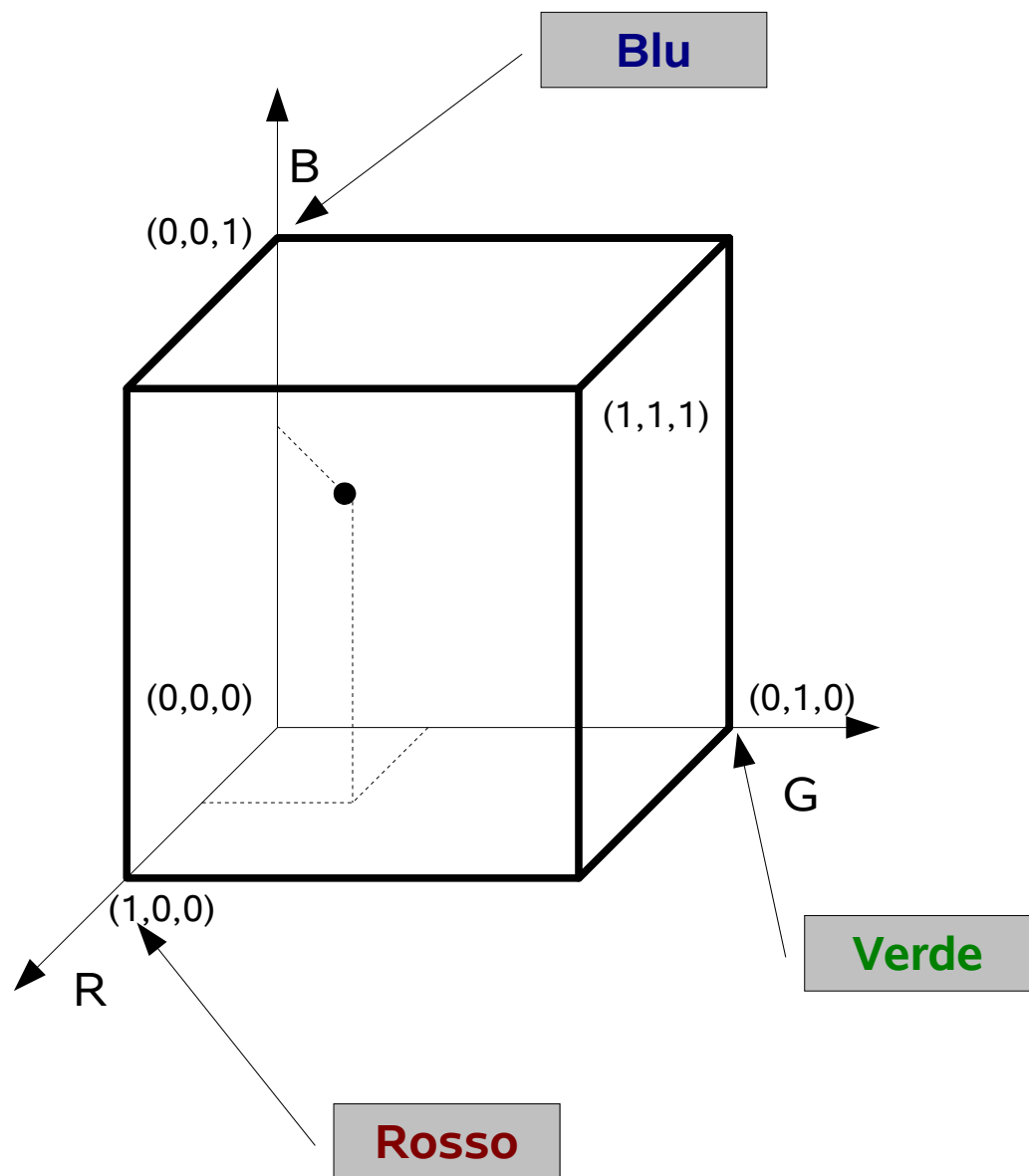


Rappresentazione di immagini

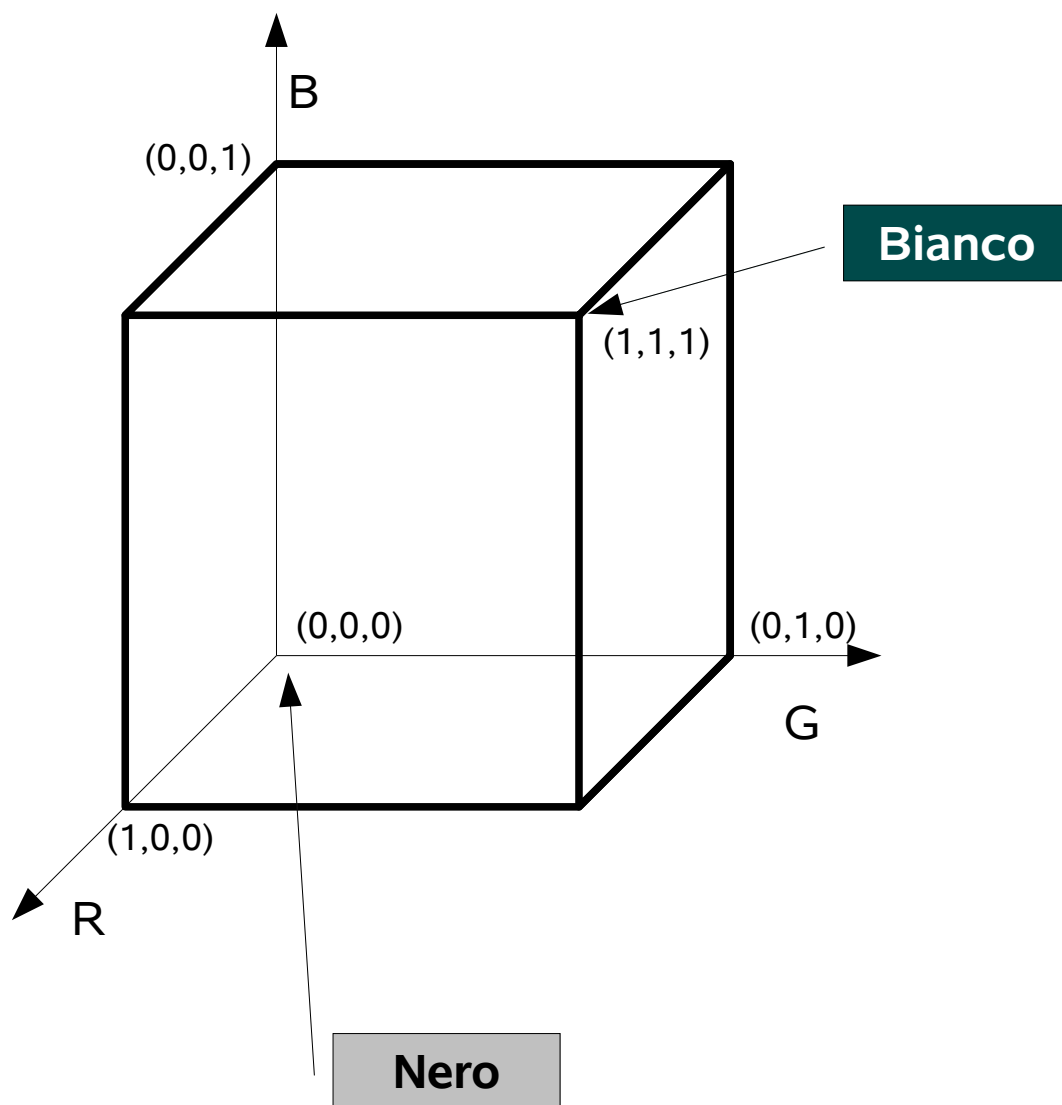
- Il modo più immediato per rappresentare un'immagine è quello di considerarla suddivisa in un reticolo di punti detti **pixel**.
- Maggiore è il numero di pixel per unità di lunghezza (dpi: dot per inch) e migliore sarà la definizione dell'immagine
- Ad ogni pixel è essere associato un colore.
- I colori possono essere definiti numericamente in vari modi (RGB, CYMK ecc)

