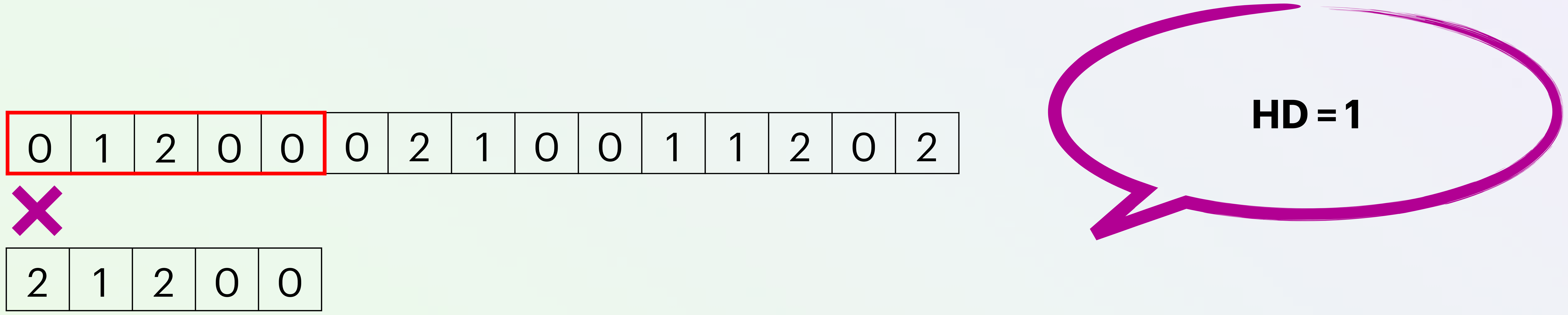
TEXT-TO-PATTERN HAMMING DISTANCE

TATIANA STARIKOVSKAYA, MAÎTRE DE CONFÉRENCES ENS ULM

TEXT-TO-PATTERN HAMMING DISTANCE

We are given two strings (= sequences of letters from a finite alphabet): a text T of length n and a pattern P of length $m \leq n$

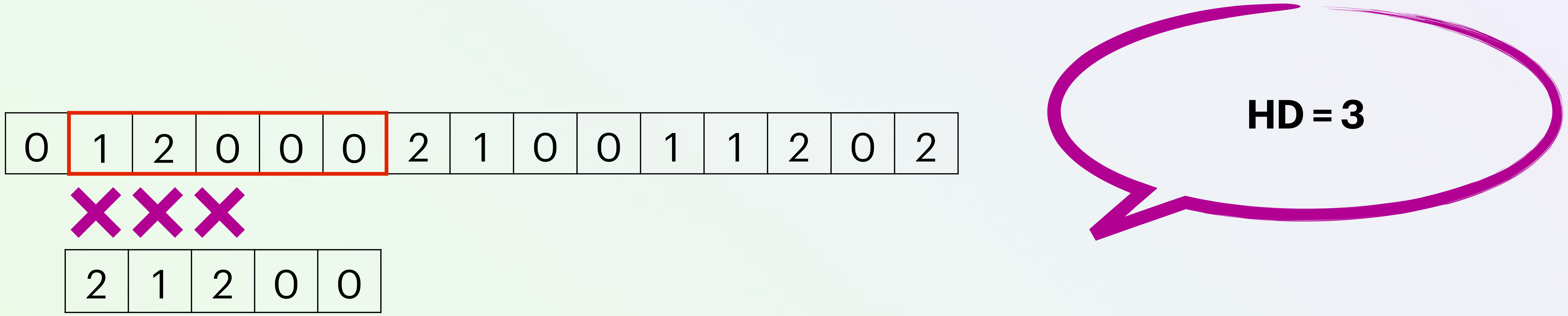
The task is to compute the Hamming distance (= number of mismatches) between each m- length substring of the text and the pattern



TEXT-TO-PATTERN HAMMING DISTANCE

We are given two strings (= sequences of letters from a finite alphabet): a text T of length n and a pattern P of length $m \leq n$

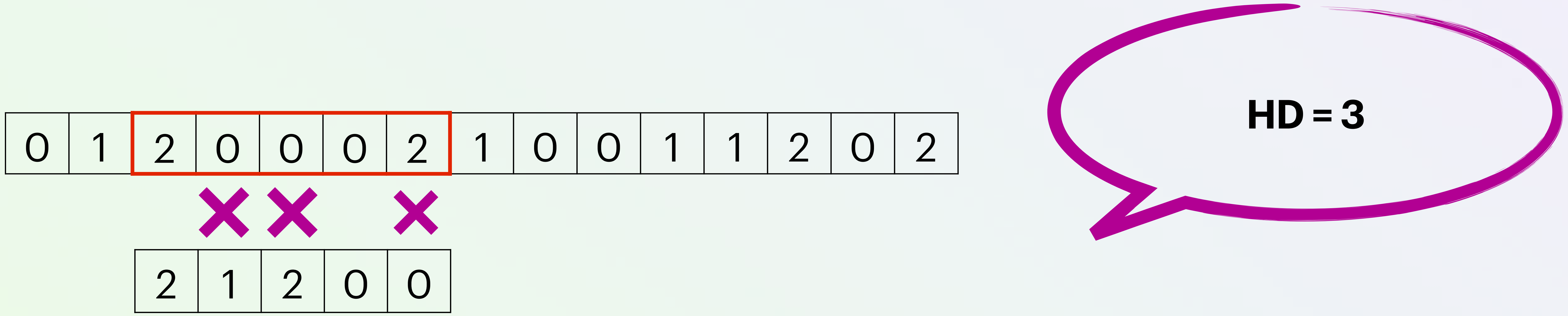
The task is to compute the Hamming distance (= number of mismatches) between each m- length substring of the text and the pattern



TEXT-TO-PATTERN HAMMING DISTANCE

We are given two strings (= sequences of letters from a finite alphabet): a text T of length n and a pattern P of length $m \leq n$

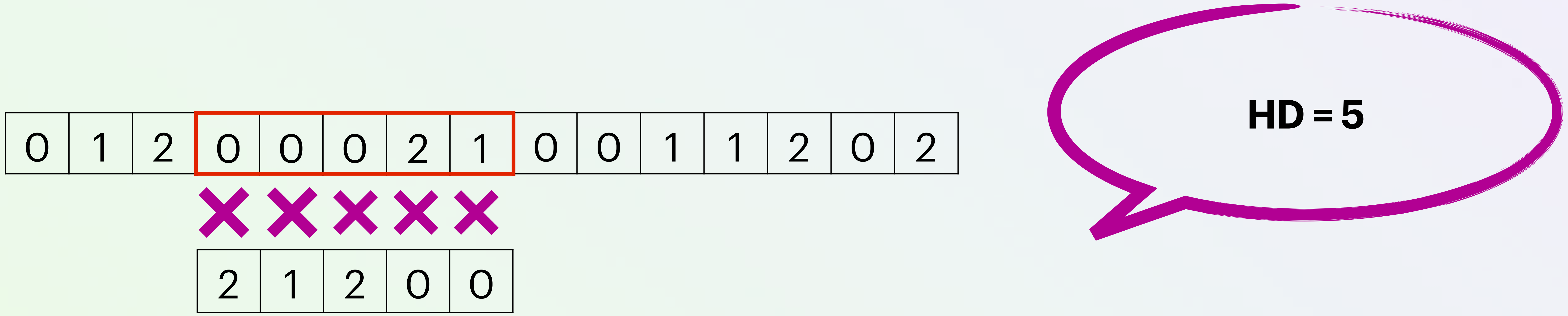
The task is to compute the Hamming distance (= number of mismatches) between each m- length substring of the text and the pattern



TEXT-TO-PATTERN HAMMING DISTANCE

We are given two strings (= sequences of letters from a finite alphabet): a text T of length n and a pattern P of length $m \leq n$

The task is to compute the Hamming distance (= number of mismatches) between each m-length substring of the text and the pattern



... and so on. A fundamental problem in algorithms on strings! Naive algorithm: $O(nm)$ time.

BEFORE WE START

- I will speak in English, **mais je parle français**
- **I love questions!** In English or in French, at any time
- There will be lots of exercises, please **participate actively**

OUTLINE

- **Reminder:** fast multiplication of polynomials via Fast Fourier Transform
- Exact algorithms for binary and general case
- Lower bound for combinatorial algorithms
- Kangaroo jumps
- Smaller space
- Approximation algorithm

REMINDER: FAST MULTIPLICATION OF POLYNOMIALS

Let me start with an algorithm for **fast multiplication of polynomials**

Half of you probably saw it last year, and half will see it this year

We will use it to compute text-to-pattern Hamming distances; it is also a basis of many other great algorithms on strings

FAST MULTIPLICATION OF POLYNOMIALS

Consider $P(x) = \sum_{i=0}^{n-1} a_i x^i$, $Q(x) = \sum_{i=0}^{n-1} b_i x^i$

We can compute $R_1(x) = P(x) + Q(x) = \sum_{i=0}^{n-1} (a_i + b_i)x^i$ in $O(n)$ time

Computing $R_2(x) = P(x) \cdot Q(x) = \sum_{k=0}^{2n-2} \sum_{i+j=k} (a_i \cdot b_j)x^k$ naively requires $O(n^2)$ time

Fast Fourier Transform: $O(n \log n)$ time

FAST MULTIPLICATION OF POLYNOMIALS

Coefficient representation: $P(x) = \sum_{i=0}^{n-1} a_i x^i$

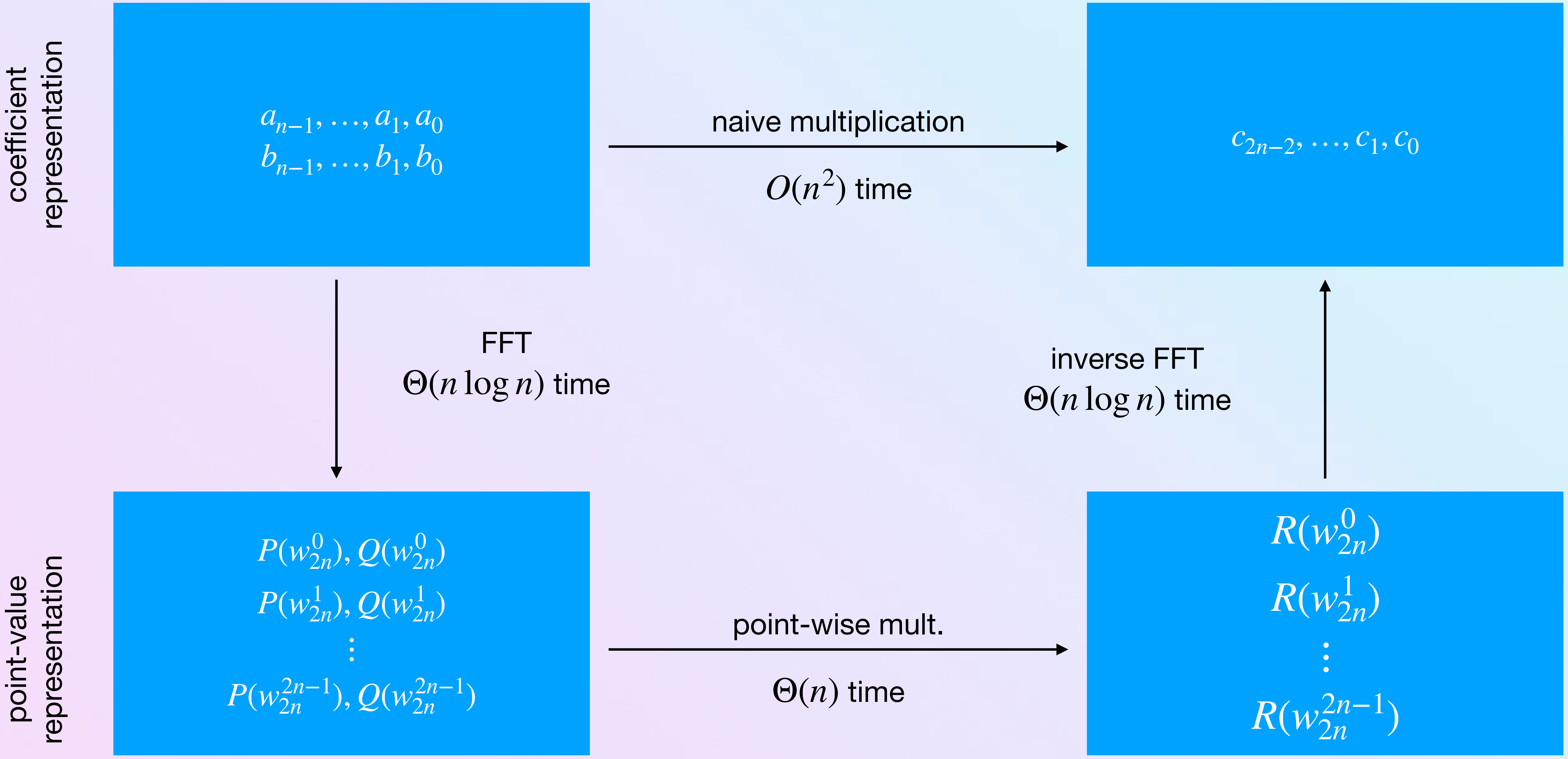
Point-value representation: $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ such that $P(x_i) = y_i$

Proof: in 2 slides

Theorem. For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$, where $x_i \neq x_j$, there exists a unique polynomial of degree $< n$ such that $P(x_i) = y_i$ for all $i = 0, \ldots, n-1$.

Given point-value representations $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ of $P(x)$ and $\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1})\}$ of $P(x)$ and of $Q(x)$, one can compute the point-value representation of $P(x) \cdot Q(x)$ in $O(n)$ time

FAST MULTIPLICATION OF POLYNOMIALS



w_{2n} - $(2n)$-th complex root of unity

FAST MULTIPLICATION OF POLYNOMIALS

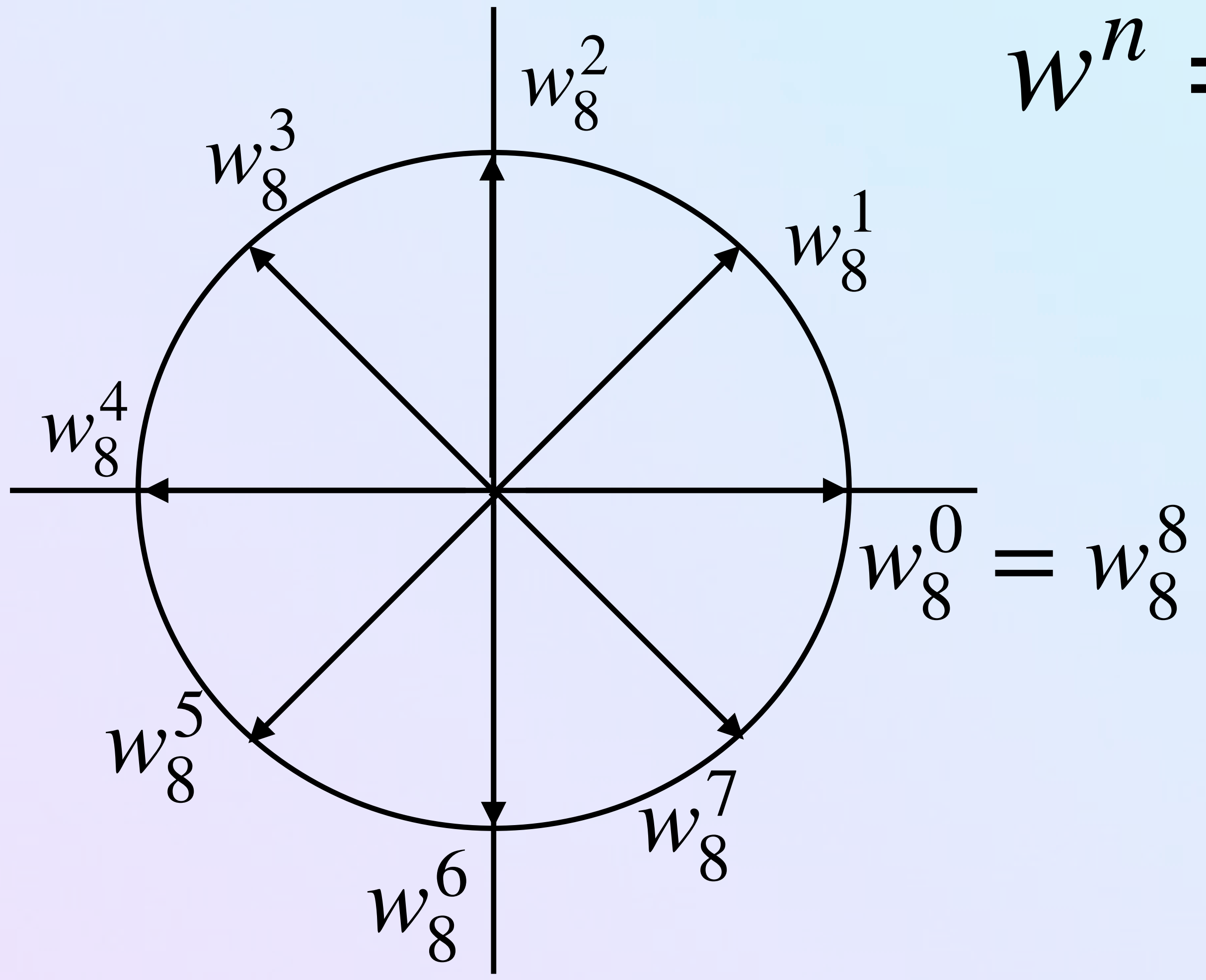
Theorem. For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$, where $x_i \neq x_j$, there exists a unique polynomial of degree $< n$ such that $P(x_i) = y_i$ for all $i = 0, \ldots, n-1$.

