Constructing the Bijective BWT and the Extended Burrows-Wheeler Transform in Linear Time

Hideo Bannai (Tokyo Medical and Dental University),
Juha Kärkkäinen (Helsinki Institute of Information Technology),
Dominik Köppl (Tokyo Medical and Dental University),
Marcin Piątkowski (Nicolaus Copernicus University)

[CPM ‘21]

the bijective BWT (BBWT) is
the BWT of
the Lyndon factorization
of an input text
with respect to $\prec_\omega$

the bijective BWT (BBWT) is
the BWT of
the Lyndon factorization 1.
of an input text
with respect to $\prec_\omega$ 2.

Lyndon words

- `a`
- `aabab`

Lyndon word is smaller than
- any proper suffix
- any rotation

Lyndon words

- a
- aabab

Lyndon word is smaller than
- any proper suffix
- any rotation

not Lyndon words:

- abaab (rotation aabab smaller)
- abab (abab not smaller than suffix ab)

Lyndon factorization [Chen+ '58]

- input: text T = 

- output: factorization $T_1 ... T_t$ with
  - T_x is Lyndon word
  - $T_x \geqslant_{\text{lex}} T_{x+1}$
  - factorization uniquely defined
  - linear time [Duval '88]

(Chen-Fox-Lyndon Theorem)

example

T = senescence

Lyndon factorization: s|enes|cen|ce

- s, enes, cen, and ce are Lyndon
- $s >_{lex} enes >_{lex} cen >_{lex} ce$

$\prec_\omega$ order

- $u \prec_\omega w :\Longleftrightarrow uuuu... <_{\text{lex}} wwww...$

- $\mathtt{ab} <_{\text{lex}} \mathtt{aba}$
- $\mathtt{aba} \prec_\omega \mathtt{ab}$

$\prec_\omega$ order

- $u \prec_\omega w :\Longleftrightarrow uuuu\ldots <_{\text{lex}} wwww\ldots$

- $ab <_{\text{lex}} aba$
- $aba \prec_\omega ab$

abababab⋯
ababaaba⋯

conjugates

- $T = T[1]\,T[2]\,\cdots\,T[n]$
- conjugates = cyclic shifts:
  - $T[1]\,T[2]\,\cdots\,T[n]$
  - $T[2]\,T[3]\,\cdots\,T[n]\,T[1]$
  - $\vdots$
  - $T[n]\,T[1]\,\cdots\,T[n\text{-}1]$

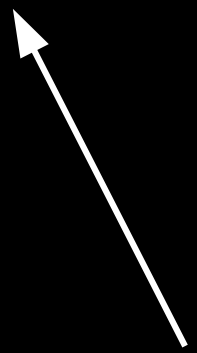
bijective BWT of senescence

s|enes|cen|ce

bijective BWT of senescence

s|enes|cen|ce

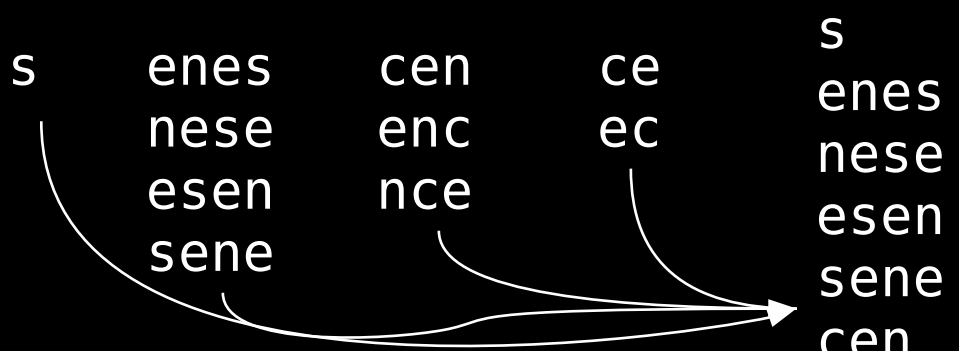
```
s    enes    cen    ce
     nese    enc    ec
     esen    nce
     sene
```



