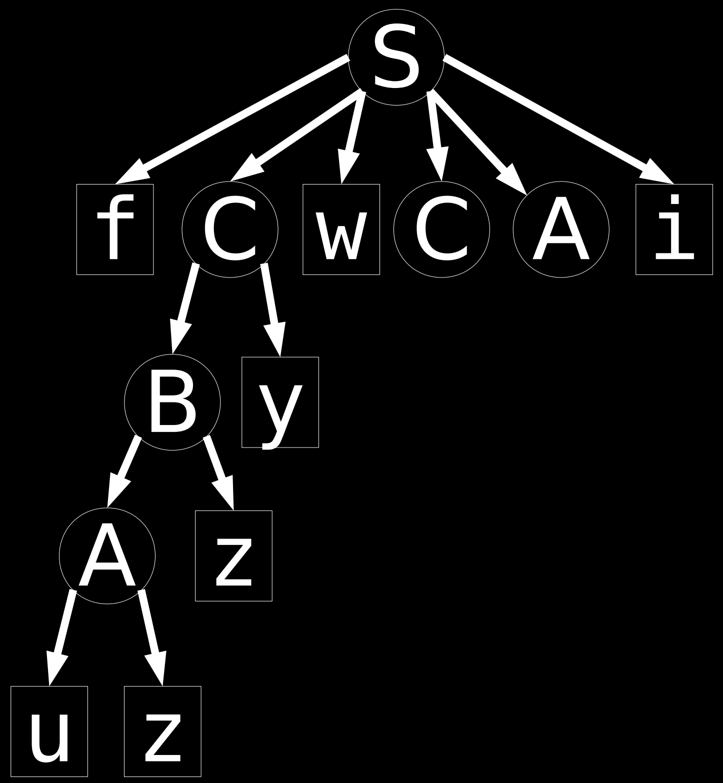
PSC 2020

Re-Pair in small space

Dominik Köppl
Tomohiro I
Isamu Furuya
Yoshimasa Takabatake
Kensuke Sakai
Keisuke Goto



