In-Place (Bijective) BWT Transforms

Kyushu University — *Dominik Köppl*

Tohoku University — Daiki Hashimoto, Diptarama, Ayumi Shinohara

data structures

Burrows-Wheeler Transform (BWT)

[Burrows,Wheeler '94]

Bijective BWT (BBWT)

[Gil,Scott '12]

BWT of bacabbabb

T = `bacabbabb$`

BWT of bacabbabb

T = bacabbabb$

all suffixes

```
bacabbabb$
 acabbabb$
  cabbabb$
   abbabb$
    bbabb$
     babb$
      abb$
       bb$
        b$
         $
```

BWT of bacabbabb

T = bacabbabb$

all suffixes

```
$      bacabbabb$
b       acabbabb$
a        cabbabb$
c         abbabb$
a          bbabb$
b           babb$
b            abb$
a             bb$
b              b$
b               $
```

prev. char

BWT of bacabbabb

T = bacabbabb$

all suffixes

prev. char	suffix	align left
$	bacabbabb$	$ bacabbabb$
b	acabbabb$	b acabbabb$
a	cabbabb$	a cabbabb$
c	abbabb$	c abbabb$
a	bbabb$	a bbabb$
b	babb$	b babb$
b	abb$	b abb$
a	bb$	a bb$
b	b$	b b$
b	$	b $

BWT of bacabbabb

T = bacabbabb$

all suffixes

prev. char	suffix
$	bacabbabb$
b	acabbabb$
a	cabbabb$
c	abbabb$
a	bbabb$
b	babb$
b	abb$
a	bb$
b	b$
b	$

align left

$	bacabbabb$
b	acabbabb$
a	cabbabb$
c	abbabb$
a	bbabb$
b	babb$
b	abb$
a	bb$
b	b$
b	$

$<_{lex}$ sort

lex. order

BWT	
b	$
b	abb$
c	abbabb$
b	acabbabb$
b	babb$
b	b$
$	bacabbabb$
a	bb$
a	bbabb$
a	cabbabb$

the BBWT is

the BWT of

the Lyndon factorization

of an input text

with respect to $\prec_\omega$

the 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 =

T_1	T_2	$\cdots$	T_t

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

(Chen-Fox-Lyndon Theorem)

example

T = bacabbabb

Lyndon factorization: b|ac|abb|abb

- b, ac, abb, and abb are Lyndon
- b $>_{lex}$ ac $>_{lex}$ abb $\geq_{lex}$ abb

$\prec_\omega$ order

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

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

$\prec_\omega$ order

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

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

ababab···
abaabaaba···

BBWT of bacabbabb

b|ac|abb|abb

BBWT of bacabbabb

b|ac|abb|abb

```
b  ac   abb   abb
   ca   bab   bab
        bba   bba
```

BBWT of bacabbabb

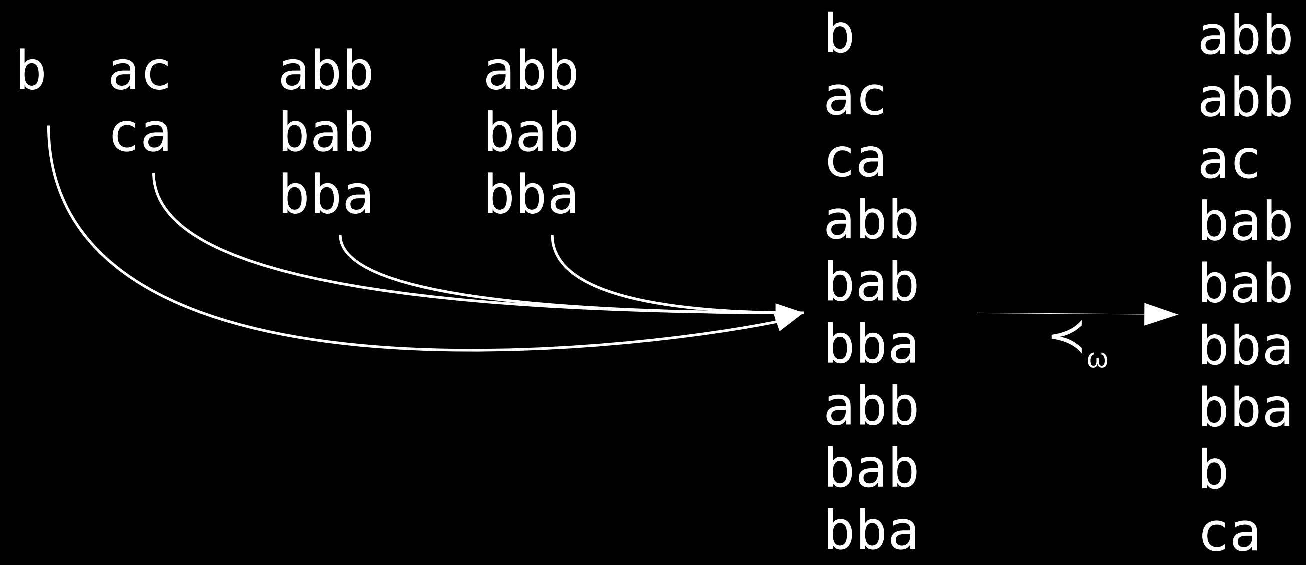
b|ac|abb|abb

```
b   ac   abb   abb
    ca   bab   bab
         bba   bba
```

```
b
ac
ca
abb
bab
bba
abb
bab
bba
```

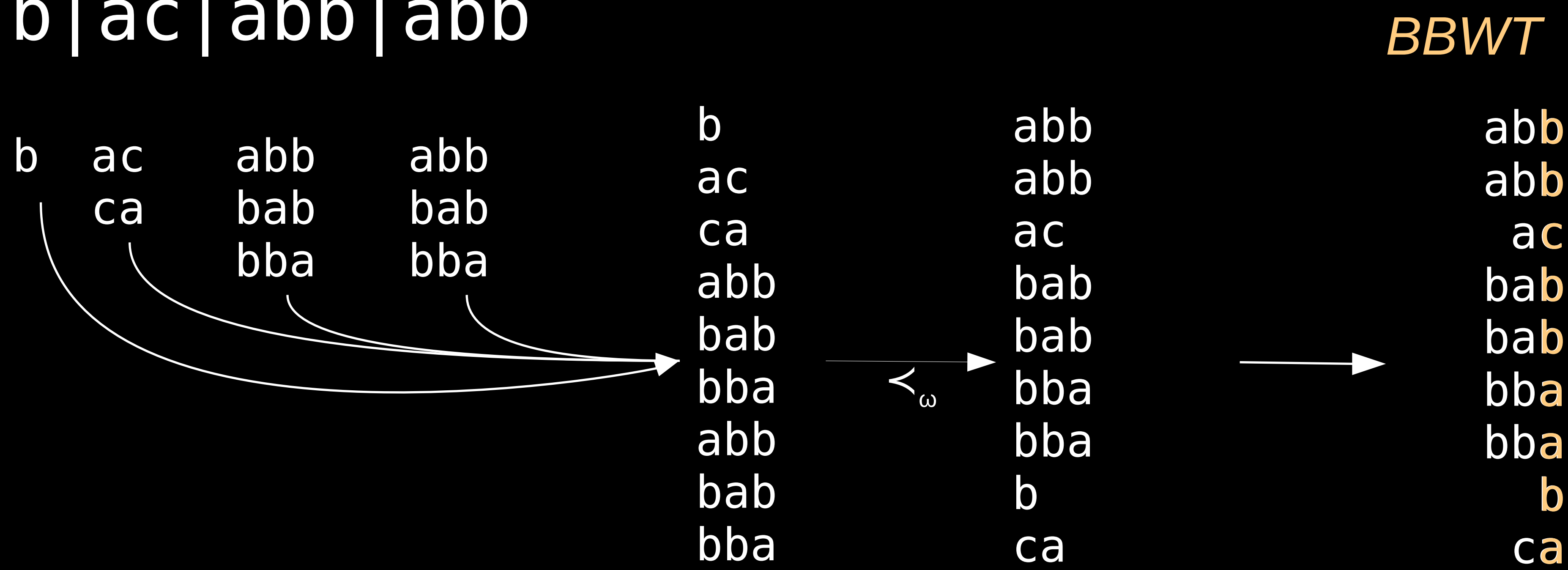
BBWT of bacabbabb

b|ac|abb|abb



BBWT of bacabbbabb

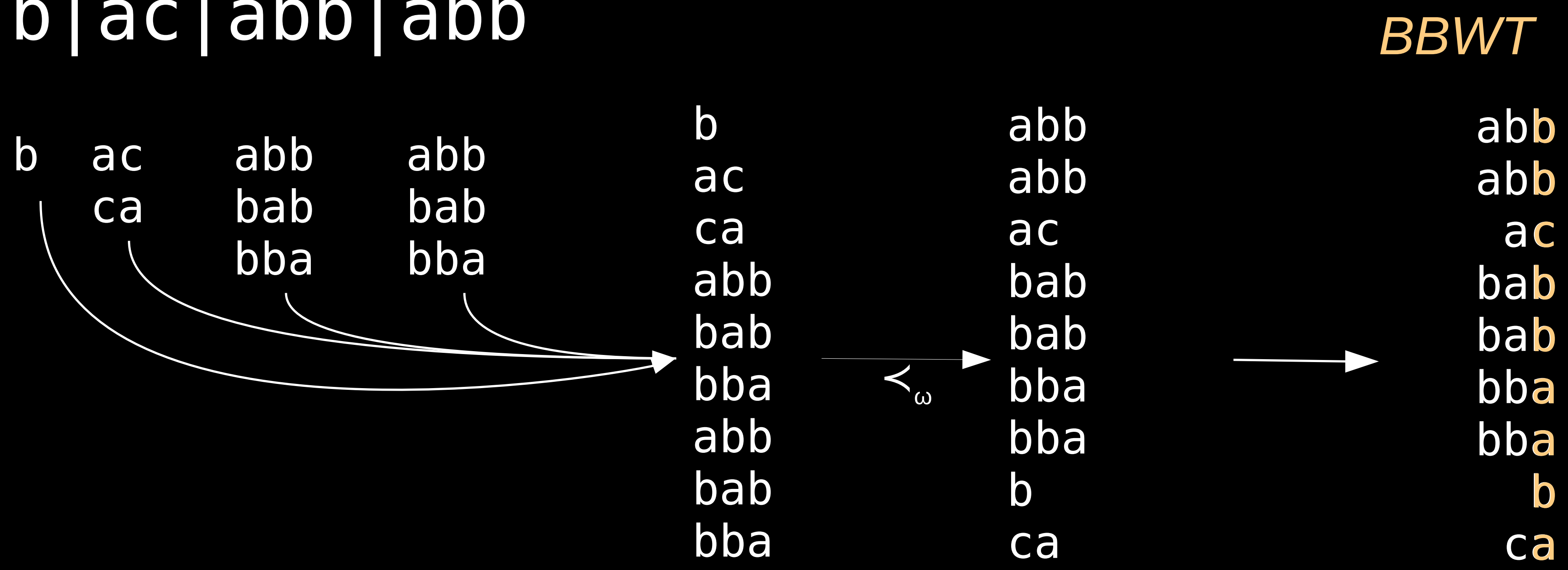
b|ac|abb|abb



BBWT(T) = bbcbbaaba

BBWT of bacabbbabb

b|ac|abb|abb



BBWT(T) = bbcbbaaba

BWT(T\$) = bbcbbb\$aaa

motivation

properties of BBWT :

- no $ necessary
- BBWT is more compressible than BWT for various inputs

[Scott and Gill '12]

- BBWT is indexible (full text index)
- is computable in O(n) time with O(n) words

