Neural Networks and Accelerate

Session 715

Eric Bainville Core OS, Vector and Numerics Group
Steve Canon Core OS, Vector and Numerics Group

Performance Libraries for CPU

That thing we do in the Vector and Numerics group

Optimized Low-Level Libraries

Optimized Low-Level Libraries

Accelerate - Image processing: vImage

Optimized Low-Level Libraries

Accelerate - Image processing: vImage

Accelerate - Signal processing: vDSP

Optimized Low-Level Libraries

Accelerate - Image processing: vImage

Accelerate - Signal processing: vDSP

Accelerate - Linear algebra: BLAS, SparseBLAS, LAPACK, LinearAlgebra

Optimized Low-Level Libraries

Accelerate - Image processing: vImage

Accelerate - Signal processing: vDSP

Accelerate - Linear algebra: BLAS, SparseBLAS, LAPACK, LinearAlgebra

Vector extensions: simd

Optimized Low-Level Libraries

Accelerate - Image processing: **vImage**

Accelerate - Signal processing: **vDSP**

Accelerate - Linear algebra: **BLAS, SparseBLAS, LAPACK, LinearAlgebra**

Vector extensions: **simd**

Lossless compression: **Compression**

Optimized Low-Level Libraries

Accelerate - Image processing: vImage

Accelerate - Signal processing: vDSP

Accelerate - Linear algebra: BLAS, SparseBLAS, LAPACK, LinearAlgebra

Vector extensions: simd

Lossless compression: Compression

Optimized for all supported CPUs

Agenda

Agenda

Lossless compression: **Compression**

Agenda

Lossless compression: Compression

Accelerate - Machine learning: BNNS

Agenda

Lossless compression: Compression

Accelerate - Machine learning: BNNS

Accelerate - Numerical integration: Quadrature

Agenda

Lossless compression: Compression

Accelerate - Machine learning: BNNS

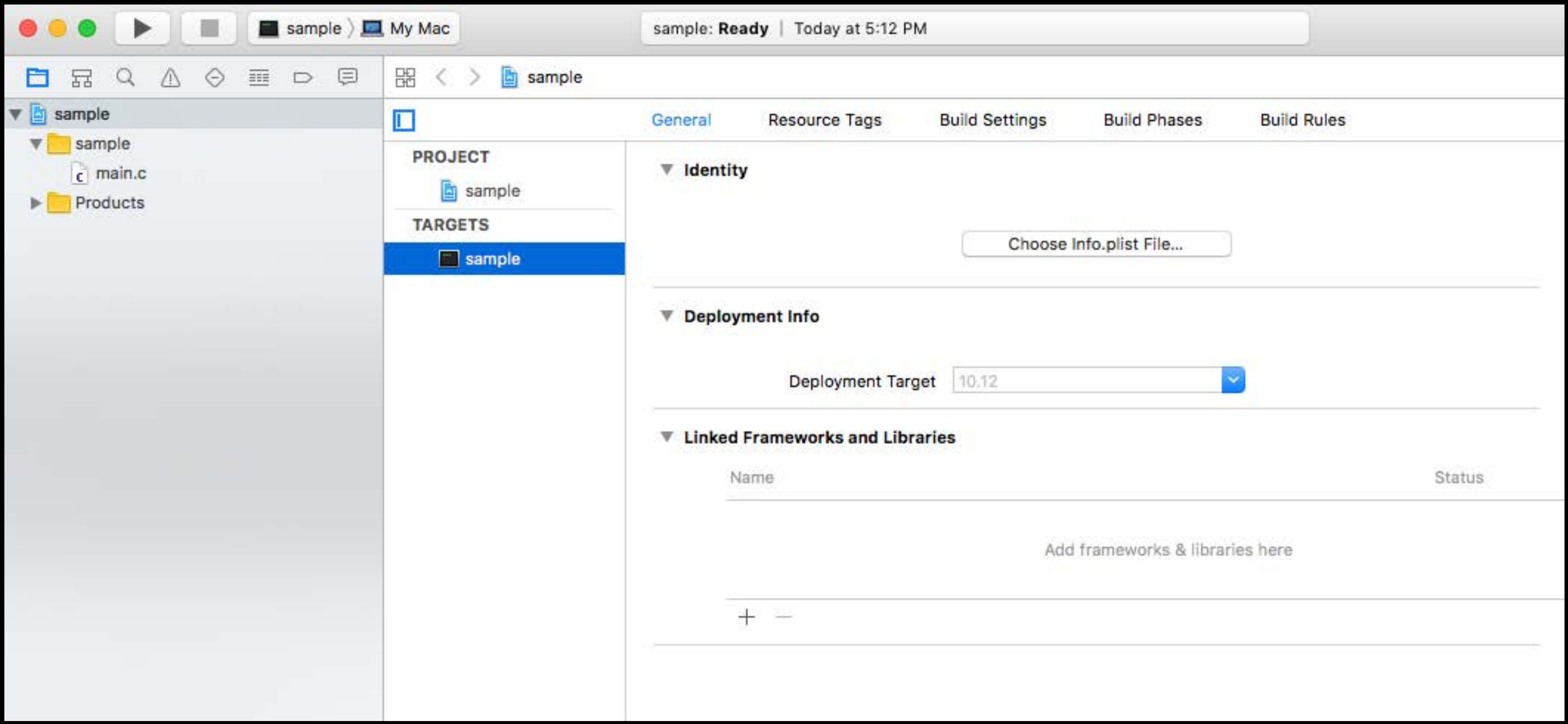
Accelerate - Numerical integration: Quadrature

Vector extensions: simd

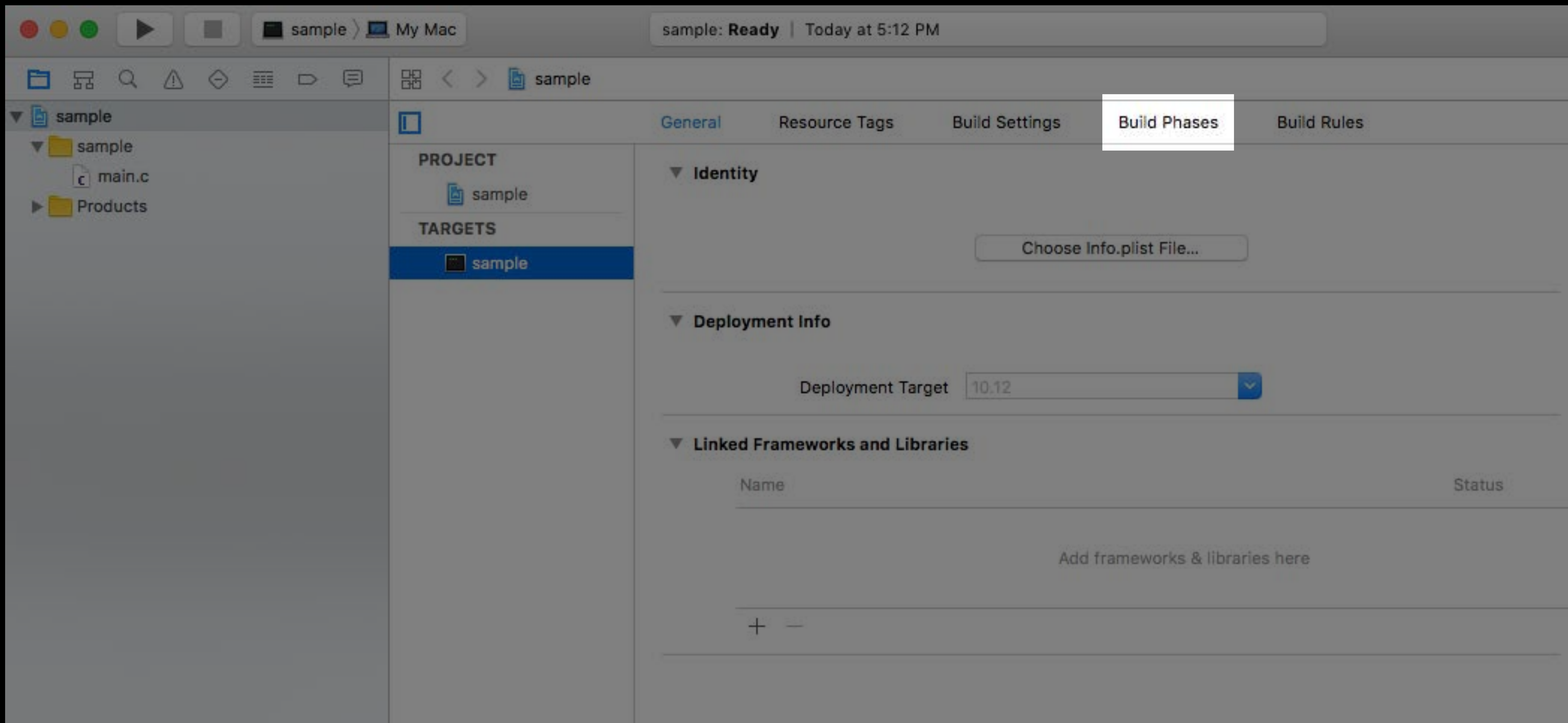
Using Accelerate

```
// Swift
import Accelerate

// C / C++
#include <Accelerate/Accelerate.h>

// Objective-C
@import Accelerate
```

