

# Converting SLP to LZ78 in almost Linear Time

---

Hideo Bannai<sup>1</sup>, Paweł Gawrychowski<sup>2</sup>,  
Shunsuke Inenaga<sup>1</sup>, Masayuki Takeda<sup>1</sup>

1. Kyushu University

2. Max-Planck-Institut für Informatik

# “Recompress” SLP to LZ78

- ✓ The problem we consider is kind of recompression, i.e., convert SLP for string  $w$  to LZ78 encoding of  $w$ .
- ✓ Example of our motivation:
  - NCD of two strings  $s$  and  $t$  w.r.t. LZ78 compressor: Determined by  $|\text{LZ78}(s)|$ ,  $|\text{LZ78}(t)|$ , and  $|\text{LZ78}(st)|$ .
  - See [Bannai et al., SPIRE 2012] for more.
- ✓ Why LZ78?
  - Widely used (GIF, PDF, TIFF, etc.).
  - Nice CSP algorithms (cf., [Gawrychowski 2011, 2012]).

# Straight Line Program (SLP)

An SLP is a sequence of productions

$$X_1 = expr_1, X_2 = expr_2, \dots, X_n = expr_n$$

- $expr_i = a \quad (a \in \Sigma)$
- $expr_i = X_l X_r \quad (l, r < i)$

- ✓ The size of the SLP is the number  $n$  of productions.
- ✓ An SLP is essentially a CFG deriving a single string.
- ✓ SLPs model outputs of grammar-based compression algorithms (e.g., Re-pair, Sequitur, LCAcomp, etc).

# Example of SLP

SLP  $G$

derivation tree for  $G$

$$X_1 = a$$

$$X_2 = b$$

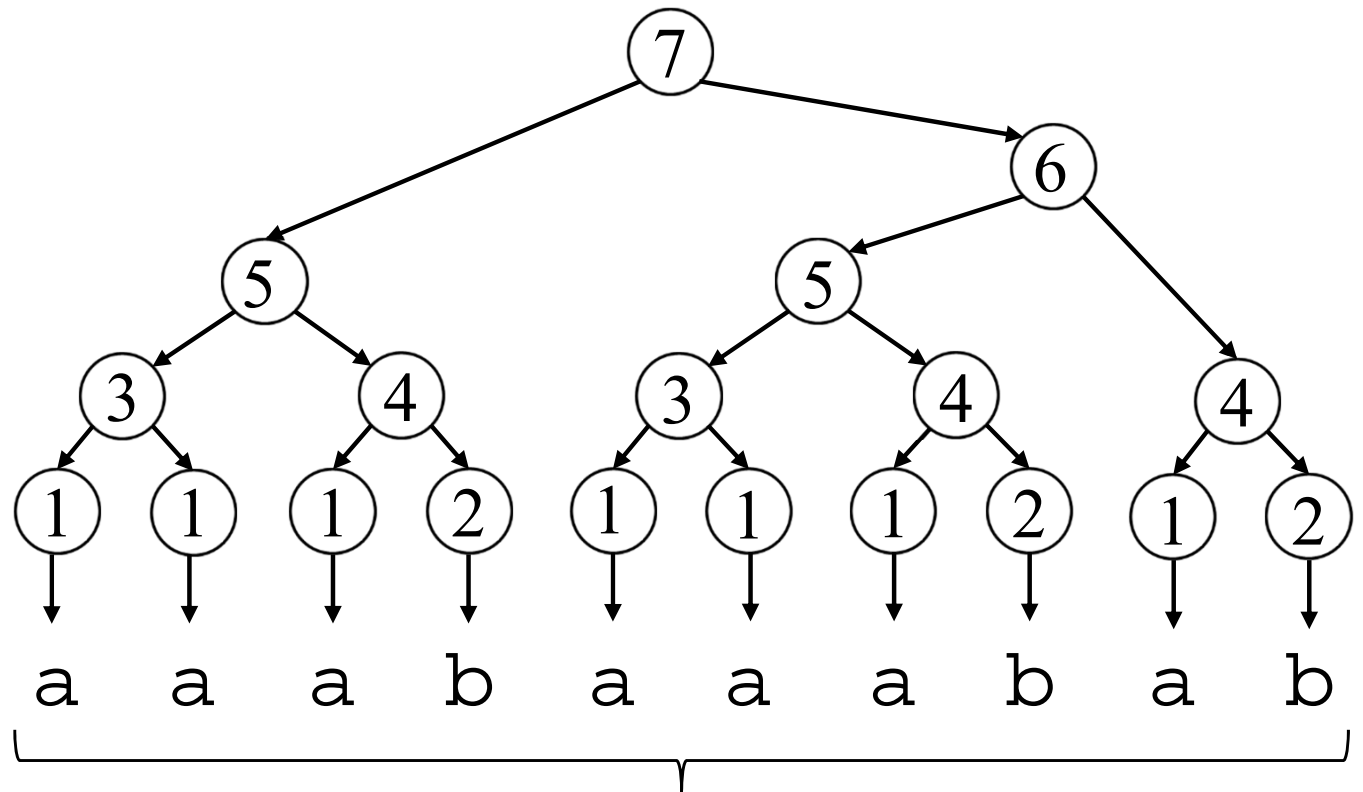
$$X_3 = X_1 X_1$$

$$X_4 = X_1 X_2$$

$$X_5 = X_3 X_4$$

$$X_6 = X_5 X_4$$

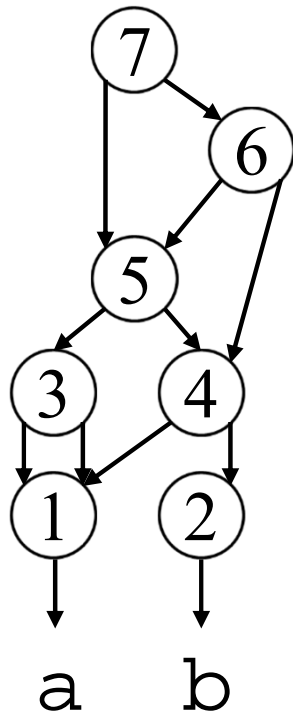
$$X_7 = X_5 X_6$$



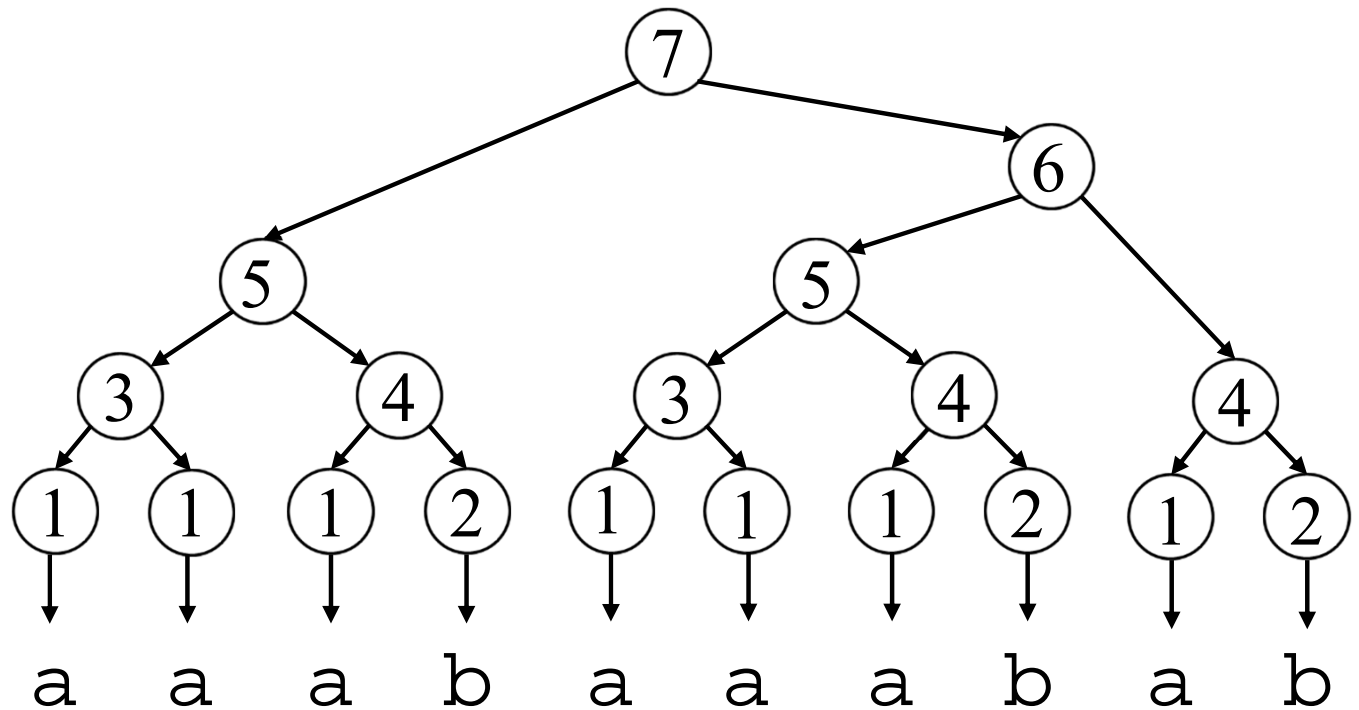
string  $w$  represented by  $G$

# DAG Representation of SLP

DAG for SLP  $S$



derivation tree for  $G$



- ✓ DAG is compressed representation of derivation tree.
- ✓ SLP is compressed representation of string.

