Burrows-Wheeler Transform

(some of) its properties and applications

Rossano Venturini

Department of Computer Science
University of Pisa

Basic Concepts in Data Compression

- ***Lossless text data compression***:
  - We would like to design a compressor that, give a text in input, represents is using the smallest possible number of bits. From this representation we must be able to reconstruct the original text without any loss of information.

- **Historical motivations**:
  - Save storage space and/or bandwidth.

0-th order compressors

S	a	b	r	a	c	a	d	a	b	r	a
C											

0-th order compressors

S	a	b	r	a	c	a	d	a	b	r	a
C											

- Build a table: for each symbol stores its frequency

char	**freq**
a	5/11
b	2/11
c	1/11
d	1/11
r	2/11

0-th order compressors

S	a	b	r	a	c	a	d	a	b	r	a

C

- Build a table: for each symbol stores its frequency
- Assign a codeword to each symbol. So that,
  - Decompression: Codewords must be uniquely decodable.

char	freq	code
a	5/11	0
b	2/11	100
c	1/11	101
d	1/11	110
r	2/11	111

0-th order compressors

S	a	b	r	a	c	a	d	a	b	r	a
C											

- Build a table: for each symbol stores its frequency
- Assign a codeword to each symbol. So that,
  - Decompression: Codewords must be uniquely decodable.
  - Minimize compress size: Shortest codewords must be assigned to most frequent symbols.

char	**freq**	**code**
a	5/11	0
b	2/11	100
c	1/11	101
d	1/11	110
r	2/11	111

0-th order compressors

[Huffman, 1956]

S	a	b	r	a	c	a	d	a	b	r	a

C 0 100 111 0 101 0 110 0 100 111 0

- Build a table: for each symbol stores its frequency
- Assign a codeword to each symbol. So that,
  - Decompression: Codewords must be uniquely decodable.
  - Minimize compress size: Shortest codewords must be assigned to most frequent symbols.
- Replace each symbol with its codeword. Compress is C+Table

char	freq	code
a	5/11	0
b	2/11	100
c	1/11	101
d	1/11	110
r	2/11	111

0-th order compressors

[Huffman, 1956]

S	a	b	r	a	c	a	d	a	b	r	a

C	0 100 111 0 101 0 110 0 100 111 0

- Decompression is easy:
    - Scan C from left to right

char	freq	code
a	5/11	0
b	2/11	100
c	1/11	101
d	1/11	110
r	2/11	111

0-th order compressors

[Huffman, 1956]

S	a	b	r	a	c	a	d	a	b	r	a

C	0 100 111 0 101 0 110 0 100 111 0

- Decompression is easy:
  - Scan C from left to right
  - Every time we identify a codeword, we emit the corresponding symbol.

char	freq	code
a	5/11	0
b	2/11	100
c	1/11	101
d	1/11	110
r	2/11	111

0-th order compressors

[Huffman, 1956]

S	a	b	r	a	c	a	d	a	b	r	a

C	0 100 111 0 101 0 110 0 100 111 0

- Decompression is easy:
  - Scan C from left to right
  - Every tim w codewo corres

Low compression!
we don't exploit
regularities in text

char	freq	code
a	5/11	0
b	2/11	100
c	1/11	101
d	1/11	110
r	2/11	111

High order compressors

- We can achieve better compression if the codeword we assign to a symbol also depends on the k symbols preceding it (its context)

High order compressors

- We can achieve better compression if the codeword we assign to a symbol also depends on the k symbols preceding it (its context)

S	a	b	r	a	c	a	d	a	b	r	a

Build a table for each context of length k in S

High order compressors

- We can achieve better compression if the codeword we assign to a symbol also depends on the k symbols preceding it (its context)

S	a	b	r	a	c	a	d	a	b	r	a

Build a table for each context of length k in S

k=2

context = ab

High order compressors

- We can achieve better compression if the codeword we assign to a symbol also depends on the k symbols preceding it (its context)

S	a	b	r	a	c	a	d	a	b	r	a

Build a table for each context of length k in S

k=2

context = ab

char	freq	code
a	0/2	-
b	0/2	-
c	0/2	-
d	0/2	-
r	2/2	0

High order compressors

- We can achieve better compression if the codeword we assign to a symbol also depends on the k symbols preceding it (its context)

S	a	b	r	a	c	a	d	a	b	r	a

Build a table for each context of length k in S

We need to store a table for each context of size k.

char	freq	code
a	0/2	-
b	0/2	-
c	0/2	-
d	0/2	-
r	2/2	0

The models are the problem!

- The compression improves because we better predict the next symbol.

The models are the problem!

- The compression improves because we better predict the next symbol.
- Problem:
  - Larger is k, smaller is the compress

The models are the problem!

- The compression improves because we better predict the next symbol.
- Problem:
  - Larger is k, smaller is the compress
  - but we have to store more tables:
    - $O(\sigma^{k+1} \log \sigma)$ bits in the worst case

The models are the problem!

- The compression improves because we better predict the next symbol.
- Problem:
  - Larger is k, smaller is the compress
  - but we have to store more tables:
    - $O(\sigma^{k+1} \log \sigma)$ bits in the worst case
- Since compress size = |C|+ size tables, this approach require a lot of tuning in order to find the best value of k (i.e., the value of k that minimizes compress size)

The models are the problem!

- The compression improves because we better predict the next symbol.
- Problem:
  - Larger is k, smaller is the compress
  - but we have to store more tables:
    - $O(\sigma^{k+1} \log \sigma)$ bits in the worst case
- Since compress size = |C|+ size tables, this approach requires a lot of tuning in order to find the best value of k (i.e., the value of k that minimizes compress size)
- Instead, we would like to have a method that use a 0-th order compressor without care about the length of the contexts

Rearranging the input

- Idea!
  - Permute the input so that it is more compressible by a 0-th order compressor

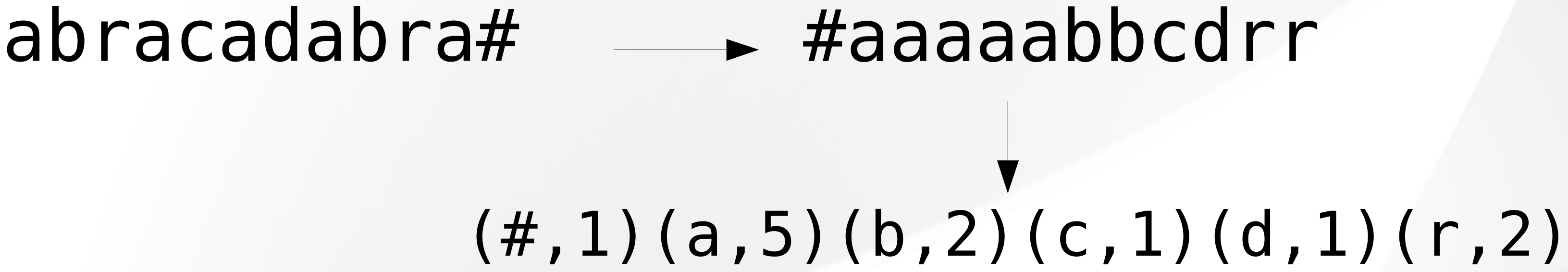
Rearranging the input

- Idea!
  - Permute the input so that it is more compressible by a 0-th order compressor
- Easiest way: sort the symbol lexicographically

```
abracadabra#   →   #aaaaabbcdrr
```

Rearranging the input

- Idea!
  - Permute the input so that it is more compressible by a 0-th order compressor
- Easiest way: sort the symbol lexicographically



```
abracadabra#  →  #aaaaabbcdrr
                      ↓
        (#,1)(a,5)(b,2)(c,1)(d,1)(r,2)
```

Rearranging the input

- Idea!
  - Permute the input s
    compressible by a C
- Easiest way: sort the

Best compression you can achieve!
Decoder must know at least the alphabet distribution.

```
abracadabra#  →  #aa      bcdrr
                     ↓
       (#,1)(a,5)(b,2)(c,1)(d,1)(r,2)
```

Rearranging the input

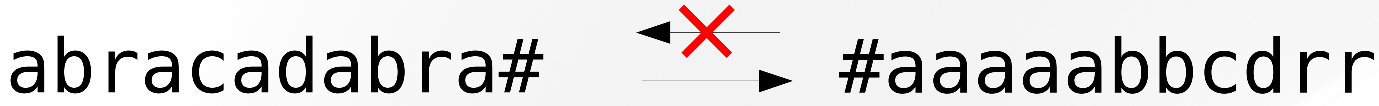
- Idea!
  - Permute the input so that it is more compressible by a 0-th order compressor
- Easiest way: sort the symbol lexicographically

```
abracadabra#   →   #aaaaabbcdrr
```

Which is the problem?

Rearranging the input

- Idea!
  - Permute the input so that it is more compressible by a 0-th order compressor
- Easiest way: sort the symbol lexicographically



abracadabra#

#aaaaabbcdrr

Which is the problem?

The transformation is not reversible!
There are 997.920 distinct strings with
this alphabet distribution!

Rearranging the input

- Idea!
  - Permute the input so that it is more compressible by a 0-th order compressor
- Easiest way: sort the symbol lexicographically

abracadabra# $\overset{\cancel{\longleftarrow}}{\longrightarrow}$ #aaaaabbcdrr

- What do you think about this one?

ard#rcaaaabb

Rearranging the input

- Idea!
  - Permute the input so that it is more compressible by a 0-th order compressor
- Easiest way: sort the symbol lexicographically

`abracadabra#` → `#aaaaabbcdrr`

- What do you think about this one?

`ard#rcaaaabb`

Similar, but it is reversible!

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = `abracadabra#`

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

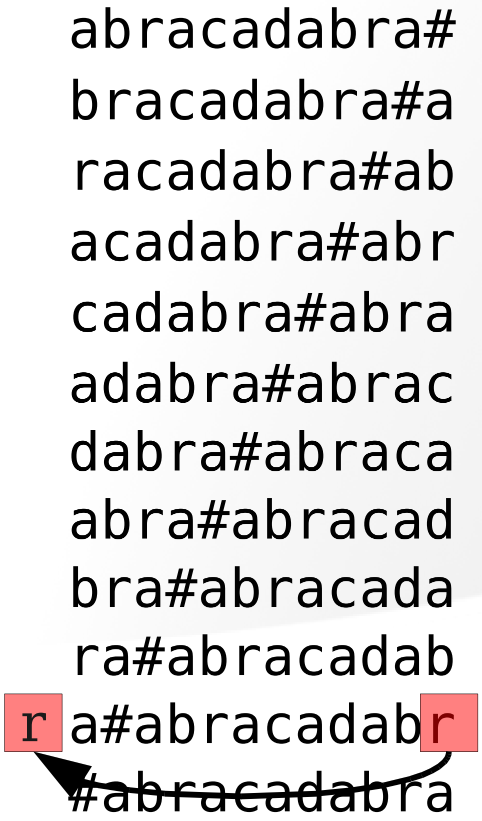
Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#



```
 abracadabra#
 bracadabra#a
 racadabra#ab
 acadabra#abr
 cadabra#abra
 adabra#abrac
 dabra#abraca
 abra#abracad
 bra#abracada
 ra#abracadab
r a#abracadabr
 #abracadabra
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```

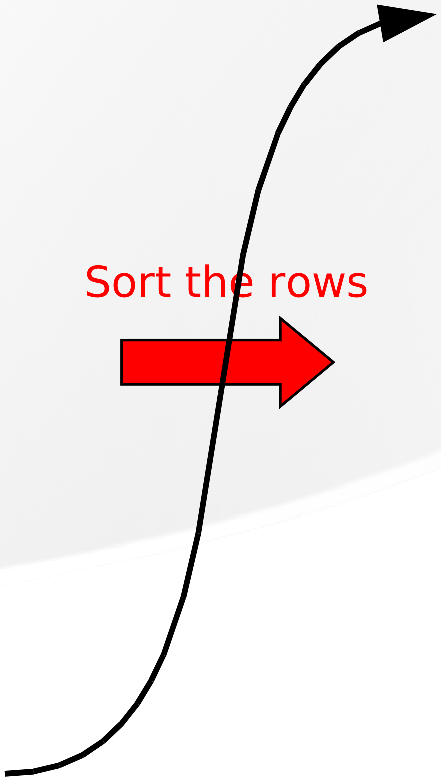
Sort the rows

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```



Sort the rows

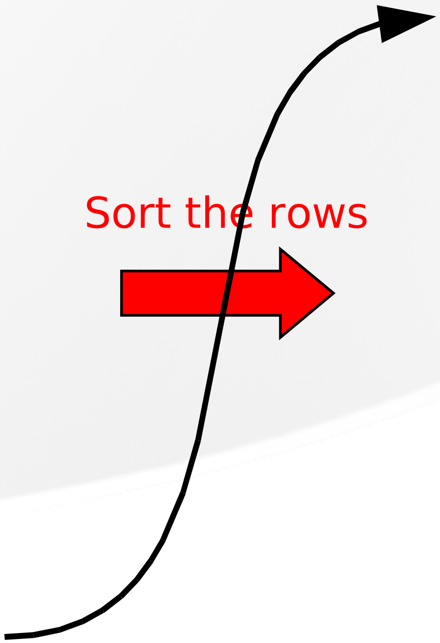
```
#   abracadabr   a
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```



Sort the rows

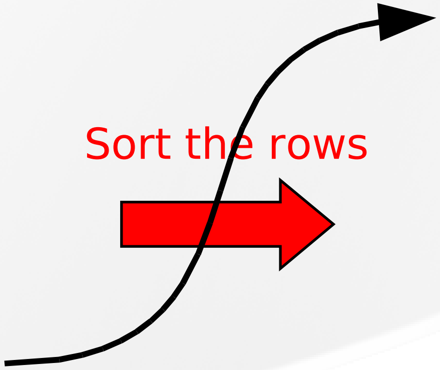
```
#   abracadabr   a
a   #abracadab   r
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```



Sort the rows

```
#   abracadabr   a
a   #abracadab   r
a   bra#abraca   d
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```

Sort the rows

```
#   abracadabr   a
a   #abracadab   r
a   bra#abraca   d
a   bracadabra   #
a   cadabra#ab   r
a   dabra#abra   c
b   ra#abracad   a
b   racadabra#   a
c   adabra#abr   a
d   abra#abrac   a
r   a#abracada   b
r   acadabra#a   b
```