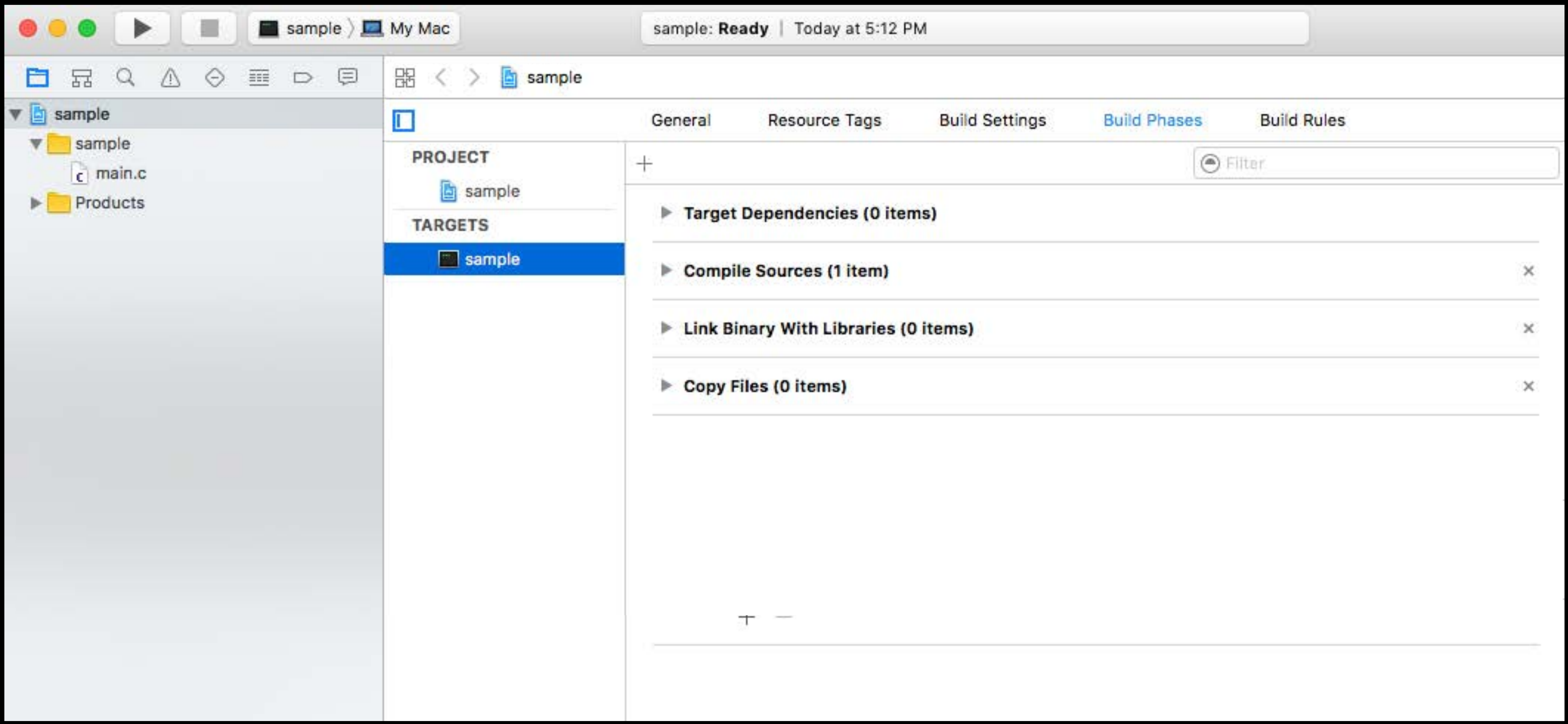
Using the Accelerate Framework



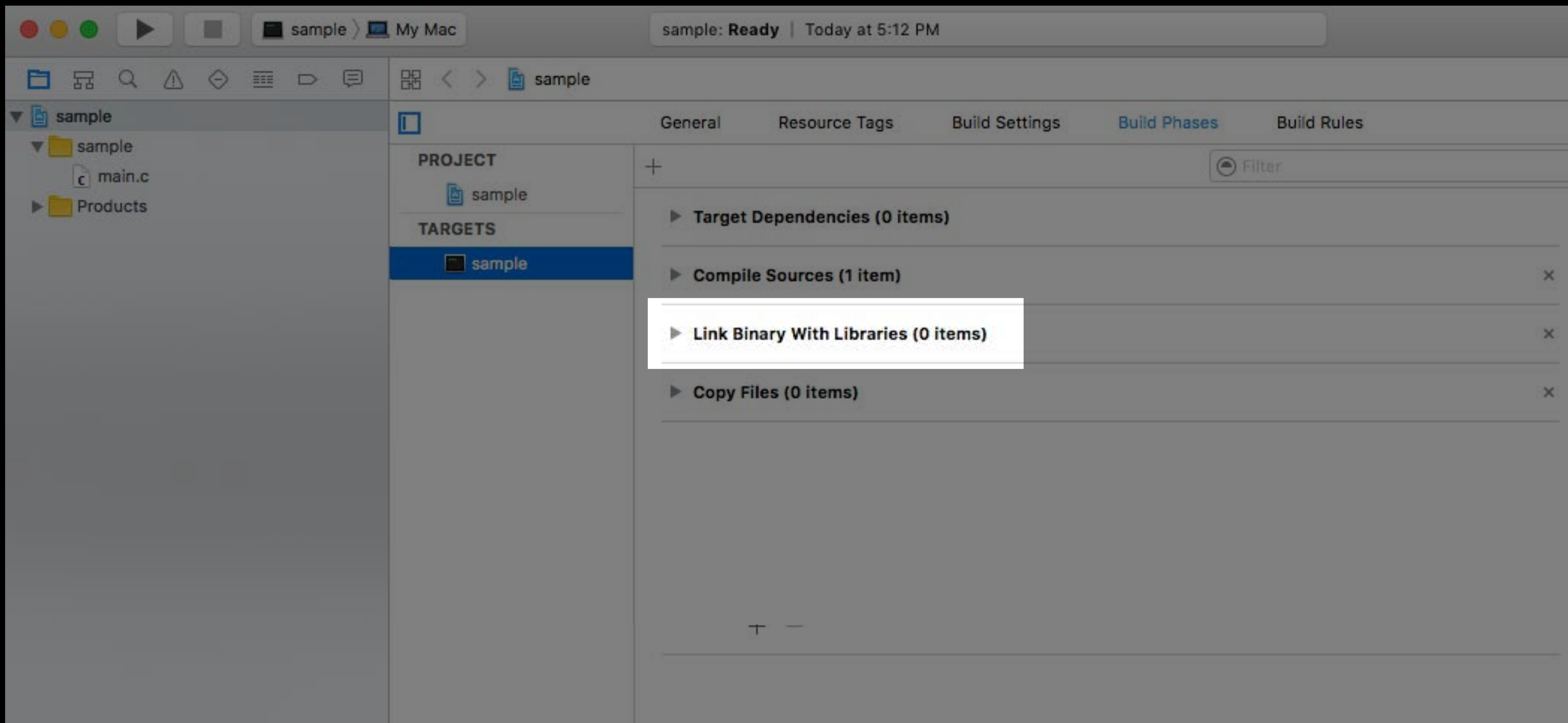
Using the Accelerate Framework



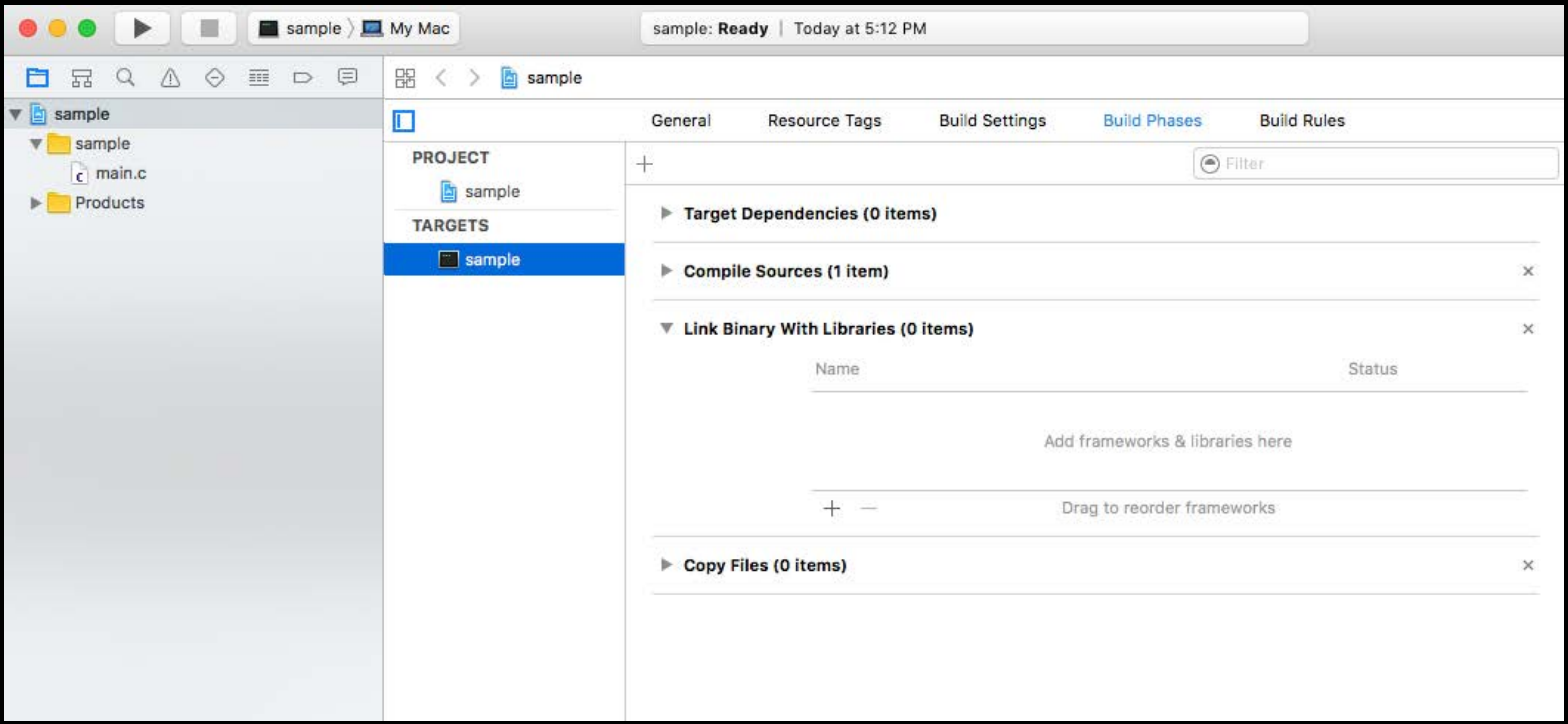
Using the Accelerate Framework



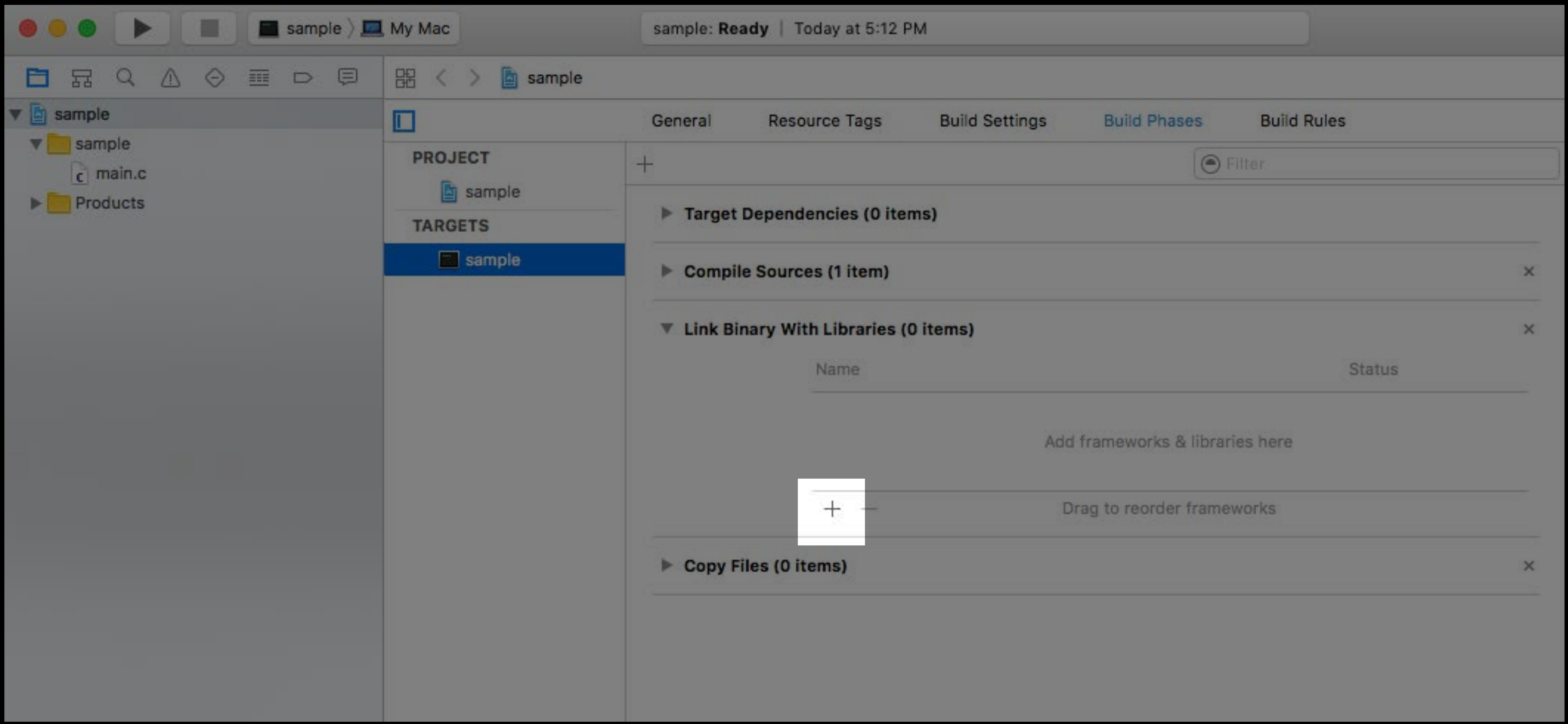
Using the Accelerate Framework



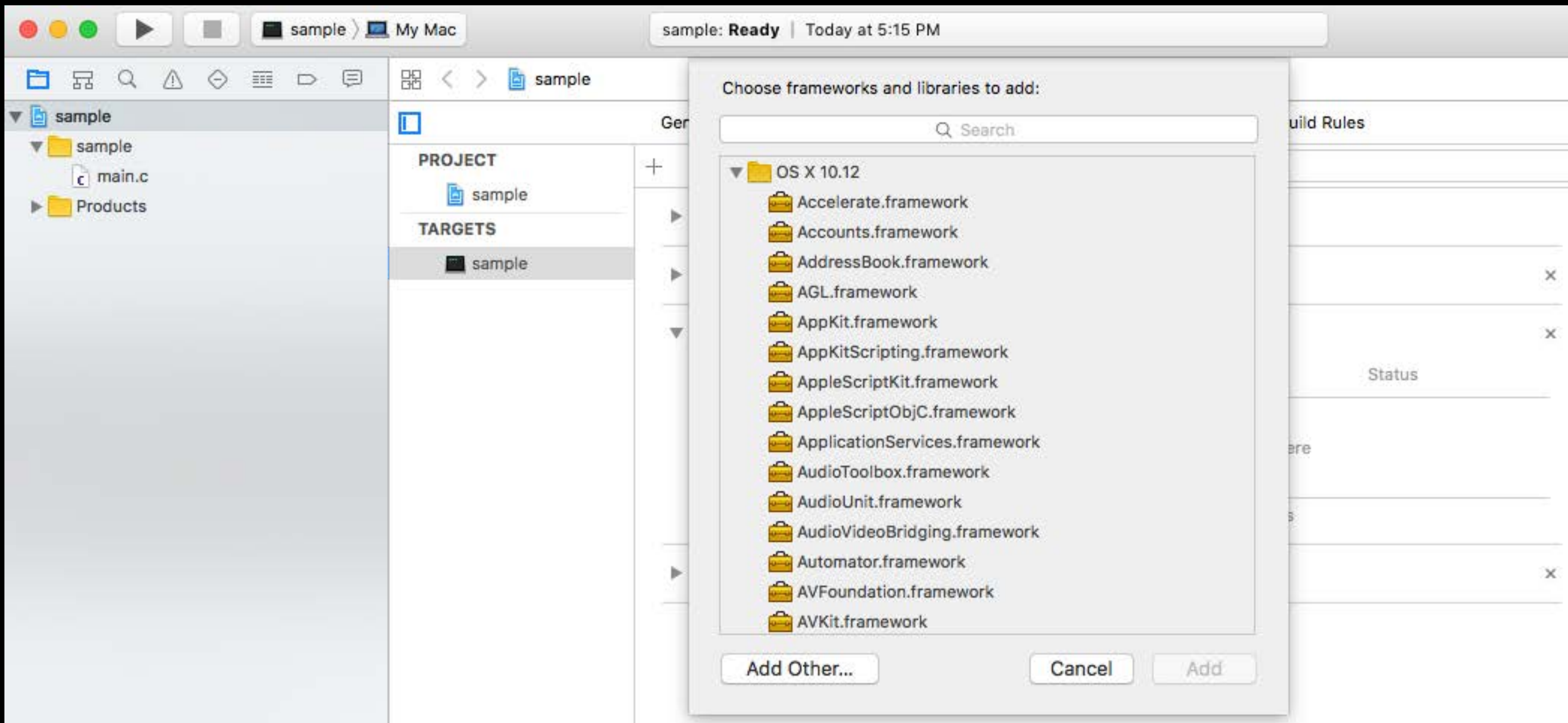
Using the Accelerate Framework



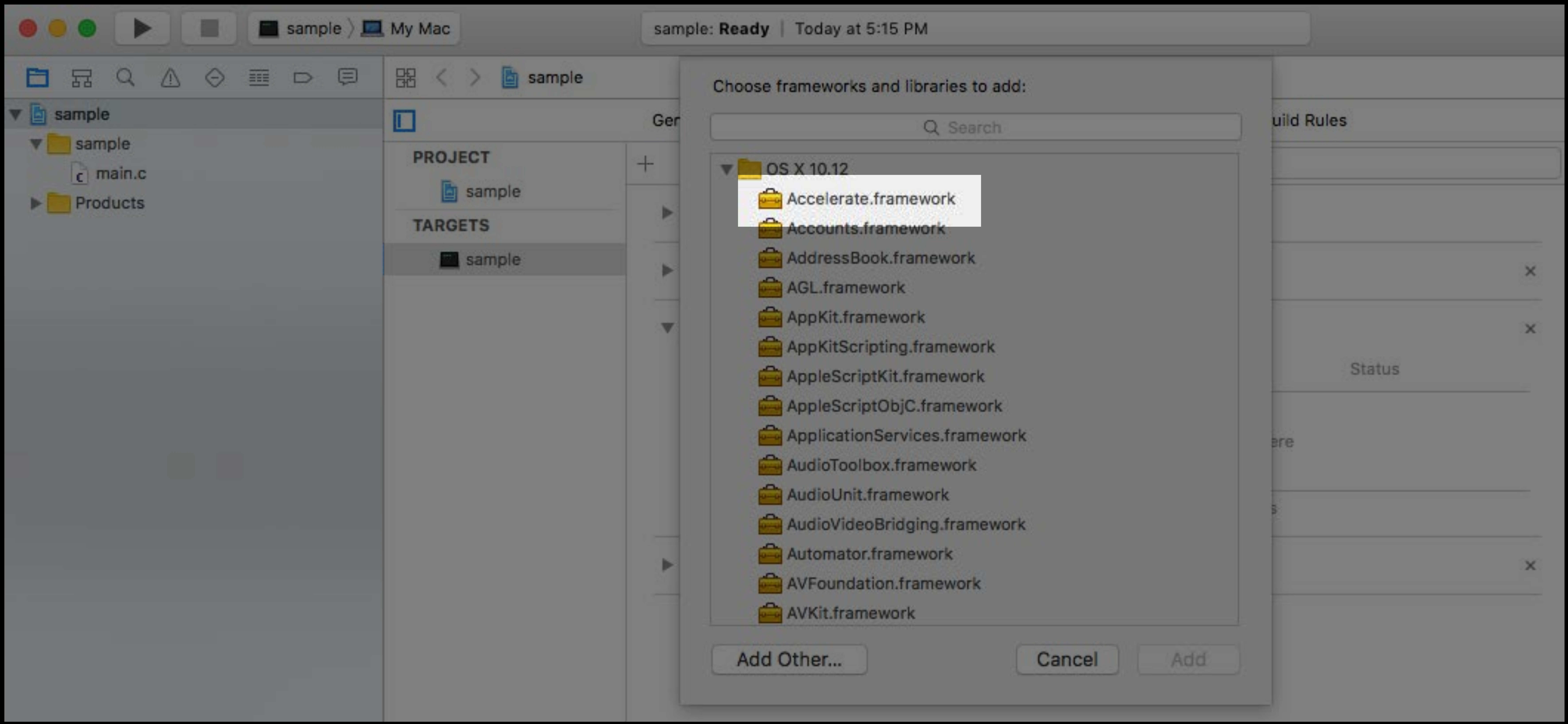
Using the Accelerate Framework



Using the Accelerate Framework



Using the Accelerate Framework



Compression

Remember last year?

