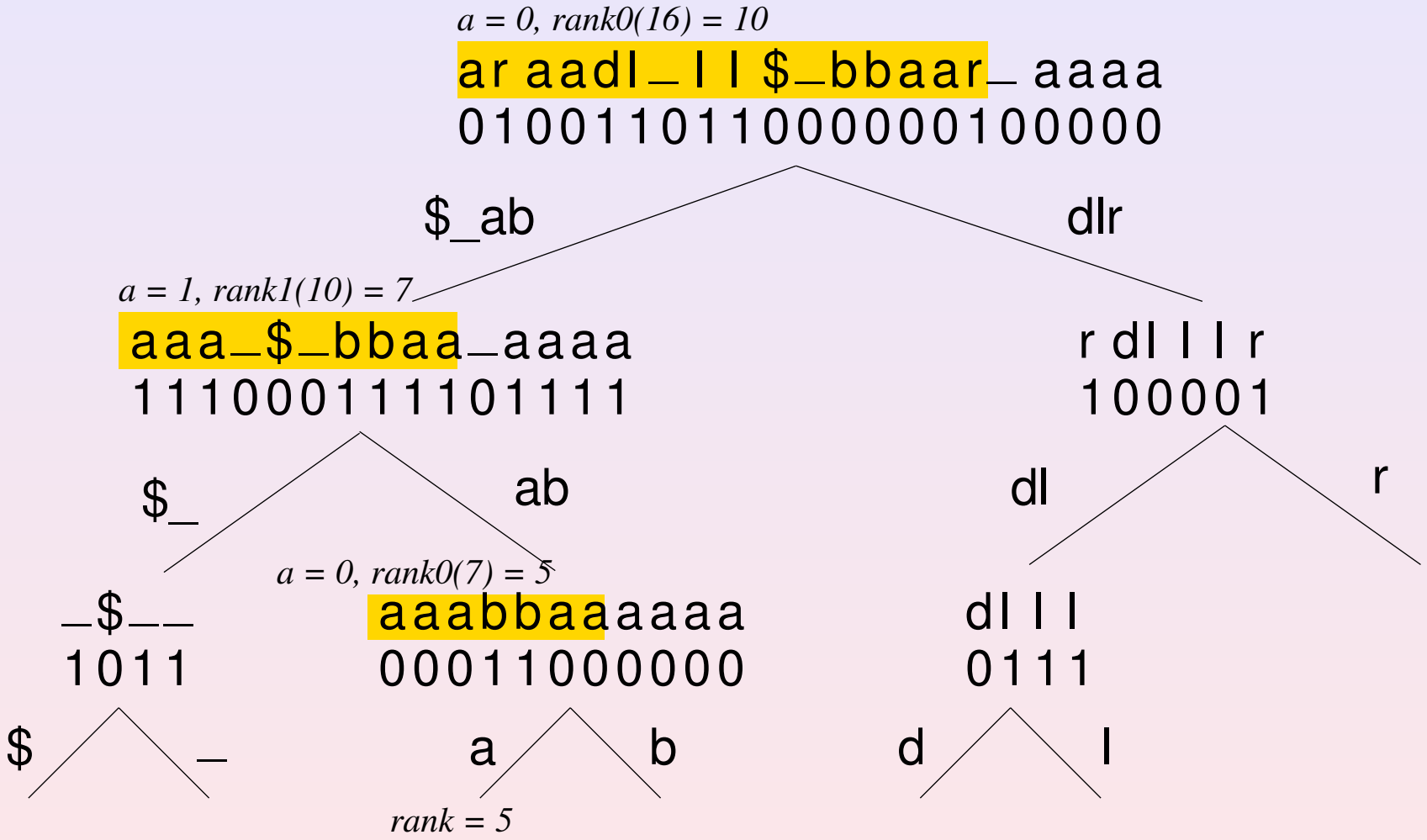
Full-text Indexes



Full-text Indexes

- This index is called the FM-index.
- In its modern version, it is basically a wavelet tree on T^{bwt}.
- It was proved that a wavelet tree,
  - Built on T^{bwt}.
  - Using zero-order compressed bitmaps.
  - So that such compression is “local” (not rare).
- reaches high-order compression of T.
- This is the compression achieved by ppm, bzip2, etc.
- It has to do with the predictability of the next symbol given k previous symbols.
- This simplifies the implementation of space-efficient FM-indexes.

Full-text Indexes



Positional Inverted Indexes

- Store the word offsets of each distinct word in a text.
- Used to display snippets and to solve phrase and proximity searches.
- Can be compressed to $nH_0(T)$ bits (word-based).
- Plus the compressed text, makes $2nH_0(T)$.

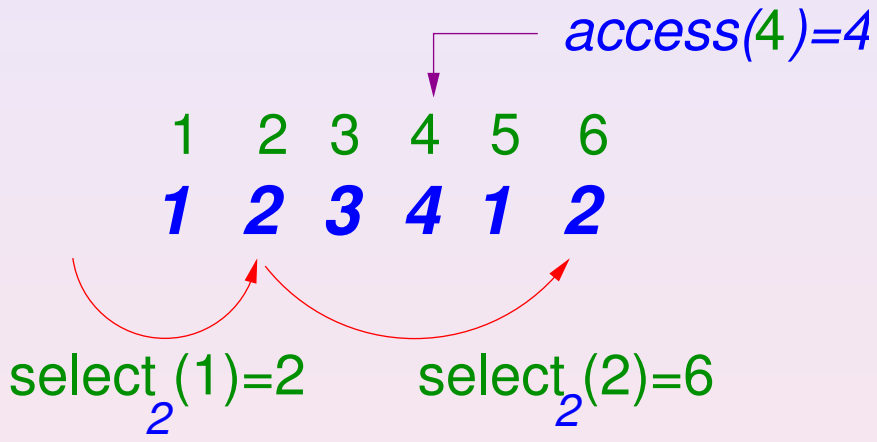
Positional Inverted Indexes

- A wavelet tree on the word identifiers uses $\approx nH_0(T)$ bits.
- It represents T: $T[i] = access(i)$.
- It represents the inverted index: the ith entry of the list of word w is $select_w(i)$.
- With *rank* queries, intersections (for phrases) can be done more efficiently.

Positional Inverted Indexes

to be or not to be

1	to	1,5
2	be	2,6
3	or	3
4	not	4



Graphs

- Directed graph $G(V, E)$ with $n = |V|$ nodes and $e = |E|$ edges.
- An adjacency list takes $n \log e + e \log n$ bits.
- Gives the neighbors of any node $v \in V$ efficiently.
- Reverse neighbors (i.e., who points to v)?
  - Represent the transposed G (and double the space).
- Is v connected to u?
  - Binary search on the list of v.

Graphs

- Concatenate all the adjacency lists

$$L = L(v_1) : L(v_2) : \ldots : L(v_n)$$

- Add sparse bitmap

$$B = 10^{|L(v_1)|}10^{|L(v_2)|}\ldots 10^{|L(v_n)|}$$

- Space: $n \log(e/n) + e \log n + O(n) + o(e \log n)$ (without compression).
- Neighbors of v_i, $L(v_i)$:

$$\begin{aligned} p &= select_1(B, i) - i \\ L(v_i)[j] &= access(L, p + j) \end{aligned}$$

Graphs

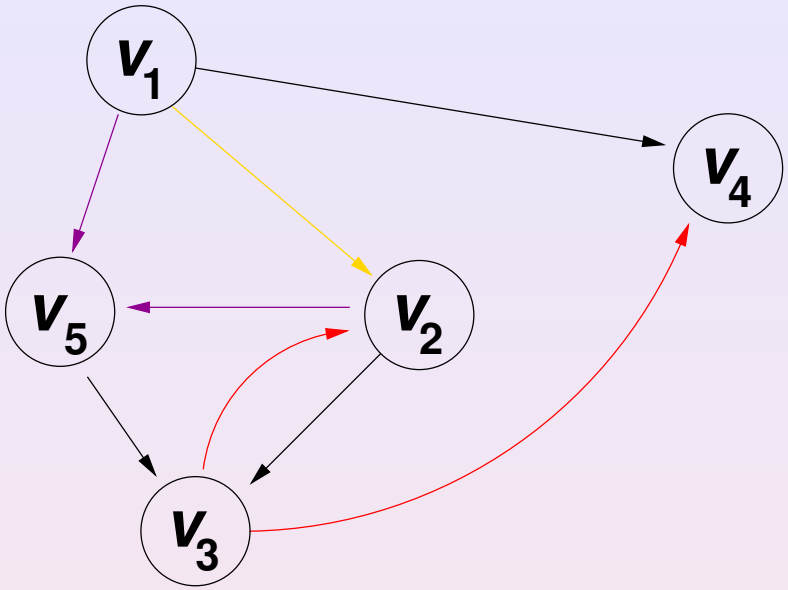
- Reverse neighbors of v_i, $R(v_i)$:

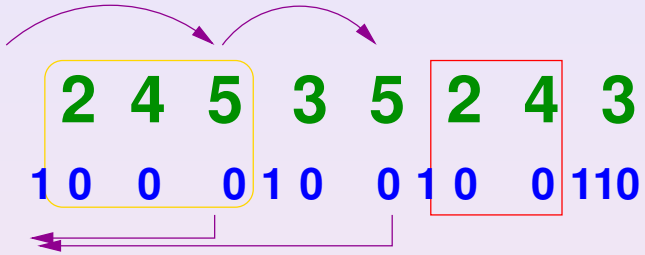
$$\begin{aligned} p &= select_i(L, j) \\ R(v_i)[j] &= select_0(B, p) - p \end{aligned}$$

- Is there a link from v_i to v_j ?

$$\begin{aligned} p_1 &= select_1(B, i) - i \\ p_2 &= select_1(B, i+1) - (i+1) \\ v_i \rightarrow v_j &\Leftrightarrow rank_j(L, p_2) - rank_j(L, p_1) > 0 \end{aligned}$$

