PSC 2024

The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica

PSC 2024

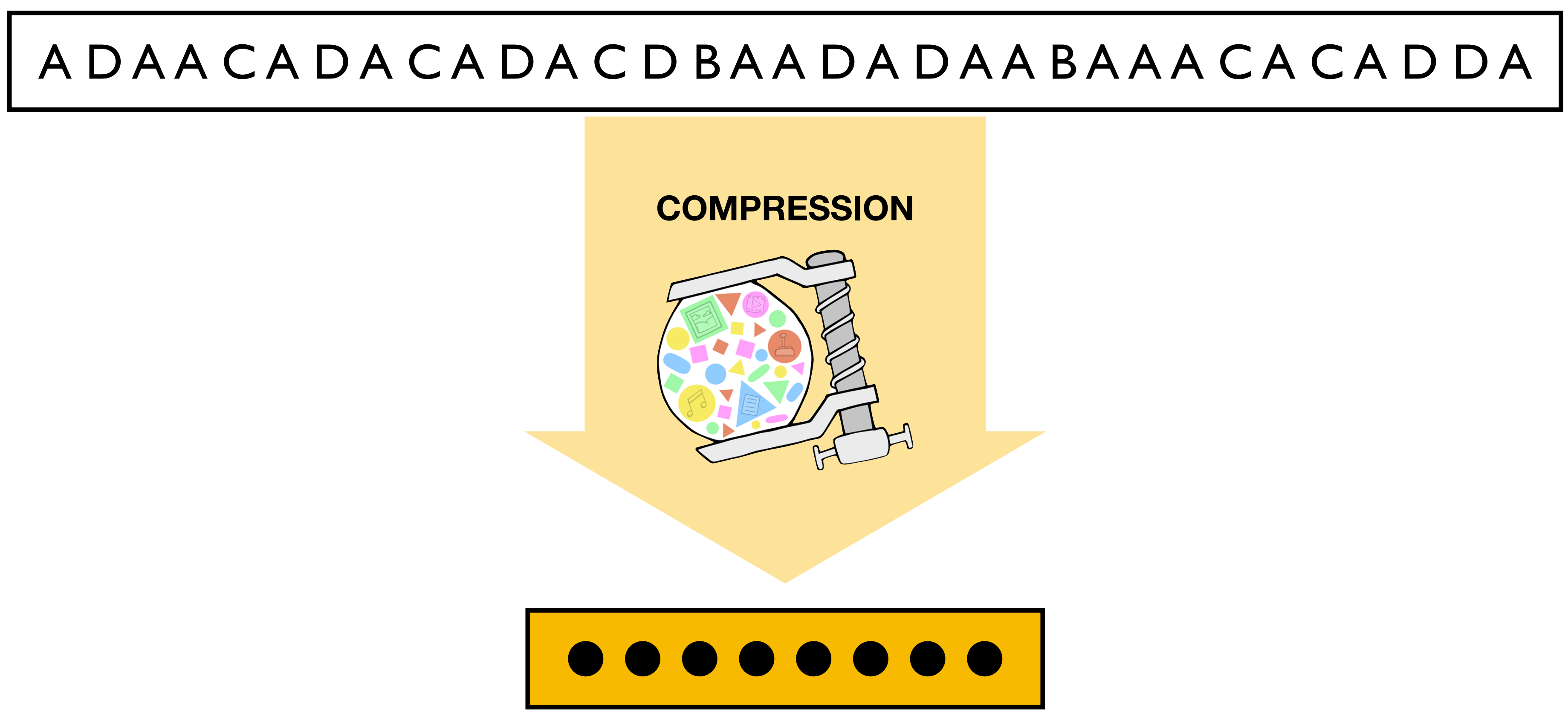
The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica



PSC 2024

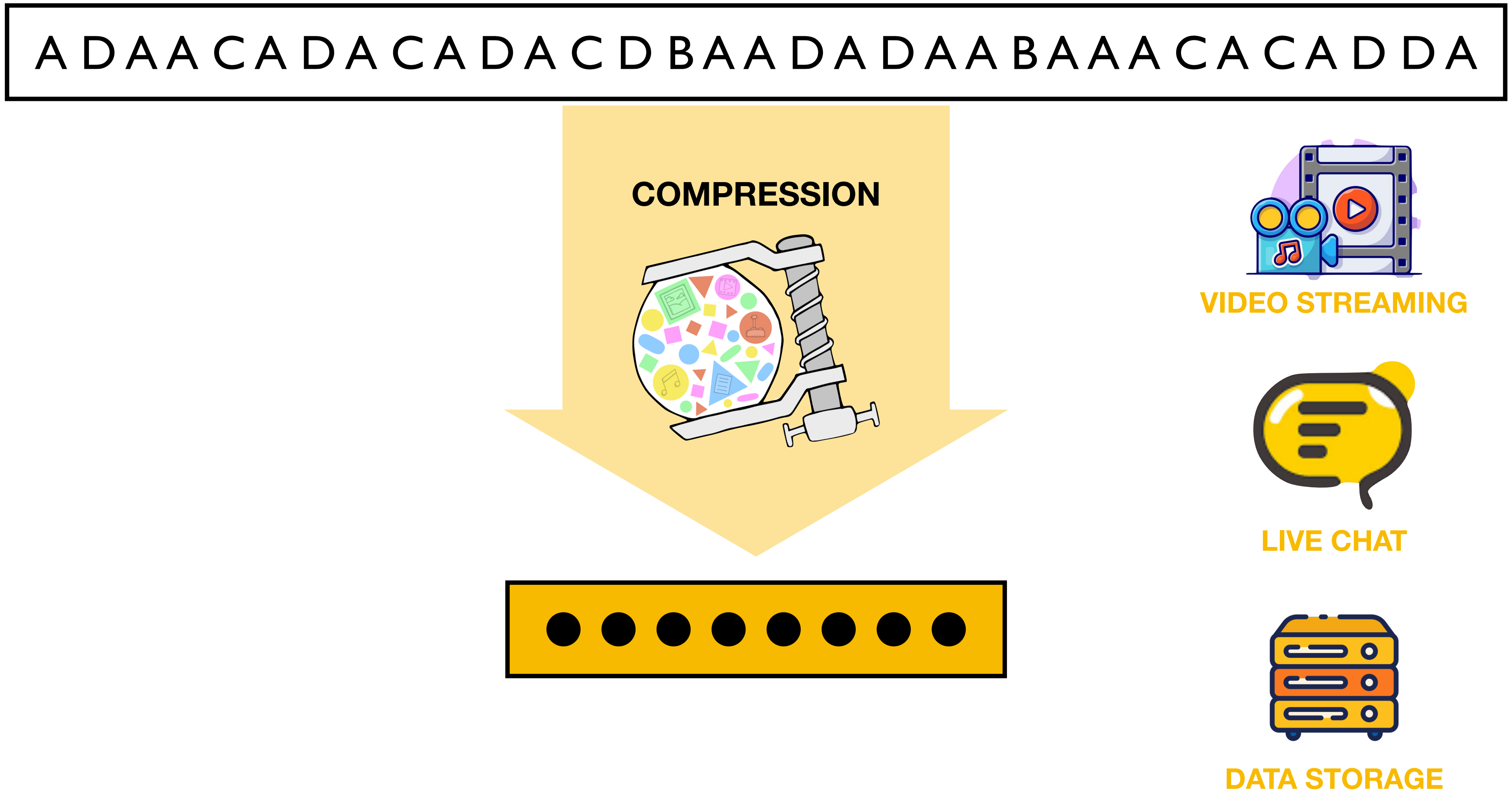
The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica



Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica

A D A A C A D A C A D A C D B A A D A D A A B A A A C A C A D D A

```
A: 0 0
B: 0 1
C: 1 0
D: 1 1
```

● ● ● ● ● ● ● ●

PSC 2024

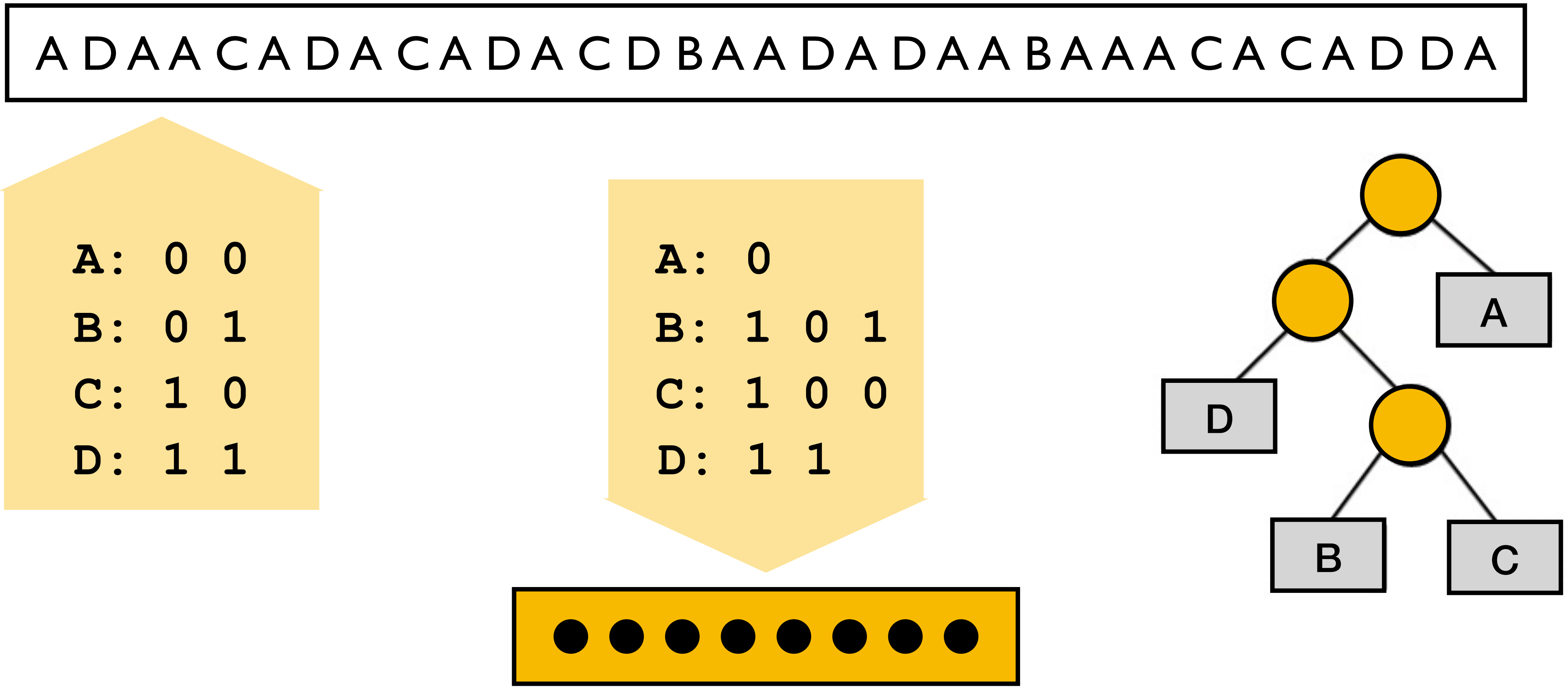
The Praga
Stringology
Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica



PSC 2024

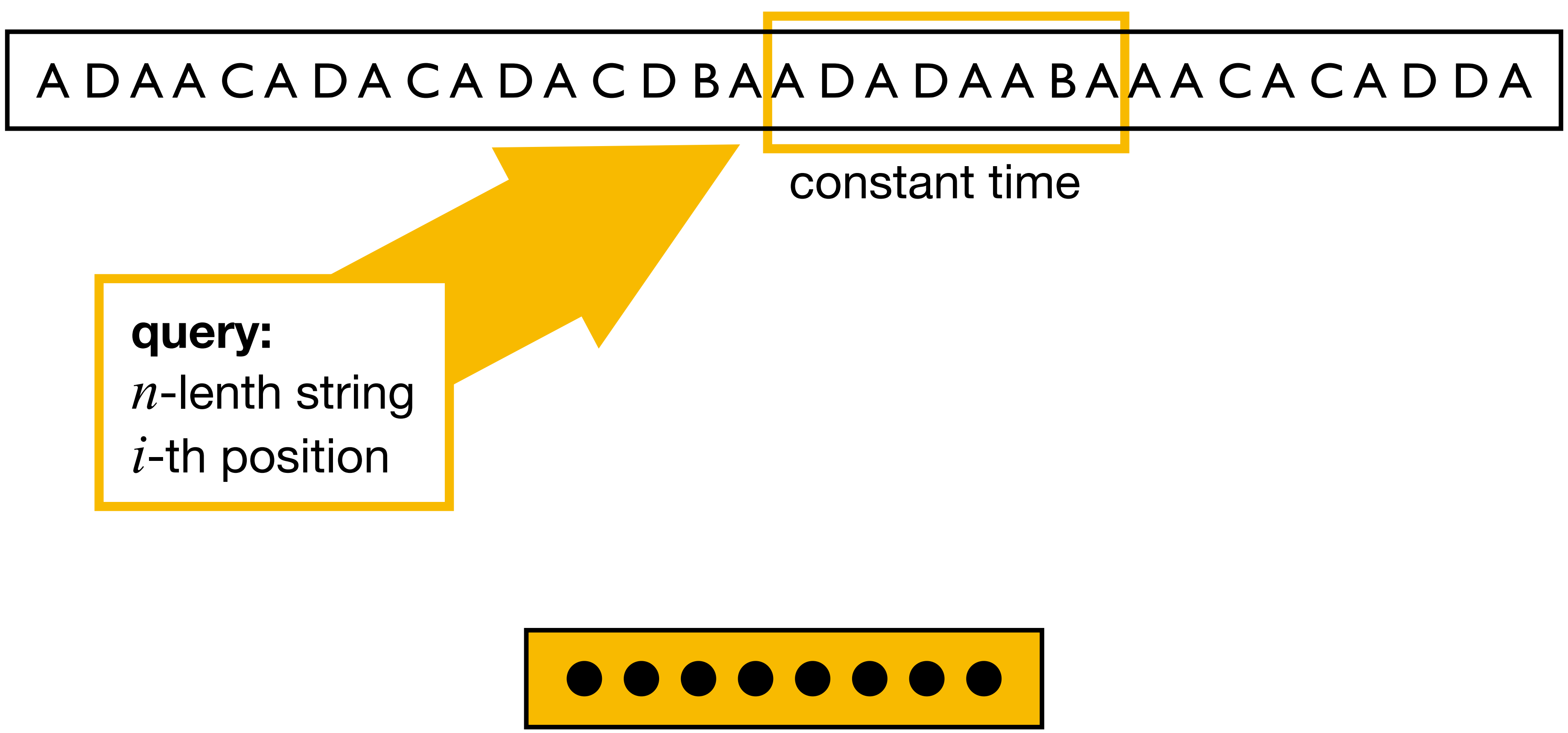
The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica



PSC 2024

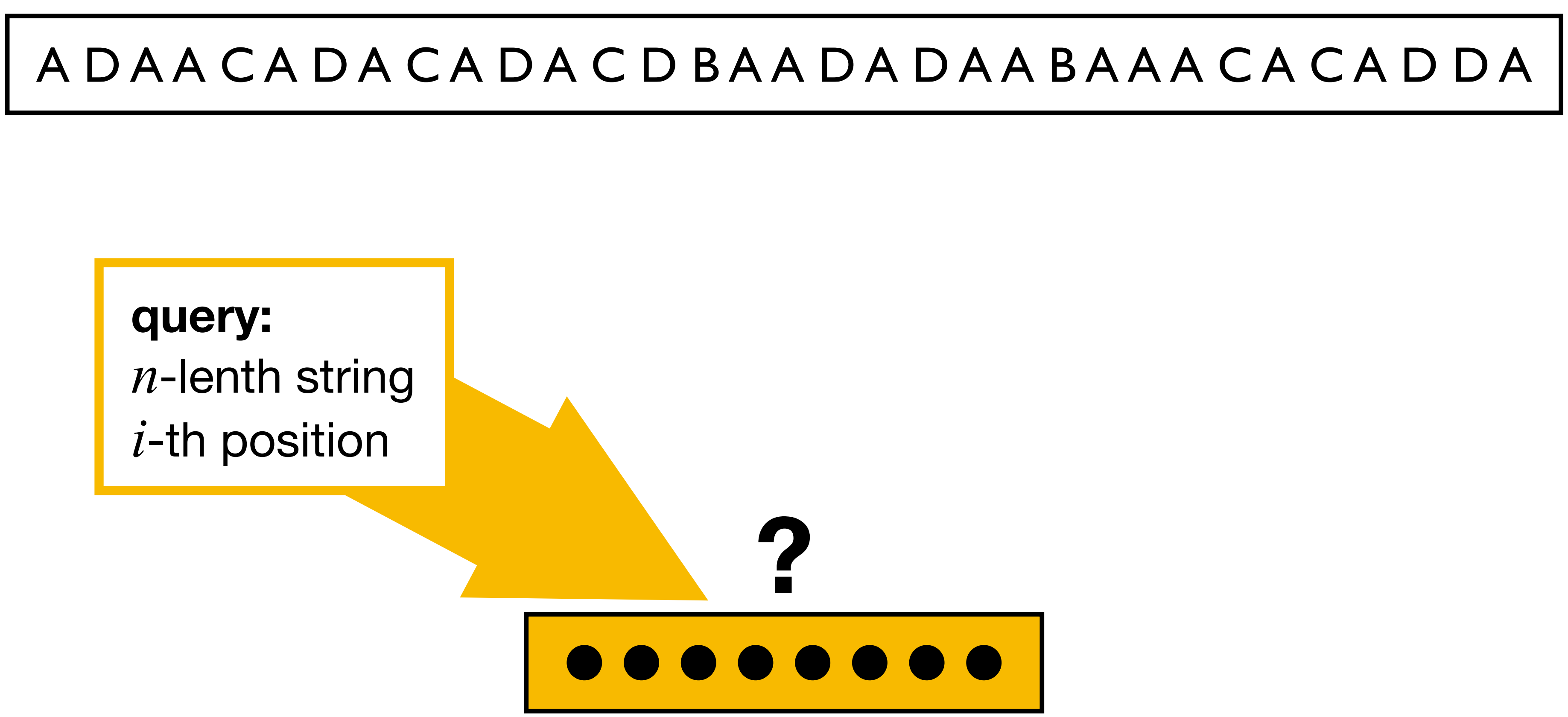
The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica



PSC 2024

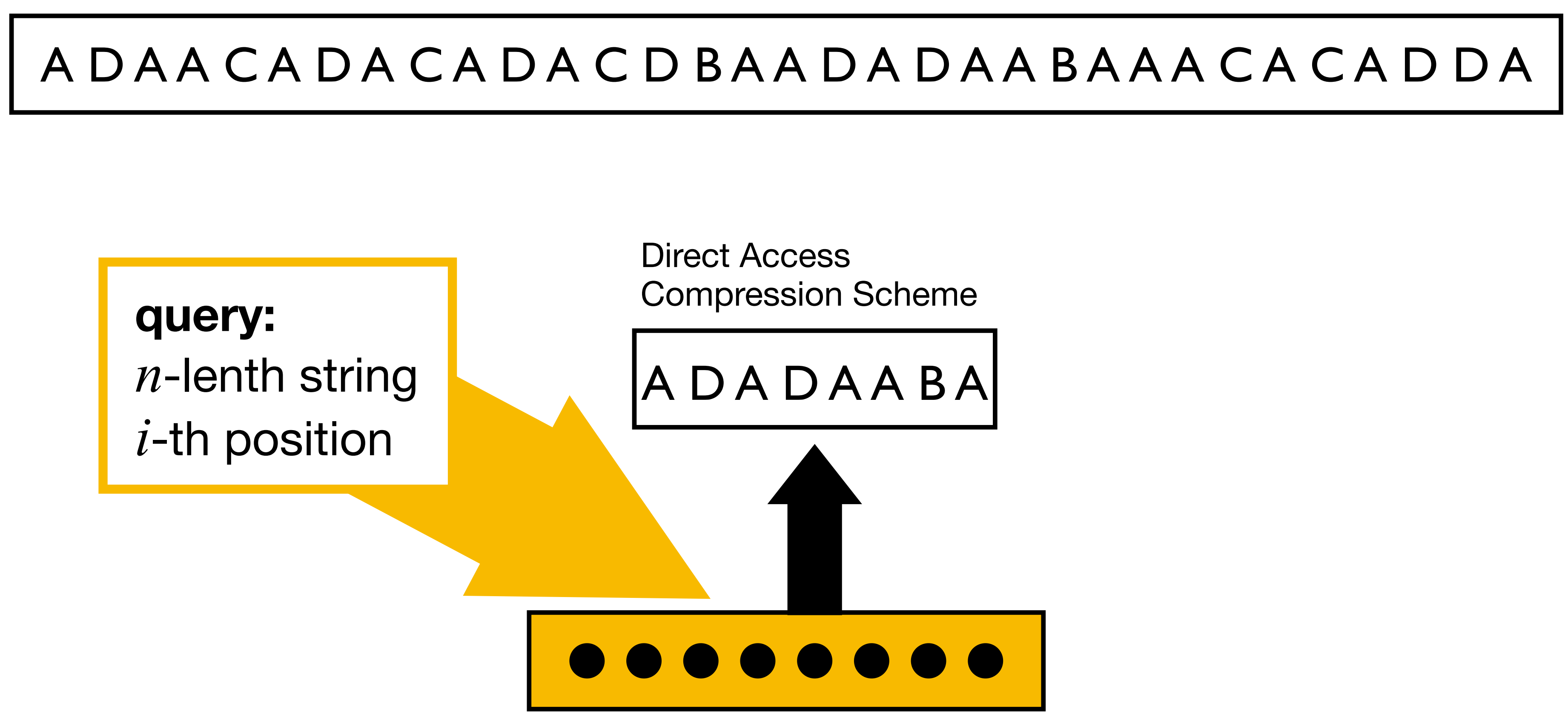
The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica



PSC 2024

The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica

Method	Overall Space	Access to $y[i]$
Sparse Sampling	$N + \lceil n/h \rceil \lceil \log(N) \rceil$	$\mathcal{O}(h\rho_{max})$
Dense Sampling	$N + n(\log\log N + \log\log\sigma)$	$\mathcal{O}(\lvert\rho(y[i])\rvert)$
Interpolative Coding	$N + \mathcal{O}(n\log(N)/\log(n))$	$\mathcal{O}(\log n)$
Wavelet Tree	$N + o(N)$	$\mathcal{O}(\lvert\rho(y[i])\rvert)$
DACs	$\mathcal{O}((N\log\log N)/(\sqrt{N_0/n}\log N) + \log\sigma)$	$\mathcal{O}(N/(n(\sqrt{N_0/n})))$
SFDC	$N + \mathcal{O}(n)$	$\mathcal{O}(\lvert\rho(y[i])\rvert)$ Expected

PSC 2024

The Praga Stringology Conference

Prague
Czech Republic
August 26-27, 2024

Refining SFDC Compression Scheme with Block Text Segmentation

Simone Faro and Alfio Spoto

Università di Catania, Dipartimento di Matematica e Informatica

Abstract. The Succinct Format with Direct Accessibility (SFDC) is an encoding scheme originally designed for efficient data compression and quick access to elements within compressed sequences. While SFDC performs well under stable character frequency conditions, its efficacy diminishes in text corpora with high variability in character frequencies, typical of natural language environments. Addressing this limitation, this paper presents three variant of SFDC based on block segmentation methods, each offering unique enhancements over the original SFDC representation. By tailoring the segmentation process to the distribution of characters within the text, these methods aim to optimize compression efficiency and decoding performance. The paper presents experimental results demonstrating the effectiveness of these approaches, highlighting their ability to improve upon the original scheme in several scenarios. The findings underscore the potential of these advanced segmentation strategies to provide superior compression and performance across a range of text datasets.

Succint Format with Direct Accessibility

SFDC (Succinct Format with Direct aCcesibility) is based on variable-length codes obtained from existing compression methods. For presentation purposes, in this paper we show how to construct our SFDC from Huffman codes.

Succint Format with Direct Accessibility

SFDC (Succinct Format with Direct aCcesibility) is based on variable-length codes obtained from existing compression methods. For presentation purposes, in this paper we show how to construct our SFDC from Huffman codes.

The SFDC encoding is relevant for the following reasons:

– it allows direct access to text characters in (expected) constant time;

Fast
Direct
Access

Succint Format with Direct Accessibility

SFDC (Succinct Format with Direct aCcesibility) is based on variable-length codes obtained from existing compression methods. For presentation purposes, in this paper we show how to construct our SFDC from Huffman codes.

The SFDC encoding is relevant for the following reasons:

- it allows direct access to text characters in (expected) constant time;
- it achieves compression ratios that, under suitable conditions, are superior to other solutions;

Fast Direct Access

Good Compression Ratios

Succint Format with Direct Accessibility

SFDC (Succinct Format with Direct aCcesibility) is based on variable-length codes obtained from existing compression methods. For presentation purposes, in this paper we show how to construct our SFDC from Huffman codes.

The SFDC encoding is relevant for the following reasons:

- it allows direct access to text characters in (expected) constant time;
- it achieves compression ratios that, under suitable conditions, are superior to other solutions;
- it offers a flexible representation that can be adapted to efficiency or to space consumption;

Fast Direct Access

Good Compression Ratios

Flexibility And Adaptability

Succint Format with Direct Accessibility

SFDC (Succinct Format with Direct aCcesibility) is based on variable-length codes obtained from existing compression methods. For presentation purposes, in this paper we show how to construct our SFDC from Huffman codes.

The SFDC encoding is relevant for the following reasons:

– it allows direct access to text characters in (expected) constant time;
– it achieves compression ratios that, under suitable conditions, are superior to other solutions;
– it offers a flexible representation that can be adapted to efficiency or to space consumption;
– it is designed to allow parallel and adaptive access to multiple data and parallel-computation;

Fast Direct Access

Good Compression Ratios

Flexibility And Adaptability

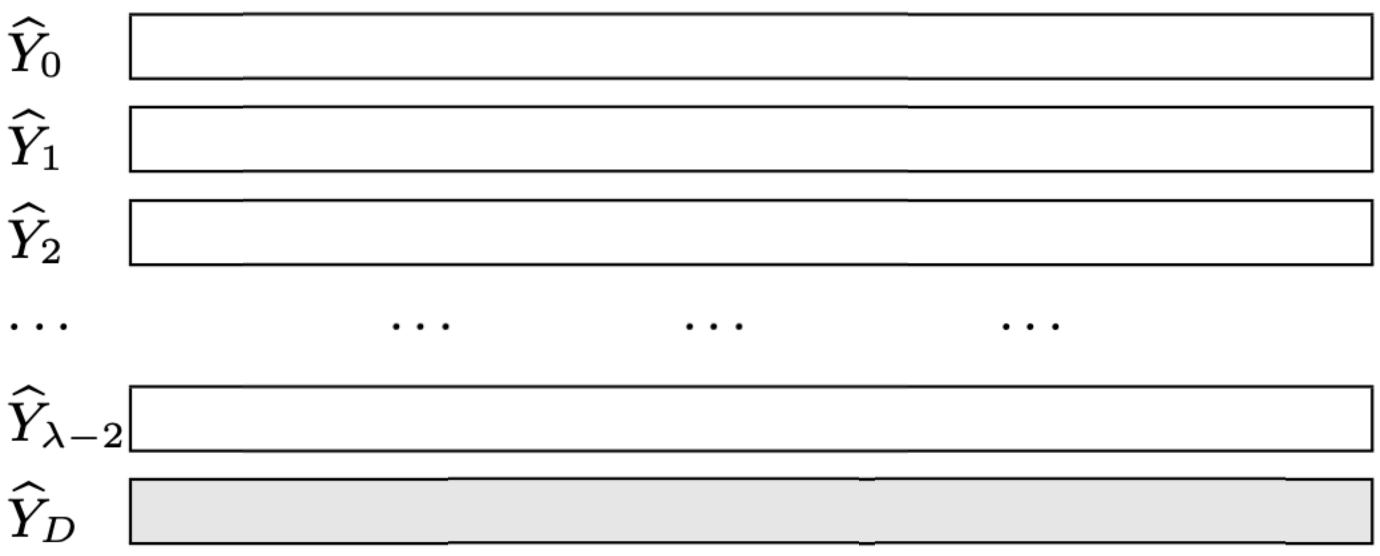
Computational Friendly Scheme

Succint Format with Direct Accessibility

The SFDC codes any string y of length n as an ordered collection of λ binary strings representing $\lambda - 1$ *fixed layers* and an additional *dynamic layer*.

