**Lecture 2:**

The Camera Image Processing Pipeline

Visual Computing Systems
Stanford CS348K, Spring 2022

Theme of the next two lectures...

The pixels you see on screen are quite different than the values recorded by the sensor in a modern digital camera.
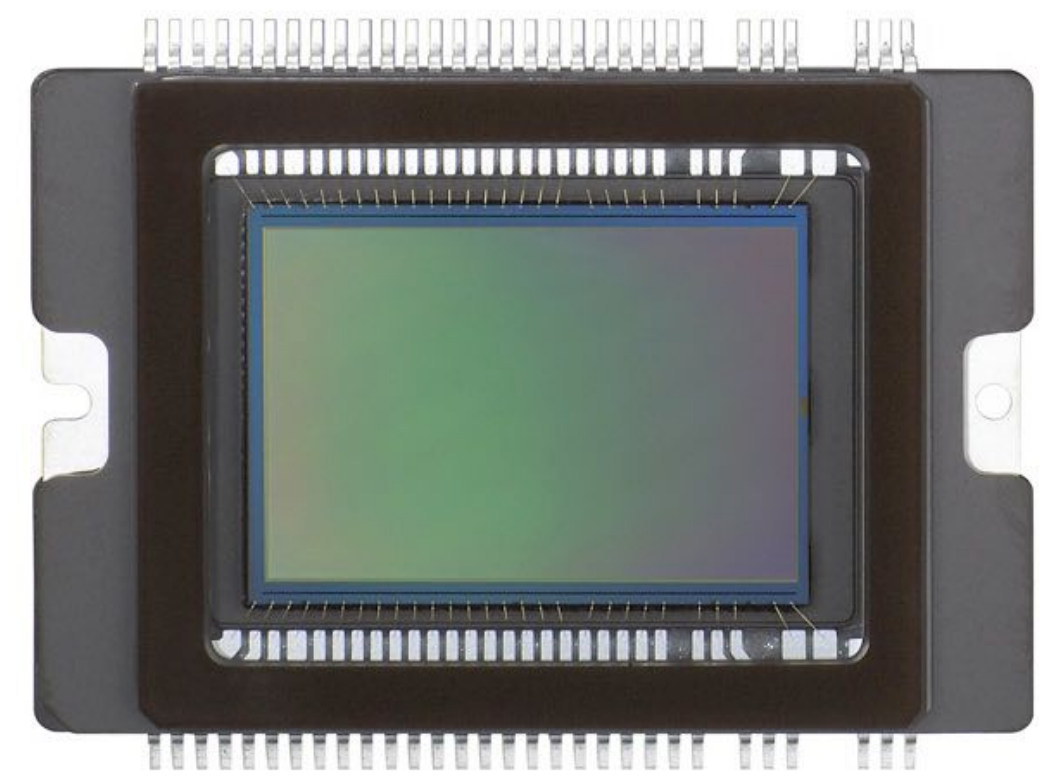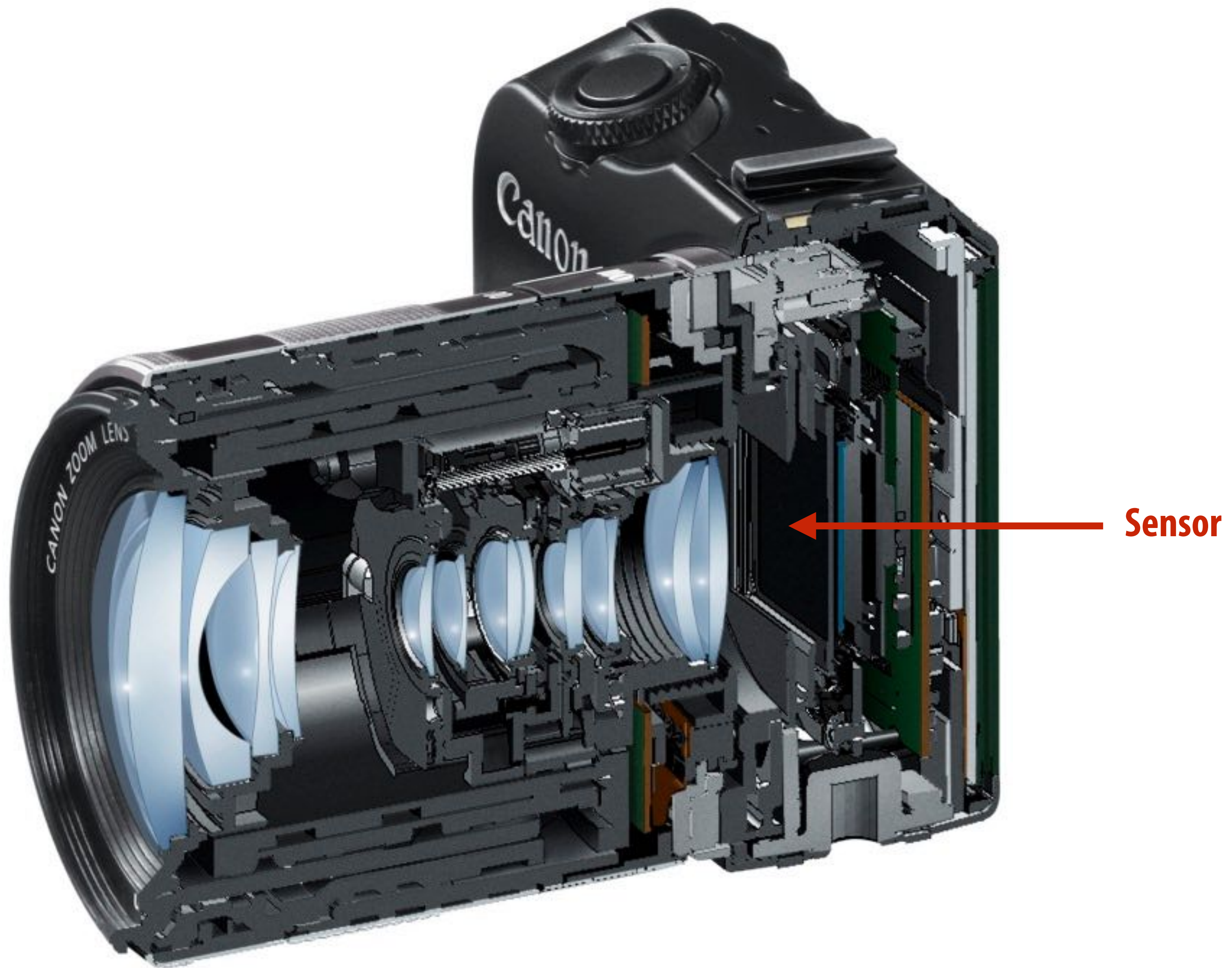
Computation is a fundamental aspect of producing high-quality pictures.

Sensor output ("RAW") → Computation → Beautiful image that impresses your Instagram friends
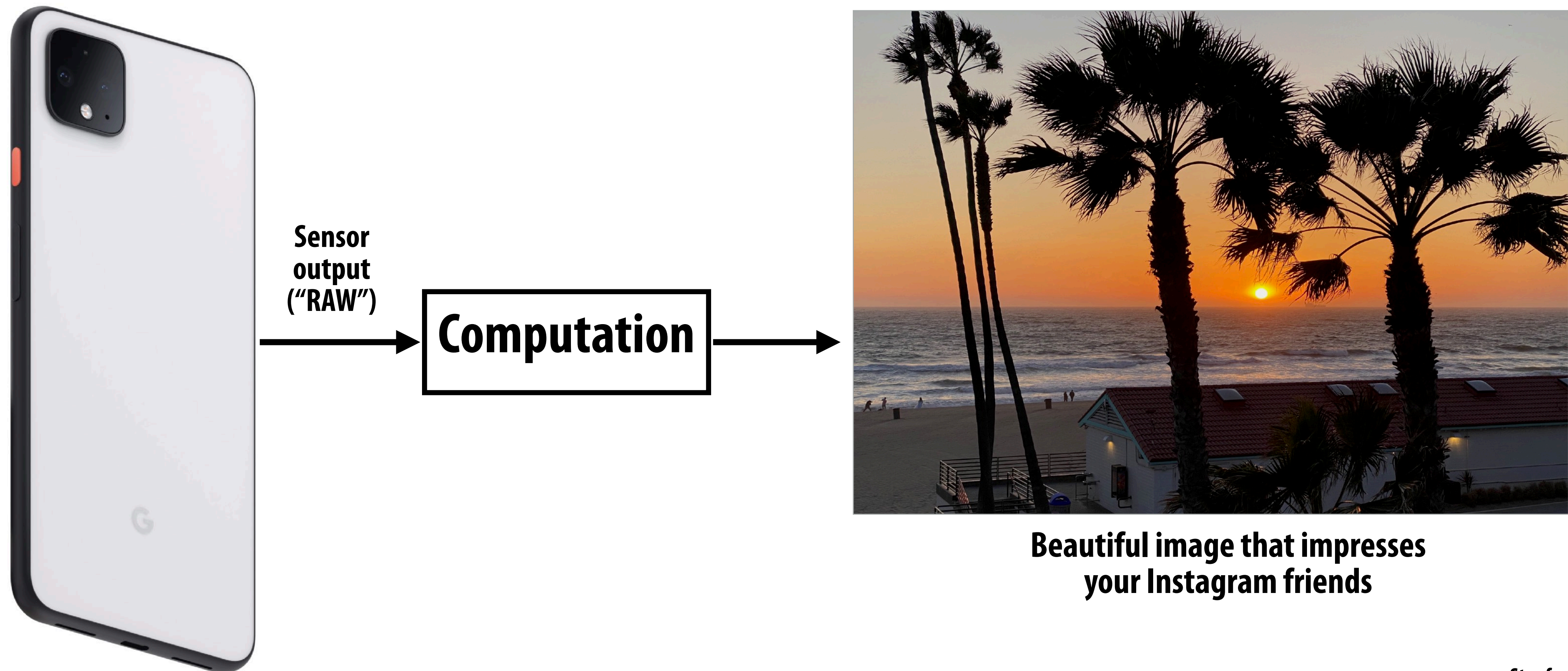
Part 1: image sensing hardware

(how a digital camera measures light, and how physical limitations of these devices place challenges on software)

Camera cross section

Sensor

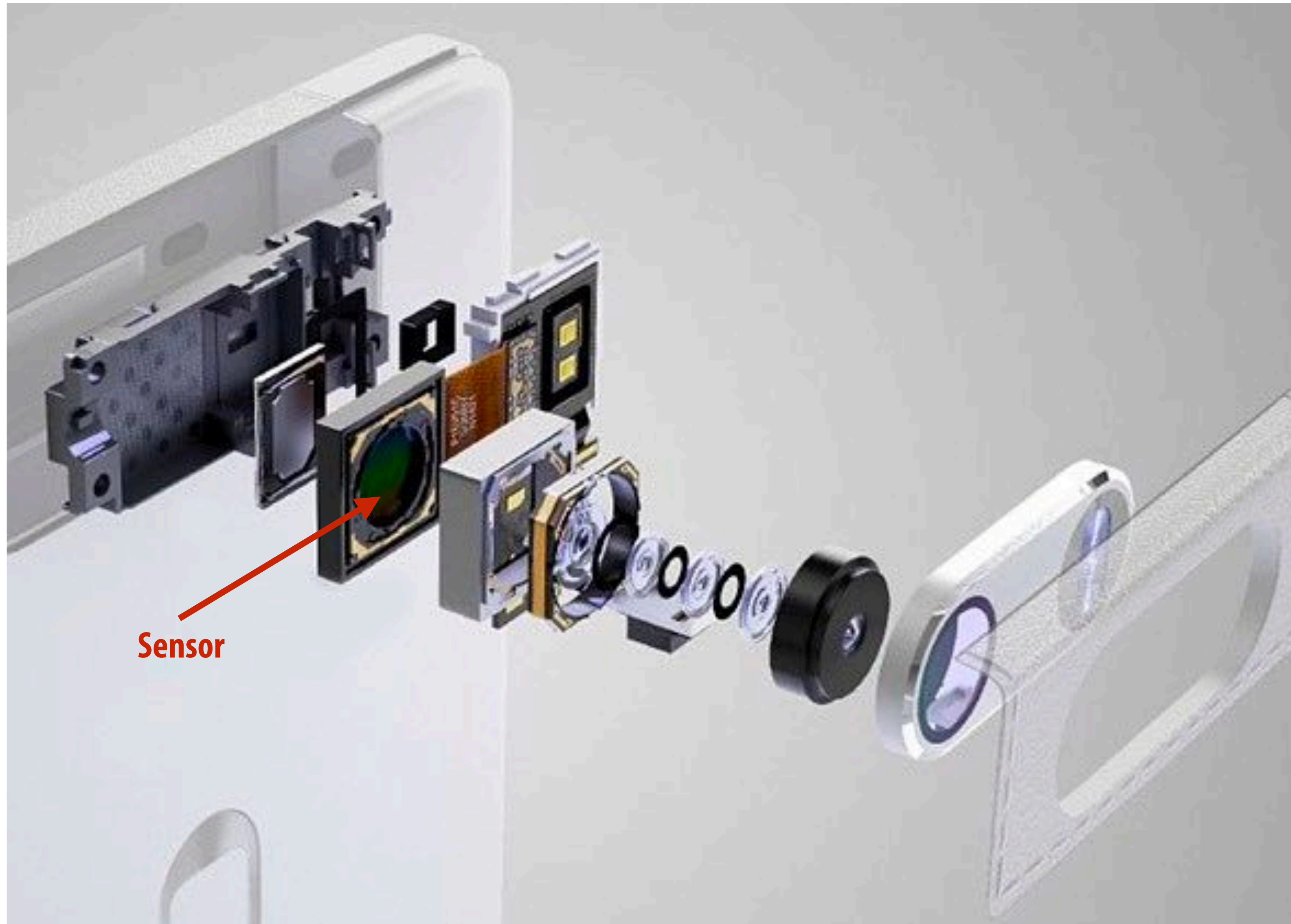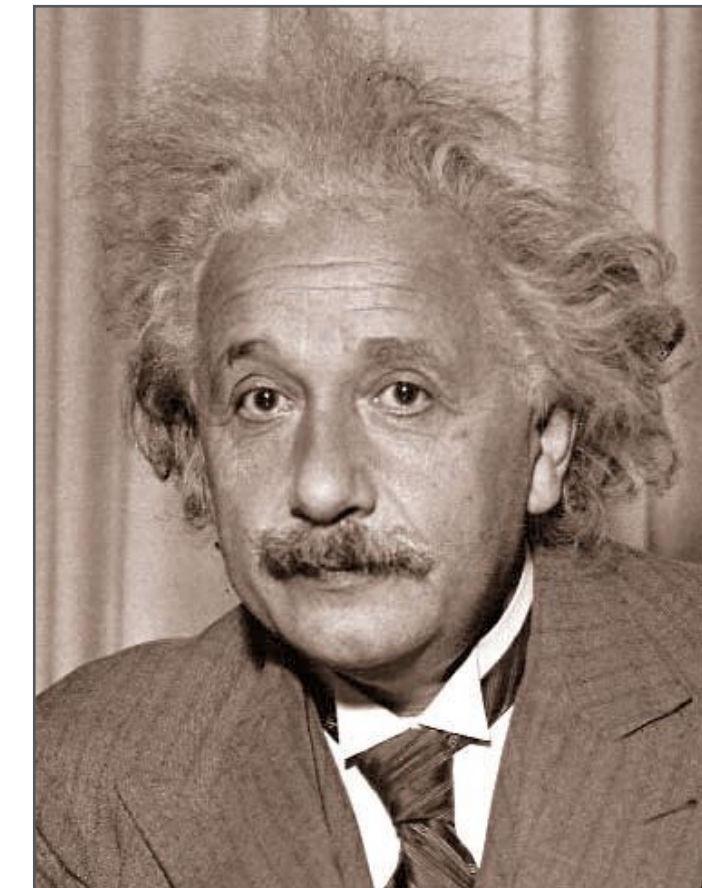**Canon 14 MP CMOS Sensor**
(14 bits per pixel)

Camera cross section

Sensor

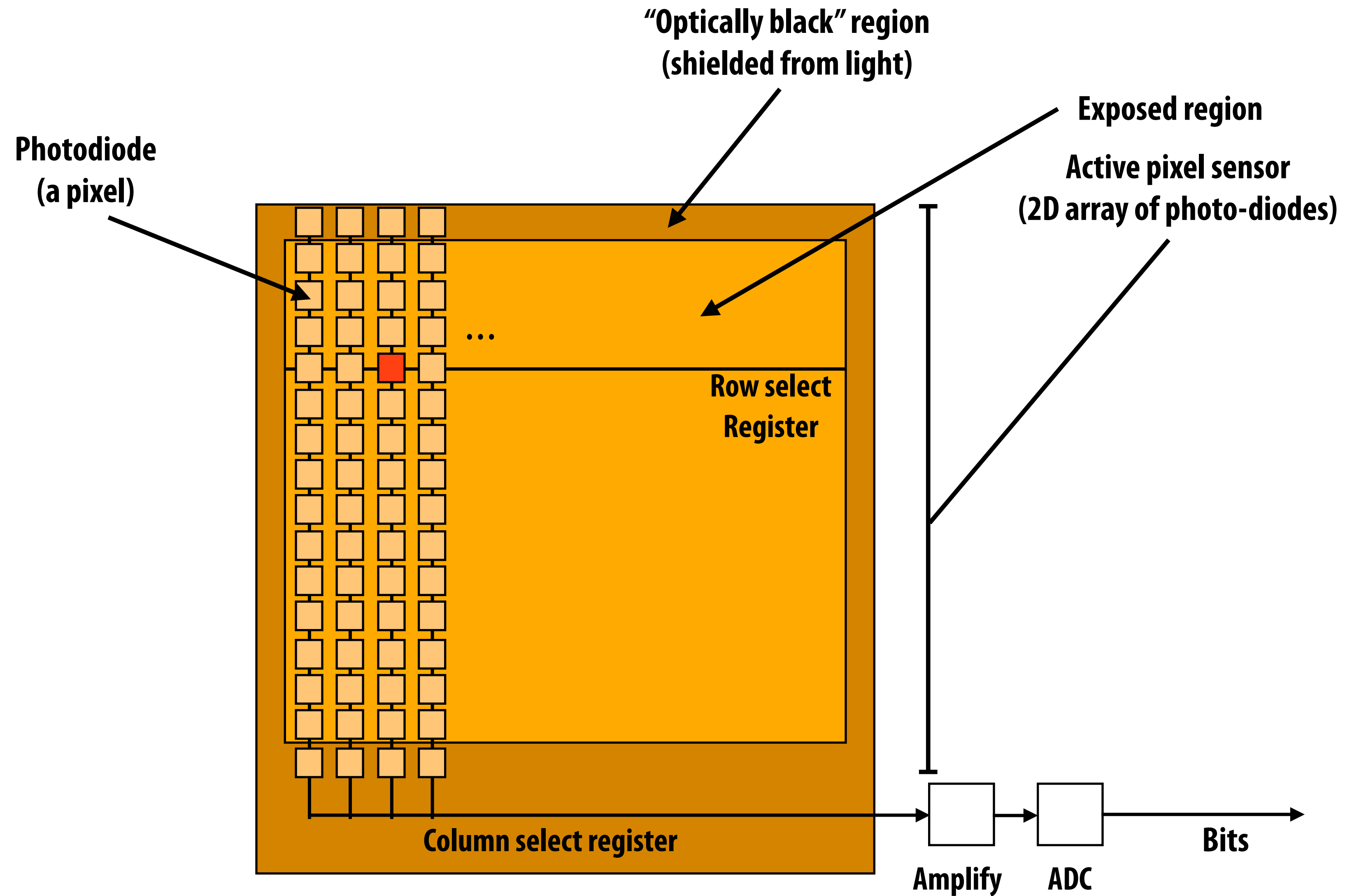# The Sensor

Photoelectric effect

Einstein's Nobel Prize in 1921 "for his services to Theoretical Physics, and especially for his discovery of the law of the photoelectric effect"

Incident photons

⊖ ⊖ **Ejected electrons**
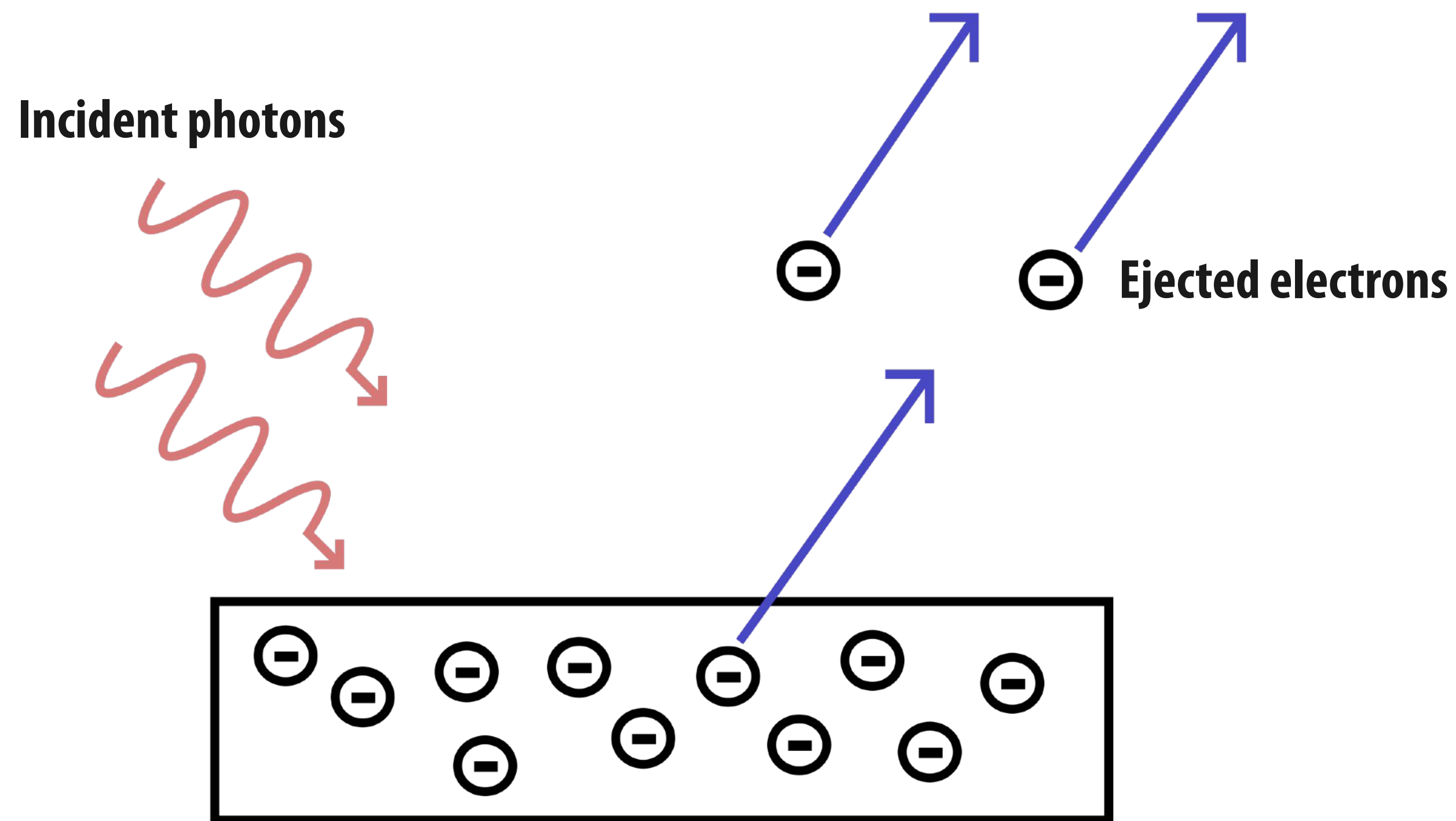
Albert Einstein

CMOS sensor

CMOS = complementary metal-oxide semiconductor

CMOS APS (active pixel sensor) pixel

Figure 3

Labels: Microlens, Red Color Filter, Amplifier Transistor, Reset Transistor, Row Select Bus, Column Bus Transistor, Photodiode, n+, Silicon Substrate, Potential Well
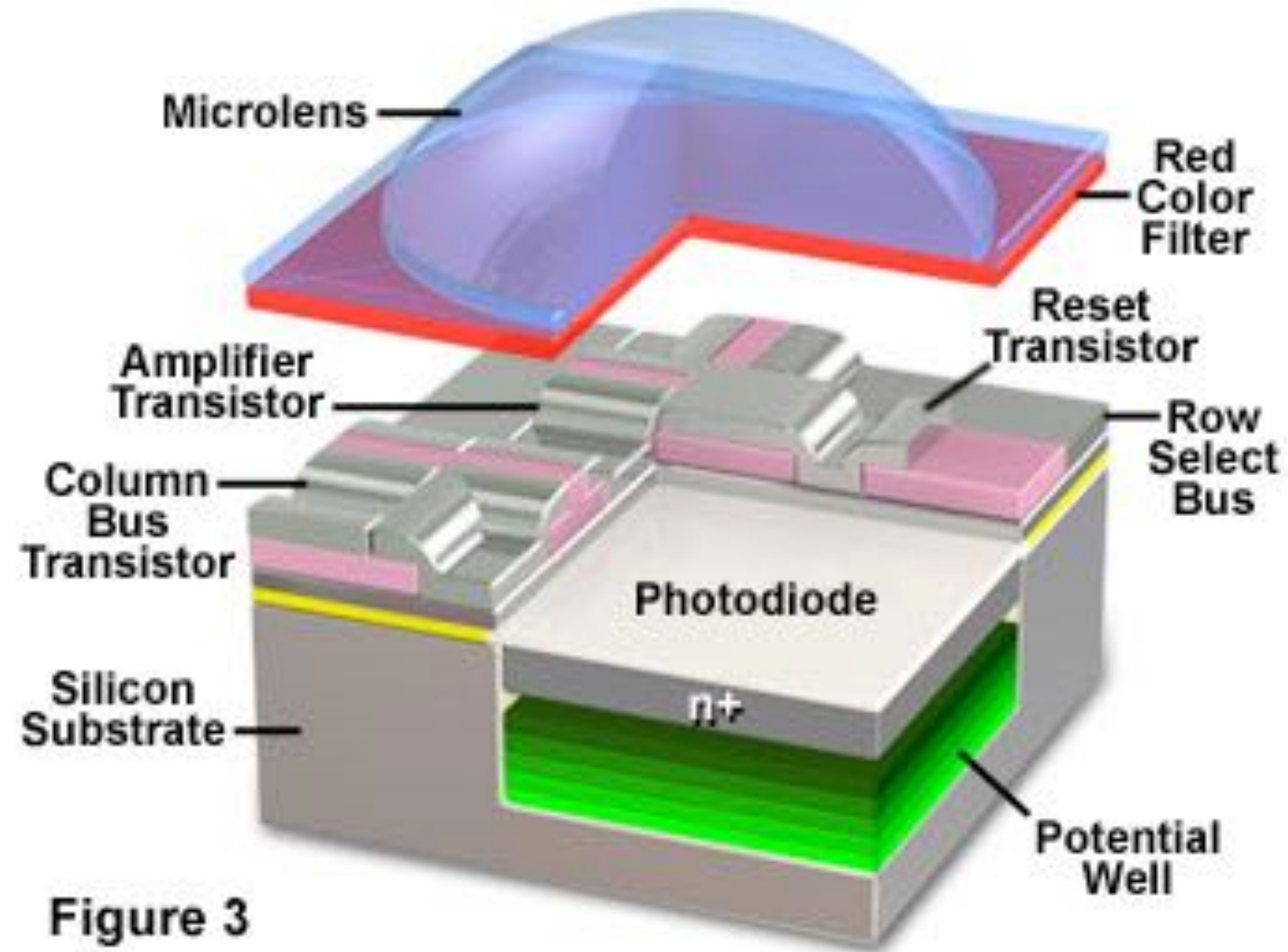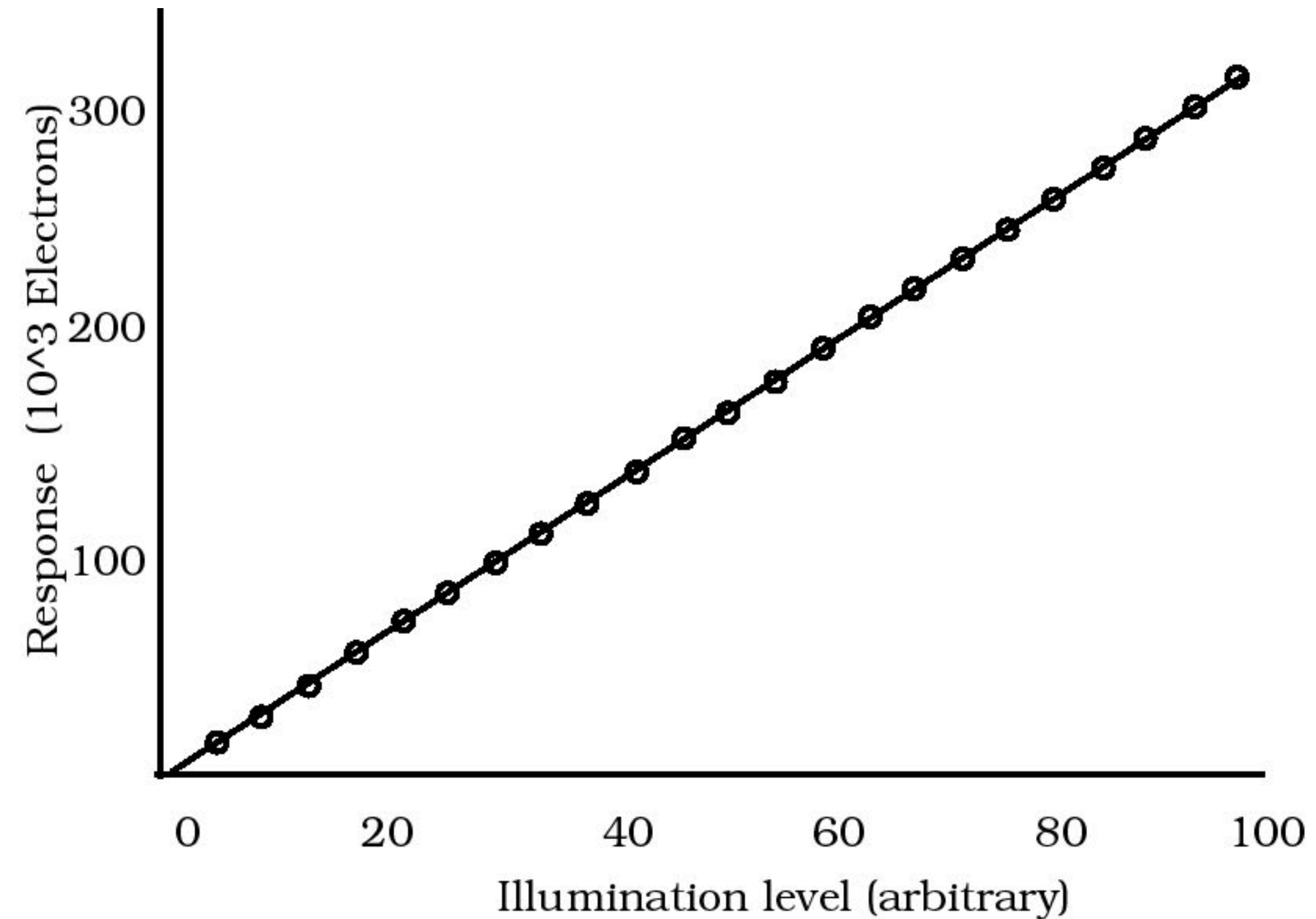
CMOS response functions are linear

Photoelectric effect in silicon:

- **Response function from photons to electrons is linear**

  (Some nonlinearity close to 0 due to noise and when close to pixel saturation)

(Epperson, P.M. et al. Electro-optical characterization of the Tektronix TK5 ..., Opt Eng., 25, 1987)

Quantum efficiency

■ **Not all photons will produce an electron (depends on quantum efficiency of the device)**

$$QE = \frac{\# \, electrons}{\# \, photons}$$

- **Human vision:** ~15%

- **Typical digital camera:** < 50%

- **Best back-thinned CCD:** > 90%
  (e.g., telescope)

Sensing Color

Electromagnetic spectrum

Describes distribution of power (energy/time) by wavelength

Below: spectrum of various common light sources:

Figure credit:
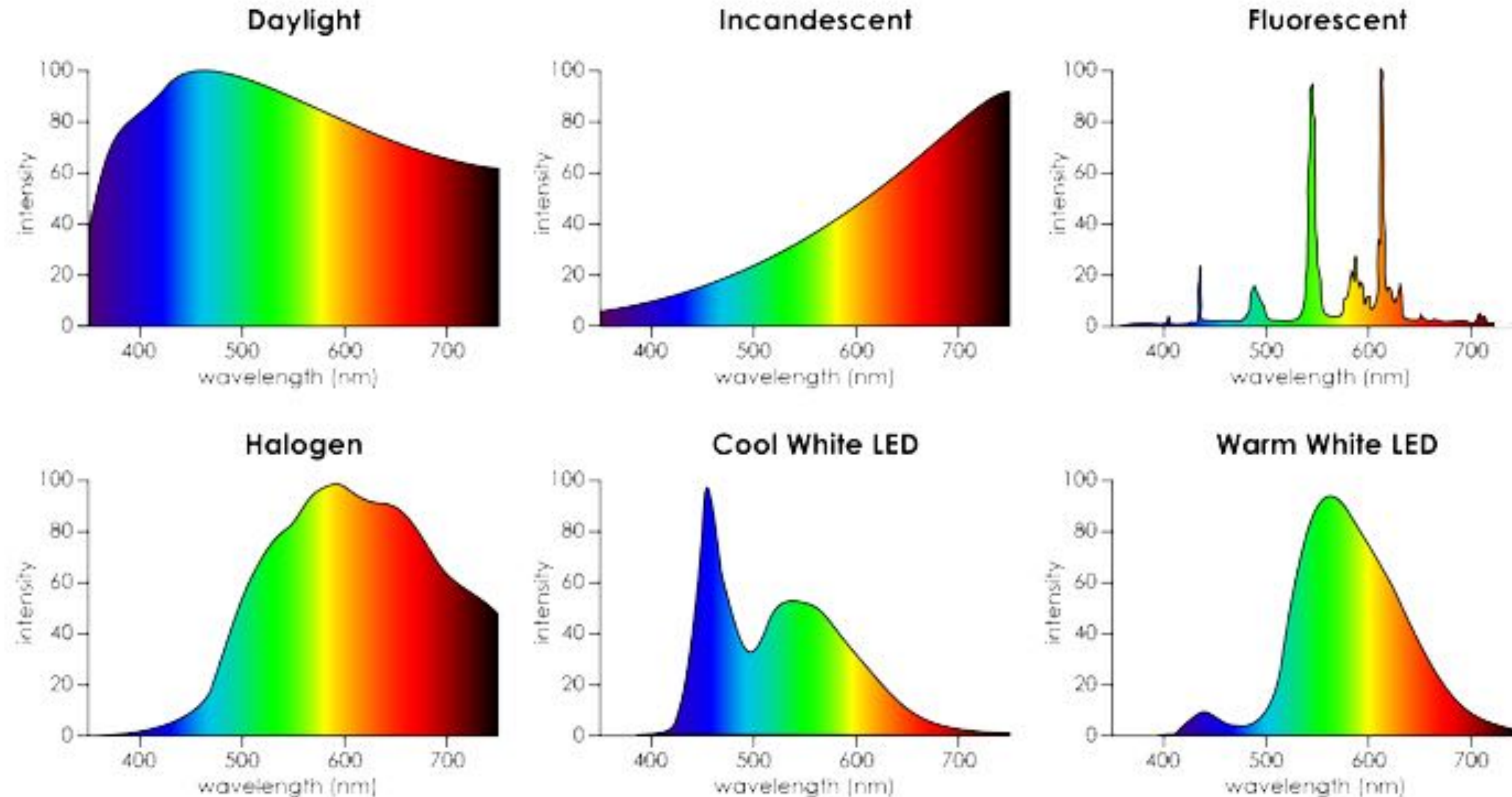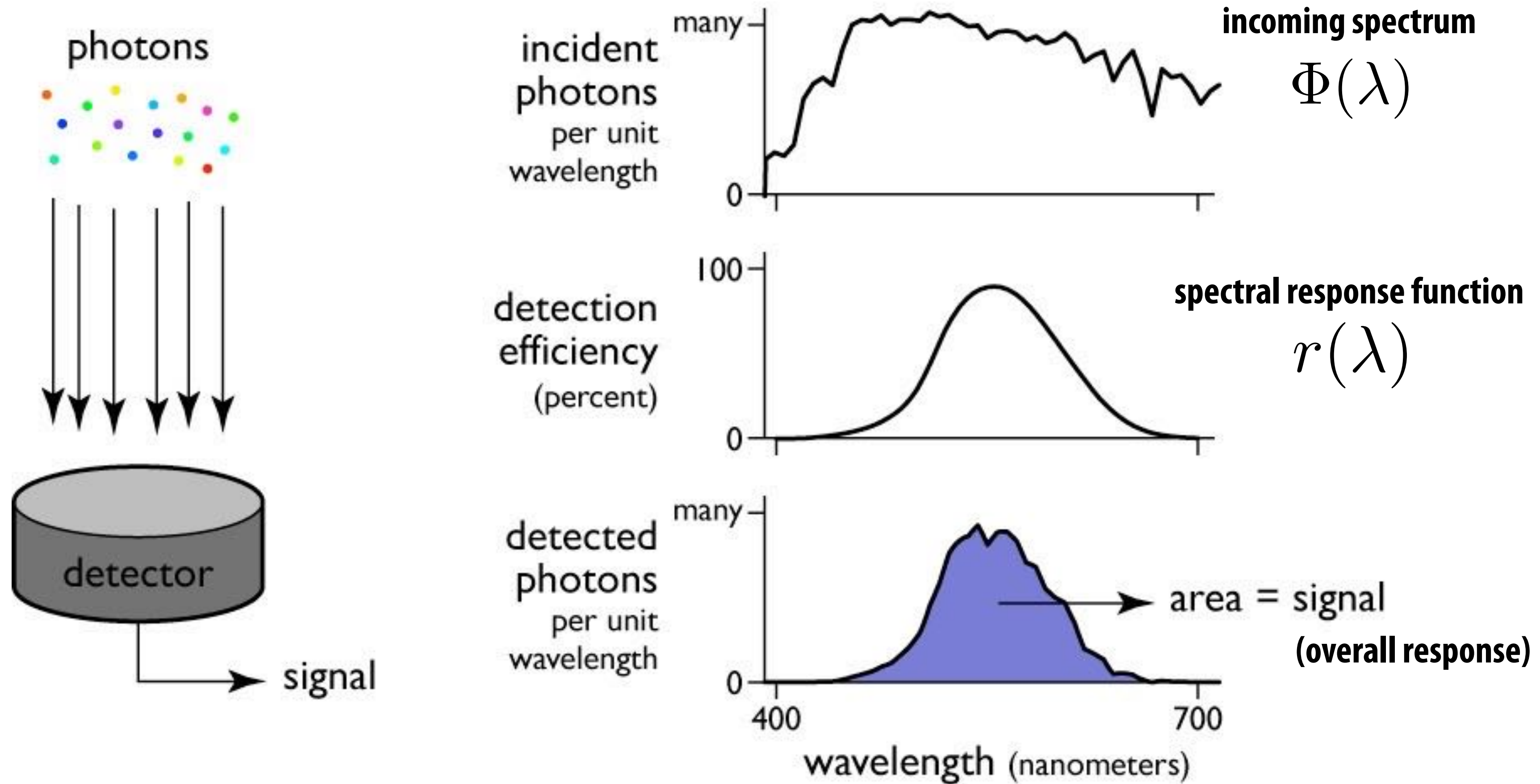
Example: warm white vs. cool white

Simple model of a light detector

incoming spectrum
$$\Phi(\lambda)$$

spectral response function
$$r(\lambda)$$
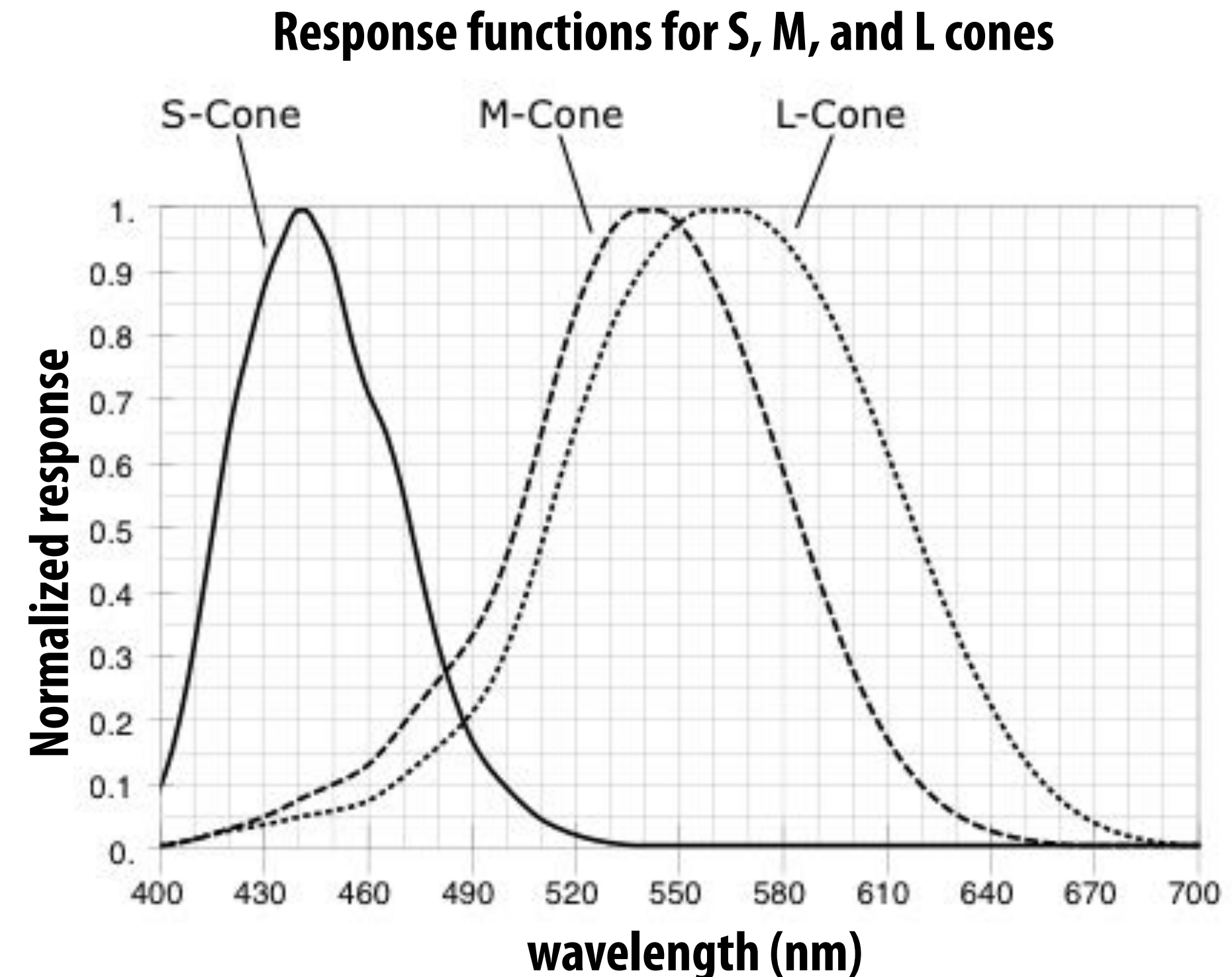
area = signal
(overall response)

$$R = \int_{\lambda} \Phi(\lambda) r(\lambda) d\lambda$$

Spectral response of cone cells in human eye

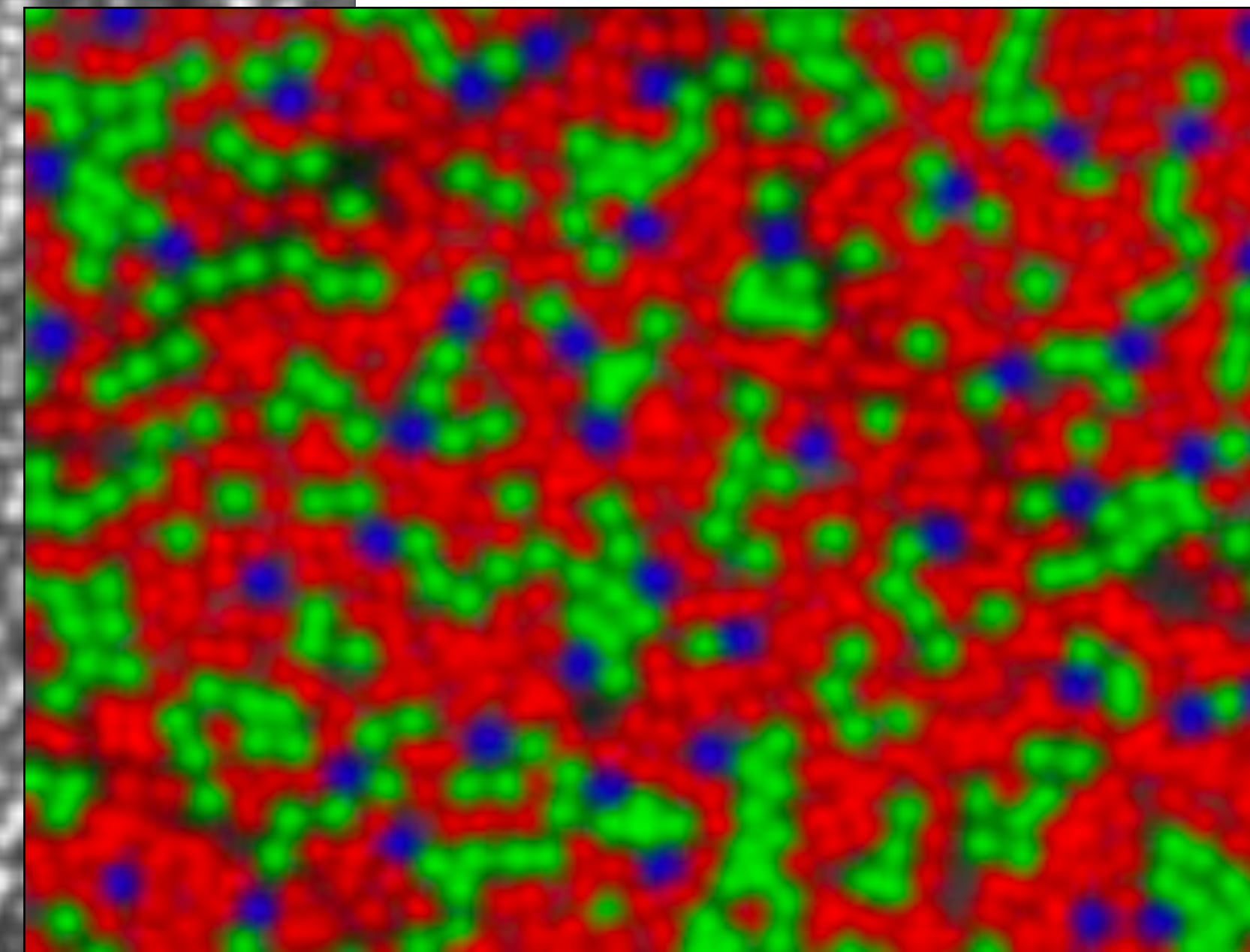Three types of cells in eye responsible for color perception: S, M, and L cones (corresponding to peak response at short, medium, and long wavelengths)

Implication: the space of human-perceivable colors is three dimensional

$$S = \int_{\lambda} \Phi(\lambda)S(\lambda)d\lambda$$

$$M = \int_{\lambda} \Phi(\lambda)M(\lambda)d\lambda$$

$$L = \int_{\lambda} \Phi(\lambda)L(\lambda)d\lambda$$

Response functions for S, M, and L cones

Human eye cone cell mosaic
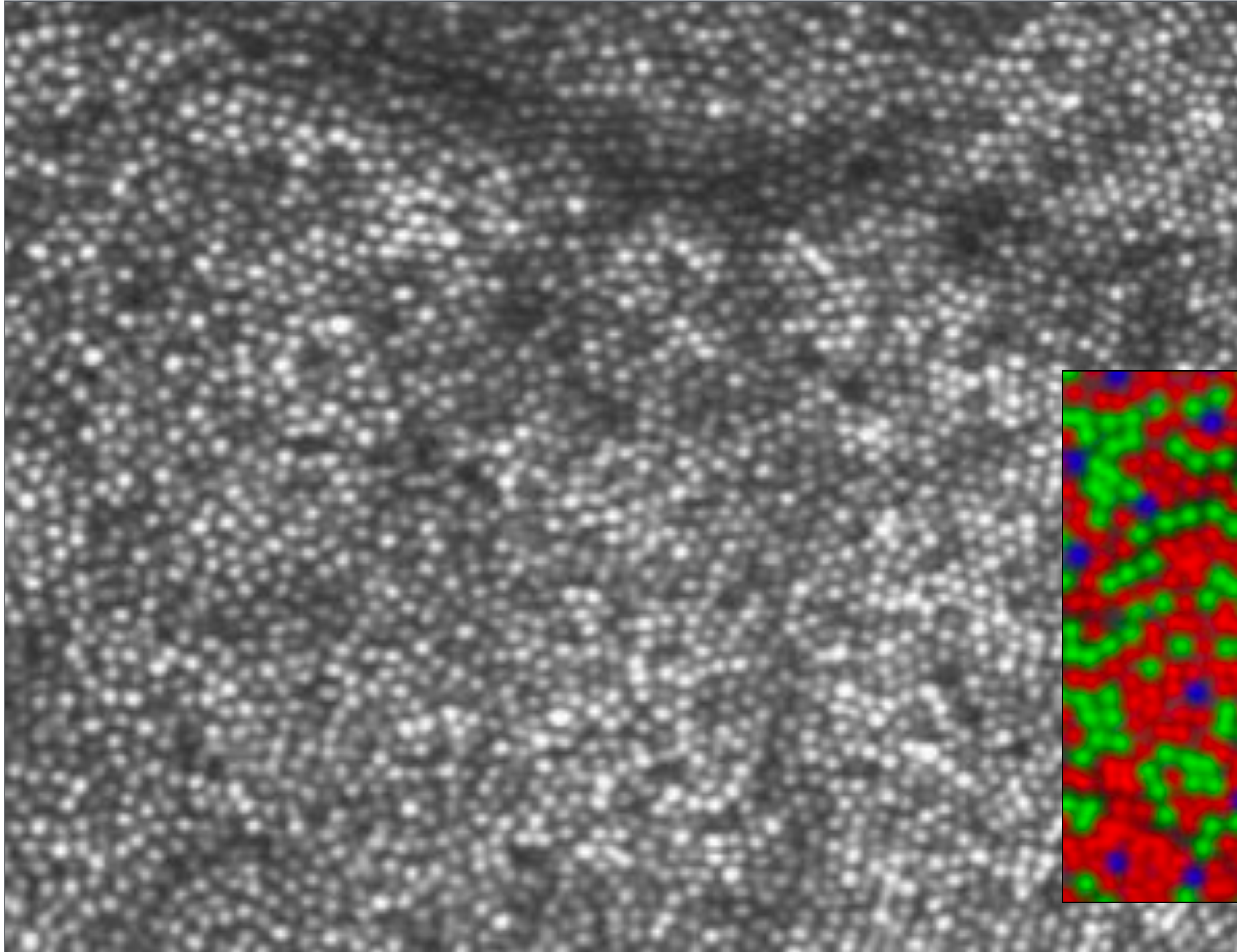
False color image:
red = L cones
green = M cones
blue = R cones

Image Credit: Ramkumar Sabesan Lab

Color filter array (Bayer mosaic)

- **Color filter array placed over sensor**

- **Result: different pixels have different spectral response (each pixel measures red, green, or blue light)**

- **50% of pixels are green pixels**

Pixel response curve: Canon 40D/50D
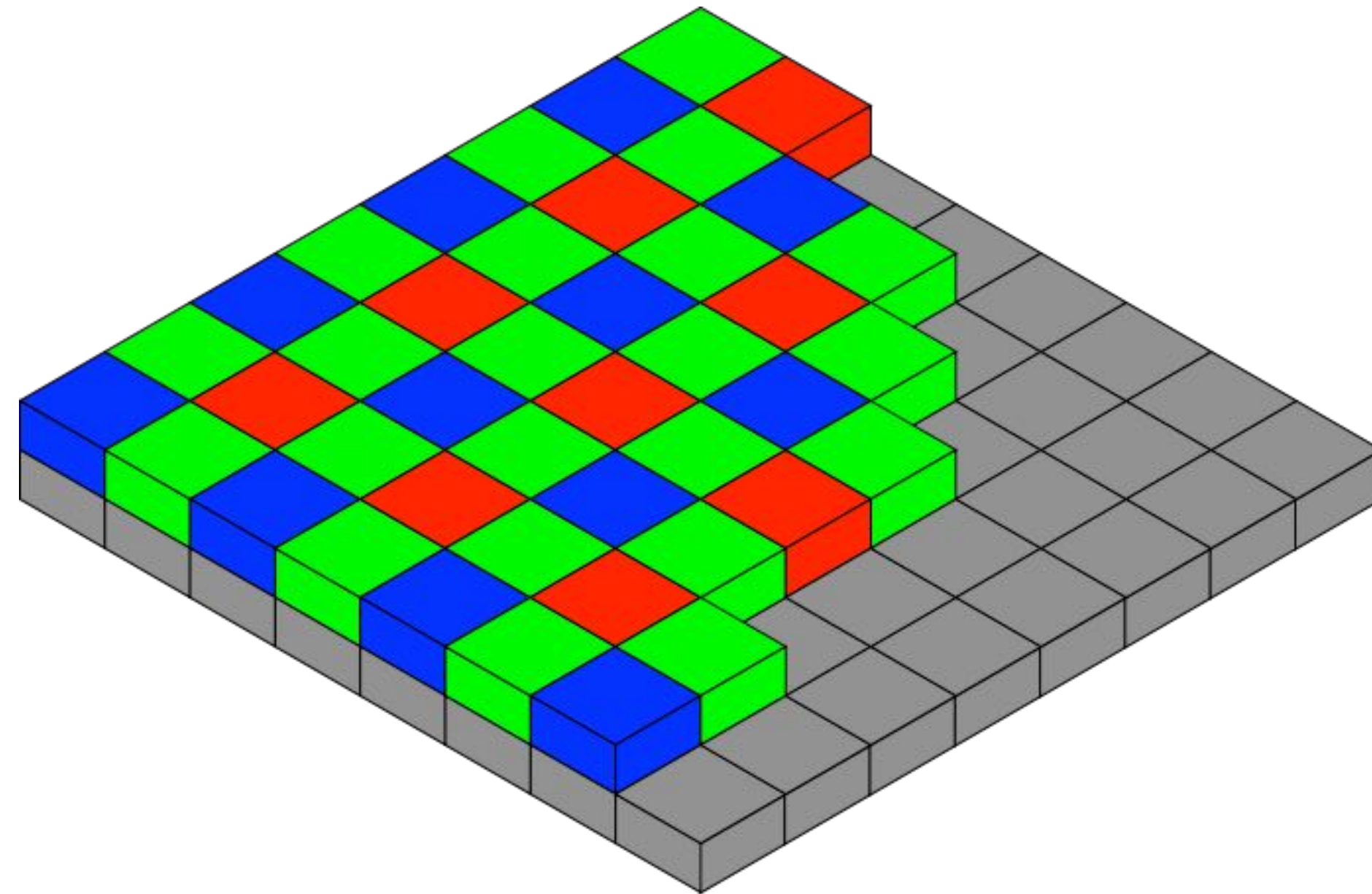
Traditional Bayer mosaic
(other filter patterns exist: e.g., Sony's RGBE)
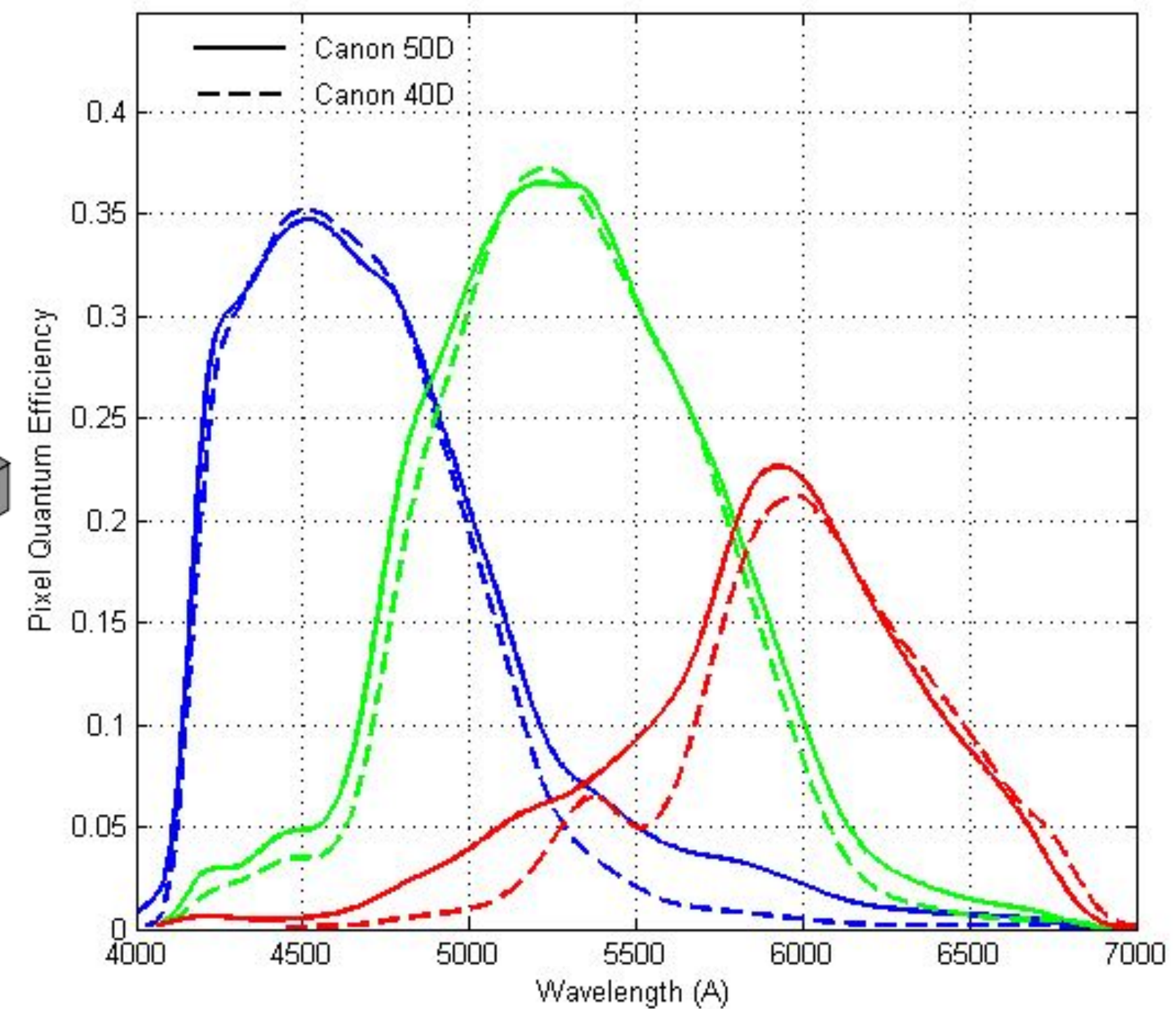
$f(\lambda)$

Light incident on camera

What sensor measures

What sensor measures
(zoomed view)