conjugates of all Lyndon factors

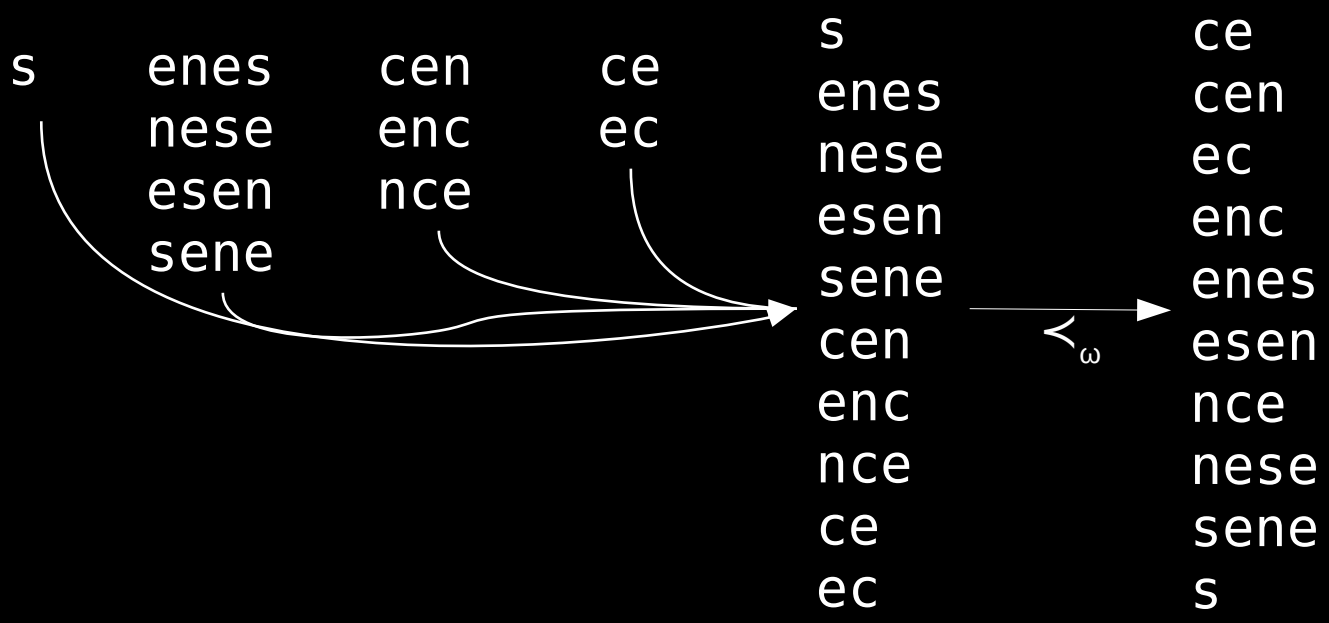
bijective BWT of senescence

s|enes|cen|ce



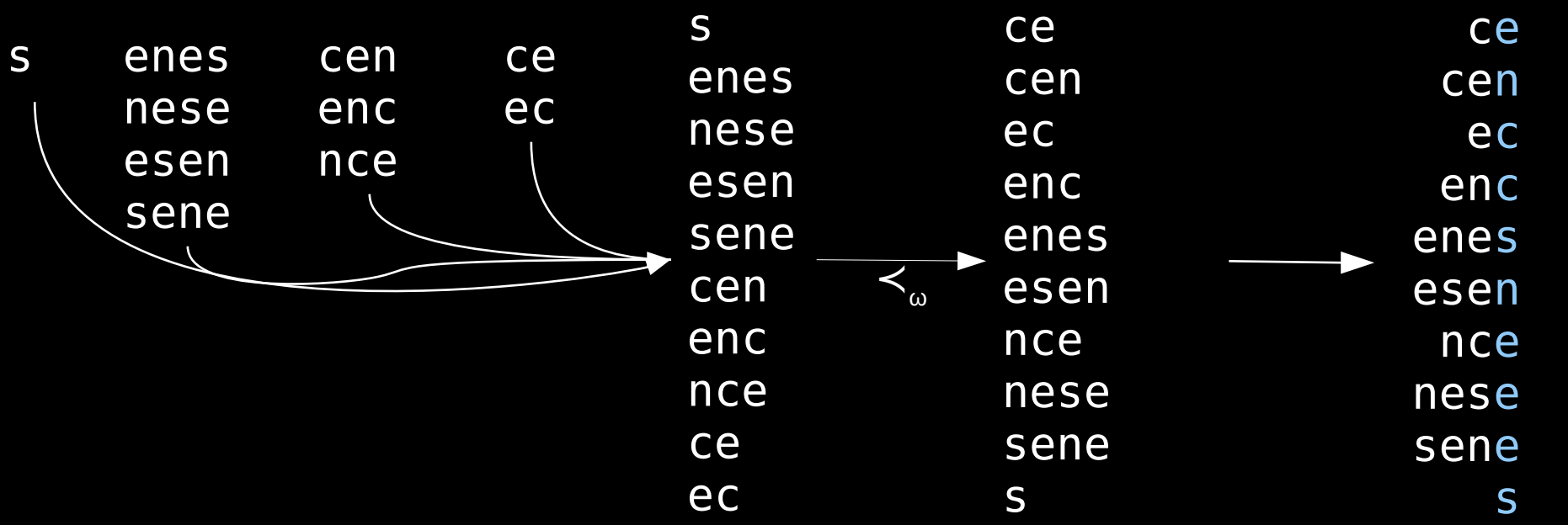
bijective BWT of senescence

s|enes|cen|ce



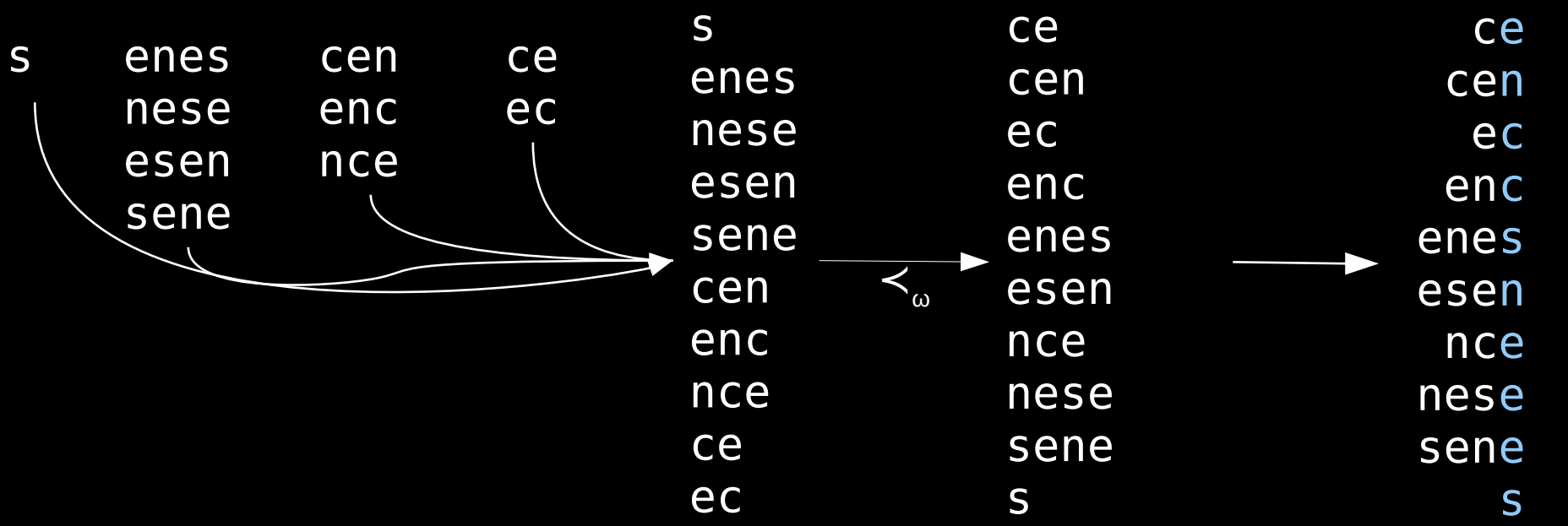
bijective BWT of senescence

s|enes|cen|ce



bijective BWT of senescence

s|enes|cen|ce



result: enccsneees

motivation

properties of BBWT :

- no $ necessary
- BBWT seems to be more compressible than BWT for some inputs

[Scott and Gill '12]

- extended BWT : special case of BBWT

[Mantaci+ ‘07]

- BBWT is indexible (full text index)

[Bannai+ '19]

however, linear time construction was only conjectured!

time

- how much time does it take to sort?
- #conjugates = number of text positions = n

  $\Rightarrow$ naively: $O(n^2)$ time

- can we use $O(n)$ time suffix sorting?

BWT of senescence

senescence

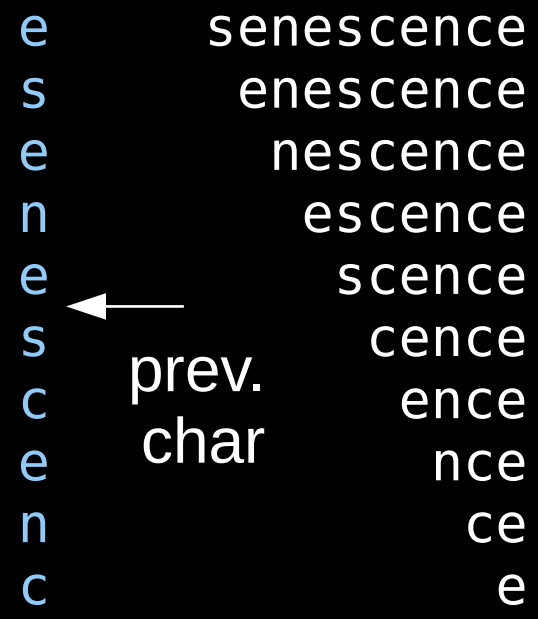
BWT of senescence

senescence

```
senescence
 enescence
  nescence
   escence
    scence
     cence
      ence
       nce
        ce
         e
```

