Suffix arrays

Suffix array

- We loose some of the functionality but we save space.

Let s = abab

Sort the suffixes lexicographically:
ab, abab, b, bab

The suffix array gives the indices of the suffixes in sorted order

2	0	3	1

How do we build it ?

- Build a suffix tree
- Traverse the tree in DFS, lexicographically picking edges outgoing from each node and fill the suffix array.

- O(n) time

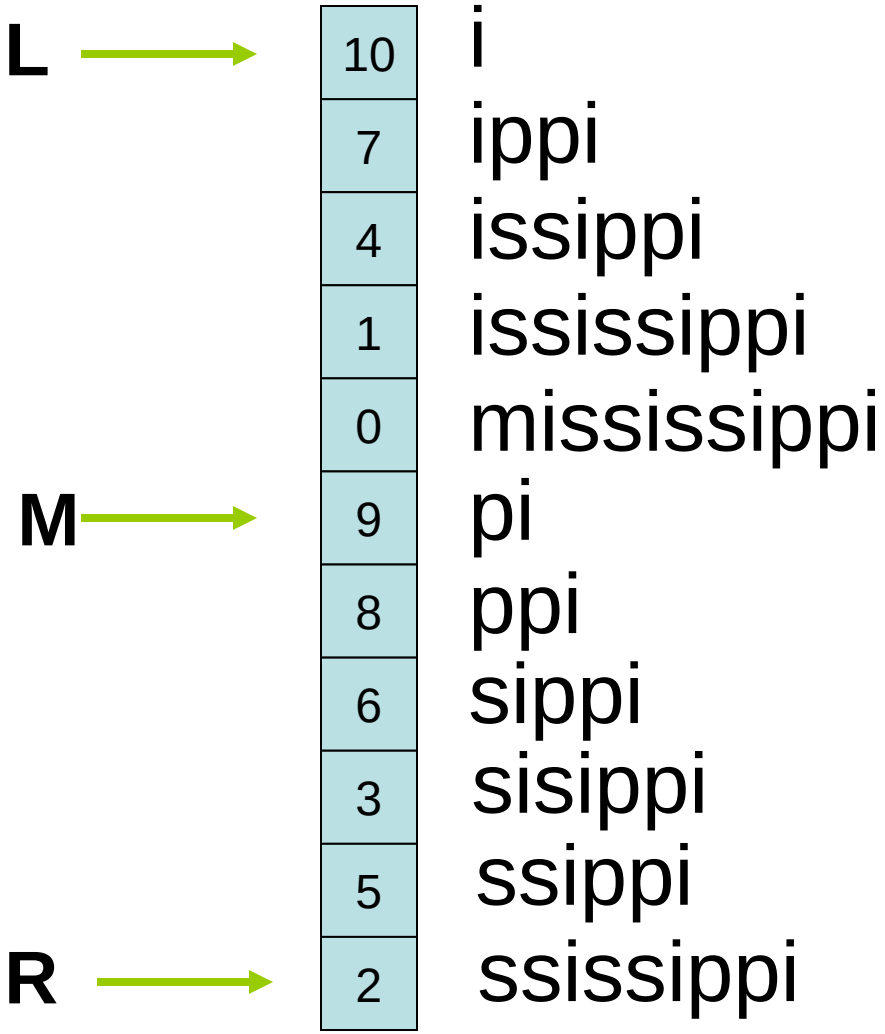
How do we search for a pattern ?

- If P occurs in T then all its occurrences are consecutive in the suffix array.
- Do a binary search on the suffix array
- Takes O(mlogn) time

Example

Let S = mississippi

Let P = issa

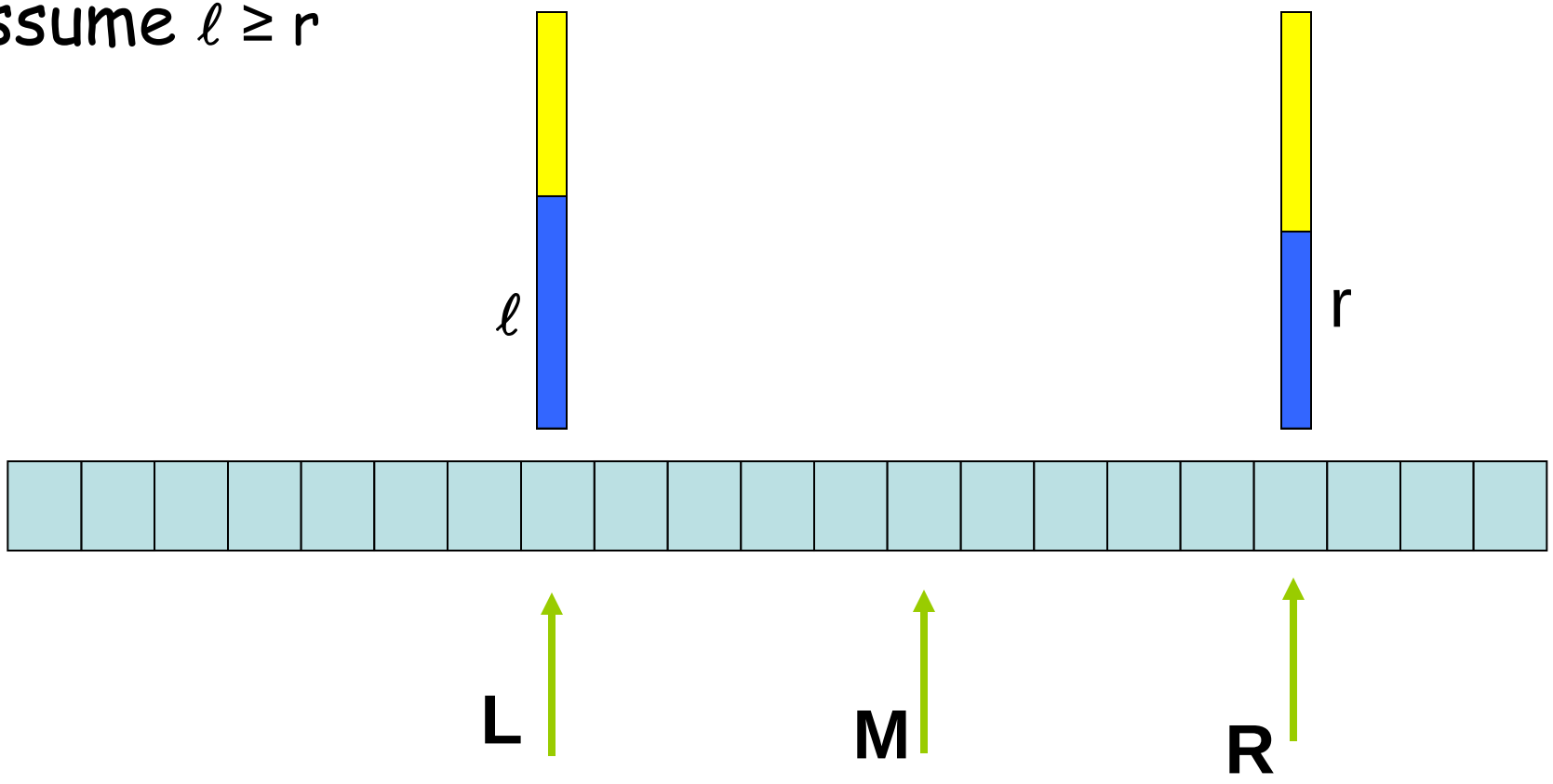


How do we accelerate the search ?

Maintain ℓ = LCP(P,L)

Maintain r = LCP(P,R)

