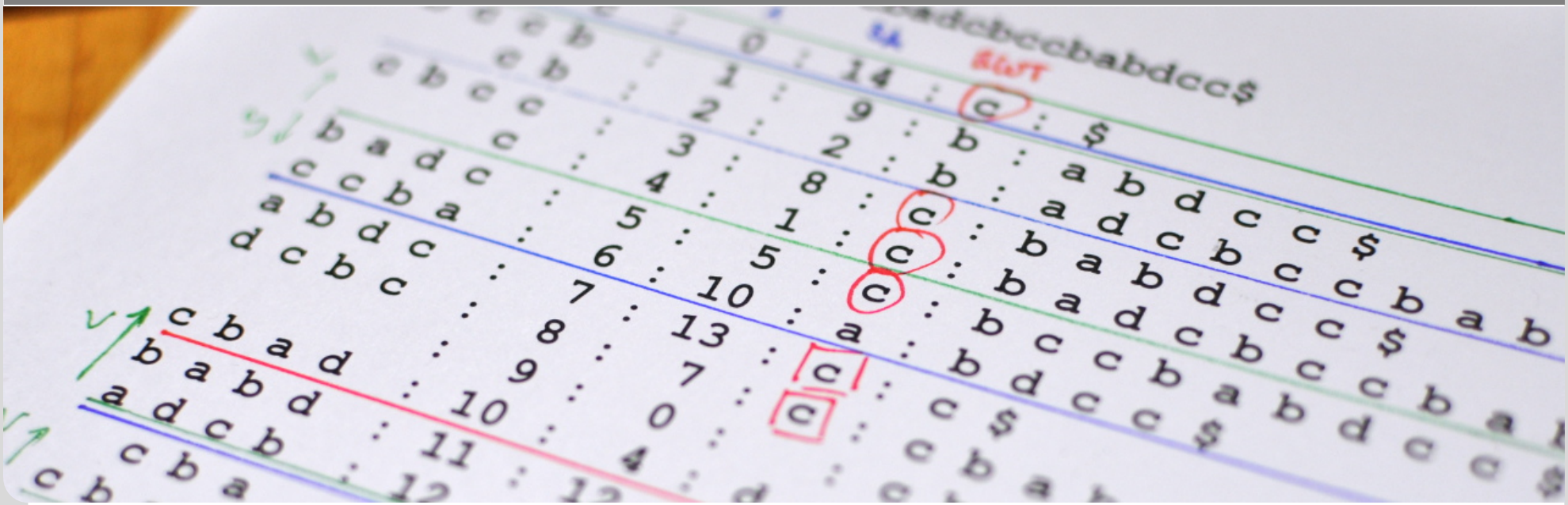
Karlsruhe Institute of Technology

Inducing Suffix and LCP Arrays in External Memory

Timo Bingmann, Johannes Fischer, and Vitaly Osipov | January 7th, 2013 @ ALENEX'13

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS



Abstract

We consider text index construction in external memory (EM). Our first contribution is an inducing algorithm for suffix arrays in external memory. Practical tests show that this outperforms the previous best EM suffix sorter [Dementiev et al., ALENEX 2005] by a factor of about two in time and I/O-volume.

Our second contribution is to augment the first algorithm to also construct the array of longest common prefixes (LCPs). This yields the first EM construction algorithm for LCP arrays. The overhead in time and I/O volume for this extended algorithm over plain suffix array construction is roughly two.

The algorithms scale far beyond problem sizes previously considered in the literature (text size of 80 GiB using only 4 GiB of RAM in our experiments).

Overview

1. Introduction and Motivation
   - Evolution of Suffix Array Construction Algorithms
   - History of LCP Construction Algorithms
2. Example of the Inducing Step in the eSAIS Algorithm
   - Inducing the Suffix Array
   - Inducing the LCP Array
   - Finding Ranks of S^*-Suffixes
3. Implementation and Experimental Results
   - Implementation Highlights
   - Experiments – eSAIS vs. DC3

Example $T = [$ 0 1 2 3 4 5 6 7 8 9 10 11 12 cababcbababb$ $]$

i	T_i
0	c a b a b c b a b a b b $
1	a b a b c b a b a b b $
2	b a b c b a b a b b $
3	a b c b a b a b b $
4	b c b a b a b b $
5	c b a b a b b $
6	b a b a b b $
7	a b a b b $
8	b a b b $
9	a b b $
10	b b $
11	b $
12	$

Example $T = [\texttt{cababcbababb\$}]$

	0	1	2	3	4	5	6	7	8	9	10	11	12

SA_i	$T_{SA_i \ldots n}$
12	$
7	a b a b b $
1	a b a b c b a b a b b $
9	a b b $
3	a b c b a b a b b $
11	b $
6	b a b a b b $
8	b a b b $
2	b a b c b a b a b b $
10	b b $
4	b c b a b a b b $
0	c a b a b c b a b a b b $
5	c b a b a b b $

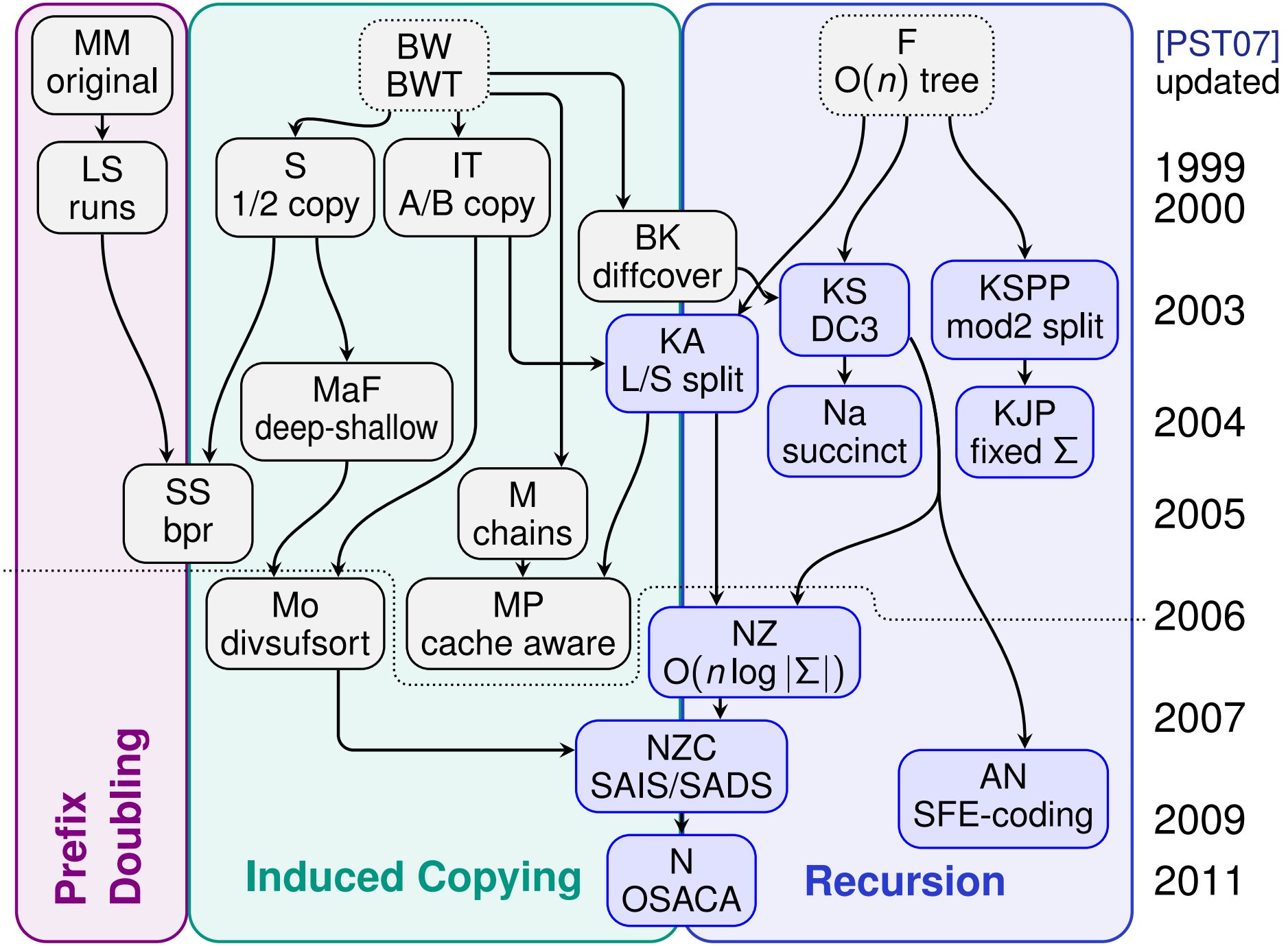
Example $T =$ [cababcbababb$]

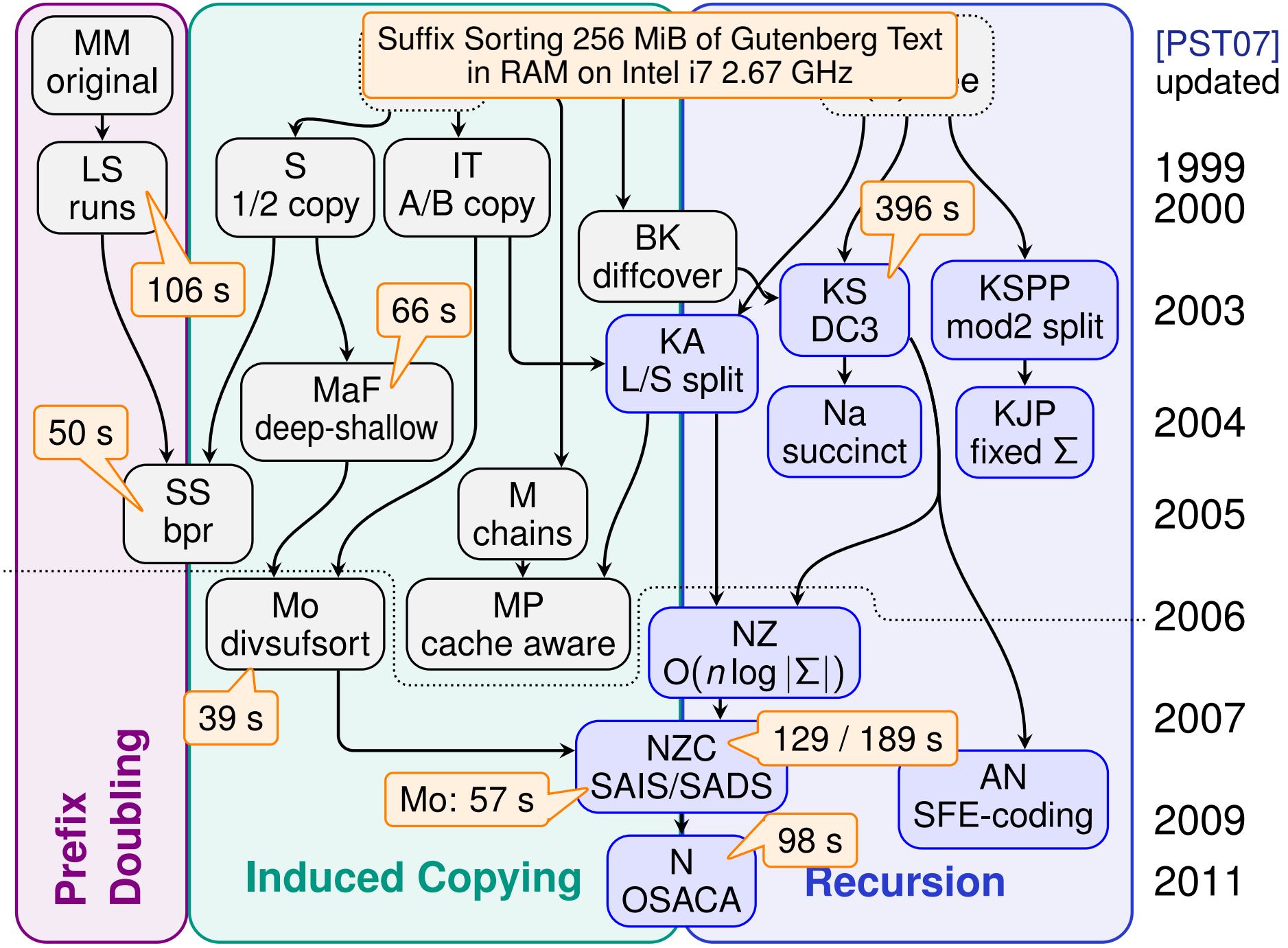
	0	1	2	3	4	5	6	7	8	9	10	11	12

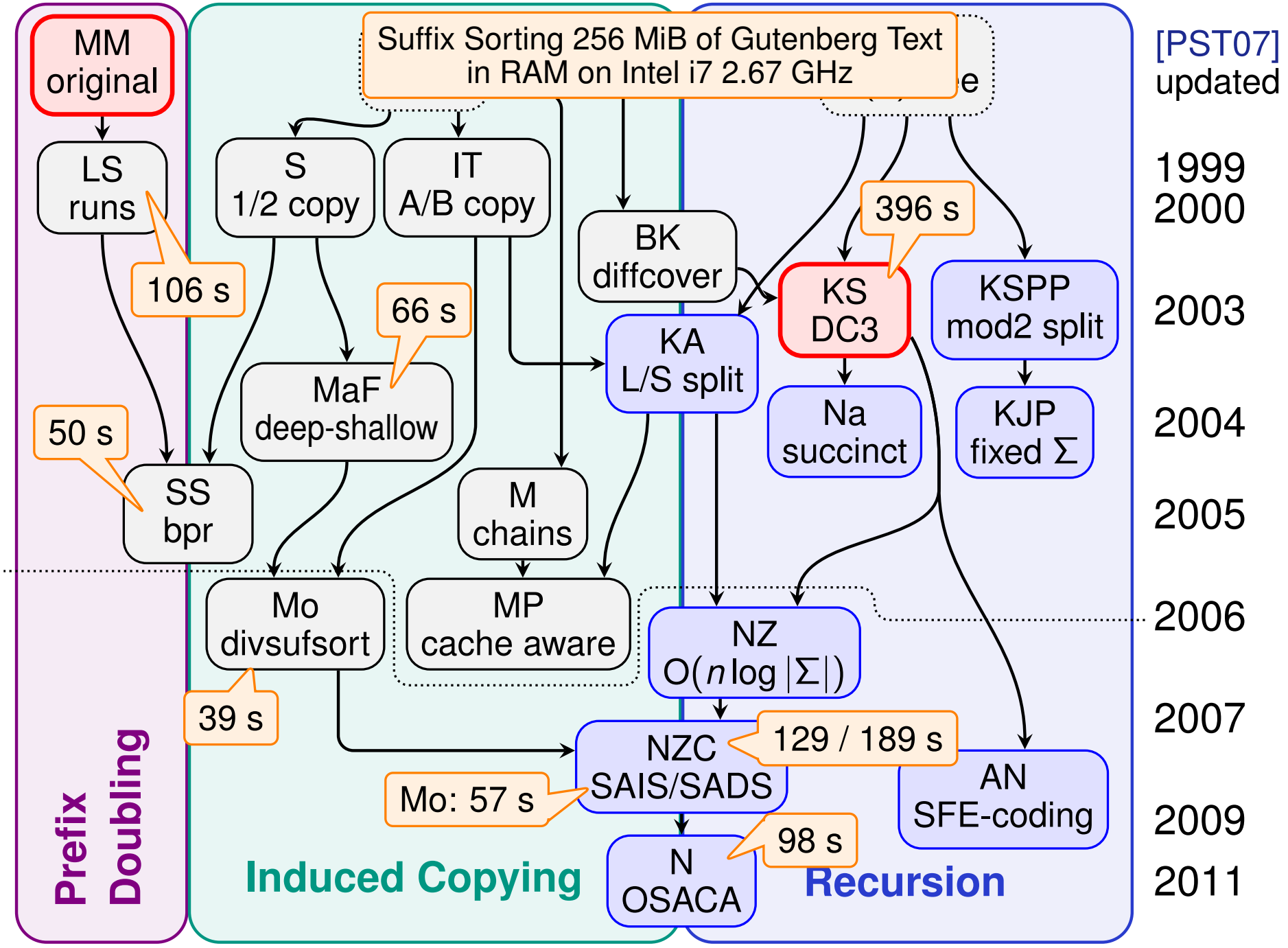
SA_i	LCP_i	$T_{SA_i \ldots n}$
12		$
7		a b a b b $
1		a b a b c b a b a b b $
9		a b b $
3		a b c b a b a b b $
11		b $
6		b a b a b b $
8	3	b a b b $
2		b a b c b a b a b b $
10		b b $
4		b c b a b a b b $
0		c a b a b c b a b a b b $
5		c b a b a b b $

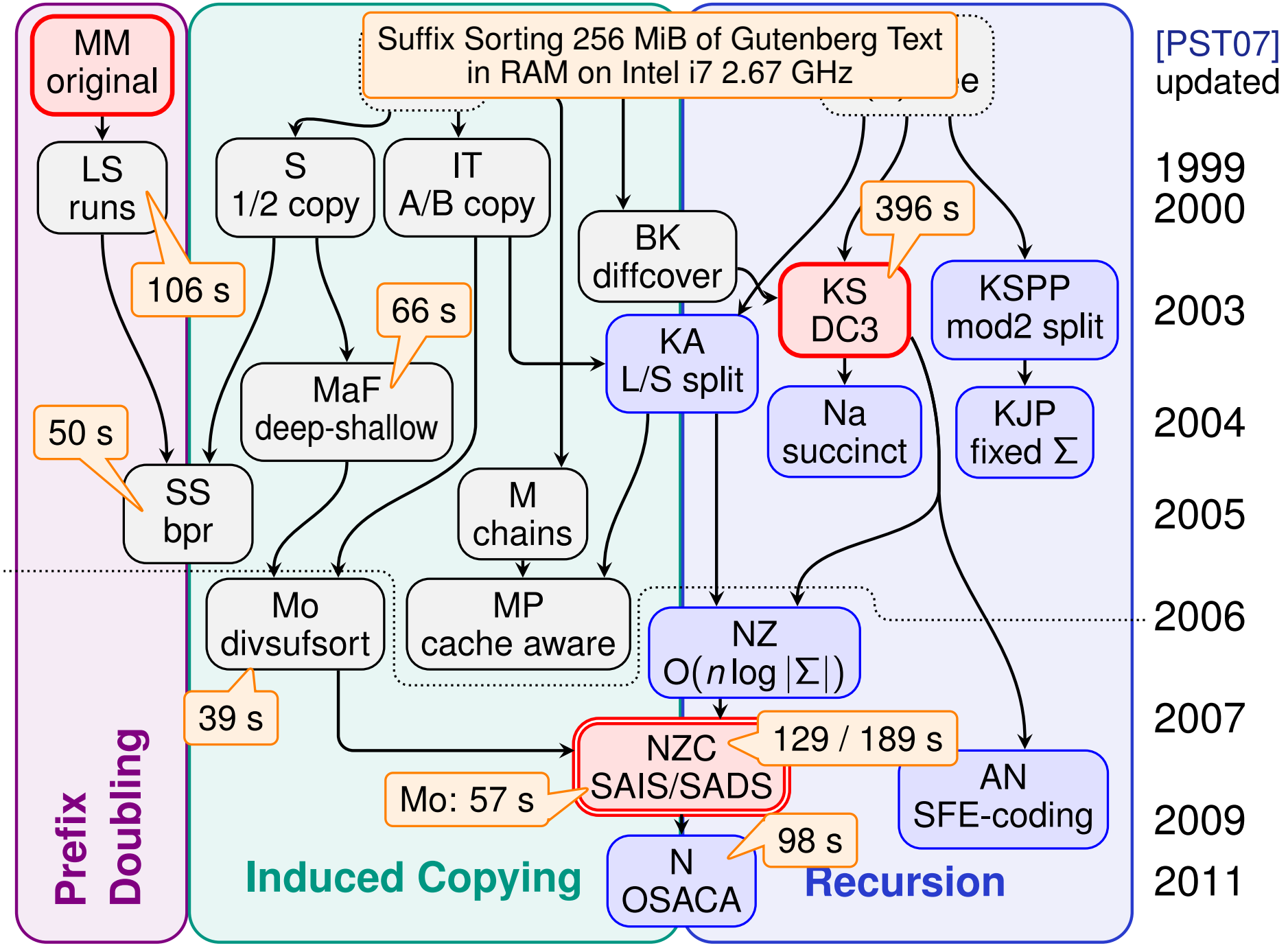
Example $T = [\texttt{cababcbababb\$}]$

		0	1	2	3	4	5	6	7	8	9	10	11	12
		c	a	b	a	b	c	b	a	b	a	b	b	$
SA_i	LCP_i													
12	-	$												
7	0	a	b	a	b	b	$							
1	4	a	b	a	b	c	b	a	b	a	b	b	$	
9	2	a	b	b	$									
3	2	a	b	c	b	a	b	a	b	b	$			
11	0	b	$											
6	1	b	a	b	a	b	b	$						
8	3	b	a	b	b	$								
2	3	b	a	b	c	b	a	b	a	b	b	$		
10	1	b	b	$										
4	1	b	c	b	a	b	a	b	b	$				
0	0	c	a	b	a	b	c	b	a	b	a	b	b	$
5	1	c	b	a	b	a	b	b	$					


MM
original
LS
runs
BW
BWT
S
1/2 copy
IT
A/B copy
BK
diffcover
F
O(n) tree
[PST07]
updated
1999
2000
KS
DC3
KSPP
mod2 split
2003
KA
L/S split
MaF
deep-shallow
Na
succinct
KJP
fixed Σ
2004
SS
bpr
M
chains
2005
Mo
divsufsort
MP
cache aware
2006
NZ
O(n log |Σ|)
2007
NZC
SAIS/SADS
AN
SFE-coding
2009
N
OSACA
2011
Prefix
Doubling
Induced Copying
Recursion


Suffix Sorting 256 MiB of Gutenberg Text in RAM on Intel i7 2.67 GHz
[PST07] updated
MM original
LS runs
106 s
S 1/2 copy
IT A/B copy
BK diffcover
396 s
KS DC3
KSPP mod2 split
66 s
MaF deep-shallow
KA L/S split
Na succinct
KJP fixed Σ
50 s
SS bpr
M chains
Mo divsufsort
MP cache aware
NZ O(n log |Σ|)
39 s
NZC SAIS/SADS
129 / 189 s
AN SFE-coding
Mo: 57 s
N OSACA
98 s
Prefix Doubling
Induced Copying
Recursion
1999
2000
2003
2004
2005
2006
2007
2009
2011


Suffix Sorting 256 MiB of Gutenberg Text in RAM on Intel i7 2.67 GHz
[PST07] updated
MM original
LS runs
106 s
S 1/2 copy
IT A/B copy
BK diffcover
396 s
KS DC3
KSPP mod2 split
66 s
MaF deep-shallow
KA L/S split
Na succinct
KJP fixed Σ
50 s
SS bpr
M chains
Mo divsufsort
MP cache aware
NZ O(n log |Σ|)
39 s
NZC SAIS/SADS
129 / 189 s
AN SFE-coding
Mo: 57 s
98 s
N OSACA
Prefix Doubling
Induced Copying
Recursion
1999
2000
2003
2004
2005
2006
2007
2009
2011


Suffix Sorting 256 MiB of Gutenberg Text in RAM on Intel i7 2.67 GHz
[PST07] updated
MM original
LS runs
106 s
S 1/2 copy
IT A/B copy
BK diffcover
396 s
KS DC3
KSPP mod2 split
66 s
MaF deep-shallow
KA L/S split
Na succinct
KJP fixed Σ
50 s
SS bpr
M chains
Mo divsufsort
MP cache aware
NZ O(n log |Σ|)
39 s
NZC SAIS/SADS
129 / 189 s
AN SFE-coding
Mo: 57 s
98 s
N OSACA
Prefix Doubling
Induced Copying
Recursion
1999
2000
2003
2004
2005
2006
2007
2009
2011

LCP Construction Algorithms

Algorithm		Construction	Time	Space
MM	1993	T $\rightarrow$ SA,LCP	$O(n \log n)$	$9n$
KLAAP	2001	T,SA $\rightarrow$ LCP	$O(n)$	$13n$
KS	2003	T $\rightarrow$ SA,LCP	$O(n)$	$O(n)$ EM
M	2004	T,SA $\rightarrow$ LCP	$O(n)$	$9n$ / $5n$
PT	2008	T,SA $\rightarrow$ v-LCP	$O(nv)$	$6n+O(\frac{n}{\sqrt{v}}+v)$
Φ-KMP	2009	T,SA $\rightarrow$ PLCP	$O(n \log n)$	$5n+\frac{3}{8}n$
GO	2011	T,SA,BWT,LF $\rightarrow$ LCP	$O(n^2)$	$11n$
F	2011	T $\rightarrow$ SA,LCP	$O(n)$	$9n$