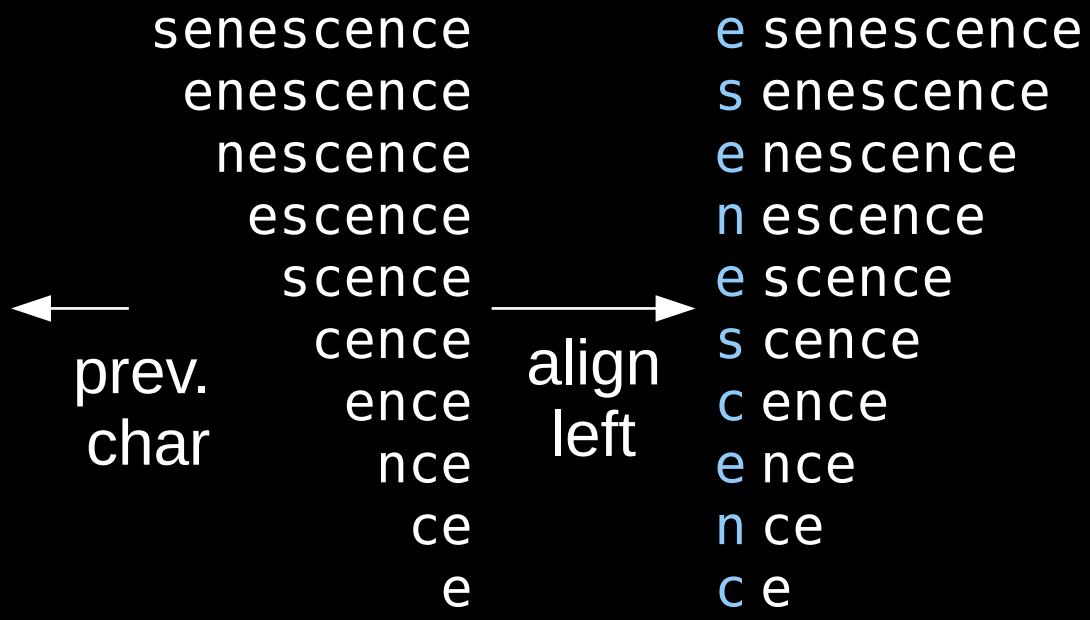
BWT of senescence

senescence

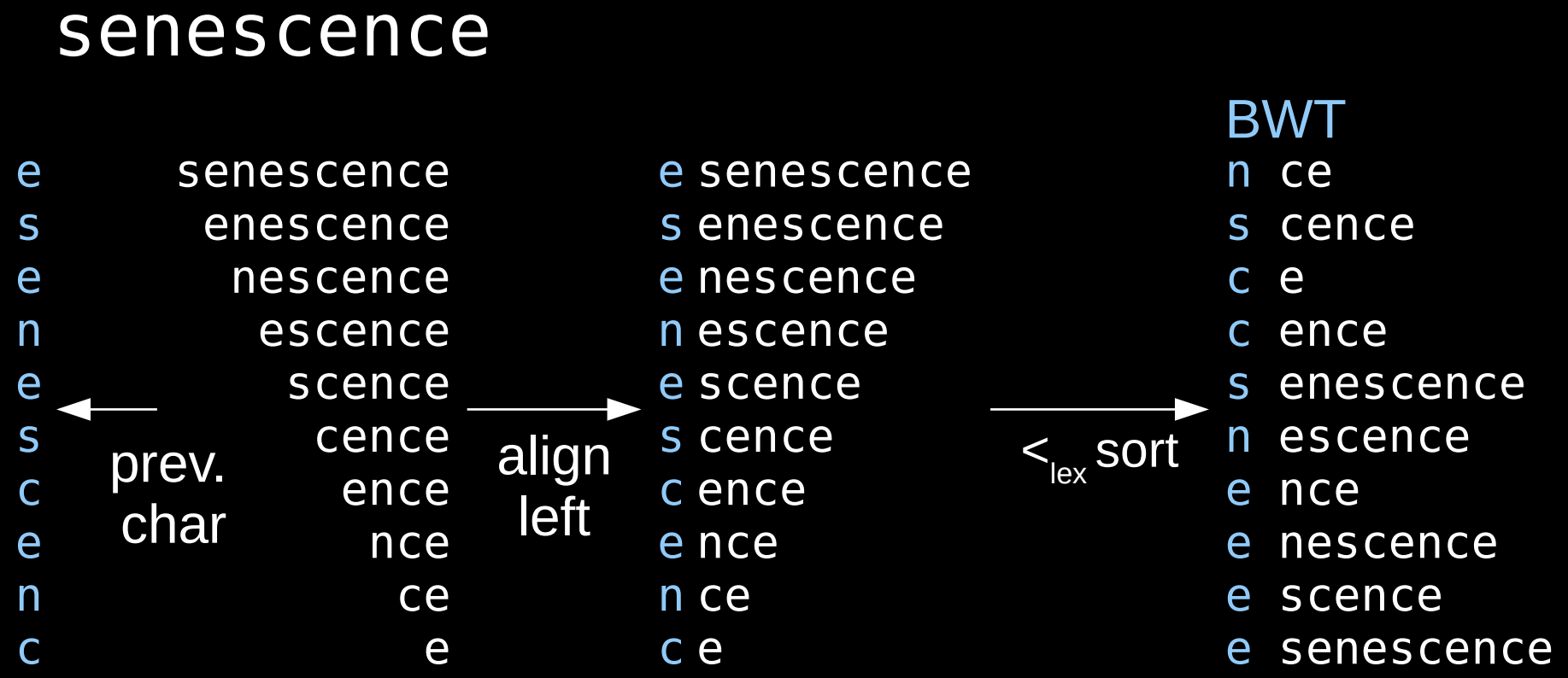


BWT of senescence

senescence

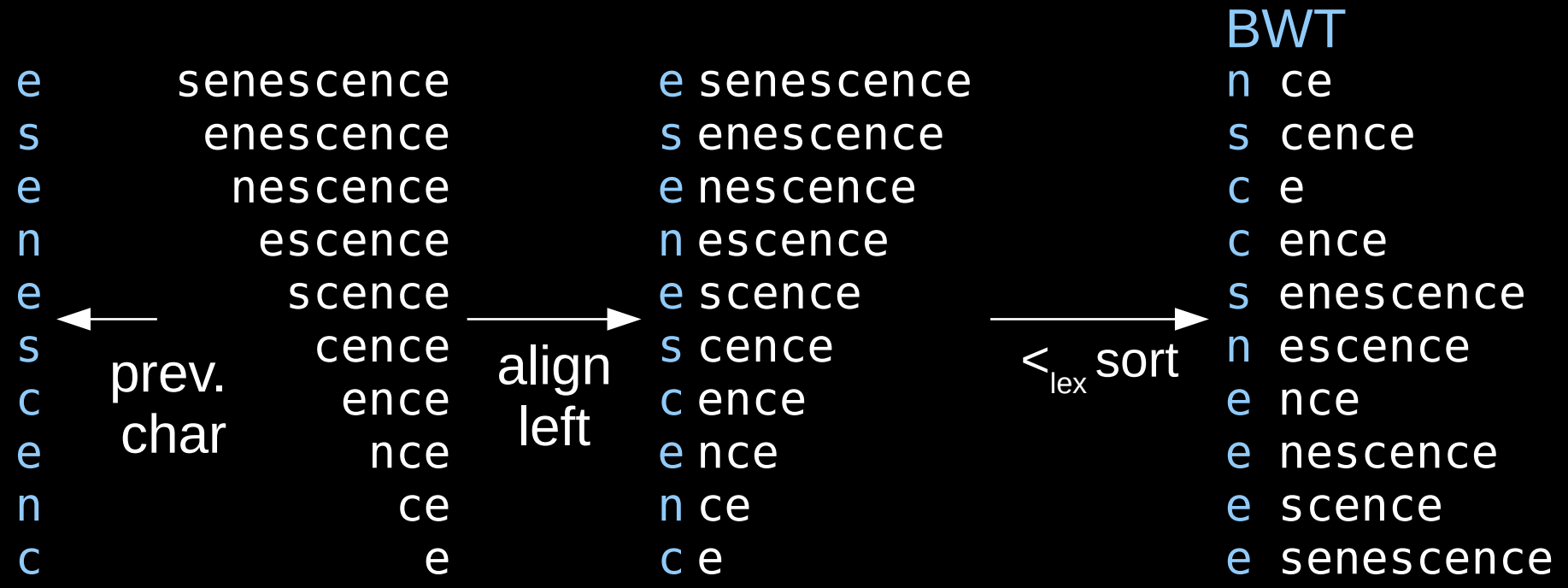


BWT of senescence



BWT of senescence

senescence



output: nsccsneeee (BBWT: enccsneees)

