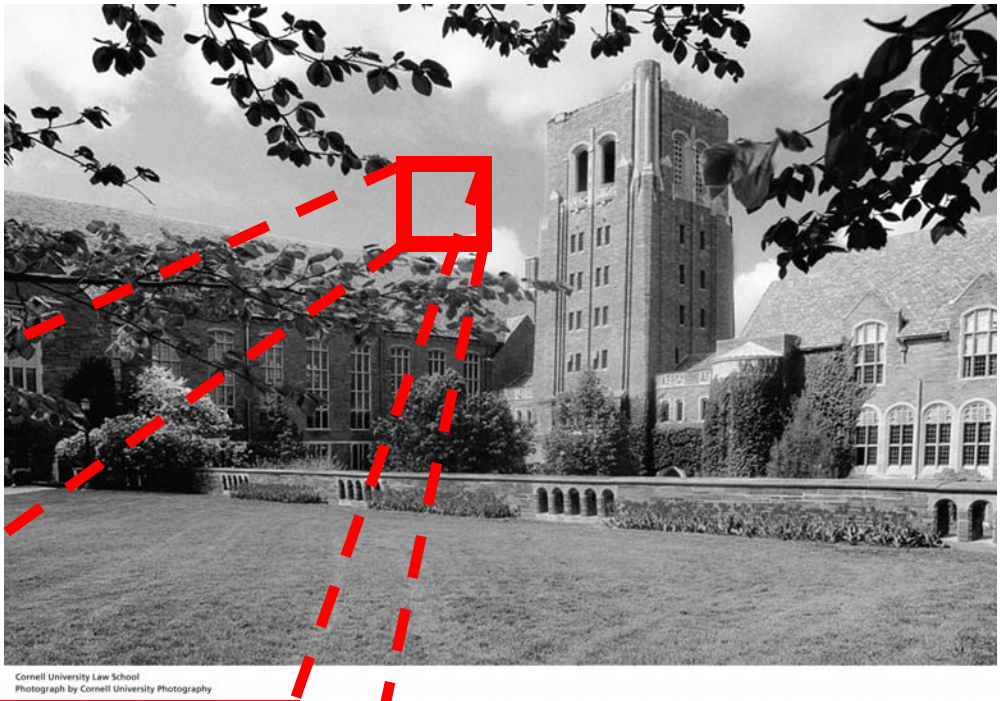
- Previous Lecture:
  - Transition probability—an example on matrices
  - Contour plot


- Today's Lecture:
  - Working with images


- Announcements:
  - Section this week in the UP B7 lab
  - Project 4 due 10/22 (Thursday)

A picture as a matrix

1458-by-2084



150	149	152	153	152	155
151	150	153	154	153	156
153	151	155	156	155	158
154	153	156	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162