[Bannai+ '19]

however, O(n) words can be too much for large n

in-place computation

- Σ: alphabet, σ := |Σ| alphabet size
- *T* : text, *n* := |*T*|
- *L* := *n* lg σ bits workspace
- aim : in-place computation

  transform *T* ↔ BWT ↔ BBWT with

  |*L*| + O(lg *n*) bits of workspace

L

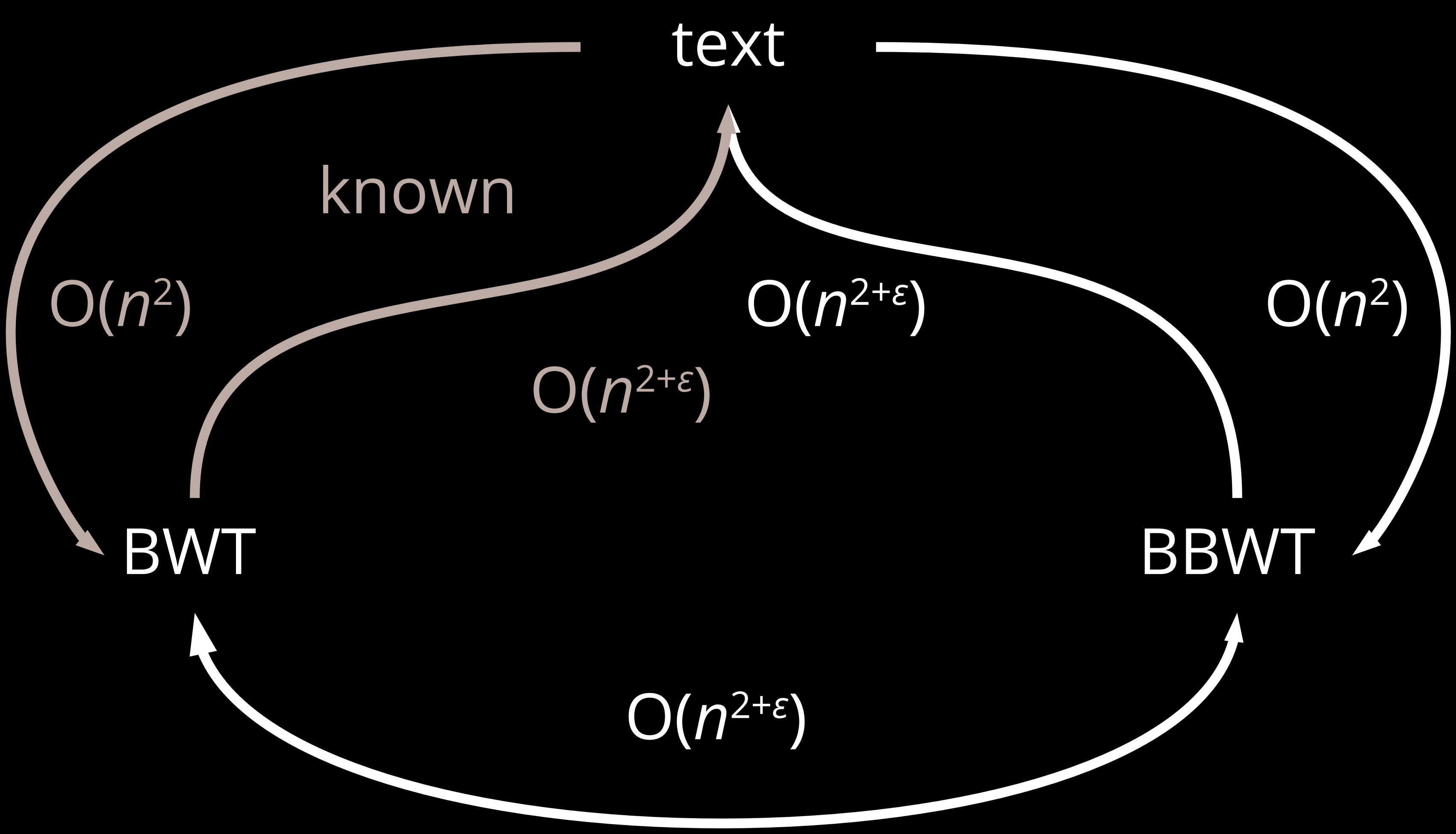
T := b a c a b b a b b

known solutions

input	output	work-space	time	reference
text	BWT	in-place	O(n^2)	Crochemore+ '15
BWT	text	in-place	O($n^{2+ε}$)	
text	BBWT	O(n lg $σ$) bits	O(n lg n/lg lg n)	Bonomo+ '14

σ : alphabet size, n : text length,
ε is a constant with 0 < ε < 1

in-place conversions



working space: $n \lg \sigma + O(\lg n)$ bits (including text)

forward search

T = bacabbabb$

F	*L*
$	b
a	b
a	c
a	b
b	b
b	b
b	$
b	a
b	a
c	a

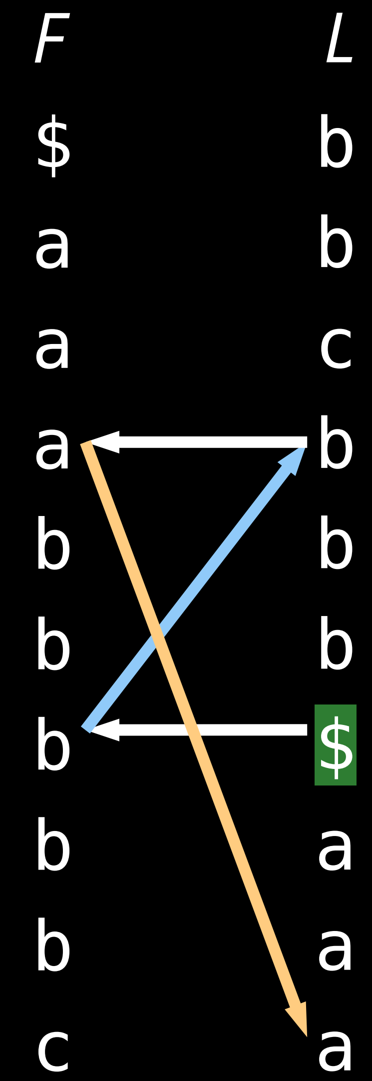
forward search

T = bacabbabb$

F	*L*
$	b
a	b
a	c
a	b
b	b
b	b
b	$
b	a
b	a
c	a

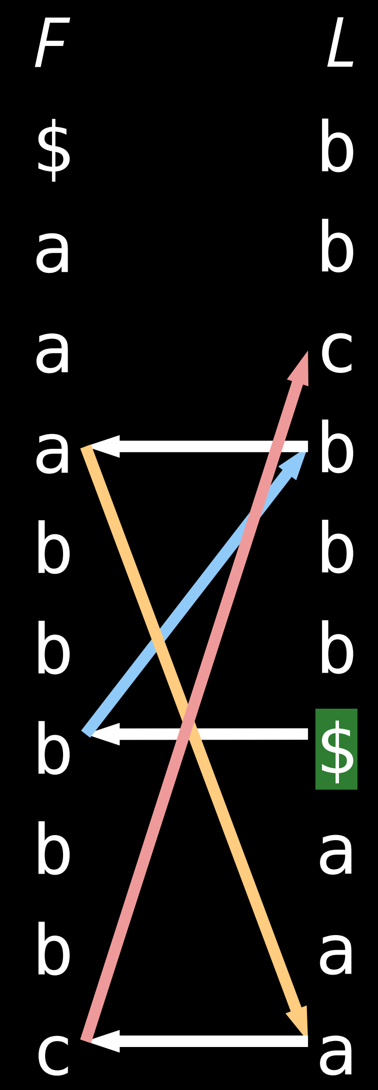
forward search

T = bacabbabb$



forward search

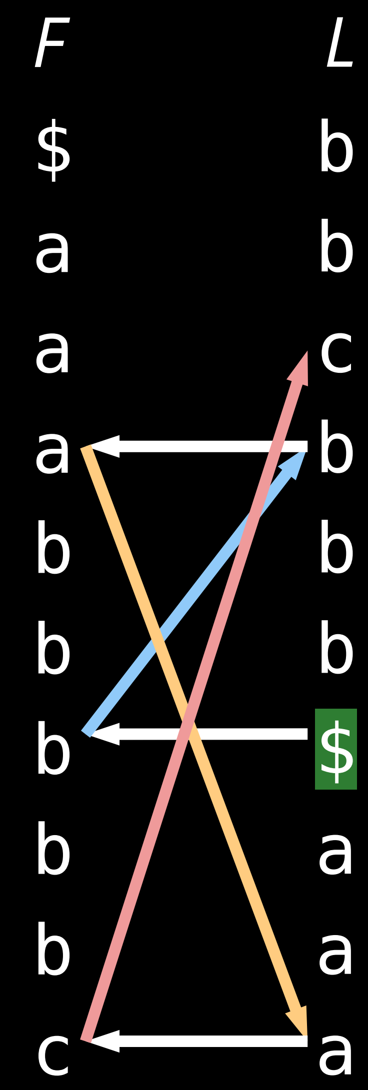
T = bacabbabb$



forward search

T = bacabbabb$

can calculate with
rank and select on *F* and *L*

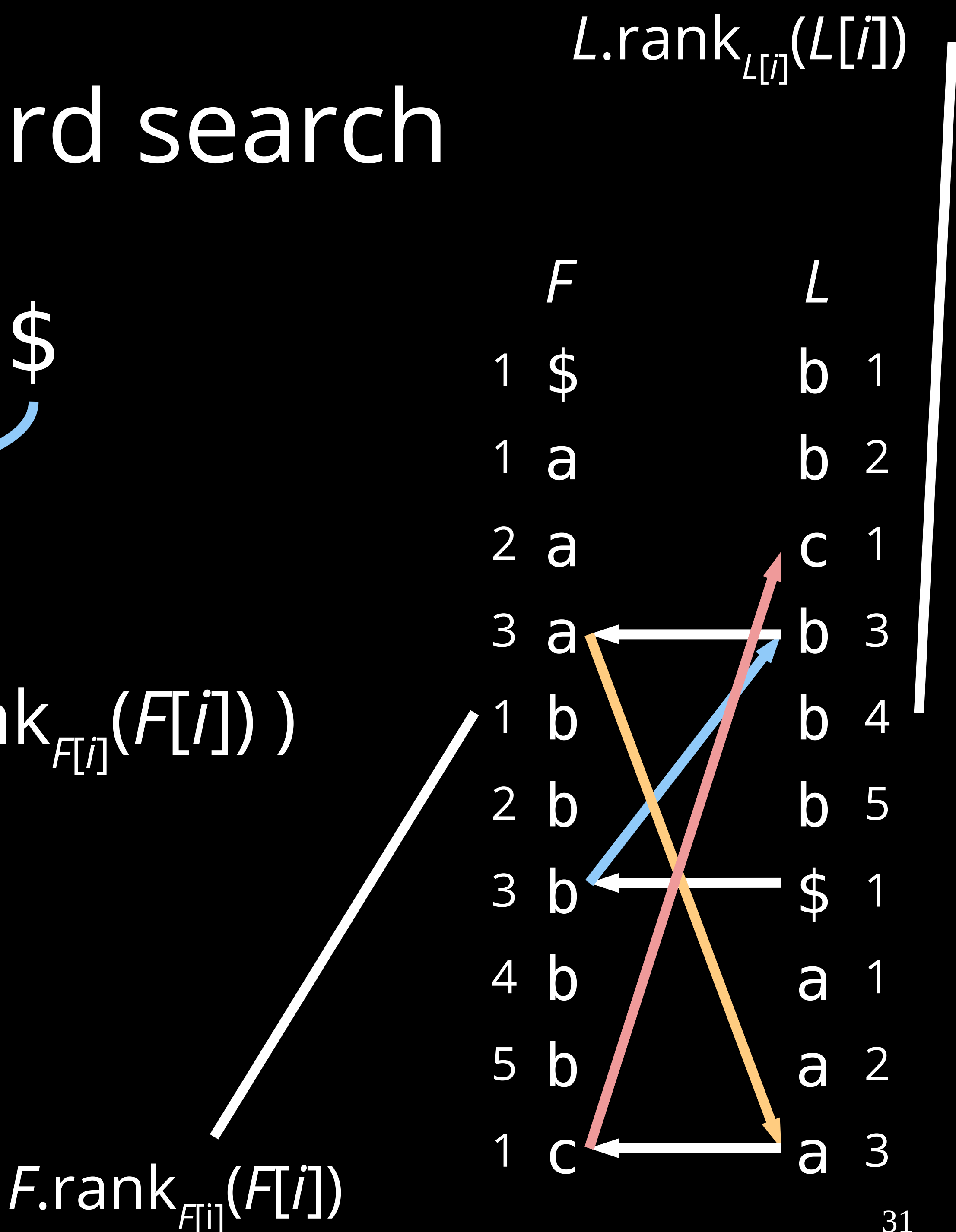


forward search

T = bacabbabb$

FL mapping:

$FL(i) = L.\text{select}_{F[i]}(\ F.\text{rank}_{F[i]}(F[i])\ )$



backward search

T = bacabbabb$

	F	L	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	b	3
1	b	b	4
2	b	b	5
3	b	$	1
4	b	a	1
5	b	a	2
1	c	a	3

$L.\text{rank}_{L[i]}(L[i])$

$F.\text{rank}_{F[i]}(F[i])$

FM index [Ferragina, Manzini '00]

backward search

$L.\text{rank}_{L[i]}(L[i])$

T = bacabbabb$

	F	L	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	b	3
1	b	b	4
2	b	b	5
3	b	$	1
4	b	a	1
5	b	a	2
1	c	a	3

$F.\text{rank}_{F[i]}(F[i])$

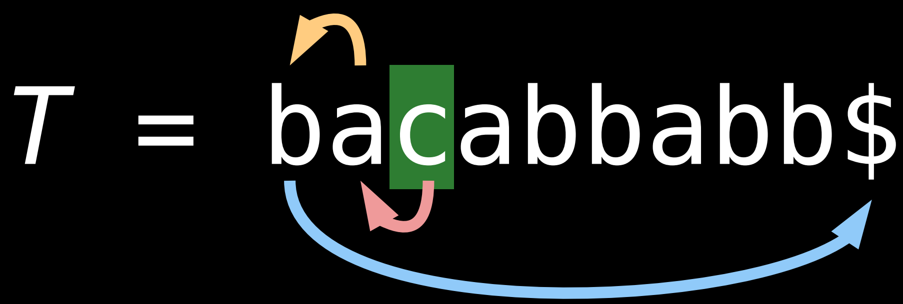
FM index [Ferragina, Manzini '00]