grammar compression

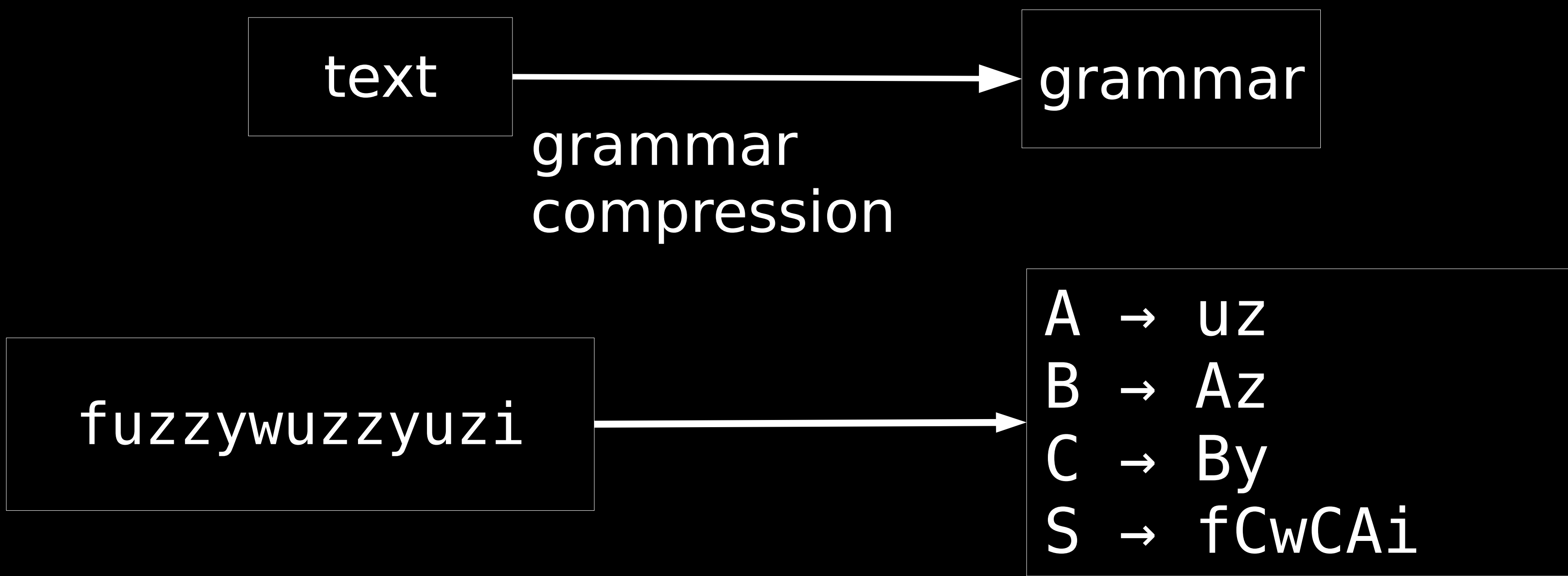
text

grammar compression

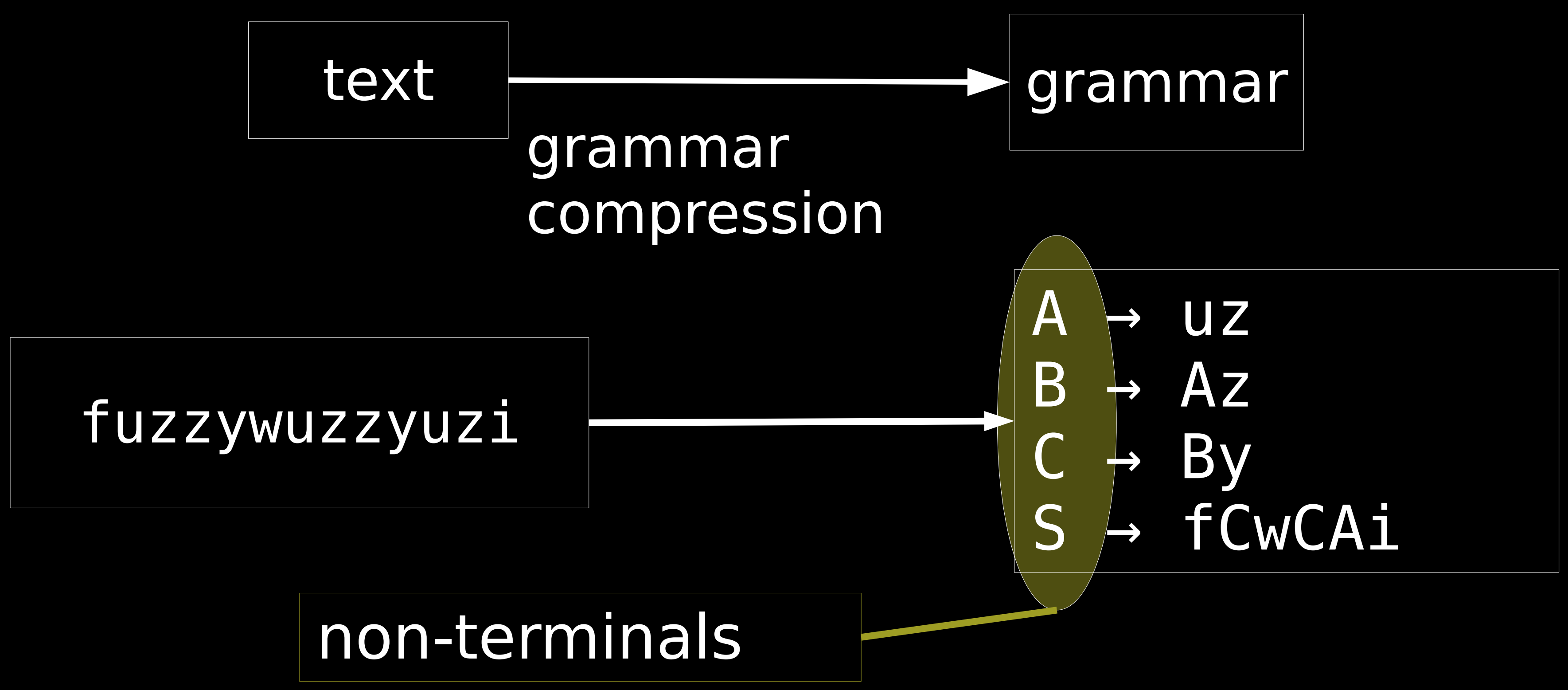
grammar compression

`fuzzywuzzyyuzi`

grammar compression



grammar compression



restore text

```
A → uz
B → Az
C → By
S → fCwCAi
```

restore text

```
A → uz
B → Az
C → By
S → fCwCAi
```

start symbol

restore text

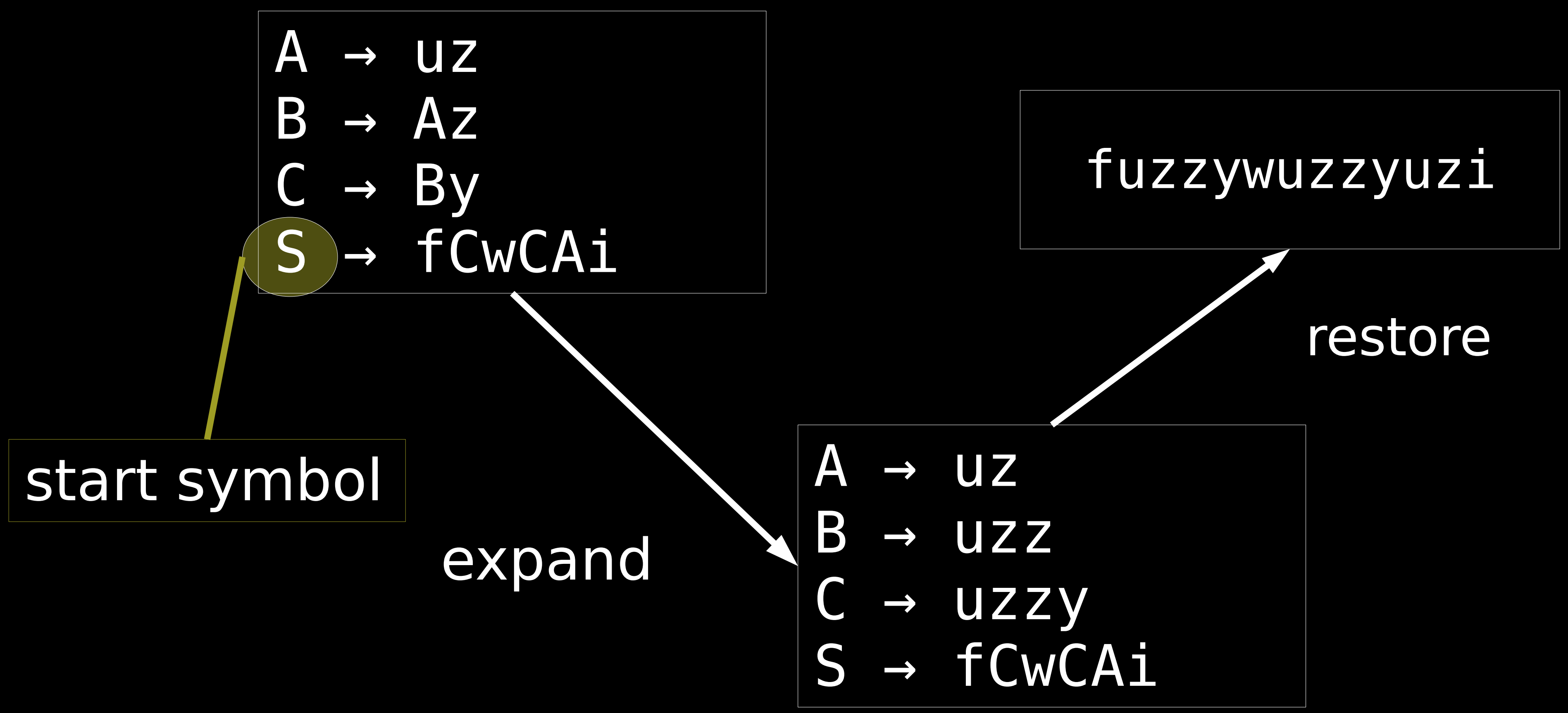
A → uz
B → Az
C → By
S → fCwCAi

start symbol

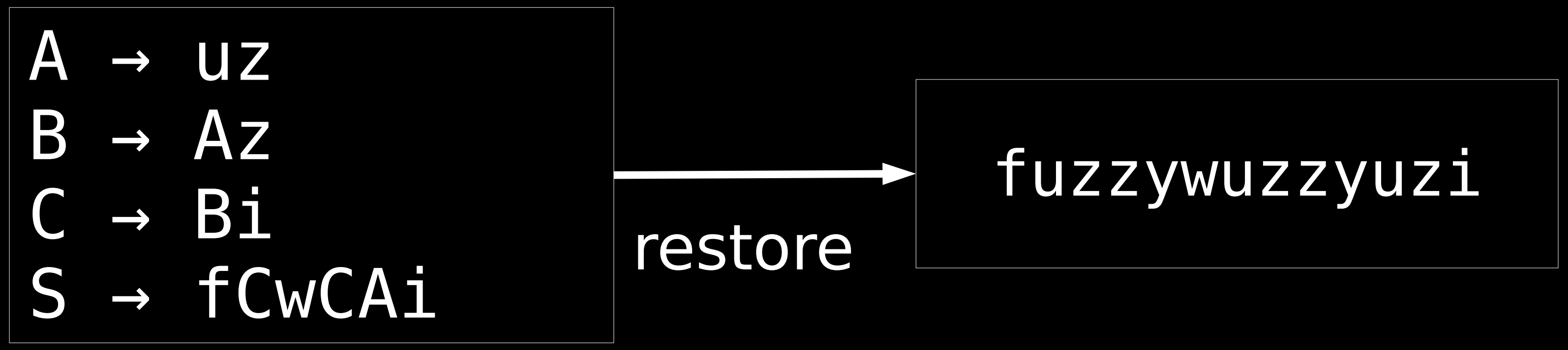
expand

A → uz
B → uzz
C → uzzy
S → fCwCAi

restore text



SLP: Straight Line Program



SLP: Straight Line Program

- only one start symbol

```
A → uz
B → Az
C → Bi
S → fCwCAi
```

restore →

```
fuzzywuzzyuzi
```

SLP: Straight Line Program

- only one start symbol
- right hand side of each rule has length two (except start symbol)

S
↓
C
↓
B
↓
A

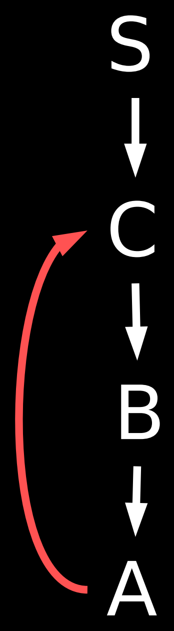
```
A → uz
B → Az
C → Bi
S → fCwCAi
```

restore →

```
fuzzywuzzyyuzi
```

SLP: Straight Line Program

- only one start symbol
- right hand side of each rule has length two (except start symbol)
- no cycles

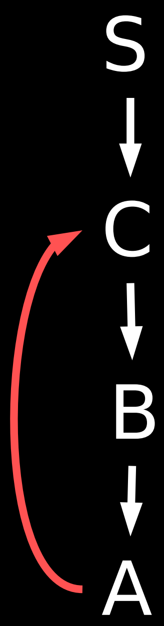


A → Cz
B → Az
C → Bi
S → fCwCAi

restore → fuzzywuzzyuzi

SLP: Straight Line Program

- only one start symbol
- right hand side of each rule has length two (except start symbol)
- no cycles
- every non-terminal has exactly one rule



A → Cz
B → Az
C → Bi
S → fCwCAi

restore → fuzzywuzzyuzi

A → zz

bigram

given : text T

- bigram : pair of characters
- bigram frequency: number of *non-overlapping* bigrams in T
- $\#(b)$:= frequency of bigram b

$$T = \texttt{fuzzywuzzzyuzi}$$

bigram

given : text T

- bigram : pair of characters
- bigram frequency: number of *non-overlapping* bigrams in T
- $\#(b)$:= frequency of bigram b

$T =$ fuzzywuzzzyuzi

$\#(zz) = 2$

$\#(fu) = 1$

Re-Pair

- is an SLP

fuzzywuzzyuzi

Re-Pair

#(uz) = 3

- is an SLP
- takes bigram with highest frequency and replaces it with new non-terminal

fuzzywuzzyuzi

Re-Pair

$\#(uz) = 3$

$A \rightarrow uz$

- is an SLP
- takes bigram with highest frequency and replaces it with new non-terminal

```
fuzzywuzzyuzi
      ↓
fA_zywA_zyA_i
```

Re-Pair

- is an SLP
- takes bigram with highest frequency and replaces it with new non-terminal
- recurses

