THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision

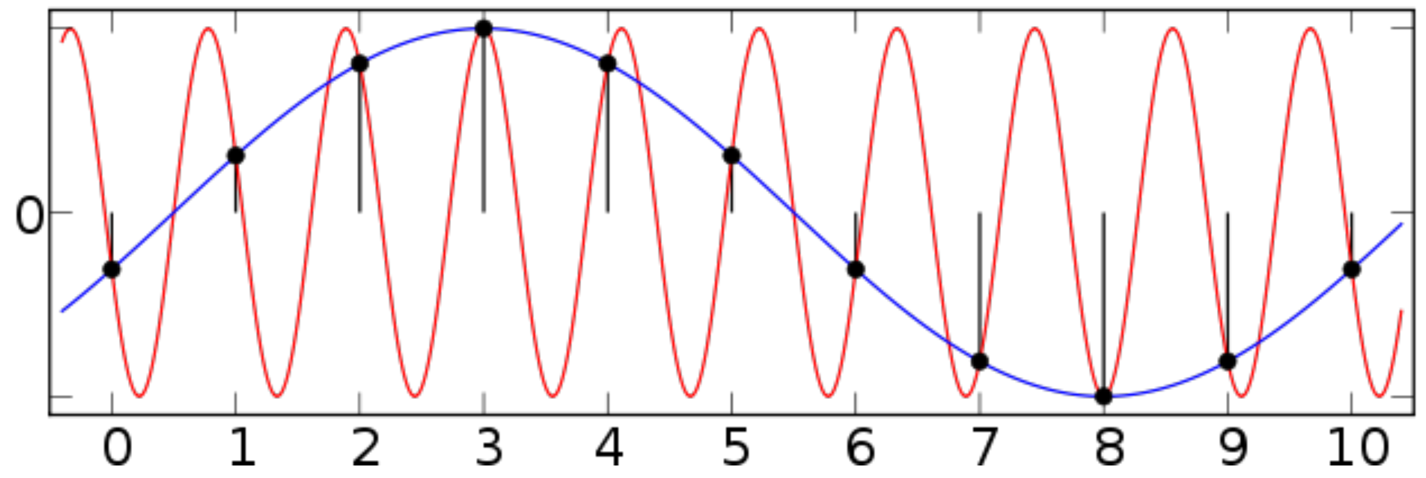


Image Credit: https://en.wikibooks.org/wiki/Analog_and_Digital_Conversion/Nyquist_Sampling_Rate

Lecture 6: Sampling

(unless otherwise stated slides are taken or adopted from **Bob Woodham, Jim Little** and **Fred Tung**)

Menu for Today (**September 23, 2024**)

Topics:

- **Sampling** theory
- **Nyquist** rate
- Color **Filter Arrays**
- **Image** encoding

Readings:

- **Today's** Lecture: Szeliski 2.3, Forsyth & Ponce (2nd ed.) 4.5, 4.6

Reminders:

- **Assignment 1:** Image Filtering and Hybrid Images due **September 26th**

Lecture 5: Re-cap The Convolution Theorem

Convolution **Theorem**:

Let $$i'(x,y) = f(x,y) \otimes i(x,y)$$

then $$\mathcal{I}'(w_x,w_y) = \mathcal{F}(w_x,w_y)\,\mathcal{I}(w_x,w_y)$$

where $\mathcal{I}'(w_x,w_y)$, $\mathcal{F}(w_x,w_y)$, and $\mathcal{I}(w_x,w_y)$ are Fourier transforms of $i'(x,y)$, $f(x,y)$ and $i(x,y)$

At the expense of two **Fourier** transforms and one inverse Fourier transform, convolution can be reduced to (complex) multiplication

Lecture 5: Re-cap The Convolution Theorem

General implementation of **convolution**:

At each pixel, (X, Y), there are	$m \times m$	multiplications
There are	$n \times n$	pixels in (X, Y)
Total:	$m^2 \times n^2$	multiplications

Convolution if FFT space:

Cost of FFT/IFFT for image: $\mathcal{O}(n^2 \log n)$

Cost of FFT/IFFT for filter: $\mathcal{O}(m^2 \log m)$

Cost of convolution: $\mathcal{O}(n^2)$

Note: not a function of filter size !!!

Lecture 5: Re-cap Median Filter

Take the **median value** of the pixels under the filter:

5	13	5	221
4	16	7	34
24	54	34	23
23	75	89	123
54	25	67	12

Image

4	5	5	7	13	16	24	34	54

	13		

Output

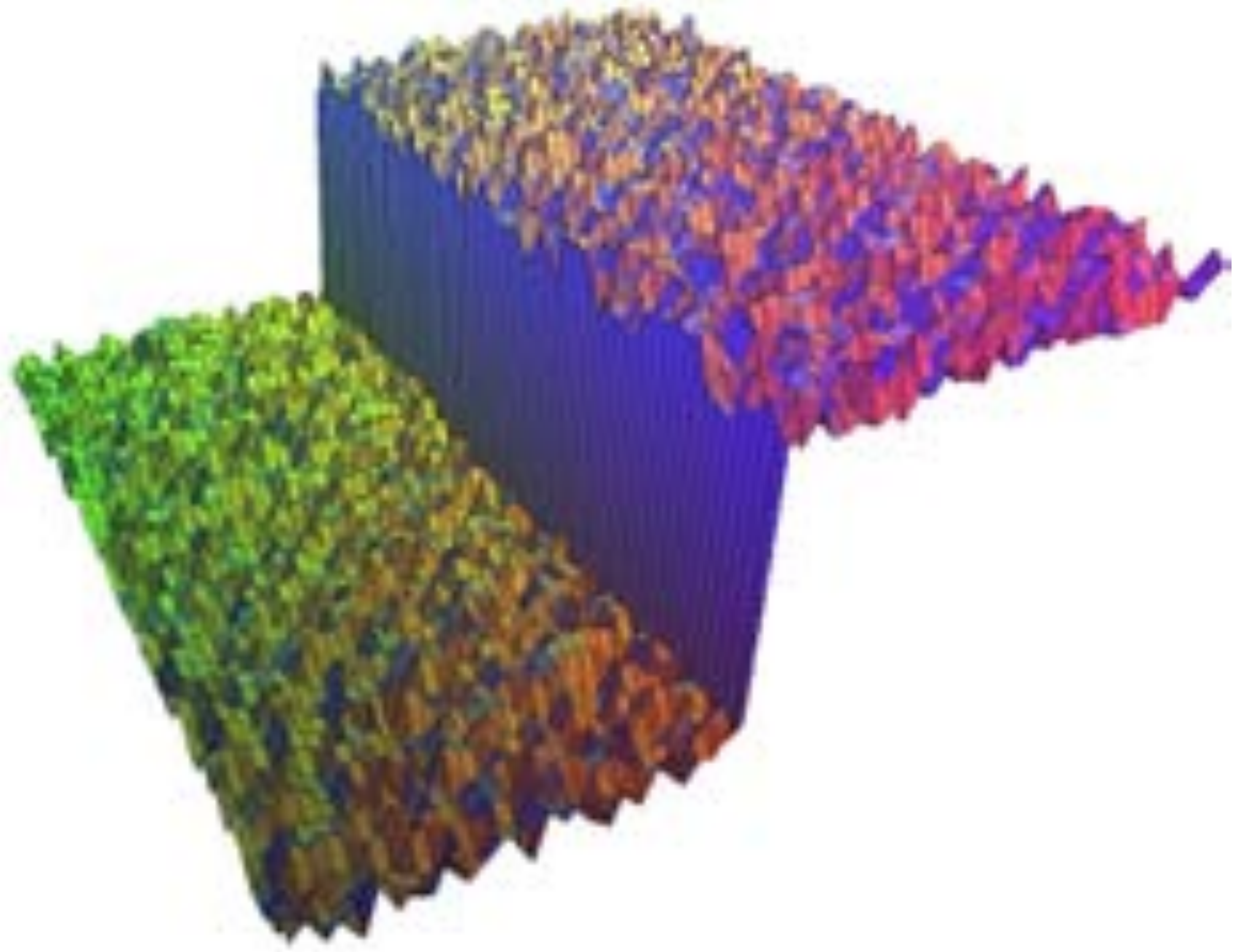
Lecture 5: Re-cap Median Filter

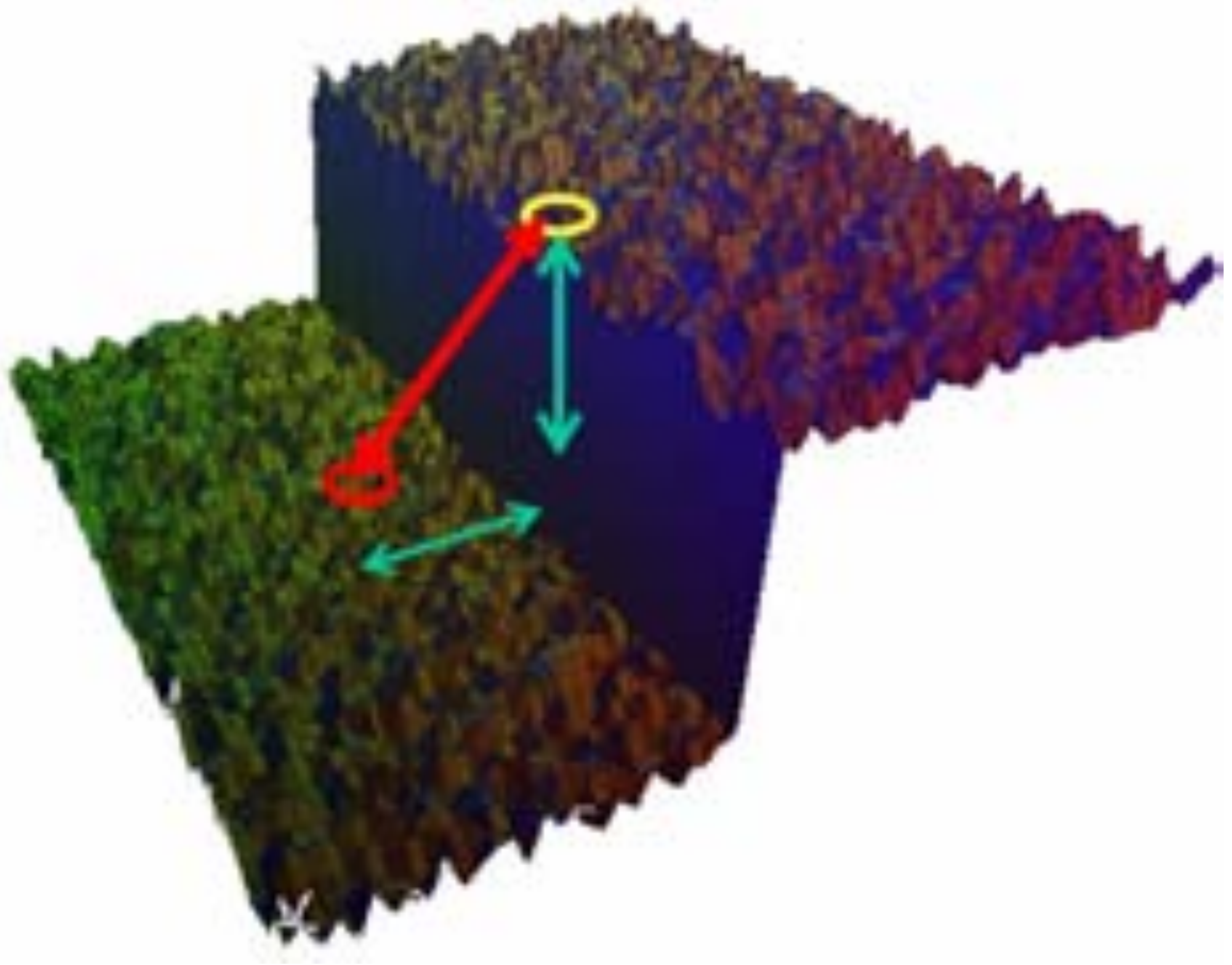
Effective at reducing certain kinds of noise, such as impulse noise (a.k.a ‘salt and pepper’ noise or ’shot’ noise)

The median filter forces points with distinct values to be more like their neighbors

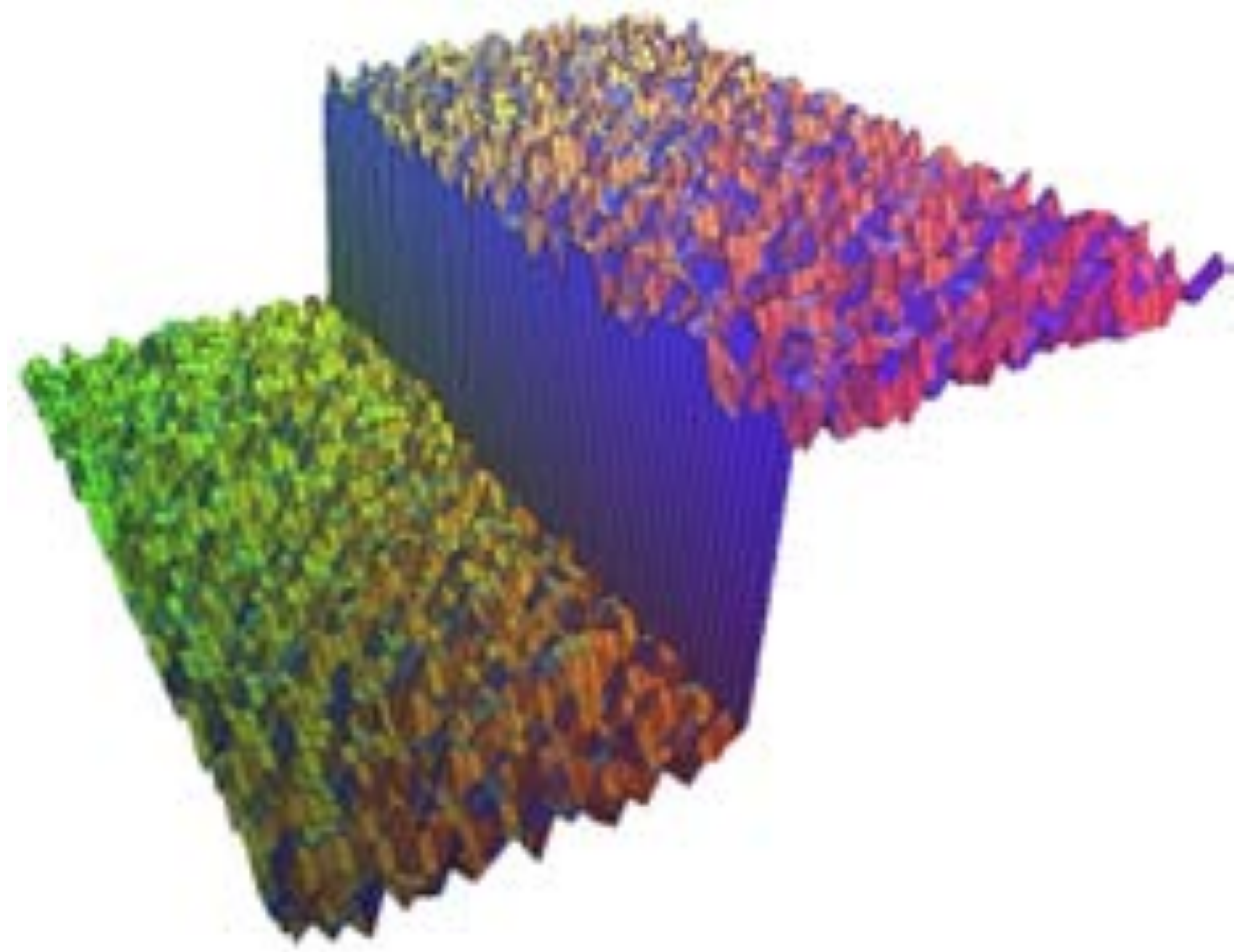
Image credit: https://en.wikipedia.org/wiki/Median_filter#/media/File:Medianfilterp.png

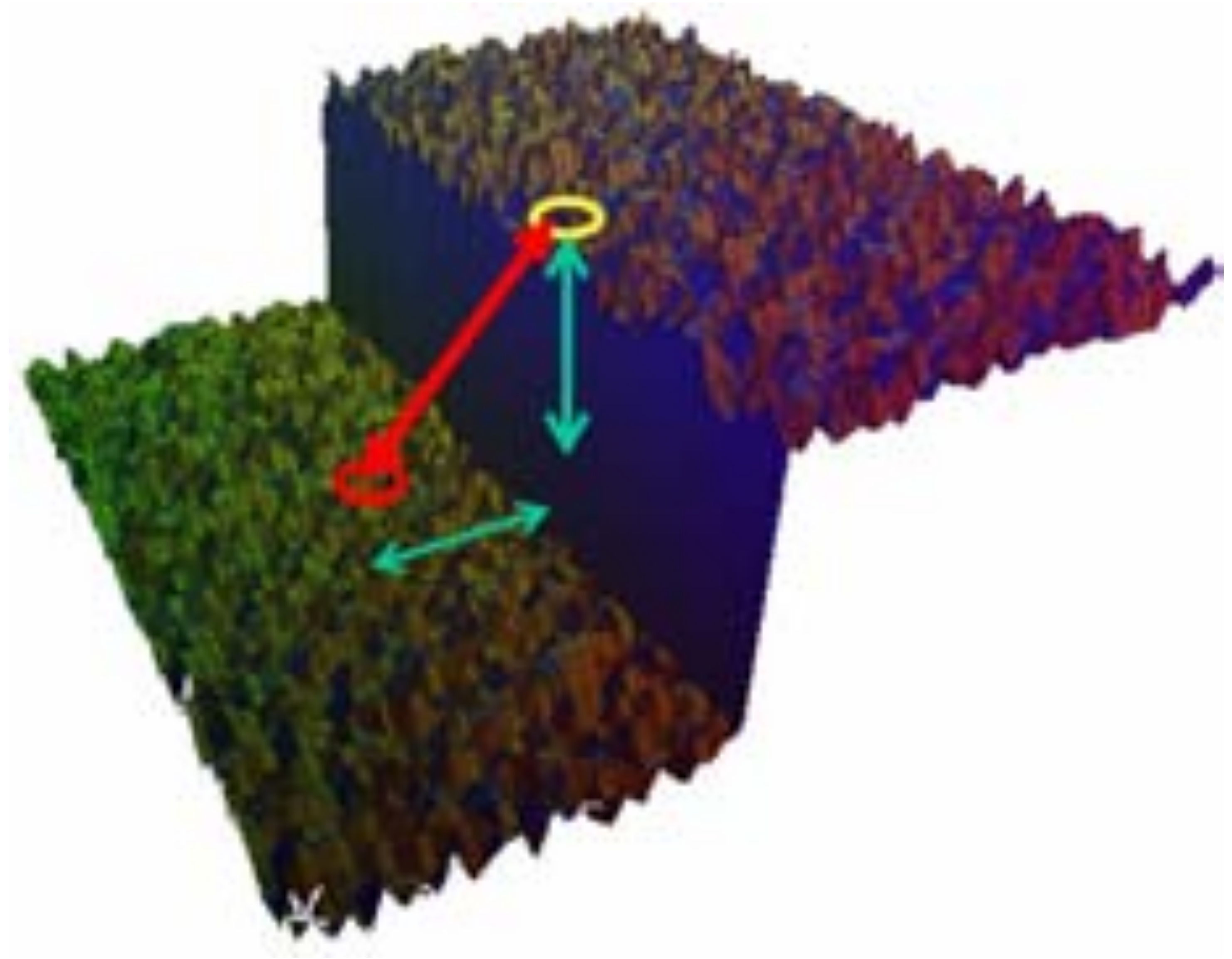
Lecture 5: Re-cap Bilateral Filter





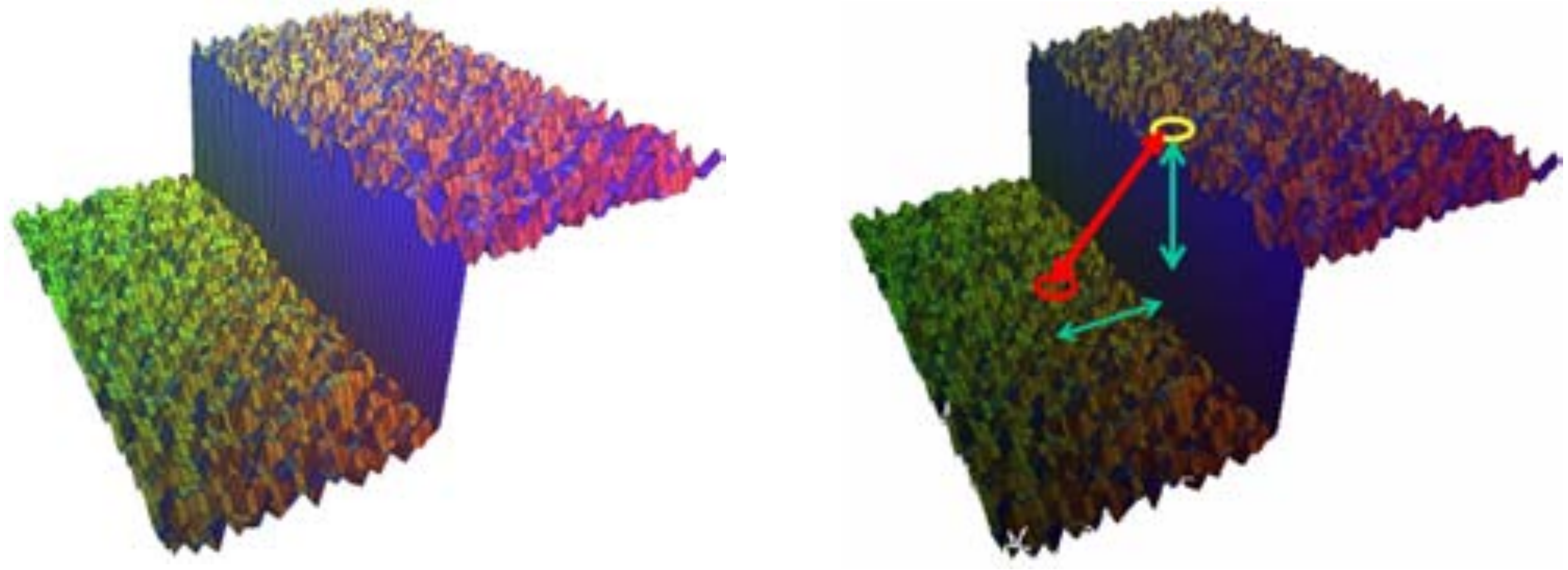
Lecture 5: Re-cap Bilateral Filter





Suppose we want to smooth a noisy step function

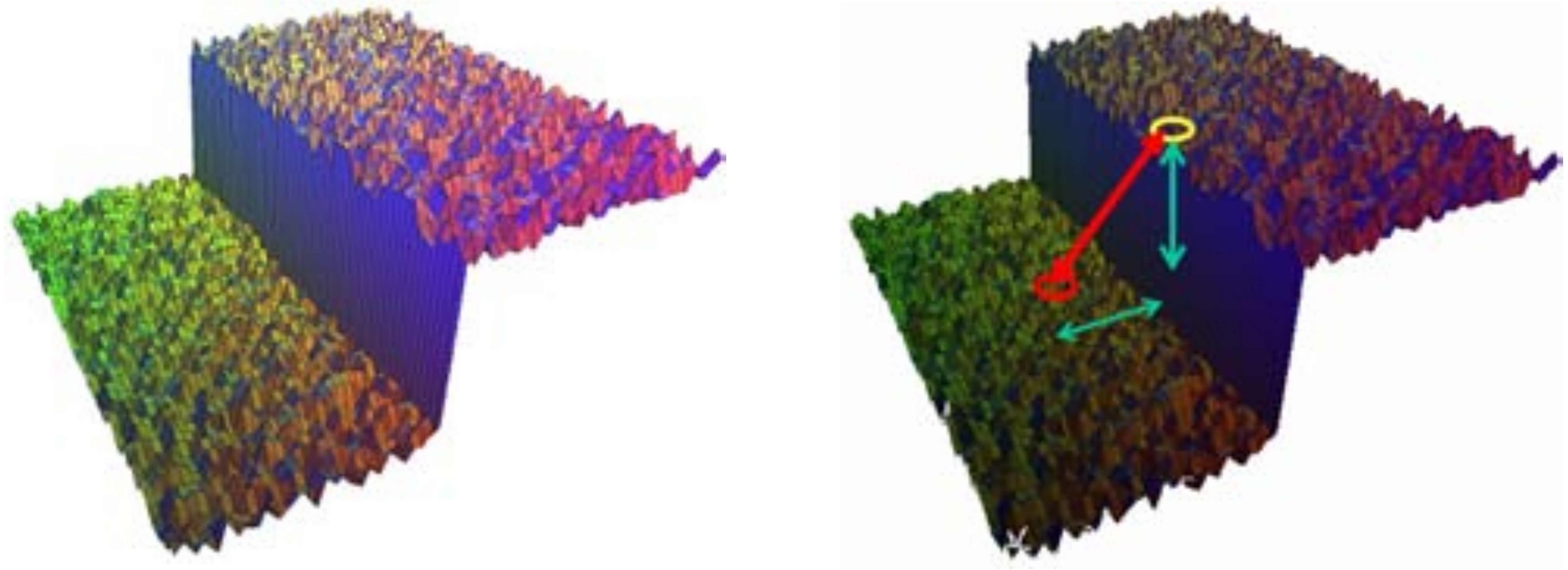
Lecture 5: Re-cap Bilateral Filter



Suppose we want to smooth a noisy step function

A Gaussian kernel performs a weighted average of points over a spatial neighbourhood..

Lecture 5: Re-cap Bilateral Filter

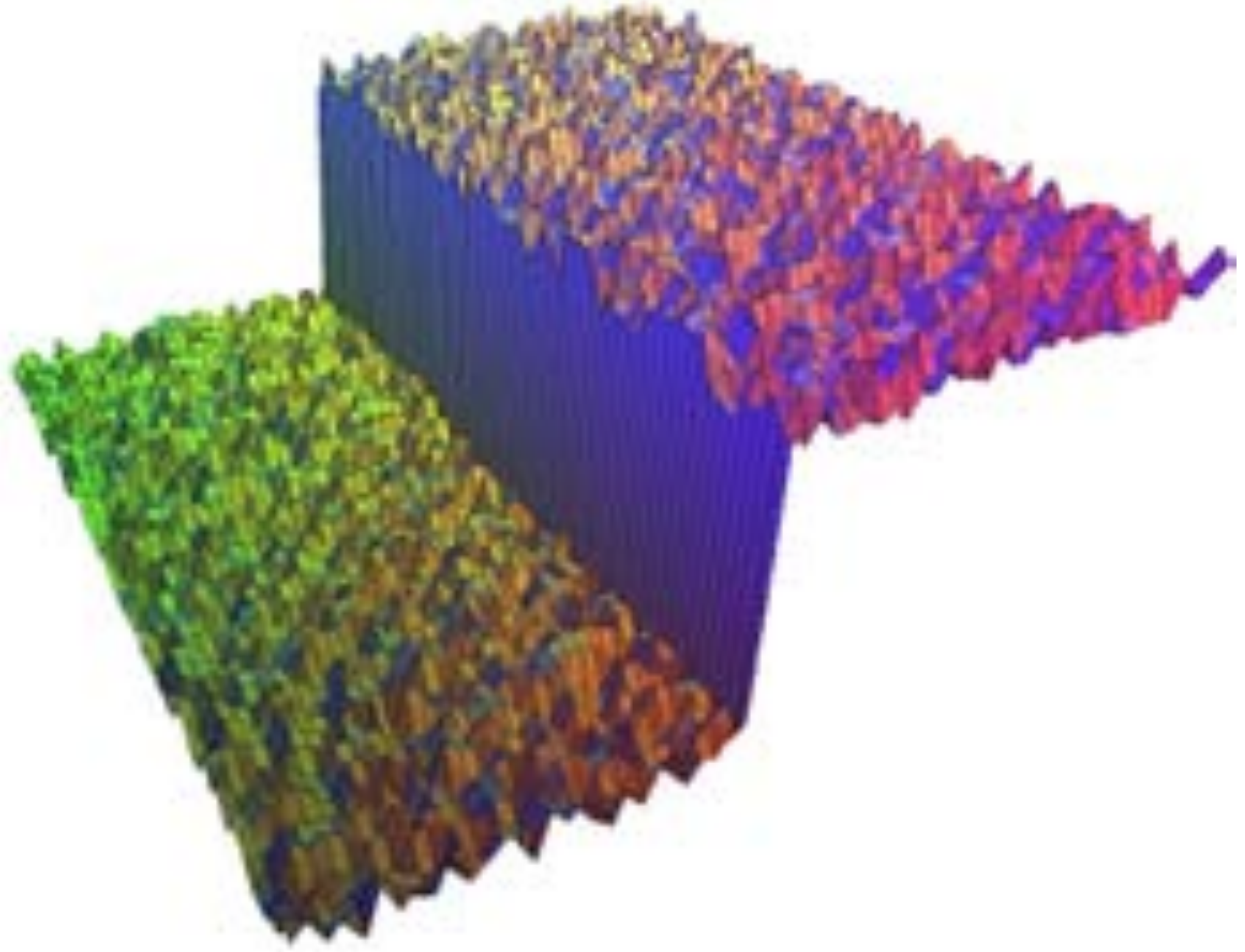


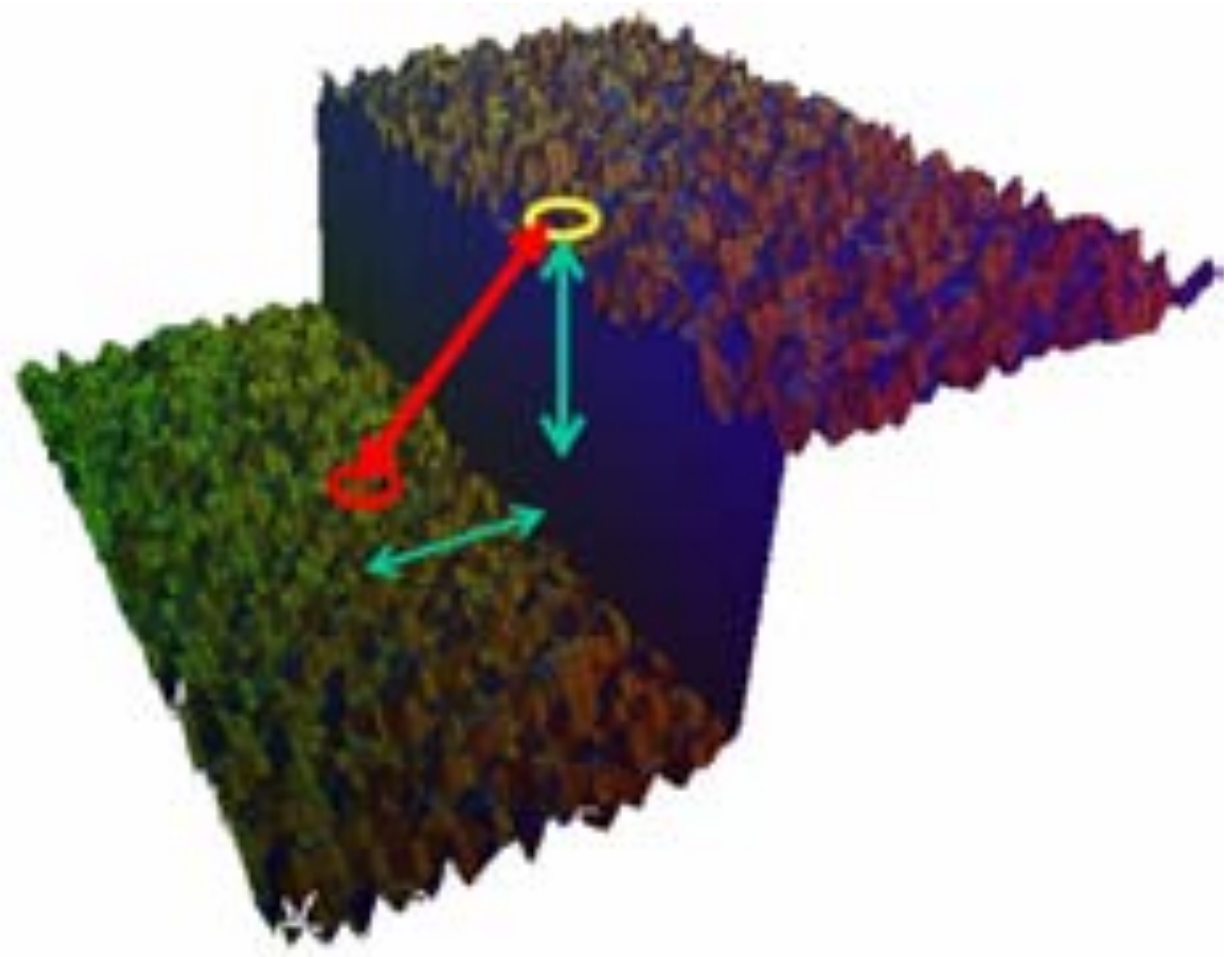
Suppose we want to smooth a noisy step function

A Gaussian kernel performs a weighted average of points over a spatial neighbourhood..

But this averages points both at the top and bottom of the step — blurring

Lecture 5: Re-cap Bilateral Filter





Suppose we want to smooth a noisy step function

A Gaussian kernel performs a weighted average of points over a spatial neighbourhood..

But this averages points both at the top and bottom of the step — blurring

Bilateral Filter idea: look at distances in **range** (value) as well as **space** x,y

Lecture 5: Re-cap Bilateral Filter

Gaussian filter: weights of neighbor at a spatial offset (x, y) away from the center pixel $I(X, Y)$ given by:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

(with appropriate normalization)

Bilateral filter: weights of neighbor at a spatial offset (x, y) away from the center pixel $I(X, Y)$ given by a product:

domain kernel	$\exp^{-\frac{x^2+y^2}{2\sigma_d^2}}$	$\exp^{-\frac{(I(X+x,Y+y)-I(X,Y))^2}{2\sigma_r^2}}$	**range** kernel

(with appropriate normalization)

Lecture 5: Re-cap Bilateral Filter Application: Denoising

Noisy Image

Gaussian Filter

Bilateral Filter

Slide Credit: Alexander Wong

Lecture 5: Re-cap Bilateral Filter Application: Cartooning

Original Image

After 5 iterations of **Bilateral** Filter

Slide Credit: Alexander Wong

Menu for Today (**September 23, 2024**)

Topics:

- **Sampling** theory
- **Nyquist** rate
- Color **Filter Arrays**
- **Image** encoding

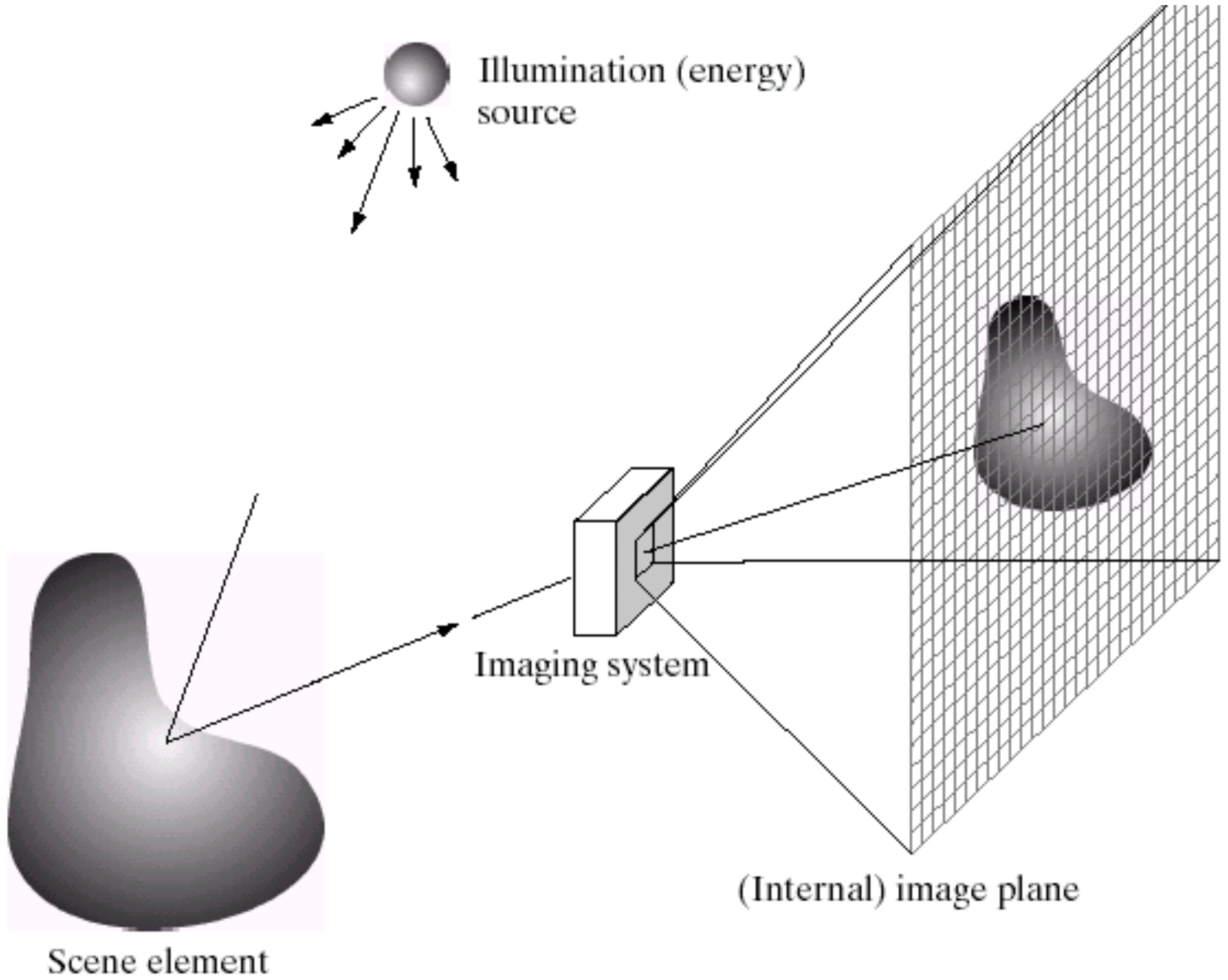
Readings:

- **Today's** Lecture: Szeliski 2.3, Forsyth & Ponce (2nd ed.) 4.5, 4.6

Reminders:

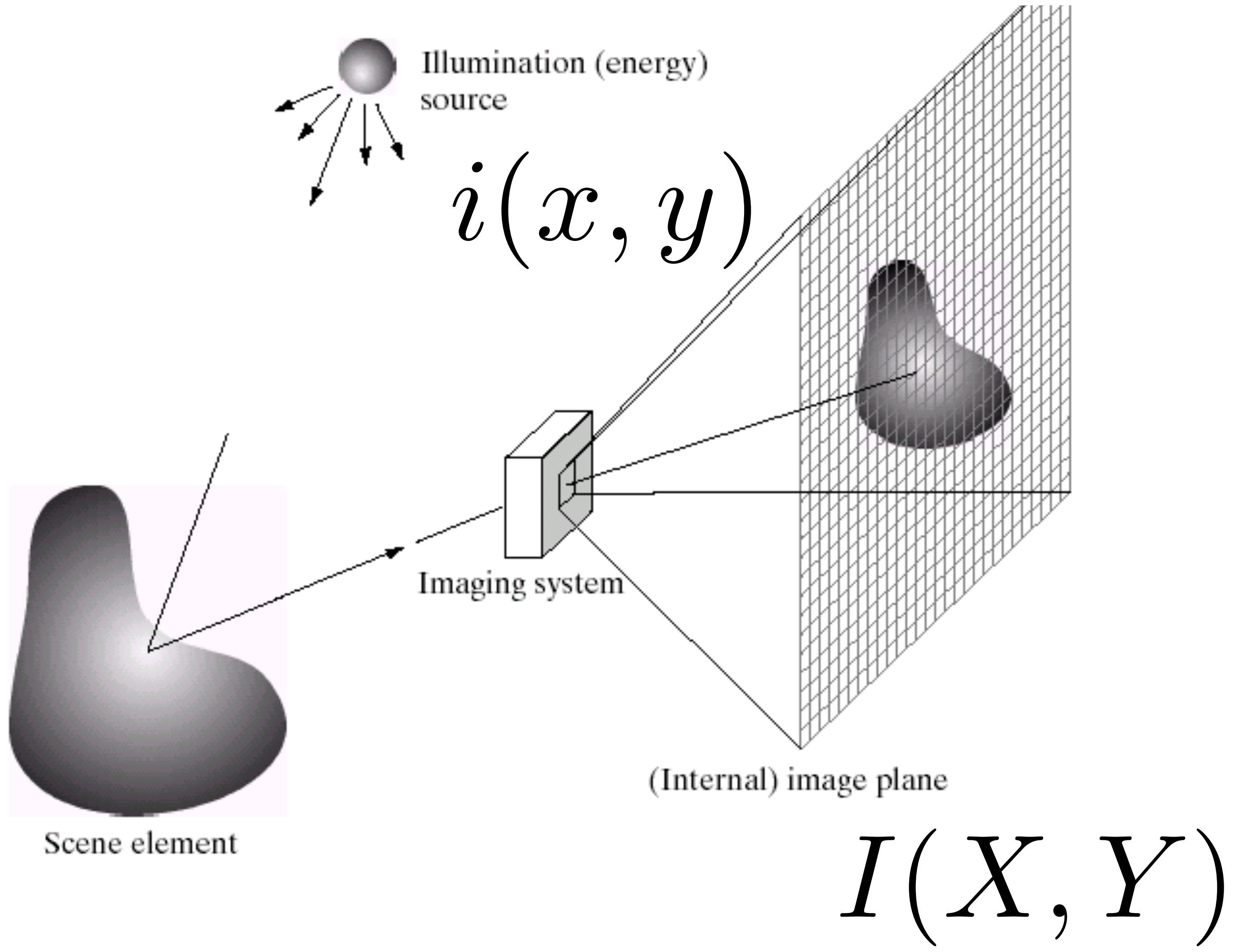
- **Assignment 1:** Image Filtering and Hybrid Images due **September 26th**

Reminder



Images are a **discrete**, or **sampled**, representation of a continuous world

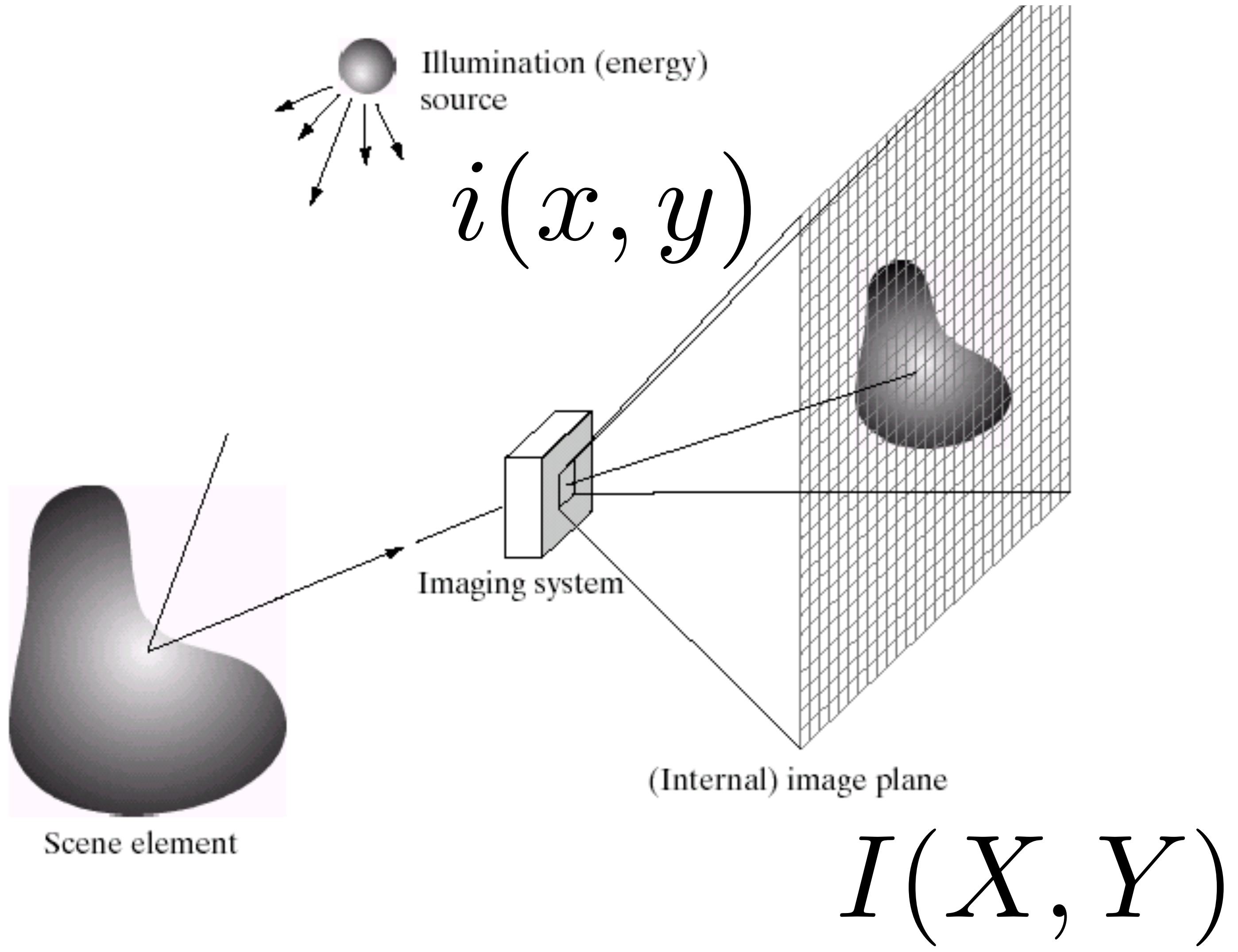
What is **Sampling**?



