# In-Place Re-Pair

クップル ドミニク　( 九州大学 )
井 智弘　( 九州工業大学 )
古谷 勇　( 北海道大学 )
高畠 嘉将　( 九州工業大学 )
酒井 健輔　( 九州工業大学 )
後藤 啓介　( 富士通研究所 )

LA Symposium 夏 '19
令和元年 7 月 31 日 ~8 月2日

# 文法圧縮

テキスト

# 文法圧縮

# 文法圧縮

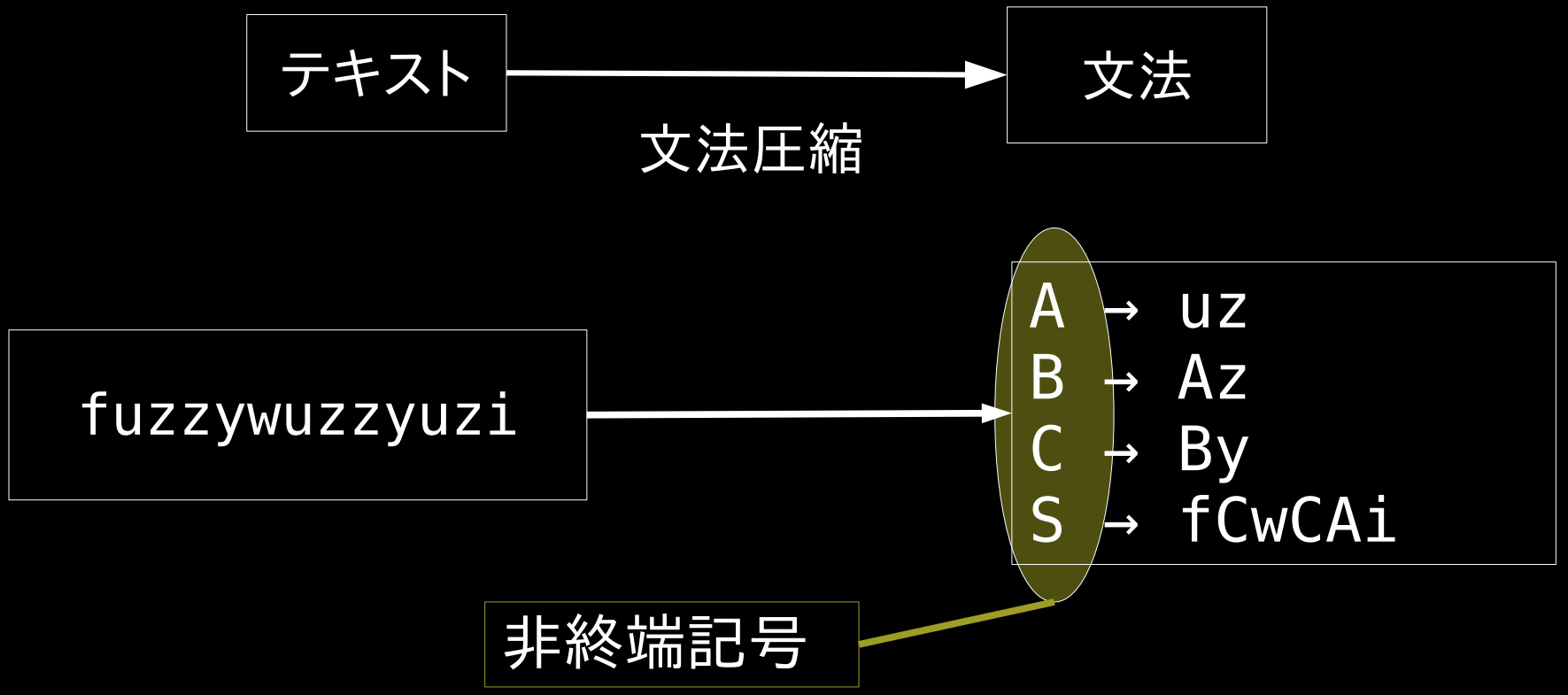

```
fuzzywuzzyuzi
```

# 文法圧縮


fuzzywuzzyuzi

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

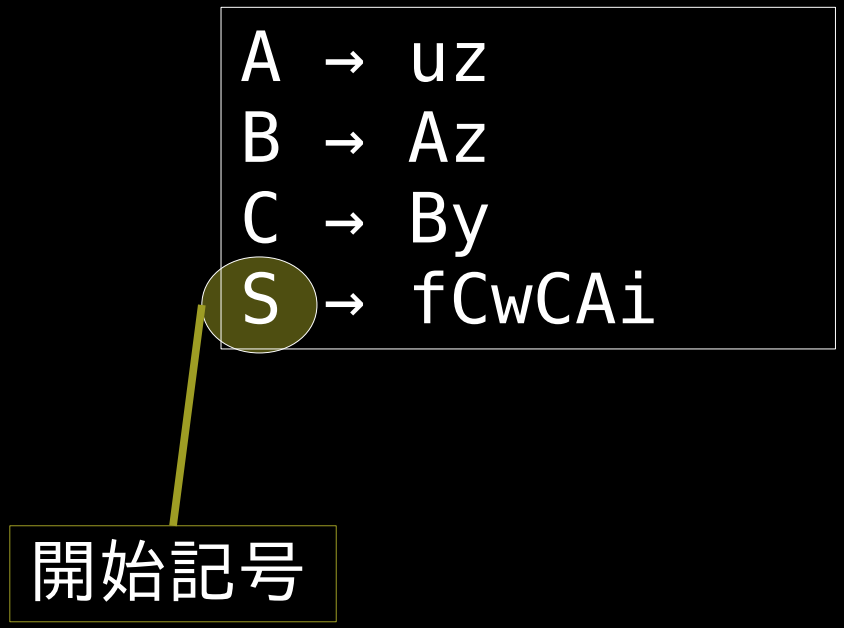