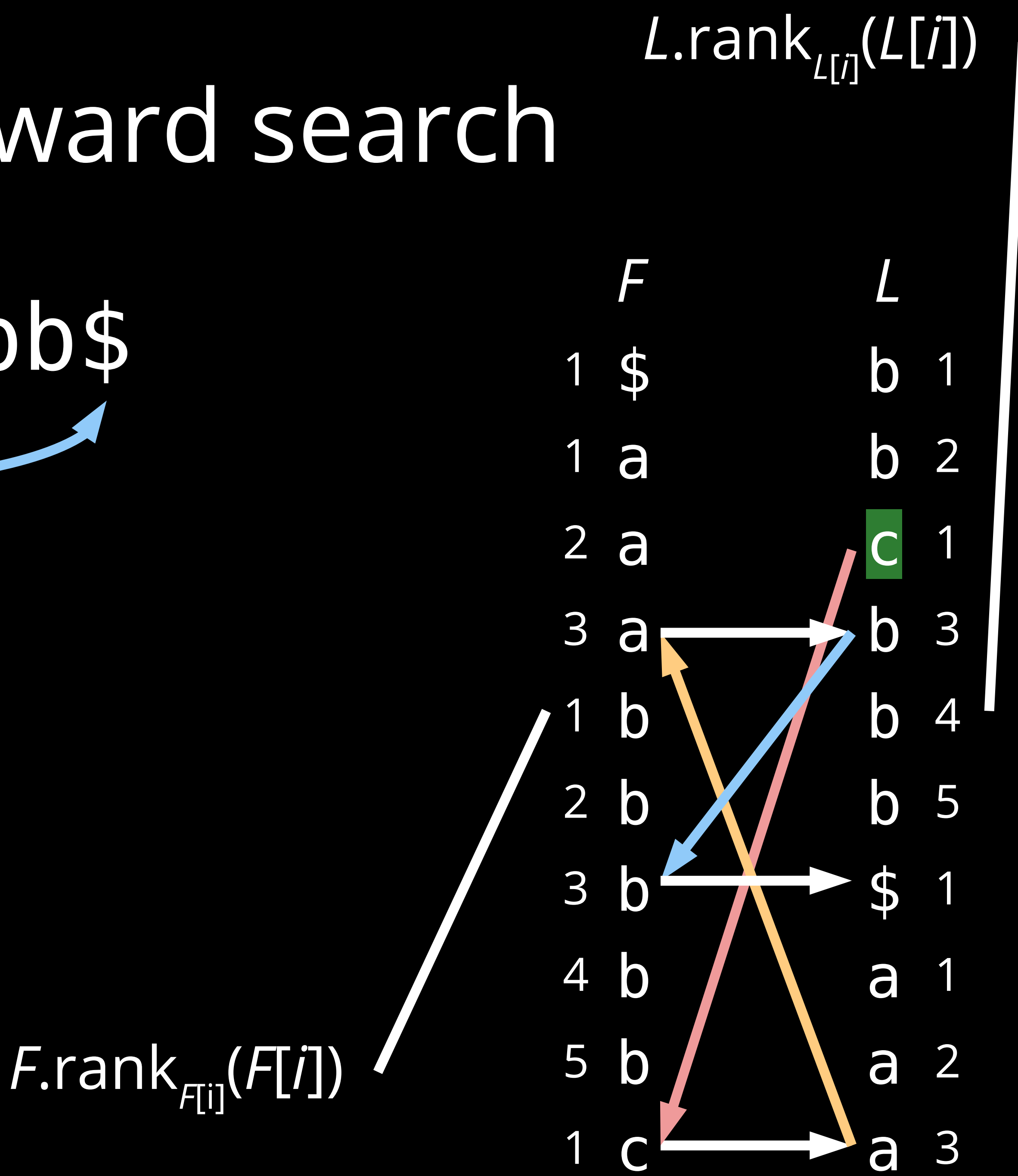
backward search

T = bacabbabb$

FM index [Ferragina, Manzini '00]

backward search





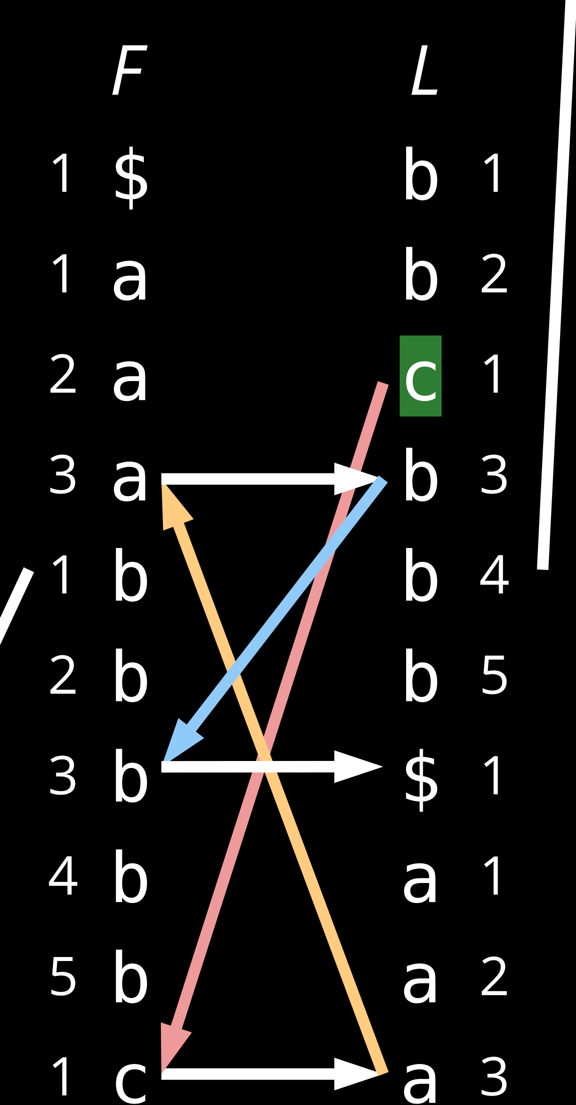
FM index [Ferragina, Manzini '00]

backward search

T = bacabbabb$

LF mapping:

$\mathrm{LF}(i) := F.\mathrm{select}_{L[i]}(\ L.\mathrm{rank}_{L[i]}(i)\ )$



FM index [Ferragina, Manzini '00]

$L.\mathrm{rank}_{L[i]}(L[i])$

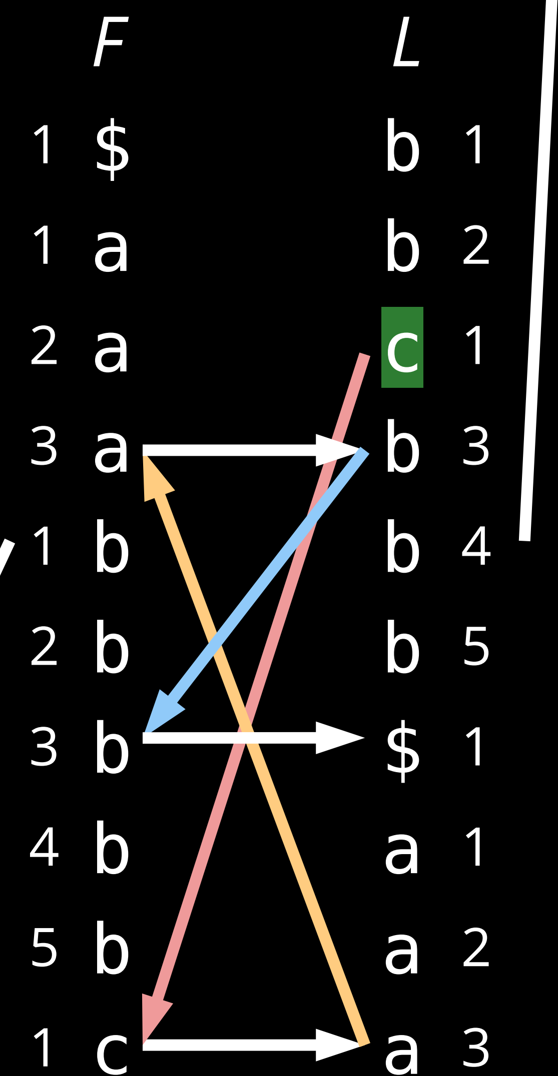
backward search

T = bacabbbabb$

LF mapping:

LF(i) := $F.\mathrm{select}_{L[i]}(\ L.\mathrm{rank}_{L[i]}(i)\ )$

= $F.\mathrm{select}_{L[i]}(1) + L.\mathrm{rank}_{L[i]}(i)-1$



$F.\mathrm{rank}_{F[i]}(F[i])$

FM index [Ferragina, Manzini '00]

$L.\text{rank}_{L[i]}(L[i])$

backward search

T = bacabbbabb$

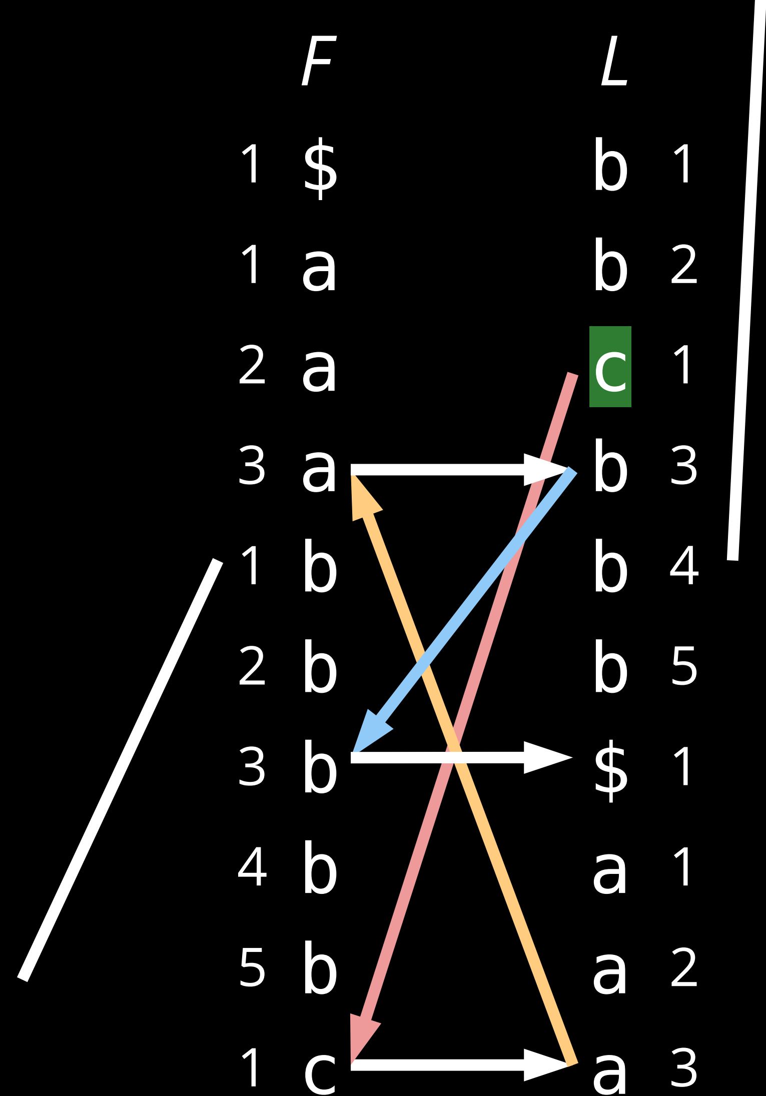
LF mapping:

$\text{LF}(i) := F.\text{select}_{L[i]}(\ L.\text{rank}_{L[i]}(i)\ )$

$= F.\text{select}_{L[i]}(1) + L.\text{rank}_{L[i]}(i)\text{-}1$

$= |\{\ j : L[j] < L[i]\}| + L.\text{rank}_{L[i]}(i)$

$F.\text{rank}_{F[i]}(F[i])$



FM index [Ferragina, Manzini '00]

LF: time complexity

If we store BWT(T) in L :