A **continuous function** $i(x, y, \lambda)$ is presented at the image sensor at each time instant

How do we convert this to a **digital signal** (array of numbers) $I(x, y, \lambda)$?

What is **Sampling**?



A **continuous function** $i(x, y, \lambda)$ is presented at the image sensor at each time instant

How do we convert this to a **digital signal** (array of numbers) $I(x, y, \lambda)$?

How can we **manipulate**, e.g., resample, this digital signal correctly?

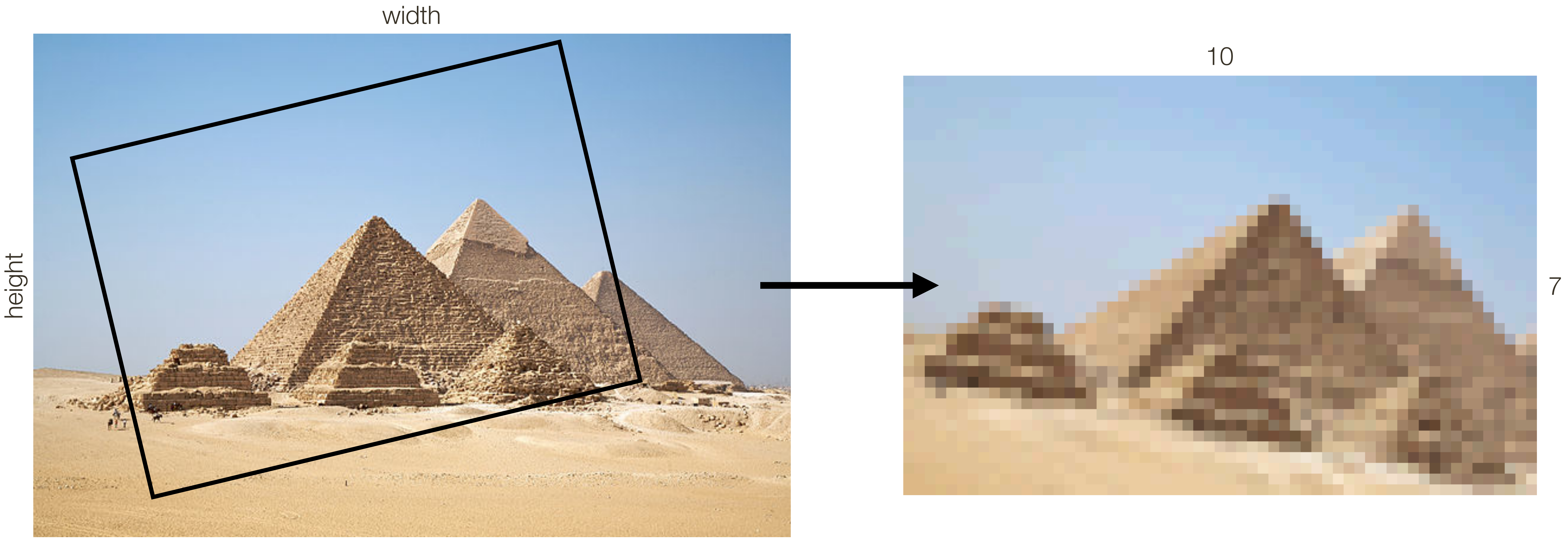
Resampling Images

width

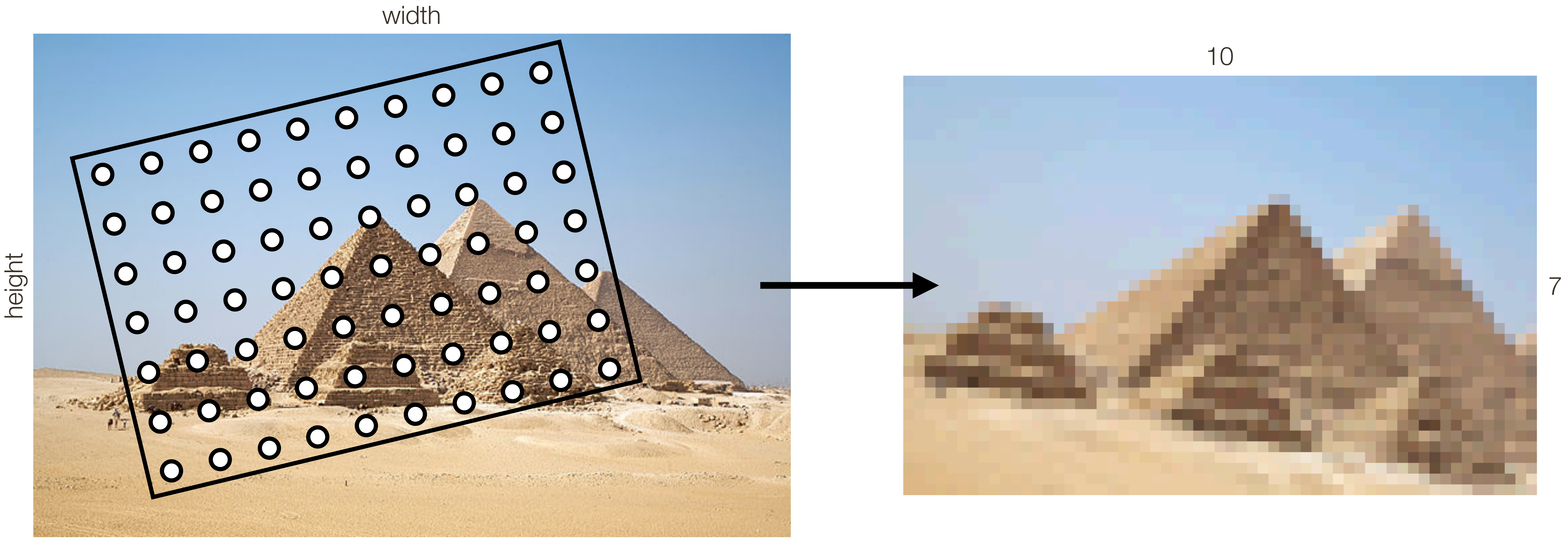
height

Resampling Images

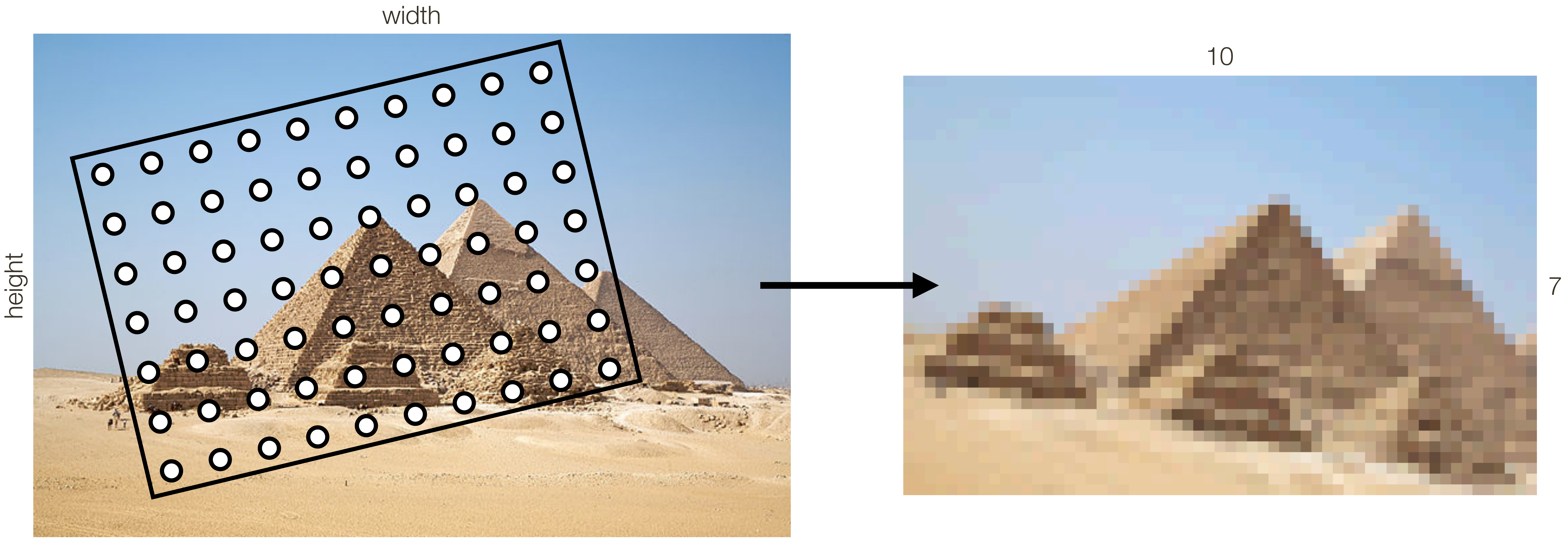
Resampling Images



Resampling Images



Resampling Images



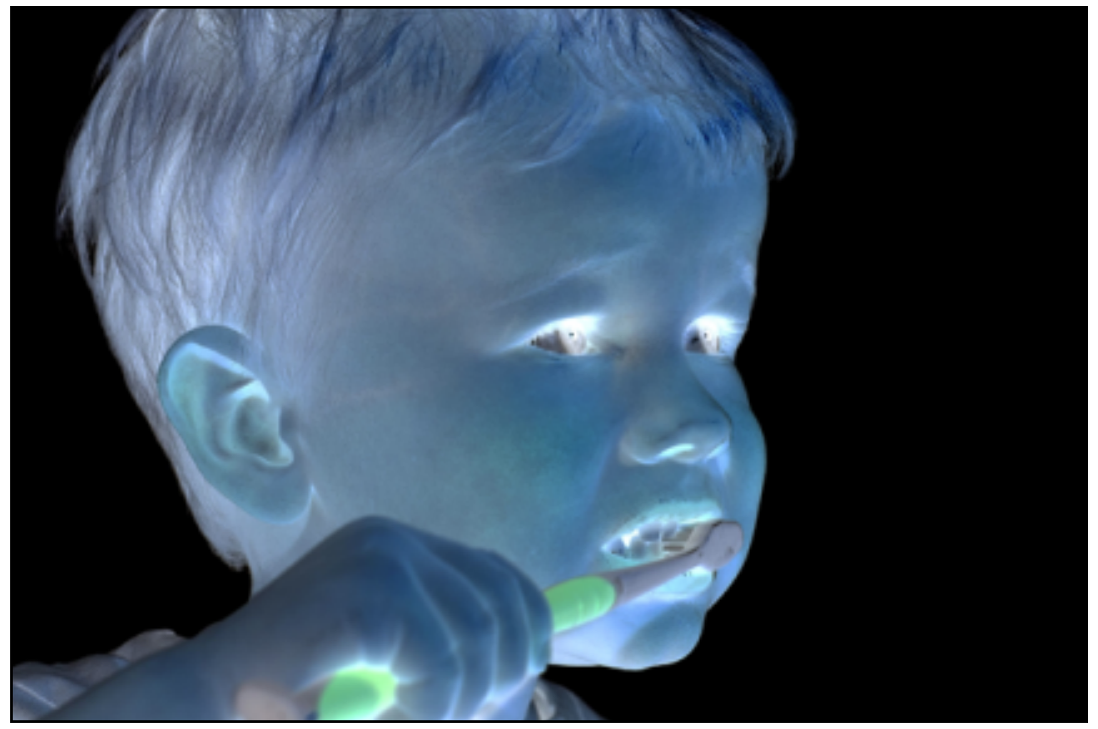
How do we correctly **generate samples** to resample or warp an image?

What types of **transformations** can we do?

$I(X,Y)$

Filtering

$I'(X,Y)$

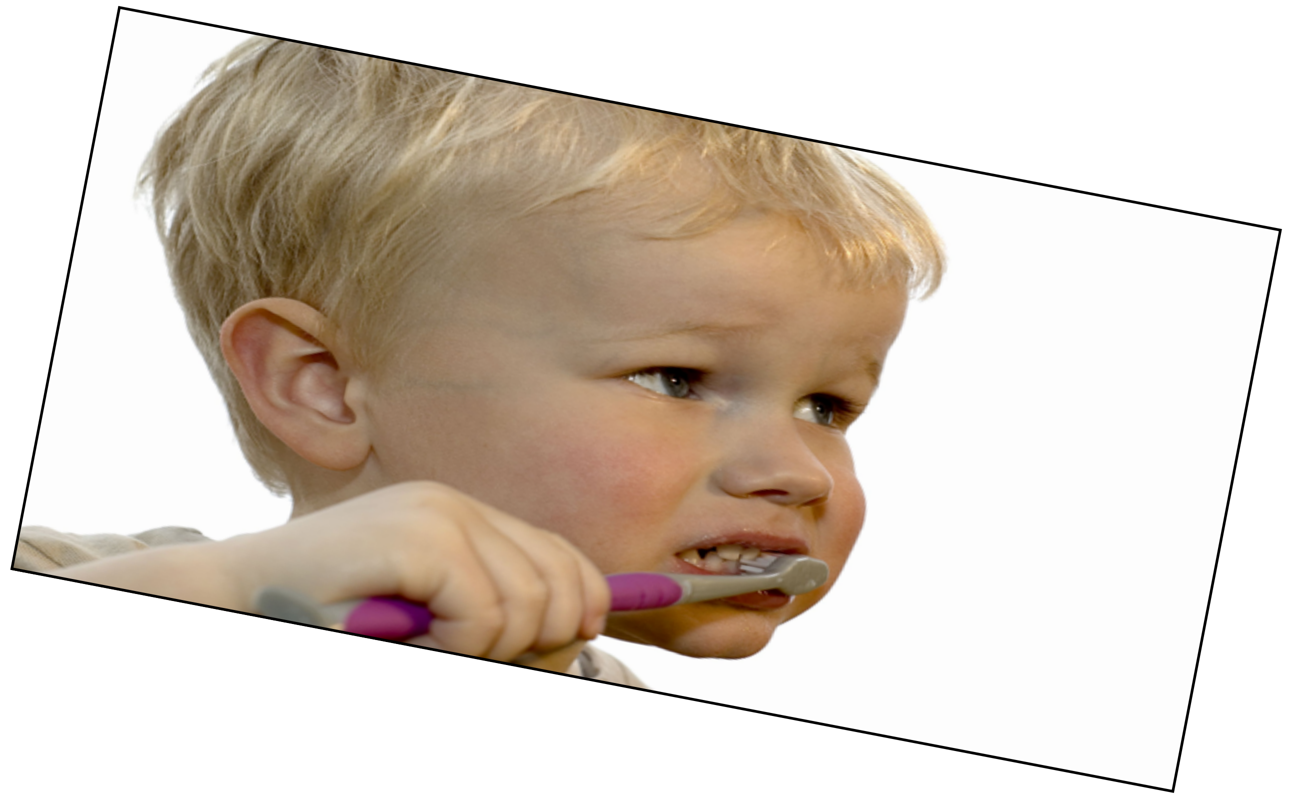


changes range of image function

$I(X,Y)$

Warping

$I'(X,Y)$



changes domain of image function

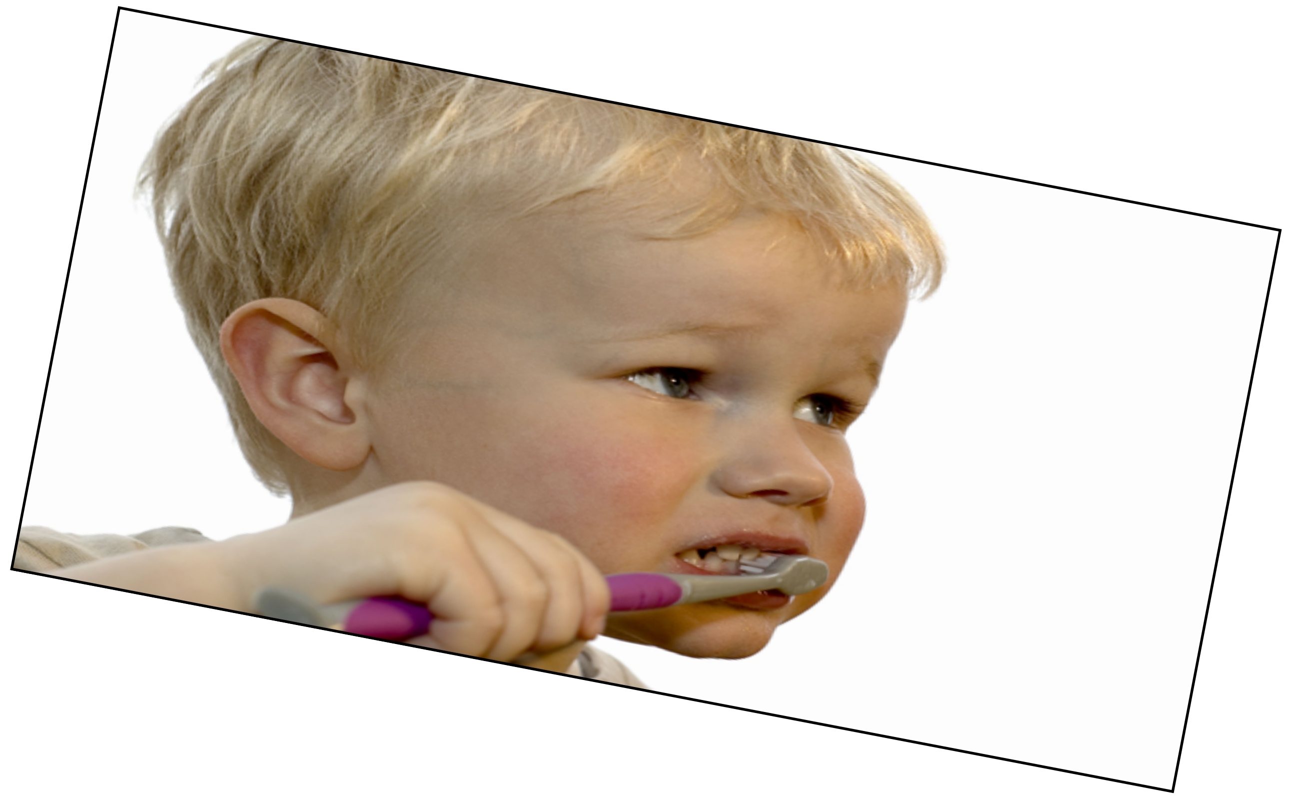
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

What types of **transformations** can we do?

$I(X, Y)$

Warping

$I'(X, Y)$

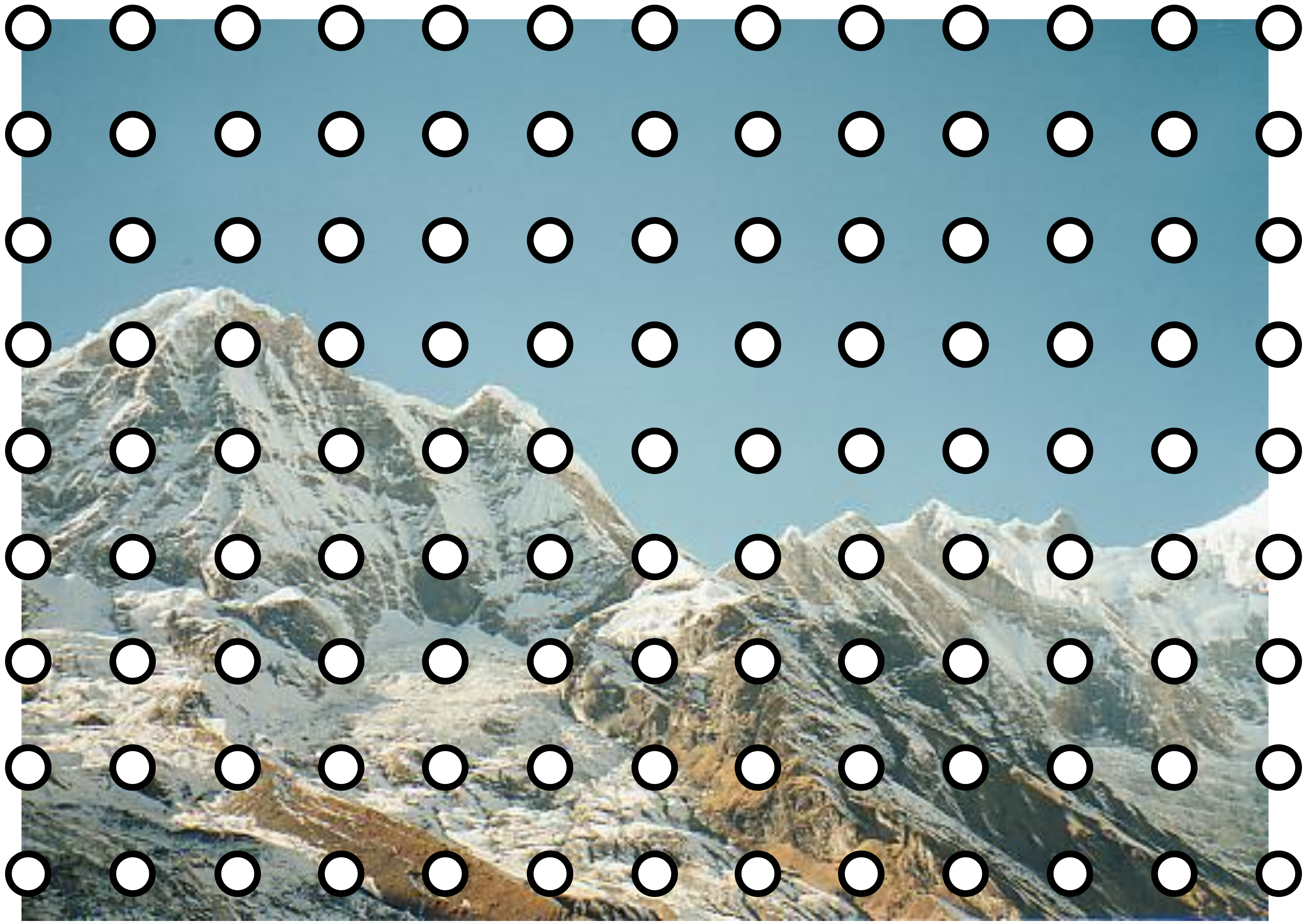


changes domain of image function

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Resampling Images

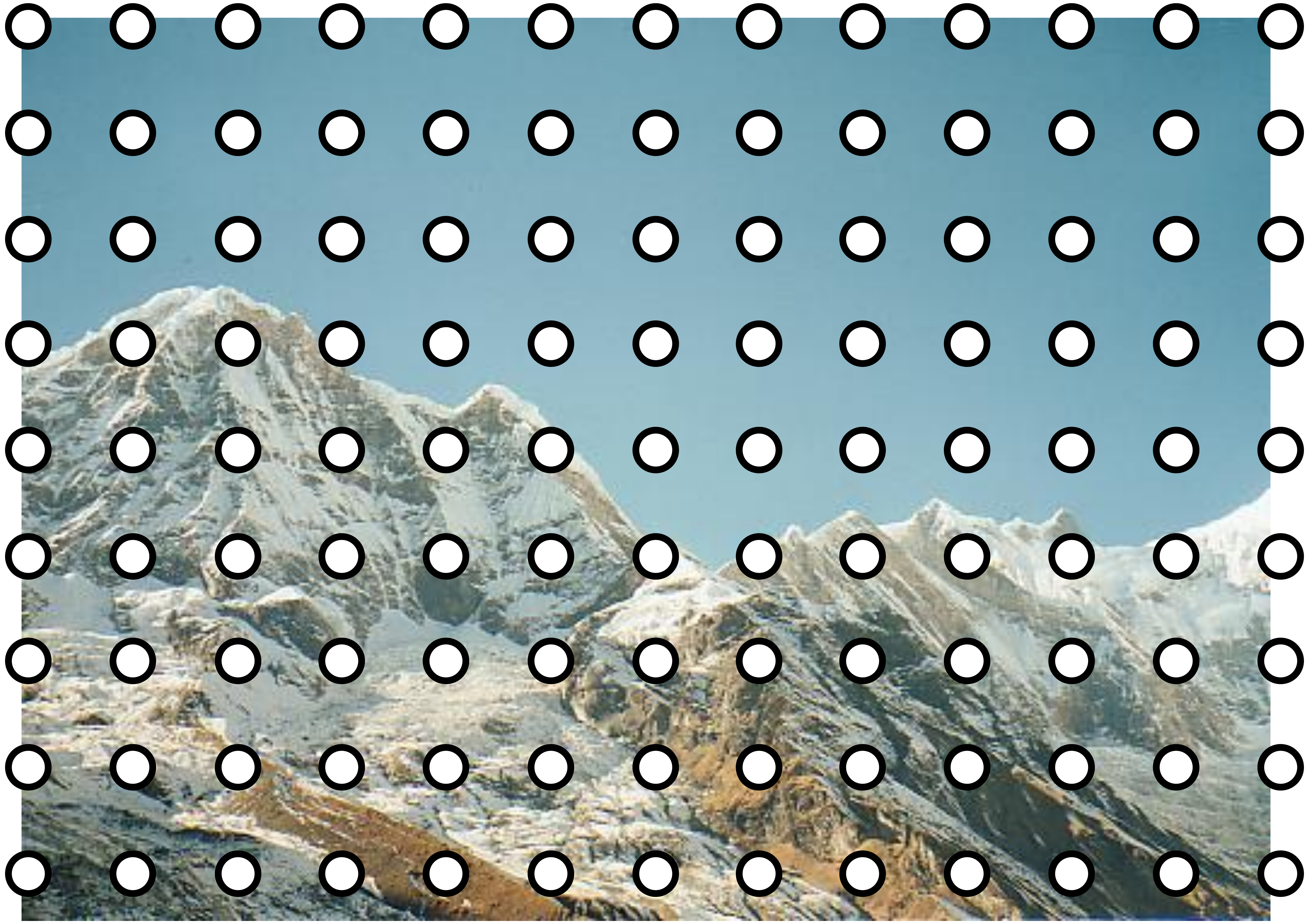
Goal: Resample the image to get a lower resolution counterpart



What is the simplest way to do this (e.g., produce image 1/5 of original size)?

Resampling Images

Goal: Resample the image to get a lower resolution counterpart



Naive Method: Form new image by taking every n-th pixel of the original image

Resampling Images

Sampling every 5-th pixel, while shifting rightwards one pixel at a time

Resampling Images

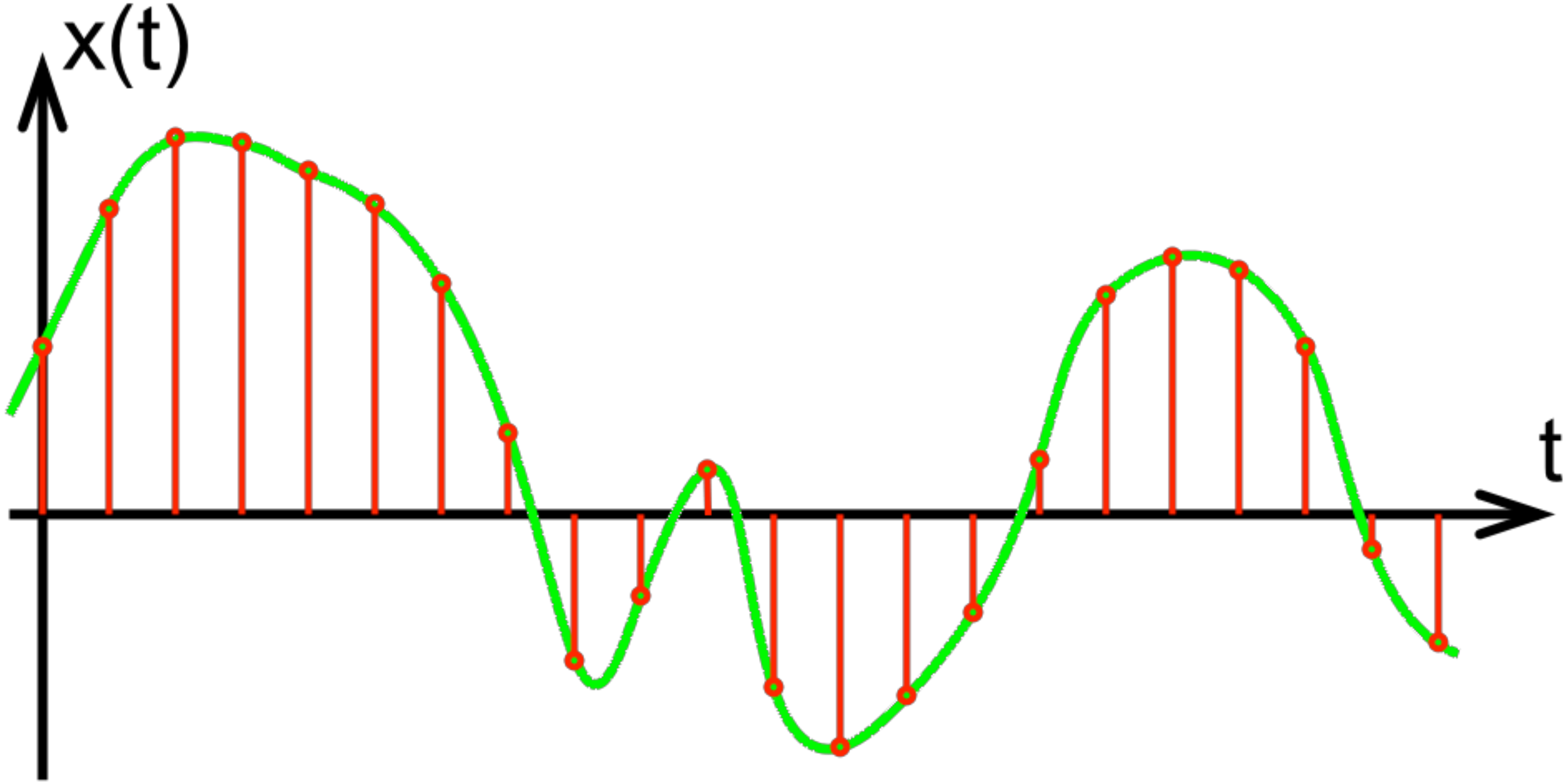
Sampling every 5-th pixel, while shifting rightwards one pixel at a time

Resampling Images

Sampling every 5-th pixel, while shifting rightwards one pixel at a time

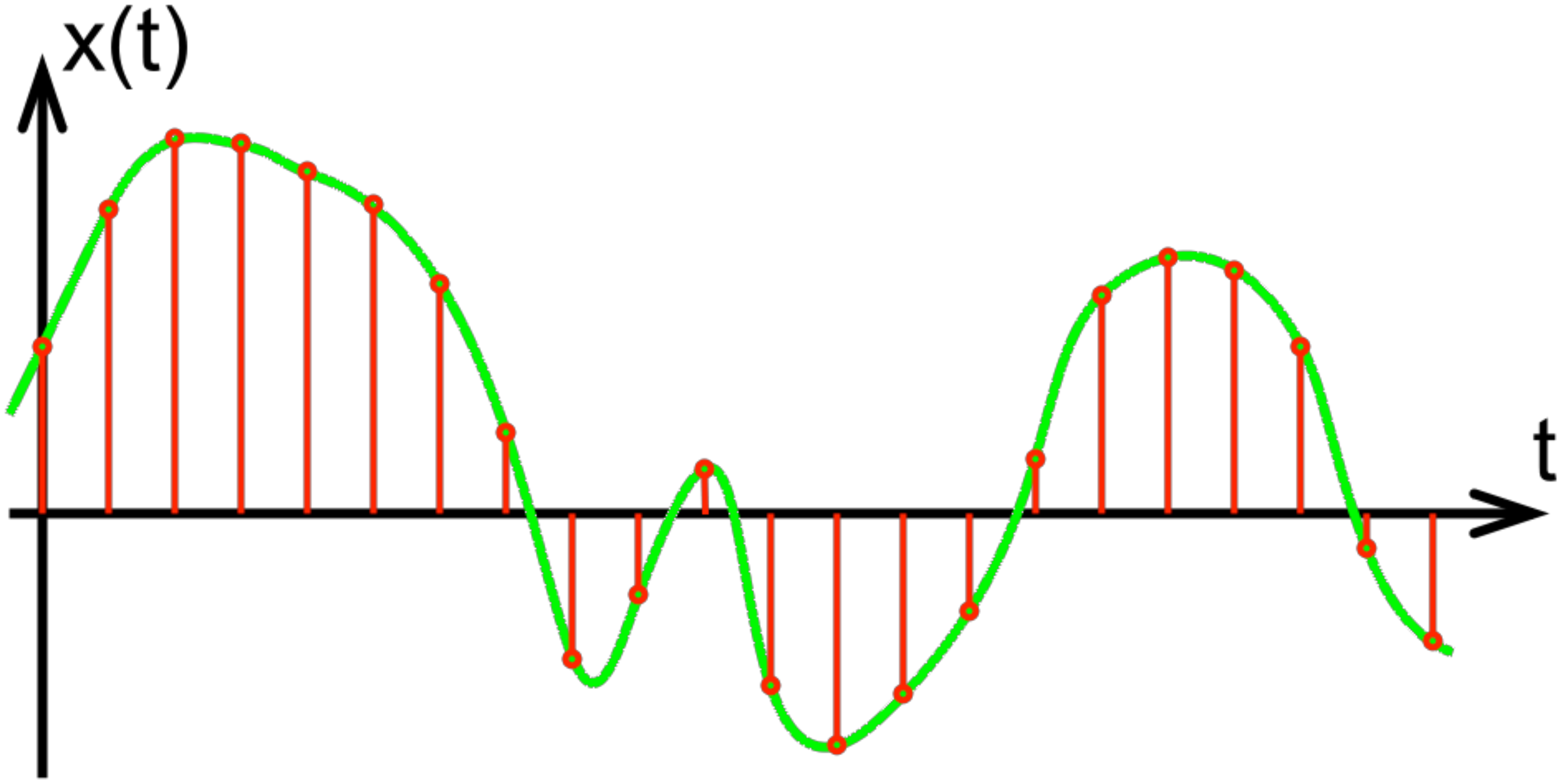
What's wrong with this method?

Example: Audio Sampling



Question: What choice/parameters do we have when sampling audio signal?

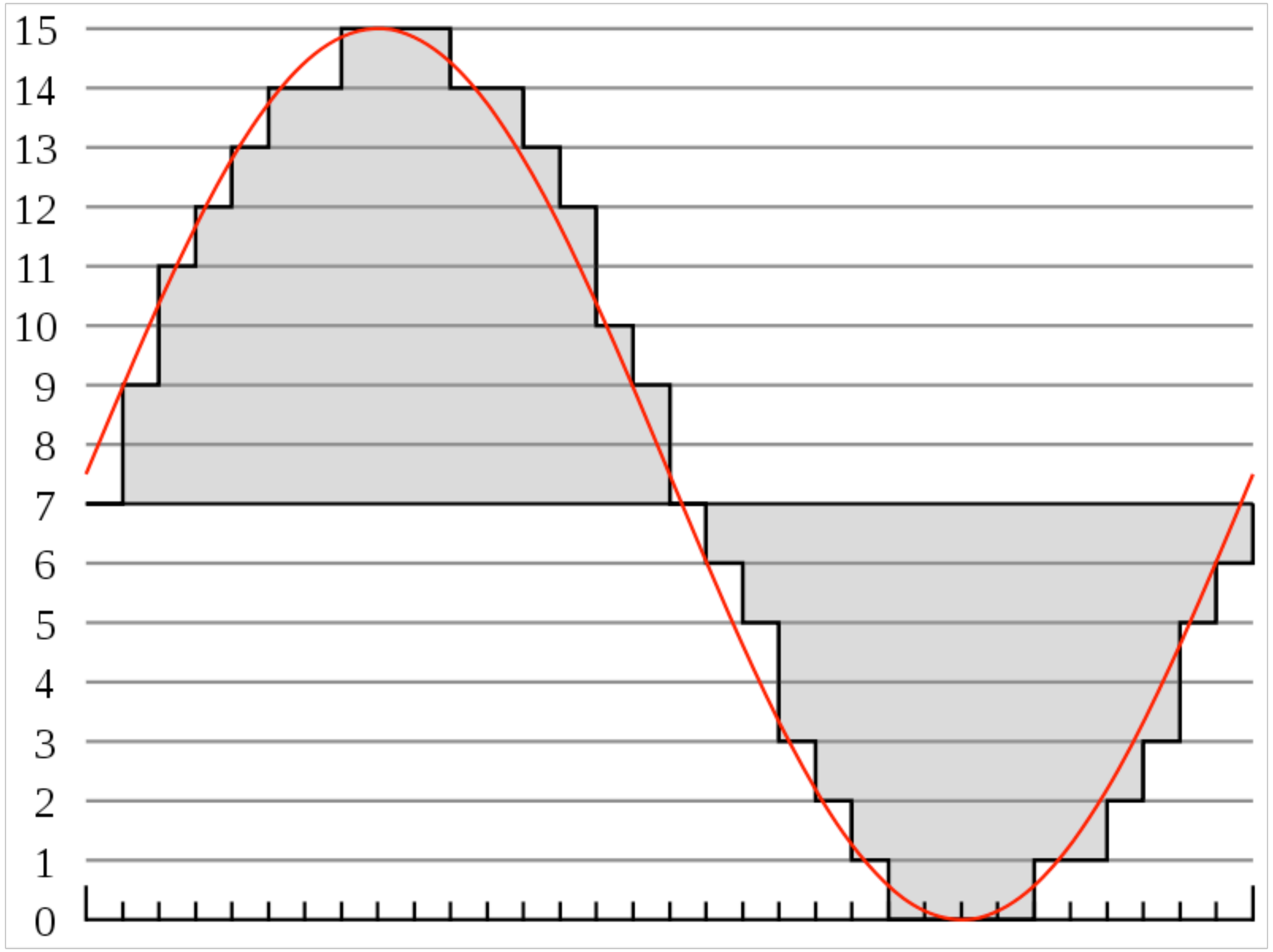
Example: Audio Sampling



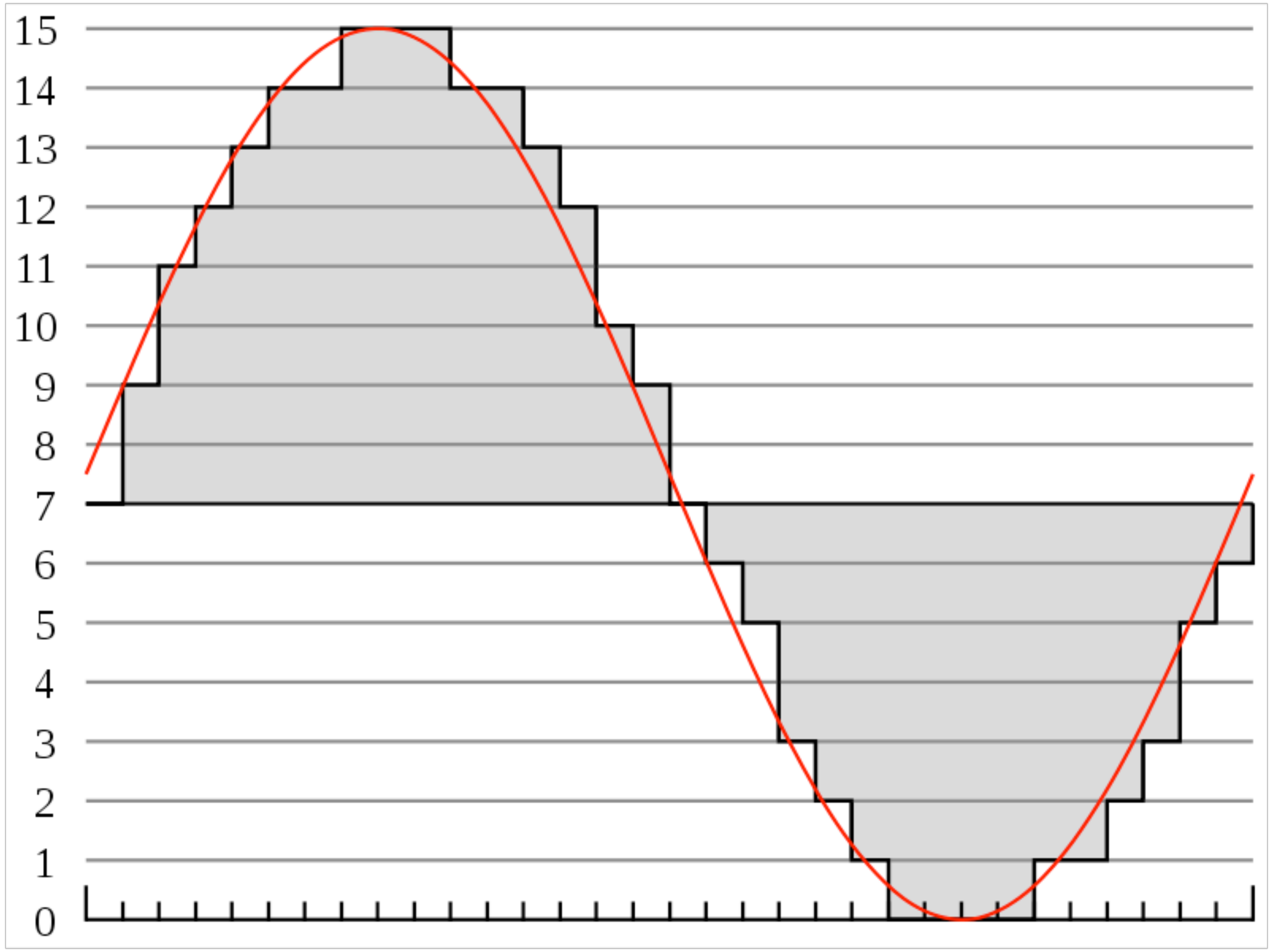
Question: What choice/parameters do we have when sampling audio signal?