Example $T = [$ 0 1 2 3 4 5 6 7 8 9 10 11 12 c a b a b c b a b a b b $]

SA_i		$T_{SA_i \ldots n}$
12		$
7		a b a b b $
1		a b a b c b a b a b b $
9		a b b $
3		a b c b a b a b b $
11		b $
6		b a b a b b $
8		b a b b $
2		b a b c b a b a b b $
10		b b $
4		b c b a b a b b $
0		c a b a b c b a b a b b $
5		c b a b a b b $

Example $T =$ [cababcbababb$]

		0	1	2	3	4	5	6	7	8	9	10	11	12
SA_i		$T_{SA_i \ldots n}$												
12		$												
7		a	b	a	b	b	$							
1		a	b	a	b	c	b	a	b	a	b	b	$	
9		a	b	b	$									
3		a	b	c	b	a	b	a	b	b	$			
11		b	$											
6		b	a	b	a	b	b	$						
8		b	a	b	b	$								
2		b	a	b	c	b	a	b	a	b	b	$		
10		b	b	$										
4		b	c	b	a	b	a	b	b	$				
0		c	a	b	a	b	c	b	a	b	a	b	b	$
5		c	b	a	b	a	b	b	$					

Example $T = [\texttt{cababcbababb\$}]$

		0	1	2	3	4	5	6	7	8	9	10	11	12
		c	a	b	a	b	c	b	a	b	a	b	b	$

SA_i		$T_{SA_i \ldots n}$
12		$
7		a b a b b $
1		a b a b c b a b a b b $
9		a b b $
3		a b c b a b a b b $
11		b $
6		b a b a b b $
8		b a b b $
2		b a b c b a b a b b $
10		b b $
4		b c b a b a b b $
0		c a b a b c b a b a b b $
5		c b a b a b b $

Example $T =$ [cababcbababb$]

0 1 2 3 4 5 6 7 8 9 10 11 12

SA_i	T_{i-1}	$T_{SA_i \ldots n}$
12	b	$
7	b	a b a b b $
1	c	a b a b c b a b a b b $
9	b	a b b $
3	b	a b c b a b a b b $
11	b	b $
6	c	b a b a b b $
8	a	b a b b $
2	a	b a b c b a b a b b $
10	a	b b $
4	a	b c b a b a b b $
0	-	c a b a b c b a b a b b $
5	b	c b a b a b b $

Example $T =$ [cababcbababb$]

	0	1	2	3	4	5	6	7	8	9	10	11	12
	c	a	b	a	b	c	b	a	b	a	b	b	$

SA_i	T_{i-1}	$T_{SA_i \ldots n}$
12	b	$
7	b	a b a b b $
1	c	a b a b c b a b a b b $
9	b	a b b $
3	b	a b c b a b a b b $
11	b	b $
6	c	b a b a b b $
8	a	b a b b $
2	a	b a b c b a b a b b $
10	a	b b $
4	a	b c b a b a b b $
0	-	c a b a b c b a b a b b $
5	b	c b a b a b b $

Example $T = [\texttt{cababcbababb\$}]$

	0	1	2	3	4	5	6	7	8	9	10	11	12
	c	a	b	a	b	c	b	a	b	a	b	b	$

SA_i	T_{i-1}	$T_{SA_i \ldots n}$
12	b	$
7	b	a b a b b $
1	c	a b a b c b a b a b b $
9	b	a b b $
3	b	a b c b a b a b b $
11	b	b $
6	c	b a b a b b $
8	a	b a b b $
2	a	b a b c b a b a b b $
10	a	b b $
4	a	b c b a b a b b $
0	-	c a b a b c b a b a b b $
5	b	c b a b a b b $

Example $T =$ [cababcbababb$]

	0	1	2	3	4	5	6	7	8	9	10	11	12
	c	a	b	a	b	c	b	a	b	a	b	b	$

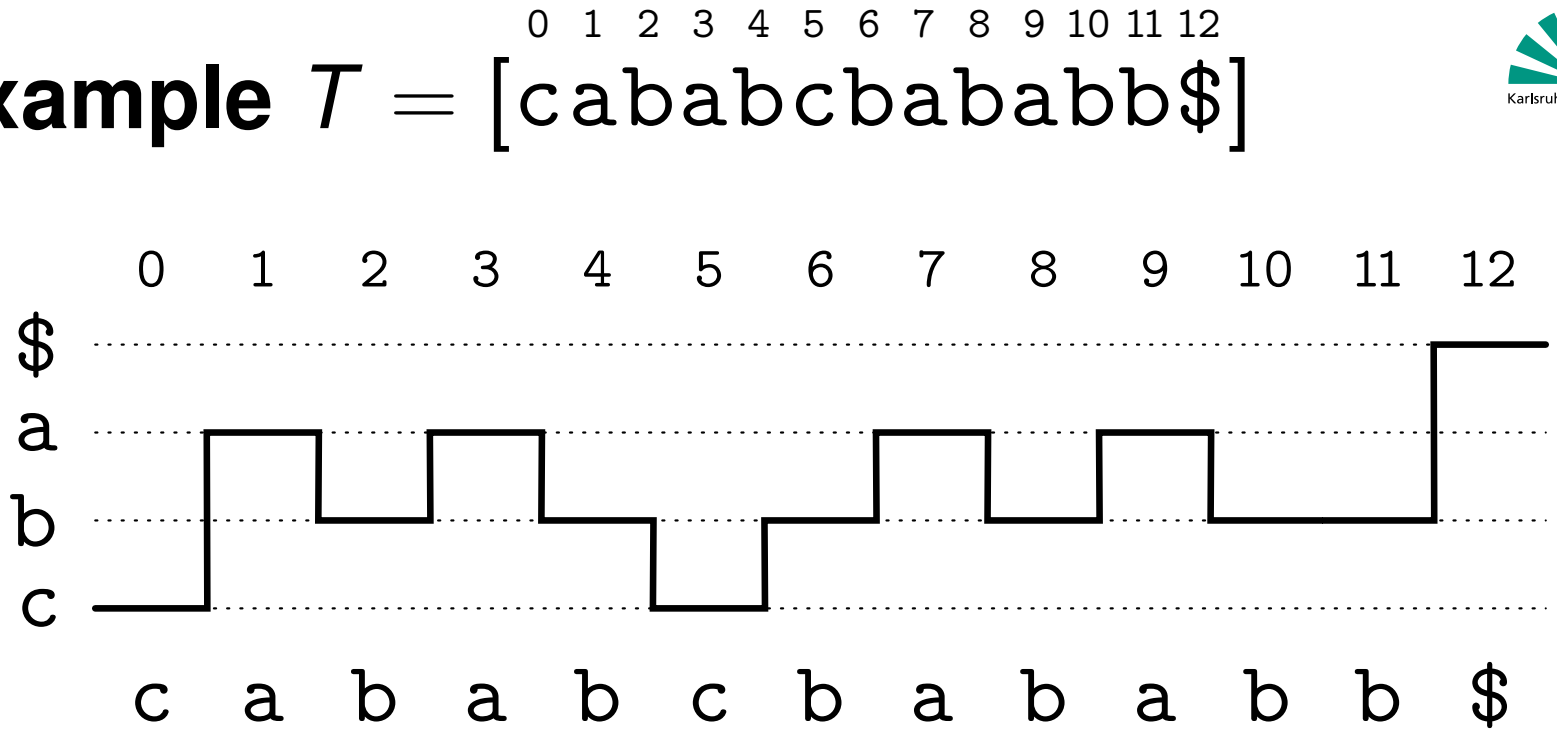
SA_i	T_{i-1}	$T_{SA_i \ldots n}$	
12	b	$	← 0
7	b	a b a b b $	← 1
1	c	a b a b c b a b a b b $	← 2
9	b	a b b $	← 3
3	b	a b c b a b a b b $	← 4
11	b	b $	
6	c	b a b a b b $	
8	a	b a b b $	
2	a	b a b c b a b a b b $	
10	a	b b $	
4	a	b c b a b a b b $	
0	-	c a b a b c b a b a b b $	
5	b	c b a b a b b $	

Example $T = [\texttt{cababcbababb\$}]$

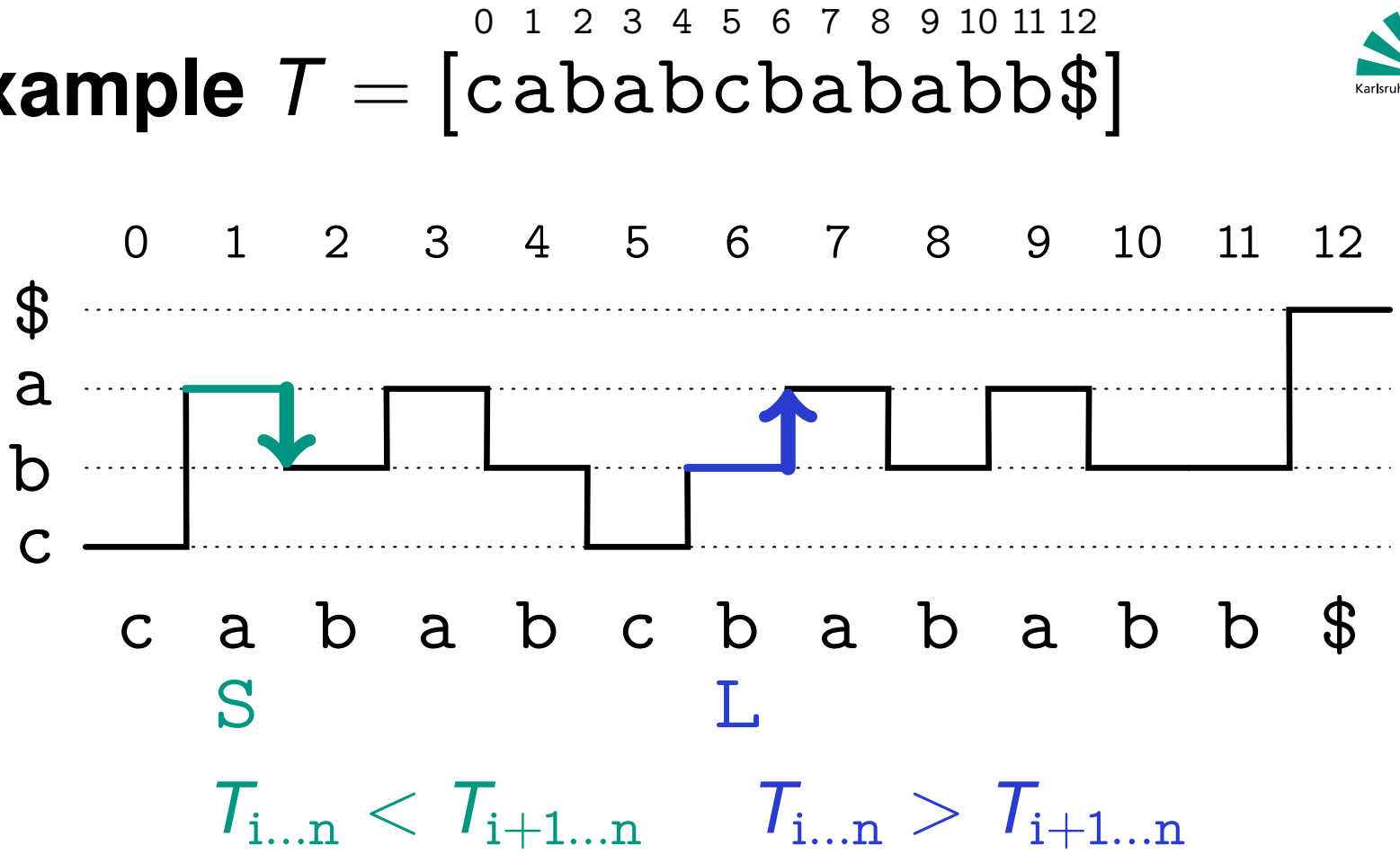
	0	1	2	3	4	5	6	7	8	9	10	11	12
	c	a	b	a	b	c	b	a	b	a	b	b	$

SA_i	T_{i-1}	$T_{SA_i \ldots n}$	
12	b	$	← b,0
7	b	a b a b b $	← b,1
1	c	a b a b c b a b a b b $	← c,2
9	b	a b b $	← b,3
3	b	a b c b a b a b b $	← b,4
11	b	b $	
6	c	b a b a b b $	
8	a	b a b b $	
2	a	b a b c b a b a b b $	
10	a	b b $	
4	a	b c b a b a b b $	
0	-	c a b a b c b a b a b b $	
5	b	c b a b a b b $	

