Homework 8

due Wed Nov 9 at start of class

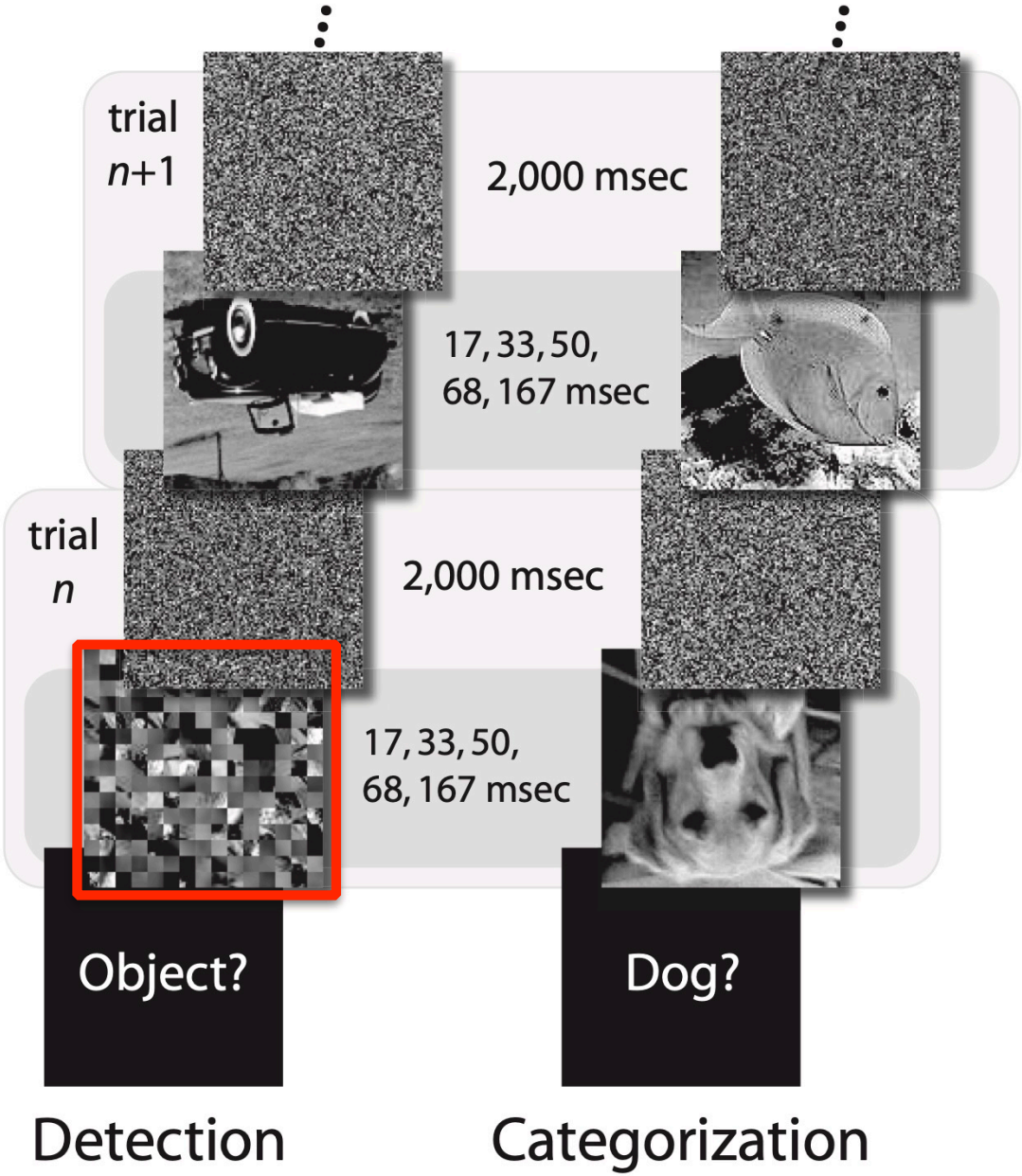
`Homework8.pdf`

`nashville.jpg`

Homework 8

- **Q1a** create a b/w version from a color version (using vectorized operations in numpy, not `for` loops)

Homework 8


trial
n+1
2,000 msec
17, 33, 50,
68, 167 msec
trial
n
2,000 msec
17, 33, 50,
68, 167 msec
scrambled
image
Object?
Dog?
Detection
Categorization

Homework 8

- **Q1b** scramble an (b/w) image

8x8 scramble grid

Homework 8

- **Q1b** scramble an (b/w) image

8x8 scramble grid

Homework 8

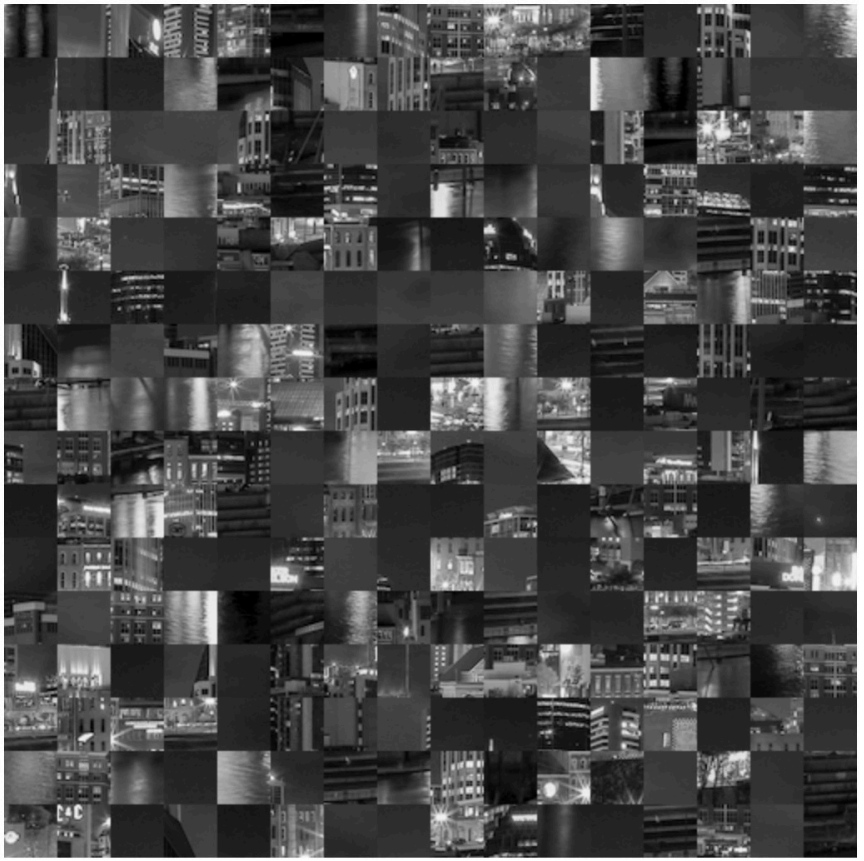
- **Q1b** scramble an (b/w) image

2x2 scramble grid

Homework 8

- **Q1b** scramble an (b/w) image

16x16 scramble grid



Homework 8

- lots of ways of doing this - I will give you a hint to one approach to try (you can do it another way)

Homework 8

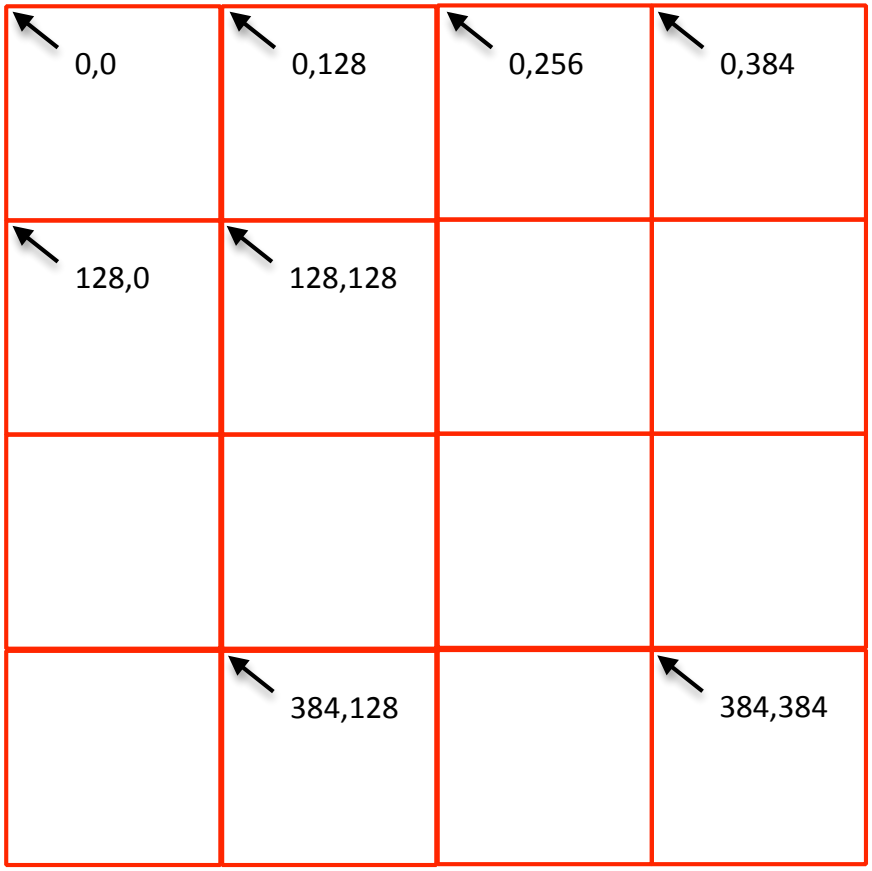
what are we trying to do?