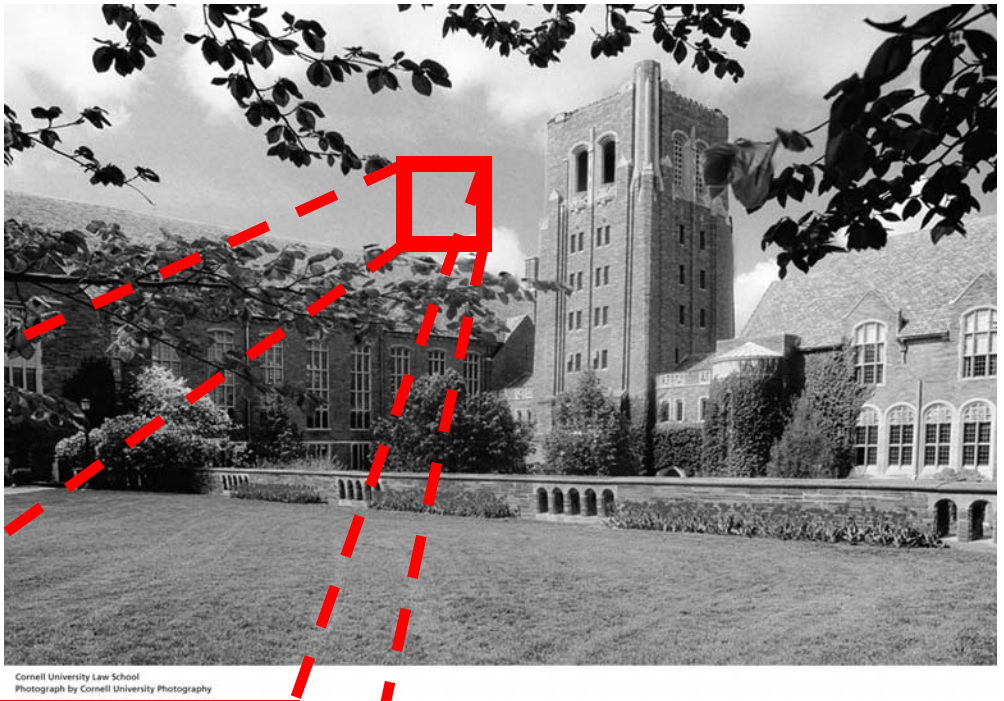
Images can be encoded in different ways

- Common formats include
  - JPEG: Joint Photographic Experts Group
  - GIF: Graphics Interchange Format
- Data are compressed
- We will work with jpeg files:
  - **`imread`**: read a .jpg file and convert it to a "normal numeric" array that we can work with
  - **`imwrite`**: write an array into a .jpg file (compressed data)

Grayness: a value in [0..255]

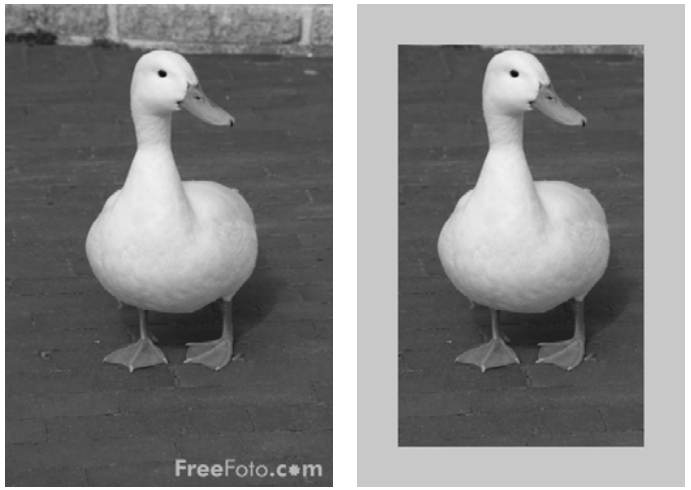
0 = black
255 = white

These are *integer* values
Type: **`uint8`**



150	149	152	153	152	155
151	150	153	154	153	156
153	151	155	156	155	158
154	153	156	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162

Let's put a picture in a frame



Things to do:

1. Read `bwduck.jpg` from memory and convert it into an array
2. Show the original picture
3. Assign a gray value (frame color) to the "edge pixels"
4. Show the manipulated picture

Reading a jpeg file and displaying the image

```
% Read jpg image and convert to
% an array P
   P = imread('bwduck.jpg');

% Show the data in 3-d array P as
% an image
   imshow(P)
```

```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color




imshow(P)
```

```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200;  % light gray
```

```
imshow(P)
```

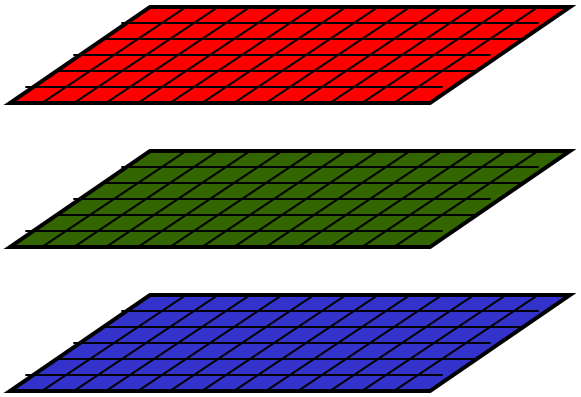
```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200;  % light gray
[nr,nc]= size(P);
for r= 1:nr
  for c= 1:nc
    % At pixel (r,c)




  end
end
imshow(P)
```

```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200;  % light gray
[nr,nc]= size(P);
for r= 1:nr
  for c= 1:nc
    % At pixel (r,c)
    if  r<=width || r>nr-width || ...
        c<=width || c>nc-width
      P(r,c)= frameColor;
    end
  end
end
imshow(P)
```

Things to consider…
1. Can we be more efficient?
2. What is the type of the values in P?

A color picture is made up of RGB matrices

Color image



3-d Array

$$0 \le A(i,j,1) \le 255$$

$$0 \le A(i,j,2) \le 255$$

$$0 \le A(i,j,3) \le 255$$

Operations on images amount to operations on matrices!

Example: Mirror Image

`LawSchool.jpg`

`LawSchoolMirror.jpg`

Solution Framework

1. Read `LawSchool.jpg` from memory and convert it into an array.
2. Manipulate the Array.
3. Convert the array to a jpg file and write it to memory.

