Introduction to the Burrows-Wheeler Transform

Giovanni Manzini

1978 (?)

Space Invaders released

- David Wheeler conceives a data compression algorithm based on reversible transformation on the input text, but considers it too slow for practical use

1994

Mandela first black South Africa president

- Mike Burrows improves the speed of the compressor. B&W co-author the technical report describing a “block sorting” lossless data compression algorithm.
- The algorithm splits the input in blocks and computes a reversible transformation that makes the text “more compressible”
- The transformation has been later called the Burrows-Wheeler transform.

1994

May 10, 1994

SRC Research Report 124

A Block-sorting Lossless Data Compression Algorithm

M. Burrows and D.J. Wheeler

digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

The BWT

```
swiss·miss·missing
```

The BWT

swiss·miss·missing

Consider all rotations of the input text

```
s wiss·miss·missin g
w iss·miss·missing s
i ss·miss·missings w
s s·miss·missingsw i
s ·miss·missingswi s
· miss·missingswis s
m iss·missingswiss ·
i ss·missingswiss· m
s s·missingswiss·m i
s ·missingswiss·mi s
· missingswiss·mis s
m issingswiss·miss ·
i ssingswiss·miss· m
s singswiss·miss·m i
s ingswiss·miss·mi s
i ngswiss·miss·mis s
n gswiss·miss·miss i
g swiss·miss·missi n
```

The BWT

```
swiss·miss·missing
```

Consider all rotations of the input text

Sort them in lexicographic order

```
· miss·missingswis s
· missingswiss·mis s
g swiss·miss·missi n
i ngswiss·miss·mis s
i ss·miss·missings w
i ss·missingswiss· m
i ssingswiss·miss· m
m iss·missingswiss ·
m issingswiss·miss ·
n gswiss·miss·miss i
s ·miss·missingswi s
s ·missingswiss·mi s
s ingswiss·miss·mi s
s s·miss·missingsw i
s s·missingswiss·m i
s singswiss·miss·m i
s wiss·miss·missin g
w iss·miss·missing s
```

The BWT

swiss·miss·missing

Consider all rotations of the input text

Sort them in lexicographic order

Take the last character of each rotation

ssnswmm··isssiiigs

		L
·	miss·missingswis	s
·	missingswiss·mis	s
g	swiss·miss·missi	n
i	ngswiss·miss·mis	s
i	ss·miss·missings	w
i	ss·missingswiss·	m
i	ssingswiss·miss·	m
m	iss·missingswiss	·
m	issingswiss·miss	·
n	gswiss·miss·miss	i
s	·miss·missingswi	s
s	·missingswiss·mi	s
s	ingswiss·miss·mi	s
s	s·miss·missingsw	i
s	s·missingswiss·m	i
s	singswiss·miss·m	i
s	wiss·miss·missin	g
w	iss·miss·missing	s

The BWT

swiss·miss·missing

Consider all rotations of the input text

Sort them in lexicographic order

Take the last character of each rotation

ssnswmm··isssiiigs

F		L
·	miss·missingswis	s
·	missingswiss·mis	s
g	swiss·miss·missi	n
i	ngswiss·miss·mis	s
i	ss·miss·missings	w
i	ss·missingswiss·	m
i	ssingswiss·miss·	m
m	iss·missingswiss	·
m	issingswiss·miss	·
n	gswiss·miss·miss	i
s	·miss·missingswi	s
s	·missingswiss·mi	s
s	ingswiss·miss·mi	s
s	s·miss·missingsw	i
s	s·missingswiss·m	i
s	singswiss·miss·m	i
s	wiss·miss·missin	g
w	iss·miss·missing	s

Things to do next

1. Prove that given the transformed text we can retrieve T

2. Show that the transformed text is easy to compress

Fundamental observation

Every column of the matrix is a permutation of the input text (try to prove it!)

swiss·miss·missing

in F s is above s because
ing ≤ **wiss···**

in L s is in the row prefixed
by **ing** hence is above s

F		L
·	miss·missingswis	s
·	missingswiss·mis	s
g	swiss·miss·missi	n
i	ngswiss·miss·mis	s
i	ss·miss·missings	w
i	ss·missingswiss·	m
i	ssingswiss·miss·	m
m	iss·missingswiss	·
m	issingswiss·miss	·
n	gswiss·miss·miss	i
s	·miss·missingswi	s
s	·missingswiss·mi	s
s	ingswiss·miss·mi	s
s	s·miss·missingsw	i
s	s·missingswiss·m	i
s	singswiss·miss·m	i
s	wiss·miss·missin	g
w	iss·miss·missing	s

