BWT for repetitive texts, part 3: Toehold lemma setup

Ben Langmead

Locate queries

	Count		Locate	
	Space	Time	Space	Time
FM Index (2000)	$O(n)$	$O(m)$	$O(n)$	$O(m + \text{occ})$
RLFM Index (2005)	$O(r)$	$O(m)$	$O(n)$	$O(m + \text{occ})$
r-index (2018)	$O(r)$	$O(m)$	$O(r)$	$O(m + \text{occ})$

n = reference length, m = query length,
r = # BWT runs

(bounds simplified)

Locate queries

How?

	Count		Locate	
	Space	Time	Space	Time
FM Index (2000)	$O(n)$	$O(m)$	$O(n)$	$O(m + \text{OCC})$
RLFM Index (2005)	$O(r)$	$O(m)$	$O(n)$	$O(m + \text{OCC})$
r-index (2018)	$O(r)$	$O(m)$	$\boxed{O(r)}$	$O(m + \text{OCC})$

n = reference length, m = query length,
r = # BWT runs

(bounds simplified)

r-index: Gagie T, Navarro G, and Prezza P. Optimal-time text indexing in BWT-runs bounded space. Proceedings of 29th SODA, ACM-SIAM. 2018. pp1459—1477.

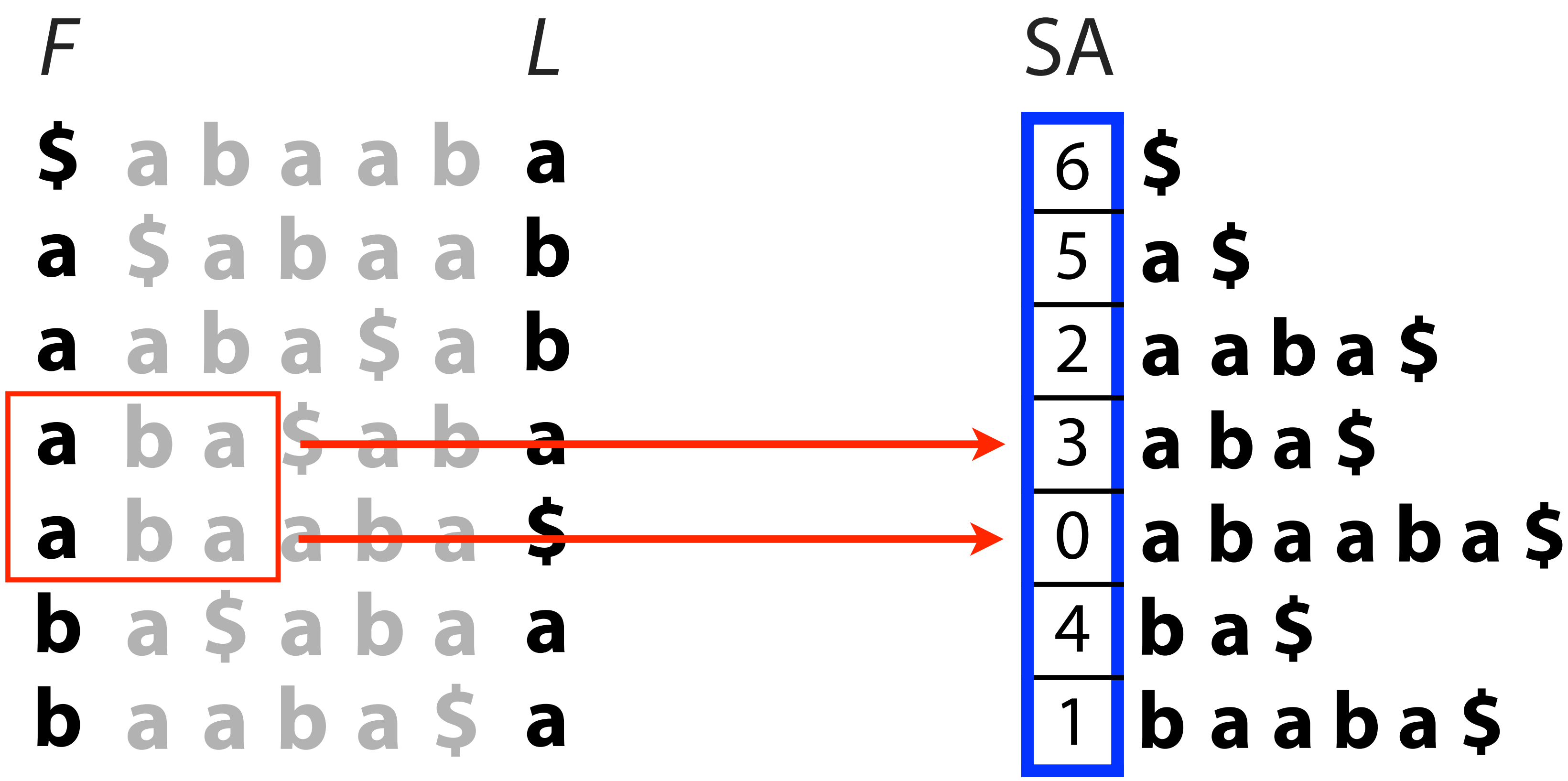
Locate query & SA samples

A precomputed "suffix array" stores the answers…

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**

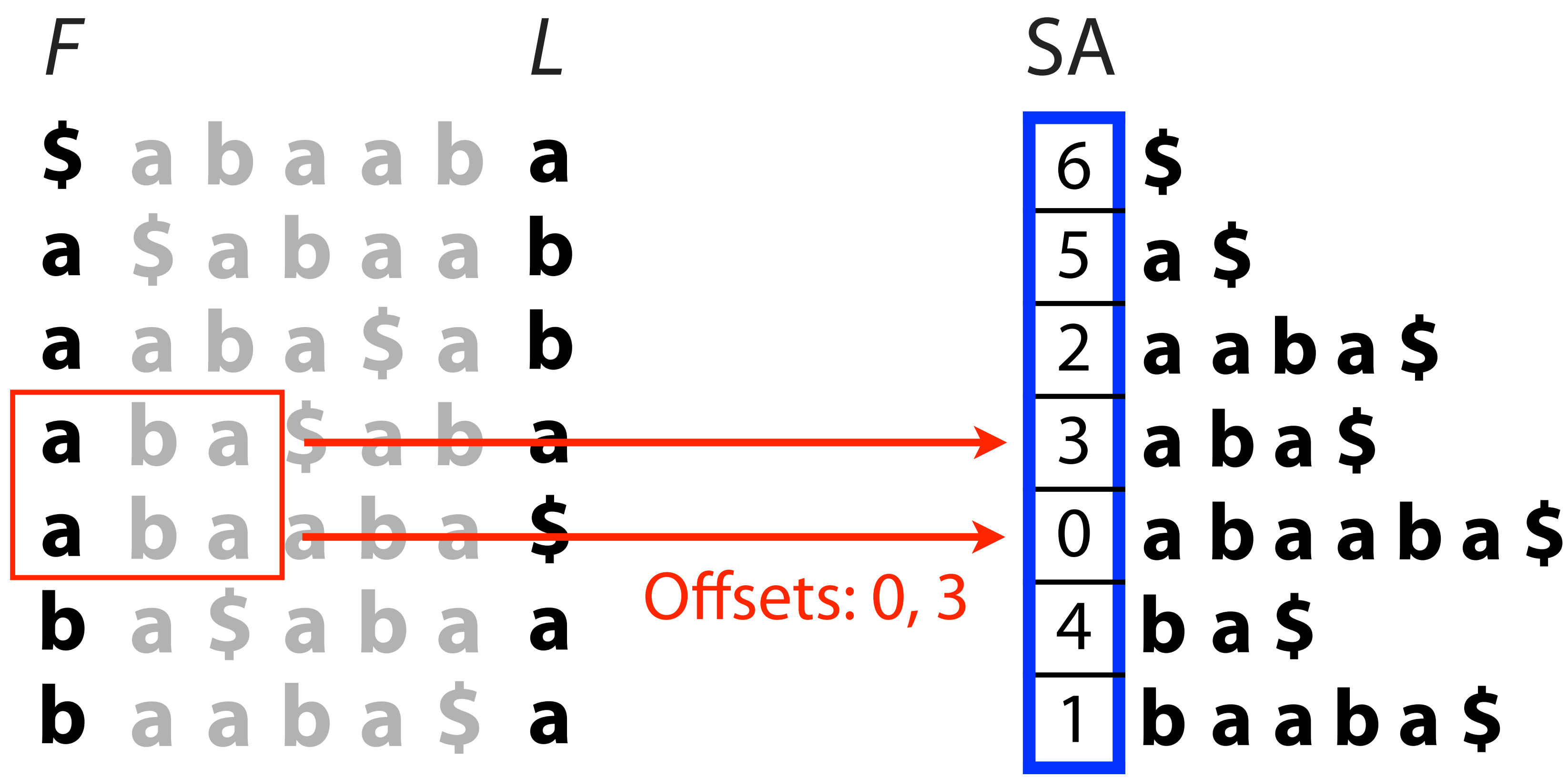
Locate query & SA samples

A precomputed "suffix array" stores the answers…



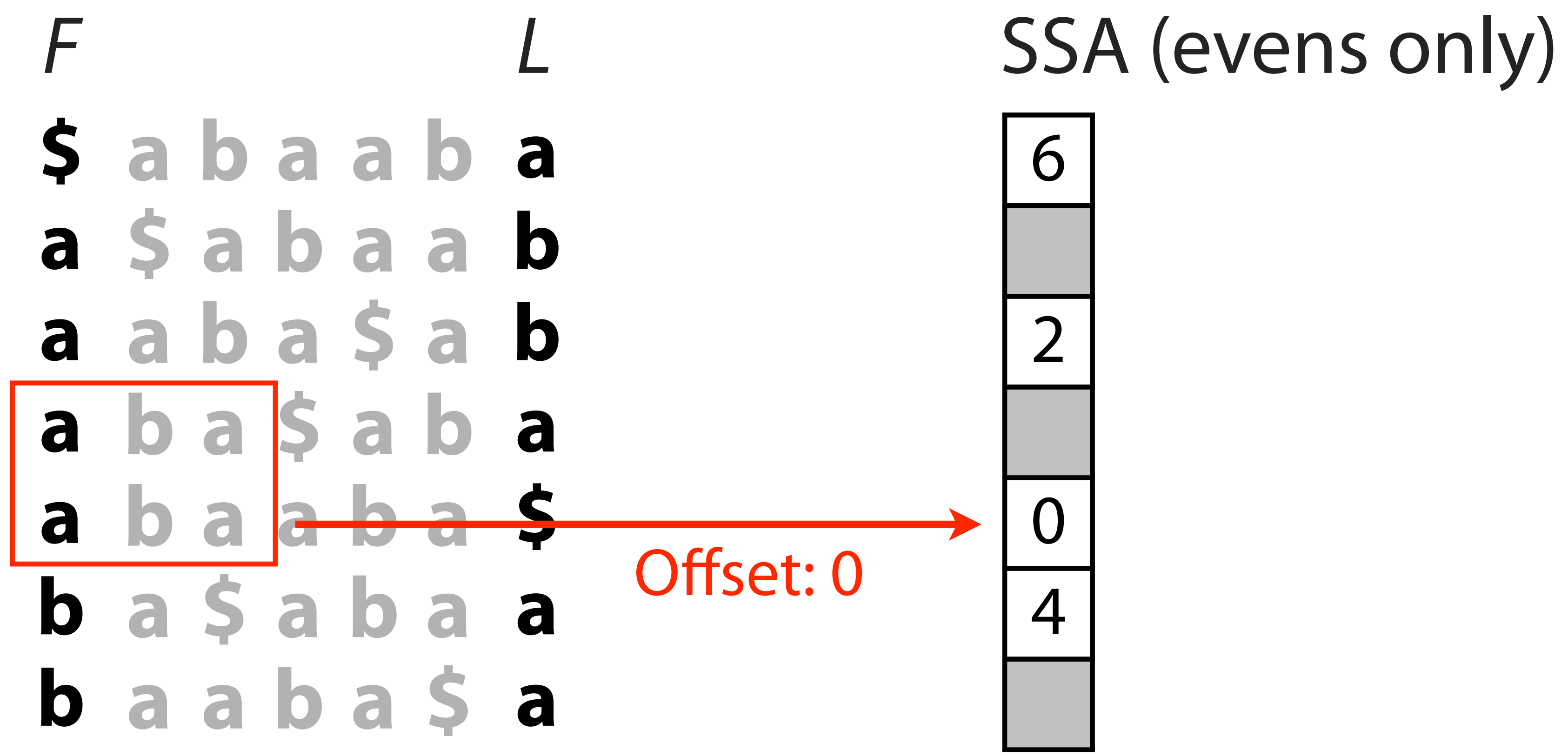
Locate query & SA samples

A precomputed "suffix array" stores the answers…

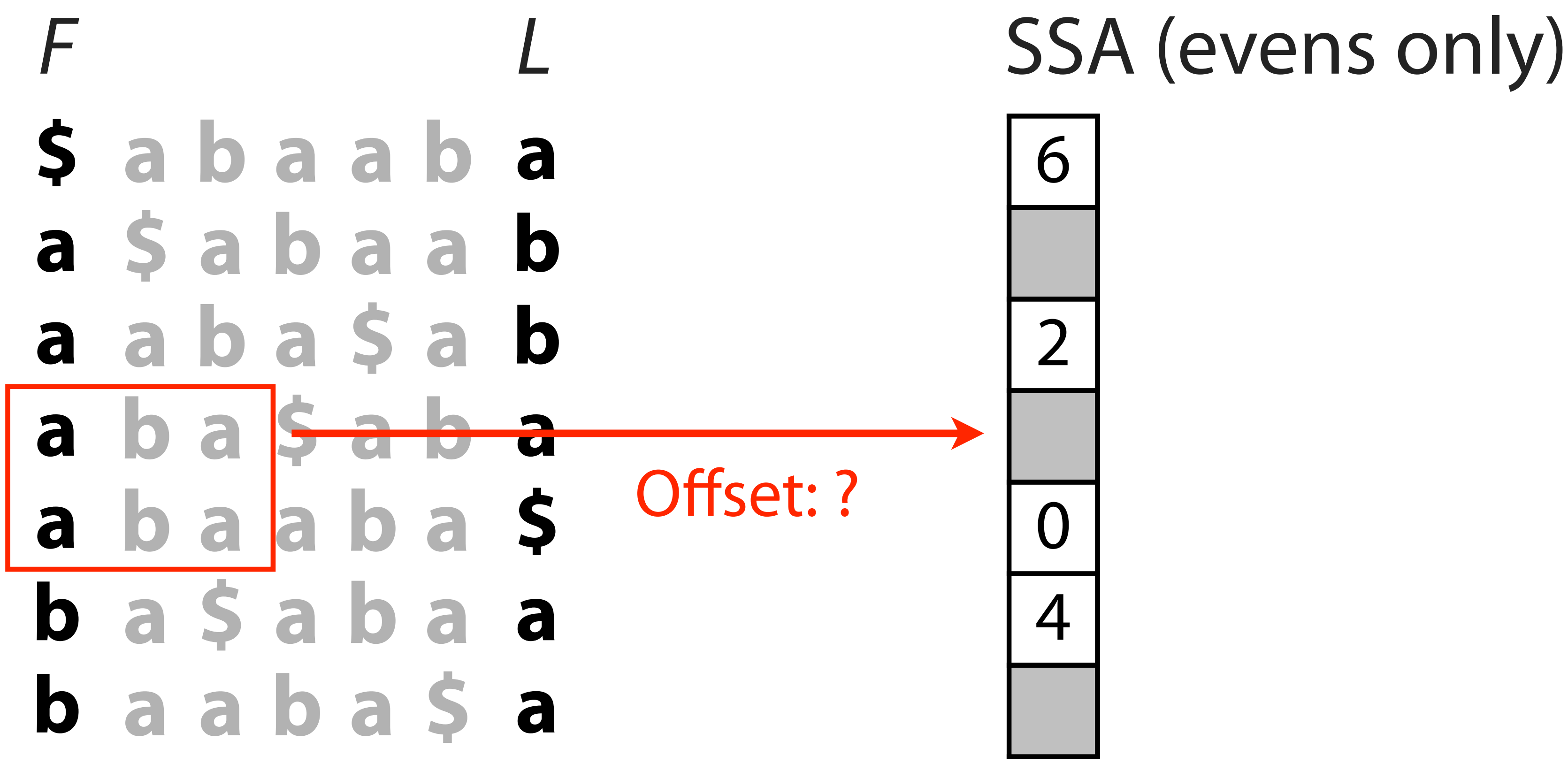


n integers is too big, so we **sample** instead

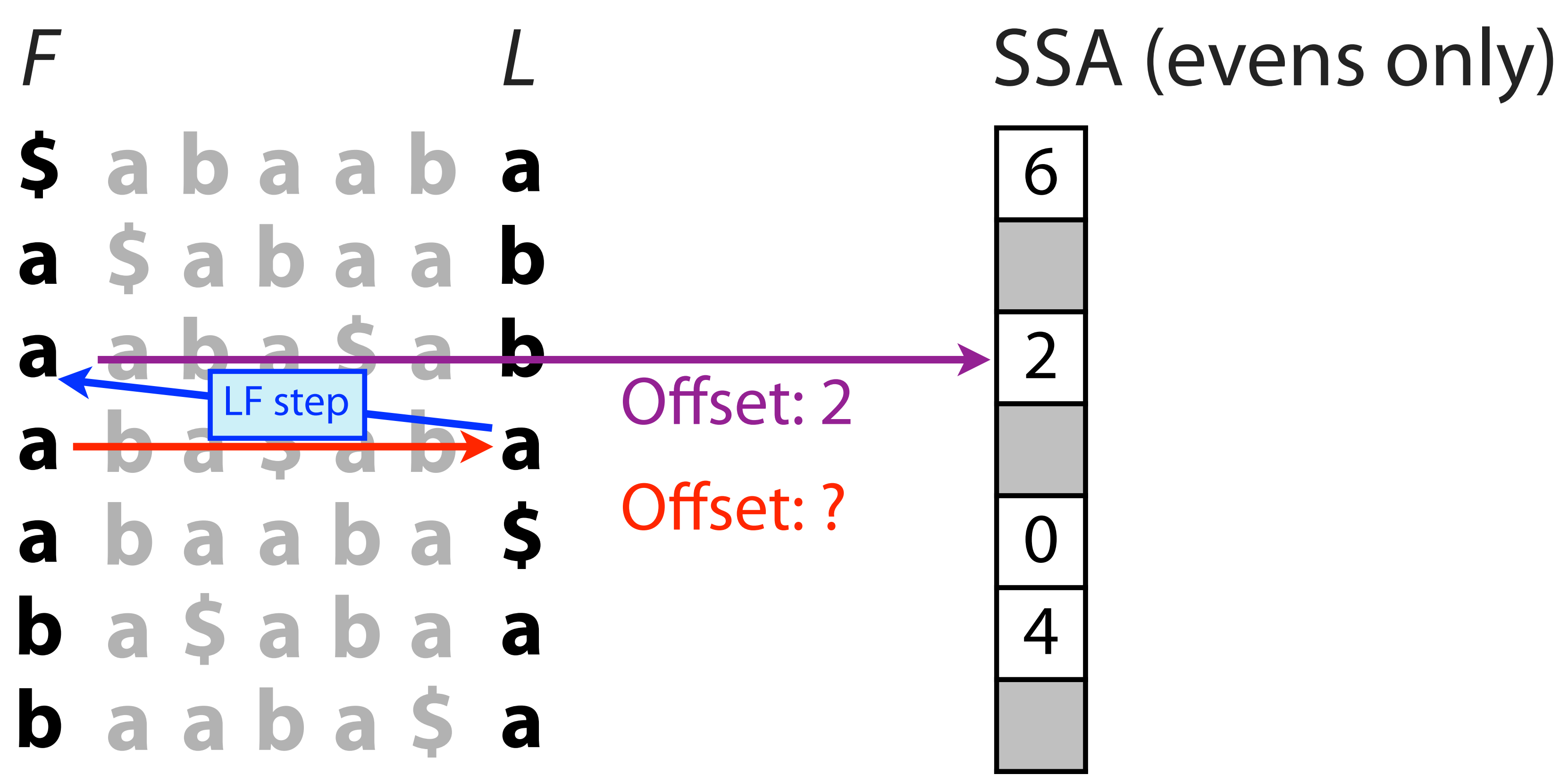
Locate query & SA samples



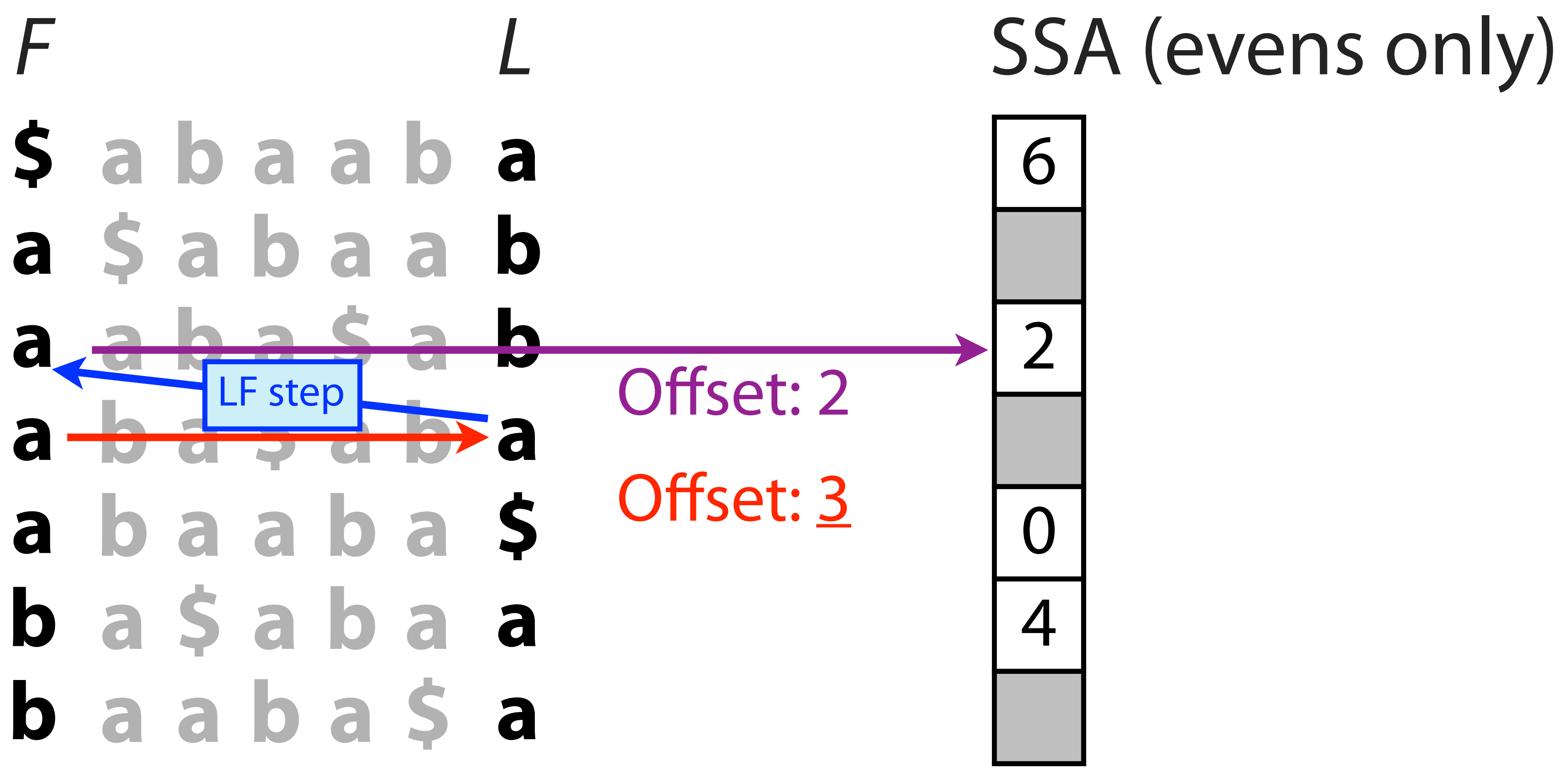
Locate query & SA samples



