Three new strategies for exact string matching

Simone Faro[1] Thierry Lecroq[2]

[1] University of Catania, Italy

[2] University of Rouen, LITIS EA 4108, France

SeqBio 2012

November 26th-27th 2012 – Marne-la-Vallée, France

Outline

1. Introduction to exact string matching
2. Multiple windows
3. Substring with no repetition
4. Multiple hashing

Outline

1. Introduction to exact string matching
2. Multiple windows
3. Substring with no repetition
4. Multiple hashing

Exact String Matching

Definition

Find all the occurrences of a pattern x of length m in a text y of length n.
$x, y \in \Sigma^*$

2 instances

- x is given first
- y is given first

Exact String Matching

Interests

- basic components of many softwares
- theoretical problems

Exact String Matching

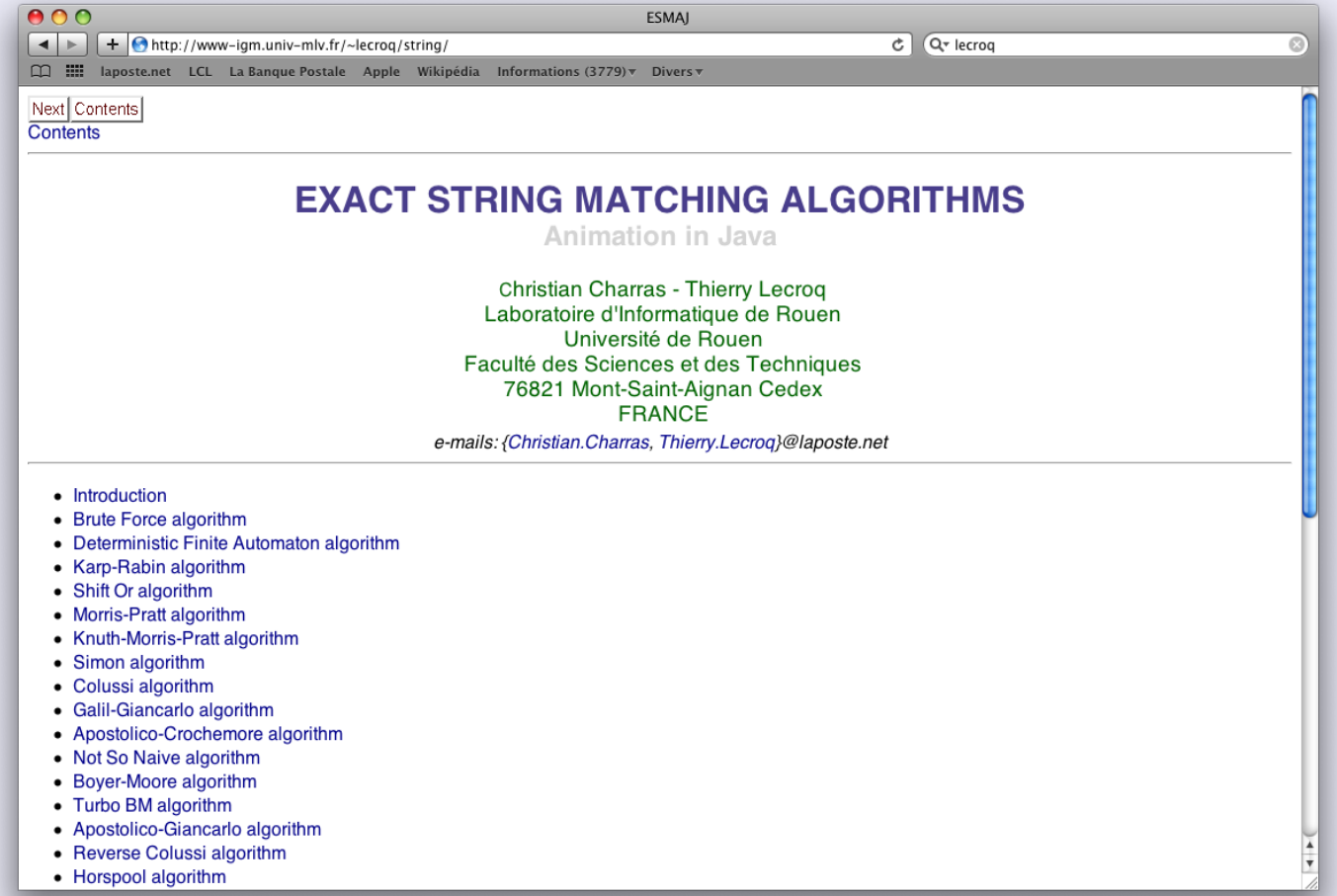
Theory

- linear time since Morris and Pratt 1970
- linear time and constant space
- $O((n \log m)/m)$ in average [Yao 1979]

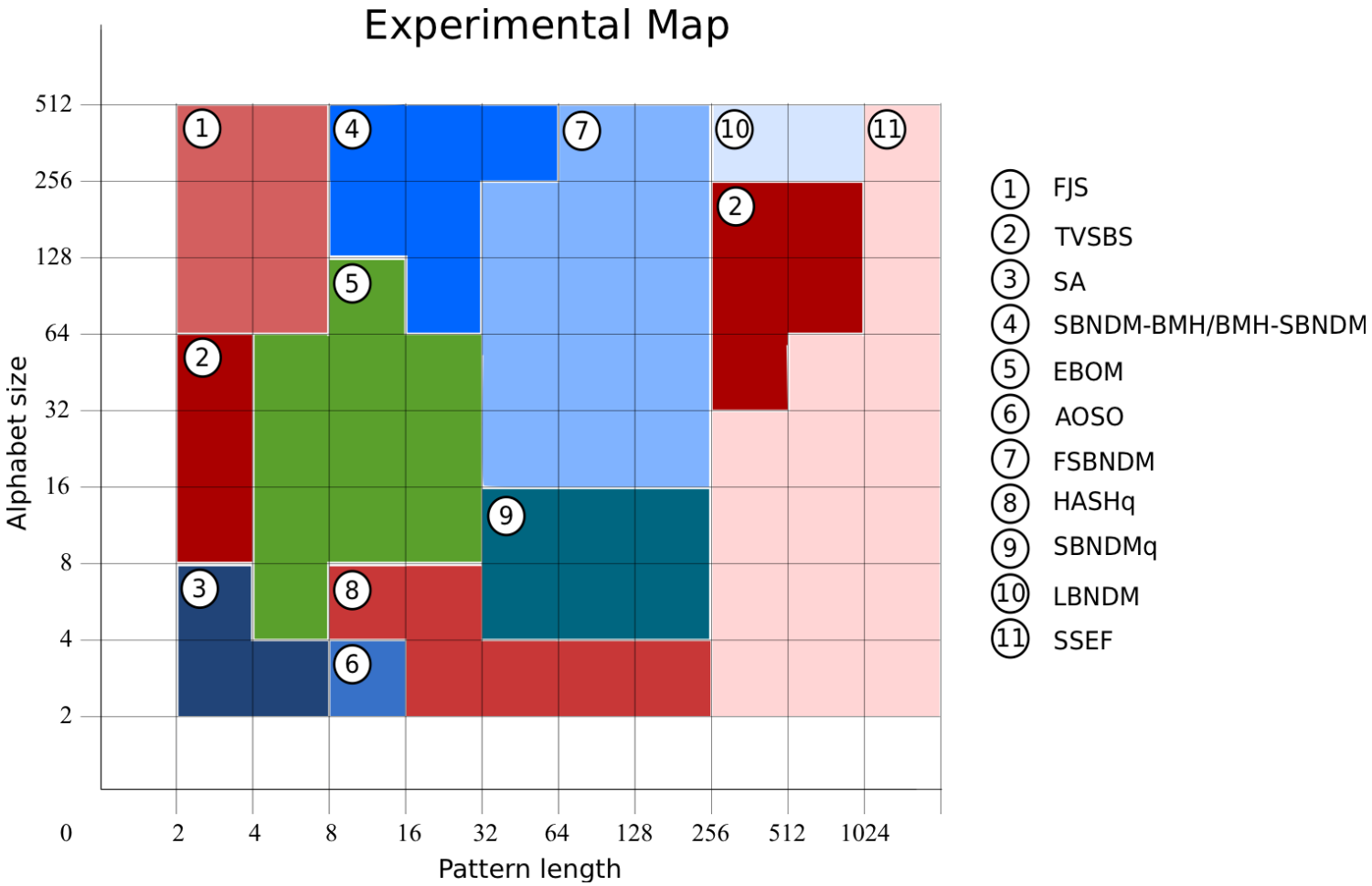
Exact String Matching

Solutions

Many!! see `http://monge.univ-mlv.fr/~lecroq/string`



Efficient solutions



S. Faro and T. Lecroq

The Exact Online String Matching Problem: a Review of the Most Recent Results

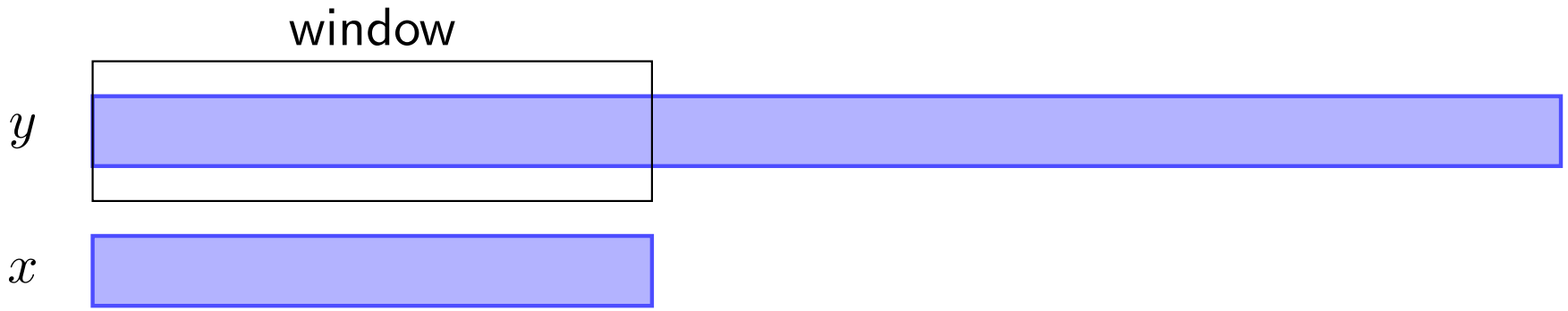
ACM Computing Surveys **45**(2) (2013) to appear.

Exact String Matching

Classical solutions

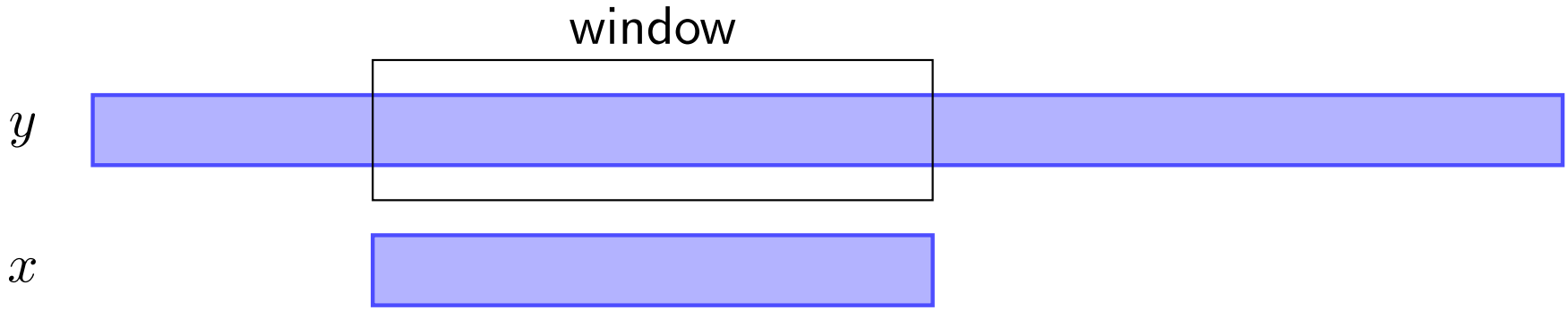
- comparisons
  - Knuth-Morris-Pratt (KMP)
  - Boyer-Moore (BM)
- automata
  - Backward DAWG Matching (with suffix automaton or oracle) (BDM)
- bit-parallelism
  - Shift Or (SO)
  - Backward Nondeterministic DAWG Matching (BNDM)
- filtering
  - Karp-Rabin (KR)

Sliding Window



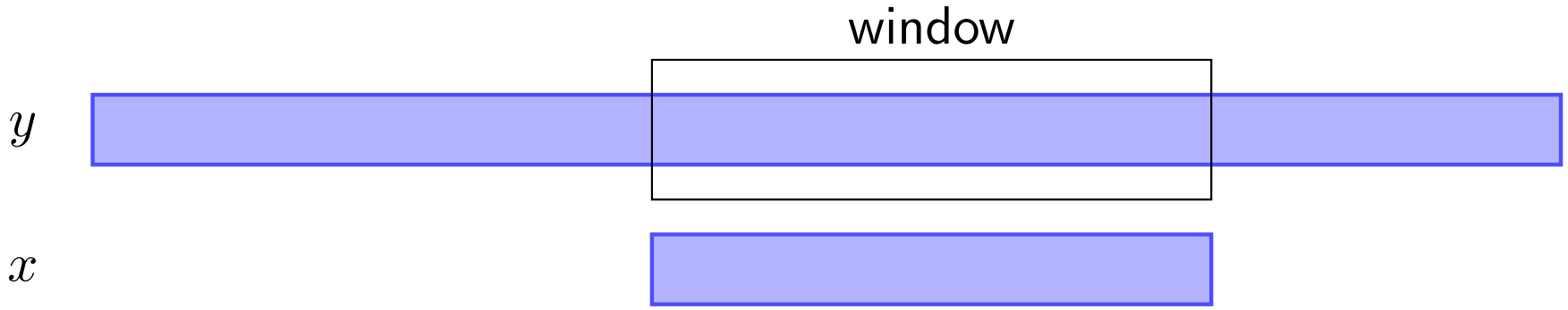
beginning

Sliding Window



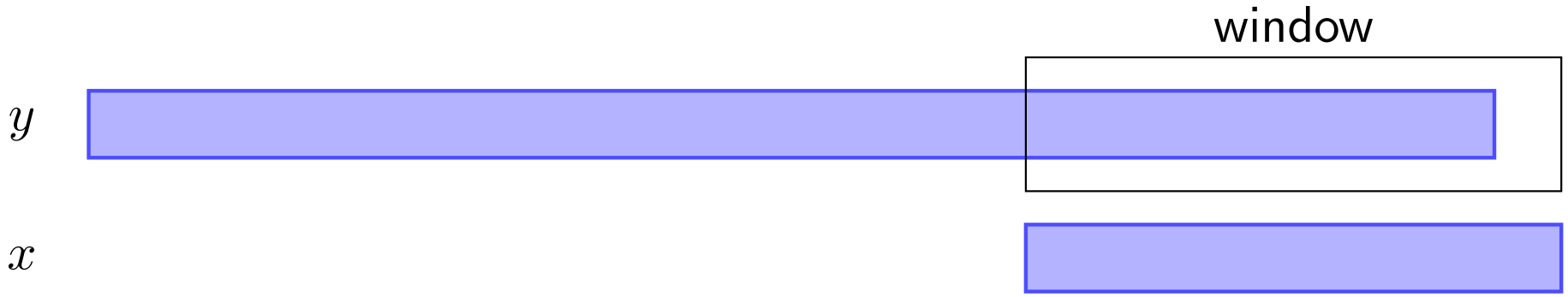
middle

Sliding Window



middle

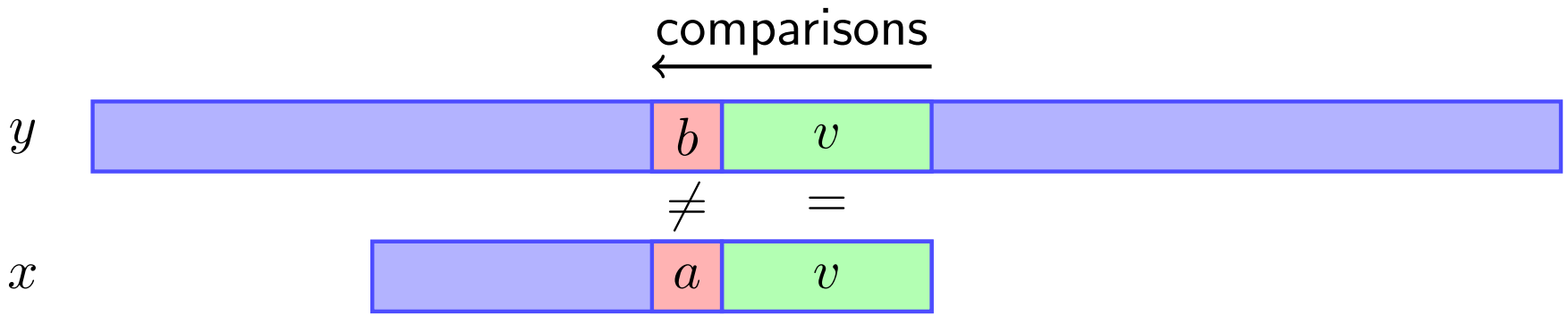
Sliding Window



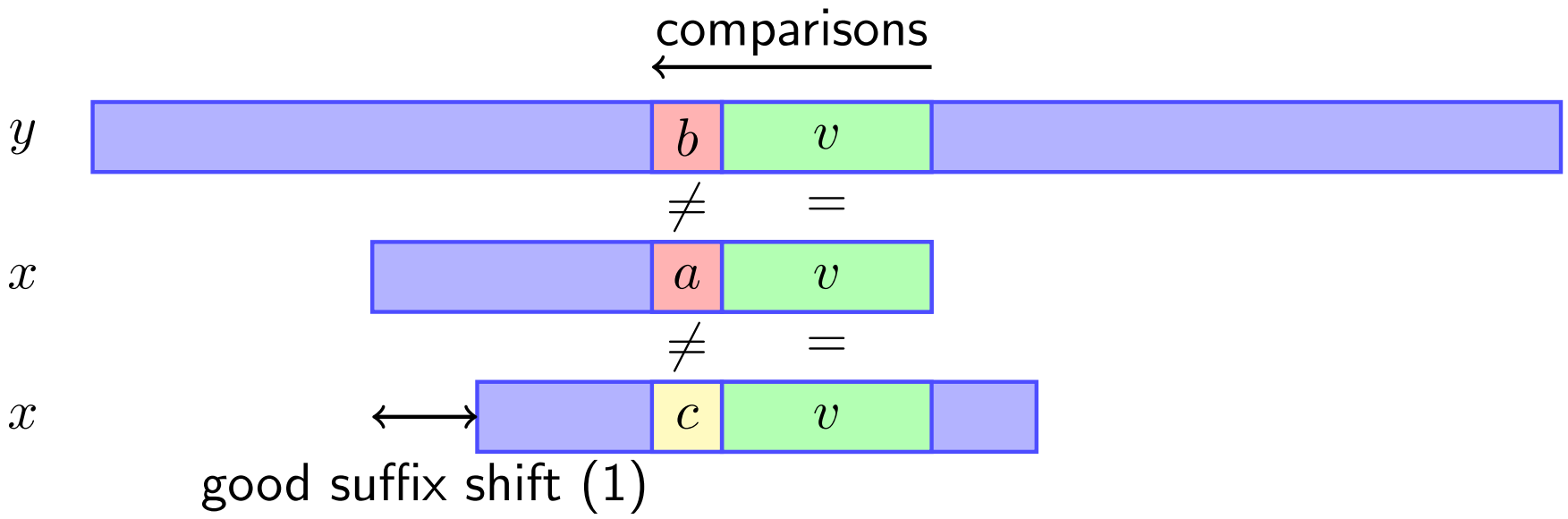
end

Boyer-Moore (1977)

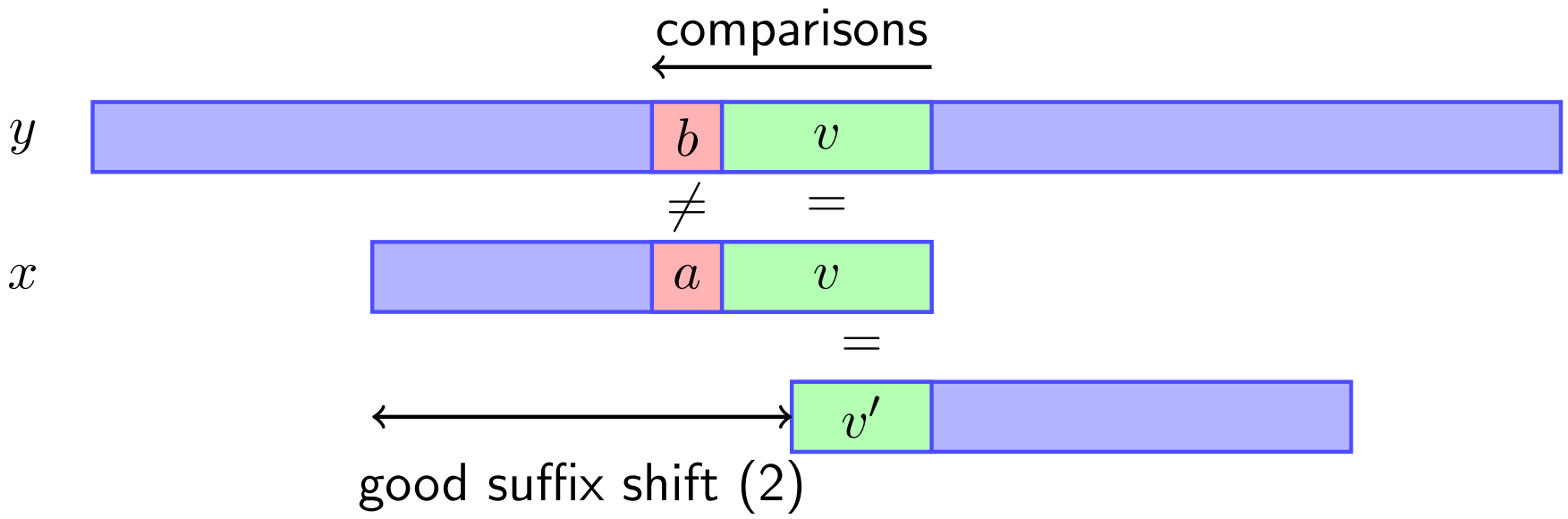
Boyer-Moore (1977)



