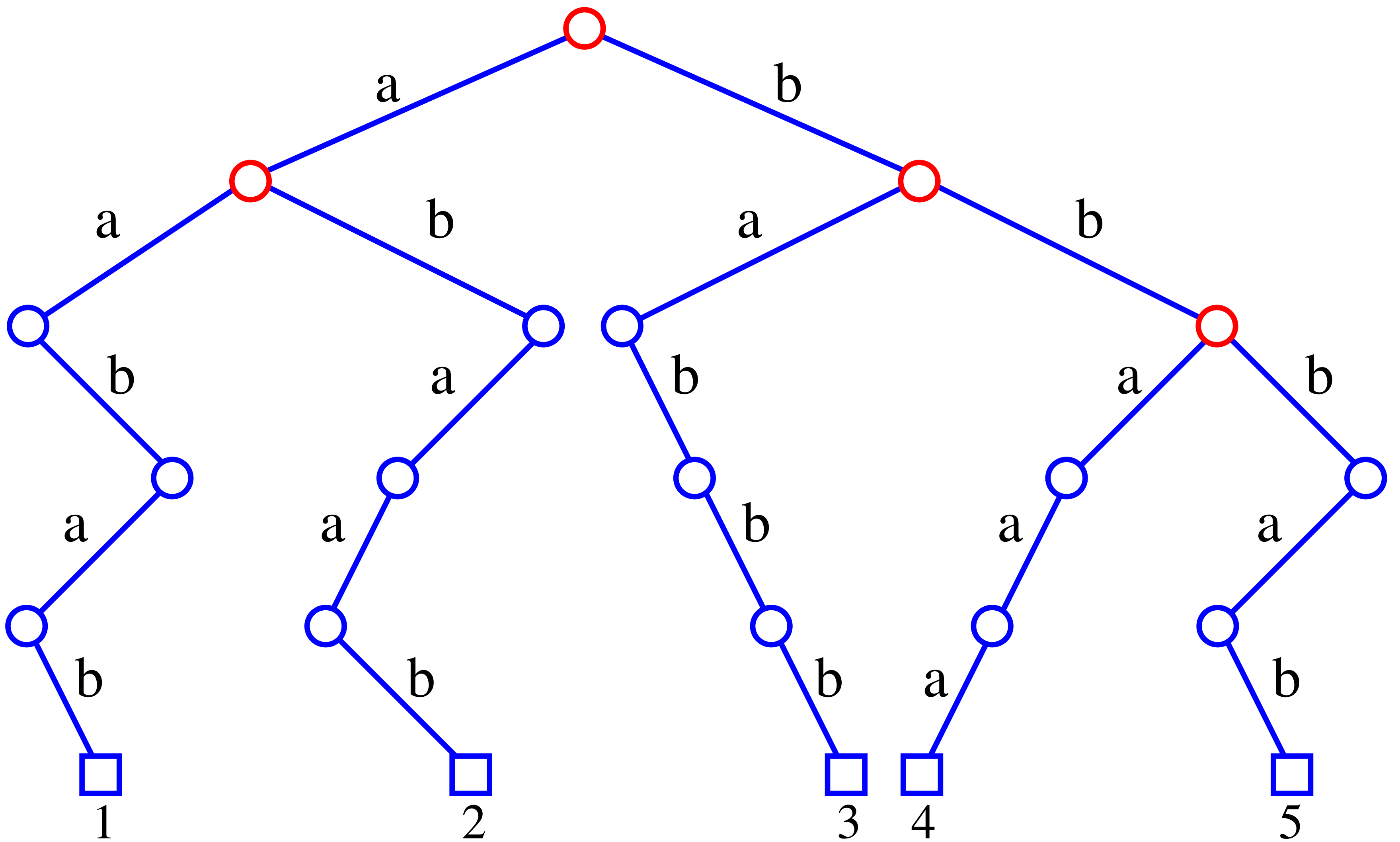
Winter 2016
COMP-250: Introduction to Computer Science

Lecture 24, April 7, 2016

Tries



Tries

- A trie is a tree-based data structure for storing strings in order to make pattern matching faster.
- Tries can be used to perform **prefix queries** for information retrieval. Prefix queries search for the longest prefix of a given string X that matches a prefix of some string in the trie.
- A trie supports the following operations on a set S of strings:

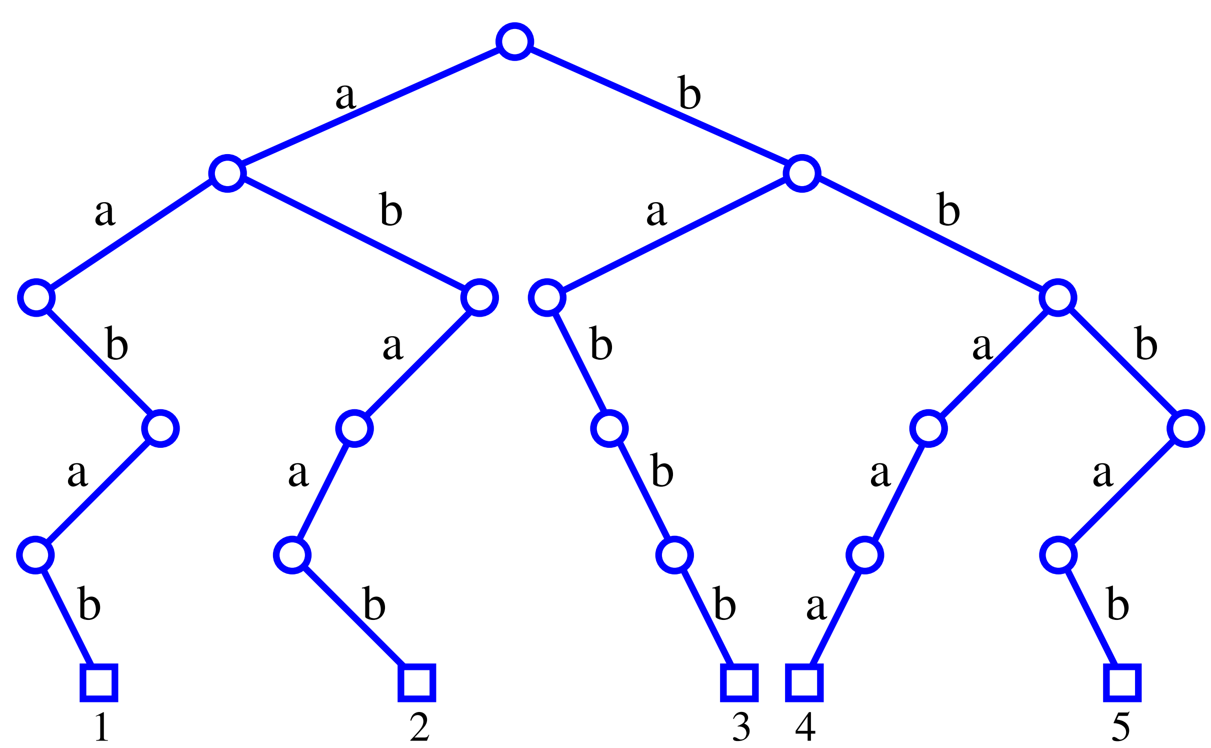
  insert(X): Insert the string X into S
  Input: String **Ouput**: None

  remove(X): Remove string X from S
  Input: String **Output**: None

  prefixes(X): Return all the strings in S that have a longest prefix of X
  Input: String **Output**: Enumeration of strings

- Let S be a set of strings from the alphabet Σ such that no string in S is a prefix to another string. A standard trie for S is an ordered tree T that:
  - Each edge of T is labeled with a character from Σ
  - The ordering of edges out of an internal node is determined by the alphabet Σ
  - The path from the root of T to any node represents a prefix in Σ that is equal to the concantenation of the characters encountered while traversing the path.

- For example, the standard trie over the alphabet $\Sigma = \{a, b\}$ for the set {aabab, abaab, babbb, bbaaa, bbbab}



Tries (cont.)

- An internal node can have 1 to *d* children when d is the size of the alphabet. Our example is essentially a binary tree.
- A path from the root of *T* to an internal node *v* at depth *i* corresponds to an *i*-character prefix of a string of *S*.
- We can implement a trie with an ordered tree by storing the character associated with an edge at the child node below it.

Compressed Tries

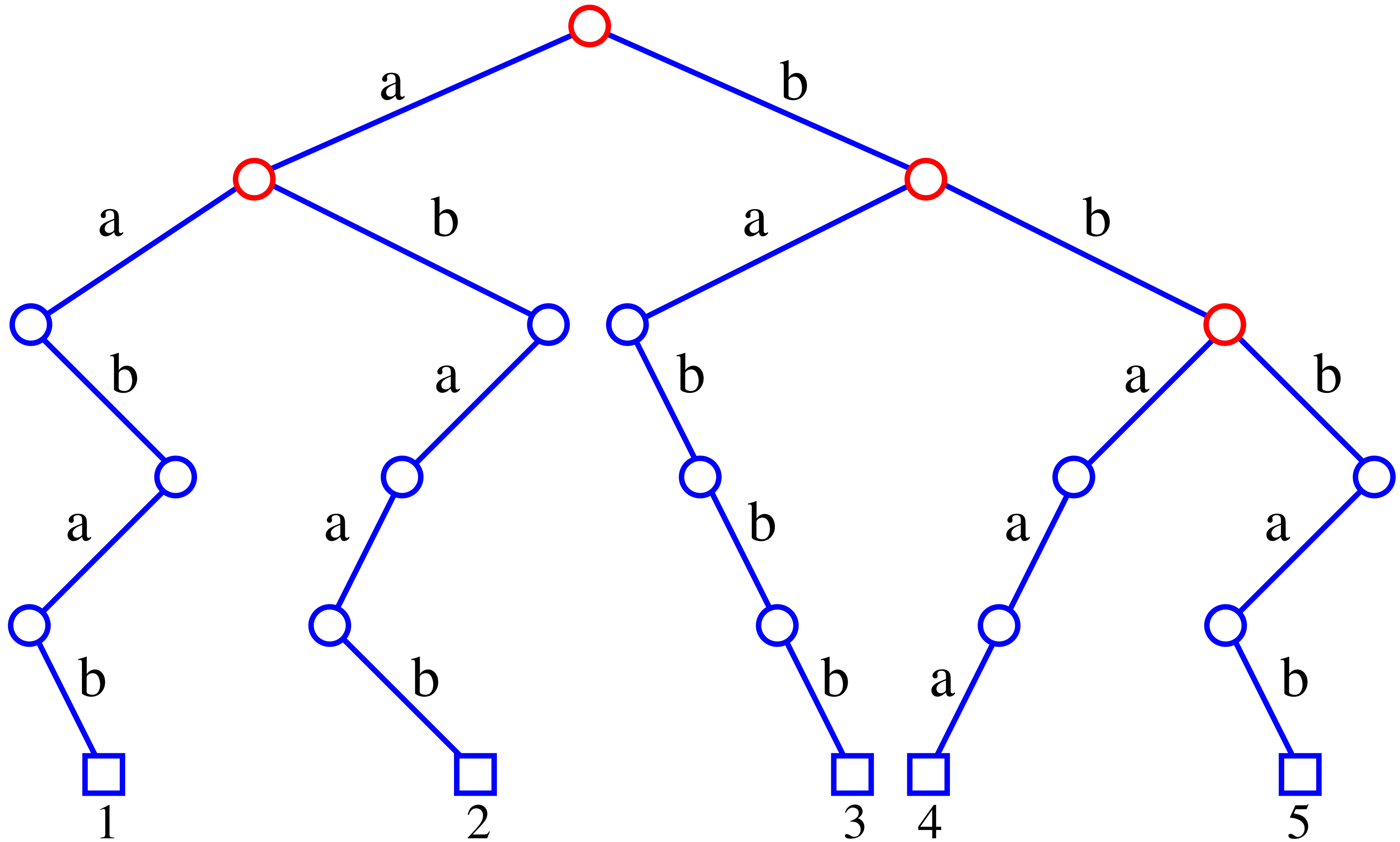
- A compressed trie is like a standard trie but makes sure that each trie had a degree of at least 2. Single child nodes are compressed into a single edge.

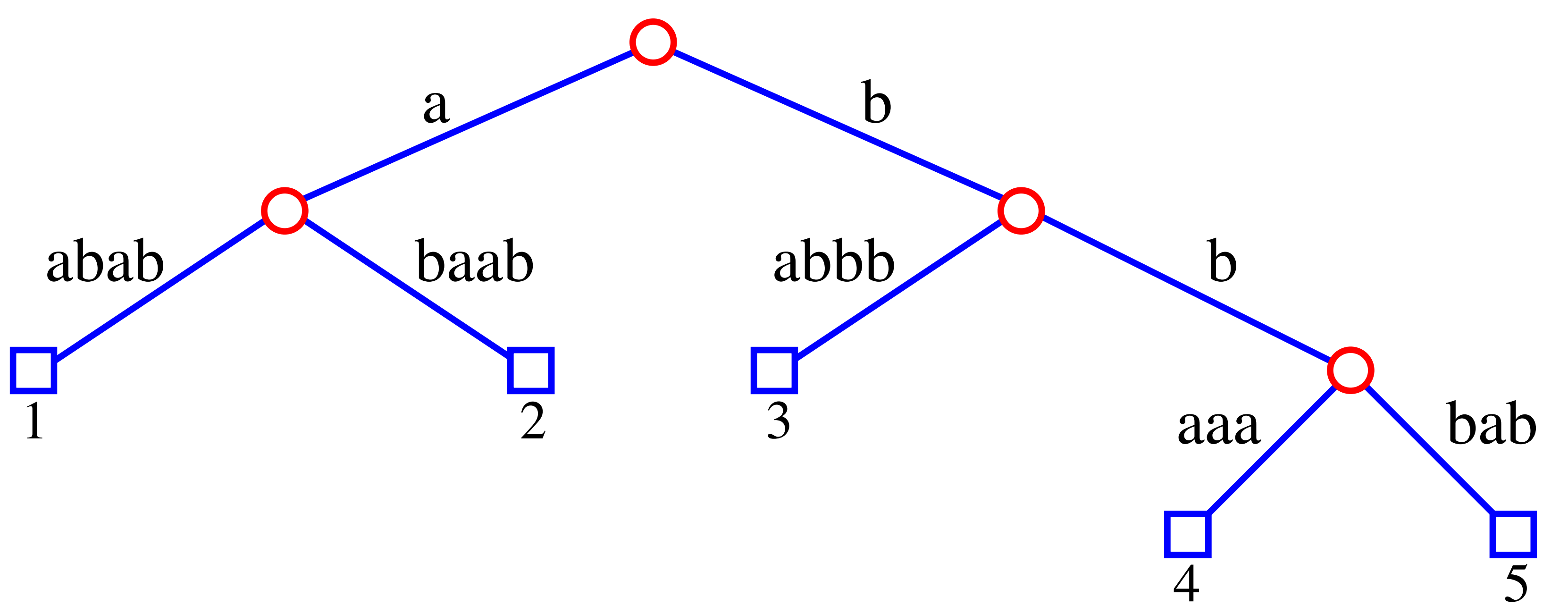
- A **critical node** is a node v such that v is labeled with a string from S, v has at least 2 children, or v is the root.

- To convert a standard trie to a compressed trie we replace an edge (v_0, v_1) by chain of nodes $(v_0, v_1 \ldots v_k)$ for $k \geq 2$ such that
  - v_0 and v_1 are critical but v_i is critical for $0<i<k$
  - each v_i has only one child

- Each internal node in a compressed trie has at least two children and each external is associated with a string. The compression reduces the total space for the trie from O(*m*) where *m* is the sum of the lengths of strings in *S* to O(*n*) where *n* is the number of strings in *S*.

