Text Indexing, Suffix Sorting & Data Compression: Common Problems and Techniques

Roberto Grossi

Dipartimento di Informatica

Università di Pisa

Roadmap

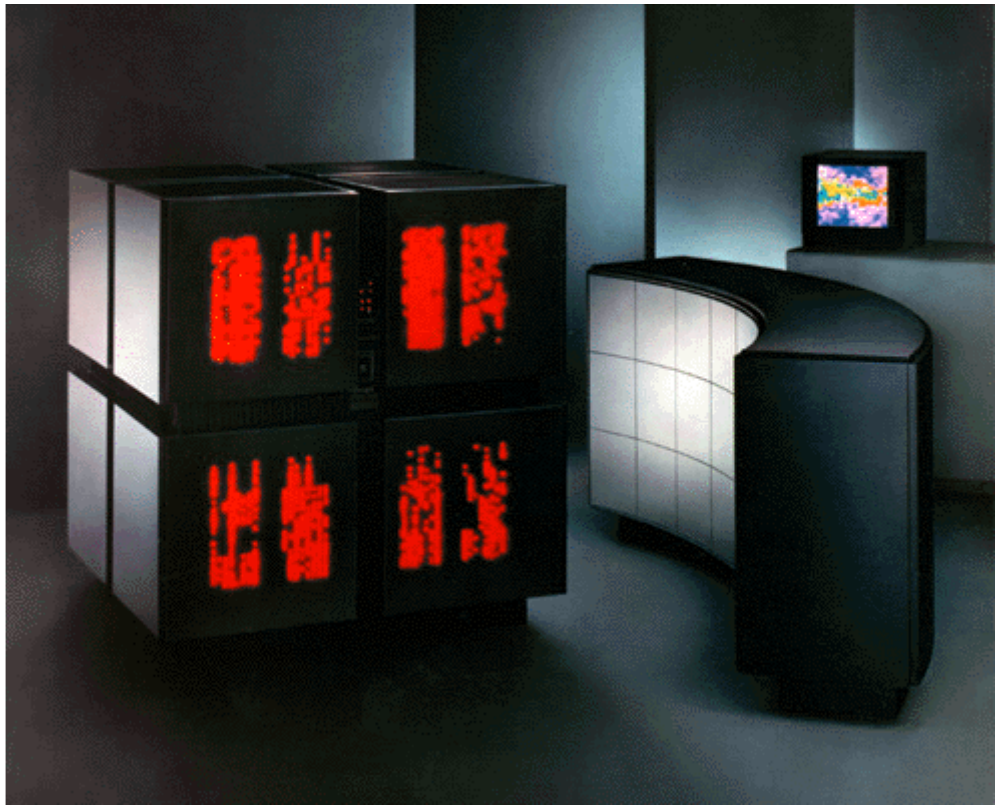
Space efficiency issues

Short stories:

- suffix array permutations
  - succinct data structures
  - compressed text indexing

Goal: give flavor, no technicalities

My first space issues...

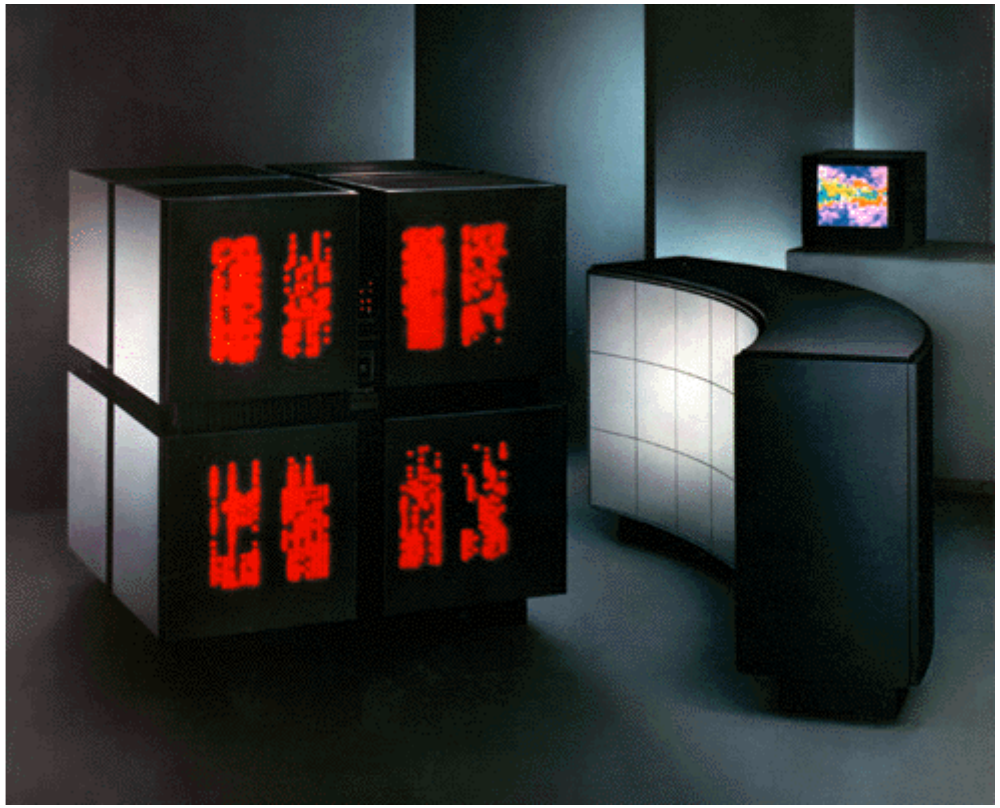


Connection Machine CM-2

- 4096 single-bit processors
- hypercube layout
- 512 MB RAM
- 25 GB Data Vault (RAID)
- cost ~ 500Keuro in 1990

AILSV parallel algorithm for suffix tree construction in O(log n) time

My first space issues...



Connection Machine CM-2

- 4096 single-bit processors
- hypercube layout
- 512 MB RAM
- 25 GB Data Vault (RAID)
- cost ~ 500Keuro in 1990

max text length:
just n = 256 chars ...

AILSV parallel algorithm for suffix tree construction in O(log n) time

Few years later at AT&T Bell Labs

Visiting Ken Church (the “gigabyte” guy) with Paolo Ferragina

AMEX phone bill

- **1 CD** as bill couldn’t be printed!
- **12 CD** for full text indexing...

Q: Just algorithm engineering or some theory behind that?

Situation in the 90s...

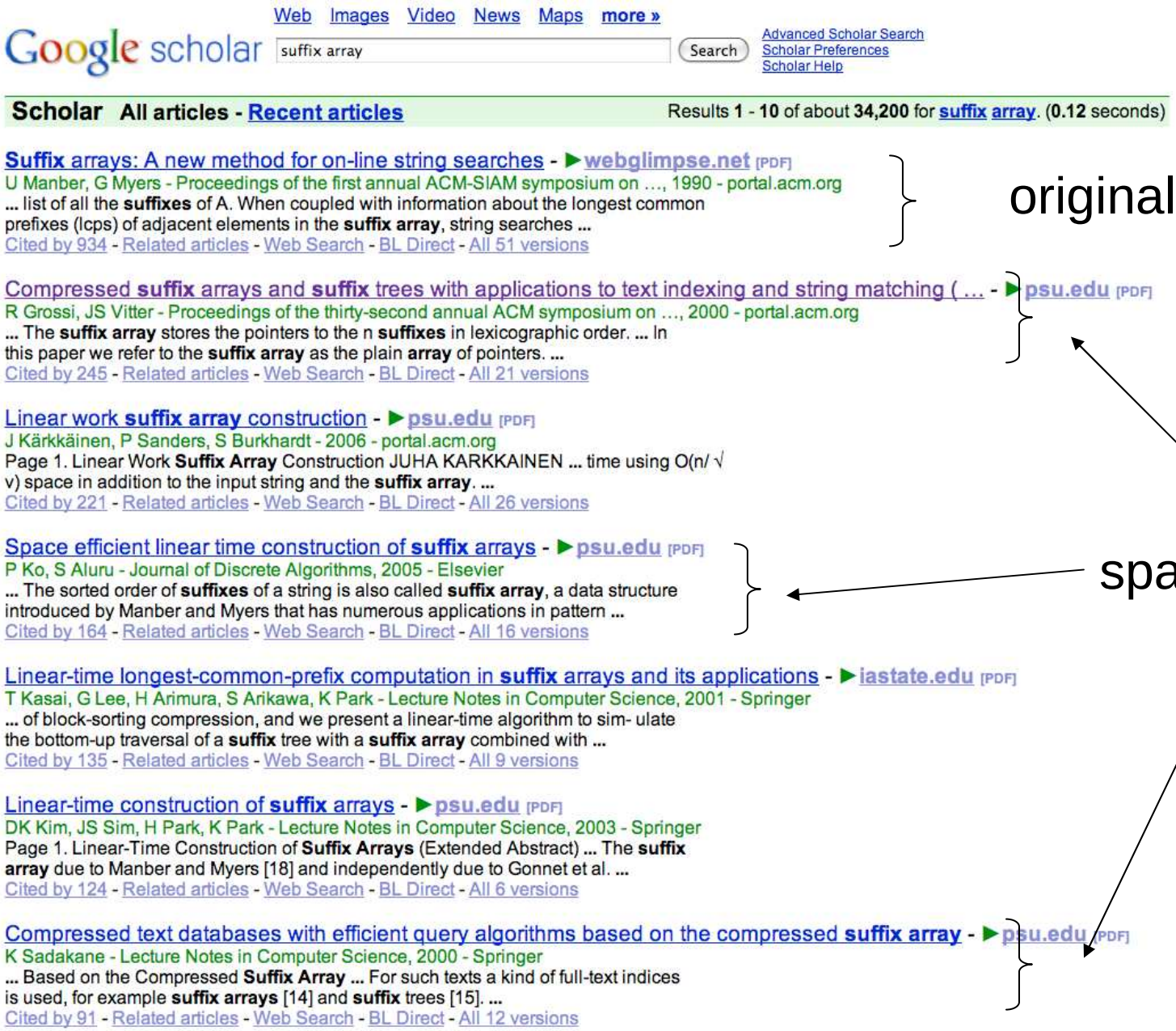
`zgrep:` searching on compressed files
but needed to decompress them on the fly

New trend: keep the file in compressed format!