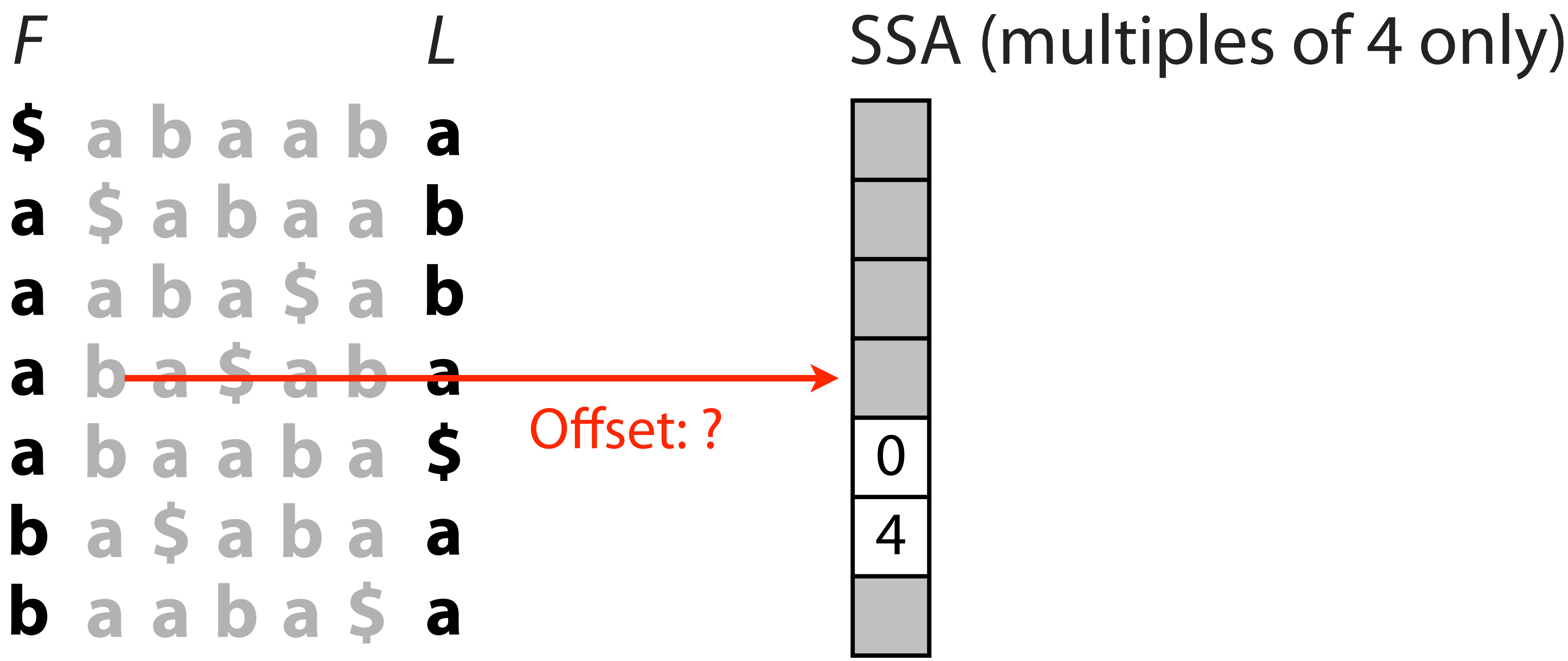
Locate query & SA samples



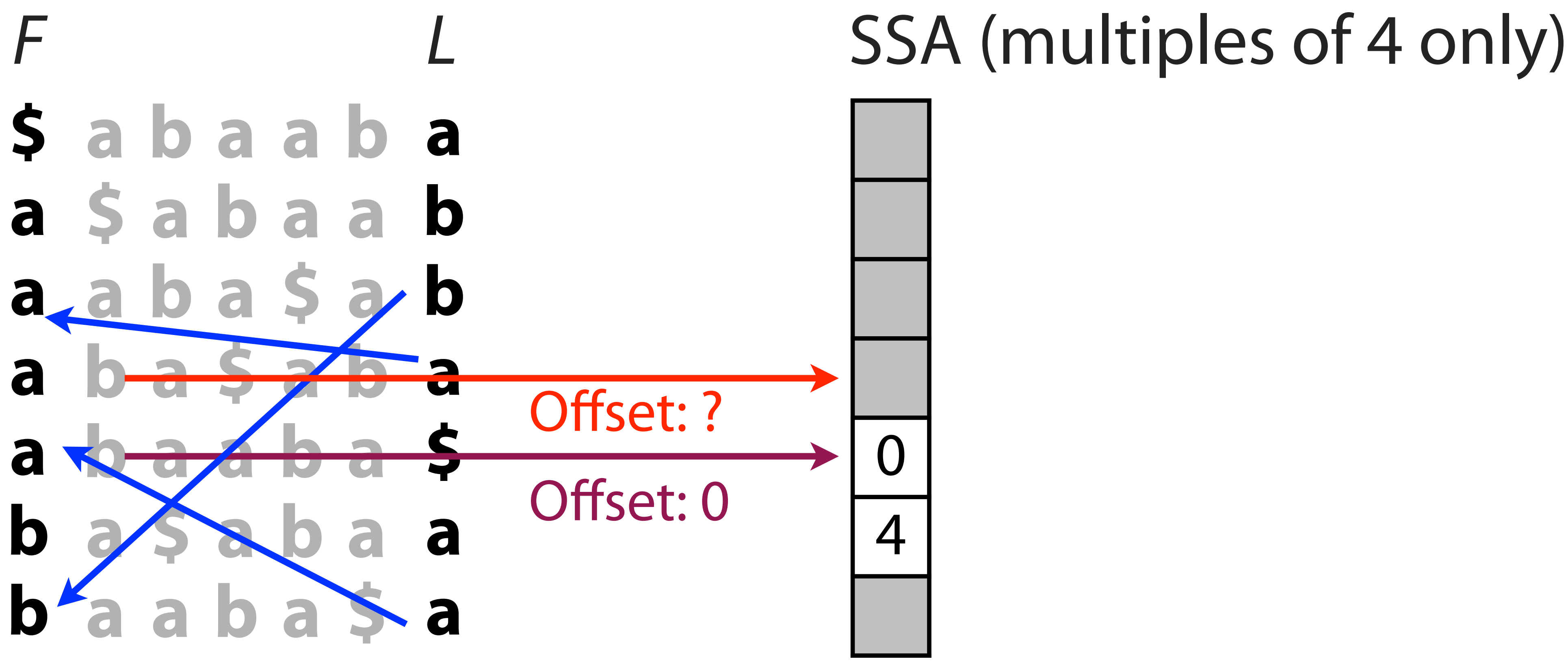
Locate query & SA samples



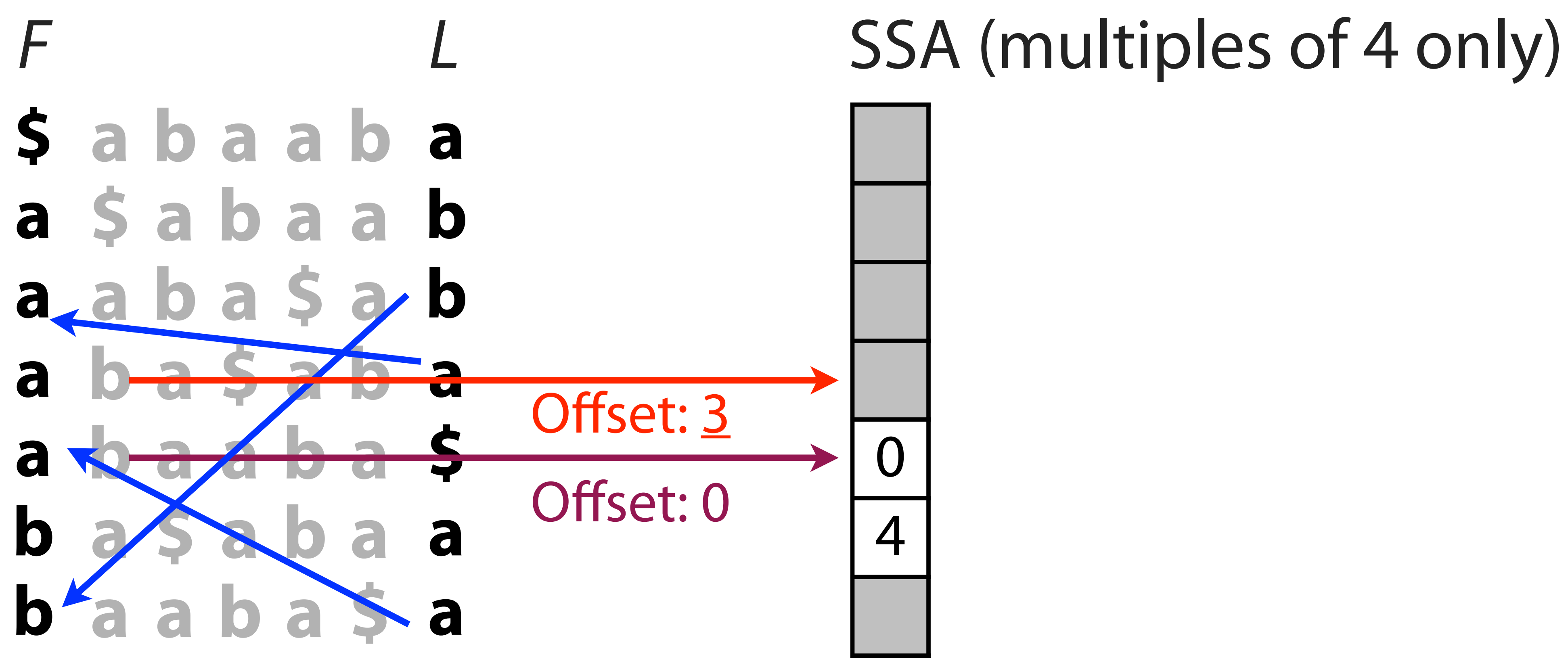
Locate query & SA samples



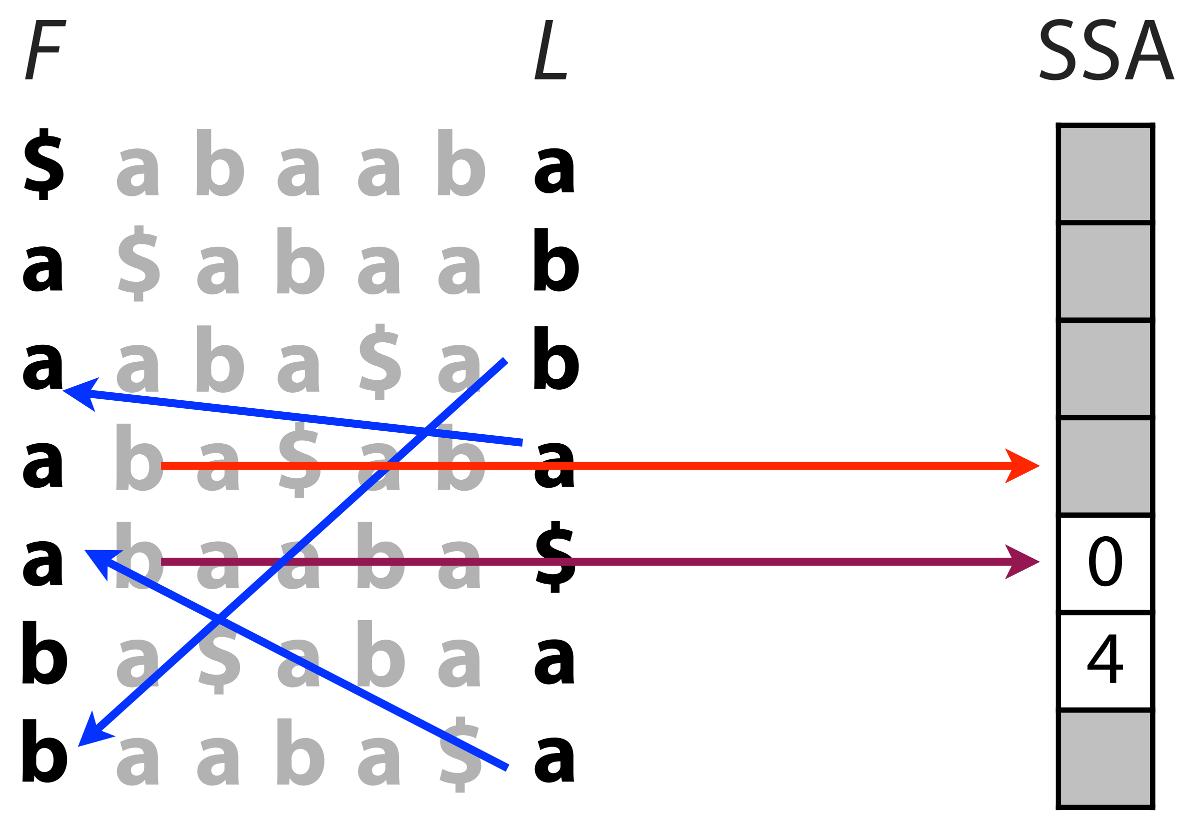
Locate query & SA samples



Locate query & SA samples



Locate query & SA samples



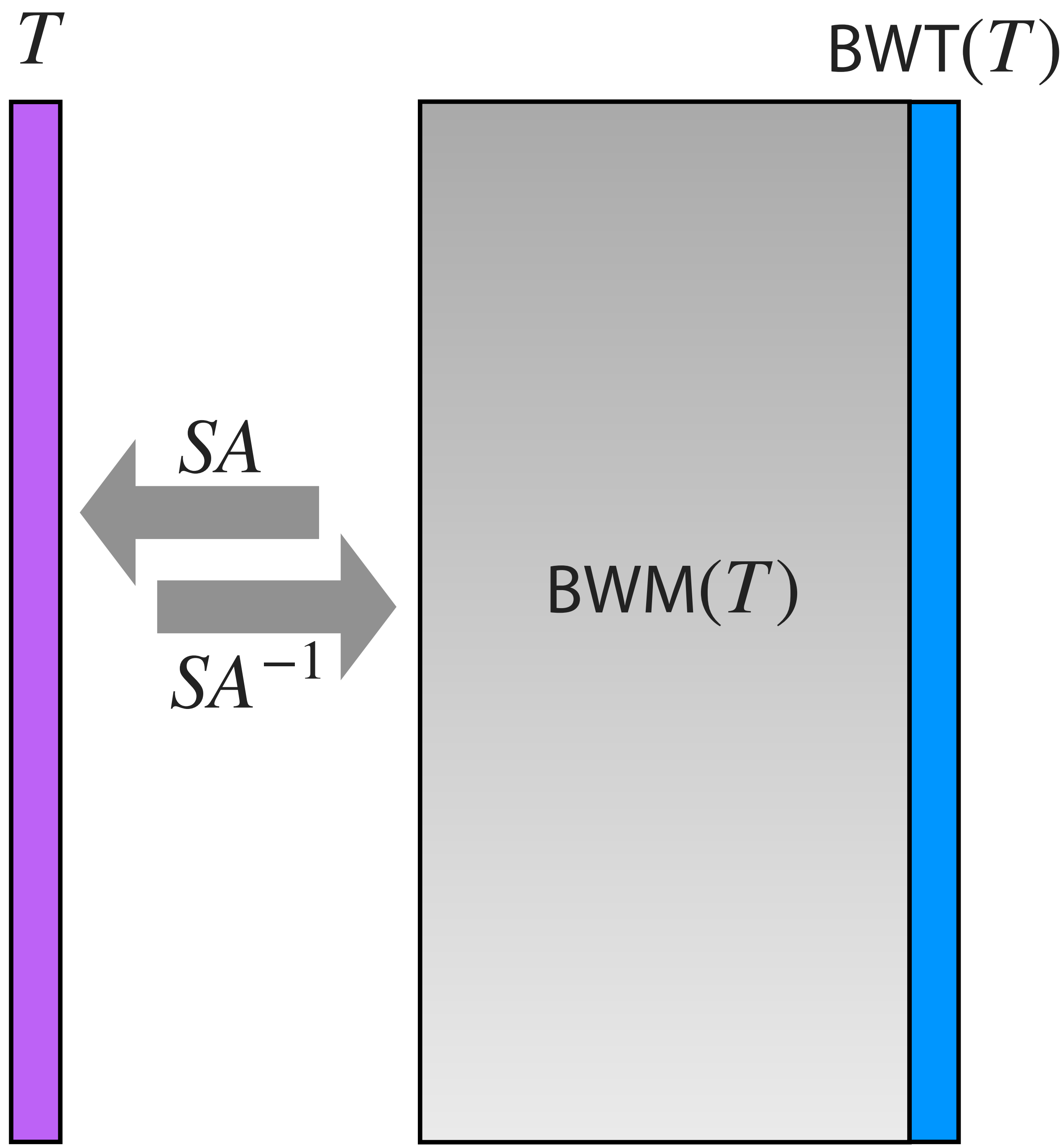
If we we store SAs for multiples of constant c, max LF steps is c, but SSA sample takes $O(n/c) = O(n)$ space

Or let c be a function of n, like $\sqrt{n} \quad \rightarrow \quad O(\sqrt{n})$ space & time

For repetitive texts, $O(r)$ space is much better, if queries can be made efficient...

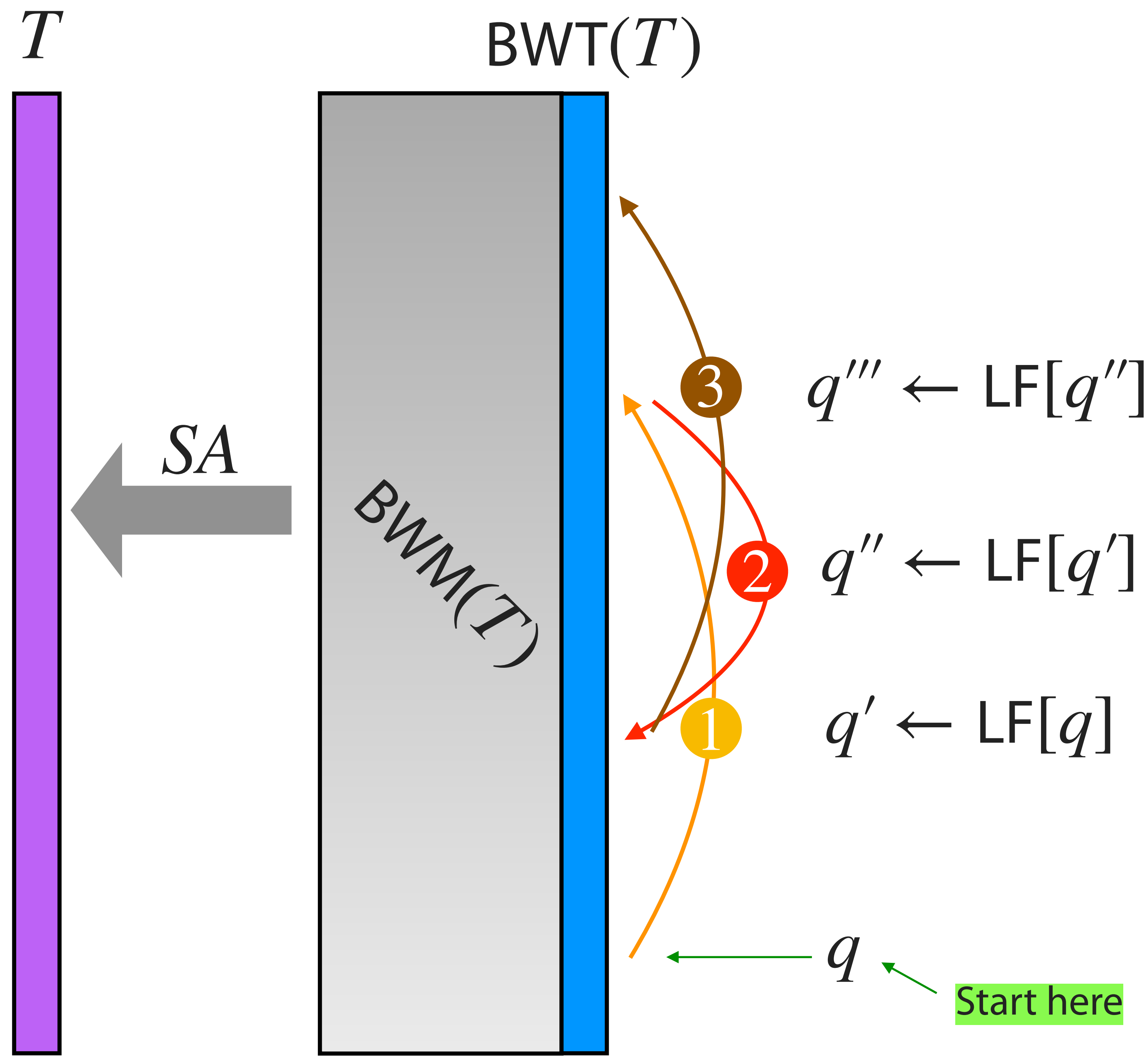
Fundamental movements

"T space"