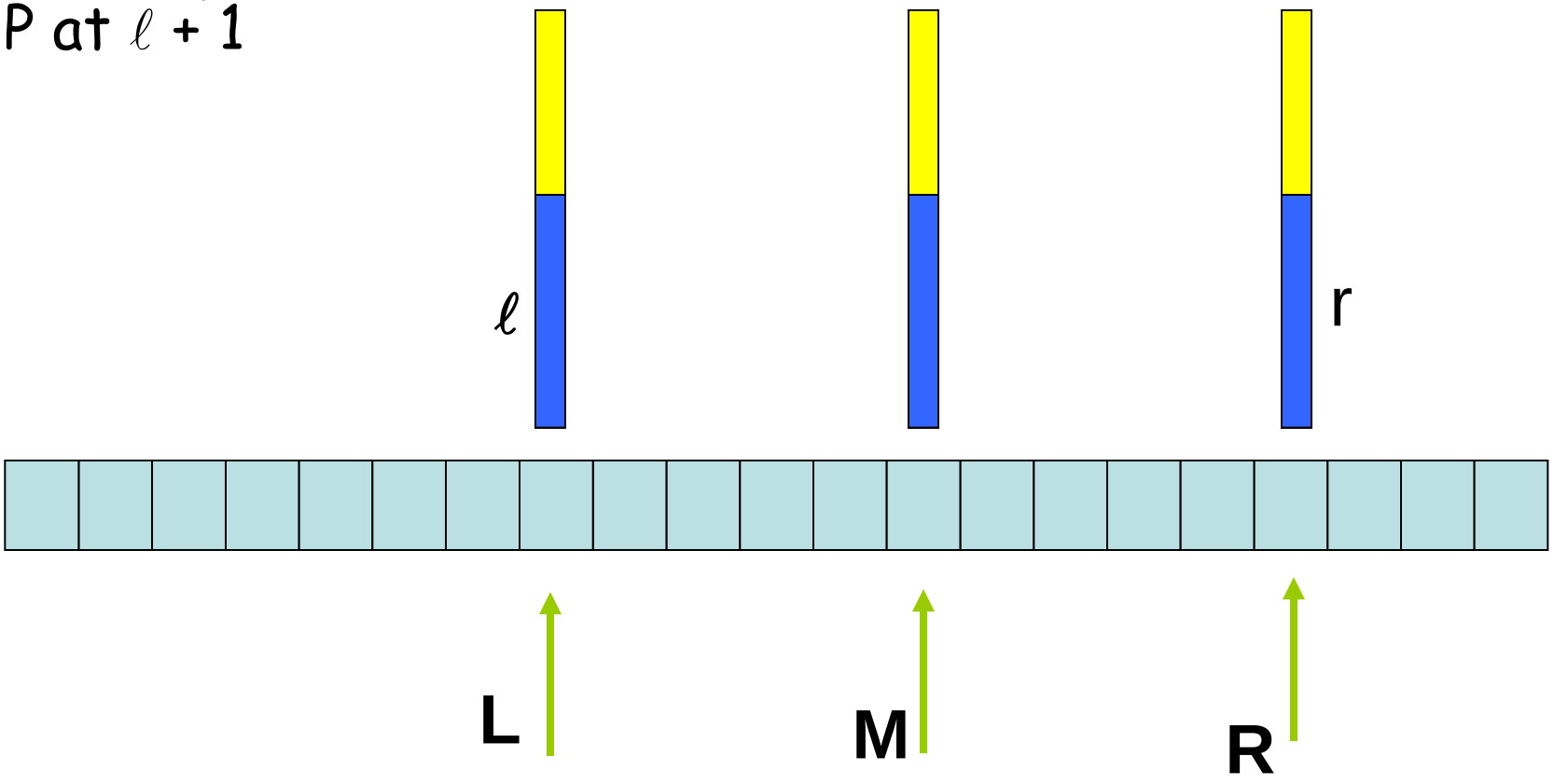
Assume $\ell \geq r$



Some intuition

If ℓ = r then

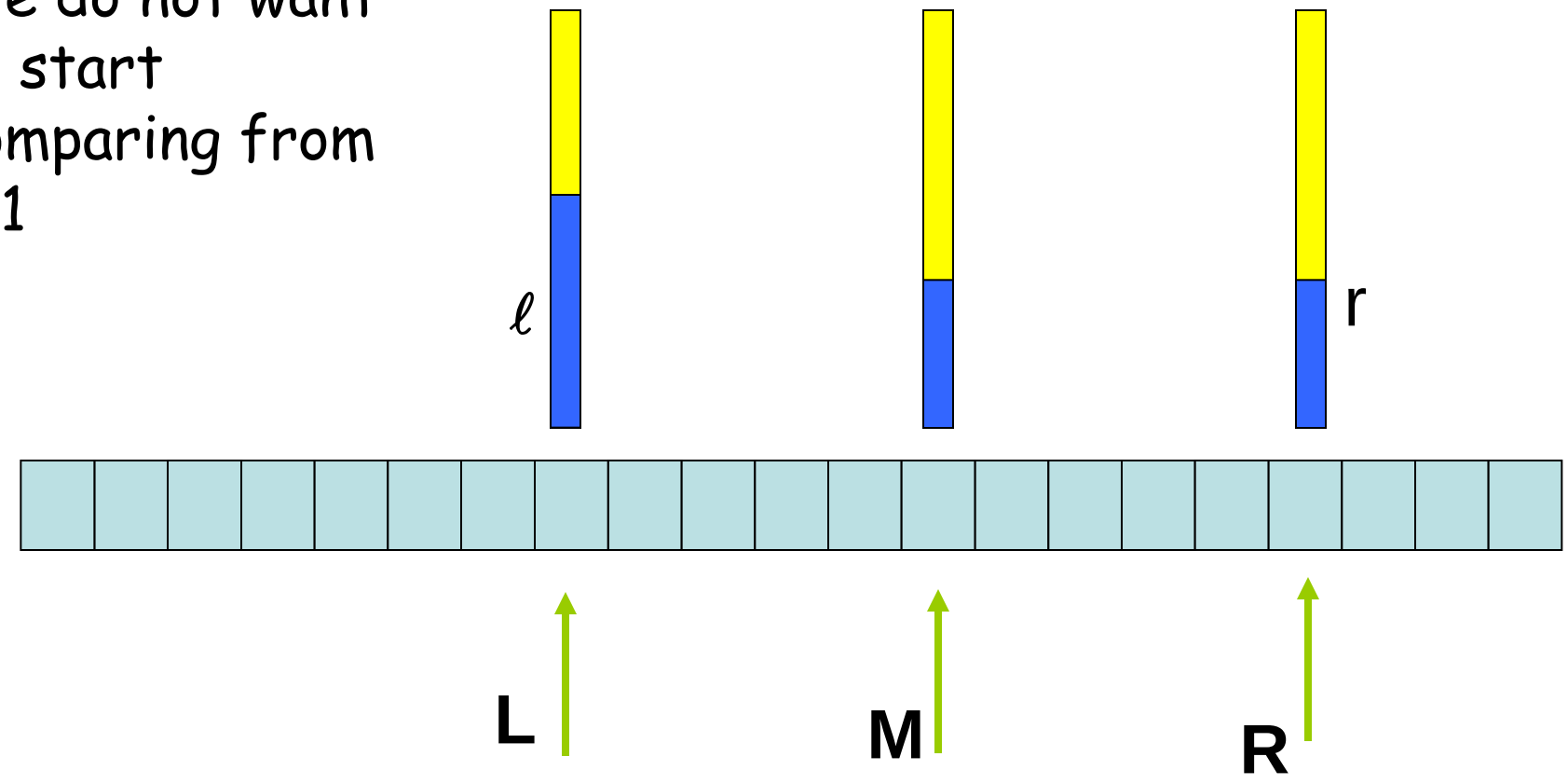
start comparing M to P at ℓ + 1



Some intuition

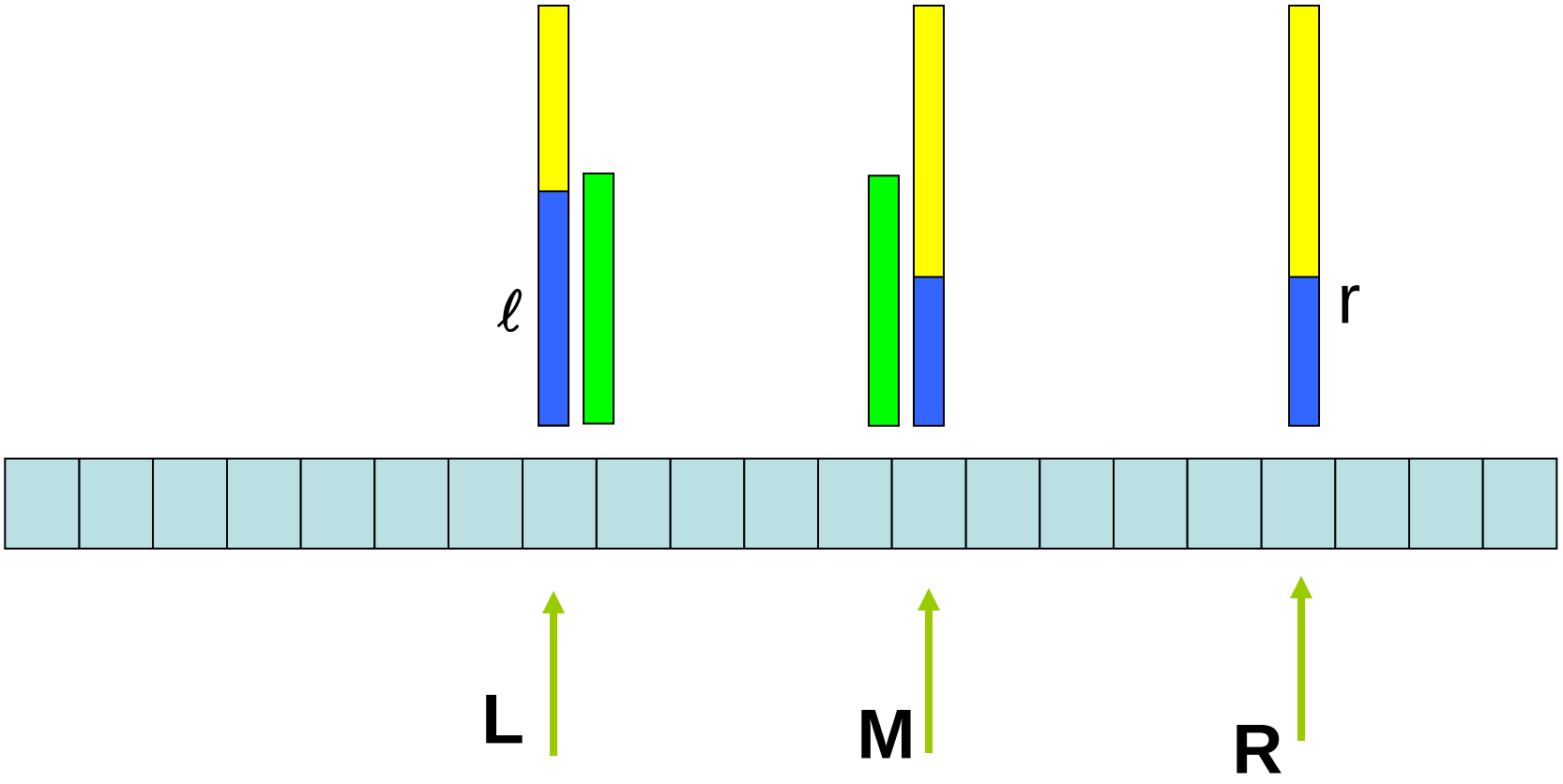
$\ell > r$

We do not want to start comparing from r+1



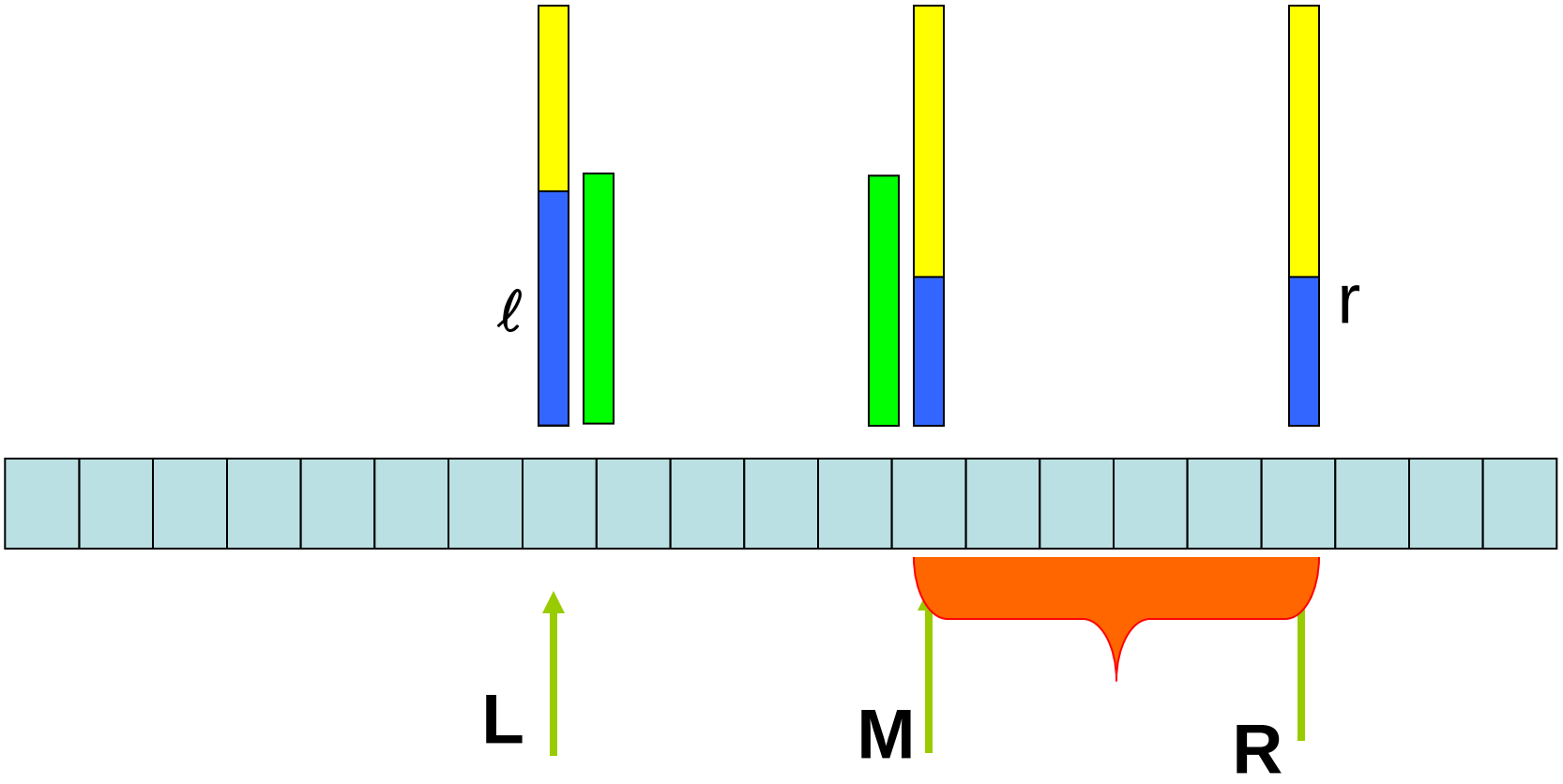
Someone whispers LCP(L,M)

