The BWT as a compressed index

Off-line text searching

We have a collection of documents (think of project Gutemberg or Wikipedia) and we want to search for information (substrings) in it.

Since the collection is know in advance, we speed-up the search building an index for the collection

The Suffix Array

The Suffix Array is the simplest indexing data structure supporting fast pattern searching

Consider for example

T = **swiss·miss·missing**

The Suffix Array

swiss·miss·missing

We consider all the suffixes of the input text

```
 1  swiss·miss·missing
 2  wiss·miss·missing
 3  iss·miss·missing
 4  ss·miss·missing
 5  s·miss·missing
 6  ·miss·missing
 7  miss·missing
 8  iss·missing
 9  ss·missing
10  s·missing
11  ·missing
12  missing
13  issing
14  ssing
15  sing
16  ing
17  ng
18  g
```

The Suffix Array

swiss·miss·missing

We consider all the suffixes of the input text

... and we sort them in lexicographic order

```
 6   ·miss·missing
11   ·missing
18   g
16   ing
 3   iss·miss·missing
 8   iss·missing
13   issing
 7   miss·missing
12   missing
17   ng
 5   s·miss·missing
10   s·missing
15   sing
 4   ss·miss·missing
 9   ss·missing
14   ssing
 1   swiss·miss·missing
 2   wiss·miss·missing
```

The Suffix Array

swiss·miss·missing

Using binary search we find that miss appears starting at positions 7 and 12

```
 6   ·miss·missing
11   ·missing
18   g
16   ing
 3   iss·miss·missing
 8   iss·missing
13   issing
 7   miss·missing
12   missing
17   ng
 5   s·miss·missing
10   s·missing
15   sing
 4   ss·miss·missing
 9   ss·missing
14   ssing
 1   swiss·miss·missing
 2   wiss·miss·missing
```

Using the suffix array we can find all the occurrences of any pattern P in T using binary search in O(|P| log |T|) time.

Enriching the suffix array with additional information we can reduce search time to O(|P| + log |T|).

The Suffix Array

swiss·miss·missing

To represent the Suffix Array we use |T| integers in the range 1 ... |T|

⇒ |T| log|T| bits

6	·miss·missing
11	·missing
18	g
16	ing
3	iss·miss·missing
8	iss·missing
13	issing
7	miss·missing
12	missing
17	ng
5	s·miss·missing
10	s·missing
15	sing
4	ss·miss·missing
9	ss·missing
14	ssing
1	swiss·miss·missing
2	wiss·miss·missing

Burrows-Wheeler Transform

swiss·miss·missing

To compute the BWT we go on transforming each suffix into a cyclic shift of T.

The last column of the resulting matrix is the BWT.

6	·miss·missingswis	s
11	·missingswiss·mis	s
18	gswiss·miss·missi	n
16	ingswiss·miss·mis	s
3	iss·miss·missings	w
8	iss·missingswiss·	m
13	issingswiss·miss·	m
7	miss·missingswiss	·
12	missingswiss·miss	·
17	ngswiss·miss·miss	i
5	s·miss·missingswi	s
10	s·missingswiss·mi	s
15	singswiss·miss·mi	s
4	ss·miss·missingsw	i
9	ss·missingswiss·m	i
14	ssingswiss·miss·m	i
1	swiss·miss·missin	g
2	wiss·miss·missing	s

Burrows-Wheeler Transform

swiss·miss·missing

We discard the Suffix Array but we keep track of the row containing the original string

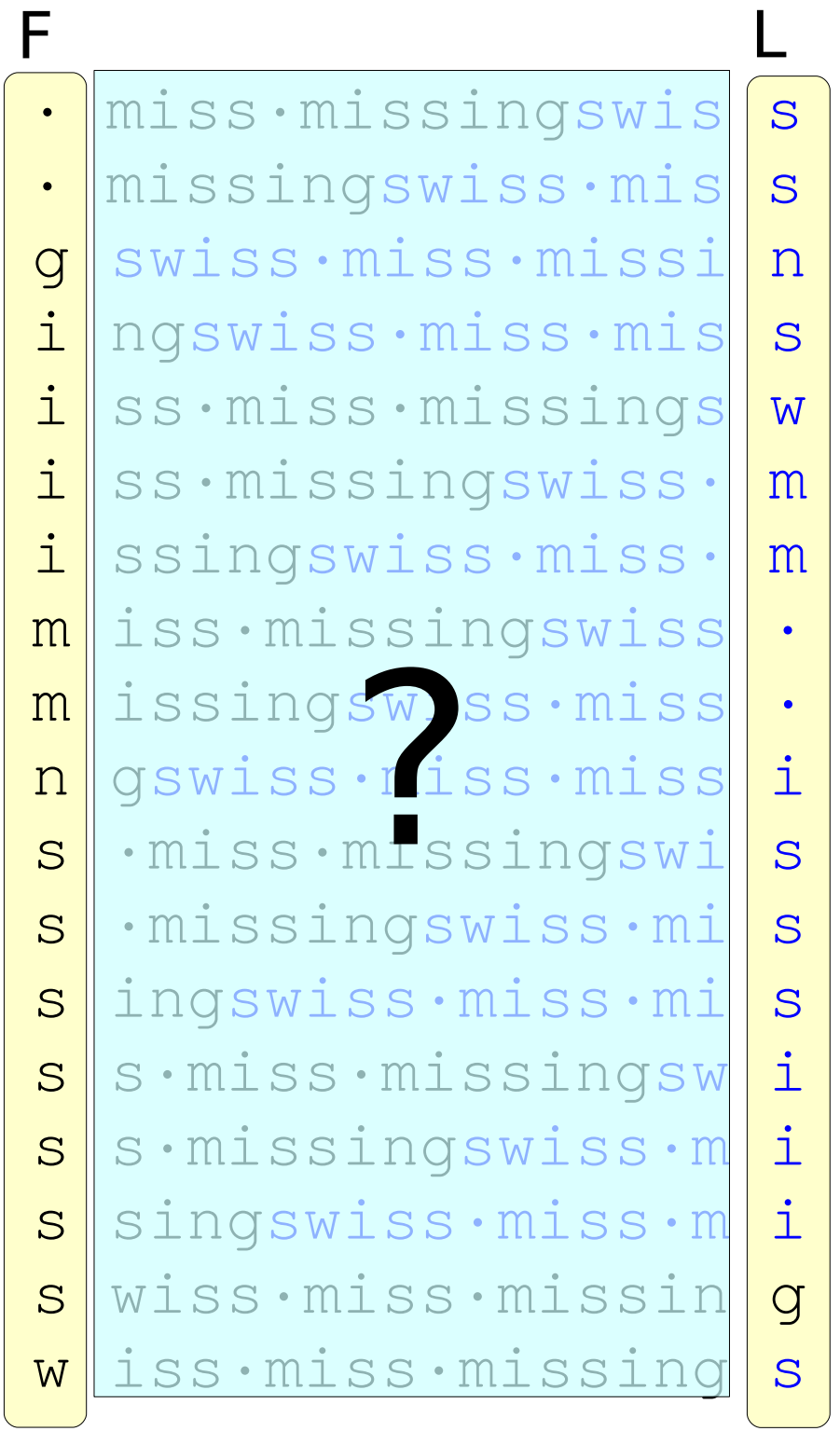
6	·miss·missingswis	s
11	·missingswiss·mis	s
18	gswiss·miss·missi	n
16	ingswiss·miss·mis	s
3	iss·miss·missings	w
8	iss·missingswiss·	m
13	issingswiss·miss·	m
7	miss·missingswiss	·
12	missingswiss·miss	·
17	ngswiss·miss·miss	i
5	s·miss·missingswi	s
10	s·missingswiss·mi	s
15	singswiss·miss·mi	s
4	ss·miss·missingsw	i
9	ss·missingswiss·m	i
14	ssingswiss·miss·m	i
1	**swiss·miss·missin**	**g**
2	wiss·miss·missing	s

