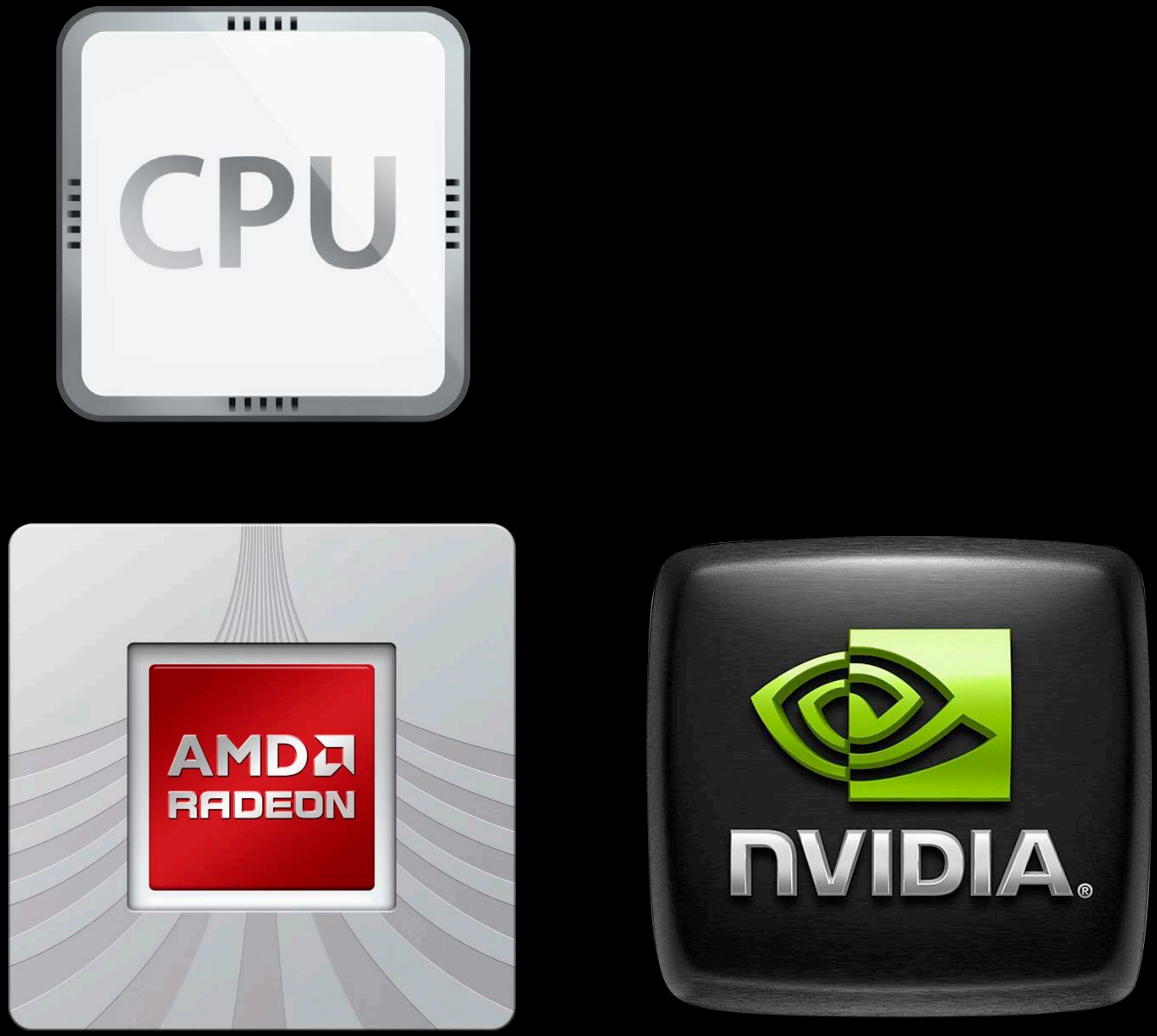
# Working with OpenCL

Session 508

**James Shearer**
Journeyman Hacker

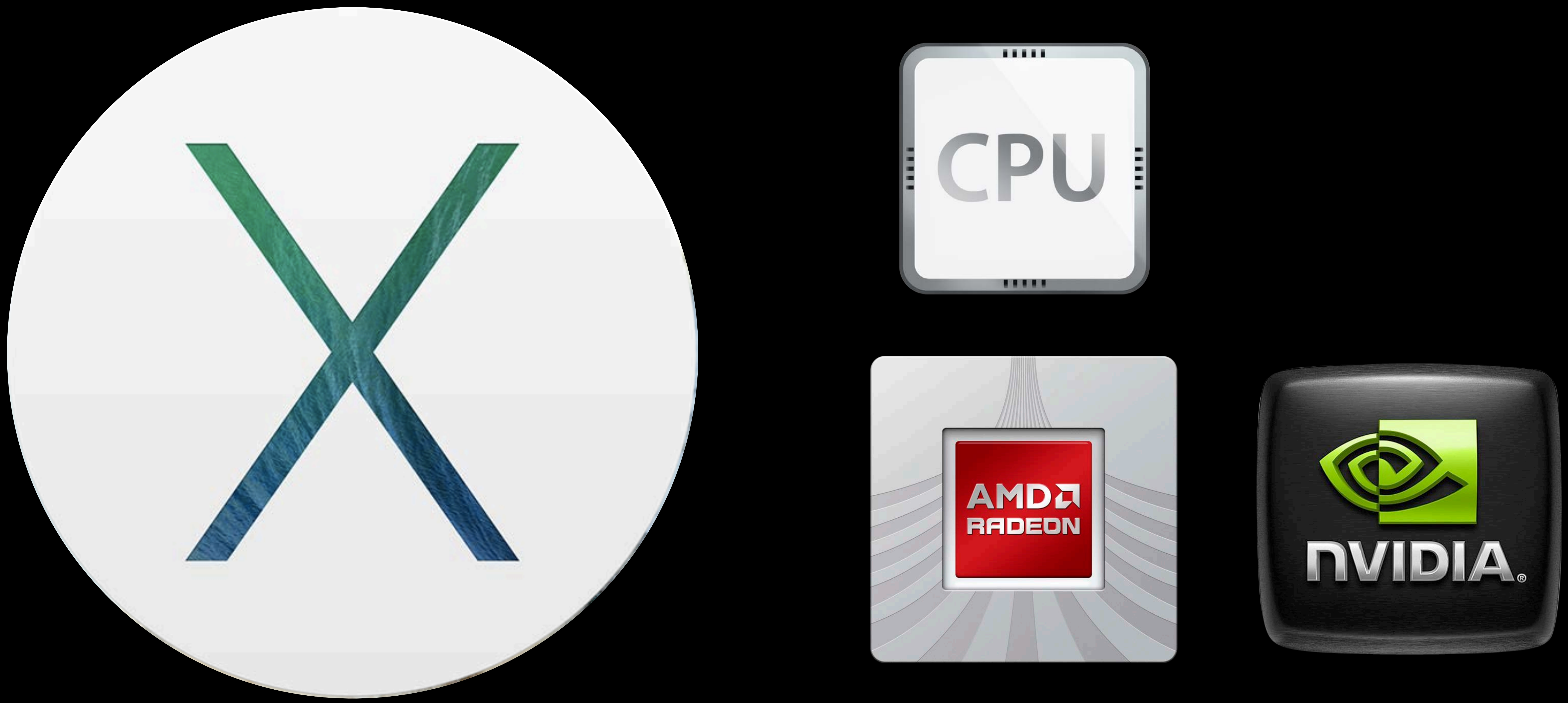
# Where Will OpenCL Work?

## Mac OS X 10.6 Snow Leopard



# Where Will OpenCL Work?

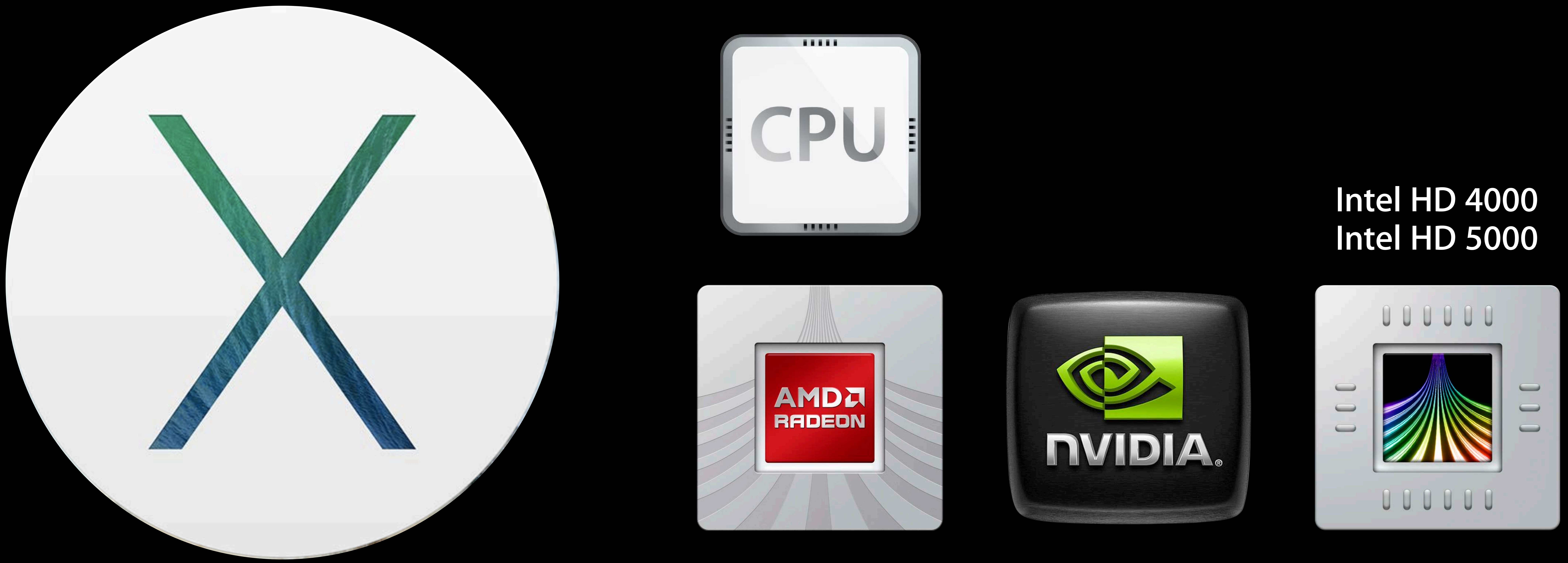
## OS X Mavericks



# Where Will OpenCL Work?

## OS X Mavericks

NEW

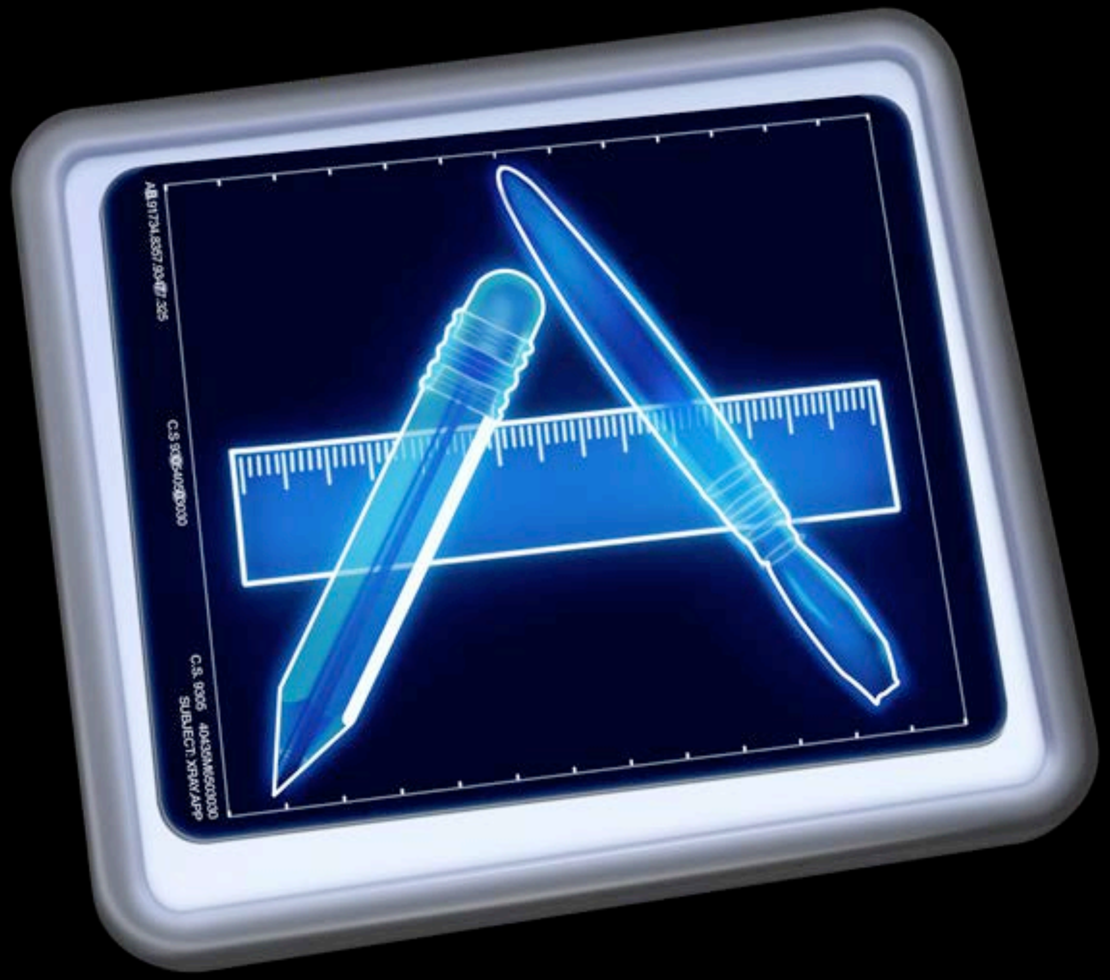


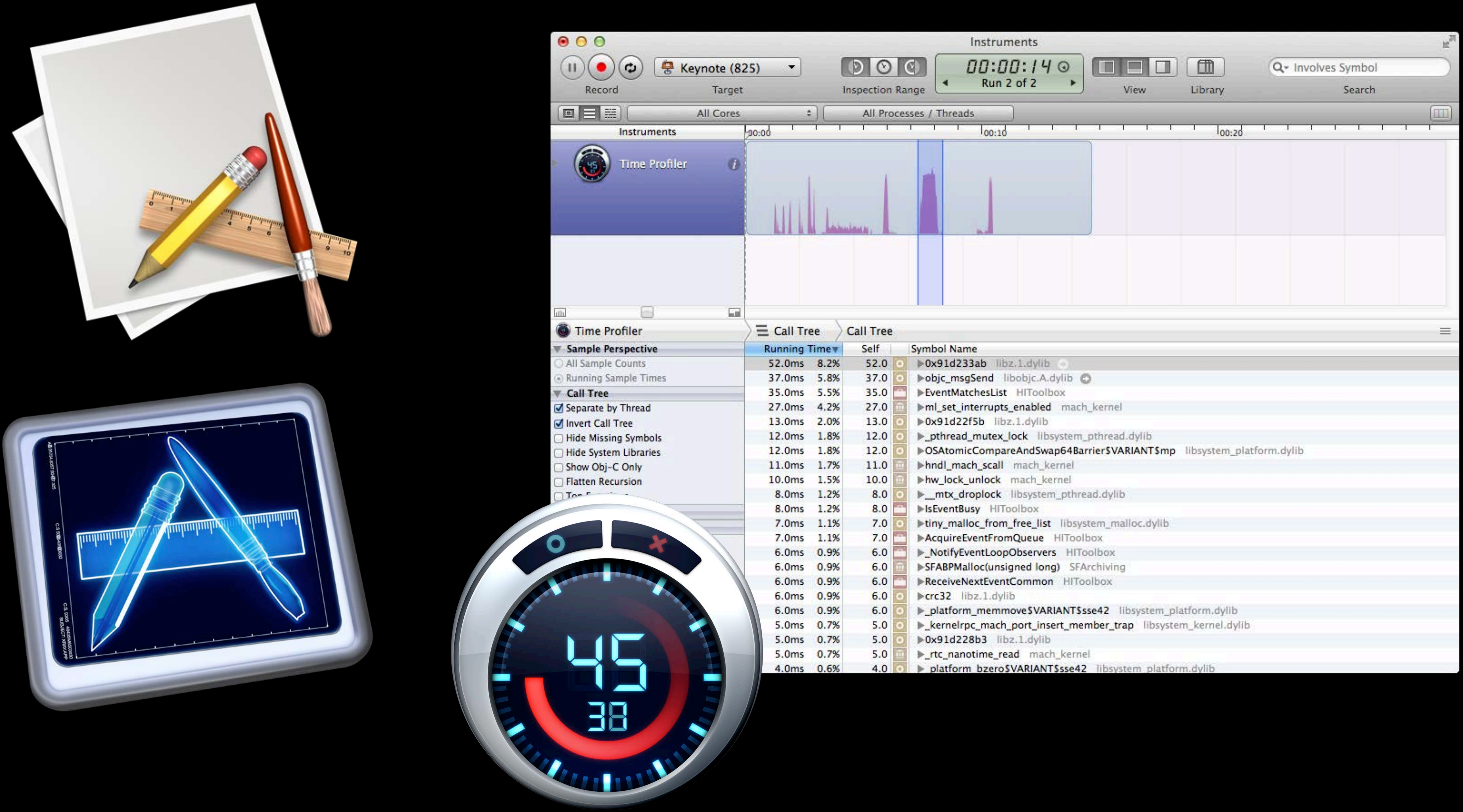
Intel HD 4000
Intel HD 5000

NEW

# CPU and GPU Supported on All Shipping Macs

□ Am I waiting for something?





Time Profiler

▼ Sample Perspective

○ All Sample Counts

● Running Sample Times

▼ Call Tree

☑ Separate by Thread

☑ Invert Call Tree

□ Hide Missing Symbols

□ Hide System Libraries

□ Show Obj-C Only

□ Flatten Recursion

□ Top Functions

| Running Time ▾ | | Self | Symbol Name | |
|---|---|---|---|---|
| 52.0ms | 8.2% | 52.0 | ▶0x91d233ab | libz.1.dylib |
| 37.0ms | 5.8% | 37.0 | ▶objc_msgSend | libobjc.A.dylib |
| 35.0ms | 5.5% | 35.0 | ▶EventMatchesList | HIToolbox |
| 27.0ms | 4.2% | 27.0 | ▶ml_set_interrupts_enabled | mach_kernel |
| 13.0ms | 2.0% | 13.0 | ▶0x91d22f5b | libz.1.dylib |
| 12.0ms | 1.8% | 12.0 | ▶_pthread_mutex_lock | libsystem_pthread.dylib |
| 12.0ms | 1.8% | 12.0 | ▶OSAtomicCompareAndSwap64Barrier$VARIANT$mp | libsystem_platform.dylib |
| 11.0ms | 1.7% | 11.0 | ▶hndl_mach_scall | mach_kernel |
| 10.0ms | 1.5% | 10.0 | ▶hw_lock_unlock | mach_kernel |
| 8.0ms | 1.2% | 8.0 | ▶__mtx_droplock | libsystem_pthread.dylib |
| 8.0ms | 1.2% | 8.0 | ▶IsEventBusy | HIToolbox |
| 7.0ms | 1.1% | 7.0 | ▶tiny_malloc_from_free_list | libsystem_malloc.dylib |
| 7.0ms | 1.1% | 7.0 | ▶AcquireEventFromQueue | HIToolbox |
| 6.0ms | 0.9% | 6.0 | ▶_NotifyEventLoopObservers | HIToolbox |
| 6.0ms | 0.9% | 6.0 | ▶SFABPMalloc(unsigned long) | SFArchiving |
| 6.0ms | 0.9% | 6.0 | ▶ReceiveNextEventCommon | HIToolbox |
| 6.0ms | 0.9% | 6.0 | ▶crc32 | libz.1.dylib |
| 6.0ms | 0.9% | 6.0 | ▶_platform_memmove$VARIANT$sse42 | libsystem_platform.dylib |
| 5.0ms | 0.7% | 5.0 | ▶_kernelrpc_mach_port_insert_member_trap | libsystem_kernel.dylib |
| 5.0ms | 0.7% | 5.0 | ▶0x91d228b3 | libz.1.dylib |
| 5.0ms | 0.7% | 5.0 | ▶_rtc_nanotime_read | mach_kernel |
| 4.0ms | 0.6% | 4.0 | ▶_platform_bzero$VARIANT$sse42 | libsystem_platform.dylib |

□ Am I waiting for something?

□ Am I waiting for something?

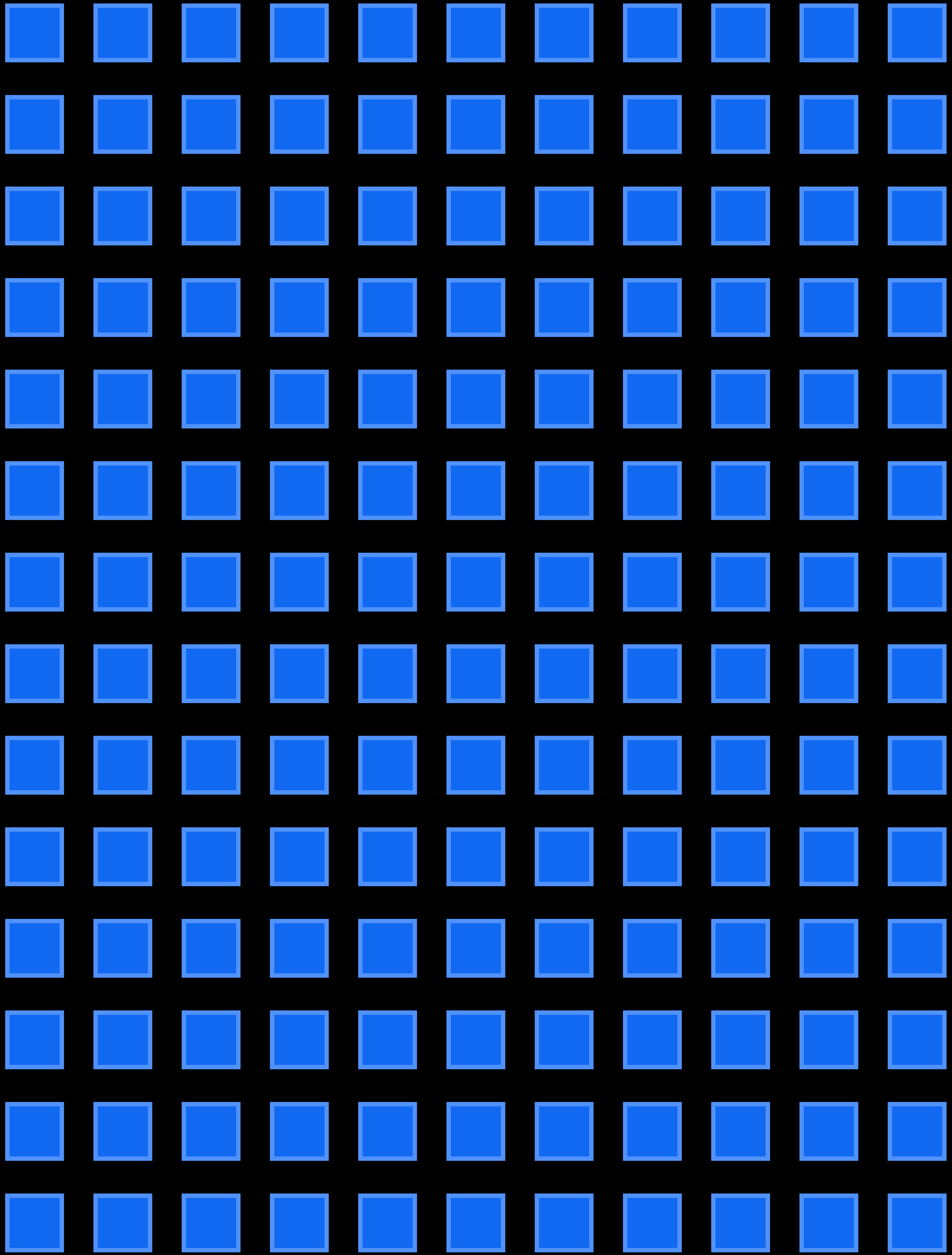
□ Have I avoided something intensive?

□ Am I waiting for something?

□ Have I avoided something intensive?

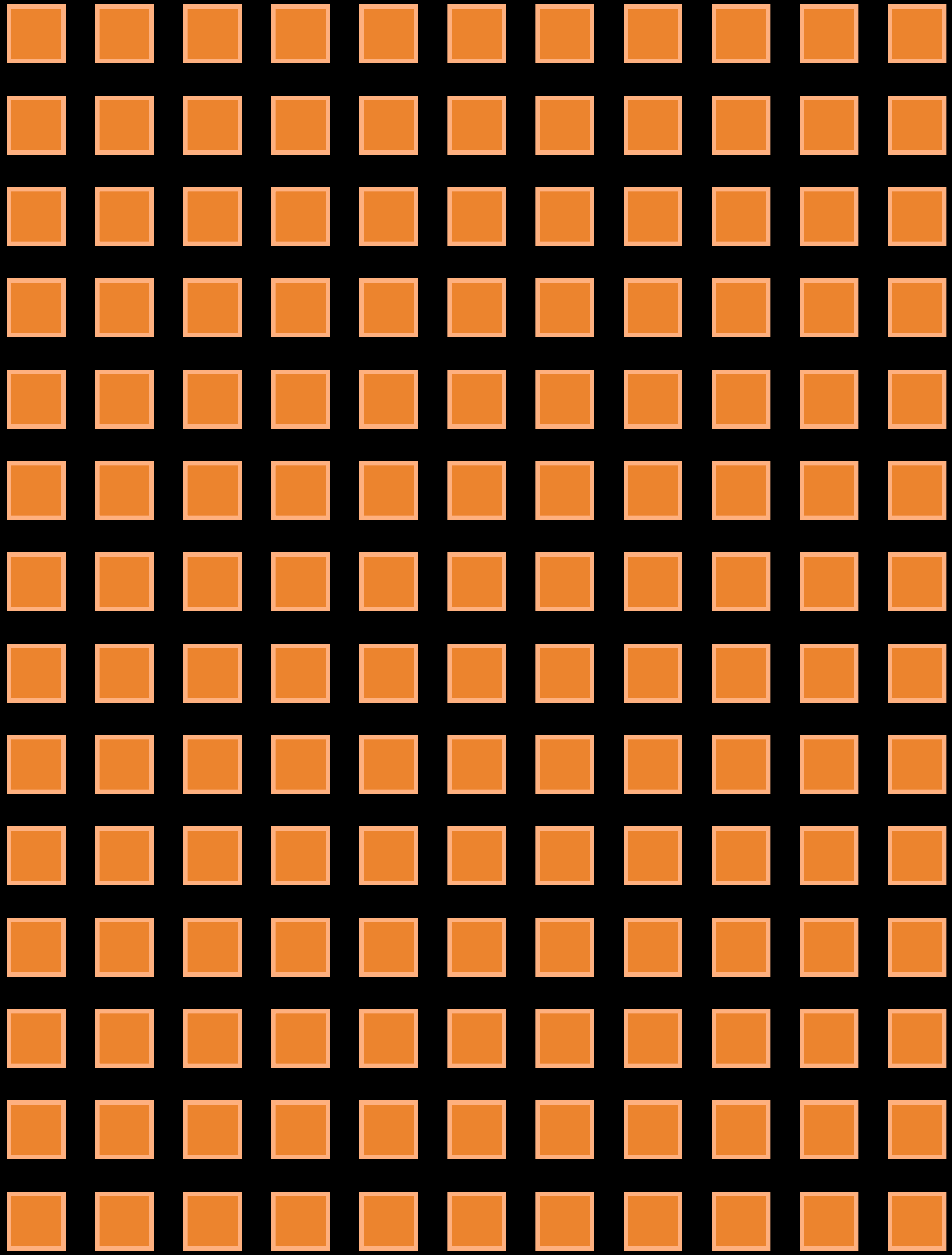
- □ Am I waiting for something?
- □ Have I avoided something intensive?
- □ Do I have a parallel workload?

pieces of data



pieces of data

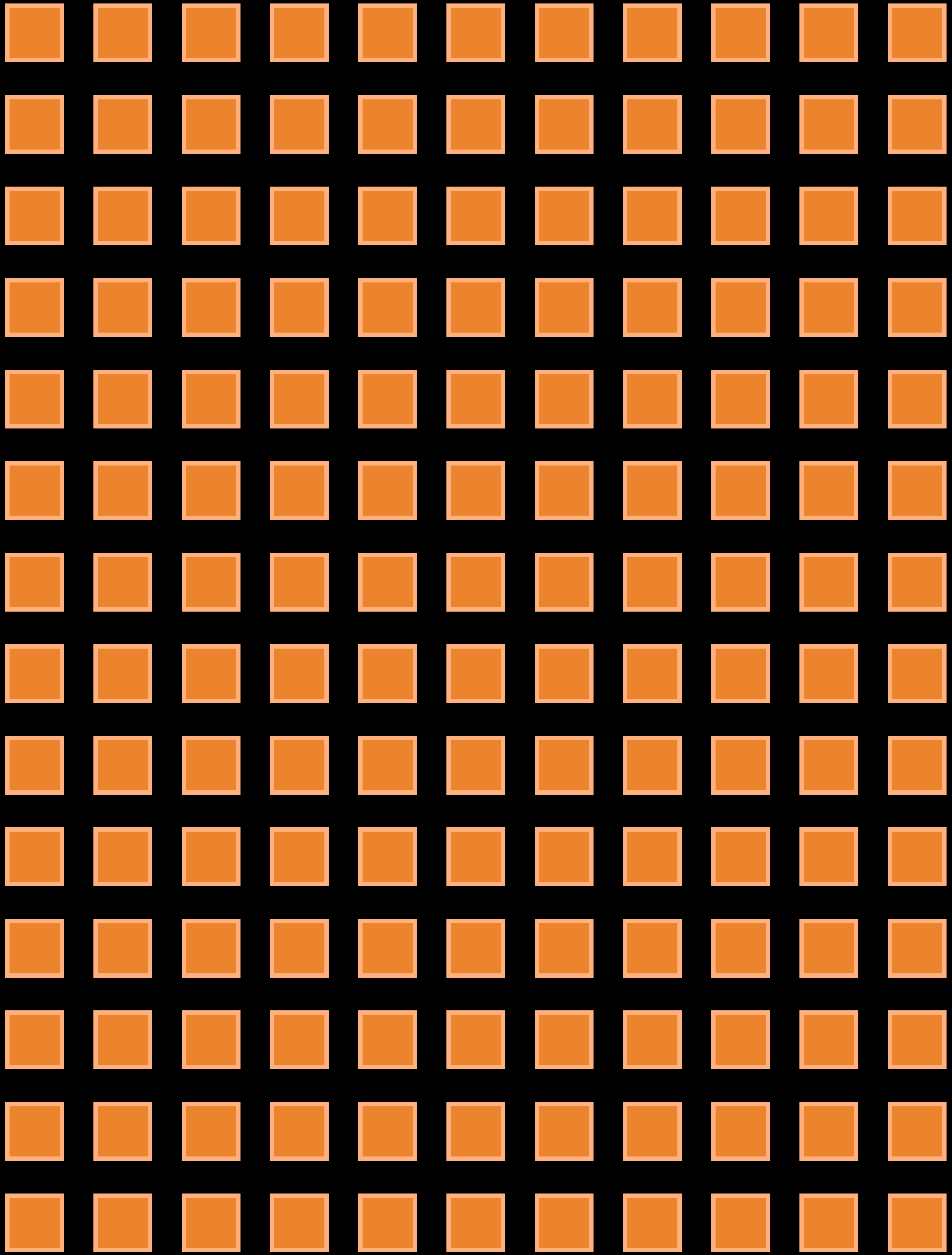
all changing in the same way



pieces of data

all changing in the same way

few dependencies



```
$ grep -re ‘huz*ah’ large.txt
```

□ Am I waiting for something?

□ Have I avoided something intensive?

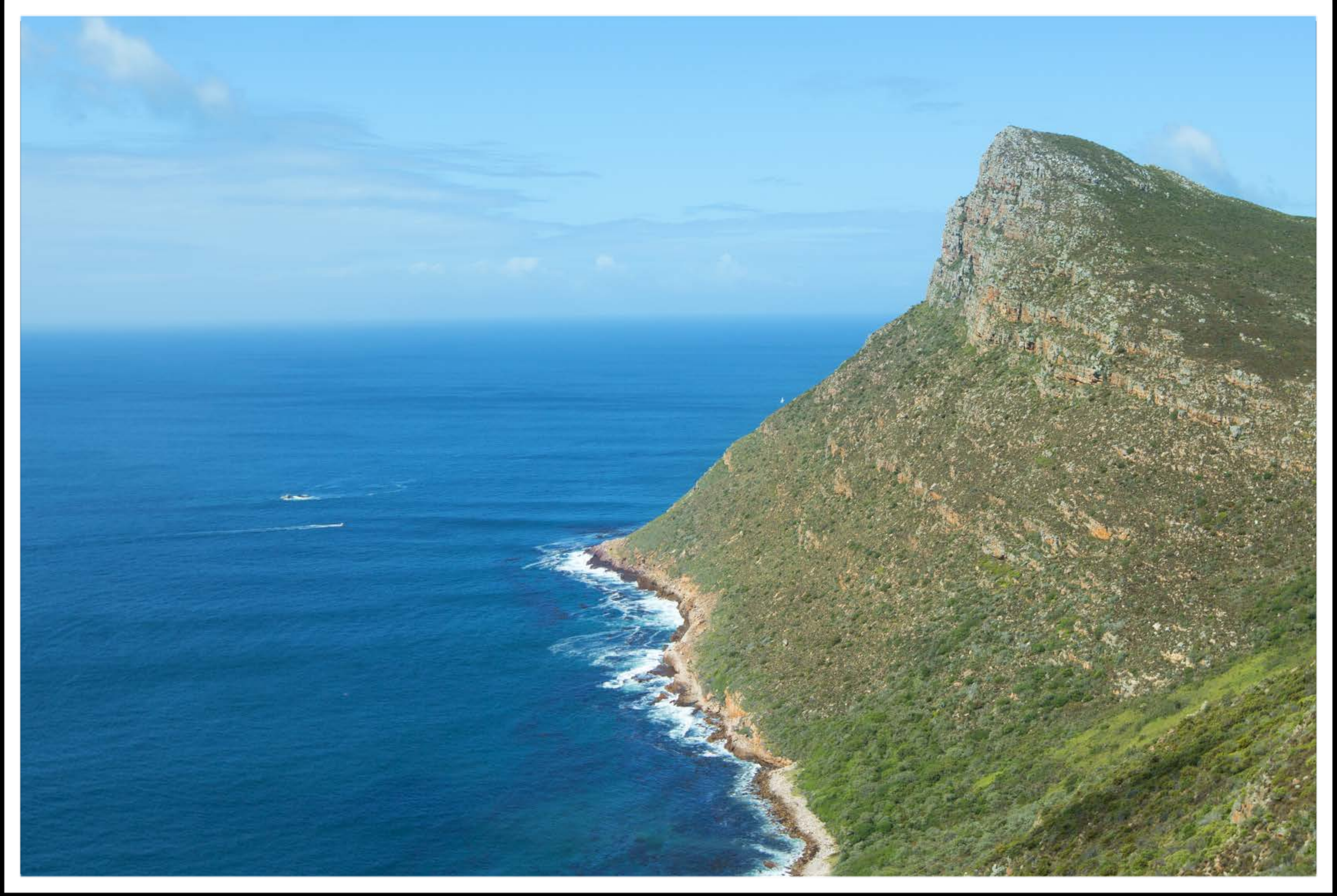
□ Do I have a parallel workload?

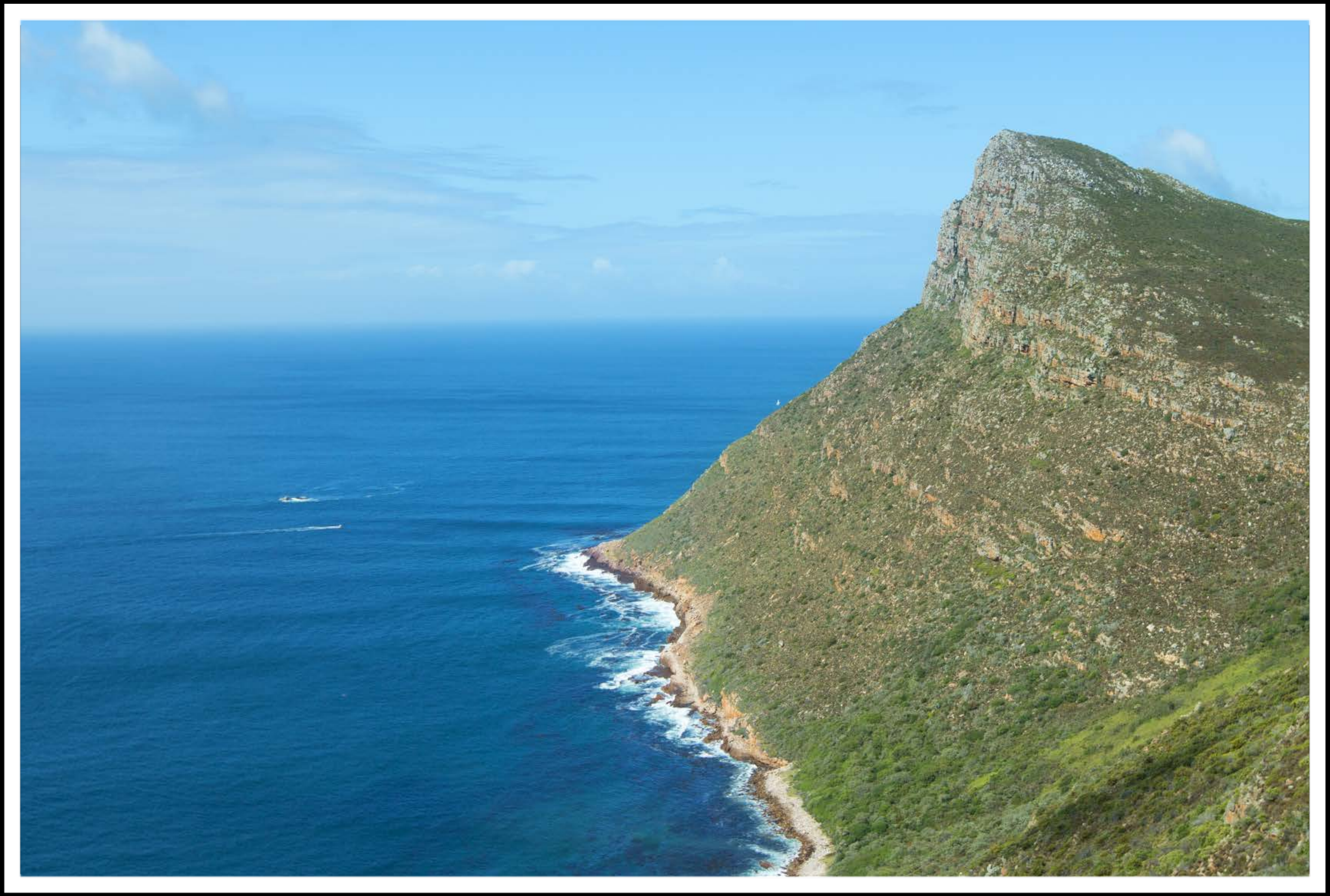
□ Am I waiting for something?

□ Have I avoided something intensive?

□ Do I have a parallel workload?

□ Can I **earn** a parallel workload?