Proof. Let $P(x) = \sum_{i=0}^{n-1} a_i x^i$. We can represent the condition $P(x_i) = y_i$ for all $i = 0, \ldots, n-1$ in the matrix form:

Vandermonde matrix, determinant $= \Pi_{0 \leq i < j \leq n-1}(x_j - x_i)$

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

COMPLEX ROOTS OF UNITY



$$w^n = 1$$

$$w_n^k = e^{2\pi ik/n} = \cos 2\pi k/n + i \sin 2\pi k/n$$

Halving property: $\{(w_{2n}^0)^2, (w_{2n}^1)^2, \ldots, (w_{2n}^{2n-1})^2\} = \{w_n^0, w_n^1, \ldots, w_n^{n-1}\}$

Cancellation property: $w_{dn}^{dk} = w_n^k$

Summation property: $\sum_{j=0}^{n-1} (w_n^k)^j = 0$ for all $k \neq 0 \pmod{n}$

FAST FOURIER TRANSFORM

$P(x) = \sum_{i=0}^{i=n-1} a_i x^i \rightarrow$ discrete Fourier transform $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\}$ (assume $n = 2^j$)

$$P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$$

↙ ↘

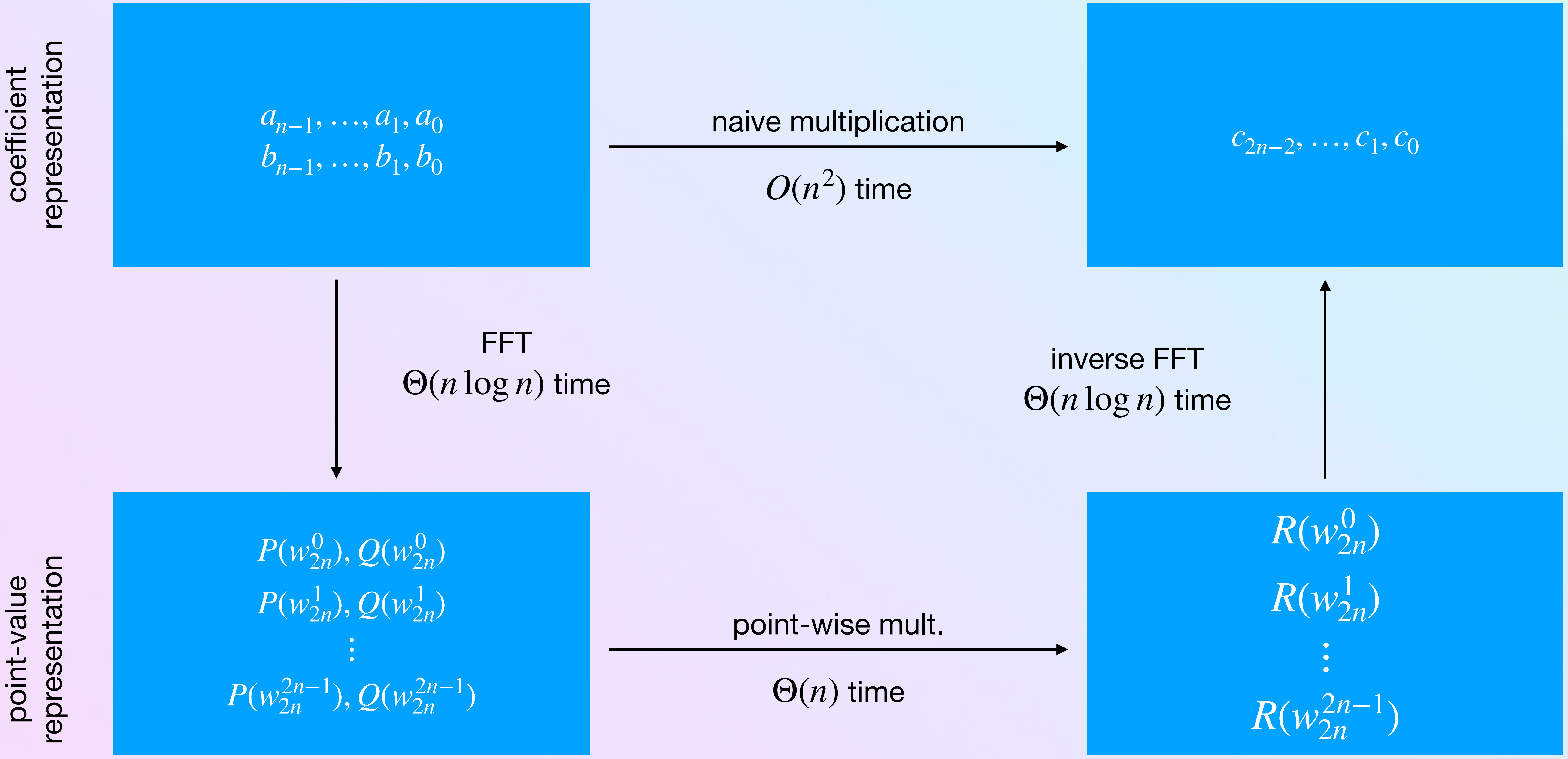
$$P_{odd}(x) = a_{n-1}x^{n/2-1} + a_{n-3}x^{n/2-2} + \ldots + a_1 \qquad P_{even}(x) = a_{n-2}x^{n/2-1} + a_{n-4}x^{n/2-1} + \ldots + a_0$$

$$P(x) = xP_{odd}(x^2) + P_{even}(x^2)$$

- Evaluate $P_{odd}(x)$ and $P_{even}(x)$ at $(w_n^0)^2, (w_n^1)^2, \ldots, (w_n^{n-1})^2$ recursively (by the halving property, $\{(w_n^0)^2, (w_n^1)^2, \ldots, (w_n^{n-1})^2\} = \{(w_{n/2}^0), (w_{n/2}^1), \ldots, (w_{n/2}^{n/2-1})\}$)
- Combine the results to compute $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\}$

$T(n) = 2T(n/2) + \Theta(n) = O(n \log n)$

FAST MULTIPLICATION OF POLYNOMIALS



w_{2n} - $(2n)$-th complex root of unity

INVERSE FOURIER TRANSFORM

Point representation $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\} \rightarrow P(x) = \sum_{i=0}^{i=n-1} a_i x^i$

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w_n & w_n^2 & \ldots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \ldots & w_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_n^{n-1} & w_n^{(n-1)2} & \ldots & w_n^{(n-1)(n-1)} \end{pmatrix}}_{V_n} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(w_n^0) \\ P(w_n^1) \\ \vdots \\ P(w_n^{n-1}) \end{pmatrix}$$

INVERSE FOURIER TRANSFORM

Point representation $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\} \rightarrow P(x) = \sum_{i=0}^{i=n-1} a_i x^i$

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = V_n^{-1} \times \begin{pmatrix} P(w_n^0) \\ P(w_n^1) \\ \vdots \\ P(w_n^{n-1}) \end{pmatrix}$$

INVERSE FOURIER TRANSFORM

Theorem. $V_n^{-1}[j,k] = w_n^{-kj}/n$

Proof.

$$(V_n^{-1}V_n)[j,j'] = \sum_{k=0}^{n-1}(V_n^{-1})[j,k](V_n)[k,j'] = \sum_{k=0}^{n-1}(w_n^{-kj}/n)(w_n^{kj}) = \sum_{k=0}^{n-1}(w_n^{k(j'-j)}/n)$$

If $j' = j$, the sum equals one. Otherwise, the sum equals zero by **Summation property**

Corollary. $a_j = \frac{1}{n}\sum_{k=0}^{n-1} P(w_n^k)w_n^{-kj}$, that is, $\{a_j\}$ is a point-value representation of a polynomial

$Q(z) = \sum_{k=0}^{n-1} y_k z^k$ and **can be computed in $O(n \log n)$ time using Fast Fourier transform!**

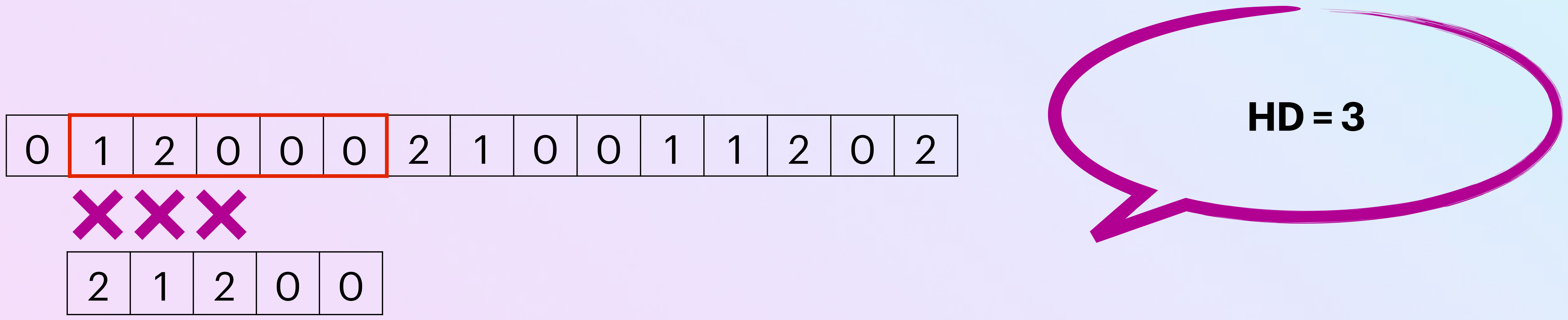
TWO POLYNOMIALS OF DEGREE AT MOST n CAN BE MULTIPLIED IN $O(n \log n)$ TIME

ALL TEXT-TO-PATTERN HAMMING DISTANCES

PROBLEM FORMULATION

We are given two strings (= sequences of letters from a finite alphabet: a text T of length n and a pattern P of length $m \leq n$

The task is to compute the Hamming distance (= number of mismatches) between each m-length substring of the text and of the pattern



CONSTANT-SIZE ALPHABETS

FISHER AND PATERSON'74

Our task is to develop an algorithm with running time $O(n \log m)$.

Given two integer vectors A, B of lengths n and m, $n \geq m$, their convolution is defined as a vector C of length $n - m$, where $C[i] = \sum_{j=1}^{m} A[i+m-j]B[j]$. Show an $O(n \log m)$-time algorithm for computing the **convolution**.

Given binary text T of length n and pattern P of length m. For a substring $T[i-m+1,i]$ of the text, express the **number of matching ones** between it and P in terms of a convolution. What about the number of matching zeros and the Hamming distance?

Derive a $O(n \log n)$-time algorithm for computing the Hamming distances between all m-length substrings of **binary** T and P and a $O(\sigma n \log n)$-time algorithm for strings over an alphabet of size σ.

GENERAL ALPHABETS

ABRAHAMSON'87

Let's now develop an algorithm with running time $O(n\sqrt{m \log m})$!

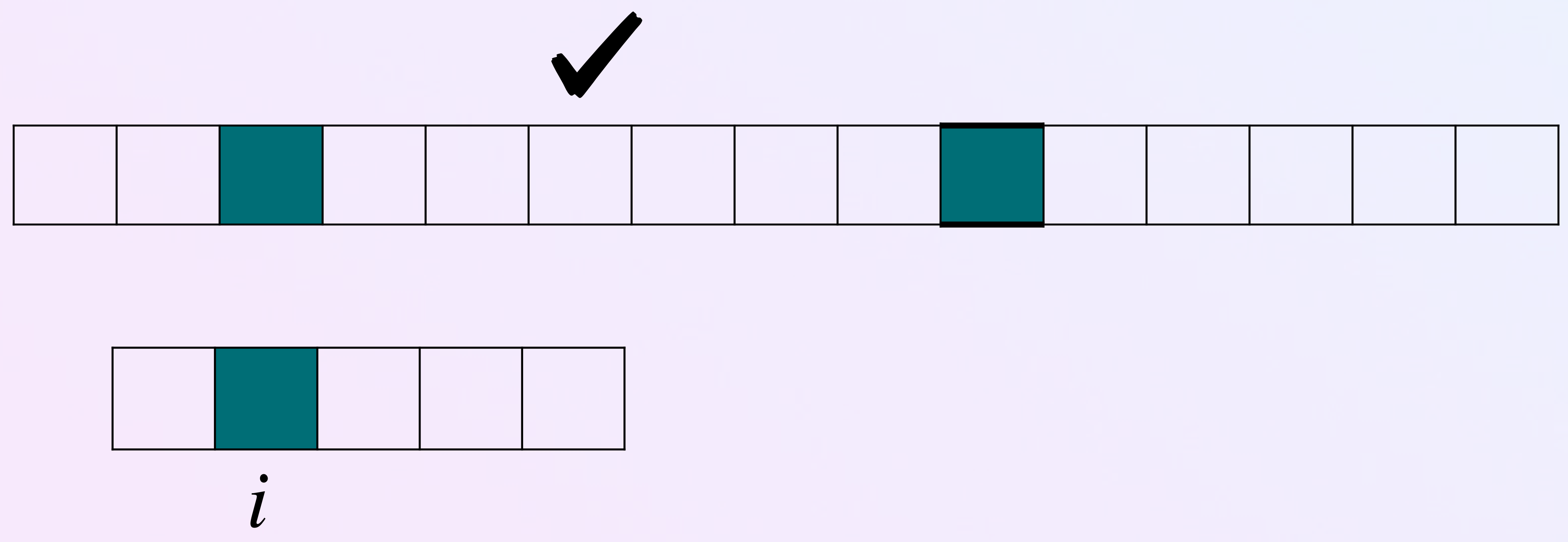
- Let's assume that $n = 2m$ for starters. A letter of T is called **frequent** if it occurs at least $\sqrt{m \log m}$ times. Number of frequent letters is $\leq 2\sqrt{m/\log m}$.
- How to compute the **number of mismatches due to frequent letters** in $O(m\sqrt{m \log m})$ time?

GENERAL ALPHABETS

ABRAHAMSON'87

Our task is to develop an algorithm with running time $O(n\sqrt{m \log m})$.

- For each position i of P such that $P[i]$ is **not frequent** mark at most $\sqrt{m \log m}$ positions in the text where $P[i]$ and its occurrence in the text are aligned.

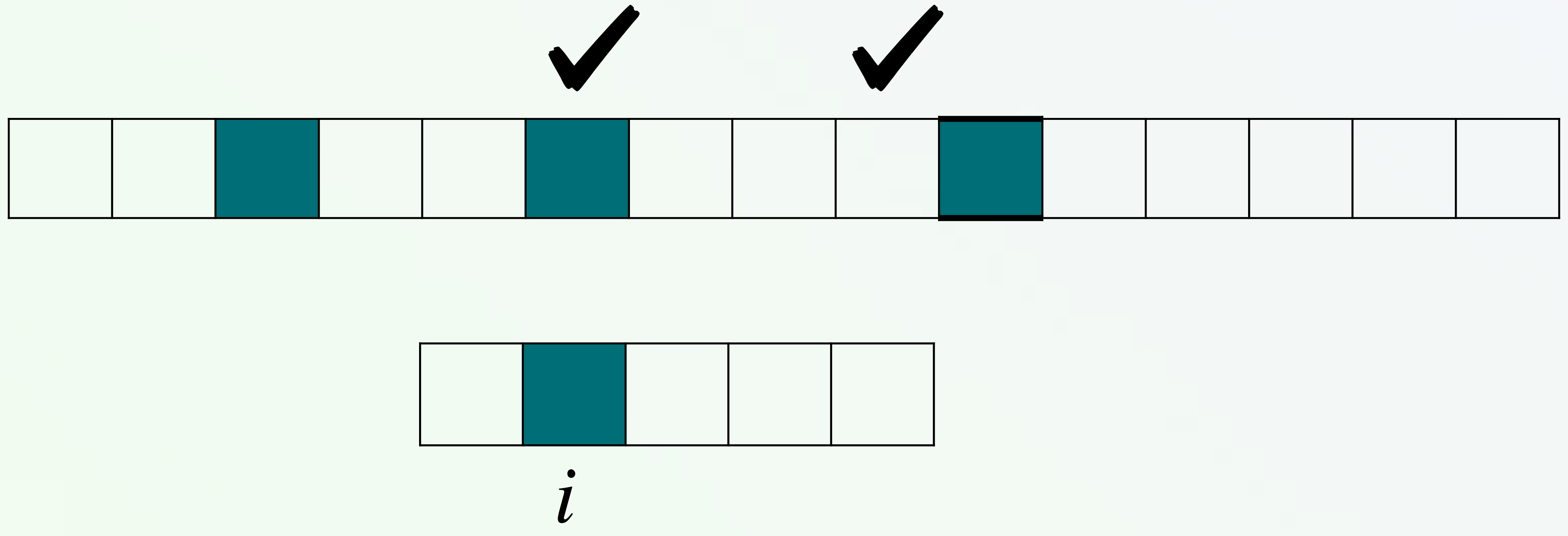


GENERAL ALPHABETS

ABRAHAMSON'87

Our task is to develop an algorithm with running time $O(n\sqrt{m \log m})$.

- For each position i of P such that $P[i]$ is **not frequent** mark at most $\sqrt{m \log m}$ positions in the text where $P[i]$ and its occurrence in the text are aligned.

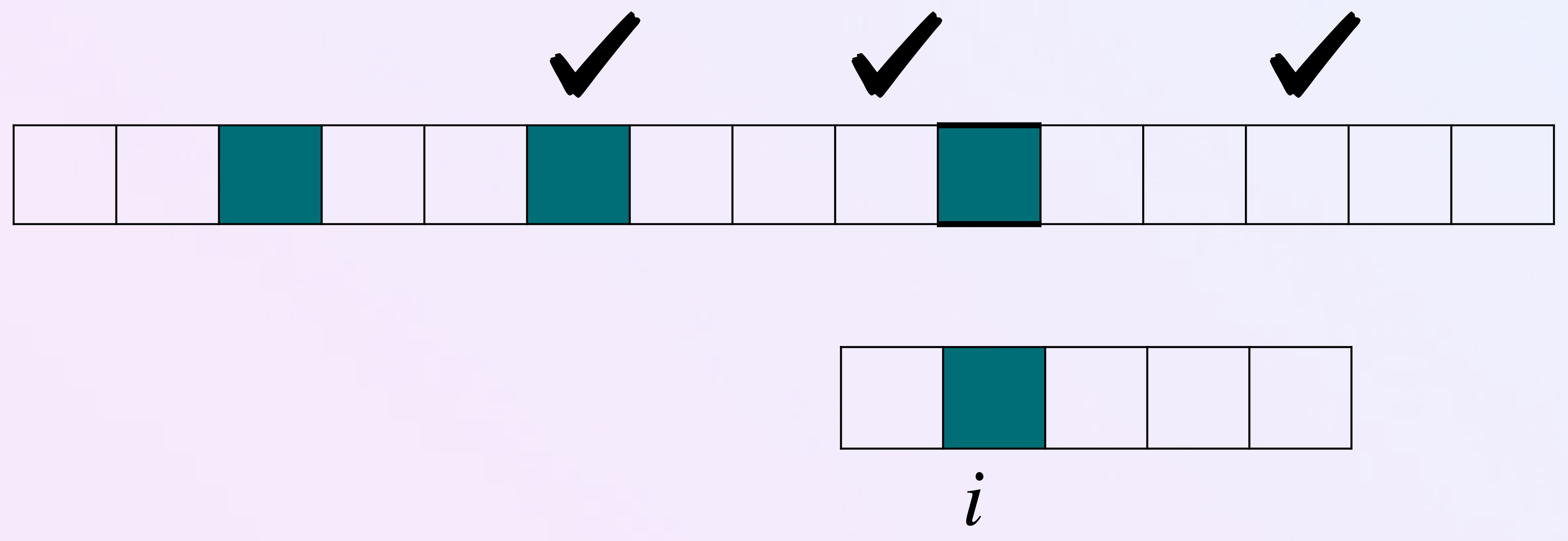


GENERAL ALPHABETS

ABRAHAMSON'87

Our task is to develop an algorithm with running time $O(n\sqrt{m \log m})$.