- An example:


a
b
a
b
a
b
b
a
a
b
b
a
a
a
b
a
a
b
b
b
a
b
1
2
3
4
5


a
b
abab
baab
abbb
b
aaa
bab
1
2
3
4
5

Prefix Queries on a Trie

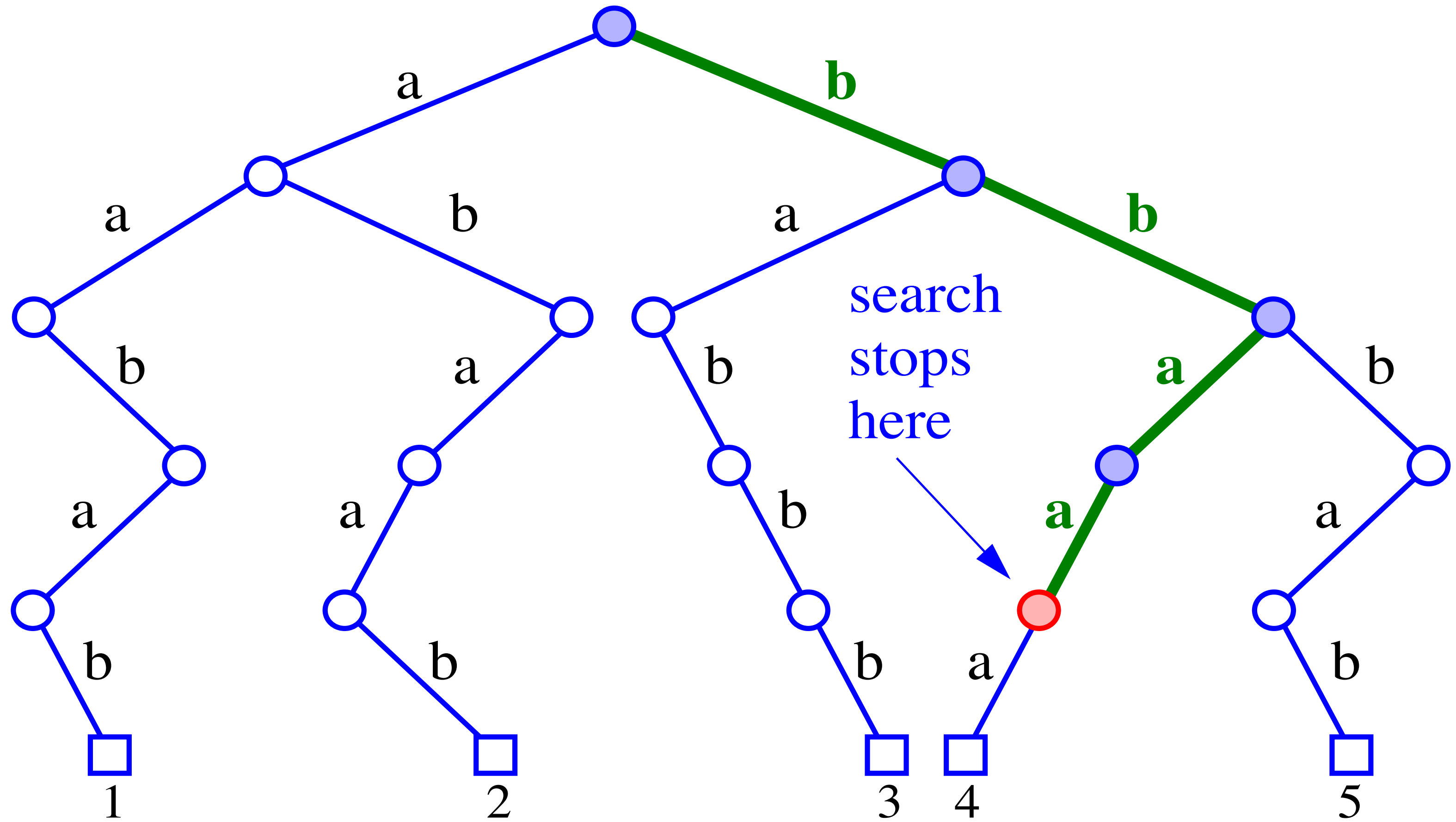
Algorithm prefixQuery(T, X):
Input: Trie T for a set S of strings and a query string X
Output: The node v of T such that the labeled nodes of the subtree of T rooted at v store the strings of S with a longest prefix in common with X

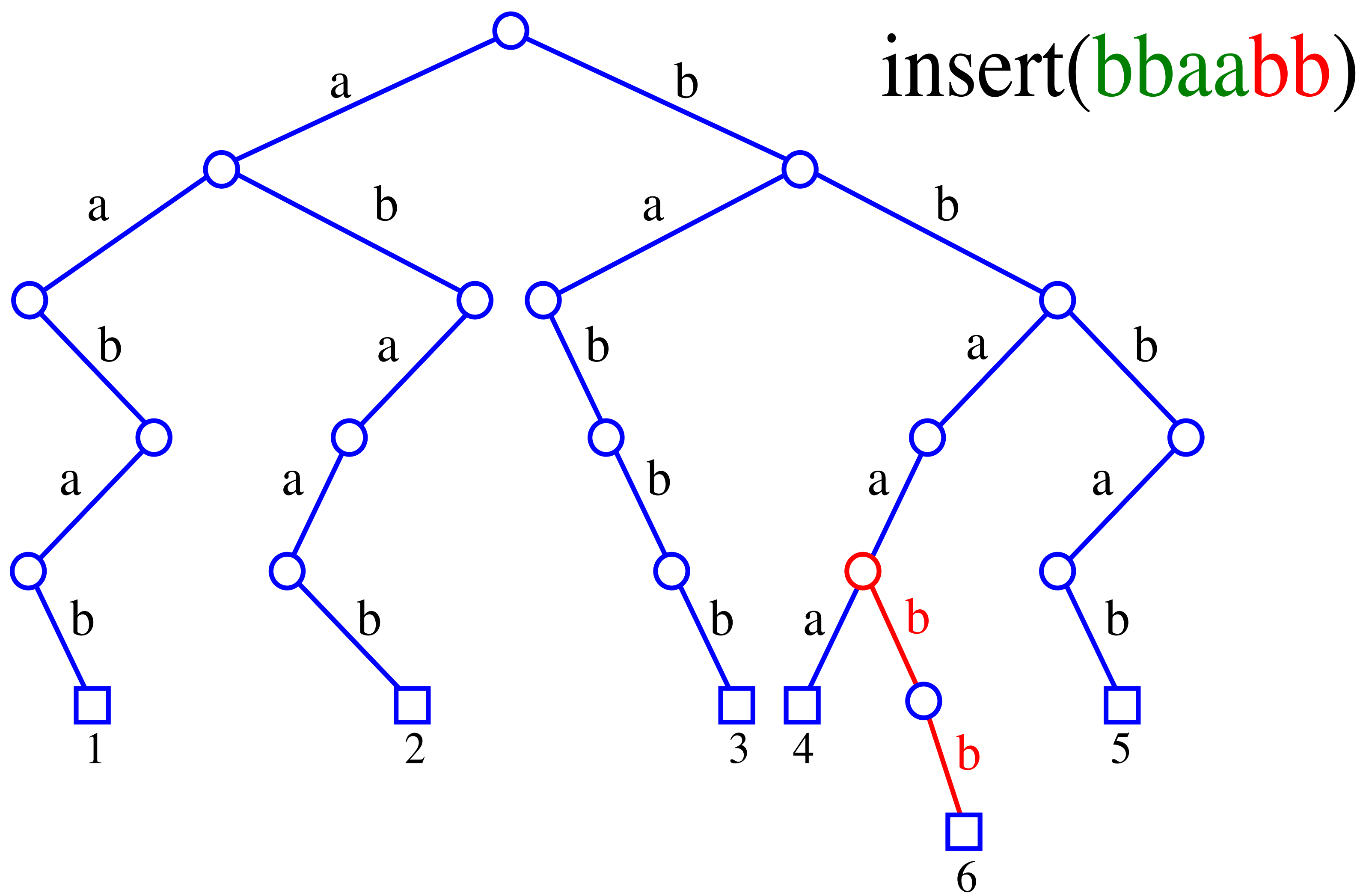
```
v←T.root()
i←0      {i is an index into the string X}
repeat
   for each child w of v do
   let e be the edge (v,w)
   Y←string(e)  {Y is the substring associated with e}
   l←Y.length()  {l=1 if T is a standard trie}
 Z←X.substring(i, i+l-1) {Z holds the next l charac
             ters of X}
   if Z = Y then
      v←w
      i←i+1{move to W, incrementing i past Z}
      break out of the for loop
   else if a proper prefix of Z matched a proper prefix
      of Y then
      v←w
      break out ot the repeat loop
until v is external or v≠w
return v
```

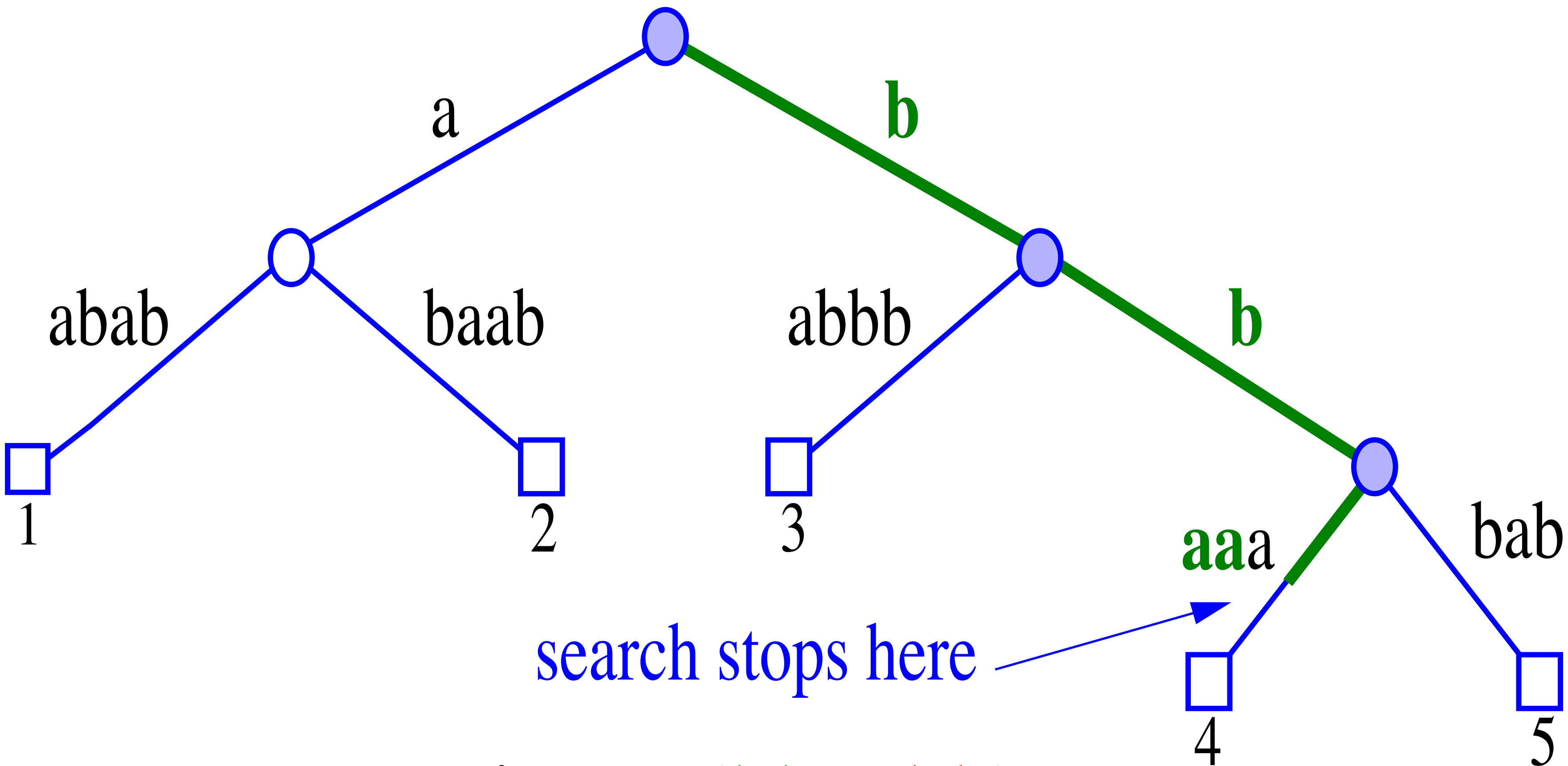
Insertion and Deletion

- Insertion: We first perform a prefix query for string X. Let us examine the ways a prefix query may end in terms of insertion.
  - The query terminates at node v. Let X_1 be the prefix of X that matched in the trie up to node v and X_2 be the rest of X. If X_2 is an empty string we label v with X and the end. Otherwise we create a new external node w and label it with X.
  - The query terminates at an edge e=(v, w) because a prefix of X match prefix(v) and a proper prefix of string Y associated with e. Let Y_1 be the part of Y that X matched to and Y_2 the rest of Y. Likewise for X_1 and X_2. Then $X=X_1+X_2$ = prefix(v) $+Y_1+X_2$. We create a new node u and split the edges(v, u) and (u, w). If X_2 is empty then we label u with X. Otherwise we create a node z which is external and label it X.
- Insertion is O(dn) when d is the size of the alphabet and n is the length of the string t insert.

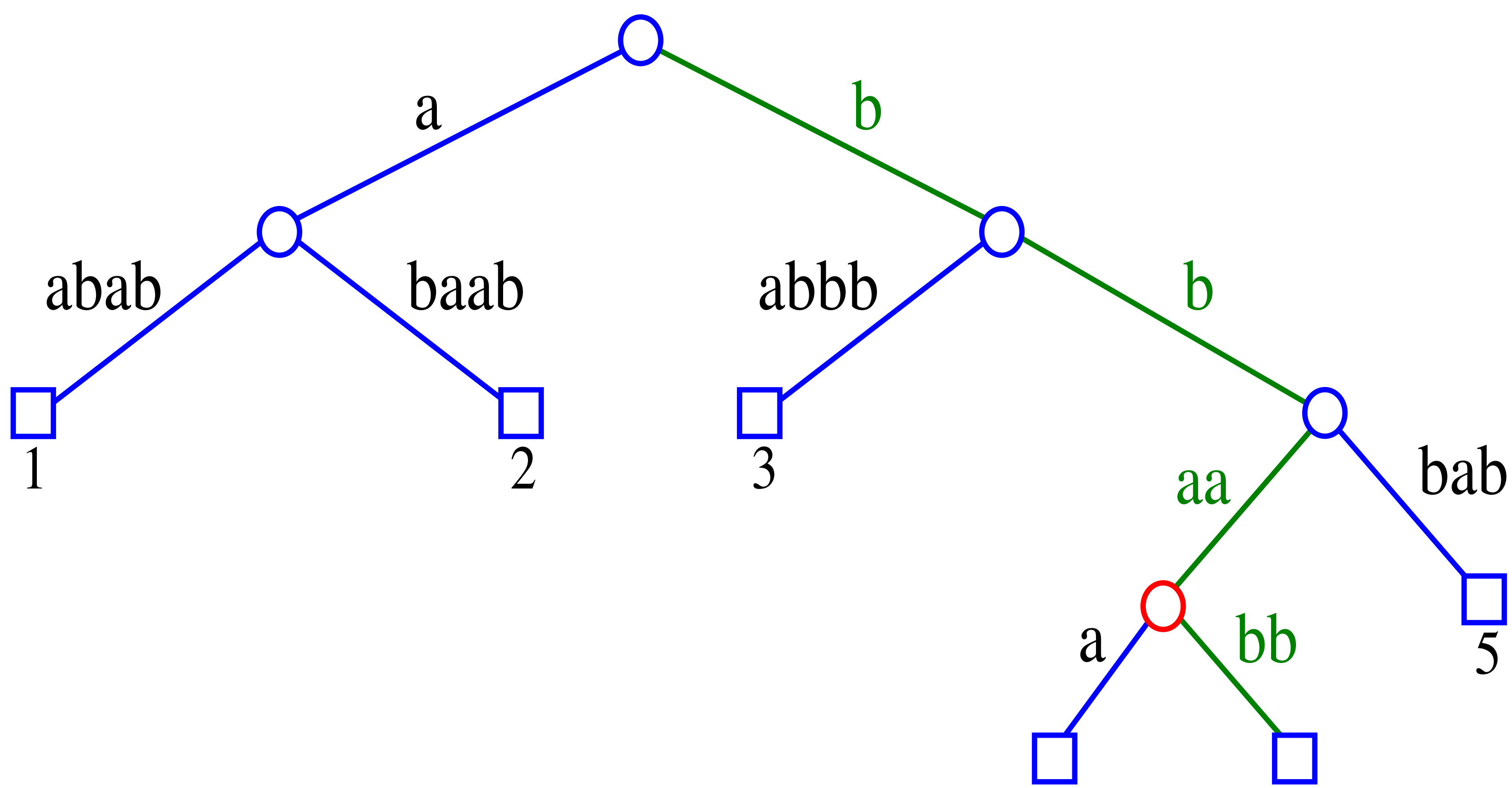
Insertion and Deletion (cont.)







insert(bbaabb)



Lempel Ziv Encoding

- Constructing the trie:
  - Let phrase 0 be the null string.
  - Scan through the text
  - If you come across a letter you haven't seen before, add it to the top level of the trie.
  - If you come across a letter you've already seen, scan down the trie until you can't match any more characters, add a node to the trie representing the new string.
  - Insert the pair (nodeIndex, lastChar) into the compressed string.
- Reconstructing the string:
  - Every time you see a '0' in the compressed string add the next character in the compressed string directly to the new string.
  - For each non-zero nodeIndex, put the substring corresponding to that node into the new string, followed by the next character in the compressed string.

Lempel Ziv Encoding (contd.)