Rappresentazione dei colori



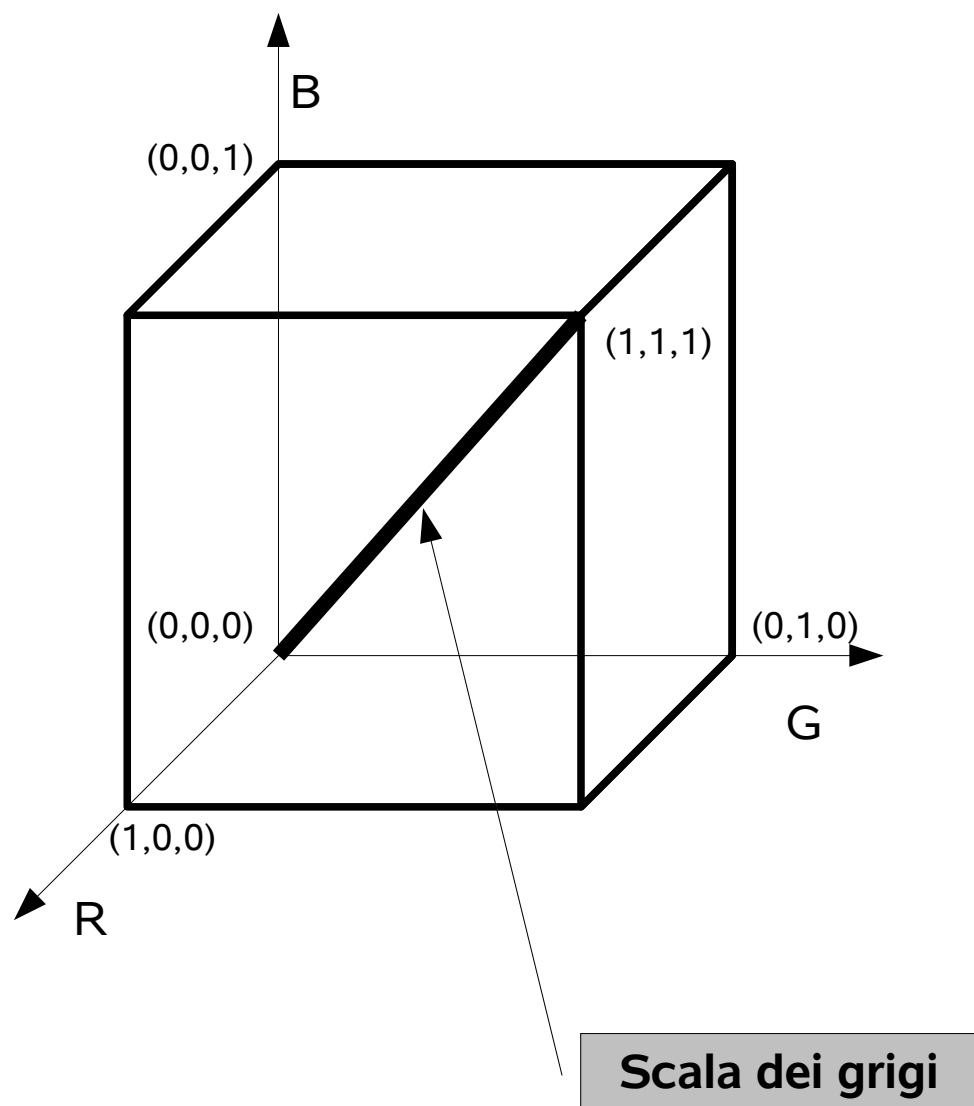
- Ad ogni **punto** del cubo corrisponde un miscela di colori proporzionali alle coordinate (dette RGB)
- Es. (0.5, 0.1, 0.2) corrisponde al 50% di **rosso**, 10% di **verde** e 20 % di **blu**