- For each position i of P such that $P[i]$ is **not frequent** mark at most $\sqrt{m \log m}$ positions in the text where $P[i]$ and its occurrence in the text are aligned.

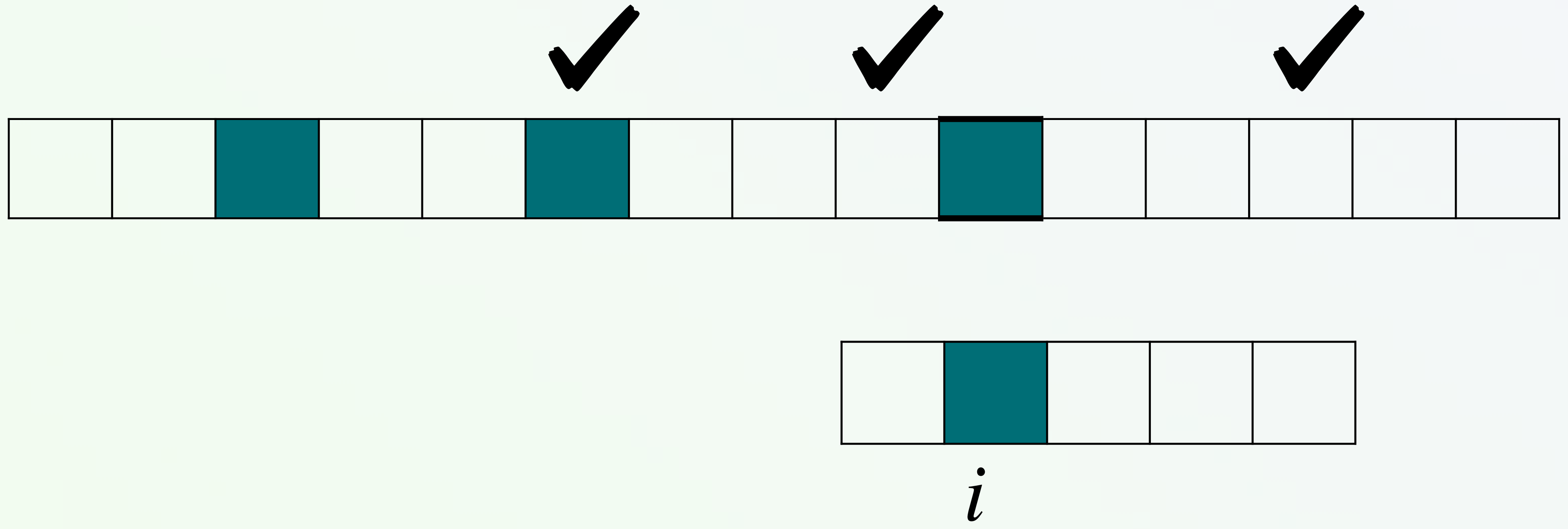


GENERAL ALPHABETS

ABRAHAMSON'87

Our task is to develop an algorithm with running time $O(n\sqrt{m \log m})$.

- For each position i of P such that $P[i]$ is **not frequent** mark at most $\sqrt{m \log m}$ positions in the text where $P[i]$ and its occurrence in the text are aligned. **Total time: $O(m\sqrt{m \log m})$.**



- We can use the marks to compute the number of **mismatches due to non-frequent letters**

GENERAL ALPHABETS

ABRAHAMSON'87

- We can sum up the mismatches due to frequent and non-frequent letters in $O(m)$ time.
- This gives a $O(m\sqrt{m \log m})$-time algorithm for the case $n = 2m$.
- Derive an $O(n\sqrt{m \log m})$-time algorithm for general n.

ALL HAMMING DISTANCES:

$O(n \log m)$ TIME FOR CONSTANT-SIZE ALPHABET AND $O(n\sqrt{m \log m})$ IN GENERAL

BIG OPEN QUESTION: IS THERE A FASTER ALGORITHM?

LOWER BOUND

COMBINATORIAL MATRIX MULTIPLICATION

Conjecture. For any $\alpha, \beta, \gamma, \varepsilon > 0$, there is no combinatorial algorithm for multiplying an $n^\alpha \times n^\beta$ matrix A with an $n^\beta \times n^\gamma$ matrix B in time $O(n^{\alpha+\beta+\gamma-\varepsilon})$.

NB! It is not clear what does combinatorial mean *precisely*. However, FFT and so boolean convolution often used in algorithms on strings are considered not to be combinatorial.

ENCODING MATRICES

$M \geq N \geq L$

A ($M \times N$):

0	1	1	0
1	0	1	0
0	0	0	1
1	1	1	0
0	1	1	1

×

B ($N \times L$):

1	0	1
1	1	0
1	0	1
1	0	0

= ?

Replace every 1 in column j of A with j and every 1 in row i of B with i

ENCODING MATRICES

$M \geq N \geq L$

N columns × M rows:

0	2	3	0
1	0	3	0
0	0	0	4
1	2	3	0
0	2	3	4

×

L columns × N rows:

1	0′	1
2	2	0′
3	0′	3
4	0′	0′

= ?

Replace every 0 in B with 0′

ENCODING MATRICES

$M \geq N \geq L$

0	2	3	0
1	0	3	0
0	0	0	4
1	2	3	0
0	2	3	4

M (rows) × N (columns)

×

1	0′	1
2	2	0′
3	0′	3
4	0′	0

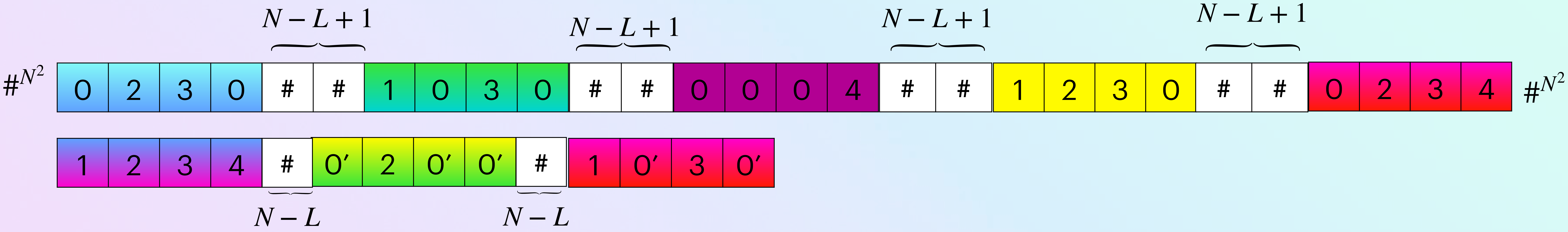
N (rows) × L (columns)

= ?

$T = \#^{N^2}$ 0 2 3 0 $\#^{N-L+1}$ 1 0 3 0 $\#^{N-L+1}$ 0 0 0 4 $\#^{N-L+1}$ 1 2 3 0 $\#^{N-L+1}$ 0 2 3 4 $\#^{N^2}$

$P =$ 1 2 3 4 $\#^{N-L}$ 0′ 2 0′ 0′ $\#^{N-L}$ 1 0′ 3 0′

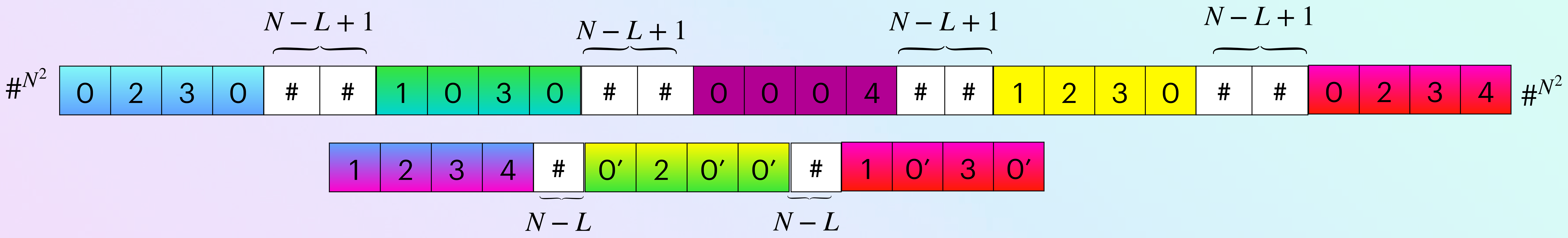
ENCODING MATRICES



For the sake of example, let's recall that $N = 4$ and $L = 3$.

What can we say about the Hamming distance at a particular alignment of T and P?

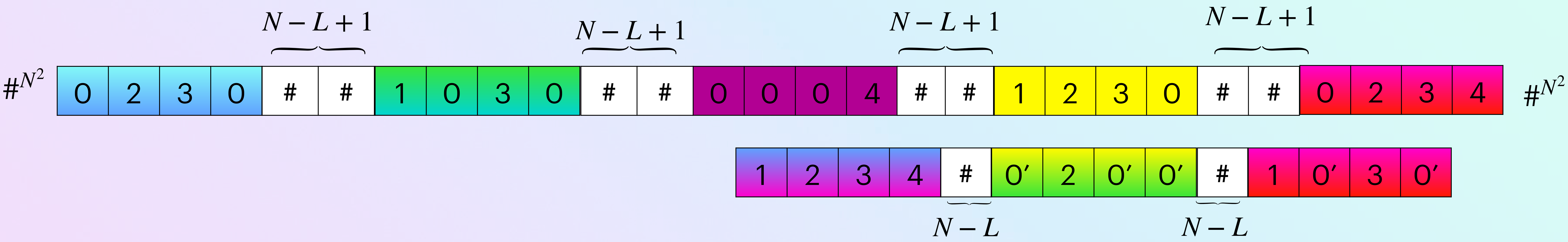
ENCODING MATRICES



For the sake of example, let's recall that $N = 4$ and $L = 3$.

What can we say about the Hamming distance at a particular alignment of T and P?

ENCODING MATRICES



For the sake of example, let's recall that $N = 4$ and $L = 3$.

What can we say about the Hamming distance at a particular alignment of T and P?

ENCODING MATRICES

A row i of the 1st matrix and a column j of the 2nd matrix generate a match iff:

- They are perfectly aligned
- There is k such that the kth bit of A and the kth bit of B are 1

For any alignment of the pattern and of the text there is **at most one aligned row-column pair** (length of a row+padding in the text is $N+1$, a column+padding in the pattern — N)

$$T = \#^{N^2}\ 0\ 2\ 3\ 0\ \#^{N-L+1}\ 1\ 0\ 3\ 0\ \#^{N-L+1}\ 0\ 0\ 0\ 4\ \#^{N-L+1}\ 1\ 2\ 3\ 0\ \#^{N-L+1}\ 0\ 2\ 3\ 4\ \#^{N^2}$$

$$P = 1\ 2\ 3\ 4\ \#^{N-L}\ 0'\ 2\ 0'\ 0'\ \#^{N-L}\ 1\ 0'\ 3\ 0'$$

ENCODING MATRICES

Length of P: $(N-1)(N-L)+NL=\Theta(N^2)=|P|$

Can be computed in $O(1)$ time!

Hamming distance =

|non-# letters in P| + |non-# letters in a |P|-length substring of T| — |matches| $\leq MN$

By computing all Hamming distances, we can derive $A \times B$!

$T = \#^{N^2}$ 0 2 3 0 $\#^{N-L+1}$ 1 0 3 0 $\#^{N-L+1}$ 0 0 0 4 $\#^{N-L+1}$ 1 2 3 0 $\#^{N-L+1}$ 0 2 3 4 $\#^{N^2}$

$P =$ 1 2 3 4 $\#^{N-L}$ 0′ 2 0′ 0′ $\#^{N-L}$ 1 0′ 3 0′

ENCODING MATRICES

BASED ON GAWRYCHOWSKI AND UZNANSKI'18

By computing all Hamming distances, we can derive $A \times B$!

$$|P| = (N-1)(N-L) + NL = \Theta(N^2)$$

$$|T| = 2N^2 + MN + (M-1)(N-L+1) = \Theta(MN)$$

Set $M = n^{1-\alpha/2}$, $N = L = n^{\alpha/2}$. By the CMM conjecture, **no combinatorial algorithm can solve the problem in $n^{1+\alpha/2-\alpha\varepsilon} = |T| \cdot |P|^{1/2-\varepsilon}$ time!**

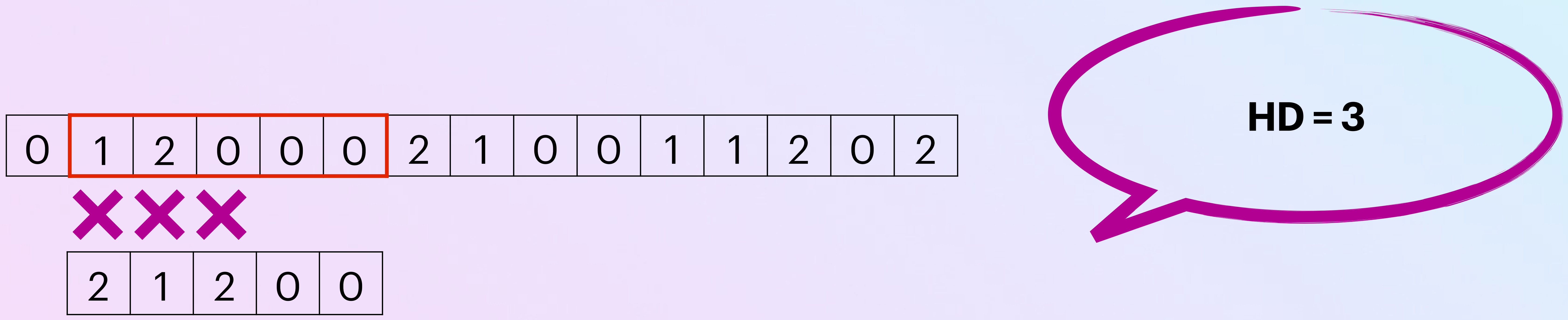
$T = \#^{N^2}$ 0 2 3 0 $\#^{N-L+1}$ 1 0 3 0 $\#^{N-L+1}$ 0 0 0 4 $\#^{N-L+1}$ 1 2 3 0 $\#^{N-L+1}$ 0 2 3 4 $\#^{N^2}$

$P =$ 1 2 3 4 $\#^{N-L}$ 0′ 2 0′ 0′ $\#^{N-L}$ 1 0′ 3 0′

KANGAROO JUMPS

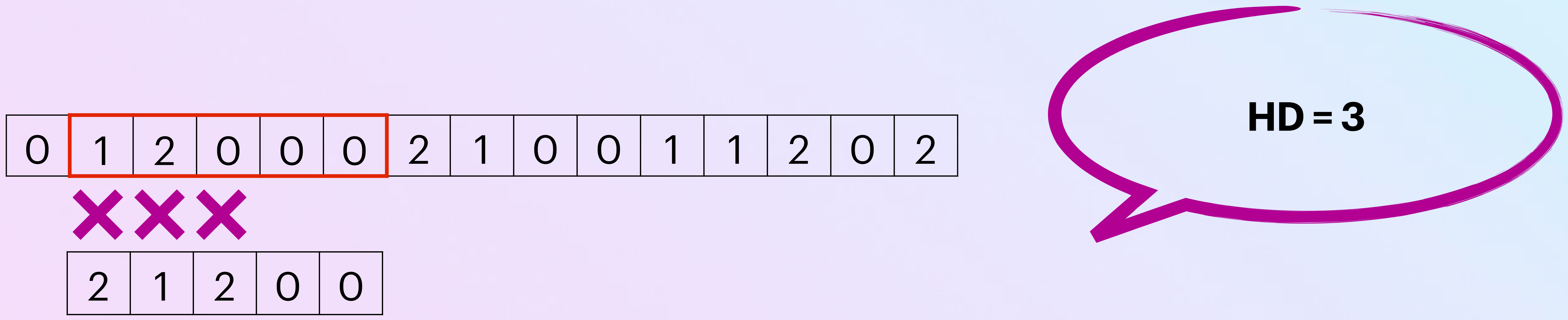
PROBLEM FORMULATION

Given a text T of length n and a pattern P of length m, compute the Hamming distance between each m-length substring of T and P



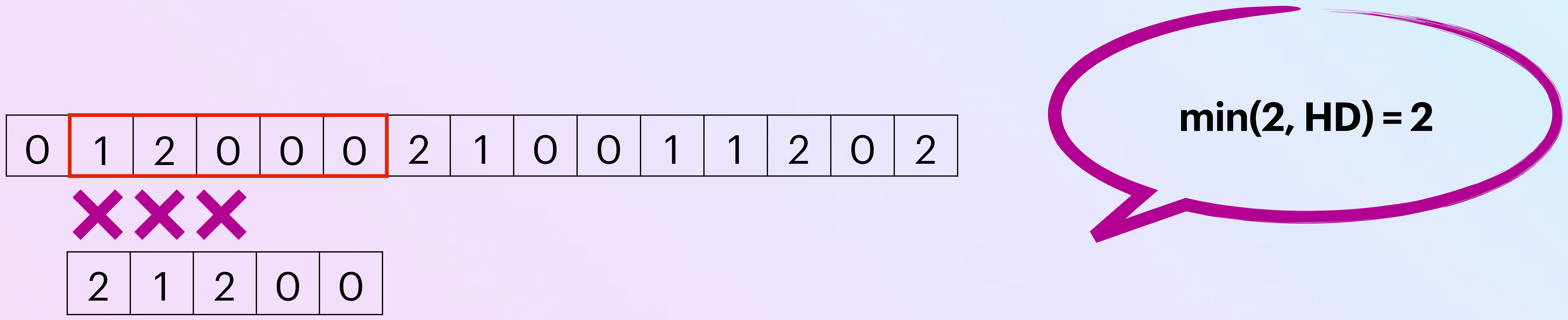
PROBLEM FORMULATION

Given a text T of length n and a pattern P of length m, compute ~~the Hamming distance~~ between each m-length substring of T and P



PROBLEM FORMULATION

Given a text T of length n, a pattern P of length m, and **an integer k**, compute **the minimum of k+1 and the Hamming distance** between each m-length substring of T and P



PROBLEM FORMULATION

In other words, if the Hamming distance d between a substring and the pattern

- $\leq k$, then we must output d
- otherwise, we can simply output $k+1$, which means that "the distance is too large"

Makes perfect sense in practice: why would you be interested in substrings that are too far from your pattern?