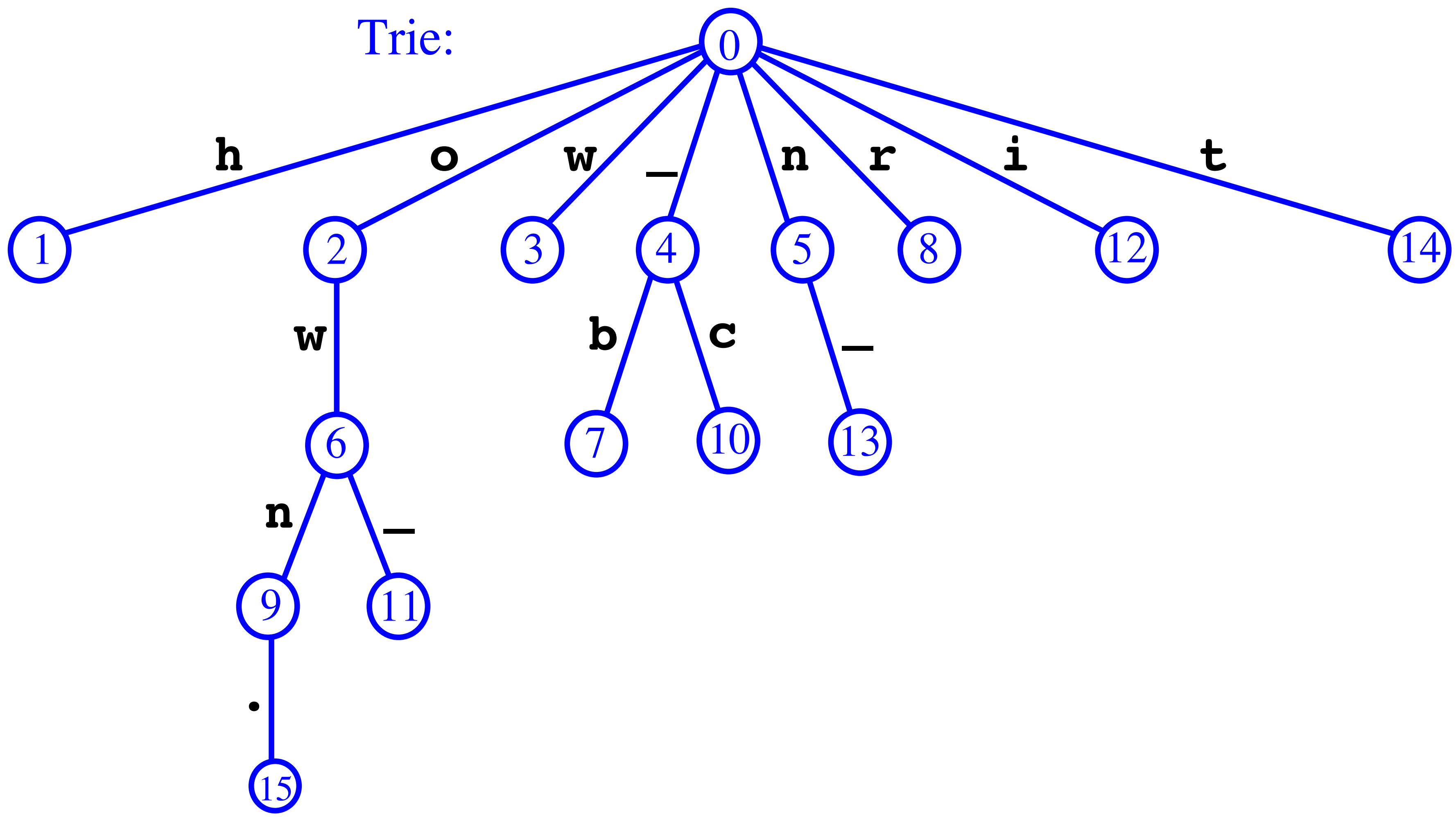
- A graphical example:

Uncompressed text: (nil) **how now brown cow in town.**

phrases: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Compressed text: 0**h**0**o**0**w**0_0**n**2**w**4**b**0**r**6**n**4**c**6_0**i**5_0**t**9**.**

Trie:

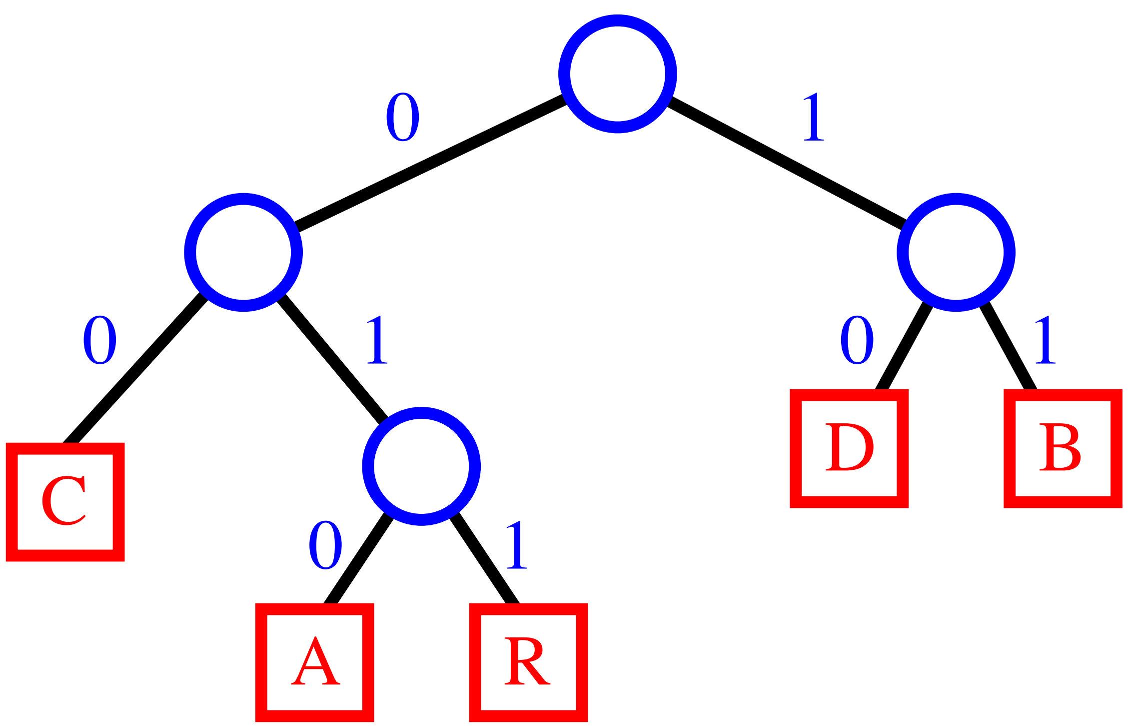


File Compression

- text files are usually stored by representing each character with an 8-bit ASCII code (type man ascii in a Unix shell to see the ASCII encoding)
- the ASCII encoding is an example of **fixed-length encoding**, where each character is represented with the same number of bits
- in order to reduce the space required to store a text file, we can exploit the fact that some characters are more likely to occur than others
- **variable-length encoding** uses binary codes of different lengths for different characters; thus, we can assign fewer bits to frequently used characters, and more bits to rarely used characters.
- Example:
  - text: java
  - encoding: a = “0”, j = “11”, v = “10”
  - encoded text: 110100 (6 bits)
- How to decode?
  - a = “0”, j = “01”, v = “00”
  - encoded text: 010000 (6 bits)
  - is this java, jvv, jaaaa ...

Encoding Trie

- to prevent ambiguities in decoding, we require that the encoding satisfies the **prefix rule**, that is, no code is a prefix of another code
  - a = “0”, j = “11”, v = “10” satisfies the prefix rule
  - a = “0”, j = “01”, v= “00” does **not** satisfy the prefix rule (the code of a is a prefix of the codes of j and v)
- we use an **encoding trie** to define an encoding that satisfies the prefix rule
  - the characters stored at the external nodes
  - a left edge means 0
  - a right edge means 1



A = 010

B = 11

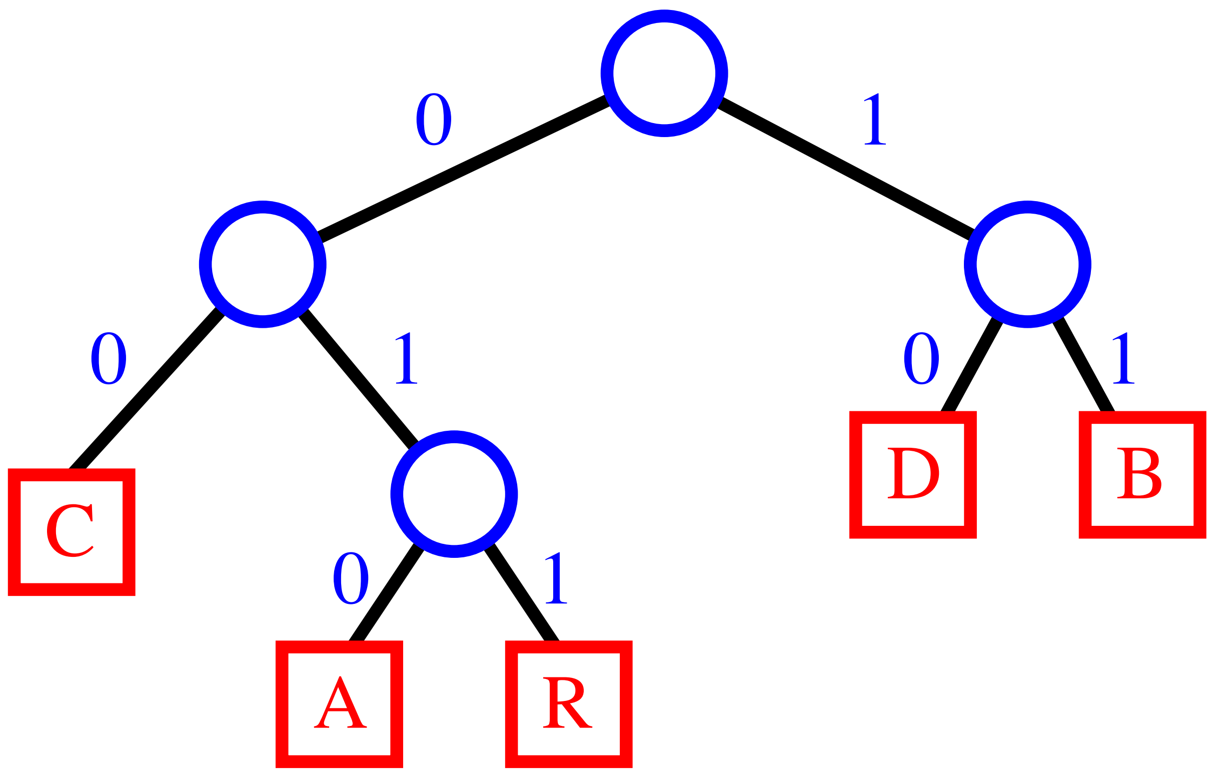
C = 00

D = 10

R = 011

Example of Decoding

- trie:



A = 010

B = 11

C = 00

D = 10

R = 011

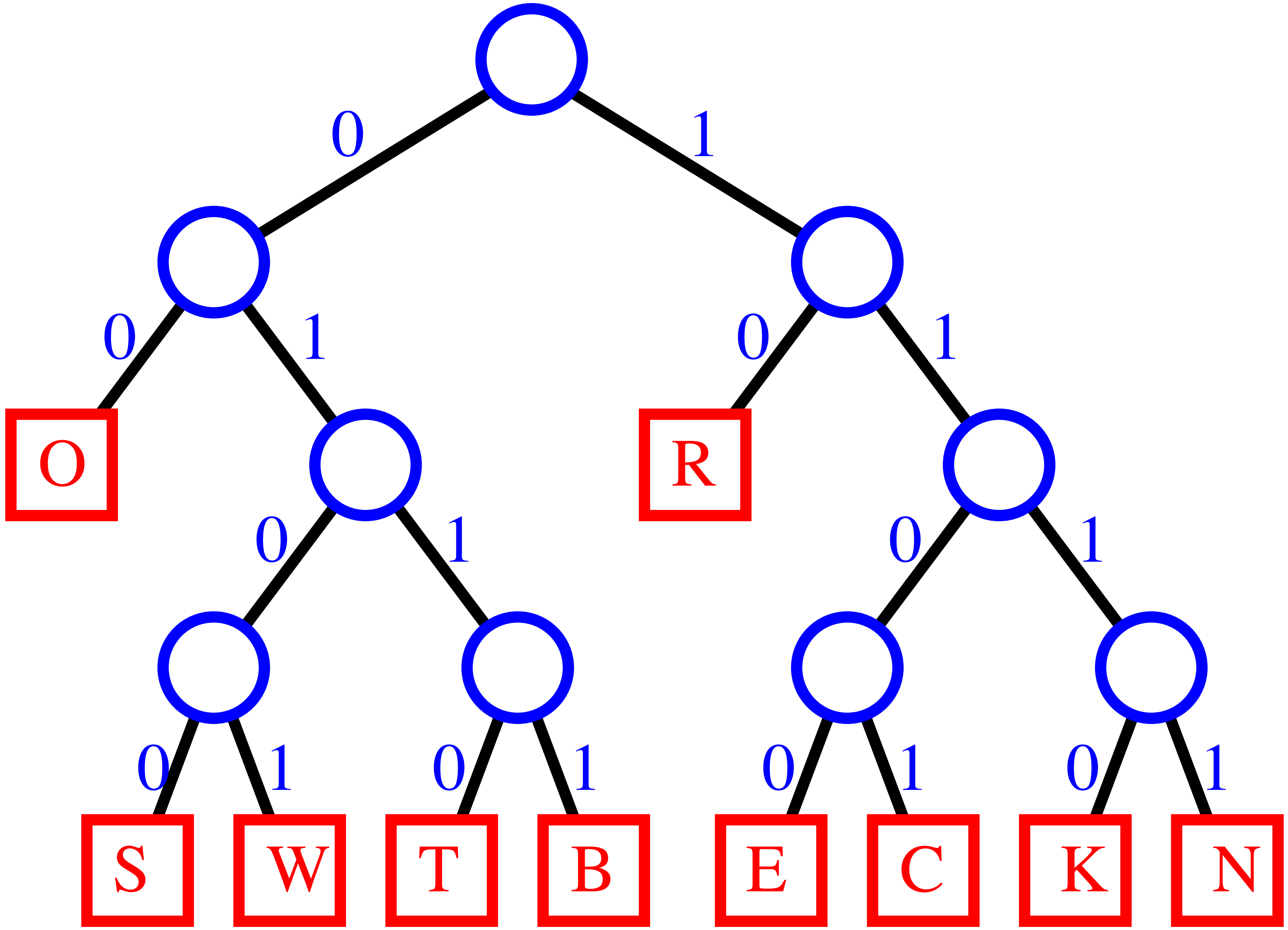
- encoded text:
  01011011010000101001011011010

- text:

A B R A C A D A B R A

010 11 011 010 00 010 10 010 11 011 010

Trie this!



