

String Algorithms and Data Structures

FM Index

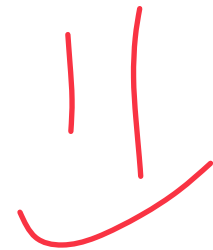
CS 199-225

Brad Solomon

March 30, 2026



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN



Department of Computer Science

Learning Objectives

Review the BWT

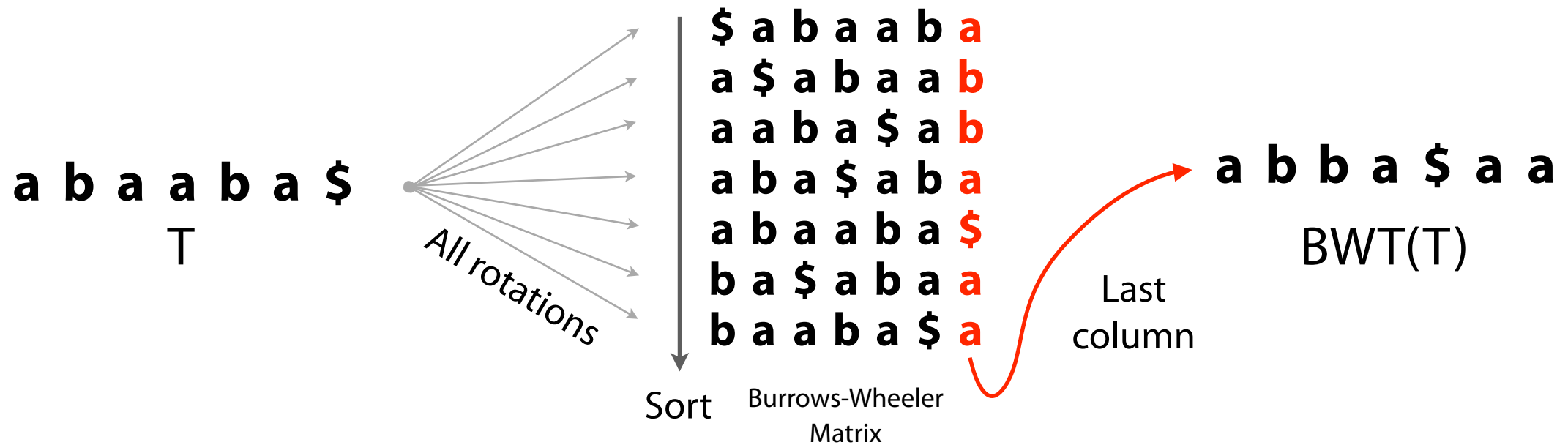


See how the FM Index extends the BWT to solve search

↳ Capstone exact pattern matching

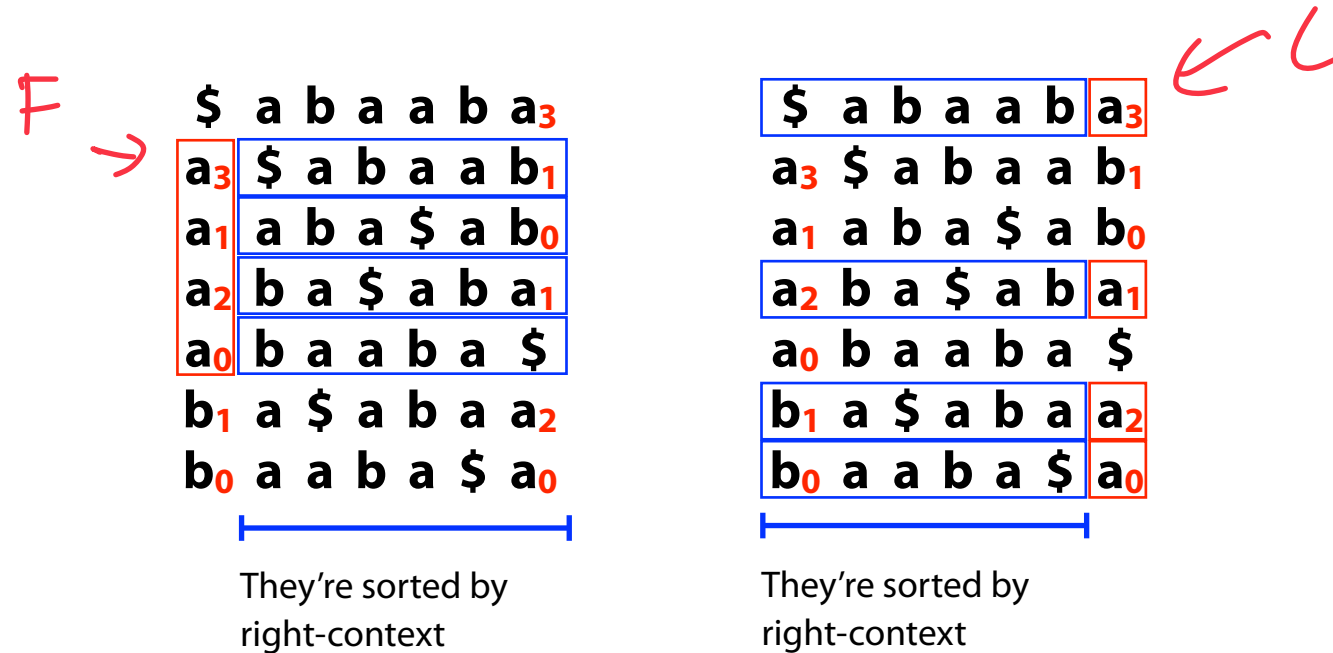
Burrows-Wheeler Transform

Reversible permutation of the characters of a string



Burrows-Wheeler Transform: LF Mapping

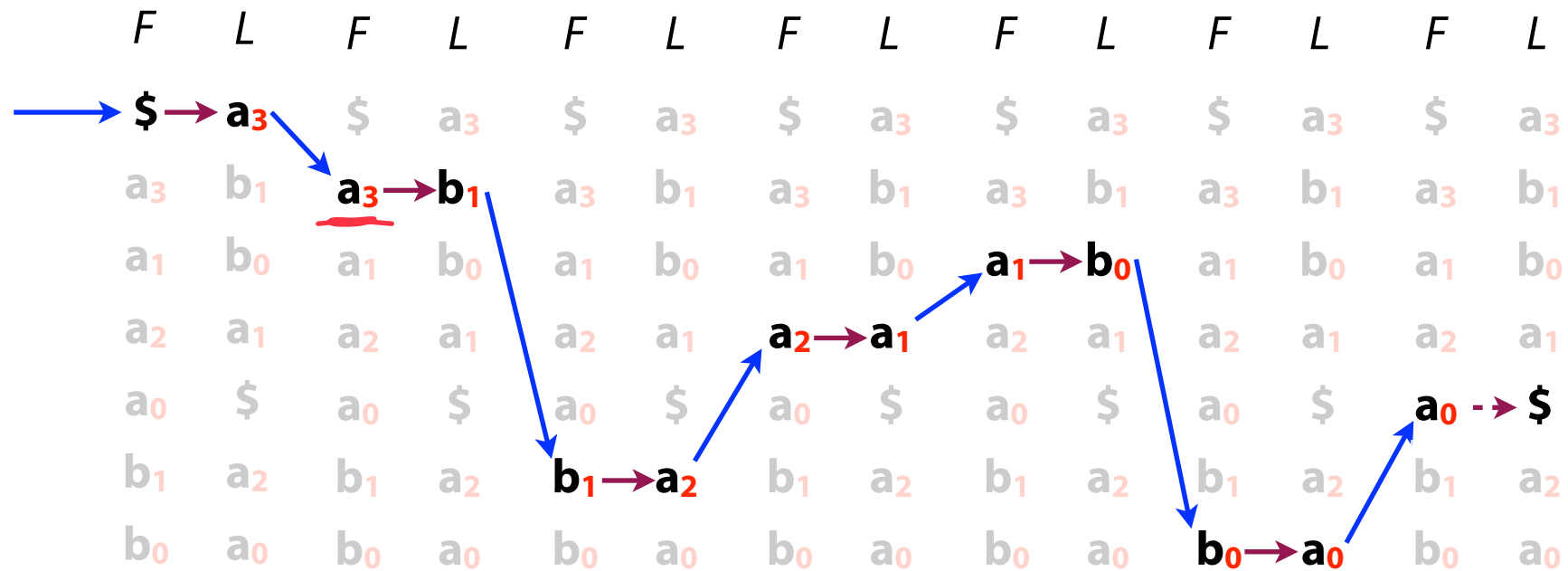
The i^{th} occurrence of a character c in L and the i^{th} occurrence of c in F correspond to the *same* occurrence in T (i.e. have same rank)



Any ranking we give to characters in T will match in F and L

Burrows-Wheeler Transform: LF Mapping

Another way to visualize:



T: a₀ b₀ a₁ a₂ b₁ a₃ \$

A review of 'F' and 'L'

$L = \text{CGGGCC}\$$ $\Sigma = \text{"ACGT"}$

How can we represent F ?

As a full text string: $F = \$\text{CCCGGG}$

As a map<string, int>: $F = \{\text{'\$': 1, 'C': 3, 'G': 3}\}$

As a vector<int>: $F = [0, 3, 3, 0]$



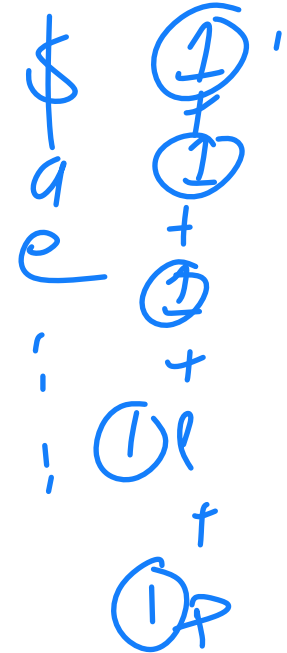
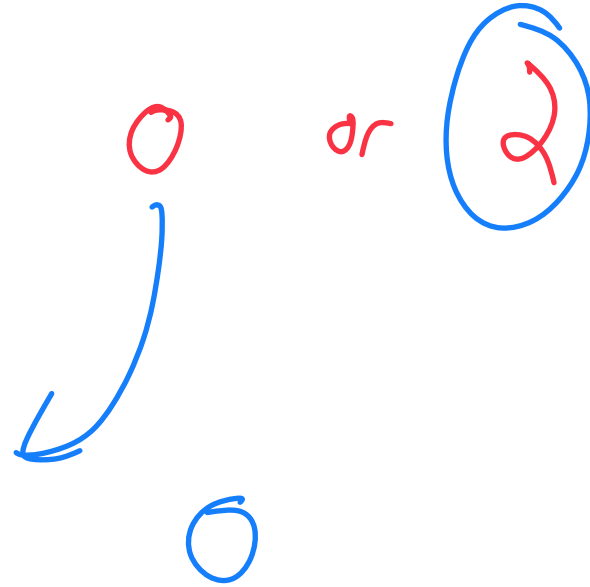
A review of 'F' and 'L'

BWT(T) = e\$1ppa = L

What row index in F contains 'e'?

What row index in L contains 'e'?

What row index in F contains the second 'p'?



A review of 'F' and 'L'

Join Code: 225



BWT(T) = e\$1ppa

What row index in F contains 'e'?

2

What row index in L contains 'e'?

0

What row index in F contains the second 'p'?

5

\$	a	p	p	l	e
a	p	p	l	e	\$
e	\$	a	p	p	l
l	e	\$	a	p	p
p	l	e	\$	a	p
p	p	l	e	\$	a

FM Index



An index combining the BWT *with a few small auxiliary data structures*

Core of index is **first (F)** and **last (L) rows** from BWM:

L is the same size as **T**

F can be represented as array of $|\Sigma|$ integers (or not stored at all!)