We can map each character in L to its image in F

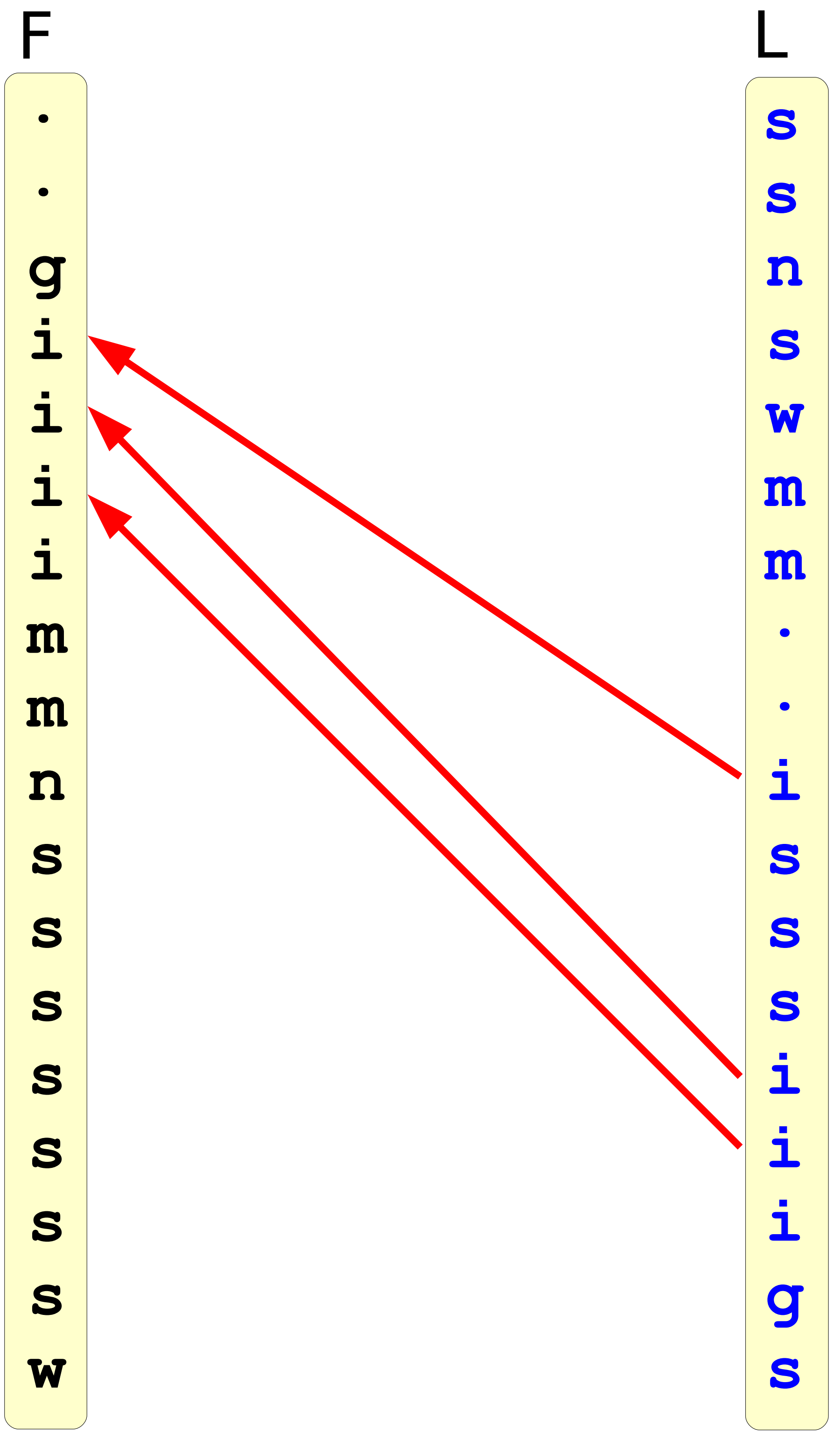
F		L
·	miss·missingswis	s
·	missingswiss·mis	s
g	swiss·miss·missi	n
i	ngswiss·miss·mis	s
i	ss·miss·missings	w
i	ss·missingswiss·	m
i	ssingswiss·miss·	m
m	iss·missingswiss	·
m	issingswiss·miss	·
n	gswiss·miss·miss	i
s	·miss·missingswi	s
s	·missingswiss·mi	s
s	ingswiss·miss·mi	s
s	s·miss·missingsw	i
s	s·missingswiss·m	i
s	singswiss·miss·m	i
s	wiss·miss·missin	g
w	iss·miss·missing	s

We can map each character in L to its image in F

F		L
·	miss·missingswis	s
·	missingswiss·mis	s
g	swiss·miss·missi	n
i	ngswiss·miss·mis	s
i	ss·miss·missings	w
i	ss·missingswiss·	m
i	ssingswiss·miss·	m
m	iss·missingswiss	·
m	issingswiss·miss	·
n	gswiss·miss·miss	i
s	·miss·missingswi	s
s	·missingswiss·mi	s
s	ingswiss·miss·mi	s
s	s·miss·missingsw	i
s	s·missingswiss·m	i
s	singswiss·miss·m	i
s	wiss·miss·missin	g
w	iss·miss·missing	s

We can map each character in L to its image in F

... even if we only have L and F ...