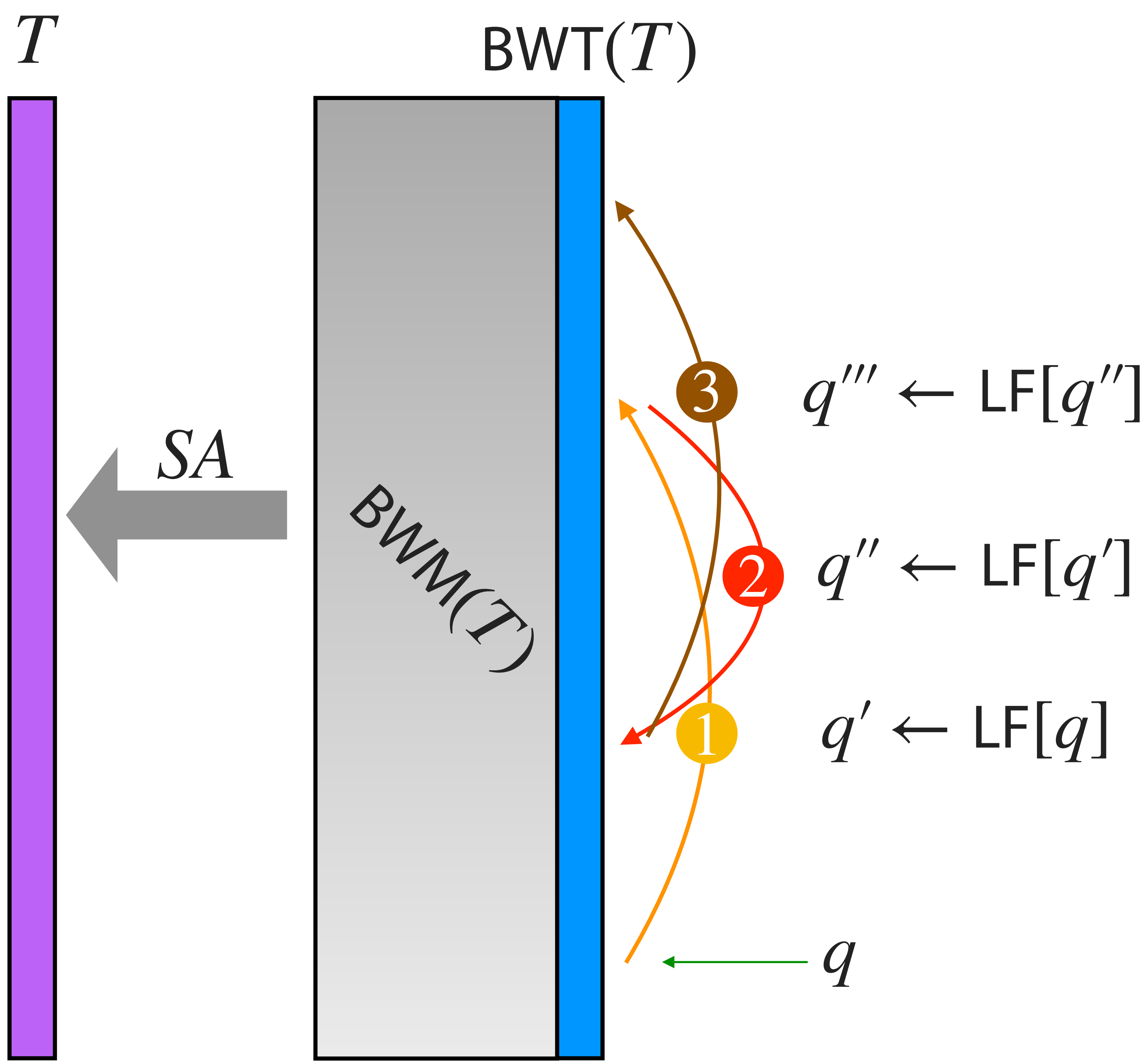


"Lex space"

Fundamental movements: LF

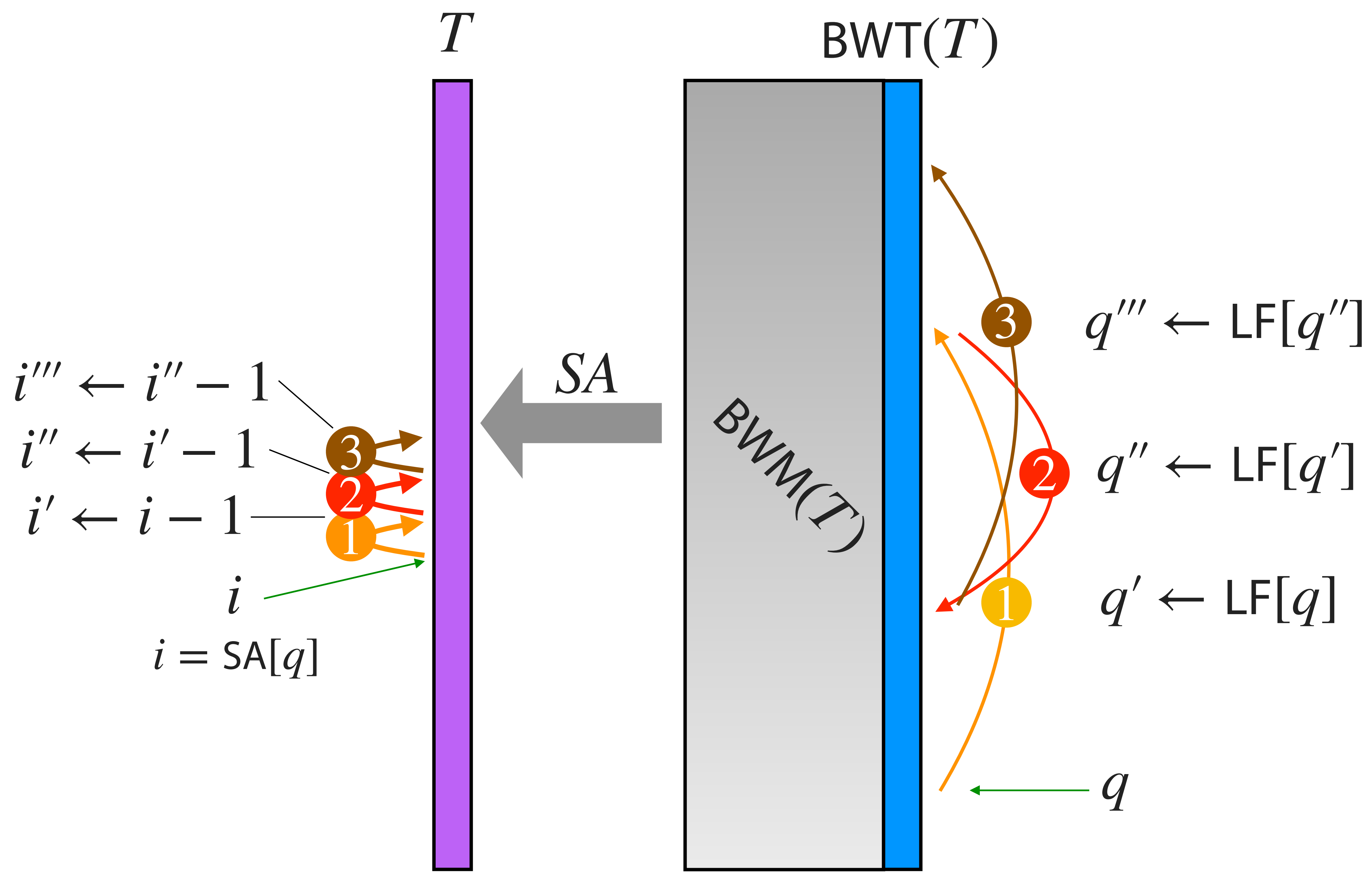


Fundamental movements: LF



An LF step (in lex space)... $\mathsf{SA}[\mathsf{LF}[q]] = \mathsf{SA}[q] - 1$ ...moves to the left by 1 in T space

Fundamental movements: LF

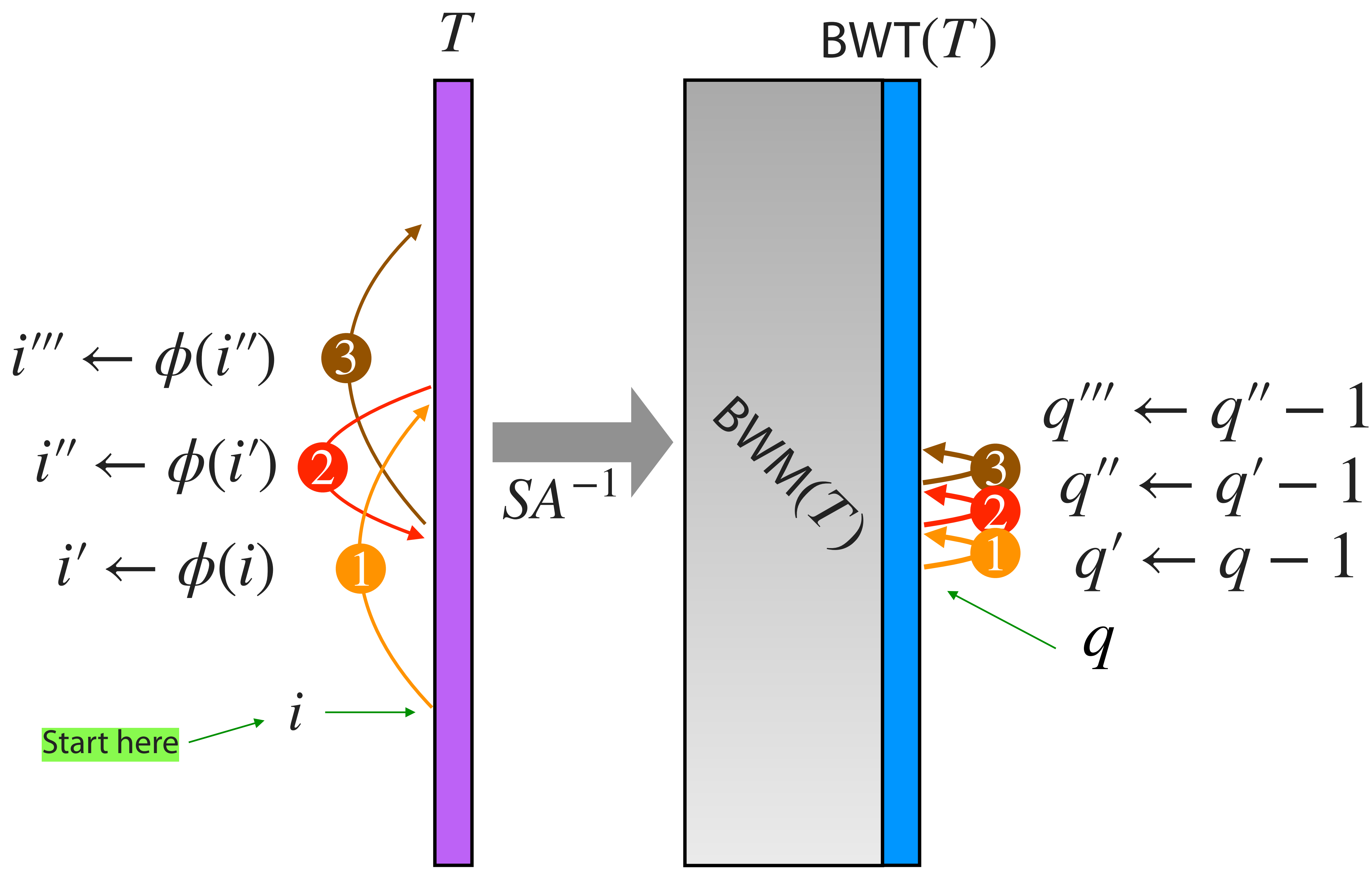


An LF step (in lex space)...

$$\text{SA}[\text{LF}[q]] = \text{SA}[q] - 1$$

...moves to the left by 1 in T space

Fundamental movements: ϕ



$$\mathrm{SA}^{-1}[\phi(i)] = \mathrm{SA}^{-1}[i] - 1$$

Fundamental movements: "Runny LF"

When T is repetitive, LF can be thought of as permuting **runs**

I.e. LF mapping is "runny"

T `row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$`

L `trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___`

F `$____________aaabbbooooooooooooooorrrrrrrrrrrrtttuuuwwwwwwwwwyyy`

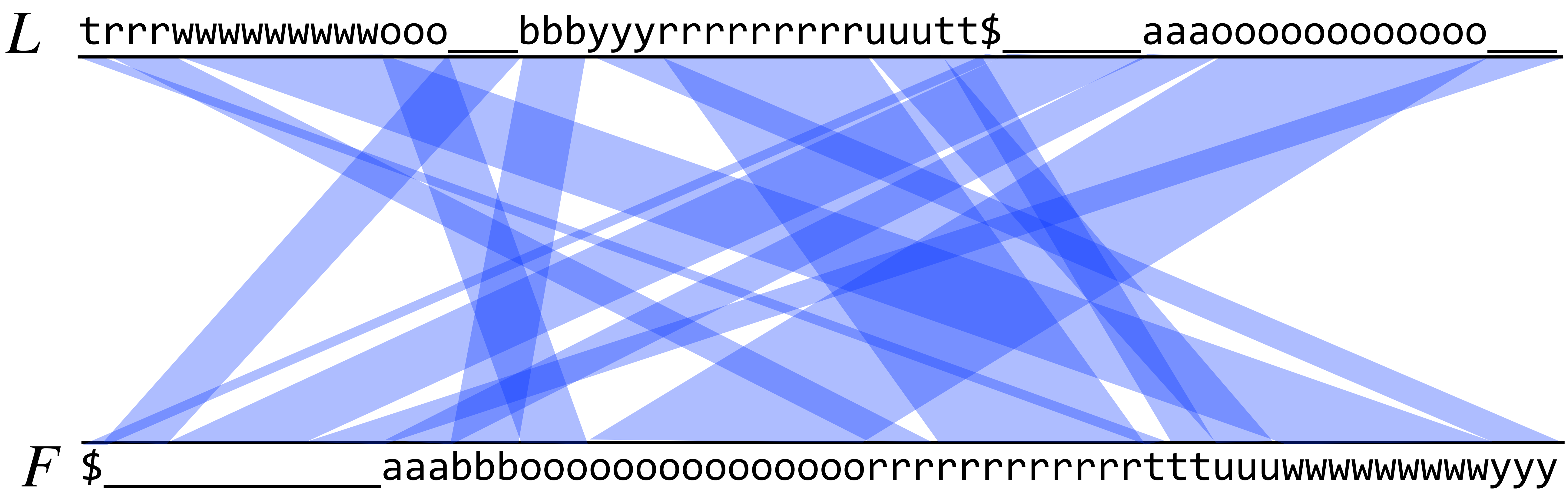
Fundamental movements: "Runny LF"