0 255

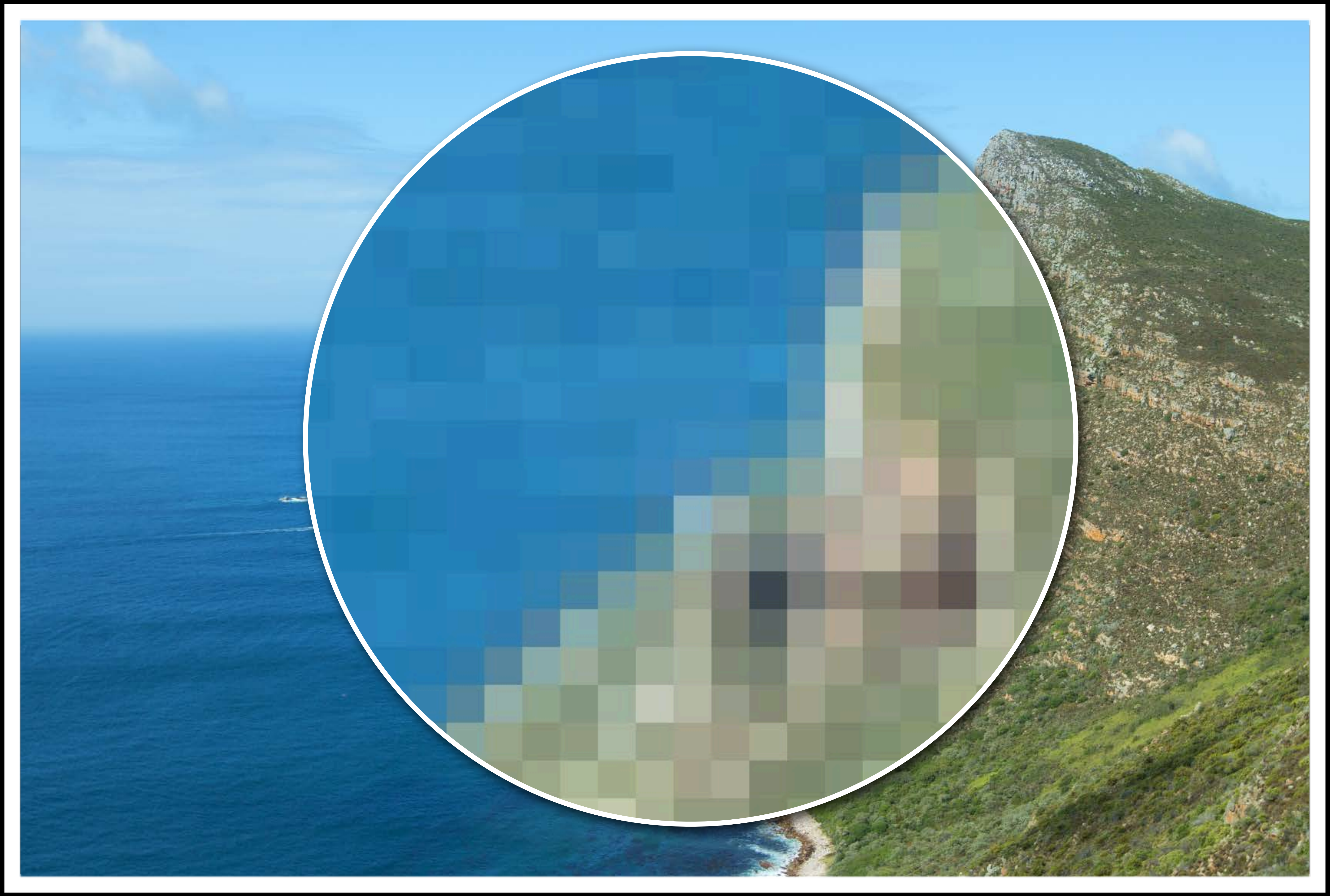
Red Histogram

0 255

Green Histogram

0 255

Blue Histogram



0 255

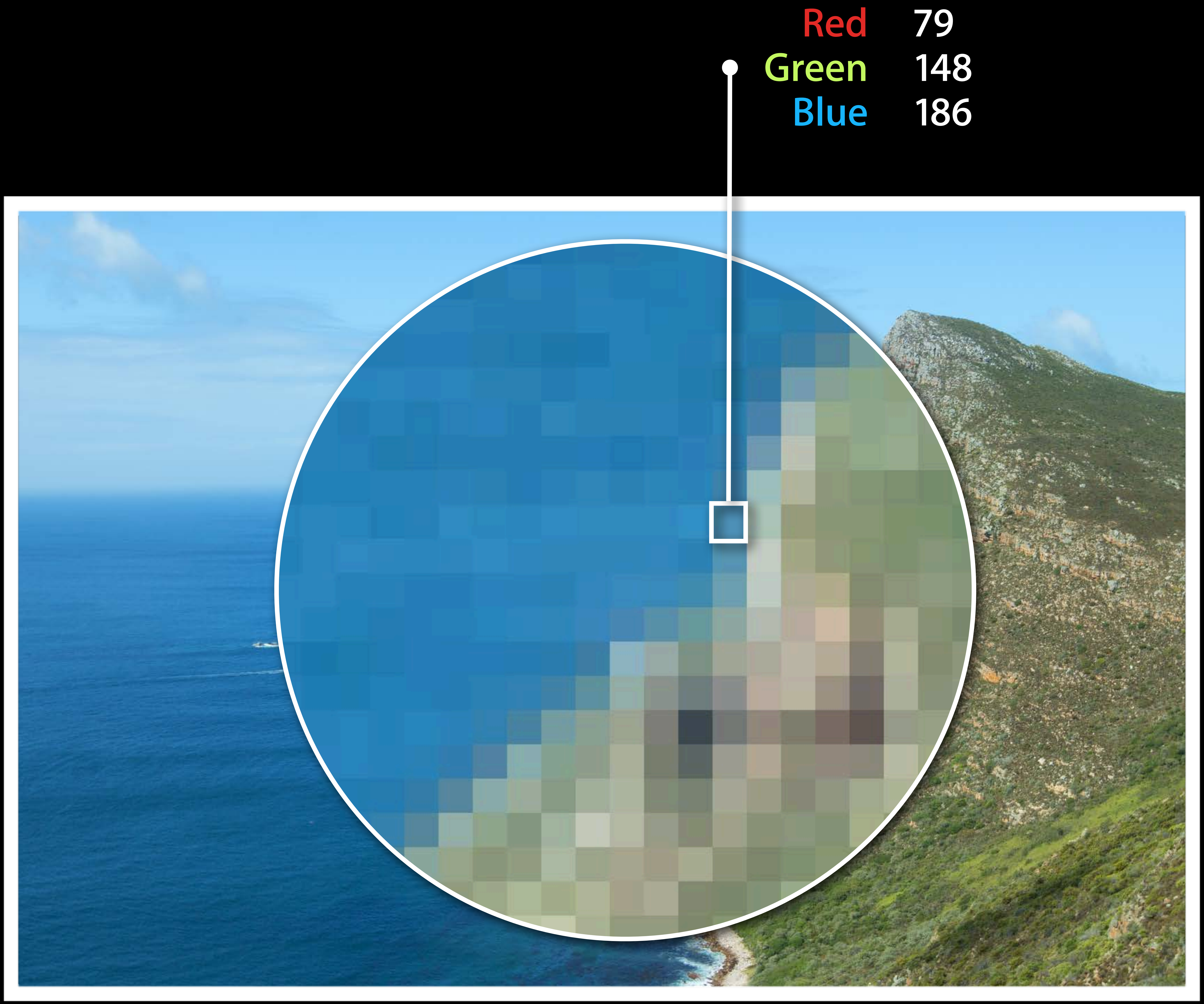
Red Histogram

0 255

Green Histogram

0 255

Blue Histogram



Red 79
Green 148
Blue 186

0 255

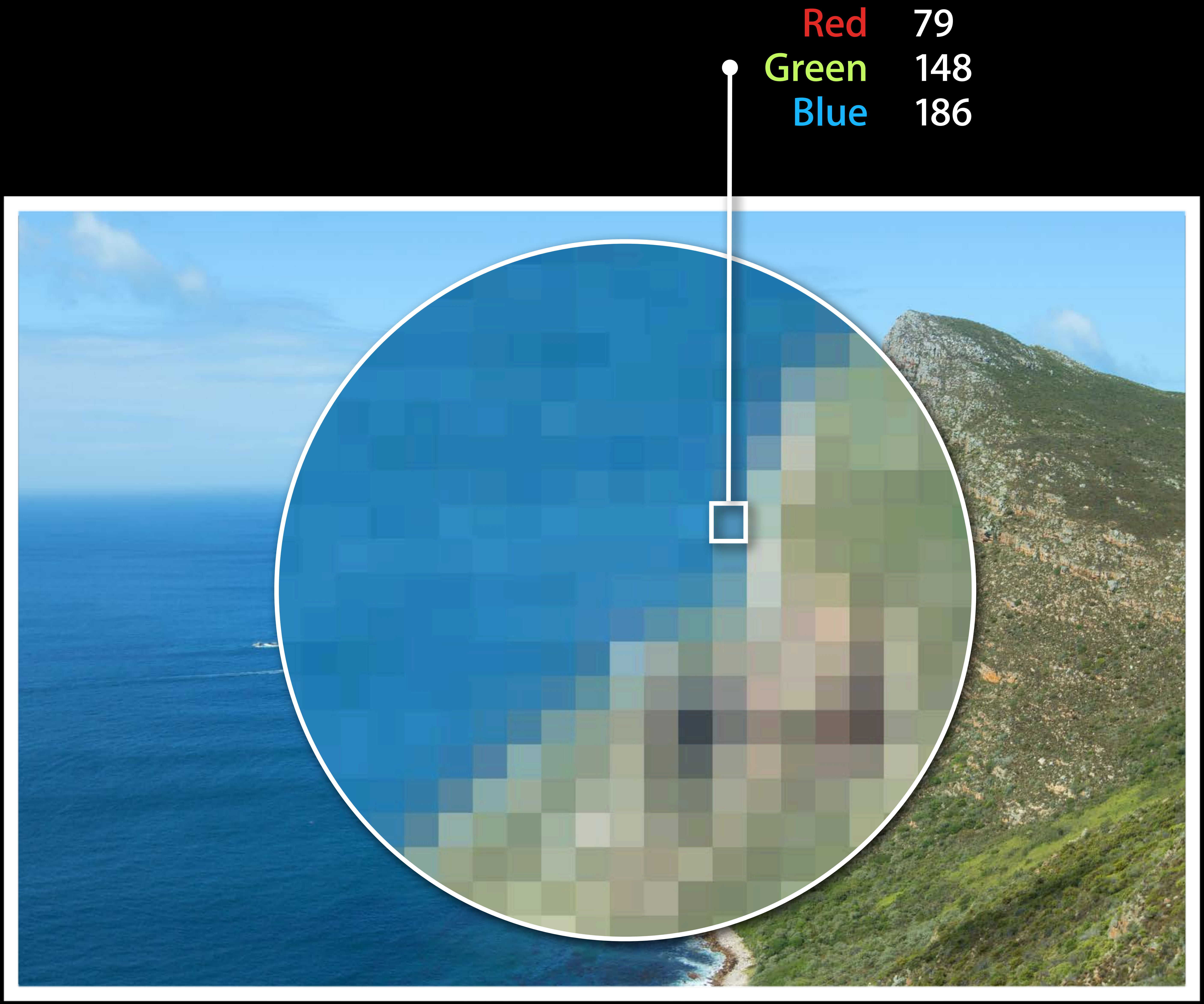
Red Histogram

0 255

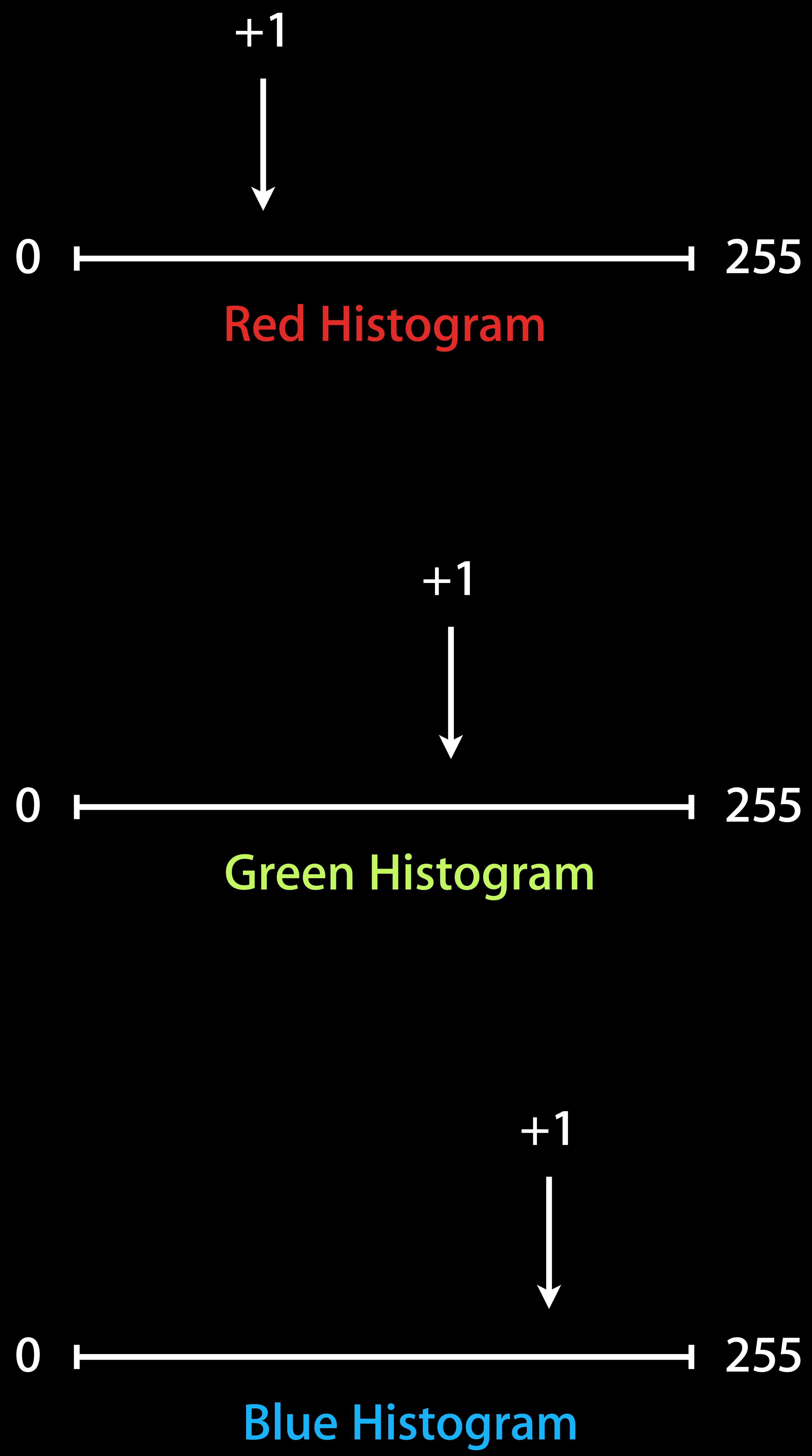
Green Histogram

0 255

Blue Histogram



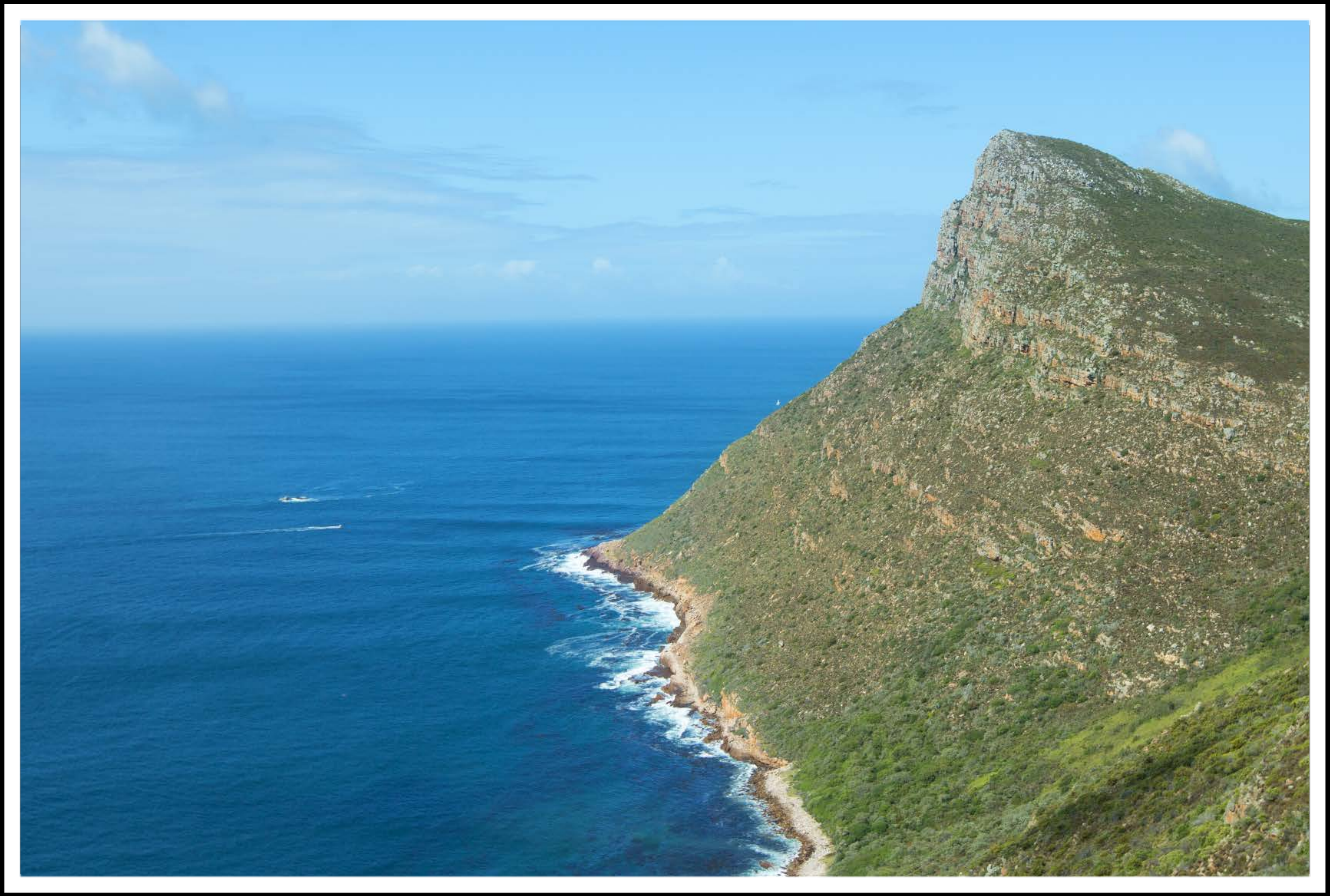
Red 79
Green 148
Blue 186

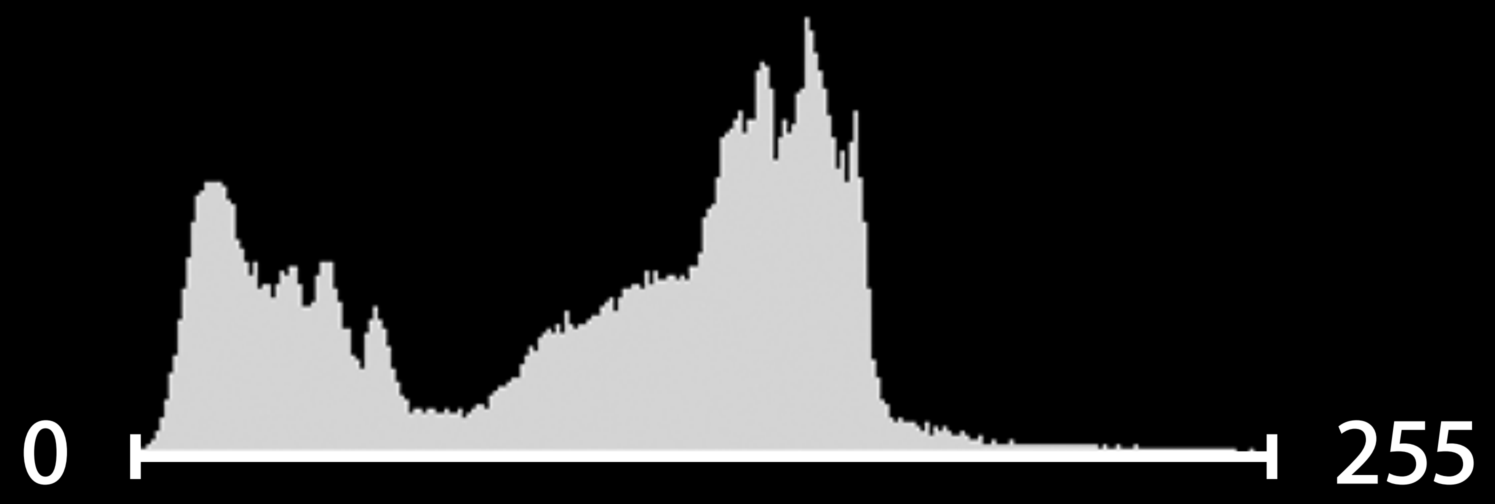


+1
0 255
Red Histogram

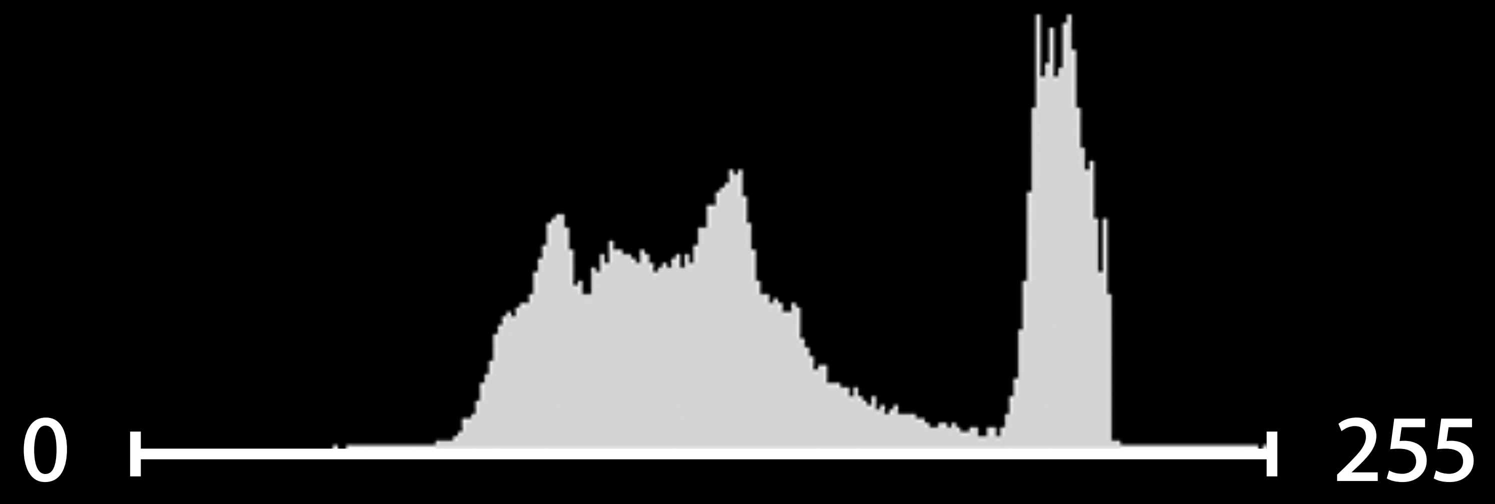
+1
0 255
Green Histogram

+1
0 255
Blue Histogram

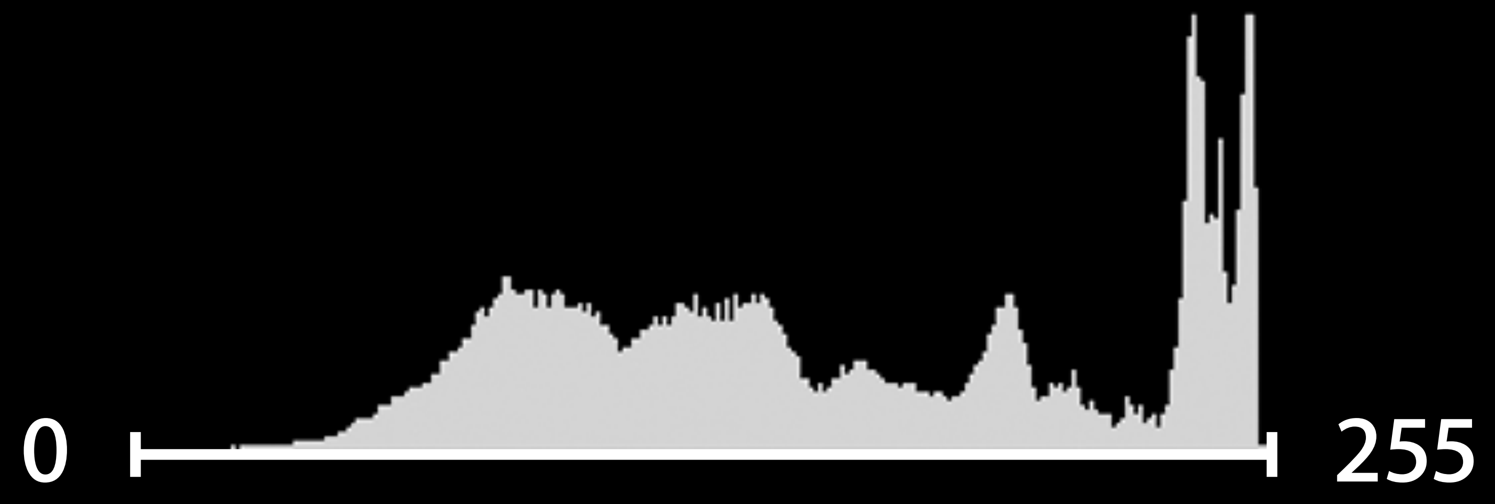




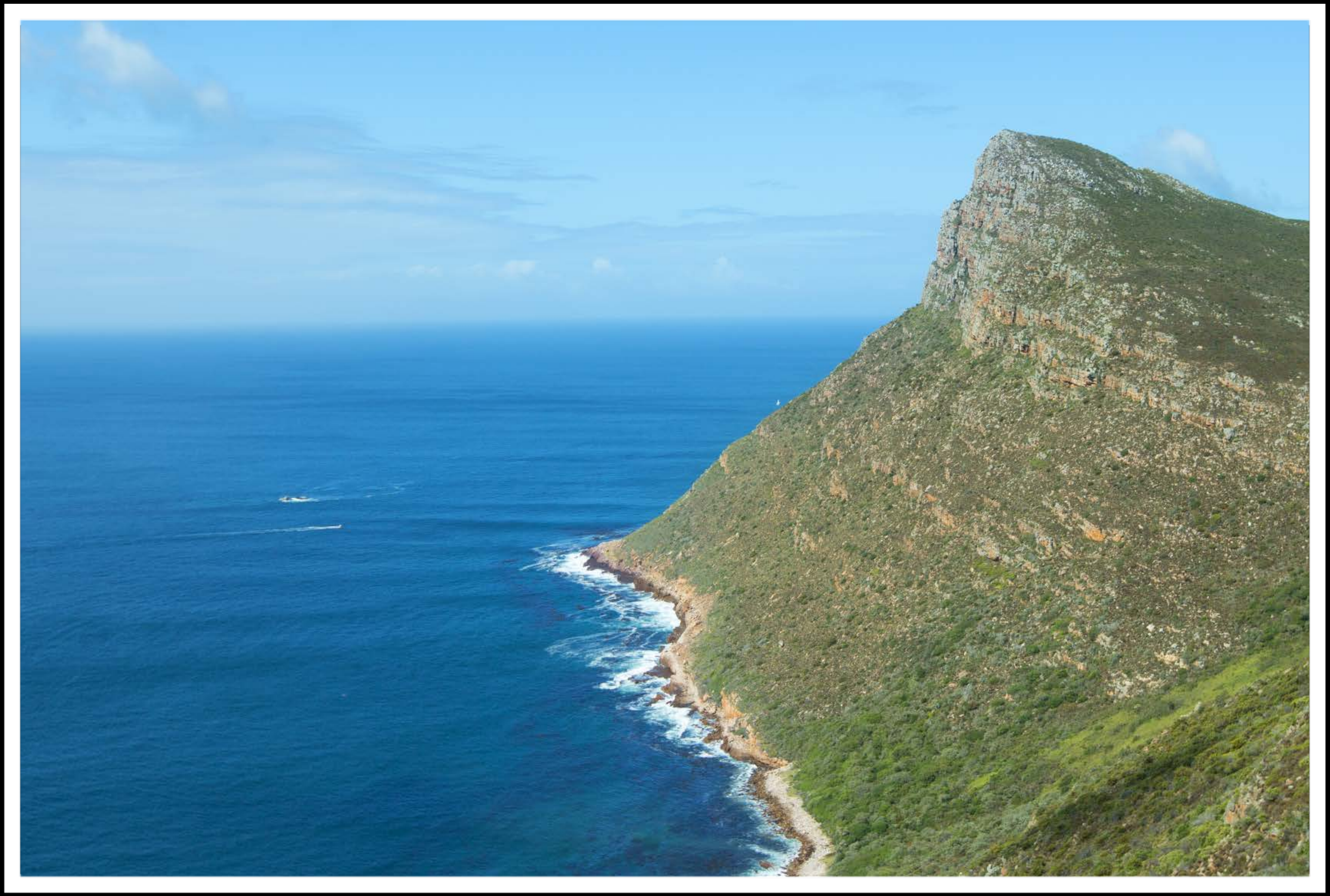
Red Histogram

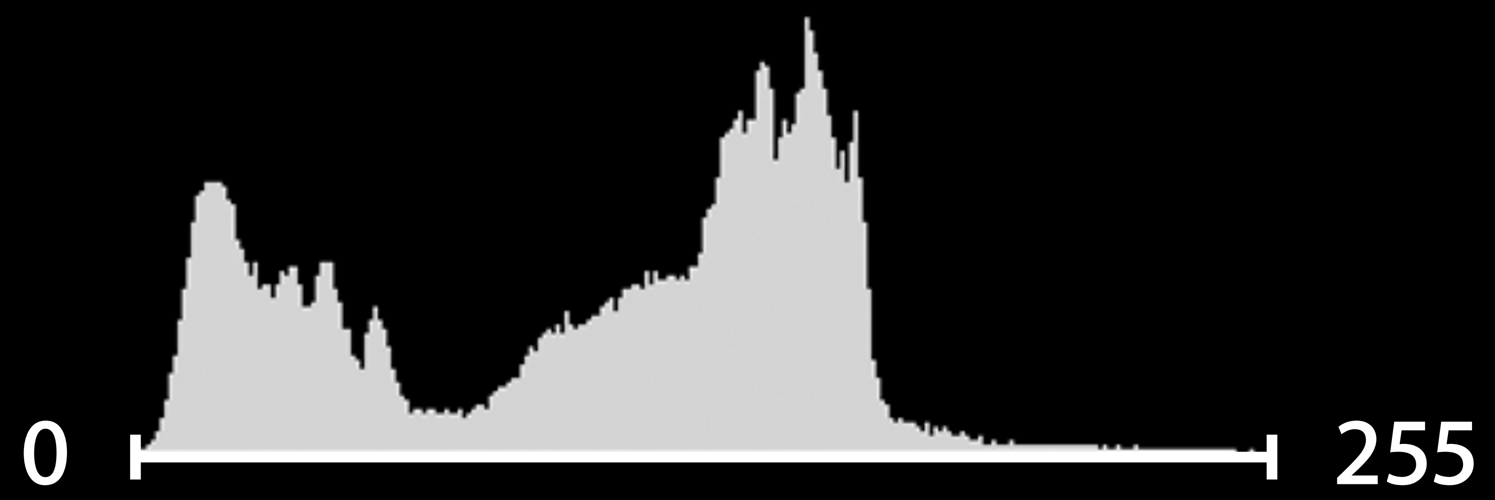


Green Histogram

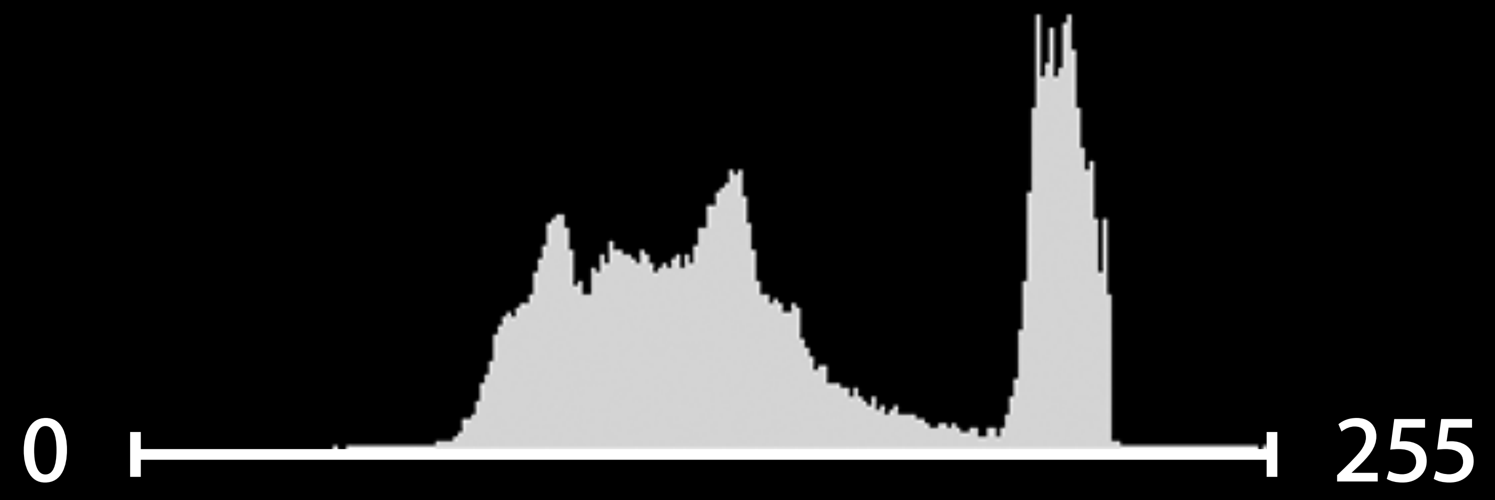


Blue Histogram

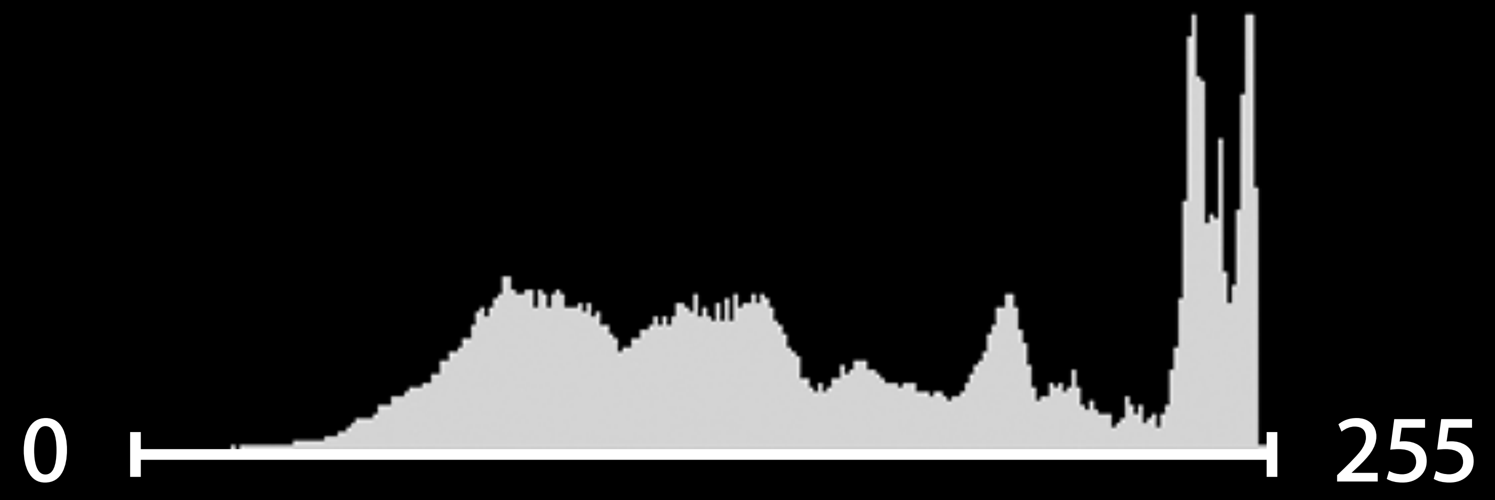




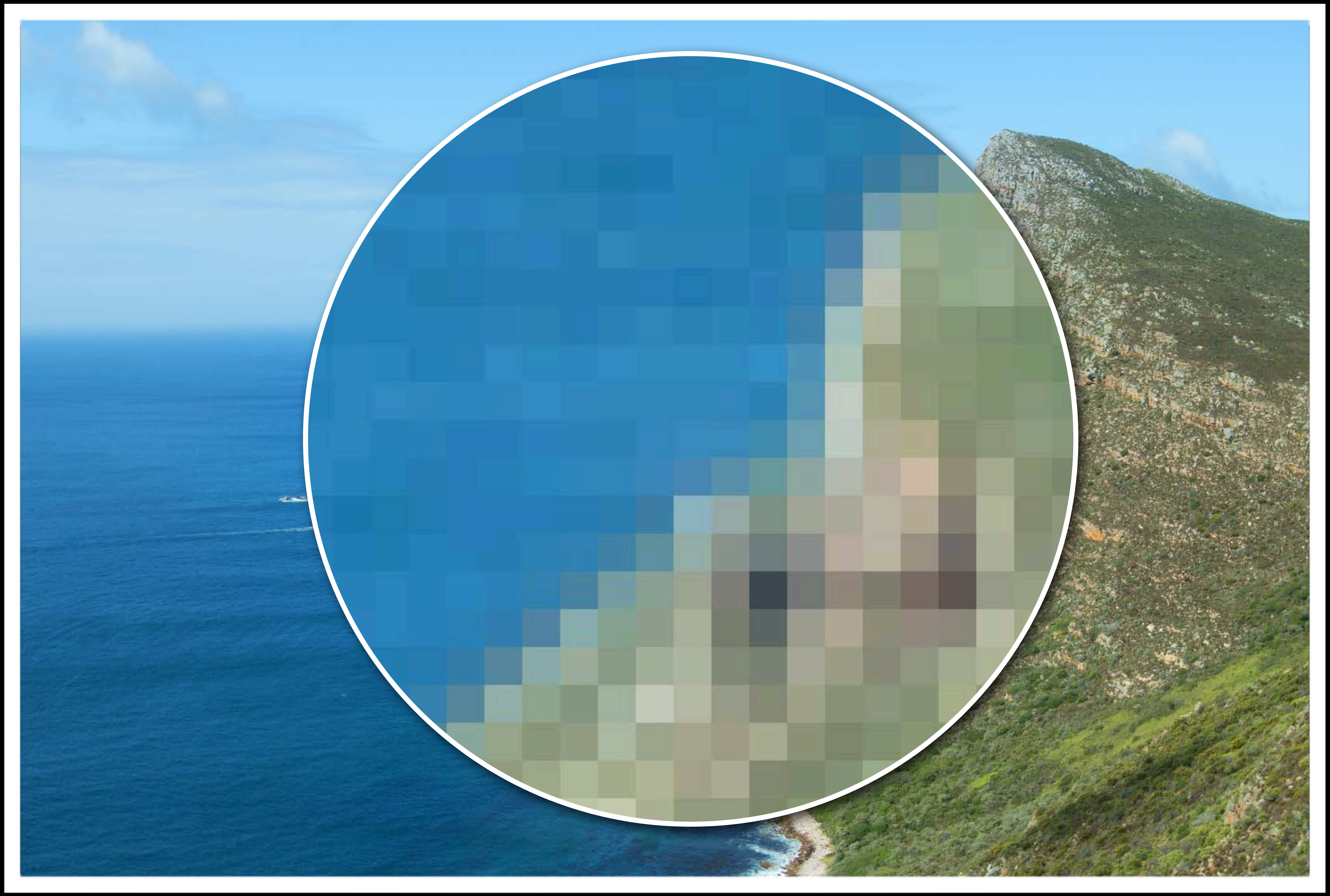
Red Histogram

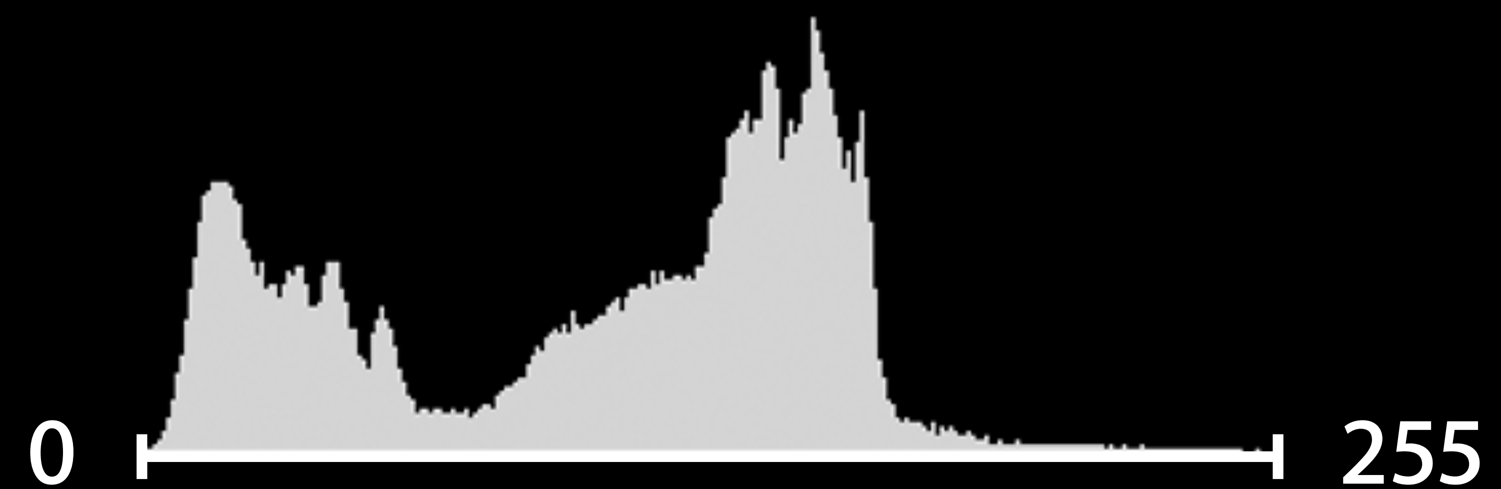


Green Histogram

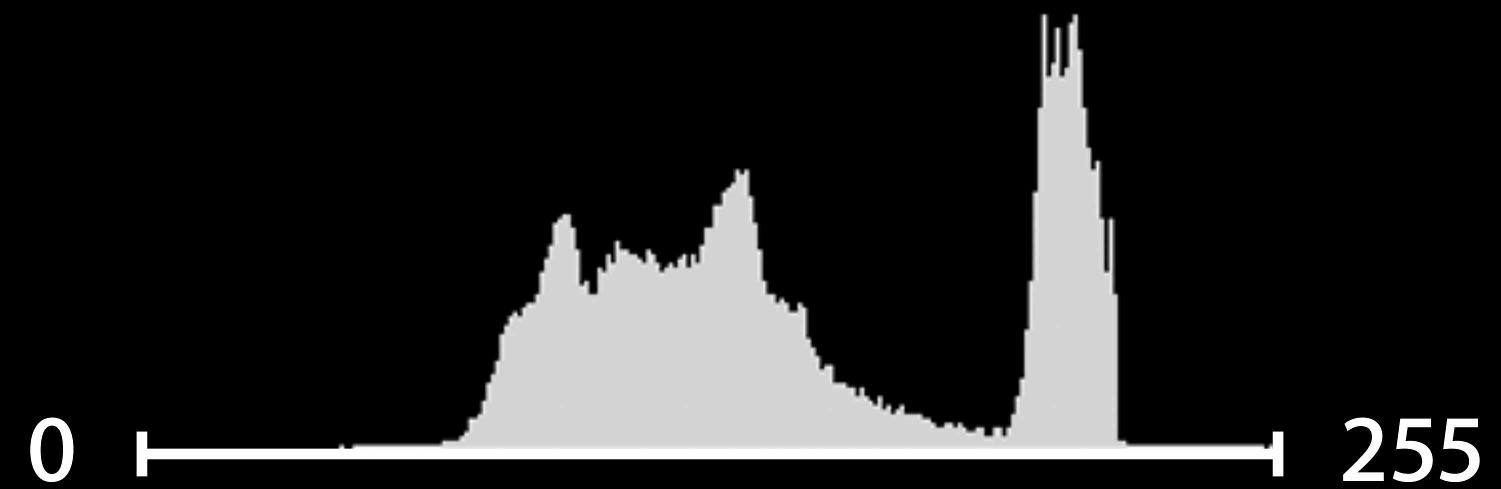


Blue Histogram

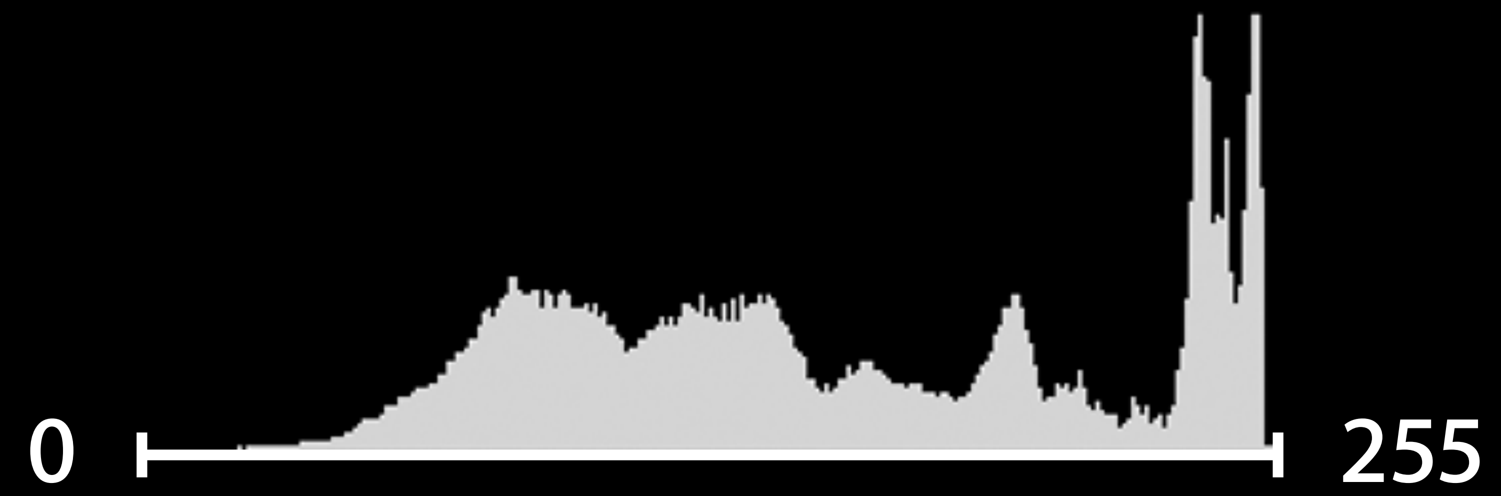




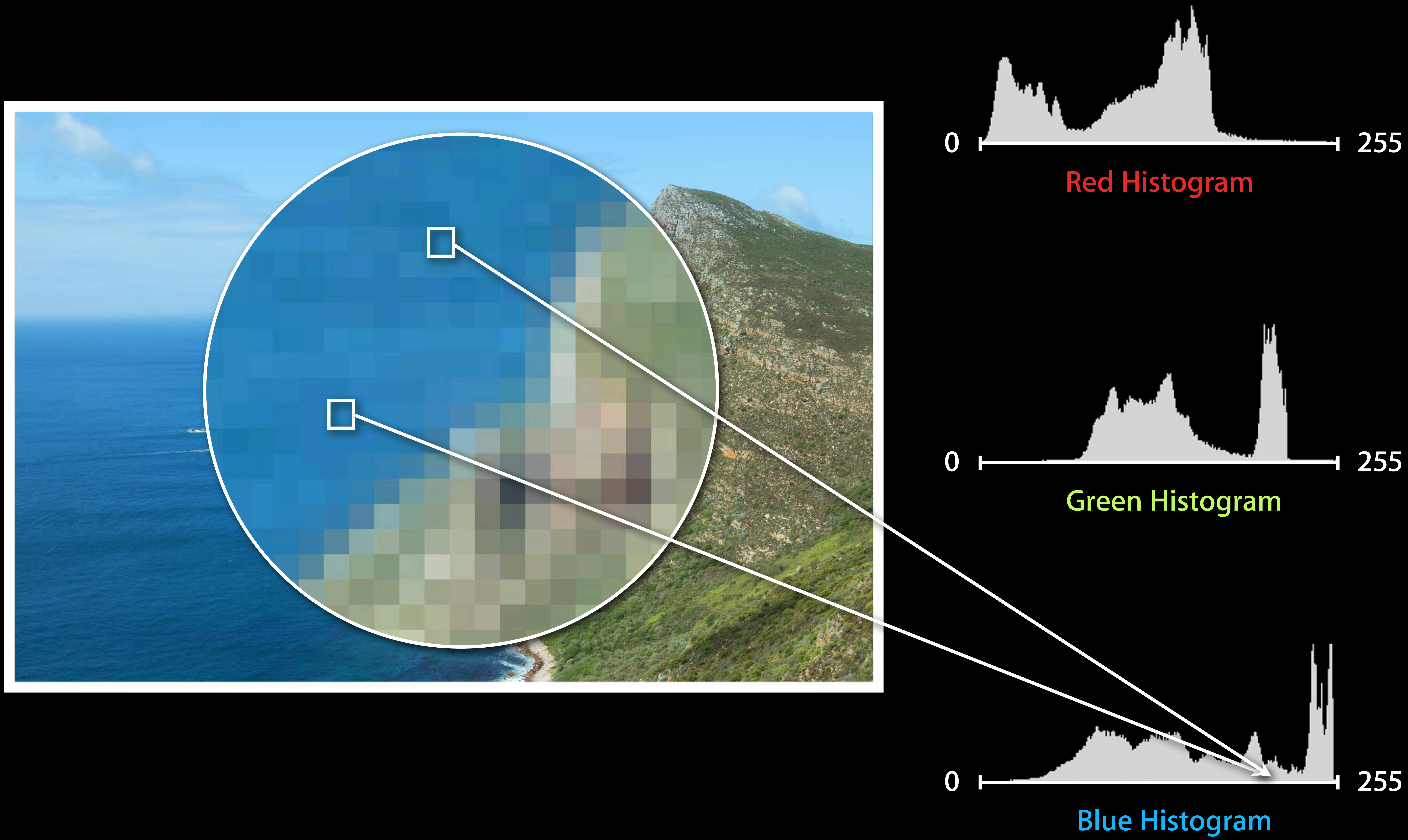