# 文法圧縮



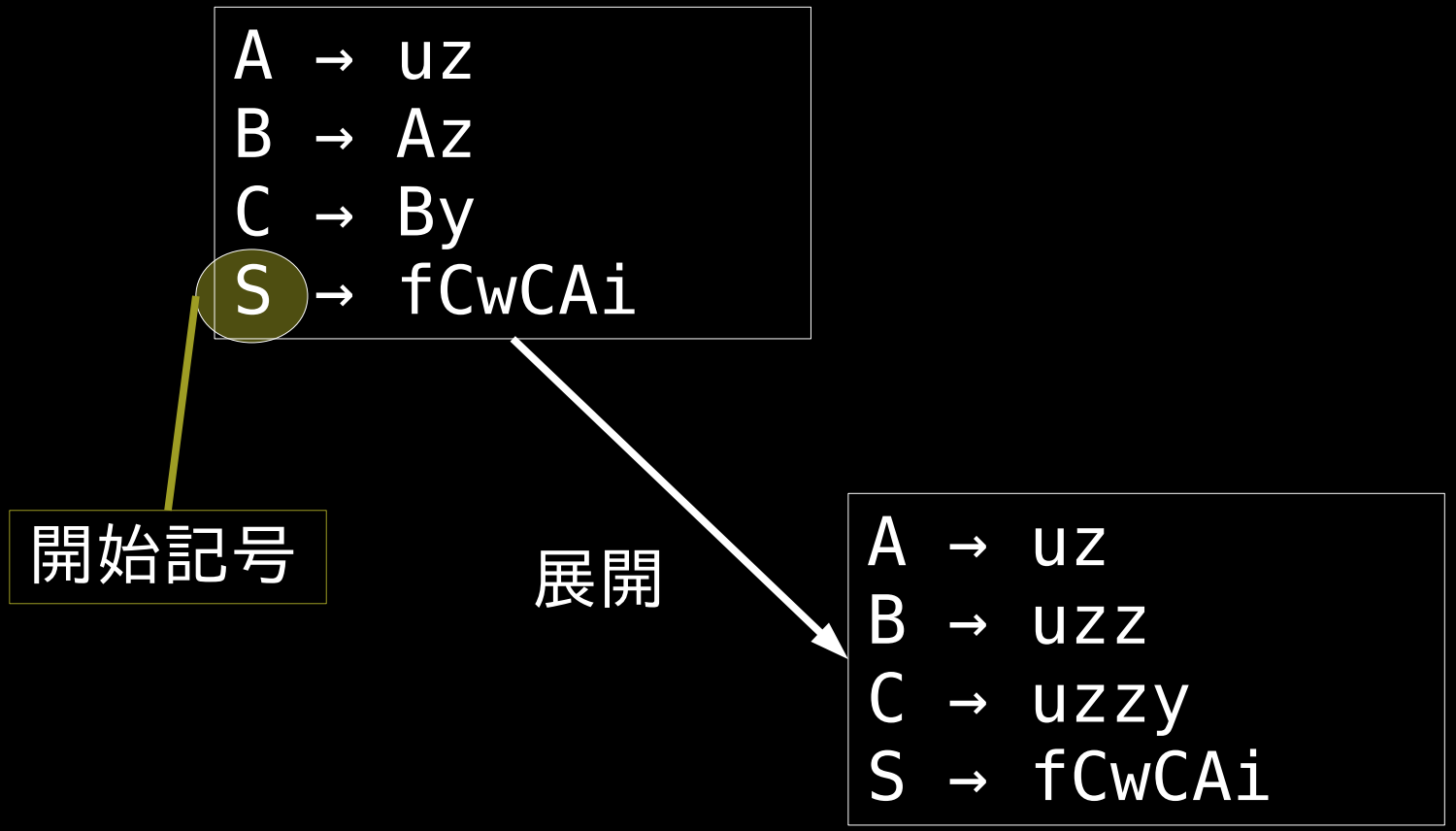
# 復元

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

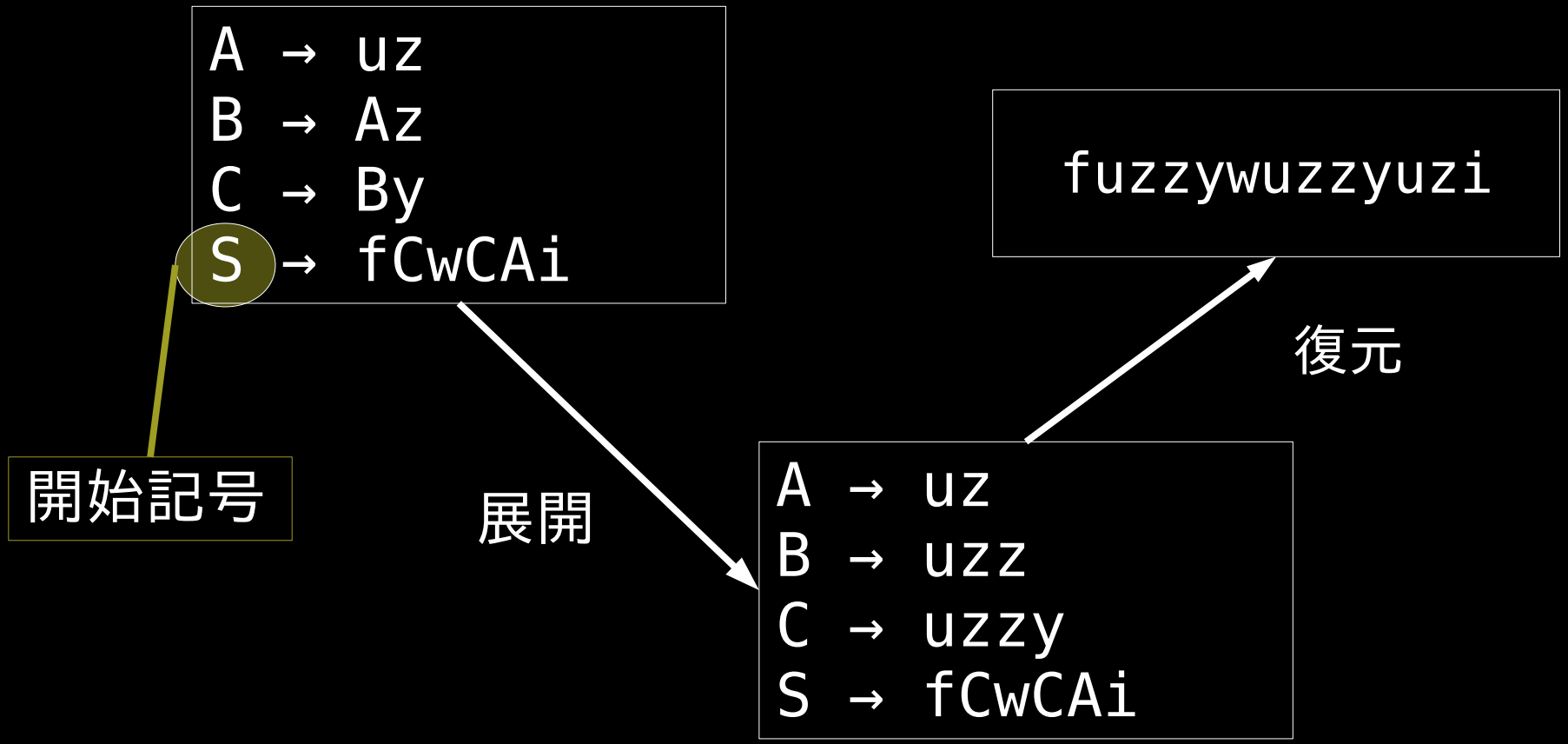
# 復元



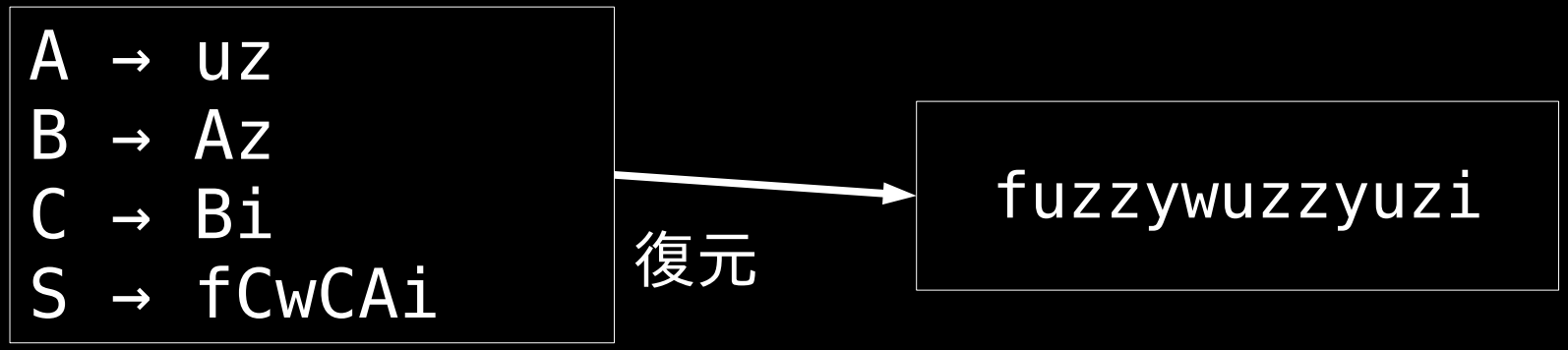
# 復元



# 復元

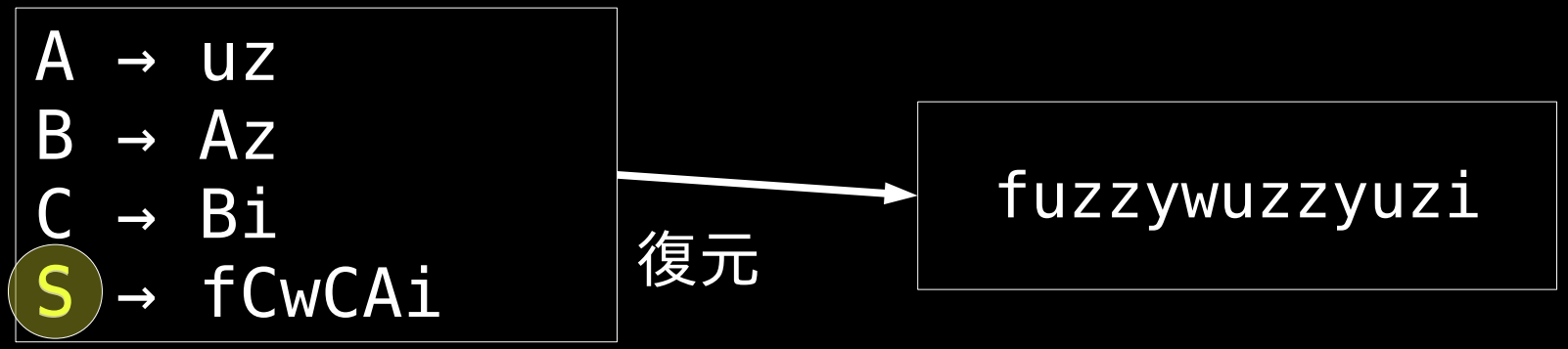

A → uz
B → Az
C → By
S → fCwCAi
開始記号
展開
A → uz
B → uzz
C → uzzy
S → fCwCAi
復元
fuzzywuzzyuzi