Defective pixel

CMOS Pixel Structure

Front-side-illuminated (FSI) CMOS

Building up the CMOS imager layers

Photodiodes
~50% Fill Factor

Pixel pitch:
A few microns
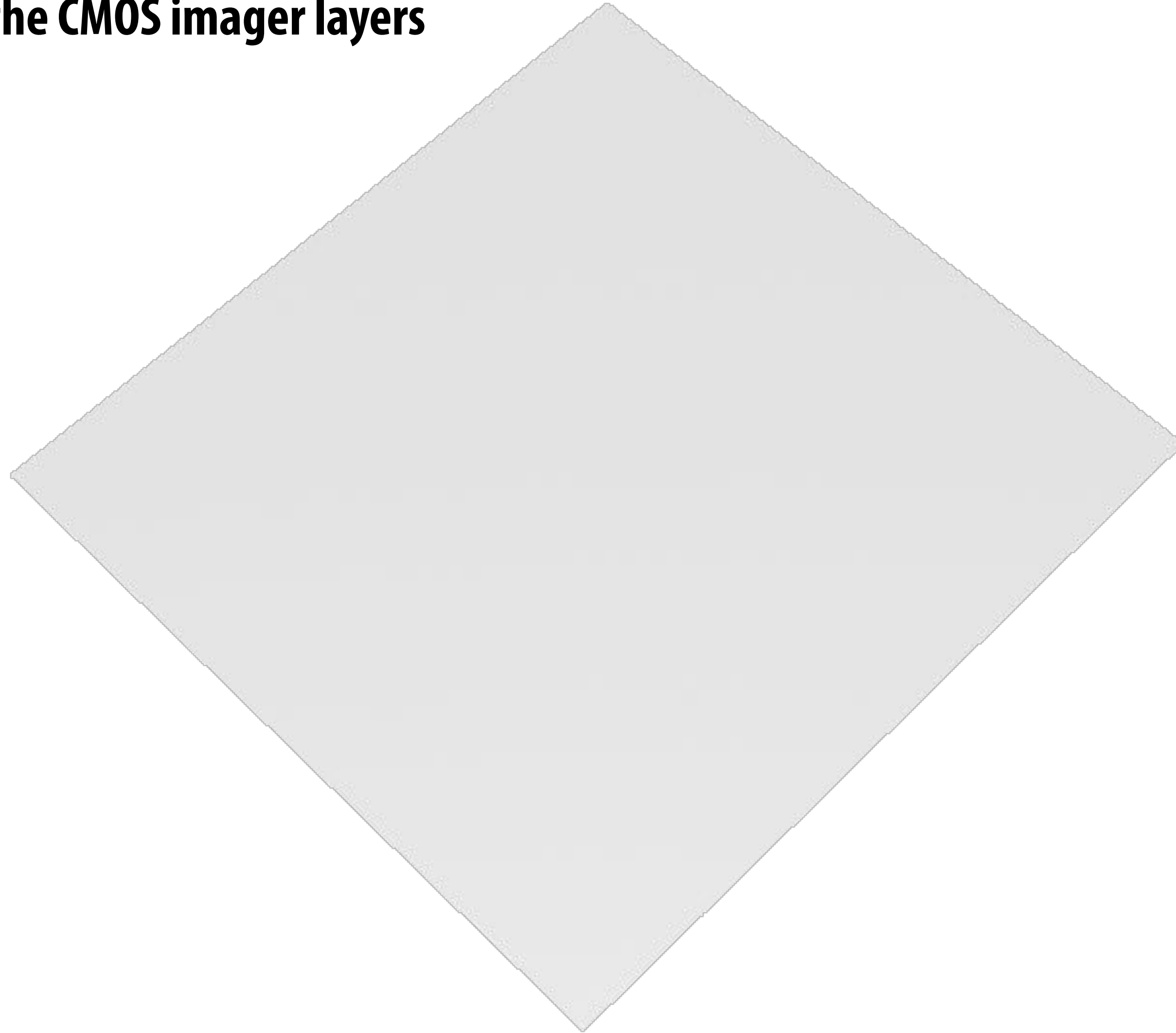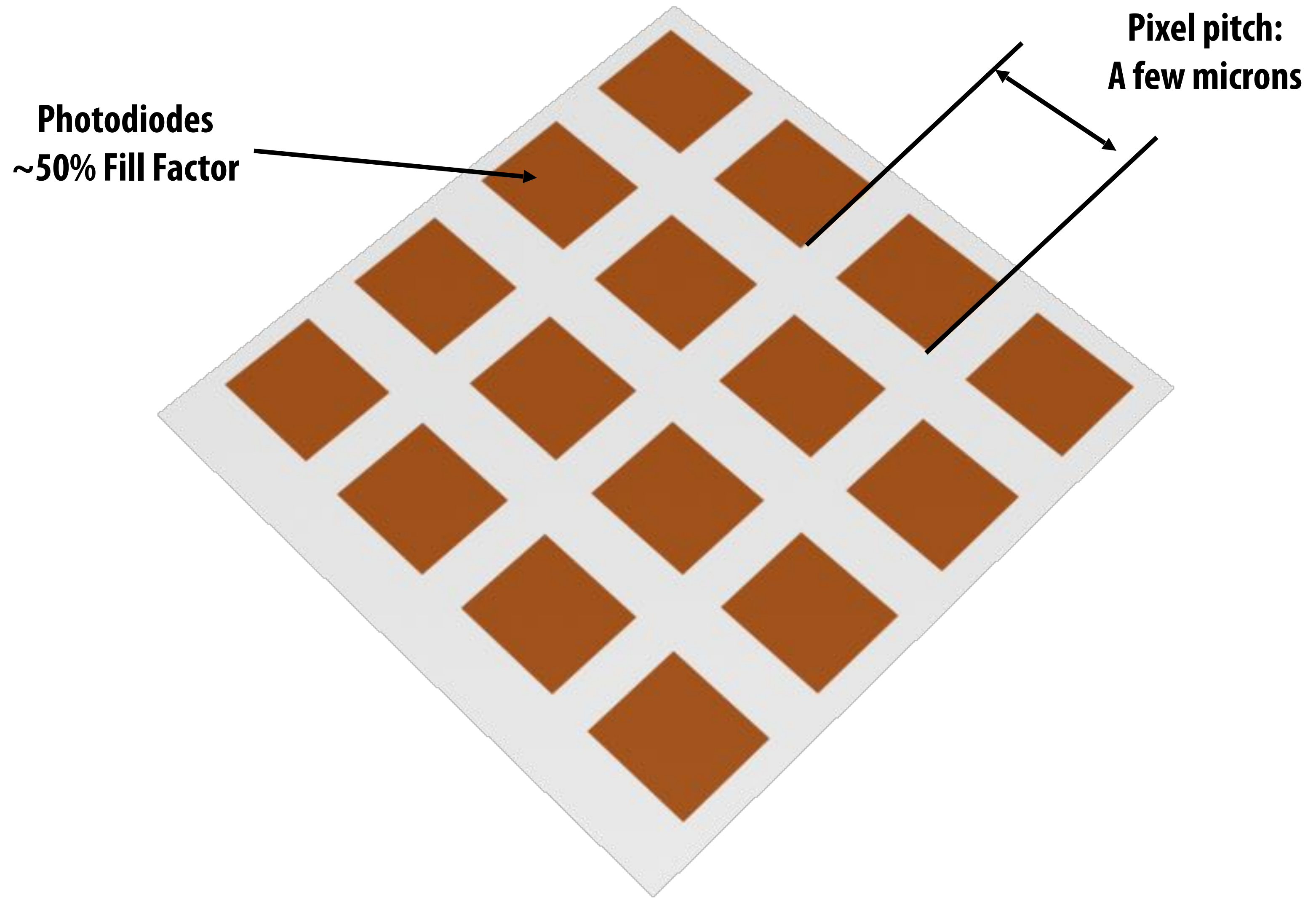
**Polysilicon
& Via 1**
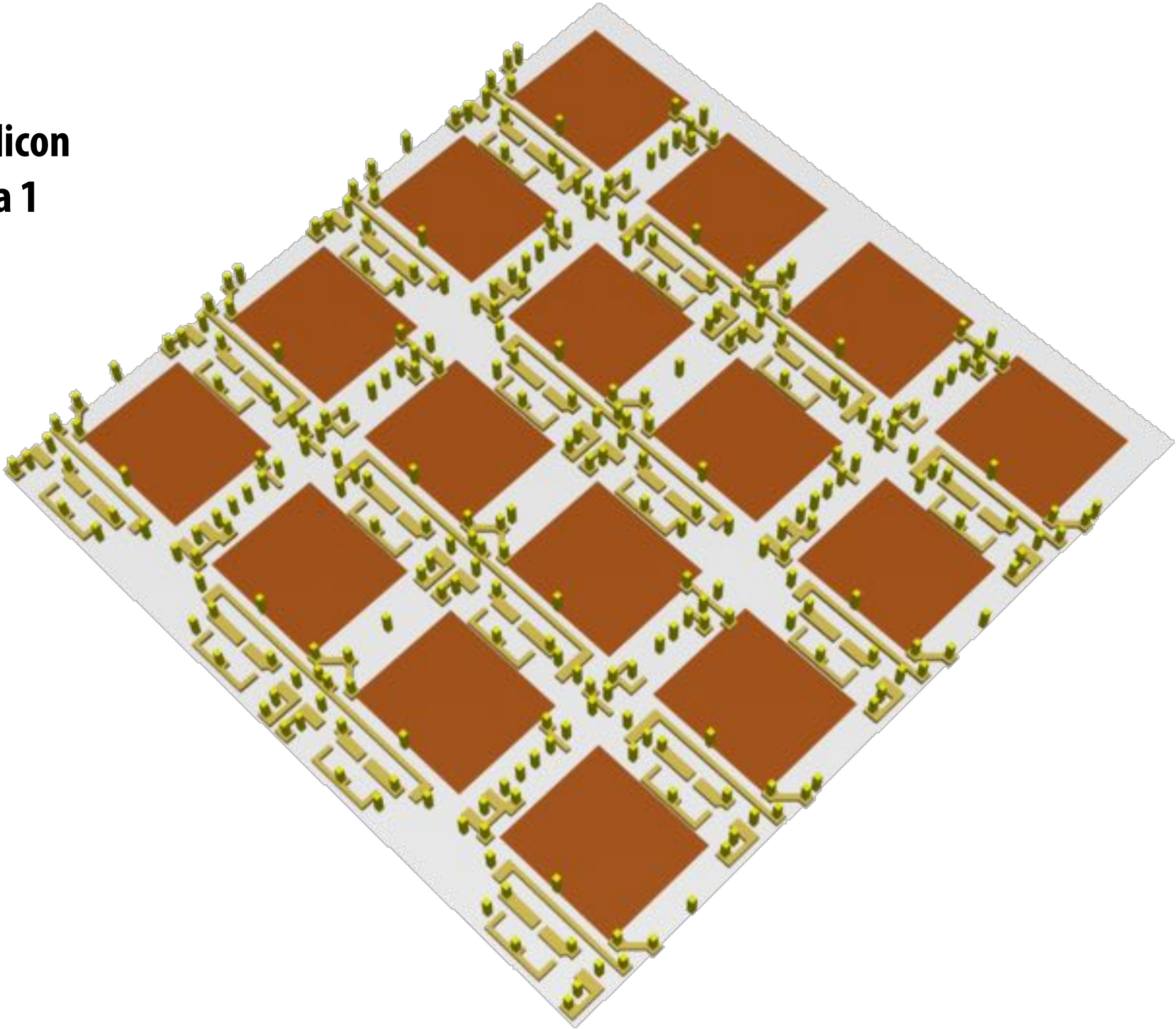
Metal 1
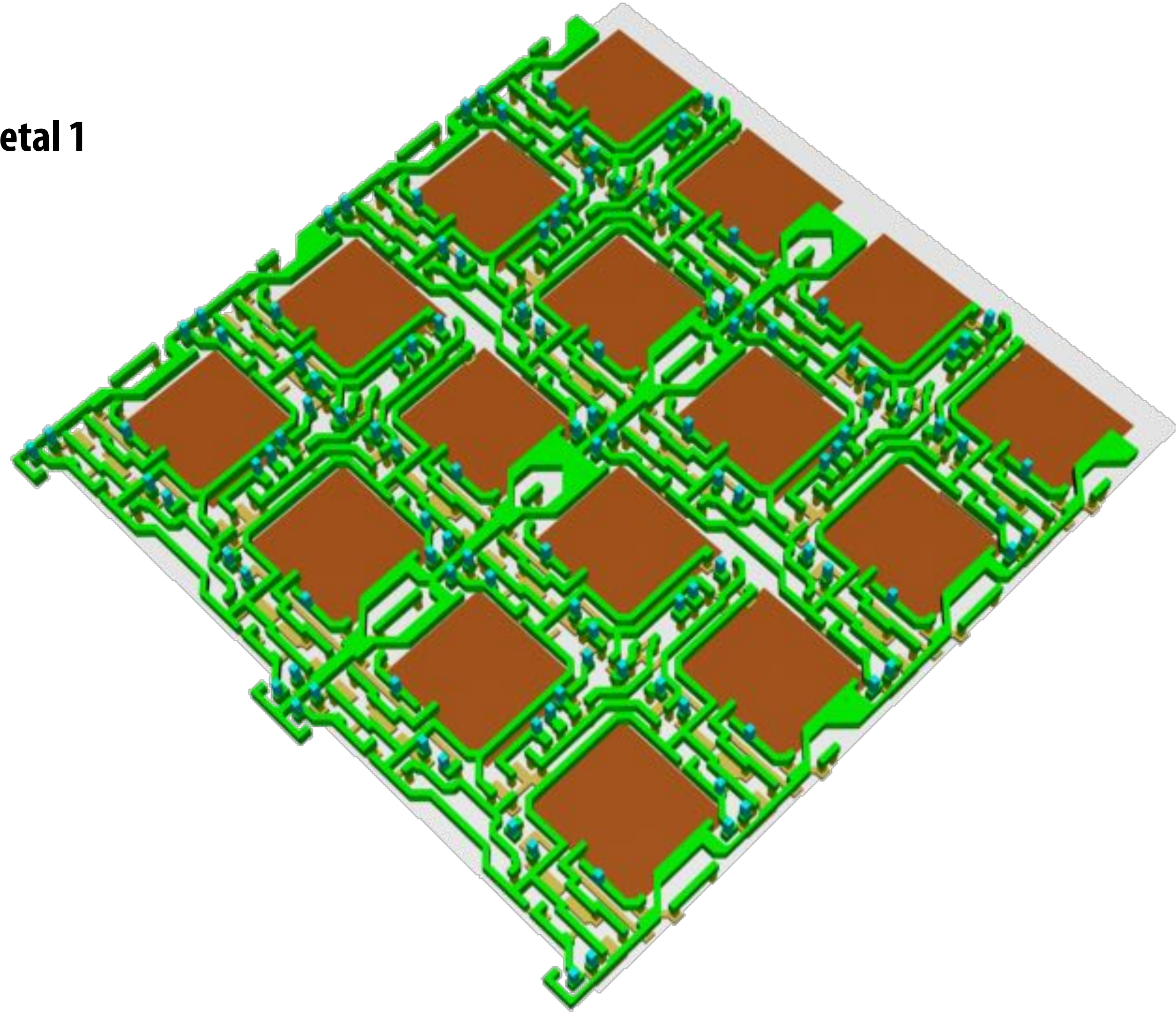
Metal 2
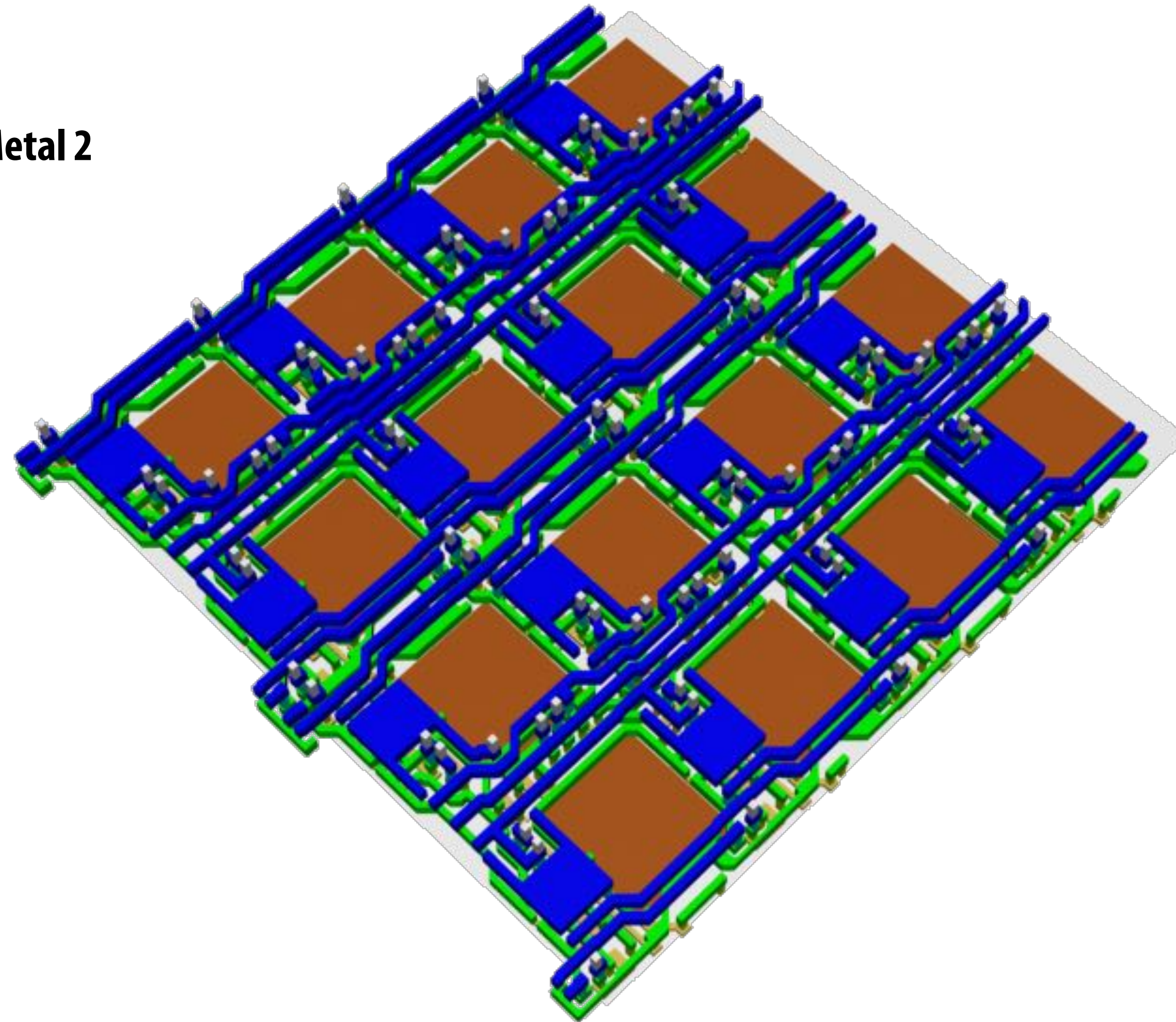
Metal 3
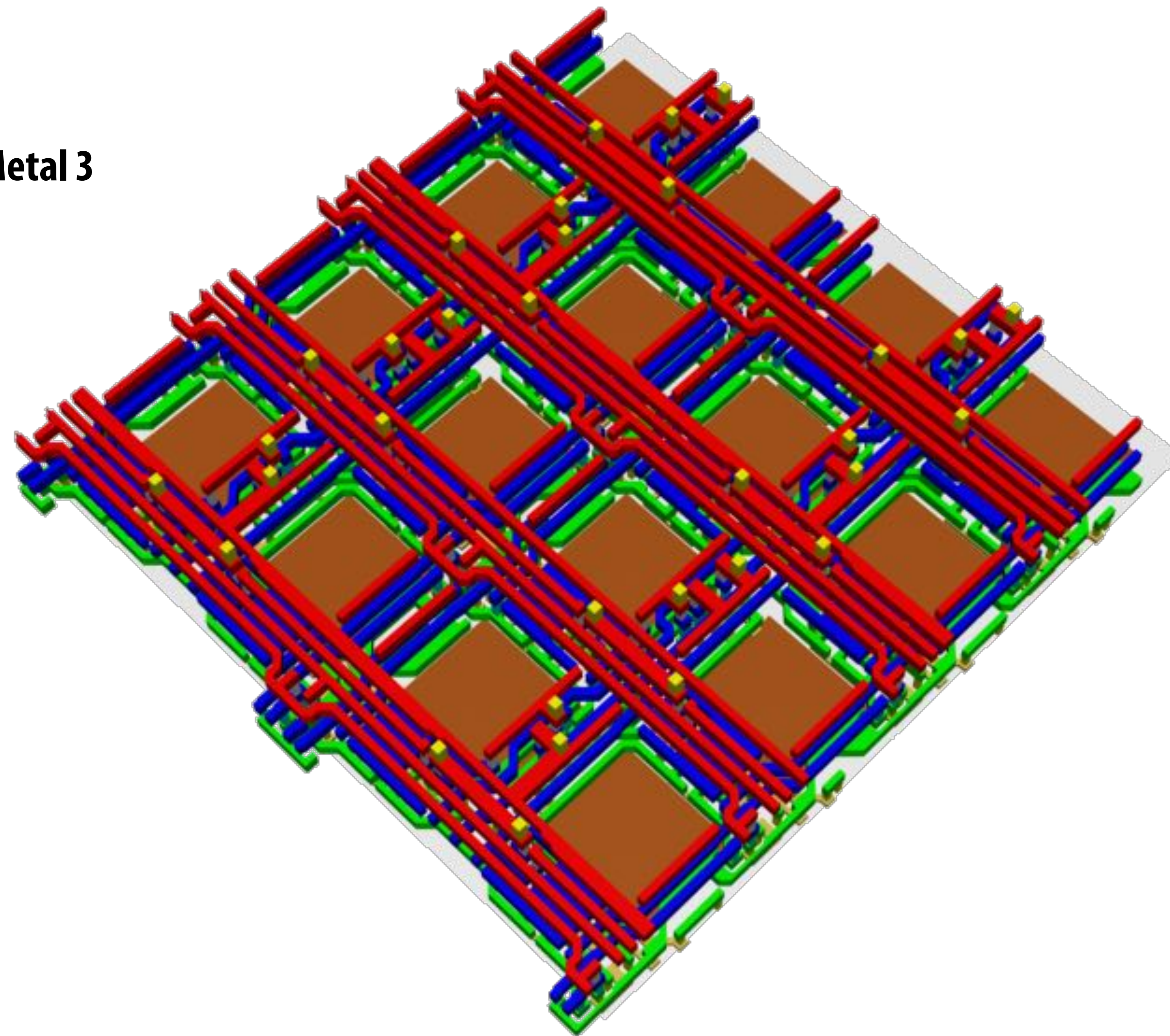
Metal 4
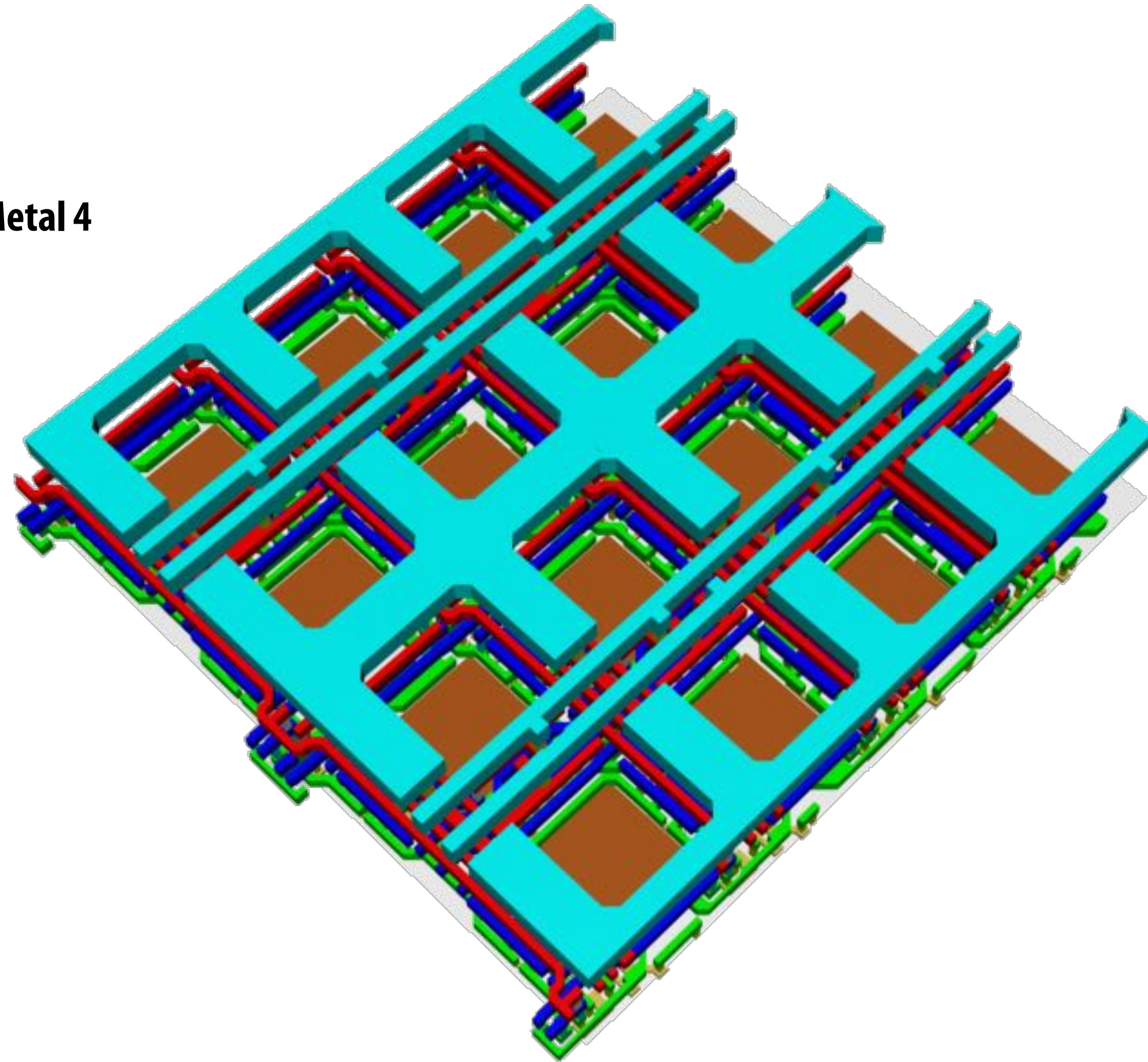
Color filter array
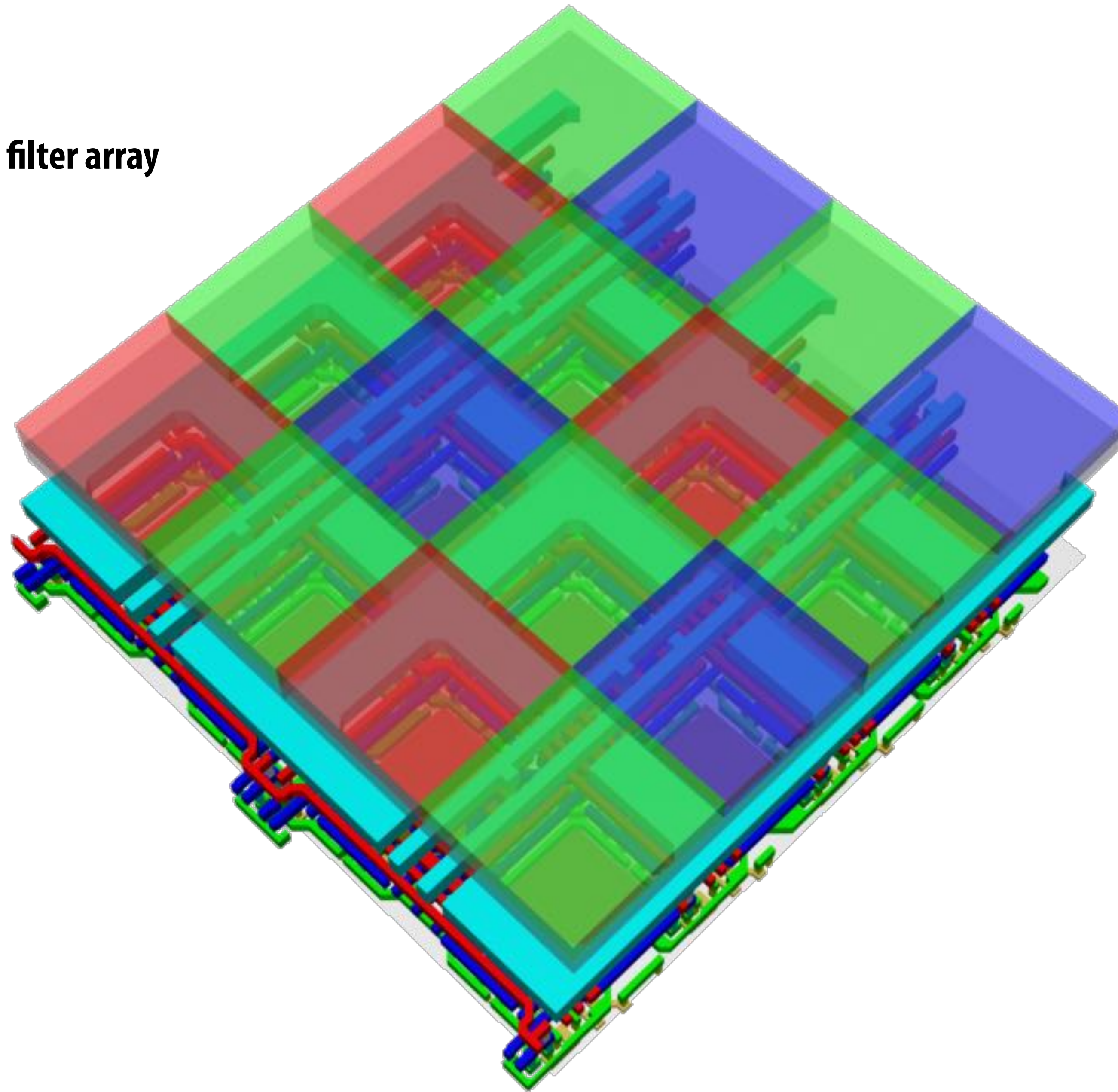
Pixel fill factor

Fraction of pixel area that integrates incoming light
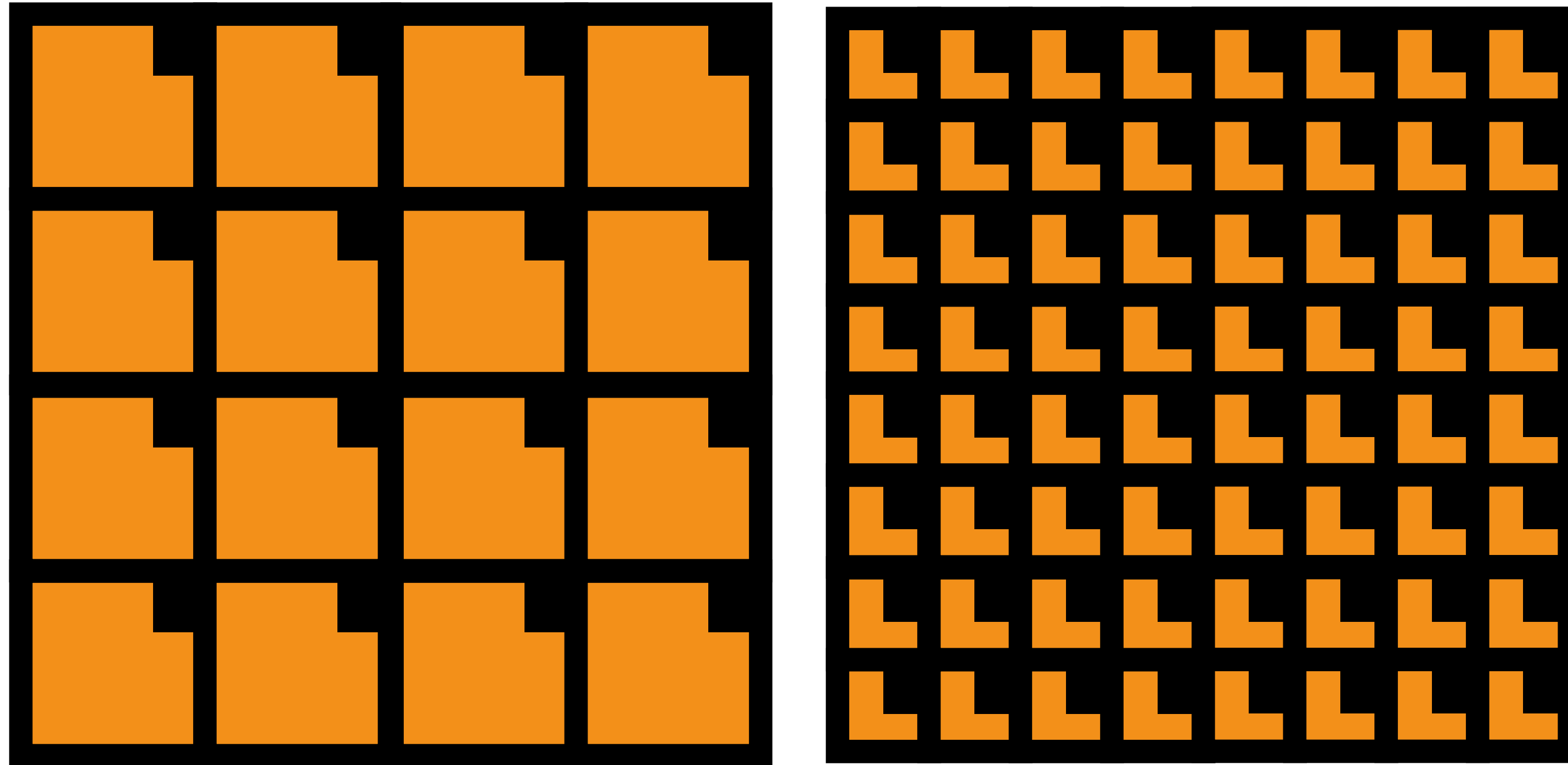
■ Photodiode area   ■ Non photosensitive (circuitry)

CMOS sensor pixel

Figure 3
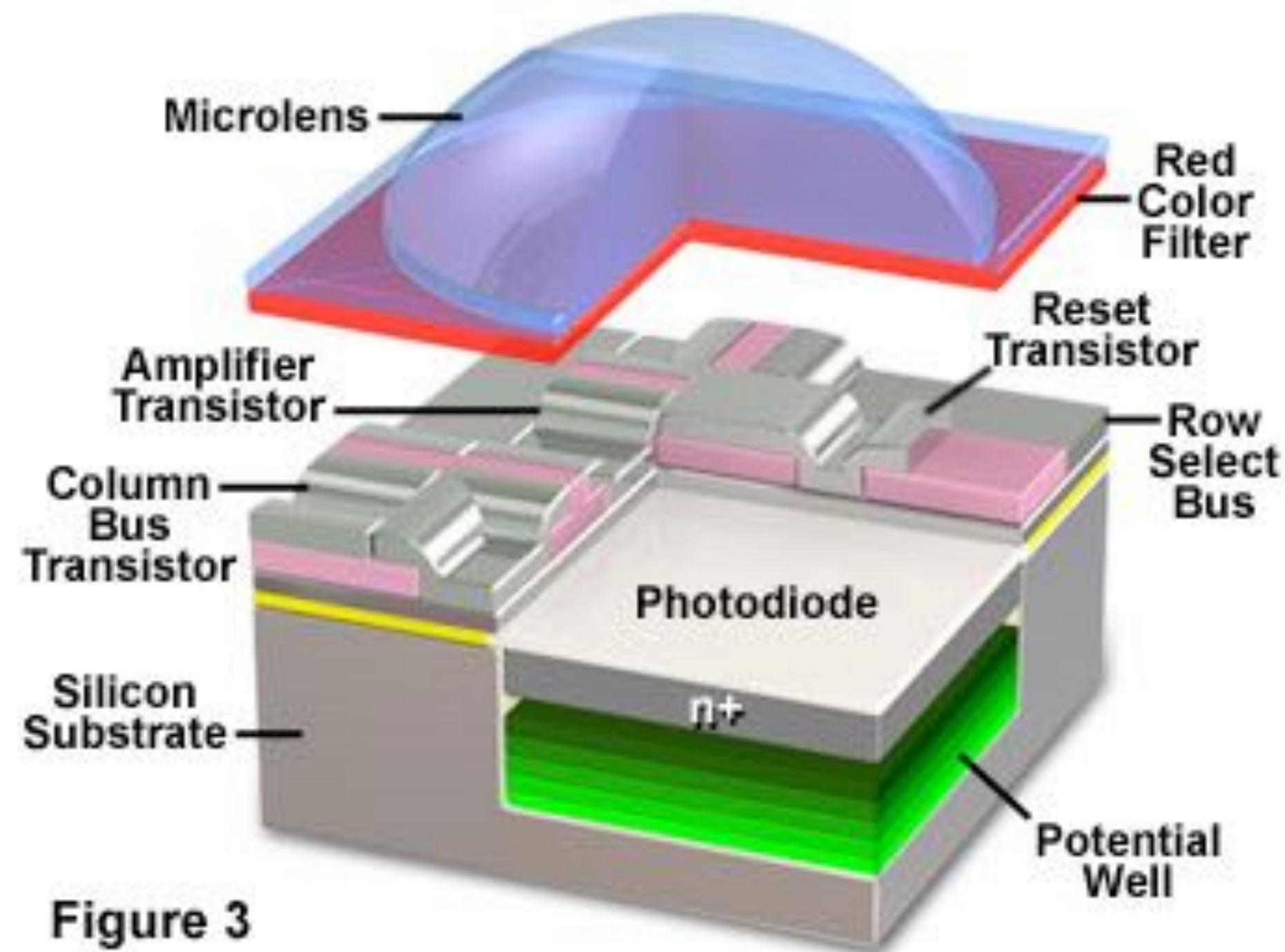
Color filter attenuates light

Microlens (a.k.a. lenslet) steers light toward photo-sensitive region (increases light-gathering capability)

Advanced question: Microlens also serves to reduce aliasing signal. Why?