Compression

LZFSE

Compression

LZFSE

Compression

LZFSE

LZFSE is now Open Source

Compression

LZFSE

LZFSE is now Open Source

Hosted on github.com/lzfse

Compression

LZFSE

LZFSE is now Open Source

Hosted on github.com/lzfse

BSD license

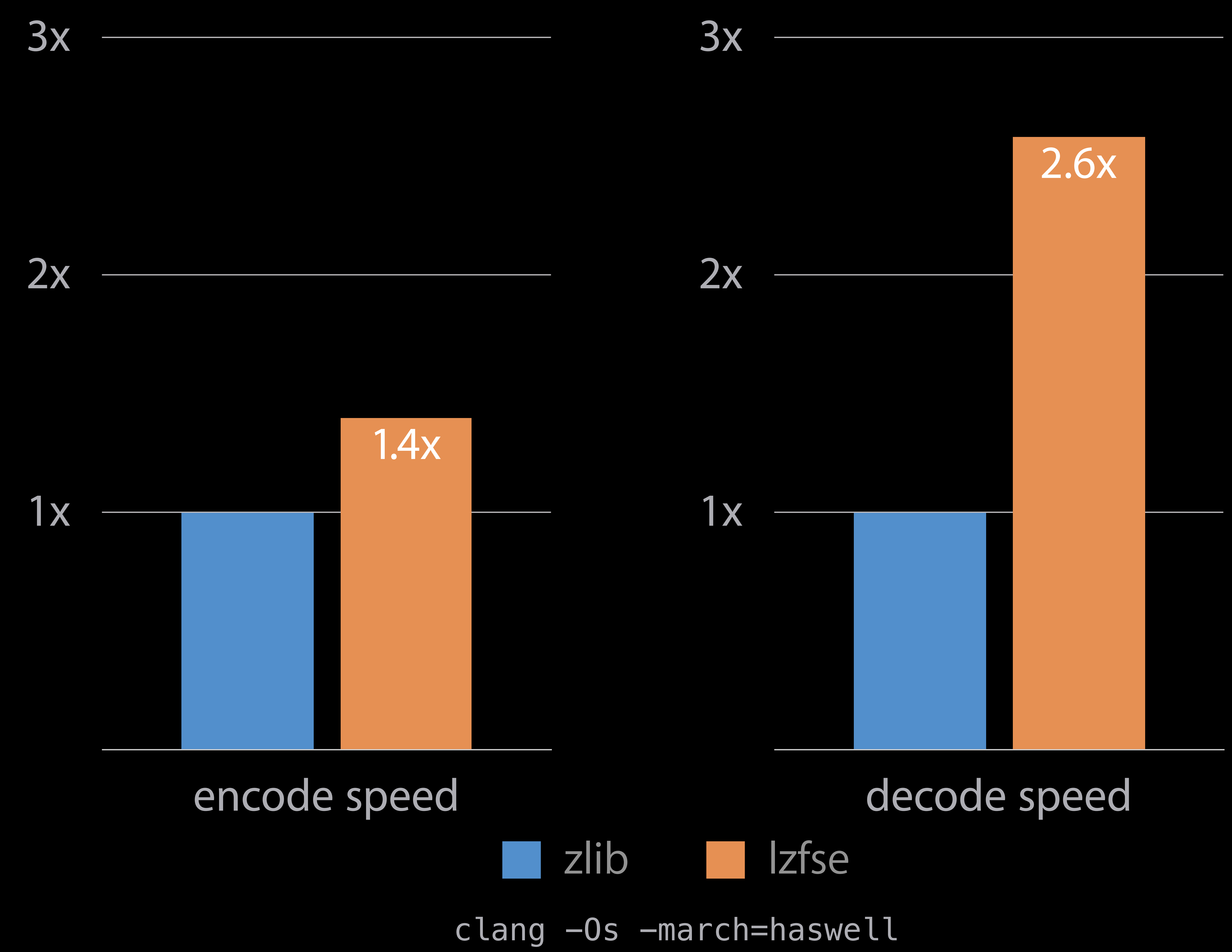
Compression

LZFSE

LZFSE is now Open Source

Hosted on github.com/lzfse

BSD license



BNNS

Basic Neural Network Subroutines

BNNS

New in Accelerate

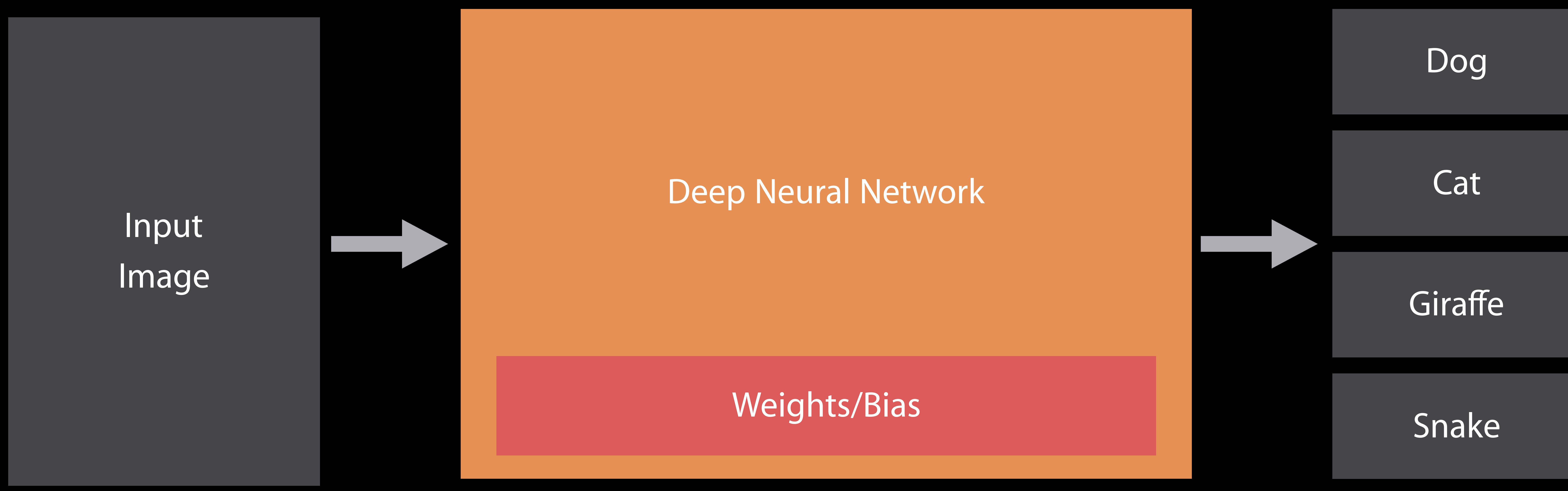
NEW

BNNS = Basic Neural Network Subroutines

BLAS = Basic Linear Algebra Subroutines

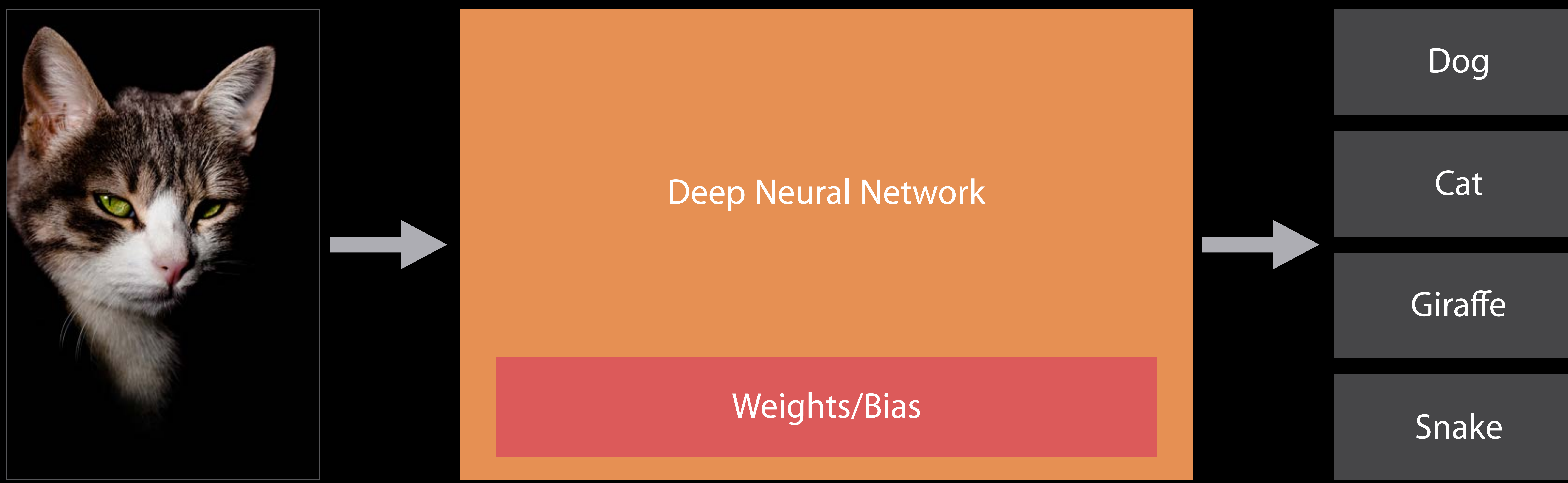
Deep Neural Network

Training



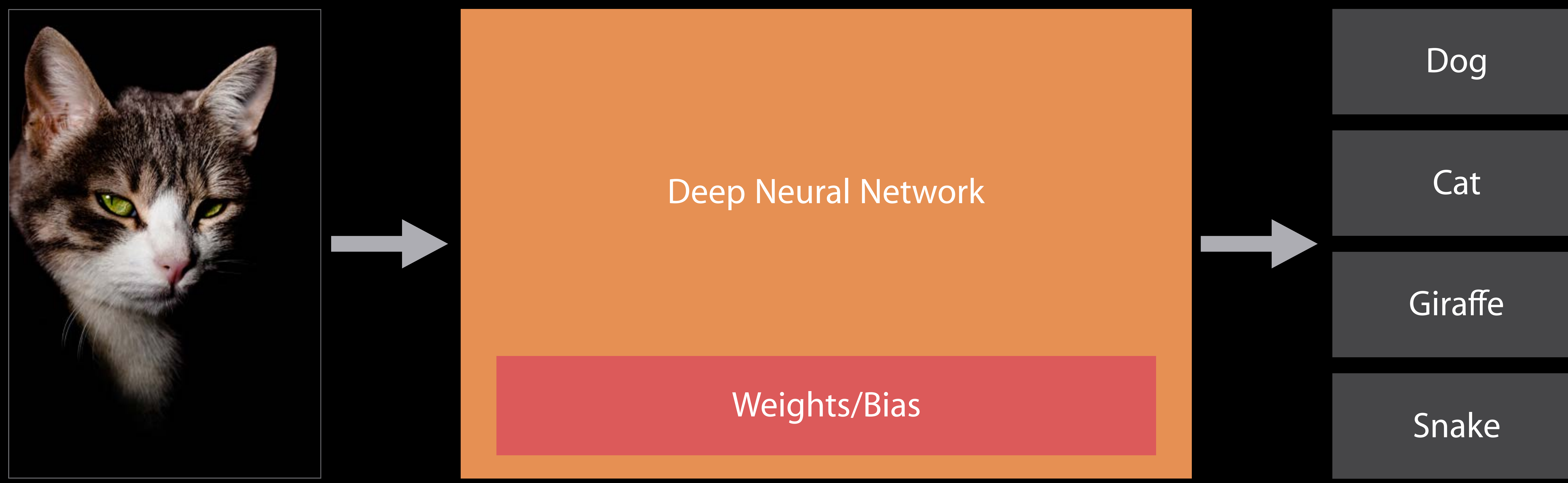
Deep Neural Network

Training



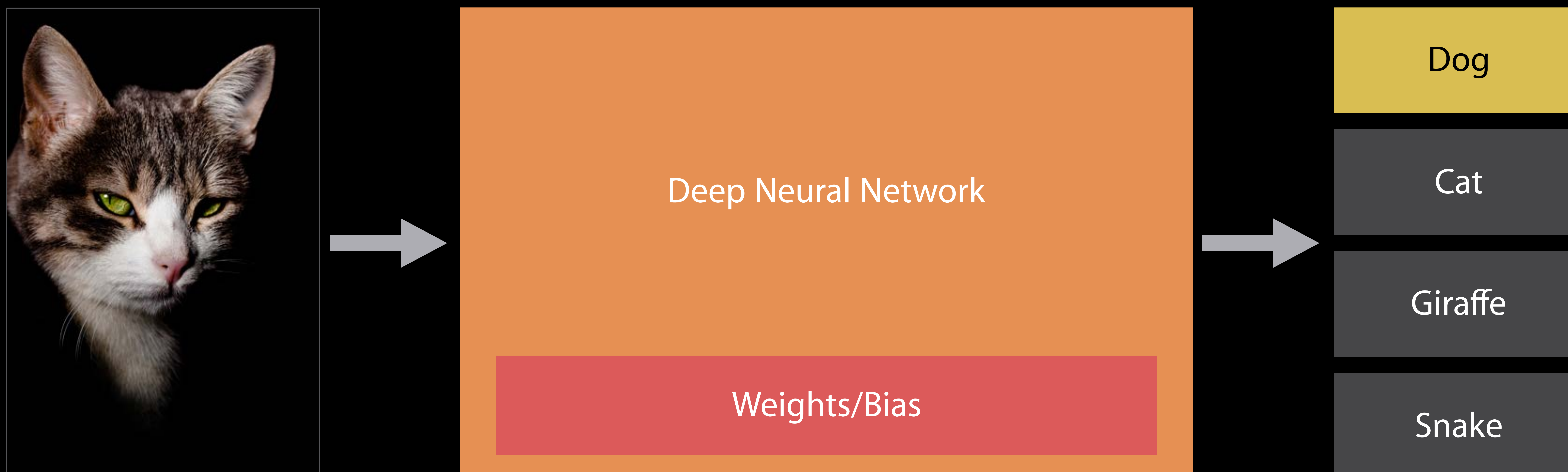
Deep Neural Network

Training



Deep Neural Network

Training



Deep Neural Network

Training

Deep Neural Network

Weights/Bias

Dog

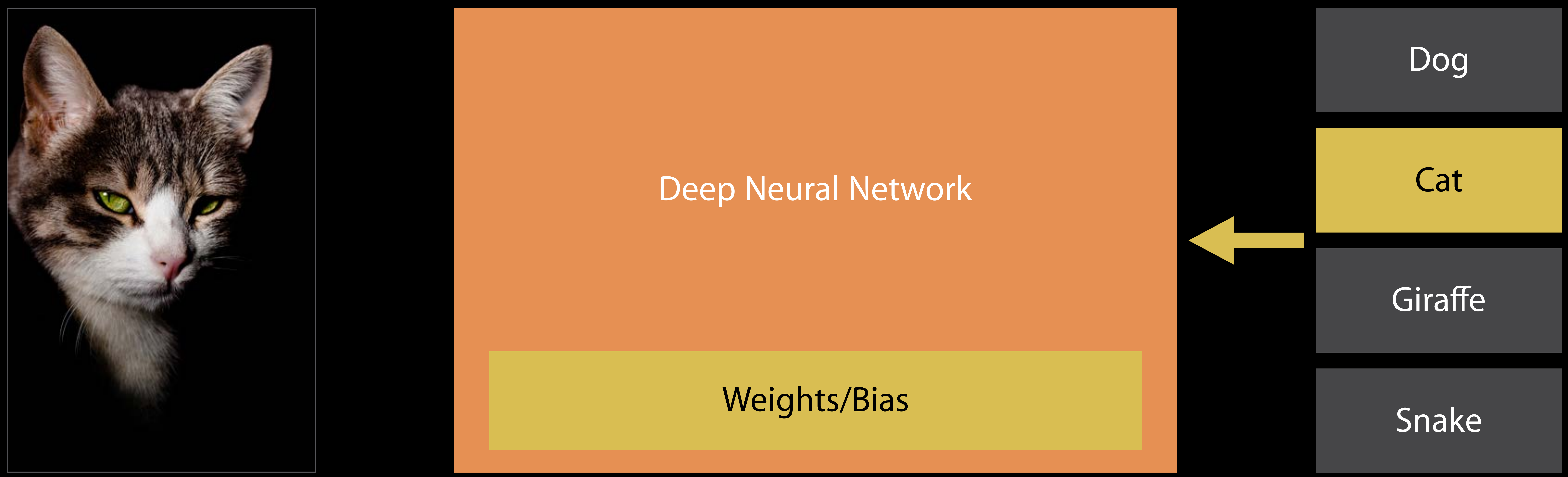
Cat

Giraffe

Snake

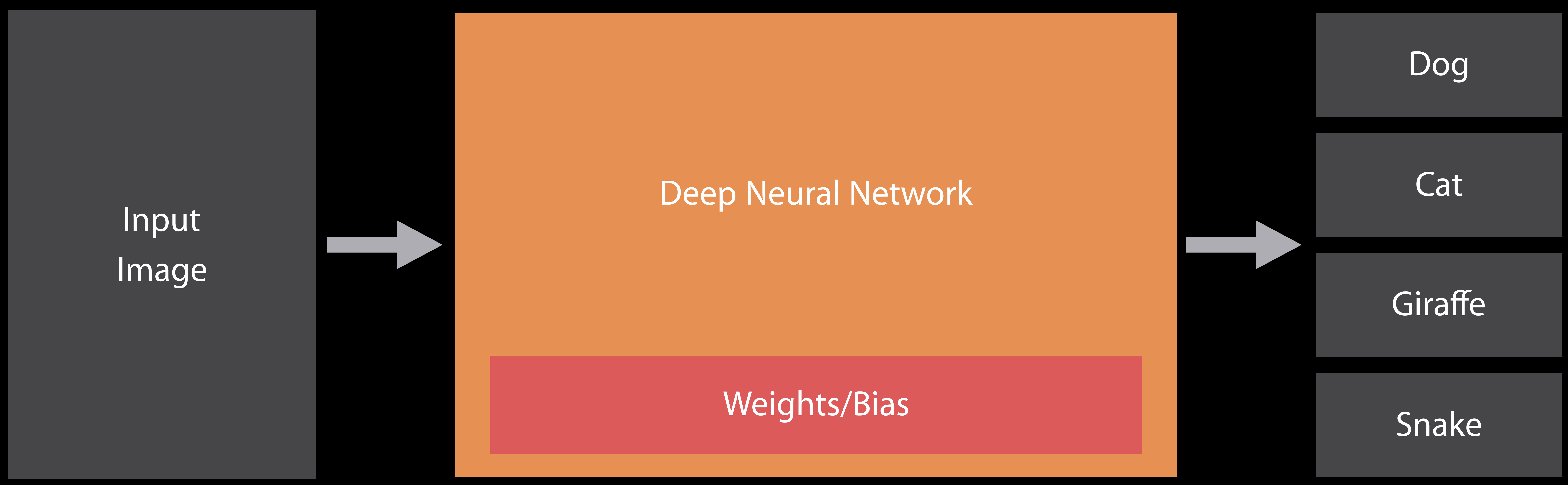