Graphs





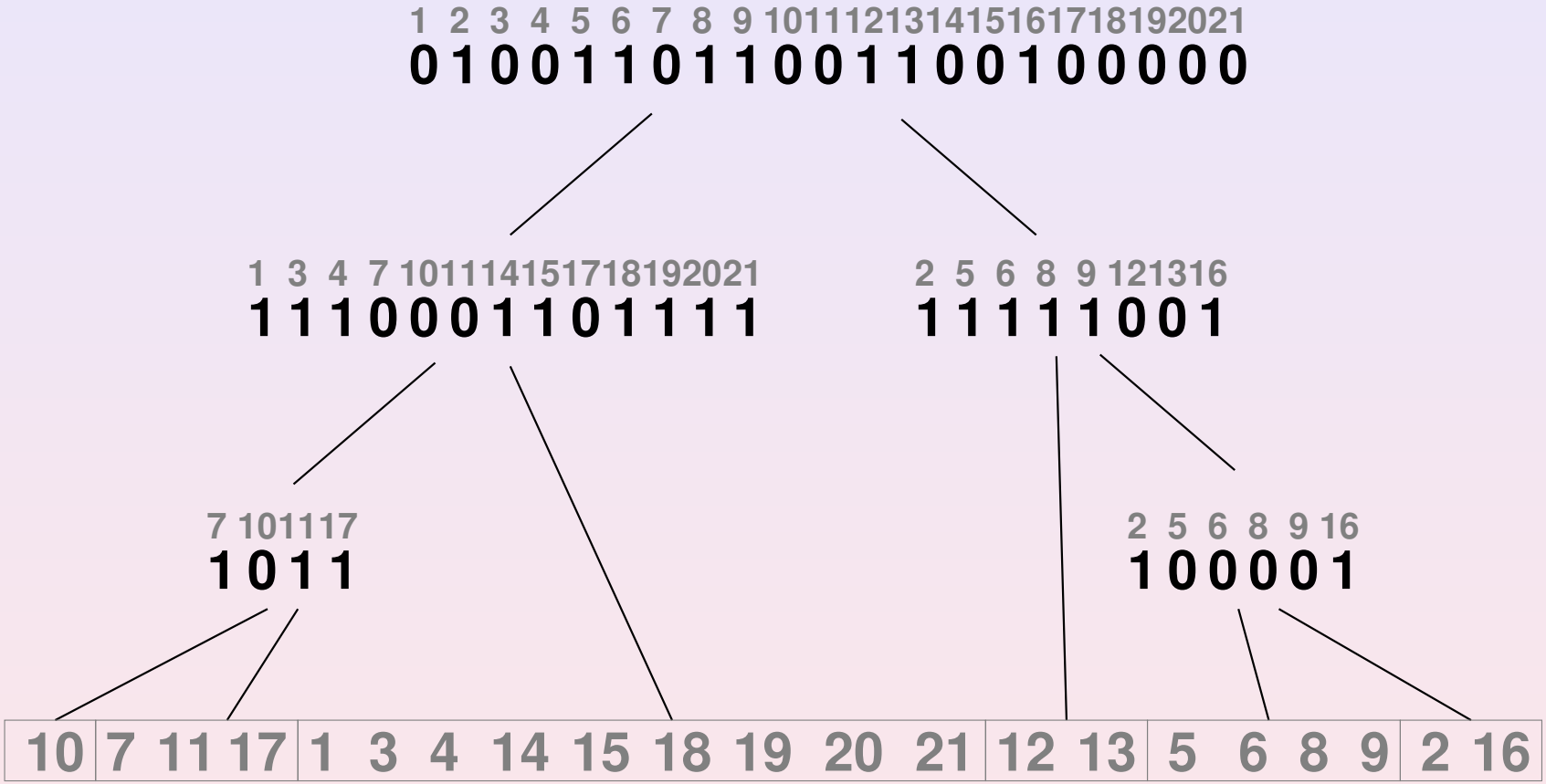
1: 2,4,5
2: 3,5
3: 2,4
4:
5: 3

Part II: Applications

Permutations

- Consider a permutation with few increasing runs.
- A wavelet-tree-like structure describes a mergesort process.
- This can be arranged in a Hu-Tucker tree shape.
- The total number of bits is less than $n(H+2)$.
- Here $H = \sum(n_i/n)\lg(n/n_i) \leq \lg \rho$,
- where n_i is the length of run i and ρ the number of runs.

Permutations



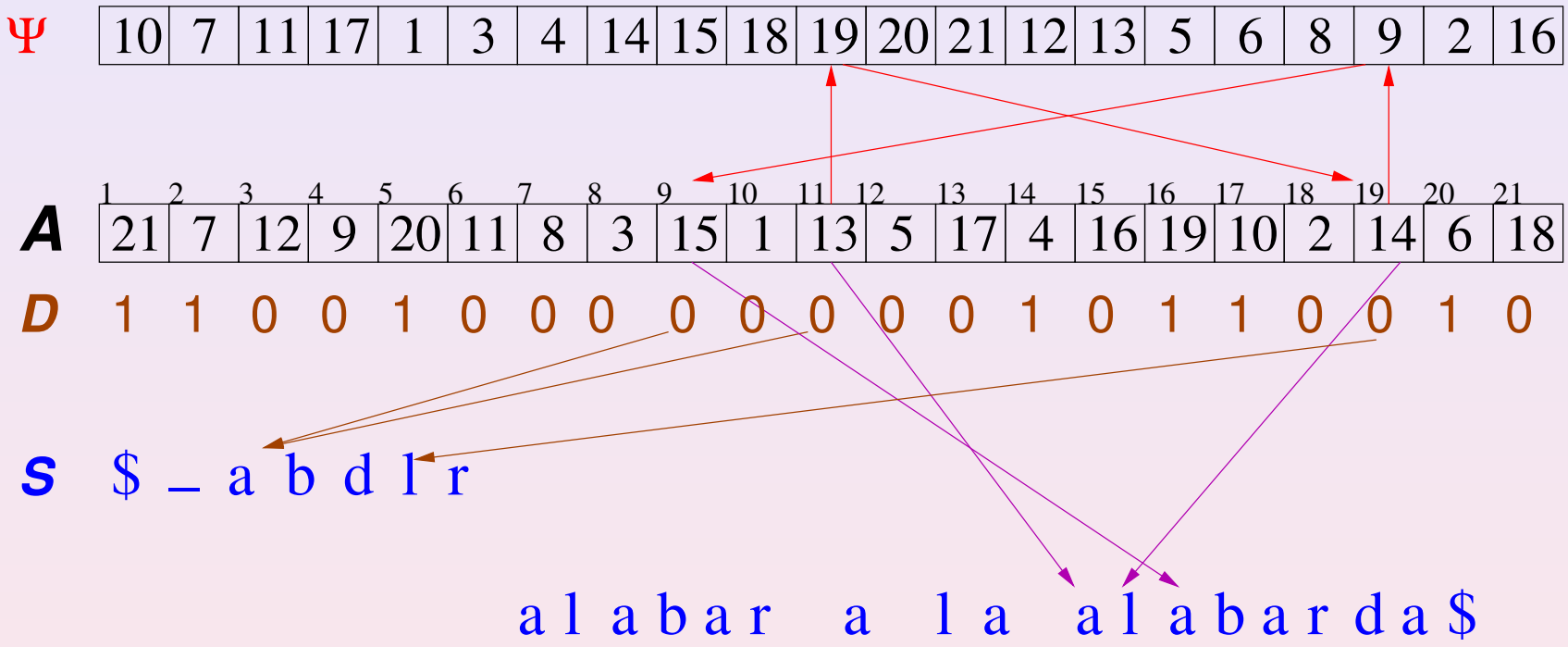
Permutations

- Any $\pi(i)$ can be computed with an upward tracking from i.
- Any $\pi^{-1}(i)$ can be computed with a downward tracking from i.
- Both can take $O(H+1)$ on average and $O(\lg \rho)$ in the worst case.
- But... do these permutations with few runs arise in practice?
  - E.g. Function Ψ has σ runs of entropy $\leq H_0(T)$.
  - This encoding enables a bidirectional compressed suffix array.

Permutations

- Function $\Psi(i) = A^{-1}[A[i] + 1]$ is used for compressed suffix arrays (CSAs).
- It is the inverse of $LF(i) = A^{-1}[A[i] - 1]$.
- A suffix array search is simulated in $O(m \log n)$ computations of Ψ.
- It has σ increasing runs, with entropy $H \leq H_0(T)$.
- A representation as a permutation enables, for example, bidirectional CSAs.
- The space is at most $n(H_0(T) + 2)$ bits.

Permutations



Generic Numeric Sequences

- Basic algorithmic problems on numeric sequences.
- Range quantile query:
  - Preprocess array $A[1, n]$ on $[1, U]$ so that, later,
  - given $[l, r]$ and k,
  - retrieve the kth smallest element in $A[l, r]$
- Best solution (after much effort): $O(n \lg n)$ bits and $O(\lg n / \lg \lg n)$ time.
- We can assume $U \leq n$.

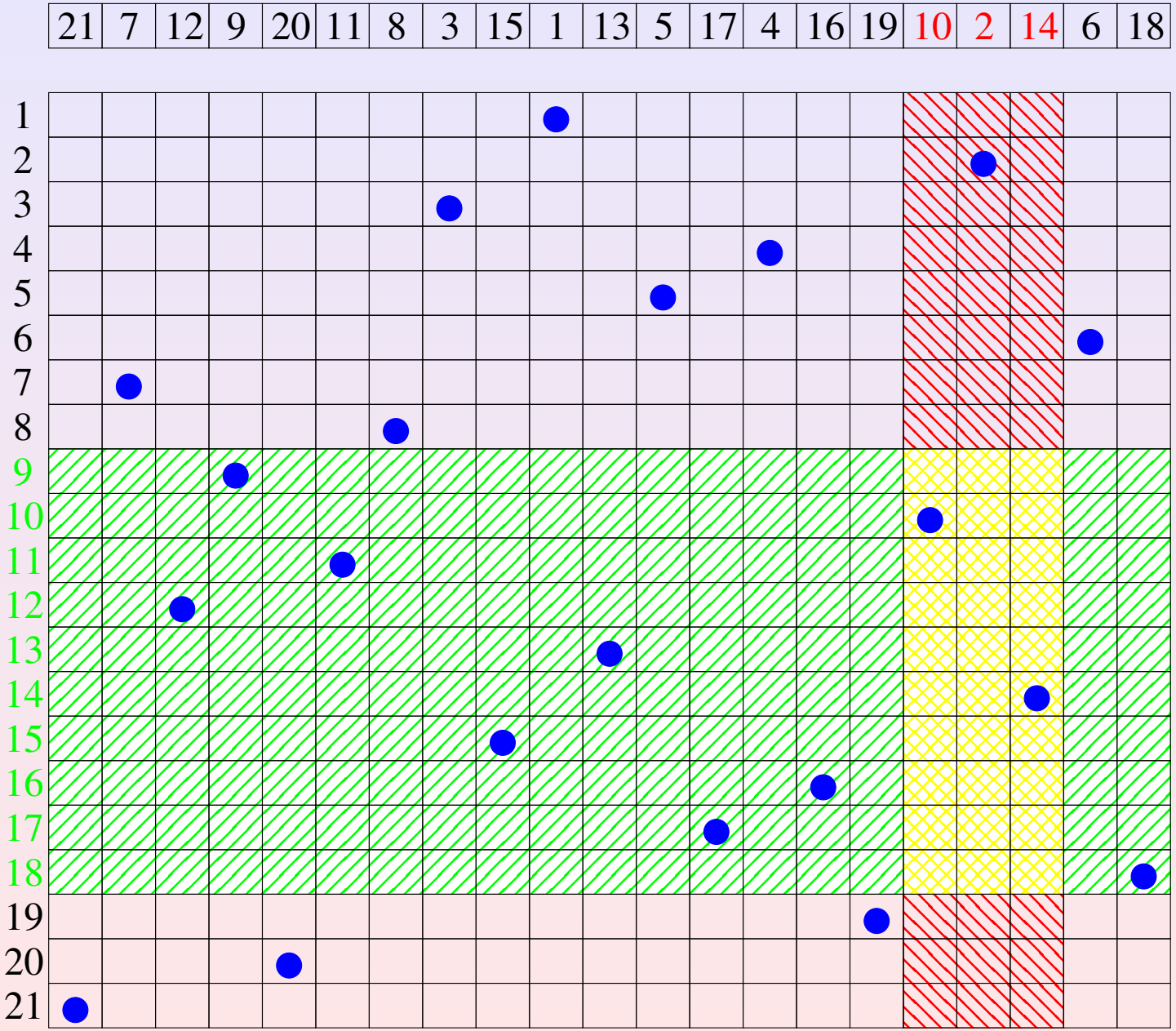
Generic Numeric Sequences

- Range quantile query on wavelet trees:
  - Start at the root v with interval $[l, r]$.
  - If $z = rank_0(B, l, r) \geq k$, the answer is on the left child:
  - $\rightarrow$ continue on left child, mapping $[l, r]$.
  - Otherwise, the answer is on the right child:
  - $\rightarrow$ continue on the right with $k = k - z$, mapping $[l, r]$.
- $n \lg U + o(n)$ bits, replacing the array.
- $O(\lg U)$ time.