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```

Sort the rows

F		L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#

```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```

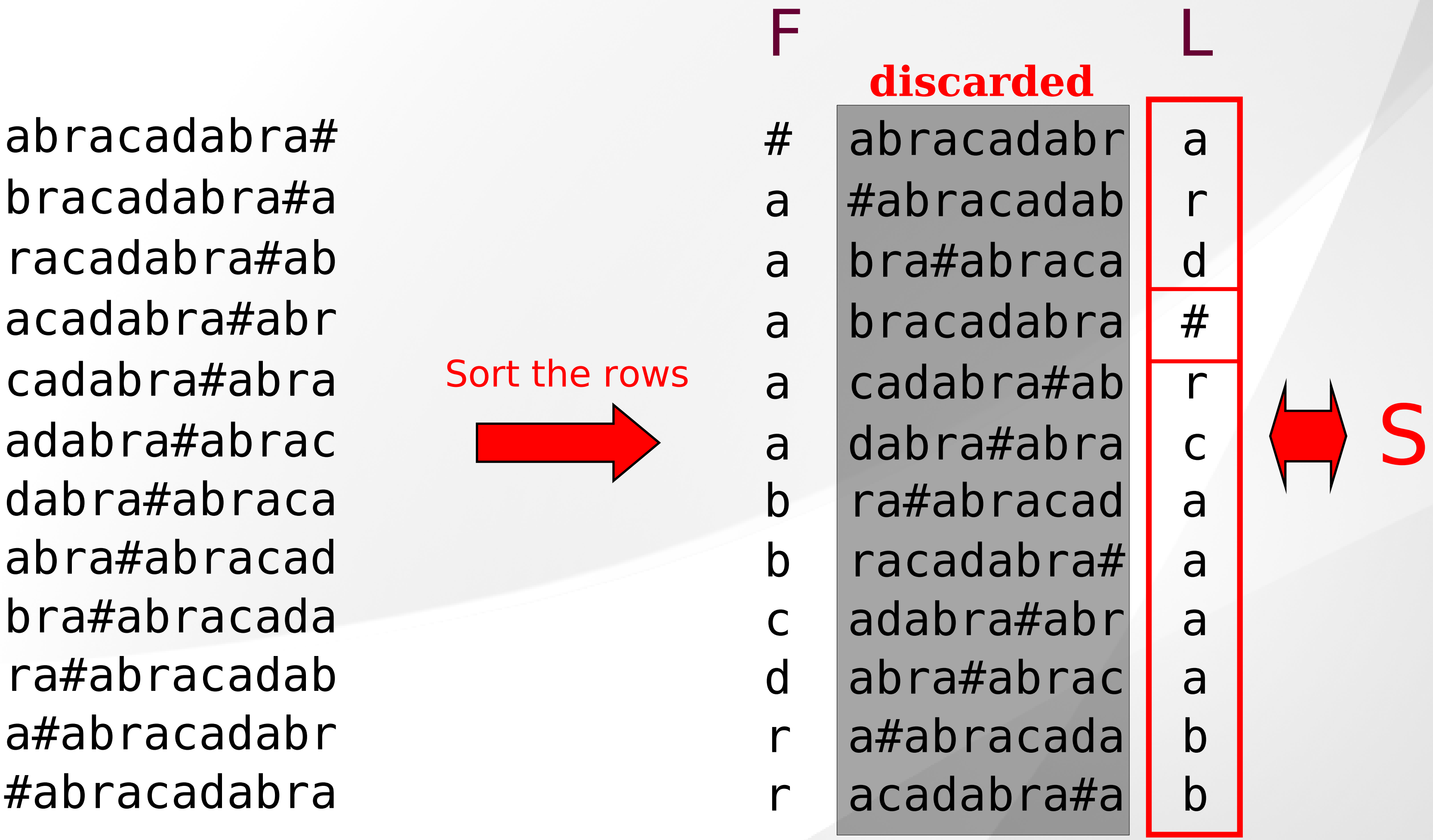
Sort the rows

F	discarded	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

Burrows-Wheeler Transform

[Burrows-Wheeler, 1994]

Let us given S = abracadabra#



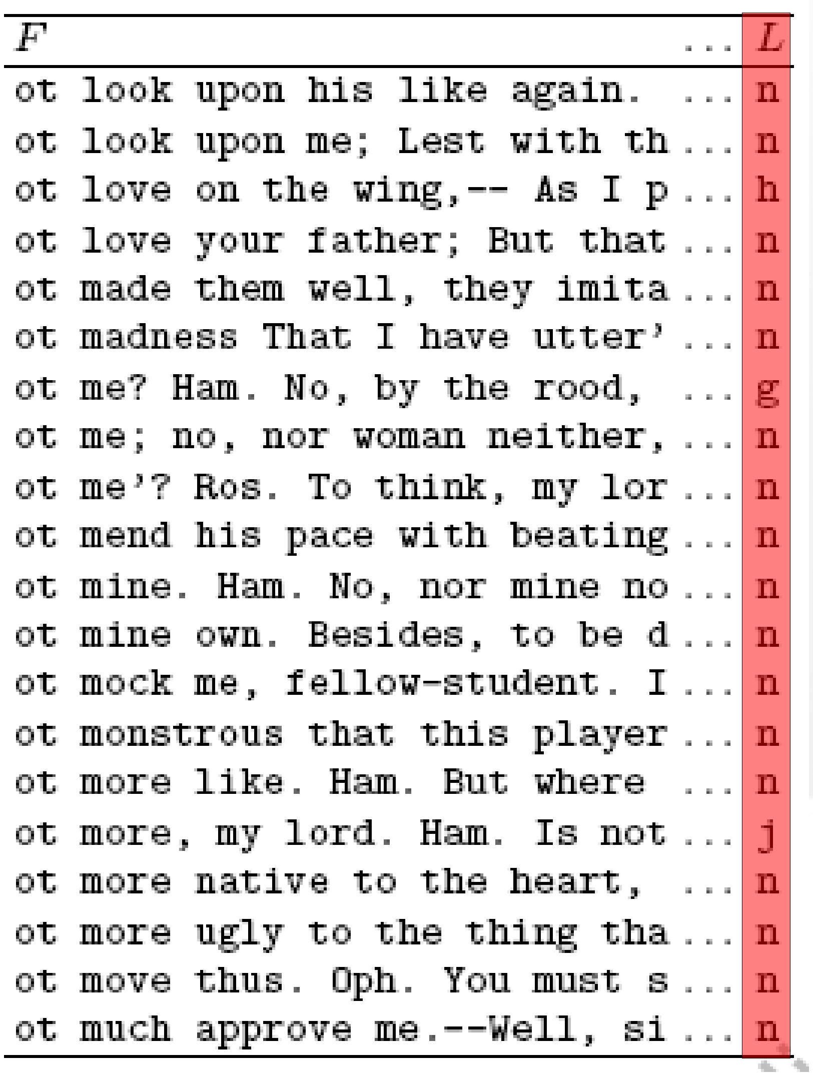
```
abracadabra#
bracadabra#a
racadabra#ab
acadabra#abr
cadabra#abra
adabra#abrac
dabra#abraca
abra#abracad
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra
```

Sort the rows

F	discarded	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

Burrows-Wheeler Transform: why it works

[Burrows-Wheeler, 1994]

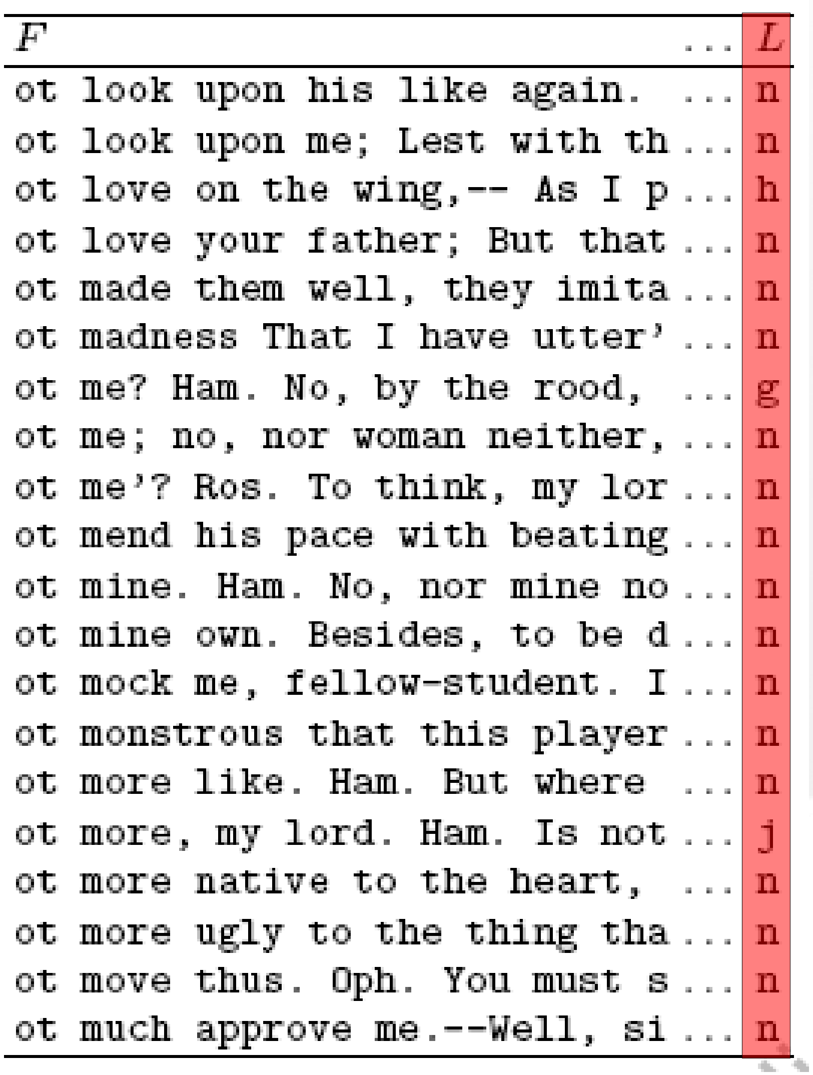


F	…	L
ot look upon his like again.	...	n
ot look upon me; Lest with th	...	n
ot love on the wing,-- As I p	...	h
ot love your father; But that	...	n
ot made them well, they imita	...	n
ot madness That I have utter'	...	n
ot me? Ham. No, by the rood,	...	g
ot me; no, nor woman neither,	...	n
ot me'? Ros. To think, my lor	...	n
ot mend his pace with beating	...	n
ot mine. Ham. No, nor mine no	...	n
ot mine own. Besides, to be d	...	n
ot mock me, fellow-student. I	...	n
ot monstrous that this player	...	n
ot more like. Ham. But where	...	n
ot more, my lord. Ham. Is not	...	j
ot more native to the heart,	...	n
ot more ugly to the thing tha	...	n
ot move thus. Oph. You must s	...	n
ot much approve me.--Well, si	...	n

Shakespeare's Hamlet

Burrows-Wheeler Transform: why it works

[Burrows-Wheeler, 1994]



F	...	L
ot look upon his like again.	...	n
ot look upon me; Lest with th	...	n
ot love on the wing,-- As I p	...	h
ot love your father; But that	...	n
ot made them well, they imita	...	n
ot madness That I have utter'	...	n
ot me? Ham. No, by the rood,	...	g
ot me; no, nor woman neither,	...	n
ot me'? Ros. To think, my lor	...	n
ot mend his pace with beating	...	n
ot mine. Ham. No, nor mine no	...	n
ot mine own. Besides, to be d	...	n
ot mock me, fellow-student. I	...	n
ot monstrous that this player	...	n
ot more like. Ham. But where	...	n
ot more, my lord. Ham. Is not	...	j
ot more native to the heart,	...	n
ot more ugly to the thing tha	...	n
ot move thus. Oph. You must s	...	n
ot much approve me.--Well, si	...	n

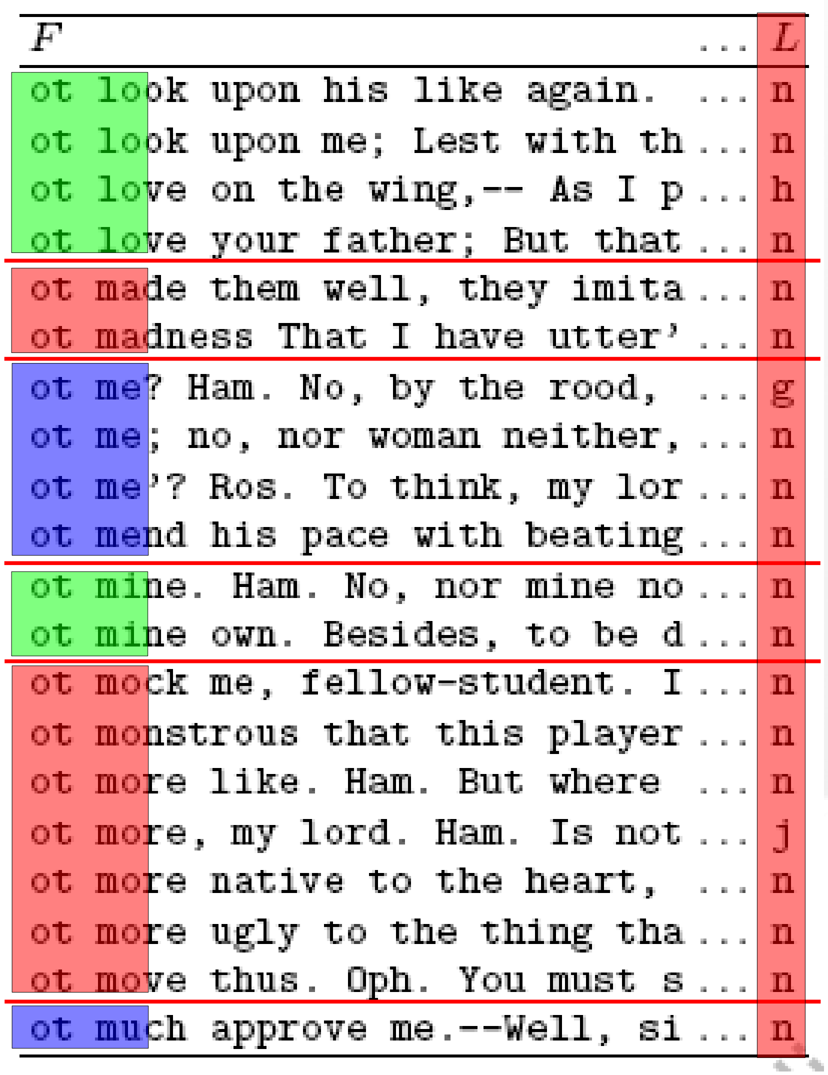
Shakespeare's Hamlet

L is locally homogeneous

Symbols followed by equal k-long contexts are clustered togheter

Burrows-Wheeler Transform: why it works

[Burrows-Wheeler, 1994]



F	...	L
ot look upon his like again.	...	n
ot look upon me; Lest with th	...	n
ot love on the wing,-- As I p	...	h
ot love your father; But that	...	n
ot made them well, they imita	...	n
ot madness That I have utter'	...	n
ot me? Ham. No, by the rood,	...	g
ot me; no, nor woman neither,	...	n
ot me'? Ros. To think, my lor	...	n
ot mend his pace with beating	...	n
ot mine. Ham. No, nor mine no	...	n
ot mine own. Besides, to be d	...	n
ot mock me, fellow-student. I	...	n
ot monstrous that this player	...	n
ot more like. Ham. But where	...	n
ot more, my lord. Ham. Is not	...	j
ot more native to the heart,	...	n
ot more ugly to the thing tha	...	n
ot move thus. Oph. You must s	...	n
ot much approve me.--Well, si	...	n

Shakespeare's Hamlet

L is locally homogeneous

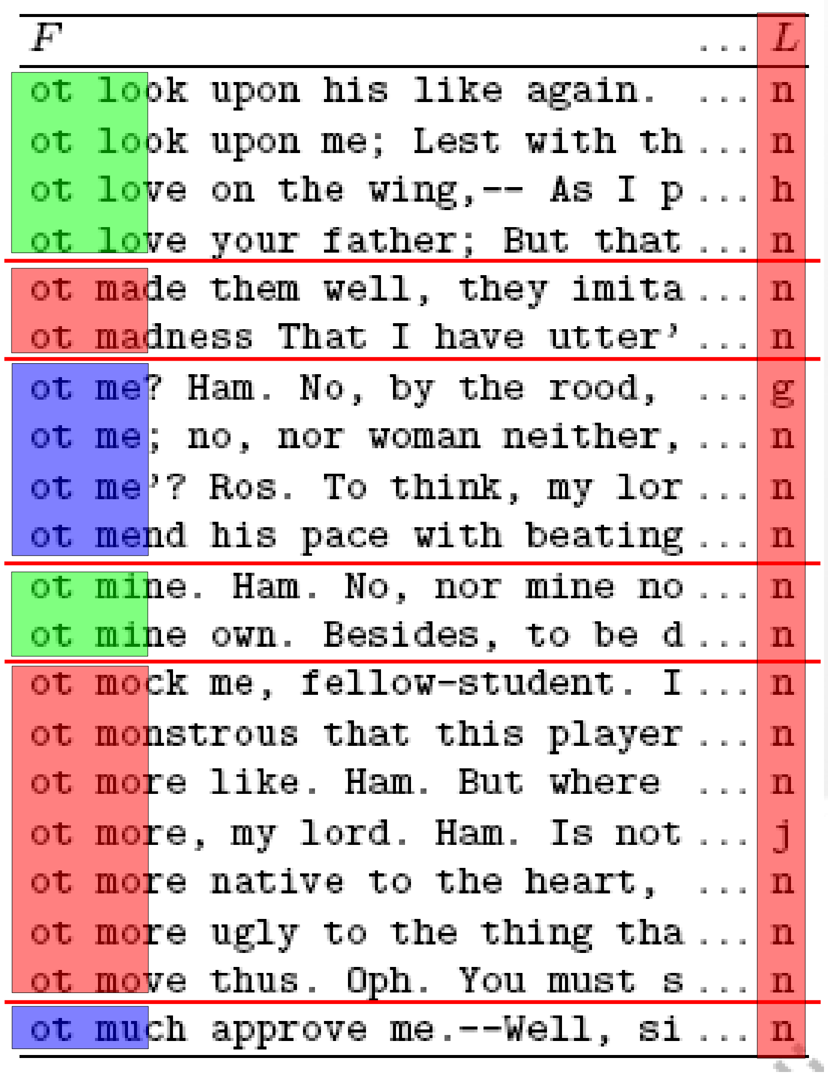
Symbols followed by equal k-long contexts are clustered togheter

For any length k

e.g., k=5

Burrows-Wheeler Transform: why it works

[Burrows-Wheeler, 1994]



F	...	L
ot look upon his like again.	...	n
ot look upon me; Lest with th	...	n
ot love on the wing,-- As I p	...	h
ot love your father; But that	...	n
ot made them well, they imita	...	n
ot madness That I have utter'	...	n
ot me? Ham. No, by the rood,	...	g
ot me; no, nor woman neither,	...	n
ot me'? Ros. To think, my lor	...	n
ot mend his pace with beating	...	n
ot mine. Ham. No, nor mine no	...	n