#(uz) = 3

A → uz

fuzzywuzzyuzi

↓

fA_zywA_zyA_i

T_1=fA_zywA_zyA_i

shrink text

T_1=fA_zywA_zyA_i

T_1=fAzywAzyAi $\#(\text{Az}) = 2$

shrink text

T_1=fA_zywA_zyA_i

T_1=fAzywAzyAi $\#(\mathrm{Az}) = 2$

B → Az

T_2=fB_ywB_yAi

shrink text

T_1=fA_zywA_zyA_i

T_1=fAzywAzyAi $\#(\text{Az}) = 2$

B → Az

T_2=fB_ywB_yAi

T_2=fBywByAi $\#(\text{By}) = 2$

shrink text

T_1=fA_zywA_zyA_i

T_1=fAzywAzyAi   #(Az) = 2

B → Az

T_2=fB_ywB_yAi

T_2=fBywByAi   #(By) = 2

C → By

T_3=fC_wC_Ai

shrink text

T_1=fA_zywA_zyA_i

T_1=fAzywAzyAi

$\#(\mathrm{Az}) = 2$

$\mathrm{B} \rightarrow \mathrm{Az}$

T_2=fB_ywB_yAi

T_2=fBywByAi

$\#(\mathrm{By}) = 2$

$\mathrm{C} \rightarrow \mathrm{By}$

T_3=fC_wC_Ai

T_3=fCwCAi

shrink text

T_1=fA_zywA_zyA_i

T_1=fAzywAzyAi

#(Az) = 2

B → Az

T_2=fB_ywB_yAi

T_2=fBywByAi

#(By) = 2

C → By

T_3=fC_wC_Ai

T_3=fCwCAi

terminate when all bigram frequencies are at most 1

A → uz
B → Az
C → By
S → fCwCAi

#(Az) = 2
B → Az

#(By) = 2
C → By

T_3=fCwCAi

final string T_3 becomes start symbol

known algorithms

Larson, Moffat'00:

$5n + 4\sigma^2 + 4\pi + n^{1/2}$ words

Bille+'17:

$\varepsilon n + n^{1/2}$ words

- n: text length
- σ: alphabet size
- π: # non-terminals
- $\varepsilon > 0$ constant

space is additional to the *rewritable* input text of n words

both run in expected linear time

our algorithms

target space:

- n lg (σ+π) + O(lg n) bits
- input as rewritable part included

- n: text length
- σ: alphabet size
- π: # non-terminals

in O(n^3) time

find bigram b with highest frequency:

- given $b = T[i]\,T[i+1]$
- $\#(b) = \#(T[i]\,T[i+1])$

  $= \max_{1 \leqslant j \leqslant n} \#(T[j]\,T[j+1])$
- can find b in O(n^2) time

in O(n^3) time

- can find b in O(n^2) time
- replace all occurrences of b in T within O(n) time
- number of all distinct bigrams is at most n ($\pi \leq n$)

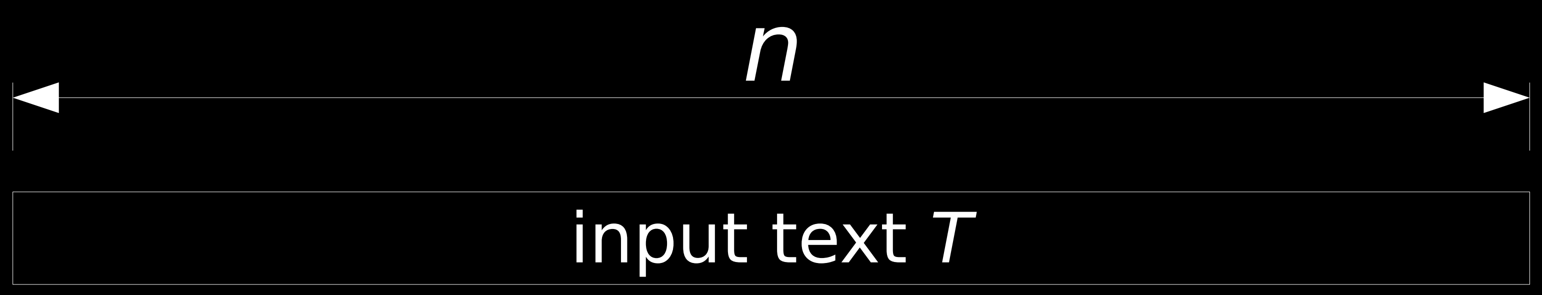
$\Rightarrow$ O(πn^2) = O(n^3) time

if $\sigma+\pi$ = O(1)

- $\sigma+\pi$: # symbols that can appear in T at any time
- if $\sigma+\pi$ is constant:
  - maintain frequencies of all bigrams in $O((\sigma+\pi)^2) = O(1)$ space in a binary search tree
  - all operations on the tree: O(1) time
  - total time: $O(\pi n) = O(n)$
- what if $\sigma+\pi = \omega(1)$, such as $\sigma+\pi = \Theta(n)$?

