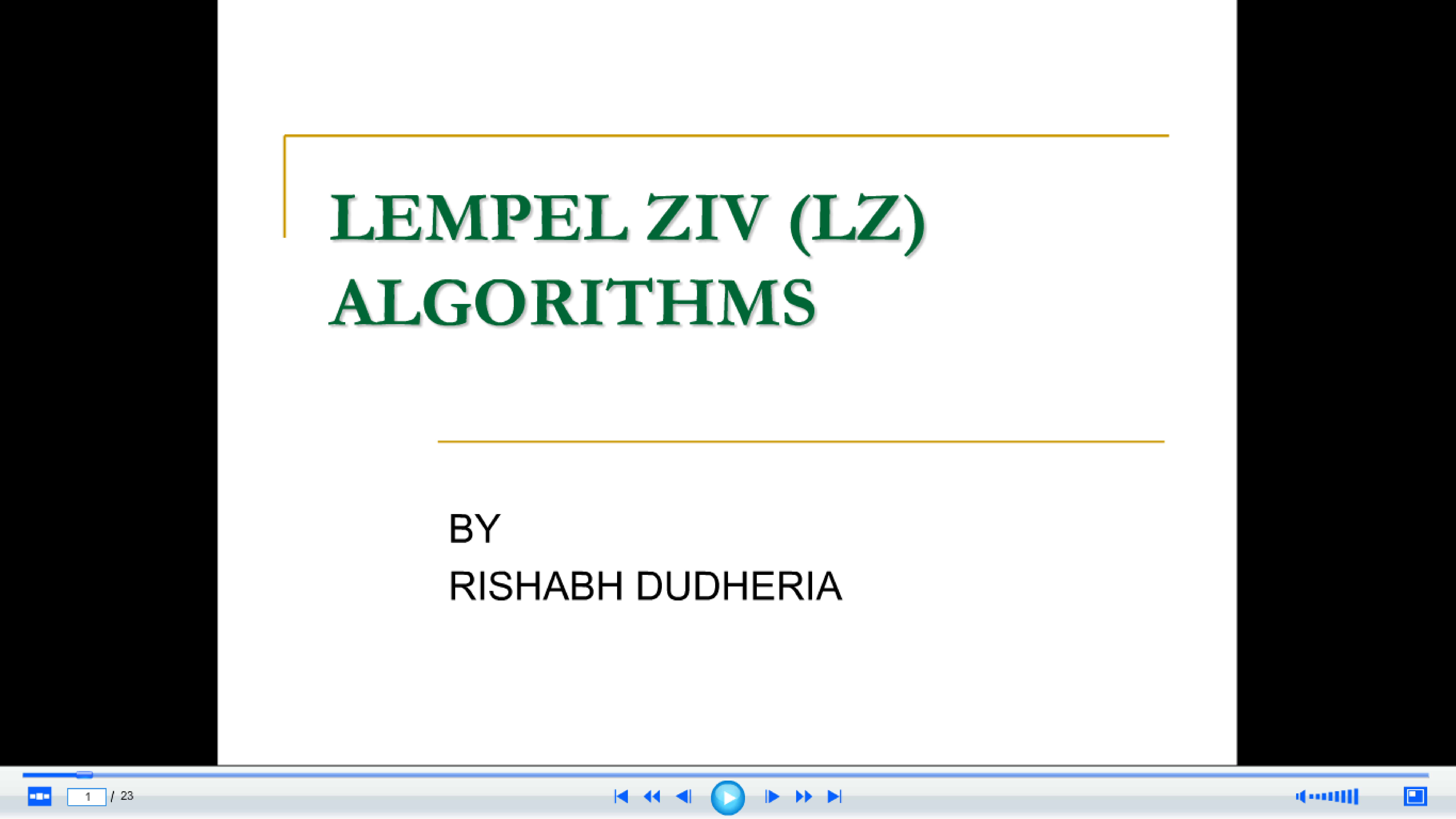
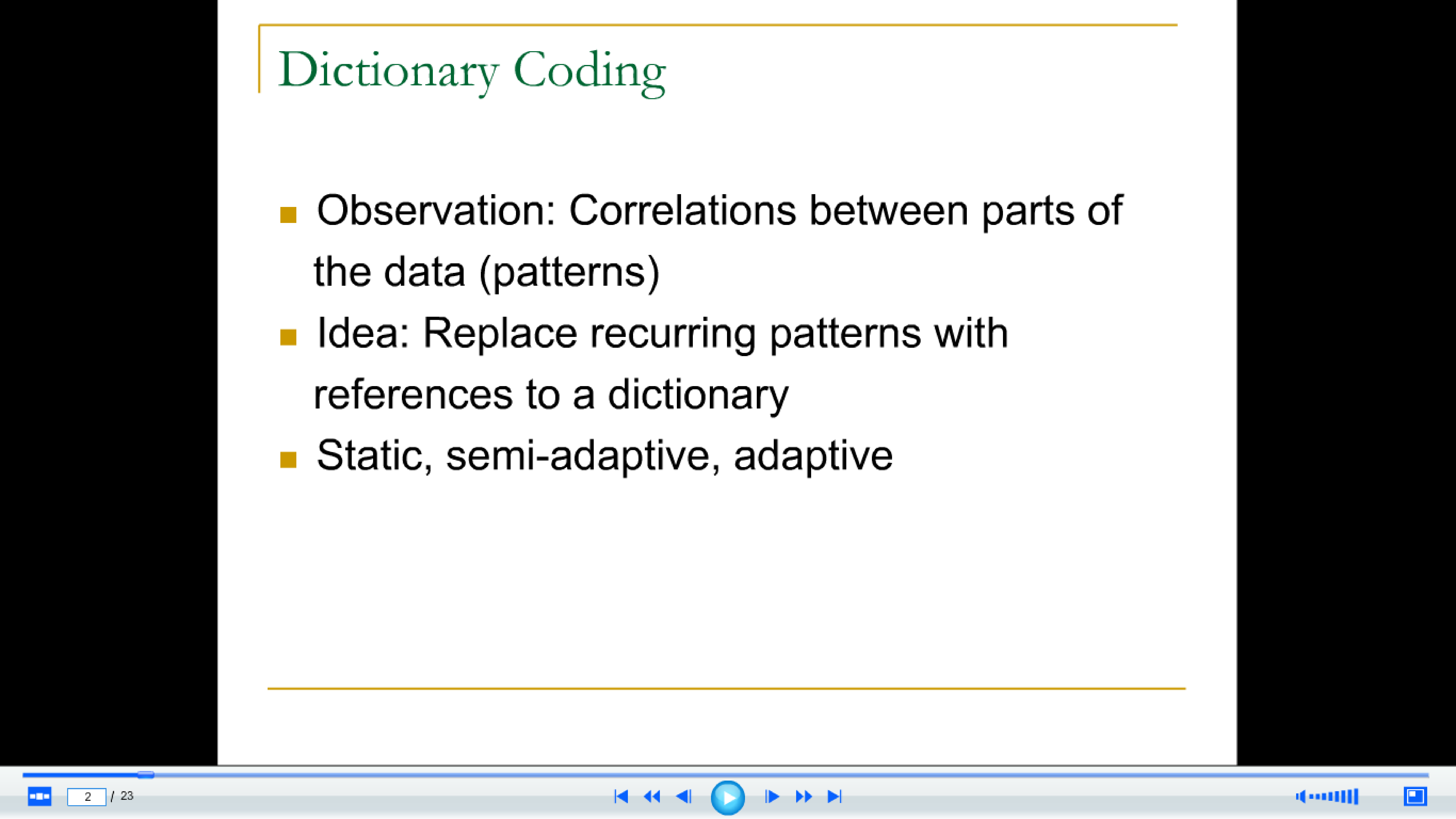