Reading and writing jpg files

```
% Read jpg image and convert to
% a 3D array A
  A = imread('LawSchool.jpg');

% Write 3D array B to memory as
% a jpg image
  imwrite(B,'LawSchoolMirror.jpg')
```

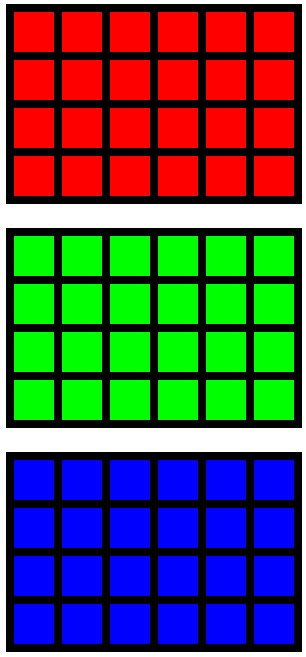
A 3-d array as 3 matrices

```
[nr, nc, np] = size(A)  % dimensions of 3-d array A
```

#rows

#columns

#layers (pages)

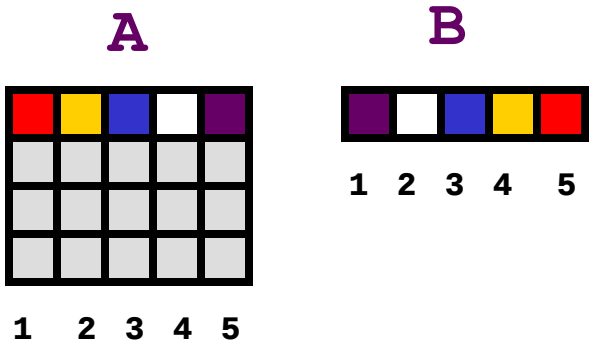


4-by-6 **`M1= A(:,:,1)`**

4-by-6 **`M2= A(:,:,2)`**

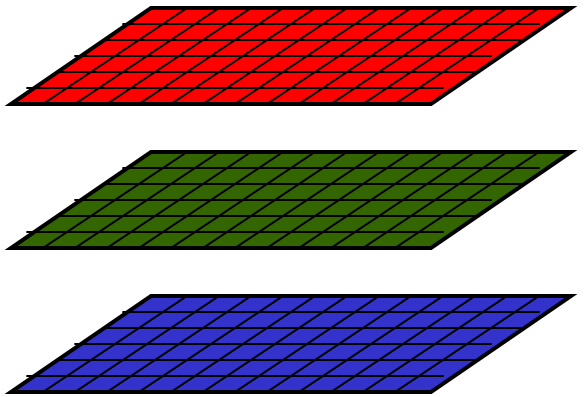
4-by-6 **`M3= A(:,:,3)`**

```
%Store mirror image of A in array B

[nr,nc,np]= size(A);
for r= 1:nr
   for c= 1:nc

        B(r,c  )= A(r,nc-c+1  );

   end
end
```



```
%Store mirror image of A in array B

[nr,nc,np]= size(A);
for r= 1:nr
   for c= 1:nc
      for p= 1:np
         B(r,c,p)= A(r,nc-c+1,p);
      end
   end
end
```

```
[nr,nc,np]= size(A);
for r= 1:nr
  for c= 1:nc
    for p= 1:np
      B(r,c,p)= A(r,nc-c+1,p);
    end
  end
end
```



```
[nr,nc,np]= size(A);
for p= 1:np
  for r= 1:nr
    for c= 1:nc
      B(r,c,p)= A(r,nc-c+1,p);
    end
  end
end
```

Both fragments create a mirror image of **A** .

A true

B false

```
[nr,nc,np]= size(A);
for r= 1:nr
  for c= 1:nc
    for p= 1:np
      B(r,c,p)= A(r,nc-c+1,p);
    end
  end
end
```

This is non-vectorized code.

```
[nr,nc,np]= size(A);
for p= 1:np
  for r= 1:nr
    for c= 1:nc
      B(r,c,p)= A(r,nc-c+1,p);
    end
  end
end
```

Both fragments create a mirror image of **A** .