The first $\lambda - 1$ binary strings have length n; we denote them by $\widehat{Y}_0, \widehat{Y}_1, \ldots, \widehat{Y}_{\lambda-2}$. Specifically, the i-th binary string $\widehat{Y}_i$ is the sequence of the i-th bits (if present, 0 otherwise) of the encodings of the characters in y, in the order in which they appear in y.

$$\widehat{Y}_i := \langle \rho(y[0])[i],\ \rho(y[1])[i],\ \ldots,\ \rho(y[n-1])[i] \rangle,$$

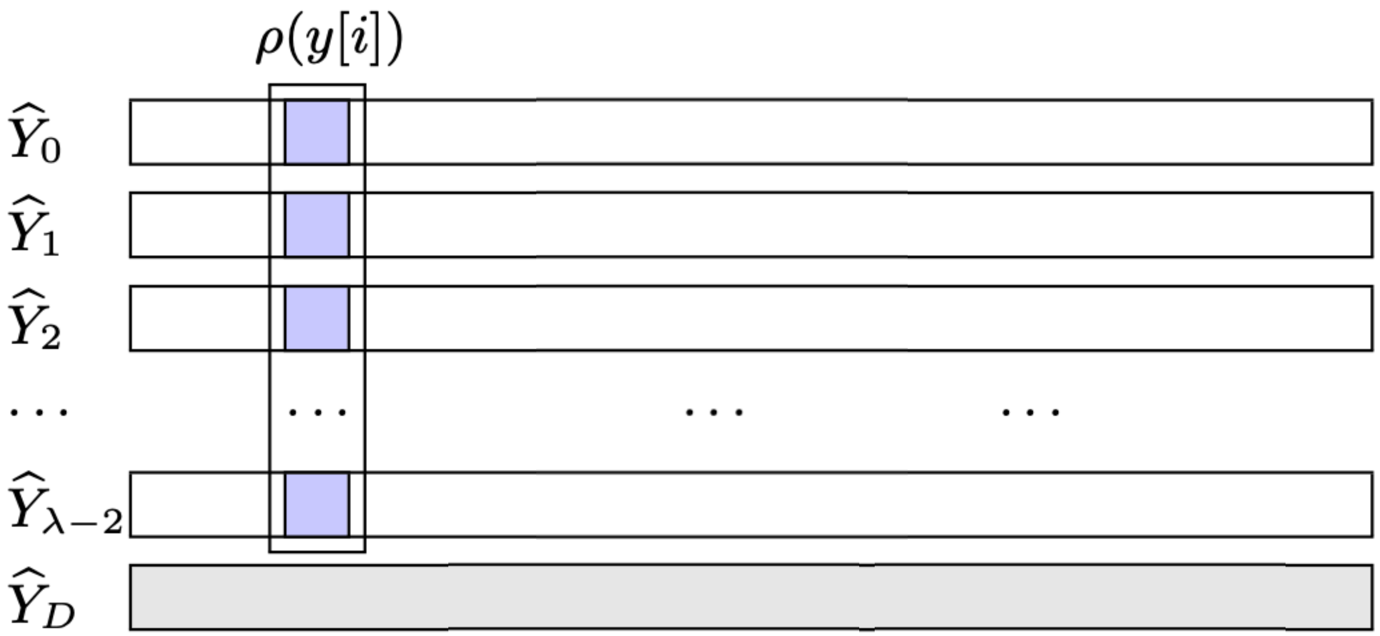


Succint Format with Direct Accessibility

The SFDC codes any string y of length n as an ordered collection of λ binary strings representing $\lambda - 1$ *fixed layers* and an additional *dynamic layer*.

The first $\lambda - 1$ binary strings have length n; we denote them by $\widehat{Y}_0, \widehat{Y}_1, \ldots, \widehat{Y}_{\lambda-2}$. Specifically, the i-th binary string $\widehat{Y}_i$ is the sequence of the i-th bits (if present, 0 otherwise) of the encodings of the characters in y, in the order in which they appear in y.

$$\widehat{Y}_i := \langle \rho(y[0])[i], \rho(y[1])[i], \ldots, \rho(y[n-1])[i] \rangle,$$

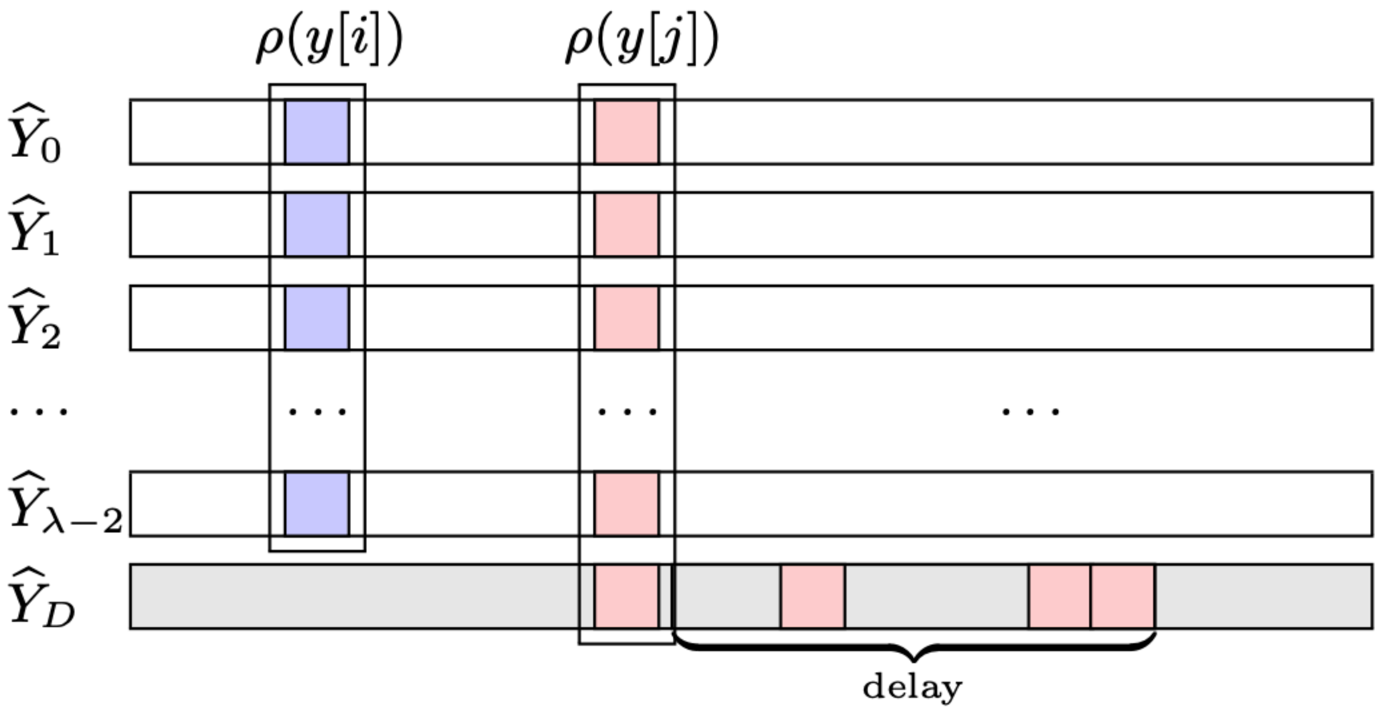


Succint Format with Direct Accessibility

The SFDC codes any string y of length n as an ordered collection of λ binary strings representing $\lambda - 1$ *fixed layers* and an additional *dynamic layer*.

The first $\lambda - 1$ binary strings have length n; we denote them by $\widehat{Y}_0, \widehat{Y}_1, \ldots, \widehat{Y}_{\lambda-2}$. Specifically, the i-th binary string $\widehat{Y}_i$ is the sequence of the i-th bits (if present, 0 otherwise) of the encodings of the characters in y, in the order in which they appear in y.

$$\widehat{Y}_i \coloneqq \langle \rho(y[0])[i],\ \rho(y[1])[i],\ \ldots,\ \rho(y[n-1])[i] \rangle,$$



Succint Format with Direct Accessibility

char	code	length
s	001	3
e	01	2
n	010	3
p	0110101	7
m	101	3
C	1100011010	10
o	1100111	7
i	11010	5
r	11101	5
_	00001	5

C o m p r e s s i o n

1100011010 · 1100111 · 101 · 0110101 · 11101 · 01 · 001 · 001 · 11010 · 1100111 · 010

Succint Format with Direct Accessibility

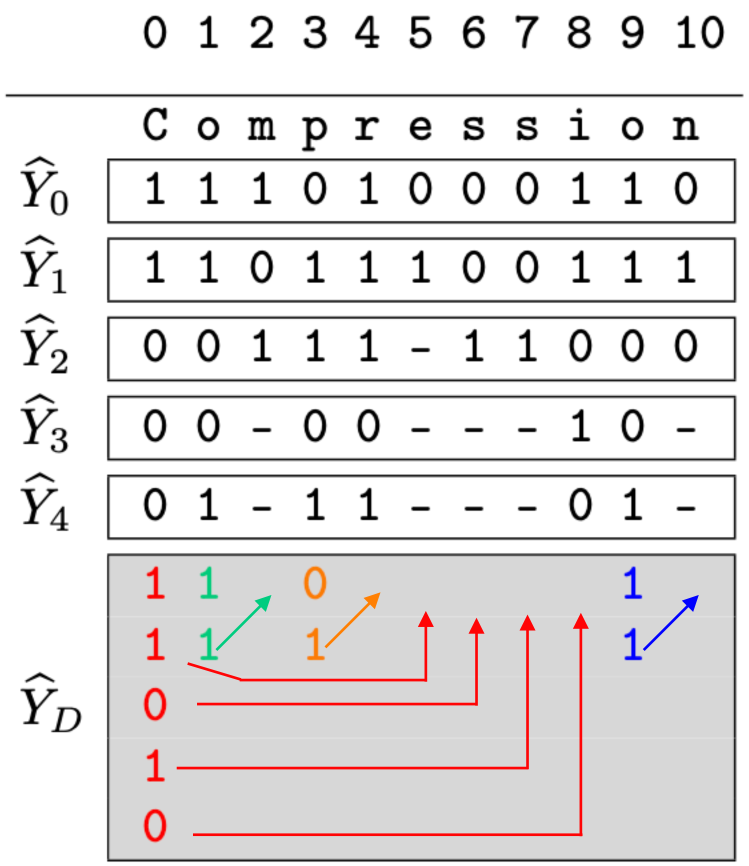
1100011010 · 1100111 · 101 · 0110101 · 11101 · 01 · 001 · 001 · 11010 · 1100111 · 010

char	code	length
s	001	3
e	01	2
n	010	3
p	0110101	7
m	101	3
C	1100011010	10
o	1100111	7
i	11010	5
r	11101	5
_	00001	5

	0	1	2	3	4	5	6	7	8	9	10
	C	o	m	p	r	e	s	s	i	o	n
$\widehat{Y}_0$	1	1	1	0	1	0	0	0	1	1	0
$\widehat{Y}_1$	1	1	0	1	1	1	0	0	1	1	1
$\widehat{Y}_2$	0	0	1	1	1	-	1	1	0	0	0
$\widehat{Y}_3$	0	0	-	0	0	-	-	-	1	0	-
$\widehat{Y}_4$	0	1	-	1	1	-	-	-	0	1	-
$\widehat{Y}_D$	1	1		0						1	
	1	1		1						1	
	0										
	1										
	0										