ot mine own. Besides, to be d	...	n
ot mock me, fellow-student. I	...	n
ot monstrous that this player	...	n
ot more like. Ham. But where	...	n
ot more, my lord. Ham. Is not	...	j
ot more native to the heart,	...	n
ot more ugly to the thing tha	...	n
ot move thus. Oph. You must s	...	n
ot much approve me.--Well, si	...	n

Shakespeare's Hamlet

L is locally homogeneous

Symbols followed by equal k-long contexts are clustered togheter

thus, L is highly compressible

Reverse the BWT

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

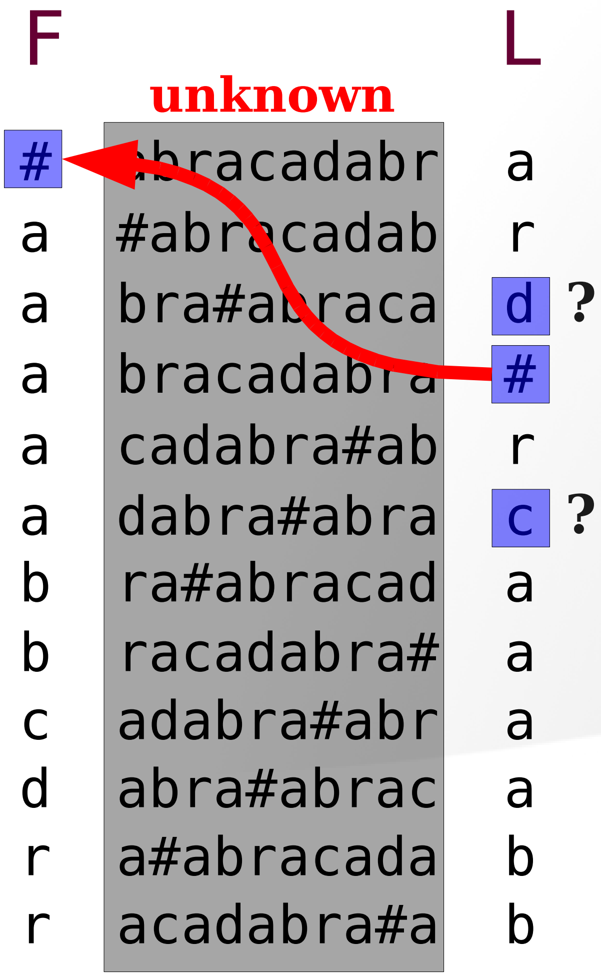
F	unknown	L	
#	abracadabr	a	
a	#abracadab	r	
a	bra#abraca	d	?
a	bracadabra	#	?
a	cadabra#ab	r	
a	dabra#abra	c	?
b	ra#abracad	a	
b	racadabra#	a	
c	adabra#abr	a	
d	abra#abrac	a	
r	a#abracada	b	
r	acadabra#a	b	

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Simple! For symbols with just one occurrence.

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



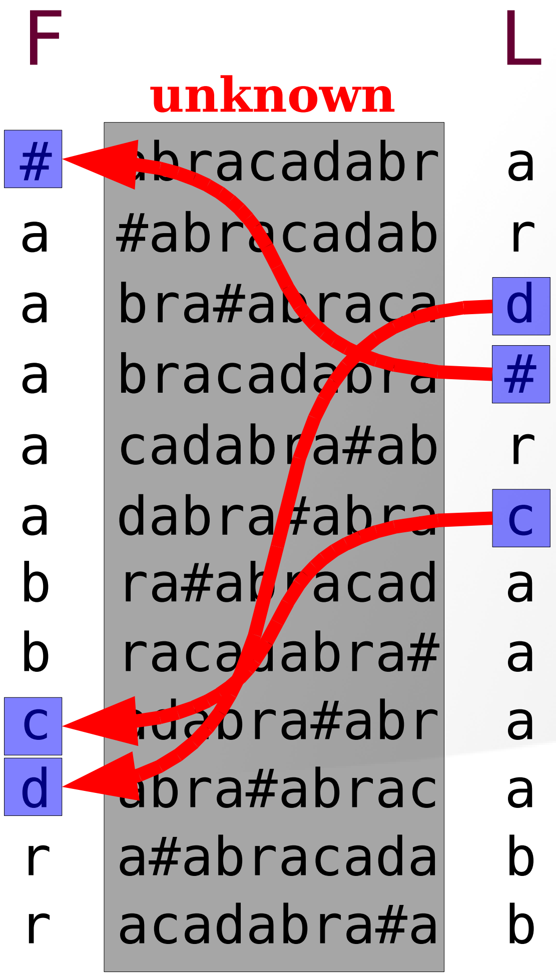
F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d ?
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c ?
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Simple! For symbols with just one occurrence.

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Simple! For symbols with just one occurrence.

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

F	unknown	L	
#	abracadabr	a	?
a	#abracadab	r	
a	bra#abraca	d	
a	bracadabra	#	
a	cadabra#ab	r	
a	dabra#abra	c	
b	ra#abracad	a	?
b	racadabra#	a	?
c	adabra#abr	a	?
d	abra#abrac	a	?
r	a#abracada	b	
r	acadabra#a	b	

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Symbols with more than one occurrence?

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

F	unknown	L	
#	abracadabr	a	?
a	#abracadab	r	
a	bra#abraca	d	
a	bracadabra	#	
a	cadabra#ab	r	
a	dabra#abra	c	
b	ra#abracad	a	?
b	racadabra#	a	?
c	adabra#abr	a	?
d	abra#abrac	a	?
r	a#abracada	b	
r	acadabra#a	b	

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

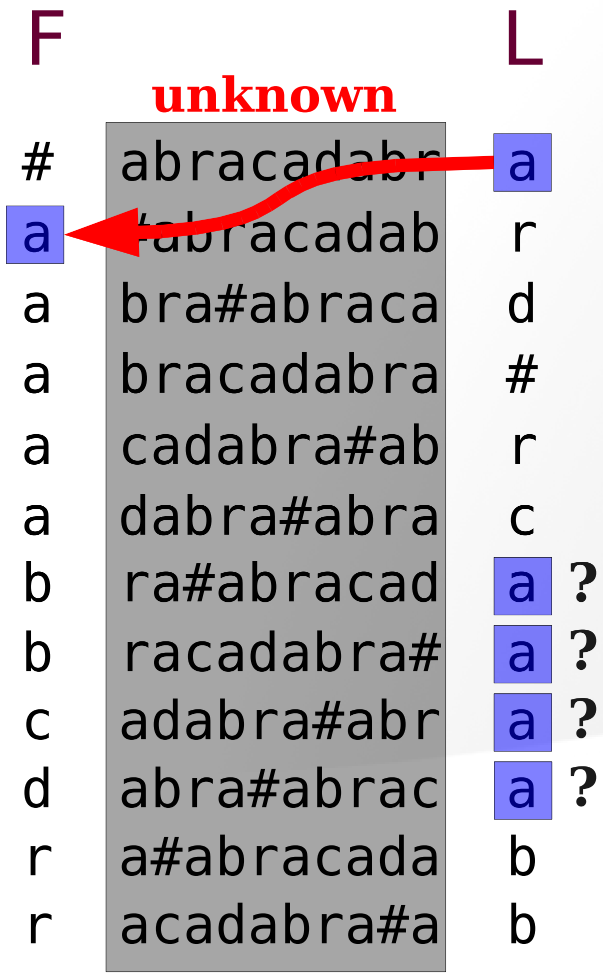
Key Property:

Equal symbols in L maintain their relative order in F:

First a in L maps to first a in F,…

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L	
#	abracadabr	a	
a	#abracadab	r	
a	bra#abraca	d	
a	bracadabra	#	
a	cadabra#ab	r	
a	dabra#abra	c	
b	ra#abracad	a	?
b	racadabra#	a	?
c	adabra#abr	a	?
d	abra#abrac	a	?
r	a#abracada	b	
r	acadabra#a	b	

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

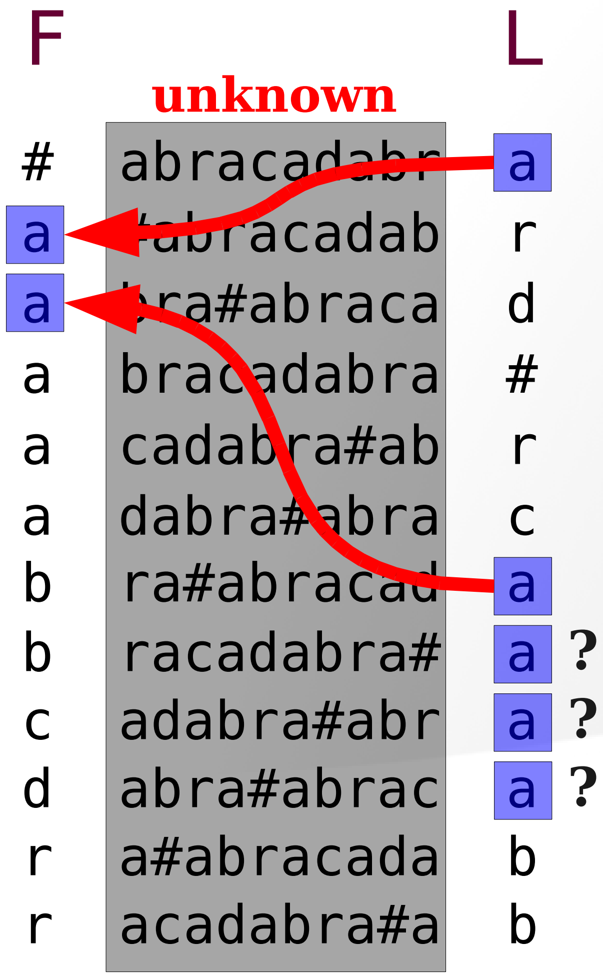
Key Property:

Equal symbols in L maintain their relative order in F:

First a in L maps to first a in F,…

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L	
#	abracadabr	a	
a	#abracadab	r	
a	bra#abraca	d	
a	bracadabra	#	
a	cadabra#ab	r	
a	dabra#abra	c	
b	ra#abracad	a	
b	racadabra#	a	?
c	adabra#abr	a	?
d	abra#abrac	a	?
r	a#abracada	b	
r	acadabra#a	b	

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

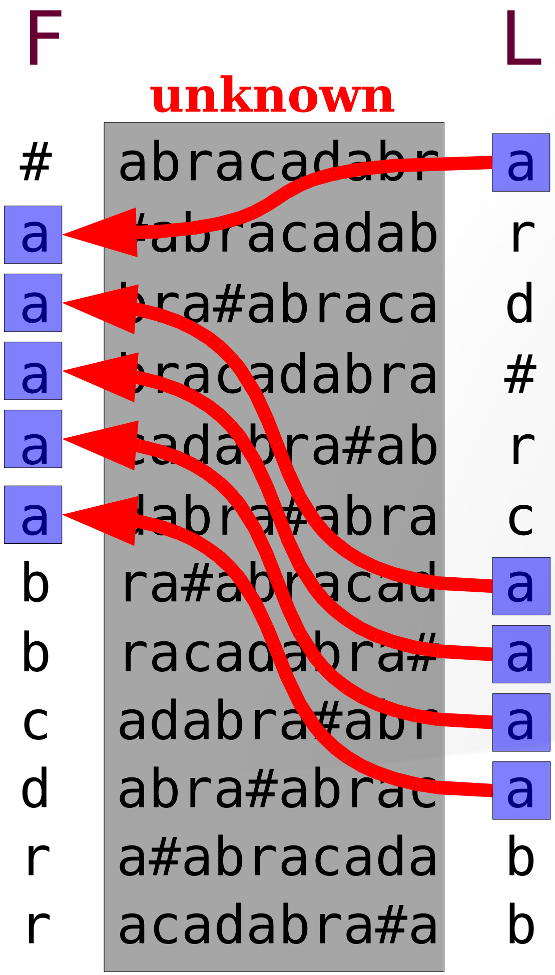
Key Property:

Equal symbols in L maintain their relative order in F:

First a in L maps to first a in F,…

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Key Property:

Equal symbols in L maintain their relative order in F:

First a in L maps to first a in F,…

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

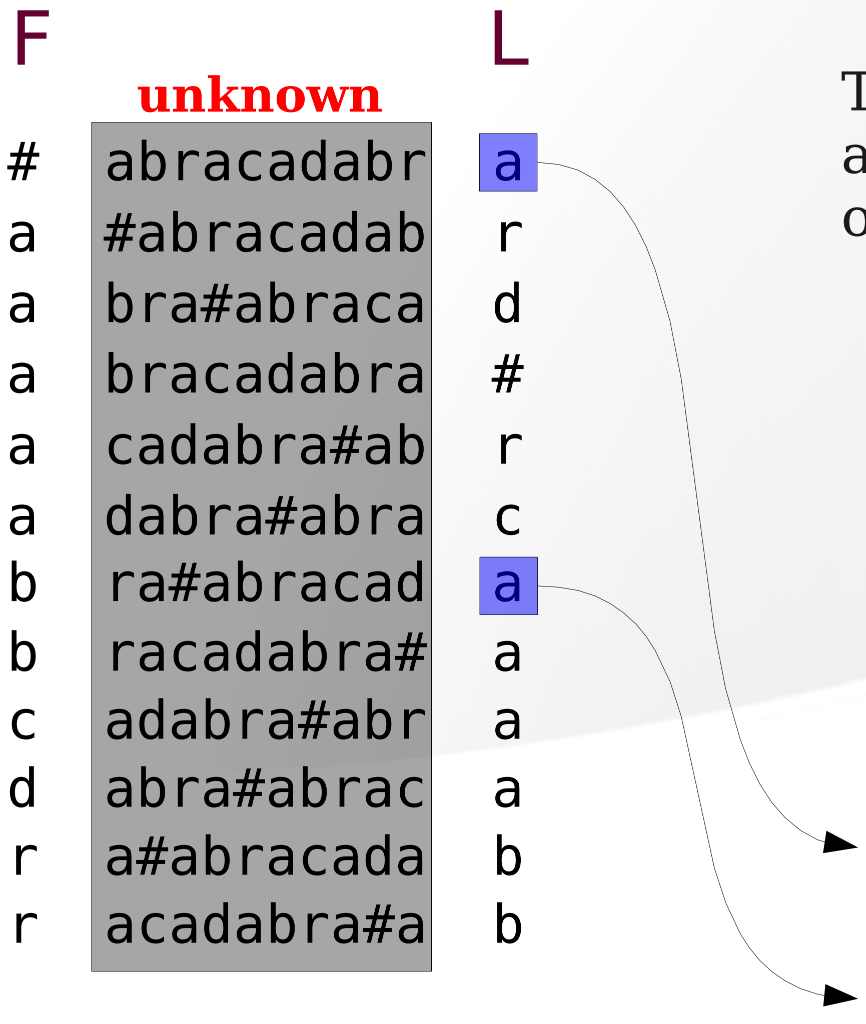
Key Property:

Equal symbols in L maintain their relative order in F:
First a in L maps to first a in F,…

Proof

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Key Property:
Equal symbols in L maintain their relative order in F:
First a in L maps to first a in F,…

Proof

#abracadabra

bra#abracada

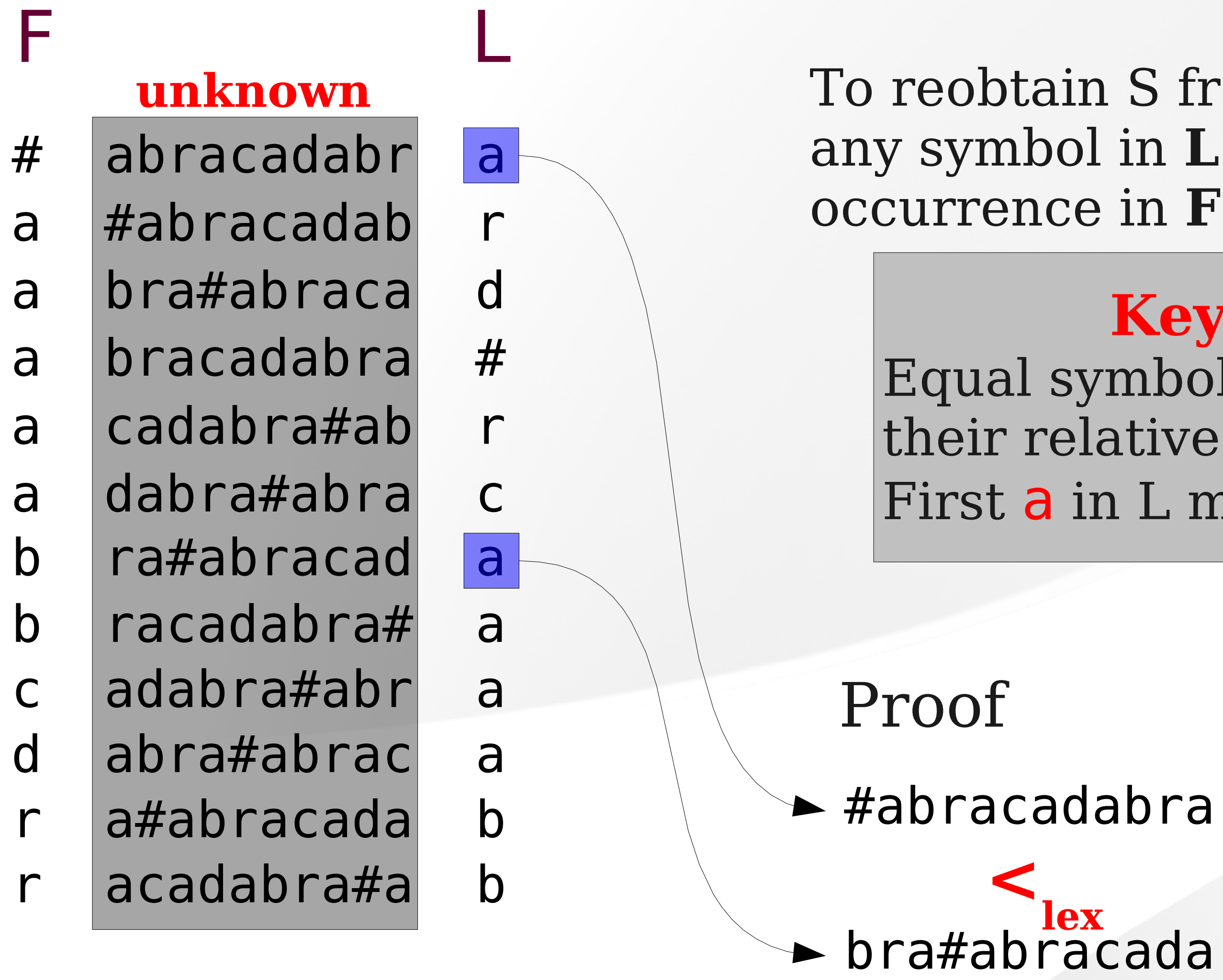
Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Key Property:

Equal symbols in L maintain their relative order in F:
First a in L maps to first a in F,…



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

Proof

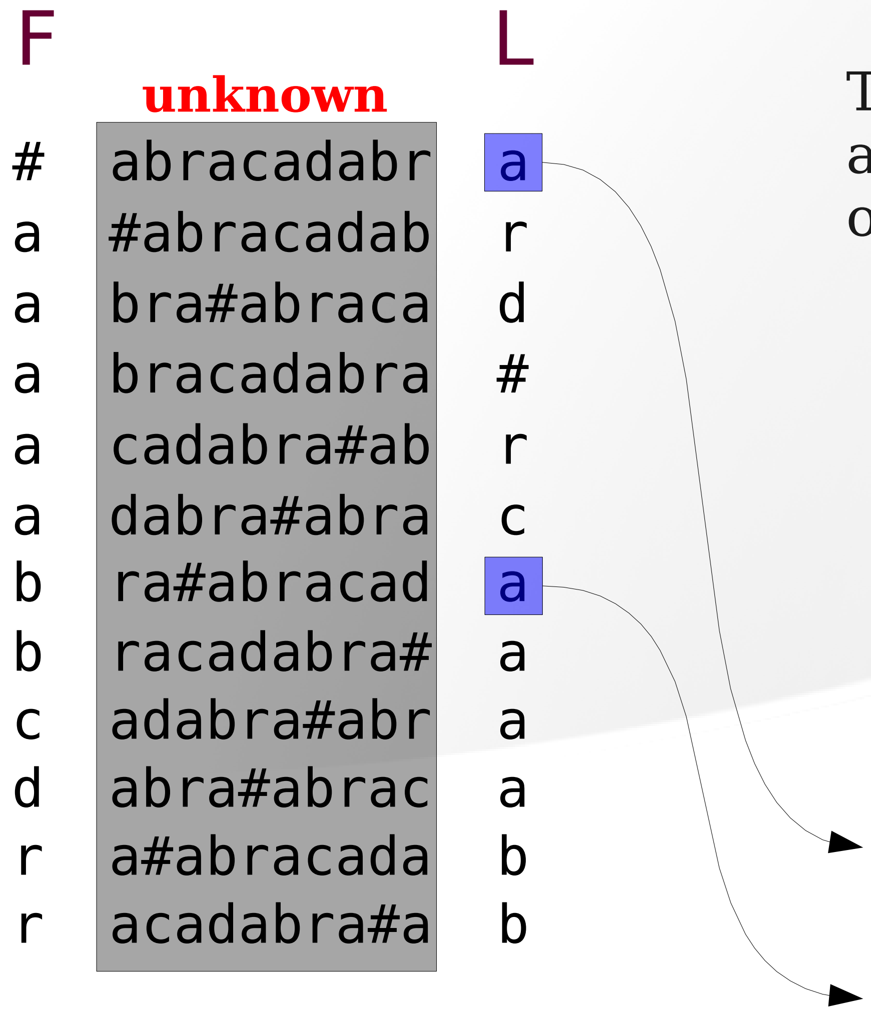
#abracadabra

$<_{lex}$