Generic Numeric Sequences

- Other problems:
  - Given $[l, r]$ and v, give the smallest number $\geq v$ in $[l, r]$
  - Given $[l, r]$ and v, give the leftmost number $\geq v$ in $[l, r]$
- They have clear geometric counterparts (e.g., list range points in order).
- Geometric problems like dominance and visibility reduce to these.

Generic Numeric Sequences

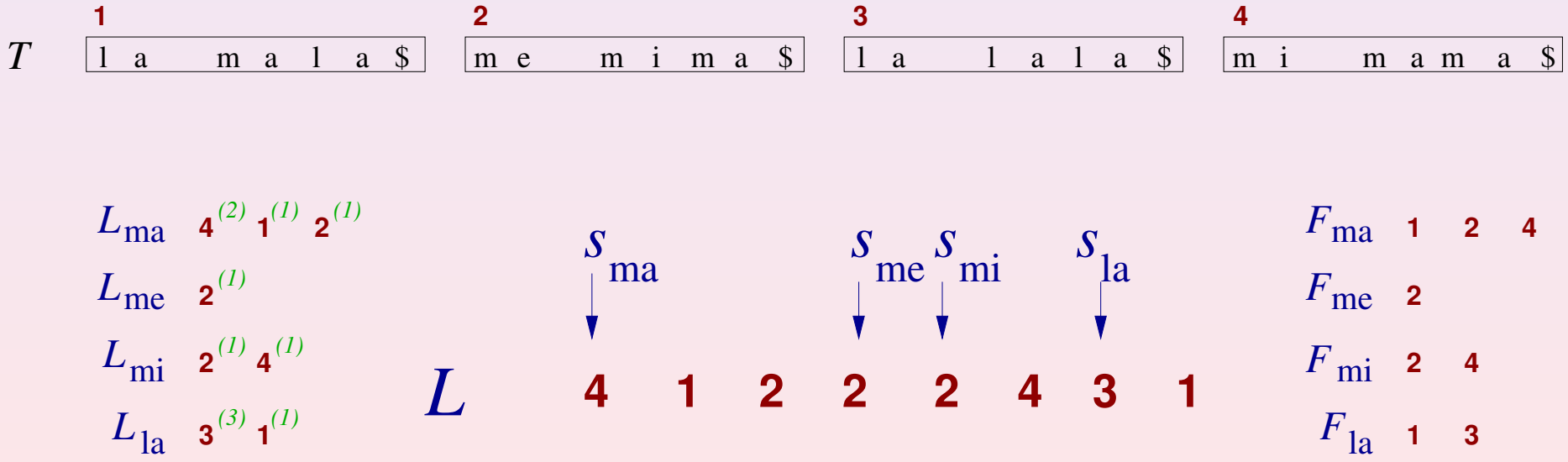


Non-positional Inverted Indexes

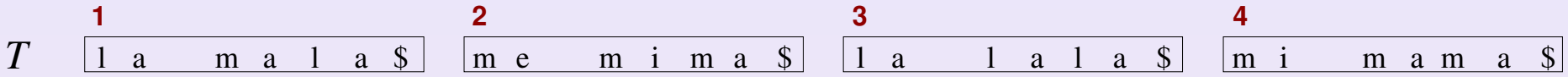
- For each term they store the documents where it appears and its frequency.
- Useful to have them sorted by document identifier:
  - For conjunctive queries (intersections).
- Useful to have them sorted by decreasing frequency:
  - For ranked queries (bag of words).

Non-positional Inverted Indexes

- A wavelet tree concatenating the lists in frequency order...
- ... can support both orderings simultaneously:
  - By decreasing frequency: access to the sequence.
  - By increasing doc id: quantile queries.



Non-positional Inverted Indexes



L_{ma} 4(2) 1(1) 2(1)

L_{me} 2(1)

L_{mi} 2(1) 4(1)

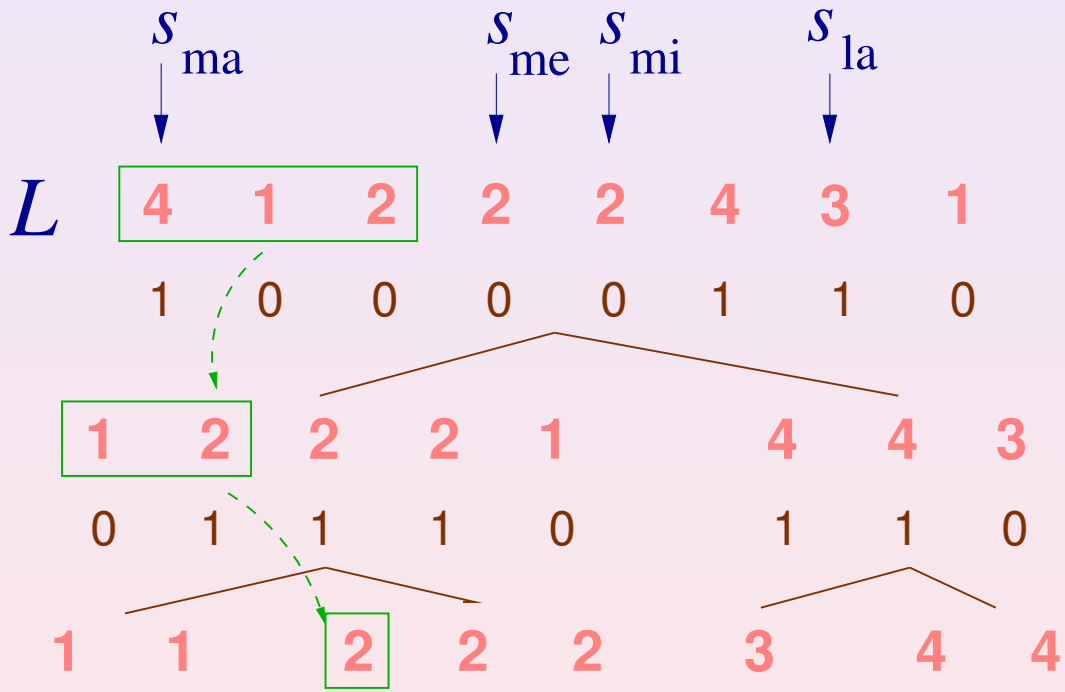
L_{la} 3(3) 1(1)

F_{ma} 1 **2** 4

F_{me} 2

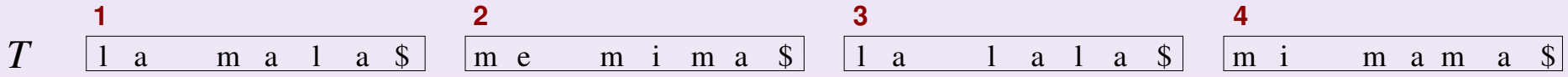
F_{mi} 2 4

F_{la} 1 3



Non-positional Inverted Indexes

- It can also simulate inverted lists of ranges of terms.
- Useful for on-the-fly stemming, prefix searches, thesaurus expansions.