LCP(L,M) > ℓ

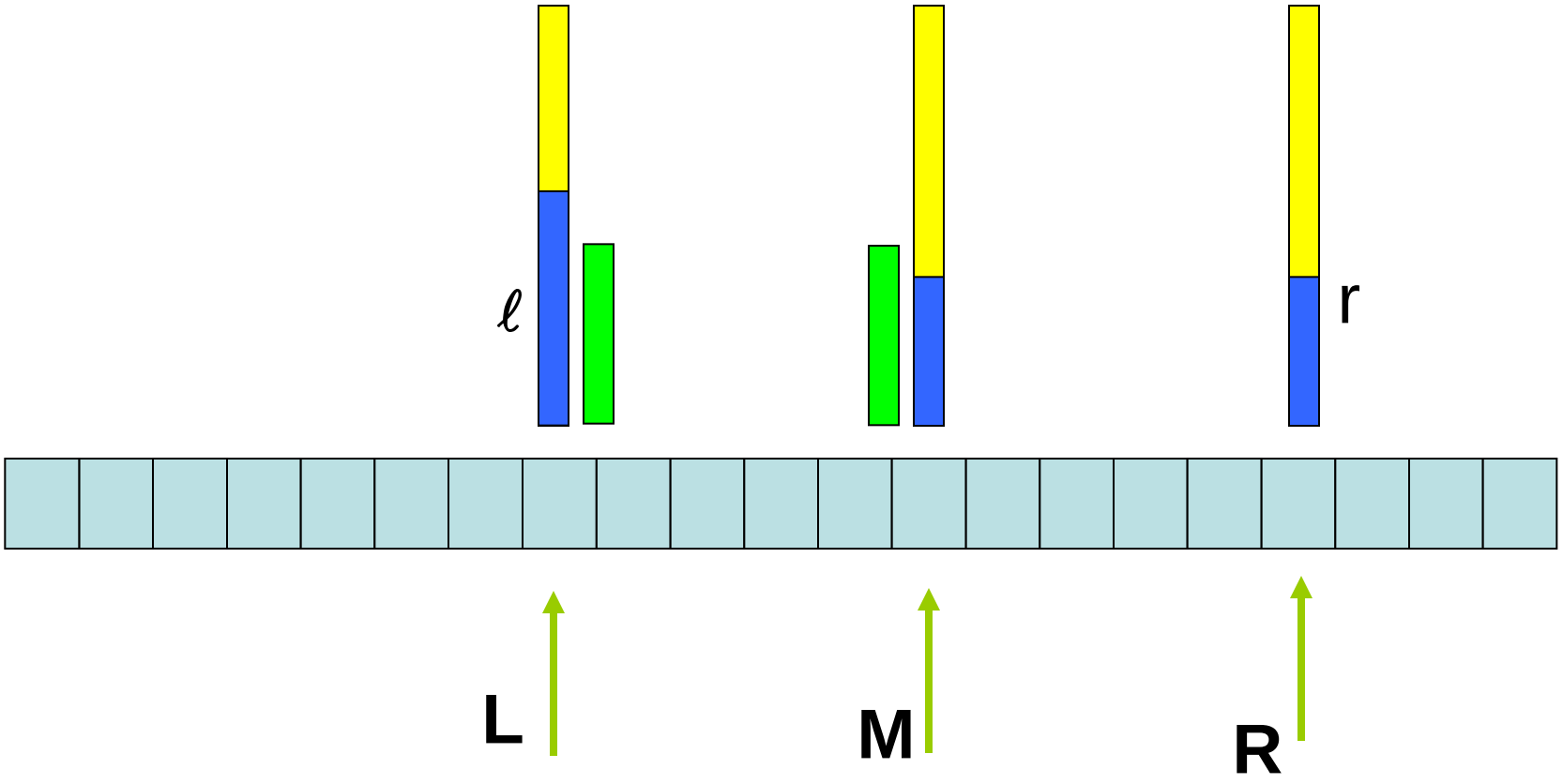


Continue in the right half

LCP(L,M) > ℓ

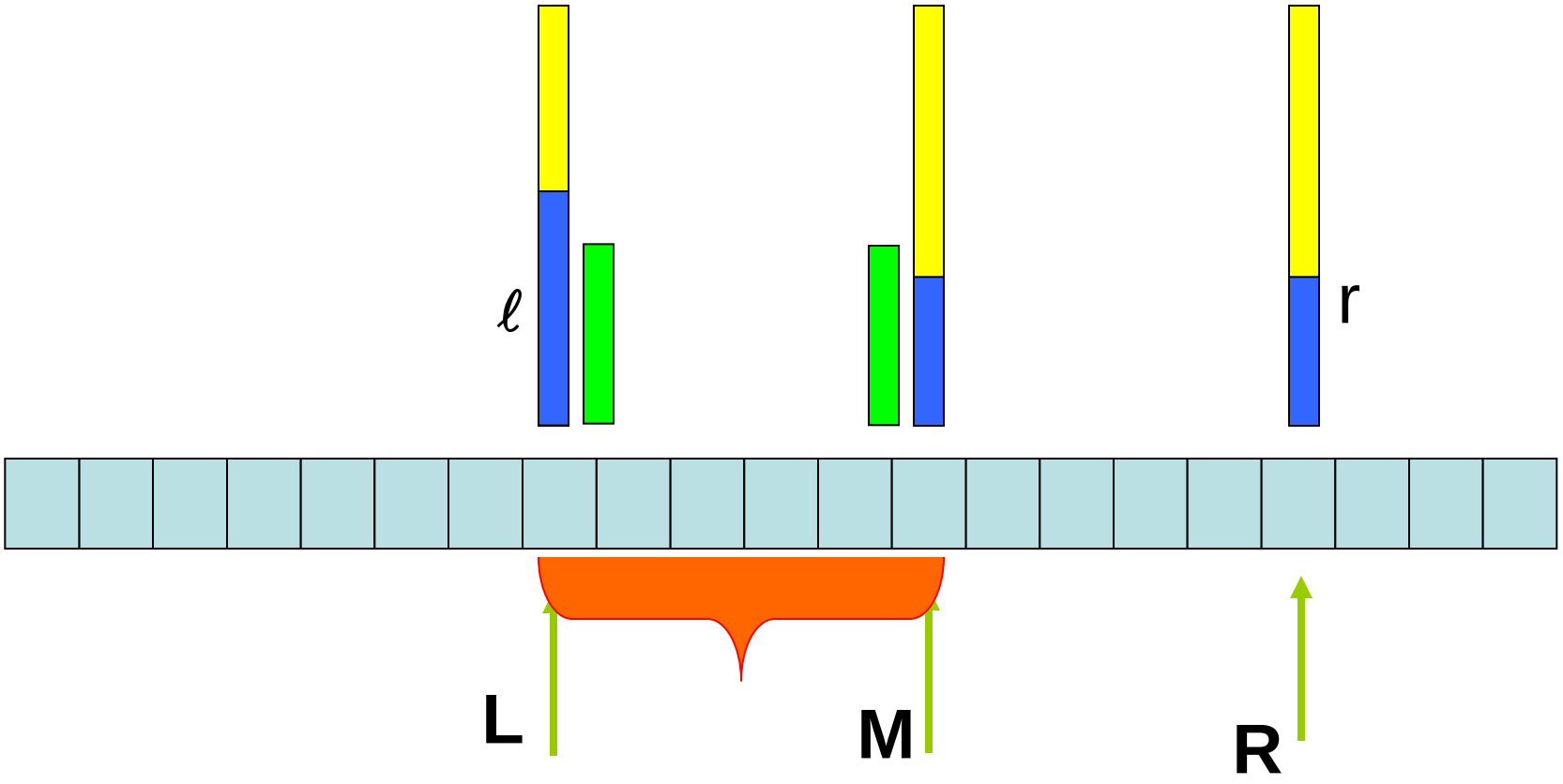


LCP(L,M) < ℓ



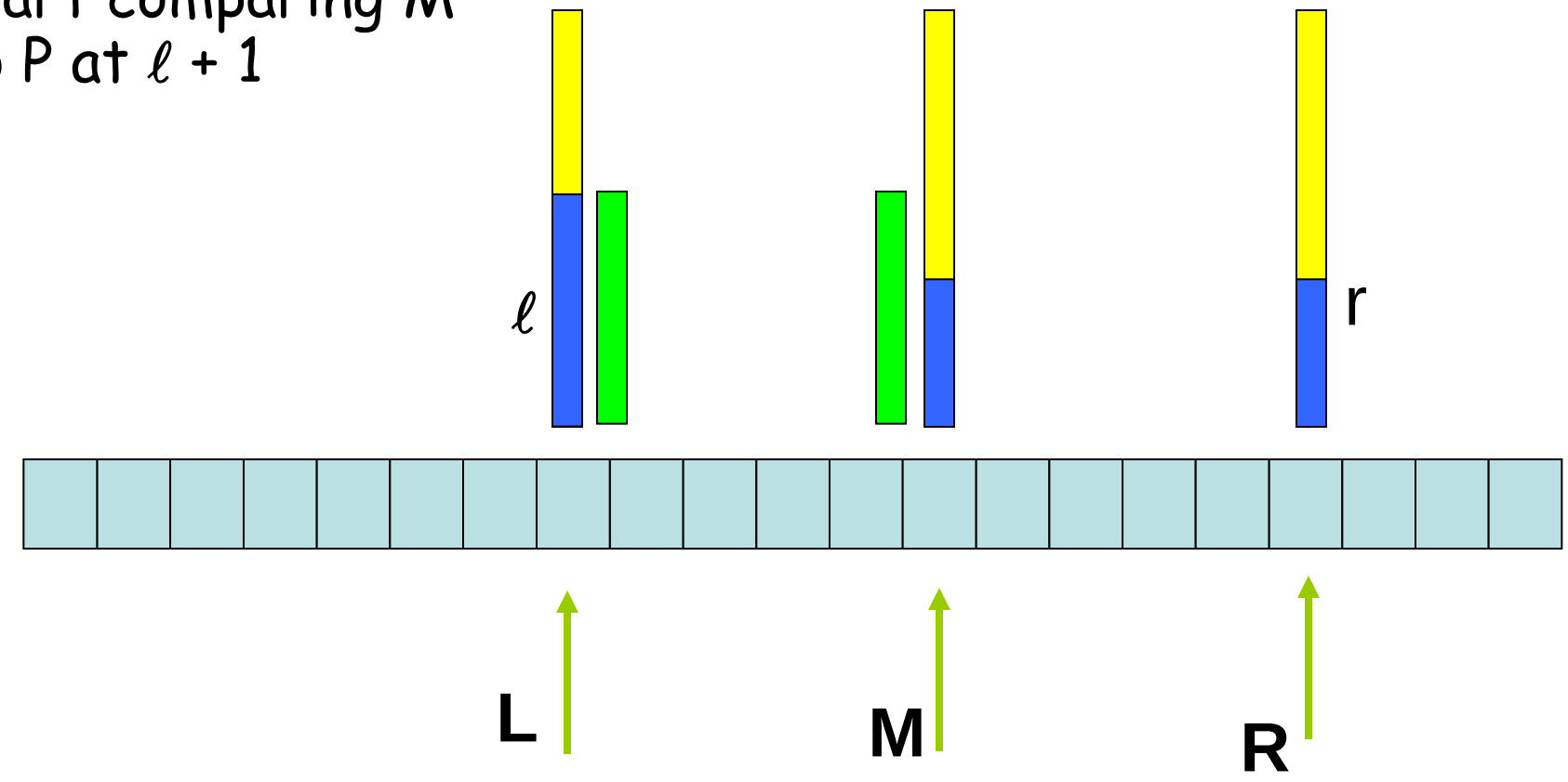
Continue in the left half

LCP(L,M) < ℓ



LCP(L,M) = ℓ

start comparing M
to P at ℓ + 1



Analysis

If we do more than a single comparison in an iteration then $\max(\ell, r)$ grows by 1 for each comparison ➔ $O(m + \log n)$ time

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

Traverse the suffixes from 1 to n, at step i compute LCP(pred(i),i) where pred(i) is the predecessor of i in the suffix array.

2 abbadabbado$
7 abbado$
5 adabbado$
10 ado$
4 badabbado$
9 bado$
3 bbadabbado$
8 bbado$
6 dabbado$
11 do$
12 o$
1 yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(12,1) ?

	2	abbadabbado$
	7	abbado$
	5	adabbado$
	10	ado$
	4	badabbado$
	9	bado$
	3	bbadabbado$
	8	bbado$
	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

Skip LCP(,2)

	2	abbadabbado$
	7	abbado$
	5	adabbado$
	10	ado$
	4	badabbado$
	9	bado$
	3	bbadabbado$
	8	bbado$
	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(9,3) ?

	2	abbadabbado$
	7	abbado$
	5	adabbado$
	10	ado$
	4	badabbado$
	9	bado$
1	3	bbadabbado$
	8	bbado$
	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(10,4) ?

	2	abbadabbado$
	7	abbado$
	5	adabbado$
	10	ado$
0	4	badabbado$
	9	bado$
1	3	bbadabbado$
	8	bbado$
	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(7,5) ?

	2	abbadabbado$
	7	abbado$
1	5	adabbado$
	10	ado$
0	4	badabbado$
	9	bado$
1	3	bbadabbado$
	8	bbado$
	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(8,6) ?

	2	abbadabbado$
	7	abbado$
1	5	adabbado$
	10	ado$
0	4	badabbado$
	9	bado$