bra#abracada

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Key Property:
Equal symbols in L maintain their relative order in F:
First a in L maps to first a in F,…

Proof

Rotate them rightward to obtain their cyclic-rot

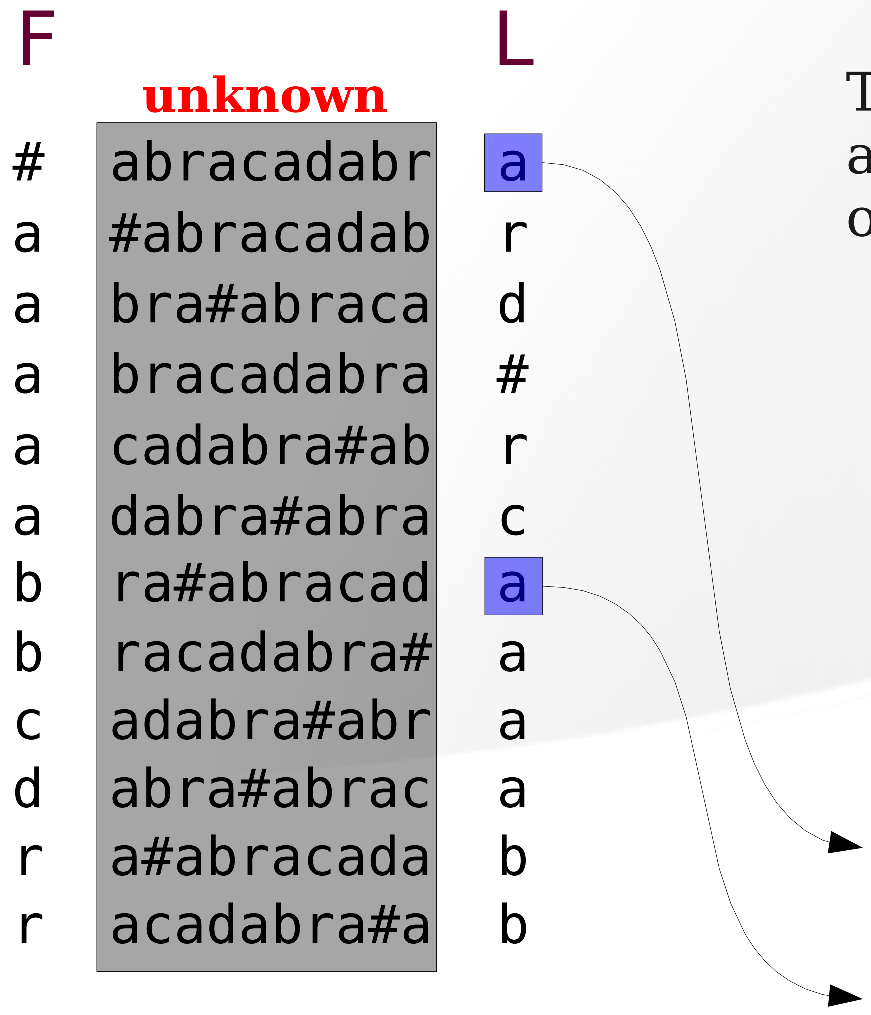
#abracadabra

$<_{\text{lex}}$

bra#abracada

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Key Property:
Equal symbols in L maintain their relative order in F:
First a in L maps to first a in F,…

Proof

Rotate them rightward i.e., move the a in front

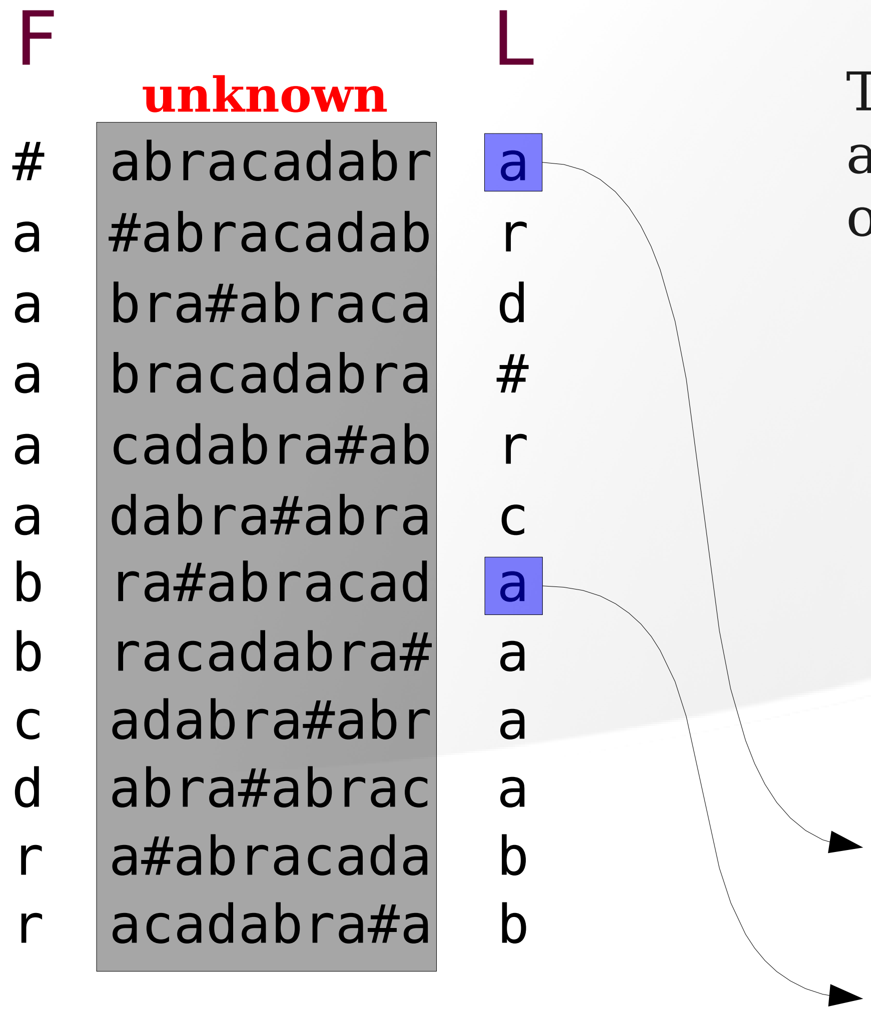
#abracadabra   a#abracadabr

$<_{lex}$

bra#abracada   abra#abracad

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Key Property:
Equal symbols in L maintain their relative order in F:
First a in L maps to first a in F,…

Proof

#abracadabra

$<_{\mathbf{lex}}$

bra#abracada

⇒

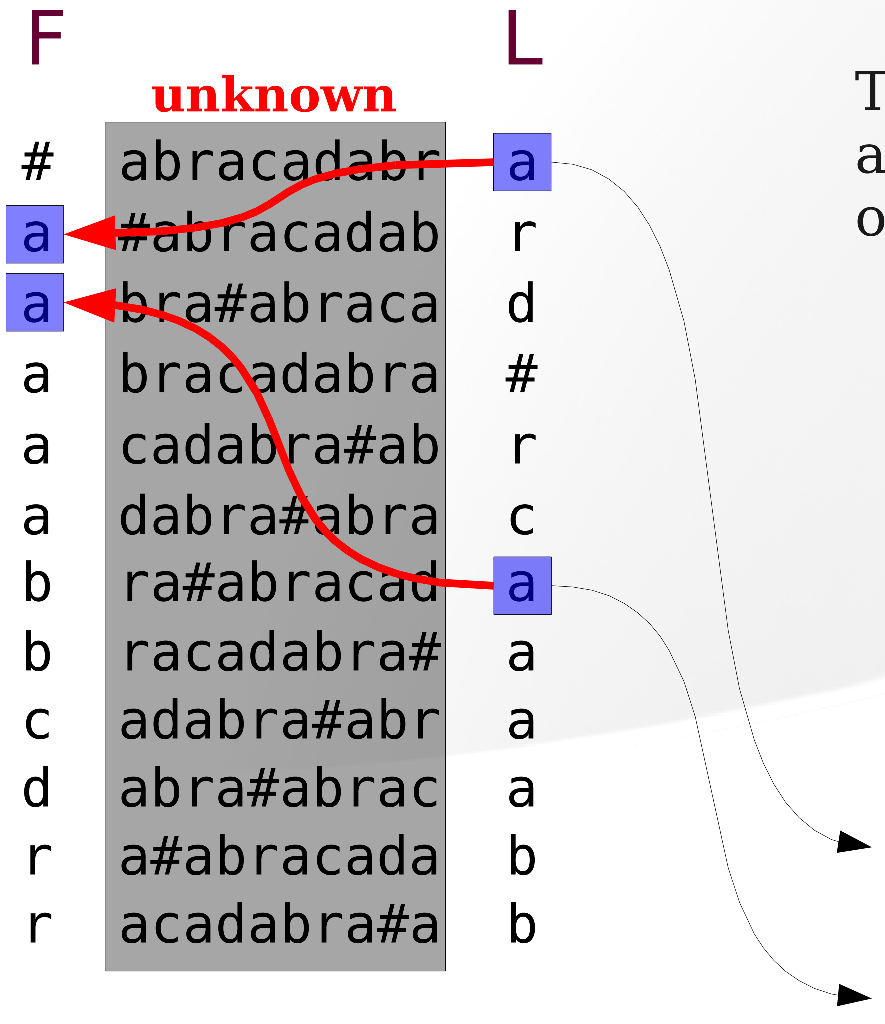
a#abracadabr

$<_{\mathbf{lex}}$

abra#abracad

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

To reobtain S from L we need to map any symbol in **L** to its corresponding occurrence in **F. LF-Mapping**

Key Property:
Equal symbols in L maintain their relative order in F:
First a in L maps to first a in F,…

Proof

$$\texttt{\#abracadabra} <_{\text{lex}} \texttt{bra\#abracada} \Rightarrow \texttt{a\#abracadabr} <_{\text{lex}} \texttt{abra\#abracad}$$

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

F	unknown	L
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

Two Key properties:

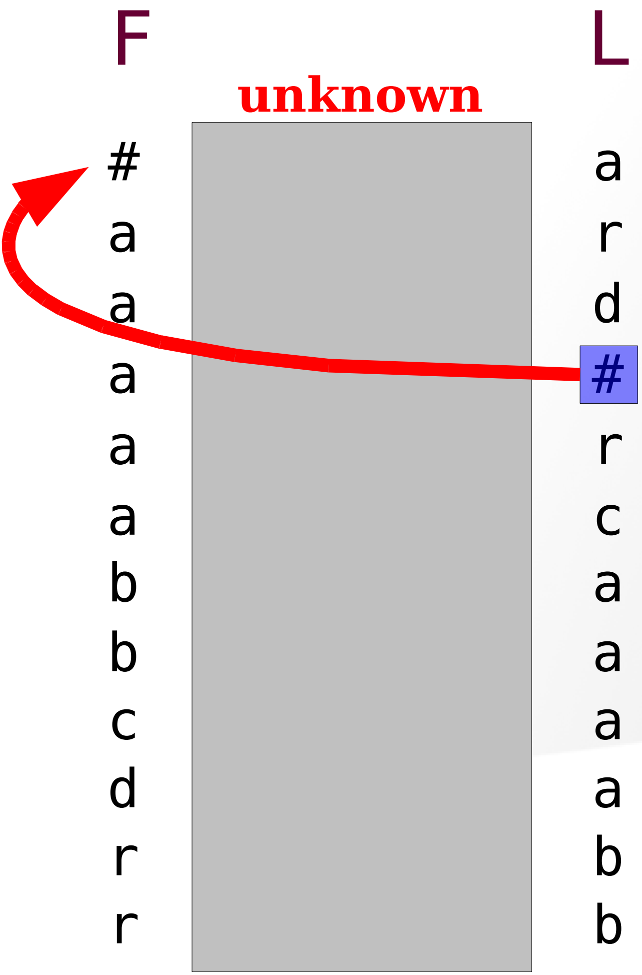
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= -----------#

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

Two Key properties:

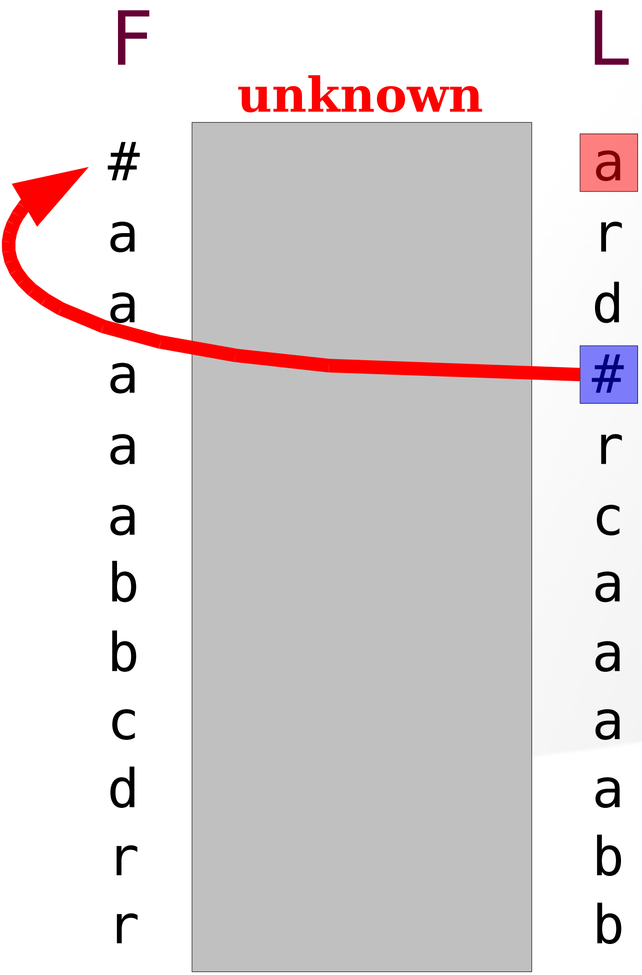
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= -----------#

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



unknown

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

We reconstruct S backward

Two Key properties:

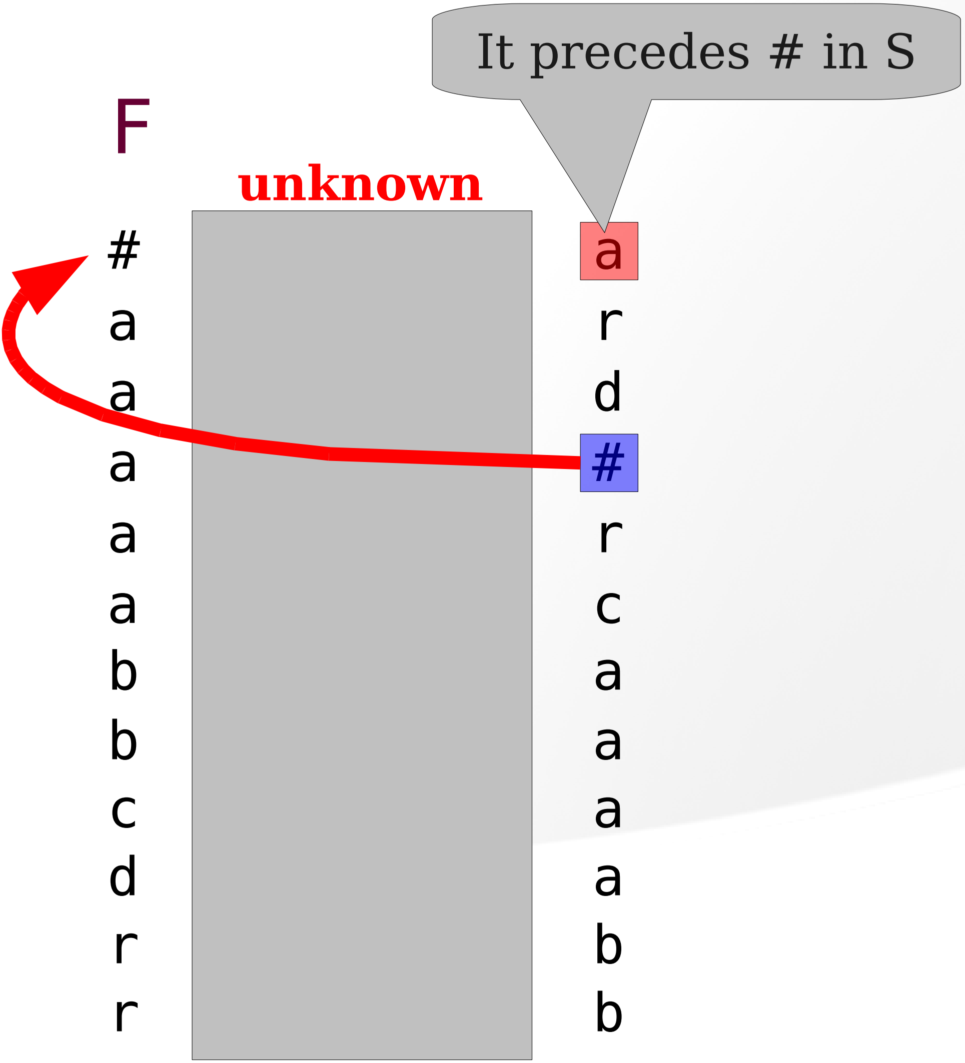
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= -----------#

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



It precedes # in S

F	unknown	
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

Two Key properties:

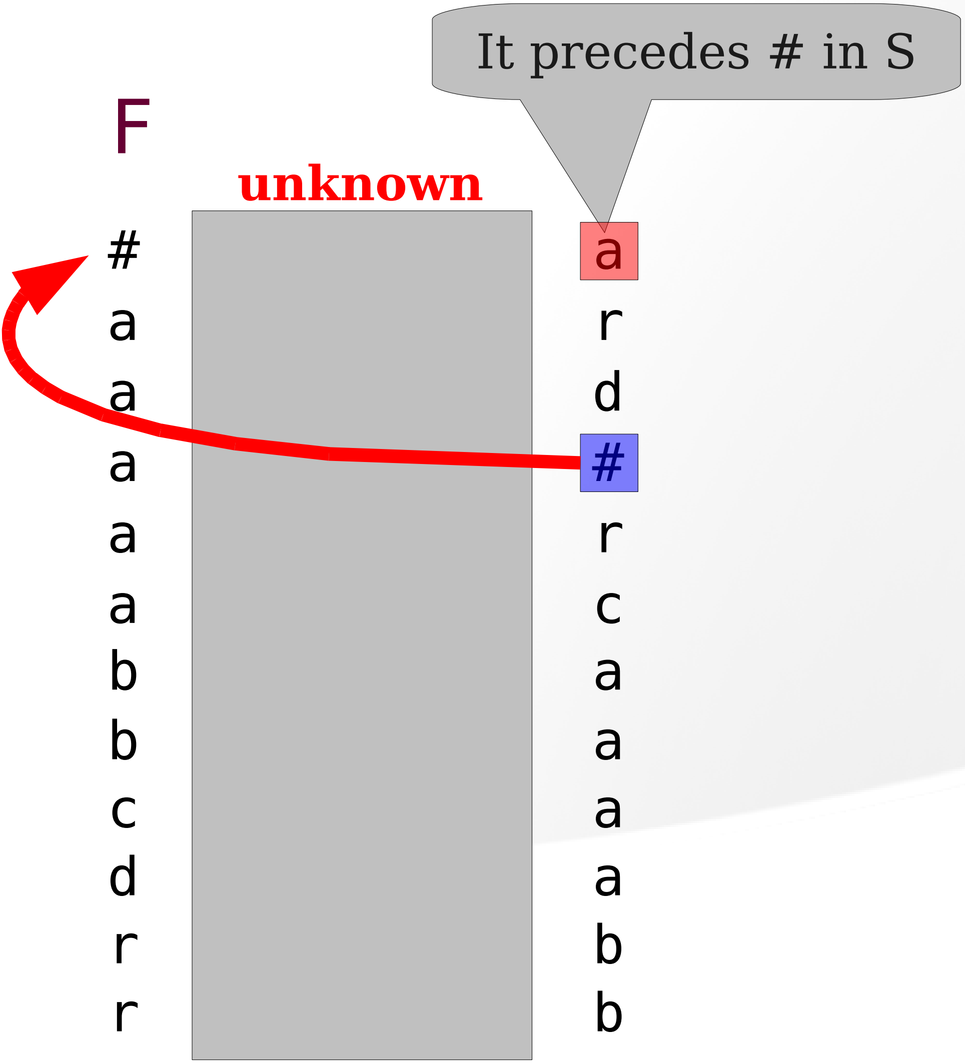
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= - - - - - - - - - - -#

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



It precedes # in S

F	unknown	
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

Two Key properties:

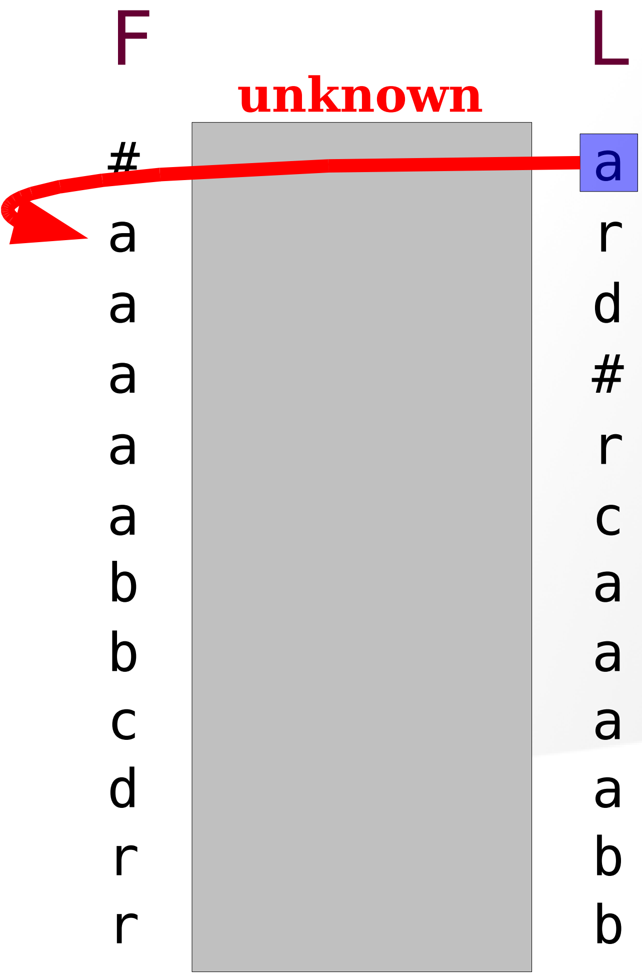
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= ----------**a#**

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



We reconstruct S backward

Two Key properties:

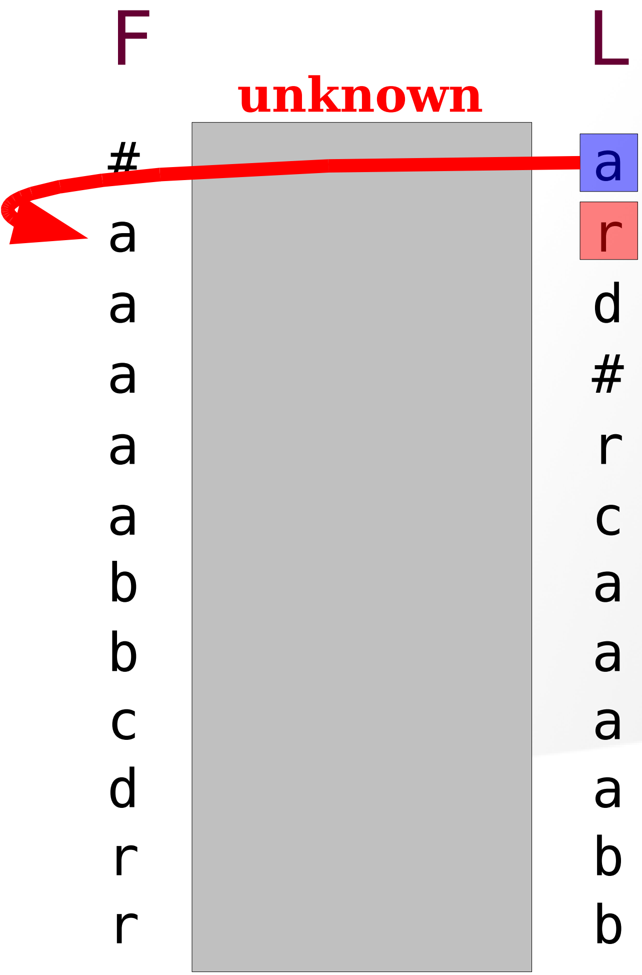
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= - - - - - - - - - - **a#**

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

Two Key properties:

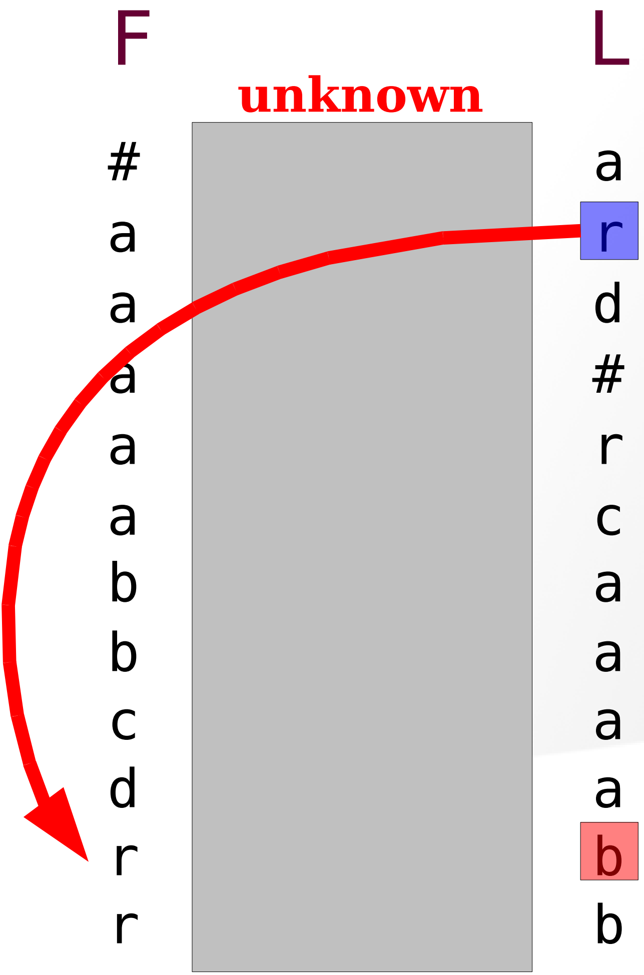
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= **-----------ra#**

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



unknown

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

We reconstruct S backward

Two Key properties:

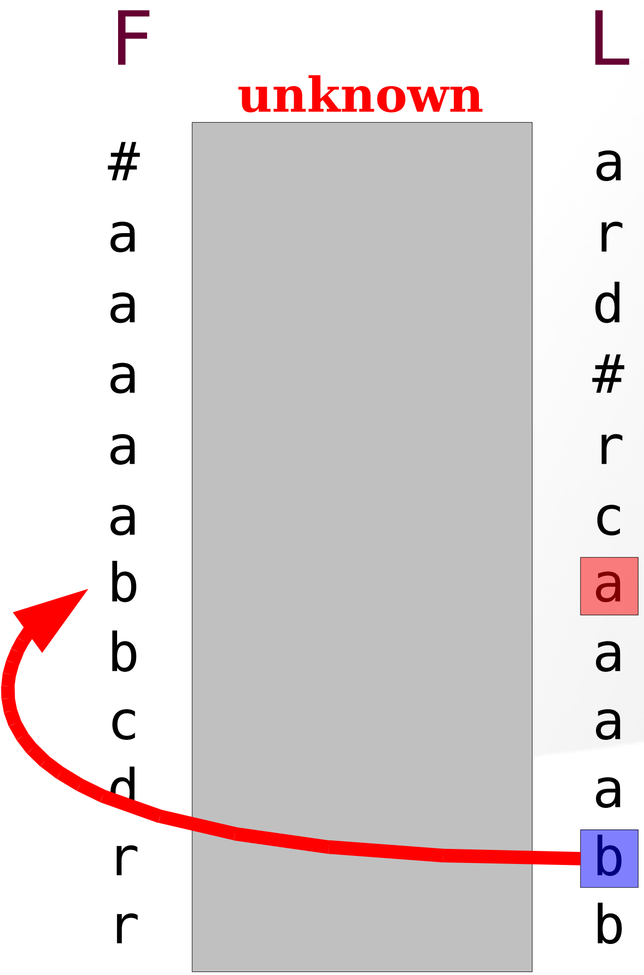
1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= **---------bra#**

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]



F	unknown	L
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

Two Key properties:

1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= **--------abra#**

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

F	unknown	L
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

Two Key properties:

1) LF maps L's to F's symbols

2) L[i] precedes F[i] in S

S= **abracadabra#**

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

F	unknown	L
#		a
a		r
a		d
a		#
a		r
a		c
b		a
b		a
c		a
d		a
r		b
r		b

We reconstruct S backward