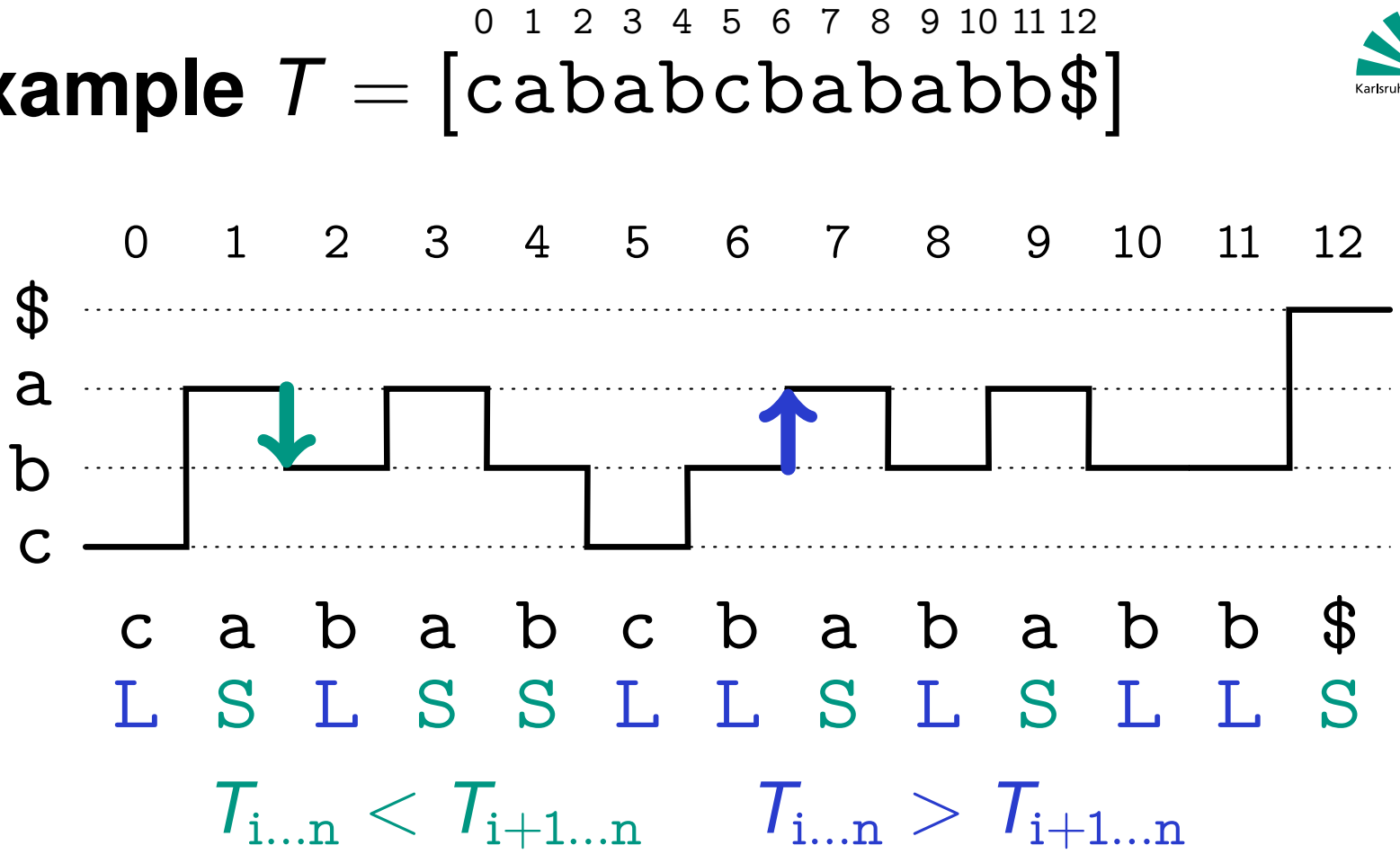
Example $T = [\texttt{cababcbababb\$}]$



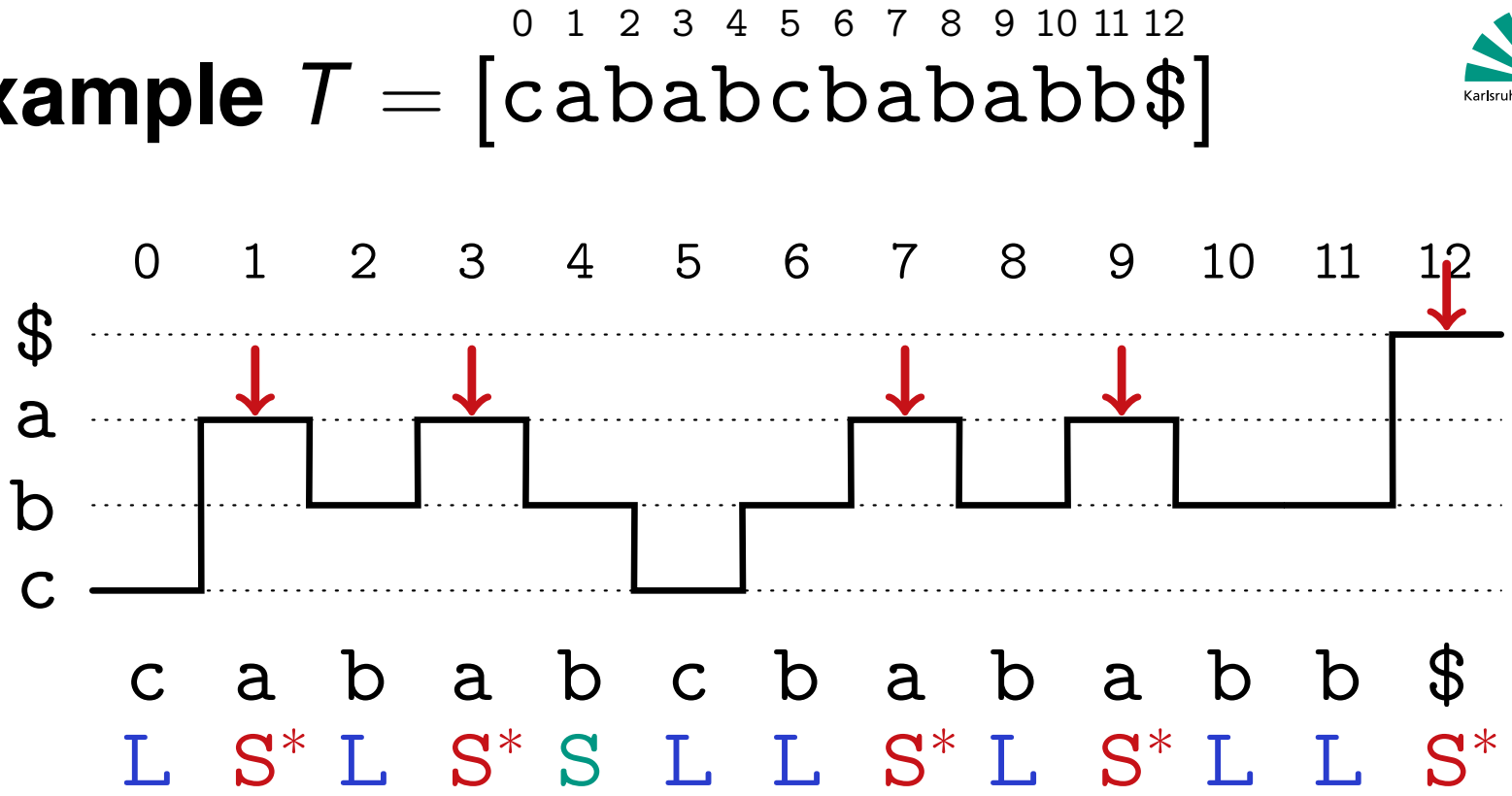
Example $T = [\texttt{cababcbababb\$}]$



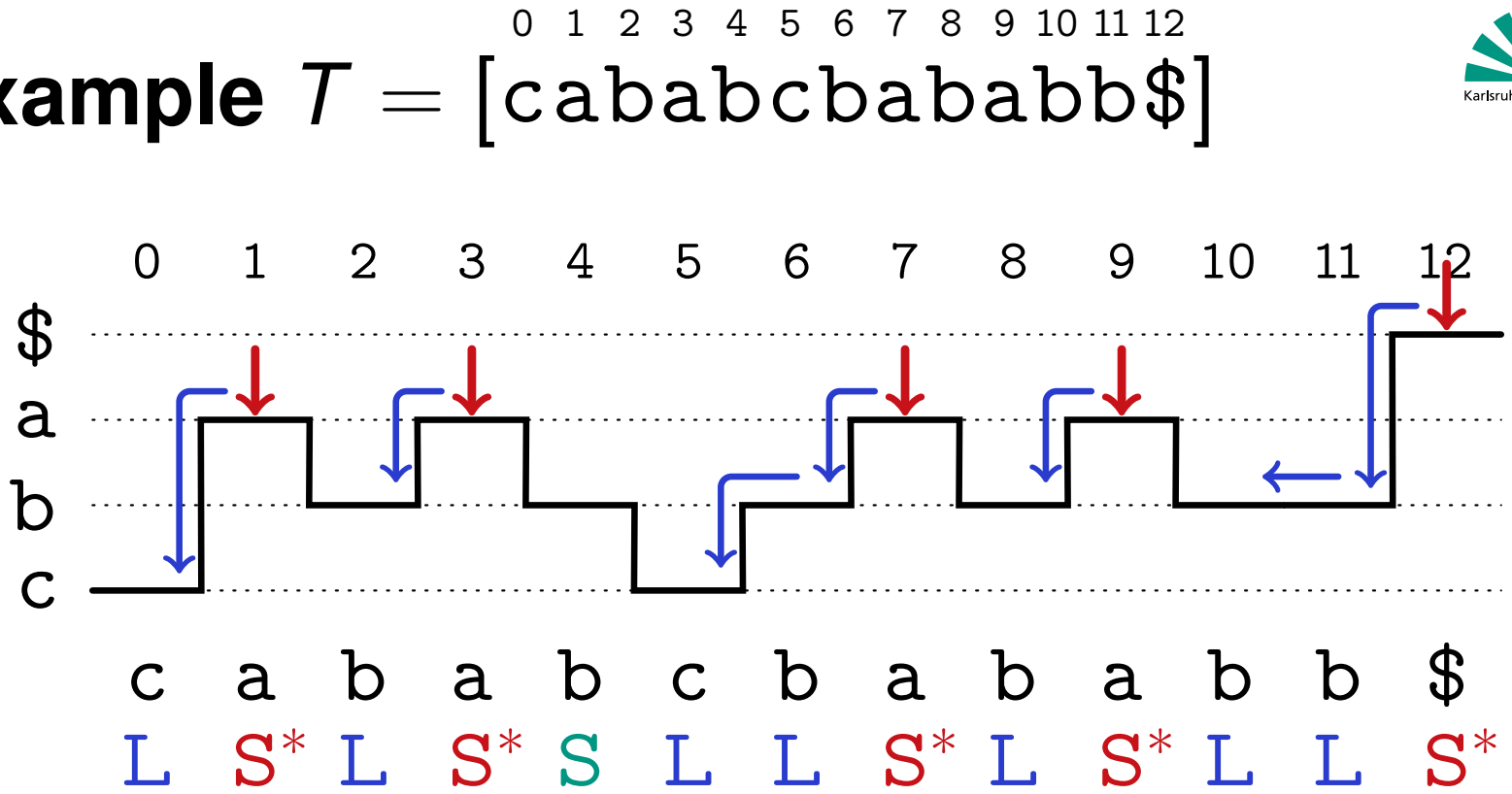
Example $T = [\texttt{cababcbababb\$}]$



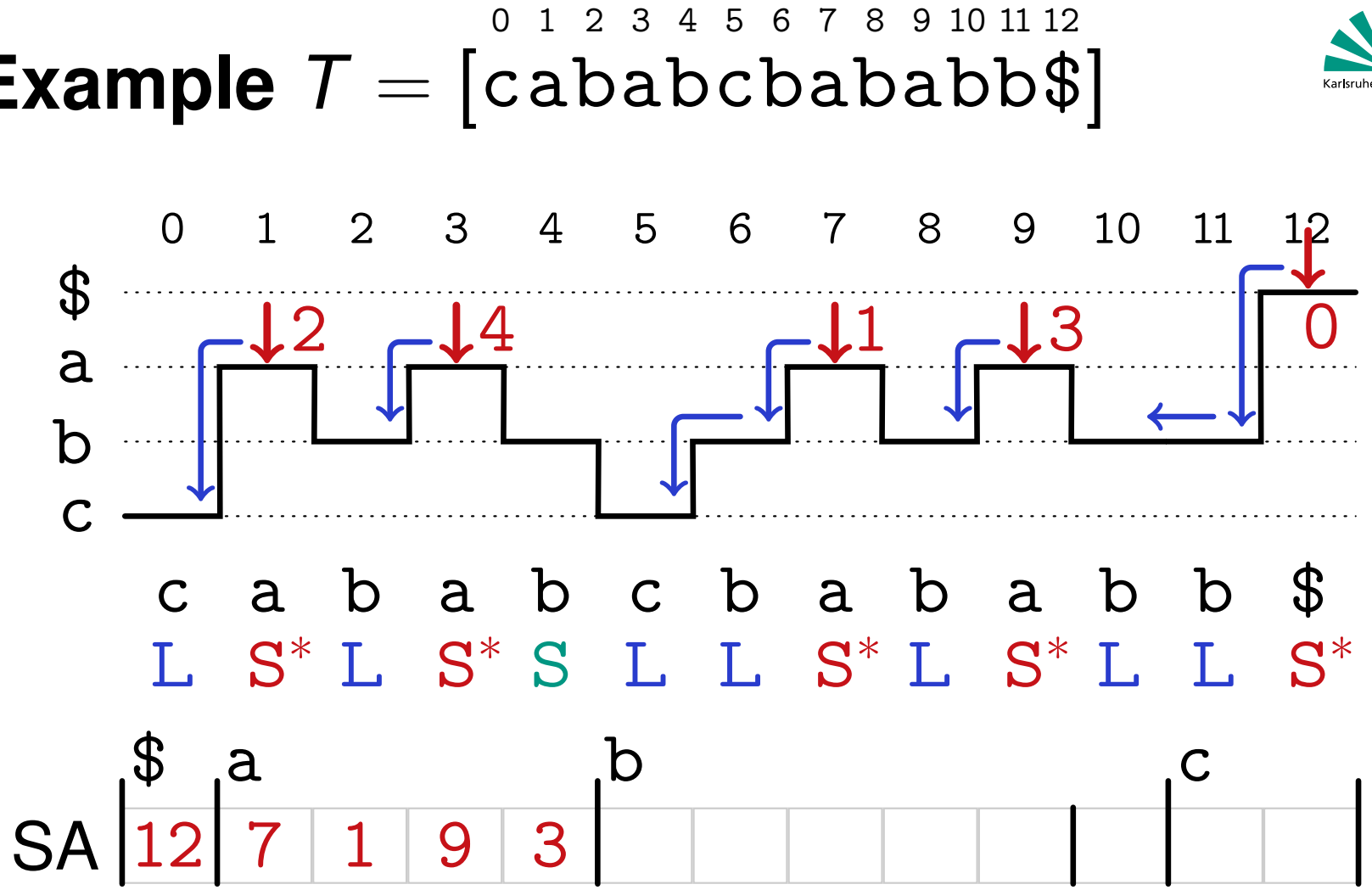
Example $T = [\texttt{cababcbababb\$}]$



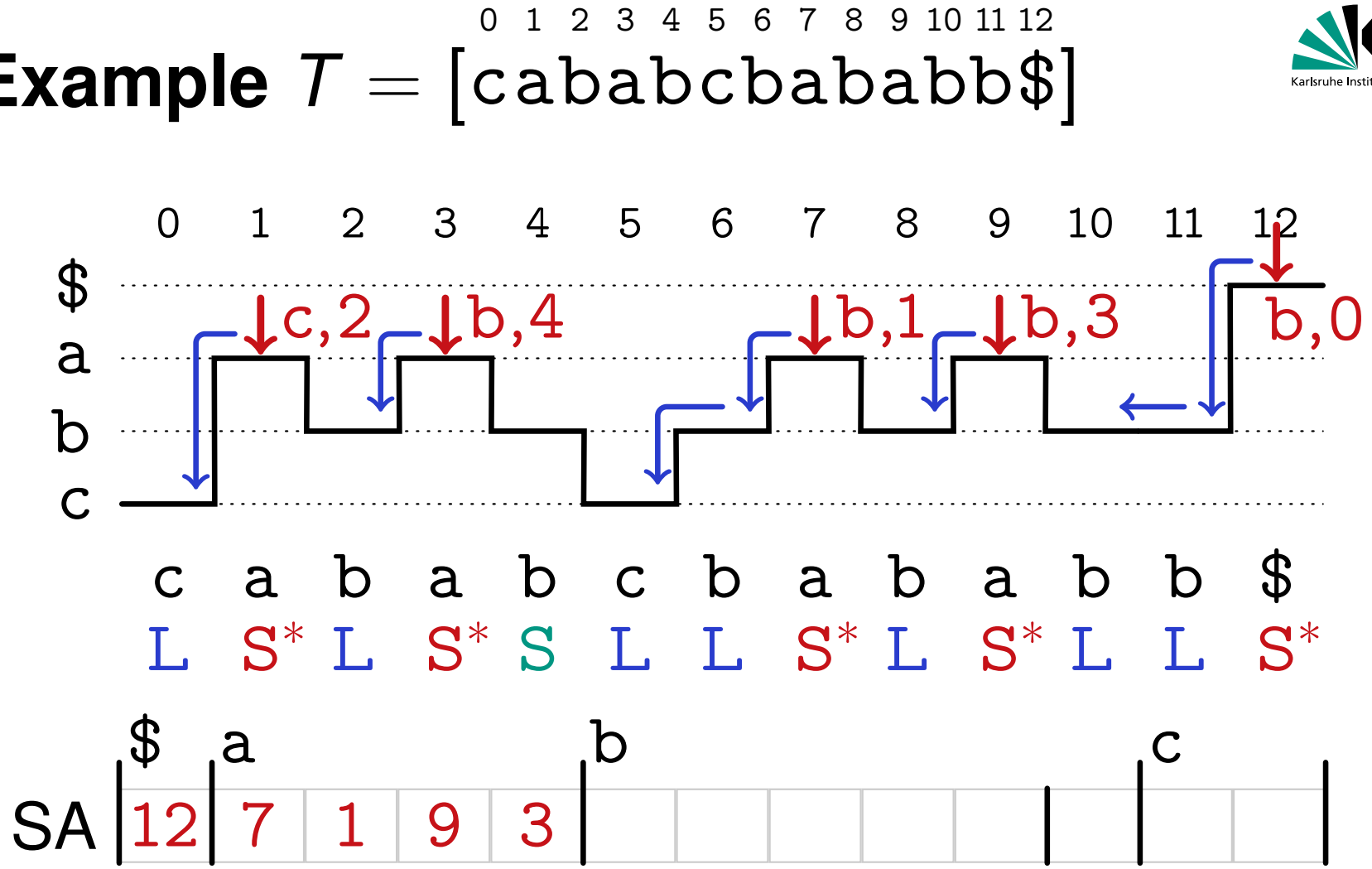
Example $T = [\texttt{cababcbababb\$}]$



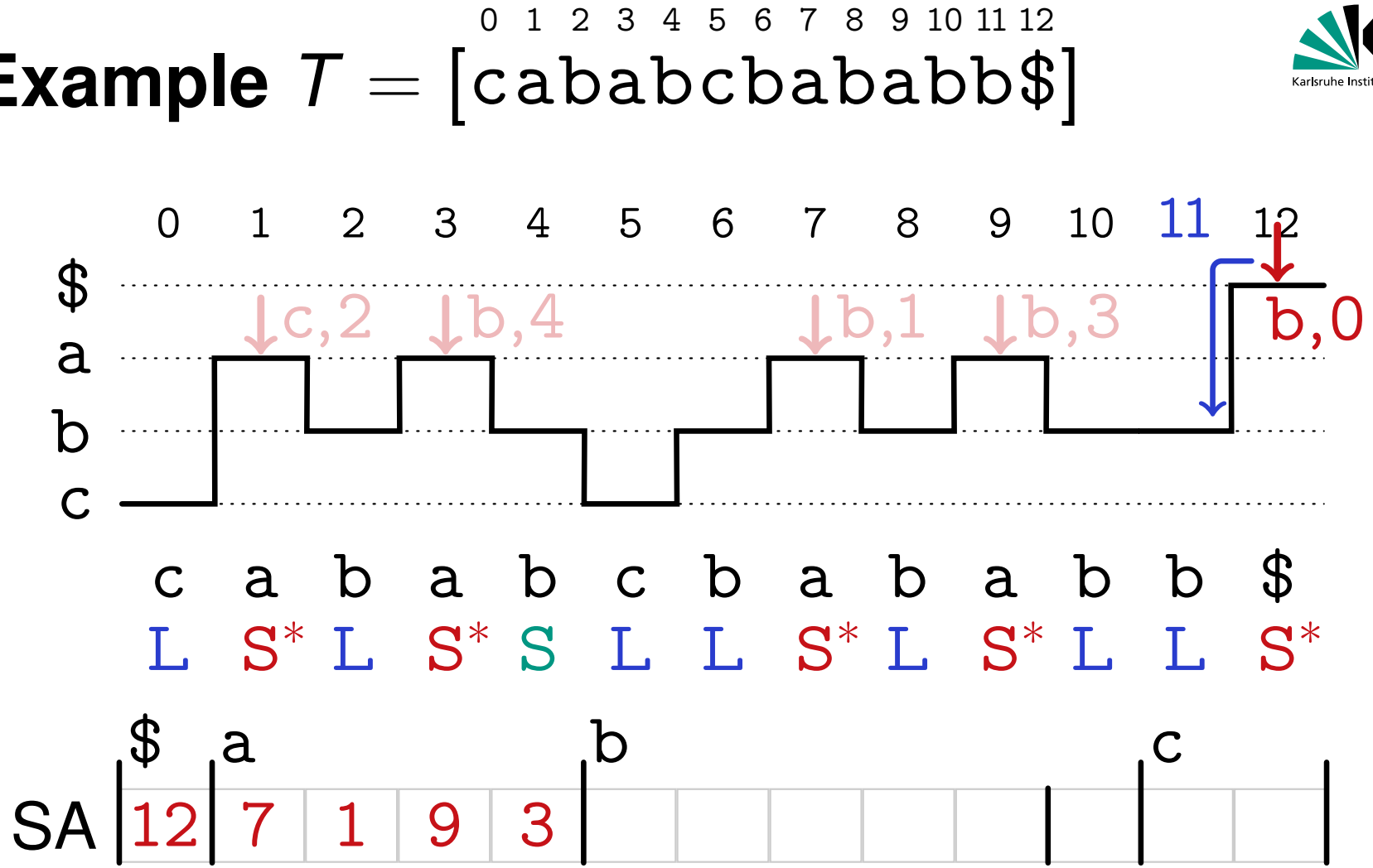
Example $T = [\texttt{cababcbababb\$}]$



