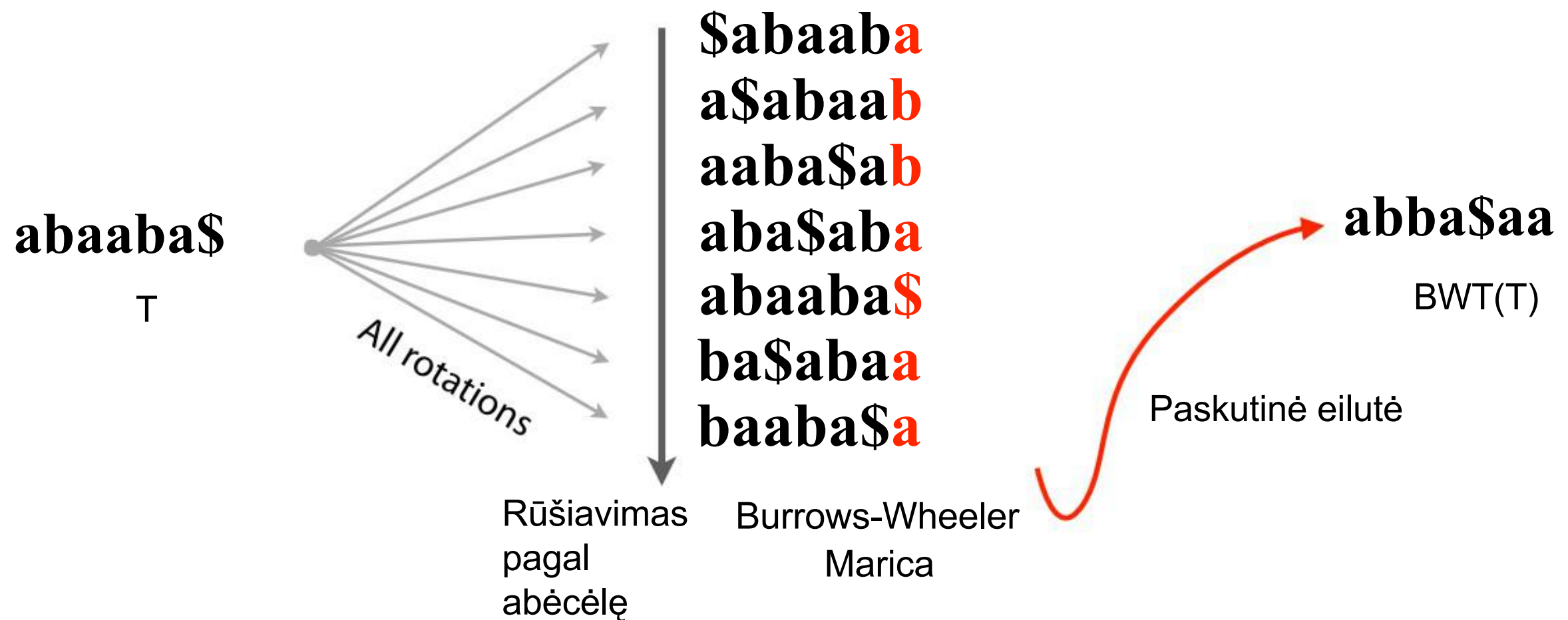


# Burrows-Wheeler Transformacija

*Grižtama* eilutės simbolių permutacija, originaliai naudota duomenų suspaudimui



Kokia nauda suspaudimui? Kaip ji gali būti grįžtama? Kaip gali būti panaudojama indeksuojant tekstą paieškai?

Burrows M, Wheeler DJ: A block sorting lossless data compression algorithm.  
*Digital Equipment Corporation, Palo Alto, CA* 1994, Technical Report 124; 1994

# Burrows-Wheeler Transformacija

```
def rotations(t):  
    """ Return list of rotations of input string t """  
    tt = t * 2  
    return [ tt[i:i+len(t)] for i in xrange(0, len(t)) ]
```

Make list of all rotations

```
def bwm(t):  
    """ Return lexicographically sorted list of t's rotations """  
    return sorted(rotations(t))
```

Sort them

```
def bwtViaBwm(t):  
    """ Given T, returns BWT(T) by way of the BWM """  
    return ''.join(map(lambda x: x[-1], bwm(t)))
```

Take last column

```
!!! bwtViaBwm('Tomoreilutè_and_tomoreilutè_and_tomoreilutè%')  
'w% wwdd__nnooaaattTmmrrrrrrrooo__ooo'  
  
!!! bwtViaBwm('It_was_the_best_of_times_it_was_the_worst_of_times%')  
's% esttssffteww_hhmmbootttt_ii__woeeaaressli_____  
  
!!! bwtViaBwm('in_the_jingle_jangle_morning_III_come_following_you%')  
'u_gleeeengj_mhl_nnnnt% nwj__lgglolo_iiiarfcmlylo_oo_'
```

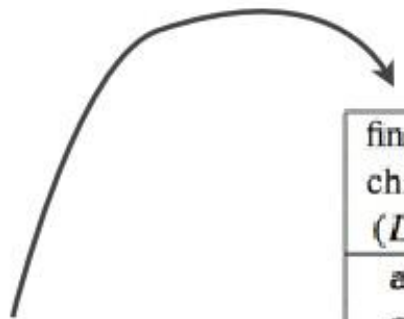
Python example: <http://nbviewer.ipython.org/6798379>

# Burrows-Wheeler Transformacija

Simboliai išrūšiuojami BWT remiantis  
dešiniuoju kontekstu

BWT(T) Suteikia papildomą struktūrą  
ir padaro tekstą labiau suspaudžiamą

bzip2...

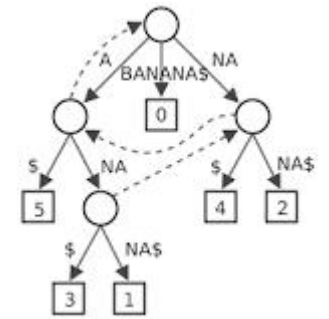


final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set \$L[i]\$ to be the
i	n turn, set \$R[i]\$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector \$R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with \$ch\$ appear in the {\em same order
i	n with \$ch\$. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Figure 1: Example of sorted rotations. Twenty consecutive rotations from the sorted list of rotations of a version of this paper are shown, together with the final character of each rotation.

# Burrows-Wheeler Transformacija

BWM primena priesagų masyvą (padaromą iš priesagų medžio)



**\$**abaaba  
a**\$**abaab  
aaba**\$**ab  
aba**\$**aba  
abaaba**\$**  
ba**\$**abaa  
baaba**\$**a

BWM(T)

6	<b>\$</b>
5	a <b>\$</b>
2	aaba <b>\$</b>
3	aba <b>\$</b>
0	abaaba <b>\$</b>
4	ba <b>\$</b>
1	baaba <b>\$</b>

SA(T)

Išrūšiavimo tvarka atitinka

# Burrows-Wheeler Transformacija

$T = \text{abaaba\$}$

Galima sukonstruoti  $BWT(T)$  iš  $SA(T)$ :

$$BWT[i] = \begin{cases} T[SA[i] - 1] & \text{if } SA[i] > 0 \\ \$ & \text{if } SA[i] = 0 \end{cases}$$

“BWT = simboliai yra per vieną poziciją kairiau priesagose iš priesagų masyvo”

$\$$ abaaba  
a $\$$ abaab  
aaba $\$$ ab  
aba $\$$ aba  
abaaba $\$$   
ba $\$$ abaa  
baaba $\$$ a

BWM(T)

6	$\$$
5	a $\$$
2	aaba $\$$
3	aba $\$$
0	abaaba $\$$
4	ba $\$$
1	baaba $\$$

SA(T)