LEMPEL ZIV (LZ) ALGORITHMS

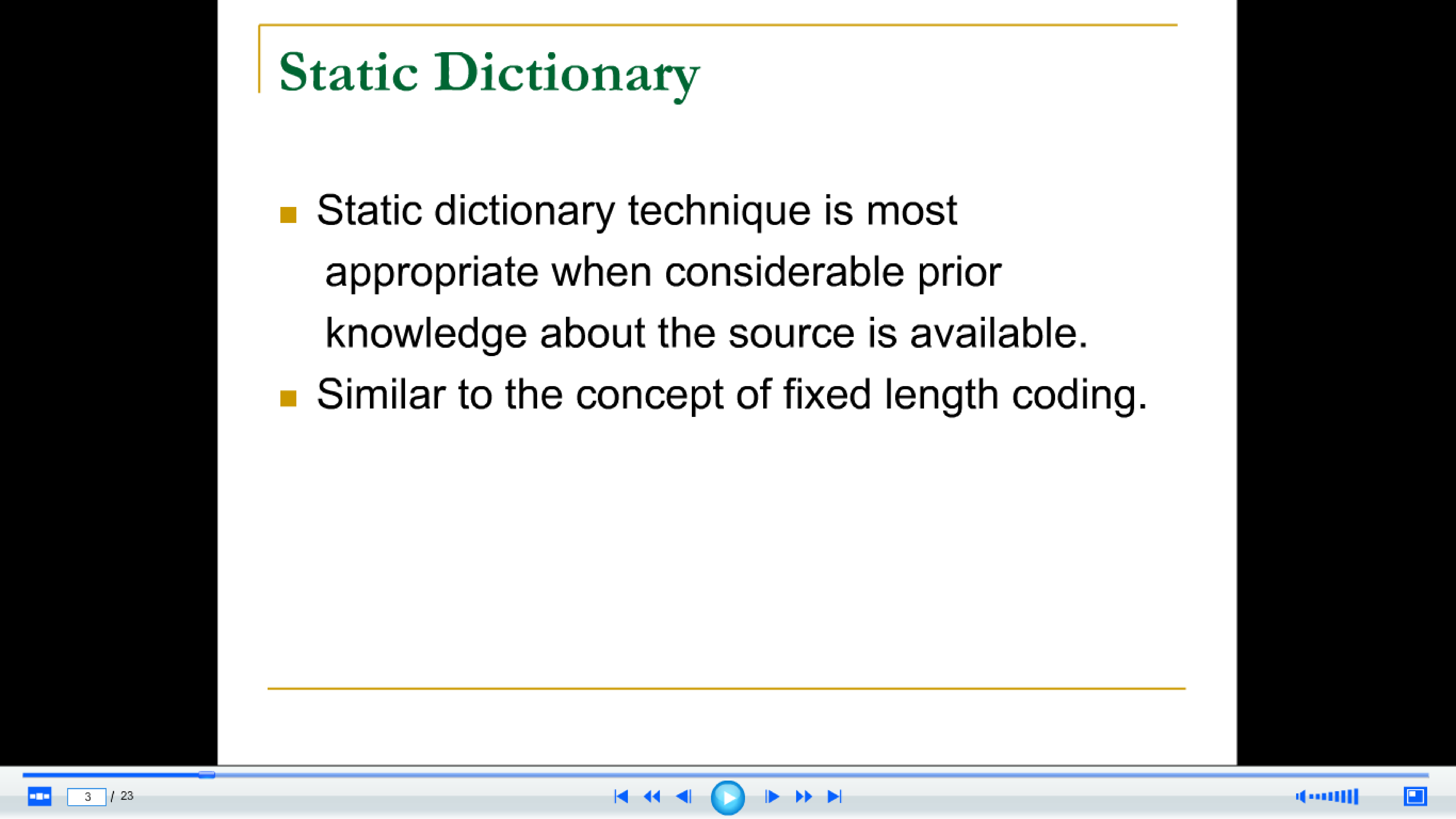
BY

RISHABH DUDHERIA



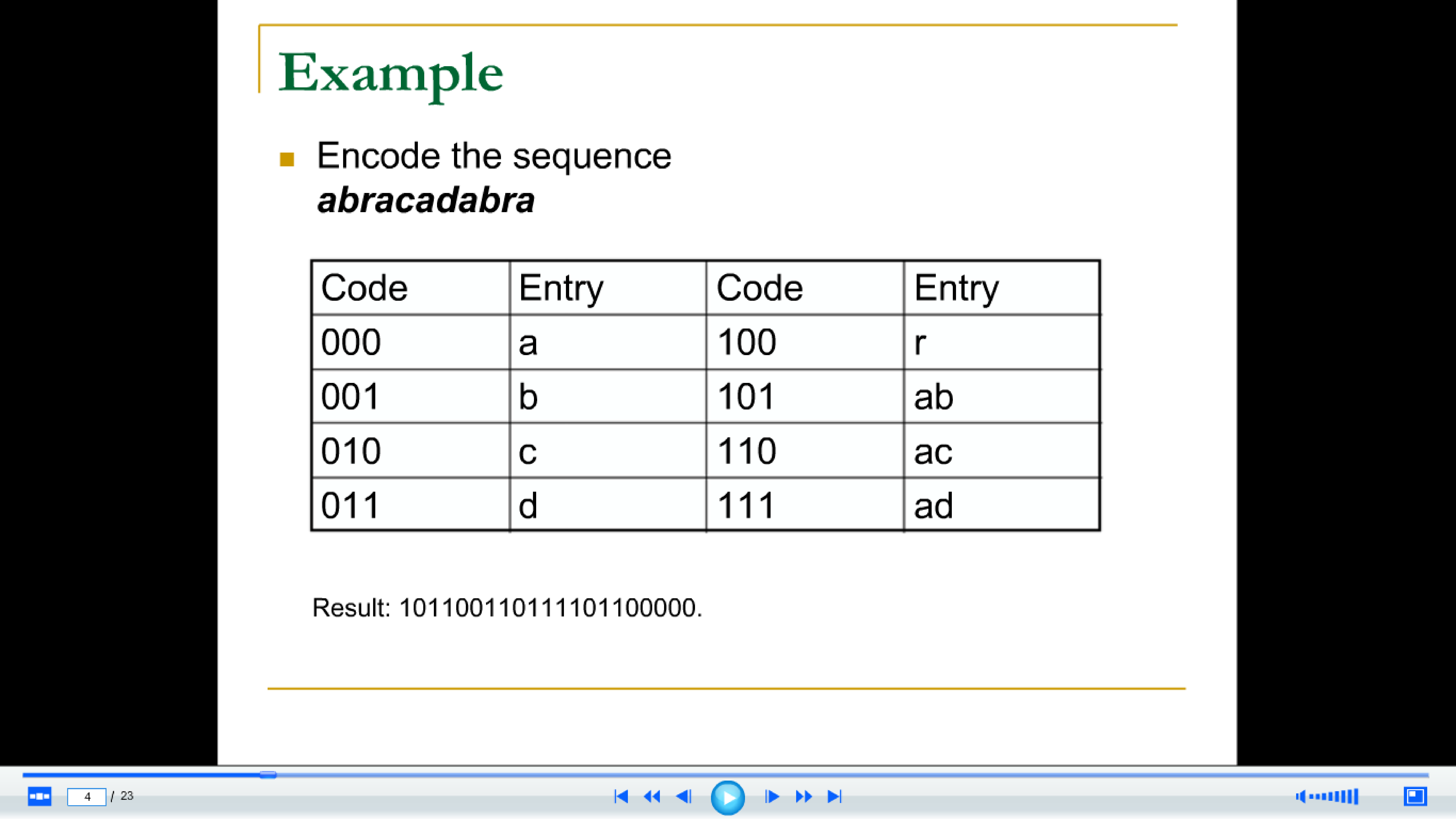
Dictionary Coding

- Observation: Correlations between parts of the data (patterns)
- Idea: Replace recurring patterns with references to a dictionary
- Static, semi-adaptive, adaptive



Static Dictionary

- Static dictionary technique is most appropriate when considerable prior knowledge about the source is available.
- Similar to the concept of fixed length coding.

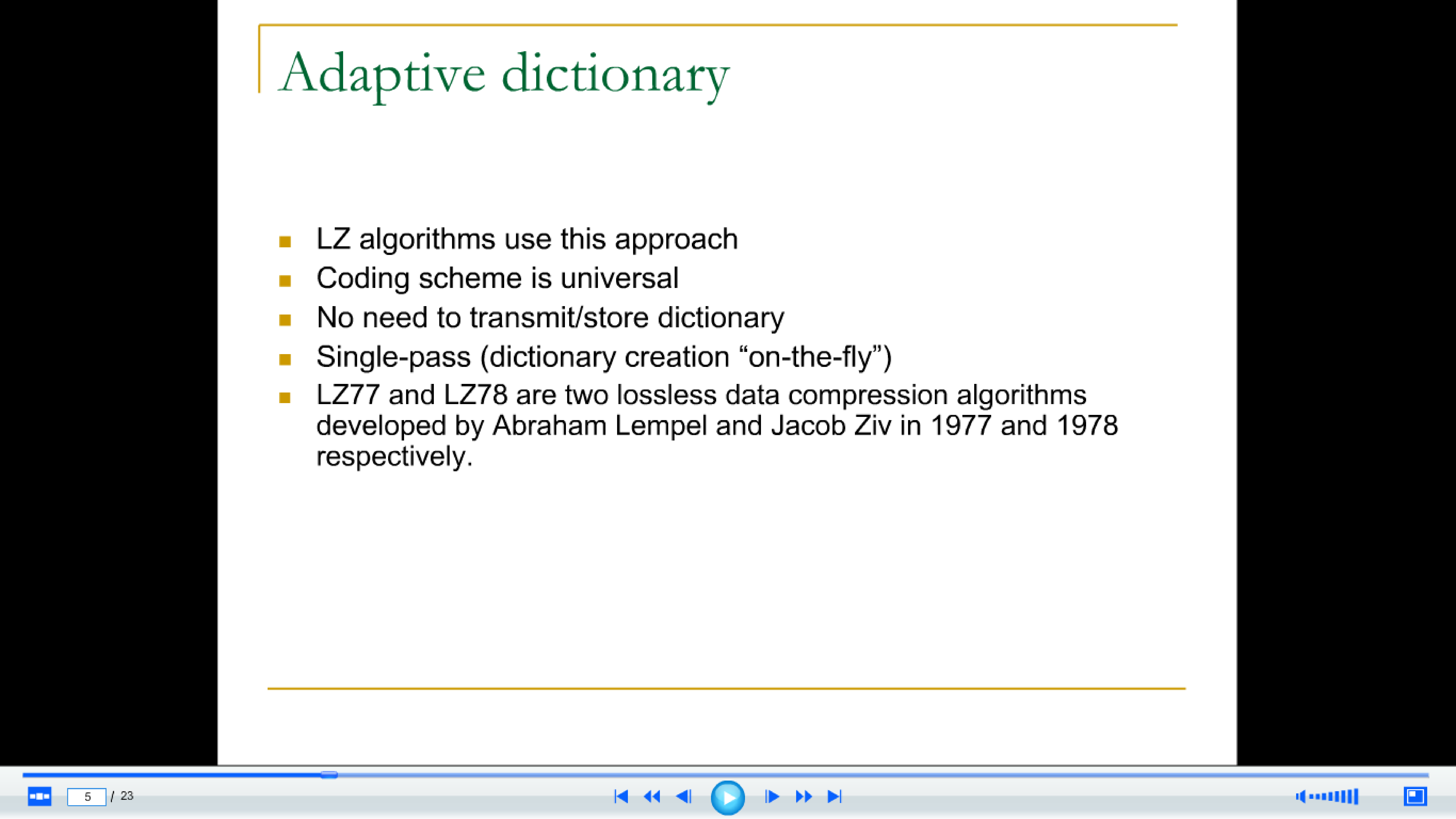


Example

- Encode the sequence
  abracadabra

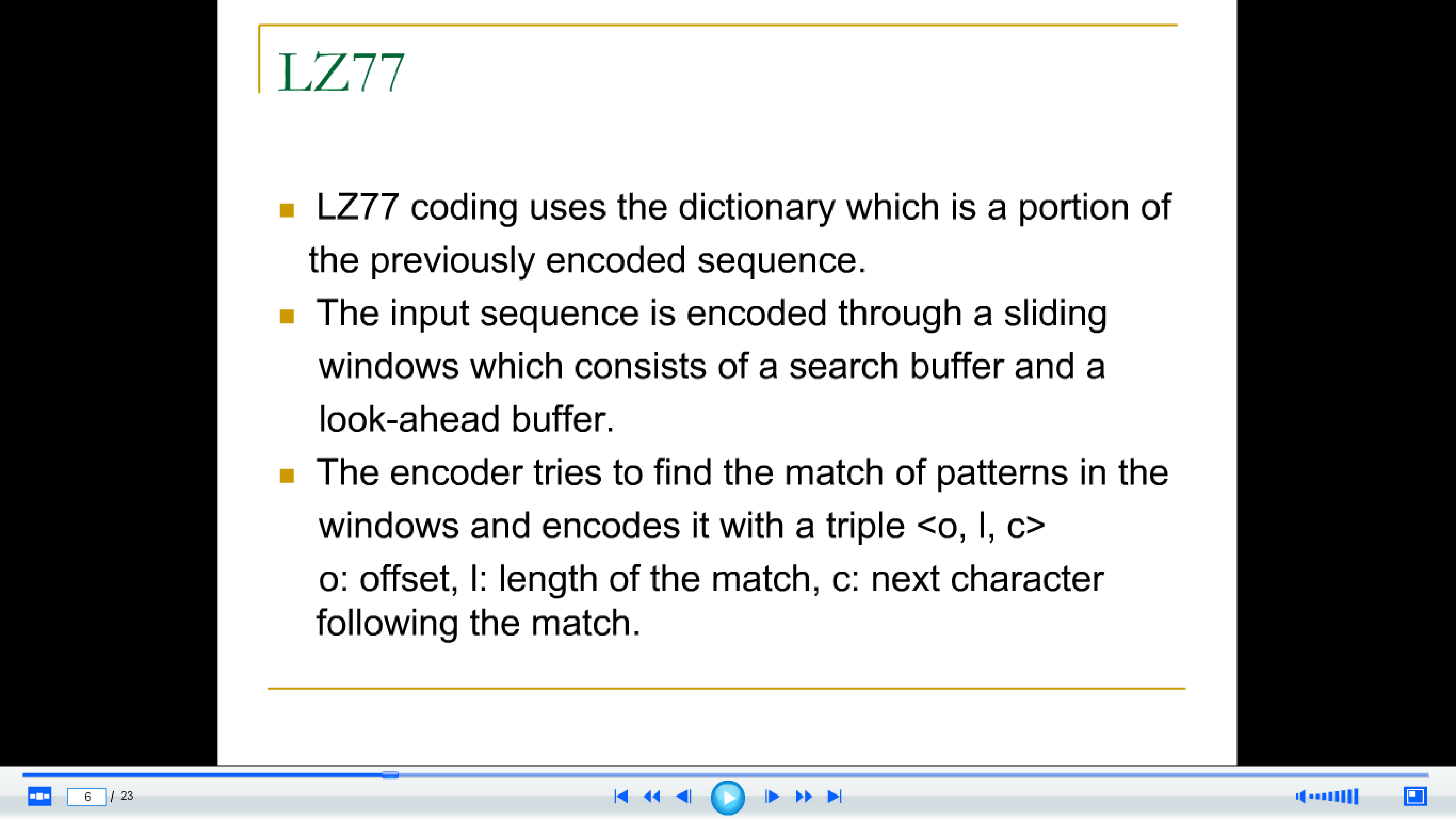
Code	Entry	Code	Entry
000	a	100	r
001	b	101	ab
010	c	110	ac
011	d	111	ad