Rappresentazione dei colori



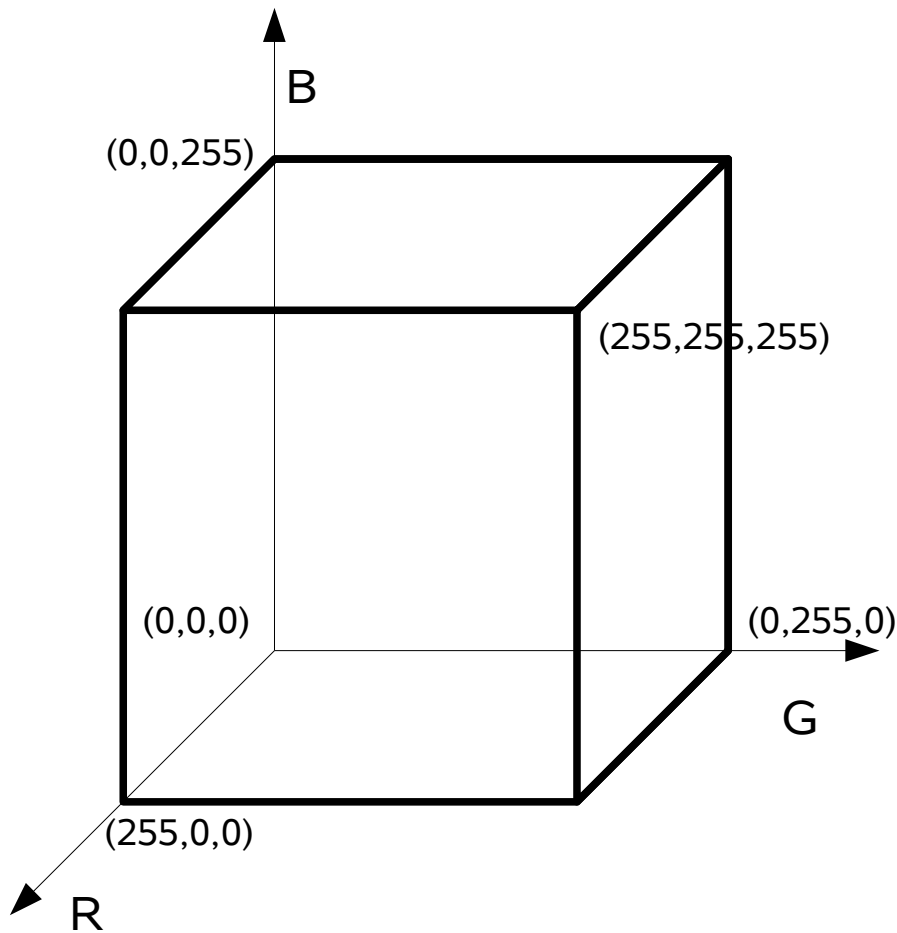
- L'origine delle coordinate corrisponde al **Nero** (assenza di colore)
- Il vertice opposto del cubo corrisponde al **Bianco** (presenza di tutti i colori)

Rappresentazione dei colori



- I punti che stanno sulla diagonale Bianco – Nero hanno la medesima percentuale dei tre colori fondamentali e corrispondono alla **scala dei grigi**

Rappresentazione dei colori



- Nella rappresentazione dei colori le coordinate, anziché essere espresse tra 0 e 1 sono espresse (quantizzate) con un intero tra 0 e 255 (un byte)
- In tal caso si parla di colore a **24 bit**

Rappresentazione di immagini

- Un immagine di 640 x 480 pixel può essere rappresentata con una matrice
- Se l'immagine è **monocromatica** ogni punto è rappresentabile con un bit

0 : punto **bianco**

1 : punto **nero**

- L'occupazione di memoria è bassa:

$640 \times 480 / 8 = 38.400$ bytes

$1024 \times 768 / 8 = 98.304$ bytes

Rappresentazione di immagini

- Se l'immagine è in **scala di grigio** ogni pixel rappresenta una tonalità di grigio compresa tra 0 e 255 (1 byte)

0 : punto **bianco**

255 : punto **nero**

- L'occupazione di memoria è intermedia:

$640 \times 480 = 307.200$ bytes

$1024 \times 768 = 786.432$ bytes

Rappresentazione di immagini

- Una matrice tridimensionale può consentire di rappresentare un'immagine a colori.
- Ogni colore è definito da tre componenti (RGB: Red, Green, Blue)
- L'intensità di ogni componente viene rappresentata da un intero compreso tra 0 e 255 (un byte). Es.
 - $\{0,0,0\}$ rappresenta il nero
 - $\{255,255,255\}$ rappresenta il bianco
- Questa è la rappresentazione dei colori a **24 bit**

Rappresentazione di immagini

- Possiamo immaginare di rappresentare un'immagine bidimensionale con una matrice di punti colorati (pixel).
- Un immagine di 640 punti di larghezza e 480 di altezza sarà rappresentabile da una matrice

`imm[480][640][3]`

- il valore **`imm[50][120][2]`** ci dirà quanto **rosso** è contenuto nel pixel di riga 50 e colonna 120

Rappresentazione di immagini

- L'immagine occuperà una quantità di memoria pari a $R \times C \times 3$ bytes dove R e C sono le righe e le colonne
- Es. 640×480 con 24 bits di colore:
occupazione: $640 \times 480 \times 3 = 921.600$ bytes (ca 1 Mbyte!)
- Es. 1024×768 con 24 bits di colore:
occupazione: $1024 \times 768 \times 3 = 2.359.296$ bytes

Formato del file Bitmap (.bmp)

- E' il formato più semplice (ma anche il più dispendioso in termini di memoria) per rappresentare le immagini
- E' un file binario costituito da:
 - un **header** (di 54 bytes)
 - una (eventuale) **palette** di colori (massimo 256 colori corrispondenti a 1024 bytes)
 - i **dati RGB** riguardanti i pixel



Formato del file Bitmap (.bmp)