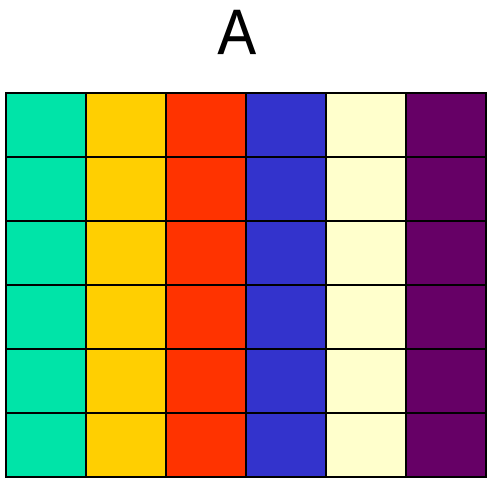
A true

B false

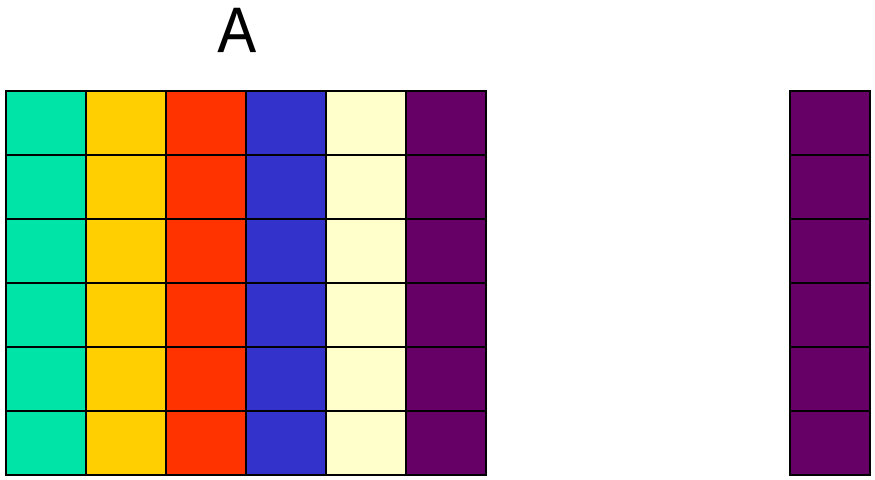
```
% Make mirror image of A -- the whole thing

A= imread('LawSchool.jpg');
[nr,nc,np]= size(A);




for r= 1:nr
  for c= 1:nc
    for p= 1:np
      B(r,c,p)= A(r,nc-c+1,p);
    end
  end
end
imshow(B)  % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')
```

```
% Make mirror image of A -- the whole thing

A= imread('LawSchool.jpg');
[nr,nc,np]= size(A);

B= zeros(nr,nc,np);
B= uint8(B);  % Type for image color values

for r= 1:nr
  for c= 1:nc
    for p= 1:np
      B(r,c,p)= A(r,nc-c+1,p);
    end
  end
end
imshow(B)  % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')
```

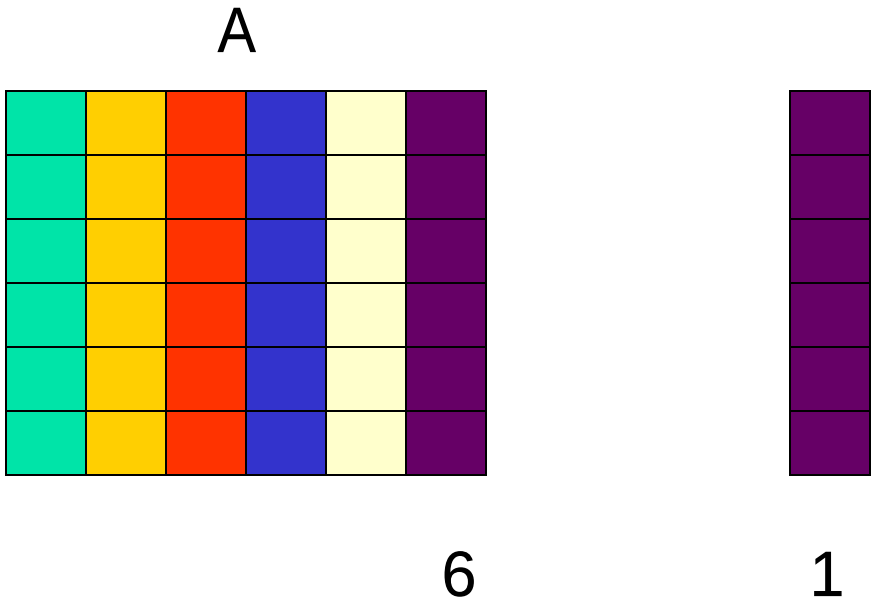
Vectorized code simplifies things…
Work with a whole column at a time