Red Histogram

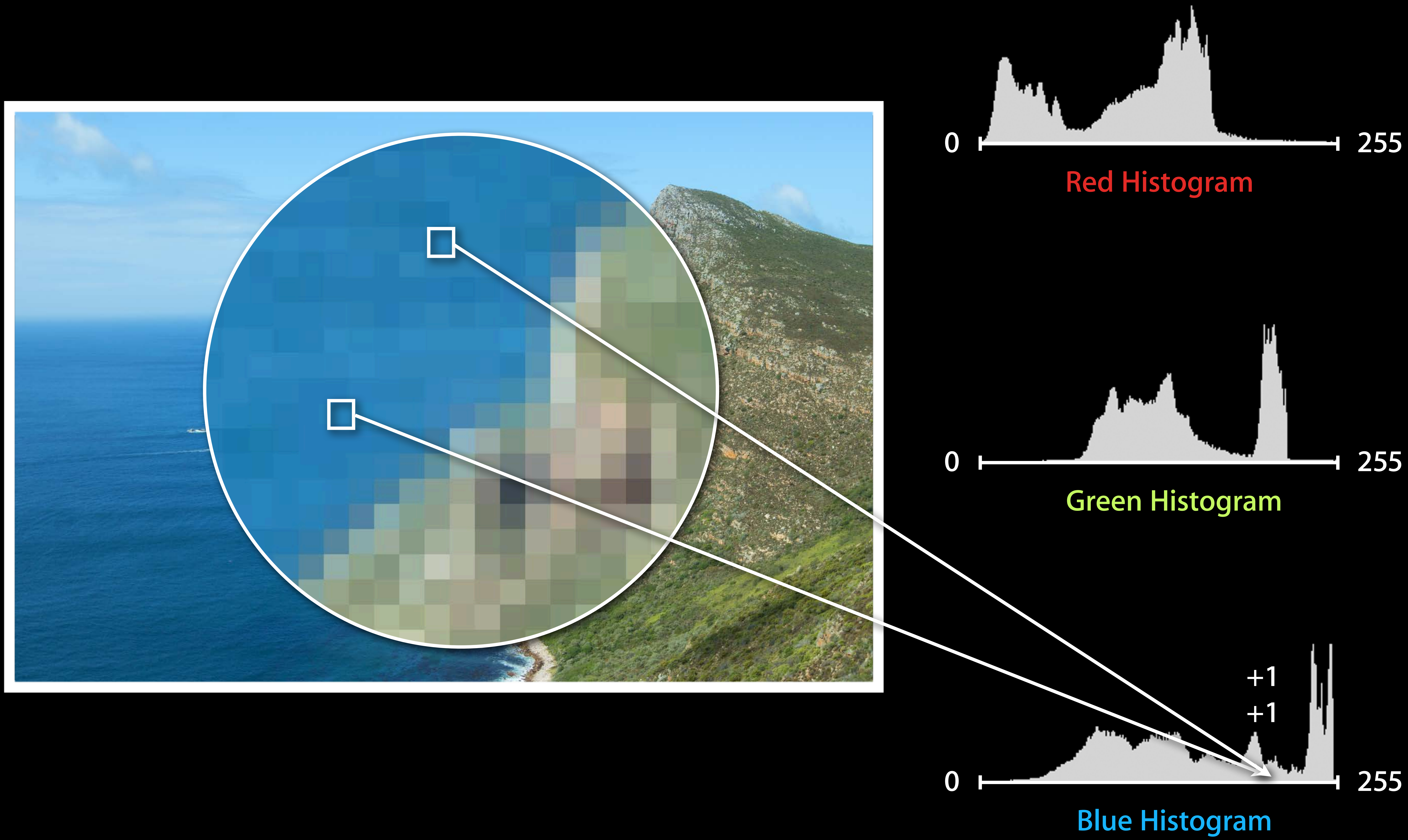


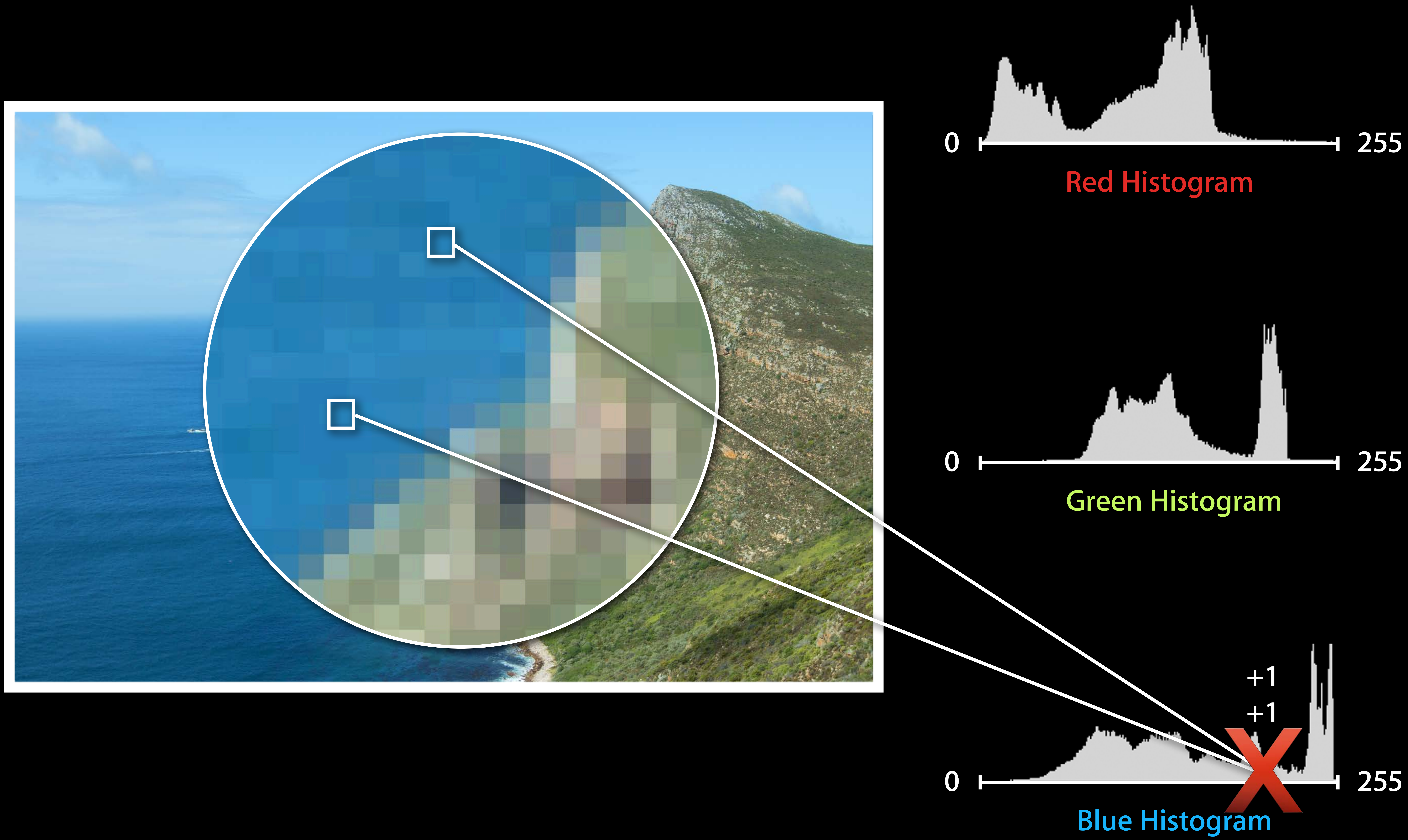
Green Histogram



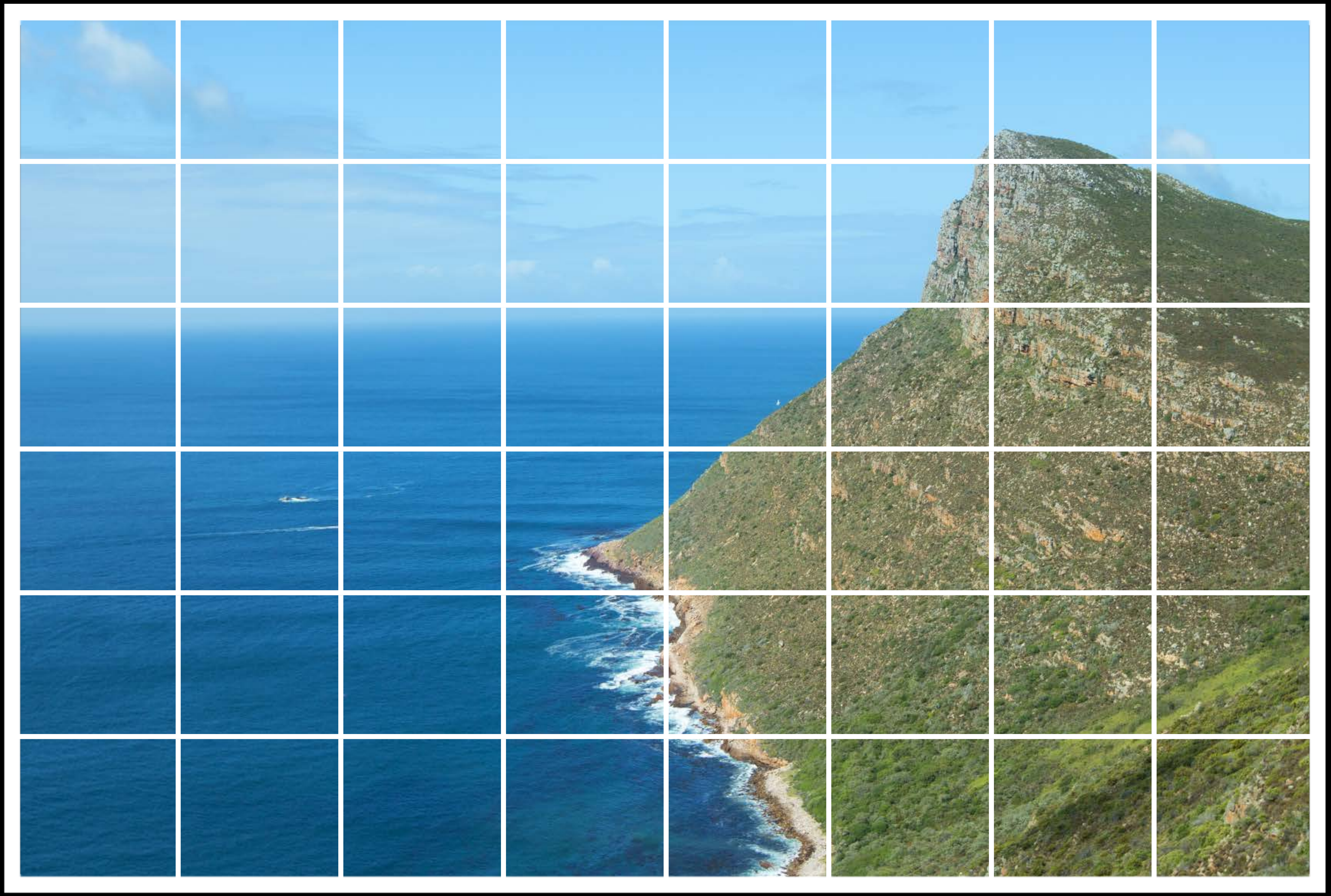
Blue Histogram

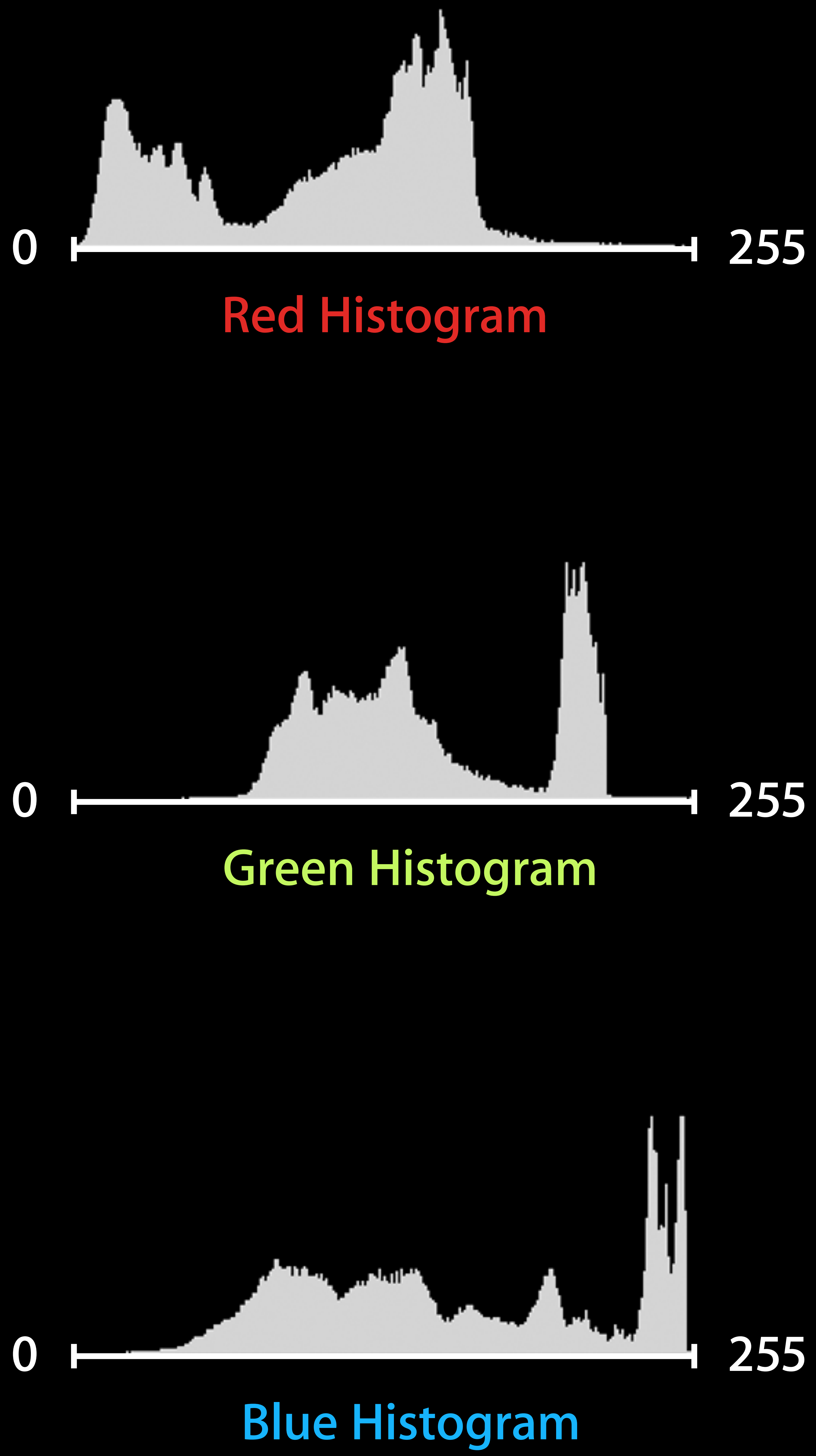

0
255
Red Histogram
0
255
Green Histogram
0
255
Blue Histogram


0
255
Red Histogram
0
255
Green Histogram
+1
+1
0
255
Blue Histogram



0
255
Red Histogram
0
255
Green Histogram
+1
+1
0
255
Blue Histogram

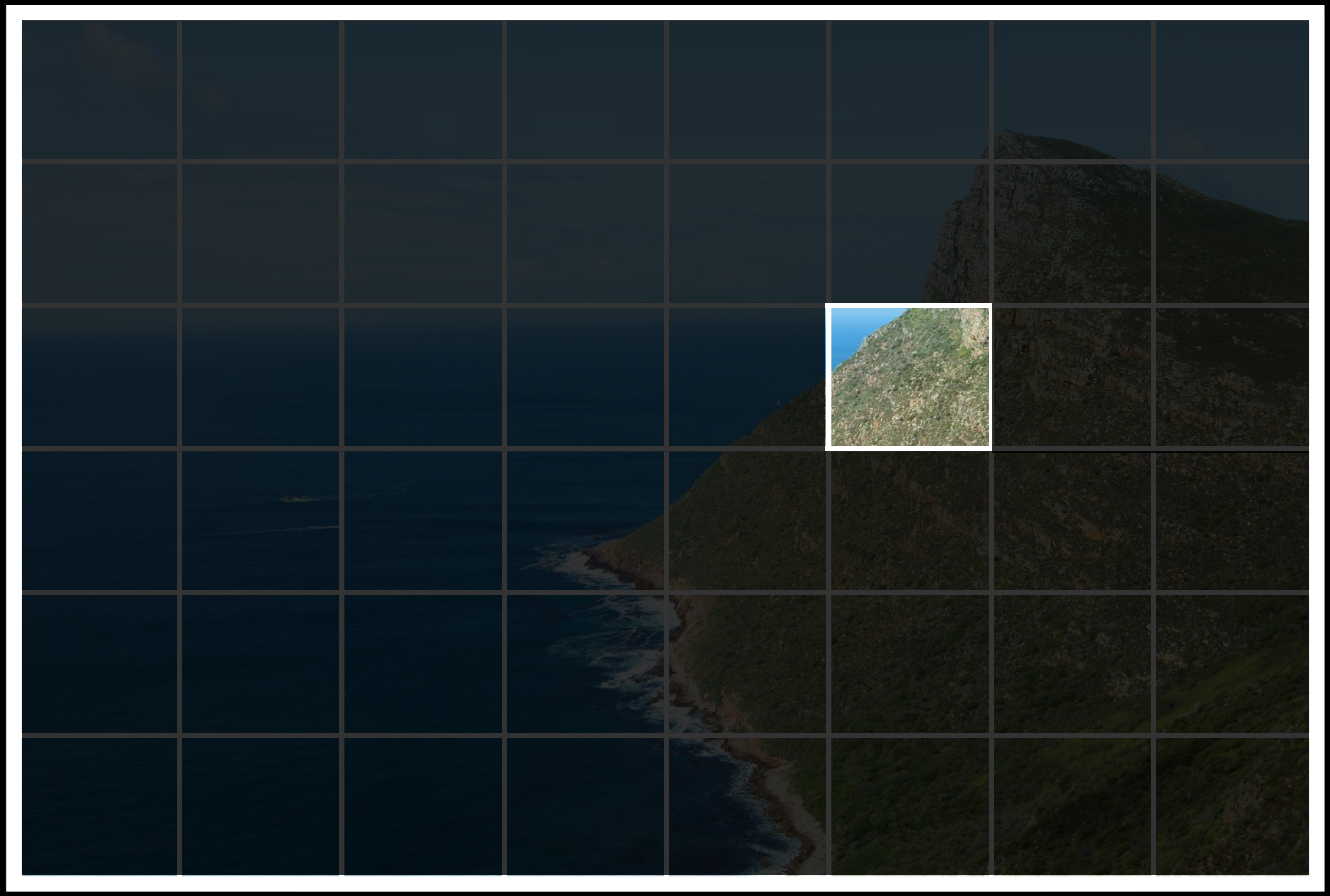


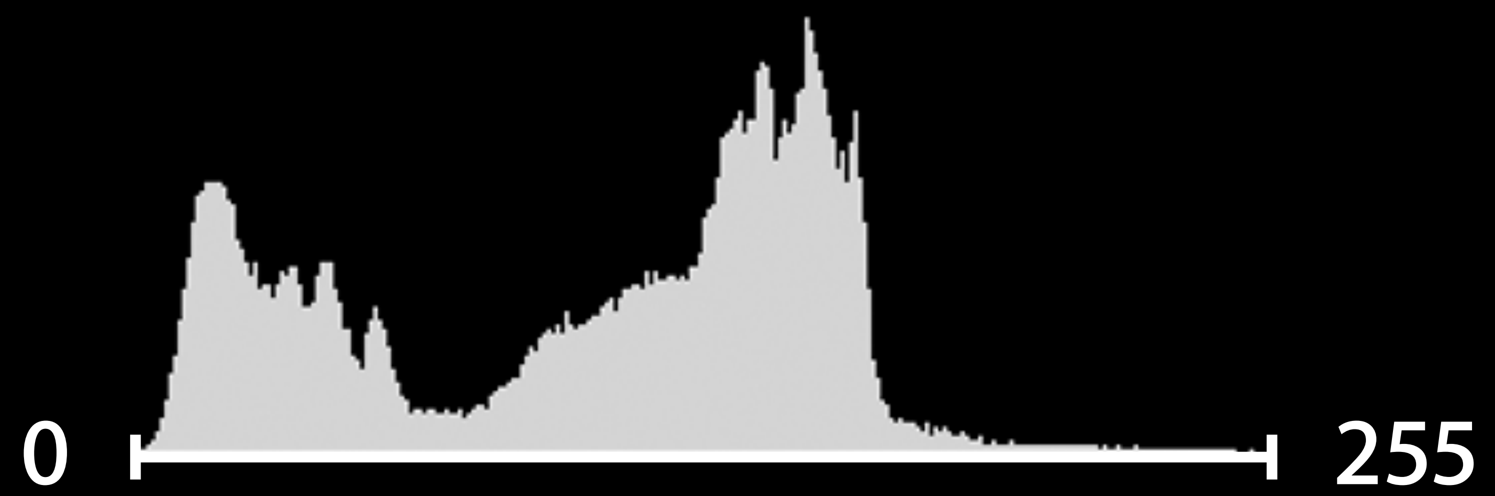


Red Histogram

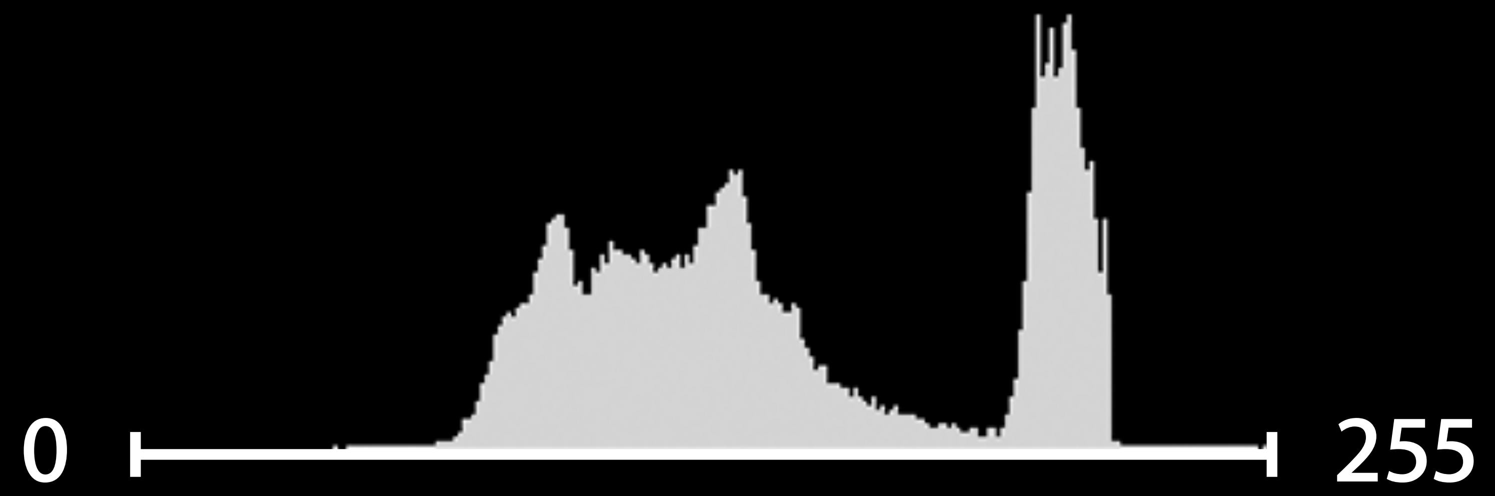
Green Histogram

Blue Histogram

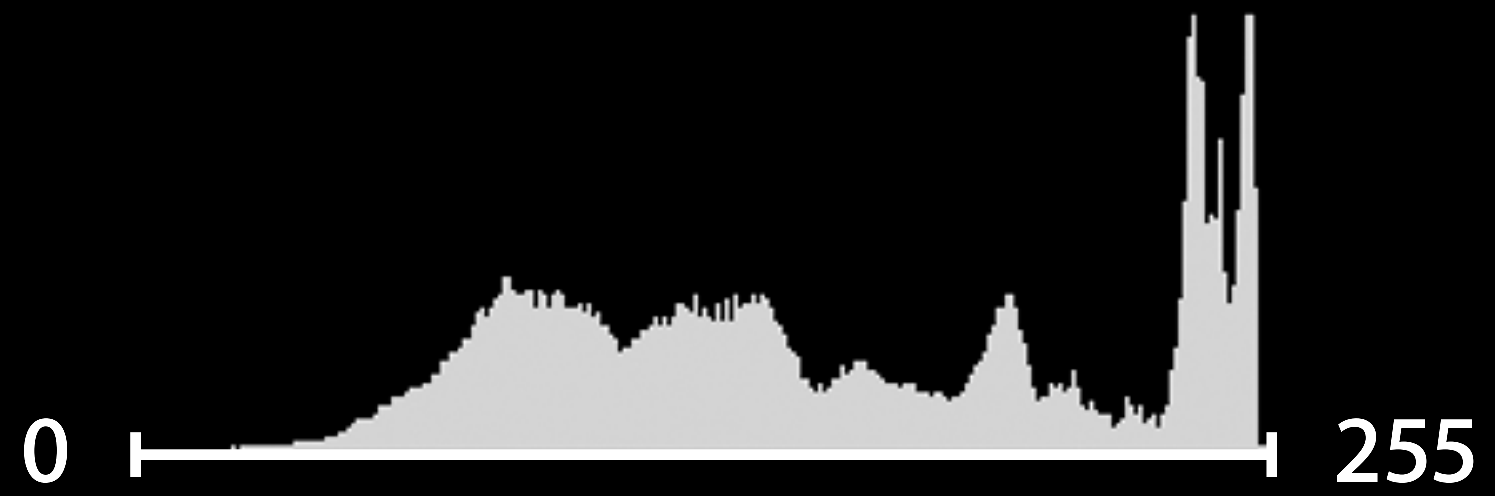




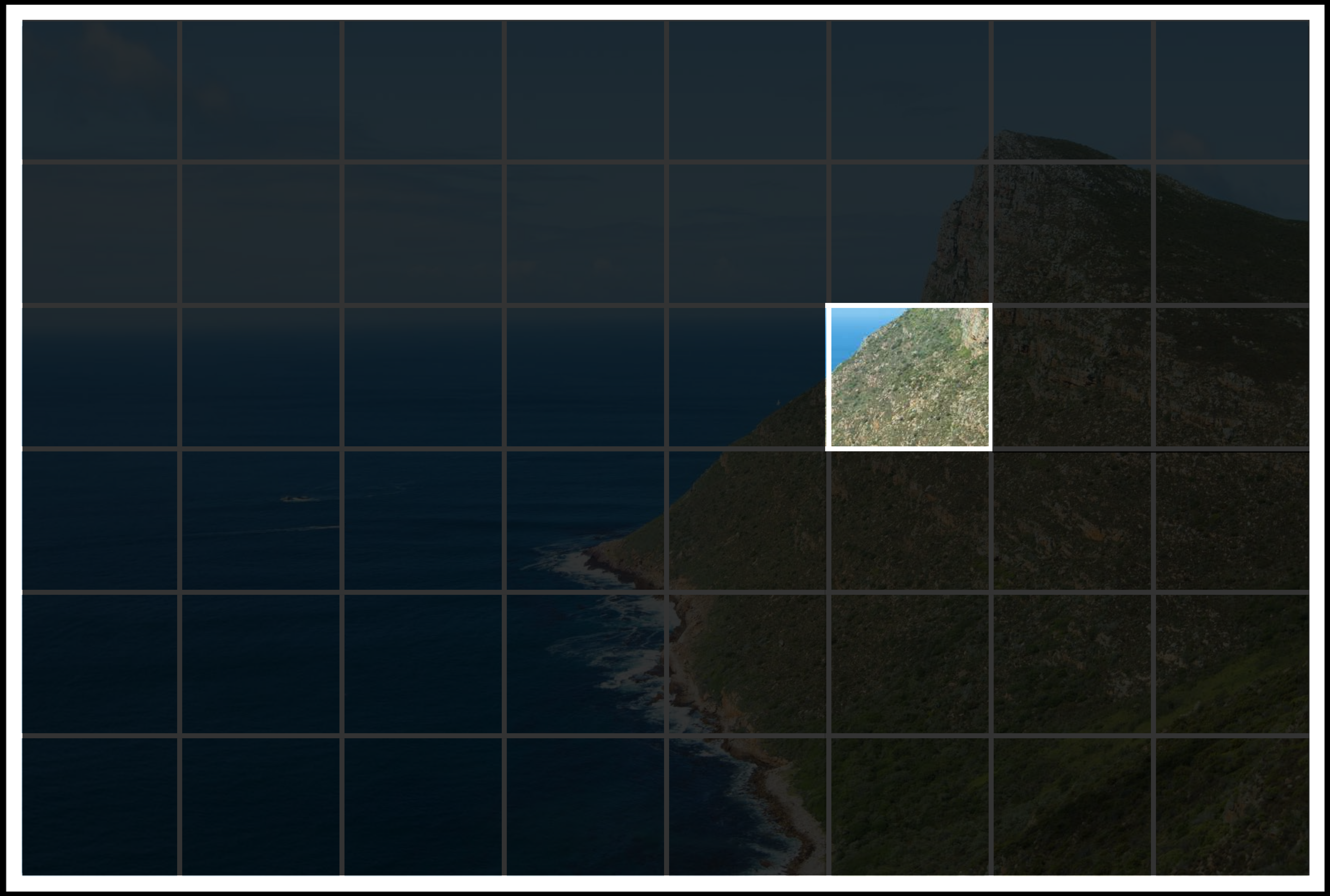
Red Histogram

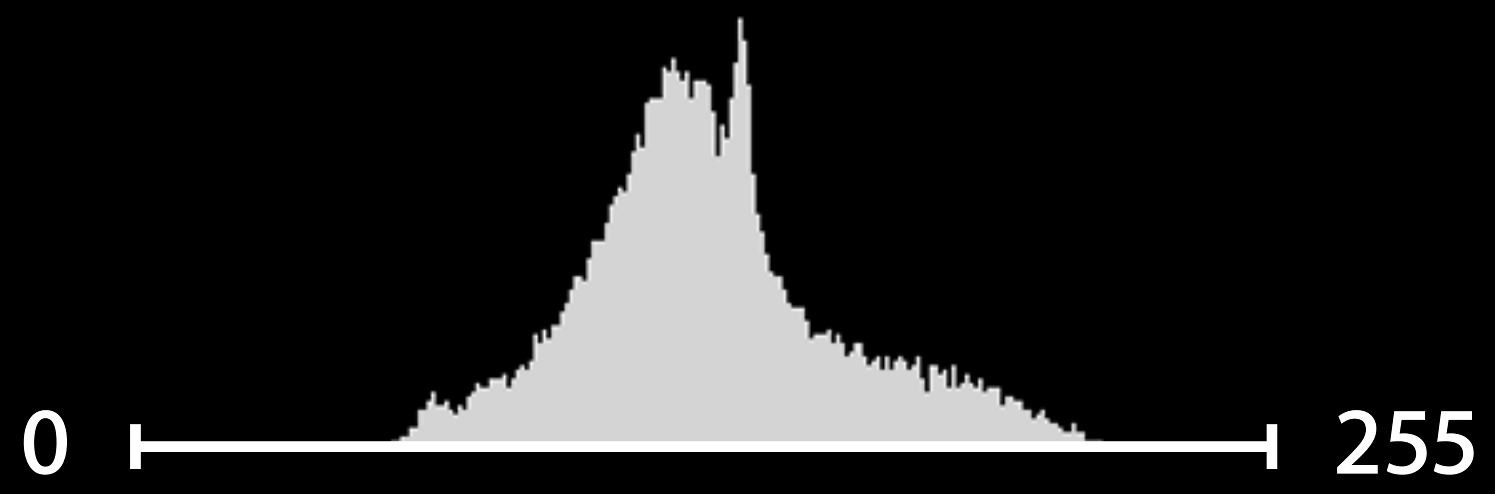


Green Histogram

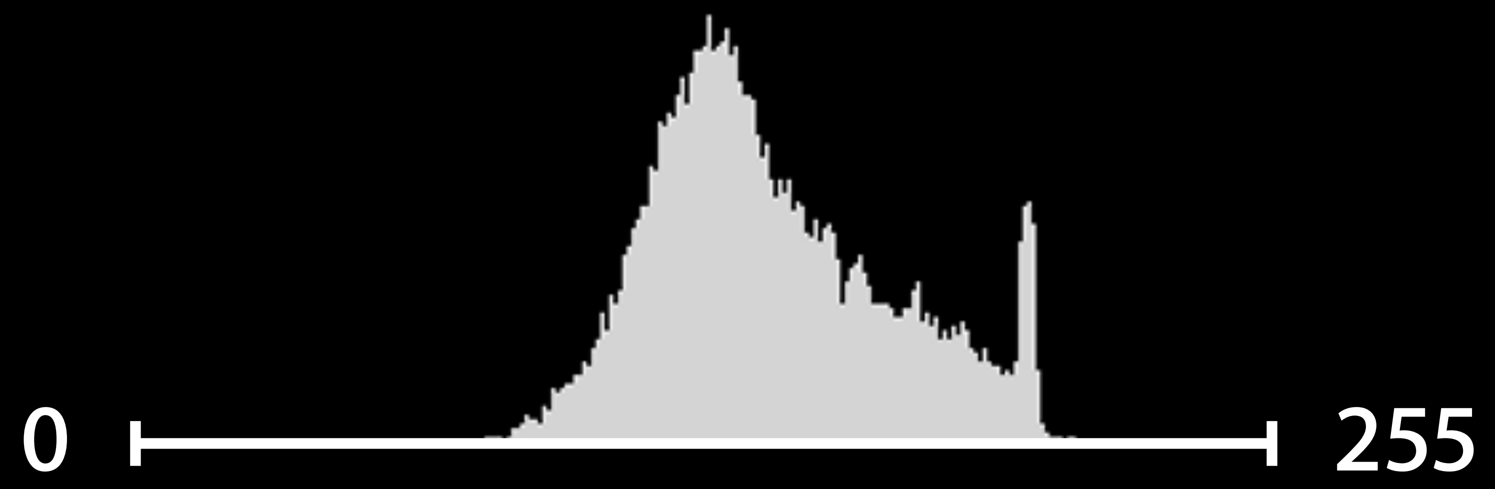


Blue Histogram

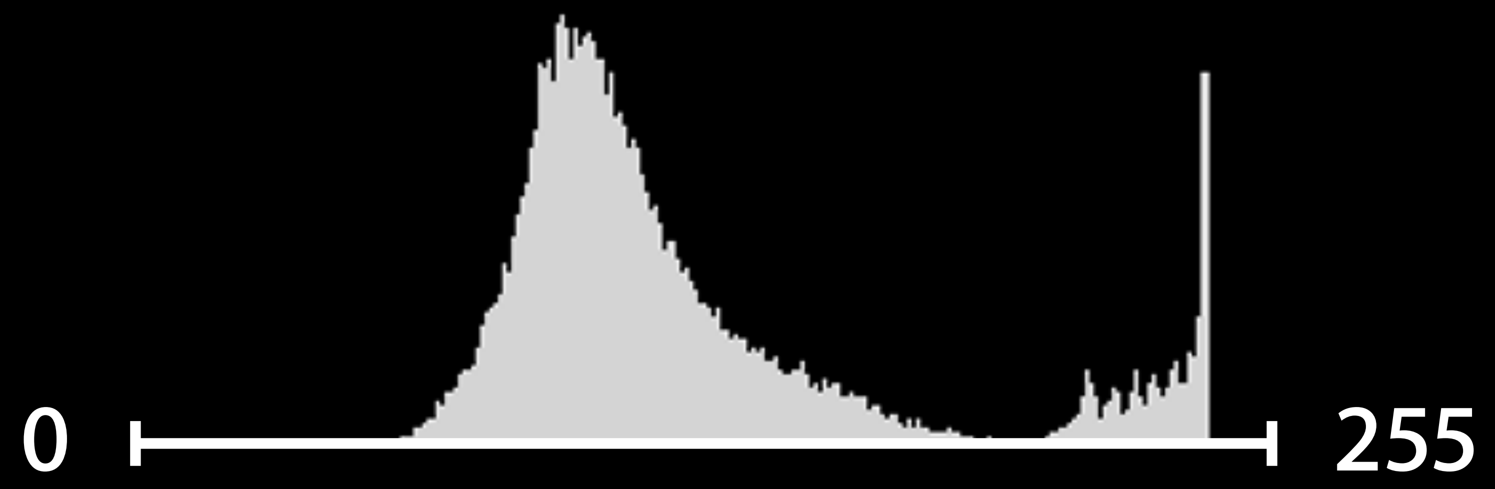




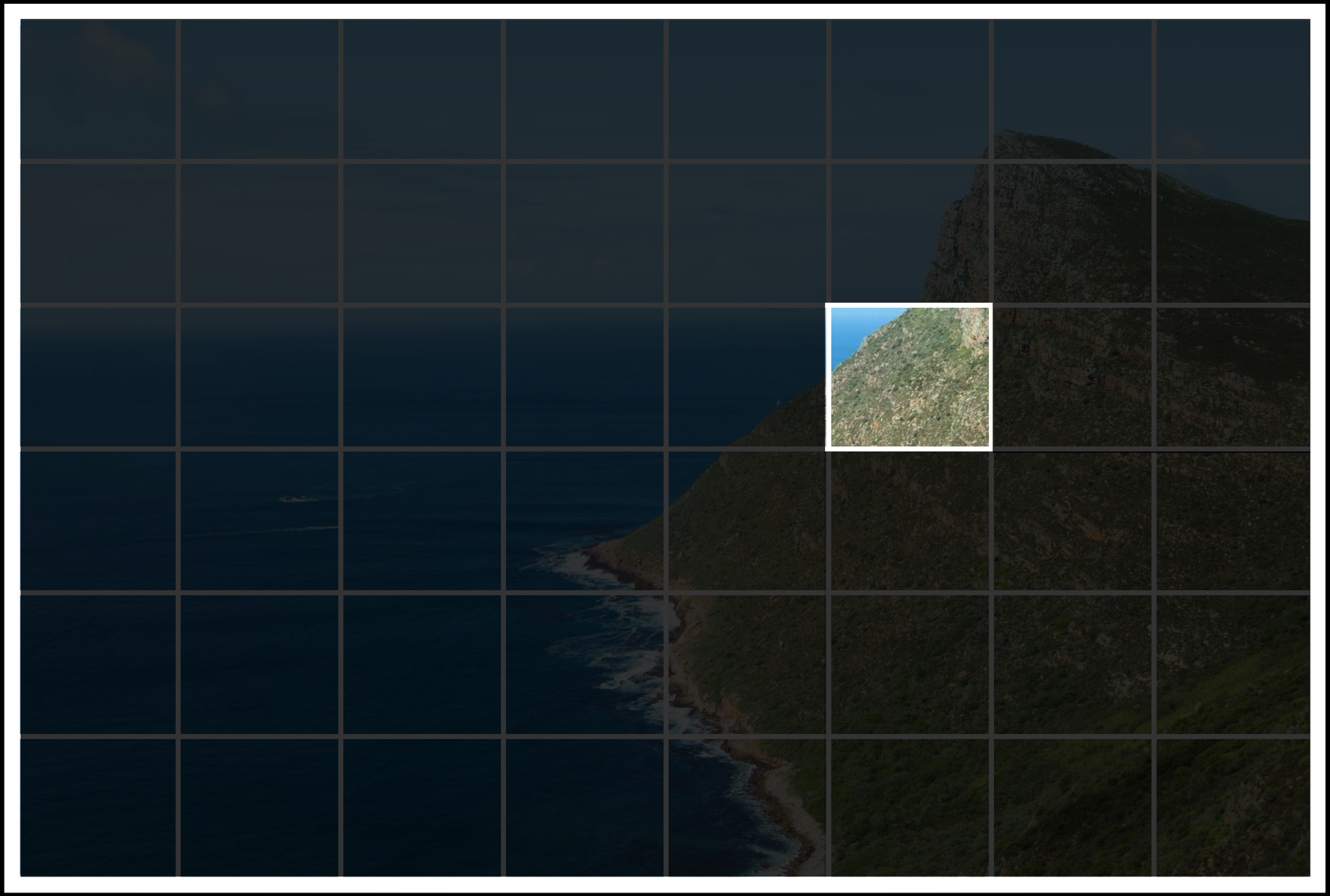
Partial Red Histogram



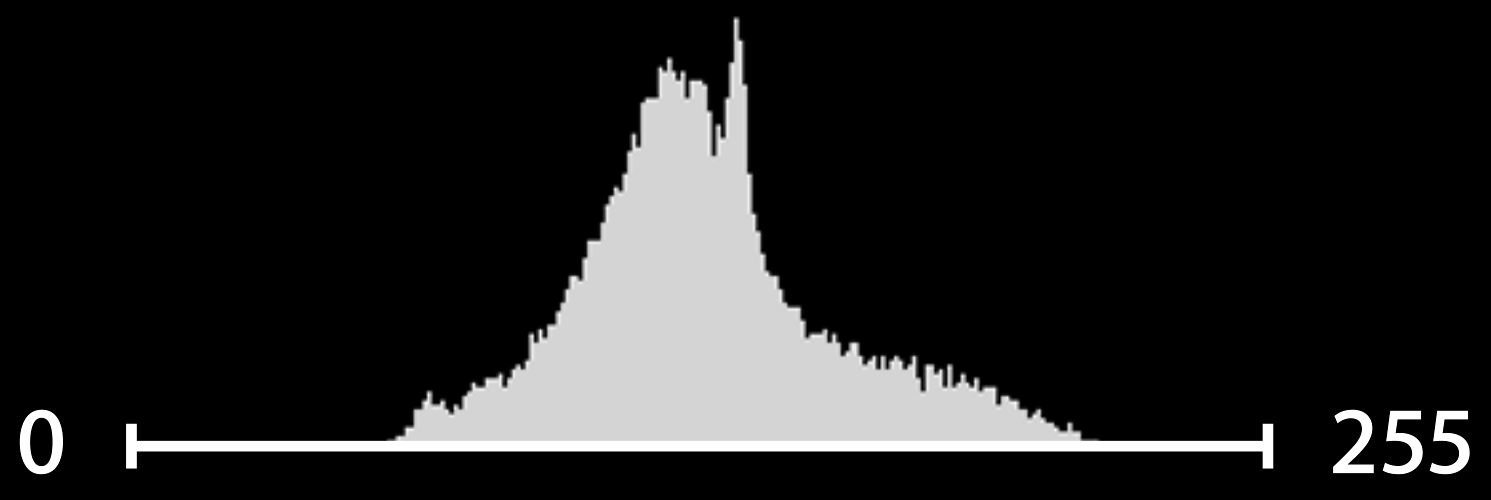
Partial Green Histogram



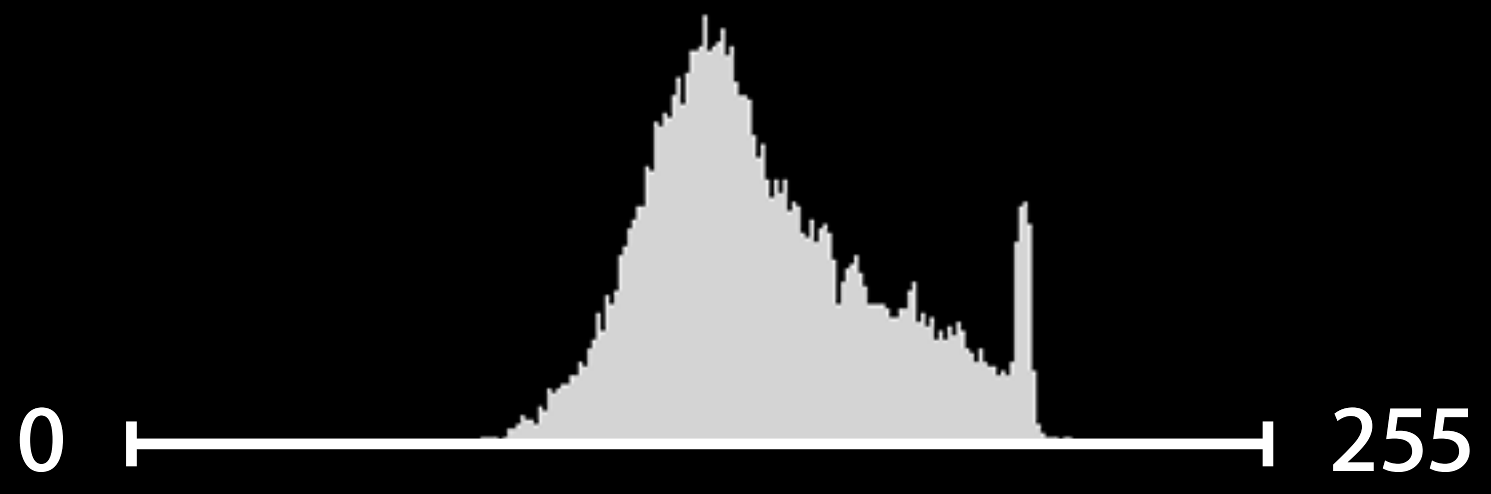
Partial Blue Histogram



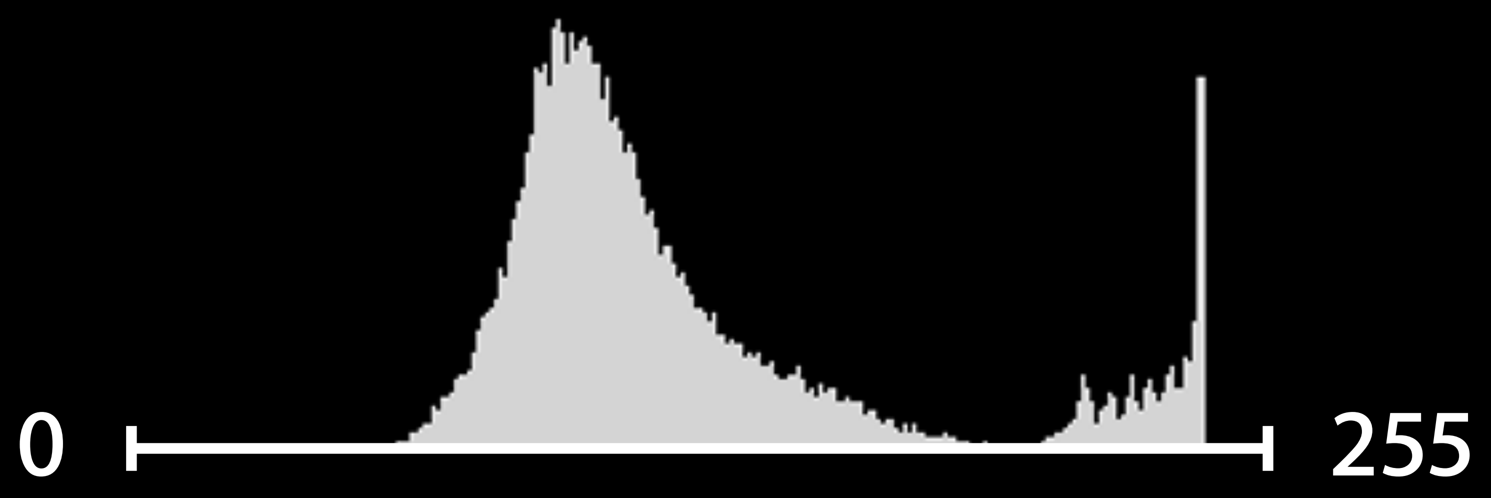
Collisions still exist, but only within a group