When T is repetitive, LF can be thought of as permuting **runs**

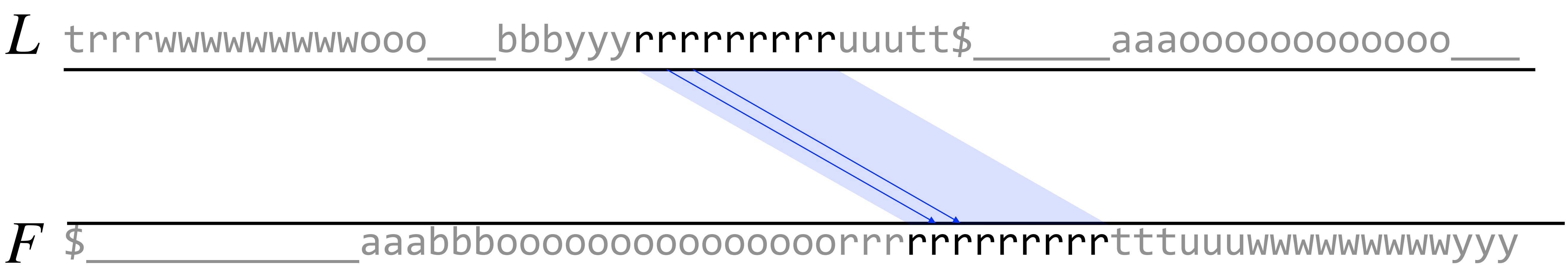
I.e. LF mapping is "runny"

```
T  row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$

L  trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___

F  $____________aaabbbooooooooooooooorrrrrrrrrrrrtttuuuwwwwwwwwwyyy
```

Runny LF


L
trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___
F
$____________aaabbbooooooooooooooorrrrrrrrrrrrtttuuuwwwwwwwwwyyy

Runny LF principle



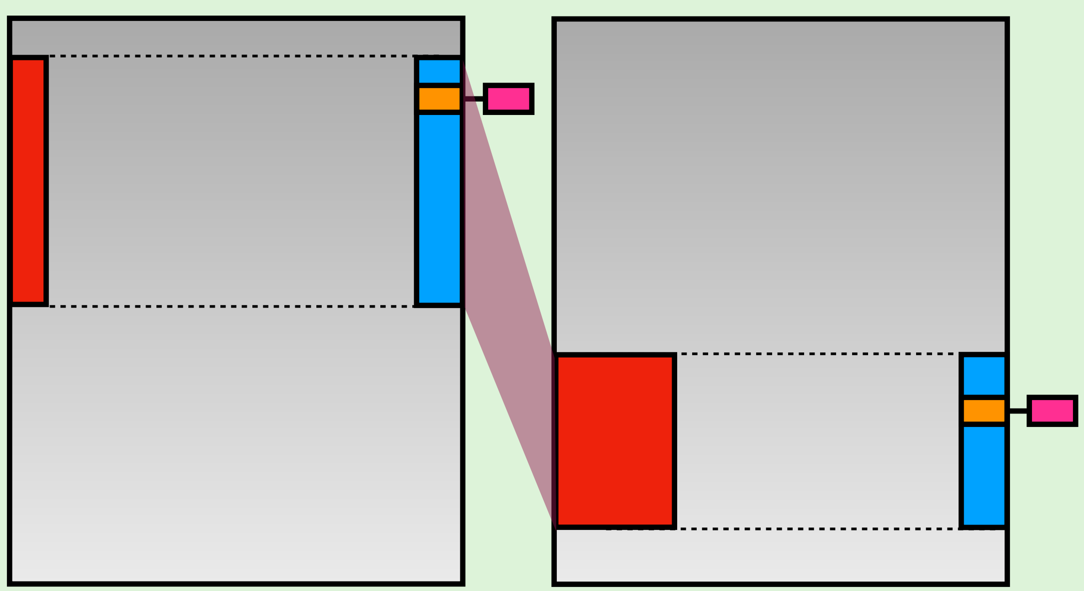
If $\mathrm{BWT}[q] = \mathrm{BWT}[q+1]$, then $\mathrm{LF}[q+1] = \mathrm{LF}[q] + 1$

Two positions in the same BWT run move through LF in a **parallel** fashion

O(r) locate queries

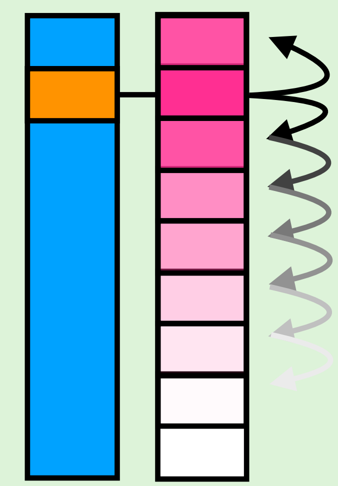
Algorithm has two components:

1



We always have a "toehold"

2



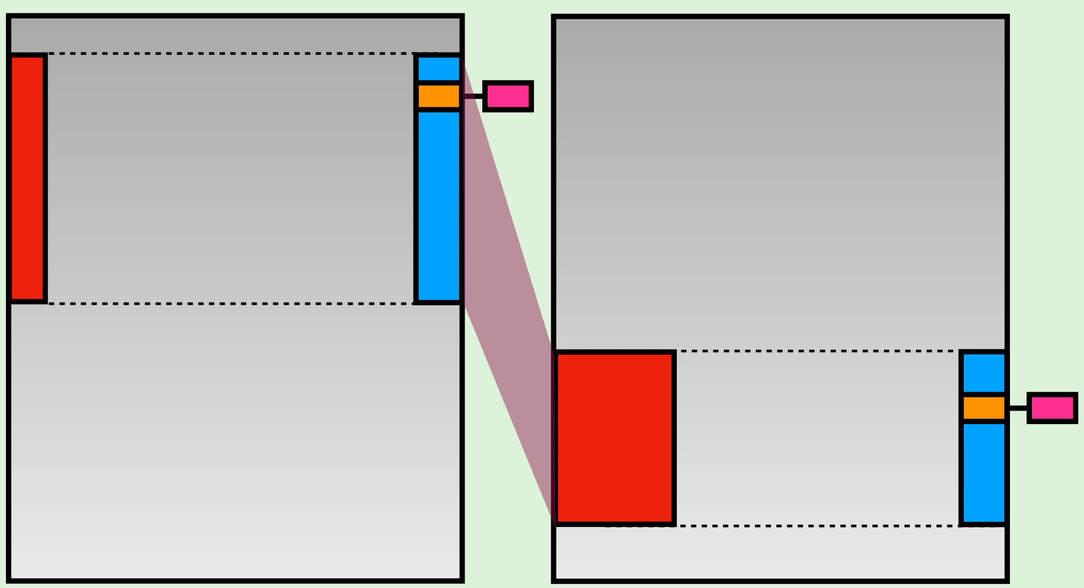
We can get the rest from the toehold

r-index: Gagie T, Navarro G, and Prezza P. Optimal-time text indexing in BWT-runs bounded space. Proceedings of 29th SODA, ACM-SIAM. 2018. pp1459—1477.

O(r) locate queries

Algorithm has two components:

1



We always have a "toehold"

2

We can get the rest from the toehold

r-index: Gagie T, Navarro G, and Prezza P. Optimal-time text indexing in BWT-runs bounded space. Proceedings of 29th SODA, ACM-SIAM. 2018. pp1459—1477.

Locate query & SA samples

T row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$

L trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___

Samples at run heads

Locate query & SA samples

T row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$

L trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___

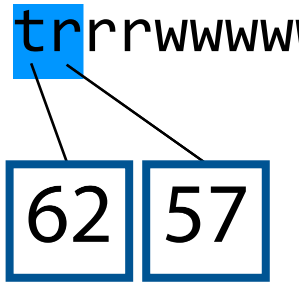
62

Samples at run heads

Locate query & SA samples

T `row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$`

L `trrrwwwwwwwwwoooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___`



Samples at run heads

Locate query & SA samples

T `row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$`

L `trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___`

62 57 44

Samples at run heads

Locate query & SA samples

```
T   row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$
L   trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___
```

62	57	44	60	58	59	54	42	56	41	63	45	61	55	53

Samples at run heads

Locate query & SA samples

T row_row_row_your_boatrow_row_row_your_boatrow_row_row_your_boat$

L trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___

62 57 44 60 58 59 54 42 56 41 63 45 61 55 53

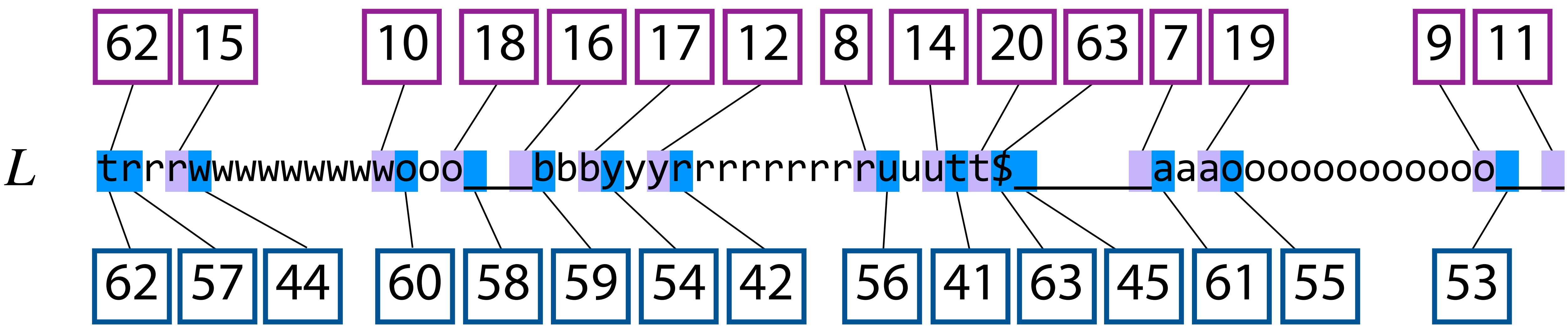
Samples at run heads

L trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___

62 15 10 18 16 17 12 8 14 20 63 7 19 9 11

Samples at run tails

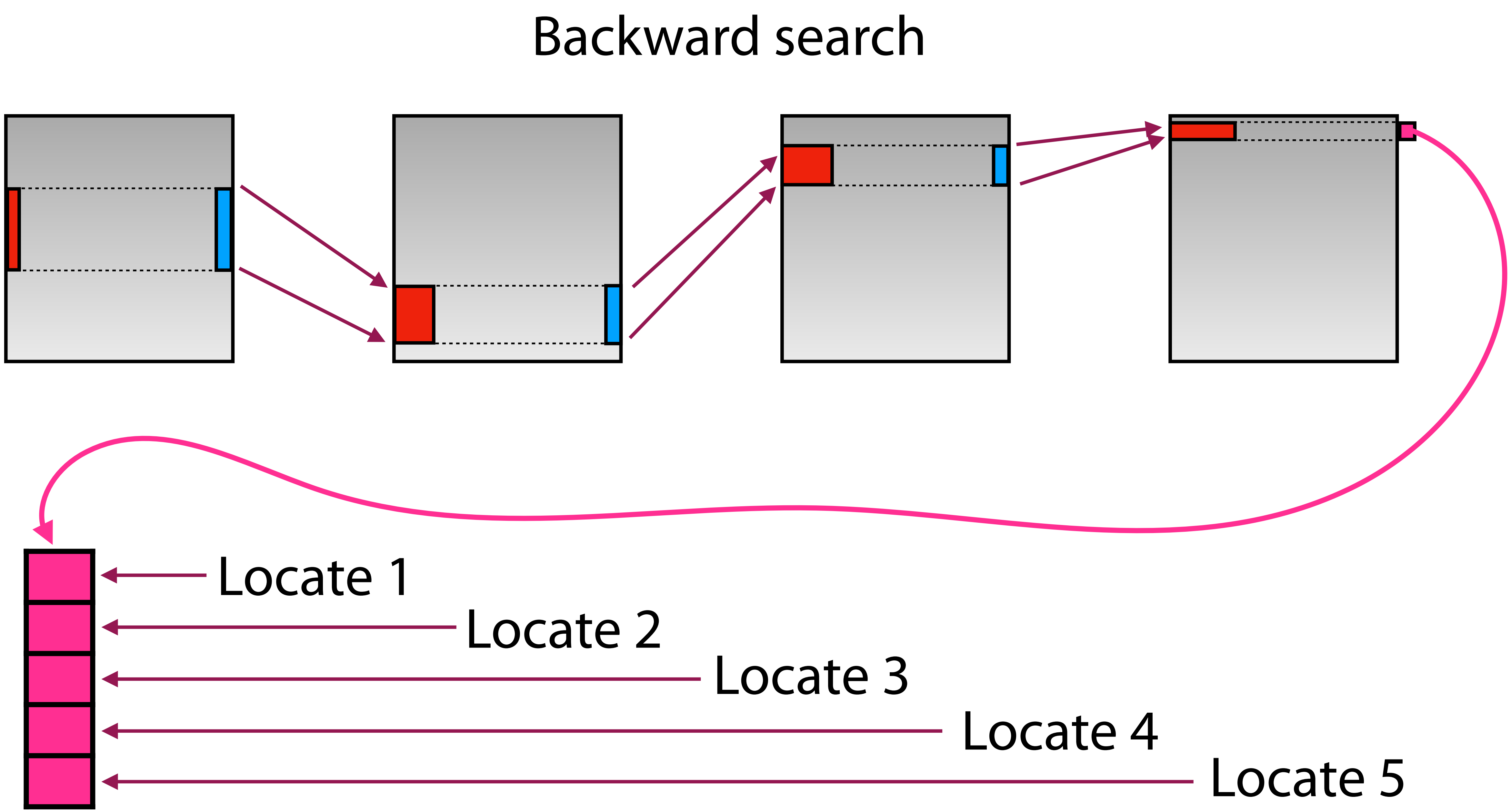
Locate query & SA samples



Samples at run heads & tails

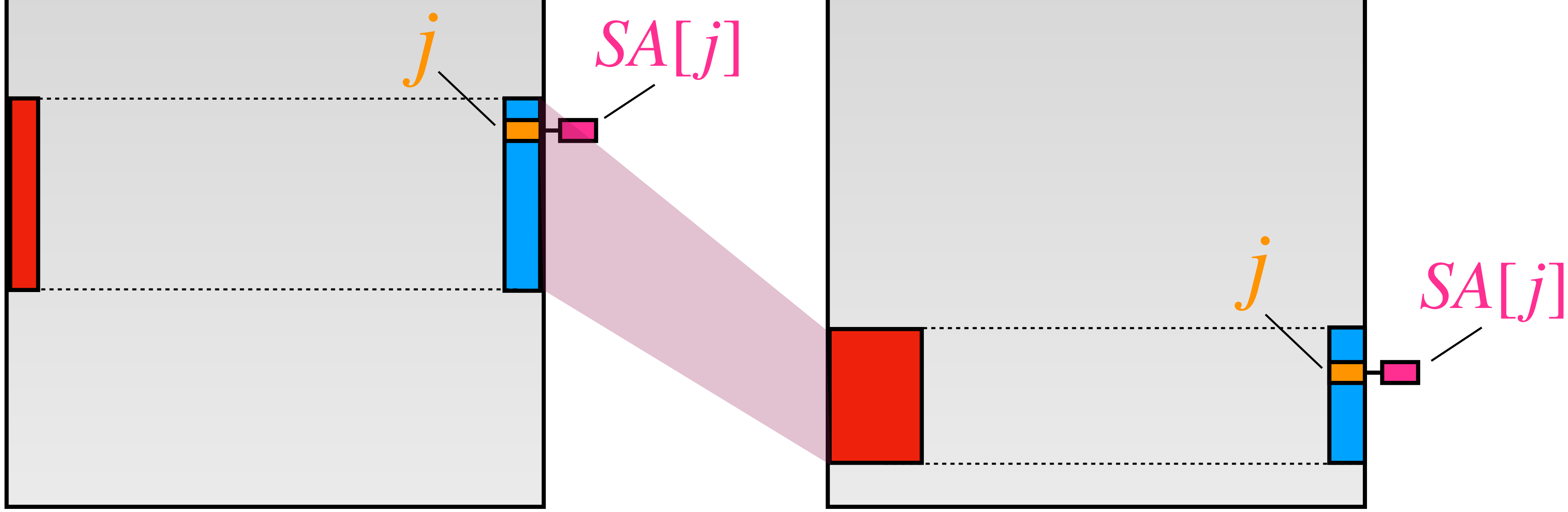
Standard backward search

Usually we do backward search first, locate queries second:



Backward search with toehold

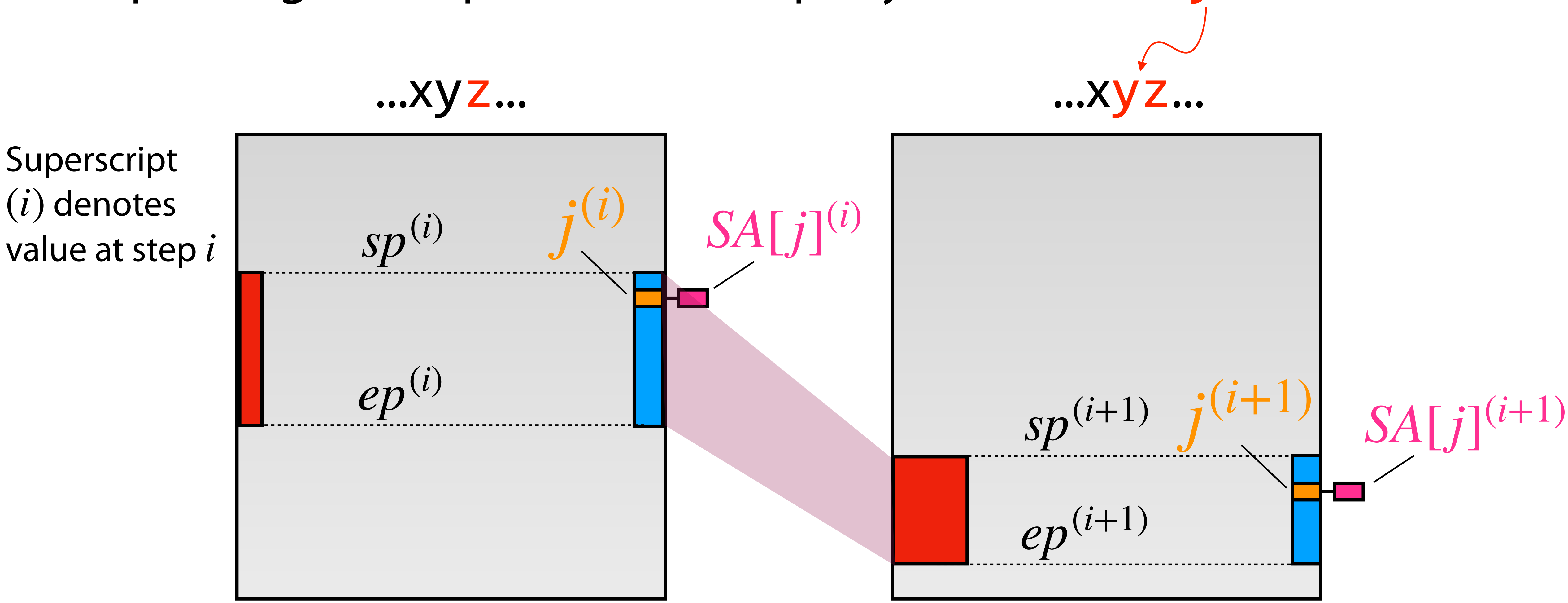
Augment backward search so that *at every step*, we have one offset j in our range $[sp, ep]$ for which we know $SA[j]$



How to update j and $SA[j]$ at each step?

Backward search with toehold

Say we have a j, $[sp, ep]$ and $SA[j]$ for step i and we're updating for step $i + 1$. Next query character is y.

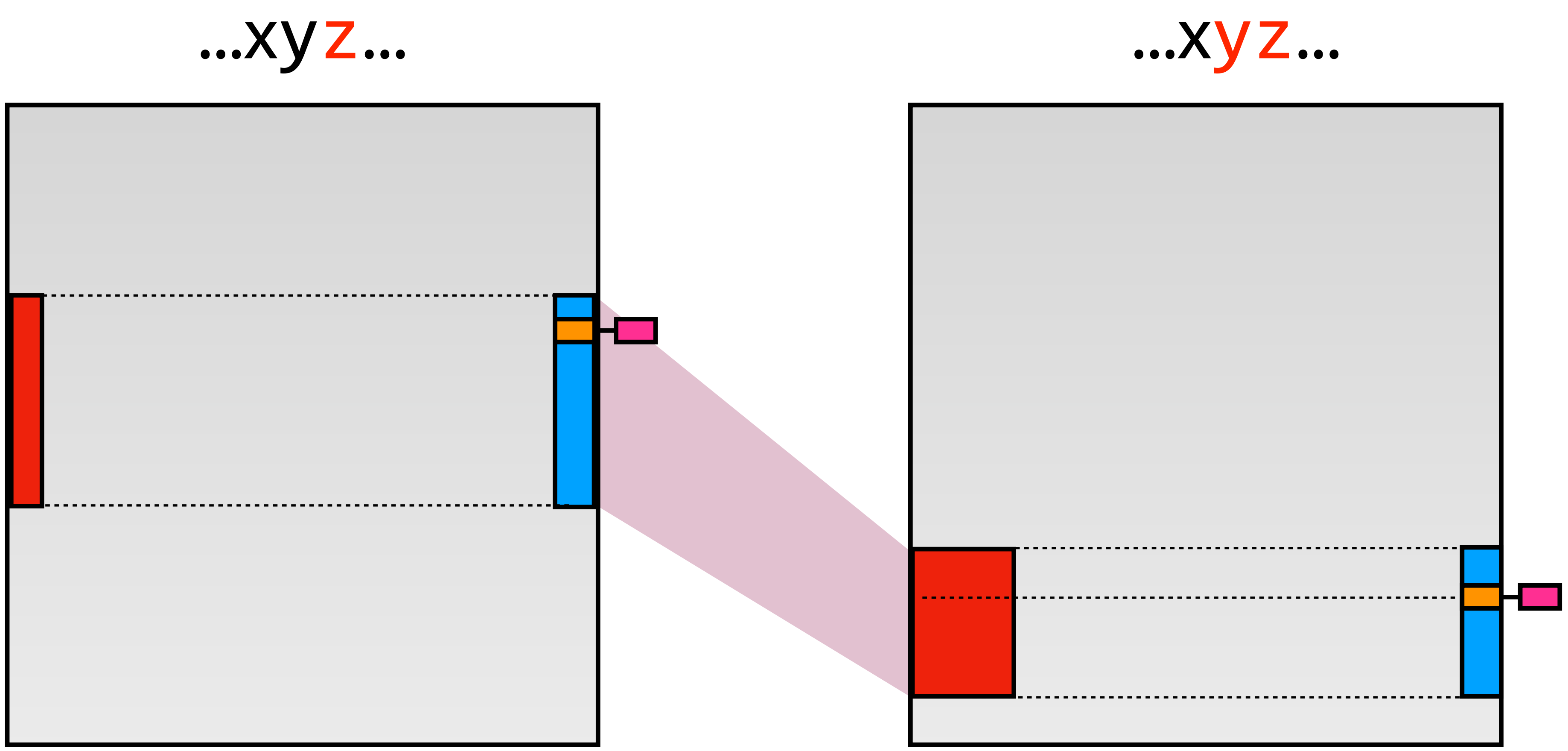


Two cases: Case 1: BWT[$j^{(i)}$] = y   Case 2: BWT[$j^{(i)}$] $\neq$ y

Backward search with toehold

Case 1: BWT$[j^{(i)}]$ = y

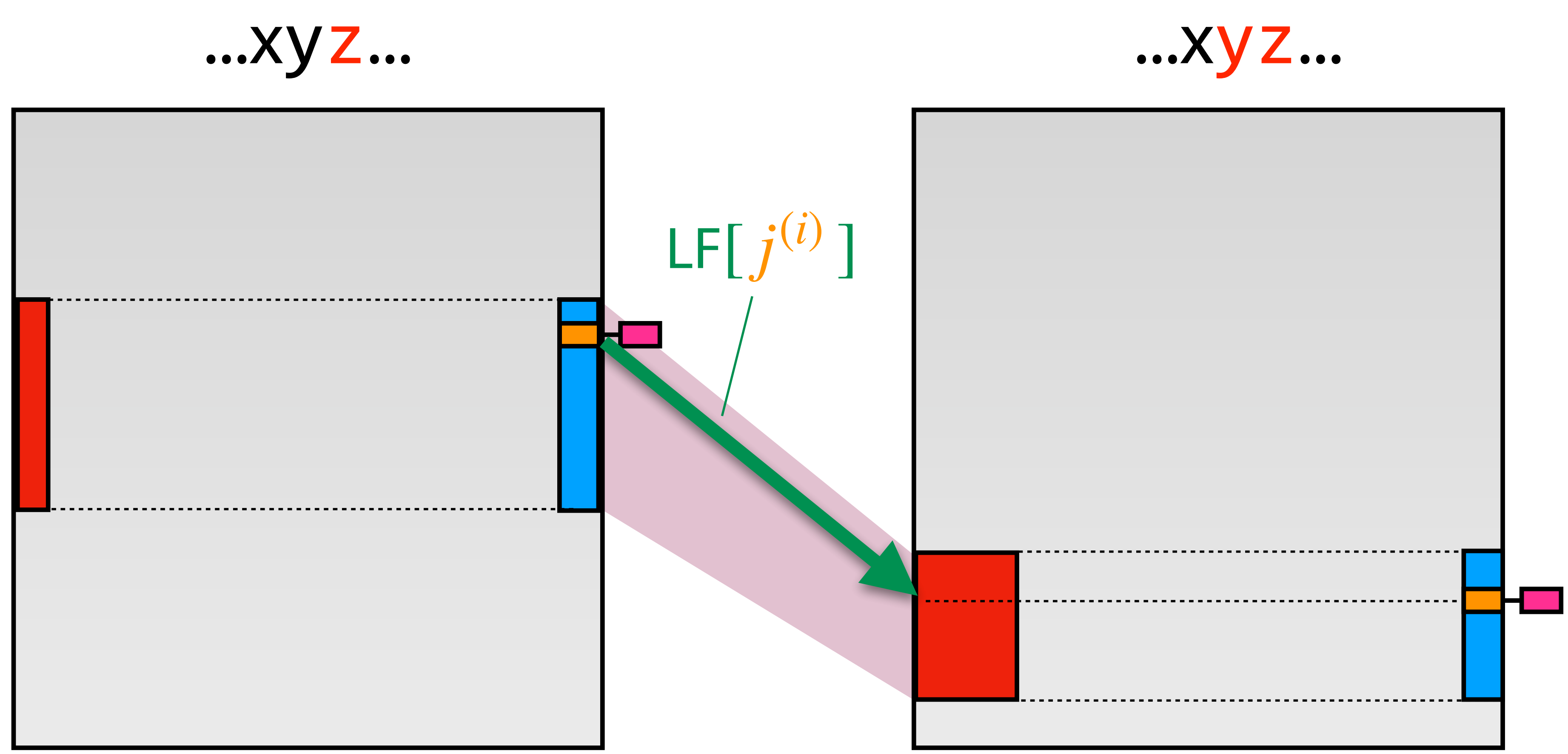
LF does all the work



Backward search with toehold

Case 1: BWT[$j^{(i)}$] = y

LF does all the work



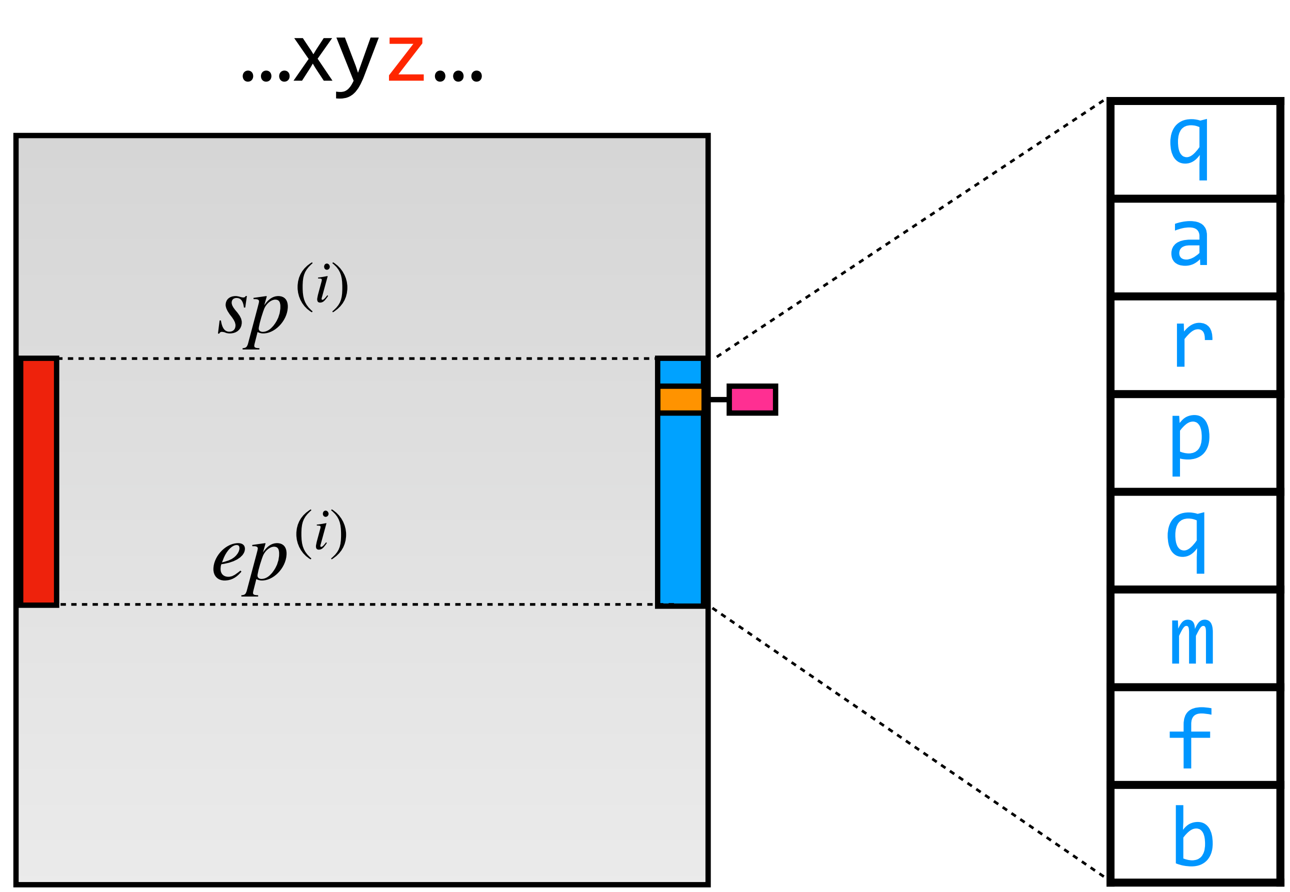
$j^{(i+1)} = \mathrm{LF}[\, j^{(i)} \,]$