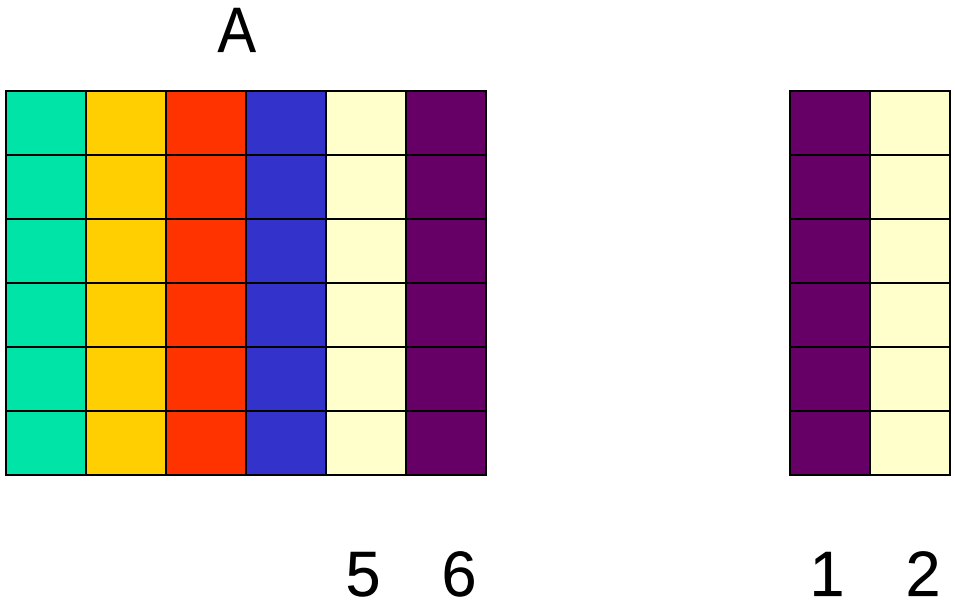
Vectorized code simplifies things…
Work with a whole column at a time



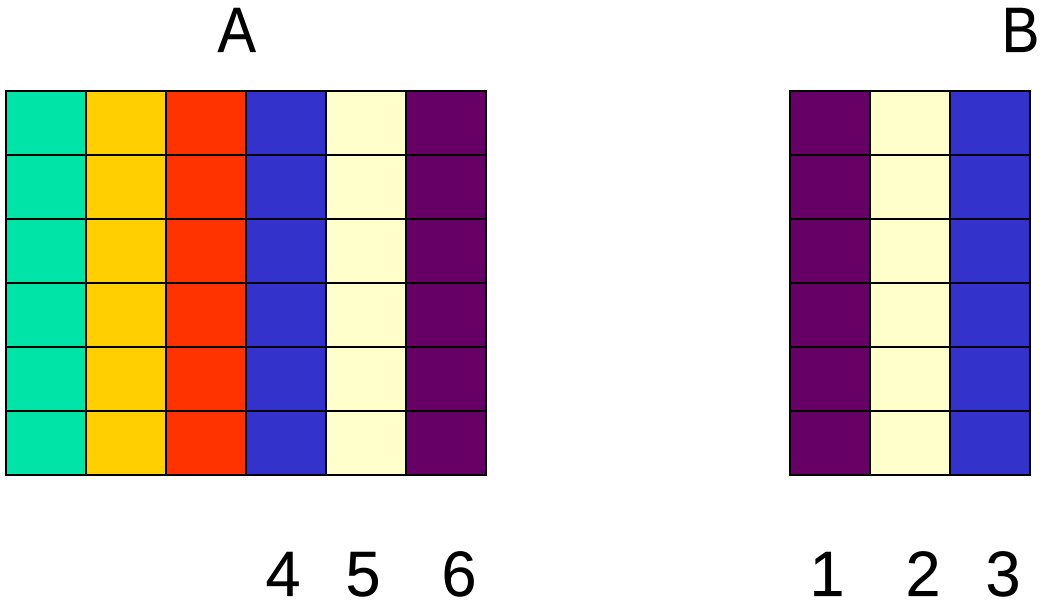
Vectorized code simplifies things…
Work with a whole column at a time