```
InvertBWT(L)
Compute LF[0,n-1];
r = 0; i = n;
while (i>0) {
    T[i] = L[r];
    r = LF[r]; i--;
    }
```

ies:

s symbols

in S

S= **abracadabra#**

Burrows-Wheeler Transform: reversibility

[Burrows-Wheeler, 1994]

Starting from L, the LF-mapping can be computed in linear time using a simple algorithm

F	unknown	L	LF
#	abracadabr	a	1
a	#abracadab	r	10
a	bra#abraca	d	9
a	bracadabra	#	0
a	cadabra#ab	r	11
a	dabra#abra	c	8
b	ra#abracad	a	2
b	racadabra#	a	3
c	adabra#abr	a	4
d	abra#abrac	a	5
r	a#abracada	b	6
r	acadabra#a	b	7

C

#	a	b	c	d	r
1	6	8	9	10	12

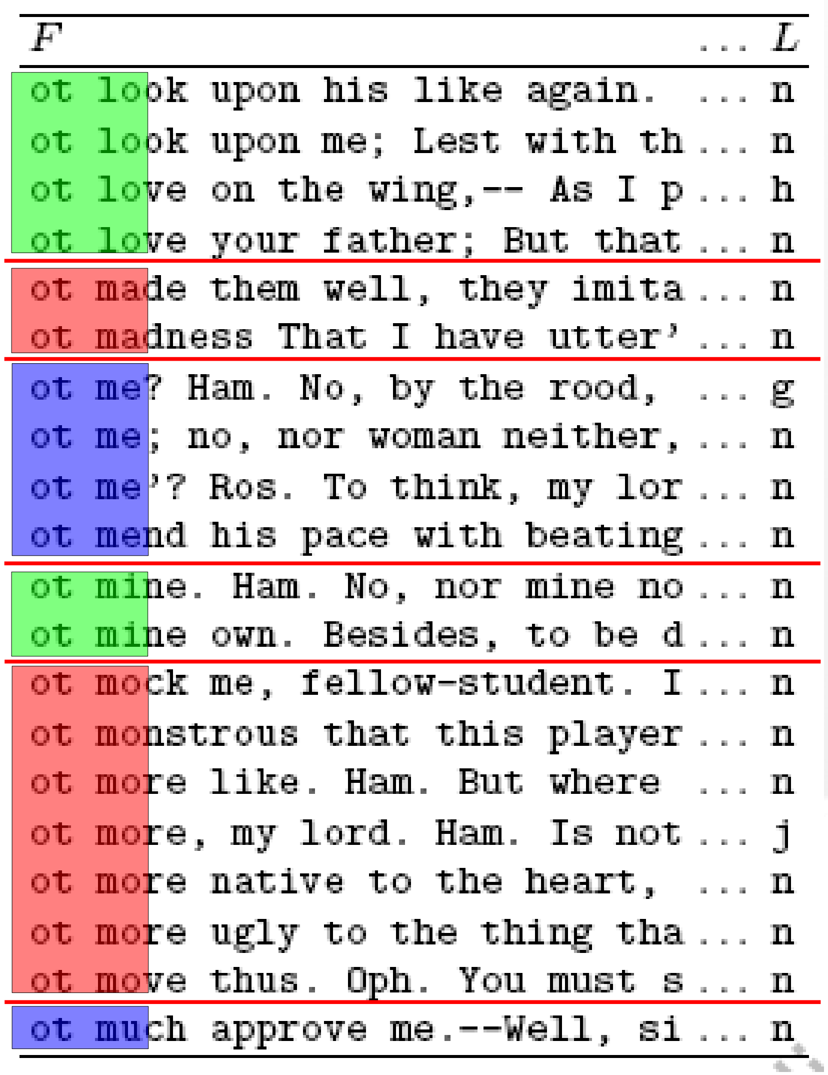
Auxiliary vector C that stores for each symbol c the number of occs in L of symbols smaller than c

- Scan L
  - Set LF[i]=C[L[i]]
  - Update the counter C[L[i]]++

Compress the BWT

Burrows-Wheeler Transform: why it works

[Burrows-Wheeler, 1994]



F	... L
ot look upon his like again.	... n
ot look upon me; Lest with th	... n
ot love on the wing,-- As I p	... h
ot love your father; But that	... n
ot made them well, they imita	... n
ot madness That I have utter'	... n
ot me? Ham. No, by the rood,	... g
ot me; no, nor woman neither,	... n
ot me'? Ros. To think, my lor	... n
ot mend his pace with beating	... n
ot mine. Ham. No, nor mine no	... n
ot mine own. Besides, to be d	... n
ot mock me, fellow-student. I	... n
ot monstrous that this player	... n
ot more like. Ham. But where	... n
ot more, my lord. Ham. Is not	... j
ot more native to the heart,	... n
ot more ugly to the thing tha	... n
ot move thus. Oph. You must s	... n
ot much approve me.--Well, si	... n

Shakespeare's Hamlet

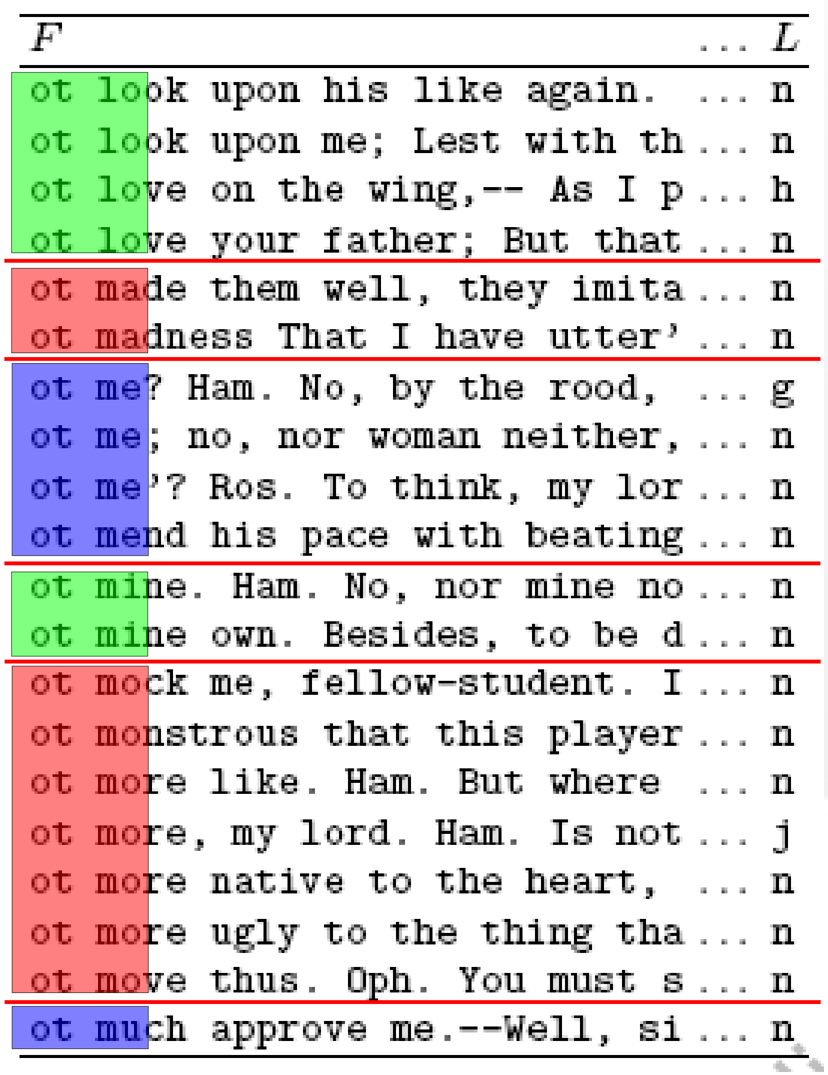
L is locally homogeneous

Symbols followed by equal contexts are clustered togheter

thus, L is highly compressible

Burrows-Wheeler Transform: why it works

[Burrows-Wheeler, 1994]



F	... L
ot look upon his like again.	... n
ot look upon me; Lest with th	... n
ot love on the wing,-- As I p	... h
ot love your father; But that	... n
ot made them well, they imita	... n
ot madness That I have utter’	... n
ot me? Ham. No, by the rood,	... g
ot me; no, nor woman neither,	... n
ot me’? Ros. To think, my lor	... n
ot mend his pace with beating	... n
ot mine. Ham. No, nor mine no	... n
ot mine own. Besides, to be d	... n
ot mock me, fellow-student. I	... n
ot monstrous that this player	... n
ot more like. Ham. But where	... n
ot more, my lord. Ham. Is not	... j
ot more native to the heart,	... n
ot more ugly to the thing tha	... n
ot move thus. Oph. You must s	... n
ot much approve me.--Well, si	... n

Shakespeare's Hamlet

L is locally homogeneous

Symbols followed by equal contexts are clustered togheter

thus, L is highly compressible

B&W suggest to compress L in 3 steps:

- Move-To-Front
- Run Length Encoding
- Statistical Encoder (e.g. Huffman)

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous

L = a r d r c a a a a b b

C =

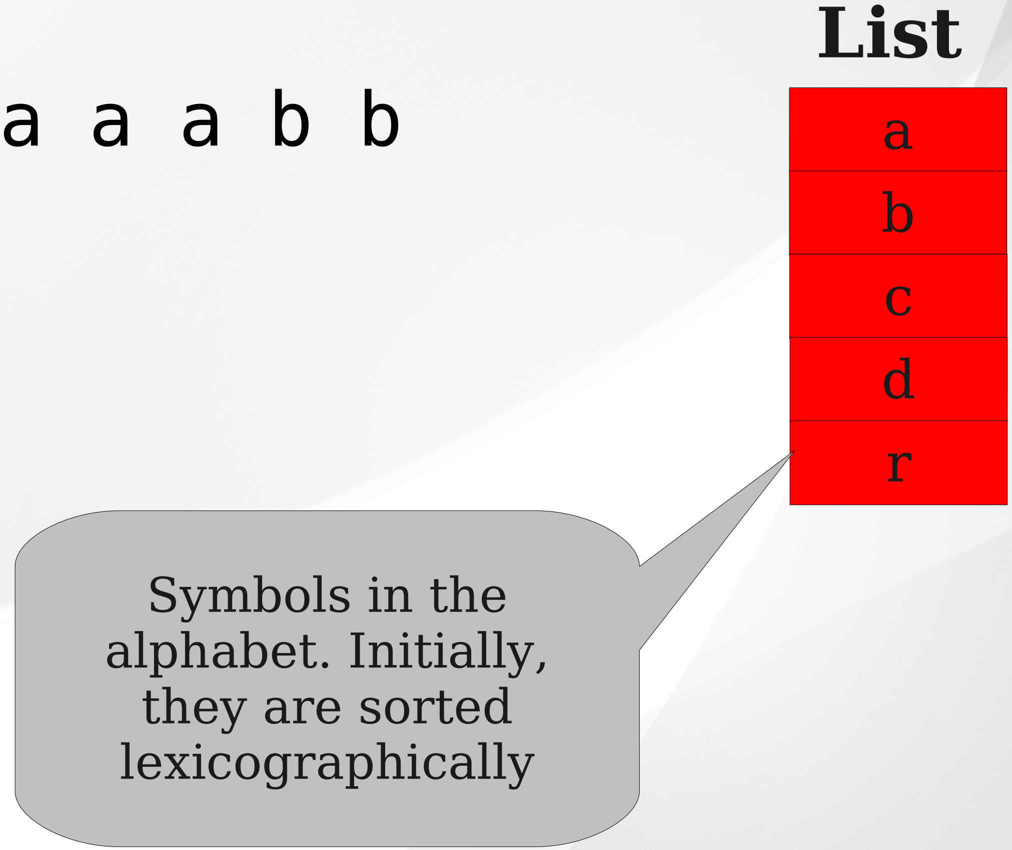
Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous

L = a r d r c a a a a b b

C =



List

a
b
c
d
r

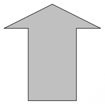
Symbols in the alphabet. Initially, they are sorted lexicographically

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous

L = a r d r c a a a a b b



C =

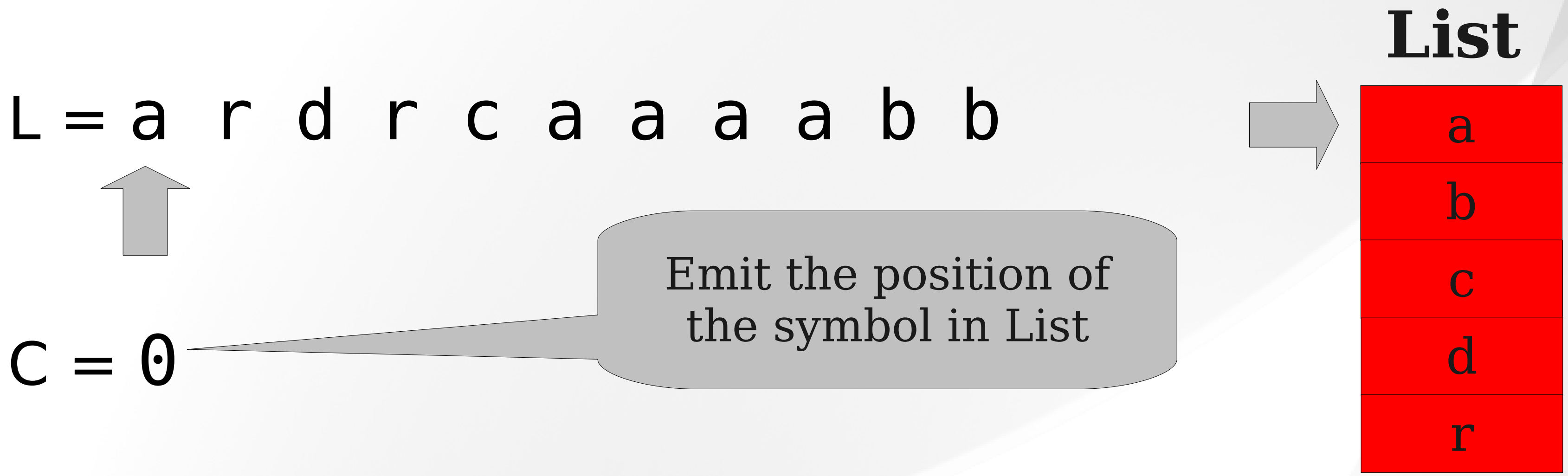
List

a
b
c
d
r

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous



List

L = a r d r c a a a a b b

C = 0

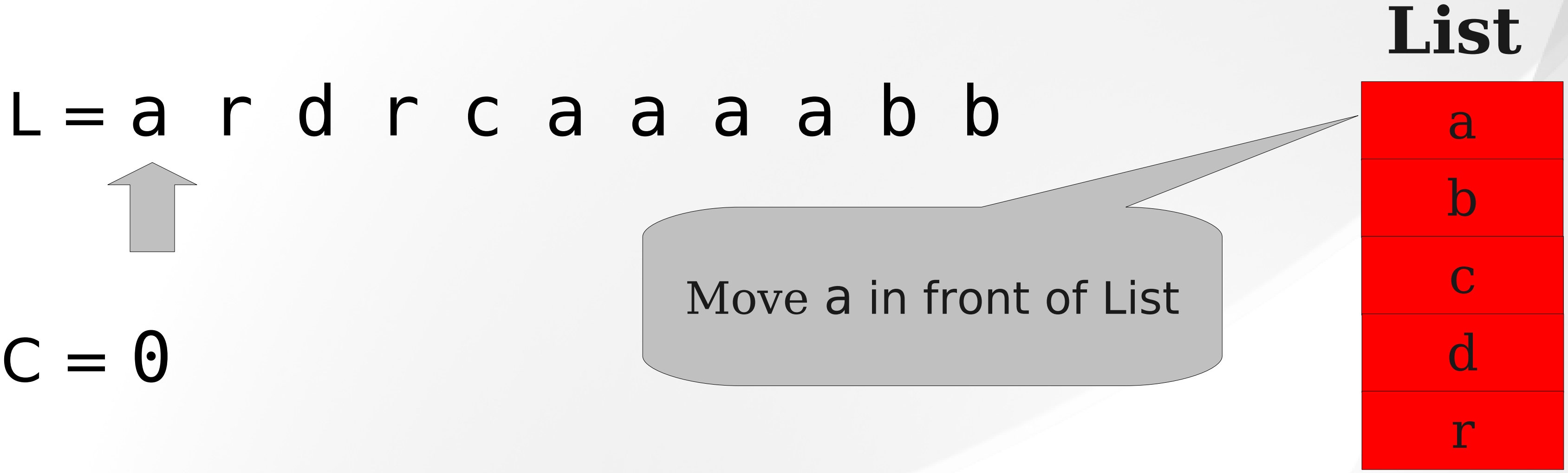
Emit the position of the symbol in List

a
b
c
d
r