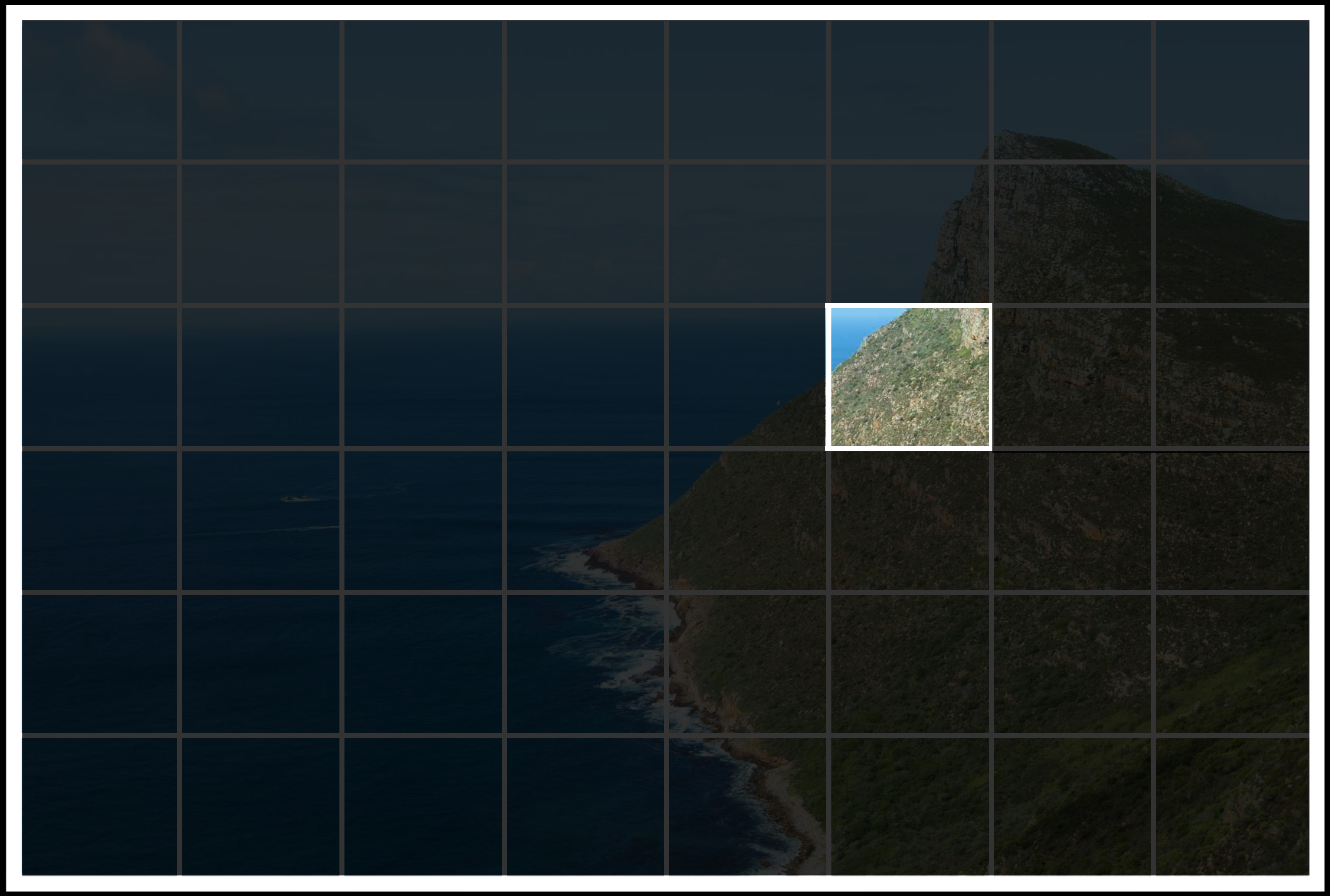
Partial Red Histogram



Partial Green Histogram

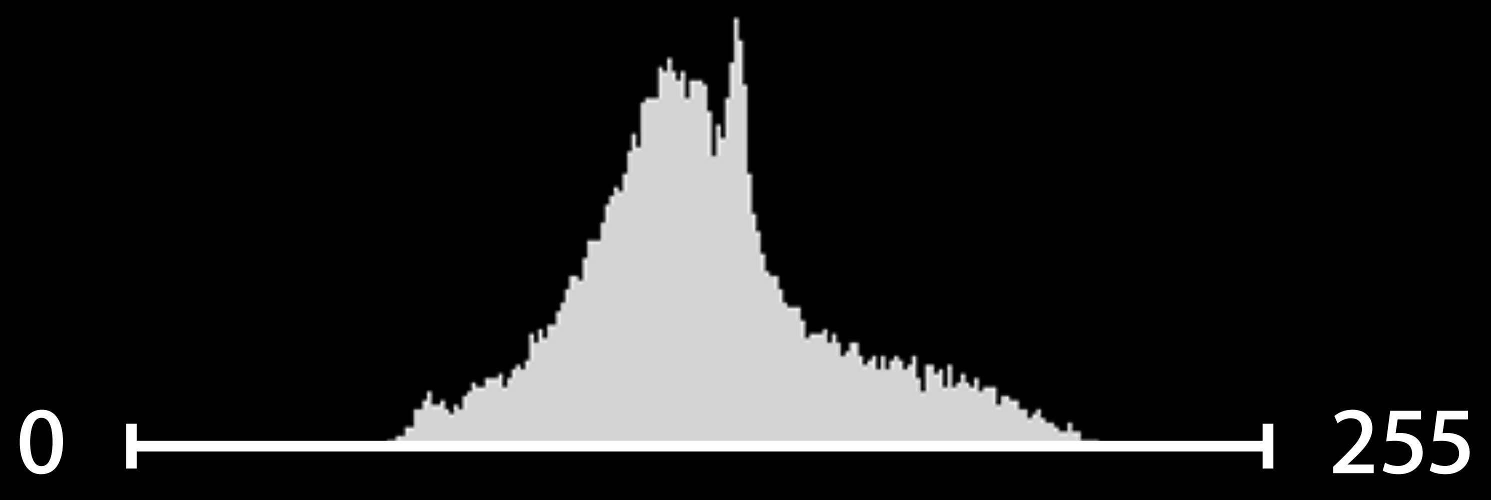


Partial Blue Histogram

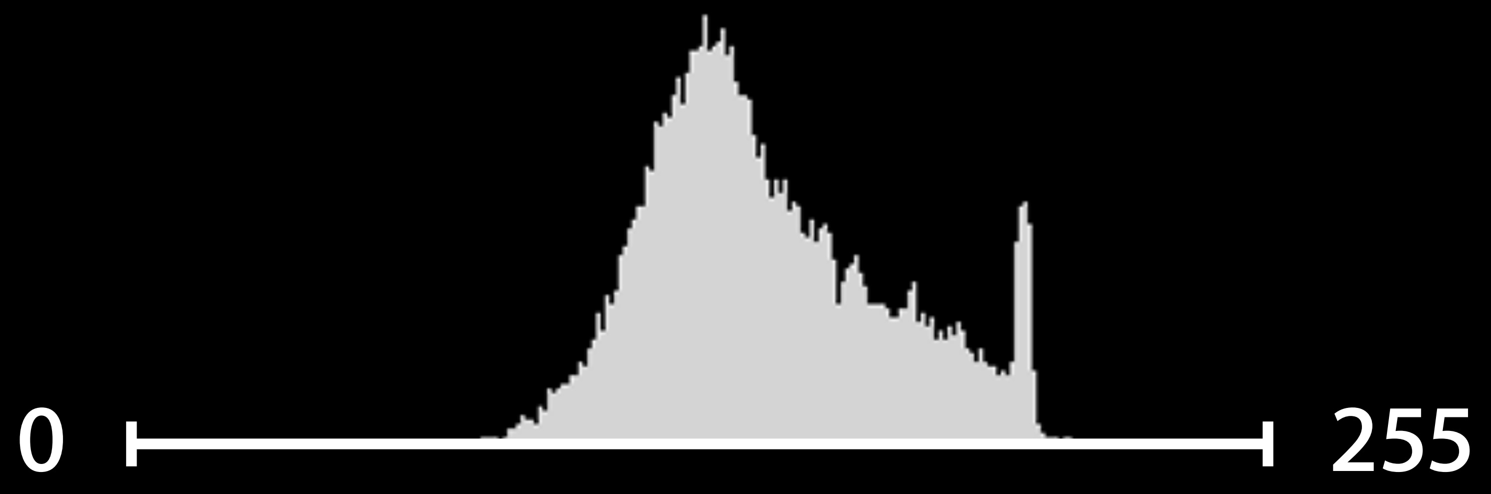


Collisions still exist, but only within a group

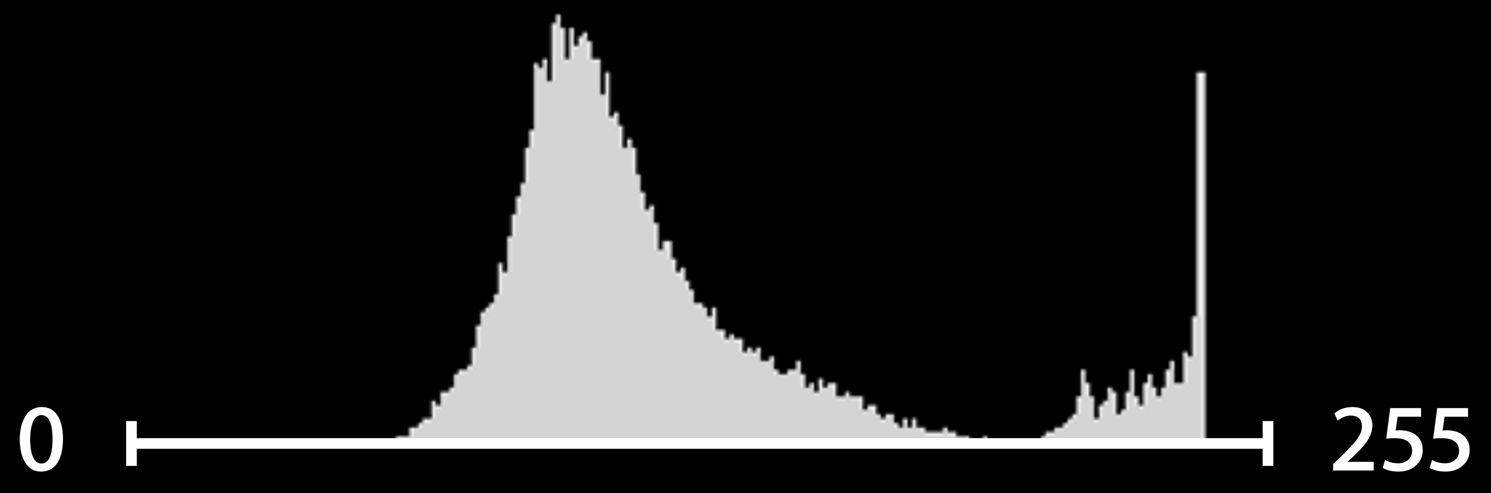
All groups run in parallel



Partial Red Histogram

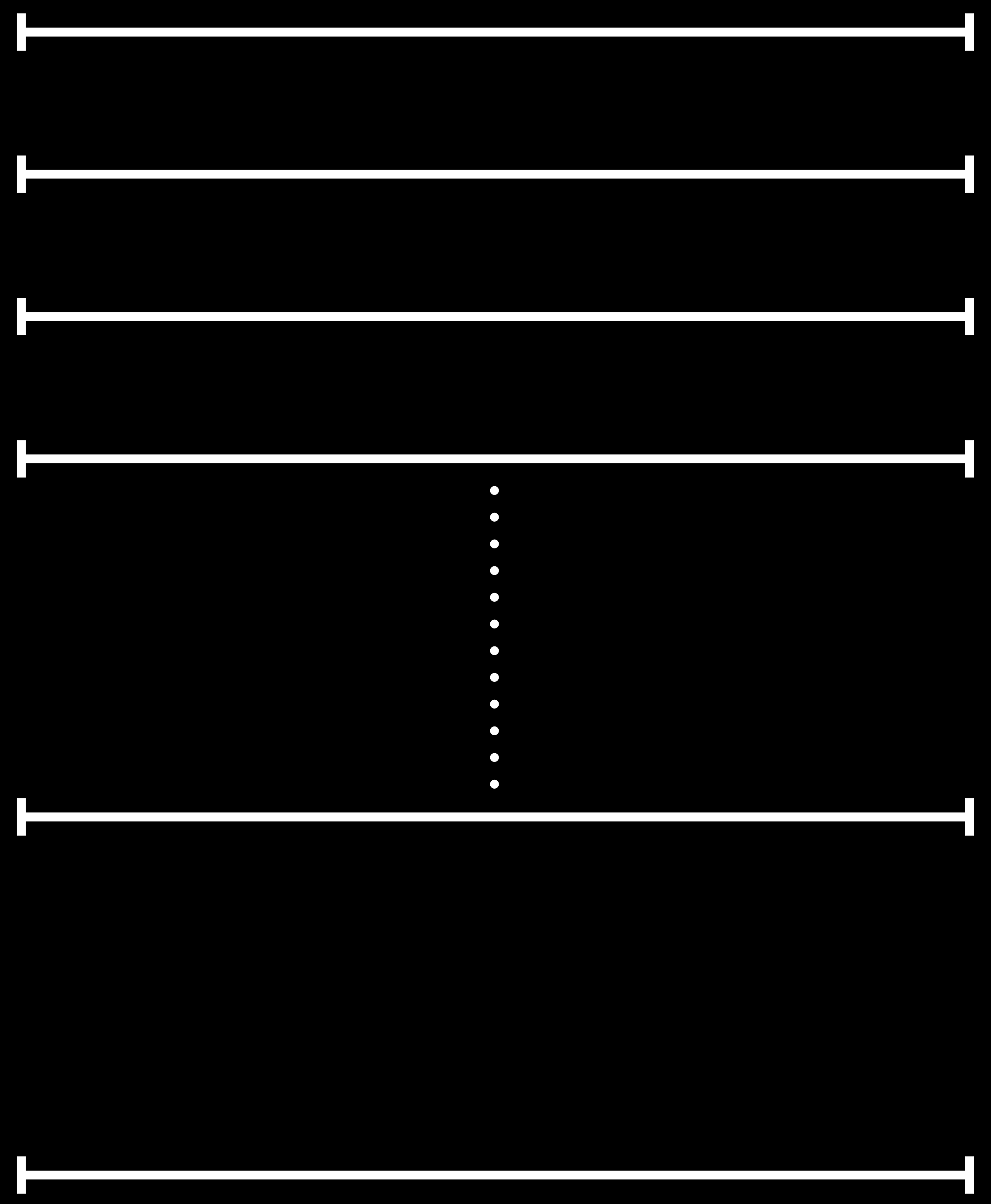


Partial Green Histogram



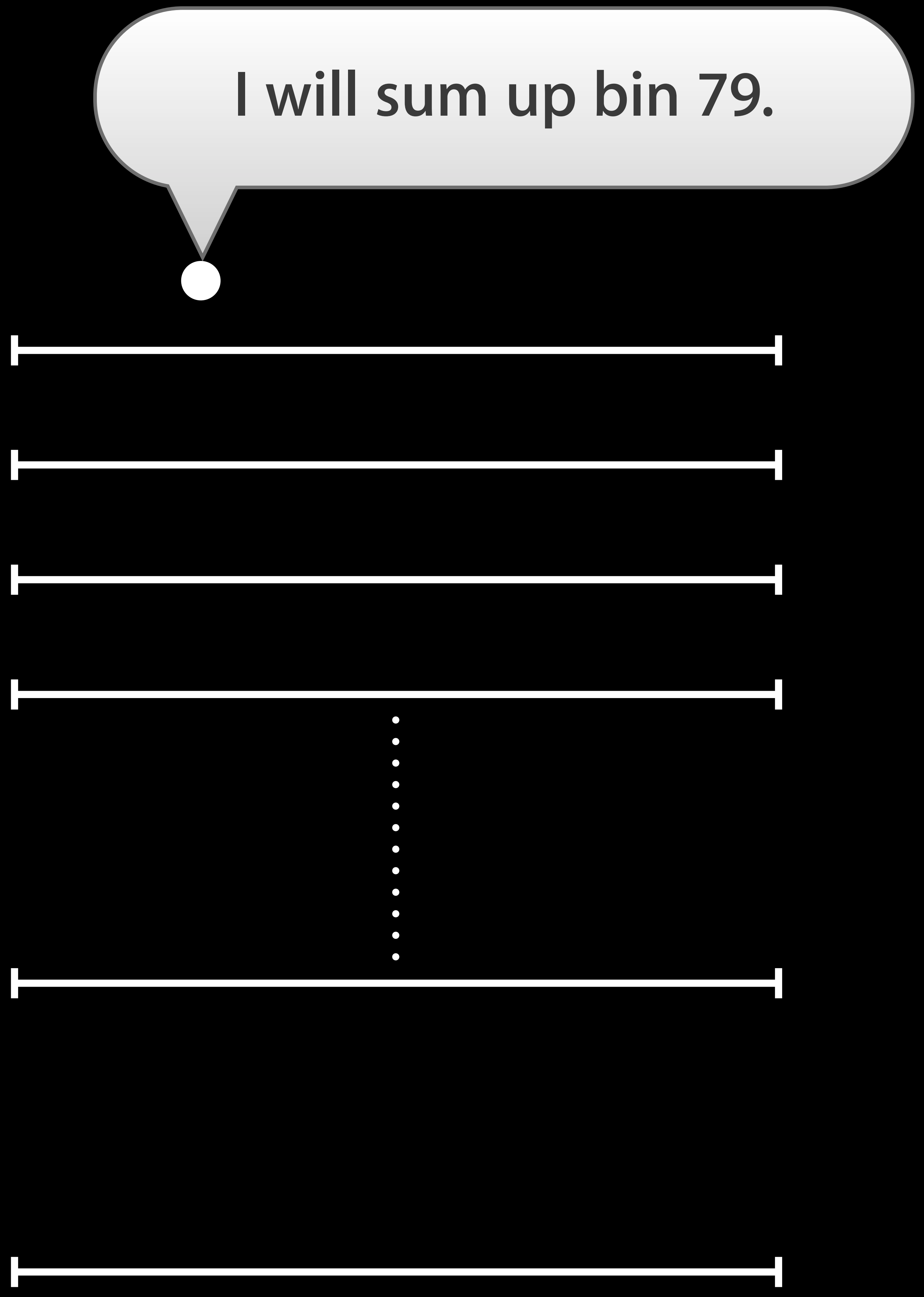
Partial Blue Histogram

Partial Red Histograms



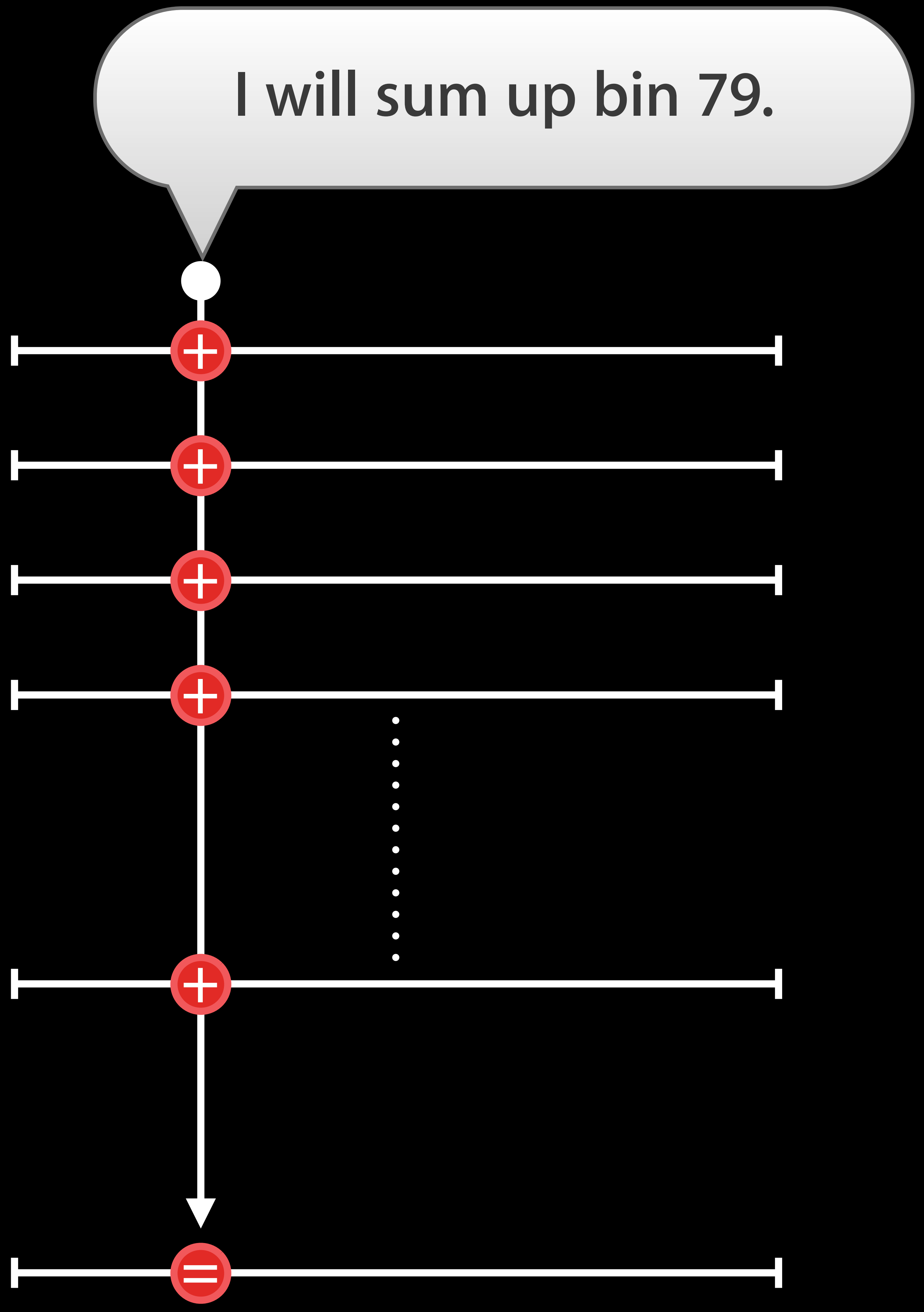
Total Red Histogram

Partial Red Histograms

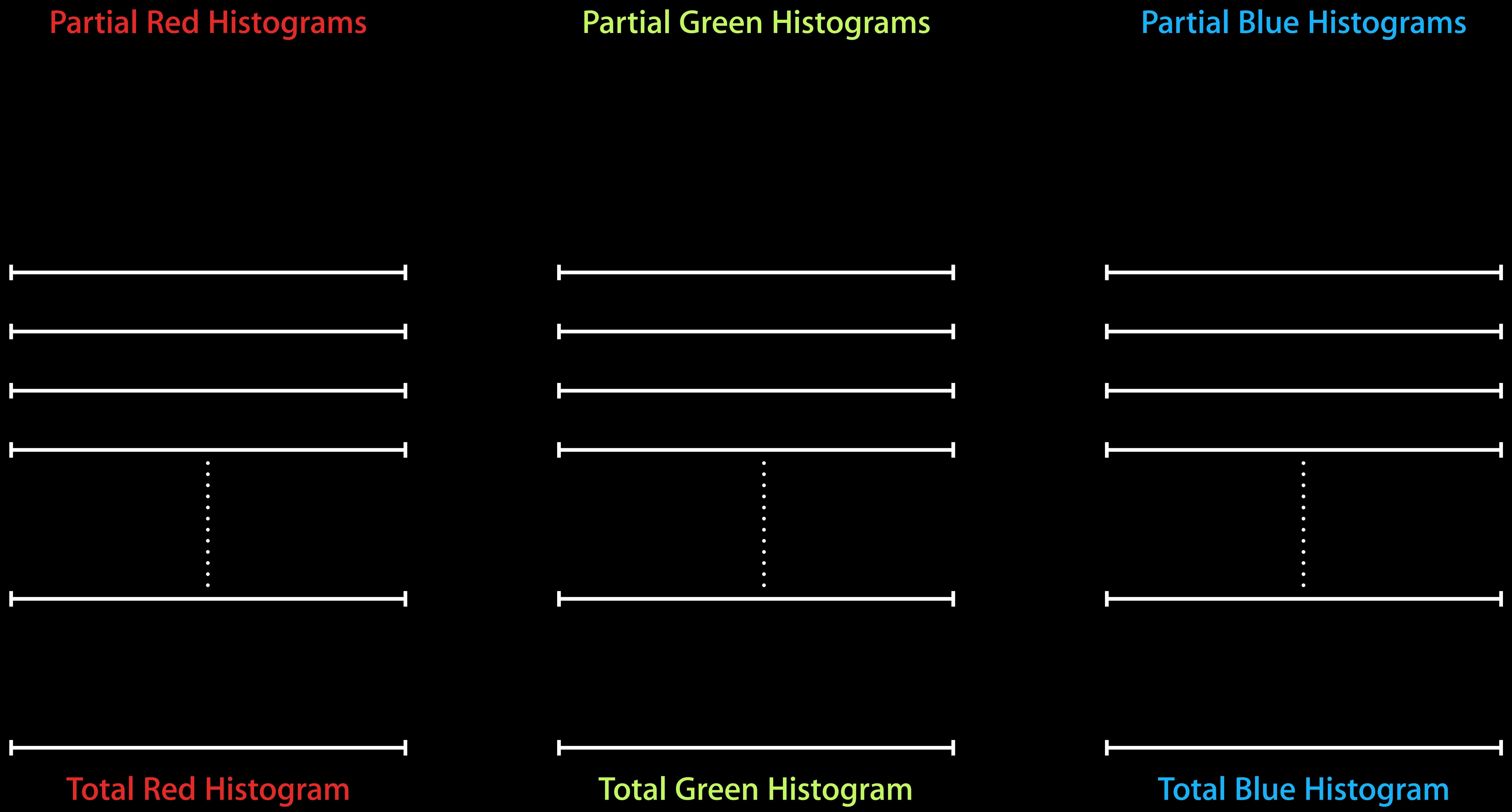


Total Red Histogram

Partial Red Histograms



Total Red Histogram



Partial Red Histograms

Partial Green Histograms

Partial Blue Histograms

Total Red Histogram

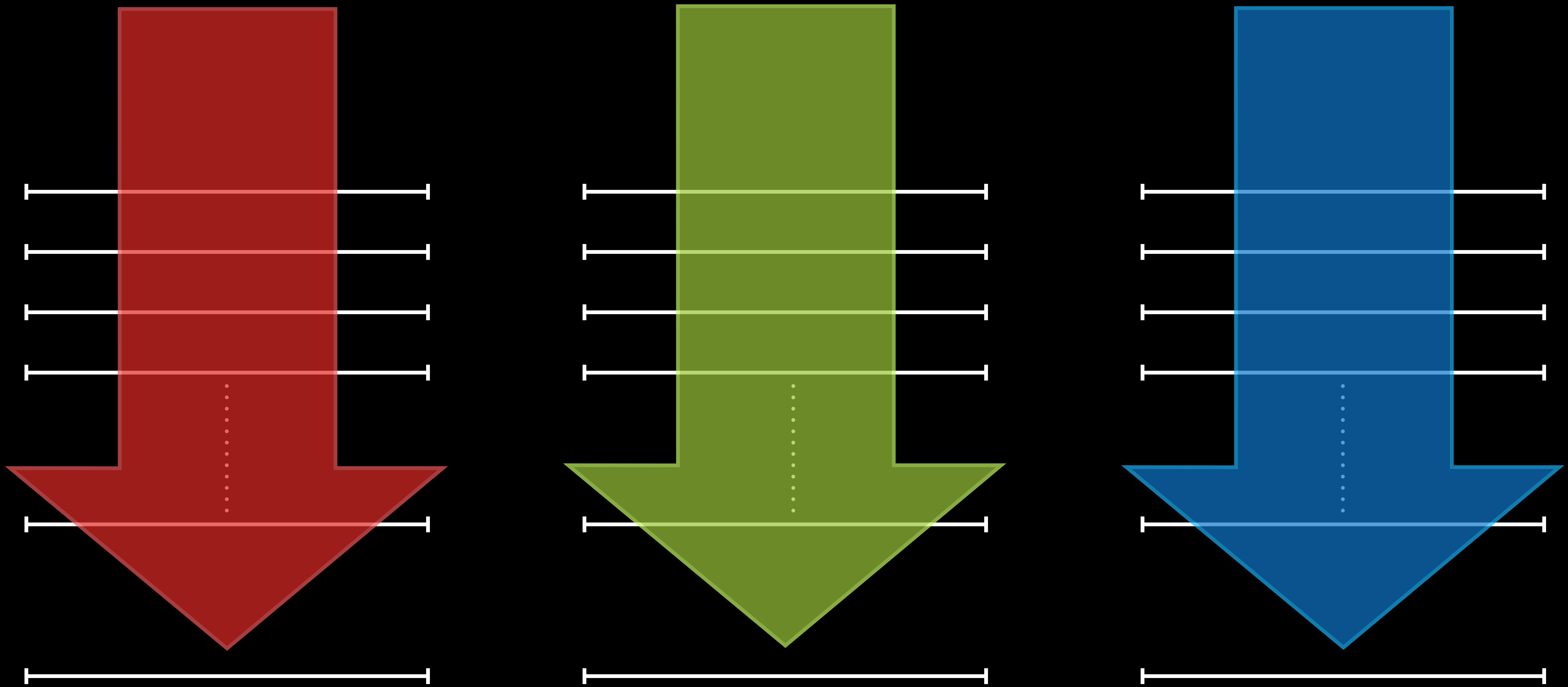
Total Green Histogram

Total Blue Histogram

Partial Red Histograms

Partial Green Histograms

Partial Blue Histograms



Total Red Histogram

Total Green Histogram

Total Blue Histogram

- □ Am I waiting for something?
- □ Have I avoided something intensive?
- □ Do I have a parallel workload?
- □ Can I earn a parallel workload?