512x512 image

use a 4x4 grid as an example

Homework 8

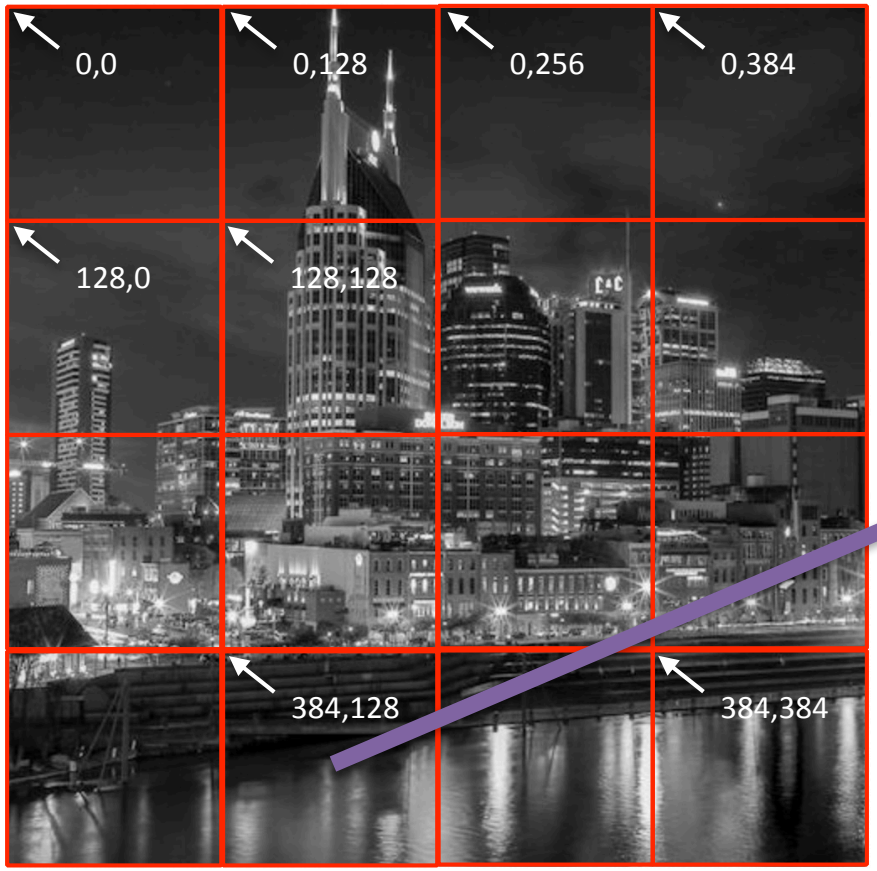
512x512 image

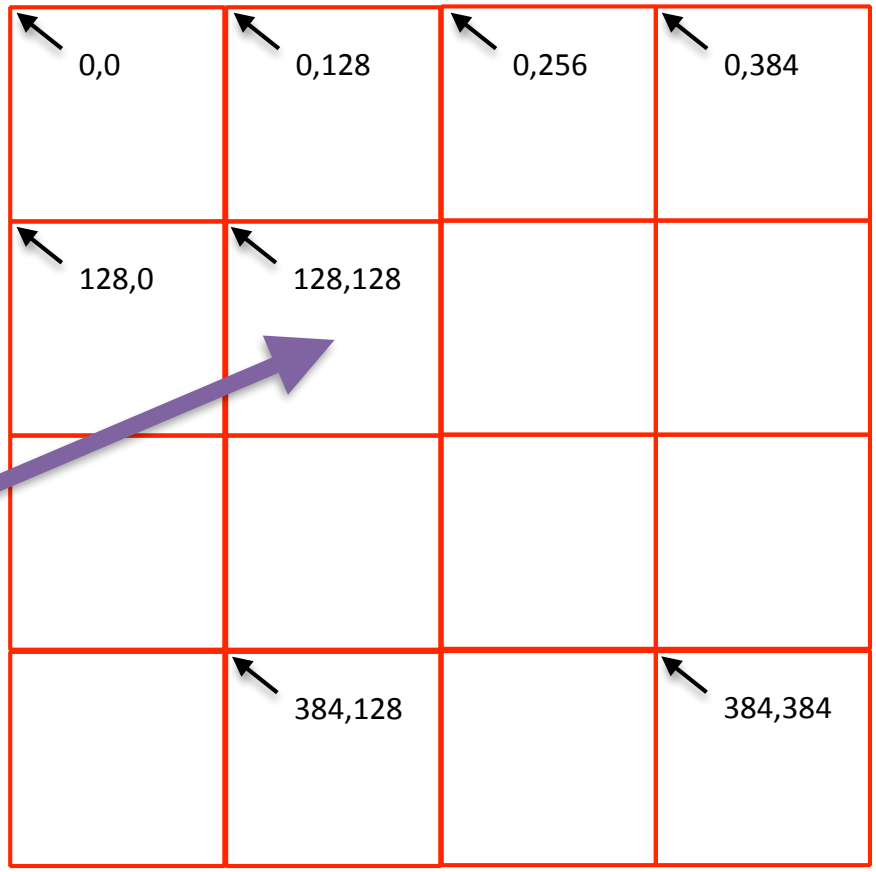


use a 4x4 grid as an example

Homework 8

512x512 image

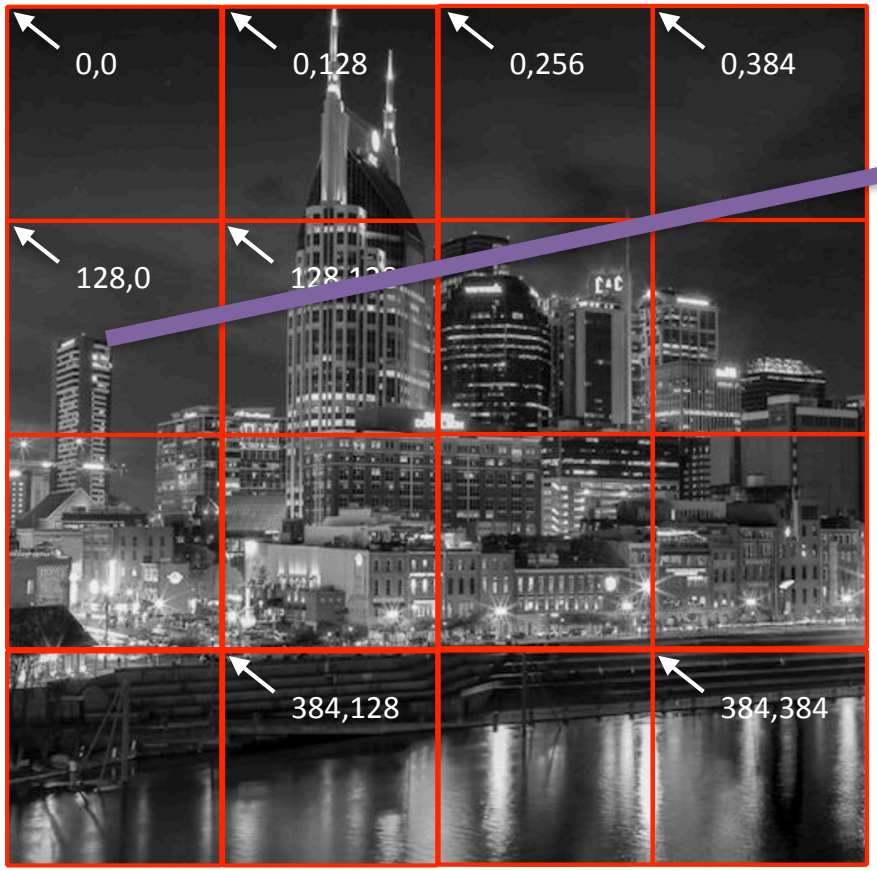


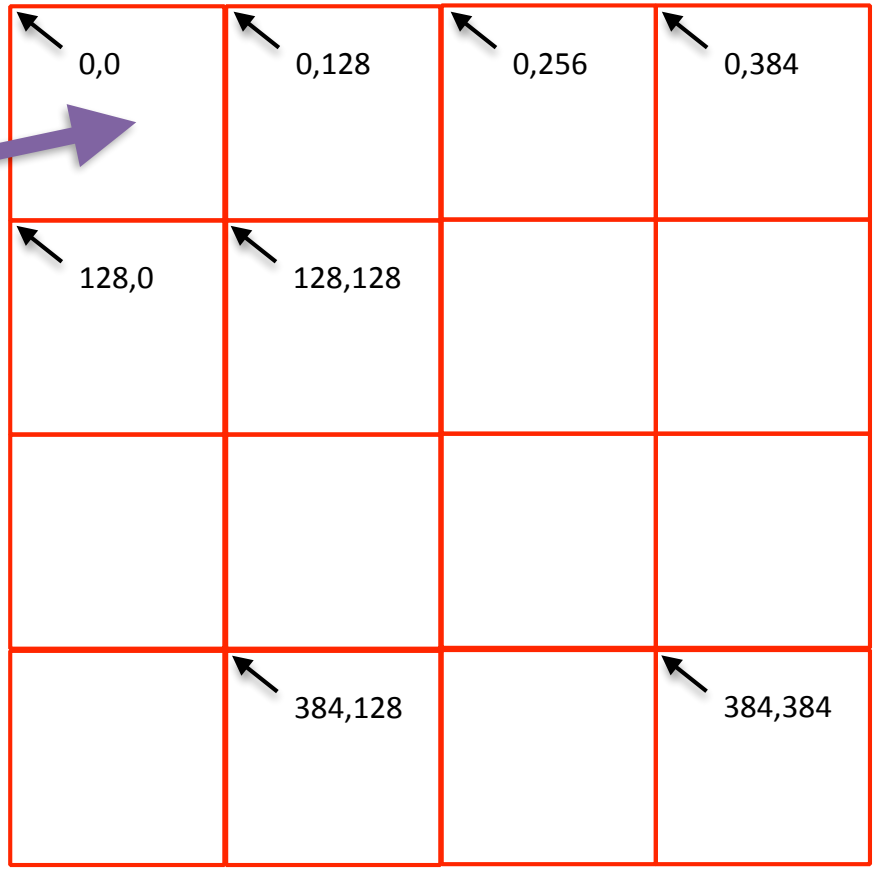


use a 4x4 grid as an example

Homework 8

512x512 image

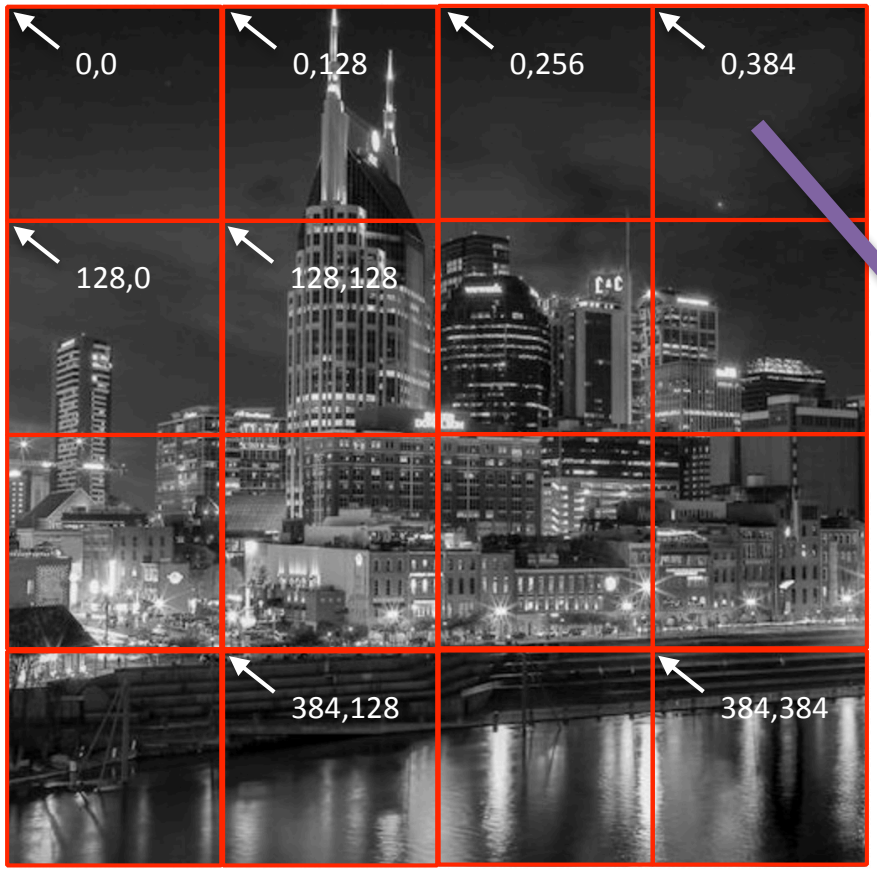


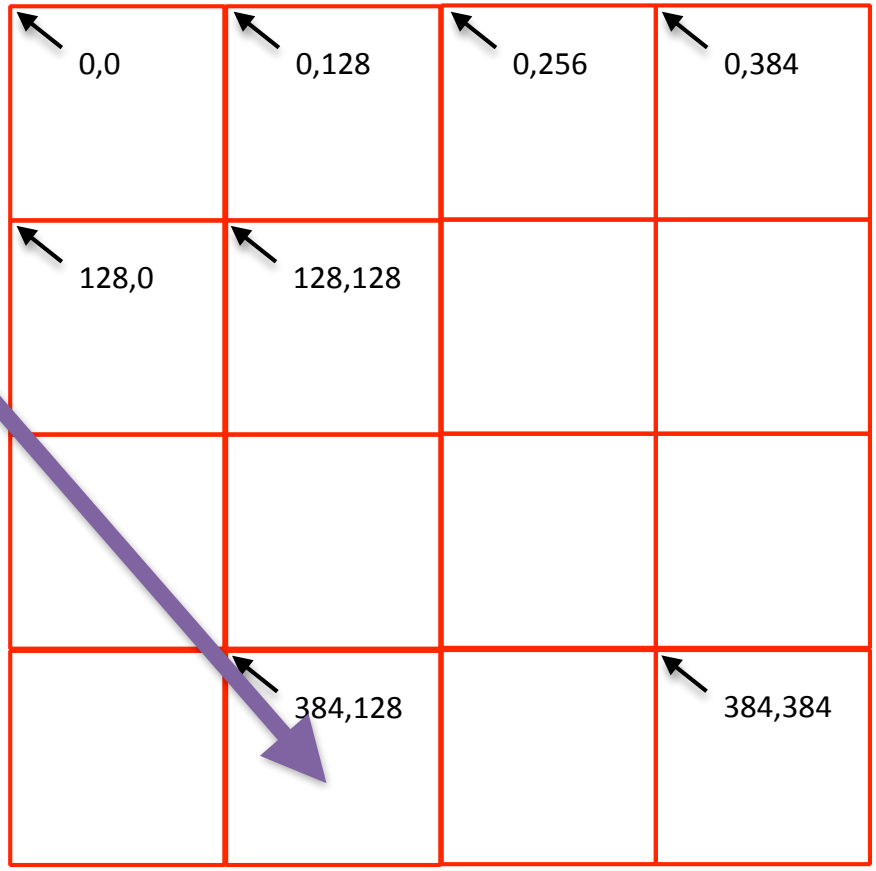


use a 4x4 grid as an example

Homework 8

512x512 image





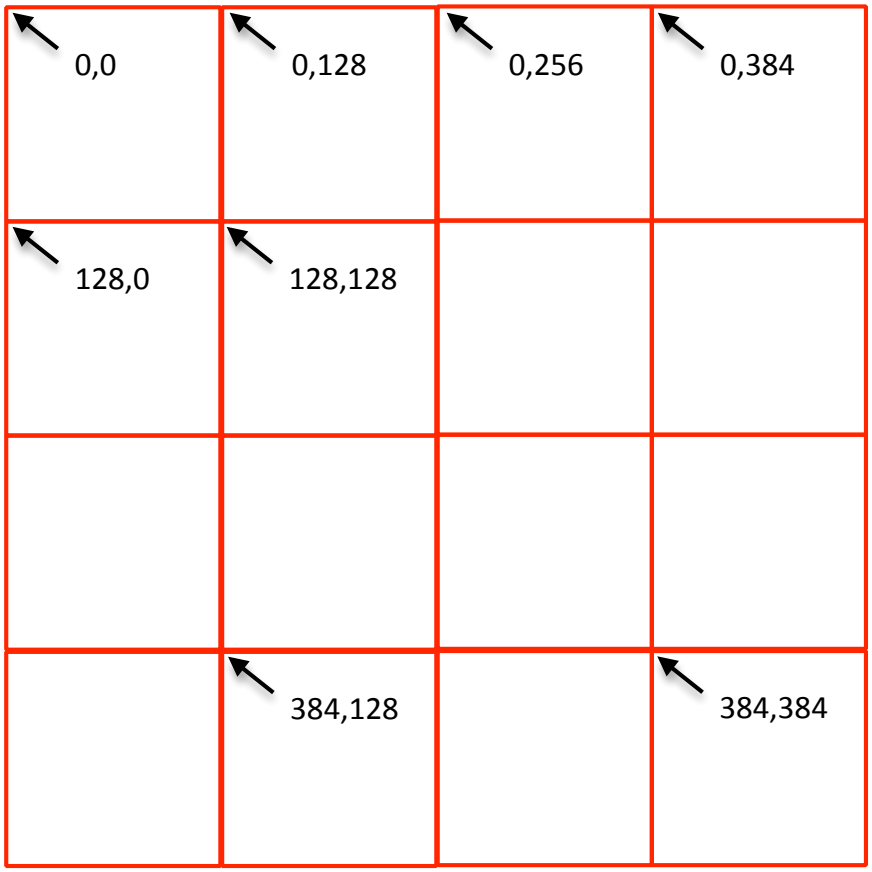
use a 4x4 grid as an example

Homework 8

scramble (permute) the coordinates of the 16 squares

fill in a blank (scrambled) numpy array

512x512 image

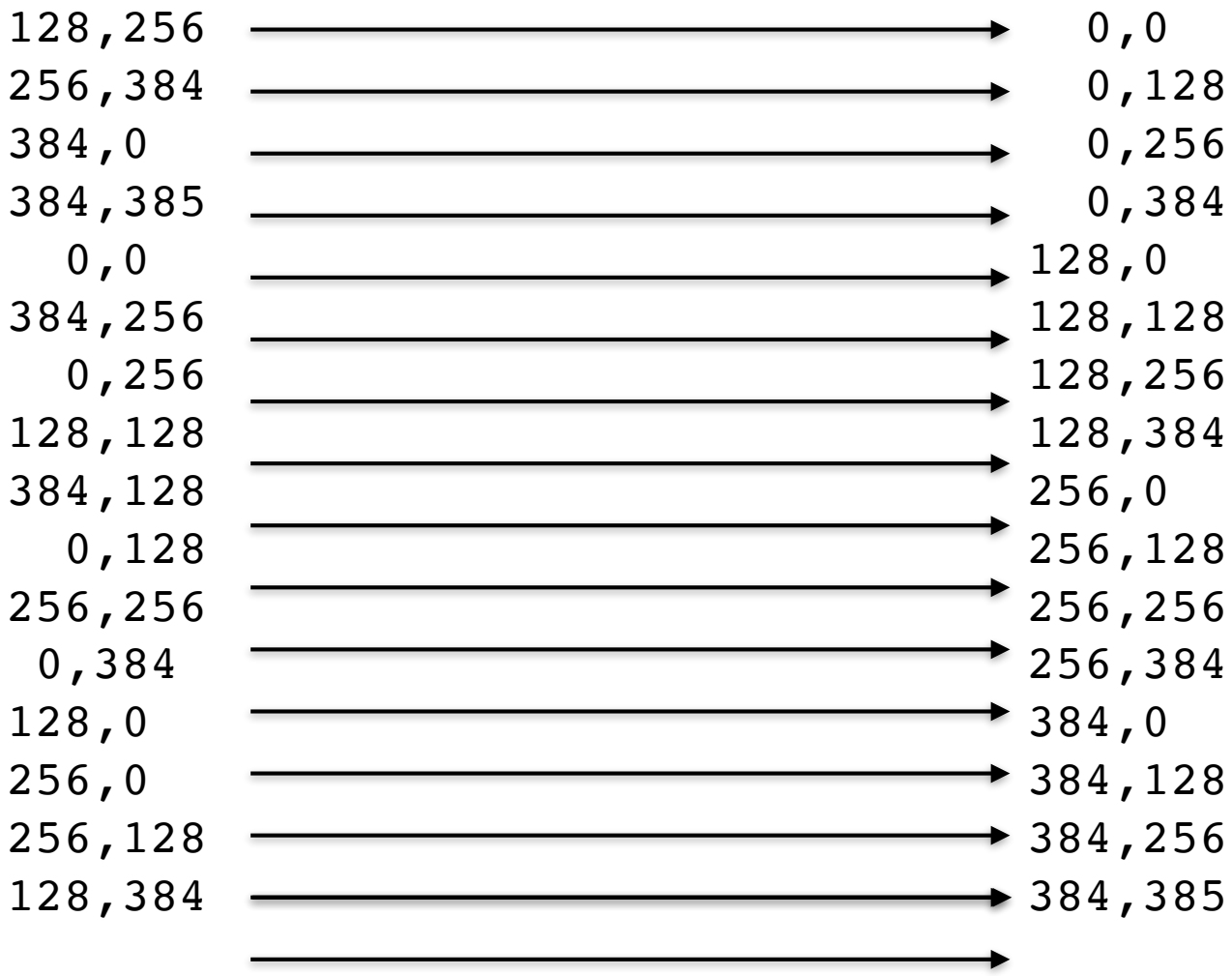


use a 4x4 grid as an example

Homework 8

scramble (permute) the coordinates of the 16 square

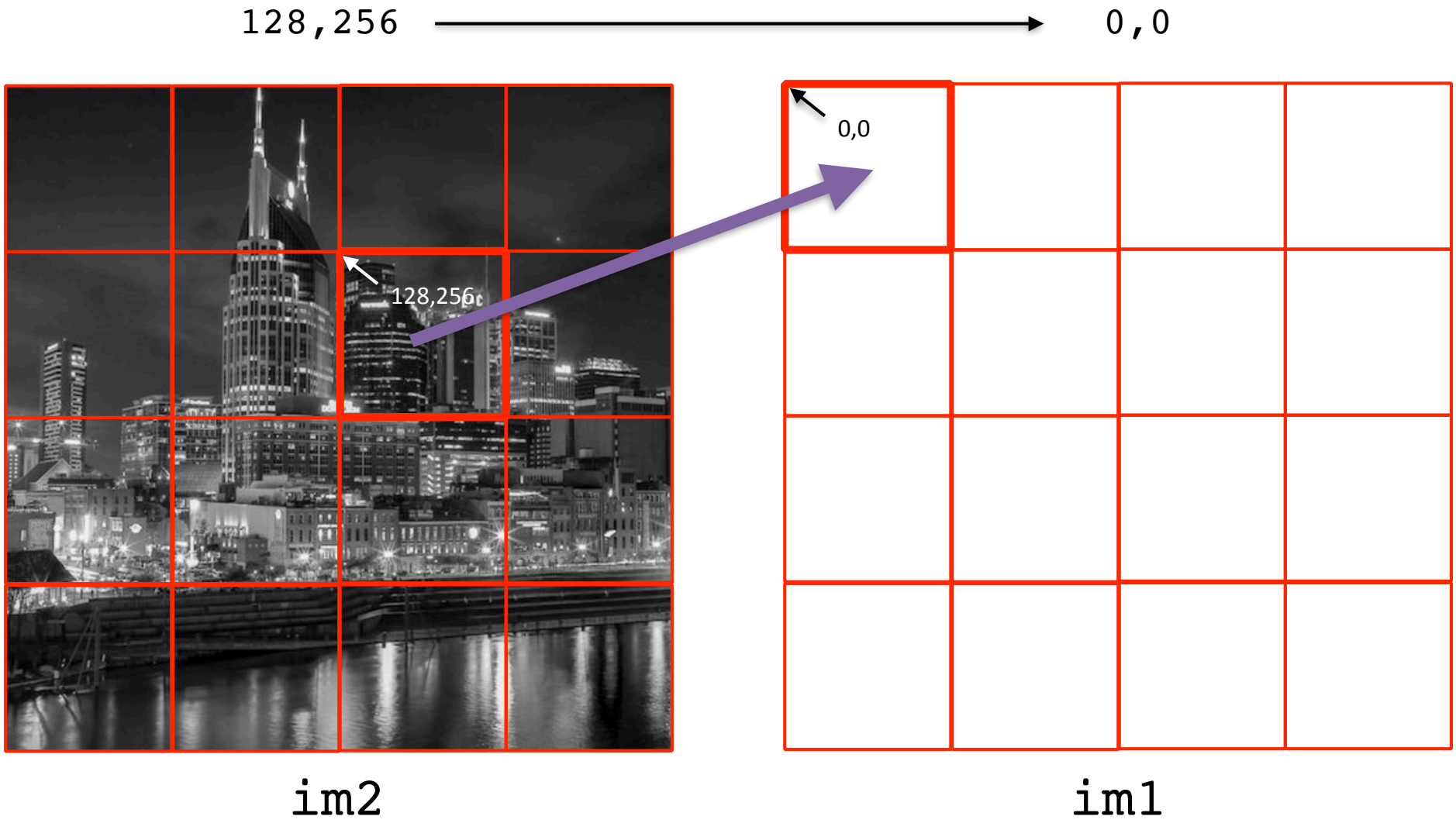