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous



List

L = a r d r c a a a a b b

C = 0

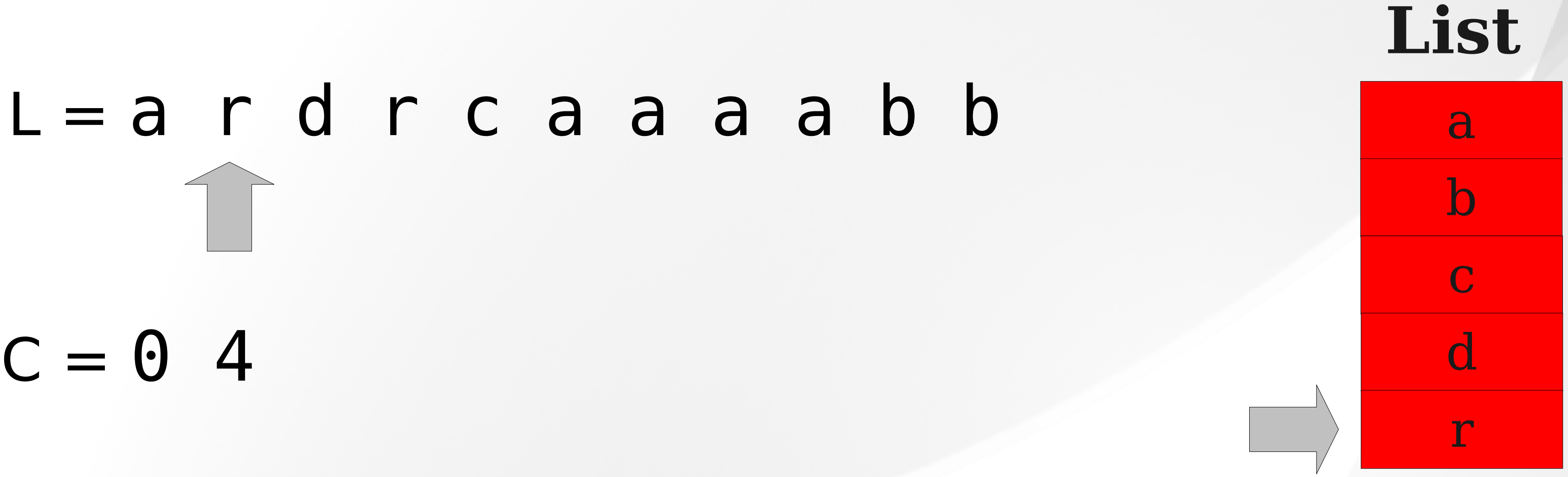
Move a in front of List

a
b
c
d
r

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous



L = a r d r c a a a a b b

C = 0 4

List

a

b

c

d

r

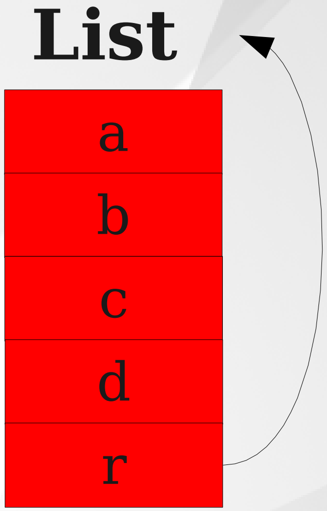
Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous

L = a r d r c a a a a b b

C = 0 4



List

a

b

c

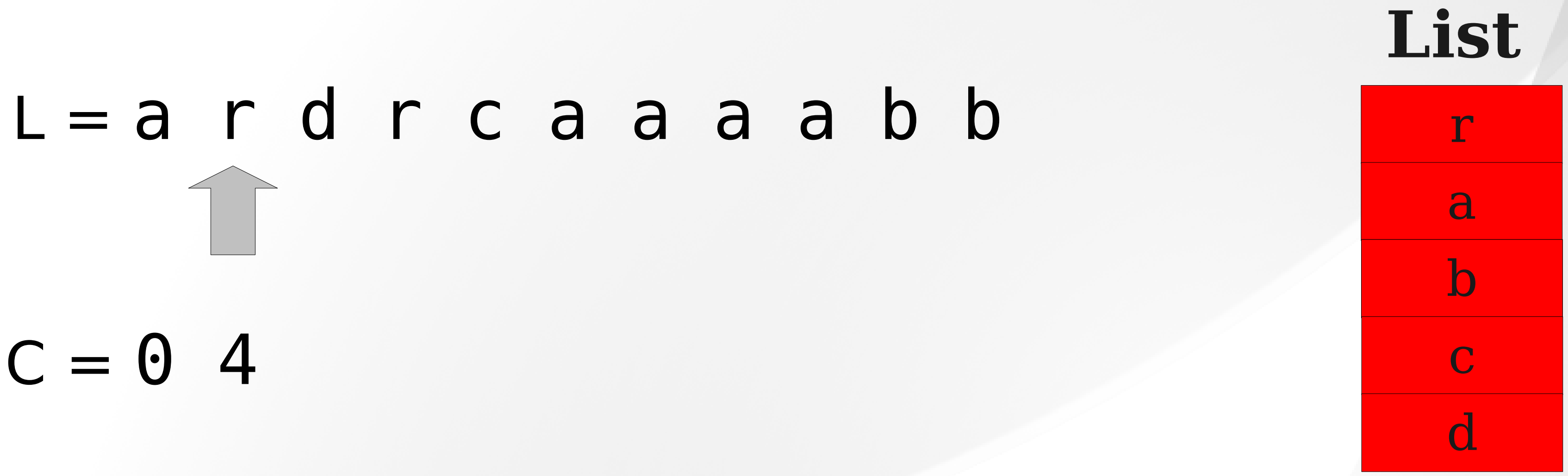
d

r

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous



L = a r d r c a a a a b b

C = 0 4

List

r
a
b
c
d

Move-to-Front

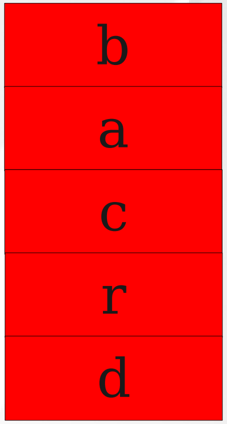
[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous

L = a r d r c a a a a b b

C = 0 4 4 1 4 3 0 0 0 4 0

List



b

a

c

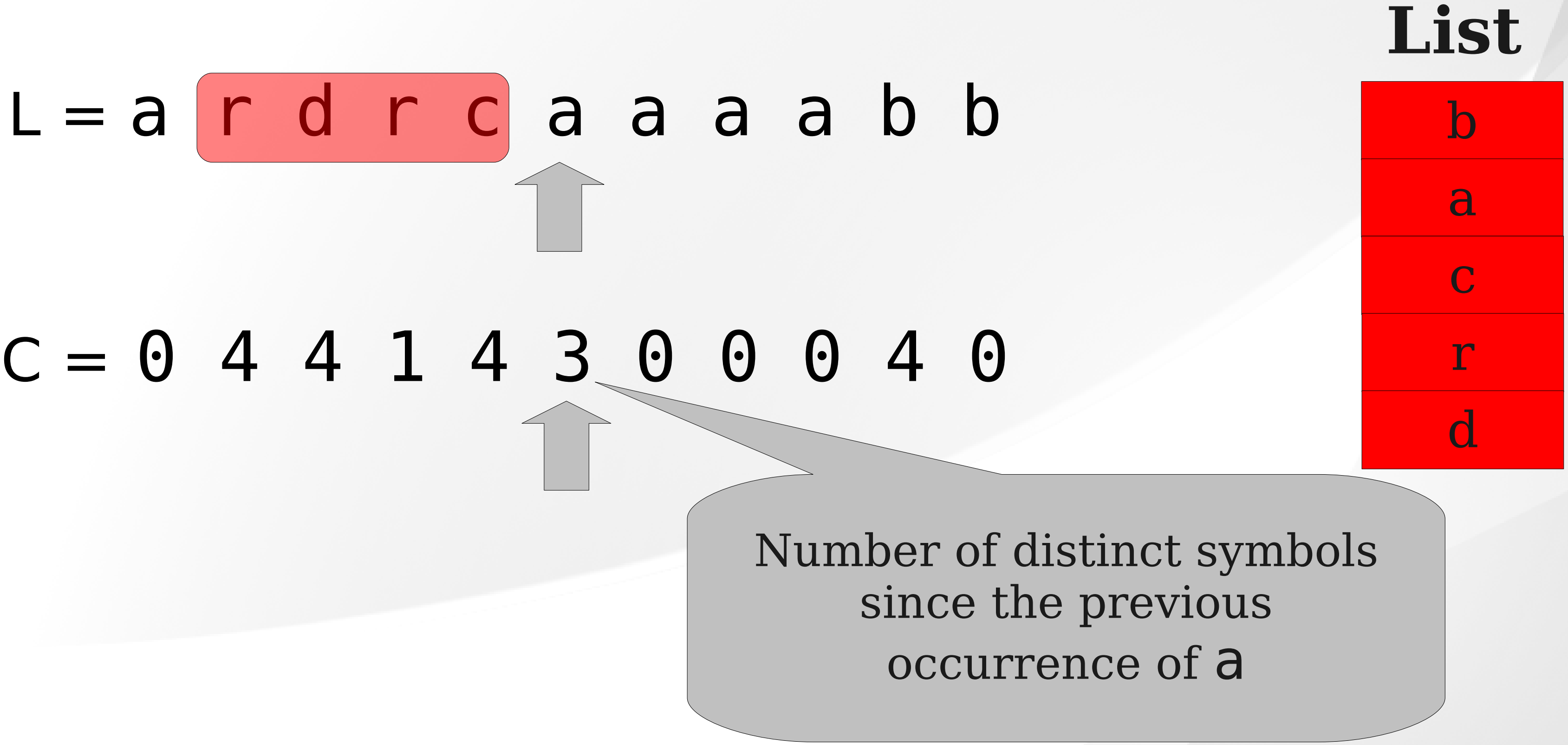
r

d

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous



L = a r d r c a a a a b b

List

b

a

c

r

d

C = 0 4 4 1 4 3 0 0 0 4 0

Number of distinct symbols since the previous occurrence of a

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous

L = a r d r c a a a a b b

C = 0 4 4 1 4 3 0 0 0 4 0

List
b
a
c
r
d

If the string is locally homogeneous,
C contain a lot of small numbers.

Move-to-Front

[Bentley-Sleator-Tarjan-Wei, 1986]

Transform L which is locally homogeneous into a new string that is globally homogeneous

L = a r d r c a a a a b b

C = 0 4 4 1 4 3 0 0 0 4 0

List

b

a

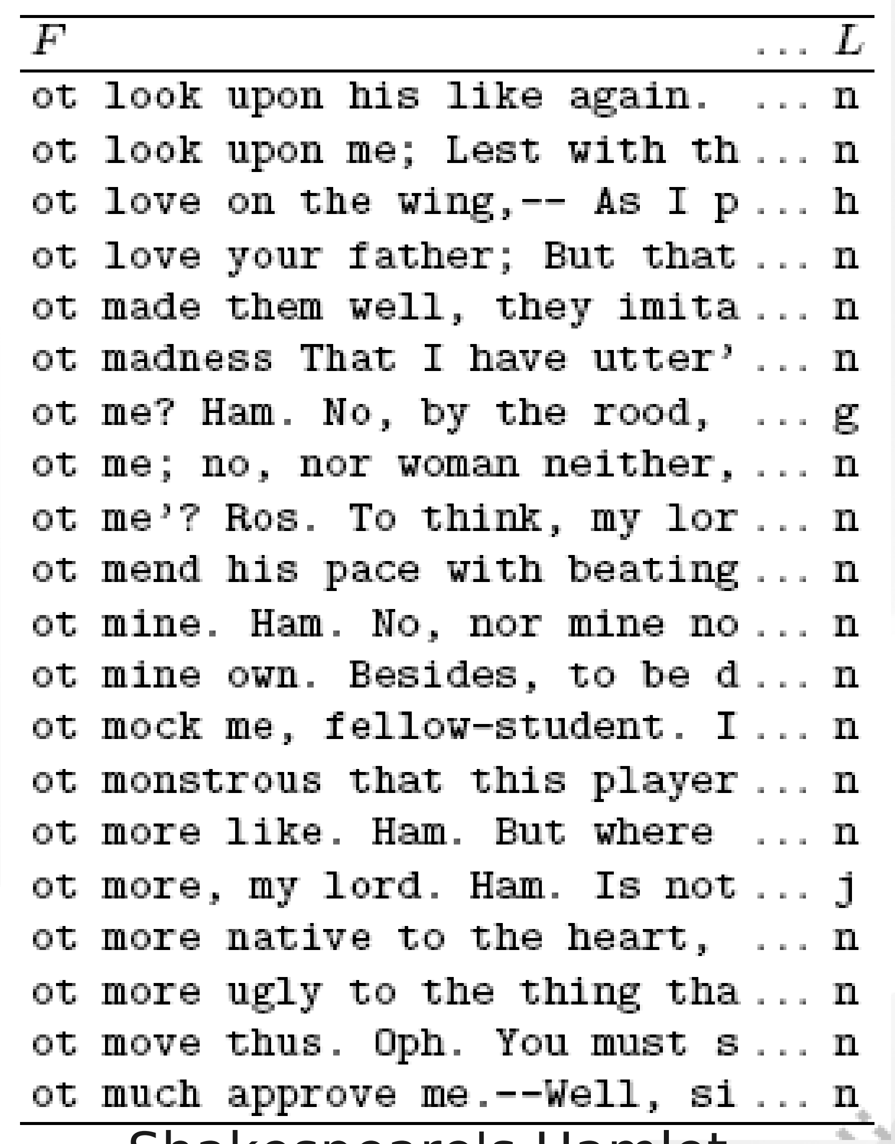
c

r

d

In particular, runs of equal symbols become runs of `0`

Burrows-Wheeler Transform: why Move-to-Front

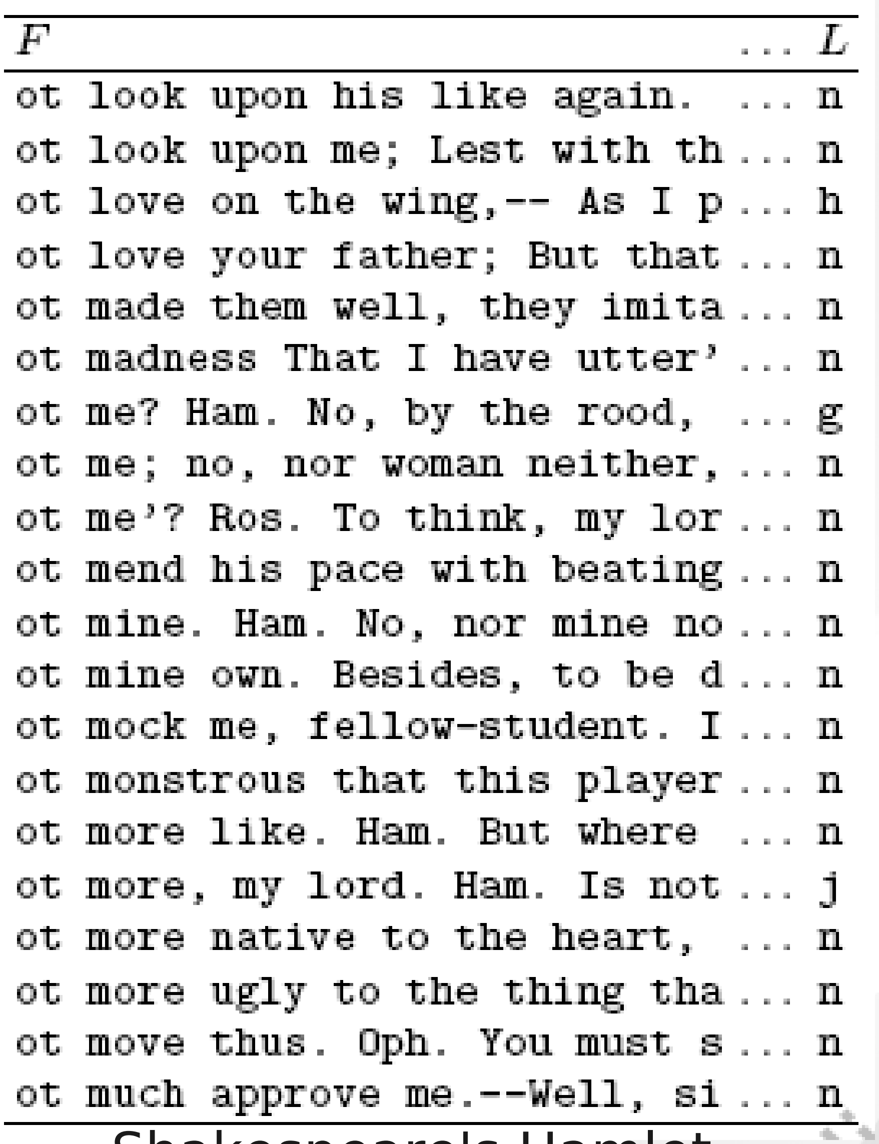


F	...	*L*
ot look upon his like again.	...	n
ot look upon me; Lest with th	...	n
ot love on the wing,-- As I p	...	h
ot love your father; But that	...	n
ot made them well, they imita	...	n
ot madness That I have utter’	...	n
ot me? Ham. No, by the rood,	...	g
ot me; no, nor woman neither,	...	n
ot me’? Ros. To think, my lor	...	n
ot mend his pace with beating	...	n
ot mine. Ham. No, nor mine no	...	n
ot mine own. Besides, to be d	...	n
ot mock me, fellow-student. I	...	n
ot monstrous that this player	...	n
ot more like. Ham. But where	...	n
ot more, my lord. Ham. Is not	...	j
ot more native to the heart,	...	n
ot more ugly to the thing tha	...	n
ot move thus. Oph. You must s	...	n
ot much approve me.--Well, si	...	n

Shakespeare's Hamlet

AGAIN!
]

Burrows-Wheeler Transform: why Move-to-Front



F	. . . *L*	MTF(L)
ot look upon his like again.	. . . n	+
ot look upon me; Lest with th	. . . n	0
ot love on the wing,-- As I p	. . . h	+
ot love your father; But that	. . . n	1
ot made them well, they imita	. . . n	0
ot madness That I have utter’	. . . n	0
ot me? Ham. No, by the rood,	. . . g	+
ot me; no, nor woman neither,	. . . n	1
ot me’? Ros. To think, my lor	. . . n	0
ot mend his pace with beating	. . . n	0
ot mine. Ham. No, nor mine no	. . . n	0
ot mine own. Besides, to be d	. . . n	0
ot mock me, fellow-student. I	. . . n	0
ot monstrous that this player	. . . n	0
ot more like. Ham. But where	. . . n	0
ot more, my lord. Ham. Is not	. . . j	+
ot more native to the heart,	. . . n	1
ot more ugly to the thing tha	. . . n	0
ot move thus. Oph. You must s	. . . n	0
ot much approve me.--Well, si	. . . n	0

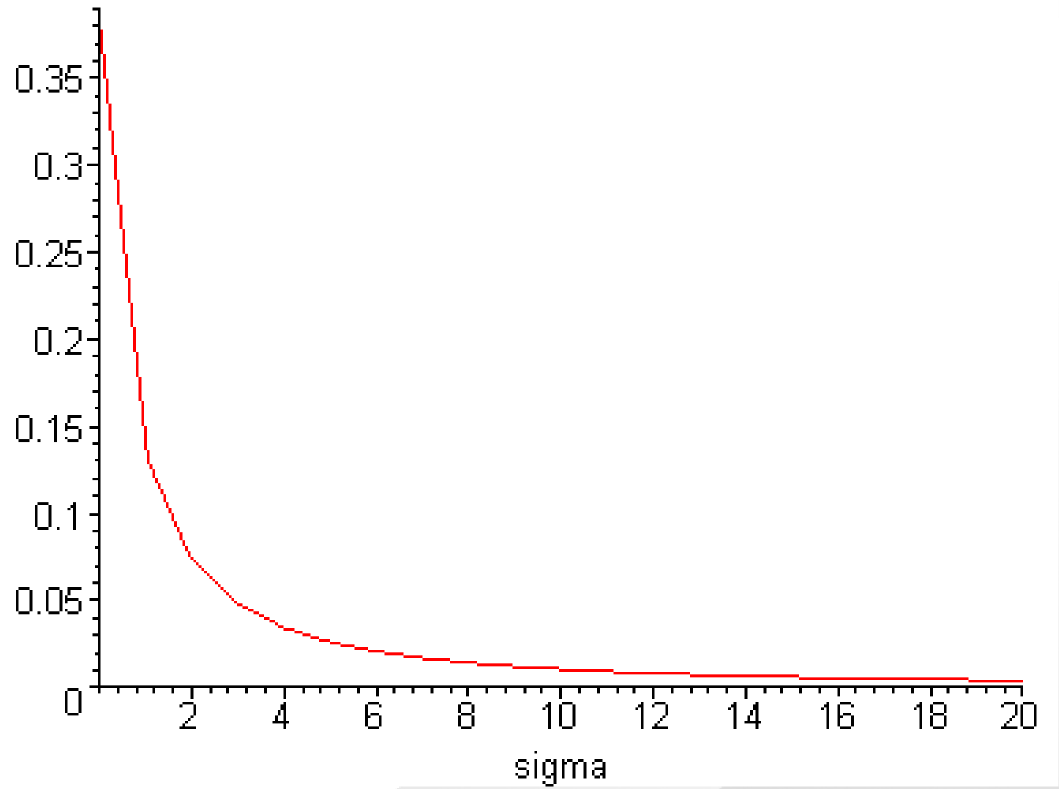
Shakespeare's Hamlet

AGAIN!

After MTF

- Now we have a string with small numbers: lots of 0s, many 1s, …
- Skewed frequencies: next steps RLE +Huffman!

Symbol frequencies



Run-Length-Encoding

Encode runs of 0s in C

