PLCP 配列に基づくテキスト圧縮

クップル　ドミニク　（九州大学 /JSPS)

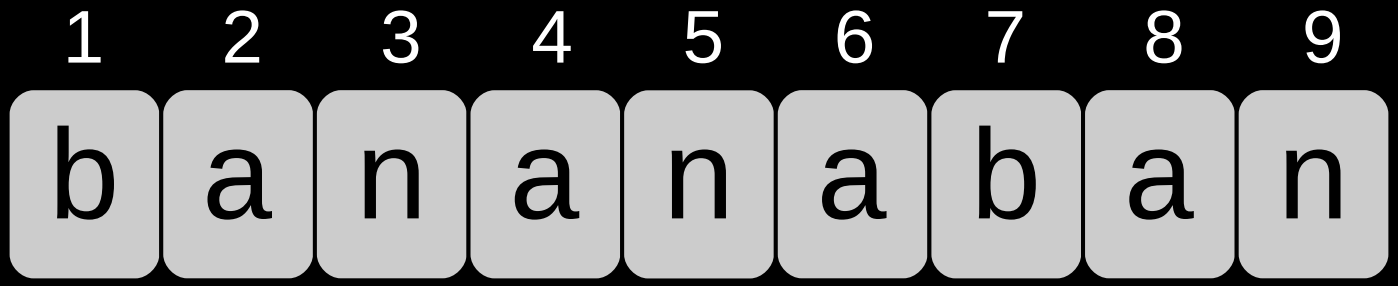
共著 :
Patrick Dinklage
Jonas Ellert
Johannes Fischer
Manuel Penschuck

greedy 圧縮

名前	方法	参考
LZ77	unidirectional	Lempel,Ziv '77
longest first	grammar	中村 + '09
lcpcomp	bidirectional	Dinklage+ '17
LZ-LFS	hybrid	Mauer+ '17 西 + '18
LZRR	bidirectional	西本 +'19