# Burrows-Wheeler Transformacija

```
def suffixArray(s):
```

```
    ““ Given T return suffix array SA(T). We use Python’s sorted  
        function here for simplicity, but we can do better. ““
```

```
    satups = sorted([(s[i:], i) for i in xrange(0, len(s))])
```

```
    # Extract and return just the offsets
```

```
    return map(lambda x: x[1], satups)
```

sudarom priesgų masyvą

```
def bwtViaSa(t):
```

```
    ““ Given T, returns BWT(T) by way of the suffix array. ““
```

```
    bw = []
```

```
    for si in suffixArray(t):
```

```
        if si == 0: bw.append('% ')
```

```
        else: bw.append(t[si/ 1])
```

```
    return ''.join(bw) # return string/ ized version of list bw
```

Surenkam pirmus simbolius  
į kairę iš priesagų masyvo

```
!!! bwtViaSa('Tomoreilutė_and_tomoreilutė_and_tomoreilutė% ')\n'w% wwdd__nnooaaattTmmrrrrrrrooo__ooo'
```

```
!!! bwtViaSa('It_was_the_best_of_times_it_was_the_worst_of_times% ')\n's% esttssffteww_hhmmbootttt_ii__woeeaaressli_____'
```

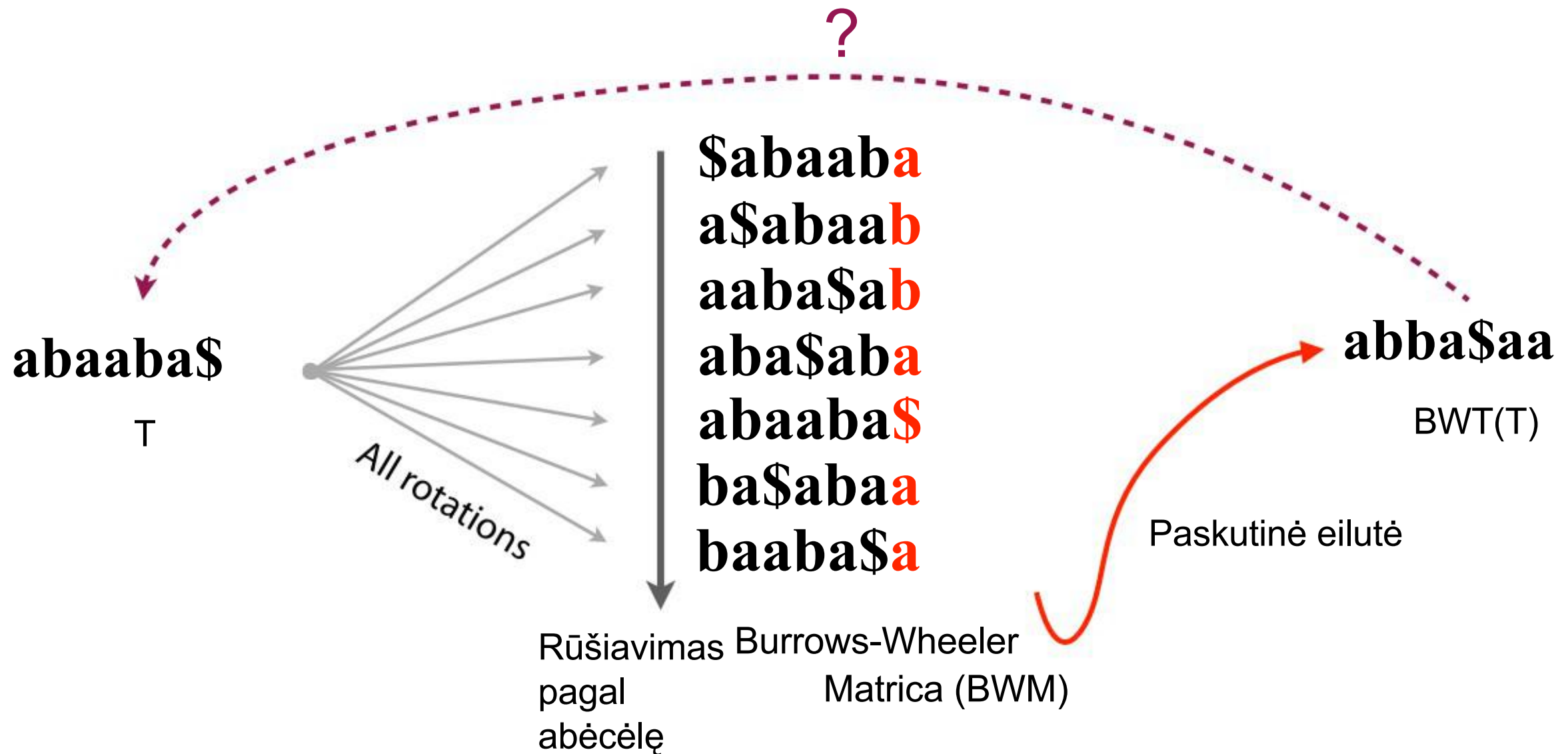
```
!!! bwtViaSa('in_the_jingle_jangle_morning_III_come_following_you% ')\n'u_gleeeengj_mhl_nnnnt% nwj__lgglolo_iiiarfcmlylo__oo_'
```

Python example: <http://nbviewer.ipython.org/6798379>



# Burrows-Wheeler Transformacija

Kaip padaryti ją grįžtama?



BWM turi esminę savybę, vadinamą *LF Atvaizdis*...

# Burrows-Wheeler Transformacija: T-rangavimas

Suteikiam kiekvinam simboliui T rangą, lygų skaičiui kiek prieš tai buvo rasta šio simbolio tekste T. Tai ž T-rangavimas

**a0 b0 a1 a2 b1 a3 \$**

Perrašom BWM įtraukdami rangus...



# Burrows-Wheeler Transformacija

BWM with T-rangavimas:

$F$							$L$
\$	$a_0$	$b_0$	$a_1$	$a_2$	$b_1$	$a_3$	
$a_3$	\$	$a_0$	$b_0$	$a_1$	$a_2$	$b_1$	
$a_1$	$a_2$	$b_1$	$a_3$	\$	$a_0$	$b_0$	
$a_2$	$b_1$	$a_3$	\$	$a_0$	$b_0$	$a_1$	
$a_0$	$b_0$	$a_1$	$a_2$	$b_1$	$a_3$	\$	
$b_1$	$a_3$	\$	$a_0$	$b_0$	$a_1$	$a_2$	
$b_0$	$a_1$	$a_2$	$b_1$	$a_3$	\$	$a_0$	

Pažiūrėkit į pirmą ( $F$ ) ir paskutinį ( $L$ ) stulpelį

Atreipkit dėmesį į  $a_s$

$a_s$  išsidėsto ta pačia tvarka  $F$  ir  $L$ . Abiejuose stulpeliuose matome:

$a_3, a_1, a_2, a_0$

# Burrows-Wheeler Transformacija

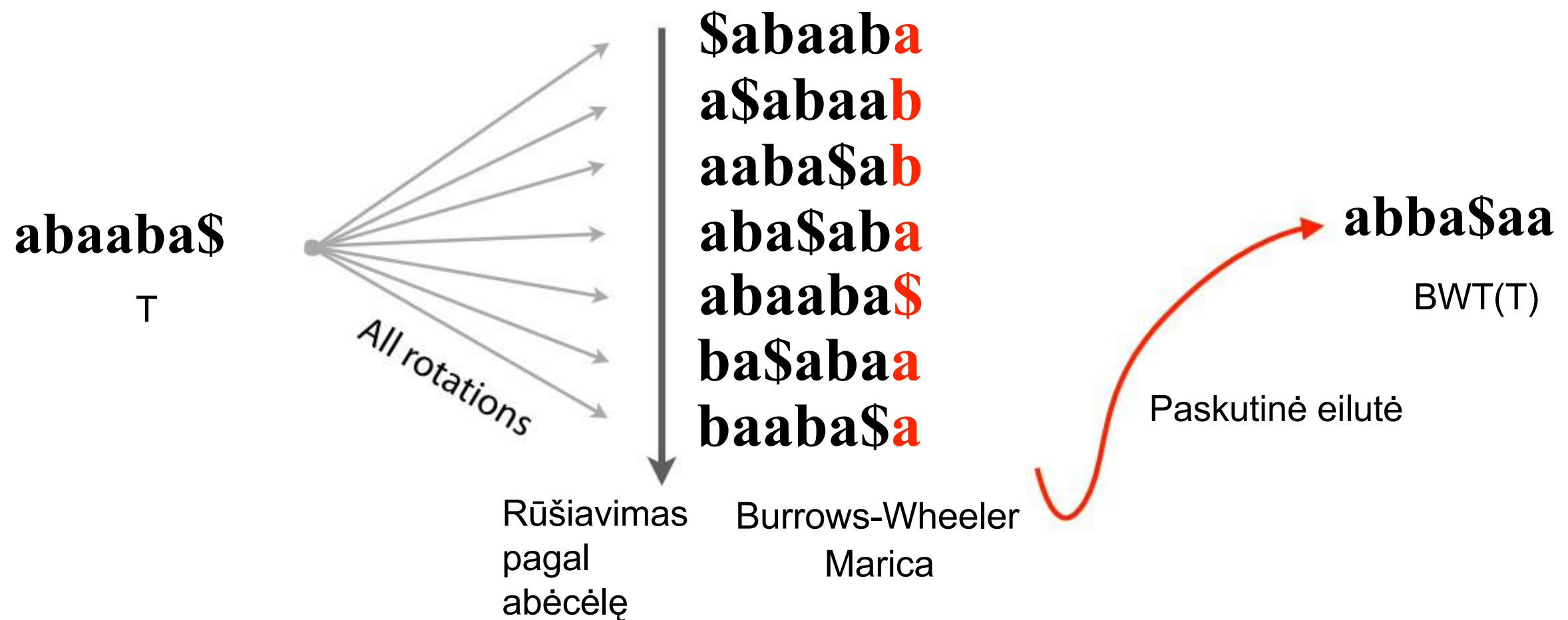
BWM with T-rangavimas:

<i>F</i>							<i>L</i>
\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	
a <sub>3</sub>	\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	<b>b<sub>1</sub></b>	
a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	a <sub>0</sub>	<b>b<sub>0</sub></b>	
a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	
a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	
<b>b<sub>1</sub></b>	a <sub>3</sub>	\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	
<b>b<sub>0</sub></b>	a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	a <sub>0</sub>	

Tas pats ir su **bs**: **b<sub>1</sub>**, **b<sub>0</sub>**

# Burrows-Wheeler Transformacija

*Grižtama* eilutės simbolių permutacija, originaliai naudota duomenų suspaudimui



Kokia nauda suspaudimui? Kaip ji gali būti grįžtama? Kaip gali būti panaudojama indeksuojant tekstą paieškai?

# Burrows-Wheeler Transformacija: LF Atvaizdis

BWM suT-rangavimu:

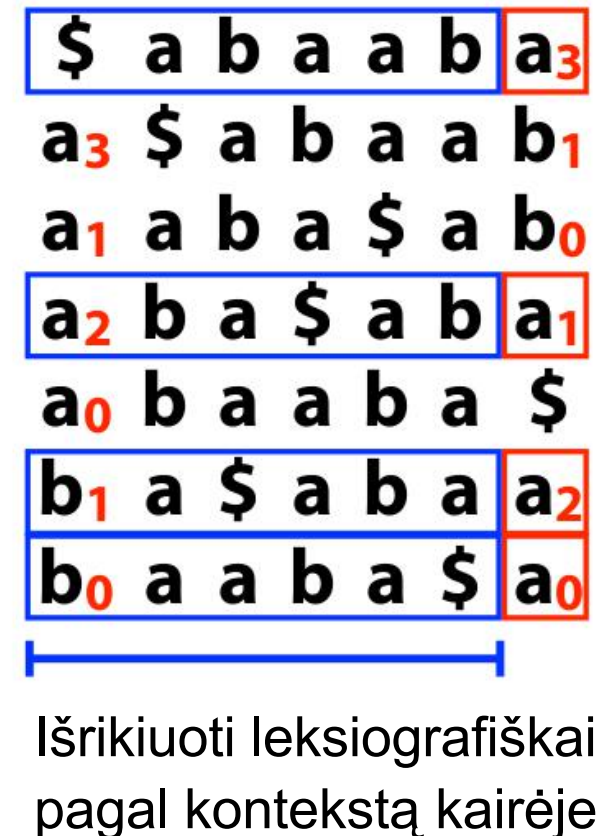
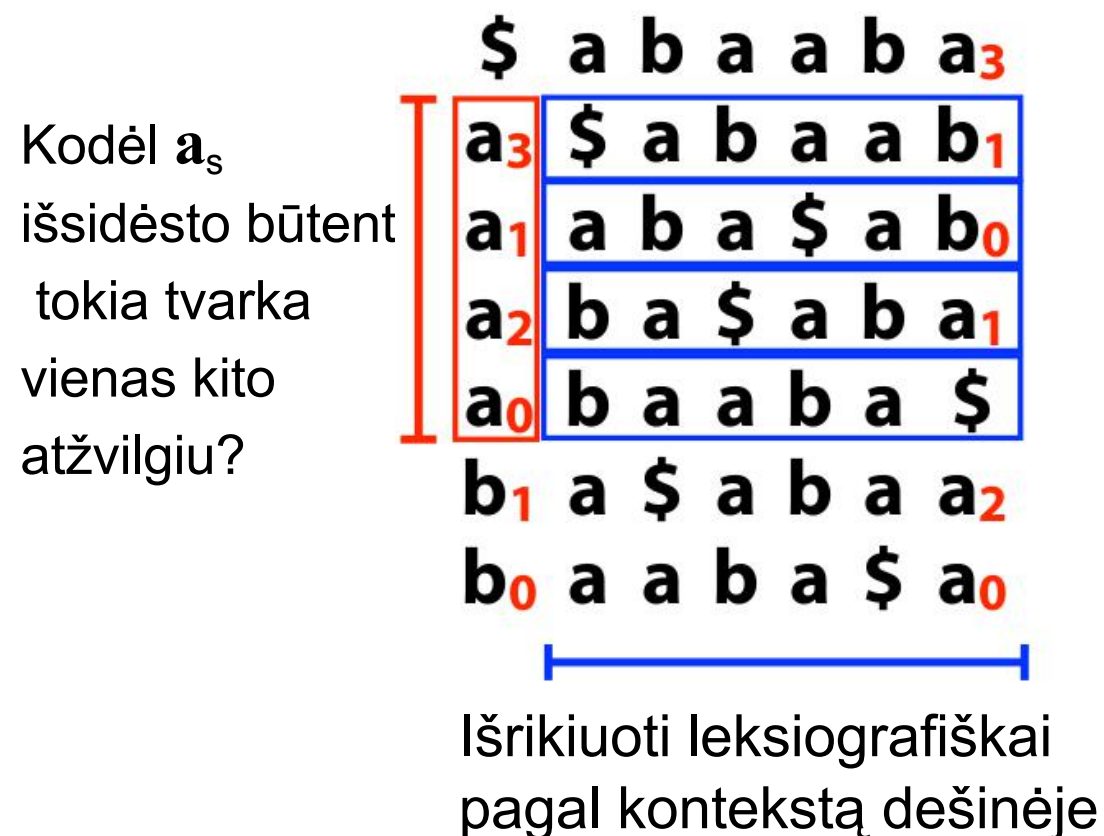
$F$	$L$
\$ a0 b0 a1 a2 b1 a3	
a3 \$ a0 b0 a1 a2 b1	
a1 a2 b1 a3 \$ a0 b0	
a2 b1 a3 \$ a0 b0 a1	
a0 b0 a1 a2 b1 a3 \$	
b1 a3 \$ a0 b0 a1 a2	
b0 a1 a2 b1 a3 \$ a0	

LF Atvaizdis:  $i$ -tasis simbolis  $c$  sutiktas  $L$  ir and the  $i$ -tasis simbolis  $c$  sutiktas  $F$  atitinka tą patį simbolį ir tą pačią poziciją  $T$

surangavus pagal  $c$  sutikimus tekste, rangų eiliškumas  $F$  ir  $L$  sutampa.