Deep Neural Network

Training



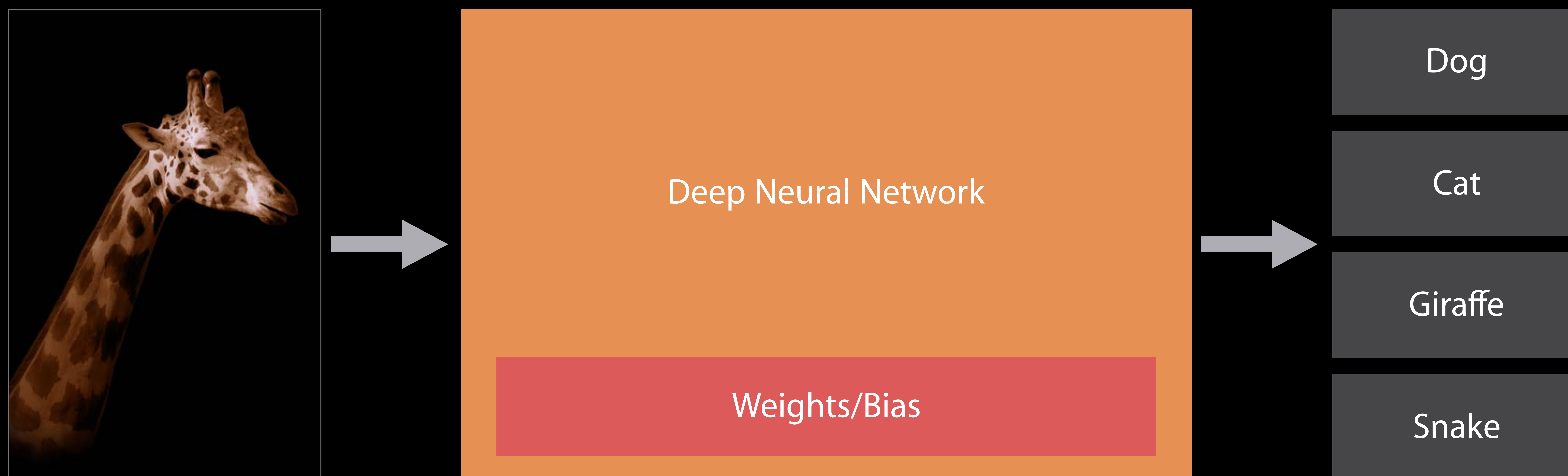
Deep Neural Network

Inference



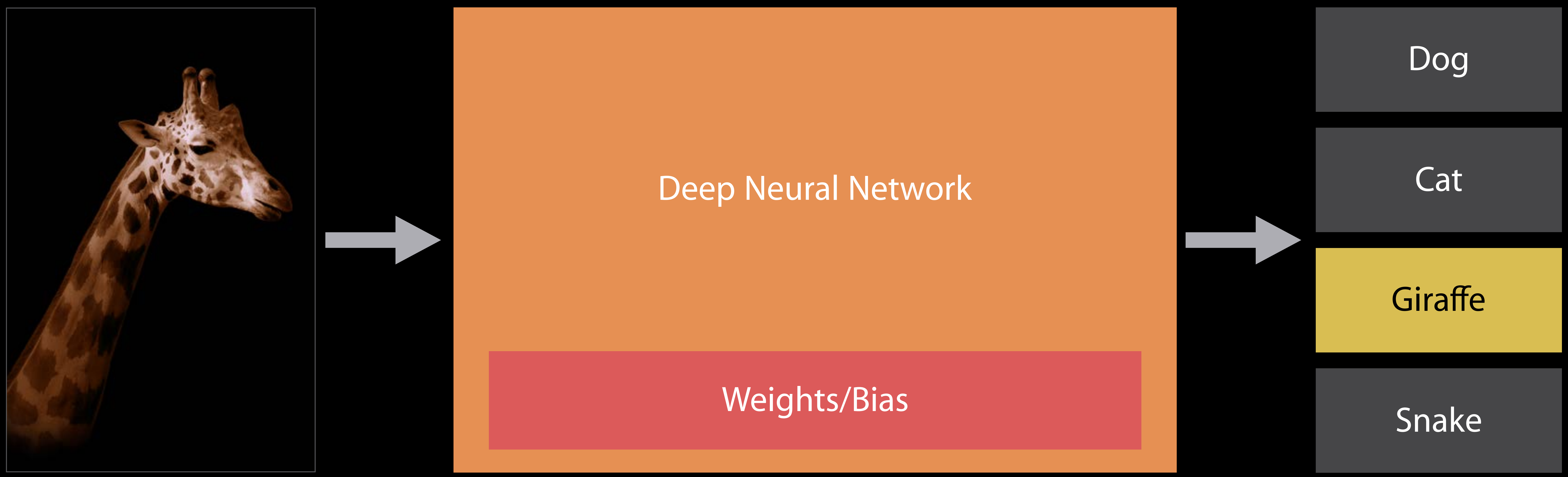
Deep Neural Network

Inference



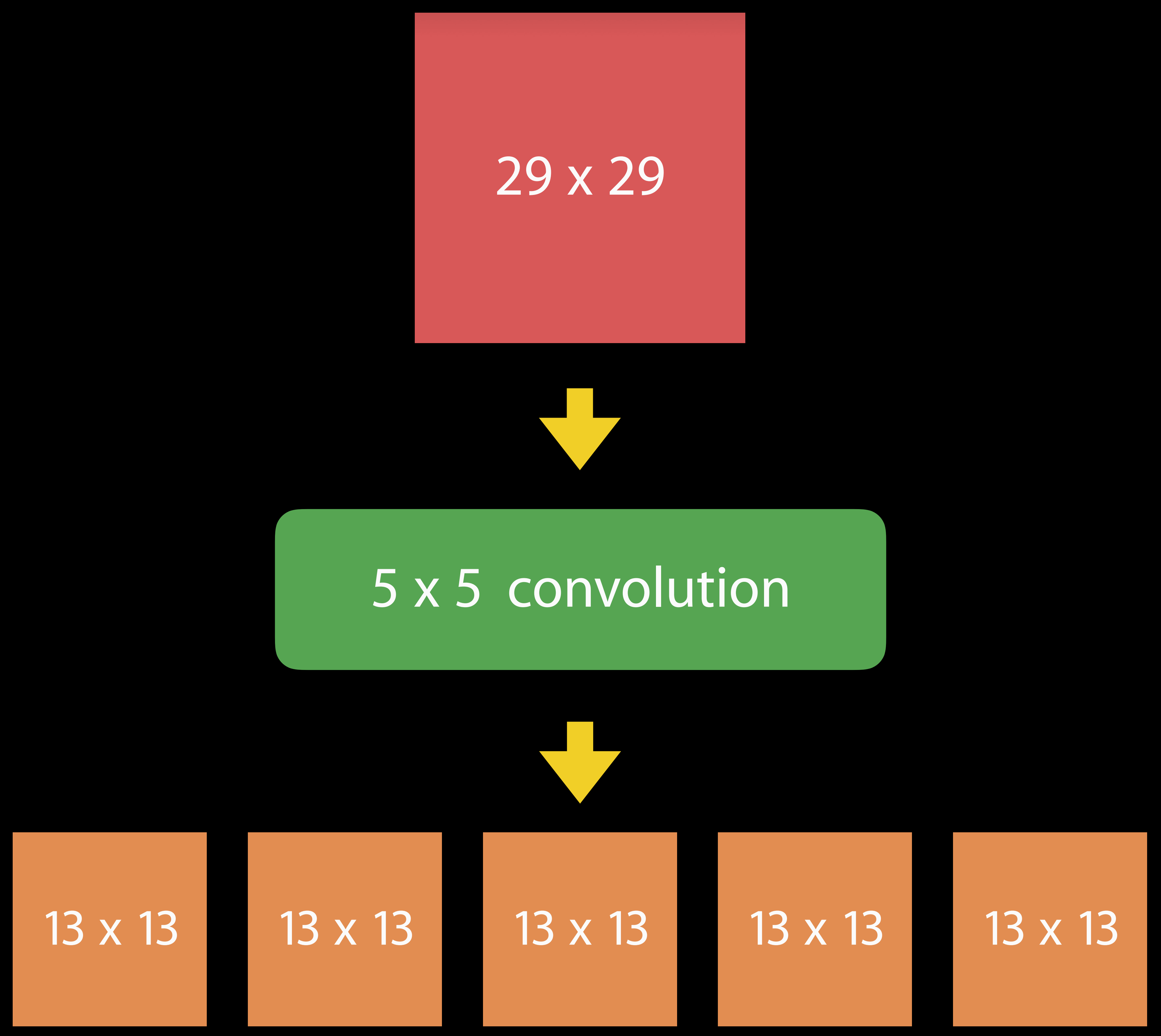
Deep Neural Network

Inference



Digit Recognition Network

Example



29 x 29 x 1

13 x 13 x 5

Digit Recognition Network

Example

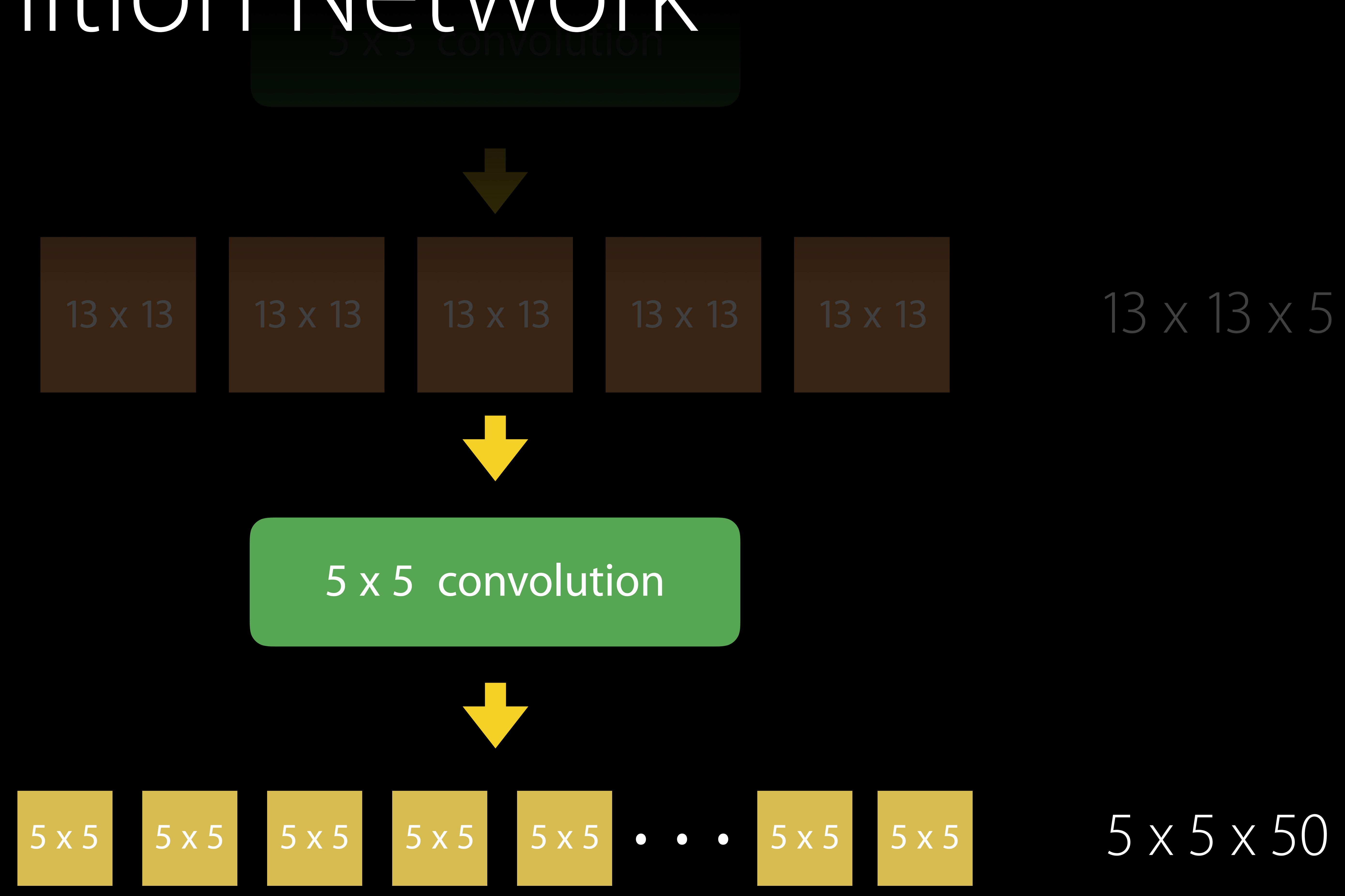
5 x 5 convolution

13 x 13 | 13 x 13 | 13 x 13 | 13 x 13 | 13 x 13

13 x 13 x 5

Digit Recognition Network

Example

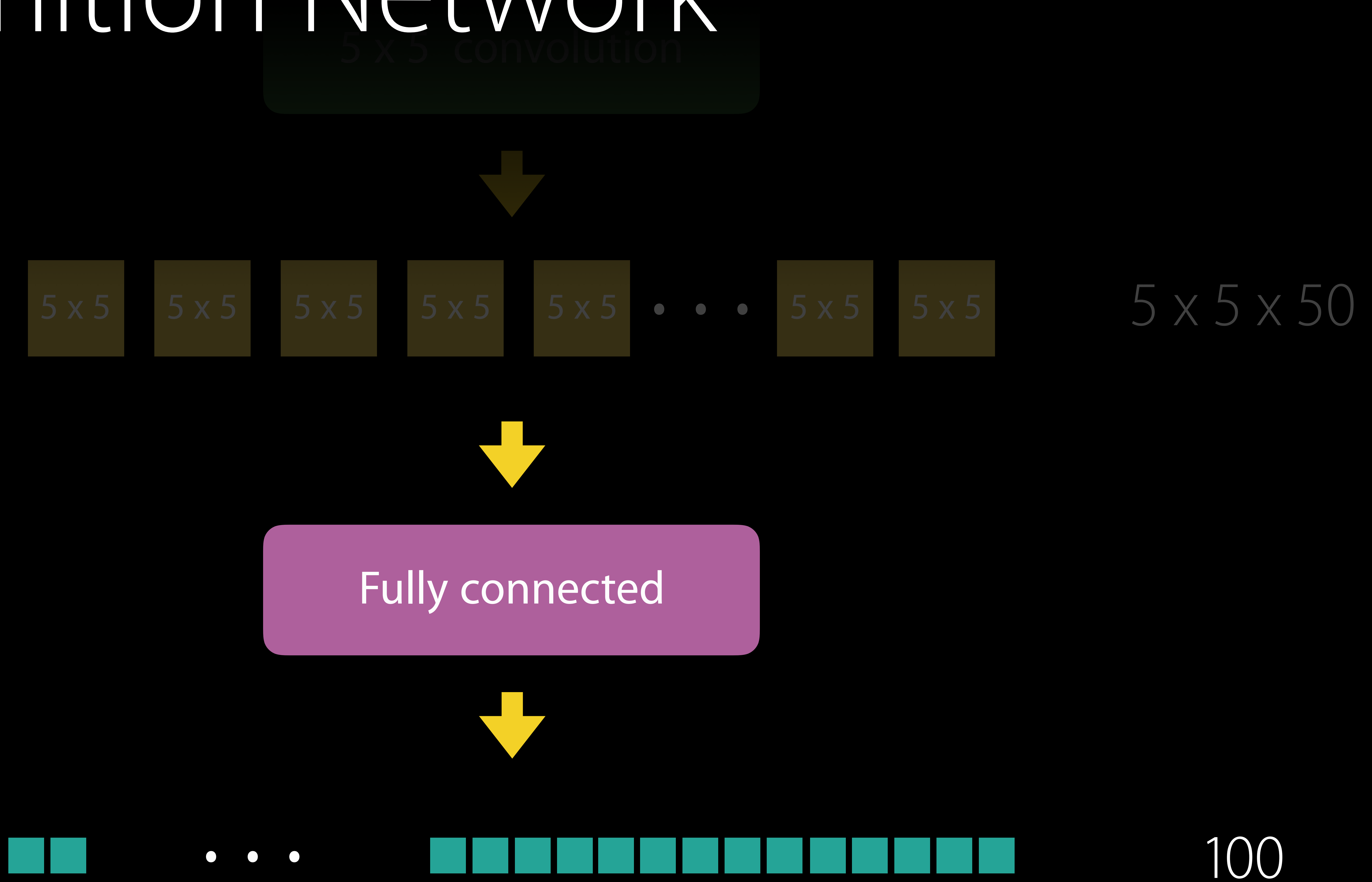


Digit Recognition Network

Example

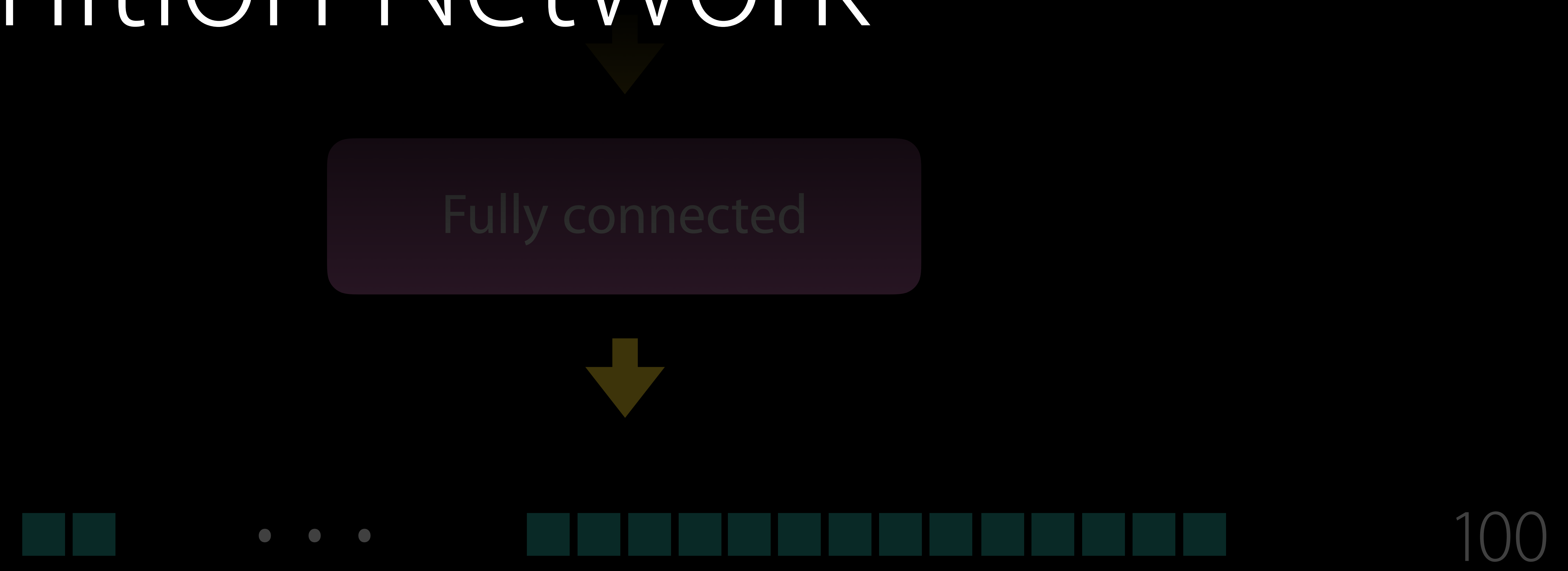
Digit Recognition Network

Example



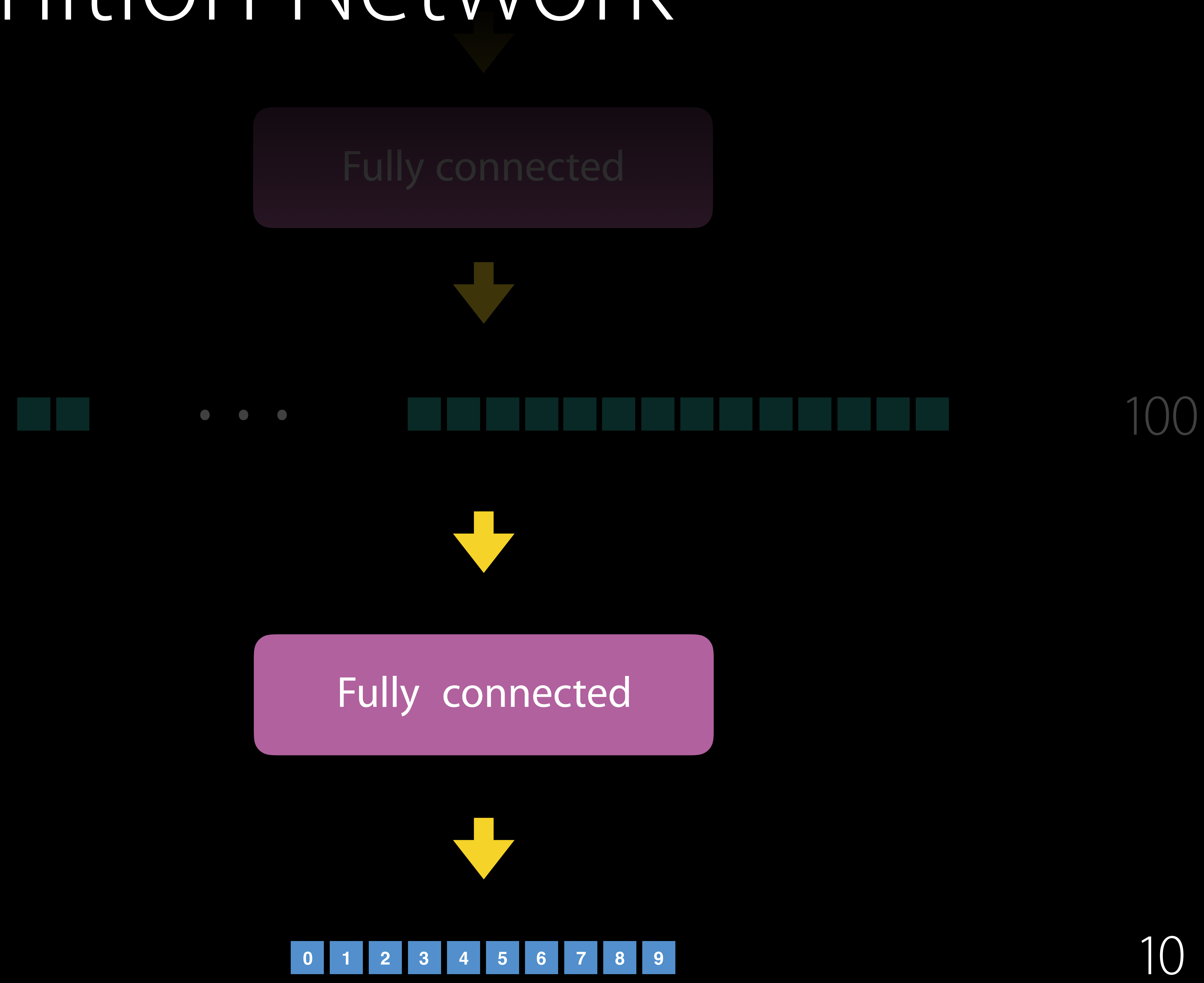
Digit Recognition Network

Example



Digit Recognition Network

Example

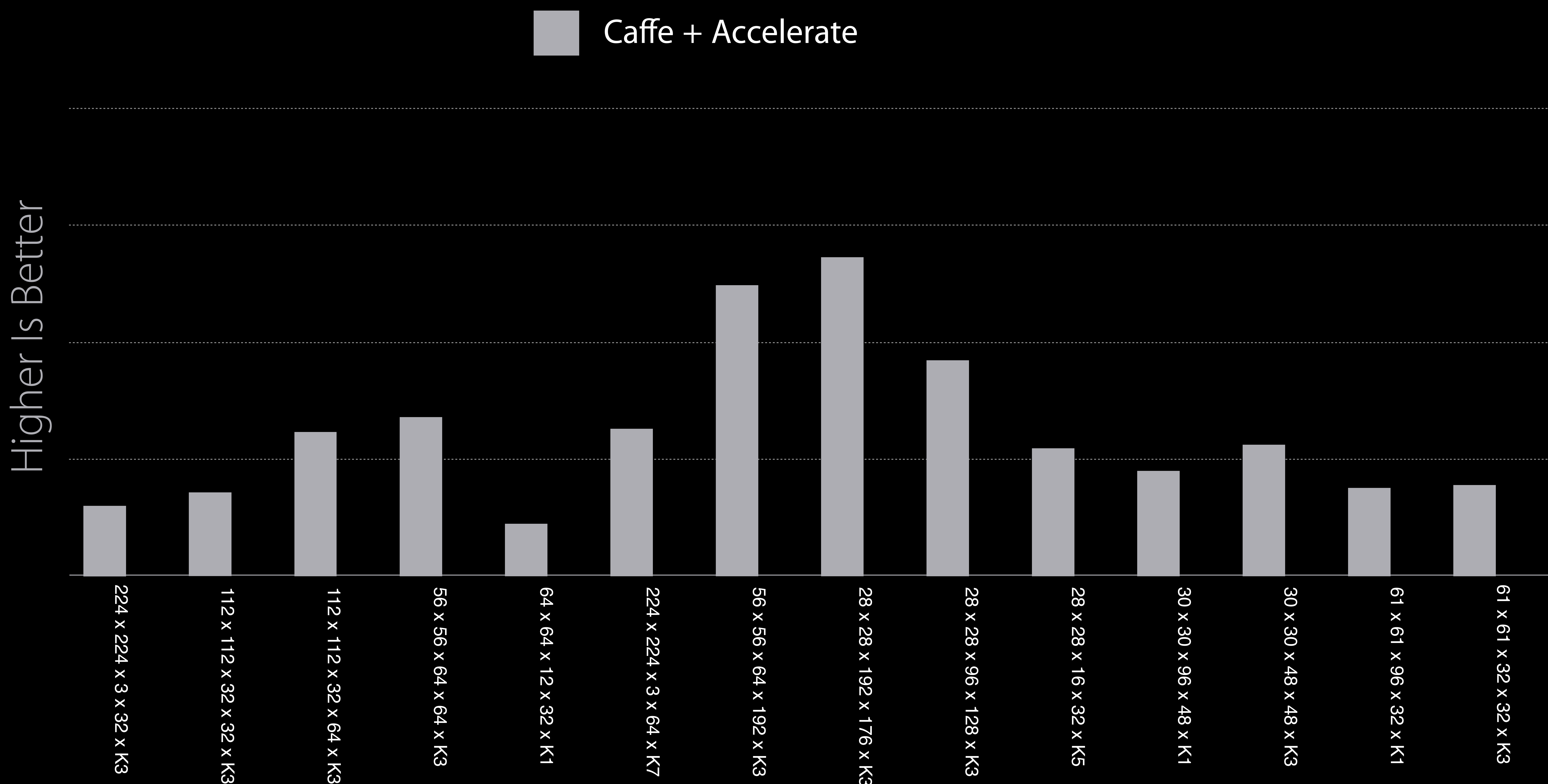


BNNS

Performance, four core Haswell MacBook Pro

BNNS

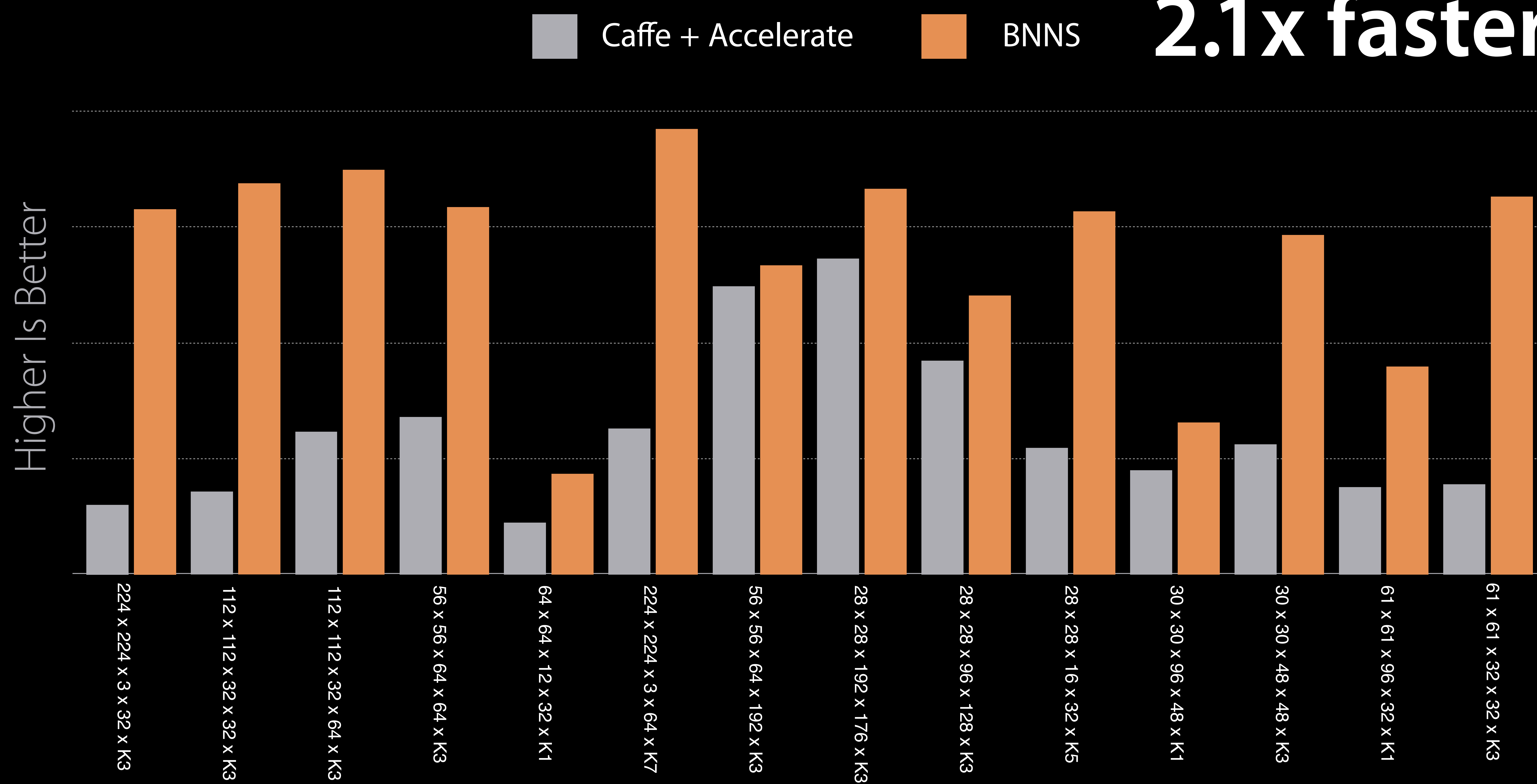
Performance, four core Haswell MacBook Pro