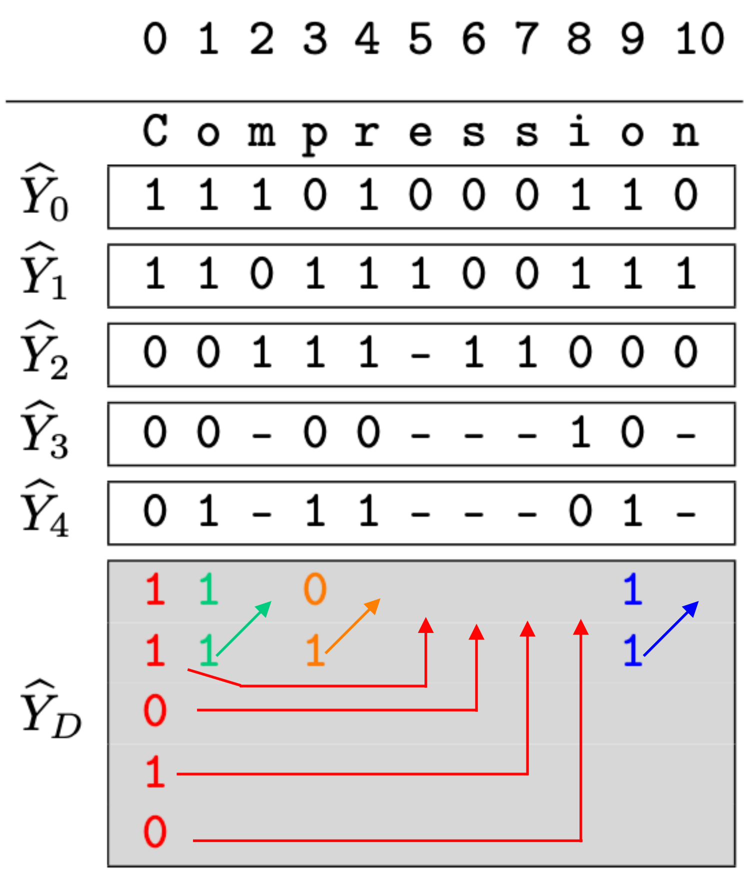
Succint Format with Direct Accessibility

char	code	length
s	001	3
e	01	2
n	010	3
p	0110101	7
m	101	3
C	1100011010	10
o	1100111	7
i	11010	5
r	11101	5
_	00001	5



Succint Format with Direct Accessibility

char	code	length
s	001	3
e	01	2
n	010	3
p	0110101	7
m	101	3
C	1100011010	10
o	1100111	7
i	11010	5
r	11101	5
_	00001	5



	0	1	2	3	4	5	6	7	8	9	10
	C	o	m	p	r	e	s	s	i	o	n
$\widehat{Y}_0$	1	1	1	0	1	0	0	0	1	1	0
$\widehat{Y}_1$	1	1	0	1	1	1	0	0	1	1	1
$\widehat{Y}_2$	0	0	1	1	1	-	1	1	0	0	0
$\widehat{Y}_3$	0	0	-	0	0	-	-	-	1	0	-
$\widehat{Y}_4$	0	1	-	1	1	-	-	-	0	1	-
$\widehat{Y}_D$	1	1	1	0	1	1	0	1	0	1	1

Succint Format with Direct Accessibility

Idle Bits (Theoretical)

$\lambda \setminus \sigma$	10	20	30
5	2.42	2.38	2.38
6	3.42	3.38	3.38
7	4.42	4.38	4.38
8	5.42	5.38	5.38

$$\lambda - (F_{\sigma+3} - 3)/F_{\sigma+1}$$

Idle Bits (Experimental)

$\lambda \setminus \sigma$	10	20	30
5	2.29	2.26	2.27
6	3.29	3.27	3.27
7	4.30	4.29	4.29
8	5.30	5.31	5.31

Average Delay (Theoretical)

$\lambda \setminus \sigma$	10	15	20	25	30
5	0.20	0.23	0.24	0.24	0.24
6	0.11	0.14	0.15	0.15	0.15
7	0.06	0.09	0.09	0.09	0.09
8	0.02	0.05	0.06	0.06	0.06

$$\frac{F_{\sigma-\lambda+3} - 3}{F_{\sigma+1}}$$

Average Delay (Experimental)

$\lambda \setminus \sigma$	10	15	20	25	30
5	0.31	0.37	0.37	0.39	0.39
6	0.15	0.20	0.18	0.17	0.21
7	0.07	0.11	0.11	0.11	0.11
8	0.02	0.06	0.06	0.07	0.06

Succint Format with Direct Accessibility

Text	σ	Max$\{\|\rho(y[i])\|\}$	Avg$\{\|\rho(y[i])\|\}$
PROTEIN	25	11	4.22
DBLP	96	21	5.26
ENGLISH	94	20	4.59

	Text	WT	DACs	SFDC			
	λ			5	6	7	8
PROTEIN	Space	6.16	6.45	5.00	6.00	7.00	8.00
	Decode	5.47	0.86	1.24	1.11	1.17	1.26
	Access	0.95	0.07	0.83	0.73	0.72	0.74
	Delay	-	-	1.05	0.51	0.29	0.12
DBLP	Space	7.68	7.23	5.26	6.00	7.00	8.00
	Decode	5.87	0.93	1.66	1.42	1.39	1.45
	Access	1.05	0.07	-	0.79	0.77	0.74
	Delay	-	-	-	2.19	0.41	0.12
ENGLISH	Space	6.72	7.42	5.00	6.00	7.00	8.00
	Decode	5.62	0.85	1.76	1.53	1.34	1.38
	Access	0.96	0.06	625	134	12.7	4.17
	Delay	-	-	49K	7.2K	870	436

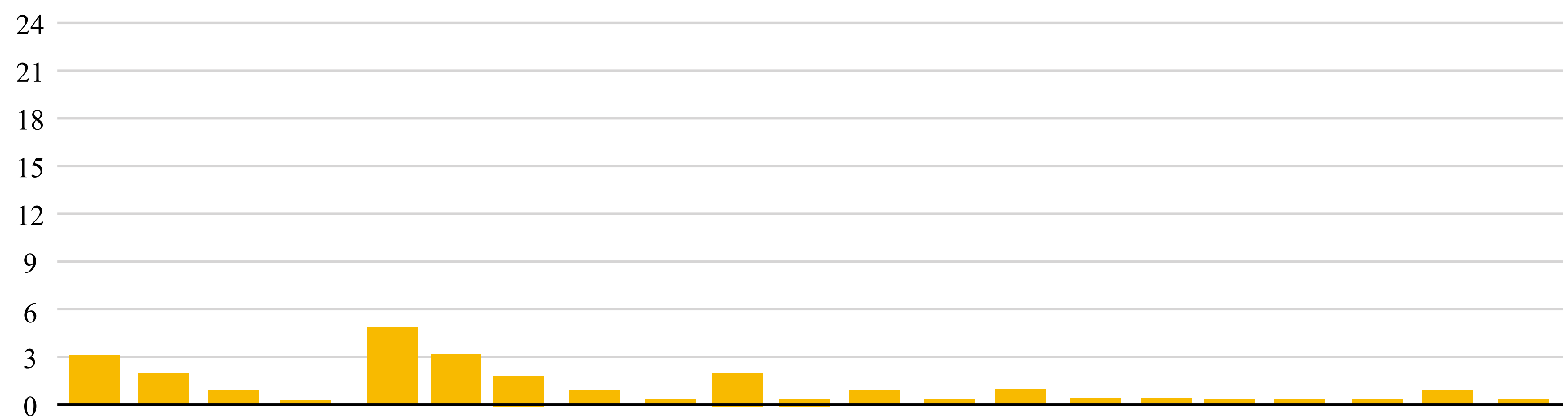
LIFO Delay Amplification

The *LIFO Delay Amplification* (LDA) phenomenon in SFDC refers to the unintended increase in decoding delay for characters appearing in a block that precedes a rare character in the text.

This phenomenon actually occurs because rare characters can induce significant delay to characters preceding them due to their long decoding paths. Such behavior is exacerbated by the LIFO strategy adopted in the SFDC scheme which assigns decoding priority to the rightmost characters in the text.

The *LIFO Delay Amplification* (LDA) phenomenon in SFDC refers to the unintended increase in decoding delay for characters appearing in a block that precedes a rare character in the text.

This phenomenon actually occurs because rare characters can induce significant delay to characters preceding them due to their long decoding paths. Such behavior is exacerbated by the LIFO strategy adopted in the SFDC scheme which assigns decoding priority to the rightmost characters in the text.

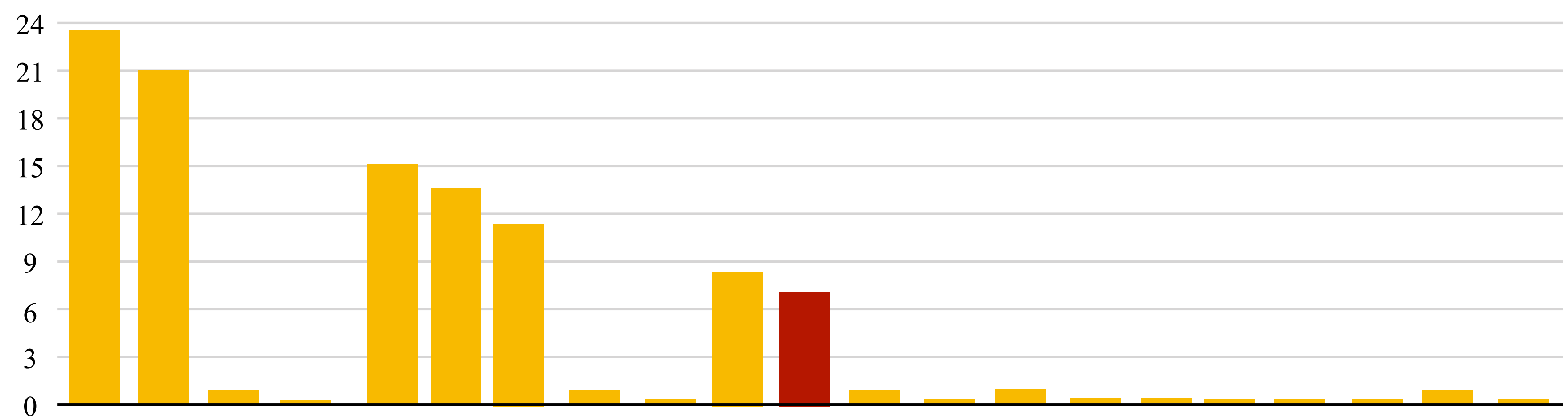


$y[i]$	p	o	o	r	_	p	o	o	r	_	c	o	m	p	r	e	s	s	i	o	n
$\|\rho(y[i])\|$	7	7	7	5	6	7	7	7	5	6	4	7	3	7	5	2	3	3	5	7	3
$d(i)$	3	2	1	0	4	3	2	1	0	3	0	1	0	1	0	0	0	0	0	1	0

LIFO Delay Amplification

The *LIFO Delay Amplification* (LDA) phenomenon in SFDC refers to the unintended increase in decoding delay for characters appearing in a block that precedes a rare character in the text.

This phenomenon actually occurs because rare characters can induce significant delay to characters preceding them due to their long decoding paths. Such behavior is exacerbated by the LIFO strategy adopted in the SFDC scheme which assigns decoding priority to the rightmost characters in the text.

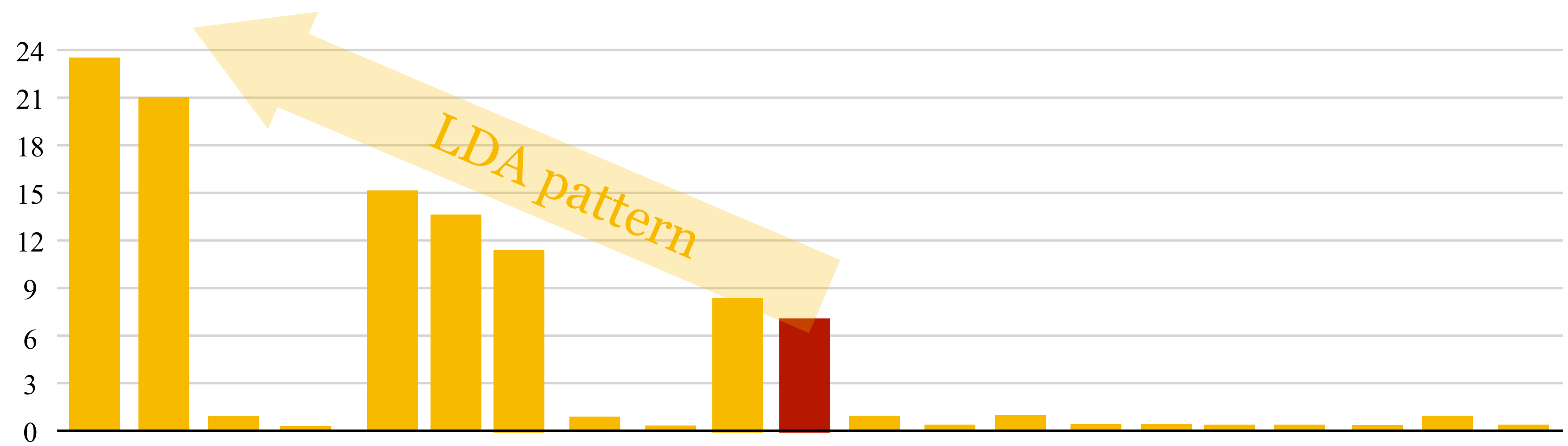


$y[i]$	p	o	o	r	_	p	o	o	r	_	C	o	m	p	r	e	s	s	i	o	n
$\|\rho(y[i])\|$	7	7	7	5	6	7	7	7	5	6	10	7	3	7	5	2	3	3	5	7	3
$d(i)$	23	21	1	0	15	13	11	1	0	8	7	1	0	1	0	0	0	0	0	1	0

LIFO Delay Amplification

The *LIFO Delay Amplification* (LDA) phenomenon in SFDC refers to the unintended increase in decoding delay for characters appearing in a block that precedes a rare character in the text.

This phenomenon actually occurs because rare characters can induce significant delay to characters preceding them due to their long decoding paths. Such behavior is exacerbated by the LIFO strategy adopted in the SFDC scheme which assigns decoding priority to the rightmost characters in the text.