# SLP の定義



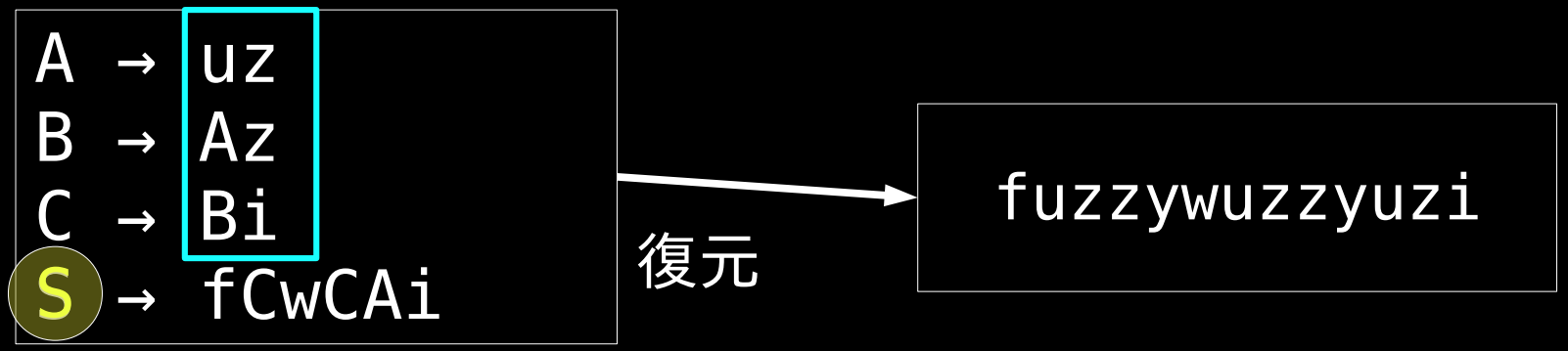
# SLP の定義

- 一つの開始記号



# SLP の定義

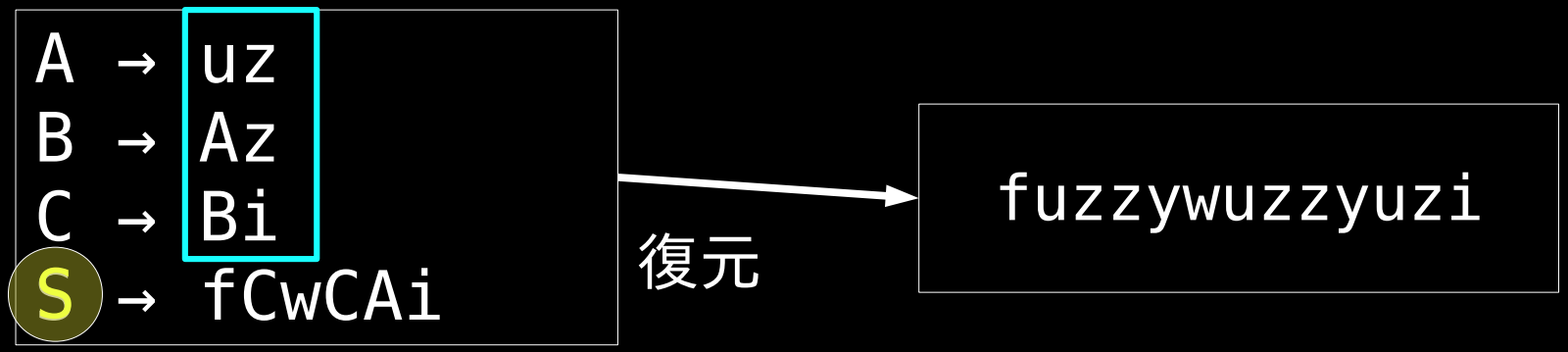
- 一つの開始記号
- 開始記号以外、右辺は２記号



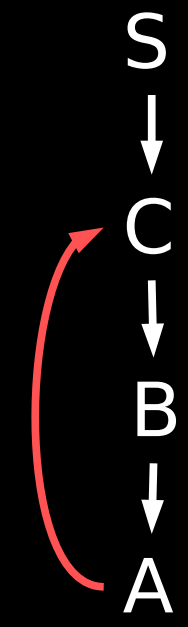
# SLP の定義

S → C → B → A

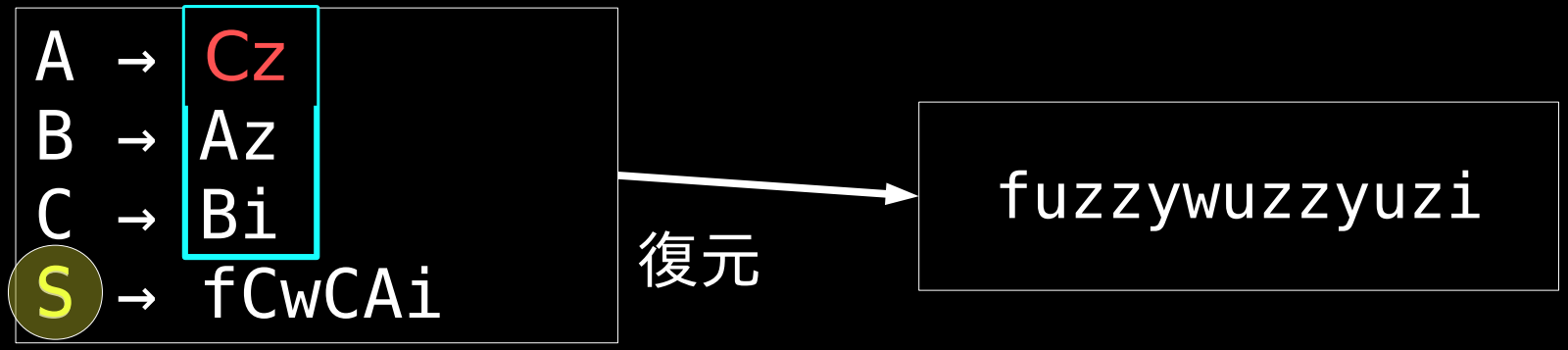
- 一つの開始記号
- 開始記号以外、右辺は２記号
- 閉路がない



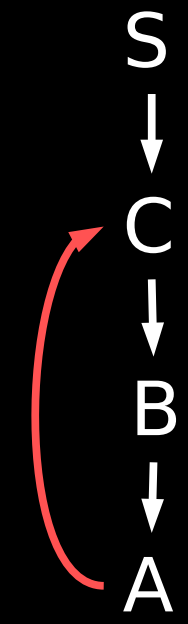
# SLP の定義




- 一つの開始記号
- 開始記号以外、右辺は２記号
- 閉路がない



# SLP の定義




- 一つの開始記号
- 開始記号以外、右辺は２記号
- 閉路がない
- 各非終端記号は一つ規則しかない

```
A → Cz
B → Az
C → Bi
S → fCwCAi

A → zz
```

復元

fuzzywuzzyuzi

# bigram の定義

- 入力：テキスト $T$
- bigram：文字ペア
- bigram の頻度は重複しない出現の数
- $\#(b) := b$ の頻度

fuzzywuzzyuzi

# bigram の定義

- 入力：テキスト $T$
- bigram：文字ペア
- bigram の頻度は重複しない出現の数
- $\#(b)$ := $b$ の頻度

fuzzywuzzyuzi

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

# Re-Pair

- SLP 文法圧縮

fuzzywuzzyuzi

# Re-Pair

- SLP 文法圧縮
- 一番頻度の高い bigram を取って、

#(uz) = 3

fuzzywuzzyuzi

# Re-Pair

- SLP 文法圧縮
- 一番頻度の高い bigram を取って、非終端記号に置換する

#(uz) = 3

A → uz

fuzzywuzzyuzi

↓

fA_zywA_zyA_i

# Re-Pair

- SLP 文法圧縮
- 一番頻度の高い bigram を取って、非終端記号に置換する
- 繰り返す

#(uz) = 3

A → uz

fuzzywuzzyuzi

↓

fA_zywA_zyA_i

$T_1$=fA_zywA_zyA_i

# 縮める

$T_1$=fA_zywA_zyA_i

$T_1$=fAzywAzyAi　#(Az)=2

# 縮める

$T_1$=fA_zywA_zyA_i

$T_1$=fAzywAzyAi

$T_2$=fB_ywB_yAi

#(Az) = 2

B → Az

# 縮める

$T_1$=fA_zywA_zyA_i

$T_1$=fAzywAzyAi　#(Az) = 2

B → Az

$T_2$=fB_ywB_yAi

$T_2$=fBywByAi　#(By) = 2

# 縮める

$T_1$=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

# 縮める

$T_1$=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

# 縮める

$T_1$=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

bigram の頻度は
すべて 1
⇒ 終わり

A → uz
B → Az
C → By
S → fCwCAi

#(Az) = 2
B → Az

#(By) = 2
C → By

$T_3$=fCwCAi

# O(n) 時間のアルゴリズム

Larson, Moffat'00:

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

Bille ら '17:

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

- $n$: 入力の文字列の長さ
- σ: アルファベットサイズ
- $\pi$: 非終端記号の数
- $\varepsilon$: 正の実数

# この話

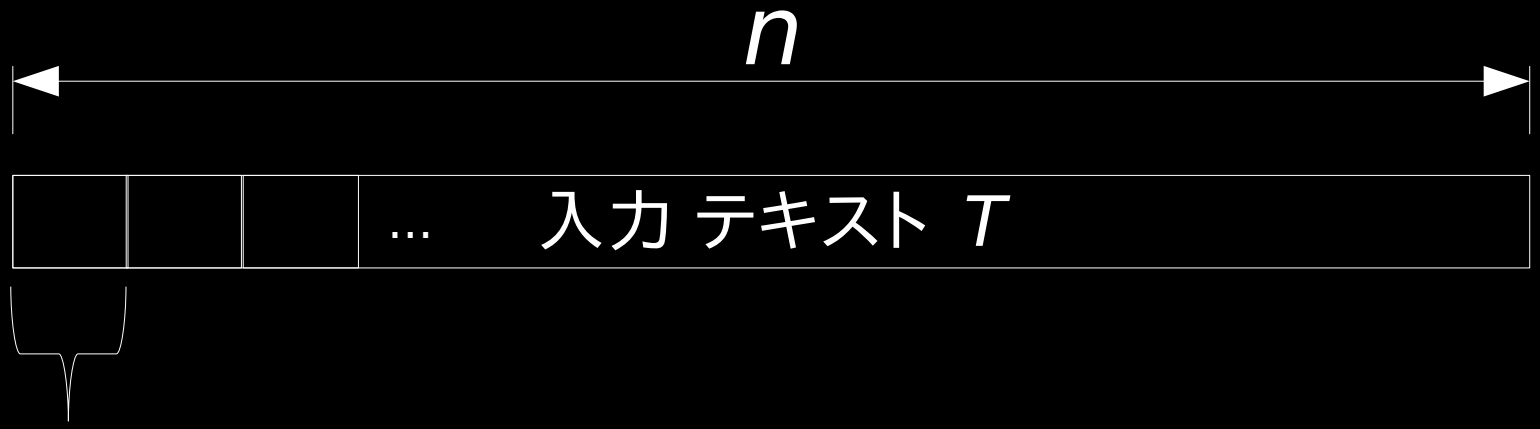
- O($n^2$) 時間
- 領域 ＝ 入力 + O(1) words のみ
- 入力が書き換えられる