This variant of the problem is called **the k-mismatch problem**

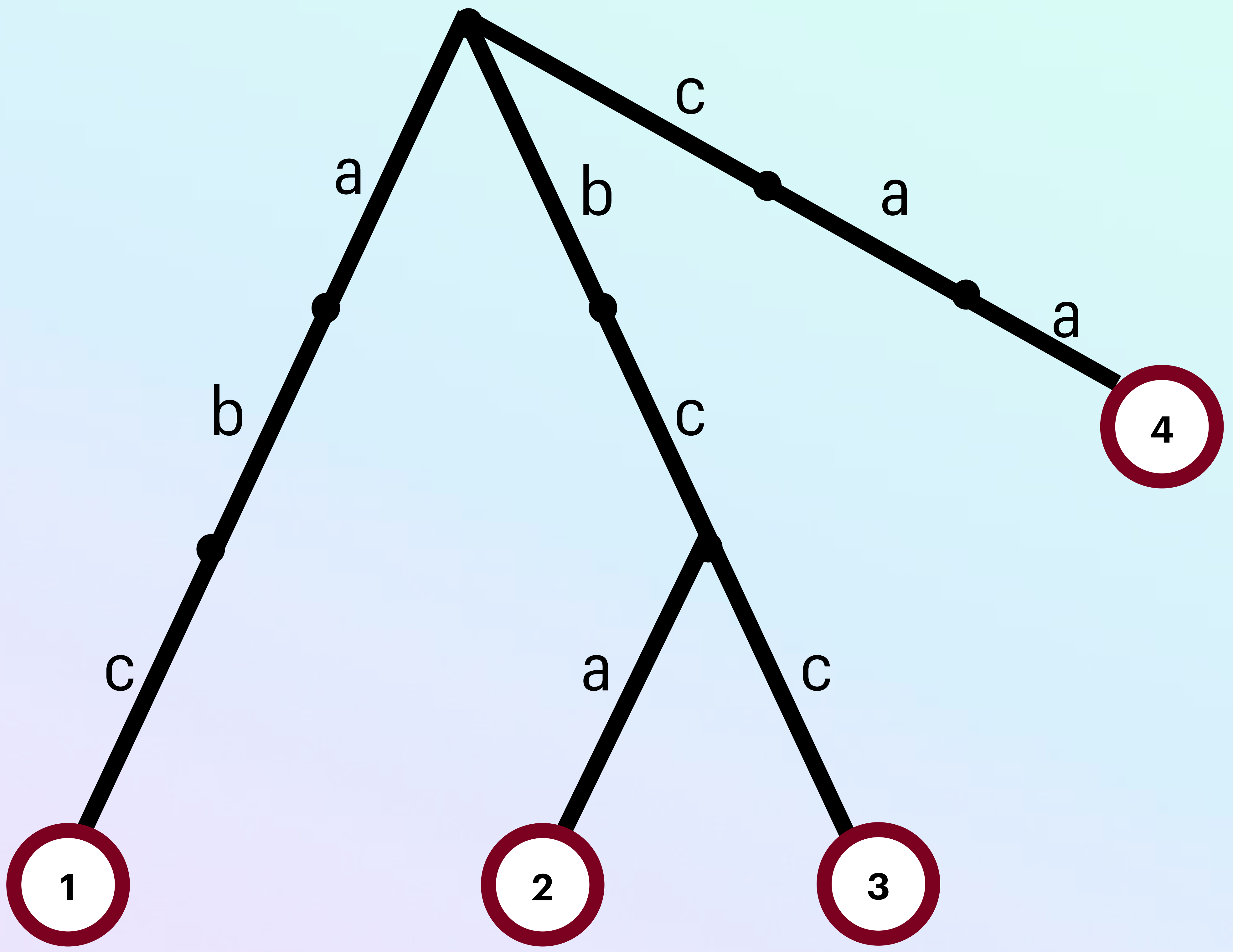
Suffix tree

TRIE

Dictionary D **=** set of strings

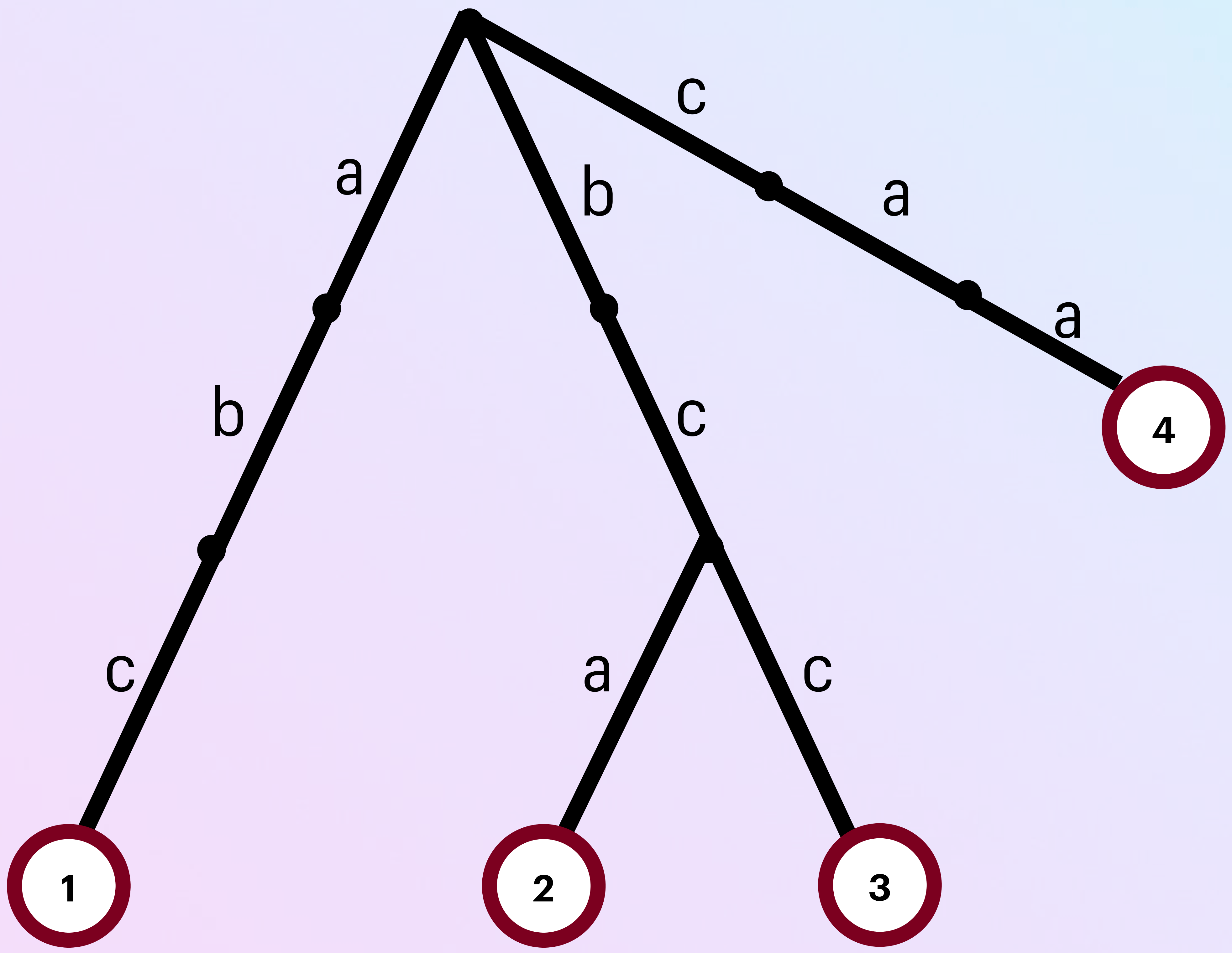
Trie for D is a tree. Every edge of the trie is labeled with a letter so that:

- For every node, outgoing edges are labeled with different letters.
- For every string $S \in D$, there is a root-to-node path that spells out S. The end of the path is labeled with the id of S.
- Every root-to-leaf path spells out a string from D.
- Space = O(total length of strings in D)

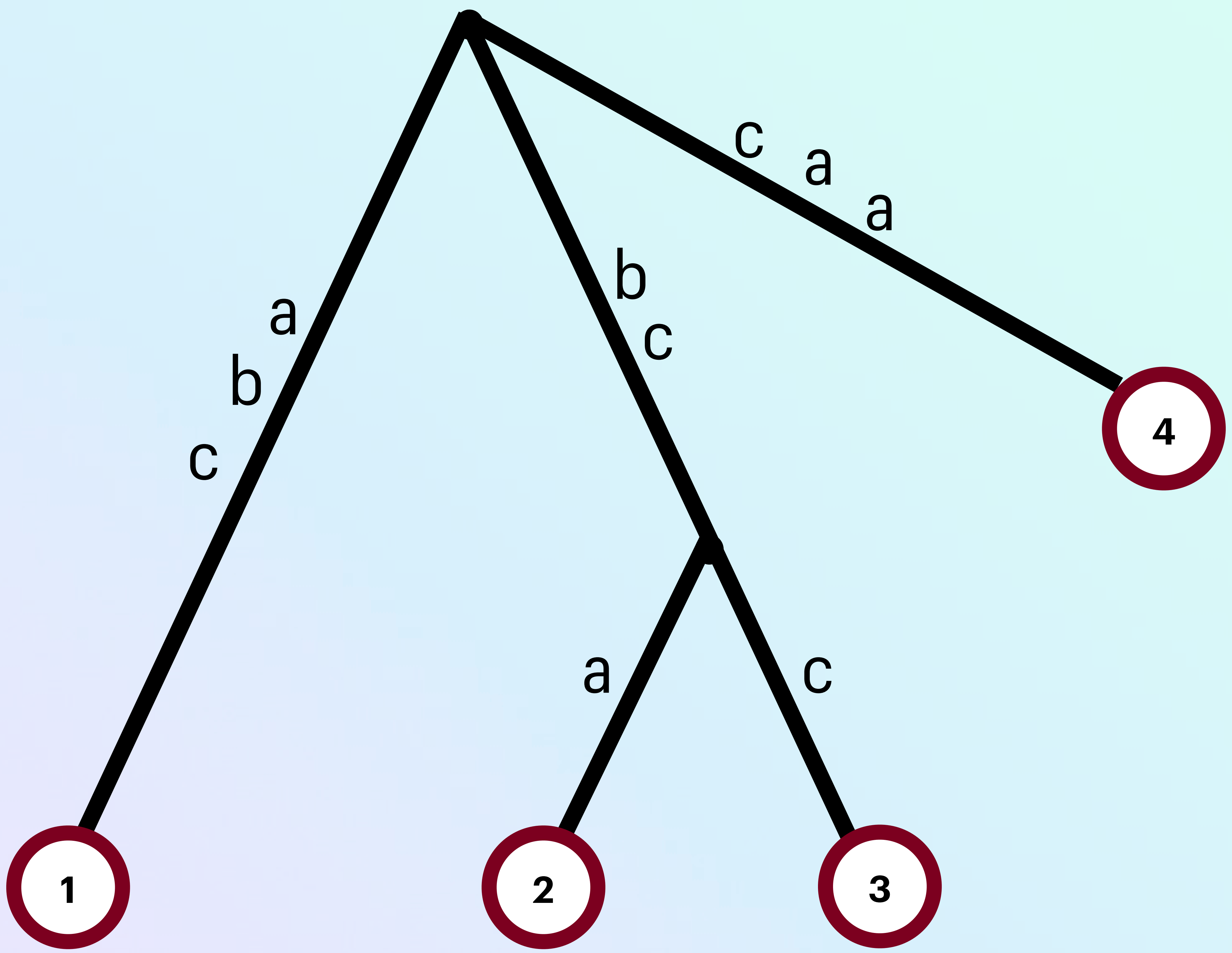


Example: $D = \{\underset{1}{abc}, \underset{2}{bca}, \underset{3}{bcc}, \underset{4}{caa}\}$

COMPACT TRIE



trie



compact trie

SUFFIX TREE

Suffixes of a string T = banana:

T[1,6] = banana

T[2,6] = anana

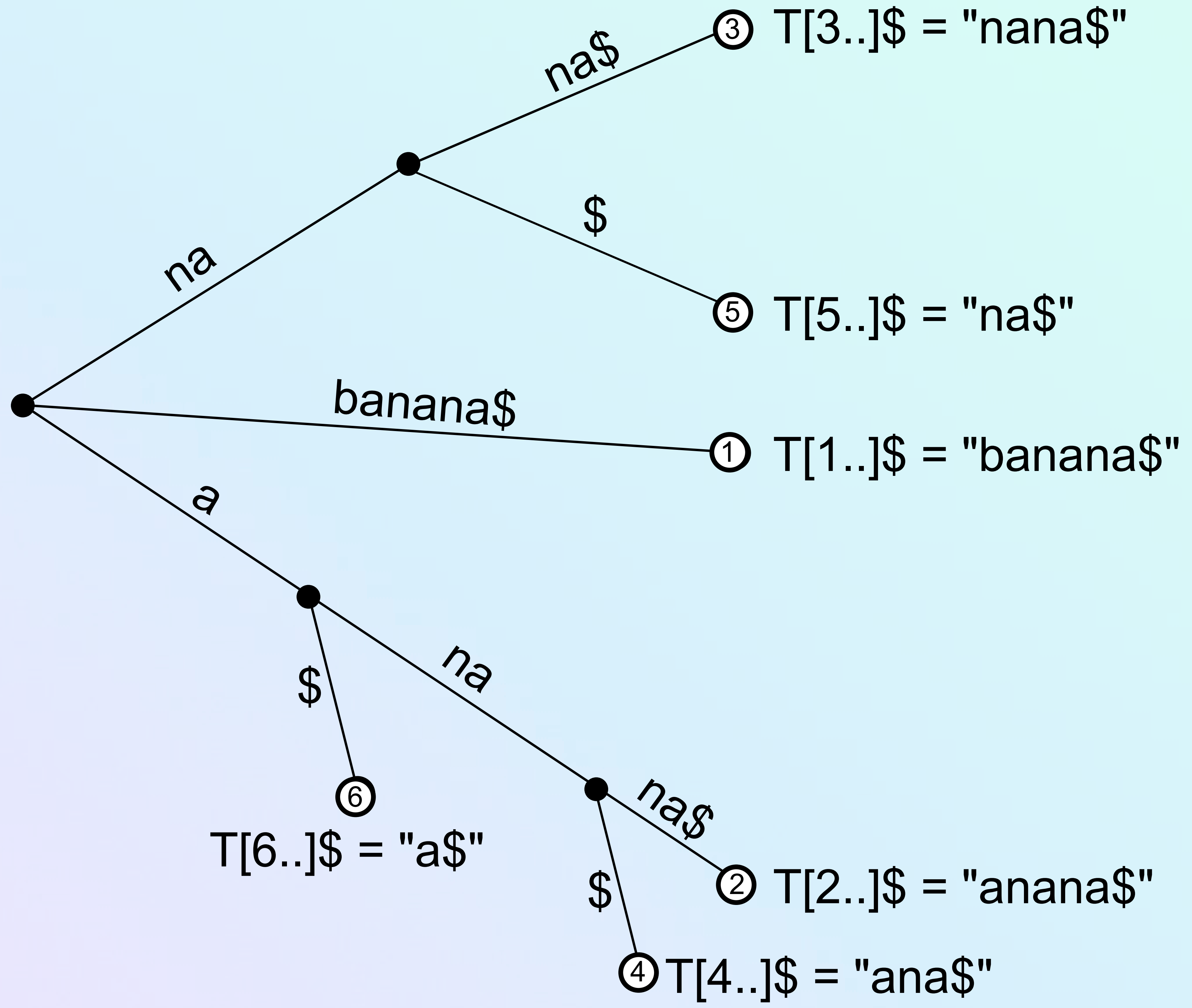
T[3,6] = nana

T[4,6] = ana

T[5,6] = na

T[6,6] = a

We append $ to each of the suffixes and build the compact trie for them.