Sampling rate and bit depth, e.g., 44.1 kHz (samples/second), 16 bits/sample

Example: Audio Sampling



Example: Audio Sampling



Quantization noise / error is the difference between black and red curves

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

$\downarrow 2$ $\downarrow 4$ $\downarrow 8$

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

$\downarrow 2$ $\downarrow 4$ $\downarrow 8$

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

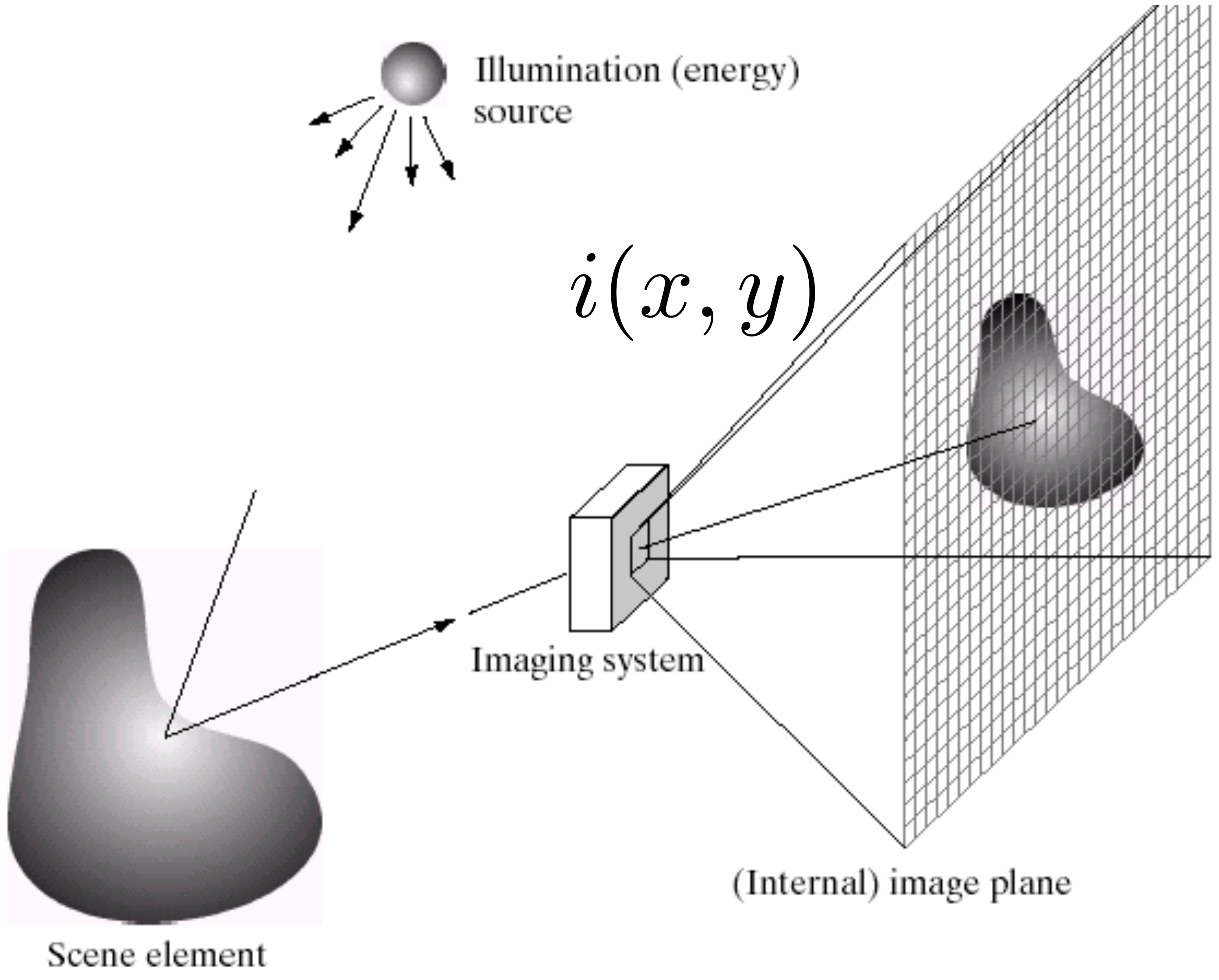
- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

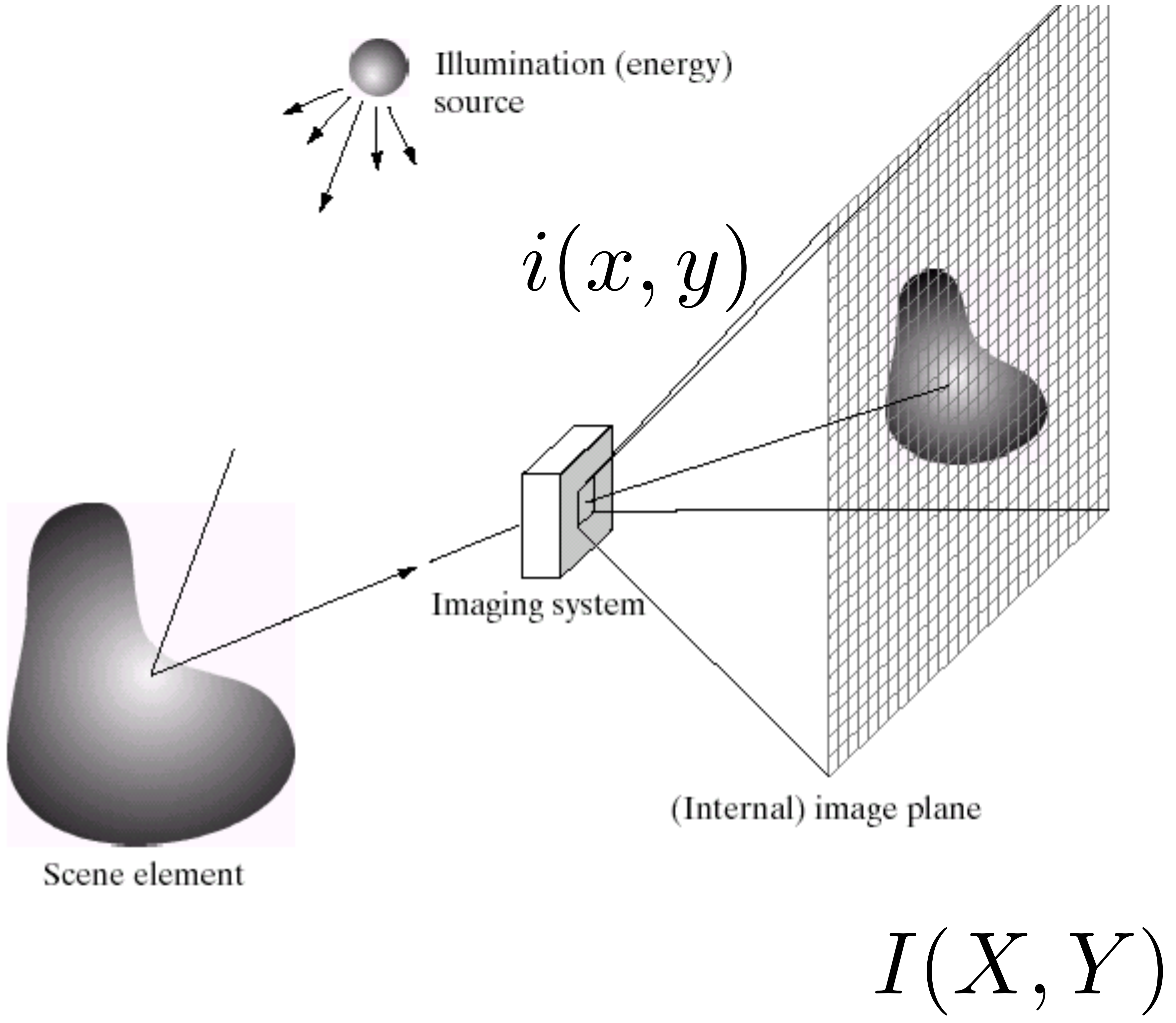
↓2 ↓4 ↓8

Example: Image Sampling



$$I(X, Y)$$

Example: Image Sampling



Sampling rate and bit depth (e.g., 8-bits)

Continuous Case: Observations

— $i(x, y)$ is a **real-valued function** of **real spatial variables**, x and y

— $i(x, y)$ is **bounded above and below**. That is

$$0 \leq i(x, y) \leq M$$

for some maximum brightness M

Continuous Case: Observations

- $i(x, y)$ is a **real-valued function** of **real spatial variables**, x and y

- $i(x, y)$ is **bounded above and below**. That is

$$0 \leq i(x, y) \leq M$$

  for some maximum brightness M

- $i(x, y)$ is **bounded in extent**. That is, $i(x, y)$ is non-zero (i.e., strictly positive) over, at most, a bounded region

Pixel Bit Rate

Recall: $0 \leq i(x, y) \leq M$

We divide the range $[0, M]$ into a finite number of equivalence classes. This is called **quantization**.

The values are called **grey-levels**.

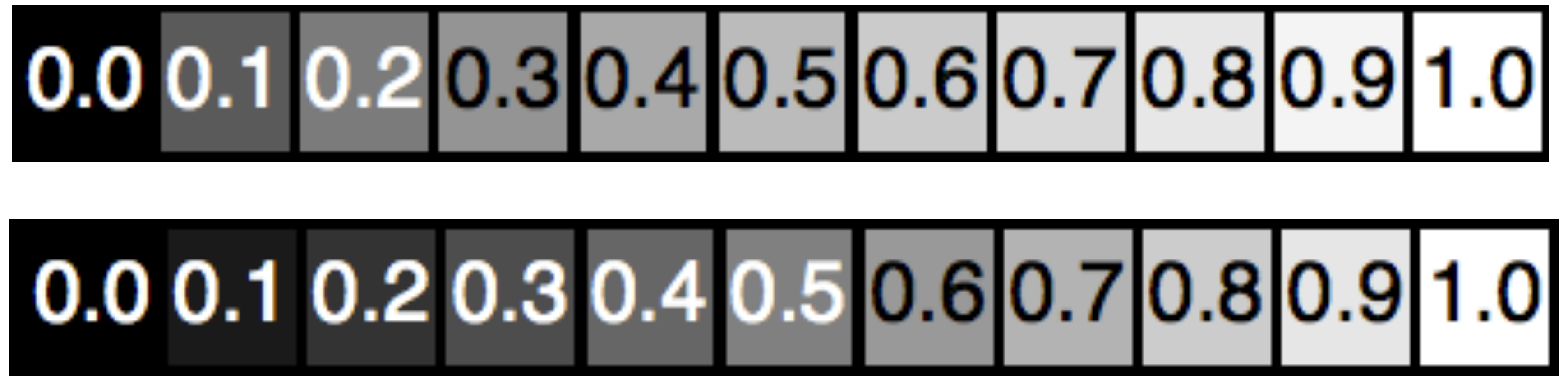
Suppose n bits-per-pixel are available. One can divide the range $[0, M]$ into evenly spaced intervals.

Typically $n = 8$ resulting in grey-levels in the range $[0, 255]$

Pixel Bit Rate

linear luminance (raw)

equal brightness steps



Recall: $0 \leq i(x, y) \leq M$

We divide the range $[0, M]$ into a finite number of equivalence classes. This is called **quantization**.

The values are called **grey-levels**.

Suppose n bits-per-pixel are available. One can divide the range $[0, M]$ into evenly spaced intervals.

Typically $n = 8$ resulting in grey-levels in the range $[0, 255]$

Sampling Theory (informal)

Question: When is $I(X, Y)$ an exact characterization of $i(x, y)$?

Sampling Theory (informal)

Question: When is $I(X,Y)$ an exact characterization of $i(x,y)$?

Question (modified): When can we reconstruct $i(x,y)$ exactly from $I(X,Y)$?

Sampling Theory (informal)

Question: When is $I(X, Y)$ an exact characterization of $i(x, y)$?

Question (modified): When can we reconstruct $i(x, y)$ exactly from $I(X, Y)$?

Intuition: Reconstruction involves some kind of **interpolation**

Sampling Theory (informal)

Question: When is $I(X,Y)$ an exact characterization of $i(x,y)$?

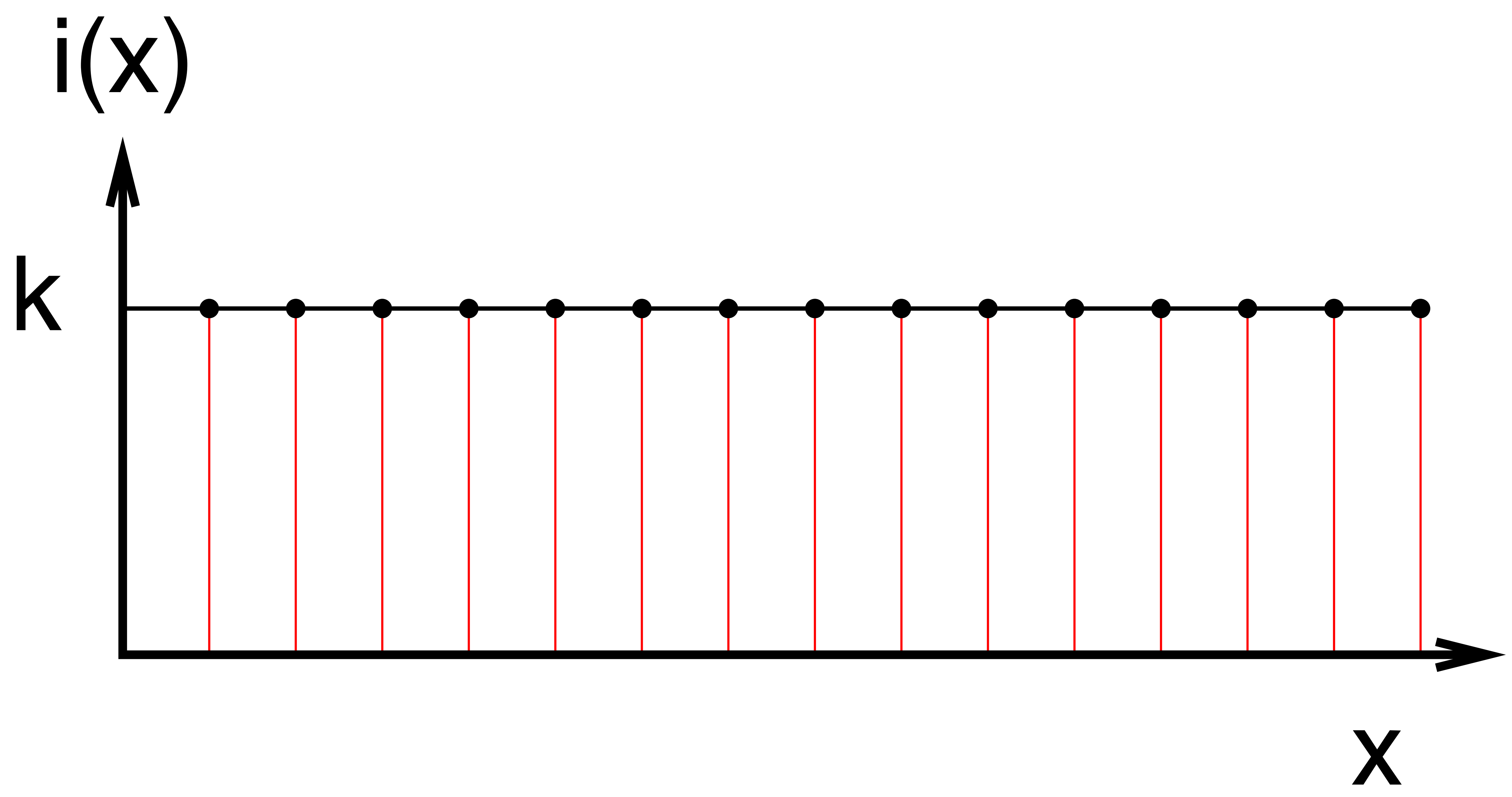
Question (modified): When can we reconstruct $i(x,y)$ exactly from $I(X,Y)$?

Intuition: Reconstruction involves some kind of **interpolation**

Heuristic: When in doubt, consider simple cases

Sampling Theory (informal)

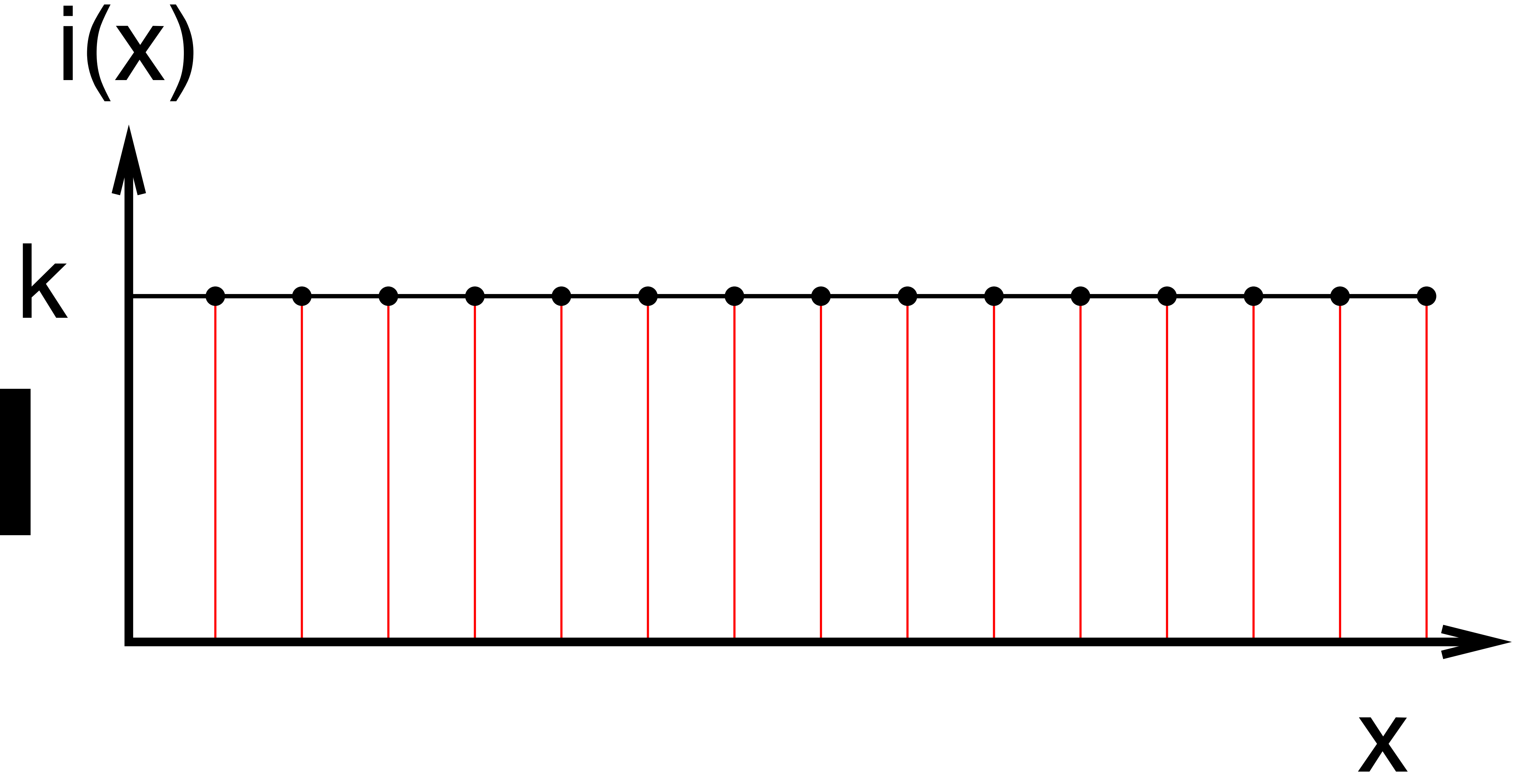
Case 0: Suppose $i(x, y) = k$ (with k being one of our gray levels)



Note: we use equidistant sampling at integer values for convenience, in general, sampling doesn't need to be equidistant

Sampling Theory (informal)

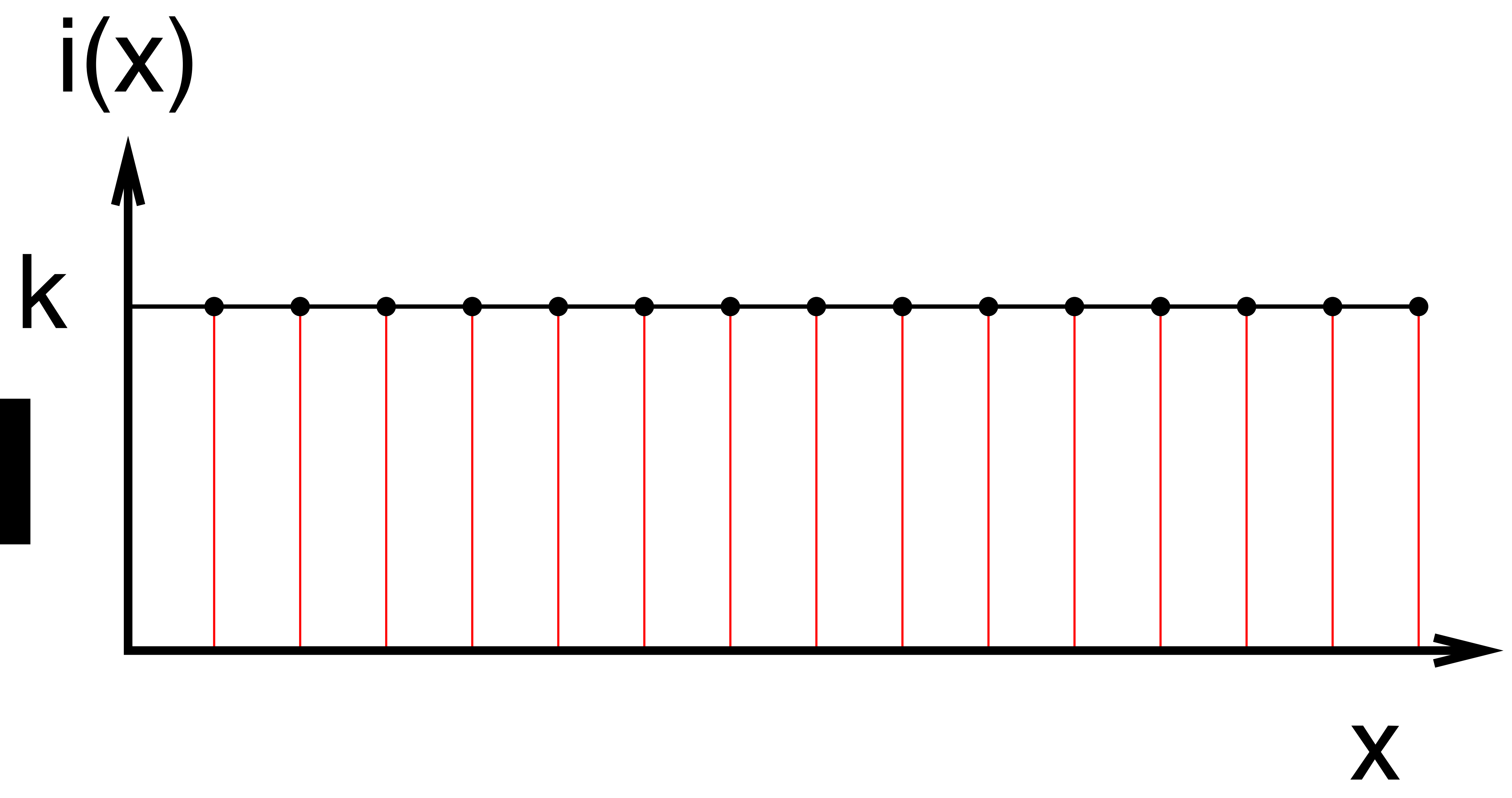
Case 0: Suppose $i(x, y) = k$ (with k being one of our gray levels)



This is **easy**!

Sampling Theory (informal)

Case 0: Suppose $i(x, y) = k$ (with k being one of our gray levels)

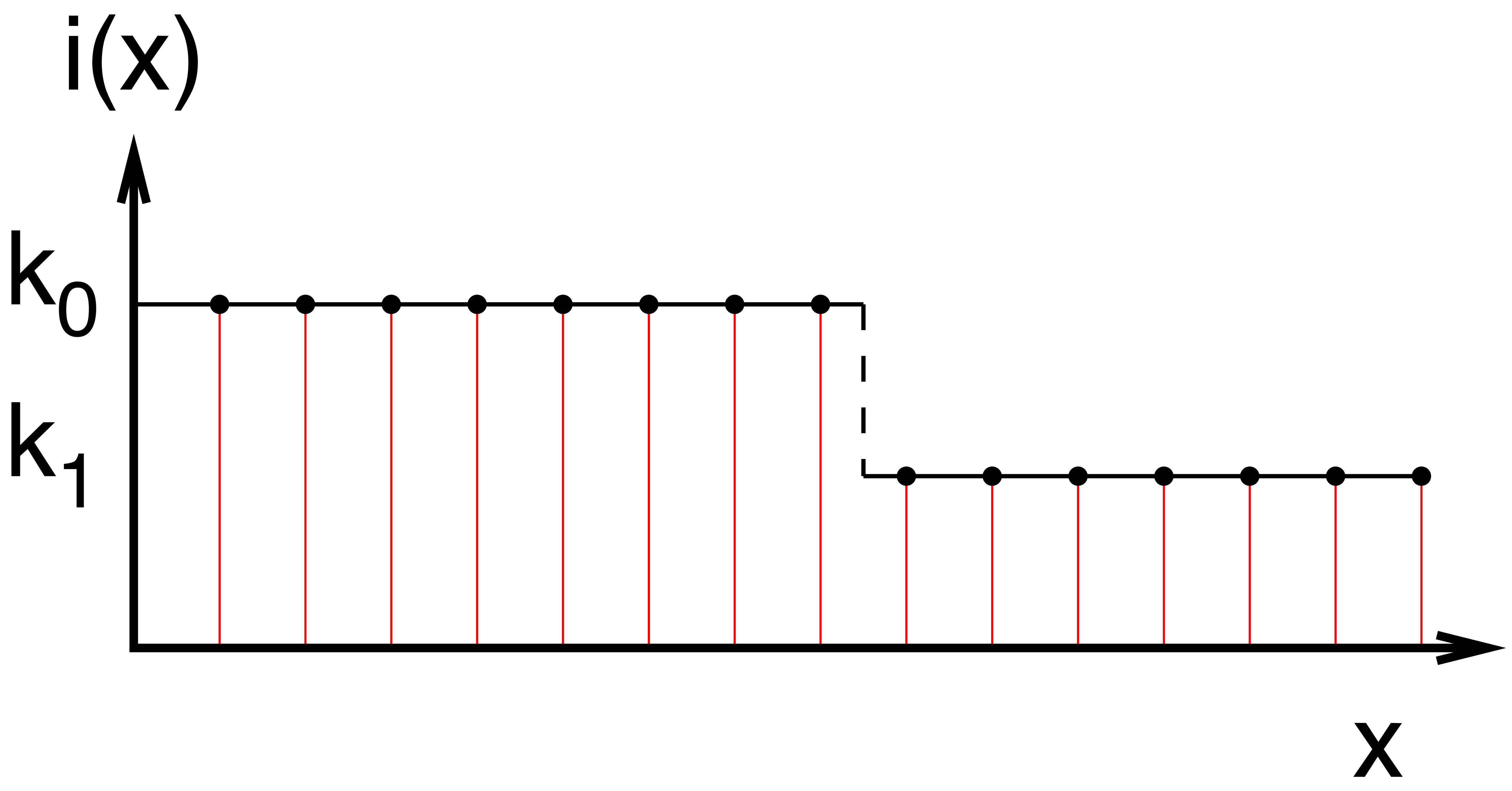


This is **easy**!

$I(X, Y) = k$. Any standard interpolation function would give $i(x, y) = k$ for non-integer x and y (irrespective oh how coarse the sampling is)

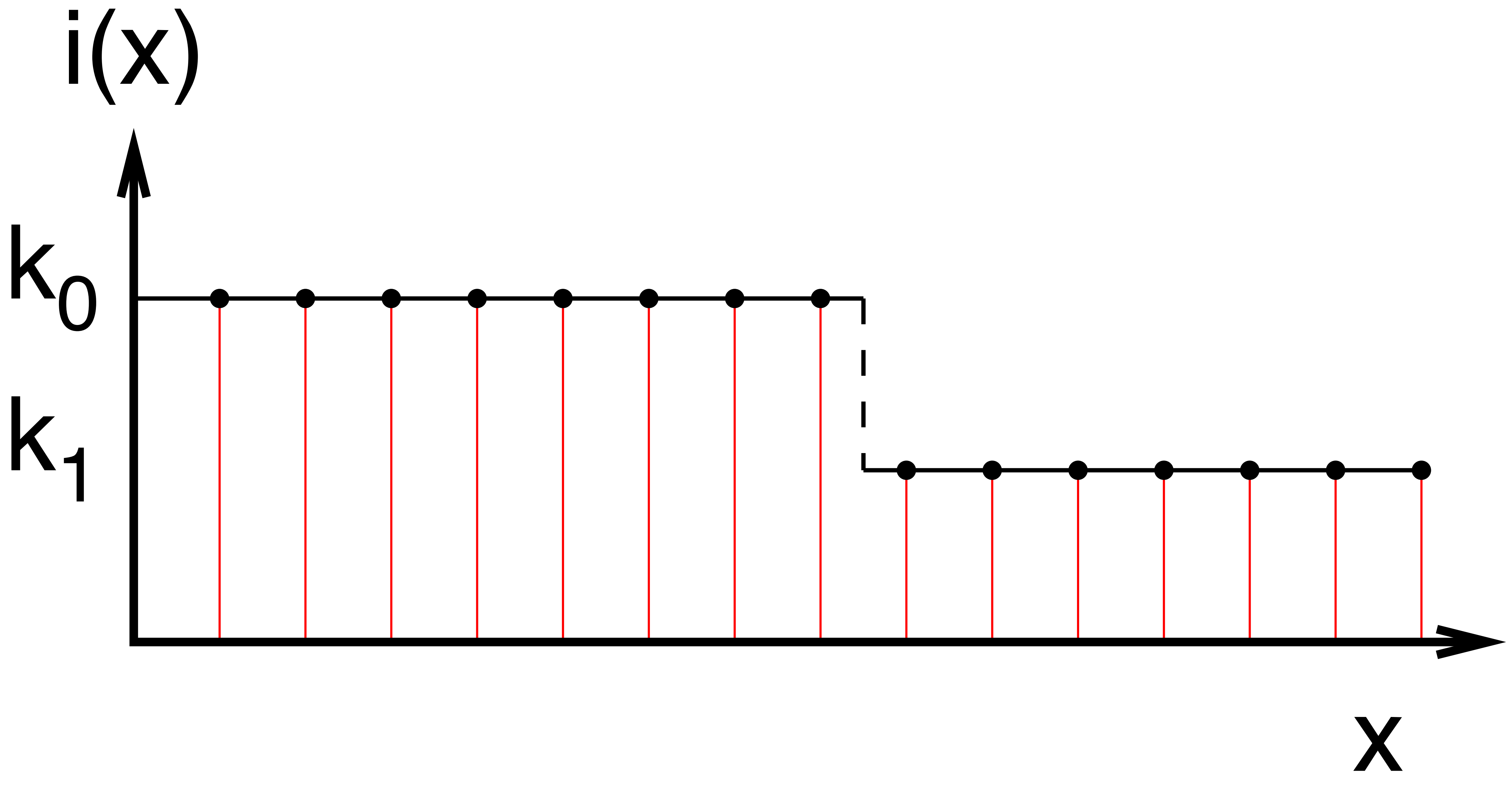
Sampling Theory (informal)

Case 1: Suppose $i(x, y)$ has a discontinuity not falling precisely at integer x, y



Sampling Theory (informal)

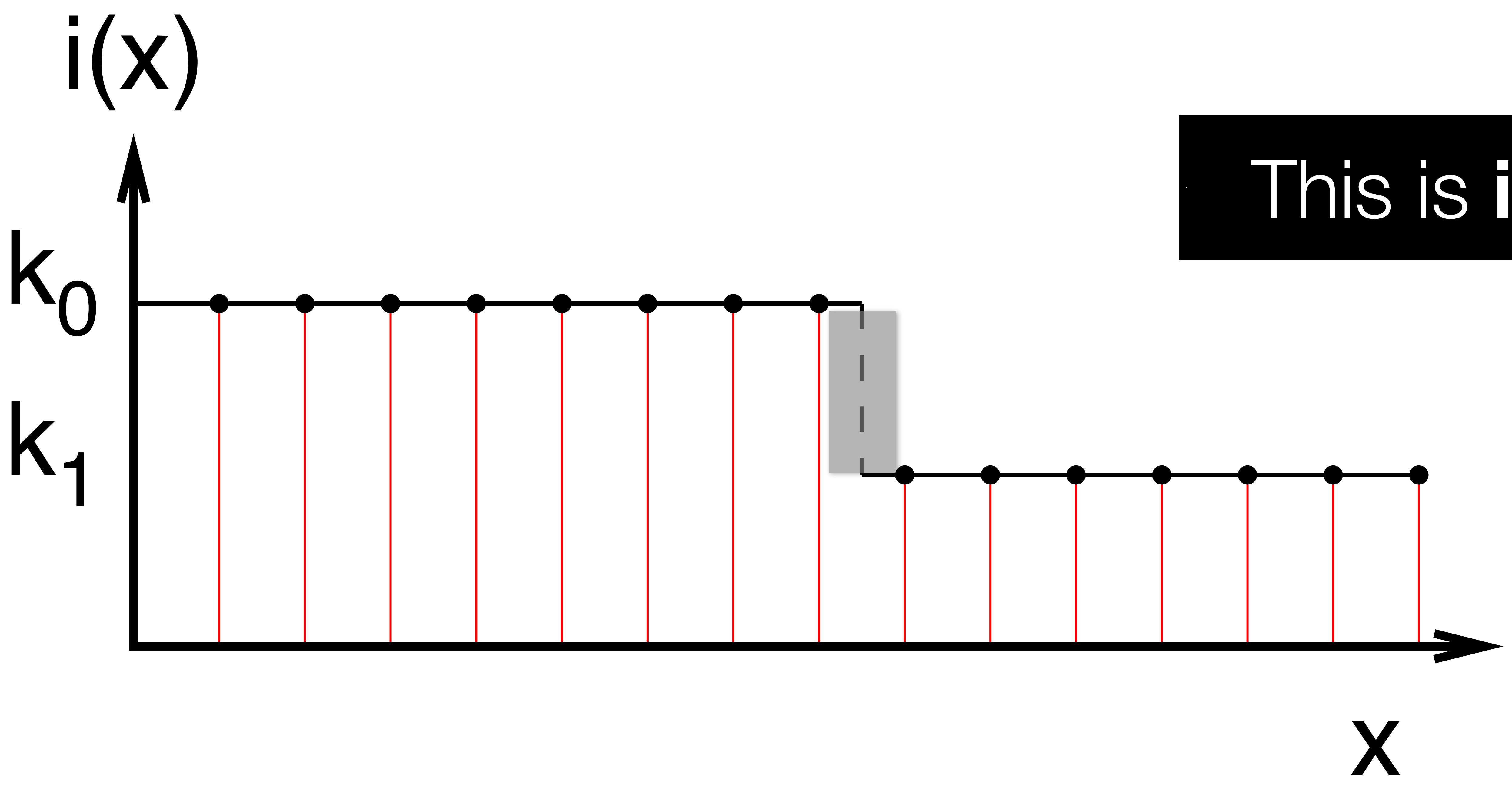
Case 1: Suppose $i(x, y)$ has a discontinuity not falling precisely at integer x, y



We cannot reconstruct $i(x, y)$ exactly because we can never know exactly where the discontinuity lies

Sampling Theory (informal)

Case 1: Suppose $i(x, y)$ has a discontinuity not falling precisely at integer x, y



This is **impossible**!

We cannot reconstruct $i(x, y)$ exactly because we can never know exactly where the discontinuity lies

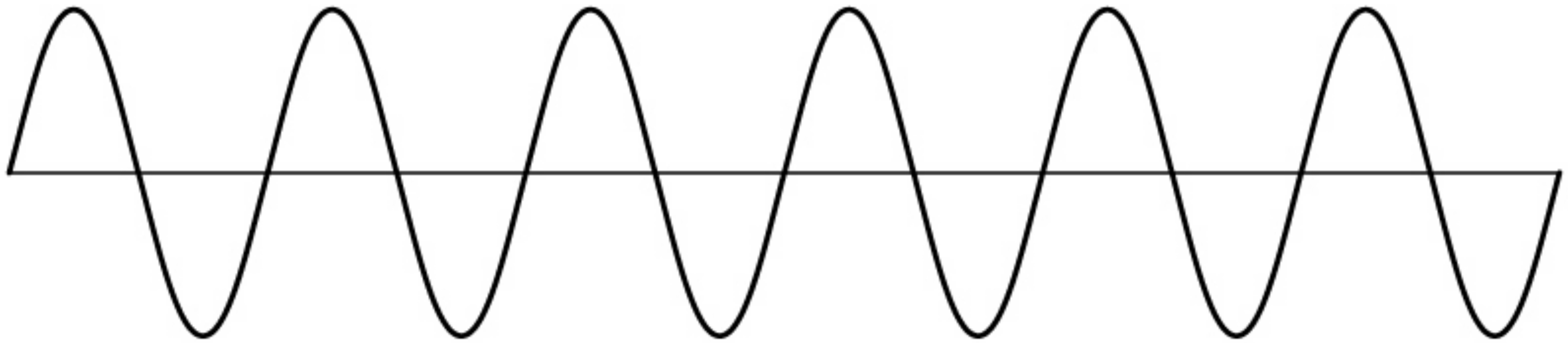
Sampling Theory (informal)

Question: How do we close the gap between “**easy**” and “**impossible**?”

Next, we build intuition based on informal argument

Example: A Simple Sine Wave

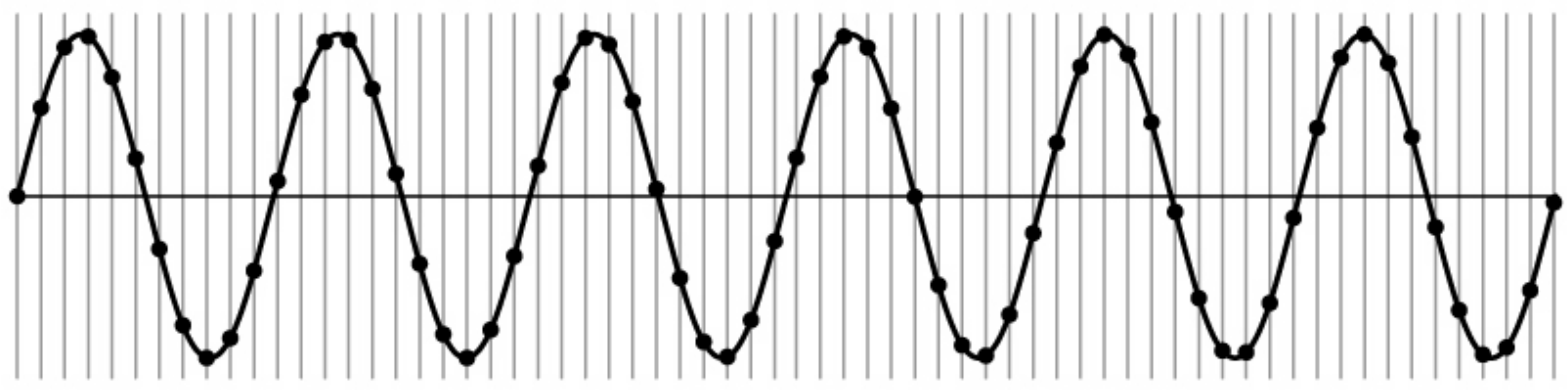
How do we discretize the signal?



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: A Simple Sine Wave

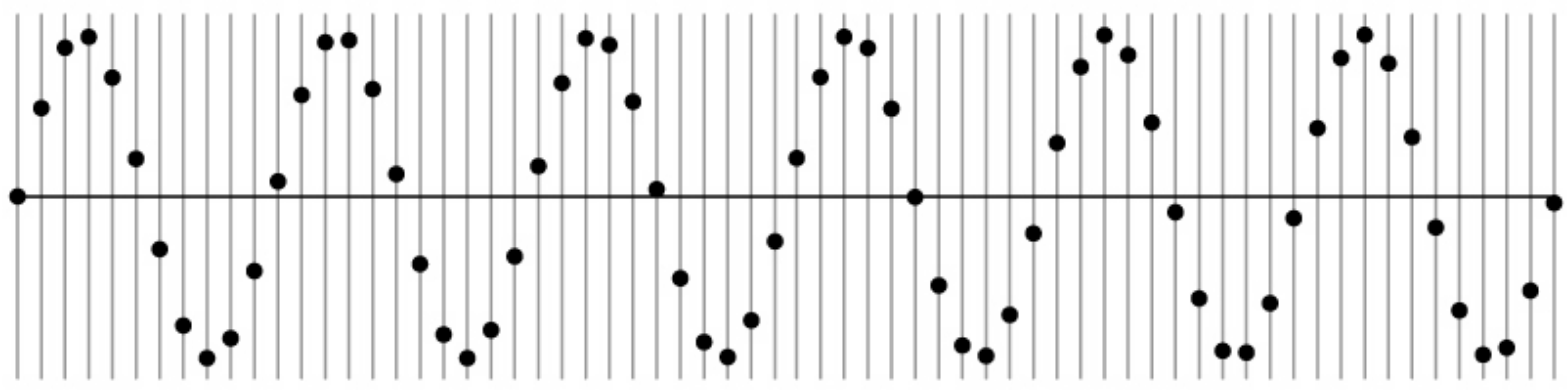
How do we discretize the signal?



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: A Simple Sine Wave

How do we discretize the signal?



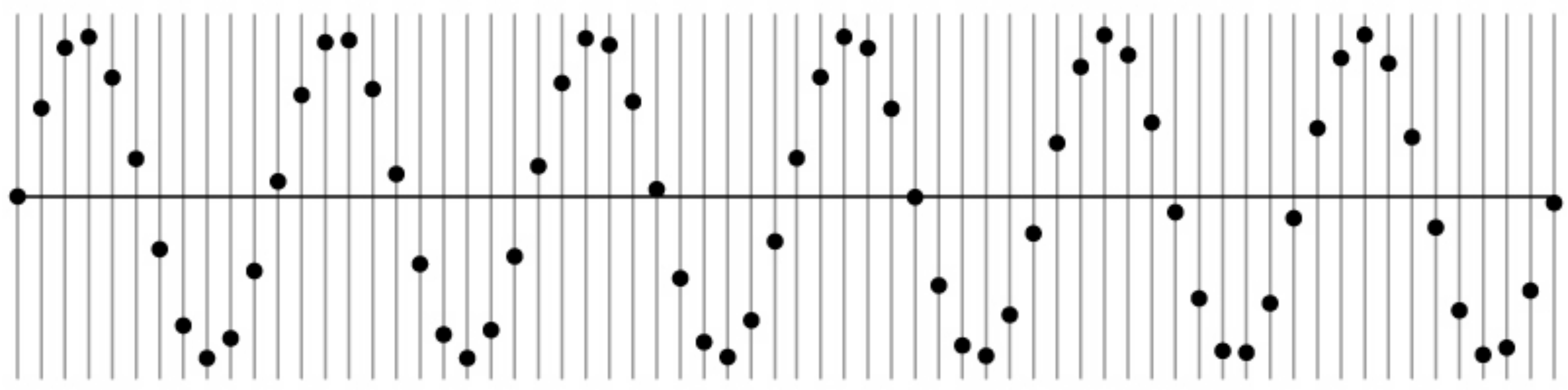
How many samples should I take?