1	3	bbadabbado$
	8	bbado$
0	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(2,7) ?

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
	10	ado$
0	4	badabbado$
	9	bado$
1	3	bbadabbado$
	8	bbado$
0	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

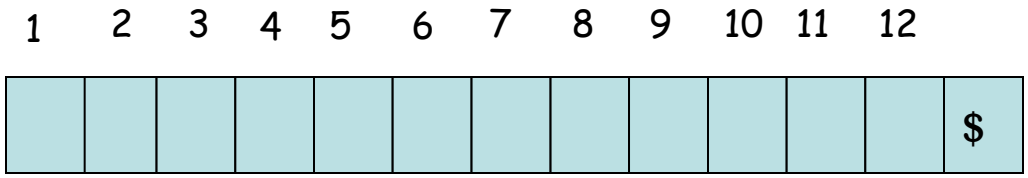
1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

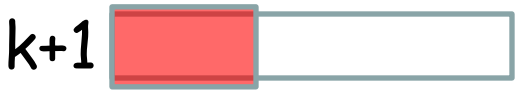
LCP(3,8) ?

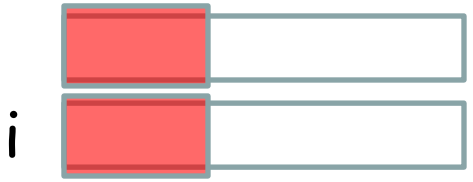
We know that this is ≥ 4

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
	10	ado$
0	4	badabbado$
	9	bado$
1	3	bbadabbado$
	8	bbado$
0	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

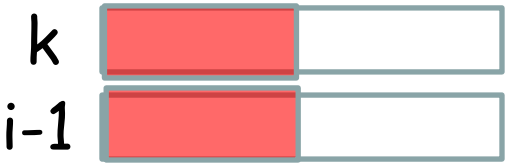
Compute LCP's of adjacent suffixes







Where is suffix k+1 ?



Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(3,8) ?

We know that this is ≥ 4

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
	10	ado$
0	4	badabbado$
	9	bado$
1	3	bbadabbado$
4	8	bbado$
0	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(4,9) ?

We know that this is ≥ 3

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
	10	ado$
0	4	badabbado$
3	9	bado$
1	3	bbadabbado$
4	8	bbado$
0	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(5,10) ?

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
2	10	ado$
0	4	badabbado$
3	9	bado$
1	3	bbadabbado$
4	8	bbado$
0	6	dabbado$
	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(6,11) ?

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
2	10	ado$
0	4	badabbado$
3	9	bado$
1	3	bbadabbado$
4	8	bbado$
0	6	dabbado$
1	11	do$
	12	o$
0	1	yabbadabbado$

Compute LCP's of adjacent suffixes

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

LCP(11,12) ?