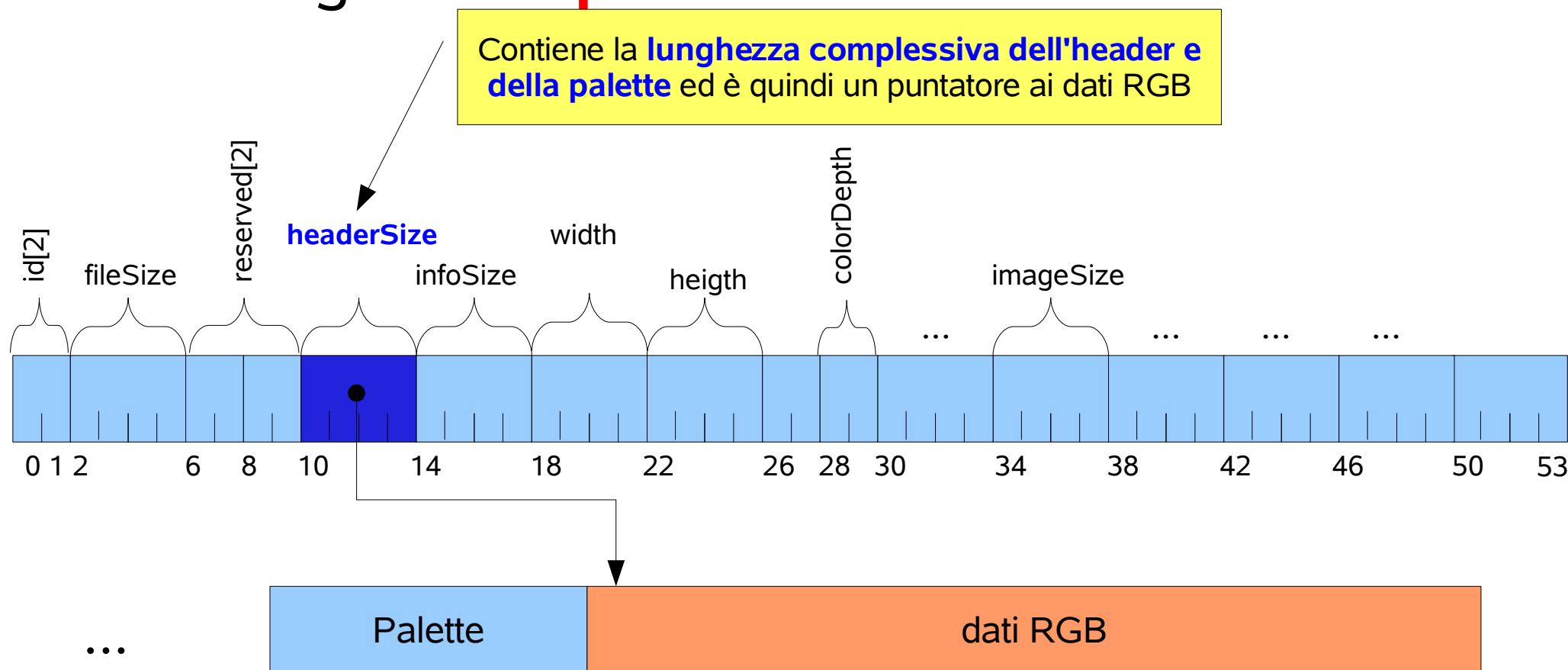
- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	03	00	00	00	04	00	00	00	01	00	18	00	00	00
00000020	00	00	30	00	00	00	13	0b	00	00	13	0b	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	00	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff
00000060	ff	ff	ff	00	00	00										



Formato del file Bitmap (.bmp)

- L'header contiene informazioni riguardanti l'immagine e la **posizione nel file** dei dati RGB

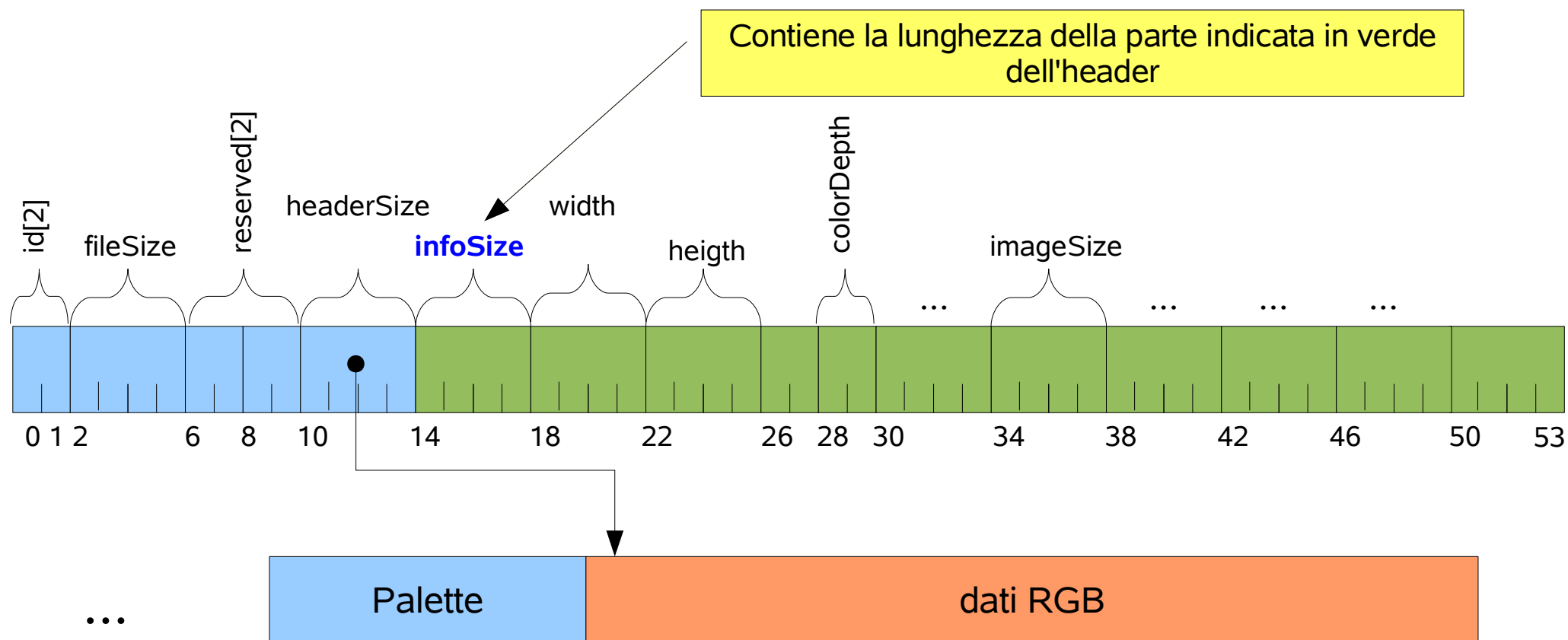


Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

[illegible]

Formato del file Bitmap (.bmp)