Using micro lenses to improve fill factor

MICRO-LENS LAYOUT

1 Pixel diagram
2 Centered micro lens in the middle of the sensor
3 Laterally displaced micro lens at the edge of the sensor

Shifted microlenses on M9 sensor.
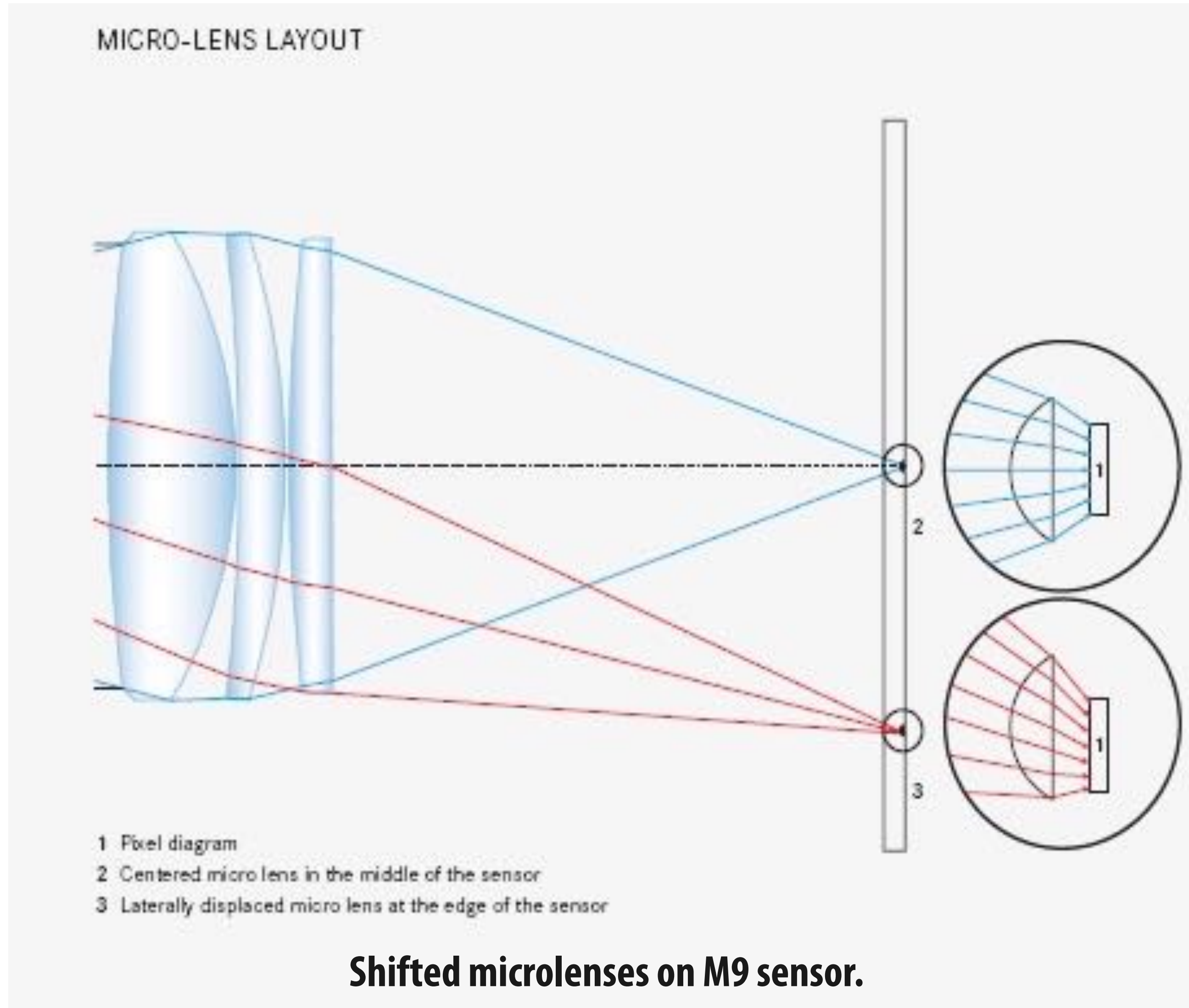
Leica M9

Optical cross-talk

Sensor architecture of a standard CMOS sensor (schematic diagram)

Incoming light

1 Microlens design with normal radius

2 Relatively large distance between color filter and photodiode

Microlens

Colour filter

Photodiode

Pixel

With some CMOS sensors, rays of incoming light at large angles of incidence can fail to reach the photodiode of the corresponding pixel and reach only the adjacent pixel. Or they are shadowed or reflected on the way to the pixel with the effect that the overall amount of light received by the pixels is less than the amount arriving through the microlenses.

Pixel optics for minimizing cross-talk

Sensor architecture of the Leica Max 24 MP sensor (schematic diagram)

1 Microlens design with varying radius

2 Relatively short distance between color filter and photodiode

In the case of the Leica Max 24 MP sensor, and in contrast to standard CMOS sensors, even light rays with large angles of incidence, e.g. from wide-angle lenses or large apertures, are captured precisely by the photodiodes of the sensor. This is enabled by the special microlens design and the smaller distance between the colour filter and photodiode, which allows more light to enter the system, and ensures that it falls more directly on the respective photodiodes.

Backside illumination sensor

- **Traditional CMOS: electronics block light**
- **Idea: move electronics underneath light gathering region**
  - **Increases fill factor**
  - **Reduces cross-talk due since photodiode closer to microns**
  - **Implication 1: better light sensitivity at fixed sensor size**
  - **Implication 2: equal light sensitivity at smaller sensor size (shrink sensor)**
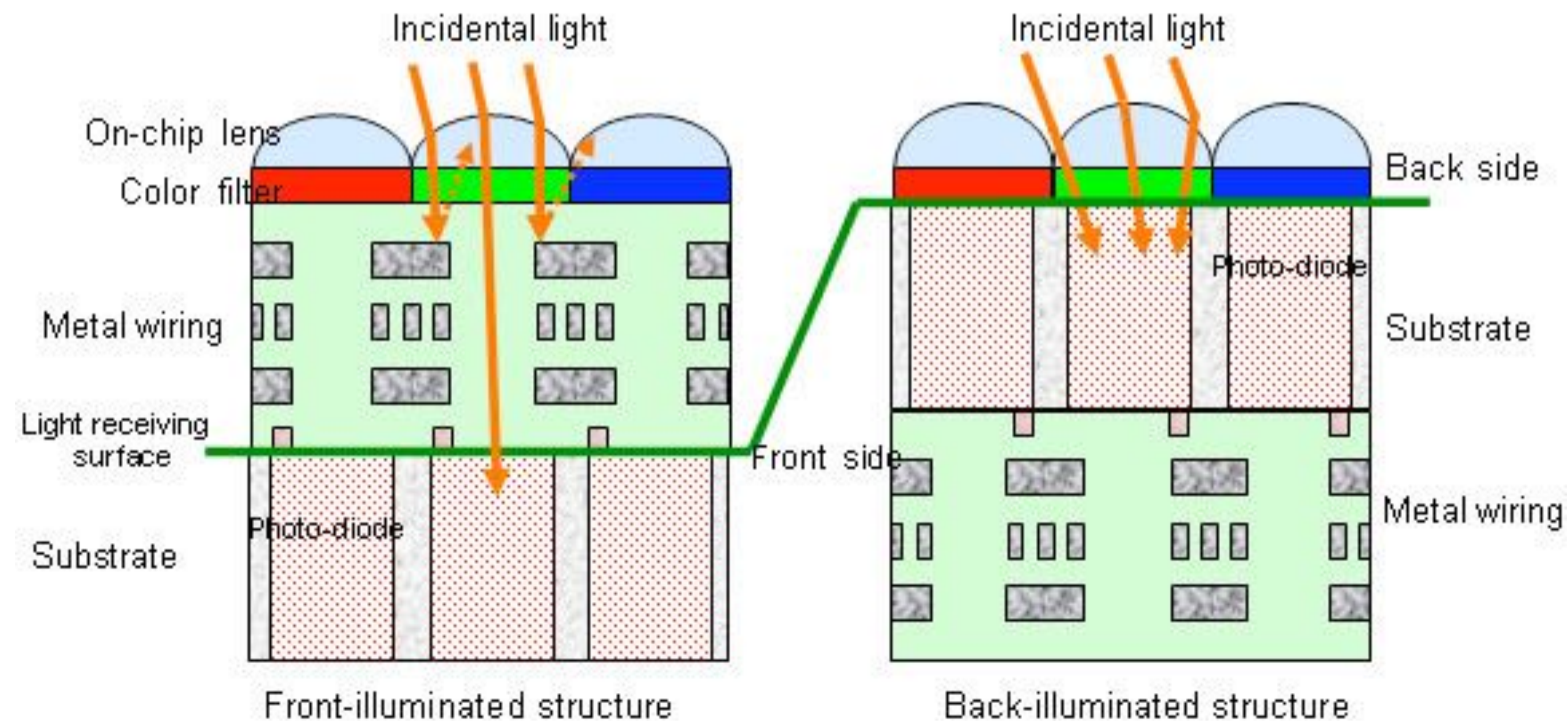
Pixel saturation and noise

Saturated pixels
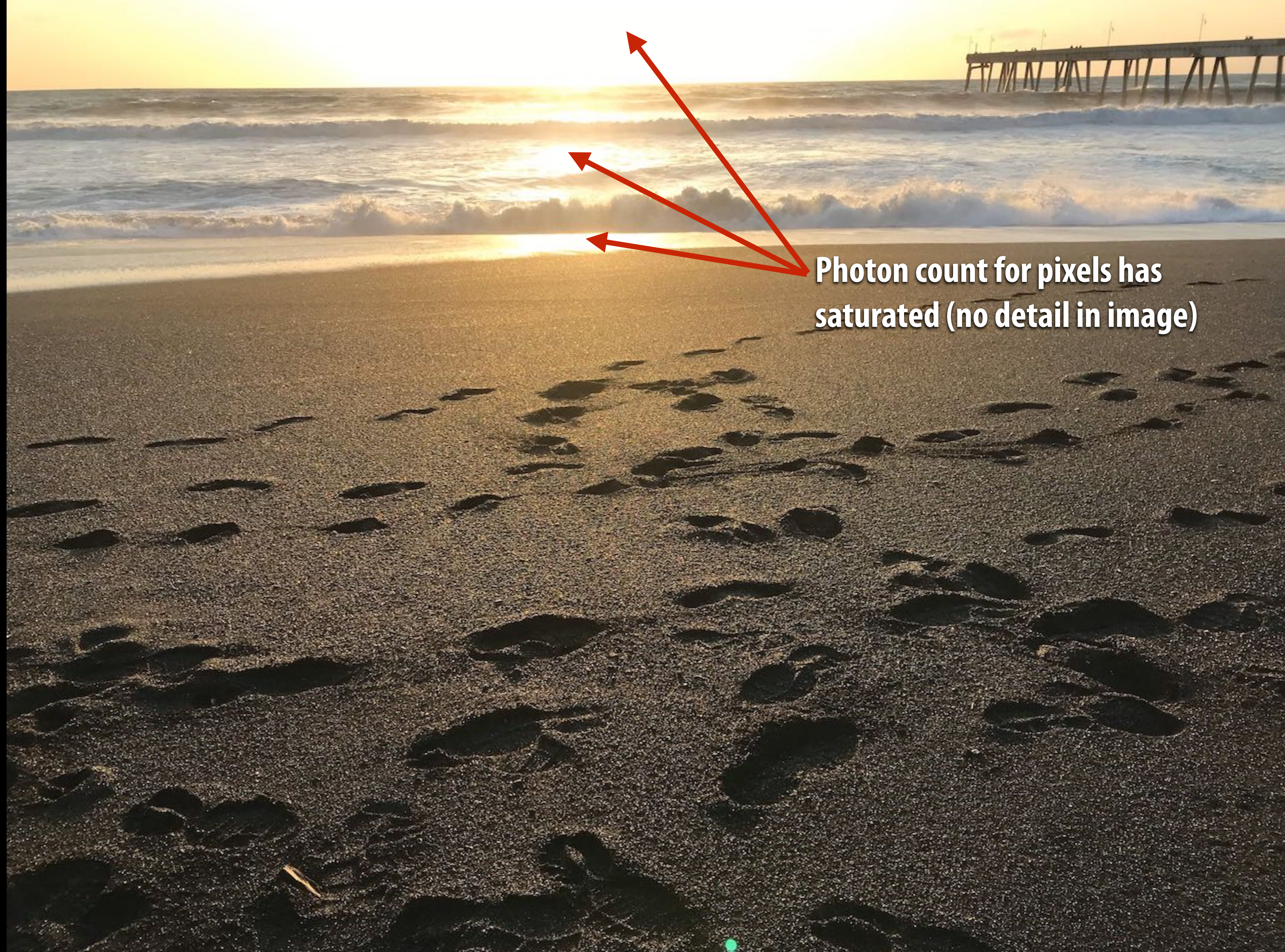
Photon count for pixels has saturated (no detail in image)

Full-well capacity
Pixel saturates when photon capacity is exceeded

Digital Cameras: Sensor Full Well

© Roger N. Clark
www.clarkvision.com

Canon 5D
Canon 1D Mark II

Canon 1D Mark III

Canon 5D Mark II   Nikon D3

Canon 1D
Mark IV

Canon 20D

Canon 40D   Canon 350D   Canon 10D

Nikon
D300   Nikon D200   Nikon D50

Canon 50D

Canon S60   Nikon D70

Canon
7D

Canon S70

Y-axis: Electrons — 0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000, 110000

X-axis: Pixel Pitch (microns) — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Saturated pixels

Bigger sensors = bigger pixels (or more pixels?)

- **iPhone X (1.2 micron pixels, 12 MP)**

- **Nikon D7000 (APS-C)**
  (4.8 micron pixels, 16 MP)

- **Nikon D4 (full frame sensor)**
  (7.3 micron pixels, 16 MP)

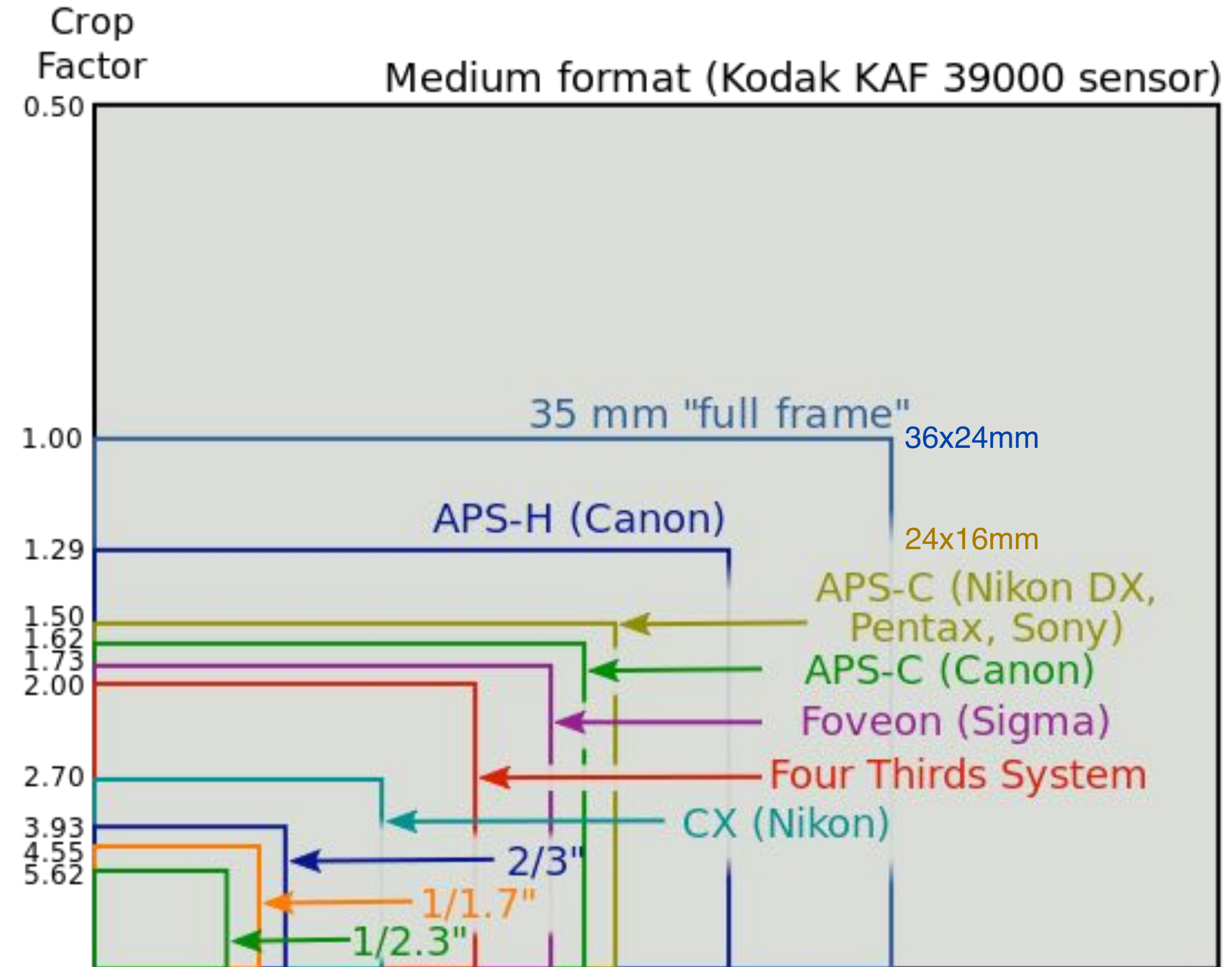- **Implication: very high pixel count sensors can be built with current CMOS technology**
  - **Full frame sensor with iPhone X pixel size ~ 600 MP sensor**
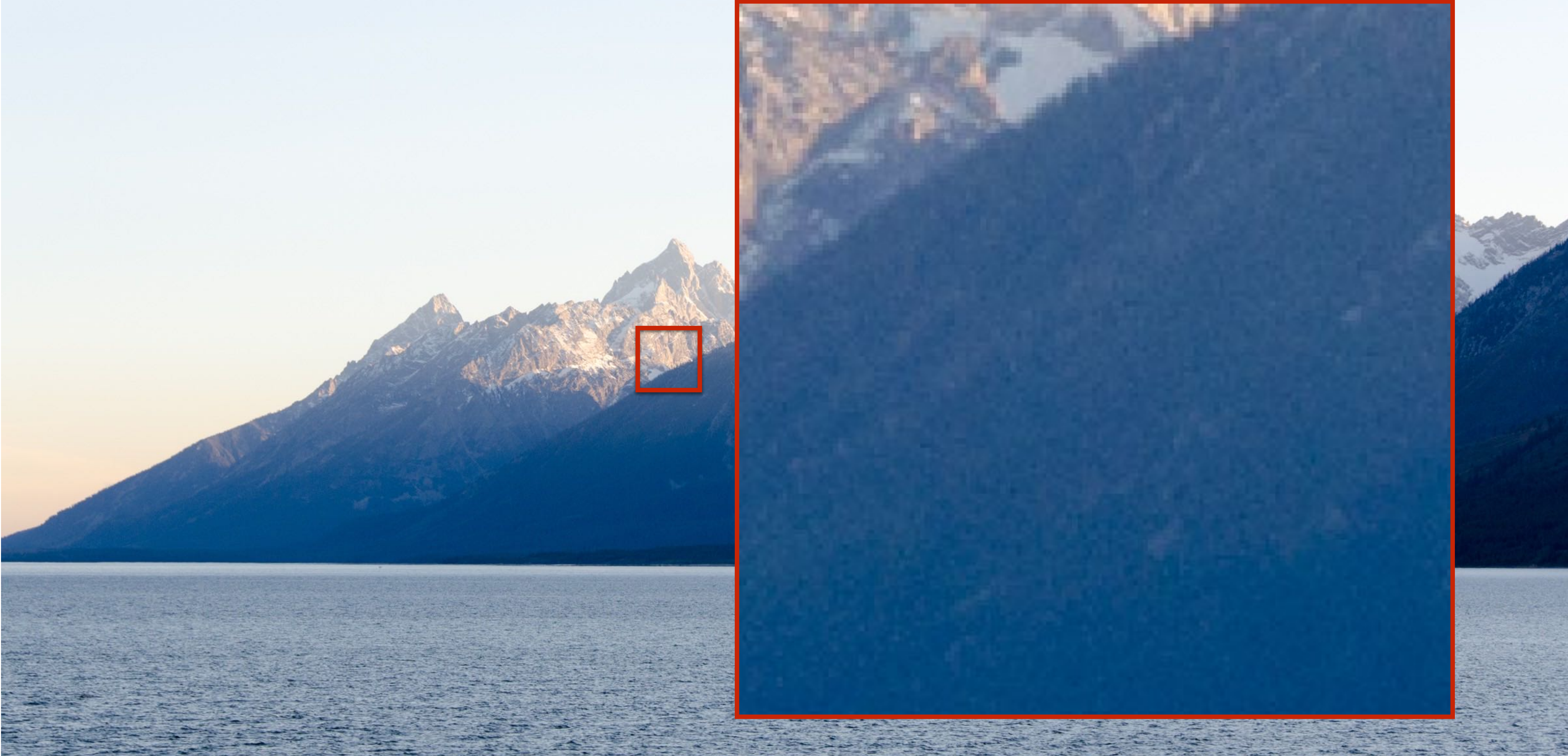
Measurement noise

We've all been frustrated by noise in
low-light photographs

(or in shadows in day time images)

Measurement noise

Measurement noise

Sources of measurement noise

- **Photon shot noise:**

  - **Photon arrival rate takes on Poisson distribution**

  - **Standard deviation = sqrt(N)   (N = number of photon arrivals)**

  - **Signal-to-noise ratio (SNR) = N/sqrt(N)**

  - **Implication: brighter the signal, the higher the SNR**

- **Dark-shot noise**

  - **Due to leakage current in sensor**

  - **Electrons dislodged due to thermal activity (increases exponentially with sensor temperature)**

- **Non-uniformity of pixel sensitivity (due to manufacturing defects)**

- **Read noise**

  - **e.g., due to amplification / ADC**

Dark shot noise / read noise

Black image examples: Nikon D7000, High ISO

1 sec exposure

Read noise

Digital Cameras:
Sensor Read Noise
(electrons)
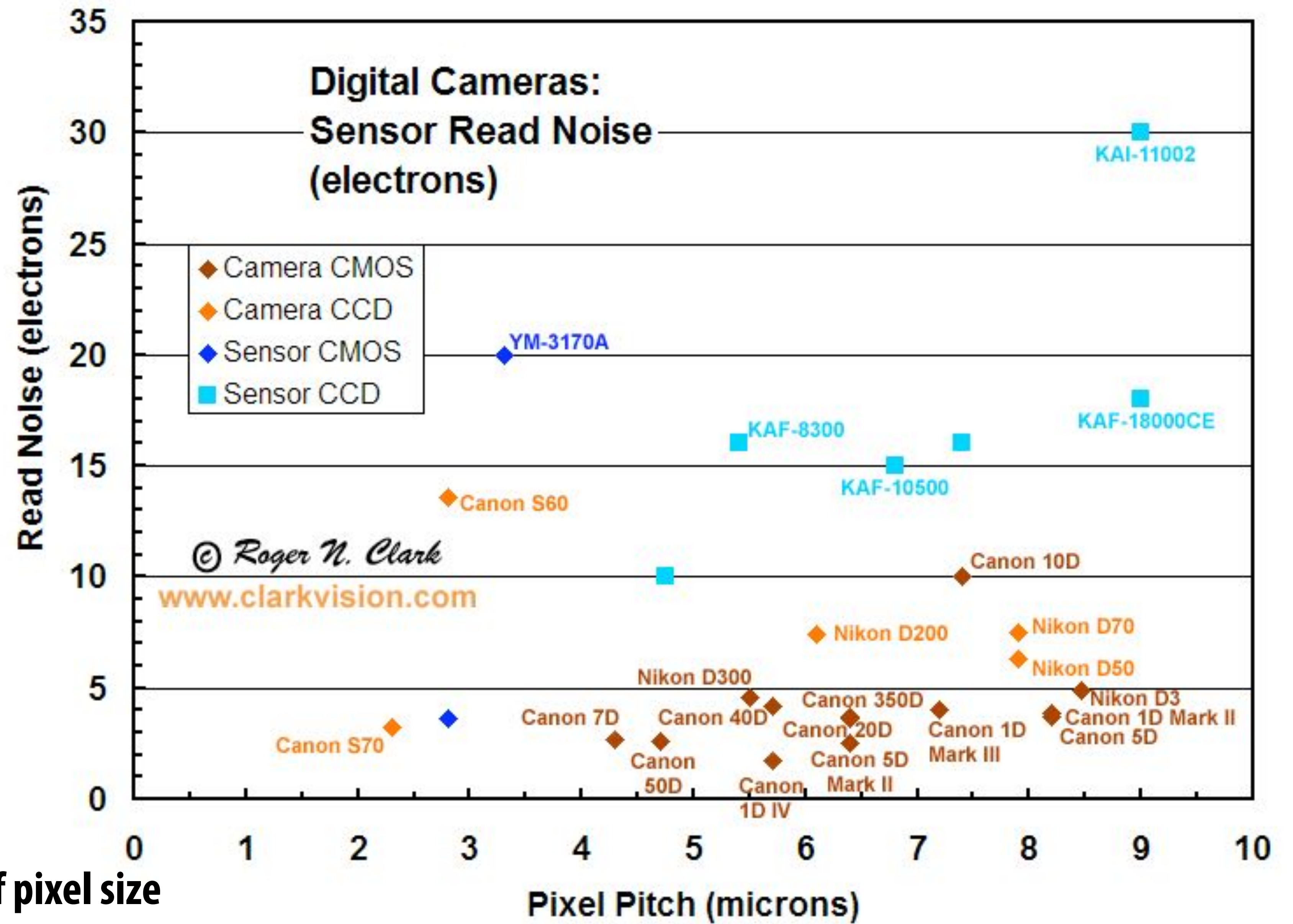
Read noise is largely independent of pixel size
Large pixels + bright scene = large N
So, noise determined largely by photon shot noise

Maximize light gathering capability

- **Goal: increase signal-to-noise ratio**

  - Dynamic range of a pixel (ratio of brightest light measurable to dimmest light measurable) is determined by the noise floor (minimum signal) and the pixel's full-well capacity (maximum signal)

- **Use big pixels**

  - Nikon D4: 7.3 um

  - iPhone X: 1.2 um
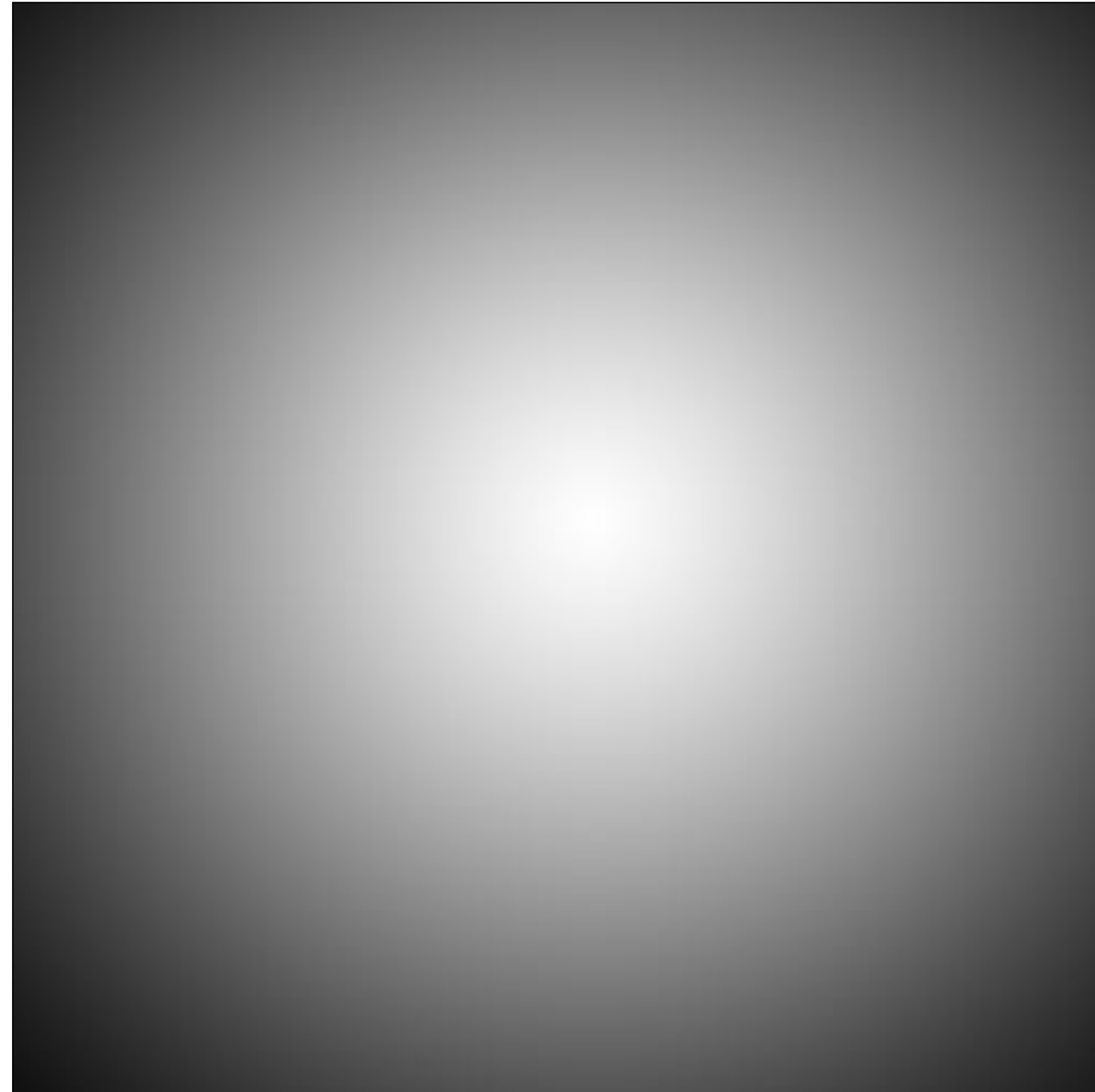
- **Manufacture sensitive pixels**

  - Good materials

  - High fill factor

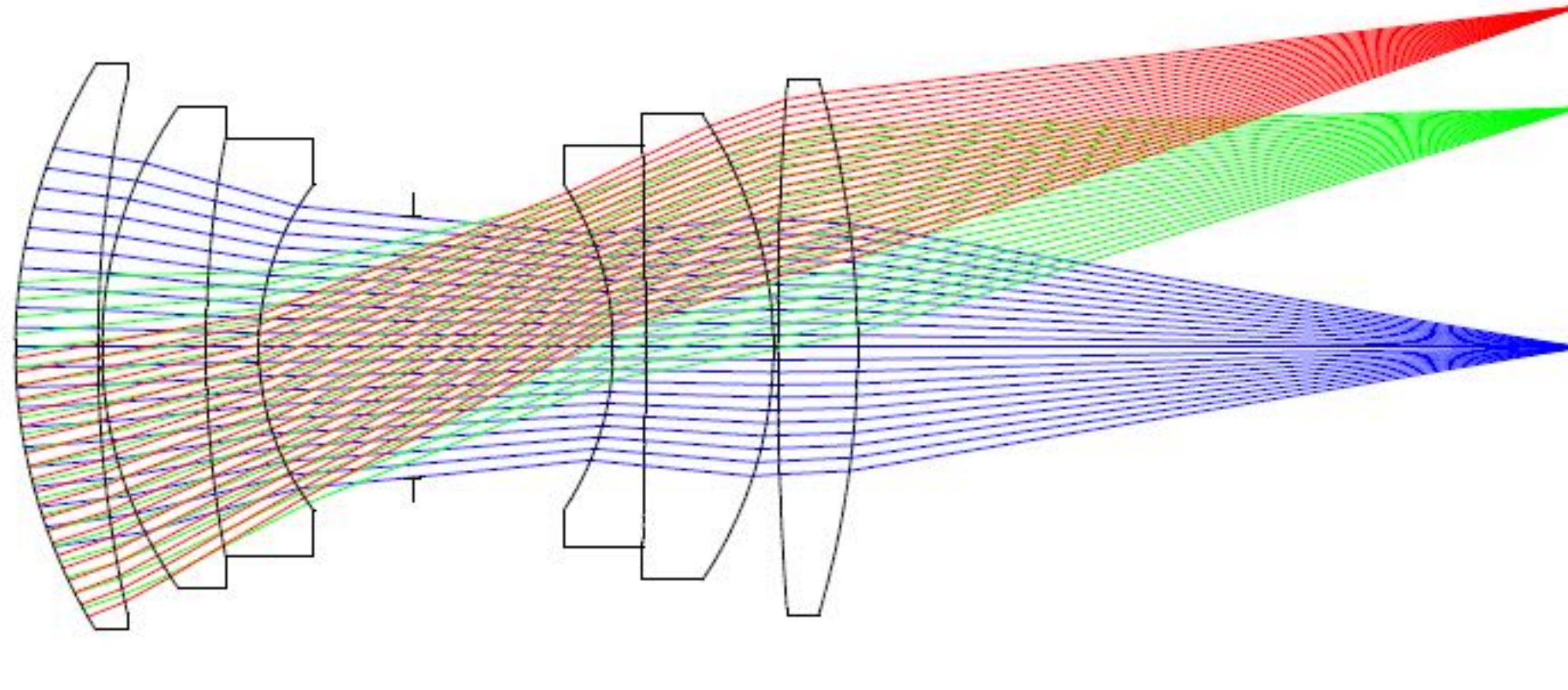# Artifacts arising from lenses

Vignetting

This is a photograph of a white wall

(Note: I contrast-enhanced the image to show effect more prominently)

Types of vignetting

Optical vignetting: less light reaches edges of sensor due to physical obstruction in lens