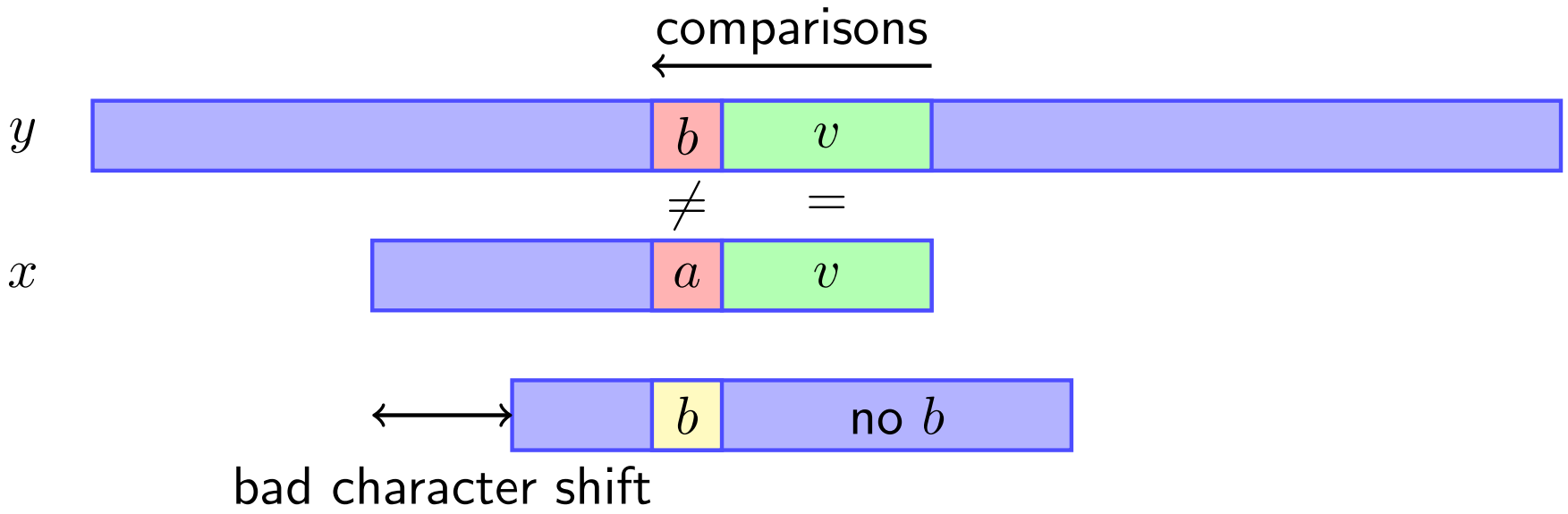
Boyer-Moore (1977)



Boyer-Moore (1977)



Boyer-Moore (1977)



Fast Search (Cantone-Faro, 2003)

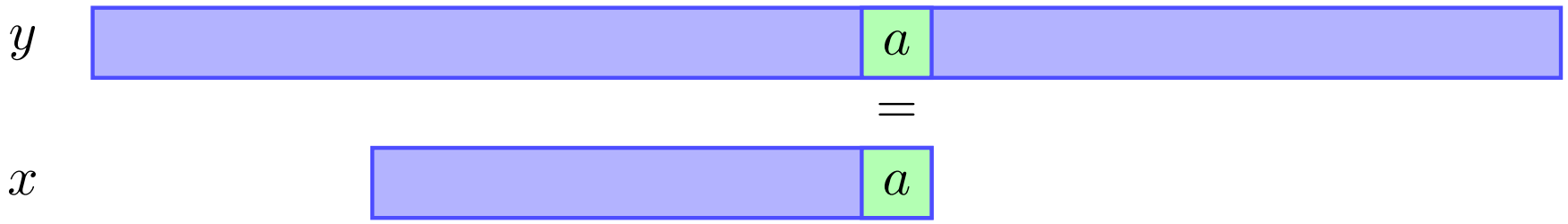
- uses the bad character shift when a mismatch occurs with the pattern righmost character
- uses the good suffix shift otherwise

TVSBS (Thathoo-Virmani-Lakshmi-Balakrishnan-Sekar, 2006)

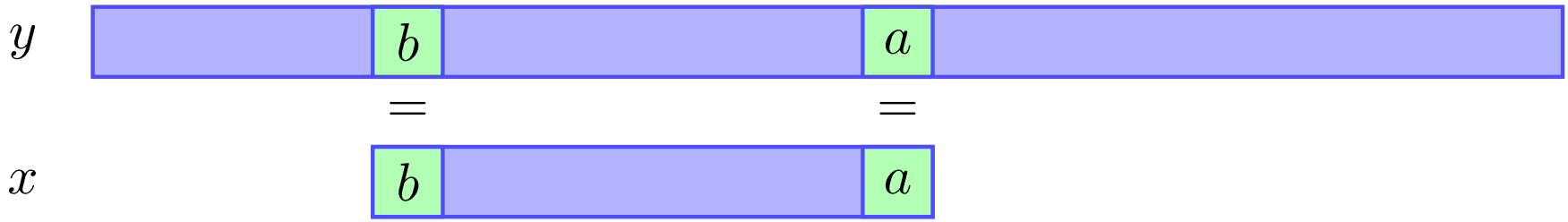
- compares first the rightmost character of the window then the leftmost, then all the others
- uses both the righmost character of the window and the following to perform the shift

TVSBS (Thathoo-Virmani-Lakshmi-Balakrishnan-Sekar, 2006)

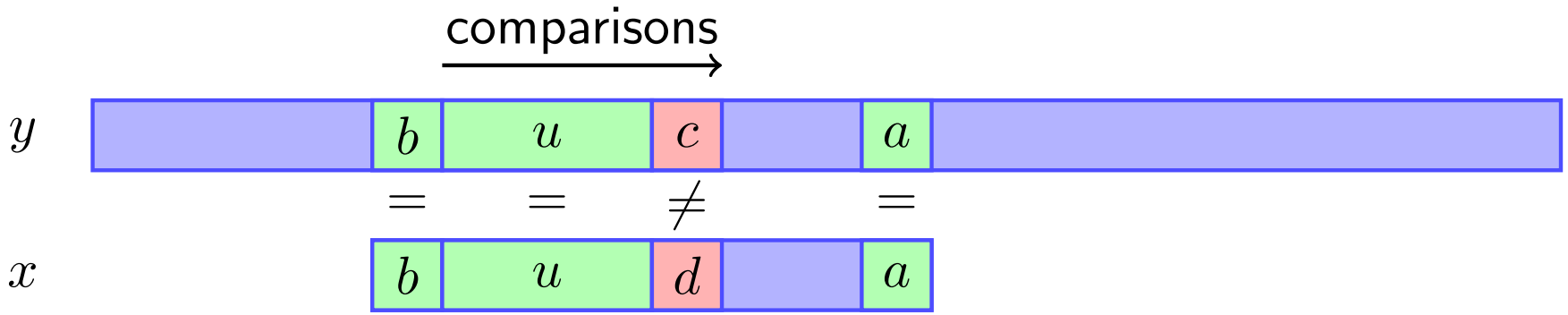
TVSBS (Thathoo-Virmani-Lakshmi-Balakrishnan-Sekar, 2006)



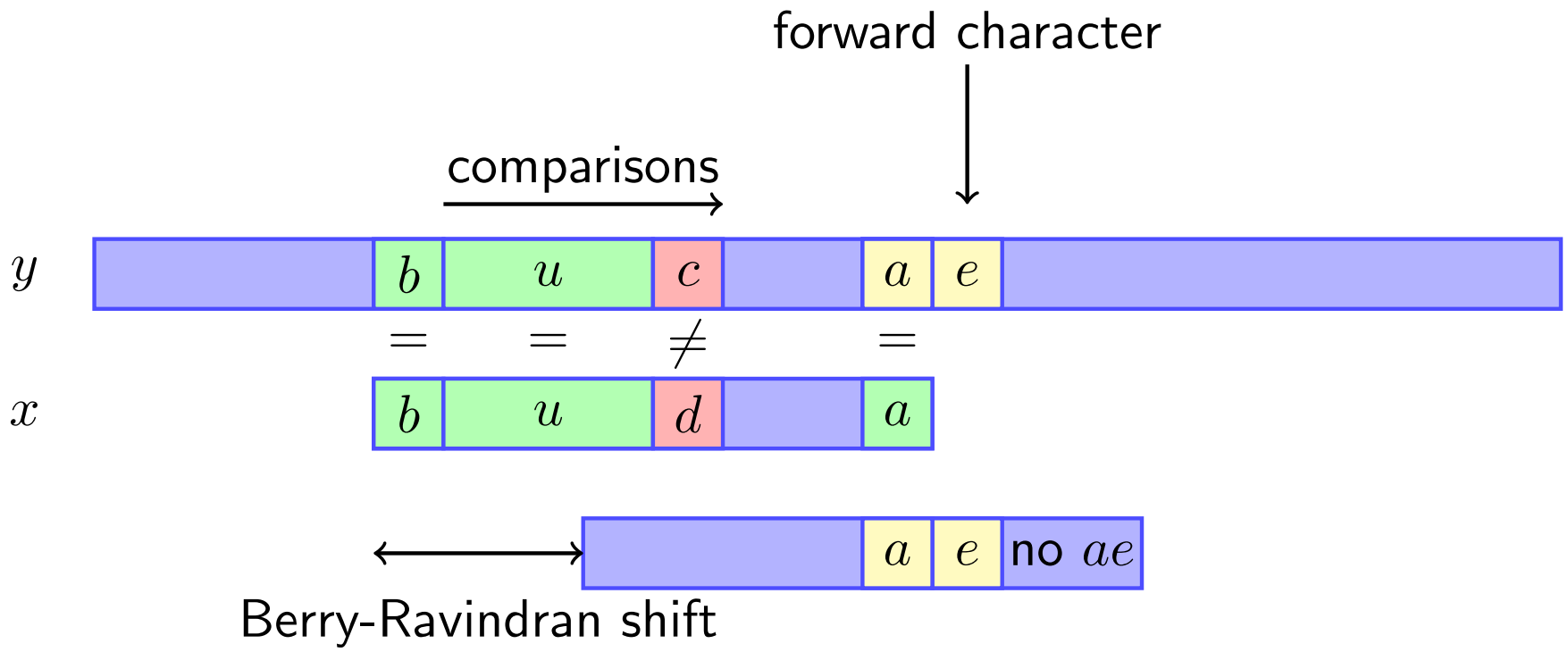
TVSBS (Thathoo-Virmani-Lakshmi-Balakrishnan-Sekar, 2006)



TVSBS (Thathoo-Virmani-Lakshmi-Balakrishnan-Sekar, 2006)



TVSBS (Thathoo-Virmani-Lakshmi-Balakrishnan-Sekar, 2006)



BNDM (Navarro & Raffinot, 1998)

x C A T A

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y C C A T A C

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	1	1	1	1		

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	1	1	1	1		

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	1	1	1	1		
	0	0	1	0		

And with S_T

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		1	1	1	1		
S_C	1	0	0	0		0	0	1	0		
S_G	0	0	0	0		0	0	1	0		
S_T	0	0	1	0							

And with S_T

$=$

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		1	1	1	1		
S_C	1	0	0	0		0	0	1	0		And with S_T
S_G	0	0	0	0		0	0	1	0		$=$
S_T	0	0	1	0		0	1	0	0		Shift

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	0	1	0	0		

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	0	1	0	0		
	0	1	0	1		

And with S_A

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	0	1	0	0		
	0	1	0	1		
	0	1	0	0		

And with S_A

$=$

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		0	1	0	0		
S_C	1	0	0	0		0	1	0	1		And with S_A
S_G	0	0	0	0		0	1	0	0		$=$
S_T	0	0	1	0		1	0	0	0		Shift

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	1	0	0	0		

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		1	0	0	0		
S_C	1	0	0	0		1	0	0	0		
S_G	0	0	0	0							
S_T	0	0	1	0							

And with S_C

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		1	0	0	0		
S_C	1	0	0	0		1	0	0	0		And with S_C
S_G	0	0	0	0		1	0	0	0		$=$
S_T	0	0	1	0							

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		1	0	0	0		
S_C	1	0	0	0		1	0	0	0		And with S_C
S_G	0	0	0	0		1	0	0	0		$=$
S_T	0	0	1	0		0	0	0	0		Shift

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A
S_A	0	1	0	1
S_C	1	0	0	0
S_G	0	0	0	0
S_T	0	0	1	0

y	C	C	A	T	A	C
	0	0	0	0		

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		0	0	0	0		
S_C	1	0	0	0		1	0	0	0		
S_G	0	0	0	0							
S_T	0	0	1	0							

And with S_C

BNDM (Navarro & Raffinot, 1998)

x	C	A	T	A	y	C	C	A	T	A	C
S_A	0	1	0	1		0	0	0	0		
S_C	1	0	0	0		1	0	0	0		
S_G	0	0	0	0		0	0	0	0		
S_T	0	0	1	0							

And with S_C

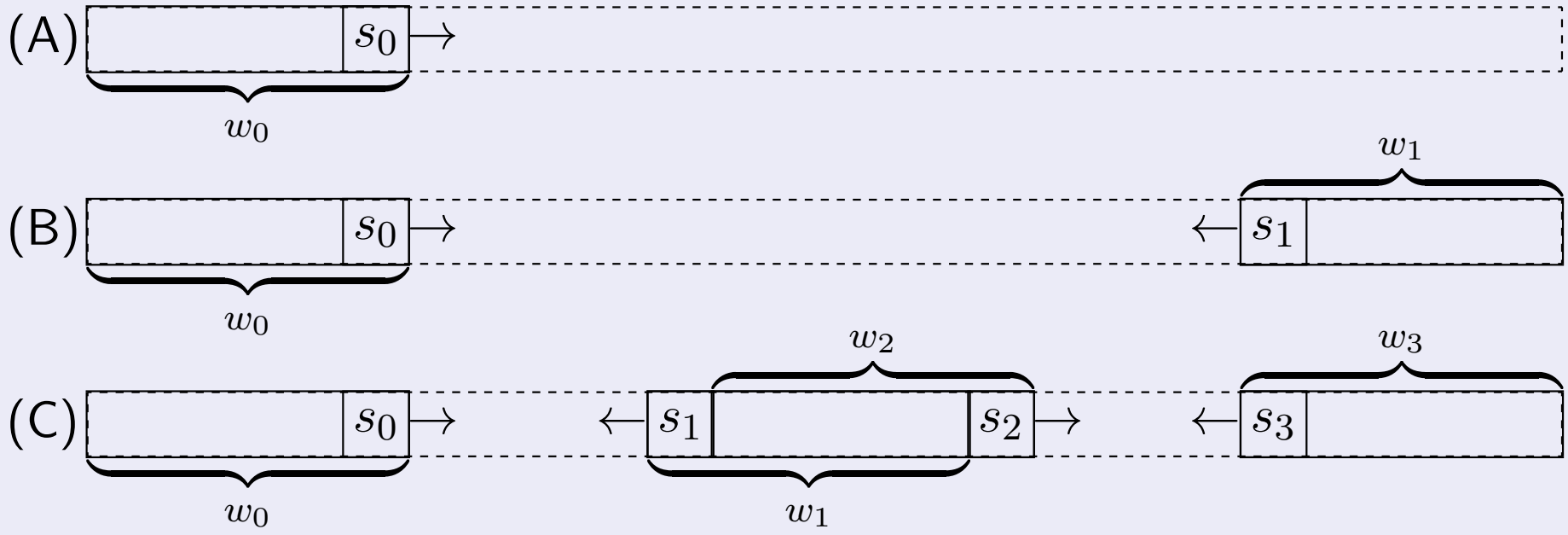
$=$

Outline

1. Introduction to exact string matching
2. Multiple windows
3. Substring with no repetition
4. Multiple hashing

Partition the text in $k/2$ substrings and use k windows

$m < n/k$ and k is even



A general scheme for the multiple sliding windows matcher with:

(A) 1 window

(B) 2 windows

(C) 4 windows

A General Multiple Sliding Windows Approach

Then process simultaneously the k different text windows

$$w_0, w_1, \ldots, w_{k-1}$$

where

$$w_{2i} = y[(2n/k)i..(2n/k)i + m - 1]$$

left windows, goes to right

and

$$w_{2i+1} = y[(2n/k)(i+1) - 1..(2n/k)(i+1) + m - 2]$$

right windows, goes to left

for $i = 0, \ldots, (k-2)/2$

Ending situation

For each couple of windows (w_{2i}, w_{2i+1})
the sliding process ends
when the window w_{2i} slides over the window w_{2i+1}

no occurrence can be missed
due to the $m - 1$ overlapping characters between adjacent substrings

SMART: String Matching Algorithm Research Tool

http://www.dmi.unict.it/~faro/smart/

SMART. String Matching Algorithms Research Tool

http://www.dmi.unict.it/~faro/smart/

SMART | DOWNLOAD | DOCUMENTATION | POINTERS

smart string matching research tool

by simone faro and thierry lecroq

1 TOOL

40 YEARS

85 ALGOS

12 TEXTS

smart (string matching algorithms research tool) is a tool which provides a standard framework for researchers in string matching. It helps users to test, design, evaluate and understand existing solutions for the exact string matching problem. Moreover it provides the implementation of (almost) all string matching algorithms and a wide corpus of text buffers.

In the last **40 years of research** in computer science string matching was one of the most extensively studied problem, mainly due to its direct applications to such diverse areas as text, image and signal processing, speech analysis and recognition, data compression, information retrieval, computational biology and chemistry. Moreover String matching algorithms are also basic components used in implementations of practical softwares existing under most operating systems.

Since 1970 **more than 80 string matching algorithms** have been proposed, and more than 50% of them in the last ten years. The smart tool provides a comprehensive collection of all string matching algorithms, inplemented in C programming language, and helps researcher to perform experimental results and compare them from a practical point of view. Smart provides a practical and standard platform for testing string matching algorithms and sharing results with the community.

The smart tool provides also **a corpus of 12 texts** on which the string matching algorithms can be tested. Texts in the corpus are of different types, including natural language texts, genome sequences, protein sequences, and random texts with a uniform distribution of characters.

the smart tool

- smart documentation
- implemented algorithms
- data resources
- experimental results
- string matching authors
- string matching bibliography

fast download

- Linux binaries
- Mac OS X binaries
- Source codes
- Smart corpus

SMART: String Matching Algorithm Research Tool

- more than 80 string matching algorithms
- a corpus of 12 texts
- select/deselect string matching algorithms
- output experimental results in LaTeX, xml, html and txt formats
- easy to plug new algorithms

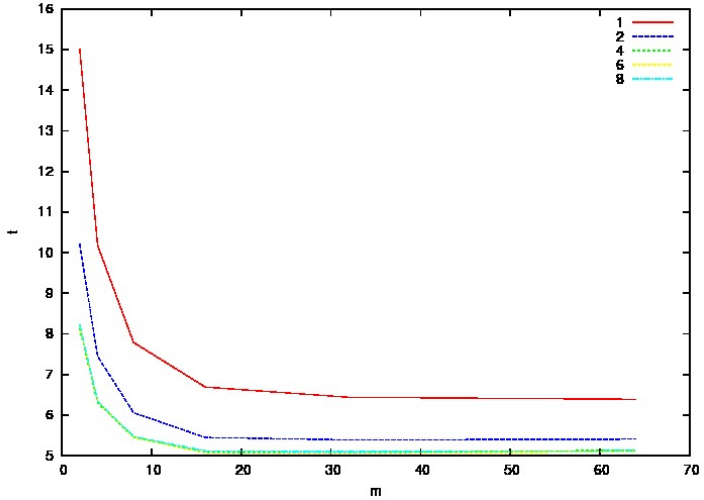
Experiments

Random texts of length $n = 4,000,000$ on alphabets of size 16, 32 and 64 for $m = 2, 4, 8, 16, 32, 64$ with algorithms:

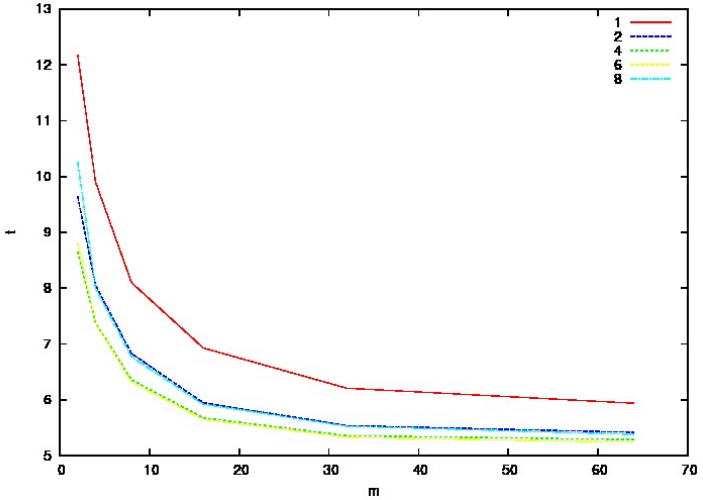
- Fast Search
- TVSBS
- SBNDM
- FSBNDM

and $k = 1, 2, 4, 6, 8$ windows

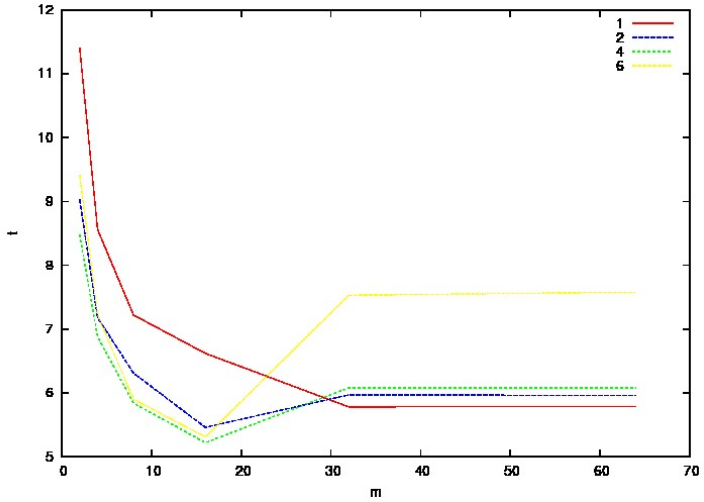
Experimental results



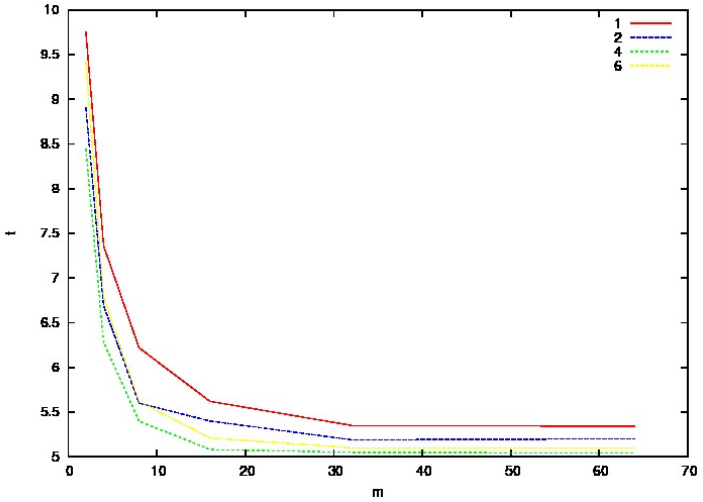
Fast Search



TVSBS



SBNDM



FSBNDM

Experiments

Random texts of length $n = 4,000,000$ on alphabets of size 16, 32 and 64 for $m = 2, 4, 8, 16, 32, 64, 128, 256, 512$ and with a protein and a natural language (English) files with algorithms:

