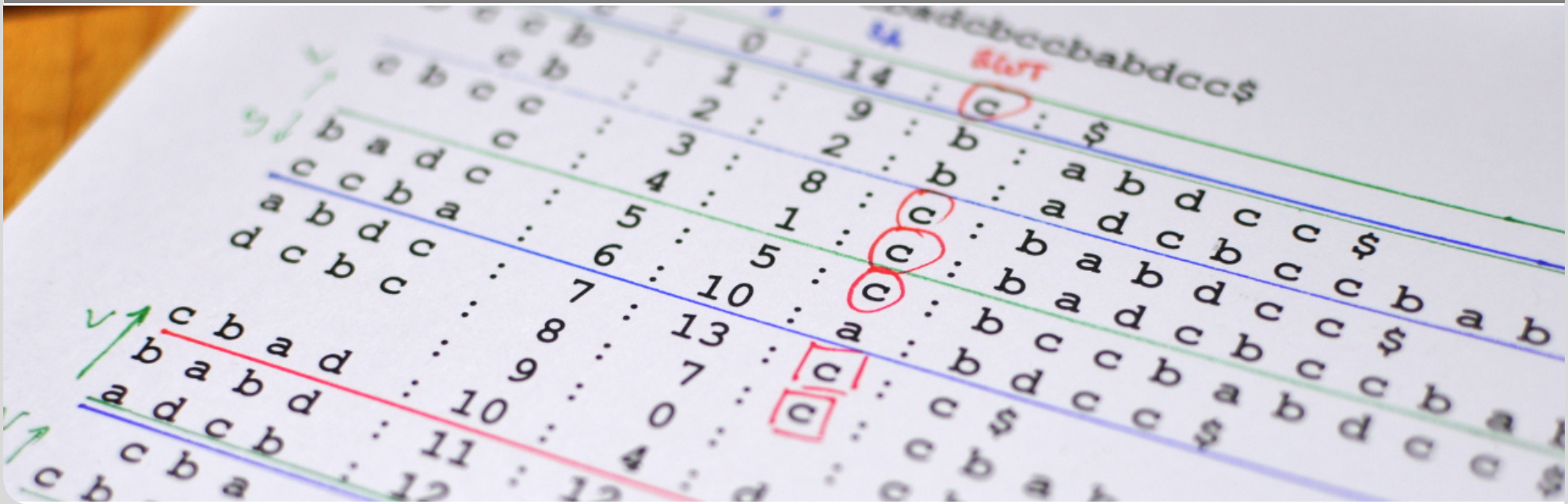
Karlsruhe Institute of Technology

Suffix Sorting Algorithms

Timo Bingmann | Text-Indexierung Vorlesung 2016-12-01

INSTITUTE OF THEORETICAL INFORMATICS – ALGORITHMICS



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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
i	T_i														
0	d	b	a	d	c	b	c	c	b	a	b	d	c	c	$
1	b	a	d	c	b	c	c	b	a	b	d	c	c	$	
2	a	d	c	b	c	c	b	a	b	d	c	c	$		
3	d	c	b	c	c	b	a	b	d	c	c	$			
4	c	b	c	c	b	a	b	d	c	c	$				
5	b	c	c	b	a	b	d	c	c	$					
6	c	c	b	a	b	d	c	c	$						
7	c	b	a	b	d	c	c	$							
8	b	a	b	d	c	c	$								
9	a	b	d	c	c	$									
10	b	d	c	c	$										
11	d	c	c	$											
12	c	c	$												
13	c	$													
14	$														

Example $T =$ [dbadcbccbabdcc$]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	d	b	a	d	c	b	c	c	b	a	b	d	c	c	$

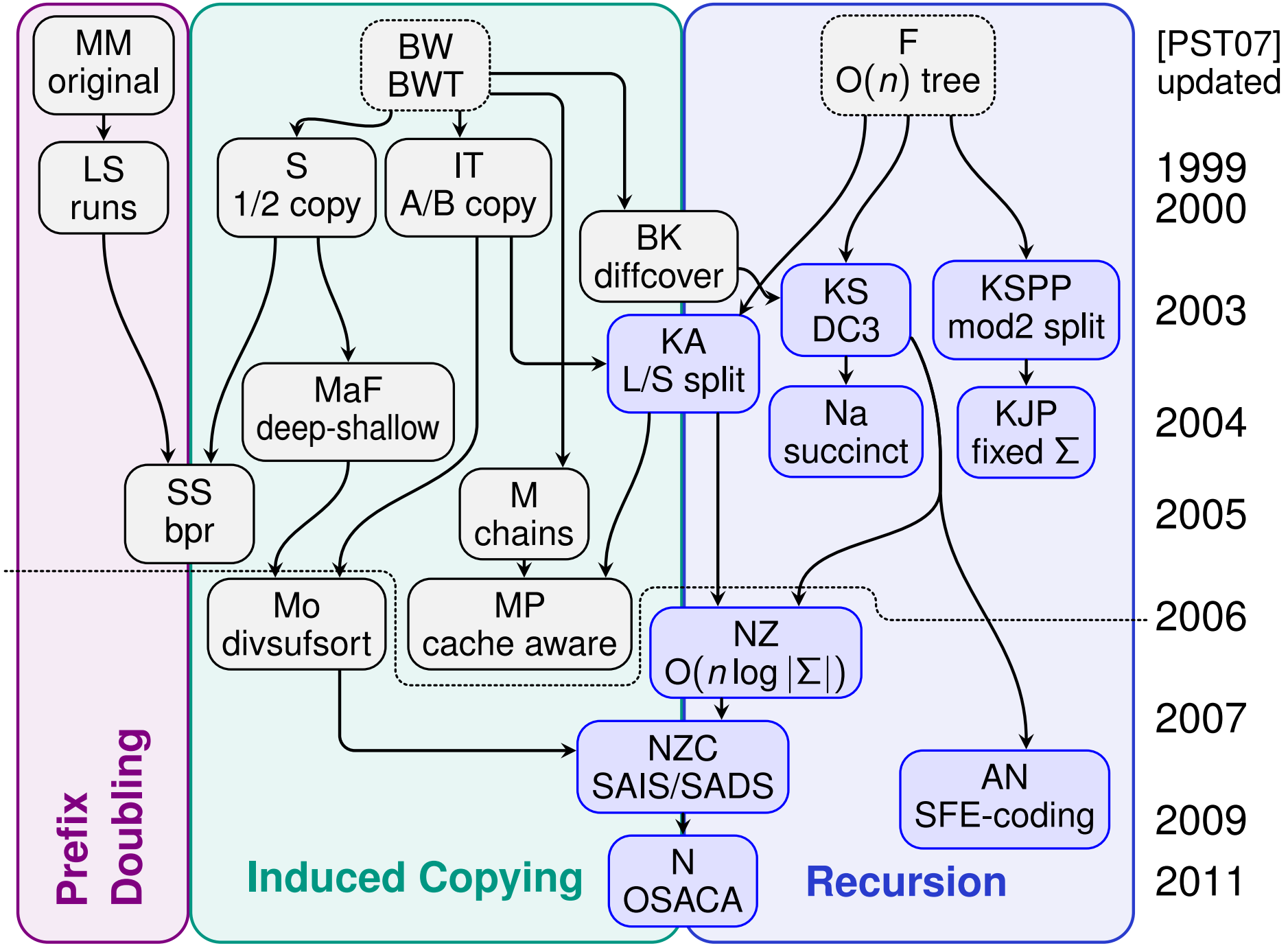
SA_i	$T_{SA_i \ldots n}$
14	$
9	a b d c c $
2	a d c b c c b a b d c c $
8	b a b d c c $
1	b a d c b c c b a b d c c $
5	b c c b a b d c c $
10	b d c c $
13	c $
7	c b a b d c c $
4	c b c c b a b d c c $
12	c c $
6	c c b a b d c c $
0	d b a d c b c c b a b d c c $
3	d c b c c b a b d c c $
11	d c c $

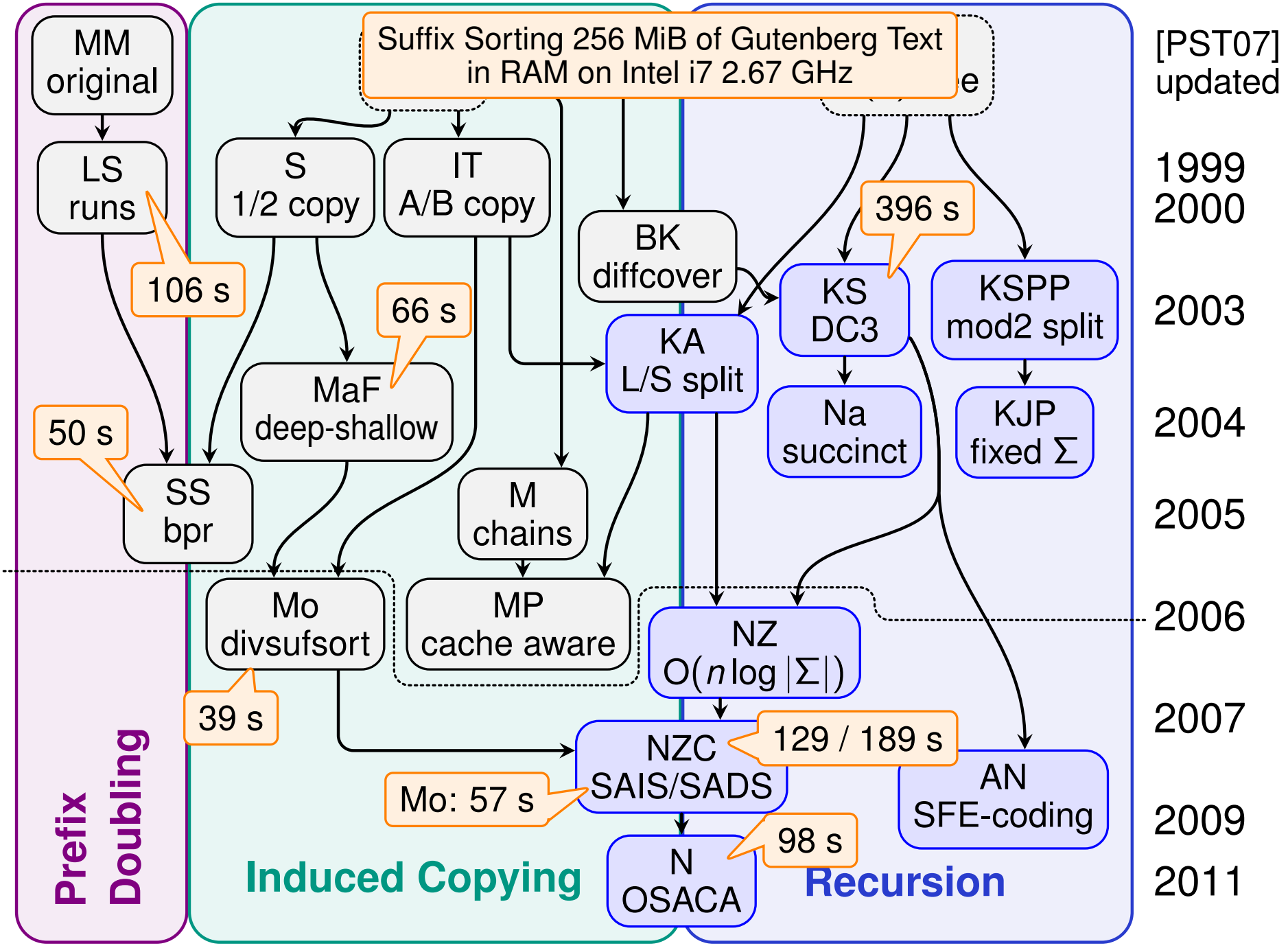
Example $T = [\texttt{dbadcbccbabdcc\$}]$

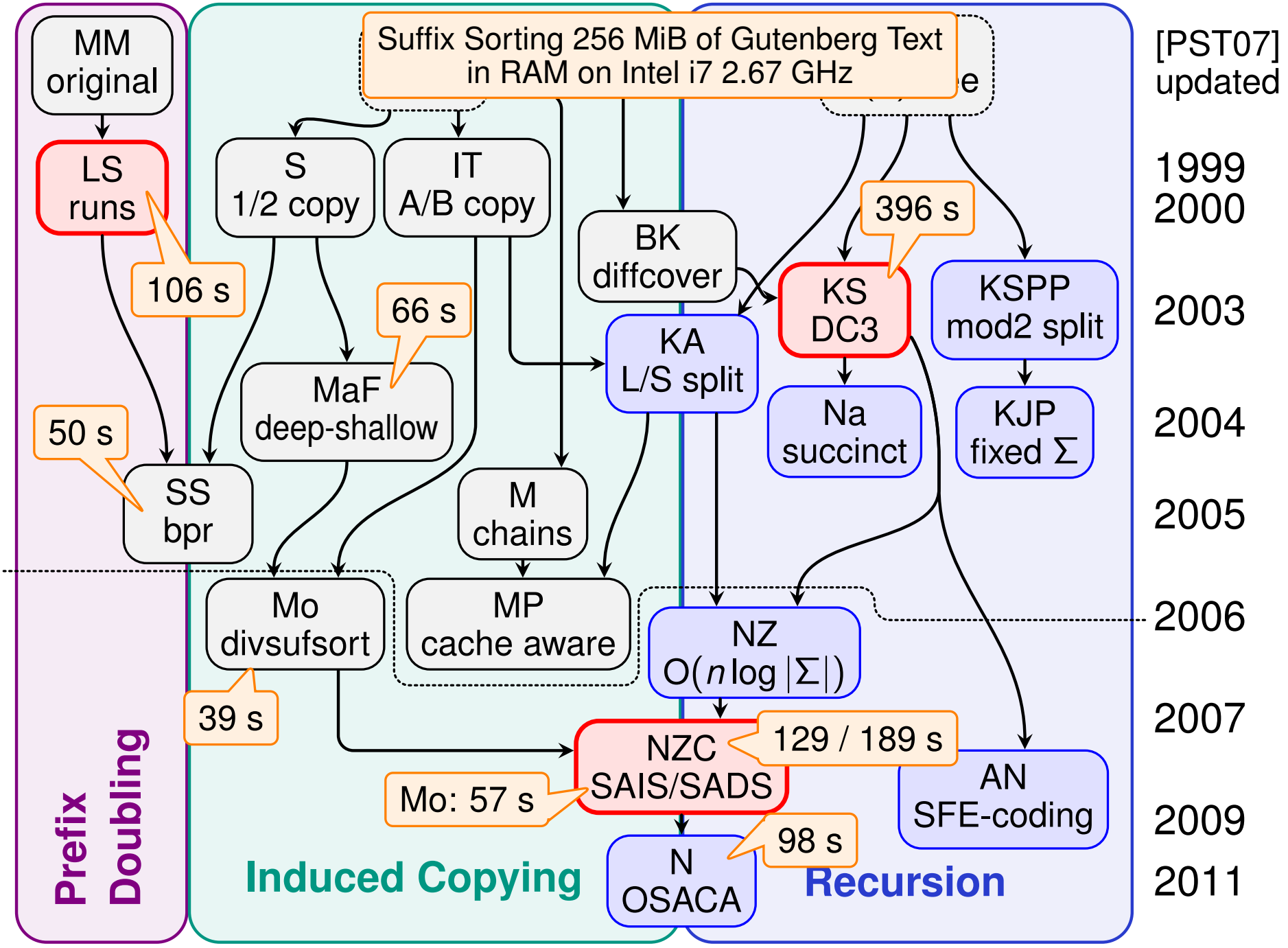
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SA_i	LCP_i	$T_{SA_i \ldots n}$														
14		$														
9		a	b	d	c	c	$									
2		a	d	c	b	c	c	b	a	b	d	c	c	$		
8		b	a	b	d	c	c	$								
1	2	b	a	d	c	b	c	c	b	a	b	d	c	c	$	
5		b	c	c	b	a	b	d	c	c	$					
10		b	d	c	c	$										
13		c	$													
7		c	b	a	b	d	c	c	$							
4		c	b	c	c	b	a	b	d	c	c	$				
12		c	c	$												
6		c	c	b	a	b	d	c	c	$						
0		d	b	a	d	c	b	c	c	b	a	b	d	c	c	$
3		d	c	b	c	c	b	a	b	d	c	c	$			
11		d	c	c	$											