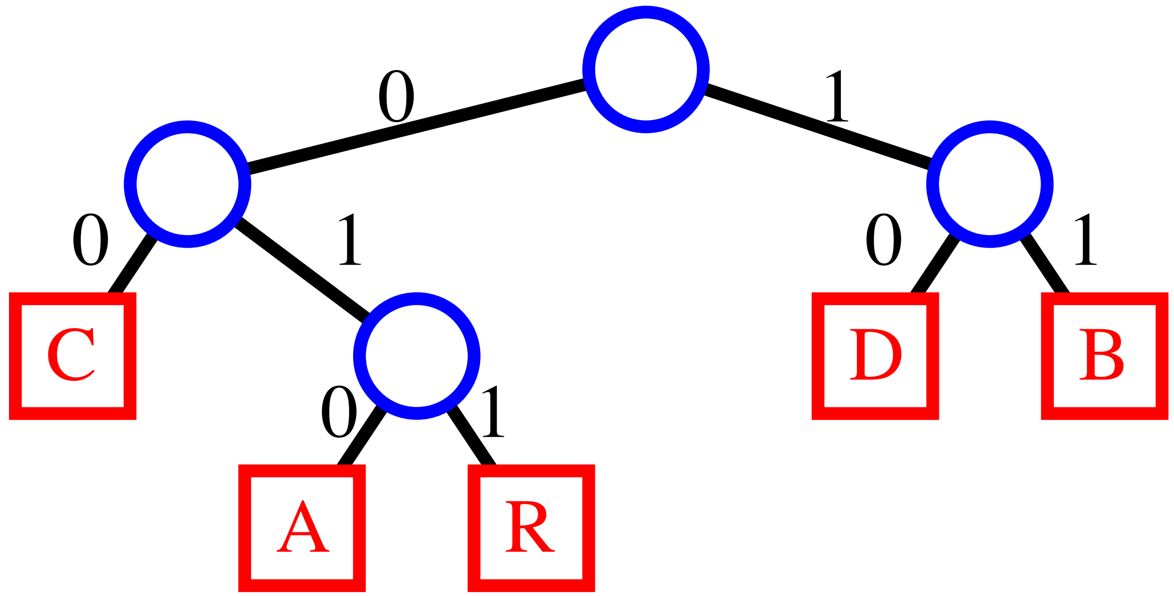
R O B E R T O K N O W S C S

10 00 0111 1100 10 0110 00 1110 1111 00 0101 0100 1101 0100

1000011111001001100011101111000101010011010100

Optimal Compression

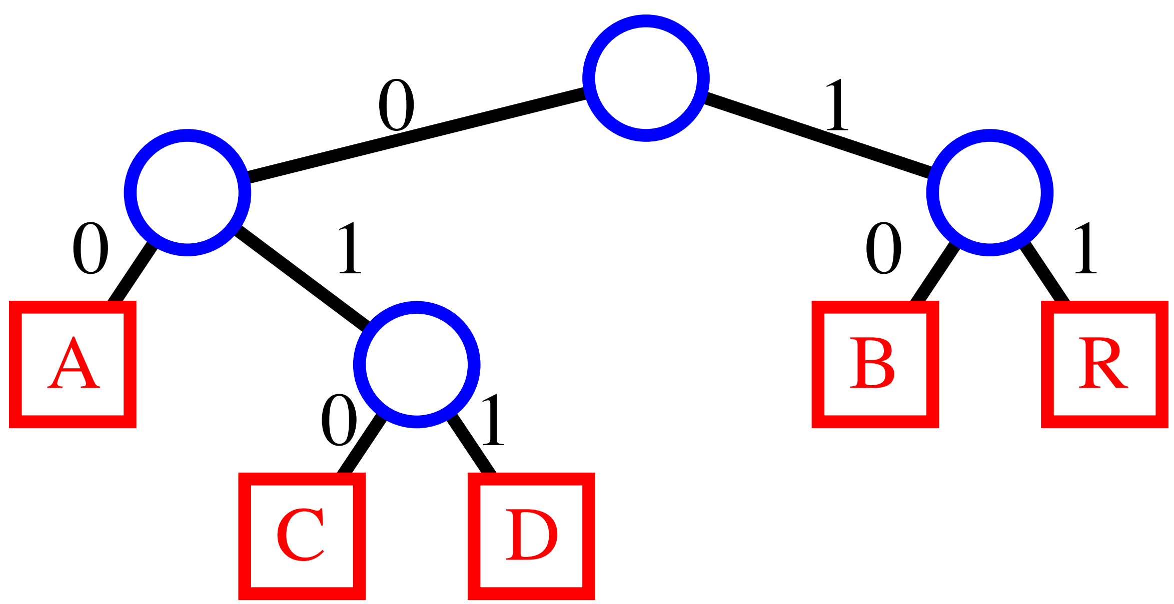
- An issue with encoding tries is to insure that the encoded text is as short as possible:



ABRACADABRA

01011011010000101001011011010

29 bits



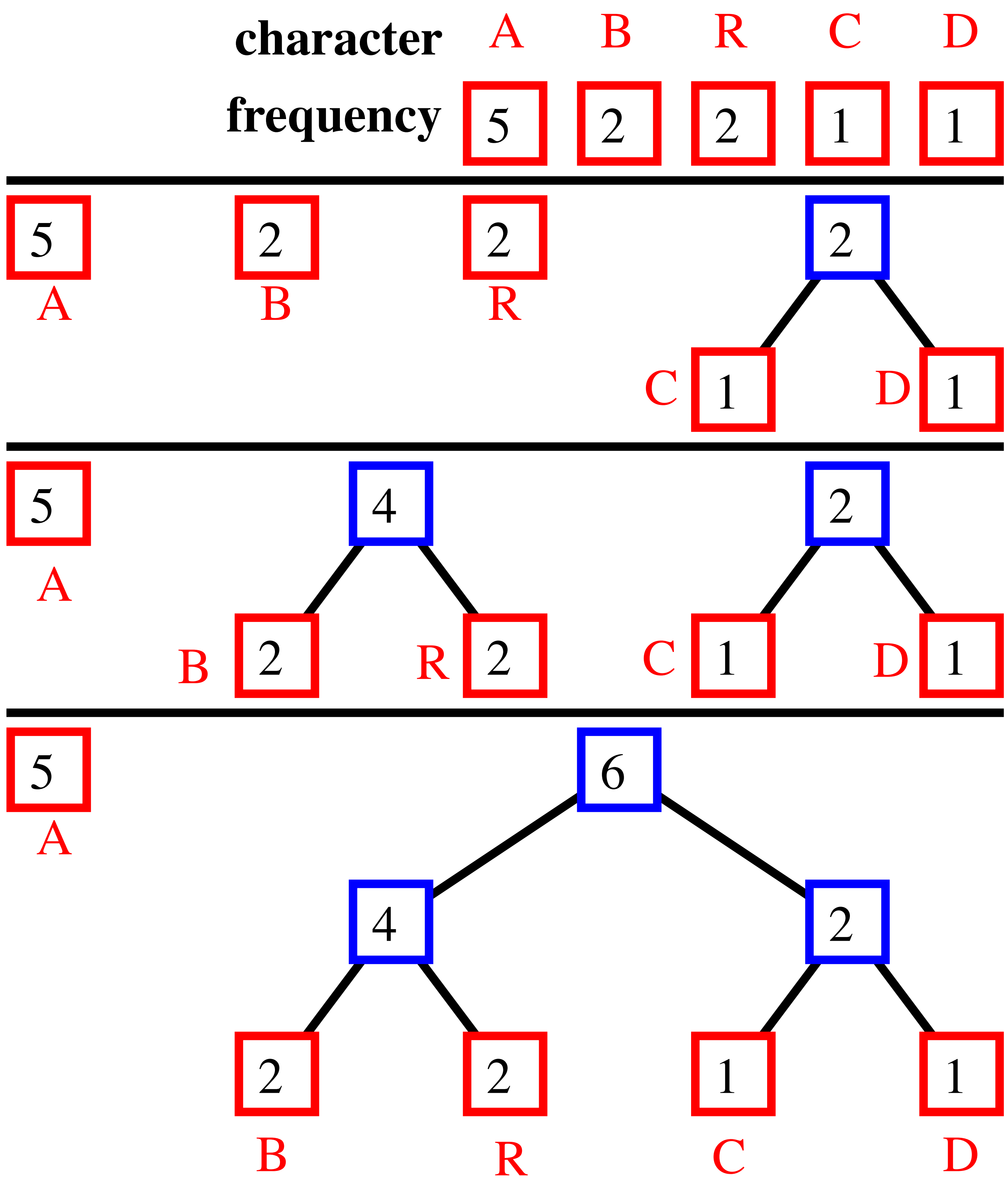
ABRACADABRA

001011000100001100101100

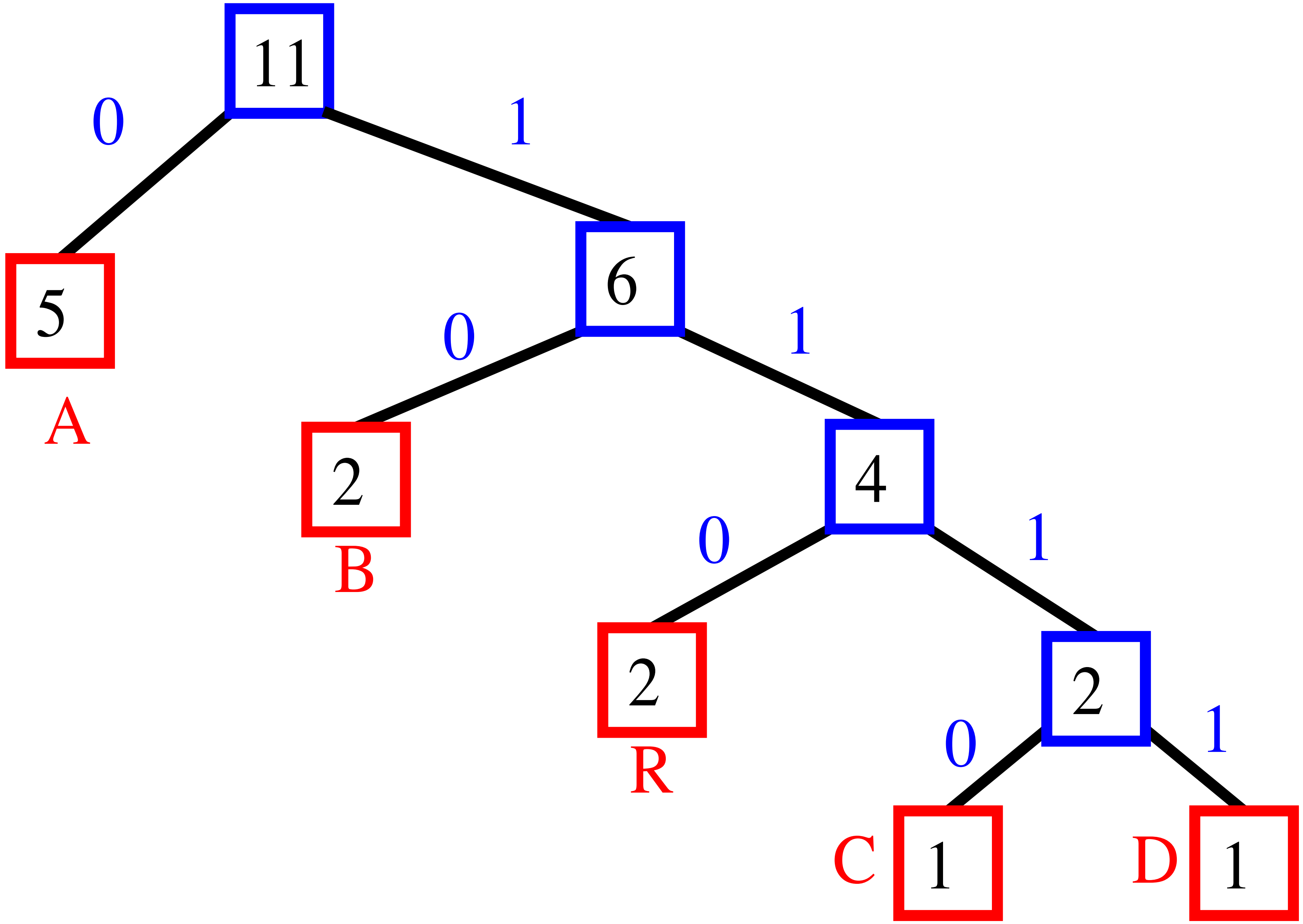
24 bits

Huffman Encoding Trie

ABRACADABRA



Another Huffman Encoding Trie



A B R A C A D A B R A

0 10 110 0 1100 0 1111 0 10 110 0

23 bits

Construction Algorithm

- with a Huffman encoding trie, the encoded text has minimal length

Algorithm Huffman(X):
Input: String X of length n
Output: Encoding trie for X

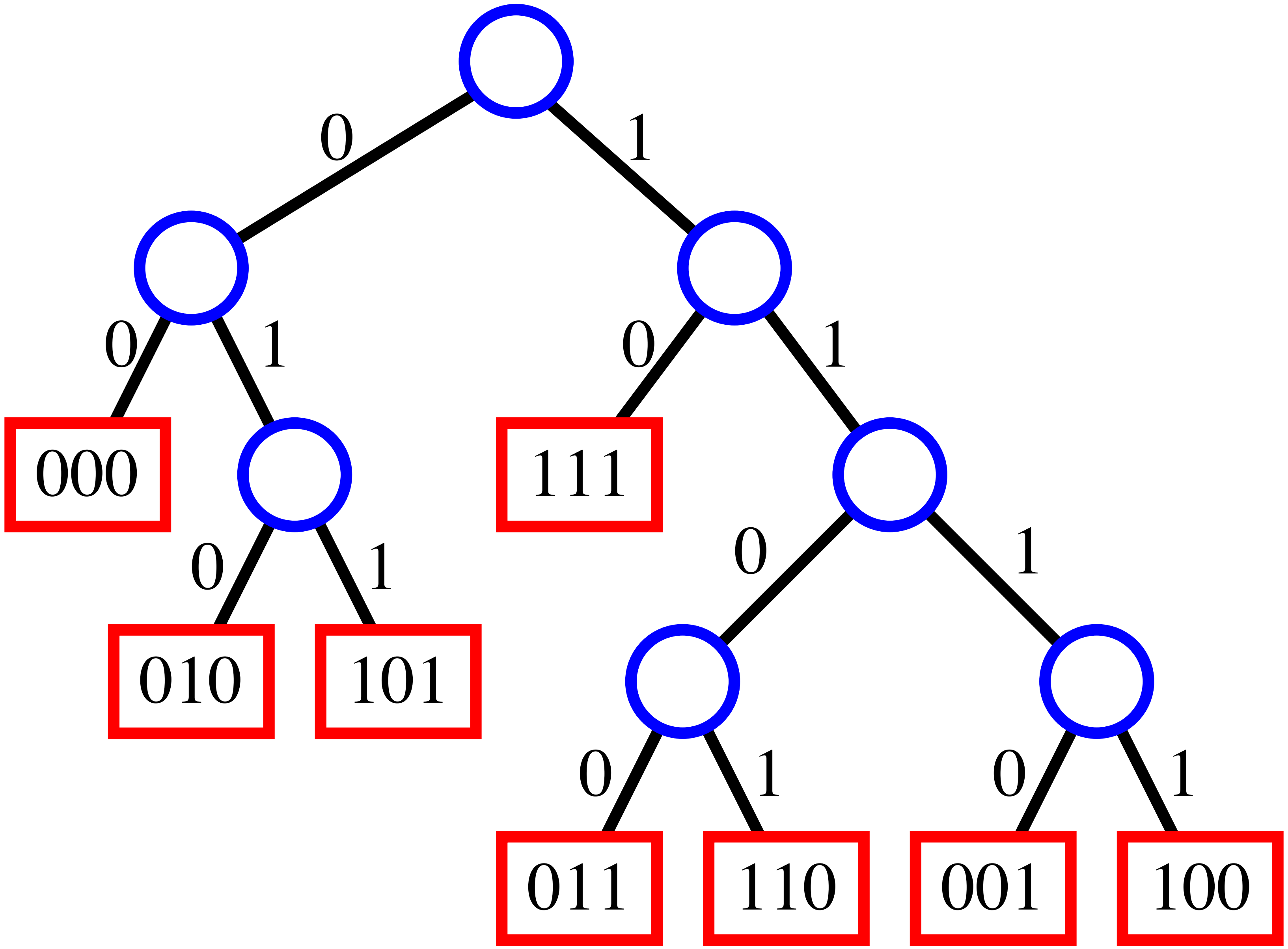
Compute the frequency $f(c)$ of each character c of X.
Initialize a priority queue Q.

for each character c in X **do**
  Create a single-node tree T storing c
  Q.insertItem($f(c)$, T)
while Q.size() > 1 **do**
  $f_1 \leftarrow Q$.minKey()
  $T_1 \leftarrow Q$.removeMinElement()
  $f_2 \leftarrow Q$.minKey()
  $T_2 \leftarrow Q$.removeMinElement()
  Create a new tree T with left subtree T_1 and right subtree T_2.
  Q.insertItem($f_1 + f_2$)
return tree Q.removeMinElement()

- runing time for a text of length n with k distinct characters: O(n + k log k)

Image Compression

- we can use Huffman encoding also for binary files (bitmaps, executables, etc.)
- common groups of bits are stored at the leaves
- Example of an encoding suitable for b/w bitmaps