- □ Am I waiting for something?

✓

- □ Have I avoided something intensive?
- □ Do I have a parallel workload?

✓

- □ Can I earn a parallel workload?

- □ Am I waiting for something?

✓

- □ Have I avoided something intensive?
- □ Do I have a parallel workload?

✓

- □ Can I earn a parallel workload?

OpenCL

□ Am I waiting for something?

✓

□ Have I avoided something intensive?

□ Do I have a parallel workload?

✓

□ Can I earn a parallel workload?

OpenCL

CPU or GPU ?

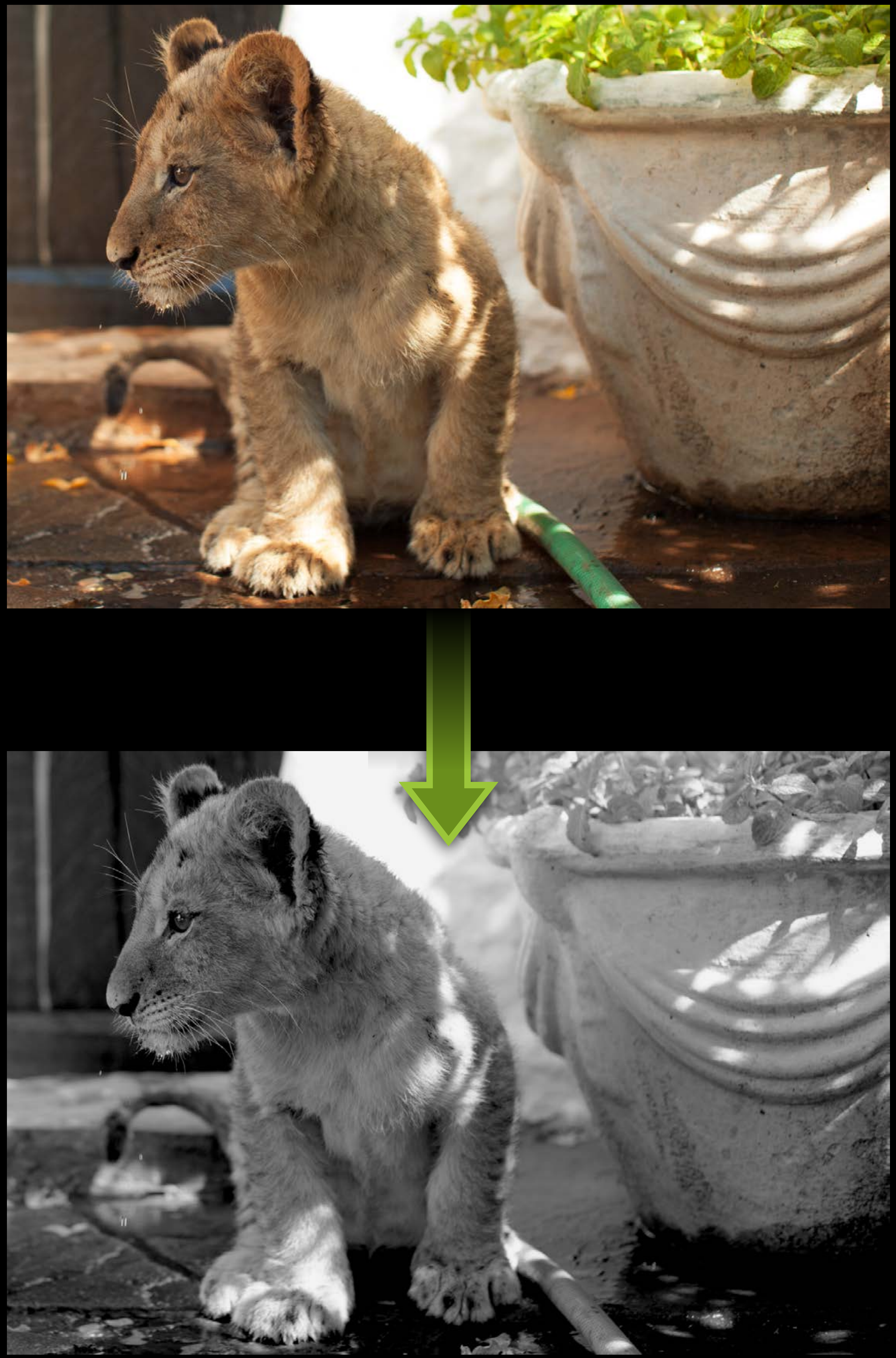
- □ Am I waiting for something?
- □ Have I avoided something intensive?
- □ Do I have a parallel workload?
- □ Can I earn a parallel workload?
- □ Where is my data now?
- □ Where is my data destined?
- □ How hard am I working?

OpenCL

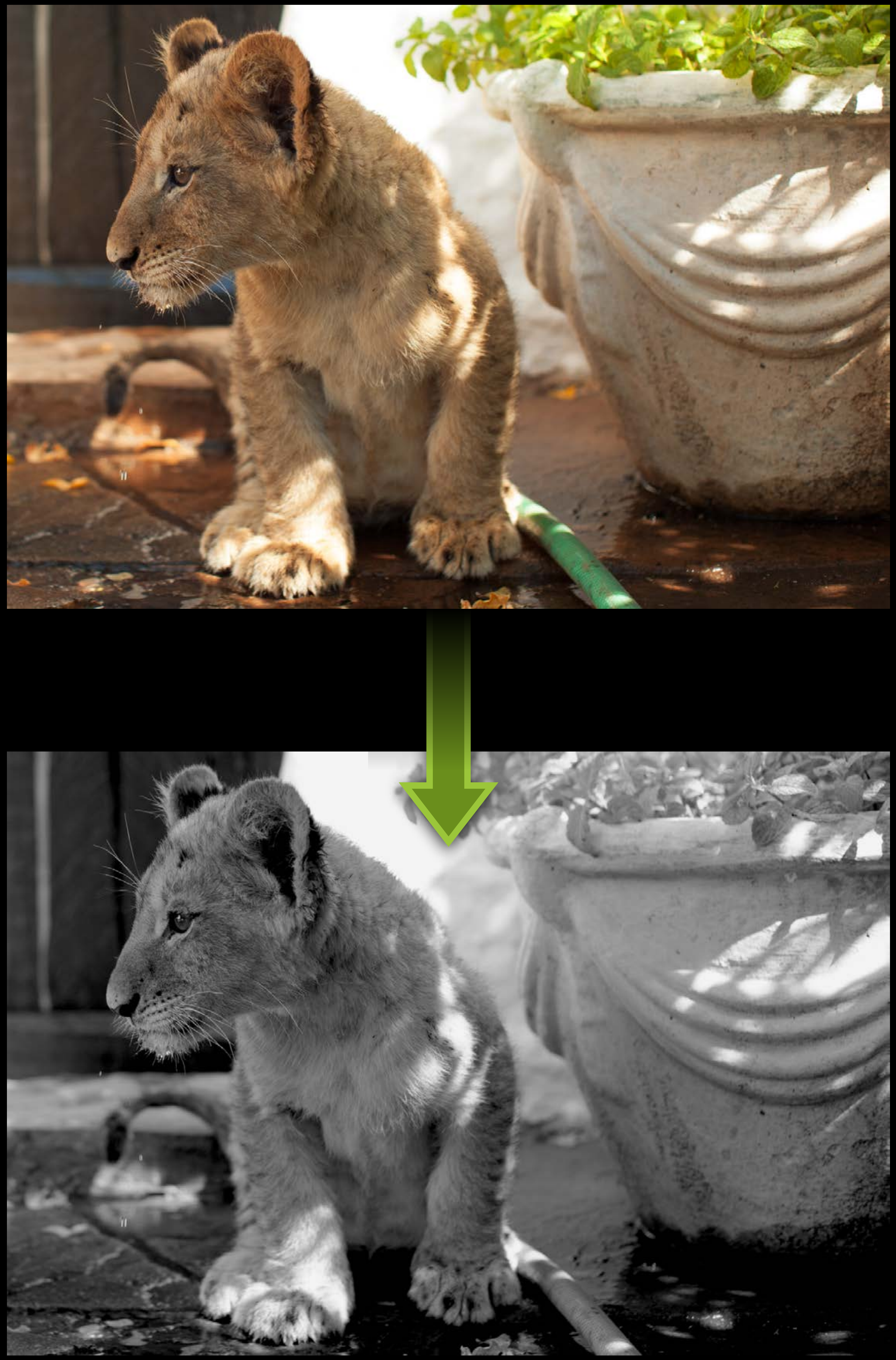
CPU or GPU ?

# Host

# Host



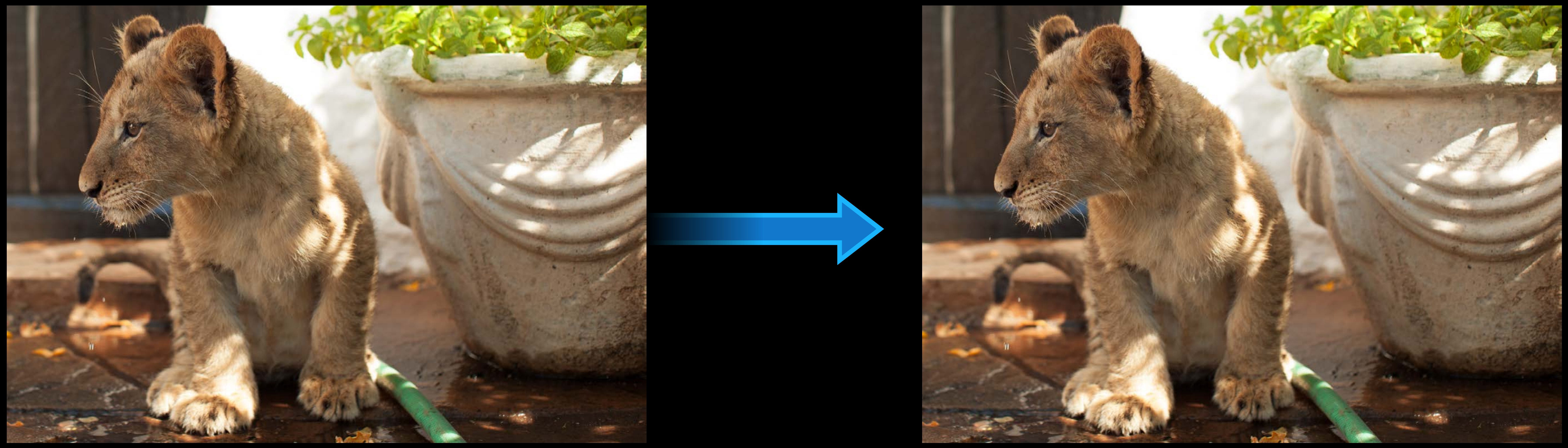
Host



Total time = compute

Host

OpenCL Device

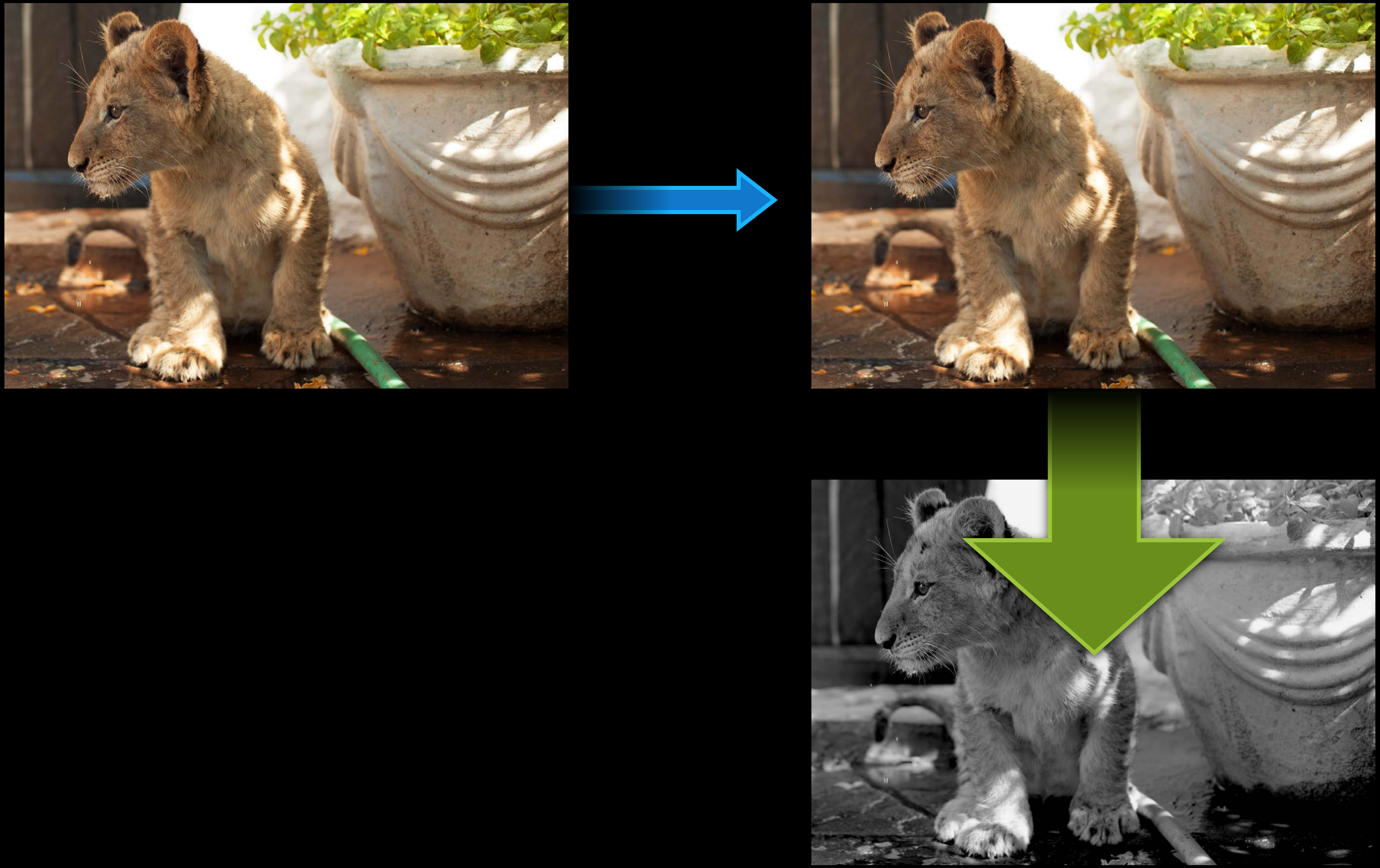


Host

OpenCL Device

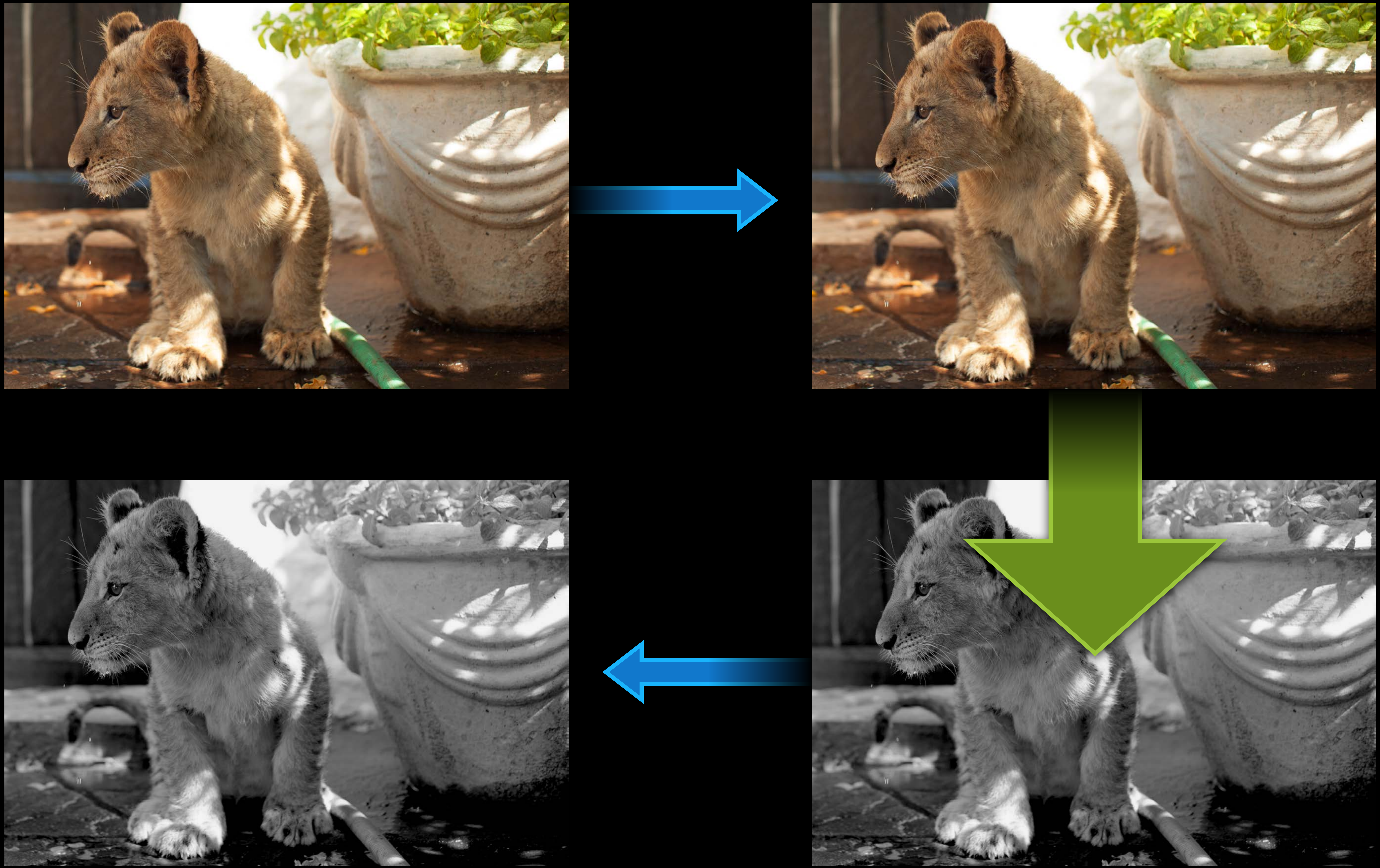
Host

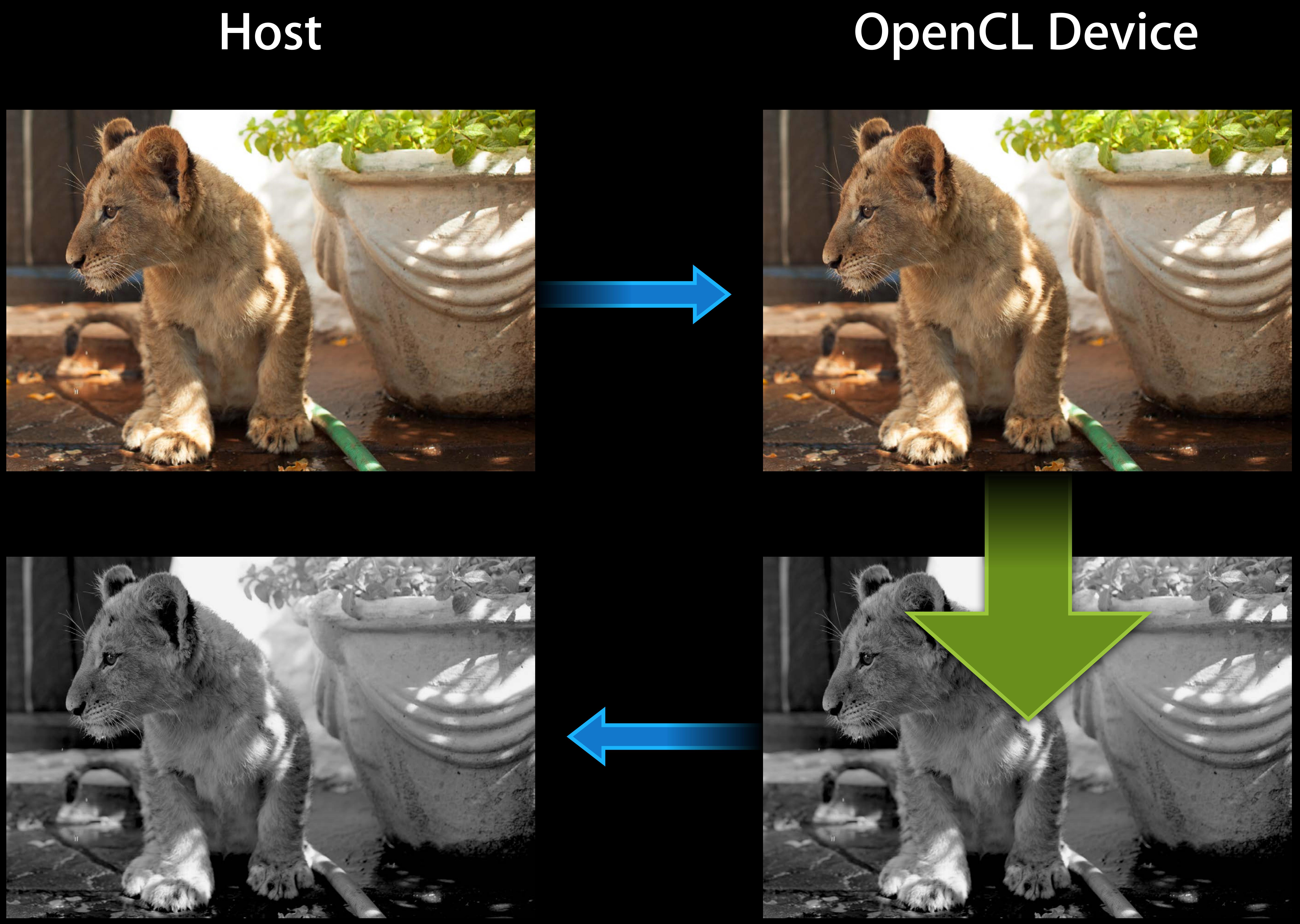
OpenCL Device



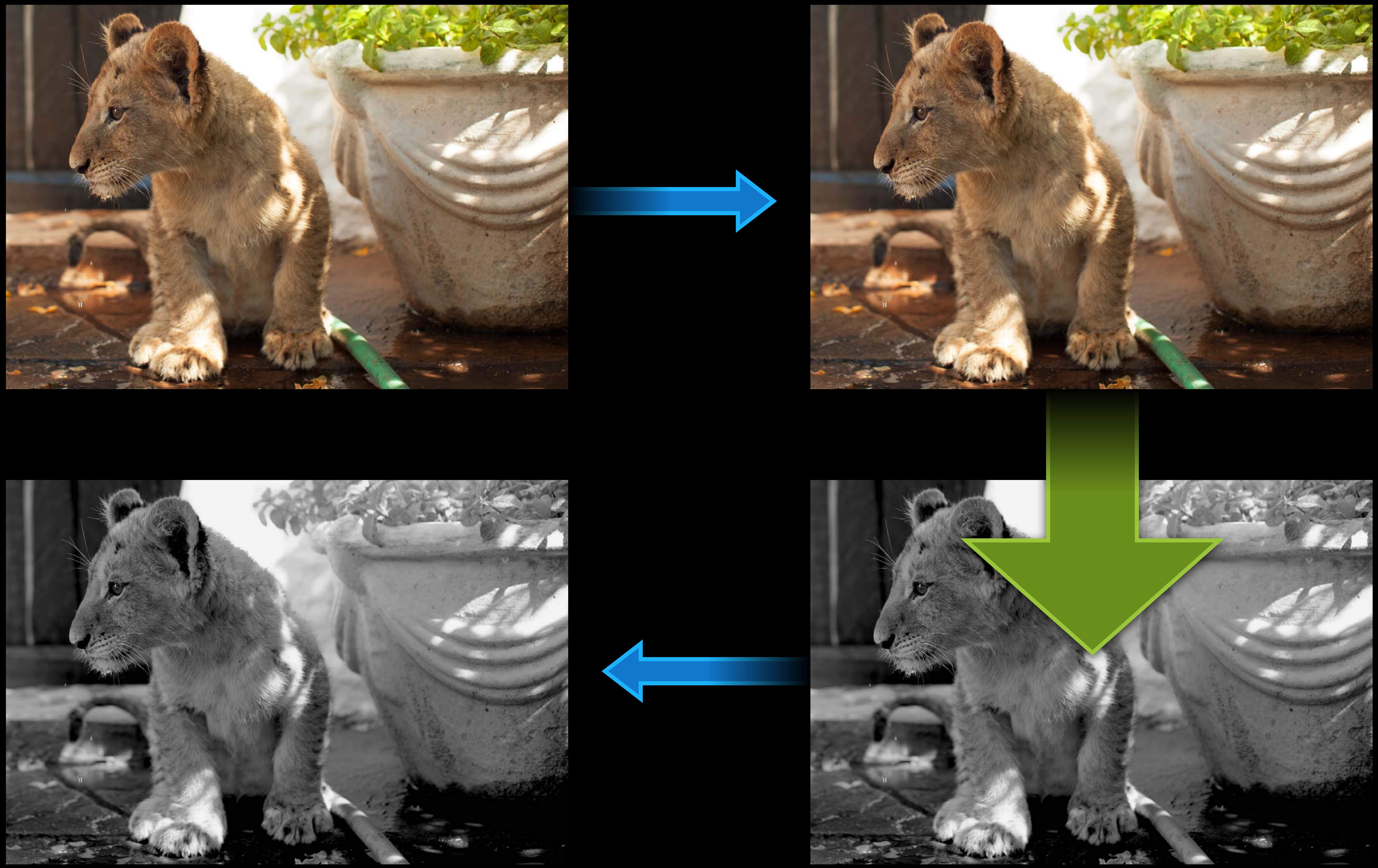
Host

OpenCL Device





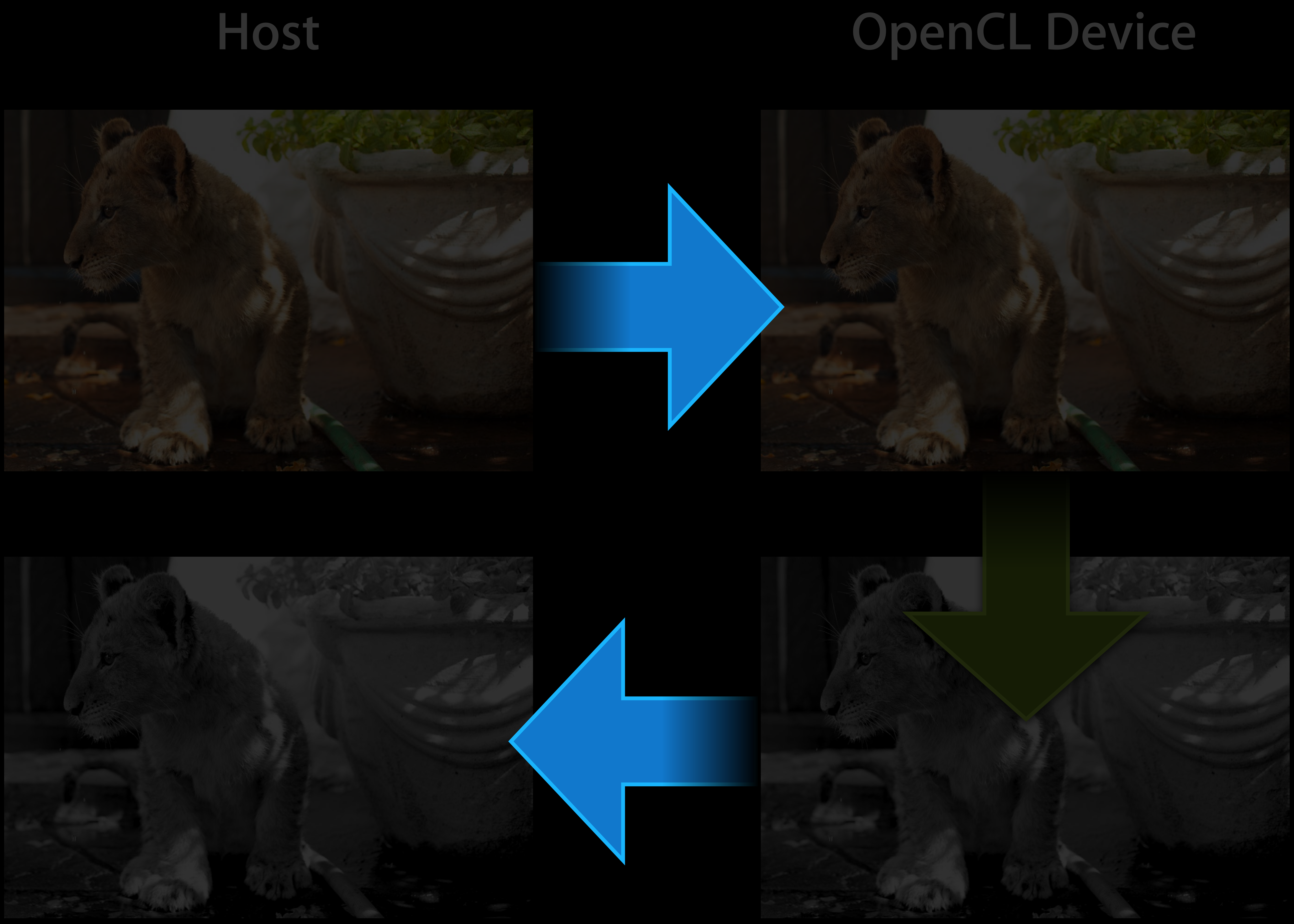
Total time = compute



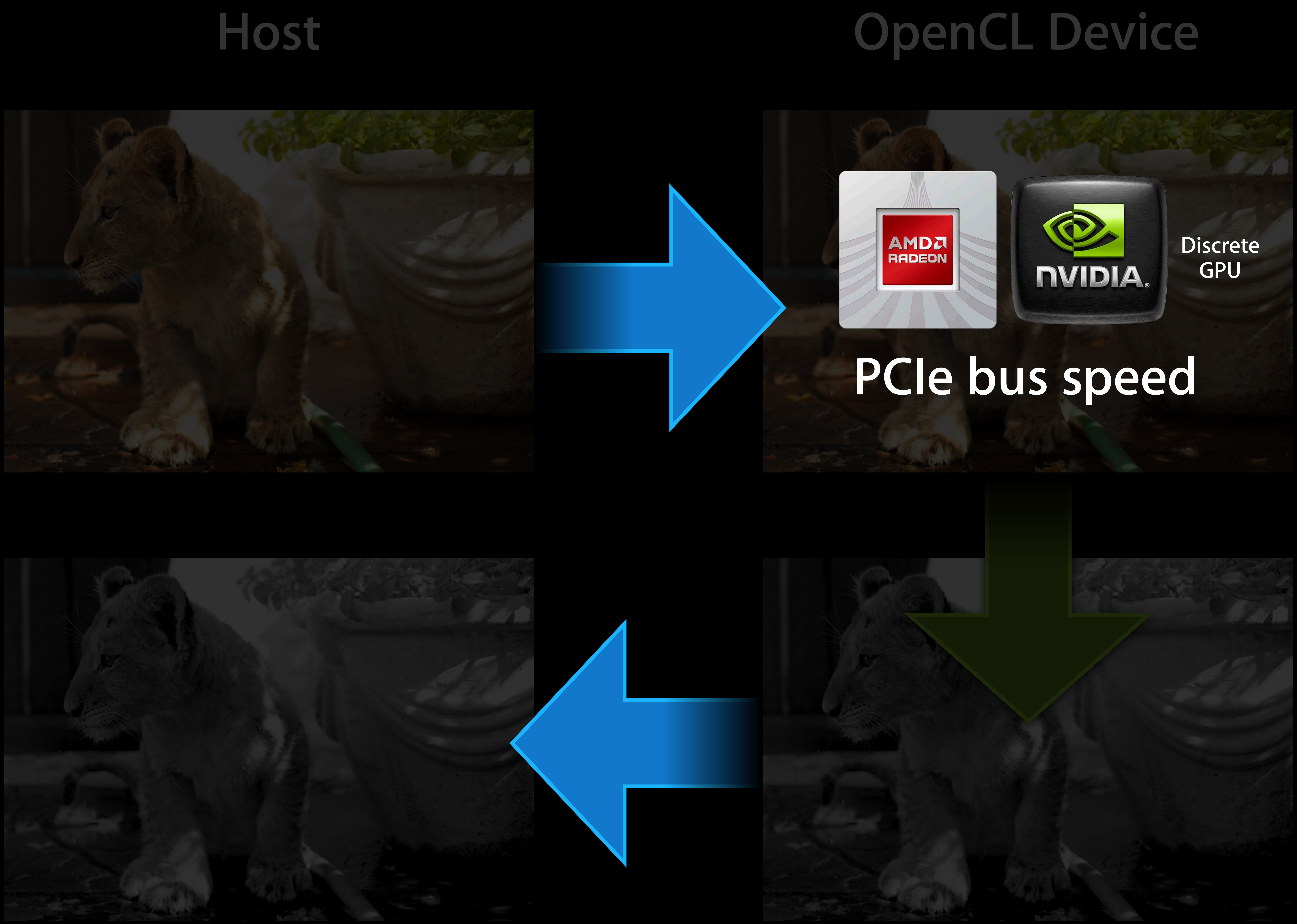
Host

OpenCL Device

Total time = compute + transfer time



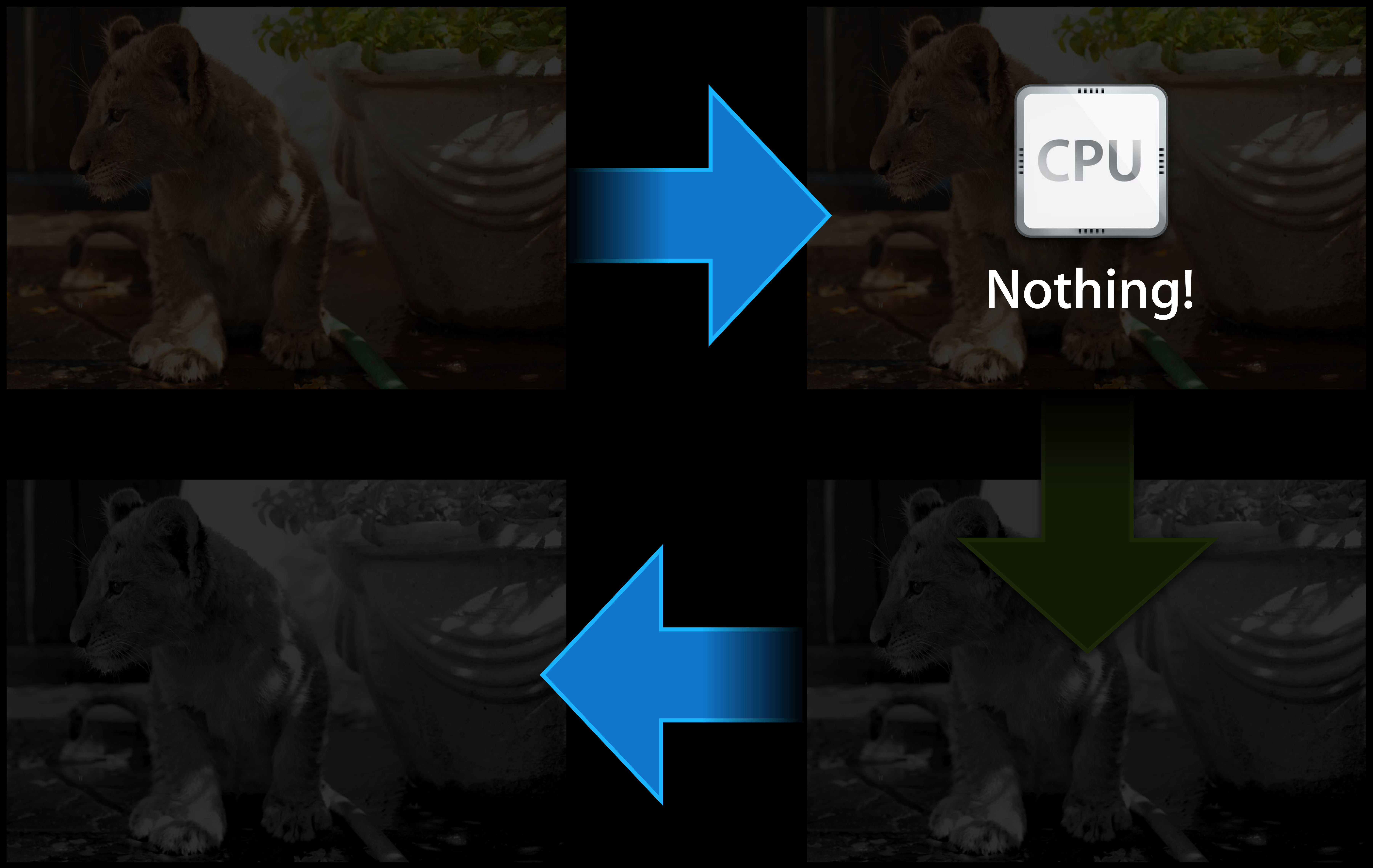
Total time = compute + **transfer time**



Total time = compute + transfer time

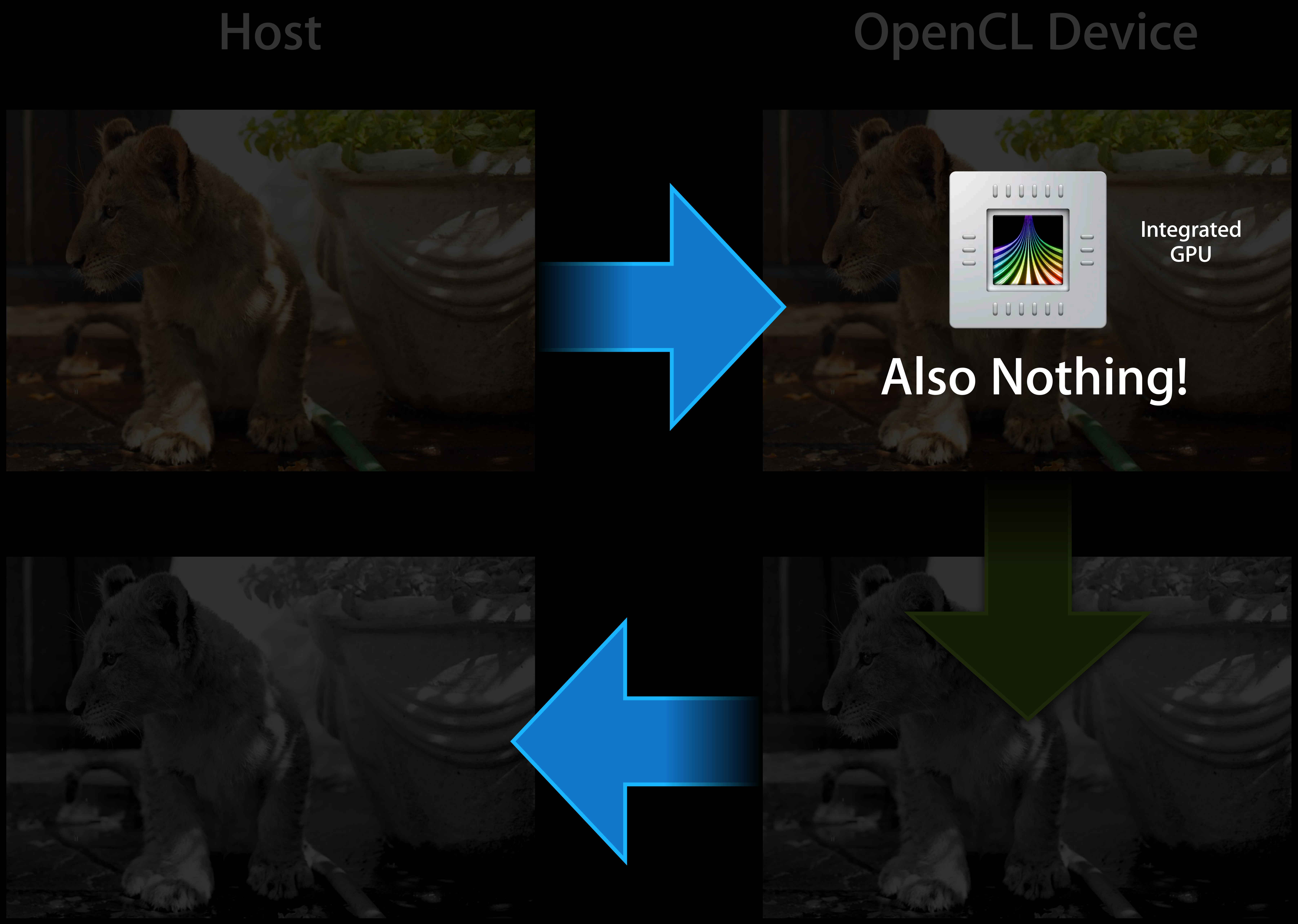
Host

OpenCL Device



Nothing!