Search using the BWT

Suppose we want to count the occurrences of mis

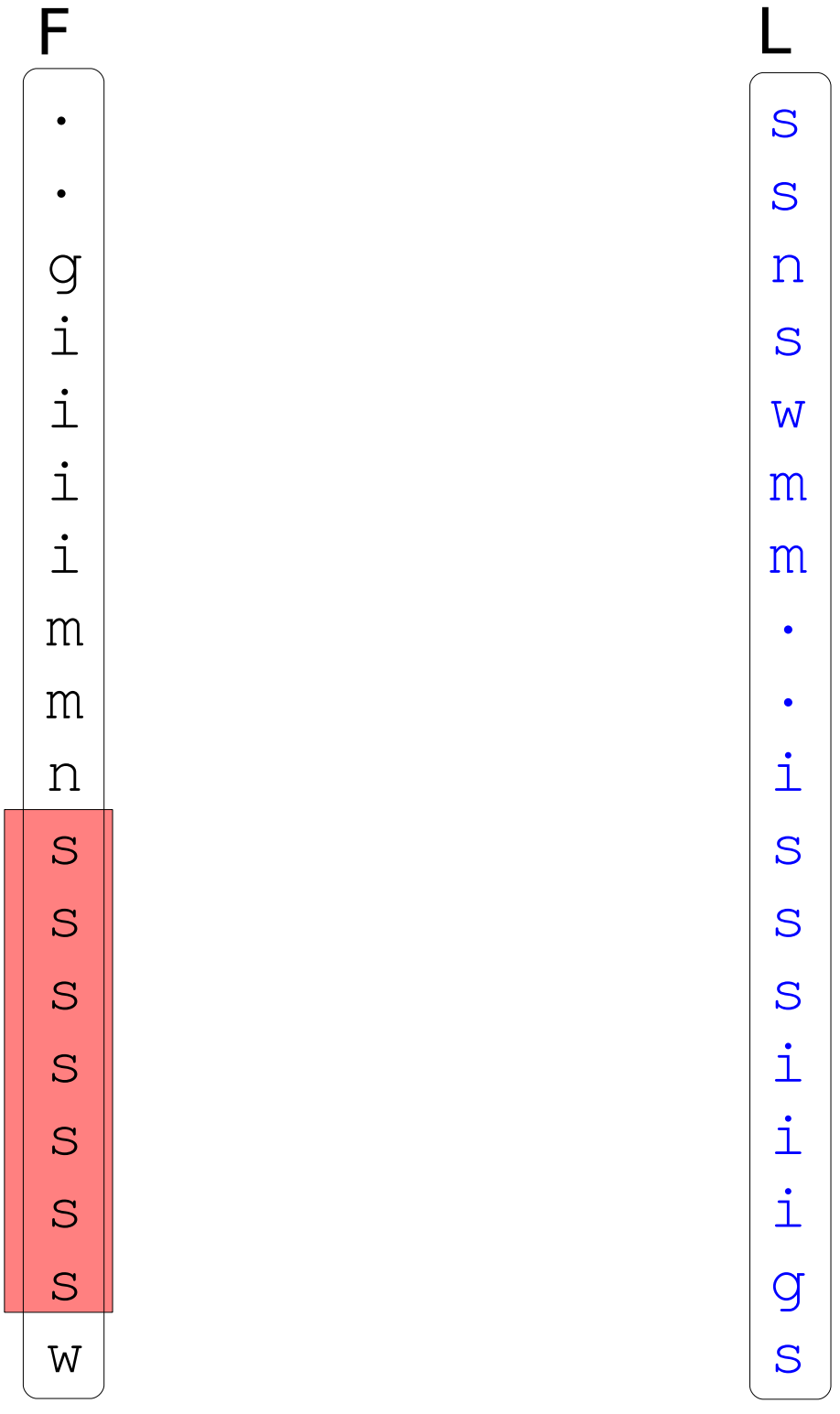
Working with only F and L we successively find the range of rows prefixed by:

s, is, mis



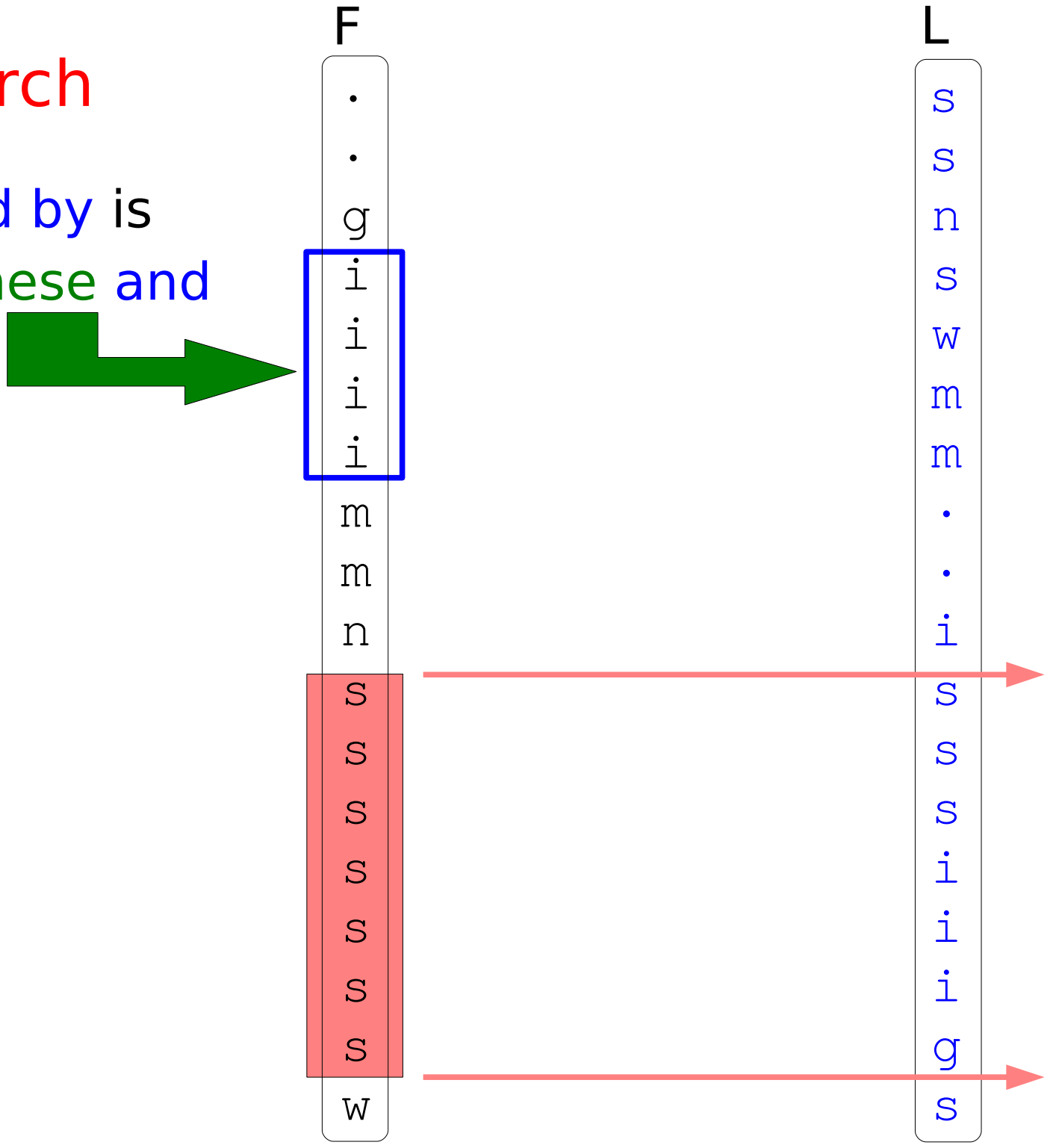
Backward search

The rows prefixed by s are easy to find



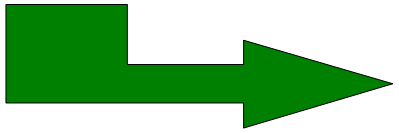
Backward search

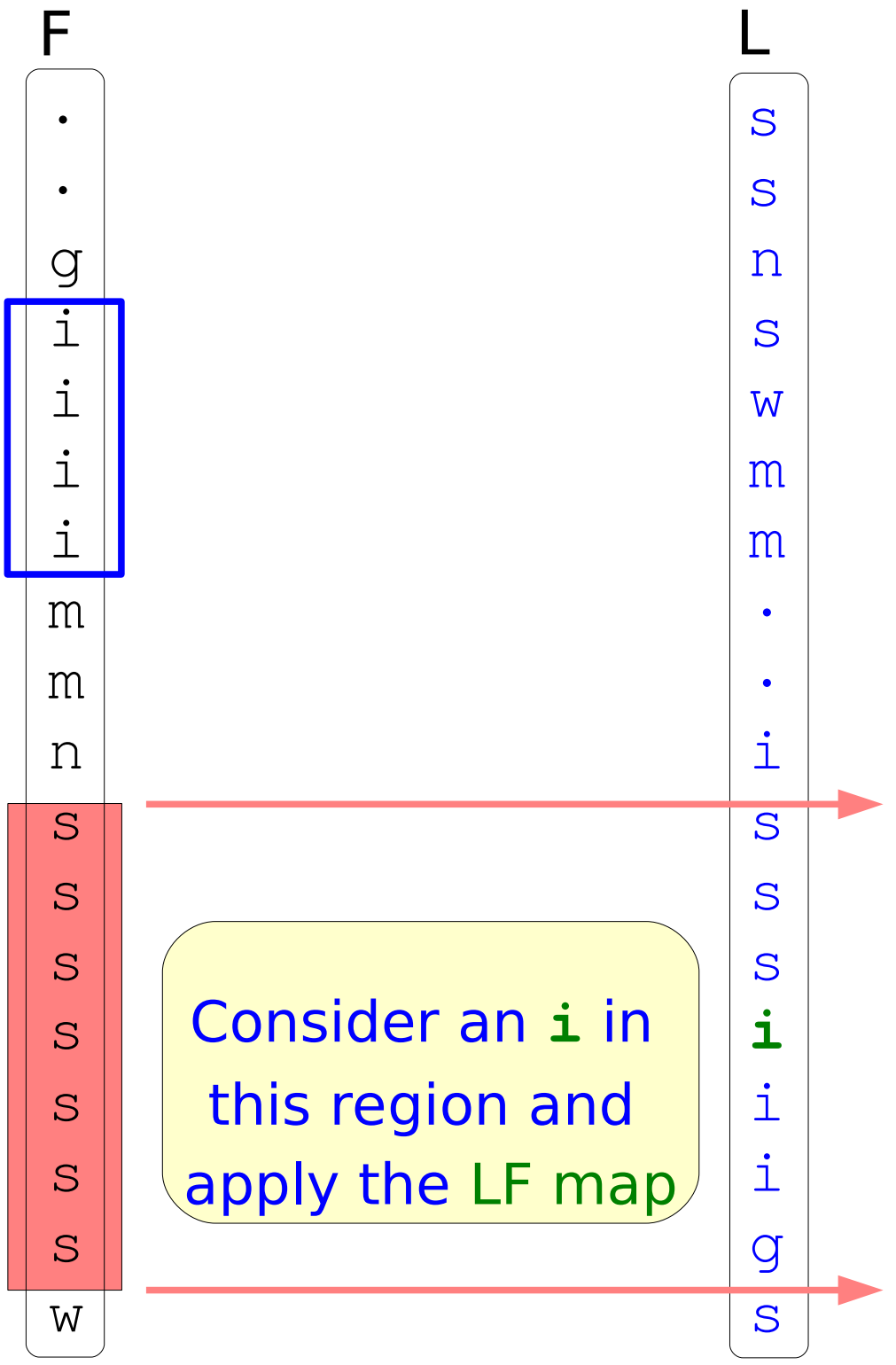
The rows prefixed by is are a subset of these and are consecutive.



Backward search

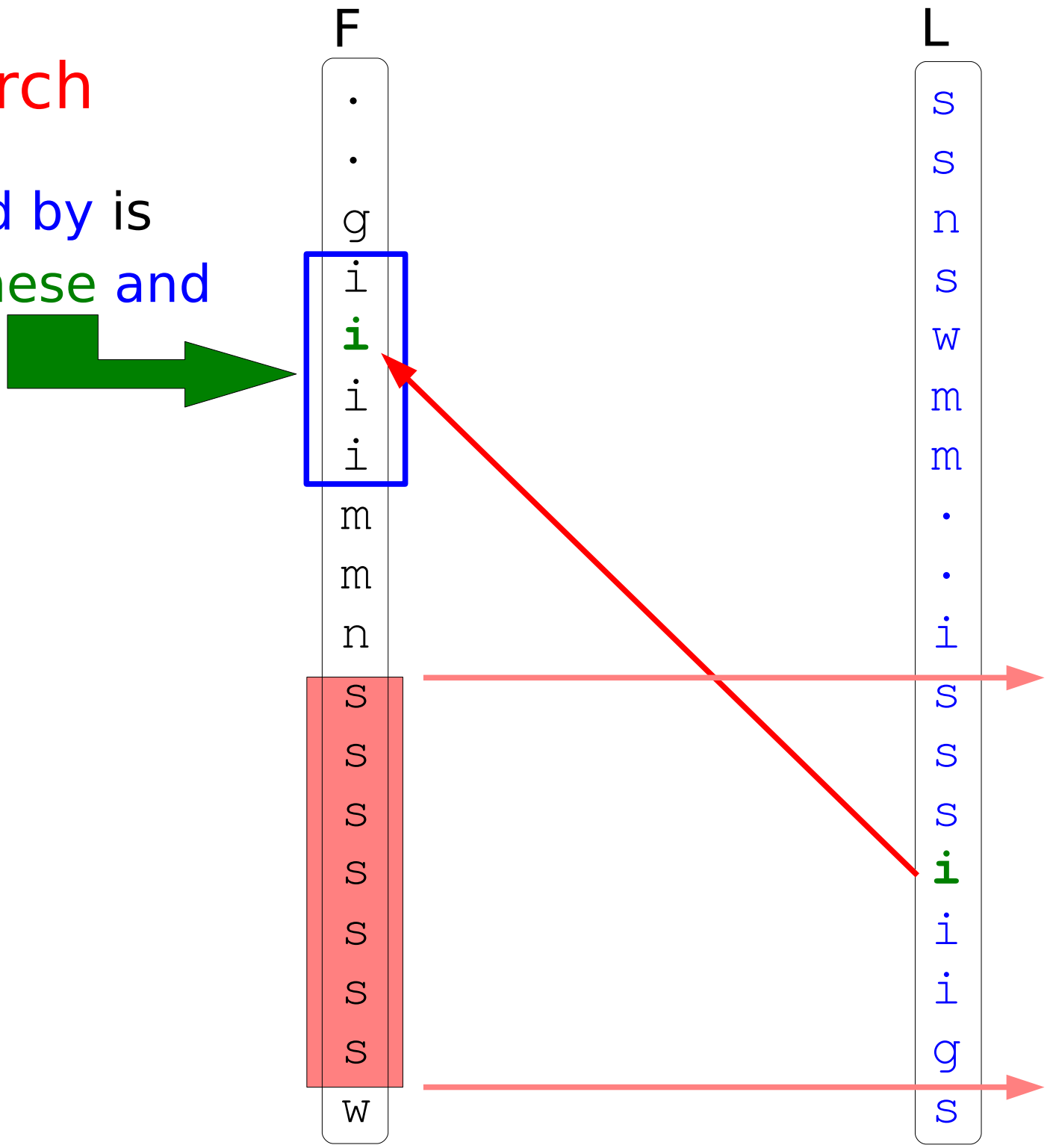
The rows prefixed by **i** are a subset of these and are consecutive.