$SA[j]^{(i+1)} = SA[j]^{(i)} - 1$

Backward search with toehold

Case 2: BWT[$j^{(i)}$] $\neq$ y

Case 2a: **No** element in BWT[$sp^{(i)} .. ep^{(i)}$] equals y

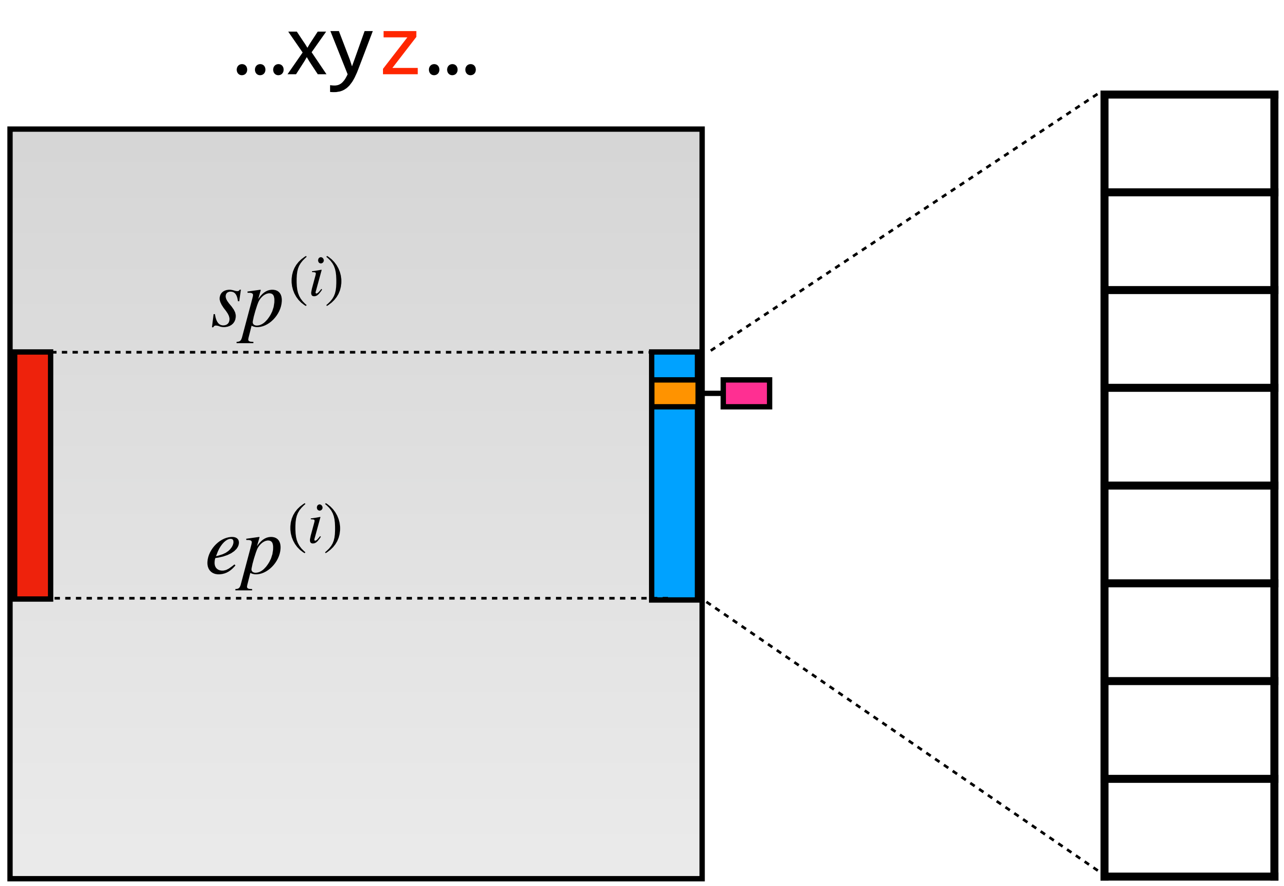


No matches found, no locate queries needed

Case 2b (next slide): Some elements in BWT[$sp^{(i)} .. ep^{(i)}$] match y, but BWT[$j^{(i)}$] doesn't

Backward search with toehold

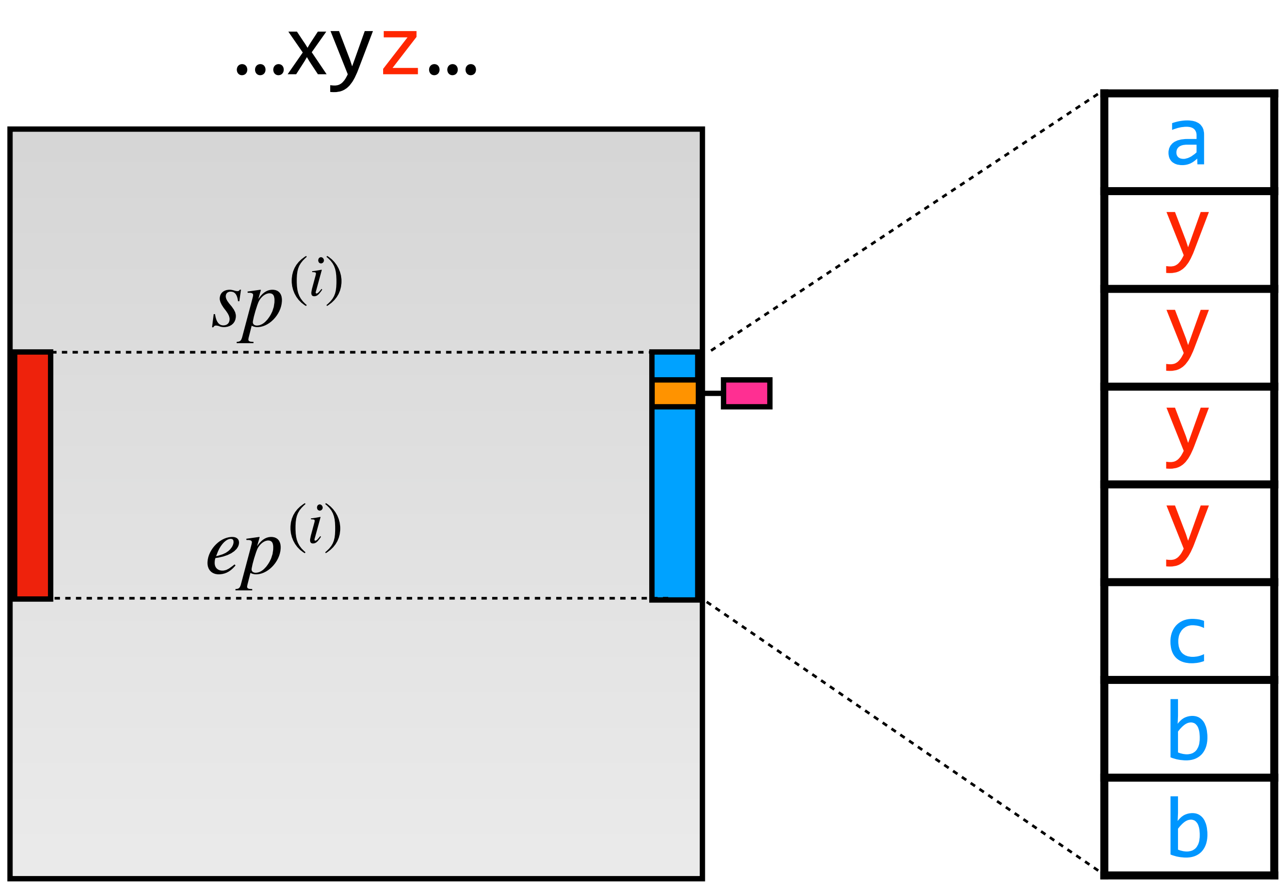
<u>Case 2b</u>: Some elements in BWT[$sp^{(i)} \ldots ep^{(i)}$] match y, but BWT[$j^{(i)}$] doesn't



BWT[$sp^{(i)} \ldots ep^{(i)}$] **must** contain a y-run head or tail.

Backward search with toehold

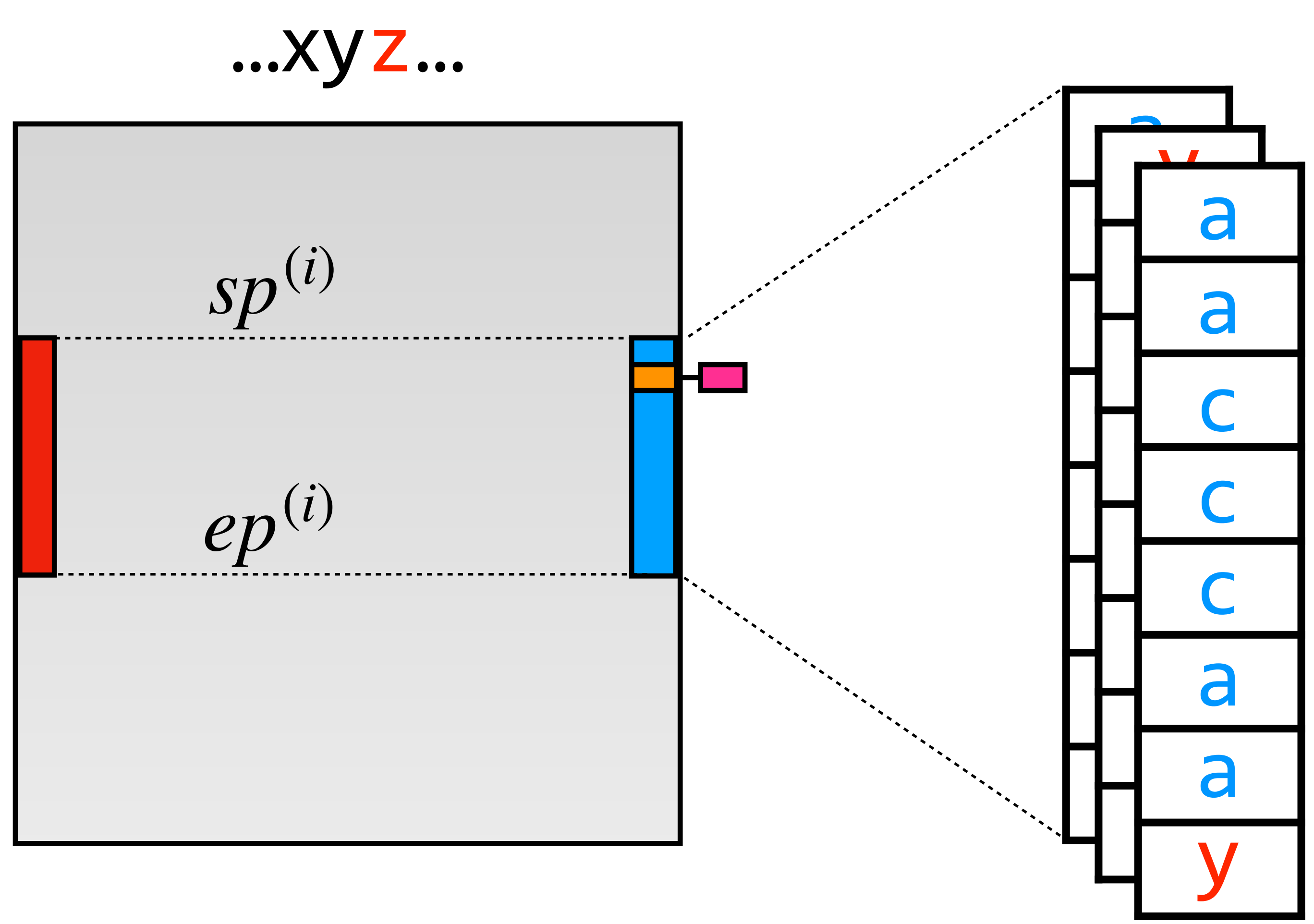
Case 2b: Some elements in BWT[$sp^{(i)} .. ep^{(i)}$] match y, but BWT[$j^{(i)}$] doesn't



BWT[$sp^{(i)} .. ep^{(i)}$] **must** contain a y-run head or tail.

Backward search with toehold

Case 2b: Some elements in BWT[$sp^{(i)} \,..\, ep^{(i)}$] match y, but BWT[$j^{(i)}$] doesn't

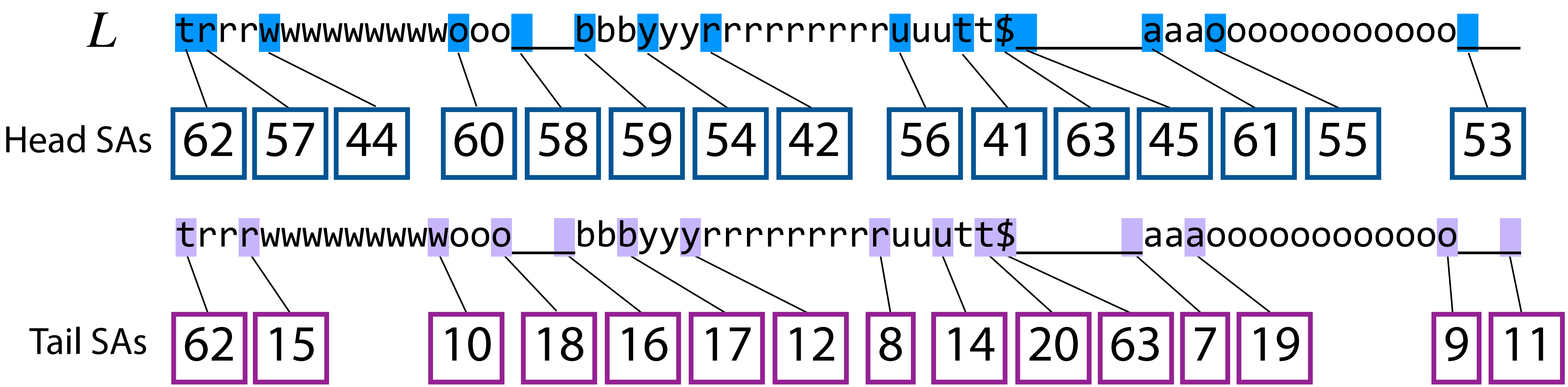


BWT[$sp^{(i)} \,..\, ep^{(i)}$] **must** contain a y-run head or tail.

Let $j^{(i+1)}$ be the offset of a y-head or y-tail. Look up $SA[j^{(i+1)}]$ in our structure holding SA items at heads/tails.

