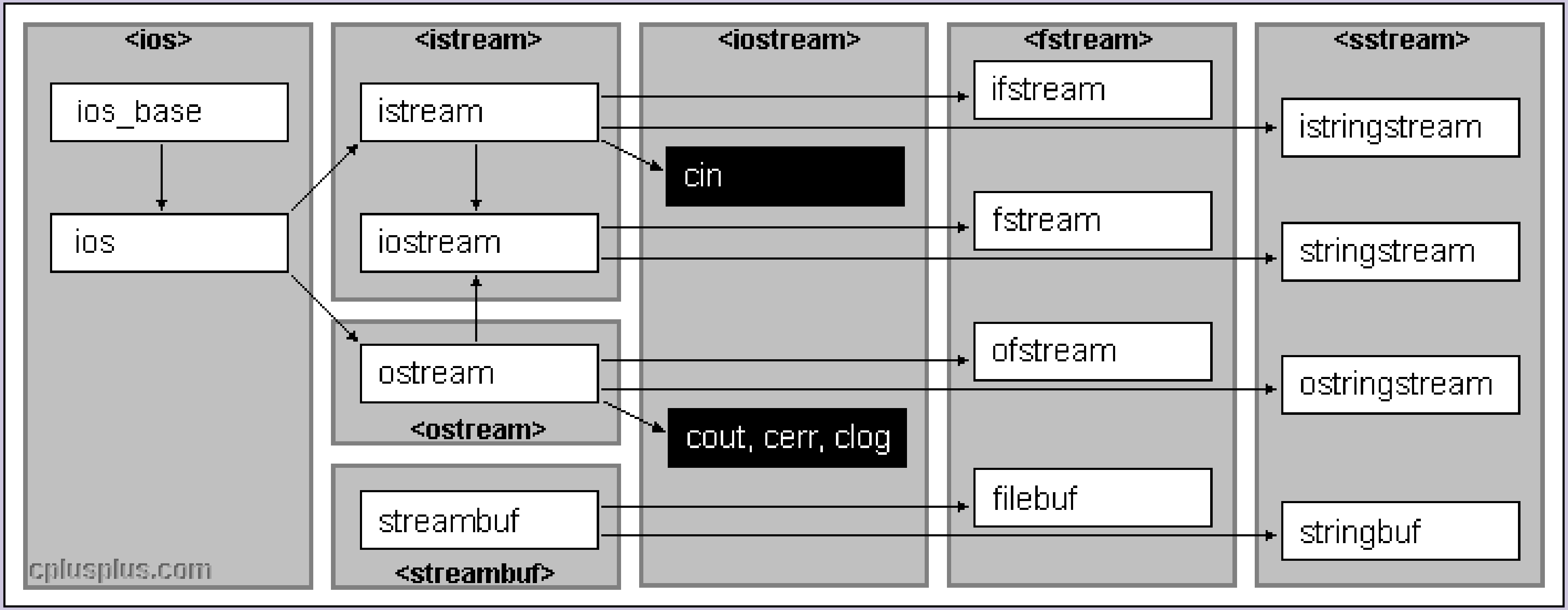
Streams

Version 1: Dr. Ofir Pele
Version 2: Dr. Erel Segal-Halevi

Class hierarchy



Output stream

- The **ostream** object overloads the **<<** operator for each basic type.
- The operator returns a reference to the output stream, which allows combined output:

```
std::cout << “2 + 3 = “ << 2 + 3 << std::endl;
```

Standard output stream objects

- **`cout`** **-** attached to stdout.
- **`cerr`** **-** attached to stderr, unbuffered.
- **`clog`** **-** attached to stderr, buffered.

We can redirect stdout and stderr
to different files; *see folder 2*.

Other output stream objects (folder 2)

- **`ostringstream -`** attached to a string.
- **`ofstream -`** attached to a file.