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

We're discarding **T** — *we can recover it from L!*

FM Index: Querying

$P = A_x A_y A_z$

L is characters before F

\$	B	B	B	A	A	A₀
A₀	\$	B	B	B	A	A₁
A₁	A	\$	B	B	B	A₂
A₂	A	A	\$	B	B	B₀
B₀	A	A	A	\$	B	B₁
B₁	B	A	A	A	\$	B₂
B₂	B	B	A	A	A	\$

look for A_y

FM Index: Querying

$P = B \ A \ B$

\$ B B B A A **A₀**

A₀ \$ B B B A **A₁**

A₁ A \$ B B B **A₂**

A₂ A A \$ B B **B₀**

B₀ A A A \$ B **B₁** ✓

B₁ B A A A \$ **B₂**

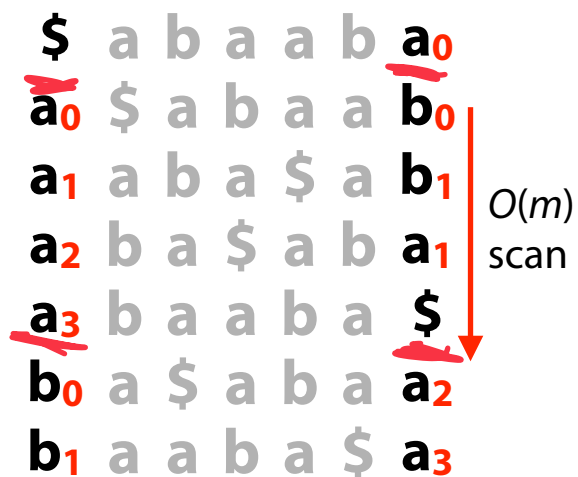
B₂ B B A A A **\$** ✓

Search for A

FM Index: Lingerin9 Issues

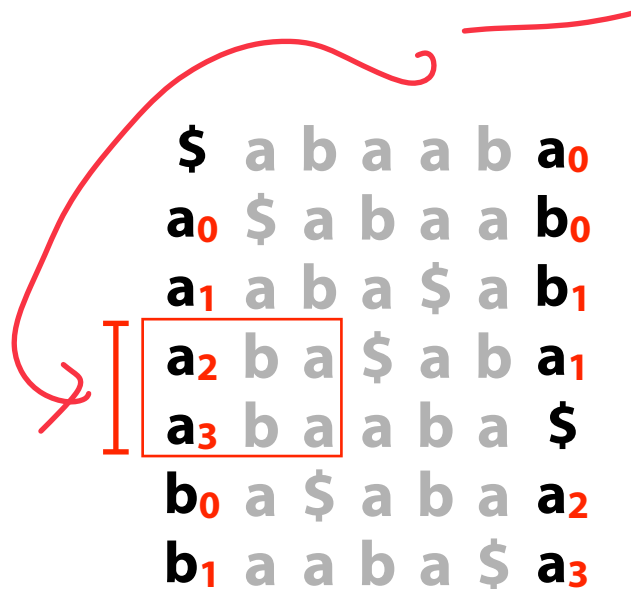
FM Index: Lingering Issues

(1) Scanning for preceding character in L is slow



We don't store ranks!

(2) Need way to find where matches occur in T :



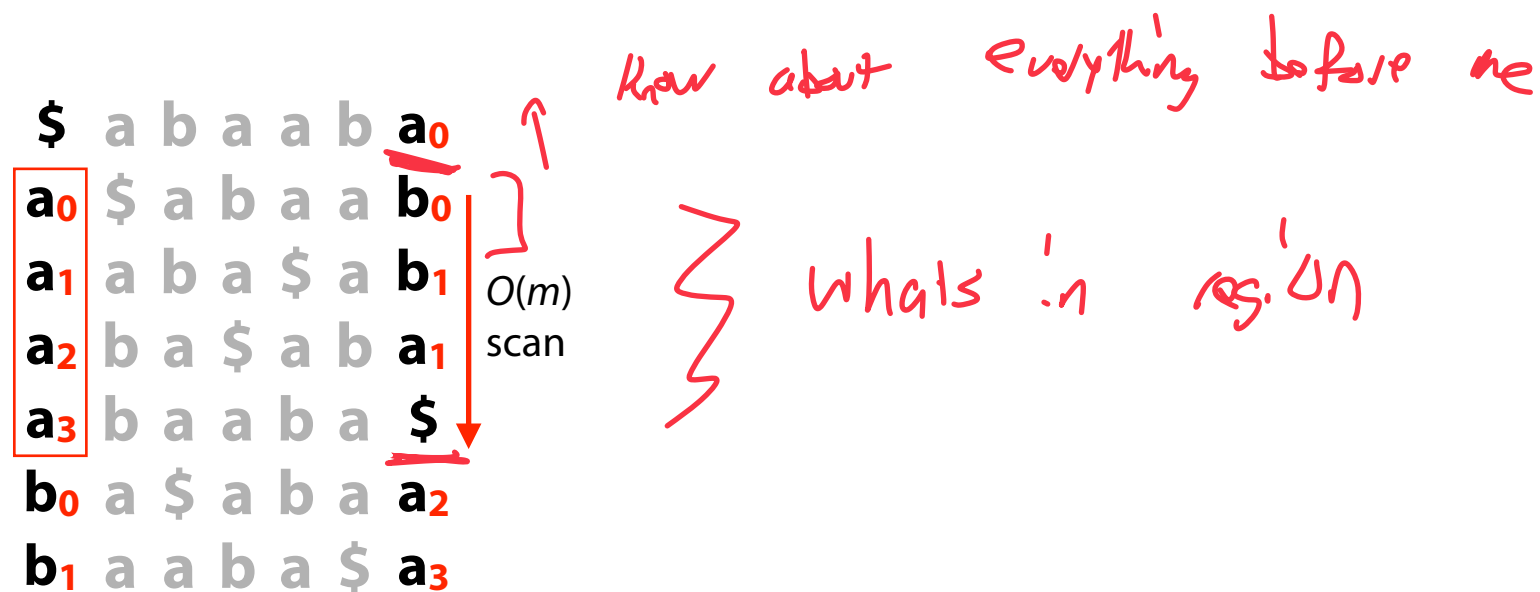
Current output: [3,4]

Location in T : [0,3]

This is where our auxiliary data structures come in...

FM Index: Fast rank calculations

Is there a fast way to determine which *specific* **b**s precede the **a**s in our range?



More generally, given a range in L and a character to search, how can we quickly find all matches (and their ranks)?

FM Index: Occurrence Table

Idea: pre-calculate cumulative # **as**, **bs** in L up to every row:

\mathcal{E}

L	a	b
a	1	0
b	1	<u>1</u>
b	1	2
a	2	2
\$	2	2
a	3	2
a	4	2

FM Index: Occurrence Table

Idea: pre-calculate cumulative # **a**s, **b**s in L up to every row:

L	a	b
a	1	0
b	1	1
b	1	2
a	2	2
\$	2	2
a	3	2
a	4	2

FM Index: Occurrence Table

Query: 'aba'

Idea: pre-calculate cumulative # **as**, **bs** in *L* up to every row:

<i>F</i>	<i>L</i>	a	b	
\$	<u>a</u>	1	0	0
a	b	1	1	1 ←
a	b	1	2	2
a	a	2	2	3
a	<u>\$</u>	2	2	4 ←
b	a	3	2	5
b	a	4	2	6

FM Index: Occurrence Table

Query: 'aba'

Idea: pre-calculate cumulative # **as**, **bs** in *L* up to every row:

<i>F</i>	<i>L</i>	a	b
\$	a	1	0
a	b	1	1
a	b	1	2
a	a	2	2
a	\$	2	2
b	a	3	2
b	a	4	2

2 bs exist in range $\begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$
0 bs up to & including this row
↳ The rank before any range begins
2 bs up to & including this row $O(1)$
↳ Last item in range
lookup for # of ranks occurrences

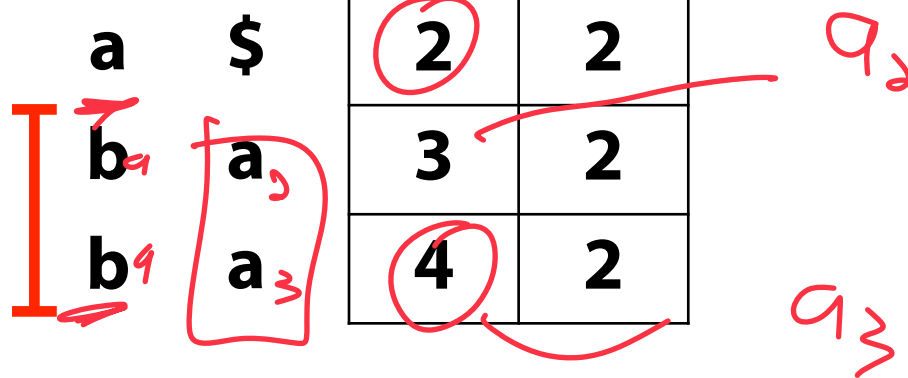
FM Index: Occurrence Table

Query: 'aba'

Idea: pre-calculate cumulative # **a**s, **b**s in *L* up to every row:

<i>F</i>	<i>L</i>	a	b
\$	a ₀	1	0
a	b	1	1
a	b	1	2
a	a ₁	2	2
a	\$	2	2
b ₁	a ₂	3	2
b ₂	a ₃	4	2

What values of **a** (including rank) should I look up next?



Join Code: 225

FM Index: Occurrence Table

Query: 'bb'

What two indices should I look up? What ranks did we find?

<i>F</i>	<i>L</i>	a	b
\$	a	1	0
a	b	1	1
a	\$	1	1
b	b	1	2
b	b	1	3
b	b	1	4
b	a	2	4

b₁, b₂, b₃





FM Index: Occurrence Table

An index combining the BWT with *a few small auxiliary data structures*

Occurrence table speeds up L lookup by implicitly storing **ranks**

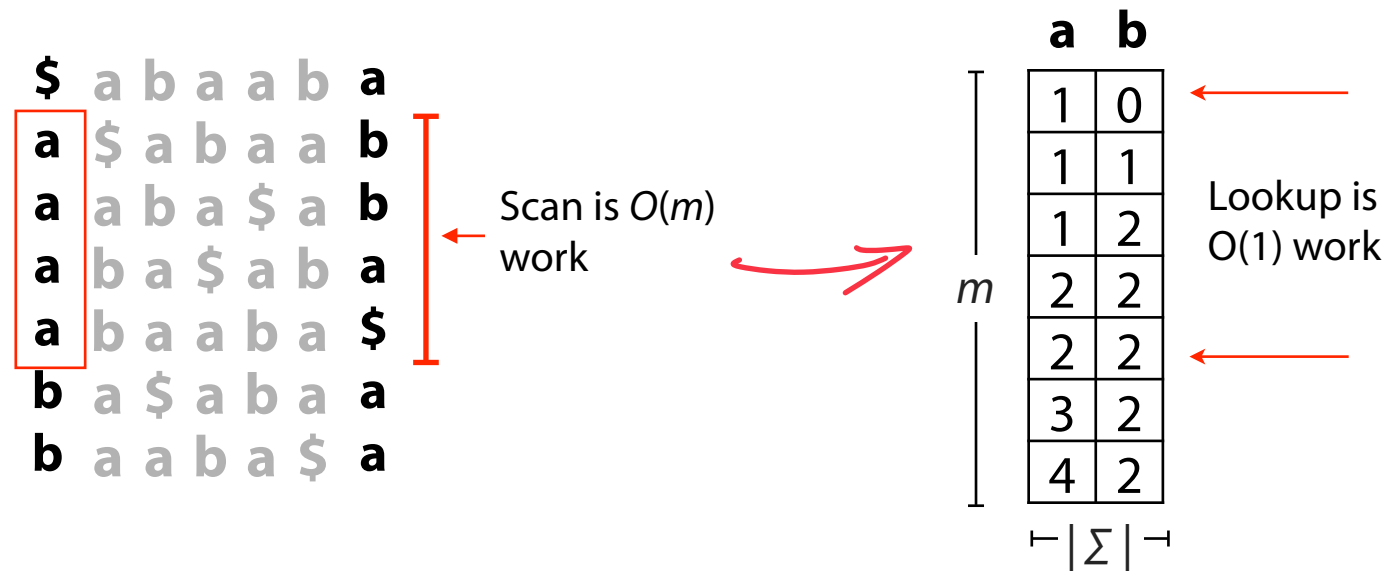


Table is $m \times |\Sigma|$ integers — that's worse than a suffix array!

FM Index: Occurrence Table

Next idea: pre-calculate # **a**s, **b**s in L up to *some* rows, e.g. every 5th row.
Call pre-calculated rows checkpoints.