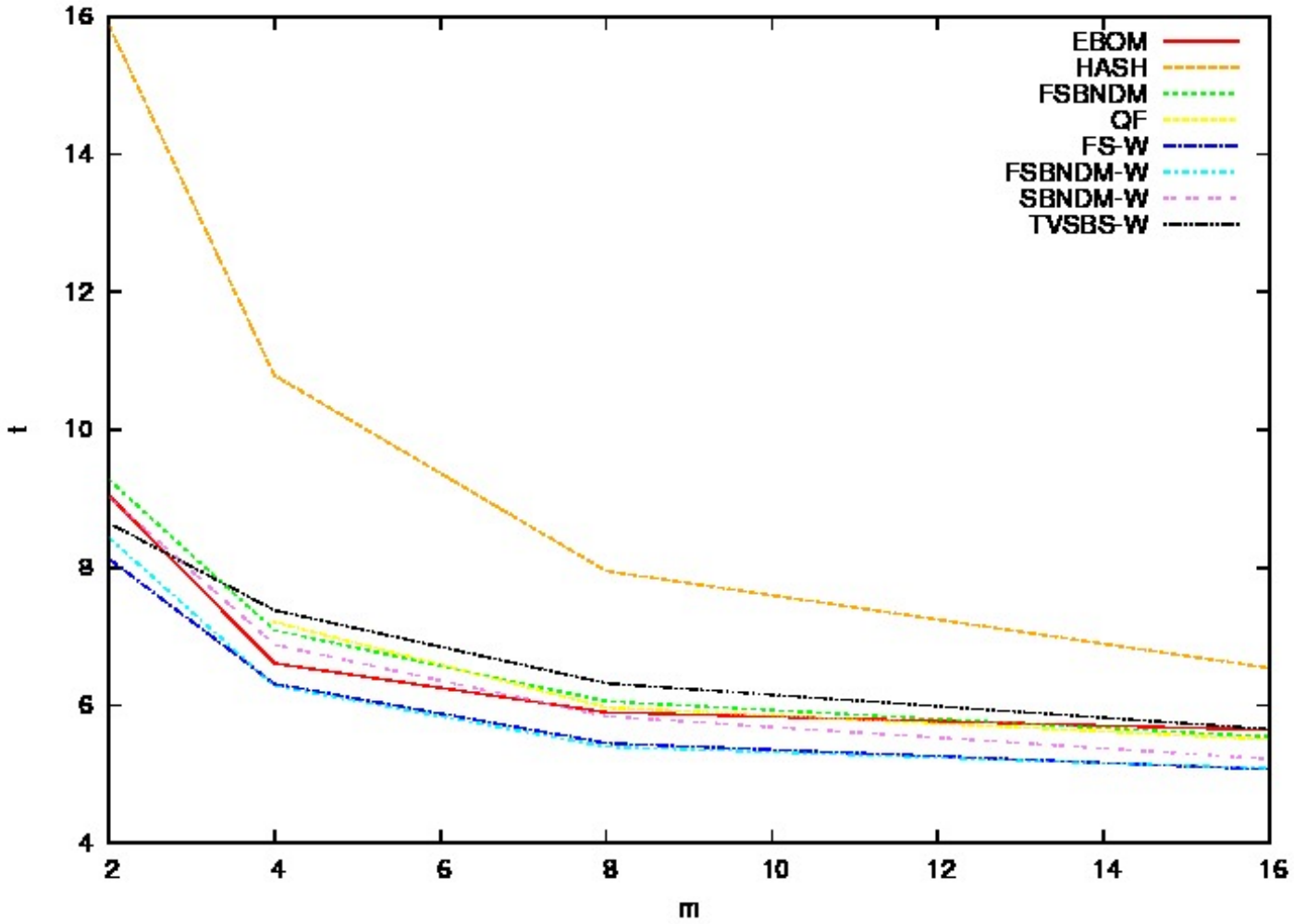
known

- EBOM
- HASH
- FSBNDM
- QF

new

- FS-W
- FSBNDM-W
- SBNDM-W
- TVSBS-W

Experimental results



Alphabet size 32, short patterns

Perspectives

- implement other known exact string matching algorithms in this multiple window framework
- truly parallelize
- apply it to other related problems

Reference

S. Faro and T. Lecroq
A Multiple Sliding Windows Approach to Speed Up String Matching Algorithms
In: (R. Klasing editor, Proceedings of the 11th International Symposium on Experimental Algorithms (SEA 2012), Bordeaux, France, 2012), Lecture Notes in Computer Science 7276, Springer-Verlag, Berlin, 172-183.

Outline

1. Introduction to exact string matching
2. Multiple windows
3. **Substring with no repetition**
4. Multiple hashing

Problem with bit-parallel algorithms

Solutions

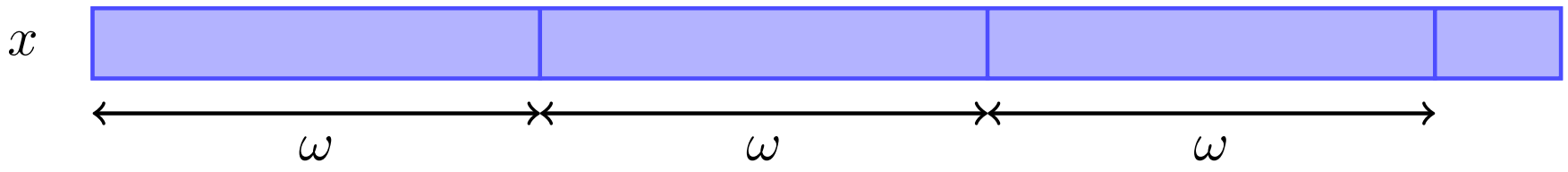
- LBNDM (Peltola & Tarhio, 2003)
- BXS (Durian, Peltola, Salmela & Tarhio, 2010)
- Factorized BNDM (Cantone, Faro & Giaquinta, 2010)

LBNDM (Long patterns BNDM, Peltola & Tarhio, 2003)

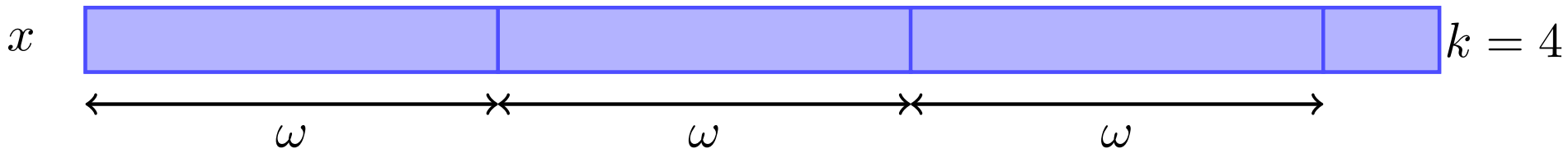
- partition $x = x^0 x^1 \cdots x^{\lfloor m/k \rfloor - 1}$ where $|x^j| = k = \lfloor (m-1)/\omega \rfloor + 1$
- construct a new pattern x' where $x'[j]$ is the set of characters of x^j
- search for x' in y using the BNDM algorithm by considering every k-th character

LBNDM (Long patterns BNDM, Peltola & Tarhio, 2003)

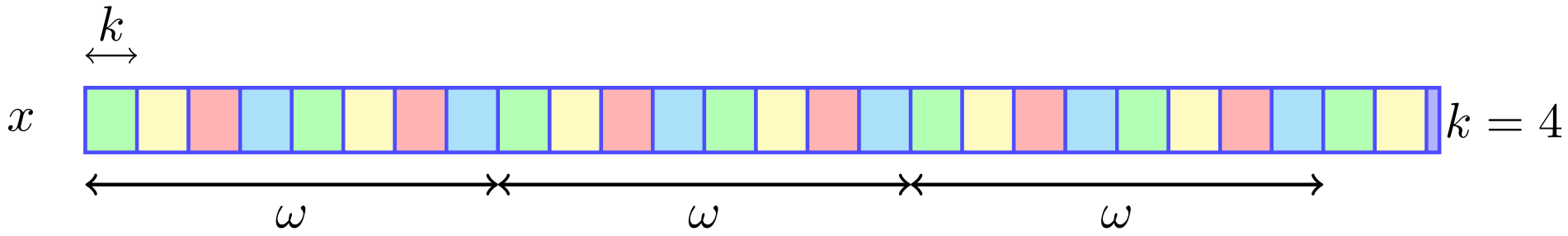
LBNDM (Long patterns BNDM, Peltola & Tarhio, 2003)



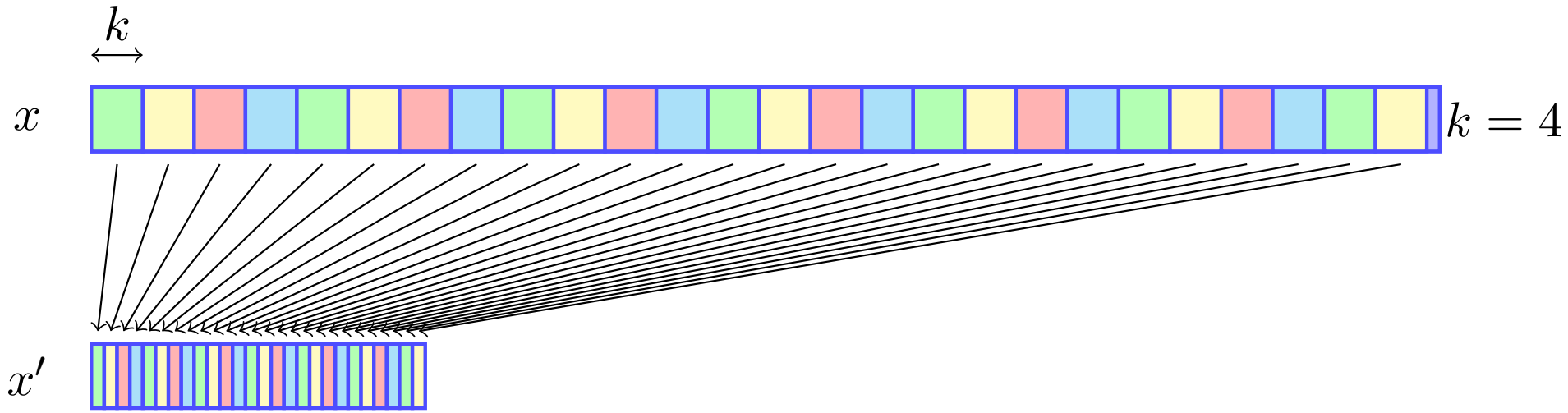
LBNDM (Long patterns BNDM, Peltola & Tarhio, 2003)



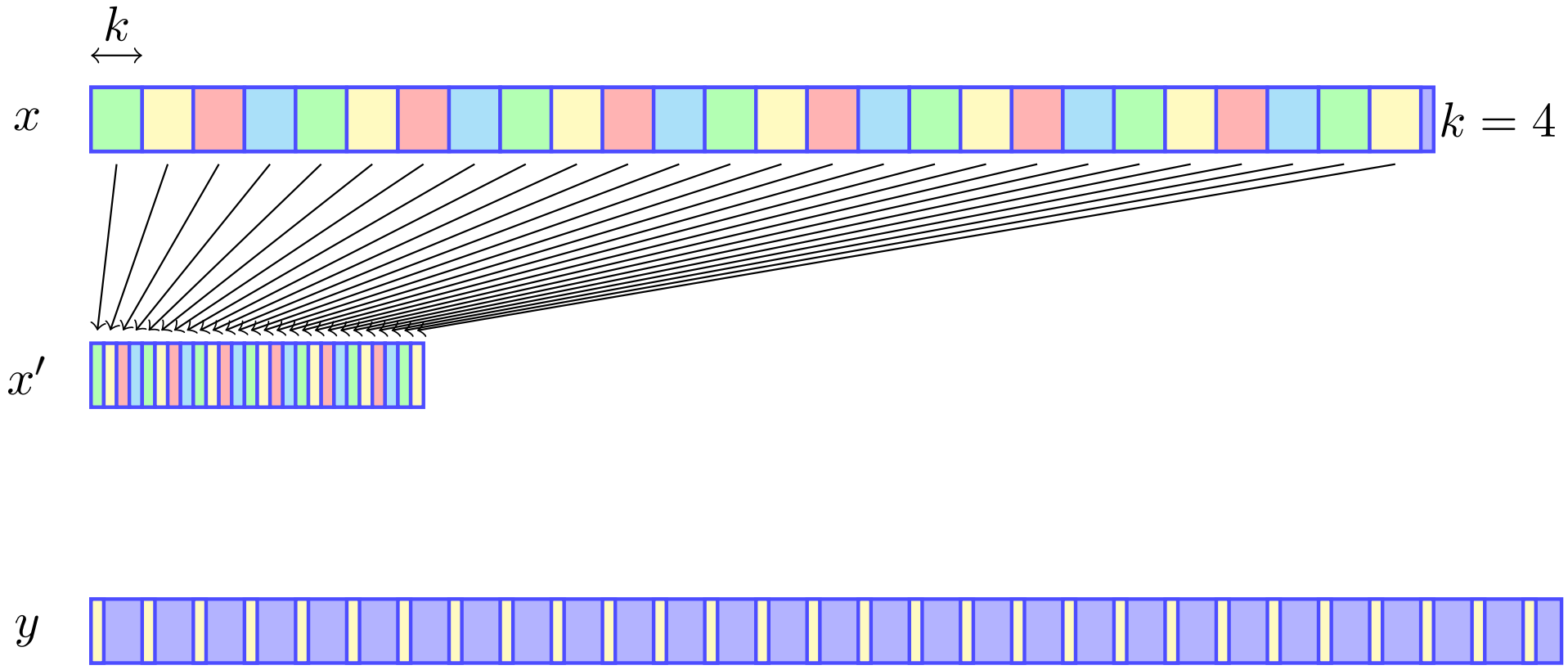
LBNDM (Long patterns BNDM, Peltola & Tarhio, 2003)



LBNDM (Long patterns BNDM, Peltola & Tarhio, 2003)



LBNDM (Long patterns BNDM, Peltola & Tarhio, 2003)



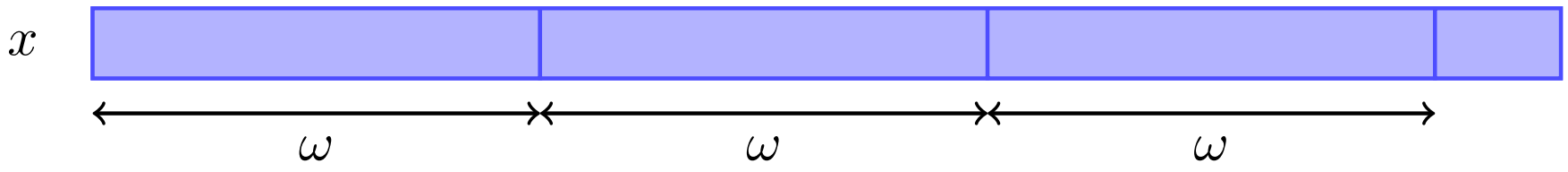
BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)

- partition $x = x^0 x^1 \cdots x^{\lfloor m/\omega \rfloor}$ where $|x^j| = \omega$
- construct a new pattern x' where $x'[j]$ is the set of characters of $x[j + \omega]$
- search for x' in y using the BNDM algorithm by using a "circular" automaton

BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)

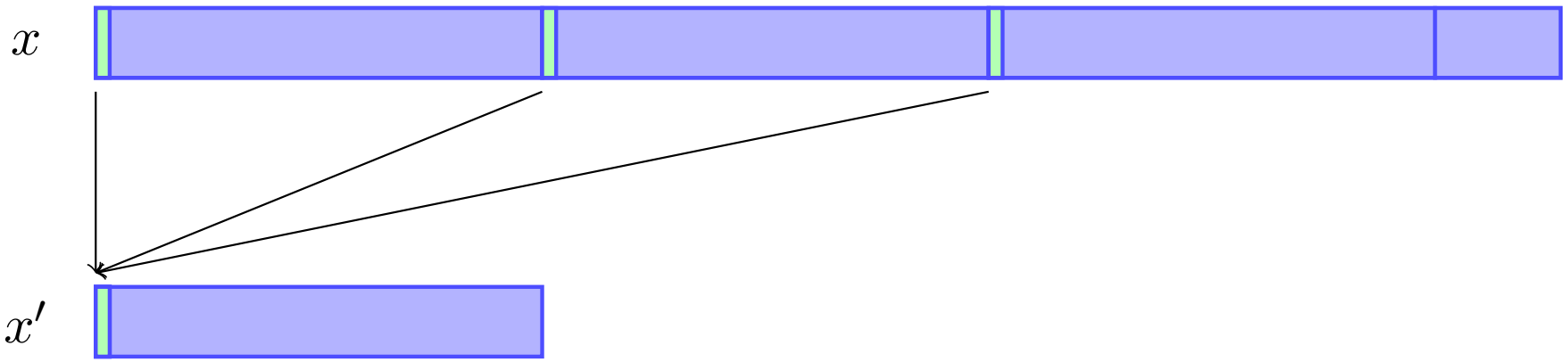
x

BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)

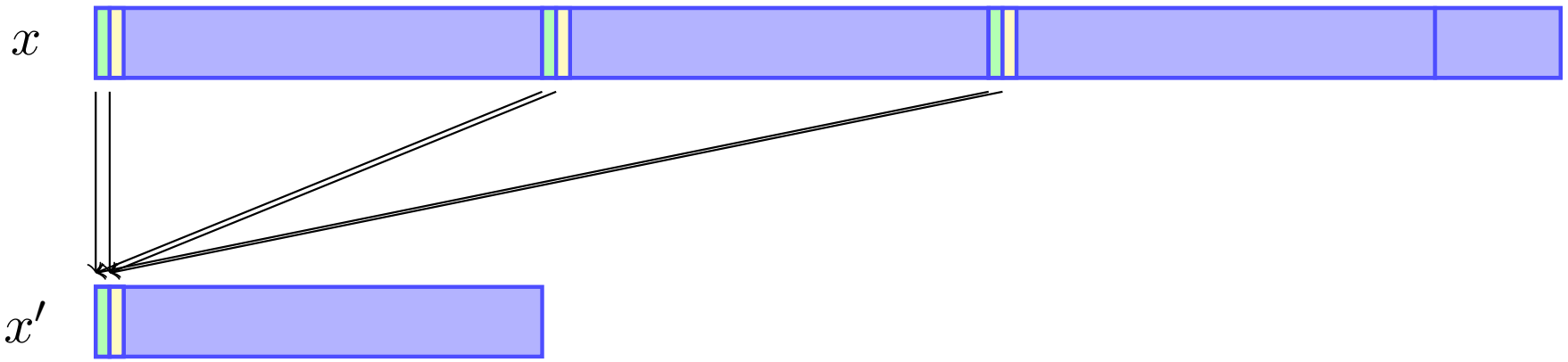


BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)

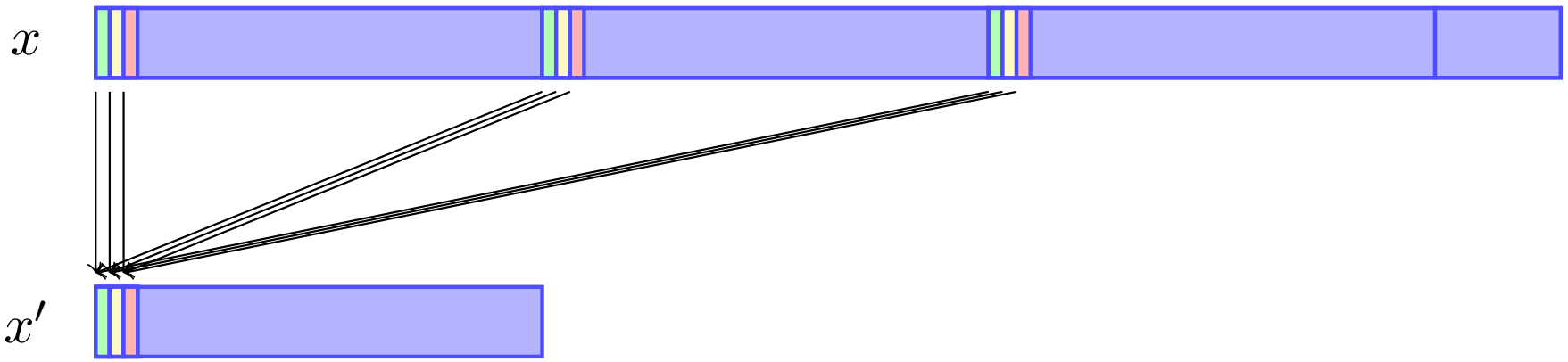
BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)



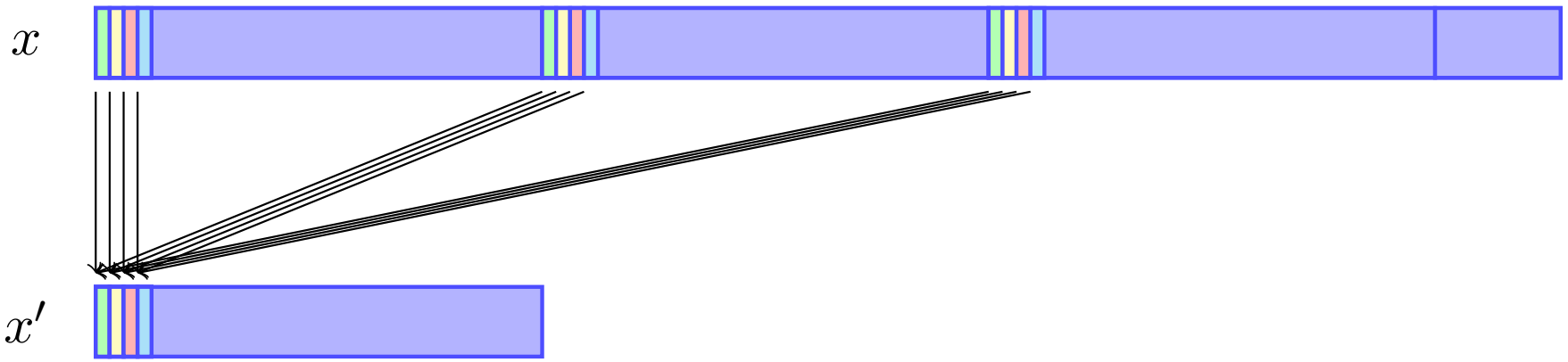
BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)



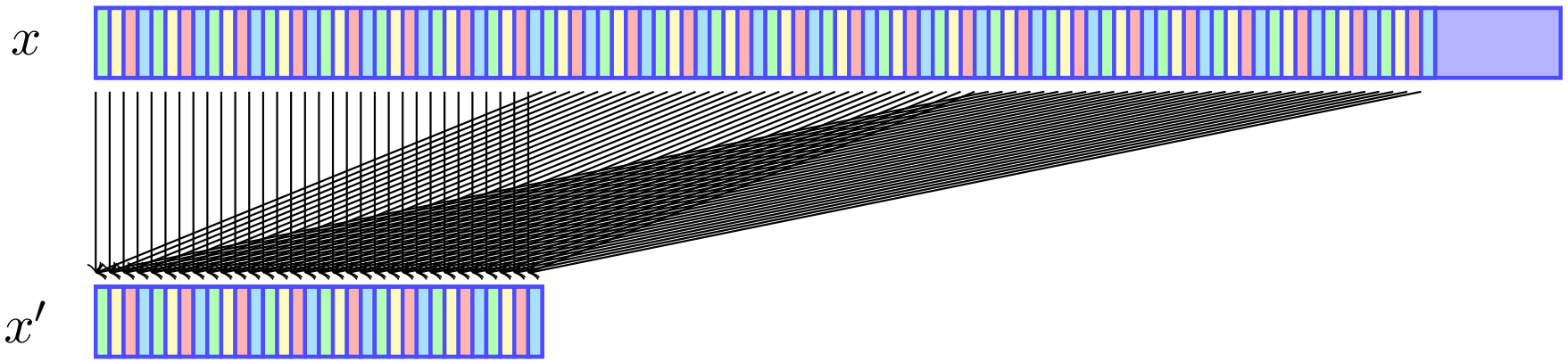
BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)



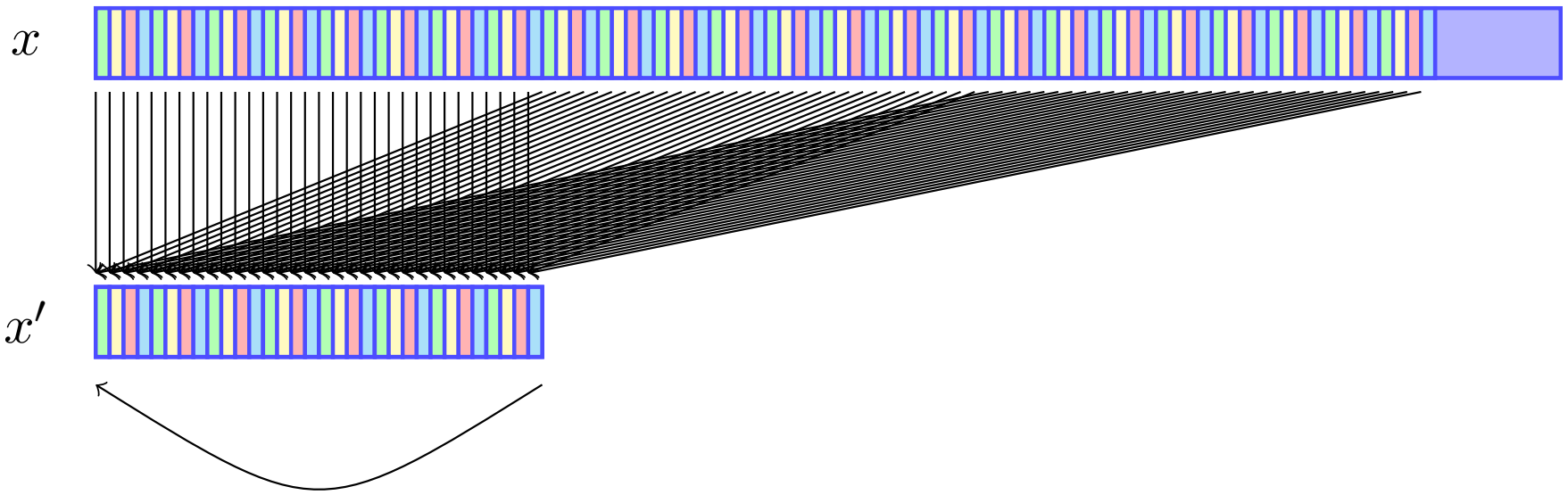
BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)



BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)



BXS (BNDM algorithm with Extended Shift, Durian, Peltola, Salmela & Tarhio, 2010)



Substring with no repetition

Definition

A substring s of x is a substring with no repetition (SNR) if any character $c \in \Sigma$ appears at most once in s.