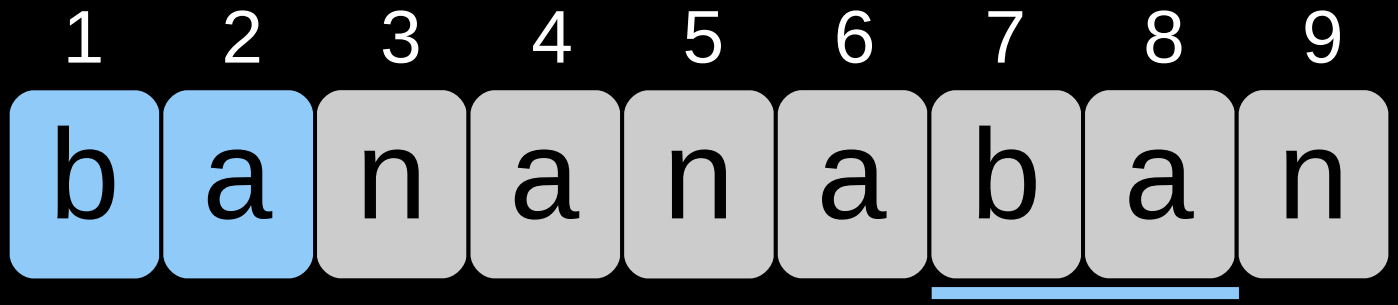
bidirectional 圧縮

- 部分文字列を参照に置換



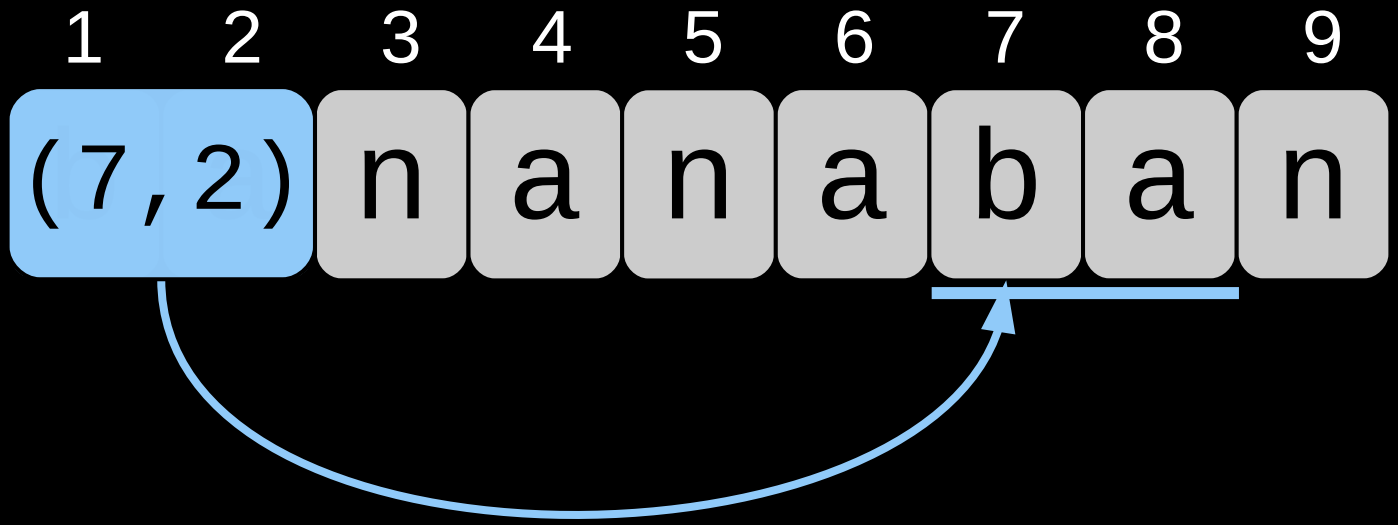
bidirectional 圧縮

- 部分文字列を参照に置換
- 任意な順序



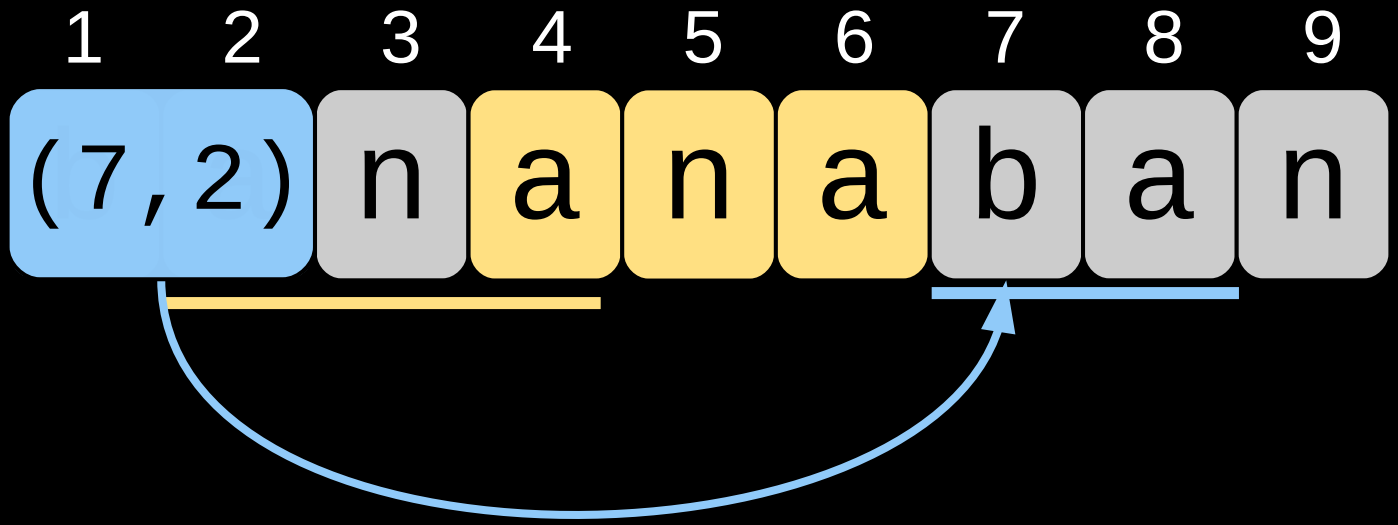
bidirectional 圧縮

- 部分文字列を参照に置換
- 任意な順序



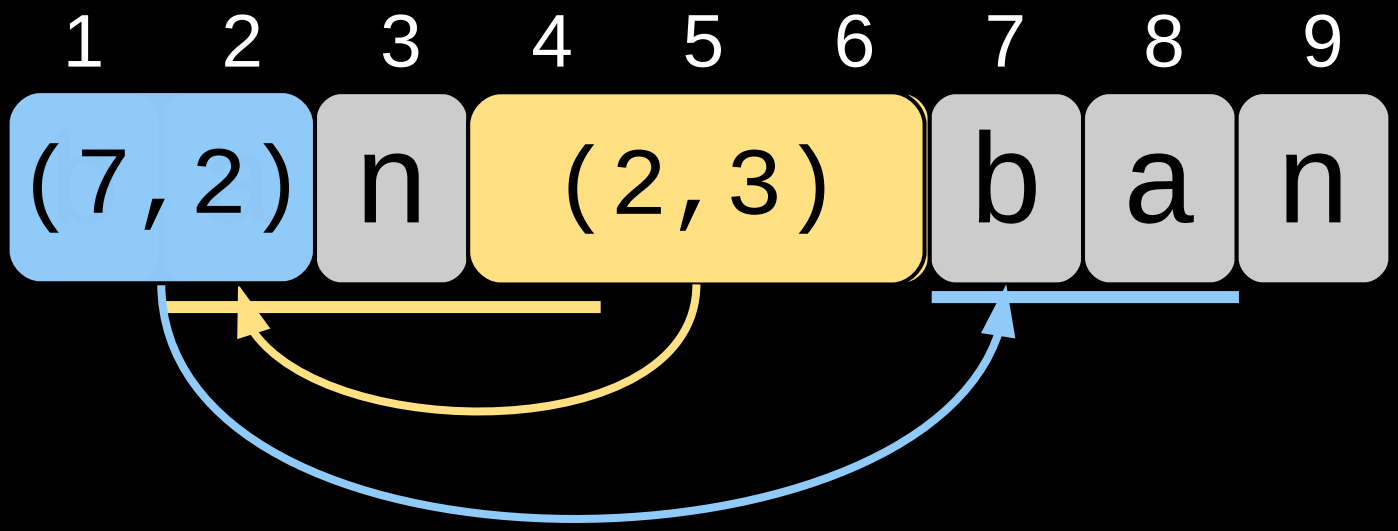
bidirectional 圧縮

- 部分文字列を参照に置換
- 任意な順序
- 自己参照も OK



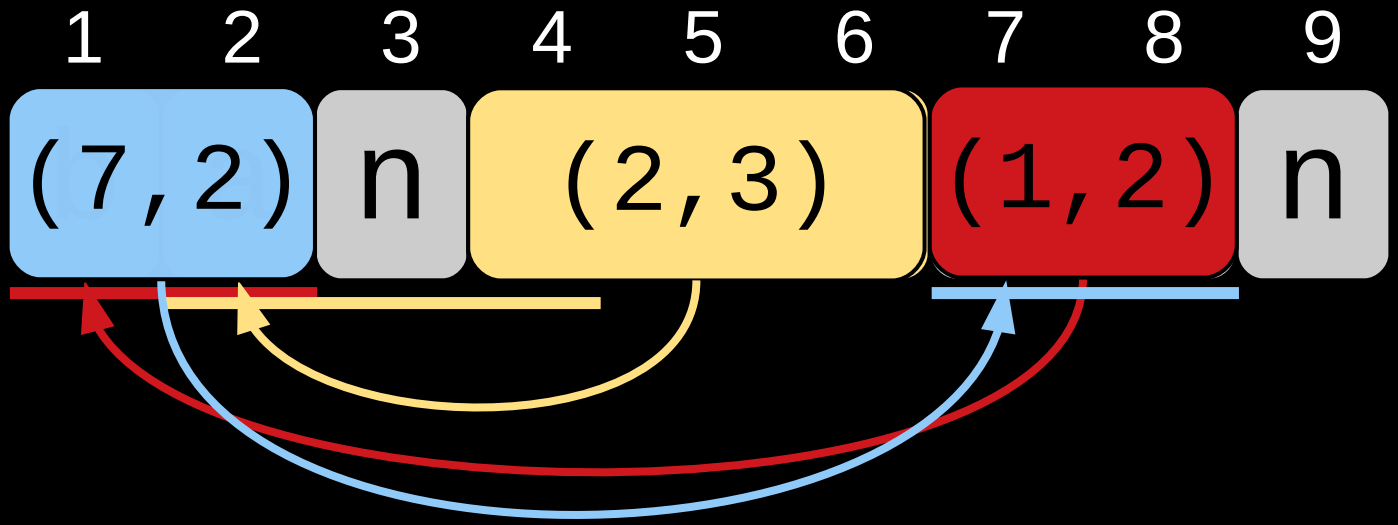
bidirectional 圧縮

- 部分文字列を参照に置換
- 任意な順序
- 自己参照も OK



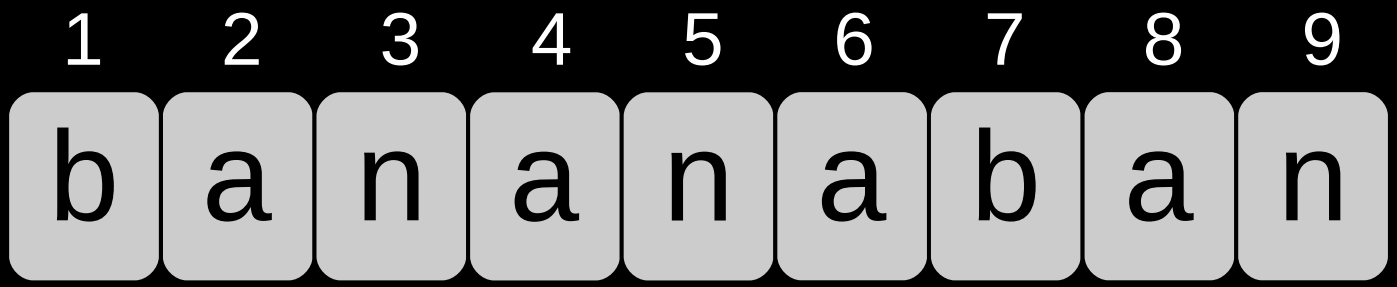
bidirectional 圧縮

- 部分文字列を参照に置換
- 任意な順序
- 自己参照も OK
- 復元のため、 cycle は禁止



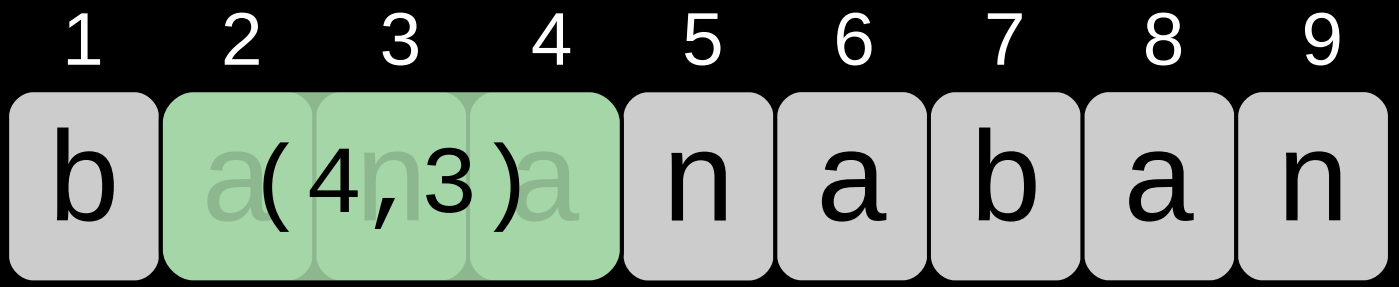
lcpcomp

- 最長の繰り返し部分文字列を置換、
- 繰り返す



lcpcomp

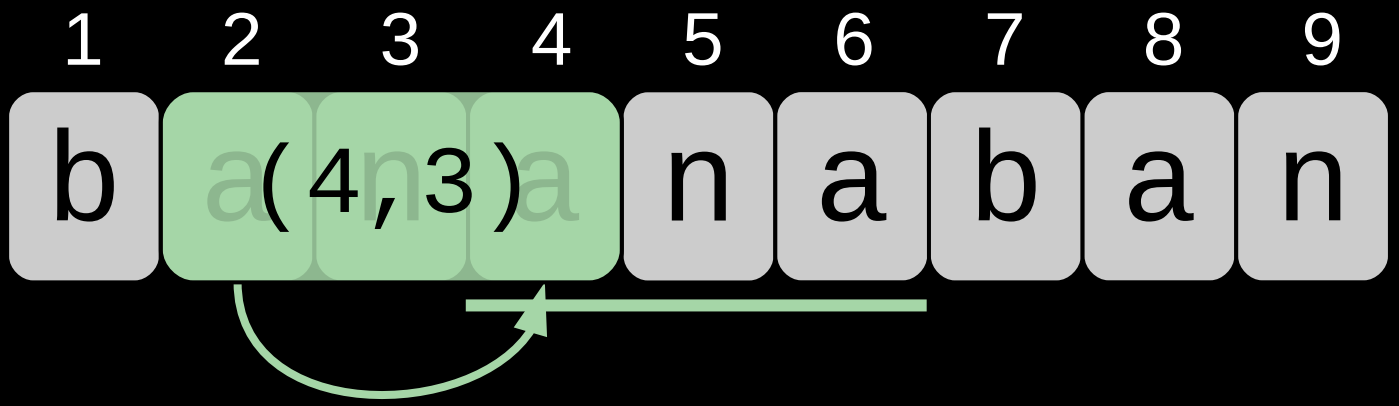
- 最長の繰り返し部分文字列を置換、
- 繰り返す



位置４から３文字をコピーし、

lcpcomp

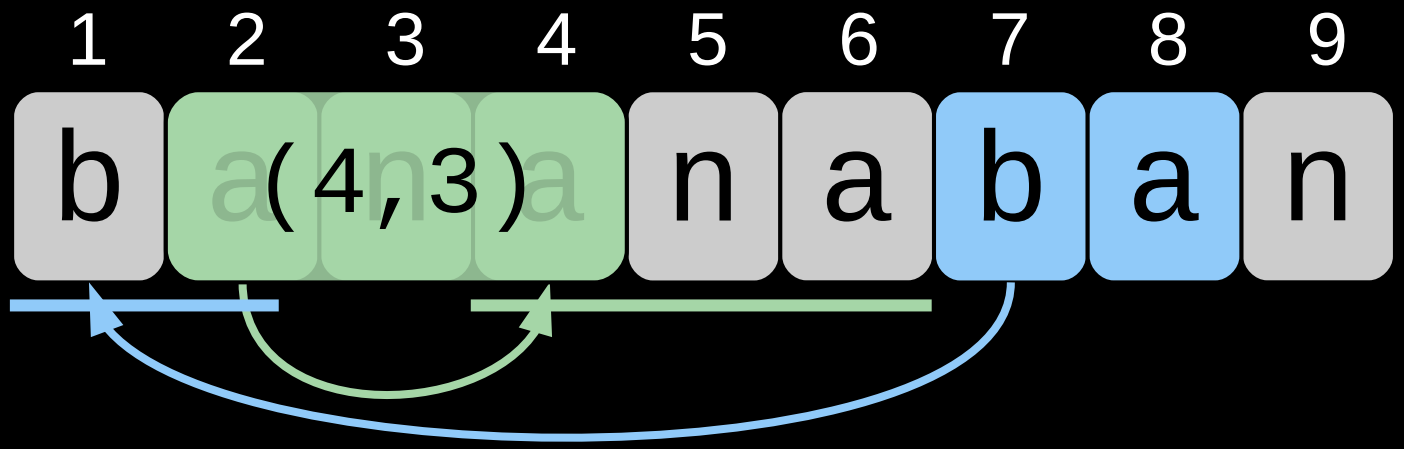
- 最長の繰り返し部分文字列を置換、
- 繰り返す



位置４から３文字をコピーし、

lcpcomp

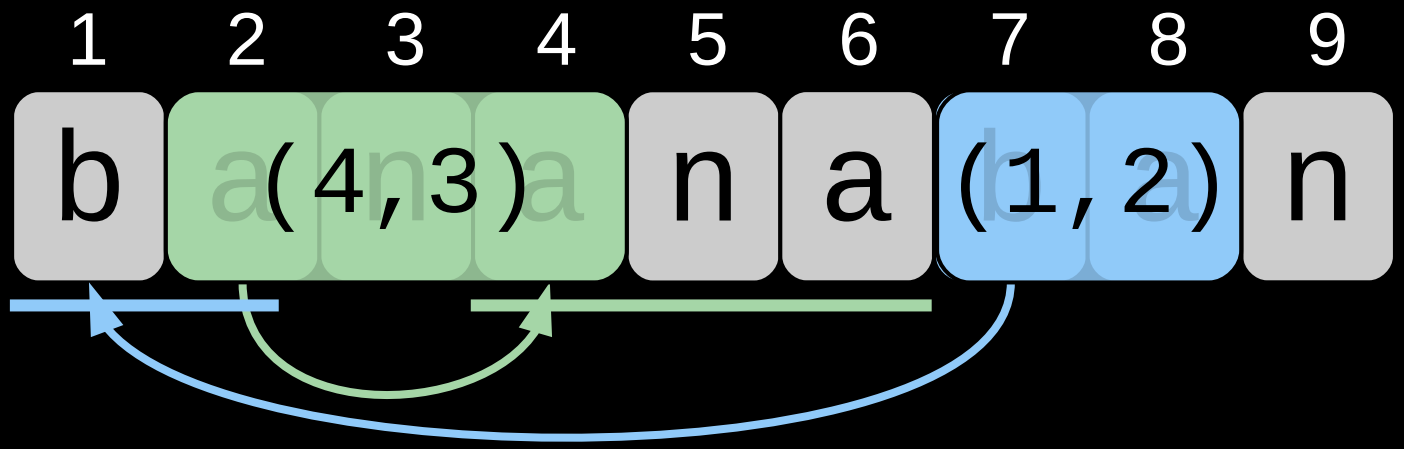
- 最長の繰り返し部分文字列を置換、
- 繰り返す



位置４から３文字をコピーし、
位置１から２文字をコピーする

lcpcomp

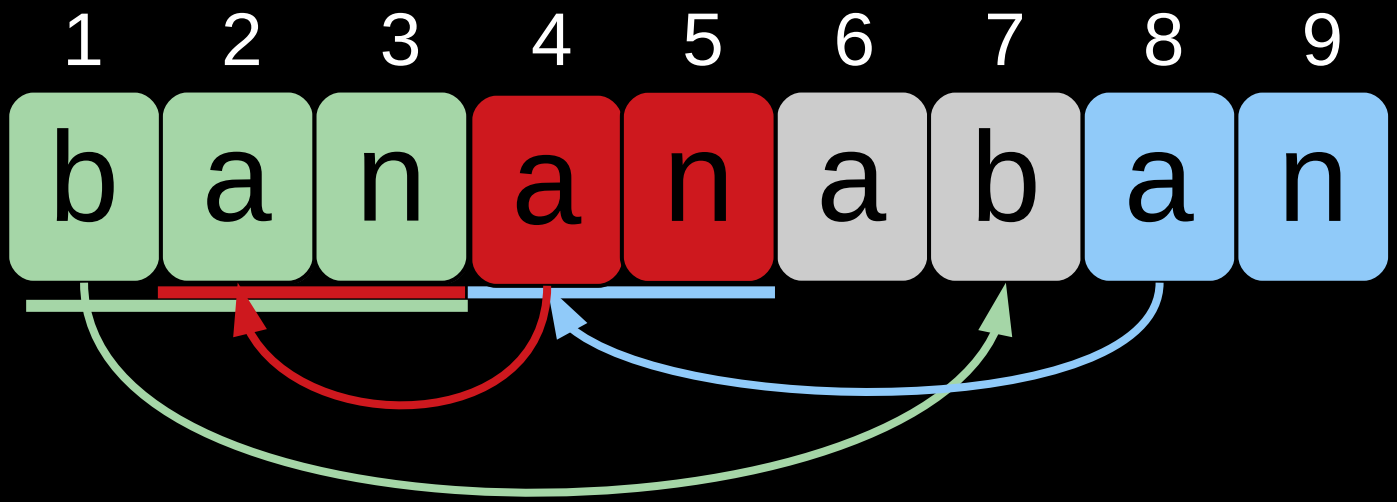
- 最長の繰り返し部分文字列を置換、
- 繰り返す



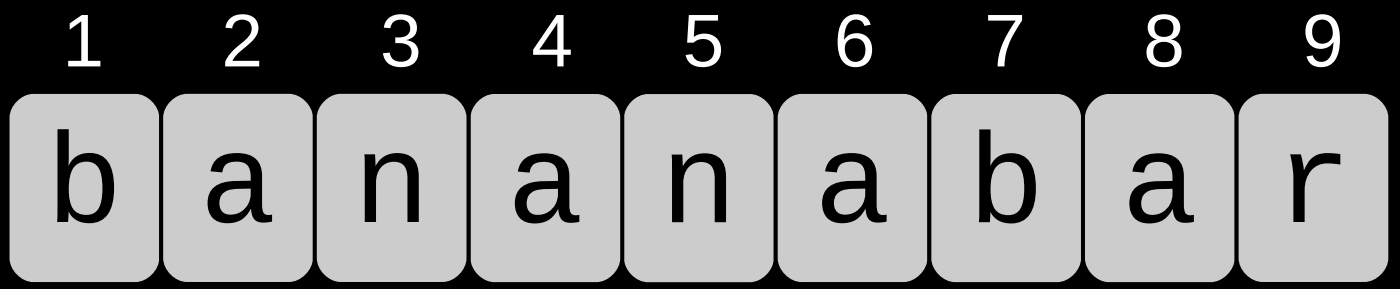
位置４から３文字をコピーし、
位置１から２文字をコピーする

復元

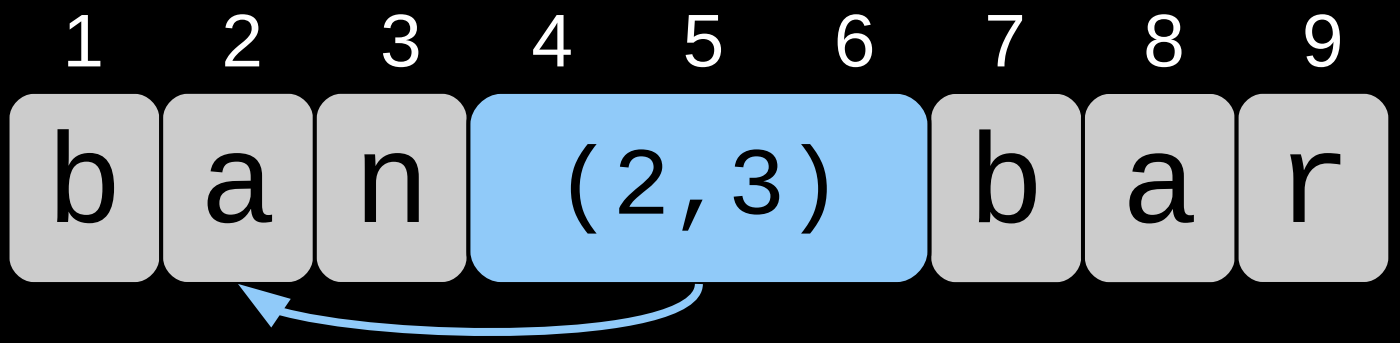
- cycle を防ぐためのルール
- 参照は辞書式順序の小さい接尾辞



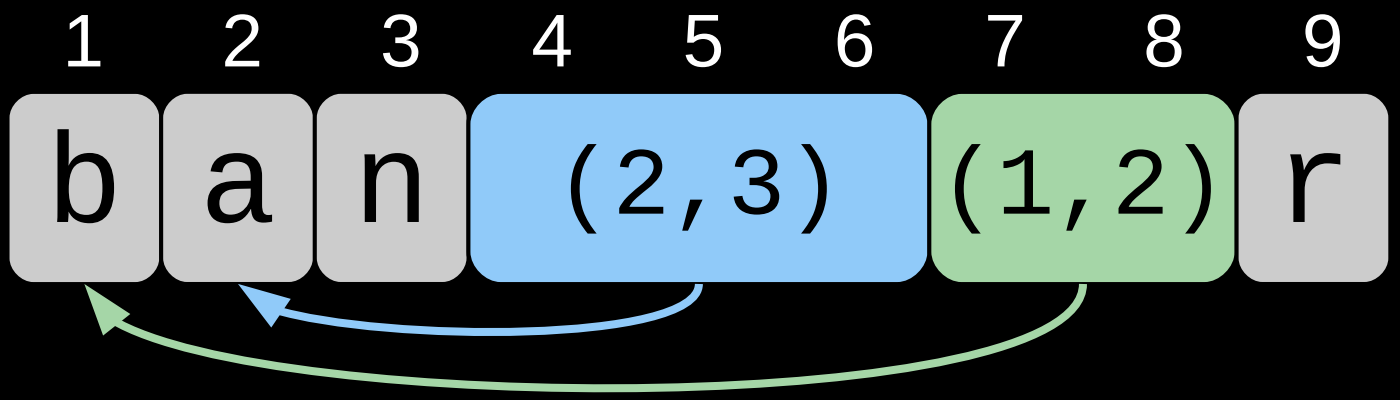
LZ77 に比べて



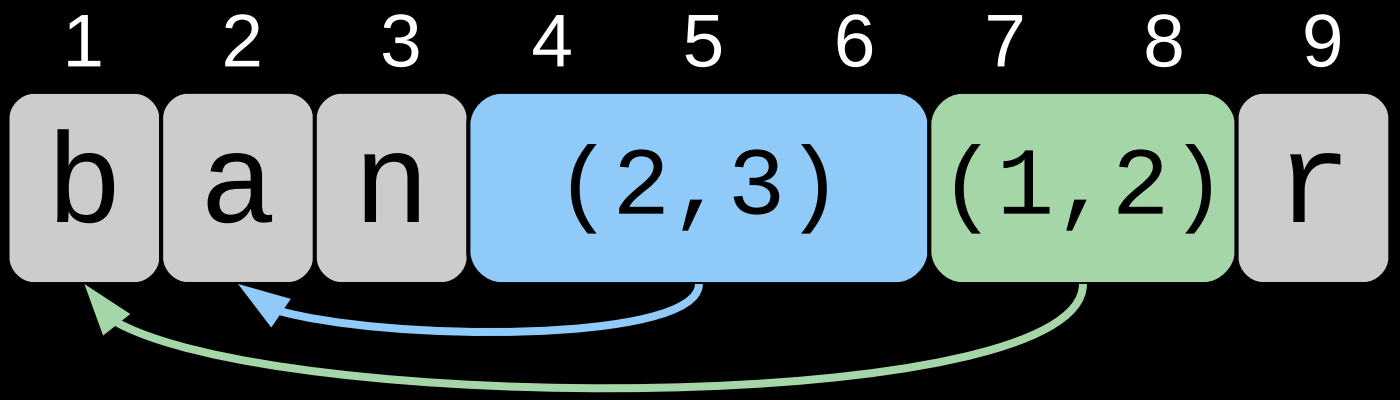
LZ77 に比べて



LZ77 に比べて



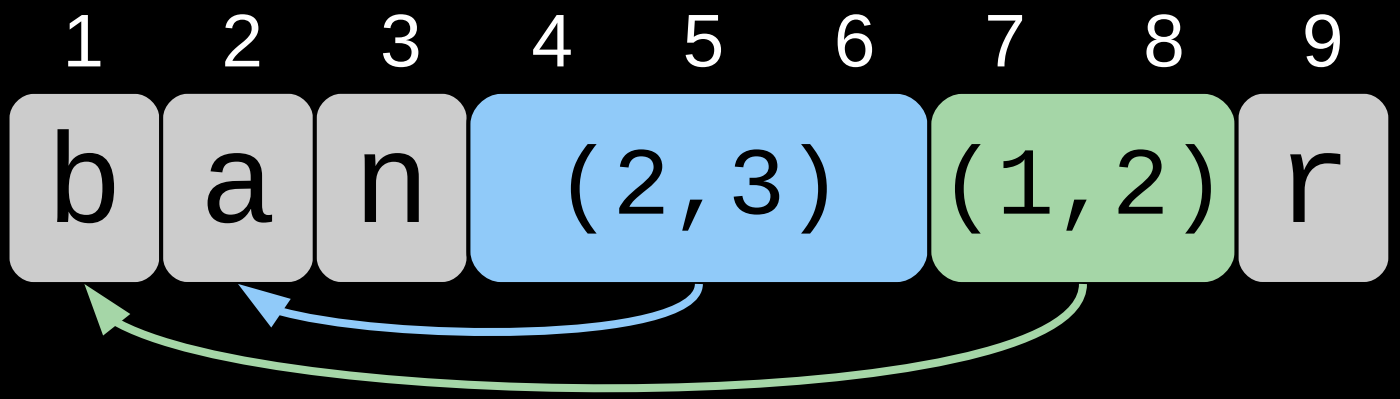
LZ77 に比べて



- LZ77

  参照はテキスト順序で前の位置

LZ77 に比べて



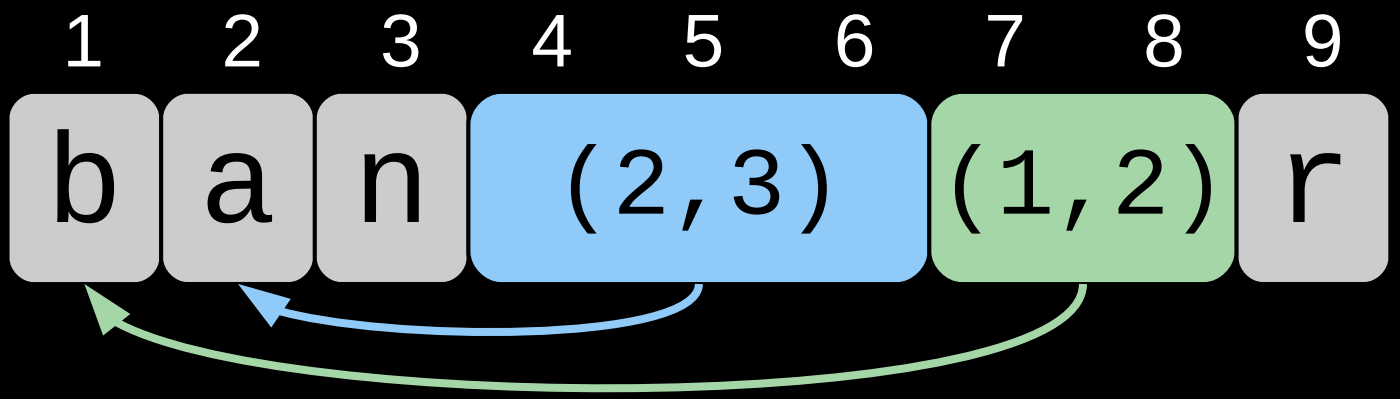
- LZ77

  参照はテキスト順序で前の位置

- lcpcomp

  参照は辞書式順序で前の接尾辞

LZ77 に比べて



- LZ77

  参照はテキスト順序で前の位置

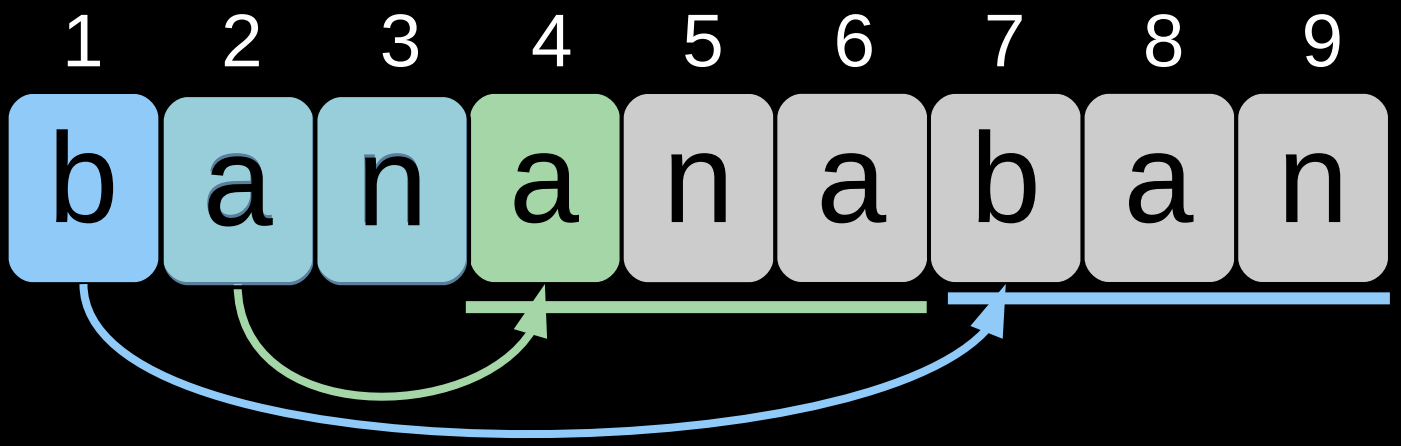
- lcpcomp

  参照は辞書式順序で前の接尾辞