# LZ78 Factorization

The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a a a b a a a b a b$

LZ78 trie of  $w$

0

# LZ78 Factorization

The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a \mid a \ a \ b \ a \ a \ a \ b \ a \ b$   
 $f_1 \mid$

LZ78 trie of  $w$

0

# LZ78 Factorization

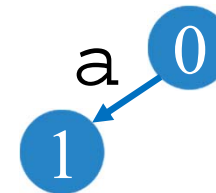
The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a \mid a \ a \ b \ a \ a \ a \ b \ a \ b$   
 $f_1 \mid$

LZ78 trie of  $w$





# LZ78 Factorization

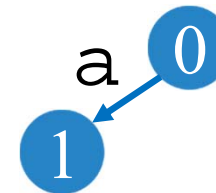
The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a \mid a \ a \mid b \ a \ a \ a \ b \ a \ b$   
 $f_1 \mid f_2 \mid$

LZ78 trie of  $w$



# LZ78 Factorization

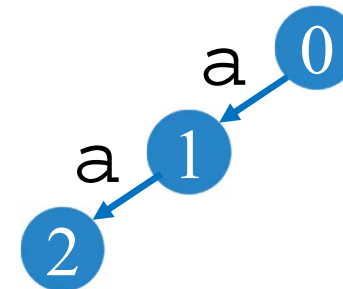
The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a \mid a \ a \mid b \ a \ a \ a \ b \ a \ b$   
 $f_1 \mid f_2 \mid$

LZ78 trie of  $w$



# LZ78 Factorization

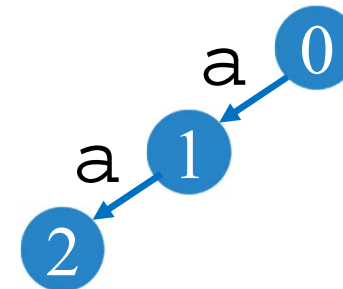
The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a \mid a \ a \mid b \mid a \ a \ a \ b \ a \ b$   
 $f_1 \mid f_2 \mid f_3 \mid$

LZ78 trie of  $w$



# LZ78 Factorization

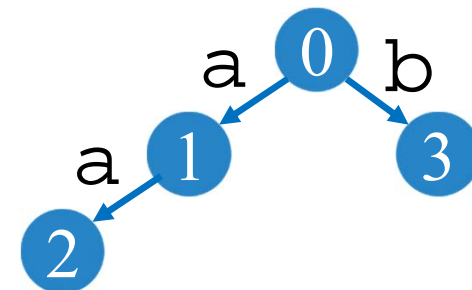
The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a \mid a \ a \mid b \mid a \ a \ a \ b \ a \ b$   
 $f_1 \mid f_2 \mid f_3 \mid$

LZ78 trie of  $w$



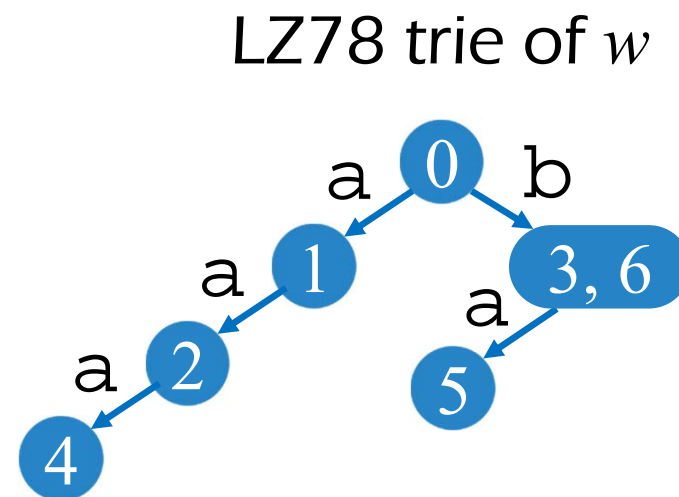
# LZ78 Factorization

The LZ78 factorization of string  $w$  is a factorization

$$w = f_1 f_2 \dots f_m$$

where  $f_j$  is the longest prefix of  $f_j \dots f_m$  such that  $f_j = f_k c$  for some  $0 \leq k < j$  (let  $f_0 = \varepsilon$ ) and  $c \in \Sigma$ .

$w = a \mid a \ a \mid b \mid a \ a \ a \mid b \ a \mid b$   
 $f_1 \mid f_2 \mid f_3 \mid f_4 \mid f_5 \mid f_6$



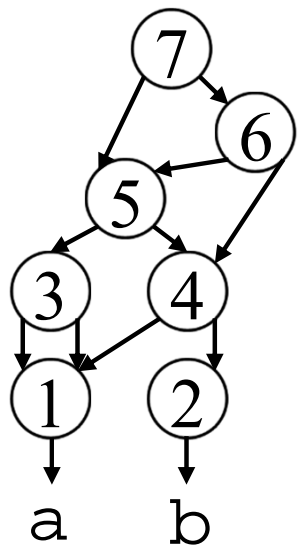
# Converting SLP to LZ78

## Problem

Given an SLP  $G$  of size  $n$  representing string  $w$ , compute the LZ78 trie of the same string  $w$ .

# Converting SLP to LZ78 [Cont.]

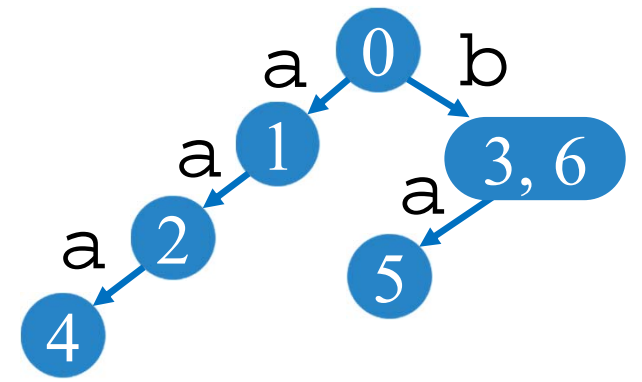
SLP  $G$  of size  $n$



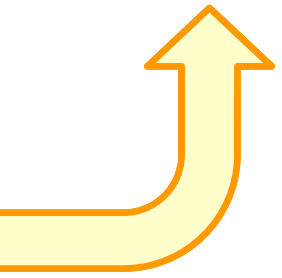
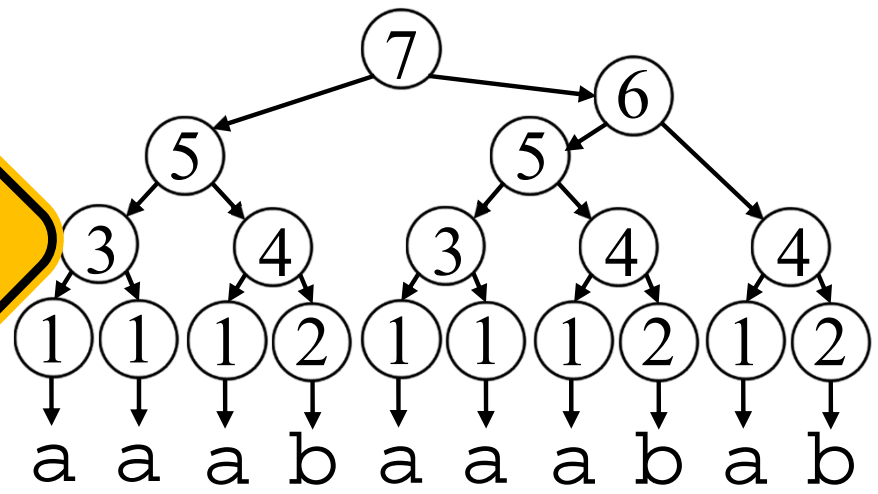
Go this way!



LZ78 trie of size  $m$



derivation tree of  $G$



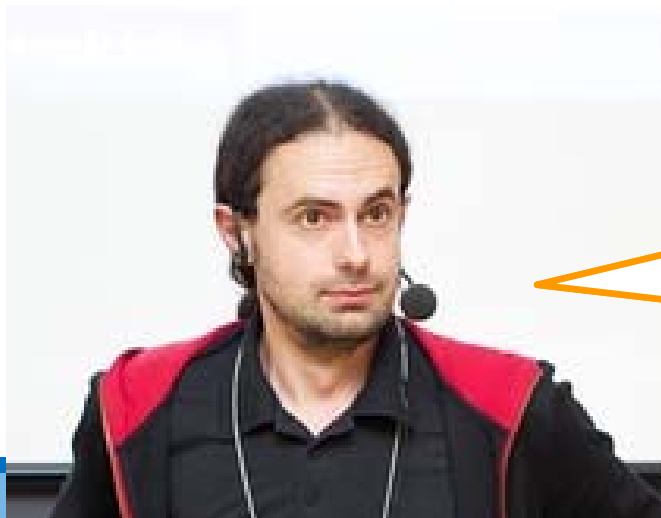
$O(2^n)$  time!!