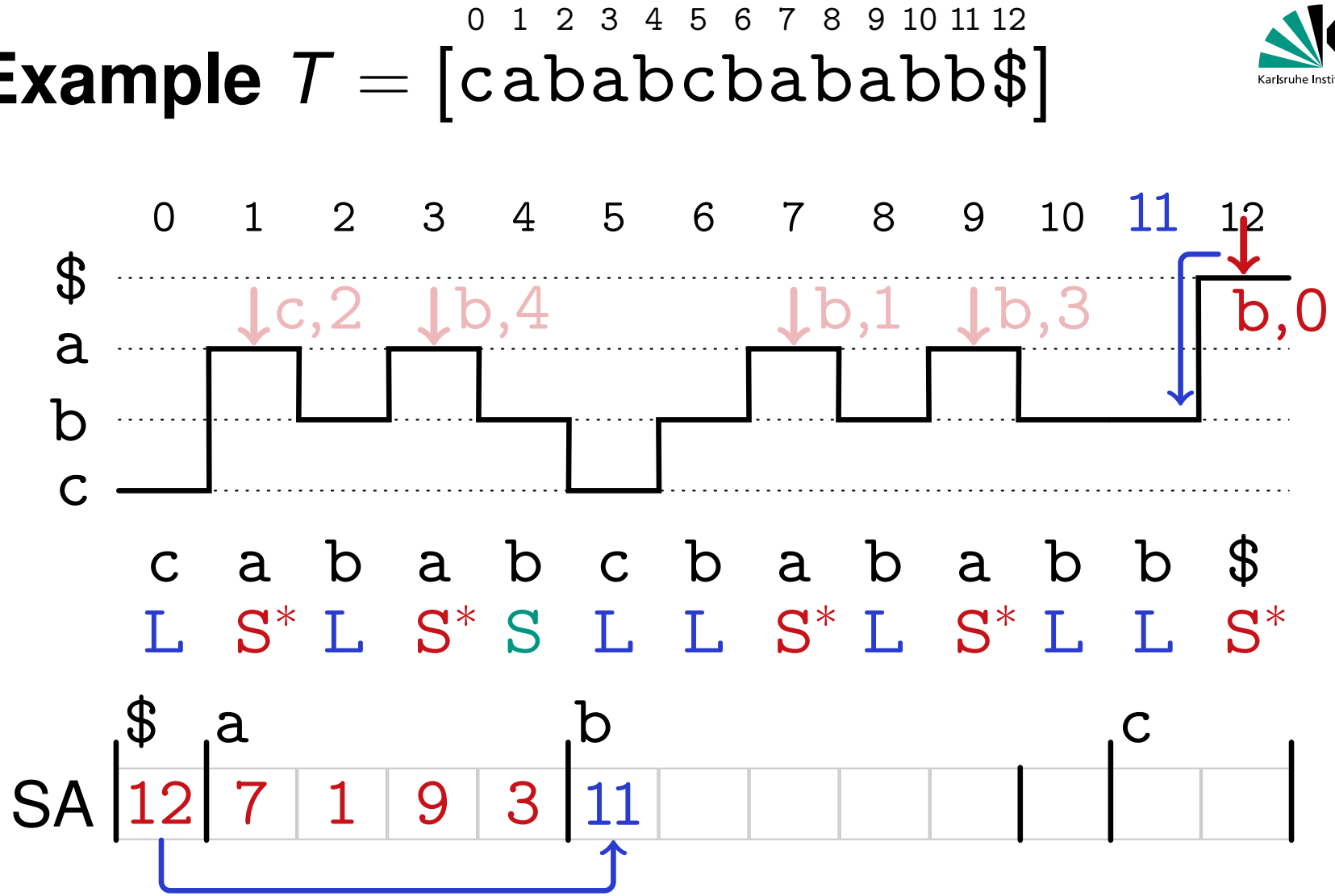
Example $T = [\texttt{cababcbababb\$}]$



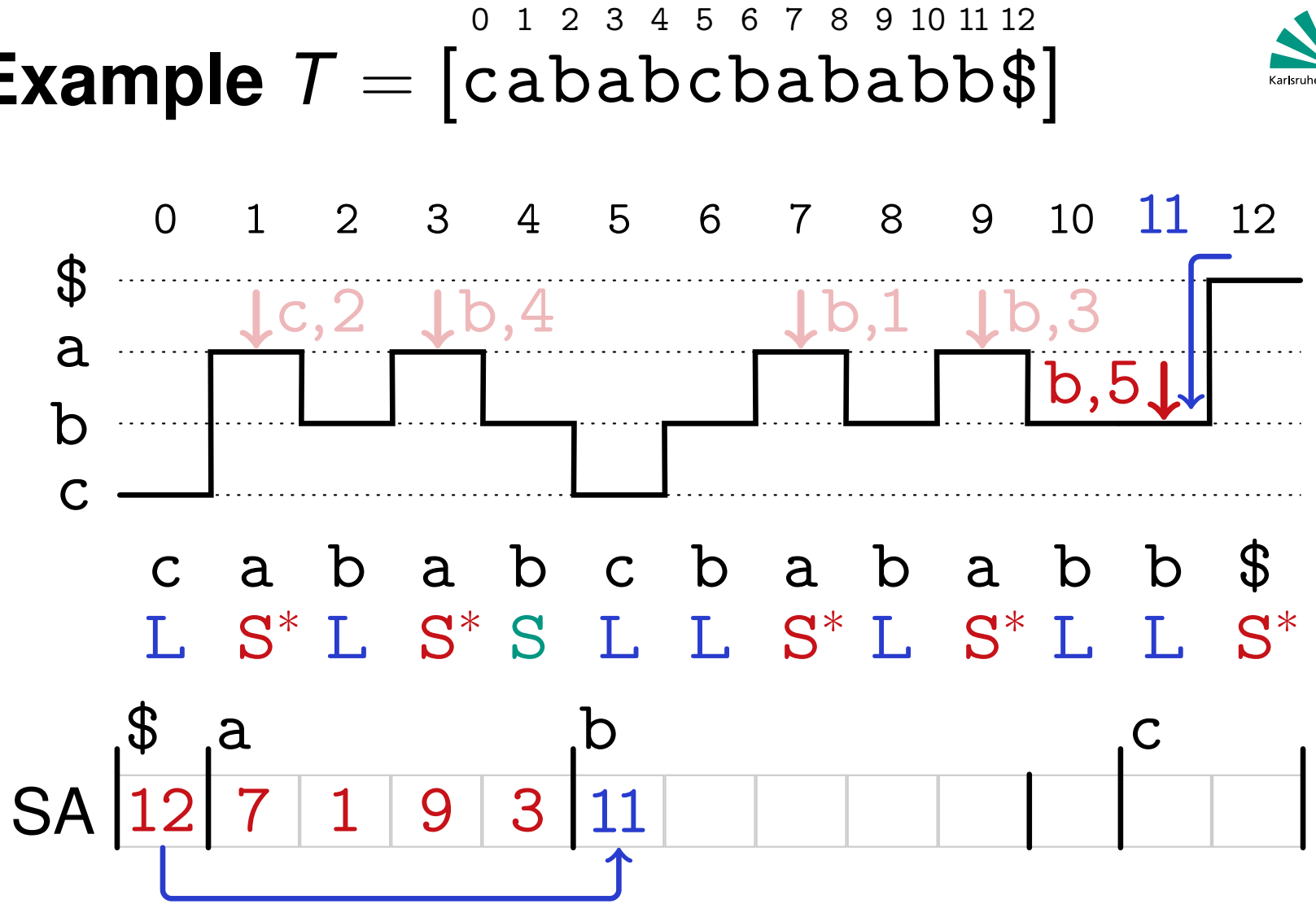
Example $T = [\texttt{cababcbababb\$}]$



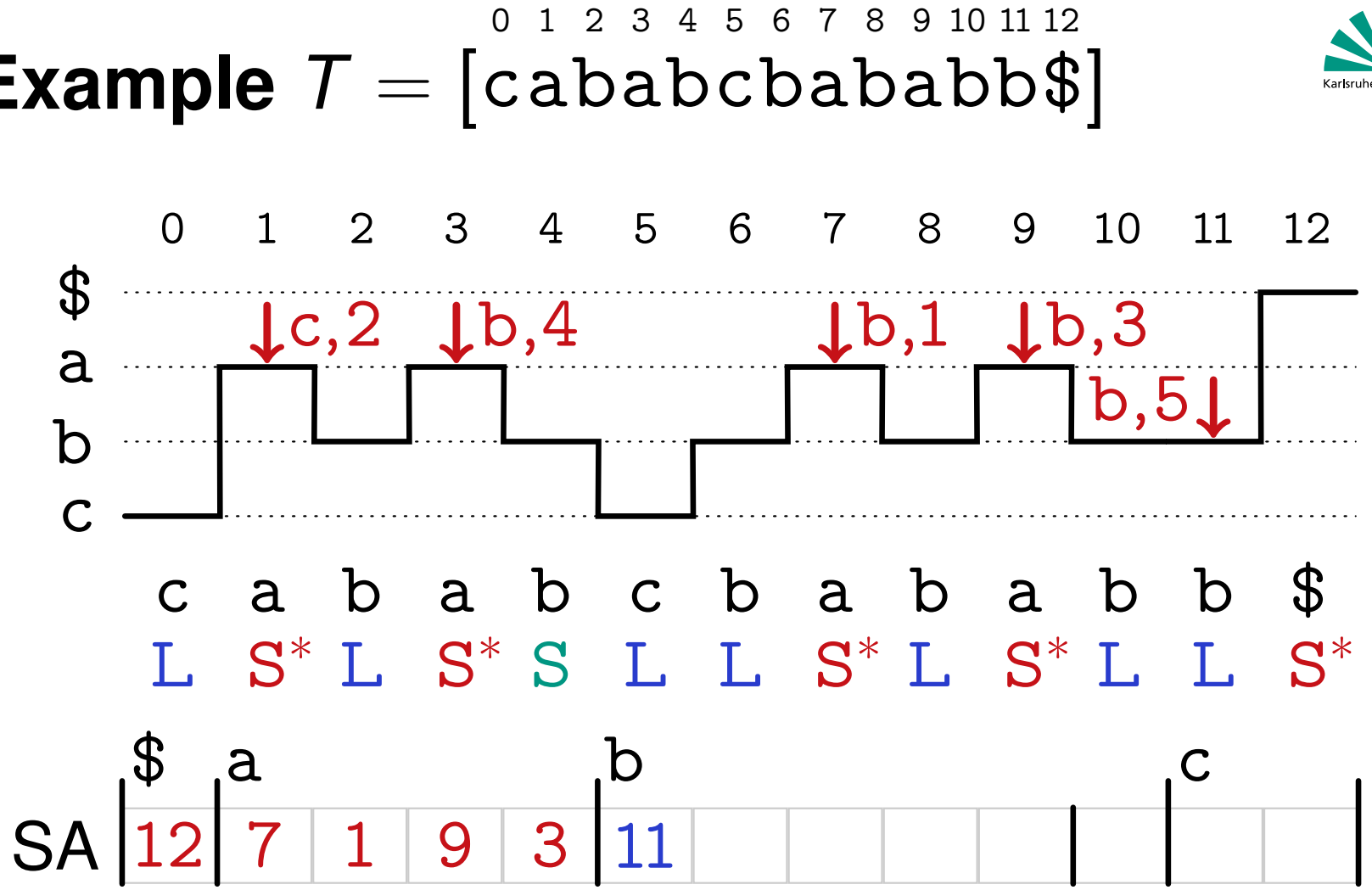
Example $T = [\texttt{cababcbababb\$}]$



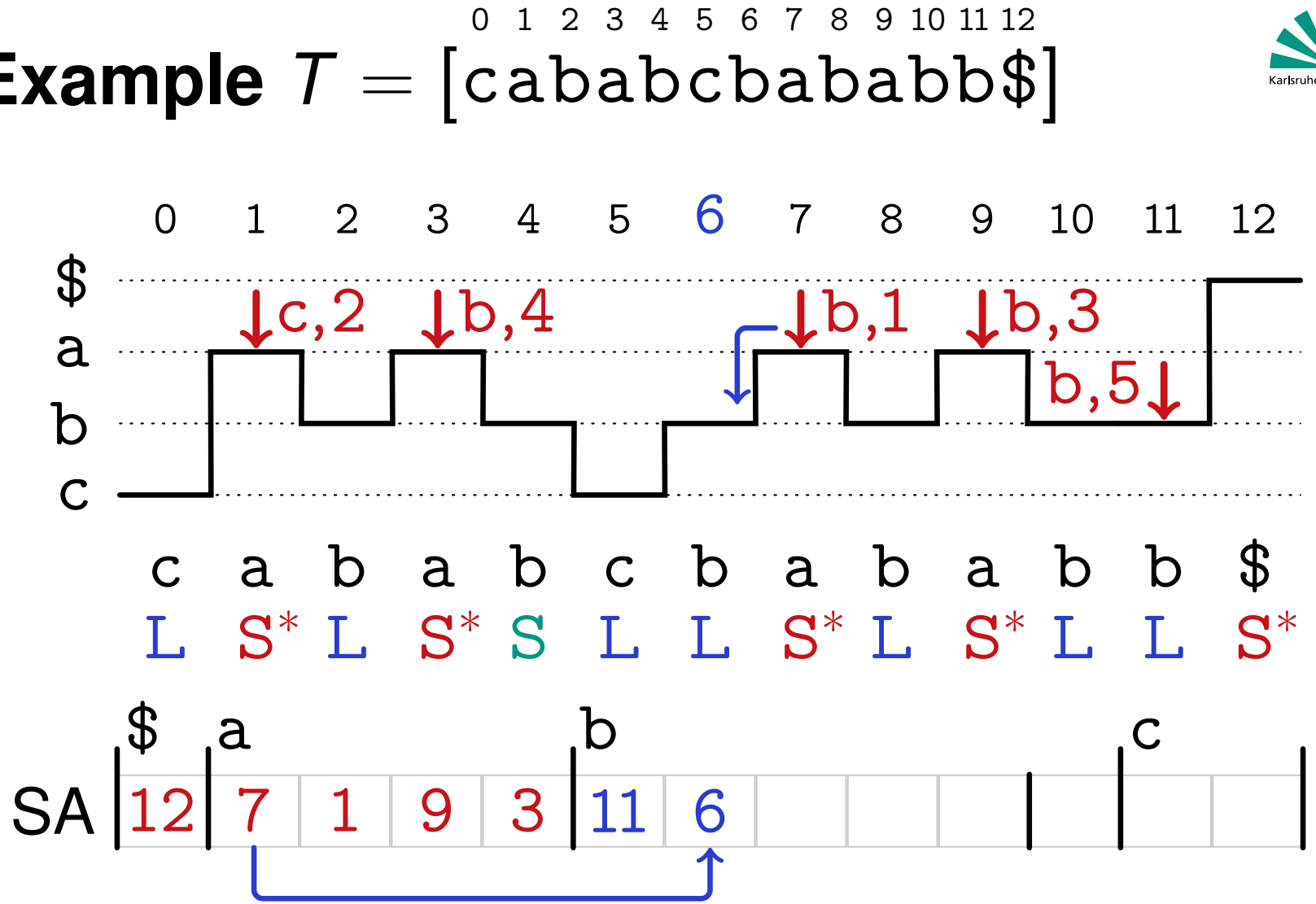
Example $T = [\texttt{cababcbababb\$}]$



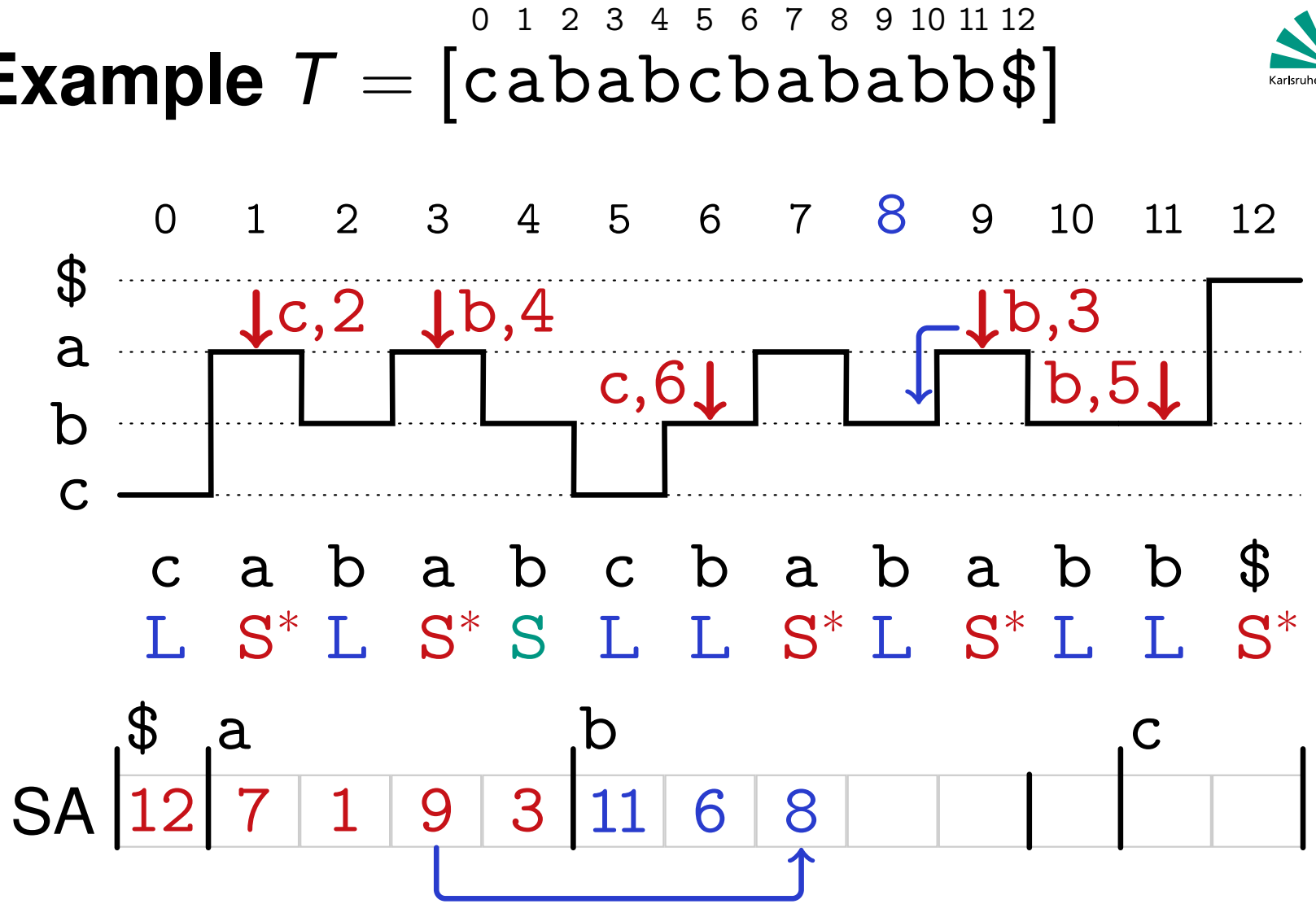
Example $T = [\texttt{cababcbababb\$}]$



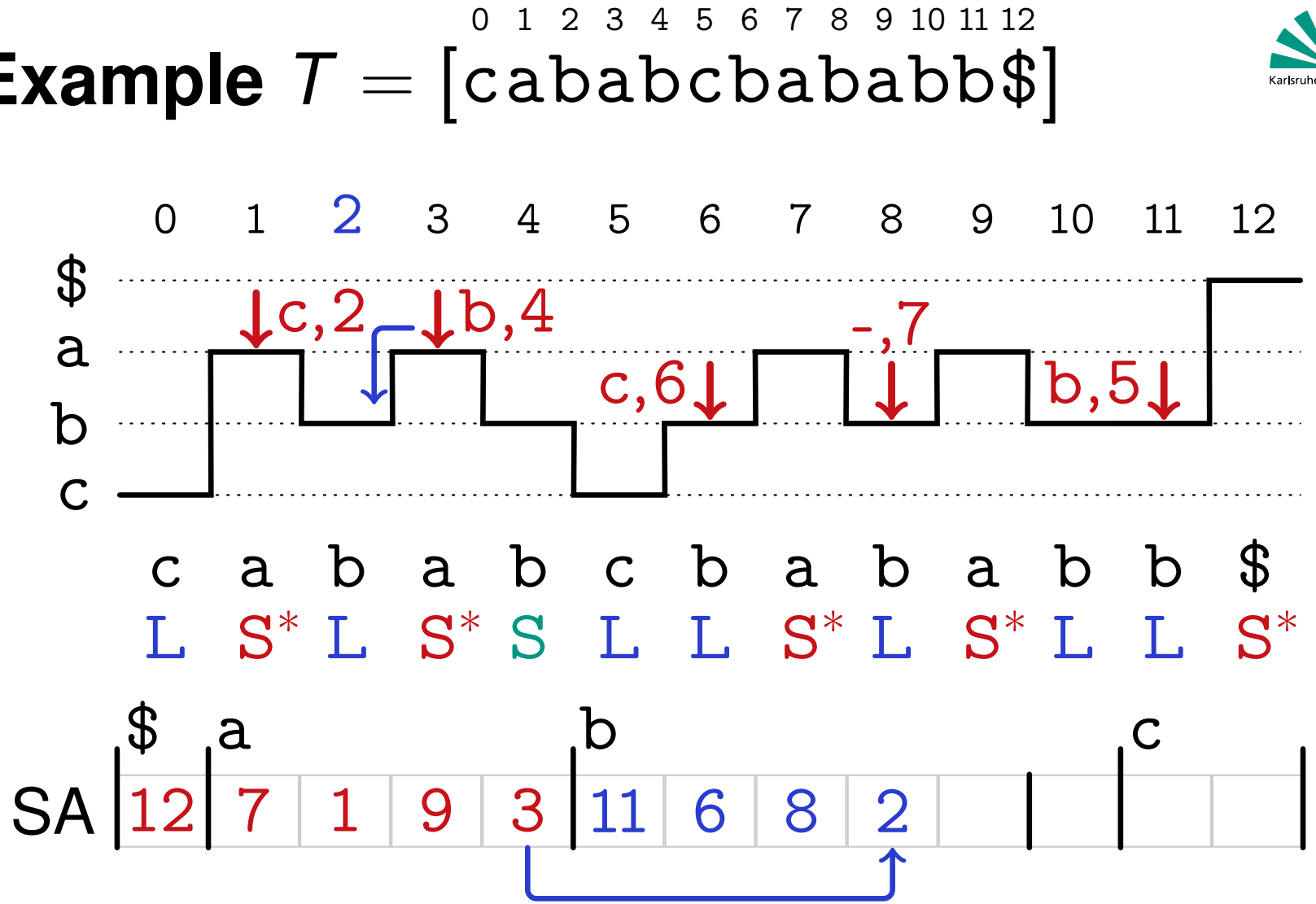
Example $T = [\texttt{cababcbababb\$}]$



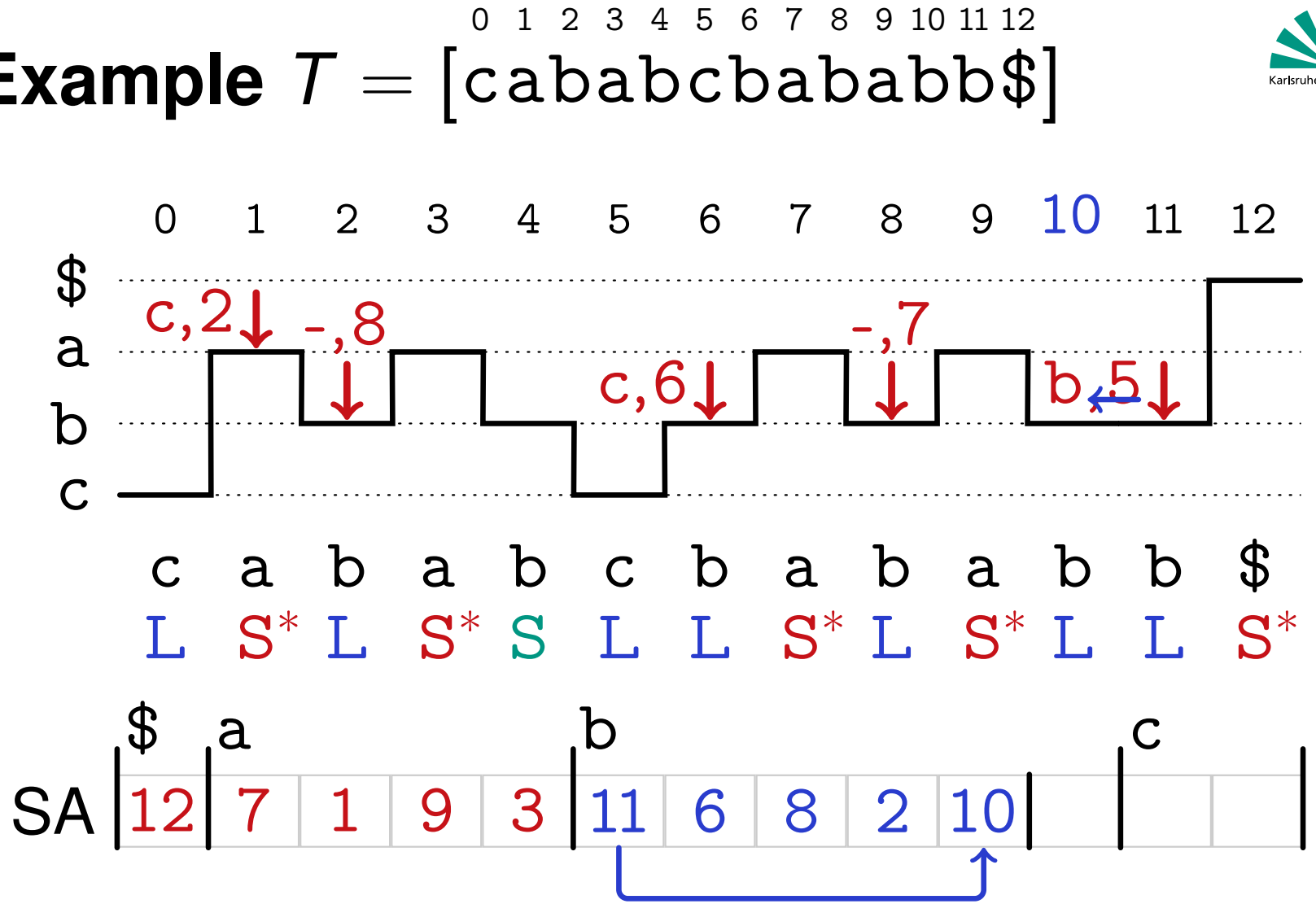