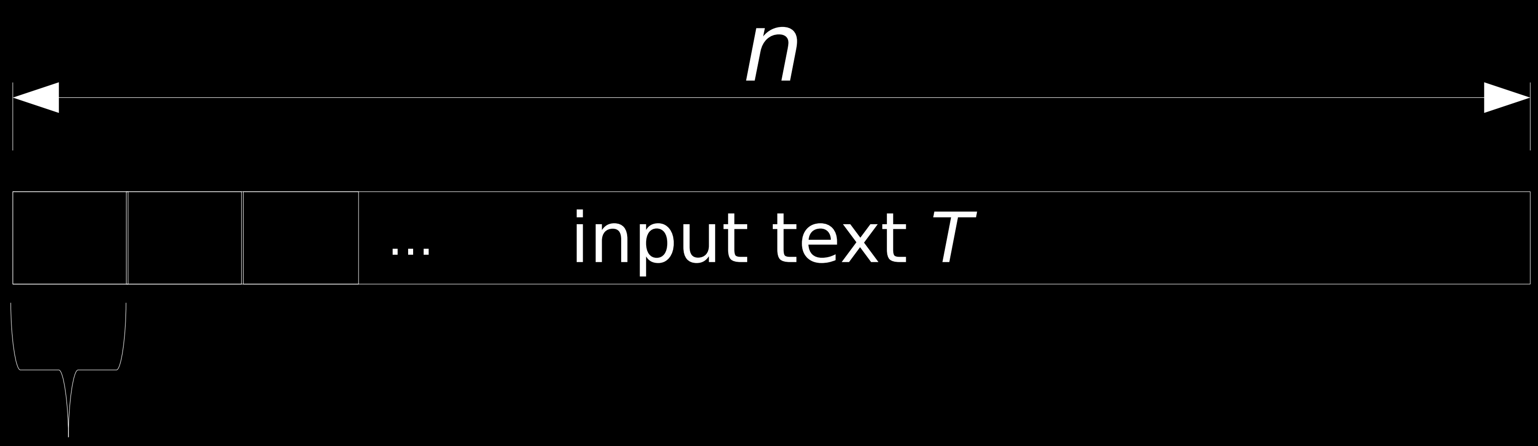
general approach

aim in this talk: $O(n^2)$ time



general approach

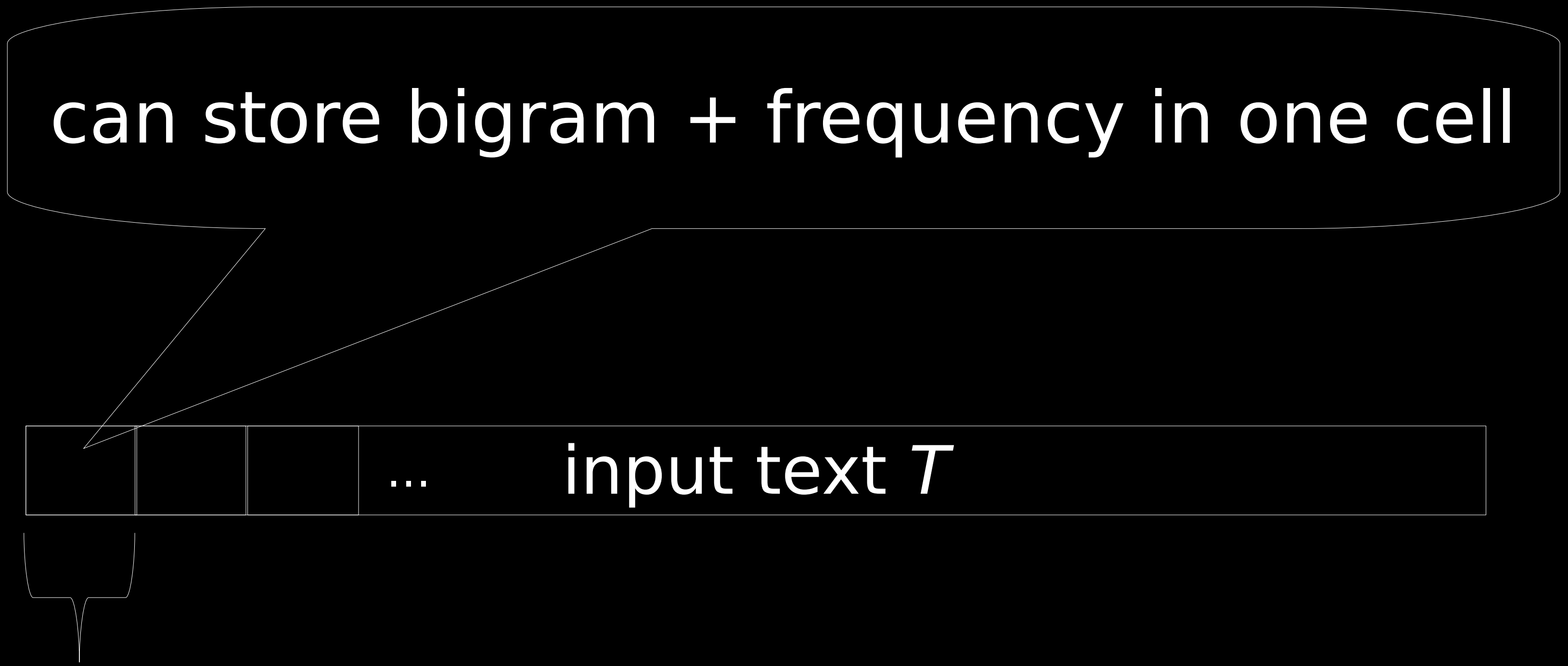
aim in this talk: $O(n^2)$ time



one cell takes O(1) words

(for lg σ bits cells : consult the paper)

assumption



one cell takes O(1) words

(for lg σ bits cells : consult the paper)

idea

- bigram replacement frees up space

  ⇒ can maintain more frequencies
- for that: divide algorithm into rounds
- at beginning of k-th round :
  - f_k : number of frequencies we can maintain
  - task: compute the frequencies of the f_k most frequent bigrams

f_k

```
#(zb) = 33
#(wy) = 33
#(cx) = 31
    …
#(ao) = 21
```

T_i	f_k

k-th round, number of rules: *i*

f_k

```
#(zb) = 33
#(wy) = 33
#(cx) = 31
     ...
#(ao) = 21
```

maintain frequencies

T_i	f_k

k-th round, number of rules: *i*

the most frequent bigram
among those we did not store

#(da) = 20

f_k:

#(zb) = 33
#(wy) = 33
#(cx) = 31
...
#(ao) = 21

maintain frequencies

T_i	f_k

k-th round, number of rules: i

the most frequent bigram
among those we did not store

#(da) = 20

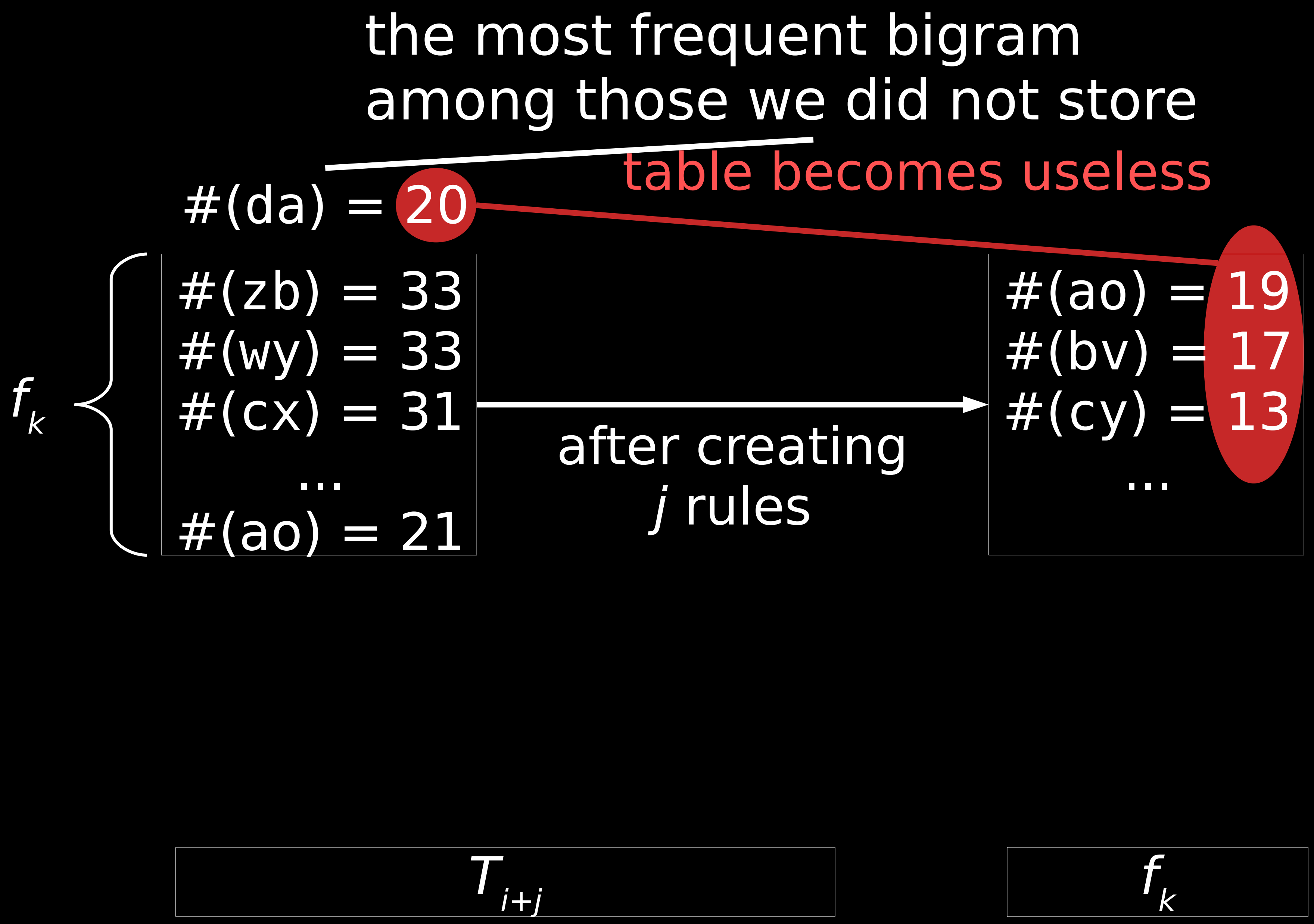
f_k {
#(zb) = 33
#(wy) = 33
#(cx) = 31
...
#(ao) = 21

→ after creating j rules →

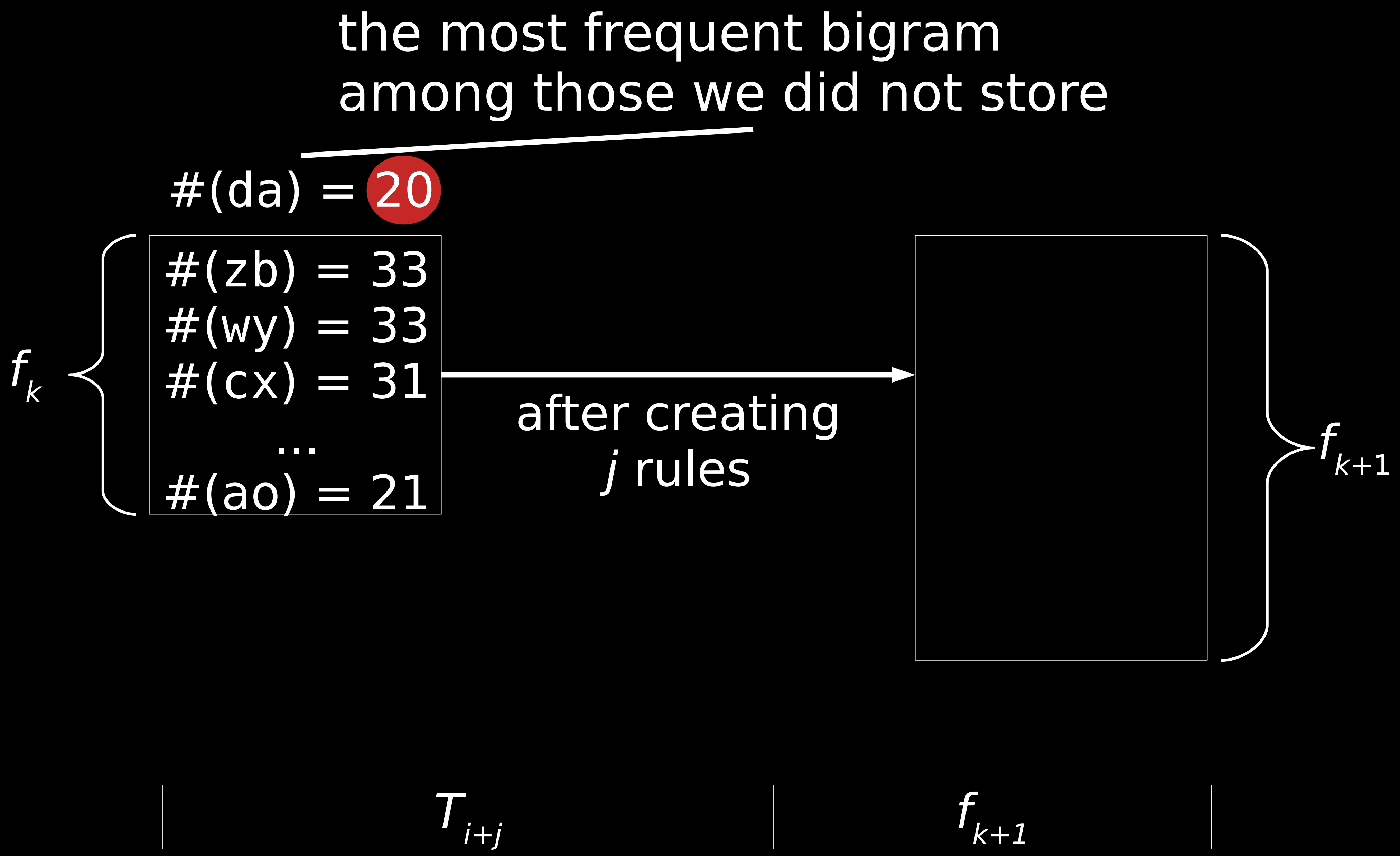
#(ao) = 19
#(bv) = 17
#(cy) = 13
...