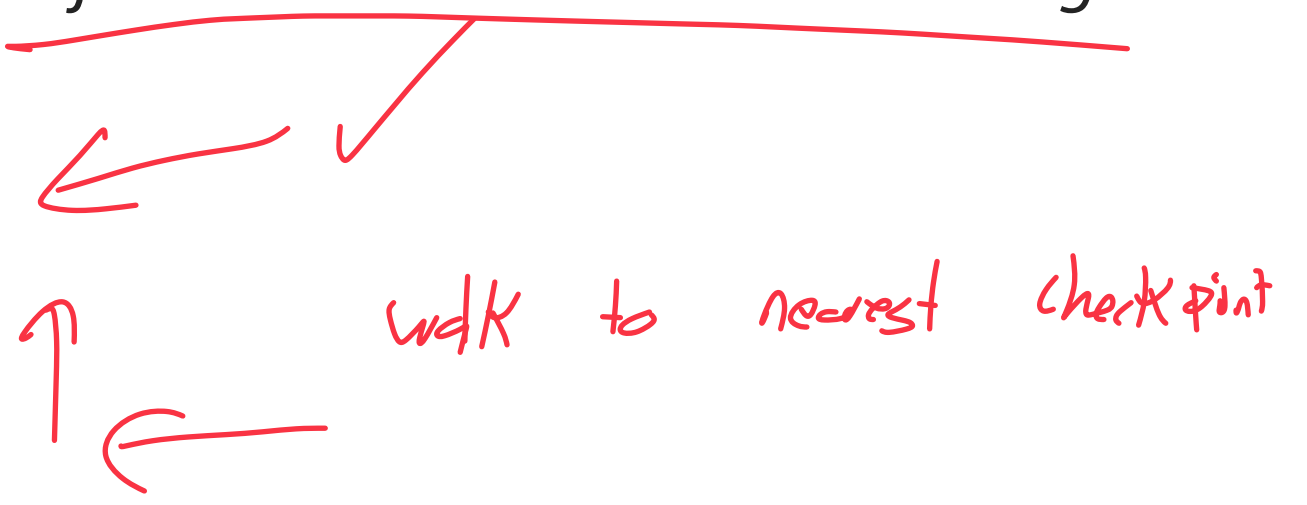
F	L	a	b
\$	a	1	0
a	b		
a	b		
a	a		
a	\$		
b	a	3	2
b	a		

Claim: Still $O(1)$ time to
get rank & occurrences

FM Index: Occurrence Table

To resolve a lookup for a non-checkpoint row, walk to nearest checkpoint. Use value at that checkpoint, *adjusted for characters we saw along the way.*

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



FM Index: Occurrence Table



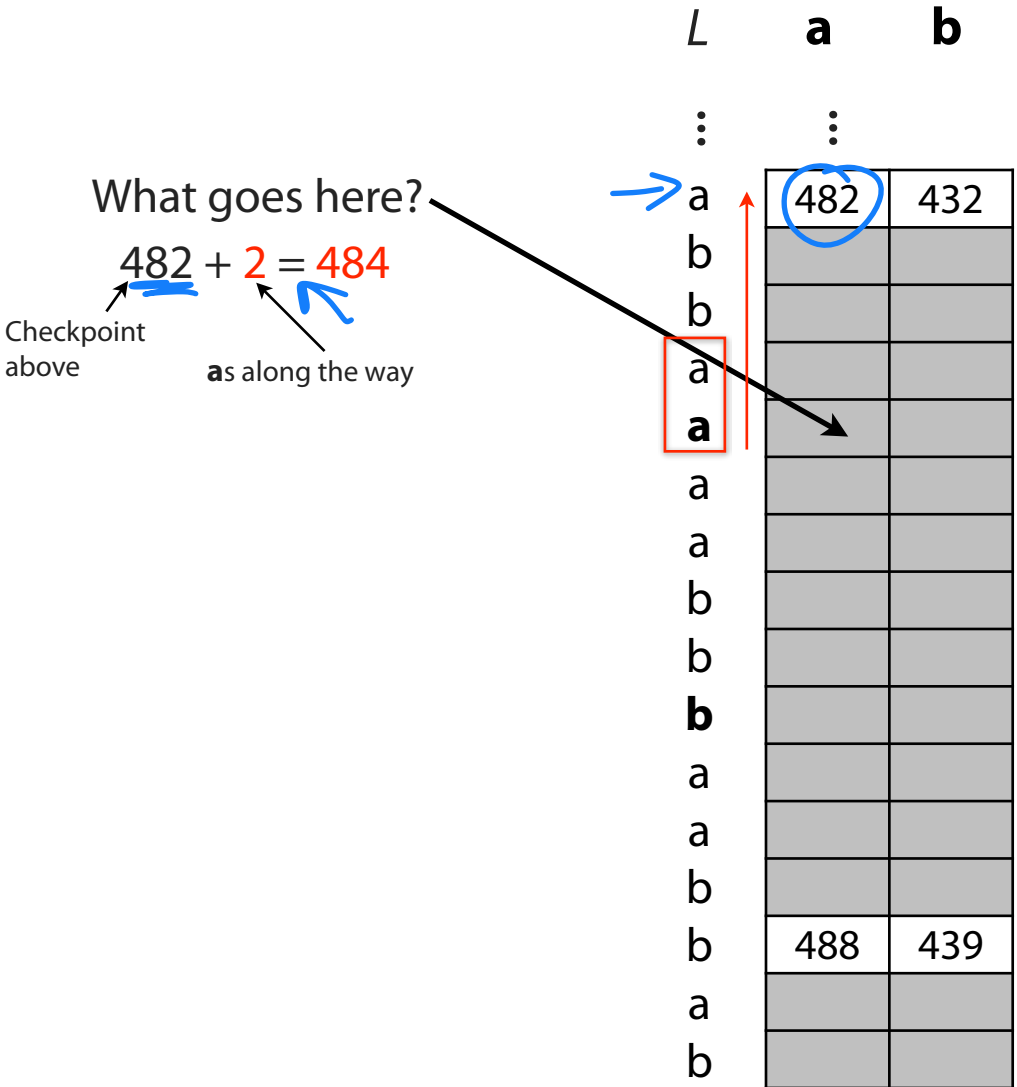
Join Code: 225

What goes here?

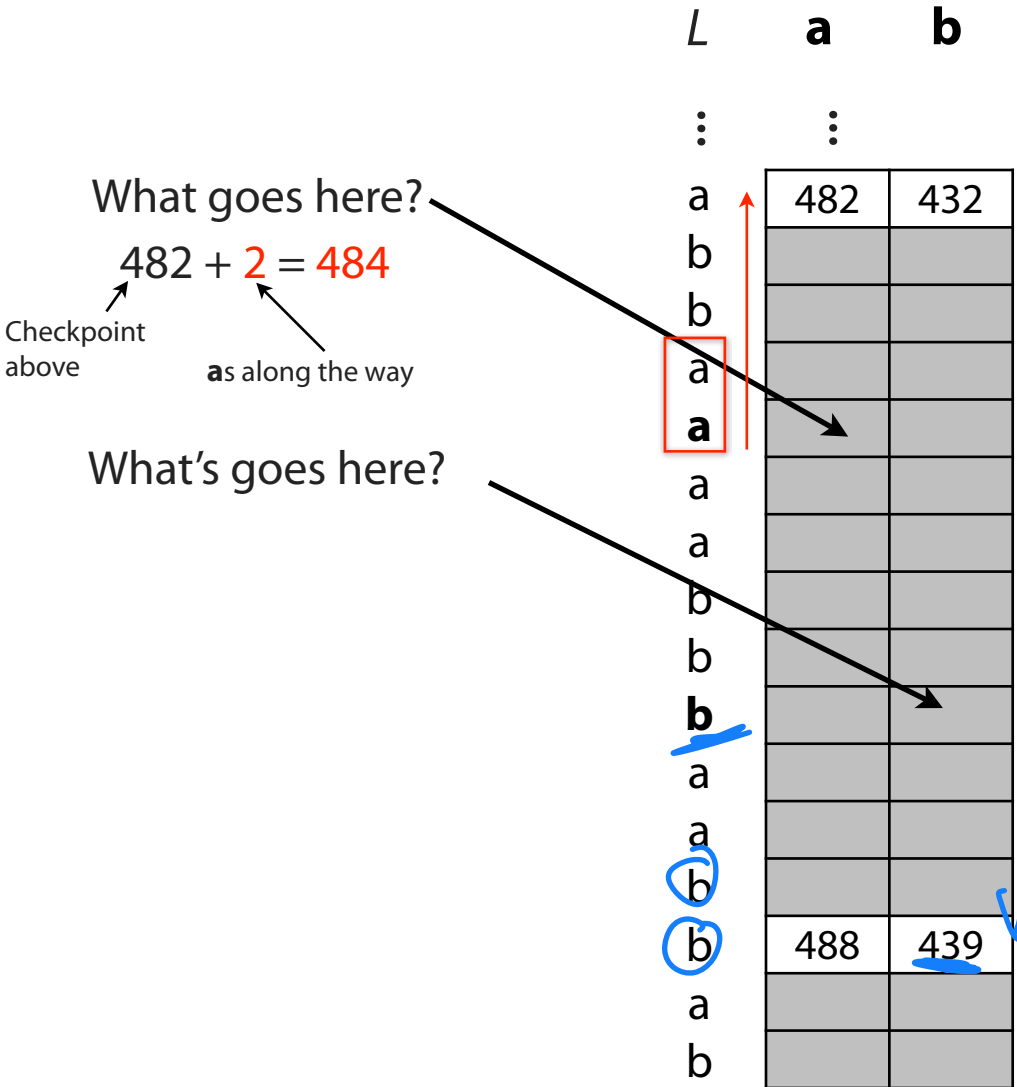
<i>L</i>	a	b
⋮	⋮	
a	482	432
b		
b		
a		
a		
a		
a		
b		
b		
b		
a		
a		
b		
b	488	439
a		
b		