– $L[i]$ = BWT[i]: O(1) time

  ⇒ for any c : $L.\mathrm{rank}_c(i)$ in O(n) time

– $\mathrm{LF}(i) = |\{ j : L[j] < L[i]\}| + L.\mathrm{rank}_{L[i]}(i)$

$|\{ j : L[j] < L[i]\}|$: O(n) time

$L.\mathrm{rank}_{L[i]}(i)$: O($n$) time

FL: time complexity

- $\mathrm{FL}(i) = L.\mathrm{select}_{F[i]}(\ F.\mathrm{rank}_{F[i]}(F[i])\ )$

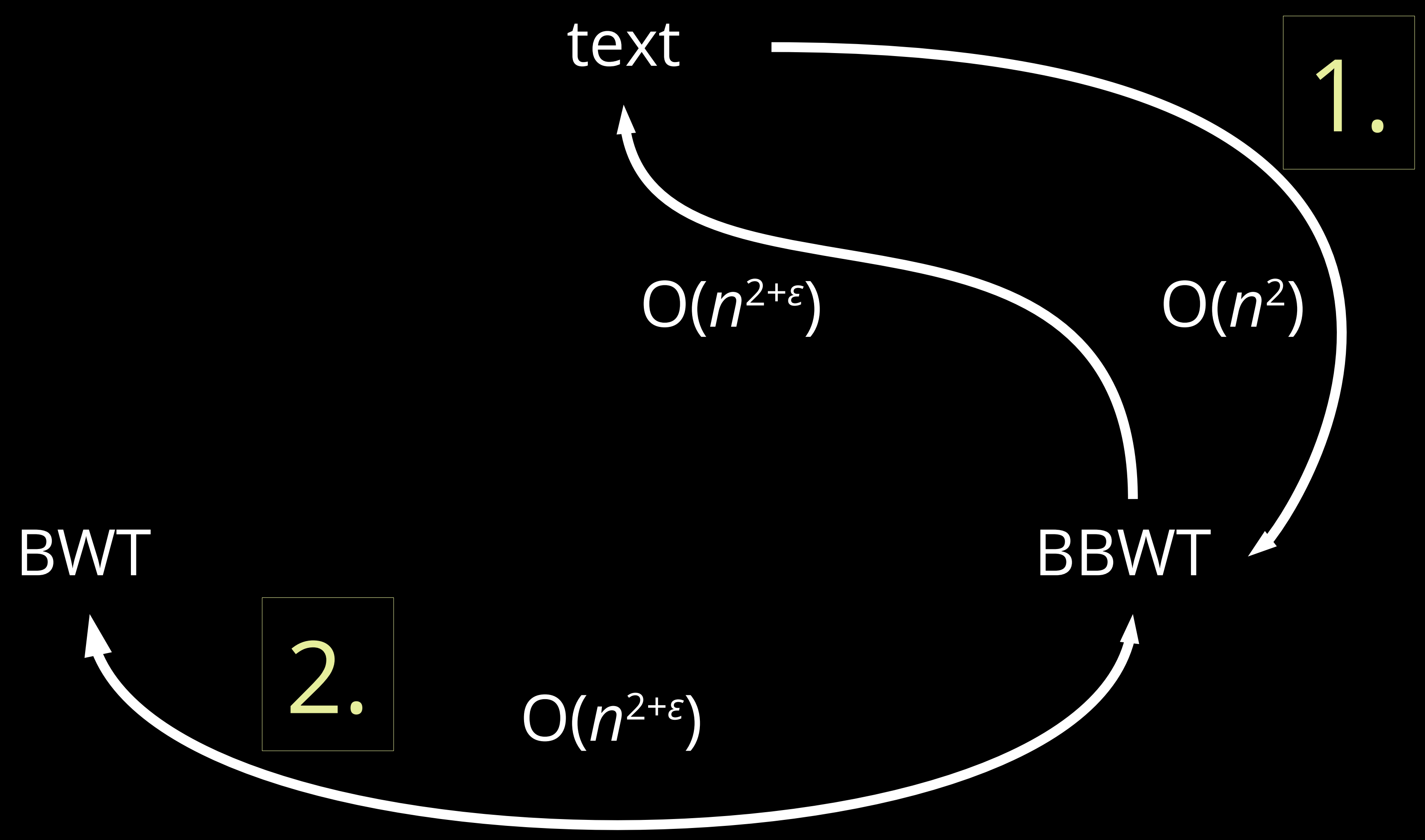
  $= L.\mathrm{select}_{F[i]}\ (\ i - |\{\ j : L[j] < i\}|\ )$

- If we know $F[i]$: $\mathrm{FL}(i)$ in $O(n)$ time
- however, the fastest in-place computation of $F[i]$ takes $O(n^{1+\varepsilon})$ time

[Munro,Raman '96]

for any constant ε with $0 < \varepsilon < 1$

road map



working space: $n \lg \sigma + O(\lg n)$ bits (including text)

text → BBWT

text → BBWT

for each Lyndon factor T_x with x = 1 up to t :

prepend $T_x[|T_x|]$ to BBWT

$p \leftarrow 1$ (insert position in BBWT)

for each $i = |T_x|$-1 down to 1 :

$p \leftarrow \mathrm{LF}(p) + 1$

insert $T_x[i]$ at BBWT[p]

[Bonomo+ '14]

text → BBWT

T = bacabbabb

- Lyndon factorization:

  b|ac|abb|abb

- first: insert b

text → BBWT

T = bacabbabb

- Lyndon factorization:

  b|ac|abb|abb

- first: insert b

	F	L	
1	b	b	1

text → BBWT

T = bacabbabb

- Lyndon factorization:
  b|ac|abb|abb
- first: insert b

	F	*L*	
1	b	b	1

how to calculate?

	F	*L*	
1	a	b	1
2	a	b	2
3	a	c	1
1	b	b	3
2	b	b	4
3	b	a	1
4	b	a	2
5	b	b	5
1	c	a	3

BBWT(T_1 T_2)

T = b | ac | abb | abb = T_1 T_2 T_3 T_4

- next Lyndon factor: ac

	F	*L*	
1	b	b	1

BBWT(T_1 T_2)

T = b | ac | abb | abb = T_1 T_2 T_3 T_4

- next Lyndon factor: ac

	F	L	
1	b	b	1

	F	L	
1	b	c	1
1	c	b	1

BBWT($T_1\ T_2$)

T = b | ac | abb | abb = $T_1\ T_2\ T_3\ T_4$

- next Lyndon factor: ac

	F	L	
1	b	b	1

	F	L	
1	b	c	1
1	c	b	1

	F	L	
1	a	c	1
1	b	b	1
1	c	a	1

BBWT($T_1 T_2 T_3$)

T = b|ac|abb|abb

- next Lyndon factor: abb

	F	L	
1	a	c	1
1	b	b	1
1	c	a	1

BBWT(T_1 T_2 T_3)

T = b | ac | abb | abb

- next Lyndon factor: abb

	F	*L*	
1	a	c	1
1	b	b	1
1	c	a	1

	F	*L*	
1	a	b	1
1	b	c	1
2	b	b	2
1	c	a	1

BBWT(T_1 T_2 T_3)

T = b | ac | abb | abb

- next Lyndon factor: abb

	F	L	
1	a	c	1
1	b	b	1
1	c	a	1

	F	L	
1	a	b	1
1	b	c	1
2	b	b	2
1	c	a	1

	F	L	
1	a	b	1
1	b	c	1
2	b	b	2
3	b	b	3
1	c	a	1

BBWT(T_1 T_2 T_3)