Pixel vignetting: light reaching pixel at an oblique angle is less likely to hit photosensitive region than light incident from straight above (e.g., obscured by electronics)
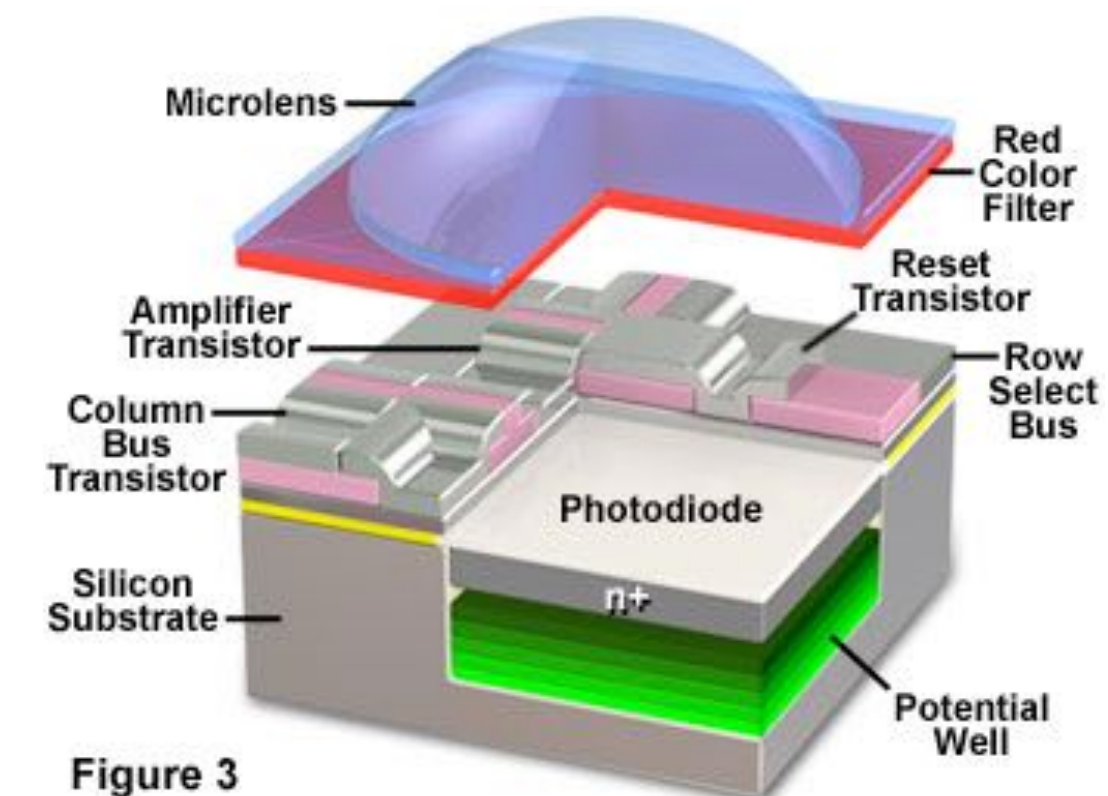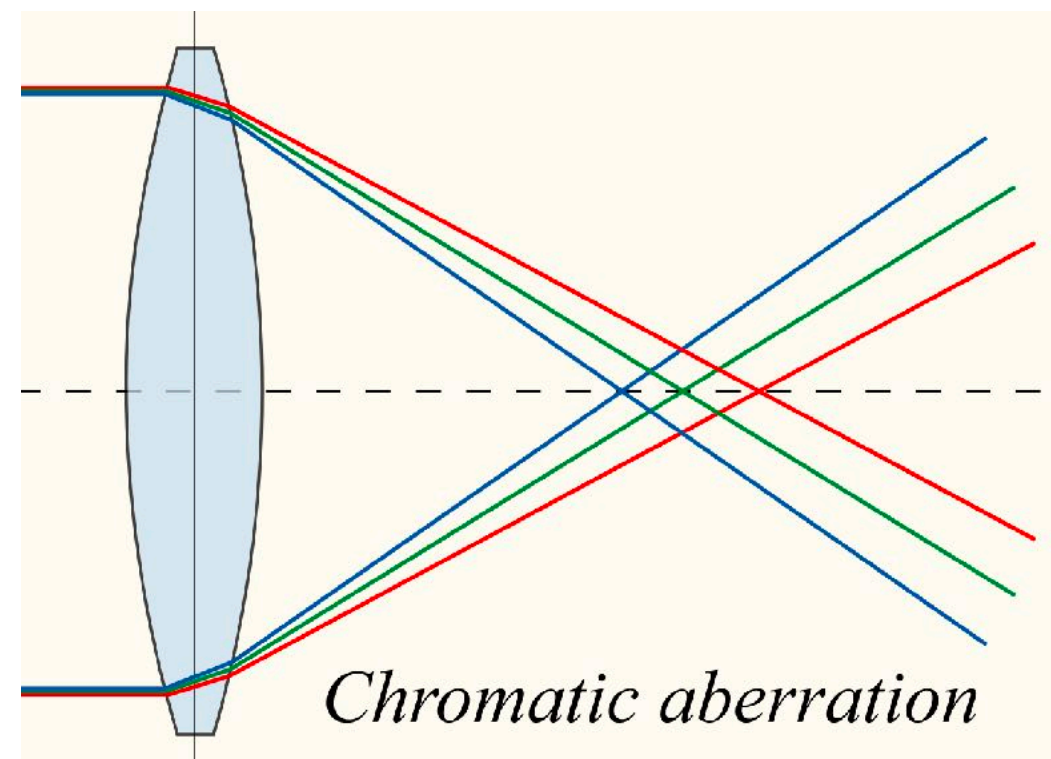
— **Microlens reduces pixel vignetting**

Figure 3

Image credit: Mark Butterworth

Chromatic aberration

Different wavelengths of light are refracted by different amounts
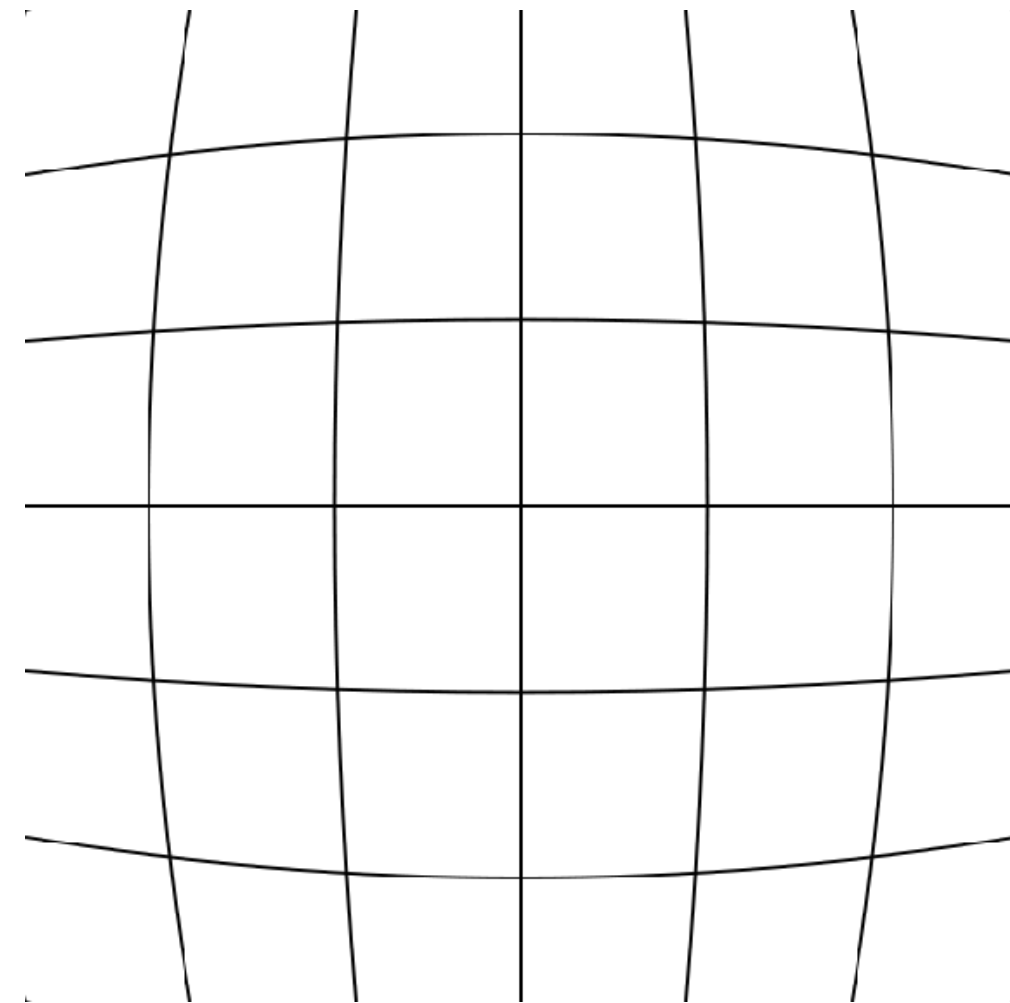

Chromatic aberration

More challenges

- ■ **Chromatic shifts over sensor**
  - **Pixel light sensitivity changes over sensor due to interaction with microlens (Index of refraction depends on wavelength, so some wavelengths are more likely to suffer from cross-talk or reflection. Ug!)**
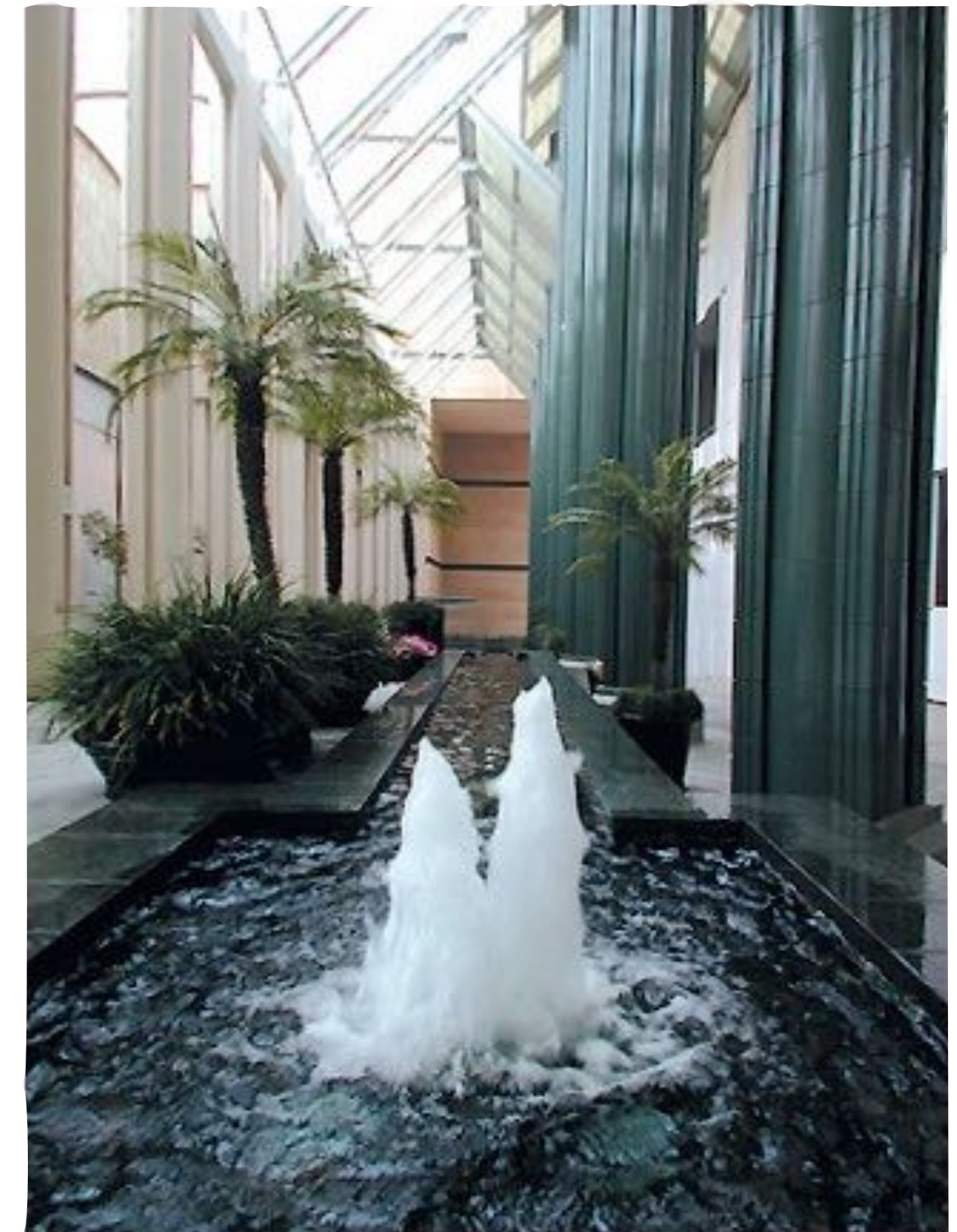
- ■ **Lens distortion**

Pincushion distortion

Captured Image

Corrected Image

The message so far

- **Physical constraints of image formation by a camera create artifacts in the recorded image**

- **We are going to rely on processing to reduce / correct for these artifacts**

# A simple RAW image processing pipeline

Given the physical reality of how a lens+sensor system works, now let's look at how software transforms raw sensor output into a high-quality RGB image.

Adopting terminology from Texas Instruments OMAP Image Signal Processor pipeline

(since public documentation exists)

Assume: receiving 12 bits/pixel Bayer mosaiced data from sensor

Optical clamp: remove sensor offset bias

```
output_pixel = input_pixel - [average of pixels from optically black region]
```
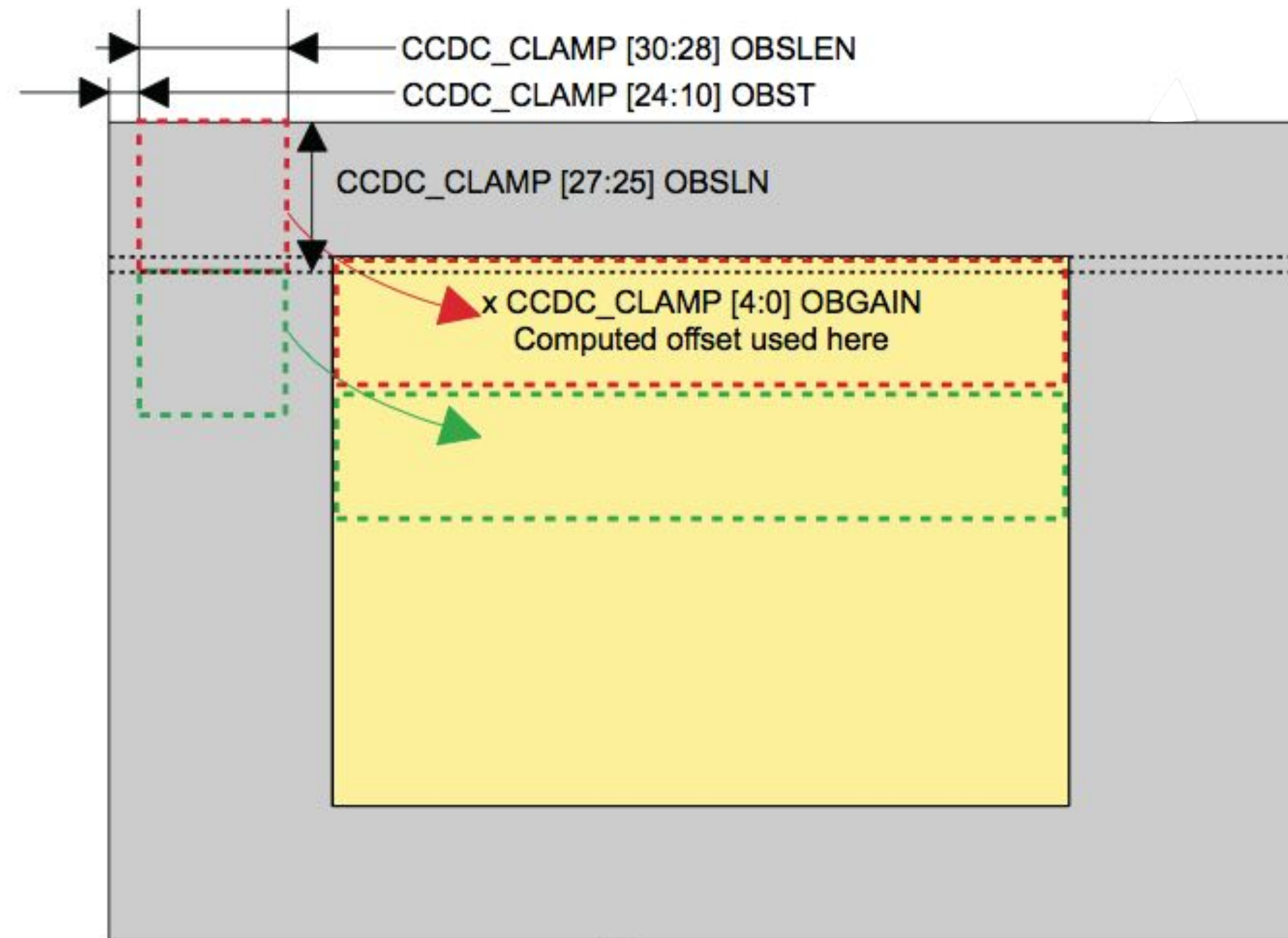

Remove bias due to sensor black level
(from nearby sensor pixels at time of shot)

Correct for defective pixels

- **Store LUT with known defective pixels**

  - e.g., determined on manufacturing line, during sensor calibration and test

- **Example correction methods**

  - Replace defective pixel with neighbor

  - Replace defective pixel with average of neighbors

  - Correct defect by subtracting known bias for the defect

```
output_pixel = (isdefectpixel(current_pixel_xy)) ?
                  average(previous_input_pixel, next_input_pixel) :
                  input_pixel;
```

"Hot pixel" suppression

```
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH * HEIGHT];


for (int j=0; j<HEIGHT; j++) {
   for (int i=0; i<WIDTH; i++) {
      float min_value = min( min(input[(j-1)*WIDTH + i], input[(j+1)*WIDTH + i]),
                             min(input[j*WIDTH + i-1], input[j*WIDTH + i+1]) );
      float max_value = max( max(input[(j-1)*WIDTH + i], input[(j+1)*WIDTH + i]),
                             max(input[j*WIDTH + i-1], input[j*WIDTH + i+1]) );
      output[j*WIDTH + i] = clamp(min_value, max_value, input[j*WIDTH + i]);
   }
}
```

This filter clamps pixels to the min/max of its cardinal neighbors
(e.g., hot-pixel suppression — no need for a lookup table)

Lens shading compensation

■ **Correct for vignetting artifacts**

- **Good implementations will consider wavelength-dependent vignetting**
  (that creates chromatic shift over the image)

Need to invert the vignetting effect

■ **Possible implementations:**

- **Use "flat-field photo" stored in memory**

  - **e.g., lower resolution buffer, upsampled on-the-fly**

  - **Use analytic function to model required correction**
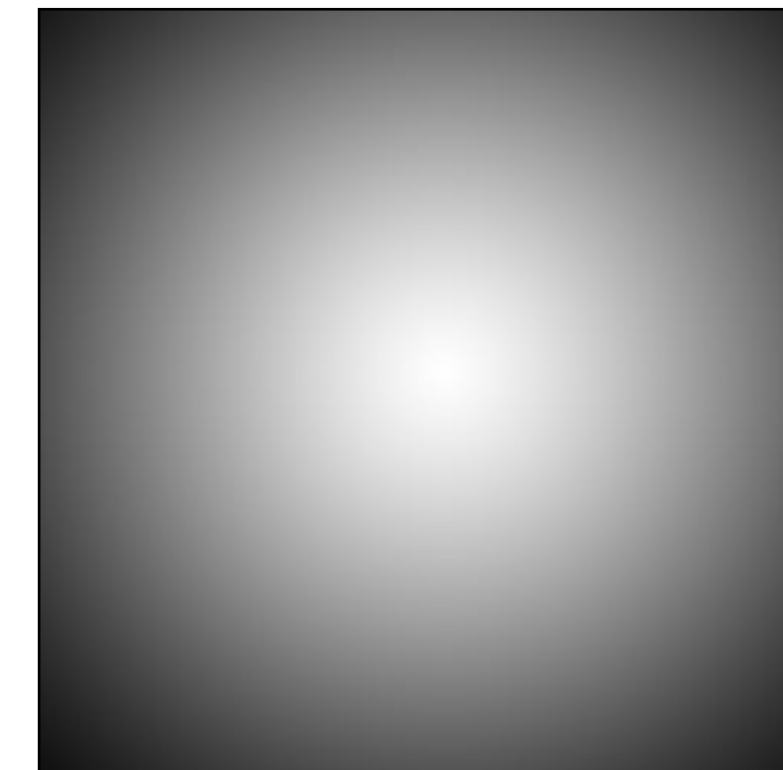
```
gain = upsample_compensation_gain_buffer(current_pixel_xy);
output_pixel = gain * input_pixel;
```

Demosiac

- **Produce RGB image from mosaiced input image**

- **Basic algorithm: bilinear interpolation of mosaiced values (need 4 neighbors)**

- **More advanced algorithms:**

  - **Bicubic interpolation (wider filter support region… may overblur)**

  - **Good implementations attempt to find and preserve edges in photo**
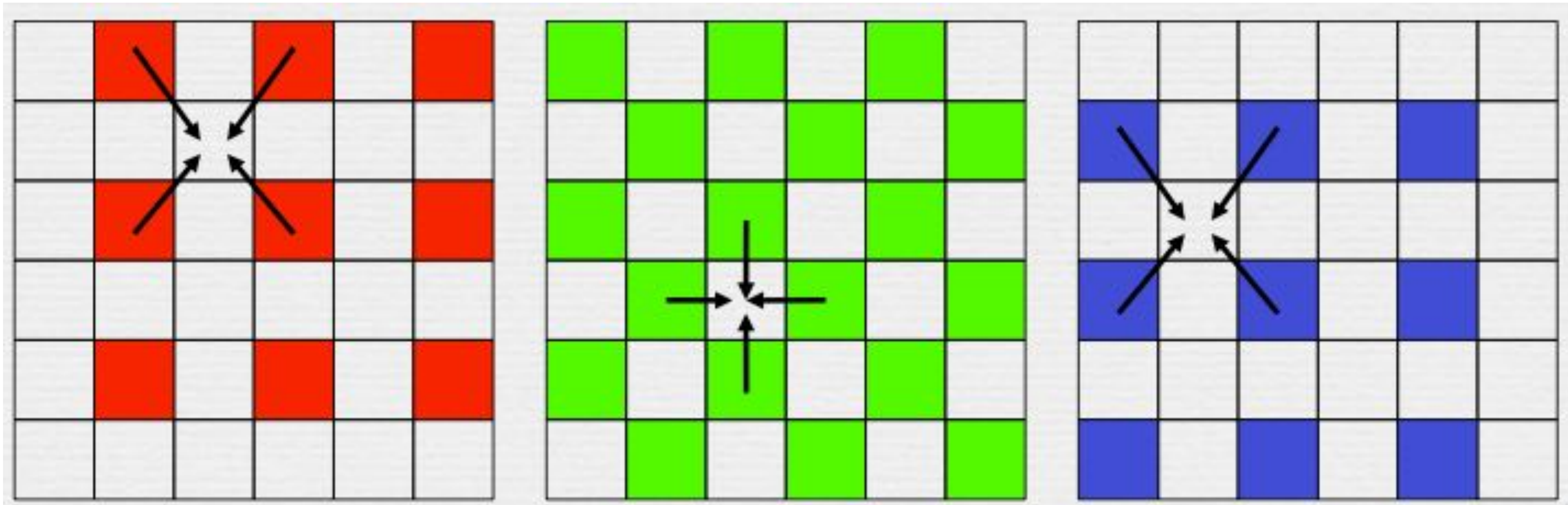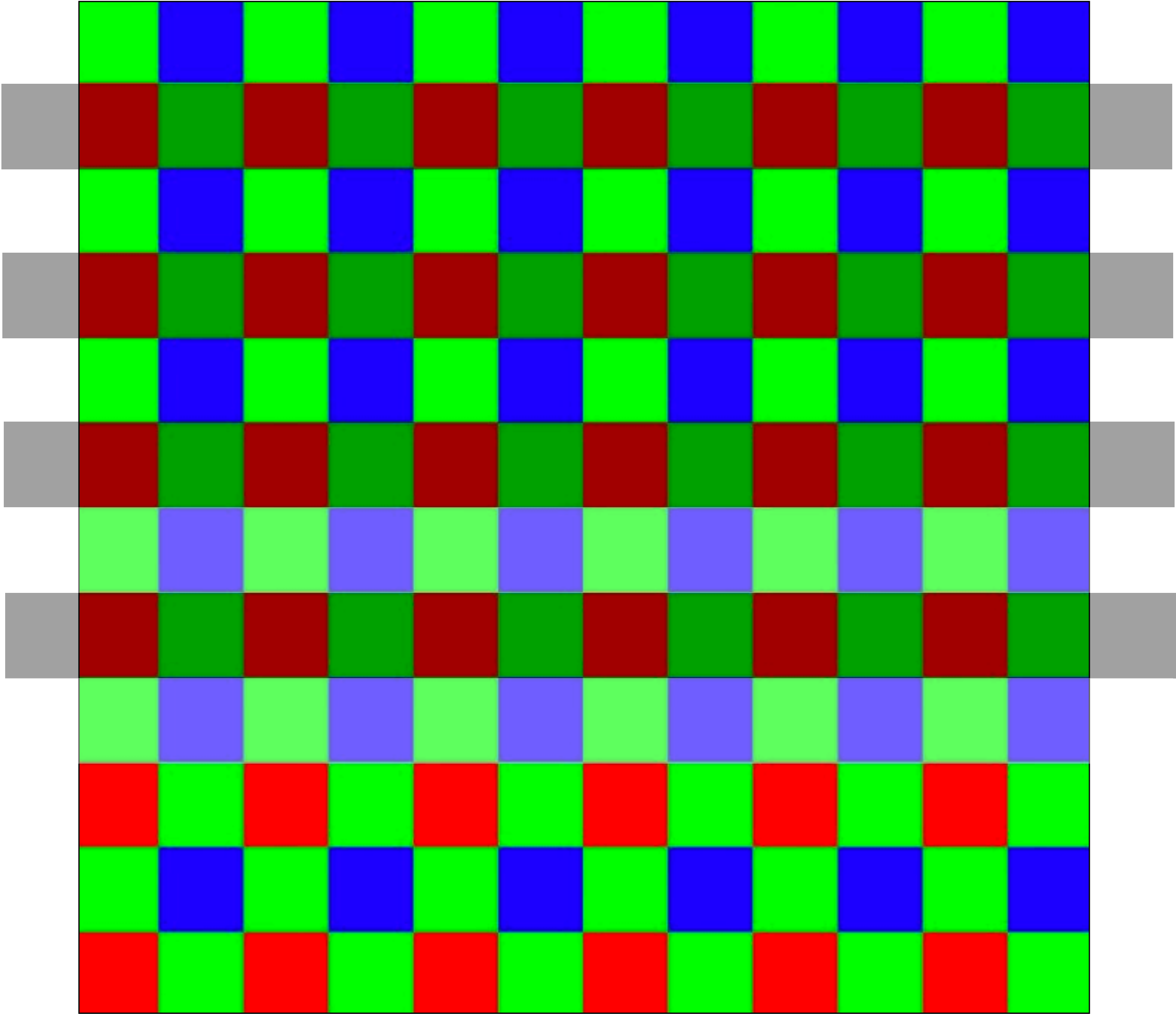
Demosaicing errors

What will demosaiced result look like if this black and white signal was captured by the sensor?
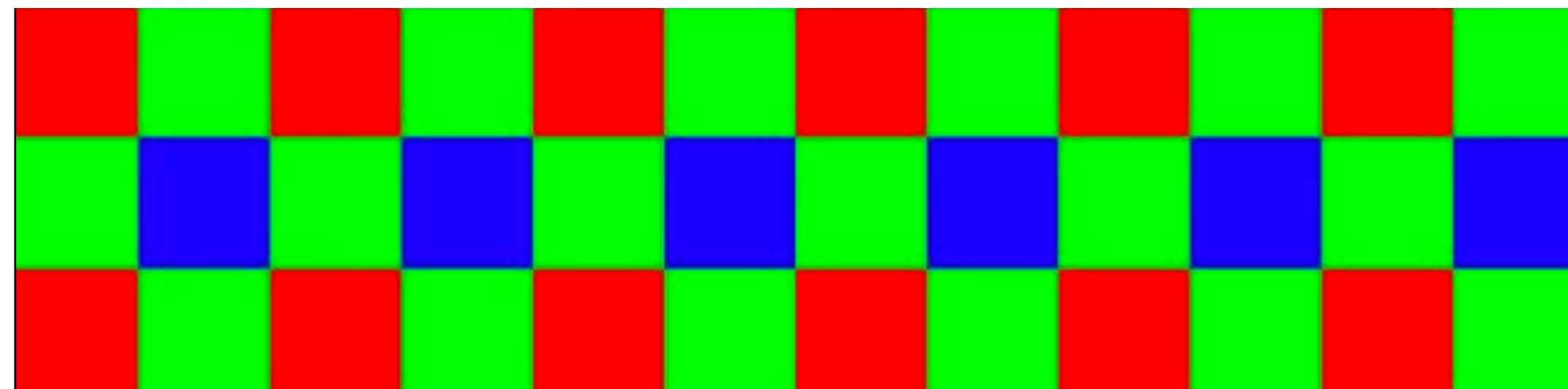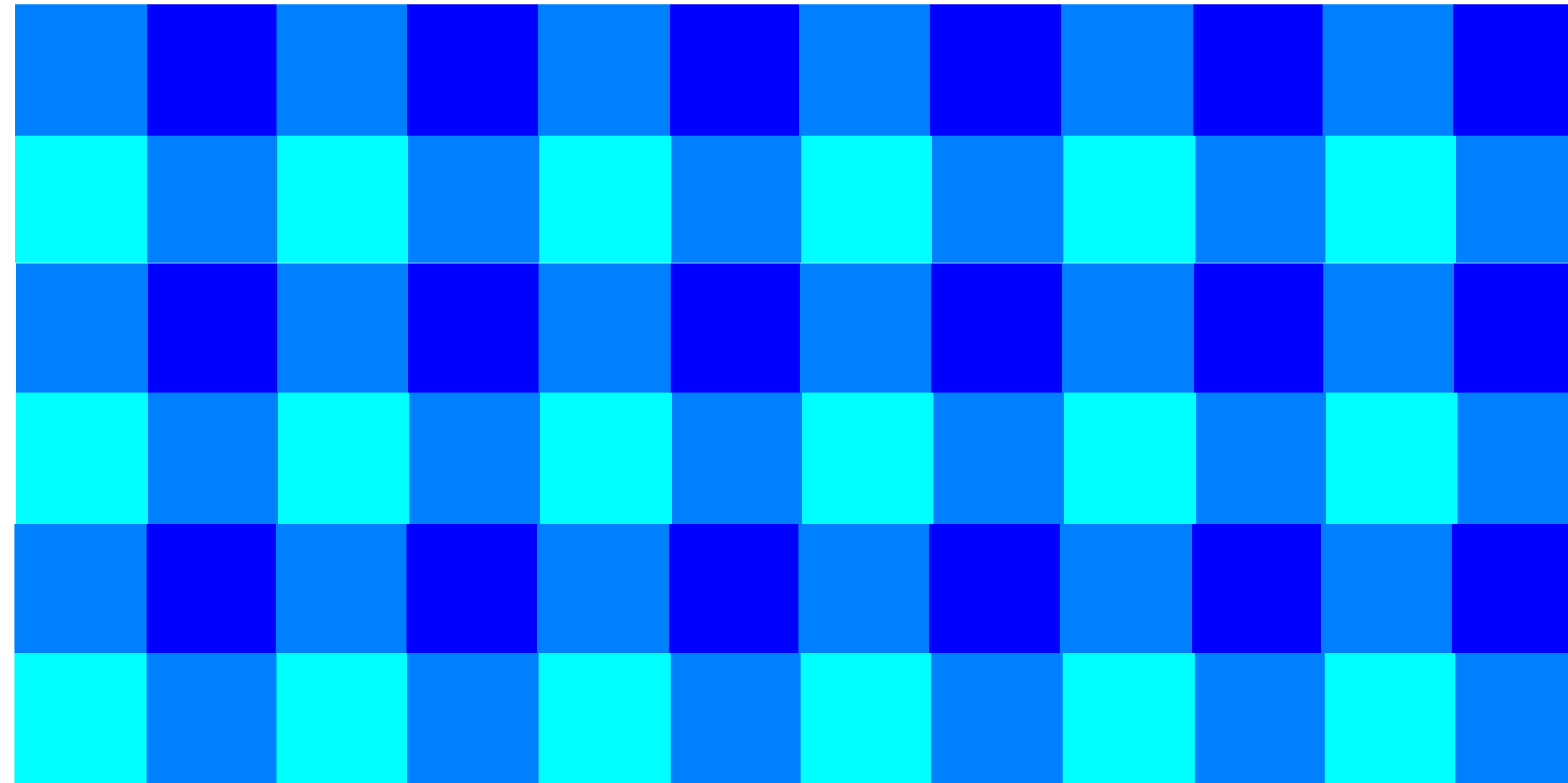
Demosaicing errors
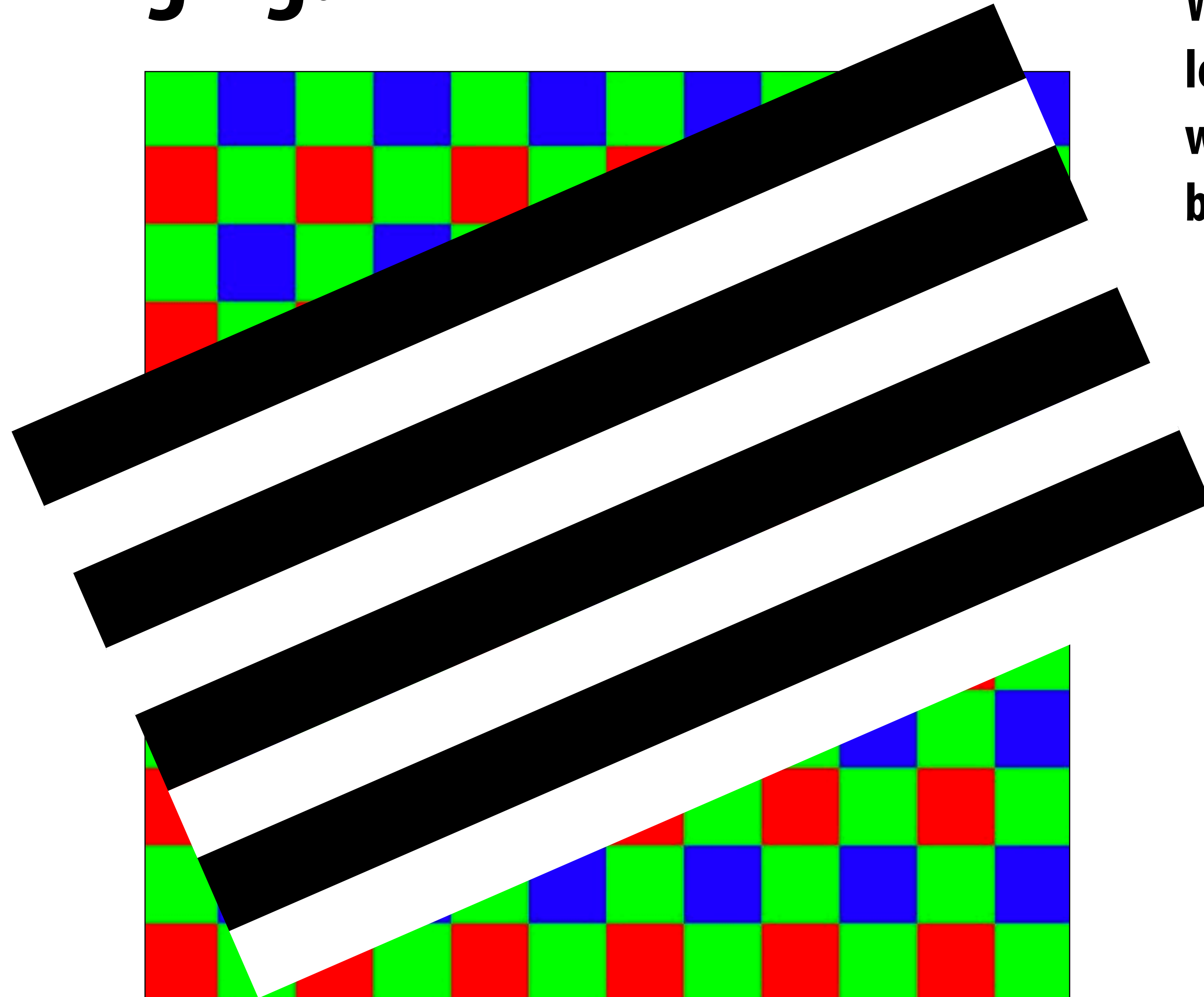

(Visualization of signal and Bayer pattern)

Demosaicing errors

No red measured.