Output stream manipulators (folder 3)

- We can "write" to ostream, functions that do not create any output, but rather change some variables of the ostream.
- For example:
-
  ```
      cout << setprecision(4) << 1234.5678
           << setprecision(100) << 1234.5678
  ```
- "setprecision" does not print anything – it just modifies the precision level of the stream.
- *How does it work?* – operator overloading!

  http://cs.brown.edu/~jwicks/libstdc++/html_user/iomanip-source.html

Input stream

- **istream** is the type defined by the library for input streams.
- **cin** is a global object of type **istream** attached to stdin.
- Example:

```
#include <iostream>
int i;
std::cin >> i; // reads an int
```

Other input stream objects

- **`istringstream`** **-** attached to a string.
- **`ifstream`** **-** attached to a file.

Input stream continued

- When an error occurs (typically because the input format is not what we expect) **cin** enters a **failed** state and evaluates to **false**.
  - **istream** overloads the ! opeator and the void* (conversion) operator
- normal usage:

```
ifstream fin("database.tsv");
while (fin >> name >> phone) {
 // do something with name,phone
}
```

Input stream errors

- In failed state `istream` will produce no input.
- `istream` rewinds on error.
- Use `clear()` method to continue reading.

More I/O methods (folder 5)

- Both **ostream** and **istream** have additional methods:
  - **ostream& put(char ch)**
  - **ostream& write(char const *str, int length)**
  - **int get() // read one char**
  - **istream& get(char& ch) // read one char**
  - **getline(char *buffer, int size,**
    int delimiter = '\n')

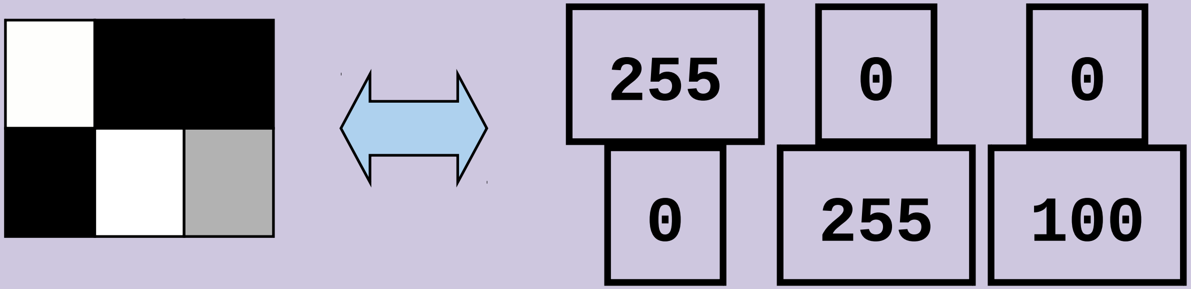
- Examples:

```
std::cout.put(‘a’);
char ch1, ch2, str[256];
std::cin.get(ch1).get(ch2);
std::cin.getline(str, 256);
```

Binary files

Leading example: image files

- Images are stored as matrices of numbers (pixels)
- Here, we deal with gray-scale images
- *8* bits per pixel
  - i.e. each pixel between 0 and 255
- 255 is white, 0 is black, others are gray



255	0	0
0	255	100

storing images

- How can we store images on files?
- For each image we want to store:
  - width
  - height
  - number of bytes per pixel
  - the pixels
- Requirements: read/write easily, save space, save computation, etc.

storing images

First try: text files

cons:

- long
- needs parsing

pros:

- readable by humans
- easy to edit

“myImg.txt”

```
width = 3
height = 2
bytes_per_pixel = 1
255 0     0
0     255 100
```

storing images

Better solution: **Binary files**

- Save the data the way the computer holds it

pros:

- Smaller
- No parsing (faster)

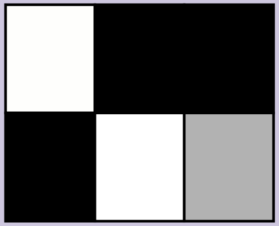
cons:

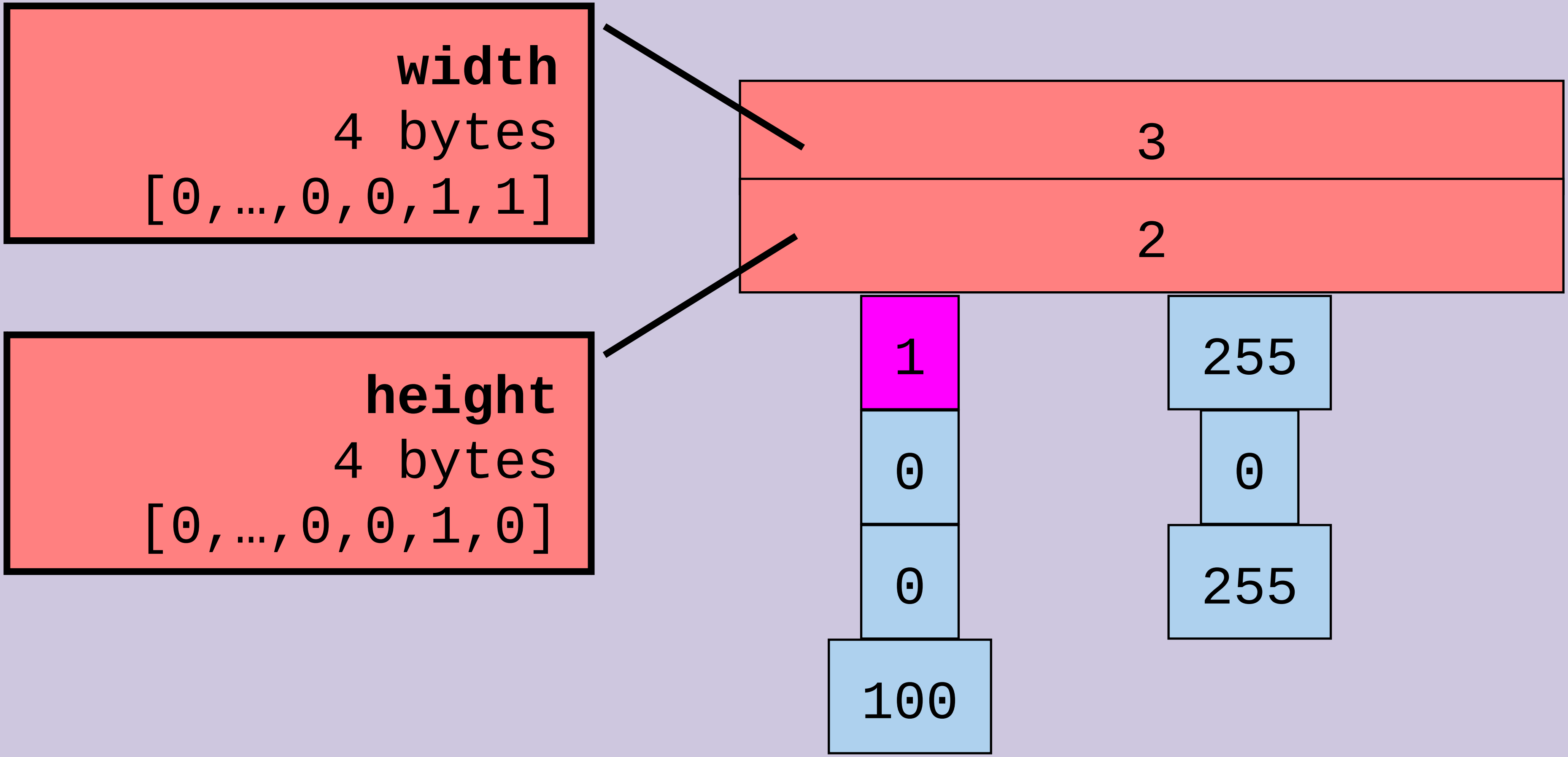
- hard to read for humans
- Machine dependant

- Widely used:

JPEG, mp3, BMP, other data

Images as binary files





width
4 bytes
[0,…,0,0,1,1]

height
4 bytes
[0,…,0,0,1,0]