Result: 1011001101111101100000.



Adaptive dictionary

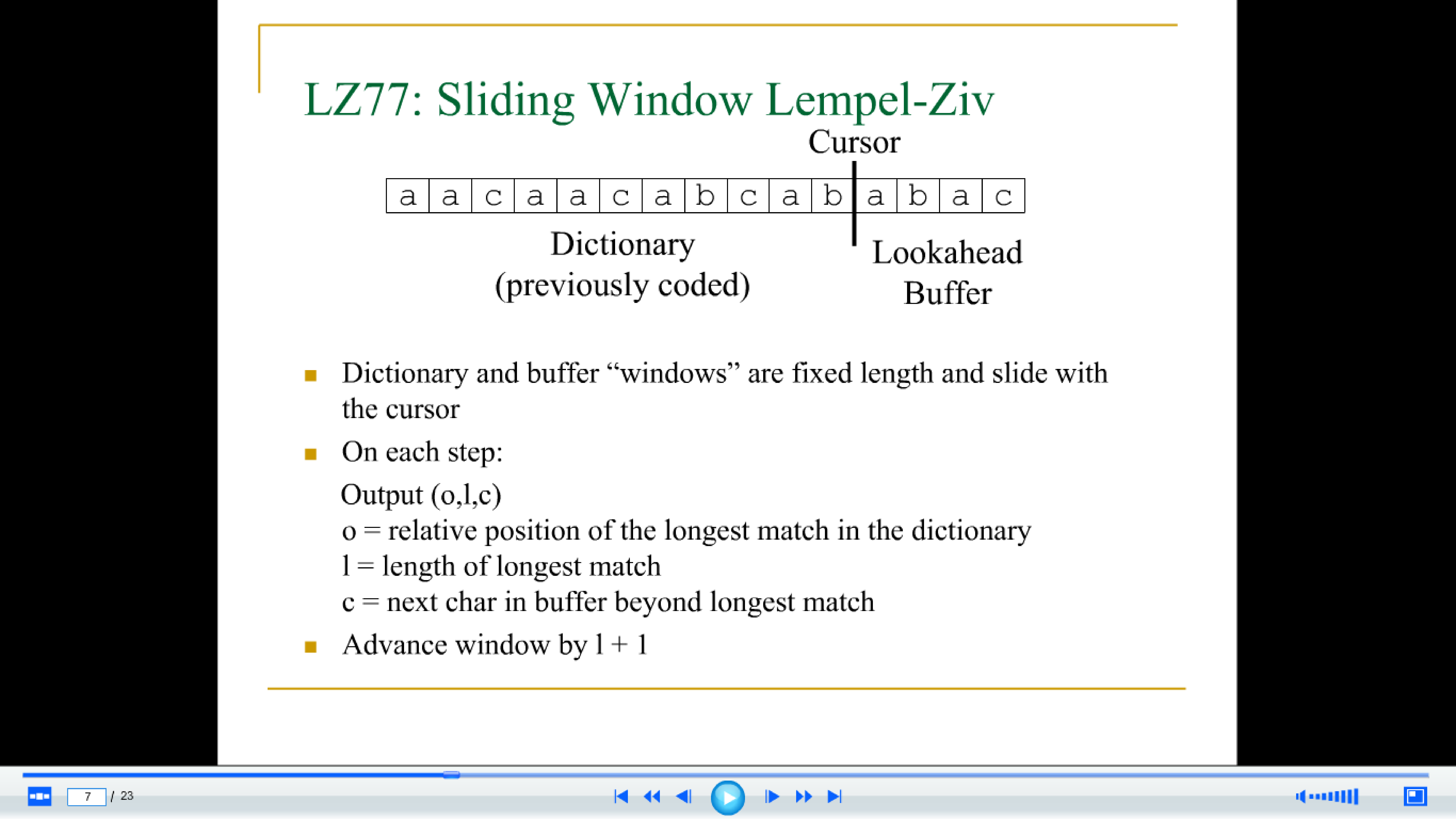
- LZ algorithms use this approach
- Coding scheme is universal
- No need to transmit/store dictionary
- Single-pass (dictionary creation “on-the-fly”)
- LZ77 and LZ78 are two lossless data compression algorithms developed by Abraham Lempel and Jacob Ziv in 1977 and 1978 respectively.



LZ77

- LZ77 coding uses the dictionary which is a portion of the previously encoded sequence.
- The input sequence is encoded through a sliding windows which consists of a search buffer and a look-ahead buffer.
- The encoder tries to find the match of patterns in the windows and encodes it with a triple <o, l, c>

  o: offset, l: length of the match, c: next character following the match.



LZ77: Sliding Window Lempel-Ziv

Cursor

a	a	c	a	a	c	a	b	c	a	b	a	b	a	c

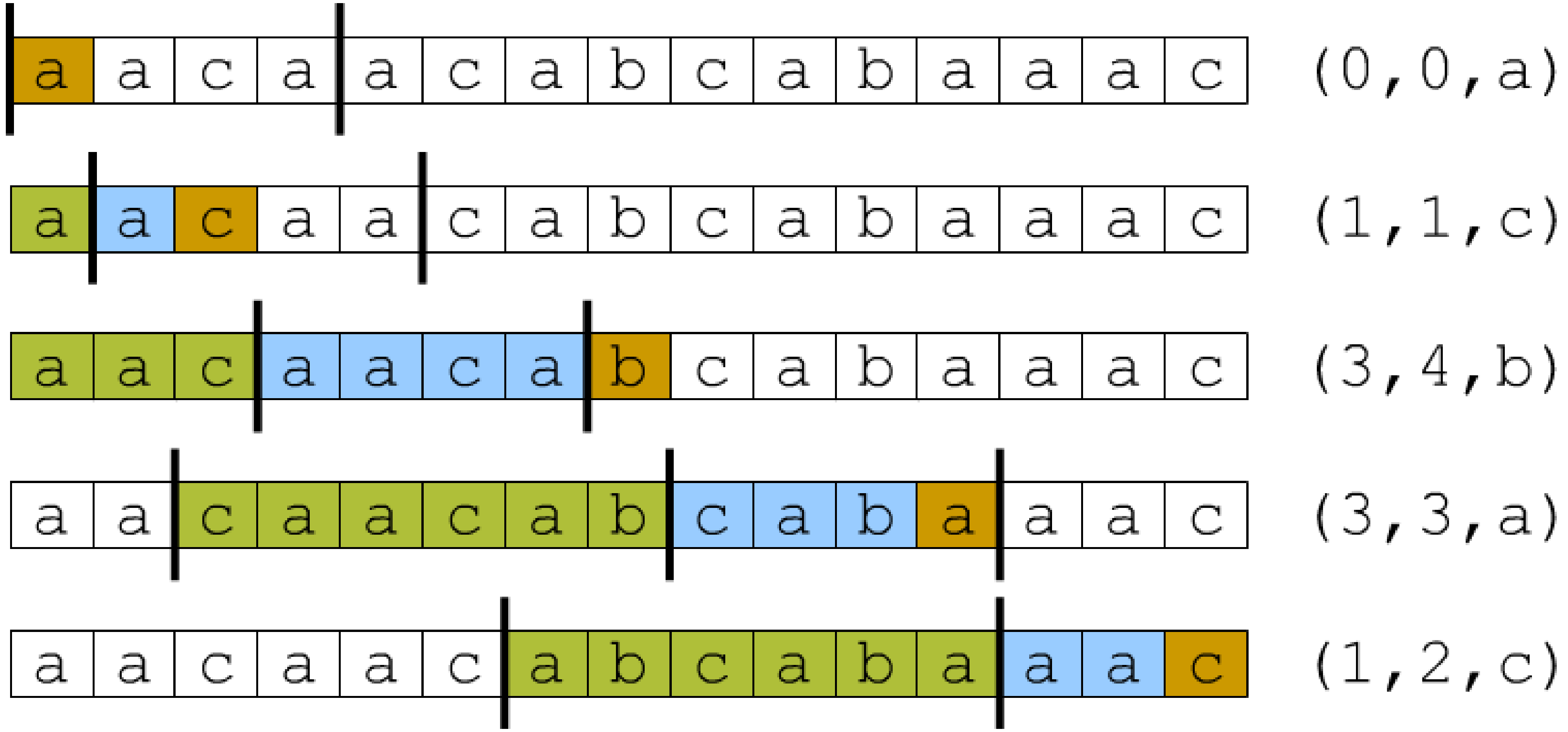
Dictionary
(previously coded)

Lookahead
Buffer

- Dictionary and buffer "windows" are fixed length and slide with the cursor
- On each step:

  Output (o,l,c)
  o = relative position of the longest match in the dictionary
  l = length of longest match
  c = next char in buffer beyond longest match
- Advance window by l + 1