Interpolation of green yields dark/light pattern.
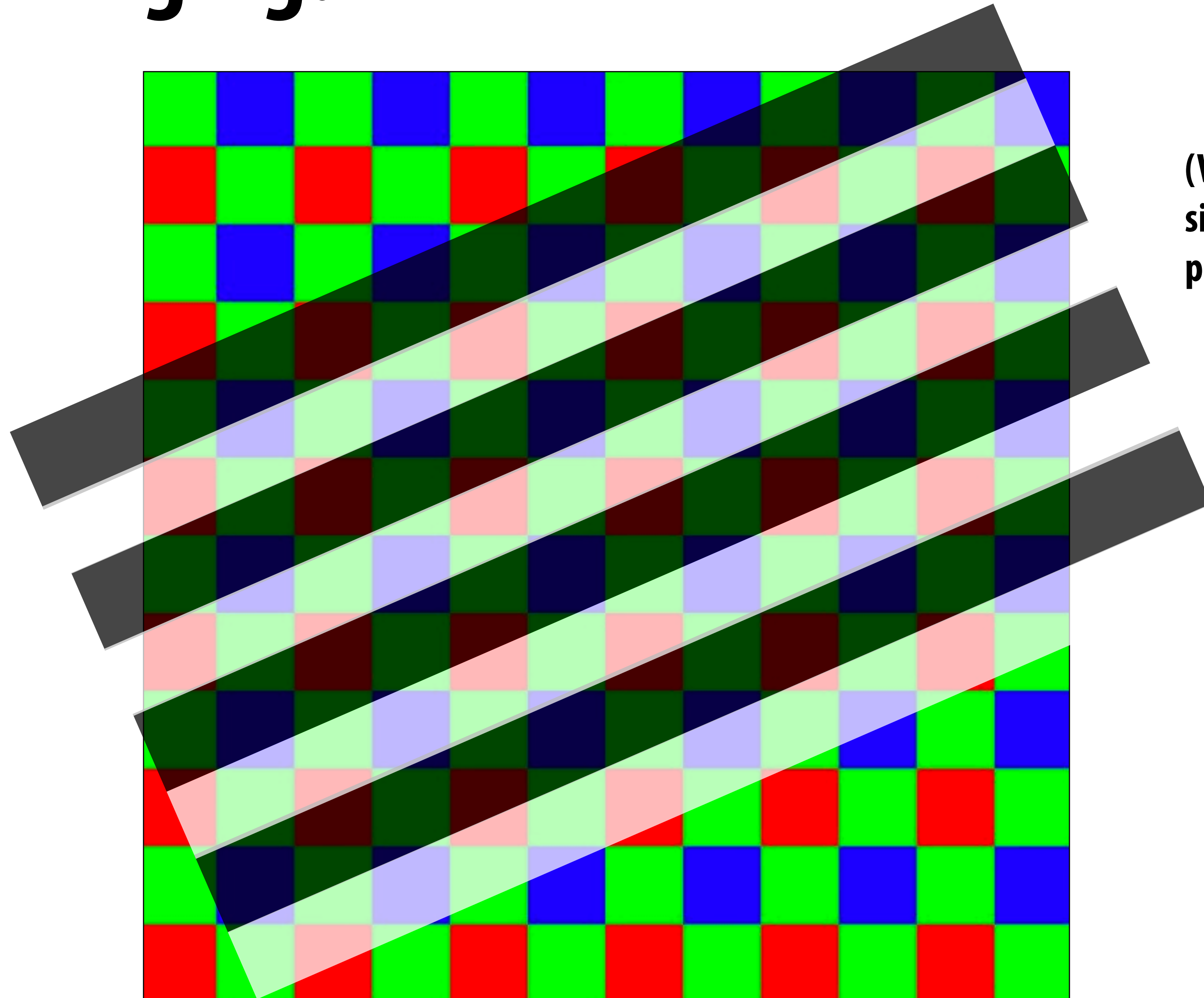
Why color fringing?

What will demosaiced result look like if this black and white signal was captured by the sensor?
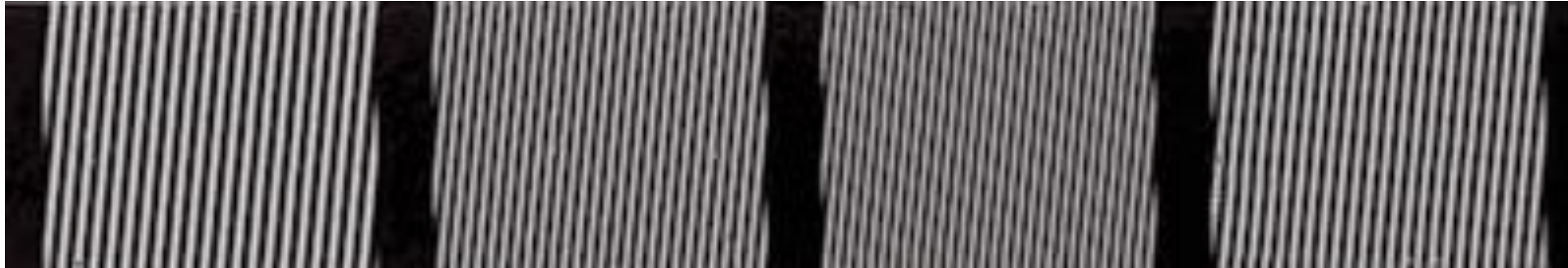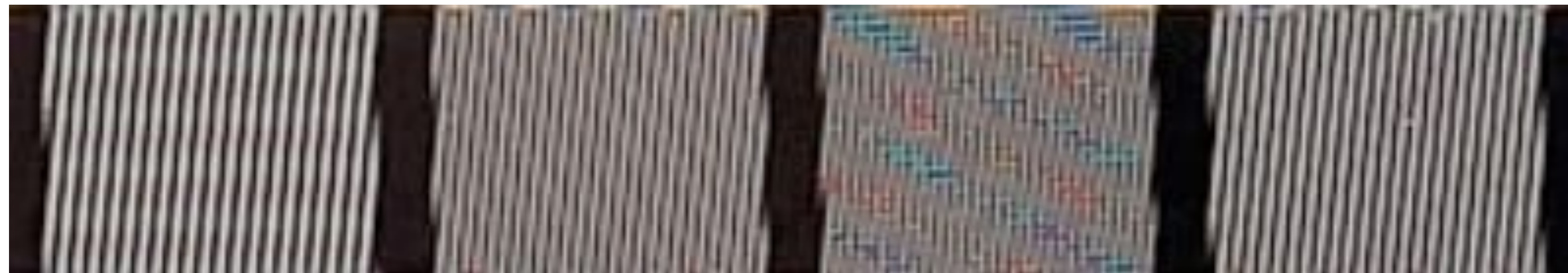
Why color fringing?

(Visualization of signal and Bayer pattern)

Demosaicing errors
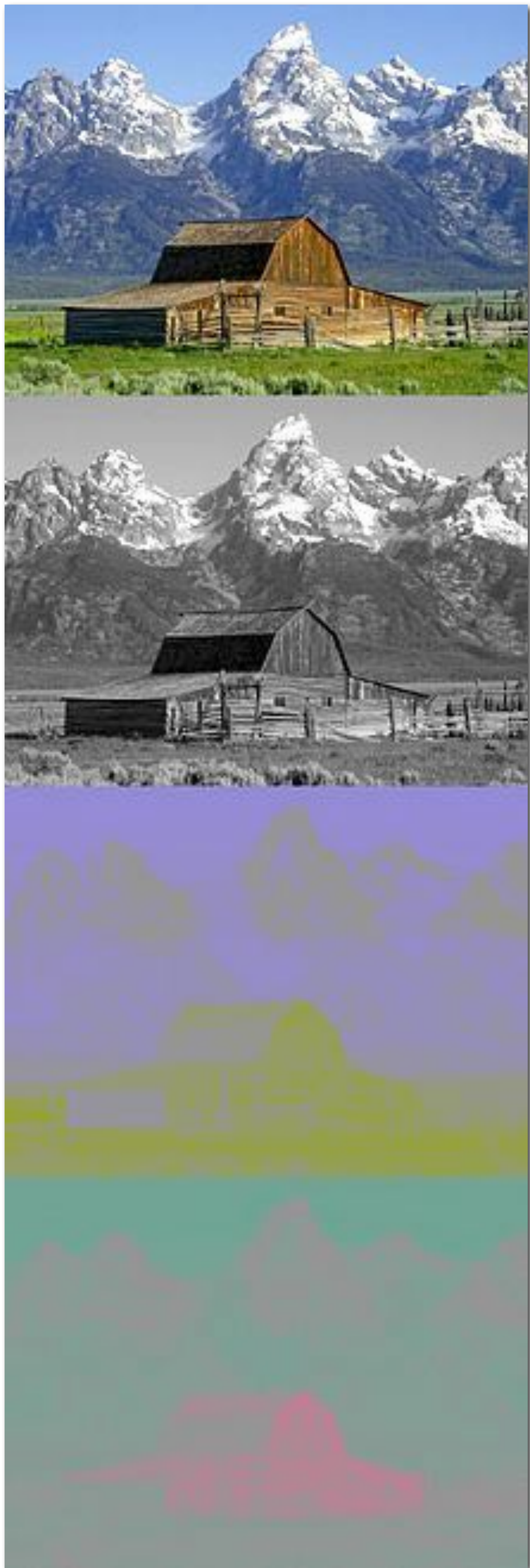
- **Common difficult case: fine diagonal black and white stripes**
- **Result: moire pattern color artifacts**

RAW data from sensor

RGB result after demosaic

Y'CbCr color space

Colors are represented as point in 3-space

RGB is just one possible basis for representing color

Y'CbCr separates luminance from hue in representation
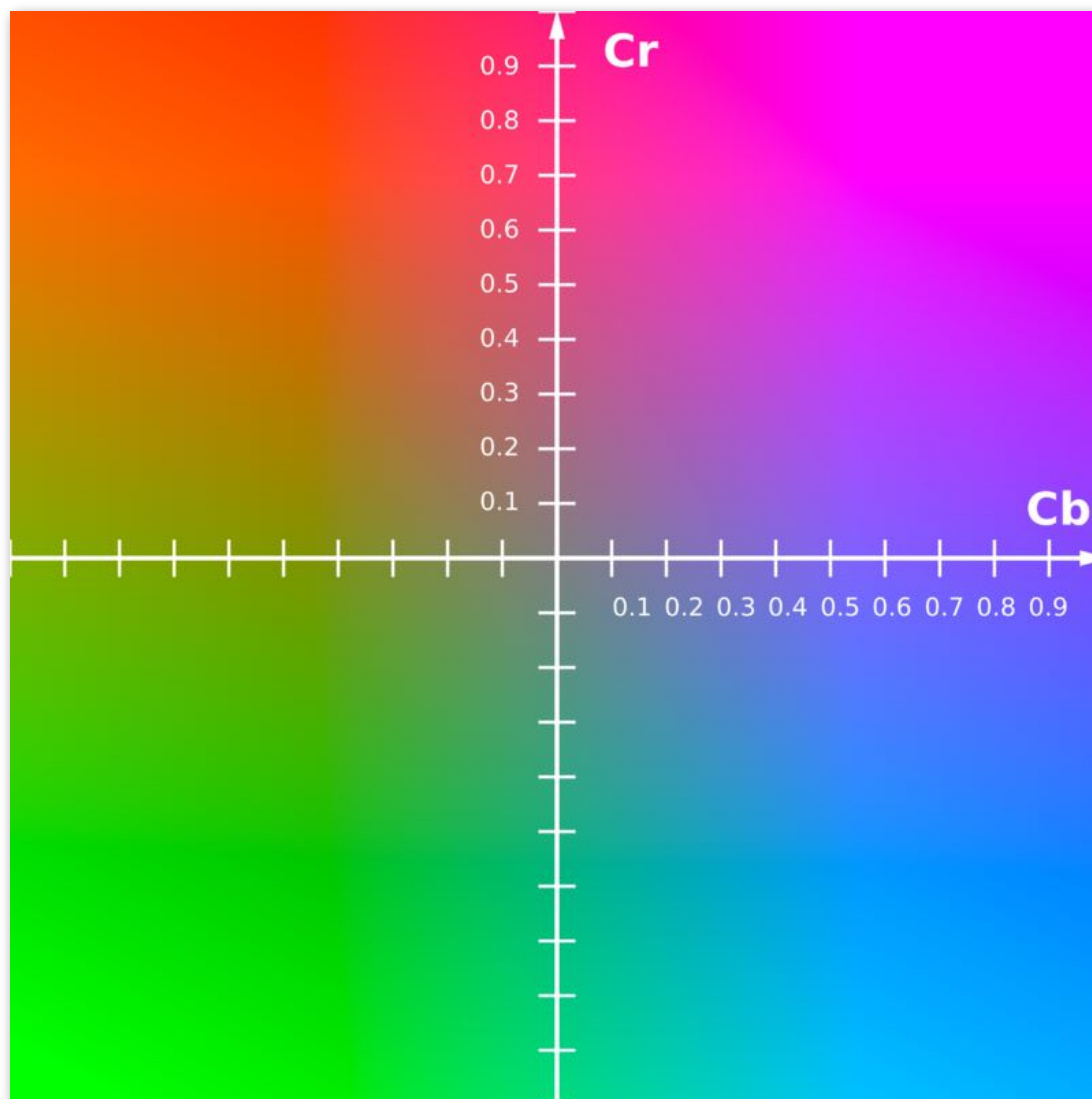
Y' = luma: perceived luminance
Cb = blue-yellow deviation from gray
Cr = red-cyan deviation from gray

"Gamma corrected" RGB
(primed notation indicates perceptual (non-linear) space)
We'll describe what this means this later in the lecture.

Conversion matrix from R'G'B' to Y'CbCr:

$$Y' = 16 + \frac{65.738 \cdot R'_D}{256} + \frac{129.057 \cdot G'_D}{256} + \frac{25.064 \cdot B'_D}{256}$$

$$C_B = 128 + \frac{-37.945 \cdot R'_D}{256} - \frac{74.494 \cdot G'_D}{256} + \frac{112.439 \cdot B'_D}{256}$$

$$C_R = 128 + \frac{112.439 \cdot R'_D}{256} - \frac{94.154 \cdot G'_D}{256} - \frac{18.285 \cdot B'_D}{256}$$

Y'

Cb

Cr

Example: compression in Y'CbCr

Original picture of Kayvon

Example: compression in Y'CbCr

Contents of CbCr color channels downsampled by a factor of 20 in each dimension
(400x reduction in number of samples)

Example: compression in Y'CbCr

Full resolution sampling of luma (Y')

Example: compression in Y'CbCr

Reconstructed result
(looks pretty good)

Better demosaic

- **Convert demosaiced RGB value to YCbCr**

- **Low-pass filter (blur) or median filter CbCr channels**

- **Combine filtered CbCr with full resolution Y from sensor to get RGB**

- **Trades off spatial resolution of chroma information to avoid objectionable color fringing**

White balance

- Adjust relative intensity of rgb values (goal: make neutral tones in scene appear neutral in image)

```
output_pixel = white_balance_coeff * input_pixel
// note: in this example, white_balance_coeff is vec3
// (adjusts ratio of red-blue-green channels)
```

- The same "white" object will generate different sensor response when illuminated by different spectra. Camera needs to infer what the lighting in the scene was.

White balance example

WB : Daylight

Temp 5,500

Tint + 10

White balance example

WB : Fluorescent

Temp   3,800

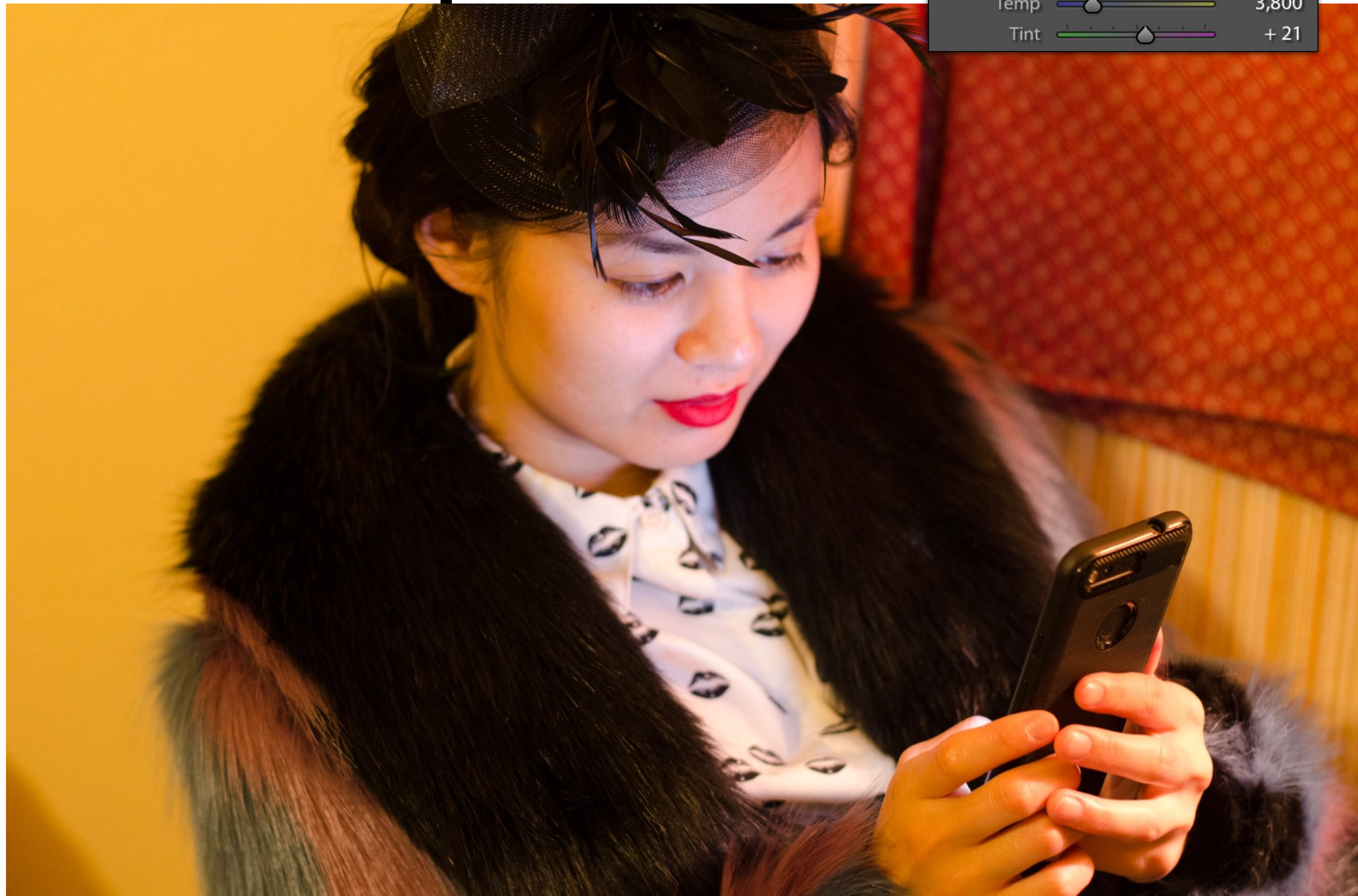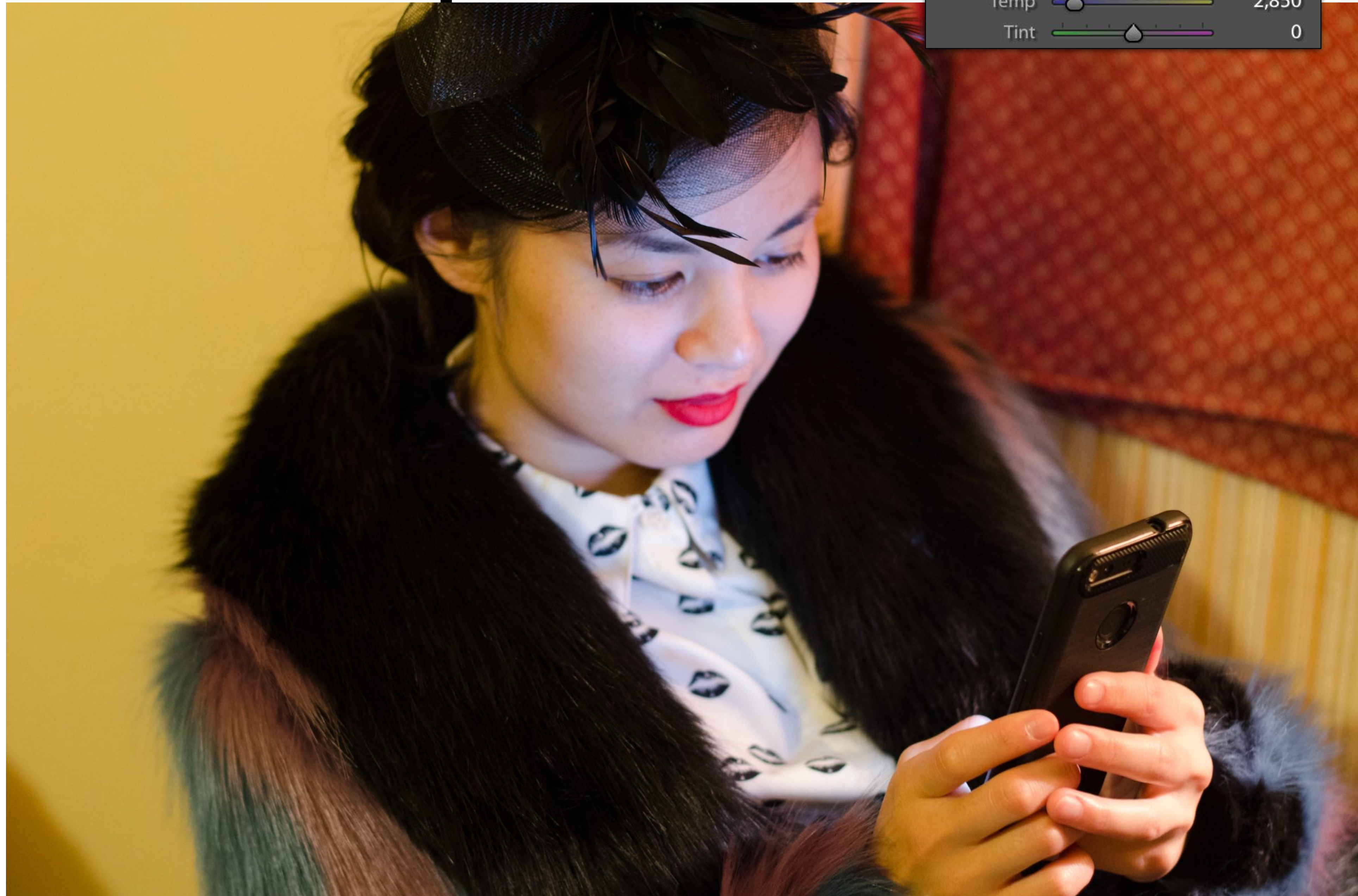Tint   + 21

White balance example

WB : Tungsten

Temp   2,850

Tint   0

White balance algorithms

- **White balance coefficients depend on analysis of image contents**

    - **Calibration based: get value of pixel of "white" object: (r_w, g_w, b_w)**
        - **Scale all pixels by ($1/r_w$, $1/g_w$, $1/b_w$)**
    - **Heuristic based: camera must guesse which pixels correspond to white objects in scene**
        - **Gray world assumption: make average of all pixels in image gray**
        - **Brightest pixel assumption: find brightest region of image, make it white ([1,1,1])**

- **Modern white-balance algorithms are based on learning correct scaling from examples**

    - **Create database of images for which good white balance settings are known (e.g., manually set by human)**
    - **Learning mapping from image features to white balance settings**
    - **When new photo is taken, use learned model to predict good white balance settings**

Scale r,g,b values so these pixels are close to (1,1,1)