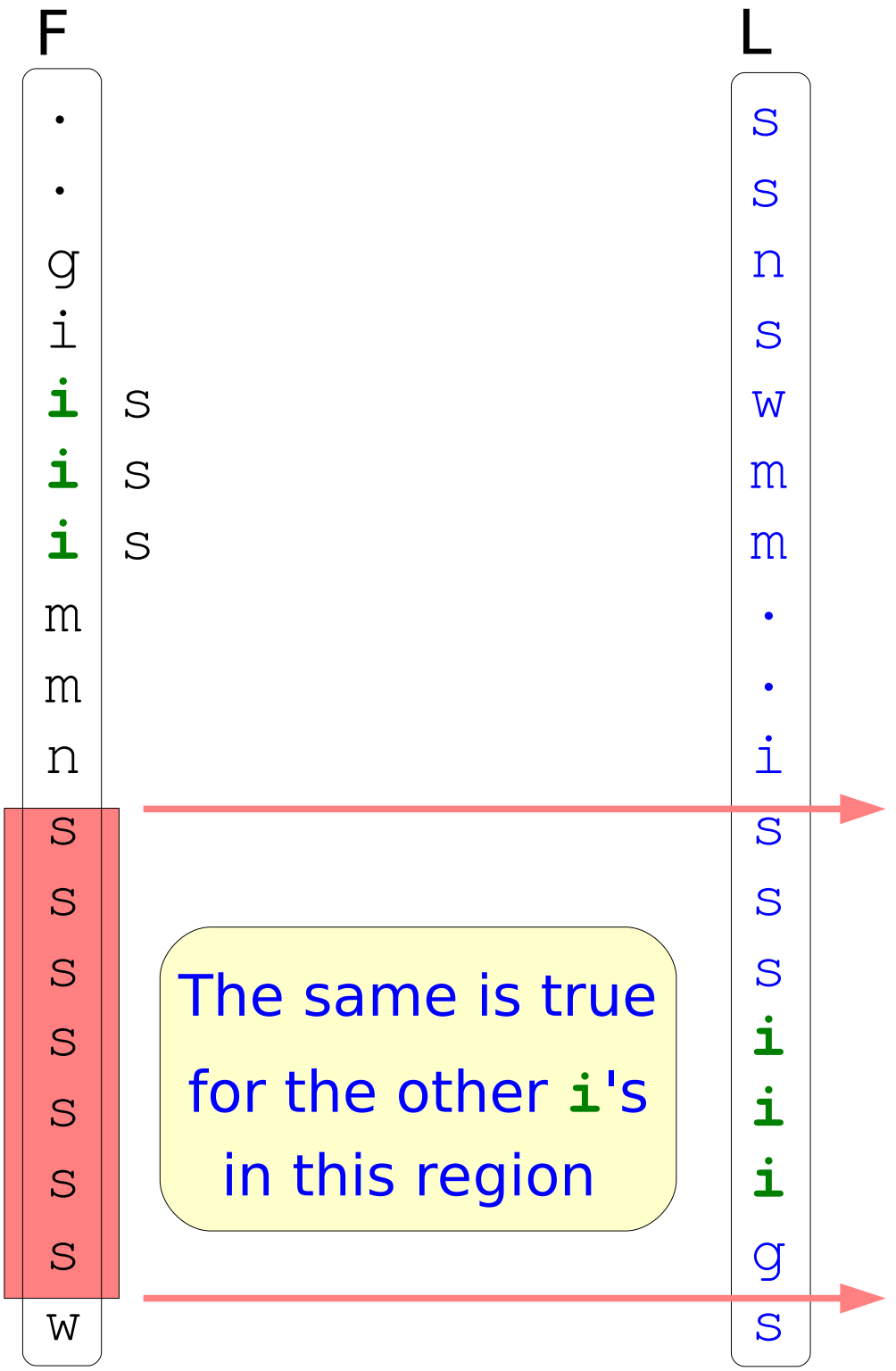


Backward search

The rows prefixed by **i** are a subset of these and are consecutive.