Can I take as many samples as I want?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: A Simple Sine Wave

How do we discretize the signal?



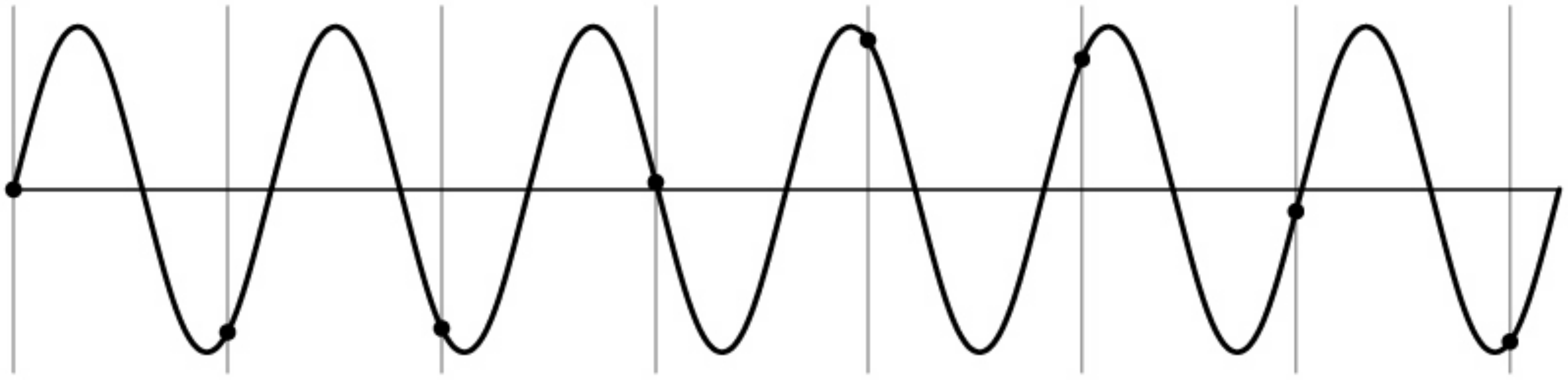
How many samples should I take?

Can I take as few samples as I want?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: A Simple Sine Wave

How do we discretize the signal?

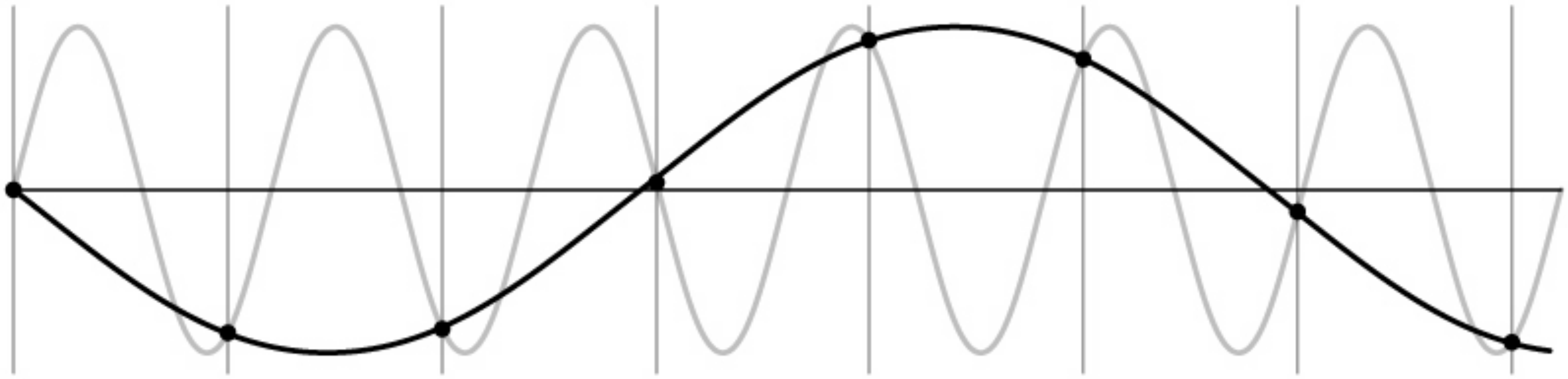


Signal can be confused with one at lower frequency

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: A Simple Sine Wave

How do we discretize the signal?

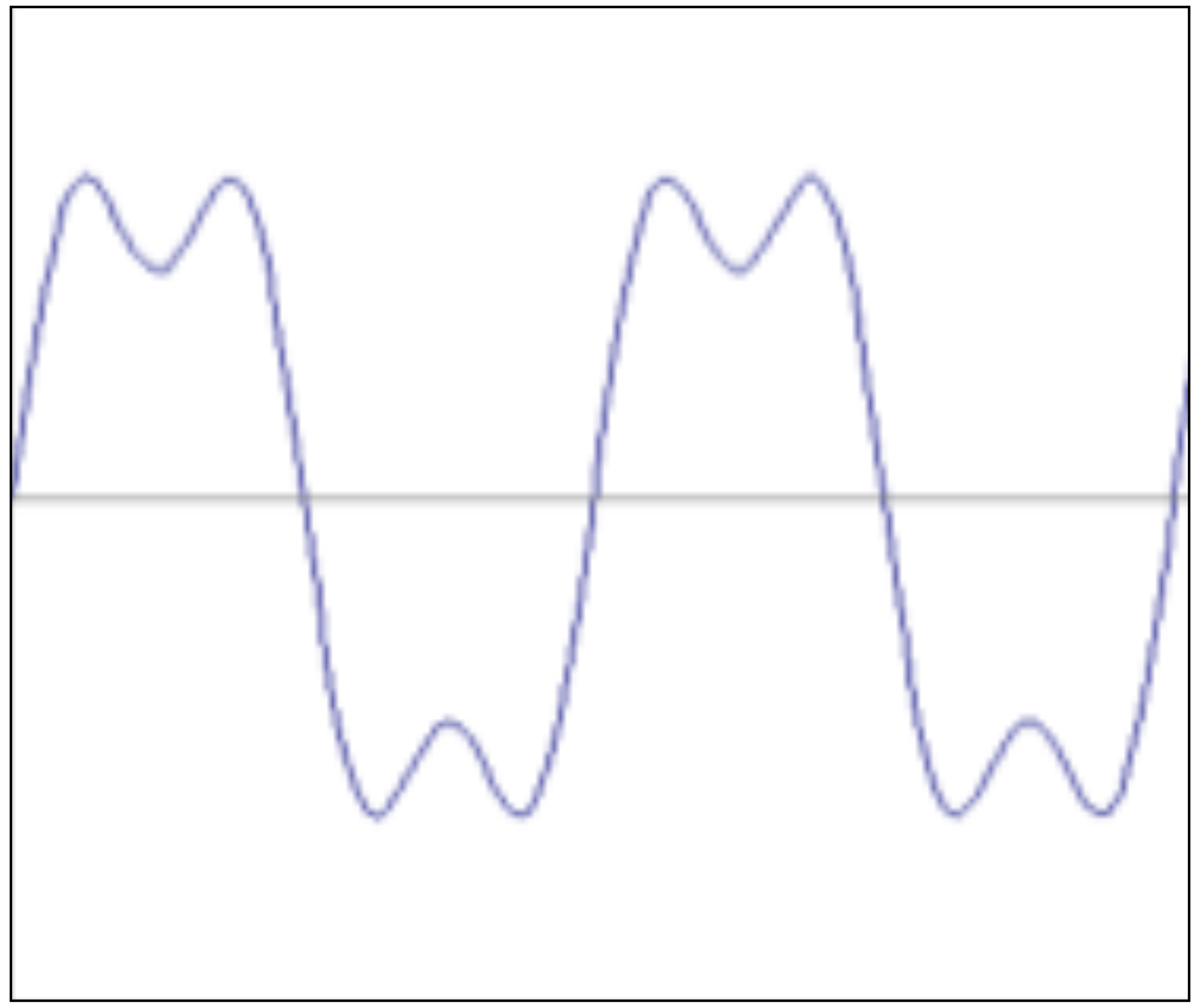


Signal can be confused with one at lower frequency
— This is called “Aliasing”

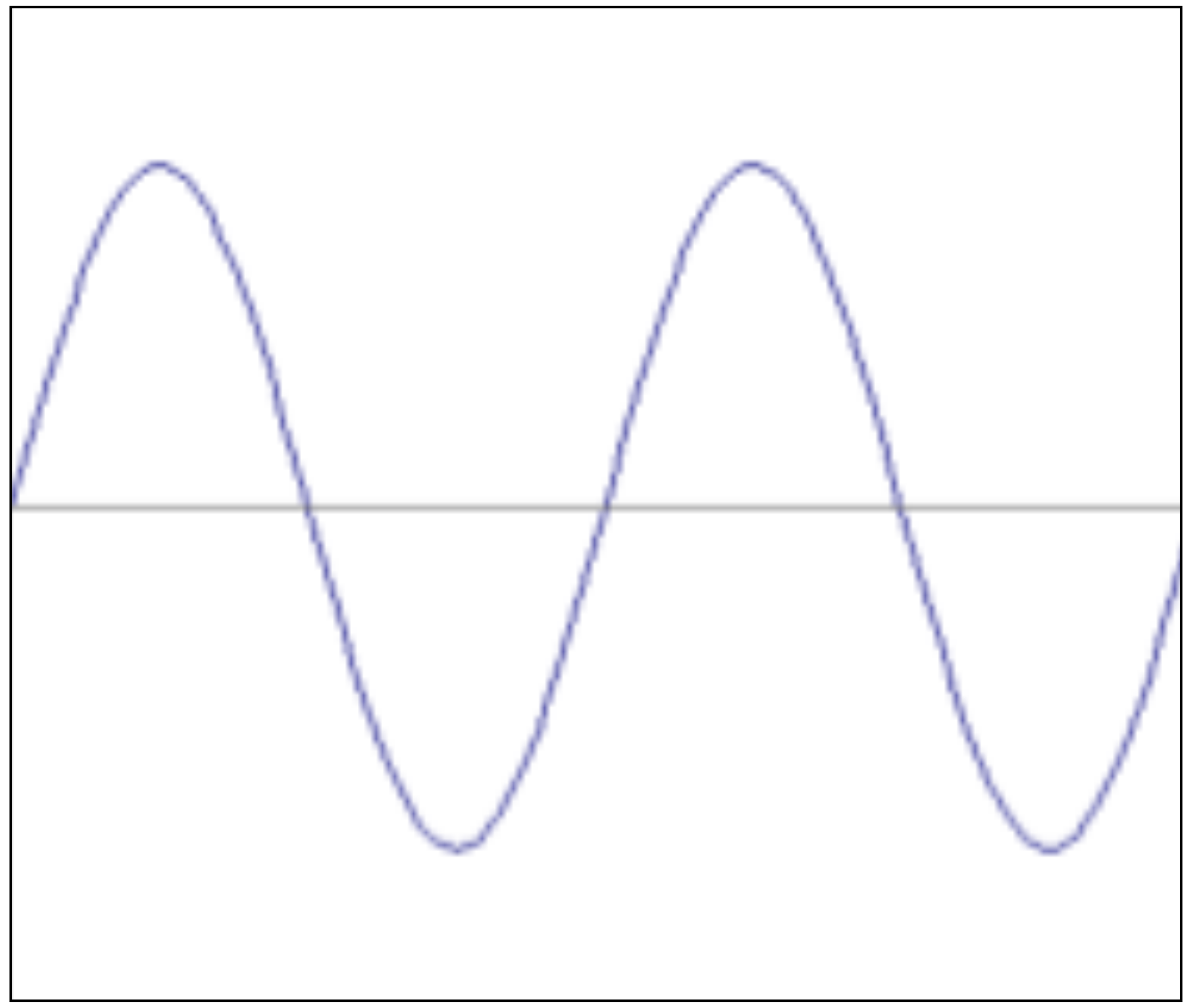
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Recall: **Fourier** Representation

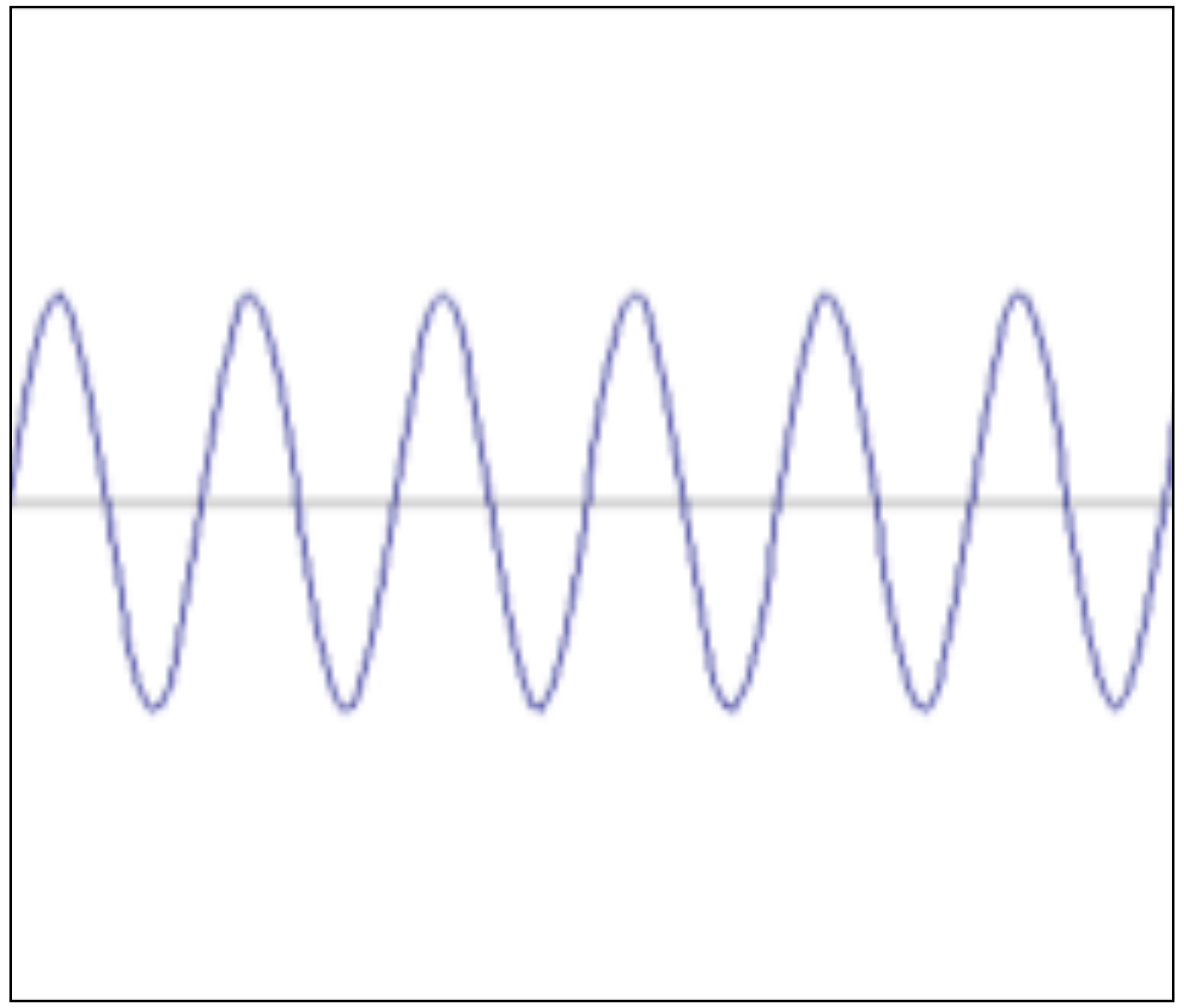
Any signal can be written as a sum of sinusoidal functions



=



+



$$f(x) = \sin(2\pi x) + \frac{1}{3}\sin(2\pi 3x)$$

$$\sin(2\pi x)$$

$$\frac{1}{3}\sin(2\pi 3x)$$

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Nyquist Sampling Theorem

To avoid aliasing a signal must be sampled at twice the maximum frequency:

$$f_s > 2 \times f_{max}$$

where f_s is the sampling frequency, and f_{max} is the maximum frequency present in the signal

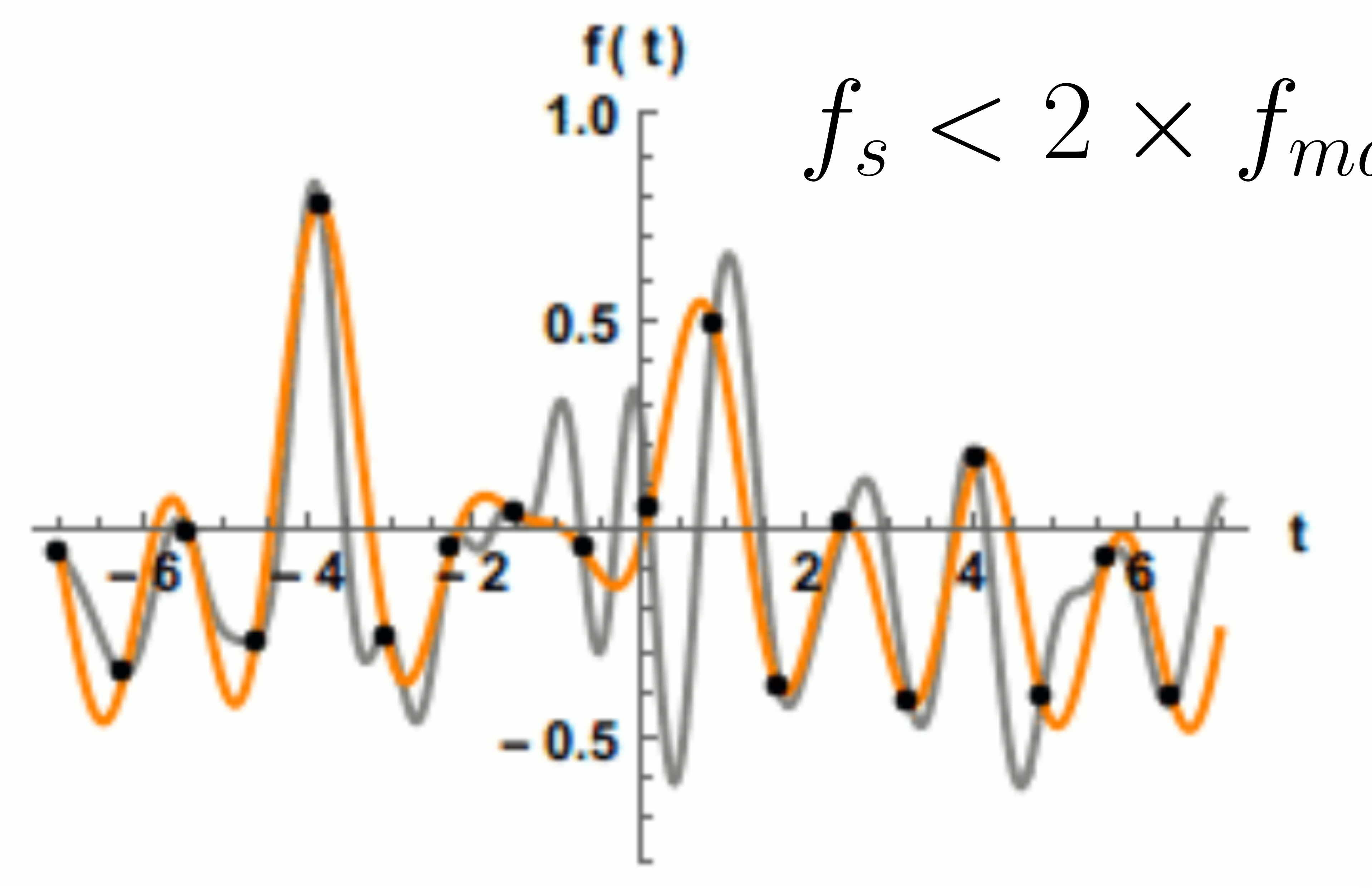
Futhermore, Nyquist's theorem states that a signal is **exactly recoverable** from its **samples** if sampled at the **Nyquist rate** (or higher)

Note: that a signal must be **bandlimited** for this to apply (i.e., it has a maximum frequency)

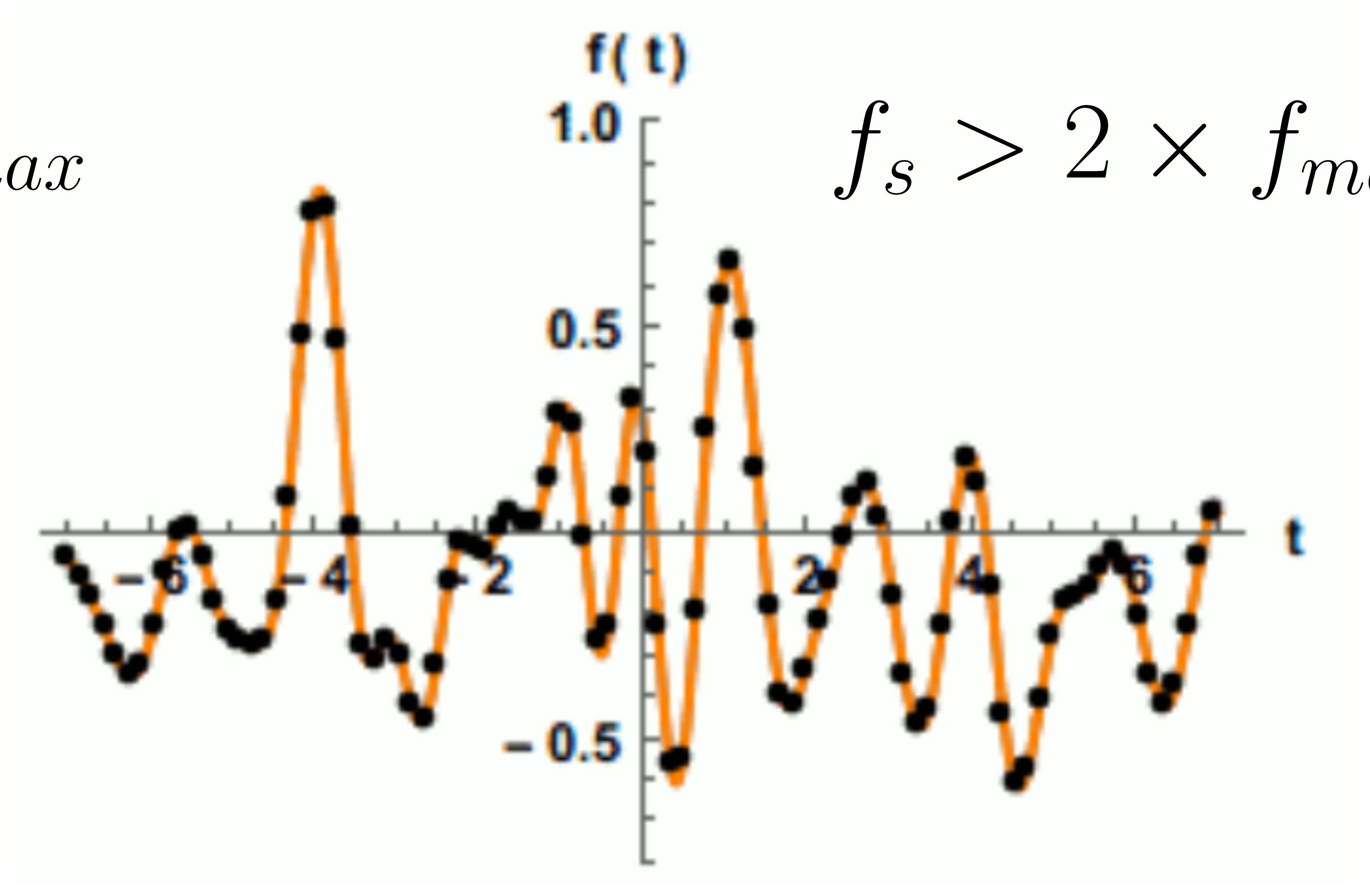
Reconstruction with **Bandlimited** Signal

It can be shown that a bandlimited and correctly sampled signal can be reconstructed exactly via interpolation with a **sinc** function (sin(x)/x)

(This is the Fourier Transform pair of a box filter, which in frequency domain is a pure low-pass filter)



$f_s < 2 \times f_{max}$



$f_s > 2 \times f_{max}$

https://en.wikipedia.org/wiki/Whittaker%E2%80%93Shannon_interpolation_formula

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

$\downarrow 2$ $\downarrow 4$ $\downarrow 8$

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

$\downarrow 2$ $\downarrow 4$ $\downarrow 8$

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

$\downarrow 2$ $\downarrow 4$ $\downarrow 8$

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- Aliasing causes undesirable artifacts in audio reproduction
- e.g., if we take an audio signal and simply drop every second sample, the highest frequencies will be aliased… we hear robotic sounding distortion

```
import scipy.io.wavfile as wavfile

rate, signal = wavfile.read("stevie.wav")

data=signal[0:(rate*10),:] # 10 seconds of audio

data_2=data[0:-1:2,:] # select every 2nd sample
data_4=data[0:-1:4,:] # select every 4th sample
data_8=data[0:-1:8,:] # select every 8th sample

wavfile.write('test2.wav', int(rate/2), data_2)
wavfile.write('test4.wav', int(rate/4), data_4)
wavfile.write('test8.wav', int(rate/8), data_8)
```

Original

↓2 ↓4 ↓8

Audio **Aliasing**

- We can reduce the aliasing artifacts by **pre-filtering** with a low pass filter
- e.g., if we apply smoothing with a Gaussian filter standard deviation 2.0 for each octave (factor 2) of downsampling we get a better result:

↓8

↓8 with pre-filtering

- Note we have still lost some of the high frequency content, but the crunchy sounding distortion due to aliasing has now gone

Audio **Aliasing**

- We can reduce the aliasing artifacts by **pre-filtering** with a low pass filter
- e.g., if we apply smoothing with a Gaussian filter standard deviation 2.0 for each octave (factor 2) of downsampling we get a better result:

↓8 ↓8 with pre-filtering

- Note we have still lost some of the high frequency content, but the crunchy sounding distortion due to aliasing has now gone

Audio **Aliasing**

- We can reduce the aliasing artifacts by **pre-filtering** with a low pass filter
- e.g., if we apply smoothing with a Gaussian filter standard deviation 2.0 for each octave (factor 2) of downsampling we get a better result:

$\downarrow 8$ $\downarrow 8$ with pre-filtering

- Note we have still lost some of the high frequency content, but the crunchy sounding distortion due to aliasing has now gone

Sampling Theory (informal)

Exact reconstruction requires constraint on the rate at which i(x,y) can change between samples

- "rate of change" means derivative
- the formal concept is **bandlimited signal**
- "bandlimit" and "constraint on derivative" are linked

Think of music

- bandlimited if it has some maximum **temporal frequency**
- the upper limit of human hearing is about 20 kHz

Think of imaging systems. Resolving power is measured in

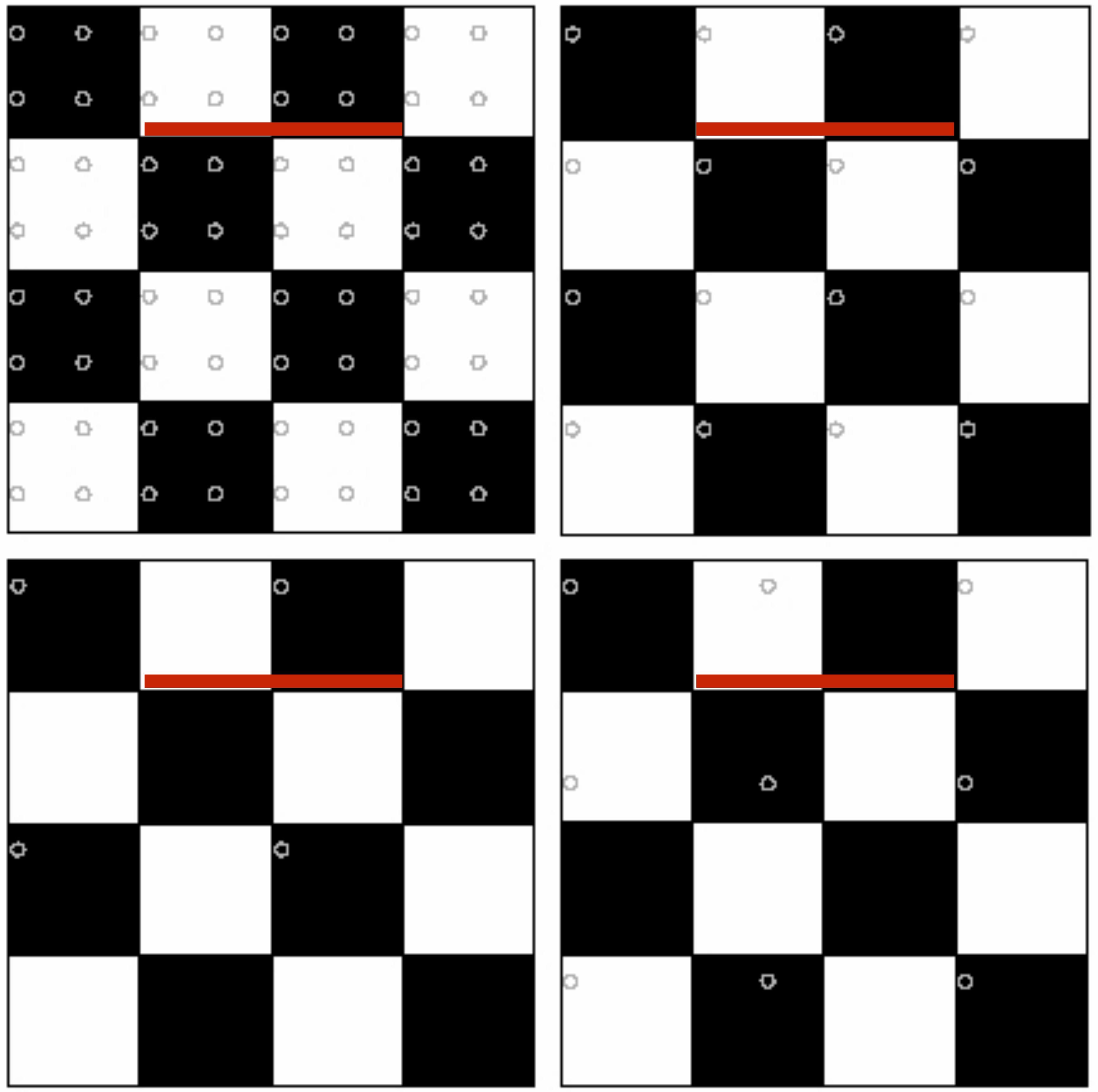
- "line pairs per mm" (for a bar test pattern)
- "cycles per mm" (for a sine wave test pattern)

An image is bandlimited if it has some maximum **spatial frequency**

Sampling

It is clear that *some* information may be lost when we work on a discrete pixel grid.

$$f_s > 2 \times f_{max}$$

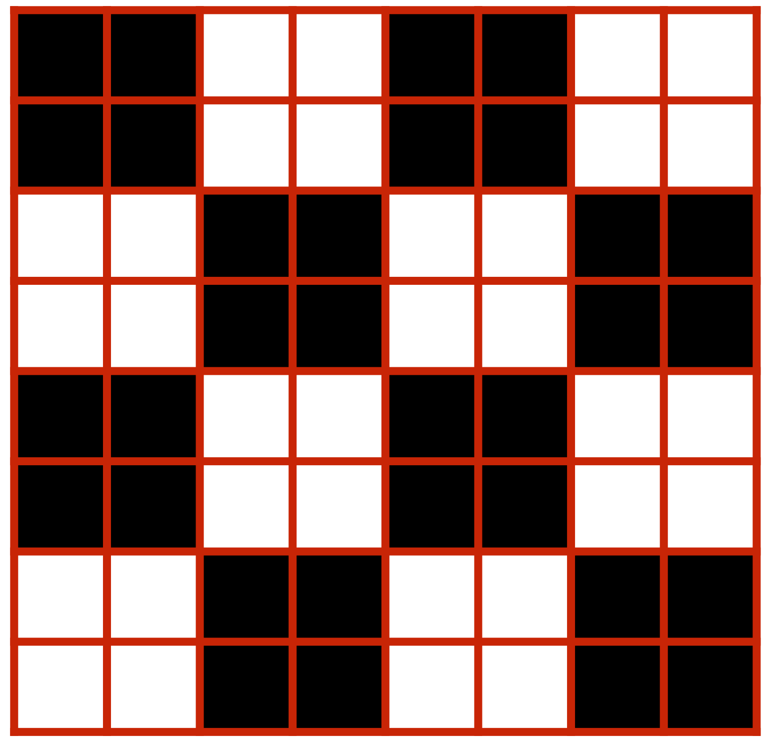


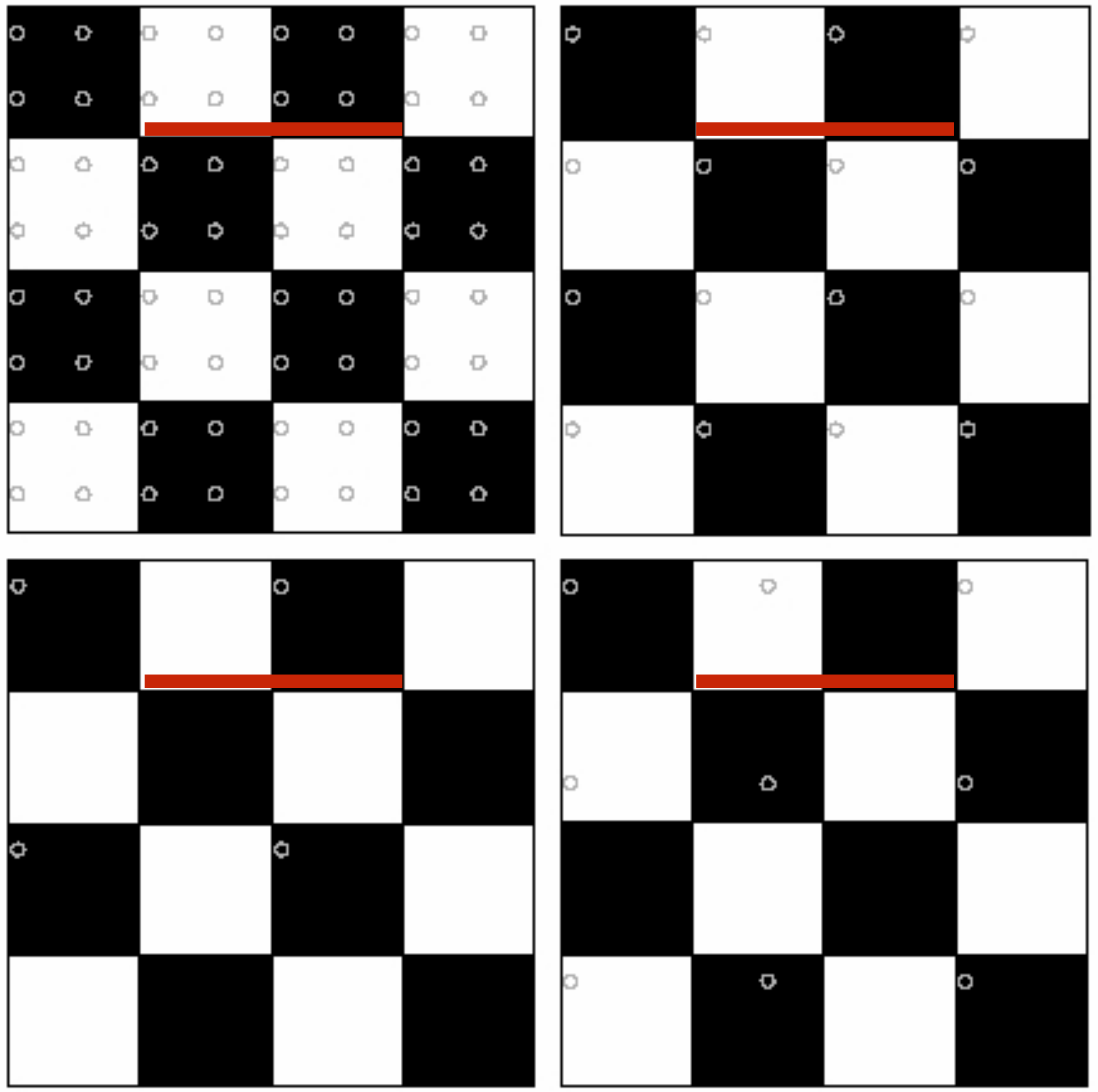
Forsyth & Ponce (2nd ed.) Figure 4.7

Sampling

It is clear that *some* information may be lost when we work on a discrete pixel grid.

$$f_s > 2 \times f_{max}$$



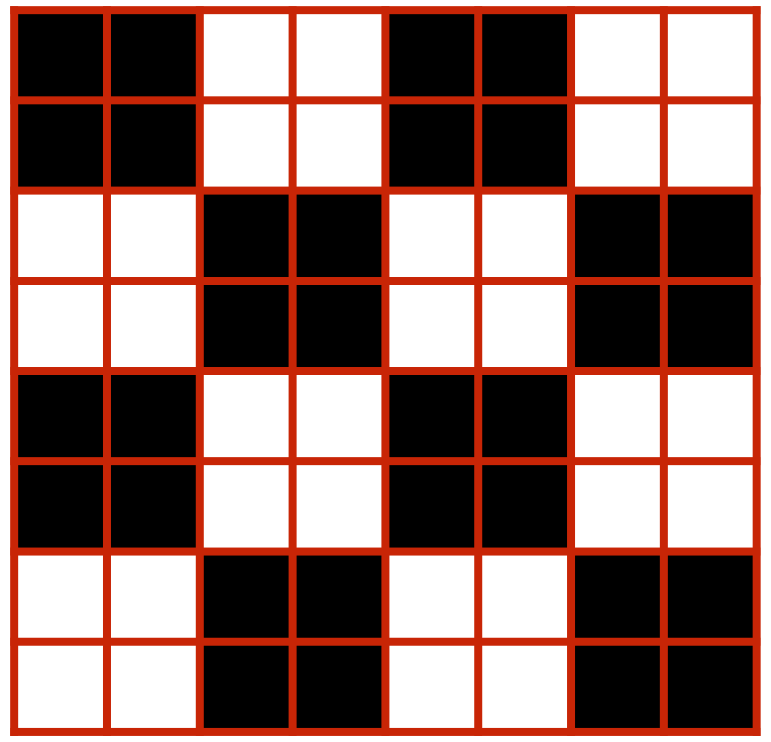


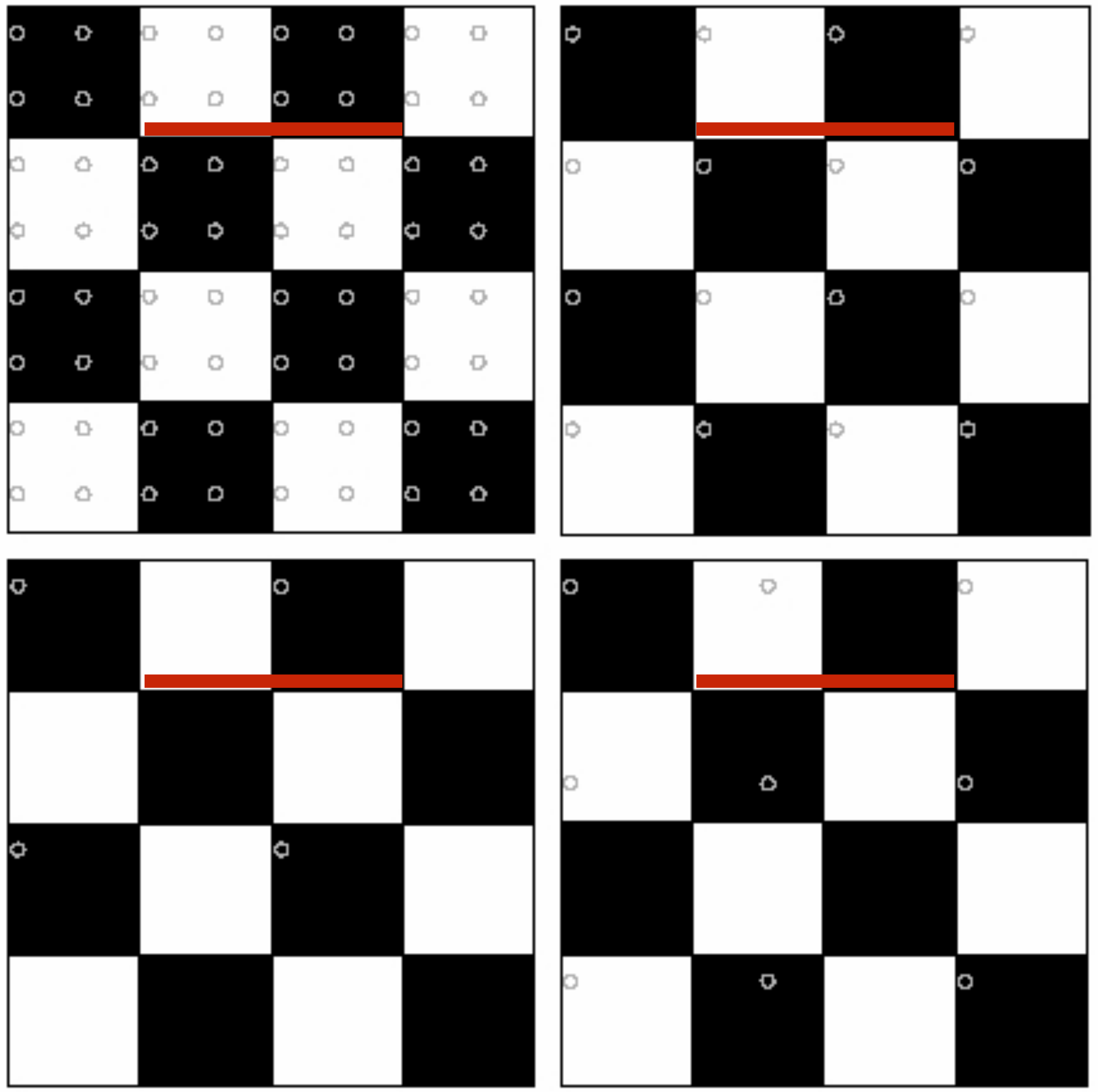
Forsyth & Ponce (2nd ed.) Figure 4.7

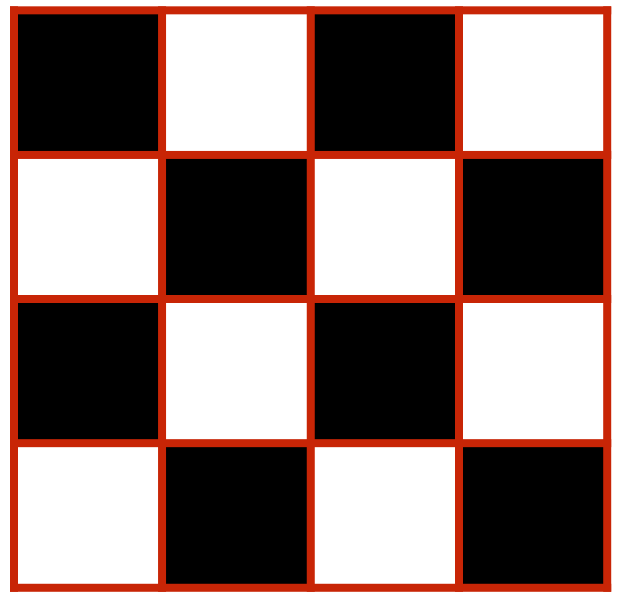
Sampling

It is clear that *some* information may be lost when we work on a discrete pixel grid.