SUFFIX TREE

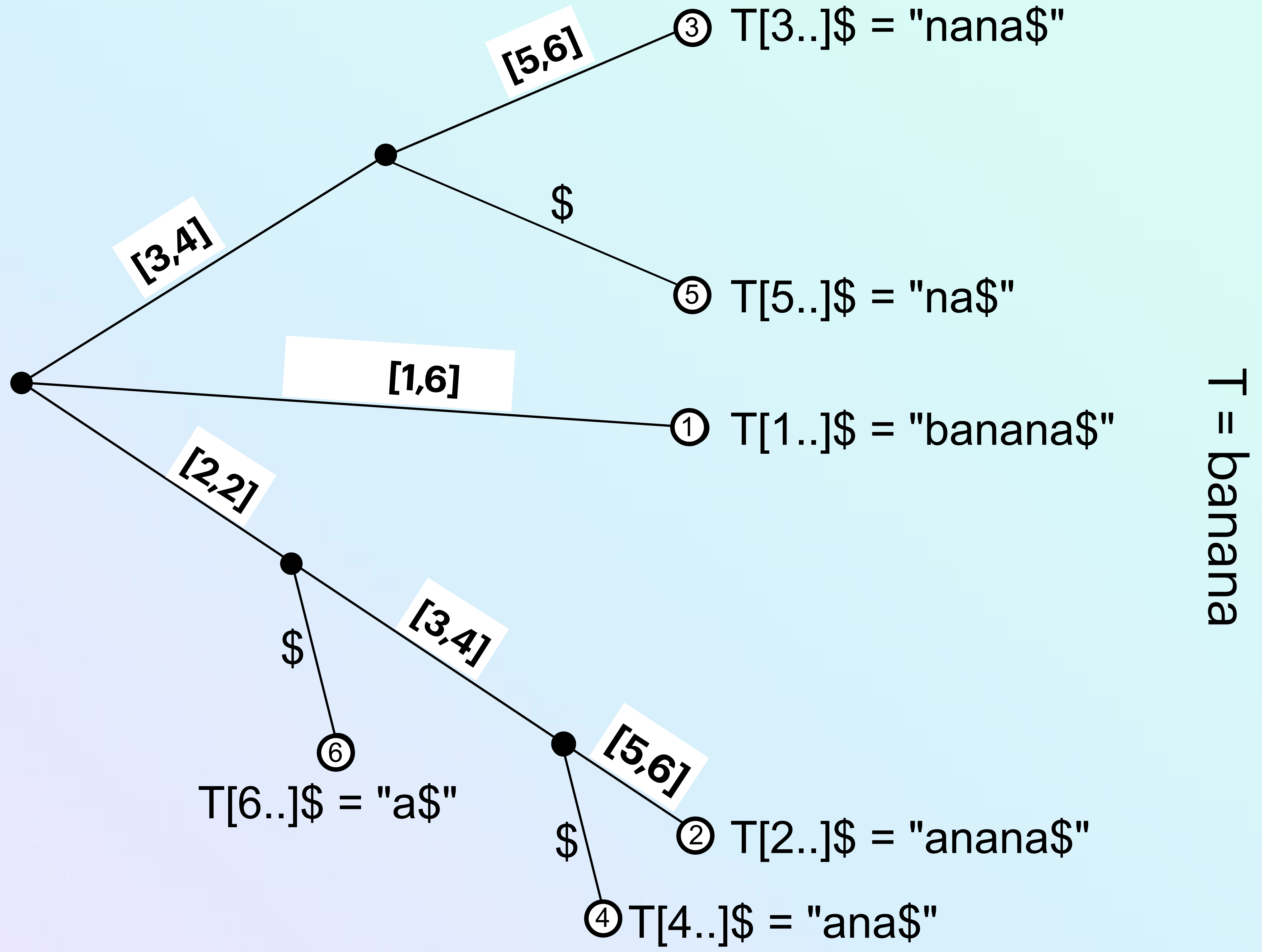
Storing the labels on the edges can take $\Theta(|T|^2)$ space.

To save the space, we represent each label as two numbers: the left and the right endpoints of the label in T.

Number of leaves: $|T|$

Number of nodes: $\leq 2|T| - 1$

Number of edges: $\leq 2|T| - 2$

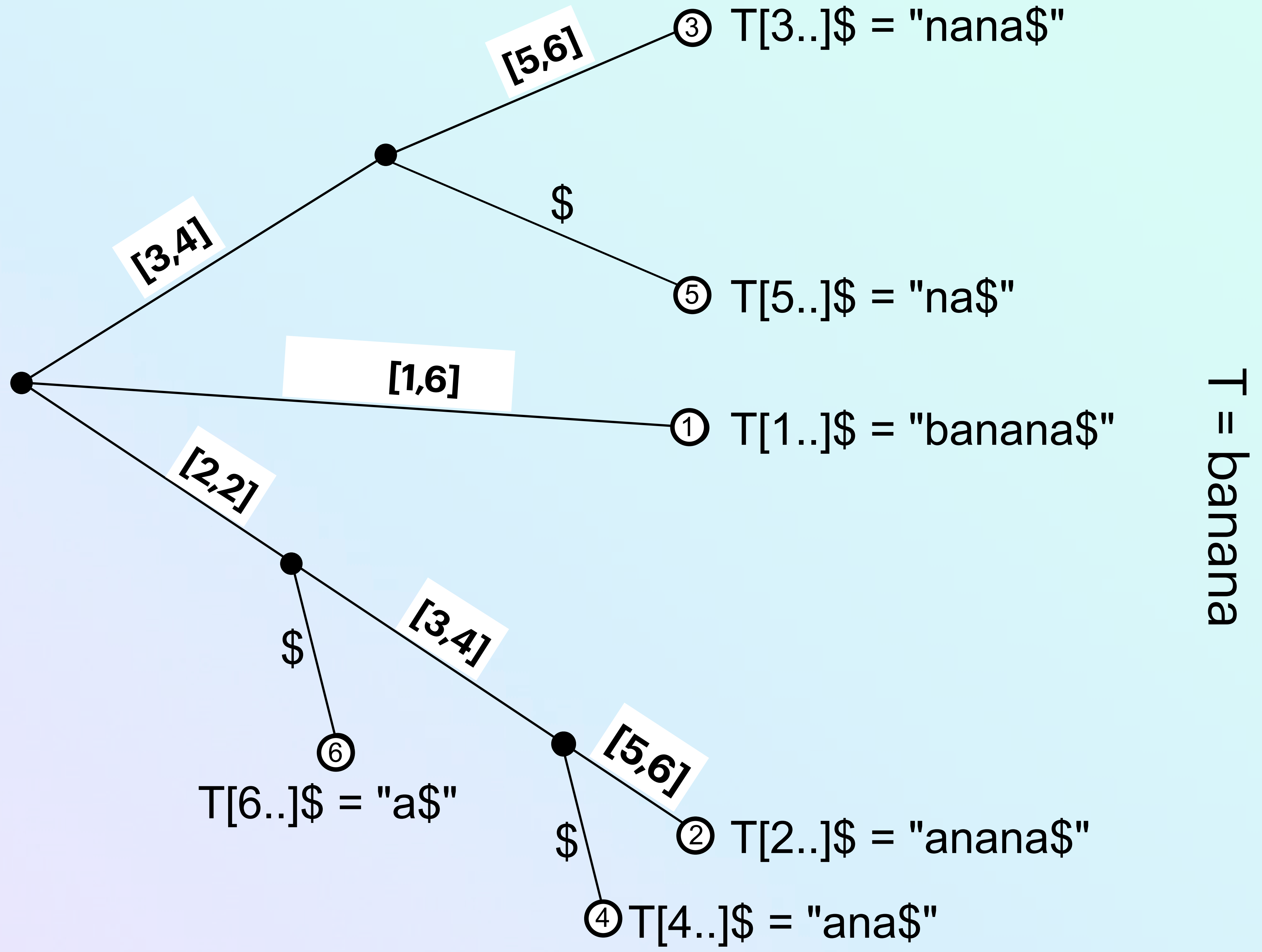


this is the final suffix tree!

SUFFIX TREE

Can be built in $O(|T|)$ time for **any** alphabet [Farach'97]

Exercise: How much time do we need to build a tree that contains suffixes of the text T and the pattern P?

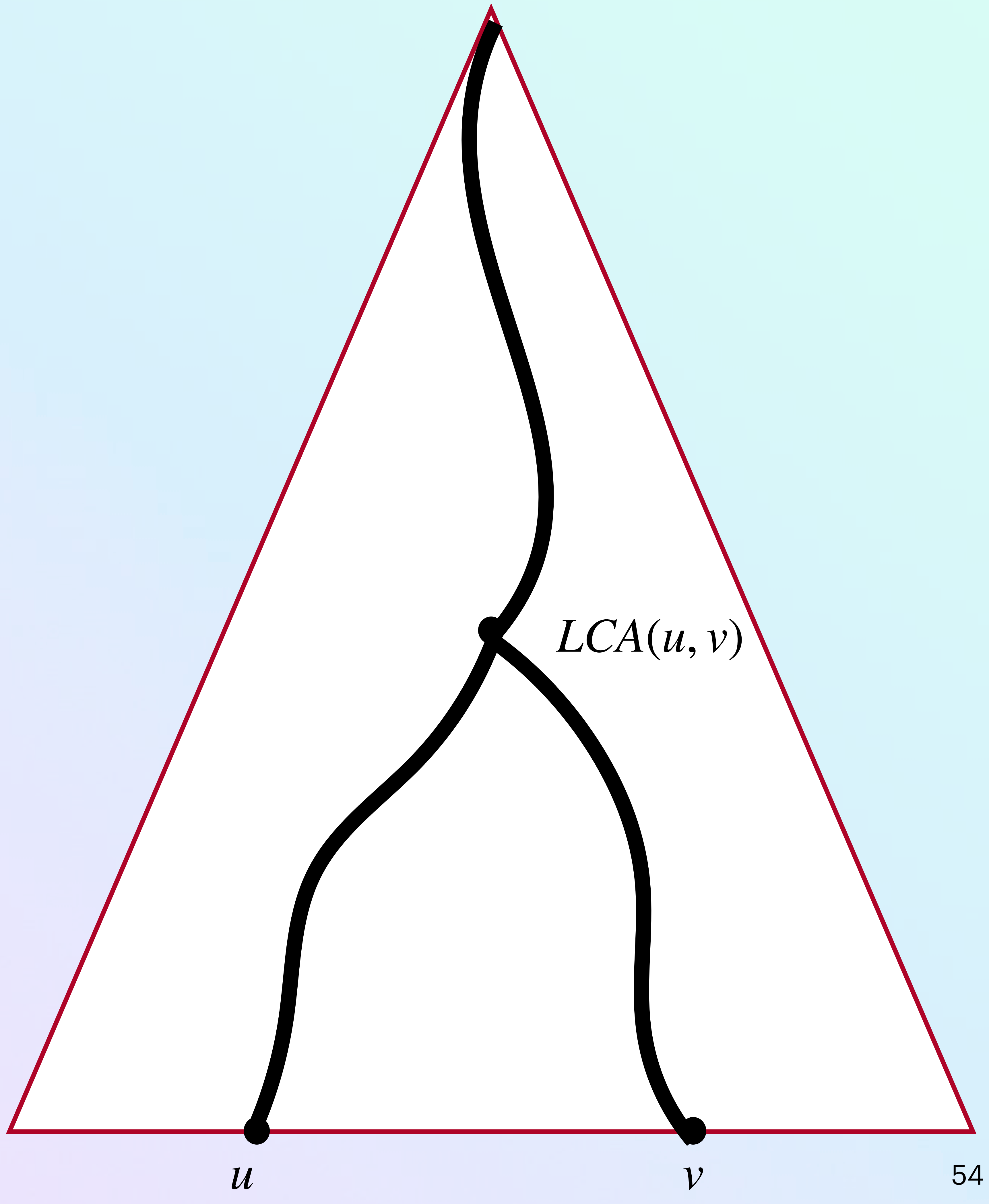


this is the final suffix tree!

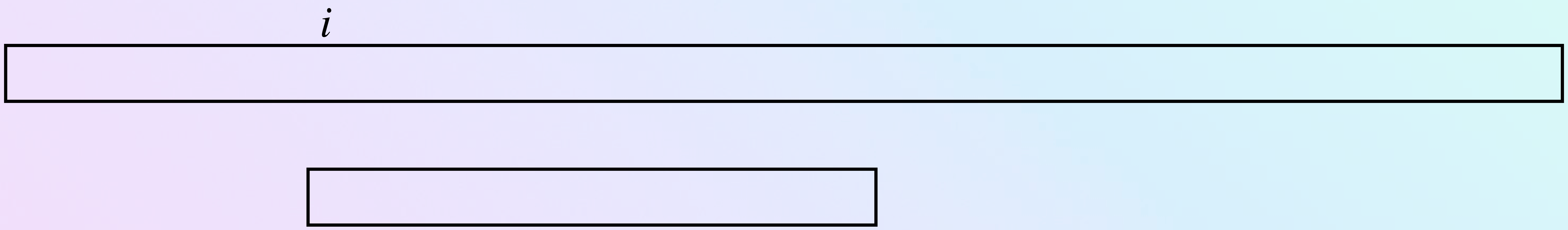
LOWEST COMMON ANCESTORS

A tree of size $O(n)$ can be processed in time $O(n)$ to support lowest common ancestor (LCA) queries in constant time. [Fischer, Hein'06]

$LCA(u, v)$ must return the lowest node that is an ancestor of both u and v.



KANGAROO JUMPS

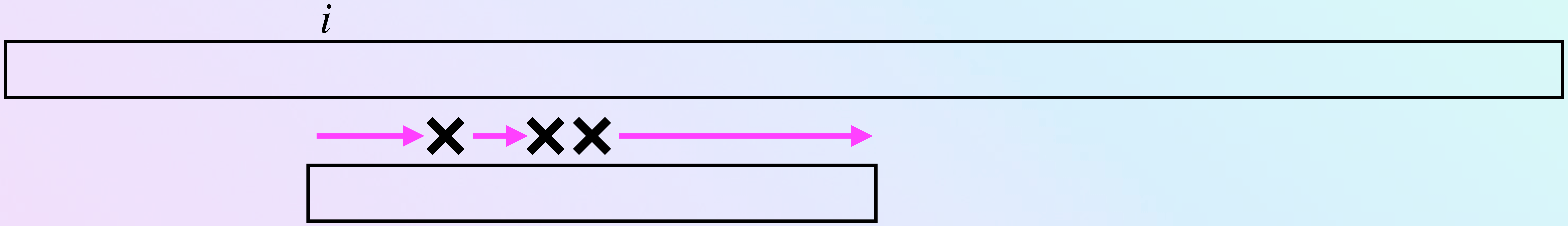


How to decide if the Hamming distance between P and T at position i is at most k?

Imagine that there is an oracle that tells us the maximum ℓ such that $T[k, k+\ell] = P[j, j+\ell]$ in $O(1)$ time.

Exercise: using the oracle, the question above can be solved in $O(k)$ time.

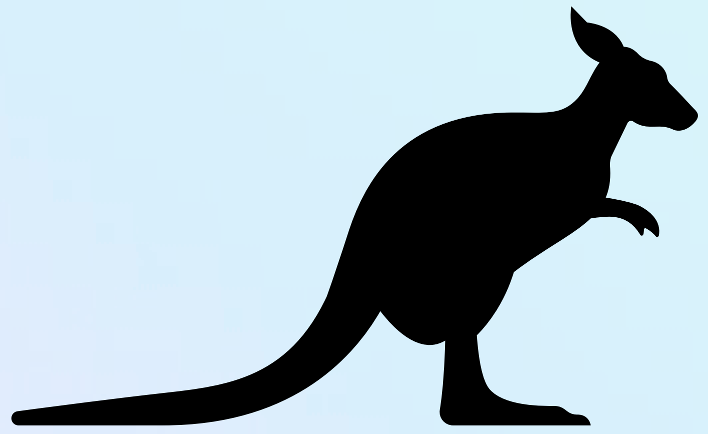
KANGAROO JUMPS



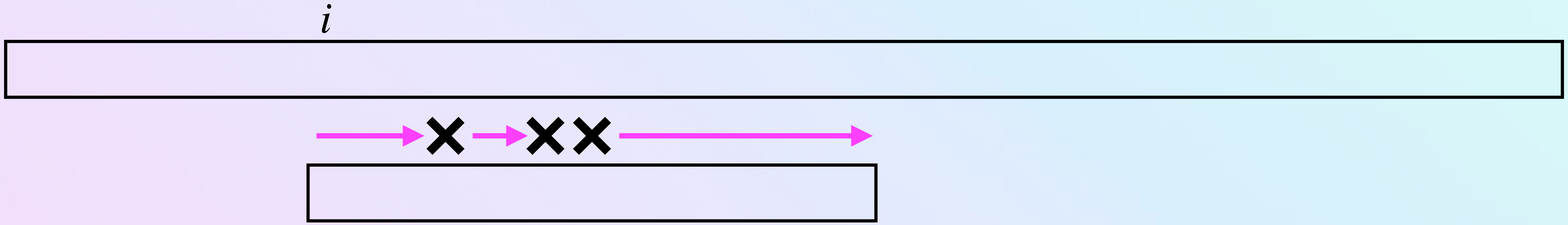
How to decide if the Hamming distance between P and T at position i is at most k?

Imagine that there is an oracle that tells us the maximum ℓ such that $T[k, k+\ell] = P[j, j+\ell]$ in $O(1)$ time.

Exercise: using the oracle, the question above can be solved in $O(k)$ time.



KANGAROO JUMPS

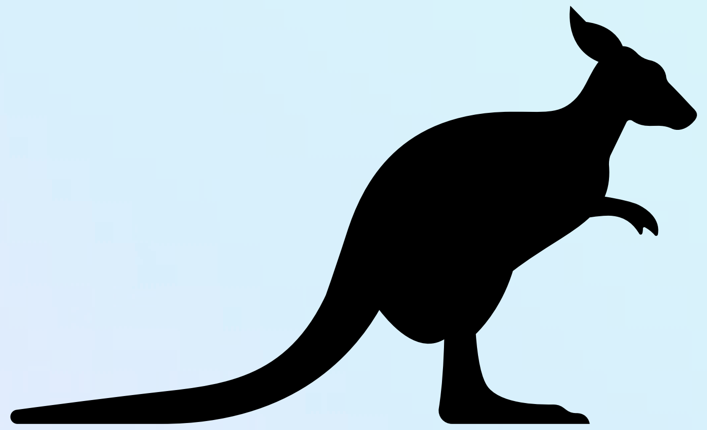


How to decide if the Hamming distance between P and T at position i is at most k?

Imagine that there is an oracle that tells us the maximum ℓ such that $T[k, k+\ell] = P[j, j+\ell]$ in $O(1)$ time.

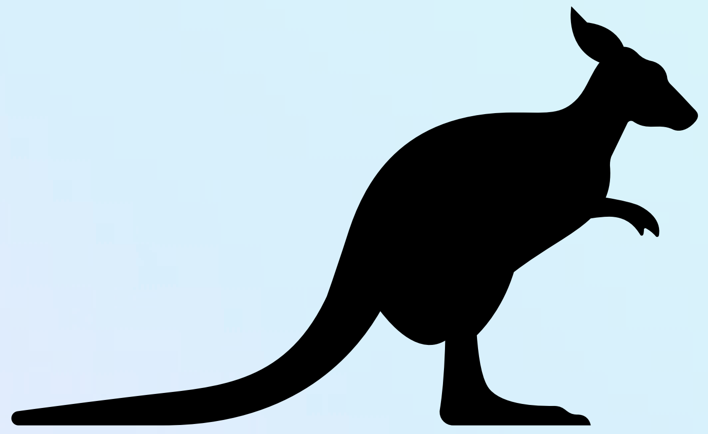
Exercise: using the oracle, the question above can be solved in $O(k)$ time.