L_{ma} $4^{(2)}$ $1^{(1)}$ $2^{(1)}$

L_{me} $2^{(1)}$

L_{mi} $2^{(1)}$ $4^{(1)}$

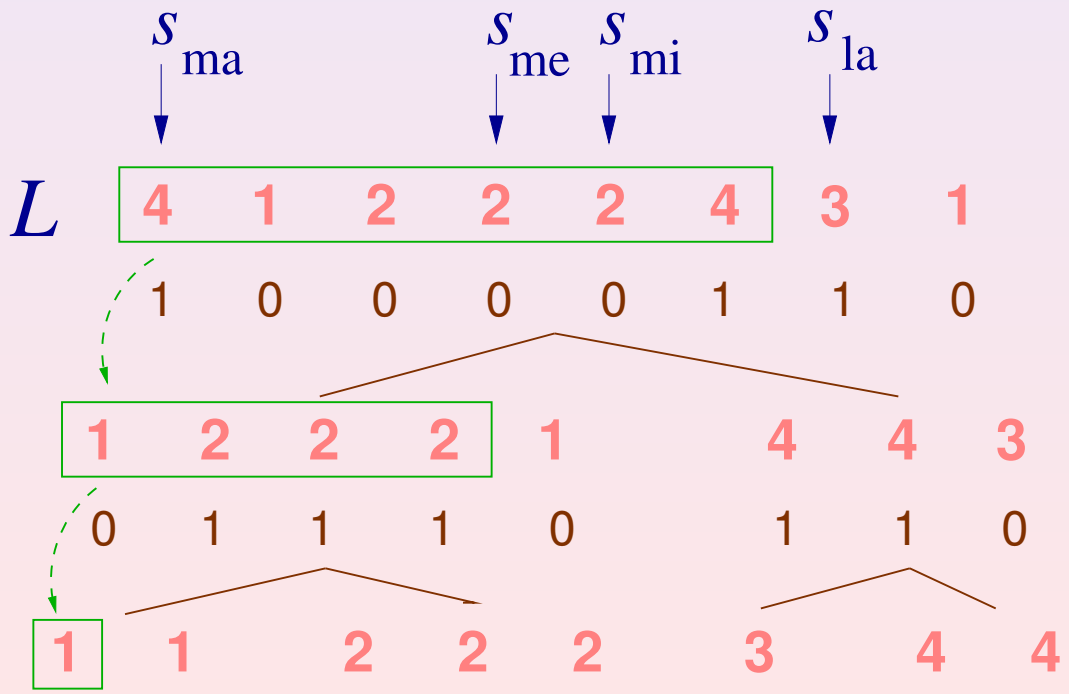
L_{la} $3^{(3)}$ $1^{(1)}$

F_{ma} 1 2 4

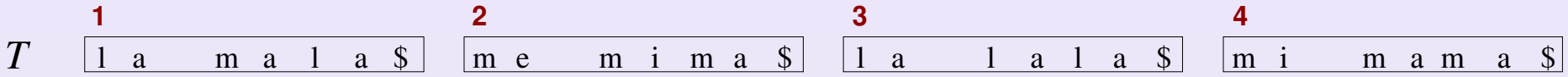
F_{me} 2

F_{mi} 2 4

F_{la} 1 3



Non-positional Inverted Indexes



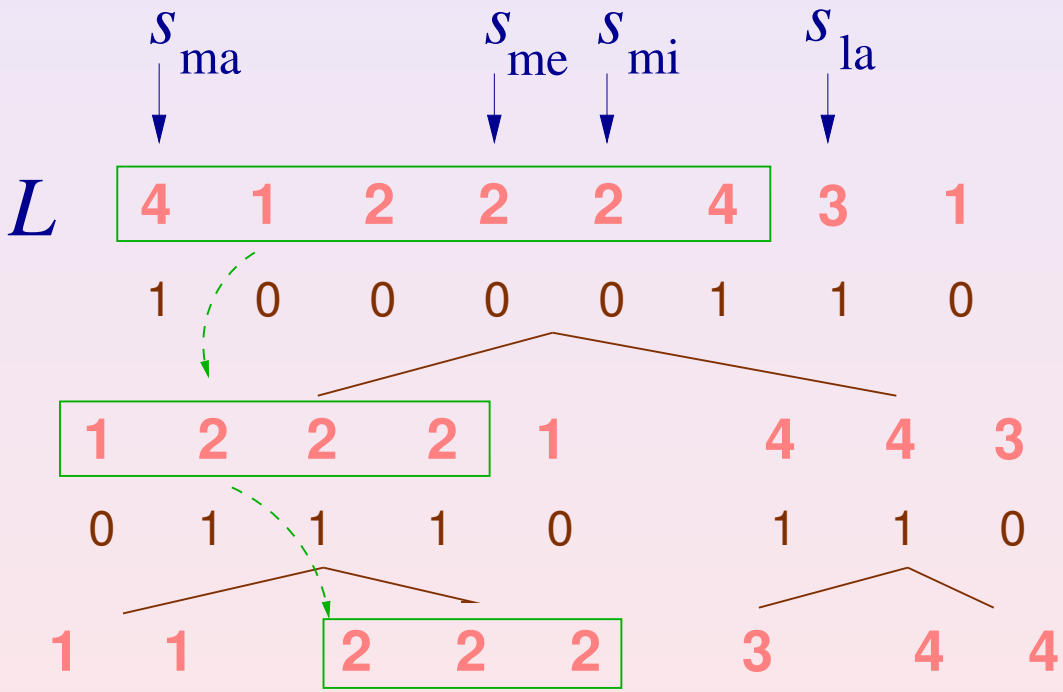
L_{ma} $4^{(2)}$ $1^{(1)}$ $2^{(1)}$

L_{me} $2^{(1)}$

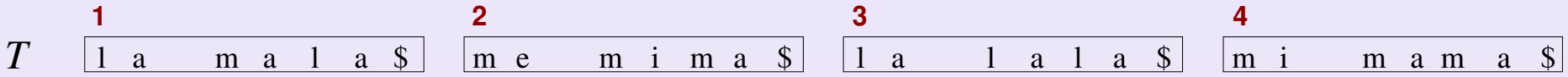
L_{mi} $2^{(1)}$ $4^{(1)}$

L_{la} $3^{(3)}$ $1^{(1)}$

F_{ma}	1	2	4
F_{me}	2		
F_{mi}	2	4	
F_{la}	1	3	



Non-positional Inverted Indexes



L_{ma} $4^{(2)}$ $1^{(1)}$ $2^{(1)}$

L_{me} $2^{(1)}$

L_{mi} $2^{(1)}$ $4^{(1)}$

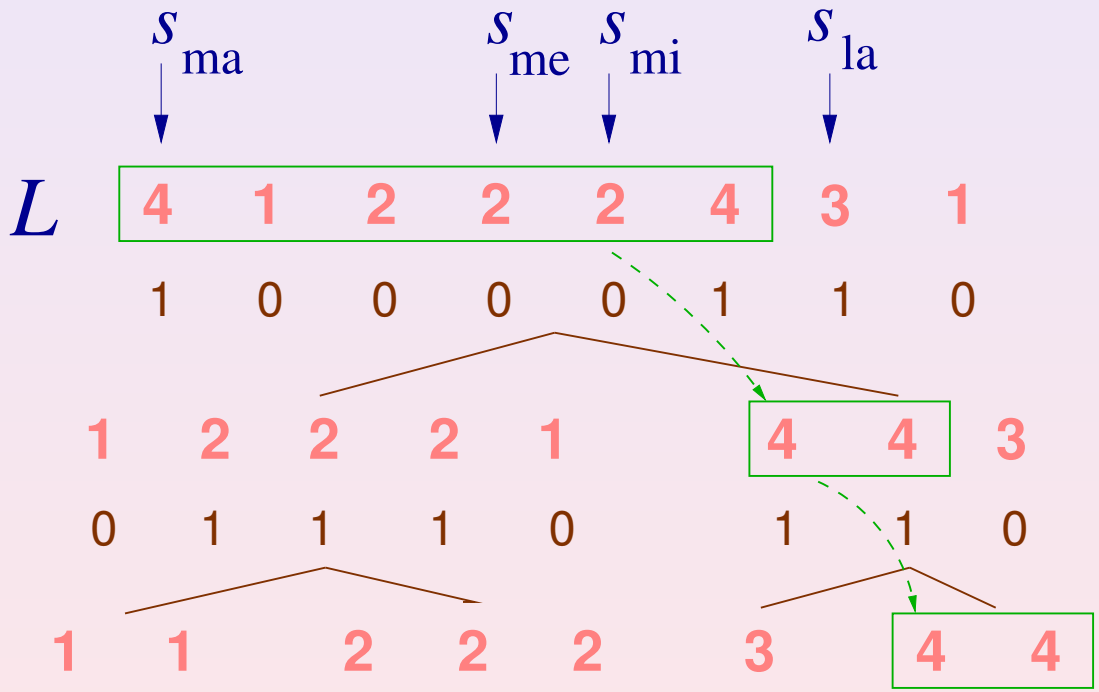
L_{la} $3^{(3)}$ $1^{(1)}$

F_{ma} 1 2 4

F_{me} 2

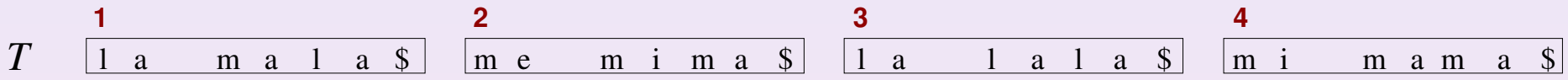
F_{mi} 2 4

F_{la} 1 3



Non-positional Inverted Indexes

- It can intersect lists with a fast native algorithm.
- Backtracking as long as no list is empty.
- Easy to generalize to thresholded queries.