Example $T =$ [dbadcbccbabdcc$]

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SA_i	LCP_i	$T_{SA_i \ldots n}$														
14	-	$														
9	0	a	b	d	c	c	$									
2	1	a	d	c	b	c	c	b	a	b	d	c	c	$		
8	0	b	a	b	d	c	c	$								
1	2	b	a	d	c	b	c	c	b	a	b	d	c	c	$	
5	1	b	c	c	b	a	b	d	c	c	$					
10	1	b	d	c	c	$										
13	0	c	$													
7	1	c	b	a	b	d	c	c	$							
4	2	c	b	c	c	b	a	b	d	c	c	$				
12	1	c	c	$												
6	2	c	c	b	a	b	d	c	c	$						
0	0	d	b	a	d	c	b	c	c	b	a	b	d	c	c	$
3	1	d	c	b	c	c	b	a	b	d	c	c	$			
11	2	d	c	c	$											


MM
original
LS
runs
SS
bpr
BW
BWT
S
1/2 copy
IT
A/B copy
MaF
deep-shallow
M
chains
Mo
divsufsort
MP
cache aware
BK
diffcover
KA
L/S split
F
O(n) tree
KS
DC3
KSPP
mod2 split
Na
succinct
KJP
fixed Σ
NZ
O(n log |Σ|)
NZC
SAIS/SADS
N
OSACA
AN
SFE-coding
Prefix
Doubling
Induced Copying
Recursion
[PST07]
updated
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
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
50 s
SS bpr
Na succinct
KJP fixed Σ
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	$O(n)$	$11n$ S-EM
F	2011	T $\rightarrow$ SA,LCP	$O(n)$	$9n$

Sorting $T = [\texttt{dbadcbccbabdcc\$}]$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SA_i	$T_{SA_i \ldots n}$														
0	d	b	a	d	c	b	c	c	b	a	b	d	c	c	$
1	b	a	d	c	b	c	c	b	a	b	d	c	c	$	
2	a	d	c	b	c	c	b	a	b	d	c	c	$		
3	d	c	b	c	c	b	a	b	d	c	c	$			
4	c	b	c	c	b	a	b	d	c	c	$				
5	b	c	c	b	a	b	d	c	c	$					
6	c	c	b	a	b	d	c	c	$						
7	c	b	a	b	d	c	c	$							
8	b	a	b	d	c	c	$								
9	a	b	d	c	c	$									
10	b	d	c	c	$										
11	d	c	c	$											
12	c	c	$												
13	c	$													
14	$														

Sorting $T = [\texttt{dbadcbccbabdcc\$}]$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
14	$														
2	a	d	c	b	c	c	b	a	b	d	c	c	$		
9	a	b	d	c	c	$									
1	b	a	d	c	b	c	c	b	a	b	d	c	c	$	
5	b	c	c	b	a	b	d	c	c	$					
8	b	a	b	d	c	c	$								
10	b	d	c	c	$										
4	c	b	c	c	b	a	b	d	c	c	$				
6	c	c	b	a	b	d	c	c	$						
7	c	b	a	b	d	c	c	$							
12	c	c	$												
13	c	$													
0	d	b	a	d	c	b	c	c	b	a	b	d	c	c	$
3	d	c	b	c	c	b	a	b	d	c	c	$			
11	d	c	c	$											

Sorting $T = [\texttt{dbadcbccbabdcc\$}]$

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SA_i	$T_{SA_i \ldots n}$															
14		$														
9		a	b	d	c	c	$									
2		a	d	c	b	c	c	b	a	b	d	c	c	$		
1		b	a	d	c	b	c	c	b	a	b	d	c	c	$	
8		b	a	b	d	c	c	$								
10		b	d	c	c	$										
5		b	c	c	b	a	b	d	c	c	$					
13		c	$													
4		c	b	c	c	b	a	b	d	c	c	$				
7		c	b	a	b	d	c	c	$							
6		c	c	b	a	b	d	c	c	$						
12		c	c	$												
0		d	b	a	d	c	b	c	c	b	a	b	d	c	c	$
3		d	c	b	c	c	b	a	b	d	c	c	$			
11		d	c	c	$											

Sorting $T = [\texttt{dbadcbccbabdcc\$}]$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	d	b	a	d	c	b	c	c	b	a	b	d	c	c	$

SA_i	$T_{SA_i \ldots n}$
14	$
9	a b d c c $
2	a d c b c c b a b d c c $
8	b a b d c c $
1	b a d c b c c b a b d c c $
5	b c c b a b d c c $
10	b d c c $
13	c $
7	c b a b d c c $
4	c b c c b a b d c c $
12	c c $
6	c c b a b d c c $
0	d b a d c b c c b a b d c c $
3	d c b c c b a b d c c $
11	d c c $

LS – Larsson, Sadakane [LS99]

Idea (from [MM93]): Sort suffixes by up to $2h$ characters depth by using information about suffixes sorted up to h characters depth.

Definition: Suffixes are in h-order if the first h characters are sorted.

Original [MM93]: Sorting the suffixes using, for each suffix s_i, the position in the h-order of s_i as first key, and the position of $s_i + h$ in the same order as the second key, yields the $2h$-order.

Definitions [LS99]: If SA_h is an h-order, then

1. A maximal sequence of adjacent suffixes in SA_h which have the same initial h characters is called an h-group.
2. A h-group of size one is called sorted (or singleton), otherwise it is unsorted. A maximal sequence of sorted groups is a combined sorted group.

LS – Larsson, Sadakane [LS99] I

Given a text T of length $n := |T|$.

1. Place suffixes, enumerated by $0, \ldots, n$ into an array SA_0 as $SA_0[i] = i$.
2. Sort array SA_0 by $T[i]$ for index i to get SA_1. Set $h = 1$.
3. For each $i = 0, \ldots, n$ set $ISA_1[i]$ to the (current) 1-group rank of suffix i, which is the last index in $SA_1[i]$ which holds a suffix with the same initial character as suffix i.
4. For each unsorted group in SA_1 or combined sorted group occupying the subarray $SA_1[a..b]$, set $L[a]$ to its length or negated length, respectively.

LS – Larsson, Sadakane [LS99] II

5. Sort each unsorted group in SA_h (with multikey-quicksort), using $ISA_h[SA_h[i] + h]$ as the sort key for index i.
6. Mark splitting positions between unequal keys in the unsorted groups.
7. Set $h := 2h$, create new groups by splitting at the marked positions.
8. Recalculate ISA_h for all processed groups, and update L accordingly.
9. if SA_h still contains any unsorted group, goto 5.

Worst-case runtime: $O(n \log n)$. Space: $8n$.

What is a worst-case input?

LS – Example

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$T[i]$	t	o	b	e	o	r	n	o	t	t	o	b	e	$
$SA_0[i]$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$h = 1,\ SA_1[i]$	13	2	11	3	12	6	1	4	7	10	5	0	8	9
$ISA_1[i]$	11	6	1	3	6	10	5	6	11	11	6	1	3	0
$ISA_1[SA_1[i]+1]$		3	3	6	0		1	10	11	1		6	11	6
$h = 2,\ SA_2[i]$	13	2	11	12	3	6	1	10	4	7	5	0	9	8
$ISA_2[i]$	11	6	1	4	8	10	5	9	13	11	6	1	3	0
$ISA_2[SA_2[i]+2]$		8	0				4	3				1	1	
$h = 4,\ SA_4[i]$	13	11	2	12	3	6	10	1	4	7	5	0	9	8
$ISA_4[i]$	11	7	2	4	8	10	5	9	13	11	6	1	3	0
$ISA_2[SA_2[i]+4]$												8	0	
$h = 8,\ SA_8[i]$	13	11	2	12	3	6	10	1	4	7	5	9	0	8

DC3 – Kärkkäinen, Sanders [KS03]

Idea: Divide-and-conquer – Recursively calculate ranks for only a difference cover D of every X suffixes, induce remaining $X - D$ suffixes by merging ranks and characters.

- One of the first linear time suffix sorting algorithms.
- Also optimal in other models like PRAM and external memory.

Definition: A difference cover D_n is a subset of $X = 0, \ldots, n-1$ such that $X = \{i - j \bmod n \mid i, j \in D_n\}$.
Examples: $D_3 = \{1, 2\}$, $D_7 = \{0, 1, 3\}$, $D_{13} = \{0, 1, 3, 9\}$.
For any $n \in \mathbb{N}$, D_n exists with $|D_n| = O(\sqrt{n})$.

DC3 – Kärkkäinen, Sanders [KS03]

Karlsruhe Institute of Technology

1. Select triples $(T[i..i+2])$ for $i \bmod 3 \neq 0$ and sort them.
2. Find lexicographic names T' of triples, i.e., such that $T'[i] < T'[j]$ iff $T[i..i+2] < T[j..j+2]$.
3. Construct recursive text $T_{12} = [T'[i] \mid i \bmod = 1] \oplus [T'[i] \mid i \bmod = 2]$, such that first and second half represent the whole text as triple names.
4. Unless all lexicographic names are unique, recurse on T_{12} (with $|T_{12}| \leq \frac{2|T|}{3}$ and alphabet size $\leq \frac{2|T|}{3}$).
5. Take resulting SA_{12} and invert to ISA_{12}, if names are unique compute ISA_{12} directly.
6. With $T[i]$ and ISA_{12} construct three sequences, such that each suffix i is represented with an entry in $S_{i \bmod 3}$:
   $S_0 = [(t_i, t_{i+1}, r_{i+1}, r_{i+2}, i) \mid i \bmod 3 = 0]$,
   $S_1 = [(r_i, t_i, r_{i+1}, i) \mid i \bmod 3 = 1]$, and
   $S_2 = [(r_i, t_i, t_{i+1}, r_{i+2}, i) \mid i \bmod 3 = 2]$.
7. Merge together S_0, S_1, and S_2 to deliver SA.

Suffix Sorting with DC3: Example

	0	1	2	3	4	5	6	7	8	9	10
$T =$	d	b	a	c	b	a	c	b	d	$	$

$= [\, t_i \,]_{i=0,\ldots,n-1}$

triples	(bac,1),	(bac,4),	(bd$,7),	(acb,2)	(acb,5),	(d$$,8)
sorted	(acb,2)	(acb,5),	(bac,1),	(bac,4),	(bd$,7),	(d$$,8)
equal 0/1	0	0	1	0	1	1
prefix sum	0	0	1	1	2	3

$R =$ 1 1 2 0 0 3 $

r_1 r_4 r_7 r_2 r_5 r_8

$\mathrm{SA}_R =$ 3 4 0 1 2 5 $ $\quad \mathrm{ISA}_R =$ 2 3 4 0 1 5 $

$S_0 = [(\mathtt{d},\mathtt{b},2,0,0),(\mathtt{c},\mathtt{b},3,1,3),(\mathtt{c},\mathtt{b},4,5,6)]$ $\quad (t_i, t_{i+1}, r_{i+1}, r_{i+2}, i)$

$S_1 = [(2,\mathtt{b},0,1),(3,\mathtt{b},1,4),(4,\mathtt{b},5,7)]$ $\quad (r_i, t_i, r_{i+1}, i)$

$S_2 = [(0,\mathtt{a},\mathtt{c},3,2),(1,\mathtt{a},\mathtt{c},4,5),(5,\mathtt{d},\$,6,8)]$ $\quad (r_i, t_i, t_{i+1}, r_{i+2}, i)$

$\mathrm{SA}_T = \mathrm{Merge}(\mathrm{Sort}(S_0), \mathrm{Sort}(S_1), \mathrm{Sort}(S_2))$

Merging in DC3

Last step: $SA_T = \text{Merge}(\text{Sort}(S_0), \text{Sort}(S_1), \text{Sort}(S_2))$, where

$$(t_i, t_{i+1}, r_{i+1}, r_{i+2}, i) \in S_0 < (r_j, t_j, r_{j+1}, j) \in S_1$$
$$:\Leftrightarrow (t_i, r_{i+1}) < (t_j, r_{j+1})$$

$$(t_i, t_{i+1}, r_{i+1}, r_{i+2}, i) \in S_0 < (r_j, t_j, t_{j+1}, r_{j+2}, j) \in S_2$$
$$:\Leftrightarrow (t_i, t_{i+1}, r_{i+2}) < (t_j, t_{j+1}, r_{j+2})$$

$$(r_i, t_i, r_{i+1}, i) \in S_1 < (r_j, t_j, t_{j+1}, r_{j+2}, j) \in S_2$$
$$:\Leftrightarrow (r_i) < (r_j)$$

Why do we merge exactly like this?

DC3 – Time and Space Complexity

Worst-case complexity of DC3/skew3: $O(n)$.

It follows the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n < 3\,, \\ T(\frac{2}{3}n) + \Theta(n) & \text{if } n \geq 3\,, \end{cases}$$

which can be solved using an (extended) master theorem.

Correctness follows from the way the merging relation compares suffixes by comparing characters and the recursively calculated ranks.

What is a worst-case input?

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

		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	$					

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

		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	$					

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

		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 $

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

		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}													
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	$					

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

		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}													
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	$					

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

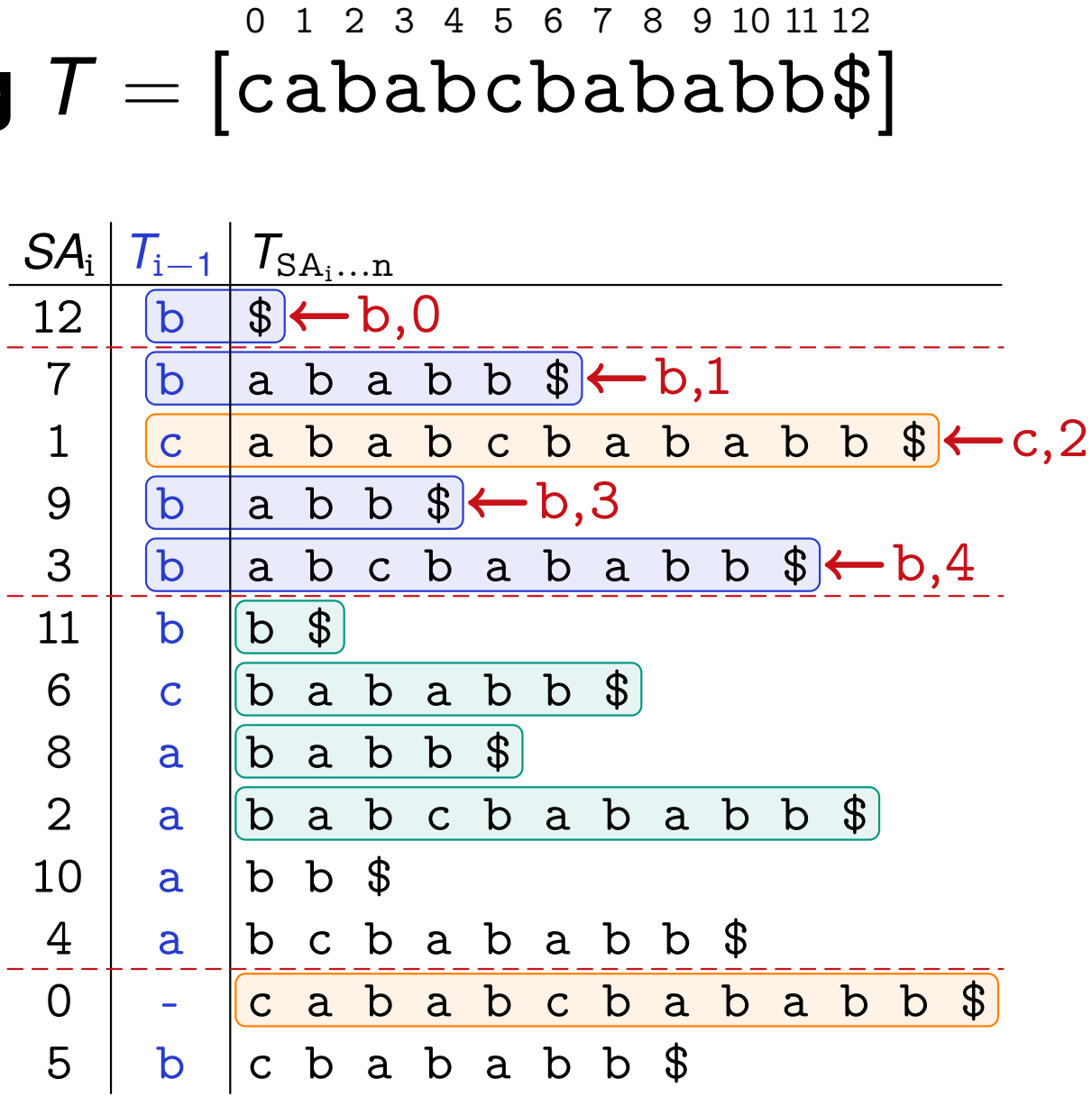
		0	1	2	3	4	5	6	7	8	9	10	11	12
		c	a	b	a	b	c	b	a	b	a	b	b	$
SA_i	T_{i-1}													
12	b	$												
7	b	a	b	a	b	b	$							
1	c	a	b	a	b	c	b	a	b	a	b	b	$	
9	b	a	b	b	$									
3	b	a	b	c	b	a	b	a	b	b	$			
11	b	b	$											
6	c	b	a	b	a	b	b	$						
8	a	b	a	b	b	$								
2	a	b	a	b	c	b	a	b	a	b	b	$		
10	a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