T = b | ac | abb | abb

- next Lyndon factor: abb

	F	L	
1	a	c	1
1	b	b	1
1	c	a	1

	F	L	
1	a	b	1
1	b	c	1
2	b	b	2
1	c	a	1

	F	L	
1	a	b	1
1	b	c	1
2	b	b	2
3	b	b	3
1	c	a	1

	F	L	
1	a	b	1
2	a	c	1
1	b	b	2
2	b	b	3
3	b	a	1
1	c	a	2

text → BBWT *in-place*

- `|bacabbabb`

text → BBWT *in-place*

- |bacabbabb
- b|acabbabb

text → BBWT *in-place*

- |bacabbabb
- b|acabbabb
- bac|abbabb

text → BBWT *in-place*

- |bacabbabb
- b|acabbabb
- bac|abbabb
- cba|abbabb

text → BBWT *in-place*

- |bacabbbabb
- b|acabbbabb
- bac|abbbabb
- cba|abbbabb
- cbaabb|abb

text → BBWT *in-place*

- |bacabbabb
- b|acabbabb
- bac|abbabb
- cba|abbabb
- cbaabb|abb

⋮

text → BBWT *in-place*

- |bacabbabb
- b|acabbabb
- bac|abbabb
- cba|abbabb
- cbaabb|abb

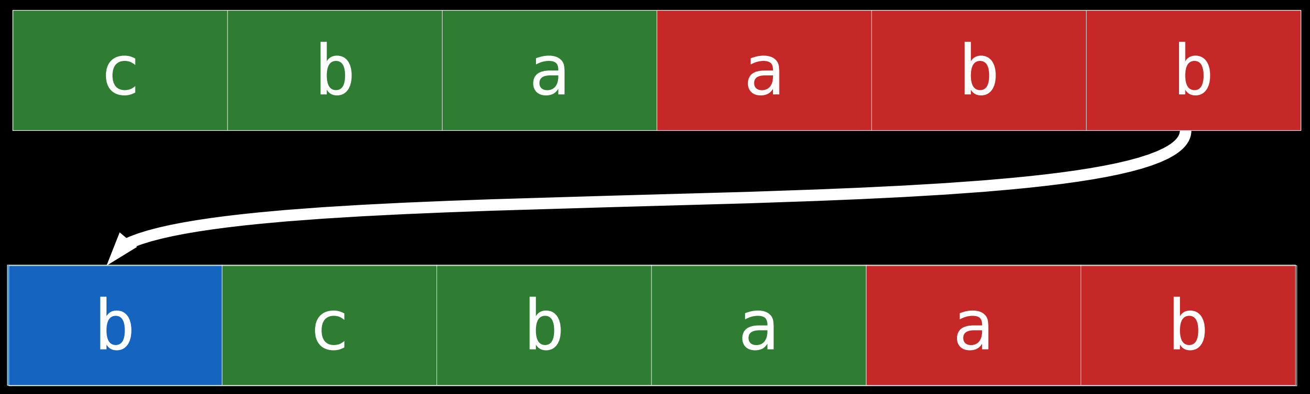
  ⋮

- bbcbbaaba|

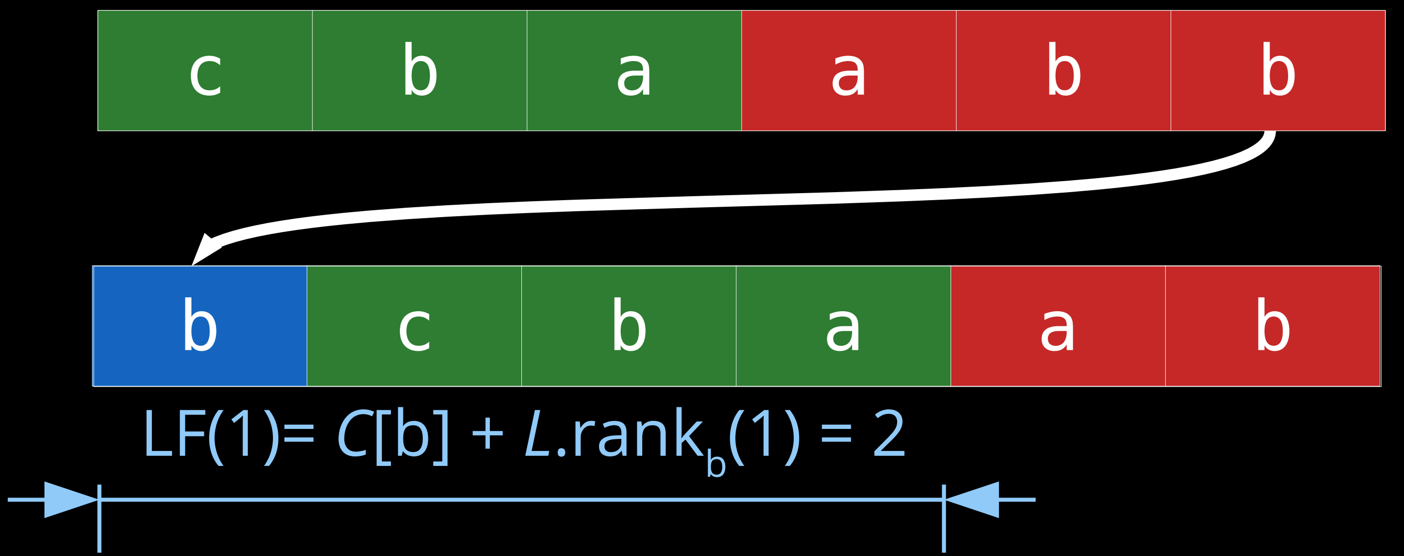
detailed transformation

c	b	a	a	b	b

detailed transformation

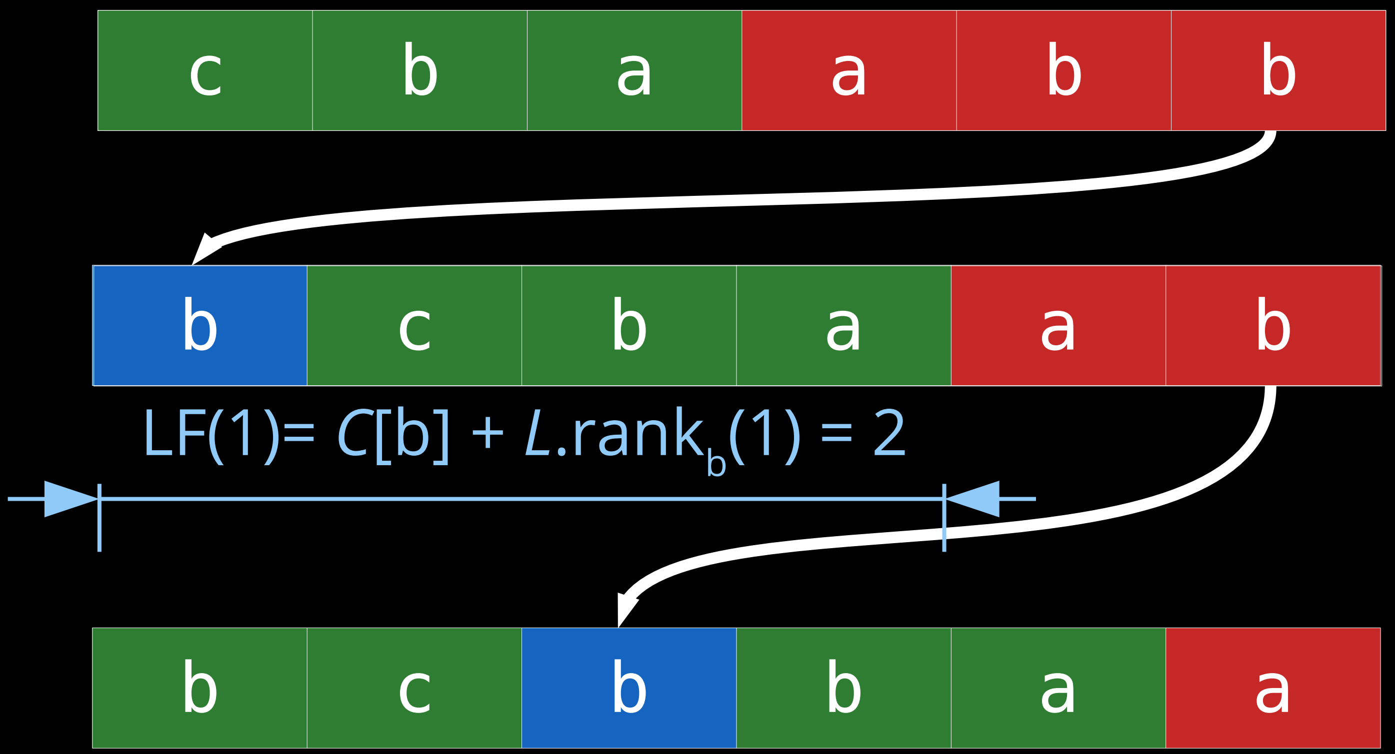


detailed transformation



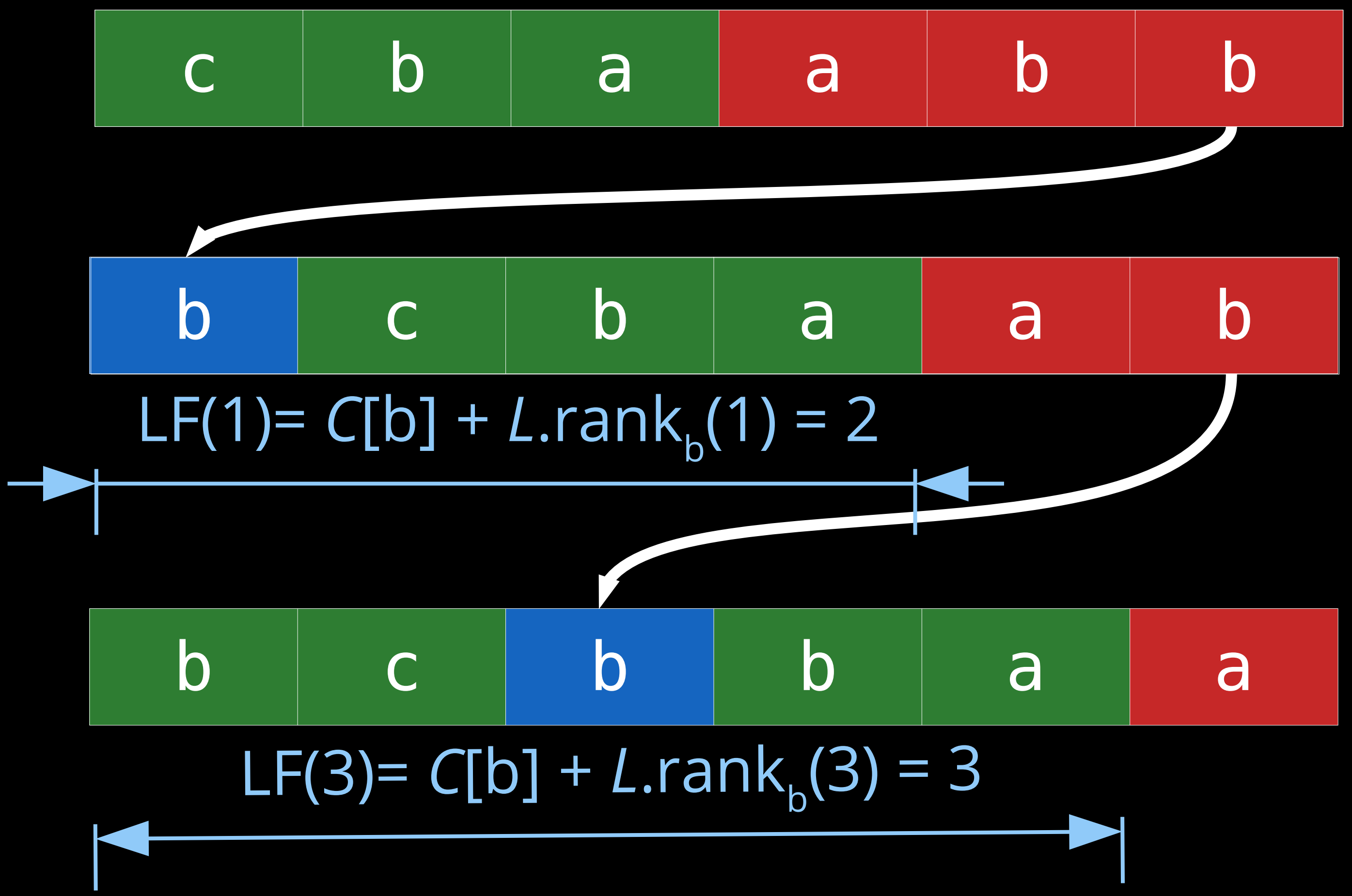
where $C[\mathrm{b}] := |\{\, j : L[j] < \mathrm{b} \,\}|$

detailed transformation



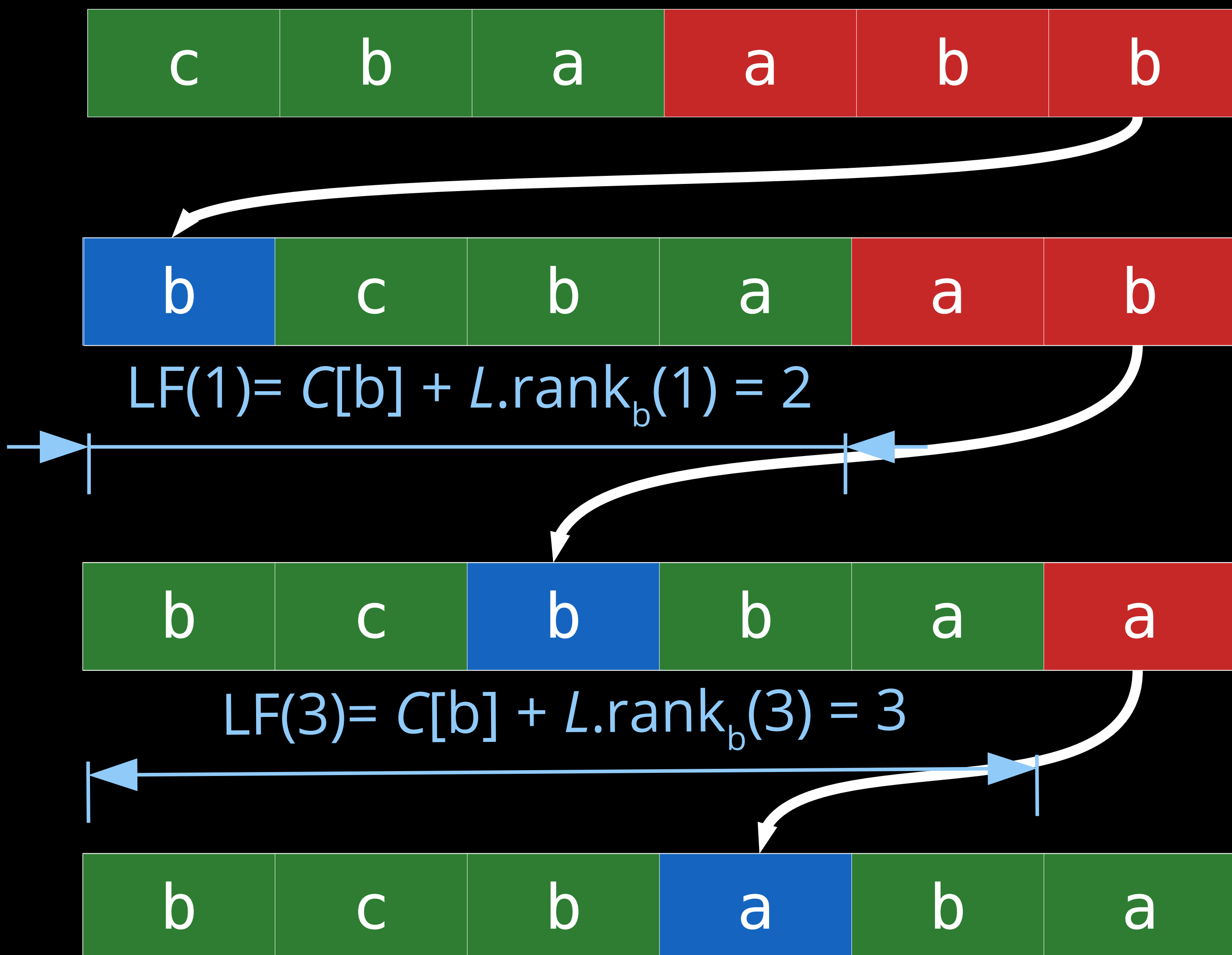
where $C[b] := |\{ j : L[j] < b \}|$

detailed transformation



where $C[\mathrm{b}] := |\{\, j : L[j] < \mathrm{b} \,\}|$

detailed transformation



where $C[b] := |\{ j : L[j] < b \}|$

BWT → BBWT

BWT → BBWT *in-place*

- Duval's algorithm
  - computes Lyndon factorization
  - it runs in $O(n\, t_L)$ time,
    where t_L is the time for accessing an entry of T
- algorithm uses linear scans from any $T[i]$ to $T[i+1]$

⇒ emulate this with FL mapping

⇒ $O(n^{2+\varepsilon})$ time only with L storing BWT

BWT → BBWT *in situ*

T = b | ac | abb | abb

	F	*L*	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	b	3
1	b	b	4
2	b	b	5
3	b	$	1
4	b	a	1
5	b	a	2
1	c	a	3

BWT → BBWT *in situ*

T = b | ac | abb | abb

	F	*L*	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	b	3
1	b	b	4
2	b	b	5
3	b	$	1
4	b	a	1
5	b	a	2
1	c	a	3