# この話

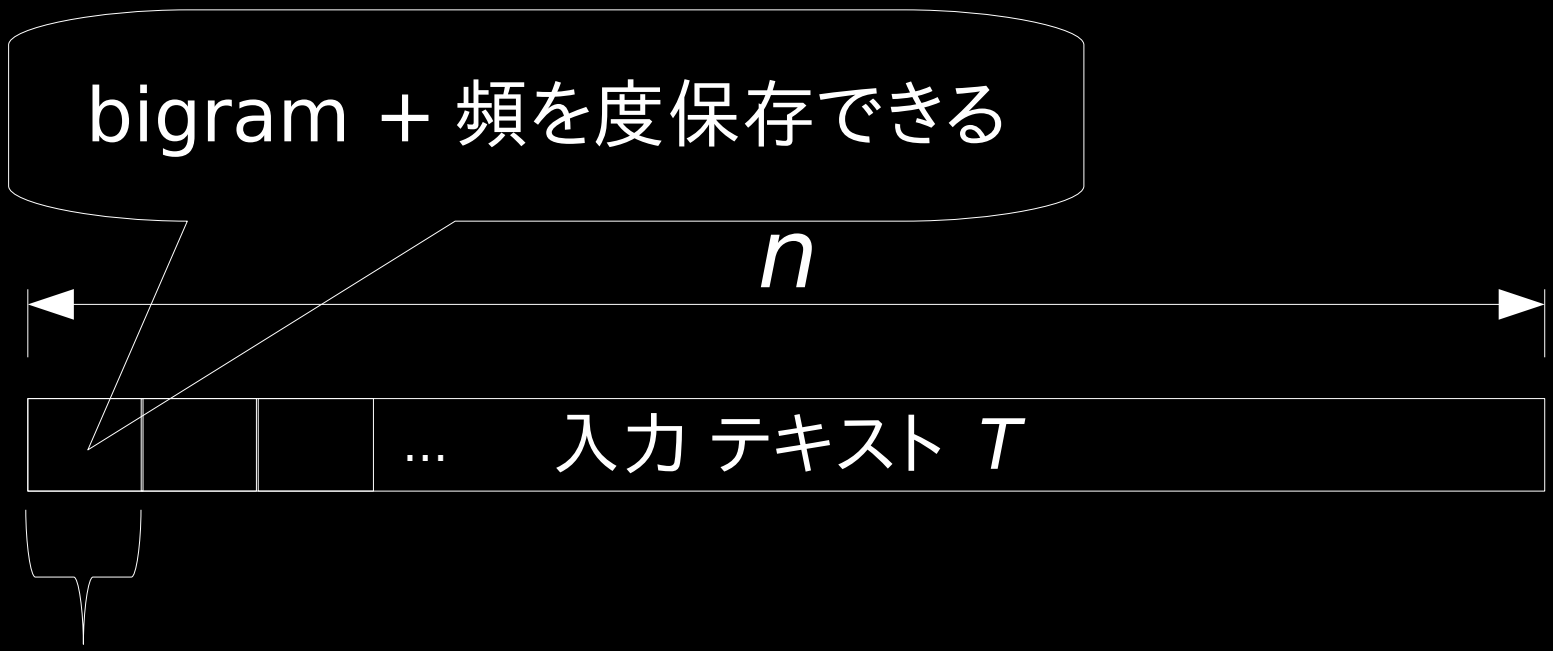
- O($n^2$) 時間
- 領域 ＝ 入力 + O(1) words のみ
- 入力が書き換えられる

# この話

- O($n^2$) 時間
- 領域 ＝ 入力 + O(1) words のみ
- 入力が書き換えられる




この発表: 文字のサイズは O(1) words
(lg $\sigma$ bits も可能 )




この発表:　文字のサイズは O(1) words
( この制限がなくても可能 )

# $O(n^3)$ 時間

# $O(n^3)$ 時間

- 一番頻度の大きい bigram $b$ を探す

# O($n^3$) 時間

- 一番頻度の大きい bigram $b$ を探す
- $b$ のある出現位置を $i$ とすると、
  - $\#(b) = \#(\mathrm{T}[i]\mathrm{T}[i+1])$
    $= \max_{1 \leqslant j \leqslant n} \#(T[j]T[j+1])$ である

# $O(n^3)$ 時間

- 一番頻度の大きい bigram $b$ を探す
- $b$ のある出現位置を $i$ とすると、
  - $\#(b) = \#(T[i]T[i+1])$
    $= \max_{1 \leqslant j \leqslant n} \#(T[j]T[j+1])$ である
  - $O(n^2)$ 時間で計算できる

$O(n)$ 時間

# O($n^3$) 時間

- 一番頻度の大きい bigram $b$ を探す
- $b$ のある出現位置を $i$ とすると
  - $\#(b) = \#(T[i]T[i+1])$
    $= \max_{1 \leqslant j \leqslant n} \#(T[j]T[j+1])$ である
  - O($n^2$) 時間で計算できる
- O($n$) 時間で $b$ の出現を置換できる

O($n$) 時間

# O($n^3$) 時間

- 一番頻度の大きい bigram $b$ を探す
- $b$ のある出現位置を $i$ とすると
  - $\#(b) = \#(T[i]T[i+1])$
    $= \max_{1 \leqslant j \leqslant n} \#(T[j]T[j+1])$ である
  - O($n^2$) 時間で計算できる
- O($n$) 時間で $b$ の出現を置換できる
- 異なる bigram の数は高々 $n$ である

O($n$) 時間

# O($n^3$) 時間

- 一番頻度の大きい bigram $b$ を探す
- $b$ のある出現位置を $i$ とすると、
  - $\#(b) = \#(\mathrm{T}[i]\mathrm{T}[i+1])$
    $= \max_{1 \leqslant j \leqslant n} \#(T[j]T[j+1])$ である
  - O($n^2$) 時間で計算できる
- O($n$) 時間で $b$ の出現を置換できる
- 異なる bigram の数は高々 $n$ である

  ⇒ O($n^3$) 時間

O($n$) 時間

# 方針

# 方針

- 置換の後、入力の領域を空ける

# 方針

- 置換の後、入力の領域を空ける
  ⇒ もっと頻度を保存できる

# 方針

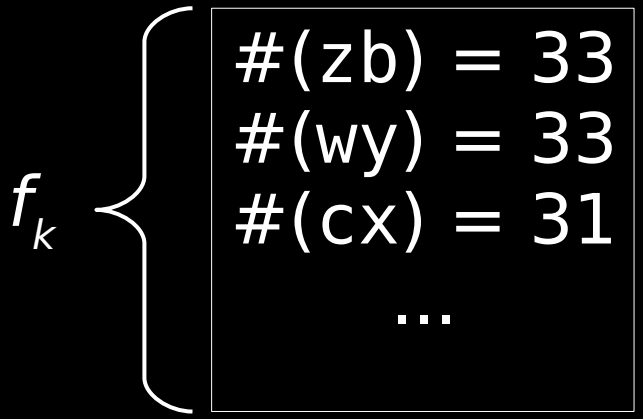
- 置換の後、入力の領域を空ける
  ⇒ もっと頻度を保存できる
- algorithm を round で区切る

# 方針

- 置換の後、入力の領域を空ける
  ⇒ もっと頻度を保存できる
- algorithm を round で区切る
- $k$ 番目の round の初め：
  - $f_k$ : 保存できる頻度の数

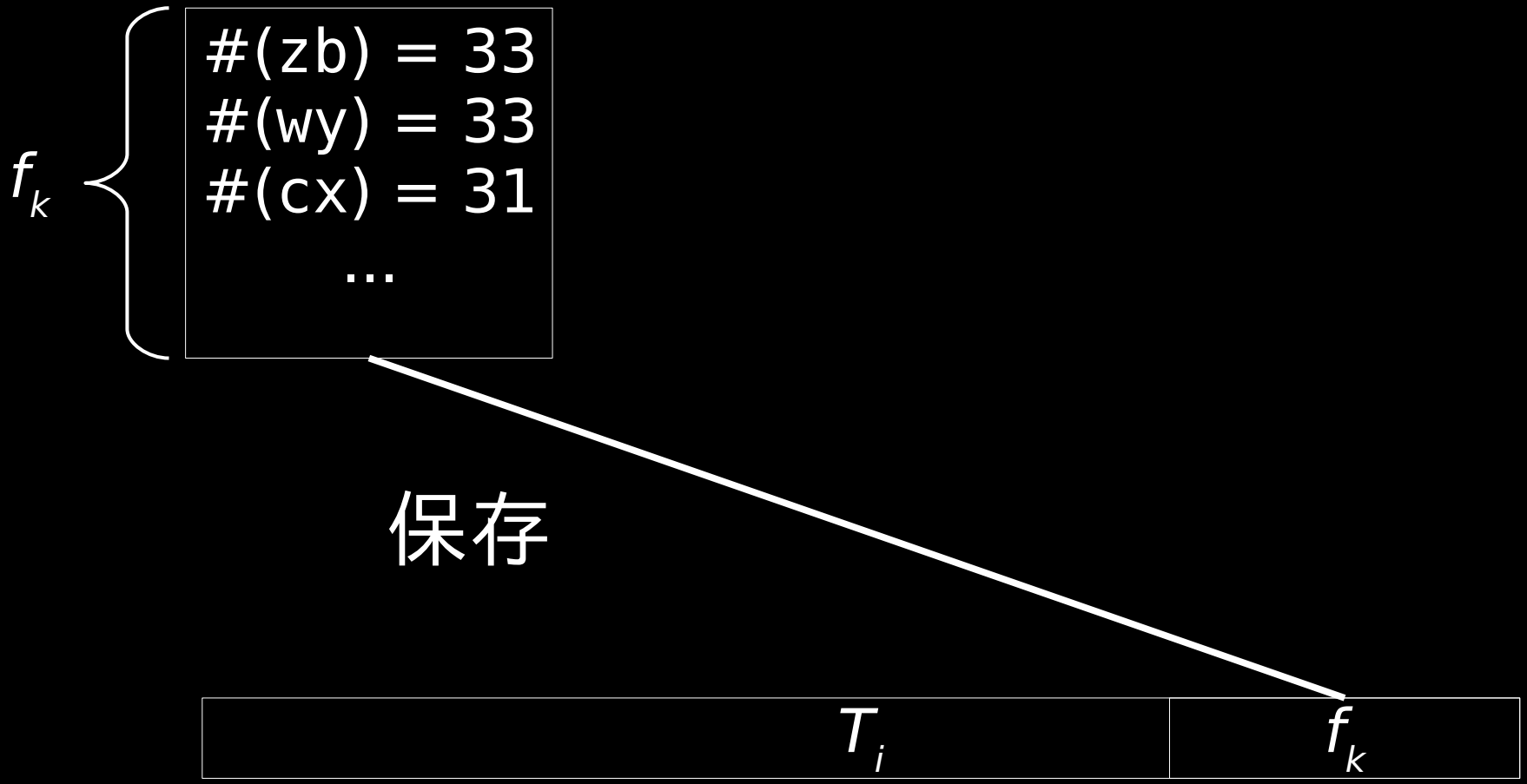
# 方針

- 置換の後、入力の領域を空ける
  ⇒ もっと頻度を保存できる
- algorithm を round で区切る
- $k$ 番目の round の初め：
  - $f_k$ : 保存できる頻度の数
  - $f_k$ の一番大きい頻度を計算する