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

	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	$	← 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 $	

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



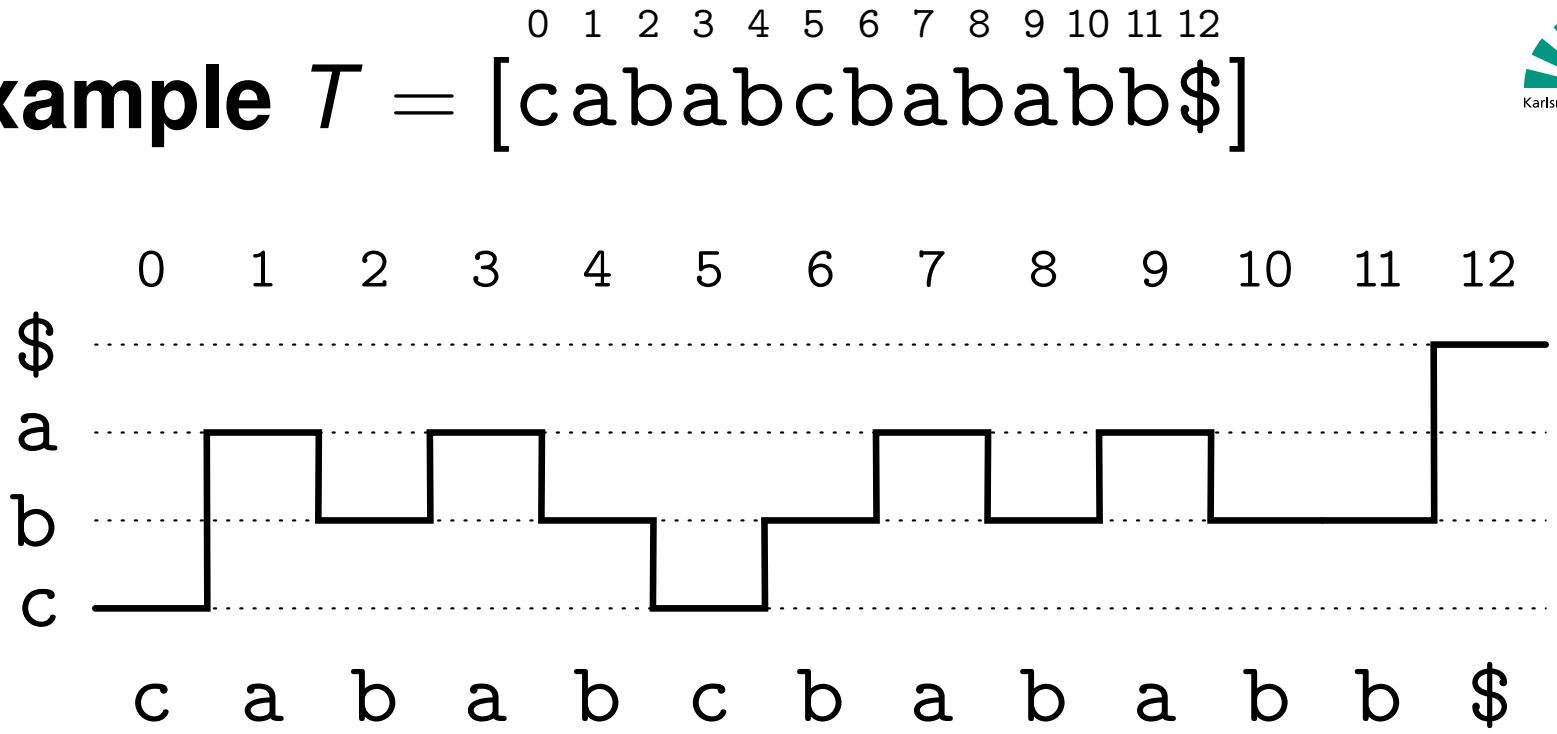
Inducing Suffixes [KA05], [NZC09]

Idea of inducing suffixes stems from [KA05], but turning this idea into an efficient and fast algorithm is not easy.

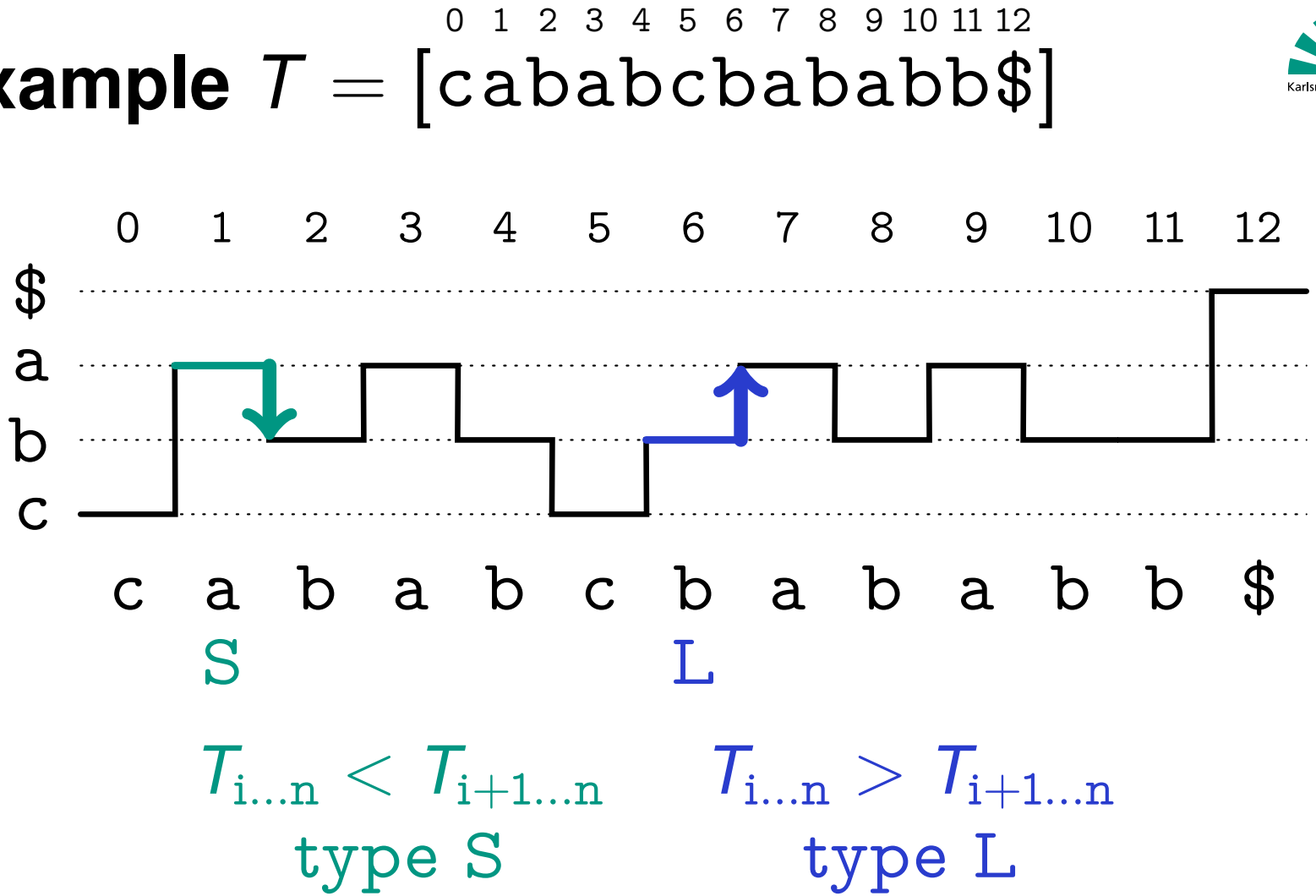
Today, most practical fast suffix sorting algorithms use inducing.

- Fastest internal memory algorithm: divsufsort [Mor06] – highly engineering, uses inducing, but is not recursive, hence theoretically not linear.
- Best internal memory algorithm: SAIS [NZC09] and OSACA [Non11] – two-phase inducing with recursion, linear worst-case time, OSACA: $5n + \mathrm{O}(1)$ optimal space.

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



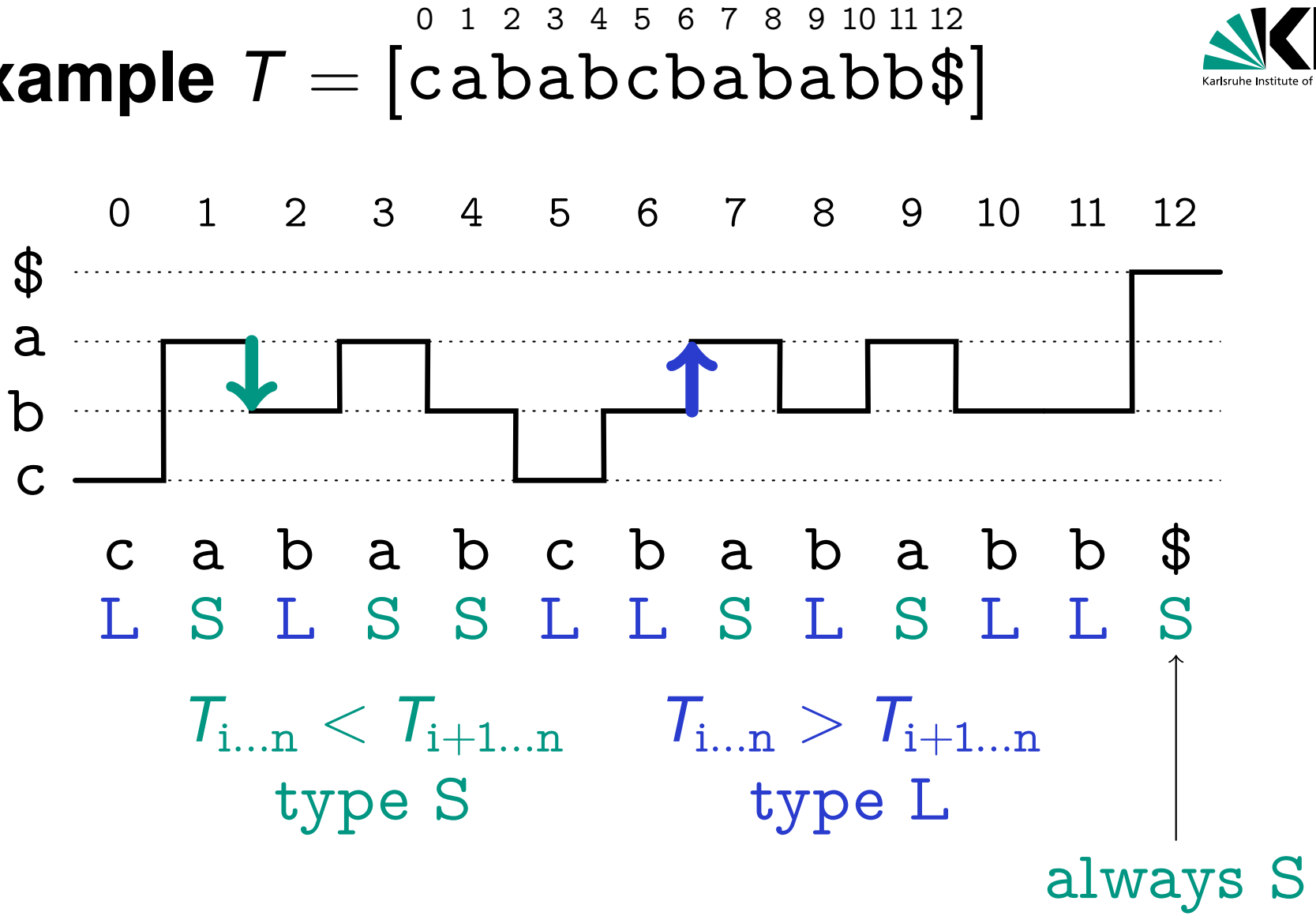
Example $T = [\texttt{cababcbababb\$}]$



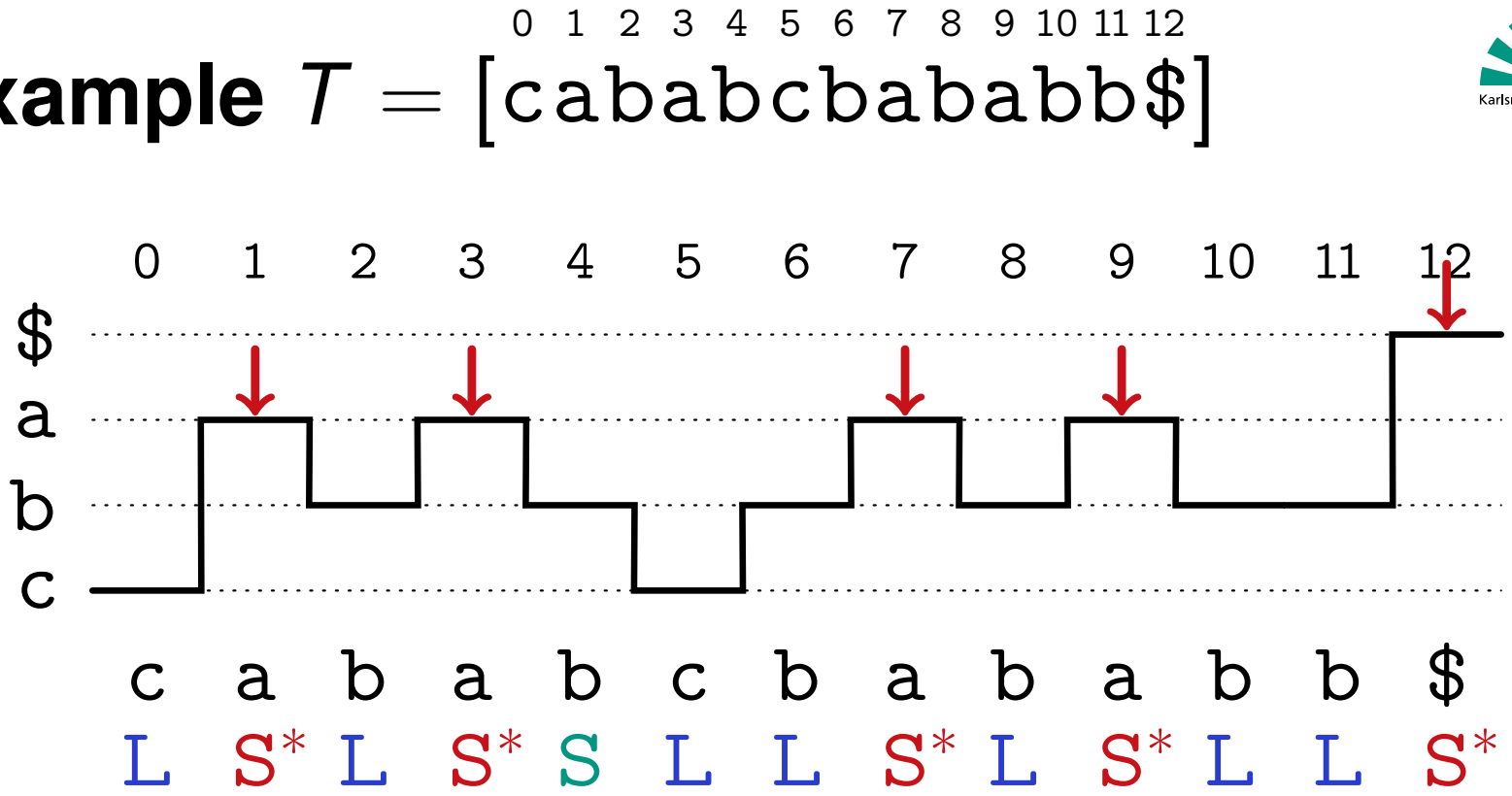
$T_{i\ldots n} < T_{i+1\ldots n}$ type S