BWT → BBWT *in situ*

T = b|ac|abb|abb

- with FL mapping + Duval we detect the first Lyndon factor b|a …

	F	*L*	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	b	3
1	b	b	4
2	b	b	5
3	b	$	1
4	b	a	1
5	b	a	2
1	c	a	3

construction of a cycle

T = b|ac|abb|abb

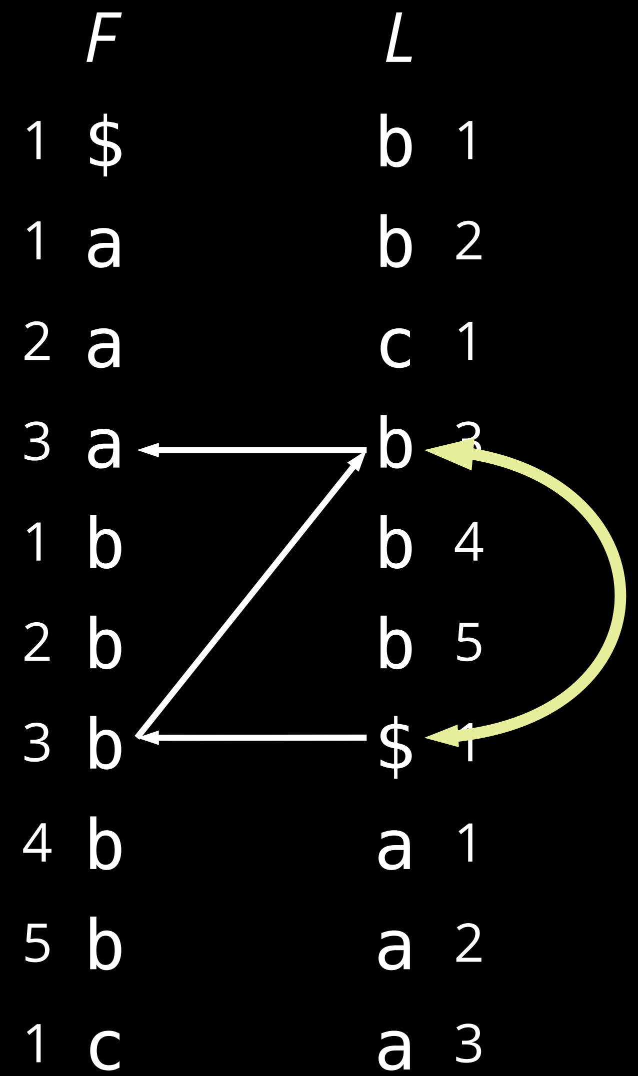
- aim: create cycle b → b

	F	*L*	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	b	3
1	b	b	4
2	b	b	5
3	b	$	1
4	b	a	1
5	b	a	2
1	c	a	3

construction of a cycle

T = b | ac | abb | abb

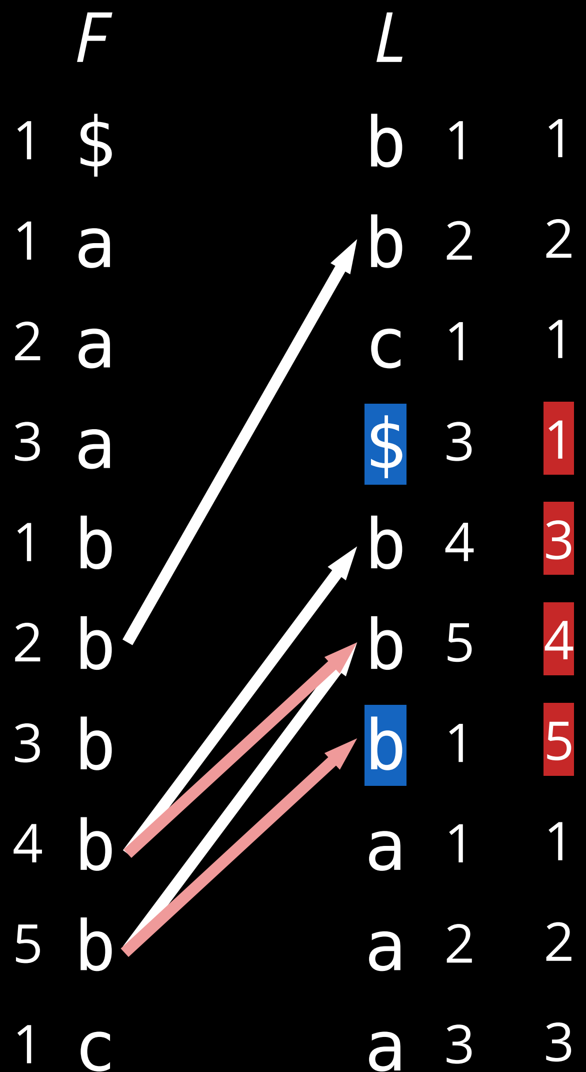
- aim: create cycle b → b
- since FL maps $ to *T*[1] we want to exchange $ and b



construction of a cycle

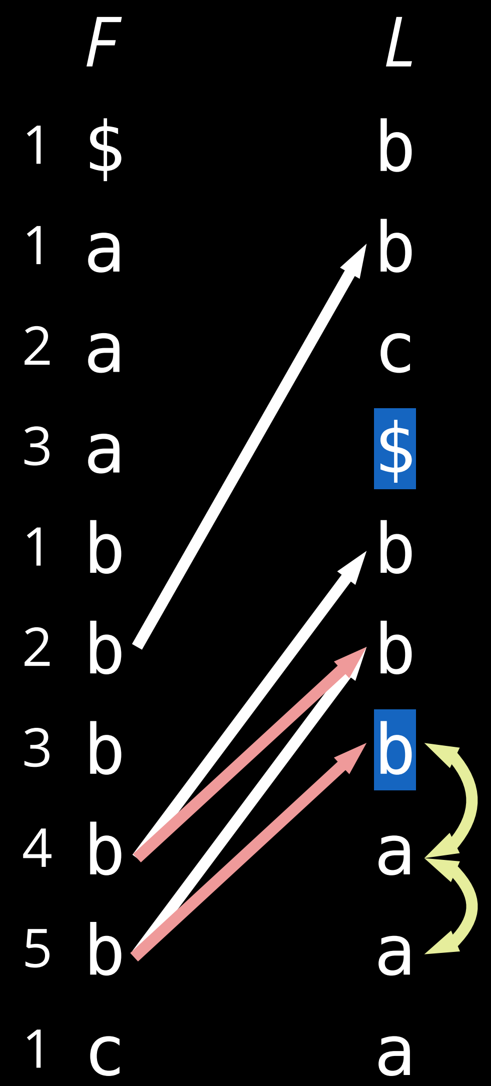
T = b | ac | abb | abb

- aim: create cycle b → b
- since FL maps $ to *T*[1] we want to exchange $ and b
- however: might not work
- need to fix red arrows



construction of a cycle

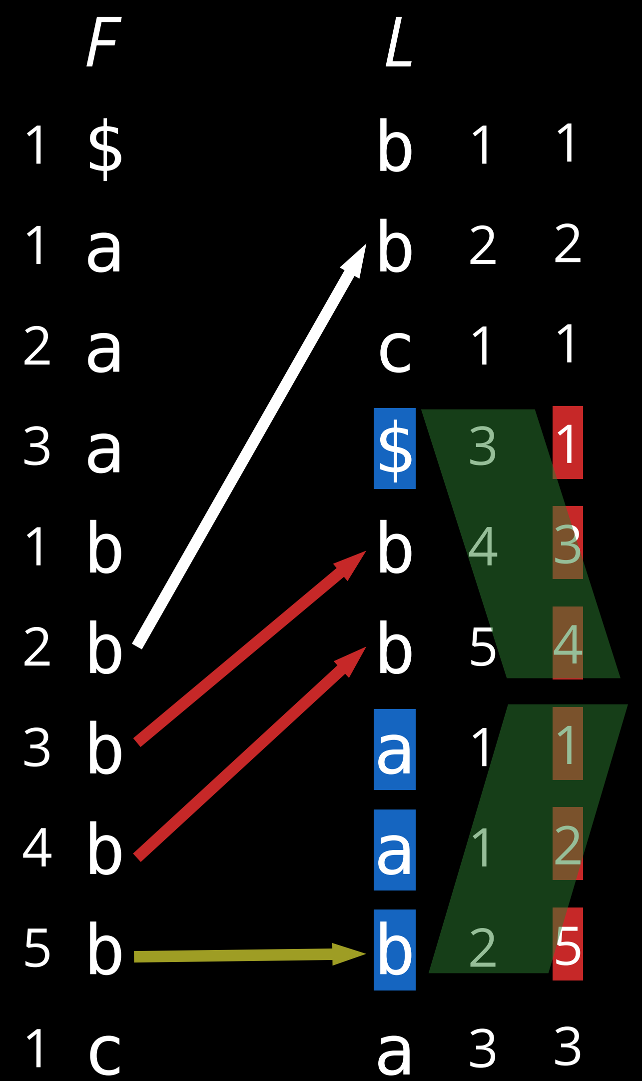
- since there are two red arrows:
- switch below the exchanged b the next two entries



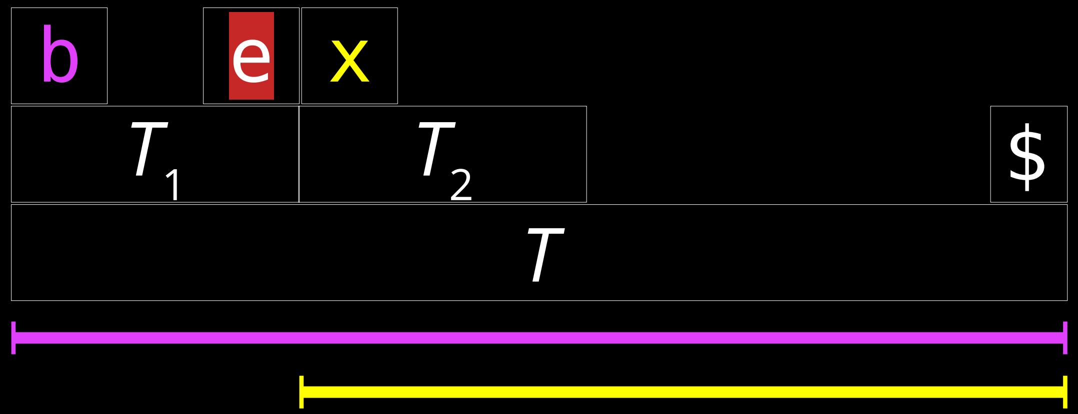
construction of a cycle

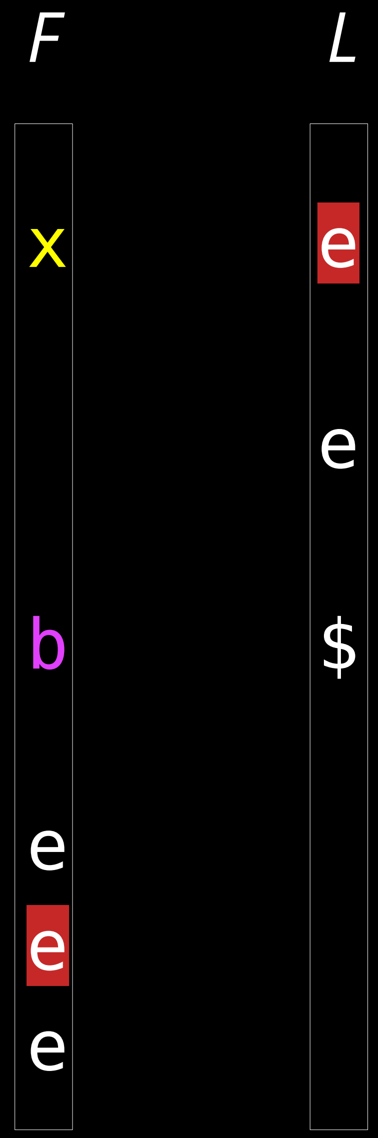
- the cycle moved below the exchange

⇒ modified LF mapping just “moved”