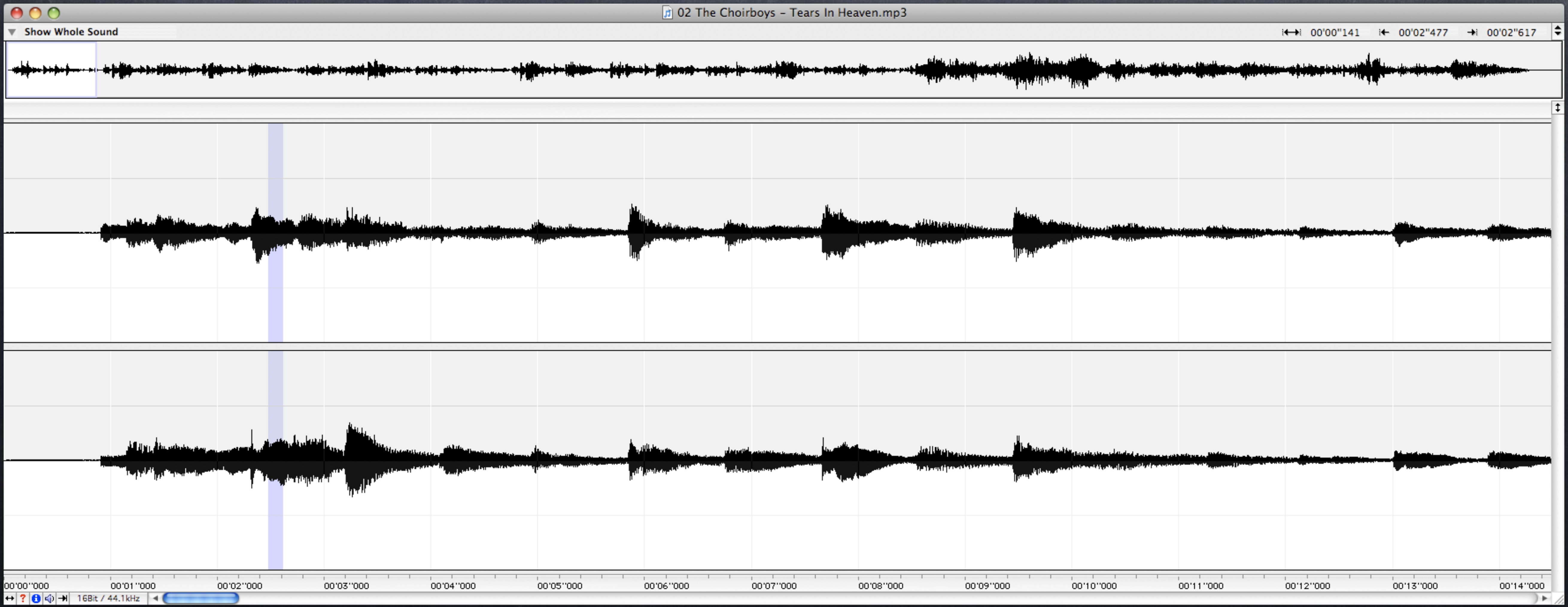


Data Representation/ Lossy Compression

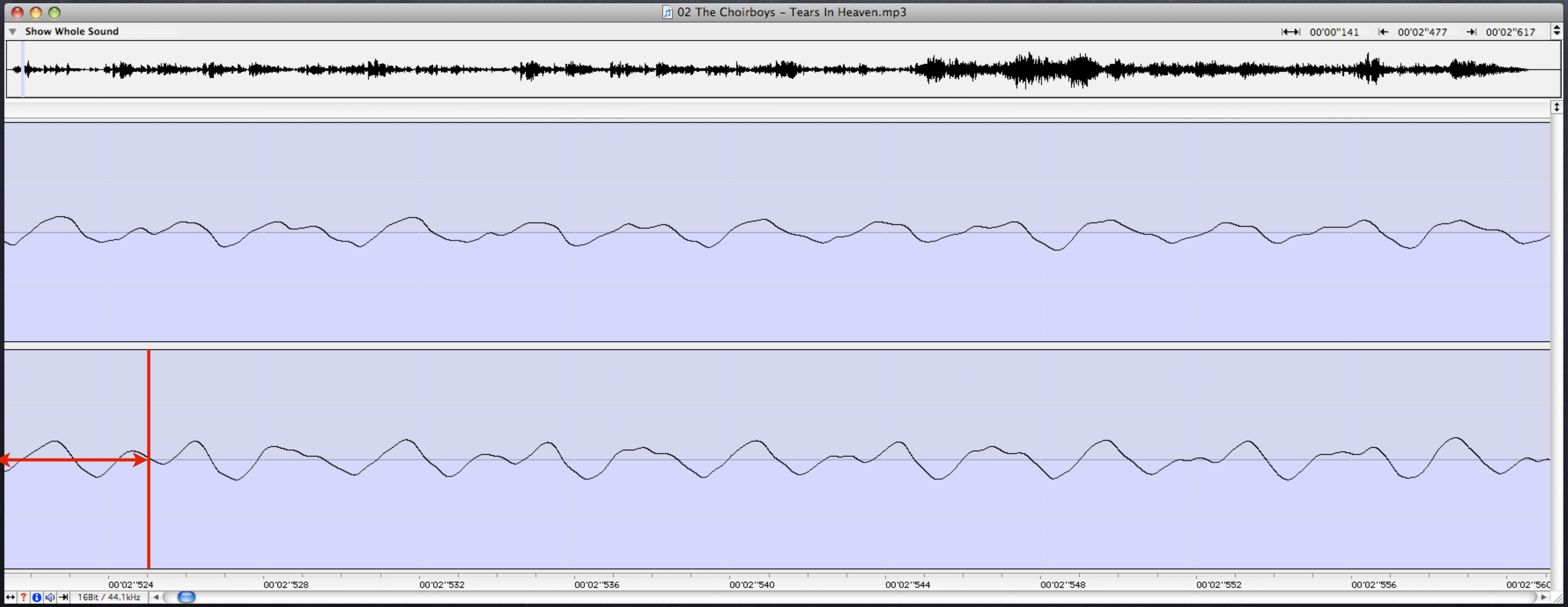
- Sound formats
- Image formats
- Movie formats

Data Representation

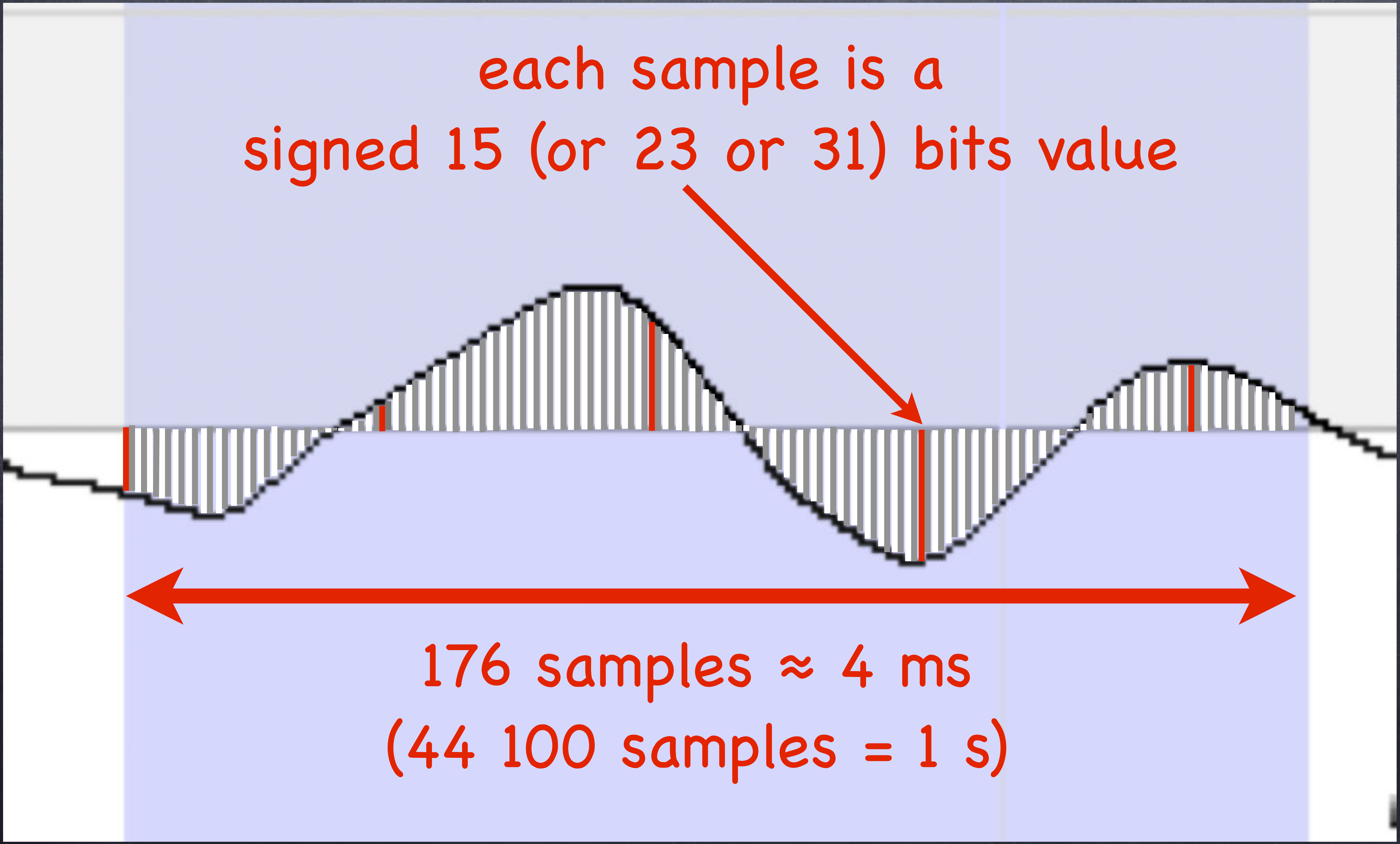
- sound formats



Sound formats



AIFF Sound format



AIFF Sound format

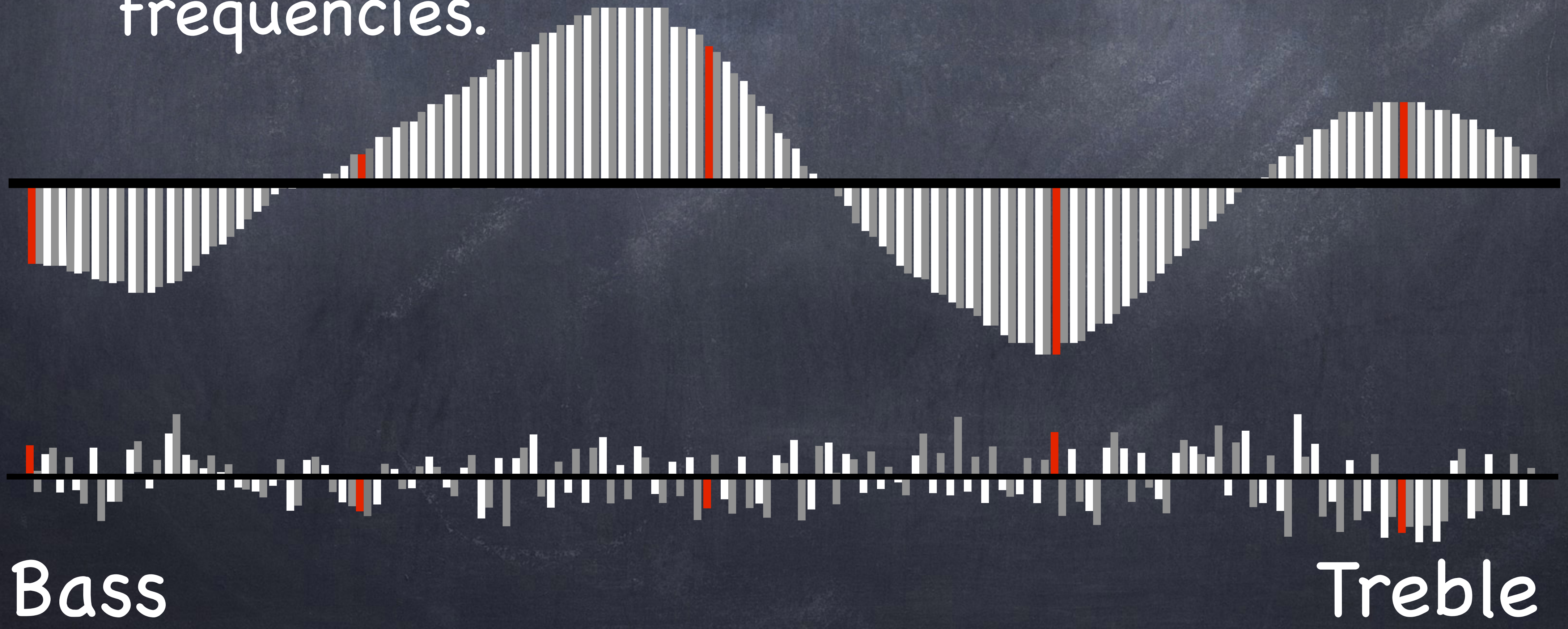
- 44 100 samples / second
- 16 b = 2 B / sample
  (or 24 b = 3 B / sample
  or 32 b = 4 B / sample)
- stereo = two channels
- 2 x 2 x 44 100 = 176,4 kB/s
- CD ≈ 700 MB ≈ 75 minutes

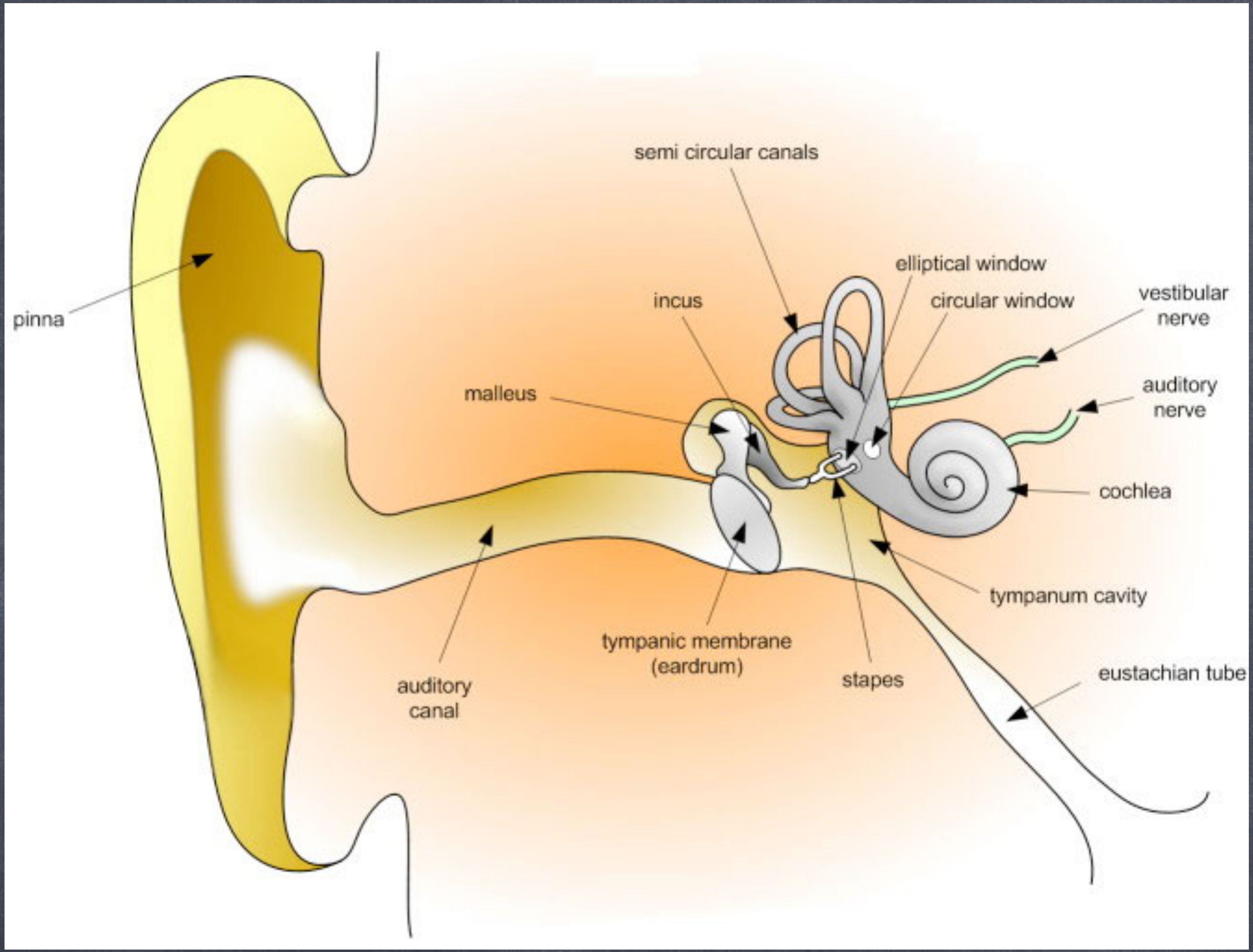
AIFF Sound format

- why 44 100 samples / second ?
- because it is in the correct range...
- because 44 100 is divisible by 2,3,4,5,6,7,9,10

MP3 Sound format

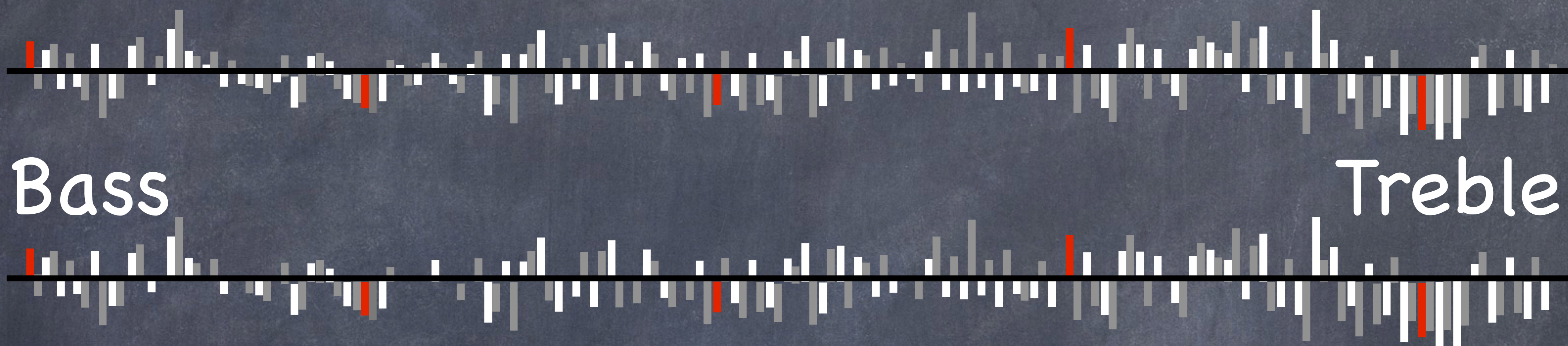
- Based on Fourrier transform.
- 576 samples of amplitude / time are converted to 576 samples of distinct frequencies.