⇒ どちらも greedy かつ cycle がない

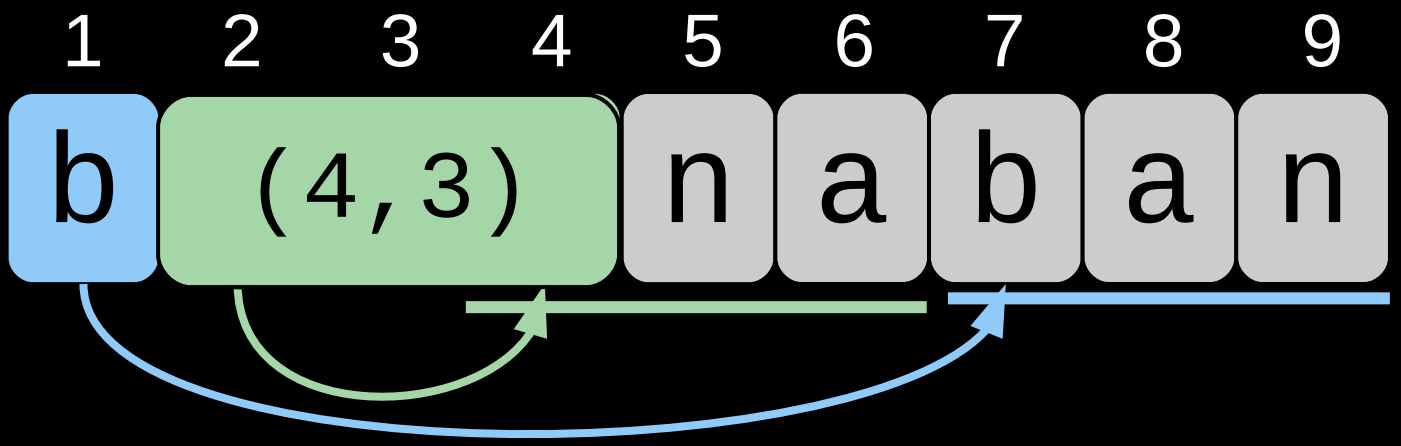
lcpcomp

- どの参照を作るか選択できる



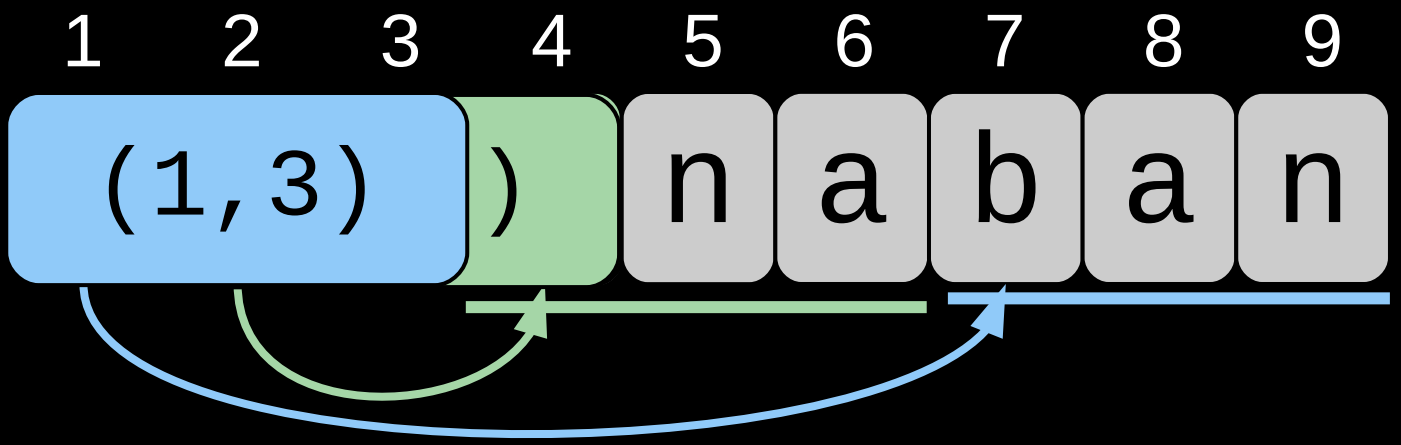
lcpcomp

- どの参照を作るか選択できる



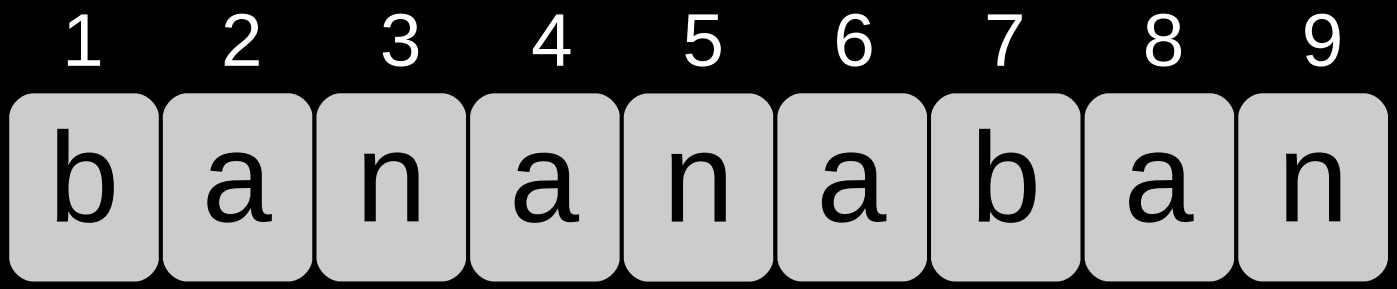
lcpcomp

- どの参照を作るか選択できる



plcpcomp

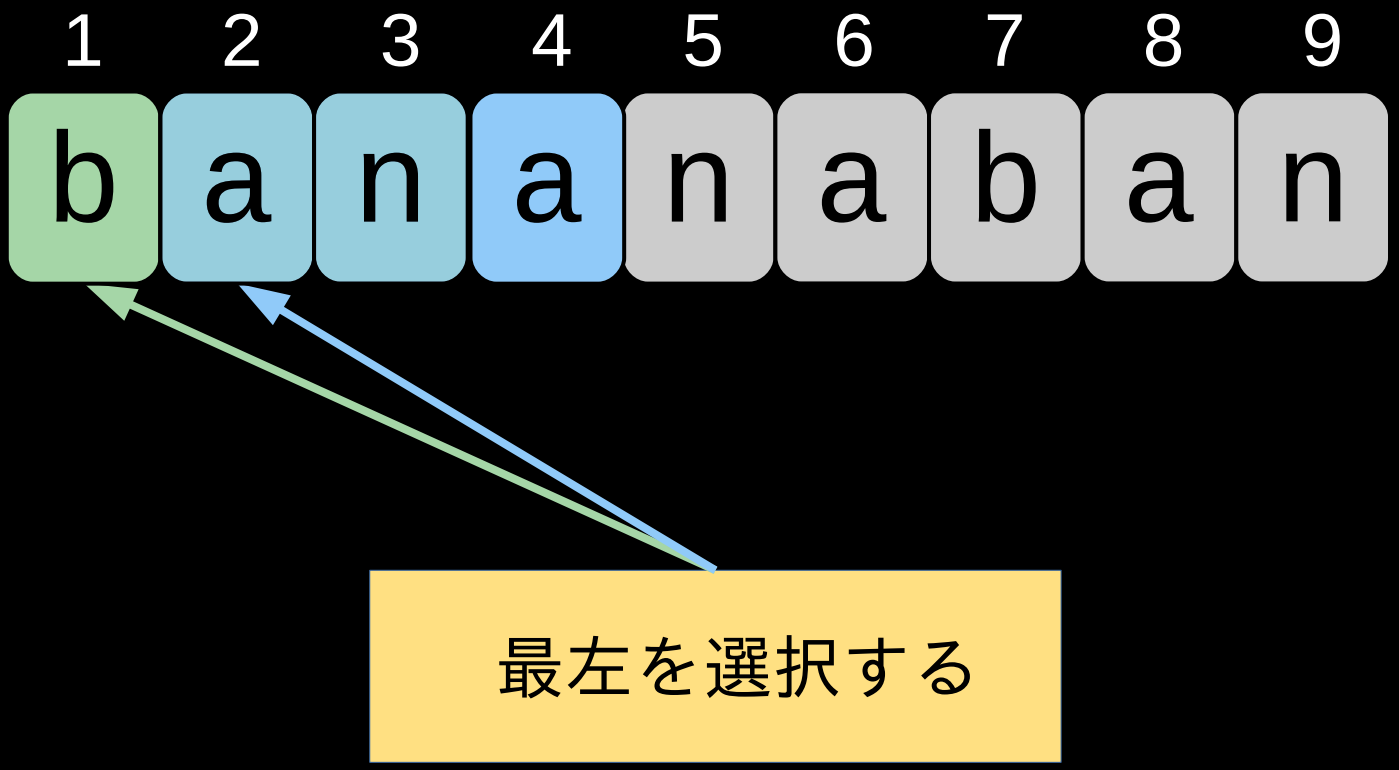
plcpcomp は今回の新しい提案方法



最左を選択する

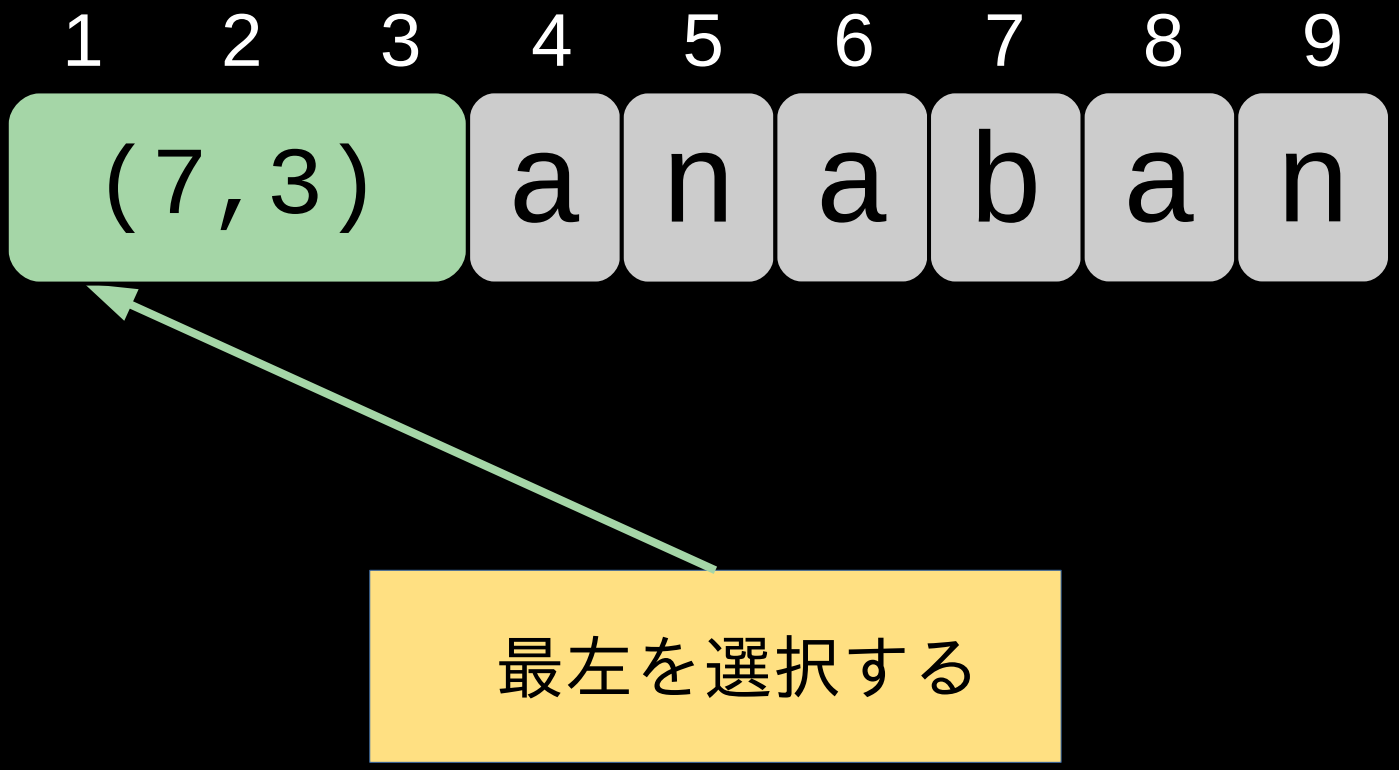
plcpcomp

plcpcomp は今回の新しい提案方法



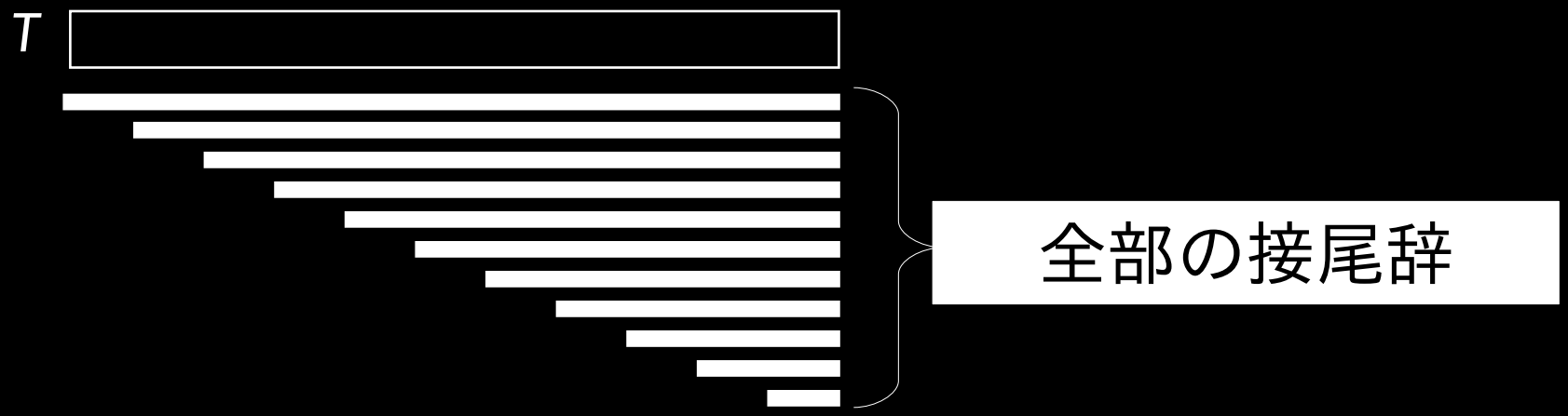
plcpcomp

plcpcomp は今回の新しい提案方法



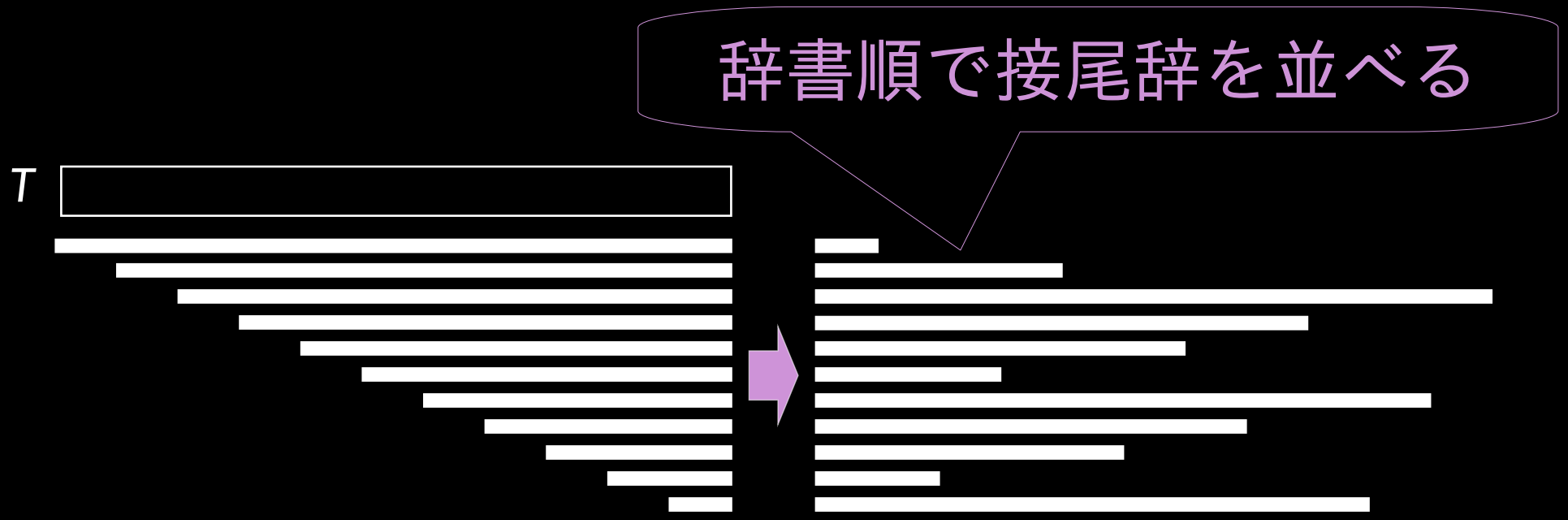
テキストデータ構造

テキストデータ構造



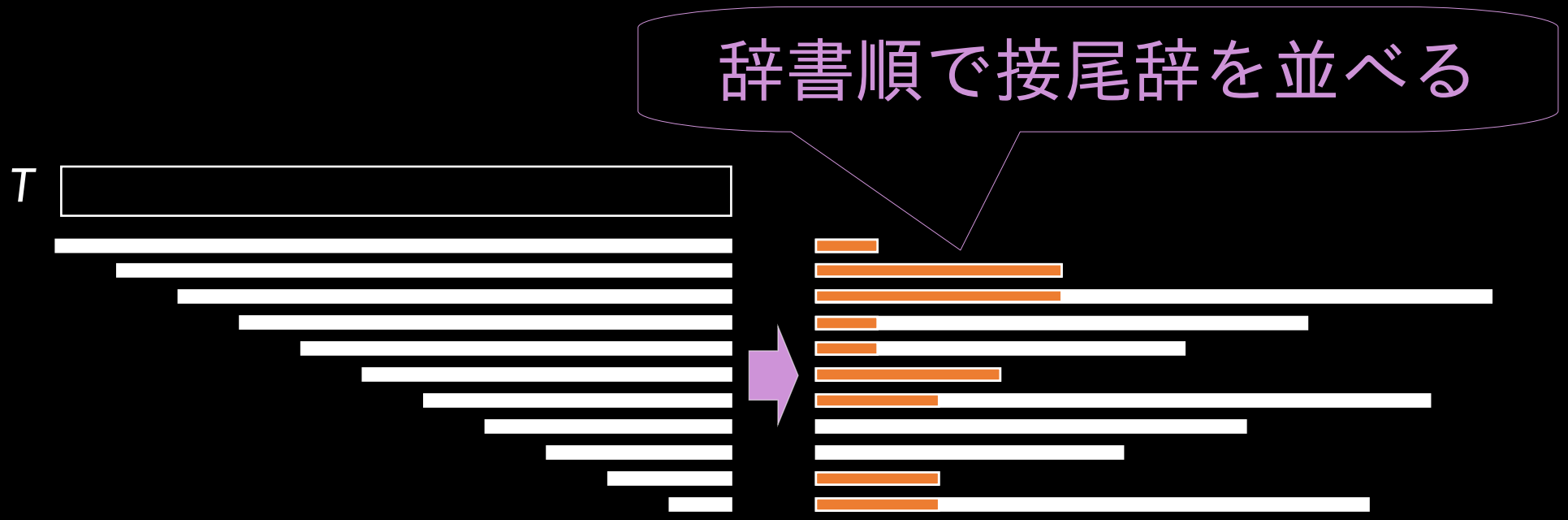
テキストデータ構造

- SA: 接尾辞配列



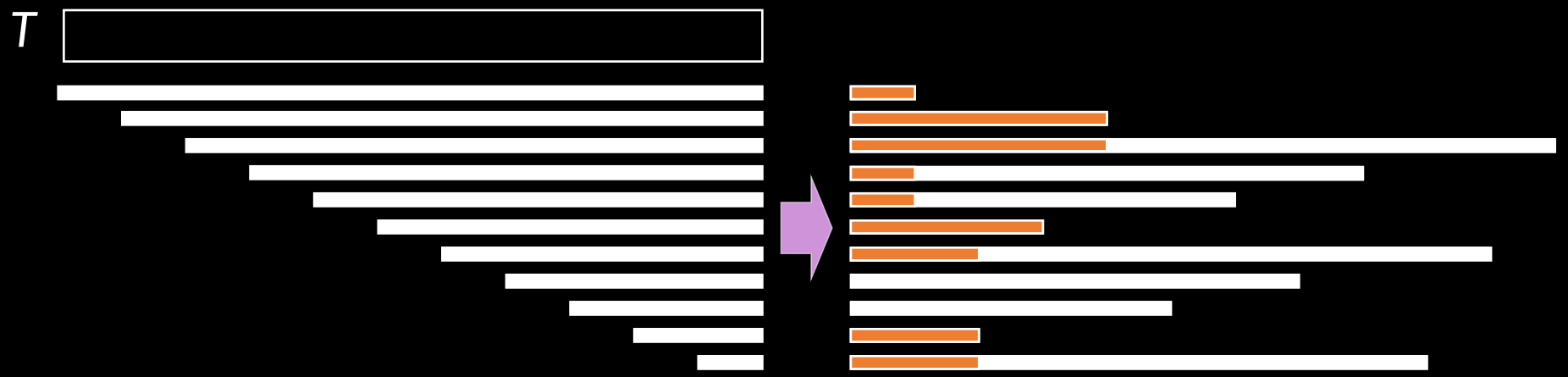
テキストデータ構造

- SA: 接尾辞配列
- LCP: 隣の接尾辞の最長一致の接頭辞



テキストデータ構造

- SA: 接尾辞配列
- LCP: 隣の接尾辞の最長一致の接頭辞
- PLCP[i]: T[i..] という接尾辞の LCP の長さ



どうやって？

目的

- 最長の繰り返し部分文字列 S を見つける
- S = ２つの接尾辞の LCP

そのために

- |S| = PLCP[i]
- S は　argmax PLCP[i] から出現する
- PLCP[i] は i 段目の接尾辞と Φ[i] 段目の接尾辞の LCP の長さ

Φ 配列

Φ[i] := SA[ISA [i] - 1]

Φ	7	4	5	8	9	-	2	6	1
SA	6	8	4	2	7	1	9	5	3
ISA	6	4	9	3	8	1	5	2	7
	1	2	3	4	5	6	7	8	9
	b	a	n	a	n	a	b	a	n

Φ 配列

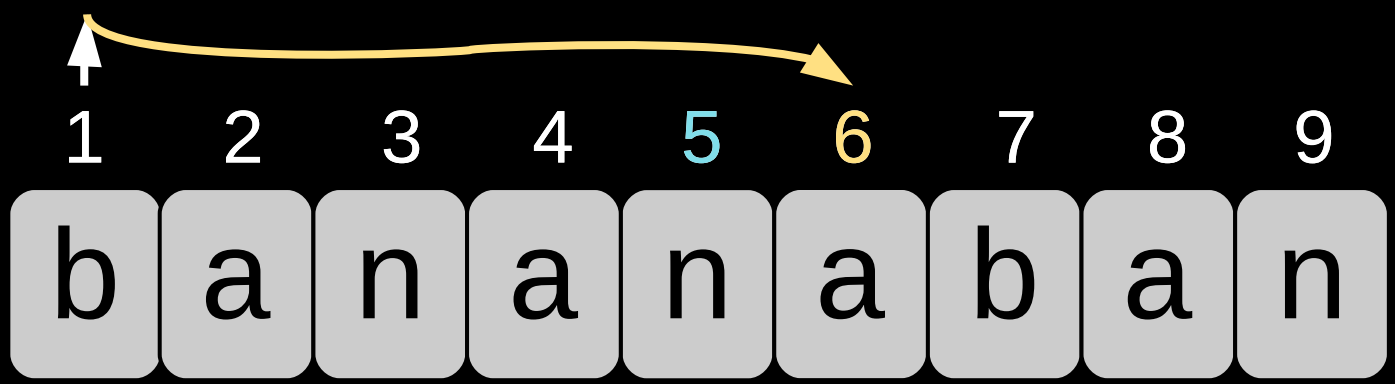
Φ[i] := SA[ISA [i] - 1]

Φ	7	4	5	8	9	-	2	6	1
SA	6	8	4	2	7	1	9	5	3
ISA	6	4	9	3	8	1	5	2	7
	1	2	3	4	5	6	7	8	9
	b	a	n	a	n	a	b	a	n

Φ 配列

Φ[i] := SA[ISA [i] - 1]

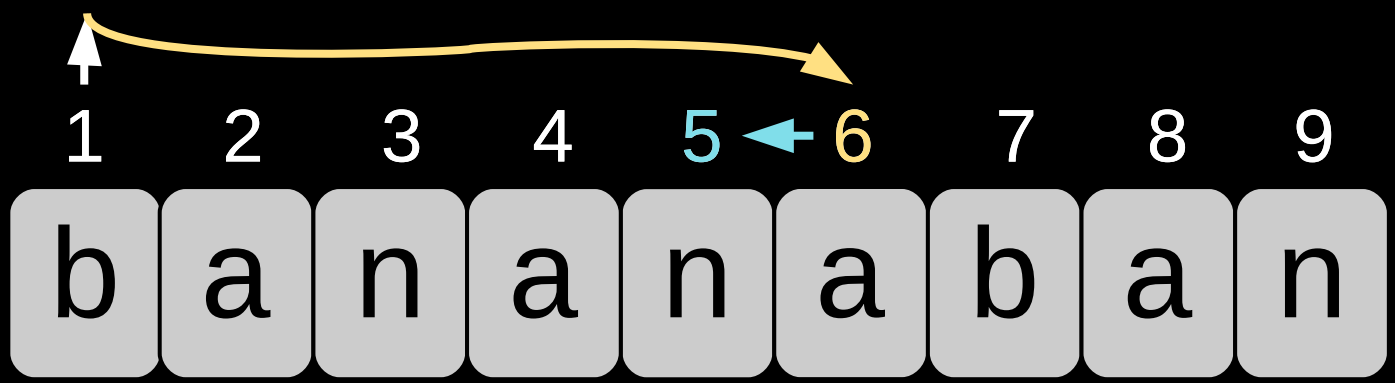
Φ	7	4	5	8	9	-	2	6	1
SA	6	8	4	2	7	1	9	5	3
ISA	6	4	9	3	8	1	5	2	7



Φ 配列

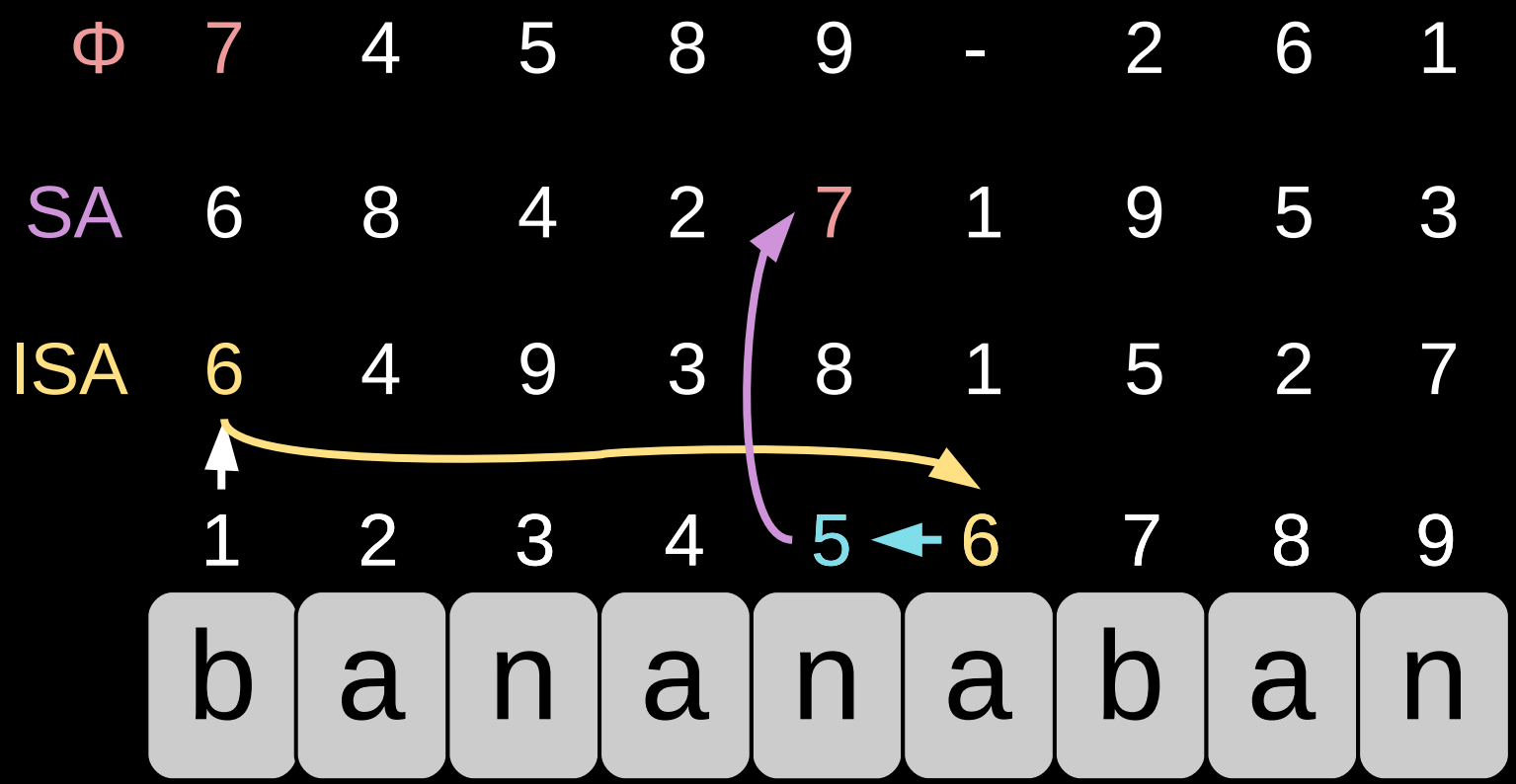
Φ[i] := SA[ISA [i] - 1]

Φ	7	4	5	8	9	-	2	6	1
SA	6	8	4	2	7	1	9	5	3
ISA	6	4	9	3	8	1	5	2	7



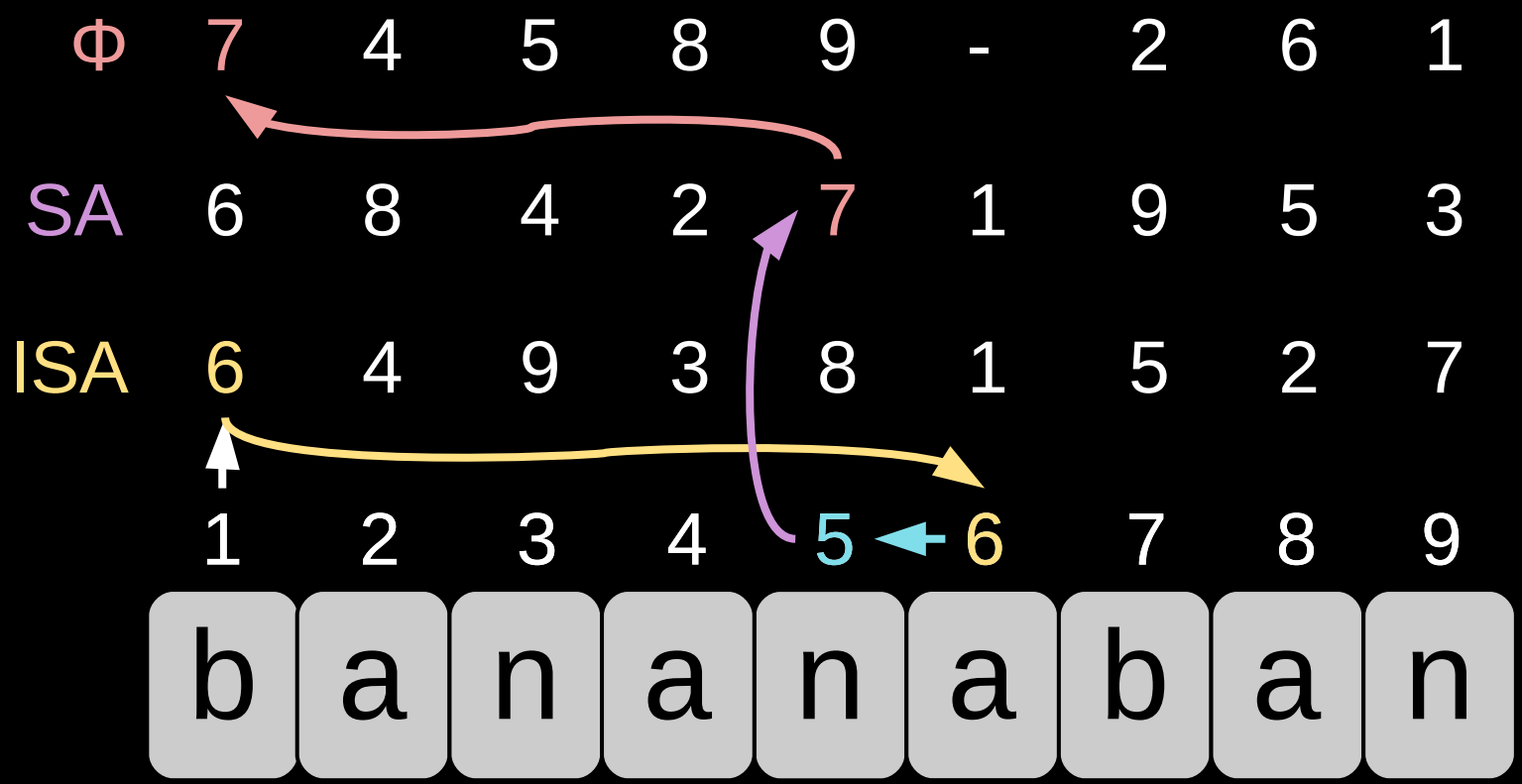
Φ 配列

Φ[i] := SA[ISA [i] - 1]



Φ 配列

Φ[i] := SA[ISA [i] - 1]