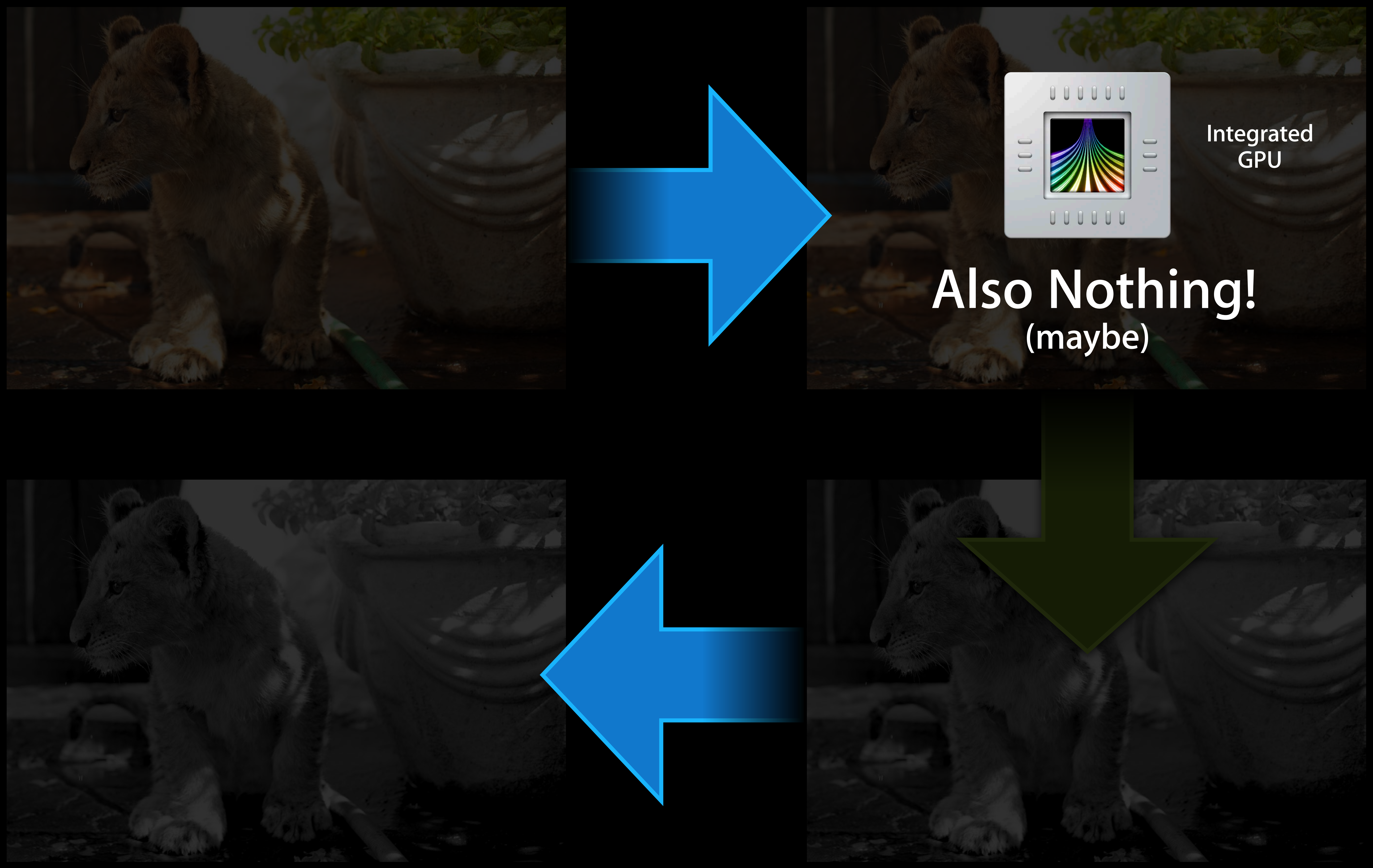
Total time = compute + transfer time



Total time = compute + transfer time

Host

OpenCL Device



Integrated GPU

**Also Nothing!**

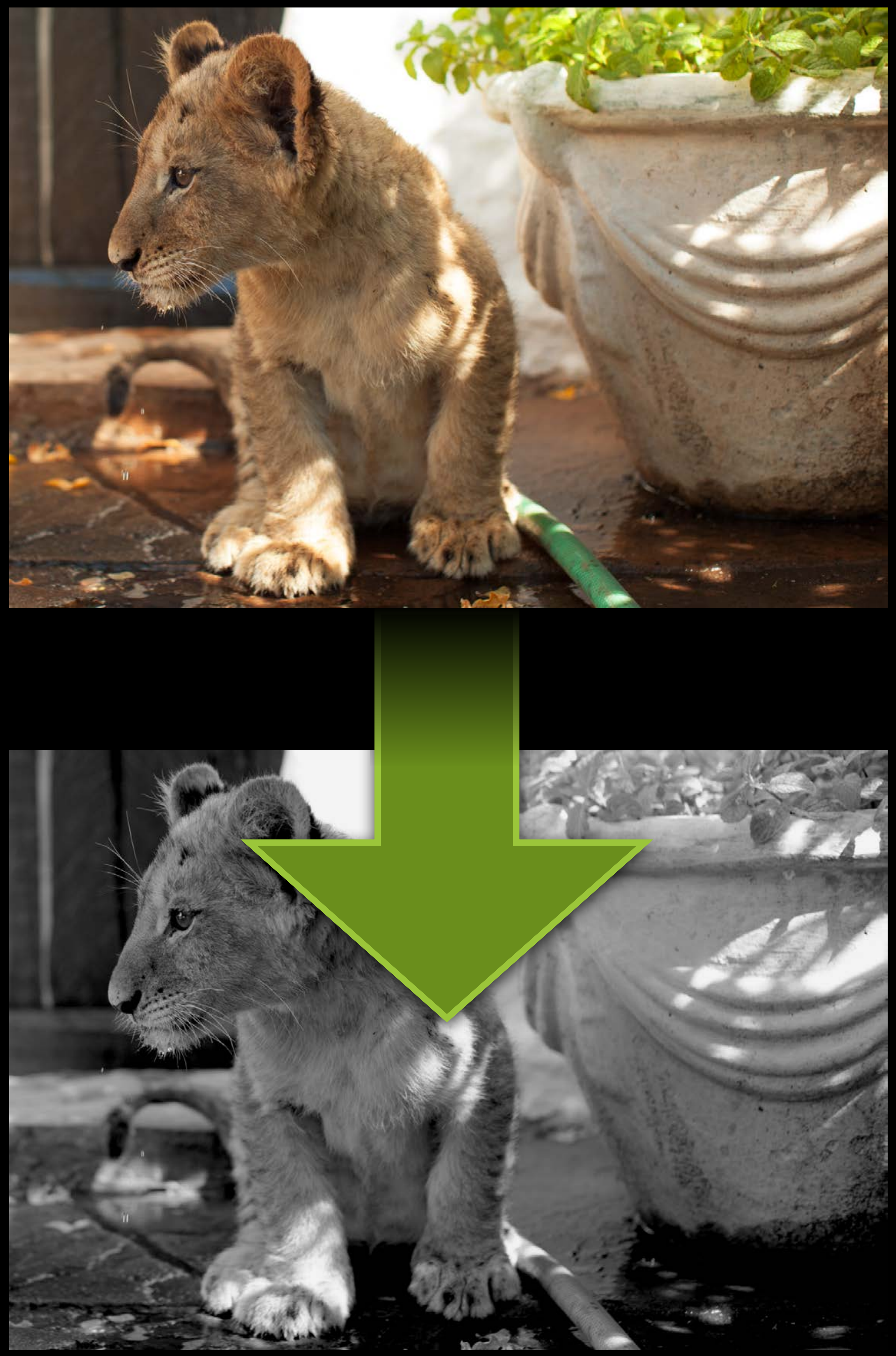
(maybe)