$T_{i\ldots n} > T_{i+1\ldots n}$ type L

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



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

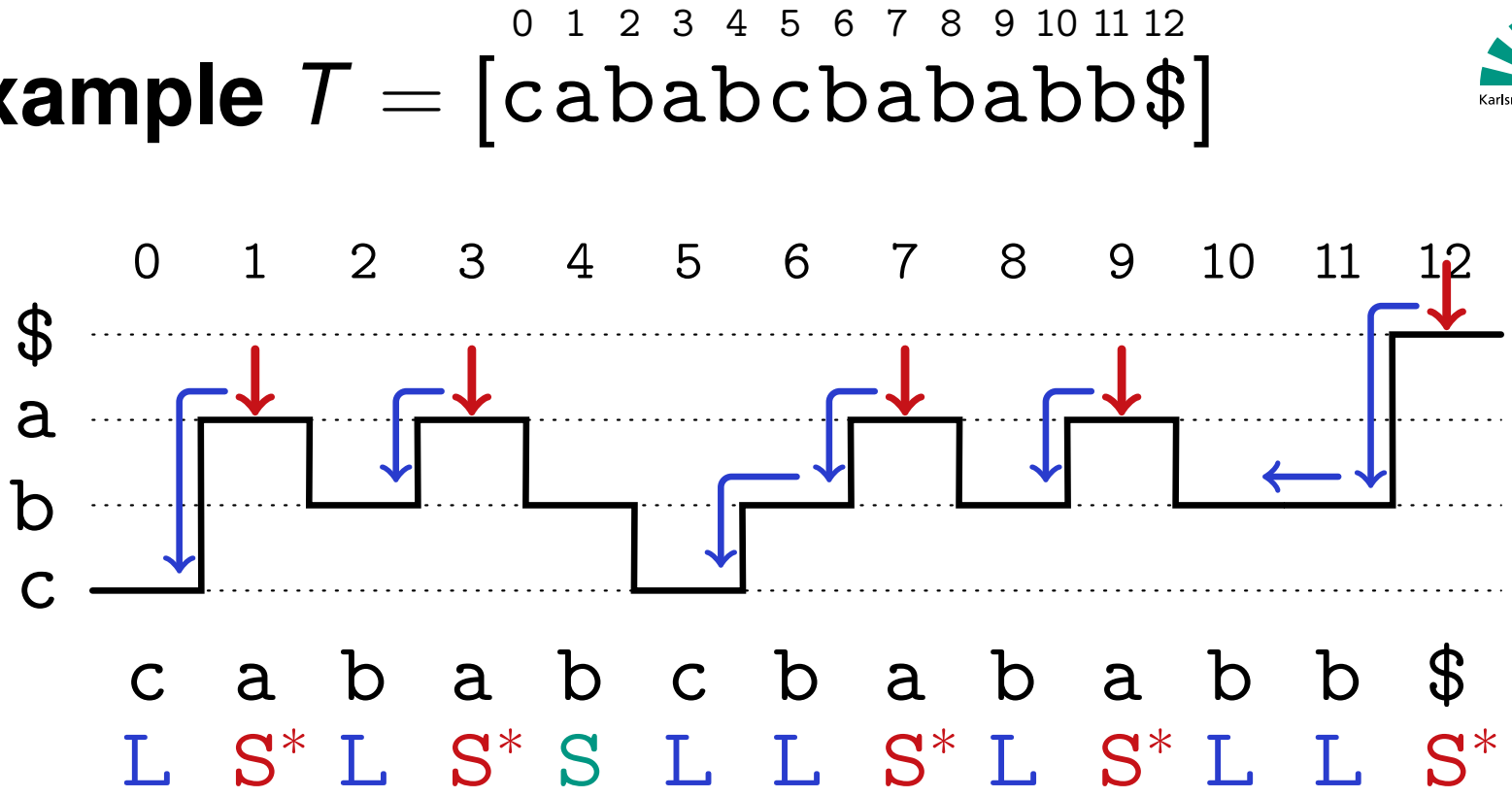


S^* = "left-most" type S suffix

except suffix 0, which is never S^*

[NZC09] calls them LMS-suffixes

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

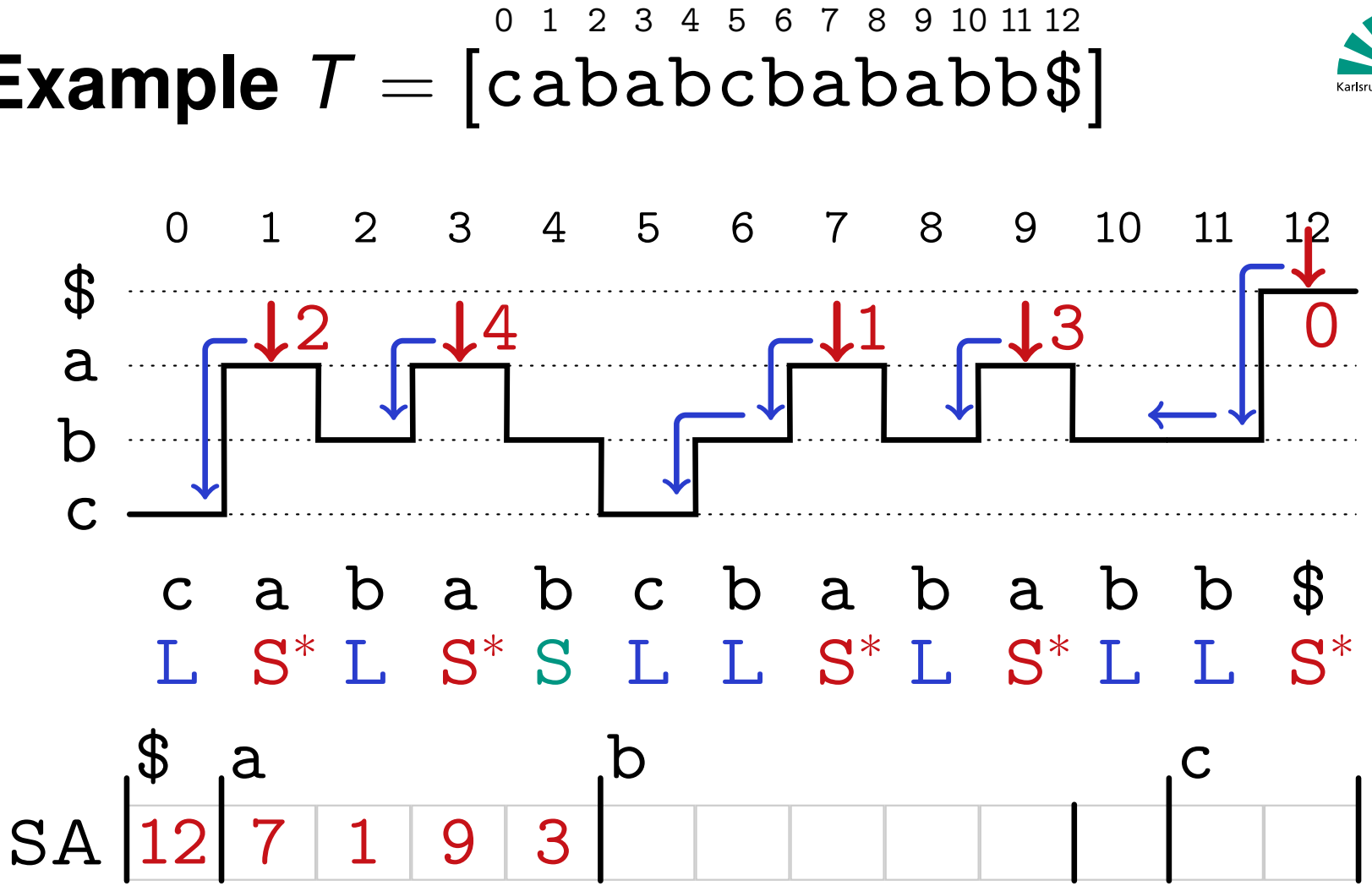


S^* = "left-most" type S suffix

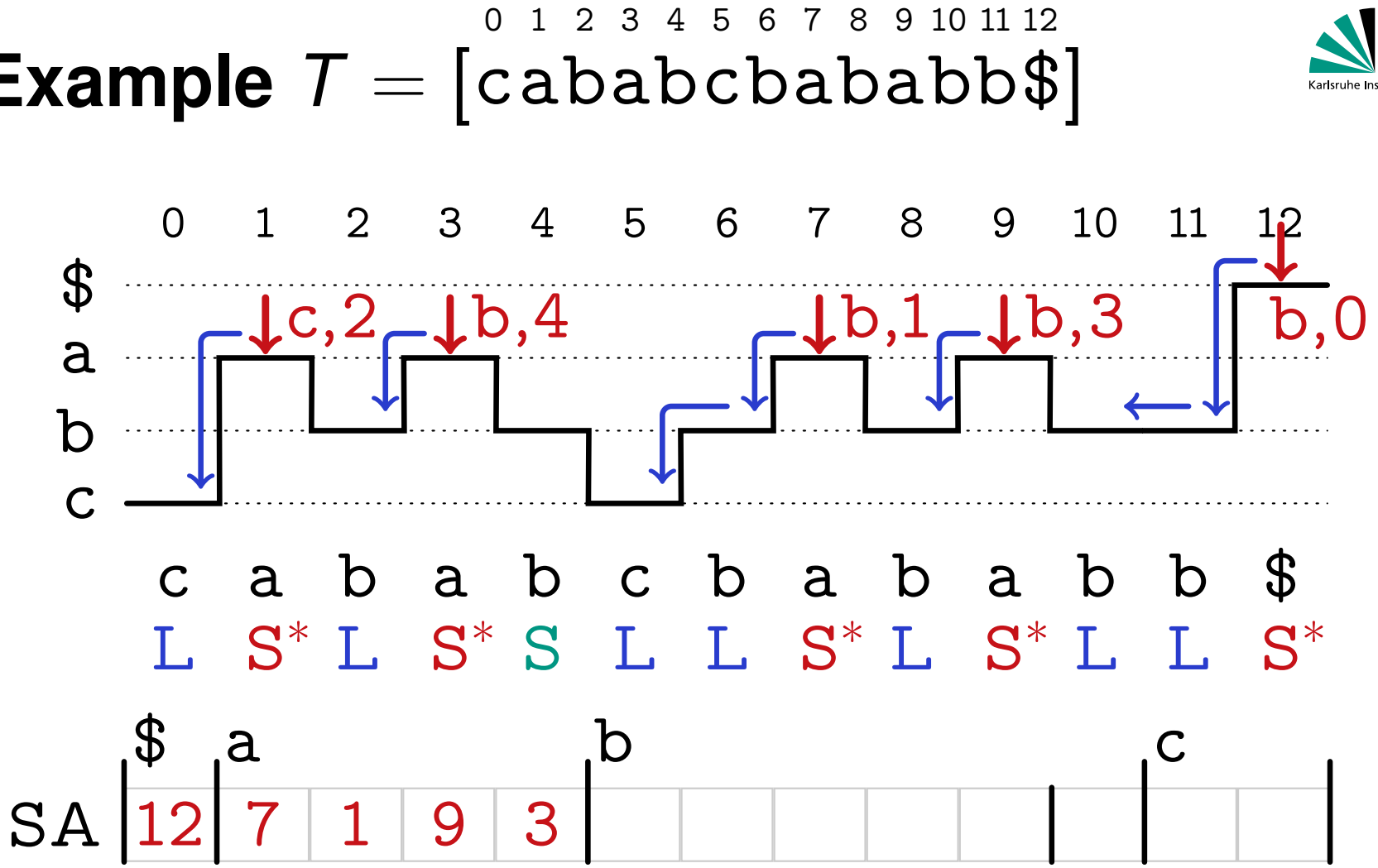
except suffix 0, which is never S^*

[NZC09] calls them LMS-suffixes

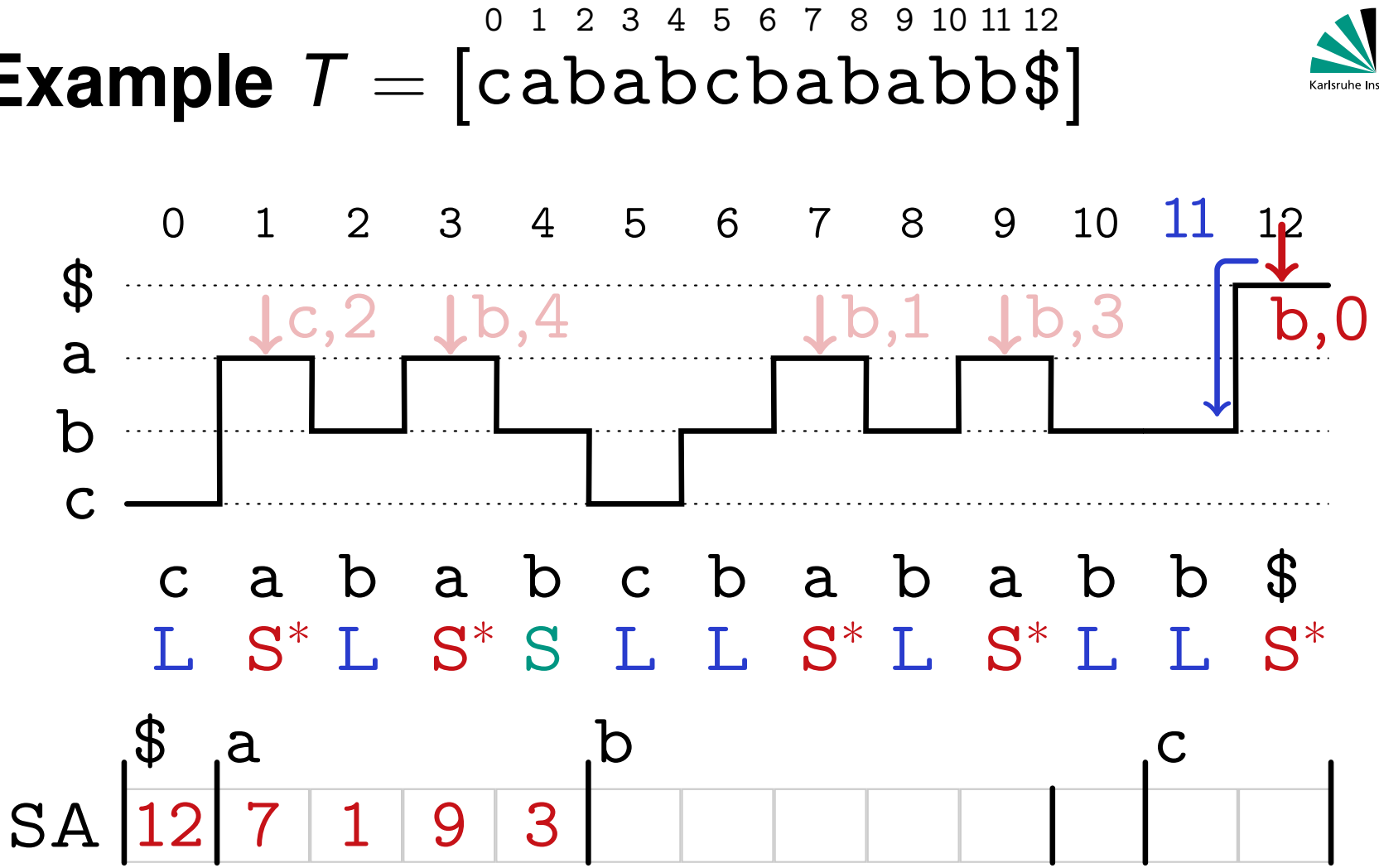
Example $T = [\texttt{cababcbababb\$}]$



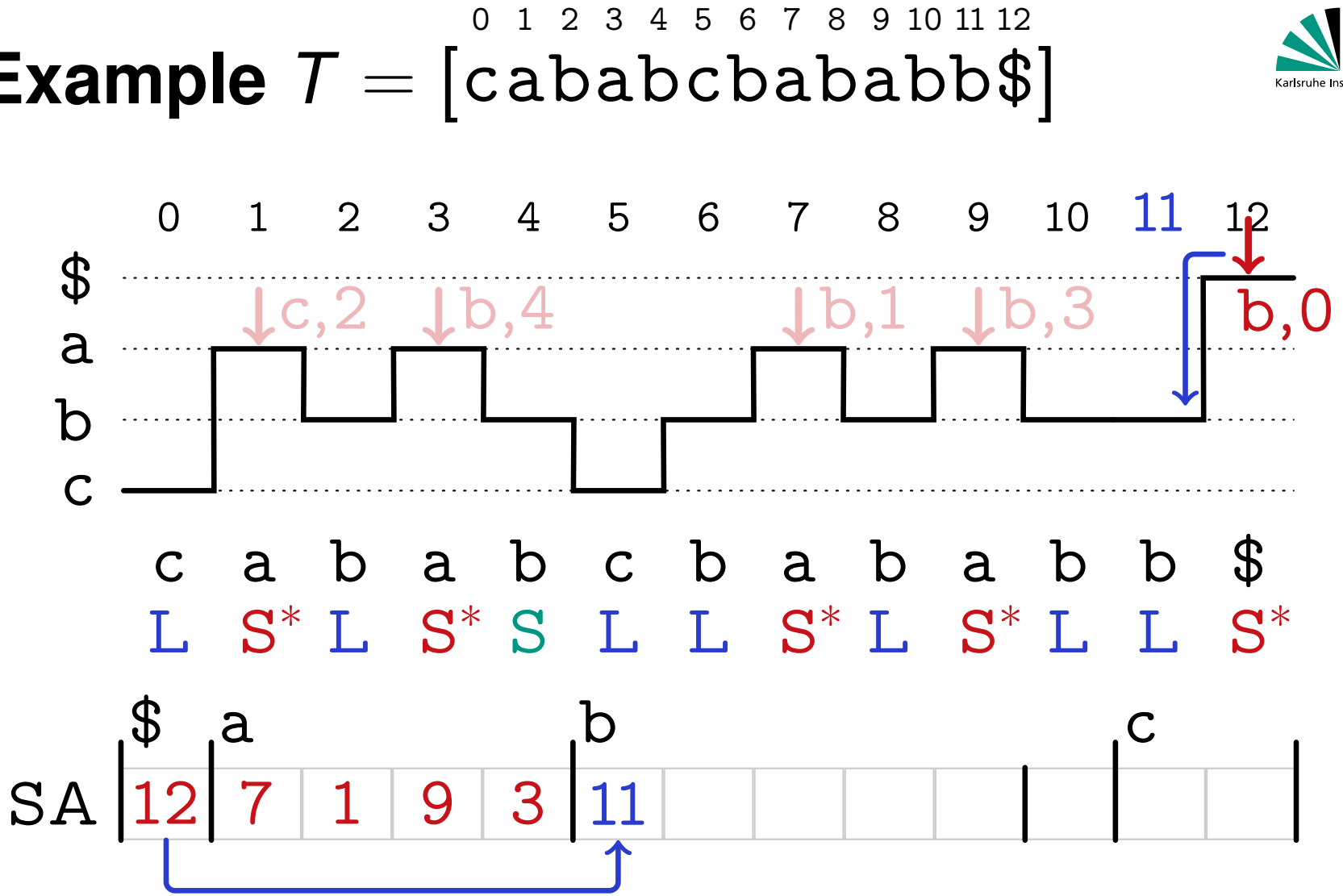
Example $T = [\texttt{cababcbababb\$}]$