# Previous Result & our Hero Joined

Theorem 1 [Bannai, Inenaga, Takeda, SPIRE 2012]

Given an SLP of size  $n$  describing string  $w$ , we can compute the LZ78 trie of  $w$  of size  $m$  in  $O(nL + m \log N)$  time.

- $N$  is the length of uncompressed string  $w$
- $L$  is the length of longest LZ78 factor of  $w$



Hmmm, this can be improved!



# New Result

## Theorem 2 (new!)

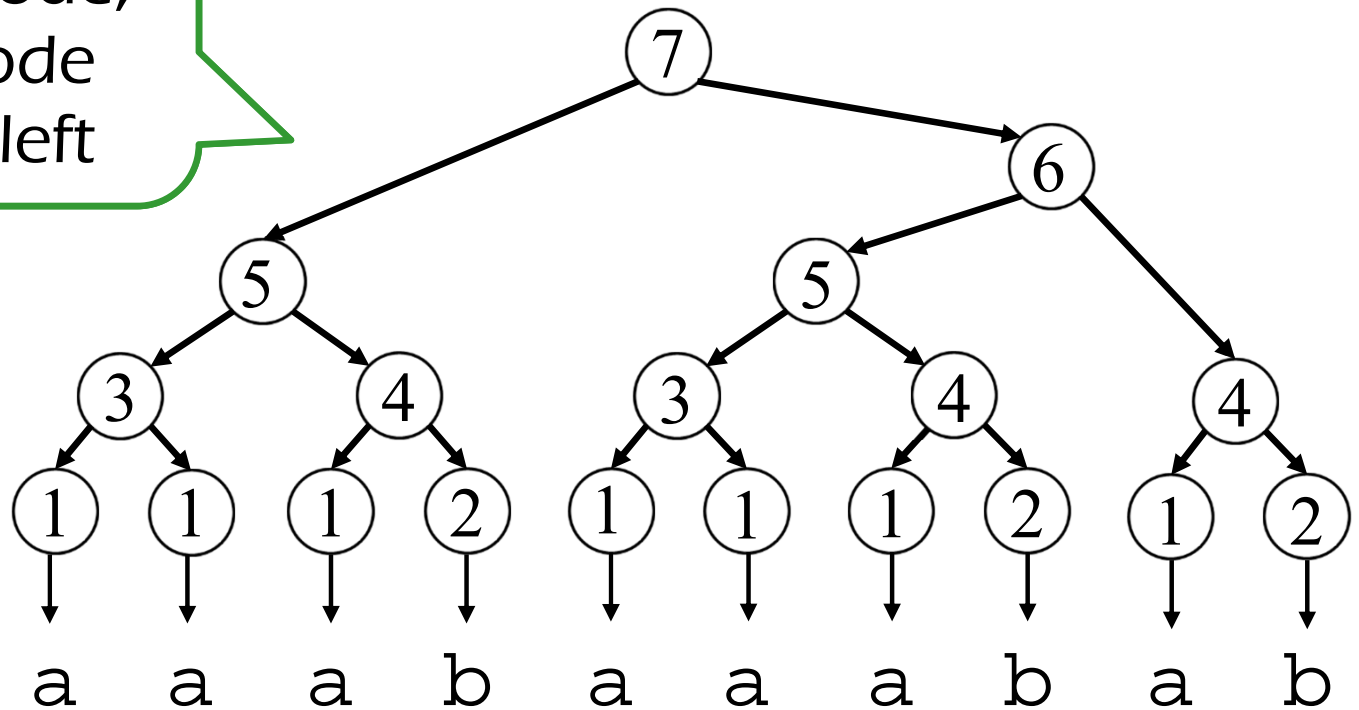
Given an SLP of size  $n$  describing string  $w$ , we can compute the LZ78 trie of  $w$  of size  $m$  in  $O(n + m \log m)$  time.

- ✓ Improved from  $O(nL + m \log N)$  to  $O(n + m \log m)$ 
  - $L = O(\sqrt{N})$  is the length of the longest LZ78 factor

# G-parsing

Prune the subtree rooted at internal node, if the label of the node also appears to the left

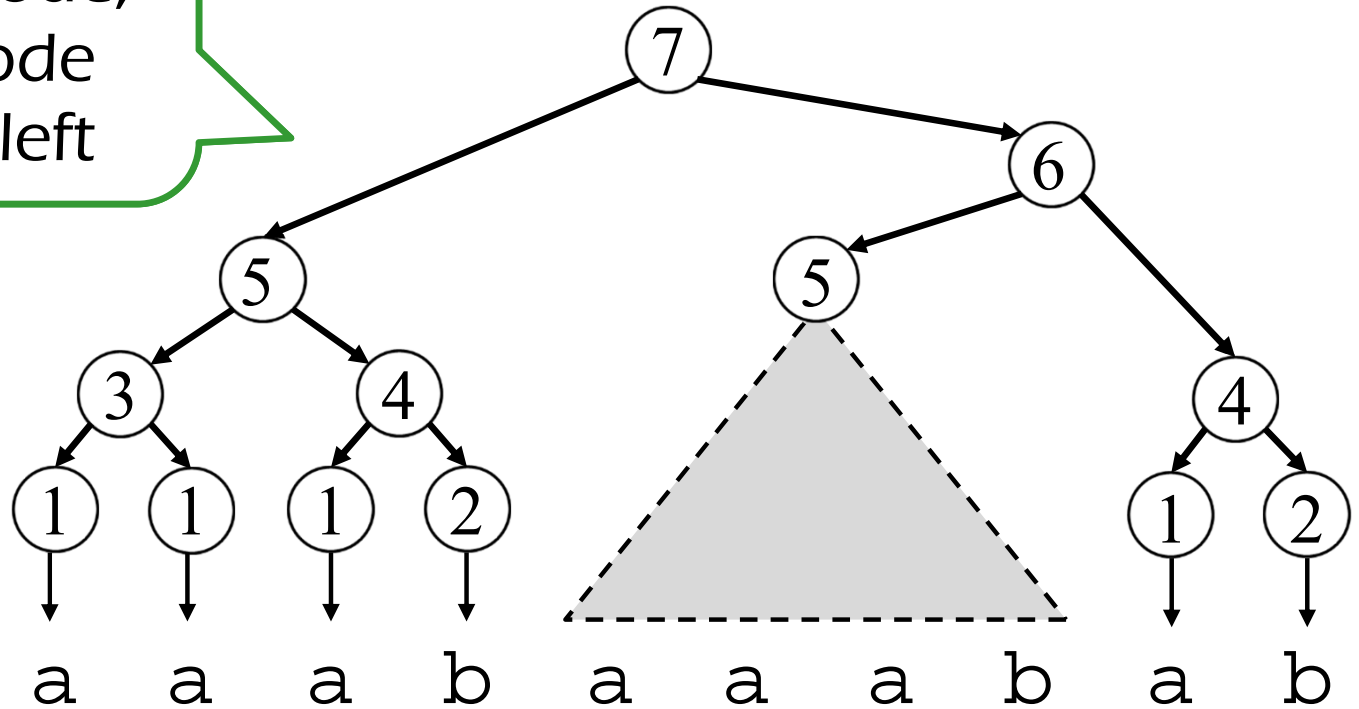
derivation tree for SLP  $G$



# G-parsing

Prune the subtree rooted at internal node, if the label of the node also appears to the left