SNR

SNR

automaton

Maximal SNR

automaton

SNR

SNR

automaton

Maximal SNR

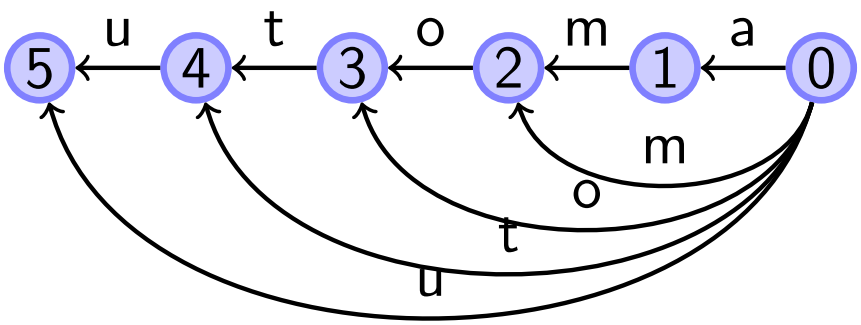
automaton

A New Fast Suffix Automaton Based Algorithm

Properties

- $|s| \leq \min\{\Sigma, m\}$
- an SNR admits a suffix automaton where there does not exist two states having incoming transitions labeled with the same symbol

SNR Suffix Automaton



The Preprocessing Phase

Find a maximal SNR of the pattern, i.e. an SNR of maximal length.
$O(n)$ time

Length of the maximal SNR

In many practical cases the length of the maximal SNR is not large enough if compared with the pattern length.

Average length of the maximal SNR

m	2	4	8	16	32	64	128	256	512	1024
genome	1.72	2.62	3.20	3.64	3.89	3.99	4.00	4.00	4.00	4.00
protein	1.91	3.46	5.43	6.98	8.20	9.27	10.08	10.95	11.70	12.27
English	1.99	3.81	6.25	7.83	8.96	9.83	10.46	11.07	11.51	12.18

Longer SNRs

In order to allow longer SNR it is convenient to use a condensed alphabet whose symbols are obtained by combining groups of q symbols.
A hash function $h : \Sigma^q \leftarrow \{0, \ldots, MAX - 1\}$ can be used for combining the group of symbols, for a fixed constant value MAX.
Let $[u]^q = h(u)$, for each $u \in \Sigma^q$.

Condensed alphabet

$$
\begin{array}{lcl}
x & : & \textbf{agcctgatcg} \\
x_2 & : & [\textbf{ag}]^2[\textbf{gc}]^2[\textbf{cc}]^2[\textbf{ct}]^2[\textbf{tg}]^2[\textbf{ga}]^2[\textbf{at}]^2[\textbf{tc}]^2[\textbf{cg}]^2 \\
x_3 & : & [\textbf{agc}]^3[\textbf{gcc}]^3[\textbf{cct}]^3[\textbf{ctg}]^3[\textbf{tga}]^3[\textbf{gat}]^3[\textbf{atc}]^3[\textbf{tcg}]^3 \\
x_4 & : & [\textbf{agcc}]^4[\textbf{gcct}]^4[\textbf{cctg}]^4[\textbf{ctga}]^4[\textbf{tgat}]^4[\textbf{gatc}]^4[\textbf{atcg}]^4
\end{array}
$$

It turns out that the length of the maximal SNR, though quite less than m in most cases, is quite larger than the size of a computer word (which typically is 32 or 64). This leads to larger shift in a suffix automata based algorithm.

Length of the maximal SNR using condensed characters

The average length of the maximal SNR in patterns randomly extracted from a genome sequence. The SNR have been computed using condensed alphabets on q characters, where q ranges from 1 to 8.

q/m	2	4	8	16	32	64	128	256	512	1024
1	1.72	2.62	3.20	3.64	3.89	3.99	4.00	4.00	4.00	4.00
2	1.00	2.86	5.45	7.61	9.19	10.41	11.23	11.97	12.64	13.09
4	–	1.00	4.94	12.33	22.91	32.75	39.89	45.12	50.84	54.29
6	–	–	3.00	10.81	24.56	42.19	55.69	66.31	74.35	82.48
8	–	–	1.00	8.98	24.50	51.55	88.03	116.3	140.8	163.2

Experimental results on a genome sequence

m	2	4	8	16	32	64	128	256	512	1024	2048	4096
BDM	21.04	14.66	9.90	7.40	5.94	5.15	4.79	4.68	4.90	5.42	7.22	10.59
BNDM	19.52	12.56	8.92	6.72	5.50	5.55	5.51	5.47	5.58	5.49	5.50	5.49
SBNDM	12.25	9.13	7.66	6.27	5.14	5.14	5.12	5.13	5.12	5.13	5.13	5.14
BXS	19.57	13.88	9.27	6.88	5.47	5.15	4.99	5.52	523.2	-	-	-
F-BNDM	15.49	10.74	8.71	7.09	5.78	5.10	5.03	5.03	5.02	5.03	5.05	5.05
LBNDM	27.62	15.24	9.79	7.28	5.80	5.38	5.36	8.45	26.93	25.28	22.50	20.67
BSDM	20.94	17.49	14.42	13.05	11.99	11.52	11.55	11.39	11.42	11.38	11.46	11.50
$BSDM_2$	11.43	9.26	8.66	8.31	7.82	7.44	7.15	6.89	6.60	6.51	6.37	6.24
$BSDM_3$	-	7.44	5.92	5.57	5.43	5.38	5.33	5.31	5.28	5.30	5.28	5.28
$BSDM_4$	-	9.67	5.61	4.99	4.79	4.73	4.66	4.64	4.63	4.63	4.66	4.66
$BSDM_5$	-	-	5.99	5.00	4.77	4.66	4.61	4.58	4.57	4.56	4.58	4.58
$BSDM_6$	-	-	6.86	5.09	4.69	4.58	4.53	4.49	4.50	4.47	4.50	4.50
$BSDM_7$	-	-	8.81	5.25	4.71	4.55	4.51	4.47	4.45	4.47	4.47	4.49
$BSDM_8$	-	-	14.88	5.57	4.80	4.56	4.51	4.50	4.48	4.48	4.49	4.50

S. Faro and T. Lecroq
A Fast Suffix Automata Based Algorithm for Exact Online String Matching
In: (N. Moreira and R. Reis editors, Proceedings of the 17th International Conference on Implementation and Application of Automata (CIAA 2012), Porto, Portugal, 2012), Lecture Notes in Computer Science 7381, Springer-Verlag, Berlin, 149-158.

Outline

1. Introduction to exact string matching
2. Multiple windows
3. Substring with no repetition
4. **Multiple hashing**

Multiple exact string matching

Problem

Find all the occurrences of the patterns in $X = \{x_0, \ldots, x_{k-1}\}$ in y.
$m = \min\{|x_i|\}$

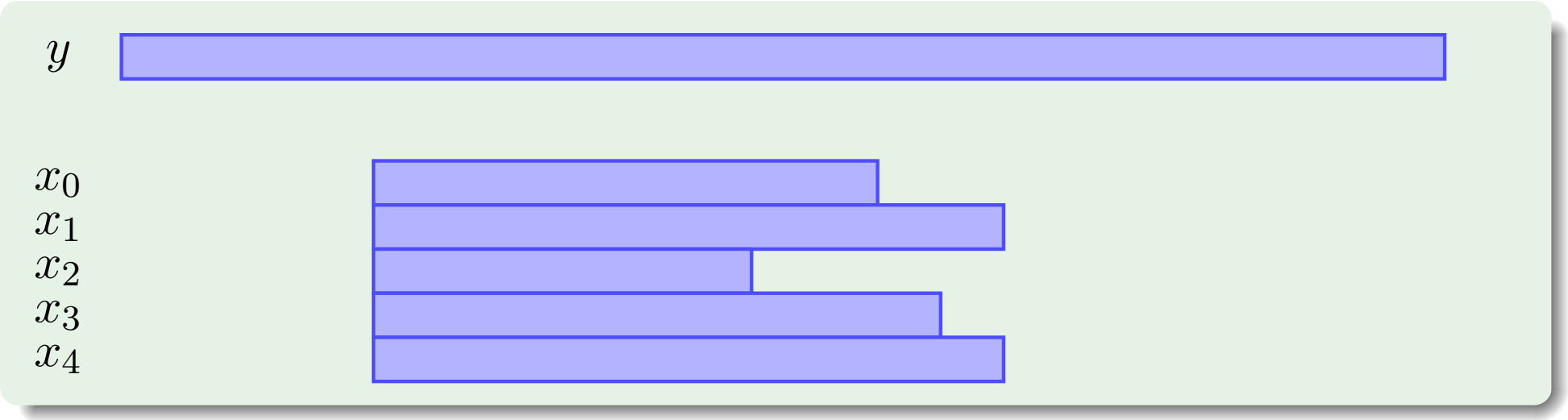
Classical solutions

- Aho & Corasick (1975)
- Commentz-Walter (1979)
- Wu & Manber (1994)
- Multi-Reverse-Factor (1999)
- MBNDM

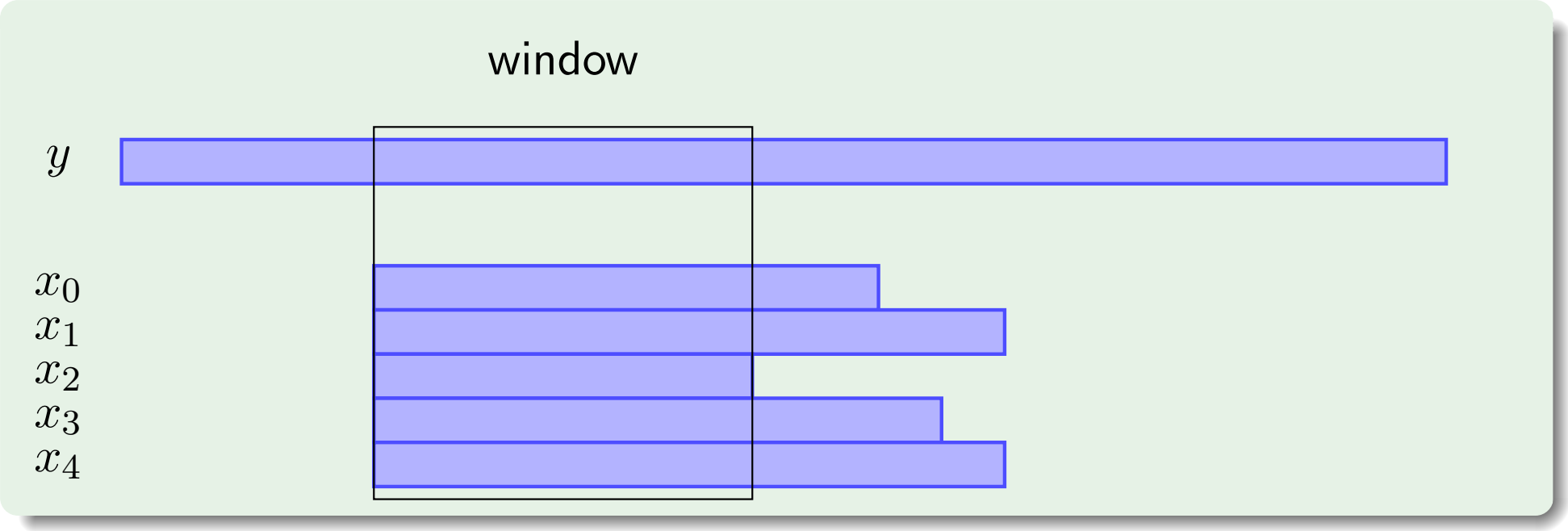
Wu-Manber algorithm

- considers blocks of length ℓ
- blocks are hashed using a function h into values less than $maxvalue$
- $shift[h(x_i[j-\ell+1..j])] = \min\{|x_i|, m-\ell+1\}$
- prefixes are also hashed
- implemented in `agrep`