Amir, Benson, Farach ’94: text scanning algorithm
Karkkainnen, Sutinen, Ukkonen ’96: text indexing algorithms
...more and more papers after the above ones

NOW: compressed text indexing (text + index)
~ space **`(bzip|gzip)`** text alone

Google Scholar search "**`suffix array`**"



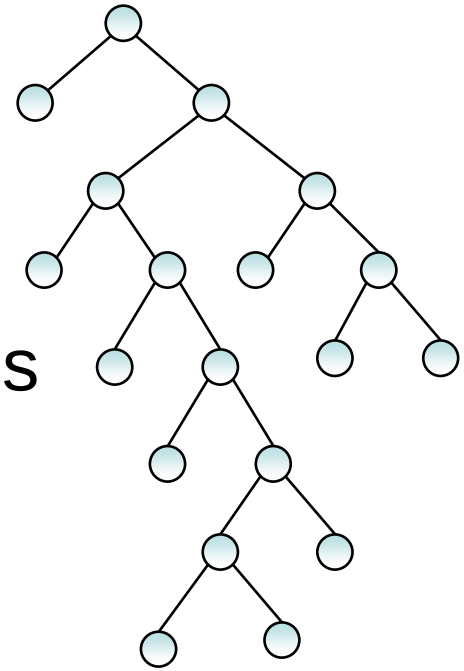
original paper

space issues

First story: Suffix Array (SA)

THE PERMUTATION in Stringology
[Gonnet ’87, Manber, Myers ’90]

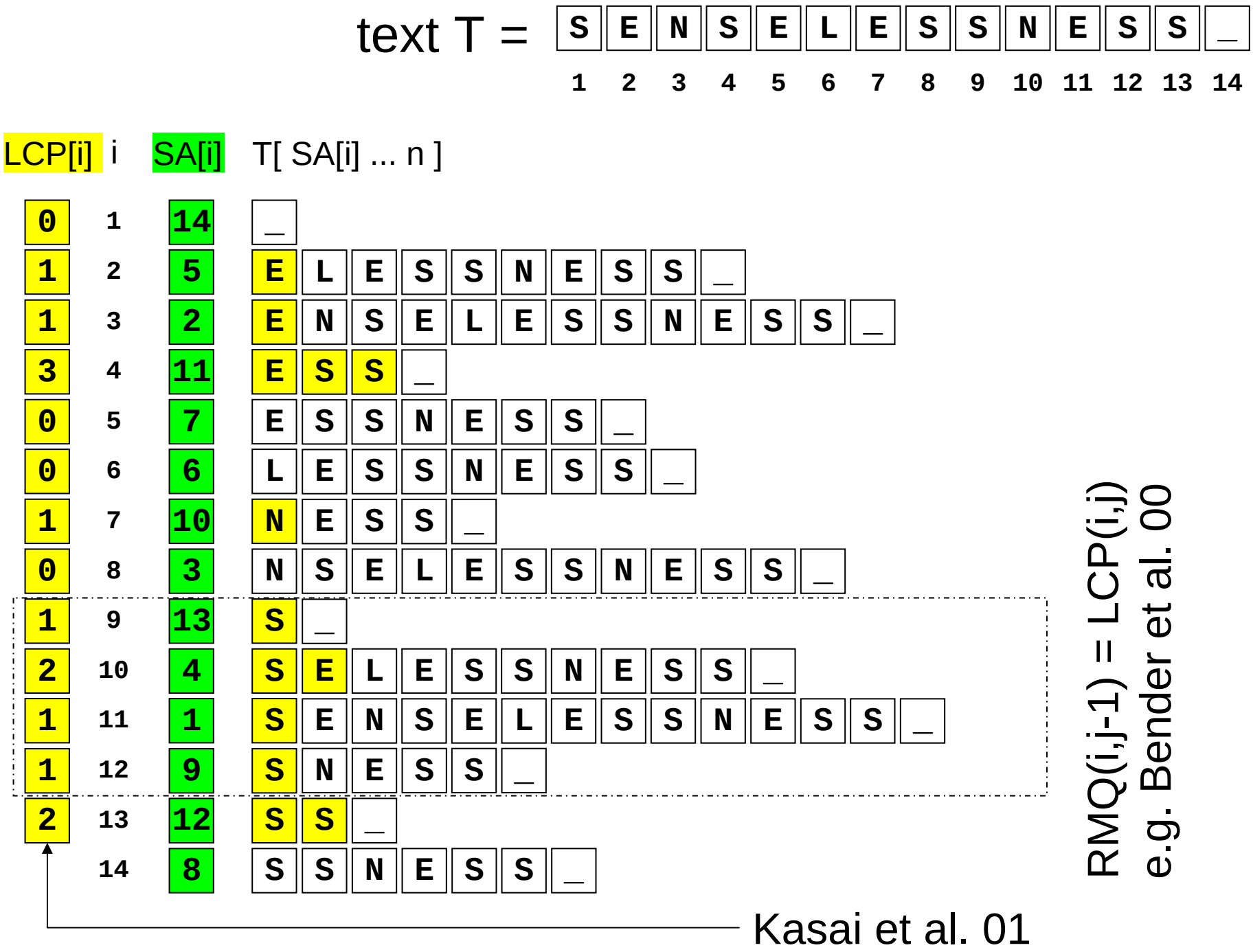
More than just a simplification of suffix trees
[Weiner ‘73, Mc Creight ‘76]



text T =

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

i	SA[i]	T[SA[i] ... n]
1	14	_
2	5	E L E S S N E S S _
3	2	E N S E L E S S N E S S _
4	11	E S S _
5	7	E S S N E S S _
6	6	L E S S N E S S _
7	10	N E S S _
8	3	N S E L E S S N E S S _
9	13	S _
10	4	S E L E S S N E S S _
11	1	S E N S E L E S S N E S S _
12	9	S N E S S _
13	12	S S _
14	8	S S N E S S _


text T = S E N S E L E S S N E S S _
1 2 3 4 5 6 7 8 9 10 11 12 13 14
LCP[i] i SA[i] T[SA[i] ... n]
0 1 14 _
1 2 5 E L E S S N E S S _
1 3 2 E N S E L E S S N E S S _
3 4 11 E S S _
0 5 7 E S S N E S S _
0 6 6 L E S S N E S S _
1 7 10 N E S S _
0 8 3 N S E L E S S N E S S _
1 9 13 S _
2 10 4 S E L E S S N E S S _
1 11 1 S E N S E L E S S N E S S _
1 12 9 S N E S S _
2 13 12 S S _
14 8 S S N E S S _
Kasai et al. 01
RMQ(i,j-1) = LCP(i,j)
e.g. Bender et al. 00

Disproving two beliefs on LCP info

When searching for a pattern P of length m into text T of length n:

- O(m log n) time is the **best** without LCP info (?)

- O(m + log n) time using **extra** space for LCP info (?)

What if using a plain SA permutation?

Implicit search in lexicographically sorted array
[Andersson et al. ’94, ’95, ’01]

$$\Theta\left(\frac{m \log\log n}{\log\log(4 + \frac{m \log\log n}{\log n})} + m + \log n\right)$$

Upper bound can be better than what we say O(m log n)
Lower bound is for n independent strings of length m

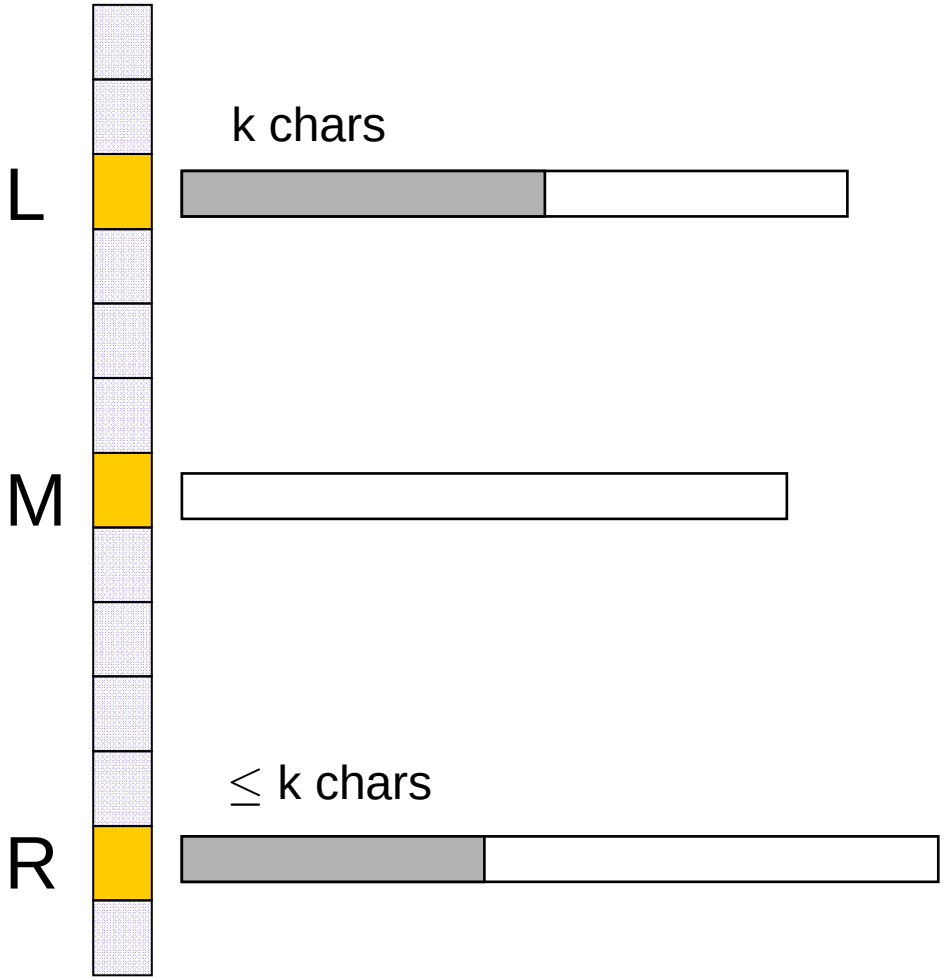
Need extra space for LCP info?