Summing up

From L (the BWT) we can recover F

The relative order of the occurrences of any given character in F and L is the same

We can easily build a map telling us where each character in L is in F

We call this the LF map.

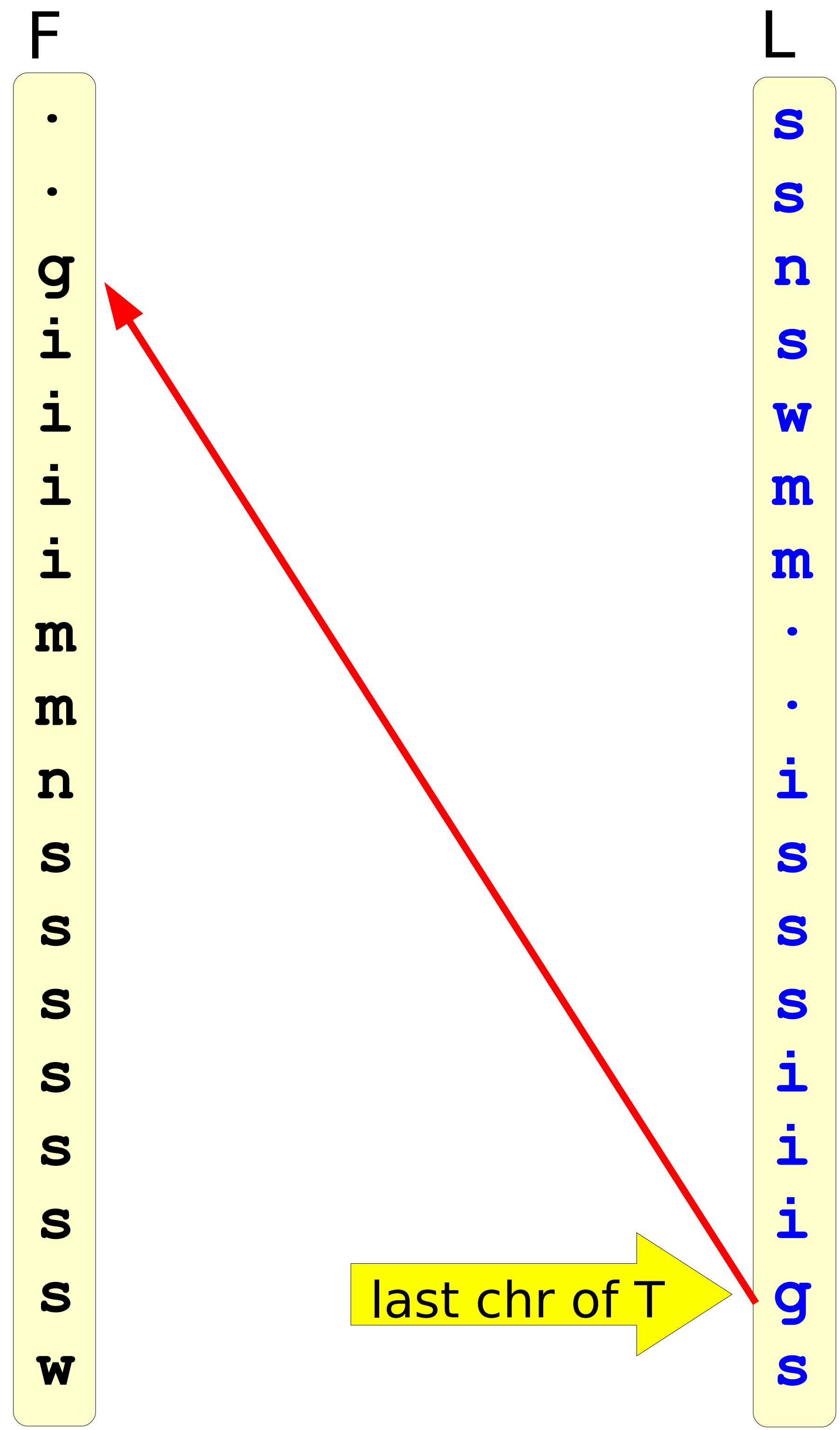
Using the LF map we retrieve T right-to-left

T = g

F		L
.		s
.		s
g		n
i		s
i		w
i		m
i		m
m		.
m		.
n		i
s		s
s		s
s		s
s		i
s		i
s		i
s	last chr of T	g
s		s
w		