BNNS

Performance, four core Haswell MacBook Pro

2.1x faster



BNNS

Features

BNNS

Features

Low-level compute functions for CPU

BNNS

Features

Low-level compute functions for CPU

Inference only

BNNS
Features

Low-level compute functions for CPU

Inference only

Convolution layers

BNNS

Features

Low-level compute functions for CPU

Inference only

Convolution layers

Pooling layers

BNNS

Features

Low-level compute functions for CPU

Inference only

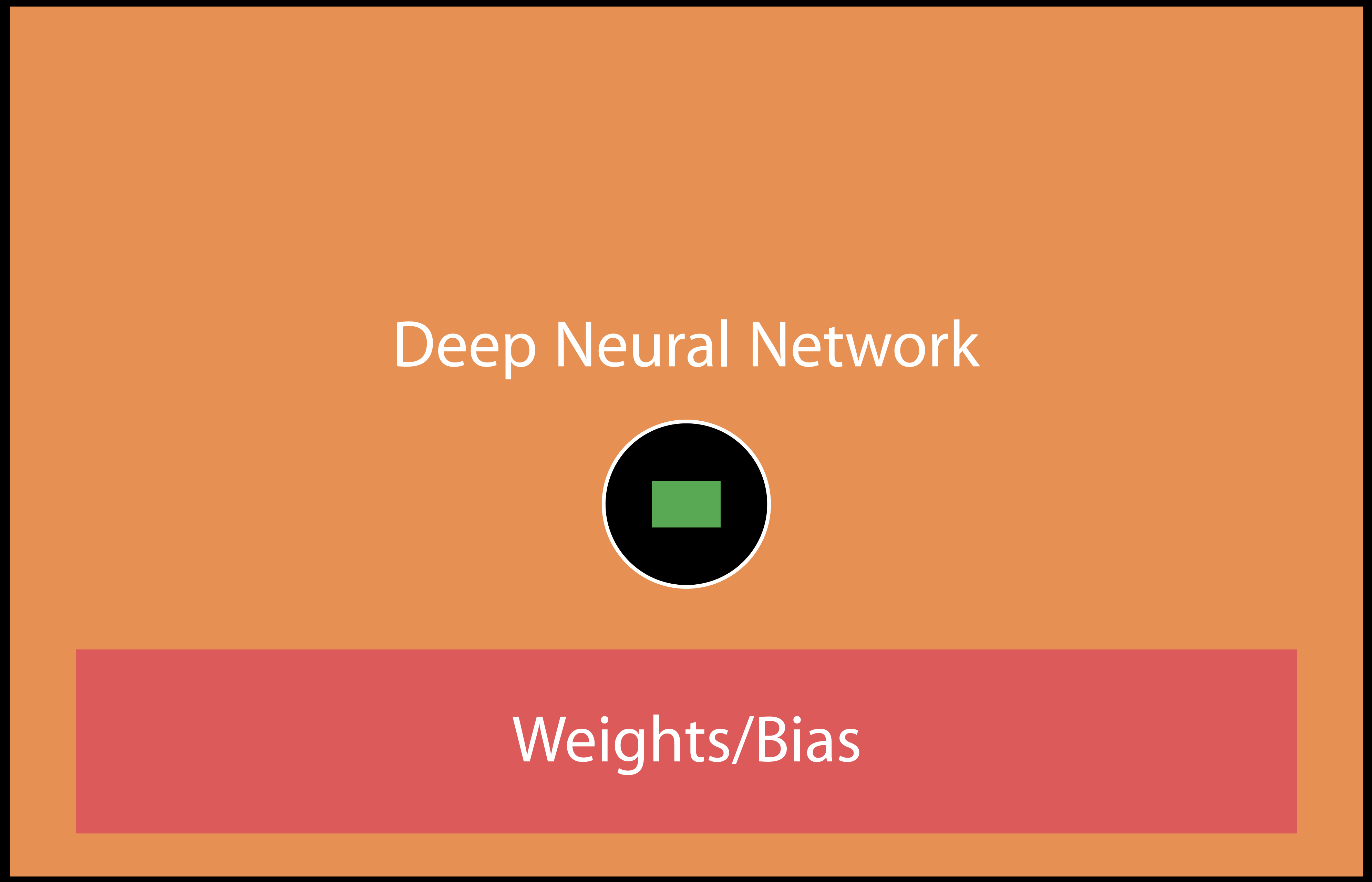
Convolution layers

Pooling layers

Fully connected layers

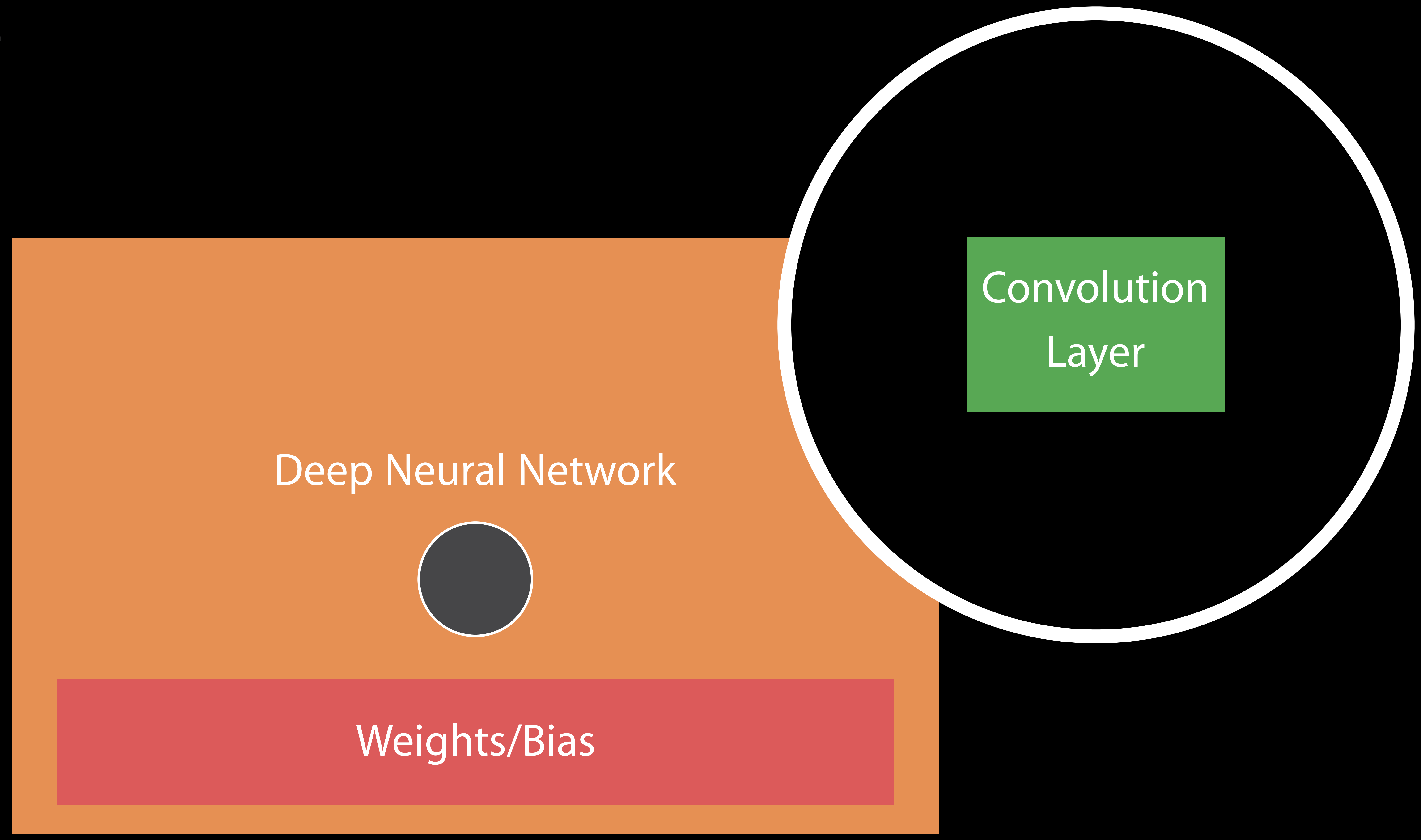
Deep Neural Network

Convolution layer



Deep Neural Network

Convolution layer



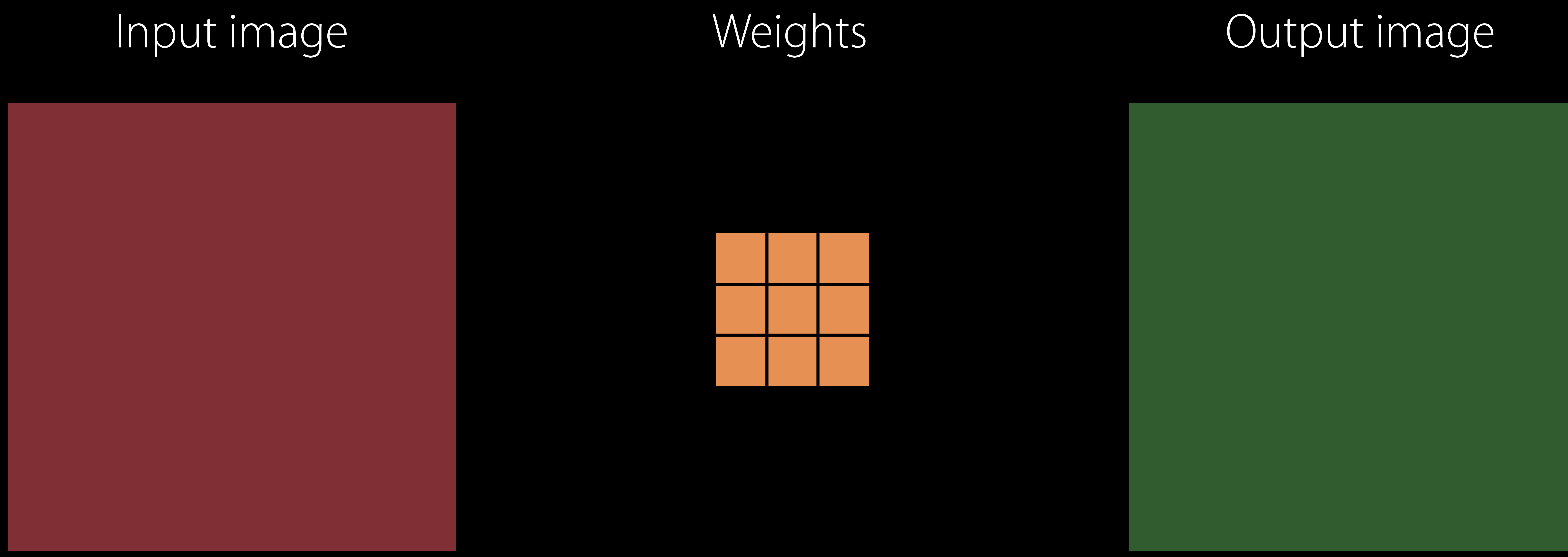
Convolution Layer

Input image

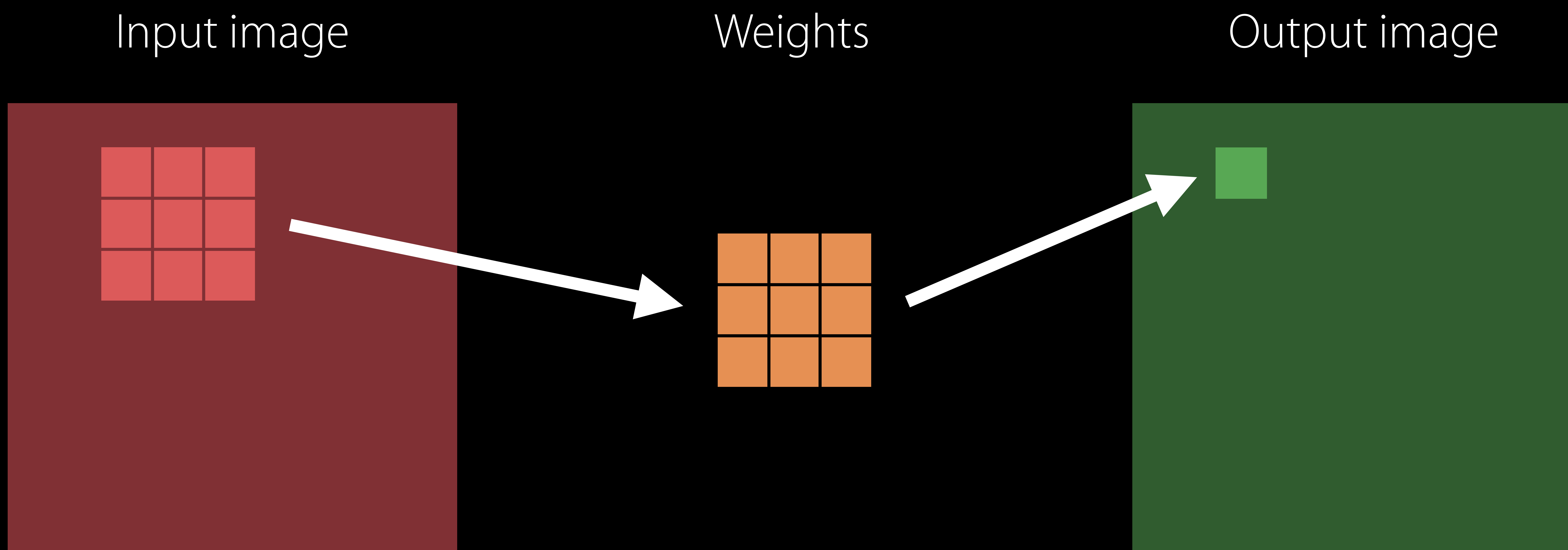
Weights

Output image

Convolution Layer

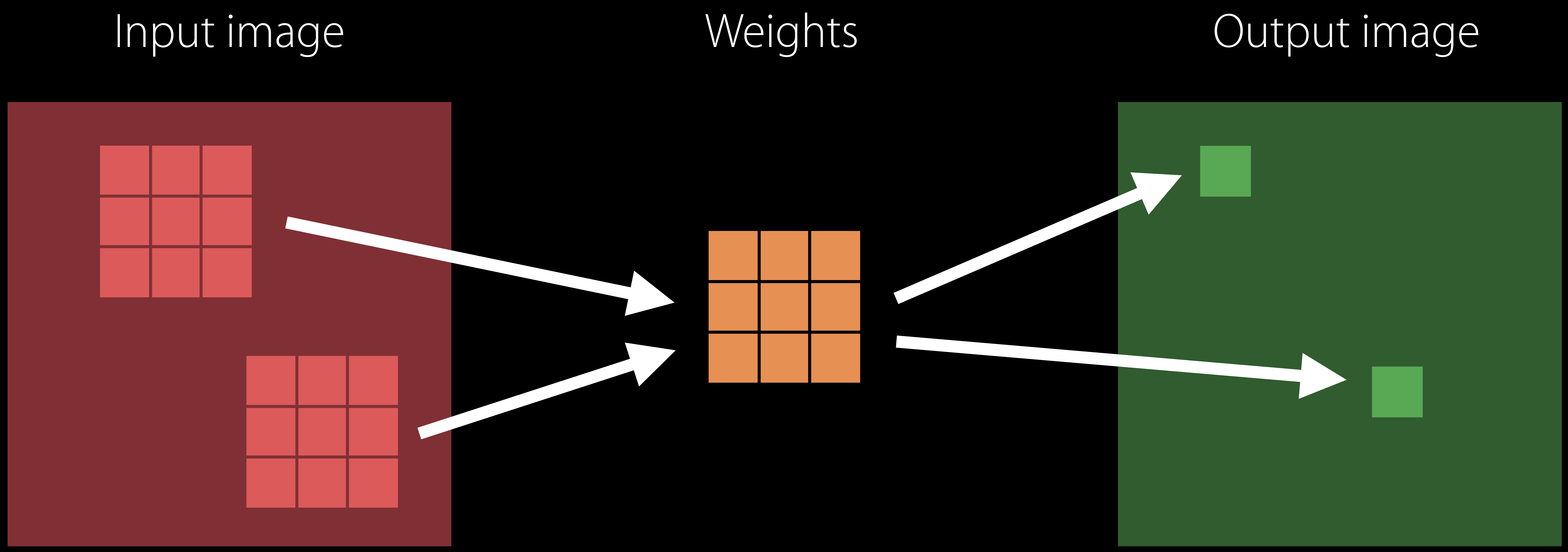


Convolution Layer



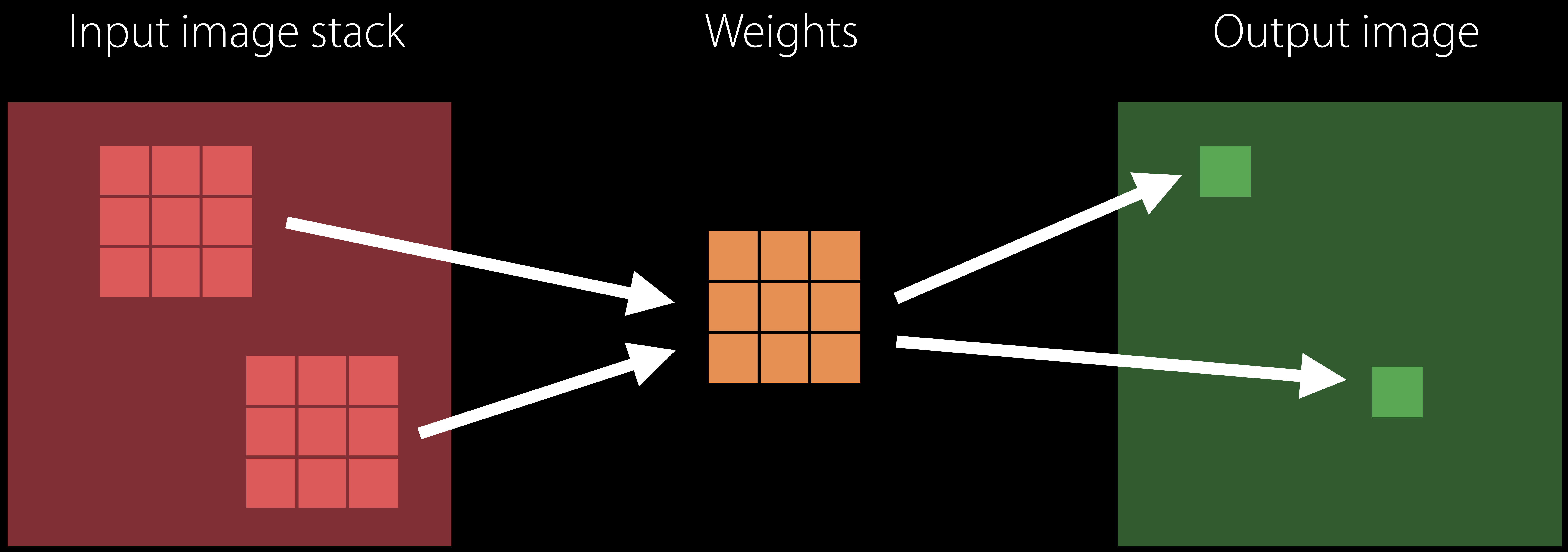
$$O(x, y) = \sum_{kx,ky} W(kx, ky) I(x + kx, y + ky)$$

Convolution Layer

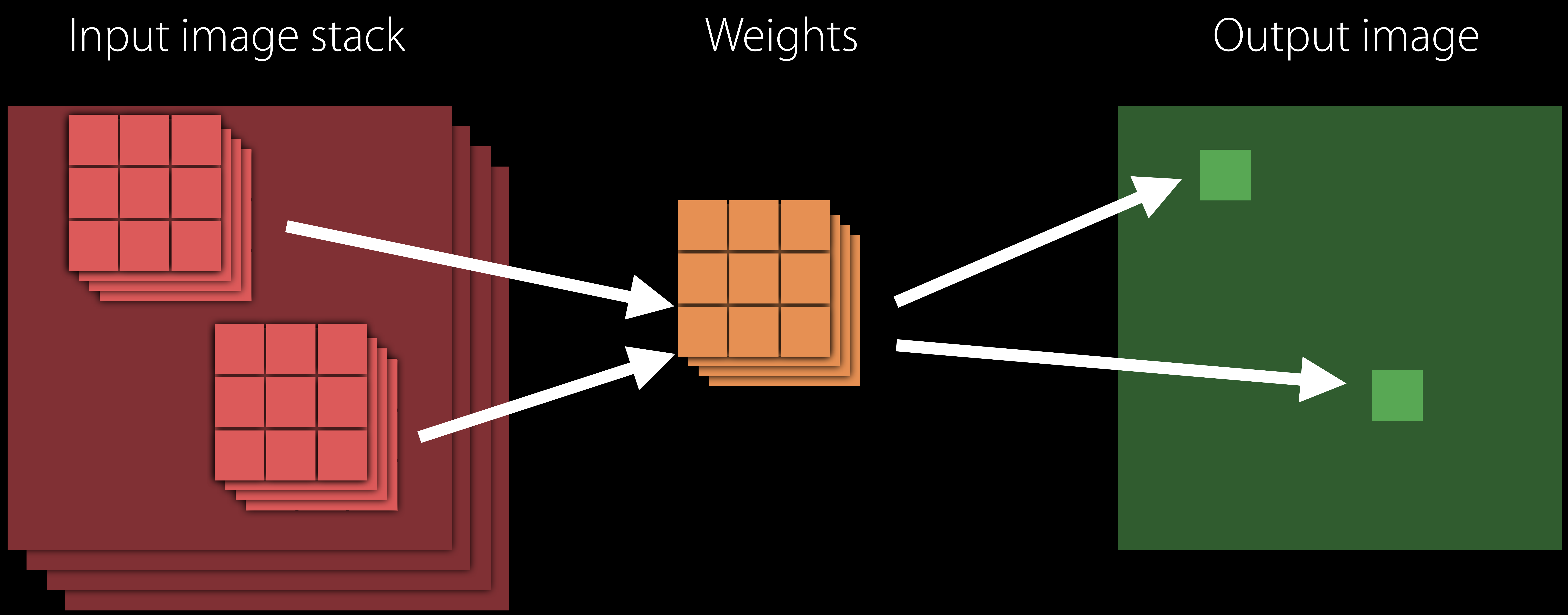


$$O(x,y) = \sum_{kx,ky} W(kx,ky)I(x+kx,y+ky)$$

Convolution Layer

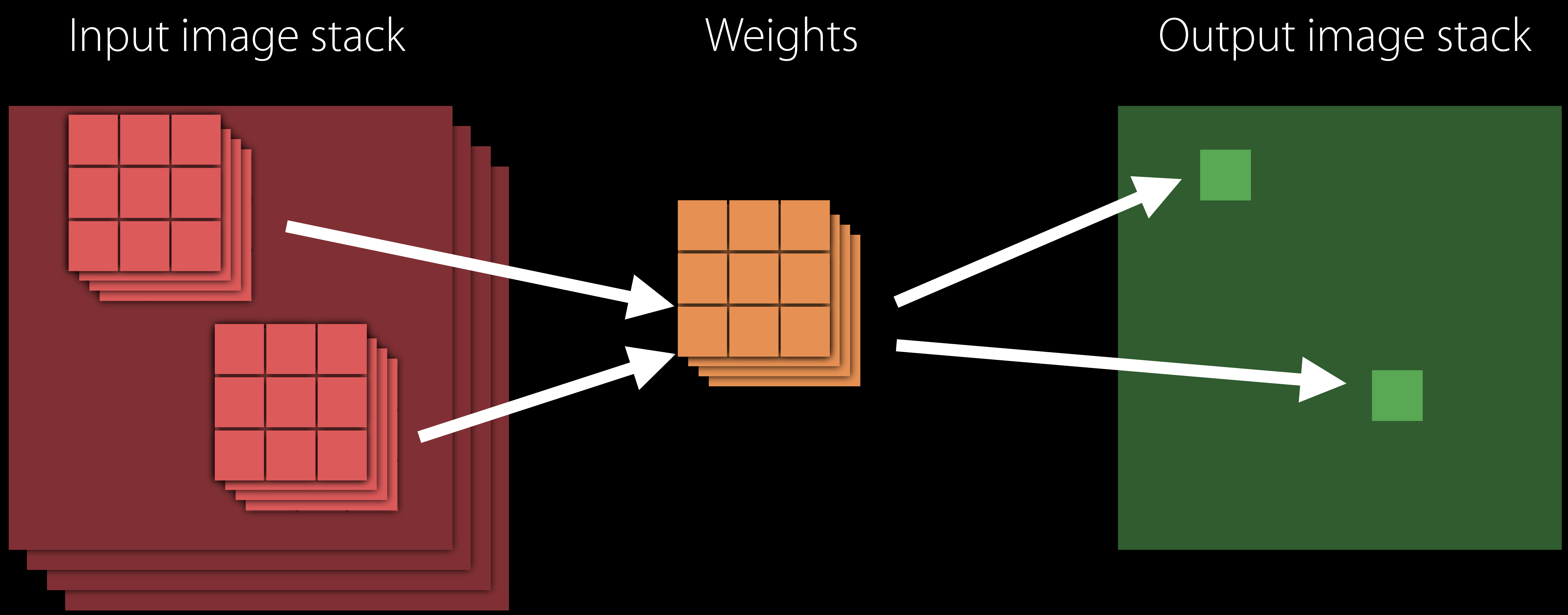


Convolution Layer



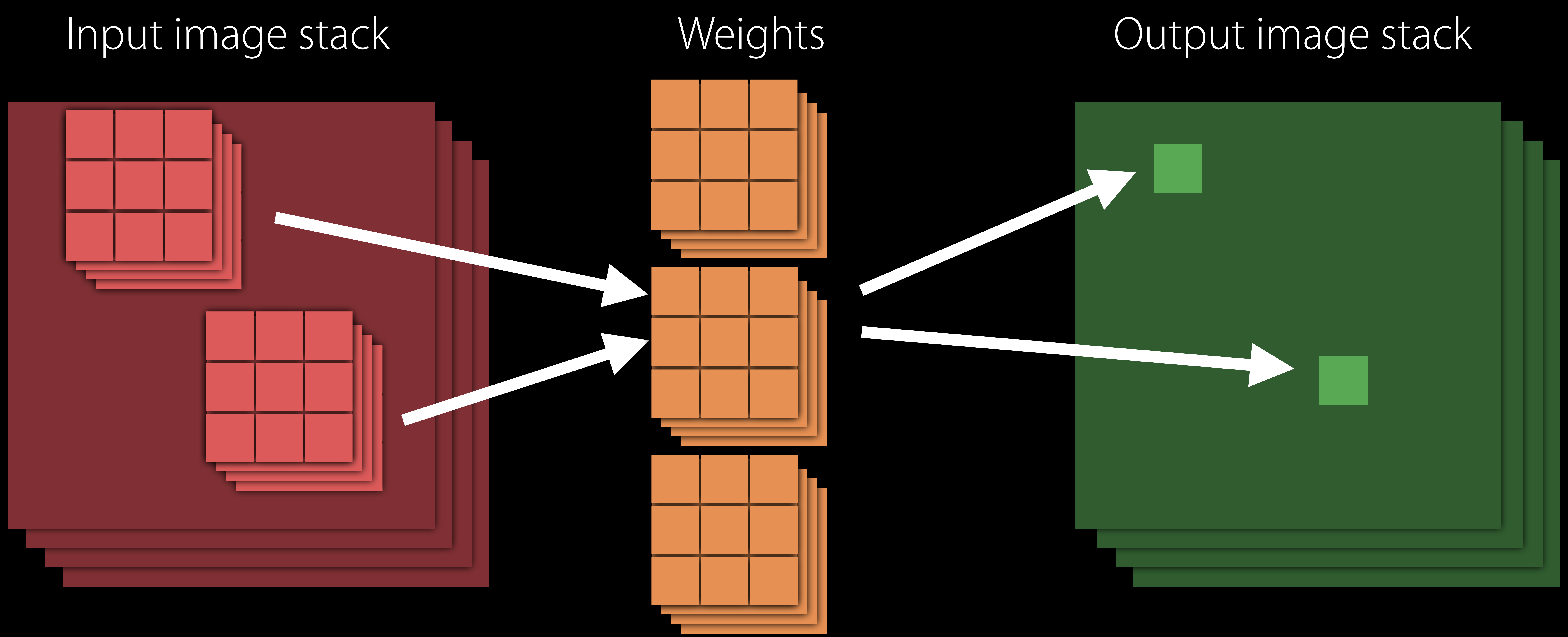
$$O(x,y) = \sum_{kx,ky,ic} W(kx,ky,ic)I(x+kx,y+ky,ic)$$

Convolution Layer



$$O(x,y) = \sum_{kx,ky,ic} W(kx,ky,ic)I(x+kx,y+ky,ic)$$

Convolution Layer



$$O(x, y, oc) = \sum_{kx, ky, ic} W(kx, ky, ic, oc) I(x + kx, y + ky, ic)$$

Convolution Layer

Example

Input image stack: 224 x 224 x 64

Output image stack: 222 x 222 x 96

Weights: 3 x 3 x 64 x 96

Floating point operations: 5.45 billion

All layers: 1-2 trillion

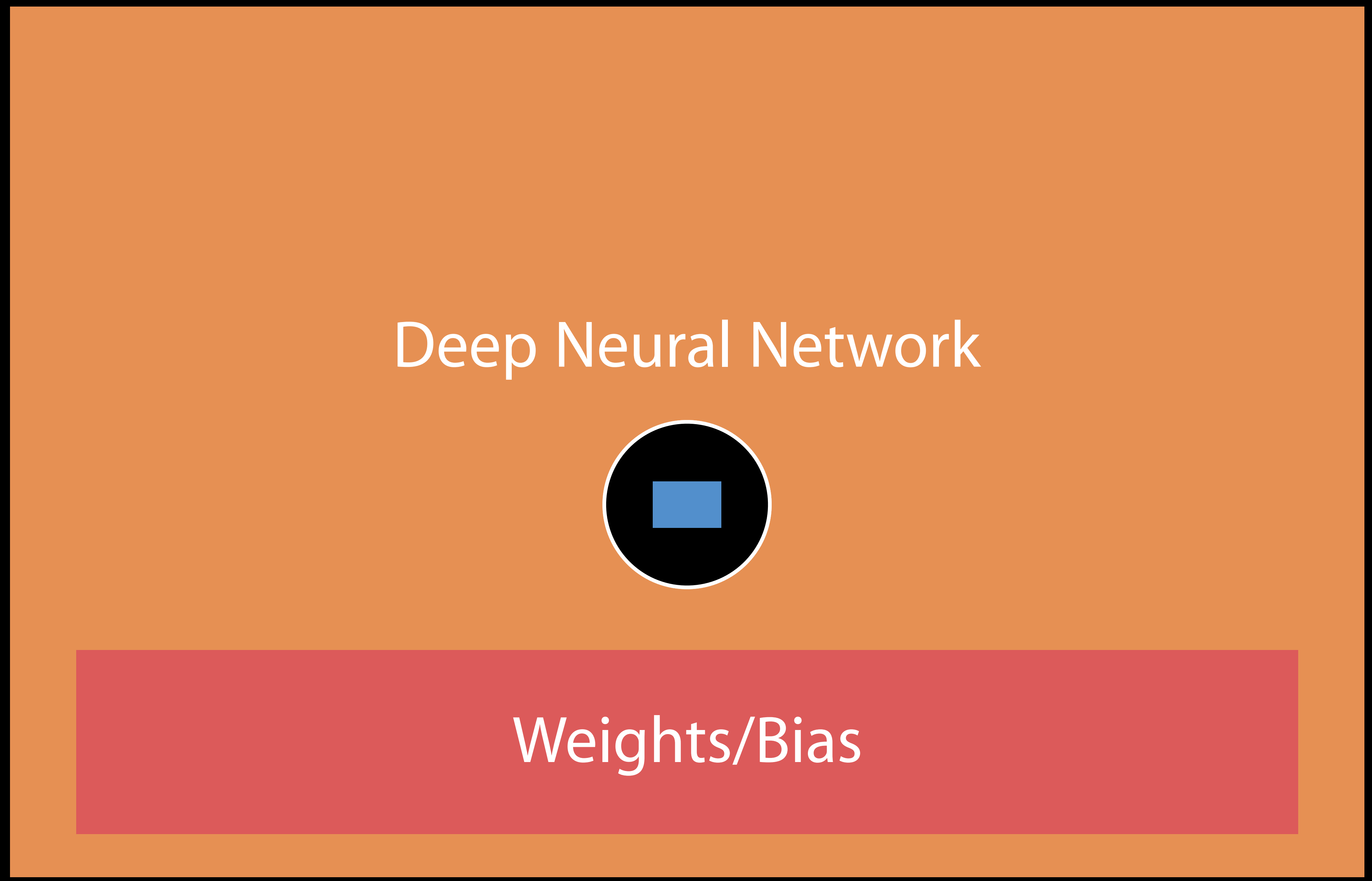
```c
#include <Accelerate/Accelerate.h>

// Describe input stack
BNNSImageStackDescriptor in_stack = {
    .width = 224,                       // width
    .height = 224,                      // height
    .channels = 64,                     // channels
    .row_stride = 224,                  // increment to next row (pix)
    .image_stride = 224*224,            // increment to next channel (pix)
    .data_type = BNNSDataTypeFloat32    // storage type
};
```

```c
#include <Accelerate/Accelerate.h>

// Describe convolution layer
BNNSConvolutionLayerParameters conv = {
    .k_width = 3,                                   // kernel height
    .k_height = 3,                                  // kernel width
    .x_padding = 0,                                 // X padding
    .y_padding = 0,                                 // Y padding
    .x_stride = 1,                                  // X stride
    .y_stride = 1,                                  // Y stride
    .in_channels = 64,                              // input channels
    .out_channels = 96,                             // output channels
    .weights = {
        .data_type = BNNSDataTypeFloat16,           // weights storage type
        .data = weights                             // pointer to weights data
    }
};
```

```c
#include <Accelerate/Accelerate.h>

// Create convolution layer filter
BNNSFilter filter = BNNSFilterCreateConvolutionLayer(
    &in_stack,     // BNNSImageStackDescriptor for input stack
    &out_stack,    // BNNSImageStackDescriptor for output stack
    &conv,         // BNNSConvolutionLayerParameters
    NULL);         // BNNSFilterParameters (NULL = defaults)


// Use the filter ...


// Destroy filter
BNNSFilterDestroy(filter);
```