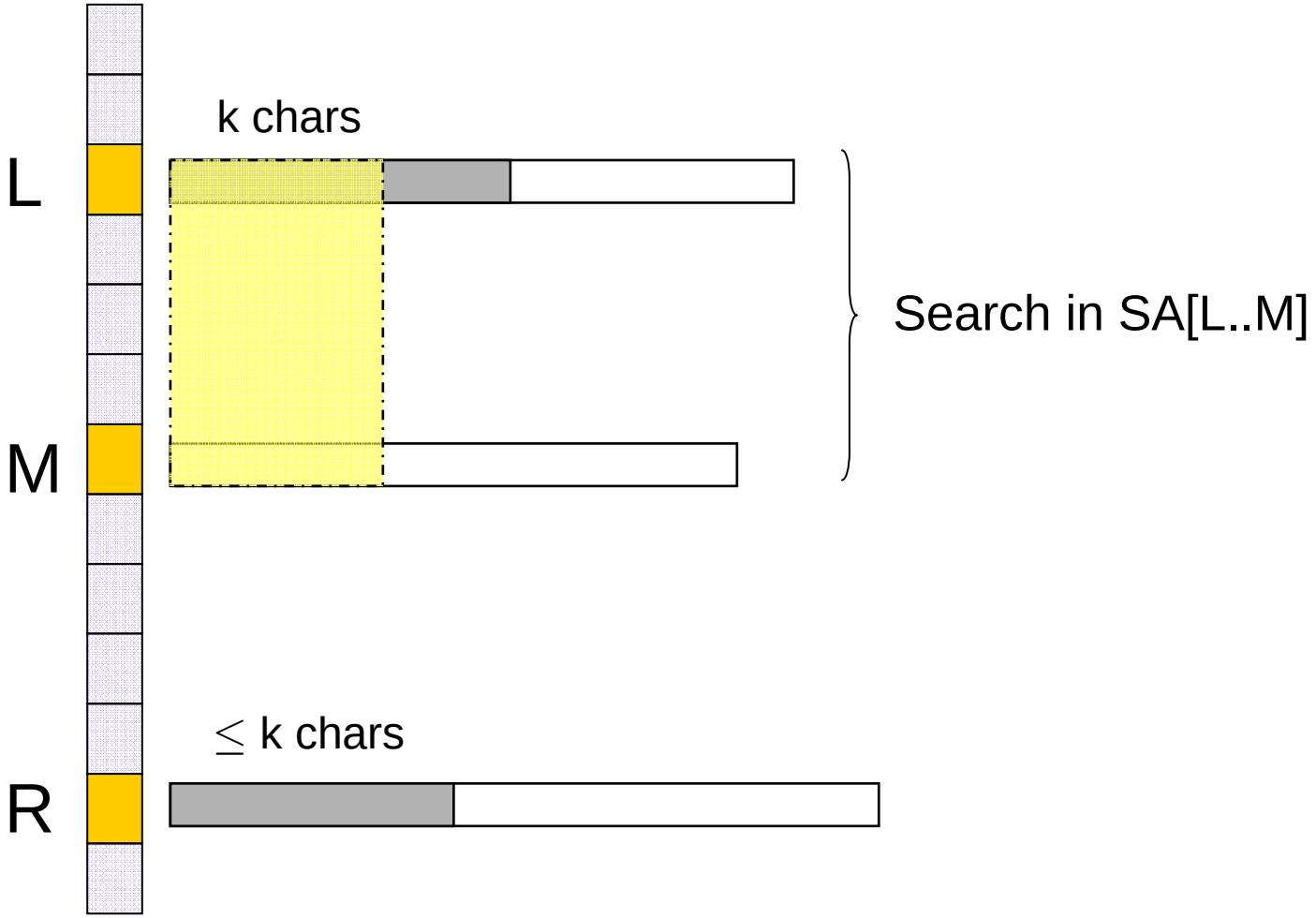
Let's first quickly review SA search [Manber, Myers '09]

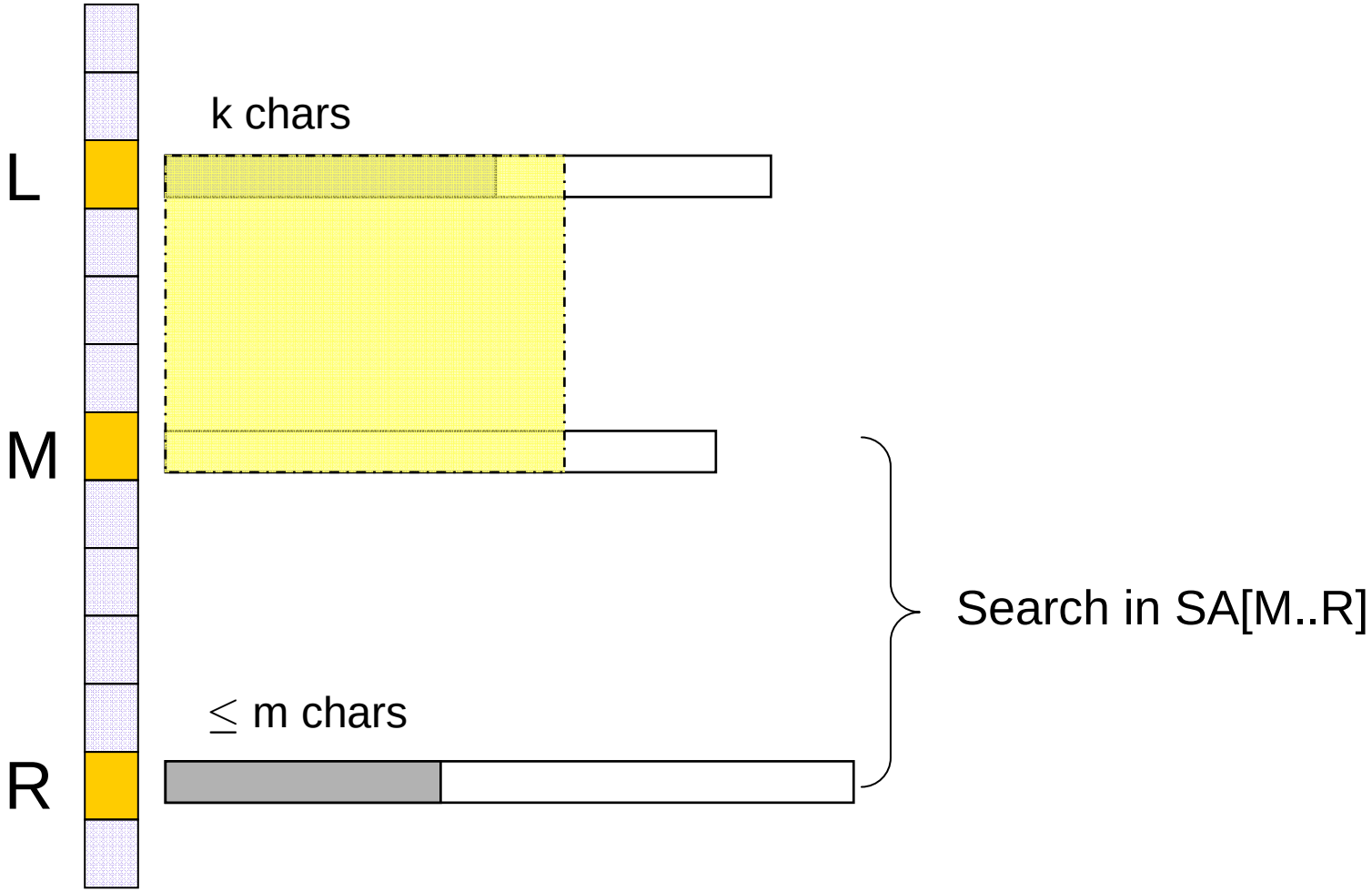
- Invariant for binary search step in SA[L..R]
  - *Pre*: matched k < m chars
  - *Post*: matched k' chars (k $\leq$ k' $\leq$ m)

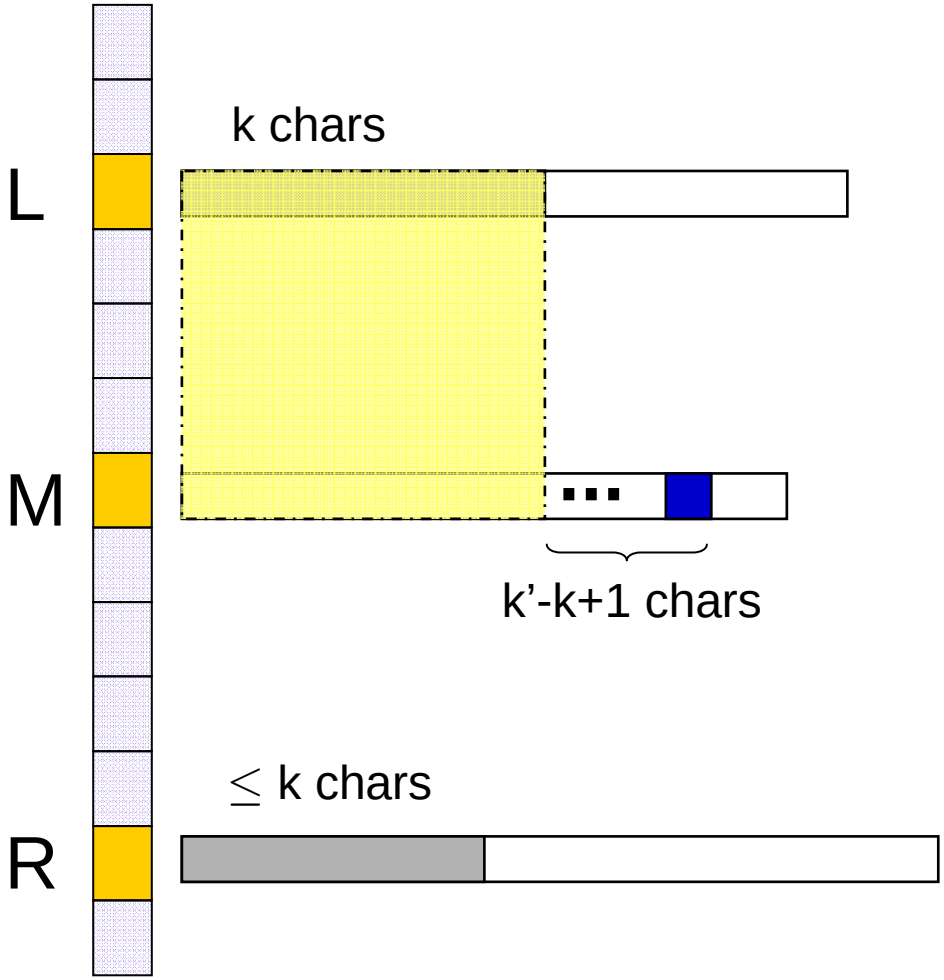
- Proceed by visual case analysis

“Visual” analysis of one case




k chars
L
M
Search in SA[L..M]
≤ k chars
R


k chars
L
M
R
≤ m chars
Search in SA[M..R]



Match k'-k chars and
decide SA[L..M] or SA[M..R]
according to the **mismatch**

Making SA search “parsimonius”

It is possible store an lcp value in $O(\log^2 n)$ bits,
so that it can be compared by fetching just O(1) of its bits

SA search can be modified so that
it accesses O(1) bits of LCP info per step

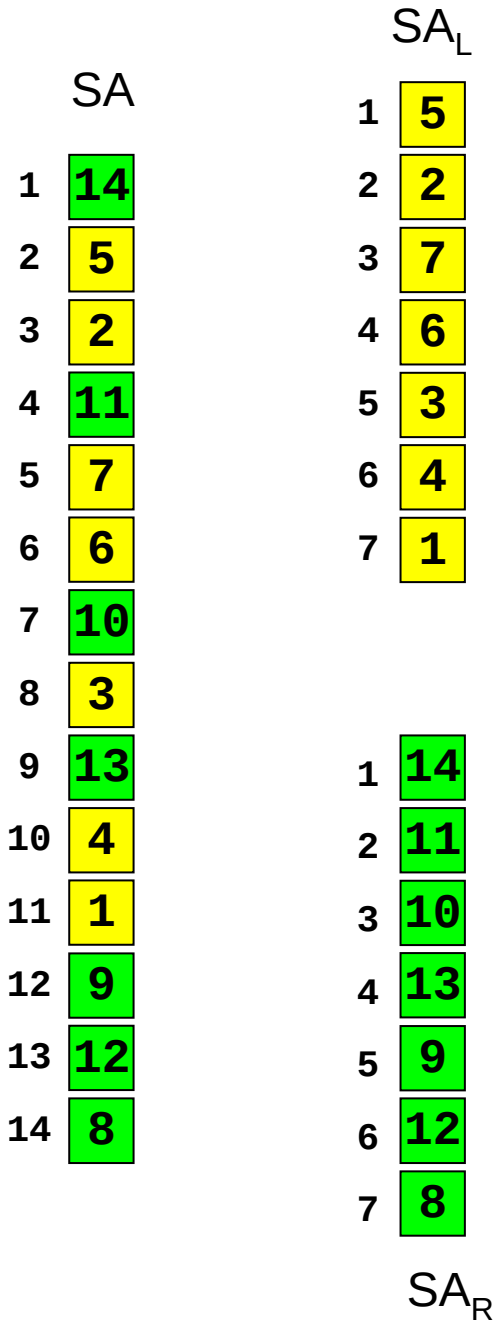
O(m + log n) search on a “perturbed” SA without extra space for LCP info