Example $T = [\texttt{cababcbababb\$}]$



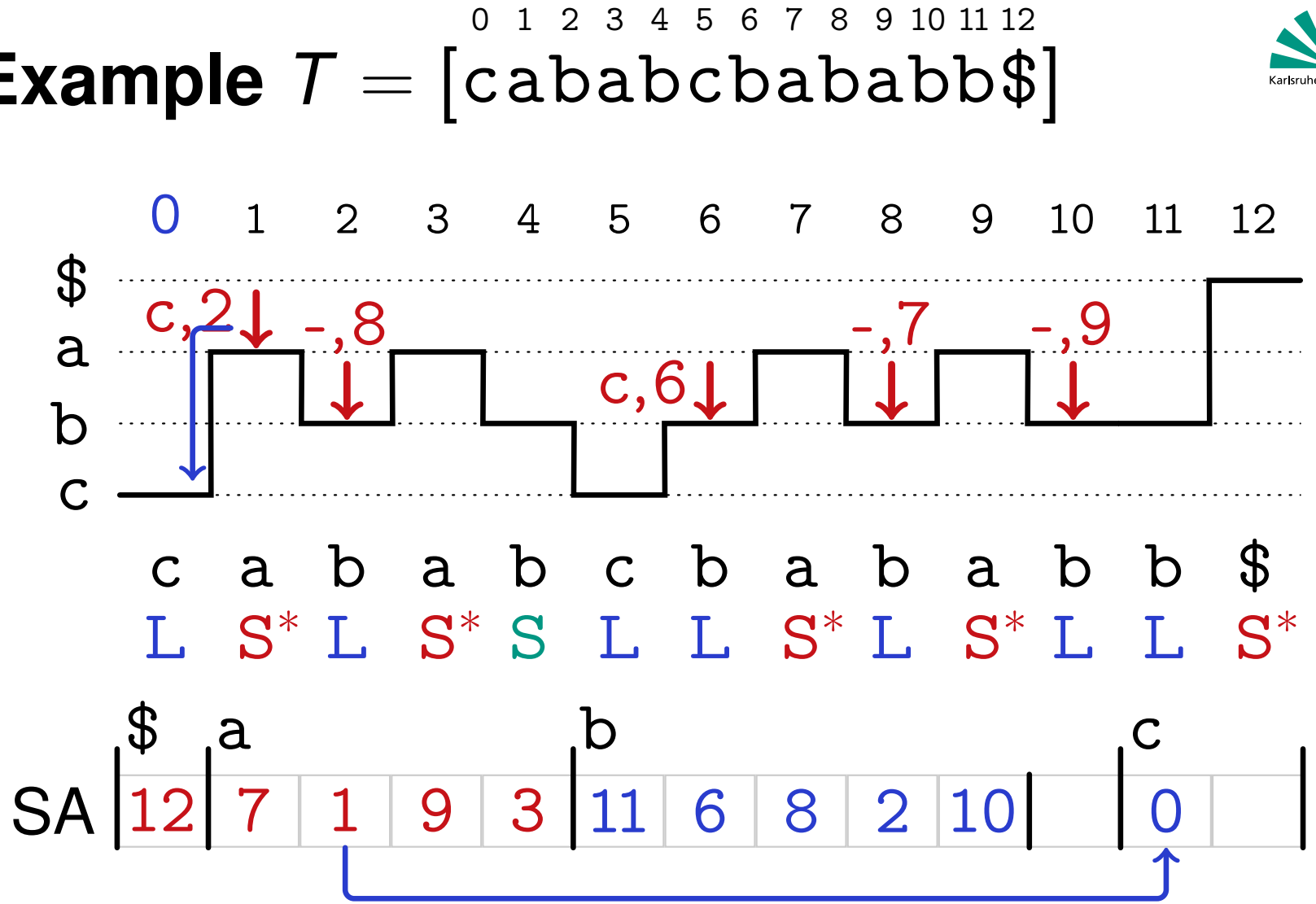
Example $T = [\texttt{cababcbababb\$}]$



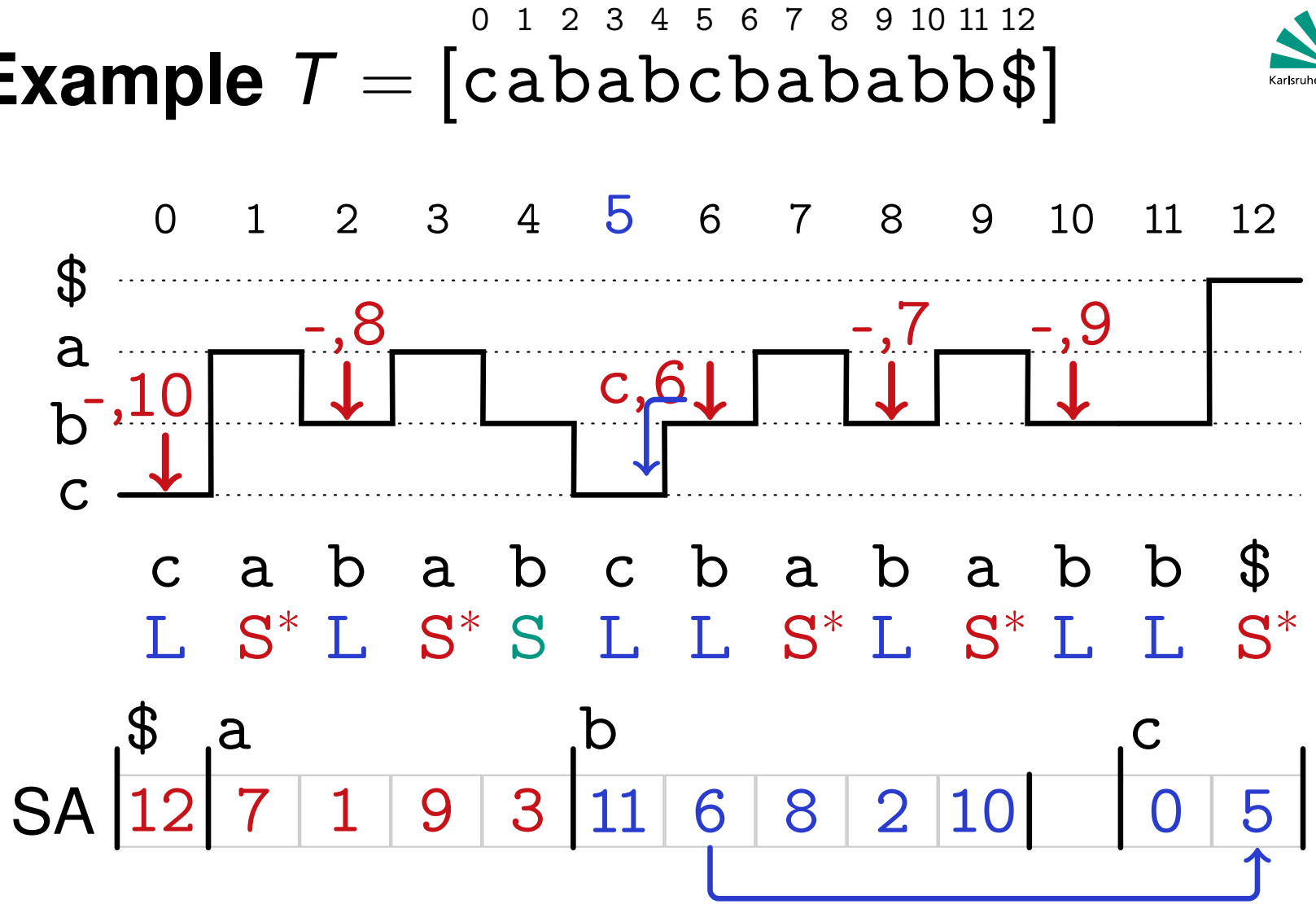
Example $T = [\texttt{cababcbababb\$}]$



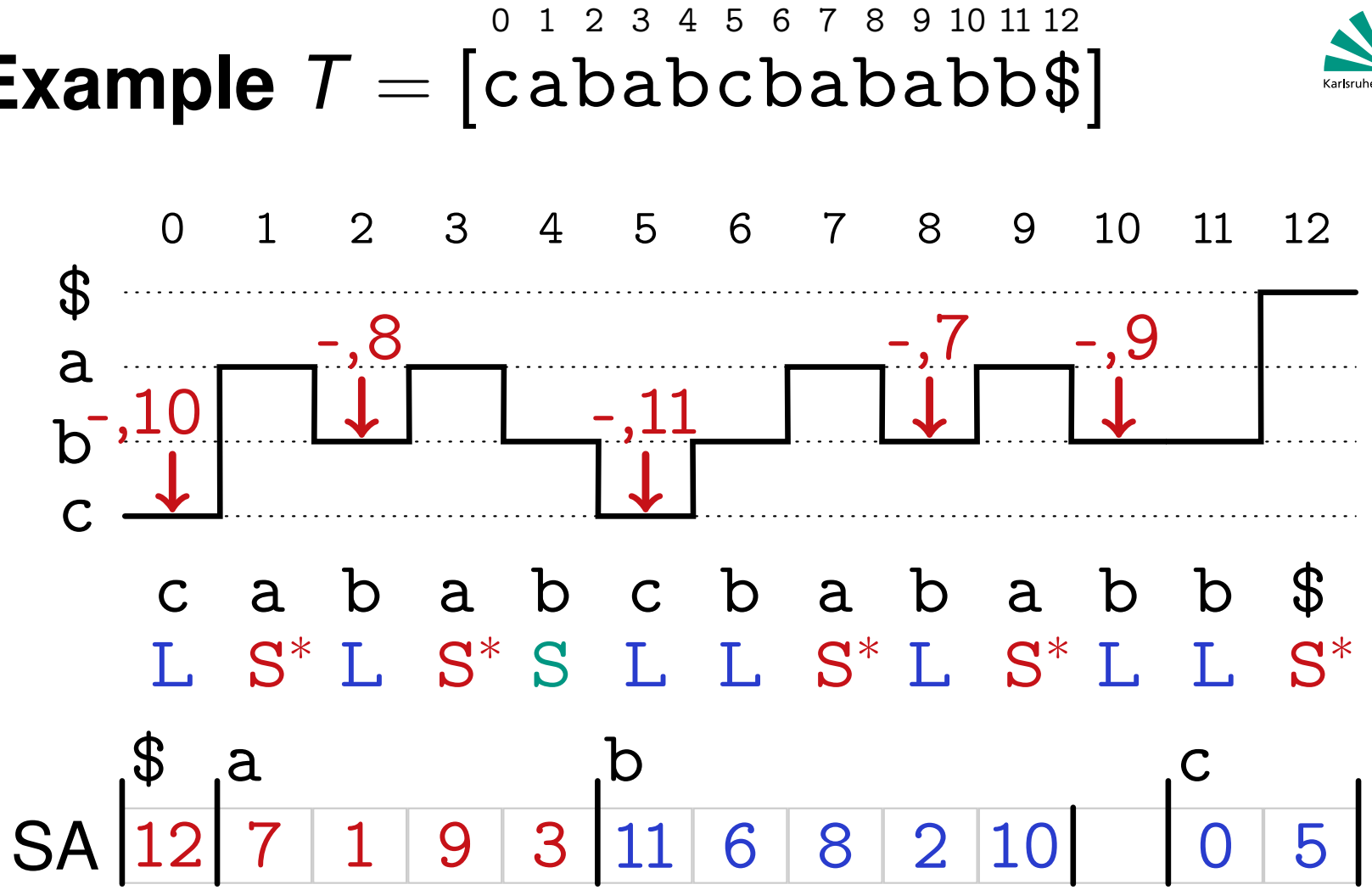
Example $T = [\texttt{cababcbababb\$}]$



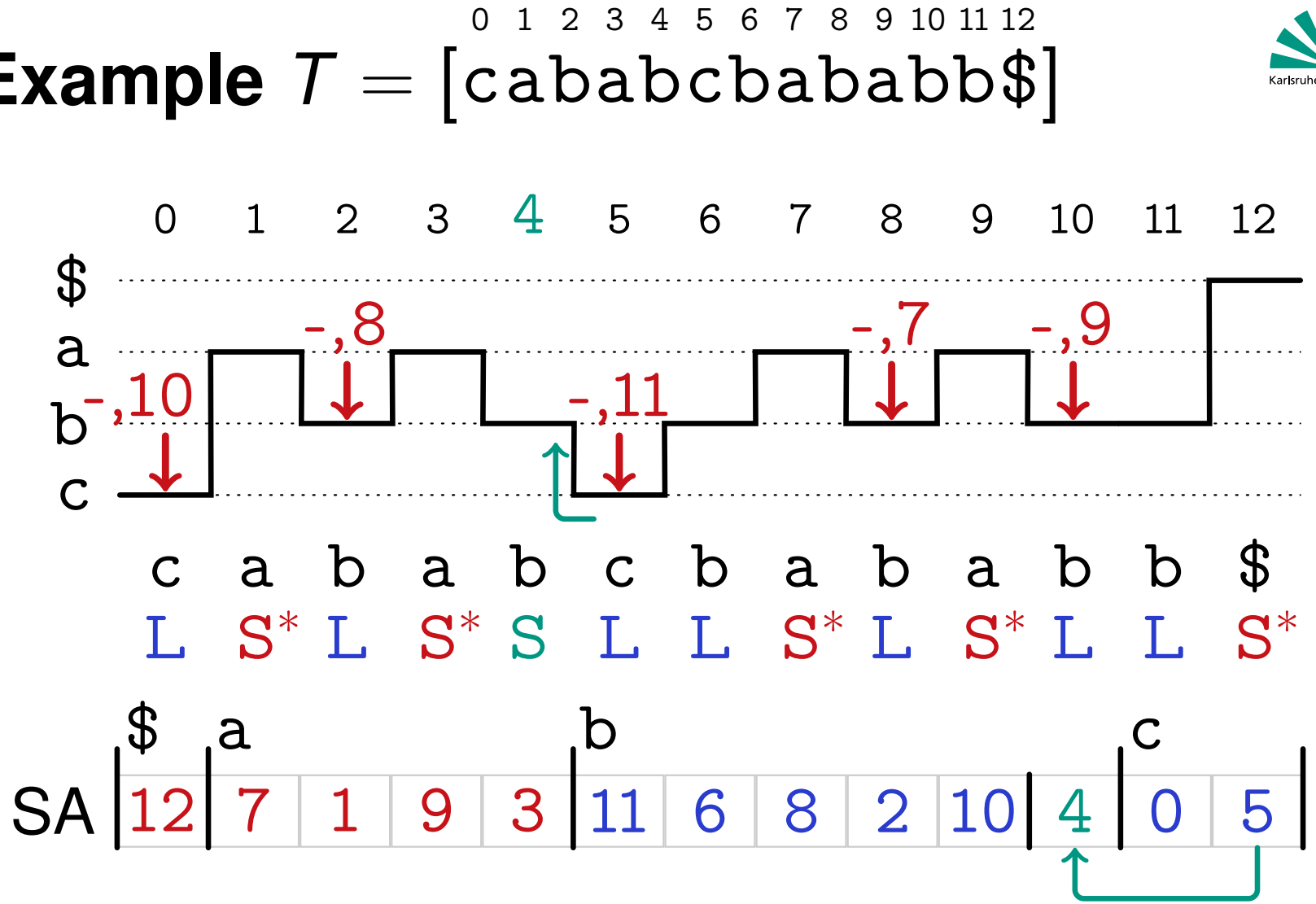
Example $T = [\texttt{cababcbababb\$}]$



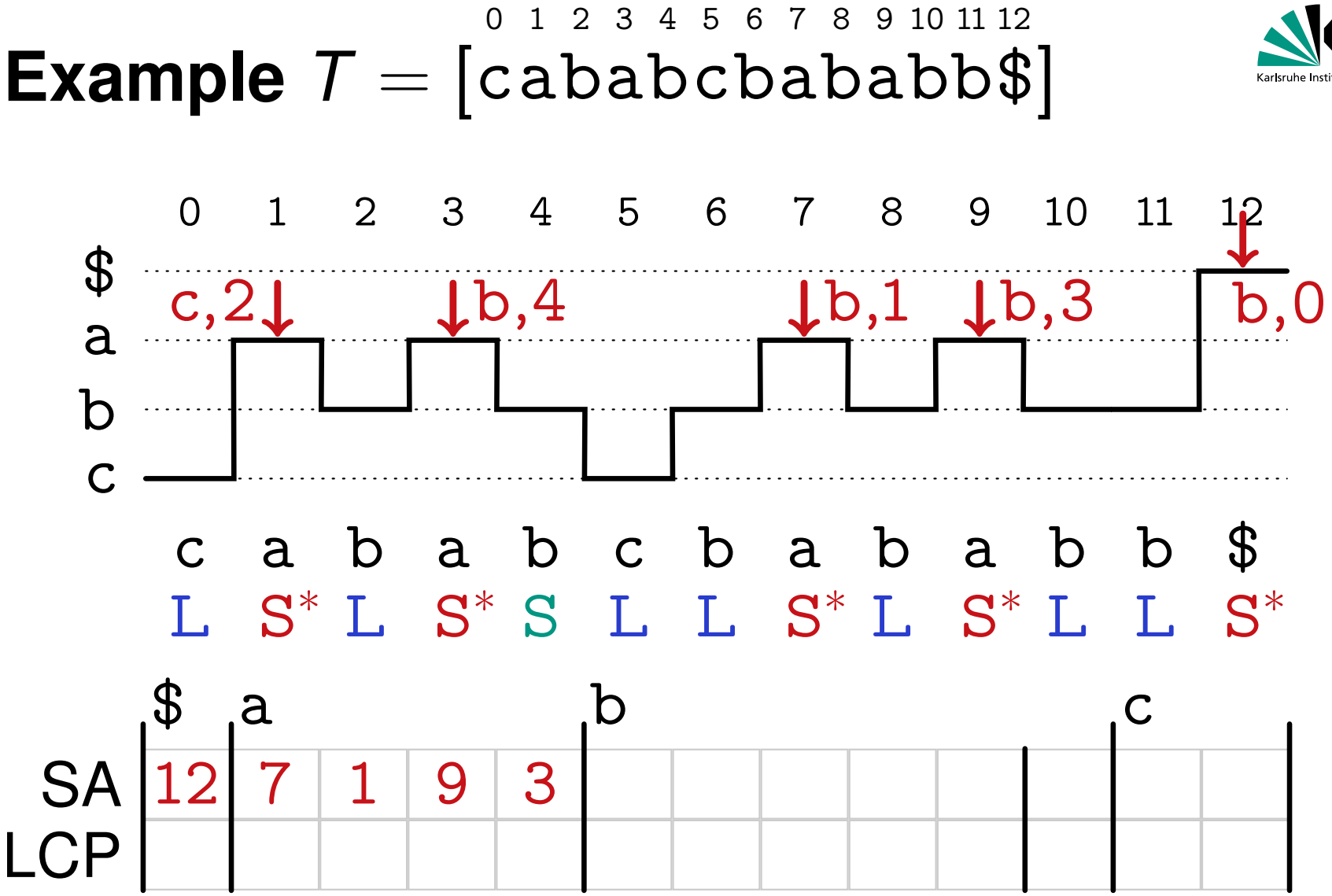
Example $T = [\texttt{cababcbababb\$}]$



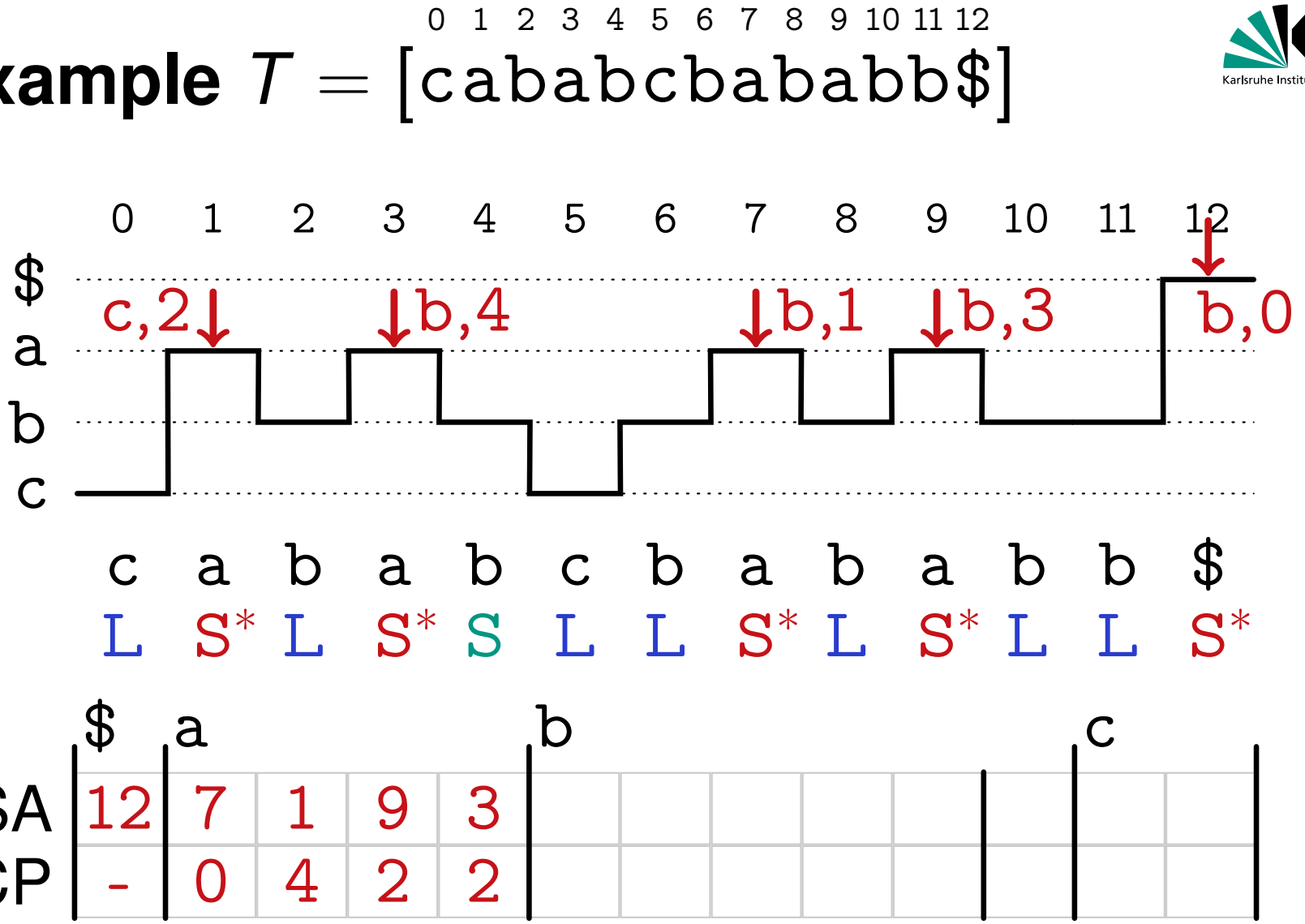