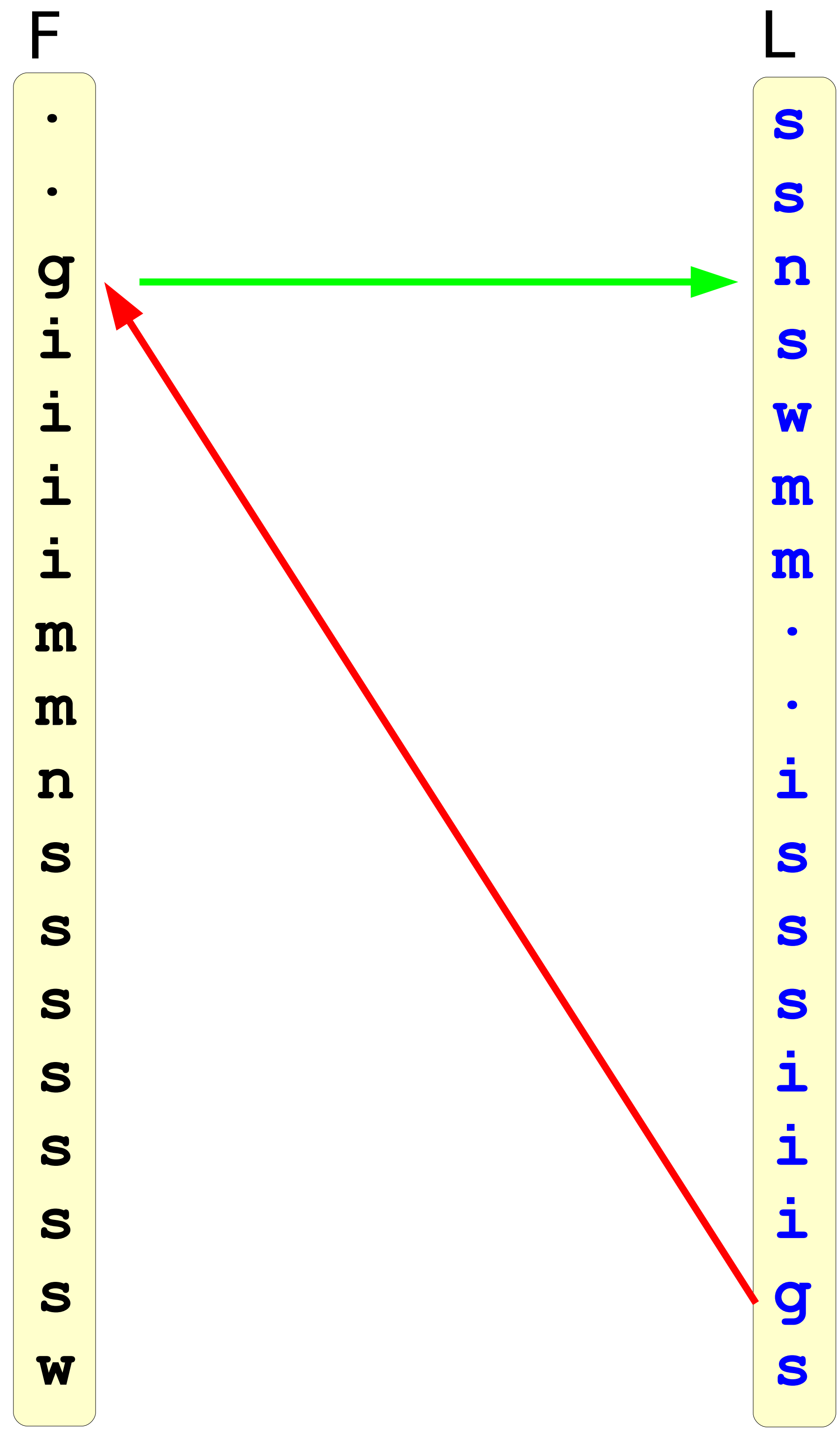
Using the LF map we retrieve T right-to-left