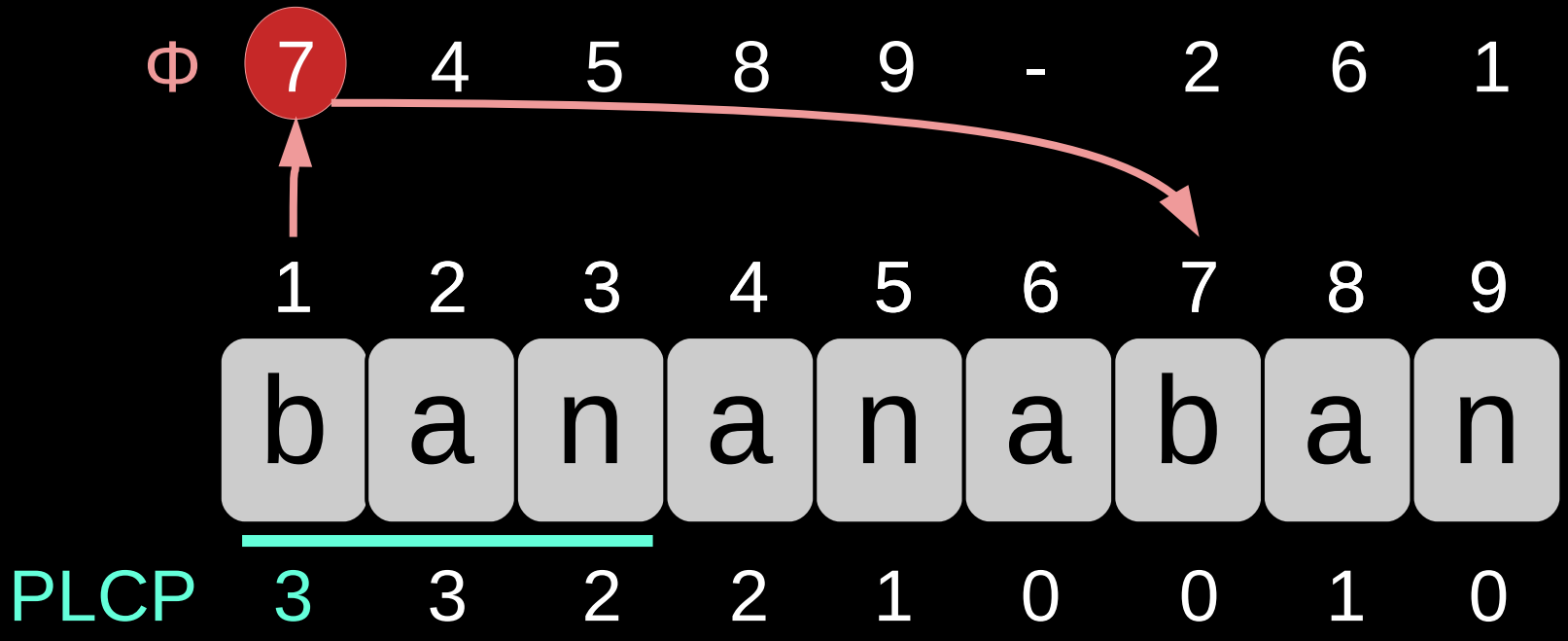


アルゴリズムの方針

Φ	7	4	5	8	9	-	2	6	1
	1	2	3	4	5	6	7	8	9
	b	a	n	a	n	a	b	a	n
PLCP	3	3	2	2	1	0	0	1	0

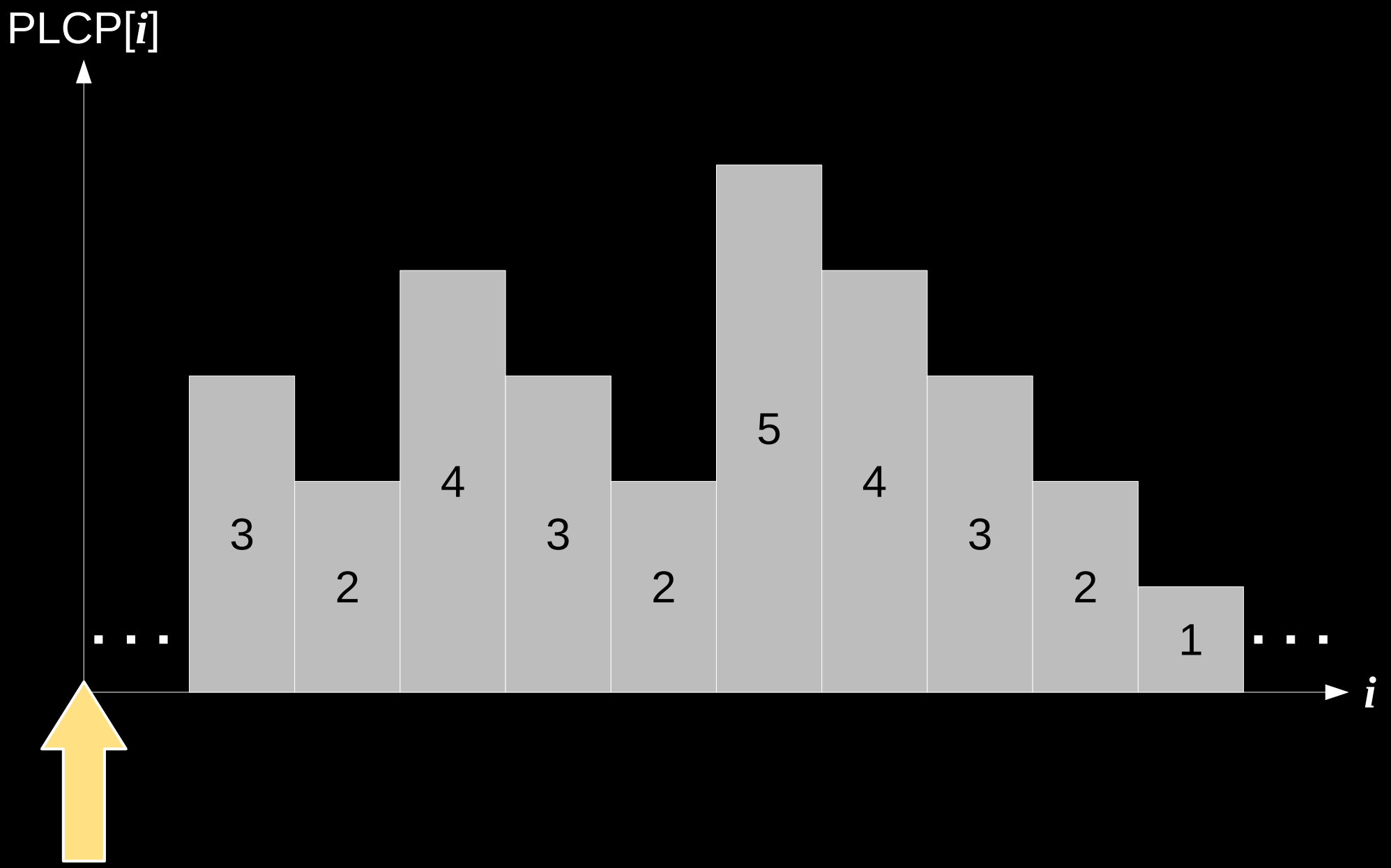
- PLCP: 参照の長さ
- Φ: 参照がどこから始まるか

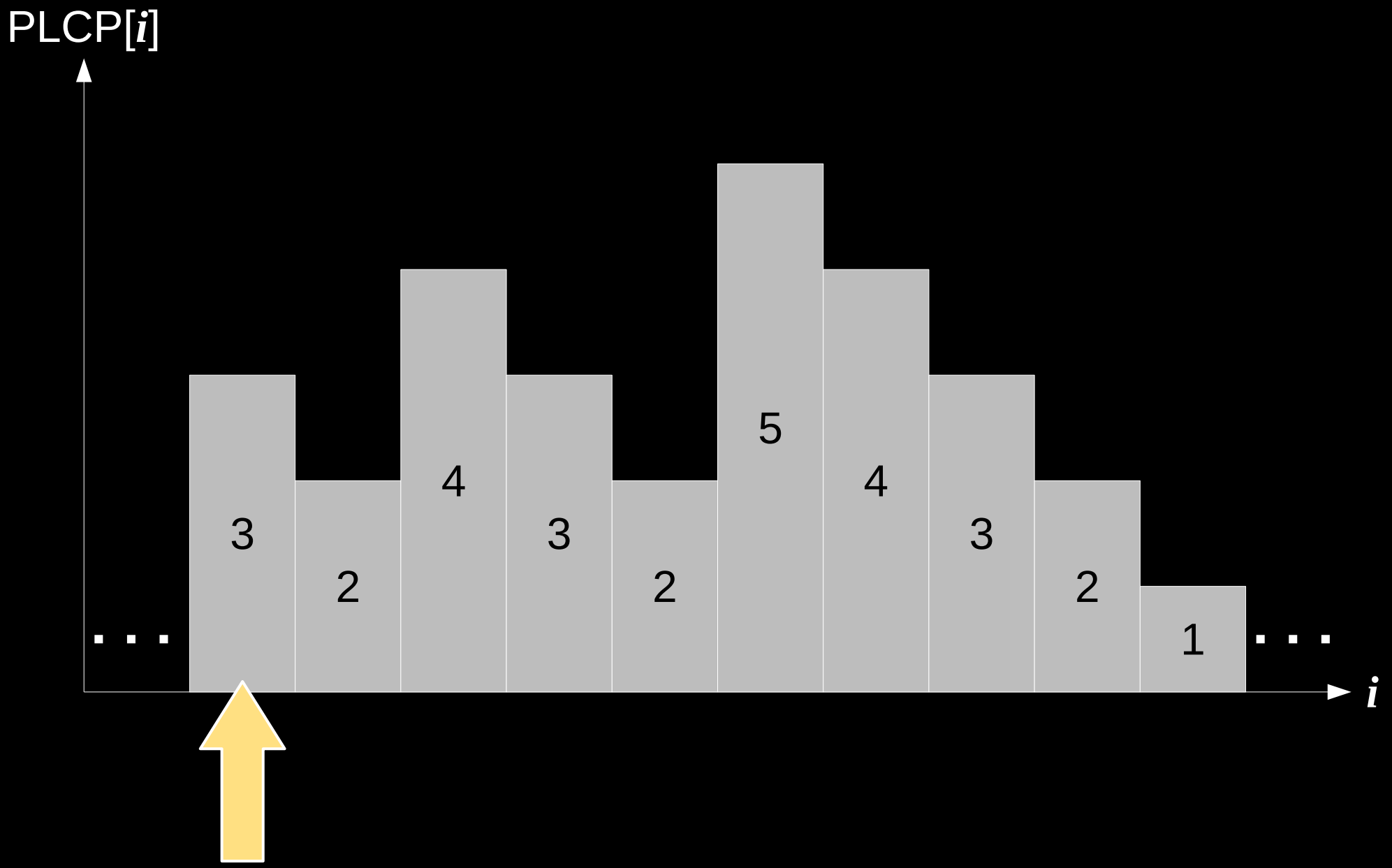
アルゴリズムの方針

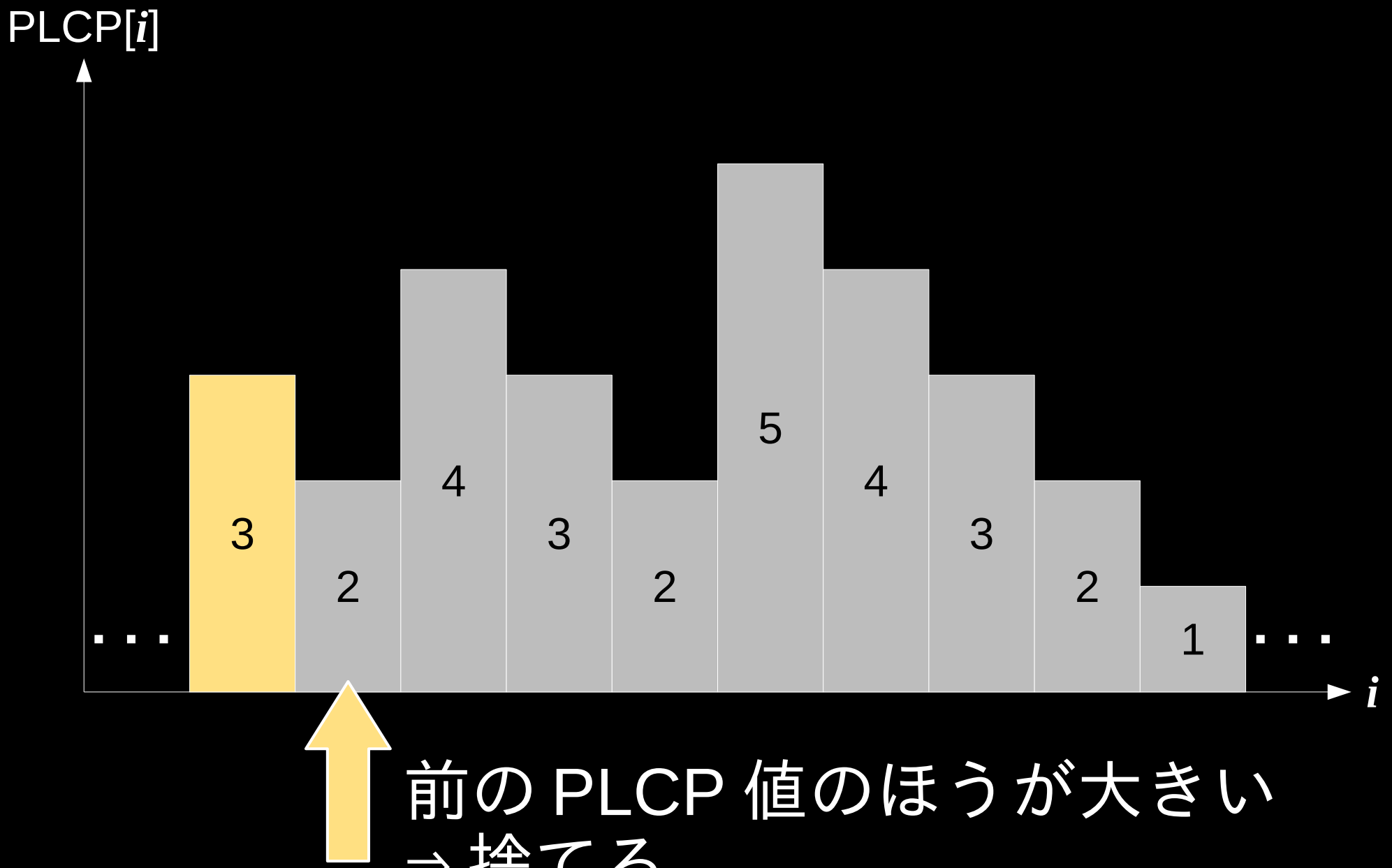


- PLCP: 参照の長さ
- Φ: 参照がどこから始まるか

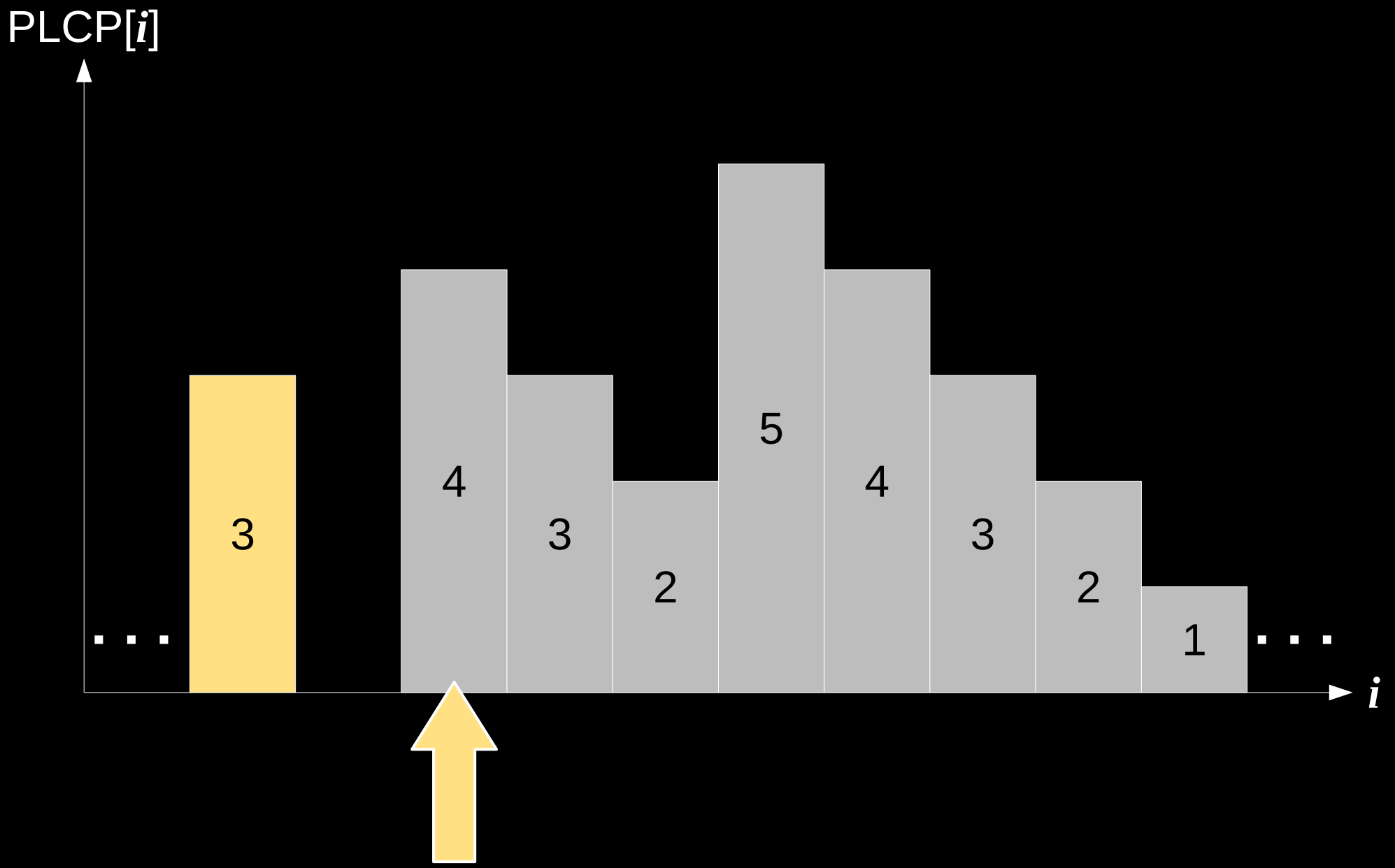
算出

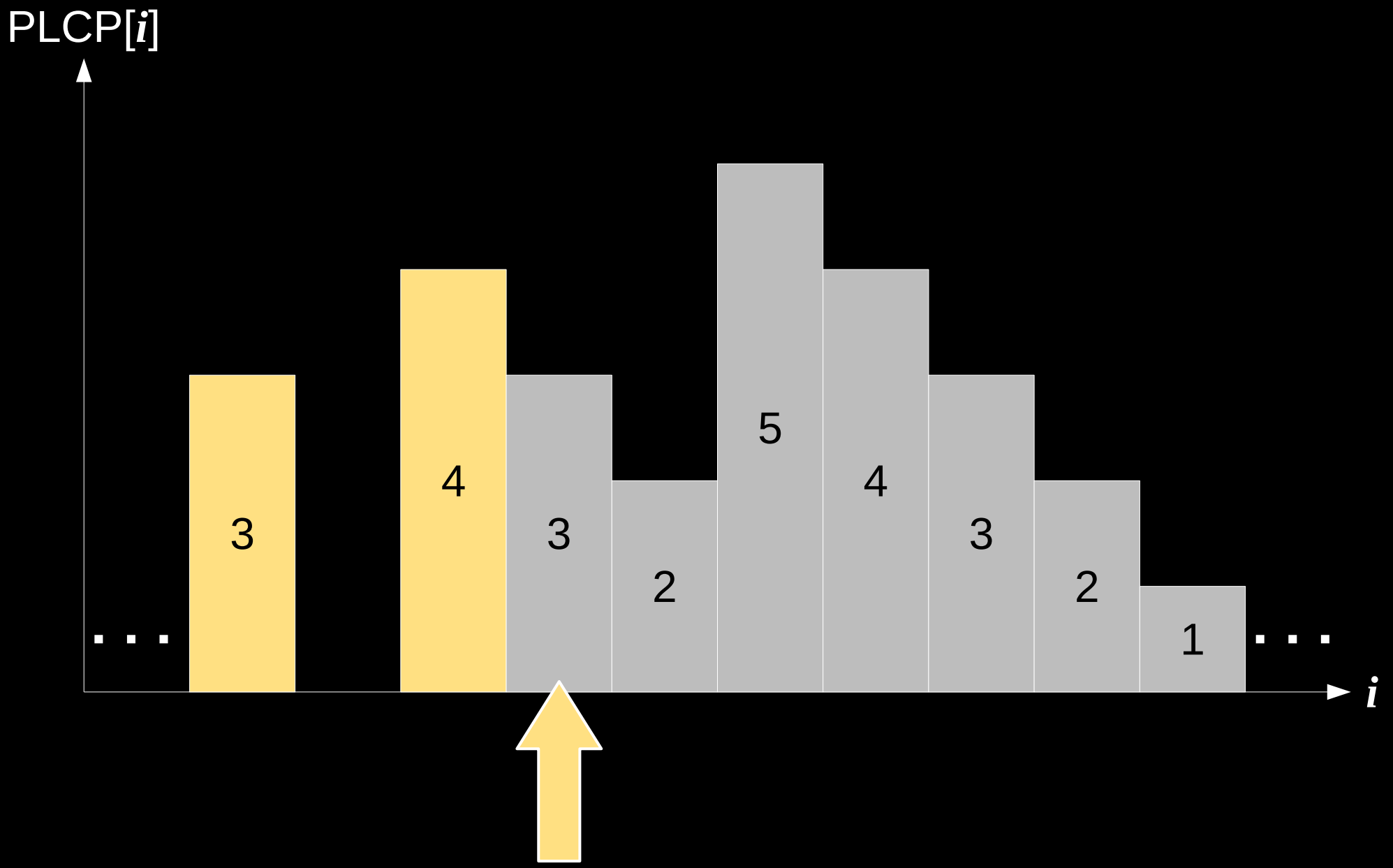