Denoising

Original
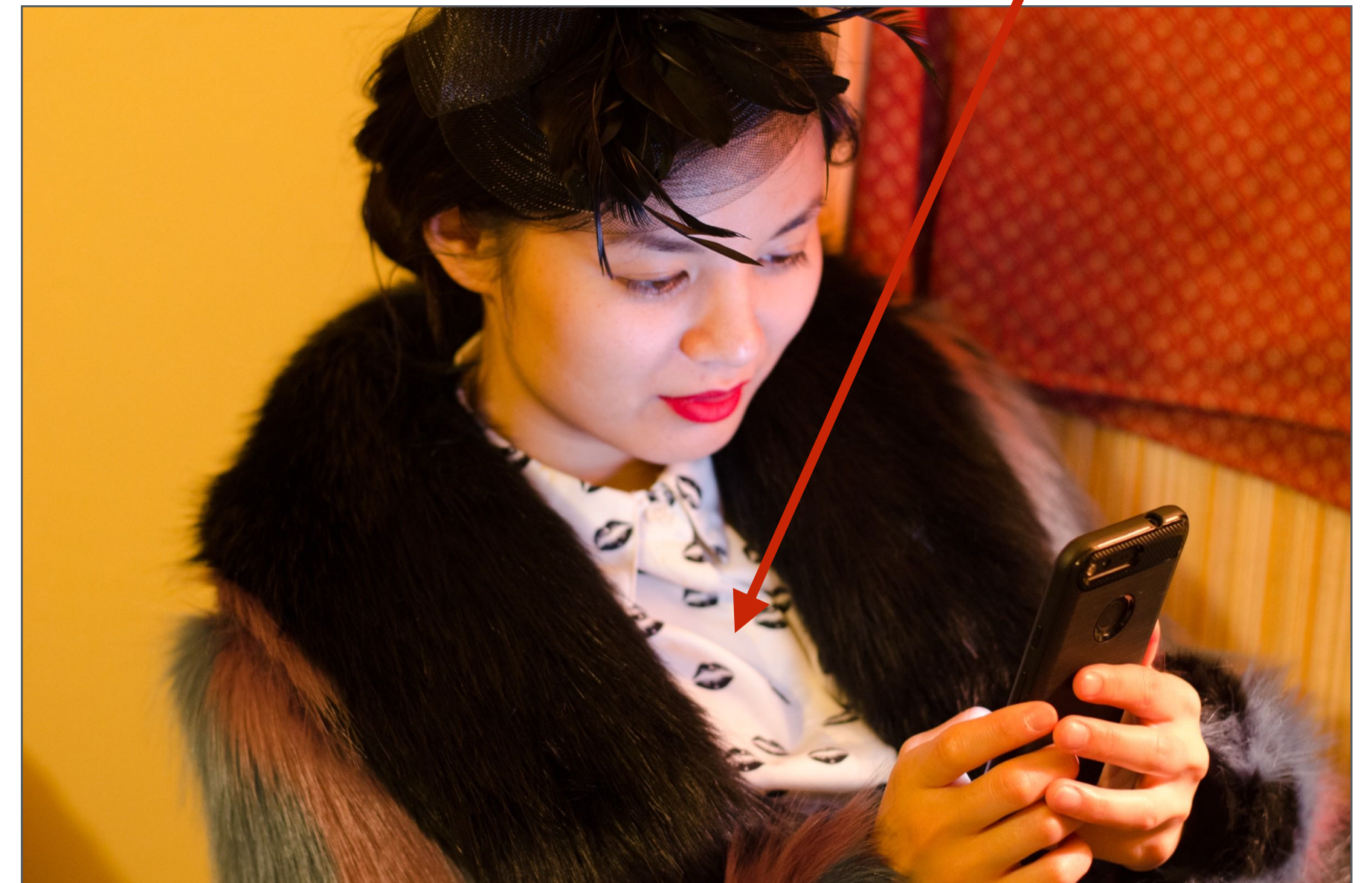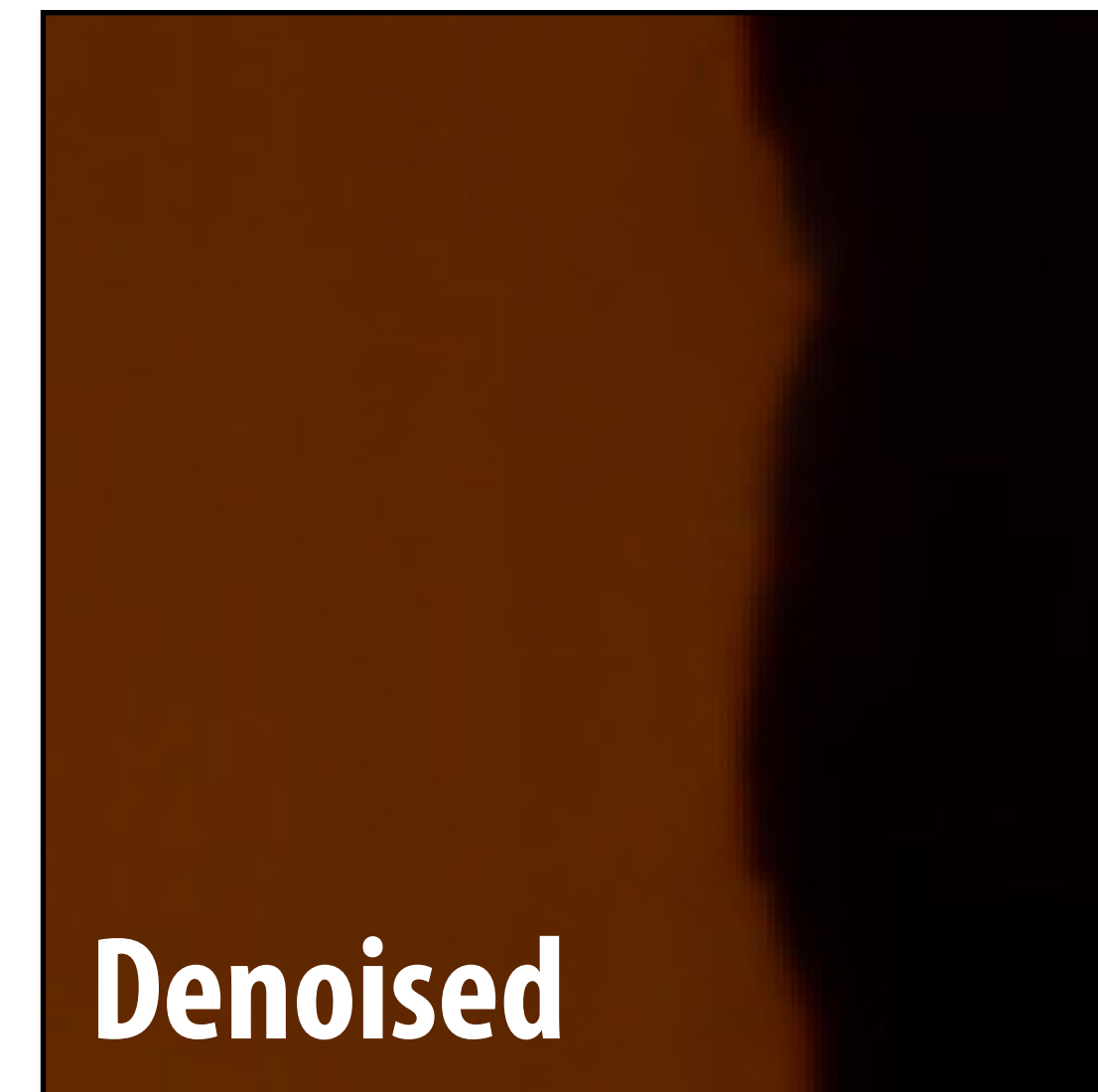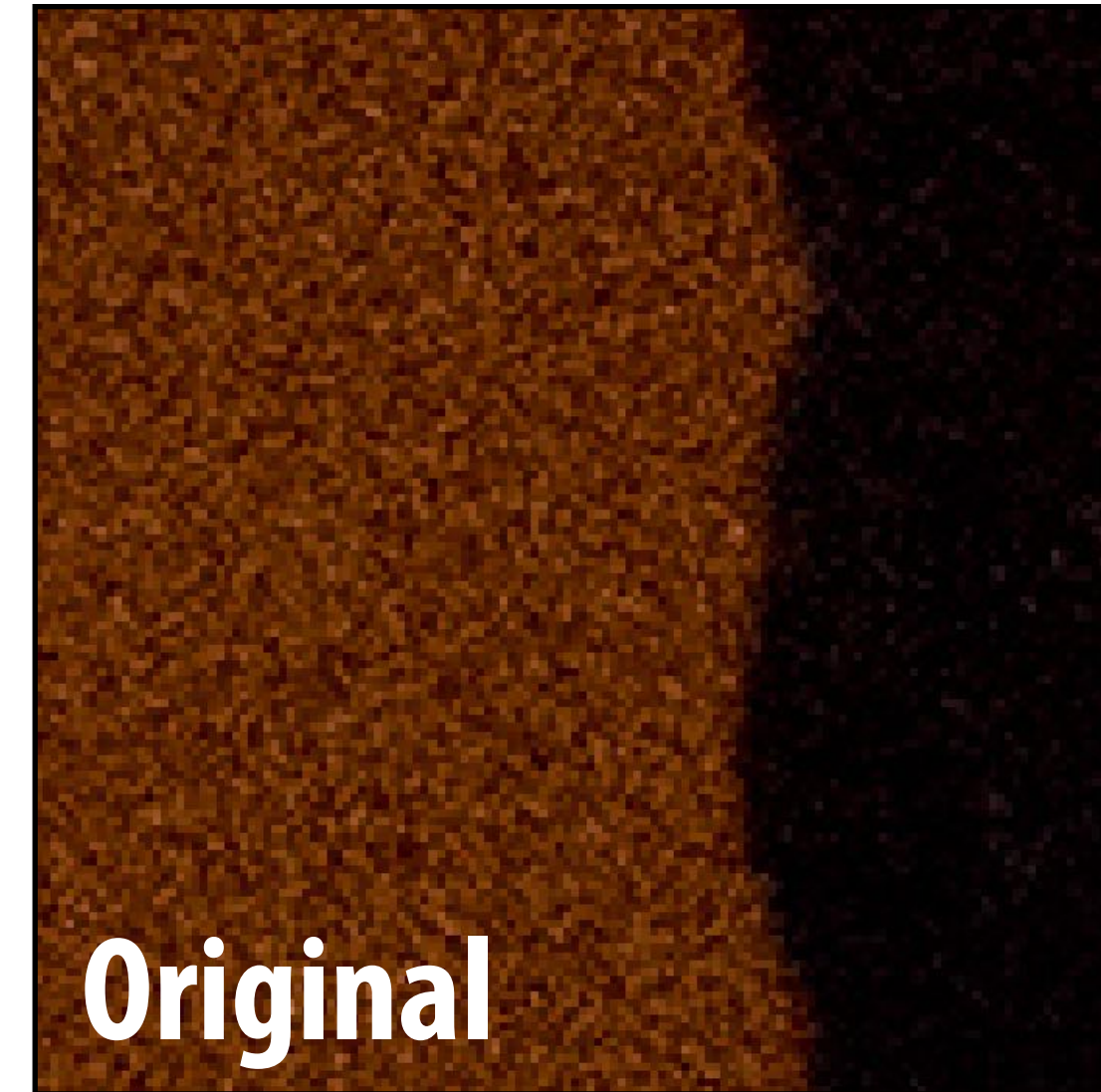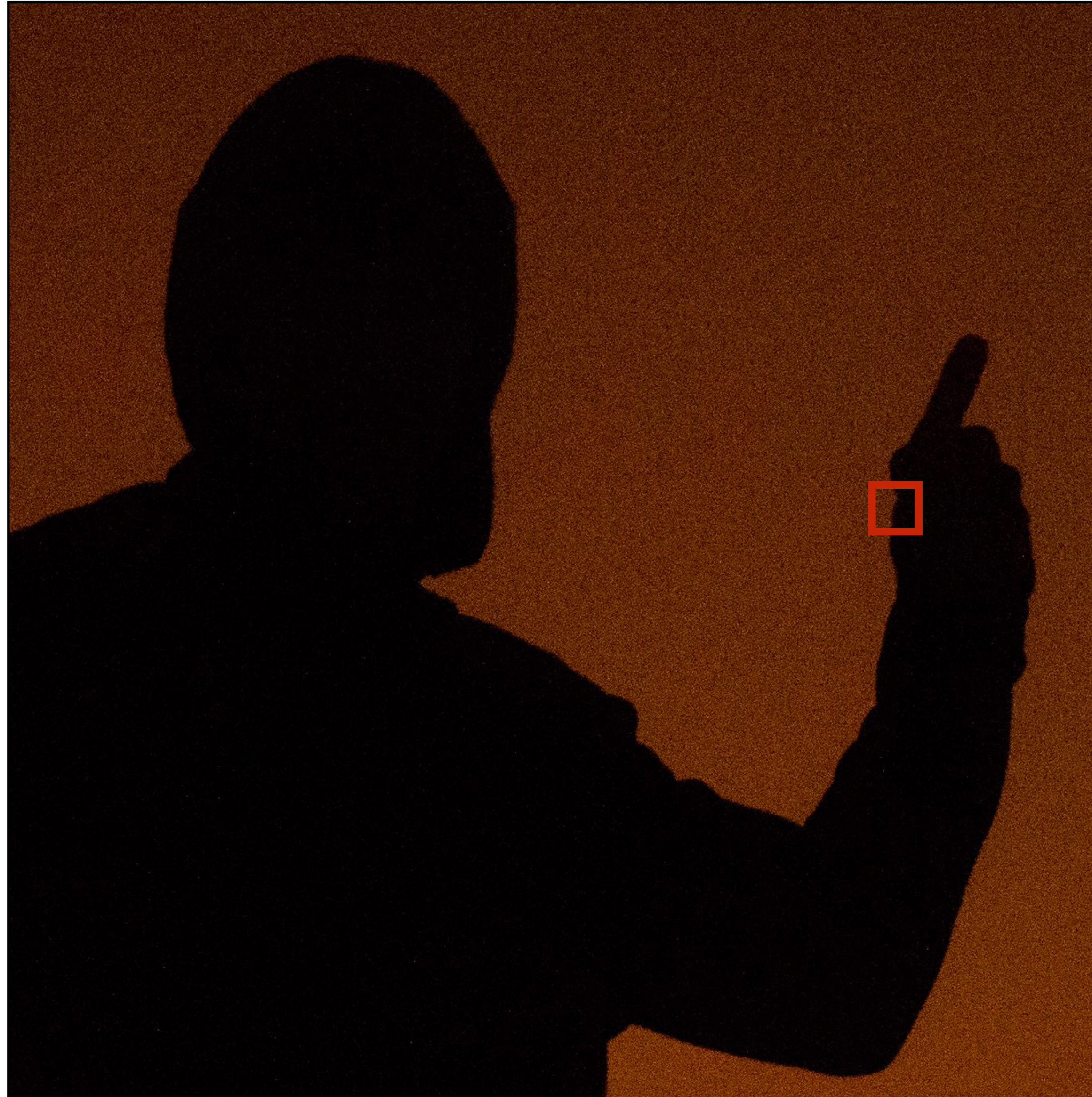
Denoised

Low light conditions need long exposure…
blur due to camera shake

Low light photo: many regions underexposed (short exposure) to avoid blur + some regions overexposed
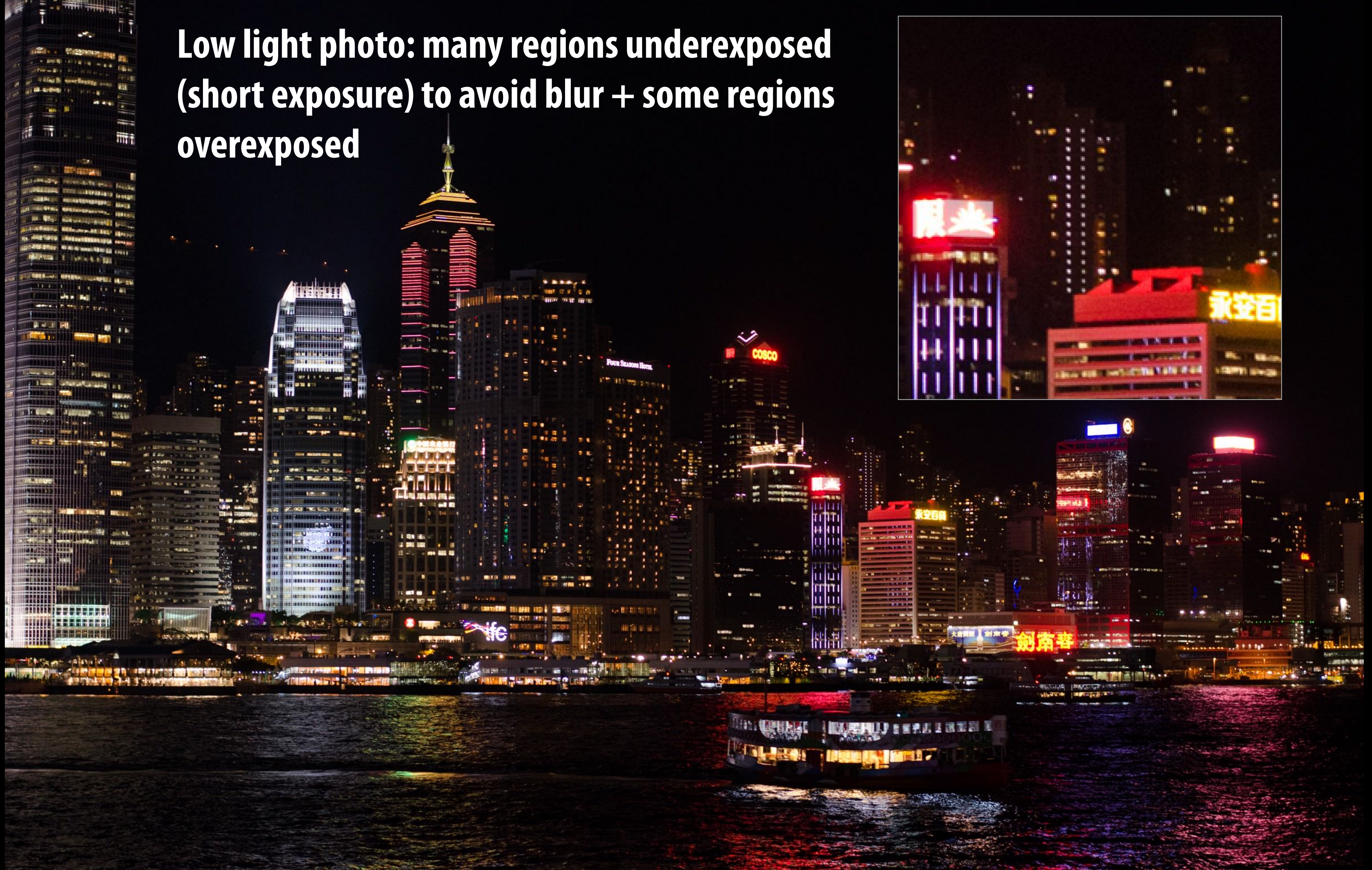
Brightened image to see detail in dark regions, notice noise in dark regions

Attempt to denoise… splotchy effect remains

Long exposure: walking people are blurred…

Long exposure: walking people are blurred…

Also: still significant noise in dark regions

Reduce noise via image processing: denoising via downsampling
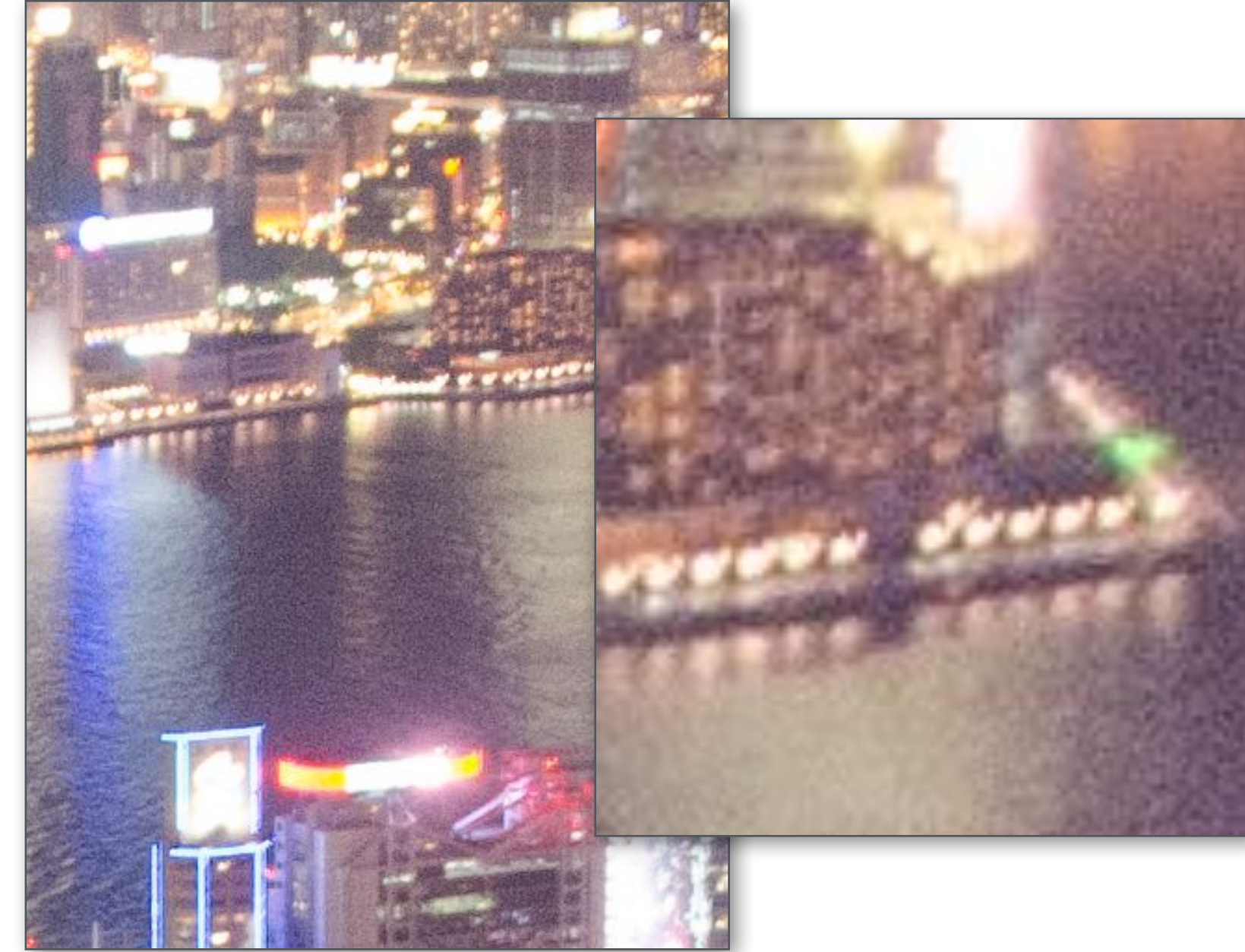
Downsample via
point sampling

(noise remains)

Downsample via averaging

Noise reduced

Like a smaller number of
bigger pixels!

Discrete 2D convolution

$$(f * I)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i,j)I(x - i, y - j)$$

output image
(the result of convolving f with input image I)

filter

input image

Consider a $f(i,j)$ **that is nonzero only when:** $-1 \leq i, j \leq 1$

Then:

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i,j)I(x - i, y - j)$$

And we can represent f(i,j) as a 3x3 matrix of values where:

$$f(i,j) = \mathbf{F}_{i,j}$$ **(often called: "filter weights", "filter kernel")**

Simple 3x3 box blur in C code

```
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH * HEIGHT];


float weights[] = {1./9, 1./9, 1./9,
                   1./9, 1./9, 1./9,
                   1./9, 1./9, 1./9};


for (int j=0; j<HEIGHT; j++) {
   for (int i=0; i<WIDTH; i++) {
      float tmp = 0.f;
      for (int jj=0; jj<3; jj++)
         for (int ii=0; ii<3; ii++)
            tmp += input[(j+jj)*(WIDTH+2) + (i+ii)] * weights[jj*3 + ii];
      output[j*WIDTH + i] = tmp;
   }
}
```

For now: ignore boundary pixels and assume output image is smaller than input (makes convolution loop bounds much simpler to write)

7x7 box blur

Original

Blurred
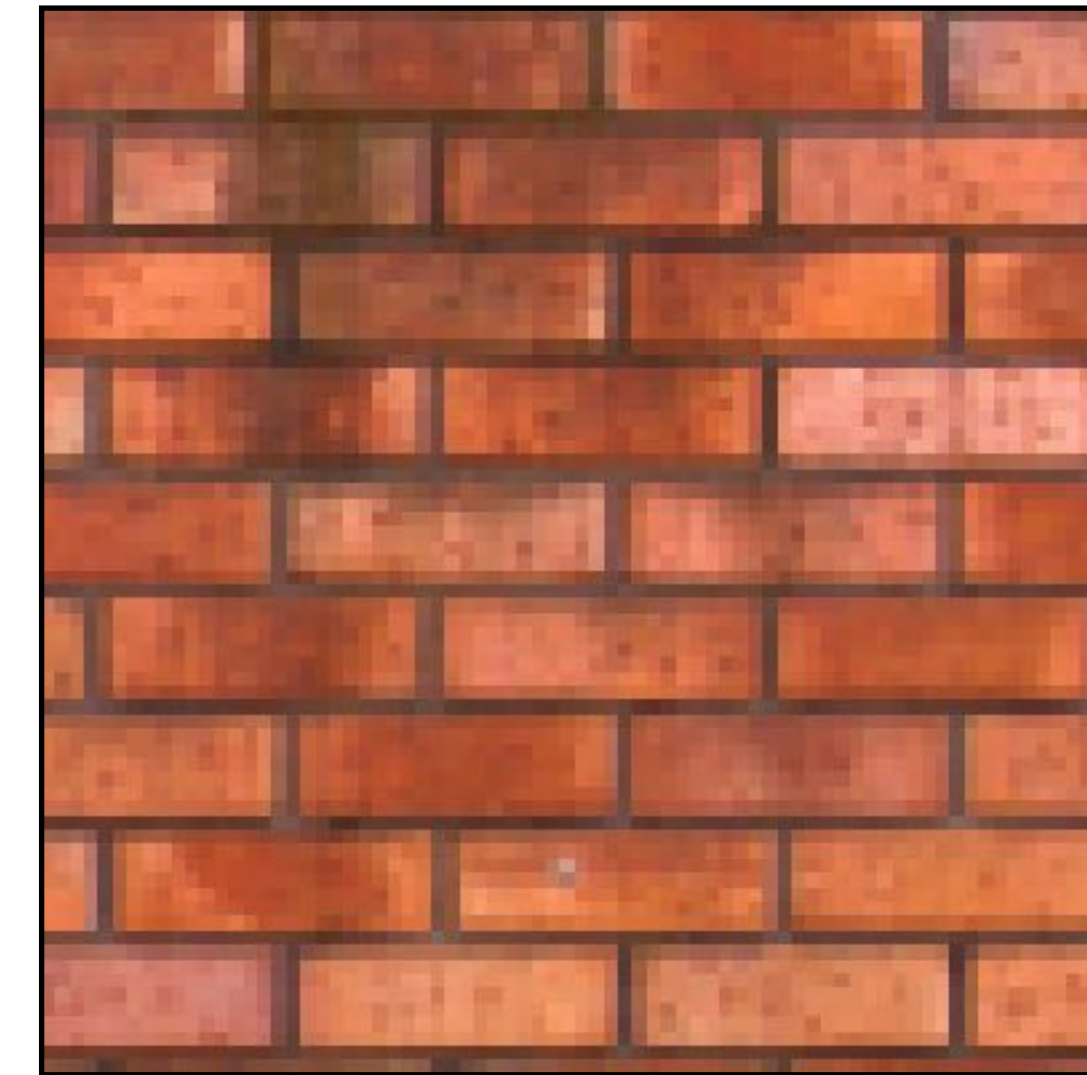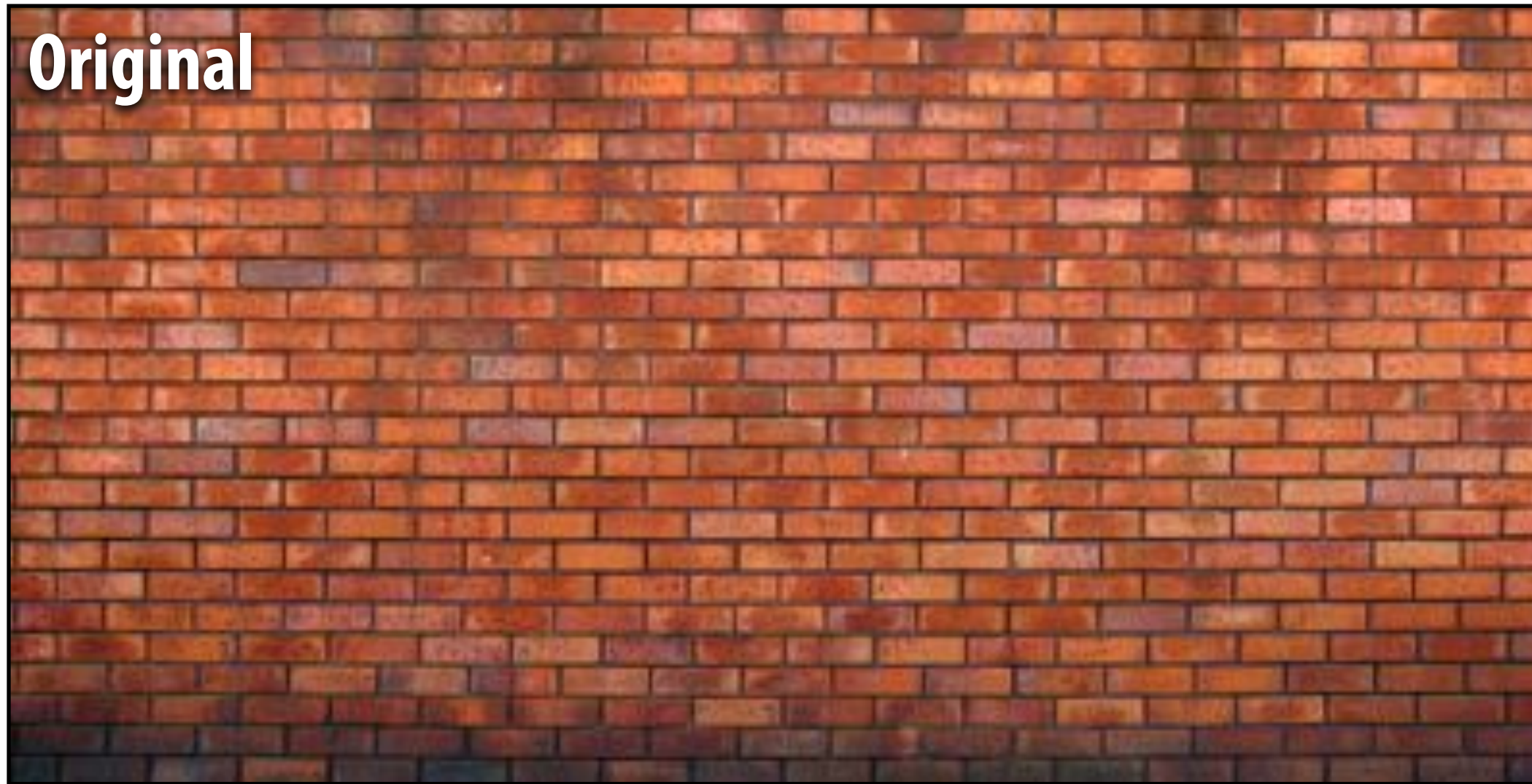
Gaussian blur

■ **Obtain filter coefficients from sampling 2D Gaussian**

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

■ **Produces weighted sum of neighboring pixels (contribution falls off with distance)**

— **In practice: truncate filter beyond certain distance for efficiency**

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Note: this is a 5x5 truncated Gaussian filter

7x7 gaussian blur

Original

Blurred
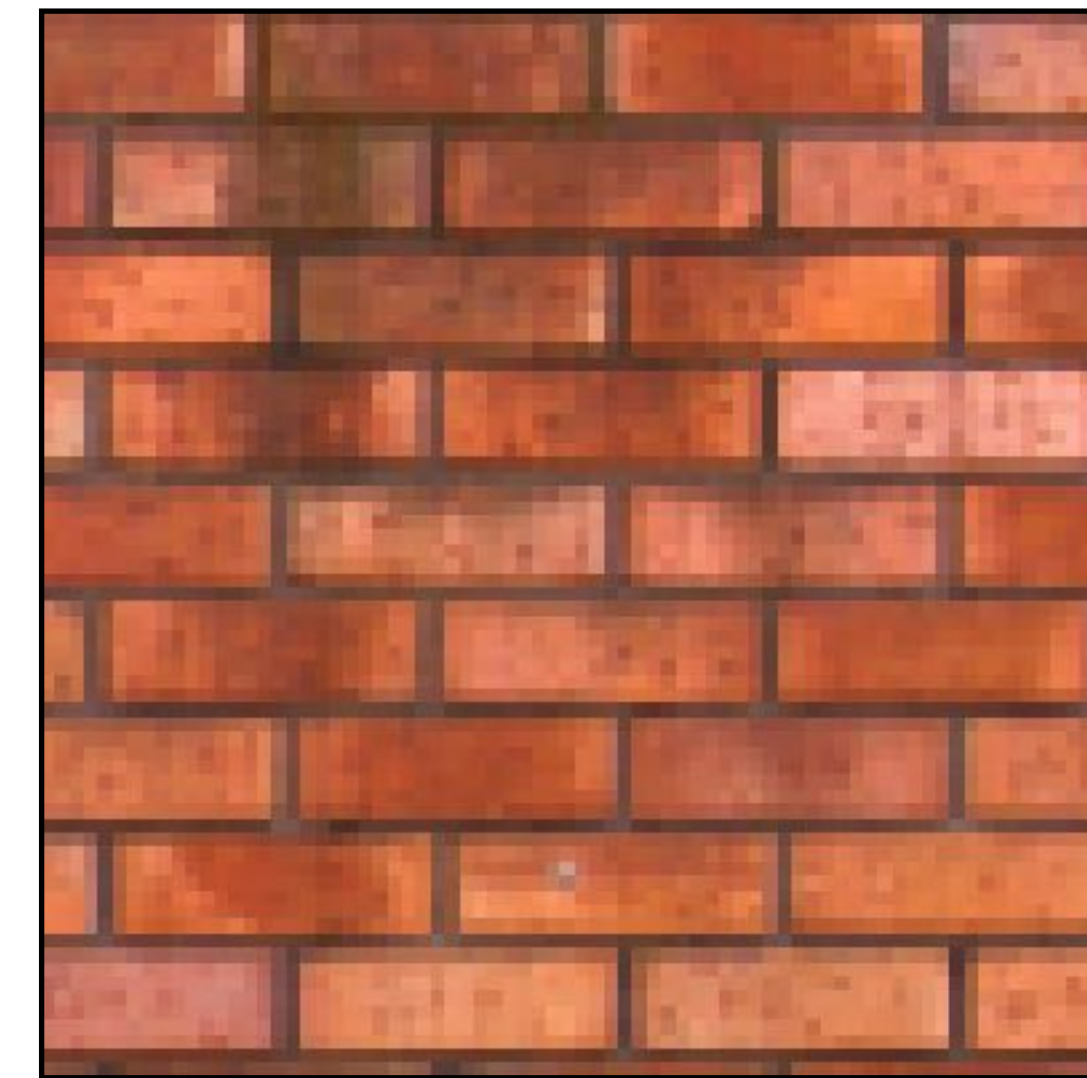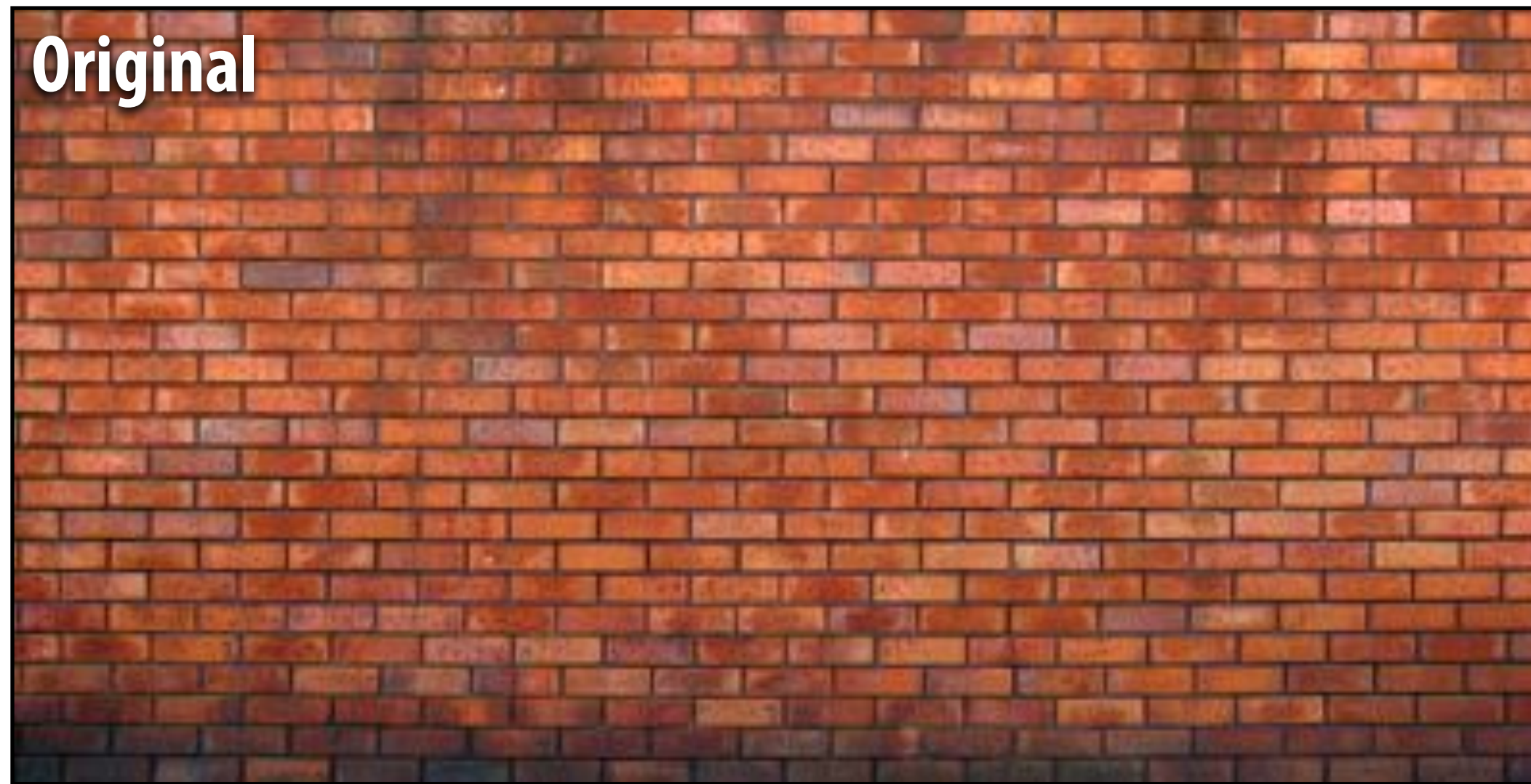
3x3 sharpen filter