L_{ma} $4^{(2)}$ $1^{(1)}$ $2^{(1)}$

L_{me} $2^{(1)}$

L_{mi} $2^{(1)}$ $4^{(1)}$

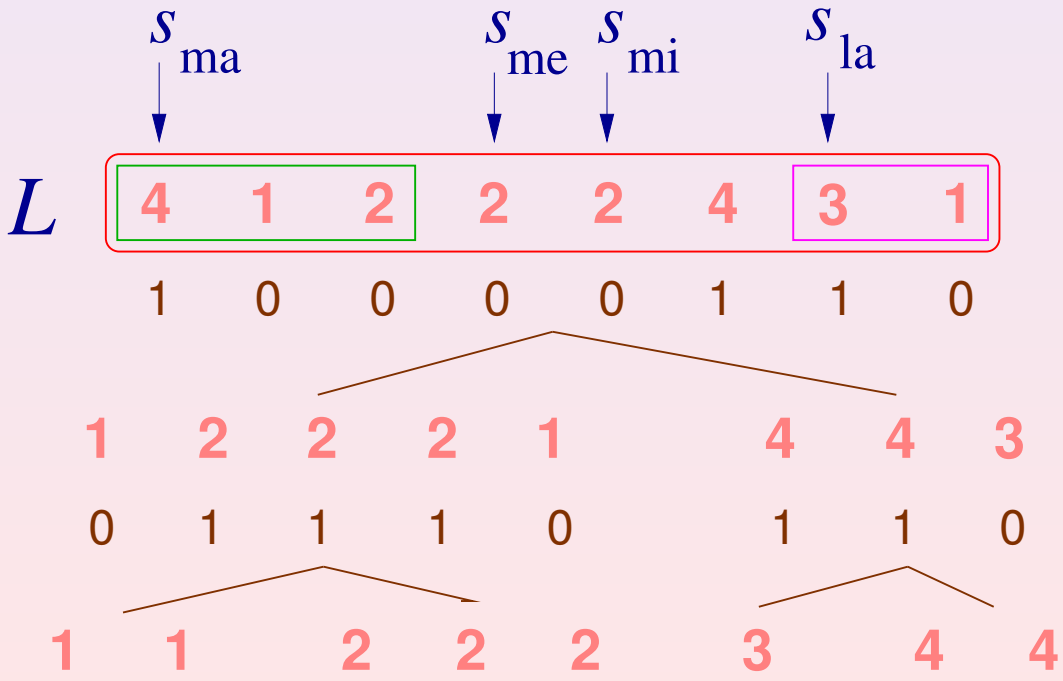
L_{la} $3^{(3)}$ $1^{(1)}$

F_{ma} 1 2 4

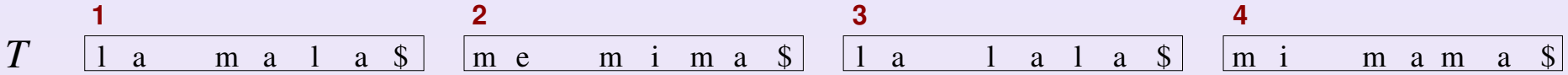
F_{me} 2

F_{mi} 2 4

F_{la} 1 3



Non-positional Inverted Indexes



L_{ma} $4^{(2)}$ $1^{(1)}$ $2^{(1)}$

L_{me} $2^{(1)}$

L_{mi} $2^{(1)}$ $4^{(1)}$

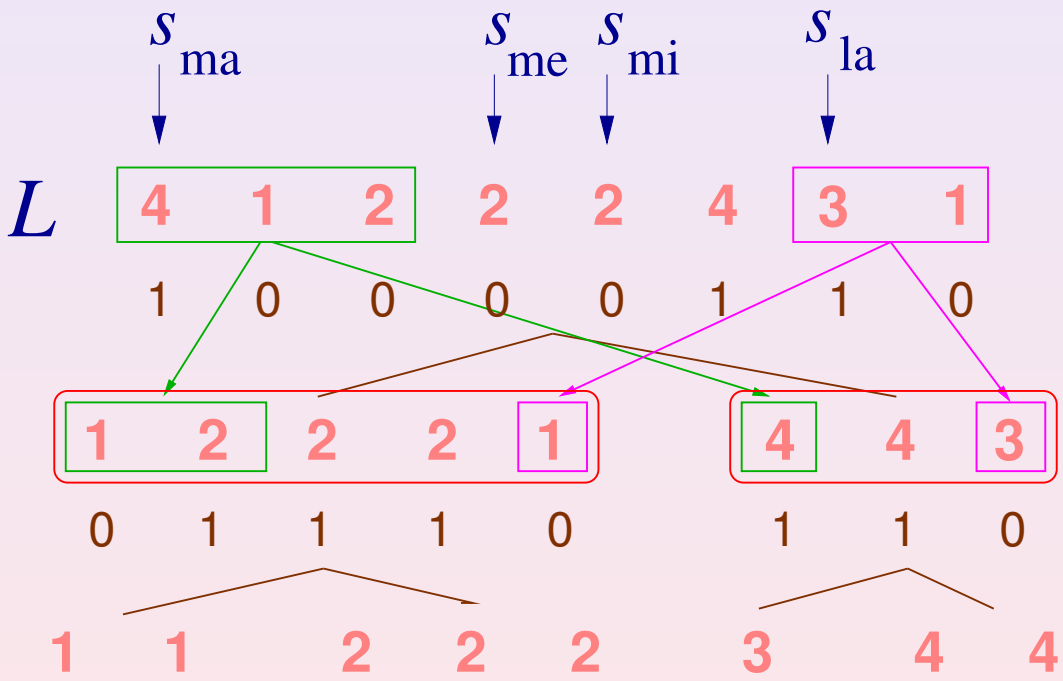
L_{la} $3^{(3)}$ $1^{(1)}$

F_{ma} 1 2 4

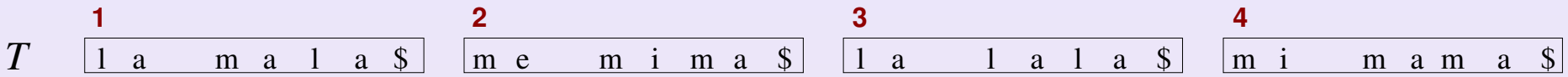
F_{me} 2

F_{mi} 2 4

F_{la} 1 3



Non-positional Inverted Indexes



L_{ma} $4^{(2)}$ $1^{(1)}$ $2^{(1)}$

L_{me} $2^{(1)}$

L_{mi} $2^{(1)}$ $4^{(1)}$

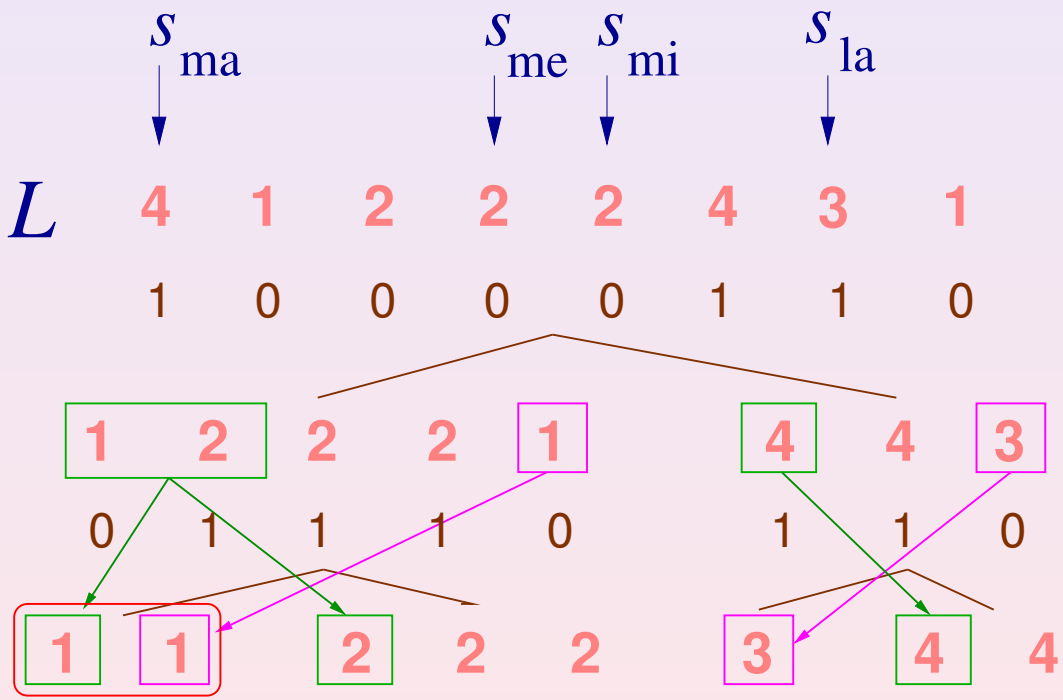
L_{la} $3^{(3)}$ $1^{(1)}$

F_{ma} 1 2 4

F_{me} 2

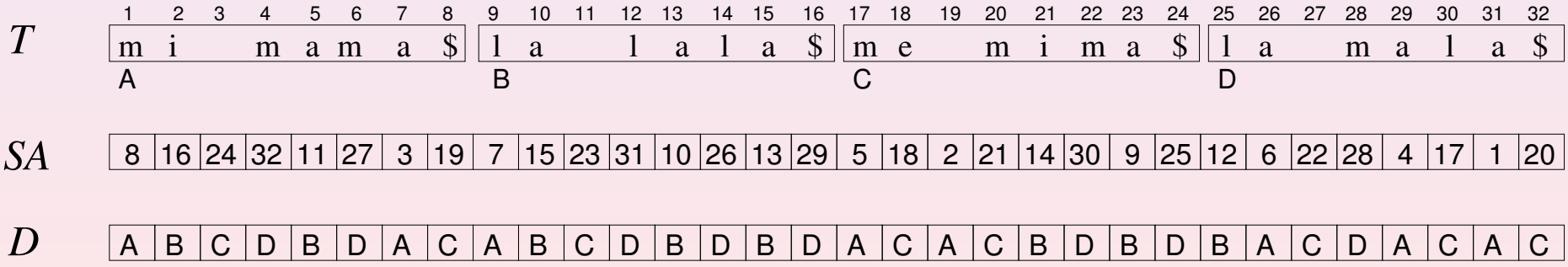
F_{mi} 2 4

F_{la} 1 3



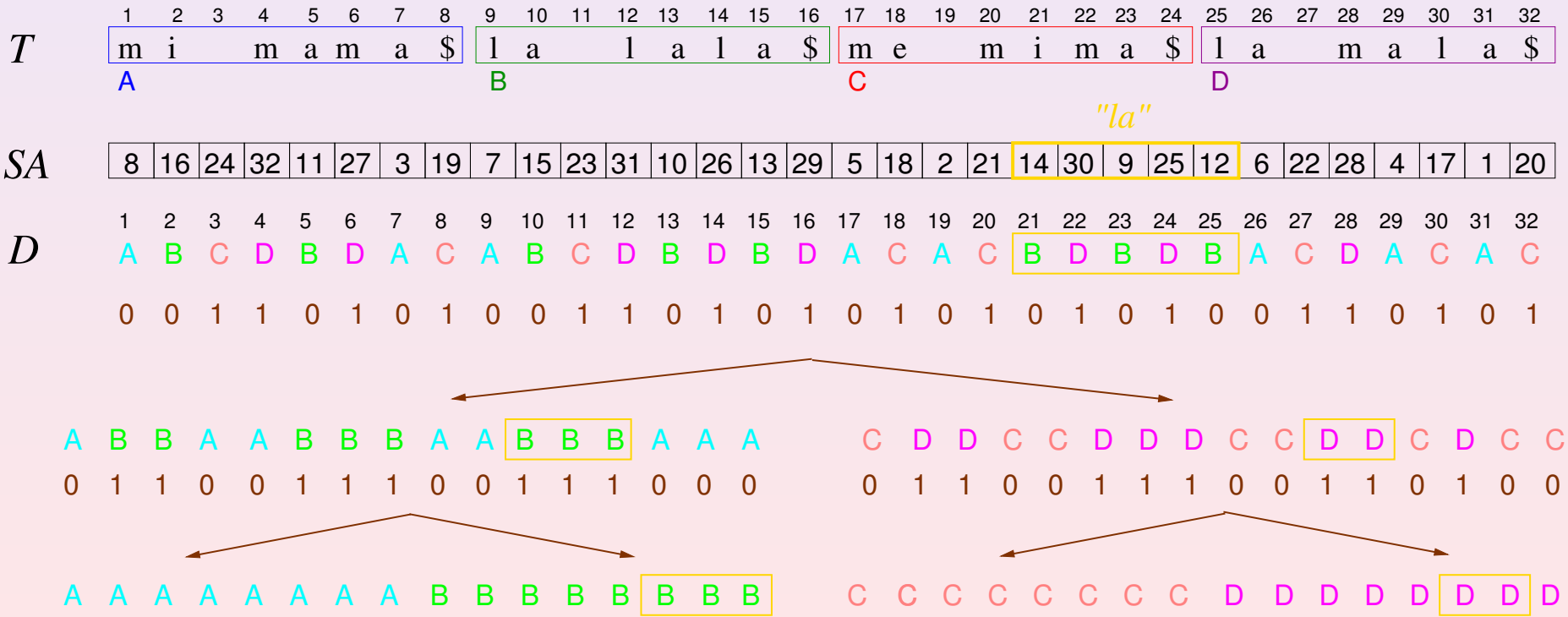
Document Retrieval Indexes

- Document retrieval on general string collections.
- Given a pattern P and a collection of texts $T_1, \ldots, T_D$,
  - In which documents does P appear?
  - With which frequencies?
  - Give me only the k highest-frequency documents.
- One of the main tools: the document array.



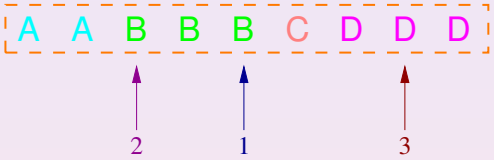
Document Retrieval Indexes

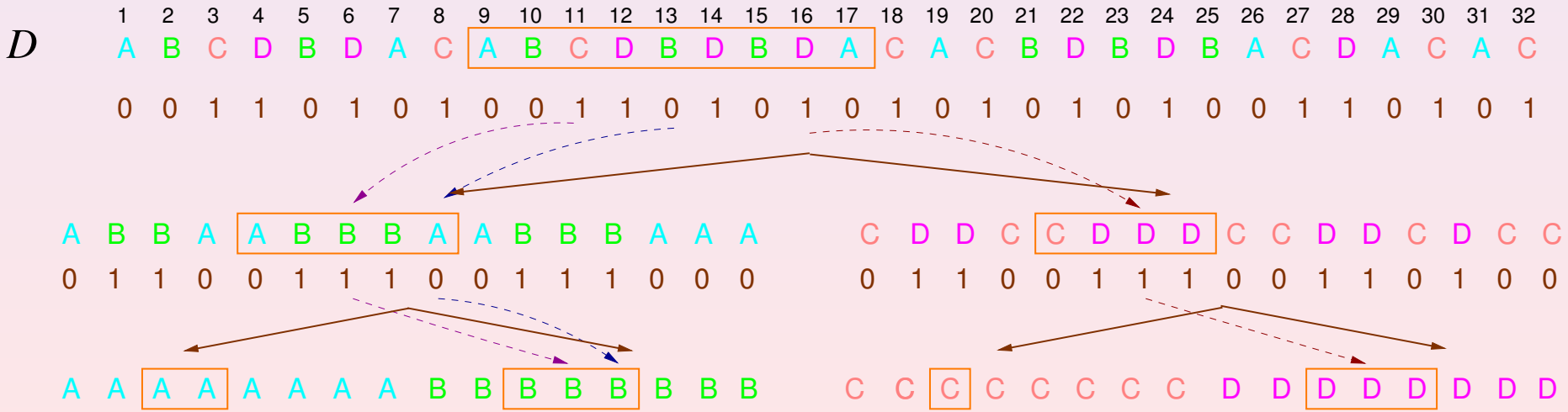
- Document listing (with frequencies) reduces to backtracking on the wavelet tree.
- $O(d \lg(D/d))$ time to report the d documents.