FM Index: Occurrence Table

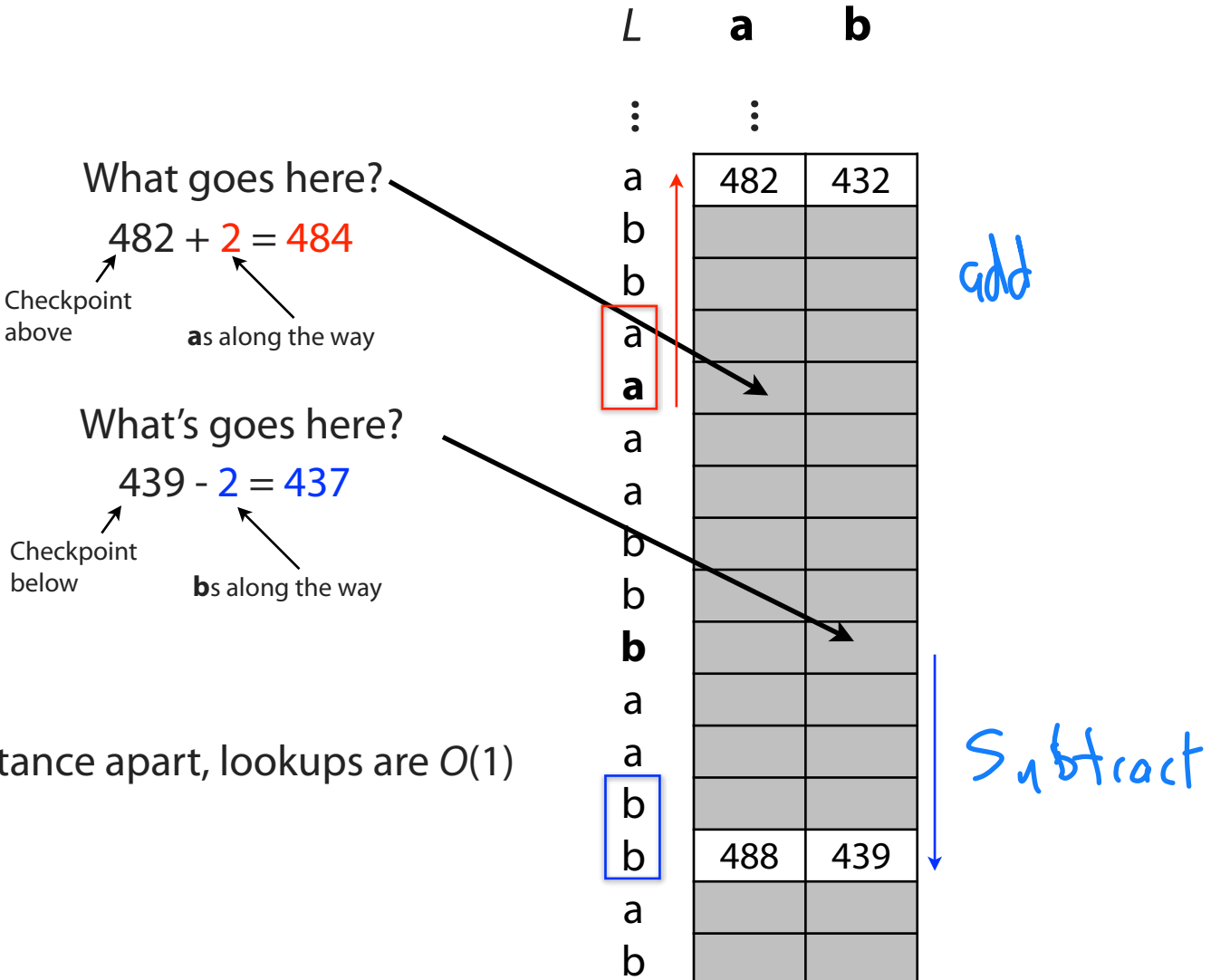
1



FM Index: Occurrence Table



FM Index: Occurrence Table



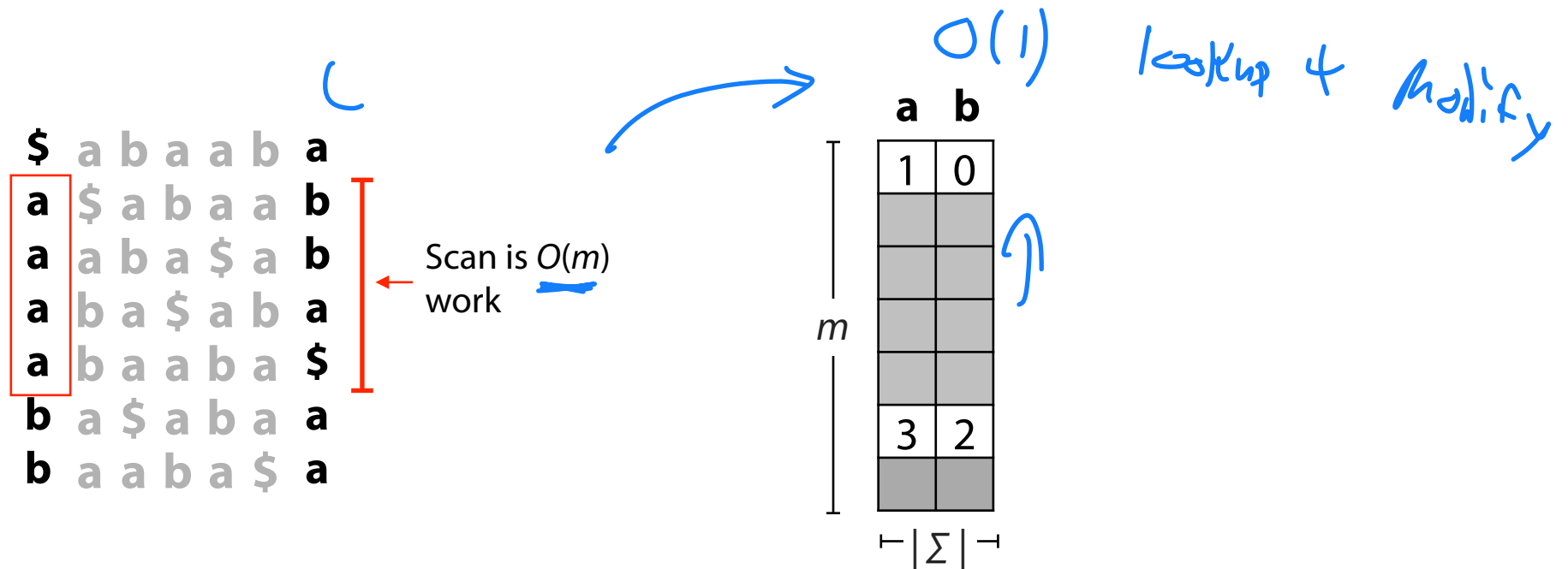
If checkpoints are $O(1)$ distance apart, lookups are $O(1)$



FM Index: Occurrence Table

An index combining the BWT with *a few small auxiliary data structures*

Occurrence table speeds up L lookup by implicitly storing **ranks**

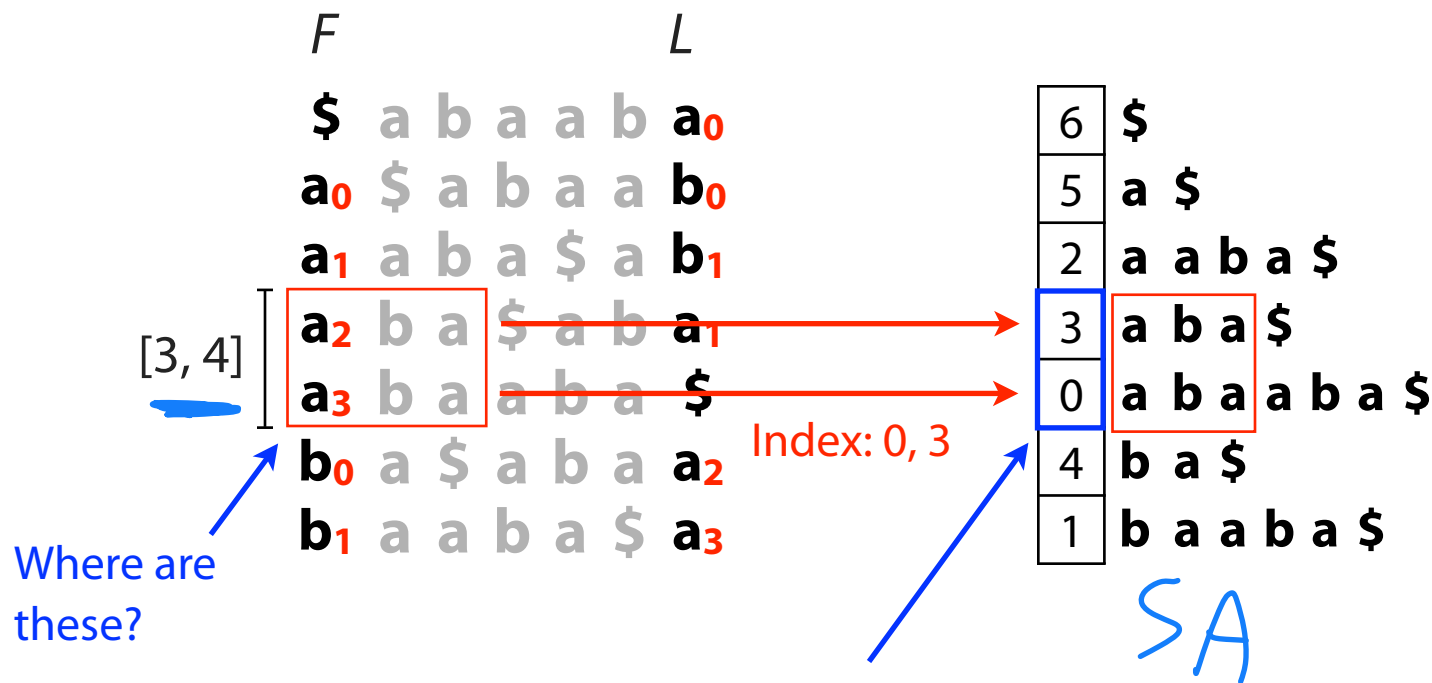


Checkpoints reduce the storage costs (Still $O(m)$ but better than SA)

FM Index: Querying

Problem 2: We don't know *where* the matches are in T...

$P = \mathbf{aba}$ Got the same range, [3, 4], we would have got from suffix array



FM Index: Suffix Array Sampling

Idea: store some suffix array elements, but not all

<i>F</i>		<i>L</i>	<i>SA'</i> (evens only)				
\$	a	b	a	a	b	a	6
a	\$	a	b	a	a	b	
a	a	b	a	\$	a	b	2
a	b	a	\$	a	b	a	
a	b	a	a	b	a	\$	0
b	a	\$	a	b	a	a	4
b	a	a	b	a	\$	a	

FM Index: Suffix Array Sampling

Idea: store some suffix array elements, but not all

<i>F</i>		<i>L</i>		SA' (evens only)			
\$	a	b	a	a	b	a	6
a	\$	a	b	a	a	b	
a	a	b	a	\$	a	b	2
a	b	a	\$	a	b	a	X
a	b	a	a	b	a	\$	0
b	a	\$	a	b	a	a	4
b	a	a	b	a	\$	a	

Lookup for row 4 succeeds

Lookup for row 3 fails - SA entry was discarded

FM Index: Suffix Array Sampling

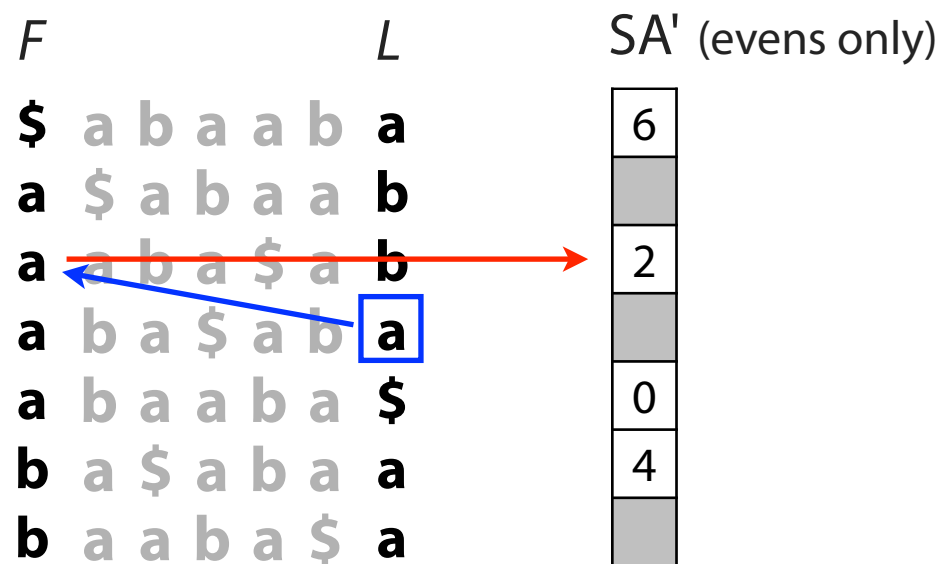
LF Mapping tells us that "a" at the end of row 3 corresponds to...

<i>F</i>		<i>L</i>	<i>SA'</i> (evens only)				
\$	a	b	a	a	b	a	6
a	\$	a	b	a	a	b	
a	a	b	a	\$	a	b	2
a	b	a	\$	a	b	a	
a	b	a	a	b	a	\$	0
b	a	\$	a	b	a	a	4
b	a	a	b	a	\$	a	

FM Index: Suffix Array Sampling

LF Mapping tells us that "a" at the end of row 3 corresponds to...

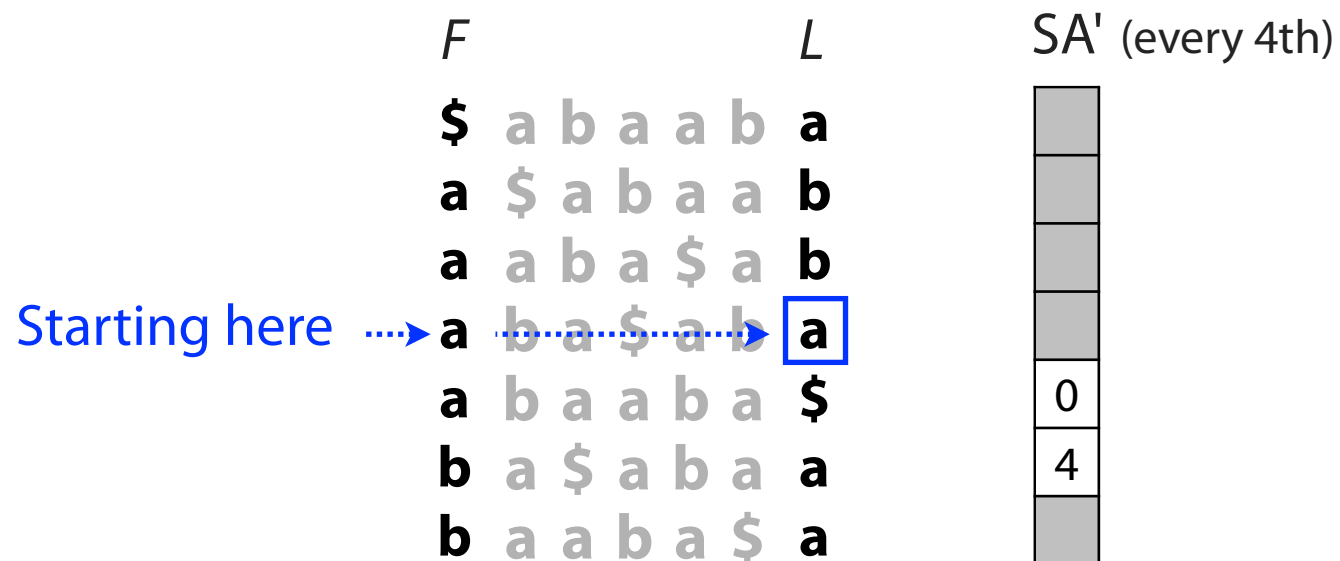
... "a" at the beginning of row 2



If saved SA values are $O(1)$ positions apart in T , resolving index is $O(1)$ time