# Burrows-Wheeler Transformacija: LF Atvaizdis

Kodėl LF Atvaizdis veikia?



Kodėl  $a_s$  išsidėsto būtent tokia tvarka vienas kito atžvilgiu?

$c$  simbolio sutikimai  $F$  yra išrikiuoti pagal kontekstą dešinėje. Pagal tą pačią sritį išrikiuojamas ir  $L$ !

Kad ir kokį rangą priskirsi simboliams iš  $T$ , rangai  $F$  ir  $L$  sutaps

# Burrows-Wheeler Transformacija: LF Atvaizdis

BWM T-rangavimas:

<i>F</i>							<i>L</i>
\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	
a <sub>3</sub>	\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	
a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	a <sub>0</sub>	b <sub>0</sub>	
a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	
a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	
b <sub>1</sub>	a <sub>3</sub>	\$	a <sub>0</sub>	b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	
b <sub>0</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>1</sub>	a <sub>3</sub>	\$	a <sub>0</sub>	

Tekstinėms paieškoms geresnis rangavimas toks, kai rangai suteikiami pagal simbolių poziciją *F* ir *L*. Tai - B-rangavimas.

# Burrows-Wheeler Transformacija: LF Atvaizdis

BWM B-rangavimas:

<i>F</i>							<i>L</i>	
	\$	a <sub>3</sub>	b <sub>1</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>0</sub>	a <sub>0</sub>	
	a <sub>0</sub>	\$	a <sub>3</sub>	b <sub>1</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>0</sub>	
	a <sub>1</sub>	a <sub>2</sub>	b <sub>0</sub>	a <sub>3</sub>	\$	a <sub>3</sub>	b <sub>1</sub>	
	a <sub>2</sub>	b <sub>0</sub>	a <sub>0</sub>	\$	a <sub>3</sub>	b <sub>1</sub>	a <sub>1</sub>	
	a <sub>3</sub>	b <sub>1</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>0</sub>	a <sub>0</sub>	\$	
	b <sub>0</sub>	a <sub>0</sub>	\$	a <sub>3</sub>	b <sub>1</sub>	a <sub>1</sub>	a <sub>2</sub>	
	b <sub>1</sub>	a <sub>1</sub>	a <sub>2</sub>	b <sub>0</sub>	a <sub>0</sub>	\$	a <sub>3</sub>	

Mažėjantis rangas

*F* dabar turi labai paprastą struktūrą: \$, bloką a<sub>s</sub> su didėjančiais rangais, bloką b<sub>s</sub> su didėjančiais rangais.



# Burrows-Wheeler Transformacija

*F*      *L*

\$      **a**<sub>0</sub>

**a**<sub>0</sub>      **b**<sub>0</sub>

**a**<sub>1</sub>      **b**<sub>1</sub> ← Kuri BWM eilutė prasideda **b**<sub>1</sub>?

**a**<sub>2</sub>      **a**<sub>1</sub>      Praleidžiam eilutę prasidedančią \$ (1 eilutė )

**a**<sub>3</sub>      \$      Praleidžiam eilutę prasidedančią **a** (4 eilutės)

Praleidžiam eilutę prasidedančią **b**<sub>0</sub> (1 eilutė)

**b**<sub>0</sub>      **a**<sub>2</sub>

Atsakymas: eilutė 6

eilutė 6 → **b**<sub>1</sub>      **a**<sub>3</sub>

# Burrows-Wheeler Transformacija

Tegul  $T$  susideda iš 300 **A**, 400 **C**, 250 **G**, 700 **T** ir  $\$ < \mathbf{A} < \mathbf{C} < \mathbf{G} < \mathbf{T}$

Kuri BWM eilutė (indeksavimas nuo 0) prasideda **G**100? (B-rangavimas)

Praleidžiam eilutę prasidedančią **\$** (1 eilutė)

Praleidžiam eilutę prasidedančią **A** (300 eilučių)

Praleidžiam eilutę prasidedančią **C** (400 eilučių)

Praleidžiam pirmas 100 eilučių prasidedančių **G** (100 eilučių)

atsakymas: eilutė  $1 + 300 + 400 + 100 =$  **eilutė 801**

# Burrows-Wheeler Transformacija: grįžimas

Vykdam grįžtamąją transformaciją BWT( $T$ ) pradedama nuo dešinėsios  $T$  dalies ir judama kairėn

**Pradedam** pirmoje eilutėje.  $F$  privalo turėti  $\$$ .  $L$  turi simbolį, kuris yra prieš  $\$$ :  $a_0$

$a_0$ : LF Atvaizdis rodo, kad tai yra tas pats  $a$  aptikimas, kaip ir pirmas  $a$   $F$ . **Einam** į eilutę prasidedančią  $a_0$ .  $L$  turi simbolį, kuris yra prieš

$a_0$ :  $b_0$ .