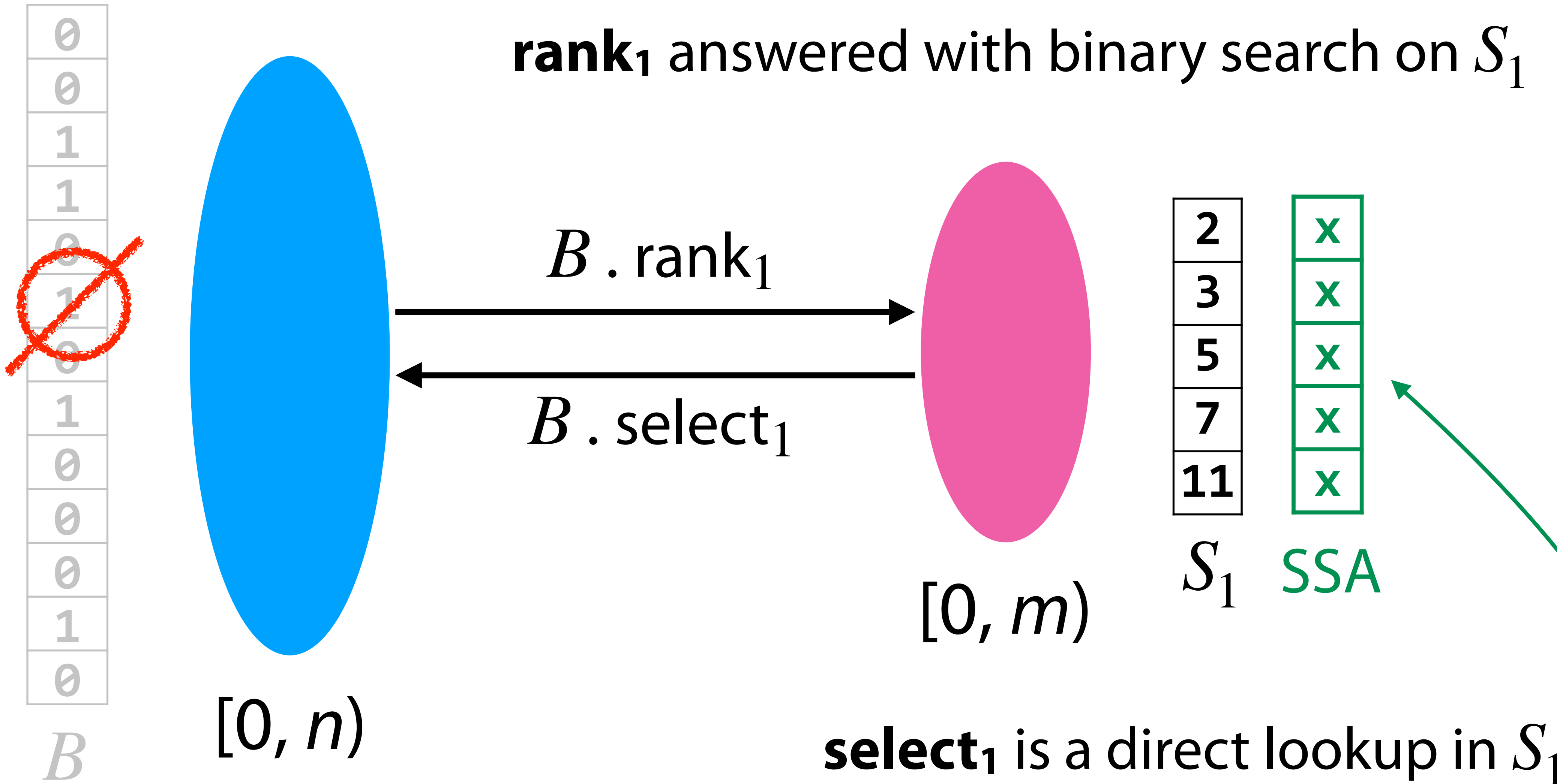
Toehold predecessor structure

How exactly to look up $SA[j^{(i+1)}]$ in our structure holding SA items at heads/tails?



Recall our discussion of the **predecessor structure** from the last video...

Run-length FM Index



Associating values with set bits is as easy as adding **more arrays** parallel to S_1

Toehold predecessor structure

Similarly, we can add associativity to the structure we referenced here:

Run-length FM Index

Predecessor queries on B

$O(\log\log n/r)$ time, $O(r)$ space
Belazzougui & Navarro, Thm A.1

Rank queries on S

$O(\log\log \sigma)$ time, $O(r)$ space
Belazzougui & Navarro, Thm 5.4

Select queries on C & B'

$O(1)$ time, $O(r)$ space
Array of cumulative counts

RLFM index:

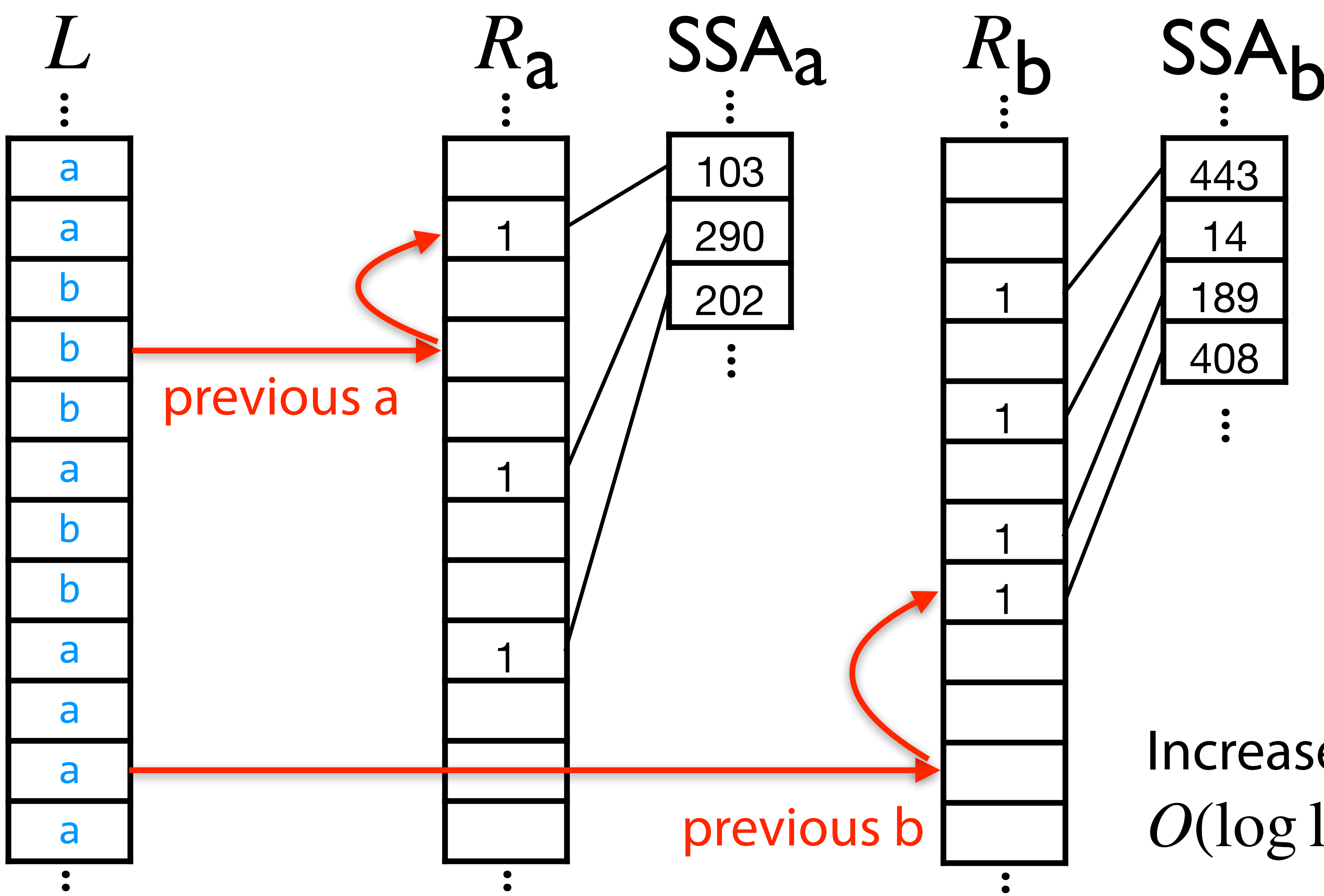
$$
\begin{aligned}
r &\leftarrow B\,.\,\mathrm{rank}_1(\mathit{off} + 1)\\
o &\leftarrow B\,.\,\mathrm{select}_1(r)\\
pv &\leftarrow S\,.\,\mathrm{rank}_c(r - 1)\\
s_1 &\leftarrow C[c]\\
s_2 &\leftarrow B'[c]\,.\,\mathrm{select}_1(pv)\\
\text{next\_off} &\leftarrow s_1 + s_2 + (\mathit{off} - o)
\end{aligned}
$$

Gagie T, Navarro G, and Prezza P. Optimal-time text indexing in BWT-runs bounded space. Proceedings of 29th SODA, ACM-SIAM. 2018. pp1459—1477.

Belazzougui, Djamal, and Gonzalo Navarro. "Optimal lower and upper bounds for representing sequences." ACM Transactions on Algorithms (TALG) 11.4 (2015): 1-21.

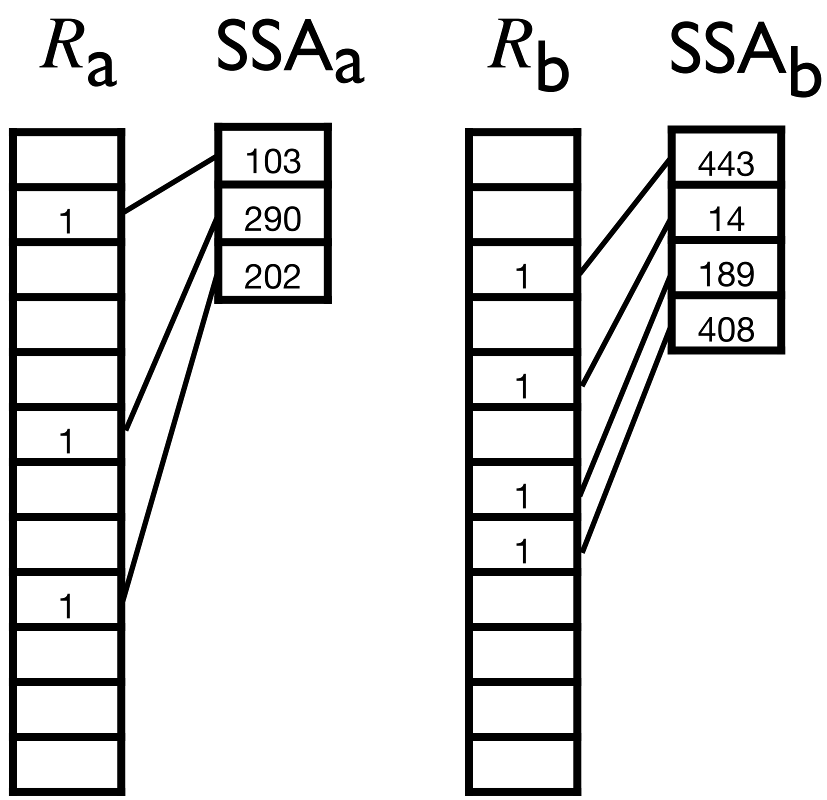
Toehold predecessor structure

Final point: structure needs to be stratified by character



Increases time to $O(\log\log(\sigma + n/r))$

Toehold predecessor structure

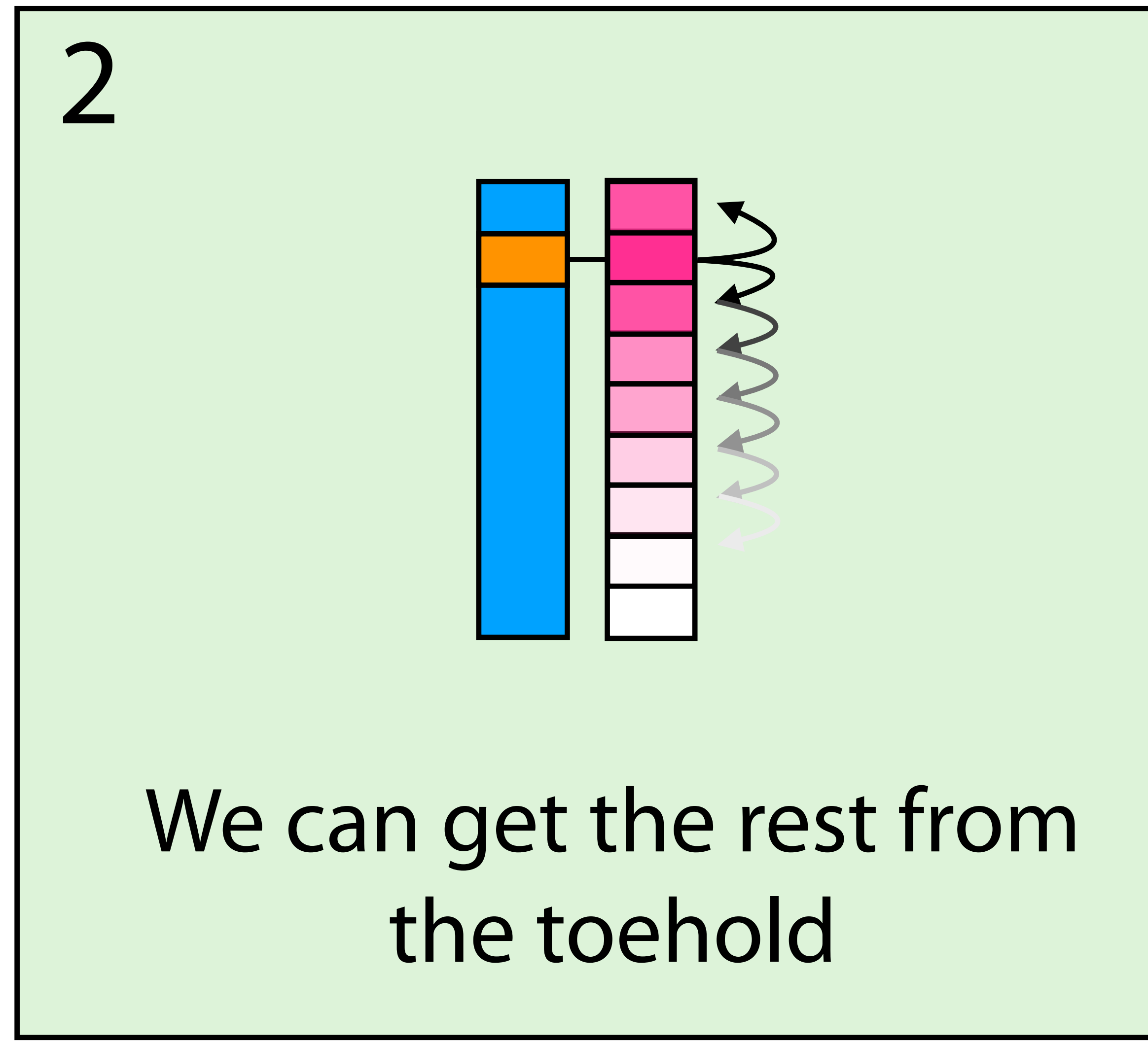


L `trrrwwwwwwwwwooo___bbbyyyrrrrrrrrruuutt$______aaaoooooooooooo___`

$	39					
_	16	18	40	45	61	63
a	46	48				
b	19	21				
o	13	15	49	60		

r	1	3	25	33
t	0	37	38	
u	34	36		
w	4	12		
y	22	24		

Next video



r-index: Gagie T, Navarro G, and Prezza P. Optimal-time text indexing in BWT-runs bounded space. Proceedings of 29th SODA, ACM-SIAM. 2018. pp1459—1477.