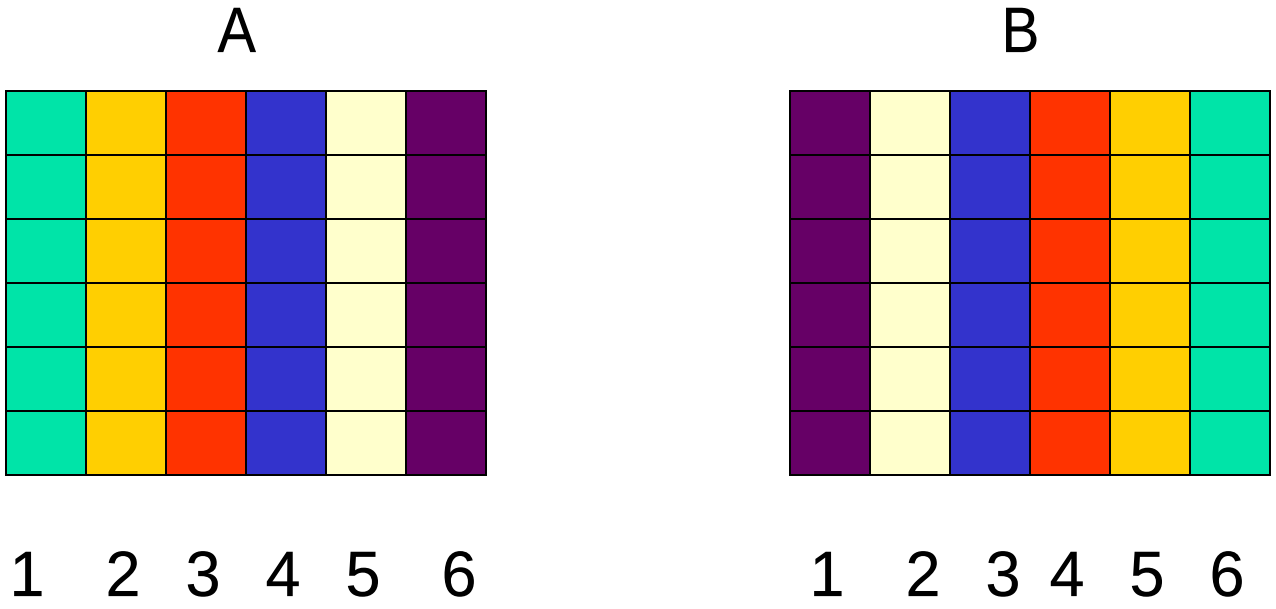
Vectorized code simplifies things…
Work with a whole column at a time



Vectorized code simplifies things…
Work with a whole column at a time



Vectorized code simplifies things…
Work with a whole column at a time



Column c in B
is column nc-c+1 in A

Accessing a submatrix

`M`

2	-1	.5	0	-3
3	8	6	7	7
5	-3	8.5	9	10
52	81	.5	7	2

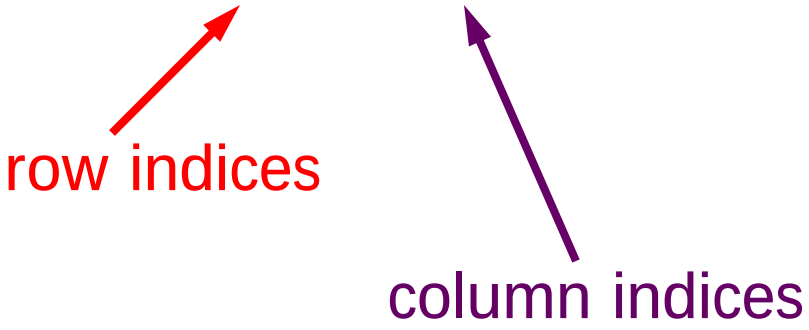
- `M` refers to the whole matrix
- `M(3,5)` refers to one component of `M`

Accessing a submatrix

`M`

2	-1	.5	0	-3
3	8	6	7	7
5	-3	8.5	9	10
52	81	.5	7	2

- `M` refers to the whole matrix
- `M(3,5)` refers to one component of `M`
- `M(2:3,3:5)` refers to a submatrix of `M`



Consider a single matrix (just one layer)

```
[nr,nc,np] = size(A);
for c= 1:nc
    B(all rows,c  ) = A(all rows,nc+1-c  );


end
```

Consider a single matrix (just one layer)

```
[nr,nc,np] = size(A);
for c= 1:nc
    B(1:nr,c ) = A(1:nr,nc+1-c );


end
```

Consider a single matrix (just one layer)

```
[nr,nc,np] = size(A);
for c= 1:nc
    B( : ,c  ) = A( : ,nc+1-c  );

end
```

The colon says "all indices in this dimension." In this case it says "all rows."

Now repeat for all layers

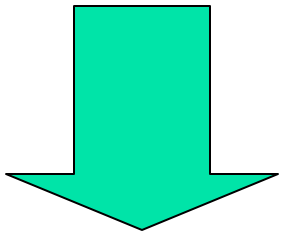
```
[nr,nc,np] = size(A);
for c= 1:nc
    B(:,c,1) = A(:,nc+1-c,1)
    B(:,c,2) = A(:,nc+1-c,2)
    B(:,c,3) = A(:,nc+1-c,3)
end
```

Vectorized code to create a mirror image

```
A = imread('LawSchool.jpg')
[nr,nc,np] = size(A);
for c= 1:nc
    B(:,c,1) = A(:,nc+1-c,1)
    B(:,c,2) = A(:,nc+1-c,2)
    B(:,c,3) = A(:,nc+1-c,3)
end
imwrite(B,'LawSchoolMirror.jpg')
```