abstract idea

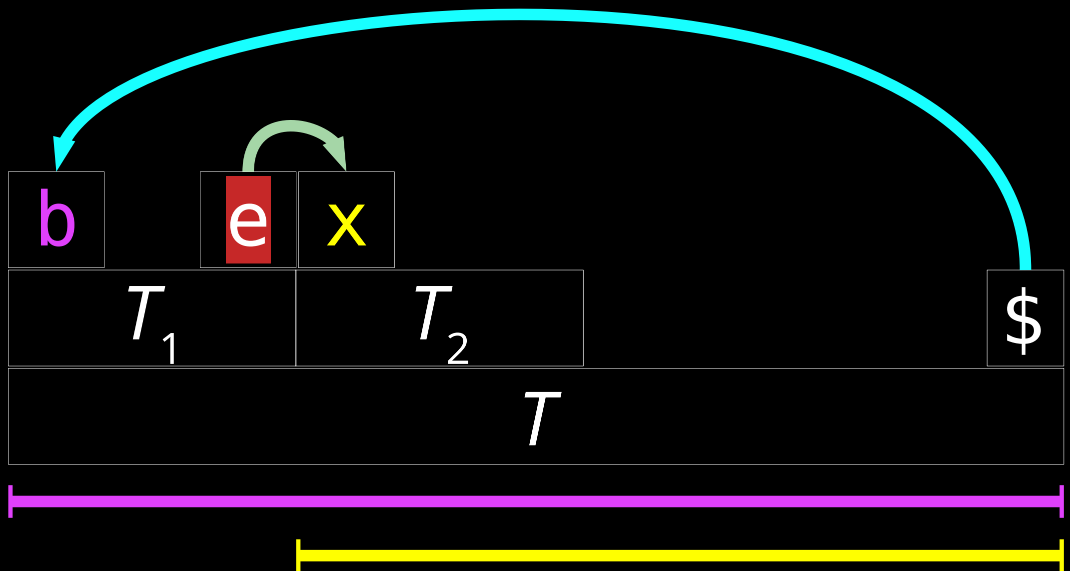


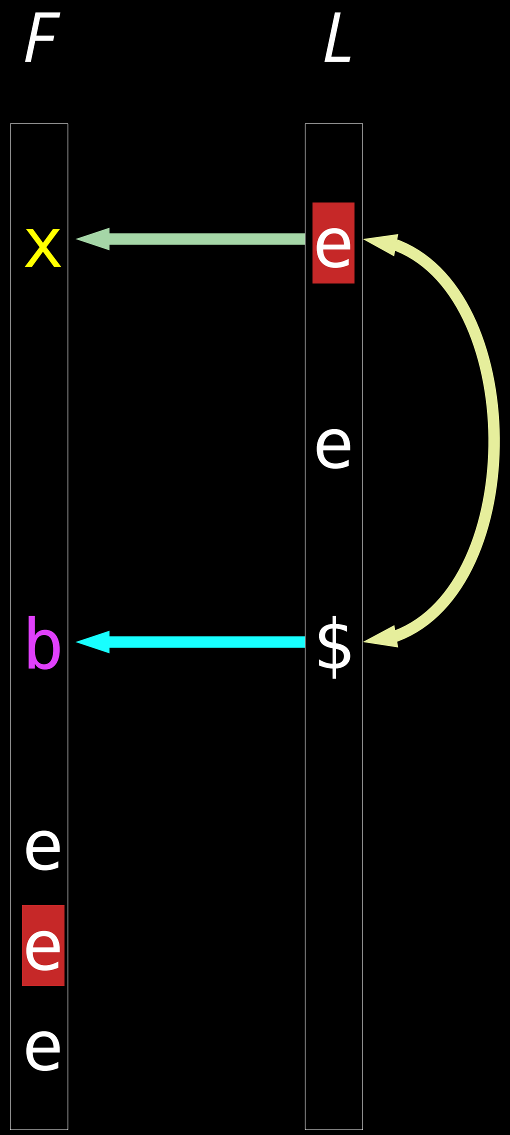


- $T_1 \geq_{\text{lex}} T_2 \geq_{\text{lex}} \cdots \geq_{\text{lex}} T_t$

  $\Rightarrow T[1..] >_{\text{lex}} T[|T_1|..]$

abstract idea

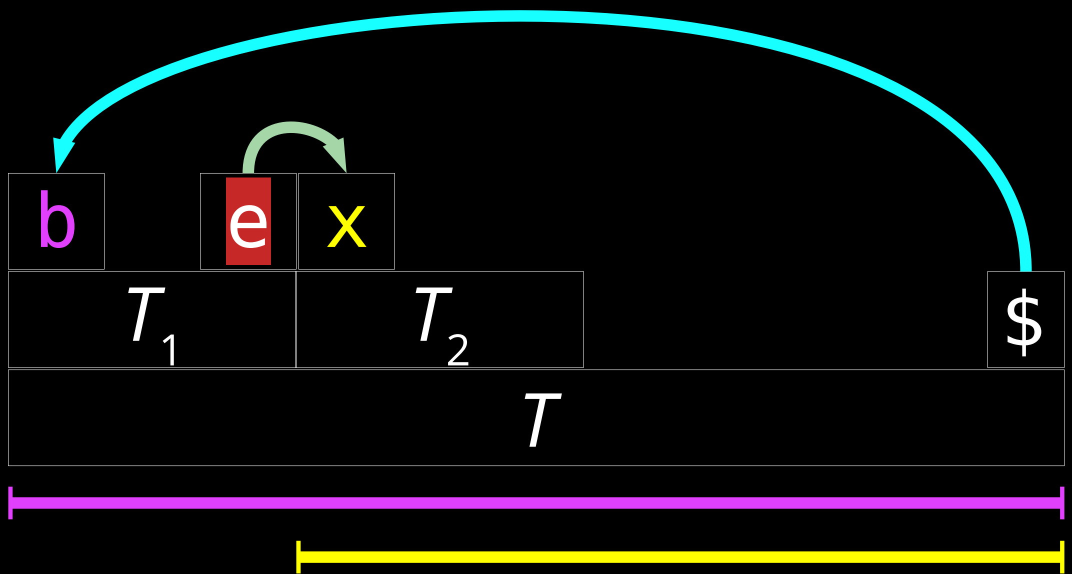


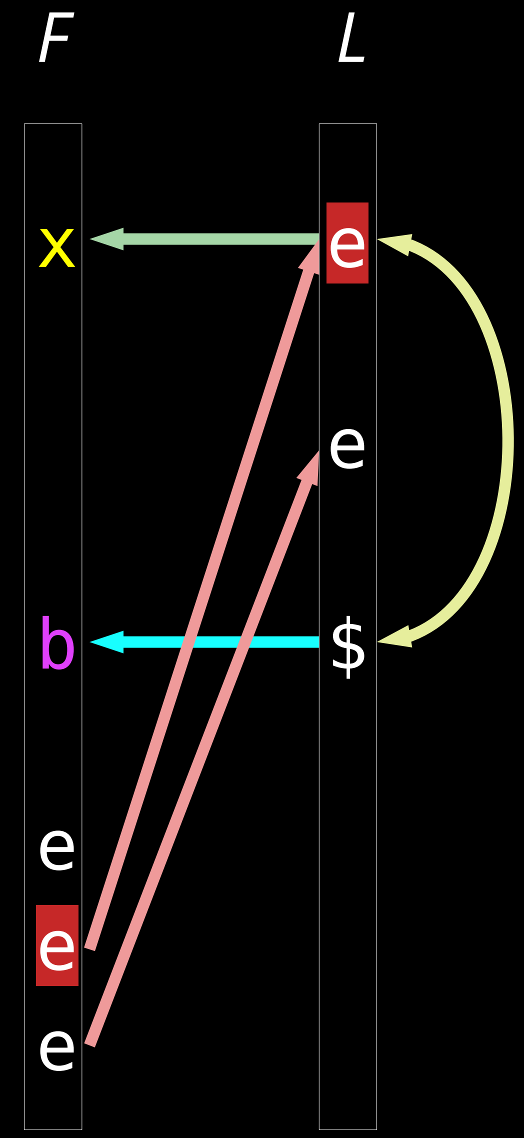


- $T_1 \geq_{\text{lex}} T_2 \geq_{\text{lex}} \cdots \geq_{\text{lex}} T_t$

  $\Rightarrow T[1..] >_{\text{lex}} T[\,|T_1|\,..]$

abstract idea



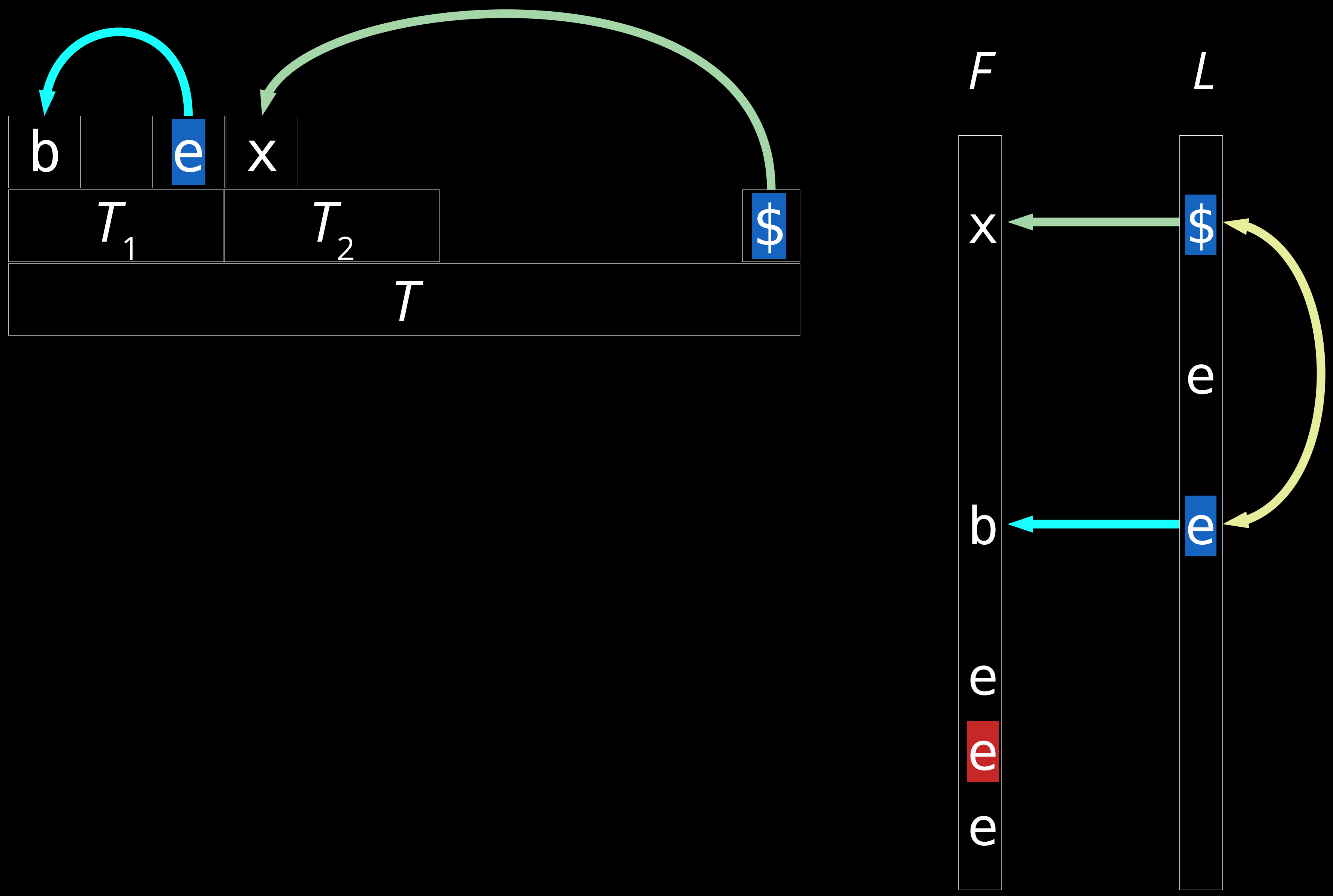


- $T_1 \geq_{\text{lex}} T_2 \geq_{\text{lex}} \cdots \geq_{\text{lex}} T_t$

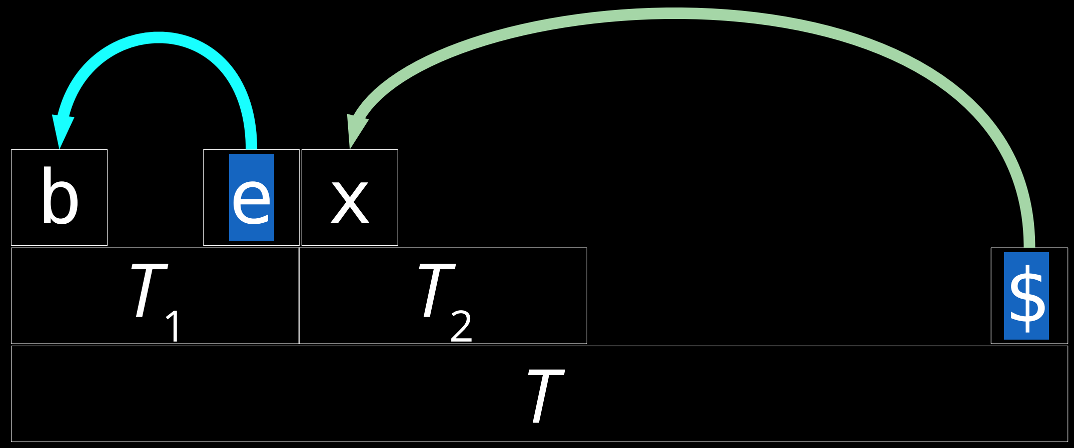
  $\Rightarrow T[1..] >_{\text{lex}} T[|T_1|..]$

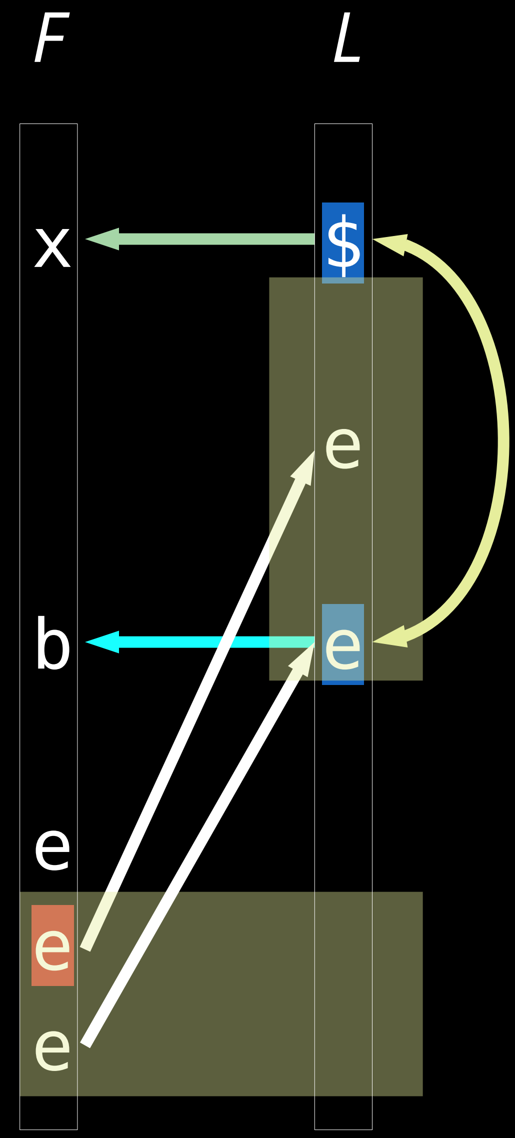
- need to change red arrows

abstract idea



abstract idea





the number of e's between the exchanged $ and e =
the number of entries to switch after the e in *F* that mapped to the exchanged e

carrying on with example

	F	*L*	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	$	1
1	b	b	3
2	b	b	4
3	b	a	1
4	b	a	2
5	b	b	5
1	c	a	3