[Francheschini & G. ’04]

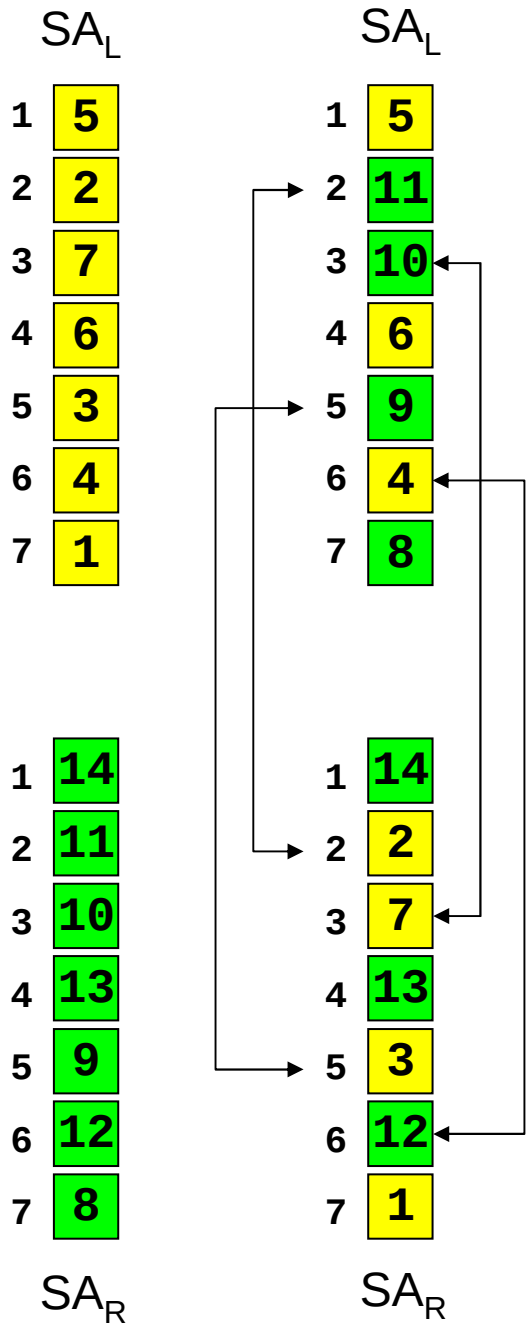
Rules of the game:

- given a read-only text T of n characters
- given a permutation of [1...n], each entry of $\lceil \log n \rceil$ bits
- workspace of just O(1) additional cells [O(log n) bits]
- how to find P in T using the above permutation

Bit stealing idea



- Partition SA permutation into SA_L and SA_R:
  - $SA_L[i] \leq n/2$ and $SA_R[i] > n/2$
- One search in SA $\Rightarrow$ one in SA_L and another in SA_R



Bit stealing idea

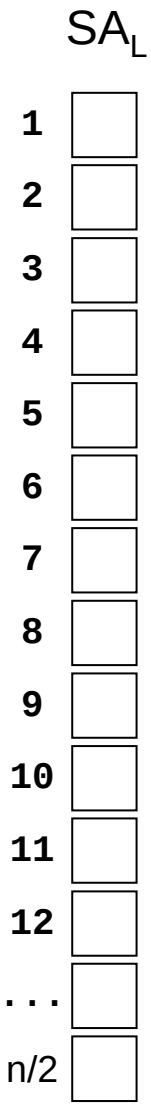
Define twin pair to encode one bit

- **0** ≡ ($SA_L[i]$, $SA_R[i]$))
- **1** ≡ ($SA_R[i]$, $SA_L[i]$))

So we have n/2 bits for “free”

Example: encoding **0110101**

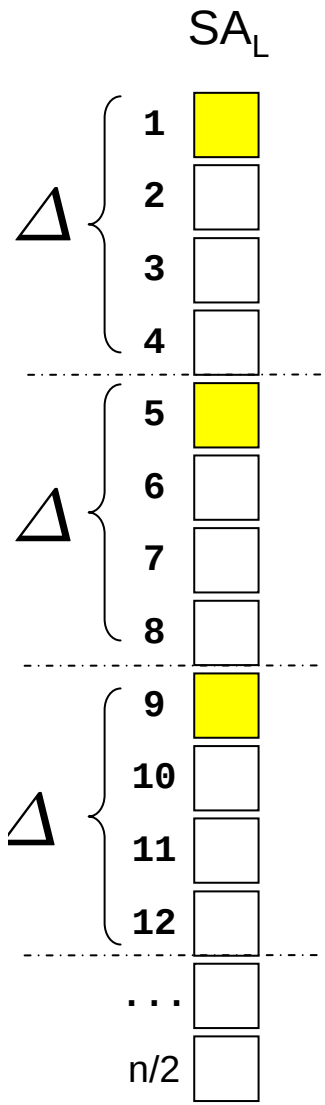
Can still retrieve any original entry from SA_L or SA_R in O(1) time



New game:
search SA_L (or SA_R) using n/4 stolen bits for LCP

recall parsomonius SA search retrieves
O(1) bits per step

Goal: O(m + log n) search time without extra space for LCP info



1. fix $\Delta = \Theta(\log^2 n)$
2. sample every other Δth suffix in SA_L
3. encode one LCP value per sample (bit stealing)
4. run parsimonius SA search and locate a run of Δ consecutive suffixes in SA_L
5. use Hirschberg's linear search twice in $O(m + \Delta^{1/2})$ time

Total time = O(m + log n)

Space-efficient computation of SA

Same game rules as implicit searching

In-place suffix sorting:

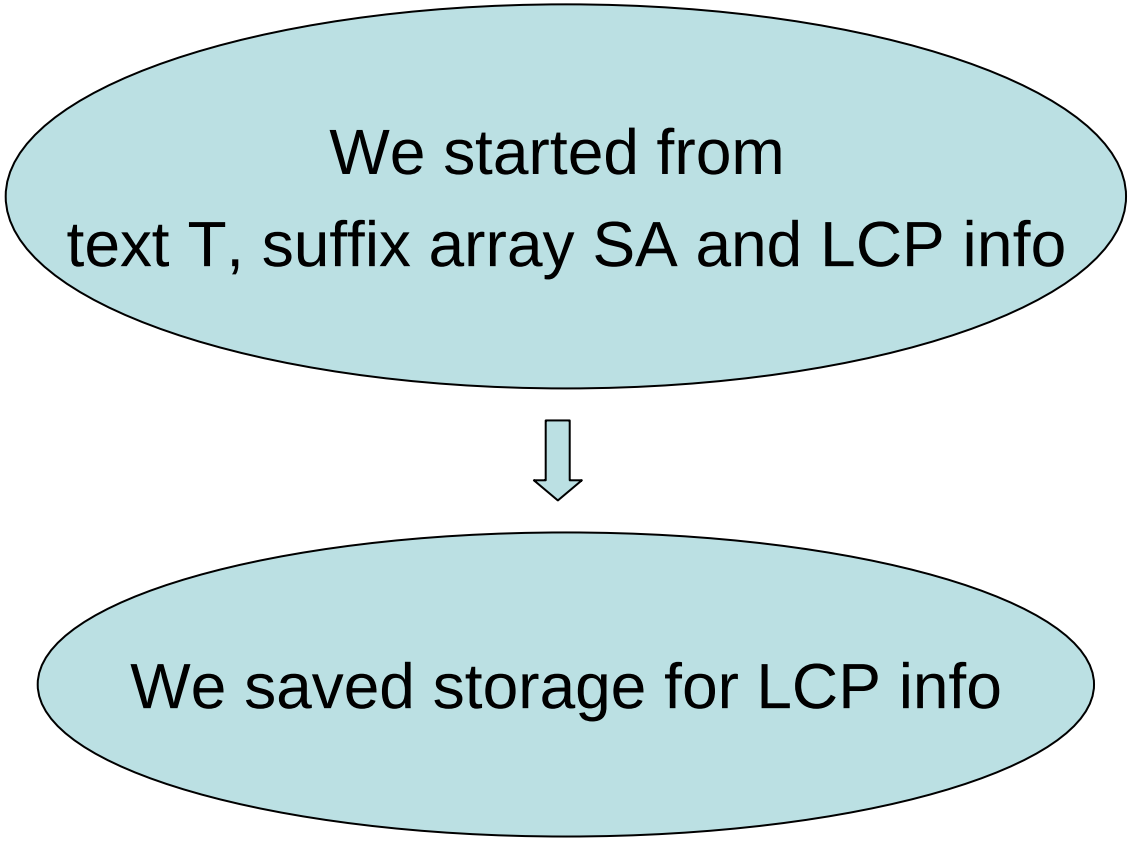
O(n log n) time and O(1) additional space

[Franceschini, Muthu ’07]

based on Ko and Aluru’s suffix sorting

Useful to compute our SA permutation

Checkpoint



Goal: can we also drop text T?

A Dedication to French thinkers

Cette denrée [savoir/informaiton], donc, se préparera sous des formes de plus en plus maniables ou comestibles ; elle se ***distribuera*** *à une clientèle de plus en plus nombreuse ; elle deviendra chose du* ***Commerce****, chose enfin qui* ***s'imite*** *et se* ***produit un peu partout****.*

Paul Valéry, La Crise de l'esprit, **1919**,

This commodity [knowledge/information], as a result,
will be prepared in more and more convenient or comestible forms; it will be ***distributed*** *to a larger and larger circle of buyers: it will be transformed into something* ***commercial****, something which is* ***imitated*** *and* ***produced almost everywere****.*

Paul Valéry The intellectual crisis, **1919** (translated by Malcolm Cowley)

Checkpoint

We started from
text T, suffix array SA and LCP info

We saved storage for LCP info

Goal: can we also drop text T?

Are all [1..n]-perms SA permutations?