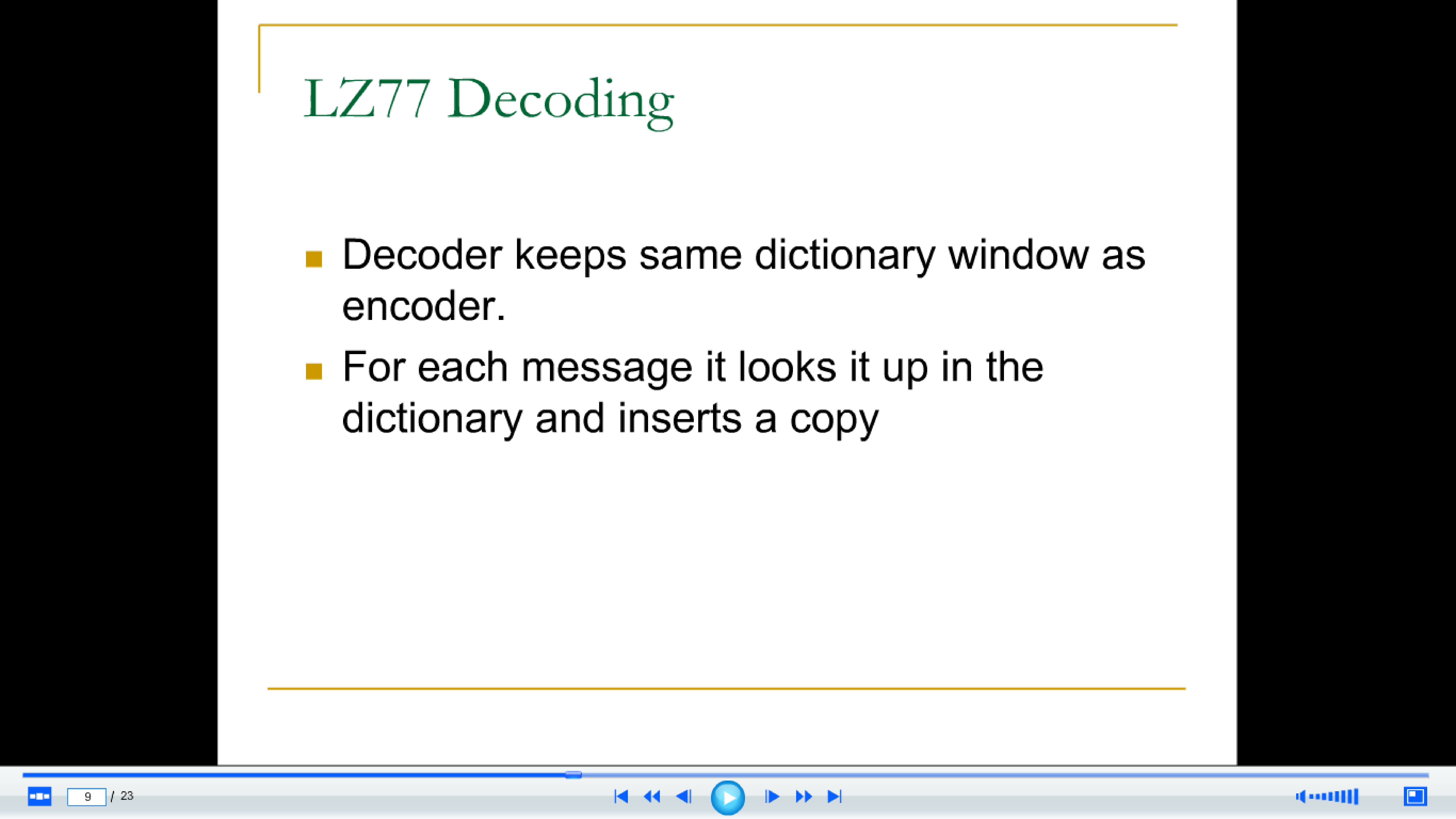
LZ77: Example



a	a	c	a	a	c	a	b	c	a	b	a	a	a	c	(0,0,a)
a	a	c	a	a	c	a	b	c	a	b	a	a	a	c	(1,1,c)
a	a	c	a	a	c	a	b	c	a	b	a	a	a	c	(3,4,b)
a	a	c	a	a	c	a	b	c	a	b	a	a	a	c	(3,3,a)
a	a	c	a	a	c	a	b	c	a	b	a	a	a	c	(1,2,c)

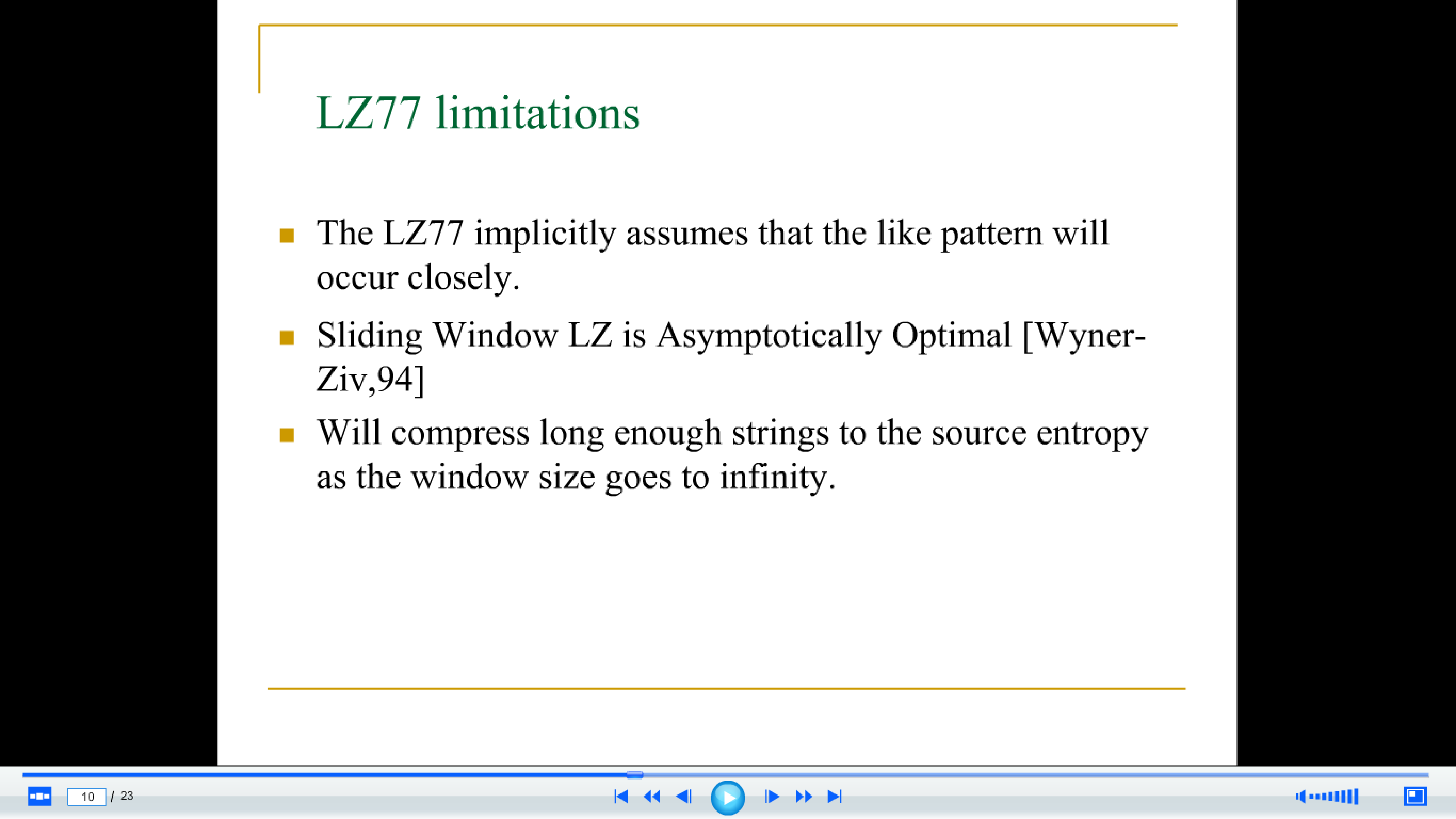
🟩 Dictionary (size = 6)

🟦 Longest match 🟧 Next character



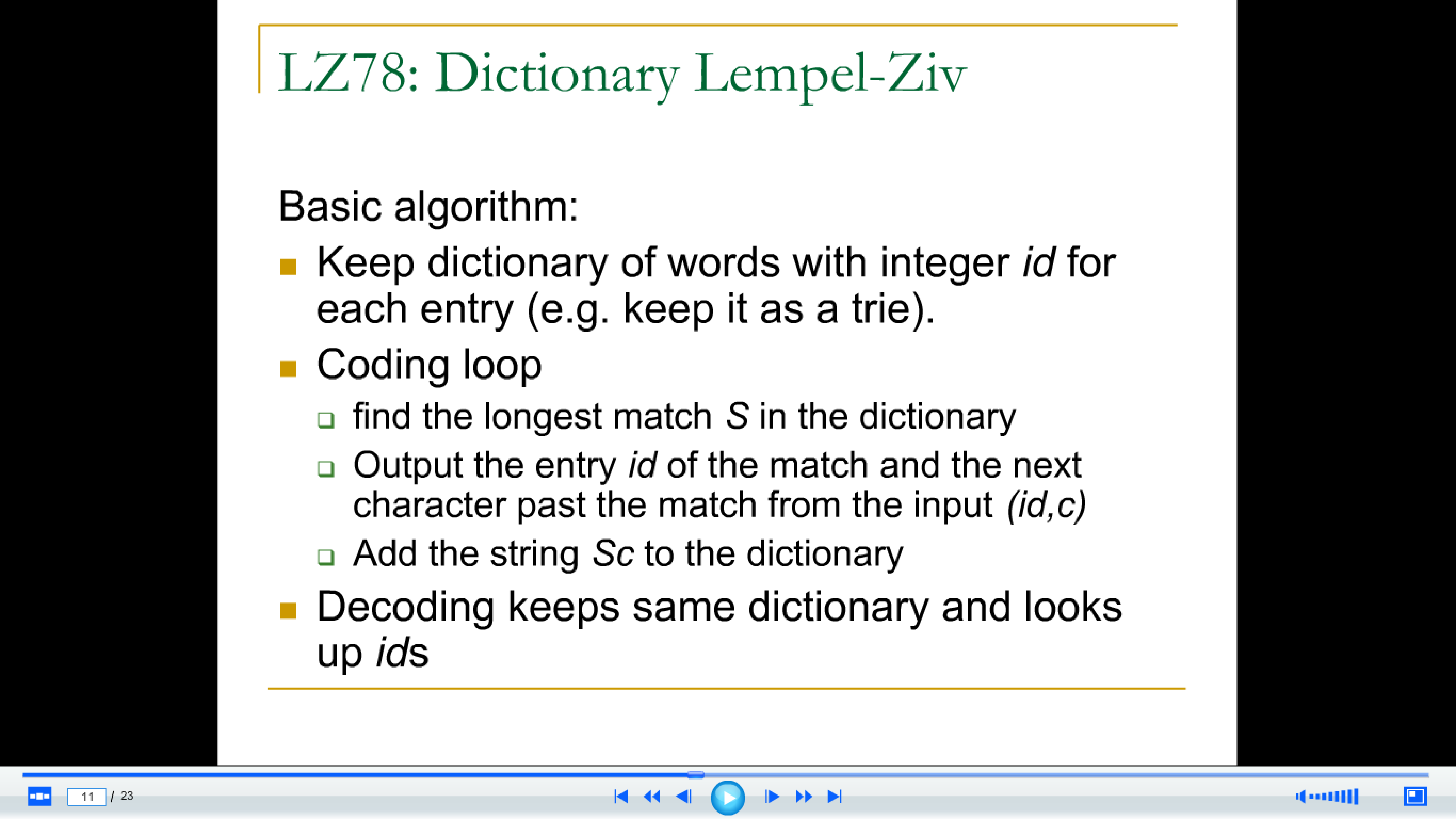
LZ77 Decoding

- Decoder keeps same dictionary window as encoder.
- For each message it looks it up in the dictionary and inserts a copy