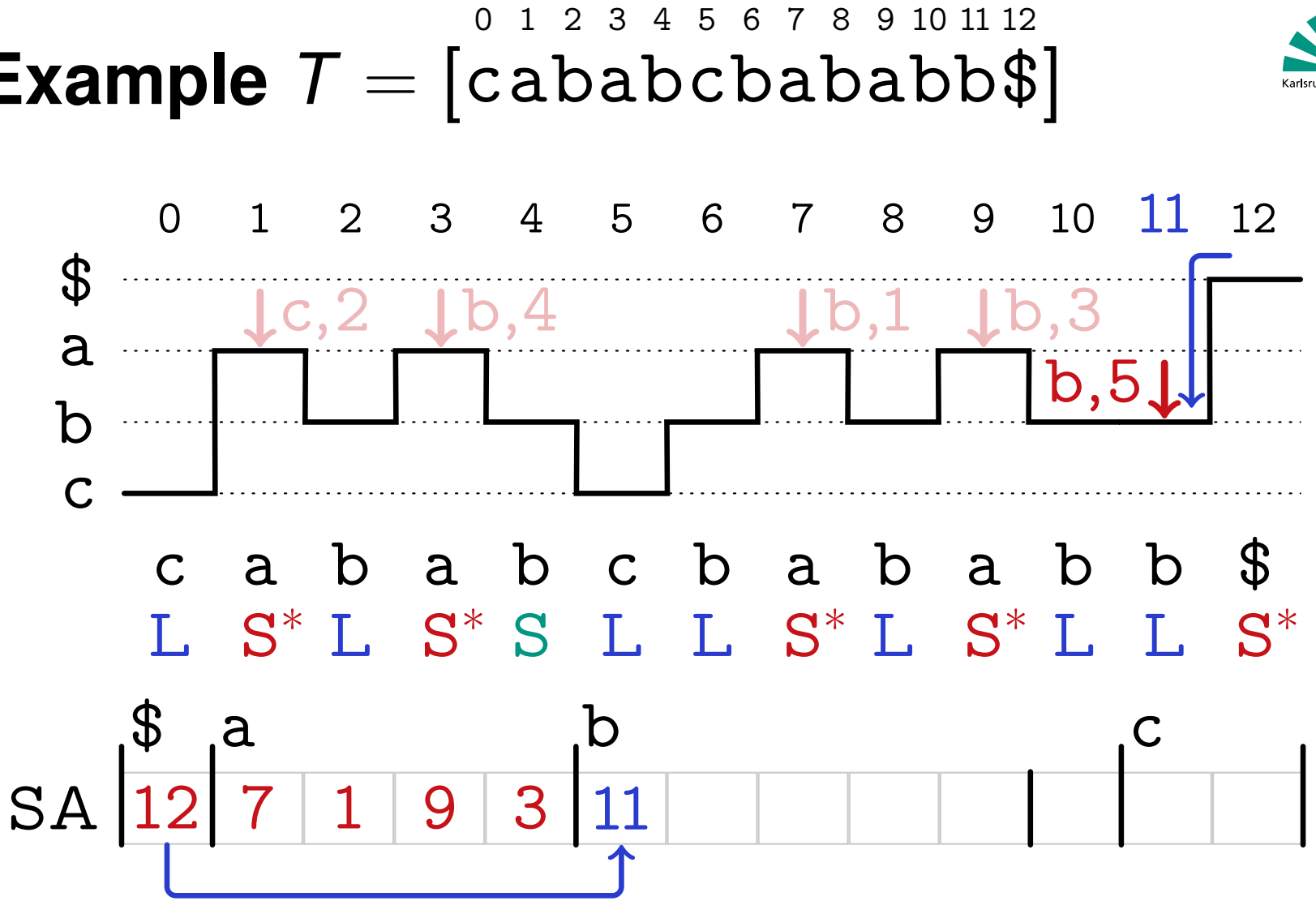
Example $T = [\texttt{cababcbababb\$}]$



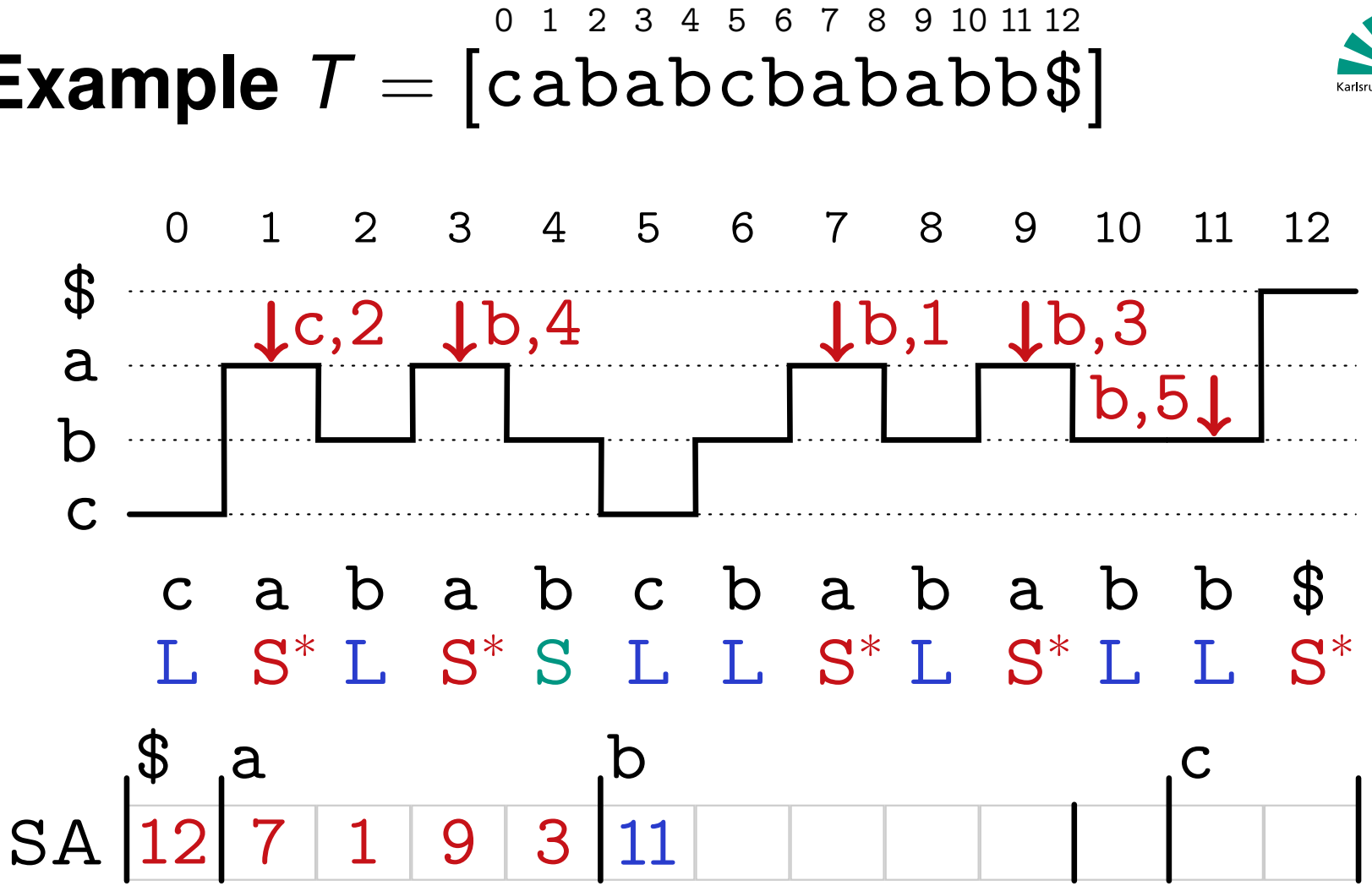
Example $T = [\texttt{cababcbababb\$}]$



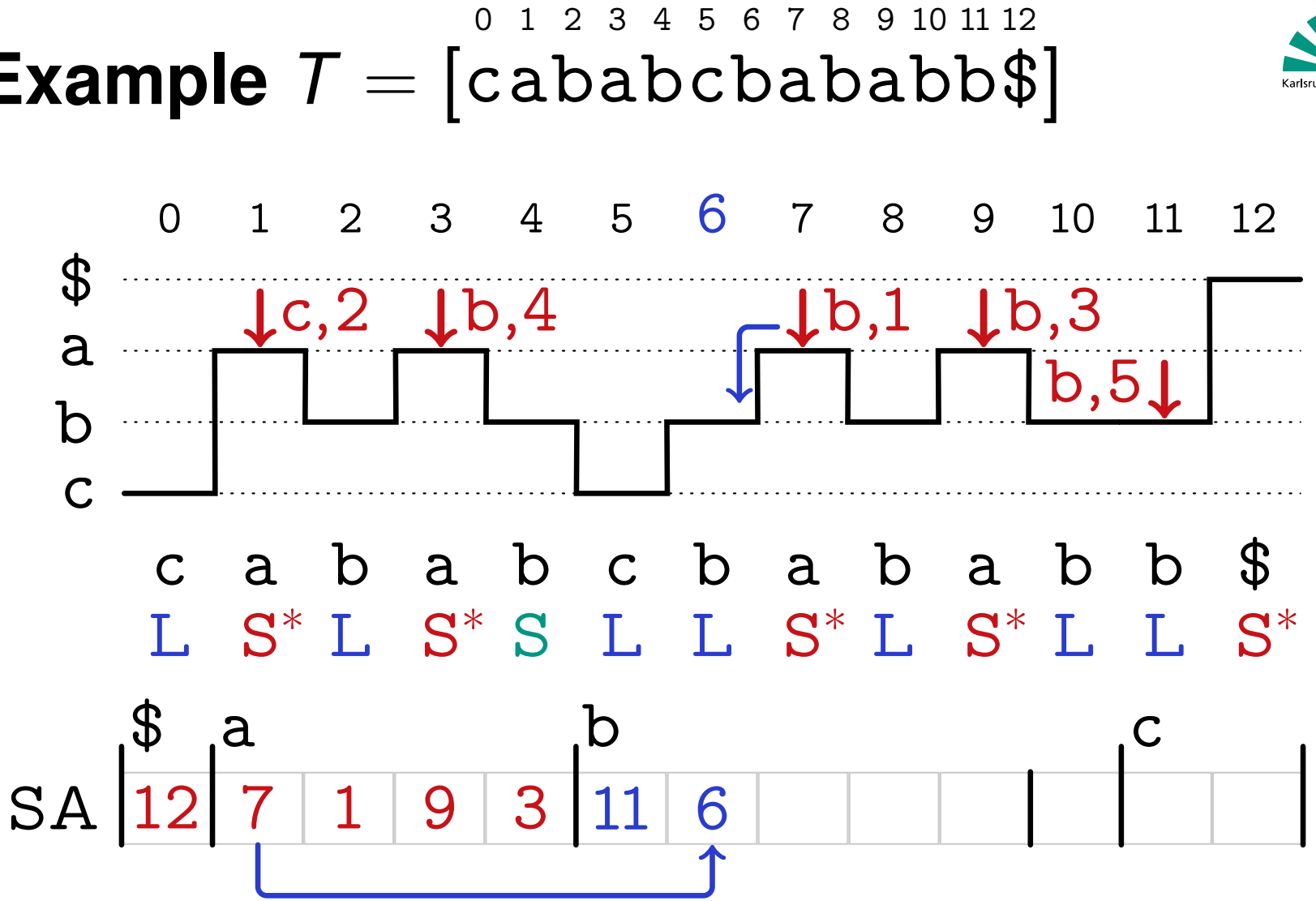
Example $T = [\texttt{cababcbababb\$}]$



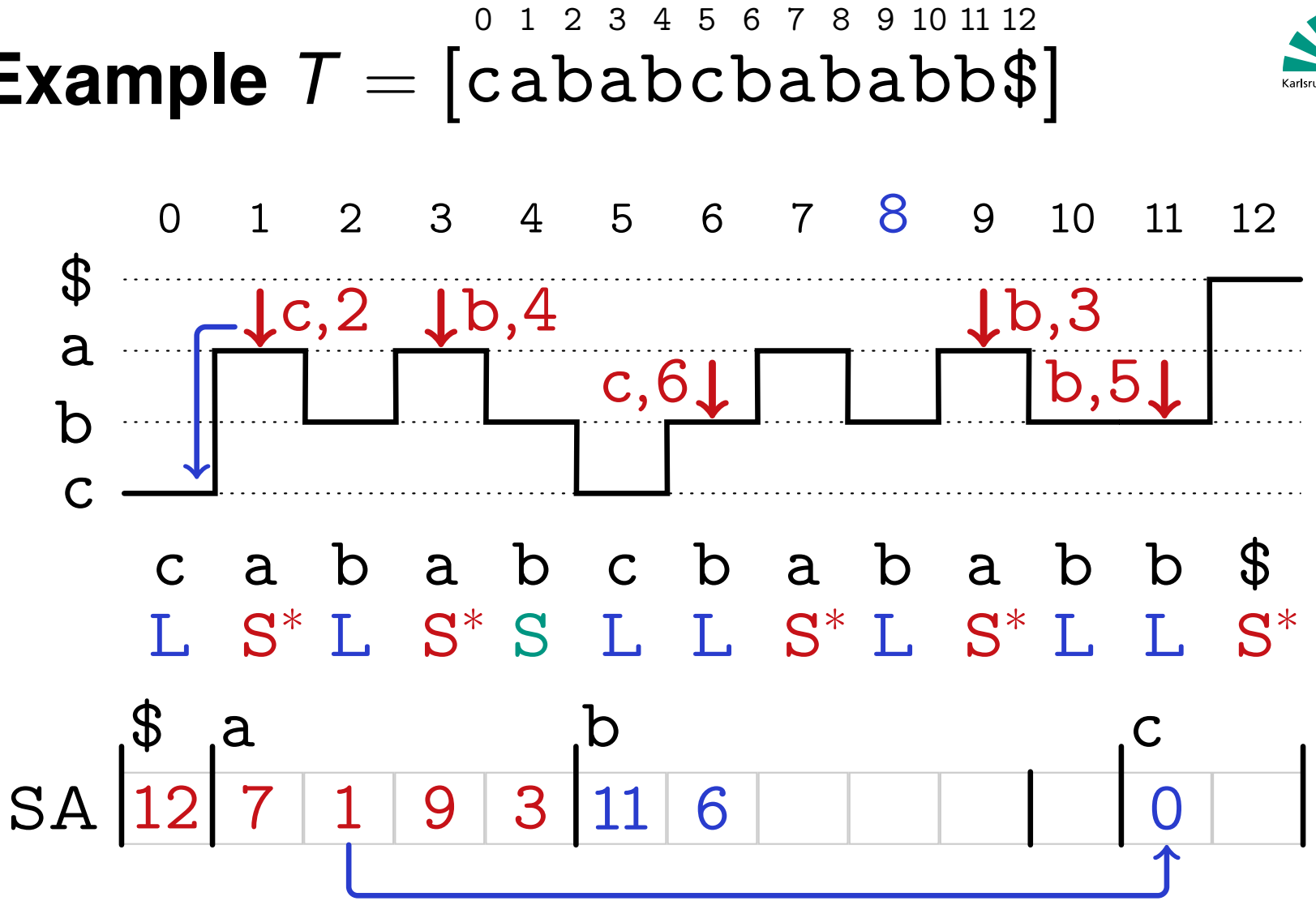
Example $T = [\texttt{cababcbababb\$}]$



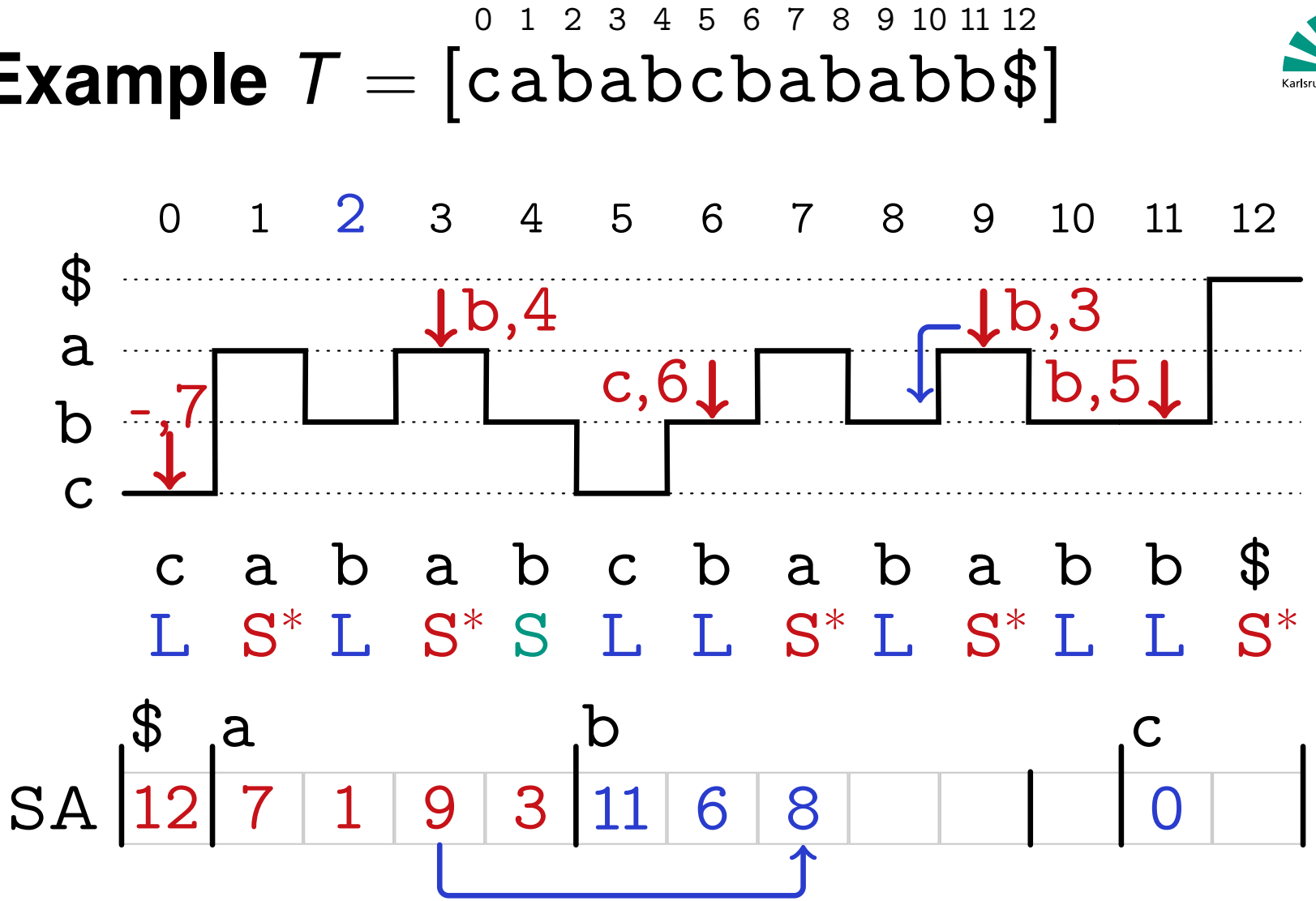
Example $T = [\texttt{cababcbababb\$}]$



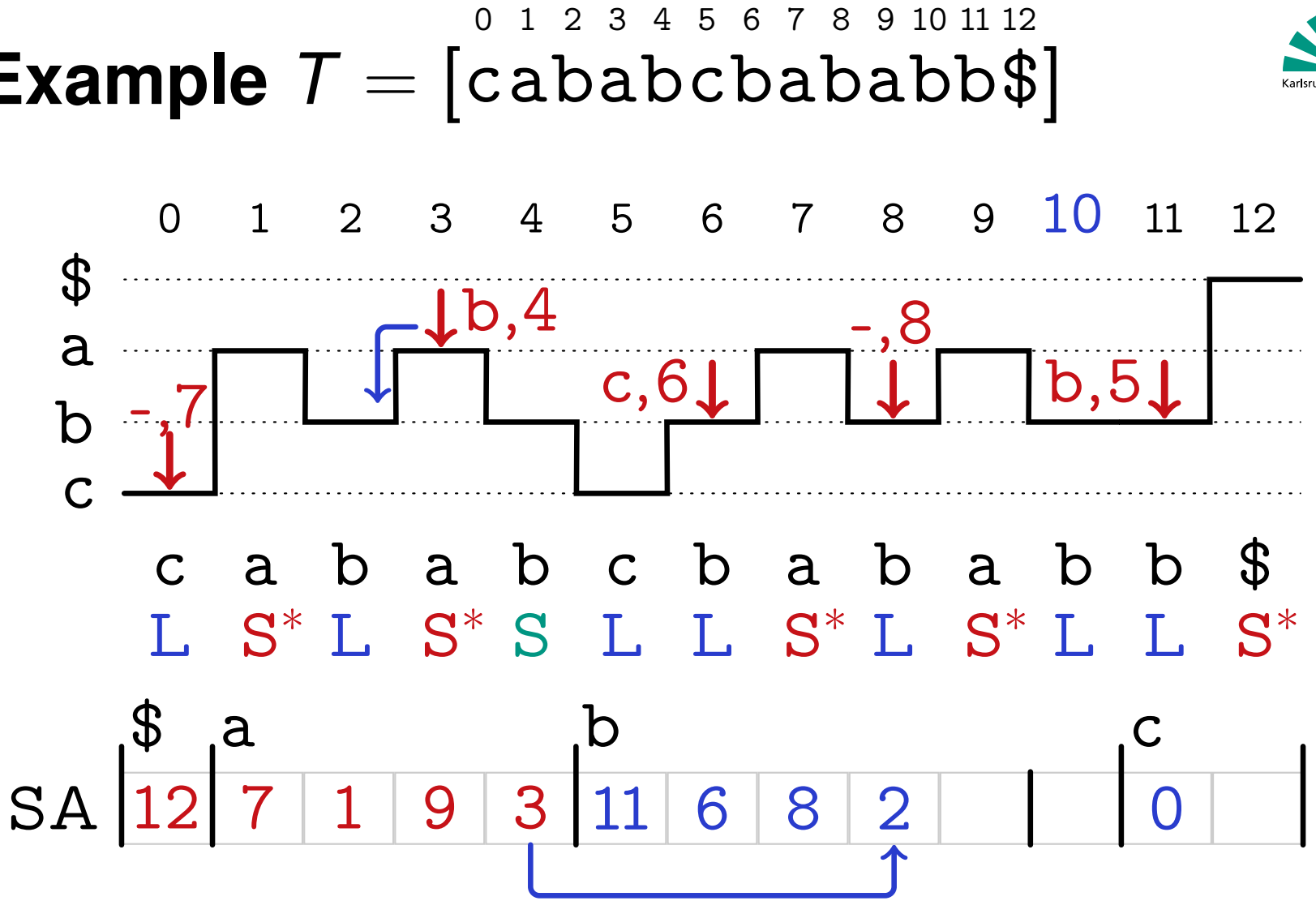
Example $T = [\texttt{cababcbababb\$}]$



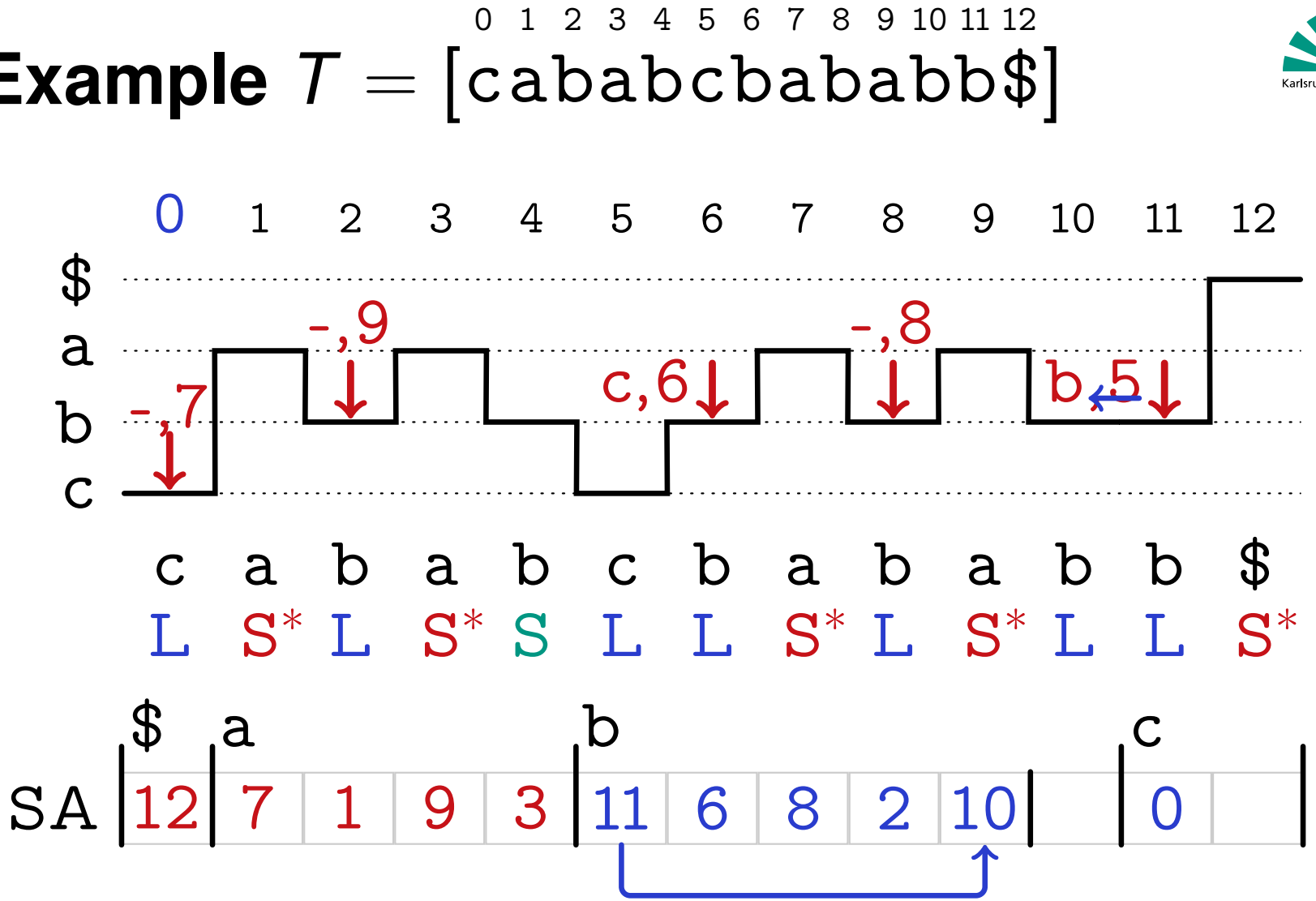
Example $T = [\texttt{cababcbababb\$}]$



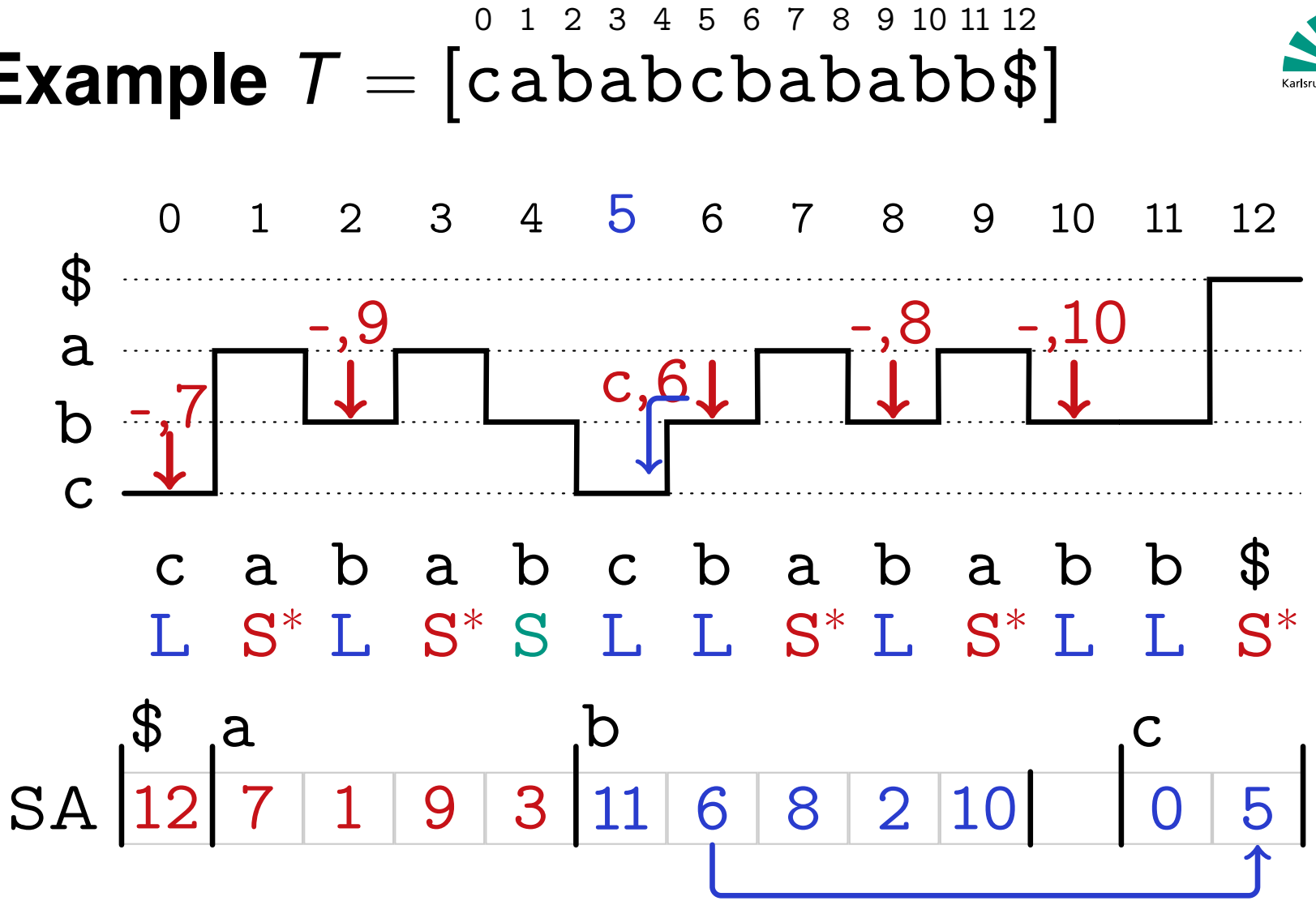
Example $T = [\texttt{cababcbababb\$}]$