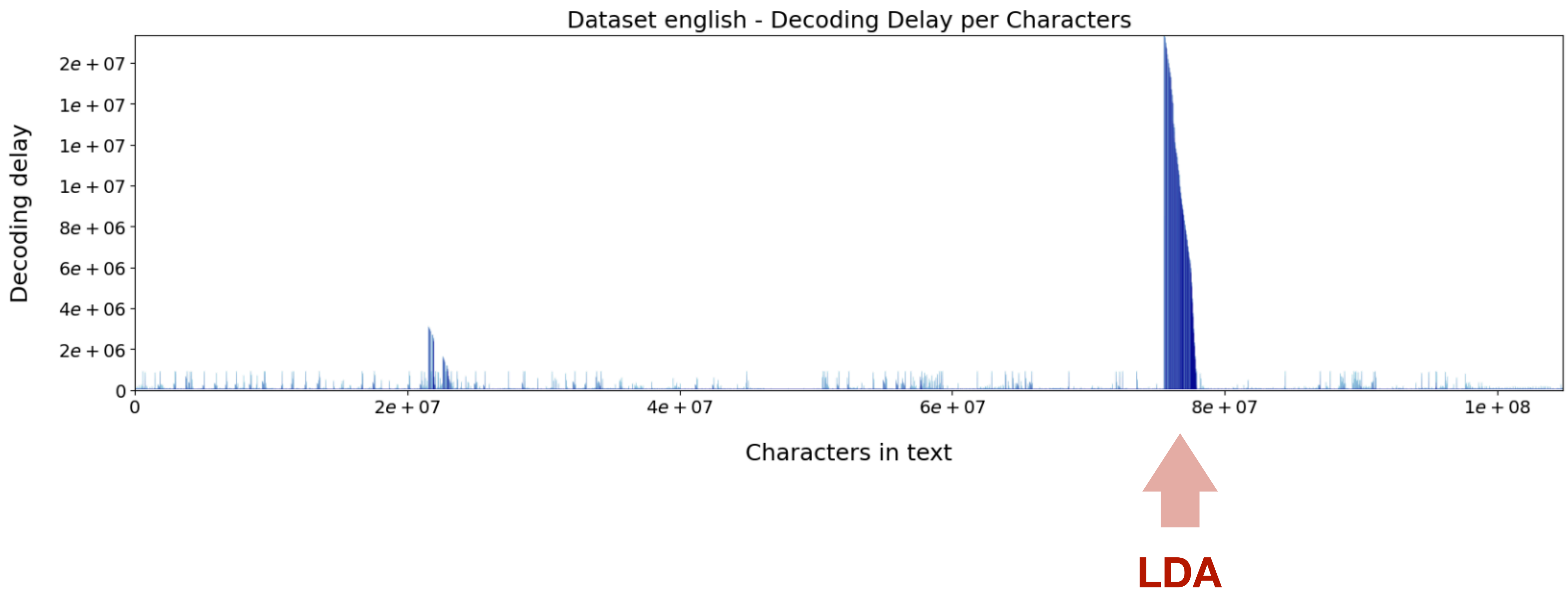
$y[i]$	p	o	o	r	_	p	o	o	r	_	C	o	m	p	r	e	s	s	i	o	n
$\|\rho(y[i])\|$	7	7	7	5	6	7	7	7	5	6	10	7	3	7	5	2	3	3	5	7	3
$d(i)$	23	21	1	0	15	13	11	1	0	8	7	1	0	1	0	0	0	0	0	1	0

LIFO Delay Amplification

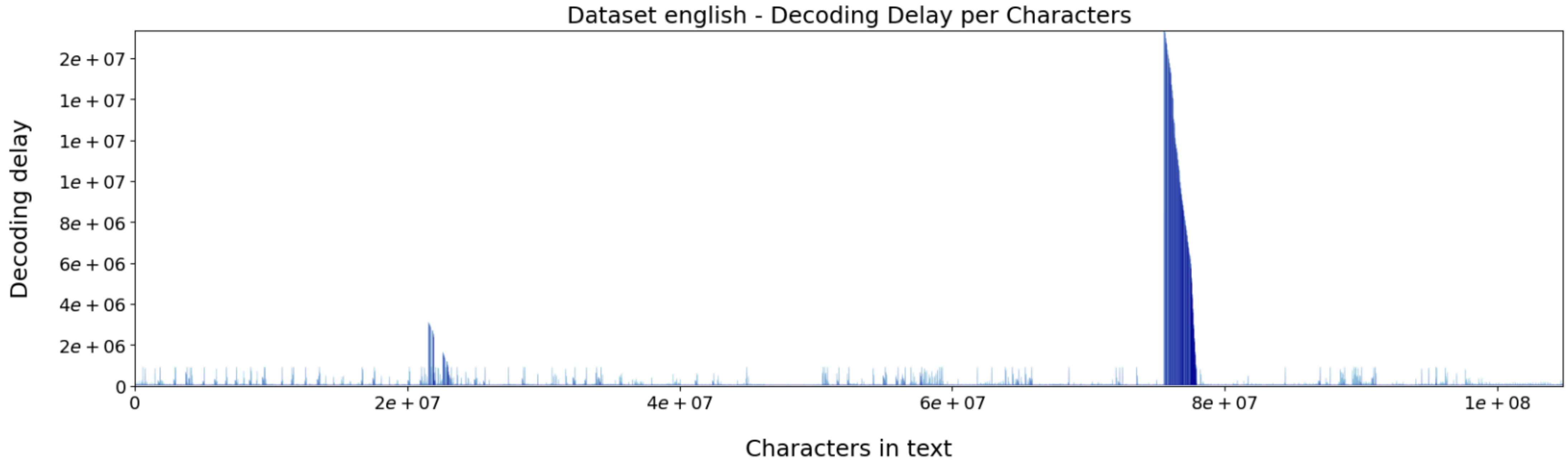
Text	σ	Max$\{\lvert\rho(y[i])\rvert\}$	Avg$\{\lvert\rho(y[i])\rvert\}$
PROTEIN	25	11	4.22
DBLP	96	21	5.26
ENGLISH	94	20	4.59

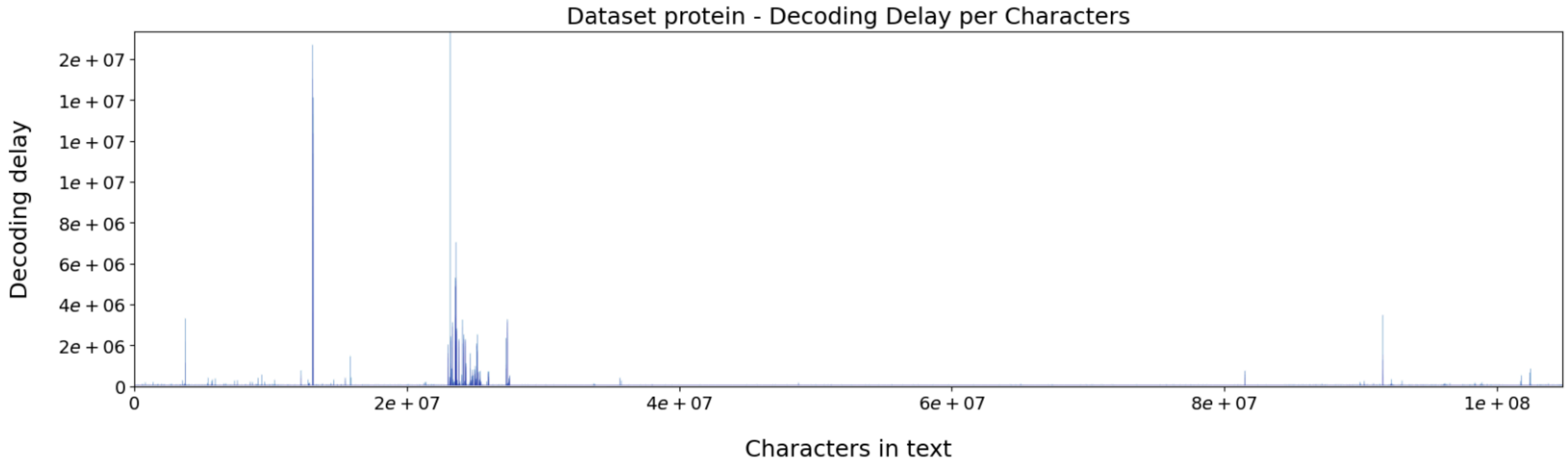
	Text	WT	DACs	SFDC			
	λ			5	6	7	8
PROTEIN	Space	6.16	6.45	5.00	6.00	7.00	8.00
	Decode	5.47	0.86	1.24	1.11	1.17	1.26
	Access	0.95	0.07	0.83	0.73	0.72	0.74
	Delay	-	-	1.05	0.51	0.29	0.12
DBLP	Space	7.68	7.23	5.26	6.00	7.00	8.00
	Decode	5.87	0.93	1.66	1.42	1.39	1.45
	Access	1.05	0.07	-	0.79	0.77	0.74
	Delay	-	-	-	2.19	0.41	0.12
ENGLISH	Space	6.72	7.42	5.00	6.00	7.00	8.00
	Decode	5.62	0.85	1.76	1.53	1.34	1.38
	Access	0.96	0.06	625	134	12.7	4.17
	Delay	-	-	49K	7.2K	870	436

LIFO Delay Amplification



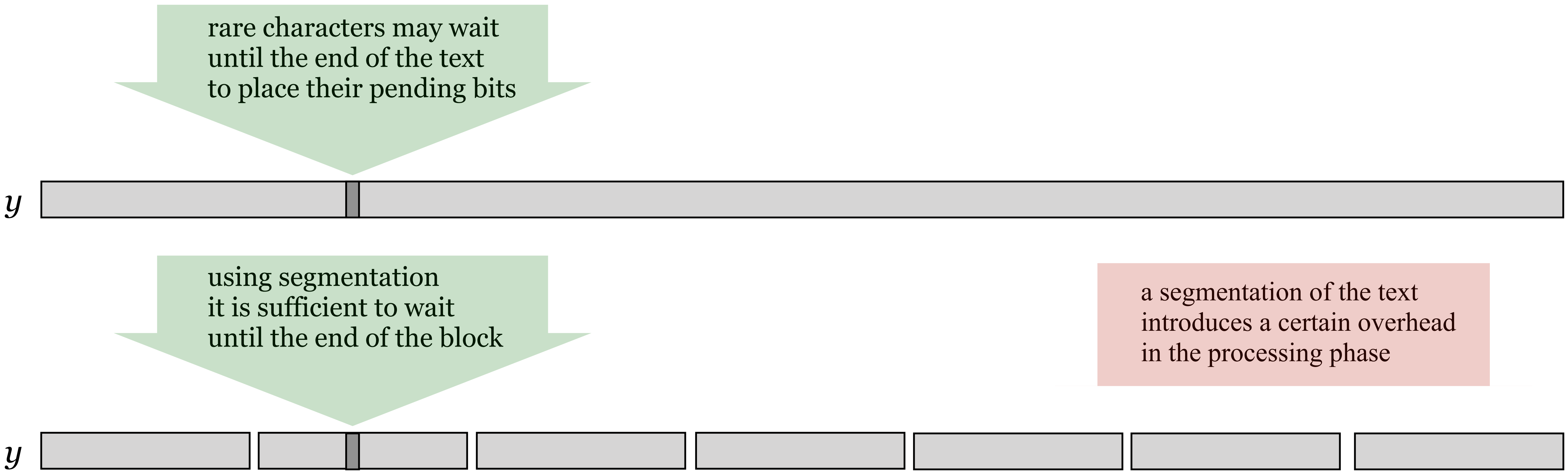
LIFO Delay Amplification





LIFO Delay Amplification

We discuss some approaches based on text segmentation to address the challenges faced by LDA, which partitions the text into smaller blocks and compresses each block separately using the SFDC method. As a general effect, dividing the text into blocks can mitigate the effects of the LDA phenomenon by allowing the pending bits in the stack to be processed in advance. Therefore, closing a block enables the placement of all pending bits, thereby reducing the waiting times for the characters in the stack.



LIFO Delay Amplification

We discuss some approaches based on text segmentation to address the challenges faced by LDA, which partitions the text into smaller blocks and compresses each block separately using the SFDC method. As a general effect, dividing the text into blocks can mitigate the effects of the LDA phenomenon by allowing the pending bits in the stack to be processed in advance. Therefore, closing a block enables the placement of all pending bits, thereby reducing the waiting times for the characters in the stack.