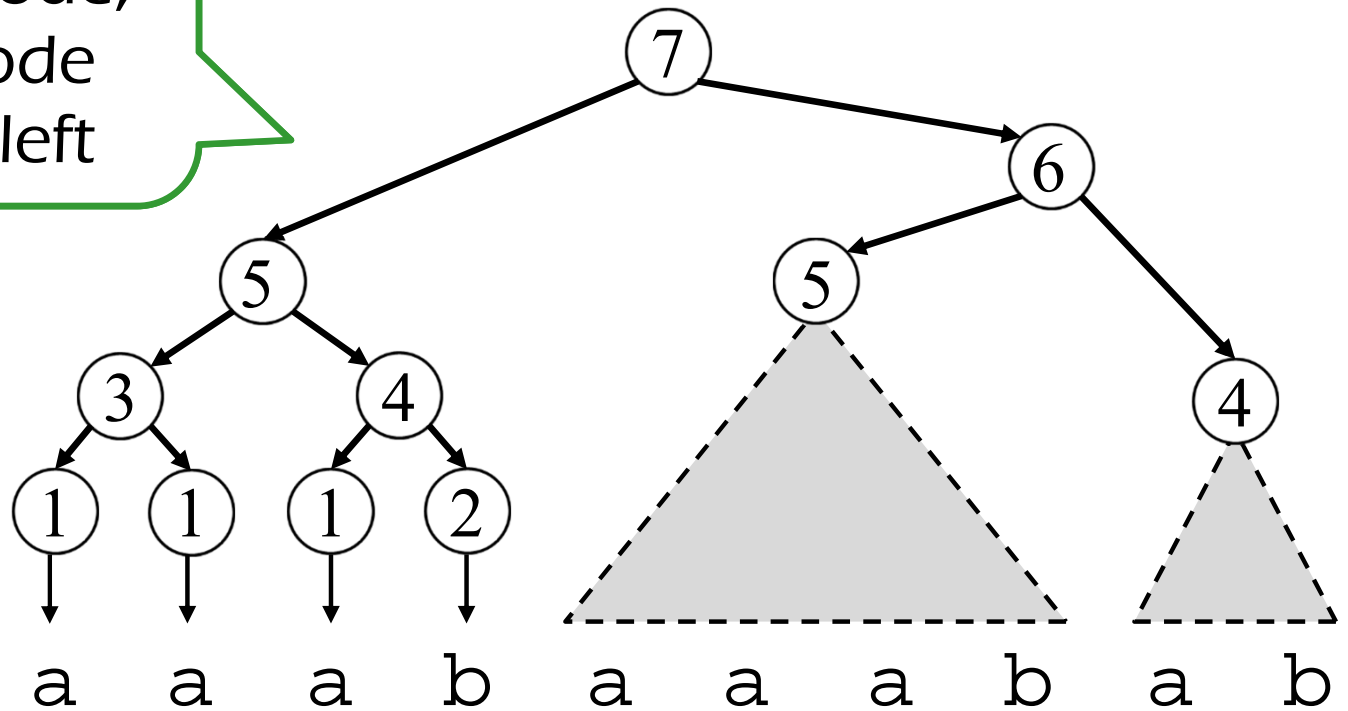
derivation tree for SLP  $G$



# G-parsing

Prune the subtree rooted at internal node, if the label of the node also appears to the left

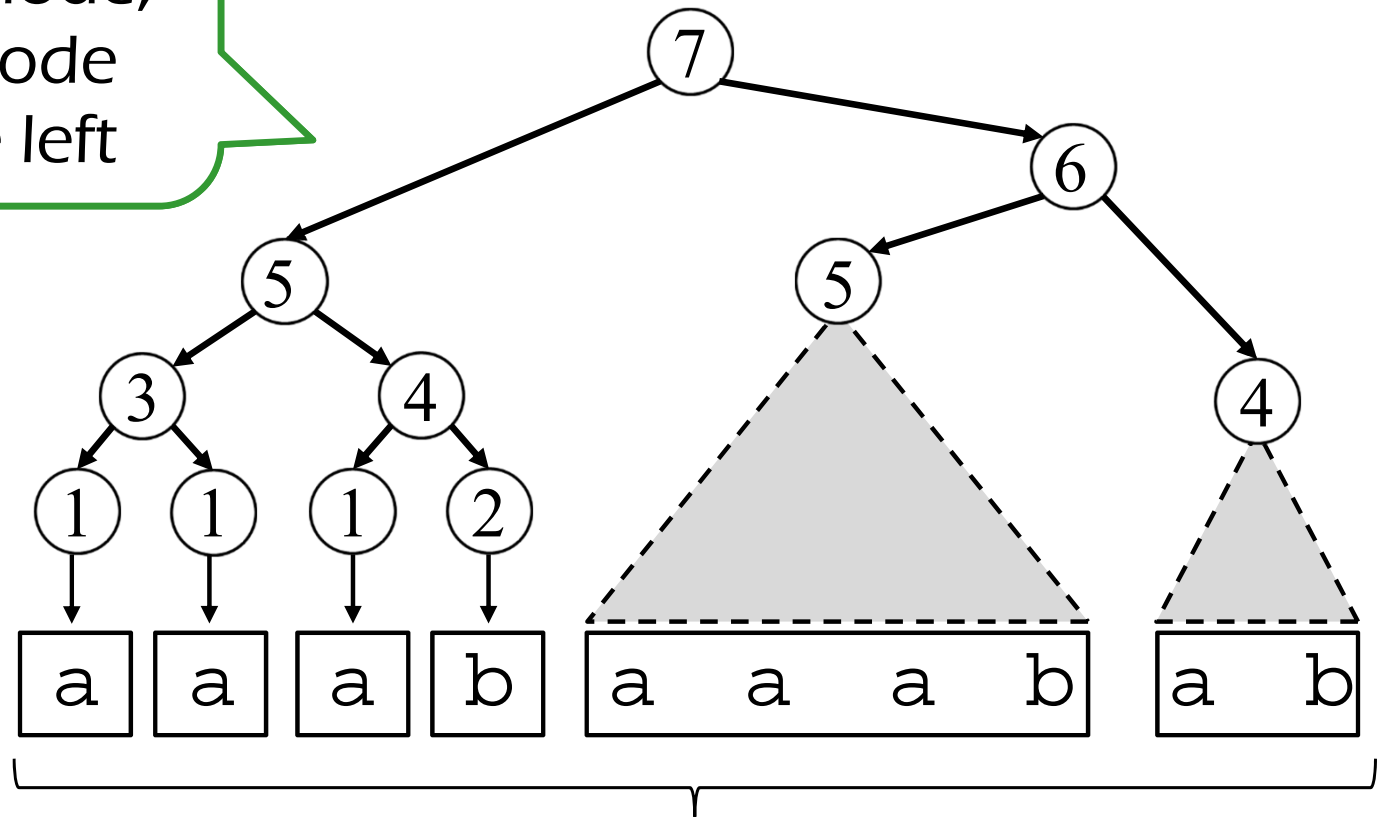
derivation tree for SLP  $G$



# G-parsing

Prune the subtree rooted at internal node, if the label of the node also appears to the left

derivation tree for SLP  $G$



$G$ -parsing of string  $w$

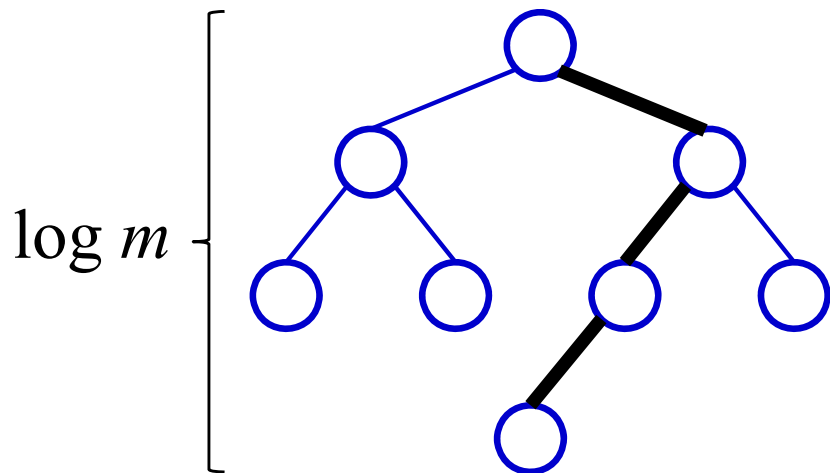
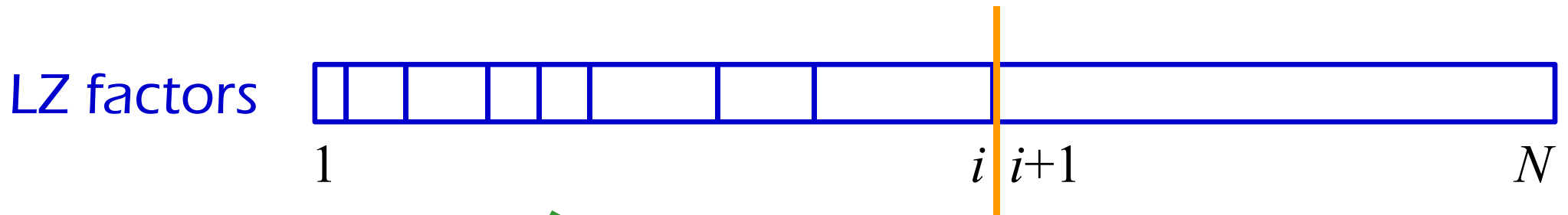
# $G$ -parsing [Cont.]

Lemma 1 [Rytter 2003]

For SLP  $G$  of size  $n$ , the  $G$ -parsing contains  $O(n)$  blocks and can be computed in  $O(n)$  time.

- ✓ Each block of  $G$ -parsing is represented by its beginning and ending positions, thus taking  $O(1)$  space.