LZ77 limitations

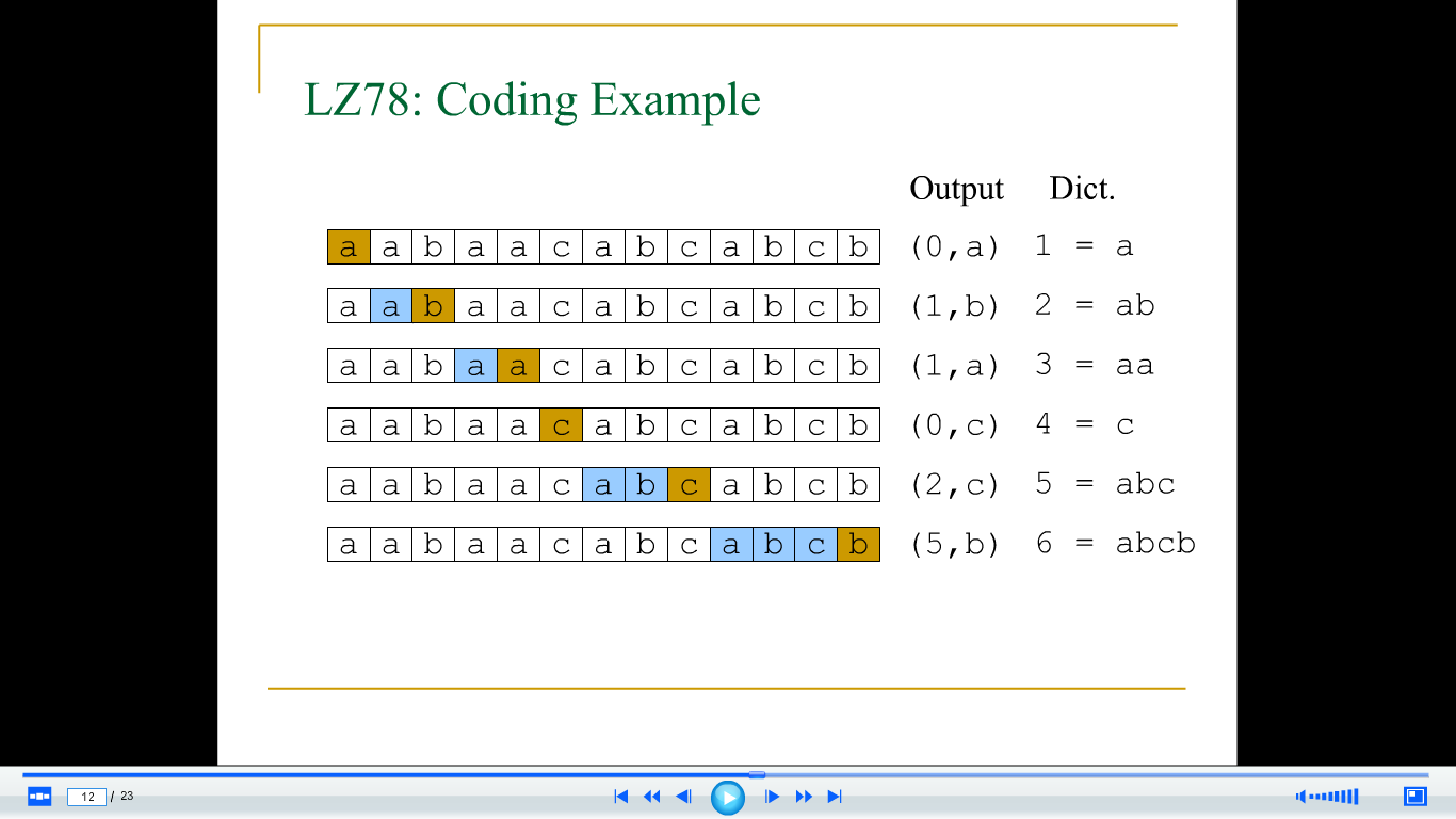
- The LZ77 implicitly assumes that the like pattern will occur closely.
- Sliding Window LZ is Asymptotically Optimal [Wyner-Ziv,94]
- Will compress long enough strings to the source entropy as the window size goes to infinity.



LZ78: Dictionary Lempel-Ziv

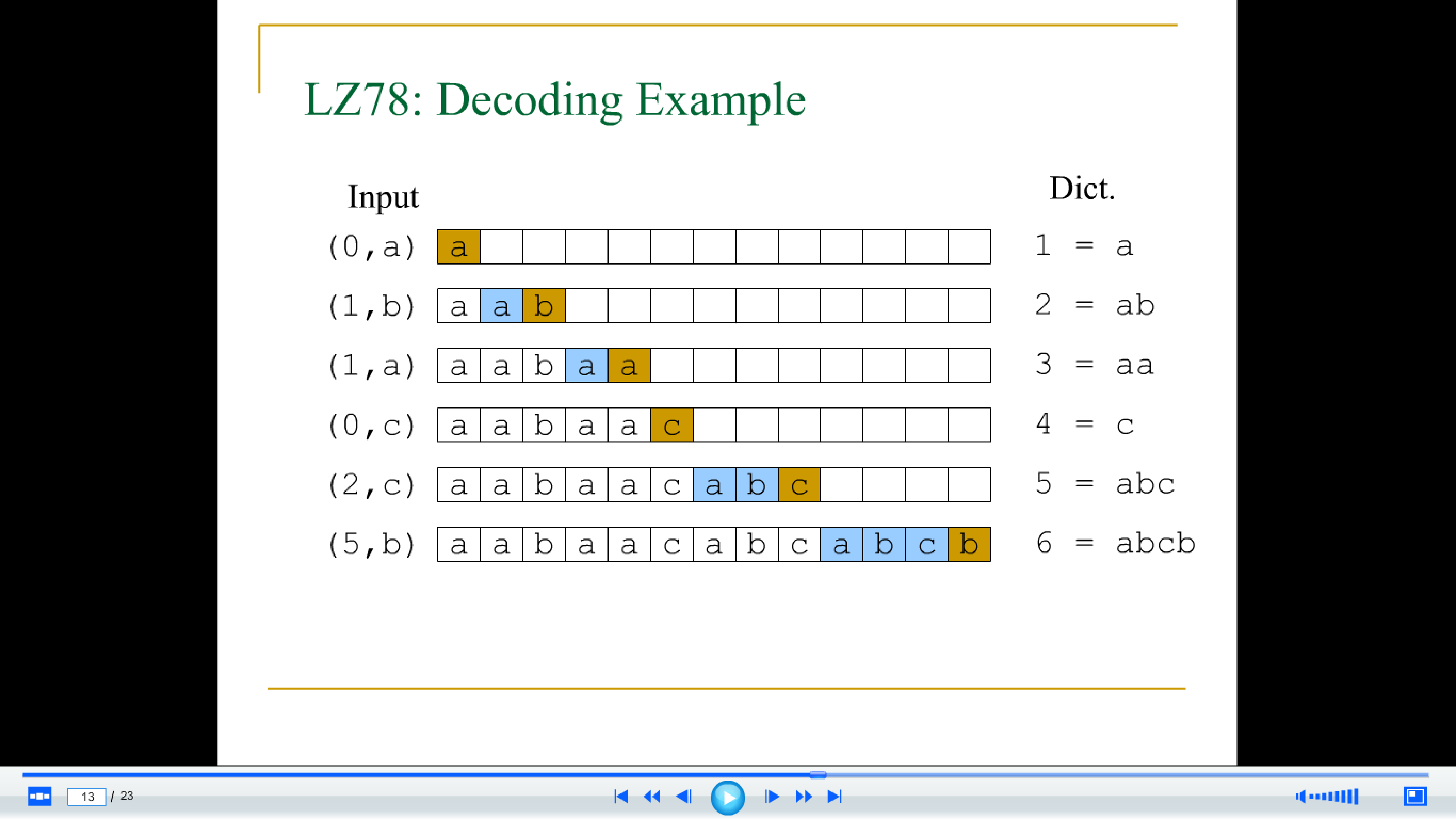
Basic algorithm:

- Keep dictionary of words with integer *id* for each entry (e.g. keep it as a trie).
- Coding loop
  - find the longest match *S* in the dictionary
  - Output the entry *id* of the match and the next character past the match from the input *(id,c)*
  - Add the string *Sc* to the dictionary
- Decoding keeps same dictionary and looks up *id*s