Example $T = [\texttt{cababcbababb\$}]$



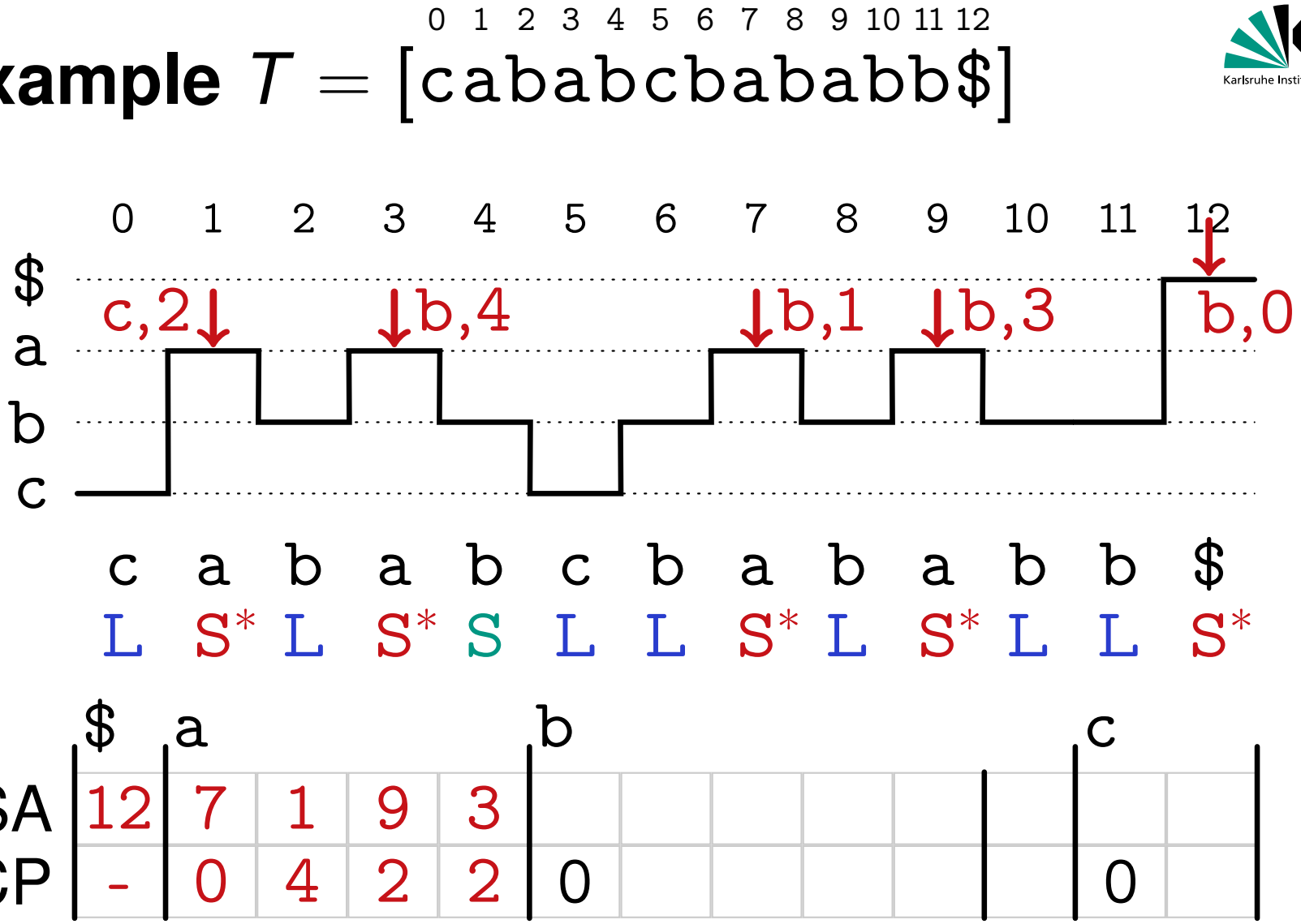
Example $T = [\texttt{cababcbababb\$}]$



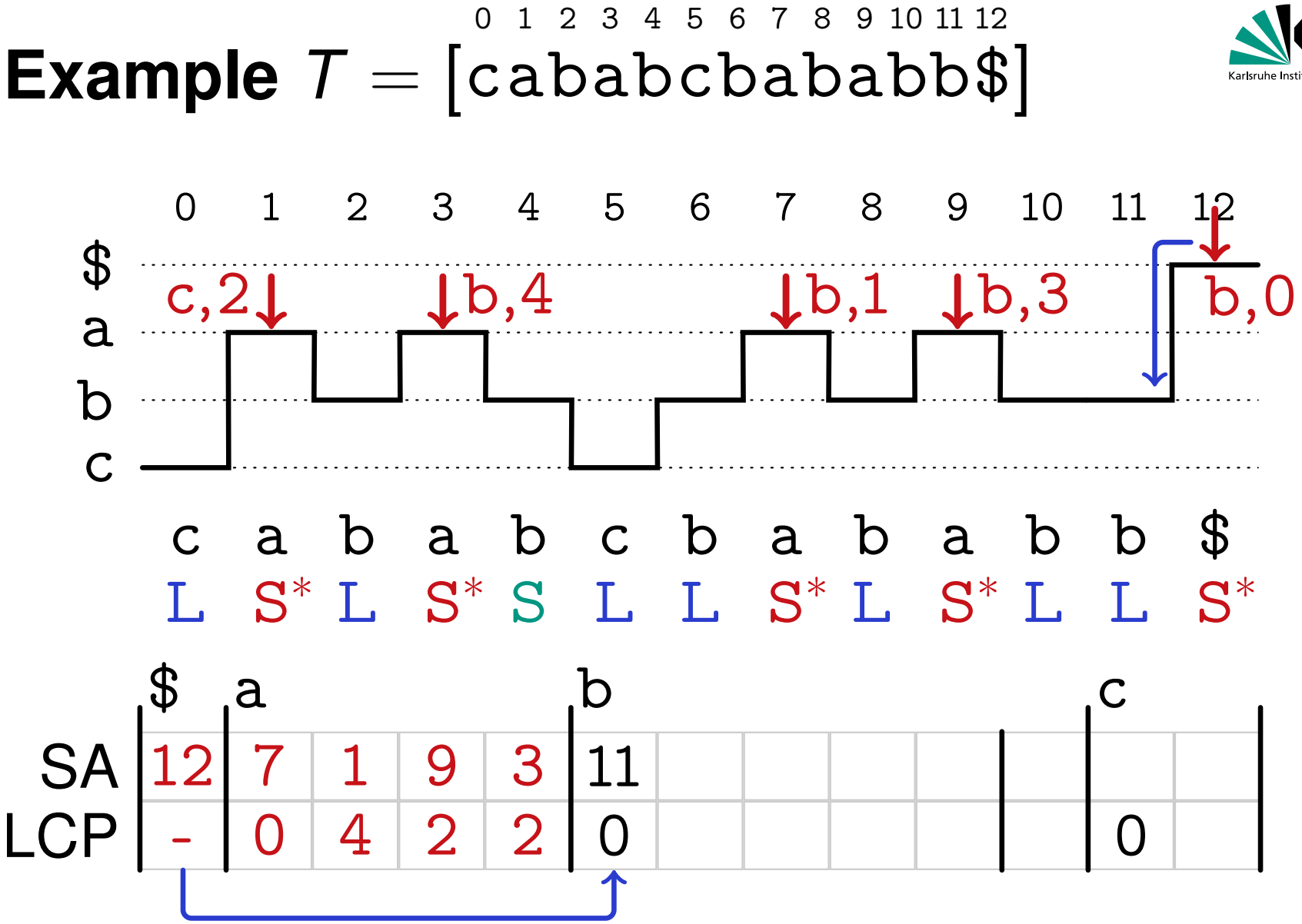
Example $T = [\texttt{cababcbababb\$}]$



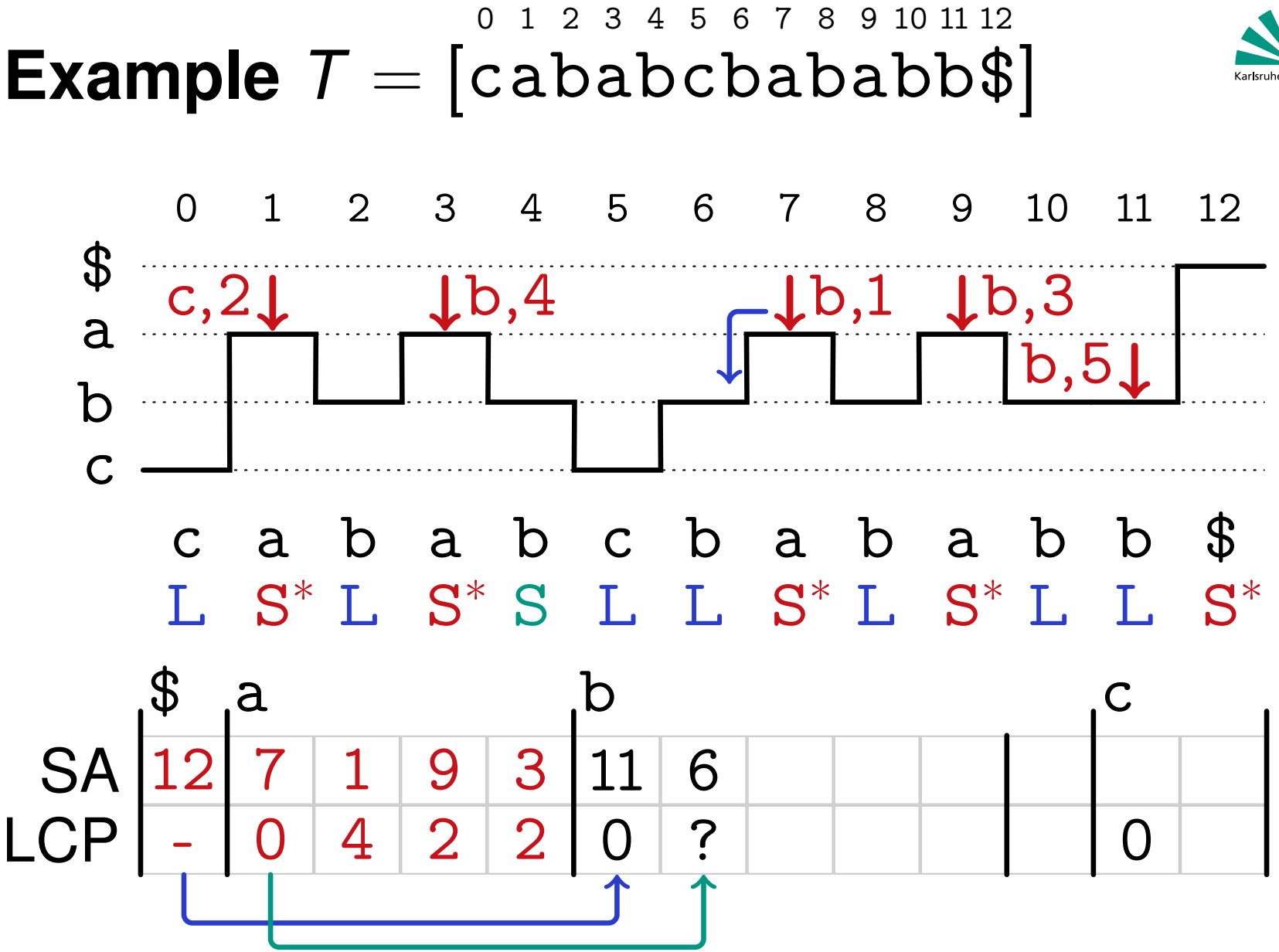
Example $T = [\texttt{cababcbababb\$}]$



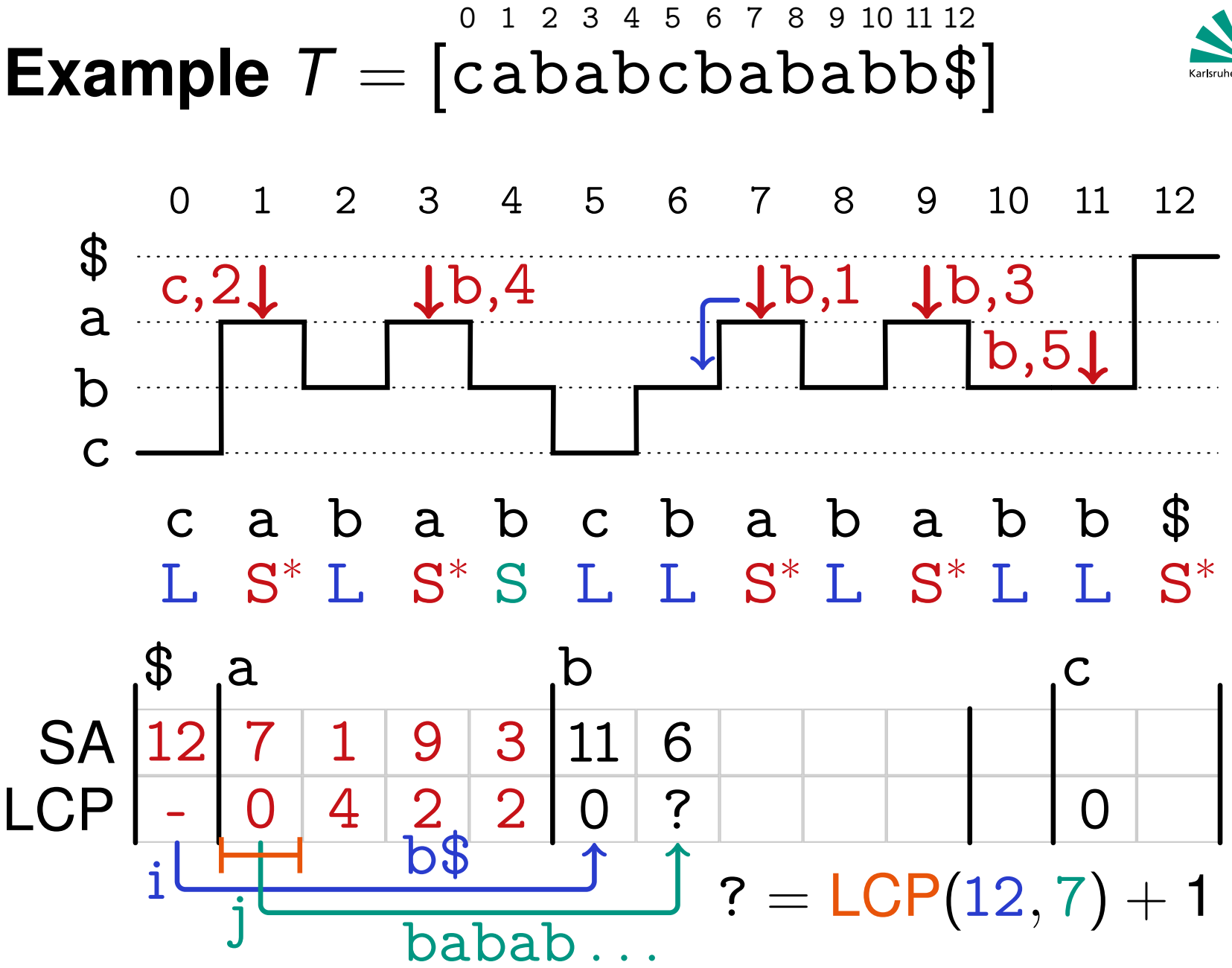
Example $T = [\texttt{cababcbababb\$}]$



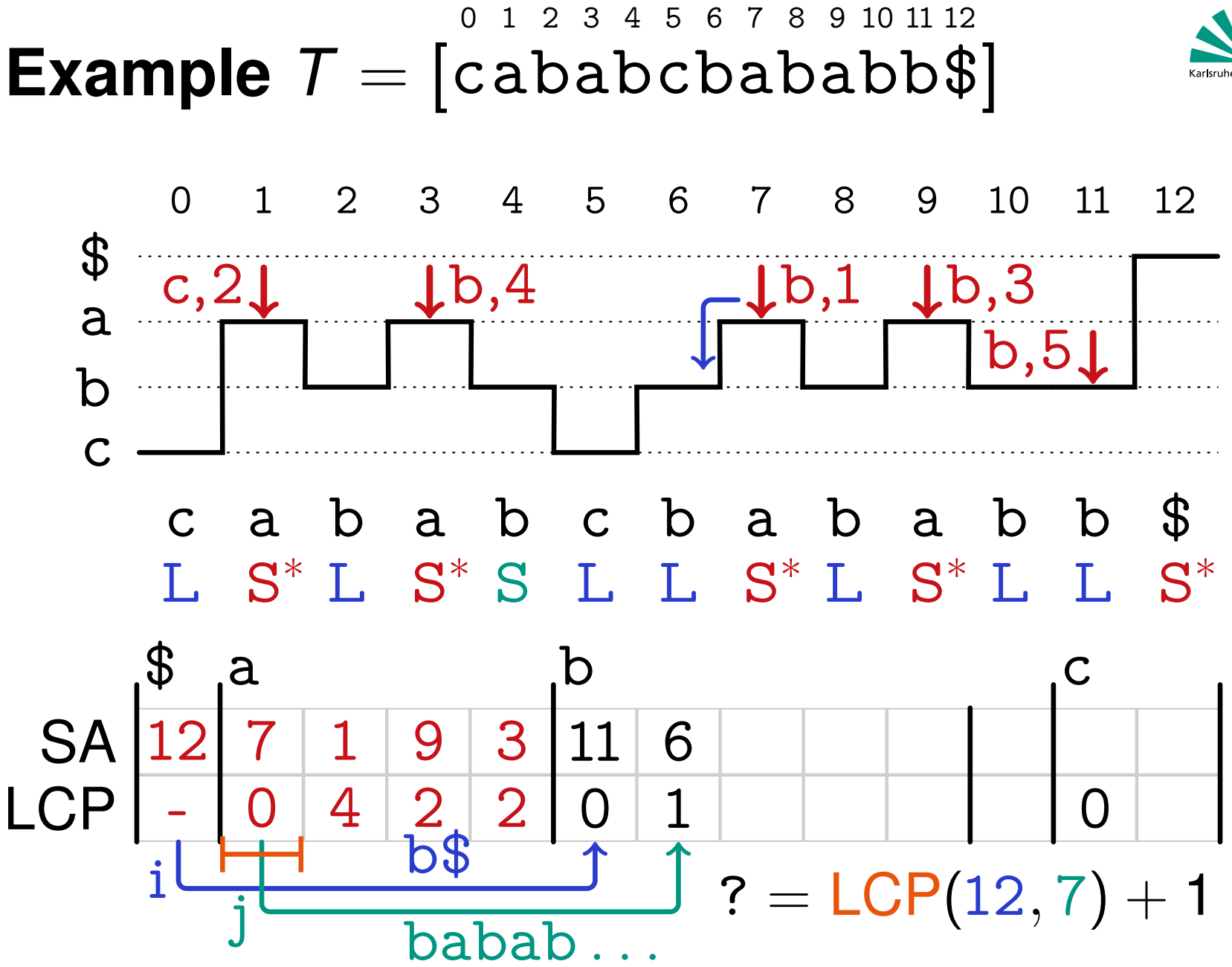
Example $T = [\texttt{cababcbababb\$}]$



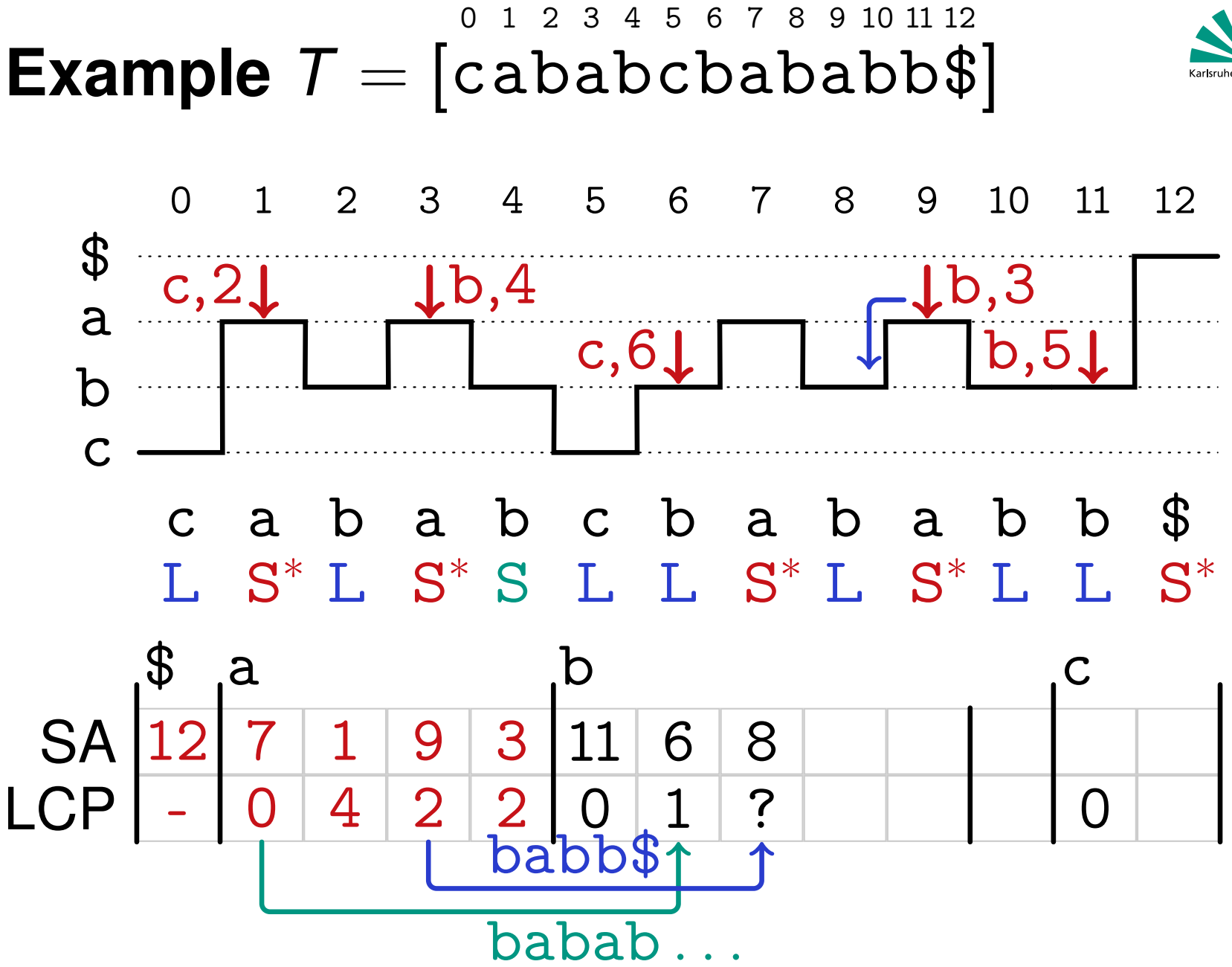
Example $T =$ [cababcbabab$]