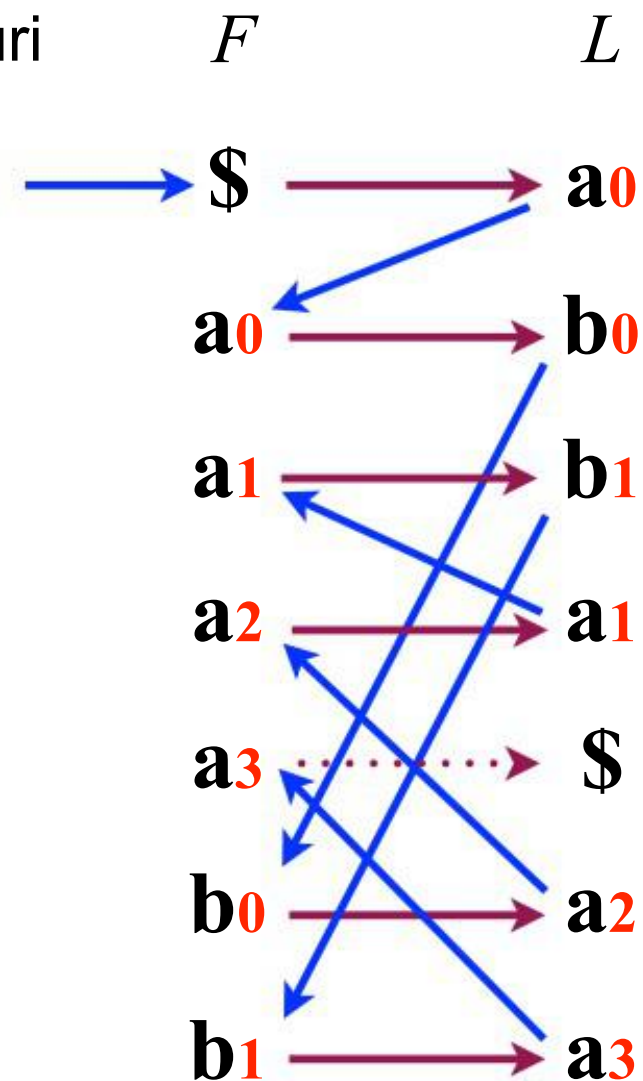
Kartojam  $b_0$ , gaunam  $a_2$

Kartojam  $a_2$ , gaunam  $a_1$

Kartojam  $a_1$ , gaunam  $b_1$

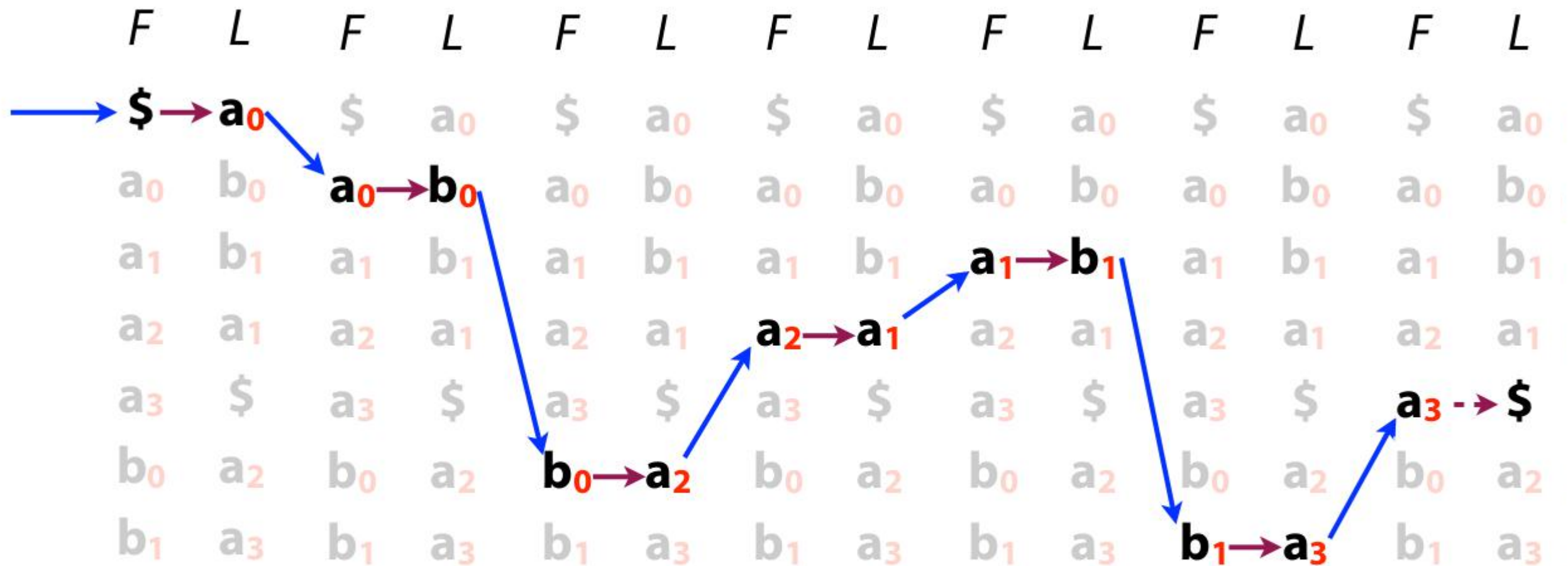
Kartojam  $b_1$ , gaunam  $a_3$

Kartojam  $a_3$ , gaunam  $\$$ , done Atvirkščiai aplankyti simboliai =  $a_3 b_1 a_1 a_2 b_0 a_0 \$ = T$



# Burrows-Wheeler Transformacija: grįžimas

Gryžimas vizualizuotas kitaip: BWT(T):



$T$ :  $a_3b_1a_1a_2b_0a_0\$$

# Burrows-Wheeler Transformacija: grįžimas

```
def rankBwt(bw):  
    ''' Given BWT string bw, return parallel list of B/ ranks. Also  
        returns tots: map from character to # times it appears. '''  
    tots = dict()  
    ranks = []  
    for c in bw:  
        if c not in tots: tots[c] = 0  
        ranks.append(tots[c])  
        tots[c] += 1  
    return ranks, tots
```

B-rangavimas ir simbolių  
skaičiavimas

```
def firstCol(tots):  
    ''' Return map from character to the range of eilutės prefixed by  
        the character. '''  
    first = {}  
    totc = 0  
    for c, count in sorted(tots.iteritems()):  
        first[c] = (totc, totc + count)  
        totc += count  
    return first
```

Pirmos eilutės struktūra

```
def reverseBwt(bw):  
    ''' Make T from BWT(T) '''  
    ranks, tots = rankBwt(bw)  
    first = firstCol(tots)  
    eilutėi = 0 # start in first eilutė  
    t = '%' # start with rightmost character  
    while bw[eilutėi] != '%':  
        c = bw[eilutėi]  
        t = c + t # prepend to answer  
        # jump to eilutė that starts with c of same rank  
        eilutėi = first[c][0] + ranks[eilutėi]  
    return t
```

Grįžimas

Python example:

<http://nbviewer.ipython.org/6860491>

# Burrows-Wheeler Transformacija

Kuo BWT naudingas duomenų kompresijai:

Išrūšiuoja simbolius pagal dešinių kontekstą - labiau suspaudžiama eilutė gaunama

Kaip vykdoma grįžtamoji transformacija:

Naudojant LF Atvaizdį, galima atkurti T nuo kairės pusės

Kaip galima panaudoti lyg sekos indeksą paieškai?

# FM Indeksas

FM Indeksas: indeksas apjungiantis BWT su keletu pagalbinių duomenų struktūrų.

“FM” verčiasi “Full-text Minute-space.”  
(išradėjų vardai Ferragina and Manzini)

Pagrindas -  $F$  ir  $L$  iš BWM:

$F$  laikomas labai parastai  
(1 integer per alfabeto simbolį)

$L$  - simbolių eilutė, kurią galima suspausti.

Labai mažai vietos!

$F$							$L$
\$	a	b	a	a	b	a	
a	\$	a	b	a	a	b	
a	a	b	a	\$	a	b	
a	b	a	\$	a	b	a	
a	b	a	a	b	a	\$	
b	a	\$	a	b	a	a	
b	a	a	b	a	\$	a	

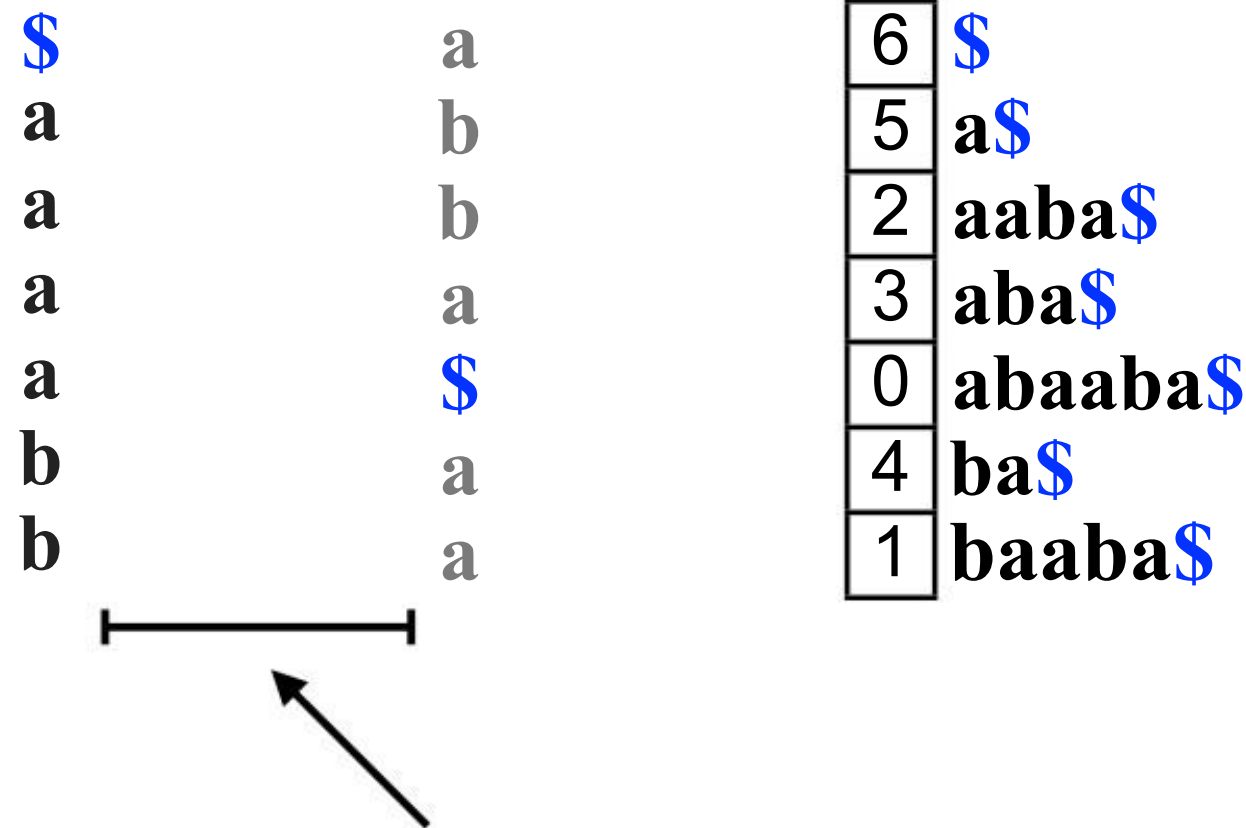
└──────────────────┘  
Not stored in indeksas

Paolo Ferragina, and Giovanni Manzini. "Opportunistic data structures with applications." *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE, 2000.