T = g



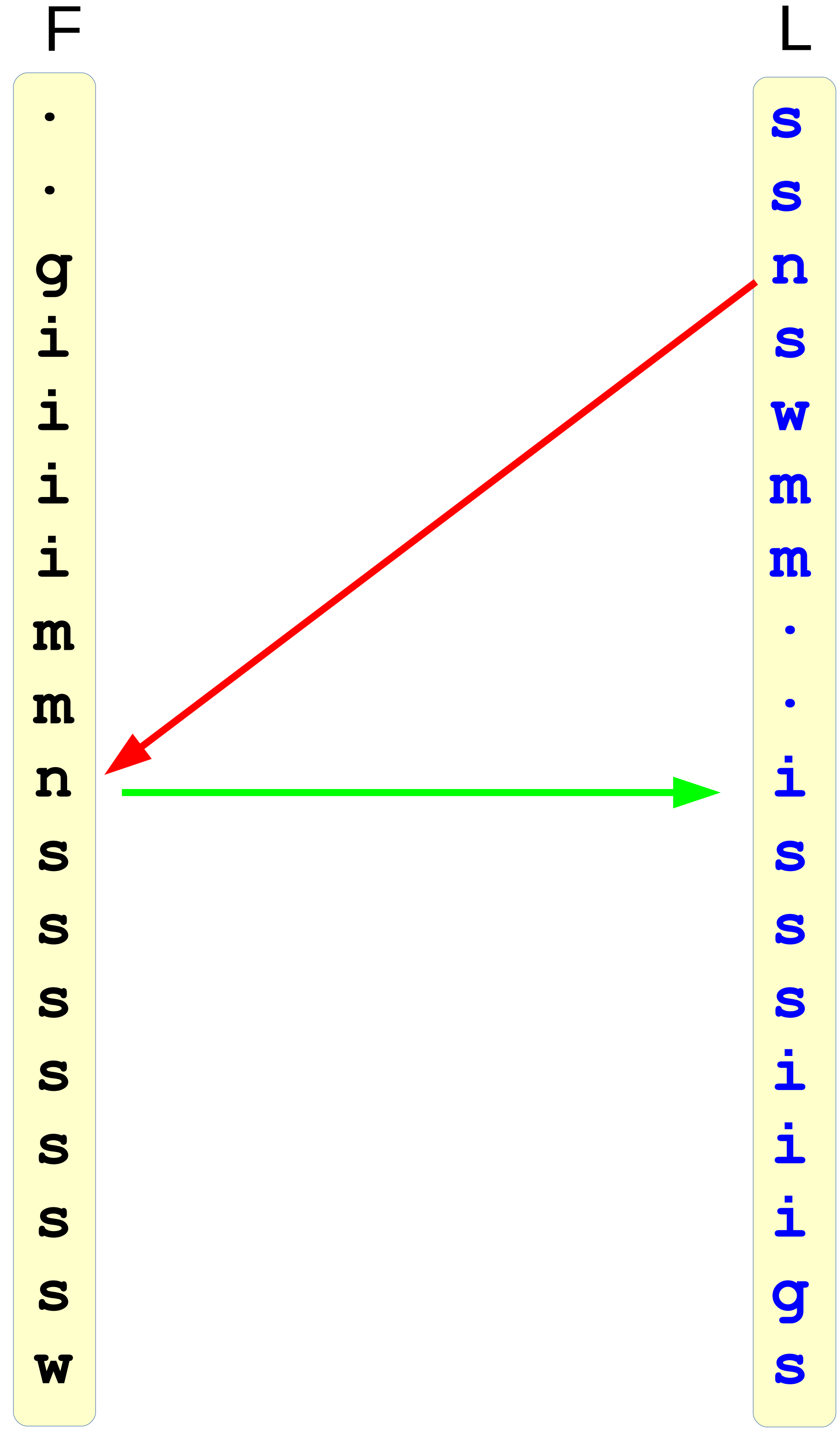
Using the LF map we retrieve T right-to-left