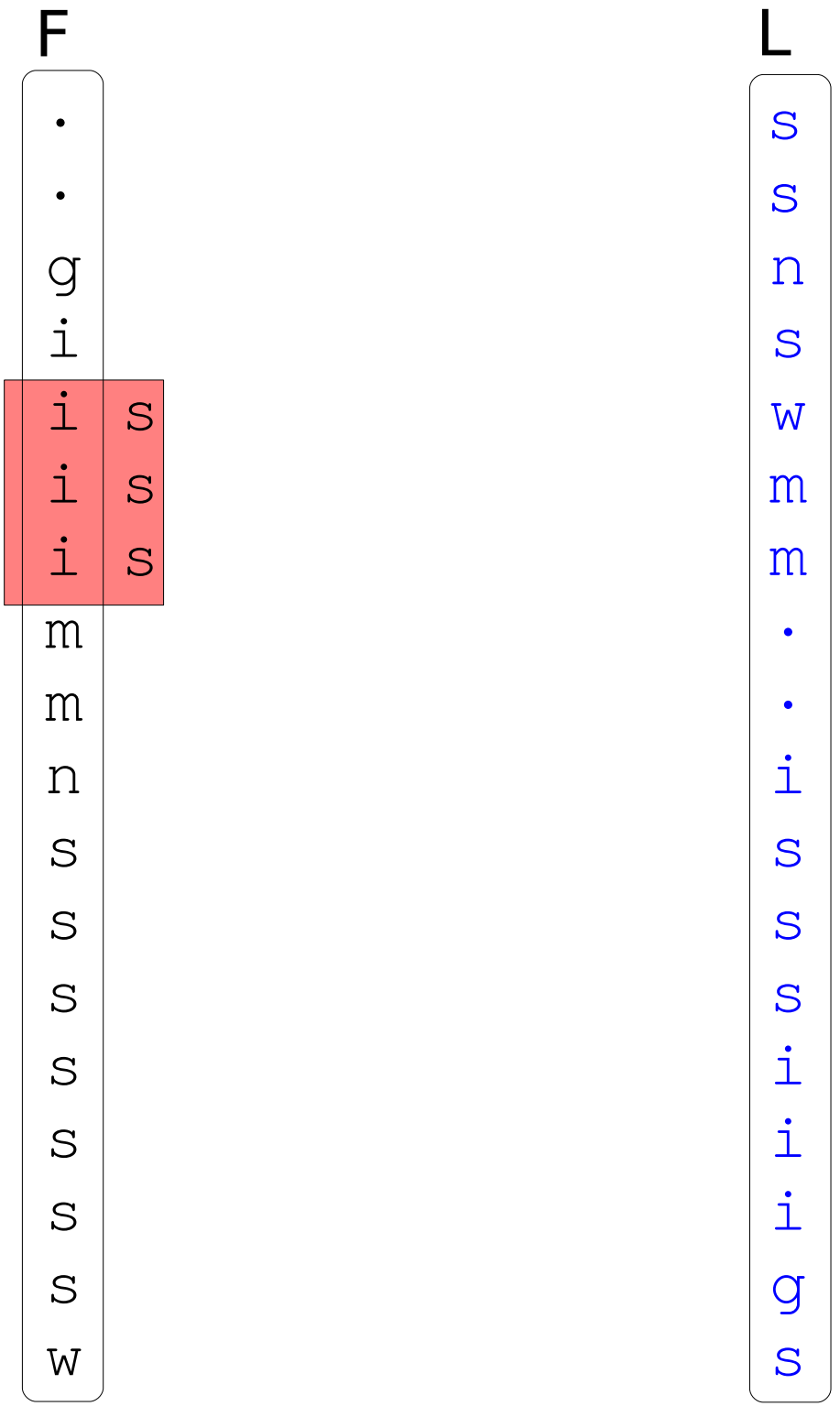


Backward search



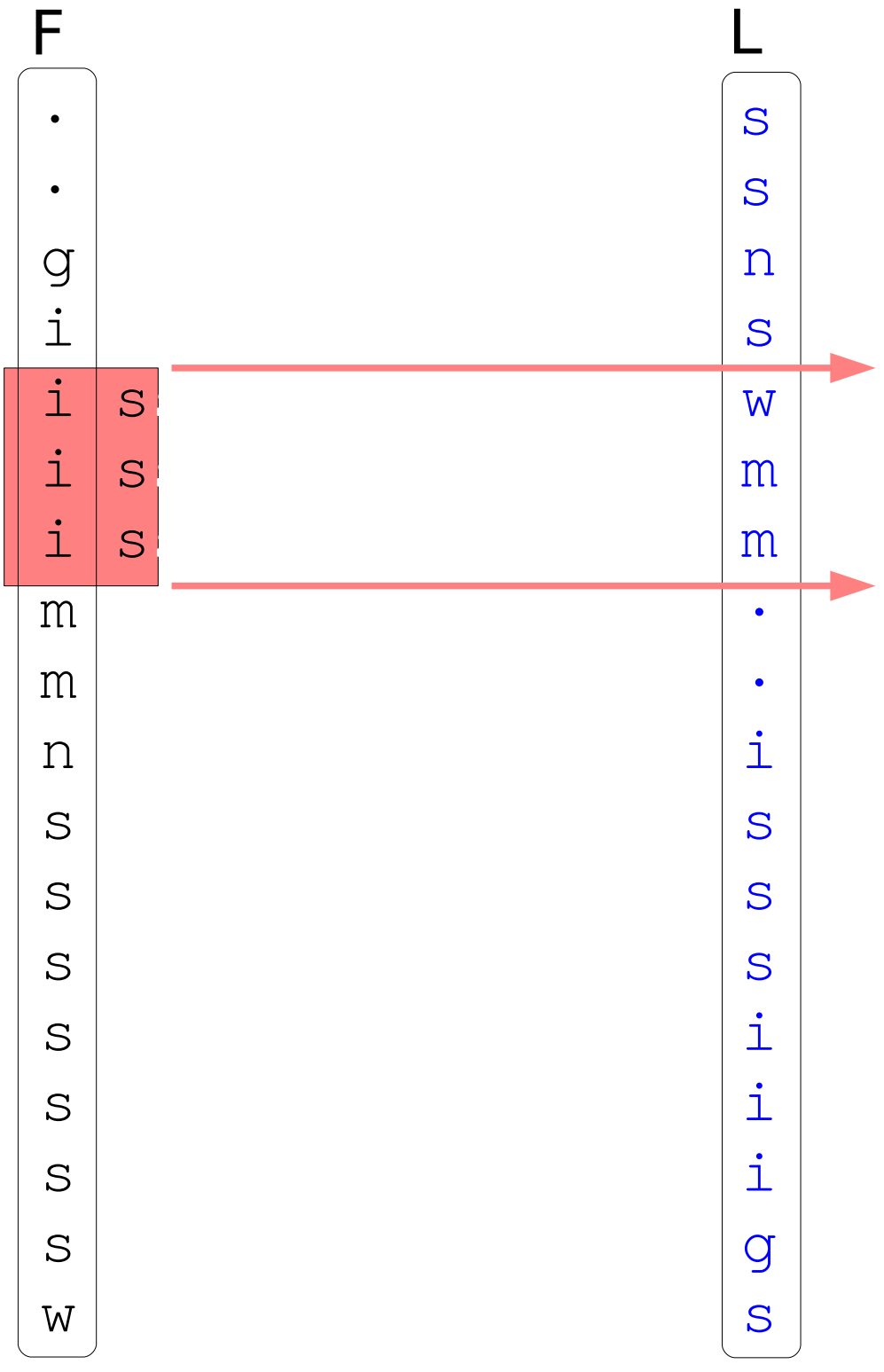
Backward search

We have found the rows prefixed by **is**!