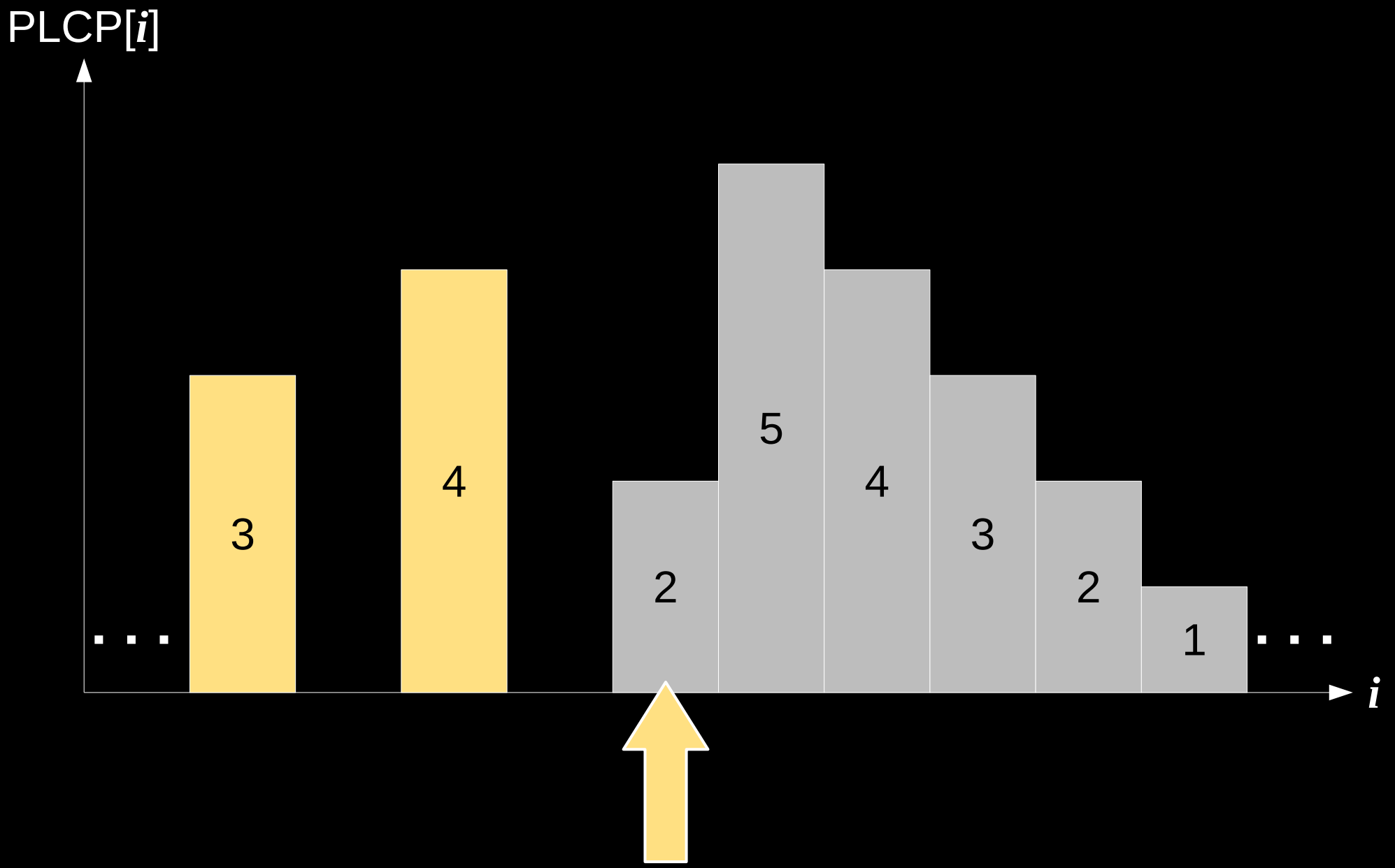

PLCP[i]
3
2
4
3
2
5
4
3
2
1
i

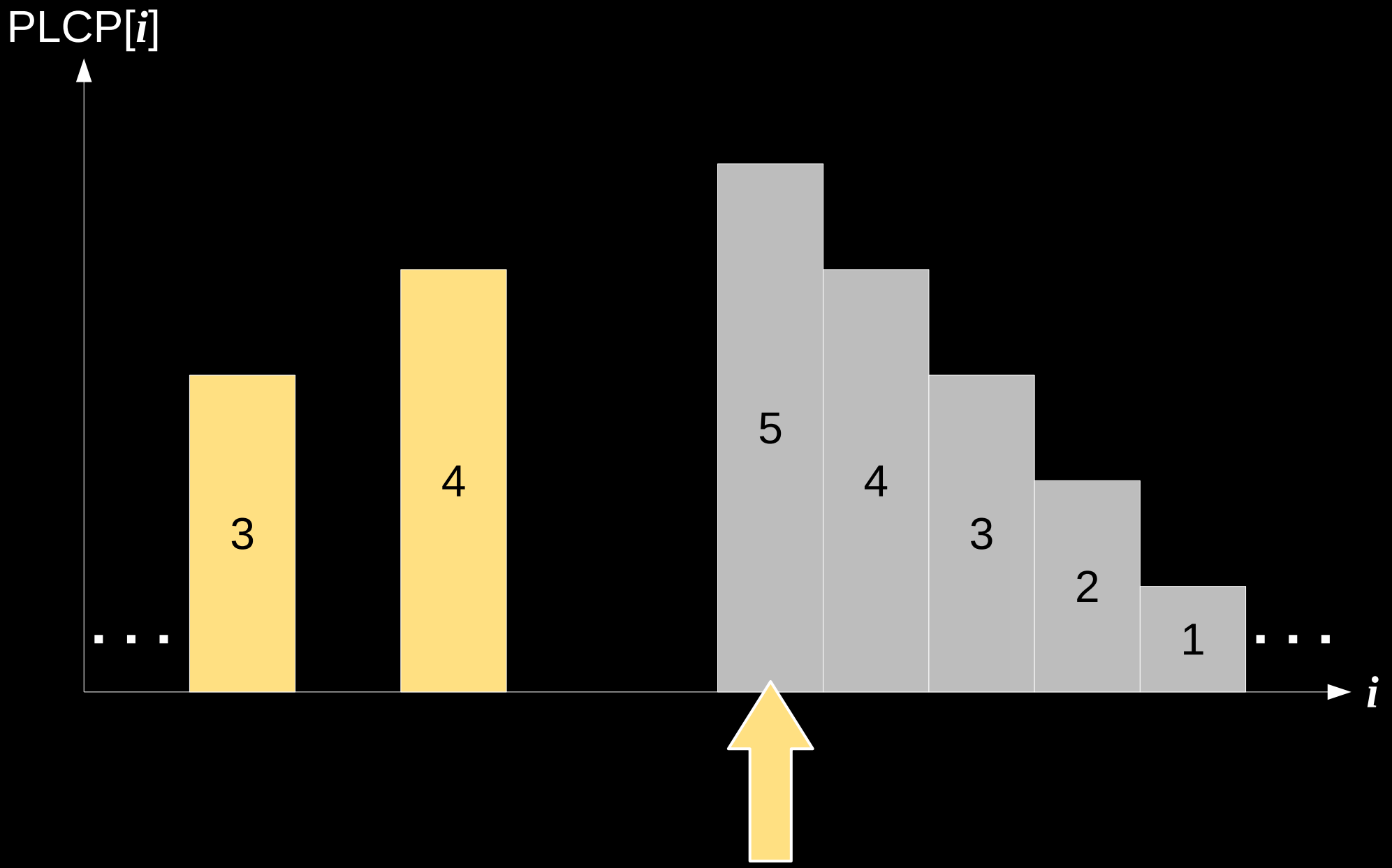


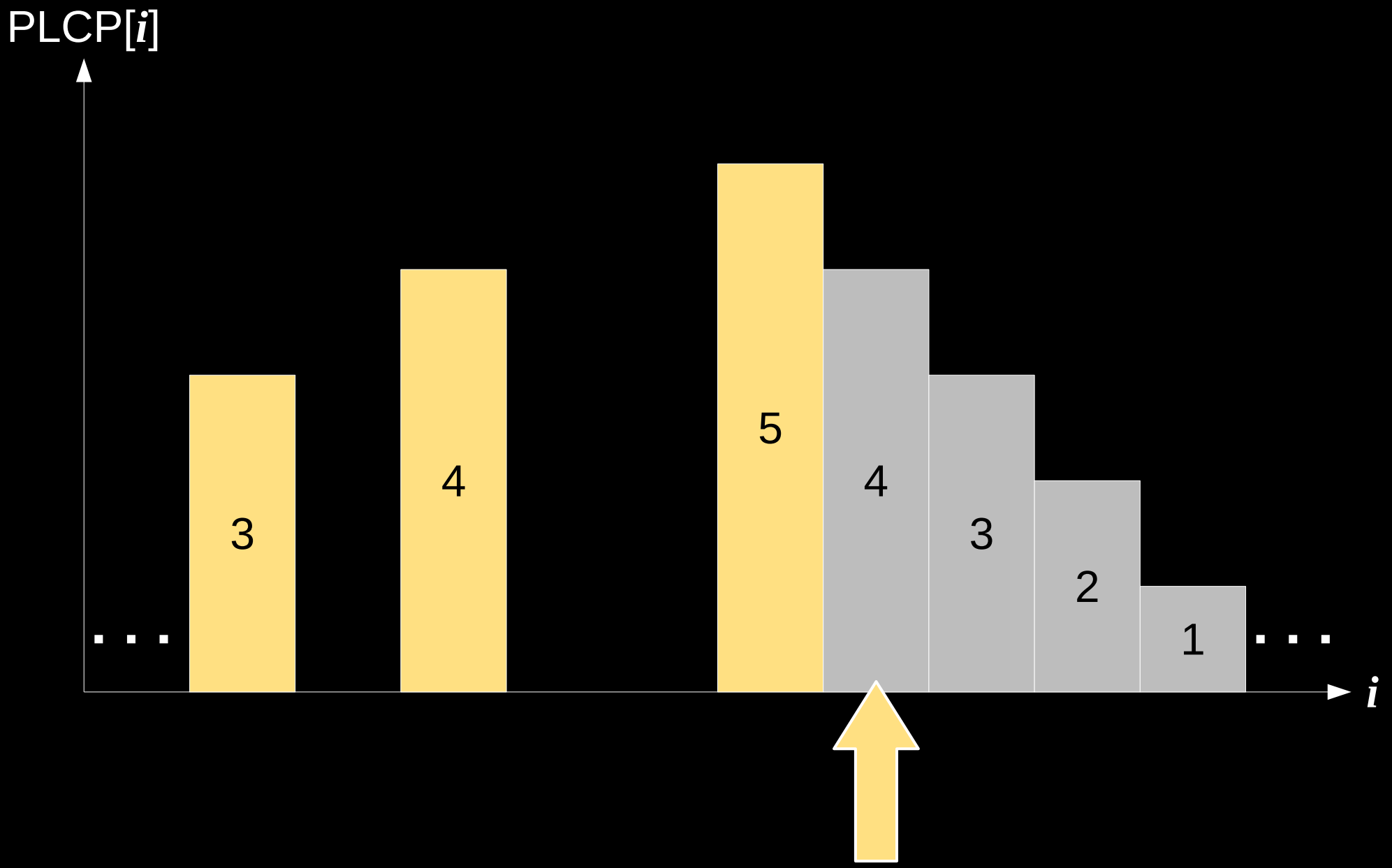
前の PLCP 値のほうが大きい
⇒ 捨てる

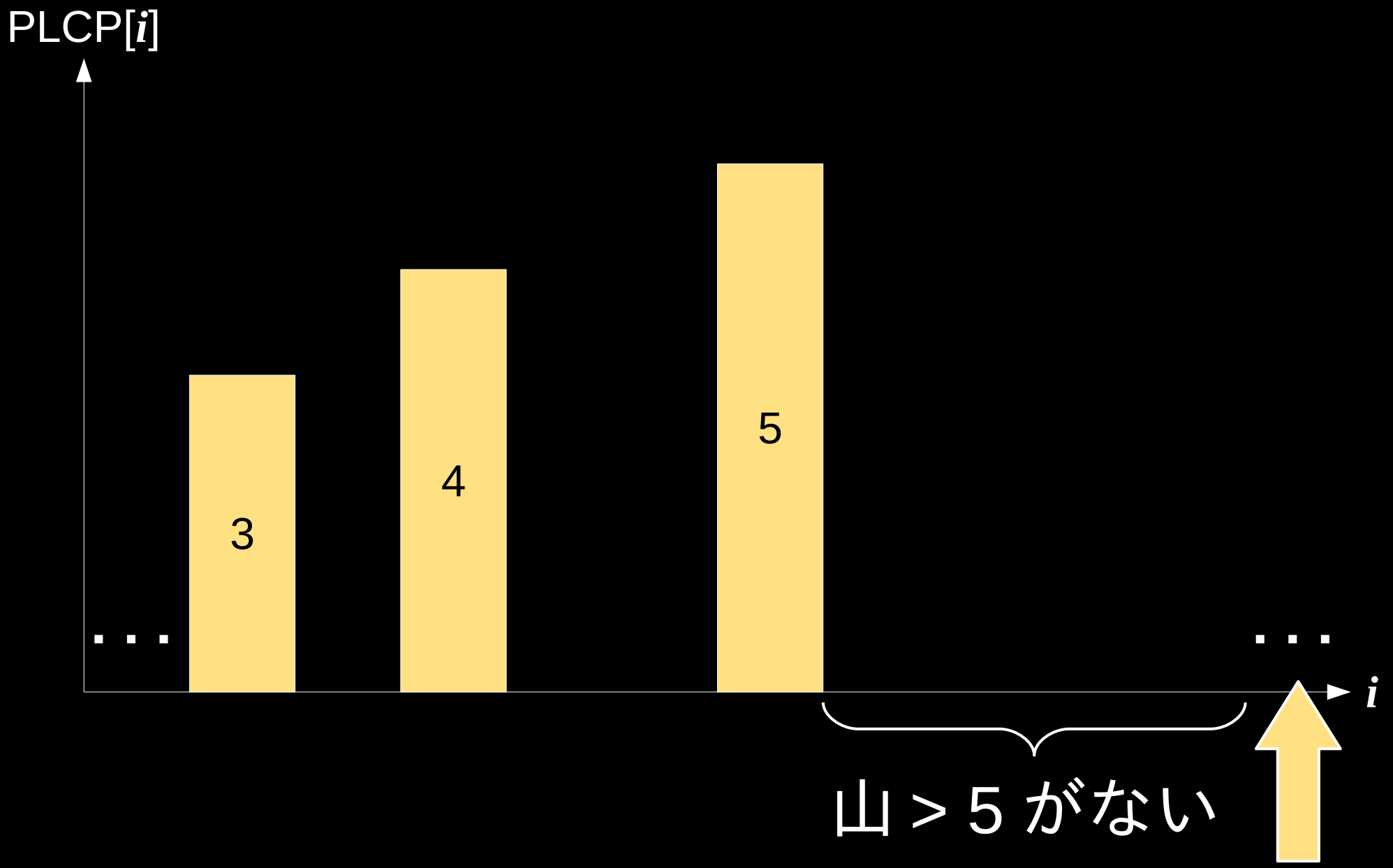

PLCP[i]
3
4
3
2
5
4
3
2
1
i

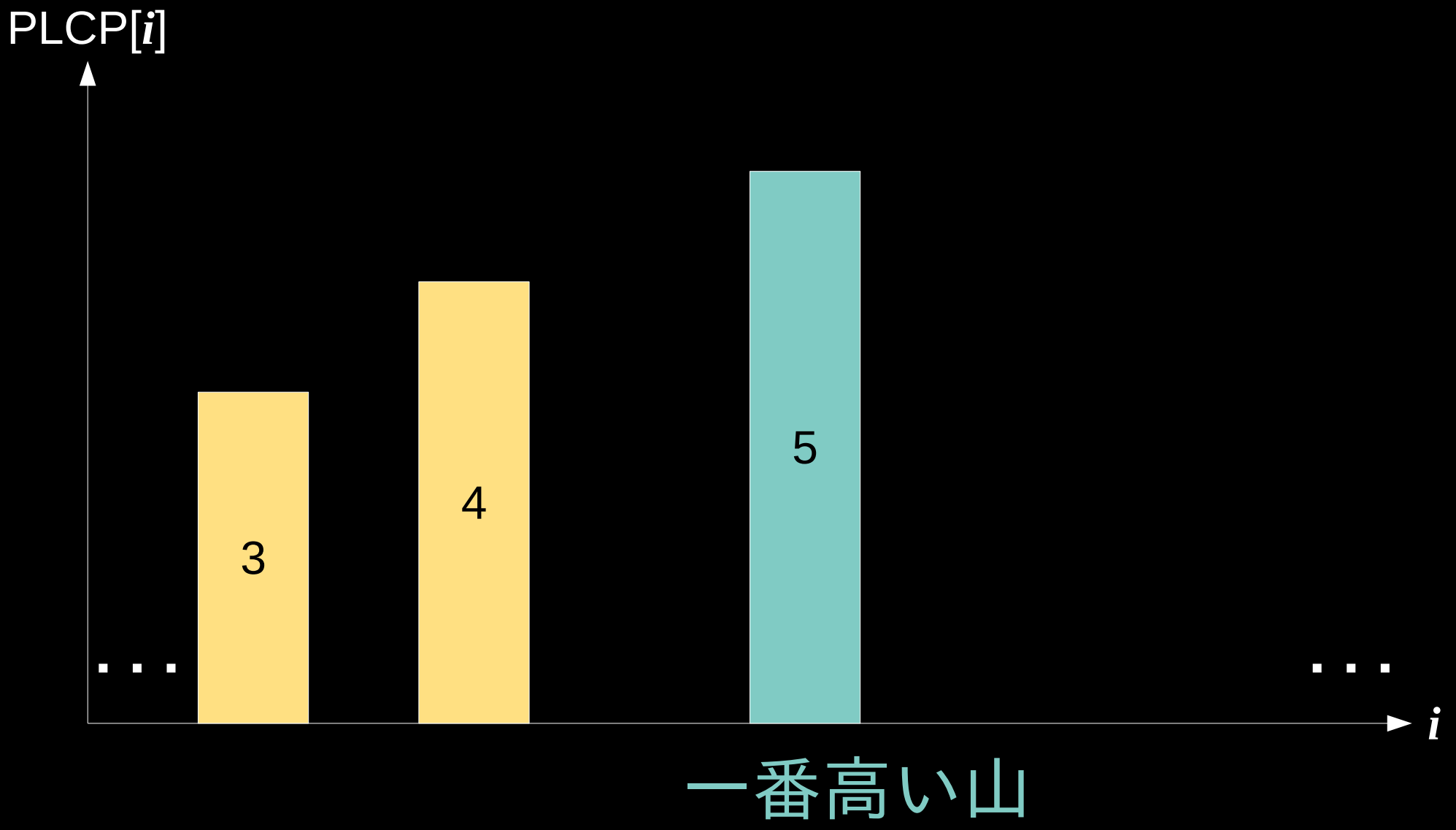

PLCP[i]
3
4
3
2
5
4
3
2
1
i

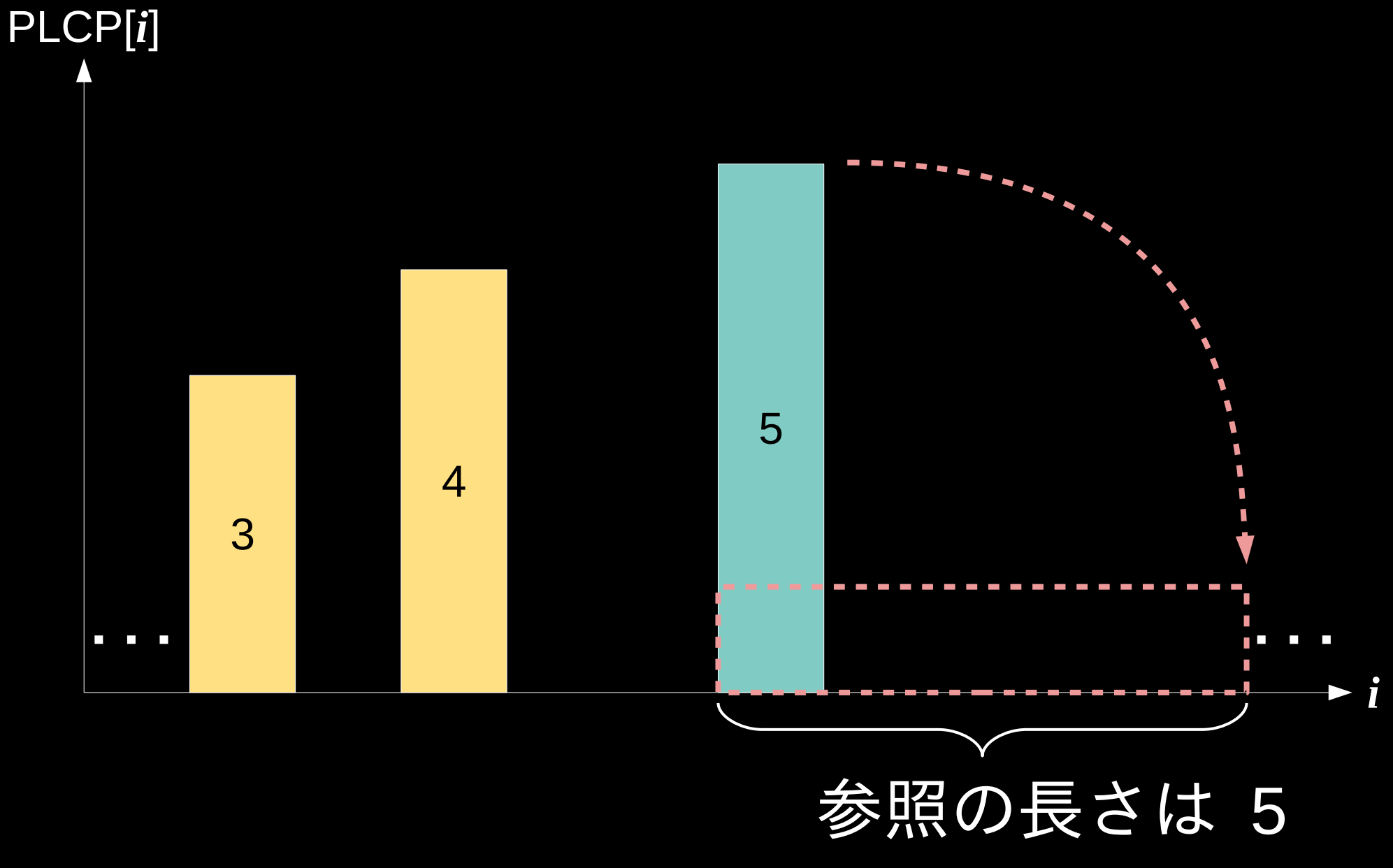

PLCP[i]
3
4
2
5
4
3
2
1
i

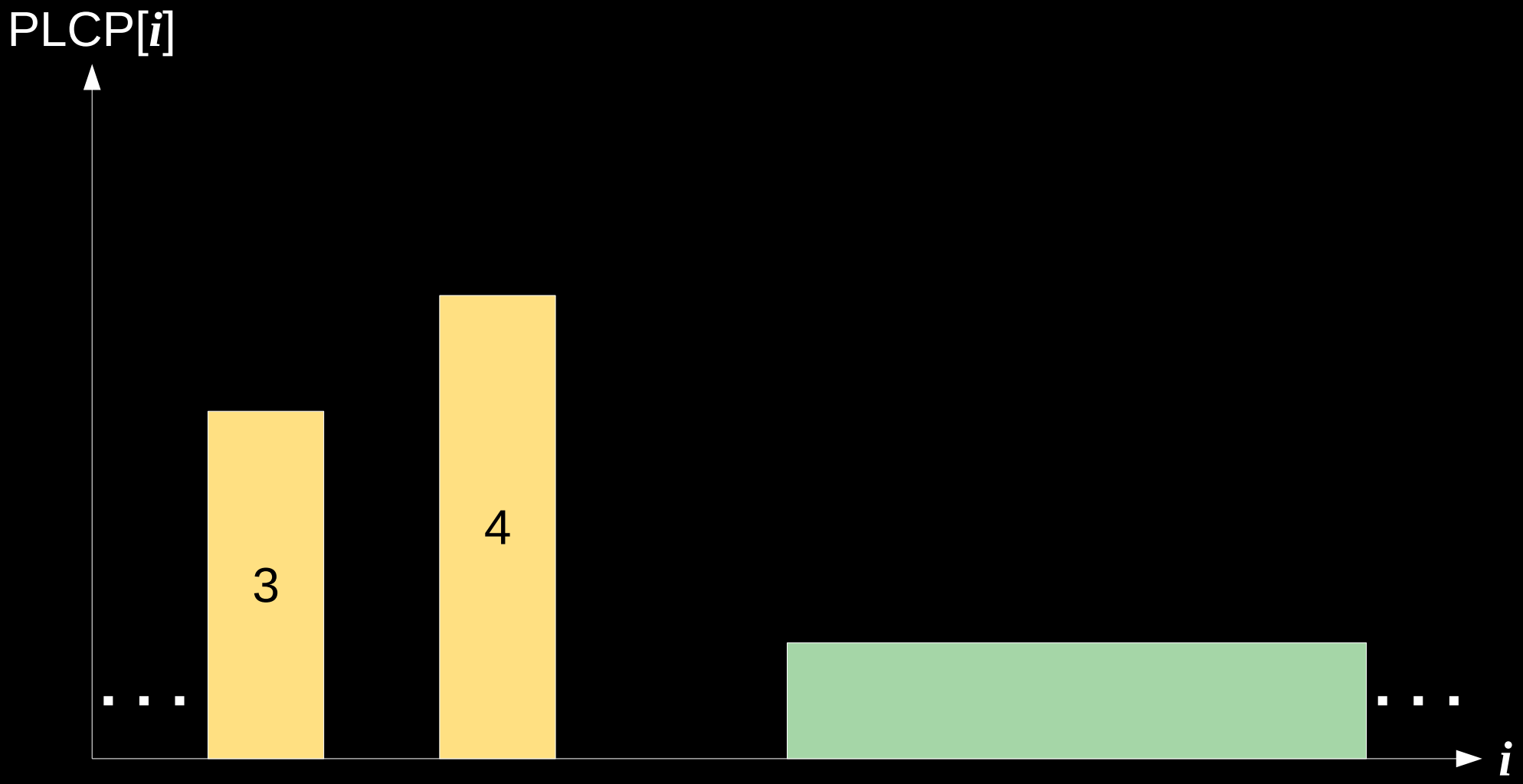

PLCP[i]
3
4
5
4
3