Exercise: implement the oracle using suffix trees.



KANGAROO JUMPS

- $O(n+m) = O(n)$ time and space to build the suffix tree containing the suffixes of the pattern and the text
- $O(n+m) = O(n)$ time to preprocess it for lowest common ancestor queries
- $O(k)$ time per position to compute $\min(k+1, \text{Ham})$
- $O(nk)$ time and $O(n)$ space in total!



K-MISMATCH

TIME $O(nk)$, SPACE $O(n)$

ARE THERE FASTER ALGORITHMS?

	Time
Amir et al.'04	$O(n\sqrt{k \log k})$
Amir et al.'04	$O((n + \frac{n}{m} \cdot k^3 \log k))$
Clifford et al.'16	$O((n + \frac{n}{m} \cdot k^2) \text{ polylog} n)$
Gawrychowski and Uznański'18	$O((m \log^2 m \log \lvert\Sigma\rvert + k\sqrt{m \log m}) \cdot n/m)$
Charalampoupoulos et al.'20	$O(n + \frac{n}{m} \cdot k^2 \log \log k)$

$O(n)$ space!

ARE THERE FASTER ALGORITHMS?

	Time
Amir et al.'04	$O(n\sqrt{k \log k})$
Amir et al.'04	$O((n + \frac{n}{m} \cdot k^3 \log k))$
Clifford et al.'16	$O((n + \frac{n}{m} \cdot k^2) \text{ polylog} n)$
Gawrychowski and Uznański'18	$O((m \log^2 m \log \lvert \Sigma \rvert + k\sqrt{m \log m}) \cdot n/m)$
Charalampoupoulos et al.'20	$O(n + \frac{n}{m} \cdot k^2 \log \log k)$

$O(n)$ space!

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

- **Small alphabet:** Number of different characters in the pattern is at most $2\sqrt{k}$
- **Medium-size alphabet:** Number of different characters in the pattern is in $[2\sqrt{k}+1, 2k)$
- **Large alphabet:** Number of different characters in the pattern is at least $2k$

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

$O(n\sqrt{k}\log m)$ time (convolutions)

- **Small alphabet:** Number of different characters in the pattern is at most $2\sqrt{k}$
- **Medium-size alphabet:** Number of different characters in the pattern is in $[2\sqrt{k}+1, 2k)$
- **Large alphabet:** Number of different characters in the pattern is at least $2k$

$O(n\sqrt{k} \log m)$ TIME

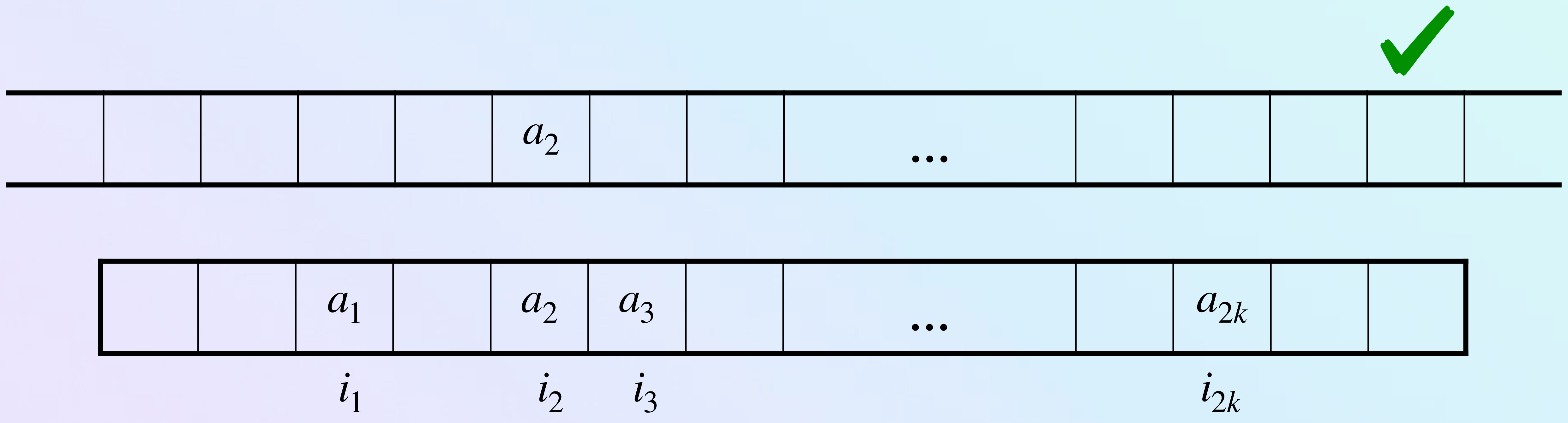
AMIR ET AL.'04

Large alphabet: Number of different symbols in the pattern is at least $2k$

Let $a_1, a_2, \ldots, a_{2k}$ be distinct characters in the pattern, and $i_1, i_2, \ldots, i_{2k}$ be the positions where they appear first in the pattern

For each i, $1 \leq i \leq n$: if $t_i = a_j$, mark $m + i - i_j$

Discard all text locations with less than k marks



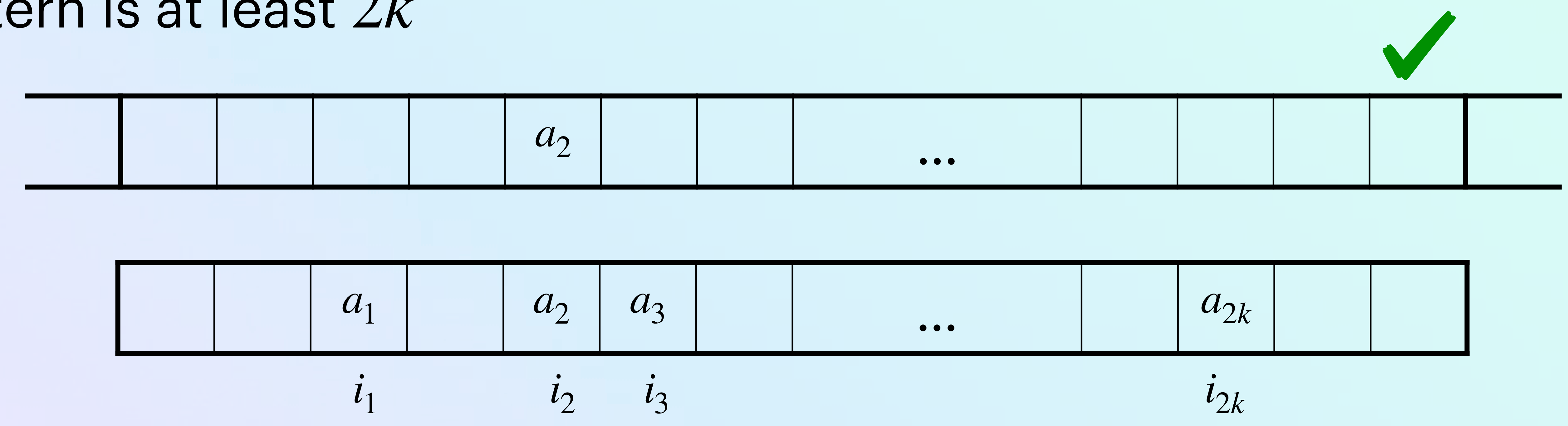
$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Large alphabet: Number of different symbols in the pattern is at least $2k$

For each i, $1 \le i \le n$: if $t_i = a_j$, mark $m + i - i_j$

Discard all text locations with less than k marks



Total number of marks is n, hence the number of **non-discarded positions** is $O(n/k)$

The endpoint of every k-mismatch occurrence must have at least k marks

Verification of non-discarded positions: $O(\frac{n}{k} \cdot k) = O(n)$ time using **kangaroo jumps**

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Medium-size alphabet: Number of different characters in the pattern is in $[2\sqrt{k}+1, 2k)$

A character that appears in the pattern at least $2\sqrt{k}$ times is called **frequent**

We consider two subcases:

- Number of frequent characters is at least $\sqrt{k}$
- Number of frequent characters is less than $\sqrt{k}$

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Number of frequent (= occurs in the pattern at least $2\sqrt{k}$ times) characters is at least $\sqrt{k}$

Exercise: show that the number of k-mismatch occurrences is $O(n/\sqrt{k})$

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Number of frequent (= occurs in the pattern at least $2\sqrt{k}$ times) characters is at least $\sqrt{k}$

Exercise: show that the number of k-mismatch occurrences is $O(n/\sqrt{k})$

- **Hint 1:** Use marks and the pigeonhole principle!

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Number of frequent (= occurs in the pattern at least $2\sqrt{k}$ times) characters is at least $\sqrt{k}$

Exercise: show that the number of k-mismatch occurrences is $O(n/\sqrt{k})$

- **Hint 1:** Use marks and the pigeonhole principle!
- **Hint 2:** Choose $\sqrt{k}$ frequent characters, and for each of them $2\sqrt{k}$ occurrences in the pattern

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Number of frequent (= occurs in the pattern at least $2\sqrt{k}$ times) characters is at least $\sqrt{k}$

Exercise: show that the number of k-mismatch occurrences is $O(n/\sqrt{k})$

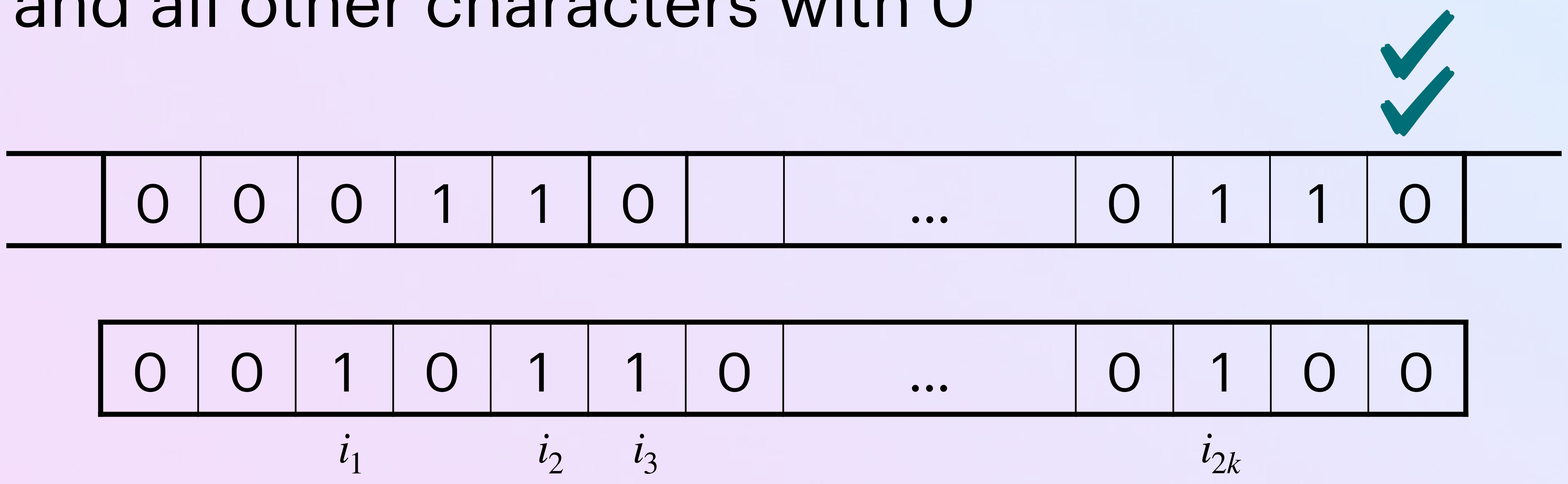
- We have $O(n/\sqrt{k})$ possible locations of k-mismatch occurrences (locations with $\geq k$ marks!)
- Each of them can be verified in $O(k)$ time via kangaroo jumps ($O(n\sqrt{k})$ in total)

Yes, but how do we find locations with $\geq k$ marks?!! Let's see...

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Replace all chosen occurrences of a frequent character in the pattern and in the text with 1, and all other characters with 0



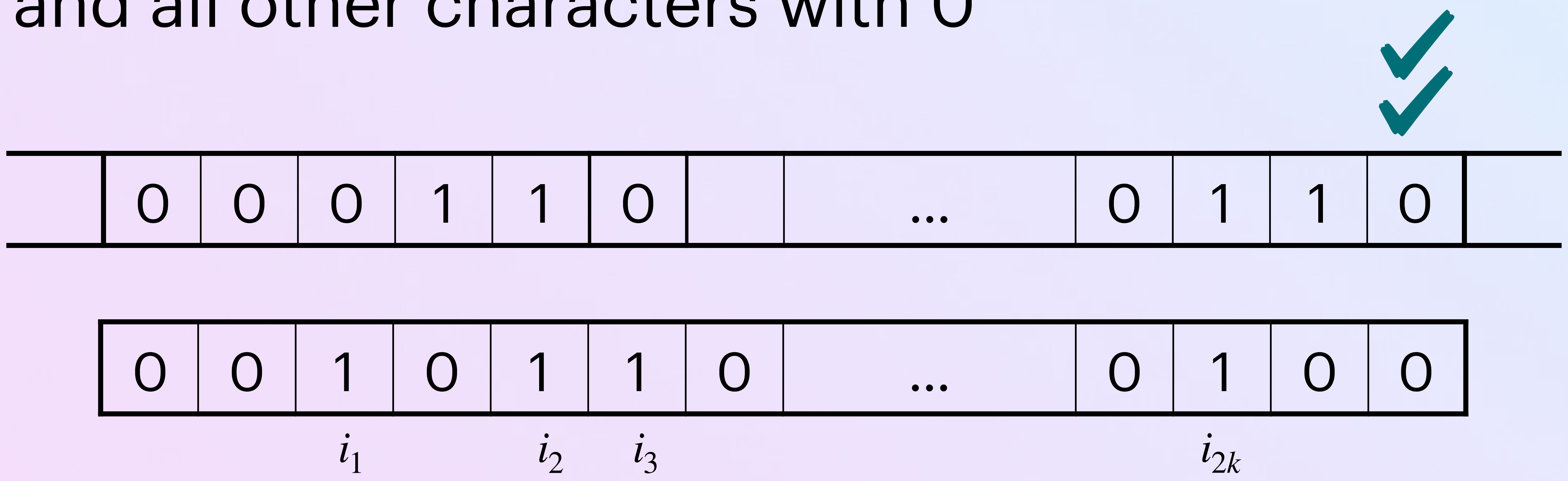
What can we say about the number of marks at a particular location?

How to compute this number?

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Replace all chosen occurrences of a frequent character in the pattern and in the text with 1, and all other characters with 0



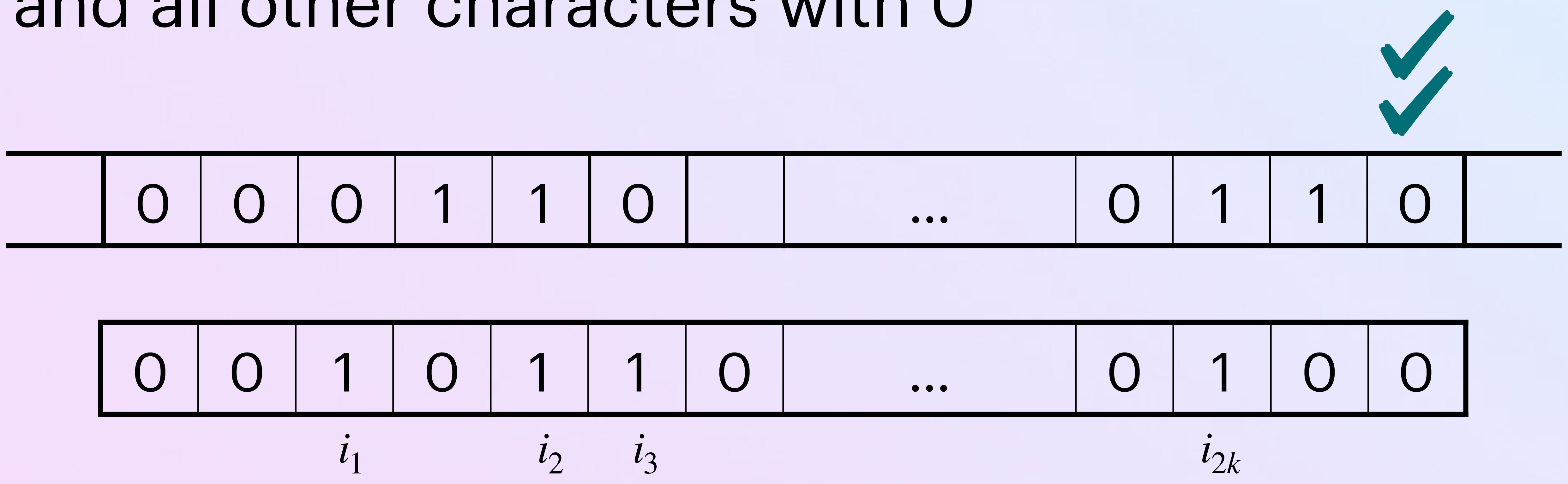
What can we say about the number of marks at a particular location? **(# of matching ones!)**

How to compute this number? **(Use convolutions, $O(n \log m)$ time)**

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Replace all chosen occurrences of a frequent character in the pattern and in the text with 1, and all other characters with 0



Marking step takes $O(n\sqrt{k}\log m)$ time

What can we say about the number of marks at a particular location? **(# of matching ones!)**

How to compute this number? **(Use convolutions, $O(n \log m)$ time)**

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Number of frequent (= occurs in the pattern at least $2\sqrt{k}$ times) characters is less than $\sqrt{k}$

- **Mismatches caused by frequent characters** can be computed in $O(n\sqrt{k}\log m)$ time (similar to the marking step we have just seen)
- We must "only" compute the mismatches due to non-frequent characters...