$$f_s > 2 \times f_{max}$$



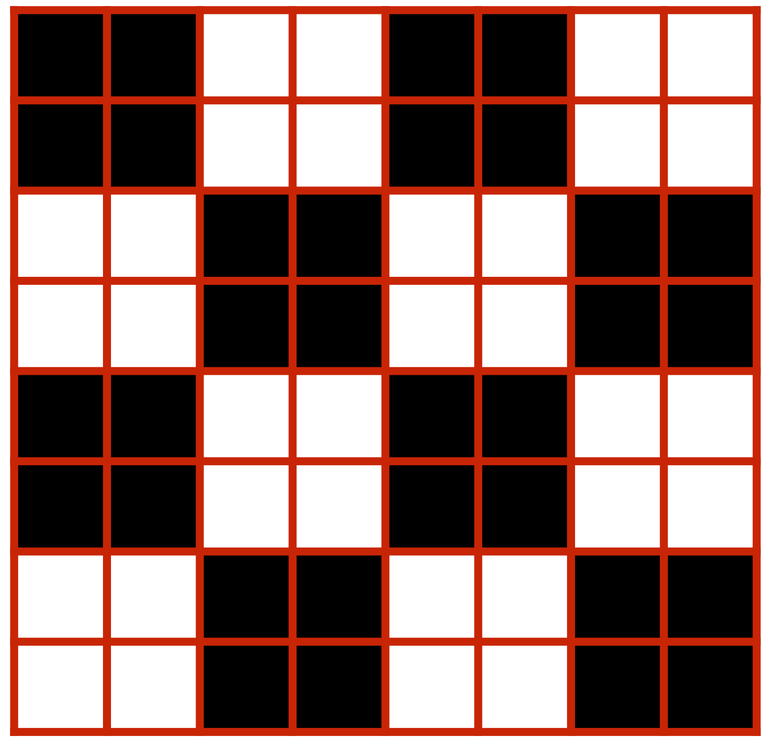




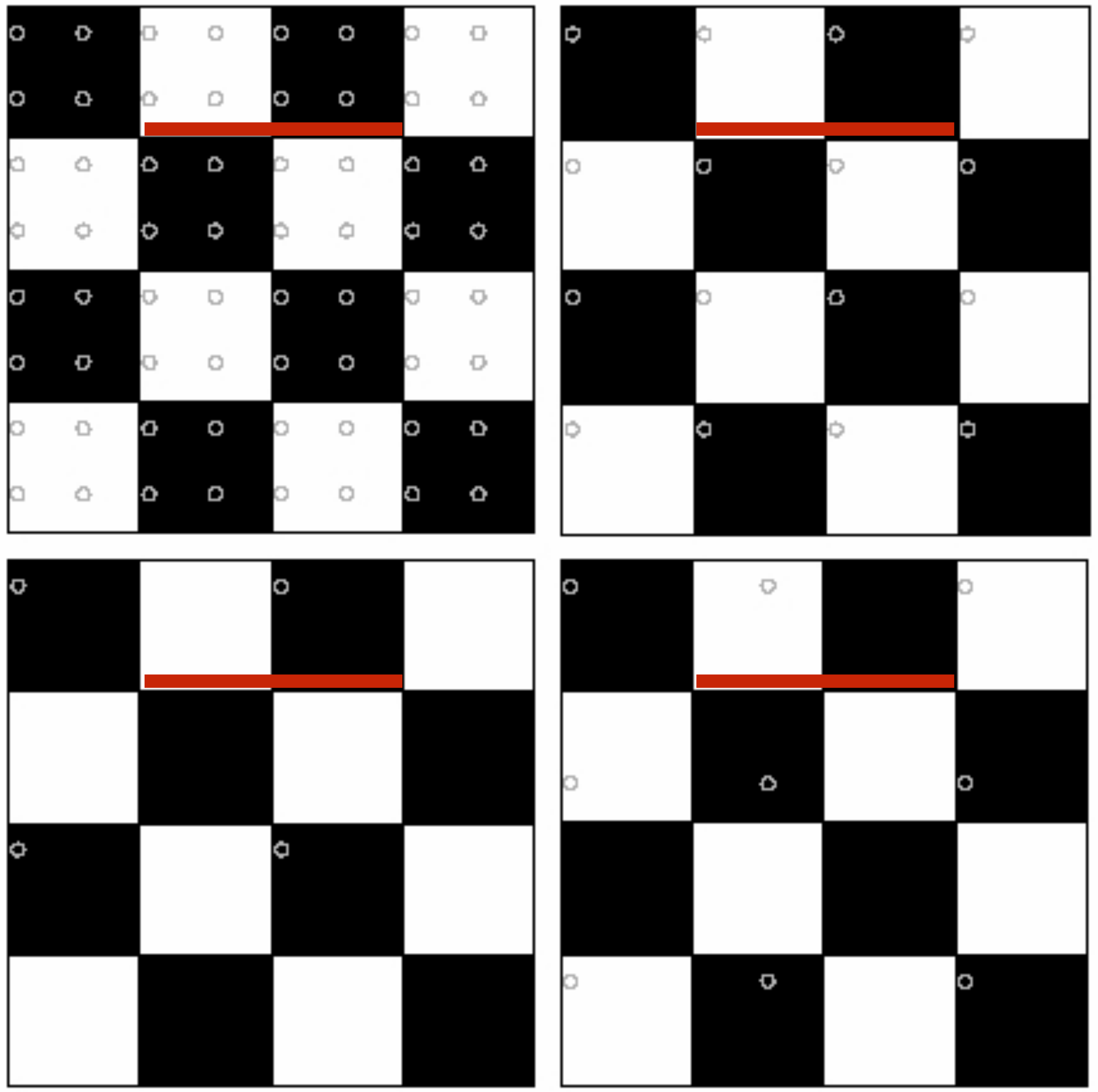
Forsyth & Ponce (2nd ed.) Figure 4.7

Sampling

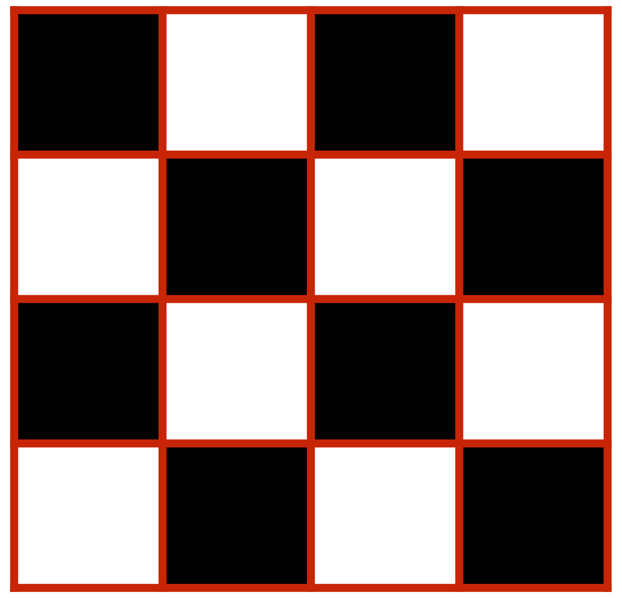
It is clear that *some* information may be lost when we work on a discrete pixel grid.

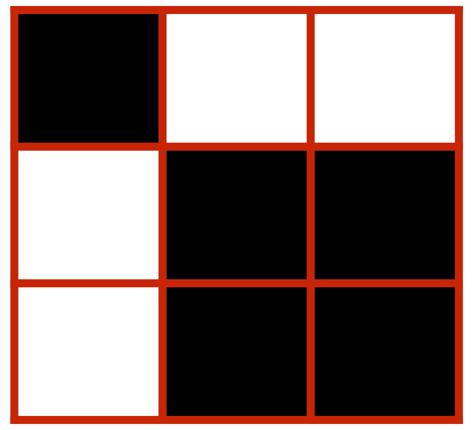


$$f_s > 2 \times f_{max}$$



Forsyth & Ponce (2nd ed.) Figure 4.7

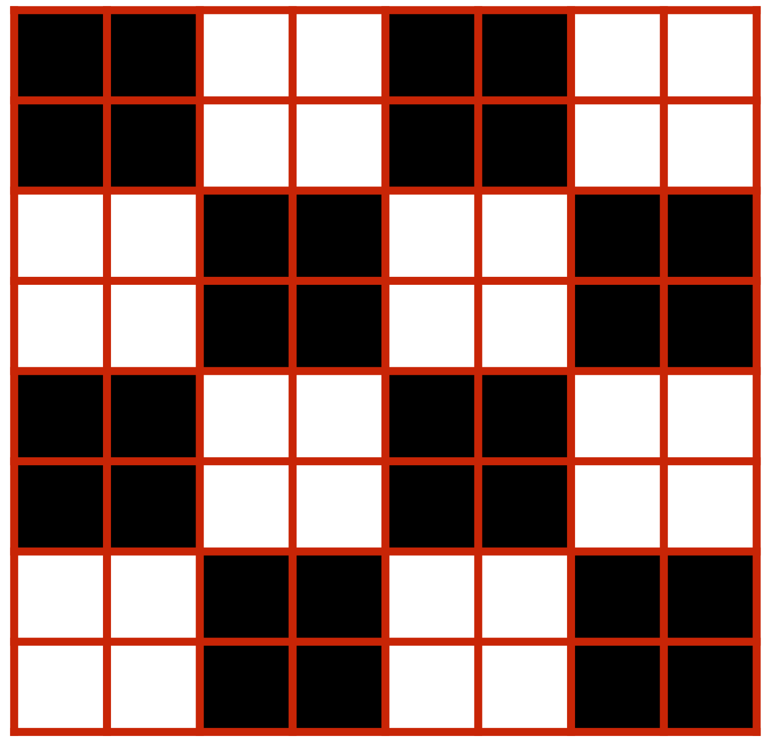


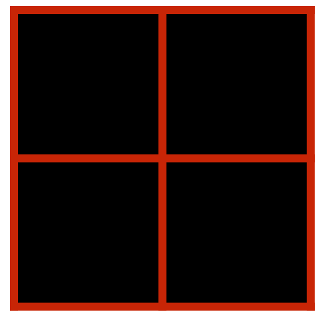


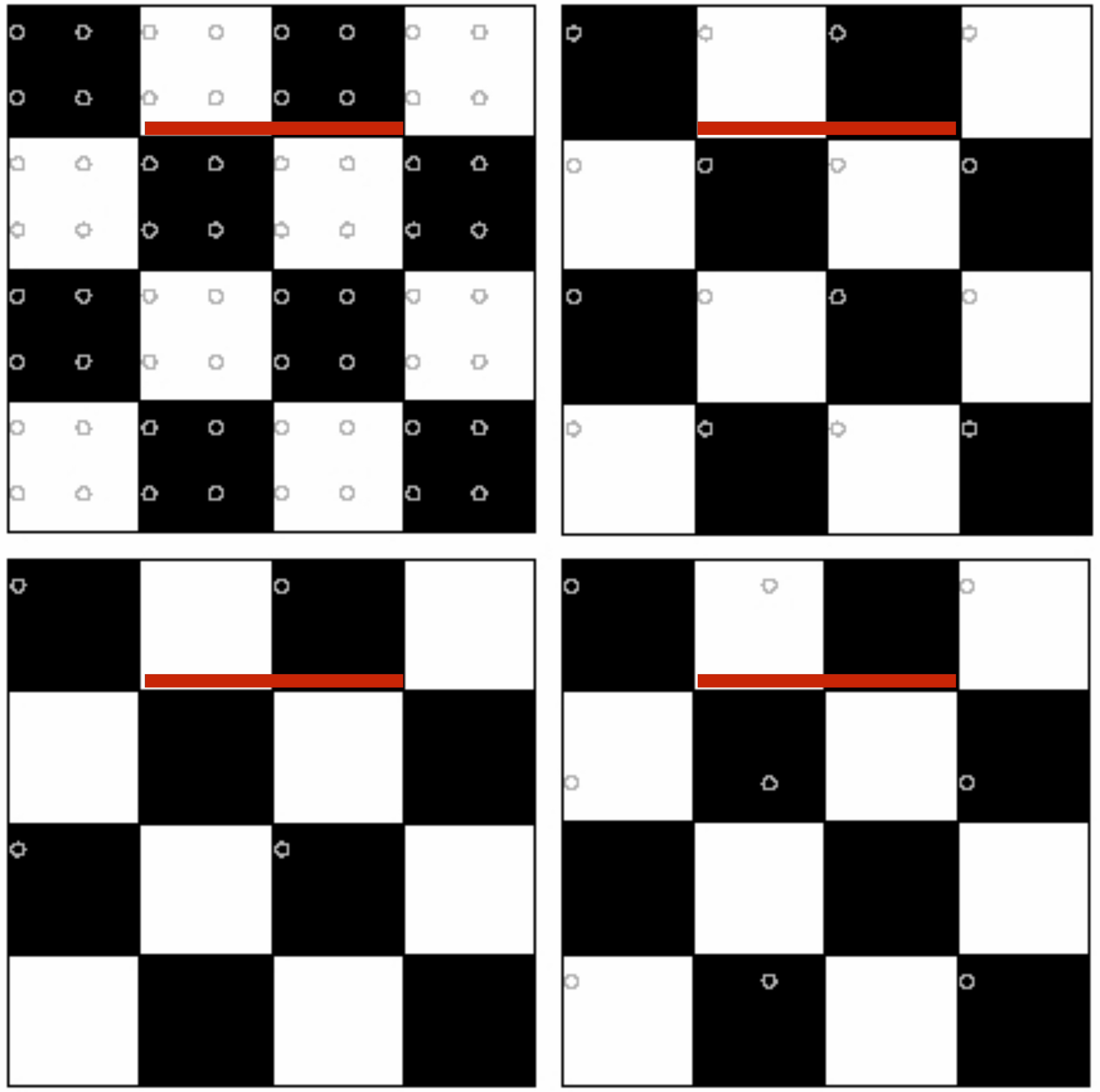
Sampling

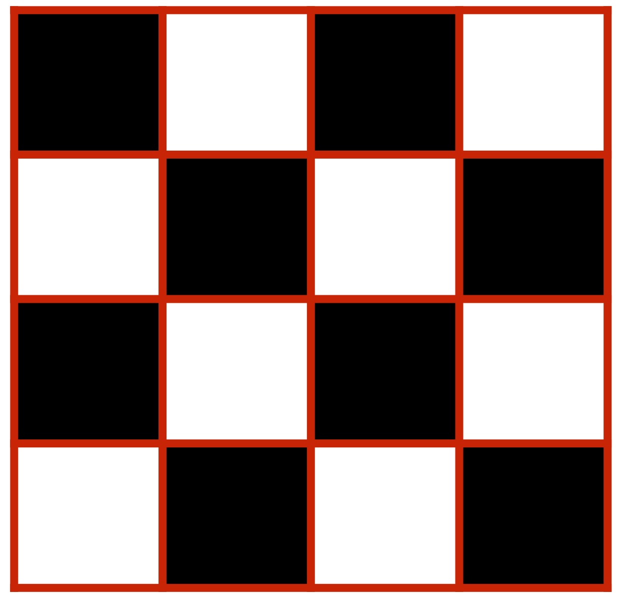
It is clear that *some* information may be lost when we work on a discrete pixel grid.

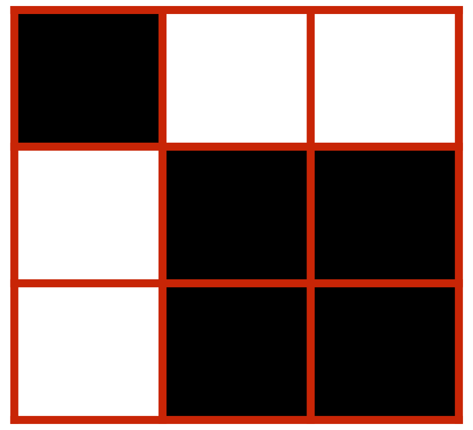
$$f_s > 2 \times f_{max}$$











Forsyth & Ponce (2nd ed.) Figure 4.7

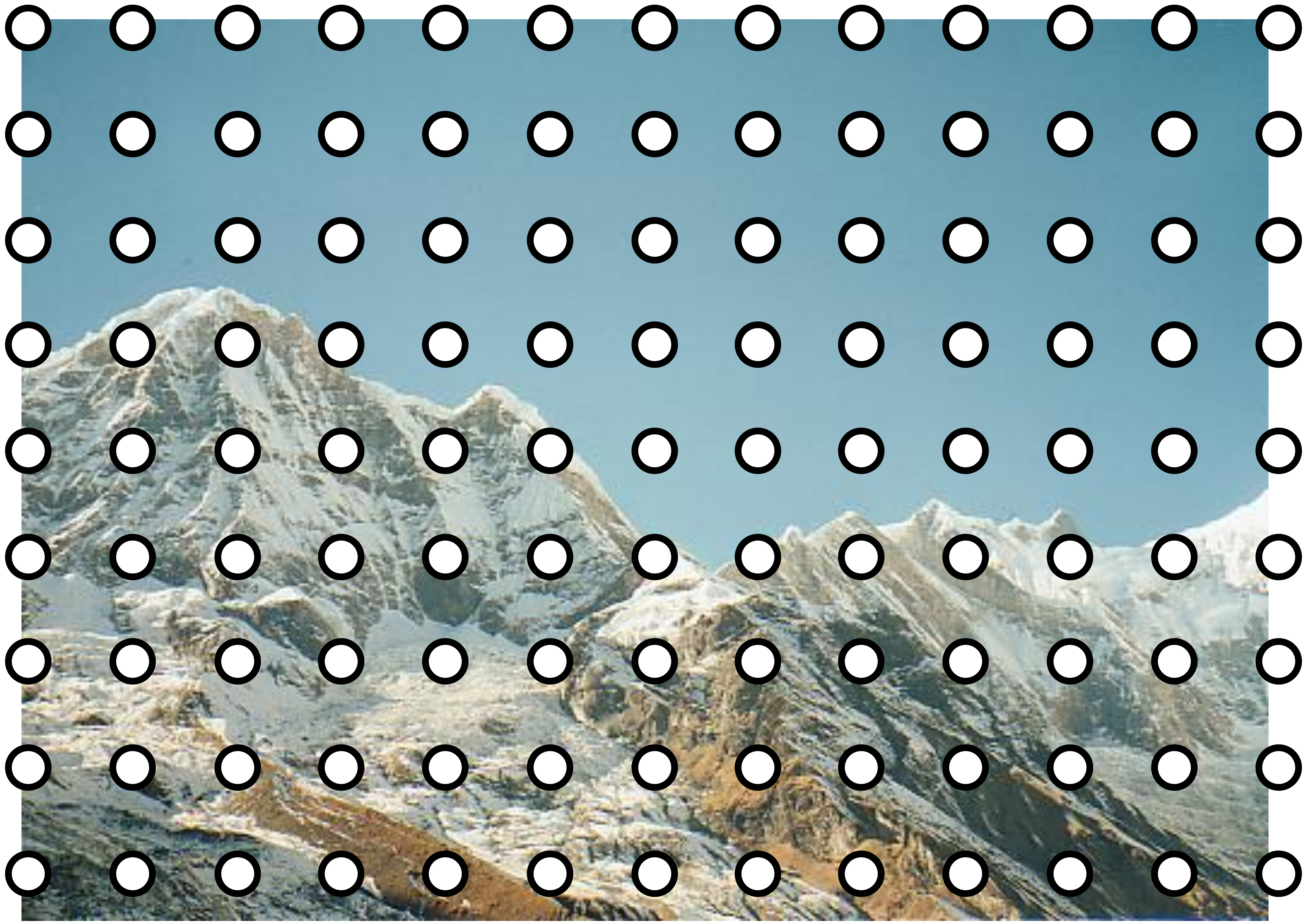
Resampling Images

Goal: Resample the image to get a lower resolution counterpart

Naive Method: Form new image by taking every n-th pixel of the original image

Resampling Images

Goal: Resample the image to get a lower resolution counterpart



Naive Method: Form new image by taking every n-th pixel of the original image

Resampling Images

With correct sigma value for a Gaussian, no information is lost

Improved Method: First blur the image (with low-pass) then take n-th pixel

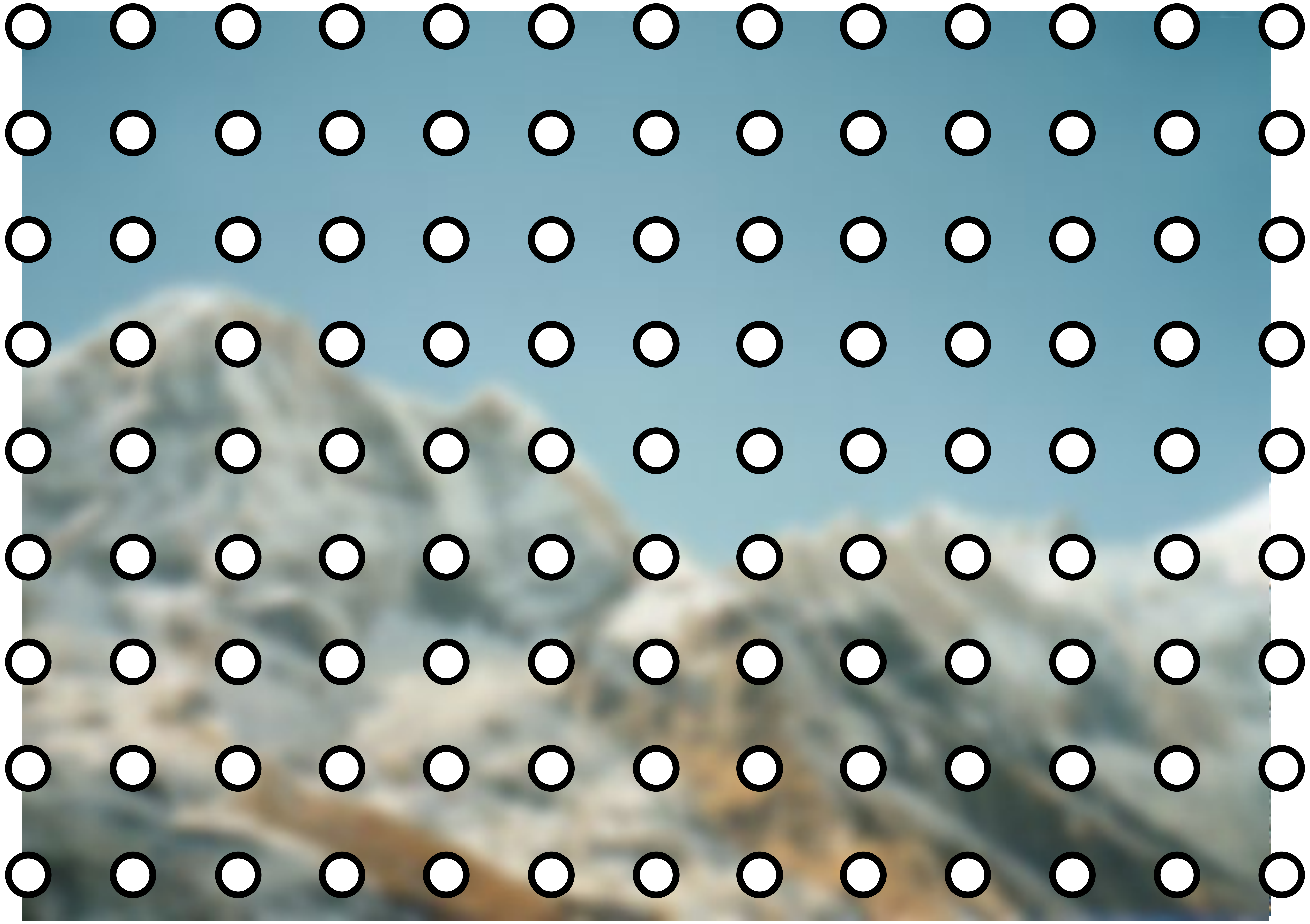
Resampling Images

With correct sigma value for a Gaussian, no information is lost

Improved Method: First blur the image (with low-pass) then take n-th pixel

Resampling Images

With correct sigma value for a Gaussian, no information is lost



Improved Method: First blur the image (with low-pass) then take n-th pixel

Aliasing Example

Sampling every 5th pixel with and without low-pass blur

No filtering

Gaussian Blur $\sigma = 3.0$

Aliasing Example

Sampling every 5th pixel with and without low-pass blur

No filtering

Gaussian Blur $\sigma = 3.0$

Aliasing Example

Sampling every 5th pixel with and without low-pass blur

No filtering

Gaussian Blur $\sigma = 3.0$

Aliasing Example

Sampling every 5th pixel with and without low-pass blur

No filtering

Gaussian Blur $\sigma = 3.0$

$$\sigma = 1/(2s)$$

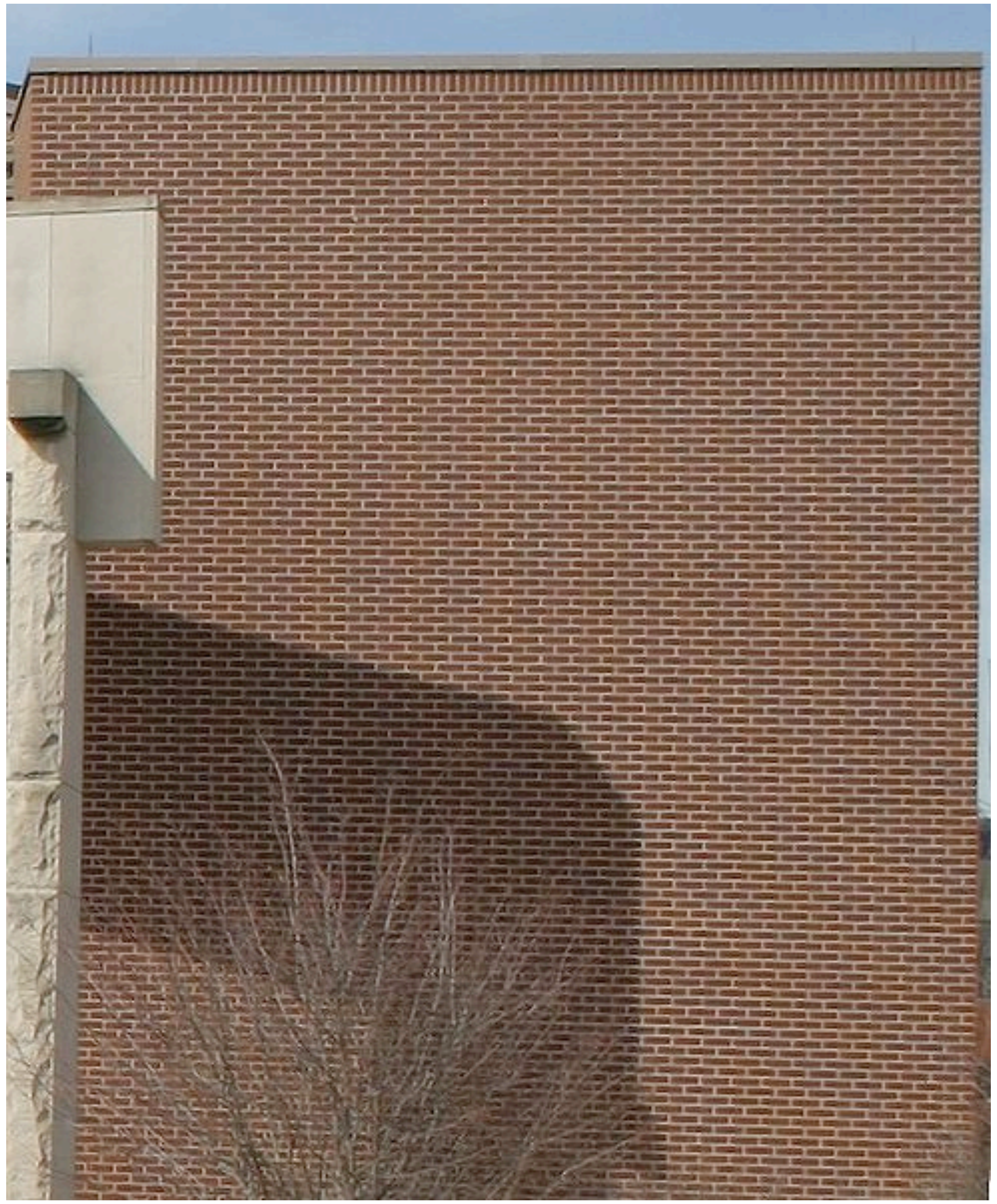
Resampling Images

every 10th pixel
(aliased)

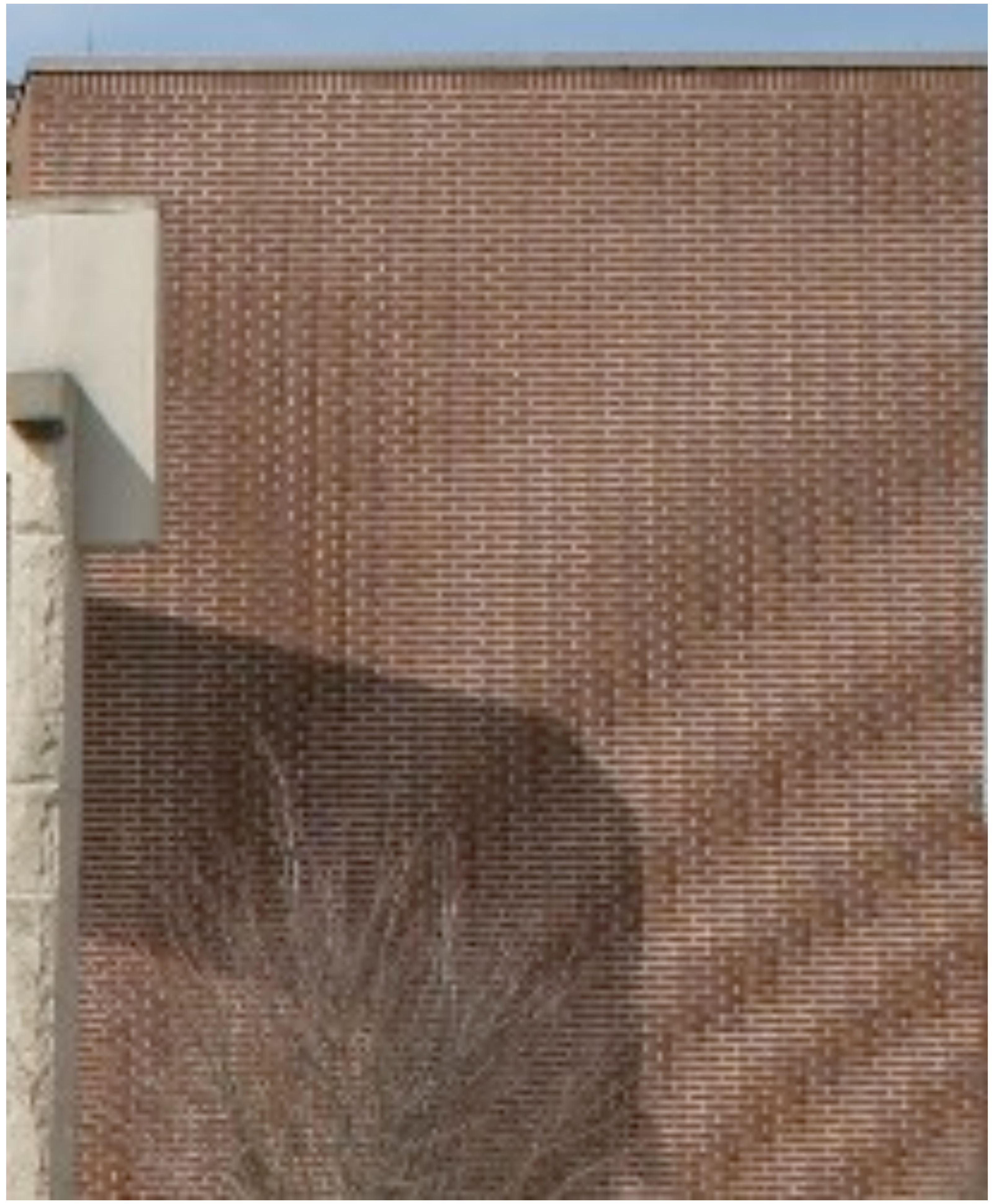
low pass filtered
(correct sampling)

- Note that selecting every 10th pixel ignores the intervening information, whereas the low-pass filter (blur) smoothly combines it
- If we shifted the original image 1 pixel to the right, the aliased image would look completely different, but the low pass filtered image would look almost the same

Image Sampling and Aliasing



$f_s > 2 \times f_{max}$



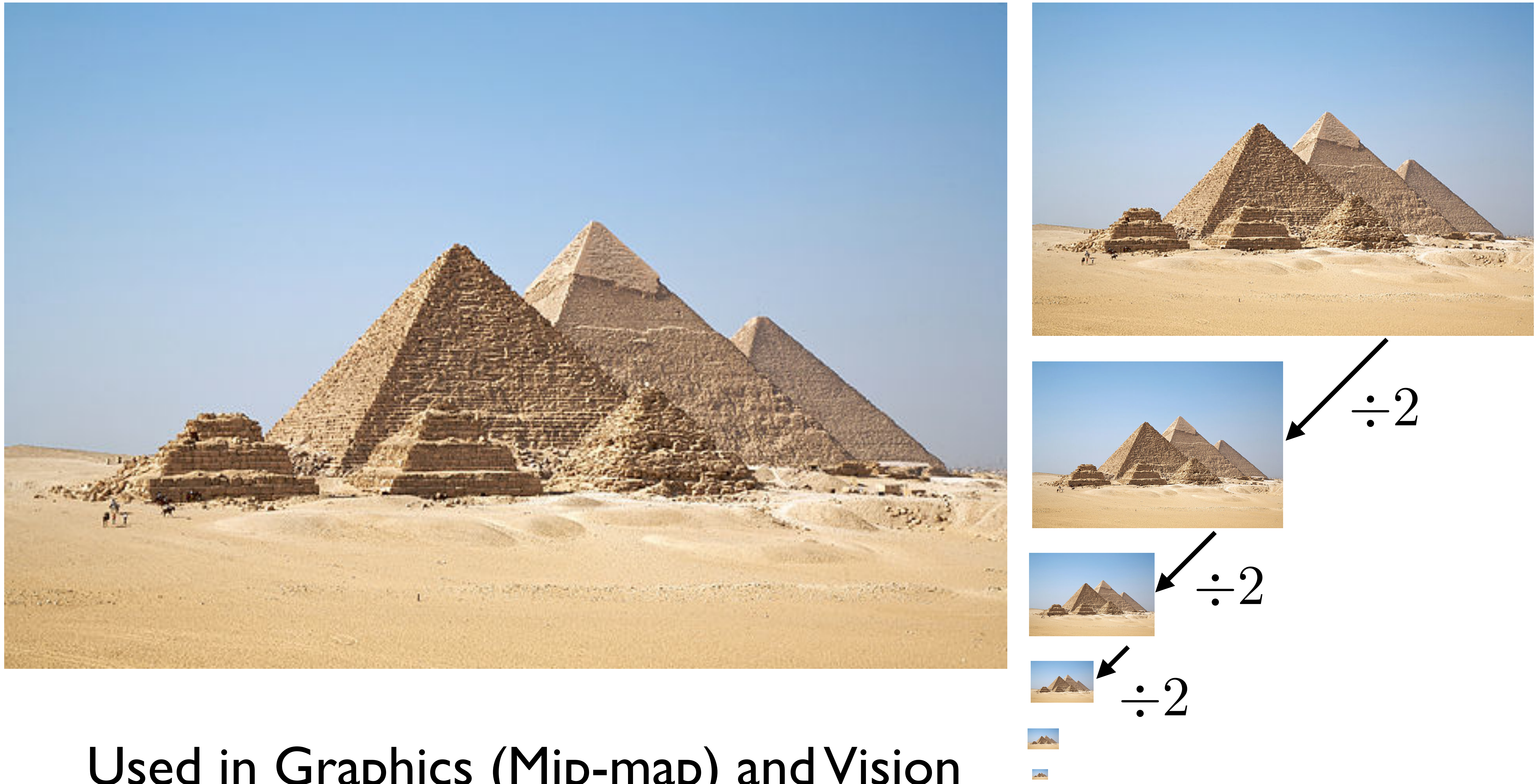
$f_s < 2 \times f_{max}$

Aliasing in Photographs

This is also known as “moire”

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Image Pyramids



Used in Graphics (Mip-map) and Vision
(for **multi-scale** processing)

$G1$

Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x,y) = I(x,y) * g_\sigma(x,y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

[Assignment 2]

$G1$  blur → 

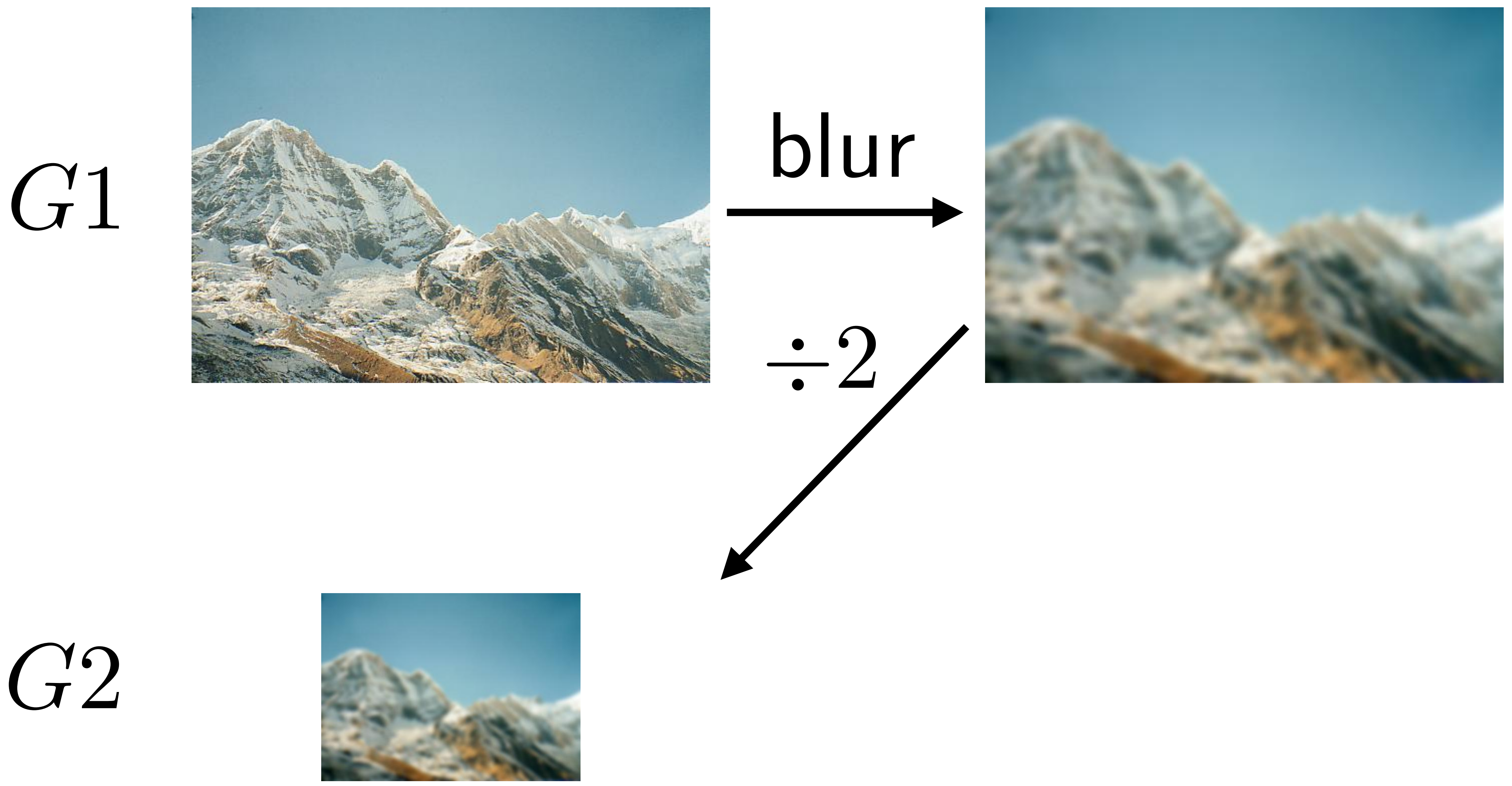
Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

[Assignment 2]



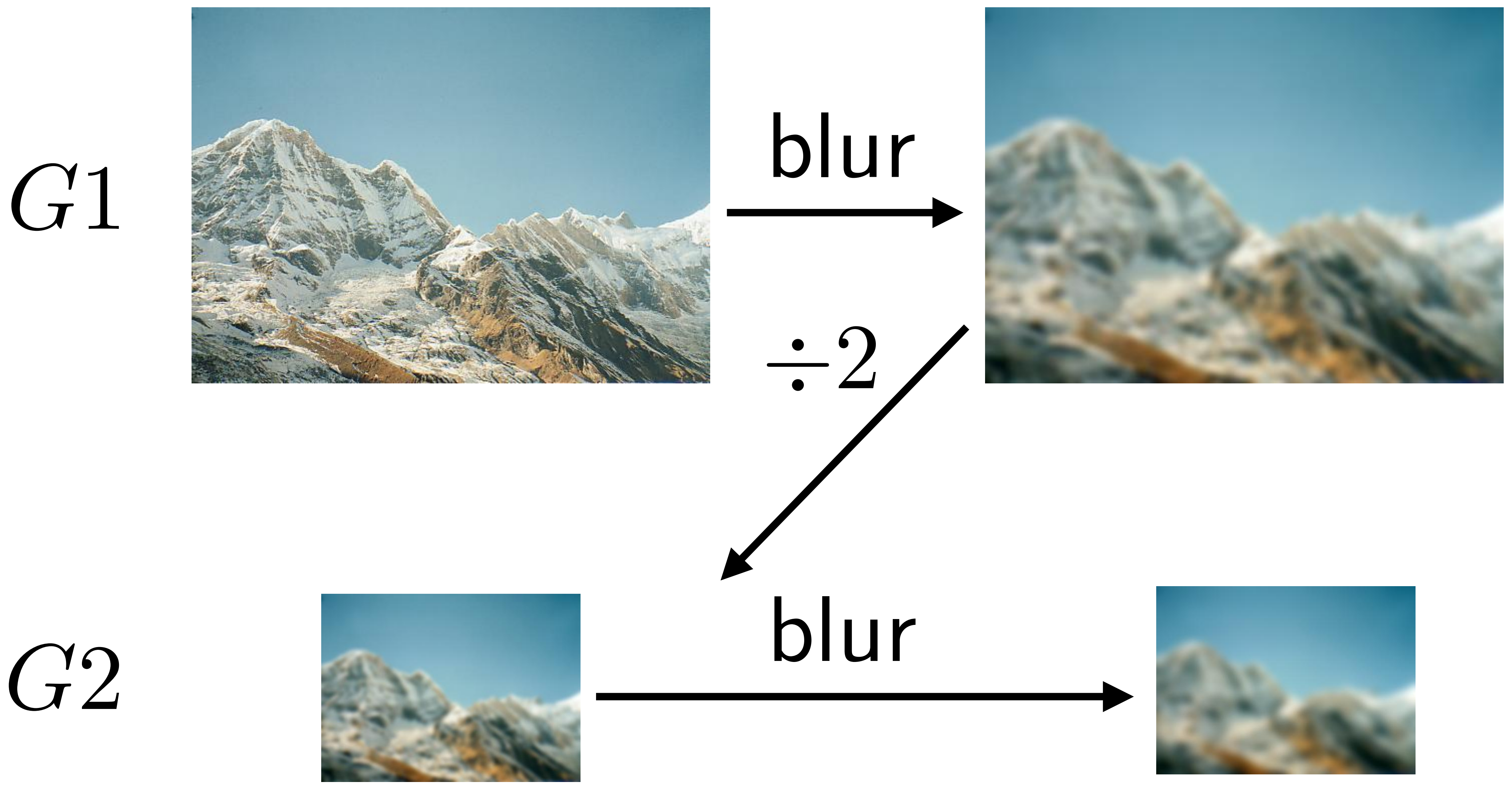
Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