Even more compact vectorized code to create a mirror image…

```
for c= 1:nc
    B(:,c,1) = A(:,nc+1-c,1)
    B(:,c,2) = A(:,nc+1-c,2)
    B(:,c,3) = A(:,nc+1-c,3)
end
```

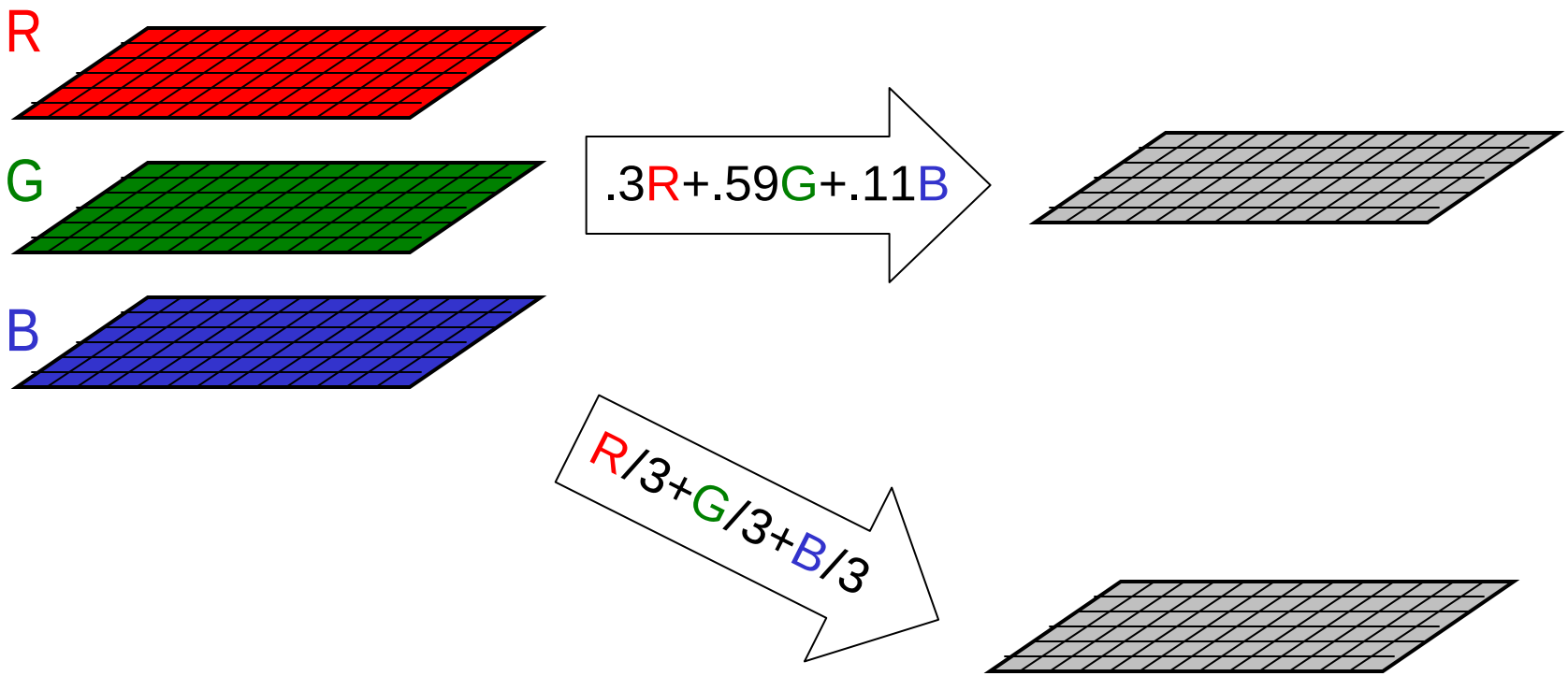


```
B = A(:,nc:-1:1,:)
```

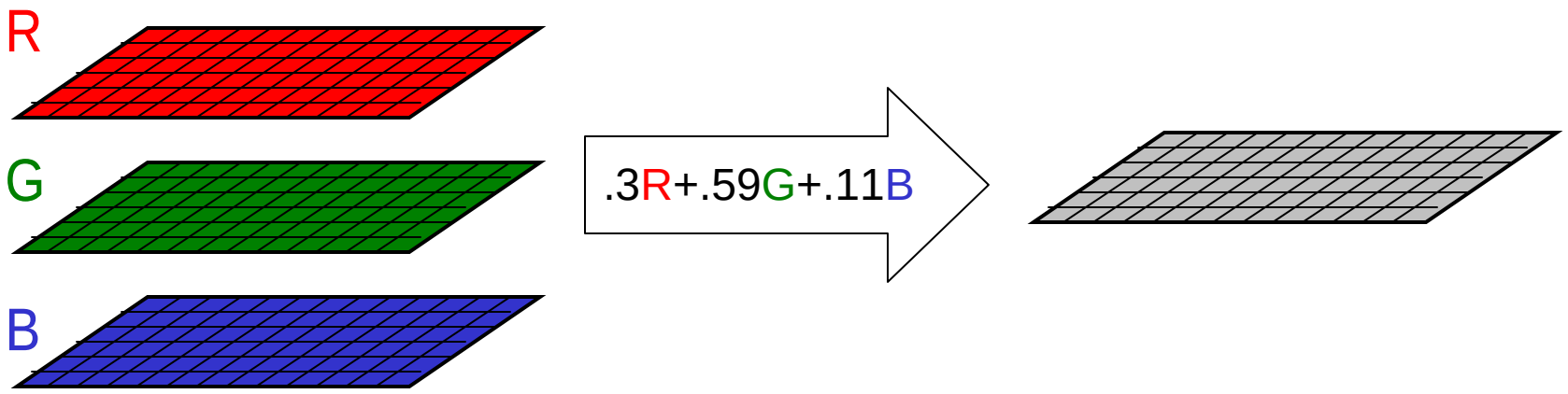
Example: color → black and white

Can "average" the three color values to get one gray value.

Averaging the RGB values to get a gray value



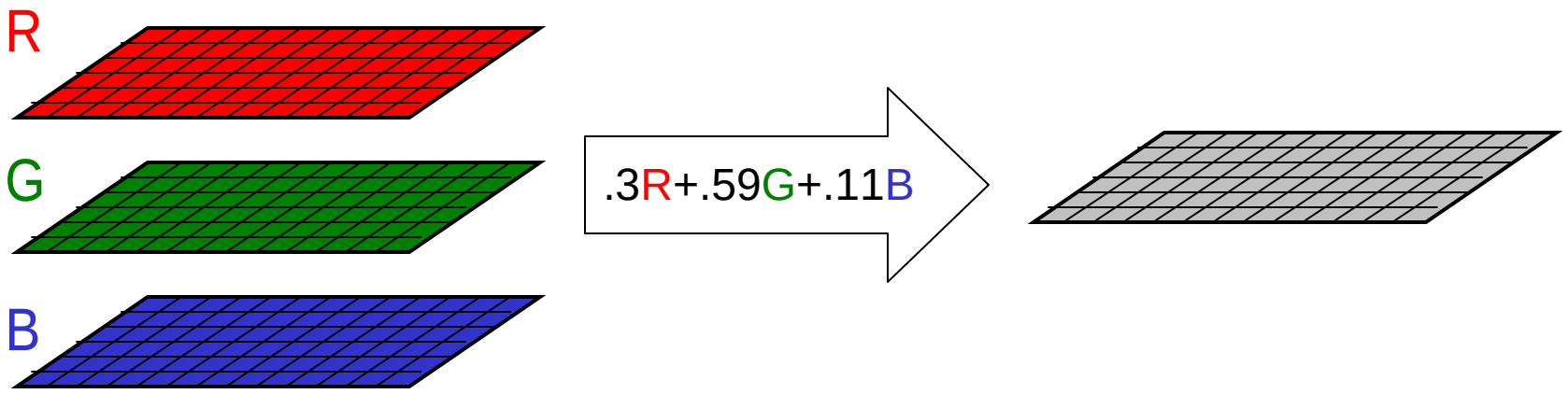
Averaging the RGB values to get a gray value




```
for  i= 1:m
    for j= 1:n
        M(i,j)= .3*R(i,j) + .59*G(i,j) + .11*B(i,j)
    end
end
```

scalar operation

Averaging the RGB values to get a gray value



M= .3*R + .59*G + .11*B

vectorized operation

Here are 2 ways to calculate the average. Are gray value matrices g and h the same given image data A?

```
for r= 1:nr
  for c= 1:nc
    g(r,c)= A(r,c,1)/3 + A(r,c,2)/3 ...
            A(r,c,3)/3;
    h(r,c)= ...
       ( A(r,c,1)+A(r,c,2)+A(r,c,3) )/3;
  end
end
```

A: yes

B: no

showToGrayscale.m

(result is a 3-d array where
the three layers are the same)

Matlab has a built-in function to convert from color to grayscale, resulting in a 2-d array:

B = rgb2gray(A)