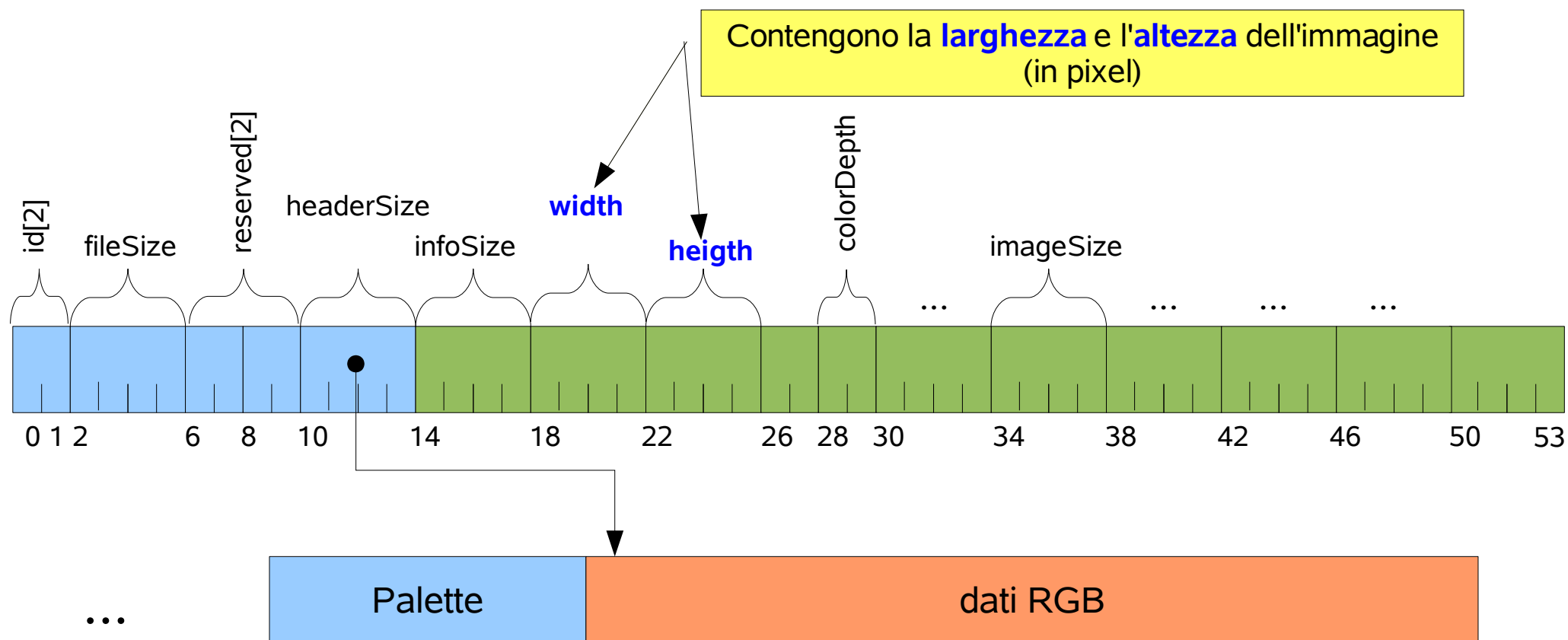
Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	36	00	00	00	<u>28</u>	<u>00</u>
00000010	<u>00</u>	<u>00</u>	03	00	00	00	04	00	00	00	01	00	18	00	00	00
00000020	00	00	30	00	00	00	13	0b	00	00	13	0b	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	00	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff
00000060	ff	ff	ff	00	00	00										

Contiene la lunghezza della parte indicata in verde
dell'header: 0x28 = 40

Formato del file Bitmap (.bmp)



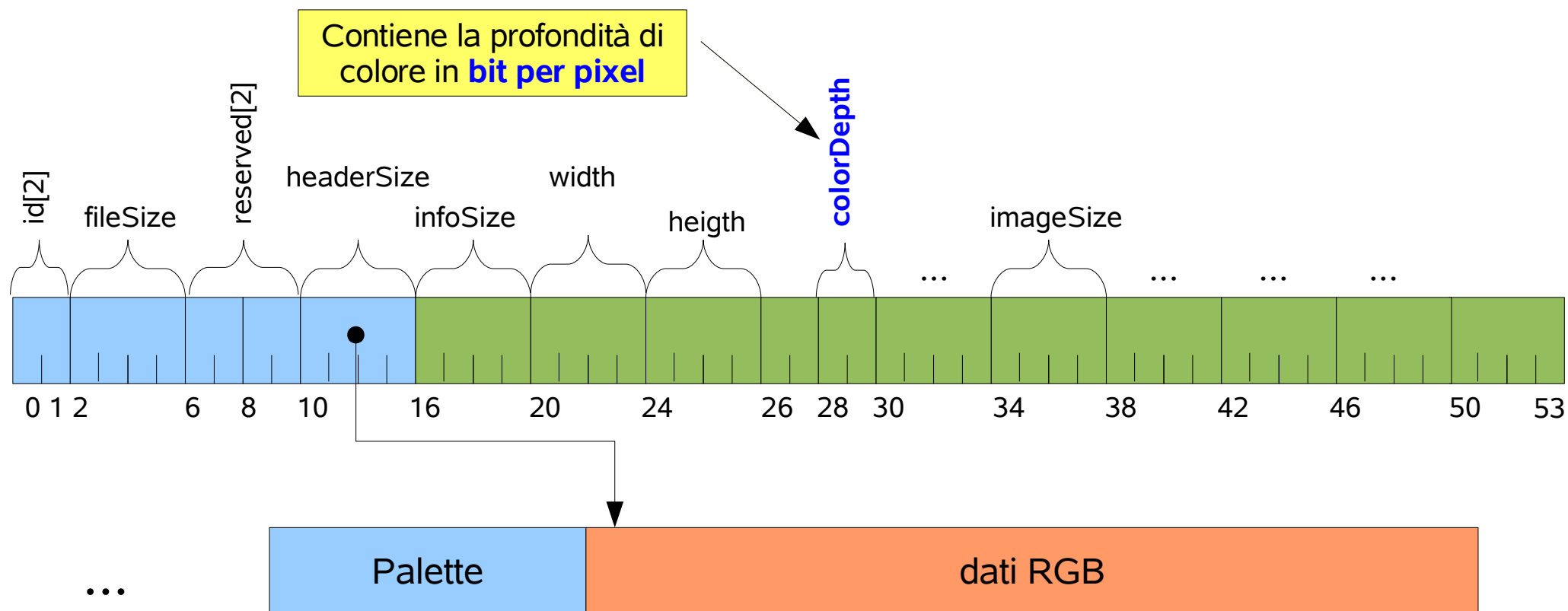
Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	<u>03</u>	<u>00</u>	<u>00</u>	<u>00</u>	<u>04</u>	<u>00</u>	<u>00</u>	<u>00</u>	01	00	18	00	00	00	00
00000020	00	00	30	00	00	00	13	0b	00	00	13	0b	00	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	ff	00	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff	ff
00000060	ff	ff	ff	00	00	00											

Larghezza e altezza dell'immagine (in pixel)

Formato del file Bitmap (.bmp)