T = ng



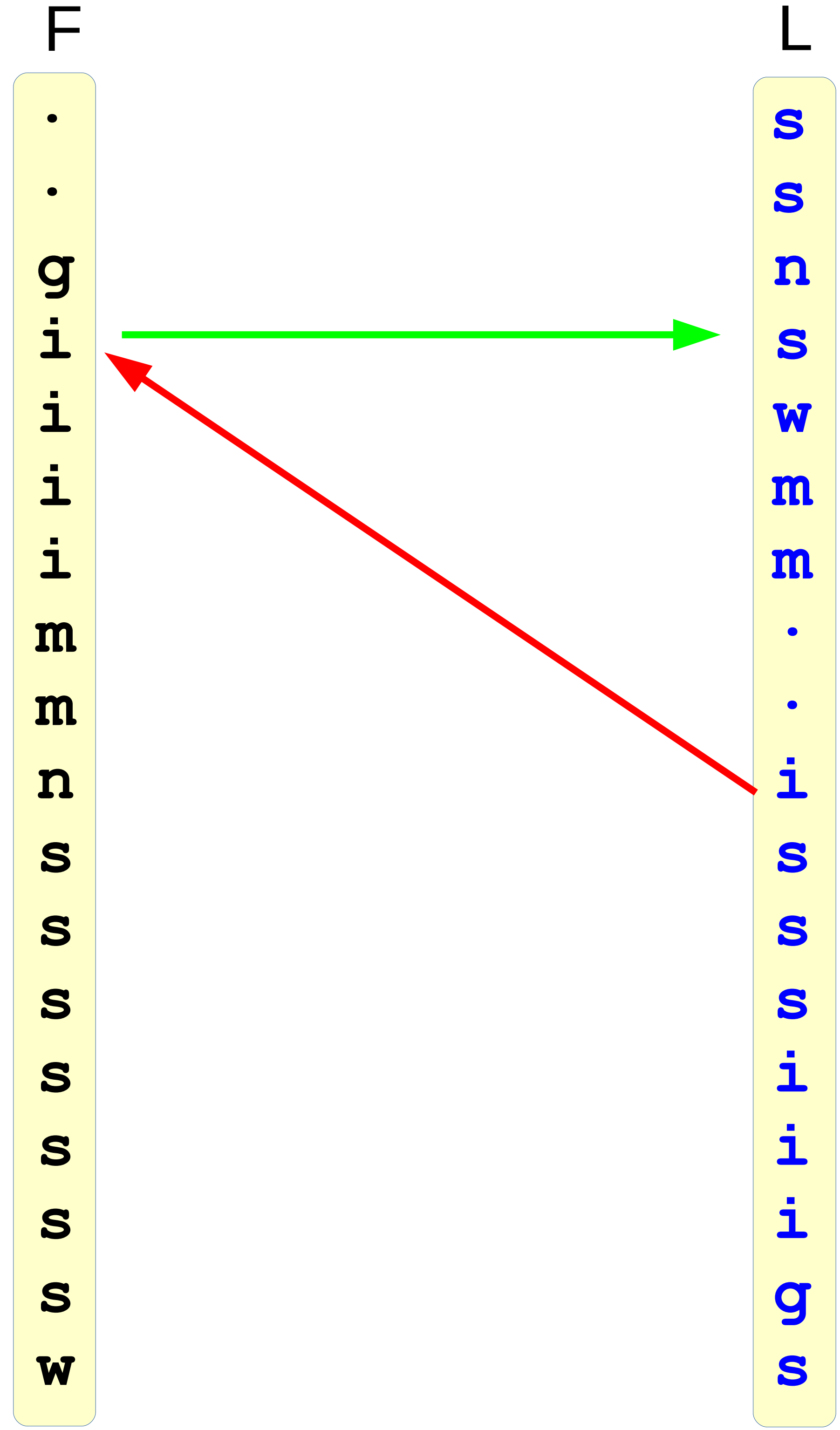
Using the LF map we retrieve T right-to-left

T = ing



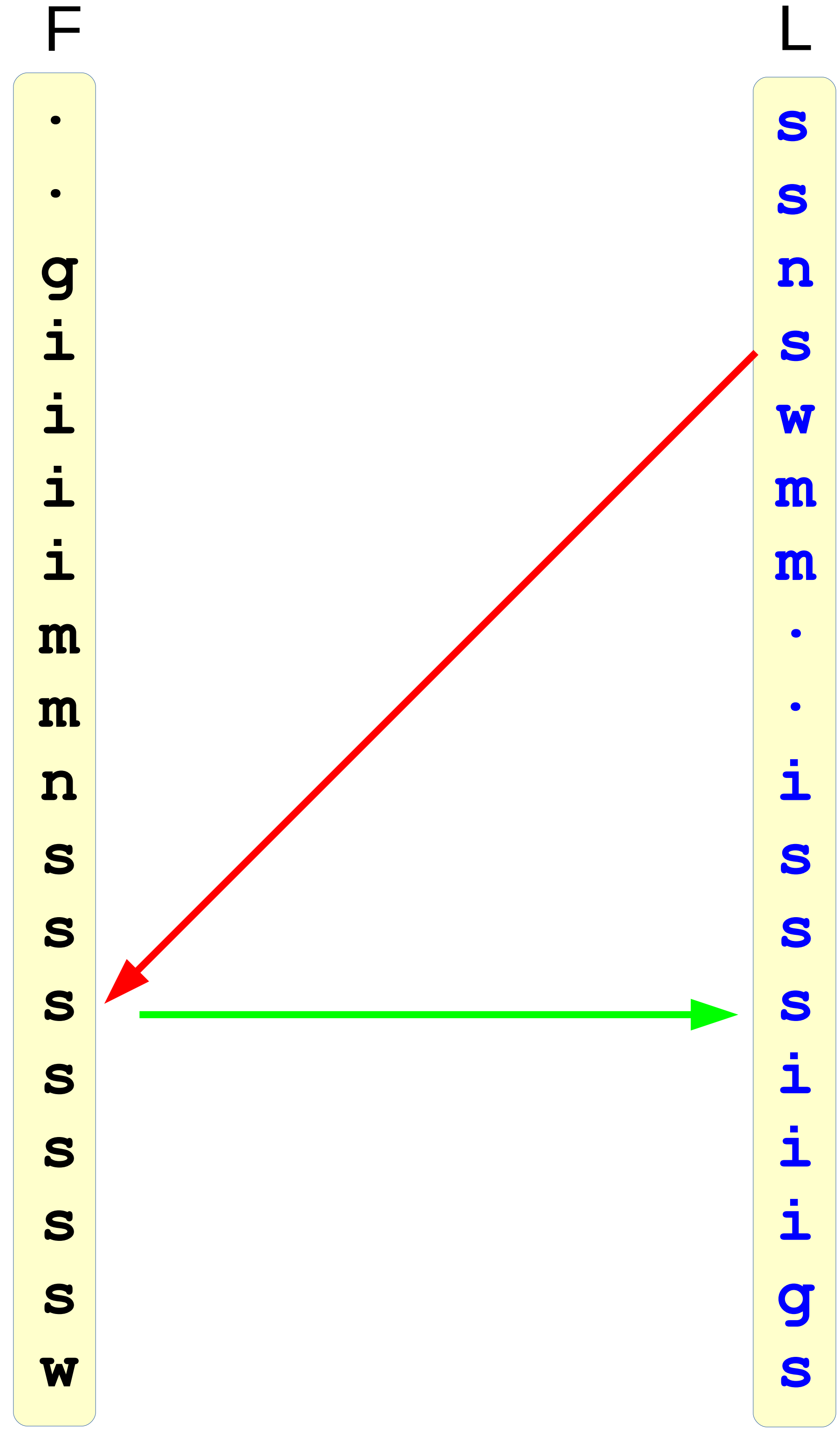
Using the LF map we retrieve T right-to-left