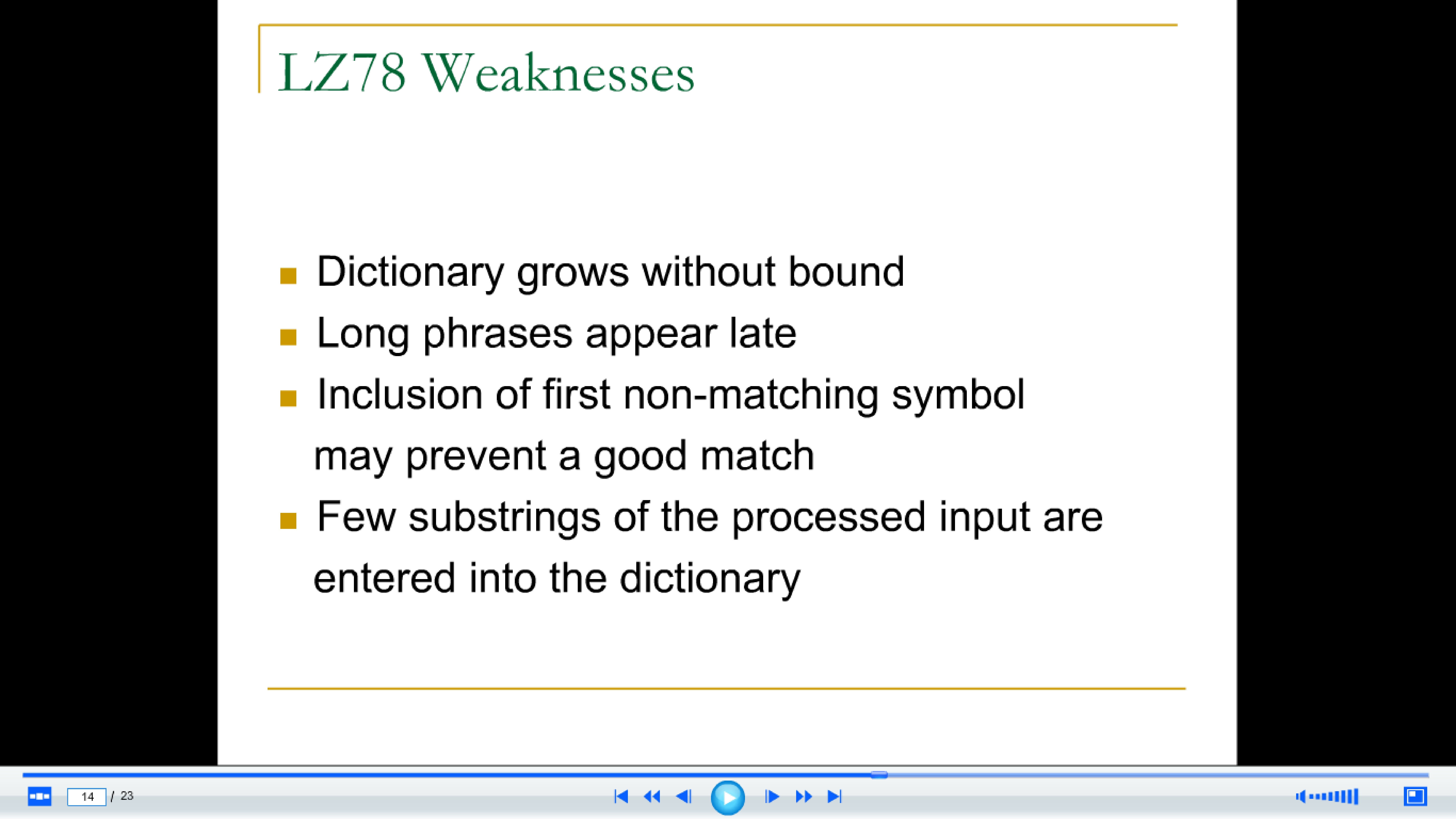
LZ78: Coding Example

													Output	Dict.
a	a	b	a	a	c	a	b	c	a	b	c	b	(0,a)	1 = a
a	a	b	a	a	c	a	b	c	a	b	c	b	(1,b)	2 = ab
a	a	b	a	a	c	a	b	c	a	b	c	b	(1,a)	3 = aa
a	a	b	a	a	c	a	b	c	a	b	c	b	(0,c)	4 = c
a	a	b	a	a	c	a	b	c	a	b	c	b	(2,c)	5 = abc
a	a	b	a	a	c	a	b	c	a	b	c	b	(5,b)	6 = abcb



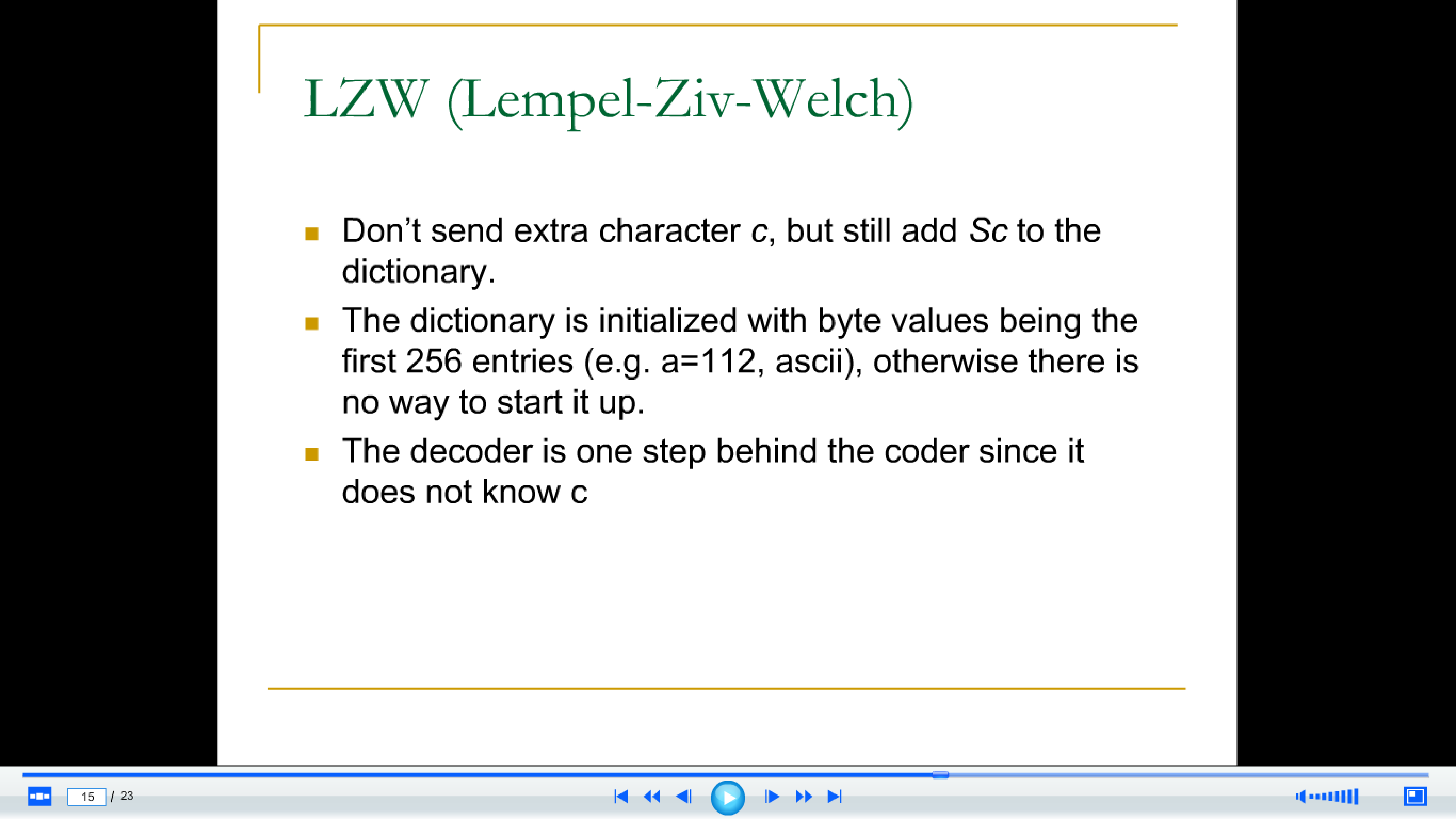
LZ78: Decoding Example

Input														Dict.
(0,a)	a													1 = a
(1,b)	a	a	b											2 = ab
(1,a)	a	a	b	a	a									3 = aa
(0,c)	a	a	b	a	a	c								4 = c
(2,c)	a	a	b	a	a	c	a	b	c					5 = abc
(5,b)	a	a	b	a	a	c	a	b	c	a	b	c	b	6 = abcb



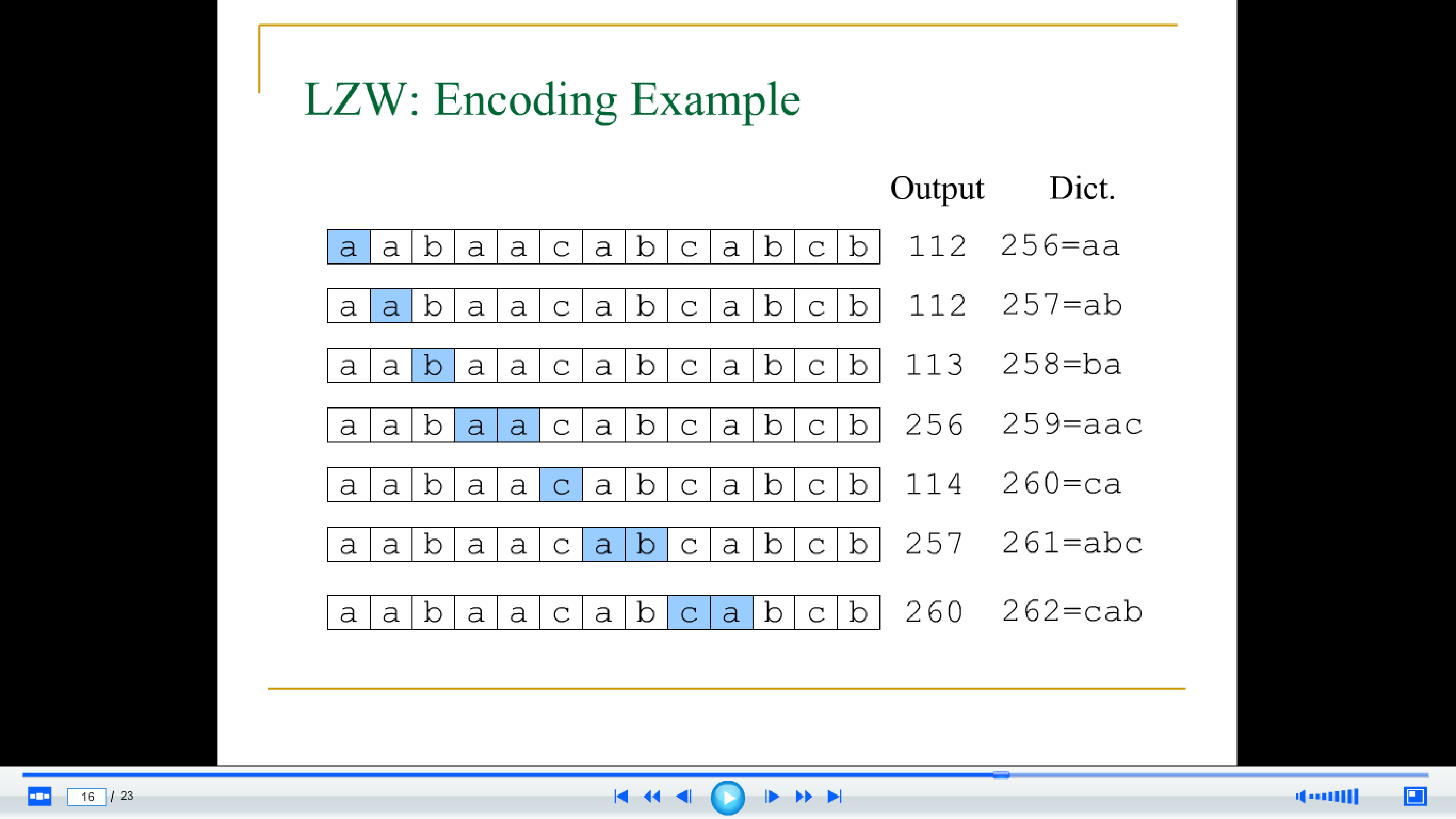
LZ78 Weaknesses

- Dictionary grows without bound
- Long phrases appear late
- Inclusion of first non-matching symbol may prevent a good match
- Few substrings of the processed input are entered into the dictionary