We evaluate the following three primary segmentation strategies:

- Fixed Length Block Segmentation
- Adaptive Huffman Encoding in Fixed Length Block Segmentation
- Rare Markers Block Segmentation

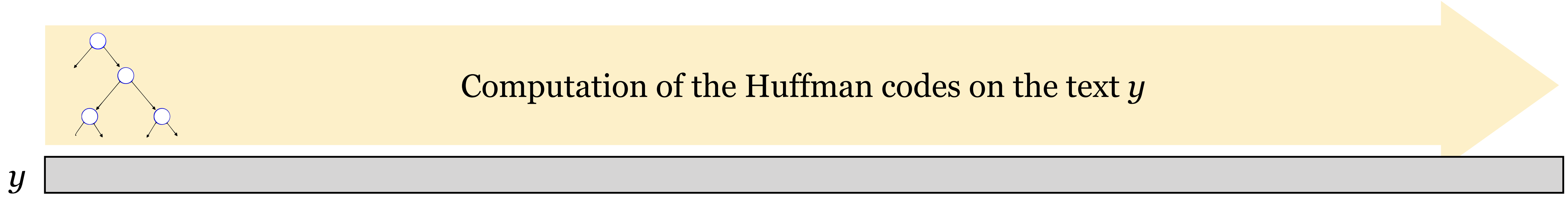
Fixed Length Block Segmentation

The Fixed Length Block (FLB) is a segmentation strategy designed to divide text into blocks of a fixed length. This approach employs a single Huffman tree that is constructed over the entire dataset to define the codeword set used across all the blocks. By referncing this single Huffman tree, the encoding process remains consistent throughout the entire text.

y 

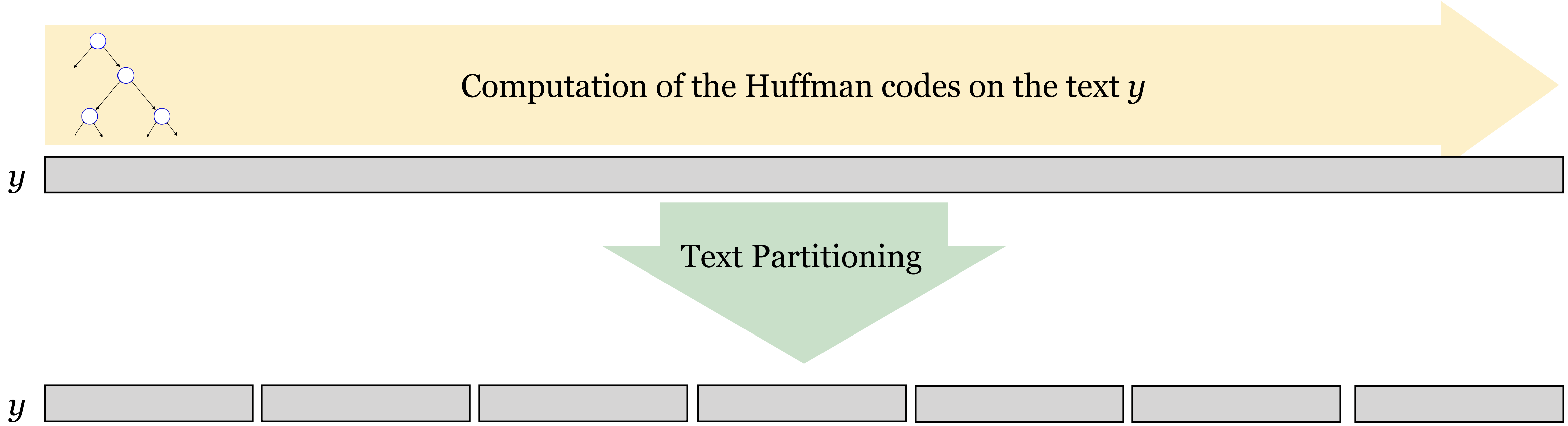
Fixed Length Block Segmentation

The Fixed Length Block (FLB) is a segmentation strategy designed to divide text into blocks of a fixed length. This approach employs a single Huffman tree that is constructed over the entire dataset to define the codeword set used across all the blocks. By referncing this single Huffman tree, the encoding process remains consistent throughout the entire text.



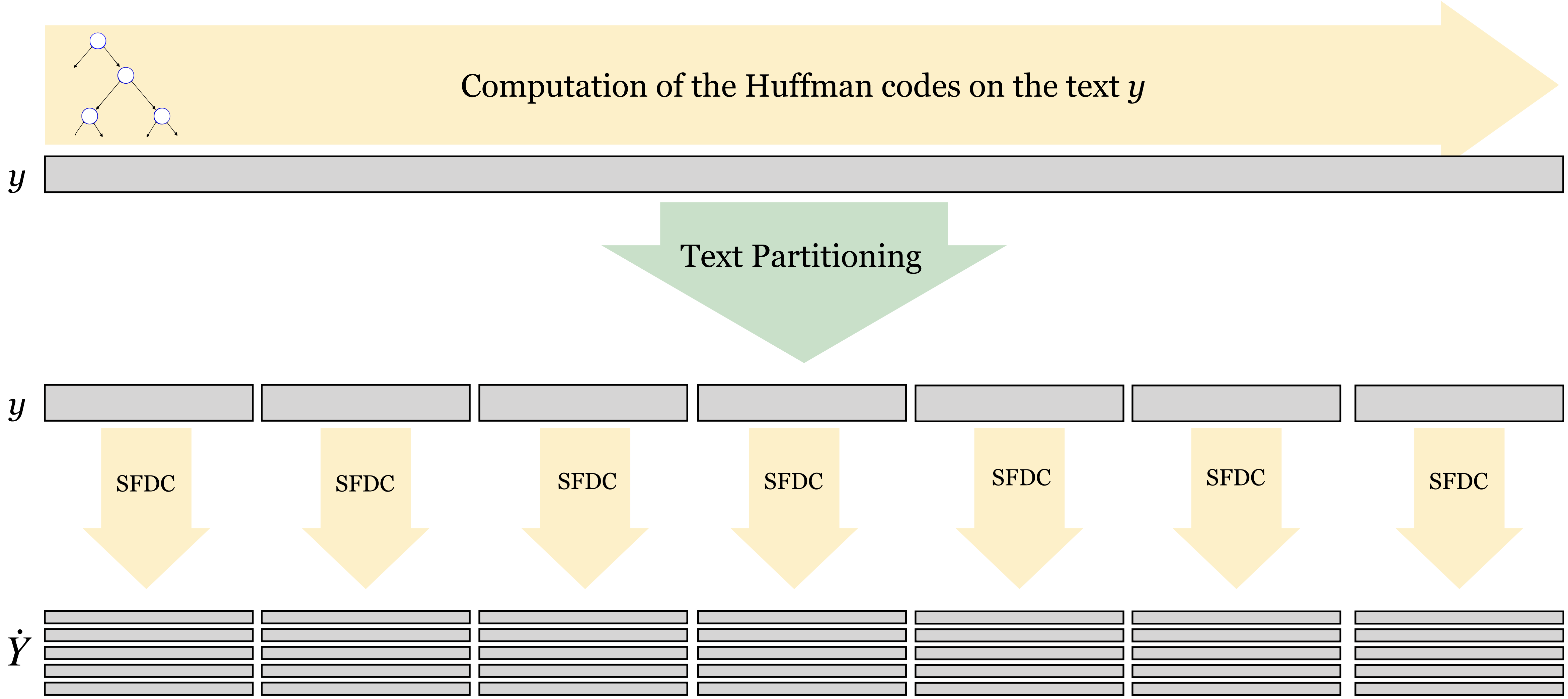
Fixed Length Block Segmentation

The Fixed Length Block (FLB) is a segmentation strategy designed to divide text into blocks of a fixed length. This approach employs a single Huffman tree that is constructed over the entire dataset to define the codeword set used across all the blocks. By referncing this single Huffman tree, the encoding process remains consistent throughout the entire text.



Fixed Length Block Segmentation

The Fixed Length Block (FLB) is a segmentation strategy designed to divide text into blocks of a fixed length. This approach employs a single Huffman tree that is constructed over the entire dataset to define the codeword set used across all the blocks. By referncing this single Huffman tree, the encoding process remains consistent throughout the entire text.



Fixed Length Block Segmentation

PROTEIN

Block Size (in KB)	Average Delay	Number of Blocks
10^0	0.26	104,858
10^1	0.45	10,486
10^2	0.94	1,049
10^3	1.02	105
10^4	1.01	11
10^5	1.06	2

SFDC Avg Delay: 1.02

Fixed Length Block Segmentation

PROTEIN

Block Size (in KB)	Average Delay	Number of Blocks
10^0	0.26	104,858
10^1	0.45	10,486
10^2	0.94	1,049
10^3	1.02	105
10^4	1.01	11
10^5	1.06	2

SFDC Avg Delay: 1.02

DBLP

Block Size (in KB)	Average Delay	Number of Blocks
10^0	2.07	104,858
10^1	2.15	10,486
10^2	2.18	1,049
10^3	2.17	105
10^4	2.16	11
10^5	2.84	2

SFDC Avg Delay: 2.19

Fixed Length Block Segmentation

PROTEIN

Block Size (in KB)	Average Delay	Number of Blocks
10^0	0.26	104,858
10^1	0.45	10,486
10^2	0.94	1,049
10^3	1.02	105
10^4	1.01	11
10^5	1.06	2

SFDC Avg Delay: 1.02

DBLP

Block Size (in KB)	Average Delay	Number of Blocks
10^0	2.07	104,858
10^1	2.15	10,486
10^2	2.18	1,049
10^3	2.17	105
10^4	2.16	11
10^5	2.84	2

SFDC Avg Delay: 2.19

ENGLISH

Block Size (in KB)	Average Delay	Number of Blocks
10^0	10.06	104,858
10^1	63.08	10,486
10^2	502.03	1,049
10^3	2,852.33	105
10^4	16,957.22	11
10^5	59,829.62	2

SFDC Avg Delay: 44,387.30

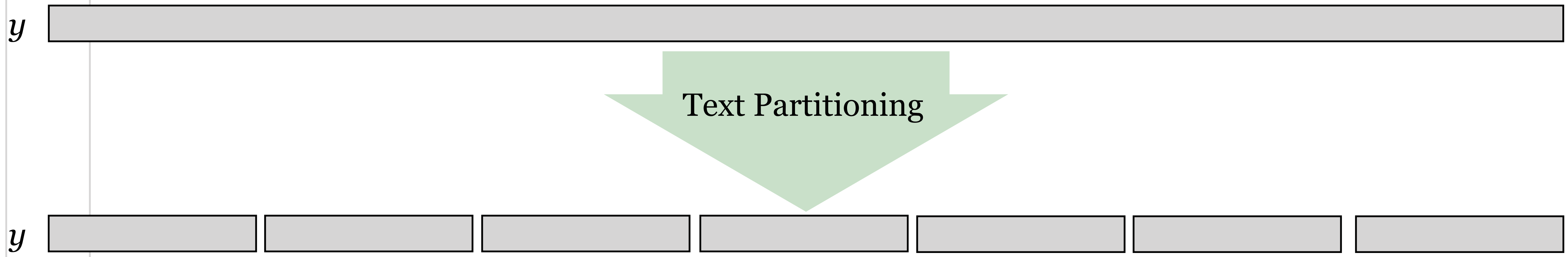
Adaptive Huffman Encoding in FLB Segmentation

The idea of Adaptive Huffman Encoding in FLB Segmentation is to create a new Huffman tree for each block obtained from the segmentation of the text. This strategy ensures that the frequency function used for tree construction more accurately reflects the character frequencies within that specific block, thereby enabling more efficient character encoding and consequently reducing the average delay within the block.

y

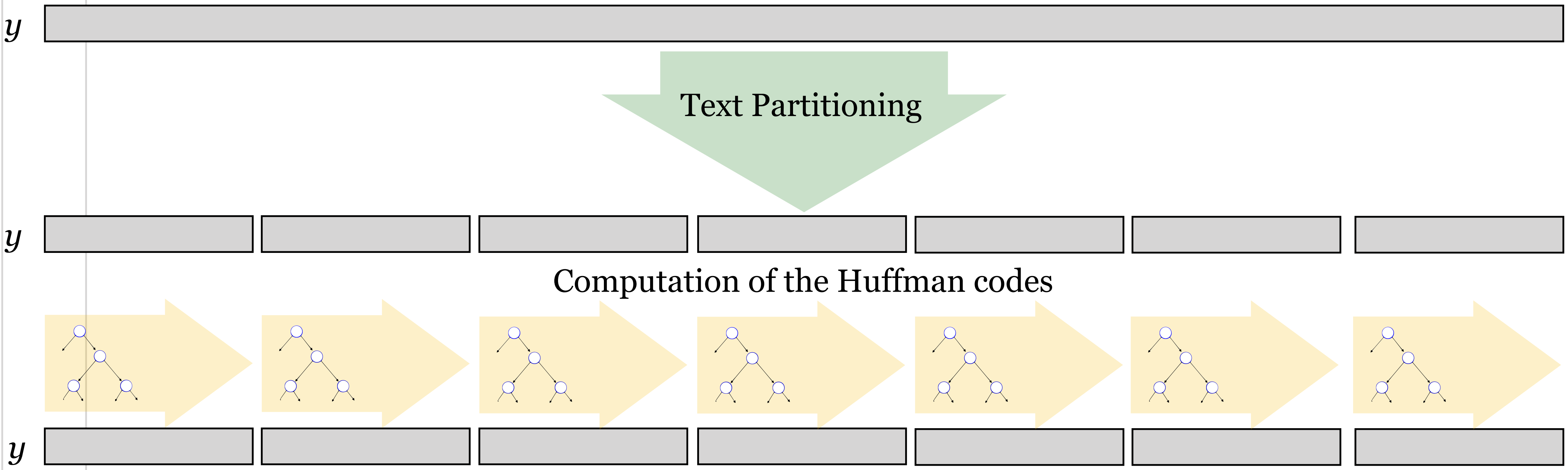
Adaptive Huffman Encoding in FLB Segmentation

The idea of Adaptive Huffman Encoding in FLB Segmentation is to create a new Huffman tree for each block obtained from the segmentation of the text. This strategy ensures that the frequency function used for tree construction more accurately reflects the character frequencies within that specific block, thereby enabling more efficient character encoding and consequently reducing the average delay within the block.