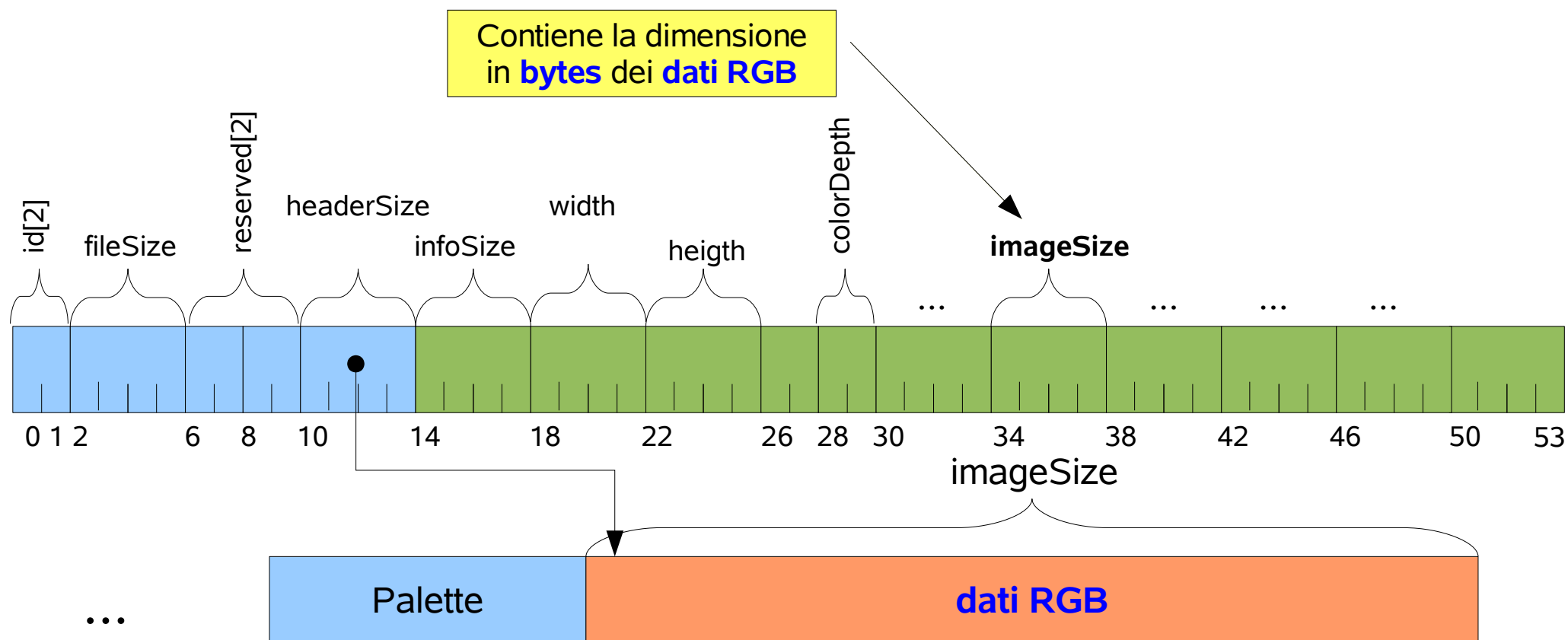
Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	03	00	00	00	04	00	00	00	01	00	<u>18</u>	<u>00</u>	00	00
00000020	00	00	30	00	00	00	13	0b	00	00	13	0b	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	00	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff
00000060	ff	ff	ff	00	00	00										

Contiene la profondità di
colore in **bit per pixel**:
0x18=24

Formato del file Bitmap (.bmp)



Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	03	00	00	00	04	00	00	00	01	00	18	00	00	00
00000020	00	00	<u>30</u>	<u>00</u>	<u>00</u>	<u>00</u>	13	0b	00	00	13	0b	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	ff	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff
00000060	ff	ff	ff	00	00	00										

Contiene la dimensione
in **bytes** dei **dati RGB**
0x30=48

I dati RGB (la parte arancione) è
formata da 48 bytes.

Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	03	00	00	00	04	00	00	00	01	00	18	00	00	00
00000020	00	00	<u>30</u>	<u>00</u>	<u>00</u>	<u>00</u>	13	0b	00	00	13	0b	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	00	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff
00000060	ff	ff	ff	00	00	00										

Contiene la dimensione
in **bytes** dei **dati RGB**
0x30=48

**Ma se sono 12 pixel in tutto e
ogni pixel occupa 3 bytes
perchè i dati RGB non sono
in totale 36 bytes?**

Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	03	00	00	00	04	00	00	00	01	00	18	00	00	00
00000020	00	00	<u>30</u>	<u>00</u>	<u>00</u>	<u>00</u>	13	0b	00	00	13	0b	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	00	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff
00000060	ff	ff	ff	00	00	00										

ff	ff	ff	ff	ff	ff	00	00	00
00	00	00	ff	ff	ff	cd	cf	2c
ff	ff	ff	00	00	00	ff	ff	ff
00	00	00	ff	ff	ff	ff	ff	ff

9 bytes (non è un multiplo di 4)

**Ogni riga
dell'immagine deve
occupare un
numero di bytes
multiplo di 4**

Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

**Vengono aggiunti dei bytes
fittizi per aggiustare
dimensione della riga la riga**

00000030	00 00 00	00 00 36	00 00 00	28 00
00000040	00 00 00	00 00 01	00 18 00	00 00
00000050	ff 00 00	00 00 13	0b 00 00	00 00
00000060	ff ff ff	00 00 00	ff ff ff	00 00 00

ff ff ff	ff ff ff	00 00 00	00 00 00
00 00 00	ff ff ff	cd cf 2c	00 00 00
ff ff ff	00 00 00	ff ff ff	00 00 00
00 00 00	ff ff ff	ff ff ff	00 00 00

12 bytes (multiplo di 4)

Formato del file Bitmap (.bmp)

- Esempio di un file di 4 righe e 3 colonne:

00000000	42	4d	66	00	00	00	00	00	00	00	36	00	00	00	28	00
00000010	00	00	03	00	00	00	04	00	00	00	01	00	18	00	00	00
00000020	00	00	<u>30</u>	<u>00</u>	<u>00</u>	<u>00</u>	13	0b	00	00	13	0b	00	00	00	00
00000030	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	00	00	00	00
00000040	00	00	00	00	00	ff	ff	ff	cd	cf	2c	00	00	00	ff	ff
00000050	ff	00	00	00	ff	ff	ff	00	00	00	00	00	00	ff	ff	ff
00000060	ff	ff	ff	00	00	00										

ff	ff	ff	ff	ff	ff	00	00	00	00	00	00
00	00	00	ff	ff	ff	cd	cf	2c	00	00	00
ff	ff	ff	00	00	00	ff	ff	ff	00	00	00
00	00	00	ff	ff	ff	ff	ff	ff	00	00	00

12 bytes (multiplo di 4)

Formato del file Bitmap (.bmp)

- Il problema si complica se consideriamo lunghezze di riga che fanno avanzare un numero di bytes pari a 1 o 2 (es. 5 pixel per riga corrispondono a 15 bytes che faranno occupare in realtà 16 bytes ad ogni riga)