fill in a blank (scrambled) numpy array



Homework 8

`im1[0:128, 0:128] = im2[128:256, 256:384]`

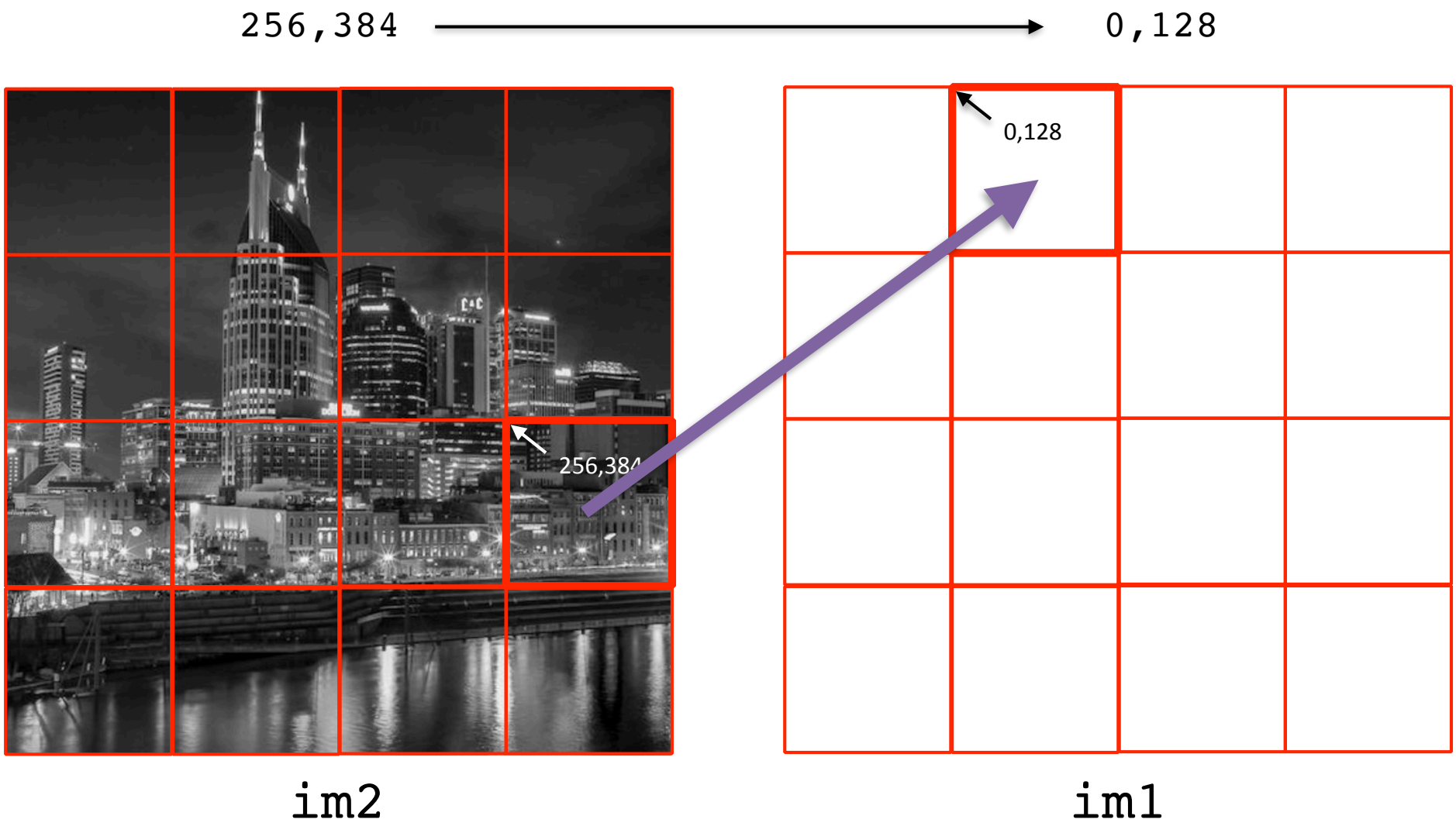
obviously, you can't hard code this



Homework 8

```
im1[0:128, 128:256] = im2[256:384, 384:512]
```

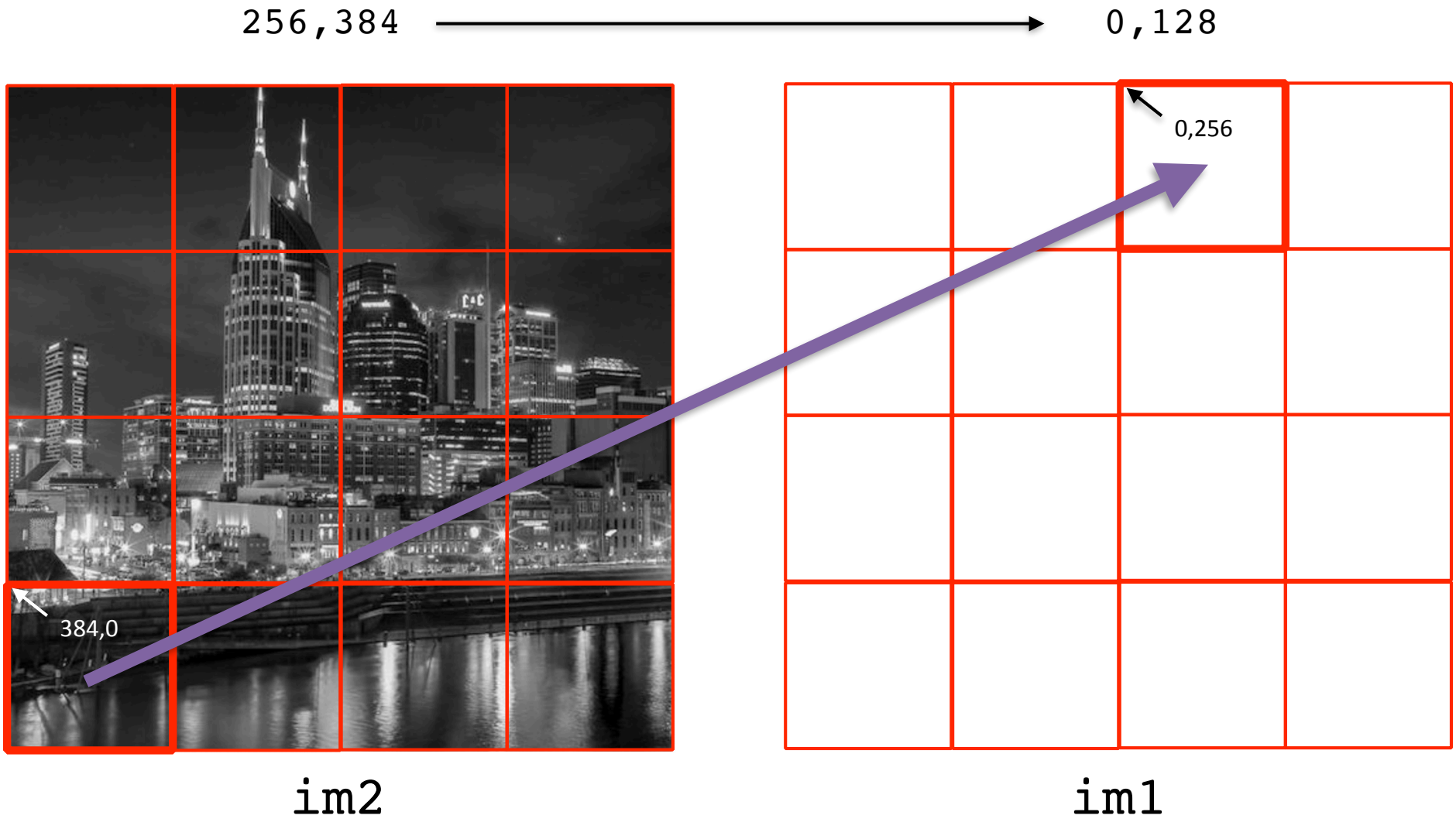
obviously, you can't hard code this



Homework 8

```
im1[0:128, 256:384] = im2[384:512, 0:128]
```

obviously, you can't hard code this



Homework 8

- **Q1c** add "salt and pepper" (normally-distributed) noise to the intensity of each image pixel

original

low noise

high noise

do this two ways: (1) where you cap out at 0 and 255,
(2) where you allow the numbers to "roll over"

("vectorized" for full credit)

download from Brightspace

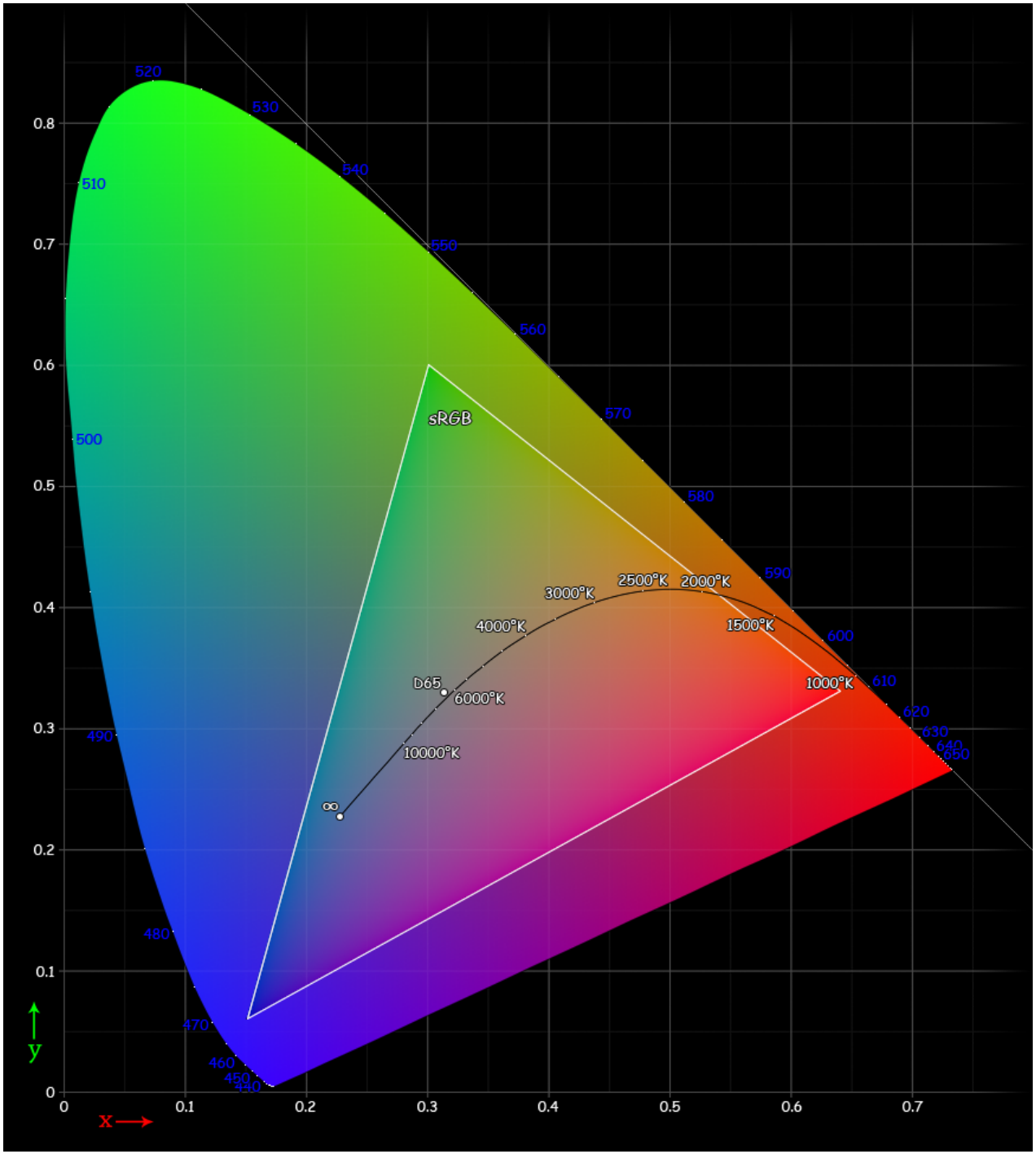
`Signals.zip`

More on Images and Signals

bit depth (monitors)