T_{i+j}	f_k

k-th round, number of rules: $i+j$



k-th round, number of rules: $i+j$



k+1-th round, number of rules : $i+j$

start of algorithm

- first round: $f_1 = O(1) =$ constant
- maintain the f_1 most frequent bigrams
- replace the most frequent bigram
- update the maintained frequencies

why updating?

fuzzywuzzyuzi

why updating?

fuzzywuzzyyuzi

#(uz) = 3
#(zz) = 2
#(zy) = 2

why updating?

fuzzywuzzyuzi

#(uz) = 3
#(zz) = 2
#(zy) = 2

why updating?

A → uz

fuzzywuzzyuzi

↓

fAzywAzyAi

#(uz) = 3
#(zz) = 2
#(zy) = 2

why updating?

A → uz

fuzzywuzzyuzi

↓

fAzywAzyAi

~~#(uz) = 3~~

#(zz) = 0

#(zy) = 2

#(Az) = 2

- for each replaced occurrence:
  - the frequencies of at most two bigrams are decremented by one

- for each replaced occurrence:
  - the frequencies of at most two bigrams are decremented by one

#(fu) = 1
#(zz) = 2

fuzz

- for each replaced occurrence:
  - the frequencies of at most two bigrams are decremented by one

#(fu) = ~~1~~ 0

#(zz) = ~~2~~ 1

fuzz

↓

fA_z

- for each replaced occurrence:
  - the frequencies of at most two bigrams are decremented by one

⇒ at end of k-th round:

$$f_{k+1} \geq f_k + \tfrac{1}{2} f_k$$

#(fu) = ~~1~~ 0

#(zz) = ~~2~~ 1

fuzz

↓

fA_z

- for each replaced occurrence:
  - the frequencies of at most two bigrams are decremented by one

#(fu) = ~~1~~ 0

#(zz) = ~~2~~ 1

fuzz

↓

fA_z

⇒ at end of k-th round:

$$f_{k+1} \ge f_k + \tfrac{1}{2} f_k$$

$$\Leftrightarrow f_{k+1} \ge (1.5)^k f_1$$

- for large $k = O(\lg n)$ $f_k = \Theta(n)$

can maintain a constant fraction of all frequencies !

- for each replaced occurrence:
  - the frequencies of at most two bigrams are decremented by one

#(fu) = ~~1~~ 0
#(zz) = ~~2~~ 1

fuzz
↓
fA_z

⇒ at end of *k*-th round:

$f_{k+1} \geq f_k + ½ f_k$

$\Leftrightarrow f_{k+1} \geq (1.5)^k f_1$

- for large $k = O(\lg n)$ $f_k = \Theta(n)$

can maintain a constant fraction of all frequencies !

⇒ there are $O(\lg n)$ rounds

time: summary

- computing frequencies of f_k bigrams:

  $O(n^2)$ time + sort(f_k) time

  $= O(n^2)$ time (since $f_k \leqslant n$)

- compute frequencies $O(\lg n)$ times

  $\Rightarrow O(n^2 \lg n)$ time

- how do we get $O(n^2)$ time?

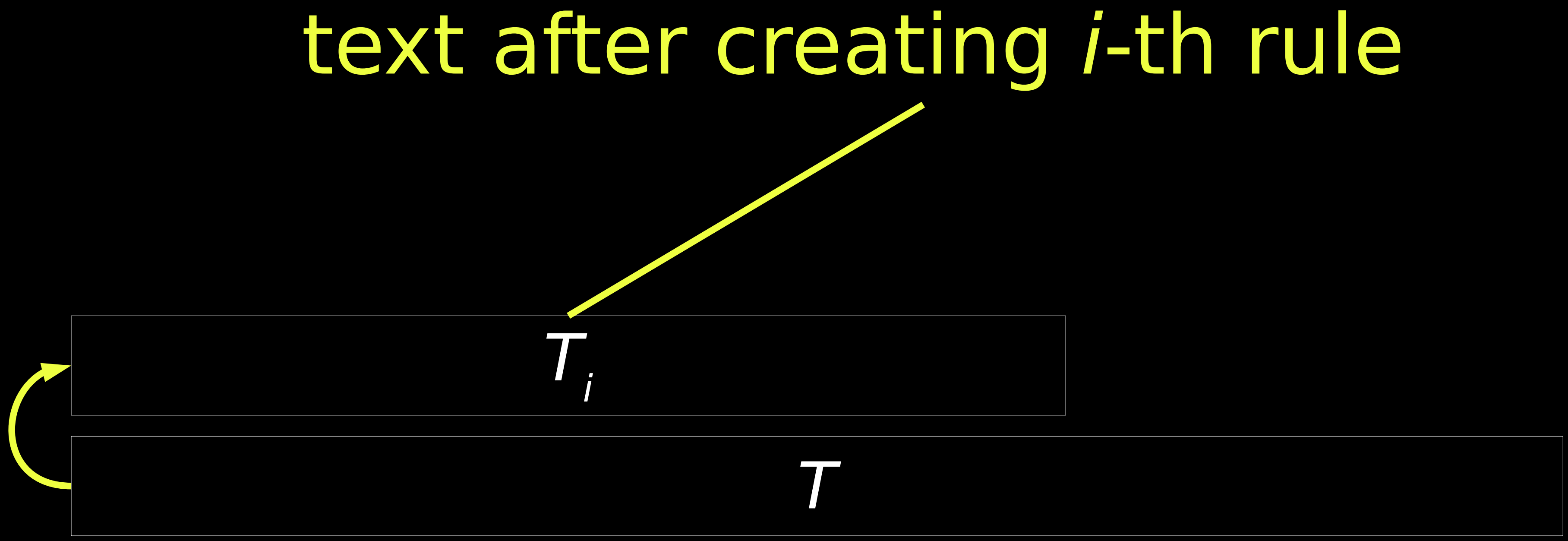
in-place sorting

- f_k : length of input integer array
- result:
  - O(f_k) space (including input)
  - O(f_k lg f_k) time

[Williams'64: heapsort]