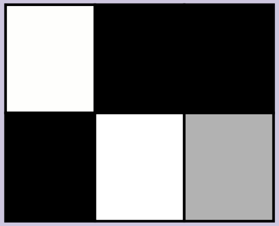
3

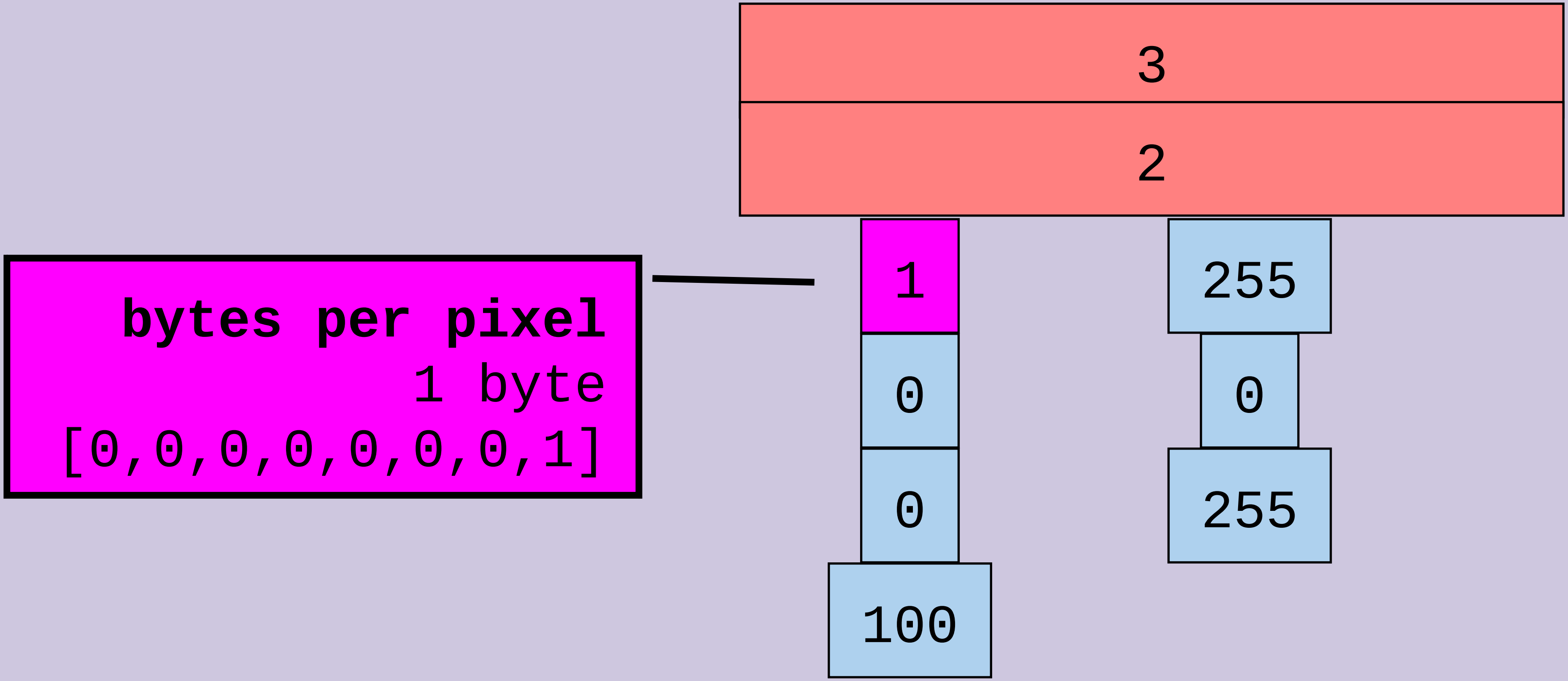
2

1
0
0
100

255
0
255

Images as binary files





3

2

bytes per pixel

1 byte

[0,0,0,0,0,0,0,1]