# LZ78 Factorization on SLP



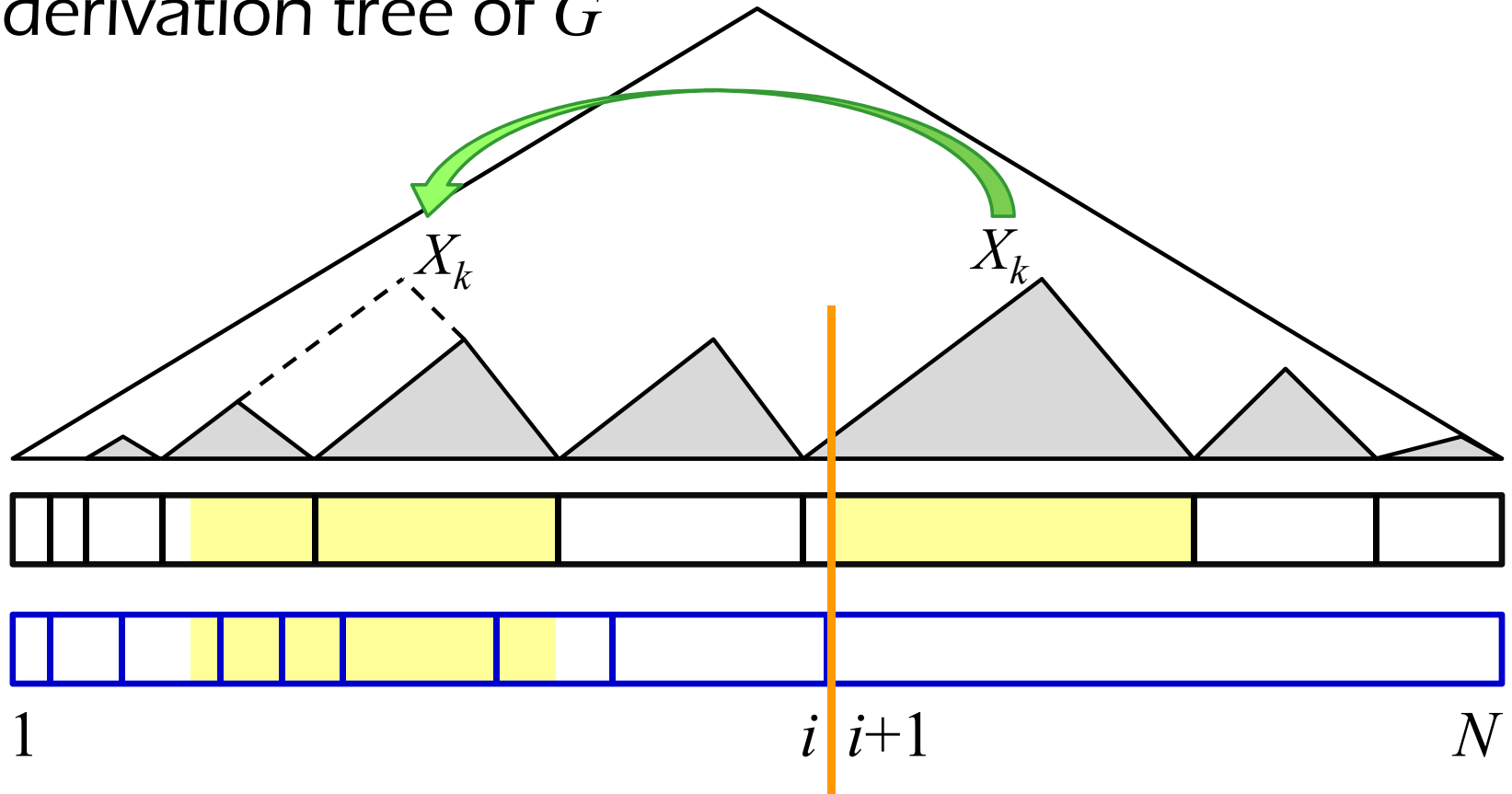
BST for previous  
LZ factors

- ✓ Suppose we have computed LZ factors for  $w[1..i]$ .
- ✓ We compare  $w[i+1..N]$  and previous LZ factors in BST.
- ✓ Each new LZ factor requires at most  $\log m$  comparisons.

How do we compare previous LZ factor  $f_j$  and  $w[i+1..N]$  ?

# LZ78 Factorization on SLP [Cont.]

derivation tree of  $G$

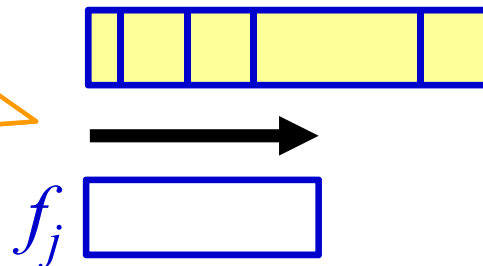


$G$ -parsing

LZ factors

1  $i$   $i+1$   $N$

We compare  $f_j$  and suffixes of LZ factors.



Next LZ factor is at least of this length + 1.



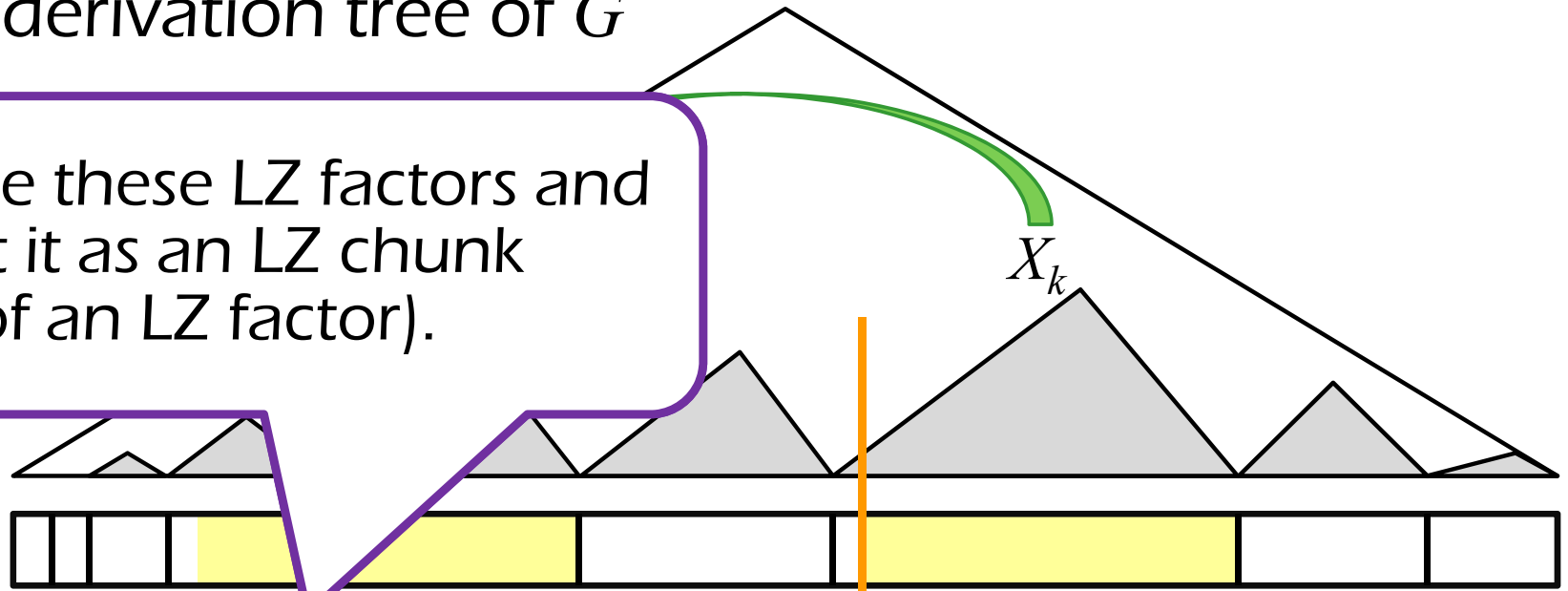


# LZ78 Factorization on SLP [Cont.]

derivation tree of  $G$

We merge these LZ factors and represent it as an LZ chunk (a suffix of an LZ factor).

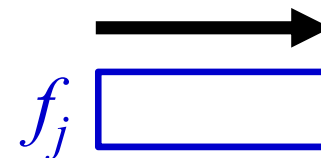
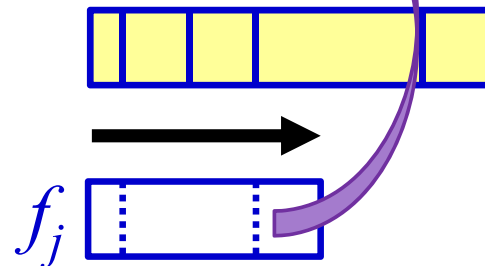
$G$ -parsing



~~LZ factors~~



LZ chunks



# LZ78 Factorization on SLP [Cont.]

- ✓ Towards an  $O(n + m \log m)$  bound:
  - Each LZ factor is computed by  $O(\log m)$  comparisons to previous LZ factors.
  - The total number of LZ chunks that are involved in comparisons is  $O(m \log m)$ .
  - So, what remains is how to perform LCP query for two any LZ chunks in  $O(1)$  time!

# LCP of Chunks

- ✓ Divide chunks into three types:
  1. Short chunks of length  $\leq \log m$
  2. Medium chunks of length  $\leq \log^2 m$
  3. Long chunks of length  $> \log^2 m$
- ✓ Maintain a dynamic LCP data structure for each of them.