$k$ 番目 round，規則の数：$i$




$k$ 番目 round，規則の数：$i$

保存しない bigram の中に一番頻度が高い

#(cx) = 20

$f_k$

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

保存


$k$ 番目 round，規則の数：$i$

保存しない bigram の中に一番頻度が高い

#(cx) = 20

$f_k$

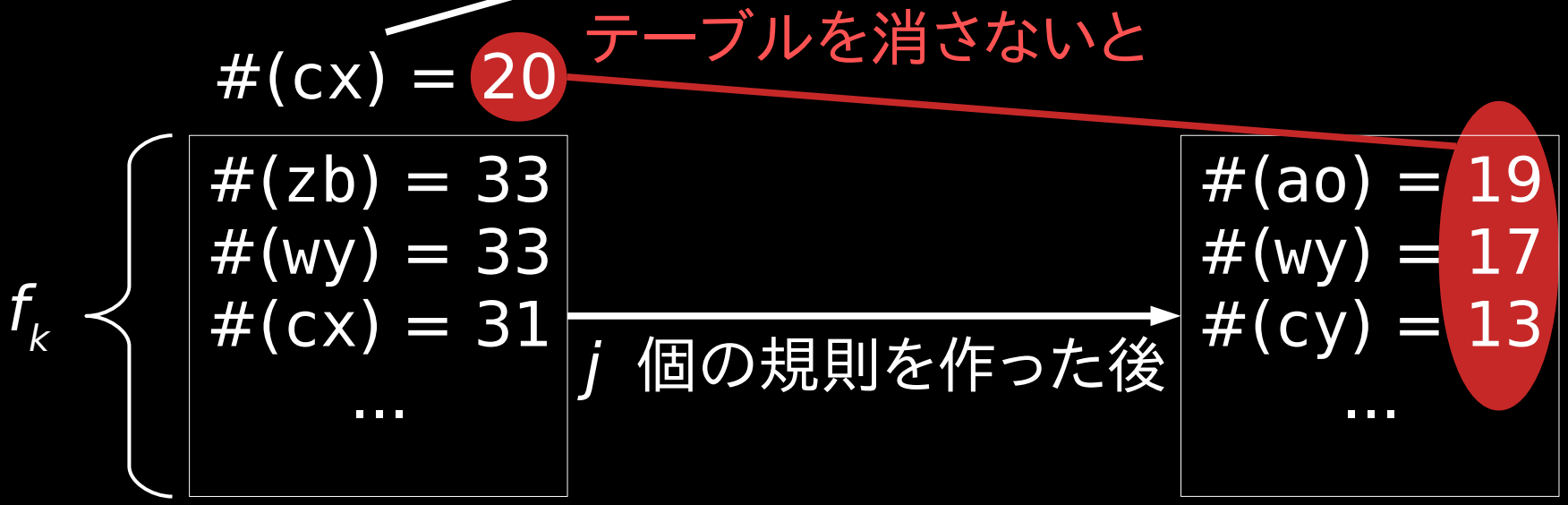
#(zb) = 33
#(wy) = 33
#(cx) = 31
...

$j$ 個の規則を作った後

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


$k$ 番目 round，規則の数：$i+j$

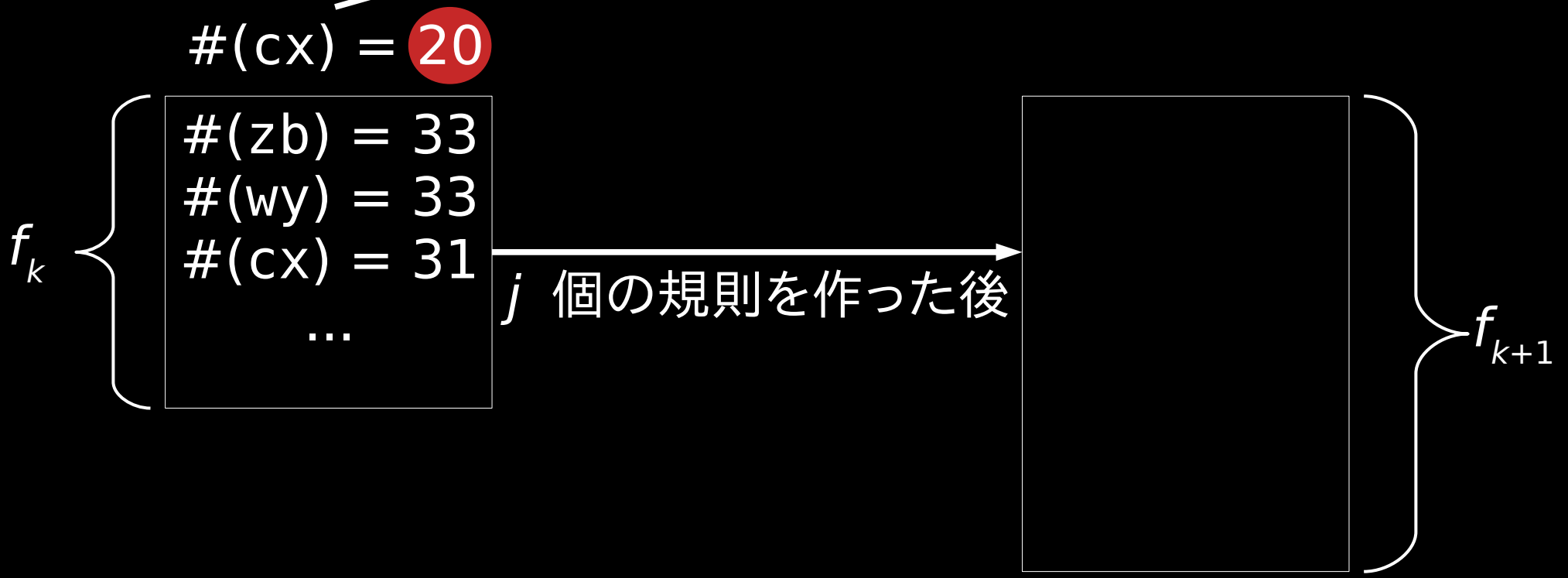
保存しない bigram の中に一番頻度が高い





$k$ 番目 round，規則の数：$i+j$

保存しない bigram の中に一番頻度が高い





$k+1$ 番目 round，規則の数：$i+j$

# algorithm

- 最初の round: $f_1$ = O(1) = 定数
- $f_1$ 個の最大の頻度を計算し、
- 最大の頻度の bigram を置換し、
- 保存した頻度を修正する

# 修正の必要性

fuzzywuzzyuzi

# 修正の必要性

fuzzywuzzyuzi

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

# 修正の必要性

fuzzywuzzyuzi

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

# 修正の必要性

A → uz

fuzzywuzzyuzi

↓

fAzywAzyAi

#(uz) = 3

#(zz) = 2

#(zy) = 2

# 修正の必要性

A → uz

fuzzywuzzyuzi

↓

fAzywAzyAi

~~#(uz) = 3~~

#(zz) = 0

#(zy) = 2

#(Az) = 2

- 各置換した位置：
  - 高々２つの頻度が減る可能性

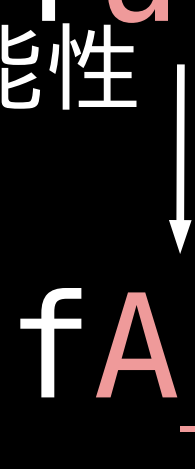
#(fu) = 1
#(zz) = 2

- 各置換した位置：
  - 高々２つの頻度が減る可能性

fuzz

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

- 各置換した位置：
  - 高々２つの頻度が減る可能性



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

- 各置換した位置：
  - 高々２つの頻度が減る可能性

⇒round $k$ の終わり：

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

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

- 各置換した位置：
  - 高々２つの頻度が減る可能性

⇒round $k$ の終わり：

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

全部の bigram の頻度を保存できる!

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

- $k = O(\lg n)$ なら $f_k = \Theta(n)$

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

- 各置換した位置：
  - 高々２つの頻度が減る可能性

⇒round $k$ の終わり：

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

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

全部の bigram の頻度を保存できる!

- $k = O(\lg n)$ なら $f_k = \Theta(n)$

⇒ 最後の round は高々 $O(\lg n)$ 番目である

# 時間の纏め

# 時間の纏め

- $f_k$ 個の bigram の頻度を計算する：

  $O(n^2)$ 時間 + sort($f_k$) の時間

  $= O(n^2)$ 時間 ($\because f_k \leqslant n$)

# 時間の纏め

- $f_k$ 個の bigram の頻度を計算する：

  $O(n^2)$ 時間 + sort($f_k$) の時間

  $= O(n^2)$ 時間 ($\because f_k \leqslant n$)

- O(lg $n$) 回、頻度を計算する

# 時間の纏め

- $f_k$ 個の bigram の頻度を計算する：

  $O(n^2)$ 時間 + sort($f_k$) の時間

  $= O(n^2)$ 時間 ($\because f_k \leqslant n$)

- $O(\lg n)$ 回、頻度を計算する

  $\Rightarrow O(n^2 \lg n)$ 時間

# 時間の纏め

- $f_k$ 個の bigram の頻度を計算する：

  $O(n^2)$ 時間 + sort($f_k$) の時間

  $= O(n^2)$ 時間 ($\because f_k \leqslant n$)