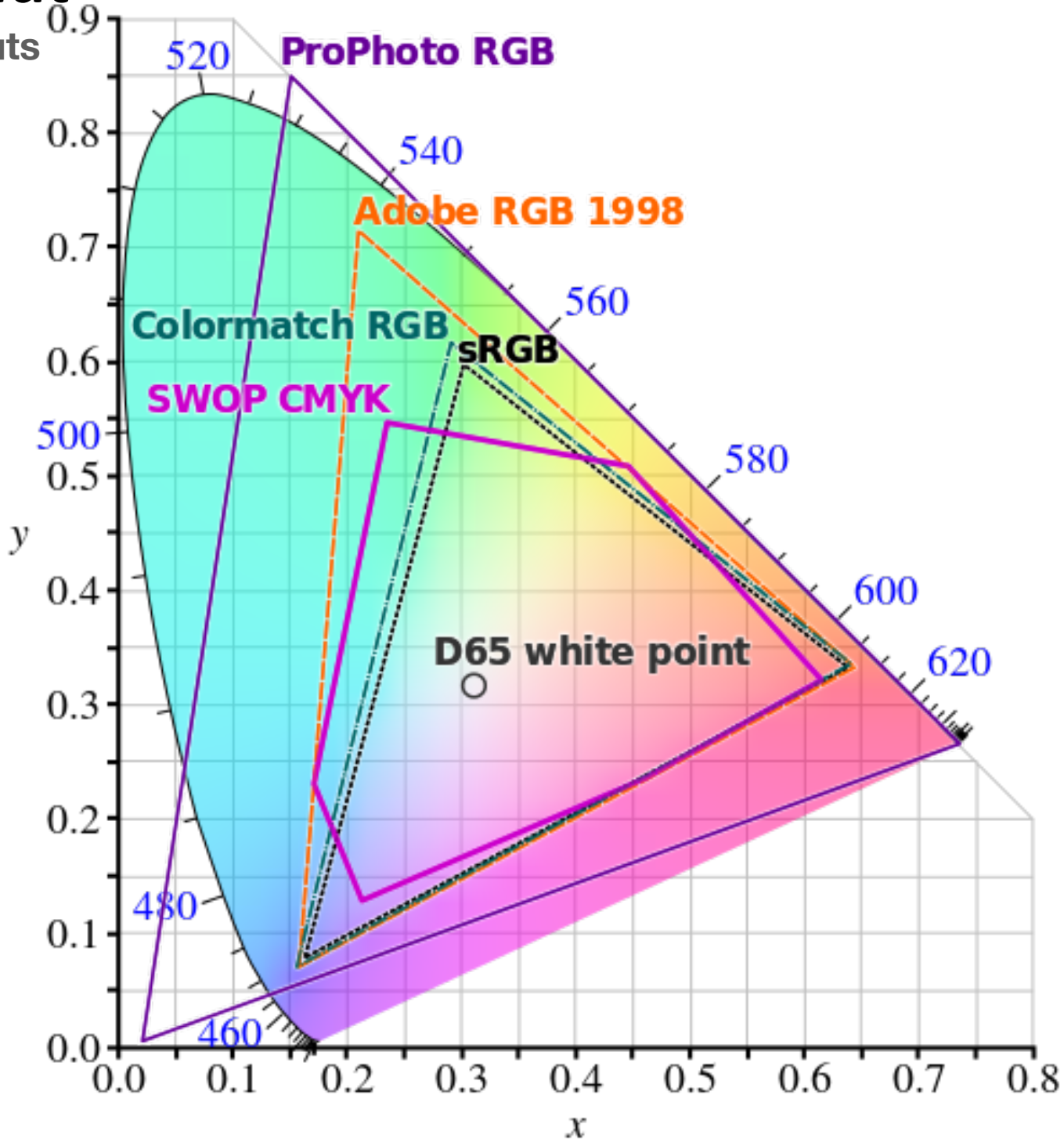
1st computers	1 bit (white/black, green/black)	
early NeXT	2 bits	
Atari, Commodore 64	4 bits	used color maps
early VGA	8 bits (total for color)	used color maps
most current monitor	24 bits (R, G, B)	
HDR monitors	30/36/48 bits (R, G, B)	
more than 3 primaries	R, G, B, C, Y, M	

color gamut


520
530
540
550
560
570
580
590
600
610
620
630
640
650
510
500
490
480
470
460
450
440
sRGB
2500°K
2000°K
3000°K
4000°K
1500°K
1000°K
D65
6000°K
10000°K
∞
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x

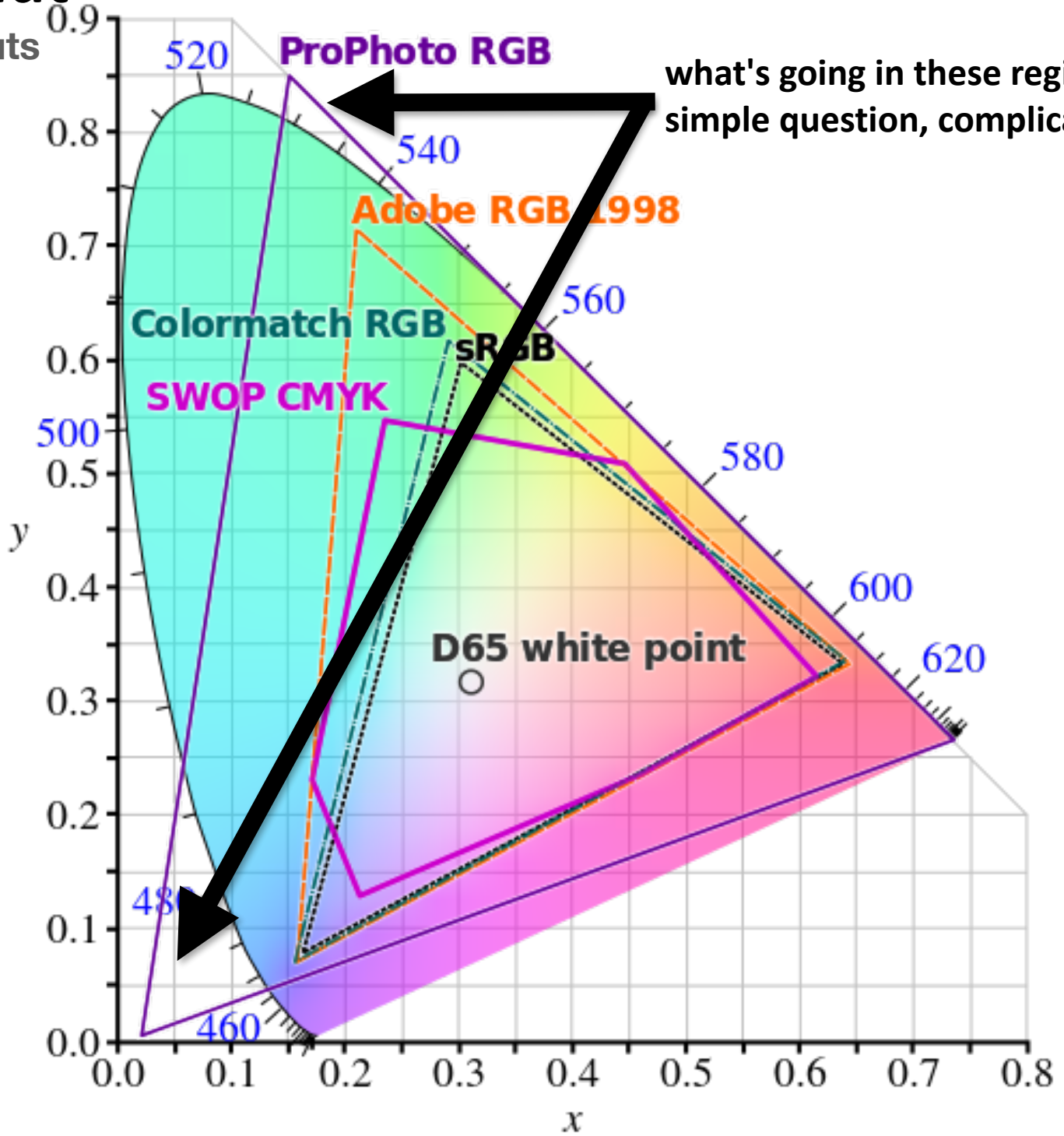
color gamut

printing gamuts

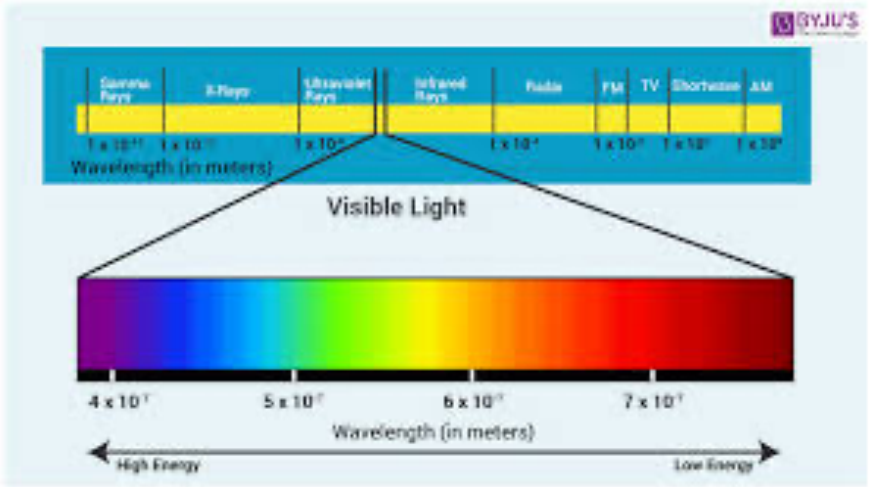


color gamut

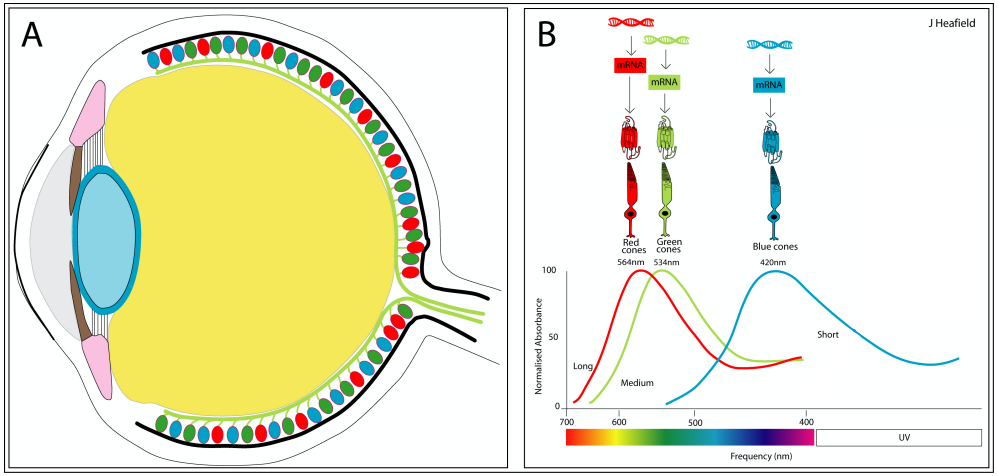
printing gamuts



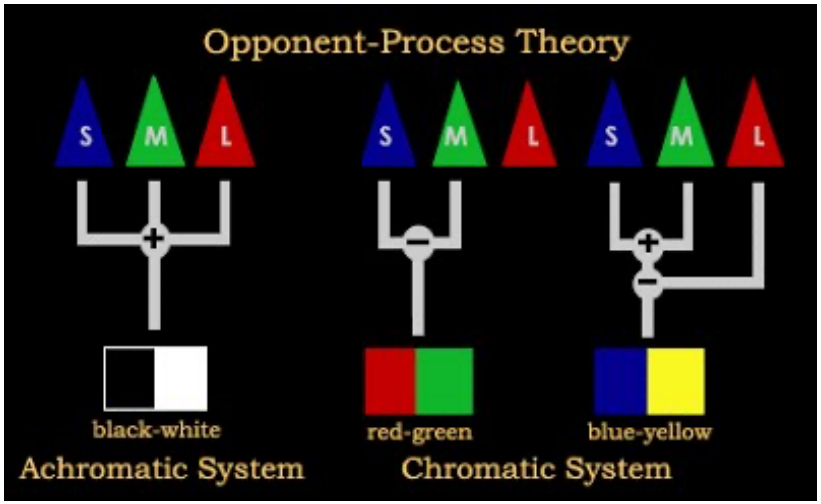
what's going in these regions?
simple question, complicated answer



not all colors we can perceive correspond to a single wavelength of light (e.g., grays, pinks, browns, magentas)