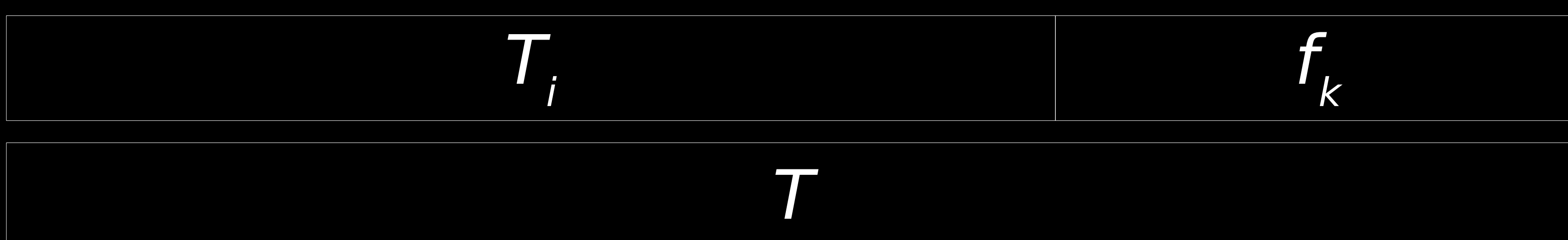
O(n^2) time

- speed up frequency computation



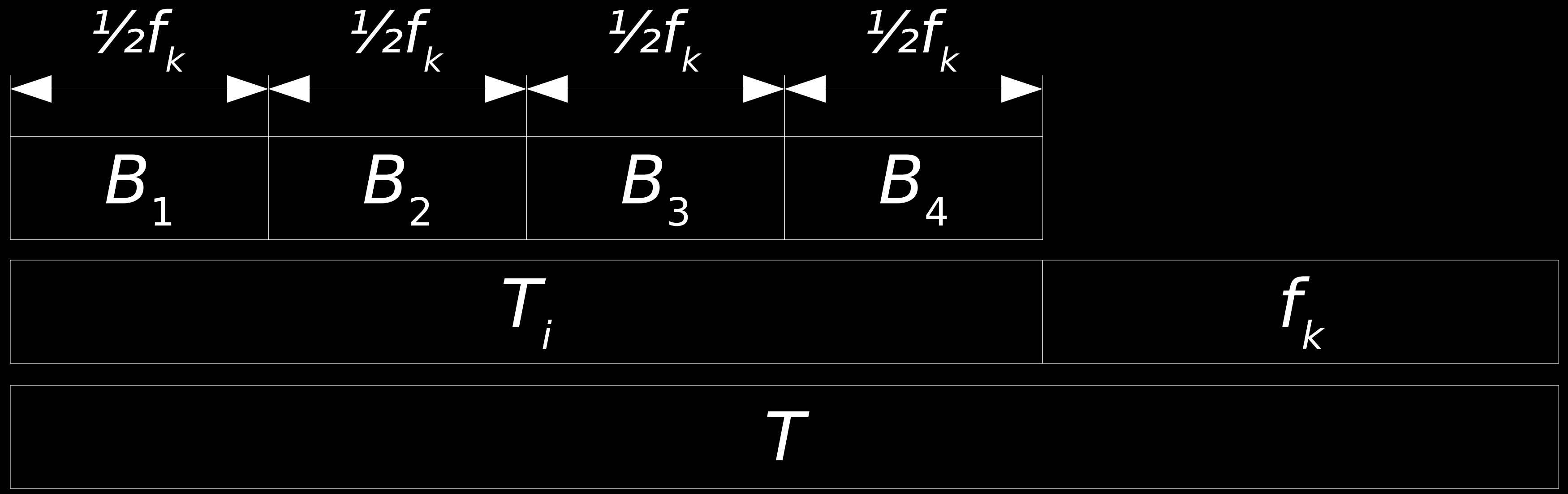
O(n^2) time

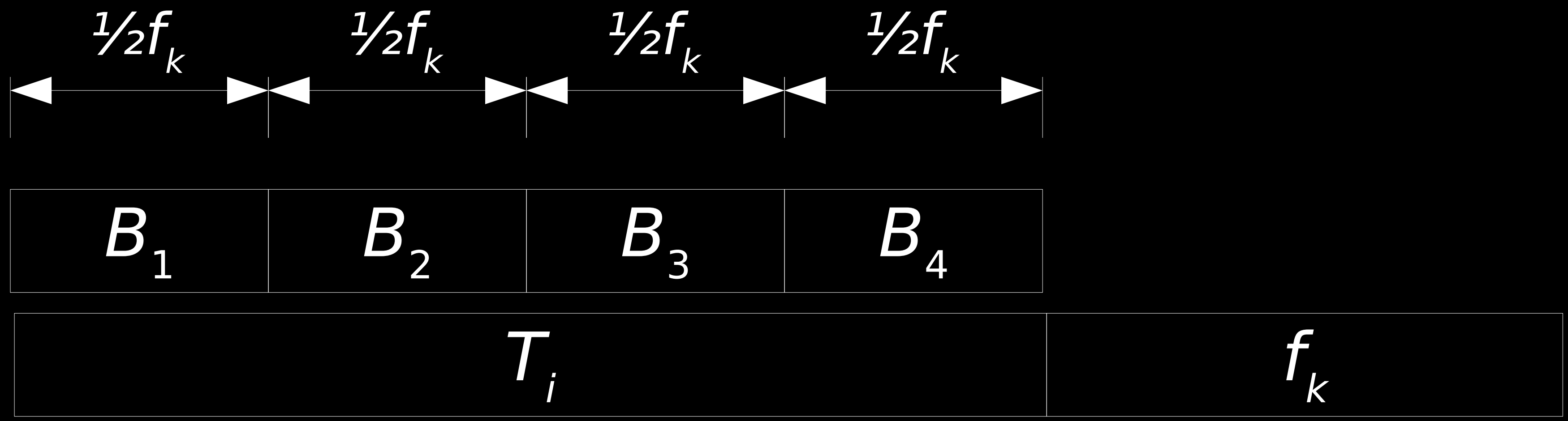
- speed up frequency computation
- have f_k space

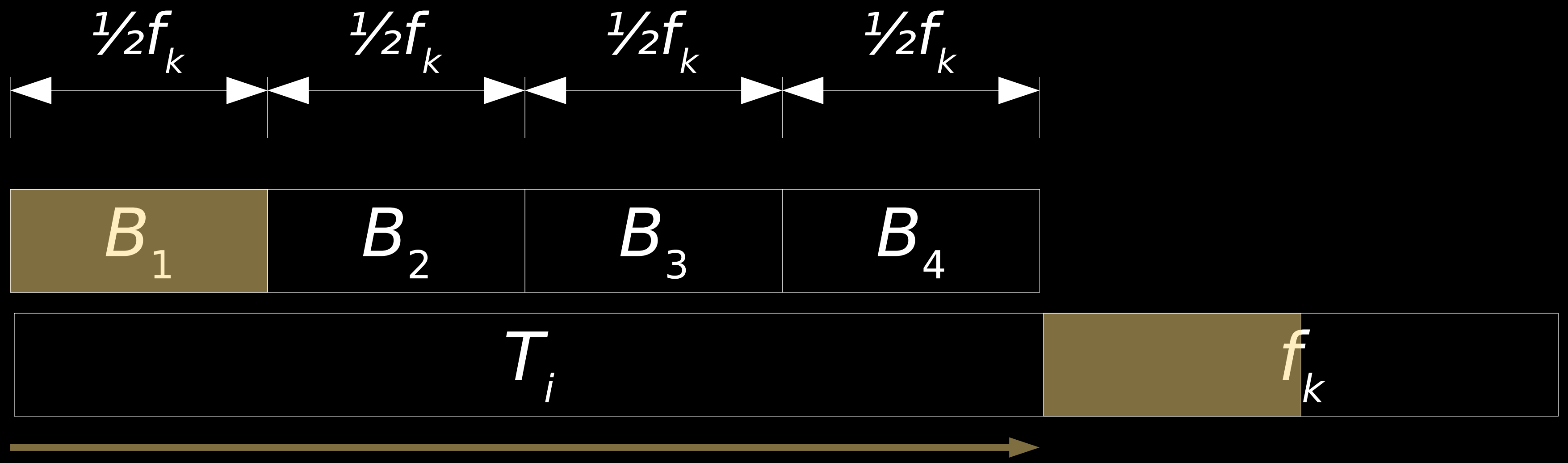


O(n^2) time

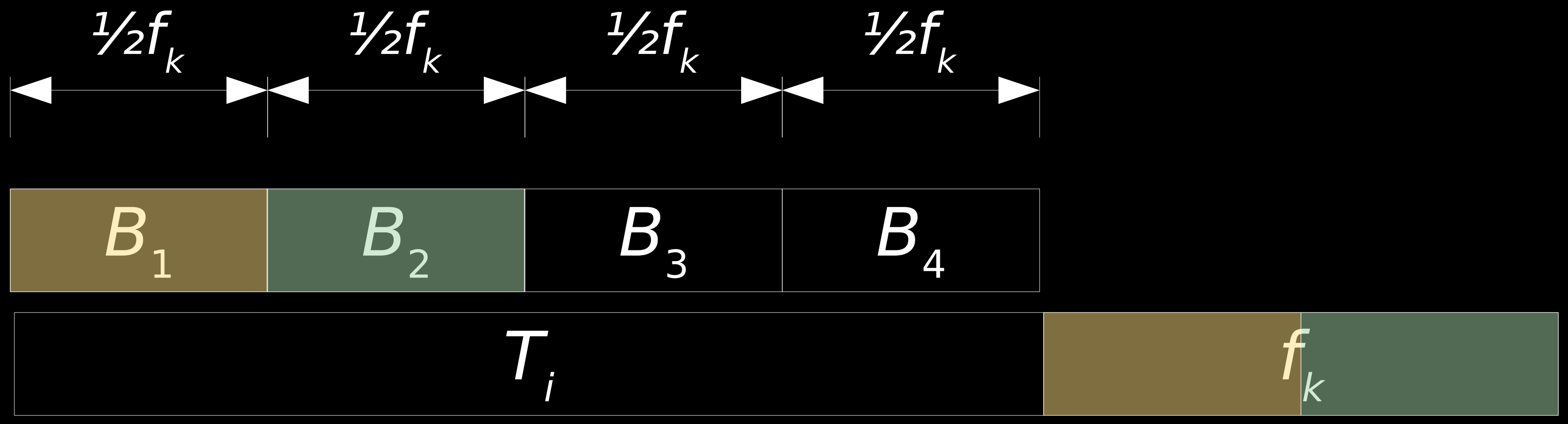
- speed up frequency computation
- have f_k space
- divide in blocks B_j with $|B_j| = ½f_k$