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
2	10	ado$
0	4	badabbado$
3	9	bado$
1	3	bbadabbado$
4	8	bbado$
0	6	dabbado$
1	11	do$
0	12	o$
0	1	yabbadabbado$

Summary

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

The length of the current LCP decreases by one at the beginning of an iteration and then increases by comparisons

	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
2	10	ado$
0	4	badabbado$
3	9	bado$
1	3	bbadabbado$
4	8	bbado$
0	6	dabbado$
1	11	do$
0	12	o$
0	1	yabbadabbado$

Another example

1	2	3	4	5	6	7	8	9
a	b	c	a	b	b	c	a	$

	4	abbca$
2	1	abcabbca$
1	8	a$
0	5	bbca$
1	2	bcabbca$
3	6	bca$
0	3	cabbca$
2	7	ca$

Analysis

1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

The length of the current LCP decreases by one at the beginning of an iteration and then increases by comparisons. Its value is at most n

➔ It can increase 2n times

➔ O(n) time algorithm

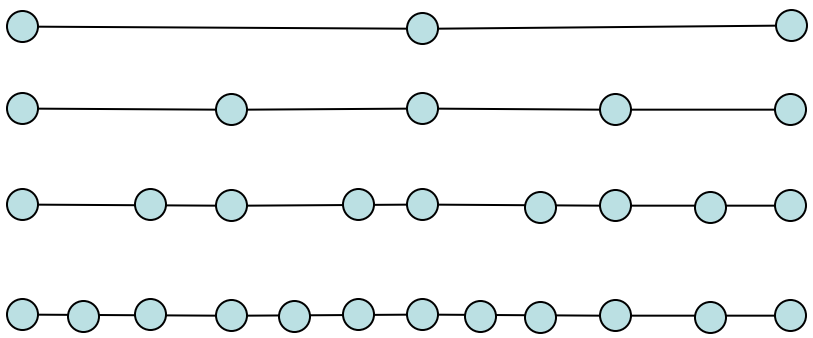
	2	abbadabbado$
5	7	abbado$
1	5	adabbado$
2	10	ado$
0	4	badabbado$
3	9	bado$
1	3	bbadabbado$
4	8	bbado$
0	6	dabbado$
1	11	do$
0	12	o$
0	1	yabbadabbado$

We need more LCPs for the search

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 6 4 9 3 8 2 7 5 10 11 0

5 1 2 0 3 1 4 0 1 0 0



Linearly many, calculate the all bottom up

Construct the suffix array without the suffix tree

Divide into triples

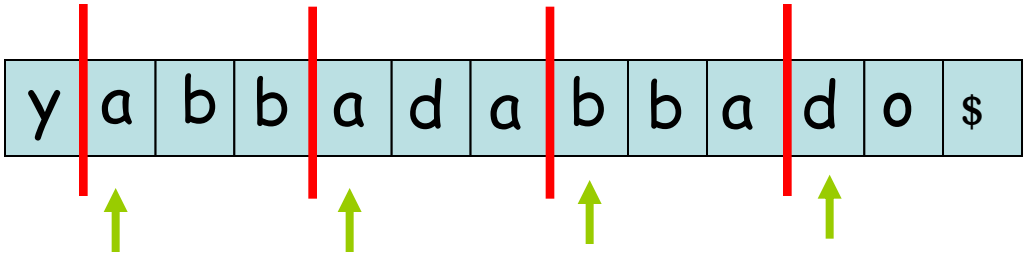


abb	ada	bba	do$

Divide into triples

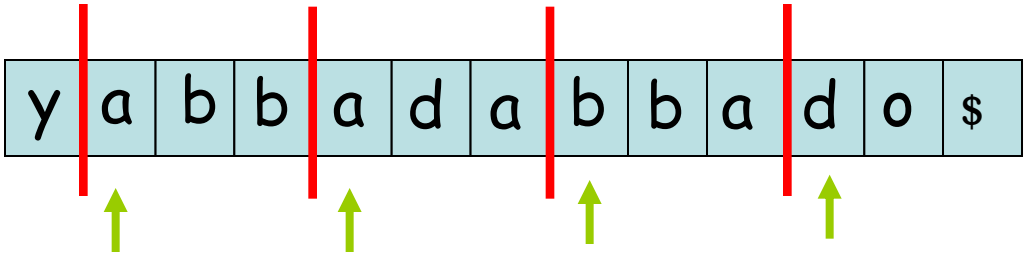
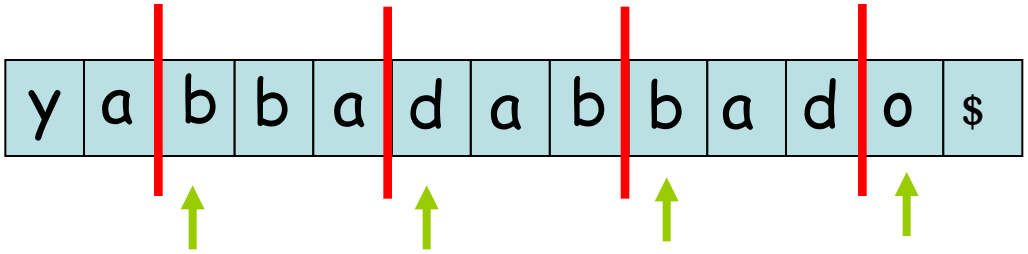
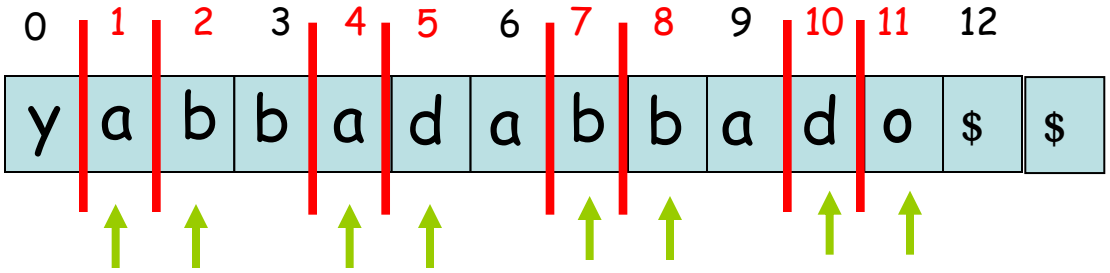


abb	ada	bba	do$

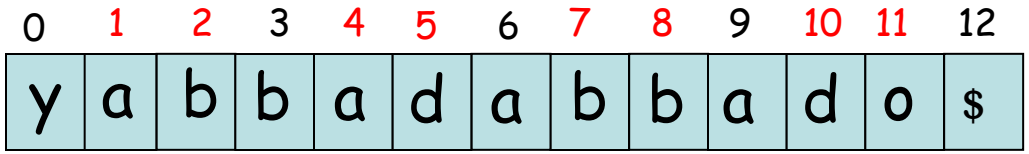


bba	dab	bad	o$$

Sort recursively 2/3 of the suffixes



	0/1	1/4	2/7	3/10	4/2	5/5	6/8	7/11
s'	abb	ada	bba	do$	bba	dab	bad	o$$
	1	2	4	6	4	5	3	7
SA(s')	0/1	1/4	6/8	4/2	2/7	5/5	3/10	7/11



ranks	1	4		2	6		5	3		7	8	

Sort the remaining third

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

(y, 1) (b, 2) (a, 5) (a, 7)

➔

(a, 5) (a, 7) (b, 2) (y, 1)

6 9 3 0

1 4 8 2 7 5 10 11

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

6 9 3 0

1 4 8 2 7 5 10 11

1

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

6 9 3 0

4 8 2 7 5 10 11

1 6

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

9 3 0

4 8 2 7 5 10 11

1 6 4

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

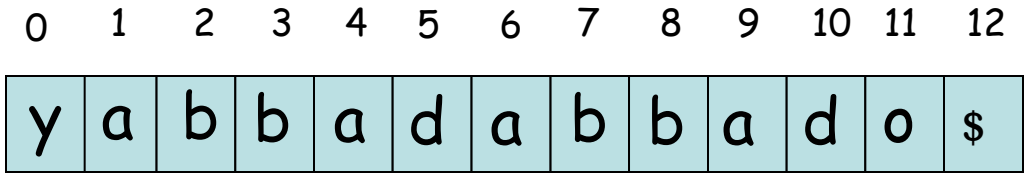
1 4 2 6 5 3 7 8

9 3 0

8 2 7 5 10 11

1 6 4 9

Merge



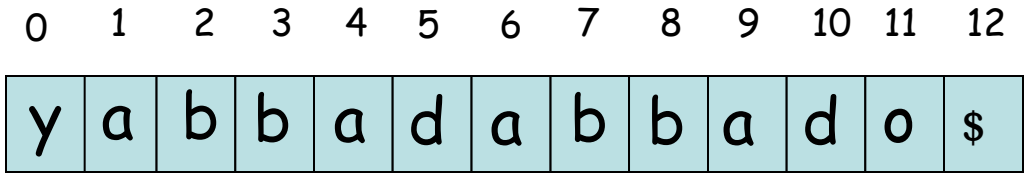
1 4 2 6 5 3 7 8

3 0

8 2 7 5 10 11

1 6 4 9 3

Merge



1 4 2 6 5 3 7 8

0

8 2 7 5 10 11

1 6 4 9 3 8

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

0

2 7 5 10 11

1 6 4 9 3 8 2

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

0

7 5 10 11

1 6 4 9 3 8 2 7

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

0

5 10 11

1 6 4 9 3 8 2 7 5

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$

1 4 2 6 5 3 7 8

0

10 11

1 6 4 9 3 8 2 7 5

Merge

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$
	1	4		2	6		5	3		7	8	

1 6 4 9 3 8 2 7 5 10 11 0

summary

0	1	2	3	4	5	6	7	8	9	10	11	12
y	a	b	b	a	d	a	b	b	a	d	o	$
	1	4		2	6		5	3		7	8	

1 6 4 9 3 8 2 7 5 10 11 0

When comparing to a suffix with index 1 (mod 3) we compare the char and break ties by the ranks of the following suffixes

When comparing to a suffix with index 2 (mod 3) we compare the char, the next char if there is a tie, and finally the ranks of the following suffixes

Burrows –Wheeler (bzip2)

Currently best algorithm for text

High level:

- Apply the Borrows-Wheeler transform
- Use move-to-front to translate the sorted characters to small integers
- Use Huffman coding

שלב I : יצירת מטריצה M שבנויה מכל ההזזות הציקליות של S:

$$S = abraca$$

$$M = \begin{array}{|c|c|c|c|c|c|c|}\hline a & b & r & a & c & a & \# \\ \hline b & r & a & c & a & \# & a \\ \hline r & a & c & a & \# & a & b \\ \hline a & c & a & \# & a & b & r \\ \hline c & a & \# & a & b & r & a \\ \hline a & \# & a & b & r & a & c \\ \hline \# & a & b & r & a & c & a \\ \hline \end{array}$$

שלב II : מיון השורות בסדר לקסיקוגרפי:

L is the Burrows Wheeler Transform

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

Claim: Every column contains all chars.