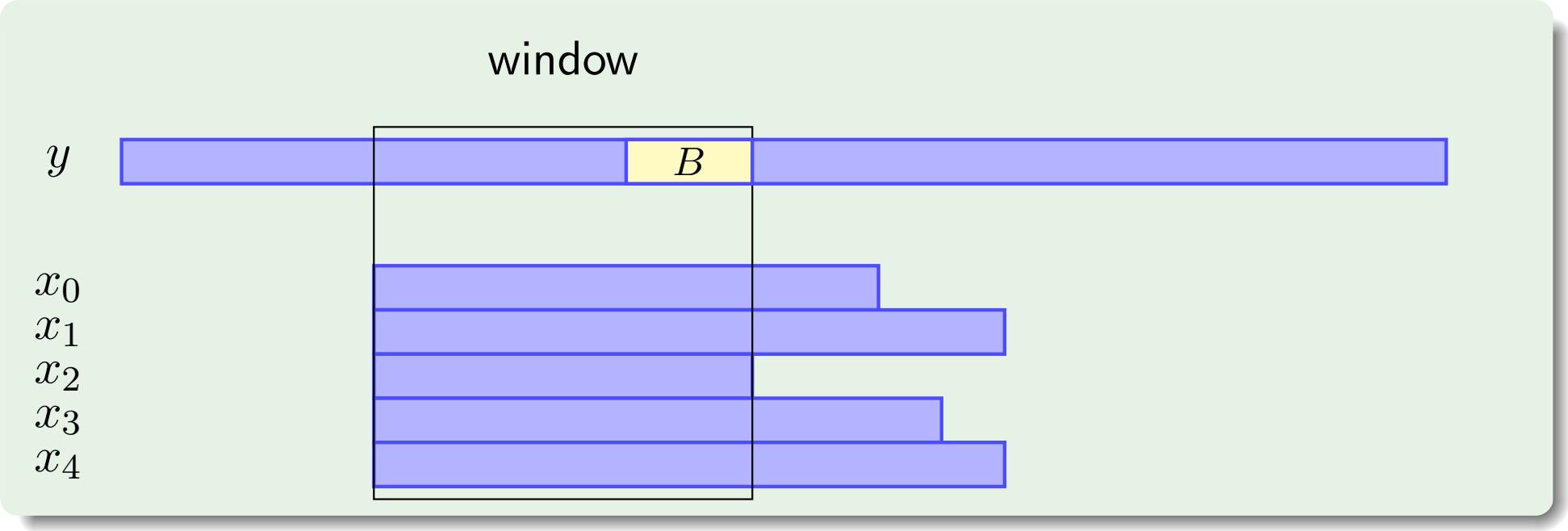
Wu-Manber algorithm



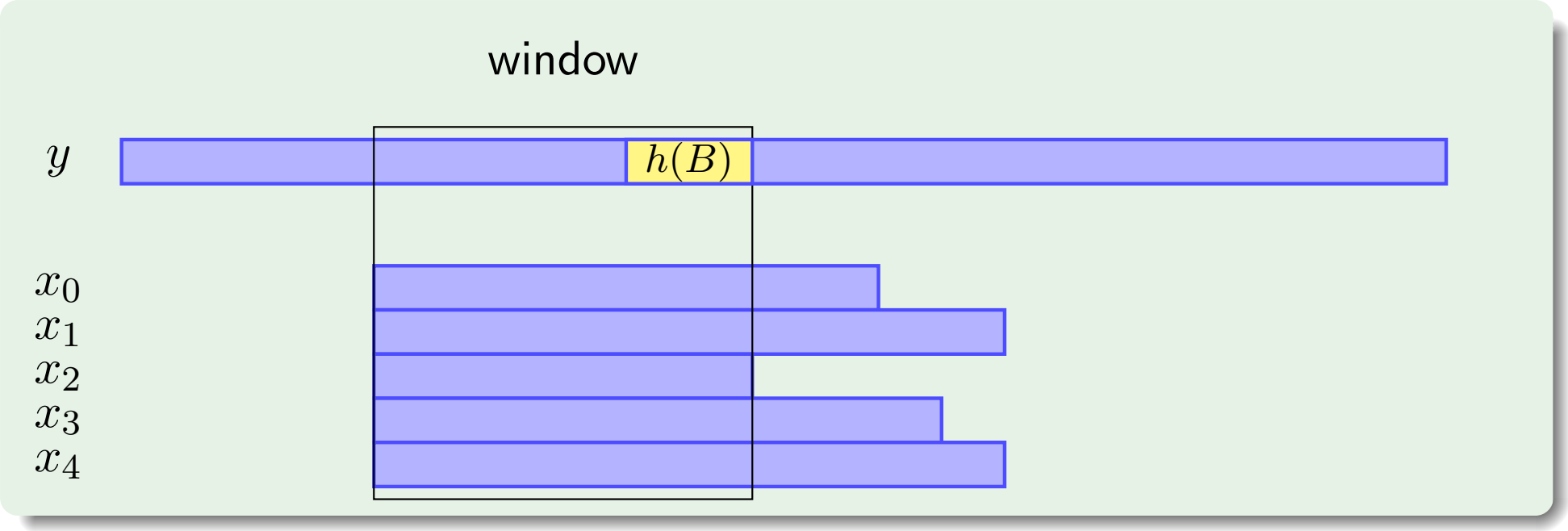
Wu-Manber algorithm



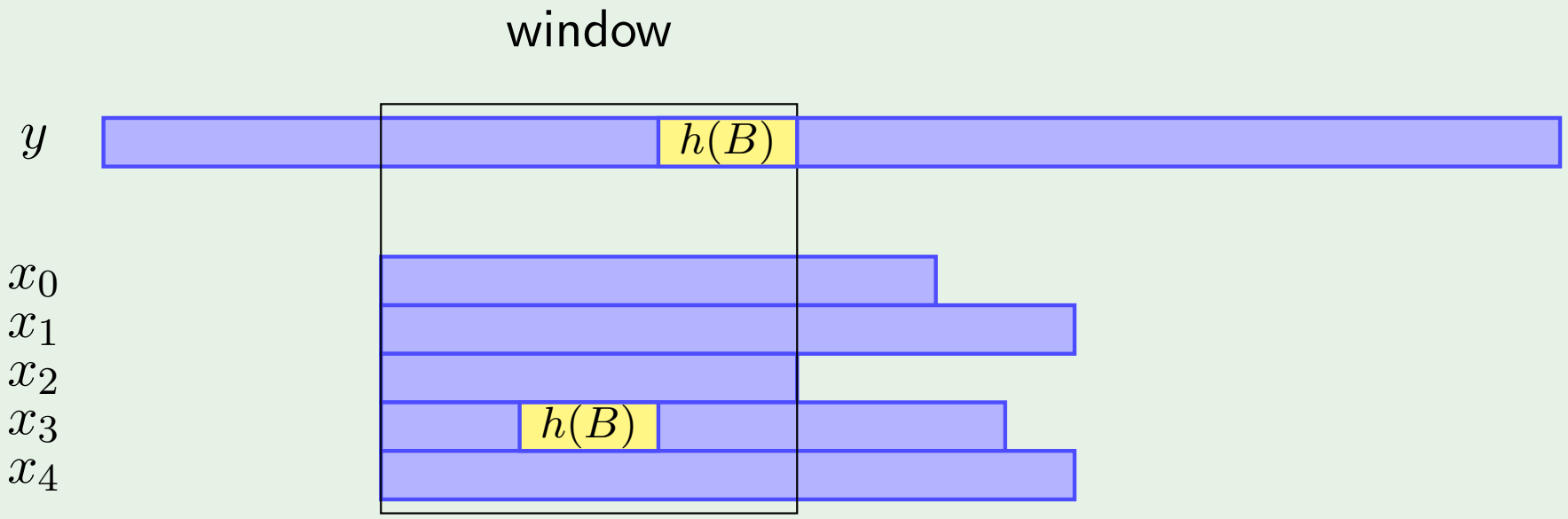
Wu-Manber algorithm



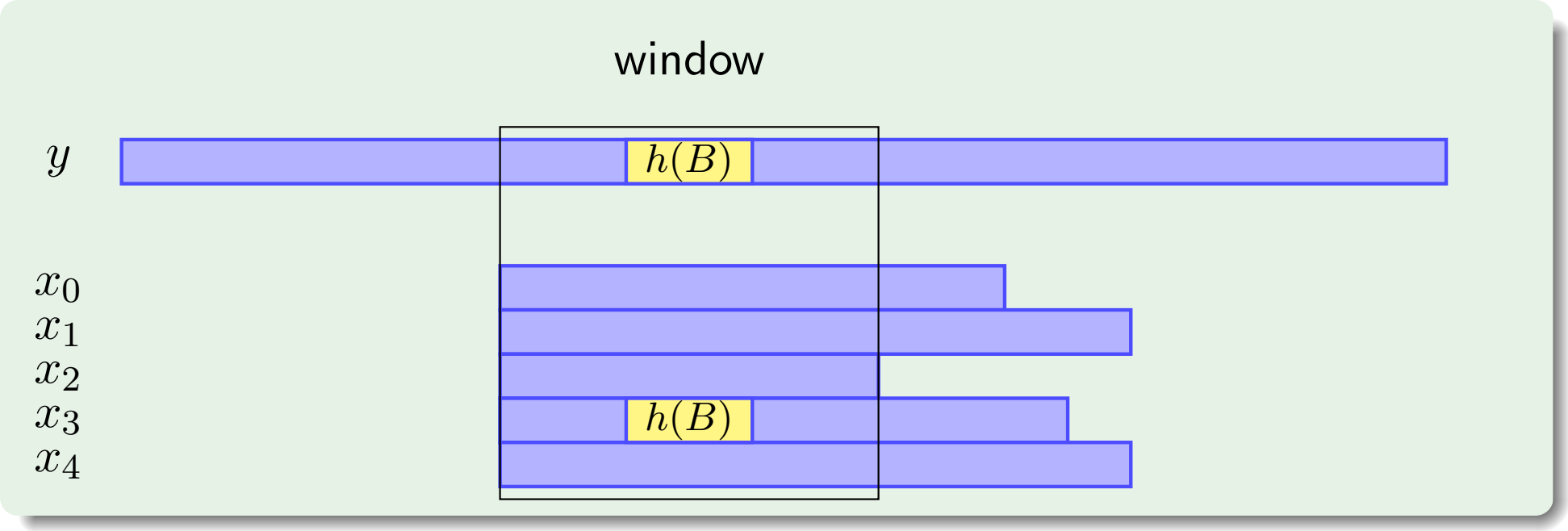
Wu-Manber algorithm



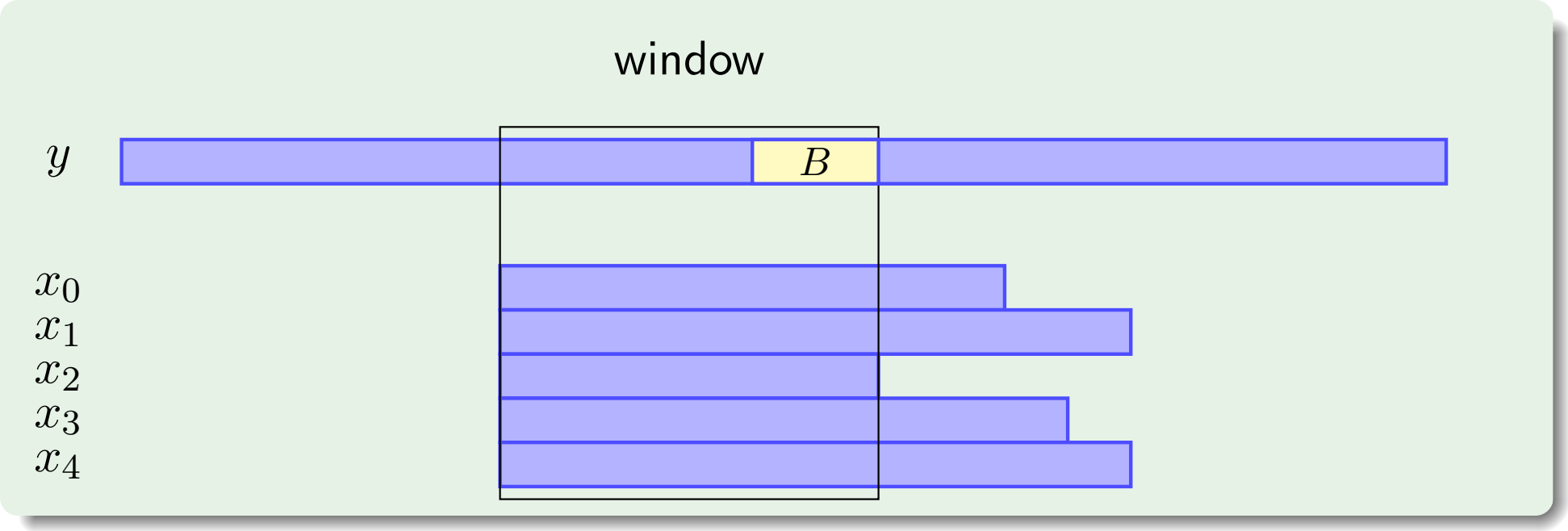
Wu-Manber algorithm



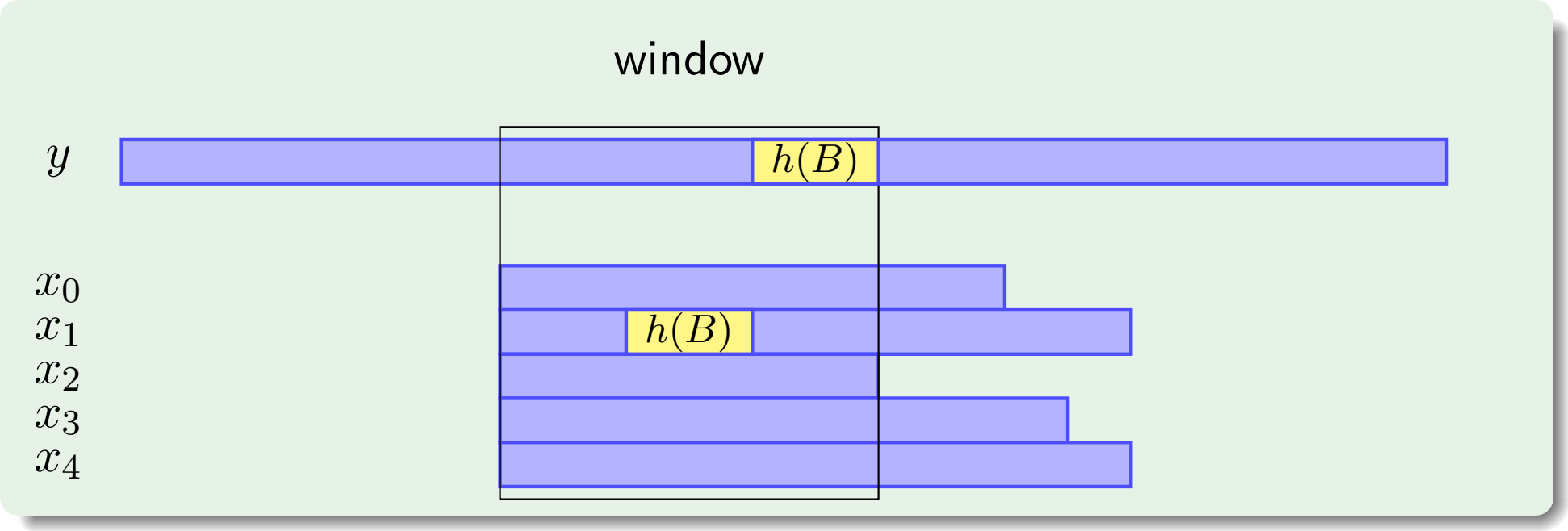
Wu-Manber algorithm



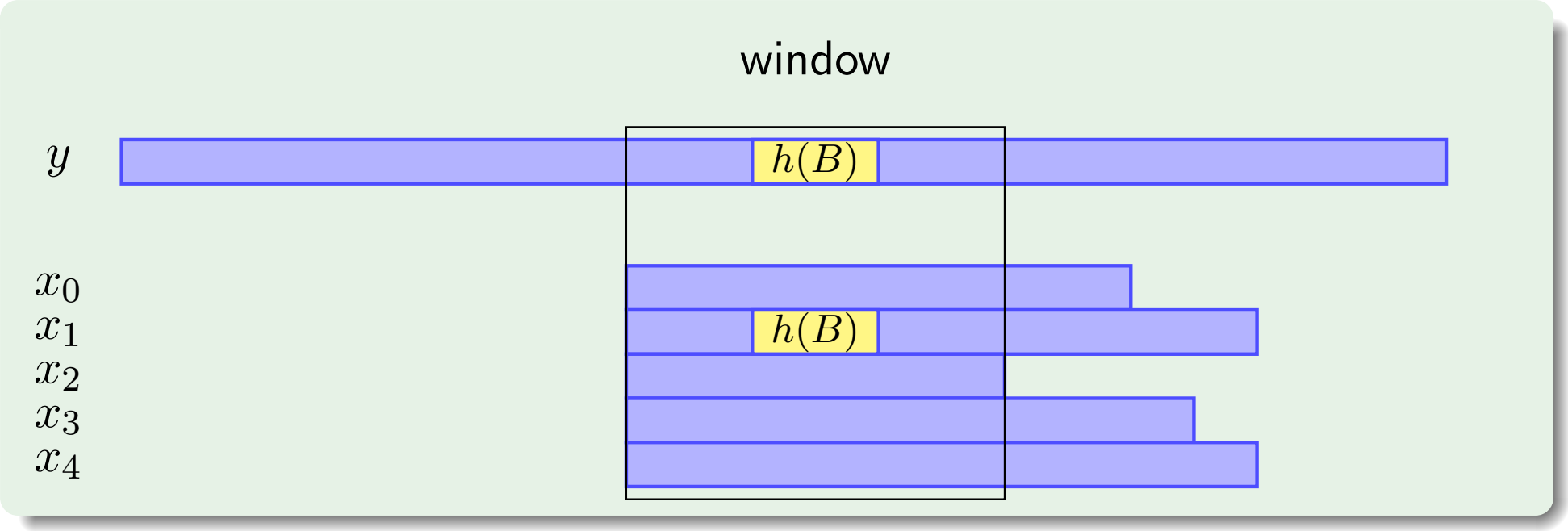
Wu-Manber algorithm



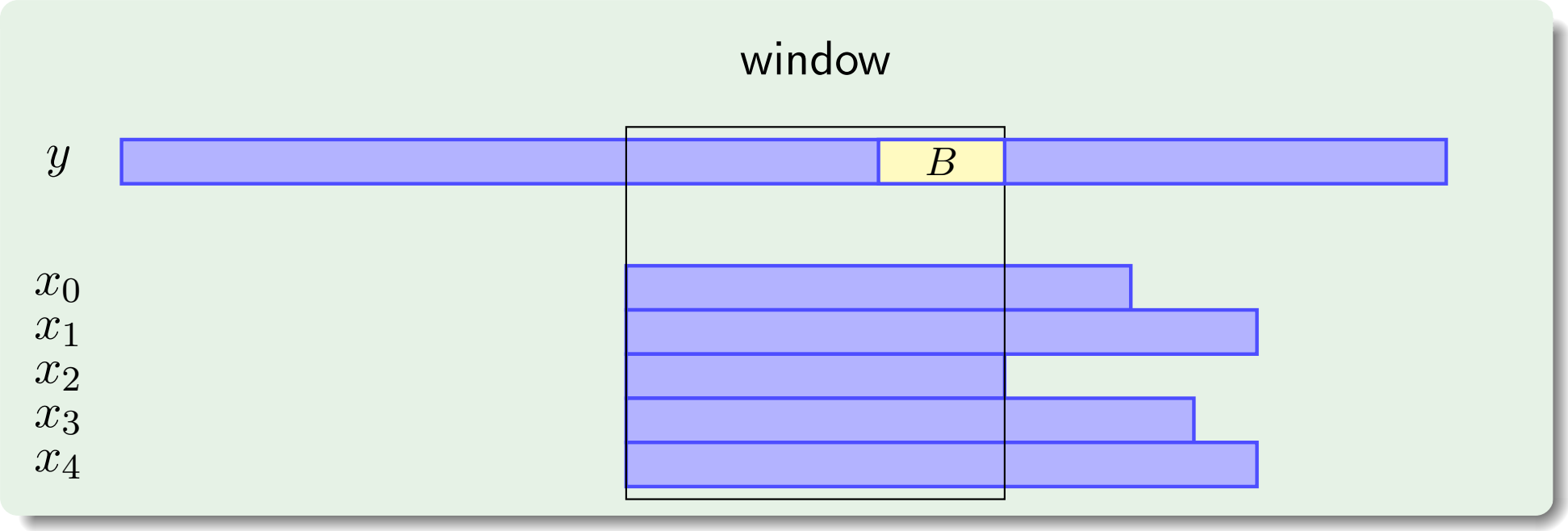
Wu-Manber algorithm



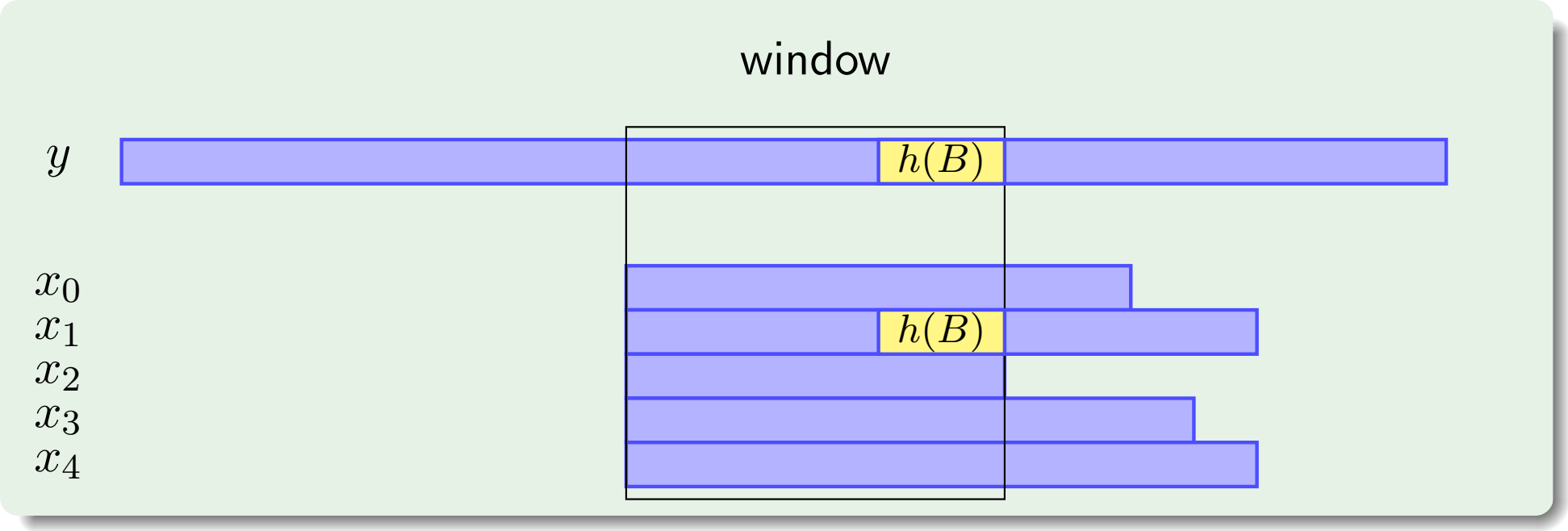
Wu-Manber algorithm



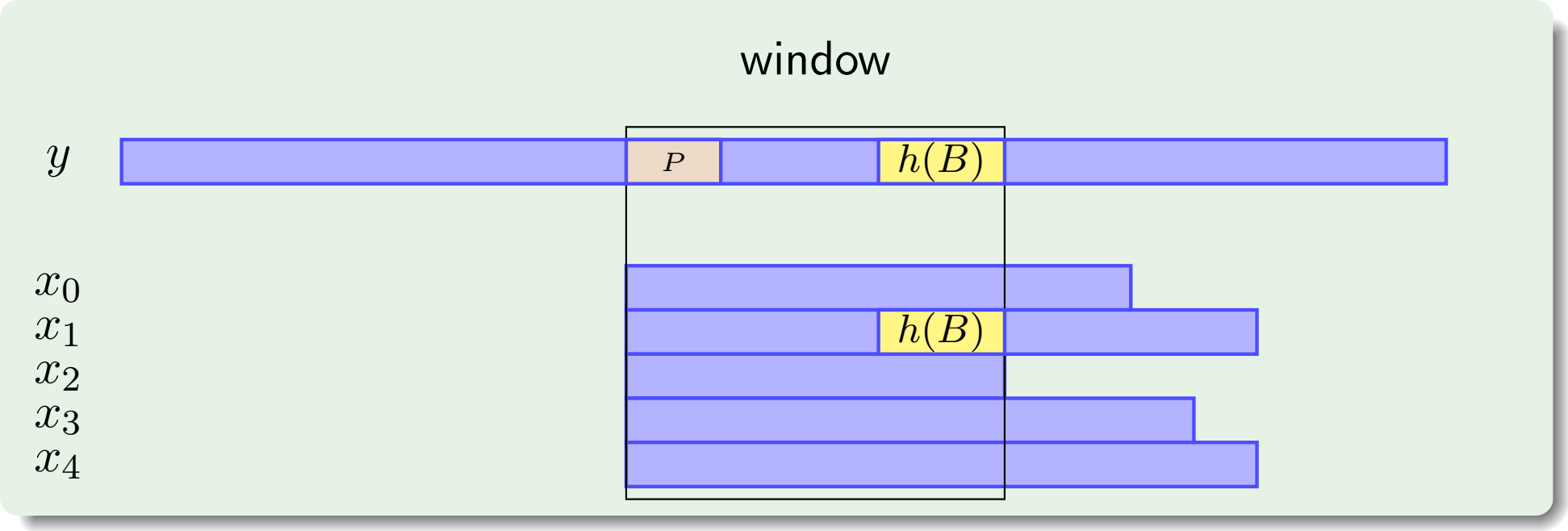
Wu-Manber algorithm



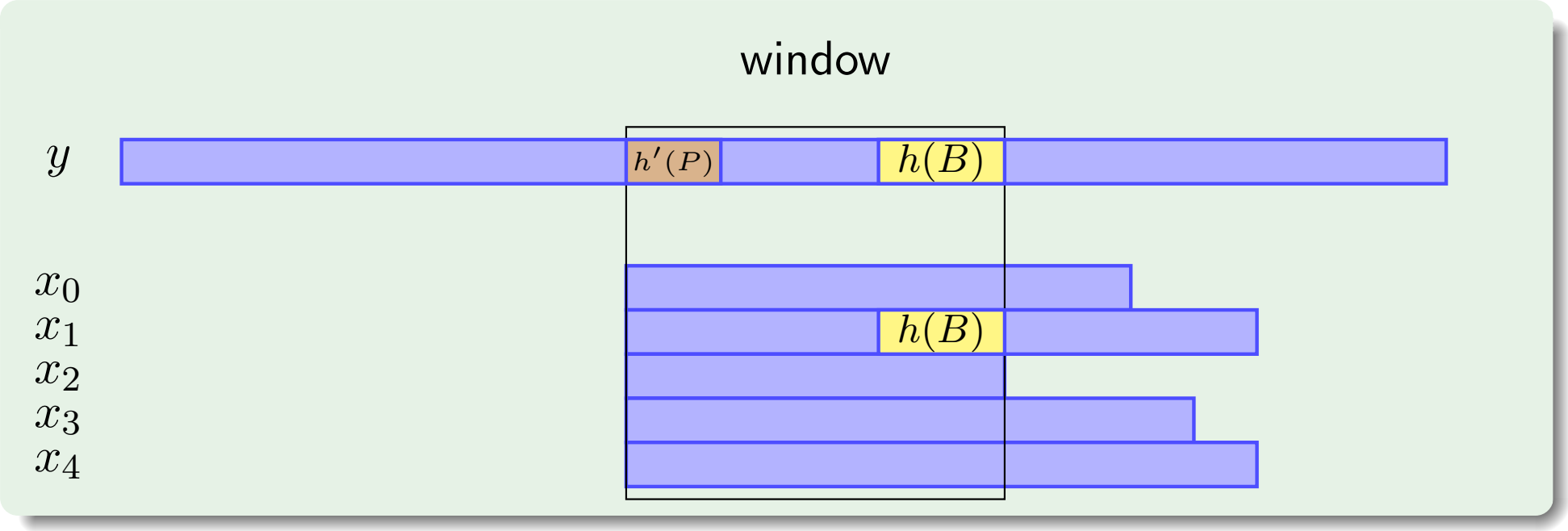
Wu-Manber algorithm



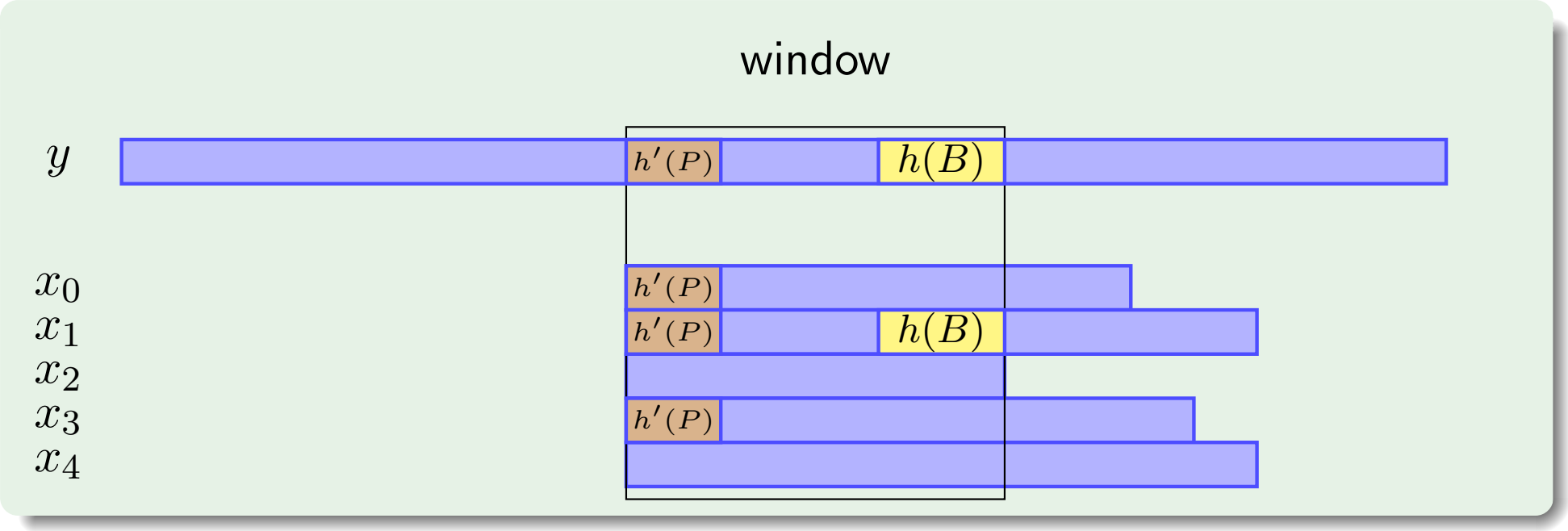
Wu-Manber algorithm



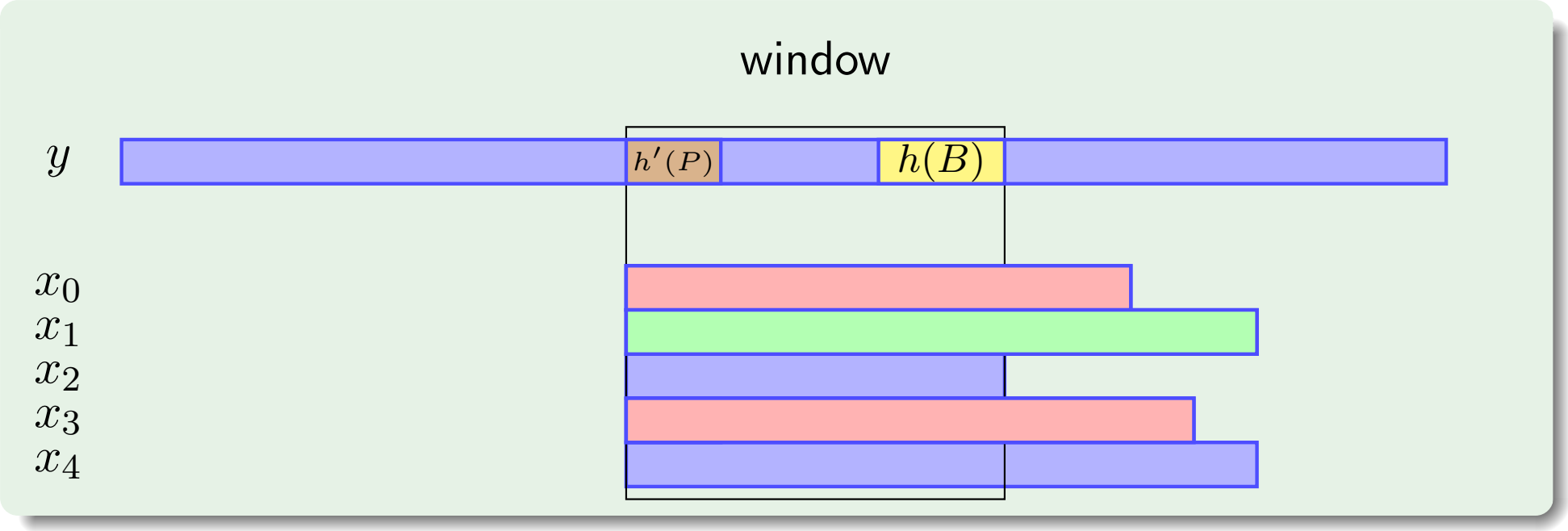
Wu-Manber algorithm



Wu-Manber algorithm



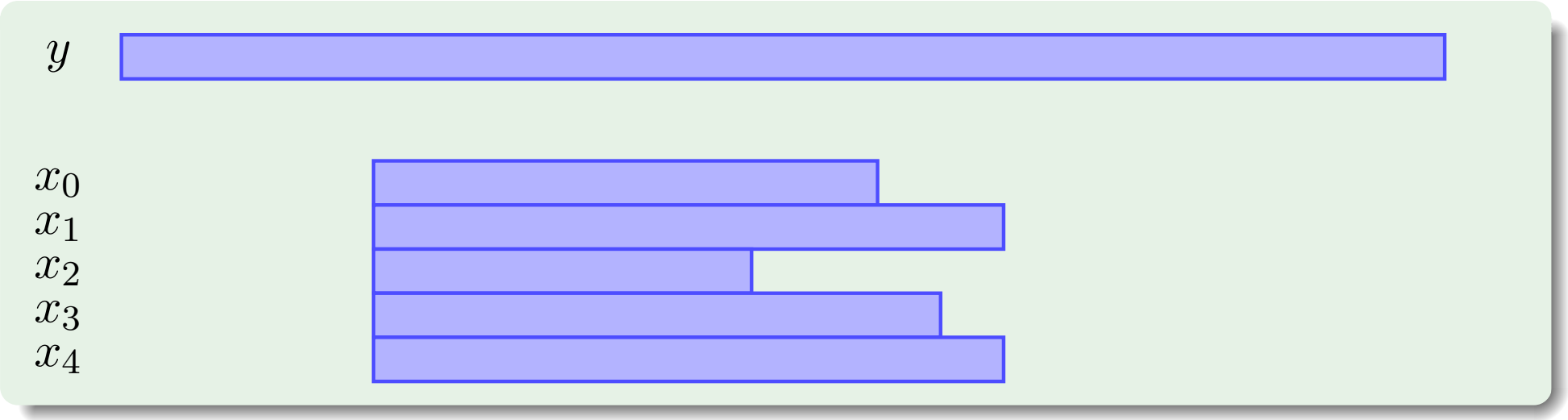
Wu-Manber algorithm



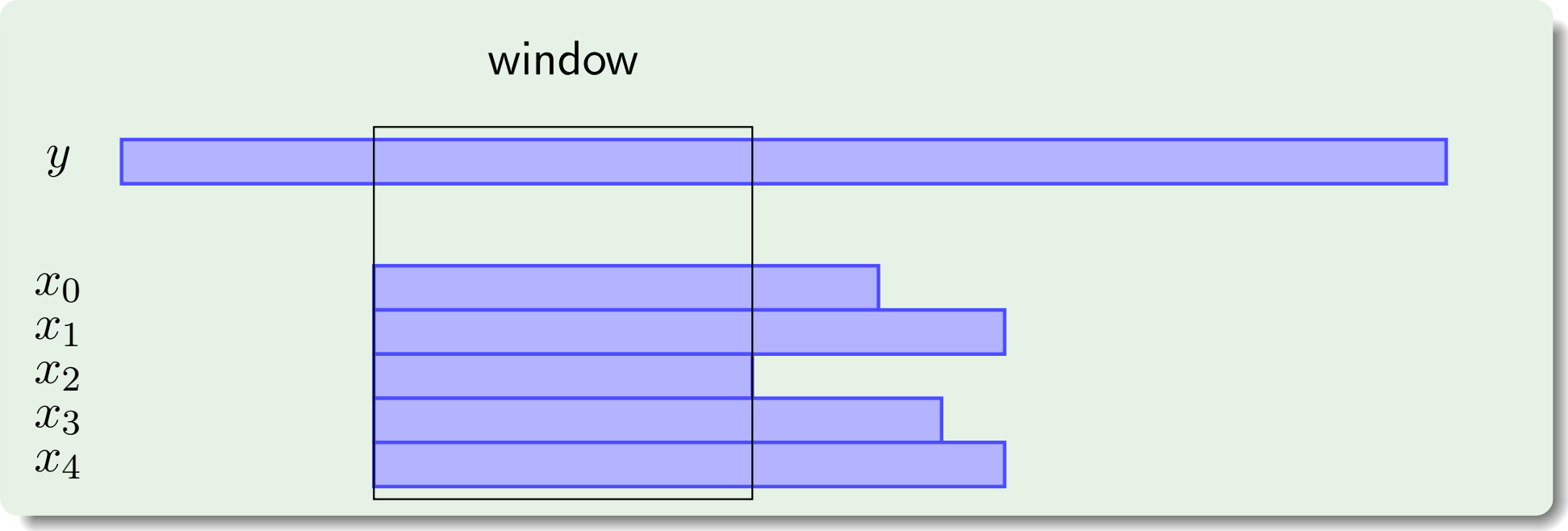
Our new method

γ hashing functions

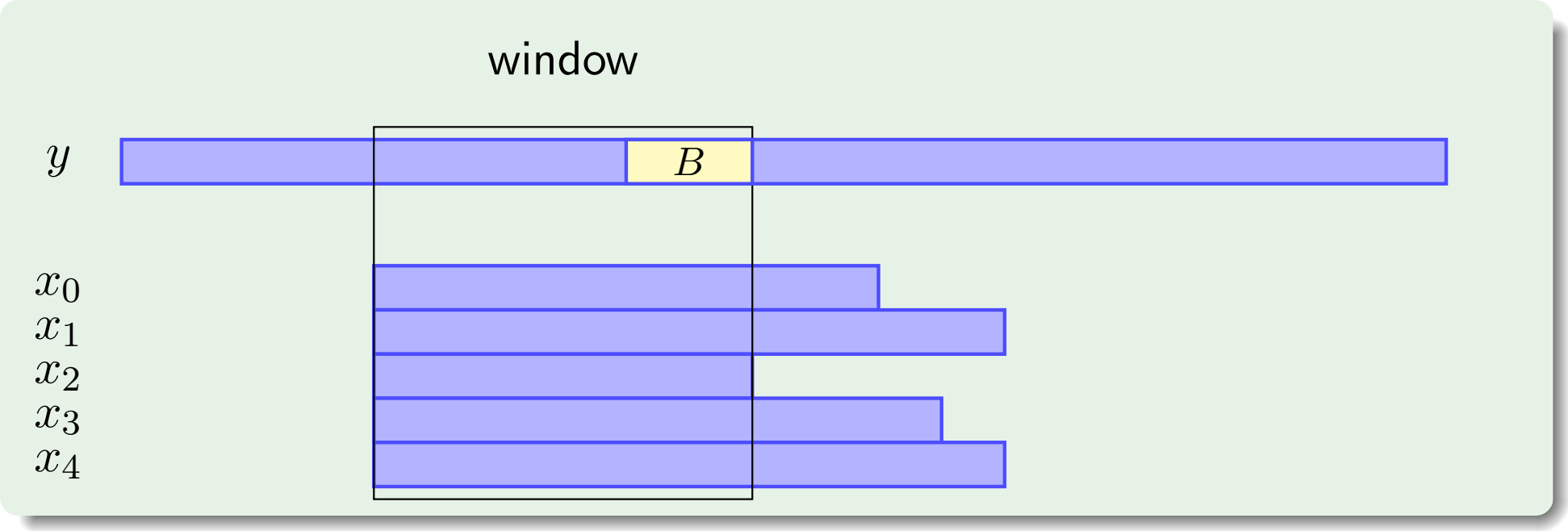
Our new method



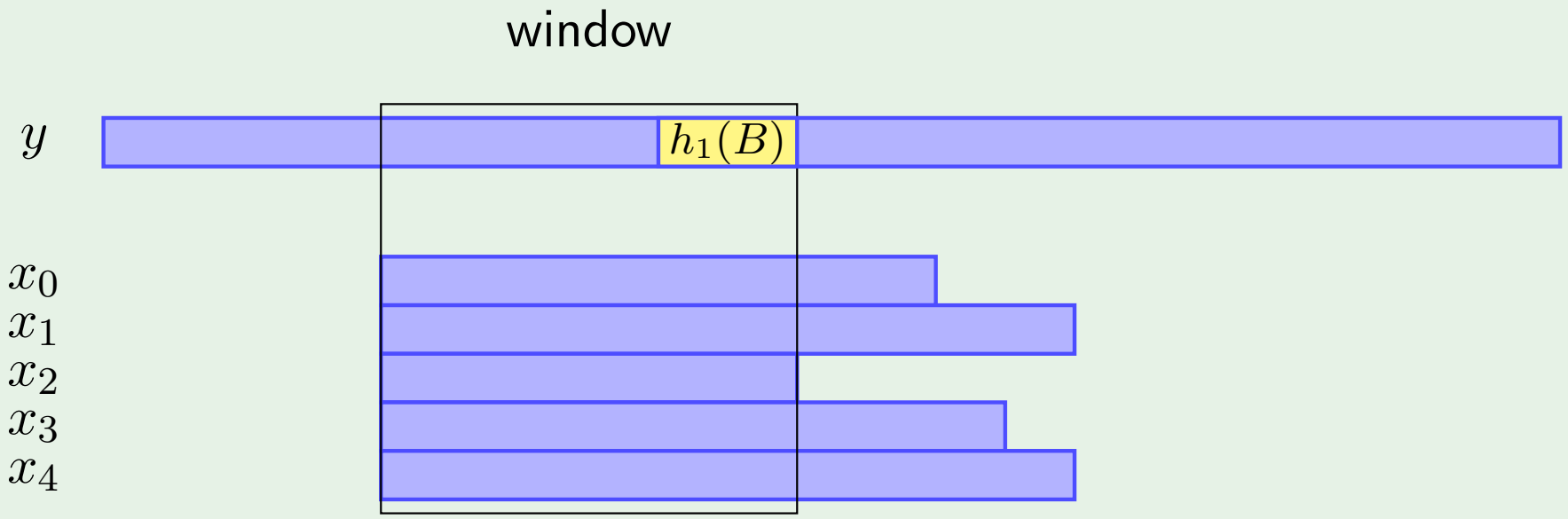
Our new method



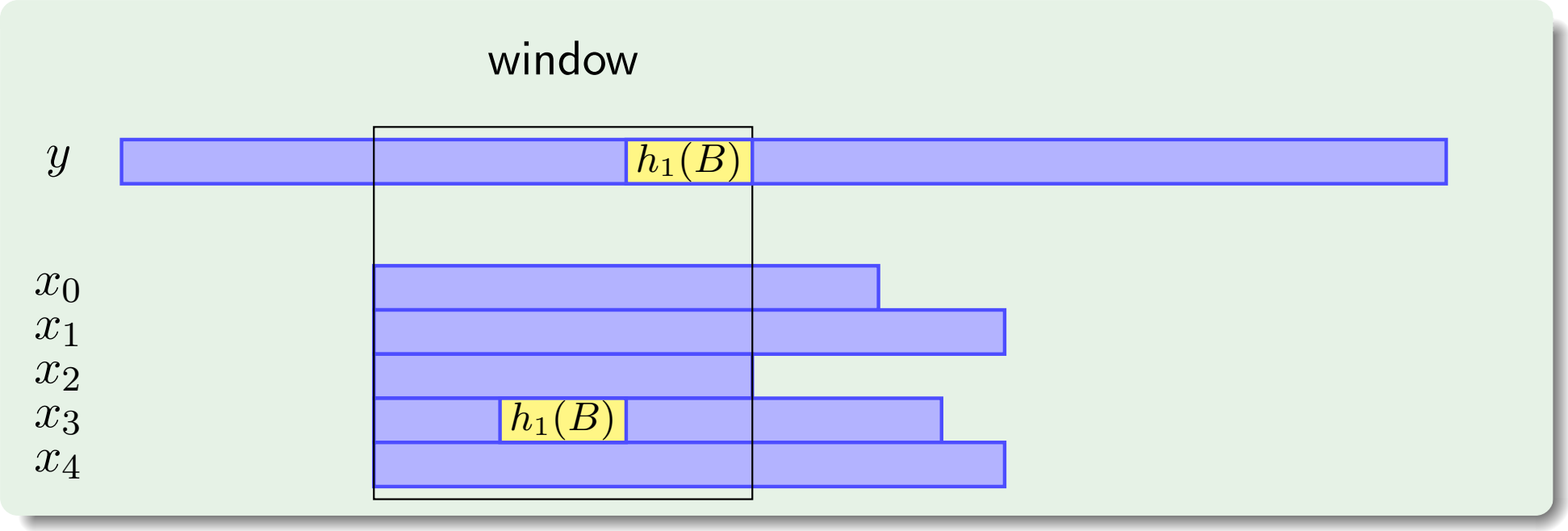
Our new method



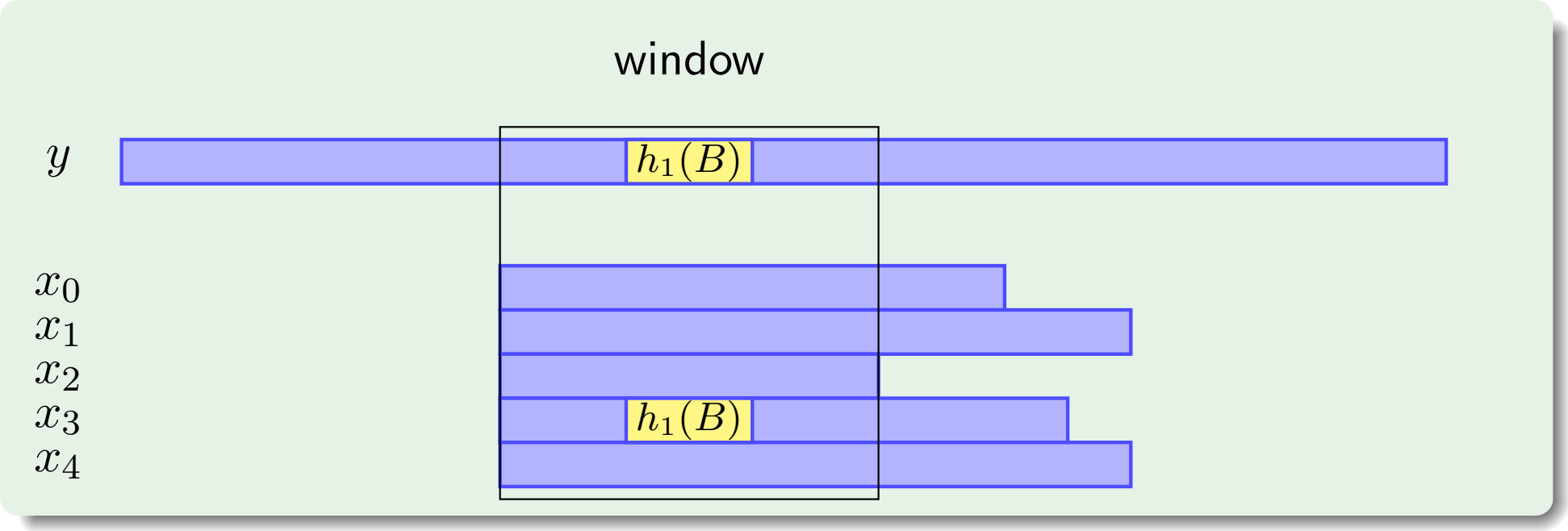
Our new method



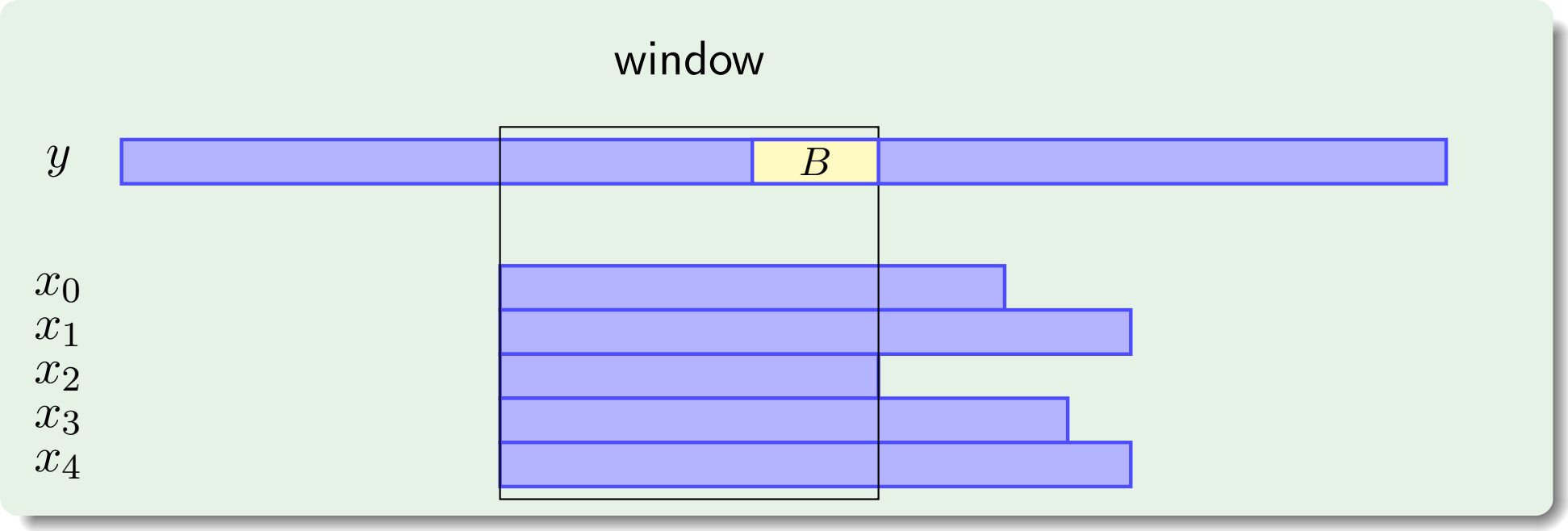
Our new method



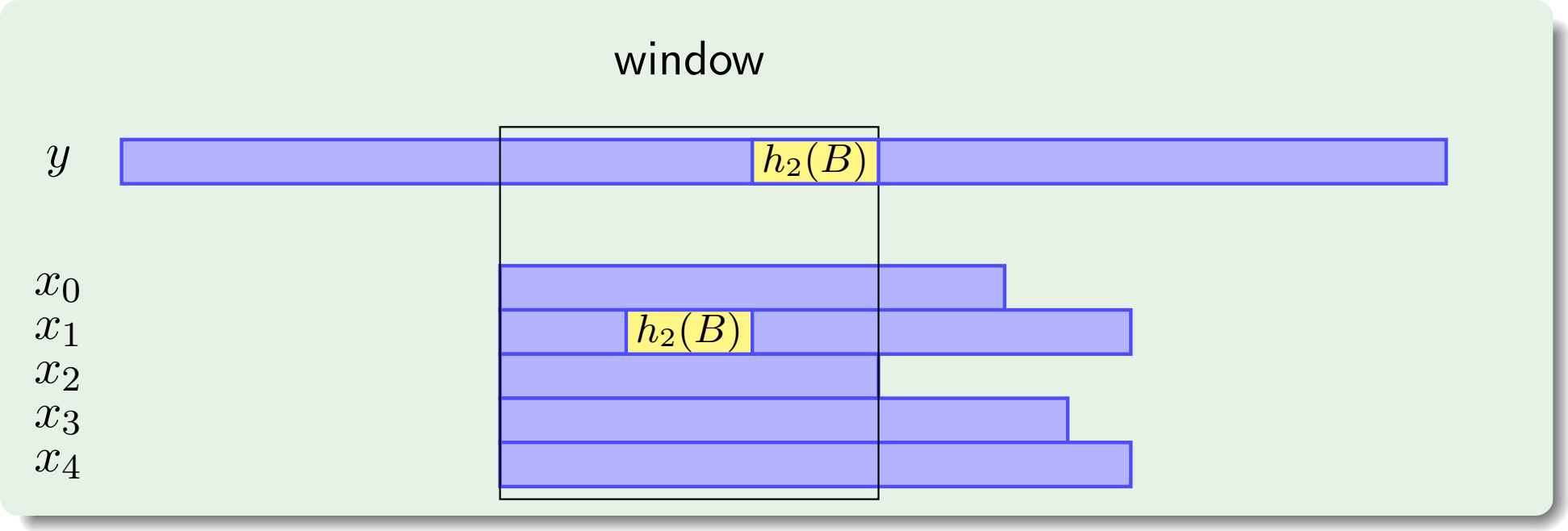
Our new method



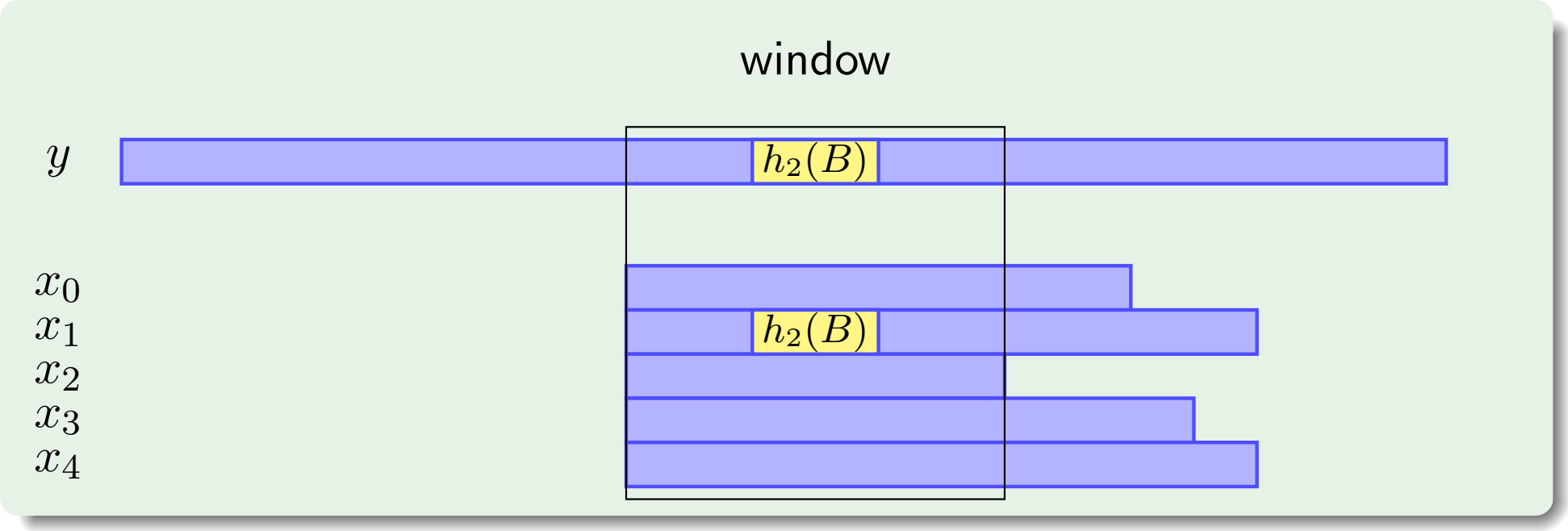
Our new method



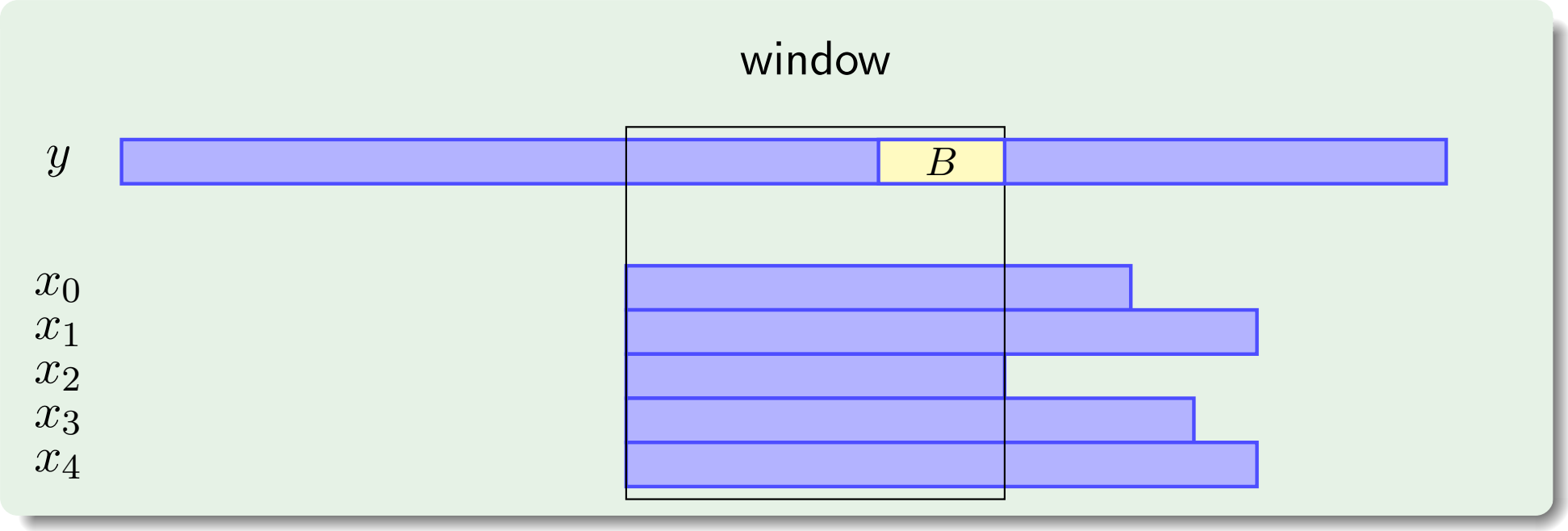