Document Retrieval Indexes

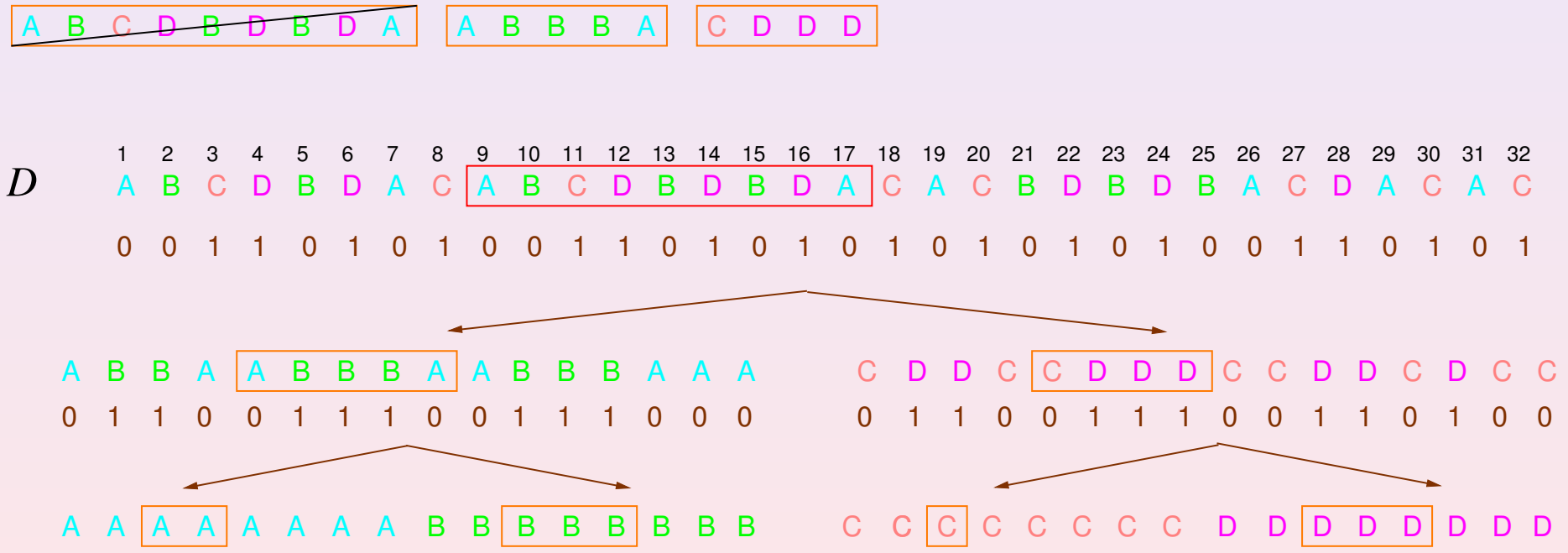
- Top-k document retrieval via range quantile queries.
- Find consecutively more refined quantiles...
- ... until the kth document seen occurs more times than the inter-quantile distance explored up to now.





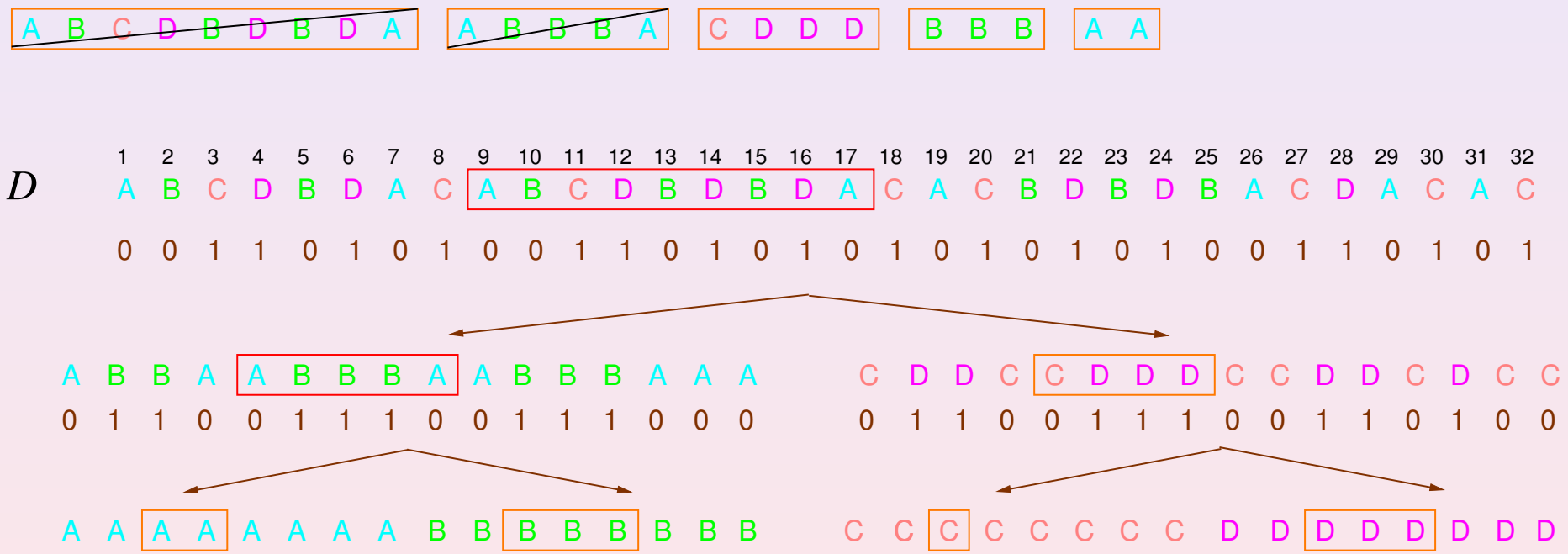
Document Retrieval Indexes

- Top-k document retrieval via prioritized wavelet tree traversal.
- Set up a priority queue sorted by interval lengths...



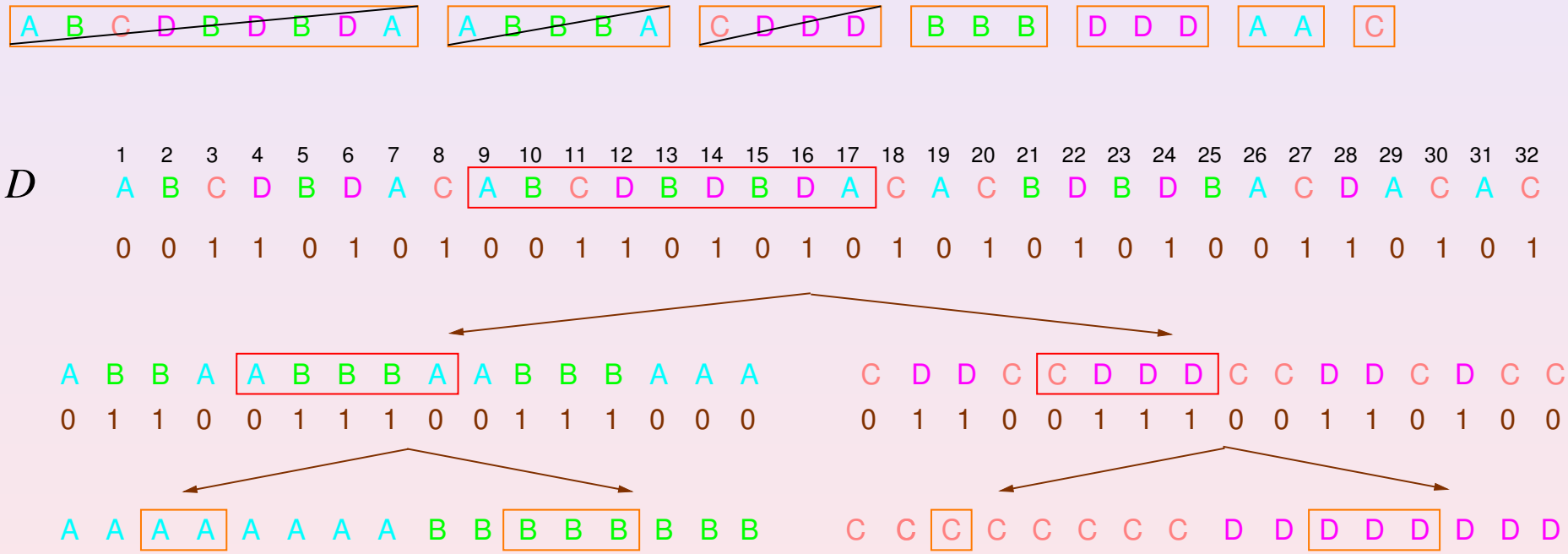
Document Retrieval Indexes

- Extract each new interval and insert its two children in the queue.



Document Retrieval Indexes

- The leaves come out in the proper order, stop after getting k.

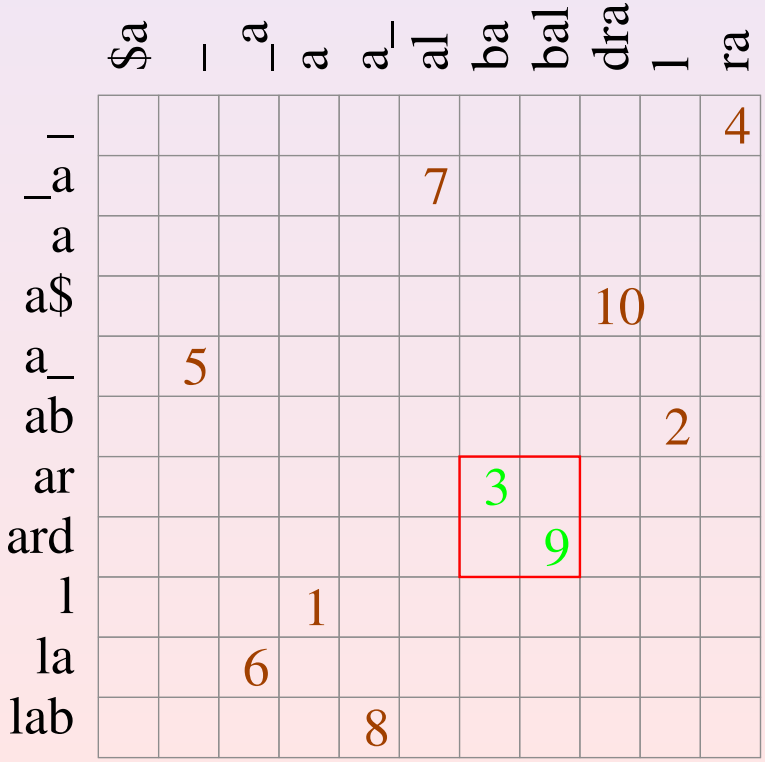


Part II: Applications

Discrete Grids

- Apart from obvious computational geometry applications...
- ... discrete grids model many subproblems in other areas.
- A typical one: pair prefixes and suffixes in grammar-based and Lempel-Ziv-based compressed indexes.

a.l.ab.ar. .a .la. a.lab.ard.a$



Discrete Grids

- More complex geometric problems: points have values in $[1, V]$.
- Find, in rectangular ranges:
  - Sum of values.
  - Average of values.
  - Variance of values.
  - Minima or maxima of values.
  - Quantiles of values.
  - Majorities.

Discrete Grids

- Some are solved by adding one-dimensional structures to the wavelet tree nodes.
- For example, consider finding the minimum value in a range, 2D-RMQ.
- We can assume n points on an $n \times n$ grid, and $V \leq n$.
- There exists a (1D) RMQ structure that, on array $A[1, n]$ of numbers,
  - Requires $2n + o(n)$ bits of space.
  - Answers queries in constant time.
  - Does not access A.