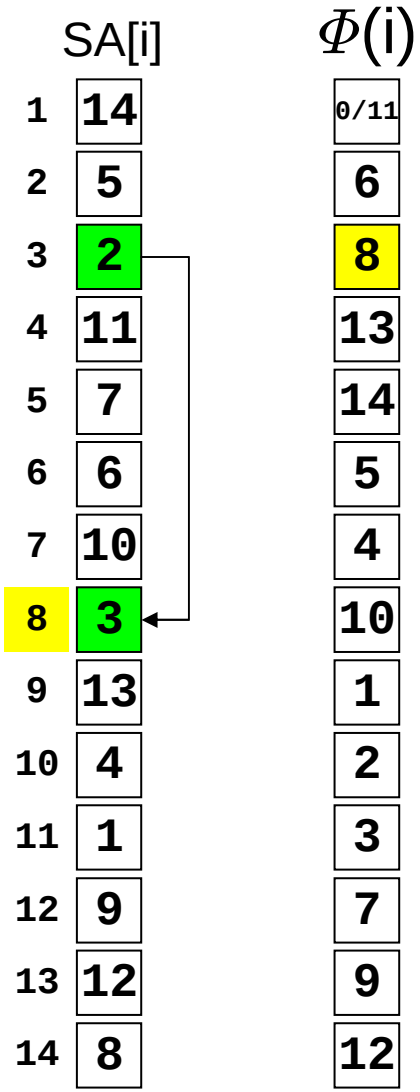
1	2	3
A	A	A
A	A	B
A	B	A
A	B	B
B	A	A
B	A	B
B	B	A
B	B	B

1	2	3
3	2	1
1	2	3
3	1	2
1	3	2
3	2	1
2	3	1
3	2	1
3	2	1

- Some permutations do **not** occur (e.g. **213**)
- Some permutations occur **more than once** (e.g. **321**)

Σ = {**A**, **B**}, n =3

Getting itchy about combinatorics of SA perms?



"suffix link" for SA:
Φ function [G.&Vitter '00]:

Φ(i) = k iff SA[k] = SA[i] + 1

rank of the next suffix in T

Φ(i)=0 when SA[i]=n or

Φ(i)=z where SA[z]=1

Necessary and SUFFICIENT conditions

Duval et al ‘02, Bannai et al ’03, Burkhardt et al ’03, Hohlweg et al ’03, Crochemore et al ‘05, He et al ’05, Kopelowitz et al ’05, Schurman et al ’05

SA is the suffix array of T **iff** for each i

- T[SA[i]] $\leq$ T[SA[i+1]]
- T[SA[i]] = T[SA[i+1]] $\Rightarrow$ Φ(i) < Φ(i+1)

SA is the suffix array of T **iff** for each i

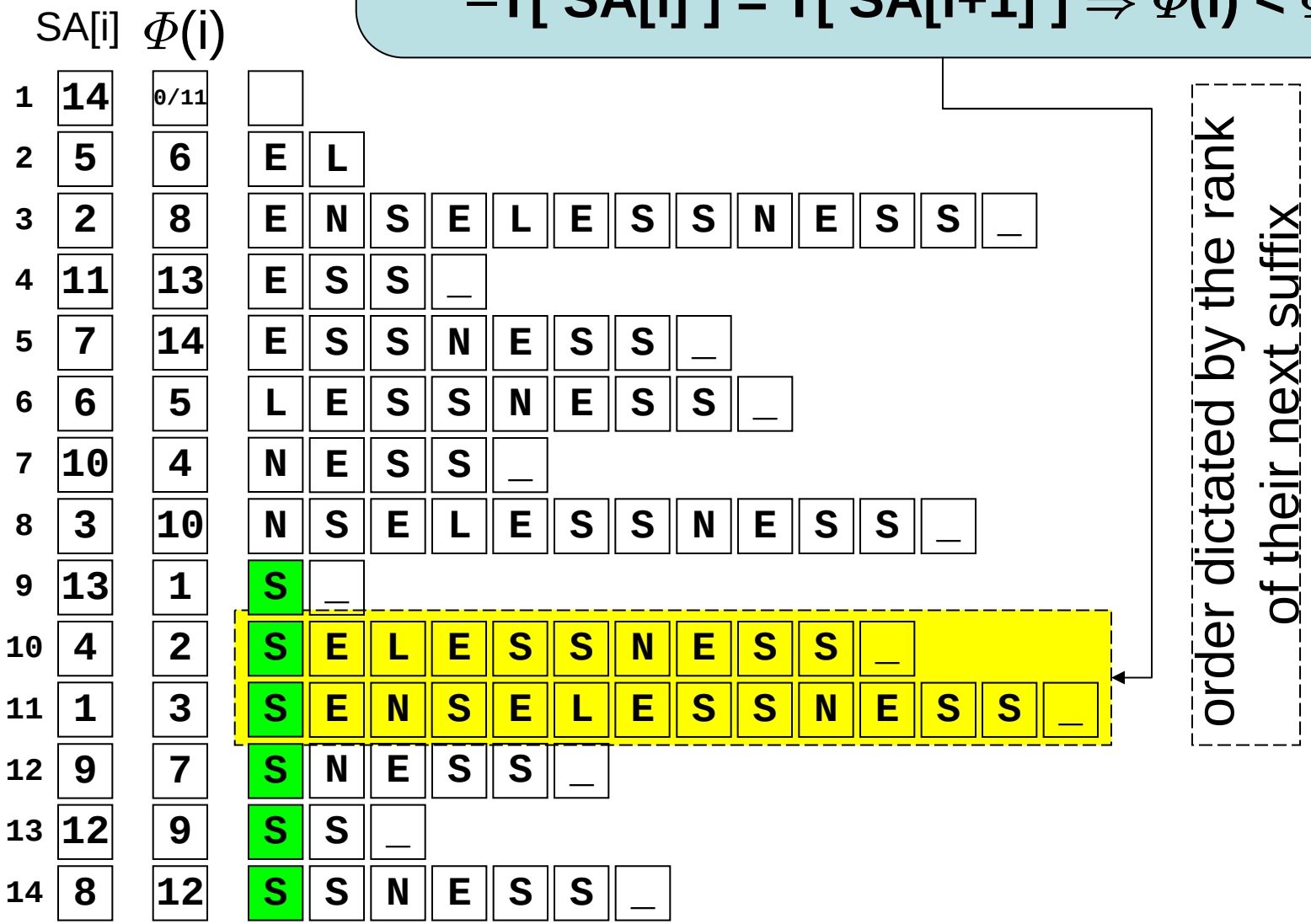
– **T[SA[i]] ≤ T[SA[i+1]]**

– T[SA[i]] = T[SA[i+1]] ⇒ Φ(i) < Φ(i+1)

SA is the suffix array of T **iff** for each i

– $T[SA[i]] \leq T[SA[i+1]]$

– **$T[SA[i]] = T[SA[i+1]] \Rightarrow \Phi(i) < \Phi(i+1)$**



How many SA permutations?

[Schurmann, Stoye '05]

	SA[i]	Φ(i)
1	14	0/11
2	5	6
3	2	8
4	11	13
5	7	14
6	6	5
7	10	4
8	3	10
9	13	1
10	4	2
11	1	3
12	9	7
13	12	9
14	8	12

Position i is a **descent** if $\Phi(i) > \Phi(i+1)$

d descents $\Rightarrow$ # strings having the same SA is

$$\binom{n+\sigma-d-1}{\sigma-d-1}$$

Are all [1..n]-perms SA permutations?

1	2	3
A	A	A
A	A	B
A	B	A
A	B	B
B	A	A
B	A	B
B	B	A
B	B	B

1	2	3
3	2	1
1	2	3
3	1	2
1	3	2
3	2	1
2	3	1
3	2	1
3	2	1

Σ = {**A**, **B**}, n =3, d = 0

$$\binom{n+\sigma-d-1}{\sigma-d-1} = \binom{3+2-0-1}{2-0-1} = \binom{4}{1} = 4$$

How many distinct SAs for all strings in Σ^n ?

[Schurmann , Stoye '05]

$\left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle$ = Eulerian number = #perms with d descents

$$\#\text{ SAs} = \sum_{d=0}^{\sigma-1} \left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle \quad \text{and} \quad \lim_{n\to\infty} \frac{\sum_{d=0}^{\sigma-1} \left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle}{\sigma^n} = 1$$

constant σ

Are all [1..n]-perms SA permutations?

1	2	3
A	A	A
A	A	B
A	B	A
A	B	B
B	A	A
B	A	B
B	B	A
B	B	B

1	2	3
3	2	1
1	2	3
3	1	2
1	3	2
3	2	1
2	3	1
3	2	1
3	2	1

Σ = {**A**, **B**}, n =3, d = 0

In our example: $\left\langle \begin{matrix} 3 \\ 0 \end{matrix} \right\rangle + \left\langle \begin{matrix} 3 \\ 1 \end{matrix} \right\rangle = 1 + 4 = 5$