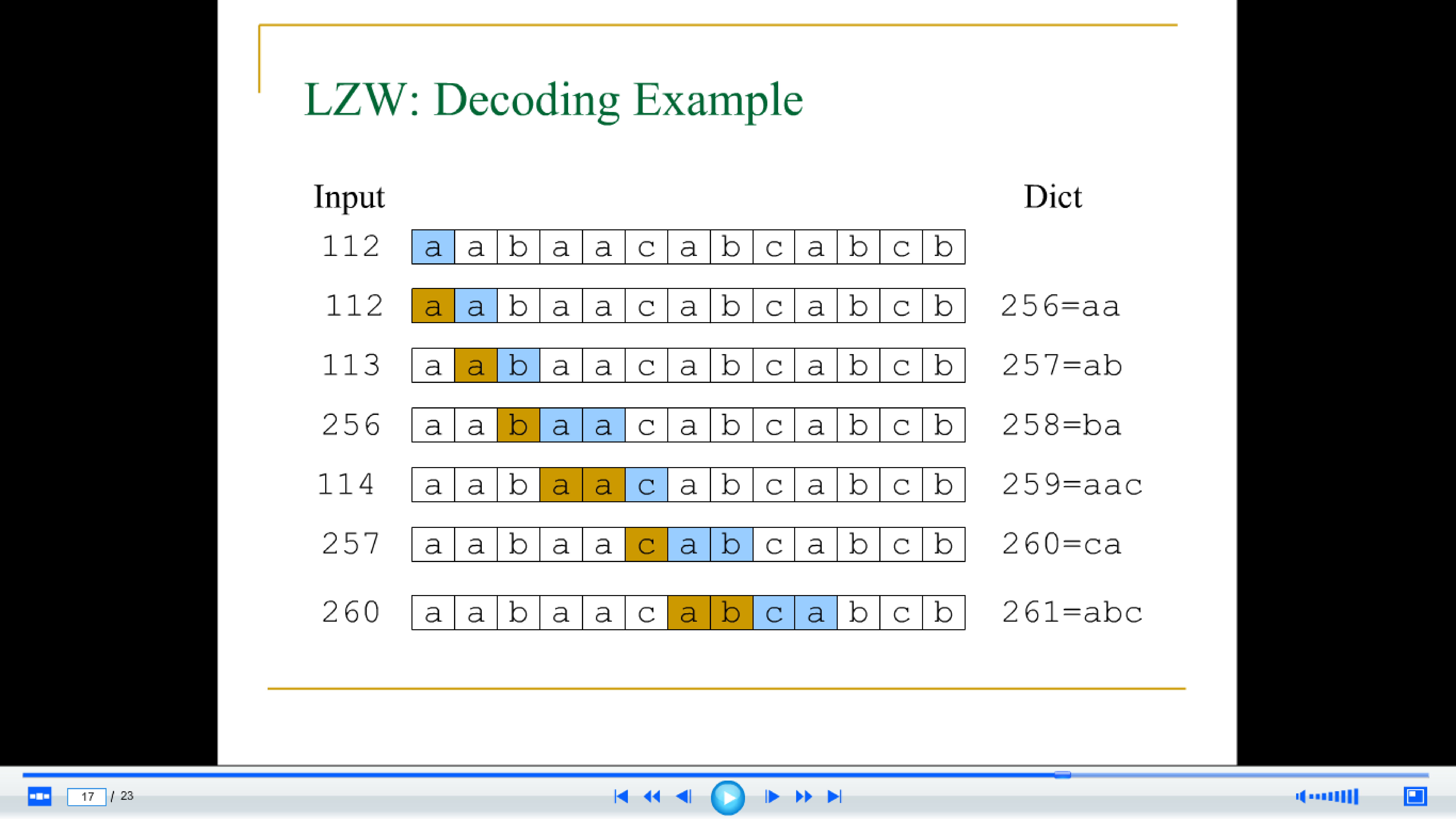
LZW (Lempel-Ziv-Welch)

- Don’t send extra character *c*, but still add *Sc* to the dictionary.
- The dictionary is initialized with byte values being the first 256 entries (e.g. a=112, ascii), otherwise there is no way to start it up.
- The decoder is one step behind the coder since it does not know c



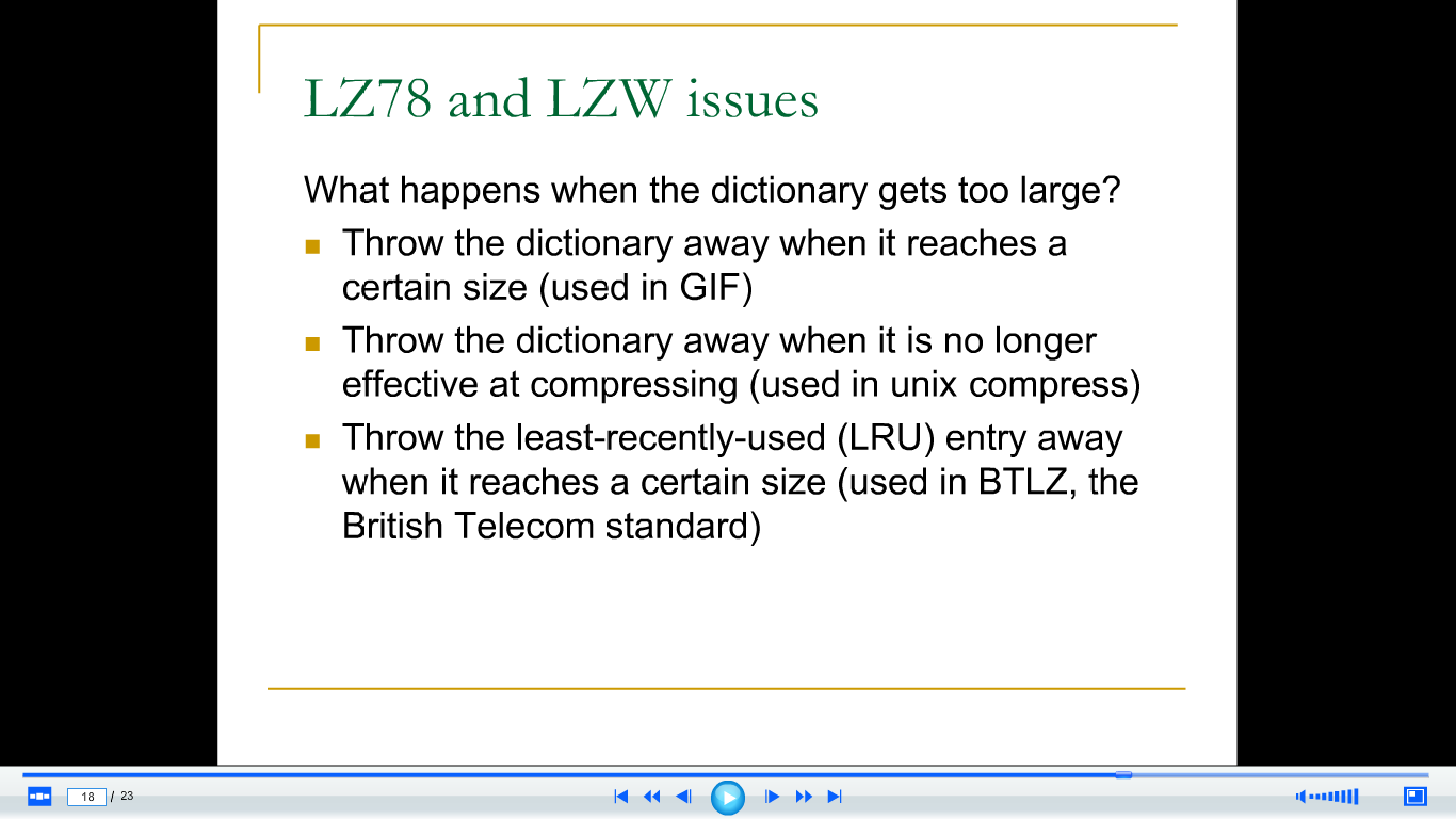
LZW: Encoding Example

													Output	Dict.
a	a	b	a	a	c	a	b	c	a	b	c	b	112	256=aa
a	**a**	b	a	a	c	a	b	c	a	b	c	b	112	257=ab
a	a	**b**	a	a	c	a	b	c	a	b	c	b	113	258=ba
a	a	b	**a**	**a**	c	a	b	c	a	b	c	b	256	259=aac
a	a	b	a	a	**c**	a	b	c	a	b	c	b	114	260=ca
a	a	b	a	a	c	**a**	**b**	c	a	b	c	b	257	261=abc
a	a	b	a	a	c	a	b	**c**	**a**	b	c	b	260	262=cab



LZW: Decoding Example

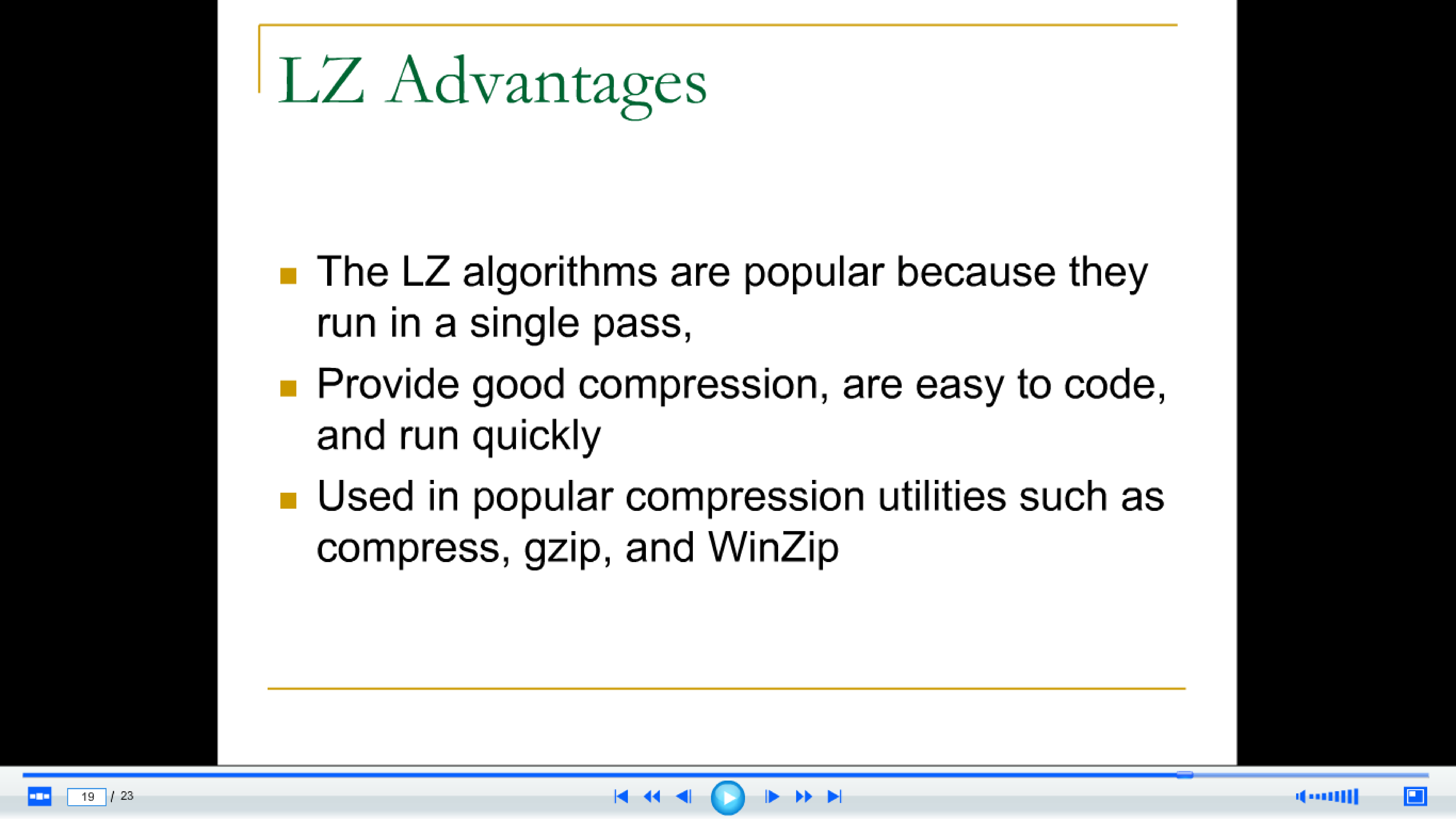
Input														Dict
112	a	a	b	a	a	c	a	b	c	a	b	c	b	
112	a	a	b	a	a	c	a	b	c	a	b	c	b	256=aa
113	a	a	b	a	a	c	a	b	c	a	b	c	b	257=ab
256	a	a	b	a	a	c	a	b	c	a	b	c	b	258=ba
114	a	a	b	a	a	c	a	b	c	a	b	c	b	259=aac
257	a	a	b	a	a	c	a	b	c	a	b	c	b	260=ca
260	a	a	b	a	a	c	a	b	c	a	b	c	b	261=abc