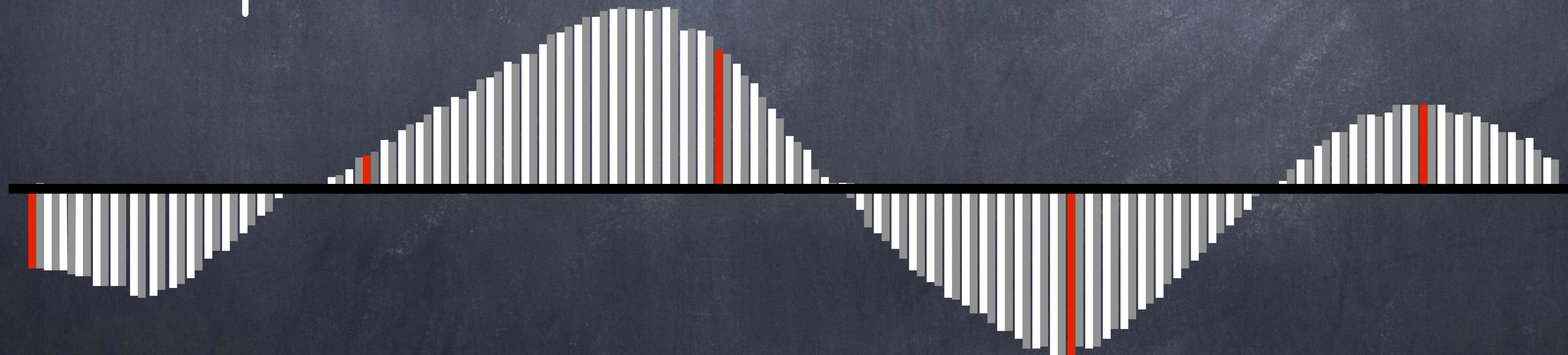


- In human ears, the cochlea is mechanically performing a process analog to the Fourrier Transform. The eardrum vibrates back and forth according to the wave-like representation of the sound. The frequency information stimulates a specific area in the cochlea.

MP3 Sound format

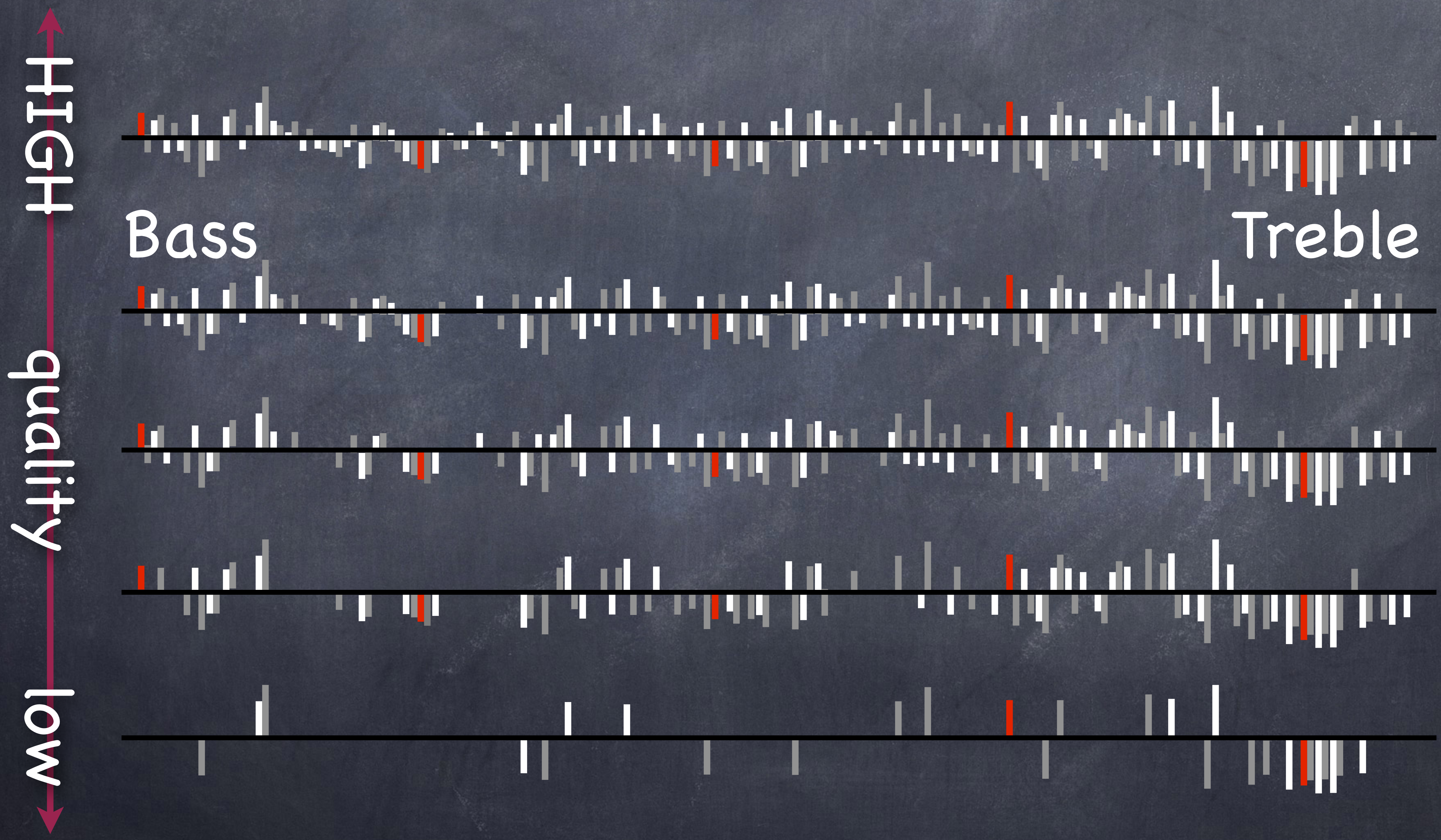


- Frequencies with small coefficients removed



- Waveform reconstructed is close to original

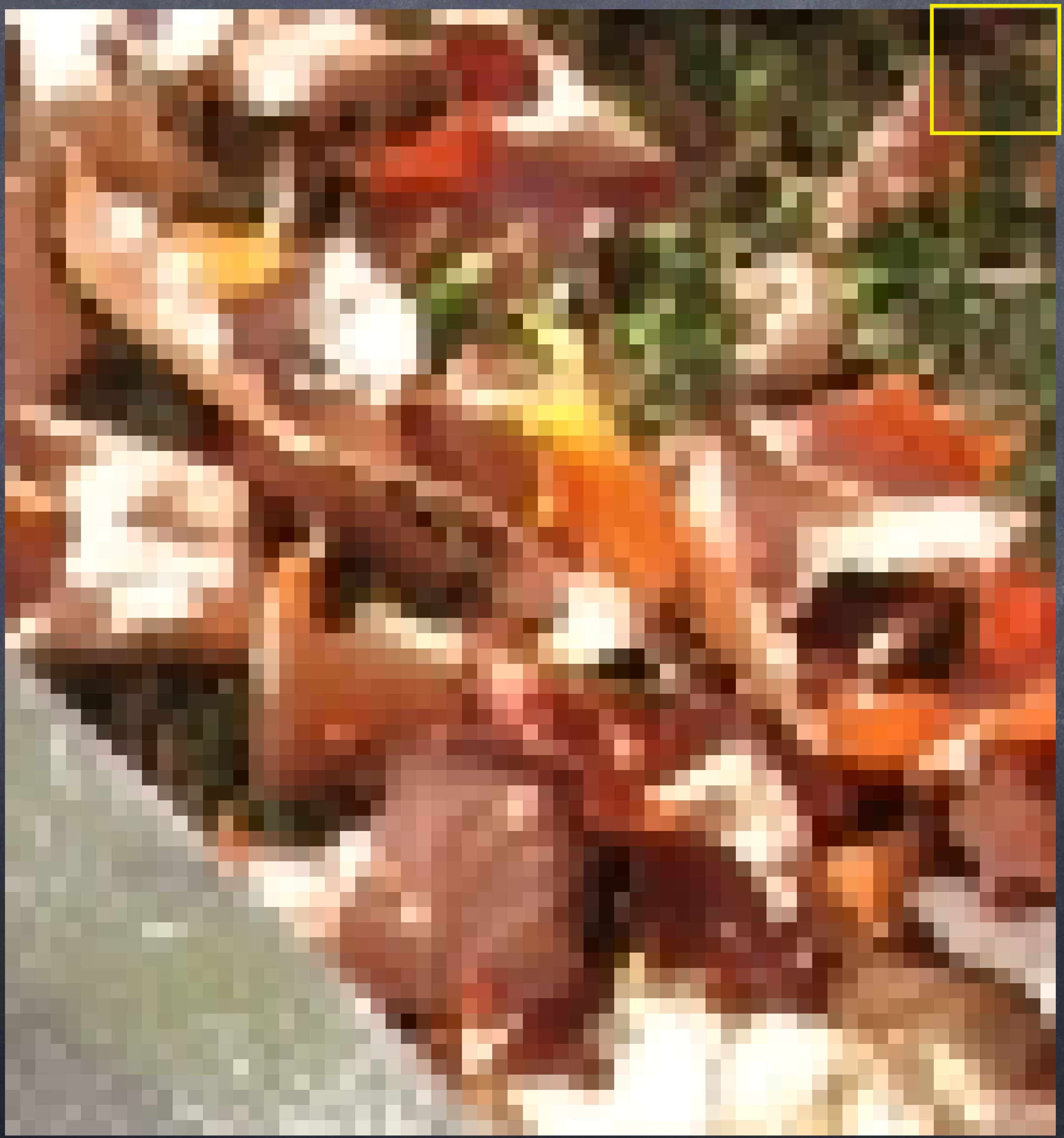
MP3 Sound format



Data Representation

- Image formats

TIFF image format

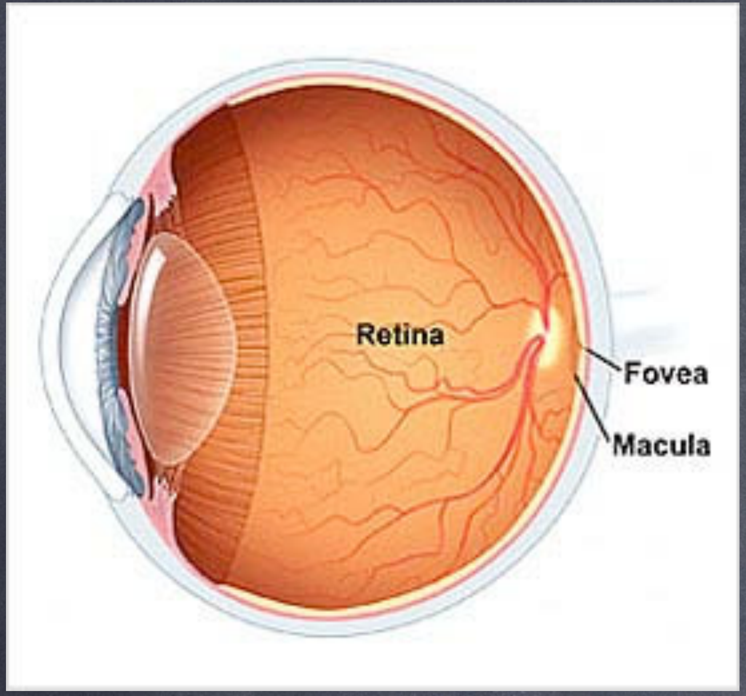


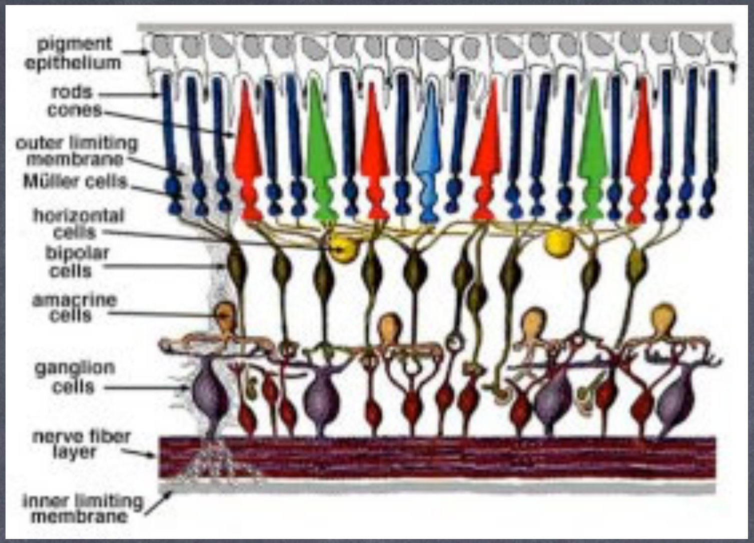
TIFF image format

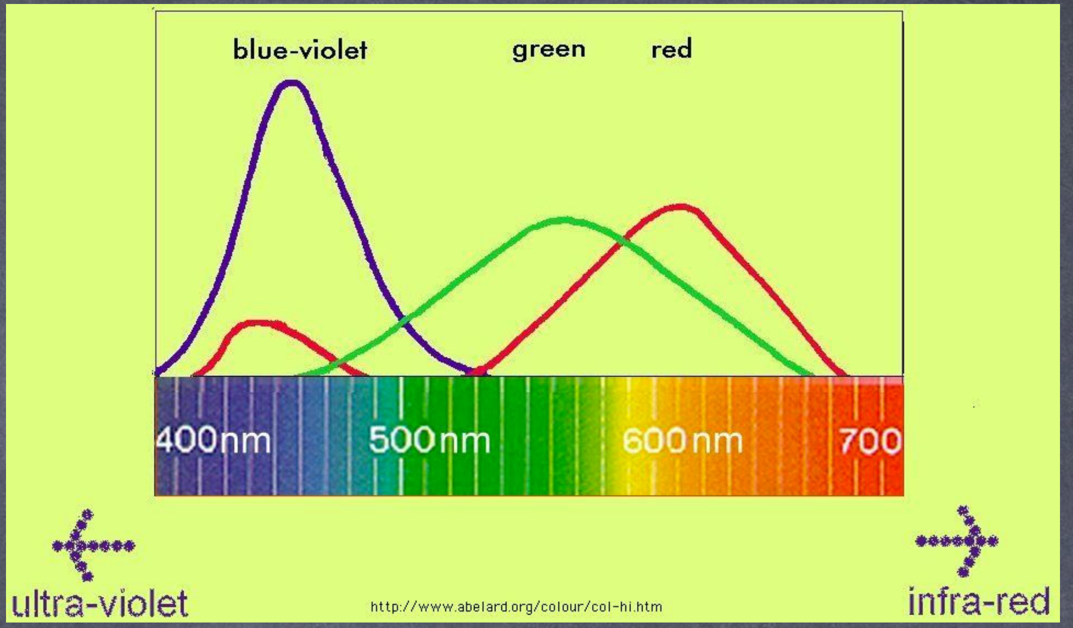
- an 8x8 sub-region of a large image:

- each individual pixel uses 24 bites: 8b for red, 8b for blue, 8b for green.