a	b	r	a	c	a	#
b	r	a	c	a	#	a
r	a	c	a	#	a	b
a	c	a	#	a	b	r
c	a	#	a	b	r	a
a	#	a	b	r	a	c
#	a	b	r	a	c	a

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

You can obtain F from L by sorting

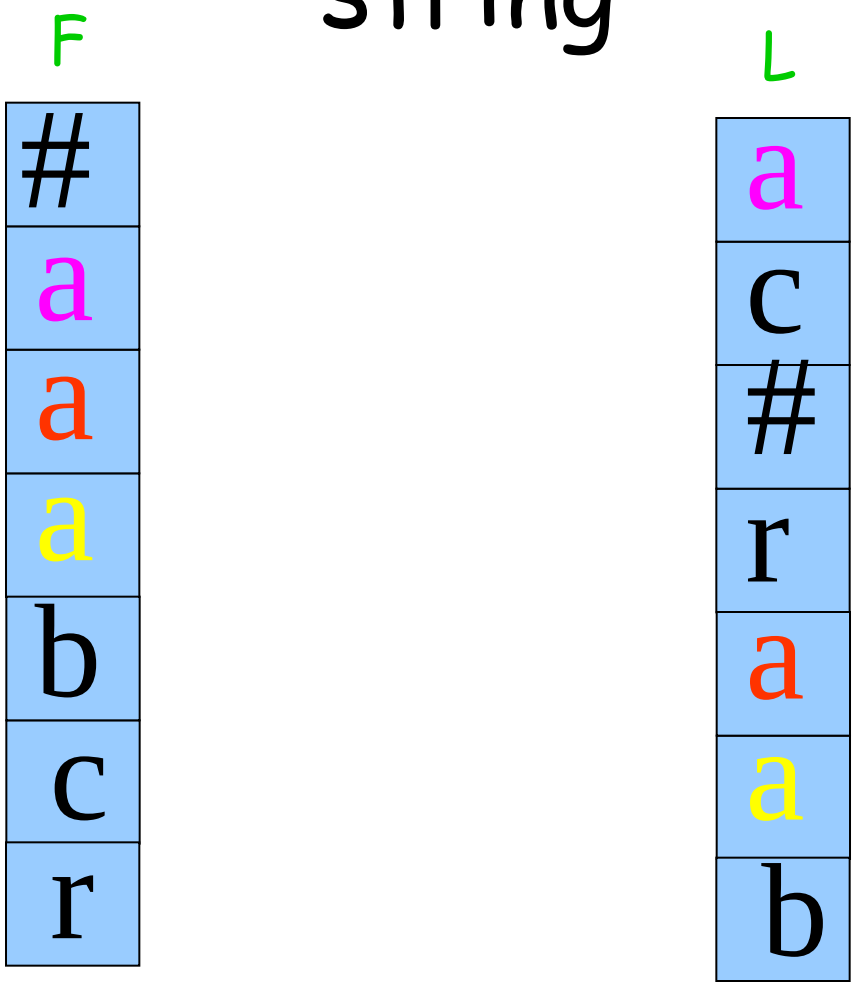
F L

#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

The "a's" are in the same order in L and in F,

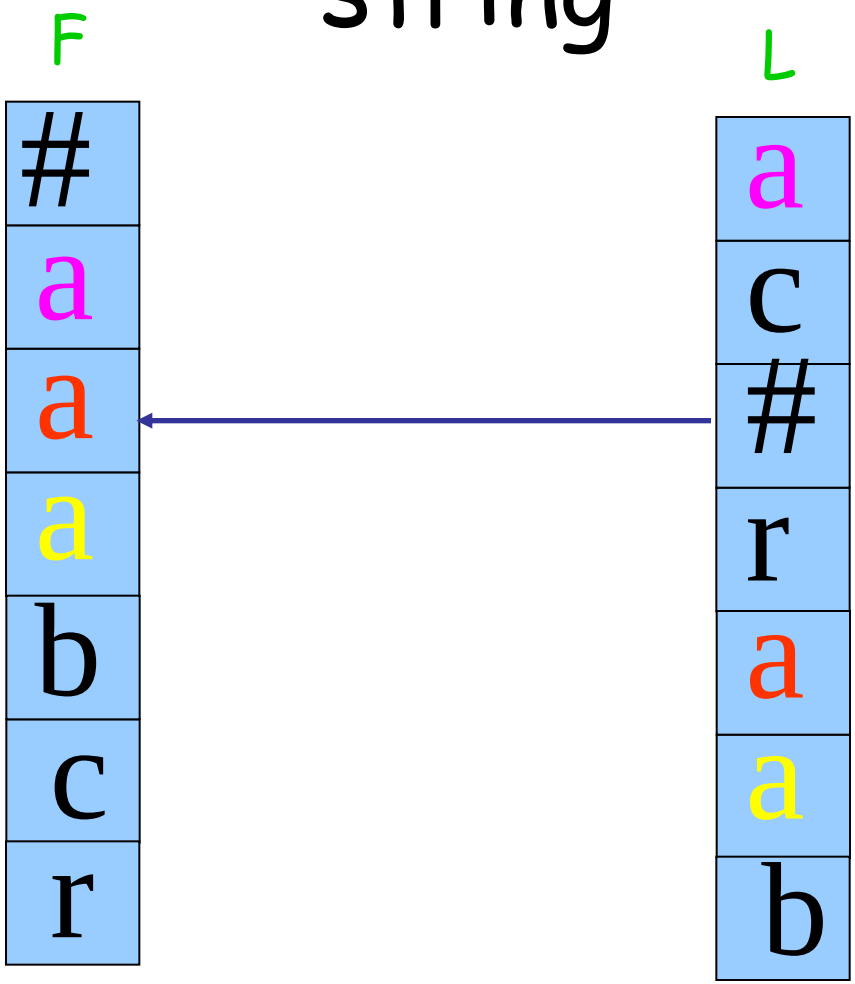
Similarly for every other char.

From L you can reconstruct the string



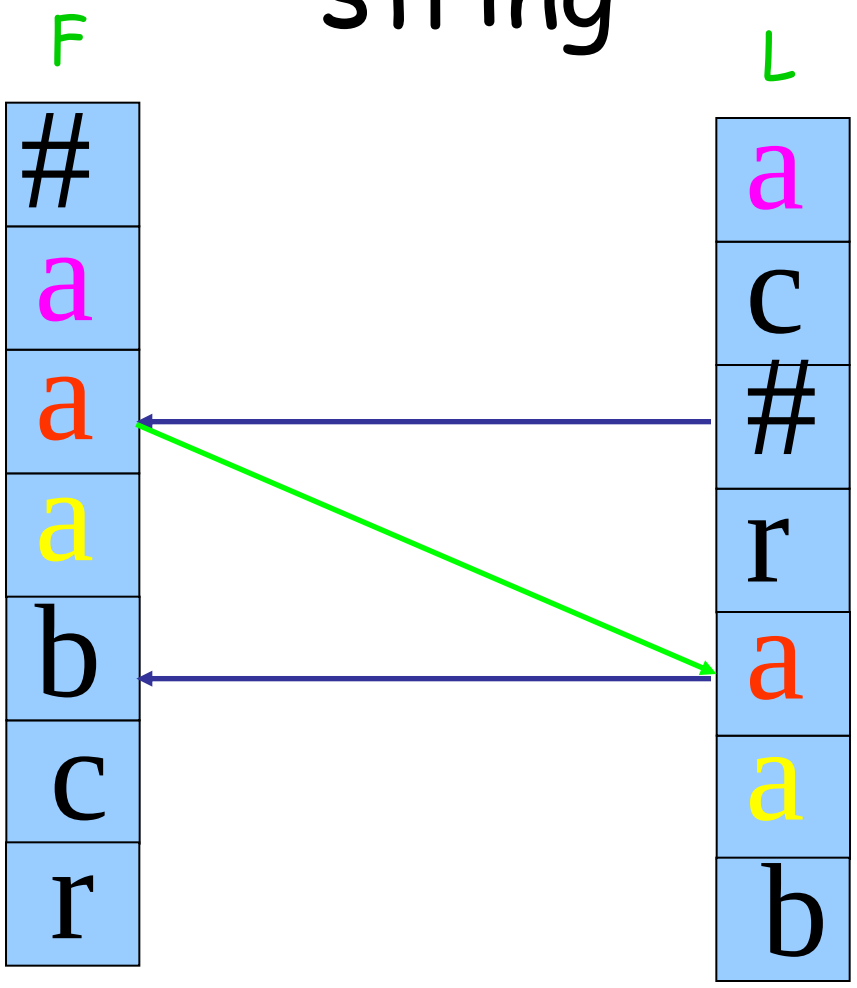
What is the first char of S ?