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Original

Sharpened
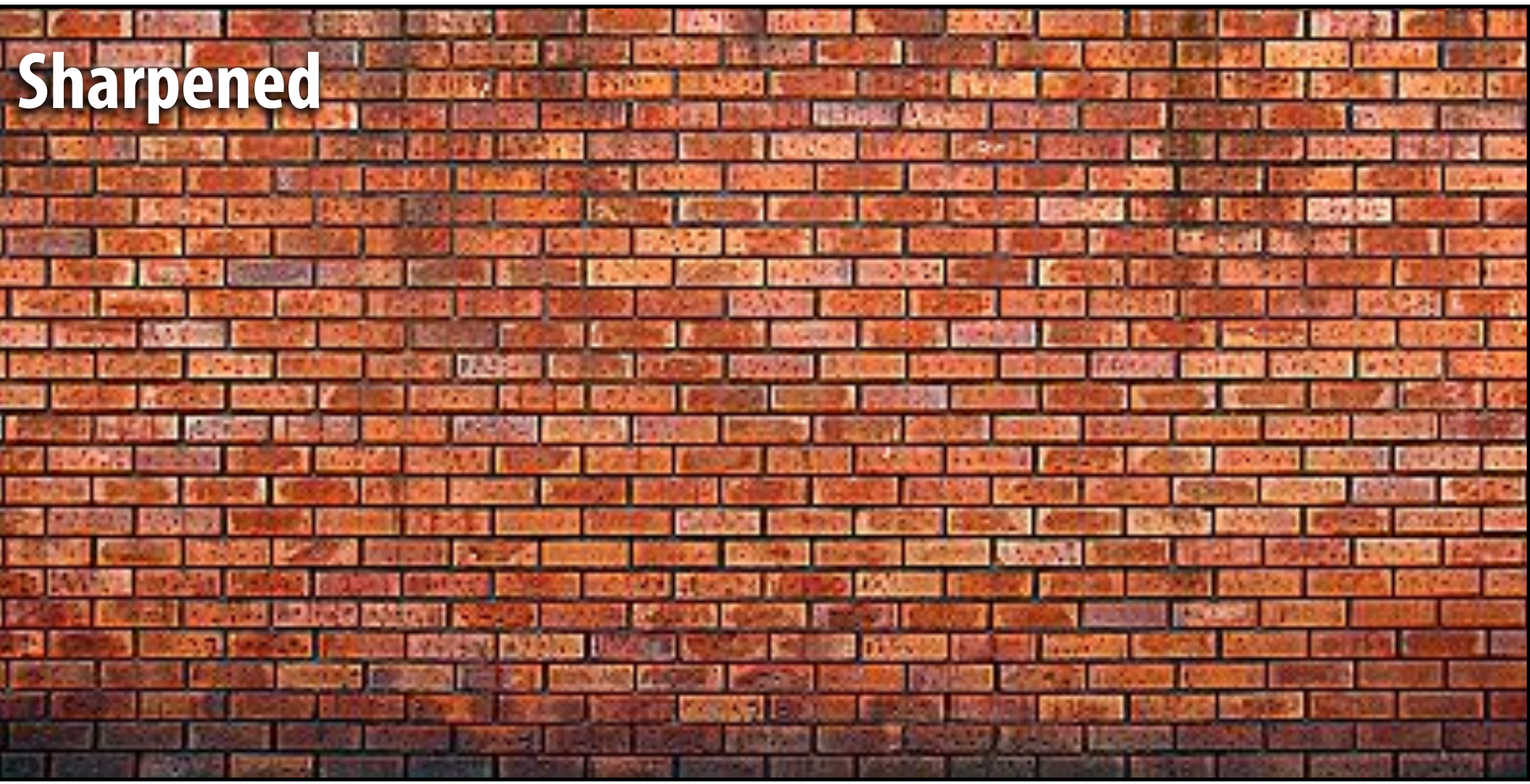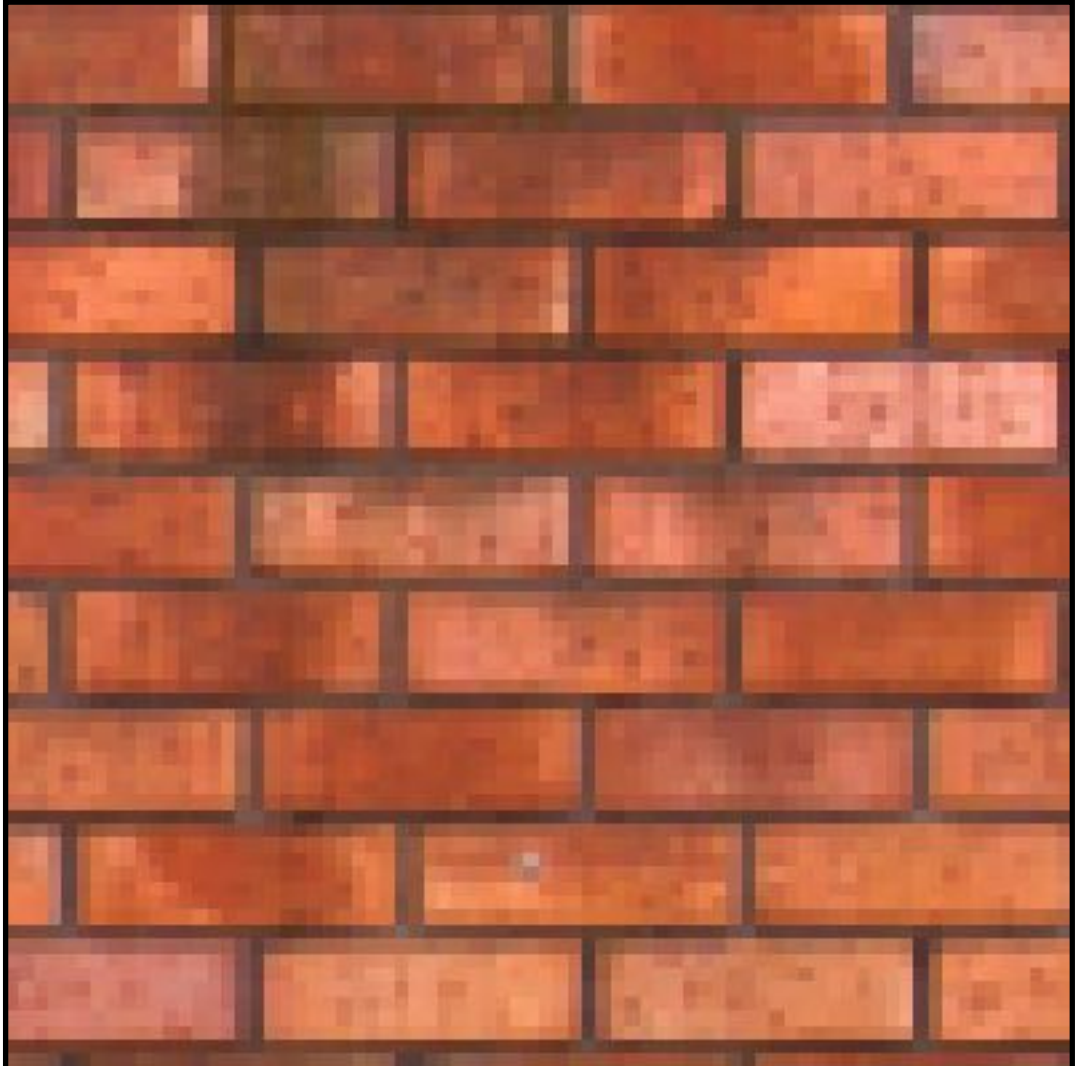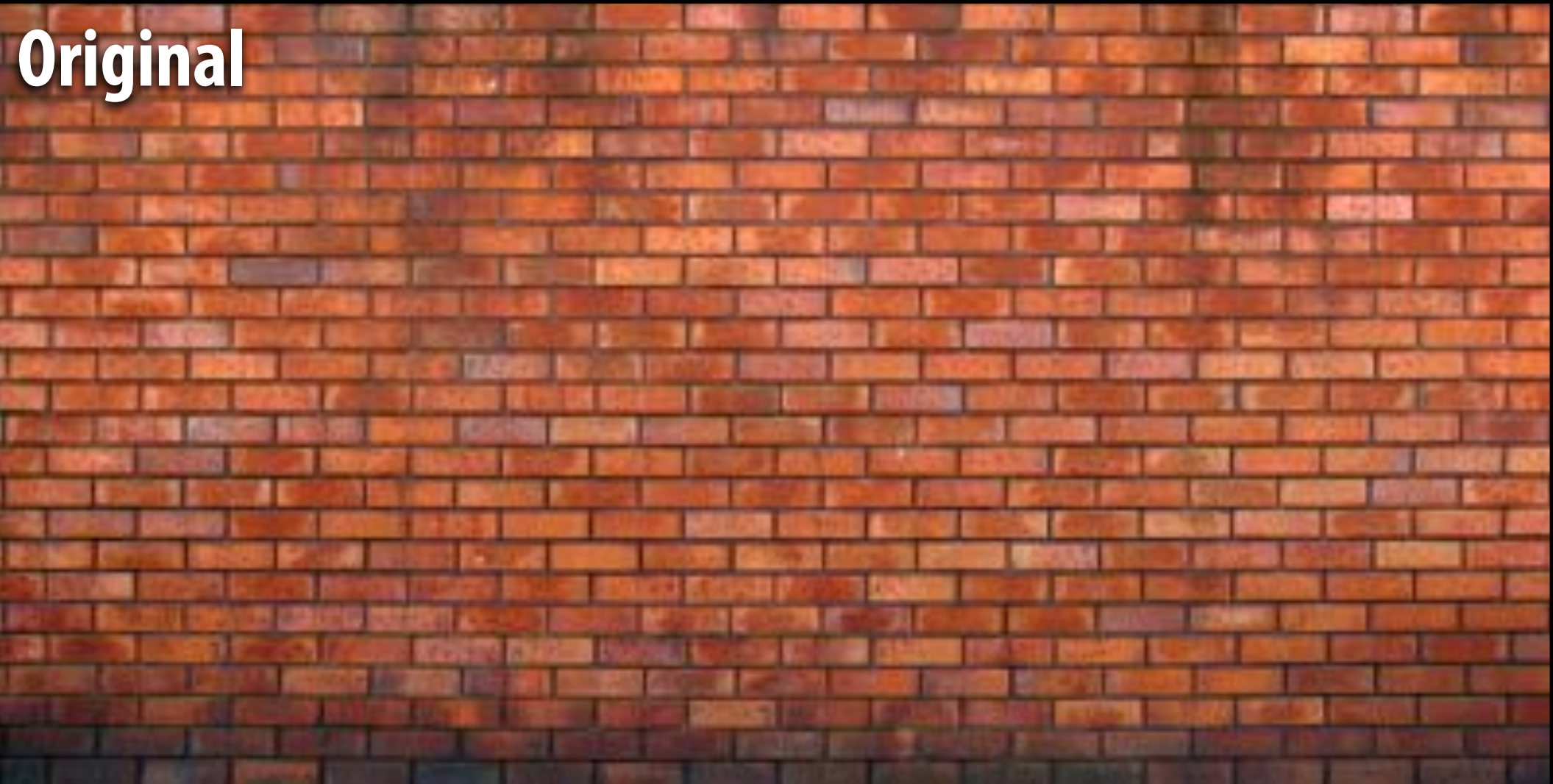
Median filter

- **Replace pixel with median of its neighbors**
  - Useful noise reduction filter: unlike gaussian blur, one bright pixel doesn't drag up the average for entire region

- **Not linear: filter weights are 1 or 0 (depending on image content)**

```
uint8 input[(WIDTH+2) * (HEIGHT+2)];
uint8 output[WIDTH * HEIGHT];
for (int j=0; j<HEIGHT; j++) {
   for (int i=0; i<WIDTH; i++) {
       output[j*WIDTH + i] =
              // compute median of pixels
              // in surrounding 5x5 pixel window
   }
}
```
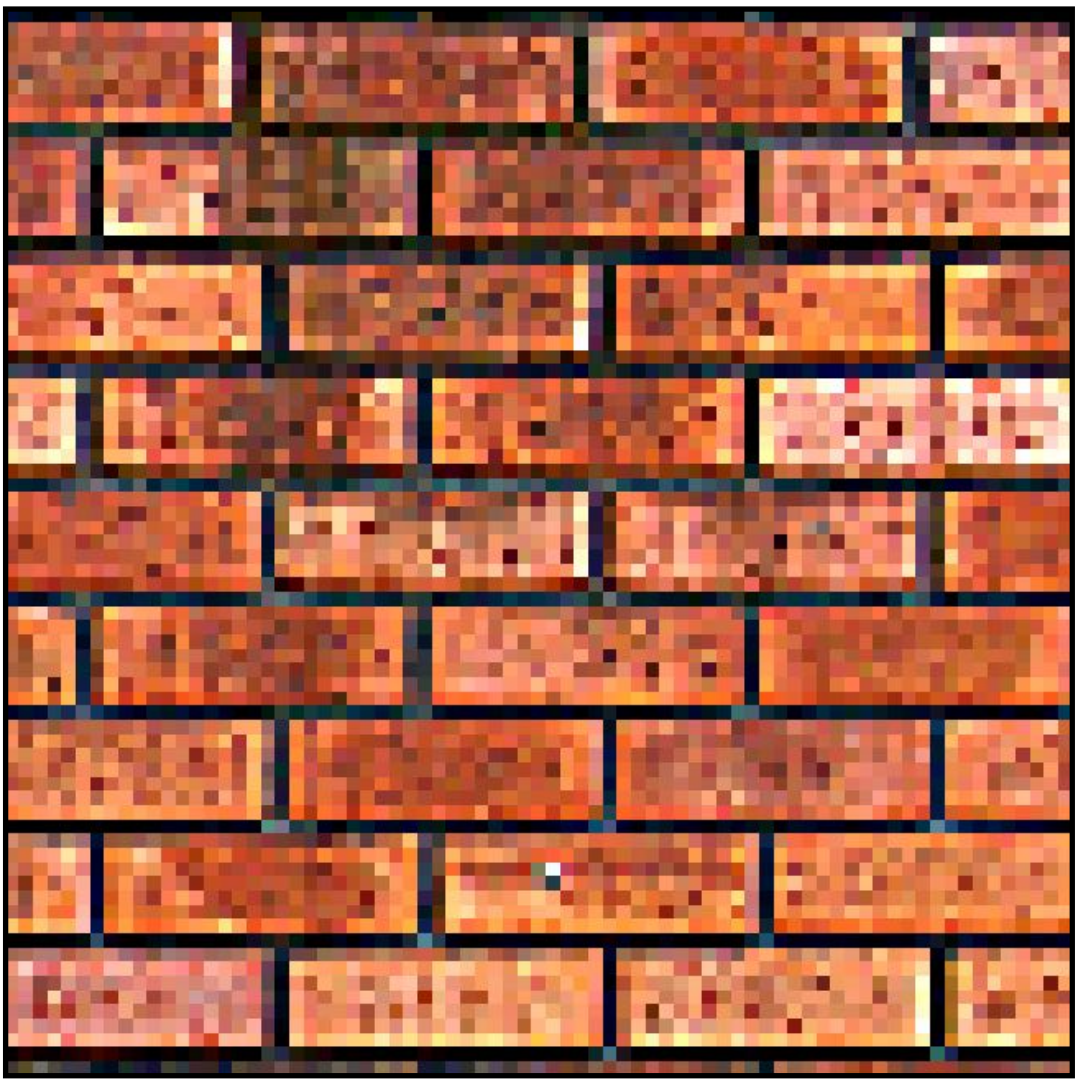

original image

1px median filter

3px median filter

10px median filter

- **Basic algorithm for NxN support region:**
  - Sort N^2 elements in support region, then pick median: $O(N^2\log(N^2))$ work per pixel
  - Can you think of an $O(N^2)$ algorithm? What about $O(N)$?

Bilateral filter

Original

Processed

Example use of bilateral filter: removing noise while preserving image edges

Bilateral filter

$$\text{BF}[I](p) = \frac{1}{W_p} \sum_{i,j} f(|I(x-i, y-j) - I(x,y)|) G_\sigma(i,j) I(x-i, y-j)$$

Normalization

For all pixels in support region of Gaussian kernel

Re-weight based on difference in input image pixel values

$$W_p = \sum_{i,j} f(|I(x-i, y-j) - I(x,y)|) G_\sigma(i,j)$$

- **The bilateral filter is an "edge preserving" filter: down-weight contribution of pixels on the "other side" of strong edges.** $f(x)$ **defines what "strong edge means"**

- **Spatial distance weight term** $f(x)$ **could itself be a gaussian**
  - **Or very simple:** $f(x) = 0$ if $x > threshold$, $1\ otherwise$

Value of output pixel (x,y) is the weighted sum of all pixels in the support region of a truncated gaussian kernel

But weight is combination of spatial distance and input image pixel intensity difference. (non-linear filter: like the median filter, the filter's weights depend on input image content)

Bilateral filter: kernel depends on image content
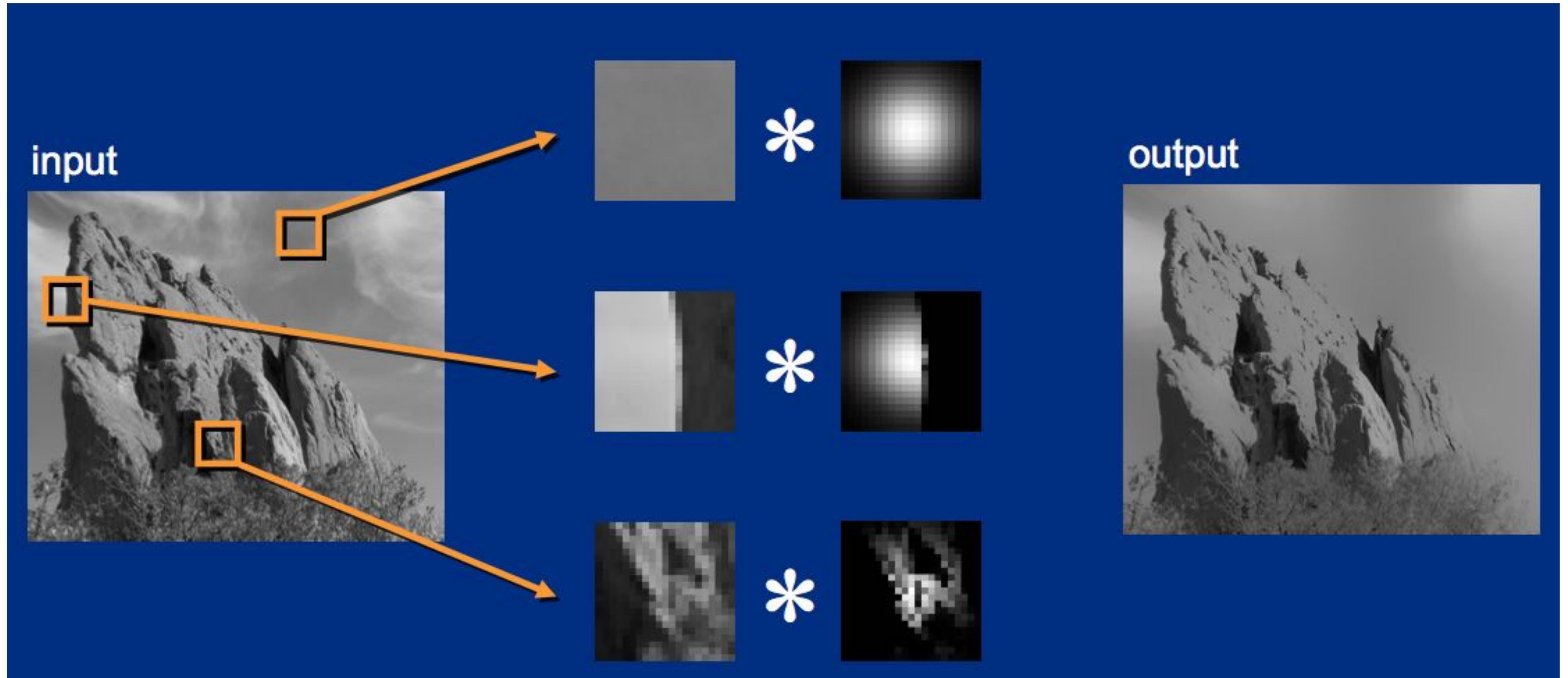
See Paris et al. [ECCV 2006] for a fast approximation to the bilateral filter

Bilateral filter

■ **Visualization of bilateral filter**

Input pixel *p*

Pixels with significantly different intensity as *p* contribute little to filtered result (they are "on the "other side of the edge"

Input image

G(): gaussian about input pixel *p*

f(): Influence of support region

G x f: filter weights for pixel *p*

Filtered output image
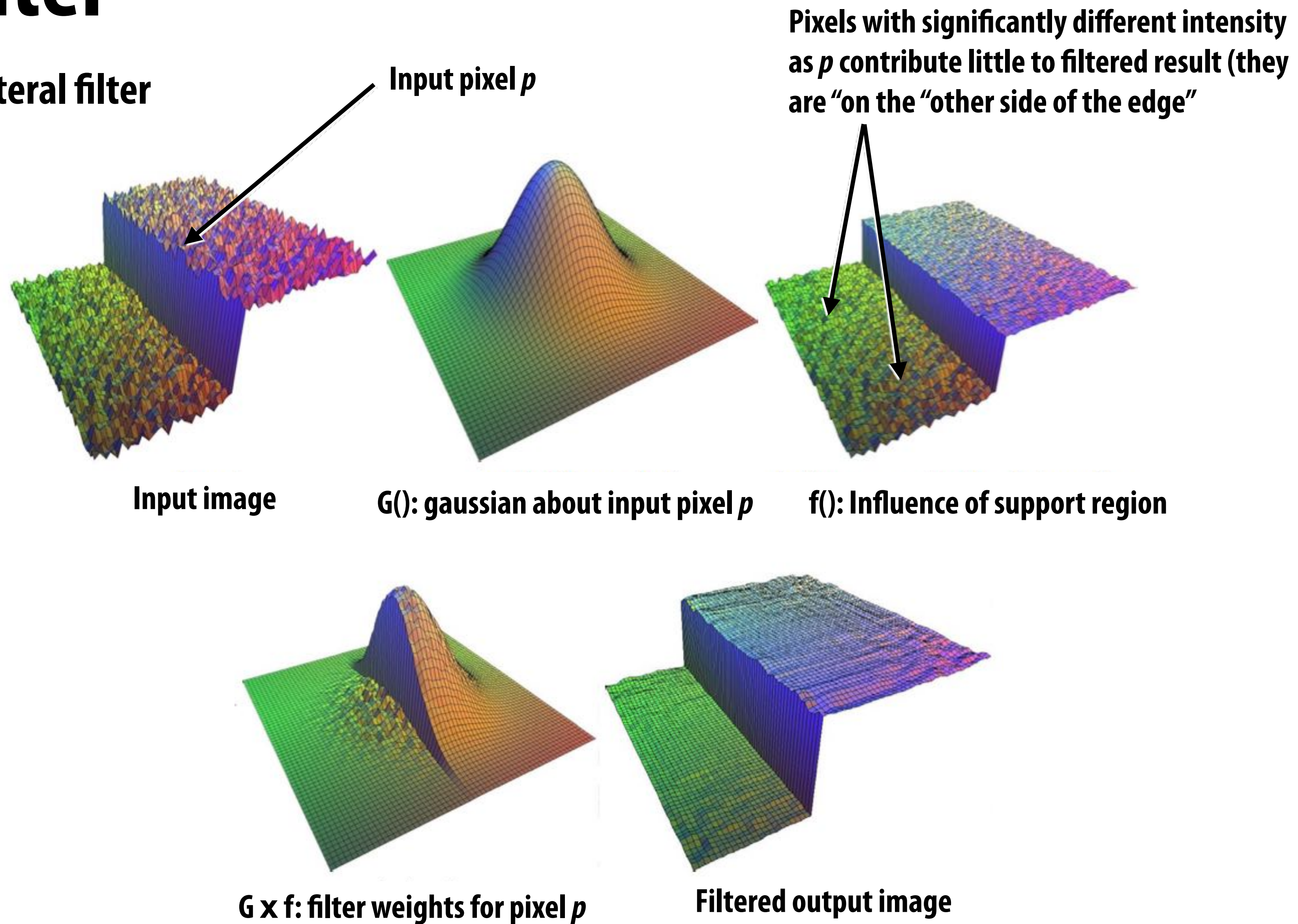
Better denoising idea: merge sequence of captures

Algorithm used in Google Pixel Phones [Hasinoff 16]

- **Long exposure: reduces noise (acquires more light), but introduces blur (camera shake or scene movement)**

- **Short exposure: sharper image, but lower signal/noise ratio**

- **Idea: take sequence of short full-resolution exposures, but align images in software, then merge them into a single sharp image with high signal to noise ratio**
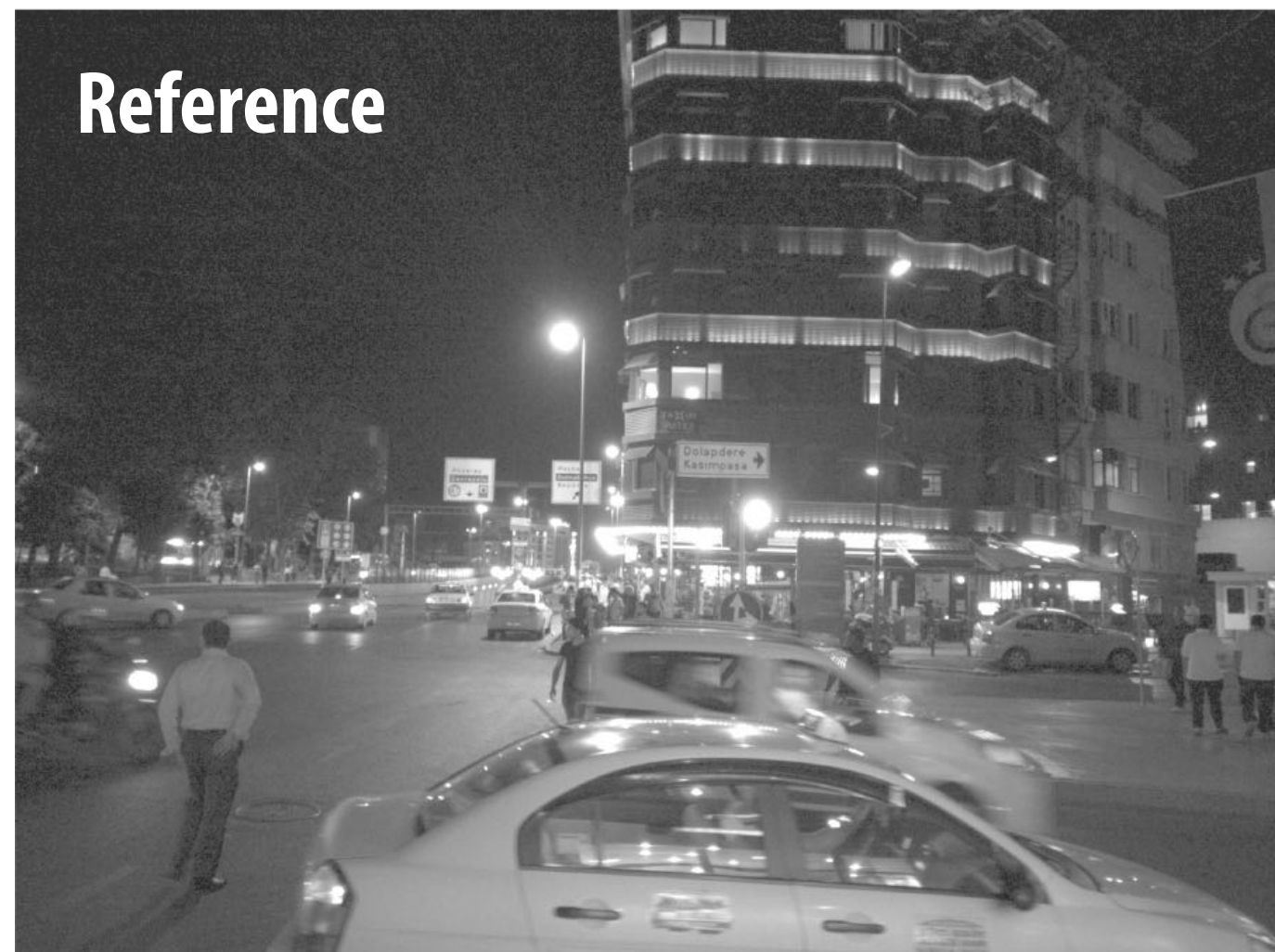
after
shutter
press

burst of raw frames

full-resolution
align & merge

Align and merge algorithm

Image pair

Reference

Frame to align

- **For each image in burst, align to reference frame (use sharpest photo as reference frame)**
  - **Compute optical flow field aligning image pair**
- **Simple merge algorithm: warp images according to flow, and sum**
- **More sophisticated techniques only merge pixels where confidence in alignment is high (tolerate noisy reference pixels when alignment fails)**

Visualization of flow

[Image credit: Hasinoff 16]
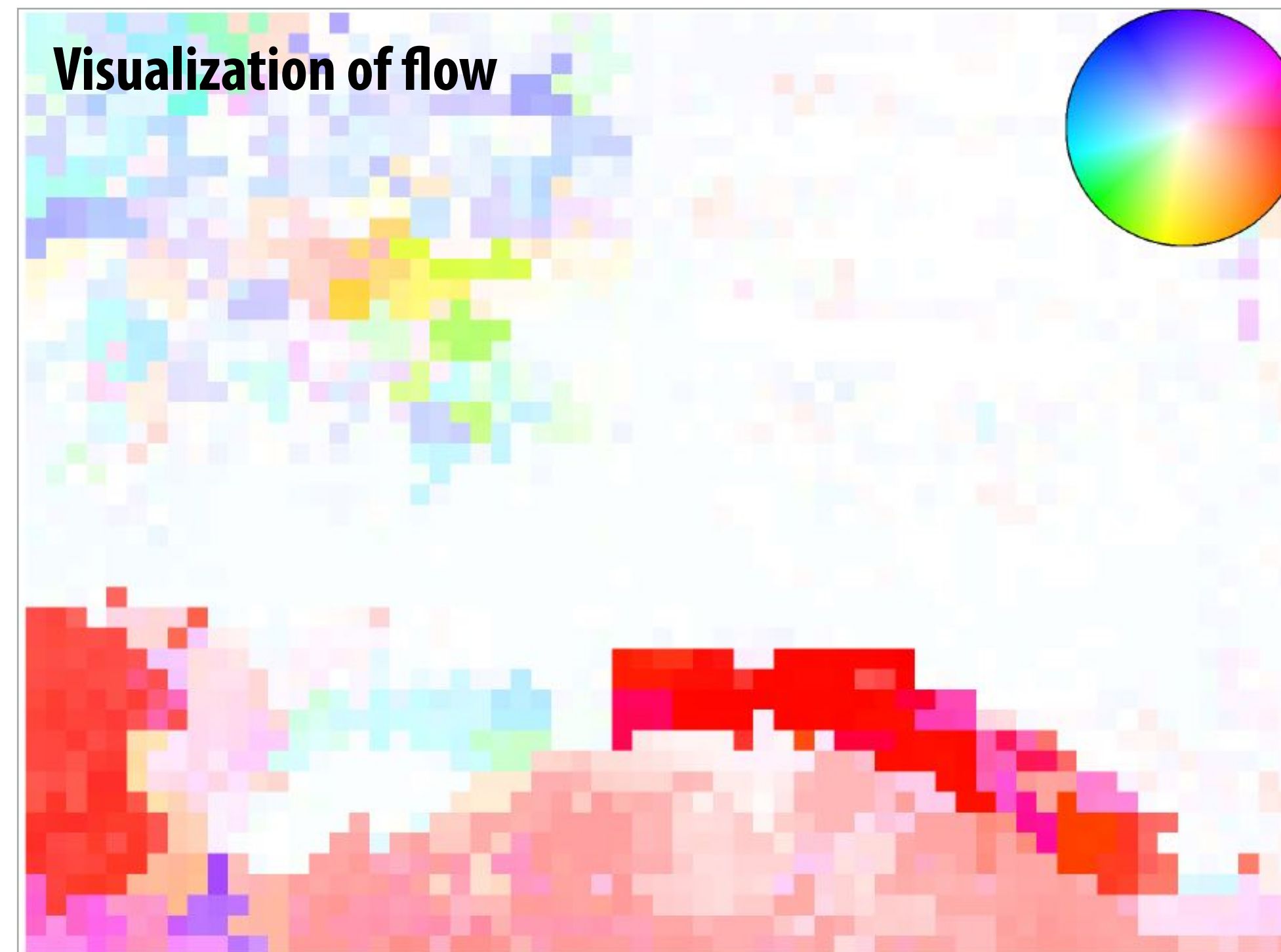
Results of align and merge

| | Reference frame | Temporal mean of images in burst (blurry) | Temporal mean with alignment | Robust merge with alignment |

Full image

Successful alignment

Alignment failure

Summary: simplified image processing pipeline

■ **Correct pixel defects**

■ **Align and merge (to create high signal to noise ration RAW image)**

■ **Correct for sensor bias (using measurements of optically black pixels)**

■ **Vignetting compensation**

(10-12 bits per pixel)
1 intensity value per pixel
■ **White balance**

Pixel values linear in energy

■ **Demosaic**

3x10 bits per pixel
RGB intensity per pixel
■ **Denoise**

Pixel values linear in energy

■ **Gamma Correction (non-linear mapping)**

■ **Local tone mapping**

3x8-bits per pixel
Pixel values perceptually linear
■ **Final adjustments sharpen, fix chromatic aberrations, hue adjust, etc.**

Next time

Acknowledgements

- **Thanks and credit for slides to Ren Ng and Marc Levoy**