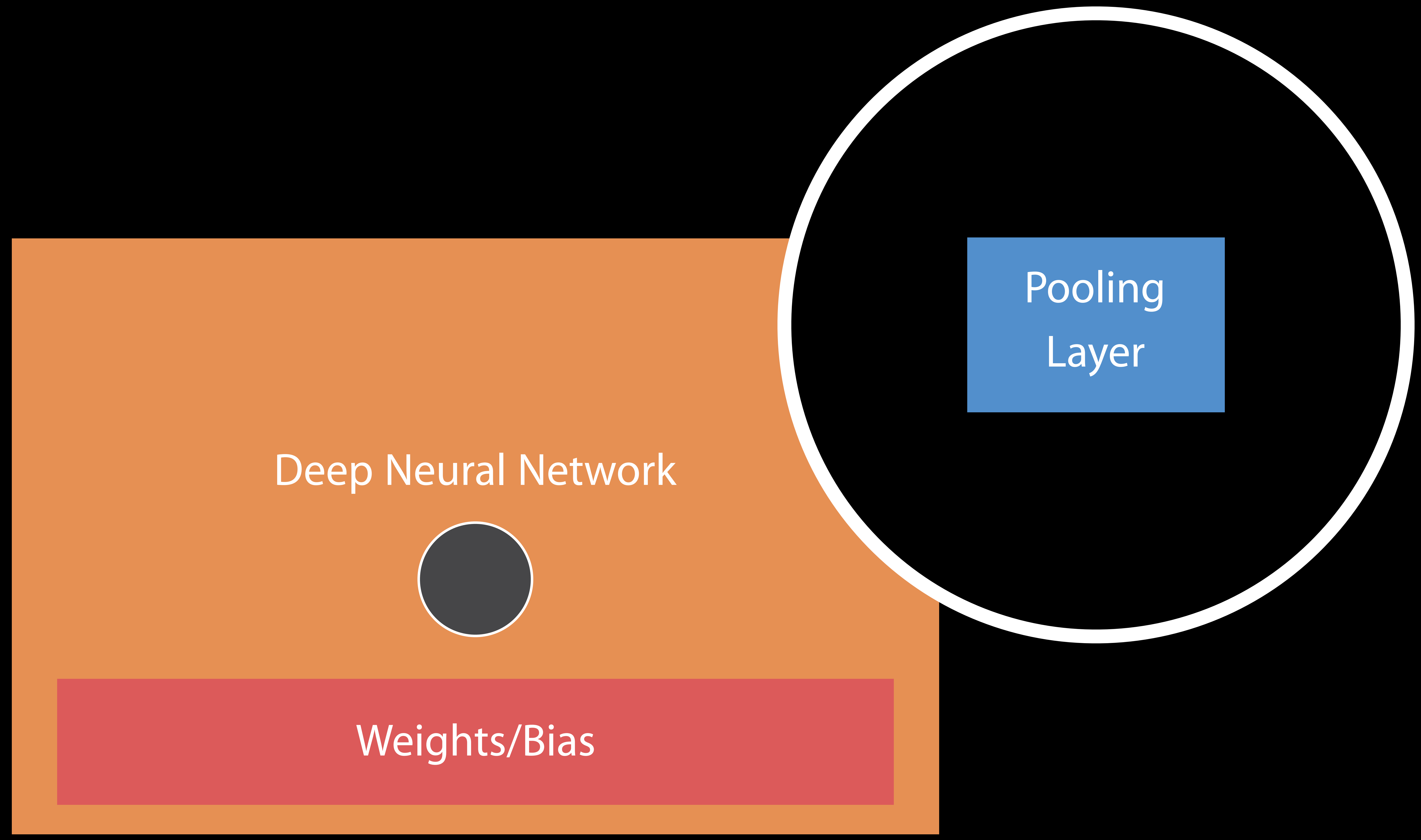
Deep Neural Network

Pooling layer

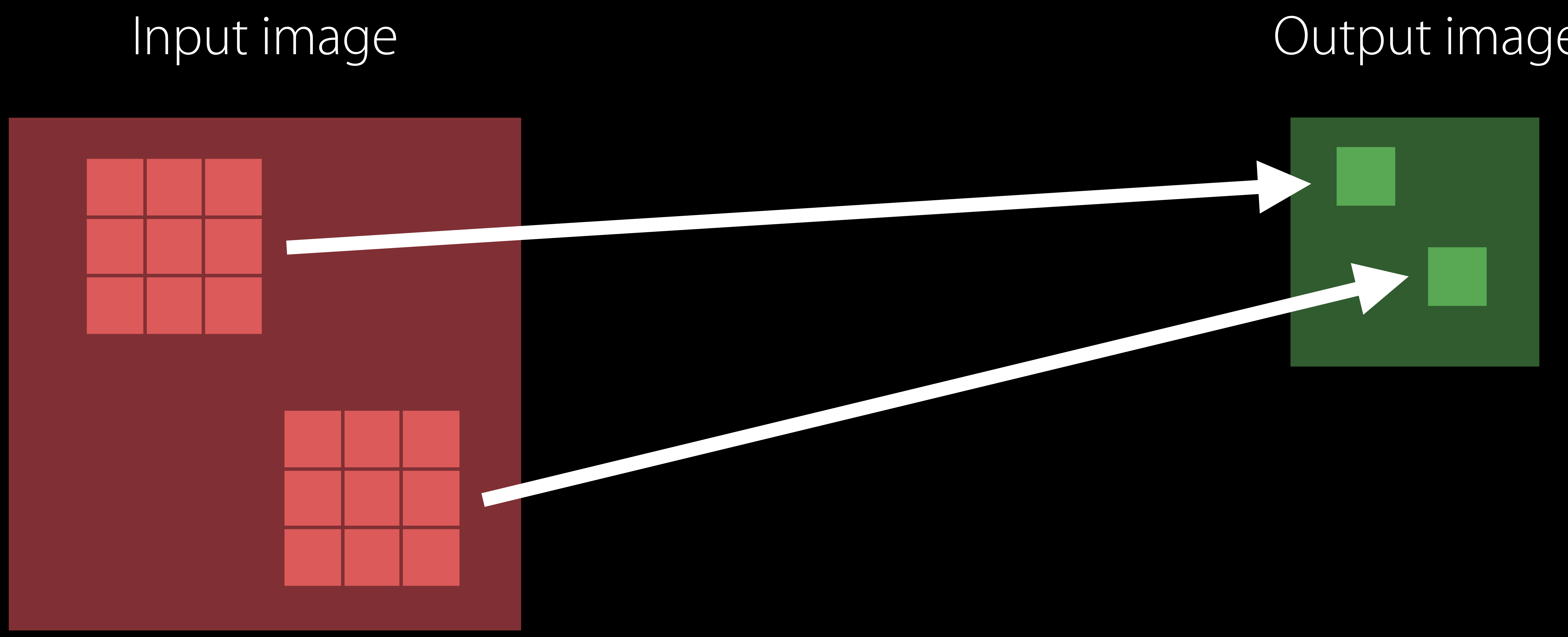


Deep Neural Network

Pooling layer



Pooling Layer



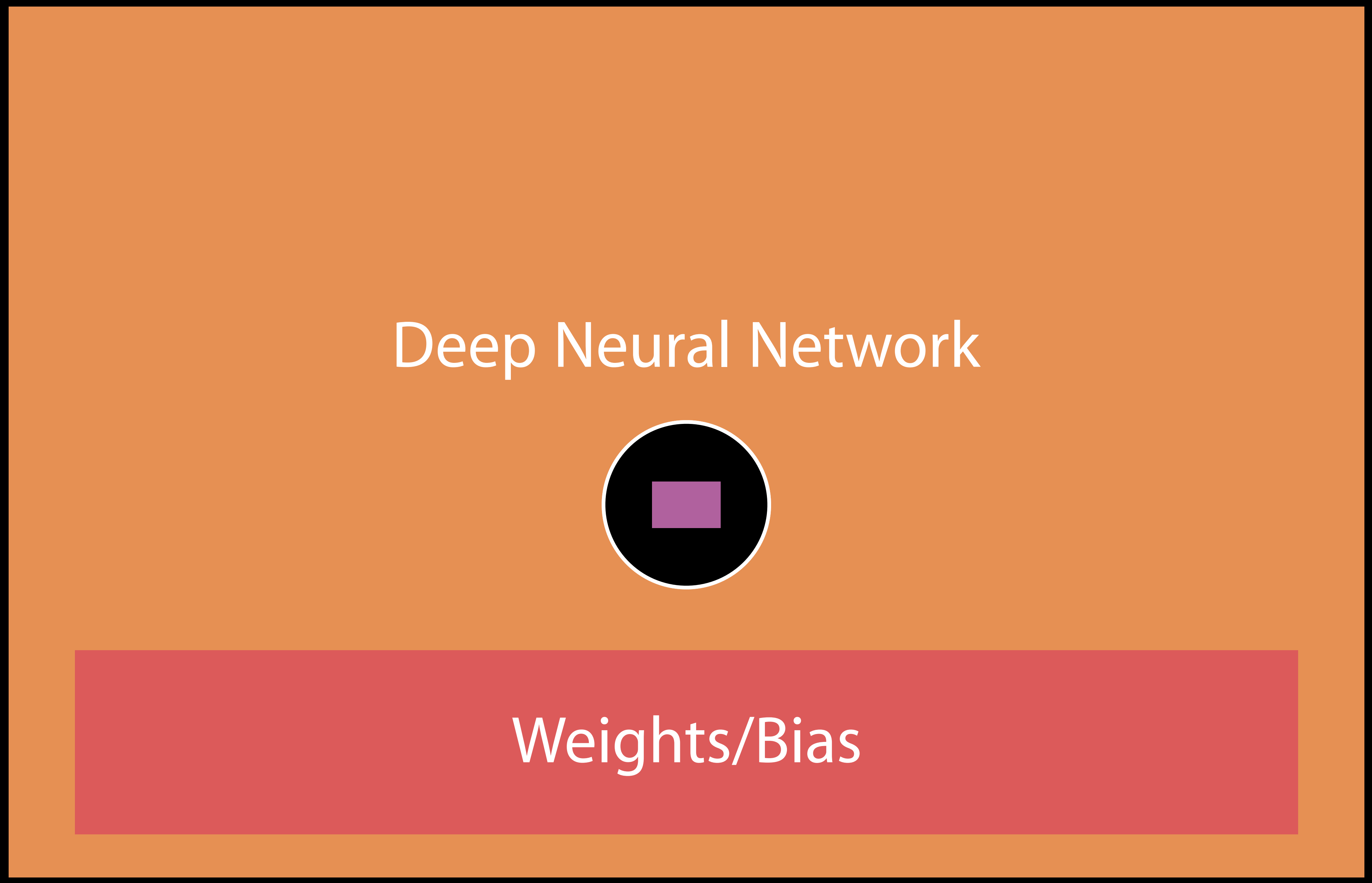
$$O(x, y, c) = \max_{i,j \leq k} I(s_x \cdot x + i, s_y \cdot y + j, c)$$

```c
#include <Accelerate/Accelerate.h>

// Describe pooling layer
BNNSPoolingLayerParameters pool = {
    .k_width = 3,                                   // kernel height
    .k_height = 3,                                  // kernel width
    .x_padding = 1,                                 // X padding
    .y_padding = 1,                                 // Y padding
    .x_stride = 2,                                  // X stride
    .y_stride = 2,                                  // Y stride
    .in_channels = 64,                              // input channels
    .out_channels = 64,                             // output channels
    .pooling_function = BNNSPoolingFunctionMax  // pooling function
};
```

```
#include <Accelerate/Accelerate.h>

// Create pooling layer filter
BNNSFilter filter = BNNSFilterCreatePoolingLayer(
    &in_stack,     // BNNSImageStackDescriptor for input stack
    &out_stack,    // BNNSImageStackDescriptor for output stack
    &pool,         // BNNSPoolingLayerParameters
    NULL);         // BNNSFilterParameters (NULL = defaults)



// Use the filter ...



// Destroy filter
BNNSFilterDestroy(filter);
```

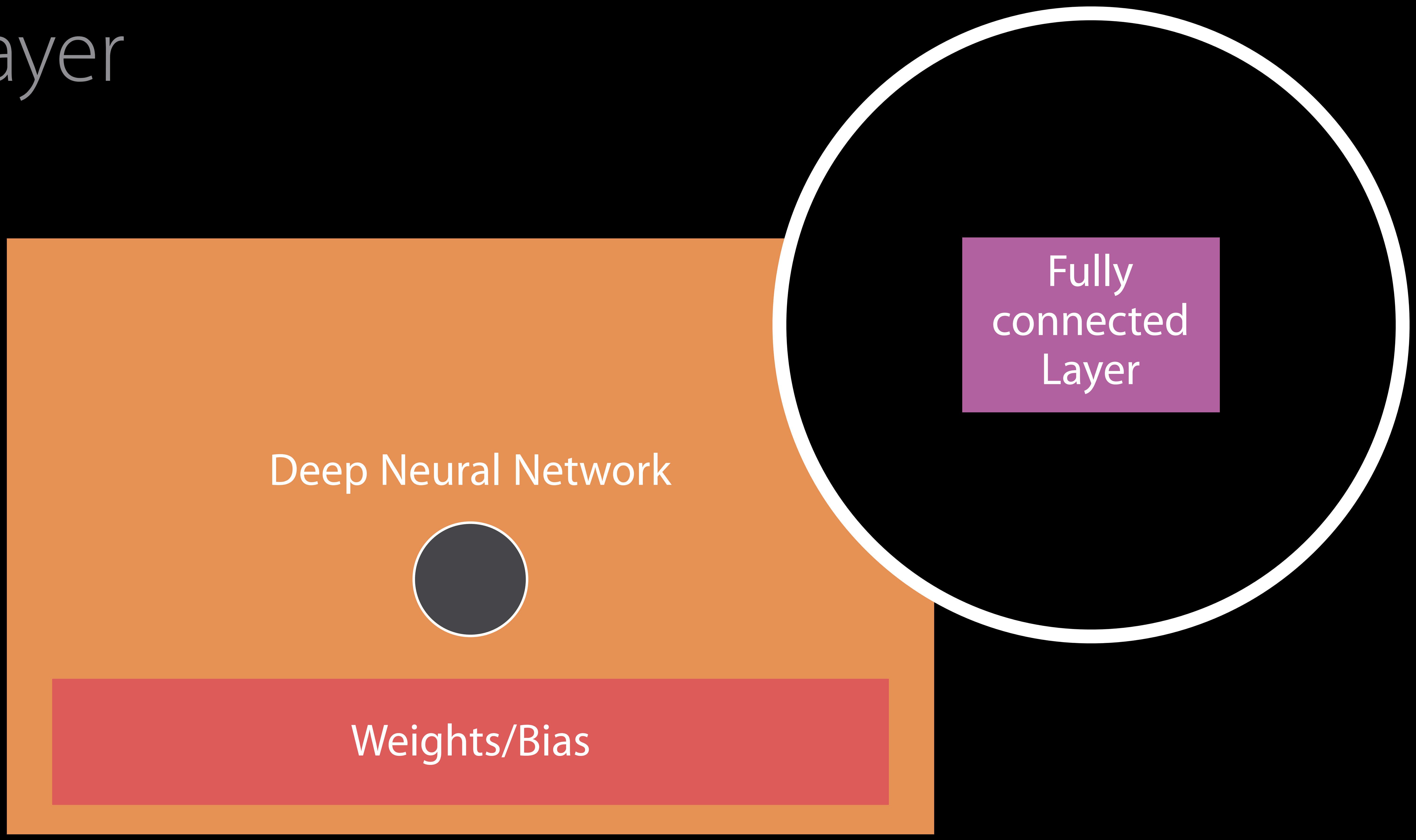
Deep Neural Network

Fully connected layer



Deep Neural Network

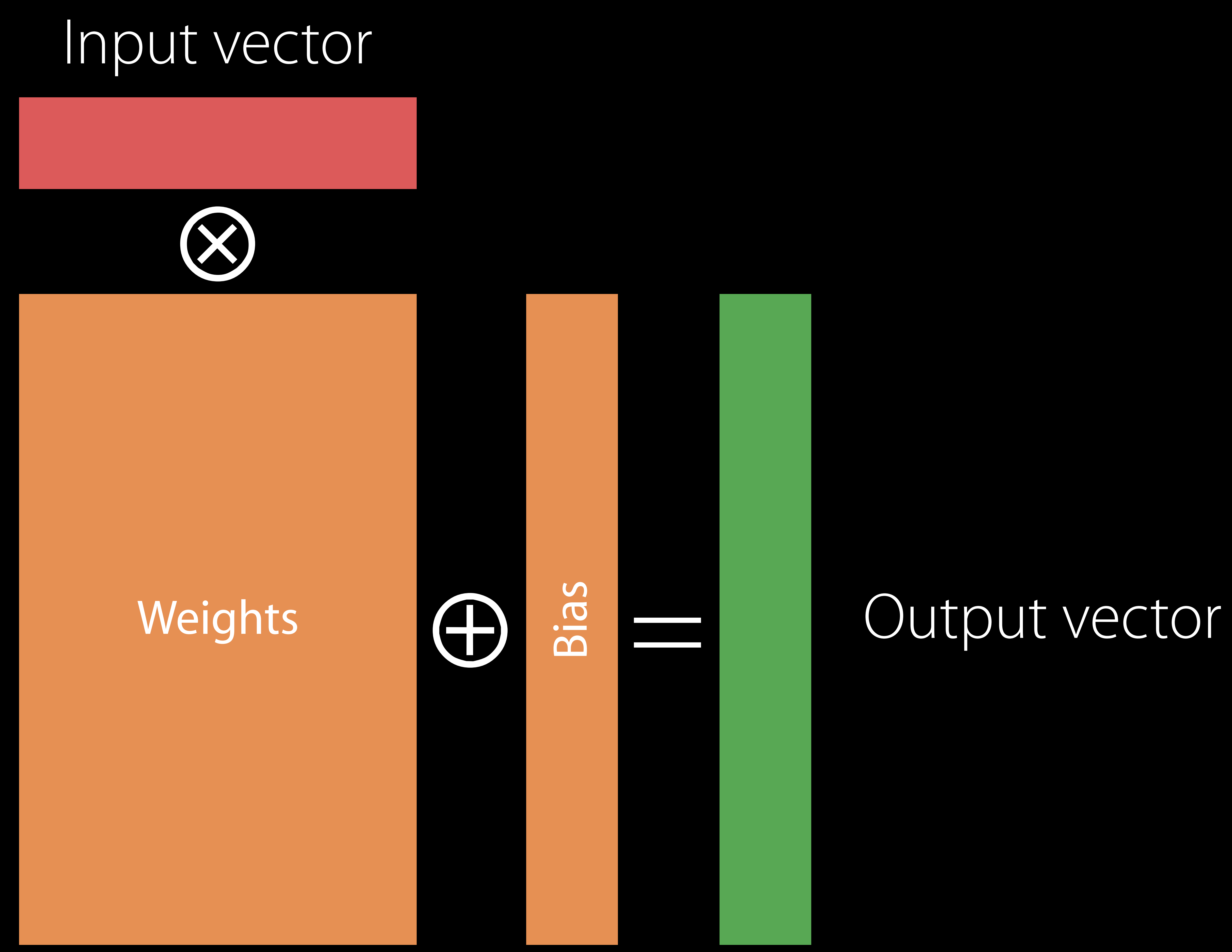
Fully connected layer



Fully Connected Layer

Input vector

Fully Connected Layer



$$O(i) = \sum_j W(i,j)I(j) + B(i)$$

```c
#include <Accelerate/Accelerate.h>

// Describe input vector
BNNSVectorDescriptor in_vec = {
    .size = 3000,                       // size
    .data_type = BNNSDataTypeFloat32    // storage type
};
```

```c
#include <Accelerate/Accelerate.h>

// Describe fully connected layer
BNNSFullyConnectedLayerParameters full = {
    .in_size = 3000,                        // input vector size
    .out_size = 20000,                      // output vector size
    .weights = {
        .data_type = BNNSDataTypeFloat16,   // weights storage type
        .data = weights                     // pointer to weights data
    }
};
```

```c
#include <Accelerate/Accelerate.h>

// Create fully connected layer filter
BNNSFilter filter = BNNSFilterCreateFullyConnectedLayer(
    &in_vec,       // BNNSVectorDescriptor for input vector
    &out_vec,      // BNNSVectorDescriptor for output vector
    &full,         // BNNSFullyConnectedLayerParameters
    NULL);         // BNNSFilterParameters (NULL = defaults)


// Use the filter ...


// Destroy filter
BNNSFilterDestroy(filter);
```

```c
#include <Accelerate/Accelerate.h>

// Apply filter to one pair of (in,out)
int status = BNNSFilterApply(filter,        // BNNSFilter
                             in,            // pointer to input data
                             out);          // pointer to output data
```

```c
#include <Accelerate/Accelerate.h>

// Apply filter to N pairs of (in,out)
int status = BNNSFilterApplyBatch(filter,        // BNNSFilter
                                  20,            // batch size (N)
                                  in,            // pointer to input data
                                  3000,          // input stride (values)
                                  out,           // pointer to output data
                                  20000);        // output stride (values)
```