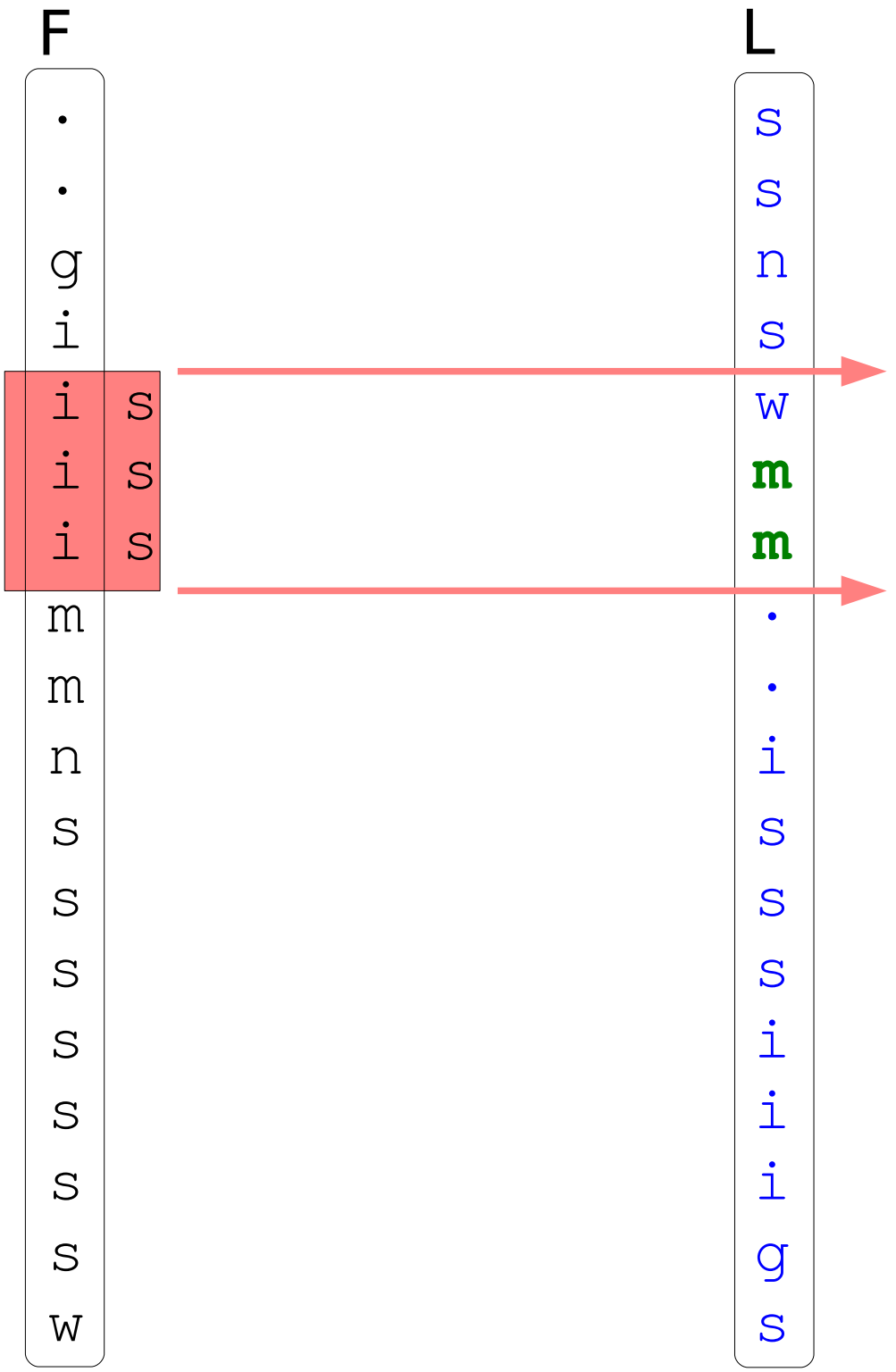
Backward search

To find the rows prefixed by **mis** we proceed in the same way



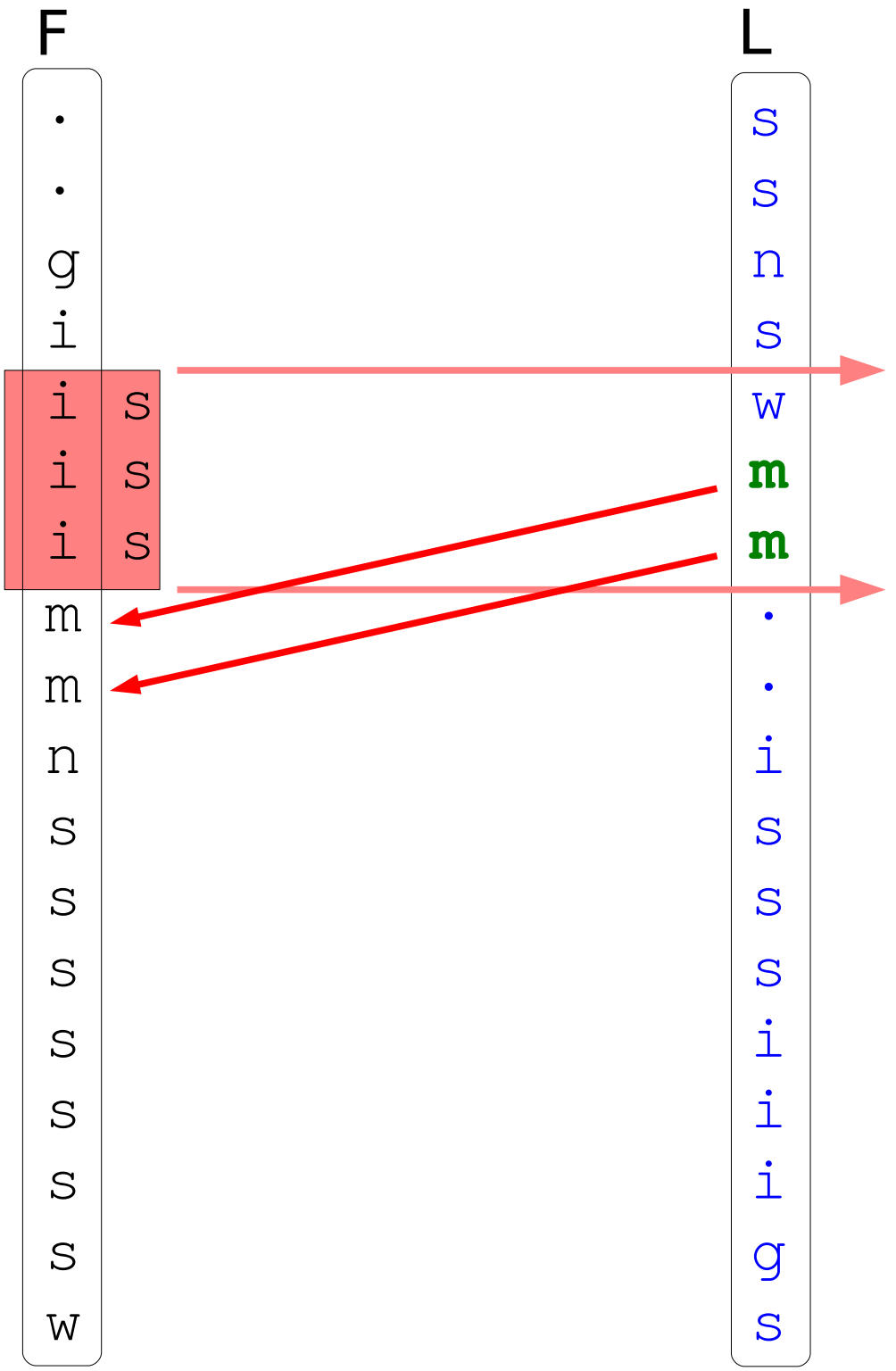
Backward search

To find the rows prefixed by **mis** we proceed in the same way



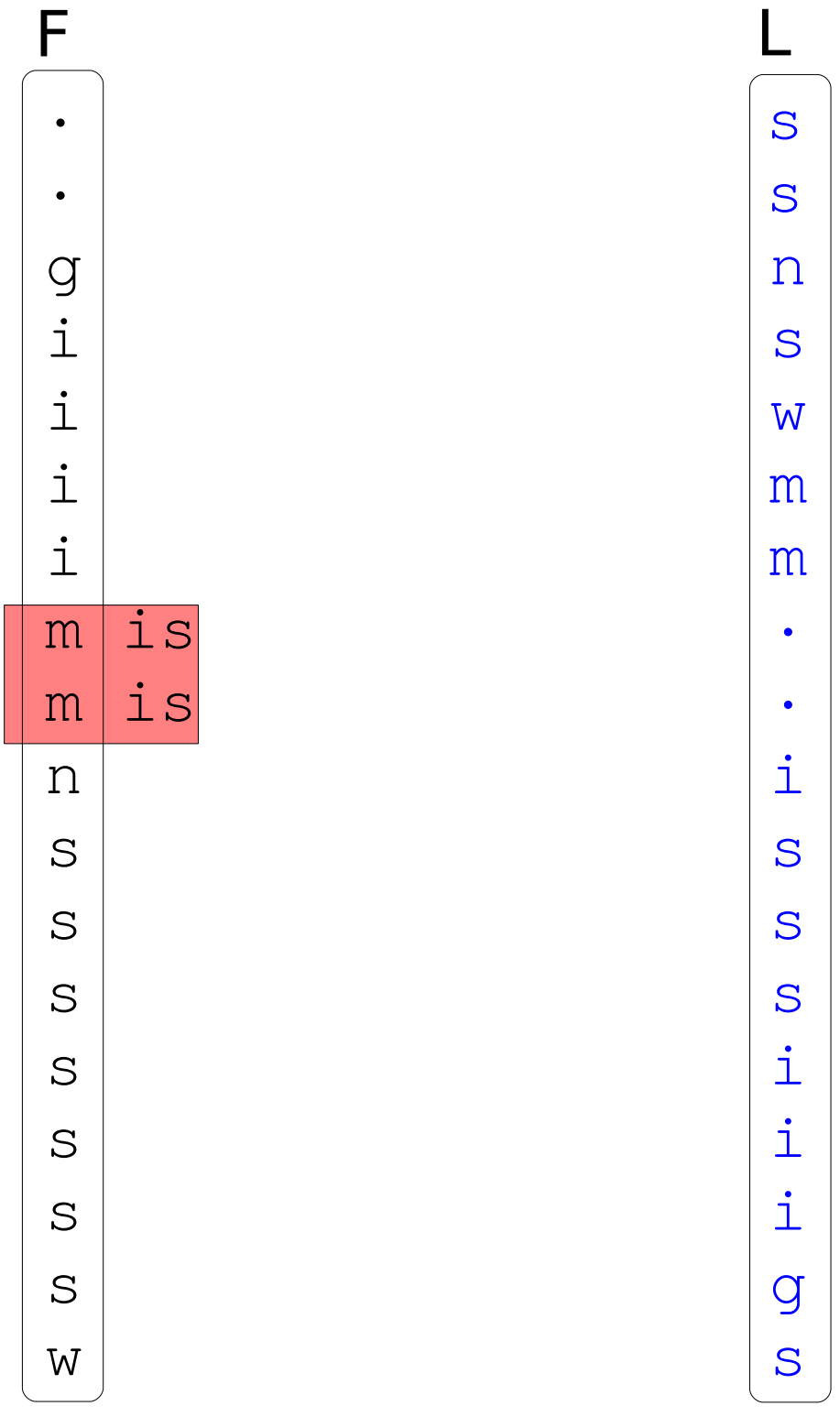
Backward search

To find the rows prefixed by **mis** we proceed in the same way



Backward search

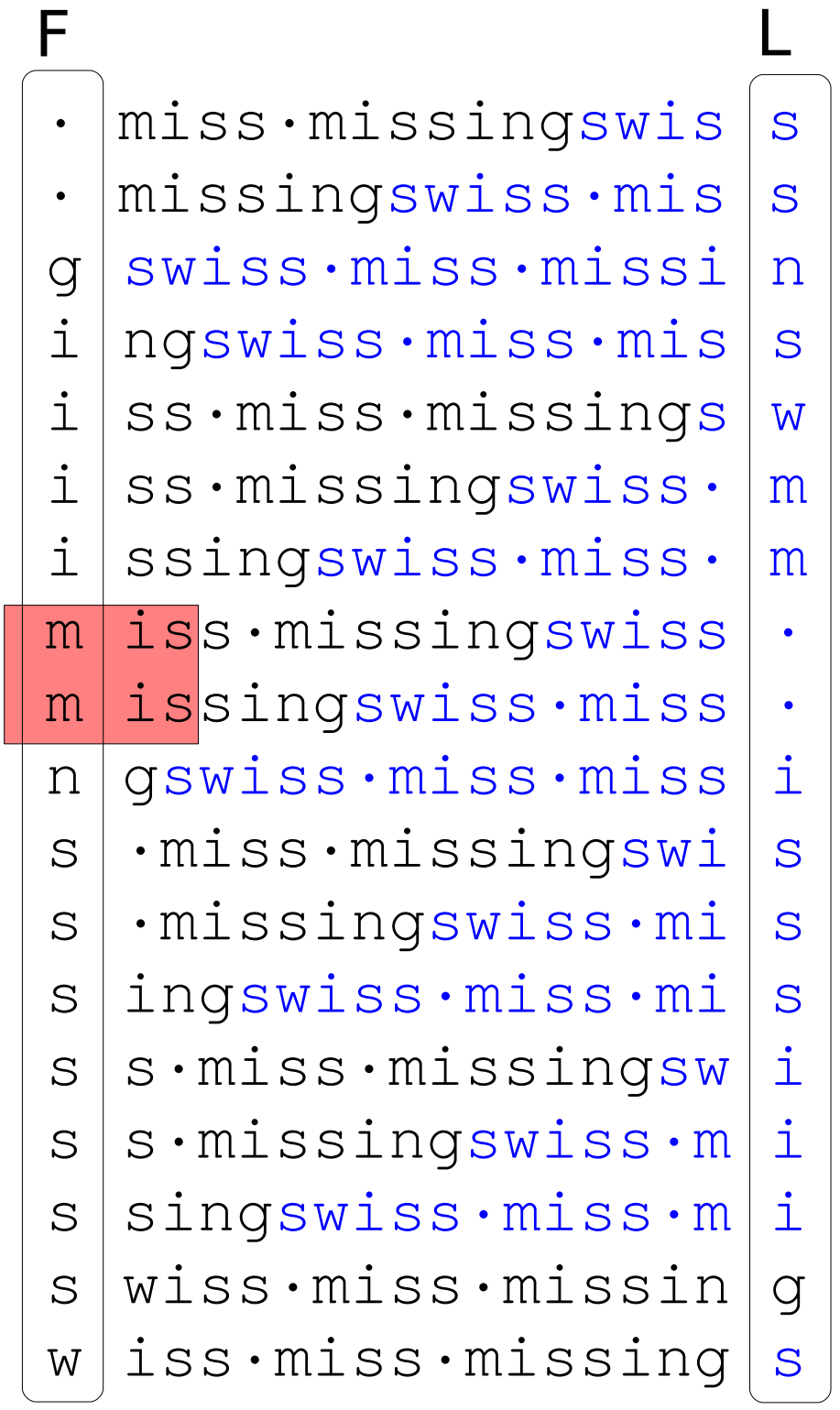
We have found the rows prefixed by **mis**!