```
L = a r d r c a a a a b b

C = 0 4 4 1 4 3 0 0 0 4 0

C'= 0 5 5 2 5 4 0 0 0 5 0
```

Replace symbol c>0 with c+1,
i.e. increase the alphabet size by 1

Run-Length-Encoding

Encode runs of 0s in C

L = a r d r c a a a a b b

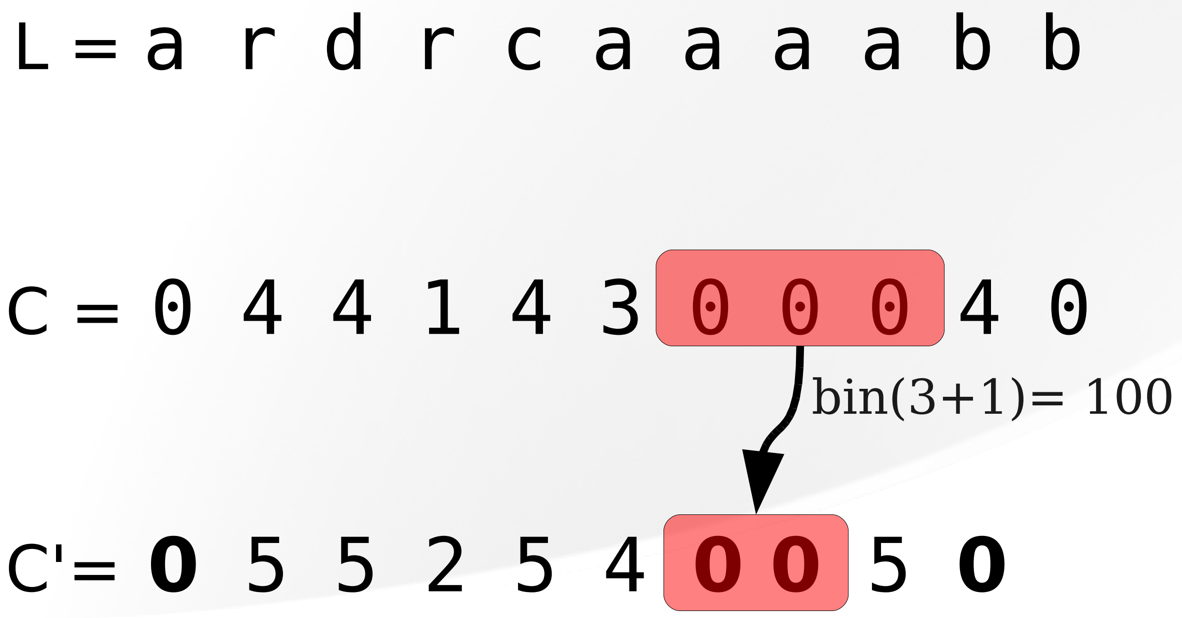
C = 0 4 4 1 4 3 0 0 0 4 0

C'= **0** 5 5 2 5 4 **0 0** 5 **0**

Encode runs of 0s with bin(length+1)
without the most significant bit. (0/1 are unused)

Run-Length-Encoding

Encode runs of 0s in C



L = a r d r c a a a a b b

C = 0 4 4 1 4 3 0 0 0 4 0

bin(3+1)= 100

C'= **0** 5 5 2 5 4 **0 0** 5 **0**

Encode runs of 0s with bin(length+1)
without the most significant bit. (0/1 are unused)

Run-Length-Encoding

Encode runs of 0s in C

L = a r d r c a a a a b b

C = 0 4 4 1 4 3 0 0 0 4 0

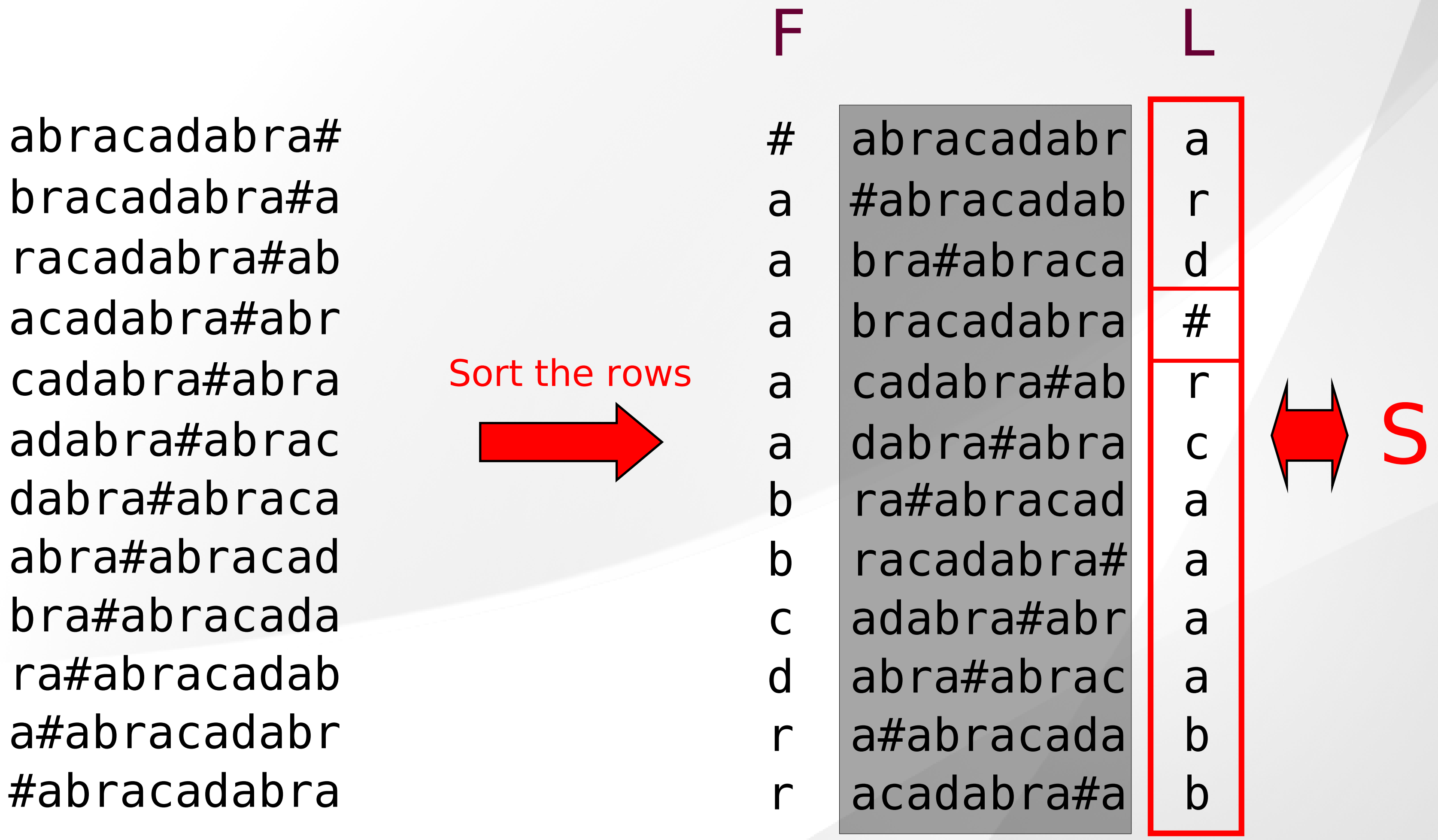
C'= **0** 5 5 2 5 4 **0** **0** 5 **0**

Final step: Huffman to encode C'.
Now it is effective, large number of 0s and 1s

How to build (efficiently) the BWT?

How to build the BWT

Let us given S = abracadabra#



abracadabra#

bracadabra#a

racadabra#ab

acadabra#abr

cadabra#abra

adabra#abrac

dabra#abraca

abra#abracad

bra#abracada

ra#abracadab

a#abracadabr

#abracadabra

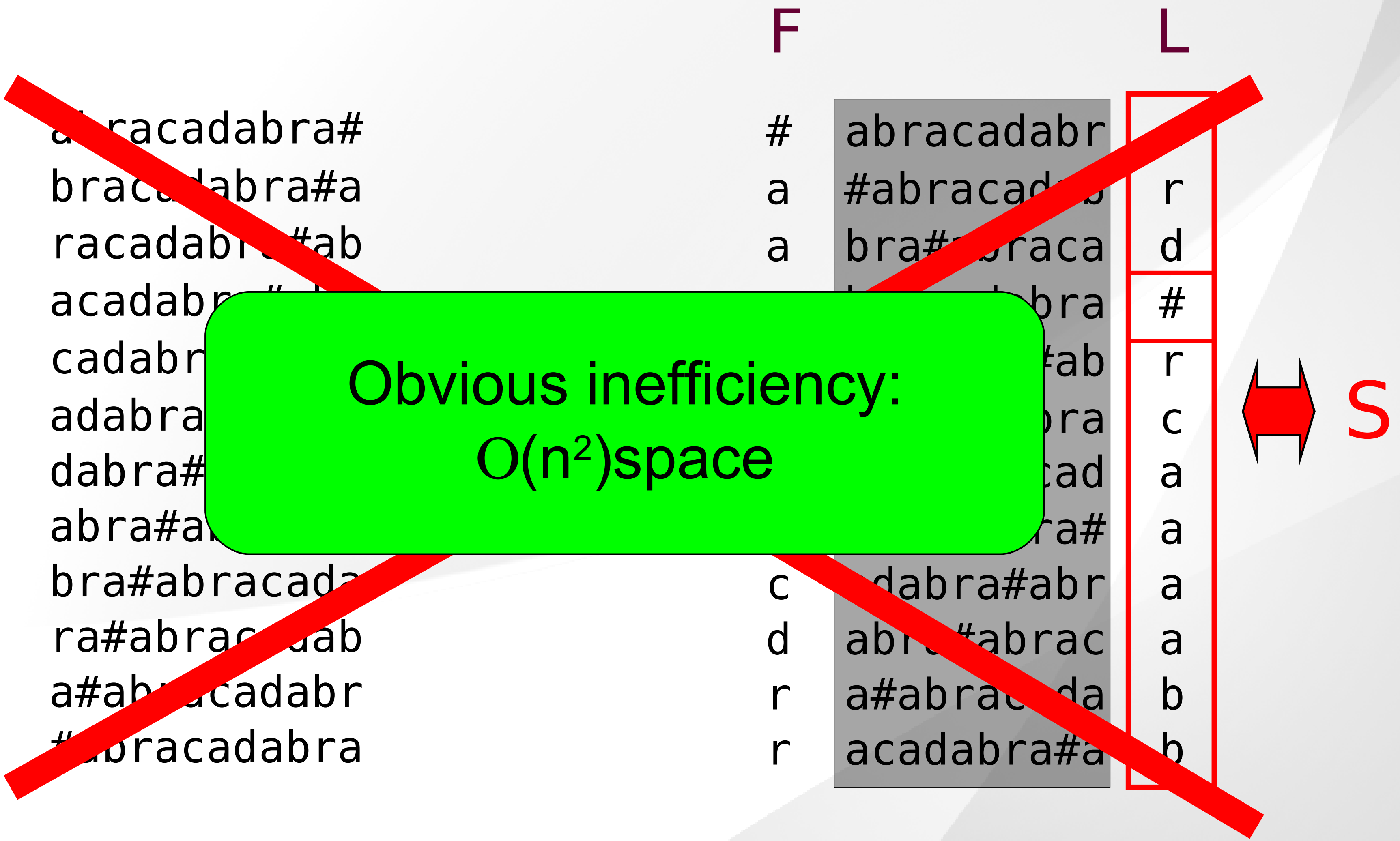
Sort the rows

F		L
#	abracadabr	a
a	#abracadab	r
a	bra#abraca	d
a	bracadabra	#
a	cadabra#ab	r
a	dabra#abra	c
b	ra#abracad	a
b	racadabra#	a
c	adabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

S

How to build the BWT?

Let us given S = abracadabra#



F L

abracadabra#
bracadabra#a
racadabra#ab
acadabr
cadabr
adabra
dabra#
abra#a
bra#abracada
ra#abracadab
a#abracadabr
#abracadabra

F		L
#	abracadabr	
a	#abracadab	r
a	bra#abraca	d
	bra	#
	#ab	r
	bra	c
	cad	a
	ra#	a
c	dabra#abr	a
d	abra#abrac	a
r	a#abracada	b
r	acadabra#a	b

S

Obvious inefficiency:
O(n^2)space

Suffix Array

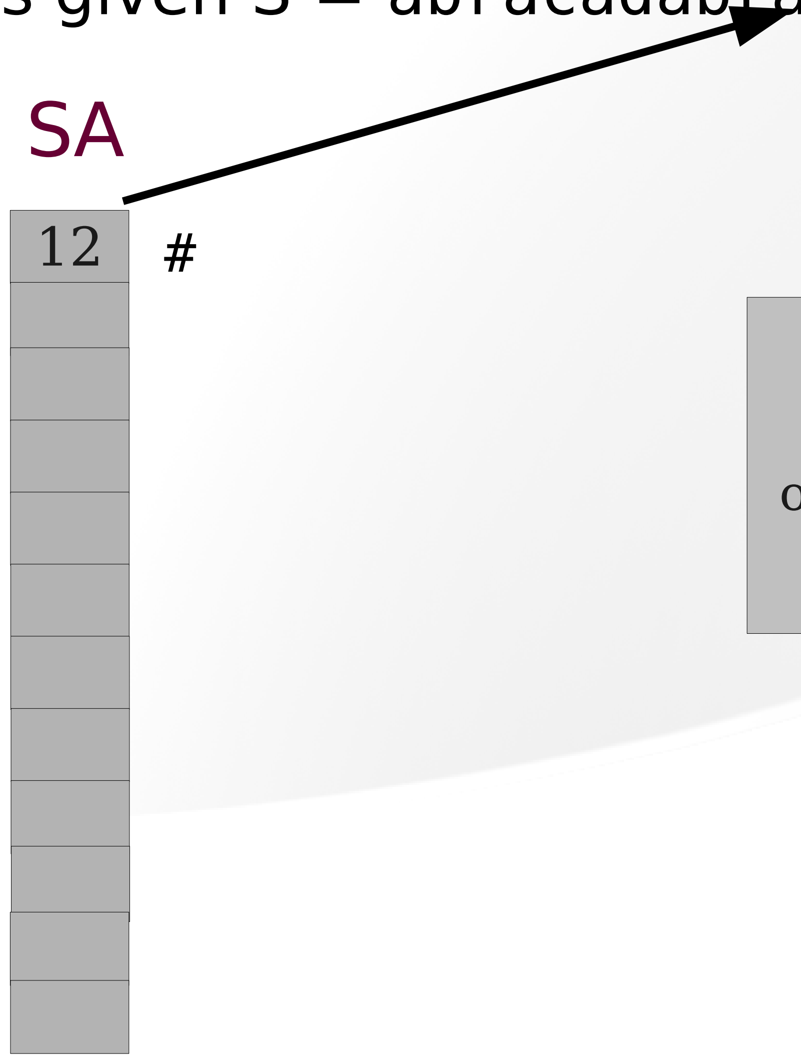
Let us given S = abracadabra#

SA	
12	#

SA stores the starting positions in S of the suffixes sorted lexicographically

Suffix Array

Let us given S = abracadabra#



SA stores the starting positions in S of the suffixes sorted lexicographically

Suffix Array

Let us given S = abracadabra#

SA

12	#
11	a#

SA stores the starting positions in S
of the suffixes sorted lexicographically

Suffix Array

Let us given S = abracadabra#

SA	
12	#
11	a#
8	abra#
1	abracadabra#
4	acadabra#
6	adabra#
9	bra#
2	bracadabra#
5	cadabra#abr
7	dabra#
10	ra#
3	racadabra#

SA stores the starting positions in S
of the suffixes sorted lexicographically

Suffix Array

Let us given S = abracadabra#

SA	
12	#
11	a#
8	abra#
1	abracadabra#
4	acadabra#
6	adabra#
9	bra#
2	bracadabra#
5	cadabra#abr
7	dabra#
10	ra#
3	racadabra#

⇔

F	L
#abracadabr	a
a#abracadab	r
abra#abraca	d
abracadabra	#
acadabra#ab	r
adabra#abra	c
bra#abracad	a
bracadabra#	a
cadabra#abr	a
dabra#abrac	a
ra#abracada	b
racadabra#a	b

Suffix Array

Let us given S = abracadabra#

SA	
12	#
11	a#
8	abra#
1	abracadabra#
4	acadabra#
6	adabra#
9	bra#
2	bracadabra#
5	cadabra#abr
7	dabra#
10	ra#
3	racadabra#

⇔

F	L
#abracadabr	a
a#abracadab	r
abra#abraca	d
abracadabra	#
acadabra#ab	r
adabra#abra	c
bra#abracad	a
bracadabra#	a
cadabra#abr	a
dabra#abrac	a
ra#abracada	b
racadabra#a	b

Suffix Array

Let us given S = abracadabra#

Given SA, we have L[i] = S[SA[i]-1]

SA	
12	#
11	a#
8	abra#
1	abracadabra#
4	acadabra#
6	adabra#
9	bra#
2	bracadabra#
5	cadabra#abr
7	dabra#
10	ra#
3	racadabra#

F	L
#abracadabr	a
racadab	r
abra#abraca	d
abracadabra	#
acadabra#ab	r
adabra#abra	c
bra#abracad	a
bracadabra#	a
cadabra#abr	a
dabra#abrac	a
ra#abracada	b
racadabra#a	b

Suffix Array

Let us given S = abracadabra#

SA		F	L
12	#	#abracadabr	a
11	a#	racadab	r
8	abra#	abra#abraca	d
1	abrac	racadabra	#
4	acada	adabra#ab	r
6	adabr	abra#abra	c
9	bra#	bra#abracad	a
2	bracadabra#	bracadabra#	a
5	cadabra#abr	cadabra#abr	a
7	dabra#	dabra#abrac	a
10	ra#	ra#abracada	b
3	racadabra#	racadabra#a	b

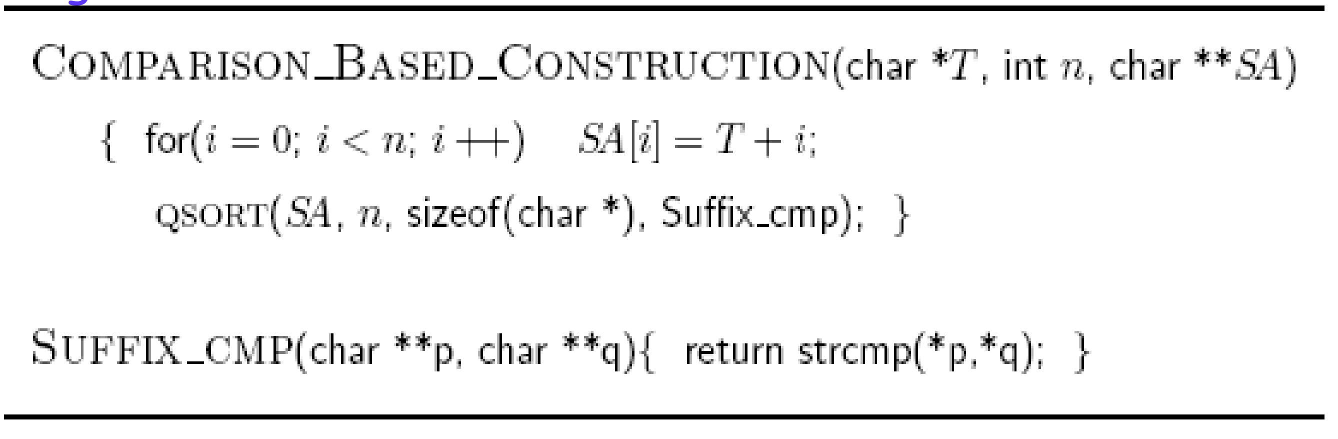
Given SA, we have L[i] = S[SA[i]-1]

However, the Suffix Array has many other applications in pattern matching!

How construct SA?

SA	
12	#
11	a#
8	abra#
1	abracadabra#
4	acadabra#
6	adabra#
9	bra#
2	bracadabra#
5	cadabra#abr
7	dabra#
10	ra#
3	racadabra#

Elegant but inefficient