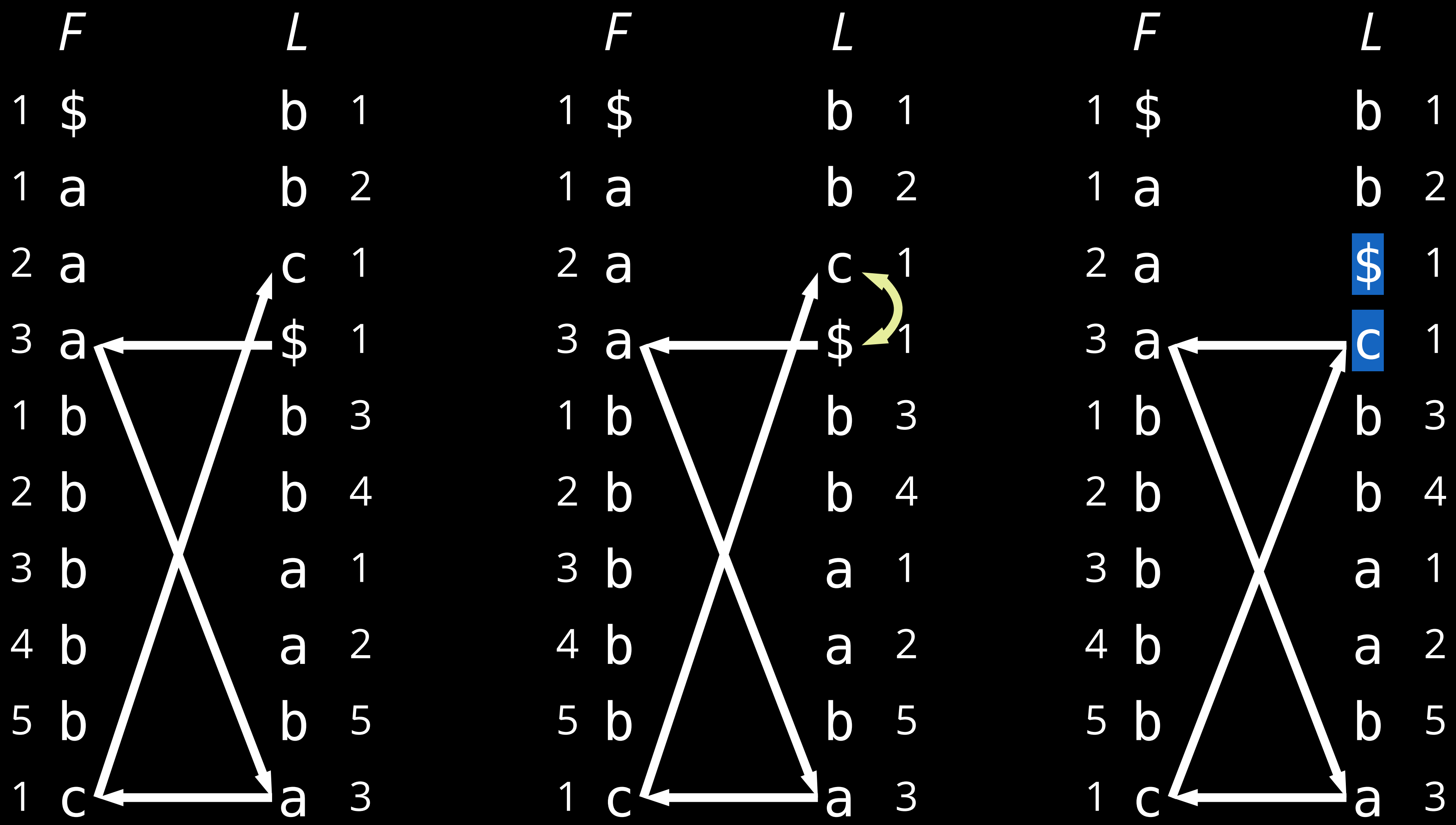
carrying on with example

	F	*L*	
1	$	b	1
1	a	b	2
2	a	c	1
3	a	$	1
1	b	b	3
2	b	b	4
3	b	a	1
4	b	a	2
5	b	b	5
1	c	a	3

carrying on with example

	F	*L*			*F*	*L*	
1	$	b	1	1	$	b	1
1	a	b	2	1	a	b	2
2	a	c	1	2	a	c	1
3	a	$	1	3	a	$	1
1	b	b	3	1	b	b	3
2	b	b	4	2	b	b	4
3	b	a	1	3	b	a	1
4	b	a	2	4	b	a	2
5	b	b	5	5	b	b	5
1	c	a	3	1	c	a	3

carrying on with example



open problems

O($n^{1+\varepsilon}$) time

- can we get rid of the FL mapping?

  (use only LF mapping)

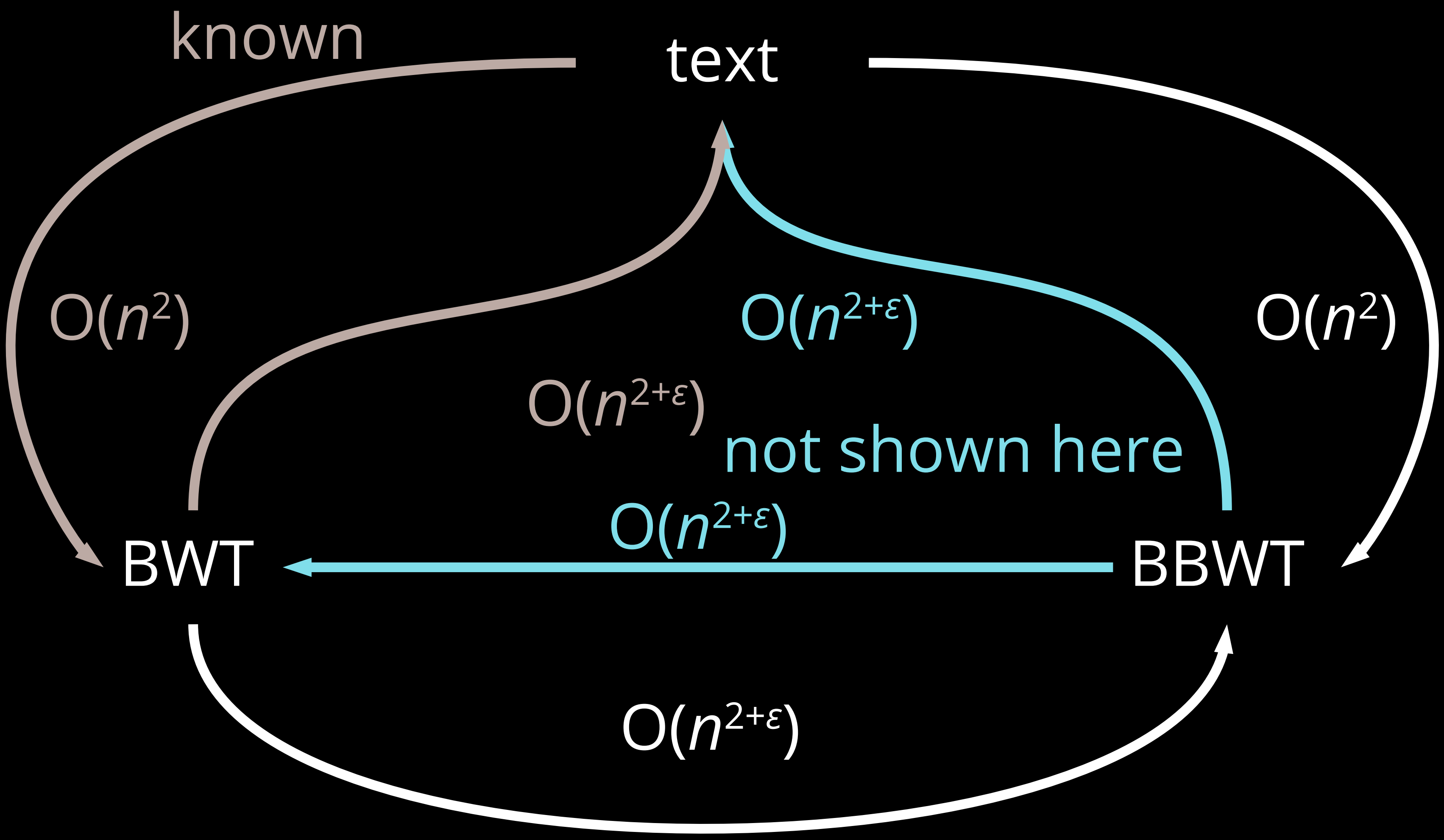
- trade-off algorithm for time ↔ space
- Is the number of distinct Lyndon words of *T* bounded by the runs in the BBWT of *T* ?

  if so:

  O(r) words run-length compressed BBWT-index

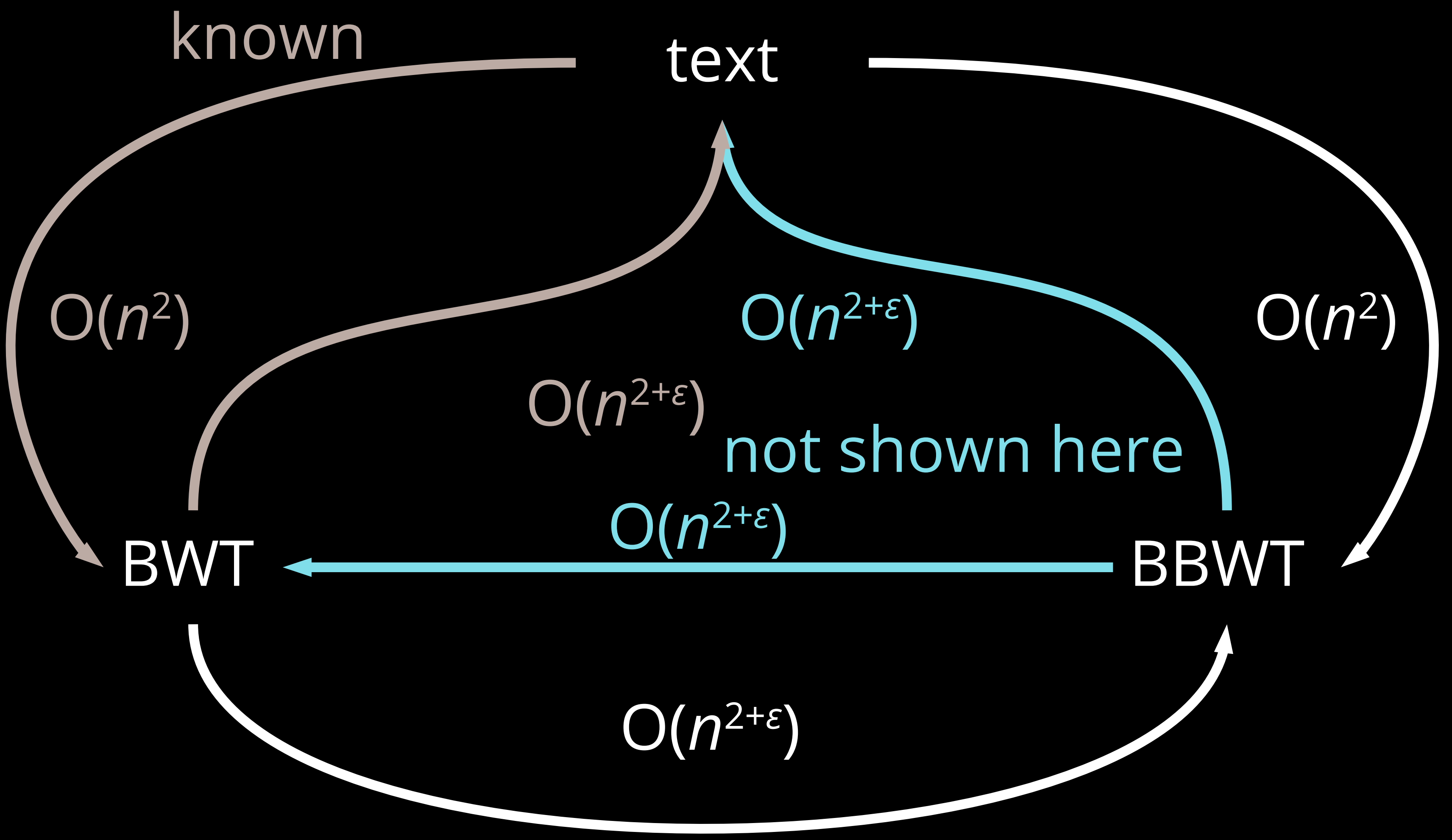
  (r : runs in BBWT)

in-place conversions



working space: $n \lg \sigma + O(\lg n)$ bits (including text)

in-place conversions



working space: $n \lg \sigma + O(\lg n)$ bits (including text)

any questions are welcome!