Backward search

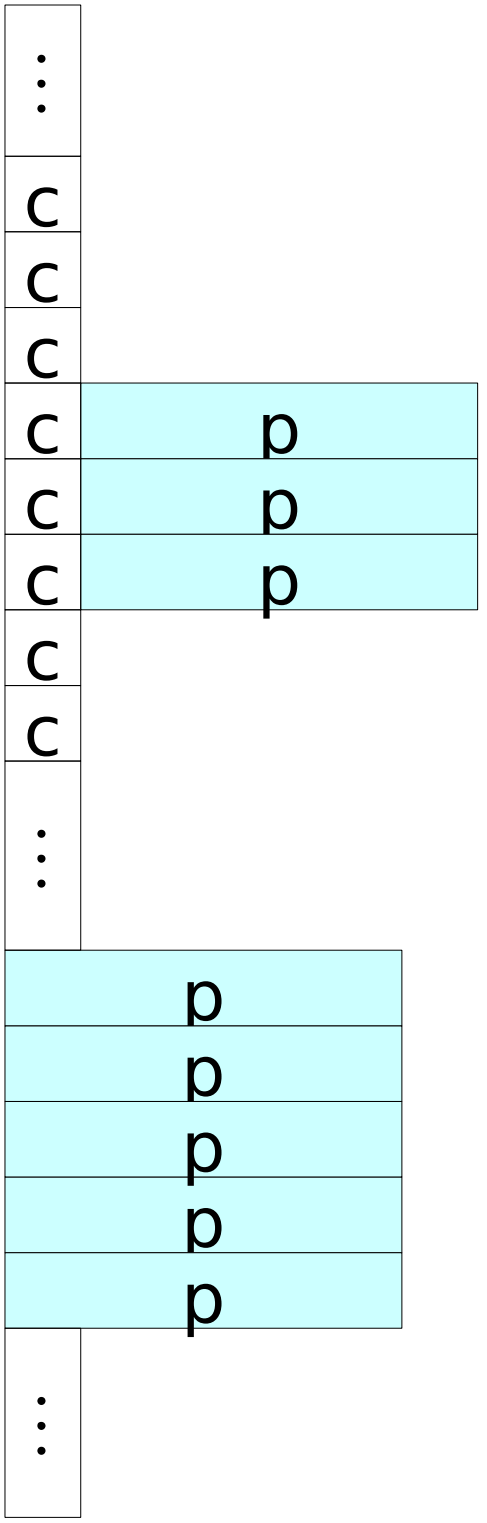
We have found the rows prefixed by **mis**!