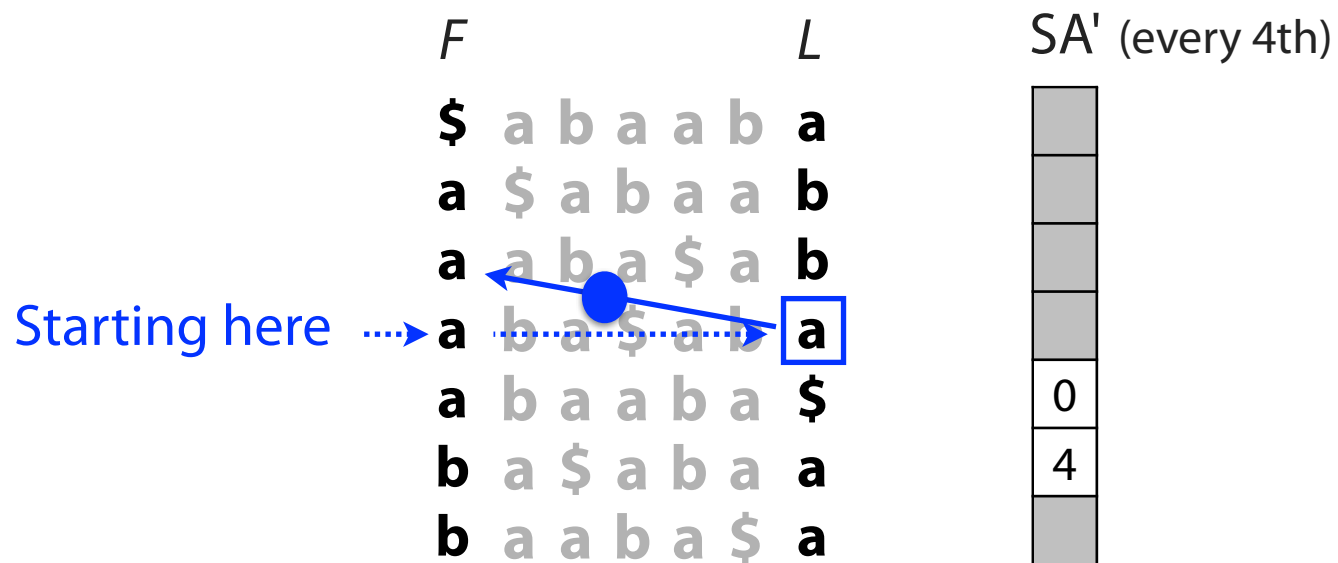
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:



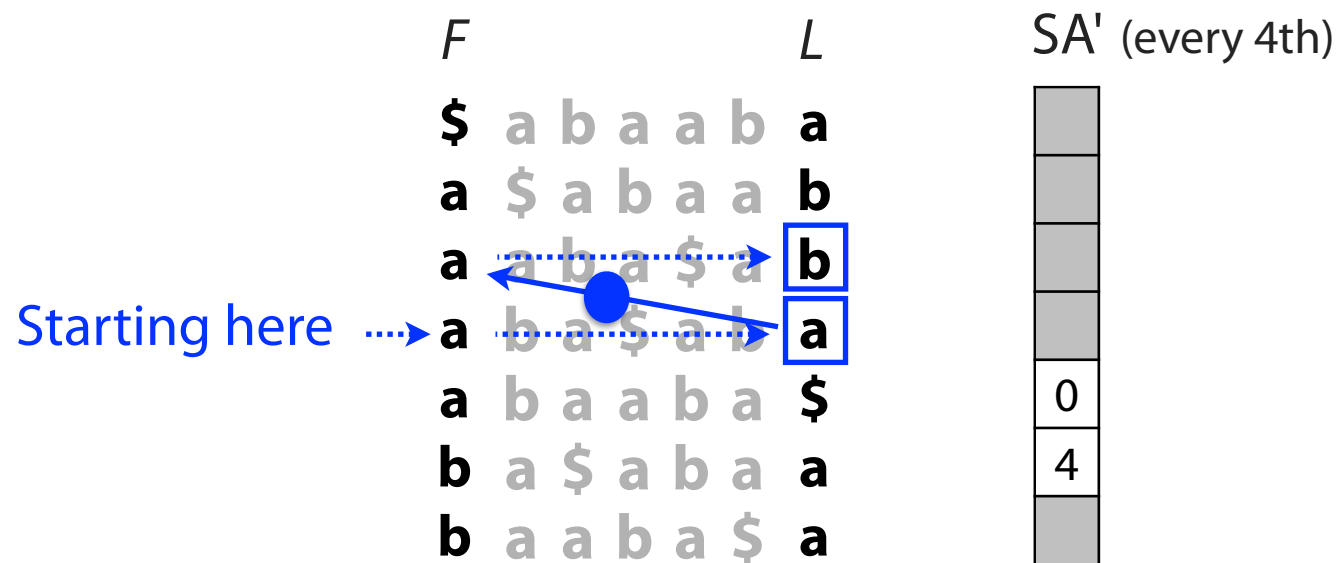
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:



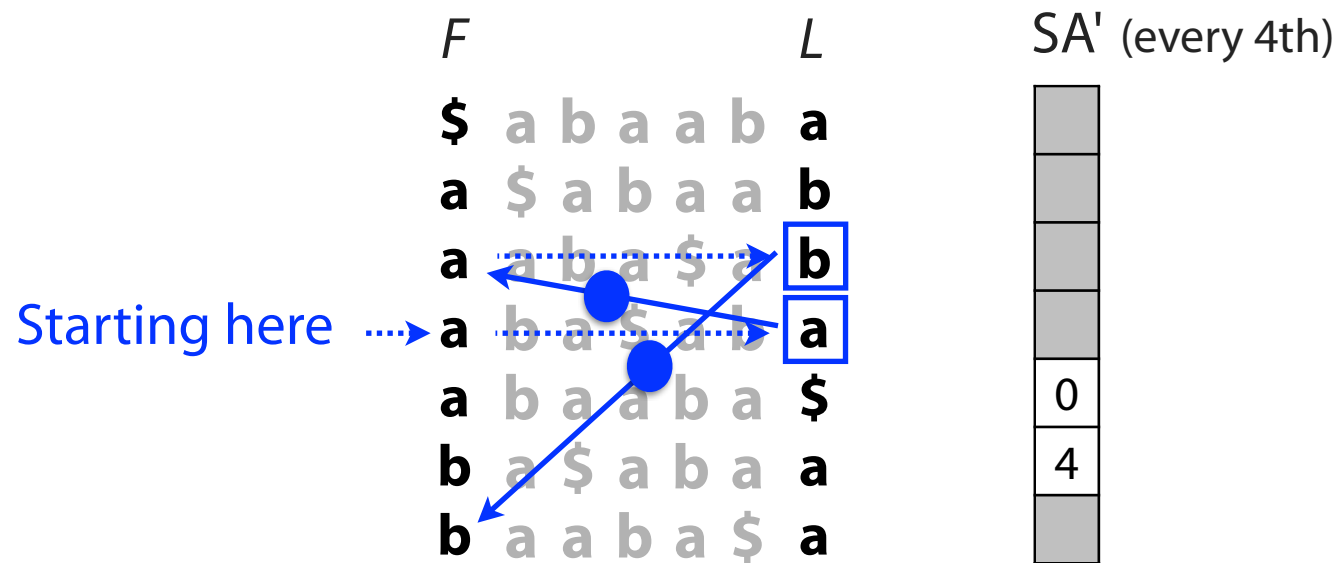
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:



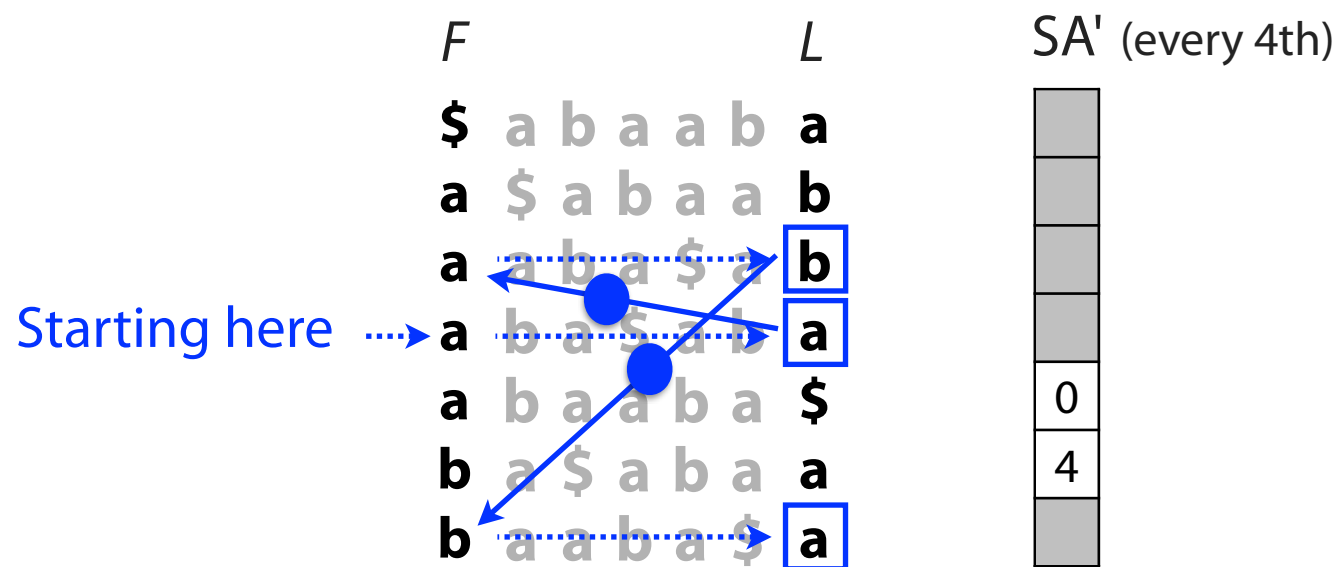
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:



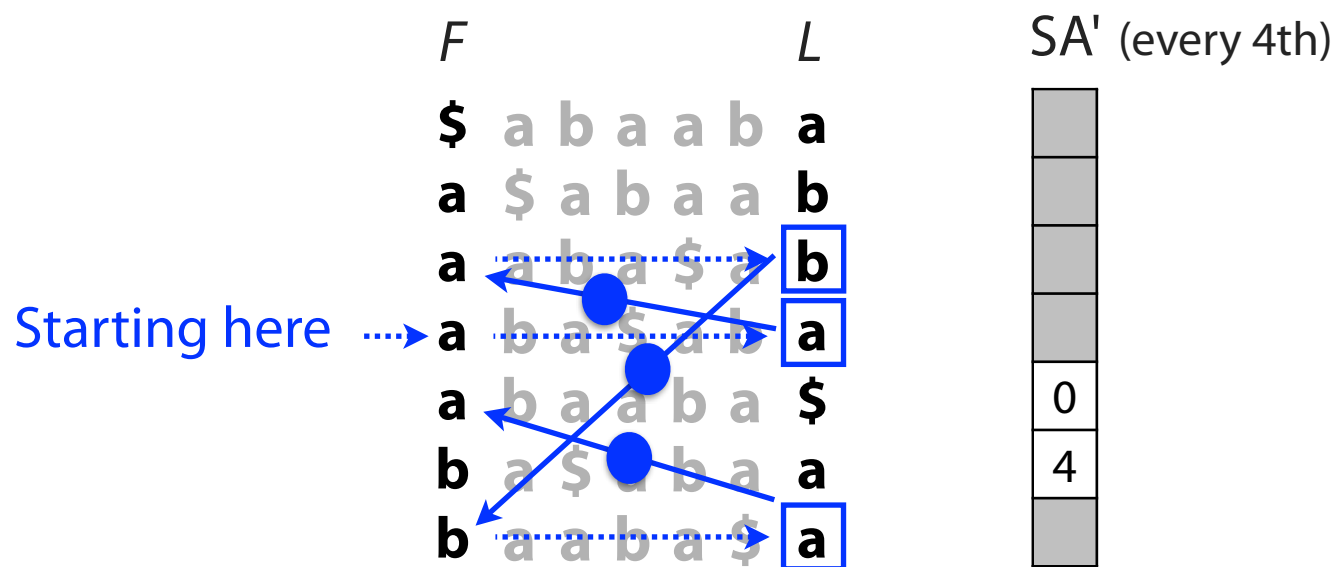
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:



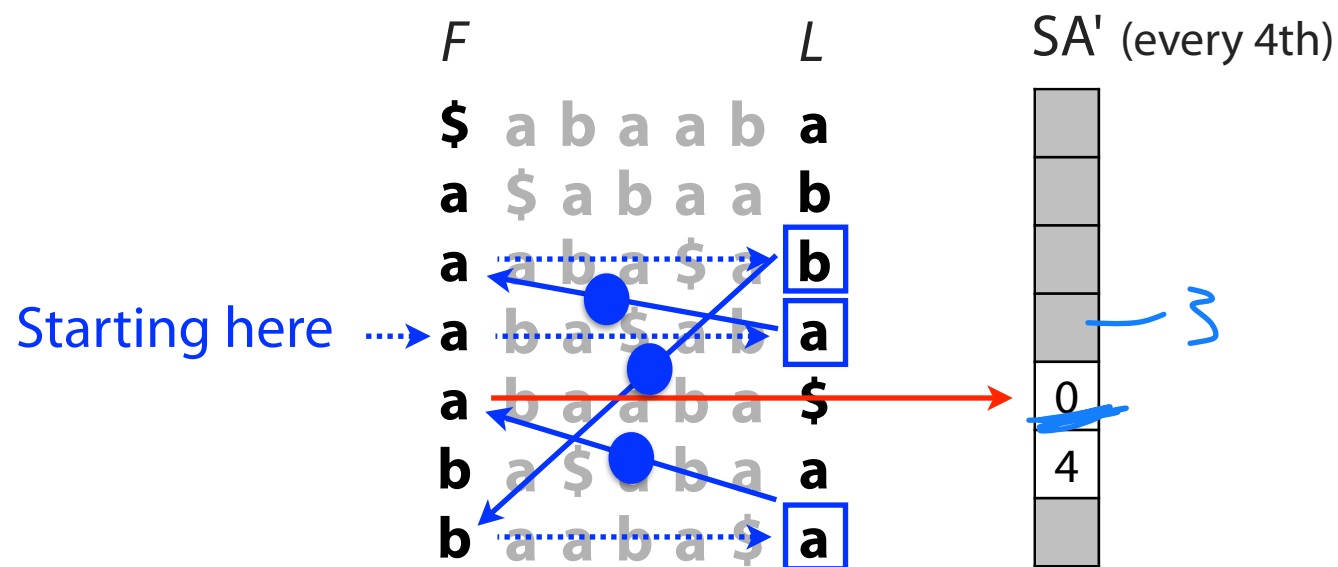
FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:



FM Index: Suffix Array Sampling

Many LF-mapping steps may be required to get to a sampled row:



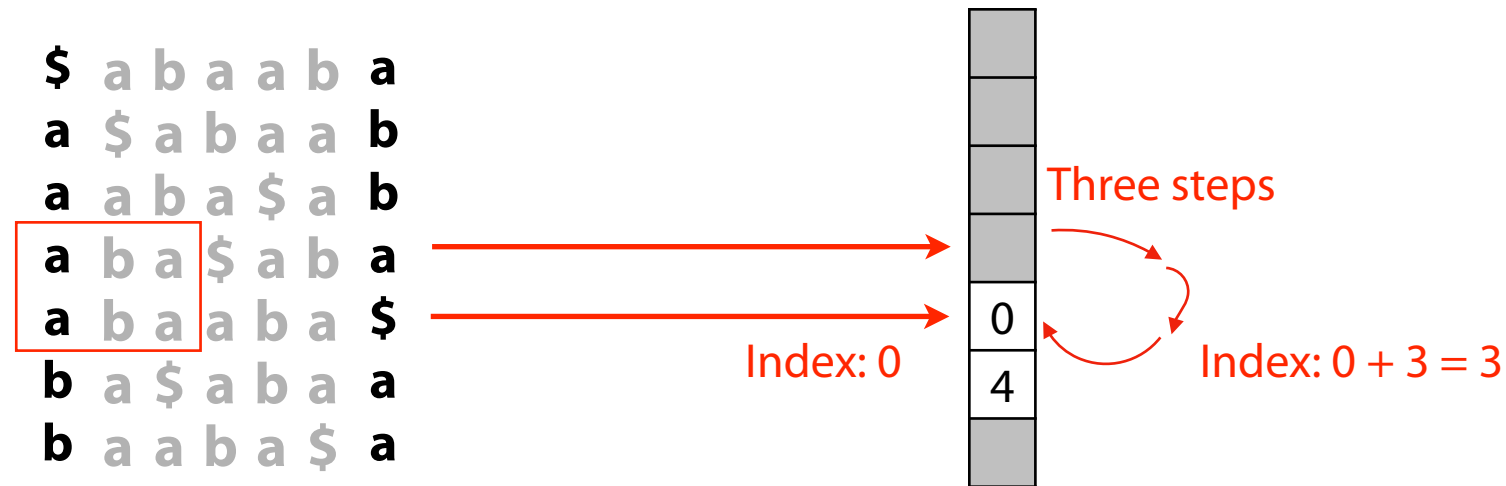
Missing value = 0 (SA val at destination) + 3 (# steps to destination) = **3**



FM Index: Suffix Array Sampling

An index combining the BWT with *a few small auxiliary data structures*

Stores all index positions in T with $O(1)$ extra work to calculate



Lets put all these pieces together...

FM Index: Querying

1) Query for last character in $\#$

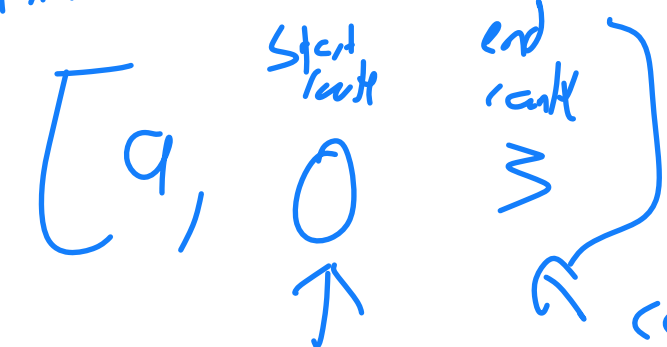
$P = \mathbf{ab} \mathbf{a}$

	F		L
	\$	a b a a b	$\mathbf{a_0}$
$\mathbf{a_0}$	\$	a b a a	\mathbf{b}
$\mathbf{a_1}$	a	b a \$ a	\mathbf{b}
$\mathbf{a_2}$	b	a \$ a b	$\mathbf{a_1}$
$\mathbf{a_3}$	b	a a b a	$\mathbf{\$}$
	\mathbf{b}	a \$ a b a	$\mathbf{a_2}$
	\mathbf{b}	a a b a \$	$\mathbf{a_3}$

indices

tell us what to look up in L

input:



count total # as

First lookup always 0

`pair<int, int> get_frange(string c, int s, int e)`

Input:

`string c`: The char we are looking for in F

`int s`: The starting *rank* value

`int e`: The ending *rank* value

Output:

A pair of values (index start, index end)

What are c , s , and e ?

$a, 0, 3$

What are the output values?

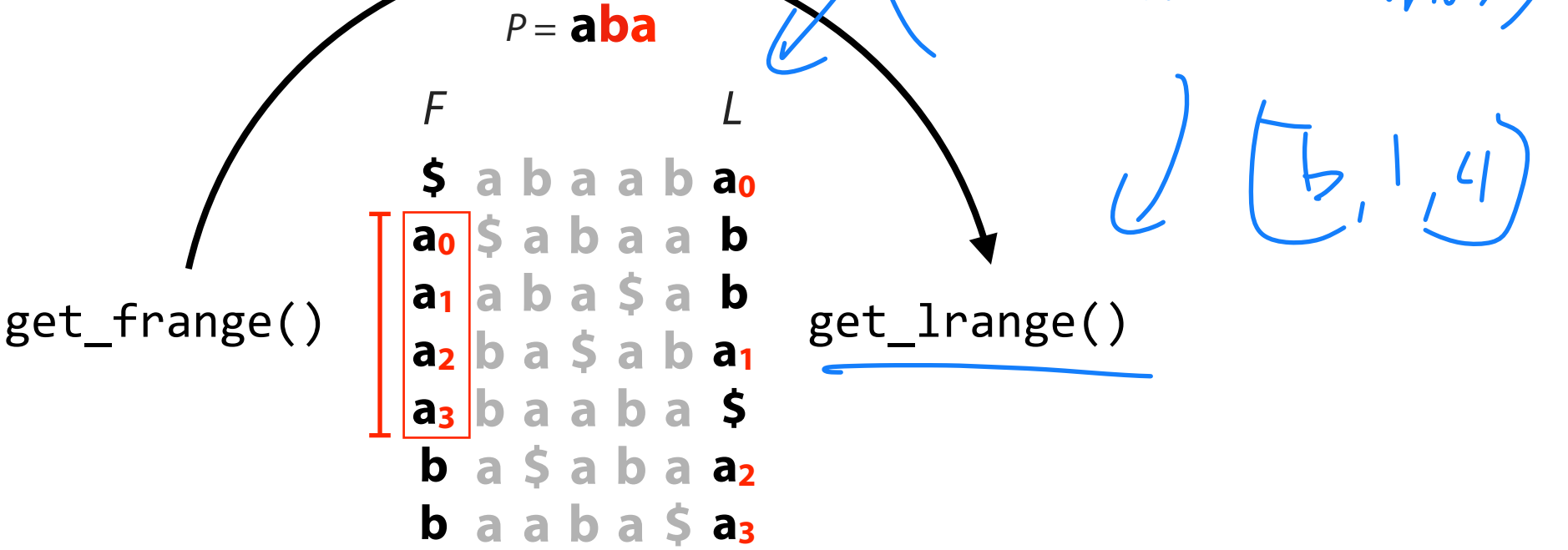
$[1, 4]$

	F	$P = \mathbf{aba}$	L
	$\$$	a b a a b	$\mathbf{a_0}$
<u>1</u>	$\mathbf{a_0}$	$\$$ a b a a	$\mathbf{b_0}$
	$\mathbf{a_1}$	a b a $\$$ a	$\mathbf{b_1}$
	$\mathbf{a_2}$	b a $\$$ a b	$\mathbf{a_1}$
<u>4</u>	$\mathbf{a_3}$	b a a b a	$\mathbf{\$}$
	$\mathbf{b_0}$	a $\$$ a b a	$\mathbf{a_2}$
	$\mathbf{b_1}$	a a b a $\$$	$\mathbf{a_3}$