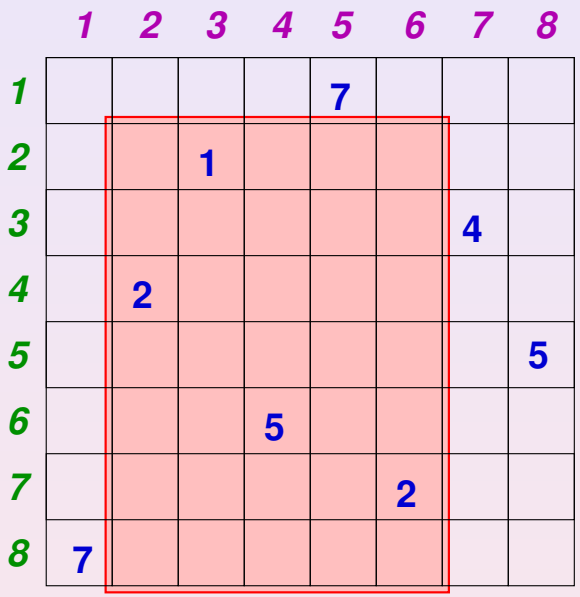
Discrete Grids

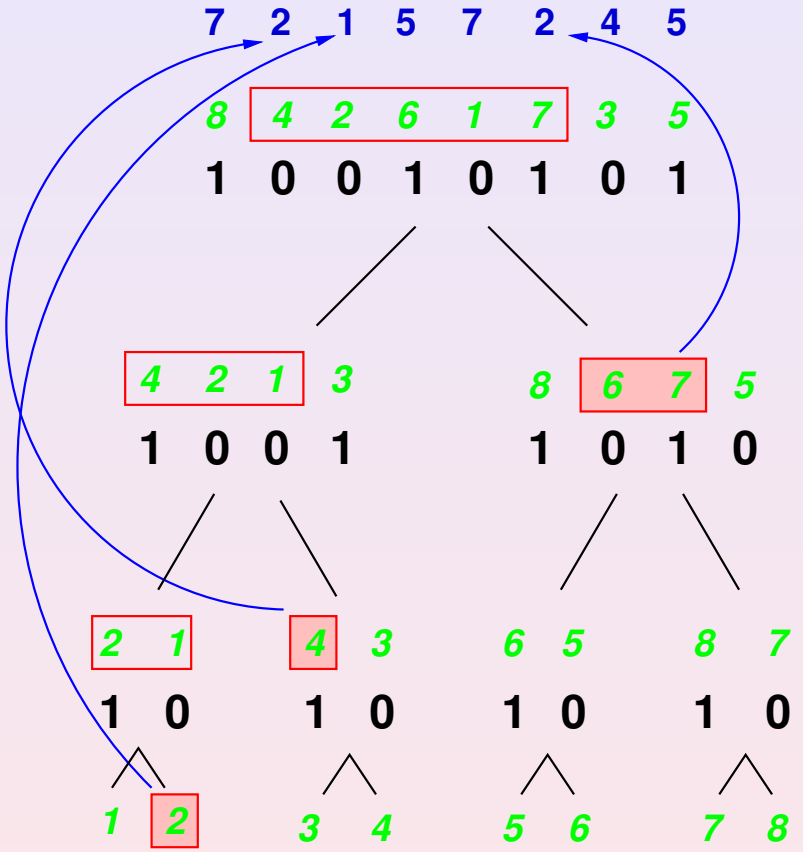
- Use a wavelet tree for the points.
- Store the values aligned to the root bitmap, in $V[1, n]$.
- Subsequences $V_v[1, n_v]$ of V correspond to $S_v[1, n_v]$.
- We store a one-dimensional RMQ structure per wavelet tree node, for $V_v[1, n_v]$.
- Overall space is $O(n \lg n)$

Discrete Grids

- A two-dimensional query range translates into $O(\lg n)$ intervals inside wavelet tree nodes.
- The minimum in the two-dimensional range is the minimum of the minima of those intervals of V_v sequences.
- Use the one-dimensional RMQ structure on each involved interval.
- Project each one towards the root to find the values, and choose the minimum.
- Overall time $O(\lg^{1+\epsilon} n)$.

Discrete Grids





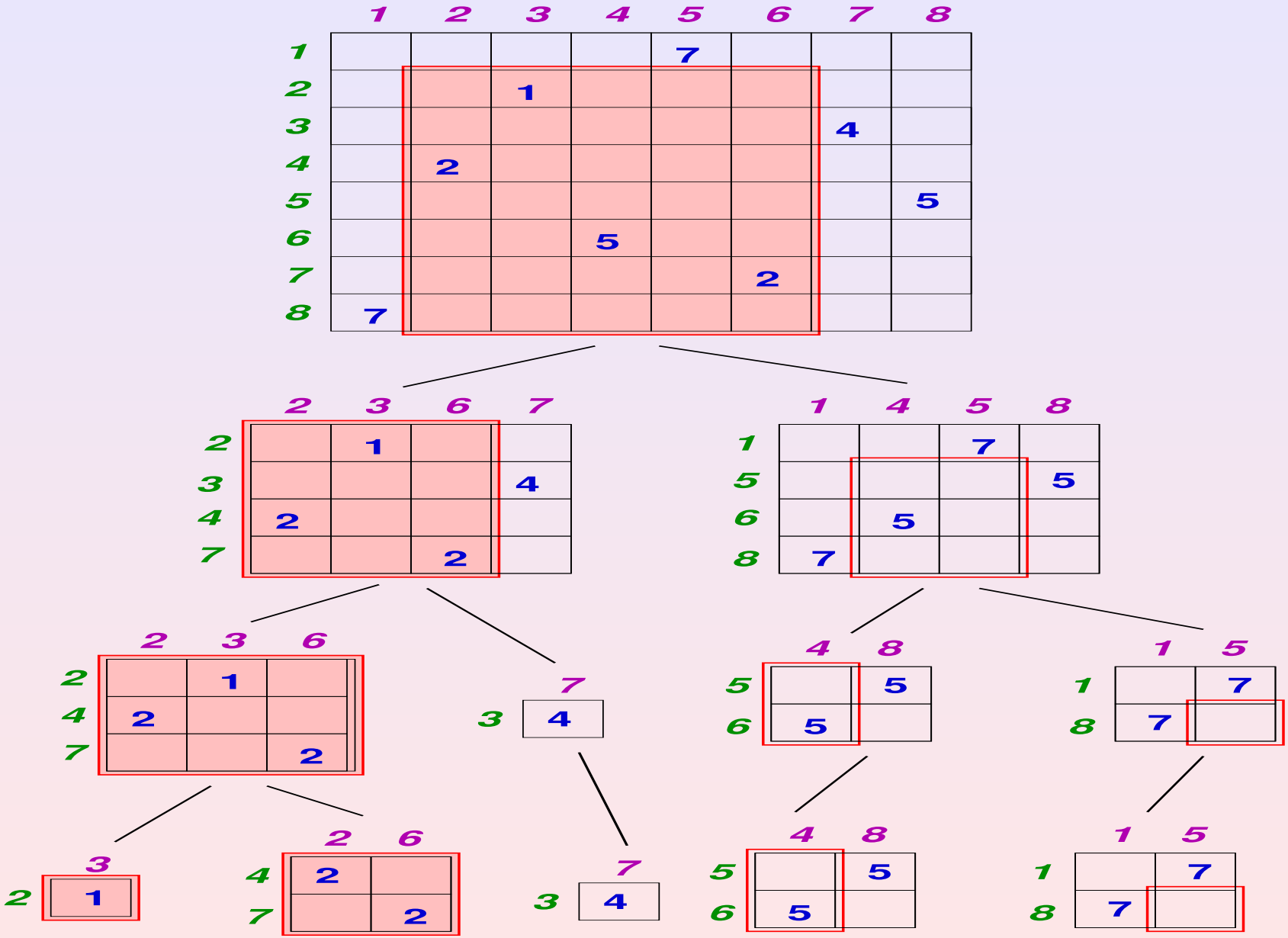
Discrete Grids

- A more complicated problem: two-dimensional range quantiles.
- We can build a wavelet tree of grids.
- These have only points, not values.
- Each wavelet tree node includes the points whose values are in a range.
- Two bitmaps tell which x and y coordinates belong to the left/right child.
- Overall space is $O(n \lg^2 n)$

Discrete Grids

- The algorithm is similar to that for one dimension.
- We count how many points are in the range mapped to the left child.
- If there are k or more, we go to the left child.
- Else, we go to the right child, subtracting the count from k.
- $O(\lg n)$ counting queries: $O(\lg^2 n / \lg \lg n)$ time.

Discrete Grids



Discrete Grids

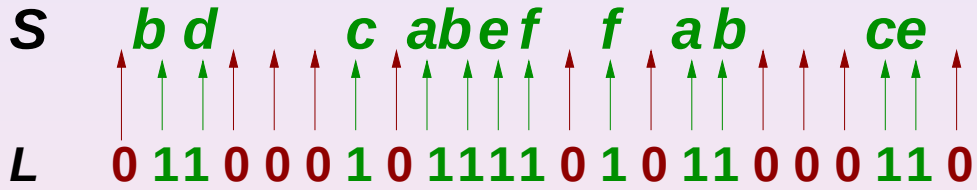
- Top-k queries on two dimensions.
- First find the $O(\lg n)$ wavelet tree intervals.
- Store them in a priority queue, sorted by minimum value.
- Take the minimum, report it, and insert the two subintervals it splits.
- Until having extracted the k minima.
- $O((k + \lg n) \lg^{\epsilon} n)$ time.

Binary Relations

- A generalization of graphs.
- We have t pairs relating n objects with σ labels.
- We can ask who is related with an object, or with a label, or if a pair is related.
- We can also ask, e.g., how many connections are there between a range of objects and a range of labels.
- Or list objects/labels related to a range of labels/objects.

Binary Relations

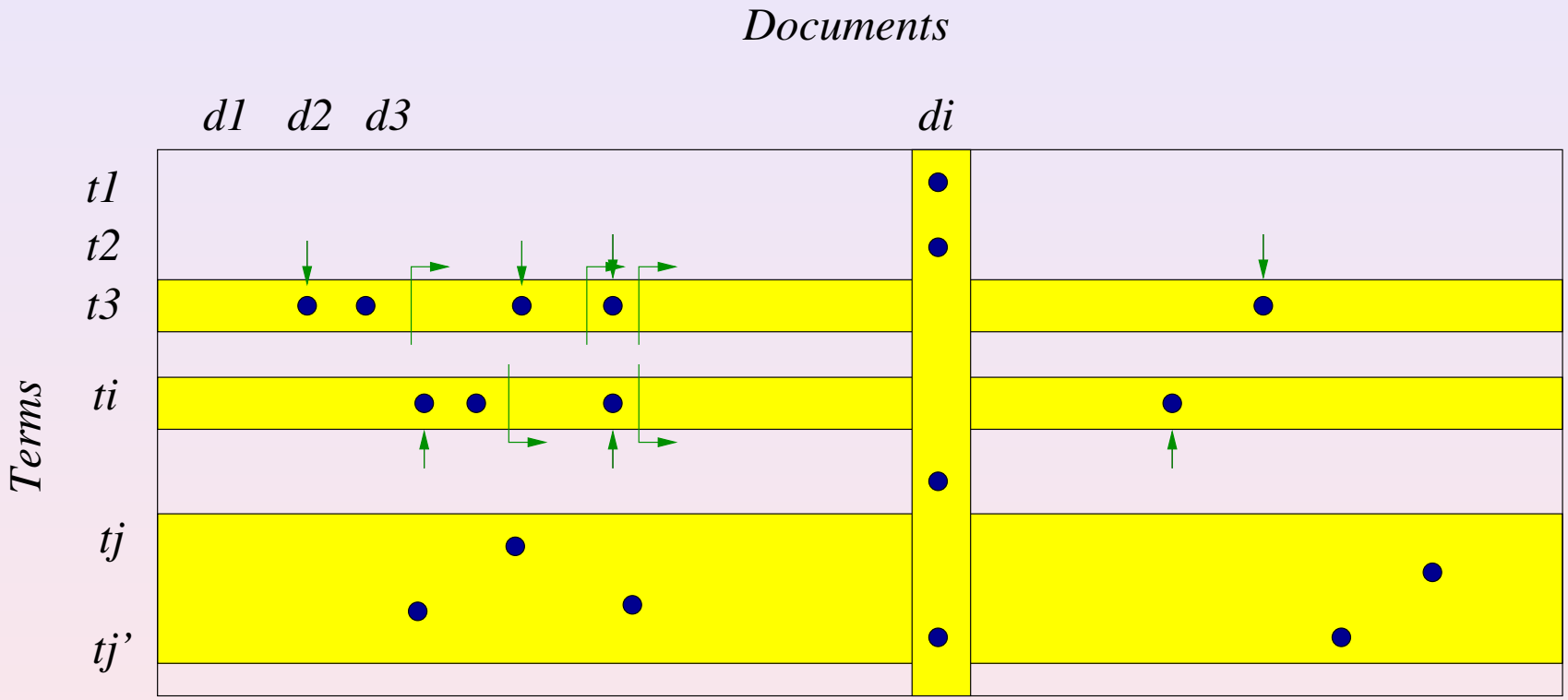
	1	2	3	4	5	6	7	8	9	10
a	0	0	0	0	1	0	1	0	0	0
b	1	0	0	0	1	0	1	0	0	0
c	0	0	0	1	0	0	0	0	0	1
d	1	0	0	0	0	0	0	0	0	0
e	0	0	0	0	1	0	0	0	0	1
f	0	0	0	0	1	1	0	0	0	0