BBWT of senescence

s|enes|cen|ce

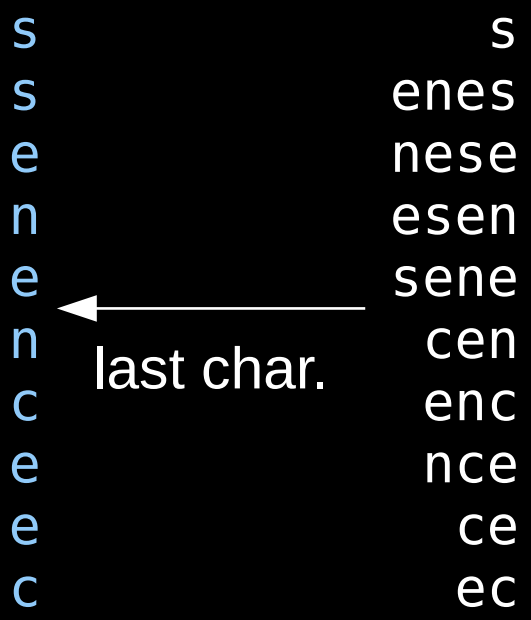
BBWT of senescence

s|enes|cen|ce

```
   s
enes
nese
esen
sene
 cen
 enc
 nce
  ce
  ec
```

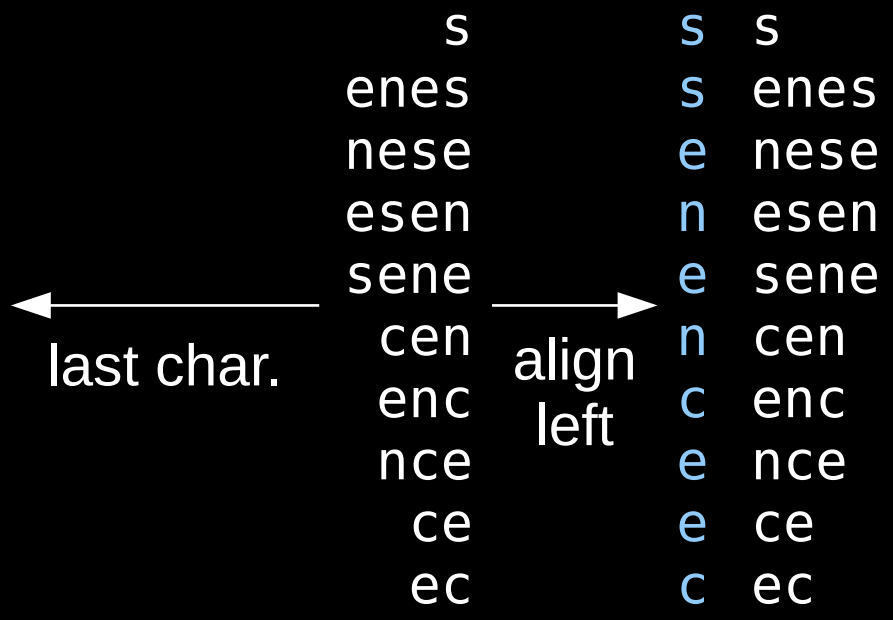
BBWT of senescence

s|enes|cen|ce



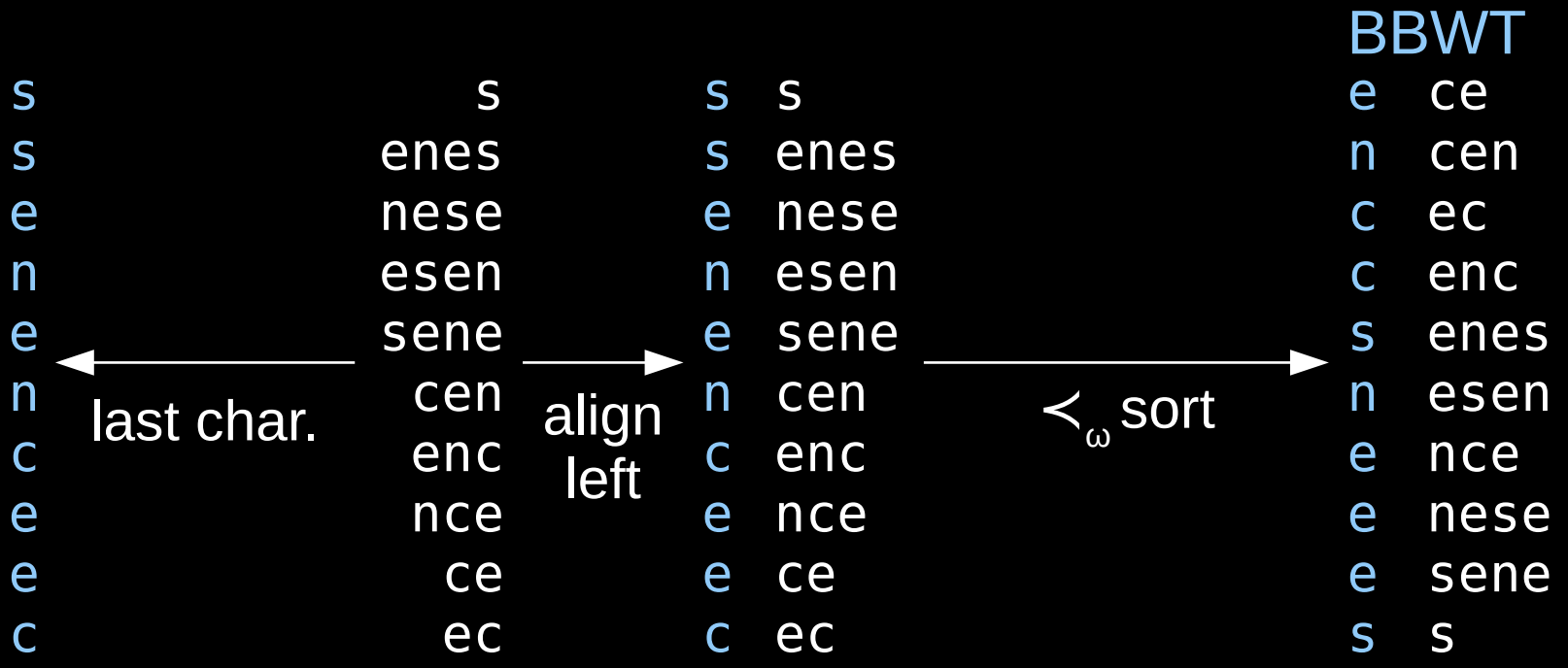
BBWT of senescence

s|enes|cen|ce



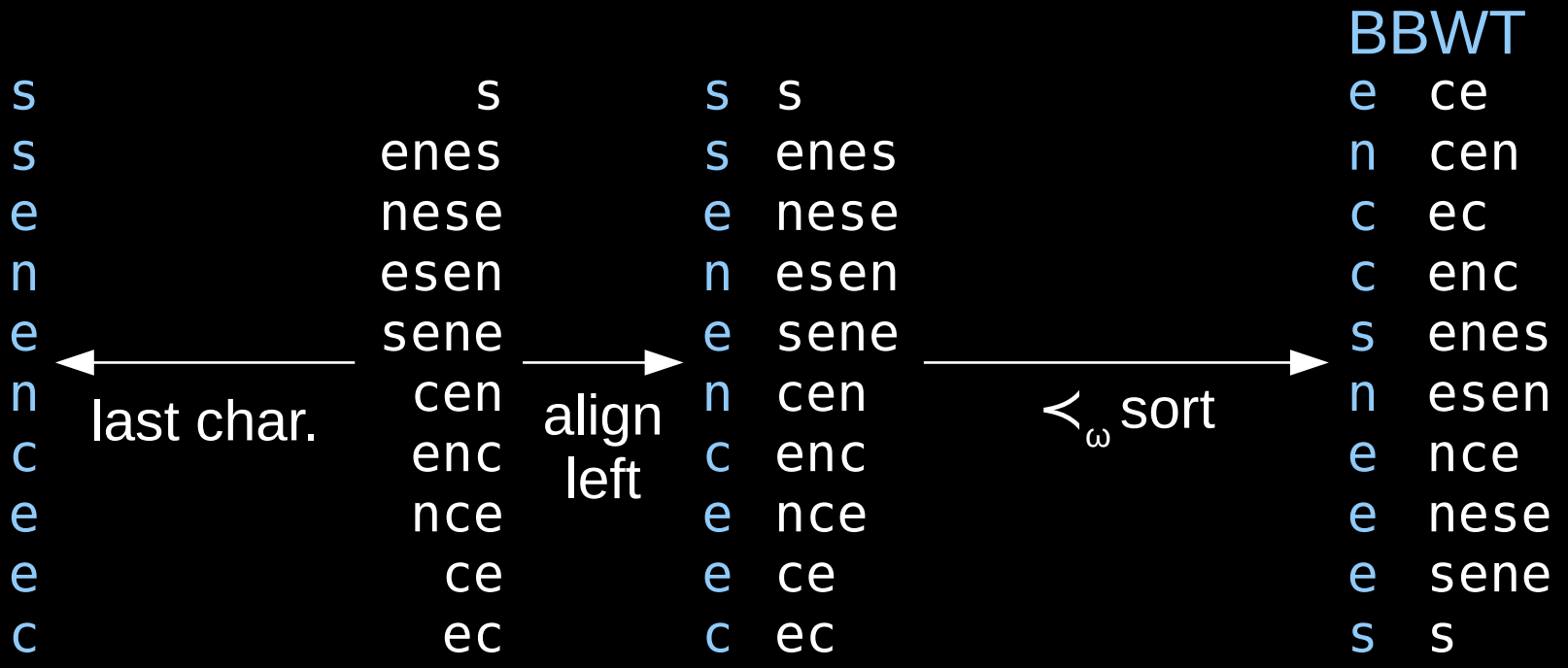
BBWT of senescence

s|enes|cen|ce



BBWT of senescence

s|enes|cen|ce



output: enccsneees

SA ⇒ BWT

- BWT[*i*] = *T* [SA[*i*]-1]