Total time = compute + **transfer time**

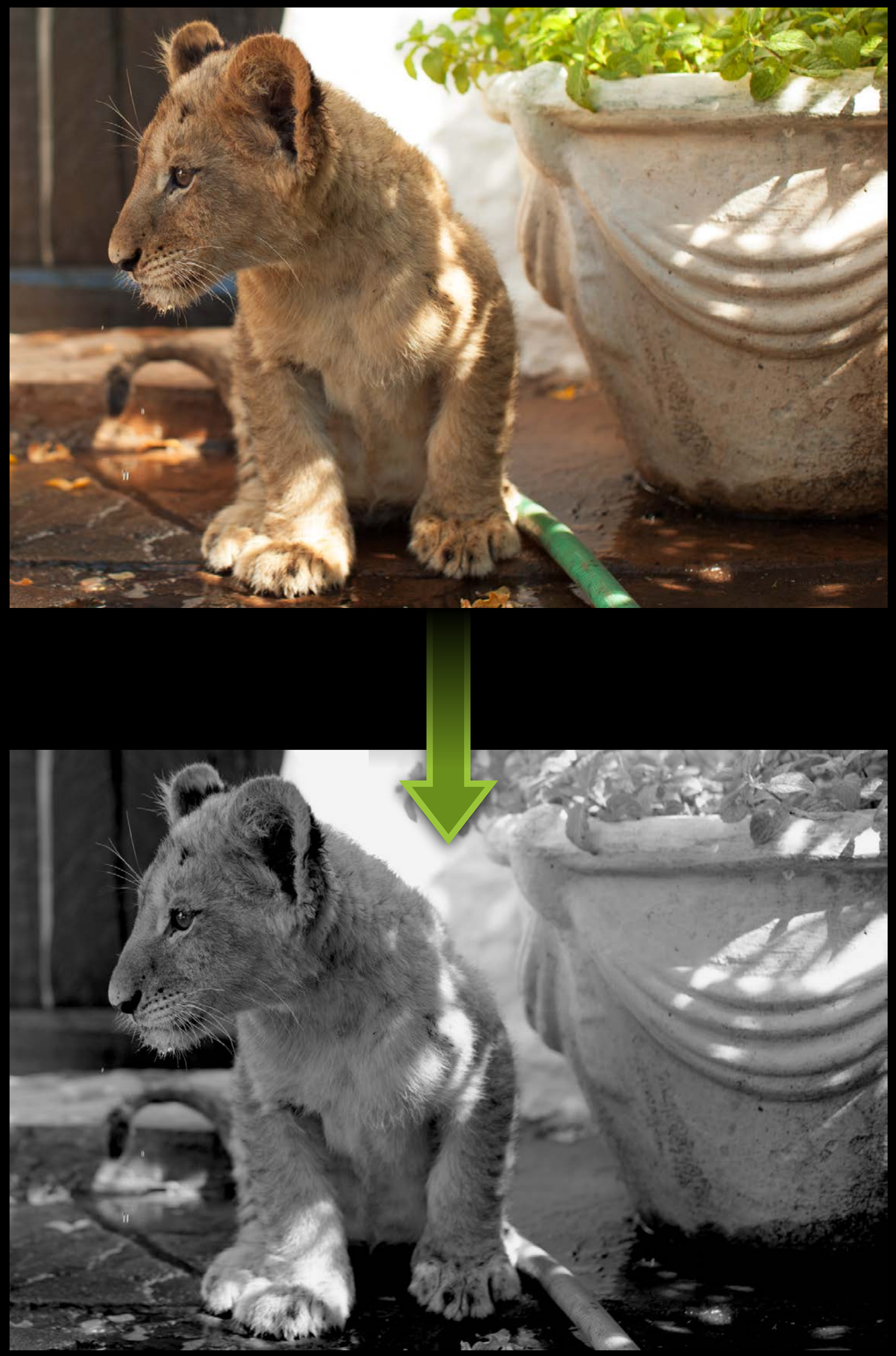
Host

OpenCL Device

## Host

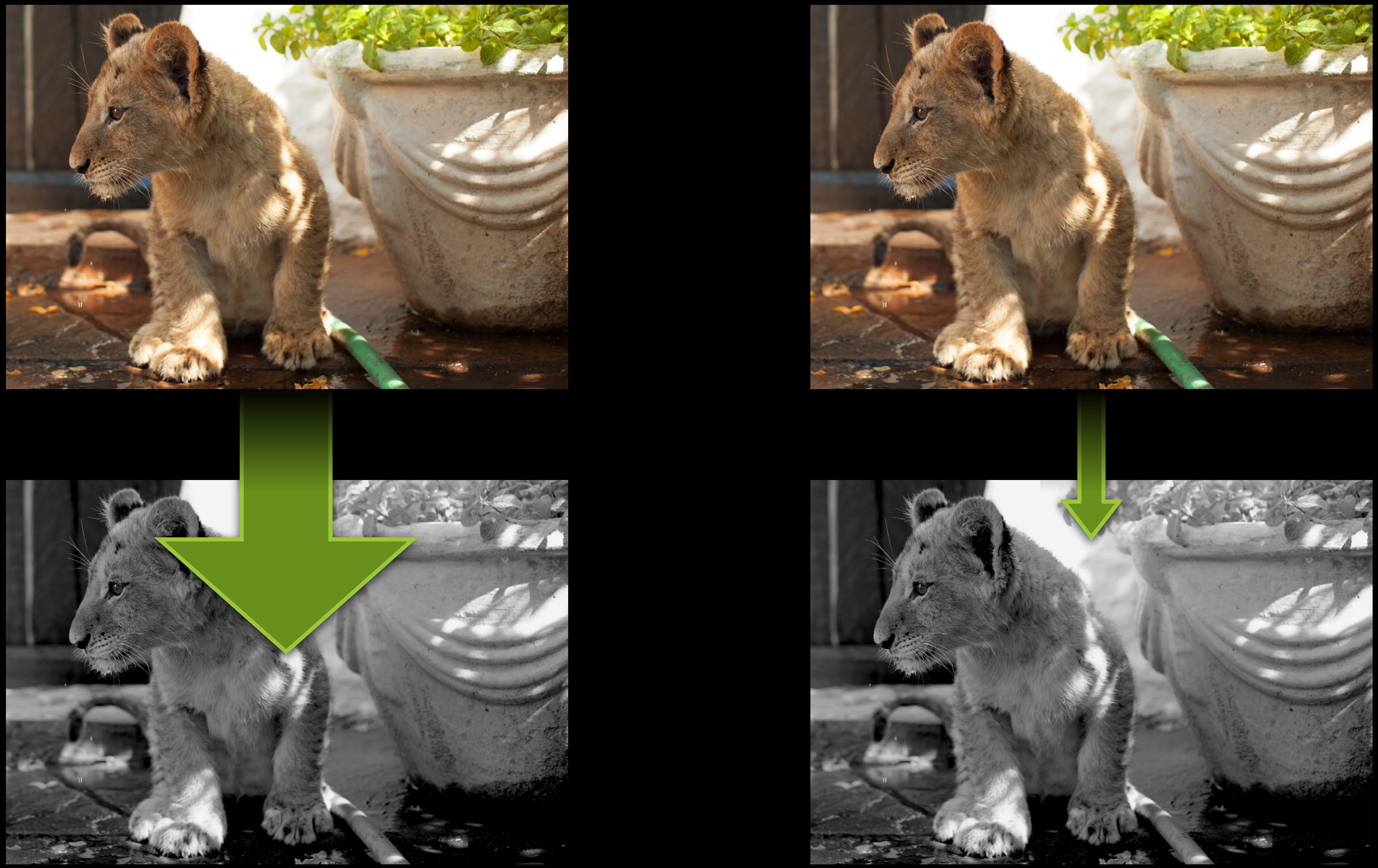


## OpenCL Device

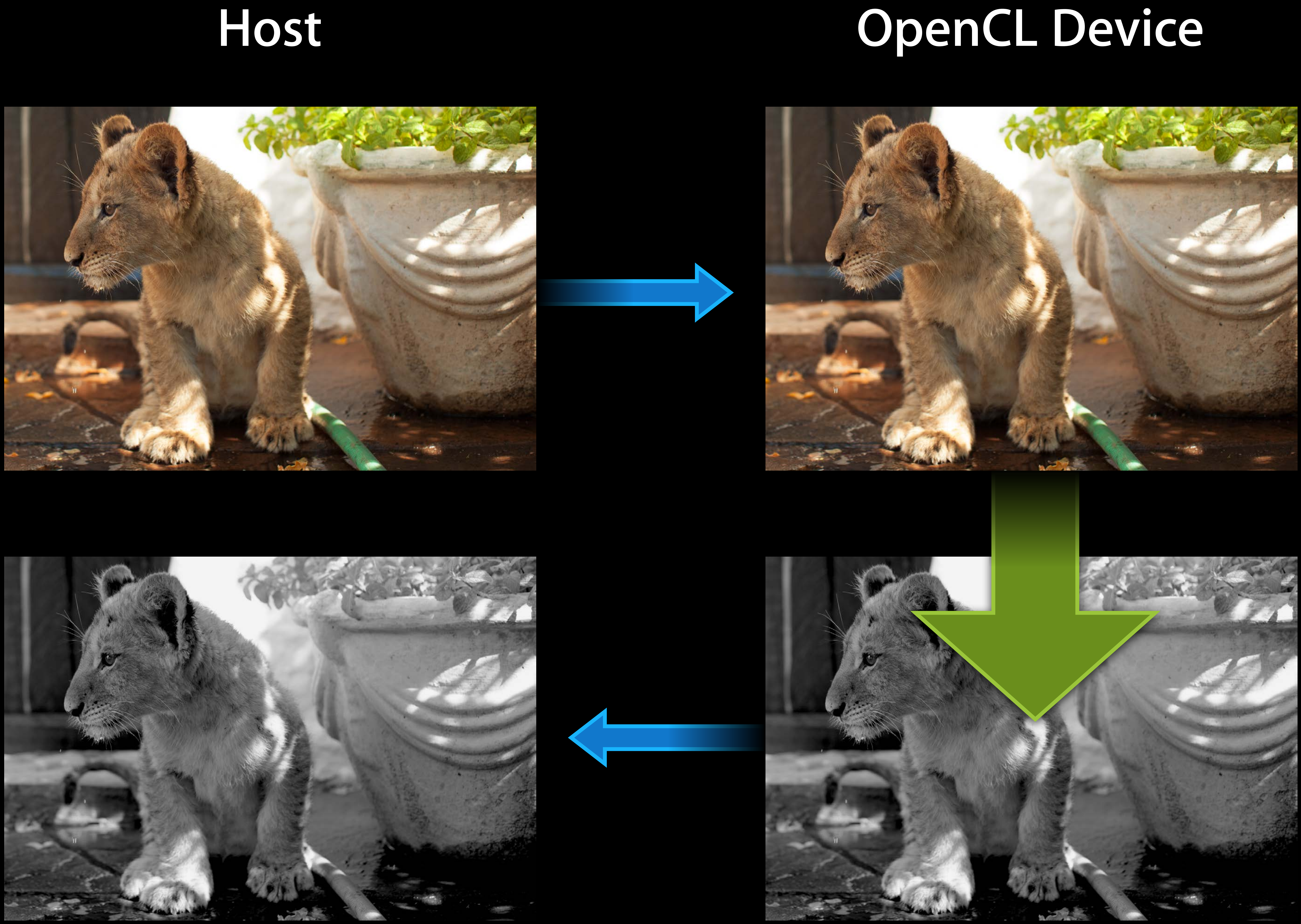


Host

OpenCL Device



OpenCL is going to win

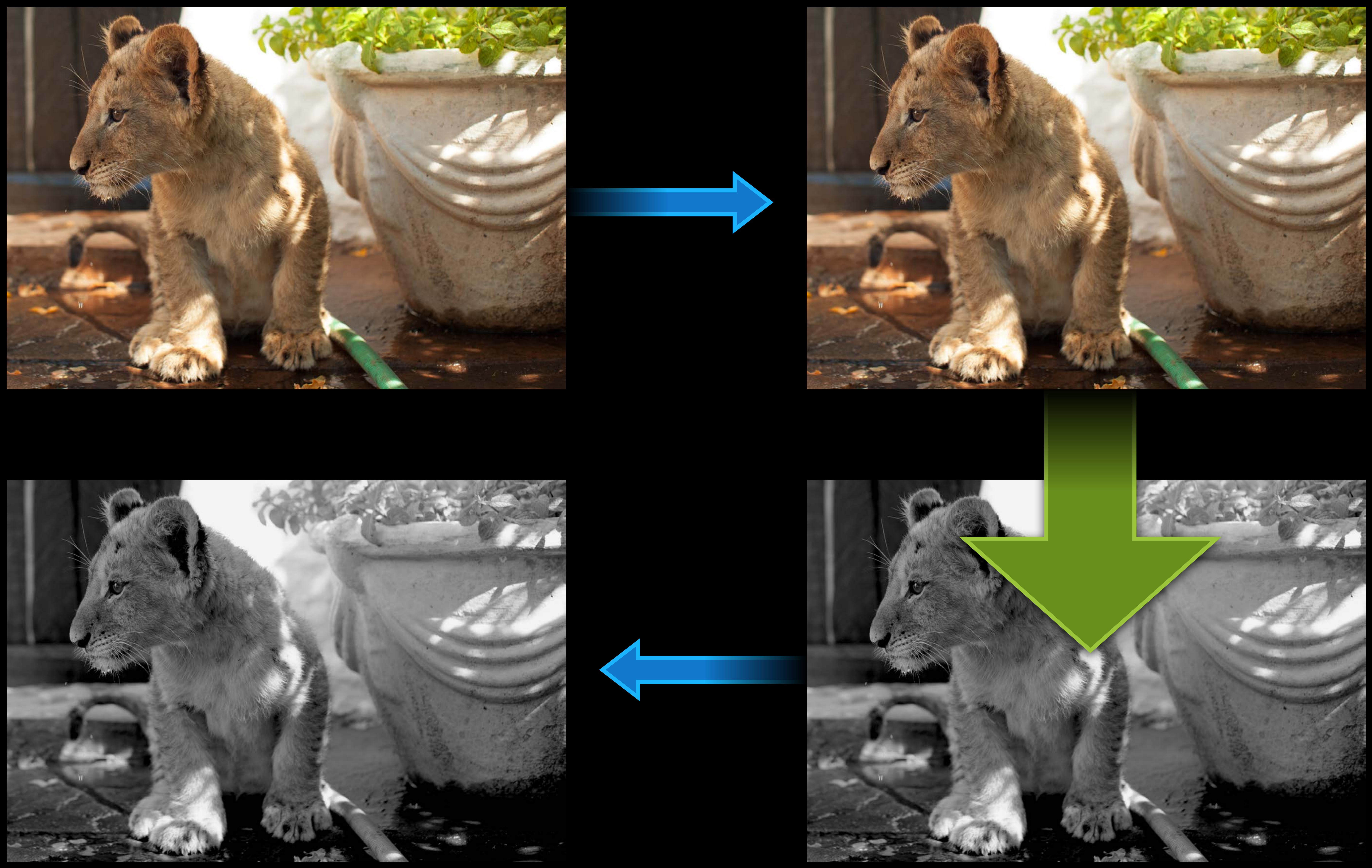


Host

OpenCL Device

Host

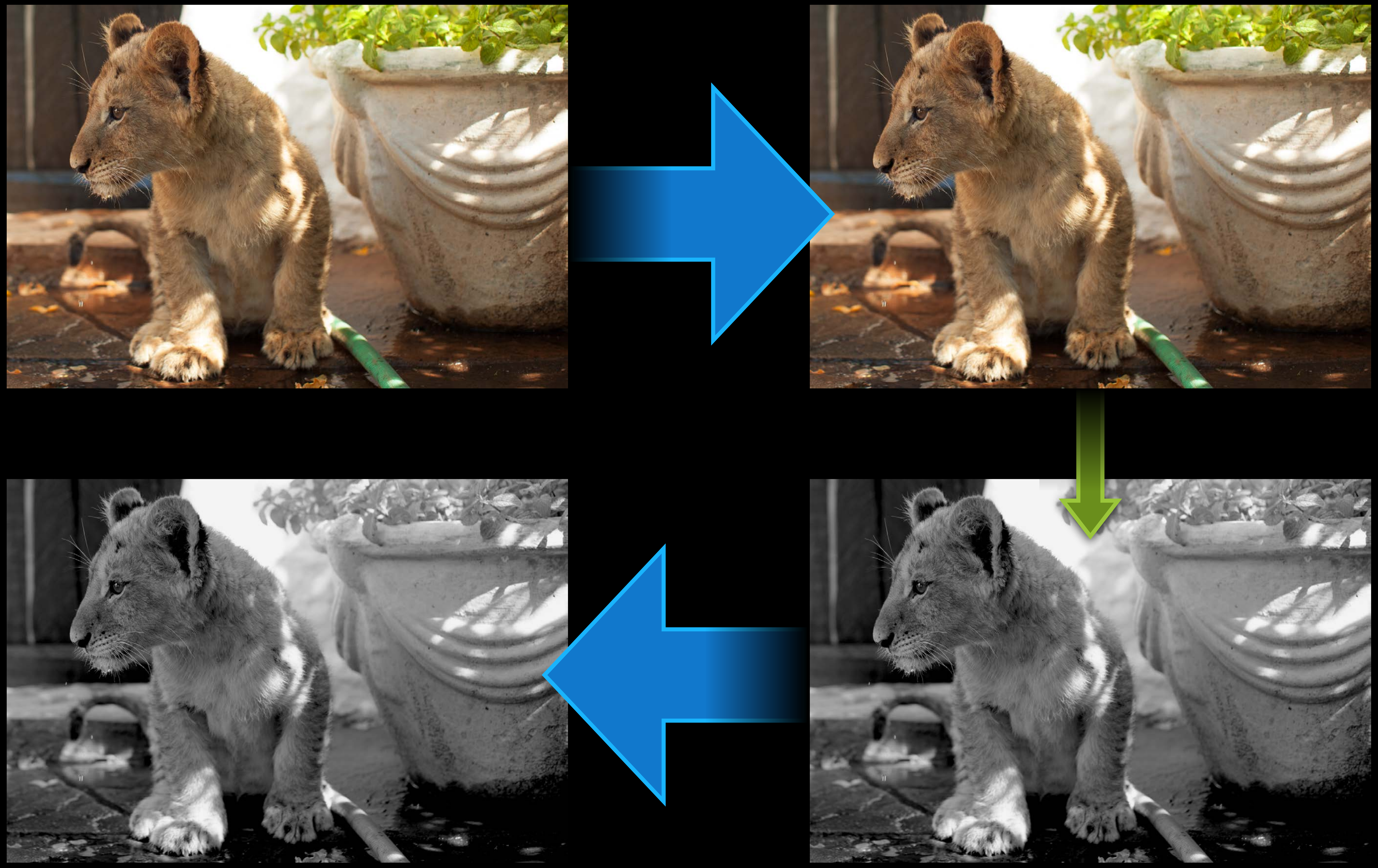
OpenCL Device



Ideal discrete GPU scenario

Host

OpenCL Device



Ideal CPU or integrated GPU scenario

Display Device

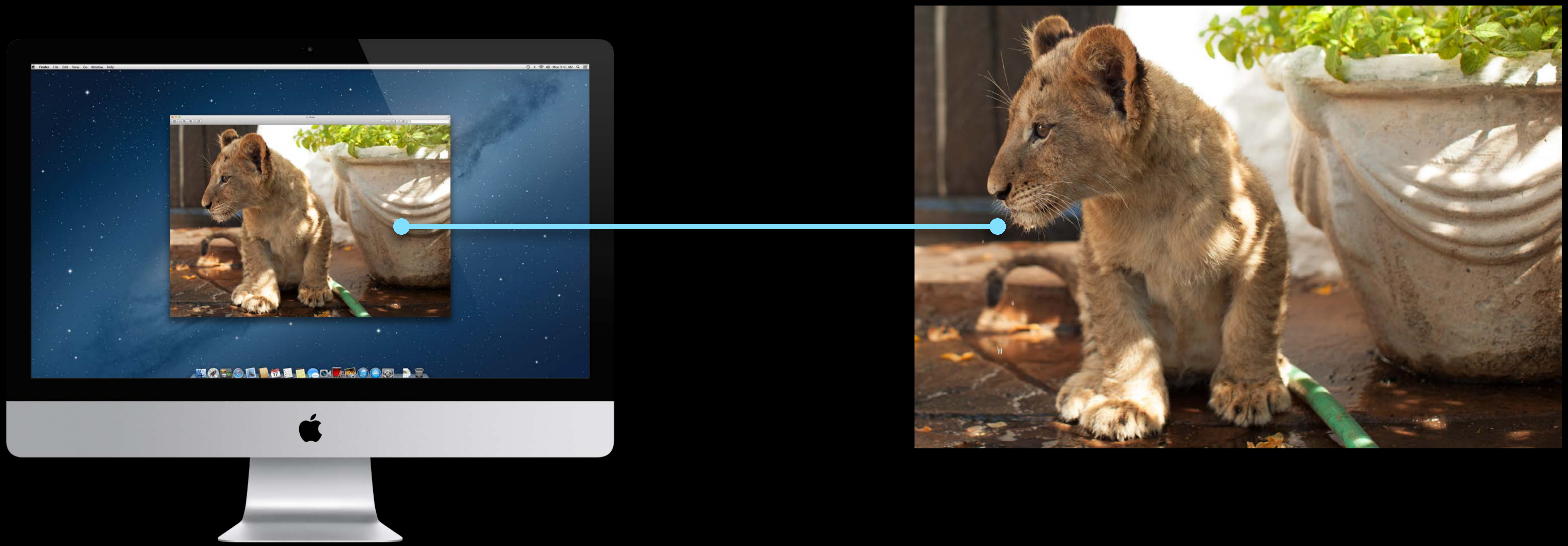
GPU

OpenCL Device

Display Device

GPU

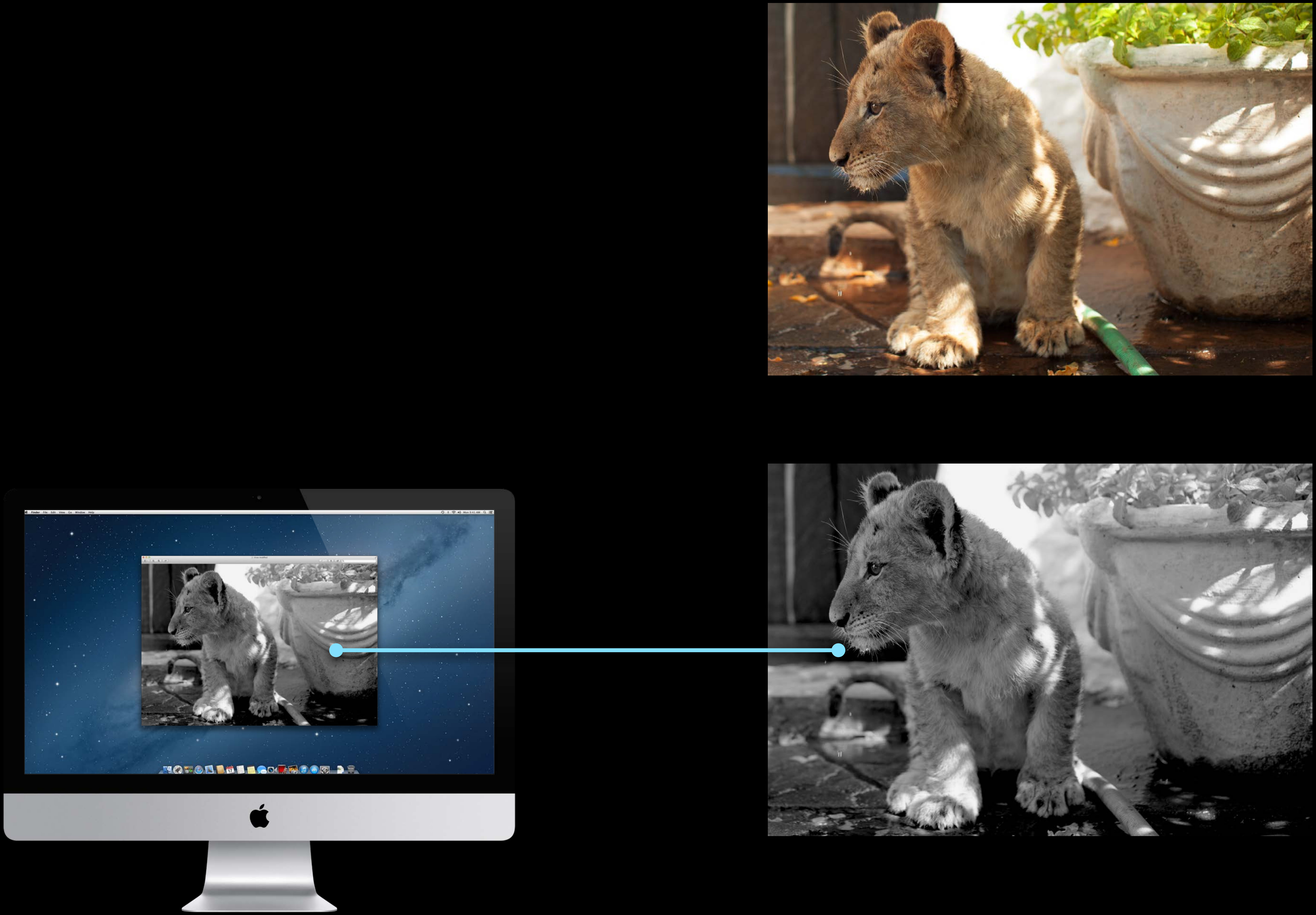
OpenCL Device



Display Device

GPU

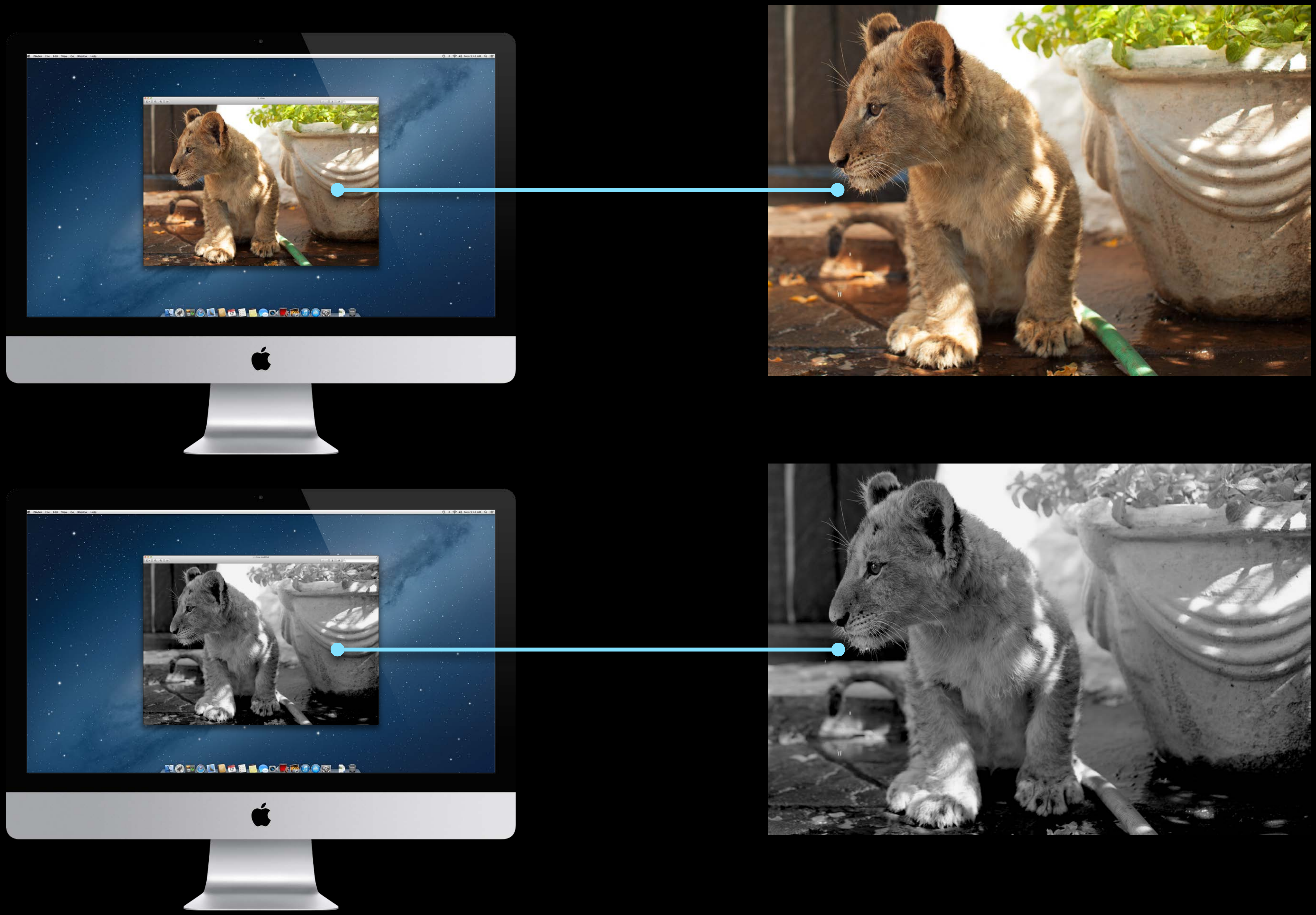
OpenCL Device

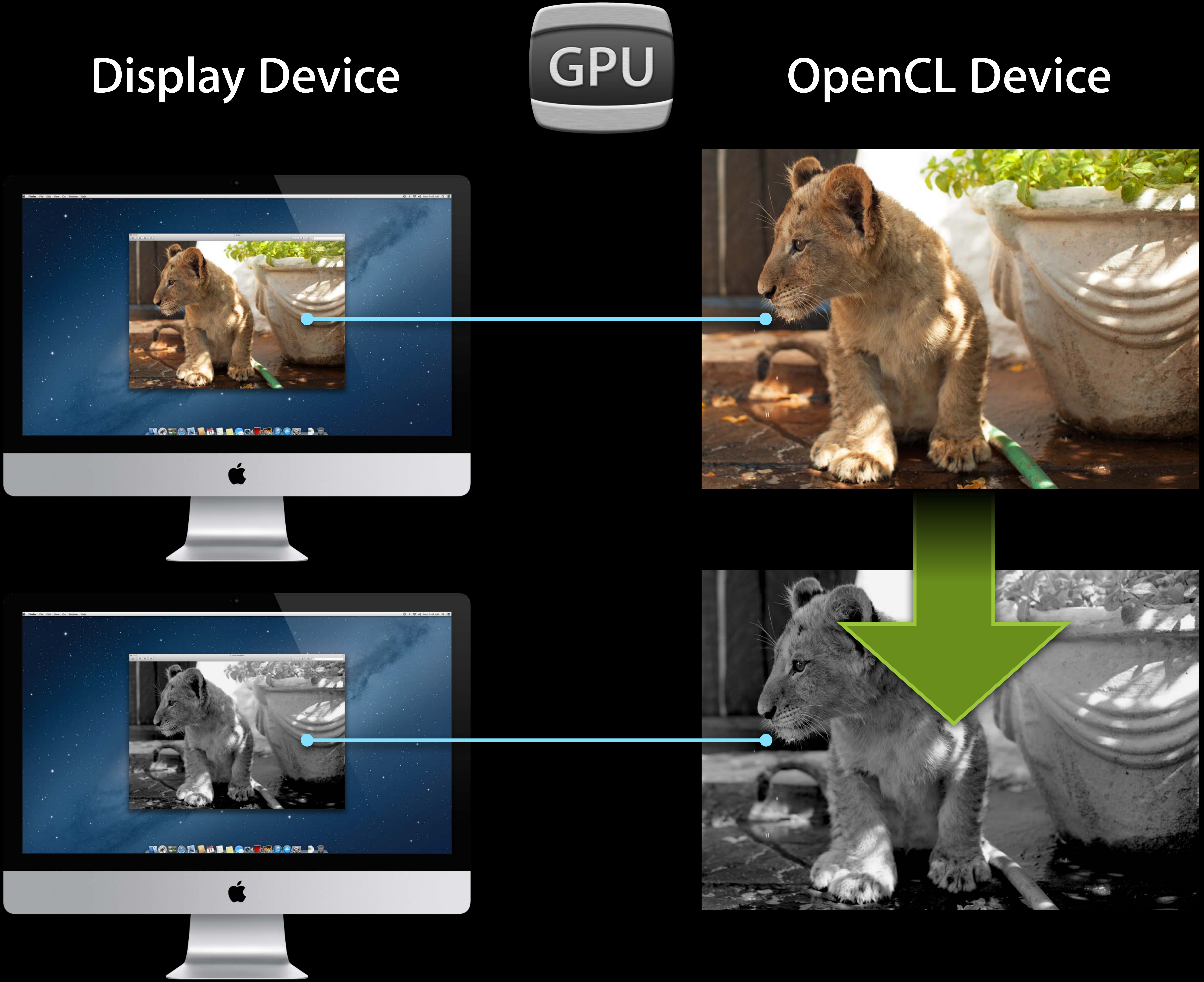


Display Device

GPU

OpenCL Device



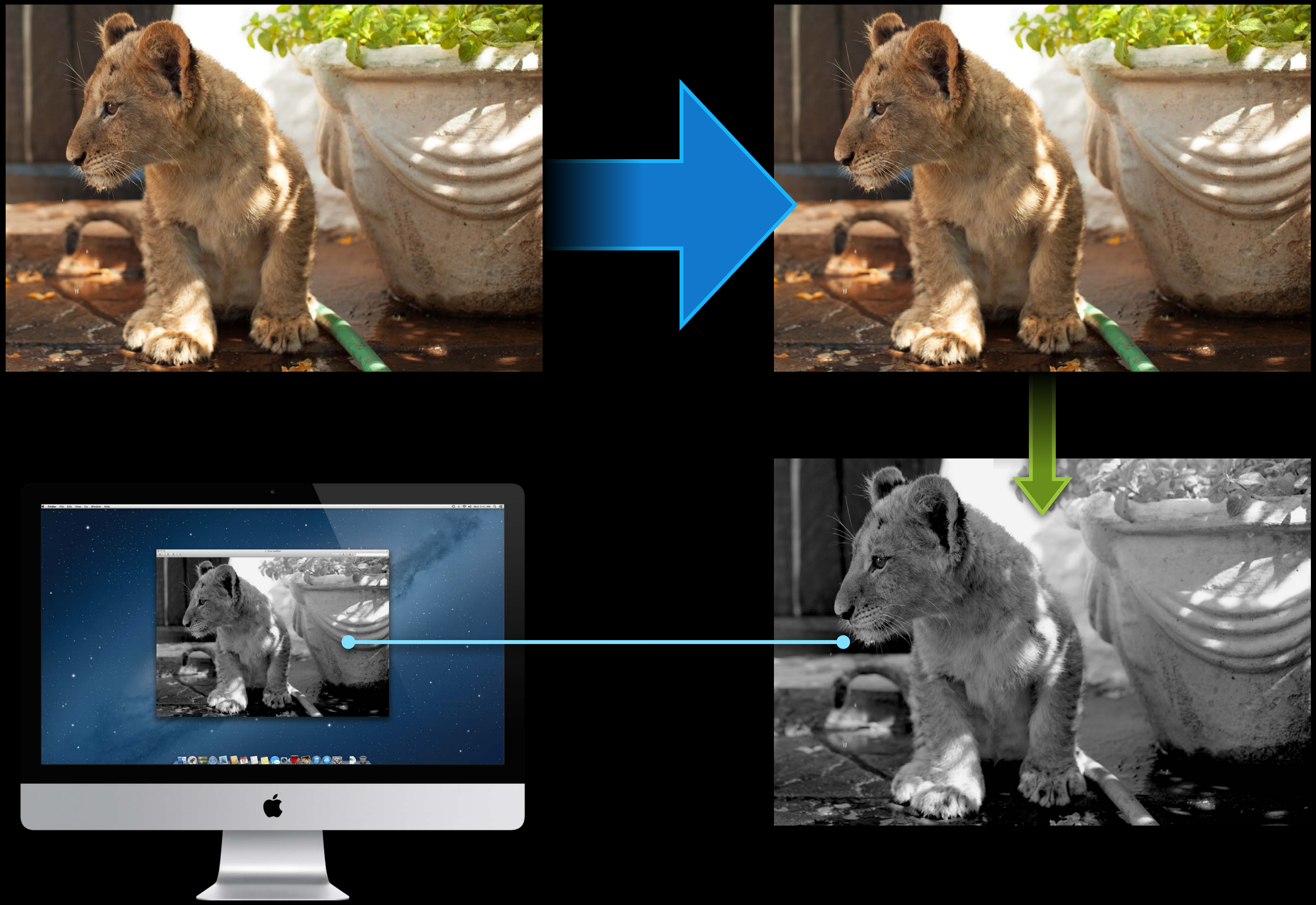

Display Device
GPU
OpenCL Device

## Host

## OpenCL Device

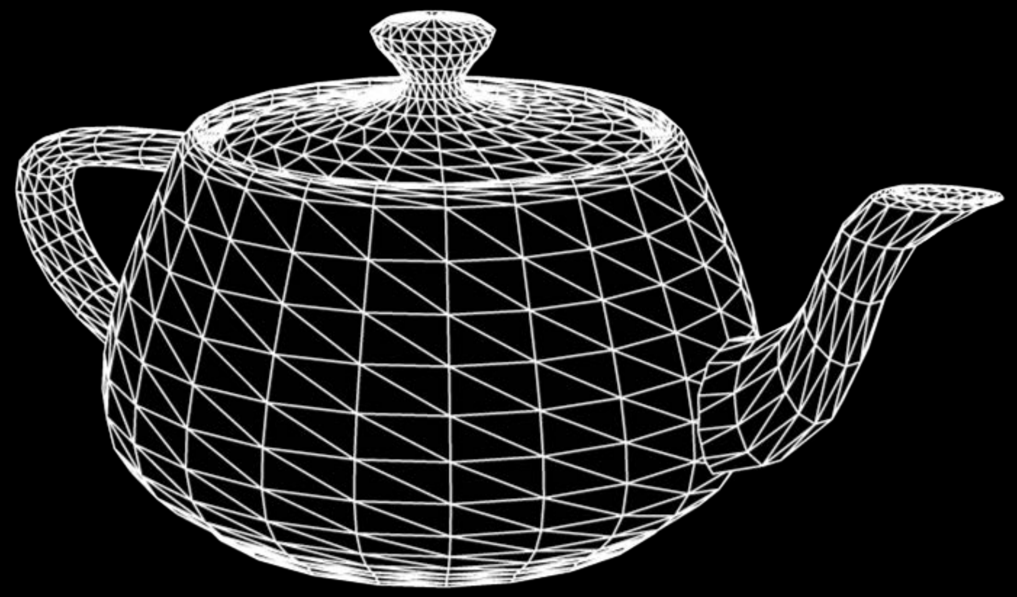
Host

OpenCL Device

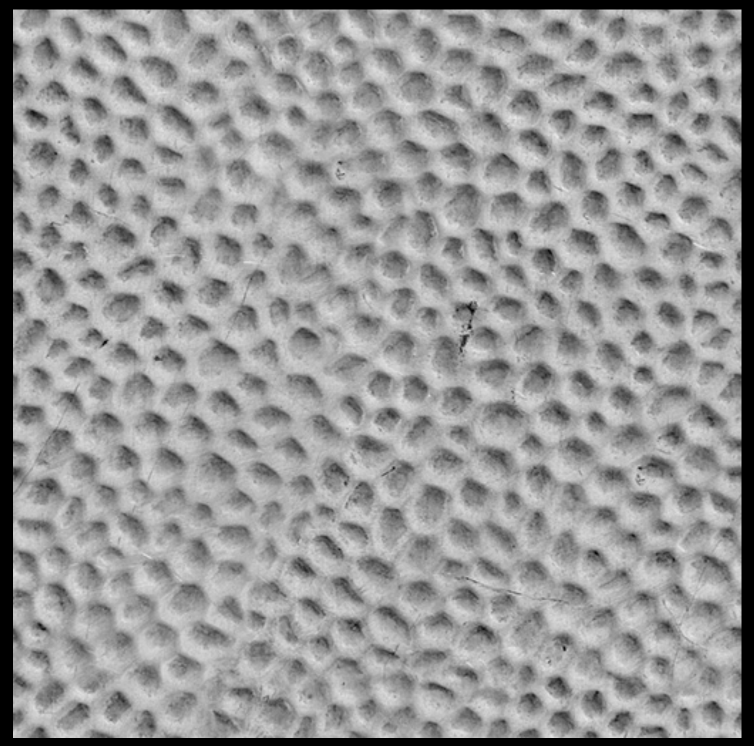


# Data on the Device

## OpenGL objects



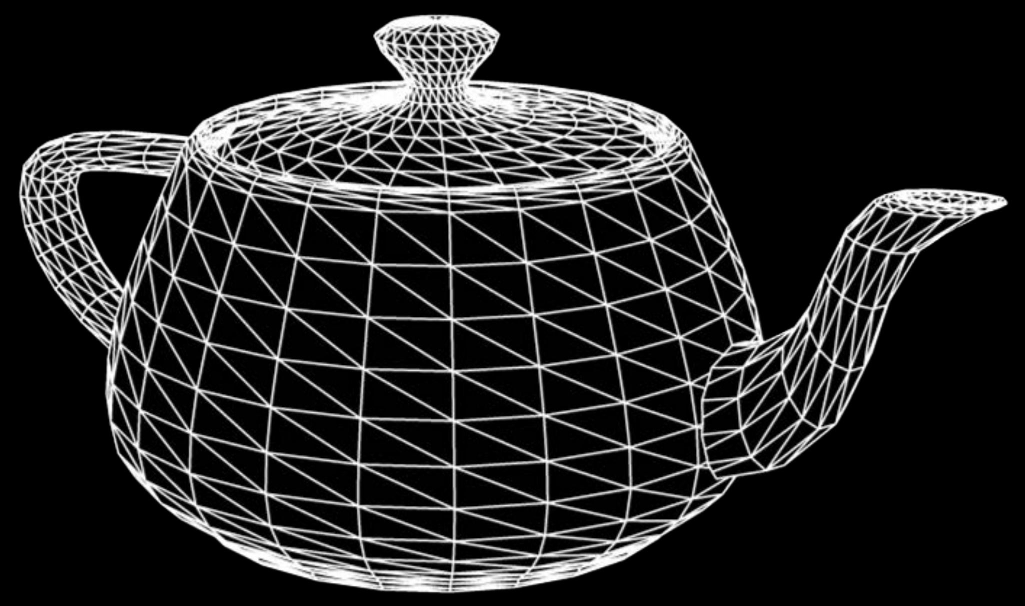
Vertex Buffer Objects



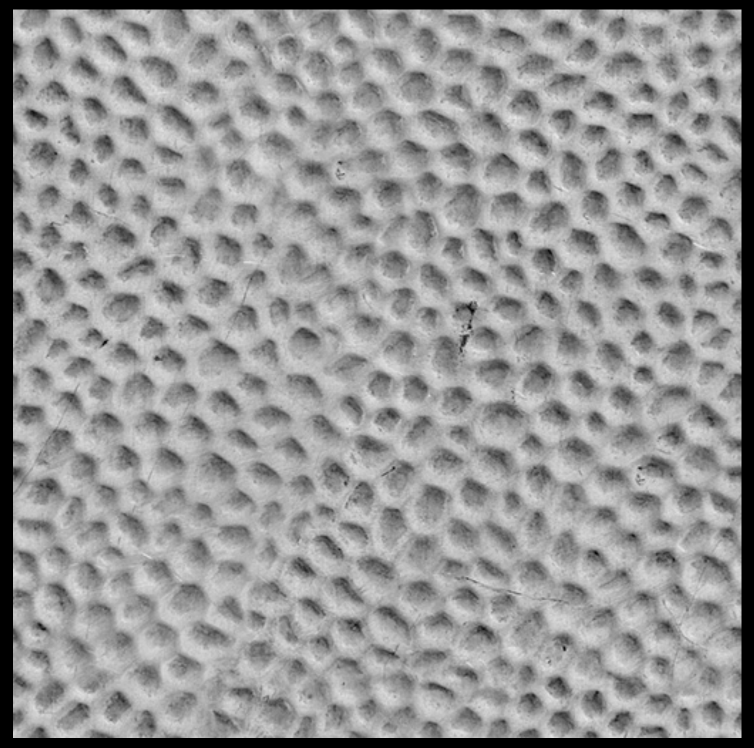
Textures

# Data on the Device

## OpenGL objects



Vertex Buffer Objects

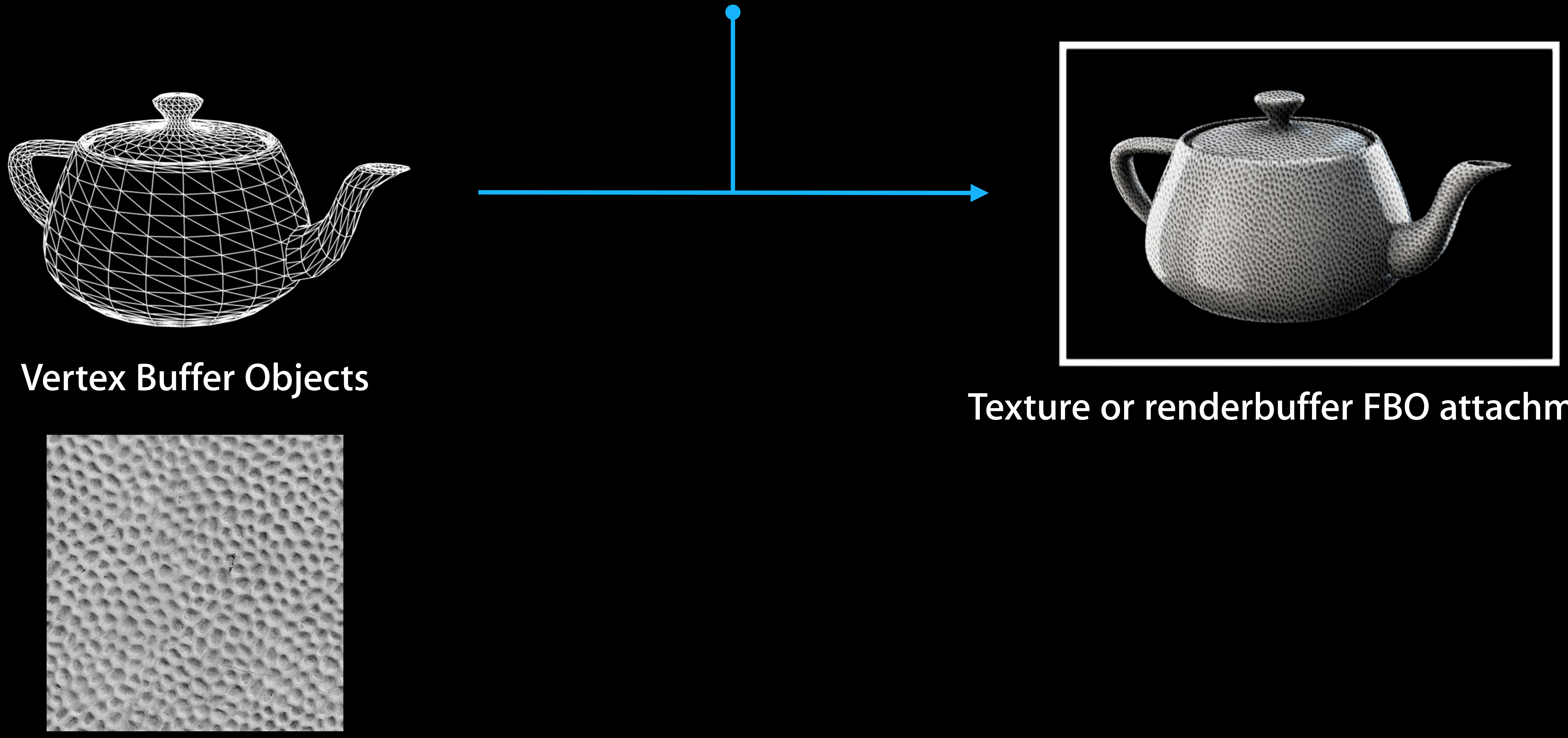


Textures

Texture or renderbuffer FBO attachments

# Data on the Device

## OpenGL objects



Shaders: Vertex, Tessellation, Geometry, Fragment

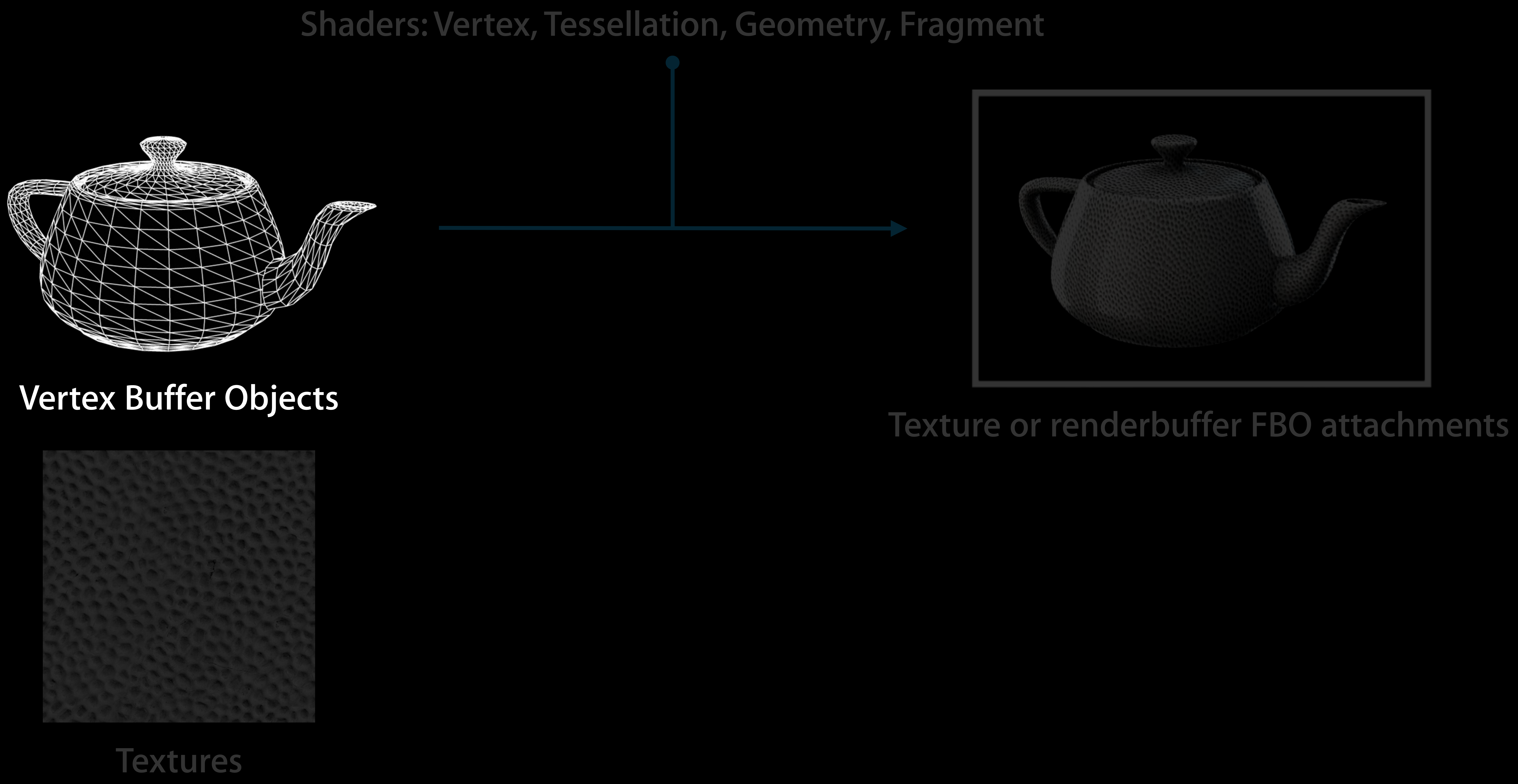
Vertex Buffer Objects

Textures

Texture or renderbuffer FBO attachments

# Data on the Device

## GL buffer objects = OpenCL buffers



Shaders: Vertex, Tessellation, Geometry, Fragment

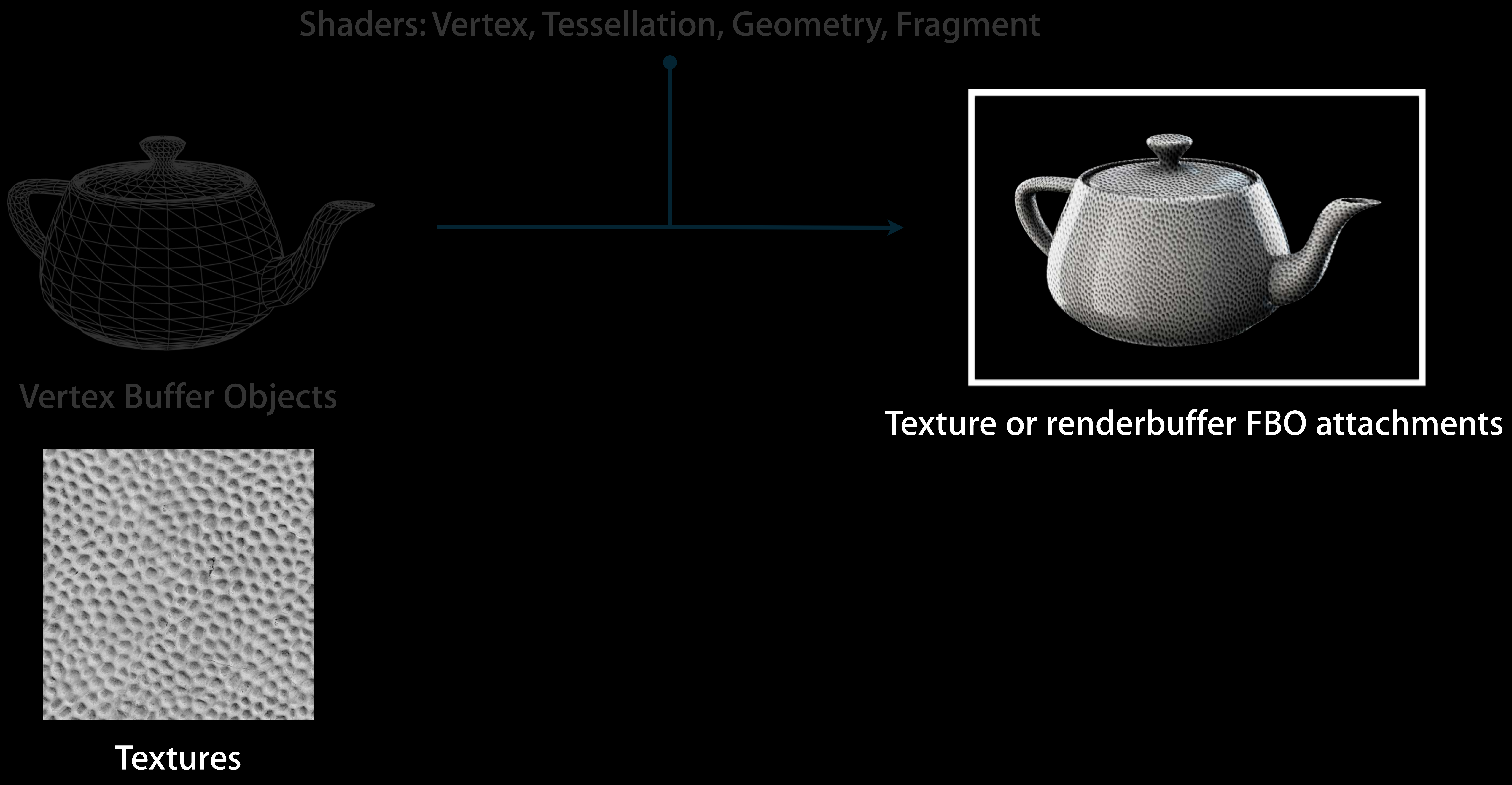
Vertex Buffer Objects

Texture or renderbuffer FBO attachments

Textures

# Data on the Device

## GL textures, Renderbuffers = OpenCL images



Shaders: Vertex, Tessellation, Geometry, Fragment

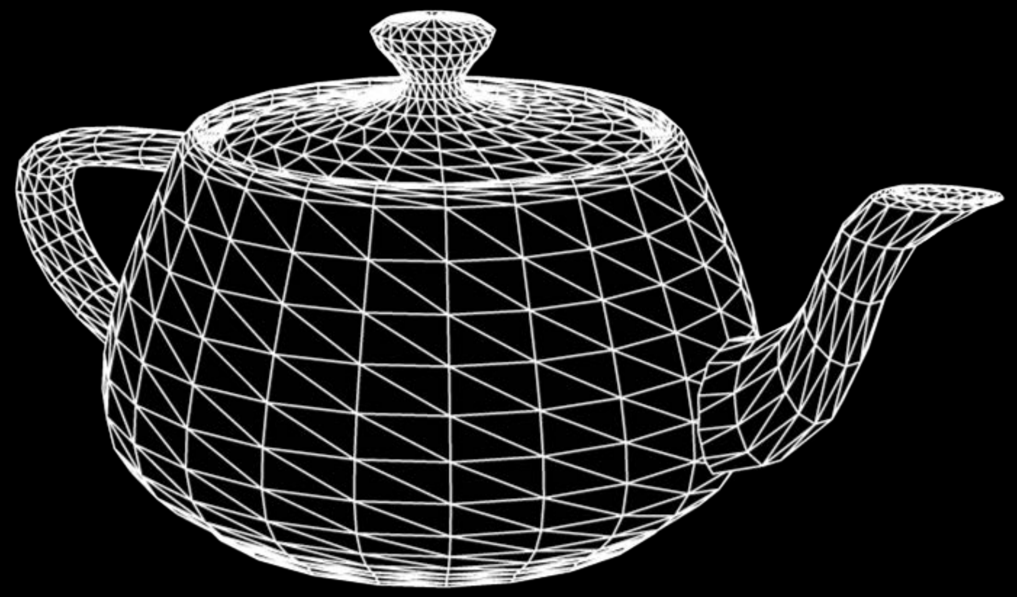
Vertex Buffer Objects

Textures

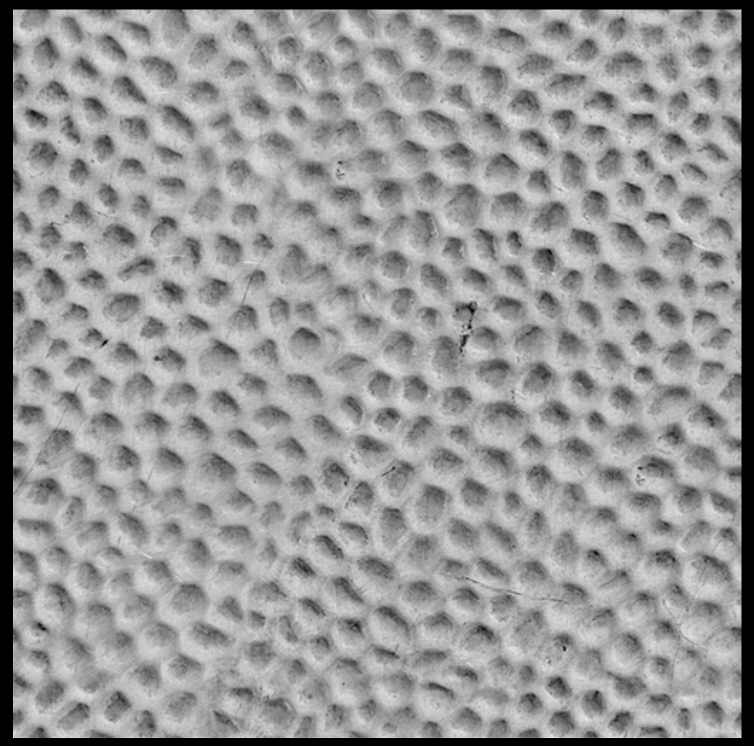
Texture or renderbuffer FBO attachments

# Data on the Device

## OpenCL joins the party

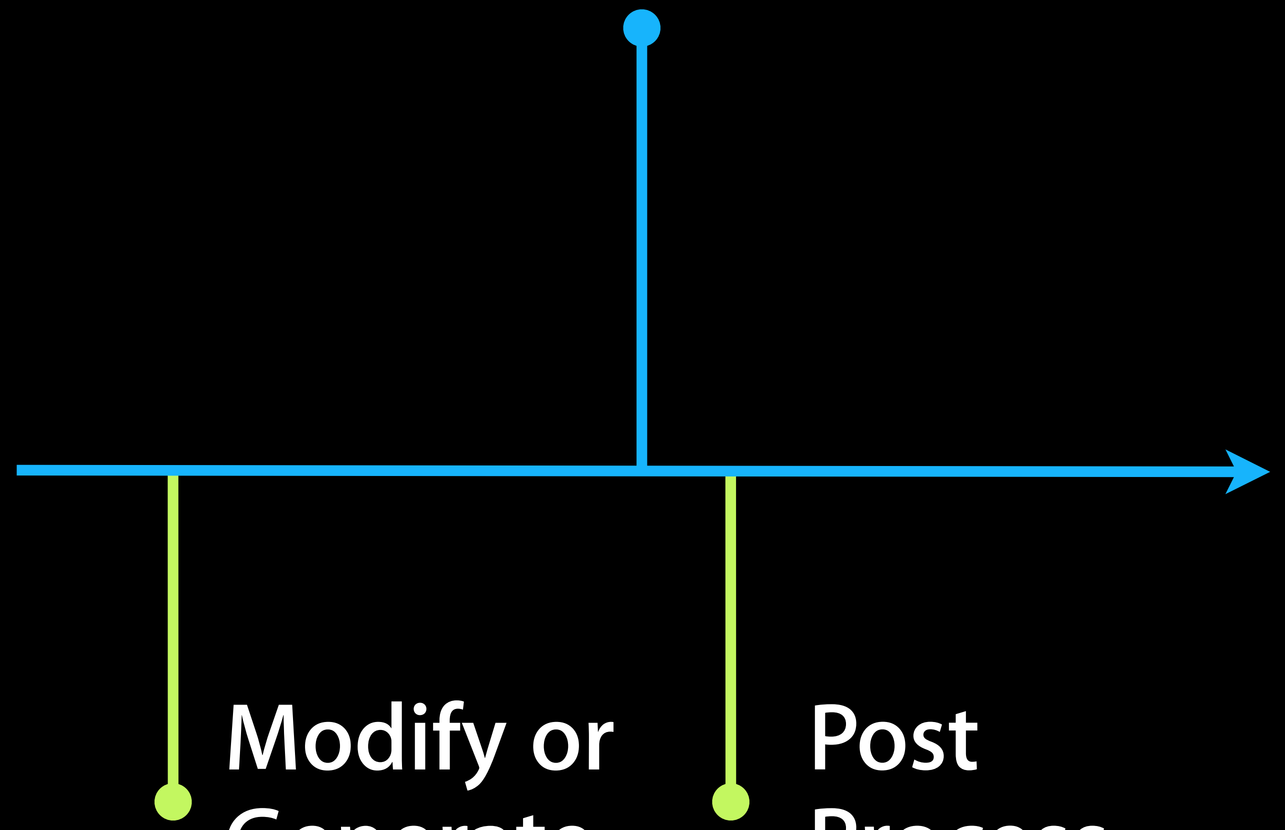


Vertex Buffer Objects



Textures

Shaders: Vertex, Tessellation, Geometry, Fragment



Modify or Generate Vertex Data

Post Process Images

Texture or renderbuffer FBO attachments

# Using Shared Objects in OpenCL

## Mountain Lion

- Flush, acquire, compute, release

```
// Done with previous GL commands
glFlush();

// Update geometry in OpenCL
cl_mem mem_objs[] = { buffer_cl };
clEnqueueAcquireGLObjects(queue, 1, mem_objs, ...);

// compute: clEnqueueNDRangeKernel(...), etc.

// Done with CL commands
clEnqueueReleaseGLObjects(queue, 1, mem_objs, ...);
```

# Using Shared Objects in OpenCL

## OSX Mavericks

- Flush, compute, flush

```
// Done with previous GL commands
glFlushRenderAPPLE();

// Update geometry in OpenCL
cl_mem mem_objs[] = { buffer_cl };
clEnqueueAcquireGLObjects(queue, 1, mem_objs, ...);

// compute: clEnqueueNDRangeKernel(...), etc.

// Done with CL commands
clEnqueueReleaseGLObjects(queue, 1, mem_objs, ...);

 // Done with CL commands
 clFlush(queue);
```

# Using Shared Objects in OpenCL

## OSX Mavericks

- Flush,            compute, flush

```
// Done with previous GL commands
glFlushRenderAPPLE();

// Update geometry in OpenCL
cl_mem mem_objs[] = { buffer_cl };
clEnqueueAcquireGLObjects(queue, 1, mem_objs, ...);

// compute: clEnqueueNDRangeKernel(...), etc.

// Done with CL commands
clEnqueueReleaseGLObjects(queue, 1, mem_objs, ...);

 // Done with CL commands
 clFlush(queue);
```

Avoids the blit!

# IOSurface

- Container for 2D image data
- Goes anywhere, everywhere
- Magically in the right place, at the right time

# IOSurface

- Container for 2D image data
- Goes anywhere, everywhere
- Magically in the right place, at the right time
- Video

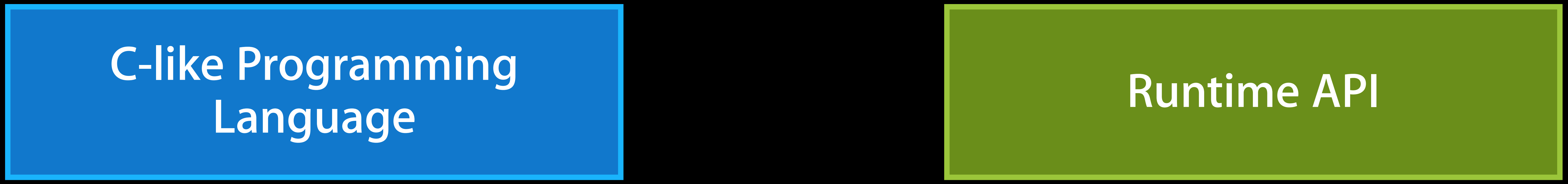
# IOSurface

- Container for 2D image data
- Goes anywhere, everywhere
- Magically in the right place, at the right time
- Video
- WWDC 2011—What's New in OpenCL
- WWDC 2010—Taking Advantage of Multiple GPUs

- □ Am I waiting for something?
- □ Have I avoided something intensive?
- □ Do I have a parallel workload?
- □ Can I earn a parallel workload?
- □ Where is my data now?
- □ Where is my data destined?
- □ How hard am I working?

# The OpenCL Programming Model

# The OpenCL Programming Model



C-like Programming Language

Runtime API

# The OpenCL Programming Model

C-like Programming Language

Runtime API

# The OpenCL Programming Model

C-like Programming Language

Runtime API

Describe your work from the perspective of one piece of data

# The OpenCL Programming Model

C-like Programming Language

Runtime API

Describe your work from the perspective of one piece of data

Guts of your loop

```
// Converting a 1920x1080 image from RGBA to HSV

// rgb is uint32_t*  1 uint32_t per RGBA 8-bit pixel
// hsv is float*     4 floats per pixel
for (y = 0; y < 1080; y++) {
  for (x = 0; x < 1920; x++) {
    int index = y * 1920 + x;               // locate
    int r = rgb[index] & 0x000000FF;        // extract R
    int g = rgb[index] & 0x0000FF00 >> 8;   // extract G
    int b = rgb[index] & 0x00FF0000 >> 16;  // extract B
    int a = rgb[index] & 0xFF000000 >> 24;  // extract A
    float rf = (float)r / 255.0f;           // convert R
    float gf = (float)g / 255.0f;           // convert G
    float bf = (float)b / 255.0f;           // convert B

    float h, s, v;
    rgb2hsv(rf, gf, bg, &h, &s, &v);        // RGB to HSV

    hsv[index*4+0] = h;                     // write
    hsv[index*4+1] = s;                     // results
    hsv[index*4+2] = v;
    hsv[index*4+3] = a / 255.0f;
  }
}
```

```
// Converting a 1920x1080 image from RGBA to HSV

// rgb is uint32_t*  1 uint32_t per RGBA 8-bit pixel
// hsv is float*     4 floats per pixel
for (y = 0; y < 1080; y++) {
  for (x = 0; x < 1920; x++) {
    int index = y * 1920 + x;               // locate
    int r = rgb[index] & 0x000000FF;        // extract R
    int g = rgb[index] & 0x0000FF00 >> 8;   // extract G
    int b = rgb[index] & 0x00FF0000 >> 16;  // extract B
    int a = rgb[index] & 0xFF000000 >> 24;  // extract A
    float rf = (float)r / 255.0f;           // convert R
    float gf = (float)g / 255.0f;           // convert G
    float bf = (float)b / 255.0f;           // convert B

    float h, s, v;
    rgb2hsv(rf, gf, bg, &h, &s, &v);        // RGB to HSV

    hsv[index*4+0] = h;                     // write
    hsv[index*4+1] = s;                     // results
    hsv[index*4+2] = v;
    hsv[index*4+3] = a / 255.0f;
  }
}
```

```
// Converting a 1920x1080 image from RGBA to HSV

// rgb is uint32_t*  1 uint32_t per RGBA 8-bit pixel
// hsv is float*     4 floats per pixel
for (y = 0; y < 1080; y++) {
  for (x = 0; x < 1920; x++) {
    int index = y * 1920 + x;               // locate
    int r = rgb[index] & 0x000000FF;        // extract R
    int g = rgb[index] & 0x0000FF00 >> 8;   // extract G
    int b = rgb[index] & 0x00FF0000 >> 16;  // extract B
    int a = rgb[index] & 0xFF000000 >> 24;  // extract A
    float rf = (float)r / 255.0f;           // convert R
    float gf = (float)g / 255.0f;           // convert G
    float bf = (float)b / 255.0f;           // convert B

    float h, s, v;
    rgb2hsv(rf, gf, bg, &h, &s, &v);        // RGB to HSV

    hsv[index*4+0] = h;                     // write
    hsv[index*4+1] = s;                     // results
    hsv[index*4+2] = v;
    hsv[index*4+3] = a / 255.0f;
  }
}
```

```
// Converting a 1920x1080 image from RGBA to HSV

// rgb is uint32_t*  1 uint32_t per RGBA 8-bit pixel
// hsv is float*     4 floats per pixel
for (y = 0; y < 1080; y++) {
  for (x = 0; x < 1920; x++) {
    int index = y * 1920 + x;               // locate
    int r = rgb[index] & 0x000000FF;        // extract R
    int g = rgb[index] & 0x0000FF00 >> 8;   // extract G
    int b = rgb[index] & 0x00FF0000 >> 16;  // extract B
    int a = rgb[index] & 0xFF000000 >> 24;  // extract A
    float rf = (float)r / 255.0f;           // convert R
    float gf = (float)g / 255.0f;           // convert G
    float bf = (float)b / 255.0f;           // convert B

    float h, s, v;
    rgb2hsv(rf, gf, bg, &h, &s, &v);        // RGB to HSV

    hsv[index*4+0] = h;                     // write
    hsv[index*4+1] = s;                     // results
    hsv[index*4+2] = v;
    hsv[index*4+3] = a / 255.0f;
  }
}
```

```
// Converting a 1920x1080 image from RGBA to HSV

// rgb is uint32_t*  1 uint32_t per RGBA 8-bit pixel
// hsv is float*     4 floats per pixel
for (y = 0; y < 1080; y++) {
  for (x = 0; x < 1920; x++) {
    int index = y * 1920 + x;               // locate
    int r = rgb[index] & 0x000000FF;        // extract R
    int g = rgb[index] & 0x0000FF00 >> 8;   // extract G
    int b = rgb[index] & 0x00FF0000 >> 16;  // extract B
    int a = rgb[index] & 0xFF000000 >> 24;  // extract A
    float rf = (float)r / 255.0f;           // convert R
    float gf = (float)g / 255.0f;           // convert G
    float bf = (float)b / 255.0f;           // convert B

    float h, s, v;
    rgb2hsv(rf, gf, bg, &h, &s, &v);        // RGB to HSV

    hsv[index*4+0] = h;                     // write
    hsv[index*4+1] = s;                     // results
    hsv[index*4+2] = v;
    hsv[index*4+3] = a / 255.0f;
  }
}
```

```
// Converting a 1920x1080 image from RGBA to HSV

// rgb is uint32_t*  1 uint32_t per RGBA 8-bit pixel
// hsv is float*     4 floats per pixel
for (y = 0; y < 1080; y++) {
  for (x = 0; x < 1920; x++) {
    int index = y * 1920 + x;               // locate
    int r = rgb[index] & 0x000000FF;        // extract R
    int g = rgb[index] & 0x0000FF00 >> 8;   // extract G
    int b = rgb[index] & 0x00FF0000 >> 16;  // extract B
    int a = rgb[index] & 0xFF000000 >> 24;  // extract A
    float rf = (float)r / 255.0f;           // convert R
    float gf = (float)g / 255.0f;           // convert G
    float bf = (float)b / 255.0f;           // convert B

    float h, s, v;
    rgb2hsv(rf, gf, bg, &h, &s, &v);        // RGB to HSV

    hsv[index*4+0] = h;                     // write
    hsv[index*4+1] = s;                     // results
    hsv[index*4+2] = v;
    hsv[index*4+3] = a / 255.0f;
  }
}
```

```
// Converting a 1920x1080 image from RGBA to HSV

// rgb is uint32_t*  1 uint32_t per RGBA 8-bit pixel
// hsv is float*     4 floats per pixel
for (y = 0; y < 1080; y++) {
  for (x = 0; x < 1920; x++) {
    int index = y * 1920 + x;               // locate
    int r = rgb[index] & 0x000000FF;        // extract R
    int g = rgb[index] & 0x0000FF00 >> 8;   // extract G
    int b = rgb[index] & 0x00FF0000 >> 16;  // extract B
    int a = rgb[index] & 0xFF000000 >> 24;  // extract A
    float rf = (float)r / 255.0f;           // convert R
    float gf = (float)g / 255.0f;           // convert G
    float bf = (float)b / 255.0f;           // convert B

    float h, s, v;
    rgb2hsv(rf, gf, bg, &h, &s, &v);        // RGB to HSV

    hsv[index*4+0] = h;                     // write
    hsv[index*4+1] = s;                     // results
    hsv[index*4+2] = v;
    hsv[index*4+3] = a / 255.0f;
  }
}
```

```c
void rgb2hsv(float r, float g, float b, float *h, float *s, float *v) {
  float cmax = r > g ? r : g; cmax = cmax > b ? cmax : b;
  float cmin = r < g ? r : g; cmin = cmin < b ? cmin : b;
  float delta = max - min;

  *v = cmax;
  if ( cmax == 0.0f || delta == 0.0f ) {
    *h = *s = 0.0f;
    return;
  }

  if ( r == cmax )
    *h = (g - b) / delta;
  else if ( g == cmax )
    *h = 2 + (b - r) / delta;
  else
    *h = 4 + (r - g) / delta;

  *h *= 60;
  if ( *h < 0.0f ) *h += 360.0f;

  *s = delta / cmax;
}
```

```
kernel void convert_rgb2hsv(read_only image2d_t in, write_only image2d_t out)
{
  size_t x = get_global_id(0);                  // locate my pixel
  size_t y = get_global_id(1);

  float4 pixel = read_imagef(in, (int2)(x,y));  // read
  float4 result = rgb2hsv(pixel);               // RGB to HSV

  write_imagef(out, (int2)(x,y), result);       // write
}
```

```
kernel void convert_rgb2hsv(read_only image2d_t in, write_only image2d_t out)
{
  size_t x = get_global_id(0);                  // locate my pixel
  size_t y = get_global_id(1);

  float4 pixel = read_imagef(in, (int2)(x,y));  // read
  float4 result = rgb2hsv(pixel);               // RGB to HSV

  write_imagef(out, (int2)(x,y), result);       // write
}
```

```
kernel void convert_rgb2hsv(read_only image2d_t in, write_only image2d_t out)
{
  size_t x = get_global_id(0);                       // locate my pixel
  size_t y = get_global_id(1);

  float4 pixel = read_imagef(in, (int2)(x,y));  // read
  float4 result = rgb2hsv(pixel);                    // RGB to HSV

  write_imagef(out, (int2)(x,y), result);        // write
}
```

```
kernel void convert_rgb2hsv(read_only image2d_t in, write_only image2d_t out)
{
  size_t x = get_global_id(0);                    // locate my pixel
  size_t y = get_global_id(1);

  float4 pixel = read_imagef(in, (int2)(x,y));  // read
  float4 result = rgb2hsv(pixel);                 // RGB to HSV

  write_imagef(out, (int2)(x,y), result);         // write
}
```

```
kernel void convert_rgb2hsv(read_only image2d_t in, write_only image2d_t out)
{
  size_t x = get_global_id(0);                  // locate my pixel
  size_t y = get_global_id(1);

  float4 pixel = read_imagef(in, (int2)(x,y));  // read
  float4 result = rgb2hsv(pixel);               // RGB to HSV

  write_imagef(out, (int2)(x,y), result);       // write
}
```

```
float4 rgb2hsv(float4 pixel) {
  float cmax = max(pixel.x, pixel.y); cmax = max(cmax, pixel.z);
  float cmin = min(pixel.x, pixel.y); cmin = min(cmin, pixel.z);
  float delta = cmax - cmin;
  float4 result = (float4)(0.0f, 0.0f, 0.0f, pixel.w);

  result.z = cmax;
  if ( cmax == 0.0f || delta == 0.0f ) {
    return result;
  }

  if ( pixel.x == cmax )
    result.x = ( pixel.y - pixel.z ) / delta;
  else if ( pixel.y == cmax )
    result.x = 2 + ( pixel.z - pixel.x ) / delta;
  else
    result.x = 4 + ( pixel.x - pixel.y ) / delta;

  result.x *= 60.0f;
  if ( result.x < 0.0f ) result.x += 360.0f;

  result.y = delta / cmax;
  return result;
}
```

```
float4 rgb2hsv(float4 pixel) {
  float cmax = max(pixel.x, pixel.y); cmax = max(cmax, pixel.z);
  float cmin = min(pixel.x, pixel.y); cmin = min(cmin, pixel.z);
  float delta = cmax - cmin;
  float4 result = (float4)(0.0f, 0.0f, 0.0f, pixel.w);

  result.z = cmax;
  if ( cmax == 0.0f || delta == 0.0f ) {
    return result;
  }

  if ( pixel.x == cmax )
    result.x = ( pixel.y - pixel.z ) / delta;
  else if ( pixel.y == cmax )
    result.x = 2 + ( pixel.z - pixel.x ) / delta;
  else
    result.x = 4 + ( pixel.x - pixel.y ) / delta;

  result.x *= 60.0f;
  if ( result.x < 0.0f ) result.x += 360.0f;

  result.y = delta / cmax;
  return result;
}
```

```
float4 rgb2hsv(float4 pixel) {
  float cmax = max(pixel.x, pixel.y); cmax = max(cmax, pixel.z);
  float cmin = min(pixel.x, pixel.y); cmin = min(cmin, pixel.z);
  float delta = cmax - cmin;
  float4 result = (float4)(0.0f, 0.0f, 0.0f, pixel.w);

  result.z = cmax;
  if ( cmax == 0.0f || delta == 0.0f ) {
    return result;
  }

  if ( pixel.x == cmax )
    result.x = ( pixel.y - pixel.z ) / delta;
  else if ( pixel.y == cmax )
    result.x = 2 + ( pixel.z - pixel.x ) / delta;
  else
    result.x = 4 + ( pixel.x - pixel.y ) / delta;

  result.x *= 60.0f;
  if ( result.x < 0.0f ) result.x += 360.0f;

  result.y = delta / cmax;
  return result;
}
```

```
void rgb2hsv(float r, float g, float b, float *h, float *s, float *v) {
  float cmax = r > g ? r : g; cmax = cmax > b ? cmax : b;
  float cmin = r < g ? r : g; cmin = cmin < b ? cmin : b;
  float delta = max - min;


  *v = cmax;
  if ( cmax == 0.0f || delta == 0.0f ) {
    *h = *s = 0.0f; return;
  }

  if ( r == cmax )
    *h = (g - b) / delta;
  else if ( g == cmax )
    *h = 2 + (b - r) / delta;
  else
    *h = 4 + (r - g) / delta;

  *h *= 60.0f;
  if ( *h < 0.0f ) *h += 360.0f;

  *s = delta / cmax;

}
```