```
COMPARISON_BASED_CONSTRUCTION(char *$T$, int $n$, char **$SA$)
    { for($i = 0$; $i < n$; $i$++)    $SA[i] = T + i$;
        QSORT($SA$, $n$, sizeof(char *), Suffix_cmp);  }

SUFFIX_CMP(char **p, char **q){  return strcmp(*p,*q);  }
```

- $\Theta(n^2 \log n)$ time in the worst-case
- O(n) space

Suffix Array

Last years, many clever algorithms that compute the Suffix Array in linear time:

- Karkkainen-Sanders [J. ACM, 2006]
- Ko-Aluru [J. Discrete Algorithms, 2005

Suffix Array

Last years, many clever algorithms that compute the Suffix Array in linear time:

- Karkkainen-Sanders [J. ACM, 2006]
- Ko-Aluru [J. Discrete Algorithms, 2005

and practical ones (no linear time):

- Manzini-Ferragina [Algorithmica, 2004]
- Maniscalco-Puglisi [ACM JEA, 2006]

Suffix Array

Last years, many clever algorithms that compute the Suffix Array in linear time:

- Karkkai
- Ko-Alur

and pract :

BWT can be computed
in linear time!!!

Wheeler discovered
the BWT in 1983
but he did not publish it because
it was considered too slow!

- Manzini-Ferragina [Algorithmica, 2004]
- Maniscalco-Puglisi [ACM JEA, 2006]

Suffix Array

Last years, many clever algorithms that compute the Suffix Array in linear time:

- Karkkai
- Ko-Aluru

and pract :

- Manzini-Ferragina [Algorithmica, 2004]
- Maniscalco-Puglisi [ACM JEA, 2006]

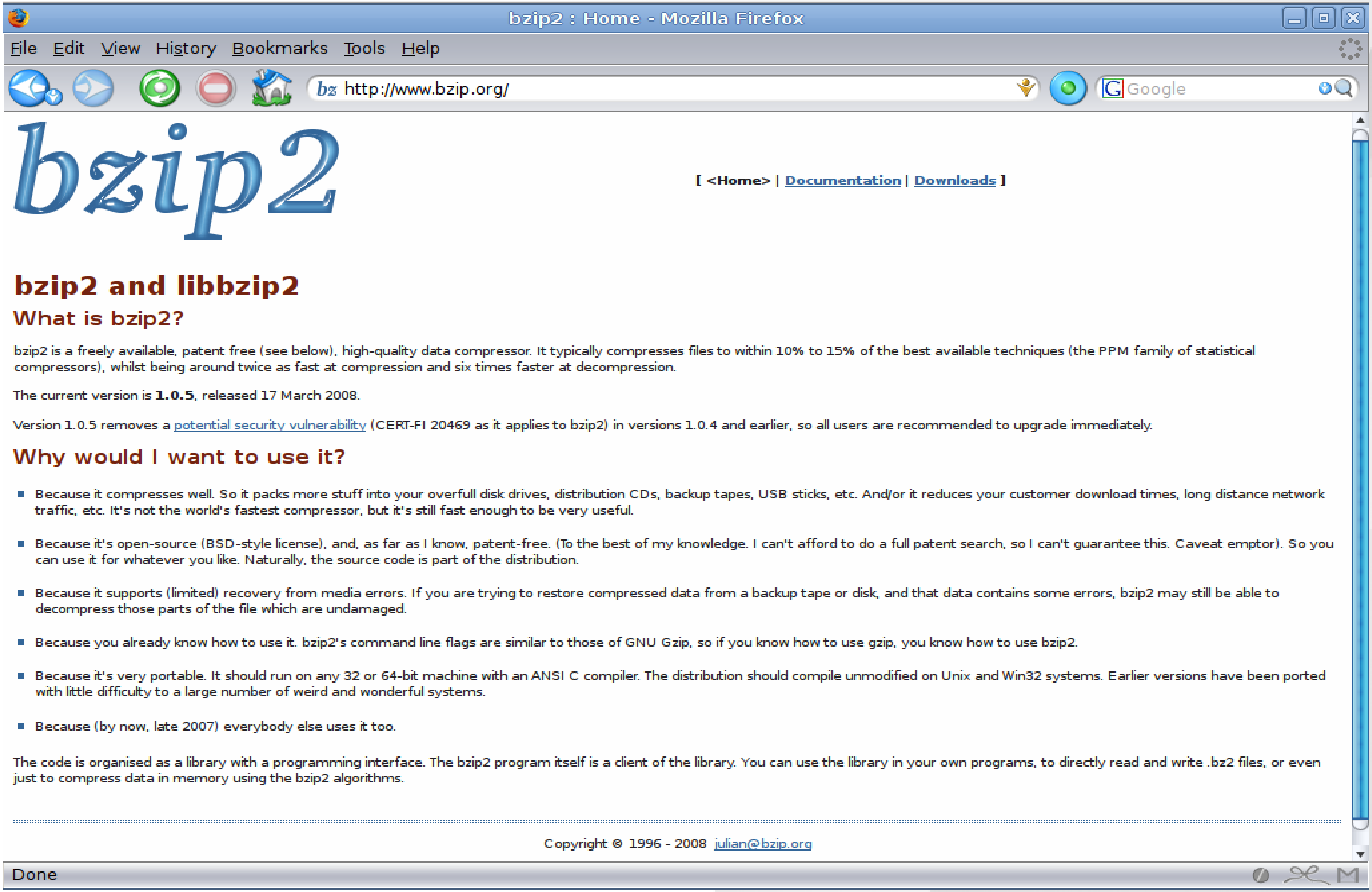
BWT can be computed
in linear time!!!

Wheeler discovered
the BWT in 1983
but he did not publish it because
it was considered too slow!

In 1994, the
paper was
rejected... and
BWT is still a
Technical Report.

A compressor based on BWT: Bzip2

You find this in your Linux distribution



bzip2 : Home - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.bzip.org/
Google
bzip2
Done

[<Home> | Documentation | Downloads]

bzip2 and libbzip2

What is bzip2?

bzip2 is a freely available, patent free (see below), high-quality data compressor. It typically compresses files to within 10% to 15% of the best available techniques (the PPM family of statistical compressors), whilst being around twice as fast at compression and six times faster at decompression.

The current version is **1.0.5**, released 17 March 2008.

Version 1.0.5 removes a potential security vulnerability (CERT-FI 20469 as it applies to bzip2) in versions 1.0.4 and earlier, so all users are recommended to upgrade immediately.

Why would I want to use it?

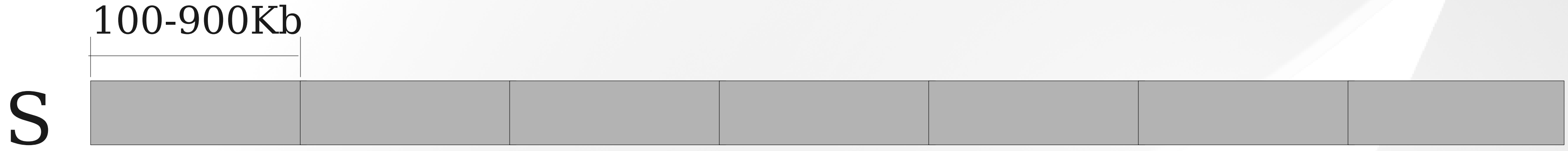
- Because it compresses well. So it packs more stuff into your overfull disk drives, distribution CDs, backup tapes, USB sticks, etc. And/or it reduces your customer download times, long distance network traffic, etc. It's not the world's fastest compressor, but it's still fast enough to be very useful.
- Because it's open-source (BSD-style license), and, as far as I know, patent-free. (To the best of my knowledge. I can't afford to do a full patent search, so I can't guarantee this. Caveat emptor). So you can use it for whatever you like. Naturally, the source code is part of the distribution.
- Because it supports (limited) recovery from media errors. If you are trying to restore compressed data from a backup tape or disk, and that data contains some errors, bzip2 may still be able to decompress those parts of the file which are undamaged.
- Because you already know how to use it. bzip2's command line flags are similar to those of GNU Gzip, so if you know how to use gzip, you know how to use bzip2.
- Because it's very portable. It should run on any 32 or 64-bit machine with an ANSI C compiler. The distribution should compile unmodified on Unix and Win32 systems. Earlier versions have been ported with little difficulty to a large number of weird and wonderful systems.
- Because (by now, late 2007) everybody else uses it too.

The code is organised as a library with a programming interface. The bzip2 program itself is a client of the library. You can use the library in your own programs, to directly read and write .bz2 files, or even just to compress data in memory using the bzip2 algorithms.

Copyright © 1996 - 2008 julian@bzip.org

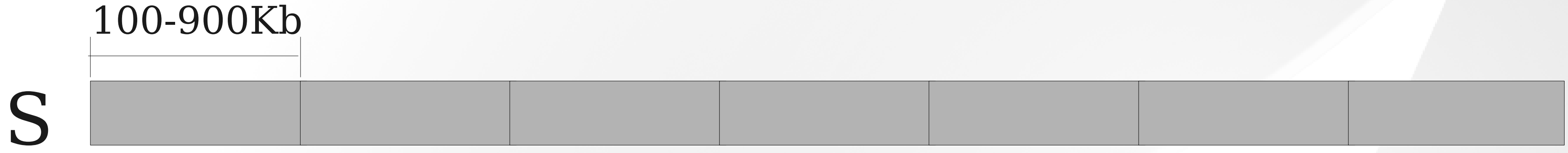
A real implementation: bzip2

It doesn't build the BWT of the whole text but uses blocking approach.



A real implementation: bzip2

It doesn't build the BWT of the whole text but uses blocking approach.



Just for time performance, the compression it achieves is worse.

bzip2 vs gzip

English 5Mb	comp. size	comp. time
gzip	2.0 Mb	1.7 secs
bzip2	1.5 Mb	2.2 secs
bigbzip	1.3 Mb	2.4 secs

bzip2 vs gzip

English 5Mb	comp. size	comp. time
gzip	2.0 Mb	1.7 secs
bzip2	1.5 Mb	2.2 secs
bigbzip	1.3 Mb	2.4 secs

English 20Mb	comp. size	comp. time	dec. time
gzip	7.8 Mb	7.2 secs	0.8 secs
bzip2	5.9 Mb	11.0 secs	4.0 secs
bigbzip	4.1 Mb	20.0 secs	5.8 secs

bzip2 vs gzip

English 5Mb	comp. size	comp. time
gzip	2.0 Mb	1.7 secs
bzip2	1.5 Mb	2.2 secs
bigbzip	1.3 Mb	2.4 secs

English 20Mb			
gzip			
bzip2	5.9 Mb	11.0 secs	4.0 secs
bigbzip	4.1 Mb	20.0 secs	5.8 secs

... but, now we can increase the block size from 900Kb to 5Mb due to new algorithmic solutions.
Improve compression, same time performace!

Some other applications

- ***FM-index/CSA:*** *Searching in compressed text.*
  - Given a text S, we can compress in a way that permits us to search any pattern P in S in time proportional to |P|!
    (Notice that |P|<<|S|)

Some other applications

- **FM-index/CSA***:* *Searching in compressed text.*
  - Given a text S, we can compress in a way that permits us to search any pattern P in S in time proportional to |P|!
    (Notice that |P|<<|S|)
- ***Booster:*** *Best theoretical class of compressors.*
  - Given an 0-th order compressor C, we can optimally partition BWT(S) in O(|S|) time in order to achieve the best compression with C.

Some other applications

- **FM-index/CSA:** *Searching in compressed text.*
  - Given a text S, we can compress in a way that permits us to search any pattern P in S in time proportional to |P|!
    (Notice that |P|<<|S|)
- ***Booster:*** *Best theoretical class of compressors.*
  - Given an 0-th order compressor C, we can optimally partition BWT(S) in O(|S|) time in order to achieve the best compression with C.
- Idea used for other data types.
  - e.g. XML. Compression of labeled trees

Some other applications

- **FM-index/CSA:** *Searching in compressed text.*
  - Given a text S, we can compress in a way that permits us to search any pattern P in S in time proportional to |P|!

These stuff require another seminar...

  - Given an 0-th order compressor C, we can optimally partition BWT(S) in O(|S|) time in order to achieve the best compression with C.
- Idea used for other data types.
  - e.g. XML. Compression of labeled trees

Some other applications

- ***FM-index/CSA:*** *Searching in compressed text.*
  - Given a text S, we can compress in a way that permits us to search any pattern P in S in time proportional to |P|!

These stuff require another seminar...

This is a threat!

-
  - Given an 0-th order compressor C, we can optimally partition BWT(S) in O(|S|) time in order to achieve the best compression with C.
- Idea used for other data types.
  - e.g. XML. Compression of labeled trees

Wake Up!!