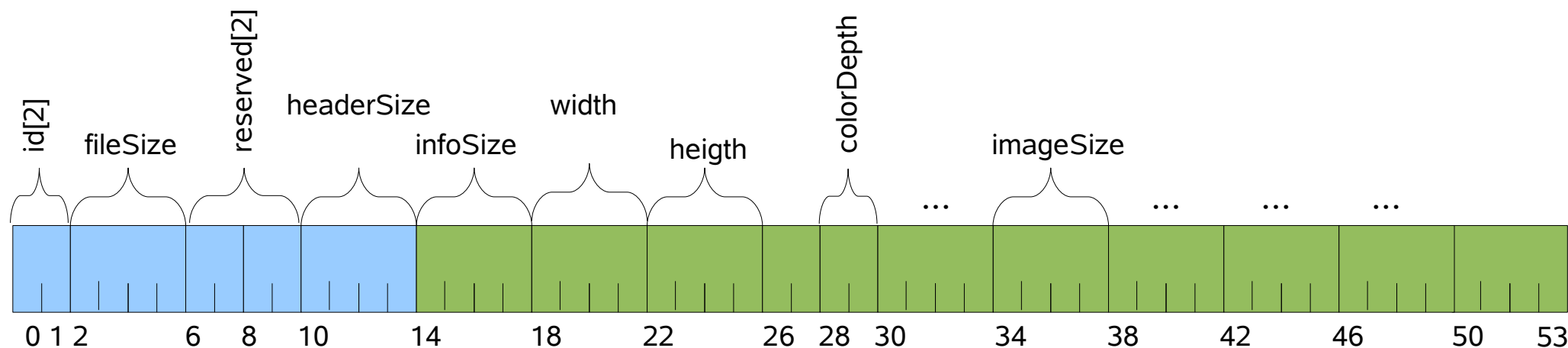
ff	ff	ff	ff	ff	ff	00	00	00	00	bb	00	01	10	aa	00
00	00	00	ff	ff	ff	cd	cf	2c	aa	42	09	20	24	bf	00
ff	ff	ff	00	00	00	ff	ff	ff	21	ae	f0	34	32	fe	00
00	00	00	ff	ff	ff	ff	ff	ff	cc	00	00	5f	10	ca	00

16 bytes (multiplo di 4)

Formato del file Bitmap (.bmp)

- Ricapitolando:
 - I dati RGB vengono allocati per riga
 - **Ogni riga ha una dimensione in bytes multipla di 4**
 - Se il numero di pixel per riga non è multiplo di 4 allora ci saranno dei bytes non utilizzati in fondo ad ogni riga
 - Ciò rende difficoltoso la mappatura dei dati RGB su un matrice

Formato del file Bitmap (.bmp)



```
struct {  
    char id[2];  
    unsigned long fileSize;  
    short int reserved[2];  
    unsigned long headerSize;
```

```
...
```

```
...
```

```
    unsigned long infoSize;  
    unsigned long width;  
    unsigned long height;  
    short int biPlanes;  
    short int colorDepth;  
    unsigned long compression;  
    unsigned long imageSize;  
    long horRes;  
    long verRes;  
    unsigned long colorsNumber;  
    unsigned long importantColor;
```

```
}
```

Formato del file Bitmap (.bmp)

```
typedef
struct {
    char id[2];
    unsigned long fileSize;
    short int reserved[2];
    unsigned long headerSize;
    unsigned long infoSize;
    unsigned long width;
    unsigned long height;
    short int biPlanes;
    short int colorDepth;
    unsigned long compression;
    unsigned long imageSize;
    long horRes;
    long verRes;
    unsigned long colorsNumber;
    unsigned long importantColor;
} BMPHEADERINFO
```

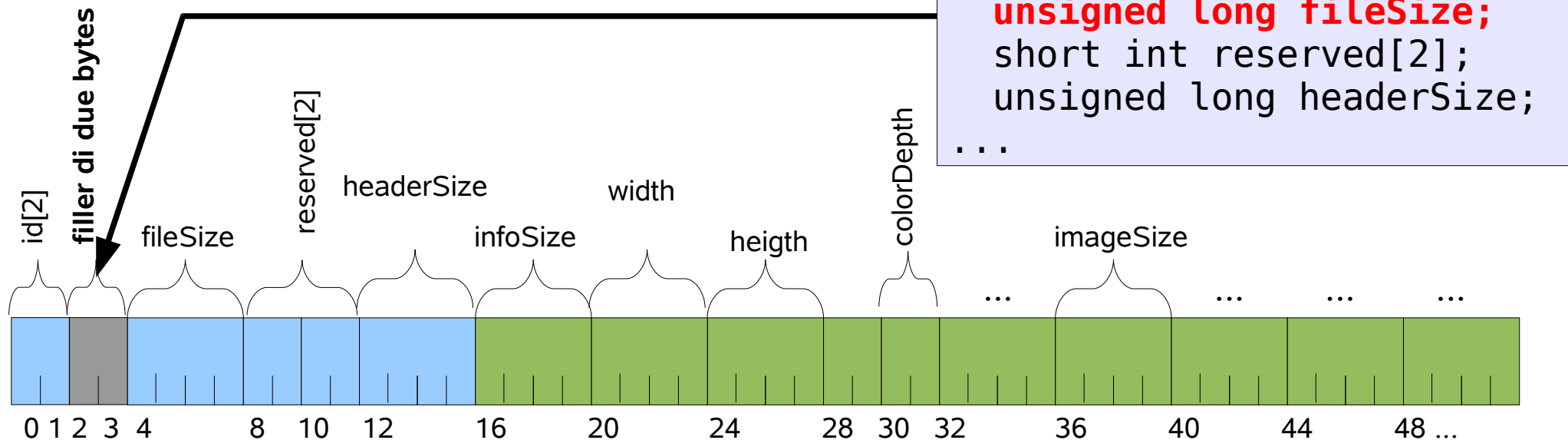
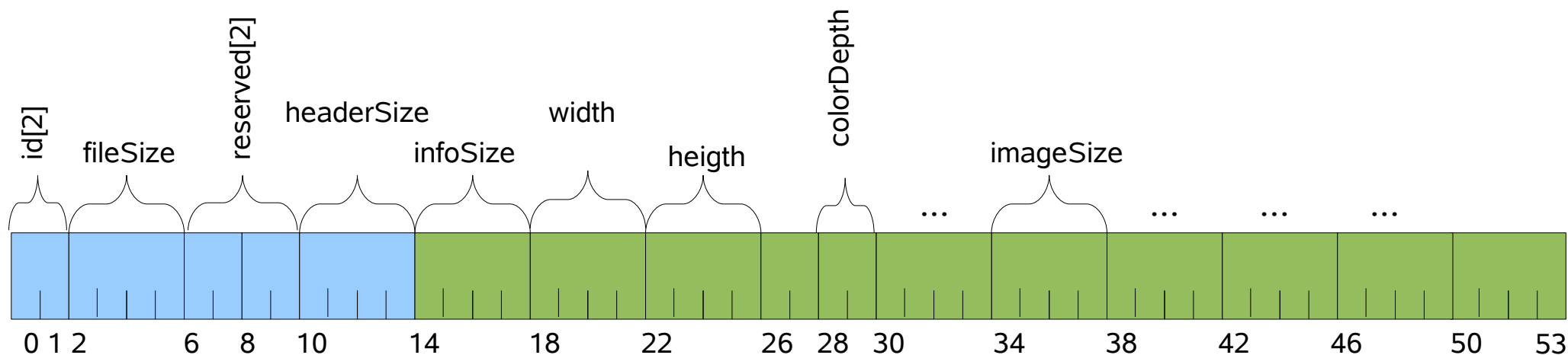
- E' necessario mantenere la **corrispondenza** tra le lunghezze:
 - **long** occupa 4 bytes (anche l'unsigned)
 - **short int** occupa 2 bytes
- Il problema è **l'allineamento** dei dati in RAM