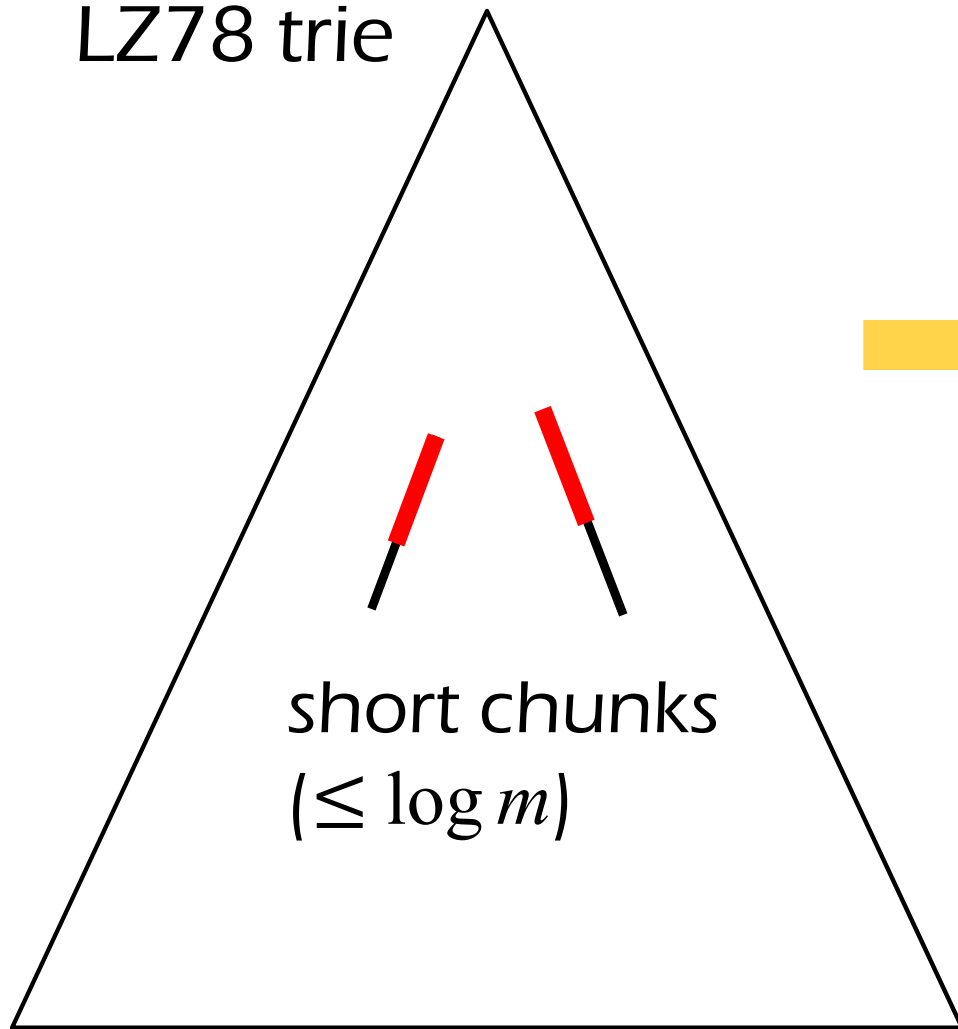
# LCP of Short Chunks

## Lemma 2

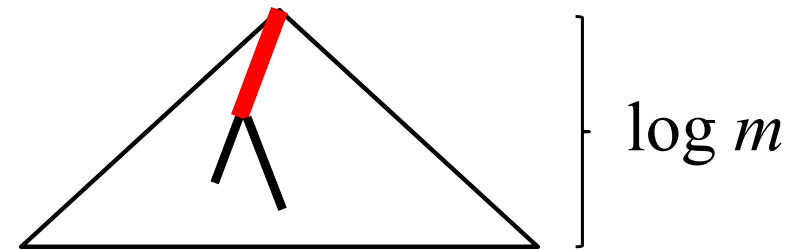
We can maintain a structure of size  $O(m \log m)$  which supports updates in  $O(\log m)$  time and LCP query of two **short chunks** ( $\leq \log m$ ) in  $O(1)$  time.

# LCP of Short Chunks [Cont.]

LZ78 trie



trie of all short chunks



$O(m \log m)$  nodes in this trie.

LCA gives LCP in  $O(1)$  time.

A careful update procedure takes only  $O(\log m)$  time, independently of alphabet.

# LCP of Medium Chunks

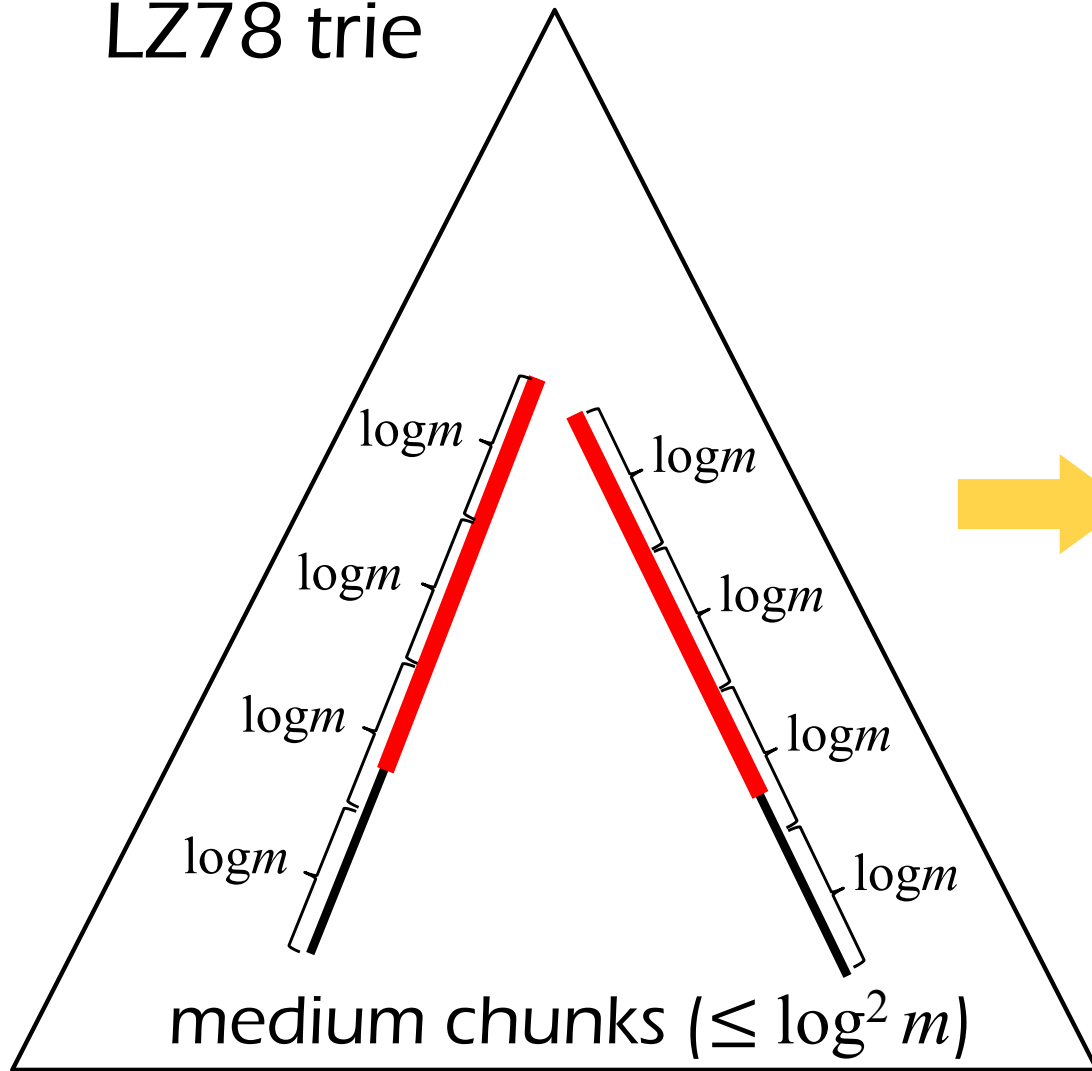
## Lemma 3

We can maintain a structure of size  $O(m \log m)$  which supports updates in  $O(\log m)$  time and LCP query of two **medium chunks** ( $\leq \log^2 m$ ) of length being multiple of  $\log m$  in  $O(1)$  time.

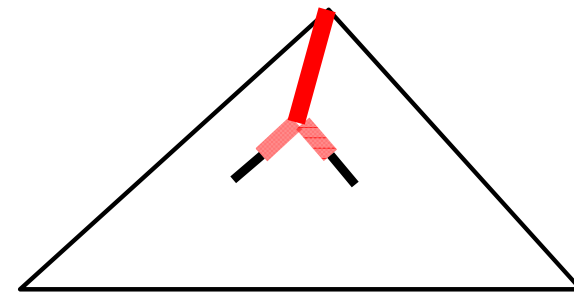
- ✓ The structure for short chunks works independently of the alphabet size.
- ✓ Regard a chunk of length  $\log m$  as a meta-character, and use the structure for short chunks.