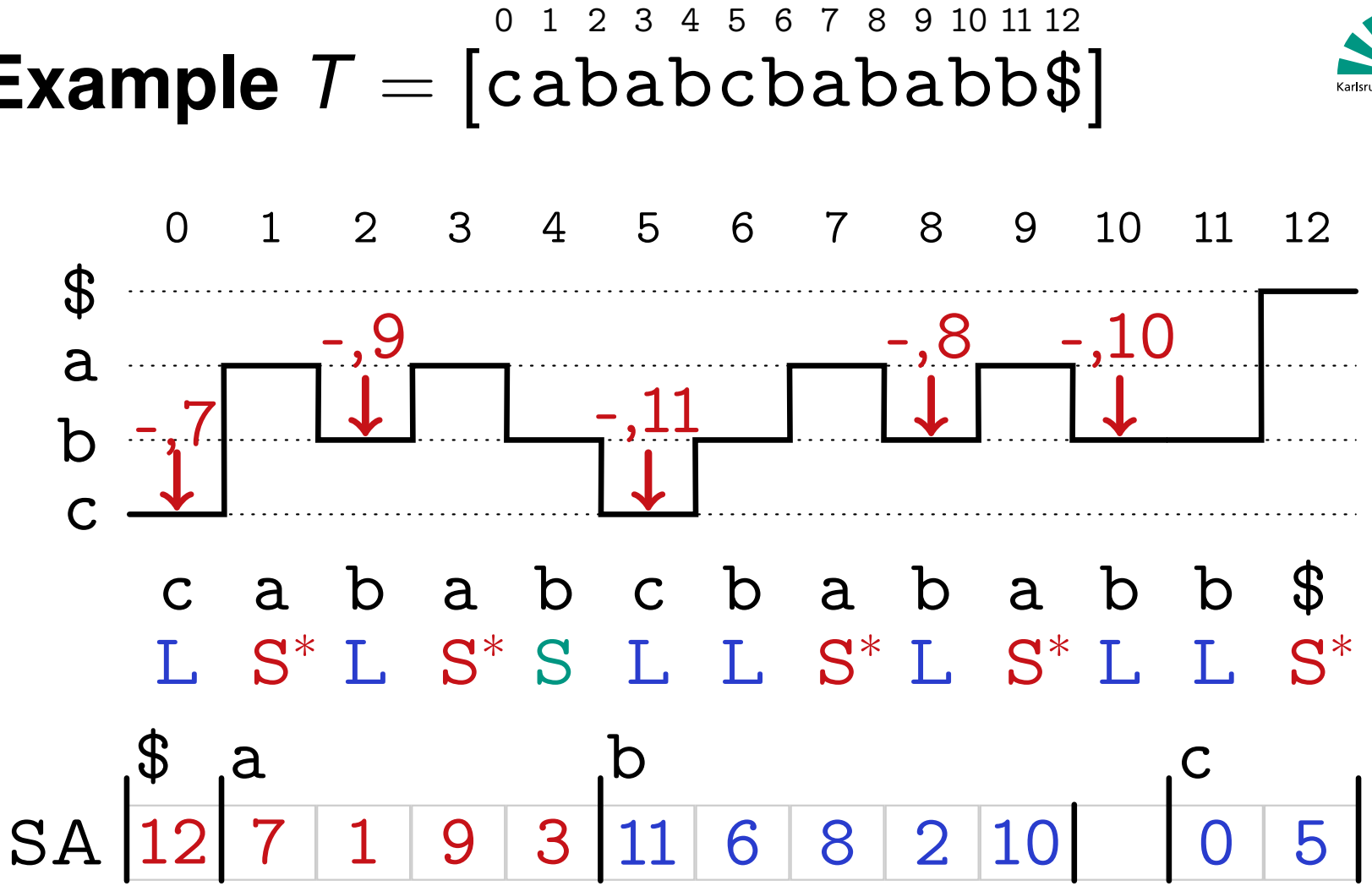
Example $T = [\texttt{cababcbababb\$}]$



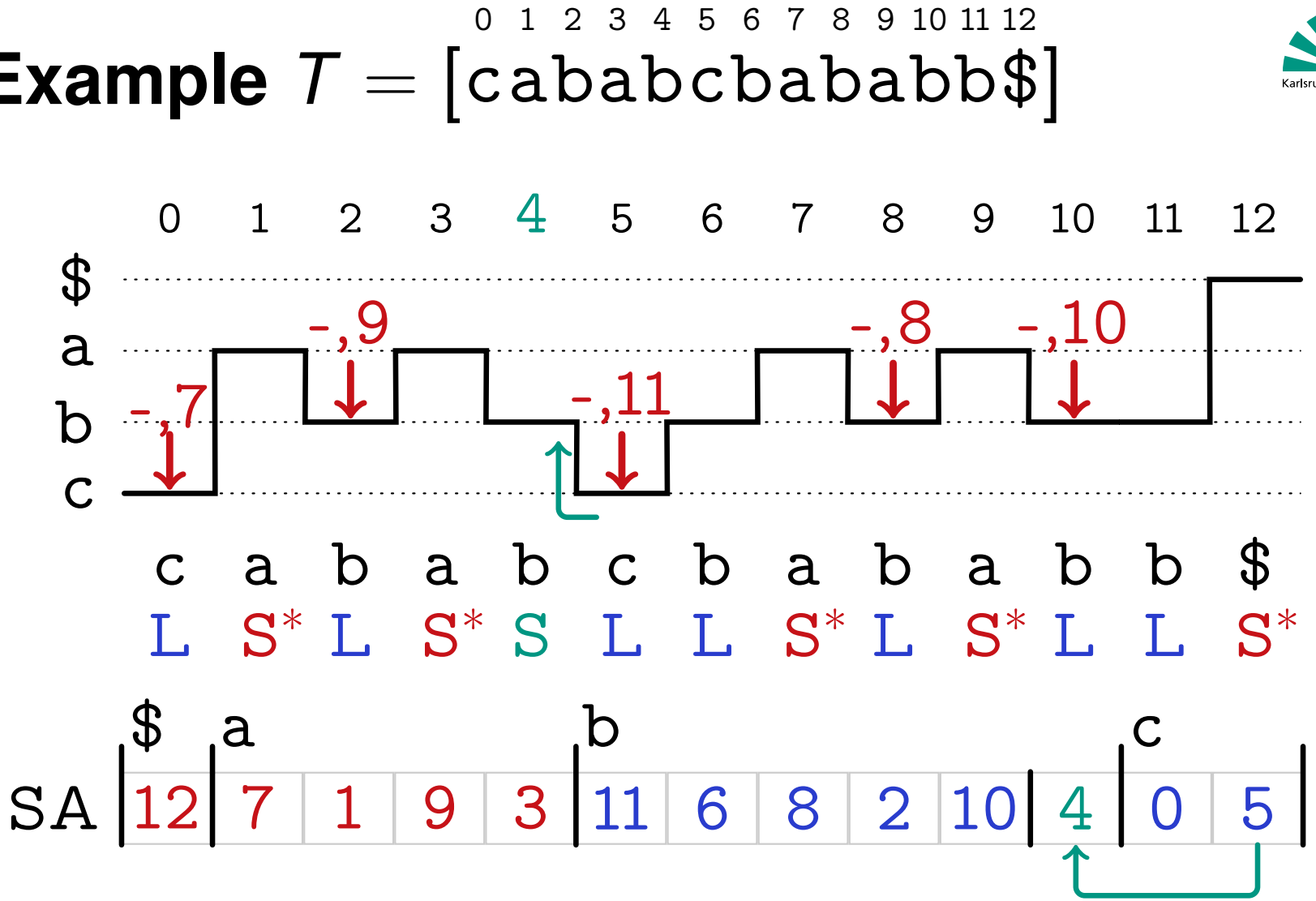
Example $T = [\texttt{cababcbababb\$}]$



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



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



SAIS – High-Level [NZC09]

1. Determine the type (L or S, and S^*) of each character by scanning right to left. Simultaneously count characters and types to determine bucket boundaries, and find S^*-substrings.
2. Phase 1: determine order of S^*-substrings (by inducing):
   1. Put all S^* positions in arbitrary order into their buckets.
   2. Induce from S^*s and Ls all L positions by scanning left-to-right.