Even if we only have F and L we can work as if we had the SA



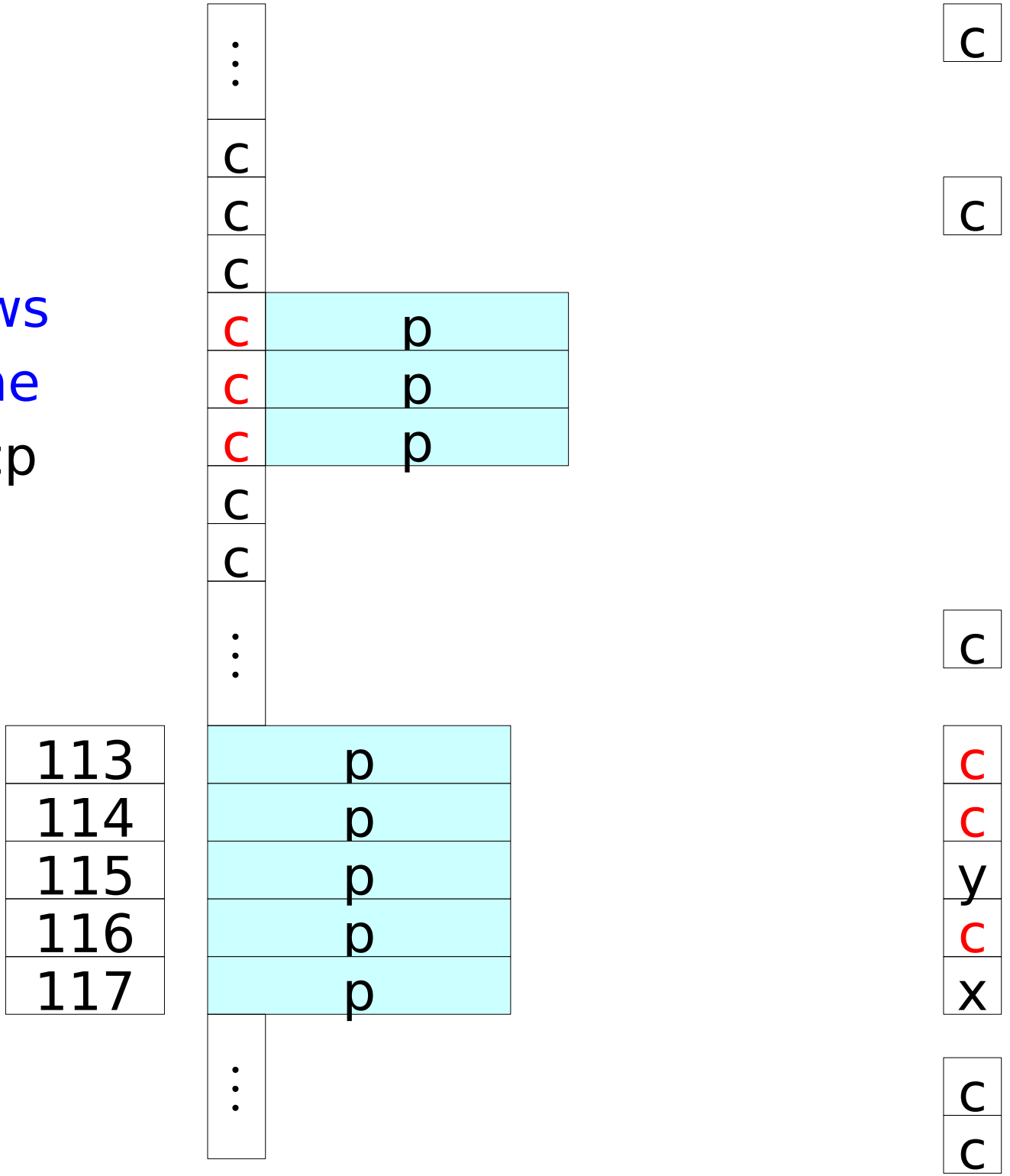
A closer look

The basic step is going from the rows prefixed by p to the rows prefixed by cp



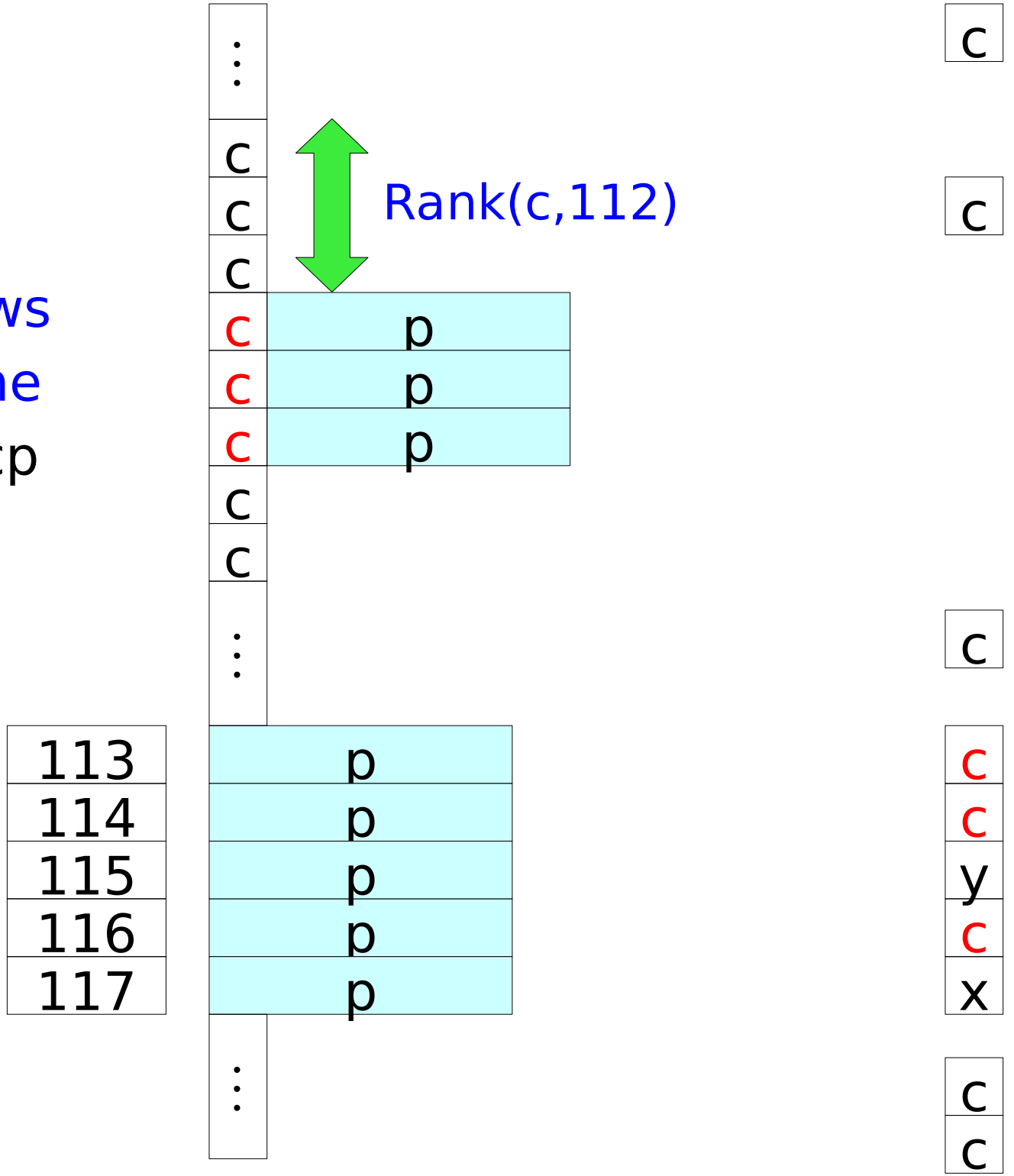
A closer look

The basic step is going from the rows prefixed by p to the rows prefixed by cp



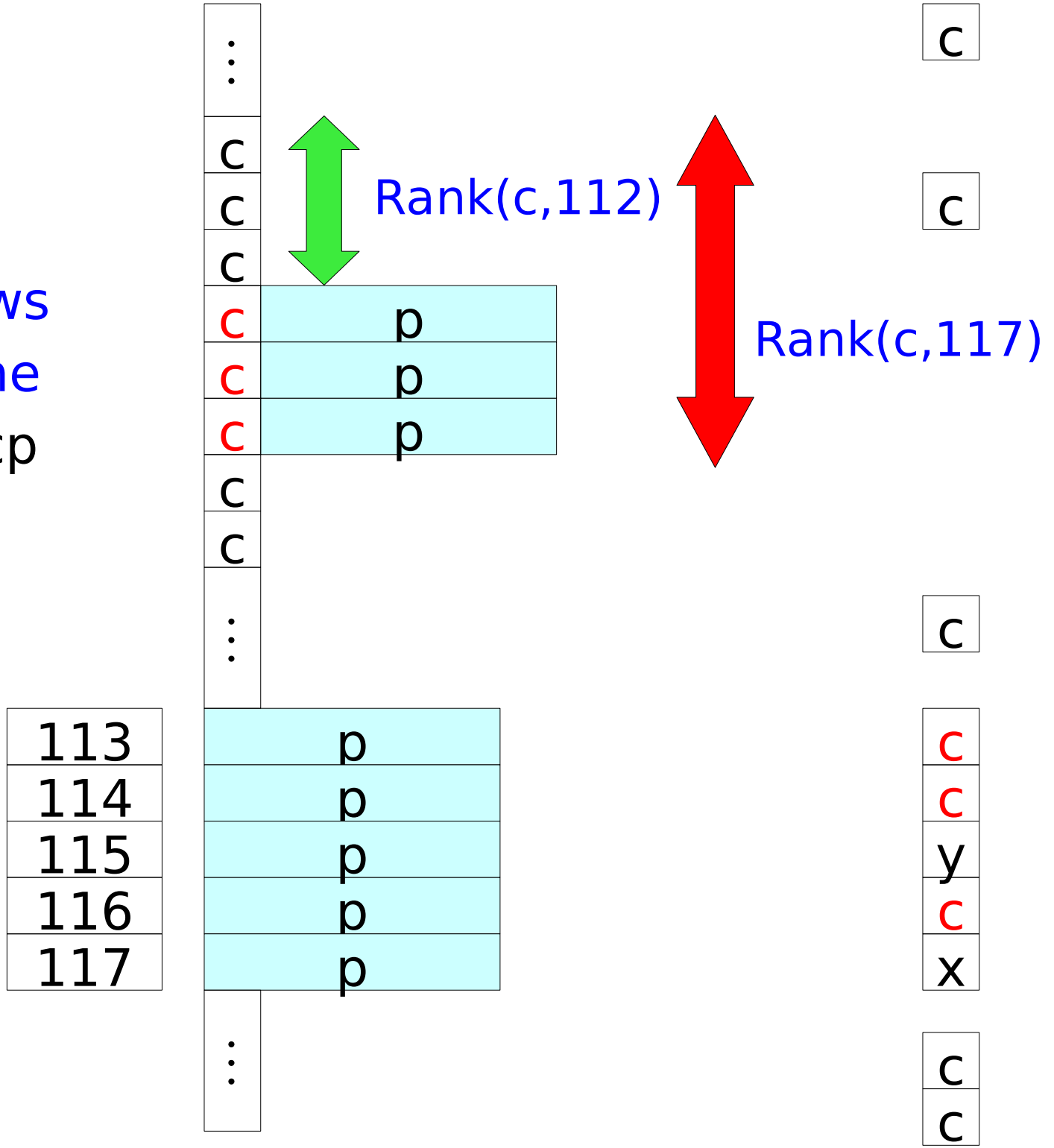
A closer look

The basic step is going from the rows prefixed by p to the rows prefixed by cp



A closer look

The basic step is going from the rows prefixed by p to the rows prefixed by cp



Summing up

Each basic step requires two rank queries on L

We can do a rank query in O(log|A|) time on a compressed sequence.

Finding the range of rows prefixed by a pattern P takes O(|P| log|A|) time (no dependency on |T|!).

We have a compressed representation of T supporting fast queries.