# FM Indeksas: užklausa

BWM susijęs su priesagų masyvu, bet nealima atlikti tiesioginės paieškos



Neturime šių stulpelių, “lengva” paieška negalima

# FM Indeksas: užklausa

Ieškom eilučių intervalo BWM(T) , kur  $P$  būtų priešdėlis.

Pradedam nuo trumpiausios  $P$  priesagos (pats dešinysis simbolis),  
ir iteratyviai ilginam presagą, kol intervalas tampa tuščias arba baigiasi  $P$ .

$P = \mathbf{ab}\mathbf{a}$

Easy to find all the  
eilutės beginning with  
 $\mathbf{a}$ , thanks to  $F$ 's  
simple structure

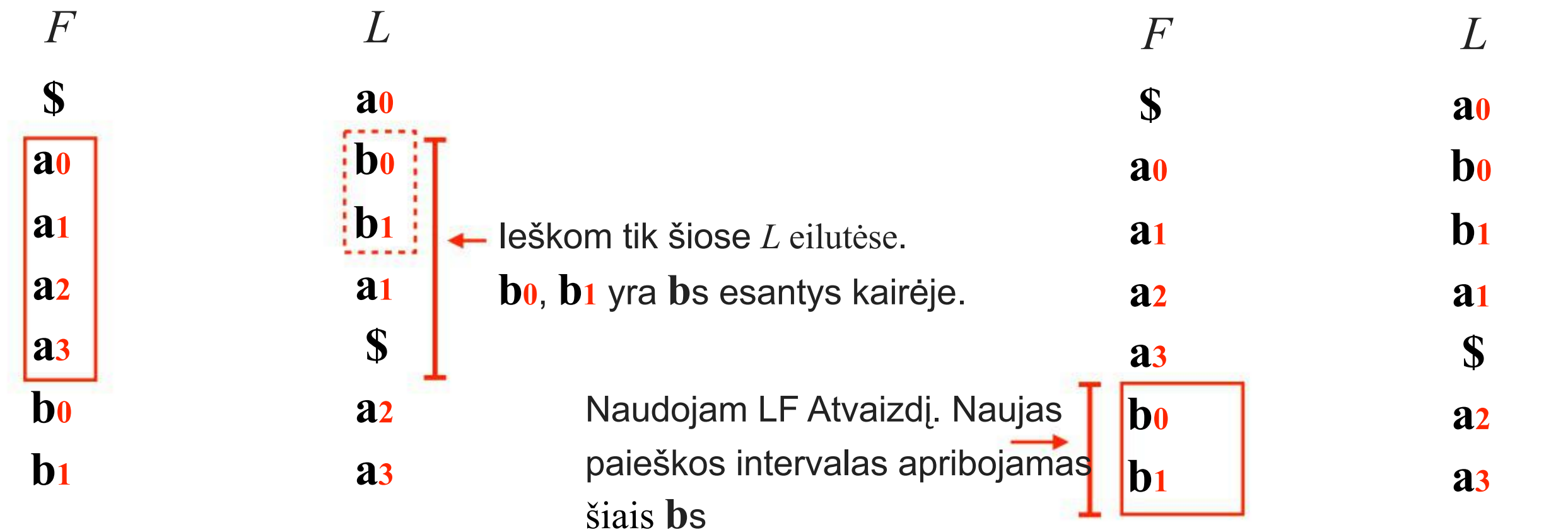
$F$	$L$
\$	$\mathbf{a3}$
$\mathbf{a0}$	$\mathbf{b1}$
$\mathbf{a1}$	$\mathbf{b0}$
$\mathbf{a2}$	$\mathbf{a1}$
$\mathbf{a3}$	\$
$\mathbf{b0}$	$\mathbf{a2}$
$\mathbf{b1}$	$\mathbf{a0}$

# FM Indeksas: užklausa

Turim eilutes prasidedančias **a**, tada ieškom eilučių, kurios prasideda **ba**

$P = \mathbf{ab}\mathbf{a}$

$P = \mathbf{a}\mathbf{b}\mathbf{a}$



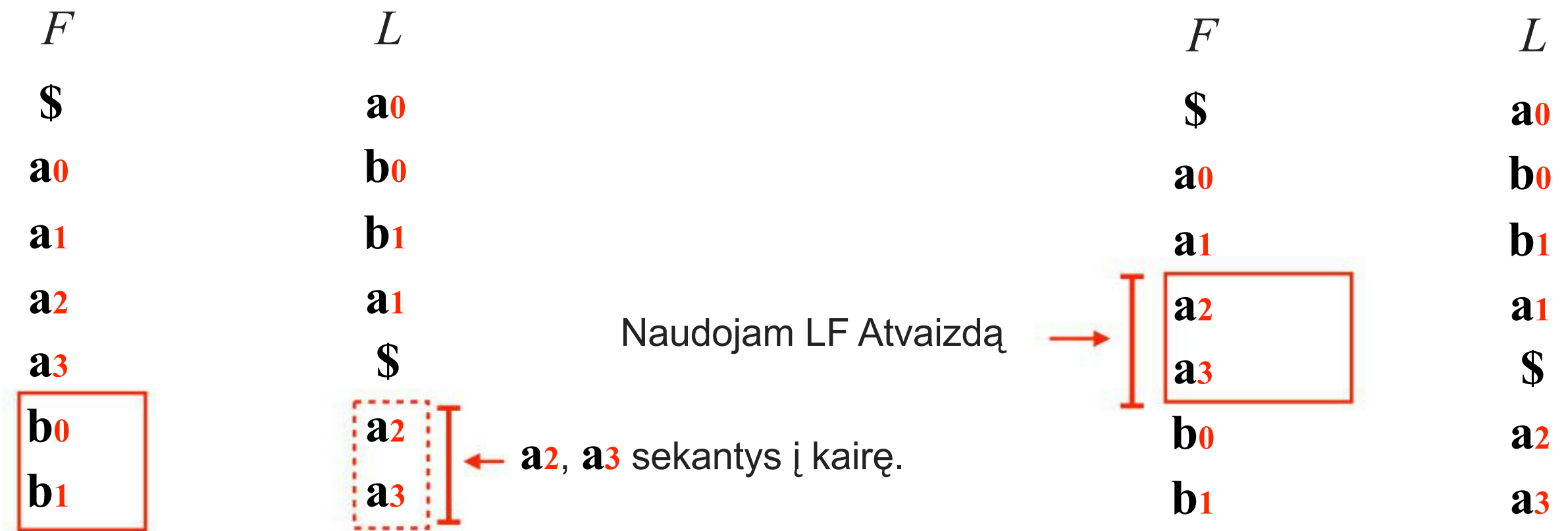
Dabar turime eilutes, kurių priešdėlis **ba**

# FM Indeksas: užklausa

Turime eilutes beprasidedančias **ba**, dabar ieškome eilutės prasidedančios **aba**