LZ78 and LZW issues

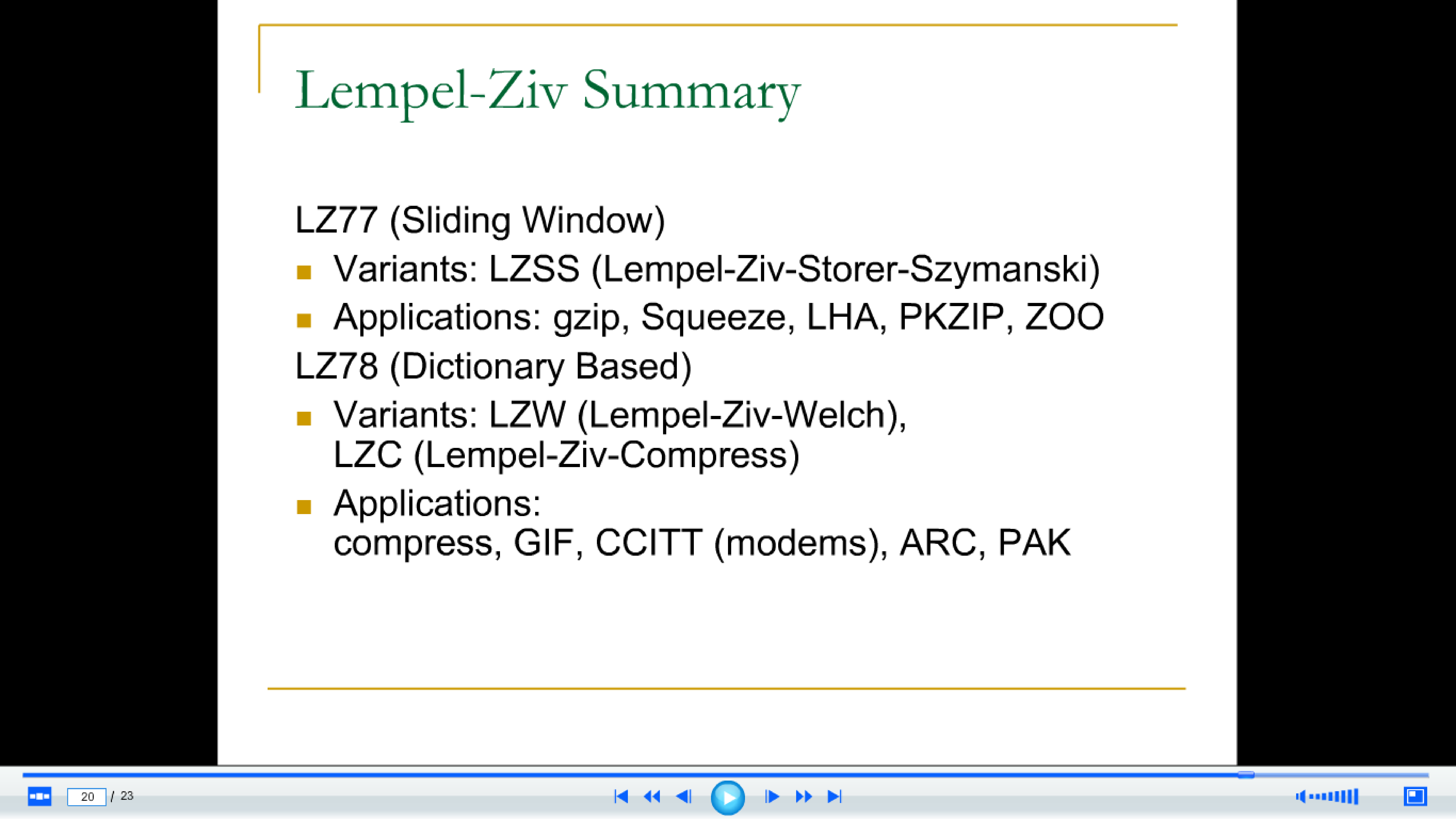
What happens when the dictionary gets too large?

- Throw the dictionary away when it reaches a certain size (used in GIF)
- Throw the dictionary away when it is no longer effective at compressing (used in unix compress)
- Throw the least-recently-used (LRU) entry away when it reaches a certain size (used in BTLZ, the British Telecom standard)



LZ Advantages

- The LZ algorithms are popular because they run in a single pass,
- Provide good compression, are easy to code, and run quickly
- Used in popular compression utilities such as compress, gzip, and WinZip



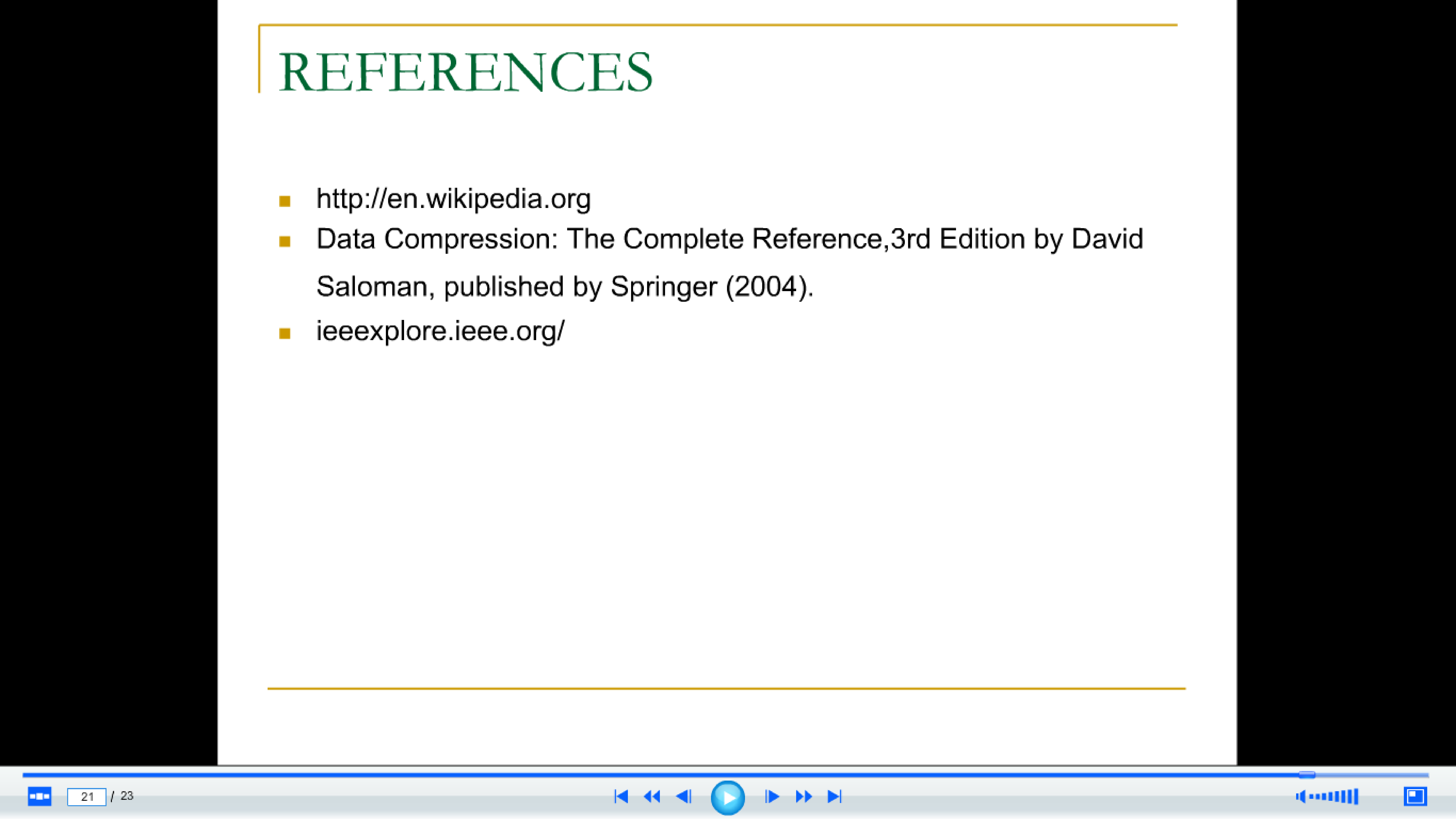
Lempel-Ziv Summary

LZ77 (Sliding Window)

- Variants: LZSS (Lempel-Ziv-Storer-Szymanski)
- Applications: gzip, Squeeze, LHA, PKZIP, ZOO

LZ78 (Dictionary Based)

- Variants: LZW (Lempel-Ziv-Welch), LZC (Lempel-Ziv-Compress)
- Applications:
  compress, GIF, CCITT (modems), ARC, PAK



REFERENCES

- http://en.wikipedia.org
- Data Compression: The Complete Reference,3rd Edition by David Saloman, published by Springer (2004).
- ieeexplore.ieee.org/

- THANK YOU