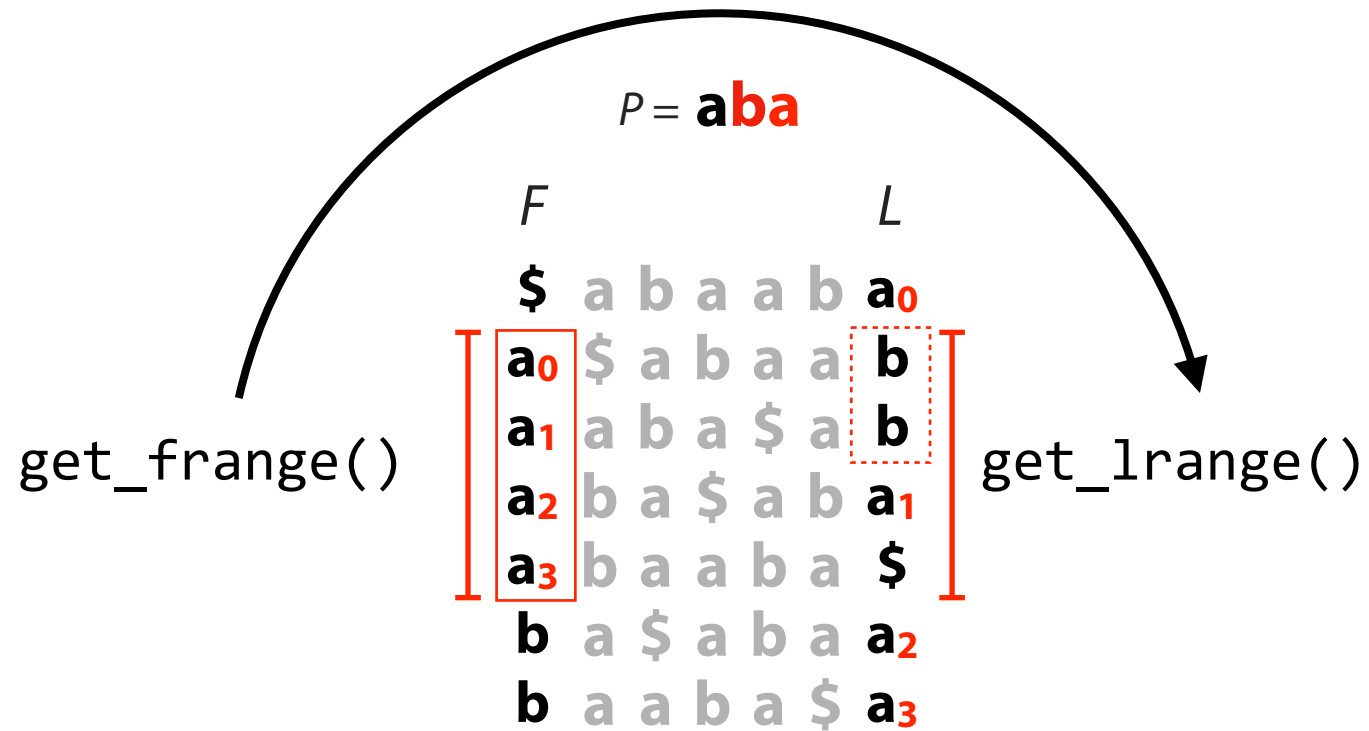
start index

end index

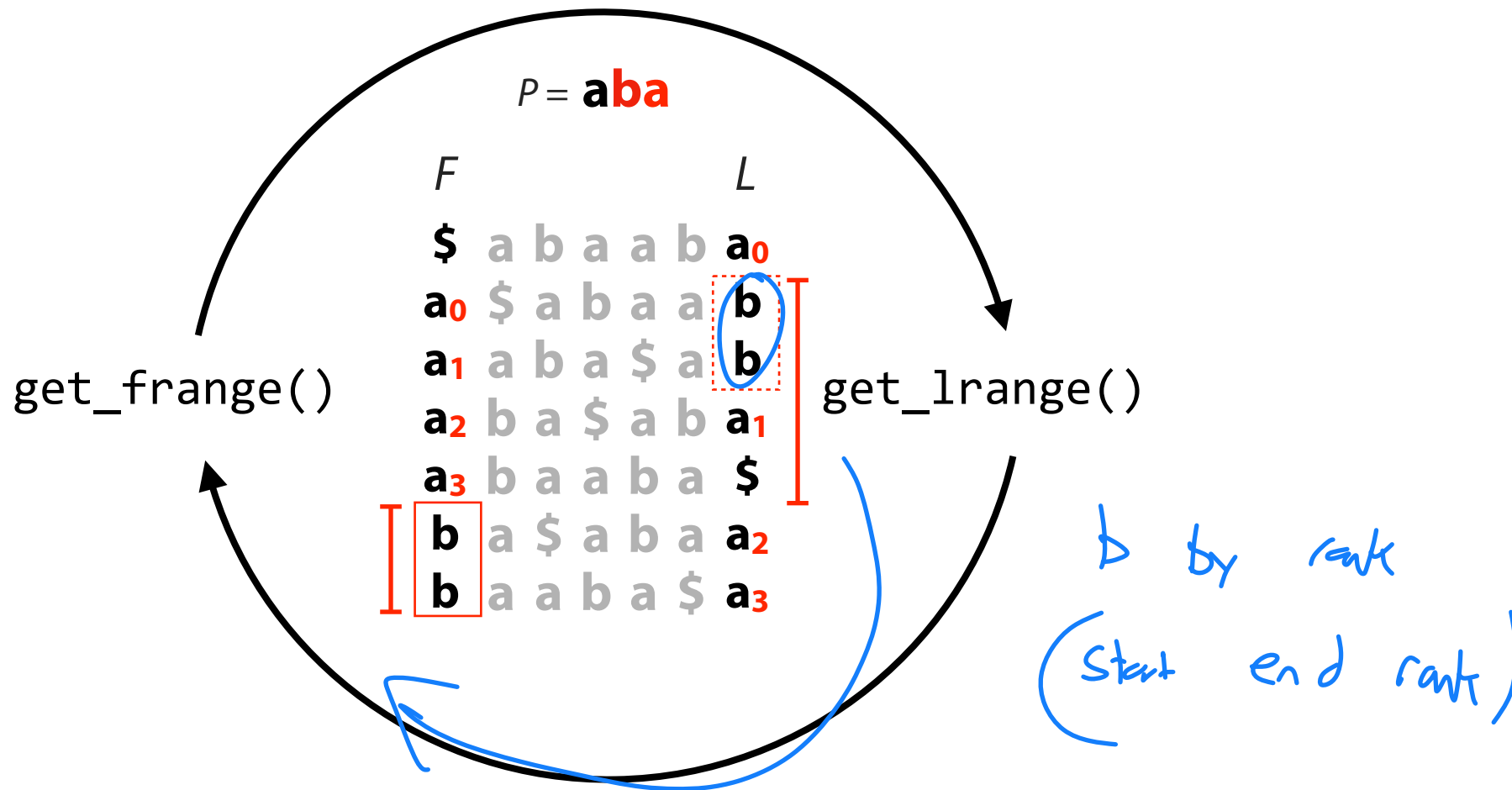
FM Index: Querying



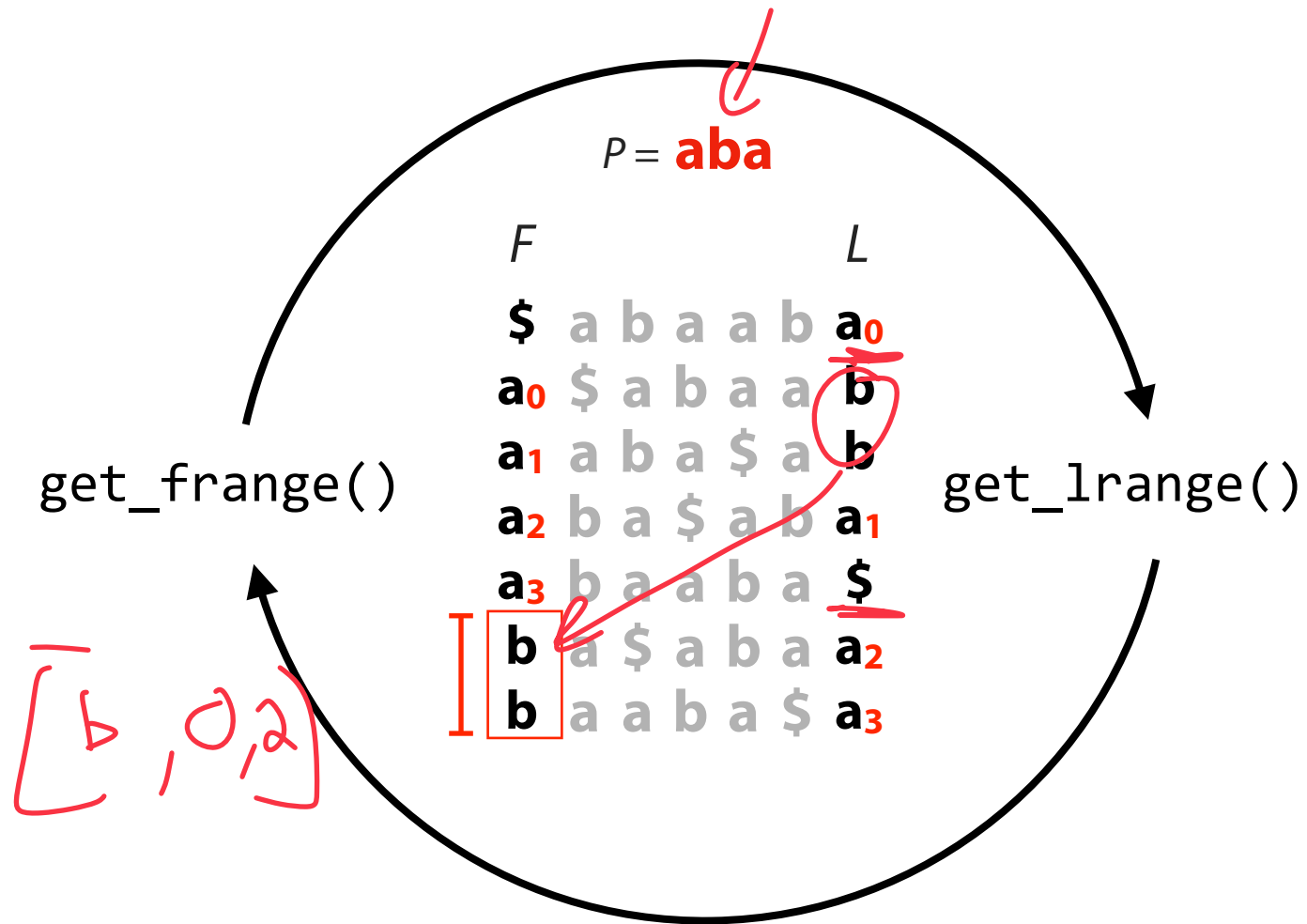
FM Index: Querying



FM Index: Querying

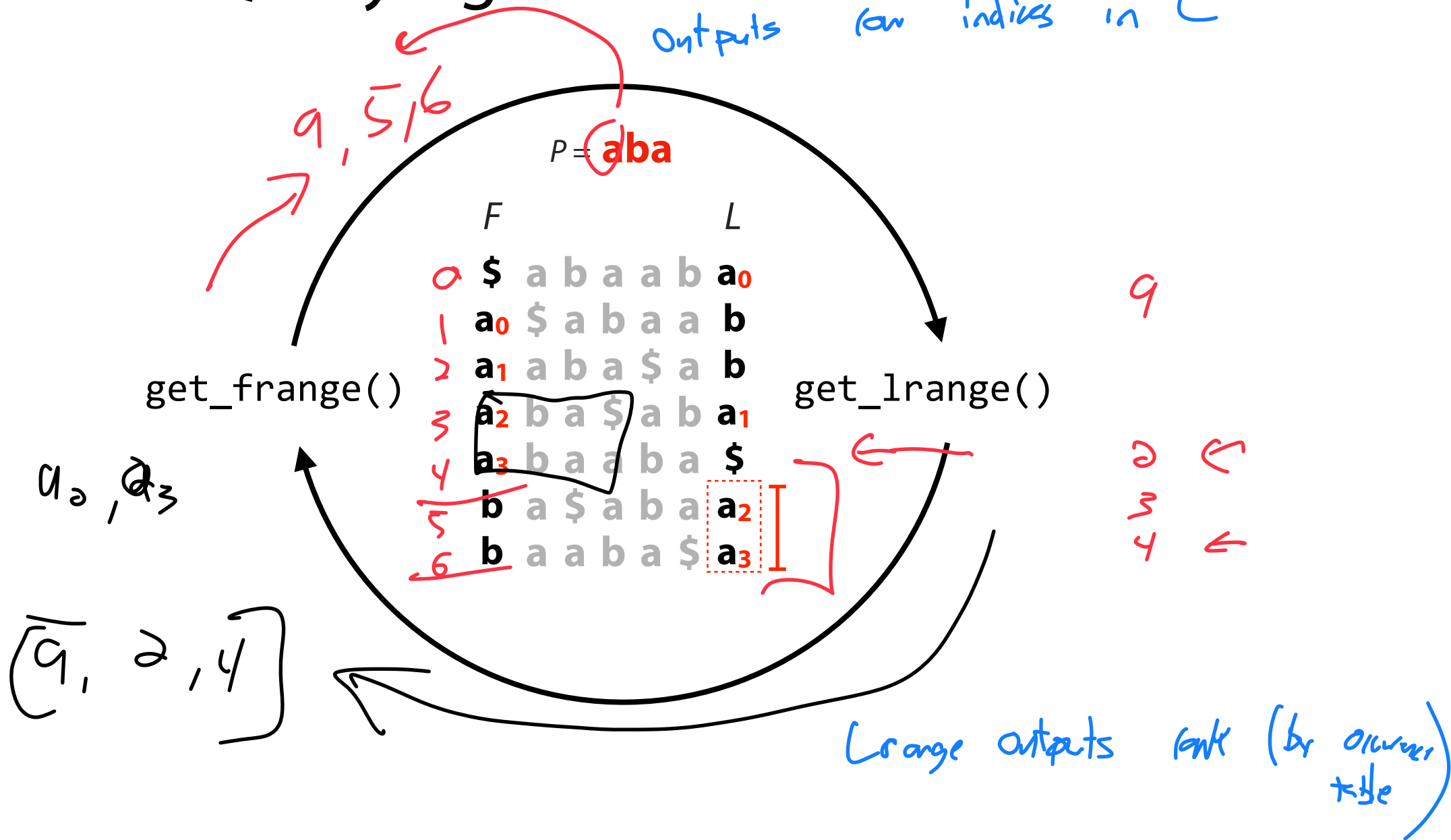


FM Index: Querying



FM Index: Querying

frange
 outputs low indices in L



`pair<int, int> get_frange(string c, int s, int e)`

Input:

string c: The char we are looking for in F

int s: The starting *rank* value

int e: The ending *rank* value

Output:

A pair of values (index start, index end)

What are c, s, and e?