- $O(\lg n)$ 回、頻度を計算する

  $\Rightarrow O(n^2 \lg n)$ 時間

- しかし、$O(n^2)$ 時間にしたい！

# 今の道具：ソート

- $f_k$： 頻度を計算したい bigram の数

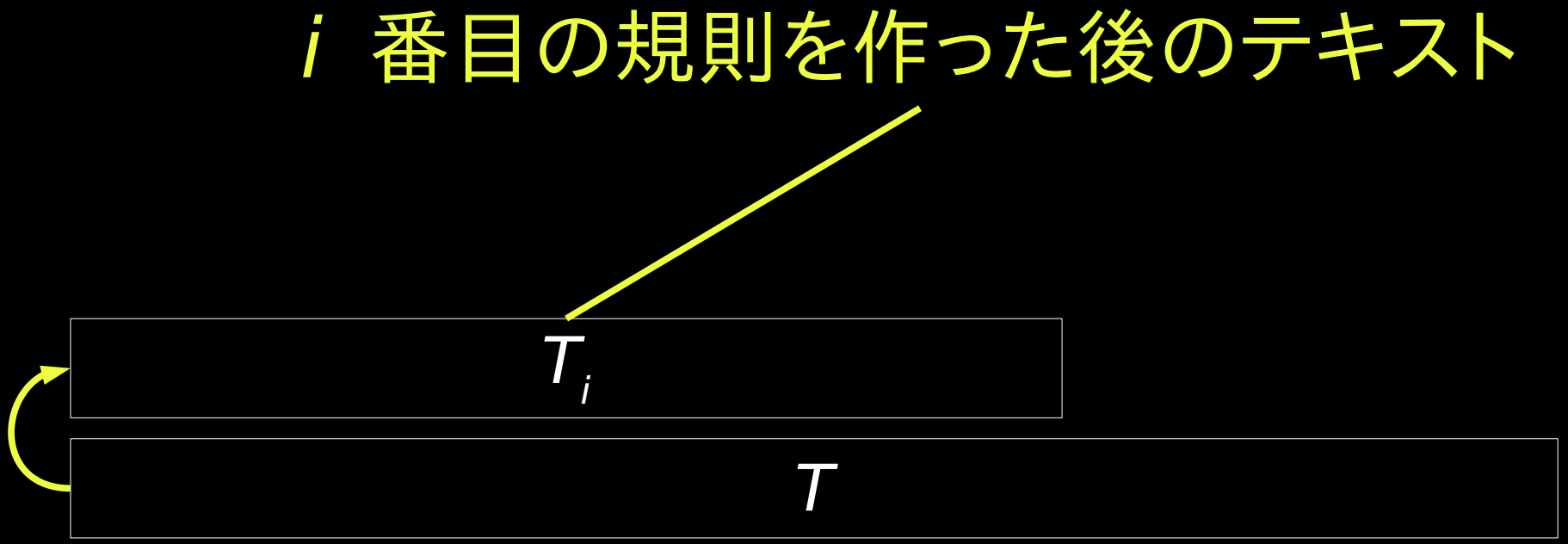
# 今の道具：ソート

- $f_k$：頻度を計算したい bigram の数
- 結果：
  - O($f_k$) 領域（入力が含む）
  - O($f_k$ lg $f_k$) 時間で頻度をソートできる

[Williams'64: heapsort]

# O($n^2$) 時間

- bigram の計算を速める



# O($n^2$) 時間

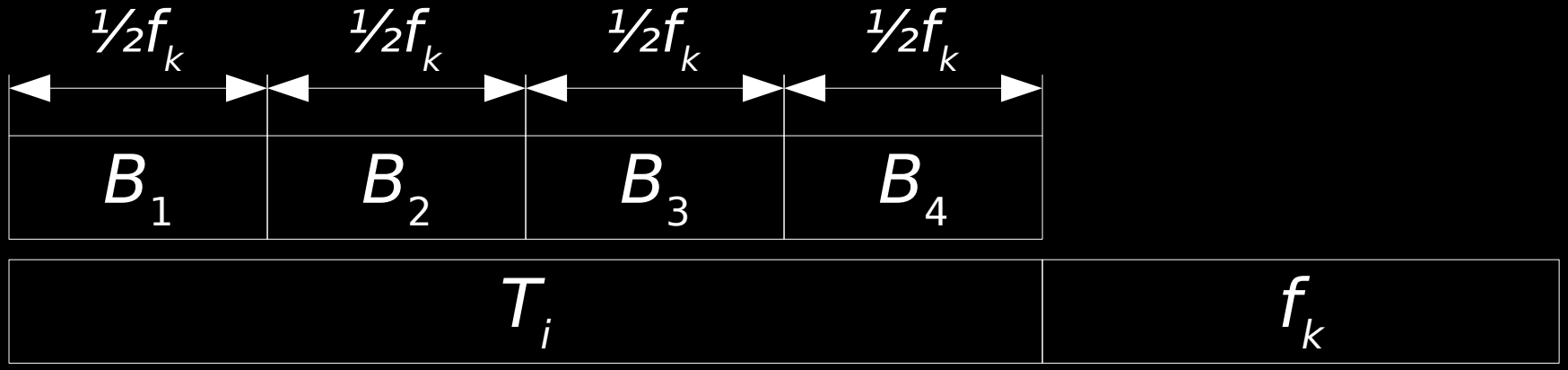
- bigram の計算を速める
- 空き領域： $f_k$ 箇所

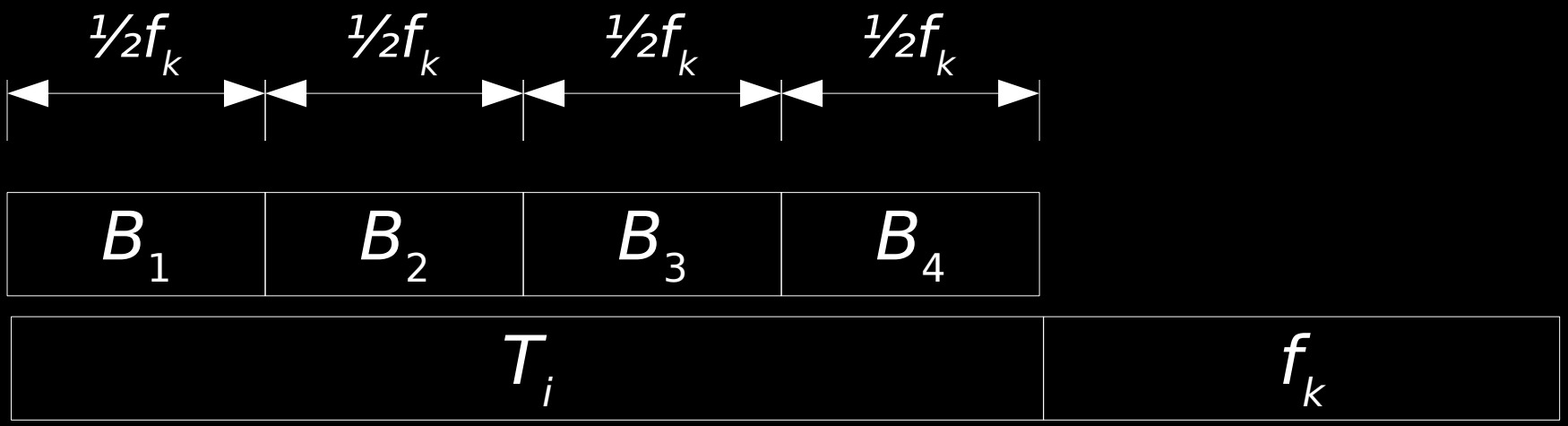
# O($n^2$) 時間

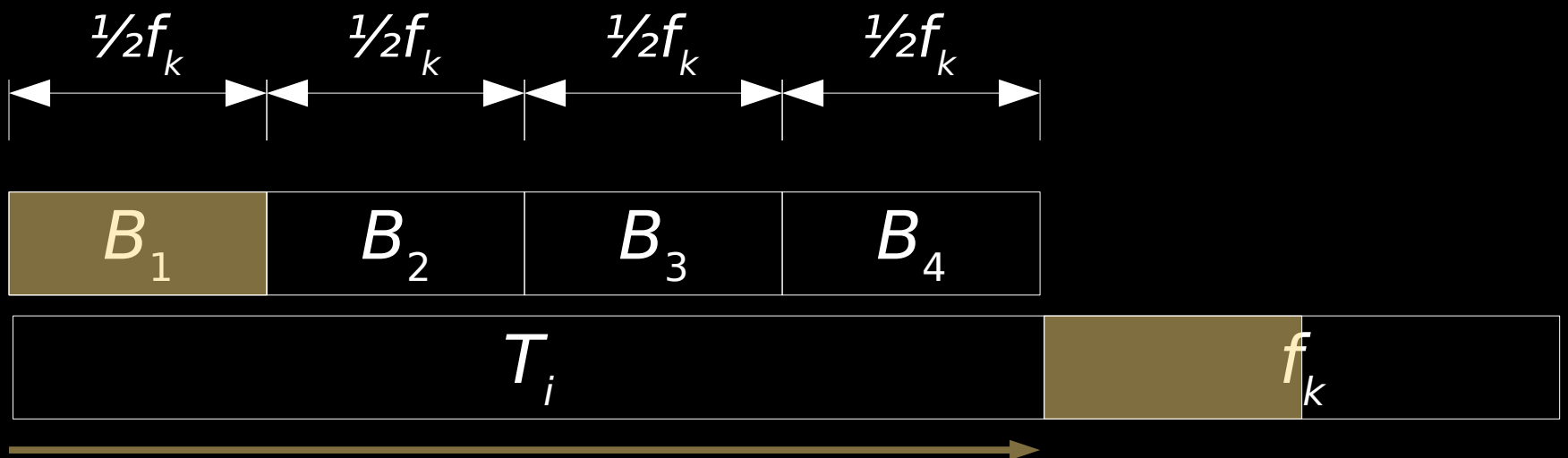
- bigram の計算を速める
- 空き領域： $f_k$ 箇所

# O($n^2$) 時間

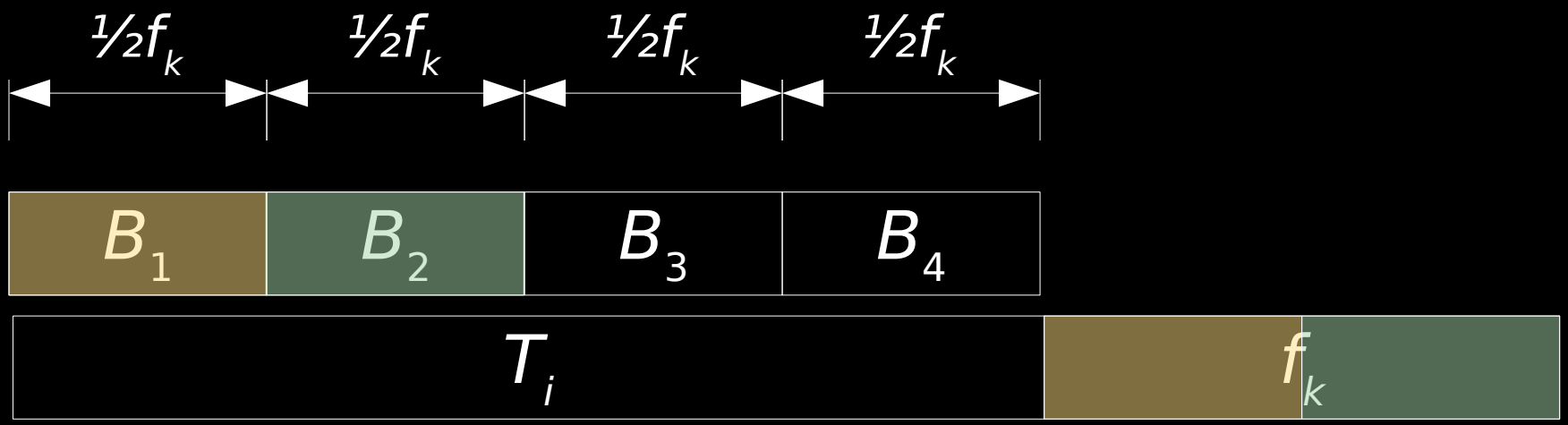
- bigram の計算を速める
- 空き領域： $f_k$ 箇所
- blocks $B_j$ で分割する，$|B_j| = ½f_k$