# LCP of Medium Chunks [Cont.]

LZ78 trie



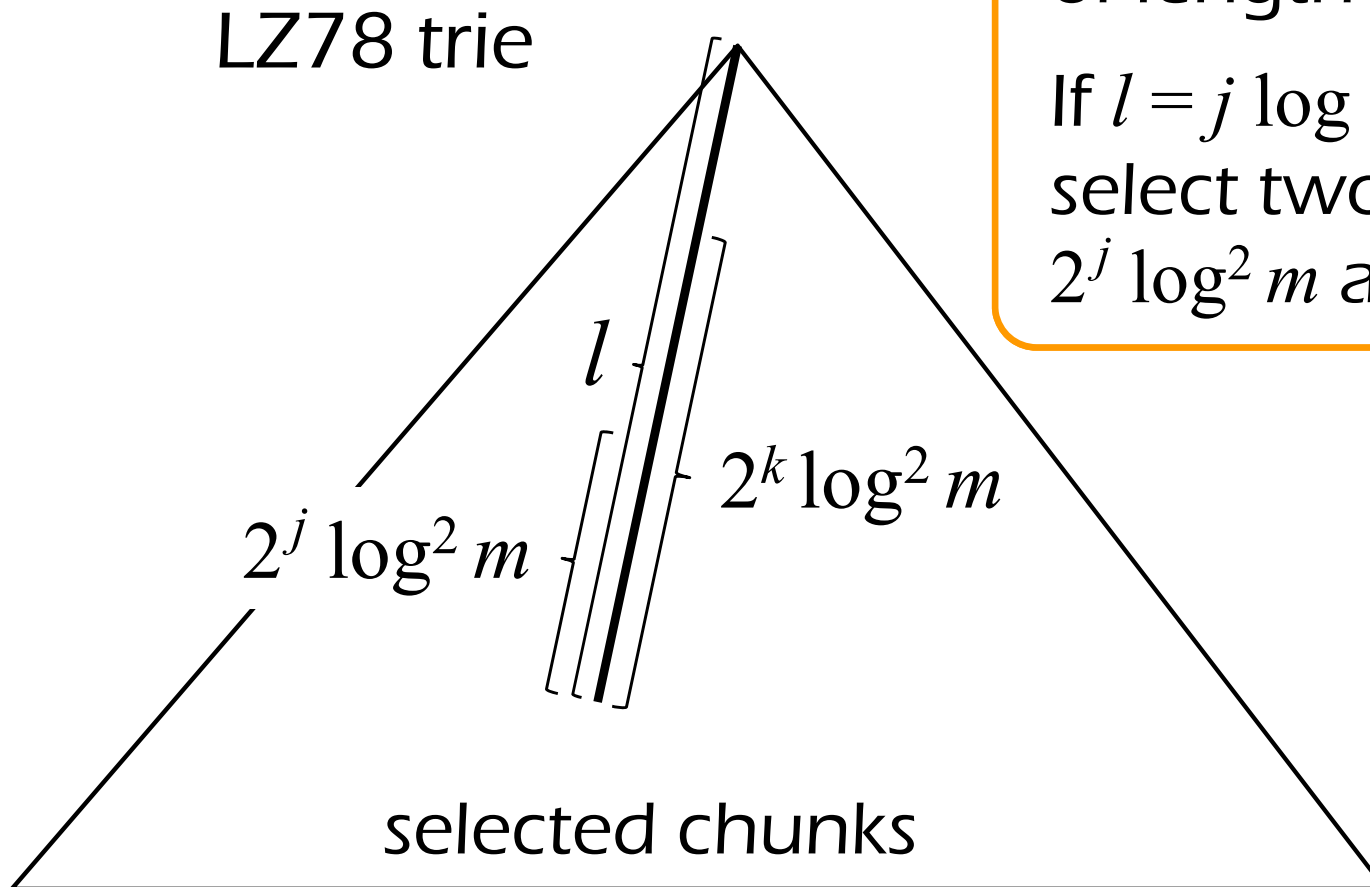
trie of short chunks  
over meta-characters



LCA gives  $\left\lfloor \frac{\text{LCP}}{\log m} \right\rfloor$ .

Last fragment can be  
computed by Lemma 2.

# Towards LCP of Long Chunks



Let  $l$  be length of LZ factor of length  $> \log^2 m$ .

If  $l = j \log m + k \pmod{\log^2 m}$ , select two chunks of length  $2^j \log^2 m$  and  $2^k \log^2 m$ .



# Towards LCP of Long Chunks [Cont.]

## Lemma 4

We can maintain a structure of size  $O(m \log m)$  which supports updates in  $O(\log m)$  time and LCP query of two **selected chunks** ( $2^k \log^2 m$ ) of the same length in  $O(1)$  time.

- ✓ Details are omitted here.
- ✓ Please see the paper for details.

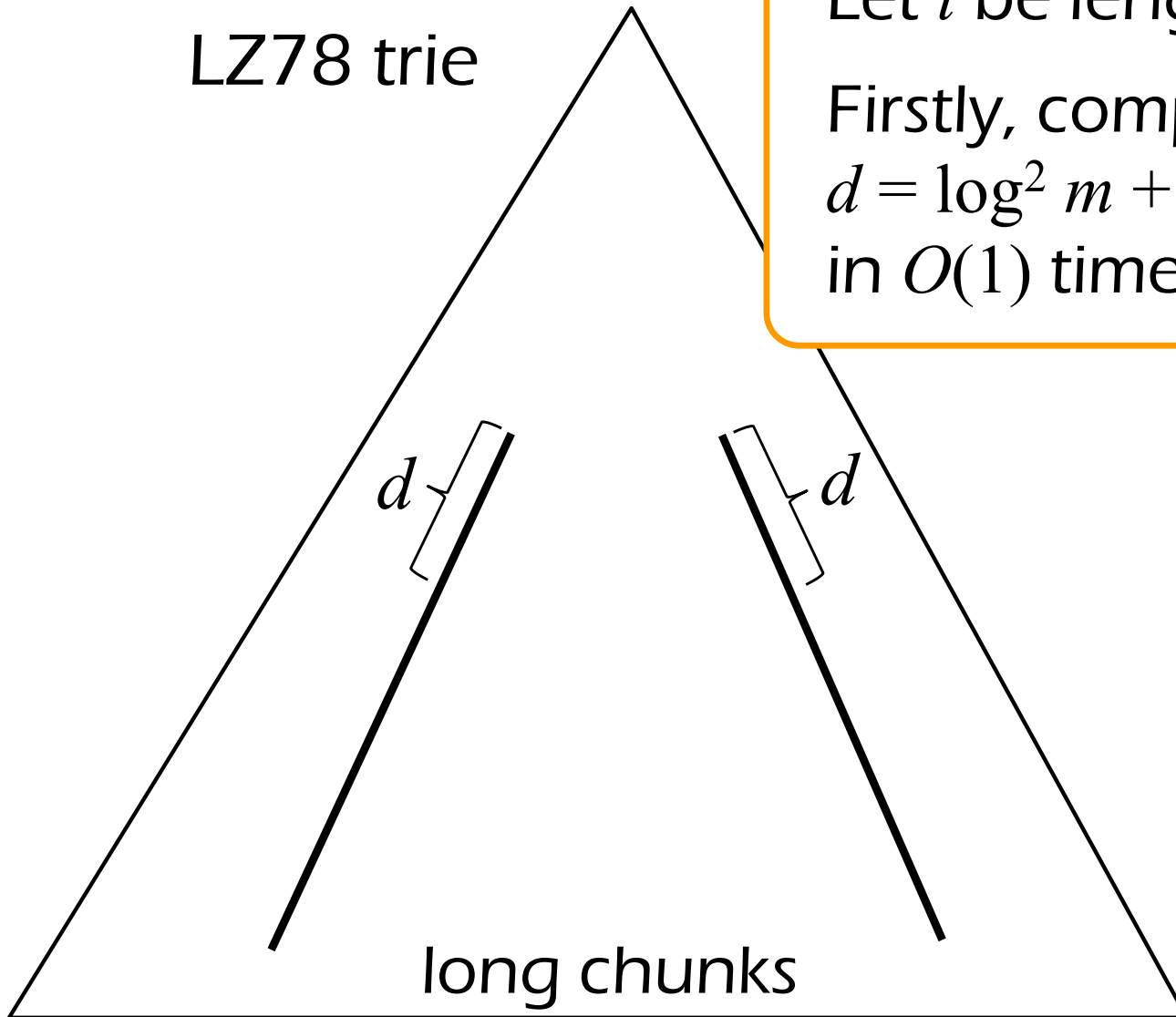
# LCP of Long Chunks

## Lemma 5

We can maintain a structure of size  $O(m \log m)$  which supports updates in  $O(\log m)$  time and LCP query of two **long chunks** ( $> \log^2 m$ ) in  $O(1)$  time.