T = sing



Using the LF map we retrieve T right-to-left

T = ssing



What about compression?

Since BWT is a permutation of T, why should it be “easier” to compress?

Burrows and Wheeler observed that the BWT is usually “locally homogeneous” and suggested to compress it with Move-To-Front followed by an Order0 encoder (Huffman/Arithmetic Coding).

final char (L)	sorted rotations
a	`n to decompress.  It achieves compression`
o	`n to perform only comparisons to a depth`
o	`n transformation}  This section describes`
o	`n transformation}  We use the example and`
o	`n treats the right-hand side as the most`
a	`n tree for each 16 kbyte input block, enc`
a	`n tree in the output stream, then encodes`
i	`n turn, set $L[i]$ to be the`
i	`n turn, set $R[i]$ to the`
o	`n unusual data. Like the algorithm of Man`
a	`n use a single set of probabilities table`
e	`n using the positions of the suffixes in`
i	`n value at a given point in the vector $R`
e	`n we present modifications that improve t`
e	`n when the block size is quite large.  Ho`
i	`n which codes that have not been seen in`
i	`n with $ch$ appear in the {\em same order`
i	`n with $ch$.                  In our exam`
o	`n with Huffman or arithmetic coding.  Bri`
o	`n with figures given by Bell~\cite{bell}.`

Figure 1: Example of sorted rotations. Twenty consecutive rotations from the sorted list of rotations of a version of this paper are shown, together with the final character of each rotation.

BWT vs $H_k(s)$ (1)

Let s = ***ippississim***,

s^R = ***mississippi***

BWT(s^R) =

```
imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi
```

BWT vs $H_k(s)$ (2)

Let s = ***ippississim***,

s^R = ***mississippi***

BWT(s^R) =

```
imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi
```

$H_1(s) = (4/11)\, H_0(pssm) + (1/11)\, H_0(i)$
$\quad + (2/11)\, H_0(pi) + (4/11)\, H_0(ssii)$

To compress up to $H_1(s)$ it suffices to compress each segment up to H_0

BWT vs $H_k(s)$ (3)

Let s = ***ippississim***,

s^R = ***mississippi***

$BWT(s^R)$ =

```
imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi
```

To compress up to $H_2(s)$ it suffices to compress each segment up to H_0

Summing up

To compress up to $H_k(s)$ it suffices to compress the corresponding partition of $BWT(s^R)$ up to H_0 (compare with PPM).

However:

- computing the partition is not easy
- which k should we choose?

It is possible to find an optimal partition in linear time, but there is a simpler alternative...

A closer look to MTF

ccbbaaabdddcccccc

11213112411411111

The integers are in the range [1,h]
h=alphabet size

Given s, $|\text{Gamma}(\text{MTF}(s))| \leq 2|s|H_0(s) + |s|$ only if in the initial MTF list the symbols are in the same order as in s, in our example: cbad.

If not, the penalty is at most log h bits per distinct symbol, for any initial status of the MTF list

$|\text{Gamma}(\text{MTF}(s))| \leq 2|s|H_0(s) + |s| + h \log h$ bits

Given three strings x,y,z

MTF(x y z) differs from MTF(x) MTF(y) MTF(z) only for the status of the MTF list at the beginning of the encoding of y and z.