½f k
½f k
½f k
½f k
B 1
B 2
B 3
B 4
T i
f k

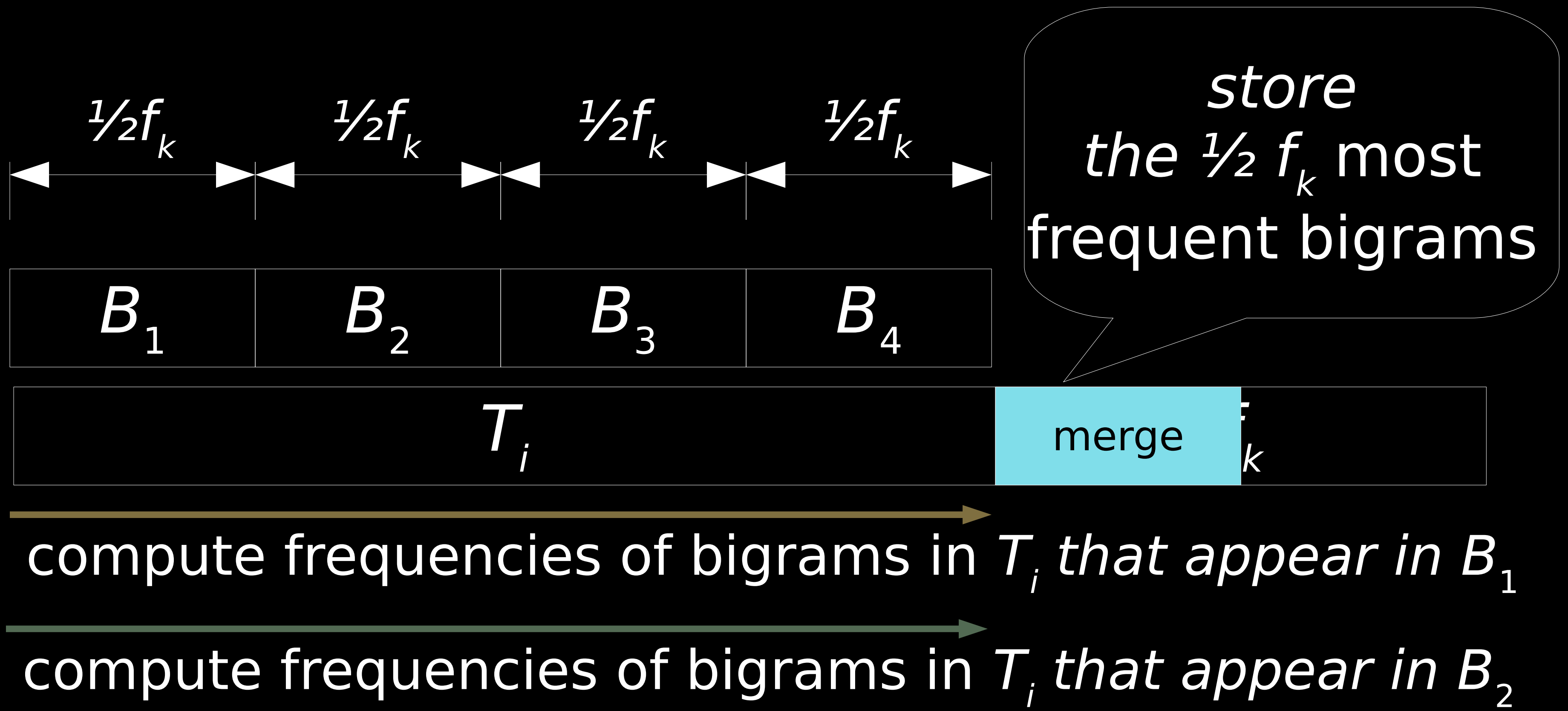


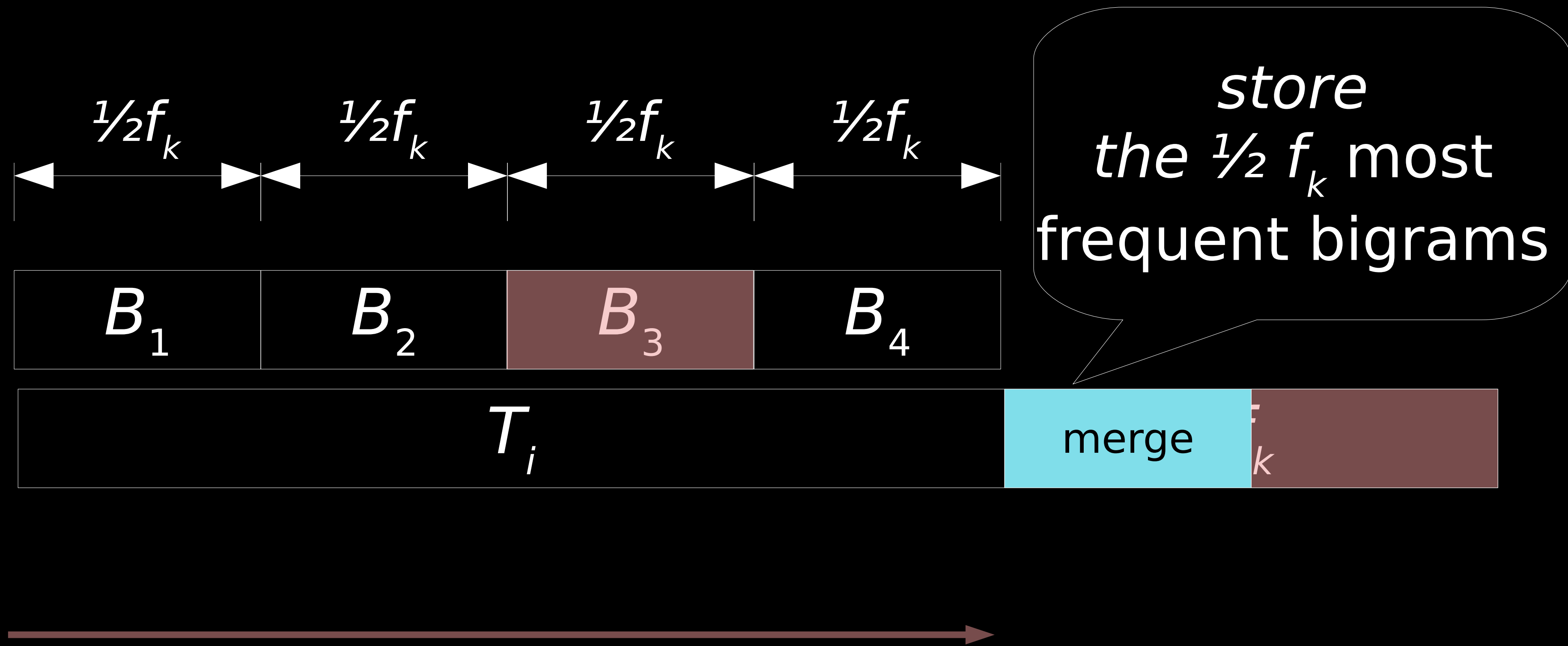
compute frequencies of bigrams in T_i *that appear in* B_1