[Assignment 2]



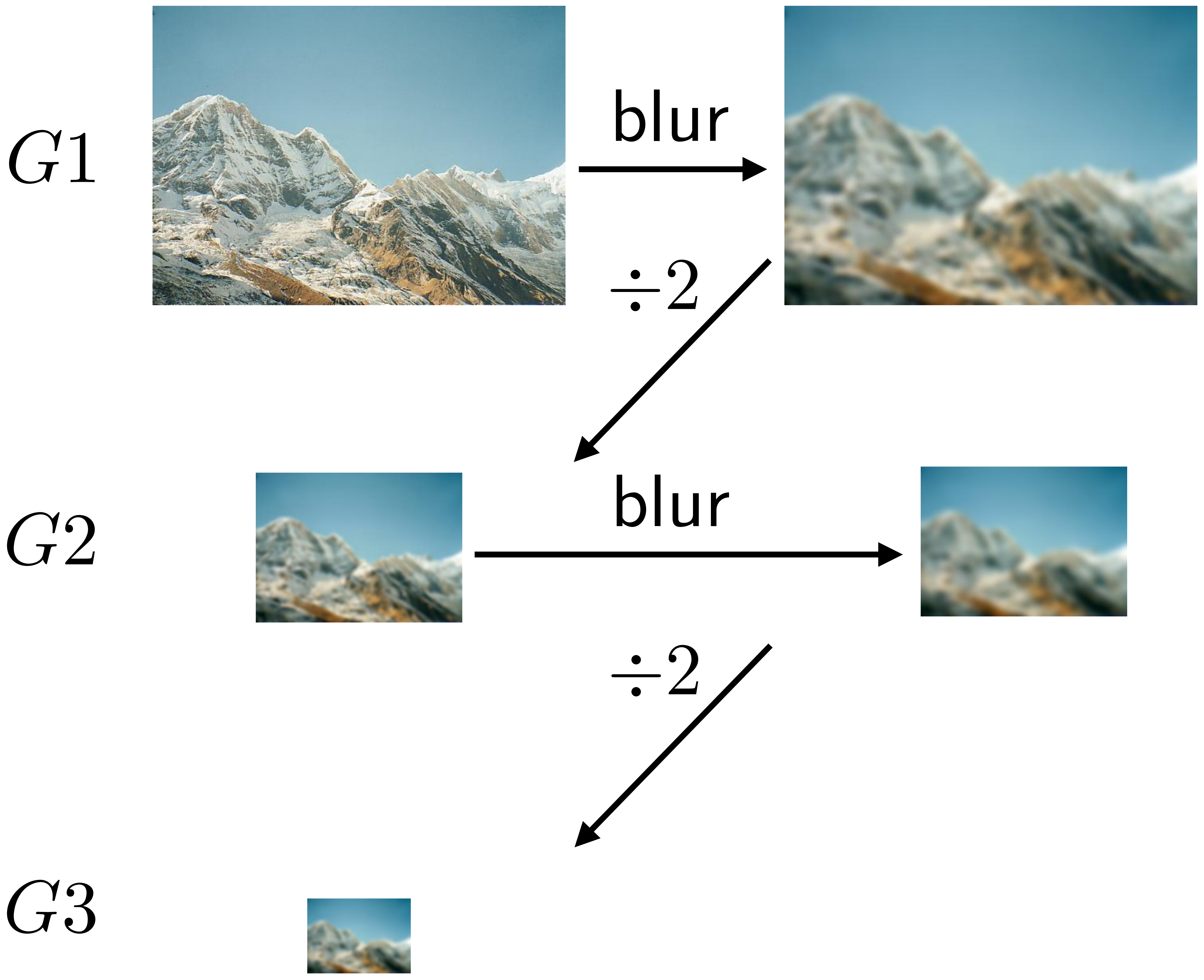
Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

[Assignment 2]



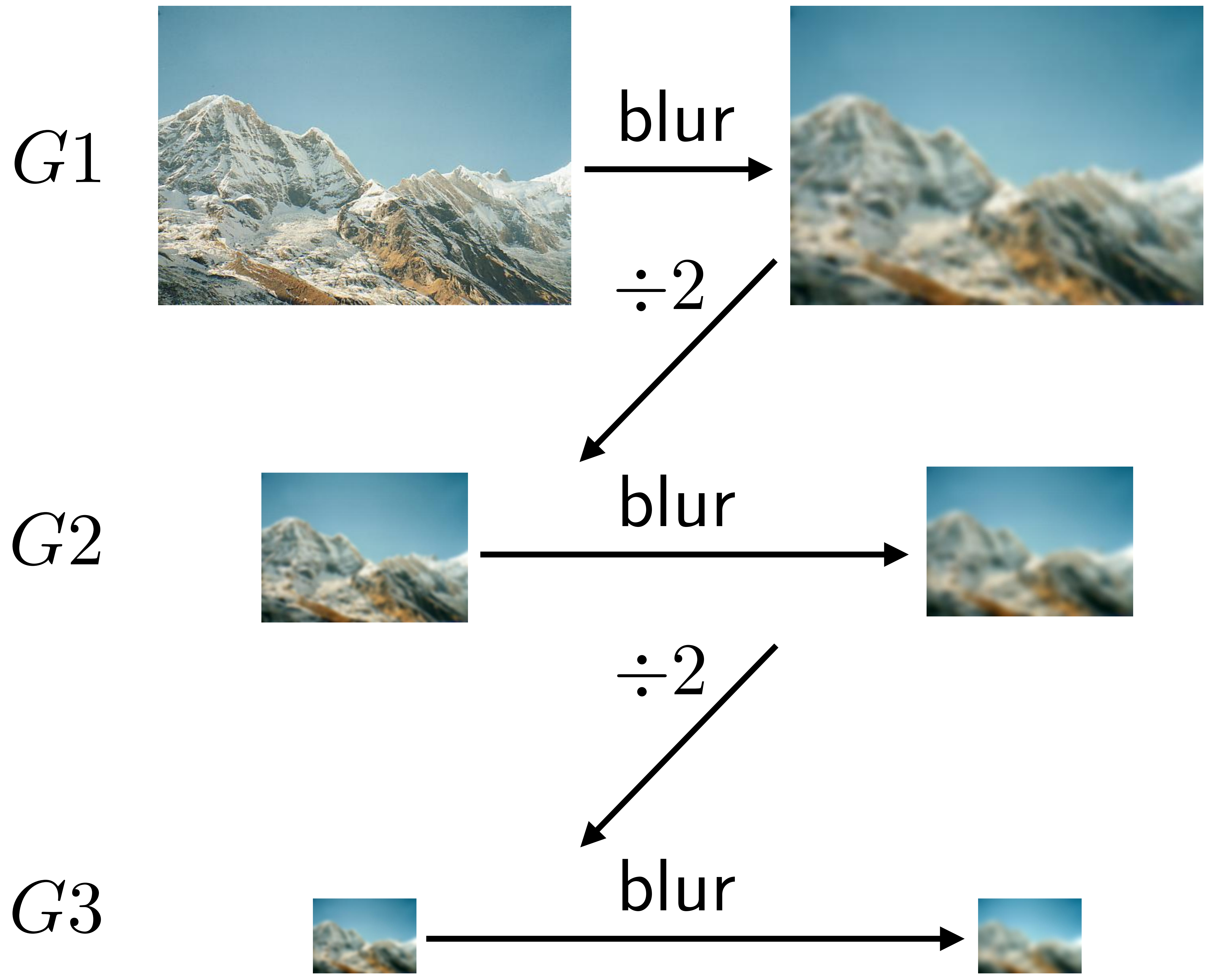
Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

[Assignment 2]



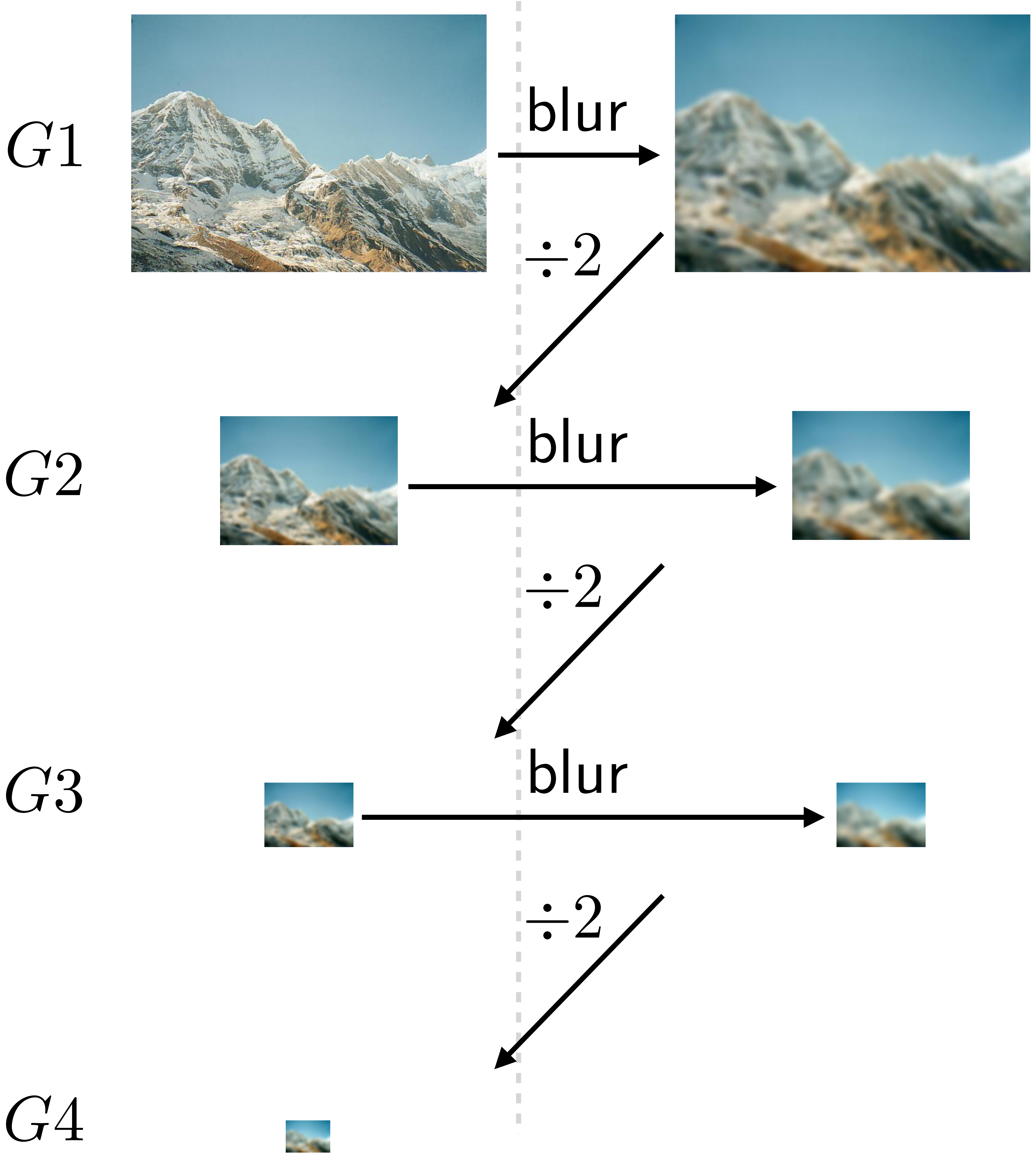
Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

[Assignment 2]



Blur with a Gaussian kernel, then select every 2nd pixel

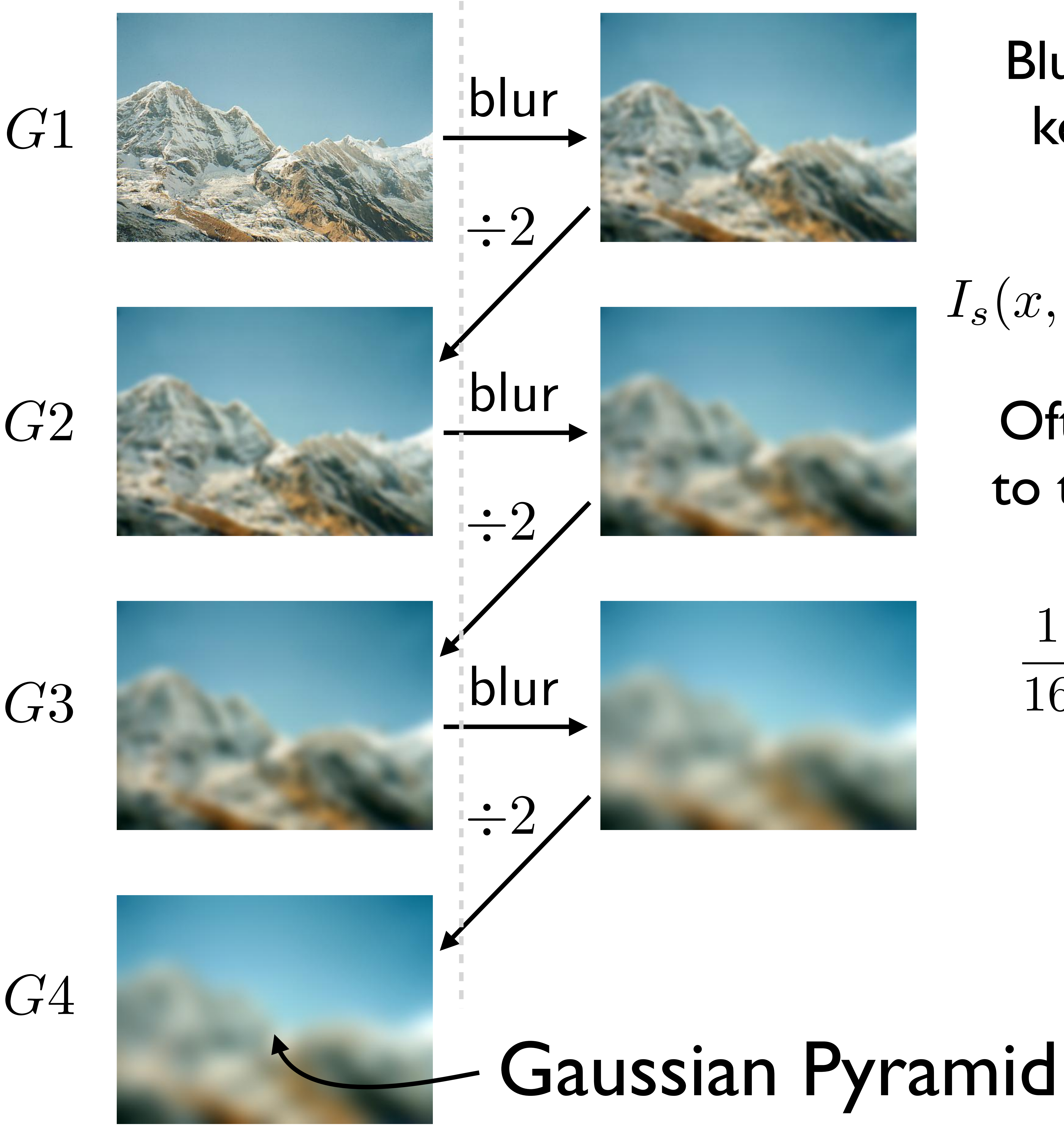
$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

Gaussian Pyramid

[Assignment 2]



Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

Often approximations to the Gaussian kernel are used, e.g.,

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

[Assignment 2]

Sampling Theory (informal)

Question: For a bandlimited signal, what if you **oversample** (i.e., sample at greater than the Nyquist rate)

Sampling Theory (informal)

Question: For a bandlimited signal, what if you **oversample** (i.e., sample at greater than the Nyquist rate)

Answer: Nothing bad happens! Samples are redundant and there are wasted bits

Sampling Theory (informal)

Question: For a bandlimited signal, what if you **oversample** (i.e., sample at greater than the Nyquist rate)

Answer: Nothing bad happens! Samples are redundant and there are wasted bits

Question: For a bandlimited signal, what if you **undersample** (i.e., sample at less than the Nyquist rate)

Sampling Theory (informal)

Question: For a bandlimited signal, what if you **oversample** (i.e., sample at greater than the Nyquist rate)

Answer: Nothing bad happens! Samples are redundant and there are wasted bits

Question: For a bandlimited signal, what if you **undersample** (i.e., sample at less than the Nyquist rate)

Answer: Two bad things happen! Things are missing (i.e., things that should be there aren't). There are artifacts (i.e., things that shouldn't be there are)

How to Prevent **Aliasing?**

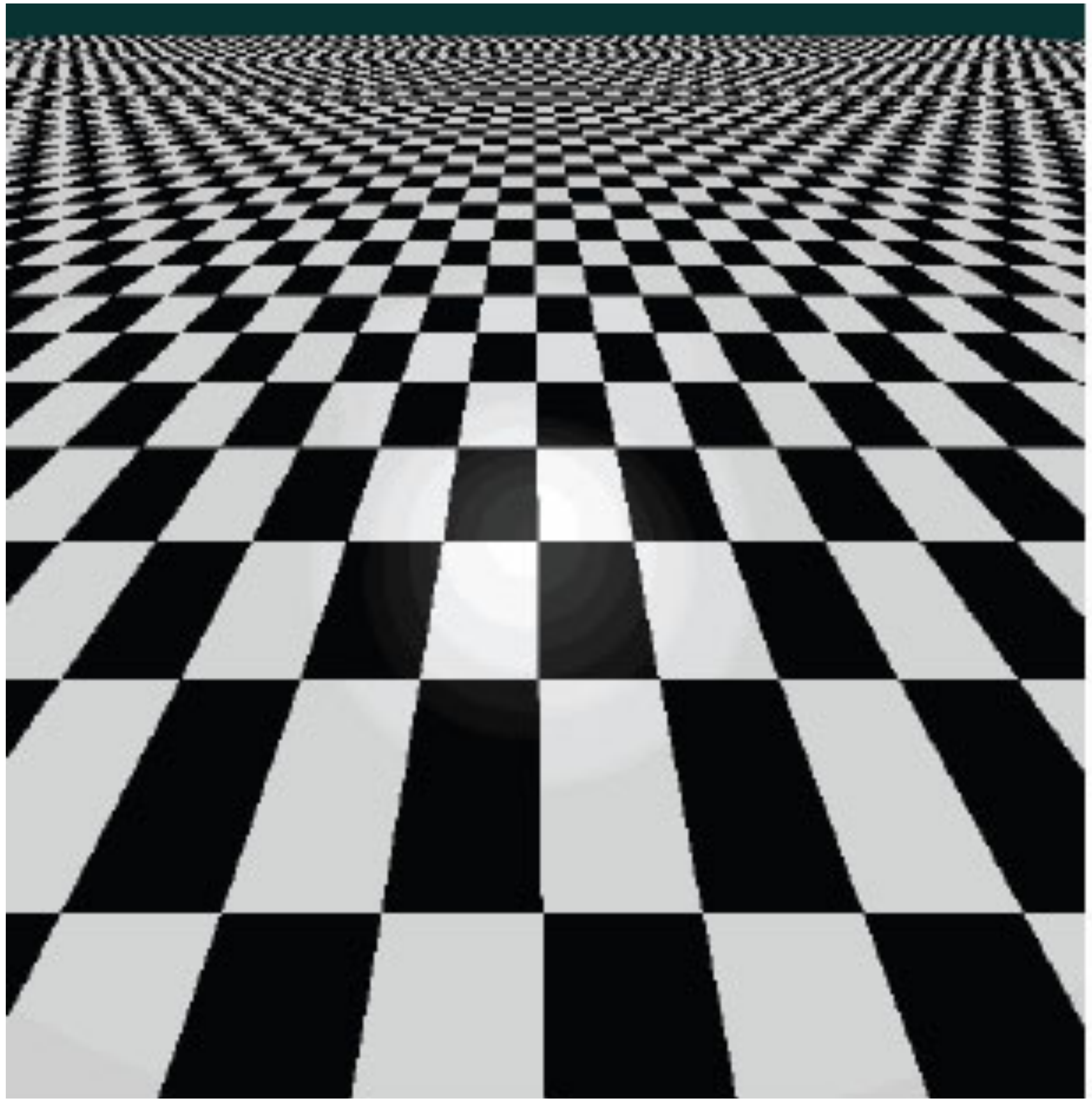
1. **Reduce the maximum frequency**, by low pass filtering i.e., **Smoothing** before sampling.

How to Prevent **Aliasing?**

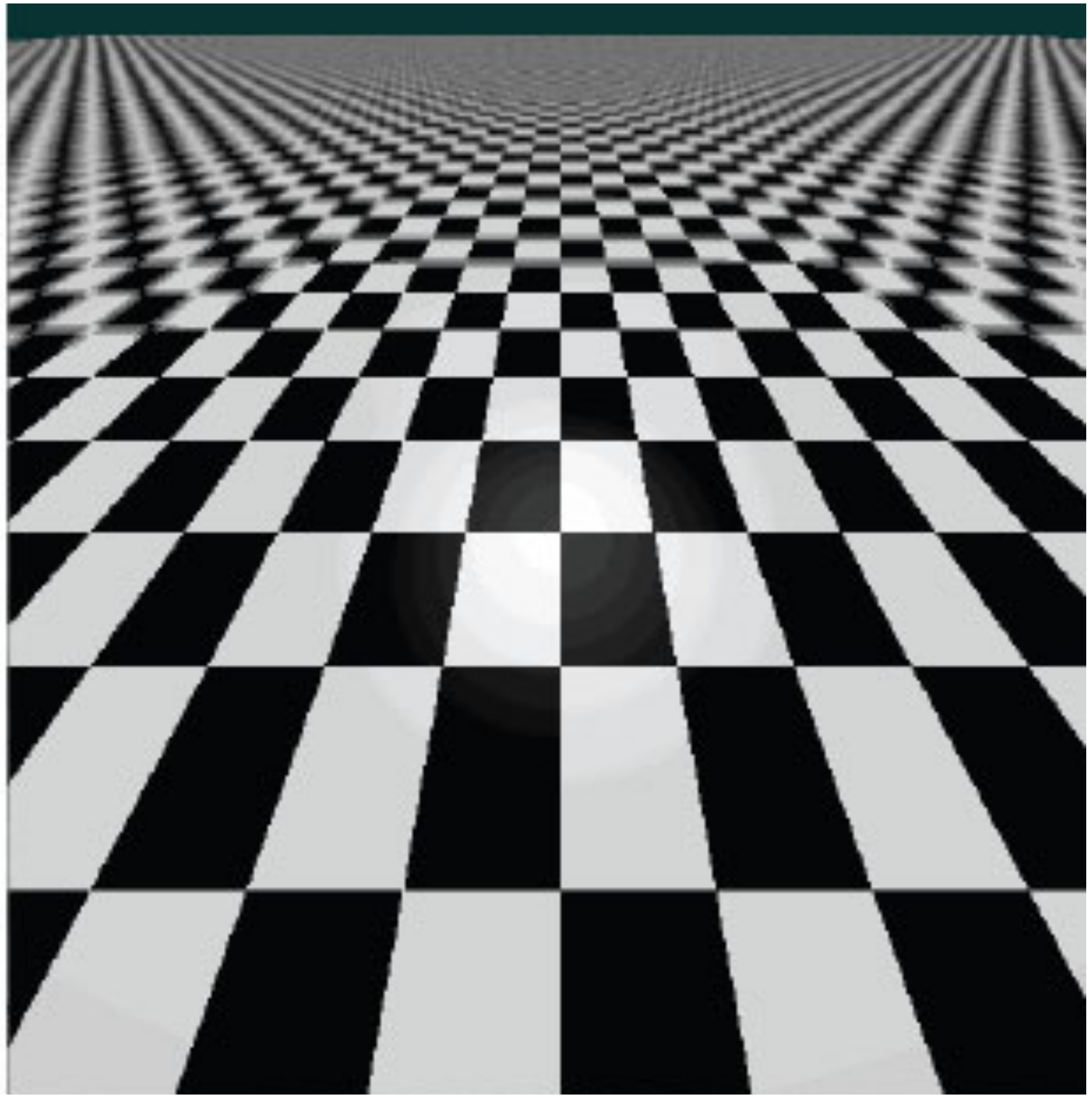
1. **Reduce the maximum frequency**, by low pass filtering i.e., **Smoothing** before sampling.

2. **Sample more frequently** i.e., oversampling — sample more than you think you need and average (i.e., area sampling)

Aliasing



aliasing artifacts



anti-aliasing by oversampling

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Temporal Aliasing

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Temporal Aliasing

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Temporal Aliasing

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Temporal Aliasing

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

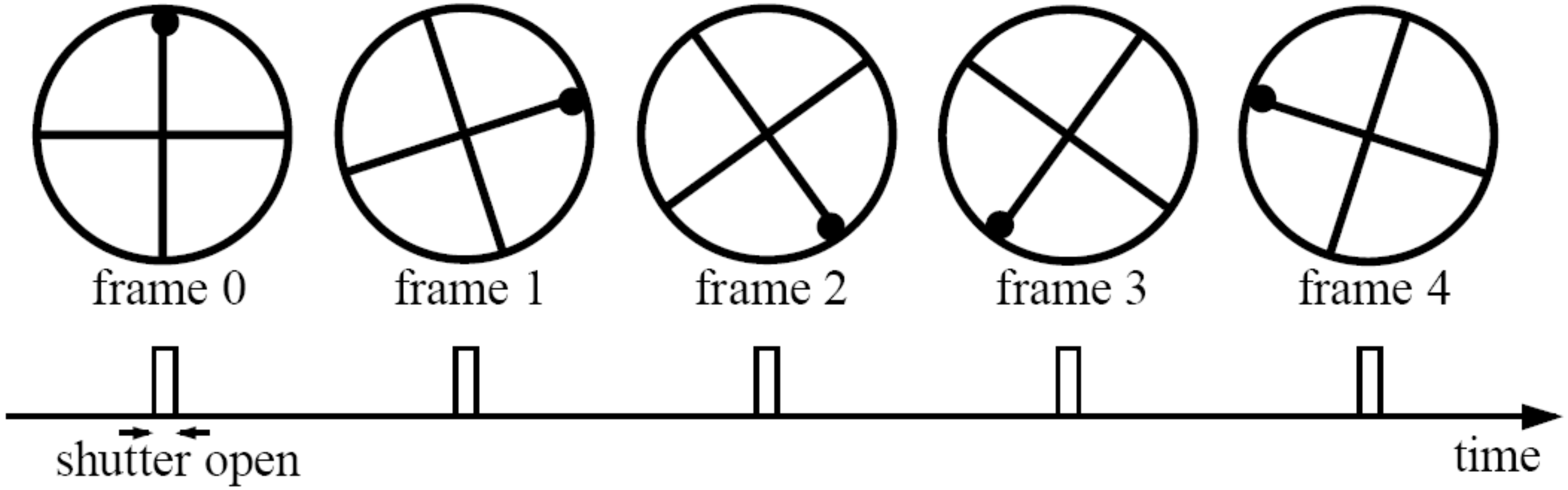
Temporal Aliasing

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Temporal Aliasing

Imagine a spoked wheel moving to the right (rotating clockwise). Mark wheel with dot so we can see what's happening.

If camera shutter is only open for a fraction of a frame time (frame time = 1/30 sec. for video, 1/24 sec. for film):



Without dot, wheel appears to be rotating slowly backwards! (counterclockwise)

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Sampling Theory (informal)

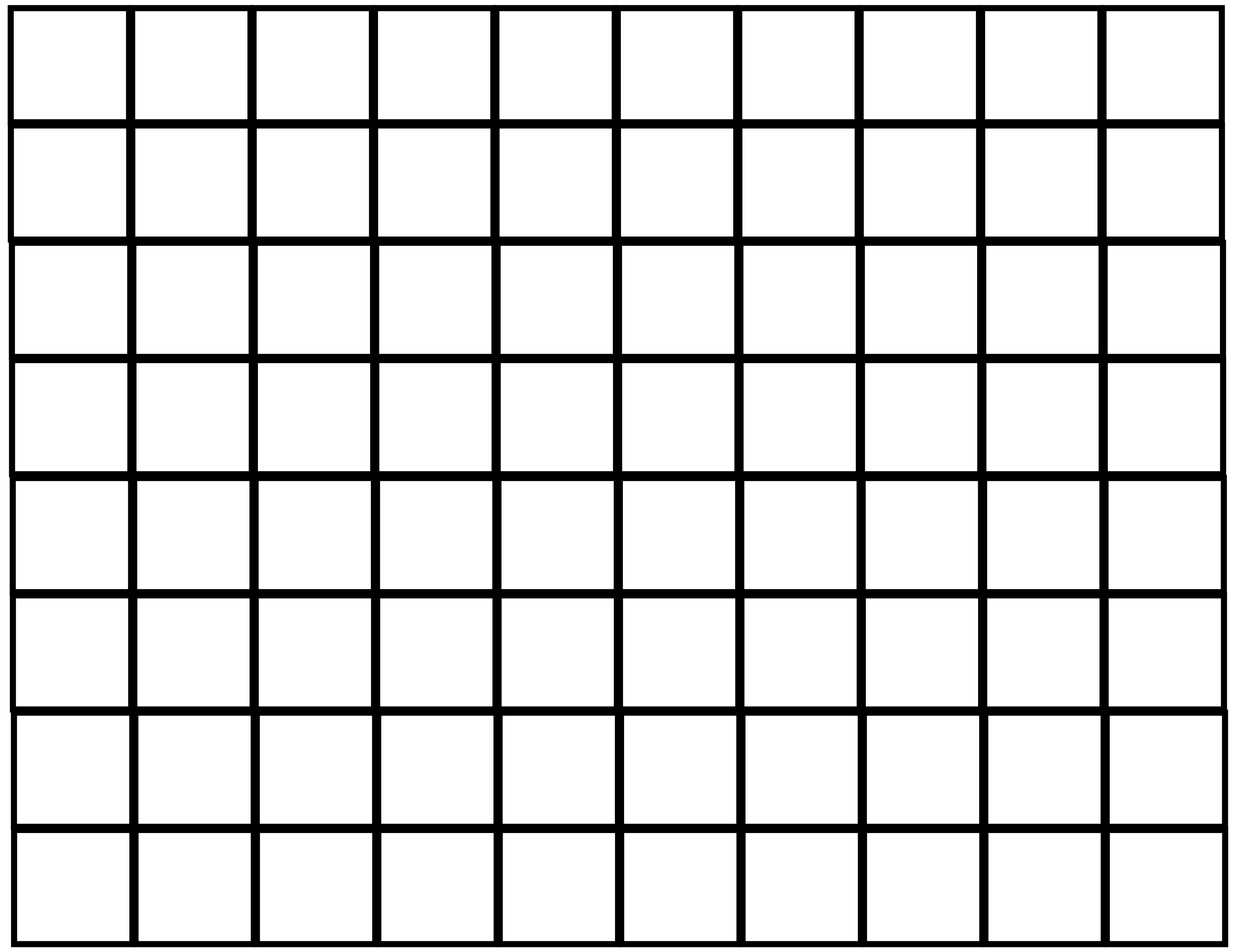
Sometimes **undersampling** is unavoidable, and there is a trade-off between "things missing" and "artifacts."

— **Medical imaging**: usually try to maximize information content, tolerate some artifacts

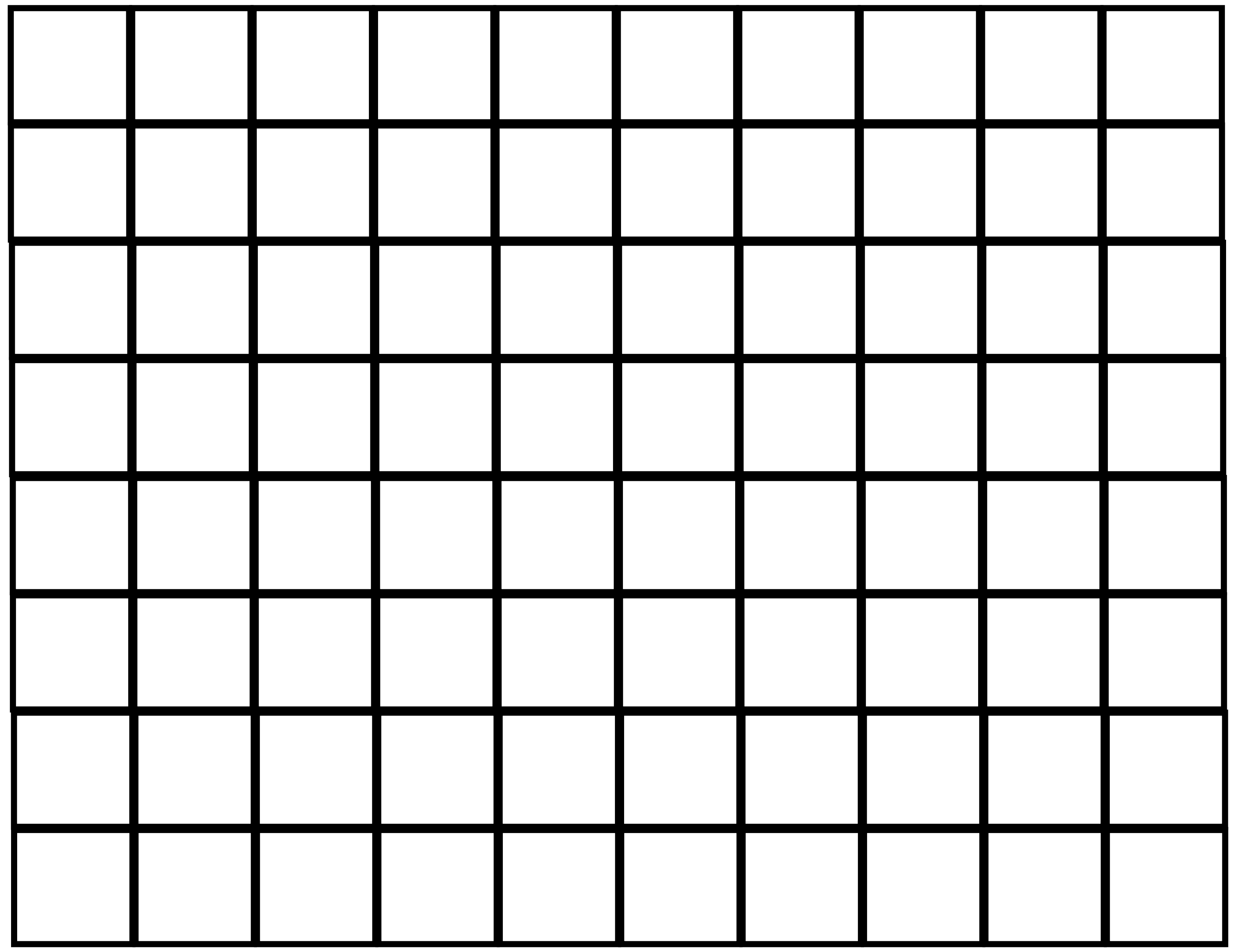
— **Computer graphics**: usually try to minimize artifacts, tolerate some information missing

Example

Sensor Resolution: 10 x 8



Sensor Resolution: 10 x 8



Example

Sensor Resolution: 10 x 8

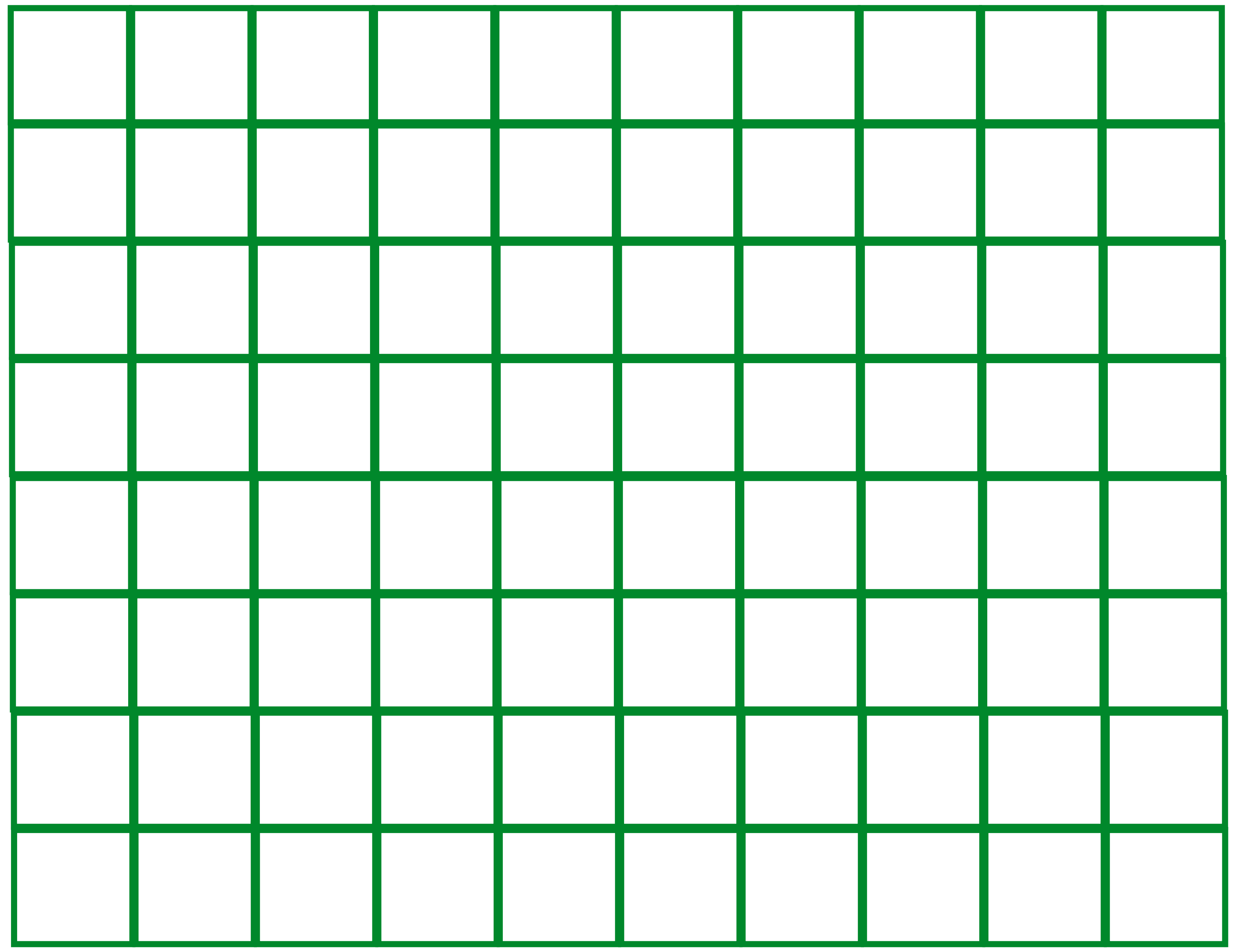


Image Resolution: 10 x 8

Sensor Resolution: 10 x 8

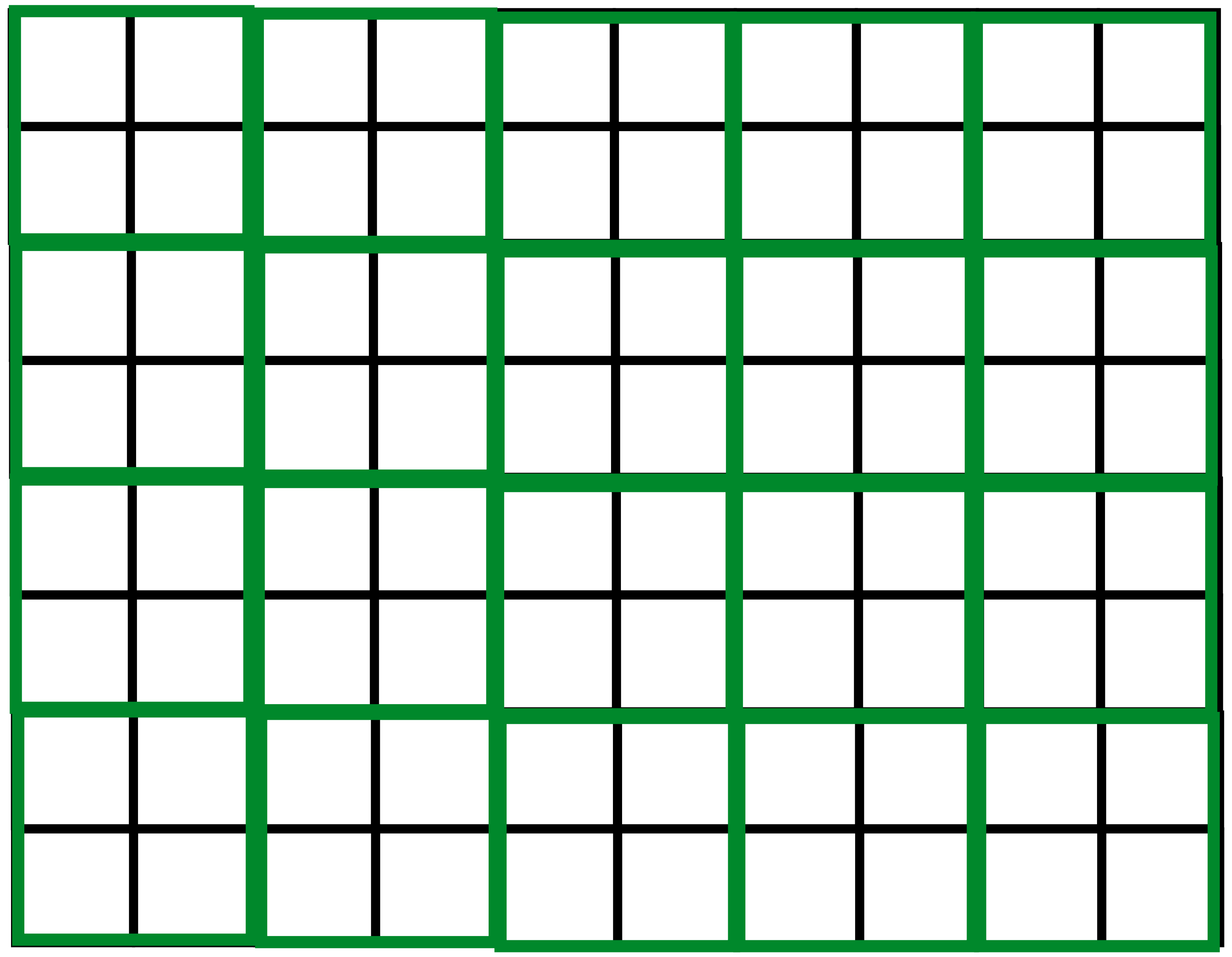
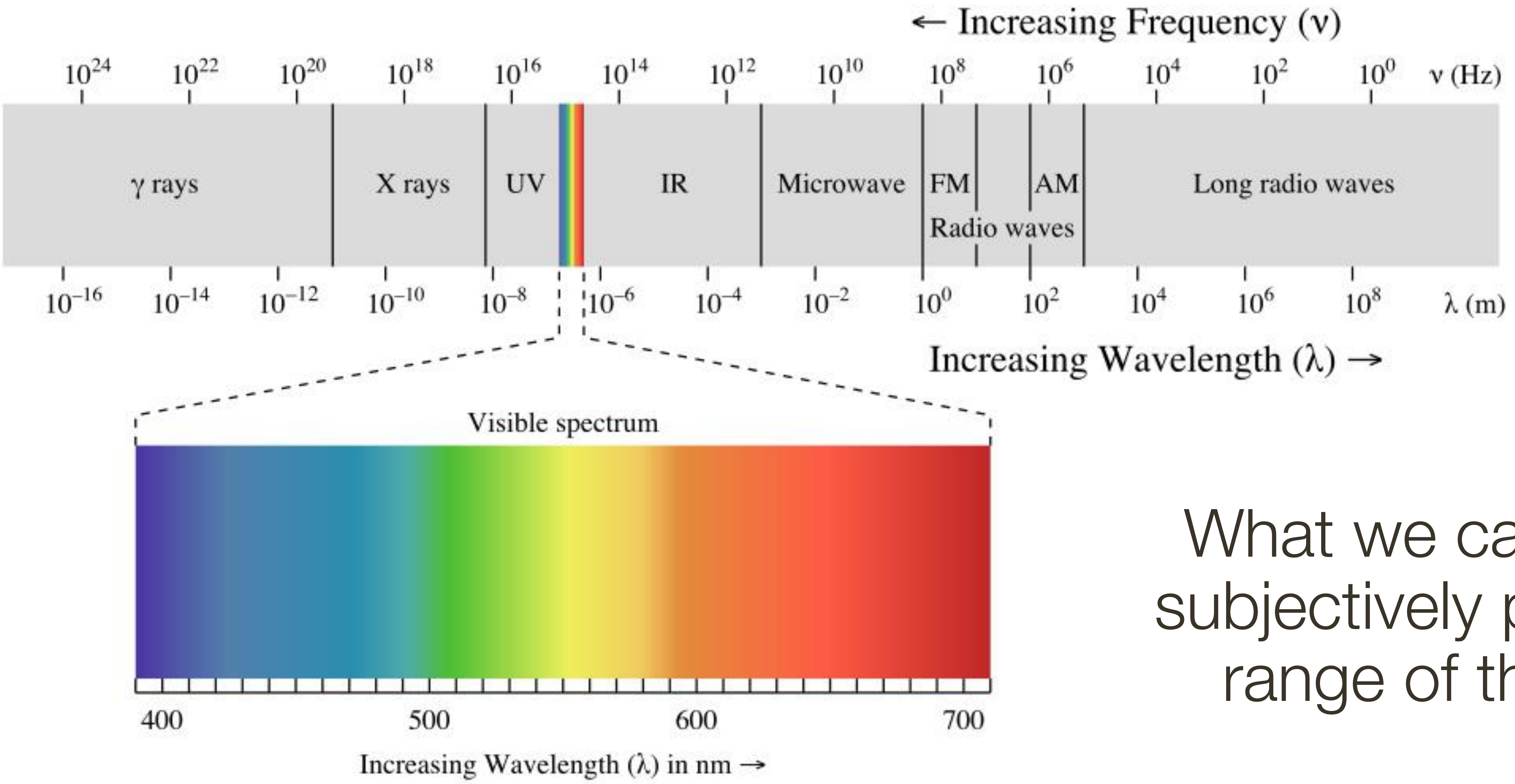


Image Resolution: 5 x 4

Color is an Artifact of Human Perception

"Color" is **not** an objective physical property of light (electromagnetic radiation).