2
1
i


PLCP[i]
3
4
5
4
3
2
1
i

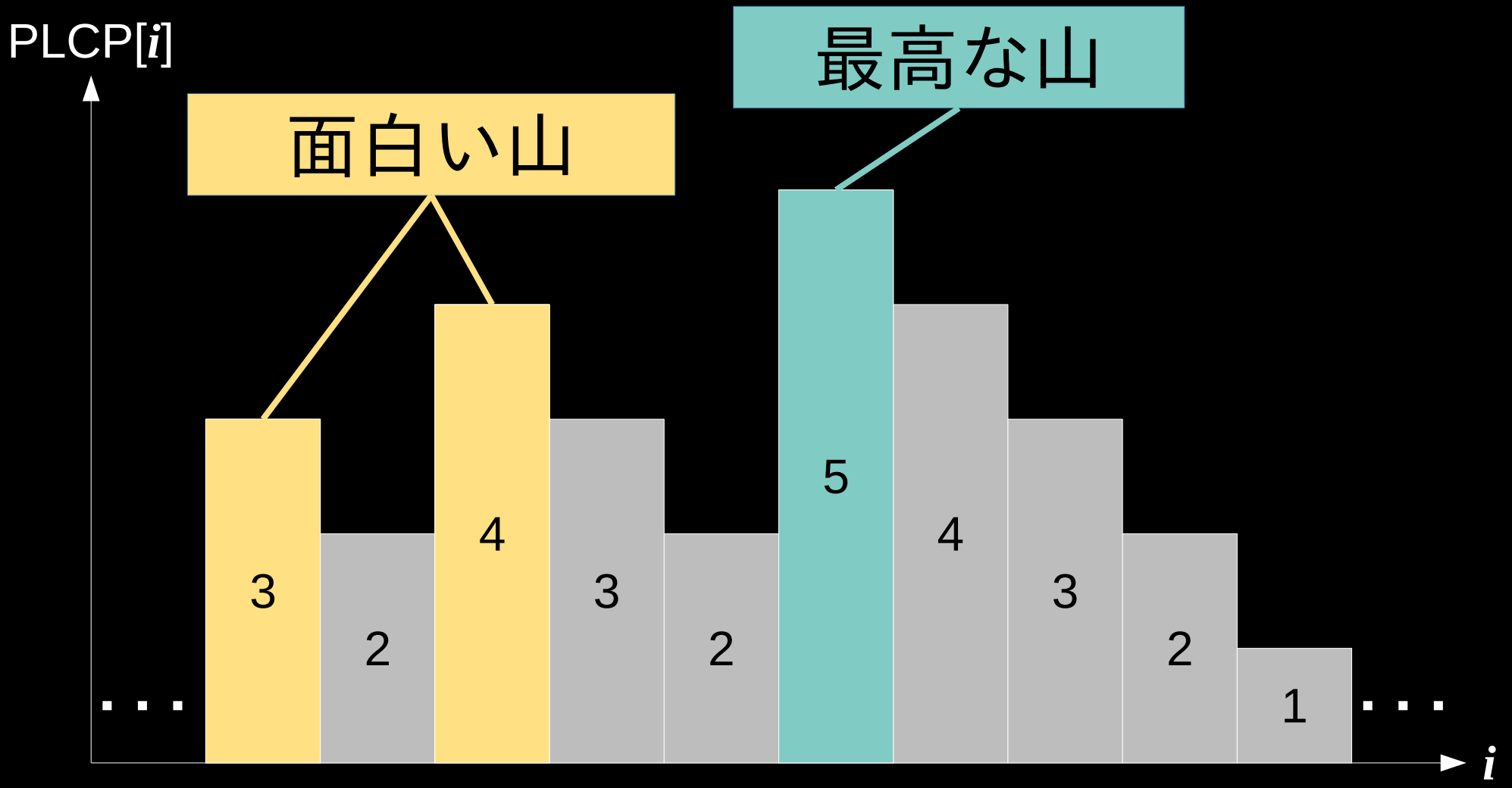

PLCP[i]
3
4
5
i
山 > 5 がない


PLCP[*i*]
3
4
5
. . .
. . .
i
一番高い山

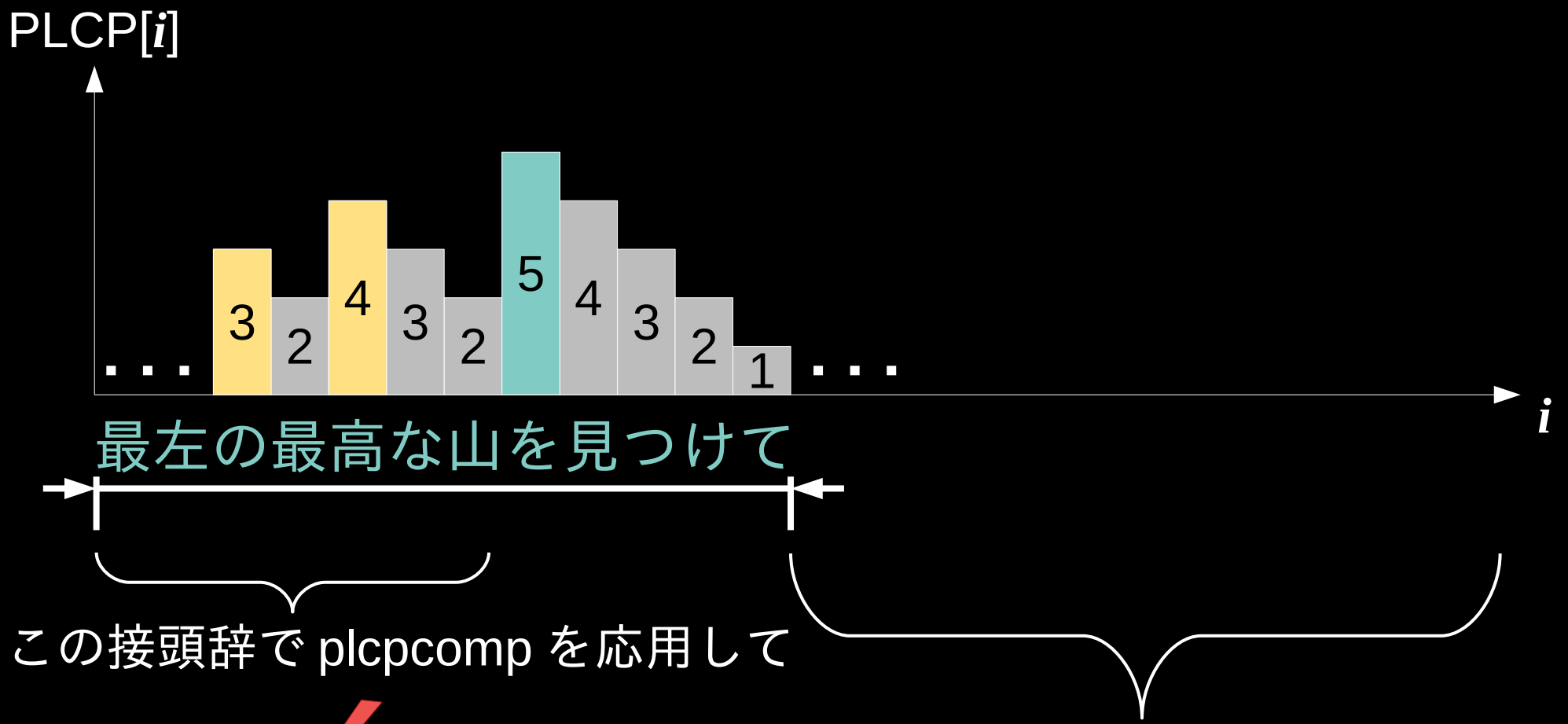

PLCP[i]
3
4
5
i
参照の長さは 5


PLCP[i]
3
4
i

2つの定義



繰り返して

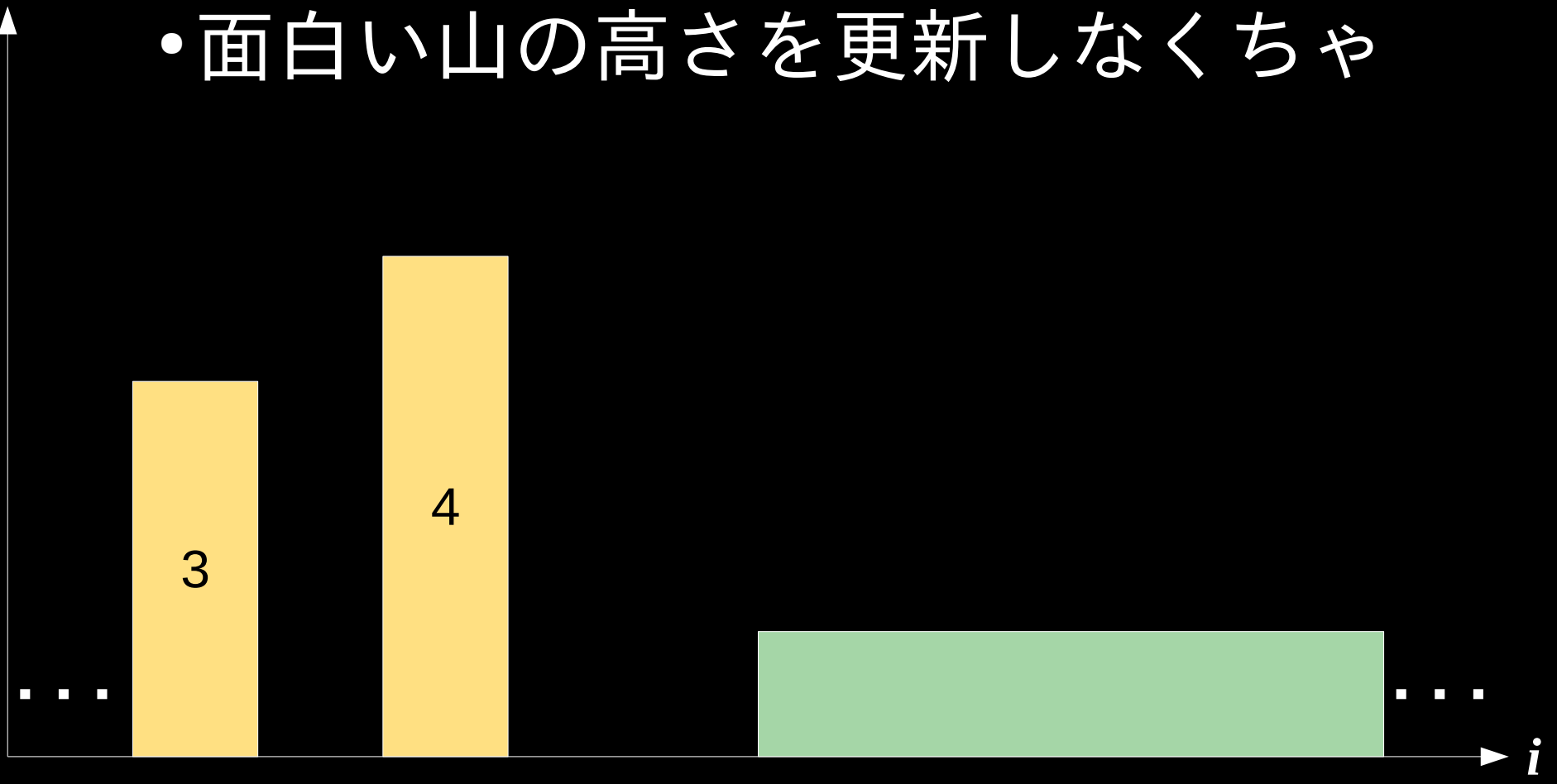


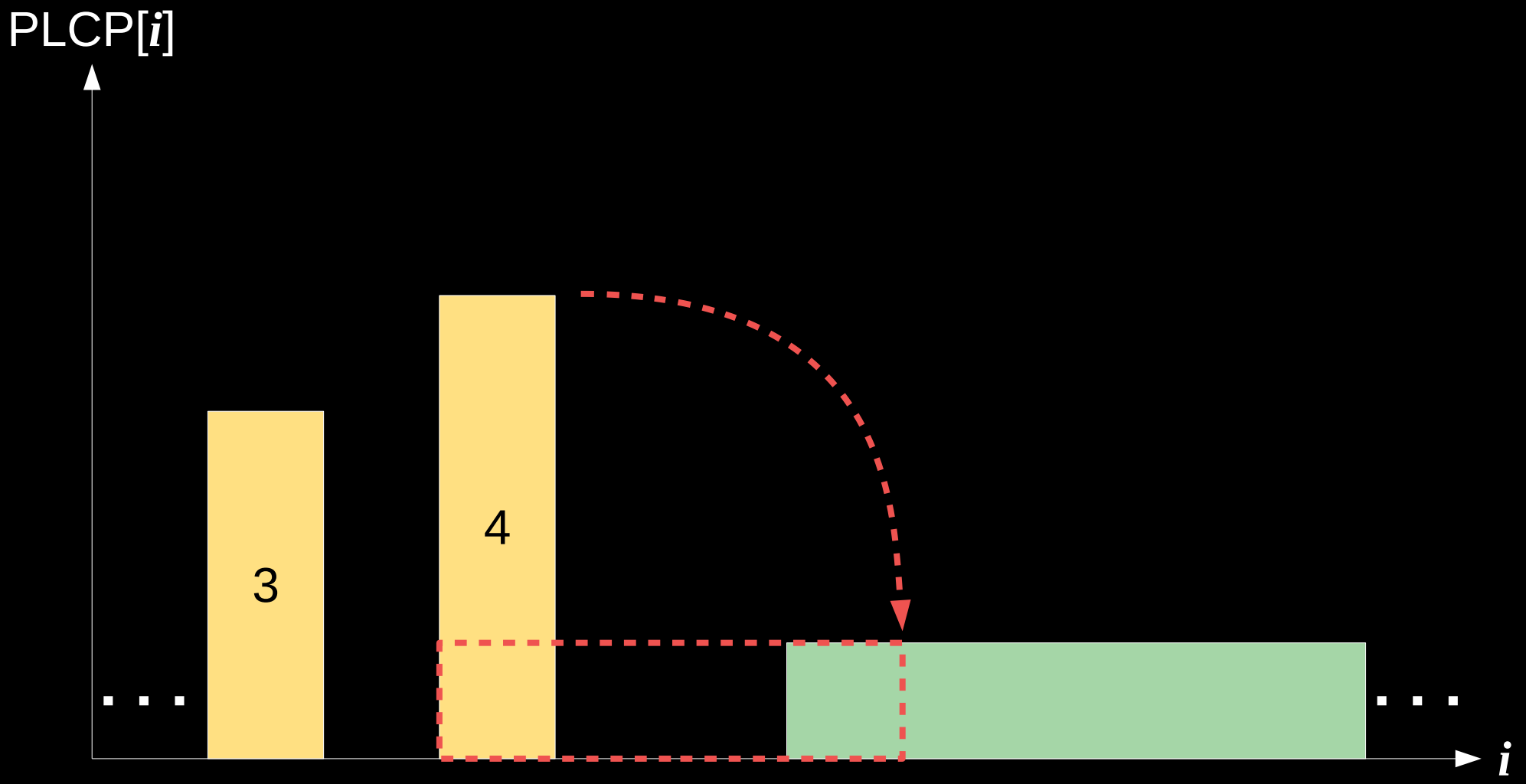
線形時間？

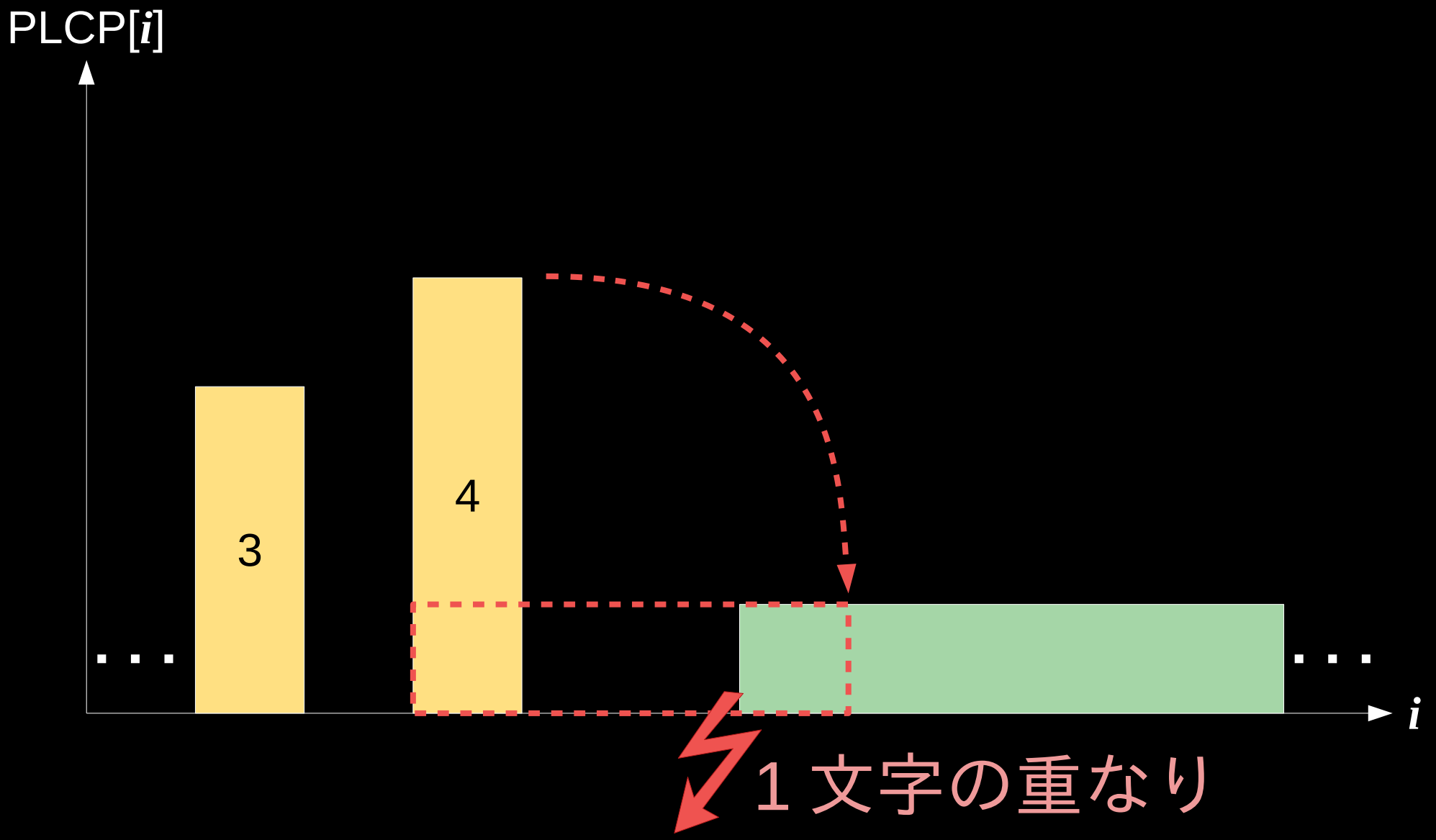
面白い山しか見ないで！

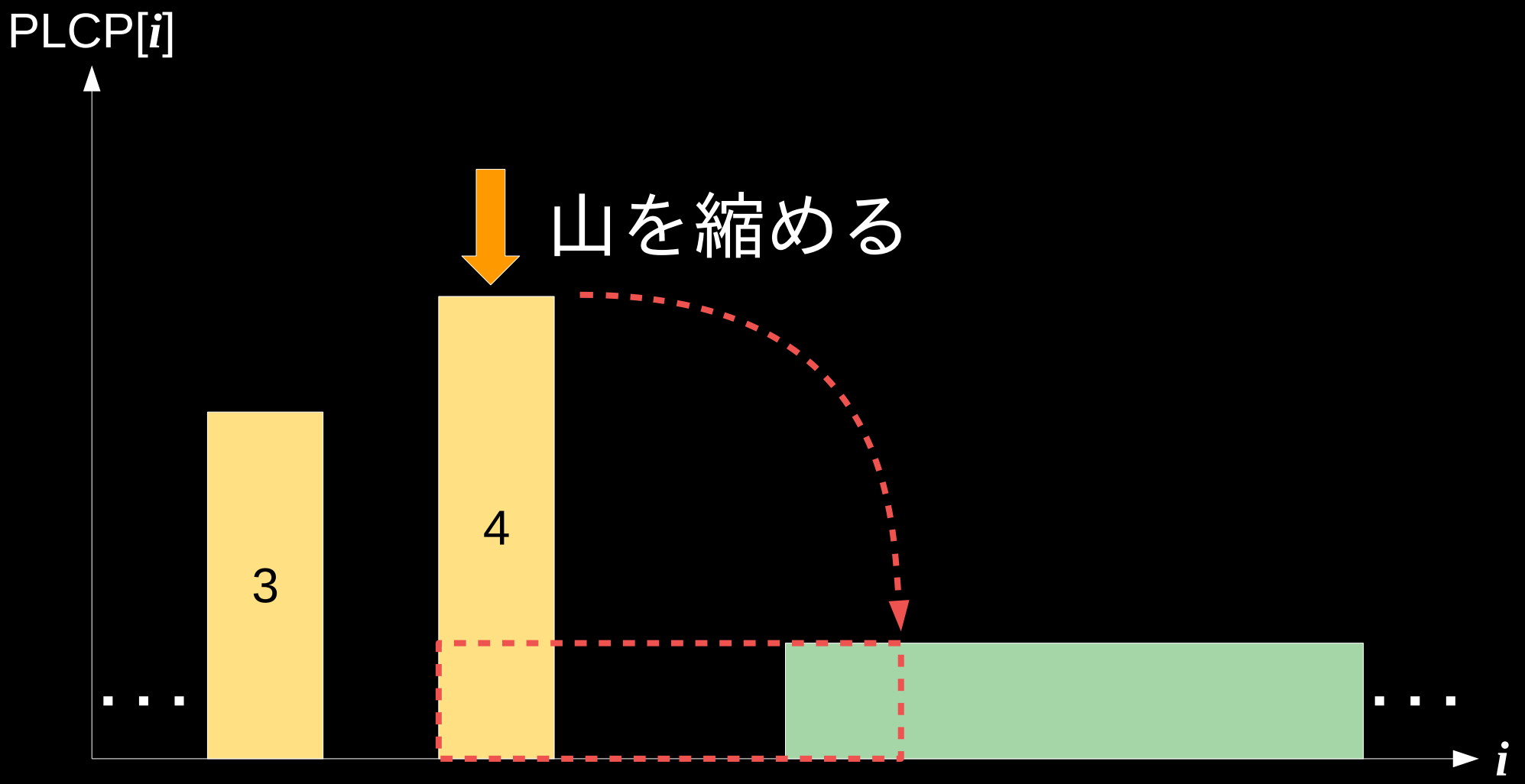
PLCP[*i*]