Our new method



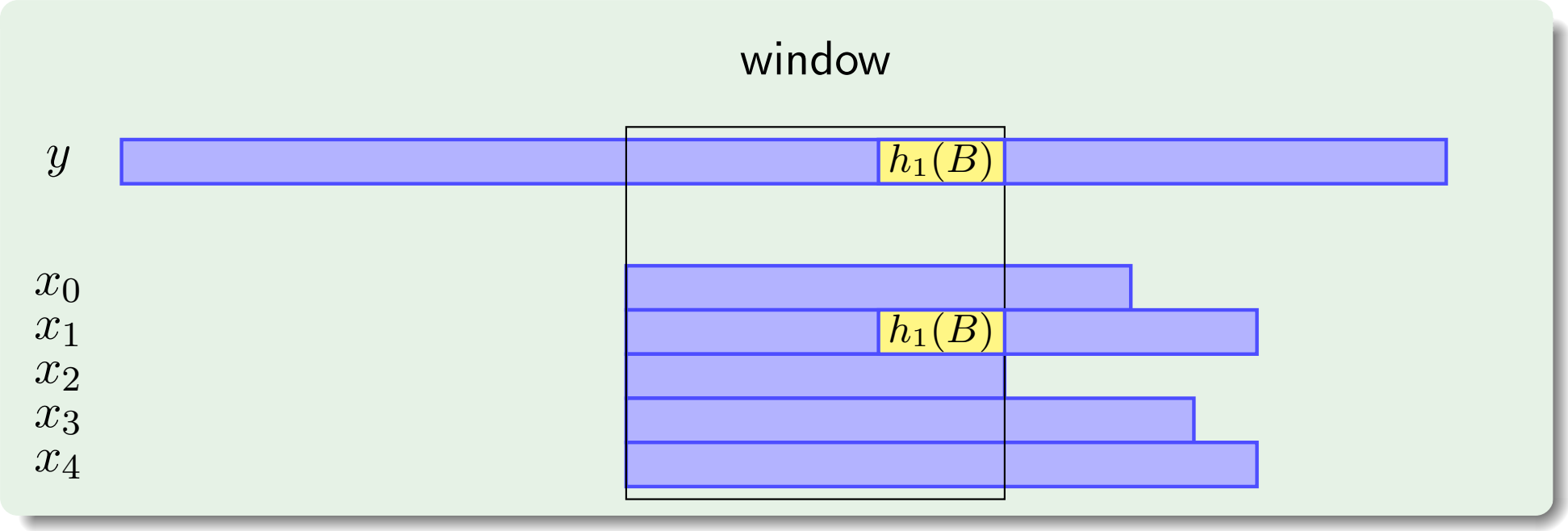
Our new method



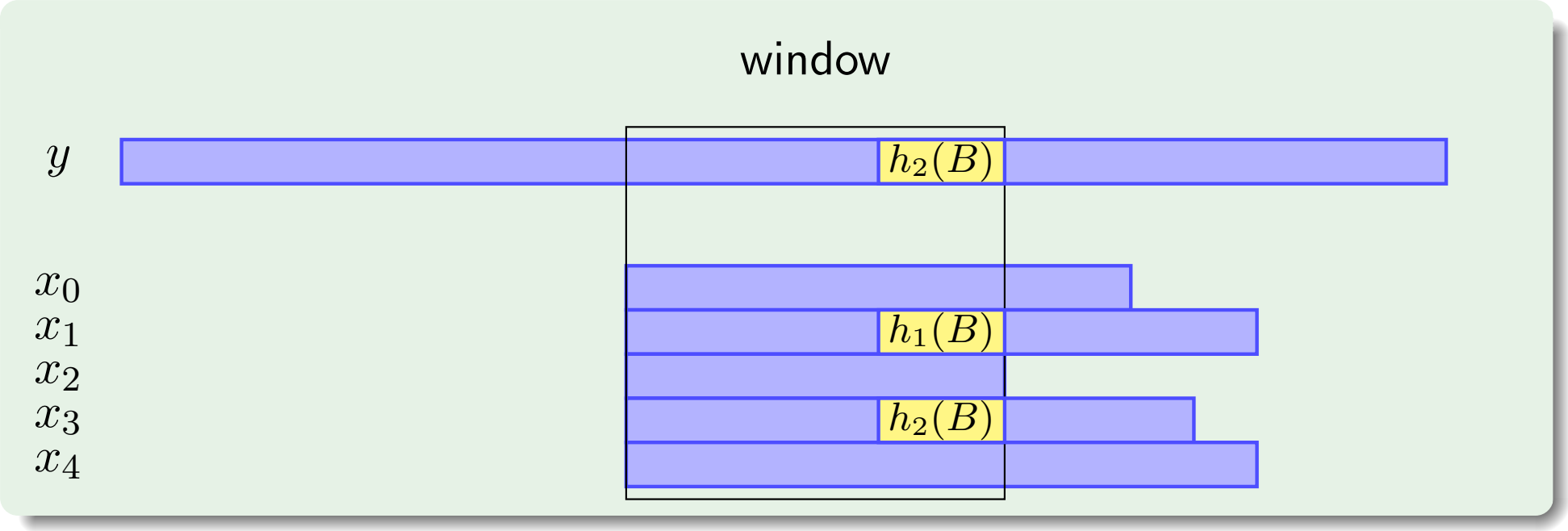
Our new method



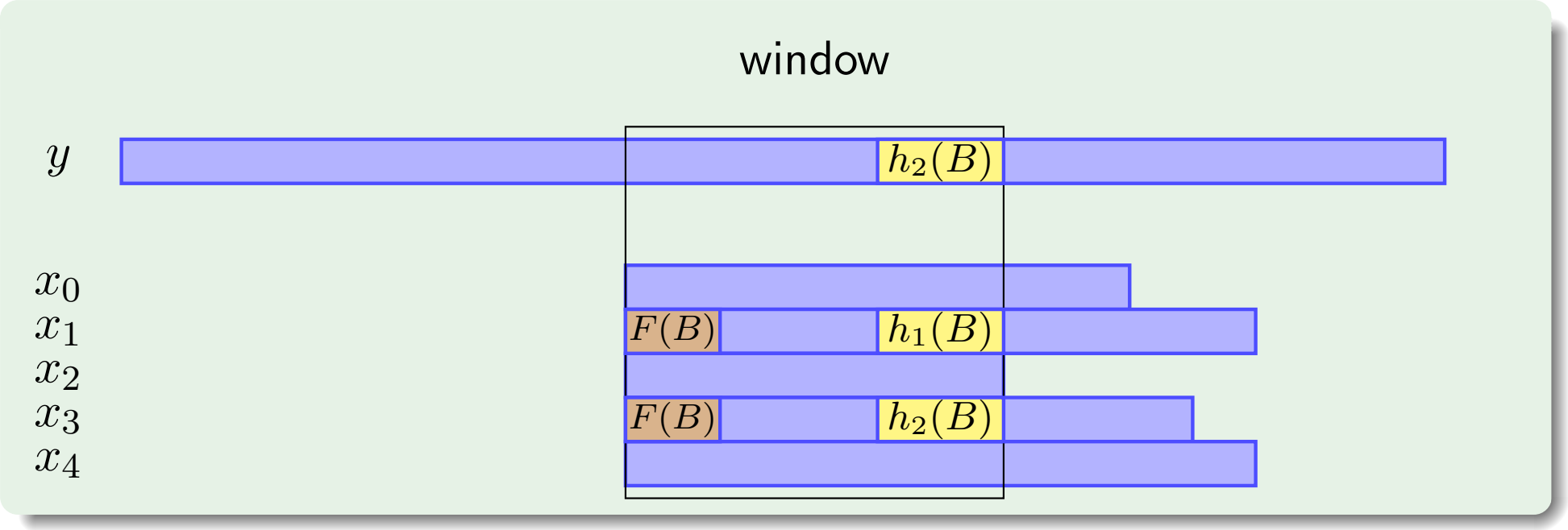
Our new method



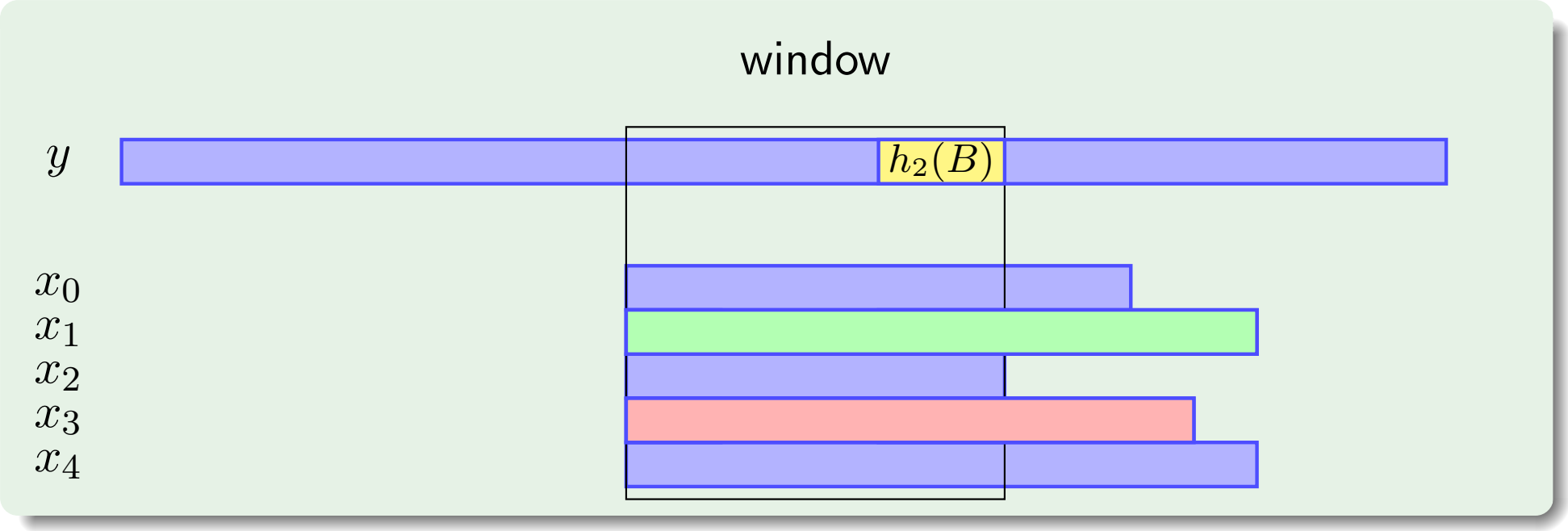
Our new method



Our new method



Our new method



Experimental results

$10,000$ patterns on a genome sequence

m	8			16			32			64			128		
MBNDM	752.32	0.81	751.5	924.18	1.05	923.1	994.26	1.08	993.1	1057.94	1.39	1056	1089.43	1.82	1087
WM$(2,2)$	2063.6	1.26	2062.35	2358.5	1.65	2356.86	2571.7	2.38	2569.36	2681.8	4.07	2677.76	2634.8	7.17	2627.63
WM$(2,3)$	**481.29**	1.47	479.8	**481.97**	2.01	479.9	**516.37**	3.10	513.2	**545.51**	5.72	539.7	**538.88**	10.29	528.5
WM$(3,2)$	**341.06**	1.36	339.6	365.18	1.68	363.5	381.18	2.55	378.6	394.11	4.62	389.4	389.89	8.12	381.7
WM$(3,3)$				**158.99**	2.23	156.7	**158.69**	3.79	154.9	**161.09**	6.79	154.3	**164.62**	12.18	152.4
WM$(4,1)$	1845.5	1.06	1844	2103.1	1.30	2101	2256.6	1.81	2254	2319.1	3.00	2316	2253.6	5.25	2248
WM$(4,2)$	**212.04**	1.28	210.7	185.05	1.83	183.2	188.04	2.99	185.0	192.83	5.48	187.3	195.04	9.82	185.2
WM$(4,3)$				**161.70**	2.23	159.4	**168.89**	4.13	164.7	**166.54**	7.67	158.8	**170.72**	14.11	156.6
WM$(6,1)$	**1463.0**	1.05	1461	1557.6	1.25	1556	1668.0	1.83	1666	1708.5	2.97	1705	1694.0	5.40	1688
WM$(6,2)$				**195.67**	1.53	194.1	203.74	2.65	201.0	218.30	4.96	213.3	213.09	9.37	203.7
WM$(6,3)$							**185.16**	3.63	181.52	**198.35**	7.57	190.78	**195.06**	14.10	180.96