# LCP of Long Chunks [Cont.]

LZ78 trie

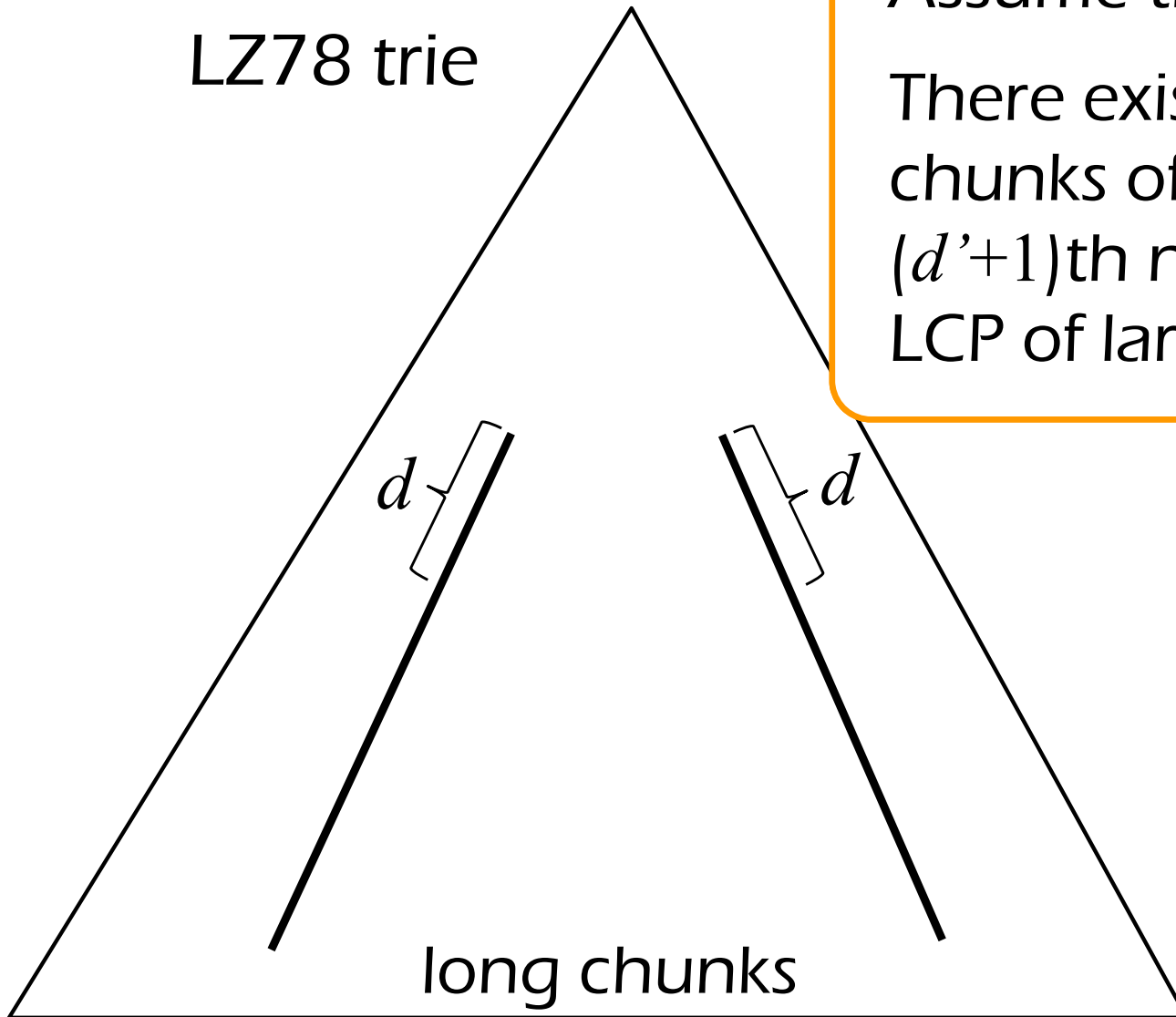


Let  $t$  be length of chunks.

Firstly, compare prefixes of length  $d = \log^2 m + (t \bmod \log^2 m)$  in  $O(1)$  time by Lemmas 2 & 3.

# LCP of Long Chunks [Cont.]

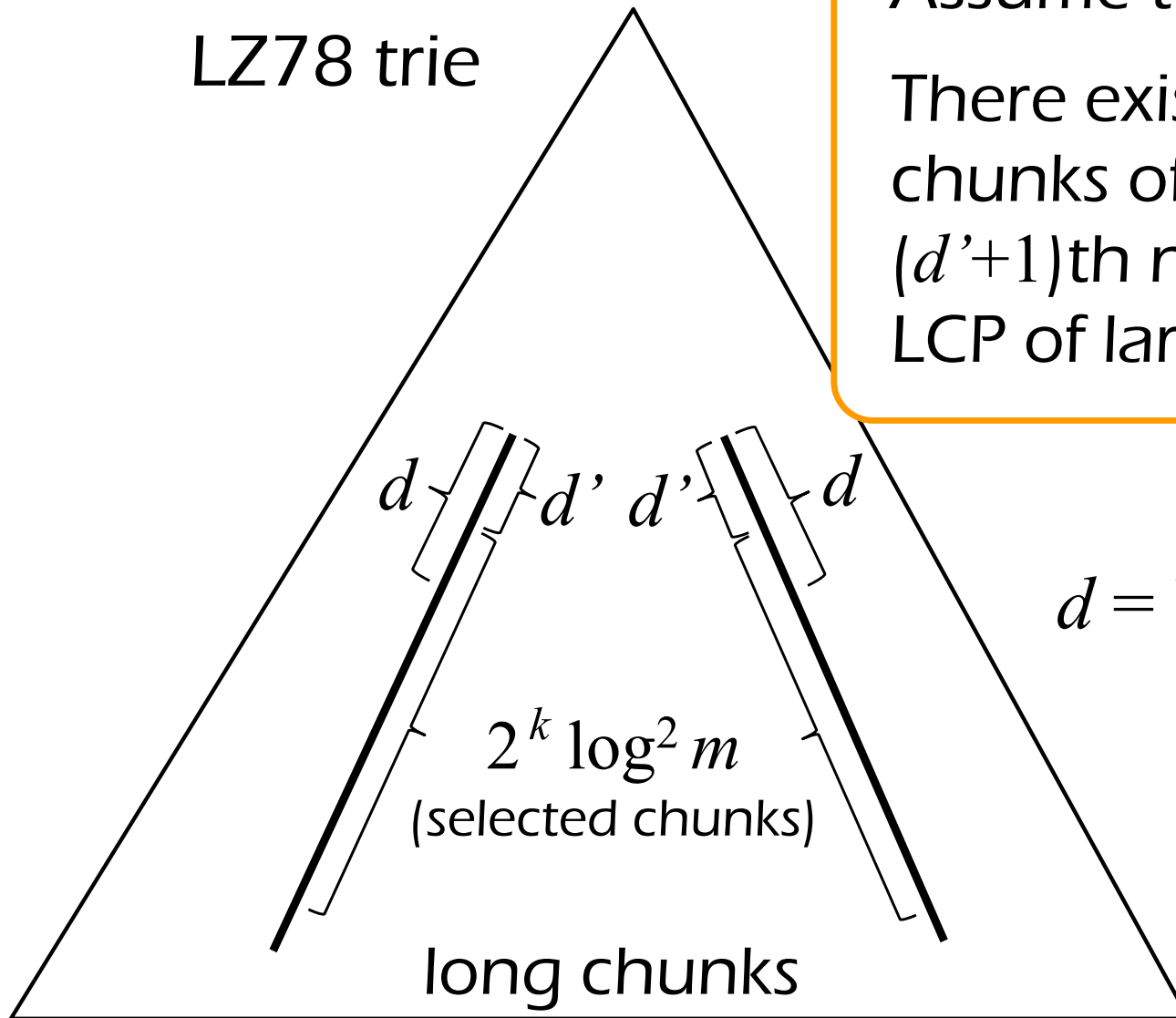
LZ78 trie



Assume the prefixes are equal.  
There exists  $d' \leq d$  s.t. selected chunks of same length starts at  $(d'+1)$ th node, so we compute LCP of largest selected chunks.

# LCP of Long Chunks [Cont.]

LZ78 trie



Assume the prefixes are equal.  
There exists  $d' \leq d$  s.t. selected chunks of same length starts at  $(d'+1)$ th node, so we compute LCP of largest selected chunks.

$$d = \log^2 m + (t \bmod \log^2 m)$$

# Dynamic $O(1)$ -time Chunk LCP

## Theorem 3

For a growing trie with at most  $m$  nodes, we can maintain a structure of size  $O(m \log m)$  which supports updates in  $O(\log m)$  time and LCP query for two **any chunks** (i.e., any paths) in  $O(1)$  time.

# Concluding Remarks

- ✓ We proposed an  $O(n + m \log m)$  time & space algorithm which converts any SLP to LZ78.
- ✓ An  $O(n + m)$  space &  $O((n + m) \log m)$  time variant exists.
- ✓ The dynamic LCP data structure can be used for any growing trie.