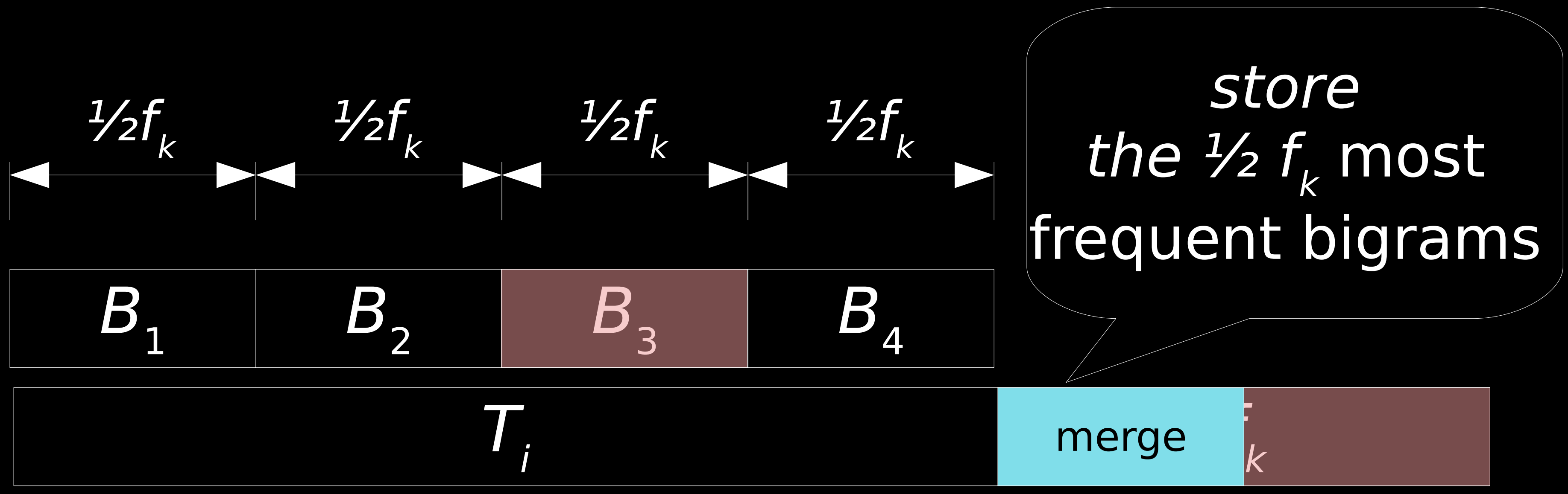
compute frequencies of bigrams in T_i *that appear in* B_1

compute frequencies of bigrams in T_i *that appear in* B_2


½f k
½f k
½f k
½f k
store
the ½ f k most
frequent bigrams
B 1
B 2
B 3
B 4
T i
merge
compute frequencies of bigrams in T i that appear in B 1
compute frequencies of bigrams in T i that appear in B 2

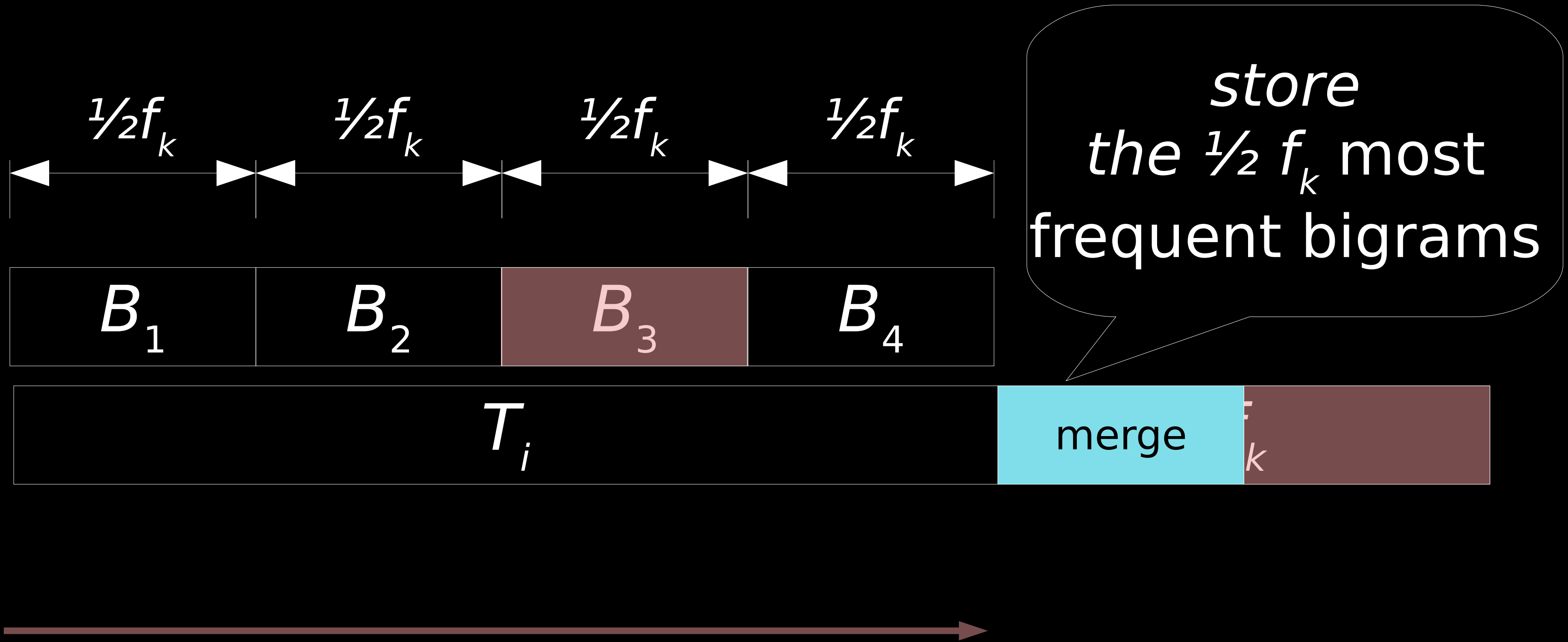


compute frequencies of bigrams in T_i *that appear in* B_3



compute frequencies of bigrams in T_i *that appear in* B_3

- \# merge = # B_j - 1 ⩽ n / f_k , $|T_i| \leqslant |T| = n$



compute frequencies of bigrams in T_i *that appear in* B_3

- # merge = # B_j - 1 ⩽ n / f_k , $|T_i|$ ⩽ $|T|$ = n
- time for bigrams in B_j : O(n lg f_k) (binary search)
- time for each merge: O(f_k lg f_k)
- total time: O((n (n+f_k) lg f_k)/ f_k) = O((n^2 lg f_k)/ f_k)

total time

$$\sum_{k=1}^{O(\lg n)} \frac{n^2}{f_k} \lg f_k = O\left(n^2 \sum_{k=1}^{\lg n} \frac{k}{1.5^k} \right) = O(n^2)$$

total time

have at most $O(\lg n)$ rounds

$O(\lg n)$

$$\sum_{k=1}^{O(\lg n)} \frac{n^2}{f_k} \lg f_k = O\left(n^2 \sum_{k=1}^{\lg n} \frac{k}{1.5^k} \right) = O(n^2)$$

total time

have at most $O(\lg n)$ rounds

$O(\lg n)$

$$\sum_{k=1}^{O(\lg n)} \frac{n^2}{f_k} \lg f_k = O\left(n^2 \sum_{k=1}^{\lg n} \frac{k}{1.5^k} \right) = O(n^2)$$

$f_k = 1.5^{k-1} f_1 = O(1.5^k)$

total time

have at most $O(\lg n)$ rounds

$\Rightarrow \lg f_k = O(k)$

$$\sum_{k=1}^{O(\lg n)} \frac{n^2}{f_k} \lg f_k = O\left(n^2 \sum_{k=1}^{\lg n} \frac{k}{1.5^k} \right) = O(n^2)$$

$f_k = 1.5^{k-1} f_1 = O(1.5^k)$

total time

have at most $O(\lg n)$ rounds

$\Rightarrow \lg f_k = O(k)$

$$\sum_{k=1}^{O(\lg n)} \frac{n^2}{f_k} \lg f_k = O\left(n^2 \sum_{k=1}^{\lg n} \frac{k}{1.5^k} \right) = O(n^2)$$

$f_k = 1.5^{k-1} f_1 = O(1.5^k)$

can we get $o(n^2)$ time ?

bit-parallel algorithm

- machine word size: $\Theta(\lg n)$ bits
- popcount: $O(\lg \lg \lg n)$ time per word
- total time:

$$O(\underbrace{n^2}_{\text{original algorithm}} \underbrace{\lg \log_\tau n}_{\text{penalty}} \lg \lg \lg n \,/\, \underbrace{\log_\tau n}_{\text{word packing}})$$

where $\tau := \sigma + \pi$: # symbols

$\Rightarrow o(n^2)$ time for $\tau = O(\text{polylog } n)$

arxiv paper

additional content:

- parallel computation
- external memory computation

  ... in-place or in small space

summary

- can compute Re-Pair in-place
  - $O(n^3)$ time : trivial
  - $O(n^2)$ time
    - in-place sorting
    - batch computing frequencies
    - assumed that $\sigma = \Theta(n)$
- general σ: need $\max(n/c \lg n, n \lg \tau) + O(\lg n)$ bits for $c > 1$
- future work:
  - $n \lg \tau + O(\lg n)$ bits
  - $o(n^2 \lg n)$ time and τ between $\omega(1)$ and $o(n)$?

thanks for listening - any questions are welcome!