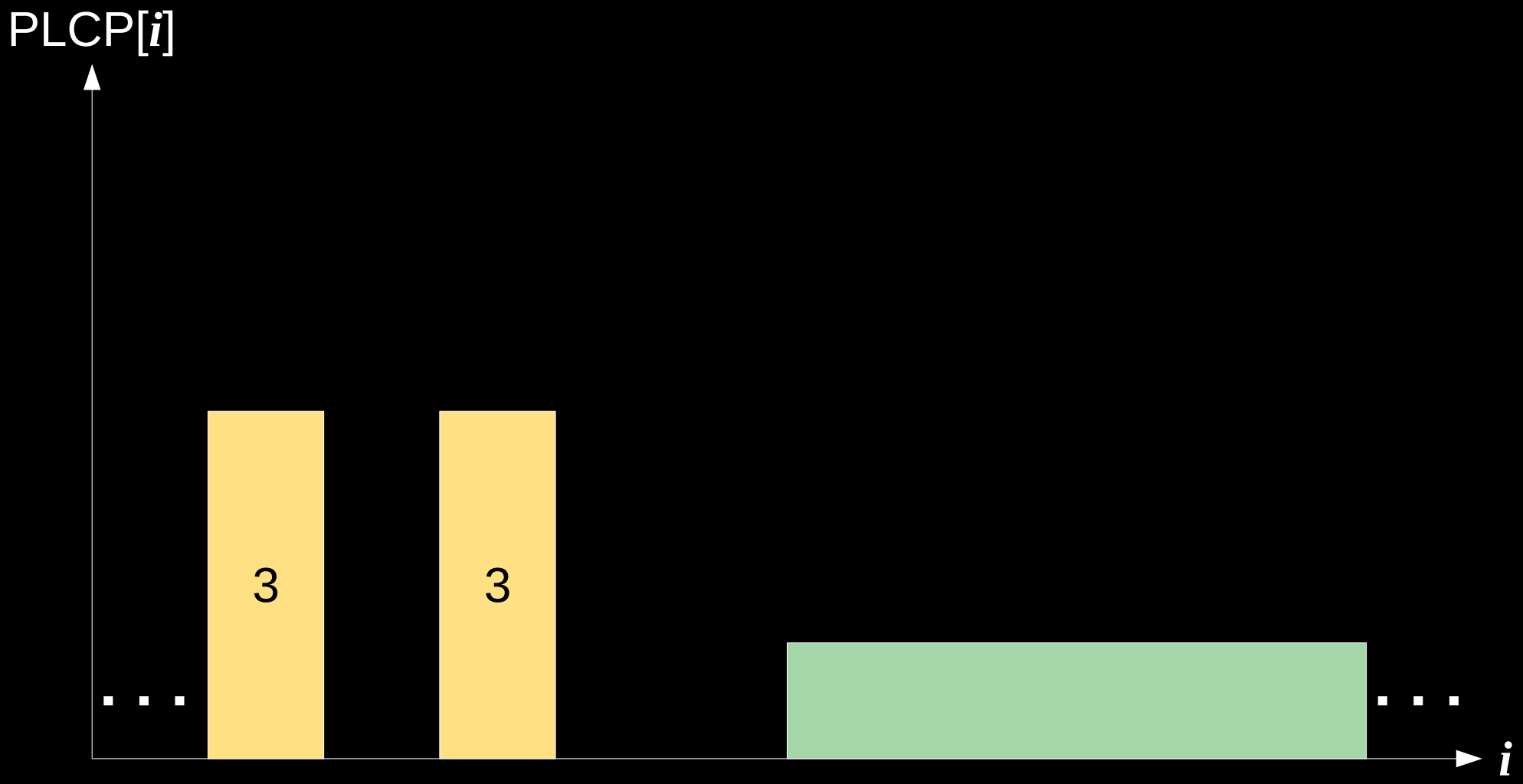
- 新しい最高な山は面白い山
- 面白い山の高さを更新しなくちゃ

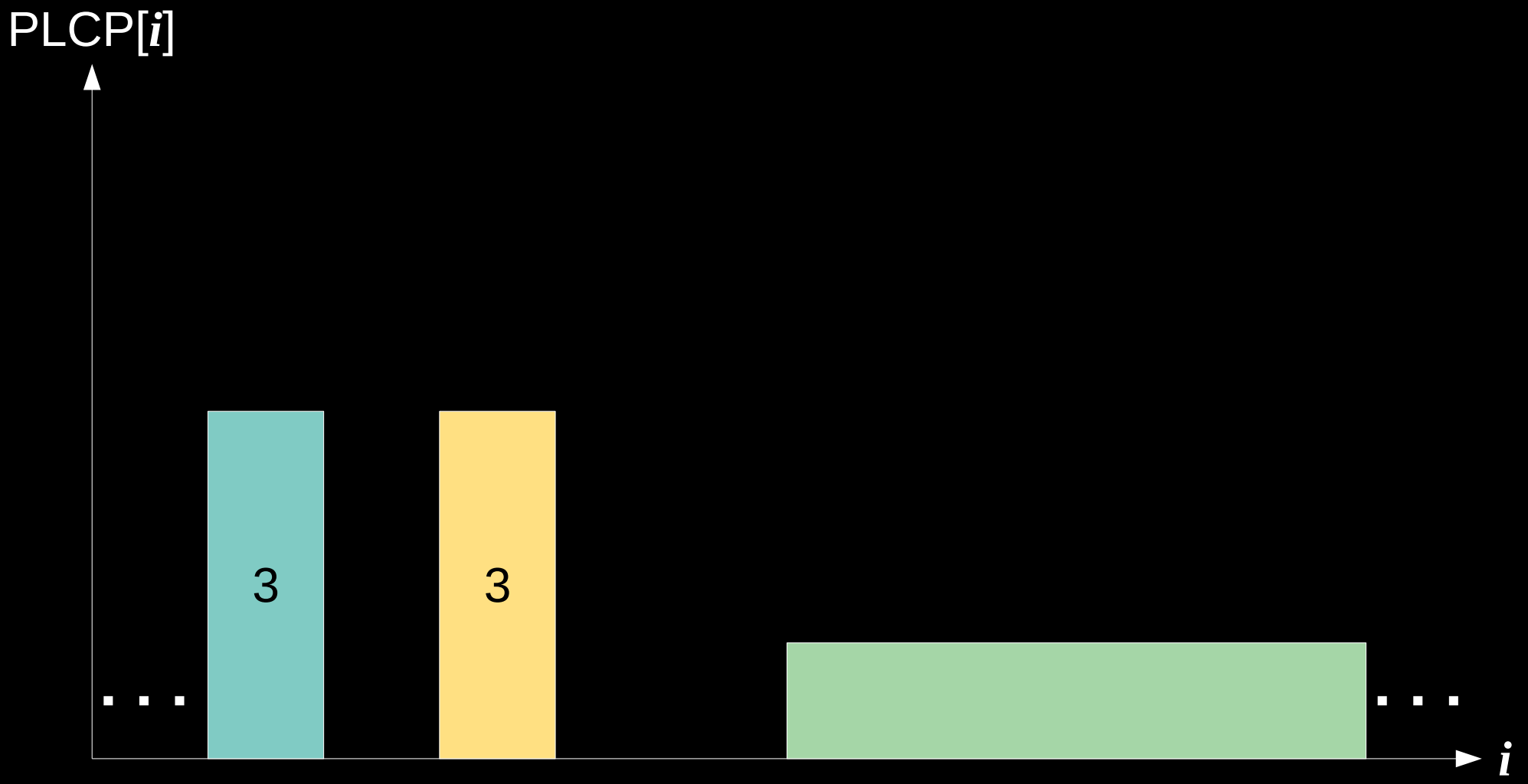



PLCP[i]
3
4
i

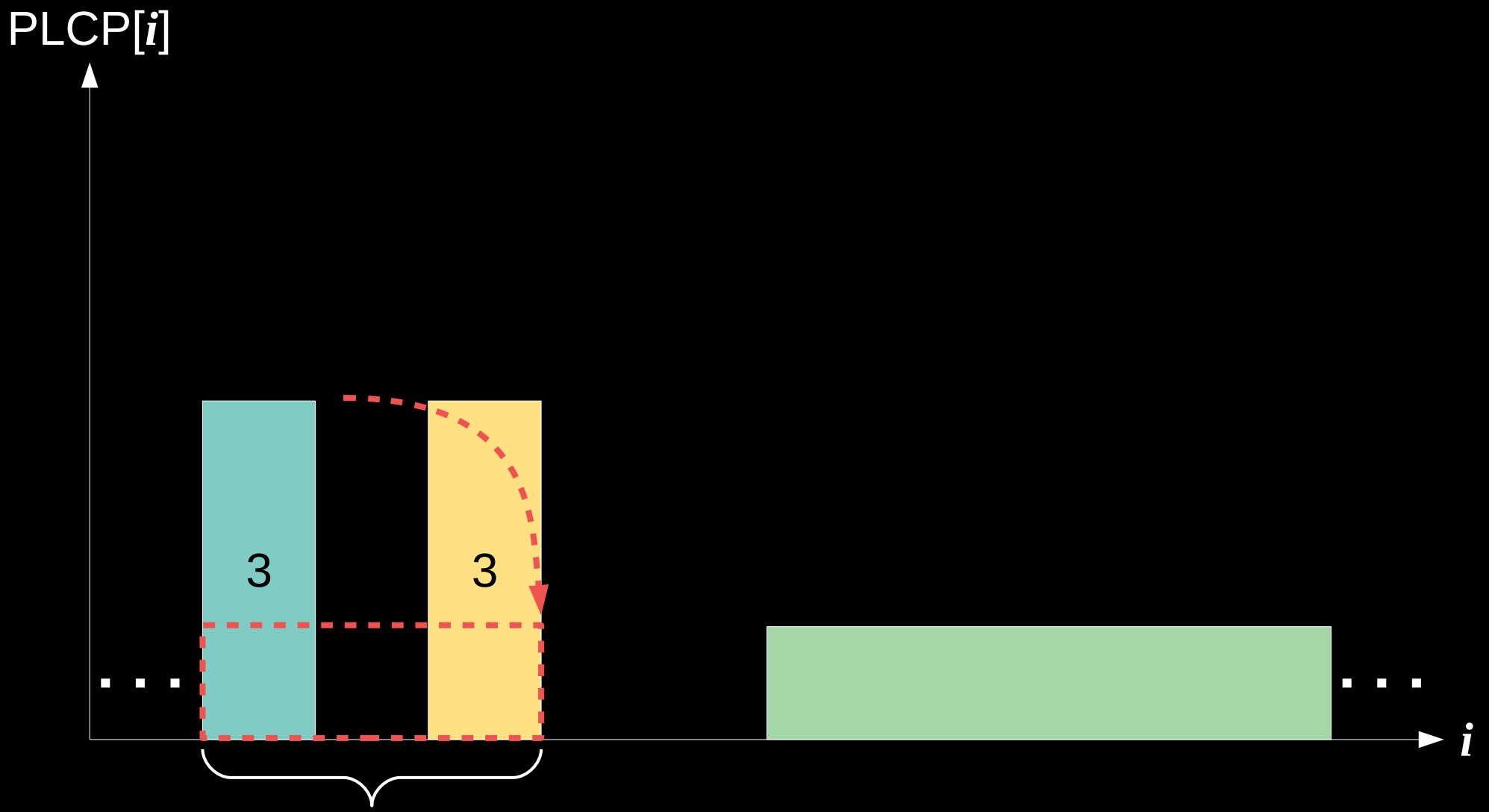

PLCP[i]
3
4
i
1 文字の重なり


PLCP[*i*]
山を縮める
3
4
i

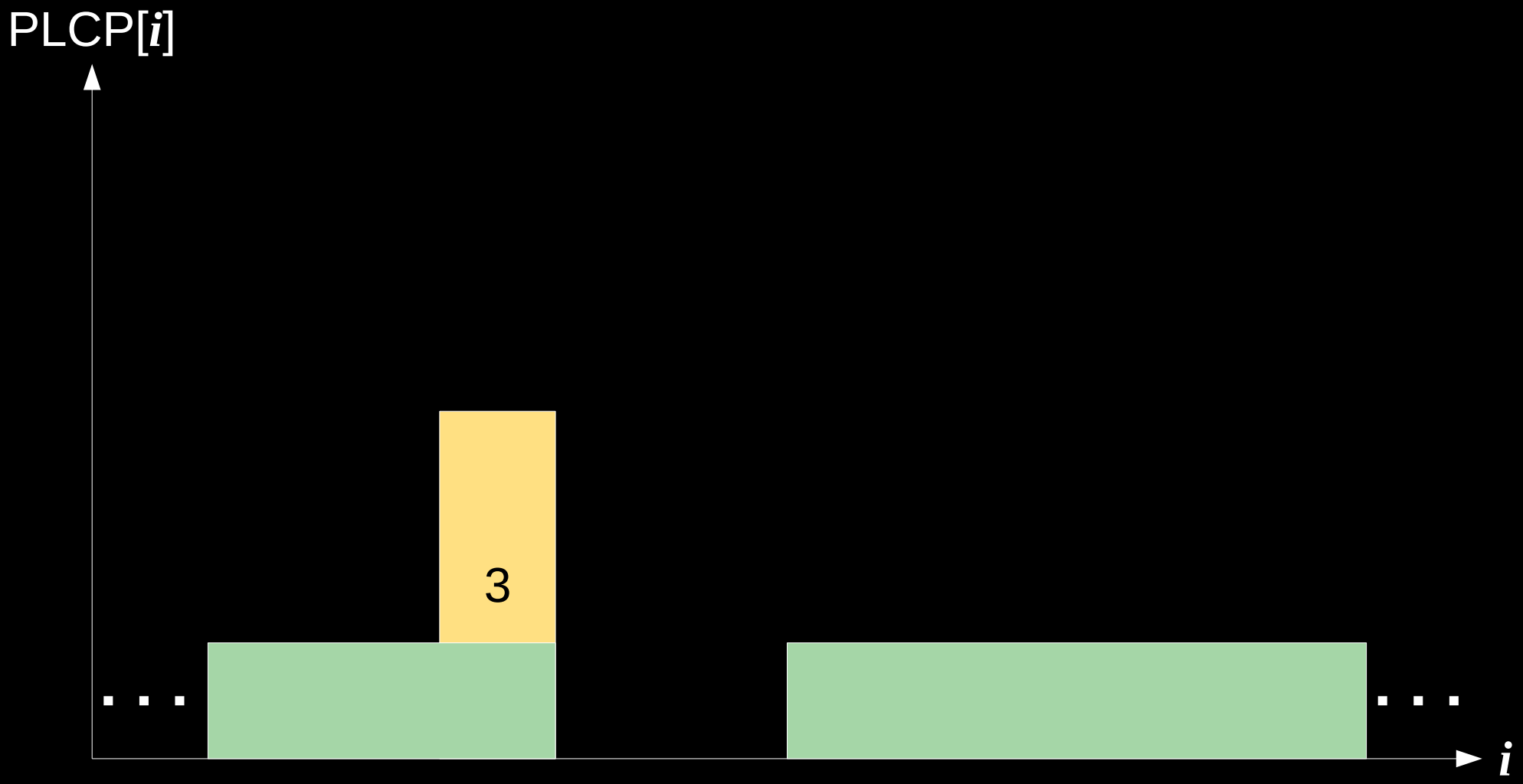

PLCP[i]
3
3
i

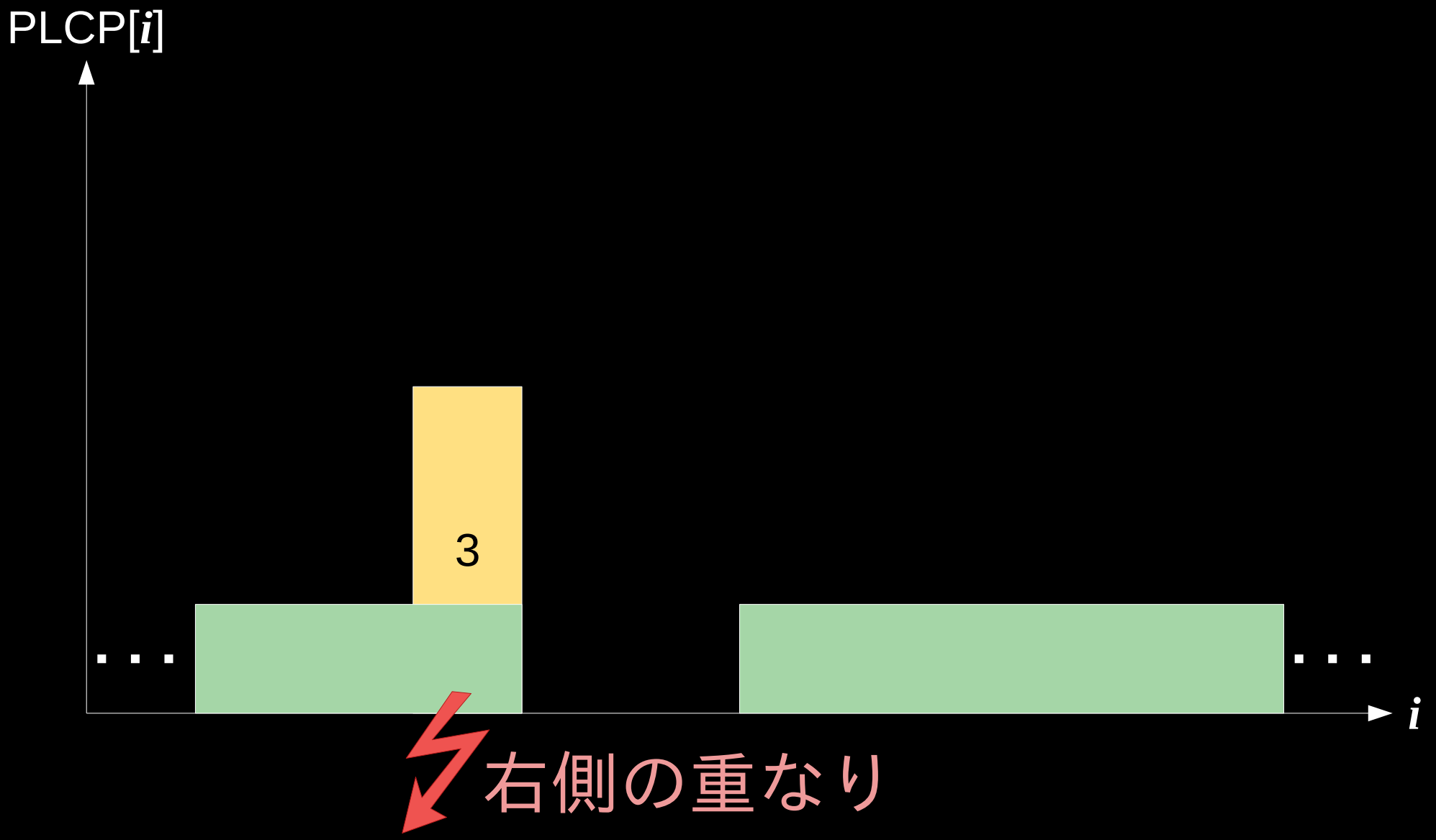


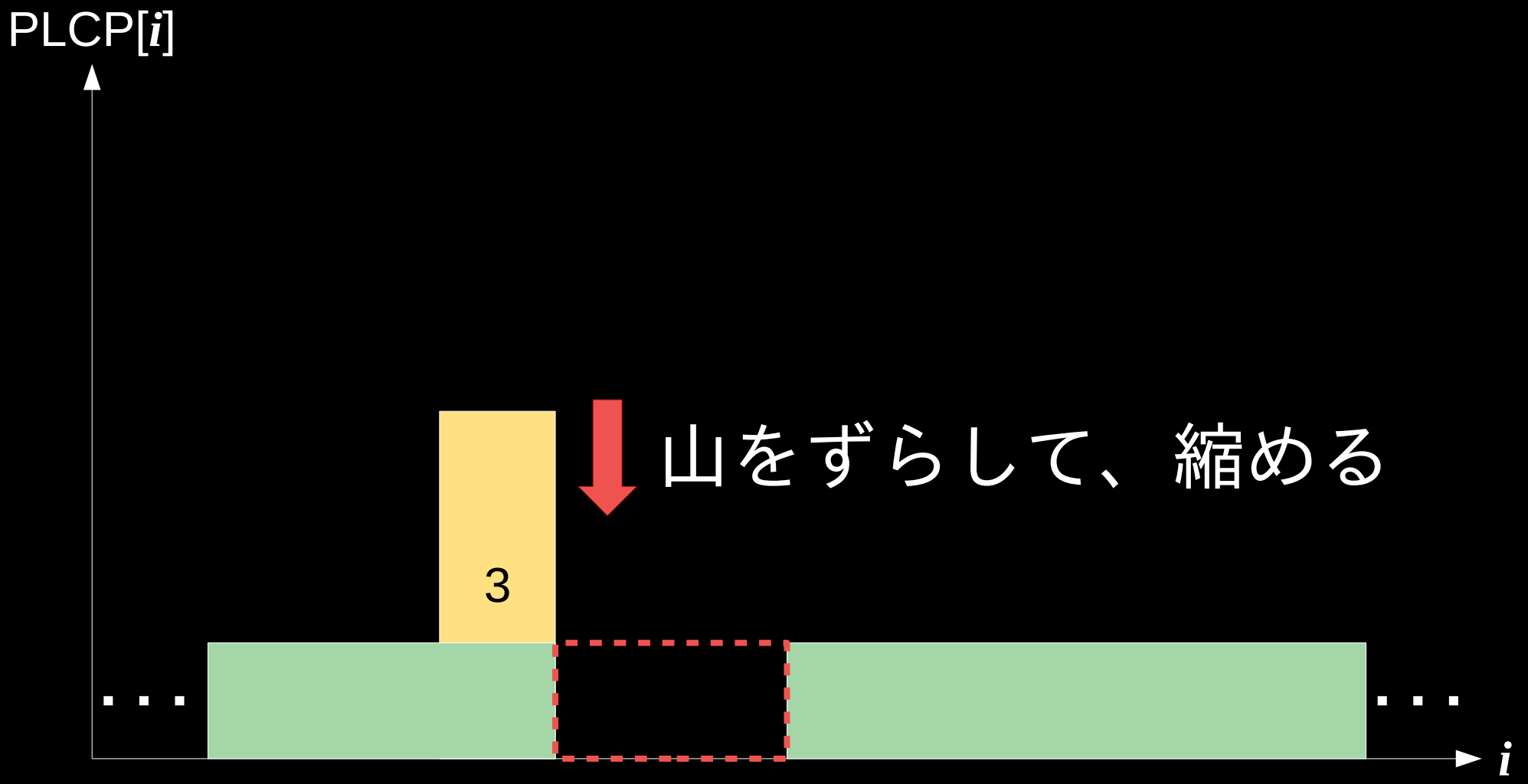
新しい最左の最高な山

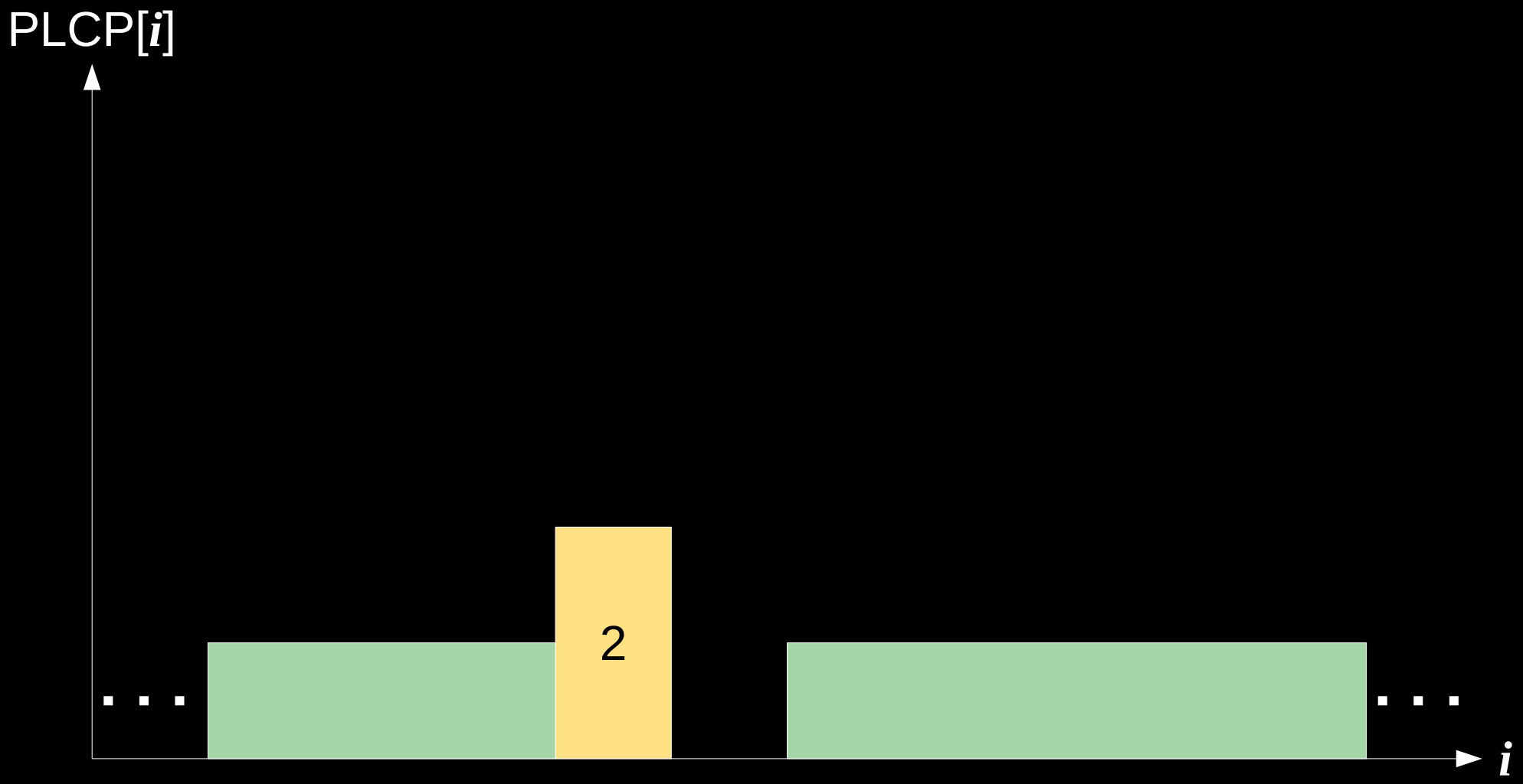


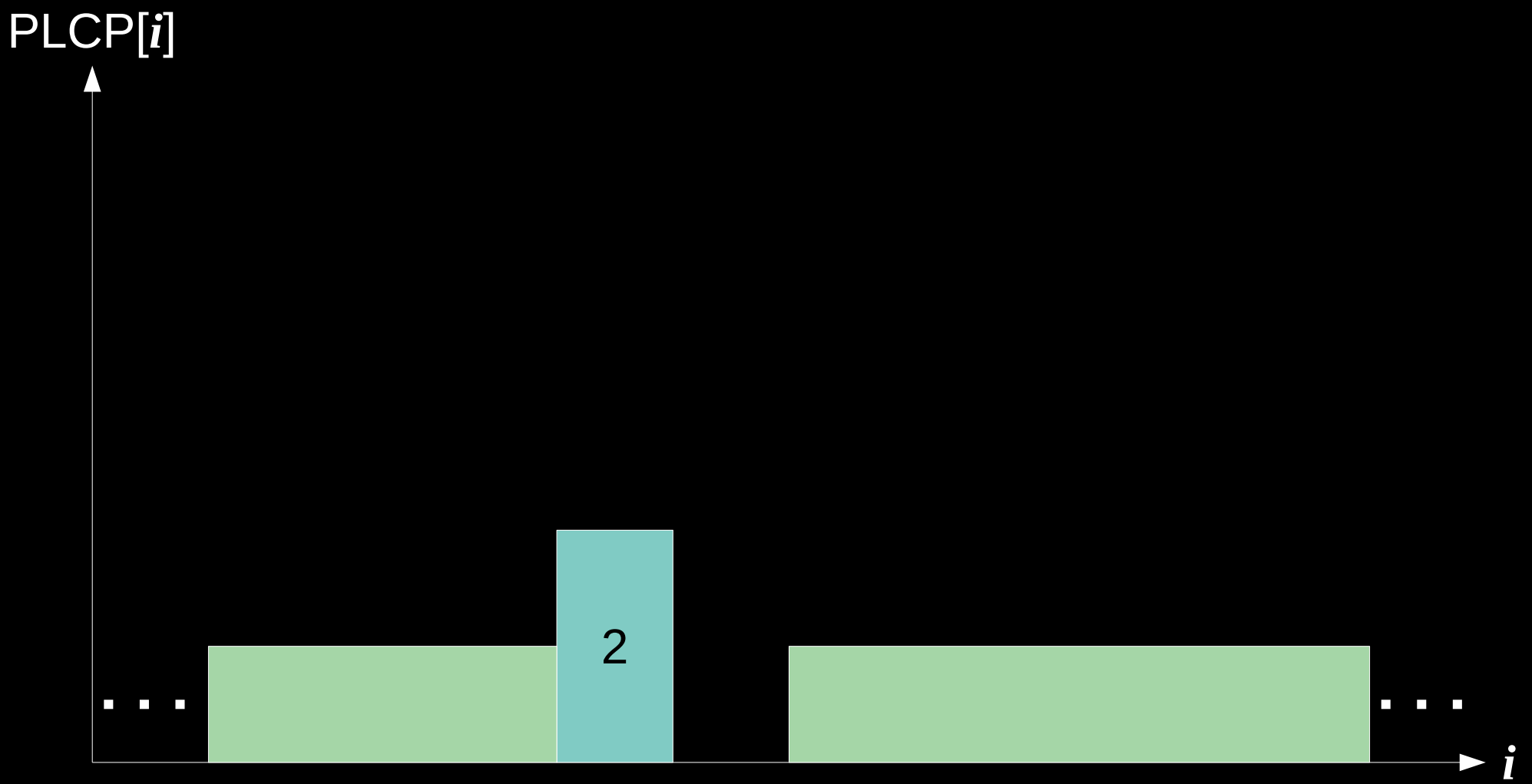
参照の長さは３


PLCP[i]
3
i

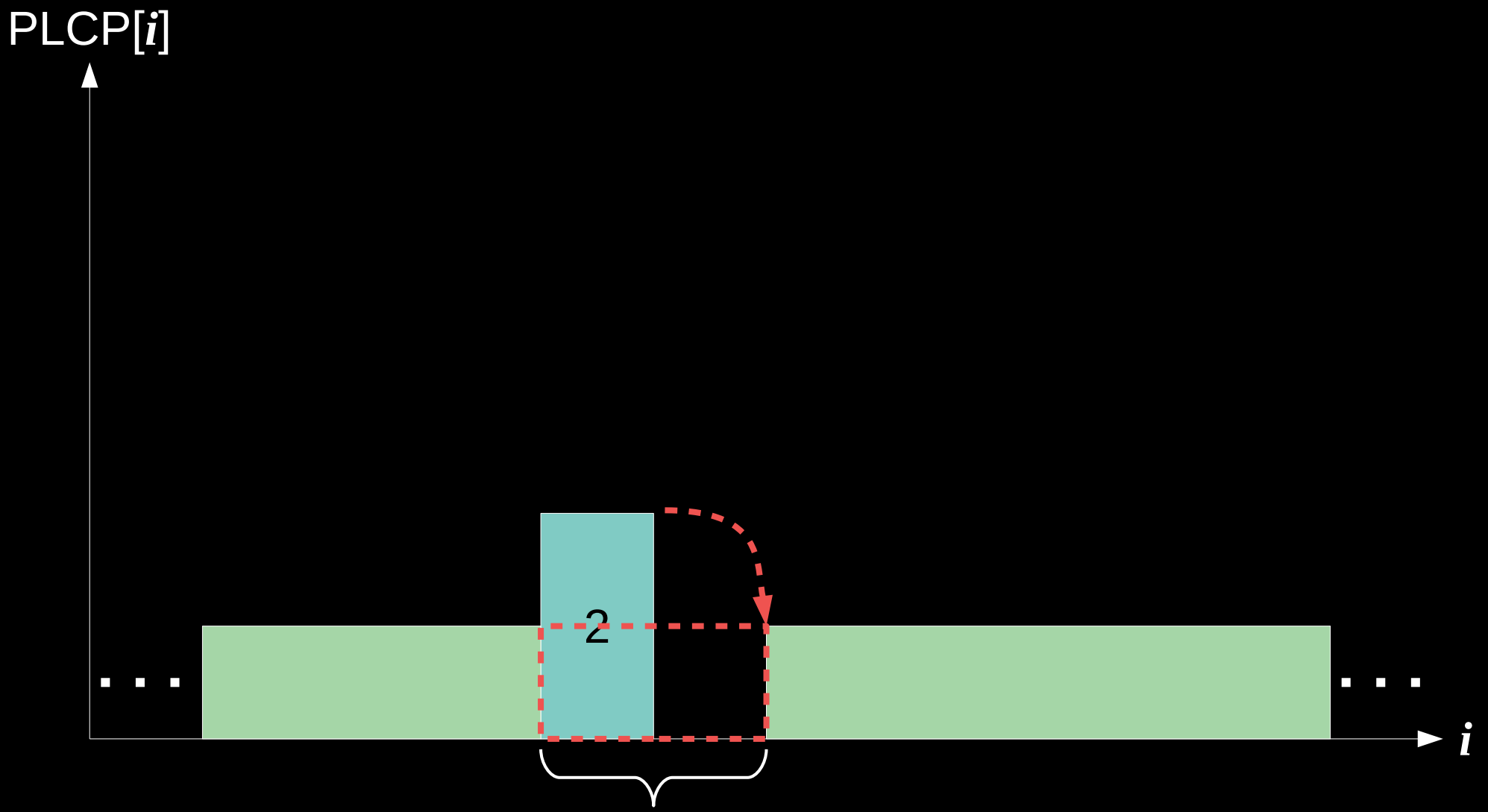

PLCP[i]
3
i
右側の重なり


PLCP[i]
山をずらして、縮める
3
i

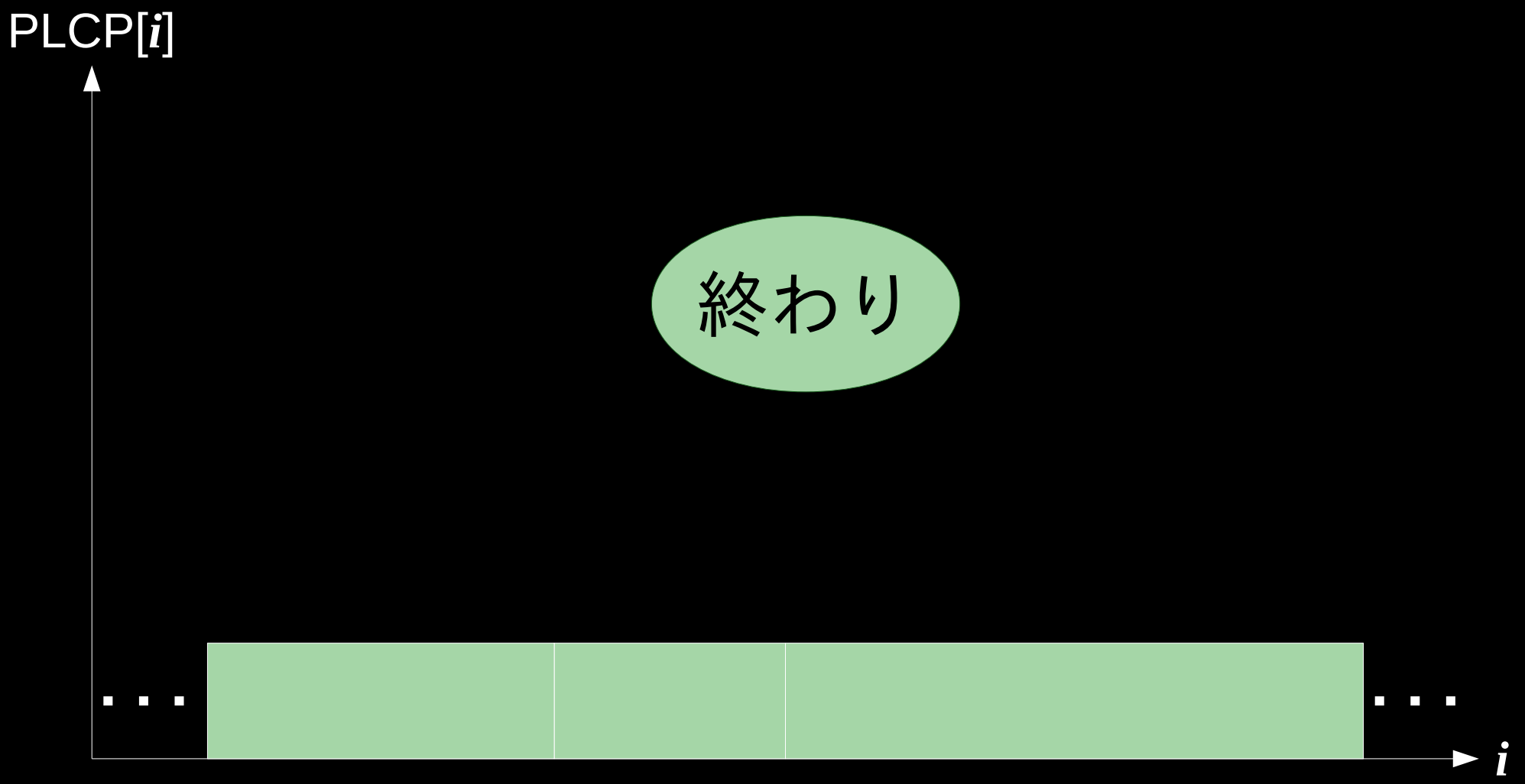

PLCP[i]
2
i



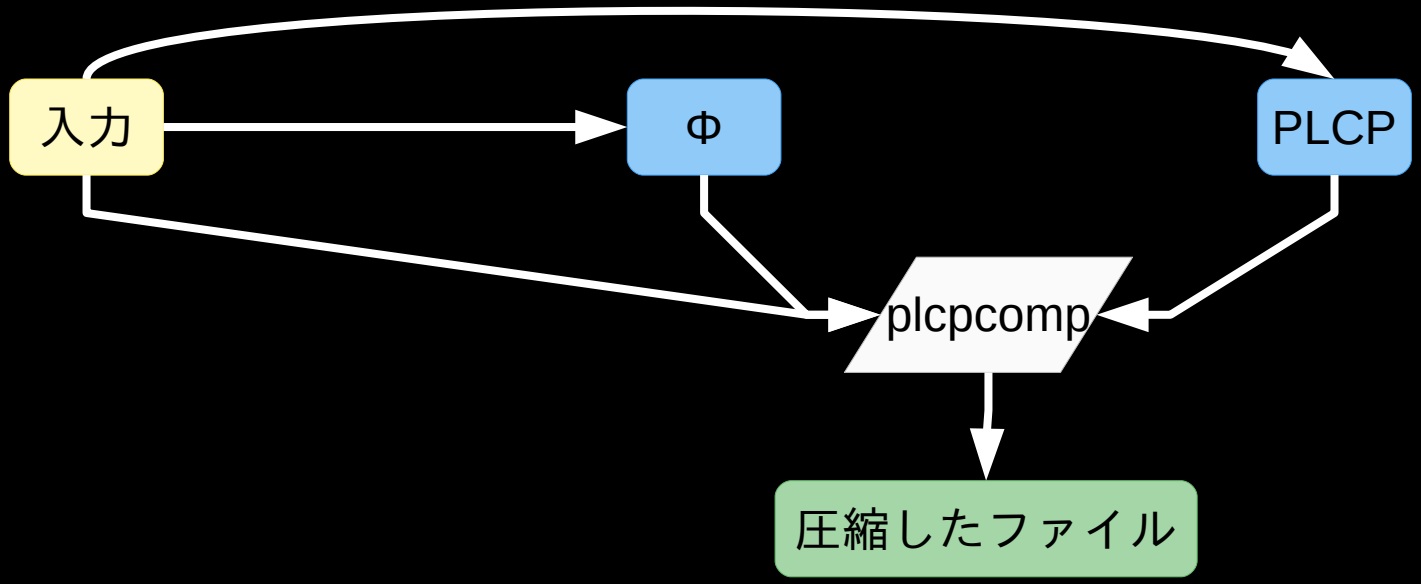
新しい最左の最高な山



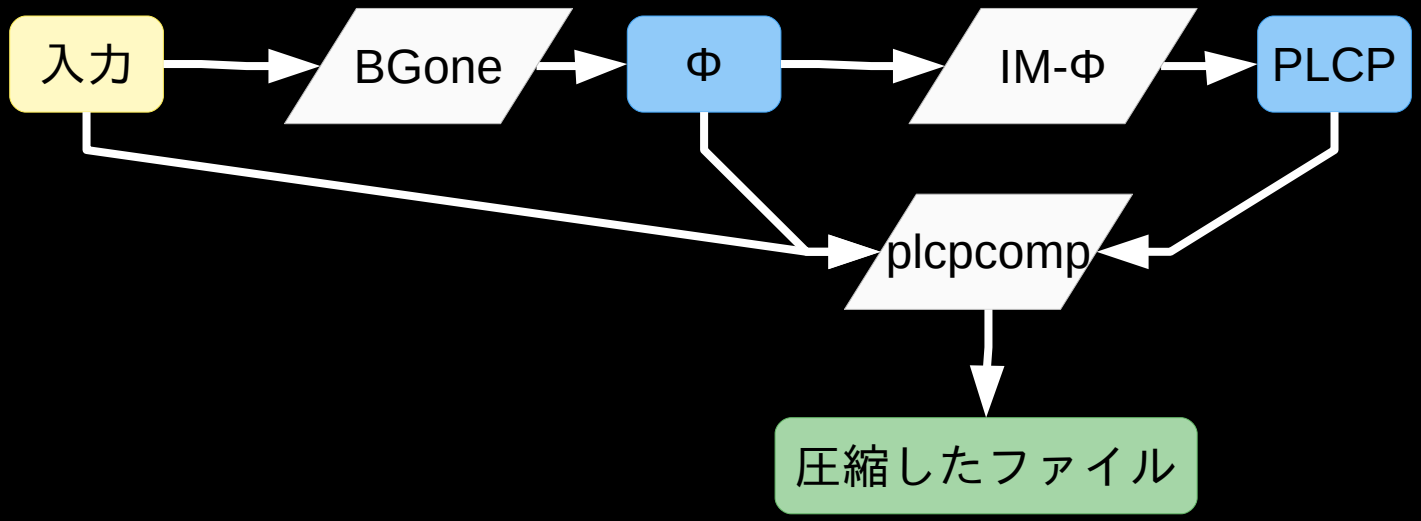
参照の長さは２


PLCP[i]
終わり
i

データ構造



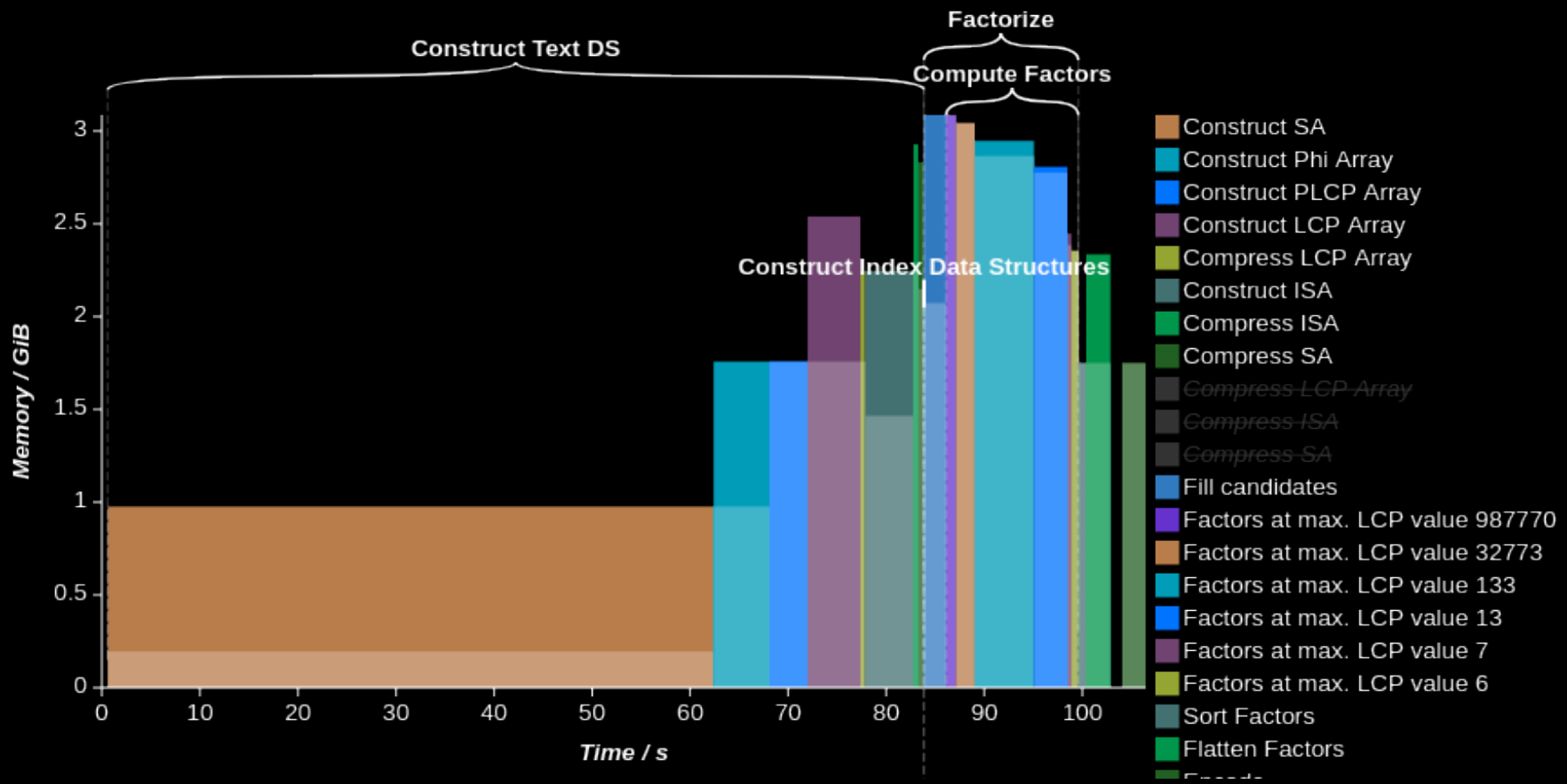
データ構造とアルゴリズム



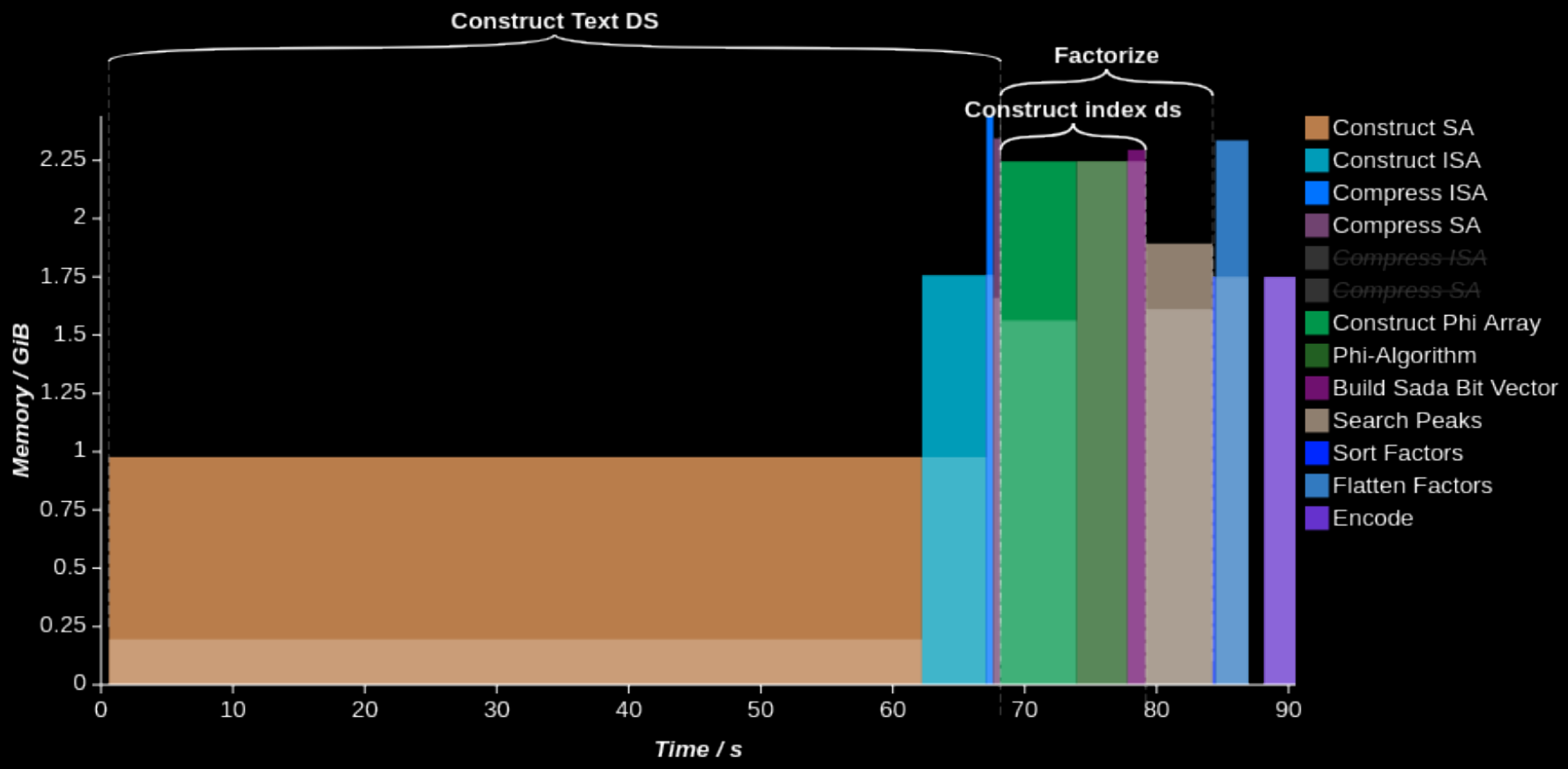
- BGone: 後藤 , 坂内 '14
- IM-Φ: Kärkkäinen+'09

線形時間

lcpcomp

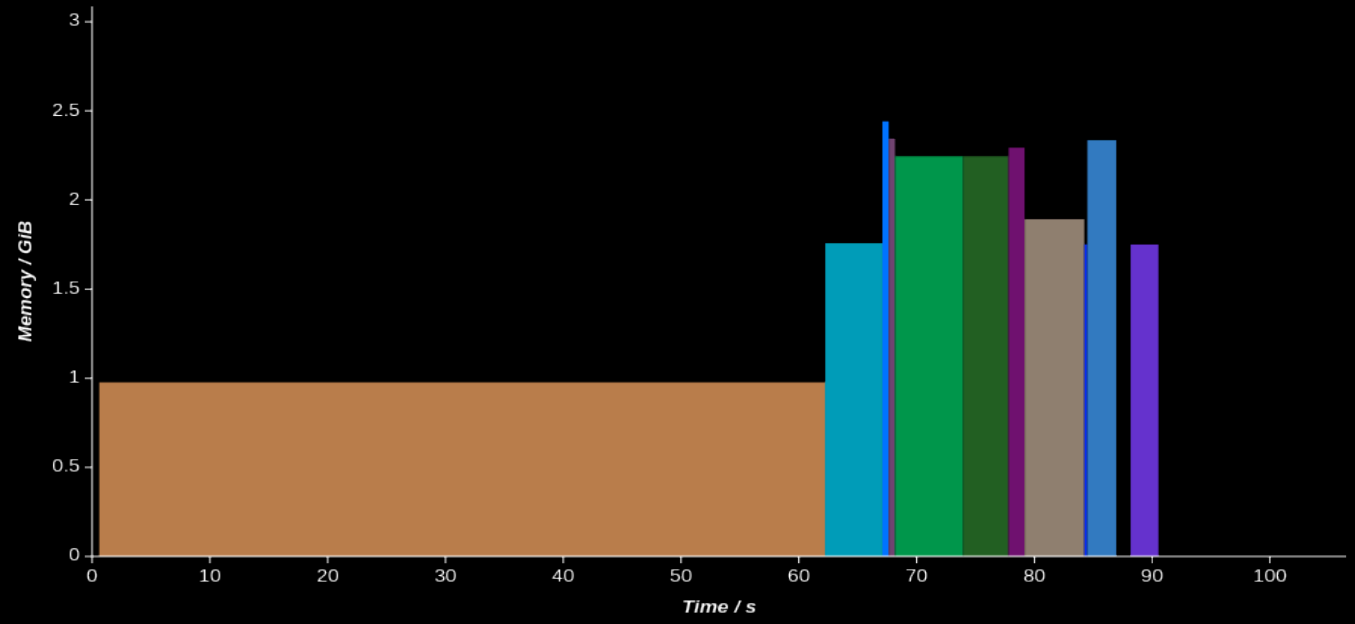

Factorize
Construct Text DS
Compute Factors
Construct Index Data Structures