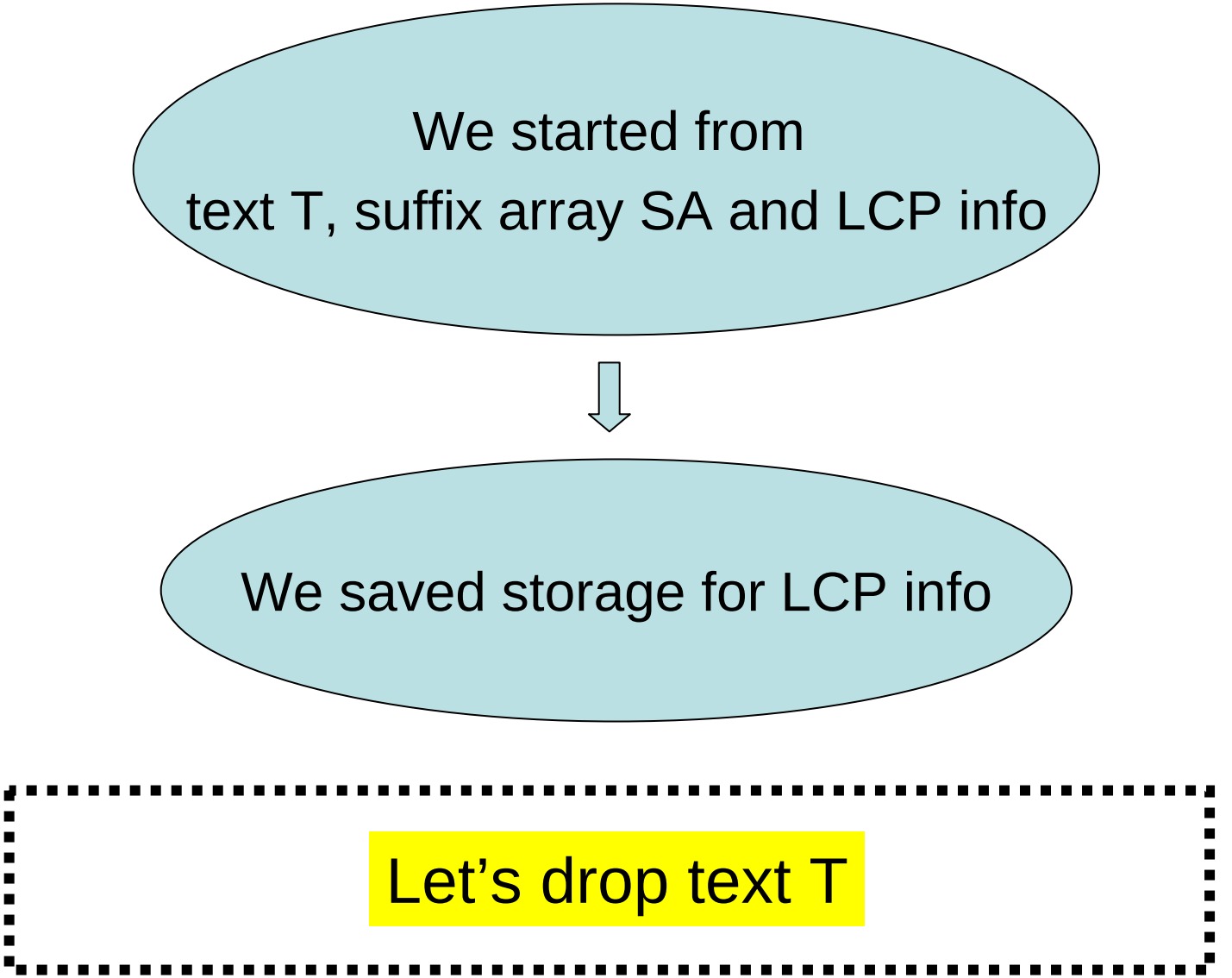
321 **123, 312, 132, 231**

Information theoretical lower bound

$$\log \left(\sum_{d=0}^{\sigma-1} \left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle \right) \sim n \log \sigma \text{ bits}$$

to be compared with $n \lceil \log n \rceil$ bits for the plain SA permutation (e.g. when σ is constant)

Checkpoint


We started from
text T, suffix array SA and LCP info
We saved storage for LCP info
Let’s drop text T

Text endmarkers make life easier

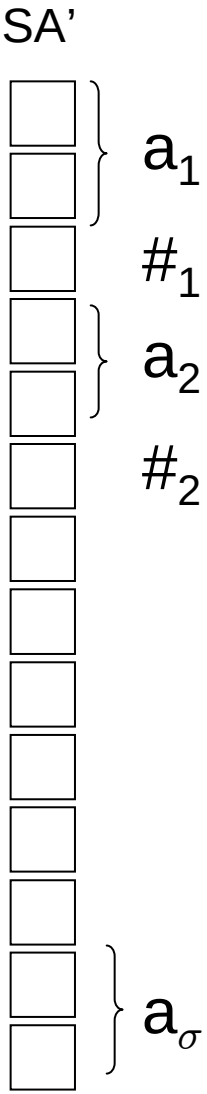
Σ' = {**A**, #, **B**}, n' = n + 1

1	2	3	4
A	A	A	#
A	A	B	#
A	B	A	#
A	B	B	#
B	A	A	#
B	A	B	#
B	B	A	#
B	B	B	#

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

all distinct perms

Conceptual transformation



Alphabet: $\Sigma = \{ a_1 < a_2 < \cdots < a_\sigma \} \rightarrow$

$\Sigma = \{ a_1 < \#_1 < a_2 < \#_2 < \cdots < a_\sigma \}$

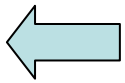
Text: $T \rightarrow T' = T \#_1 \#_2 \cdots \#_{\sigma-1}$

Resulting SA' can reconstruct text T

($\#_1$, $\#_2$, ... are not actually needed)

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

bv[i]		SA[i]	T[SA[i]]	Φ(i)
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12



Using Φ and a bitvector we can avoid storing SA

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

bv[i]		SA[i]	T[SA[i]]	Φ(i)
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12

Using Φ and a bitvector we can avoid storing SA

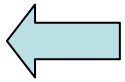
S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

		bv[i]	SA[i]	T[SA[i]]	Φ(i)
	1	_	14	_	0/11
	2	1	5	E	6
→	3		2	E	8
	4		11	E	13
	5		7	E	14
	6	1	6	L	5
	7	1	10	N	4
	8		3	N	10
	9	1	13	S	1
	10		4	S	2
	11		1	S	3
	12		9	S	7
	13		12	S	9
	14		8	S	12

Using Φ and a bitvector we can avoid storing SA

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

bv[i]		SA[i]	T[SA[i]]	Φ(i)
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12



Using Φ and a bitvector we can avoid storing SA

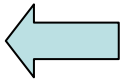
S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

	bv[i]	SA[i]	T[SA[i]]	Φ(i)
1	_	14	_	0/11
2	1	5	E	6
3		2	E	8
4		11	E	13
5		7	E	14
6	1	6	L	5
7	1	10	N	4
8		3	N	10
9	1	13	S	1
10		4	S	2
11		1	S	3
12		9	S	7
13		12	S	9
14		8	S	12

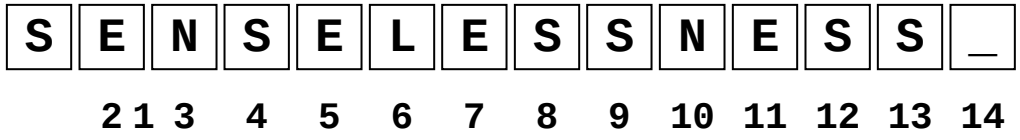
Using Φ and a bitvector we can avoid storing SA

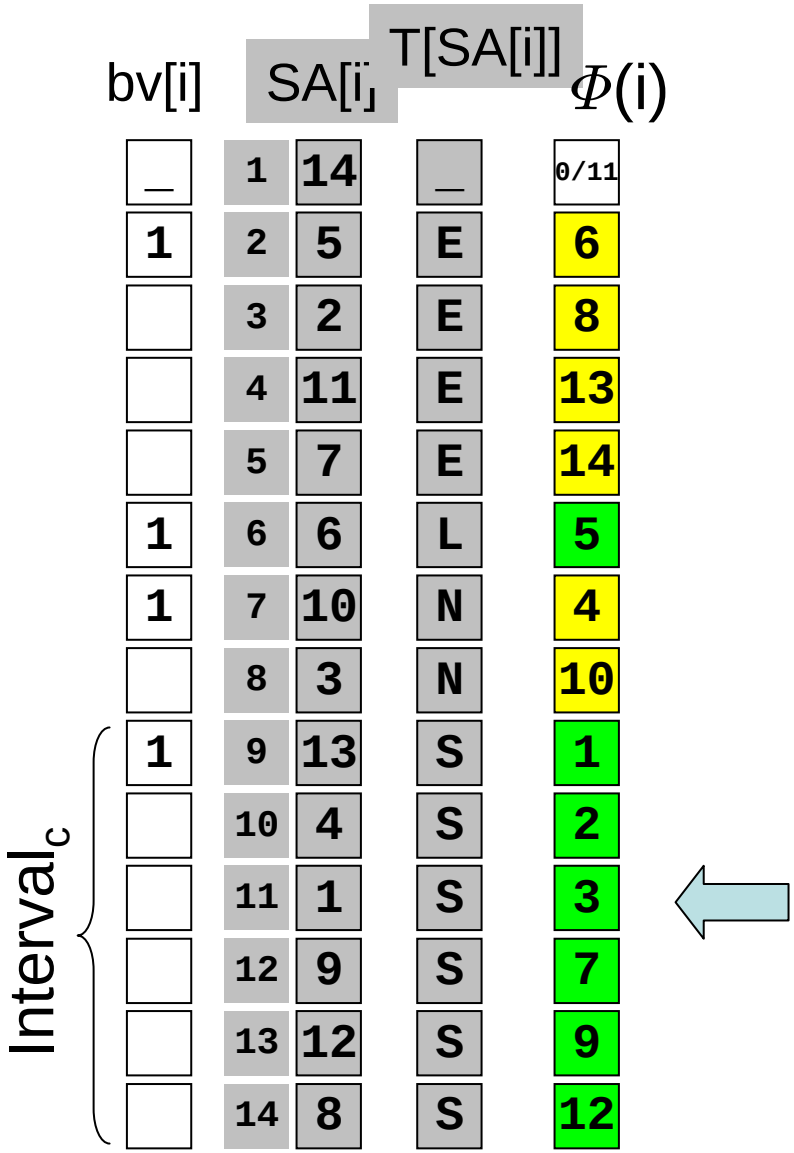
S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

bv[i]		SA[i]	T[SA[i]]	Φ(i)
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12



Using Φ and a bitvector we can avoid storing SA





Replacing the SA

- Record which SA[j]=1
- T[i] = c iff $\Phi^{(i)}(j) \in \text{Interval}_c$

Entering compression stuff...

A first simple approach

- Encode the bitvector σ log (n/σ) + o(n) bits
- Encode Φ in n log σ + o(n log σ) bits

- Compressed representation in
  n log σ + lower order bits
  [matches the lower bound seen before]

Another story is coming: succinctness

- Succinct data structures [Jacobson ’87]
- space log (# configurations) + lower order bits
- time O(1) per supported operation (if possible)

- Example:

  sets of n elements out of universe [1...m]