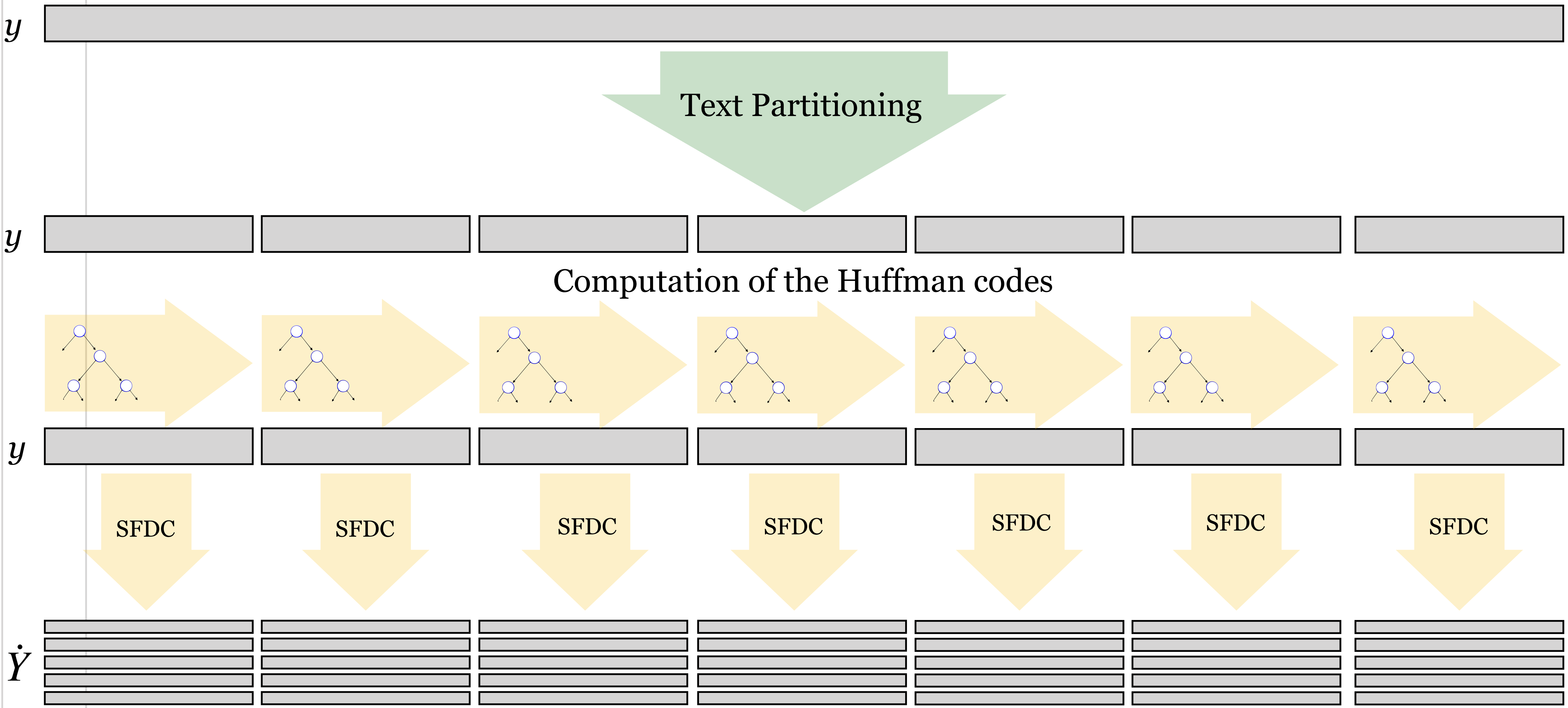
Adaptive Huffman Encoding in FLB Segmentation

The idea of Adaptive Huffman Encoding in FLB Segmentation is to create a new Huffman tree for each block obtained from the segmentation of the text. This strategy ensures that the frequency function used for tree construction more accurately reflects the character frequencies within that specific block, thereby enabling more efficient character encoding and consequently reducing the average delay within the block.



Adaptive Huffman Encoding in FLB Segmentation

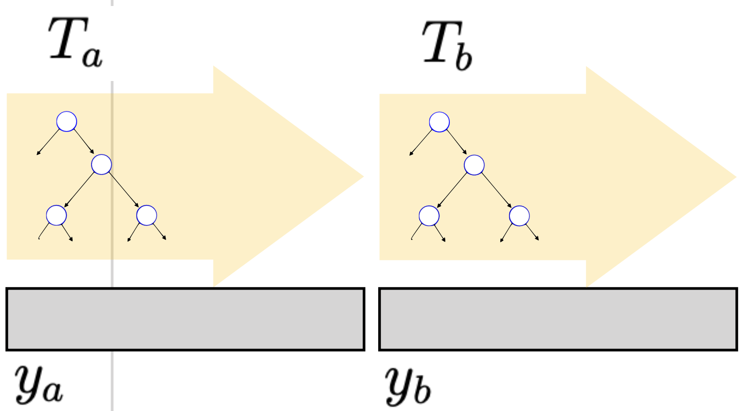
The idea of Adaptive Huffman Encoding in FLB Segmentation is to create a new Huffman tree for each block obtained from the segmentation of the text. This strategy ensures that the frequency function used for tree construction more accurately reflects the character frequencies within that specific block, thereby enabling more efficient character encoding and consequently reducing the average delay within the block.



Adaptive Huffman Encoding in FLB Segmentation

The idea of Adaptive Huffman Encoding in FLB Segmentation is to create a new Huffman tree for each block obtained from the segmentation of the text. This strategy ensures that the frequency function used for tree construction more accurately reflects the character frequencies within that specific block, thereby enabling more efficient character encoding and consequently reducing the average delay within the block.

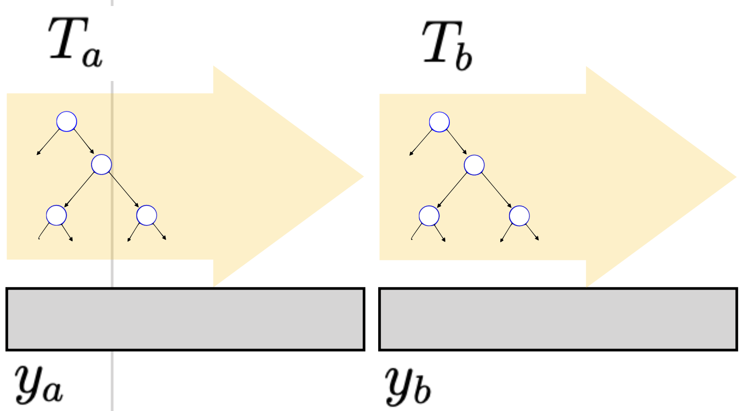
We adopt the *cosine distance metric* to compute the similarity between the trees of two adjacent blocks. In the context of AFLB, cosine distance is employed to evaluate the similarity between Huffman trees derived from continuous text blocks.



Adaptive Huffman Encoding in FLB Segmentation

The idea of Adaptive Huffman Encoding in FLB Segmentation is to create a new Huffman tree for each block obtained from the segmentation of the text. This strategy ensures that the frequency function used for tree construction more accurately reflects the character frequencies within that specific block, thereby enabling more efficient character encoding and consequently reducing the average delay within the block.

We adopt the *cosine distance metric* to compute the similarity between the trees of two adjacent blocks. In the context of AFLB, cosine distance is employed to evaluate the similarity between Huffman trees derived from continuous text blocks.



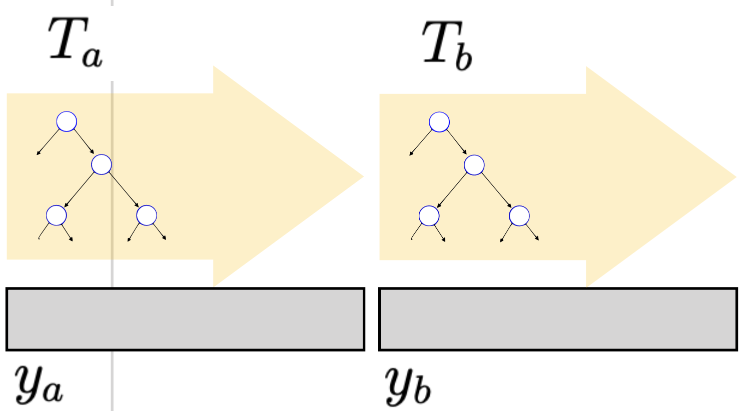
$$\delta(T_a, T_b) = \frac{f_a \cdot f_b}{||f_a|| \times ||f_b||}$$

$$\gamma = \frac{1}{N-1} \sum_{i=0}^{N-2} \delta(T_i, T_{i+1})$$

Adaptive Huffman Encoding in FLB Segmentation

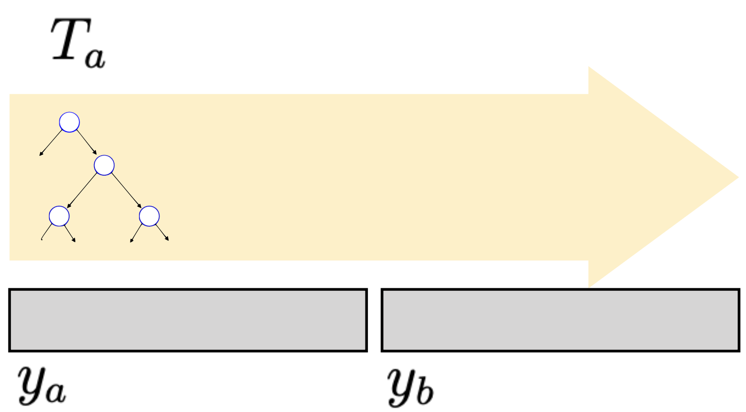
The idea of Adaptive Huffman Encoding in FLB Segmentation is to create a new Huffman tree for each block obtained from the segmentation of the text. This strategy ensures that the frequency function used for tree construction more accurately reflects the character frequencies within that specific block, thereby enabling more efficient character encoding and consequently reducing the average delay within the block.

We adopt the *cosine distance metric* to compute the similarity between the trees of two adjacent blocks. In the context of AFLB, cosine distance is employed to evaluate the similarity between Huffman trees derived from continuous text blocks.



$$\delta(T_a, T_b) = \frac{f_a \cdot f_b}{||f_a|| \times ||f_b||}$$

$$\gamma = \frac{1}{N-1} \sum_{i=0}^{N-2} \delta(T_i, T_{i+1})$$



Adaptive Huffman Encoding in FLB Segmentation

PROTEIN

Block Size (in KB)	Avg. Delay FLB	Avg. Delay AFLB
10^0	0.26	0.16
10^1	0.45	0.15
10^2	0.94	0.21
10^3	1.02	0.66
10^4	1.01	0.98
10^5	1.06	1.07

Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
104,858	39,610	6,378,498	11.760 %
10,486	2,525	417,090	0.760 %
1,049	289	50,034	0.090 %
105	54	10,072	0.020 %
11	4	876	0.012 %
2	2	448	0.004 %

DBLP

Block Size (in KB)	Avg. Delay FLB	Avg. Delay AFLB
10^0	2.07	1.16
10^1	2.15	1.48
10^2	2.18	1.63
10^3	2.17	2.03
10^4	2.16	2.02
10^5	2.84	2.82

Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
104,858	104,437	76,249,148	112.810 %
10,486	8,915	9,405,724	13.780 %
1,049	815	991,310	1.450 %
105	98	138,124	0.200 %
11	7	11,520	0.020 %
2	2	3,334	0.005 %

ENGLISH

Block Size (in KB)	Avg. Delay FLB	Avg. Delay AFLB
10^0	10.06	3.71
10^1	63.08	21.36
10^2	502.03	197.77
10^3	2,852.33	2,122.98
10^4	16,957.22	12,615.84
10^5	59,829.62	59,016.76

Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
104,858	104,559	55,760,356	95.420 %
10,486	9,391	9,156,926	15.470 %
1,049	655	802,572	1.350 %
105	88	129,328	0.220 %
11	8	13,070	0.020 %
2	2	3,368	0.015 %

Rare Marker Block Segmentation

The RMB segmentation formally identifies characters $c \in \Sigma$ with a frequency $f(c)$ below a predefined threshold, termed *rare markers*. These rare markers are used to determine the points at which the text is segmented into blocks. Thus, the text y is divided into blocks such that each block ends immediately after the next occurrence of any rare marker.

To prevent the creation of excessively small blocks when rare markers occur in close proximity, we introduce a parameter $\beta > 0$, which sets a minimum block size. Formally, a block is closed at the position of a rare marker only if the next rare marker is at least β characters away.

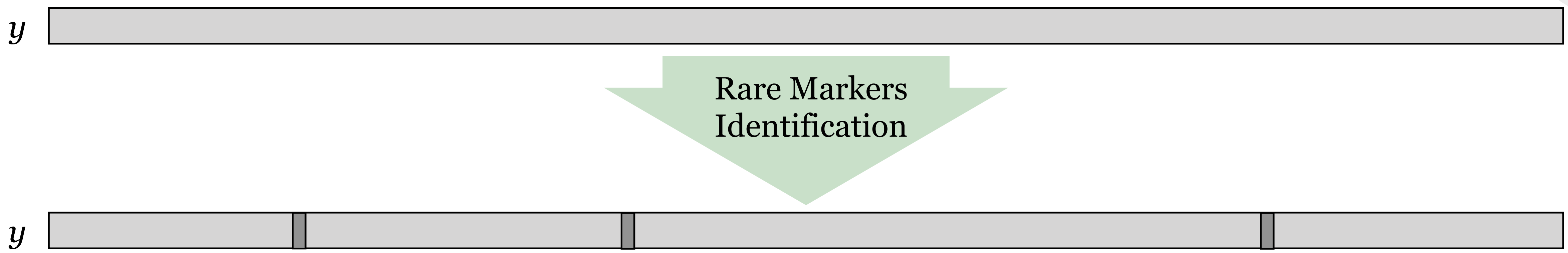
The RMB segmentation offers several advantages:

- **Efficiency:** The segmentation adapts to the inherent structure of the text, optimizing compression performance by aligning block boundaries with the distribution of low-frequency characters.
- **Scalability:** The method scales effectively with text size and complexity, adjusting dynamically to variations in text composition and character distribution.
- **Simplicity:** The use of clearly defined markers simplifies both the encoding and decoding processes, making the method practical for large datasets.