From L you can reconstruct the string



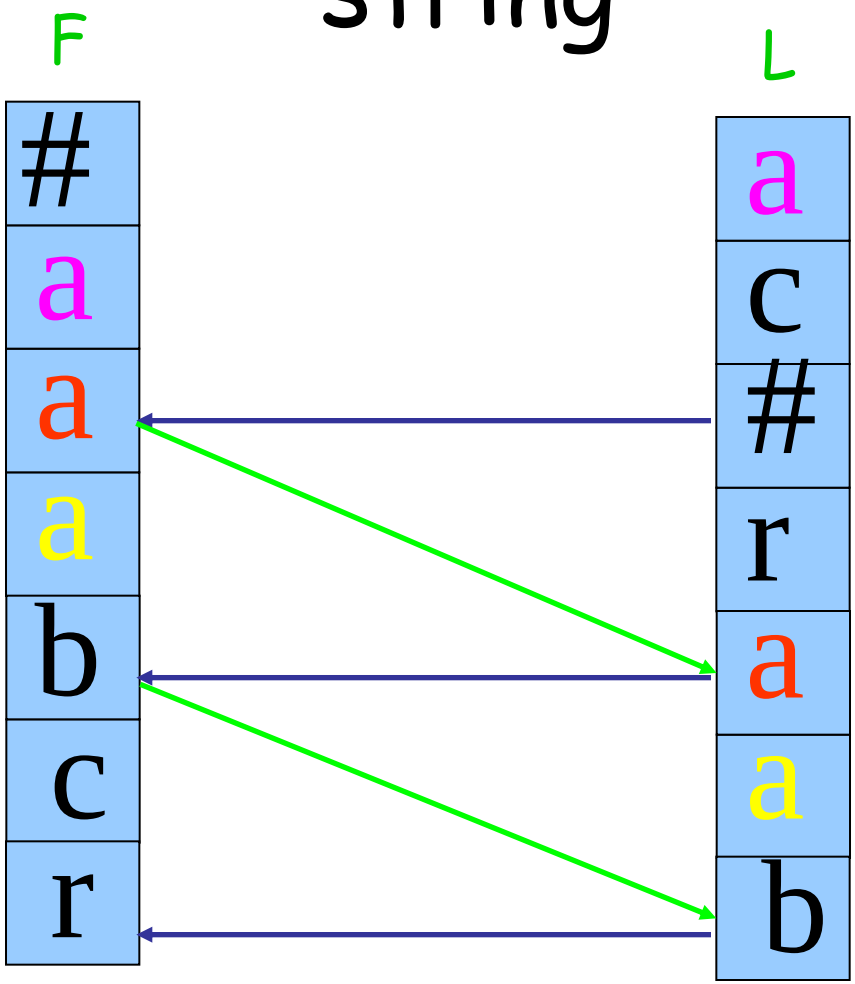
What is the first char of S ? a

From L you can reconstruct the string



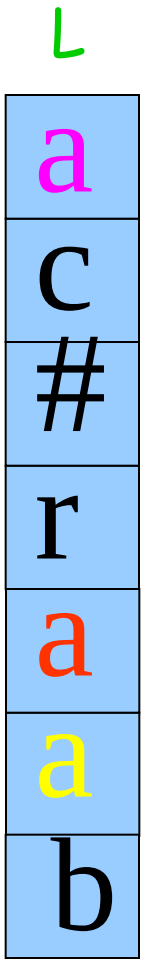
ab

From L you can reconstruct the string



abr

Compression ?



Compress the transform to a string of integers using move to front

0 2 3 2 0 3

Then use Huffman to code the integers

Why is it good ?

a	b	r	a	c	a	#
b	r	a	c	a	#	a
r	a	c	a	#	a	b
a	c	a	#	a	b	r
c	a	#	a	b	r	a
a	#	a	b	r	a	c
#	a	b	r	a	c	a

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

Characters with the same (right) context appear together

Sorting is equivalent to computing the suffix array.

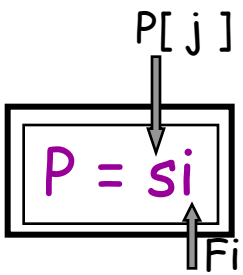
a	b	r	a	c	a	#
b	r	a	c	a	#	a
r	a	c	a	#	a	b
a	c	a	#	a	b	r
c	a	#	a	b	r	a
a	#	a	b	r	a	c
#	a	b	r	a	c	a

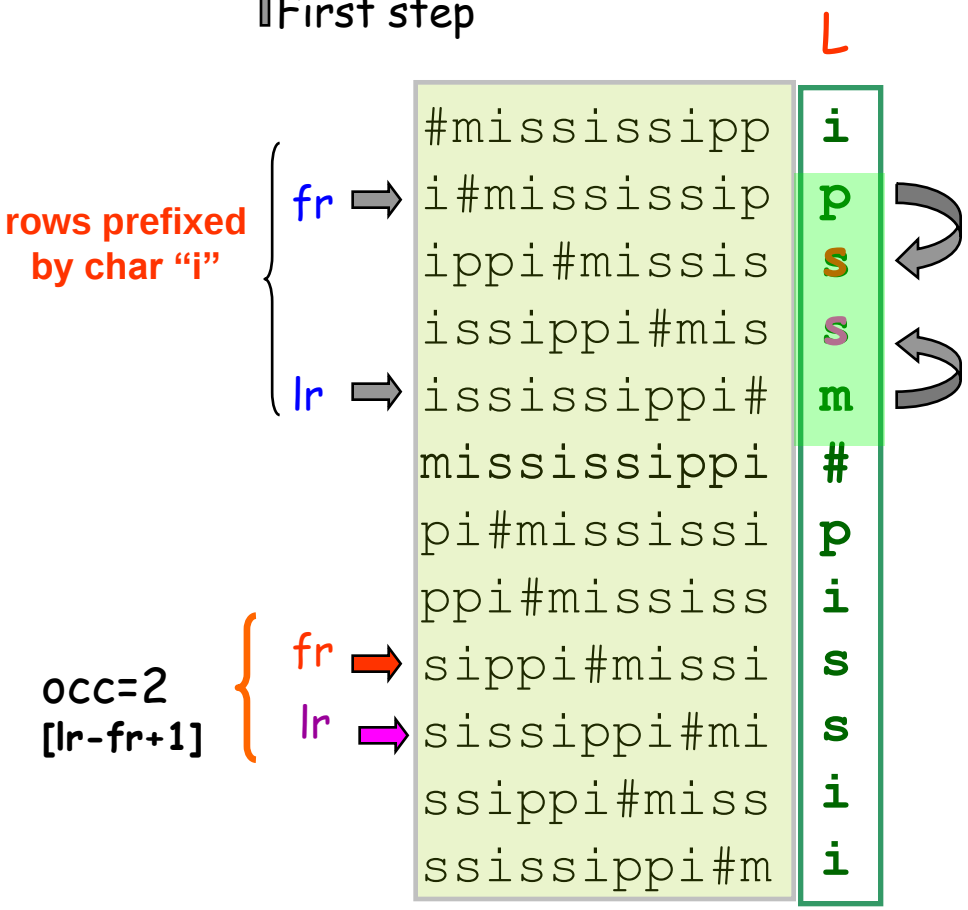
F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

Can encode and decode in linear time

Substring search using the BWT

(Count the pattern occurrences)





C

#	1
i	2
m	6
p	7
s	9

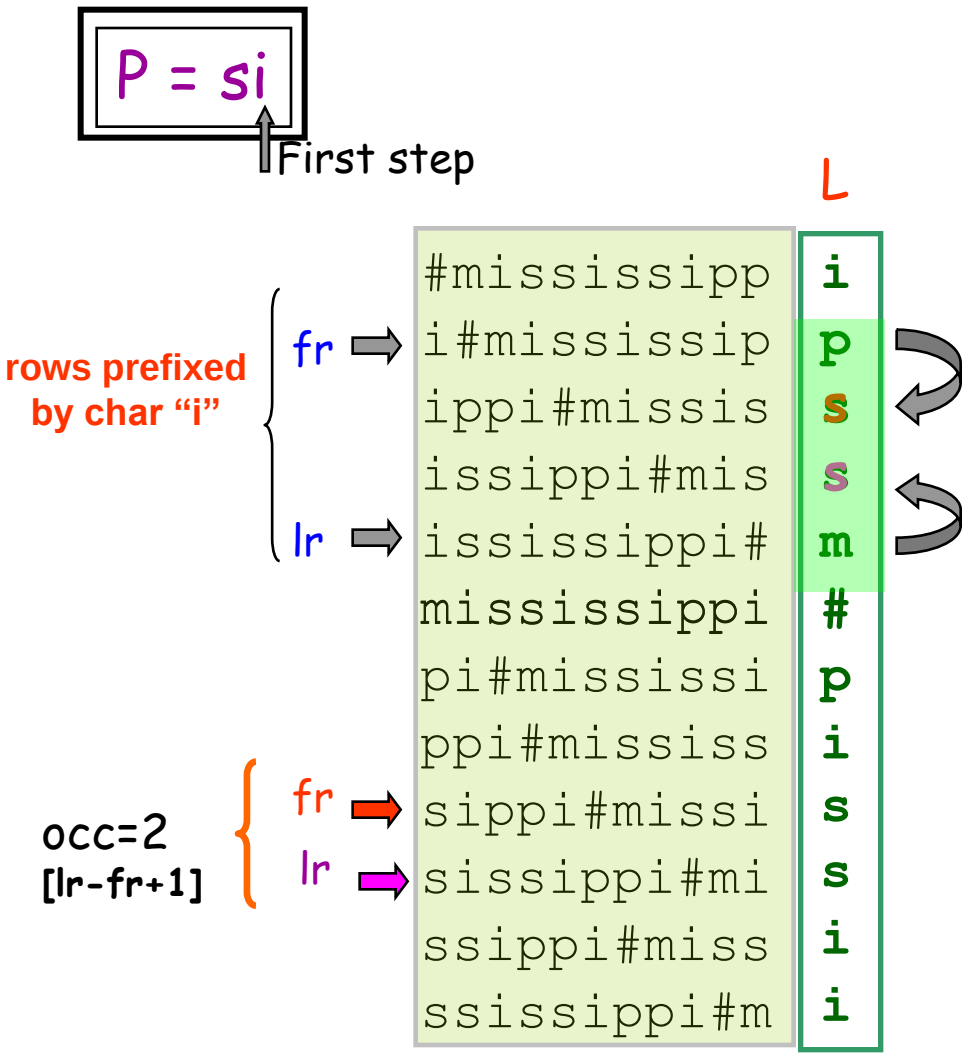
Inductive step: Given fr,lr for P[j+1,p]

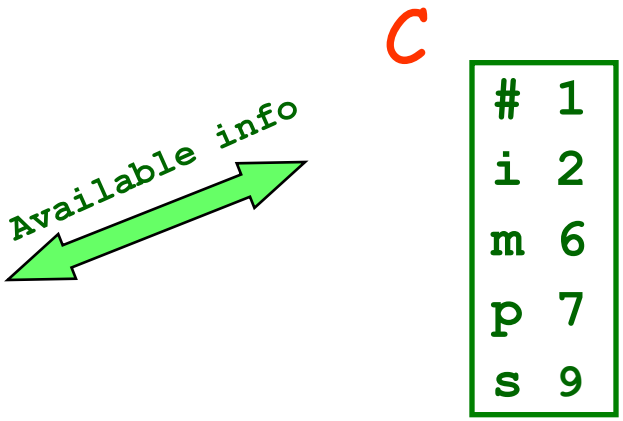
1. Take c=P[j]
2. Find the first c in L[fr, lr]
3. Find the last c in L[fr, lr]
4. L-to-F mapping of these chars

slide stolen from Paolo Ferragina@

Substring search using the BWT

(Count the pattern occurrences)