Memory / GiB
Time / s
0
0.5
1
1.5
2
2.5
3
0
10
20
30
40
50
60
70
80
90
100
Construct SA
Construct Phi Array
Construct PLCP Array
Construct LCP Array
Compress LCP Array
Construct ISA
Compress ISA
Compress SA
Compress LCP Array
Compress ISA
Compress SA
Fill candidates
Factors at max. LCP value 987770
Factors at max. LCP value 32773
Factors at max. LCP value 133
Factors at max. LCP value 13
Factors at max. LCP value 7
Factors at max. LCP value 6
Sort Factors
Flatten Factors

plcpcomp

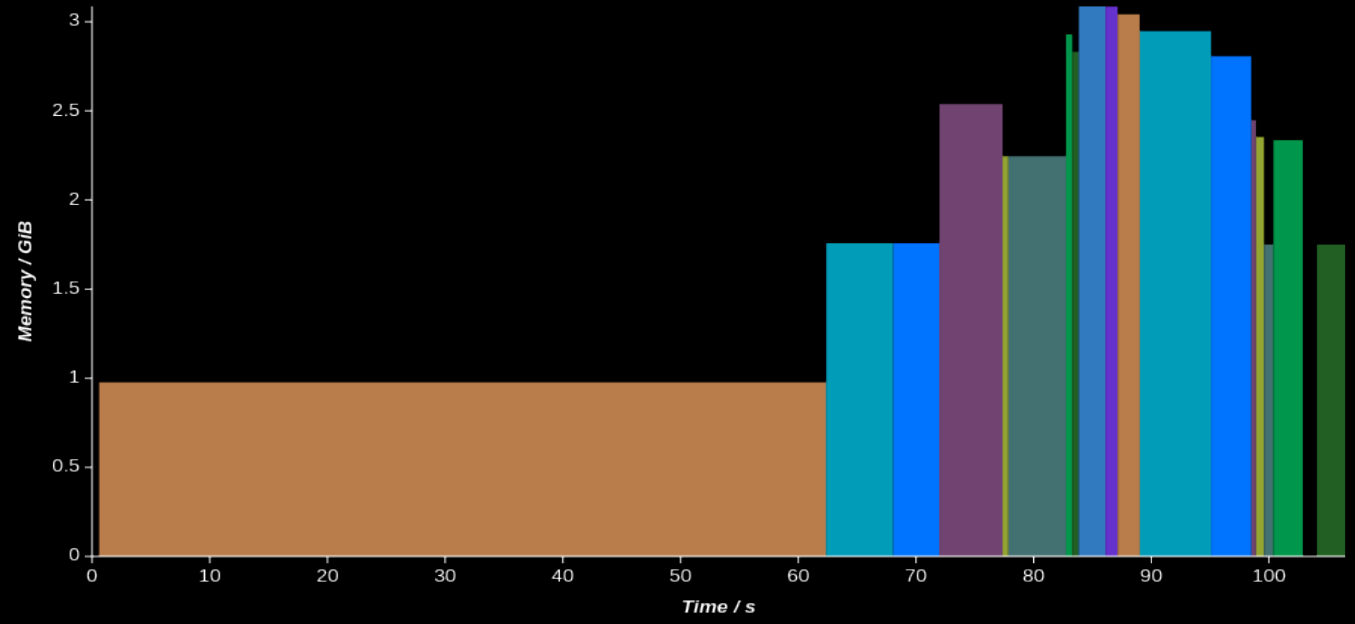

Construct Text DS
Factorize
Construct index ds
Memory / GiB
Time / s
0
0.25
0.5
0.75
1
1.25
1.5
1.75
2
2.25
0
10
20
30
40
50
60
70
80
90
Construct SA
Construct ISA
Compress ISA
Compress SA
Compress ISA
Compress SA
Construct Phi Array
Phi-Algorithm
Build Sada Bit Vector
Search Peaks
Sort Factors
Flatten Factors
Encode

見た目の比較

plcpcomp



lcpcomp



まとめ

plcpcomp

- bidirectional 圧縮
- 線形時間
- lcpcomp より実際に早い
  - テキスト、 PLCP 、 Φ を線形に走査できる

まだ説明していないこと：

- 外部メモリーアルゴリズム (EM) もある
- EM で LZ77 のような方法の中では一番早い