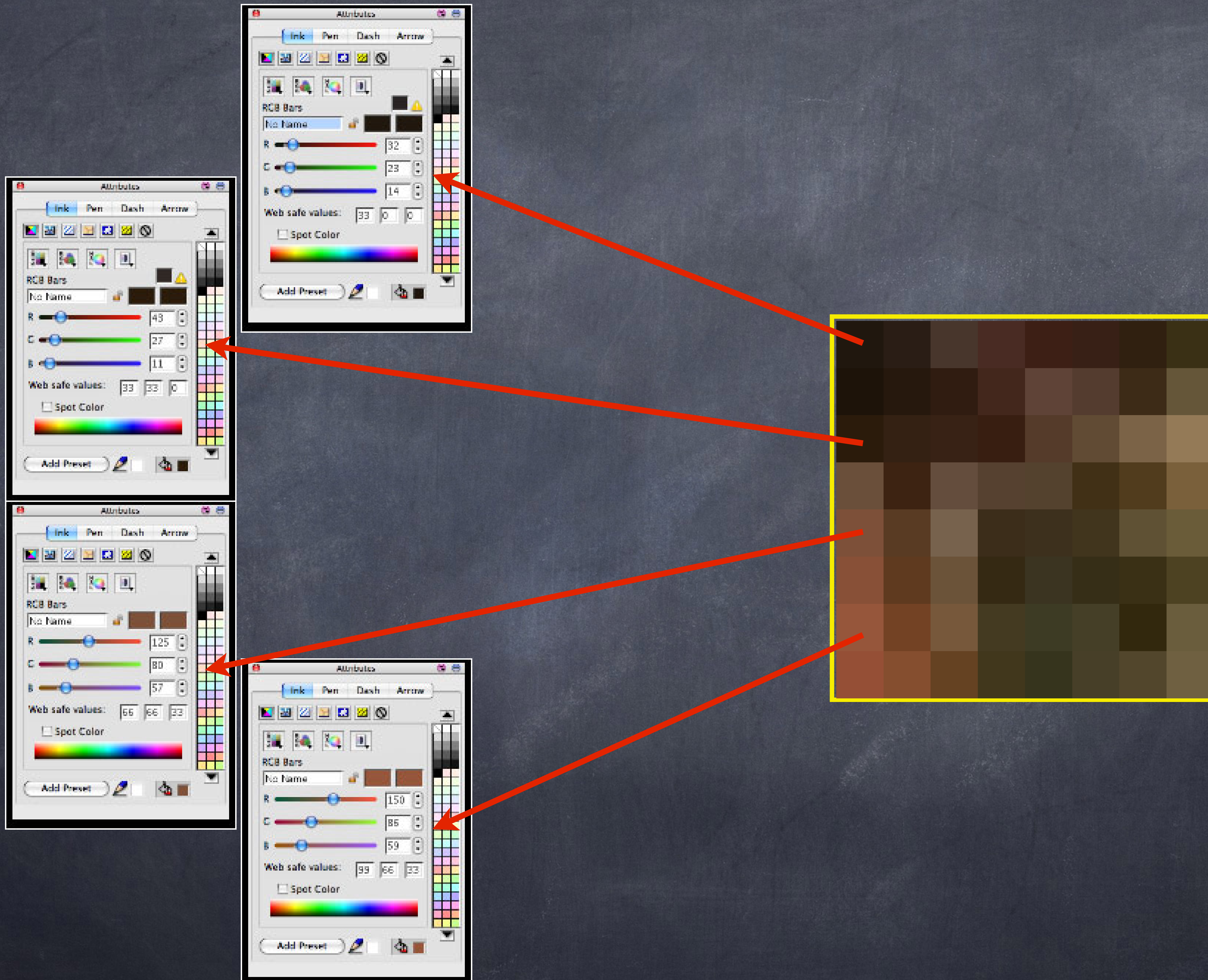
- total size = number of pixels x 3 Bytes.







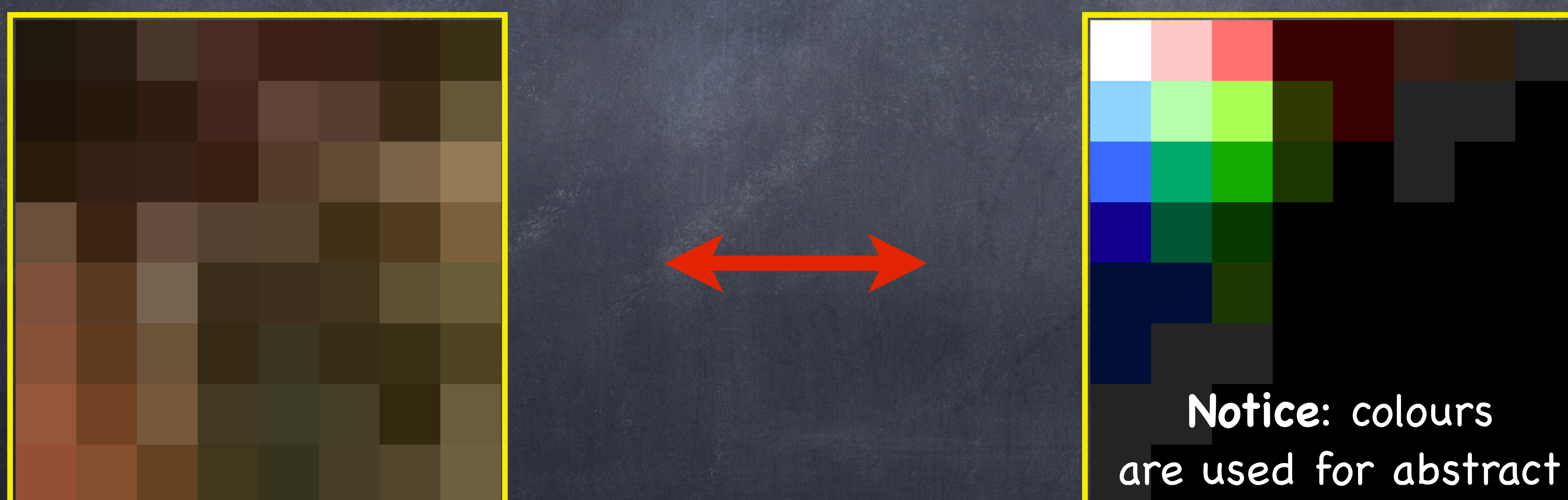
- Animal eyes focus light on the retina where an image of the environment is produced.
- This image is analysed according to 3 types of colour sensitive cones, mostly triggered near the red, green and blue bands.
- A perceived colour is a triplet (x,y,z) of excitations of the 3 types of cones.
- Two combinations of colours yielding the same triplet (x,y,z) are indistinguishable.



Attributes
Ink Pen Dash Arrow
RGB Bars
No Name
R 32
G 23
B 14
Web safe values: 33 0 0
Spot Color
Add Preset
Attributes
Ink Pen Dash Arrow
RGB Bars
No Name
R 43
G 27
B 11
Web safe values: 33 33 0
Spot Color
Add Preset
Attributes
Ink Pen Dash Arrow
RGB Bars
No Name
R 125
G 80
B 57
Web safe values: 66 66 33
Spot Color
Add Preset
Attributes
Ink Pen Dash Arrow
RGB Bars
No Name
R 150
G 86
B 59
Web safe values: 99 66 33
Spot Color
Add Preset

JPEG image format

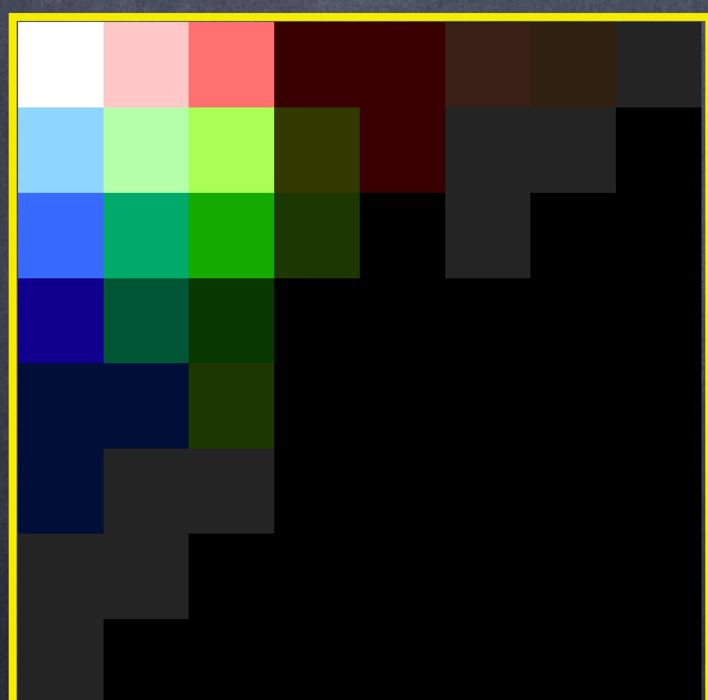
- Using a transformation similar to Fourier transform (used for audio), a so called Discrete Cosine Transform is applied to each sub-bloc of size 8x8.



Notice: colours are used for abstract data. Dark means small, bright means large.

JPEG image format

- If no data is removed, the resulting image is nearly identical to the original. Imprecision in the transform causes small errors.



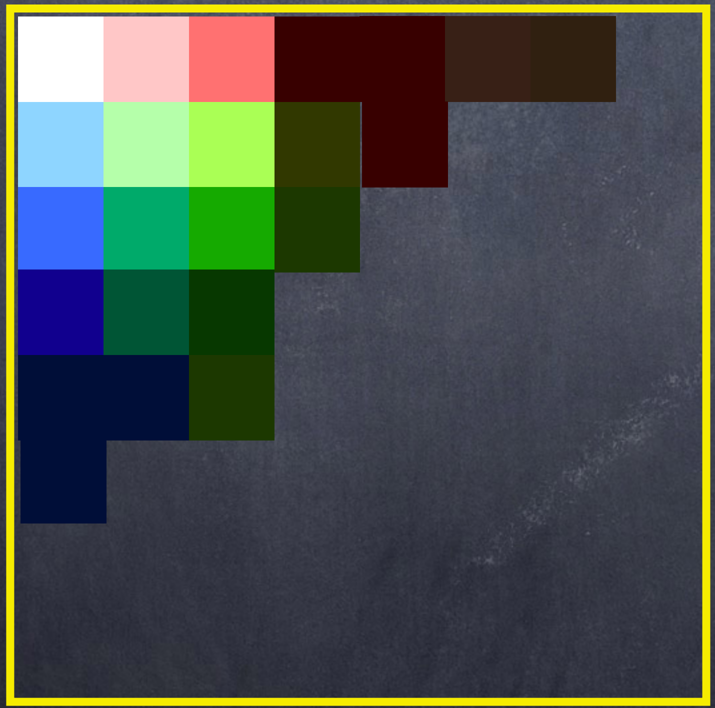
JPEG image format

- If all data very close to zero is removed, the resulting image is only slightly different from the original

Notice: colours are used for abstract data. Dark means small, bright means large.

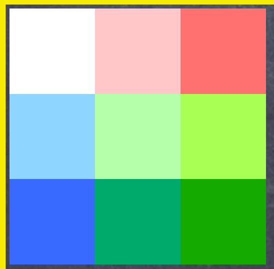
JPEG image format

- If all data close to zero is removed, the resulting image is somewhat different from the original



JPEG image format

- If all data of small magnitude is removed, the resulting image is still very similar to the original



Notice: colours are used for abstract data. Dark means small, bright means large.

JPEG image format

- If only data of large magnitude is kept, the resulting image is similar but quite different from the original. Most details are wiped out.

Data Representation

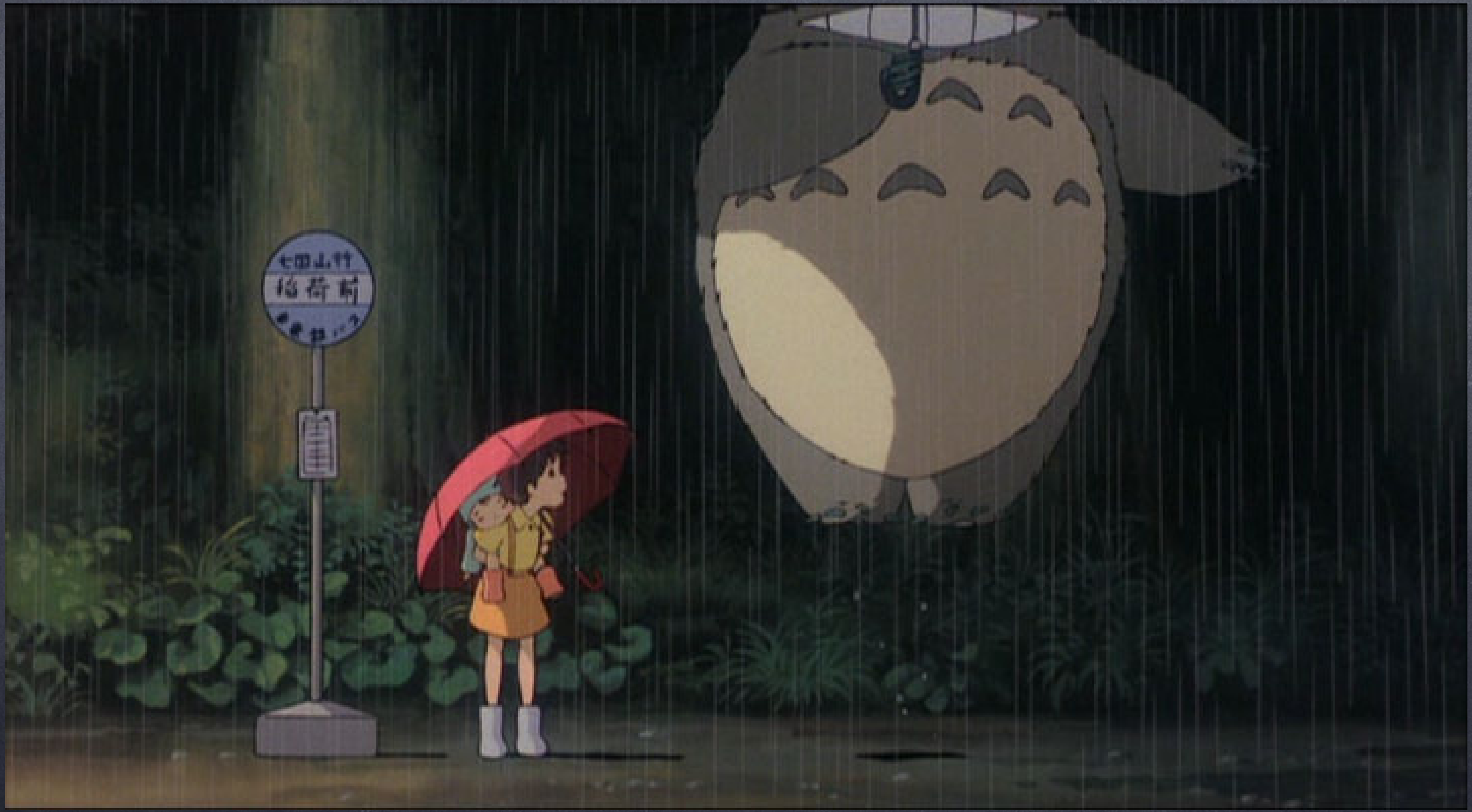
- movie formats

RAW movie format

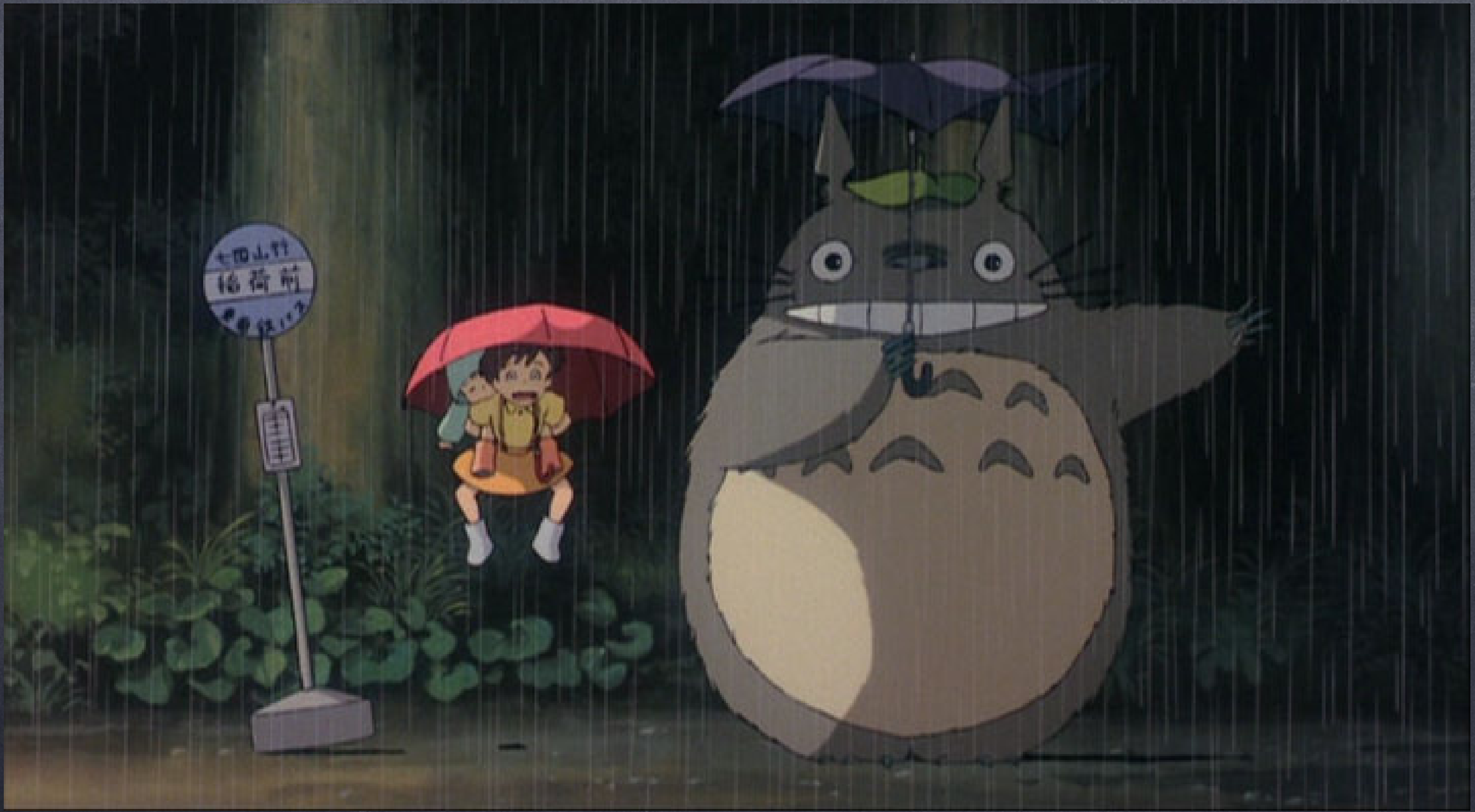
- 720×576 pixels per frame
- 24 bits (colour) per pixel
- 30 frames per second
- $30 \times 3 \times 720 \times 576 \approx 37$ MB/s ≈ 135 GB/hour
- typically 200 GB per movie !!! ($\approx$ 50 DVDs)