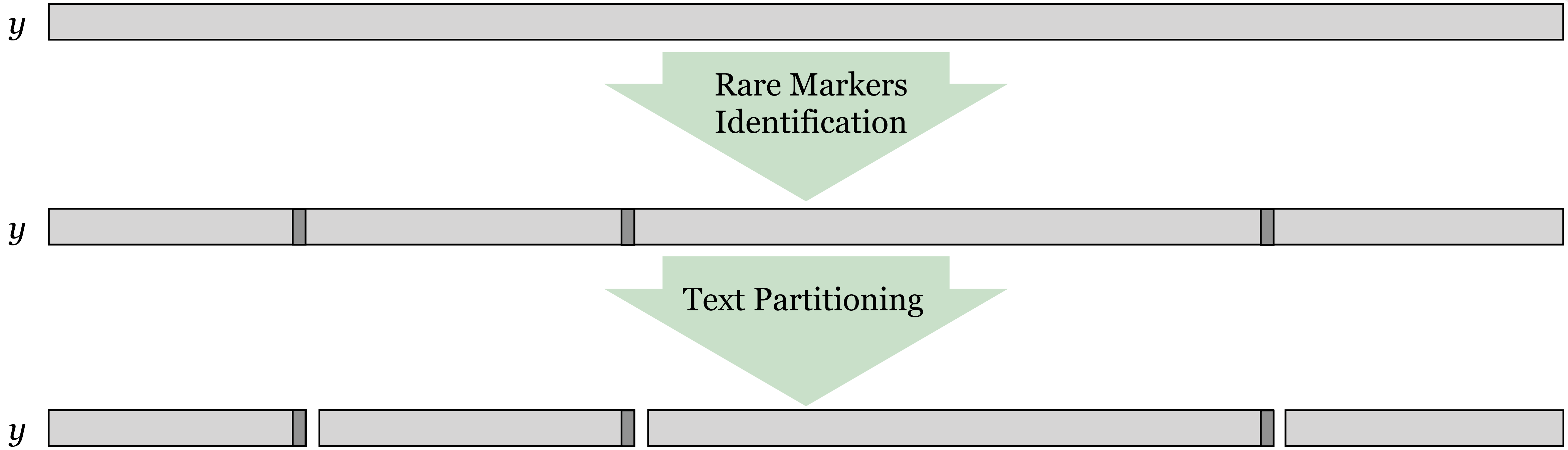
Rare Marker Block Segmentation

y

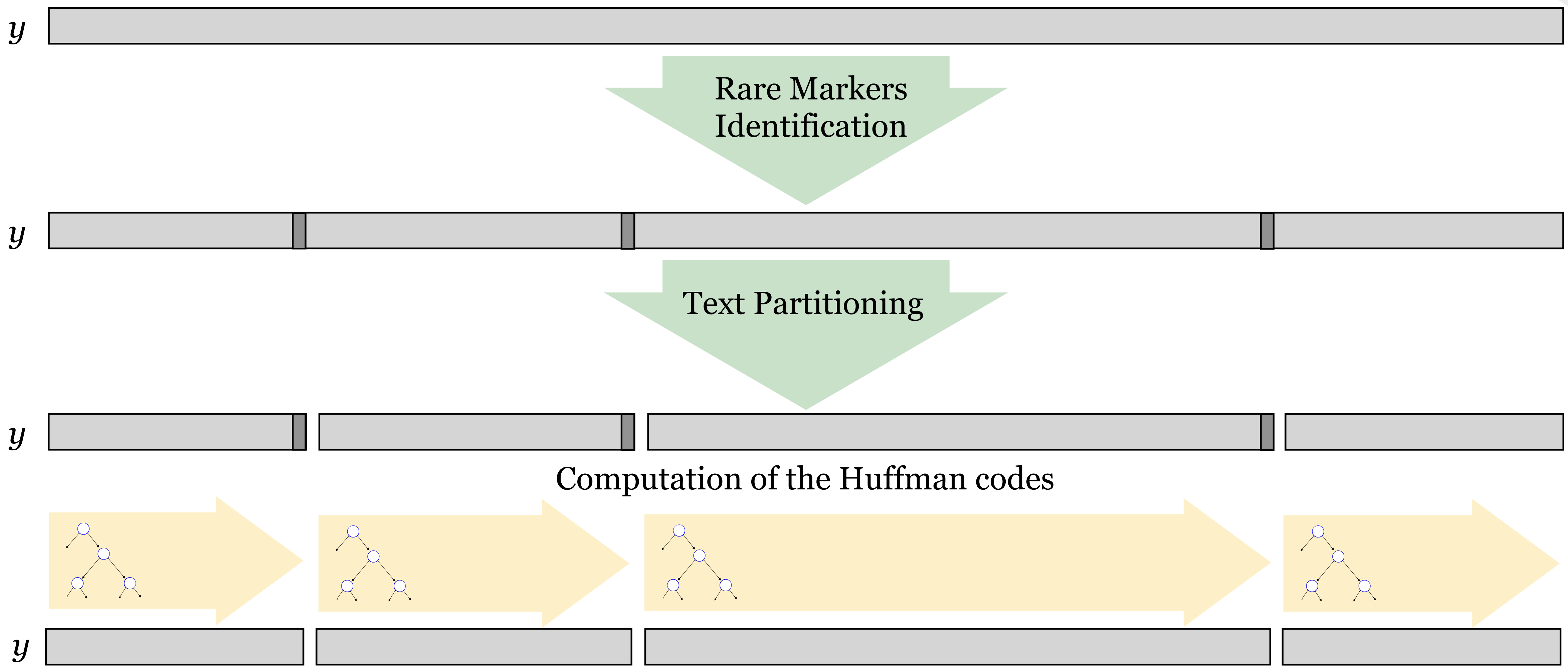
Rare Marker Block Segmentation



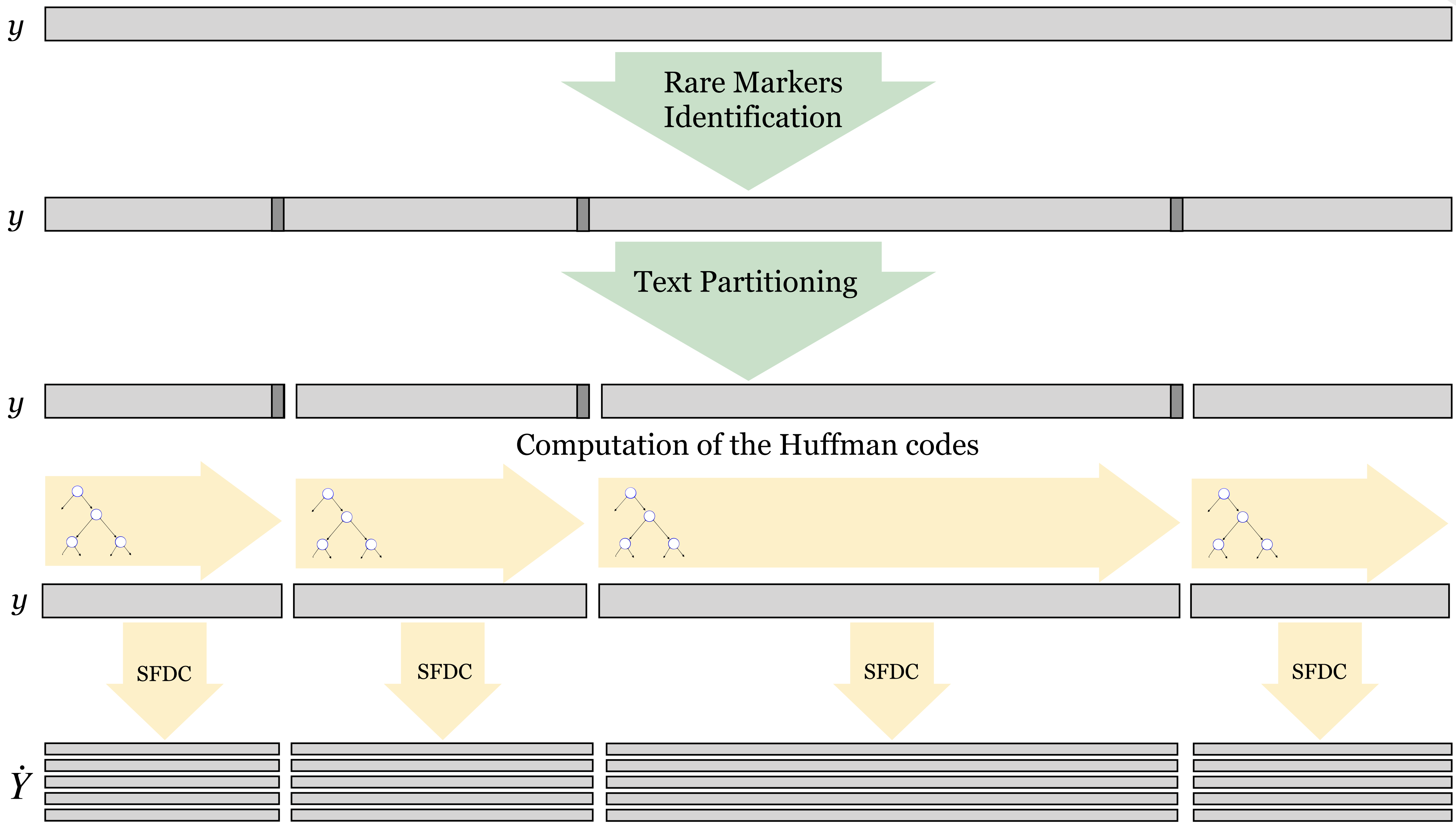
Rare Marker Block Segmentation



Rare Marker Block Segmentation



Rare Marker Block Segmentation



Rare Marker Block Segmentation

PROTEIN

FIXED BLOCK

Block Size (in KB)	Avg. Delay FLB	Avg. Delay AFLB	Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
10^0	0.26	0.16	104,858	39,610	6,378,498	11.760 %
10^1	0.45	0.15	10,486	2,525	417,090	0.760 %
10^2	0.94	0.21	1,049	289	50,034	0.090 %
10^3	1.02	0.66	105	54	10,072	0.020 %
10^4	1.01	0.98	11	4	876	0.012 %
10^5	1.06	1.07	2	2	448	0.004 %

RARE MARKER

Rare Elements	Average Delay	Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
2	0.77	114	70	13,316	0.02 %
4	0.61	4,244	1,495	258,920	0.34 %
6	0.45	19,900	12,033	487,114	0.79 %
8	0.39	22,794	19,458	863,314	1.26 %
10	0.21	35,395	31,147	1,223,612	1.96 %

Table 4. Experimental results obtained on the PROTEIN text using 5 layers. The results must be evaluated considering the standard version of SFDC shows an average delay equal to 1.02, and that the compressed text has a size of 55.36 MB.

Rare Marker Block Segmentation

DBLP

	Block Size (in KB)	Avg. Delay FLB	Avg. Delay AFLB	Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
Fixed Block	10^0	2.07	1.16	104,858	104,437	76,249,148	112.810 %
	10^1	2.15	1.48	10,486	8,915	9,405,724	13.780 %
	10^2	2.18	1.63	1,049	815	991,310	1.450 %
	10^3	2.17	2.03	105	98	138,124	0.200 %
	10^4	2.16	2.02	11	7	11,520	0.020 %
	10^5	2.84	2.82	2	2	3,334	0.005 %

	Rare Elements	Average Delay	Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
Rare Marker	2	1.93	59	49	69,746	0.09 %
	4	1.80	223	185	235,690	0.30 %
	6	1.81	323	273	331,508	0.42 %
	8	1.71	643	529	620,926	0.74 %
	10	1.66	1,171	1,002	1,021,286	1.19 %

Table 2. Experimental results obtained on the DBLP text using 6 layers. The results must be evaluated considering the standard version of SFDC shows an average delay equal to 2.19, and that the compressed text has a size of 68.91 MB.

ENGLISH

Fixed Block	Block Size (in KB)	Avg. Delay FLB	Avg. Delay AFLB	Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
	10^0	10.06	3.71	104,858	104,559	55,760,356	95.420 %
	10^1	63.08	21.36	10,486	9,391	9,156,926	15.470 %
	10^2	502.03	197.77	1,049	655	802,572	1.350 %
	10^3	2,852.33	2,122.98	105	88	129,328	0.220 %
	10^4	16,957.22	12,615.84	11	8	13,070	0.020 %
	10^5	59,829.62	59,016.76	2	2	3,368	0.015 %

Rare Marker	Rare Elements	Average Delay	Number of Blocks	Huffman Trees	Tree Size (in Byte)	Space Overhead
	2	9,680.20	24	15	16,650	0.02 %
	4	568.66	263	200	246,444	0.33 %
	6	1,048.50	464	368	415,620	0.56 %
	8	135.82	1,749	1,398	1,275,560	1.47 %
	10	114.41	2,372	1,947	1,621,346	1.83 %

Table 3. Experimental results obtained on the ENGLISH text using 5 layers. The results must be evaluated considering the standard version of SFDC shows an average delay equal to 44, 387.30, and that the compressed text has a size of 60.1 MB.

Conclusions

In this article, we have explored three primary text compression strategies: Fixed-Length Block (FLB) segmentation, Adaptive Fixed-Length Block (AFLB) segmentation, and Rare Marker Block (RMB) segmentation. Each approach offers unique benefits and addresses different aspects of the text compression challenge.

Looking forward, several promising directions for future research have been identified. One avenue is to investigate the use of rare markers as starting points of blocks rather than ending points.

Additionally, exploring the efficacy of a First-In, First-Out (FIFO) strategy as opposed to the Last-In, First-Out (LIFO) strategy currently used could provide insights into improving the decoding efficiency.

Conclusions

In this article, we have explored three primary text compression strategies: Fixed-Length Block (FLB) segmentation, Adaptive Fixed-Length Block (AFLB) segmentation, and Rare Marker Block (RMB) segmentation. Each approach offers unique benefits and addresses different aspects of the text compression challenge.

Looking forward, several promising directions for future research have been identified. One avenue is to investigate the use of rare markers as starting points of blocks rather than ending points.

Additionally, exploring the efficacy of a First-In, First-Out (FIFO) strategy as opposed to the Last-In, First-Out (LIFO) strategy currently used could provide insights into improving the decoding efficiency.

Thanks