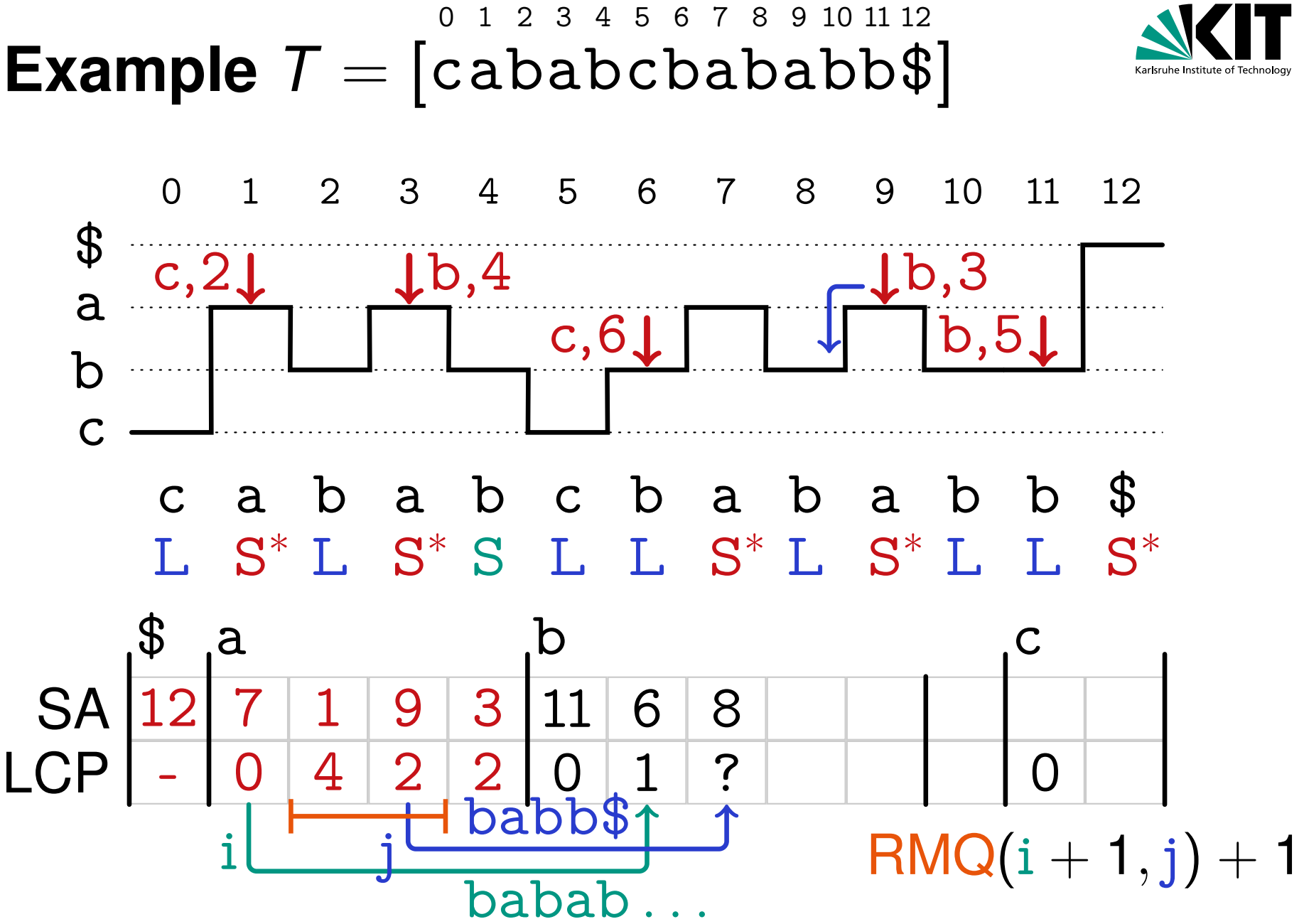
Example $T = [\texttt{cababcbabab\$}]$



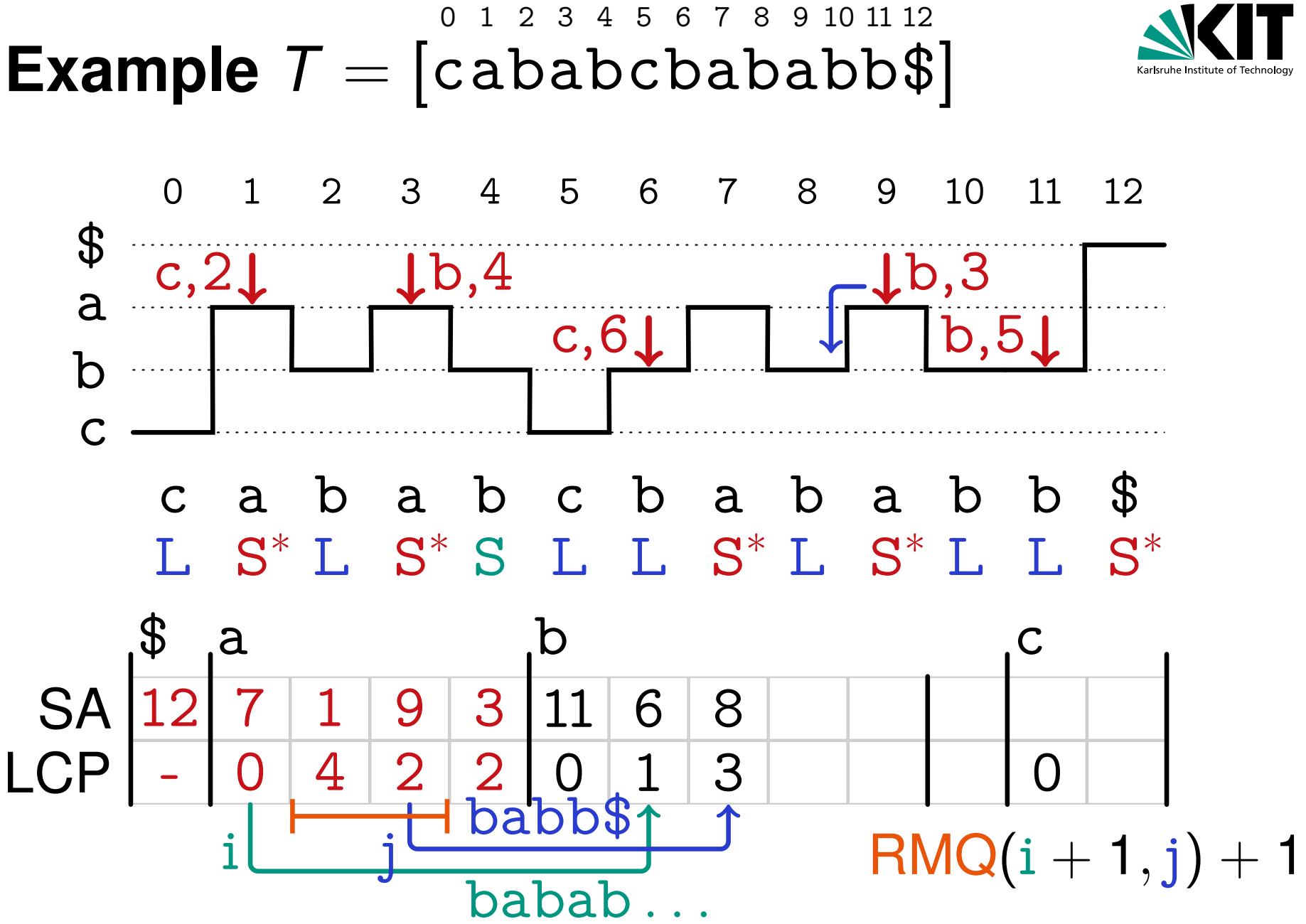
Example $T = [\texttt{cababcbababb\$}]$



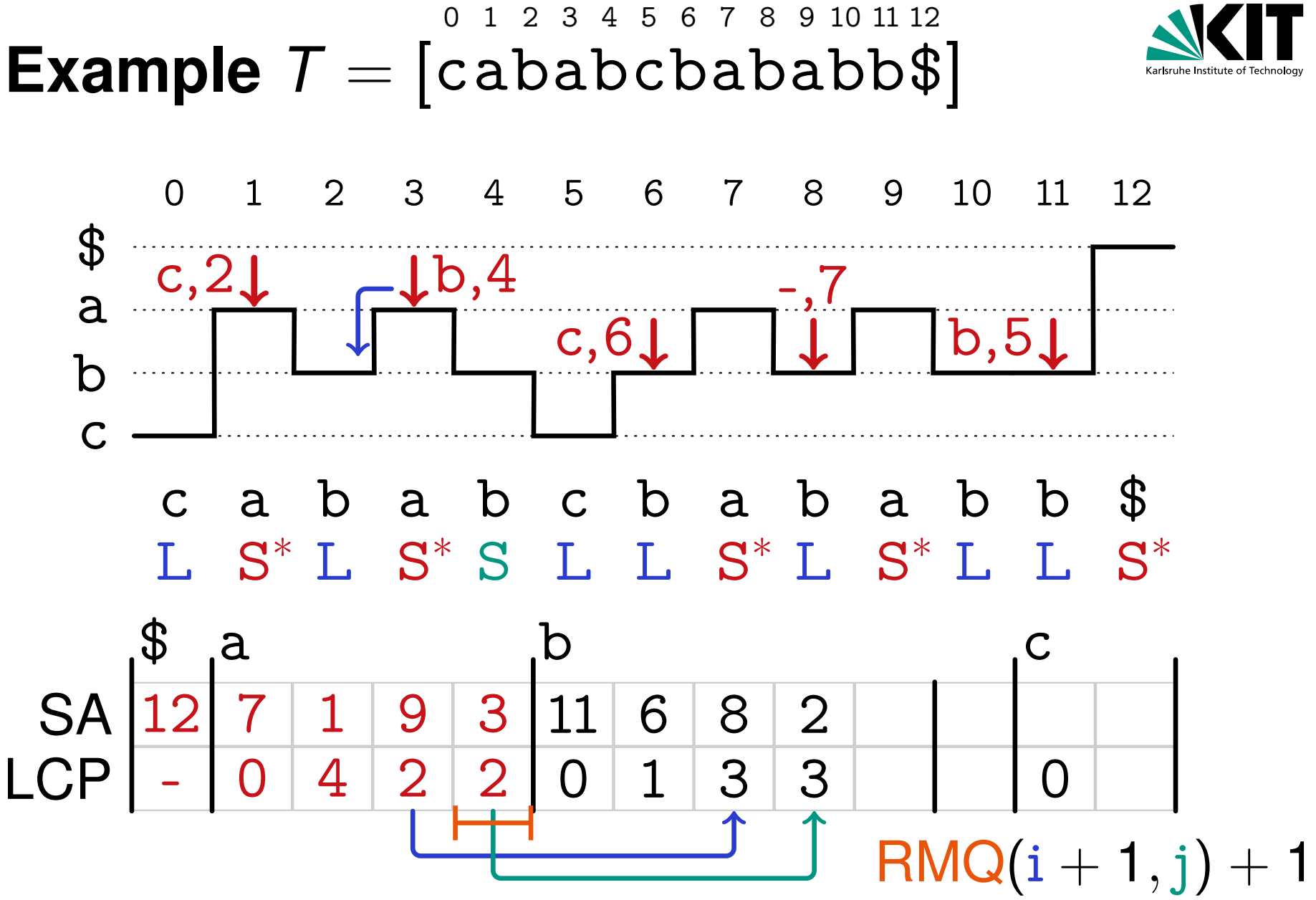
Example $T = [\texttt{cababcbababb\$}]$



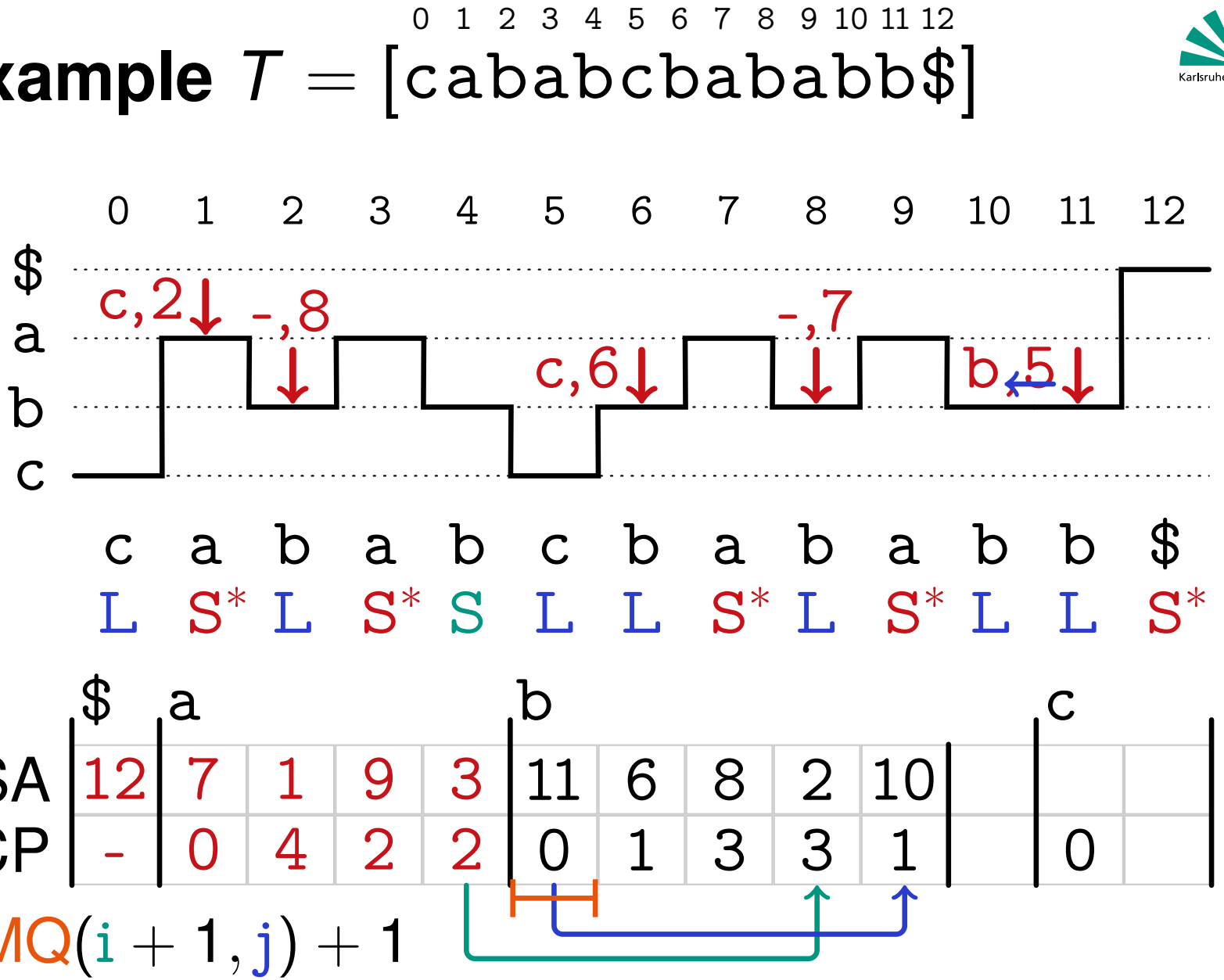
Example $T = [\texttt{cababcbababb\$}]$



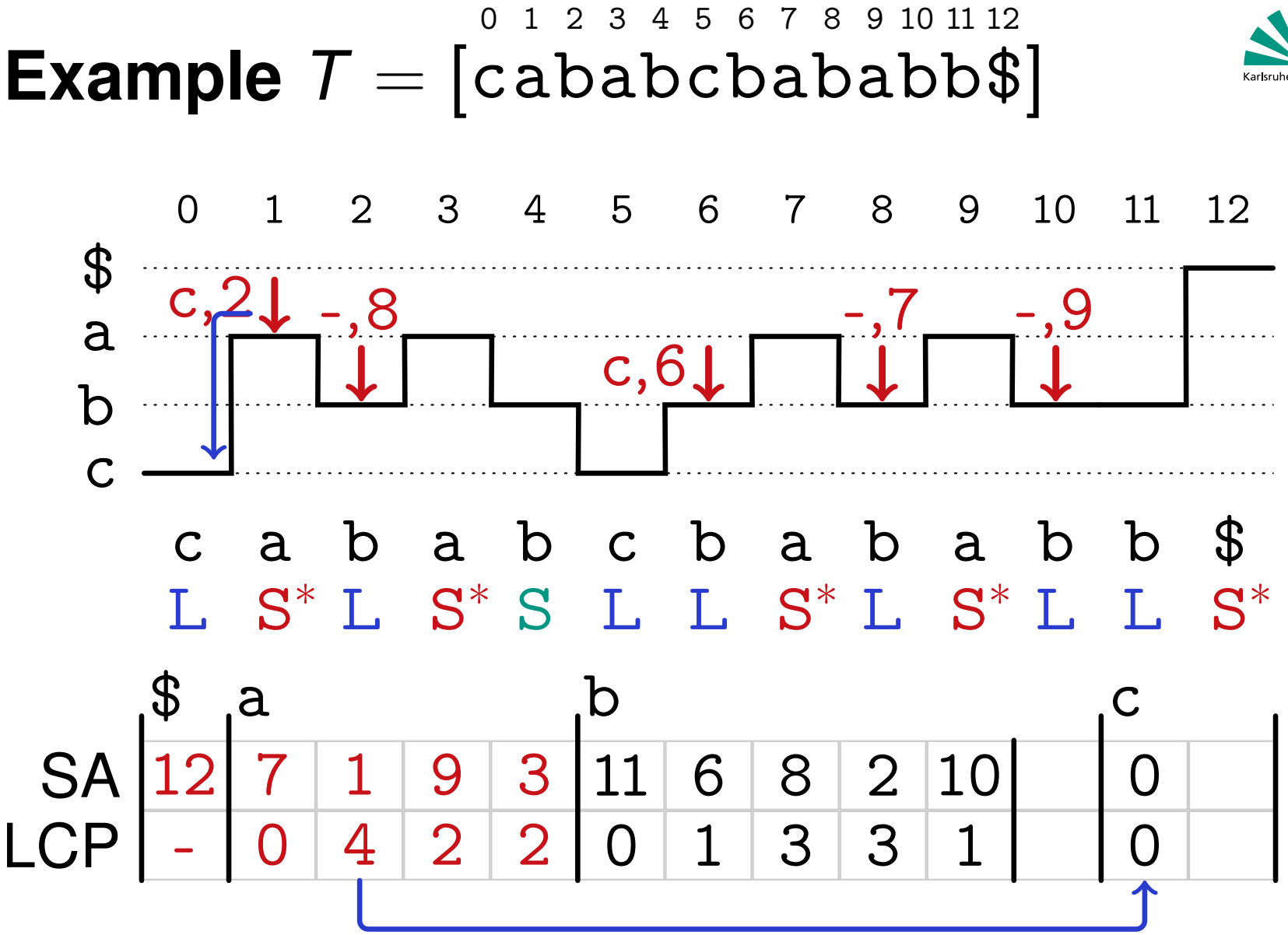
Example $T = [\texttt{cababcbababb\$}]$



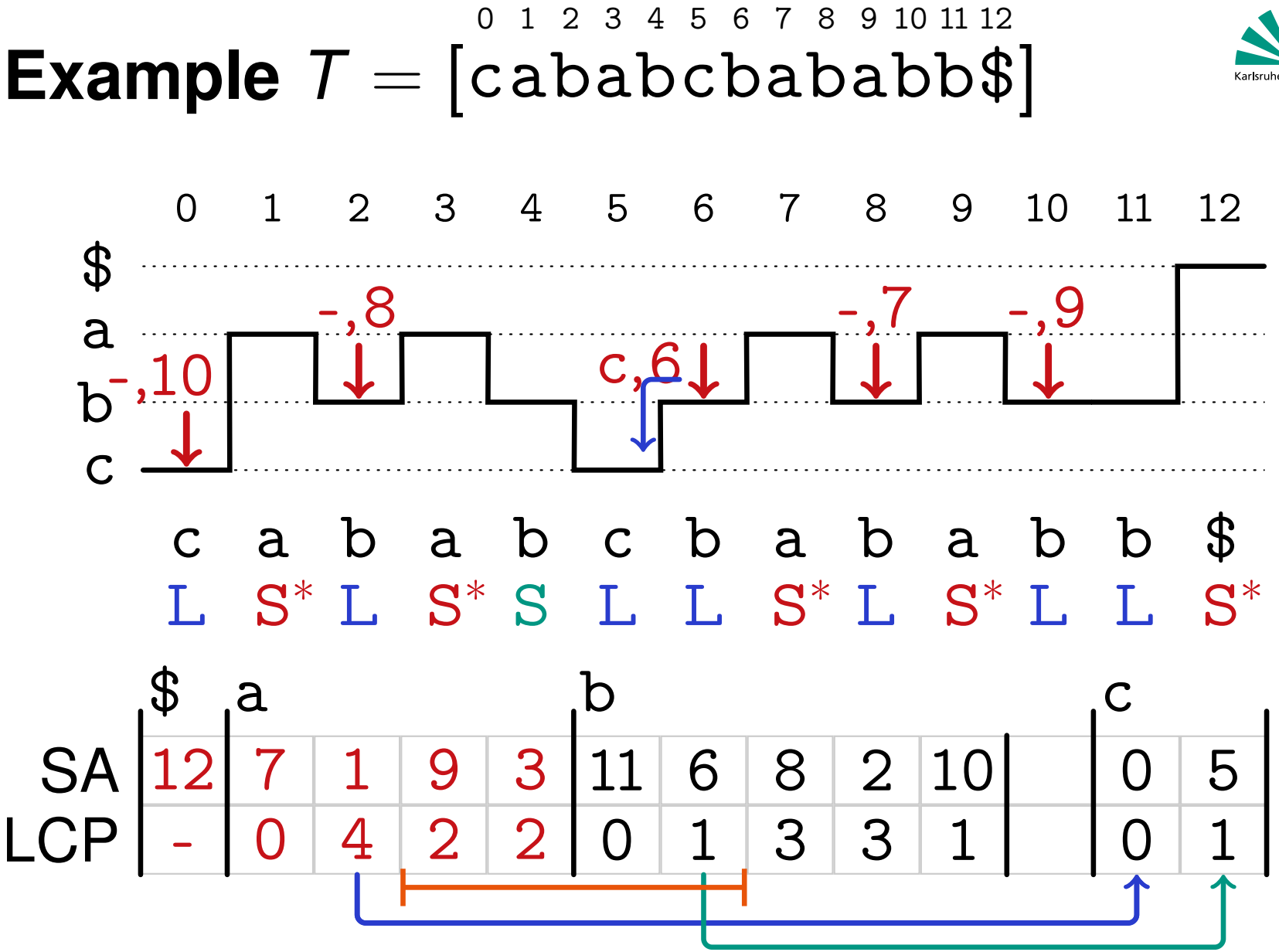
Example $T = [\texttt{cababcbababb\$}]$



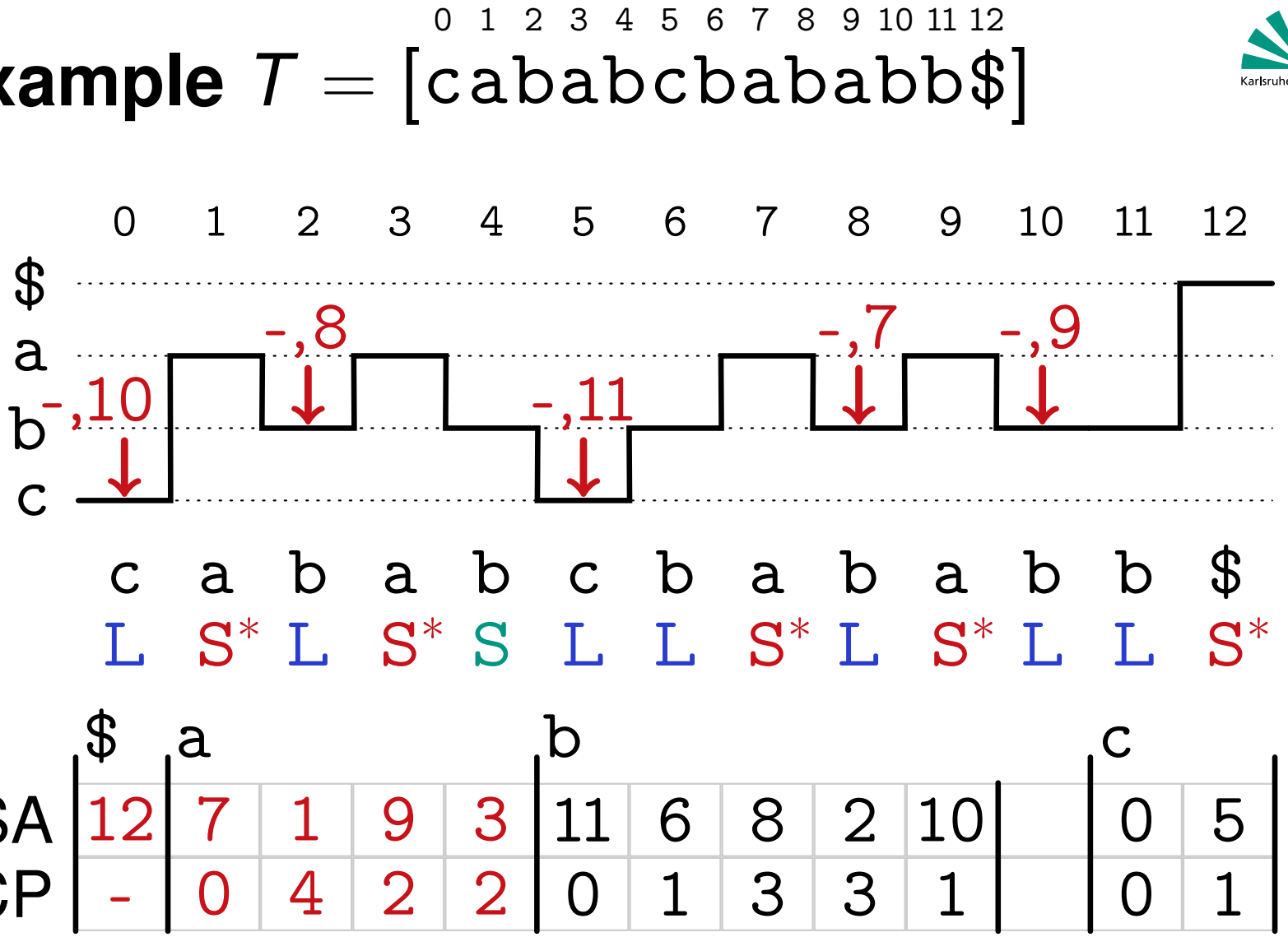
Example $T = [\texttt{cababcbababb\$}]$



Example $T = [\texttt{cababcbababb\$}]$

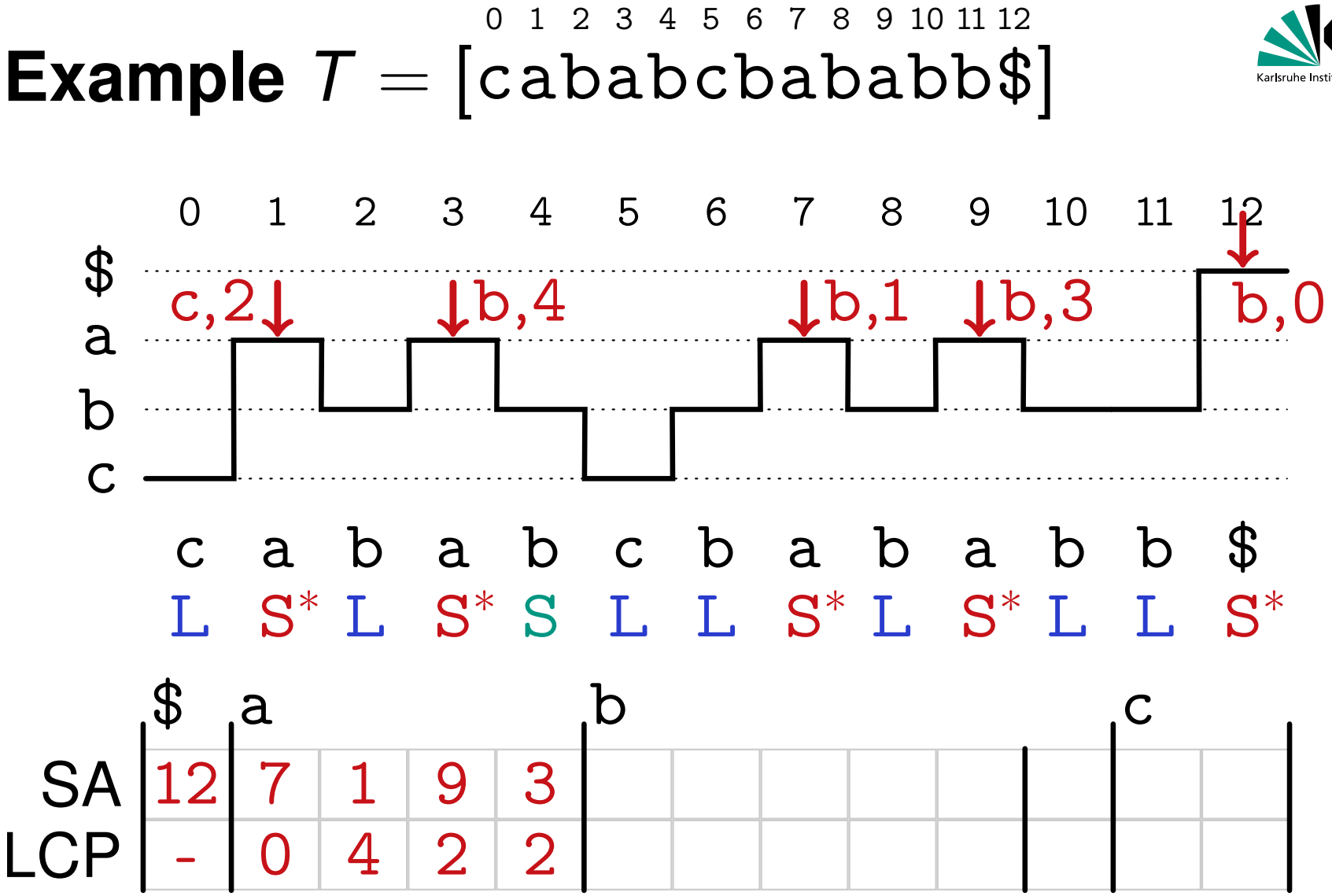


Example $T = [\texttt{cababcbababb\$}]$

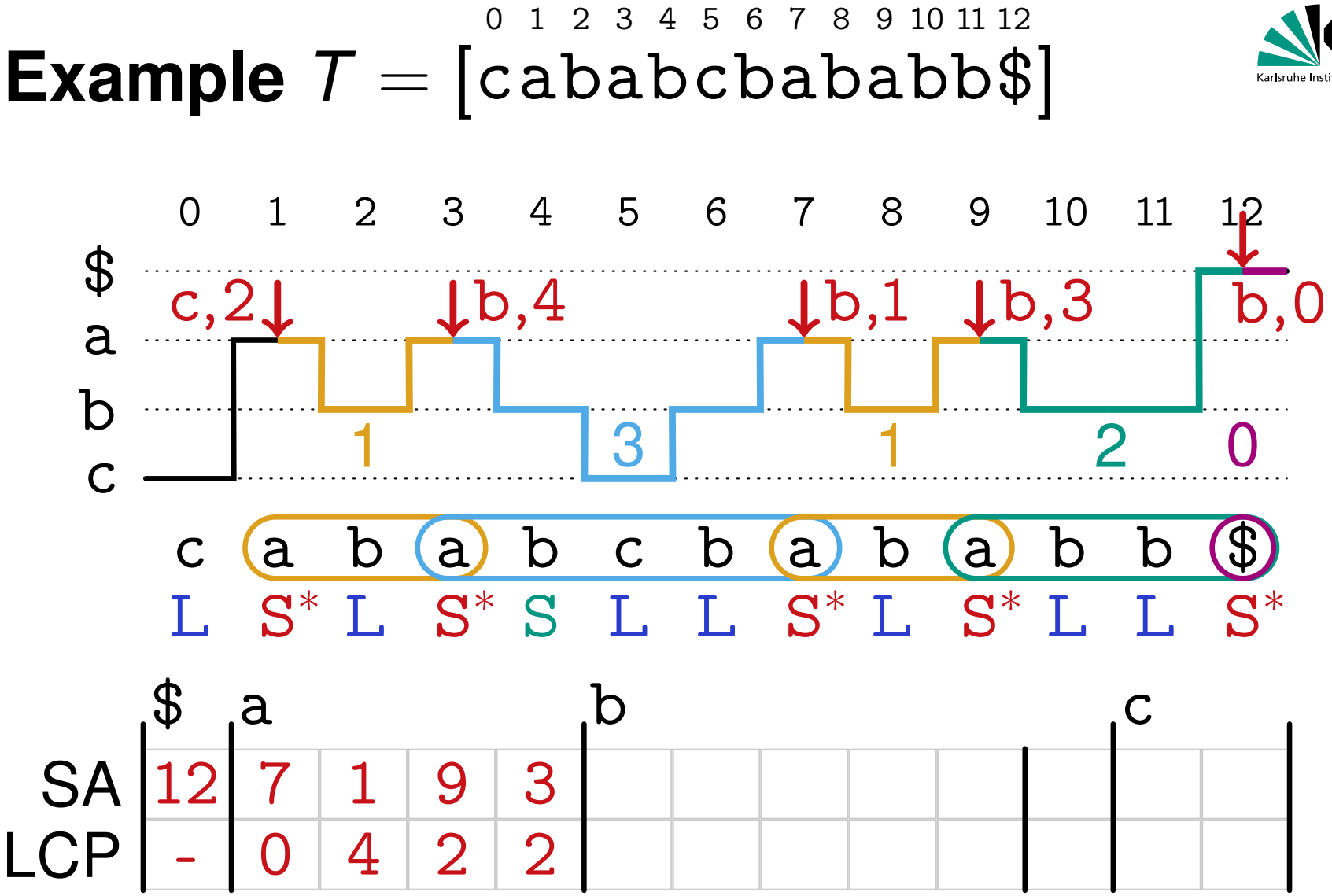


	$	a				b						c	
SA	12	7	1	9	3	11	6	8	2	10		0	5
LCP	-	0	4	2	2	0	1	3	3	1		0	1

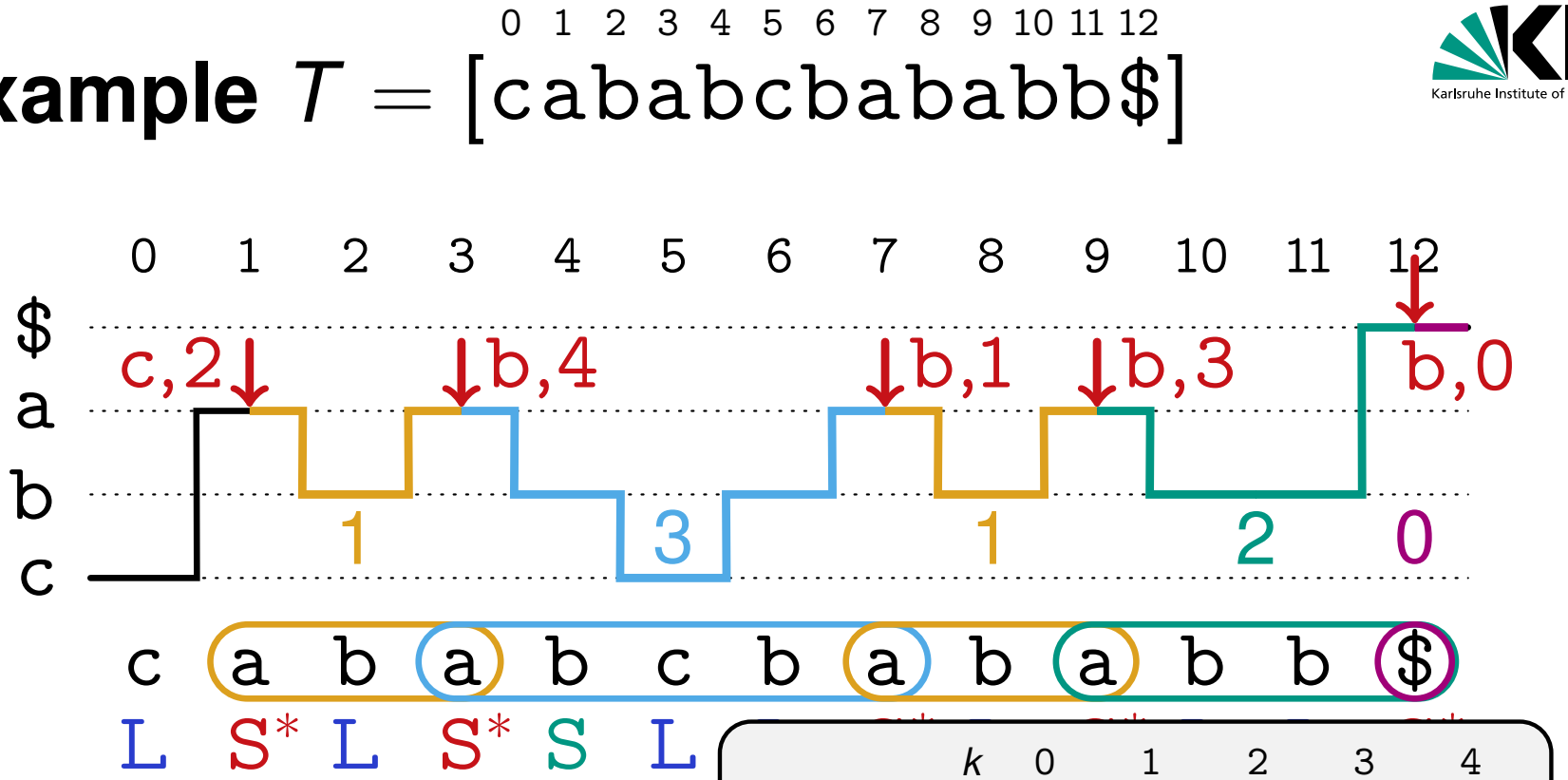
Example $T = [\texttt{cababcbababb\$}]$



Example $T = [\texttt{cababcbababb\$}]$



Example $T = [\texttt{cababcbababb\$}]$



	$	a			b
SA	12	7	1	9	3
LCP	-	0	4	2	2

k	0	1	2	3	4
R	1	3	1	2	0
SA_R	4	2	0	3	1
ISA_R	2	4	1	3	0
LCP_R	-	0	1	0	0
LCP_{S^*}	-	0	4	2	2

eSAIS with LCP Construction

- Recursion size at most $\frac{n}{2}$.
- All SA construction steps: EM scanning, sorting, and PQ operations $\Rightarrow$ amortized sorting complexity.
- LCP construction involves semi-dynamic RMQs.
  Two solutions:
  - EM algorithm in sorting complexity for $N \leq M^2$, or
  - succinct in-memory data structure for practice.

eSAIS with LCP Construction

1. Find relative order of $\mathtt{S}^*$-suffixes.
   1. Split string into $\mathtt{S}^*$-substrings.
   2. Sort $\mathtt{S}^*$-substrings and give lexicographic names.
   3. Recursively calculate suffix and LCP array of lexicographic names, if not unique.
   4. Calculate ranks of $\mathtt{S}^*$-suffixes (red arrows).
   5. Expand recursive LCP_R to $\text{LCP}_{\mathtt{S}^*}$.
2. Induce order of remaining positions using ranks of $\mathtt{S}^*$-suffixes.
   1. Induce all $\mathtt{L}$-suffixes from $\mathtt{S}^*$-suffixes.
   2. Solve RMQs internally or save to disk and process after SA.
   3. Reverse PQ order, induce all $\mathtt{S}$-suffixes from $\mathtt{L}^*$ and solve RMQs (internally or externally) for LCP.