From the bound on the previous slide:

Gamma(MTF(x y z)) ≤

$$2|x|H_0(x) + 2|y|H_0(y) + 2|z|H_0(z) + |x|+|y|+|z| + 3\, h \log h$$

The same is true if we have more than three strings!

Since MTF has "little memory", encoding the concatenation $x_1, x_2, \ldots, x_t$ produces an output bounded by:

$$\Sigma_i\ 2|x_i|H_0(x_i) + |x_i| + O(t\ h\log h) \text{ bits}$$

This is precisely what we need for the BWT!

$BWT(s^R) =$

```
imississipp
ippimississ
issippimiss
ississippim
mississippi
pimississip
ppimississi
sippimissis
sissippimis
ssippimissi
ssissippimi
```

To compress up to $H_2(s)$ it suffices to compress each segment up to H_0

Main result

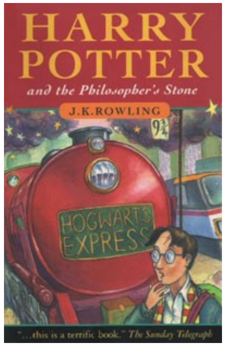
For any string s and order k:

$|\mathrm{Gamma}(\mathrm{MTF}(\mathrm{BWT}(s))| \leq$

$$2\,|s|\,H_k(s) + |s| + O(h^{k+1} \log h) \quad \text{bits}$$

The bound can be improved replacing Gamma with another encoder, and applying Run Length Encoding (RLE) on the output of MTF

1997



First Harry Potter novel published

bzip2

- In 1997 Julian Seward released the bzip compression tool based on the BWT.
- The output of the BWT was compressed using Move-to-front, RLE, and Multiple Tables Huffman Coding
- It was highly optimized, and could be used as a drop-in replacement of gzip, both command line and library version

BWT as a compressor today

- Most of the mainstream compressors released in the last 20 years are based on LZ77 parsing, eg. LZMA, Snappy, Brotli, Zstd
- LZ77 has more “free parameters” and can offer a wide range of compression/speed trade-offs
- In BWT compression we cannot easily trade compression for speed

Compression results (1)

Size	Ratio %	C.MB/s	D.MB/s	Compressor (Binary 42% + Text 58%) Silesia.tar
48616057	22.9	**1.07**	**77.11**	**LzTurbo 49**
48758739	23.0	**2.47**	**81.17**	**lzma 9**
49517150	23.4	0.46	**336.19**	**brotli 11d29**
50861542	24.0	1.68	269.97	lzham 4
51720632	24.4	1.42	**1239.95**	**LzTurbo 39**
52715921	24.9	2.03	602.56	zstd 22
54596837	25.8	**11.80**	38.94	**bzip2**
58008992	27.4	7.96	853.20	zstd 15
59273940	28.0	**59.48**	**1293.41**	**LzTurbo 32**
59581397	28.1	33.48	416.81	brotli 5
60411647	28.5	45.64	798.97	zstd 9
60813803	28.7	1.60	**2002.86**	**LzTurbo 29**
64141404	30.3	**162.02**	1372.34	**LzTurbo 31**
64191258	30.3	65.28	416.81	**brotli 4**
64711652	30.5	0.22	325.27	zopfli
67624724	31.9	62.86	692.87	lzfse
67647204	31.9	9.99	316.72	zlib 9
68225985	32.2	24.46	313.67	zlib 6

bzip2 compresses well but it is relatively slow in compression and the slowest in decompression

Compression results (2)

Size	Ratio %	C.MB/s	D.MB/s	Compressor	Text log: NASA_access_log
11355945	5.5	**0.86**	**320.68**	**LzTurbo 49**	
11907661	5.8	**0.99**	**2502.71**	**LzTurbo 39**	
11960483	5.8	**10.13**	67.81	**bzip2**	
12236072	6.0	0.51	1022.47	brotli 11d29	
12617026	6.1	1.36	1348.32	zstd 22	
13598062	6.6	2.68	265.69	lzma 9	
13651218	6.7	1.33	880.25	lzham 4	
14661031	7.1	8.67	1819.99	zstd 15	
15041556	7.3	1.13	**3732.63**	**LzTurbo 29**	
16665926	8.1	**78.89**	1245.90	brotli 5	
17387746	8.5	**117.98**	1375.73	**zstd 9**	
18279979	8.9	**187.64**	2186.17	**LzTurbo 32**	
18654669	9.1	173.25	1227.89	brotli 4	
19085875	9.3	1.50	3527.36	lizard 49	
19545036	9.5	32.75	651.55	zlib 9	

for very compressible files bzip2 is more competitive in compression but still slow...