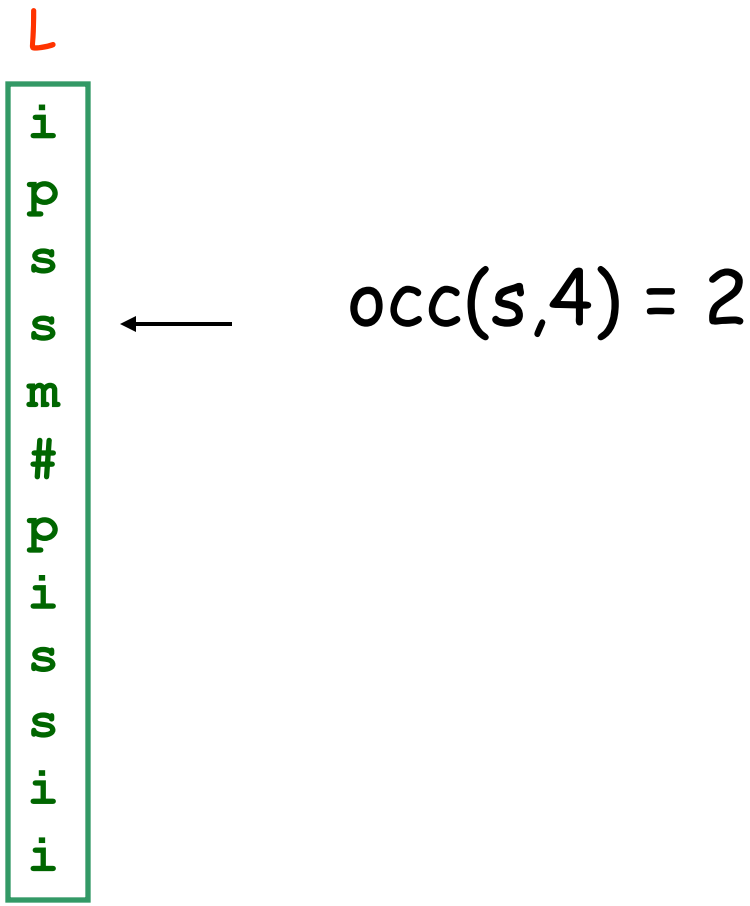
What if someone whispers how many "s" we have up to index 2 and up to index 5:

occ(s,2), occ(s,5) ?

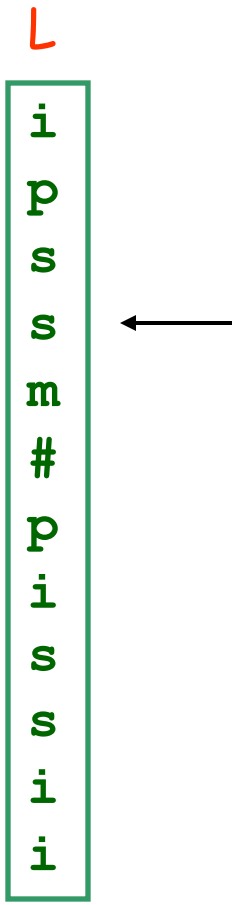
fr = C[s] + occ(s,2)
lr = C[s] + occ(s,5) - 1

slide stolen from Paolo Ferragina@

occ(a , j)



Make a bit vector for each character



0 0 1 1 0 0 0 0 1 1 0 0

occ(s,4) = rank(4)

rank(i) = how many ones are there before position i ?

How do you answer rank queries ?

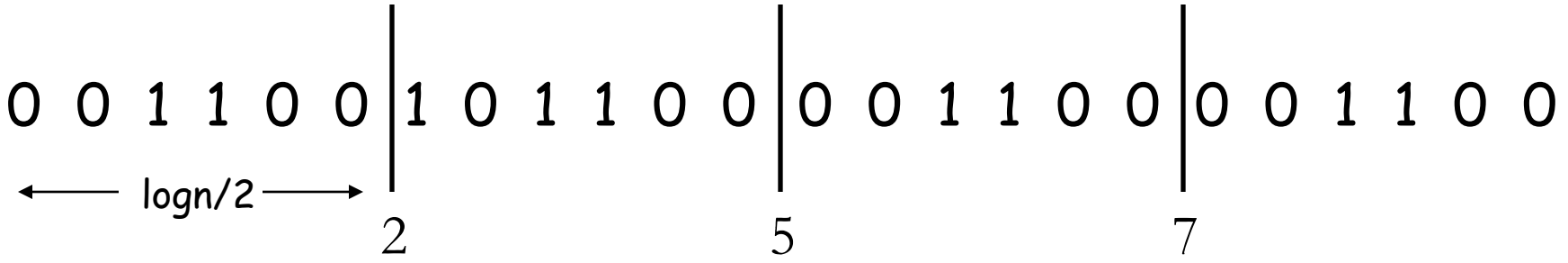
0 0 1 1 0 0 0 0 1 1 0 0

rank(i) = how many ones are there before position i ?

We can prepare a vector with all answers

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

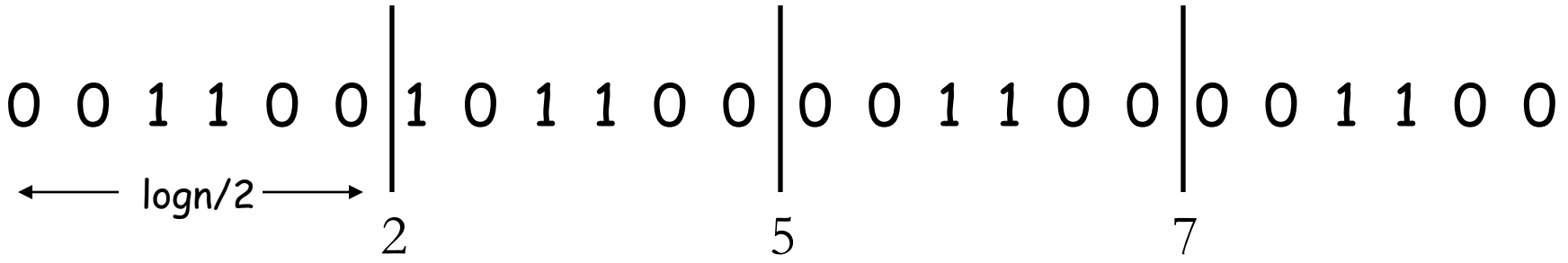
Lets do it with O(n) bits per character



Partition in 2n/log(n) blocks of size log(n)/2

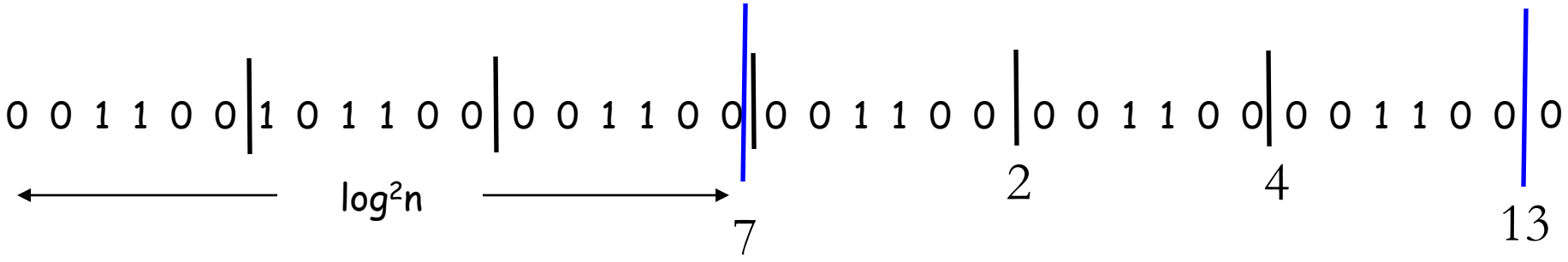
Keep the answer for each prefix of the blocks

There are $\sqrt{n}$ "kinds" of blocks, prepare a table with all answers for each block



In our solution the bit vector takes $\Theta(n)$ bits
and also the "additionals" take $\Theta(n)$ bits

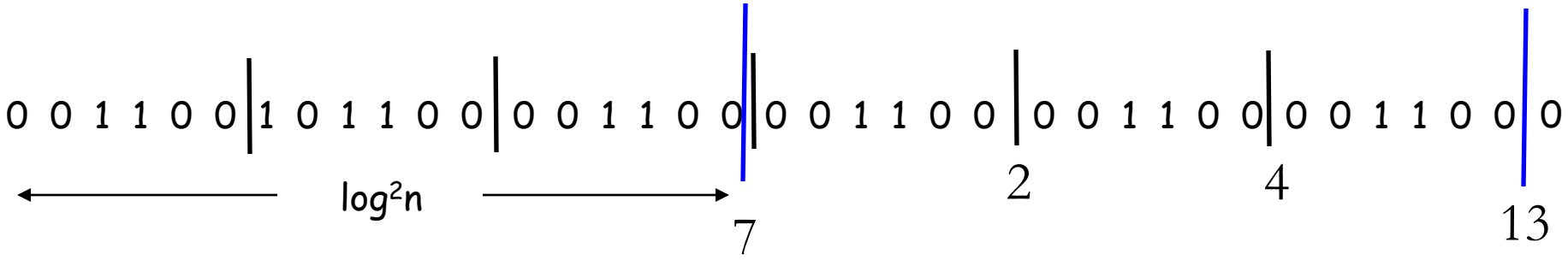
Can we do it with smaller overhead : so additionals would take o(n) ?



superblocks of size $\log^2(n)$

Each block keeps the number of one in previous blocks that are in the same superblock

Analysis



The superblock table is of size n/log (n)

The block table is of size (loglog(n)) * n/log (n)

The tables for the blocks √n log(n)loglog(n)

So the additionals take o(n) space

Next step

Do it without keeping the bit vectors themselves

Instead keep only the compressed version of the text

Saves a lot of space for compressible strings