BNNS

BNNS

Low-level compute functions for neural networks

BNNS

Low-level compute functions for neural networks

Fast and energy-efficient inference

BNNS

Low-level compute functions for neural networks

Fast and energy-efficient inference

Multiple storage types

Quadrature

Numerical integration

Quadrature

Numerical integration

NEW

$$\int_a^b f(x)\,\mathrm{d}x$$

```c
#include <Accelerate/Accelerate.h>      // Quadrature is part of Accelerate

// Describe the function to integrate
quadrature_integrate_function fun = {
    .fun = f,                               // evaluation callback
};

// Evaluates the function at n points x[i] -> y[i]
void f(void *arg, size_t n, const double *x, double *y)
{
    for (size_t i=0; i<n; i++) {
        y[i] = 1.0 / (1.0 + x[i] * x[i]);
    }
}
```

```c
#include <Accelerate/Accelerate.h>      // Quadrature is part of Accelerate

// Describe the integration method and parameters
quadrature_integrate_options opt = {
    .integrator = QUADRATURE_INTEGRATE_QAG,       // integration algorithm
    .abs_tolerance = 1.0e-8,                      // requested tolerance
    .max_intervals = 12                           // max number of intervals for QAG
};

// QNG   simple non-adaptive integrator
// QAG   simple globally adaptive integrator
// QAGS  globally adaptive integrator with convergence acceleration
```

```c
#include <Accelerate/Accelerate.h>

// Compute the integral
quadrature_status status;
double est_error;
double result = quadrature_integrate(
    &fun,         // quadrature_integrate_function, function to integrate
    -1.0,         // a, first bound of interval
    2.0,          // b, second bound of interval
    &opt,         // quadrature_integrate_options, integration method and options
    &status,      // quadrature_status, receives success/failure code
    &est_error,   // double, receives the estimated absolute error
    0, NULL);     // optional args
```

simd

Vector and geometry operations

Steve Canon Core OS, Vector and Numerics Group

simd

simd

Geometric operations on vectors and matrices for C, Objective-C, C++, and Swift

simd

Geometric operations on vectors and matrices for C, Objective-C, C++, and Swift

Closely mirrors Metal shading language

simd

Types

simd

Types

Vectors of floats, doubles, signed and unsigned integers of length 2, 3, and 4

simd

Types

Vectors of floats, doubles, signed and unsigned integers of length 2, 3, and 4

Matrices of floats and doubles, of size NxM, where N and M are 2, 3, or 4

simd

Operations

simd

Operations

Arithmetic operators on vectors and matrices

simd

Operations

Arithmetic operators on vectors and matrices

Geometry and shader functions

```
// myCode.m:
@import simd;

vector_float3 reflect(vector_float3 x, vector_float3 n) {
    return x - 2*vector_dot(x,n)*n;
}

// myCode.cpp:
#include <simd/simd.h>
using namespace simd;

float3 reflect(float3 x, float3 n) {
    return x - 2*dot(x,n)*n;
}

// myCode.swift:
import simd

func reflect(x: float3, n: float3) -> float3 {
    return x - 2*dot(x,n)*n
}
```

```
// myCode.m:
@import simd;

vector_float3 reflect(vector_float3 x, vector_float3 n) {
    return x - 2*vector_dot(x,n)*n;
}

// myCode.cpp:
#include <simd/simd.h>
using namespace simd;

float3 reflect(float3 x, float3 n) {
    return x - 2*dot(x,n)*n;
}

// myCode.swift:
import simd

func reflect(x: float3, n: float3) -> float3 {
    return x - 2*dot(x,n)*n
}
```

simd

Interoperation between languages

simd

Interoperation between languages

Vector types are compiler extensions in C, Objective-C, and C++

simd

Interoperation between languages

Vector types are compiler extensions in C, Objective-C, and C++

Swift vector types are structs

simd

Interoperation between languages

Vector types are compiler extensions in C, Objective-C, and C++

Swift vector types are structs

The compiler maps between corresponding vector types for you

```
// myHeader.h:
@import simd;

vector_float3 someFunction(vector_float3 x, vector_float3 y);



// myCode.swift:
import simd

let x = float3(1,2,3)
let y = float3(0,0,1)
// Vector types are bridged automatically.
let z = someFunction(x, y)
```

```
// myHeader.h:
@import simd;

vector_float3 someFunction(vector_float3 x, vector_float3 y);
```

```
// myCode.swift:
import simd

let x = float3(1,2,3)
let y = float3(0,0,1)
// Vector types are bridged automatically.
let z = someFunction(x, y)
```

```
// myHeader.h:
@import simd;

vector_float3 someFunction(vector_float3 x, vector_float3 y);



// myCode.swift:
import simd

let x = float3(1,2,3)
let y = float3(0,0,1)
// Vector types are bridged automatically.
let z = someFunction(x, y)
```

simd

Interoperation between languages

simd

Interoperation between languages

Swift matrix types are layout-compatible with C matrix types

```swift
import simd

//  Use initializer to convert C matrix to Swift matrix.
let mat = float4x4(CFunctionReturningMatrix())

//  Use cmatrix property to convert Swift matrix to C matrix.
let result = CFunctionConsumingMatrix(mat.cmatrix)
```

```swift
import simd

//  Use initializer to convert C matrix to Swift matrix.
let mat = float4x4(CFunctionReturningMatrix())

//  Use cmatrix property to convert Swift matrix to C matrix.
let result = CFunctionConsumingMatrix(mat.cmatrix)
```

```swift
import simd

//  Use initializer to convert C matrix to Swift matrix.
let mat = float4x4(CFunctionReturningMatrix())

//  Use cmatrix property to convert Swift matrix to C matrix.
let result = CFunctionConsumingMatrix(mat.cmatrix)
```

New Geometry Functions

NEW

```
simd_orient(x, y, …)
simd_incircle(x, a, b, c)
simd_insphere(x, a, b, c, d)
```

orient

orient

Is a set of vectors *positively oriented*?

orient

Is a set of vectors *positively oriented*?

- Do they obey the *right hand rule*?

orient

Is a set of vectors *positively oriented*?

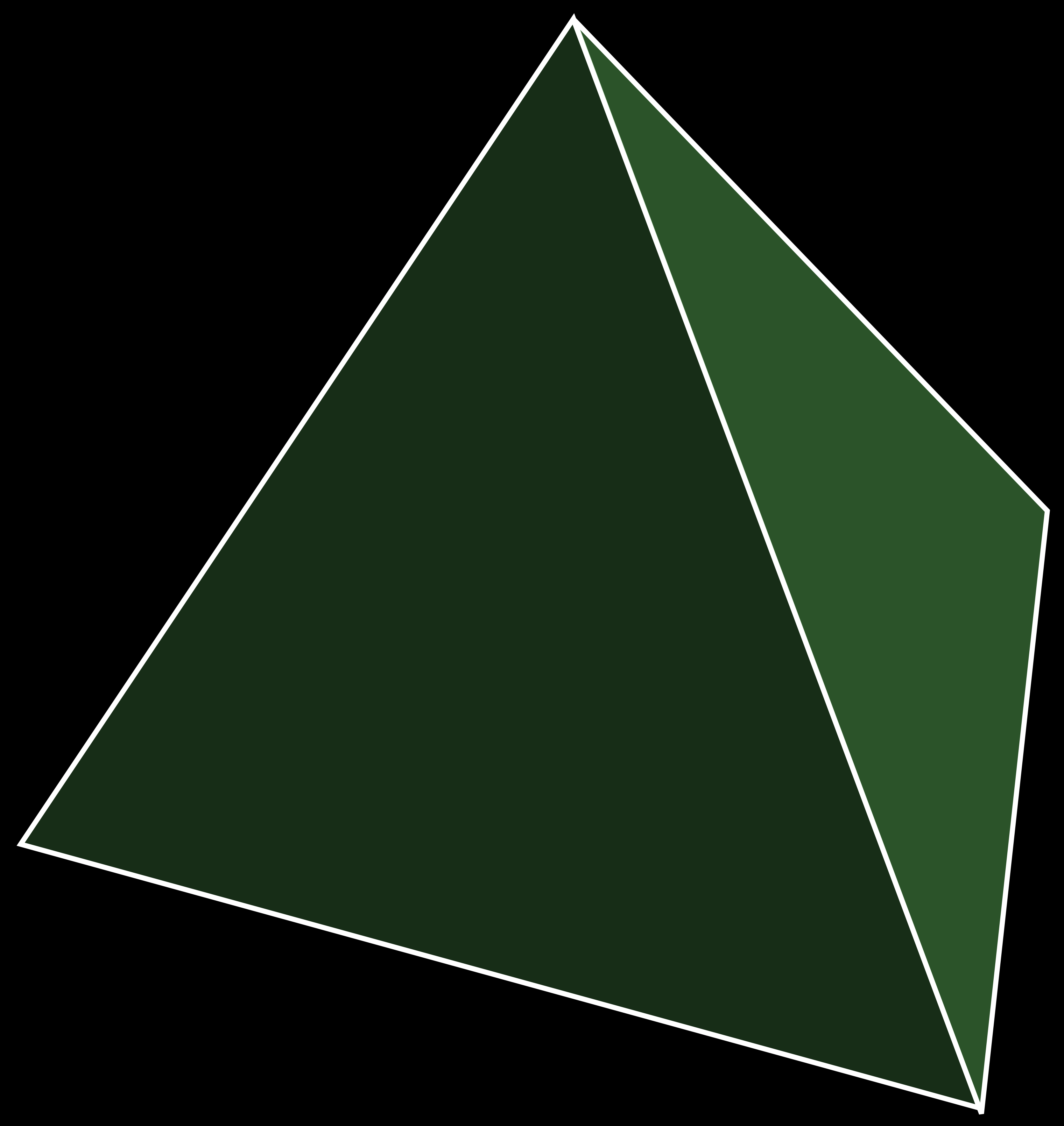
- Do they obey the *right hand rule*?
- Is their determinant positive?

orient

Is a triangle facing toward me or away from me?

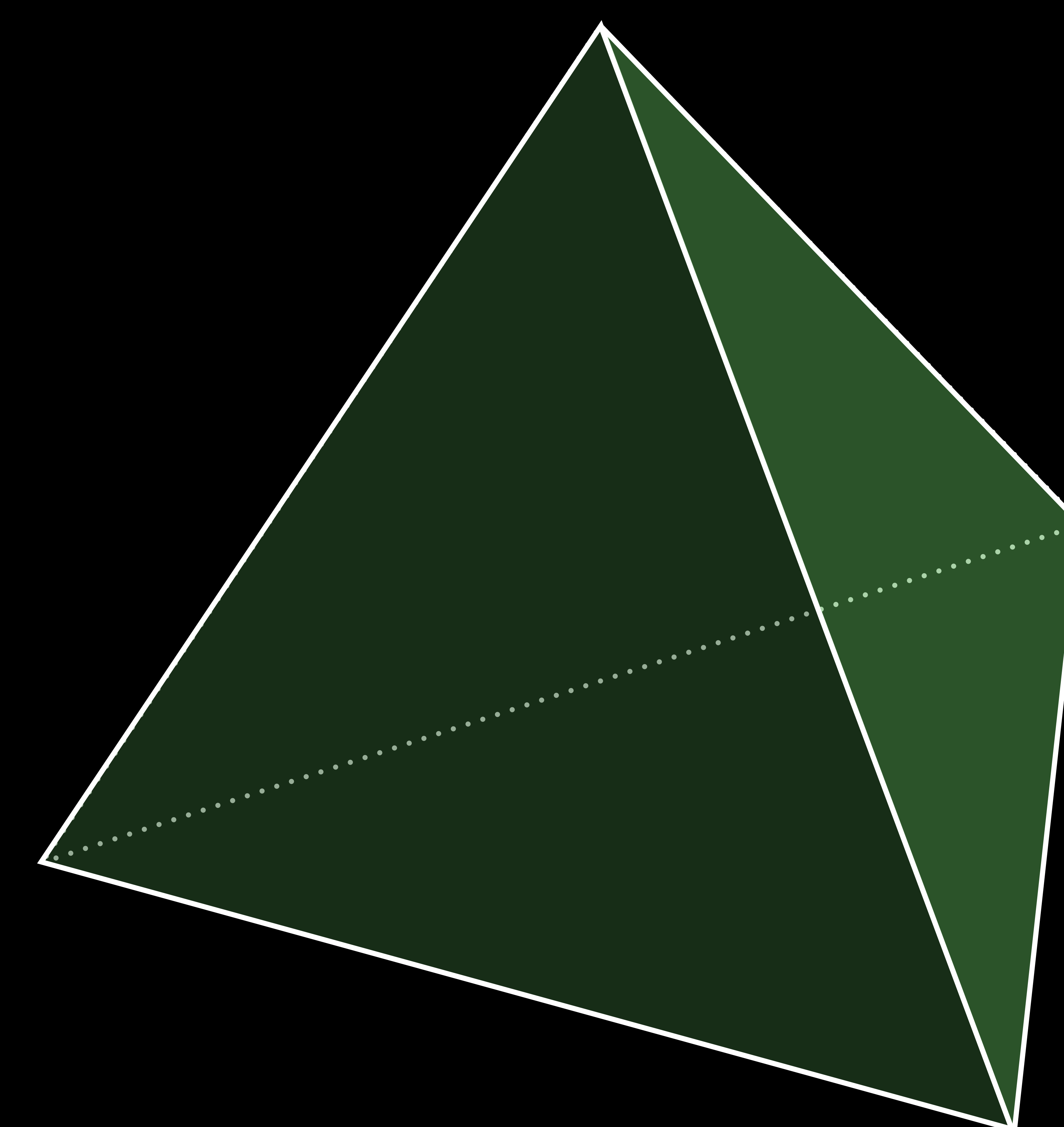
orient

Is a triangle facing toward me or away from me?



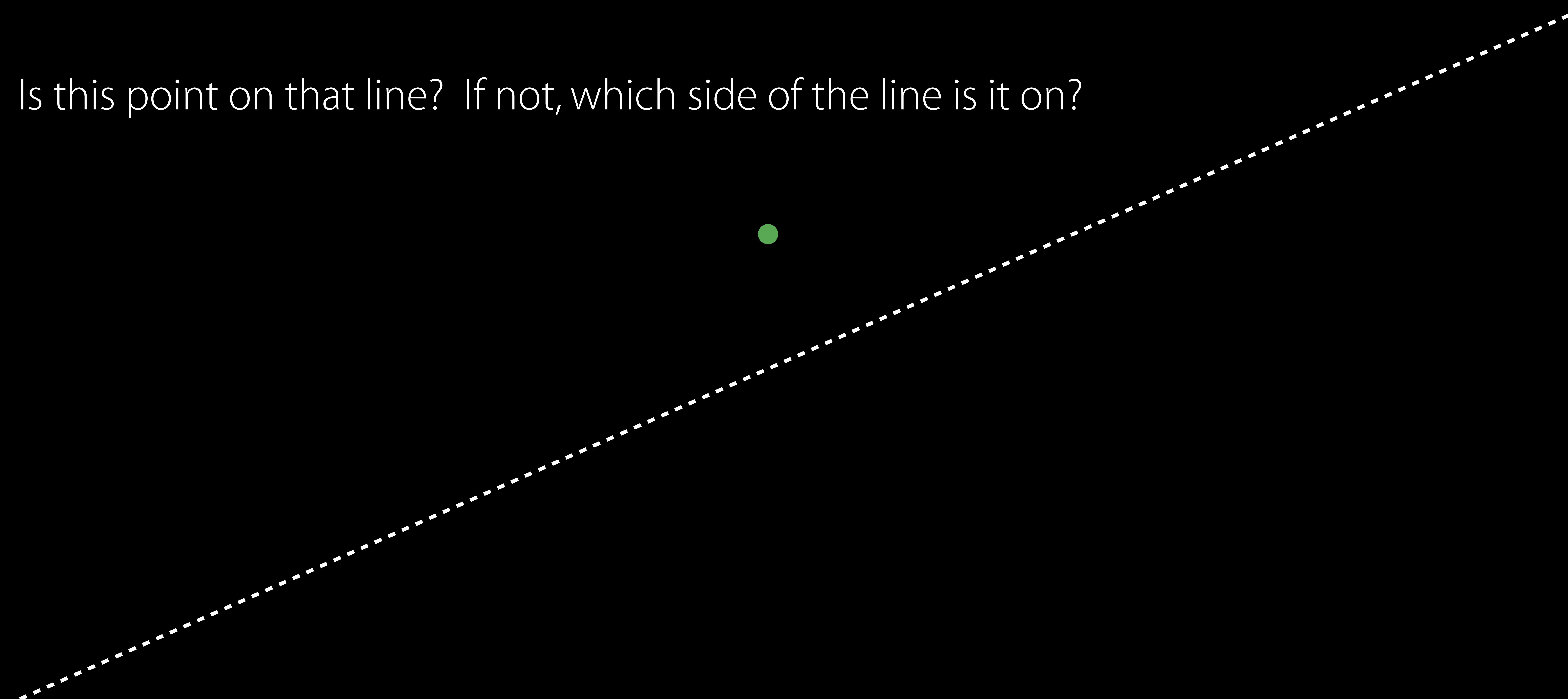
orient

Is a triangle facing toward me or away from me?



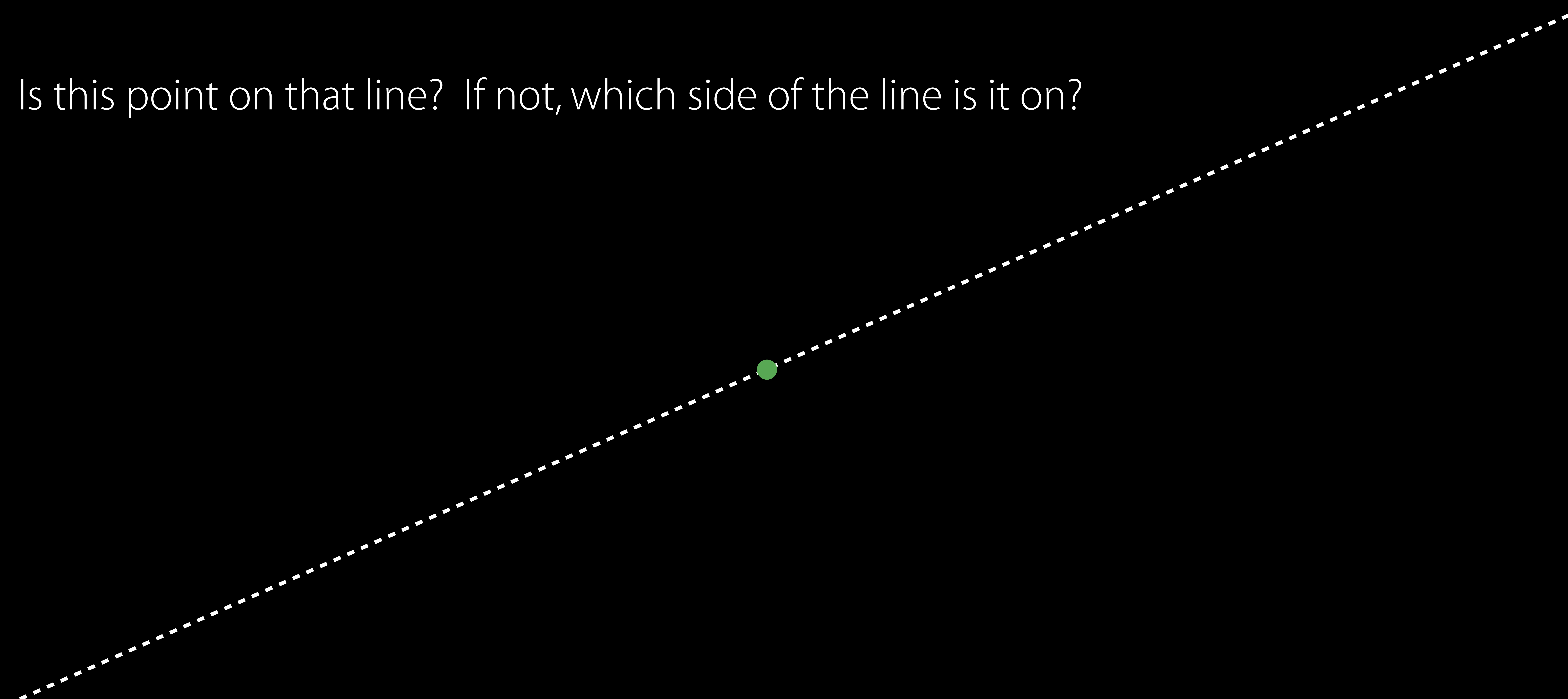
orient

Is this point on that line? If not, which side of the line is it on?



orient

Is this point on that line? If not, which side of the line is it on?

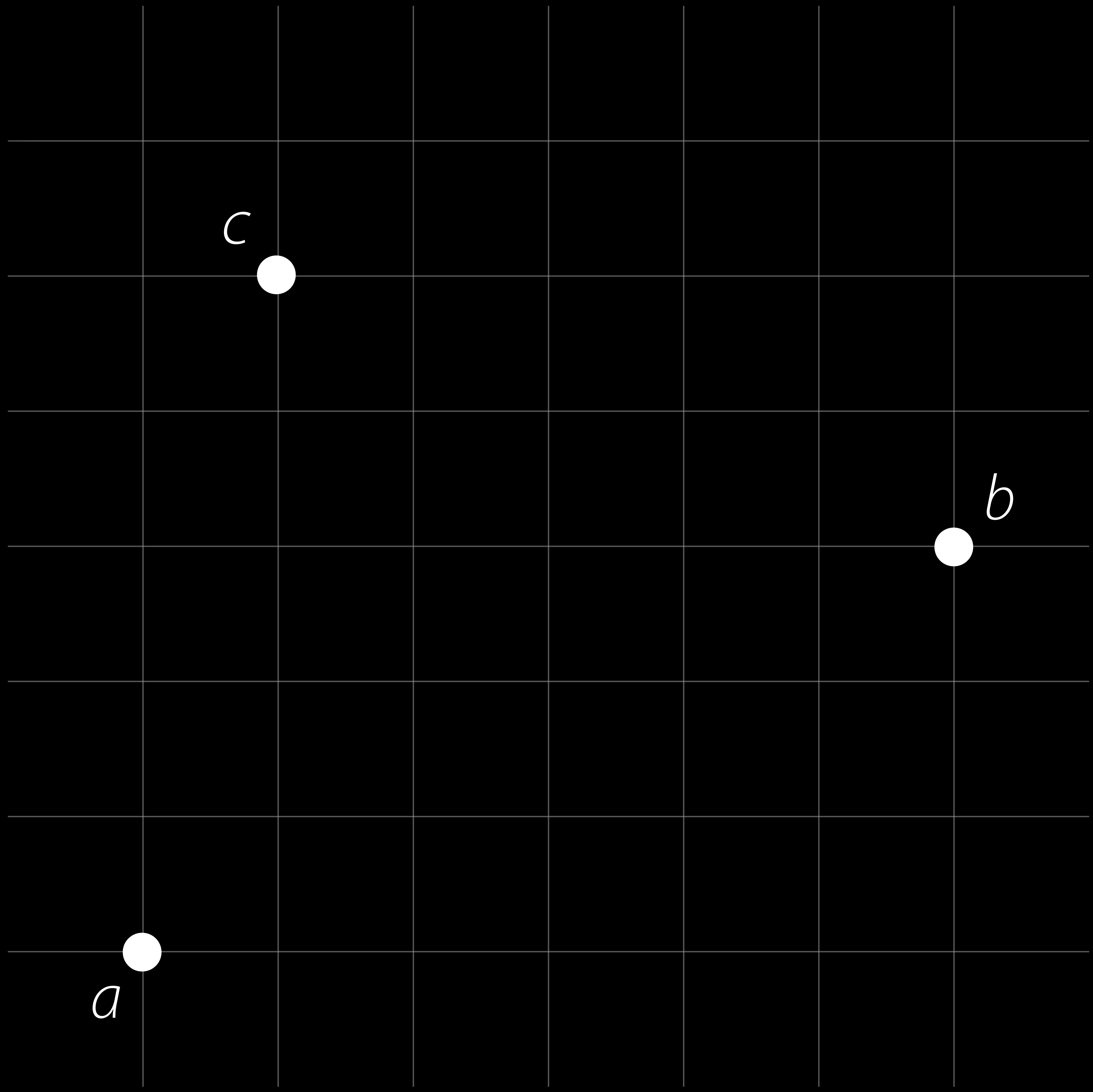


orient Example

```
let a = float2(0,0)
let b = float2(6,3)
let c = float2(1,5)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```

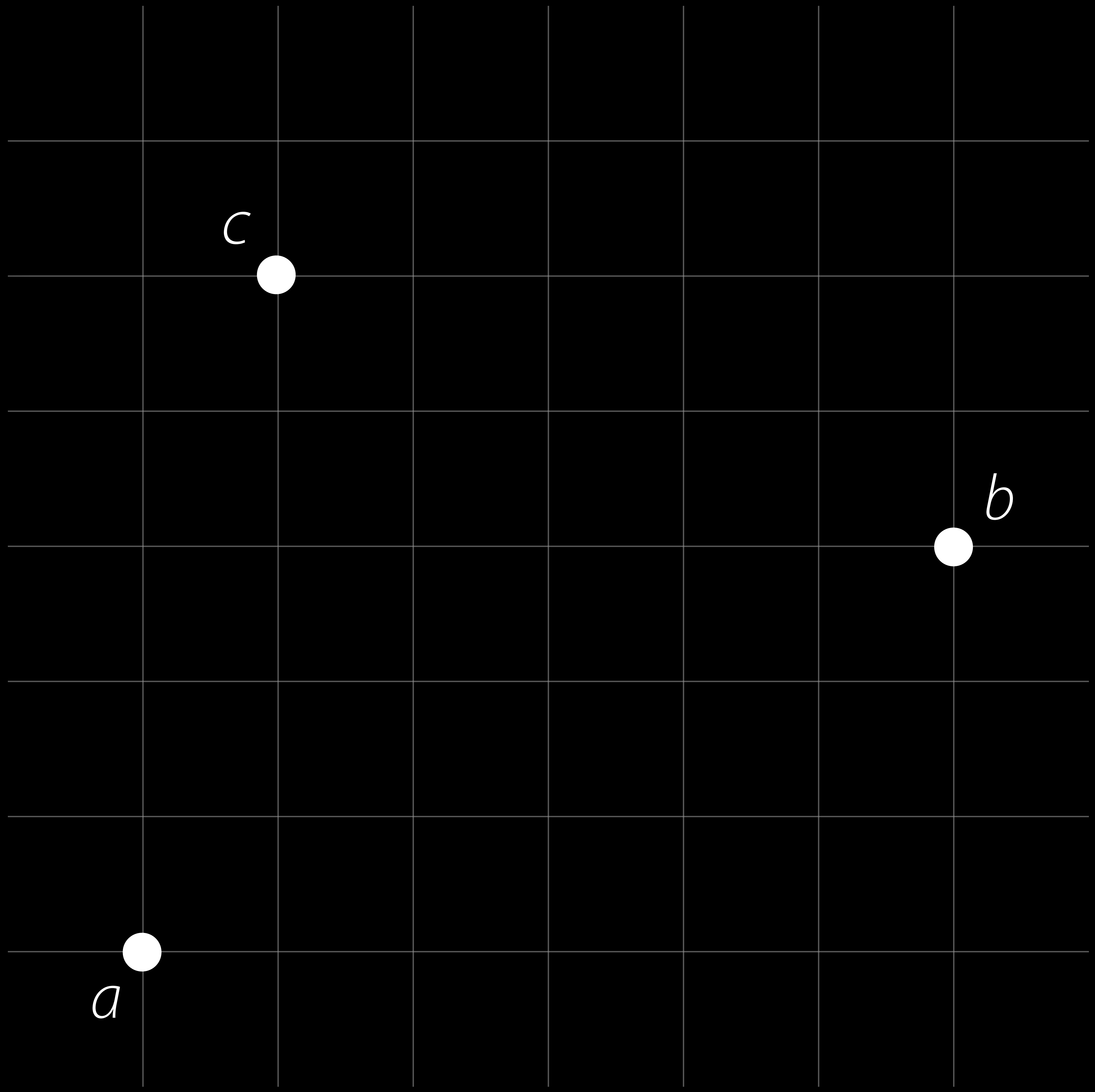
orient Example