Instead, light is characterized by its wavelength.

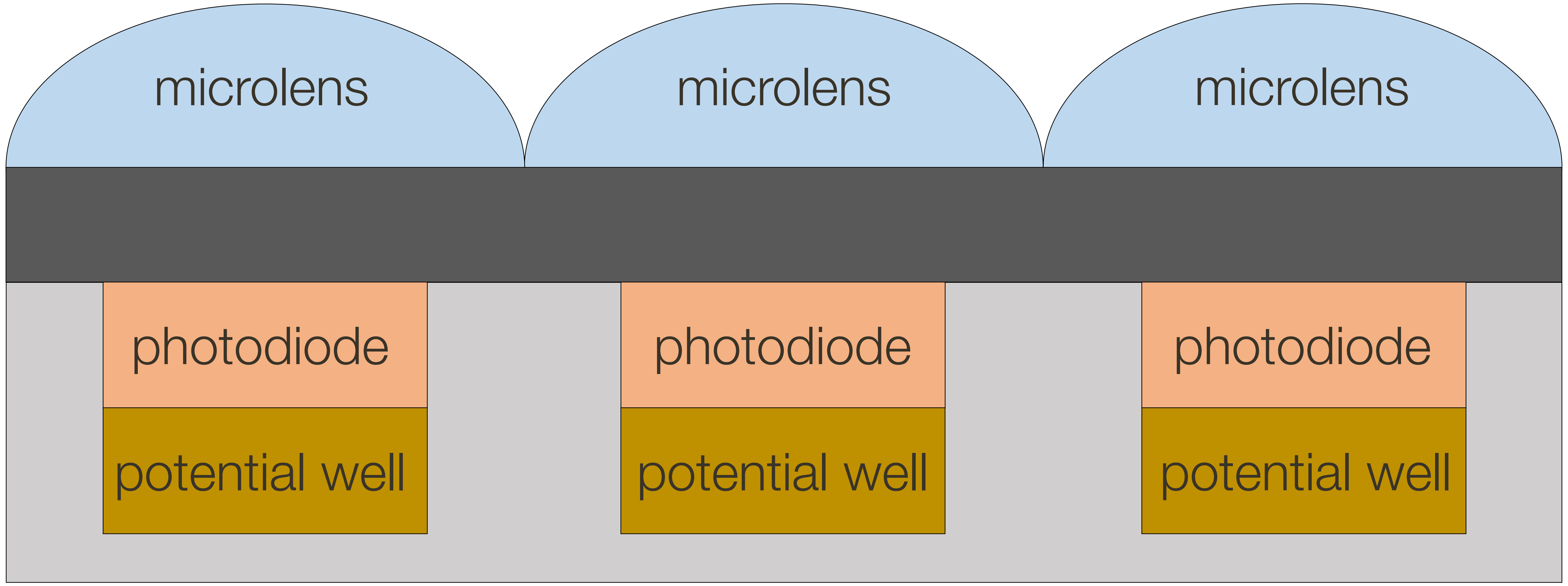


electromagnetic spectrum

What we call "color" is how we subjectively perceive a very small range of these wavelengths.

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filter Arrays (CFA)

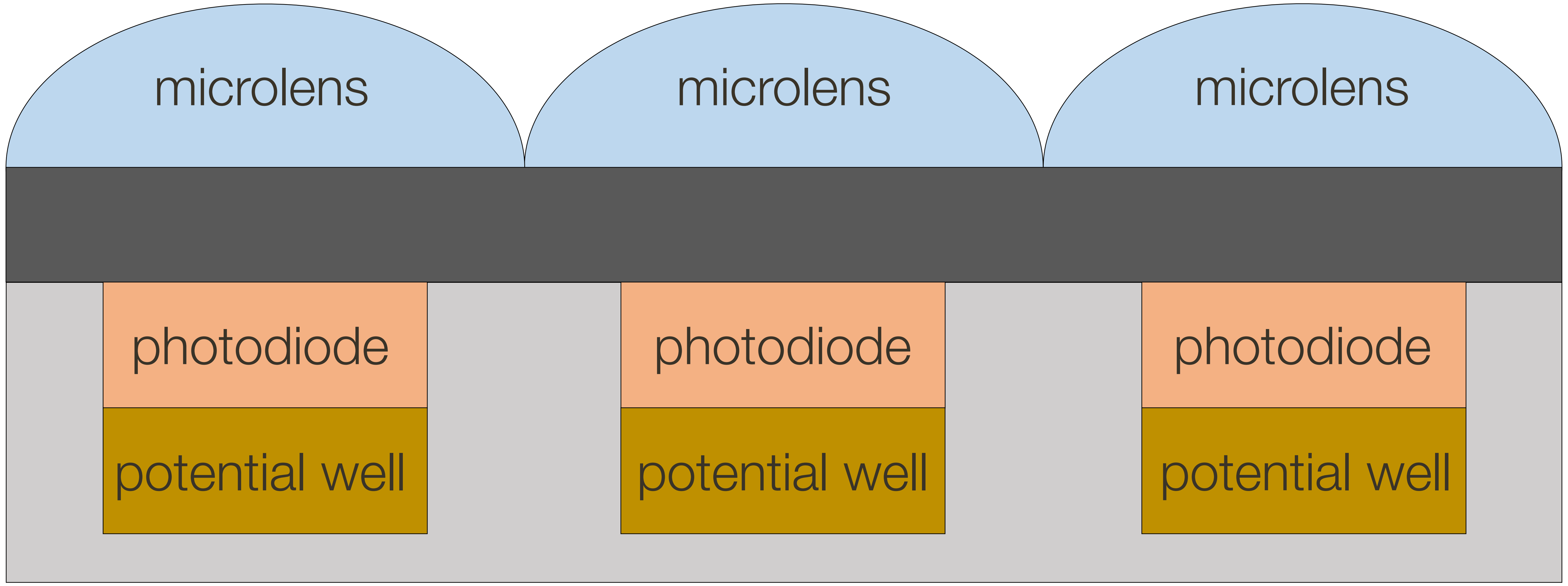


Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filter Arrays (CFA)

In addition to a camera lens,

each pixel has a microns

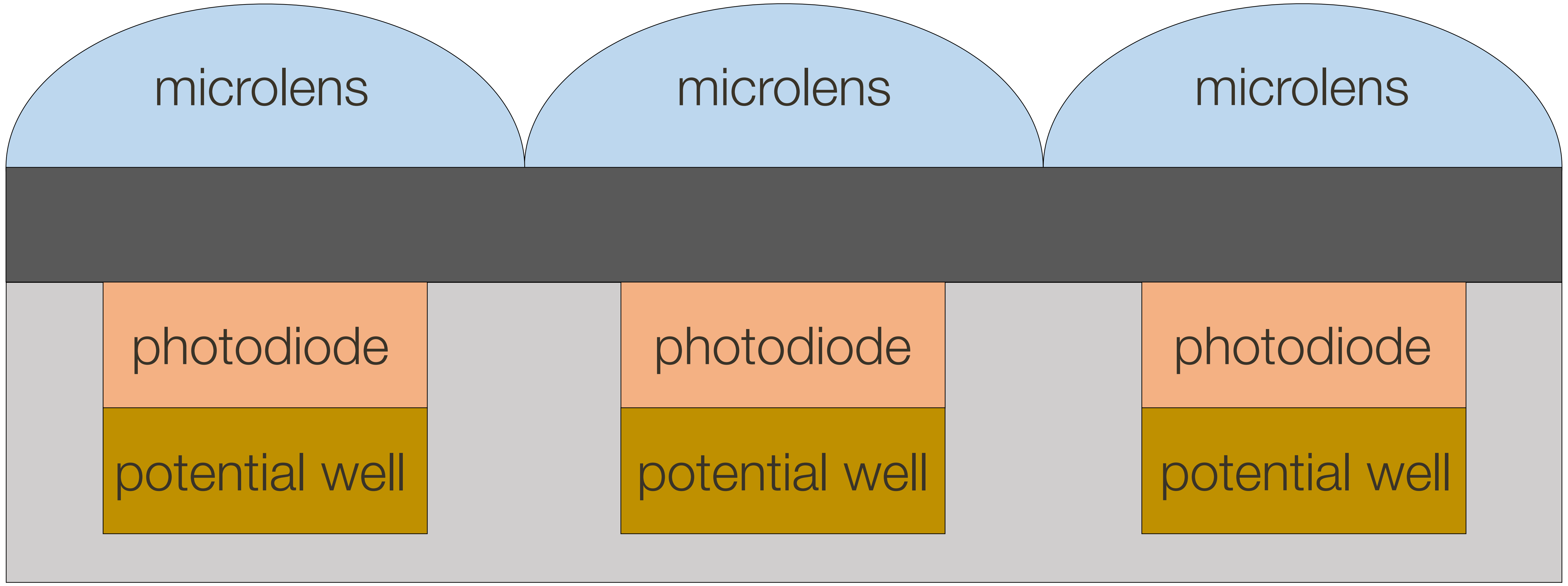


Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filter Arrays (CFA)

In addition to a camera lens,

each pixel has a microns



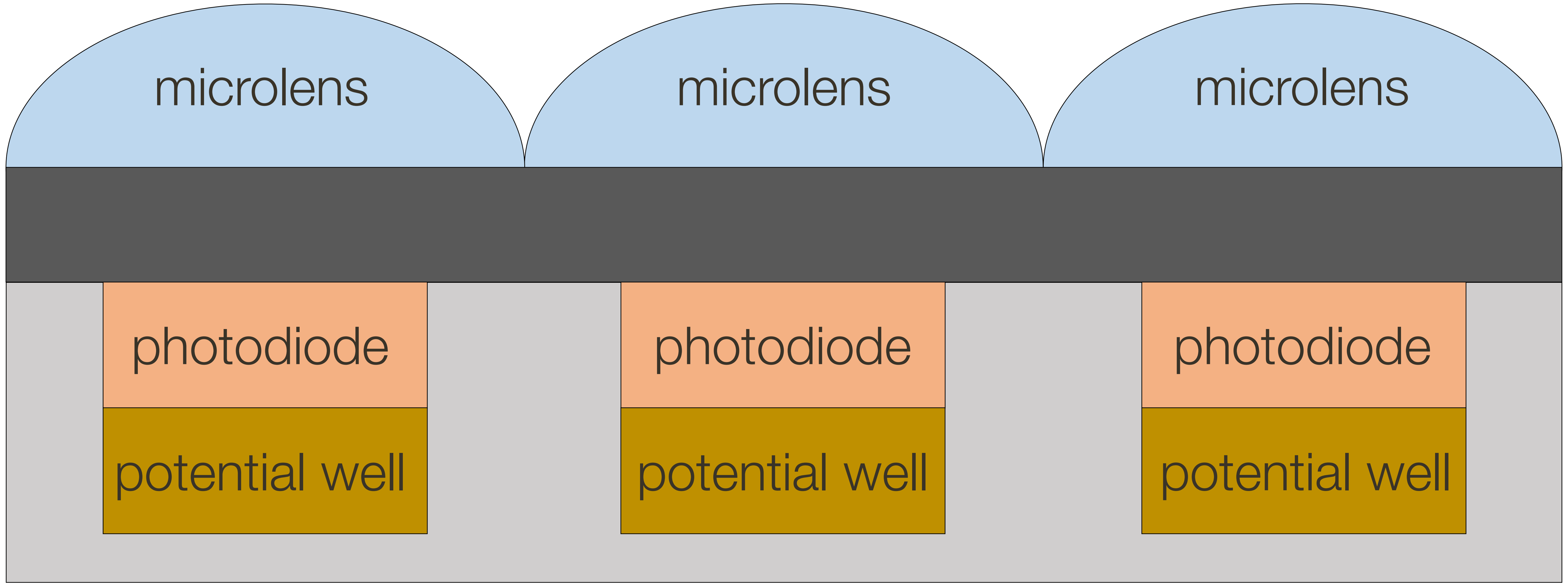
Photodiode: converts photons to electrons

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filter Arrays (CFA)

In addition to a camera lens,

each pixel has a microns



Photodiode: converts photons to electrons

Electrons stored in the **potential well**, until they are read off

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

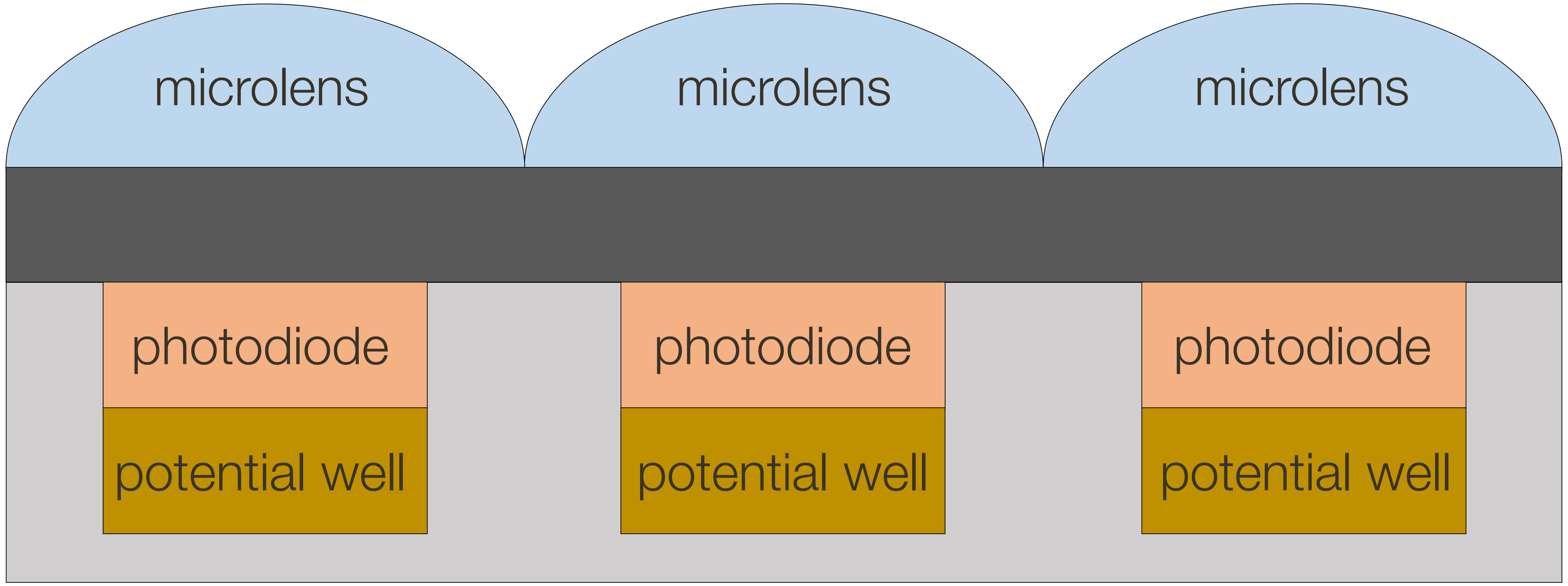
Color Filter Arrays (CFA)

In addition to a camera lens,

each pixel has a microns

Photodiode: converts photons to electrons

Electrons stored in the **potential well**, until they are read off

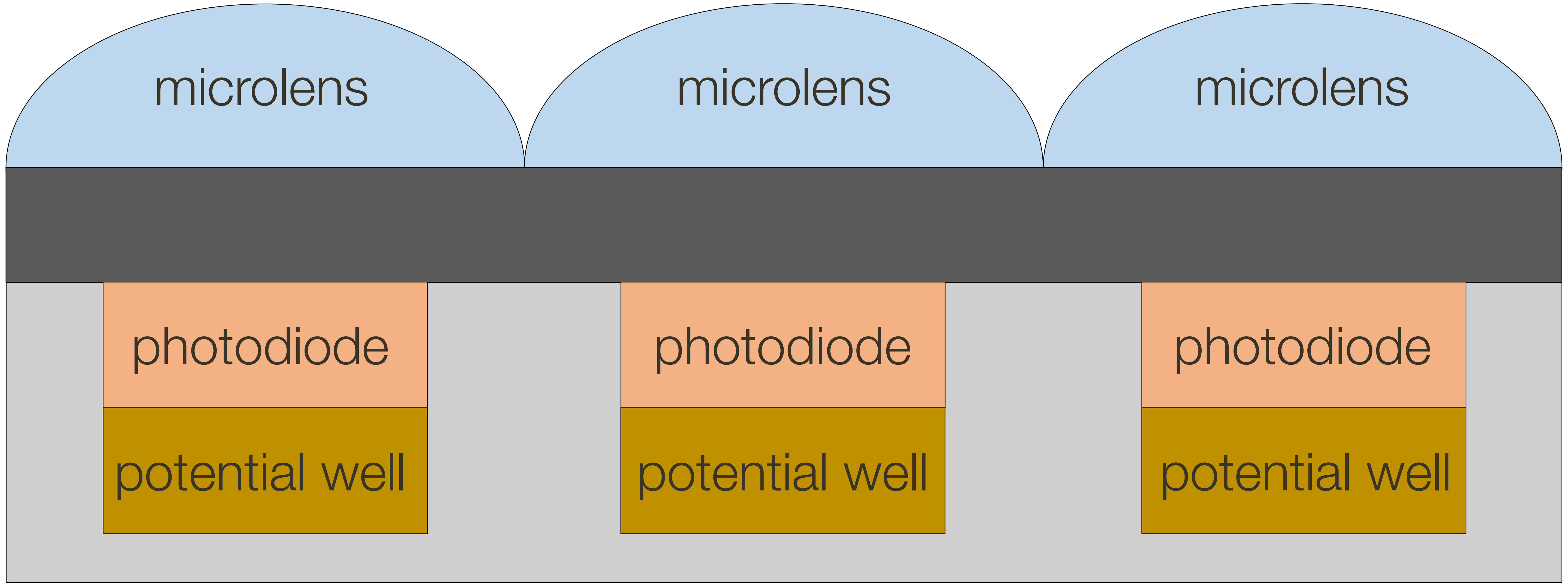


Quantum efficiency: fraction of photons being “detected” through this process
(human eye QE: 20%, film cameras QE: 10%, CCD QE: 80%)

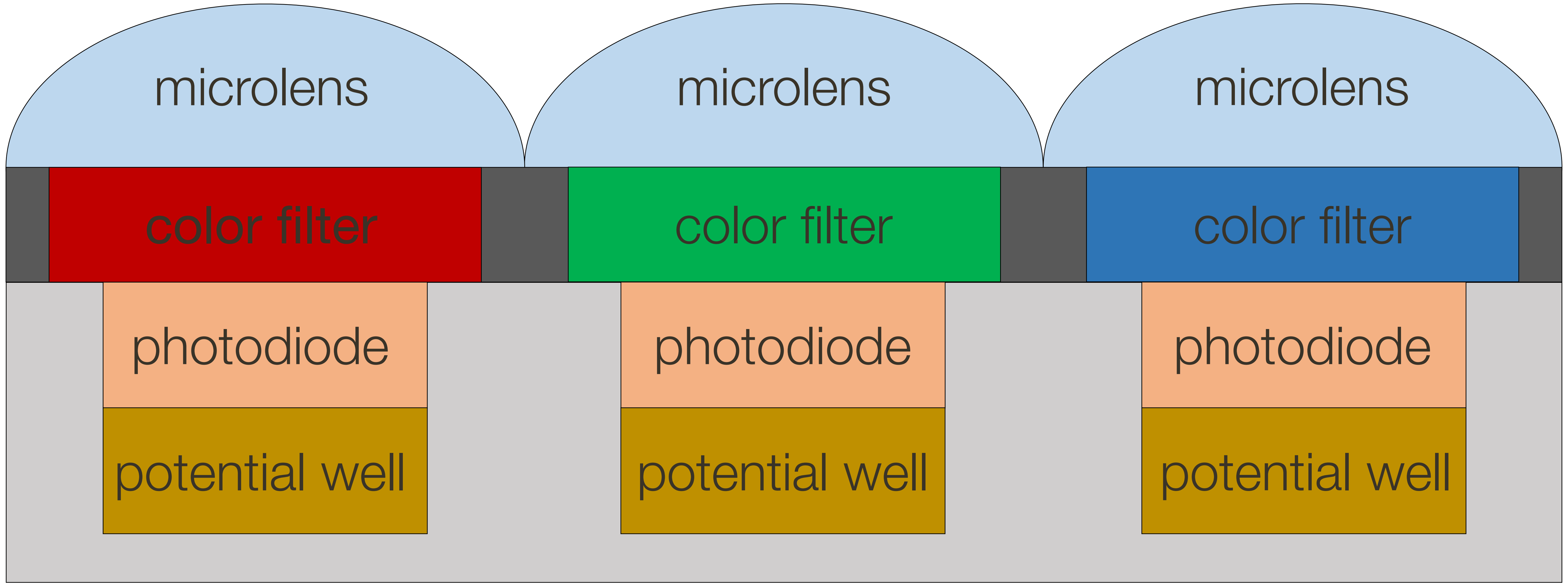
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filter Arrays (CFA)

Issue: Color Filter Array (SFA) by itself has no way of distinguishing wavelengths of light, just ability to record the amount of light incident on an element



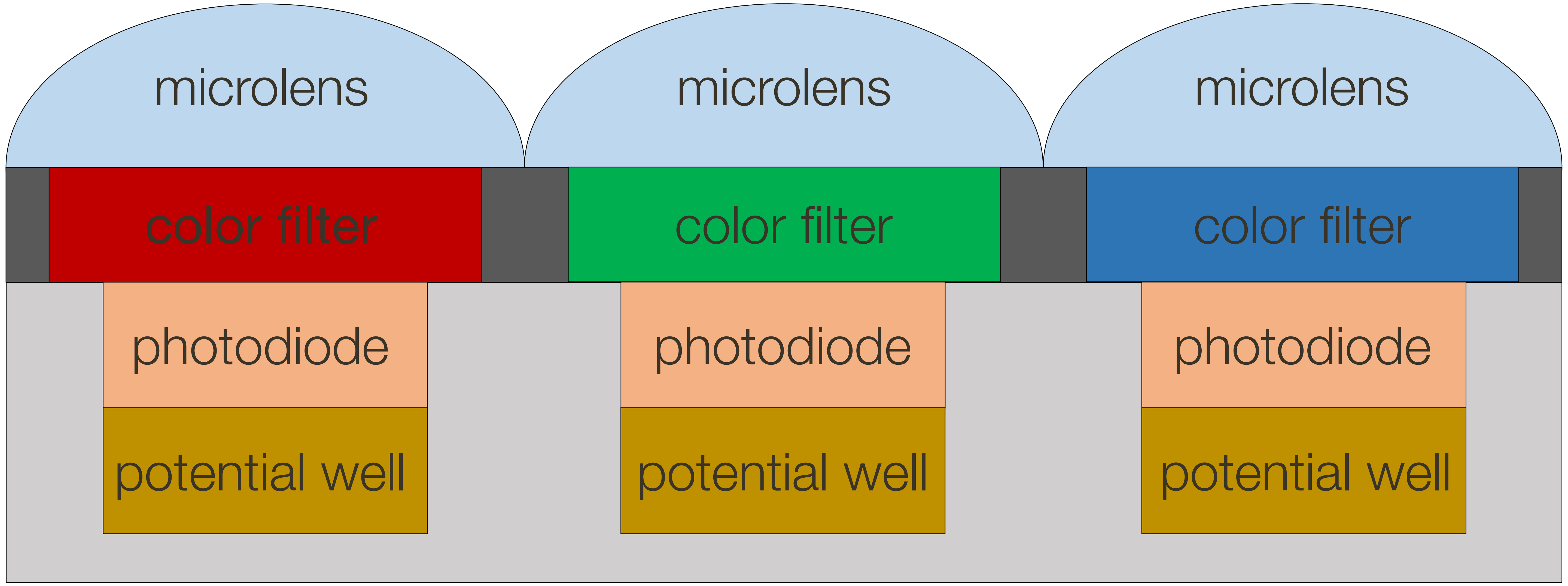
Color Filter Arrays (CFA)



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filter Arrays (CFA)

Implication: Only certain wavelengths of light are recorded at a given pixel



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filters

Two **design choices**:

— What spectral sensitivity functions $f(\lambda)$ to use for each color filter?

— How to spatially arrange ("**mosaic**") different color filters?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

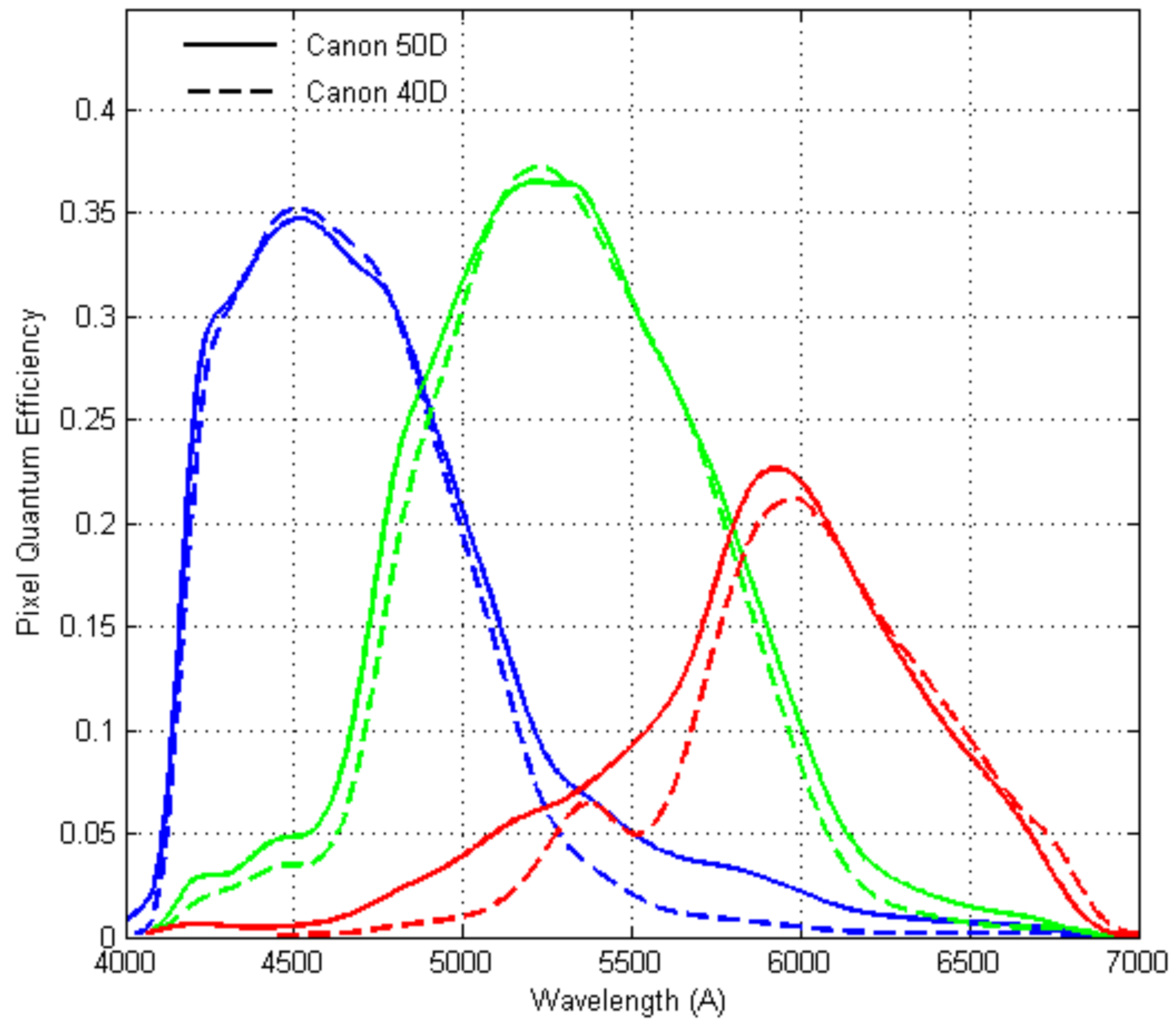
Color Filters

Two **design choices**:

— What spectral sensitivity functions $f(\lambda)$ to use for each color filter?

— How to spatially arrange ("**mosaic**") different color filters?

Canon 50D



$f(\lambda)$

Generally do not match human sensitivity

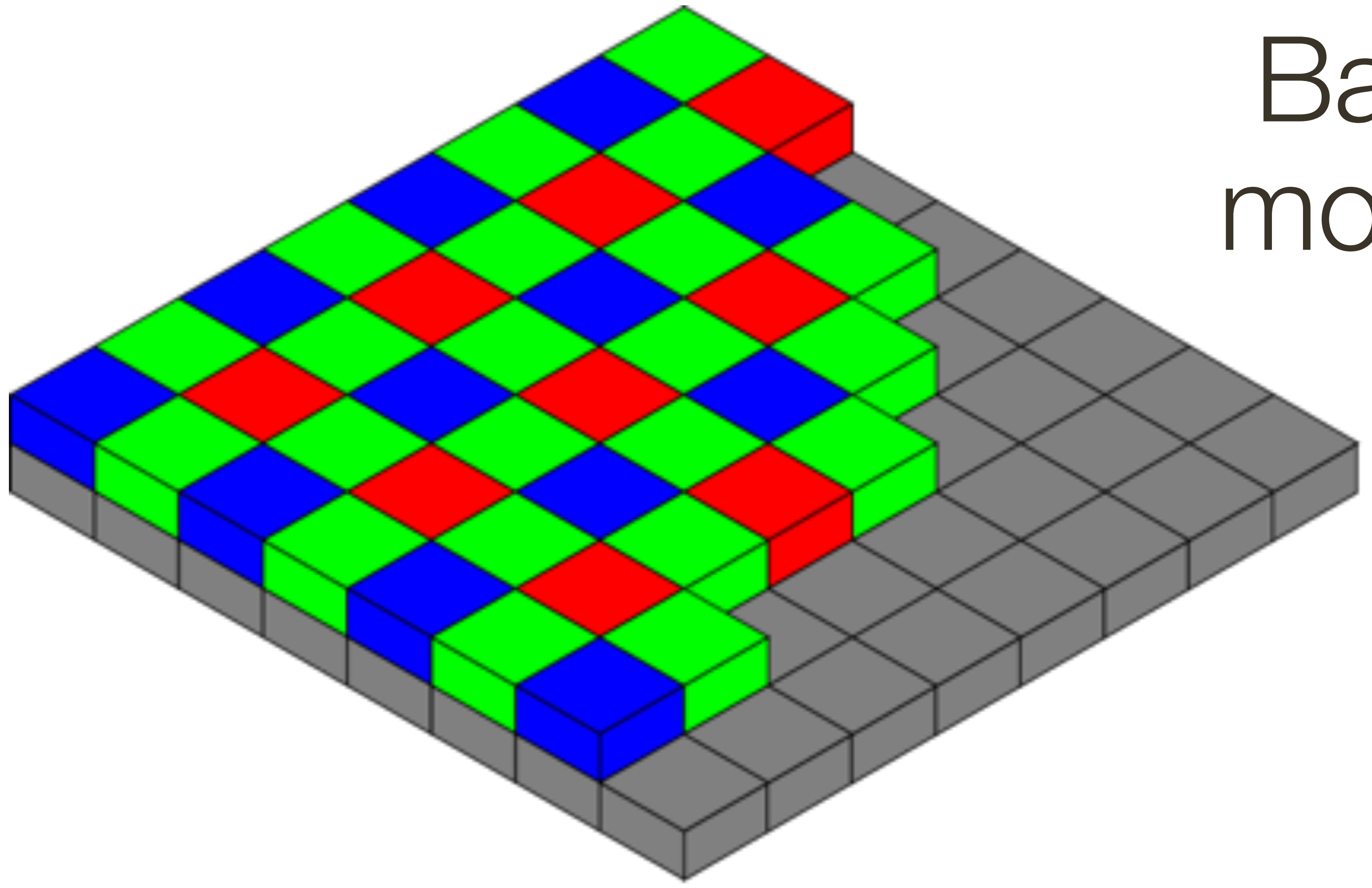
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filters

Two **design choices**:

— What spectral sensitivity functions $f(\lambda)$ to use for each color filter?

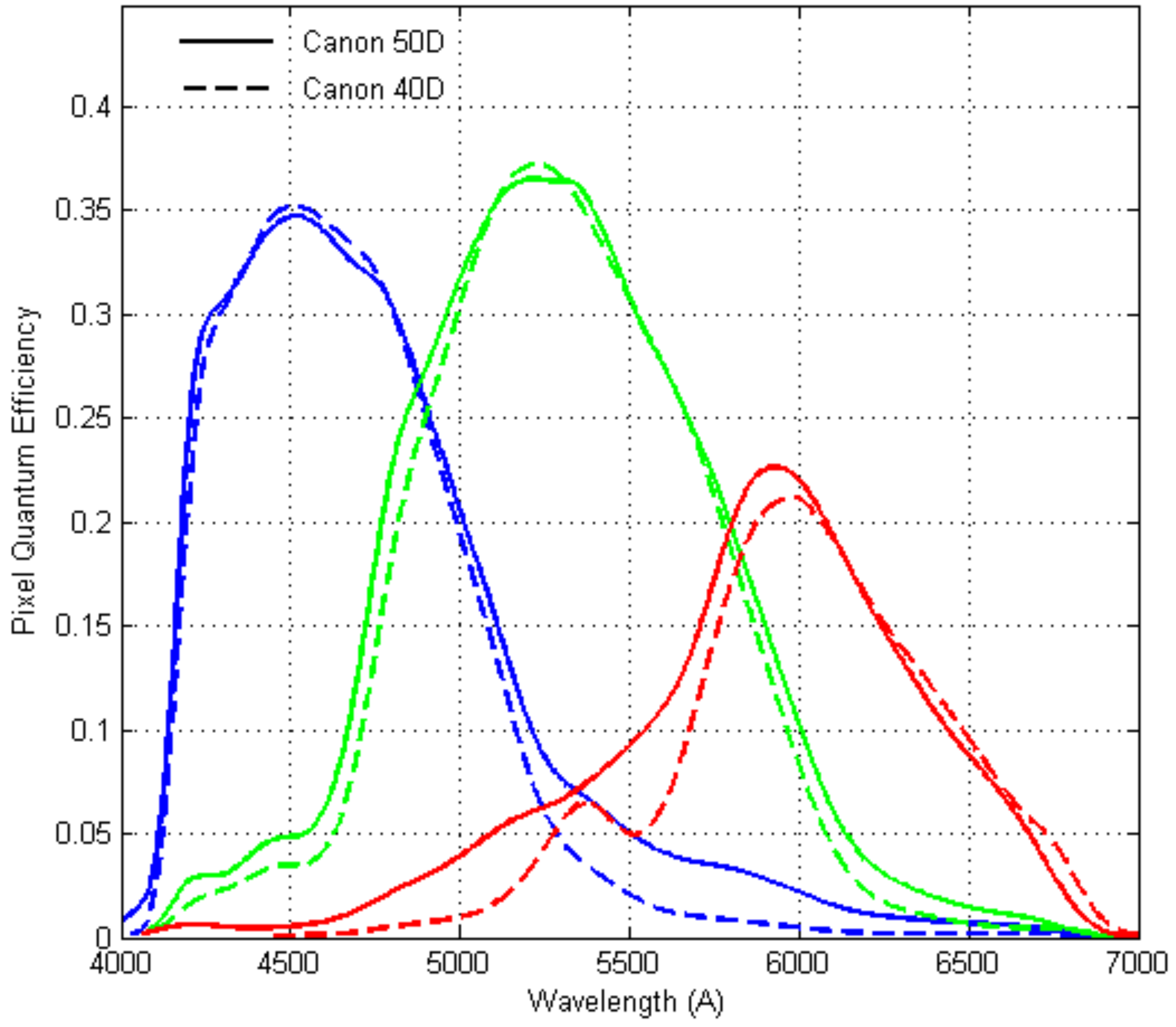
— How to spatially arrange ("**mosaic**") different color filters?



Bayer mosaic

Generally do not match human sensitivity

Canon 50D



$f(\lambda)$

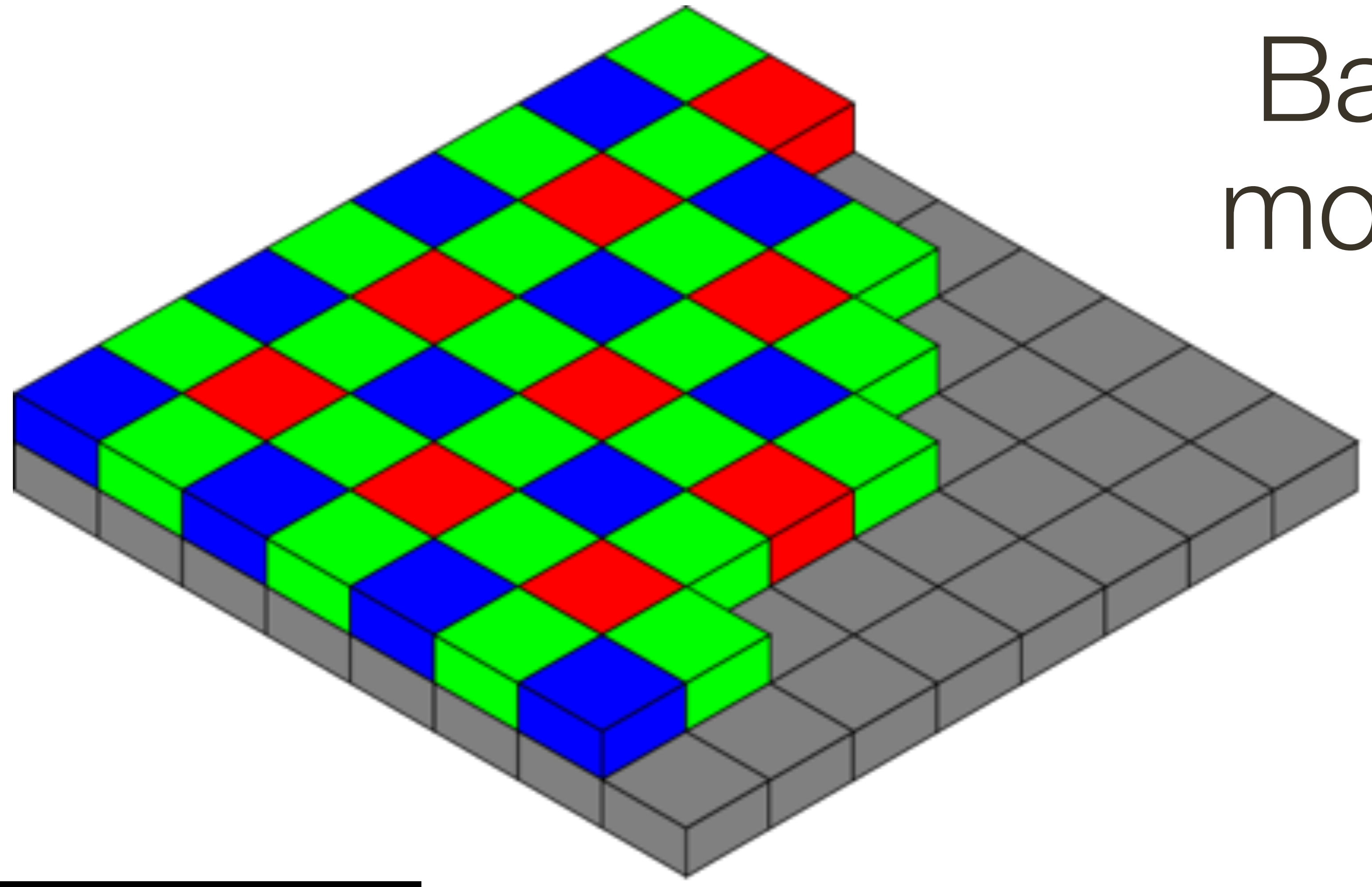
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Color Filters

Two **design choices**:

— What spectral sensitivity functions $f(\lambda)$ to use for each color filter?

— How to spatially arrange ("**mosaic**") different color filters?