   3. Induce from Ls and Ss all S positions by scanning right-to-left.
3. If S^*-substrings are not unique, recursively calculate ranks on lexicographic names of S^*-substrings.
4. Phase 2: deliver suffix array from known S^*-substrings ranks:
   1. Put all S^* into buckets in correct lexicographic order.
   2. Induce all L positions by scanning left-to-right.
   3. Induce all S positions by scanning right-to-left.

SAIS – Scanning Step, Properties

Induce from S^*s and Ls all L positions by scanning left-to-right.

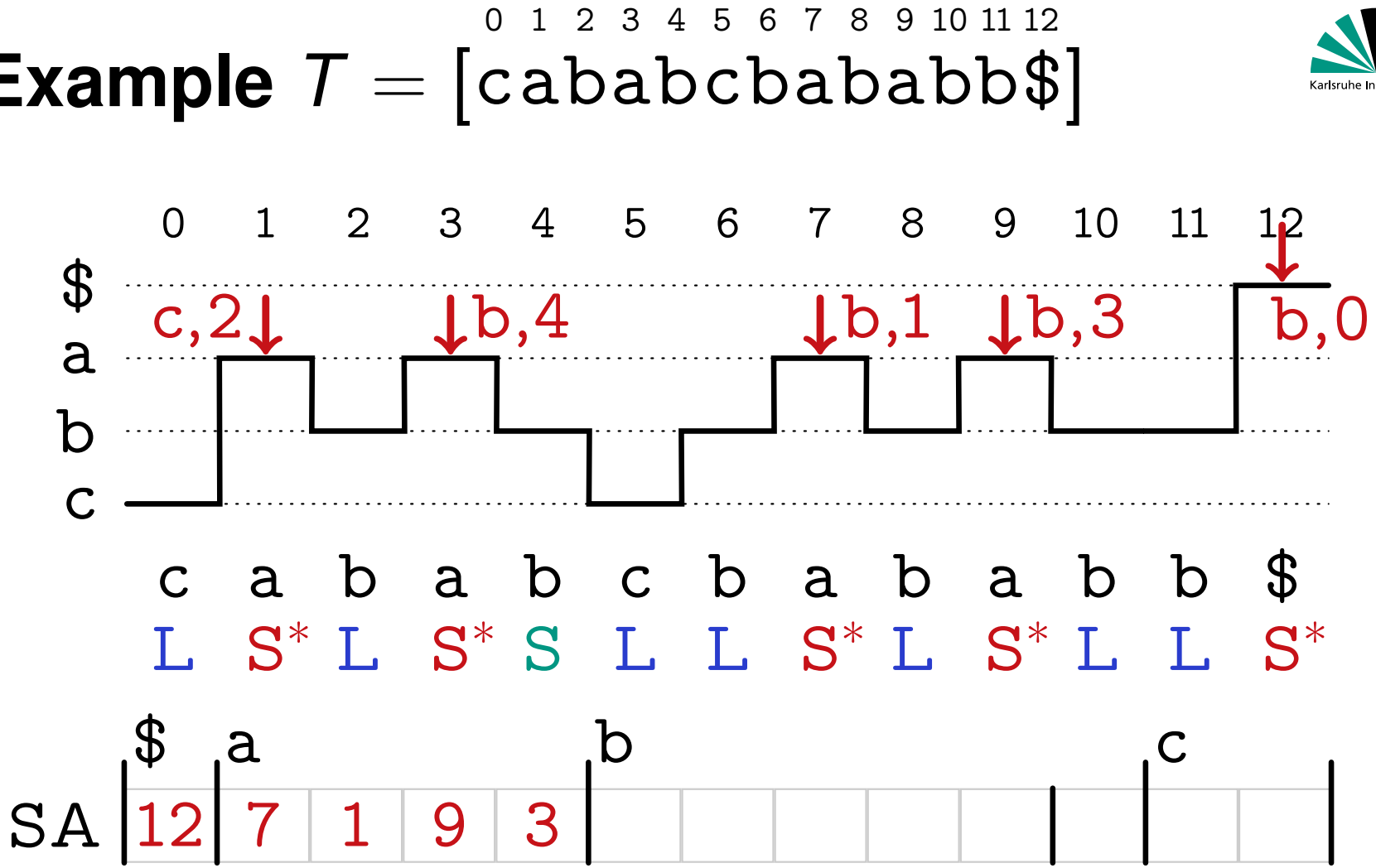
1. For each suffix i, regard type of $T[i-1]$. If type is not L, skip.
2. If type is S, put $i-1$ into the bucket $T[i-1]$ and advance bucket pointer.