  A. Total number of occurrences of such characters is at least $2k$

  B. Total number of occurrences of such characters is less than $2k$

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Number of frequent (= occurs in the pattern at least $2\sqrt{k}$ times) characters is less than $\sqrt{k}$

- **Mismatches caused by frequent characters** can be computed in $O(n\sqrt{k}\log m)$ time (similar to the marking step we have just seen)
- We must "only" compute the mismatches due to non-frequent characters…

  A. Total number of occurrences of such characters is at least $2k$

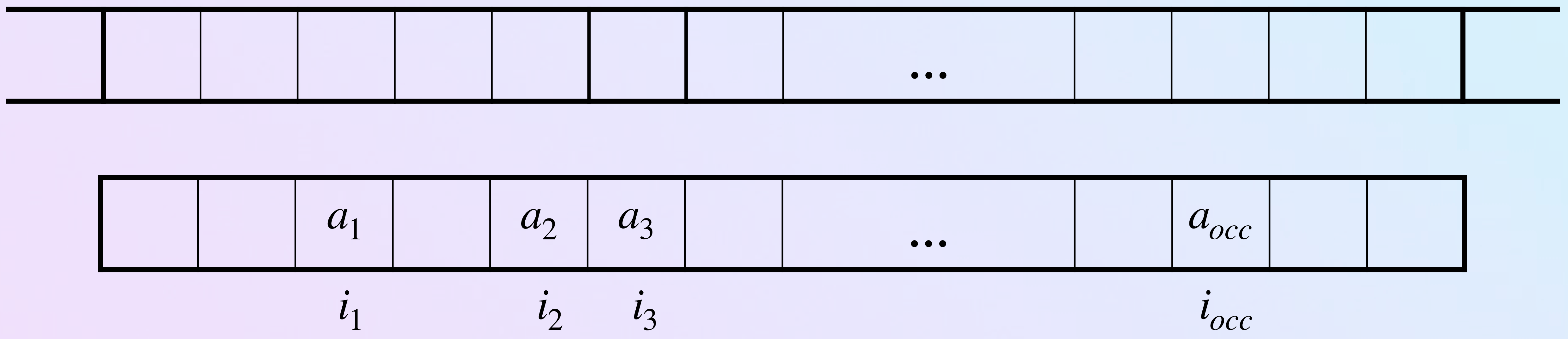
  Similar to the large alphabets!

  B. Total number of occurrences of such characters is less than $2k$

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Mismatches due to non-frequent characters, total number of occurrences $\leq 2k$

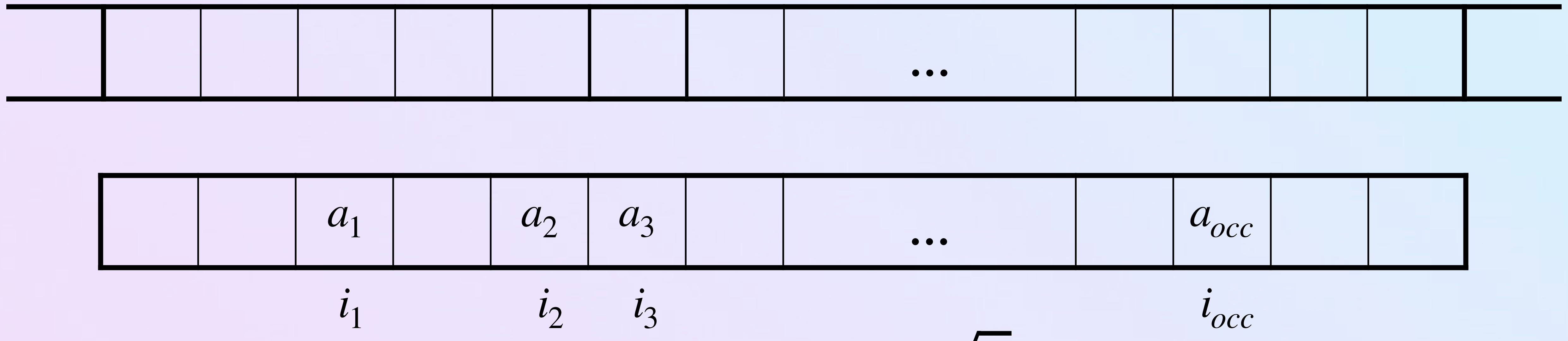


- Sort all non-frequent characters
- Divide them into $O(\sqrt{k})$ blocks of size $2\sqrt{k}$ so that one character appears only in one block
- Replace each character with the first character in its block (in the text and in the pattern)

$O(n\sqrt{k}\log m)$ TIME

AMIR ET AL.'04

Mismatches due to non-frequent characters, total number of occurrences $\leq 2k$



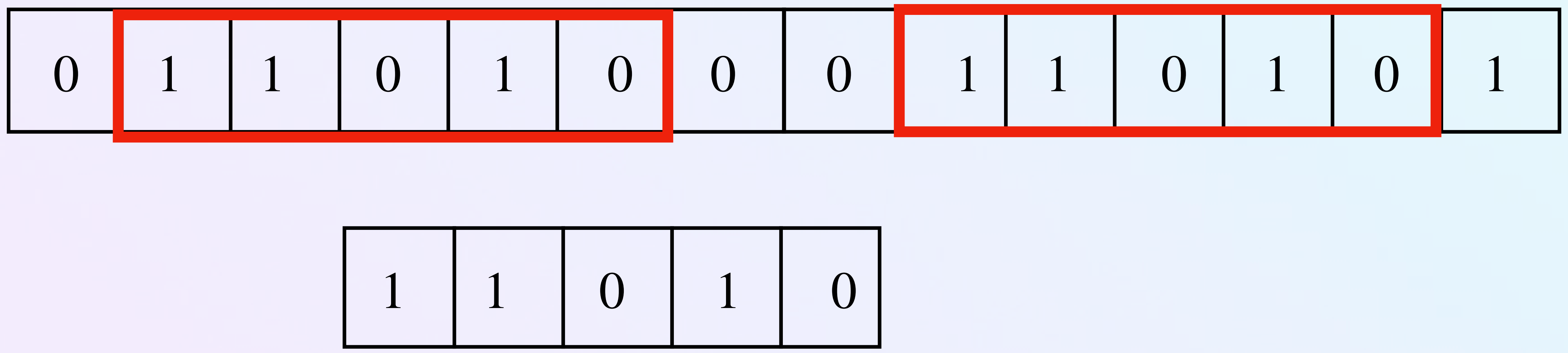
- Compute all text-to-pattern distances using $O(\sqrt{k})$ convolutions (this accounts for all mismatches when a text character and a pattern character are in different block) - $O(n\sqrt{k}\log m)$ **time!**
- For each text character, account for at most $2\sqrt{k}$ mismatches that appear when the character and the aligned pattern character are in the same block - $O(n\sqrt{k}\log m)$ **time!**

K-MISMATCH

TIME $O(n\sqrt{k}\log m)$, SPACE $O(n)$

SMALLER SPACE

EXACT PATTERN MATCHING



Given a pattern of length m and a text of length n, find all **occurrences** of the pattern in the text

EXACT PATTERN MATCHING

- More than 80 algorithms known!
- Implemented in SMART, String Matching Algorithms Research Tool: https://smart-tool.github.io/smart/
- Today we will discuss the algorithm of Karp and Rabin from 1987
- Again, some of you have probably seen it and some will see this year..
- Promise: I will show you a **new, much simpler analysis**

EXACT PATTERN MATCHING

KARP AND RABIN'87

The **Karp-Rabin fingerprint** of a string $S = s_1 s_2 \ldots s_m$ is defined as

$$\varphi(s_1 s_2 \ldots s_m) = \sum_{i=1}^{m} s_i \cdot r^{m-i} \bmod p,$$

where p is a prime and r is a random integer in $\mathbb{F}_p$.

It's a **good hash function**:

- If $S = T$, then $\varphi(S) = \varphi(T)$;
- If $S \neq T$ while the lengths of S and T are equal,

then $\varphi(S) \neq \varphi(T)$ with high probability (if p is large enough).

Let's zoom in...

EXACT PATTERN MATCHING

KARP AND RABIN'87

Let $S = s_1 s_2 \ldots s_m$, $T = t_1 t_2 \ldots t_m$, and σ be the size of the alphabet. Let $p \geq \max\{\sigma, n^c\}$, where $c > 1$ is a constant.

$$\varphi(S) = \varphi(T) \Leftrightarrow \sum_{i=1}^{m} (s_i - t_i) \cdot r^{m-i} \bmod p = 0$$

Hence, r is a root of $P(x) = \sum_{i=1}^{m} (s_i - t_i) \cdot x^{m-i}$, a polynomial over $\mathbb{F}_p$. The number of roots of this polynomial is at most m. The **probability of such event is at most** $m/p \leq 1/n^{c-1}$.

EXACT PATTERN MATCHING

KARP AND RABIN'87

- Compute the fingerprint of the pattern.
- Compare it with the fingerprint of each m-length substring of the text. If the fingerprint of the pattern is equal to the fingerprint of a substring, report it as an occurrence.
- The algorithm **never misses an occurrence** (no false-negatives)
- False-positives can happen with probability at most $1/n^{c-1}$

EXACT PATTERN MATCHING

KARP AND RABIN'87

How to compute the fingerprints?

$$\varphi(s_1 s_2 \ldots s_m) = \sum_{i=1}^{m} s_i \cdot r^{m-i} \bmod p$$

$$\varphi(s_2 \ldots s_{m+1}) = \sum_{i=1}^{m} s_{i+1} \cdot r^{m-i} \bmod p$$

This is why it's "rolling"!

Therefore, $\varphi(s_2 \ldots s_{m+1}) = (\varphi(s_1 s_2 \ldots s_m) - s_1 \cdot r^{m-1}) \cdot r + s_{m+1} \bmod p$.

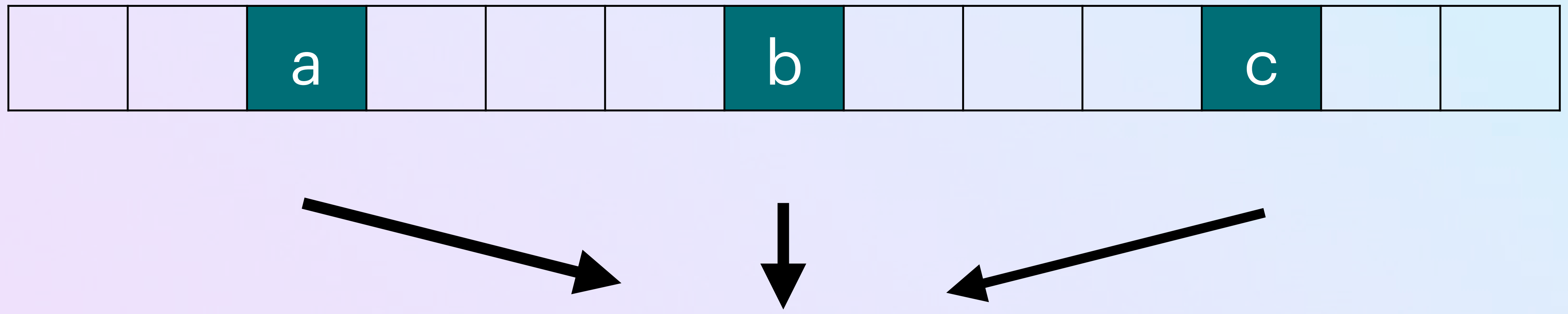
We can compute the fingerprint of the $(i+1)$-th m-length substring of the text from the fingerprint of the i-th substring in $O(1)$ space and $O(1)$ time.

Karp-Rabin algorithm: $O(1)$ extra space, $O(1)$ time per letter of the text

1-MISMATCH

BASED ON PORAT AND PORAT'09

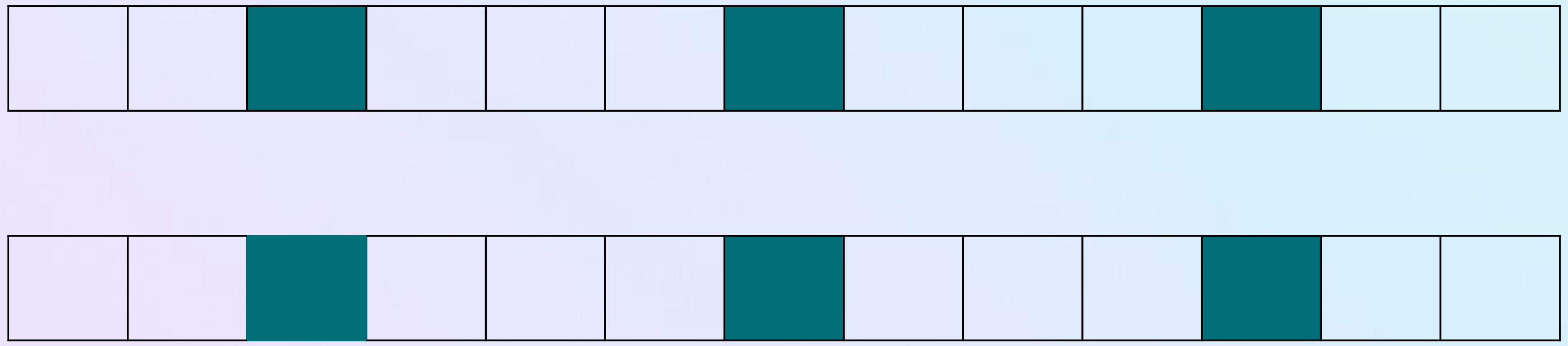
For a string X, define a string $X_r^q = X[q]X[q+r]X[q+2r]\ldots$



$$X_4^3 = abc$$

1-MISMATCH

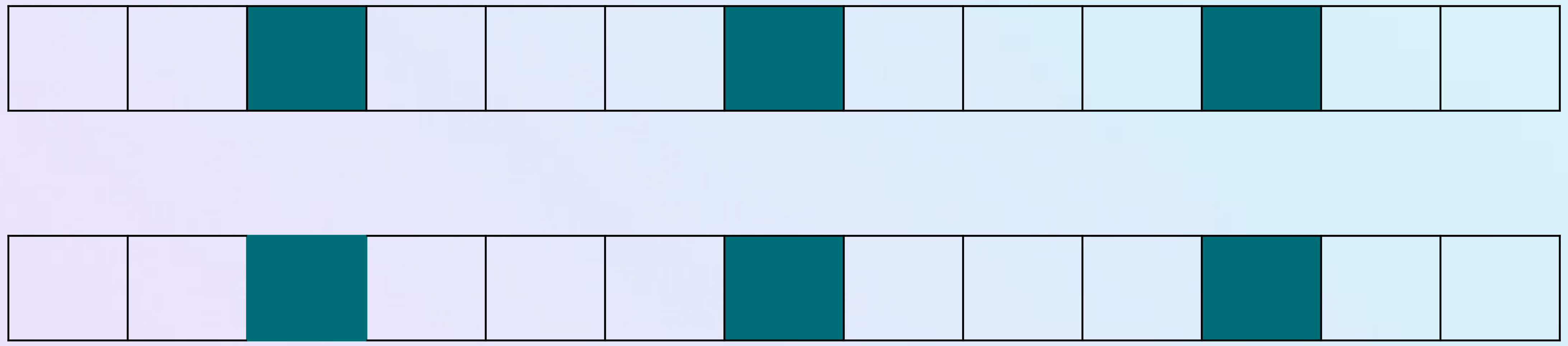
BASED ON PORAT AND PORAT'09



- Consider two strings X, Y of length m
- Q is the set of $\log m$ smallest prime numbers. By the prime number theorem, $\max Q \leq c \cdot \log m \log \log m$
- For each $q \in Q, r \in \mathbb{F}_q$ consider substrings X_q^r, Y_q^r

1-MISMATCH

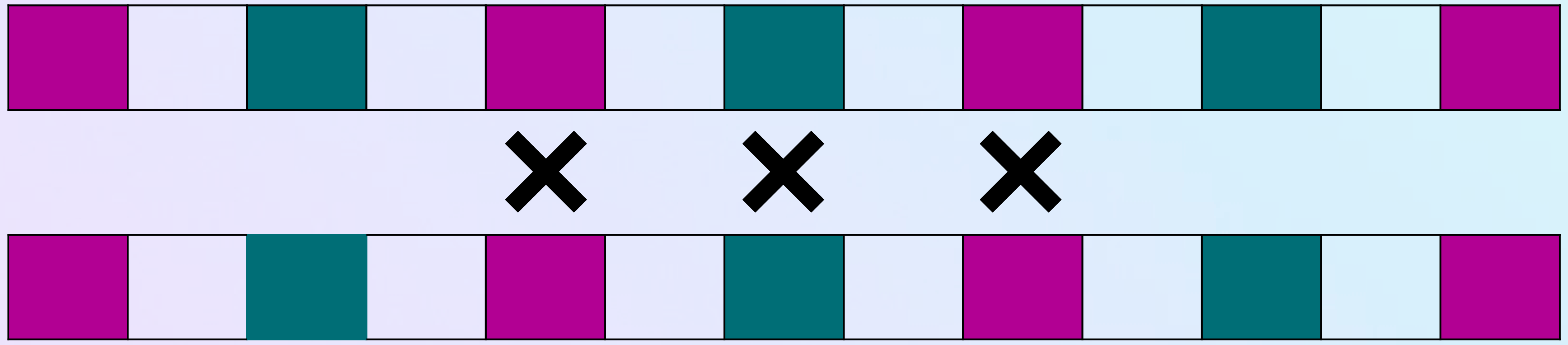
BASED ON PORAT AND PORAT'09



- If $X = Y$, what can we say about the number of mismatching pairs X_q^r, Y_q^r?
- And if the Hamming distance between X, Y is one?
- What if it is at least two?