B(n,m) = $\lceil \log \binom{m}{n} \rceil$ bits

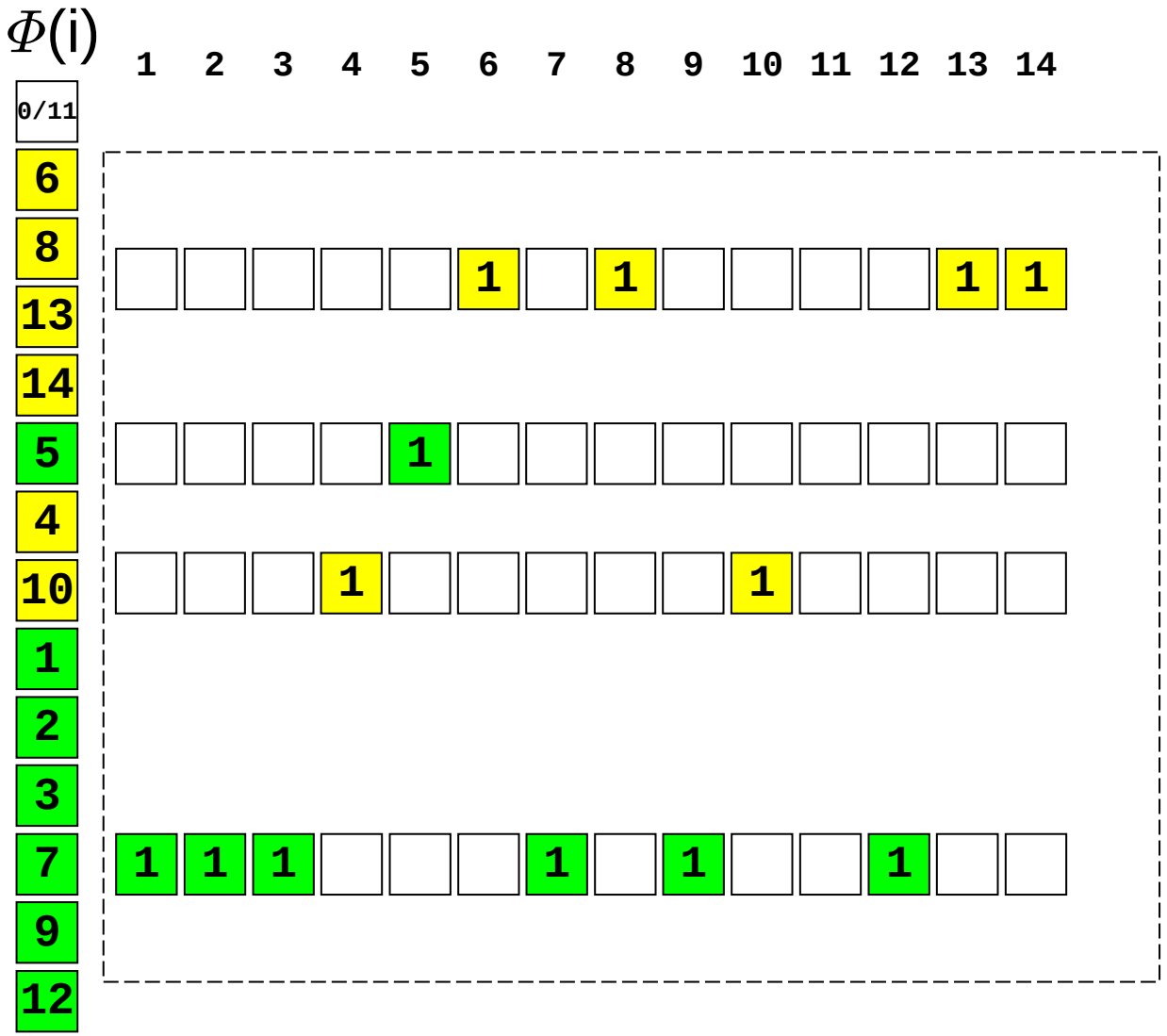
Fully indexable dictionaries (FID)

[Brodnik, Munro '99, Raman et al '02]

- Bitvector BV with n **1**s and m-n **0**s:
  - $\text{rank}_1(i)$ = #**1**s in BV[1...i]
  - $\text{select}_1(j)$ = position in BV of the jth **1**
  - same operations for **0**s
- Can solve also the precedessor problem

- space: $B(n,m) + o(m) \sim n \log(m/n) + o(m)$ bits
- time: O(1)

Using FIDs to store Φ function: Σ lists



Back to Φ

- Observation made when counting SA perms:

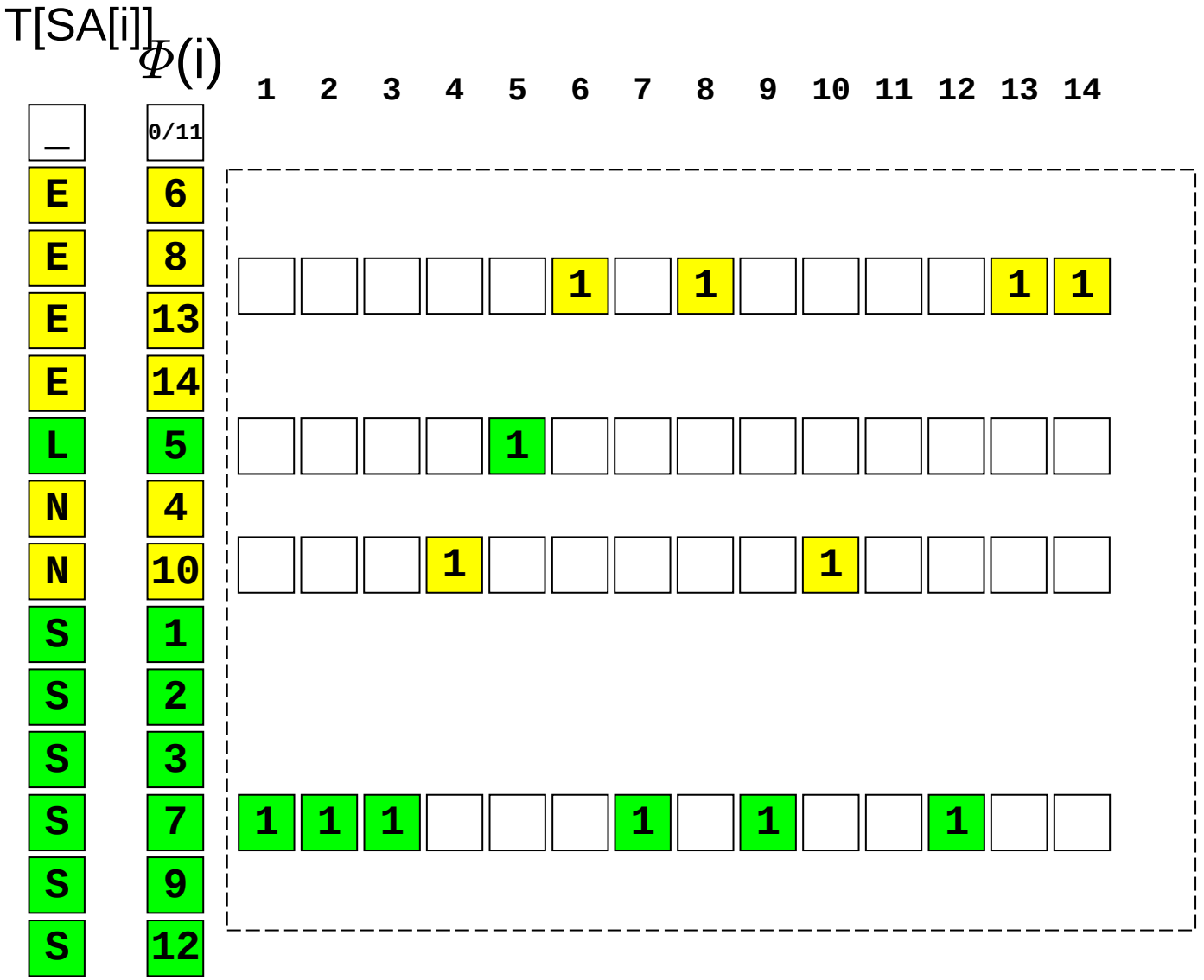
  T[SA[i]] = T[SA[i+1]] $\Rightarrow$ Φ(i) < Φ(i+1)

- Monotonicity: a maximal run of equal symbols

  T[SA[i]] = $\cdots$ = T[SA[j]] $\Rightarrow$ 1 $\leq$ Φ(i) < $\cdots$ < Φ(j) $\leq$ n

- At most σ runs, called Σ lists

Using FIDs to store Φ function: Σ lists



Storing Φ: first idea

- Concatenate all the σ bitvectors
- Build a single FID using

  B(n, n σ) + o(nσ) ~ n log σ + o(n σ) bits
- Problem: o(n σ) can be >> n log σ ...