½f_k
½f_k
½f_k
½f_k
B_1
B_2
B_3
B_4
T_i
f_k

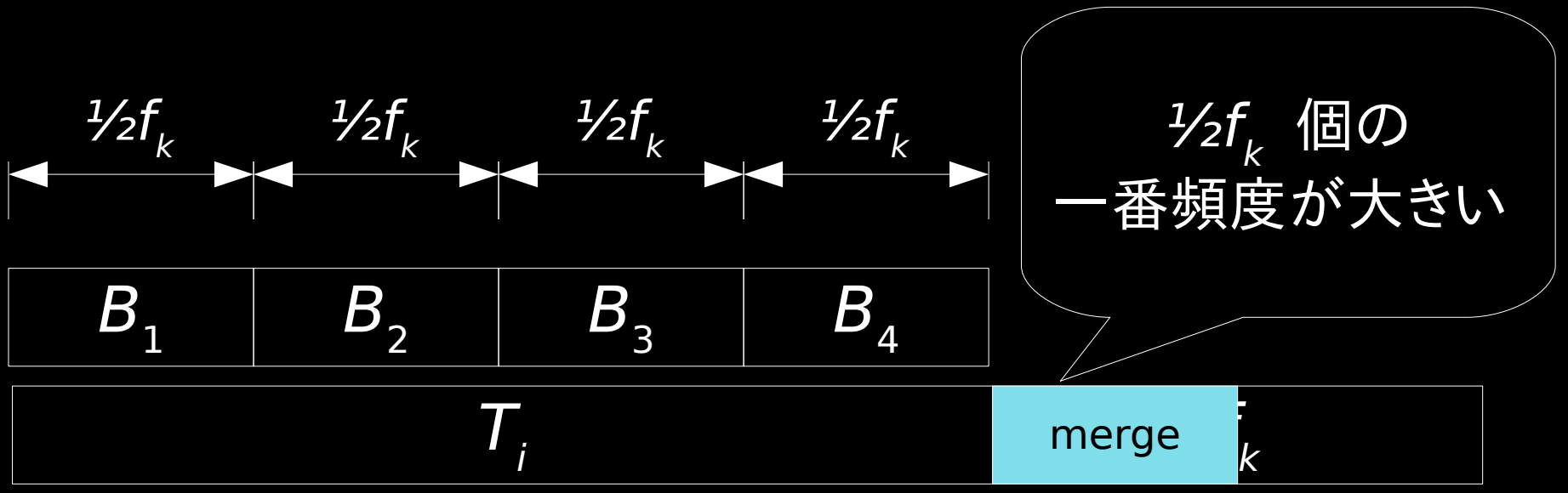



$B_1$で現れる bigram に対した$T_i$の中の頻度を計算し



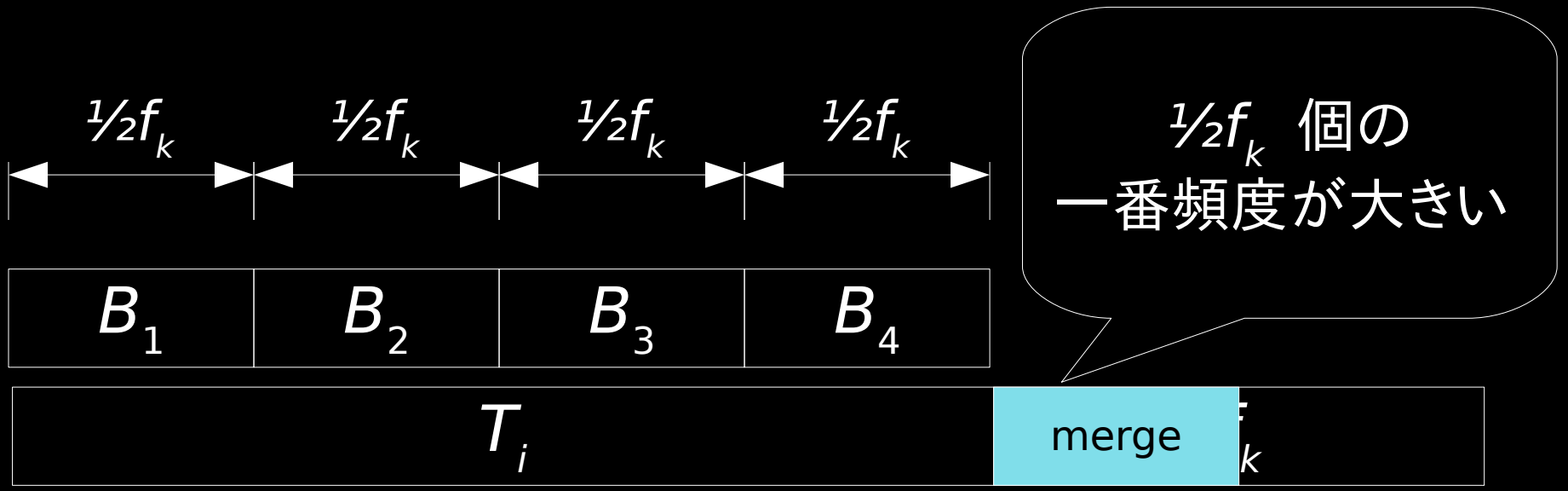

$B_1$で現れる bigram に対した$T_i$の中の頻度を計算し

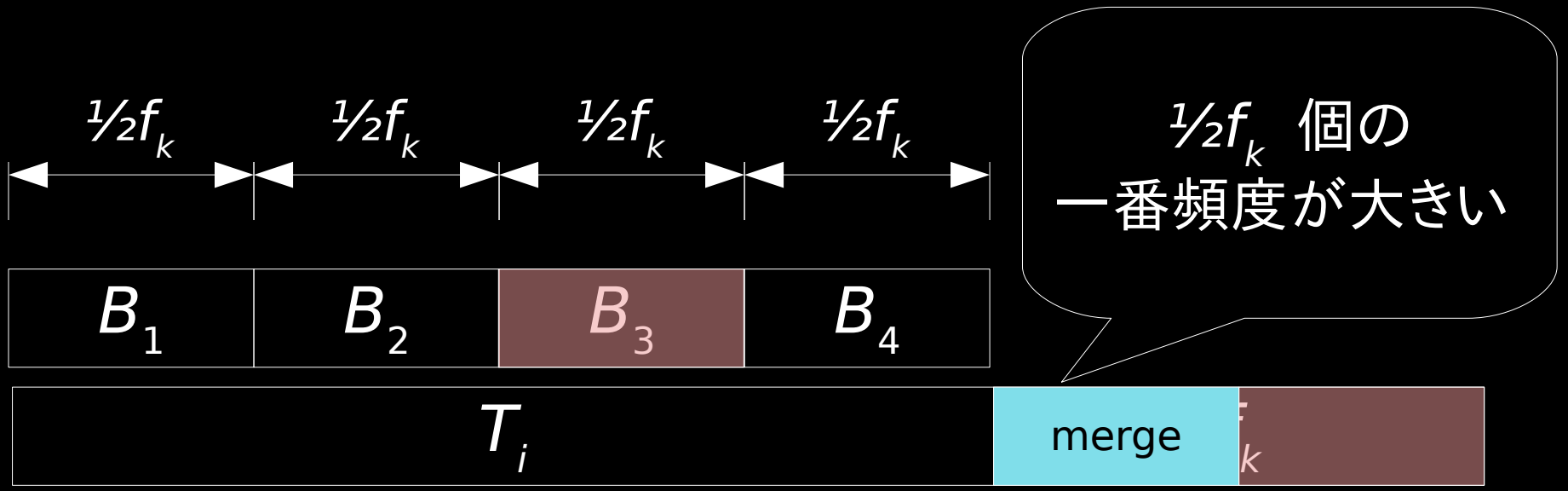
$B_2$ で現れる bigram に対した$T_i$の中の頻度を計算し



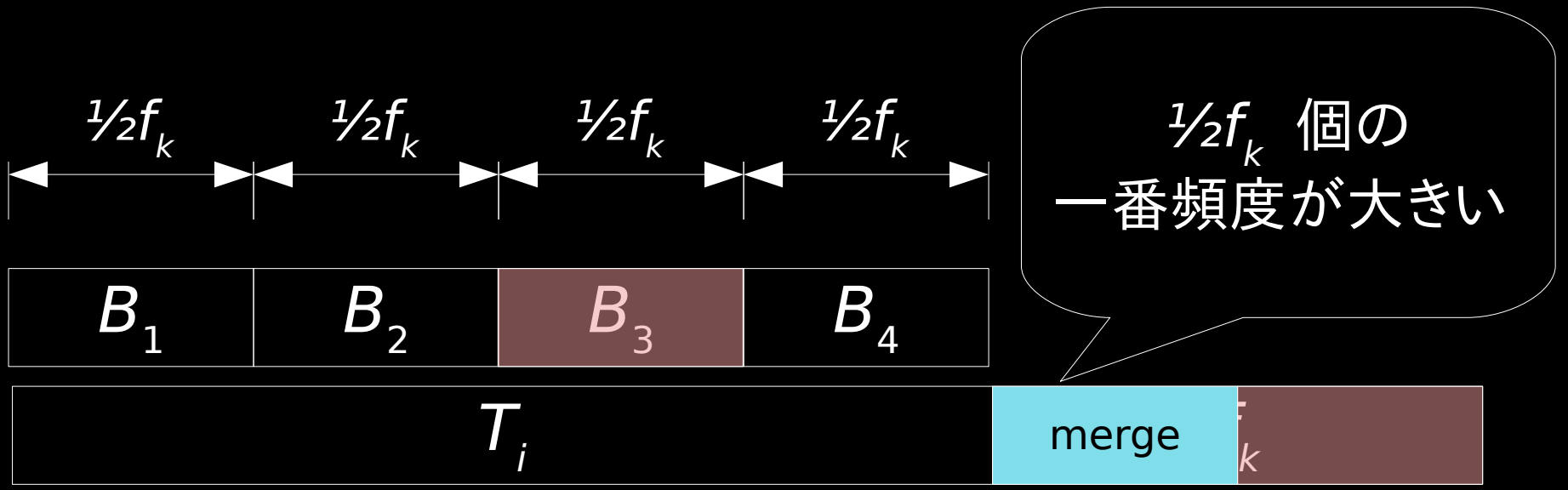

$B_1$で現れる bigram に対した$T_i$の中の頻度を計算し

$B_2$ で現れる bigram に対した$T_i$の中の頻度を計算し


½f_k
½f_k
½f_k
½f_k
½f_k 個の
一番頻度が大きい
B_1
B_2
B_3
B_4
T_i
merge
f_k

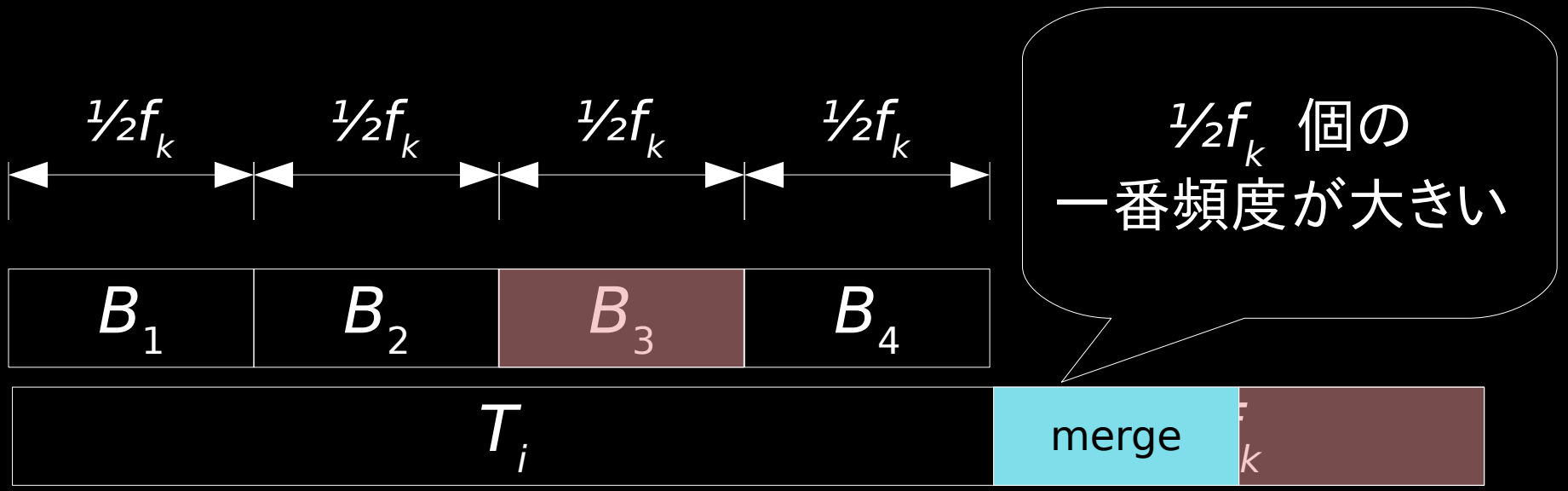



$B_3$ で現れる bigram に対した$T_i$の中の頻度を計算し




$B_3$ で現れる bigram に対した$T_i$の中の頻度を計算し

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




$B_3$ で現れる bigram に対した$T_i$の中の頻度を計算し

- #merge = # $B_j \leqslant n / f_k$, $|T_i| \leqslant |T| = n$
- $B_j$ で頻度を計算する時間： $O(n \lg f_k)$ （２分探索）
- 各 merge の時間 : $O(f_k \lg f_k)$