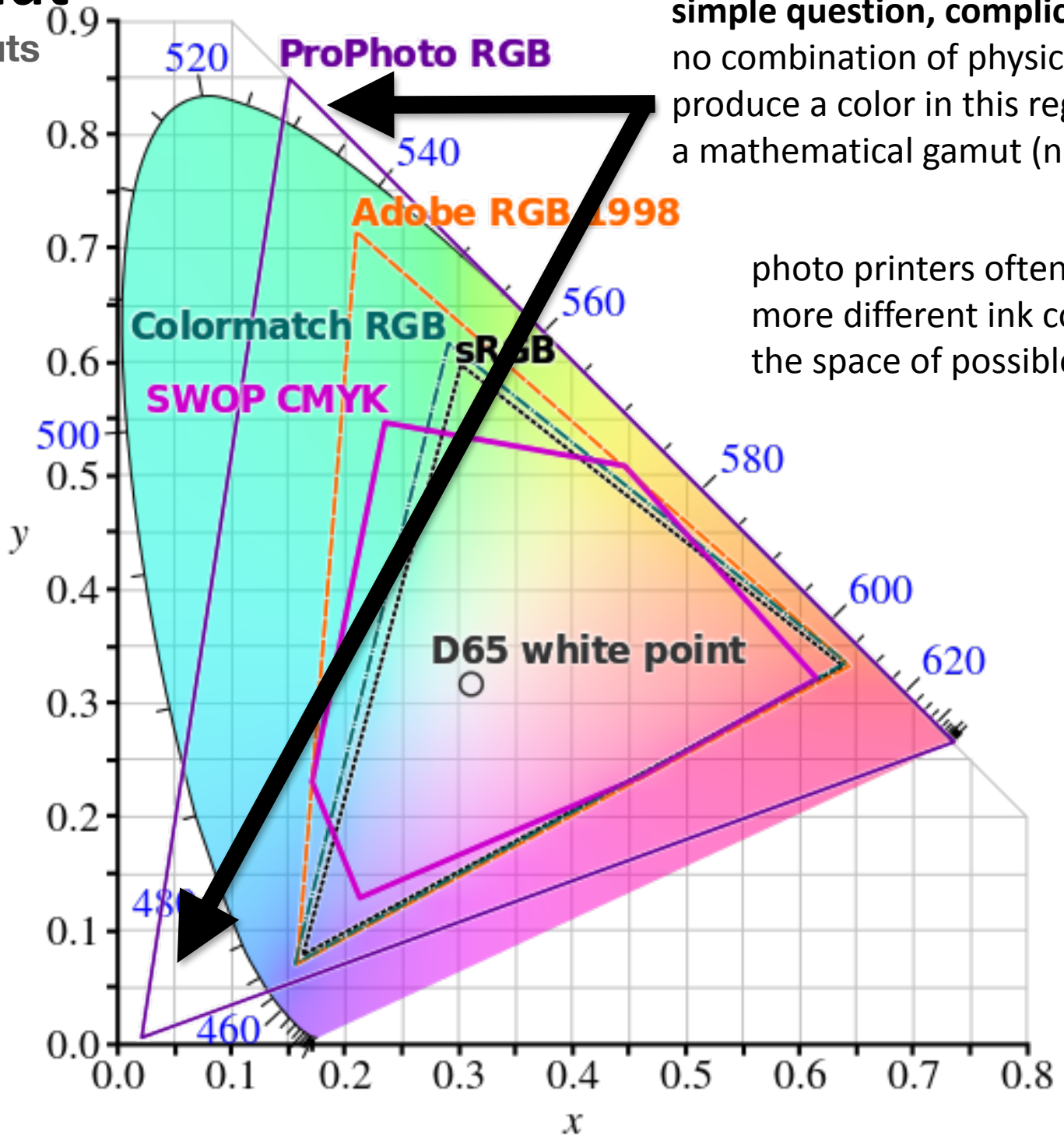
rods (low light) and cones (red, green, blue) photoreceptors in the retina



opponent-processes in higher-level color representation (black-white, red-green, blue-yellow)

color gamut

printing gamuts

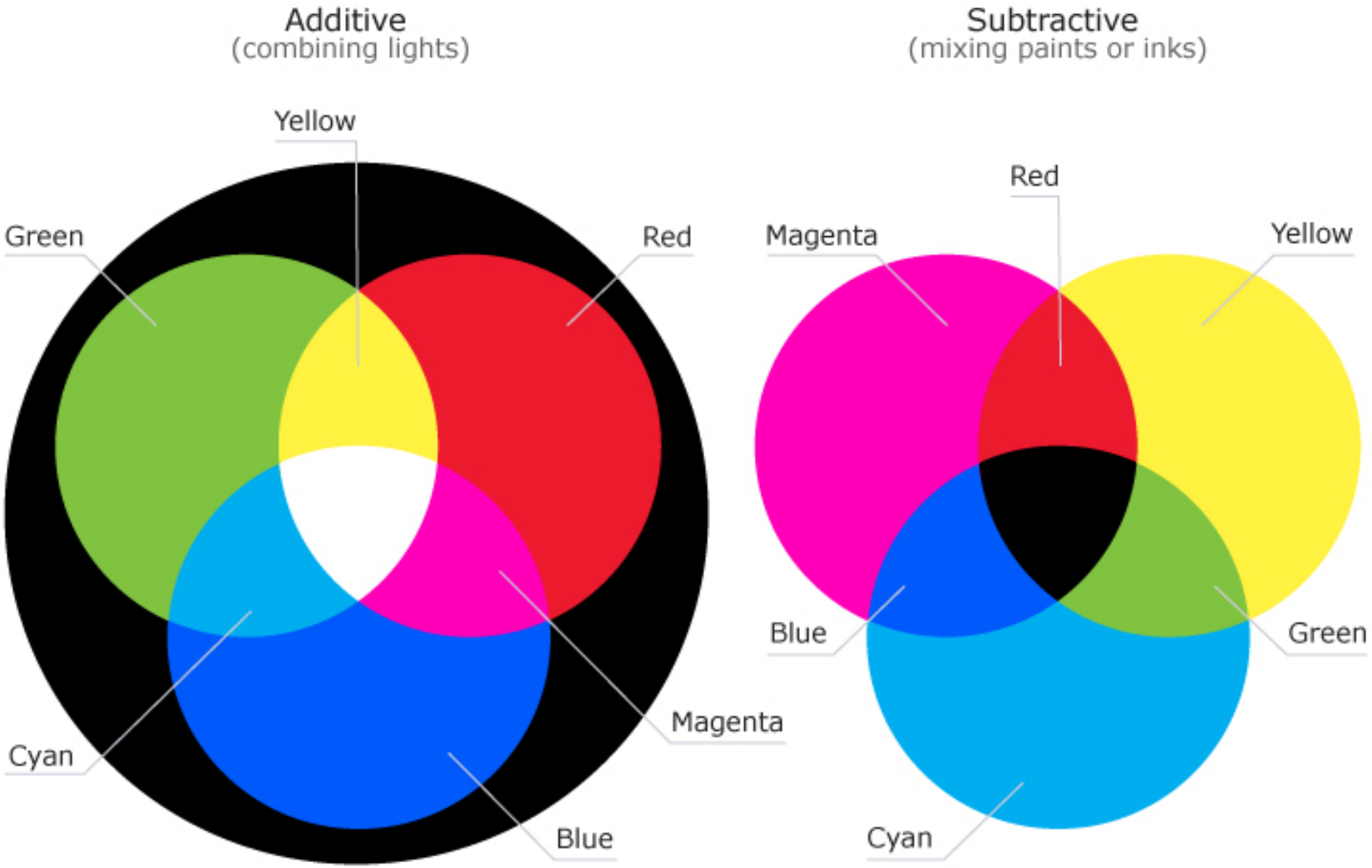


simple question, complicated answer
no combination of physical light can produce a color in this region - this is a mathematical gamut (not a real one)

photo printers often use 9 or more different ink colors to cover the space of possible colors

light mixing vs. paint mixing

light mixing is additive (base on projection)
paint mixing is subtractive (based on reflectance)



how resolution and bit depth affect size

how resolution and bit depth affect size

assuming no compression

lower resolution and bit depth audio recording
10sec audio x 44.1kHz x 2 bytes/sample
10sec audio x 44.1ksamples/sec x 2 bytes/sample
882,000 bytes

how resolution and bit depth affect size

assuming no compression

lower resolution and bit depth audio recording
10sec audio x 44.1kHz x 2 bytes/sample
10sec audio x 44.1ksamples/sec x 2 bytes/sample
882,000 bytes

high resolution and bit depth audio recording
10sec audio x 192kHz x 3 bytes/sample
10sec audio x 192ksamples/sec x 3 bytes/sample
5,760,000 bytes

how resolution and bit depth affect size

assuming no compression

lower resolution and bit depth image
320x240 image x 8 bit color
76,800 bytes

high resolution and bit depth audio recording
7360x4912 image x 42 bit color (raw)
189,799,680 bytes

how resolution and bit depth affect size

size not only affects storage needs (computer memory and disk drive space) but time to copy/transfer files

how resolution and bit depth affect size

1000 lower resolution and bit depth images
76,800,000 bytes
256Kb/sec = 5min
10Mb/sec = <8sec
10Gb/sec = <.008sec

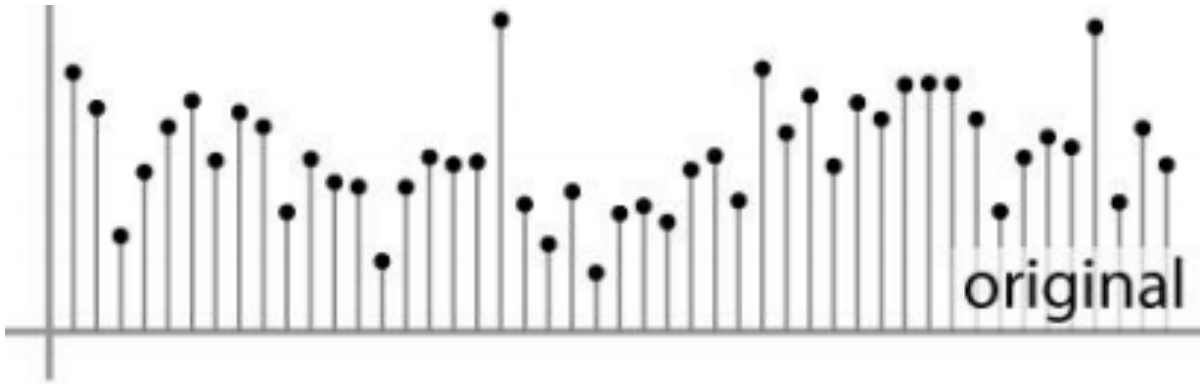
Comcast/Xfinity upload speeds capped at 20Mb/sec

1000 high resolution and bit depth images
189,799,680,000 bytes
256Kb/sec = >8 days
10Mb/sec = <6hours
10Gb/sec = <19sec

size not only affects storage needs (computer memory and disk drive space) but time to copy/transfer files

Filtering

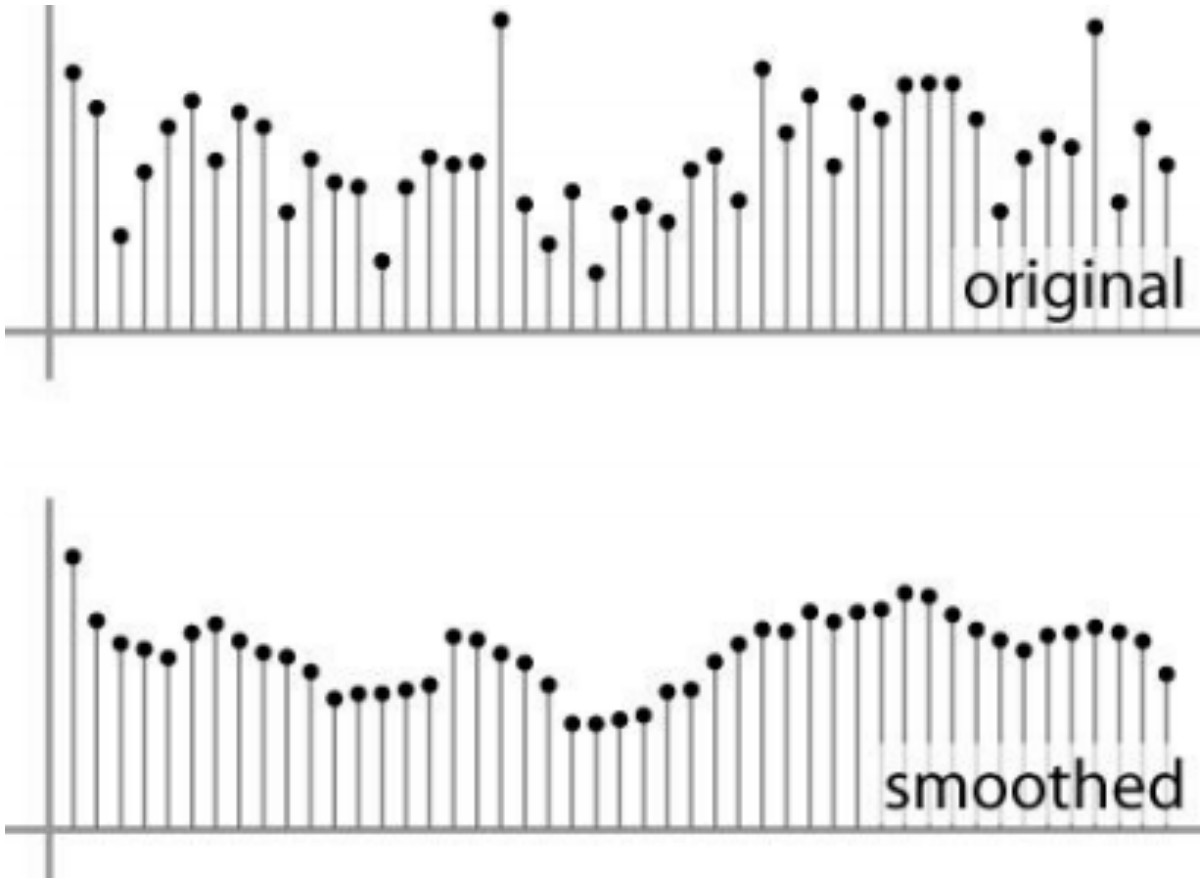
1D signal



x axis could be time or space
y axis is intensity at each time point or spatial position