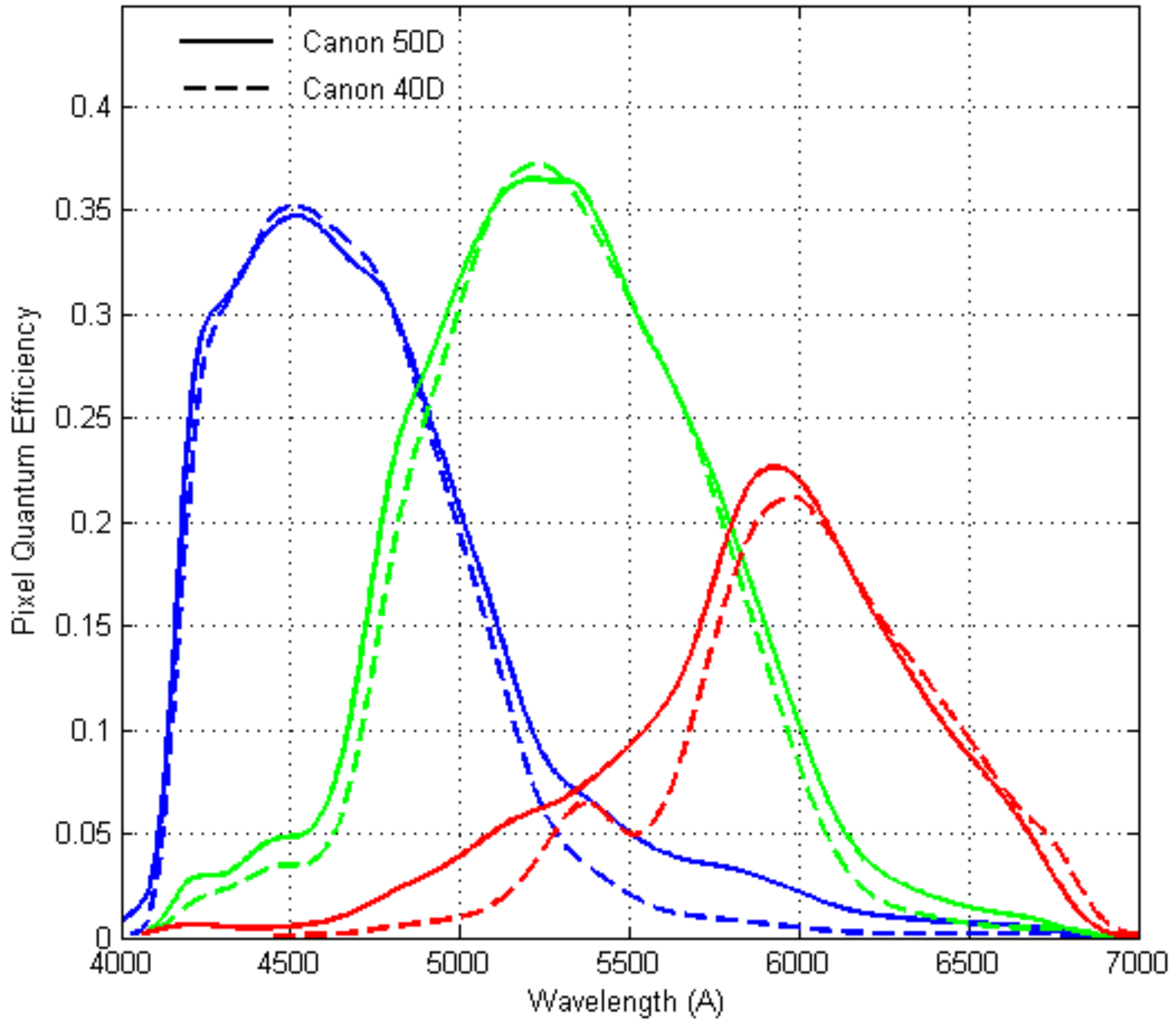


Bayer mosaic

Why more green pixels?

Generally do not match human sensitivity

Canon 50D

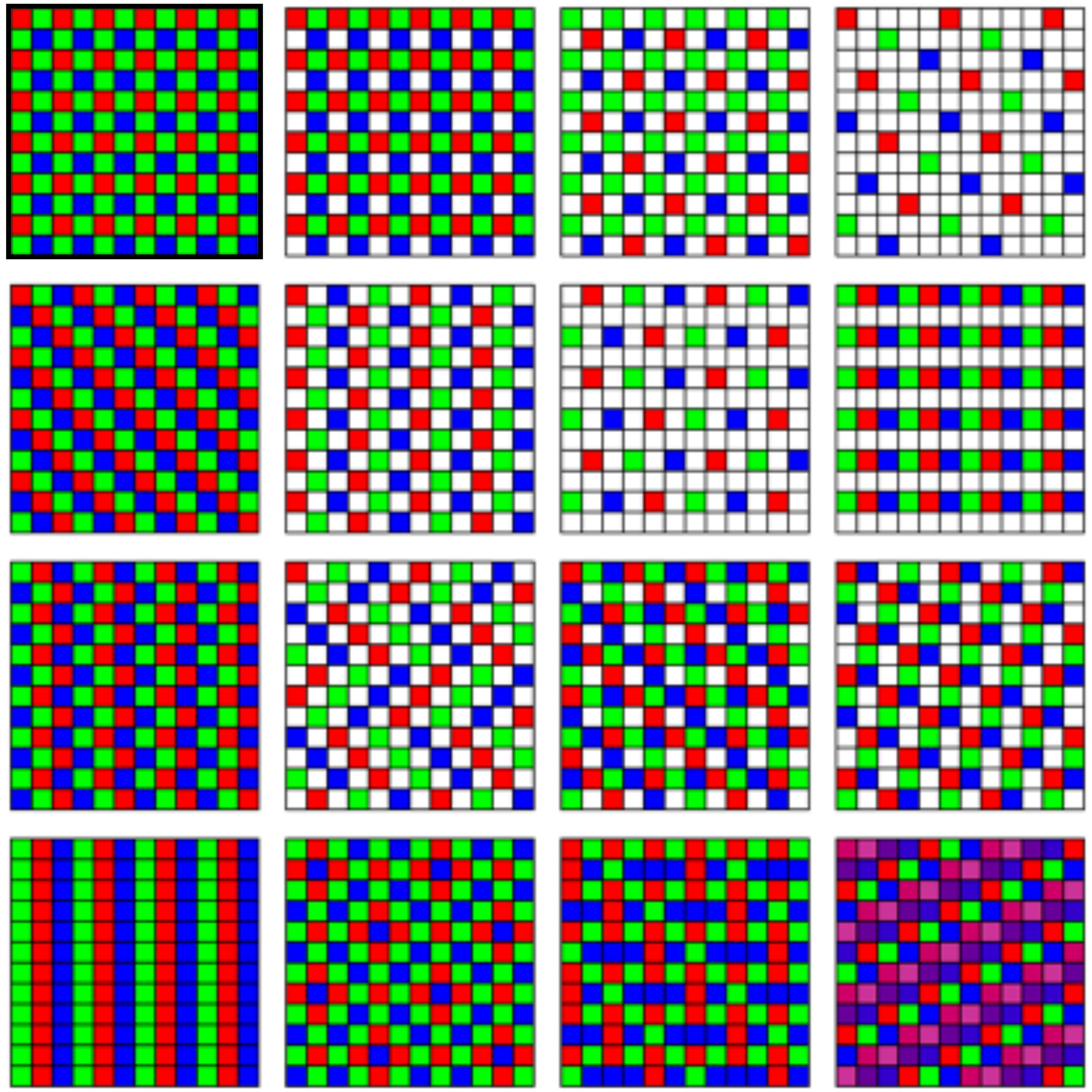


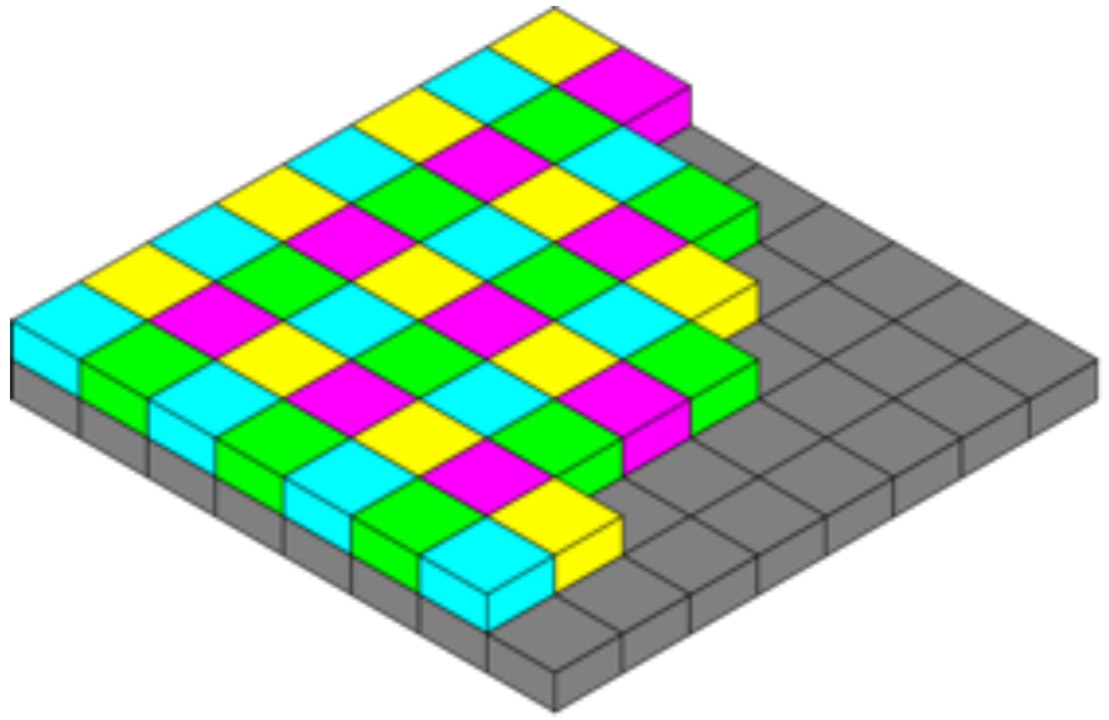
$f(\lambda)$

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Different Color Filter Arrays (CFAs)

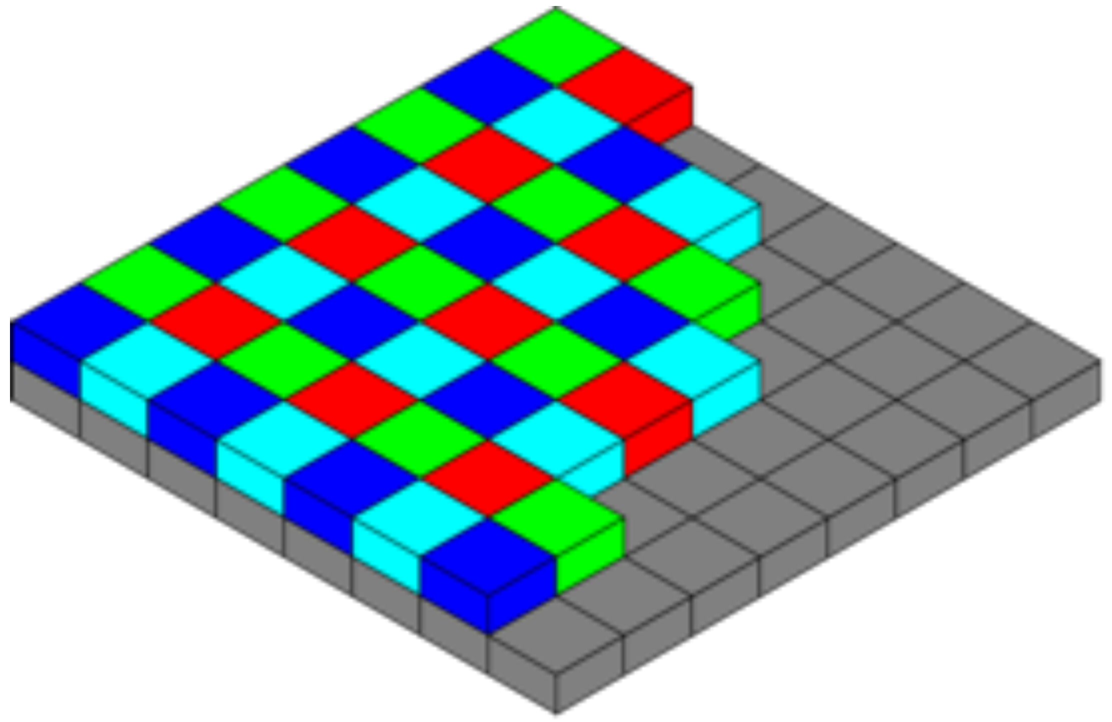
Finding the "**best**" CFA mosaic is an active research area.





CYGM

Canon IXUS, Powershot



RGBE

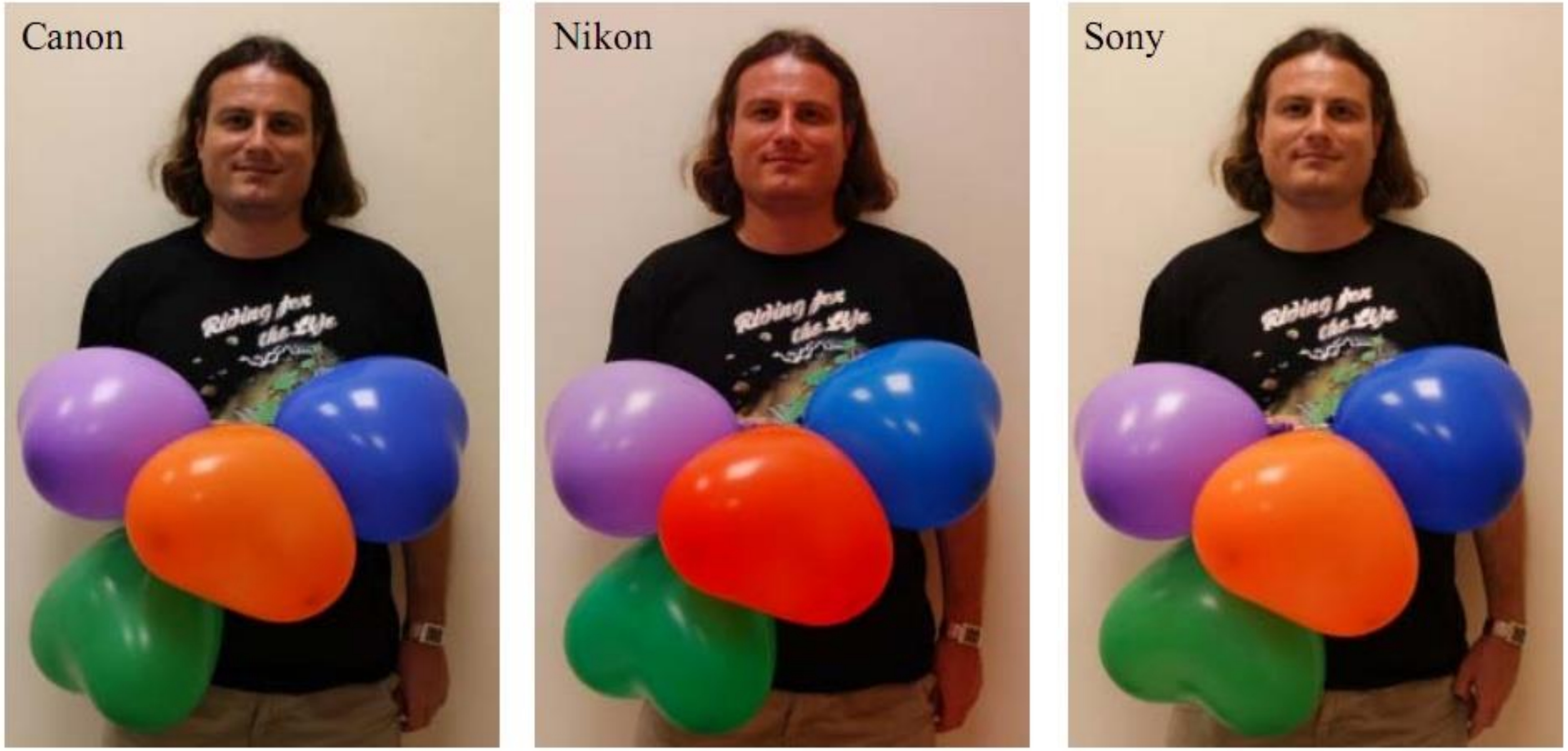
Sony Cyber-shot

How would you go about designing your own CFA? What criteria would you consider?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Many **Different Spectral Sensitivity** Functions

Each camera has its more or less unique, and most of the time secret, SSF

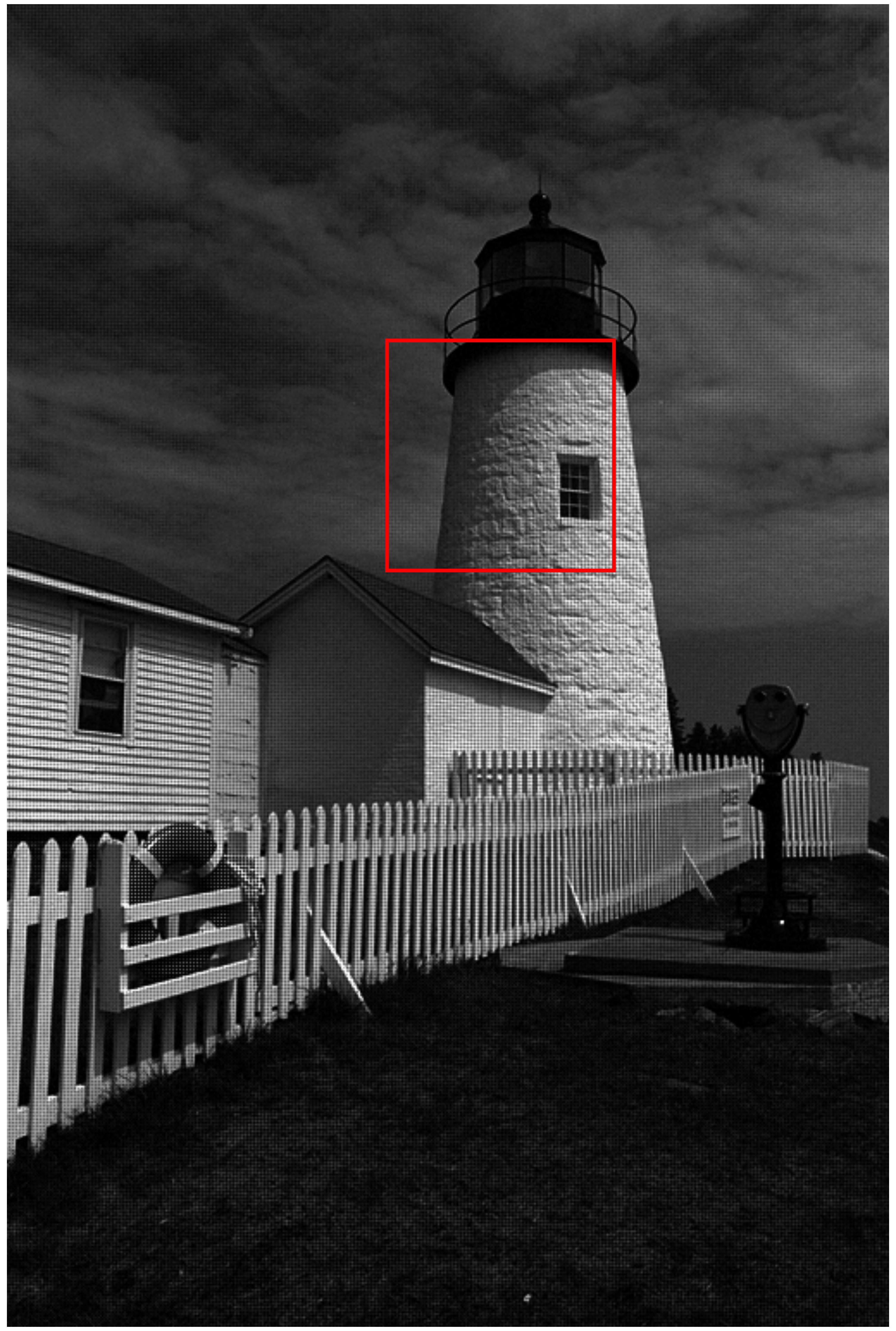


Same scene captured using 3 different cameras with identical settings

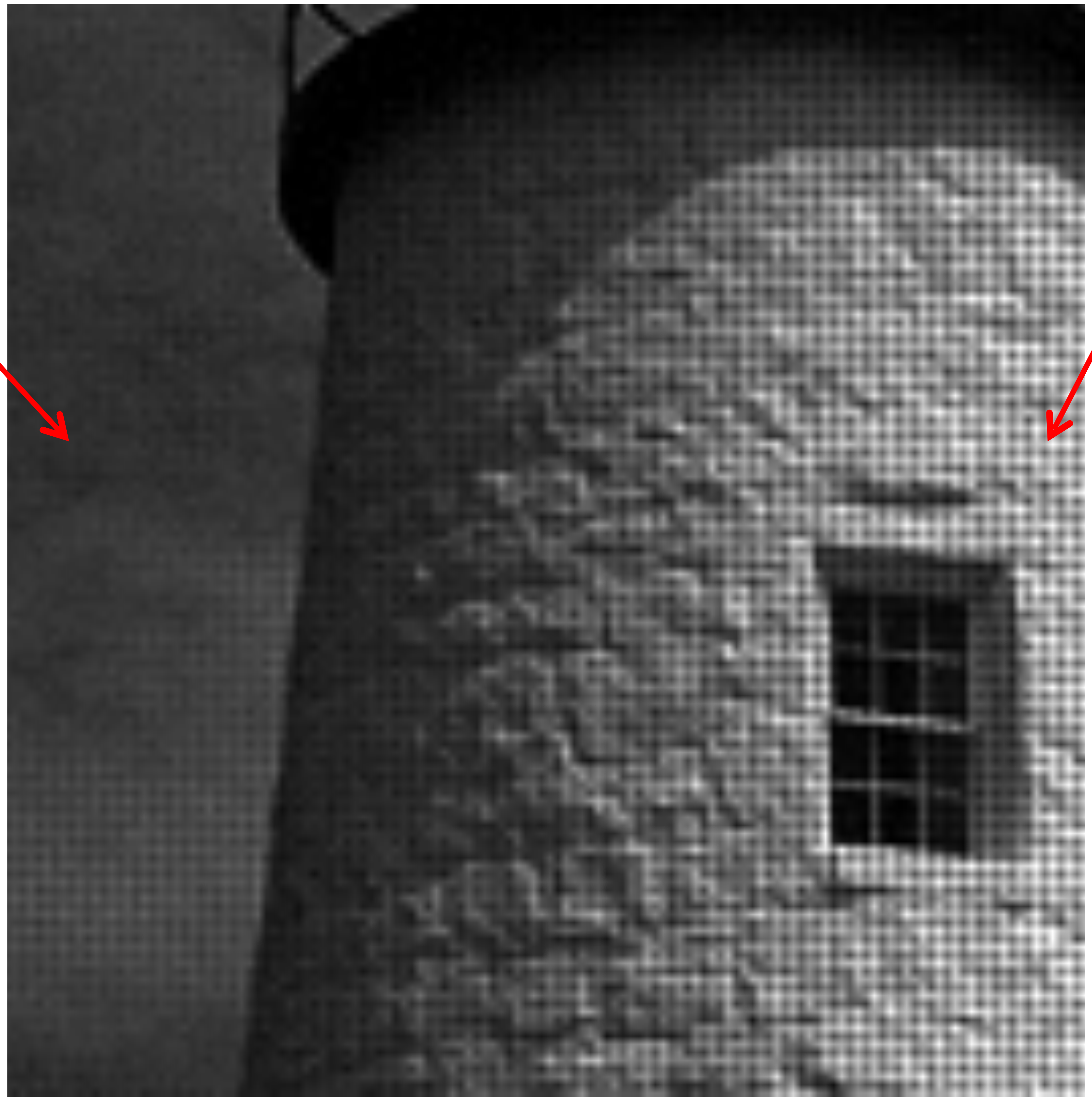
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

RAW Bayer Image

After all of this, what does an image look like?



lots of noise



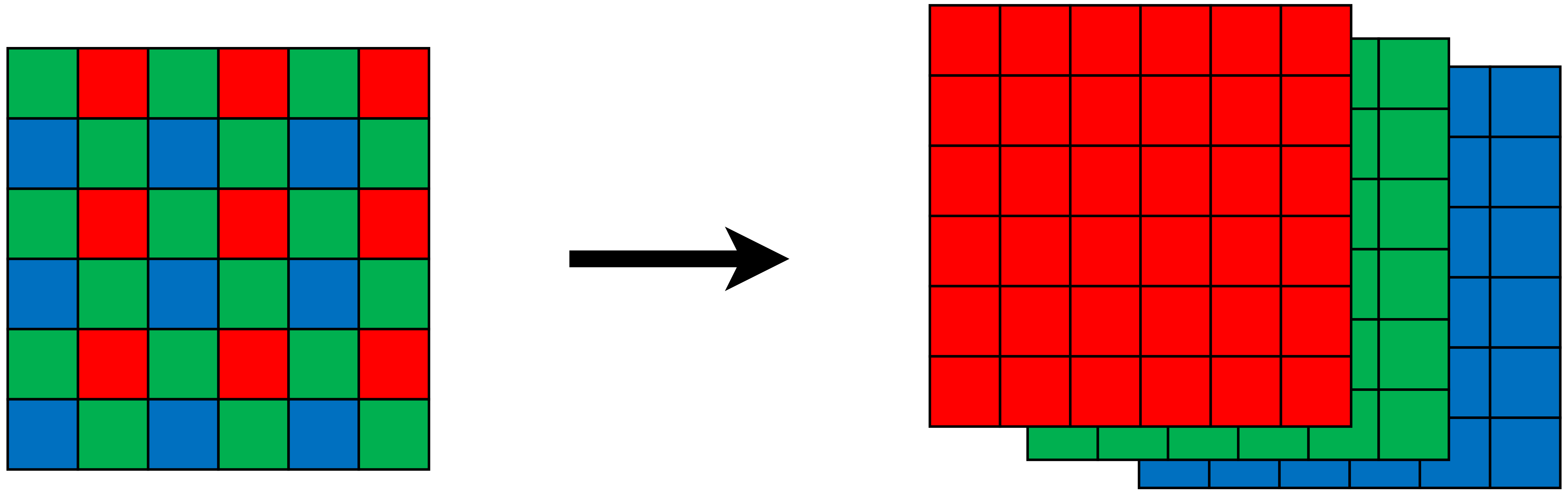
mosaicking artifacts

- Kind of disappointing
- We call this the RAW image

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

CFA **Demosicing**

Produce full RGB image from mosaiced sensor output

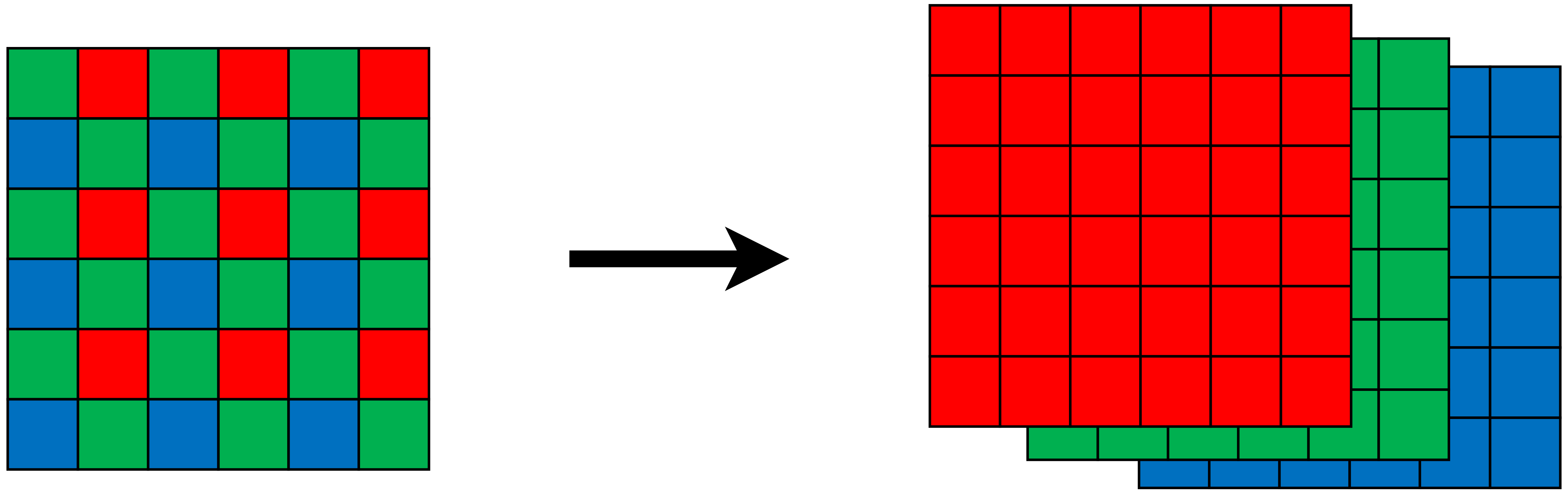


Any ideas on how to do this?

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

CFA **Demosicing**

Produce full RGB image from mosaiced sensor output



Interpolate from neighbors:

— Bilinear interpolation (needs 4 neighbors)

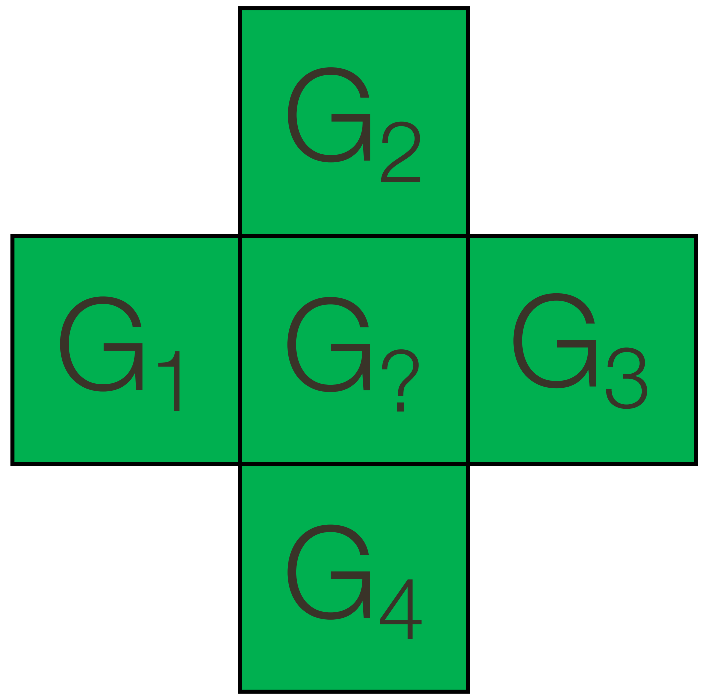
— Bicubic interpolation (needs more neighbors, may overblur)

— Edge-aware interpolation (e.g., Bilateral)

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

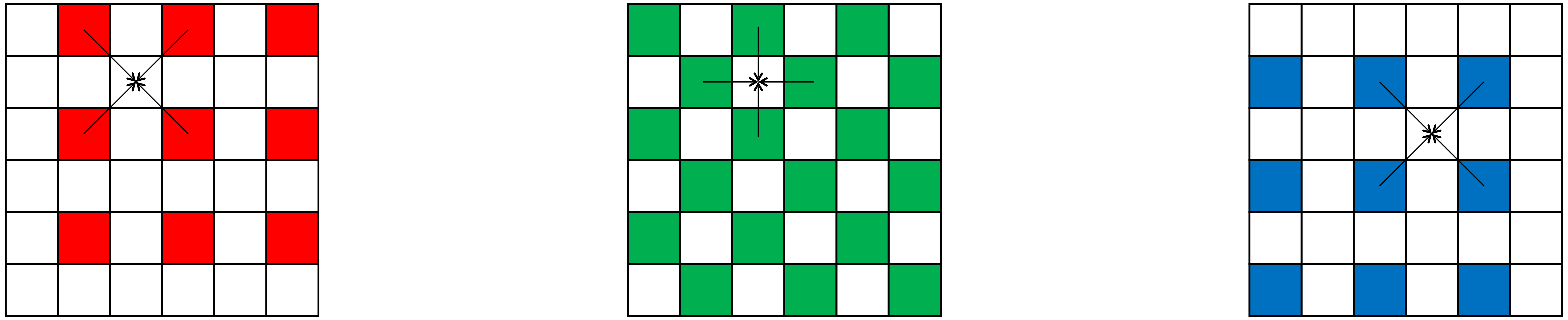
Demosaicing by Bilinear Interpolation

Bilinear interpolation: Simply average your 4 neighbors.



$$G_? = \frac{G_1 + G_2 + G_3 + G_4}{4}$$

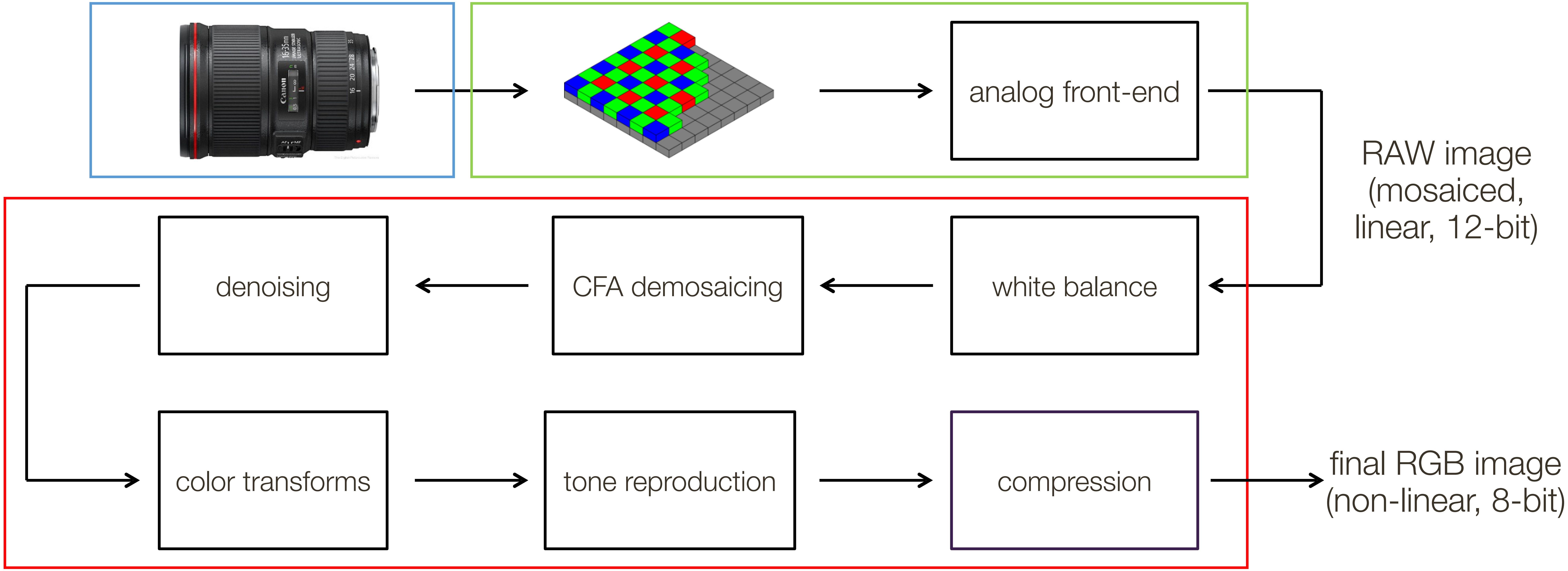
Neighborhood changes for different channels:



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

(in camera) **Image** Processing Pipeline

The sequence of image processing operations applied by the camera's image signal processor (ISP) to convert a RAW image into a "conventional" image.



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

(in camera) **White** balance

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

(in camera) **White** balance

R: 200
G: 255
B: 190

→

R-correction: + 55
G-correction: + 0
B-correction: + 65

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

(in camera) **White** balance

(in camera) **White** balance

- Humans are good at adapting to global illumination conditions: you would still describe a white object as white whether under blue sky or candle light.

(in camera) **White** balance

- Humans are good at adapting to global illumination conditions: you would still describe a white object as white whether under blue sky or candle light.
- However, when the picture is viewed later, the viewer is no longer correcting for the environment and the illuminant colour typically appears too strong.

(in camera) **White** balance

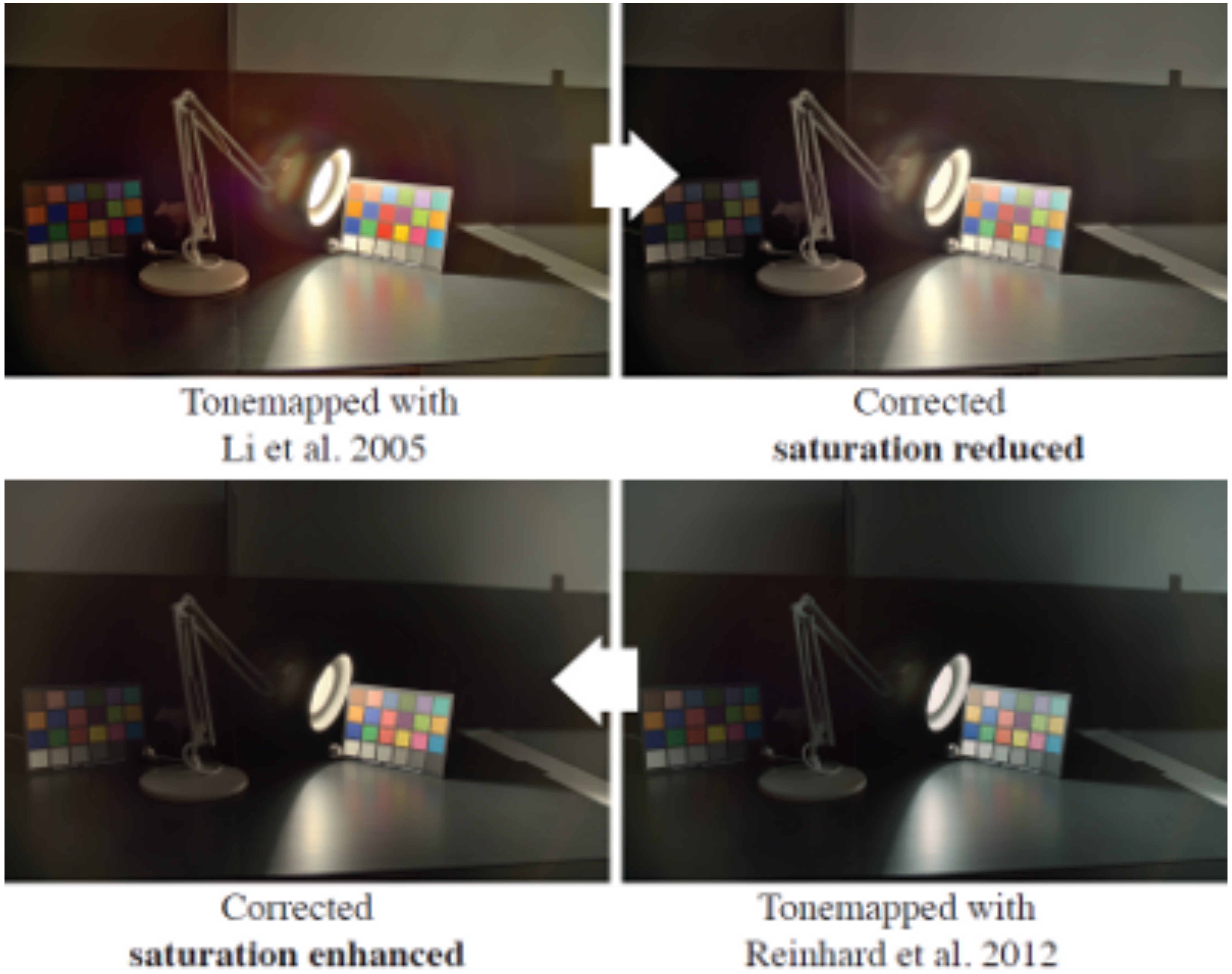
- Humans are good at adapting to global illumination conditions: you would still describe a white object as white whether under blue sky or candle light.
- However, when the picture is viewed later, the viewer is no longer correcting for the environment and the illuminant colour typically appears too strong.
- **White balancing** is the process of correcting for the illuminant

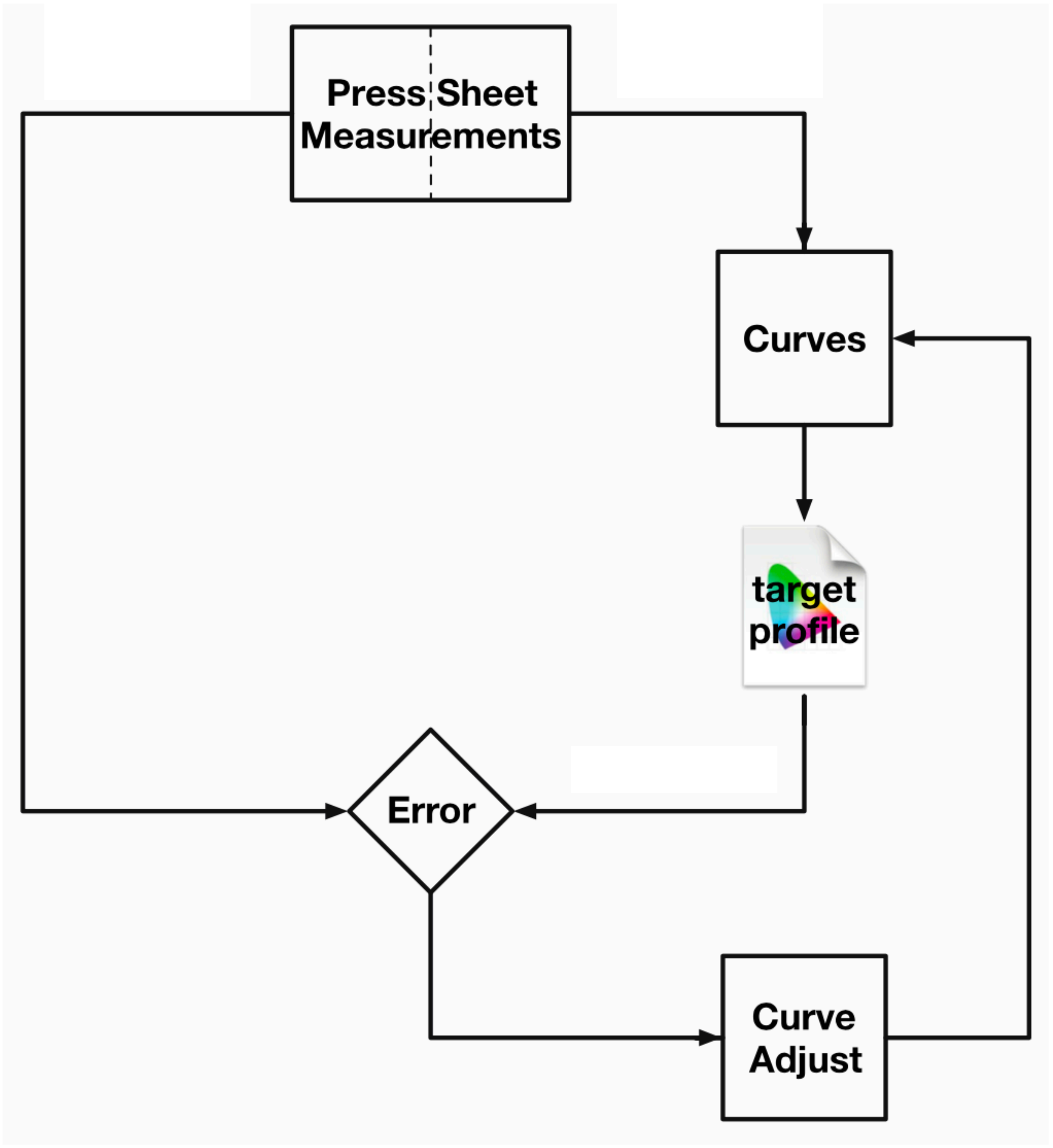
(in camera) **White** balance

- Humans are good at adapting to global illumination conditions: you would still describe a white object as white whether under blue sky or candle light.
- However, when the picture is viewed later, the viewer is no longer correcting for the environment and the illuminant colour typically appears too strong.
- **White balancing** is the process of correcting for the illuminant

- A simple white balance algorithm is to assume the scene is grey on average "greyworld", state of the art methods use learning, e.g., Barron ICCV 2015

(in camera) **Tone** reproduction





Summary

"Color" is **not** an objective physical property of light (electromagnetic radiation). Instead, light is characterized by its wavelength.

Color Filter Arrays (CFAs) allow capturing of mosaiced color information; the layout of the mosaic is called **Bayer** pattern.

Demosaicing is the process of taking the RAW image and interpolating missing color pixels per channel