WM$(8,1)$	**473.15**	0.91	472.2	332.39	1.34	331.0	343.12	2.14	340.9	378.56	3.94	374.6	356.40	7.15	349.2
WM$(8,2)$				**135.72**	1.52	134.1	**123.89**	3.23	120.6	**136.73**	6.72	130.0	**136.20**	12.83	123.3
WM$(8,3)$							152.07	4.29	147.7	168.30	10.18	158.1	170.04	20.43	149.6

Experimental results

Naive comparison per text symbol for $10,000$ patterns

m	8	16	32	64	128
MBNDM	11.94	11.76	11.74	11.74	11.76
WM$(2,2)$	54.61	54.44	54.55	54.59	54.56
WM$(2,3)$	7.379	7.388	7.380	7.385	7.398
WM$(3,2)$	4.785	4.770	4.763	4.782	4.783
WM$(3,3)$		.4145	.4151	.4153	.4147
WM$(4,1)$	44.63	44.61	44.54	44.62	44.63
WM$(4,2)$	**1.130**	1.130	1.134	1.132	1.128
WM$(4,3)$		.2634	.2636	.2631	.2626
WM$(6,1)$	33.75	33.66	33.67	33.75	33.74
WM$(6,2)$		1.527	1.530	1.525	1.522
WM$(6,3)$			.5842	.5834	.5833
WM$(8,1)$	5.105	4.971	4.937	4.929	4.929
WM$(8,2)$		**.2024**	.1912	.1903	.1903
WM$(8,3)$			**.1111**	**.1109**	**.1108**

Experimental results

Speed ups obtained via WM(q, γ) vs MBNDM

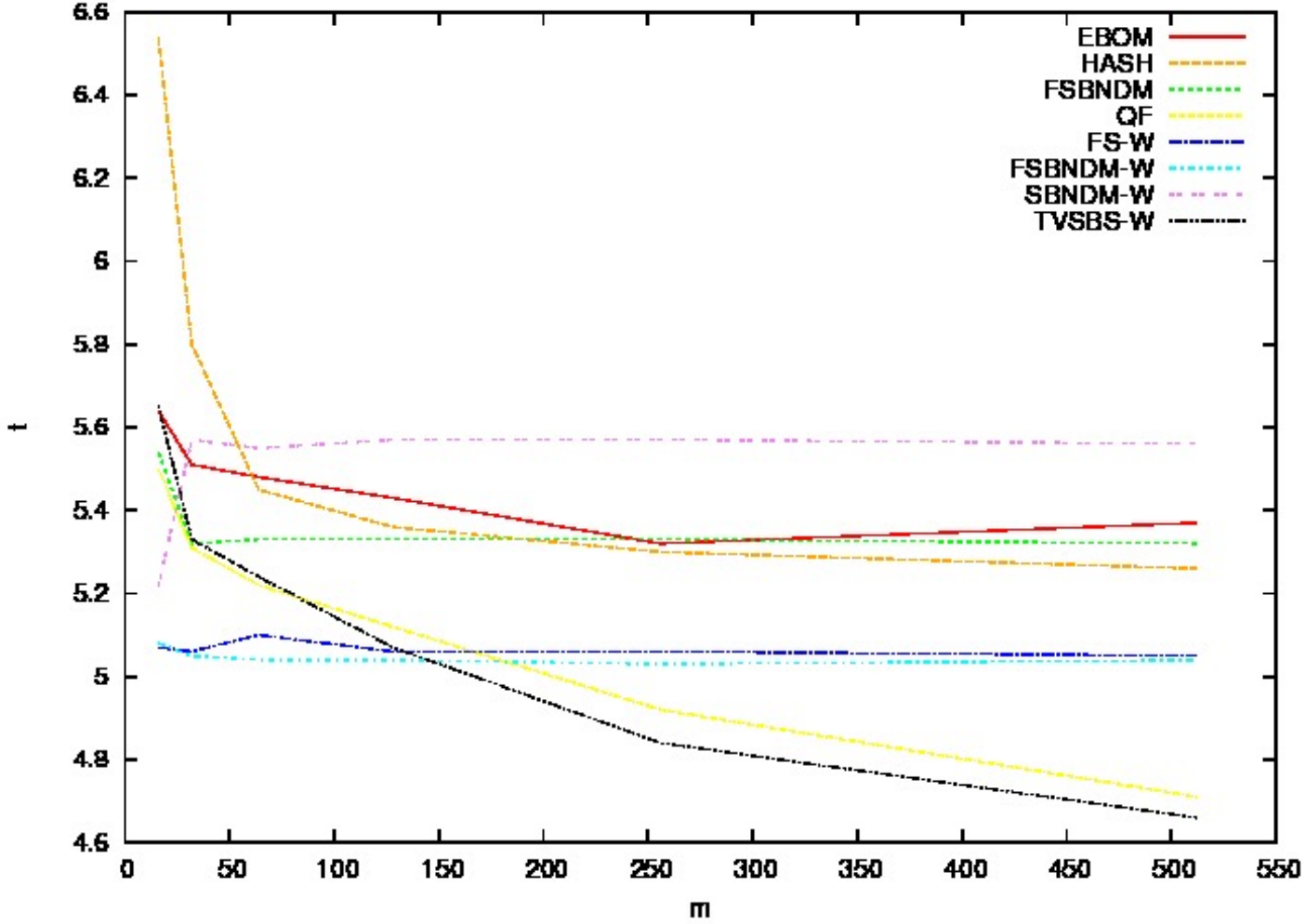
k / m	8	16	32	64	128
100	0.49	0.88	1.47	1.81	2.07
1000	0.89	0.76	1.36	1.58	1.55
$10,000$	3.53	6.80	8.02	7.73	7.99

Reference

S. Faro and T. Lecroq
Fast Searching in Biological Sequences Using Multiple Hash Functions
IEEE 12th International Conference on Bioinformatics and Bioengineering (BIBE 2012), Larnaca, Cyprus, 2012

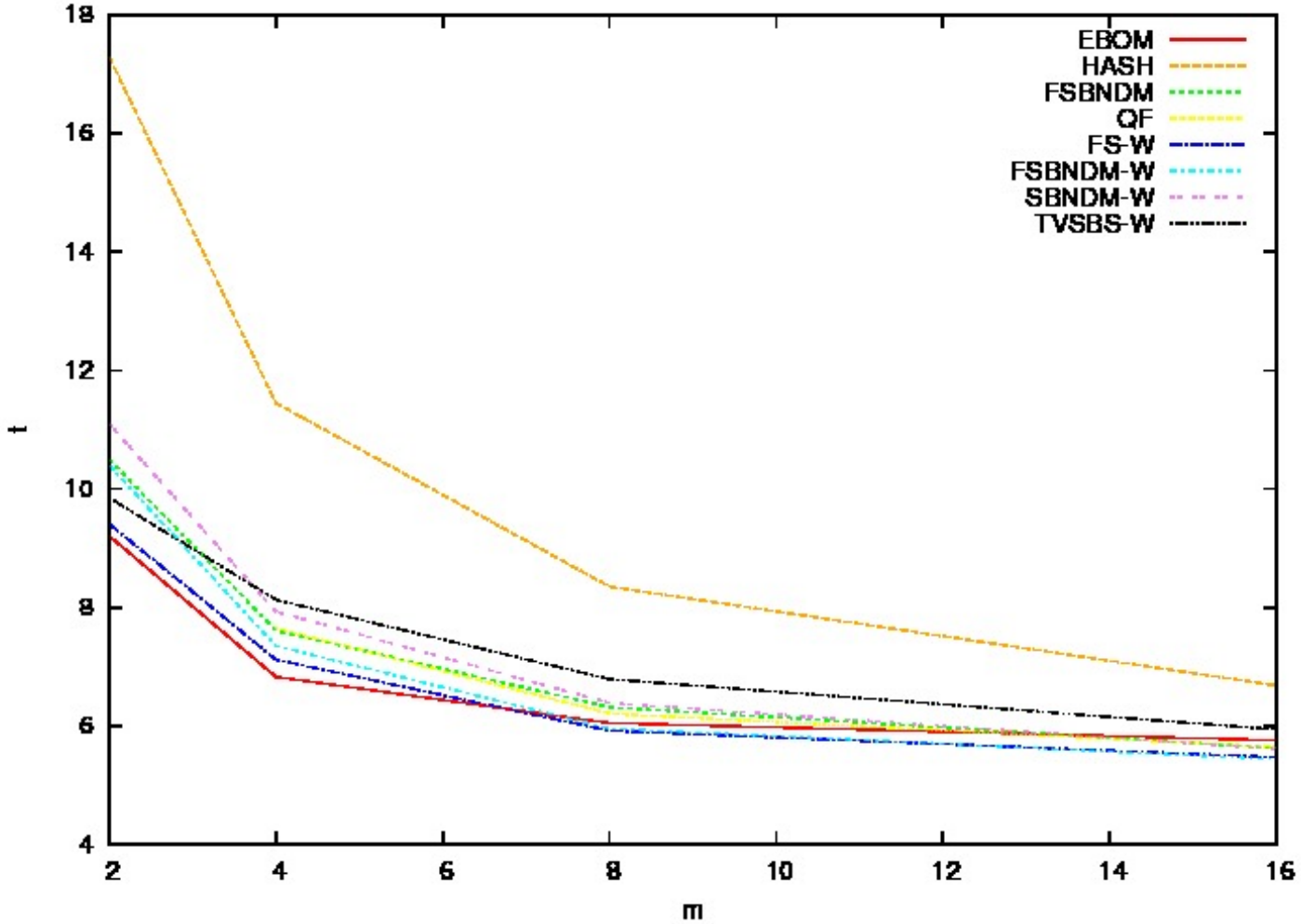
Thank you for your attention!