Formato del file Bitmap (.bmp)

```
typedef
struct {
    char id[2];
    unsigned long fileSize;
    short int reserved[2];
    unsigned long headerSize;
    unsigned long infoSize;
    unsigned long width;
    unsigned long height;
    short int biPlanes;
    short int colorDepth;
    unsigned long compression;
    unsigned long imageSize;
    long horRes;
    long verRes;
    unsigned long colorsNumber;
    unsigned long importantColor;
} BMPHEADERINFO
```

- Un dato da 4 bytes viene allocato su un indirizzo multiplo di 4
- Un dato da 2 bytes viene allocato su un indirizzo multiplo di 2
- Si possono quindi avere dei **buchi** tra i campi

Formato del file Bitmap (.bmp)



Formato del file Bitmap (.bmp)

```
int dist(void *p1, void*p2){  
    return p1-p2;  
}
```

Si può verificare stampando la distanza di ogni campo dall'inizio della struttura e la dimensione di ogni campo, con un programma di prova

```
main(int argc, char *argv[]){  
    BMPHEADERINFO b;  
    printf("%15s %4d%4d\n", "id[0]", dist(&b.id[0], &b), sizeof(b.id[0]));  
    printf("%15s %4d%4d\n", "id[1]", dist(&b.id[1], &b), sizeof(b.id[1]));  
    printf("%15s %4d%4d\n", "fileSize", dist(&b.fileSize, &b), sizeof(b.fileSize));  
    printf("%15s %4d%4d\n", "reserved", dist(&b.reserved[0], &b), sizeof(b.reserved[0]));  
    printf("%15s %4d%4d\n", "reserved", dist(&b.reserved[1], &b), sizeof(b.reserved[1]));  
    printf("%15s %4d%4d\n", "headerSize", dist(&b.headerSize, &b), sizeof(b.headerSize));  
    printf("%15s %4d%4d\n", "infoSize", dist(&b.infoSize, &b), sizeof(b.infoSize));  
    printf("%15s %4d%4d\n", "width", dist(&b.width, &b), sizeof(b.width));  
    printf("%15s %4d%4d\n", "height", dist(&b.height, &b), sizeof(b.height));  
    printf("%15s %4d%4d\n", "biPlanes", dist(&b.biPlanes, &b), sizeof(b.biPlanes));  
    printf("%15s %4d%4d\n", "colorDepth", dist(&b.colorDepth, &b), sizeof(b.colorDepth));  
    printf("%15s %4d%4d\n", "compression", dist(&b.compression, &b), sizeof(b.compression));  
    printf("%15s %4d%4d\n", "imageSize", dist(&b.imageSize, &b), sizeof(b.imageSize));  
    printf("%15s %4d%4d\n", "horRes", dist(&b.horRes, &b), sizeof(b.horRes));  
    printf("%15s %4d%4d\n", "verRes", dist(&b.verRes, &b), sizeof(b.verRes));  
    printf("%15s %4d%4d\n", "colorsNumber", dist(&b.colorsNumber, &b),  
           sizeof(b.colorsNumber));  
    printf("%15s %4d%4d\n", "importantColor", dist(&b.importantColor, &b),  
           sizeof(b.importantColor));  
}
```

Formato del file Bitmap (.bmp)

id[0]	0	1
id[1]	1	1
fileSize	4	4
reserved	8	2
reserved	10	2
headerSize	12	4
infoSize	16	4
width	20	4
height	24	4
biPlanes	28	2
colorDepth	30	2
compression	32	4
imageSize	36	4
horRes	40	4
verRes	44	4
colorsNumber	48	4
importantColor	52	4

Se i campi fossero allocati di seguito **sommando la seconda e la terza colonna** di ogni riga si dovrebbe ottenere **la seconda colonna della riga successiva**.

Il risultato indica che il campo **fileSize** invece di iniziare al byte 2 della struttura inizia al byte 4.

Formato del file Bitmap (.bmp)

```
typedef
struct {
    char id[2];
    unsigned long fileSize;
    short int reserved[2];
    unsigned long headerSize;
    unsigned long infoSize;
    unsigned long width;
    unsigned long height;
    short int biPlanes;
    short int colorDepth;
    unsigned long compression;
    unsigned long imageSize;
    long horRes;
    long verRes;
    unsigned long colorsNumber;
    unsigned long importantColor;
} __attribute__((__packed__)) BMPHEADERINFO
```

- Dichiarando la struttura **PACKED** si compattano tutti i campi evitando la formazione dei buchi.

Attributo PACKED della struttura

Formato del file Bitmap (.bmp)

id[0]	0	1
id[1]	1	1
fileSize	2	4
reserved	6	2
reserved	8	2
headerSize	10	4
infoSize	14	4
width	18	4
height	22	4
biPlanes	26	2
colorDepth	28	2
compression	30	4
imageSize	34	4
horRes	38	4
verRes	42	4
colorsNumber	46	4
importantColor	50	4

La stampa del programma di prova fornisce il risultato adeguato.