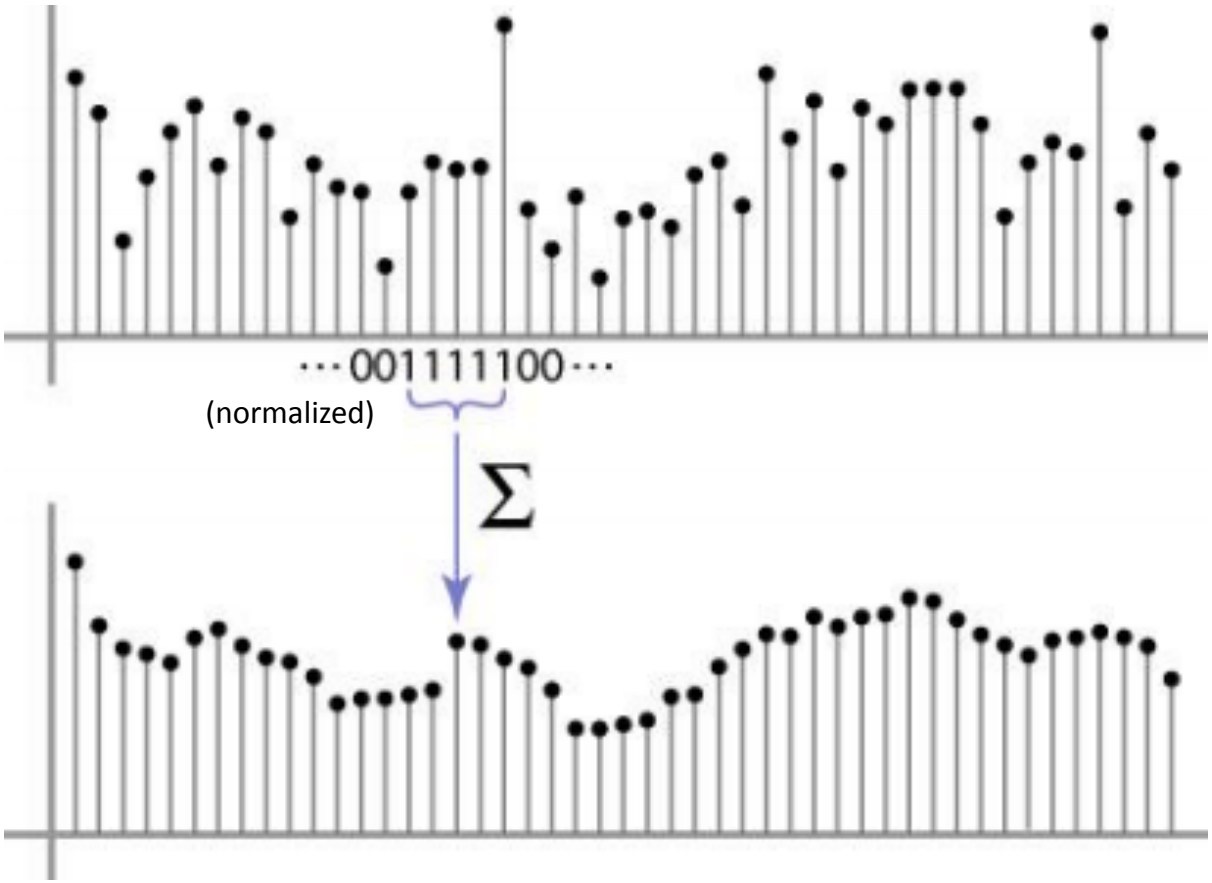
For example:
pixel intensity of 1D image
amplitude of a sound wave over time
electrical potential at one electrode over time
MR measurement in one brain voxel

e.g., a moving average filter (blur)



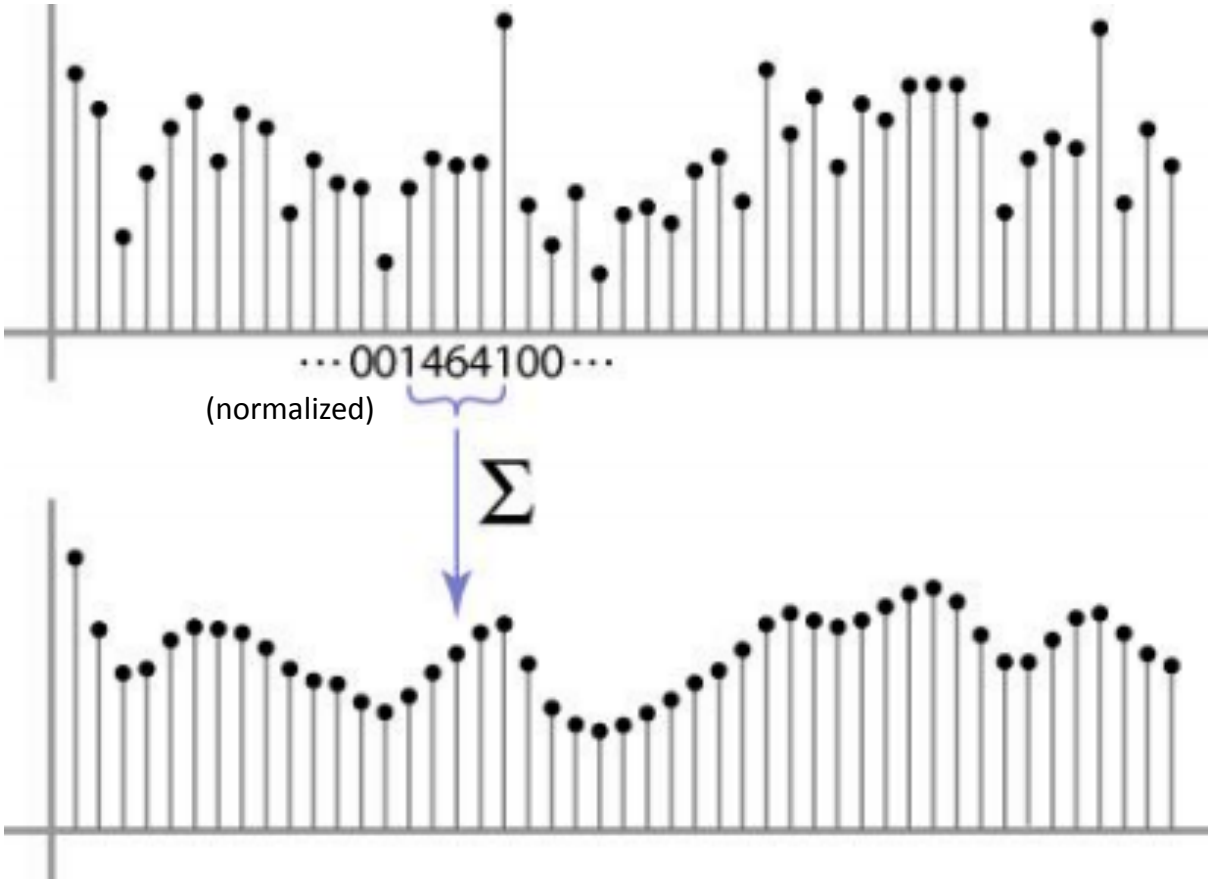
e.g., a moving average filter

```
weights = np.array([ 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, ]) / 5
```



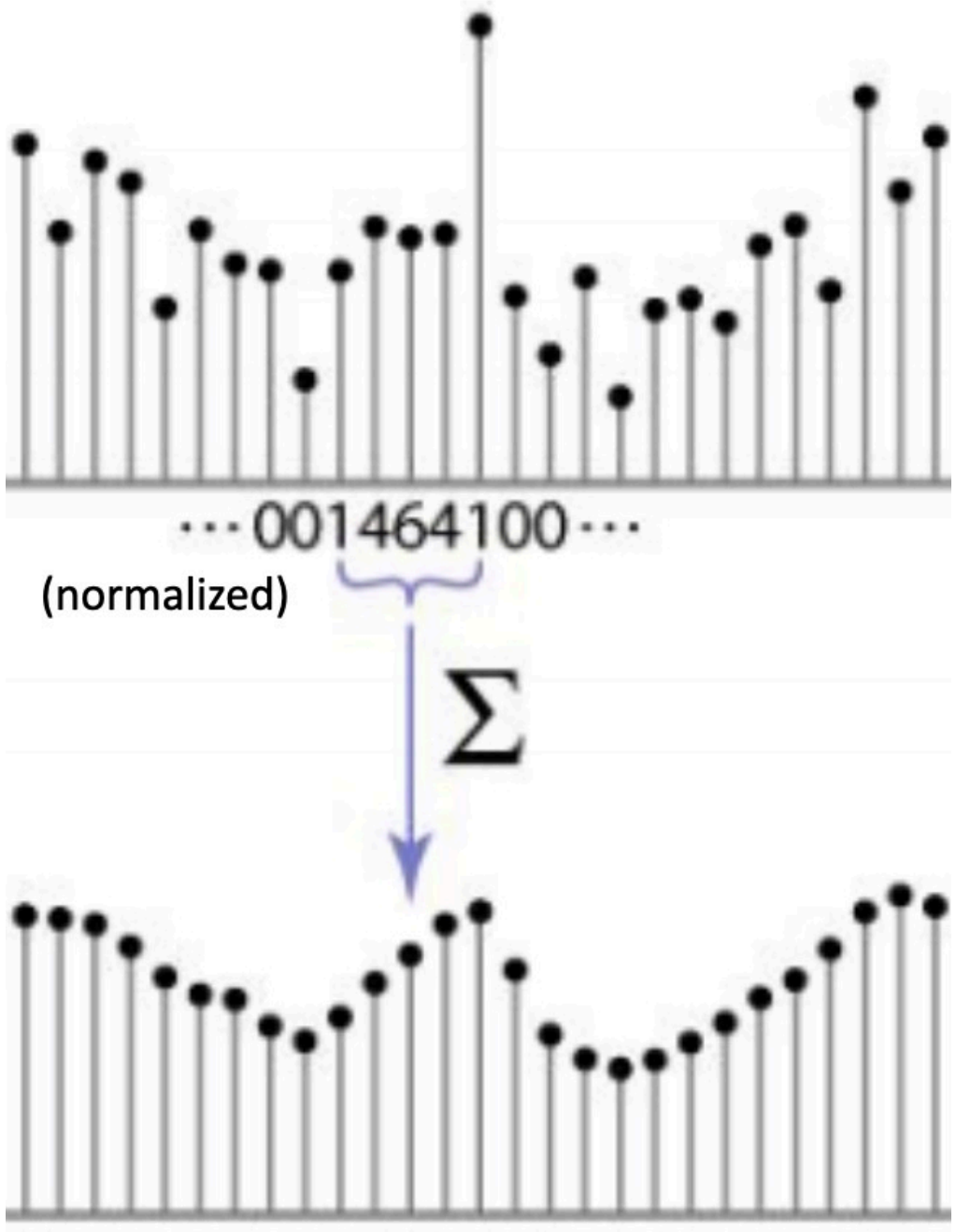
e.g., a moving average filter

```
weights = np.array([ 0, 0, 0, 0, 1, 4, 6, 4, 1, 0, 0, 0, 0, ]) / 16
```



e.g., a moving average filter

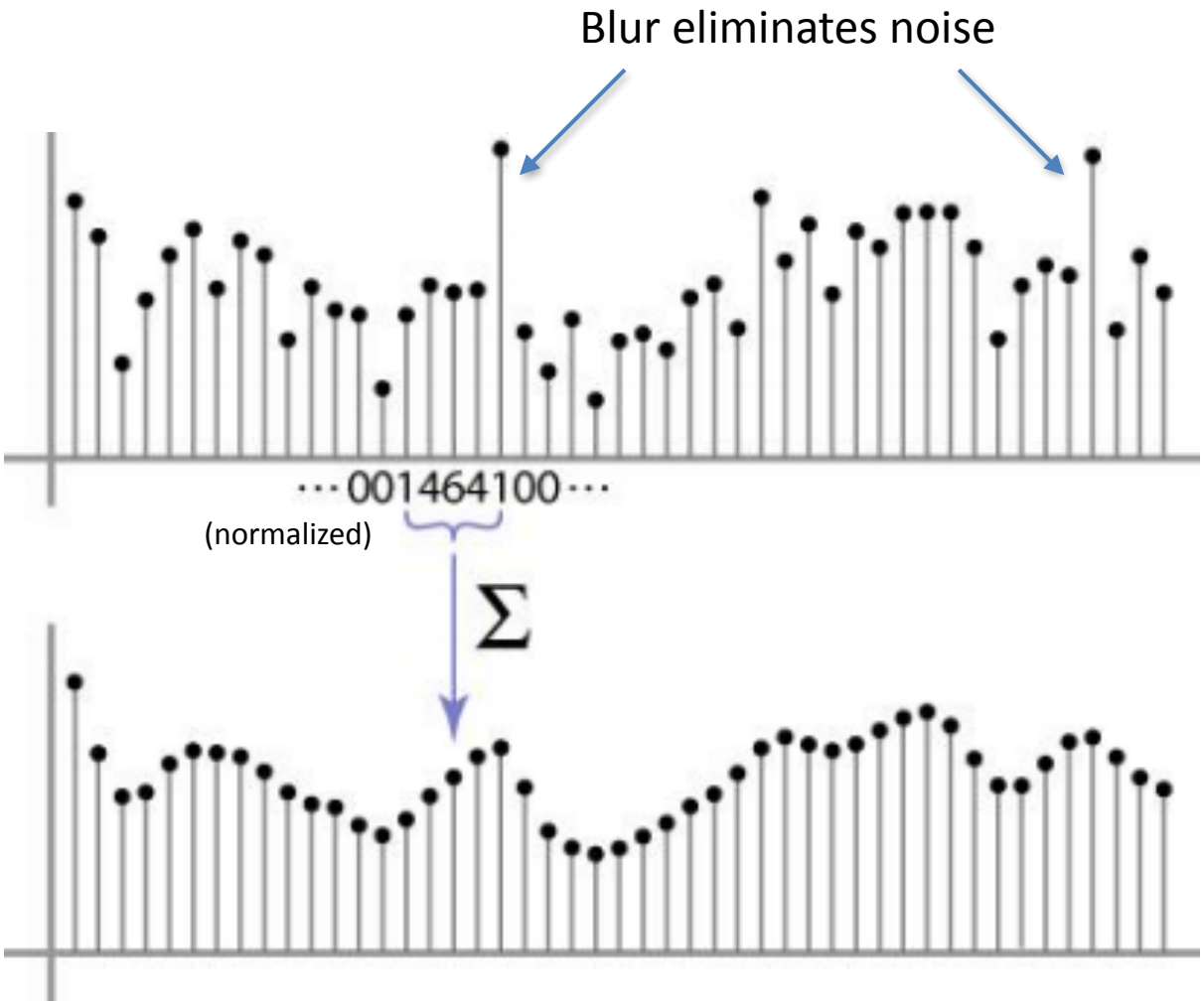
```
weights = np.array([ 0, 0, 0, 0, 1, 4, 6, 4, 1, 0, 0, 0, 0, ]) / 16
```