$P = \mathbf{a}ba$

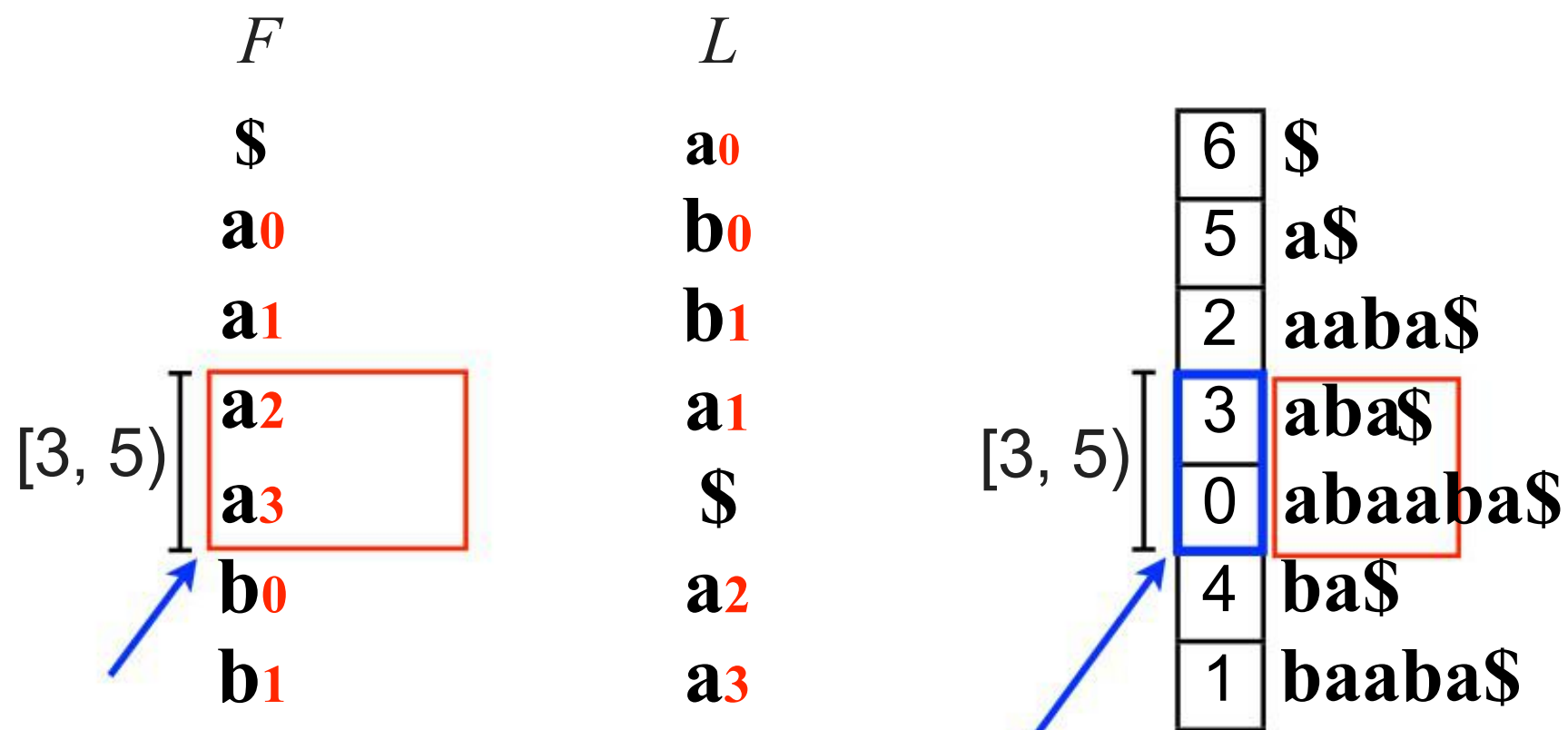
$P = \mathbf{aba}$



Dabar turim eilutes turinčias priešdėlį **aba**

# FM Indeksas: užklausa

$P = \text{aba}$  Jei naudotume priešdėlių masyvą, tai iš kart gautume ir poziciją  $[3, 5)$  sekoje  $T$



Where tai yra sekoje  
T?

Jei naudotume priešdėlių medį, kur tai yra.

# FM Indeksas: užklausa

Kai  $P$  nėra  $T$ , galiausiai neberandame sekančio į kairę simbolio:

$$P = \mathbf{bba}$$

eilutės su  
with **ba** priešdėliu

$F$
\$
<b>a0</b>
<b>a1</b>
<b>a2</b>
<b>a3</b>
<b>b0</b>
<b>b1</b>

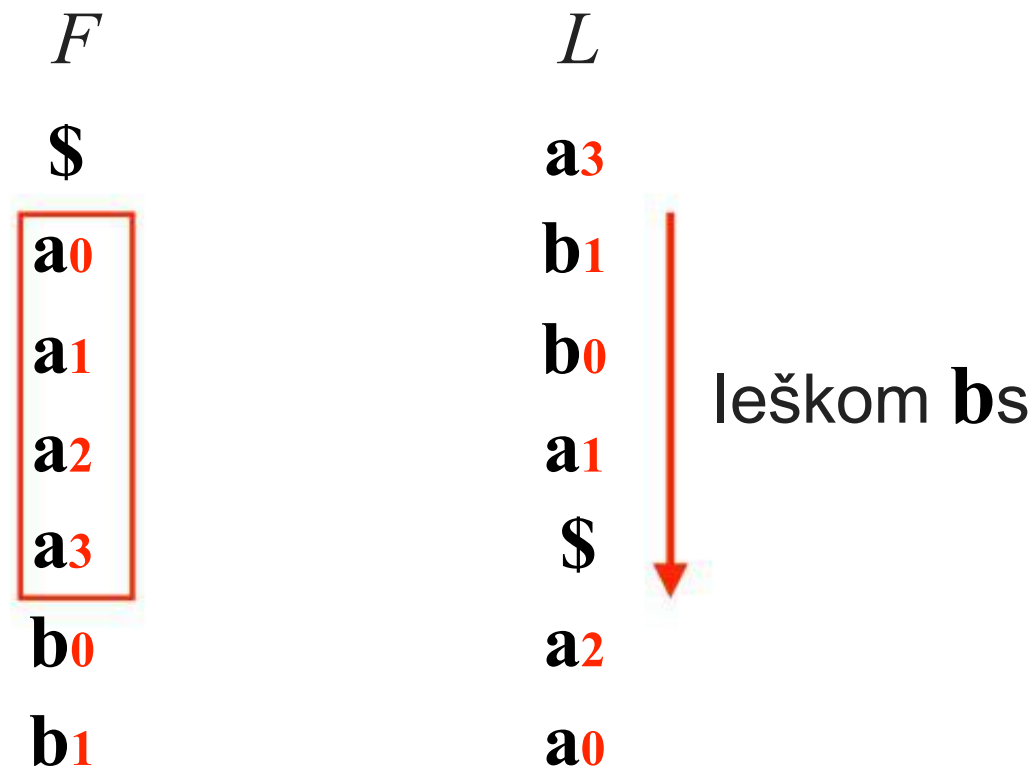
$L$
<b>a0</b>
<b>b0</b>
<b>b1</b>
<b>a1</b>
\$
<b>a2</b>
<b>a3</b>

← Nėrabs!

# FM Indeksas: užklausa

Jei skanuojam paprastai simbolio  $L$  sekoje, paieška gali būti labai lėta  $O(m)$