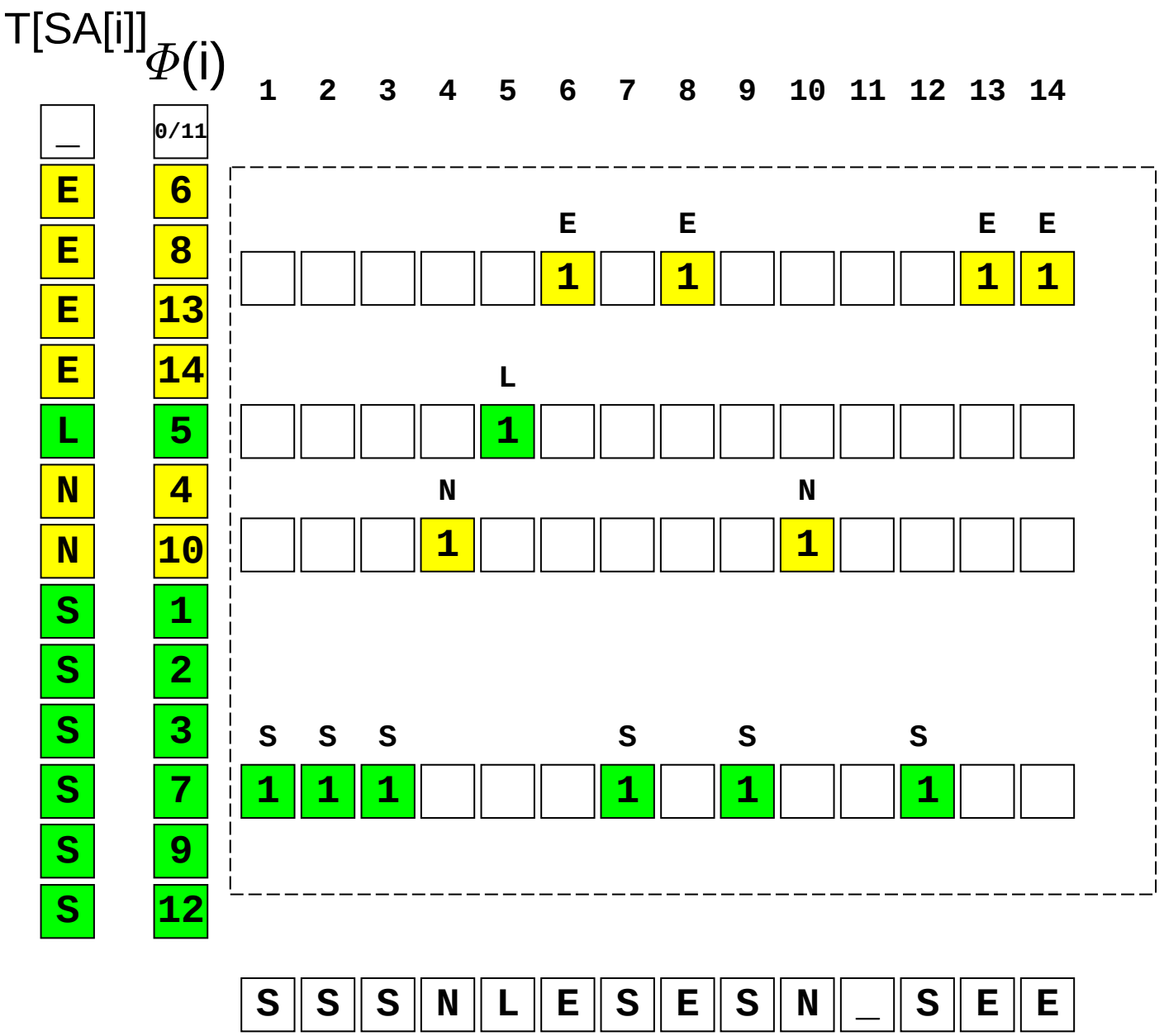
Better idea: Wavelet tree

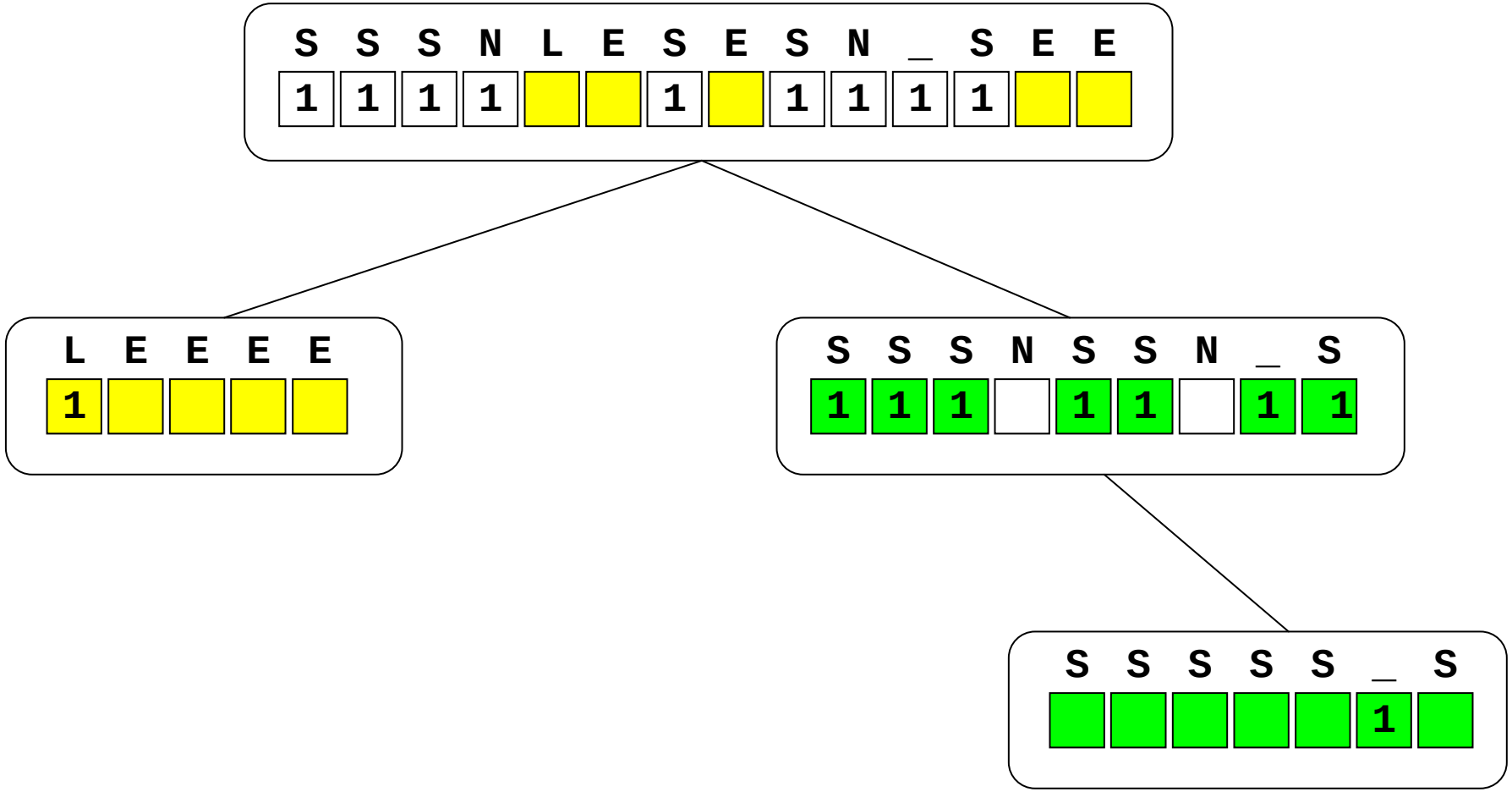
[G., Gupta, Vitter ’03]

- Extend a FID from bitvectors to strings S $\in \Sigma^n$:
  - $\text{rank}_c(i)$ = #cs in S[1...i]
  - $\text{select}_c(j)$ = position in V of the jth c in S
  - for any c $\in \Sigma$

- Space n log σ + o(n log σ) bits

- Time O(log σ)


T[SA[i]]
Φ(i)
1 2 3 4 5 6 7 8 9 10 11 12 13 14
_ 0/11
E 6
E 8
E 13
E 14
L 5
N 4
N 10
S 1
S 2
S 3
S 7
S 9
S 12
E E E E
1 1 1 1
L
1
N N
1 1
S S S S S S
1 1 1 1 1 1
S S S N L E S E S N _ S E E



Wavelet tree

Compressed suffix array (CSA)

G. & Vitter ’00, Sadakane ‘00

- Using additional (n/k) log n bits, we can obtain SA[i] in O(k) time using Φ. If k = $\omega(\log_\sigma n)$, this is o(n log σ) bits.

G., Gupta, Vitter ’03

- High order entropy

The whole story is long...

- Φ function and CSA [G., Vitter ’00]
- Avoiding to store the text [Sadakane ’00]
- High order entropy and link to BWT– FM-index [Ferragina, Manzini’00]
- Wavelet tree [G., Gupta, Vitter, ’03]
- Space-efficient constrcution [Hon, Sadakane, Wong ’03]
- Alphabet-friendly FM-index [Ferragina, Makinen, Manzini, Navarro ‘04]
- ... many other results.....nice survery paper [Makinen, Navarro]

Unsolved problems: 10 years old open problems

- CSA using n H_k + o(n log σ) bits and O(log$^{\epsilon}$) decompress time $\rightarrow$ O(1) time?
- As of now, CSA makes random access to memory:
  - can we retrieve p pointers from SA with O(p/B) transfers (B = blocksize)?
  - can we decompress t consecutive symbols from T with O(p/B) transfers?
- Can we add/remove a text to CSA in O(n) time?

- Technological transfer? [pizza-chili web site by Ferragina and Navarro]

THANKS!

THE END

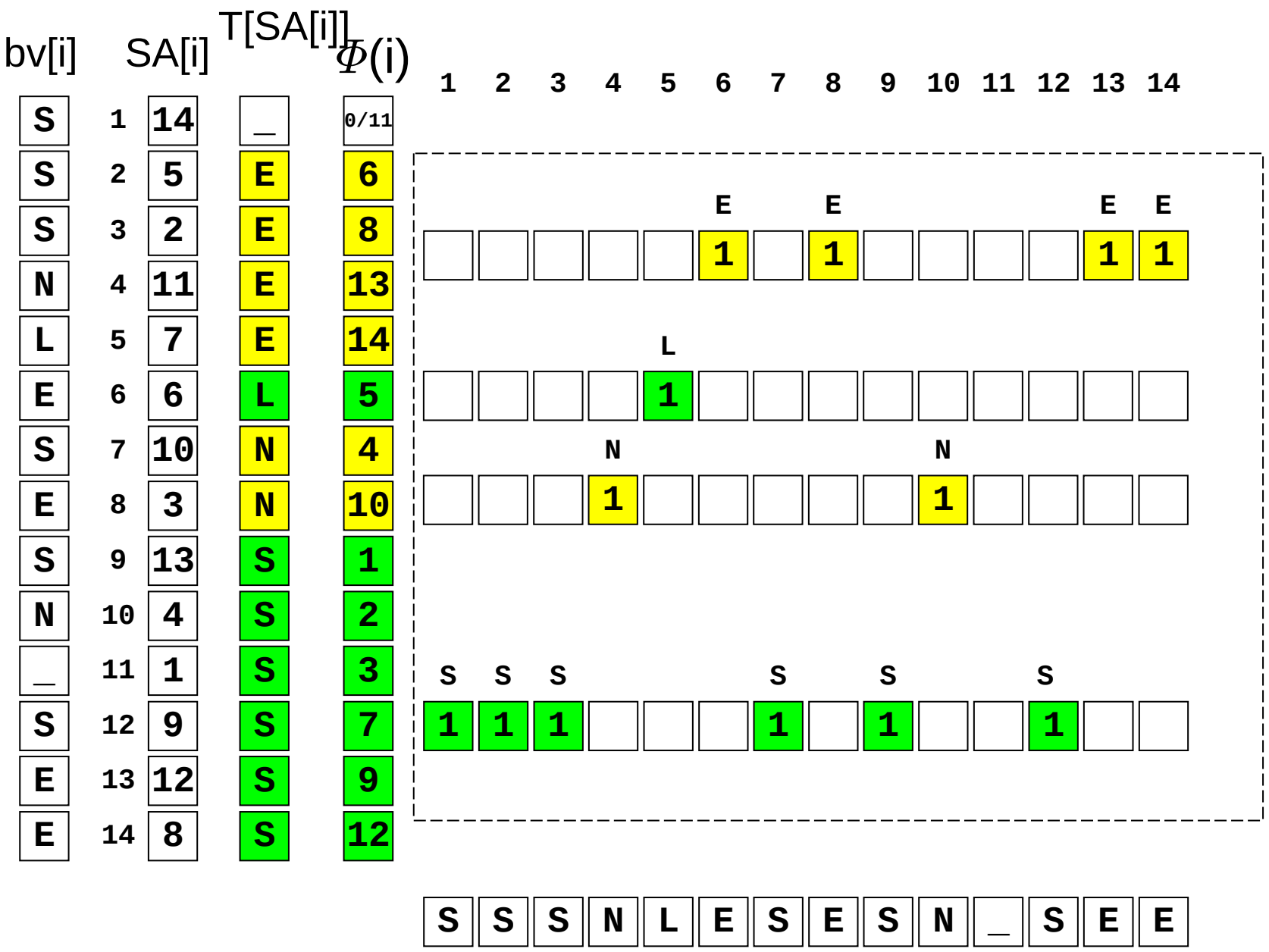
Eulerian numbers

Recursive enumeration of Eulerian numbers

a) $\left\langle {n \atop 0} \right\rangle = 1$

b) $\left\langle {n \atop d} \right\rangle = 0$ for $n \leq d$, and

c) $\left\langle {n \atop d} \right\rangle = (d+1)\left\langle {n-1 \atop d} \right\rangle + (n-d)\left\langle {n-1 \atop d-1} \right\rangle$


bv[i]
SA[i]
T[SA[i]]
Φ(i)
1 2 3 4 5 6 7 8 9 10 11 12 13 14
S 1 14 _ 0/11
S 2 5 E 6
S 3 2 E 8
N 4 11 E 13
L 5 7 E 14
E 6 6 L 5
S 7 10 N 4
E 8 3 N 10
S 9 13 S 1
N 10 4 S 2
_ 11 1 S 3
S 12 9 S 7
E 13 12 S 9
E 14 8 S 12
E E E E
1 1 1 1
L
1
N N
1 1
S S S S S S
1 1 1 1 1 1
S S S N L E S E S N _ S E E