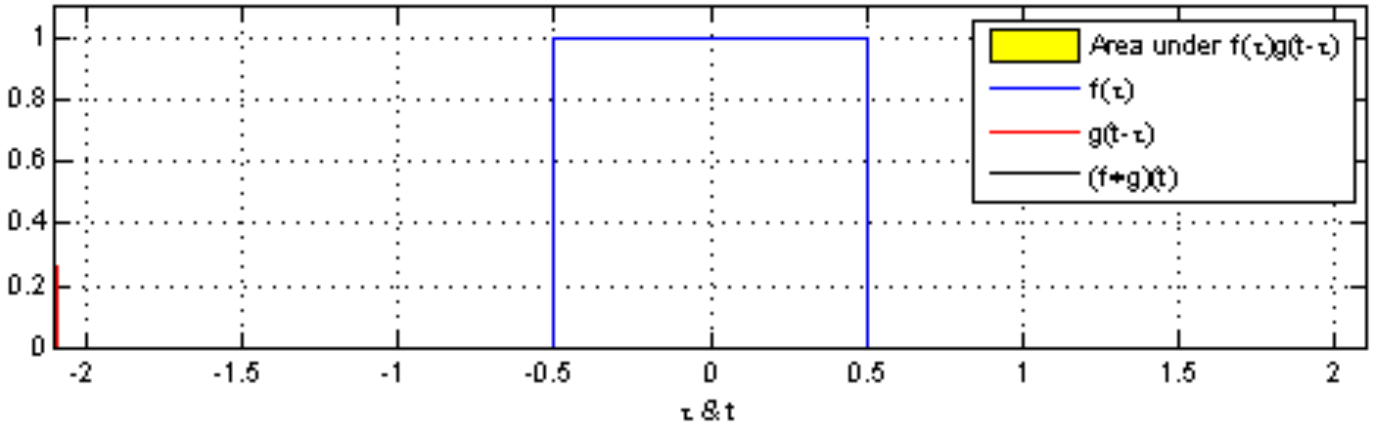
weights sum to 1

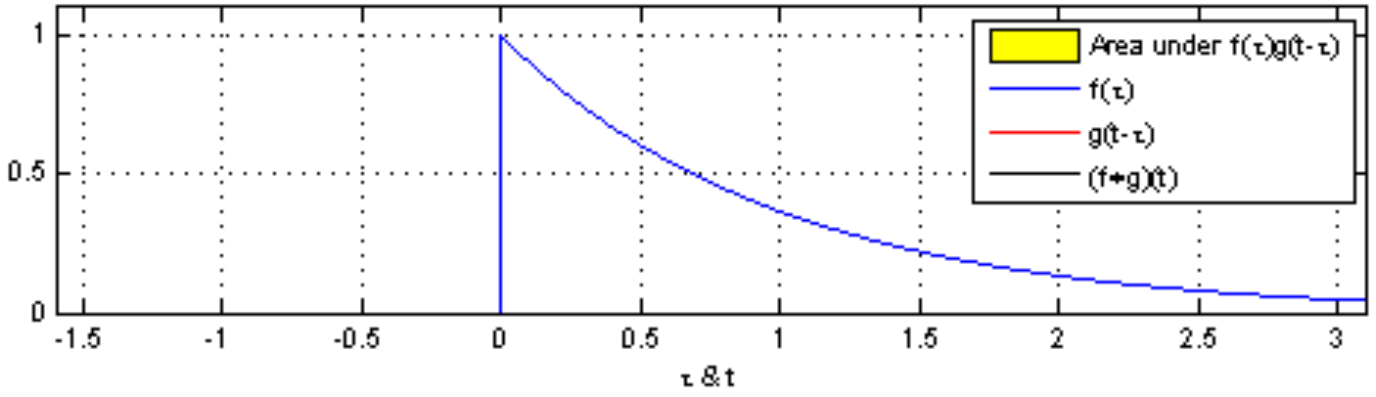
e.g., a moving average filter

```
weights = np.array([ 0, 0, 0, 0, 1, 4, 6, 4, 1, 0, 0, 0, 0, ]) / 16
```



Illustration





0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

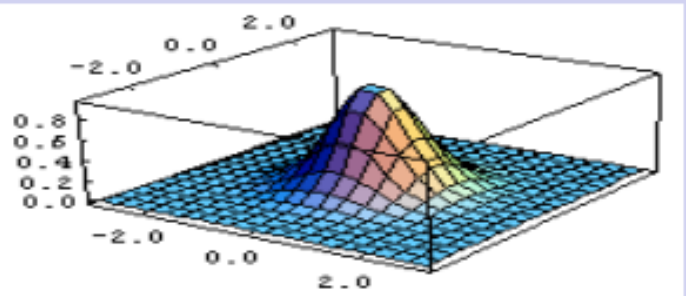
$I[i, j]$

(normalized)

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$F[u, v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

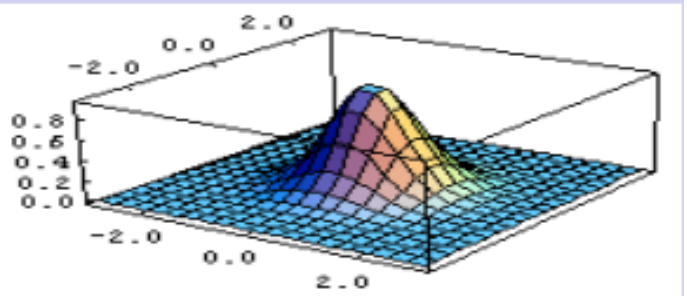
$I[\,i, j\,]$

(normalized)

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$F[u, v]$



Blur eliminates noise

$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

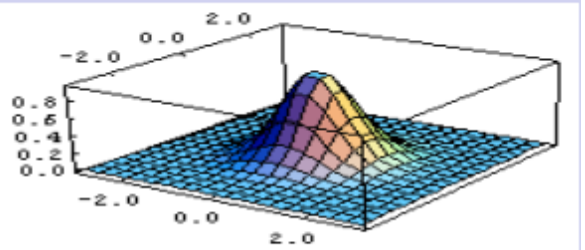
$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\, I_{in}(i-u, j-v)$$

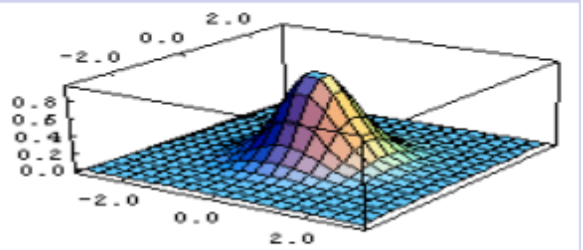
filtered image pixel (i,j)

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\, I_{in}(i-u, j-v)$$

filtered image pixel (i,j)

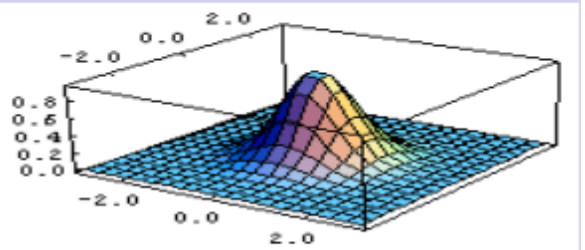
pixels from input image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[i,j]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

also called a kernel

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\, I_{in}(i-u, j-v)$$

filtered image pixel (i,j)

pixels from filter

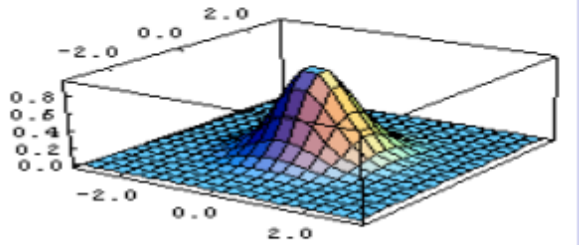
pixels from input image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

(kernel size) filter size

also called a kernel

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\, I_{in}(i-u, j-v)$$

filtered image pixel (i,j)

pixels from filter

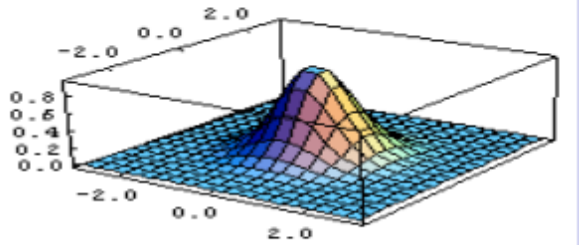
pixels from input image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\, I_{in}(i-u, j-v)$$

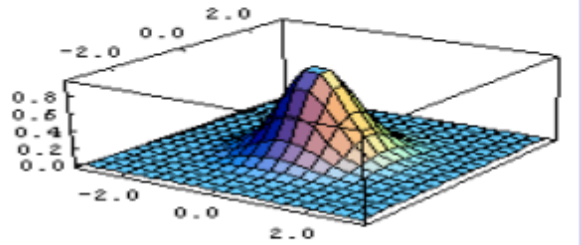
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

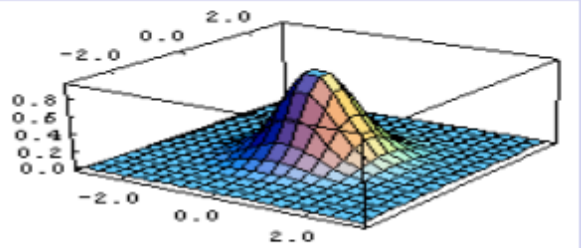
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

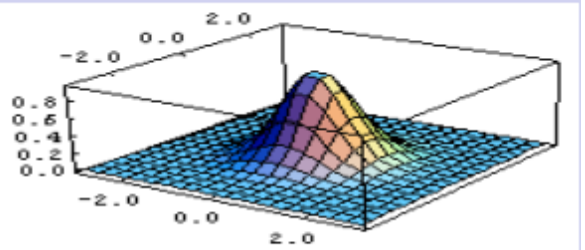
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

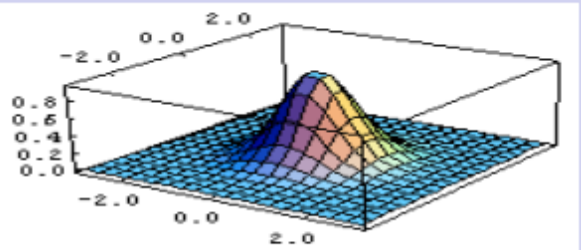
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

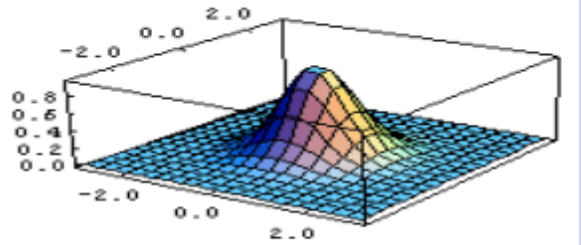
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

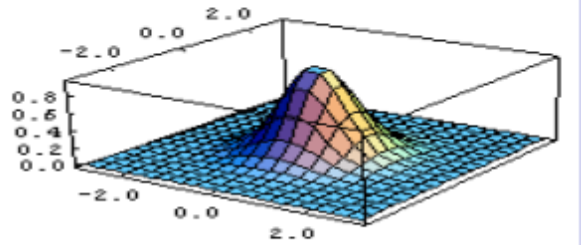
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

"weighted sum"

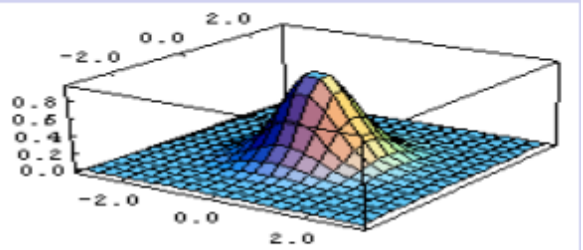
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

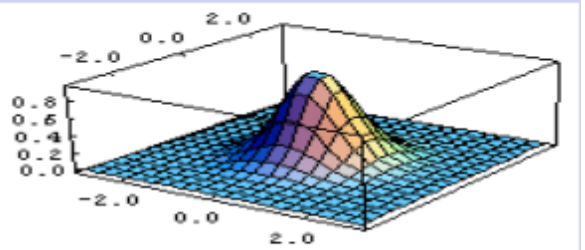
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

"weighted sum"

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

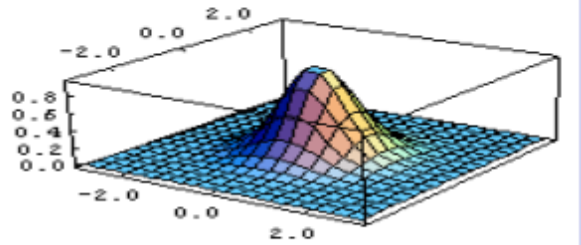
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$I[\,i,j\,]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$F[u,v]$



$$f(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

Convolution

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

discrete version

in Python, you can't define a numpy array with negative indices like this, so you'd need to adjust accordingly

Convolution

$$I_{out}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

discrete version

$$I_{out}(x,y) = \int\int F(u,v)\ I_{in}(x-u, y-v)\ du\ dv$$

continuous version

Convolution

$$I_{out}(i,j) = F(i,j) * I_{in}(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v)\ I_{in}(i-u, j-v)$$

discrete version

*symbolically in mathematics, * is convolution (never multiplication)*

$$I_{out}(x,y) = F(x,y) * I_{in}(x,y) = \int\int F(u,v)\ I_{in}(x-u, y-v)\ du\ dv$$

continuous version

Convolution

$$I_{out}(t) = F(t) * I_{in}(t) = \sum_{\tau=-k}^{k} F(\tau)\ I_{in}(t-u)$$

discrete version

1D (e.g., sound)

$$I_{out}(t) = F(t) * I_{in}(t) = \int F(\tau)\ I_{in}(i-\tau)\ d\tau$$

continuous version

Convolution Operations

commutative

$$f(t) * g(t) = g(t) * f(t)$$

associative

$$f(t) * (g(t) * h(t)) = (f(t) * g(t)) * h(t)$$

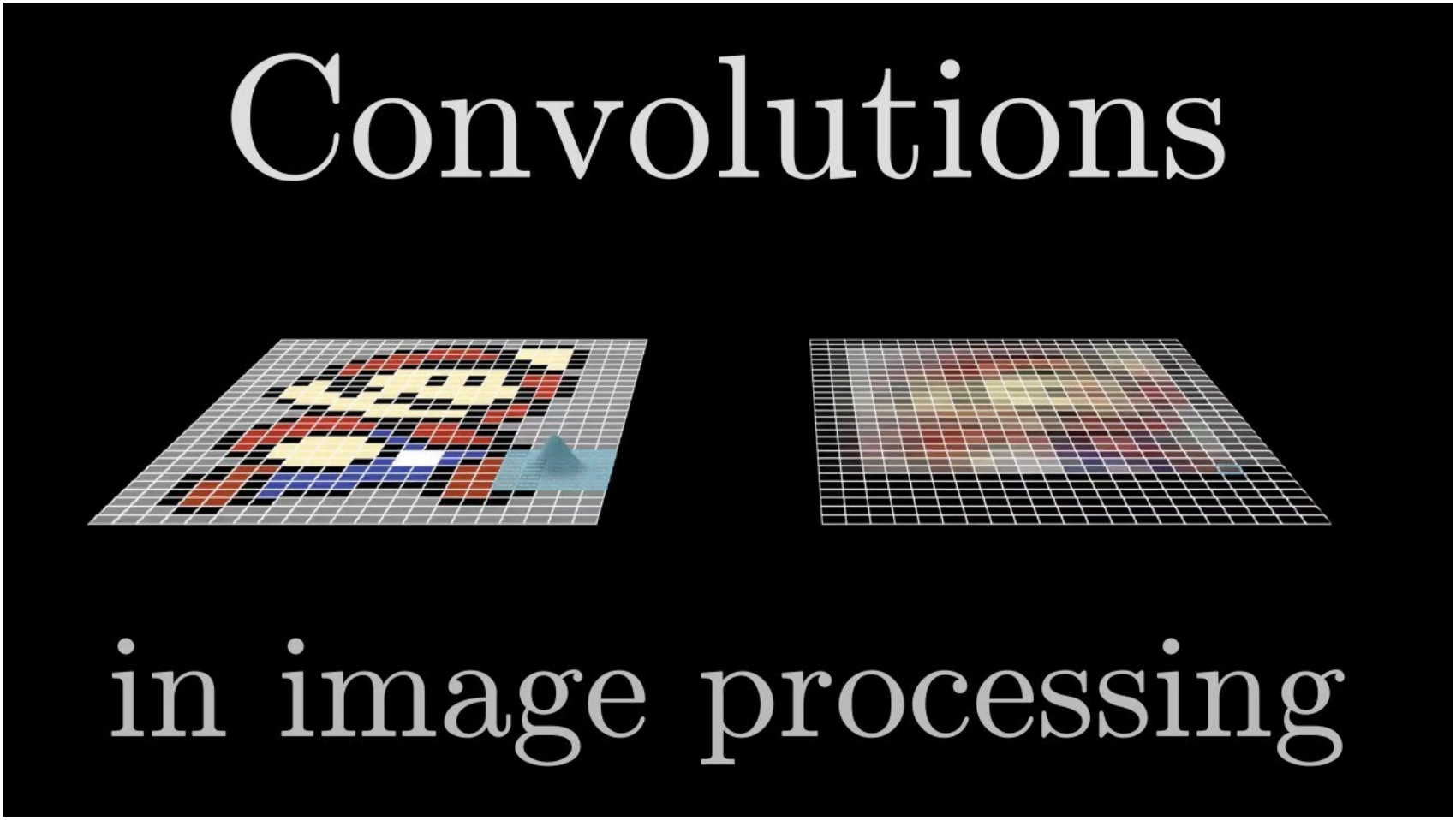
distributive

$$f(t) * (g(t) + h(t)) = f(t) * g(t) + f(t) * h(t)$$

linear

$$\alpha(f(t) * g(t)) = (\alpha f(t)) * g(t) = f(t) * (\alpha g(t))$$

what's the "signal" and what's the "filter" is arbitrary



https://www.youtube.com/watch?v=8rrHTtUzyZA

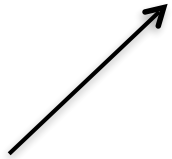
watch from 1:00 to 7:30
(rest is implementation in Julia)