- SA[*i*] = starting position of the *i* th smallest suffix respective to lexicographic order

SAIS:

- well-known SA construction algorithm
- O(*n*) time for integer alphabets

[Nong+ ’11]

LSA ⇒ BBWT

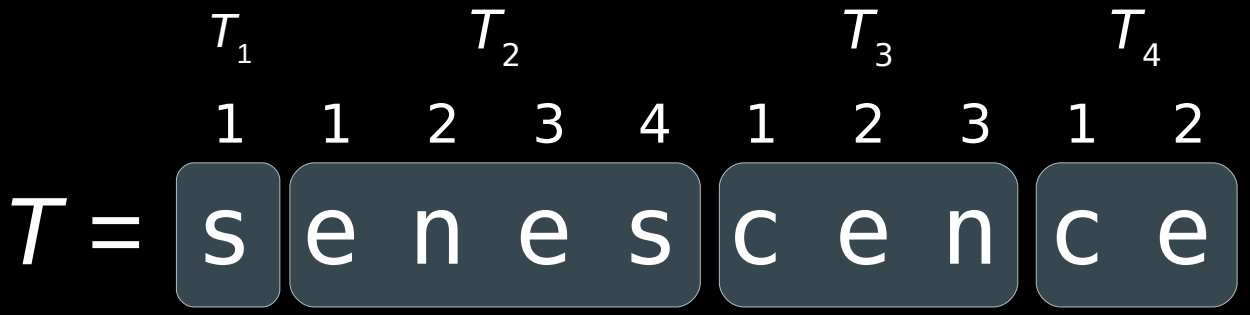
- BBWT[*i*] = *T* [LSA[*i*]-1]
- LSA[*i*] = starting position of *i* th smallest conjucate respective to $\prec_\omega$ order

[Hon+ '12]

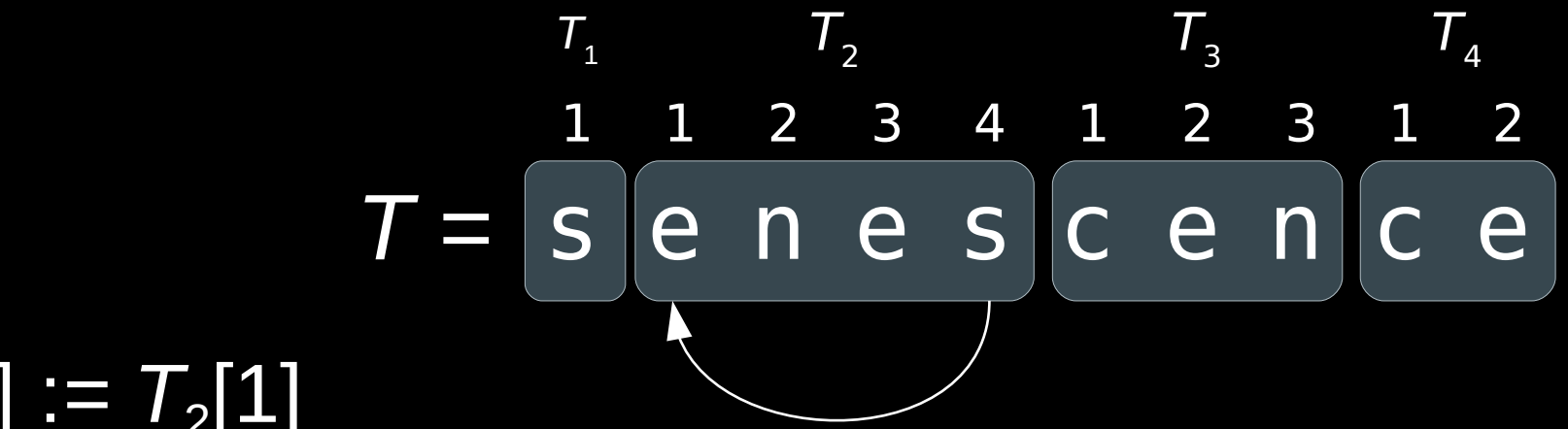
LSAIS: LSA construction algorithm

- sorts conjugates instead of suffixes
- uses $\prec_\omega$ order instead of lex. order