# The OpenCL Programming Model

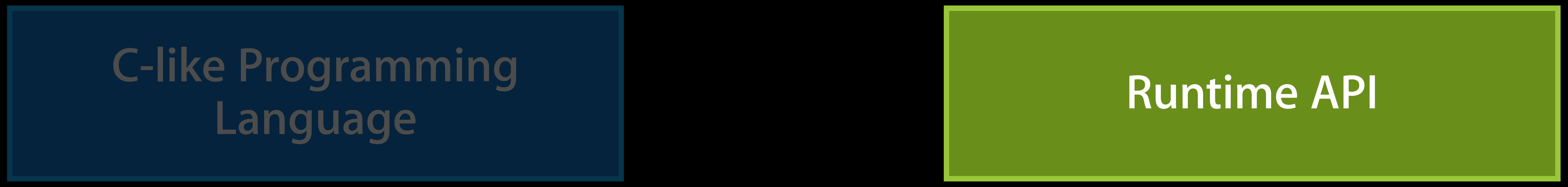
C-like Programming Language

Runtime API

Describe your work from the perspective of one piece of data

Guts of your loop

# The OpenCL Programming Model



C-like Programming Language

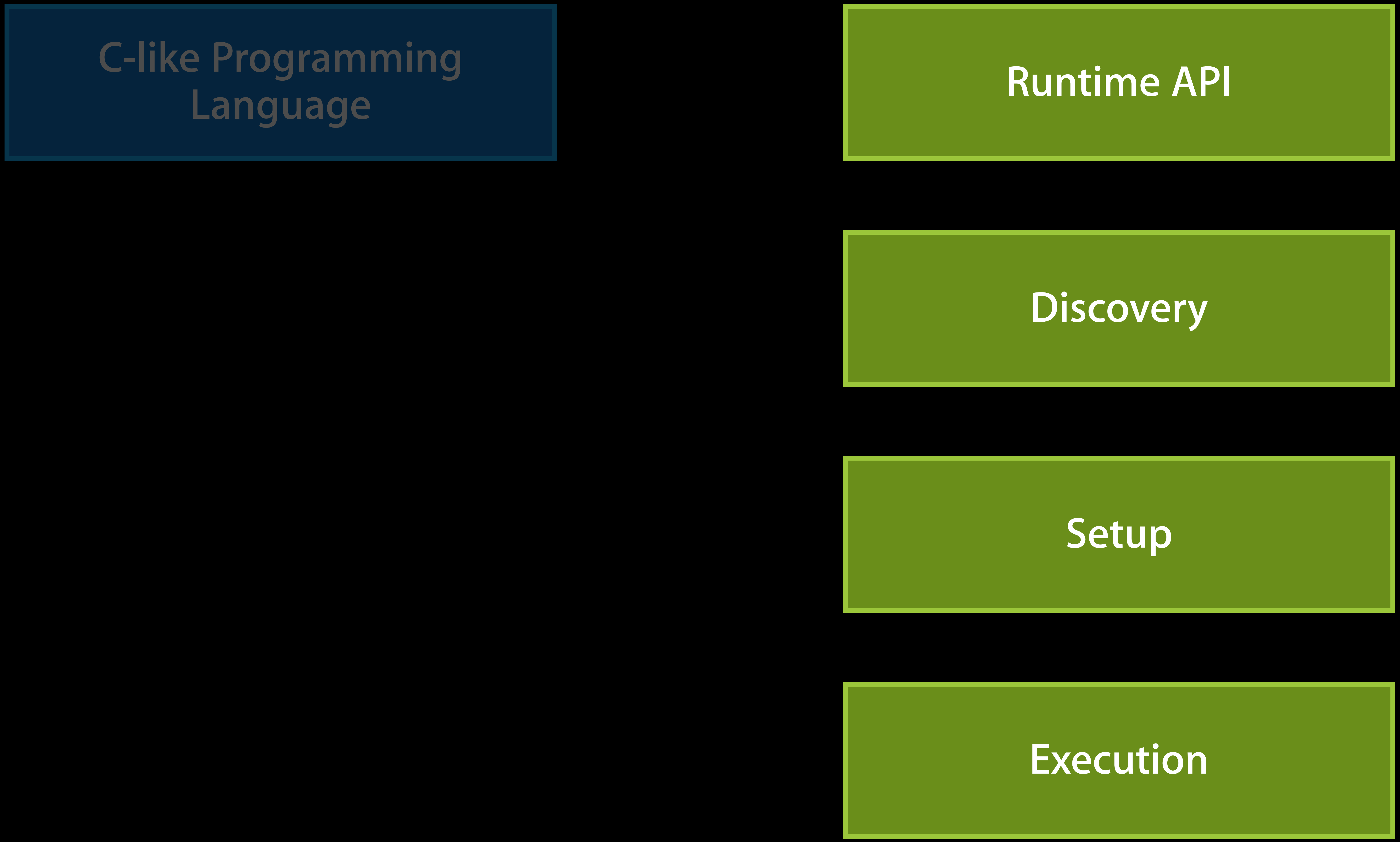
Runtime API

# The OpenCL Programming Model

C-like Programming Language

Runtime API

# The OpenCL Programming Model



C-like Programming Language

Runtime API

Discovery

Setup

Execution

# The OpenCL Programming Model

C-like Programming Language

What devices are in my Mac?

Given a device, what’s the best way to break up my work?

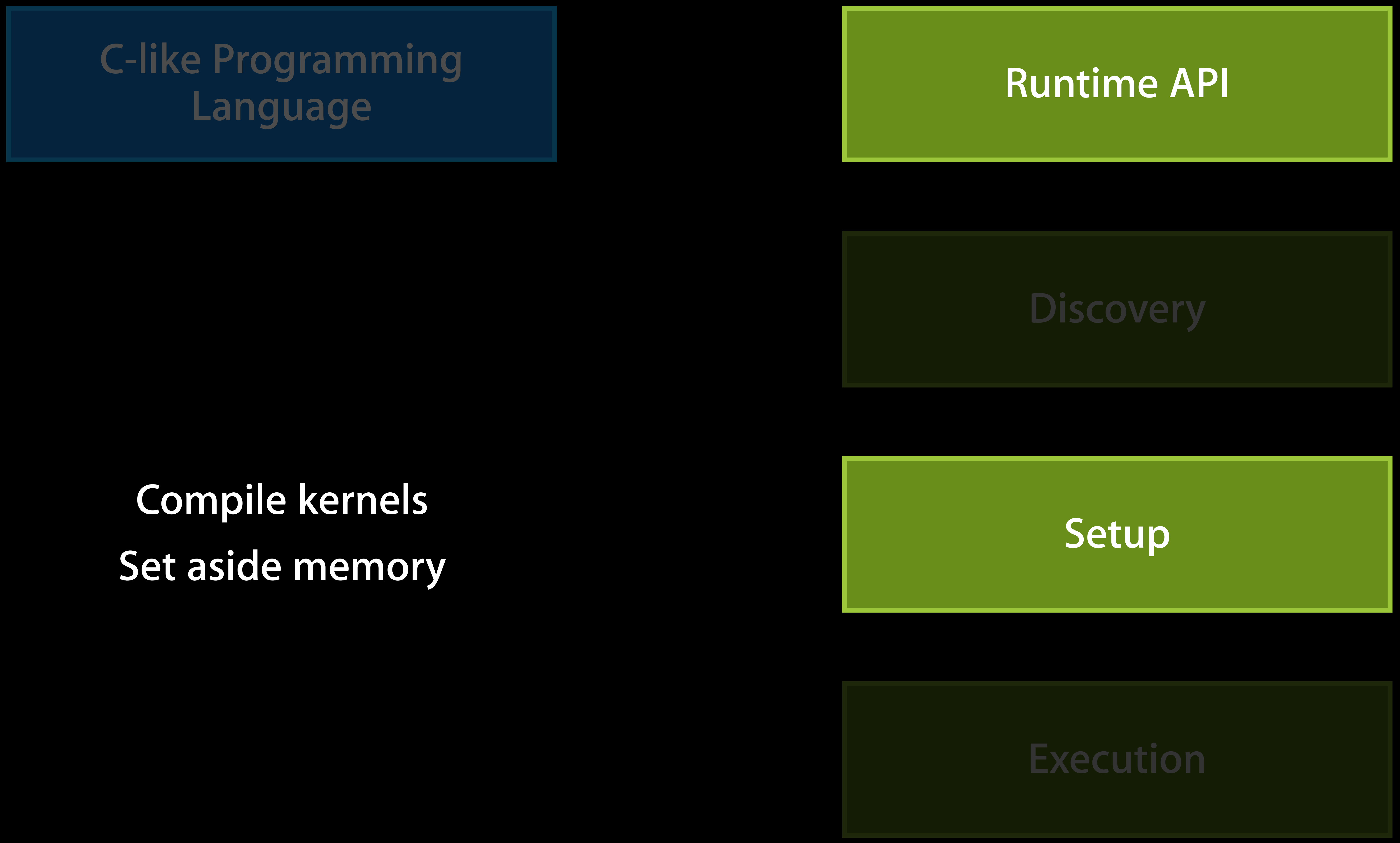
Runtime API

Discovery

Setup

Execution

# The OpenCL Programming Model



C-like Programming Language

Runtime API

Discovery

Compile kernels

Set aside memory

Setup

Execution

# The OpenCL Programming Model

C-like Programming Language

Runtime API

Discovery

Setup

Send commands to the device

Run the kernel!

Execution

# Practical Tasks with OpenCL

Abe Stephens, PhD

# Getting the Most Out of OpenCL in 10.9

- Decreasing startup time
- Saving power
- Getting more performance

# What Contributes to Slow Startup?

- clBuildProgram, clCompileProgram, clLinkProgram

# OpenCL Kernel Program Loading

- OpenCL C source compiled at runtime
- LLVM bitcode files compiled by Xcode
- Executable binary cached on first launch

# How Much Faster?

- Depends on program complexity & system behavior

```
#define READ_PIXEL(x,y) read_imagef(input_img,CLK_FILTER_NEAREST|
CLK_ADDRESS_CLAMP,coord+(int2)((x),(y)))
#define READ_PIXEL_LUM(x,y) convert_to_lum(READ_PIXEL((x),(y)))
...
kernel void processImage(read_only image2d_t input_img, write_only image2d_t
debug_img)
{
  const int2 coord = (int2)(get_global_id(0),get_global_id(1));
  const float p0 = READ_PIXEL_LUM(-1,-1);
  const float p1 = READ_PIXEL_LUM( 0,-1);
  const float p2 = READ_PIXEL_LUM( 1,-1);
  const float p3 = READ_PIXEL_LUM(-1, 0);
  ...
```

# How Much Faster?

- Depends on program complexity & system behavior
- Simple video example (2011 Mac Book Pro)
  - First launch:

    Source: 200ms

    Bitcode: 80ms
  - Warm launch

    Source: 1.5ms

    Bitcode: 1.1ms

    Executable Binary: 0.1ms

# Recommended Steps

- Compile *.cl source files to *.gpu_32.bc files in Xcode
- Load *.bc files and pass to clCreateProgramWithBinary

```
NSString* clPath = [[NSBundle mainBundle]
                    pathForResource:@"kernels"ofType:@"cl.gpu_32.bc"];
NSData* binary = [NSData dataWithContentsOfFile:clPath];

size_t binary_size              = [binary length];
const unsigned char *binary_ptr = [binary bytes];
cl_int status = 0;

p = clCreateProgramWithBinary(c, 1, &d, &binary_size, &binary_ptr, &status,
&err);

if ((err = clBuildProgram(p, 1, &d, NULL, NULL, NULL))) { ... }
```

# Save Binary to the Cache After Building

```
size_t binary_size;
clGetProgramInfo(p, CL_PROGRAM_BINARY_SIZES, sizeof(bin_size), &bin_size,
NULL);

NSMutableData* binary = [NSMutableData dataWithLength:binary_size];

unsigned char* bin_ptr = [binary mutableBytes];
clGetProgramInfo(p, CL_PROGRAM_BINARIES, [binary length], &bin_ptr, NULL);

[binary writeToFile:cache_file atomically:YES];
```

# Using an Executable Binary Cache

- When the app launches, try to load from the cache folder

```
NSArray* cache_path =

NSSearchPathForDirectoriesInDomains(NSCachesDirectory, NSUserDomainMask,
YES);

NSString* cache_file =
[NSString stringWithFormat:@"%@/%@/kernels.bin",[cache_path objectAtIndex:0],
[[NSBundle mainBundle] bundleIdentifier]];

NSData* binary = [NSData dataWithContentsOfFile:cache_file];
```

# Loading the Binary

- Fallback to the *.bc file if there is an error

```
if (binary) {

  size_t bin_size              = [binary length];
  const unsigned char *bin_ptr = [binary bytes];
  cl_int status = 0;
  p = clCreateProgramWithBinary(c, 1, &d, &bin_size, &bin_ptr, &status,
&err);

  // Abort if the error was anything other than CL_INVALID_BINARY
  if (err && (status != CL_INVALID_BINARY)) { /*Abort*/ }

  // If the binary loaded successfully, call clBuildProgram
  if (!err && (err = clBuildProgram(p, 1, &d, NULL, NULL, NULL))) { ... }
}
if (!p)
  // Fallback: Rebuild the program from source or bitcode
```

# Loading the Binary

- Fallback to the *.bc file if there is an error

```
if (binary) {

  size_t bin_size              = [binary length];
  const unsigned char *bin_ptr = [binary bytes];
  cl_int status = 0;
  p = clCreateProgramWithBinary(c, 1, &d, &bin_size, &bin_ptr, &status,
&err);

  // Abort if the error was anything other than CL_INVALID_BINARY
  if (err && (status != CL_INVALID_BINARY)) { /*Abort*/ }

  // If the binary loaded successfully, call clBuildProgram
  if (!err && (err = clBuildProgram(p, 1, &d, NULL, NULL, NULL))) { ... }
}
if (!p)
  // Fallback: Rebuild the program from source or bitcode
```

# Loading the Binary

- Fallback to the *.bc file if there is an error

```
if (binary) {

  size_t bin_size              = [binary length];
  const unsigned char *bin_ptr = [binary bytes];
  cl_int status = 0;
  p = clCreateProgramWithBinary(c, 1, &d, &bin_size, &bin_ptr, &status,
&err);

  // Abort if the error was anything other than CL_INVALID_BINARY
  if (err && (status != CL_INVALID_BINARY)) { /*Abort*/ }

  // If the binary loaded successfully, call clBuildProgram
  if (!err && (err = clBuildProgram(p, 1, &d, NULL, NULL, NULL))) { ... }
}
if (!p)
  // Fallback: Rebuild the program from source or bitcode
```

# Faster Program Loading

Time in milliseconds

| | 33 lines | 1130 lines | 4055 lines |
|---|---|---|---|
| First launch source | 200 | 2285 | 3000 |
| First launch bitcode | 80 | 1770 | 1750 |
| Warm launch source | 1.5 | 2.9 | 1800 |
| Warm launch bitcode | 1.1 | 2.1 | 2.32 |
| Executable binary | 0.1 | 0.5 | 0.6 |

# Using the Integrated or Discrete GPU

# Dual GPU Laptops

- OpenGL apps either:
  - Only run on discrete
  - Support automatic graphics switching
- Save power by switching from discrete to integrated

Mac Book Pro Retina

Intel HD 4000

Discrete

Integrated

# What’s New in 10.9

- OpenCL apps may choose to support switching too

# Handling Device Changes in NSOpenGLView

```
NSOpenGLContext* gl = [self openGLContext];
GLint newVirtualScreen = [gl currentVirtualScreen];
if ([self virtualScreen] != newVirtualScreen) {
  [self setVirtualScreen:newVirtualScreen];

  // Adapt usage to any GL capability changes
```

# Handling Device Changes in NSOpenGLView

```
NSOpenGLContext* gl = [self openGLContext];
GLint newVirtualScreen = [gl currentVirtualScreen];
if ([self virtualScreen] != newVirtualScreen) {
  [self setVirtualScreen:newVirtualScreen];

  // Adapt usage to any GL capability changes

  // Get the CL device for the virtual screen
  cl_device_id newDevice = NULL;
  clGetGLContextInfoAPPLE(c,[gl CGLContextObj],
                          CL_CGL_DEVICE_FOR_CURRENT_VIRTUAL_SCREEN_APPLE,
                          sizeof(newDevice),&newDevice,NULL);

  // Adapt usage to CL capability changes
}
```

# What OpenCL Does Automatically

- Most objects are context-level and work on all devices
  - cl_mem (images and buffers)
  - cl_kernel and set kernel args
  - cl_program, if built for both devices
  - cl_event dependencies

# What You Need to Check

- Context must contain both devices
- Build programs for both devices (for bitcode both are .gpu_32.bc)
- Create a command queue for each device

# Extensions

- For example: cl_khr_fp64

```
double
```

# Extensions

- For example: cl_khr_fp64

```
float // Enough precision?
```

# Kernel Info

CL_KERNEL_WORK_GROUP_SIZE might be different

# Performance Features

# Buffers vs. Images

- Buffer objects
  - Read/write from a kernel using pointers
  - Support atomic operations
  - May or may not be cached
- Image objects
  - Either read-only or write-only from a kernel
  - Hardware filtering
  - GPU texture cache, low latency

# Sometimes You Might Want Both

```
kernel void histogramGather(..., global float* histo) {
  ...
  histo[index] = count;
  ...
}

kernel void equalization(..., read_only image2d_t histo) {
  ...
  float interpolated = read_imagef(histo, CLK_FILTER_LINEAR|...).r;
  ...
}
```

# image2d from Buffer

- cl_khr_image2d_from_buffer
- Sampling modes and capabilities of an ordinary image2d

```
cl_image_format fmt = { CL_RGBA, CL_FLOAT };
size_t pixel = 16;
cl_mem_flags flags = CL_MEM_READ_WRITE;
cl_mem buffer_mem = clCreateBuffer(c,flags,w*h*pixel,NULL,&err);
cl_image_desc desc = {
  .image_type        = CL_MEM_OBJECT_TYPE_IMAGE2D,
    .image_width       = w,
    .image_height      = h,
    .image_row_pitch   = pitch,
    ...
    .buffer            = buffer_mem,
 };
 cl_mem img = clCreateImage(c,flags,&fmt,&desc,NULL,&err);
```

# image2d from Buffer Alignment

- CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT
  - For CL_MEM_USE_HOST_PTR buffers
- CL_DEVICE_IMAGE_PITCH_ALIGNMENT

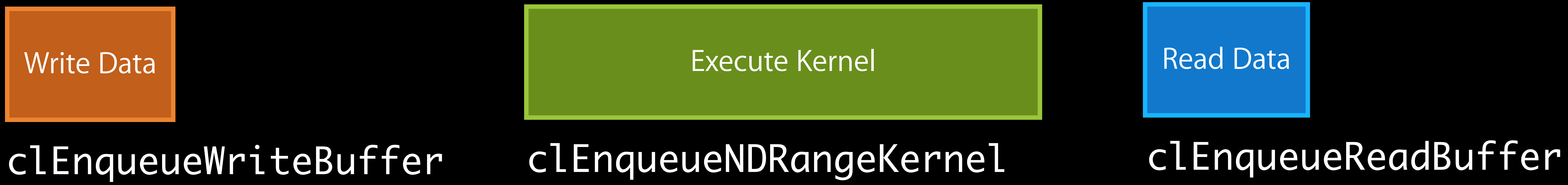
```
cl_image_desc desc = {
  .image_type        = CL_MEM_OBJECT_TYPE_IMAGE2D,
    .image_width       = w,
    .image_height      = h,
    .image_row_pitch   = pitch,
    ...
    .buffer            = buffer_mem,
  };
```

# Data Movement in Compute Apps

## Many apps write/execute/read

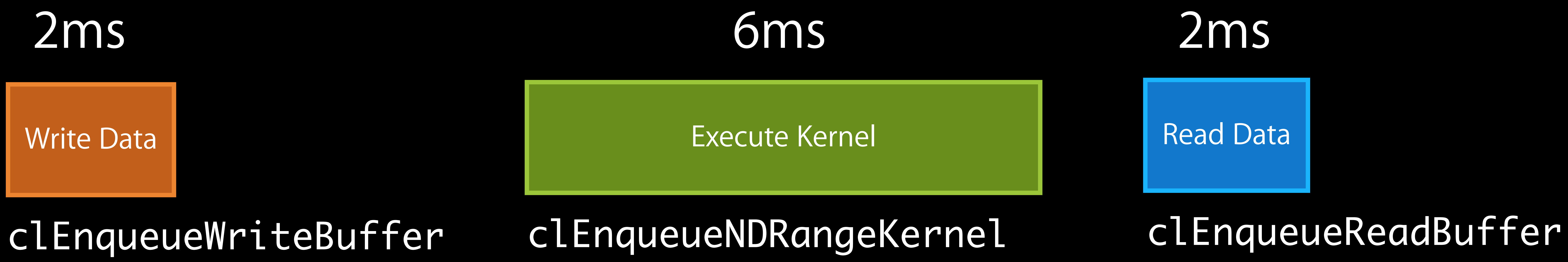
# Data Movement in Compute Apps

## Many apps write/execute/read



# Data Movement in Compute Apps

## Many apps write/execute/read



2ms

Write Data

`clEnqueueWriteBuffer`

6ms

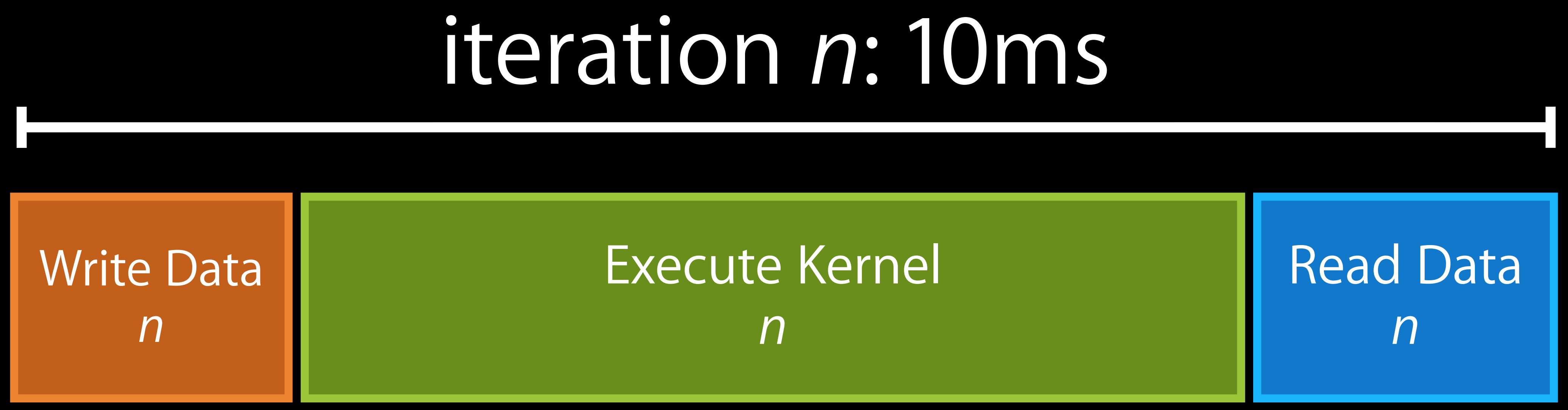
Execute Kernel

`clEnqueueNDRangeKernel`

2ms

Read Data

`clEnqueueReadBuffer`



iteration *n*: 10ms

Write Data *n*

Execute Kernel *n*

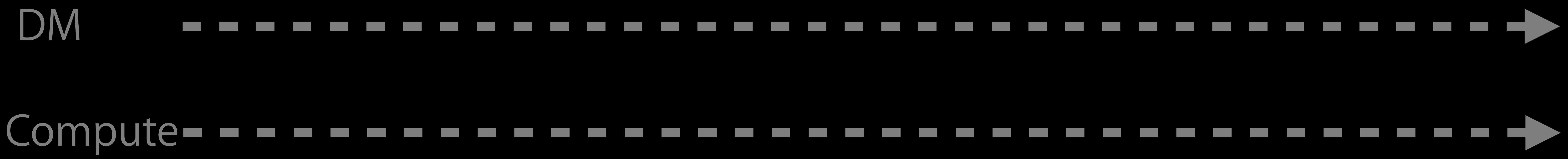
Read Data *n*

iteration *n*: 10ms
Kernel
Read Data *n-1*
Write Data *n*
Execute Kernel *n*
Read Data *n*
Write Data *n+1*
Execute Kernel *n+1*
Read Data *n+1*

100 iterations in 1000 ms

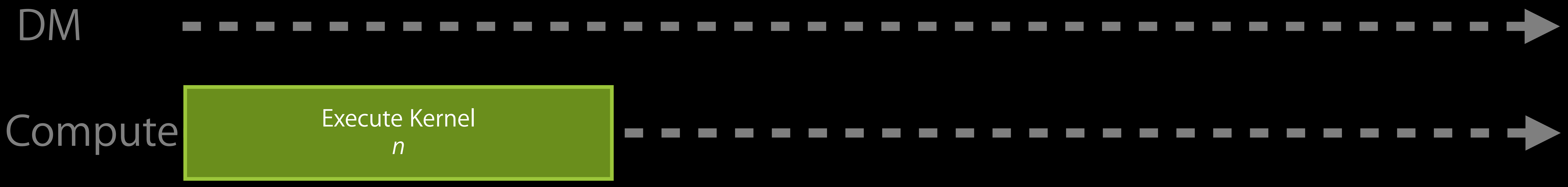
# Overlapping Read/Write and Compute

## Using DMA hardware



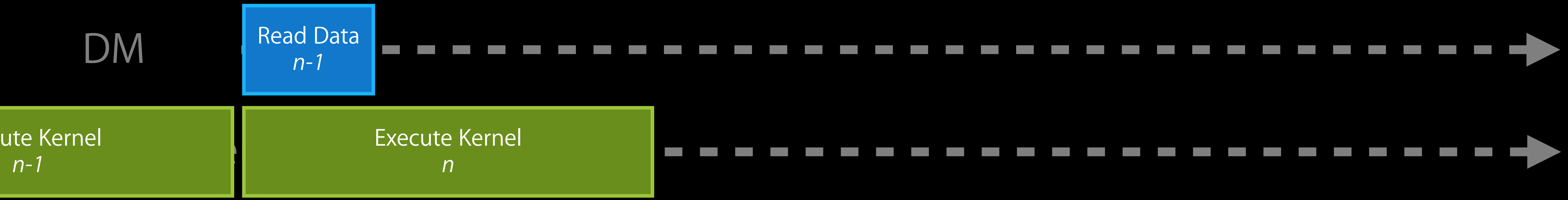
# Overlapping Read/Write and Compute

## Using DMA hardware



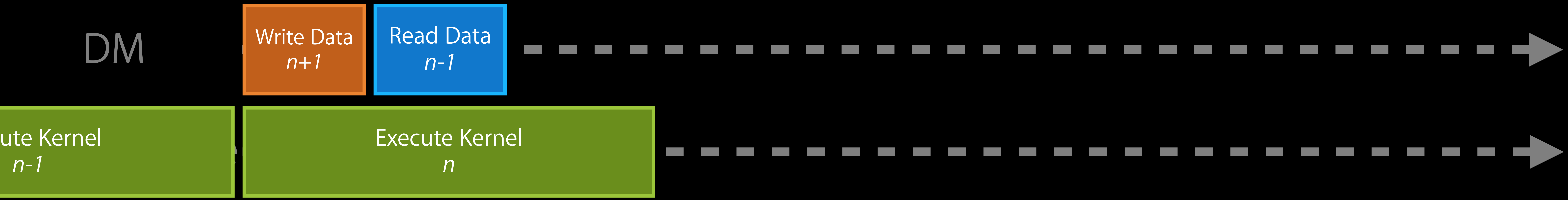
# Overlapping Read/Write and Compute

## Using DMA hardware



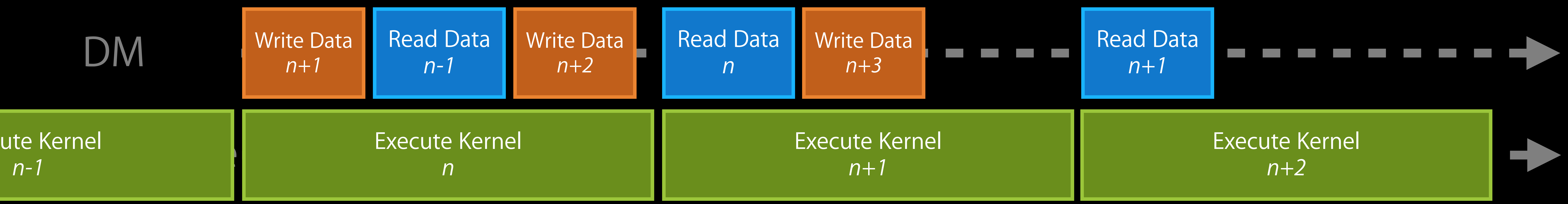
# Overlapping Read/Write and Compute

## Using DMA hardware



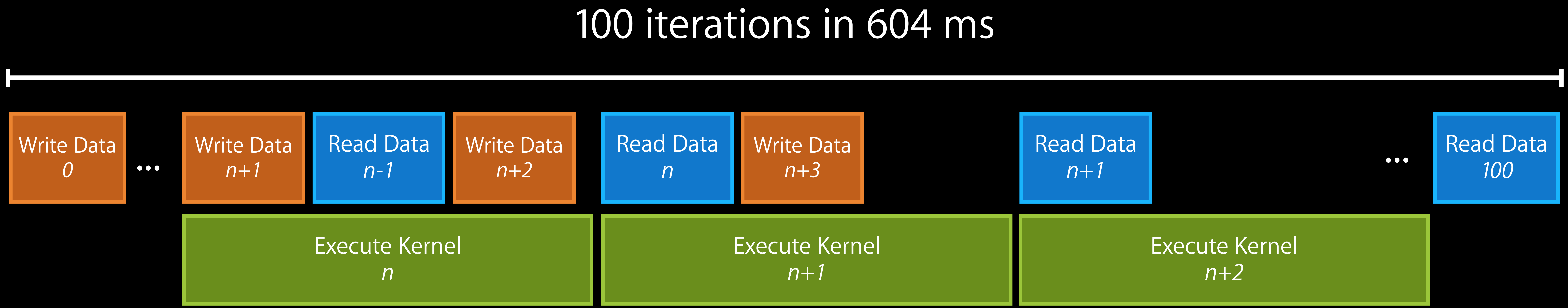
# Overlapping Read/Write and Compute

## Using DMA hardware



# Overlapping Read/Write and Compute

## Using DMA hardware



# Overlapping Read/Write and Compute

## For M input and output buffers

```
clEnqueueWriteBuffer(q,data[0],CL_FALSE,...);
clSetKernelArg(k,0,sizeof(cl_mem),&data[0]);
clEnqueueNDRangeKernel(q,k,...);

for (int i=1;i!=M-1;++i) {
  clEnqueueWriteBuffer(q,data[(i+1)],CL_FALSE,...);
  clSetKernelArg(k,0,sizeof(cl_mem),&data[i]);
  clEnqueueNDRangeKernel(q,k,...);
  clEnqueueReadBuffer(q,data[(i-1)],CL_FALSE,...);
}

clSetKernelArg(k,0,sizeof(cl_mem),&data[(M-1)]);
clEnqueueNDRangeKernel(q,k,...);
clEnqueueReadBuffer(q,data[(M-1)],CL_FALSE,...);
clFlush(q);
```

All enqueued commands
are non-blocking

# Programming Tips

- Prefer page aligned pointers for host data
    - CL_MEM_USE_HOST_PTR buffers and images
    - Source and destination for read and write commands

# Programming Tips

- Prefer page aligned pointers for host data
  - CL_MEM_USE_HOST_PTR buffers and images
  - Source and destination for read and write commands

```
cl_float2* host_ptr;
posix_memalign(&host_ptr,PAGE_SIZE,num_bytes);
...
cl_mem m = clCreateBuffer(c,CL_MEM_USE_HOST_PTR,num_bytes,host_ptr,&err);
...
free(host_ptr);
```

# Avoid clFinish

- Rarely needed in production code
- Useful for isolating problems or timing
    - Also use CL_LOG_ERROR=stderr
    - printf on the GPU

# OpenCL in Mavericks

- Faster program loading
- Save power with graphics switching
- Reduce data copying

# OpenCL Enhancements in Adobe Premiere Pro CC

**David McGavran**
Senior Engineering Manager Premiere Pro
Adobe Systems Inc.

# New Graphics Card Support

- Premiere Pro CS 6 card list
  - ATI Radeon HD 6750M
  - ATI Radeon HD 6770M

# New Graphics Card Support

- Premiere Pro CS 6 card list
  - ATI Radeon HD 6750M
  - ATI Radeon HD 6770M
- Premiere Pro CC card list
  - All CS 6 cards
  - AMD Radeon HD 7950
  - GeForce GT 650M
  - GeForce GTX 675MX
  - GeForce GTX 680
  - GeForce GTX 680MX
  - Quadro K5000
  - Any card that meets minimum requirements of 1G of RAM and basic shader tests

# Enhanced Performance

- Now uses pinned memory for faster access
- Uses OpenCL 1.2 extension for cl_khr_image2d_from_buffer
- Takes advantage of multiple GPU's for render

# More Accelerated Effects

## Intrinsics

Adjustment layers

Color space conversion

Deinterlacing

Compositing

Blending modes

Nested Sequences

Multicam

Time remapping

## Transitions

Additive dissolve

Cross dissolve

Dip to black

Dip to white

Film Dissolve

Push

## Effects

Alpha Adjust

Black & White

Brightness & Contrast

Color Balance

Color Pass

Color Replace

Crop

Drop Shadow

Sharpen Extract

Fast Color Corrector

Feather Edges

Gamma Correction

Garbage Matte

Horizontal Flip

Invert

Luma Corrector

Luma Curve

Noise

Proc Amp

RGB Color Corrector

RGB Curves

Sharpen

Three-way Color Corrector

Timecode

Tint

Track Matte

Ultra Keyer

Veritcal Flip

Video Limiter

Warp Stabilizer

# More Accelerated Effects

## Intrinsics

Adjustment layers

Color space conversion

Deinterlacing

Compositing

Blending modes

Nested Sequences

Multicam

Time remapping

## Transitions

Additive dissolve

Cross dissolve

Dip to black

Dip to white

Film Dissolve

Push

Wipe

Slide

## Effects

Alpha Adjust

Black & White

Brightness & Contrast

Color Balance

Color Pass

Color Replace

Crop

Drop Shadow

Sharpen Extract

Fast Color Corrector

Feather Edges

Gamma Correction

Garbage Matte

Horizontal Flip

Invert

Luma Corrector

Luma Curve

Noise

Proc Amp

RGB Color Corrector

RGB Curves

Sharpen

Three-way Color Corrector

Timecode

Tint

Track Matte

Ultra Keyer

Veritcal Flip

Video Limiter

Warp Stabilizer

Gaussian Blur

Directional Blur

Fast Blur

Lumetri Deep Color Engine

# Lumetri Deep Color Engine

- ASCCombined
- BleachBypass
- Fade
- 1DLut
- 1D3DLut
- 3DLut
- ColorMatch
- ColorMatch2
- ColorSpace
- Convolve
- ConvolveH
- ConvolveV
- Gain
- GaussianBlurRange
- HDRLayer
- AutoColorMatch
- GainOffset
- Primary
- Technicolor3strip
- Tinting
- SimplePrimary
- StereoColorMatch
- SecondaryPass1
- SecondaryPass2
- SecondaryPass3
- SecondaryPass4
- ShadingMask
- AntiAliasComposerH
- AntiAliasComposerV
- AntiAliasH
- AntiAliasV
- BloomH
- BloomV
- Copperplate
- CrayonDrawing
- Day2NiteH
- Day2NiteV
- DegrainPass1
- DegrainPass2 Dithering
- Emboss
- Inversion
- Keyer
- KuwaharaFilter5x5
- KuwaharaFilter7x7
- LegalizeNTSC
- LegalizePAL
- MedianFilter3x3H
- MedianFilter3x3V
- MedianFilter5x5H
- MedianFilter5x5V
- MedianFilter7x7H
- MedianFilter7x7V
- Night
- Outline
- PaletteCut
- SepiaTone
- SobelOperator
- Technicolor2strip

# Performance Improvements

- 30% Faster
  - Using pinned memory and cl_khr_image2d_from_buffer on a multilayer render
- 200% Faster
  - Taking all optimizations into account on same project

# Other Possibilities for OpenCL

- □ Increase set of supported effects
- □ Supporting third party effects
- □ GPU encoding and decoding
- □ Multiple GPU support
- □ GPU scopes

# Other Possibilities for OpenCL

- ☑ Increase set of supported effects
- □ Supporting third party effects
- □ GPU encoding and decoding
- □ Multiple GPU support
- □ GPU scopes

# Other Possibilities for OpenCL

- ☑ Increase set of supported effects
- ☑ Supporting third party effects
- □ GPU encoding and decoding
- □ Multiple GPU support
- □ GPU scopes

# Other Possibilities for OpenCL

- ☑ Increase set of supported effects
- ☑ Supporting third party effects
- □ GPU encoding and decoding
- ☑ Multiple GPU support
- □ GPU scopes

*Demo*

# Apple Evangelists

## Contact information

**Allan Schaffer**
Graphics and Game Technologies Evangelist
aschaffer@apple.com

**Apple Developer Forums**
http://devforums.apple.com/

**Developer Documentation**
http://developer.apple.com/library/

# Related Sessions

| | |
|---|---|
| What's New in OpenGL for OS X | Marina<br>Thursday 2:00PM |
| Core Image Effects and Techniques | Mission<br>Friday 10:15AM |

# Labs

| | |
|---|---|
| OpenCL Lab | Graphics and Games Lab B<br>**Thursday 4:30PM** |
| OpenGL and OpenGL ES Lab | Graphics and Games Lab A<br>**Thursday 2:00PM** |
| Core Image Lab | Graphics and Games Lab B<br>**Friday 11:30AM** |

WWDC2013