Binary Relations

- On Web graphs, this gives queries over whole domains.
- On inverted indexes (labels are terms and objects are documents):
  - Ranges of labels: stemming, prefixes, thesaurus expansion.
  - Ranges of objects: temporal, versioned, hierarchical collections.
  - Vertical stripes: vocabulary of (ranges of) documents.
  - Style and plagiarism analysis?
  - Horizontal stripes: document frequencies.
  - Simulate inverted list of a range of words.
  - List vocabulary of a range of documents.

Binary Relations

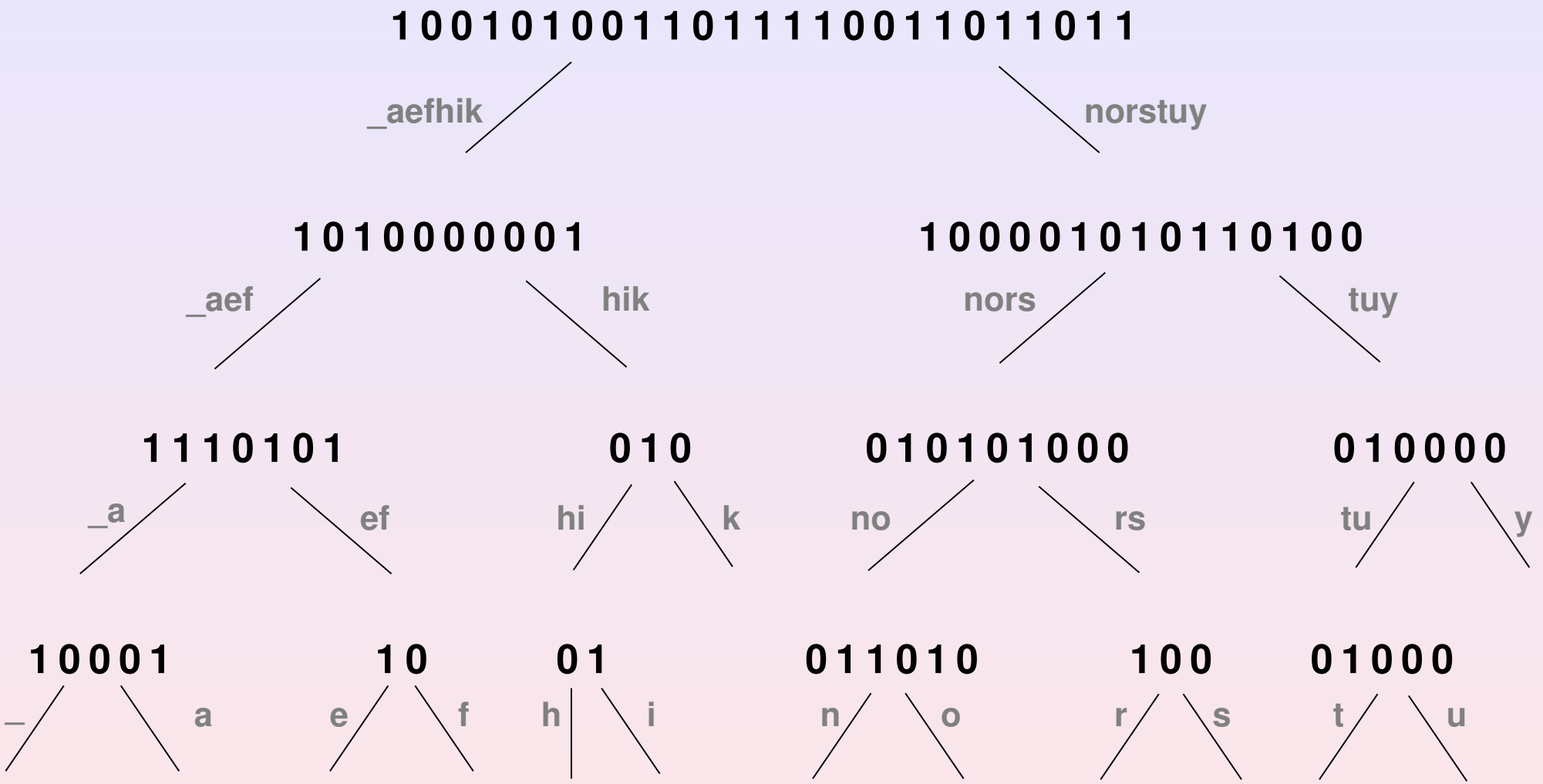


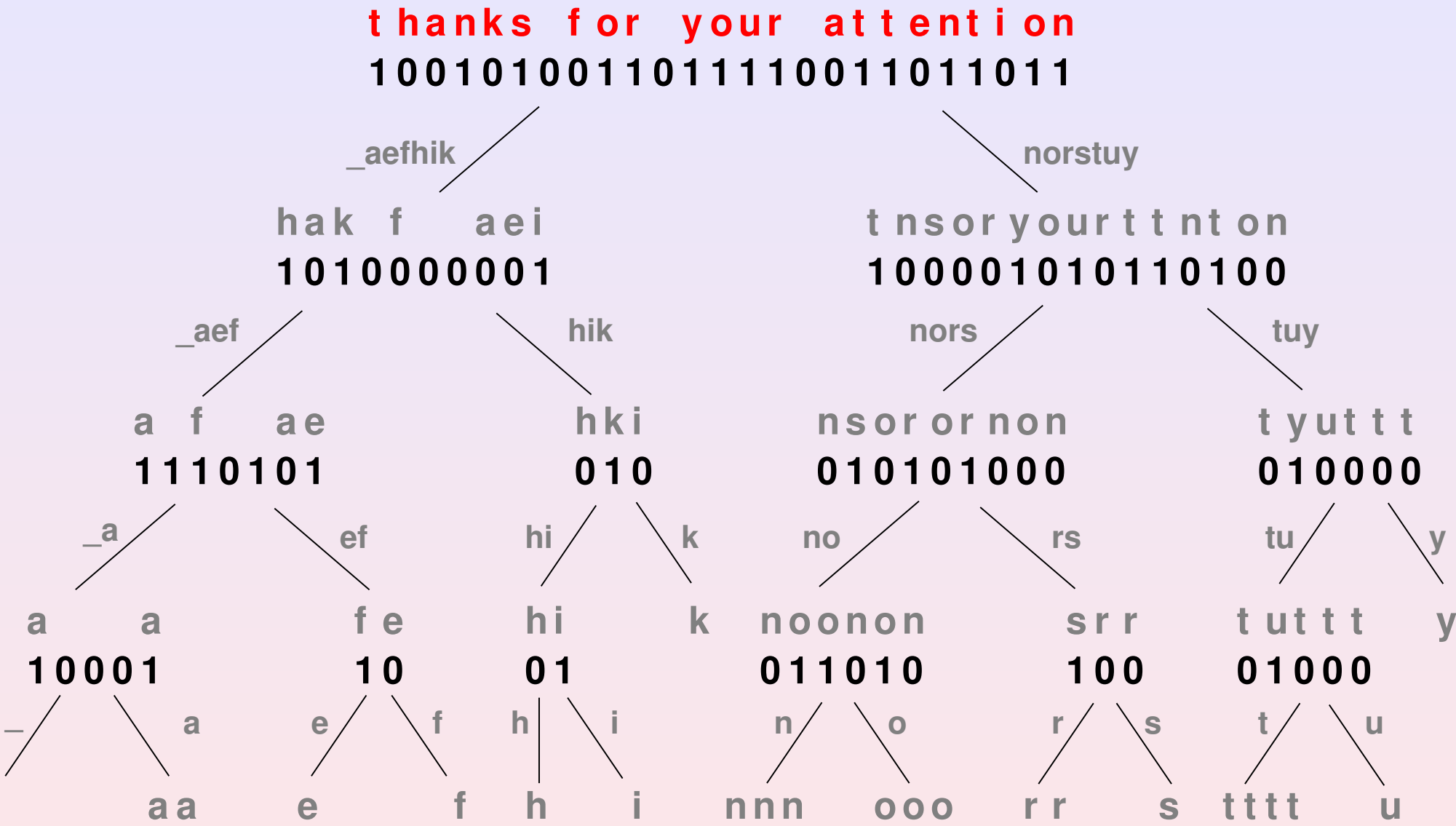
Dynamism

- Can support insertions and deletions of symbols.
- All operations take time $O((\lg n / \lg\lg n)(1 + \lg\sigma / \lg\lg n))$.
- Probably near-optimal since dynamic range counting is $\Omega((\lg n / \lg\lg n)^2)$.
- However [N. and Nekrich, arXiv 2012]
  - If all you want is *access*, *rank*, *select*, and indels...
  - You can get the optimal $O(\lg n / \lg\lg n)$ time.
  - Worst-case for queries, amortized for indels.
  - Zero-order compressed space, as usual.

Conclusions

- Simple and versatile data structure.
- Supports different views of the data within the same space.
- Enables succinct and compressed representations.
- Good (near-log) query times for many problems.
- Lots of applications.
- Wavelet trees for all!


1001010011011111001101101 1
_aefhik
norstuy
1010000001
1000010101101 00
_aef
hik
nors
tuy
1110101
010
010101000
010000
_a
ef
hi
k
no
rs
tu
y
10001
10
01
011010
100
01000
_
a
e
f
h
i
n
o
r
s
t
u


thanks for your attention
1001010011011111001101101 1
_aefhik
norstuy
hak f aei
1010000001
tnsor yourttntton
1000010101101 00
_aef
hik
nors
tuy
a f ae
1110101
hki
010
nsor or non
010101000
t yutttt
010000
_a
ef
hi
k
no
rs
tu
y
a a
10001
f e
10
hi
01
k
noonon
011010
srr
100
t uttt
01000
y
_
a
e
f
h
i
n
o
r
s
t
u
aa
e
f
h
i
nnn
ooo
r r
s
tttt
u