rewinding

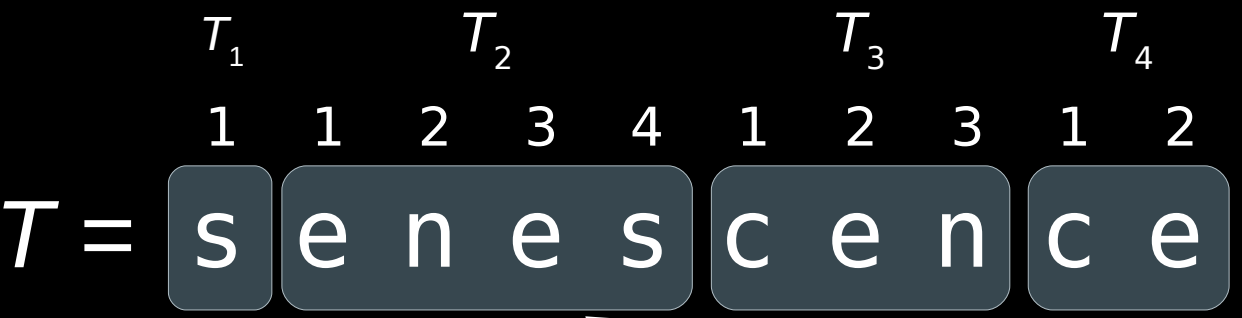


rewinding



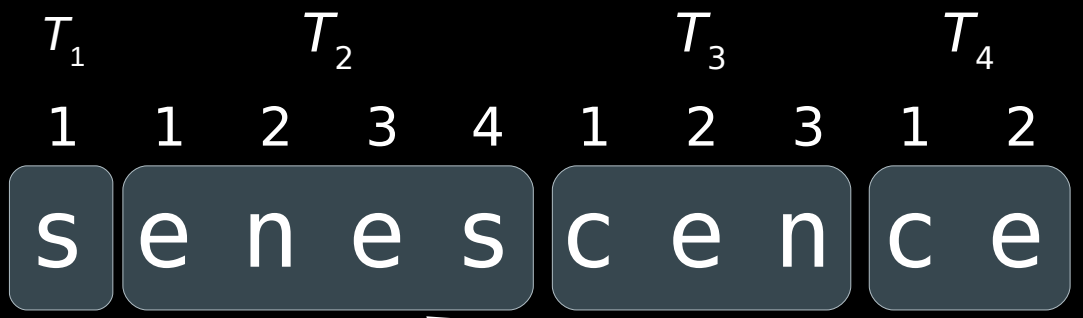
- $T_2[5] := T_2[1]$

rewinding



- $T_2[5] := T_2[1]$
- $T_2[4..] := T_2[4]T_2[1]T_2[2]T_2[3]T_2[4]T_2[1]...$

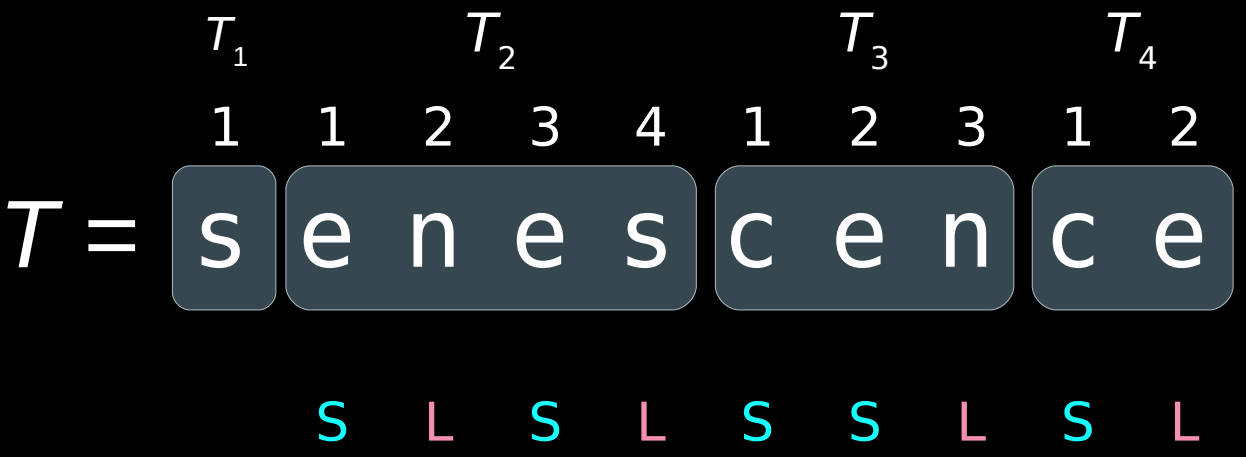
rewinding



- $T_2[5] := T_2[1]$
- $T_2[4..] := T_2[4]T_2[1]T_2[2]T_2[3]T_2[4]T_2[1]...$

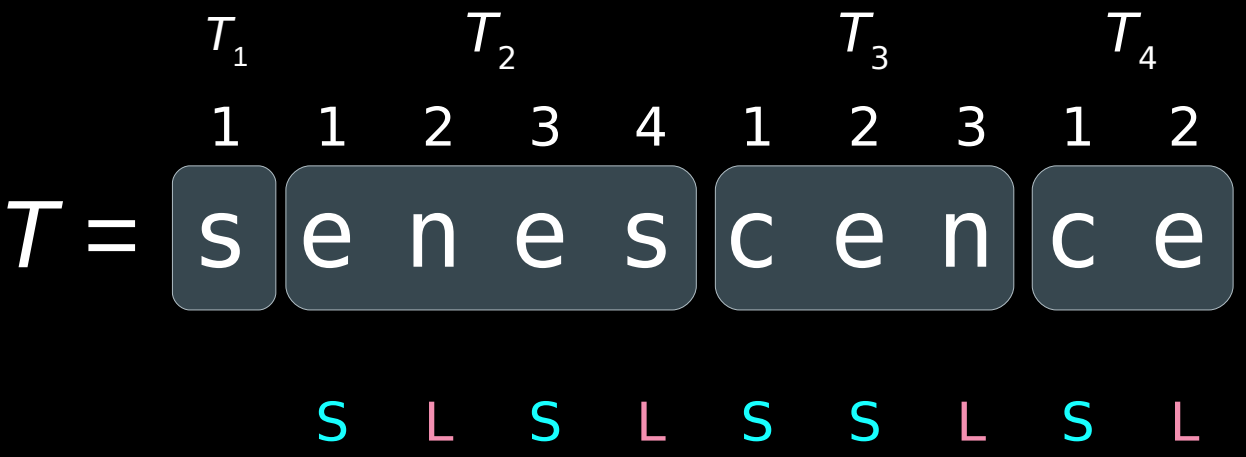
  in general, for any x define:
  - $T_x[|T_x|+j\ ] := T_x[(|T_x|+j\text{-}1) \bmod |T_x|+1]\ ,\ \ j \geq 0$
  - $T_x[0] := T_x[|T_x|]$
  - $T_x[\ i..] := T_x[\ i\ ]\ ...\ T_x[|T_x|]\ T_x[1]\ ...$