Recursion on array of lexnames of S^*-substrings. Each S^*-suffix defines a S^*-substring by taking the prefix up and including the next S^*-position. The sentinel \$ is a S^*-substring by itself.

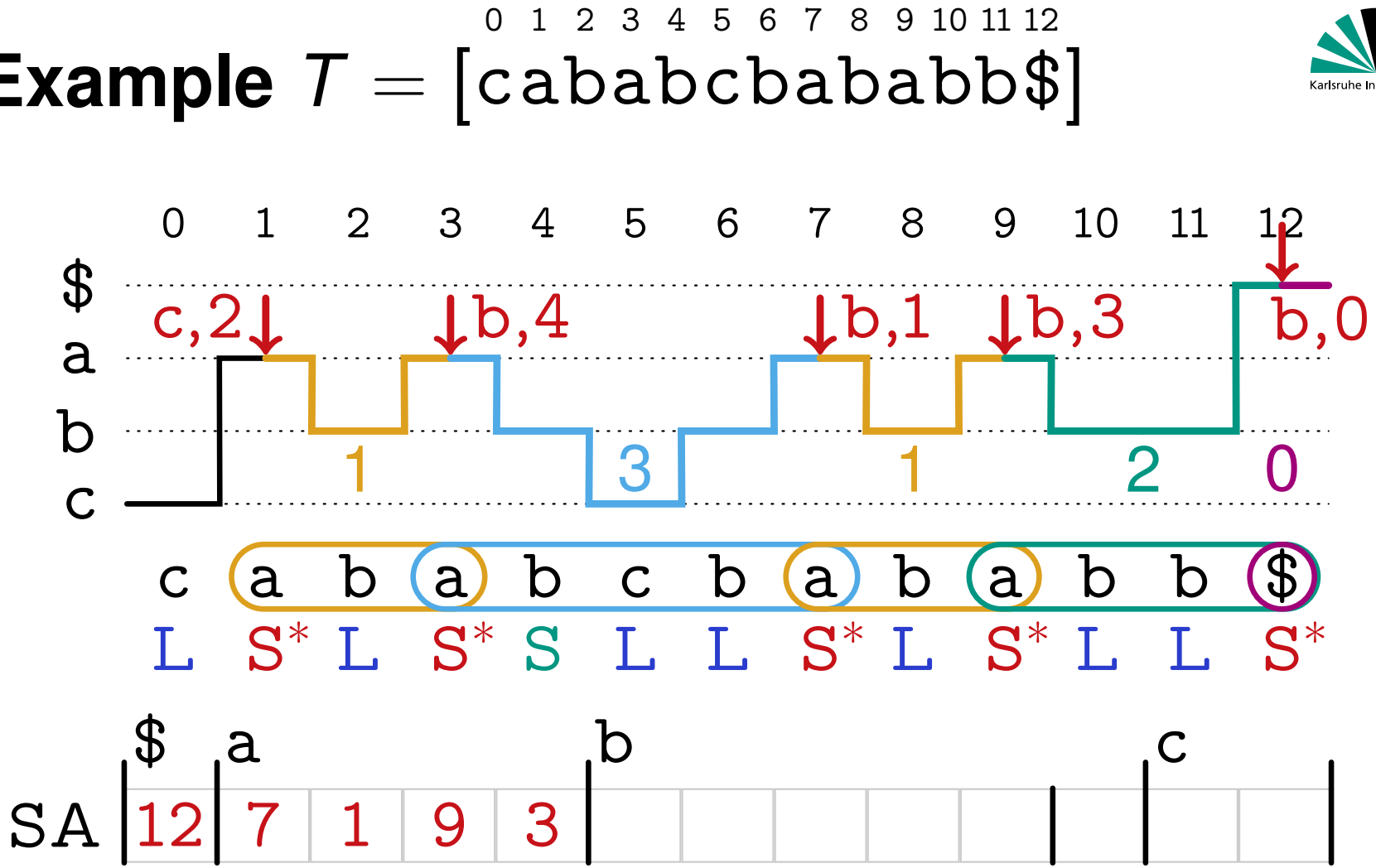
Time Complexity: $O(n)$

- Only linear scans per recursion step.
- Recursion on at most $\frac{n}{2}$ S^*-substrings.

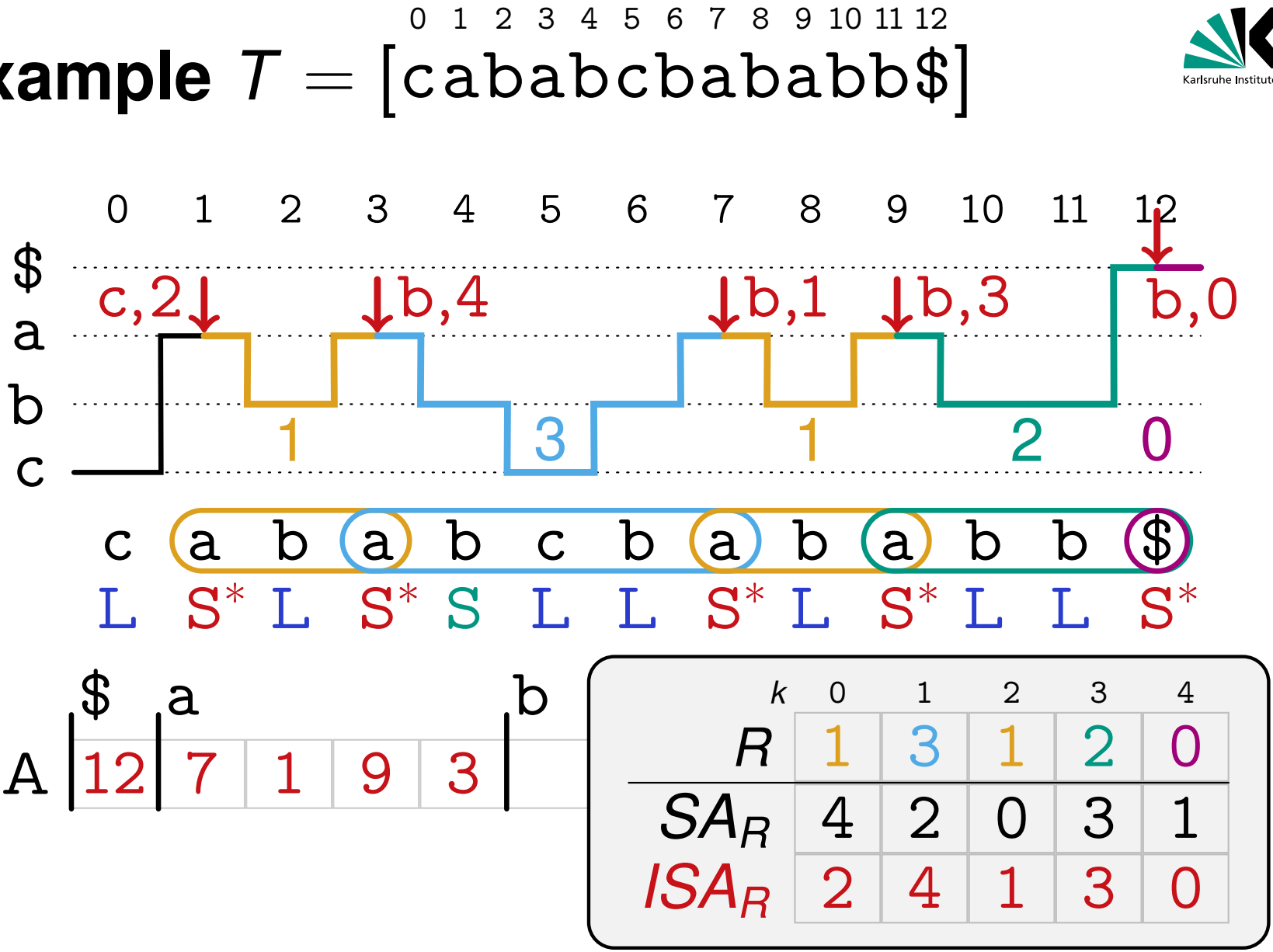
Example $T = [\texttt{cababcbababb\$}]$



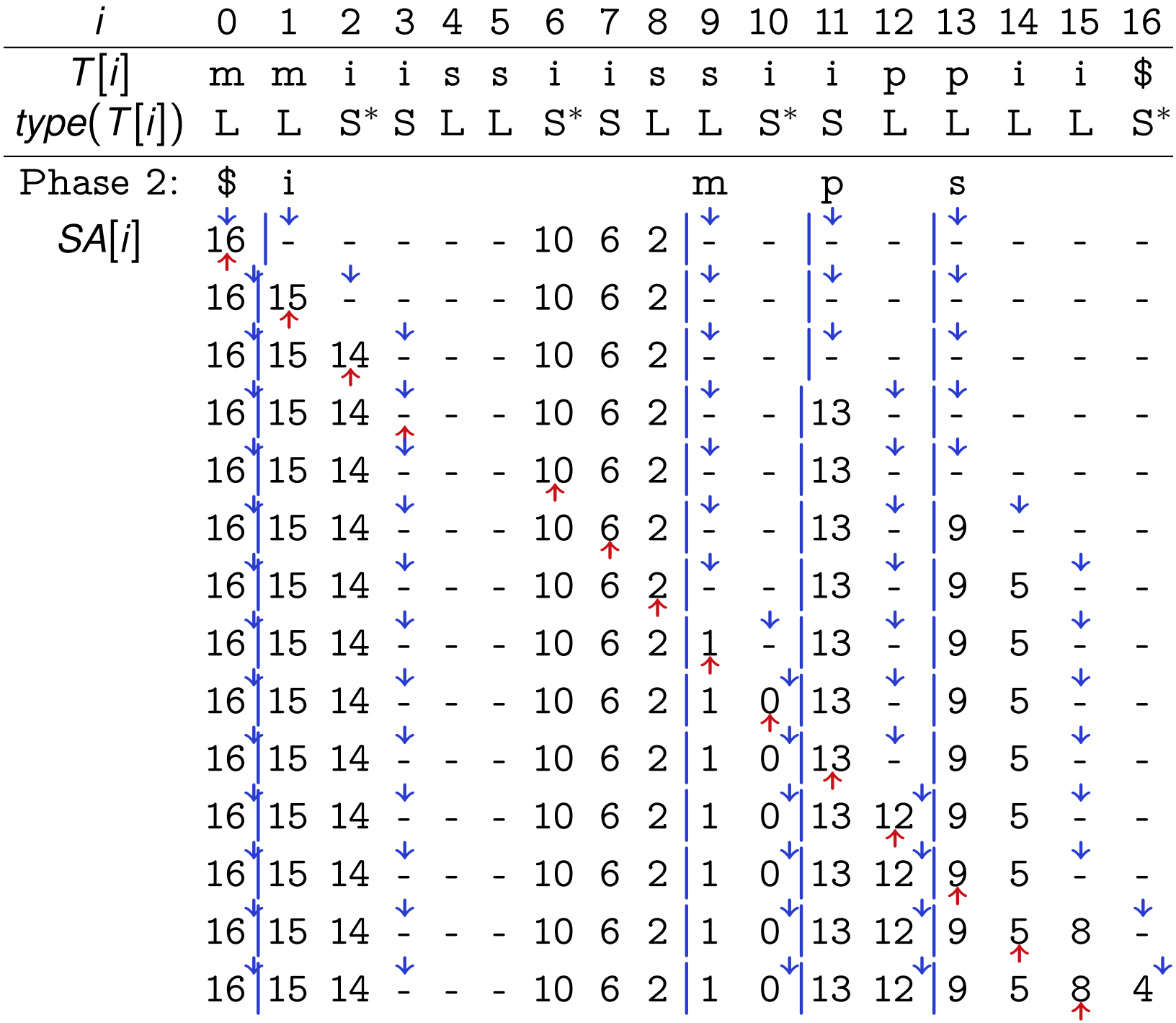
Example $T = [\texttt{cababcbababb\$}]$



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



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


i 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
T[i] m m i i s s i i s s i i p p i i $
type(T[i]) L L S* S L L S* S L L S* S L L L L S*
Phase 2: $ i m p s
SA[i]
16 - - - - - 10 6 2 - - - - - - - -
16 15 - - - - 10 6 2 - - - - - - - -
16 15 14 - - - 10 6 2 - - - - - - - -
16 15 14 - - - 10 6 2 - - 13 - - - - -
16 15 14 - - - 10 6 2 - - 13 - - - - -
16 15 14 - - - 10 6 2 - - 13 - 9 - - -
16 15 14 - - - 10 6 2 - - 13 - 9 5 - -
16 15 14 - - - 10 6 2 1 - 13 - 9 5 - -
16 15 14 - - - 10 6 2 1 0 13 - 9 5 - -
16 15 14 - - - 10 6 2 1 0 13 - 9 5 - -
16 15 14 - - - 10 6 2 1 0 13 12 9 5 - -
16 15 14 - - - 10 6 2 1 0 13 12 9 5 - -
16 15 14 - - - 10 6 2 1 0 13 12 9 5 8 -
16 15 14 - - - 10 6 2 1 0 13 12 9 5 8 4

SAIS – Example

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T[i]$	m	m	i	i	s	s	i	i	s	s	i	i	p	p	i	i	$
$type(T[i])$	L	L	S*	S	L	L	S*	S	L	L	S*	S	L	L	L	L	S*
Phase 2:	$	i								m		p		s			
$SA[i]$	16	15	14	-	-	-	10	6	2	1	0	13	12	9	5	8	4
	16	15	14	-	-	-	10	6	3	1	0	13	12	9	5	8	4
	16	15	14	-	-	-	10	7	3	1	0	13	12	9	5	8	4
	16	15	14	-	-	-	10	7	3	1	0	13	12	9	5	8	4
	16	15	14	-	-	-	11	7	3	1	0	13	12	9	5	8	4
	16	15	14	-	-	-	11	7	3	1	0	13	12	9	5	8	4
	16	15	14	-	-	2	11	7	3	1	0	13	12	9	5	8	4
	16	15	14	-	6	2	11	7	3	1	0	13	12	9	5	8	4
	16	15	14	10	6	2	11	7	3	1	0	13	12	9	5	8	4

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.

[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.

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

Bibliography II

[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.

[KS03] Juha Kärkkäinen and Peter Sanders.
Simple linear work suffix array construction.
Proc. ICALP, 2719:943–955, 2003.

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.

[NZC09] Ge Nong, Sen Zhang, and Wai Hong Chan.
Linear suffix array construction by almost pure induced-sorting.
In Proc. Data Compression Conf. (DCC), pages 193–202. IEEE Press, 2009.

[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.

[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.