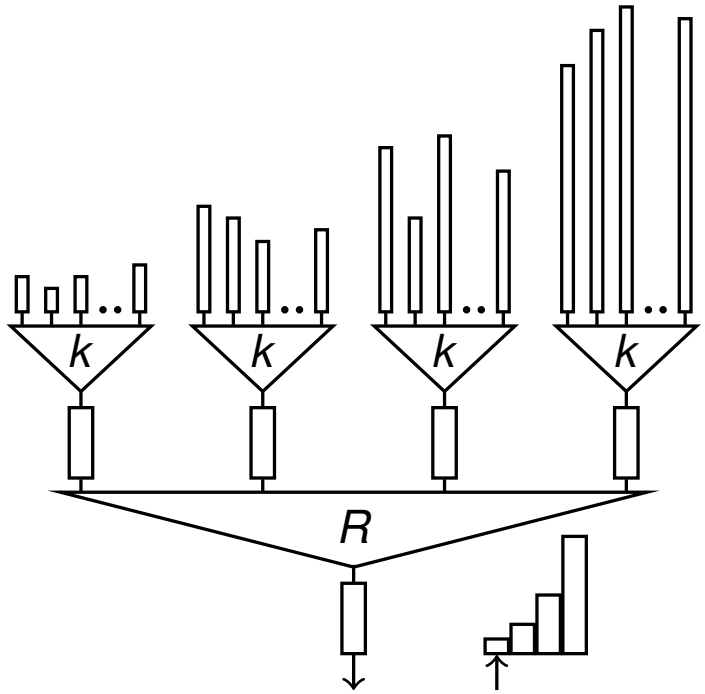
eSAIS with LCP Construction

I/Os

1. Find relative order of $\mathtt{S}^*$-suffixes.
   1. Split string into $\mathtt{S}^*$-substrings. $O(\text{scan}(n))$
   2. Sort $\mathtt{S}^*$-substrings and give lexicographic names. $O(\text{sort}(n))$
   3. Recursively calculate suffix and LCP array of lexicographic names, if not unique. $\leq \text{recurse}(\frac{n}{2})$
   4. Calculate ranks of $\mathtt{S}^*$-suffixes (red arrows). $O(\text{sort}(n))$
   5. Expand recursive LCP_R to $\text{LCP}_{\mathtt{S}^*}$. $O(\text{sort}(n))$
2. Induce order of remaining positions using ranks of $\mathtt{S}^*$-suffixes.
   1. Induce all $\mathtt{L}$-suffixes from $\mathtt{S}^*$-suffixes. $O(\text{sort}(n))$
   2. Solve RMQs internally or save to disk and process after SA.
   3. Reverse PQ order, induce all $\mathtt{S}$-suffixes from $\mathtt{L}^*$ and solve RMQs (internally or externally) for LCP. $O(\text{sort}(n))$

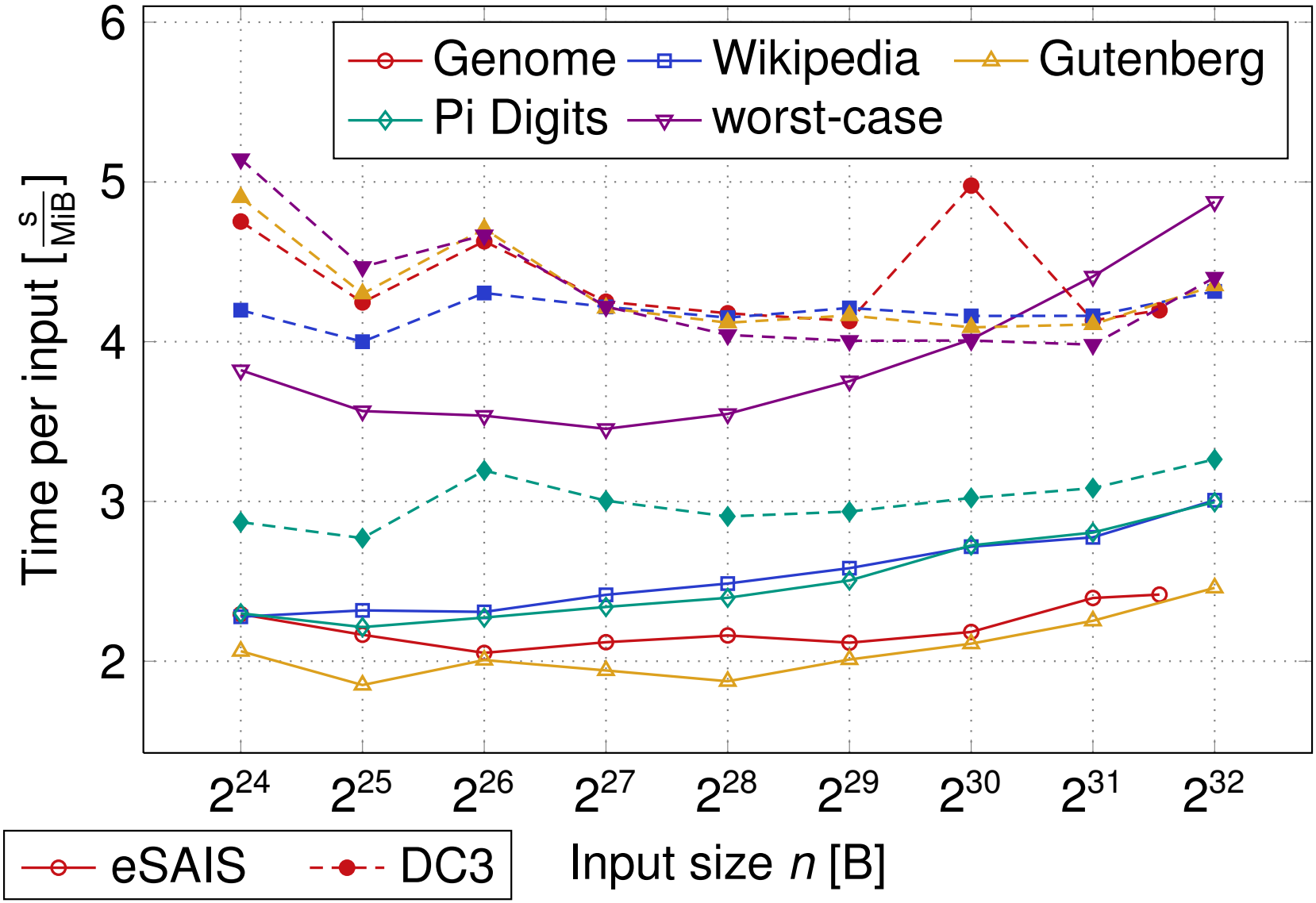
Implementation Highlights

- Implemented in C++ using STXXL.
- STXXL [DKS08] provides efficient EM sorting and a priority queue.
- Enable 40-bit string positions.
- Further EM issues: handle large S^*-substrings via splitting.
- Source code available under GPL: `http://tbingmann.de/2012/esais`

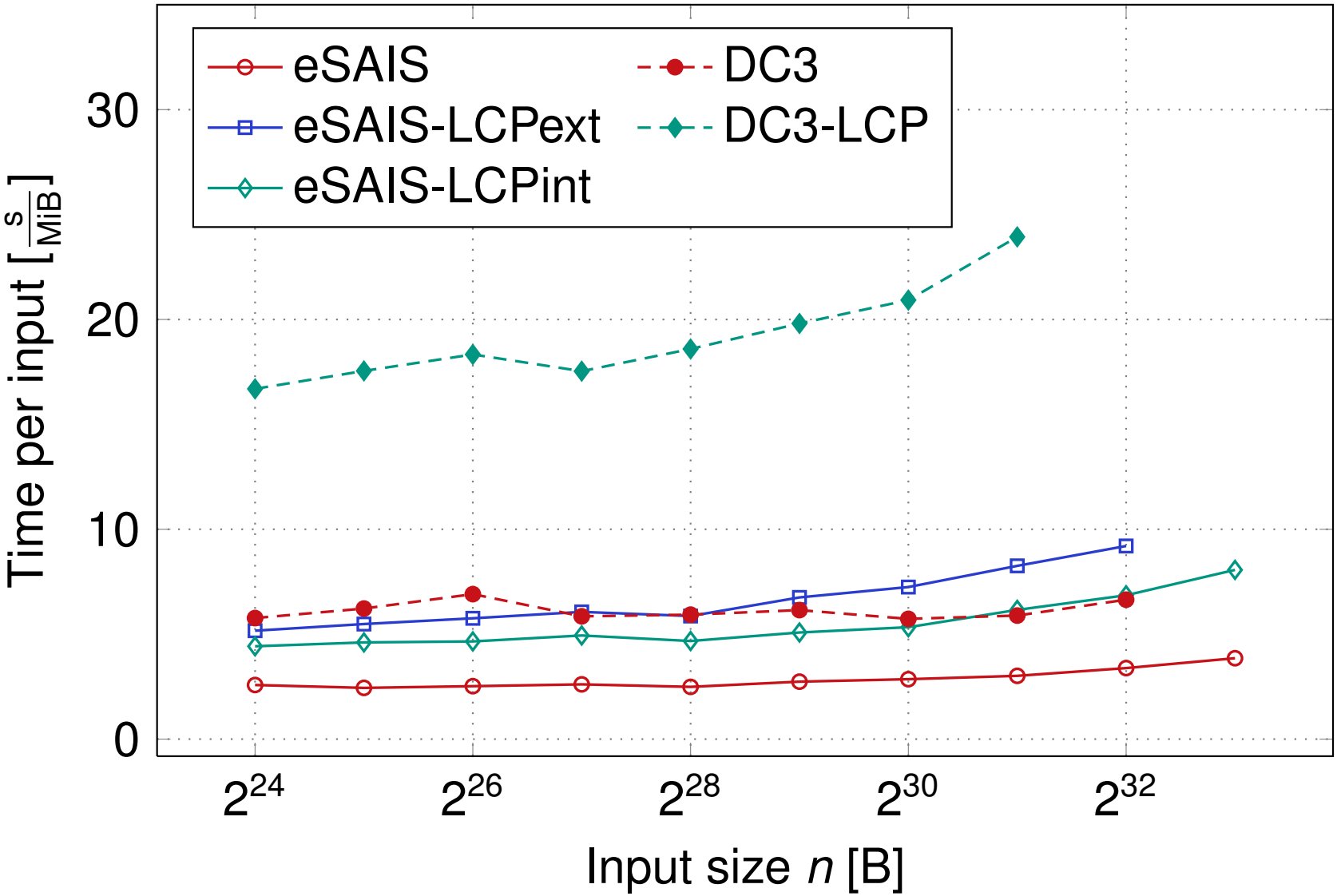


(after [San00])

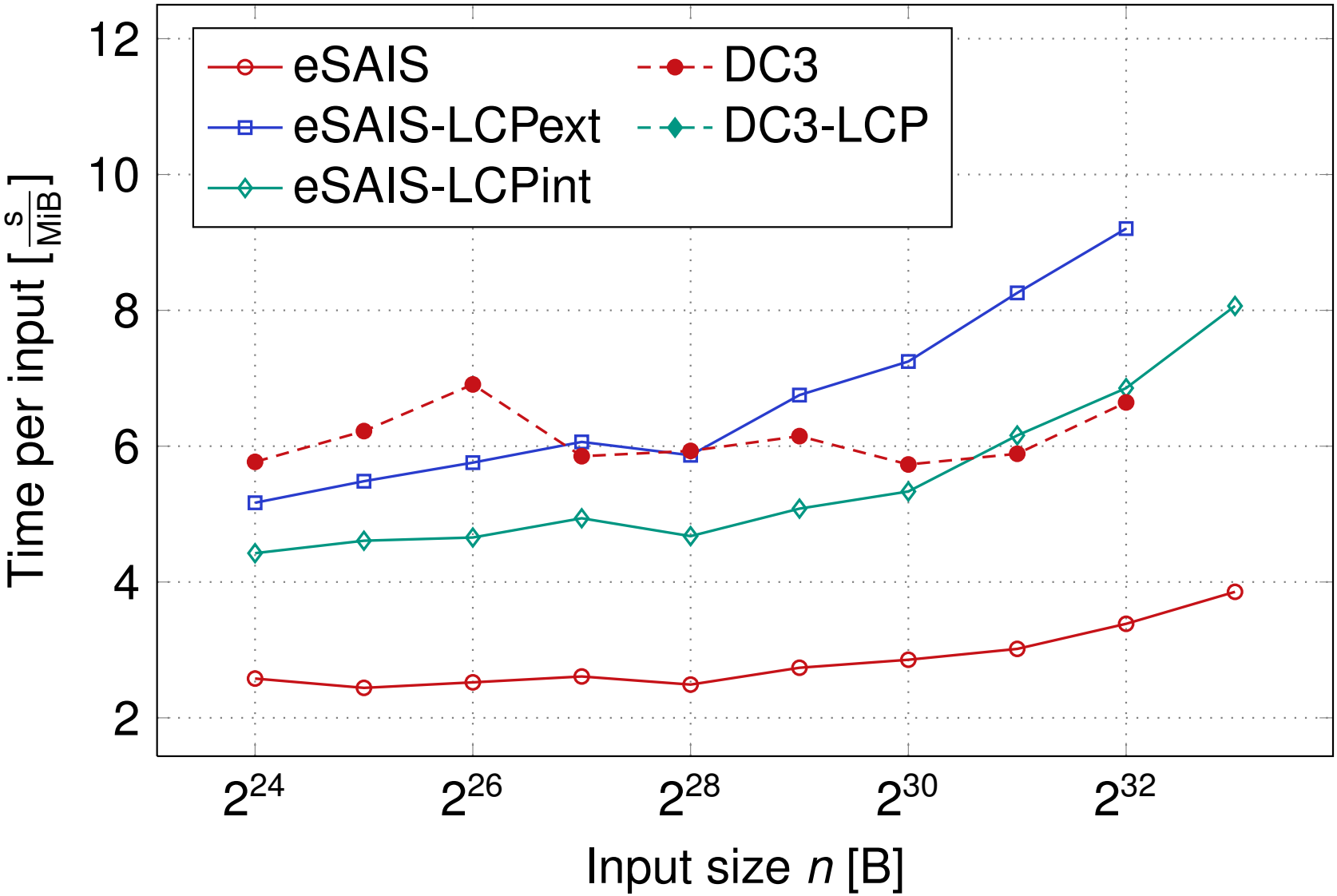
Construction SA only: eSAIS vs. DC3



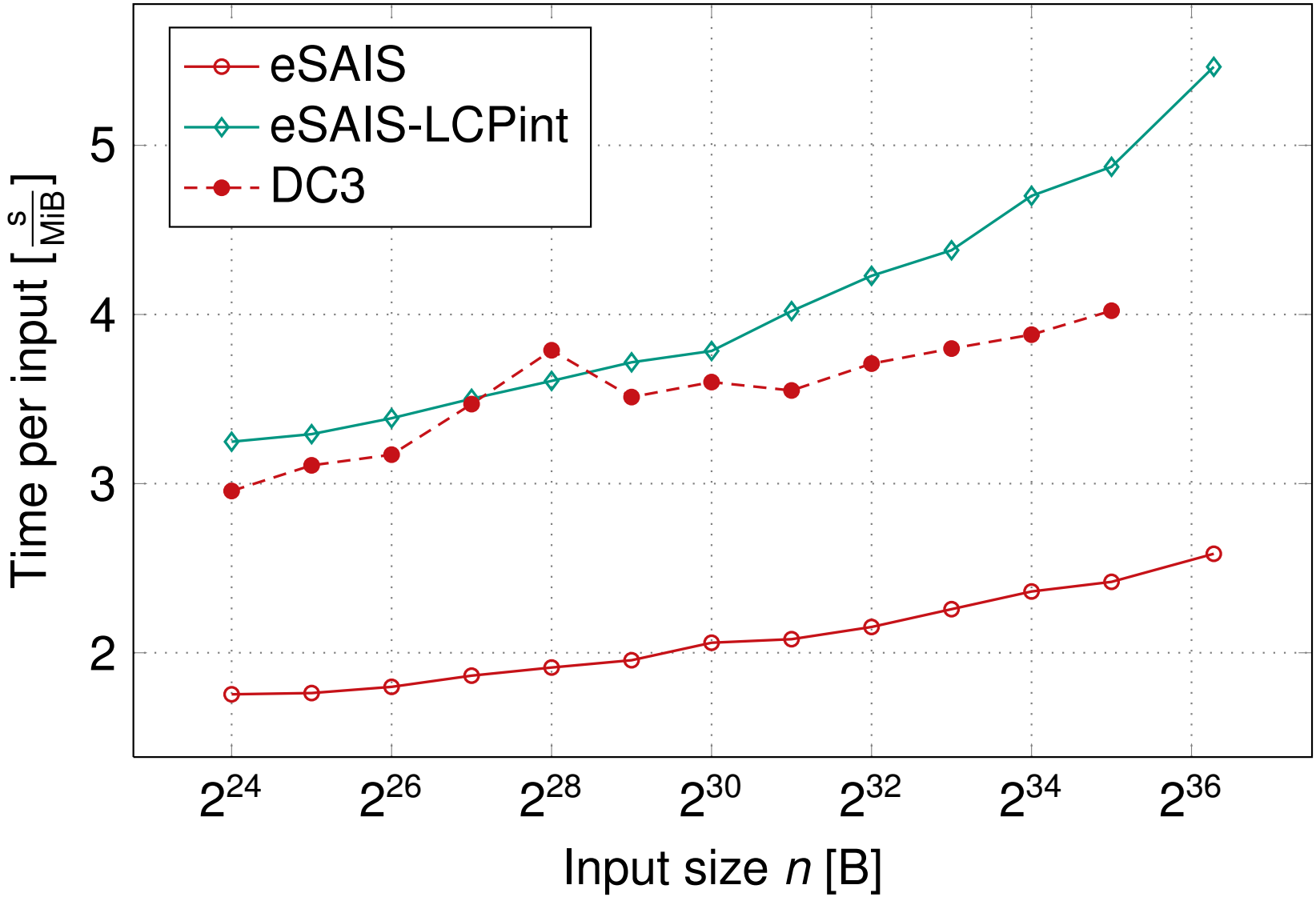
Construction Time: Gutenberg Text



Construction Time: Gutenberg Text



Construction Time: Wikipedia XML



Thank you for your attention!

Questions?

Bibliography I

[AN10] Donald A. Adjeroh and Fei Nan.
Suffix-sorting via Shannon-Fano-Elias codes.
Algorithms, 3(2):145–167, 2010.

[BK03] Stefan Burkhardt and Juha Kärkkäinen.
Fast lightweight suffix array construction and checking.
In *Proc. CPM*, volume 2676 of *LNCS*, pages 55–69. Springer, 2003.

[BW94] Michael Burrows and David J. Wheeler.
A block-sorting lossless data compression algorithm.
Technical Report 124, Digital Equipment Corporation, 1994.

[DKS08] Roman Dementiev, Lutz Kettner, and Peter Sanders.
STXXL: Standard template library for XXL data sets.
Softw. Pract. Exper., 38(6):589–637, 2008.

[Far97] Martin Farach.
Optimal suffix tree construction with large alphabets.
In *Proc. FOCS*, pages 137–143, 1997.

[Fis11] Johannes Fischer.
Inducing the LCP-array.
In *Proc. WADS*, volume 6844 of *LNCS*, pages 374–385. Springer, 2011.

Bibliography II

[GO11] Simon Gog and Enno Ohlebusch.
Fast and lightweight LCP-array construction algorithms.
In *Proc. ALENEX*, pages 25–34. SIAM Press, 2011.

[IT99] Hideo Itoh and Hozumi Tanaka.
An efficient method for in memory construction of suffix arrays.
In *Proc. SPIRE/CRIWG*, pages 81–88. IEEE Press, 1999.

[KA05] Pang Ko and Srinivas Aluru.
Space efficient linear time construction of suffix arrays.
J. Discrete Algorithms, 3(2–4):143–156, 2005.

[KJP04] Dong Kyue Kim, Junha Jo, and Heejin Park.
A fast algorithm for constructing suffix arrays for fixed-size alphabets.
In *Proc. WEA*, volume 3059 of *LNCS*, pages 301–314. Springer, 2004.

[KLA+01] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park.
Linear-time longest-common-prefix computation in suffix arrays and its applications.
In *Proc. CPM*, volume 2089 of *LNCS*, pages 181–192. Springer, 2001.

[KMP09] Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi.
Permuted longest-common-prefix array.
In *Proc. CPM*, volume 5577 of *LNCS*, pages 181–192. Springer, 2009.

Bibliography III

[KSB06] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt.
Linear work suffix array construction.
J. ACM, 53(6):1–19, 2006.

[KSPP05] Dong Kyue Kim, Jeong Seop Sim, Heejin Park, and Kunsoo Park.
Constructing suffix arrays in linear time.
J. Discrete Algorithms, 3(2–4):126–142, 2005.

[LS99] N. Jesper Larsson and Kunihiko Sadakane.
Faster suffix sorting.
Technical Report LU-CS-TR:99-214, LUNDFD6/(NFCS-3140)/1–20/(1999), Department of Computer Science, Lund University, Lund, Sweden, May 1999.

[Man04] Giovanni Manzini.
Two space saving tricks for linear time LCP array computation.
In *Proc. Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 3111 of *LNCS*, pages 372–383. Springer, 2004.

[MF04] Giovanni Manzini and Paolo Ferragina.
Engineering a lightweight suffix array construction algorithm.
Algorithmica, 40(1):33–50, 2004.

Bibliography IV

[MM93] Udi Manber and Eugene W. Myers.
Suffix arrays: A new method for on-line string searches.
SIAM J. Comput., 22(5):935–948, 1993.

[Mor06] Yuta Mori.
divsufsort, 2006.
http://code.google.com/p/libdivsufsort/.

[MP06] Michael A. Maniscalco and Simon J. Puglisi.
Faster lightweight suffix array construction.
In *Proc. IWOCA*, pages 16–29, 2006.

[MP08] Michael A. Maniscalco and Simon J. Puglisi.
An efficient, versatile approach to suffix sorting.
ACM J. Exp. Algorithmics, 12:Article no. 1.2, 2008.

[Na05] Joong Chae Na.
Linear-time construction of compressed suffix arrays using $O(n \log n)$-bit working space for large alphabets.
In *Proc. CPM*, volume 3537 of *LNCS*, pages 57–67. Springer, 2005.

[Non11] Ge Nong.
An optimal suffix array construction algorithm.
Technical report, Department of Computer Science, Sun Yat-sen University, 2011.

Bibliography V

[NZ07] Ge Nong and Sen Zhang.
Optimal lightweight construction of suffix arrays for constant alphabets.
In *Proc. WADS*, volume 4619 of *LNCS*, pages 613–624. Springer, 2007.

[NZC11] Ge Nong, Sen Zhang, and Wai Hong Chan.
Two efficient algorithms for linear time suffix array construction.
IEEE Trans. Computers, 60(10):1471–1484, 2011.

[PST07] Simon J. Puglisi, William F. Smyth, and Andrew Turpin.
A taxonomy of suffix array construction algorithms.
ACM Comput. Surv., 39(2), 2007.

[PT08] Simon J. Puglisi and Andrew Turpin.
Space-time tradeoffs for longest-common-prefix array computation.
In *Proc. ISAAC*, volume 5369 of *LNCS*, pages 124–135. Springer, 2008.

[San00] Peter Sanders.
Fast priority queues for cached memorys.
ACM J. Exp. Algorithmics, 5:Article No. 7, 2000.

[Sew00] Julian Seward.
On the performance of BWT sorting algorithms.
In *Data Compression Conference*, pages 173–182. IEEE Press, 2000.

Bibliography VI

[SS07] Klaus-Bernd Schürmann and Jens Stoye.
An incomplex algorithm for fast suffix array construction.
Softw. Pract. Exper., 37(3):309–329, 2007.