Multiple windows: experimental results



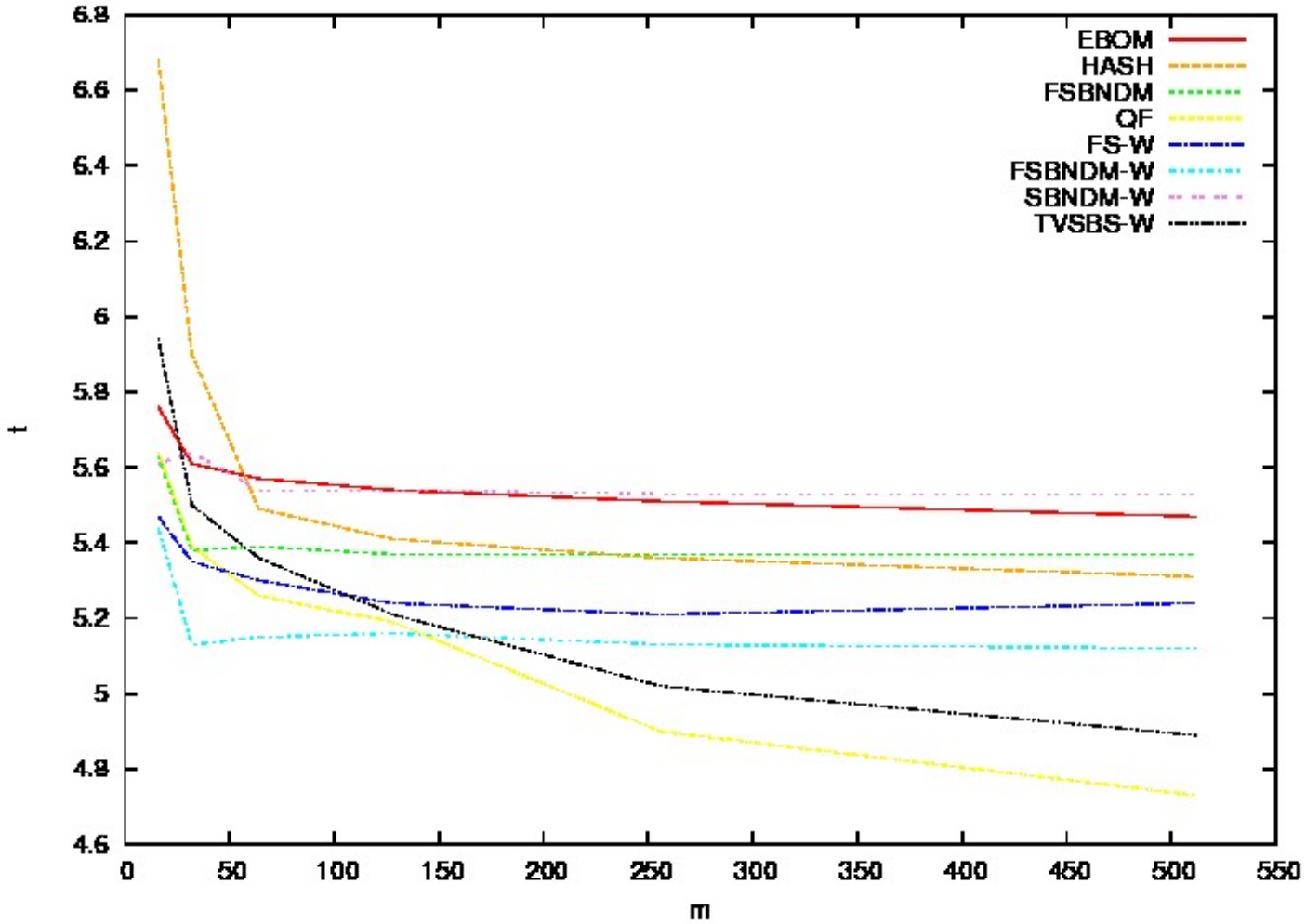
Alphabet size 32, long patterns

Multiple windows: experimental results



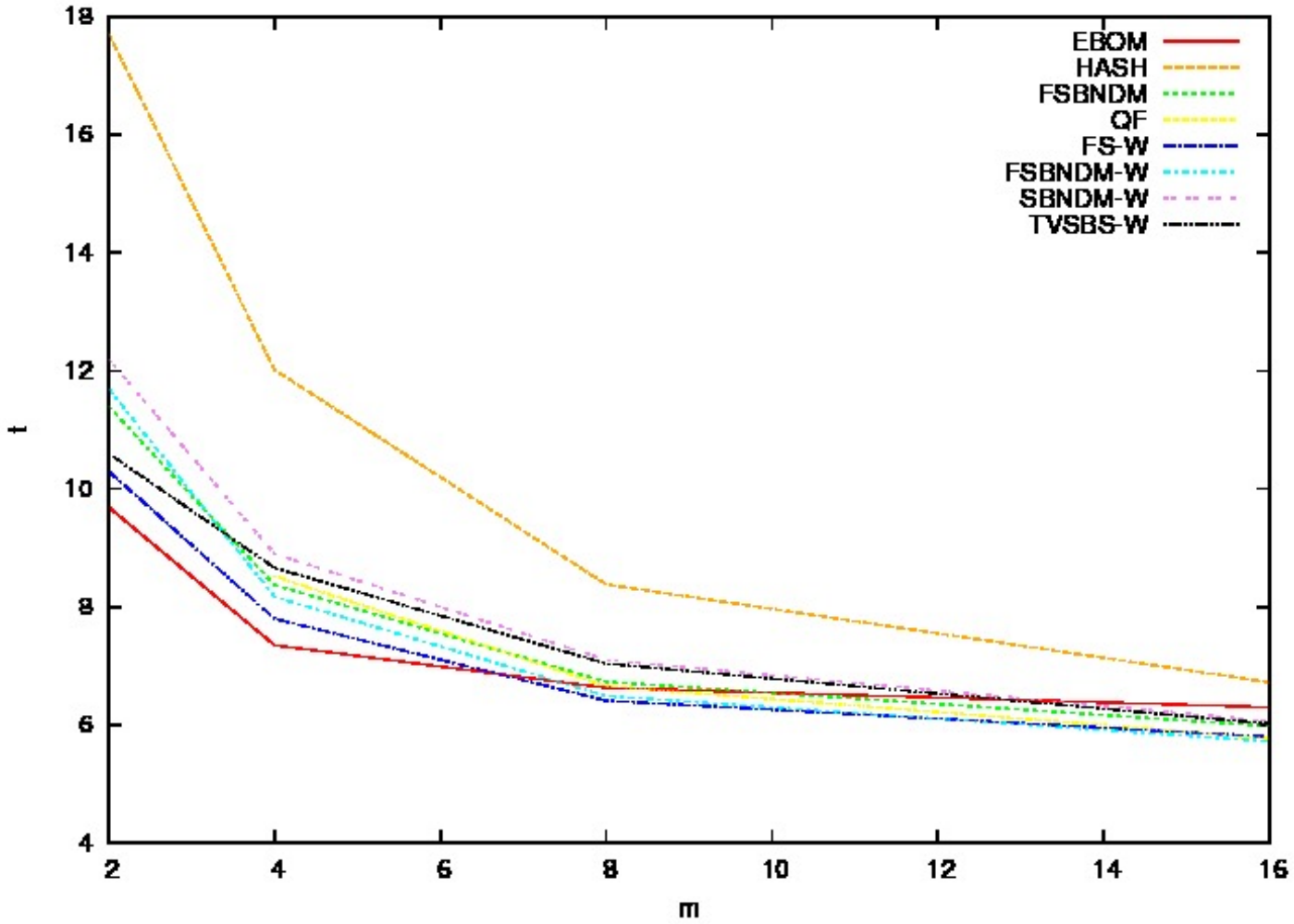
Protein alphabet (size 20), short patterns

Multiple windows: experimental results



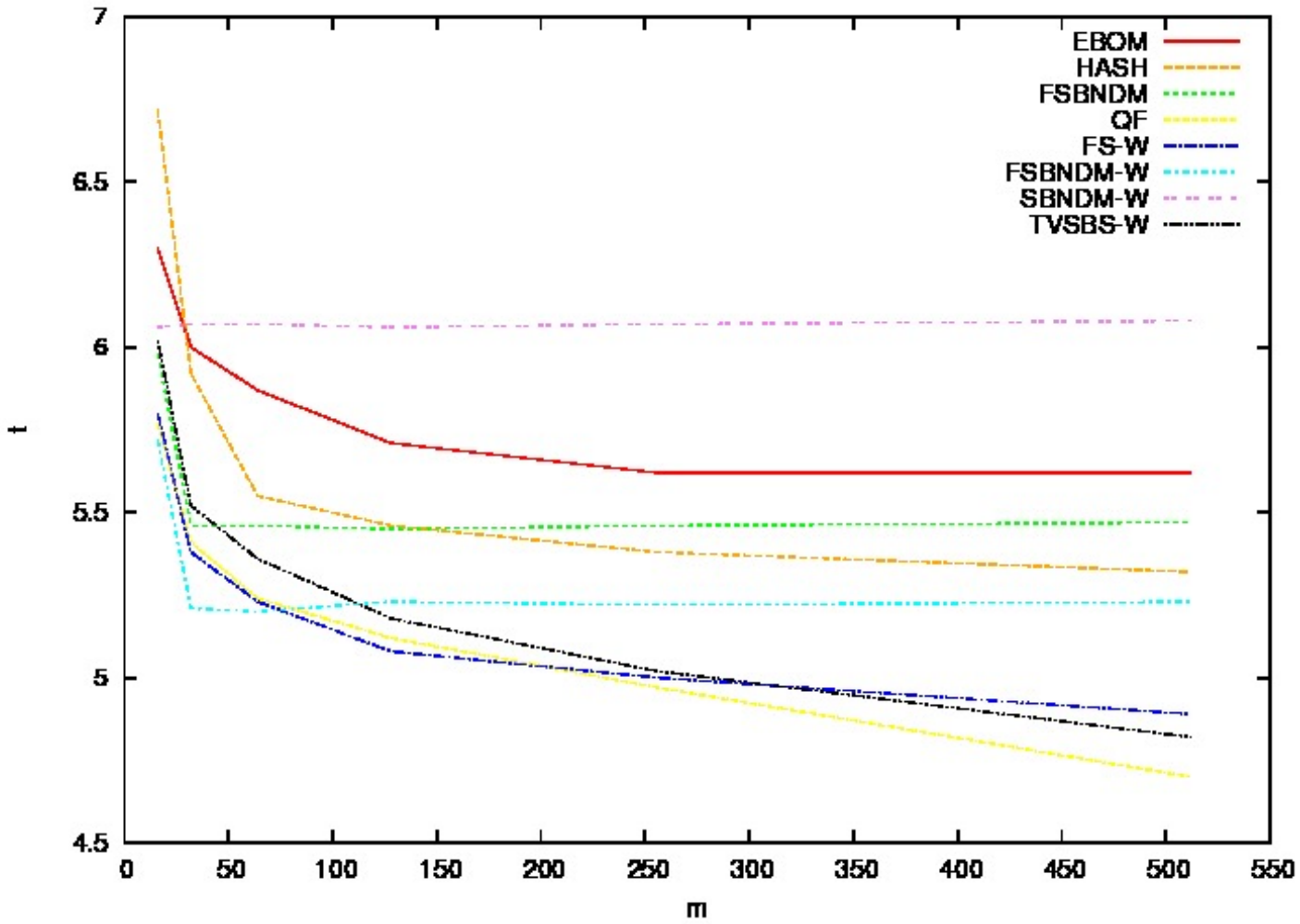
Protein alphabet (size 20), long patterns

Multiple windows: experimental results



English text, short patterns

Multiple windows: experimental results



English text, long patterns

BSDM

Experimental results on a protein sequence

m	2	4	8	16	32	64	128	256	512	1024	2048	4096
BDM	9.82	8.20	7.05	5.80	4.91	4.59	4.53	4.57	4.73	5.30	7.14	10.60
BNDM	9.27	7.67	6.74	5.61	4.81	4.83	4.80	4.80	4.81	4.80	4.82	4.83
SBNDM	9.25	5.93	4.96	4.59	4.41	4.57	4.57	4.57	4.58	4.57	4.62	4.58
BXS	8.41	7.19	6.41	5.45	4.69	4.53	4.39	4.29	4.27	4.17	4.28	105.3
F-BNDM	11.94	8.06	6.22	5.32	4.91	4.79	4.63	4.64	4.62	4.65	4.64	4.65
LBNDM	19.66	12.60	8.84	6.51	5.79	4.88	4.54	4.40	4.34	4.46	6.20	10.44
BSDM	8.37	7.58	7.15	6.89	6.63	6.37	6.14	5.92	5.71	5.56	5.47	5.37
$BSDM_2$	8.29	6.04	5.44	5.15	5.07	4.99	4.99	4.97	4.95	4.93	4.94	4.95
$BSDM_3$	-	6.58	5.25	4.85	4.71	4.64	4.62	4.59	4.59	4.58	4.60	4.60
$BSDM_4$	-	9.71	5.49	4.89	4.68	4.59	4.56	4.53	4.52	4.52	4.50	4.53
$BSDM_5$	-	-	6.04	5.07	4.79	4.68	4.65	4.61	4.61	4.61	4.62	4.64
$BSDM_6$	-	-	7.02	5.19	4.79	4.64	4.60	4.58	4.59	4.57	4.58	4.61
$BSDM_7$	-	-	9.02	5.38	4.82	4.64	4.62	4.58	4.58	4.60	4.58	4.59
$BSDM_8$	-	-	15.11	5.68	4.94	4.70	4.64	4.63	4.62	4.61	4.59	4.58

BSDM

Experimental results on a natural language text

m	2	4	8	16	32	64	128	256	512	1024	2048	4096
BDM	10.50	9.27	7.89	6.29	5.33	4.93	4.73	4.99	4.98	5.51	7.34	10.82
BNDM	10.02	8.74	7.50	6.06	5.20	5.25	5.23	5.25	5.23	5.25	5.25	5.26
SBNDM	9.68	6.39	5.47	5.01	4.76	4.99	4.99	4.99	4.99	4.97	4.98	4.98
BXS	9.12	8.25	7.20	5.91	5.06	4.76	4.50	4.35	4.27	4.09	3.92	3.90
F-BNDM	12.36	8.45	6.64	5.75	5.30	5.04	4.72	4.67	4.66	4.67	4.67	4.67
LBNDM	20.36	13.38	9.38	6.83	5.56	4.99	4.63	4.44	4.35	4.38	4.62	5.69
BSDM	8.90	8.35	7.72	7.15	6.71	6.43	6.16	6.01	5.85	5.79	5.69	5.61
$BSDM_2$	8.41	6.24	5.62	5.37	5.27	5.23	5.18	5.14	5.11	5.09	5.08	5.08
$BSDM_3$	-	6.76	5.40	5.00	4.85	4.79	4.74	4.71	4.69	4.67	4.70	4.71
$BSDM_4$	-	9.88	5.62	4.95	4.76	4.65	4.62	4.61	4.61	4.56	4.57	4.62
$BSDM_5$	-	-	6.02	5.00	4.75	4.65	4.62	4.59	4.59	4.55	4.60	4.63
$BSDM_6$	-	-	7.05	5.16	4.78	4.64	4.59	4.57	4.58	4.54	4.58	4.60
$BSDM_7$	-	-	9.26	5.41	4.84	4.66	4.60	4.56	4.59	4.56	4.57	4.58
$BSDM_8$	-	-	16.07	5.80	4.96	4.72	4.67	4.62	4.60	4.54	4.53	4.55

Multiple hashing: experimental results

100 patterns on a genome sequence

m	8			16			32			64			128		
MBNDM	**16.21**	0.20	16.01	**8.71**	0.20	8.51	8.77	0.20	8.57	8.64	0.20	8.44	8.71	0.20	8.51
WM$(2,2)$	**81.88**	0.52	81.36	**81.69**	0.51	81.18	**83.95**	0.52	83.42	**81.81**	0.53	81.28	**82.33**	0.57	81.76
WM$(2,3)$	84.12	0.64	83.48	85.30	0.65	84.65	85.67	0.66	85.01	84.27	0.67	83.60	85.19	0.73	84.45
WM$(3,2)$	**67.47**	0.52	66.95	**67.73**	0.51	67.21	**67.33**	0.52	66.80	**67.52**	0.54	66.98	**67.67**	0.58	67.09
WM$(3,3)$				92.59	0.65	91.95	91.87	0.66	91.21	91.79	0.71	91.08	92.04	0.76	91.27
WM$(4,1)$	**33.43**	0.38	33.05	30.40	0.38	30.02	28.94	0.38	28.57	29.36	0.39	28.97	29.36	0.42	28.95
WM$(4,2)$	35.24	0.53	34.71	**24.87**	0.53	24.33	**23.32**	0.52	22.79	**23.38**	0.55	22.83	**23.38**	0.60	22.78
WM$(4,3)$				26.47	0.66	25.81	23.93	0.66	23.27	23.95	0.71	23.25	23.54	0.74	22.79
WM$(6,1)$	**32.92**	0.37	32.54	20.26	0.37	19.89	16.50	0.37	16.12	15.45	0.39	15.06	14.71	0.39	14.31
WM$(6,2)$				**15.43**	0.52	14.91	11.08	0.52	10.56	10.29	0.56	9.74	9.92	0.59	9.33
WM$(6,3)$							**10.98**	0.66	10.33	**9.99**	0.71	9.28	**9.64**	0.77	8.87
WM$(8,1)$	**50.51**	0.36	50.14	**9.86**	0.37	9.49	**5.95**	0.37	5.58	**4.77**	0.40	4.37	**4.19**	0.43	3.77
WM$(8,2)$				15.61	0.53	15.08	6.79	0.54	6.26	5.13	0.57	4.56	4.46	0.63	3.83
WM$(8,3)$							8.29	0.67	7.61	5.73	0.74	4.99	4.93	0.86	4.08

Multiple hashing: experimental results

1000 patterns on a genome sequence

m	8			16			32			64			128		
MBNDM	**85.11**	0.27	84.84	**30.76**	0.29	30.47	30.76	0.31	30.45	31.11	0.33	30.78	31.30	0.35	30.95
WM$(2,2)$	**224.92**	0.63	224.2	**230.01**	0.66	229.3	**240.50**	0.72	239.7	**247.47**	0.91	246.5	**247.68**	1.23	246.4
WM$(2,3)$	129.26	0.78	128.4	128.19	0.82	127.3	128.88	0.94	127.9	132.65	1.21	131.4	133.54	1.70	131.8
WM$(3,2)$	**101.13**	0.62	100.5	**103.12**	0.67	102.4	**102.81**	0.75	102.0	**102.44**	0.94	101.5	**103.98**	1.30	102.6
WM$(3,3)$				113.67	0.84	112.8	114.07	1.02	113.0	113.61	1.35	112.2	114.36	1.85	112.5
WM$(4,1)$	189.21	0.45	188.7	192.03	0.48	191.5	201.14	0.52	200.6	206.16	0.65	205.5	209.00	0.85	208.1
WM$(4,2)$	**95.47**	0.61	94.86	**91.59**	0.69	90.91	**91.55**	0.80	90.74	**91.51**	1.05	90.46	**92.24**	1.51	90.73
WM$(4,3)$				131.81	0.82	130.9	132.33	1.03	131.3	131.56	1.38	130.1	132.66	2.05	130.6
WM$(6,1)$	**155.74**	0.44	155.3	147.53	0.47	147.0	152.15	0.52	151.6	154.20	0.64	153.5	155.51	0.87	154.6
WM$(6,2)$				**72.32**	0.64	71.68	**71.47**	0.76	70.71	**71.37**	1.00	70.37	**71.59**	1.42	70.17
WM$(6,3)$							85.67	0.96	84.70	85.74	1.34	84.40	86.16	2.04	84.13
WM$(8,1)$	**98.58**	0.43	98.14	**40.39**	0.48	39.91	33.16	0.57	32.59	31.16	0.75	30.42	31.01	1.07	29.94
WM$(8,2)$				40.79	0.65	40.14	**22.48**	0.83	21.65	20.43	1.16	19.27	20.50	1.79	18.71
WM$(8,3)$							22.86	1.06	21.79	**19.68**	1.60	18.08	**20.16**	2.71	17.45

Multiple hashing: experimental results

Naive comparison per text symbol for 1000 patterns

m	8	16	32	64	128
MBNDM	.5261	.0492	.0504	.0492	.0496
WM$(2,2)$	5.428	5.471	5.476	5.408	5.427
WM$(2,3)$	.7368	.7381	.7381	.7339	.7290
WM$(3,2)$	.4762	.4764	.4809	.4738	.4760
WM$(3,3)$	-	.0413	.0419	.0412	.0413
WM$(4,1)$	4.440	4.450	4.443	4.436	4.446
WM$(4,2)$	**.1092**	.1078	.1076	.1089	.1089
WM$(4,3)$	-	.0239	.0242	.0240	.0241
WM$(6,1)$	3.265	3.233	3.227	3.178	3.124
WM$(6,2)$	-	.0995	.0988	.0978	.0966
WM$(6,3)$	-	-	.0267	.0268	.0271
WM$(8,1)$	.5021	.3258	.2631	.2385	.2247
WM$(8,2)$	-	**.0034**	.0022	.0019	.0018
WM$(8,3)$	-	-	**.0006**	**.0005**	**.0005**