L / S type



- $T_x[\,i\,] <_{lex} T_x[\,i{+}1] \Rightarrow T_x[\,i\,]$ is S type
- $T_x[\,i\,] >_{lex} T_x[\,i{+}1] \Rightarrow T_x[\,i\,]$ is L type
- $T_x[\,i\,] = T_x[\,i{+}1] \Rightarrow T_x[\,i\,]$ and $T_x[\,i{+}1]$ are same type

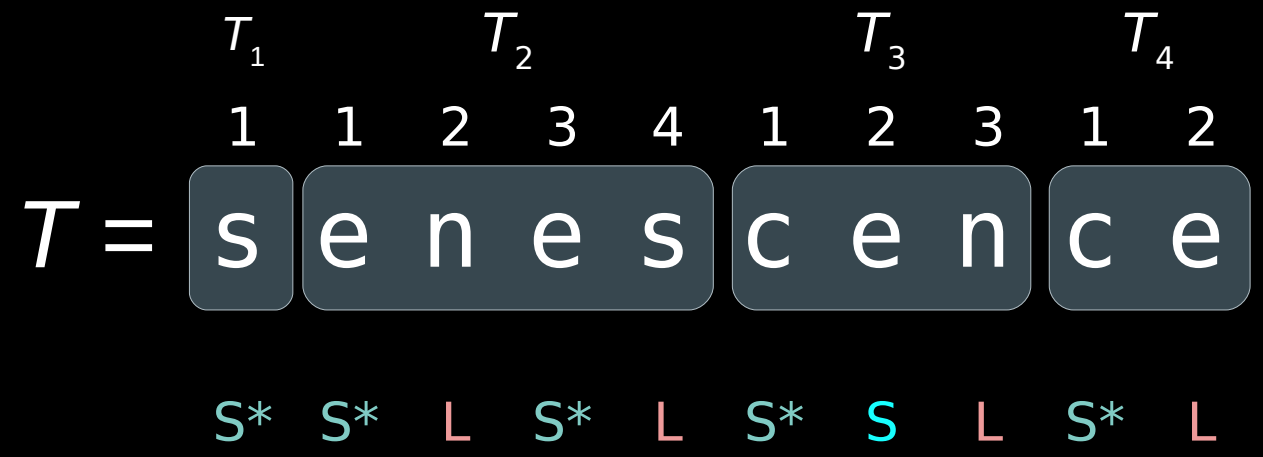
L / S type



- $T_x[\,i\,] <_{lex} T_x[\,i{+}1] \Rightarrow T_x[\,i\,]$ is S type
- $T_x[\,i\,] >_{lex} T_x[\,i{+}1] \Rightarrow T_x[\,i\,]$ is L type
- $T_x[\,i\,] = T_x[\,i{+}1] \Rightarrow T_x[\,i\,]$ and $T_x[\,i{+}1]$ are same type

thanks to the Lyndon factorization the S / L types are the same as in the original SAIS

S* type



- if $T_x[\ i\ ]$ is S type and $T_x[\ i\text{-}1\ ]$ is L type
  ⇒ $T_x[\ i\ ]$ is S* type
- $T_x[1]$ is always S* type ∀ x

LMS (substrings)

	1	2	3	4	5	6	7	8	9	10
T =	s	e	n	e	s	c	e	n	c	e
	S*	S*	L	S*	L	S*	S	L	S*	L

for $1 \le i < j \le |T_x|+1$:

$T_x[\, i..j \,]$ is called LMS if

- $T_x[\, i \,]$ and $T_x[\, j \,]$ are S* type and
- $T_x[\, k \,]$ is not S* type $\forall\, k \in [i+1 \,..\, j-1]$

start pos.	LMS
1	ss
2	ene
4	ese
6	cenc
9	cec

LSAIS algorithm

1) sort all LMS in $<_{lex}$
2) place S* types in LSA
3) induce L types
4) induce S types

sort all LMS

start pos.	LMS
1	ss
2	ene
4	ese
6	cenc
9	cec

sort all LMS

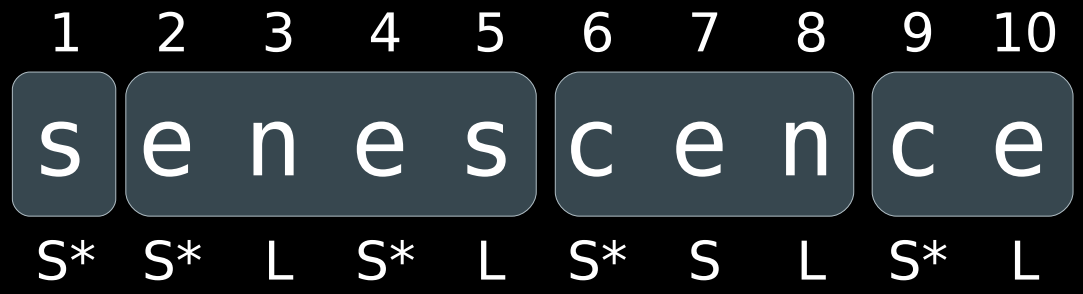
start pos.	LMS
1	ss
2	ene
4	ese
6	cenc
9	cec

$<_{lex}$ sort

start pos.	LMS
9	cec
6	cenc
2	ene
4	ese
1	ss

works by recursion like SAIS,
see paper for details

S/L type bucket allocation



LSA =

S		L	S			L		L	S
c		e				n		s	

place S* types

pos.	LMS
9	cec
6	cenc
2	ene
4	ese
1	ss

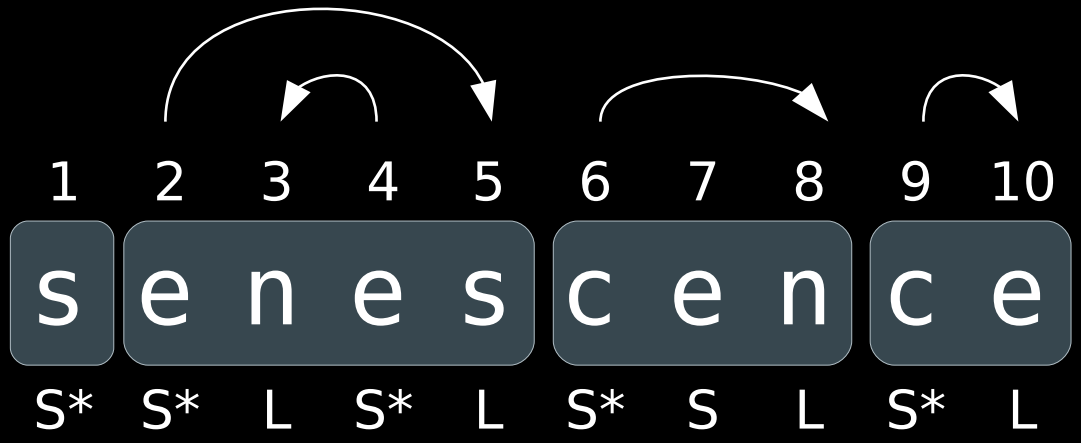
T =

1	2	3	4	5	6	7	8	9	10
s	e	n	e	s	c	e	n	c	e
S*	S*	L	S*	L	S*	S	L	S*	L

LSA =

9	6		2	4					1
S		L	S			L		L	S
c			e			n		s	

induce L types



LSA =

9	6	10	2	4		8	3	5	1
S		L	S			L		L	S
c		e				n		s	

induce S types

LSA =

9	6	10	7	2	4	8	3	5	1
S		L	S			L		L	S
c		e				n		s	

BBWT[*i*] = *T* [LSA[*i*]-1]