MPEG2 Movie Format

MPEG2 Movie Format



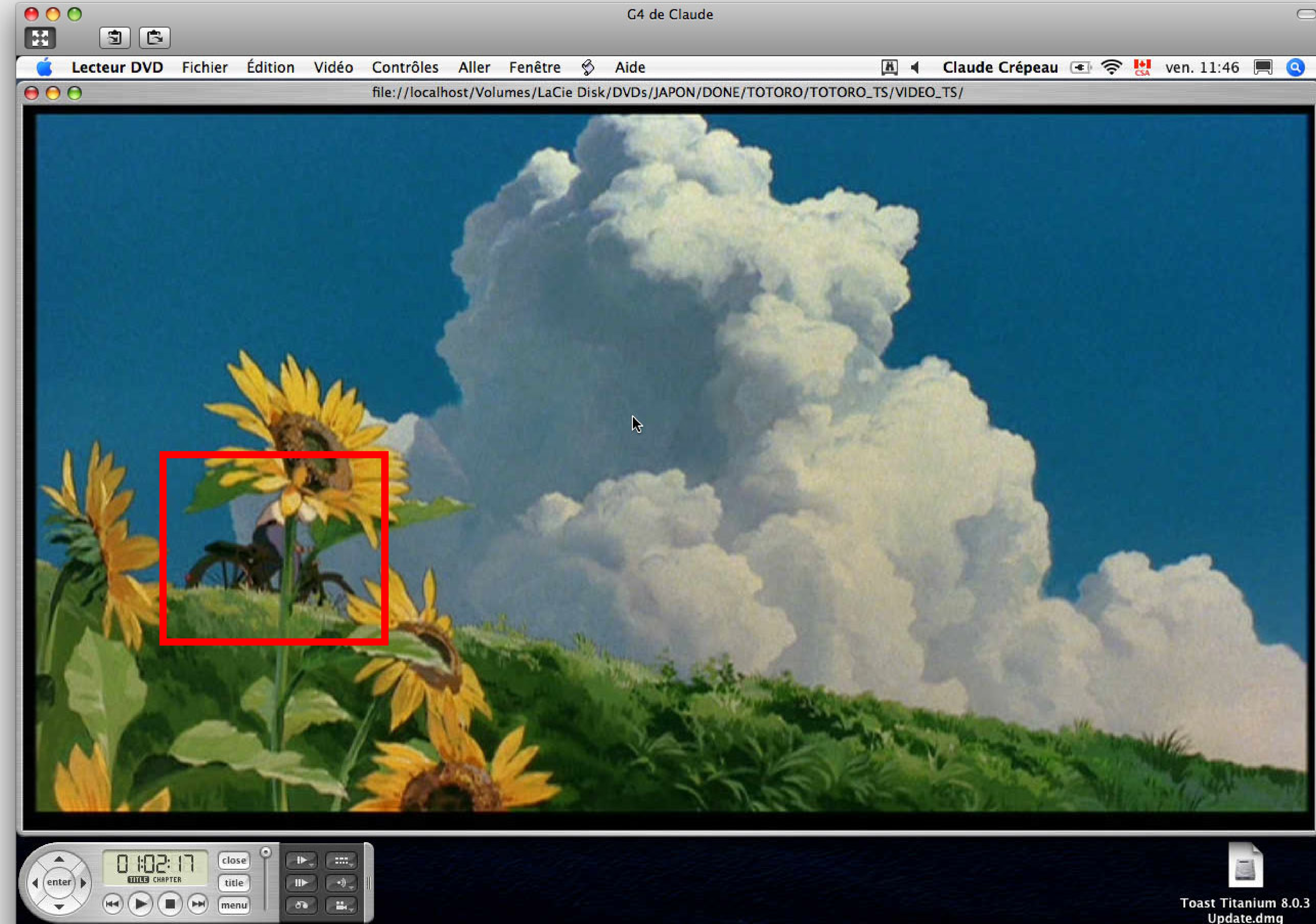
MPEG2 Movie Format

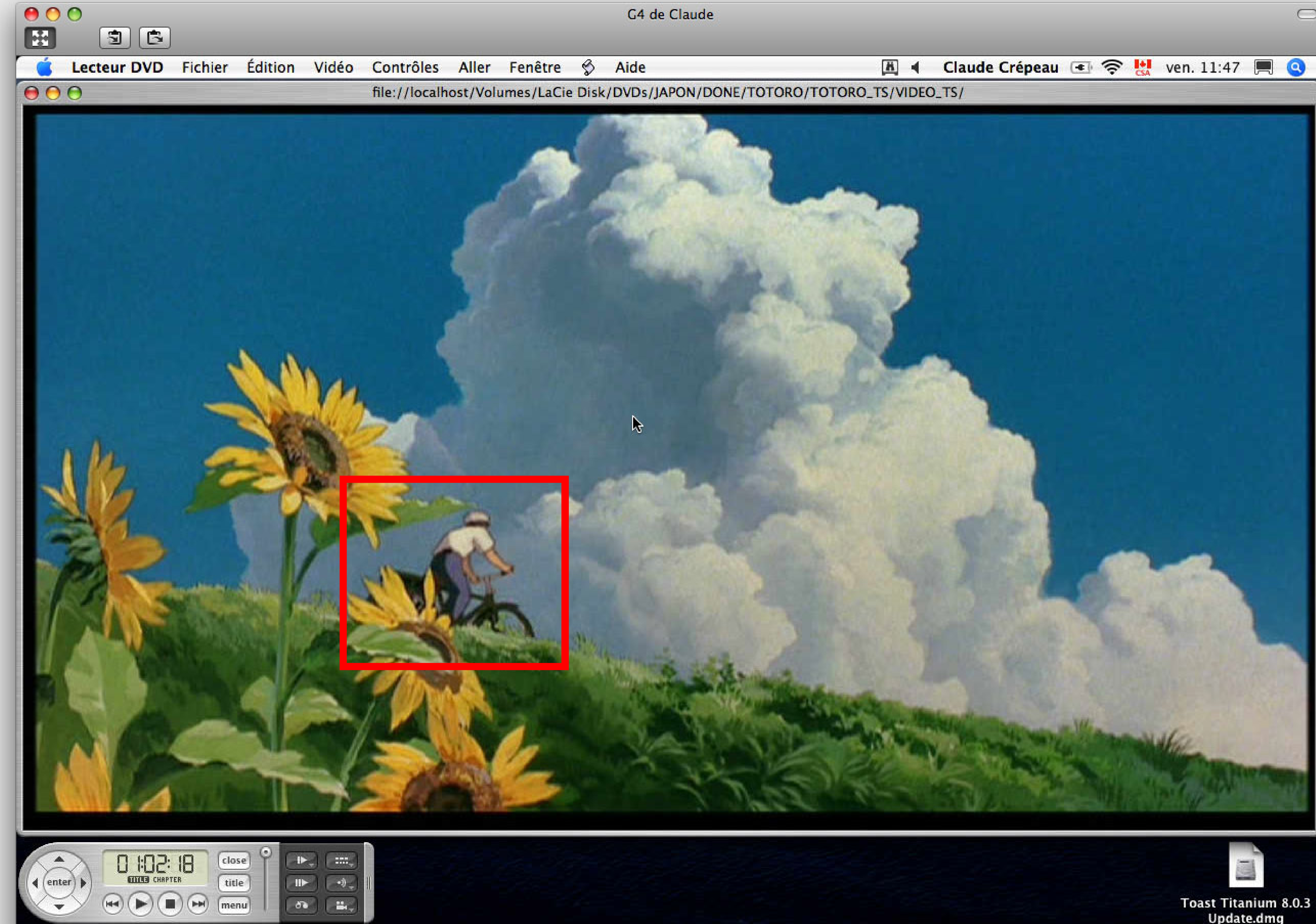


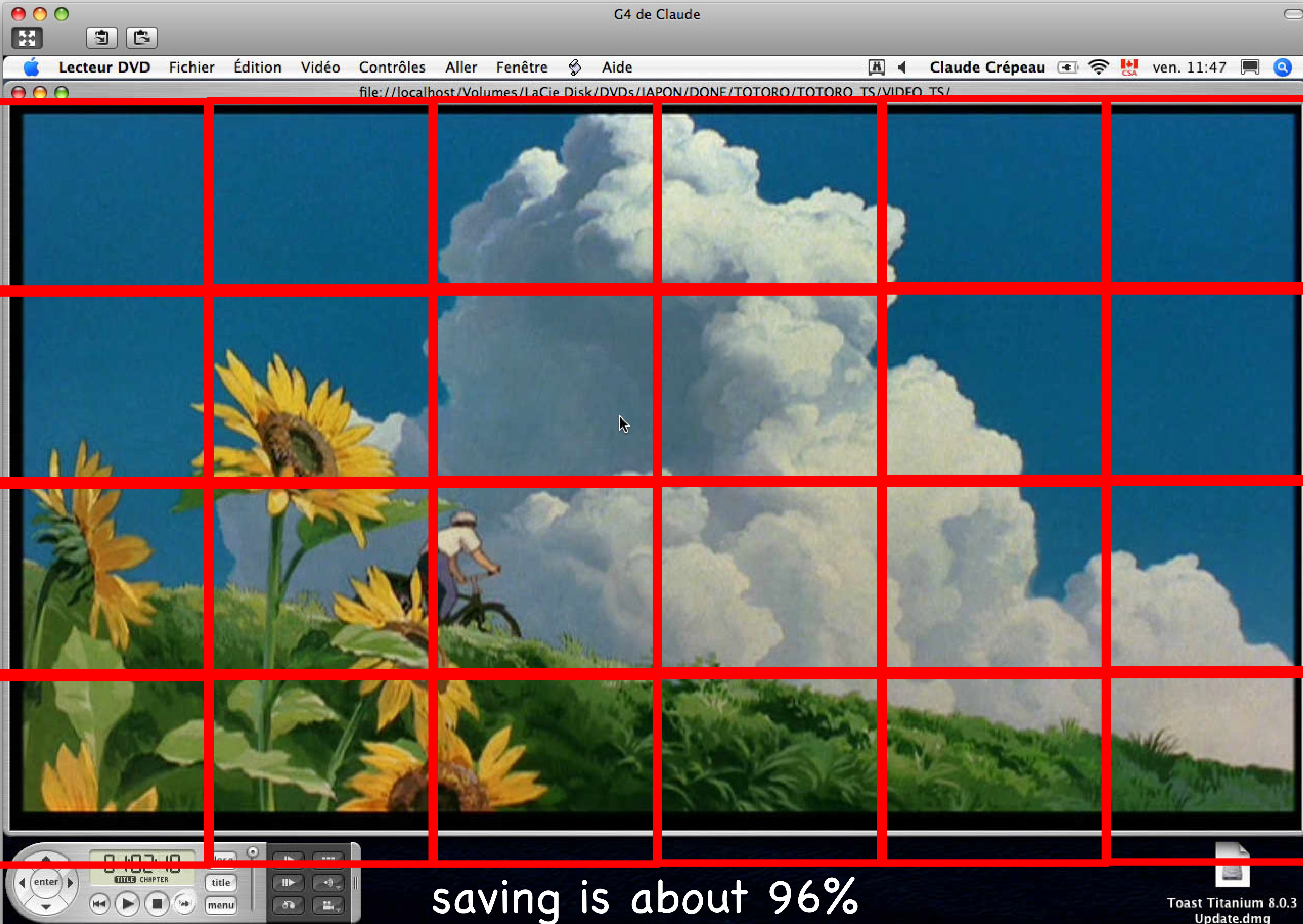
MPEG2 Movie Format

MPEG2 format

- Fixed Background images


G4 de Claude
Lecteur DVD
Fichier
Édition
Vidéo
Contrôles
Aller
Fenêtre
Aide
Claude Crépeau
ven. 11:46
file://localhost/Volumes/LaCie Disk/DVDs/JAPON/DONE/TOTORO/TOTORO_TS/VIDEO_TS/
0 1:02:17
TITLE CHAPTER
close
title
menu
enter
Toast Titanium 8.0.3
Update.dmg


G4 de Claude
Lecteur DVD
Fichier
Édition
Vidéo
Contrôles
Aller
Fenêtre
Aide
Claude Crépeau
ven. 11:47
file://localhost/Volumes/LaCie Disk/DVDs/JAPON/DONE/TOTORO/TOTORO_TS/VIDEO_TS/
0 1:02:18
TITLE CHAPTER
close
title
menu
enter
Toast Titanium 8.0.3
Update.dmg



saving is about 96%

MPEG2 format

- Travelling


G4 de Claude
Lecteur DVD
Fichier
Édition
Vidéo
Contrôles
Aller
Fenêtre
Aide
Claude Crépeau
ven. 12:00
file://localhost/Volumes/LaCie Disk/DVDs/JAPON/DONE/TOTORO/TOTORO_TS/VIDEO_TS/
Image par image
0 1:00:12
TITLE CHAPTER
close
title
menu
enter
Toast Titanium 8.0.3
Update.dmg


G4 de Claude
Lecteur DVD
Fichier
Édition
Vidéo
Contrôles
Aller
Fenêtre
Aide
Claude Crépeau
ven. 12:01
file://localhost/Volumes/LaCie Disk/DVDs/JAPON/DONE/TOTORO/TOTORO_TS/VIDEO_TS/
0 1:00:12
TITLE CHAPTER
close
title
menu
enter
Toast Titanium 8.0.3
Update.dmg


G4 de Claude
Lecteur DVD
Fichier
Édition
Vidéo
Contrôles
Aller
Fenêtre
Aide
Claude Crépeau
ven. 12:02
file://localhost/Volumes/LaCie Disk/DVDs/JAPON/DONE/TOTORO/TOTORO_TS/VIDEO_TS/
0 1:00:13
TITLE CHAPTER
close
title
menu
enter
Toast Titanium 8.0.3
Update.dmg


G4 de Claude
Lecteur DVD
Fichier
Édition
Vidéo
Contrôles
Aller
Fenêtre
Aide
Claude Crépeau
ven. 12:02
file://localhost/Volumes/LaCie Disk/DVDs/JAPON/DONE/TOTORO/TOTORO_TS/VIDEO_TS/
0 1:00:13
TITLE CHAPTER
close
title
menu
enter
Toast Titanium 8.0.3
Update.dmg

MPEG2 format

- Each image is encoded with JPEG or similar.
- Sound is encoded with MP3 or similar.
- Most frames use only small amount of info to construct from previous frames.
- A complete frame is displayed every so often to make sure the fix part or travelling part has not substantially changed.

Winter 2016
COMP-250: Introduction to Computer Science

Lecture 24, April 7, 2016