(Grant Sanderson, creator of 3blue1brown)

convolutions in Python

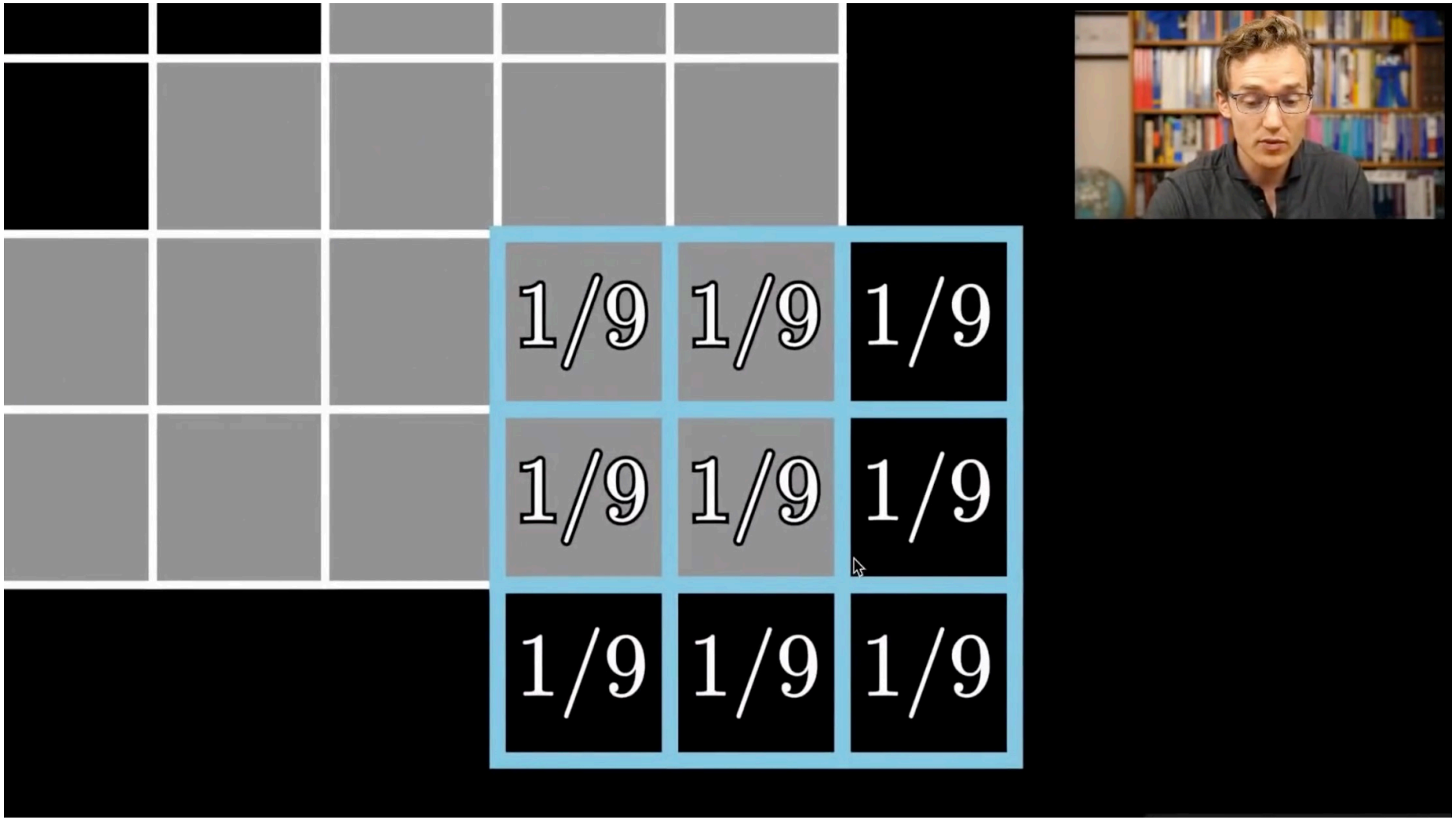
```
from scipy import signal

filtered = signal.convolve(sig, fil, mode='same')
```

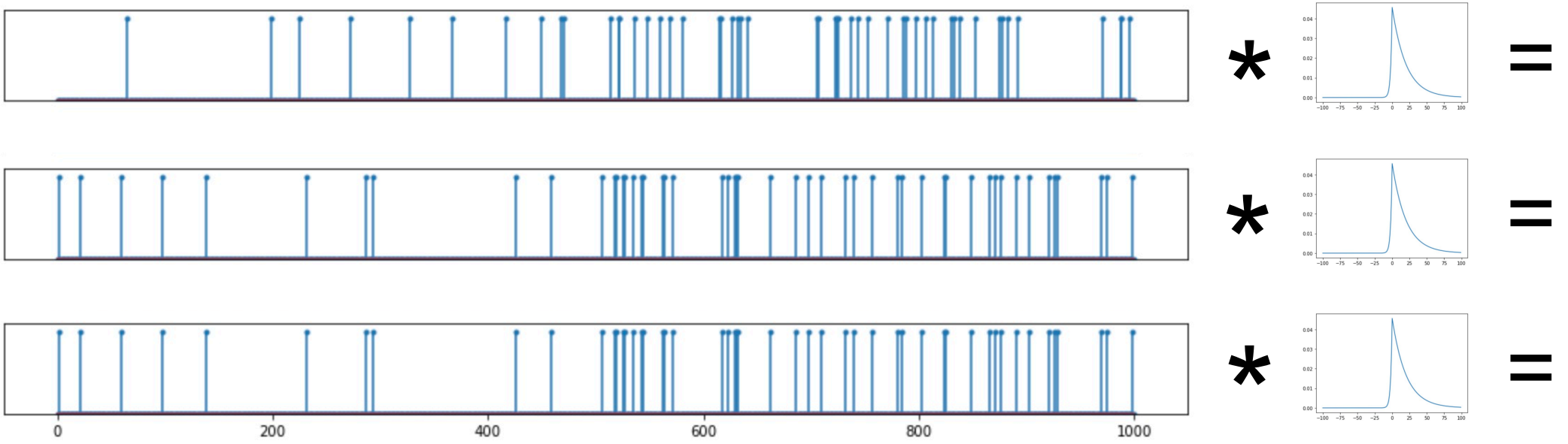


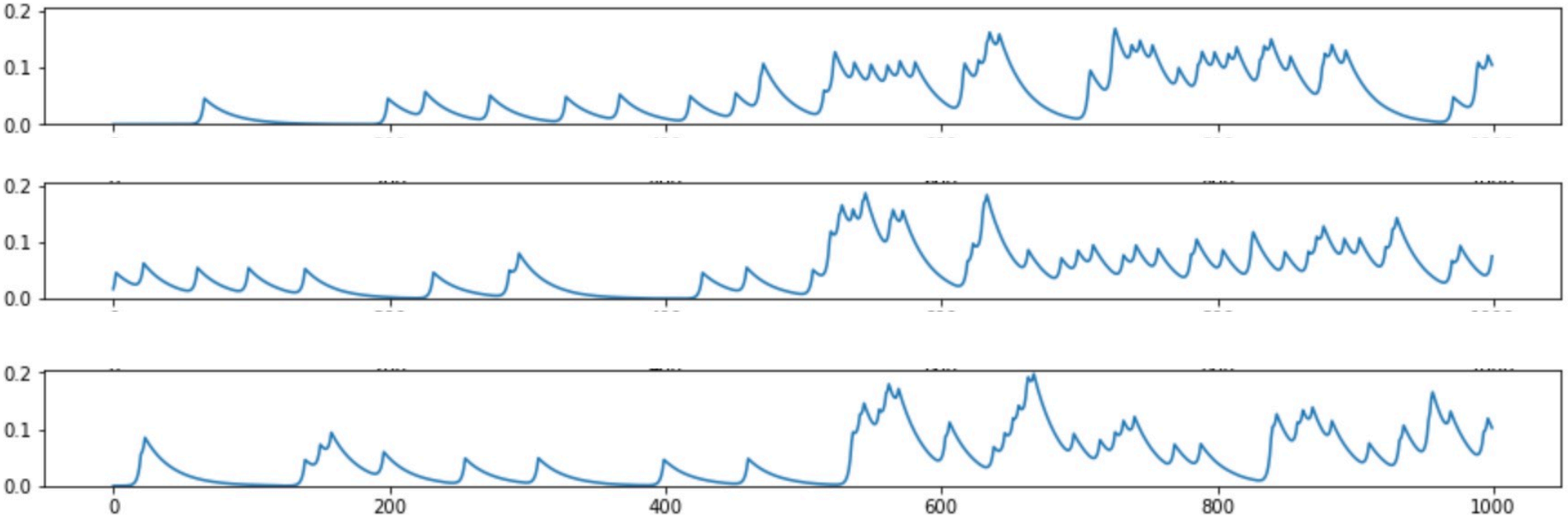
'same' means that the output is *not* padded …
the output is the *same* length as the signal

```
filtered = signal.convolve2(sig, fil, mode='same')
```

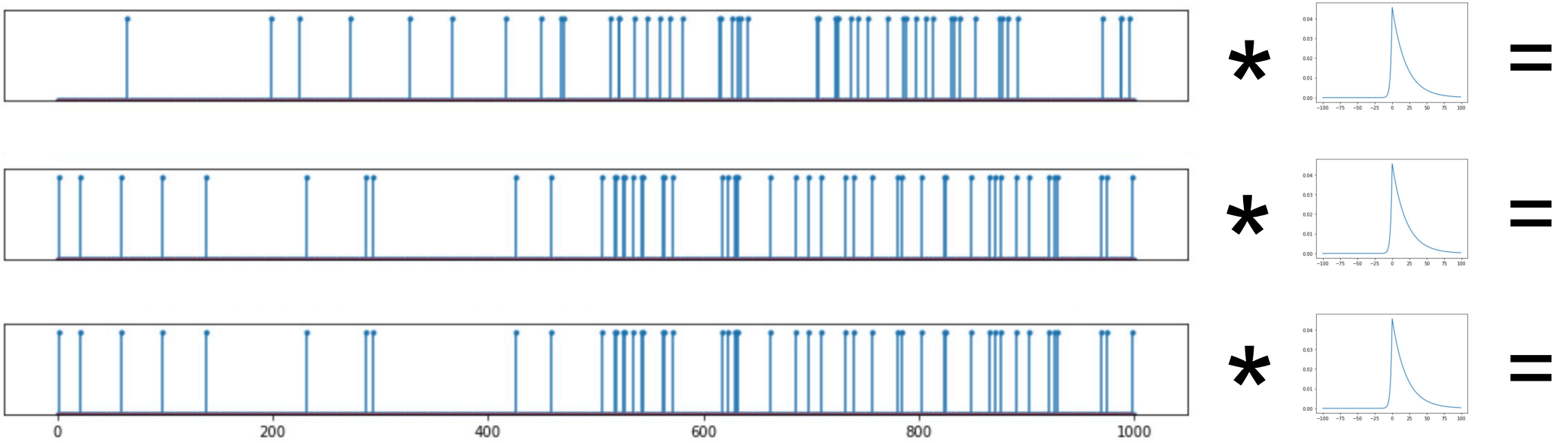

1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9

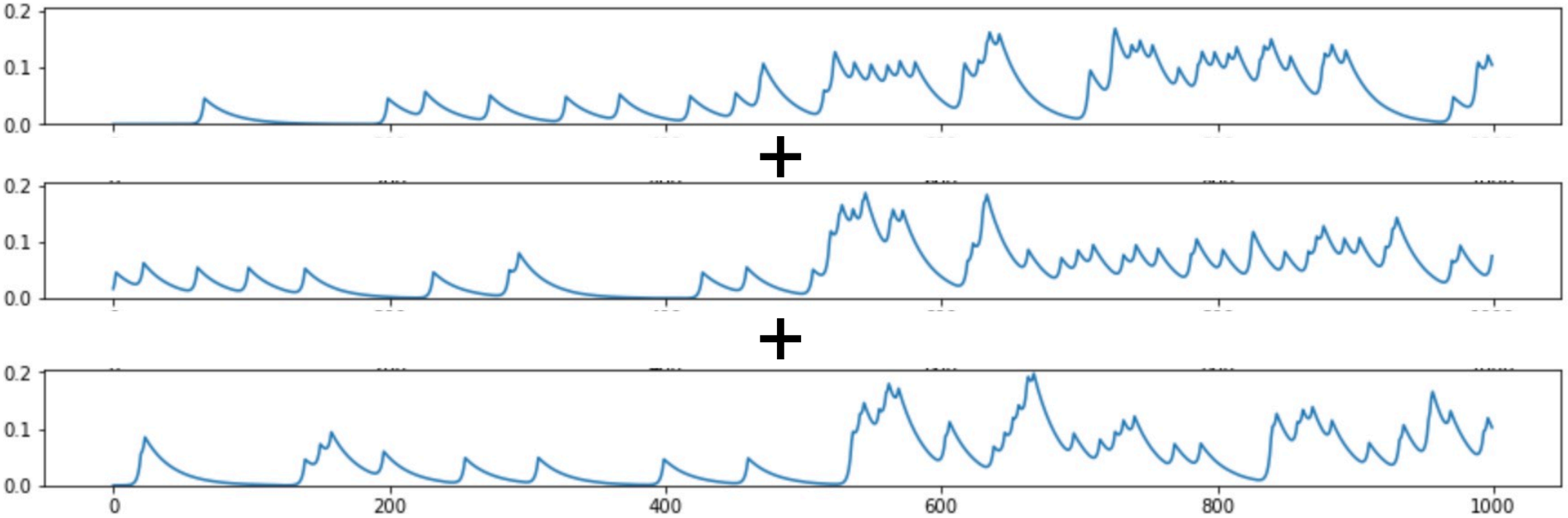
neural spiking example





neural spiking example





neural spiking example