- 全部の時間 : $O((n^2 \lg f_k)/ f_k)$

# 全部の時間

$$\sum_{k=0}^{O(\lg n)} \frac{n^2}{f_k} \lg f_k = O\left(n^2 \sum_{k}^{\lg n} \frac{k}{1.5^k}\right) = O(n^2)$$

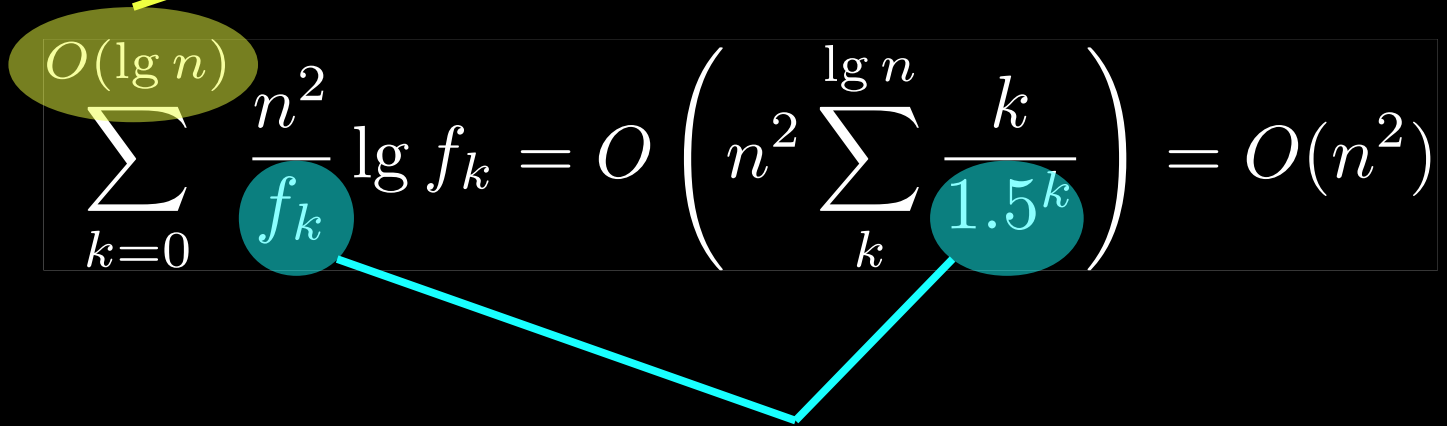
# 全部の時間

round の数は高々 O(lg *n*) である

$$\sum_{k=0}^{O(\lg n)} \frac{n^2}{f_k} \lg f_k = O\left(n^2 \sum_{k}^{\lg n} \frac{k}{1.5^k}\right) = O(n^2)$$

# 全部の時間

round の数は高々 O(lg *n*) である

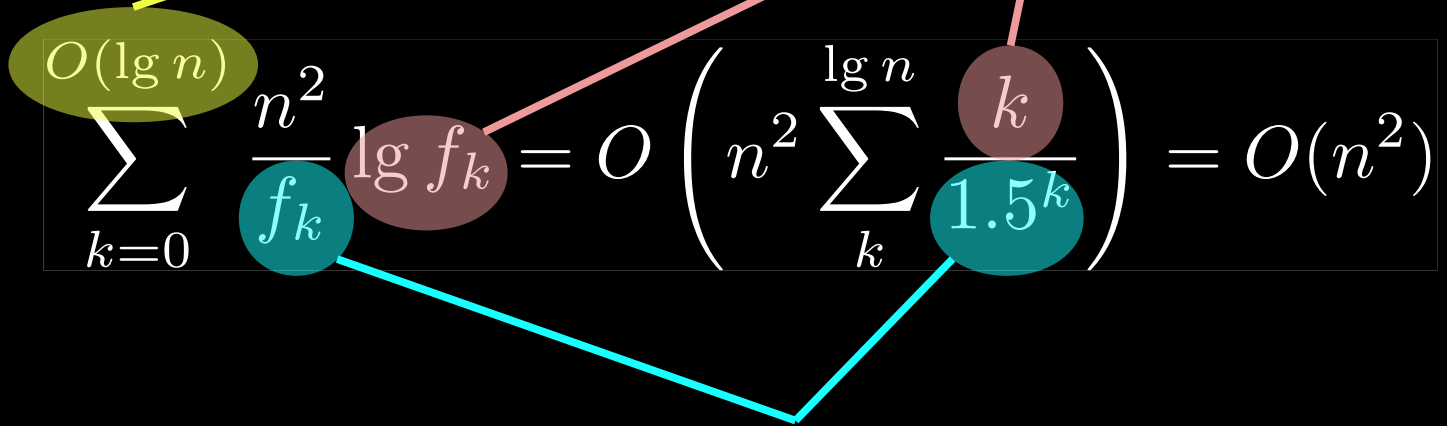



$f_k = 1.5^k f_1 = O(1.5^k)$

# 全部の時間

round の数は高々 O(lg $n$) である

⇒ lg $f_k$ = $O(k)$




$f_k = 1.5^k f_1 = O(1.5^k)$

# まとめ

- 極限領域での Re-Pair
  - $O(n^3)$ 時間：簡単
  - $O(n^2)$ 時間：今回の発表
    - 領域なしソート
    - バッチ処理で bigram の頻度を計算し
  - $o(n^2)$ 時間できるかな？

# まとめ

- 極限領域での Re-Pair
  - $O(n^3)$ 時間：簡単
  - $O(n^2)$ 時間：今回の発表
    - 領域なしソート
    - バッチ処理で bigram の頻度を計算し
  - $o(n^2)$ 時間できるかな？

終わり - 質問は大歓迎です!