1

0

0

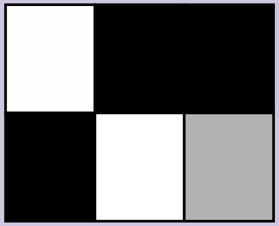
100

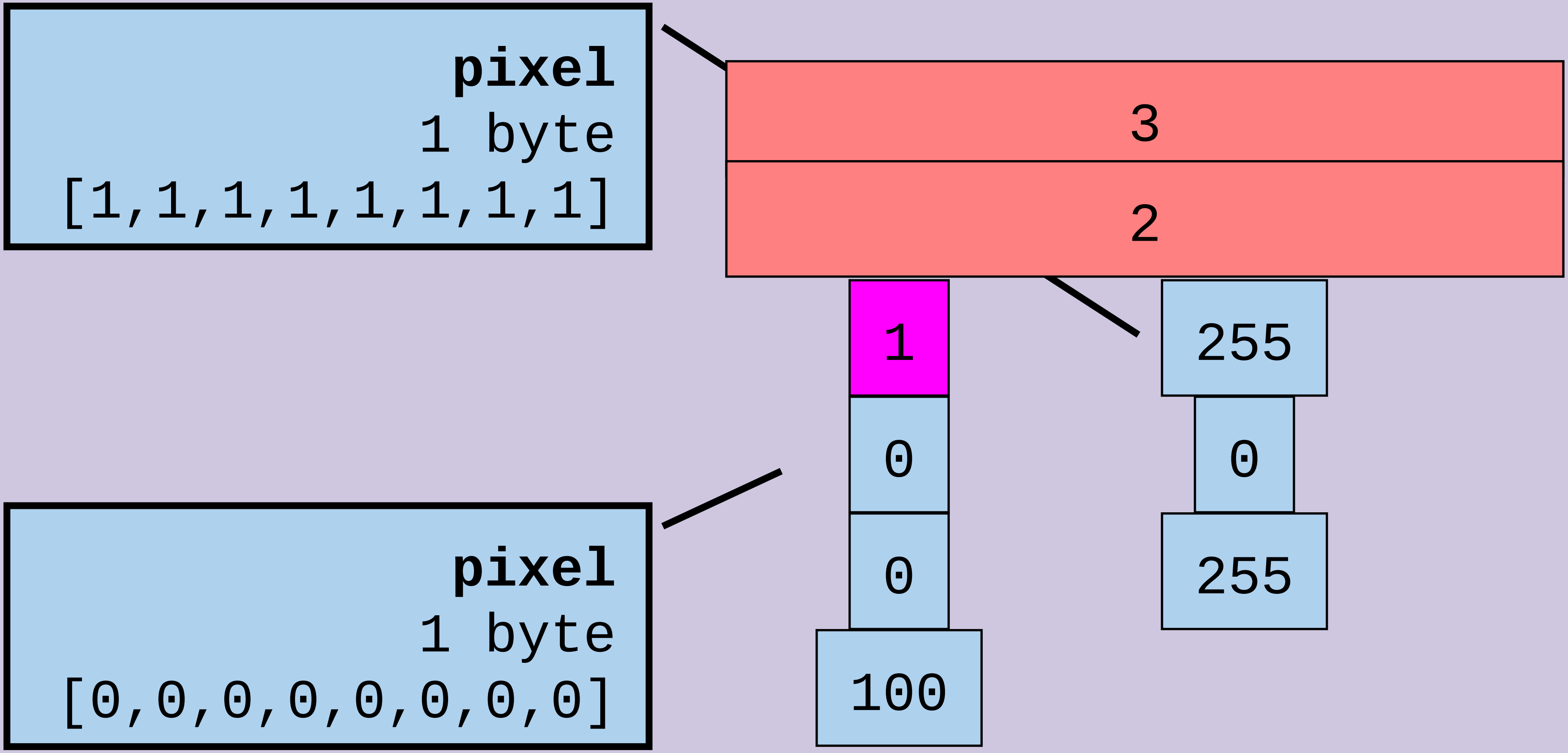
255

0

255

Images as binary files





pixel
1 byte
[1,1,1,1,1,1,1,1]

3

2

1

0

0

100

255

0

255

pixel
1 byte
[0,0,0,0,0,0,0,0]

Images as binary files - colors

In a colorful image, each pixel should contain more information than just the light intensity.

A common way to represent colors is RGB (Red, Green, Blue).

Each pixel requires 3 bytes – one for Red, one for Green, one for Blue.

See example in folder 5.