1-MISMATCH

BASED ON PORAT AND PORAT'09

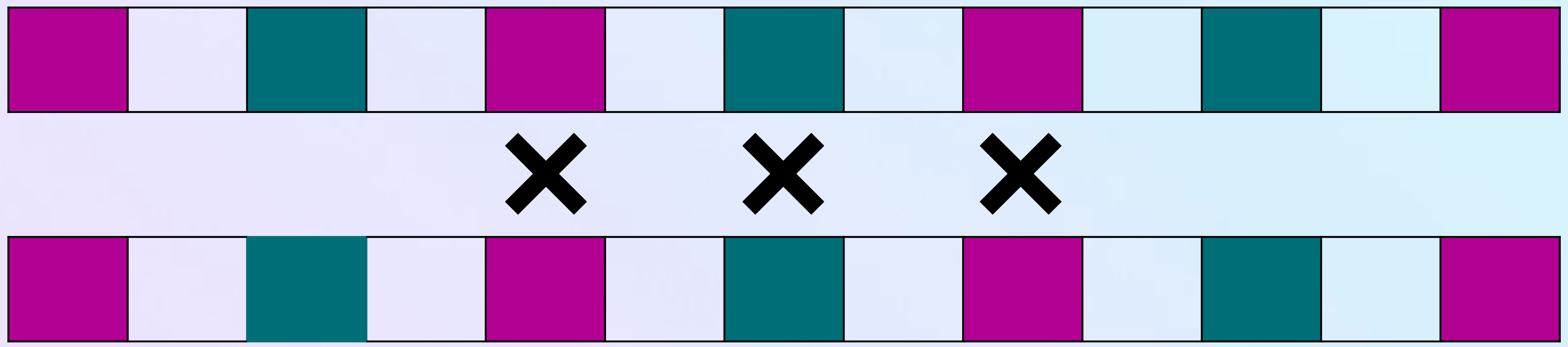


Lemma. If the Hamming distance between X, Y is at least two, then for some $q \in Q$ there exist $r_1, r_2 \in \mathbb{F}_p, r_1 \neq r_2$ such that $X_q^{r_1} \neq Y_q^{r_1}$ and $X_q^{r_2} \neq Y_q^{r_2}$.

Proof. Let $m_1 < m_2$ be the mismatch positions. If $m_1 = m_2 = r \pmod q$, then $m_2 - m_1 \vdots q$. However, $m \geq m_2 - m_1$ and $\Pi_{q \in Q} q > m$. The claim follows!

1-MISMATCH

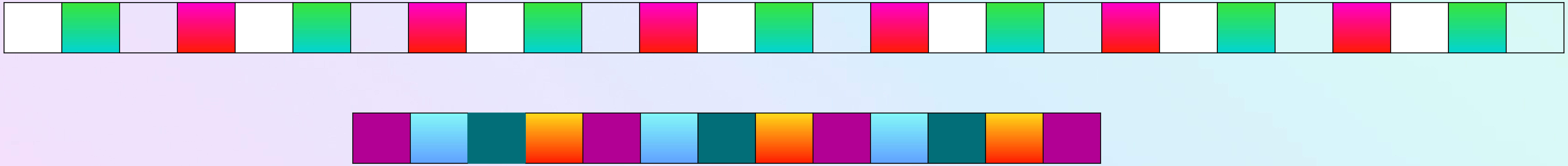
BASED ON PORAT AND PORAT'09



- Assume someone tells you which pairs X_q^r, Y_q^r are equal.
- How can you use this to deduce whether the Hamming distance between X, Y is one? Can you also deduce the mismatch position?

1-MISMATCH

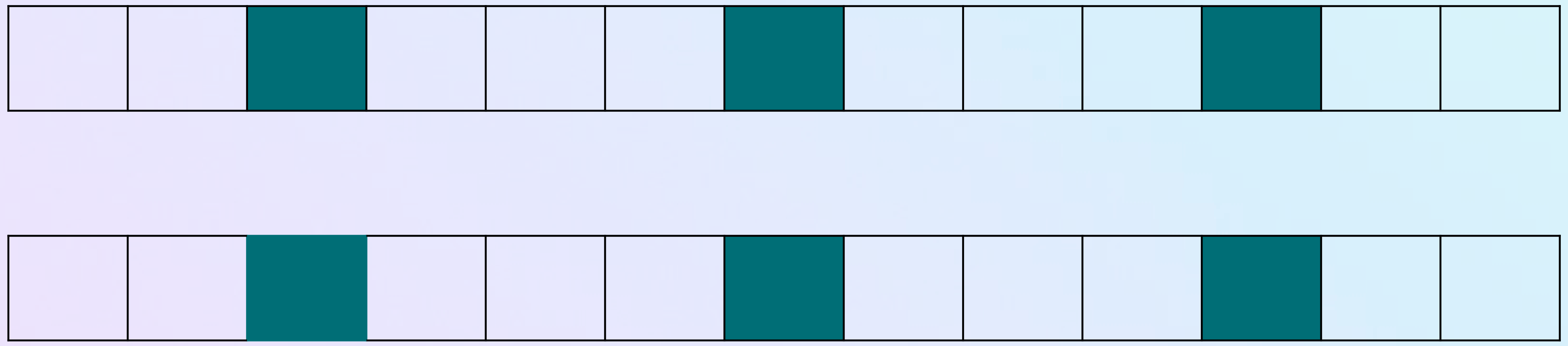
BASED ON PORAT AND PORAT'09



- Let's go back to computing text-to-pattern distances...
- For every $q \in Q, r_1, r_2 \in \mathbb{F}_q$ run the Karp-Rabin algorithm for $T_q^{r_1}$ and $P_q^{r_2}$
- This algorithm tells, for every m-length substring S, whether $S_q^r = P_q^r$
- Time $n \cdot \text{polylog } m$, (extra) space $\text{polylog } m$, error probability $1/n^c$

K-MISMATCH

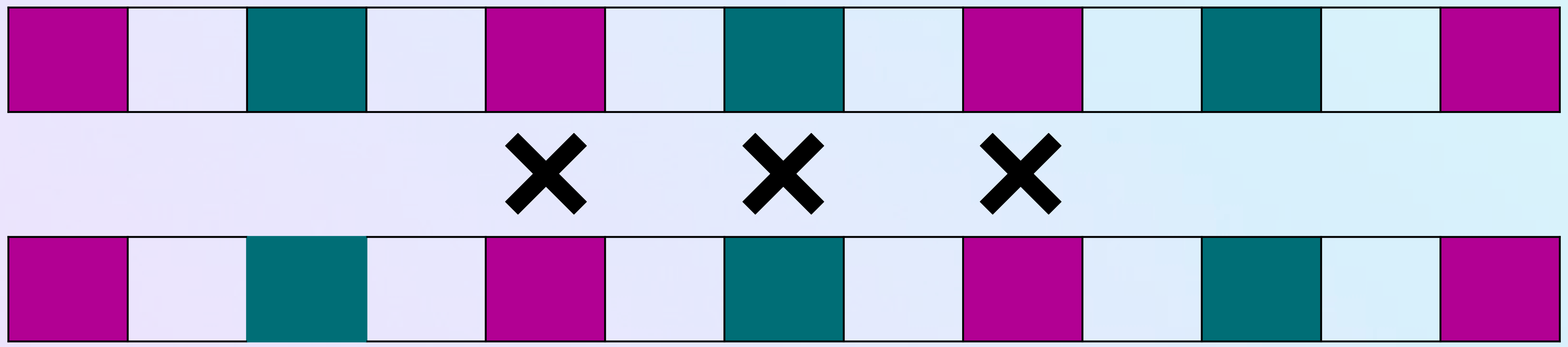
BASED ON PORAT AND PORAT'09



- Consider two strings X, Y of length m
- Q is the set of $k^2 \log m$ smallest prime numbers. By the prime number theorem, $\max Q \leq c \cdot k^2 \log m \log \log m$
- For each $q \in Q, r \in \mathbb{F}_q$ consider substrings X_q^r, Y_q^r

K-MISMATCH

BASED ON PORAT AND PORAT'09

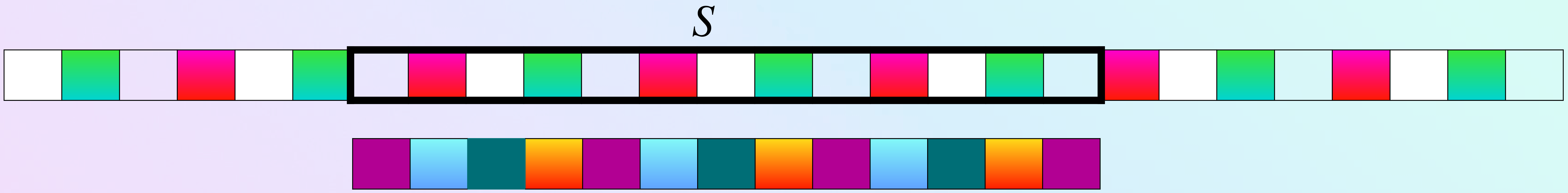


Lemma. Let $m_1, m_2, \ldots, m_\ell$, $\ell \leq k$, be mismatch positions between X, Y. For a fixed i and all $j \neq i$ there exists $q \in Q$ such that $m_i \neq m_j \pmod q$.

Proof (idea). m_j "spoils" $q \in Q$ if $m_i = m_j \pmod q$. We have seen that m_j can spoil at most $\log m$ primes, and hence there are $\leq (k-1)\log m$ spoiled primes in total. We can take any unspoiled prime to satisfy the claim of the lemma.

K-MISMATCH

BASED ON PORAT AND PORAT'09



- For every $q \in Q, r_1, r_2 \in \mathbb{F}_q$ run the 1-mismatch algorithm for $T_q^{r_1}$ and $P_q^{r_2}$
- If the algorithm tells that the Hamming distance between S_q^r, P_q^r is one and outputs the mismatch position, remember it!
- Fix all the mismatches output by the algorithm and check that the pattern equals S in $O(k)$ time using fingerprints
- Time $nk^2 \cdot \text{polylog } m$, (extra) space $k^3 \cdot \text{polylog } m$, error probability $1/n^c$

K-MISMATCH

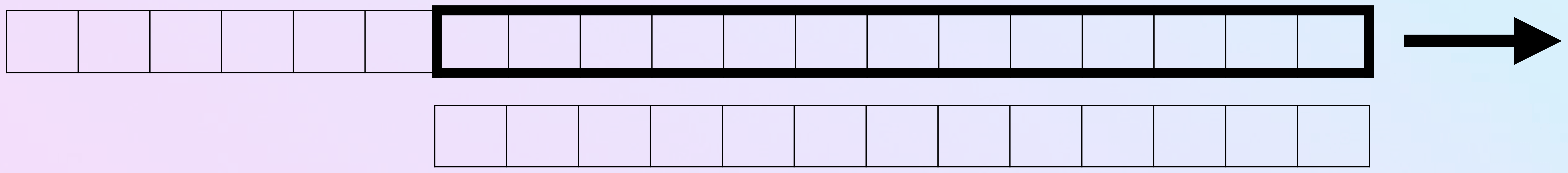
TIME $nk^2 \cdot \text{polylog } m$, **SPACE** $k^3 \cdot \text{polylog } m$

ISN'T THIS GREAT? YES, BUT...

- The algorithm we developed is randomised: we use Karp-Rabin algorithm
- We have seen faster AND deterministic algorithms!
- It uses $\Omega(k \text{ polylog } m)$ extra space, while all previous algorithms used $\Omega(n)$ space

ISN'T THIS GREAT? YES, BUT...

Porat and Porat'09 also showed an $O(\log m)$ space, $O(\log m)$ time **streaming algorithm** for exact pattern matching:



In the streaming setting,

- the text arrives one letter at a time
- we account for **all the space used**, including the space we need to store P and T

STREAMING K-MISMATCH

	Space	Time
Porat and Porat'09	$k^3 \text{polylog } m$	$nk^2 \text{polylog } m$
Clifford, Fontaine, Porat, Sach, Starikovskaya'16	$k^2 \text{polylog } m$	$n\sqrt{k} \text{polylog } m$
Golan, Kopelowitz, Porat'18	$k \text{polylog } m$	$nk \text{polylog } m$
Clifford, Kociumaka, Porat'19	$k \text{polylog } m$	$n\sqrt{k} \text{polylog } m$
Golan, Kociumaka, Kopelowitz, Porat'20	$s \cdot \text{polylog } m$	$n(k/s) \cdot \text{polylog} m$

STREAMING K-MISMATCH

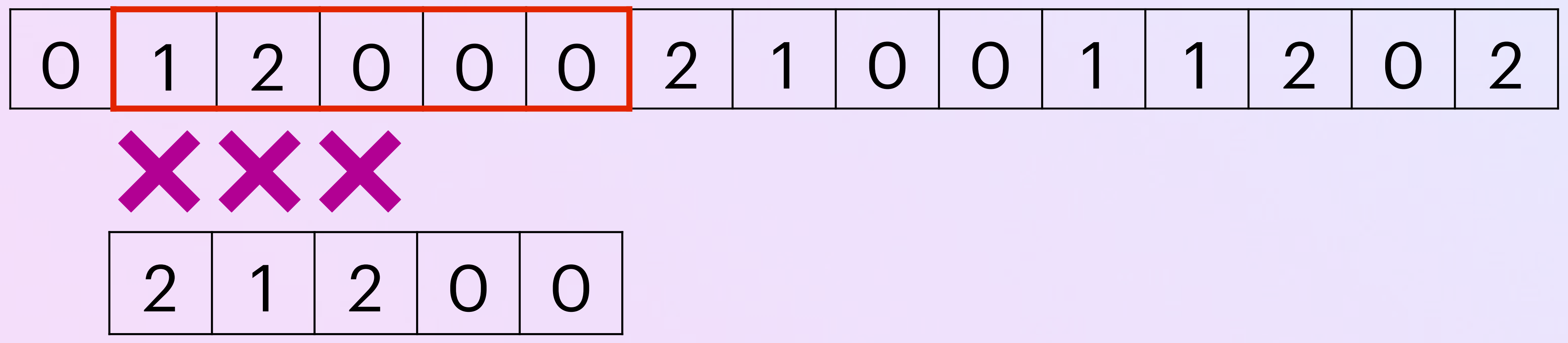
	Space	Time
Porat and Porat'09	$k^3 \text{polylog } m$	$nk^2 \text{polylog } m$
Clifford, Fontaine, Porat, Sach, Starikovskaya'16	$k^2 \text{polylog } m$	$n\sqrt{k} \text{polylog } m$
Golan, Kopelowitz, Porat'18	$k \text{polylog } m$	$\text{polylog } m$
Clifford, Kociumaka, Porat'19	$k \text{polylog } m$	$n\sqrt{k} \text{polylog } m$
Golan, Kociumaka, Kopelowitz, Porat'20	$s \cdot \text{polylog } m$	$n(k/s) \cdot \text{polylog} m$

Open question:
What's optimal space?

APPROXIMATION ALGORITHM

PROBLEM FORMULATION

Given a text T of length n, a pattern P of length m, and a constant $\varepsilon > 0$, for each m-length substring of T output a number between $(1-\varepsilon)\cdot d$ and $(1+\varepsilon)\cdot d$, where d is the Hamming distance between the substring and P



HD = 3, $\varepsilon = 1/3$,
output a number in [4,6]

WHAT DO WE KNOW?

	Time
Karloff'93	$\frac{n}{\varepsilon^2}\text{polylog}n$
Kopelowitz and Porat'15	$O(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon}\log n\log m)$
Kopelowitz and Porat'18	$O(\frac{n}{\varepsilon}\log n\log m)$
Chan et al.'20	$O(n/\varepsilon^2)$

WHAT DO WE KNOW?

	Time
Karloff'93	$\frac{n}{\varepsilon^2}\text{polylog}n$
Kopelowitz and Porat'15	$O(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon}\log n\log m)$
Kopelowitz and Porat'18	$O(\frac{n}{\varepsilon}\log n\log m)$
Chan et al.'20	$O(n/\varepsilon^2)$

Open question:
Can the dependency on ε be improved?

ALGORITHM

KOPELOWITZ AND PORAT'18

$\texttt{ApproxHam}(T[j, j + m - 1], P, \varepsilon)$

for $i = 1$ to $c \log n$

do: Pick a random $h : \Sigma \rightarrow \{1,2,\ldots,\frac{1}{\varepsilon}\}$

compute $x_i = Ham(T[j, j + m - 1], P)$

return $\max_{1 \leq i \leq c \log n} x_i$

CORRECTNESS

KOPELOWITZ AND PORAT'18

$d := \text{Ham}(T[j, j+m-1], P)$

$\mathbb{E}[x_i] = (1 - \frac{\varepsilon}{2}) \cdot d$ (after applying h, each mismatch remains a mismatch with probability $1 - \frac{\varepsilon}{2}$)

$\mathbb{E}[d - x_i] = \frac{\varepsilon}{2} \cdot d$

CORRECTNESS

KOPELOWITZ AND PORAT'18

$d := \mathrm{Ham}(T[j, j+m-1], P)$

$\mathbb{E}[x_i] = (1 - \frac{\varepsilon}{2}) \cdot d$ and therefore $\mathbb{E}[d - x_i] = \frac{\varepsilon}{2} \cdot d$

Markov's inequality

$$\Pr[x_i < (1-\varepsilon) \cdot d] = \Pr[d - x_i > \varepsilon \cdot d] \leq \frac{\mathbb{E}[d - x_i]}{\varepsilon d} = \frac{1}{2}$$

Finally, the error probability $\Pr[\max_i x_i < (1-\varepsilon) \cdot d] \leq 1/n^c$

IMPLEMENTATION

KOPELOWITZ AND PORAT'18

After picking a hash function for an iteration i, compute all text-to-pattern Hamming distances in time $O(\frac{n}{\varepsilon}\log m)$ using the algorithm for small alphabets!

Total time: $O(\frac{n}{\varepsilon}\log n \log m)$

TAKE HOME MESSAGE

- Exact algorithms for binary and general case (binary $O(n \log m)$ time, general $O(n\sqrt{m \log m})$)
- No combinatorial algorithm in time $O(nm^{1/2-\varepsilon})$ unless CMM conjecture is false
- $O(nk)$-time algorithm via **kangaroo jumps**, $O(n\sqrt{k} \log m)$ by combining kangaroo jumps + frequent characters + convolutions
- $n\sqrt{k}$ polylog$m$-time **streaming algorithm** that computes text-to-pattern Hamming distances bounded by k
- Approximation algorithm with runtime $O(\frac{n}{\varepsilon} \log n \log m)$

TAKE HOME MESSAGE

- If you have further questions or would like to discuss one of the open problems
- If you would like to do an internship (stage L3, stage M1) in this area (in France or abroad)

you can contact me via tat.starikovskaya@gmail.com

Interesting event: 2nd Workshop Complexity and Algorithms (IHP Paris, 26-28 September)

THANK YOU!