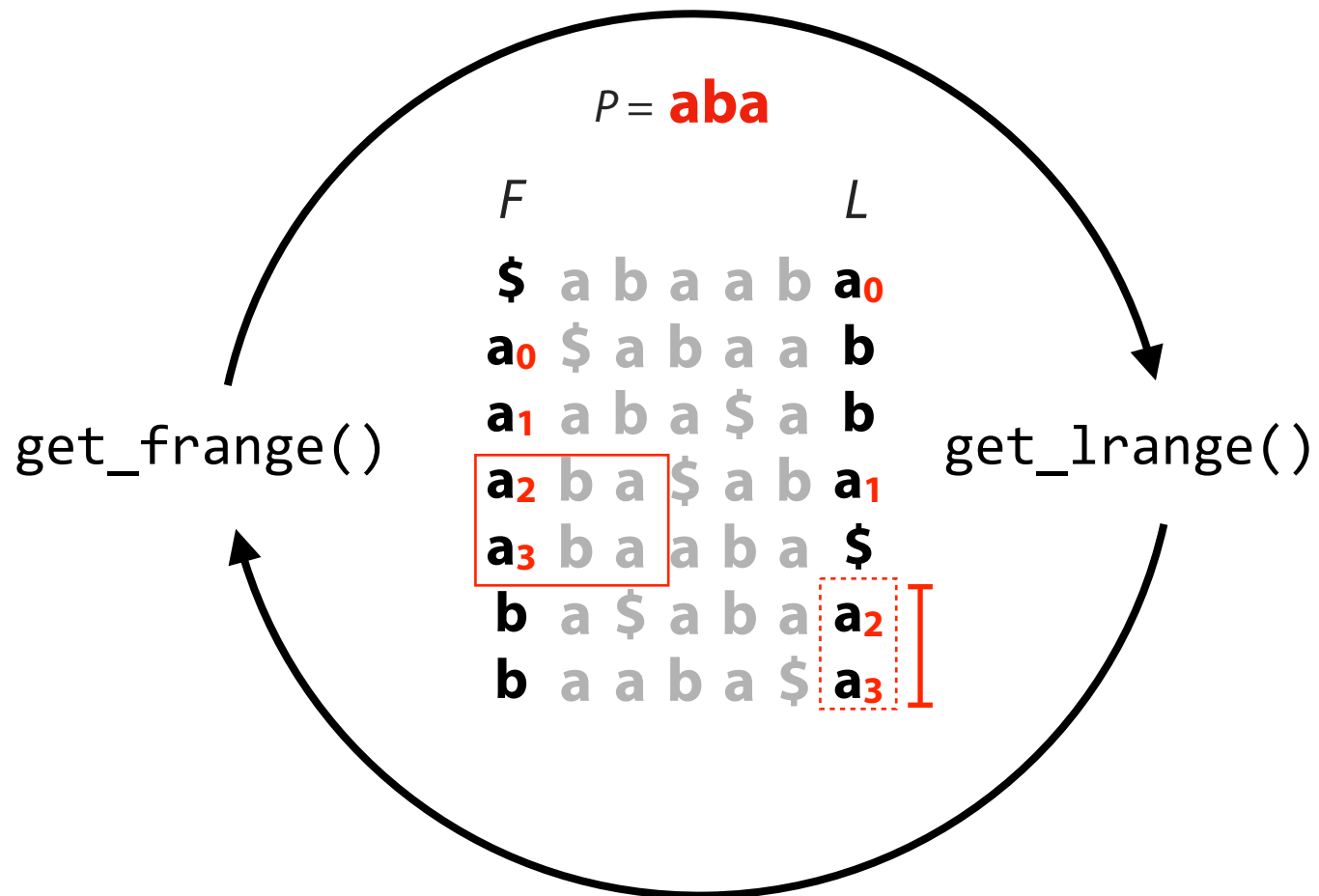
b, 0, 1

What are the output values?

5, 6

F	$P = \mathbf{aba}$	L
\$	a b a a b	a₀
a₀	\$ a b a a	b₀
a₁	a b a \$ a	b₁
a₂	b a \$ a b	a₁
a₃	b a a b a	\$
b₀	a \$ a b a	a₂
b₁	a a b a \$	a₃

FM Index: Querying



get_lrange('a', 5, 6) -> [2, 4]

$P = \mathbf{aba}$  $P = \mathbf{aba}$

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

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

get_frange('a', 2, 3) -> [3, 4]

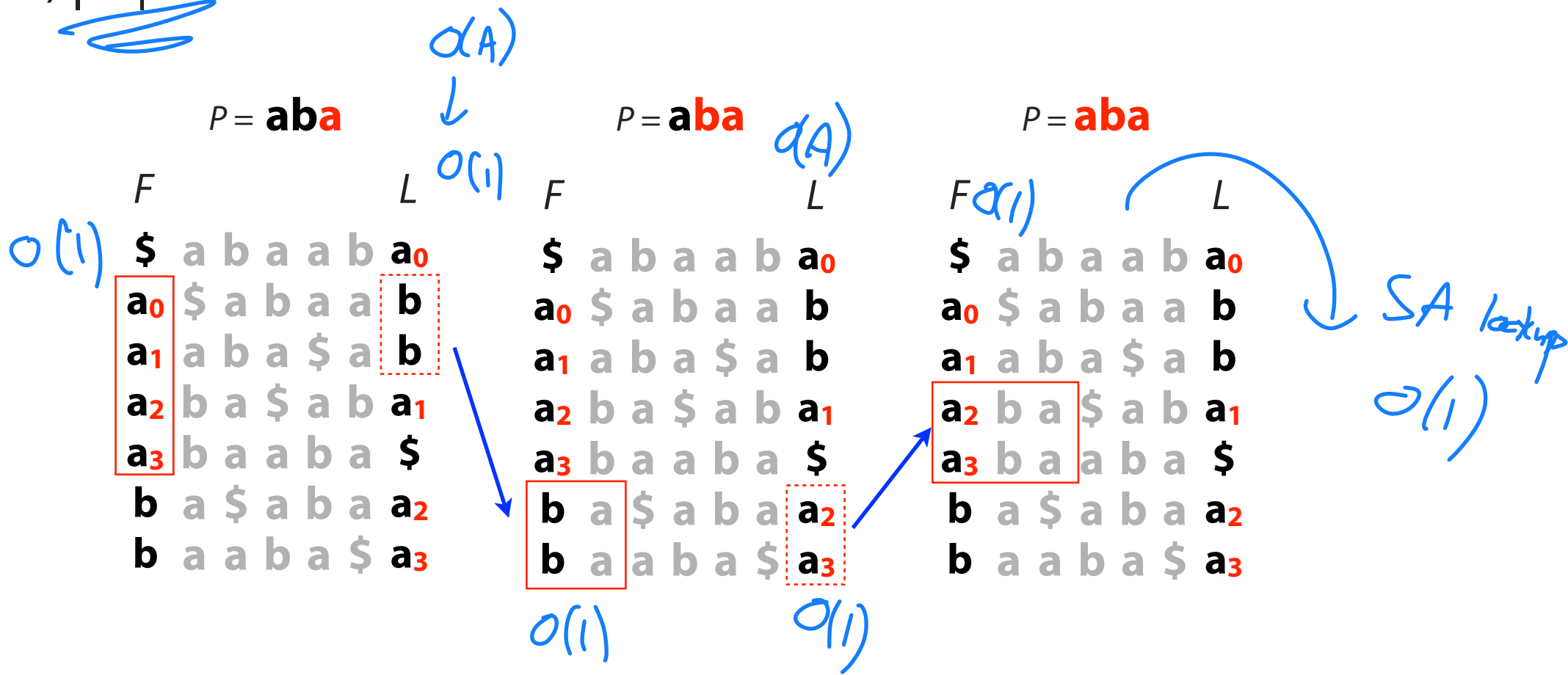
 SA[3] = 3, SA[4] = 0 --> Return {0, 3}

FM Index

OT cost is ~~A~~



$|T| = m, |P| = n$



Finding all matches of P occurs in T in FM Index is $O(n)$ time

Assignment 9: a_fmi

Learning Objective:

Construct a full FM Index

Implement exact pattern matching on a FM Index

Consider: How would you modify the provided code to handle sub-sampling in the Occurrence Table (OT) or Suffix Array (SA)?

FM Index

Components of FM Index: (blue indicates what we can adjust by changing a & b)

- First column (F): $\sim |\Sigma|$ integers
- Last column (L): m characters
- SA sample: $m \cdot a$ integers, a is fraction of SA elements kept
- OT Checkpoints: $m \cdot |\Sigma| \cdot b$ integers, b is fraction of tallies kept

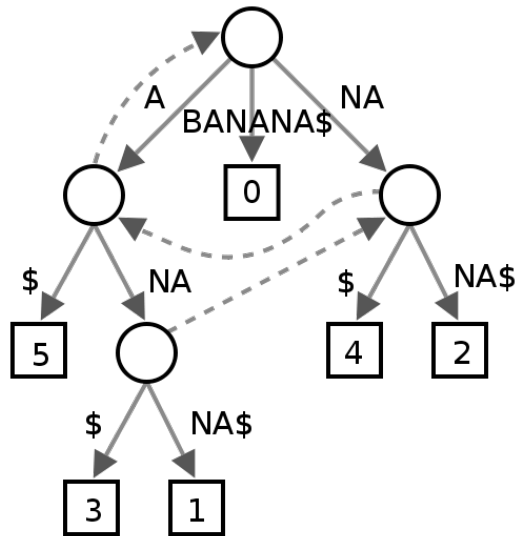
For DNA alphabet (2 bits / nt), $T =$ human genome, $a = 1/32$, $b = 1/128$:

- First column (F): 16 bytes
- Last column (L): 2 bits * 3 billion chars = 750 MB
- SA sample: 3 billion chars * 4 bytes / 32 = ~ 400 MB
- OT Checkpoints: 3 billion * 4 alphabet chars * 4 bytes / 128 = ~ 400 MB

Total \approx 1.5 GB
~0.5 bytes per input char

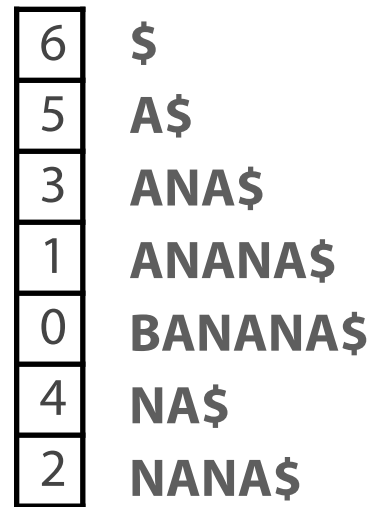
? 5 GB
100 MB
20 MB
5 B

FM Index: Small Memory Footprint



Suffix tree

≥ 45 GB



Suffix array

≥ 12 GB



FM Index

~ 1.5 GB

Suffix-Based Index Bounds



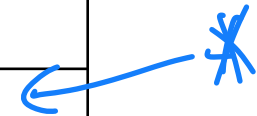
	Suffix tree	Suffix array	FM Index
Time: Does P occur?			
Time: Count k occurrences of P			
Time: Report k locations of P			
Space			
Needs T ?			
Bytes per input character			

$$m = |T|, n = |P|, k = \# \text{ occurrences of } P \text{ in } T$$

Suffix-Based Index Bounds



	Suffix tree	Suffix array	FM Index
Time: Does P occur?	$O(n)$	$O(n \log m)$	$O(n)$
Time: Count k occurrences of P	$O(n + k)$	$O(n \log m)$	$O(n)$
Time: Report k locations of P	$O(n + k)$	$O(n \log m + k)$	$O(n + k)$
Space	$O(m)$	$O(m)$	$O(m)$
Needs T ?	yes	yes	no
Bytes per input character	>15	~4	~0.5



$$m = |T|, n = |P|, k = \# \text{ occurrences of } P \text{ in } T$$