	1	2	3	4	5	6	7	8	9	10
T =	s	e	n	e	s	c	e	n	c	e
	S*	S*	L	S*	L	S*	S	L	S*	L

LSA =	9	6	10	7	2	4	8	3	5	1

LSA-1 =	10	8	9	6	5	3	7	2	4	1
T [*LSA*-1] =	e	n	c	c	s	n	e	e	e	s

time analysis

SAIS

- sorts LMS recursively
- given SAIS needs $\tau(n)$ time,
  $\tau(n) = O(n) + \tau(n/2) = O(n)$ since
  the number of LMS $\leq n/2$

LSAIS

- number of LMS can be $\Theta(n)$ in all recursion steps
- still $O(n^2)$ time? 😢

example

T = b...baababaabab....aabab

b|...|b|aabab|aabab|...|aabab

example

T = b...baababaabab....aabab

b|...|b|aabab|aabab|...|aabab

LMS:

- bb → 3
- aaba → 1
- aba → 2

assign ranks according to lexicographic order

example

T = b...baababaabab....aabab

b|...|b|aabab|aabab|...|aabab

LMS:

- bb → 3
- aaba → 1
- aba → 2

assign ranks according to lexicographic order

3|....|3|12|12|...|12 (recursion step)

example

T = b...baababaabab....aabab

b|...|b|aabab|aabab|...|aabab

LMS:

- bb → 3
- aaba → 1
- aba → 2

assign ranks according to lexicographic order

same amount

3|....|3|12|12|...|12 (recursion step)

example

T = b...baababaabab....aabab

b|...|b|aabab|aabab|...|aabab

LMS:

- bb → 3
- aaba → 1
- aba → 2

assign ranks according to lexicographic order

same amount

3|....|3|12|12|...|12 (recursion step)

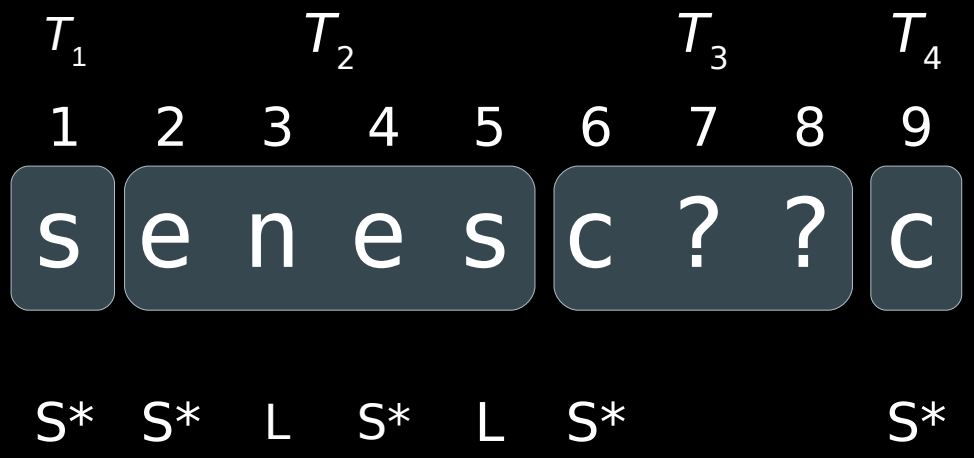
same Lyndon factorization

observation

- the string on the ranked LMS subtrings has the same Lyndon factorization
- compared to any character *c*, all LMS starting with *c* are larger with respect to $\prec_\omega$

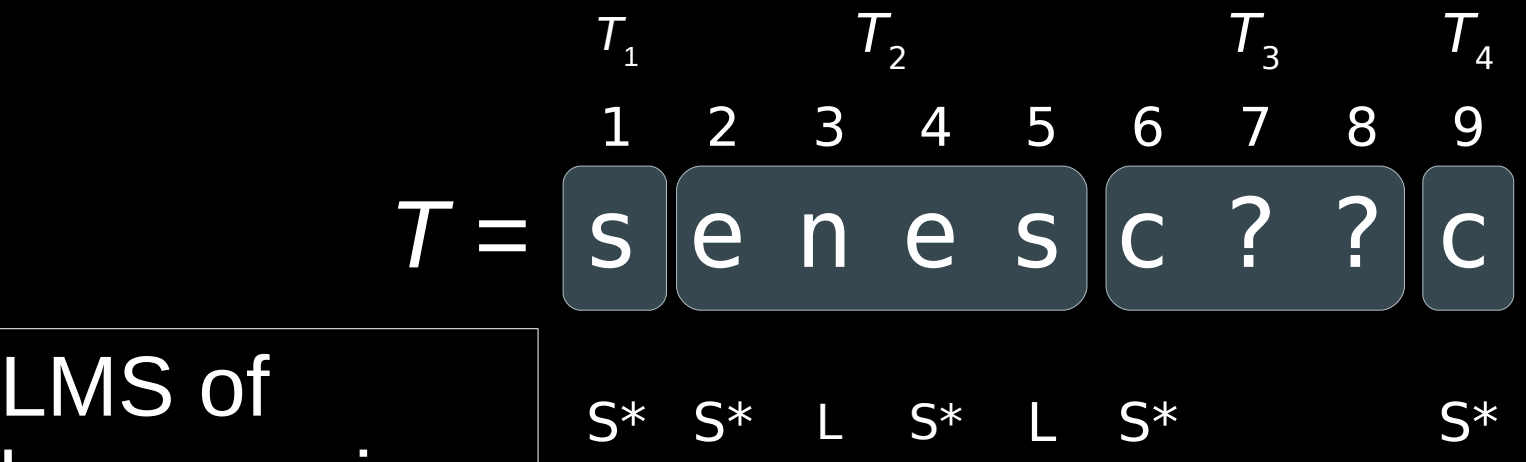
observation

- the string on the ranked LMS subtrings has the same Lyndon factorization
- compared to any character *c*, all LMS starting with *c* are larger with respect to $\prec_\omega$



observation

- the string on the ranked LMS subtrings has the same Lyndon factorization
- compared to any character *c*, all LMS starting with *c* are larger with respect to $\prec_\omega$



⇒ can omit LMS of length 1 in the recursion

summary

- bijective BWT construction
  - O(*n*) time on integer alphabets
- methods
  - adapt SAIS
  - sort conjugates in $\prec_\omega$ instead of suffixes in lexicographic order
  - skip LMS of length 1 in recursion

summary

- bijective BWT construction
  - O(*n*) time on integer alphabets
- methods
  - adapt SAIS
  - sort conjugates in $\prec_\omega$ instead of suffixes in lexicographic order
  - skip LMS of length 1 in recursion

all questions are welcome!