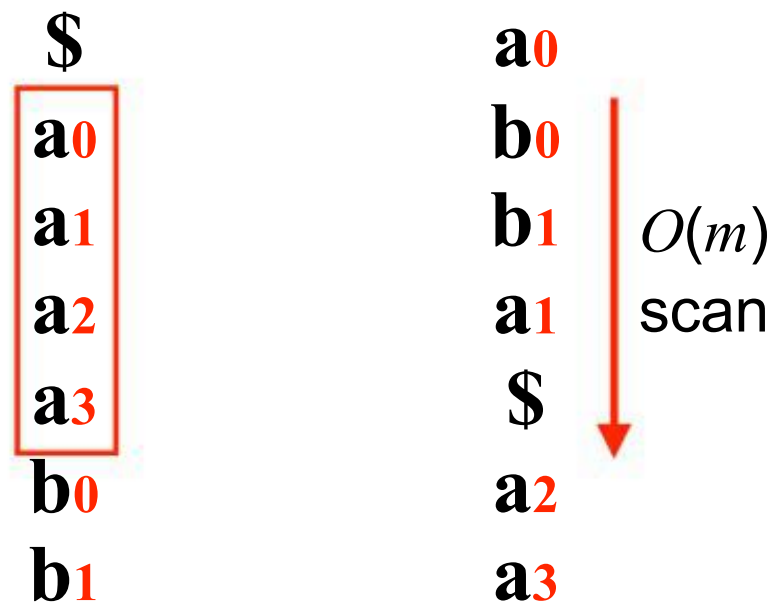
$P = \mathbf{ab}\mathbf{a}$





# FM Indeksas: Trūkumai

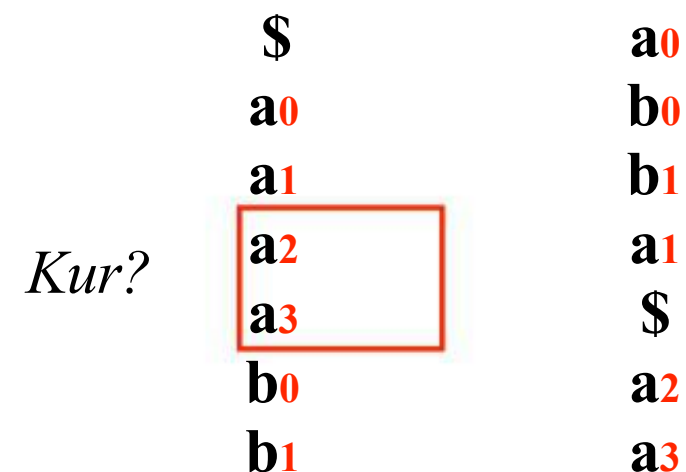
- (1) Ieškiojimas simbolio, kuris yra į kairę sekoje - lėtas



- (2) Rangų saugojimas užima daug vietos

```
def reverseBwt(bw):  
    ““ Make T from BWT(T) ““  
    ranks, tots = rankBwt(bw)  
    first = firstCol(tots)  
    eilutėi = 0  
    t = '%' m integers  
    while bw[eilutėi] != '%':  
        c = bw[eilutėi]  
        t = c + t  
        eilutėi = first[c][0] + ranks[eilutėi]  
    return t
```

- (3) Netinka pozicijai seoje nustatyti.



# FM Indeksas: greitas rangų skaičiavimas

Ar yra  $O(1)$  būdas nustatyti, kurie **bs** yra prieš **as**.

$F$	$L$
\$	<b>a</b> <sub>0</sub>
<b>a</b> <sub>0</sub>	<b>b</b> <sub>0</sub>
<b>a</b> <sub>1</sub>	<b>b</b> <sub>1</sub>
<b>a</b> <sub>2</sub>	<b>a</b> <sub>1</sub>
<b>a</b> <sub>3</sub>	\$
<b>b</b> <sub>0</sub>	<b>a</b> <sub>2</sub>
<b>b</b> <sub>1</sub>	<b>a</b> <sub>3</sub>

Idėja: iš anksto apskaičiuojam  
# **as**, **bs** , kuris yra  $L$  iki  
kiekvienos eilutės:

$F$	$L$
\$	<b>a</b>
<b>a</b>	<b>b</b>
<b>a</b>	<b>b</b>
<b>a</b>	<b>a</b>
<b>a</b>	\$
<b>b</b>	<b>a</b>
<b>b</b>	<b>a</b>

*Tally*

**a b**

1	0
1	1
1	2
2	2
2	2
3	2
4	2

Gaunam, kad **b**<sub>0</sub> ir **b**<sub>1</sub>  
yra  $L$  šiame intervale.

# FM Indeksas: greitas rangų skaičiavimas

Kita idėja: iš anksto apskaičiuojam # **as**, **bs**  $L$  iki tam tikros eilutės tik kai kurioms eilutėms (kas 5-tai) checkpoint'ai...

		<i>Tally</i>		
<i>F</i>	<i>L</i>	<b>a</b>	<b>b</b>	
\$	a	1	0	← prastai analizuojam
a	b			
a	b			
a	a			
a	\$			← Oops: nėra duomenų
b	a	3	2	← Einam iki checkpointo ir atsiskaičiuojam vertę.
b	a			

# FM Indeksas: Trūkumai

(1) Ieškiojimas simbolio, kuris yra į kairę sekoje - lėtas

\$  
**a0**  
**a1**  
**a2**  
**a3**  
b0  
b1

**a0**  
**b0**  
**b1**  
**a1**  
\$  
**a2**  
**a3**

$O(m)$   
scan

(2) Rangų saugojimas užima daug vietos

$m$   
integers

```
def reverseBwt(bw):  
    """ Make T from BWT(T) """  
    ranks, tots = rankBwt(bw)  
    first = firstCol(tots)  
    eilutėi = 0  
    t = ''  
    while bw[eilutėi] != '$':  
        c = bw[eilutėi]  
        t = c + t  
        eilutėi = first[c][0] + ranks[eilutėi]  
    return t
```

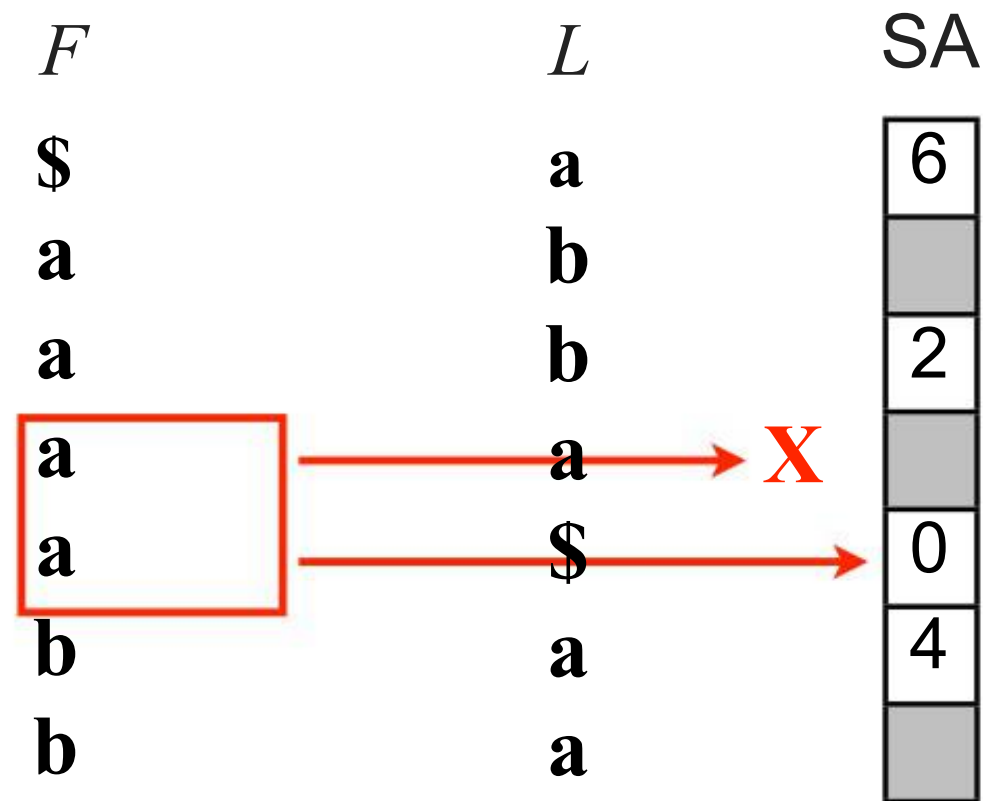
(3) Netinka pozicijai sejoje nustatyti.

*Kur?*

\$	<b>a0</b>
<b>a0</b>	<b>b0</b>
<b>a1</b>	<b>b1</b>
<b>a2</b>	<b>a1</b>
<b>a3</b>	\$
b0	<b>a2</b>
b1	<b>a3</b>

# FM Indeksas: pozicijų radimas

Idėja: laikome dalį priesaos medžio įrašų



# FM Indeksas: pozicijų radimas

LF Atvaizdis rodo, kad **a** 3-čios eilutės pabaigoje atitinka  
... **a** antros eilutės pabaigoje, kurios pozicija T yra išsaugota.