```swift
let a = float2(0,0)
let b = float2(6,3)
let c = float2(1,5)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```



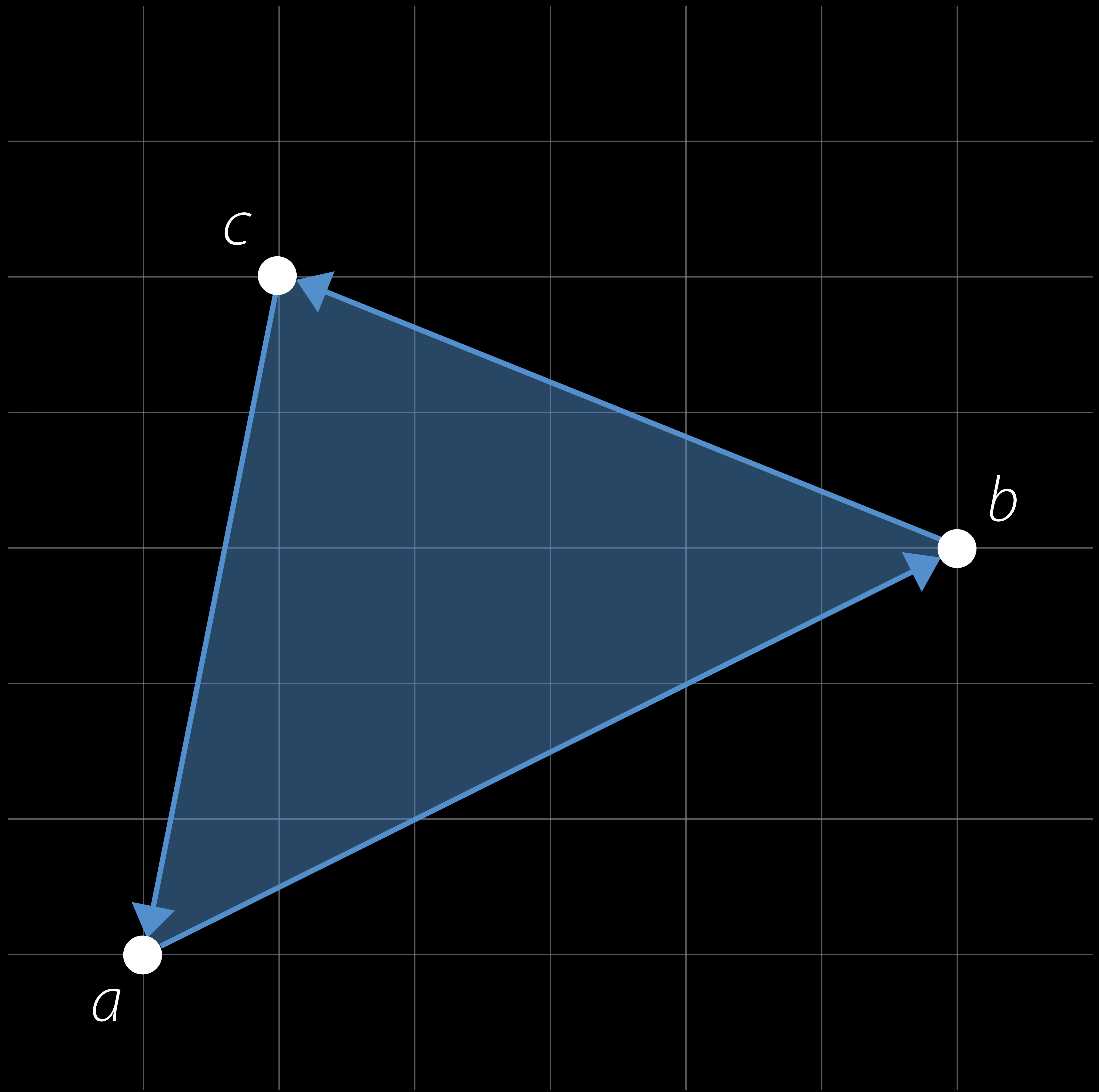
orient Example

```swift
let a = float2(0,0)
let b = float2(6,3)
let c = float2(1,5)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```



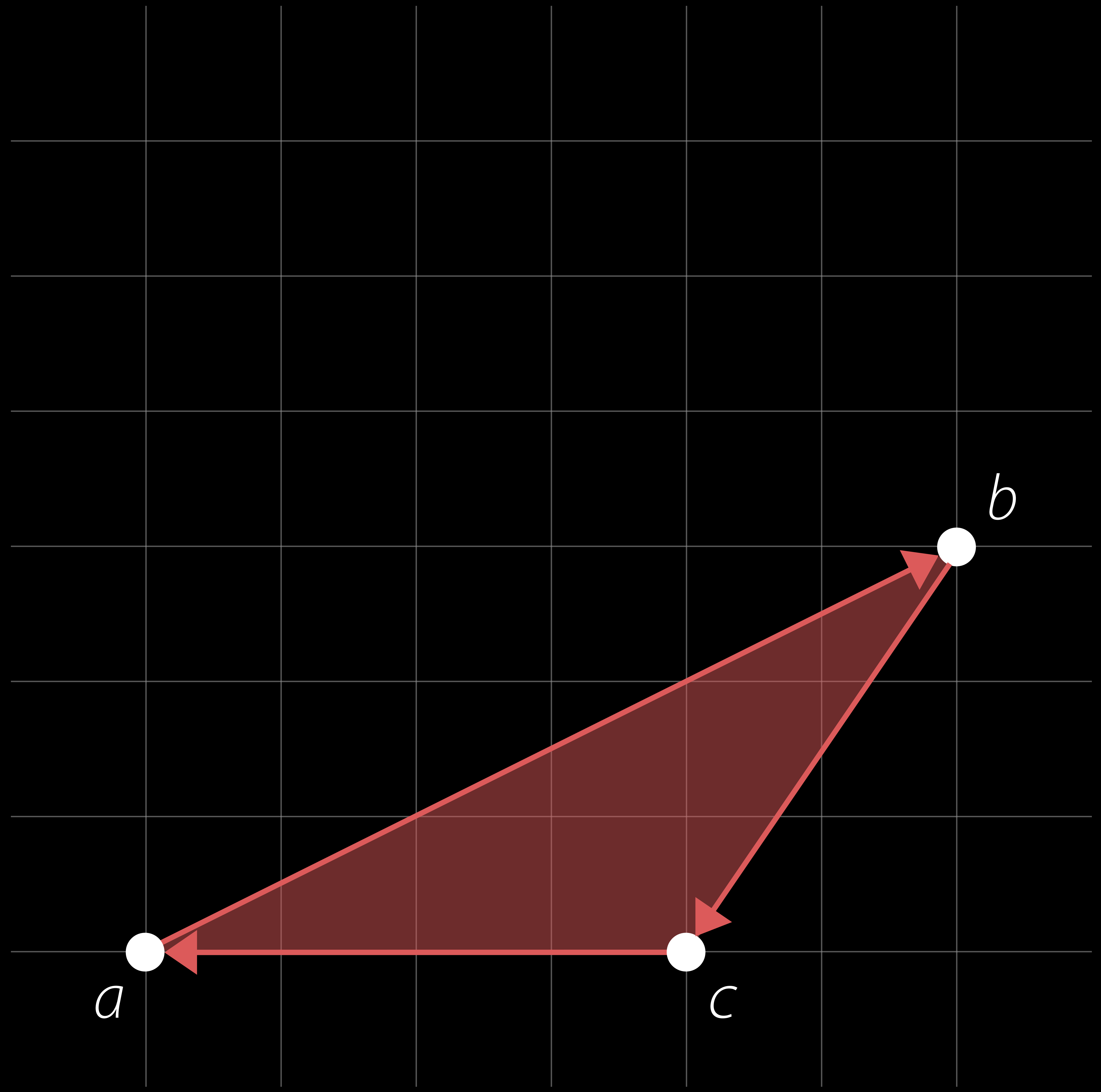
orient Example

```swift
let a = float2(0,0)
let b = float2(6,3)
let c = float2(1,5)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```



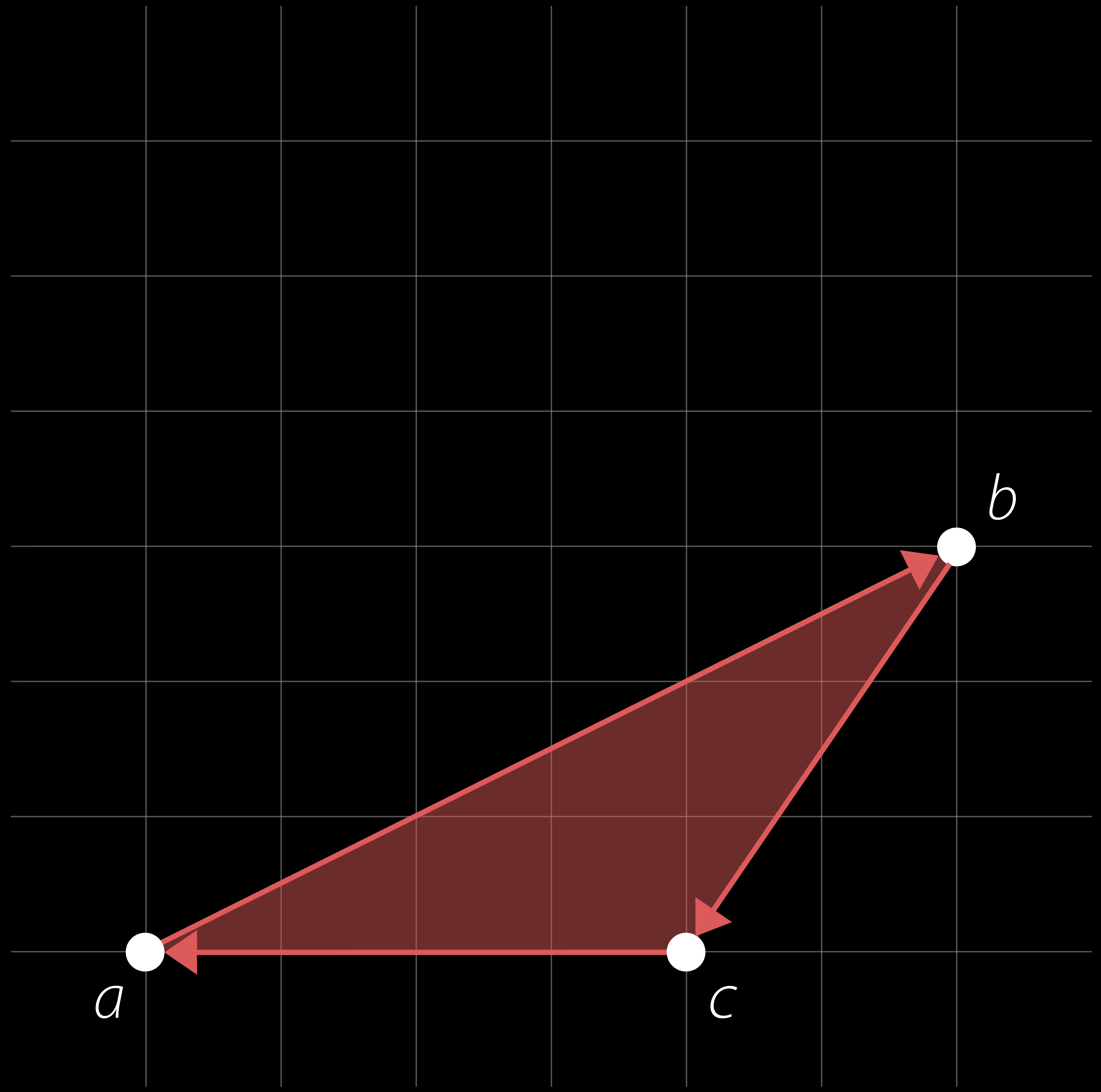
orient Example

```
let a = float2(0,0)
let b = float2(6,3)
let c = float2(4,0)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```



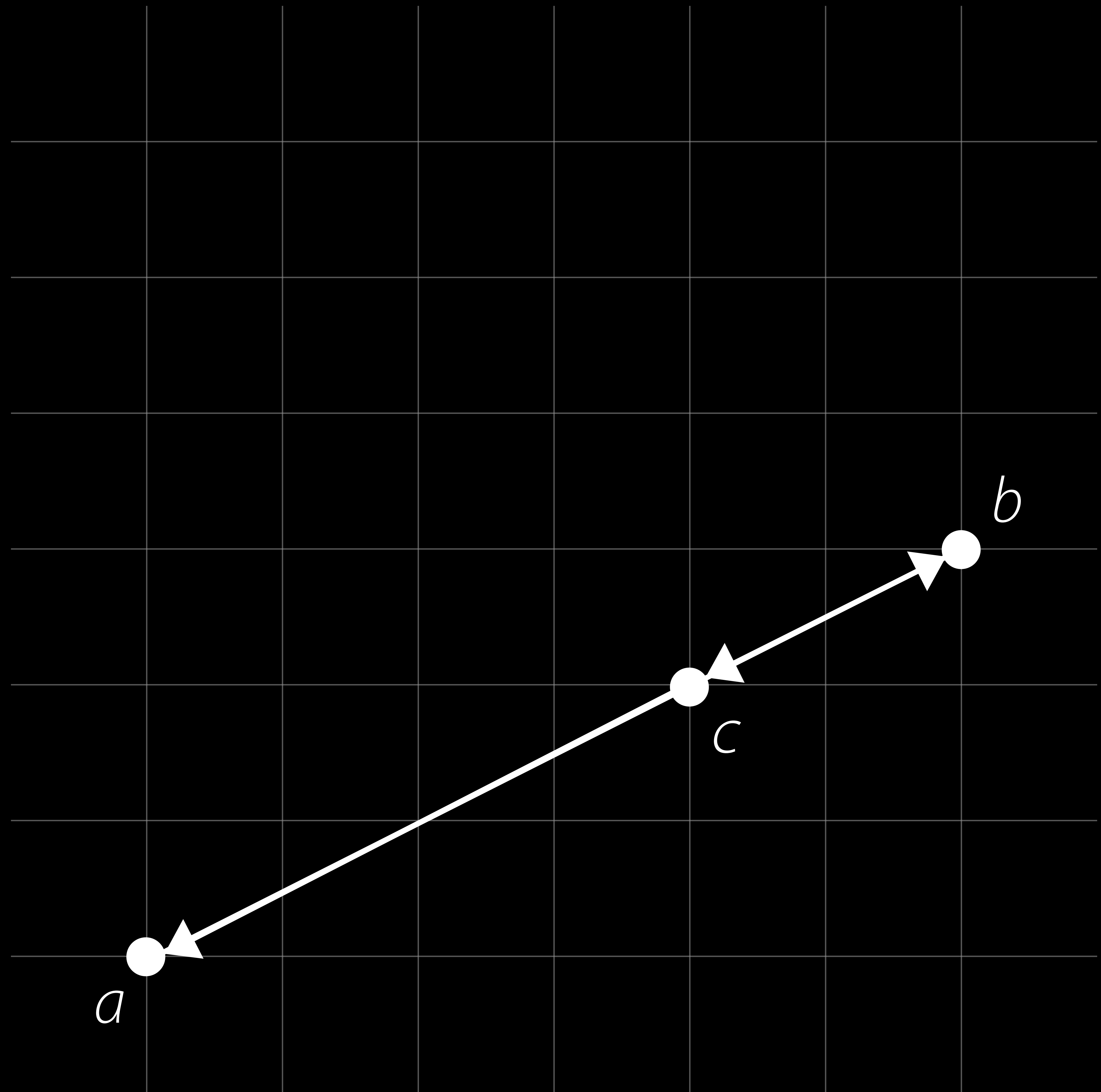
orient Example

```swift
let a = float2(0,0)
let b = float2(6,3)
let c = float2(4,0)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```



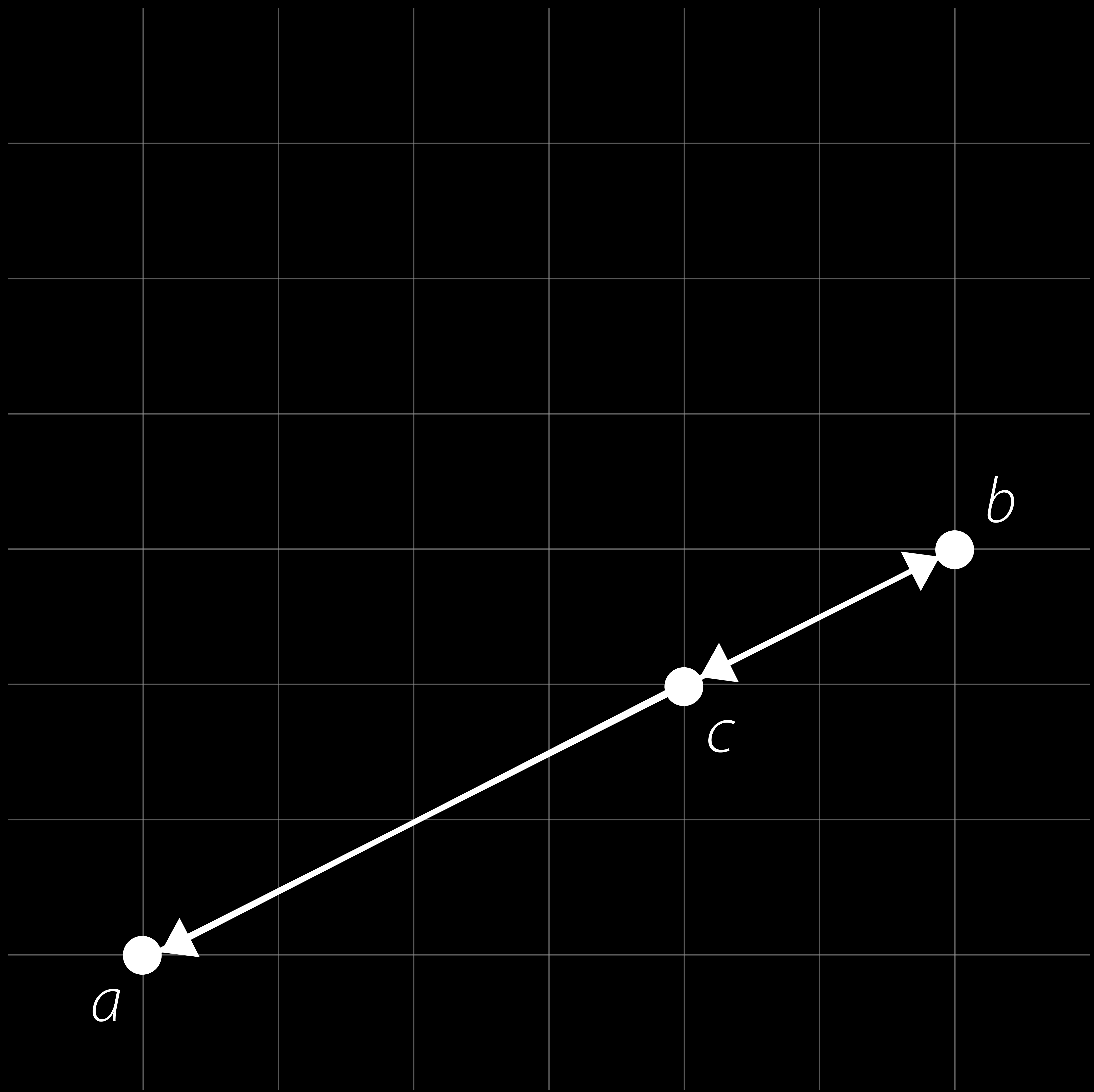
orient Example

```
let a = float2(0,0)
let b = float2(6,3)
let c = float2(4,2)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```



orient Example

```
let a = float2(0,0)
let b = float2(6,3)
let c = float2(4,2)

let orientation = simd_orient(a, b, c)

if orientation > 0 {
    print("(a,b,c) is positively oriented.")
}

else if orientation < 0 {
    print("(a,b,c) is negatively oriented.")
}

else /* orientation is zero */ {
    print("(a,b,c) are collinear.")
}
```



Numerical Stability

Numerical Stability

Orientation is numerically *unstable*

Numerical Stability

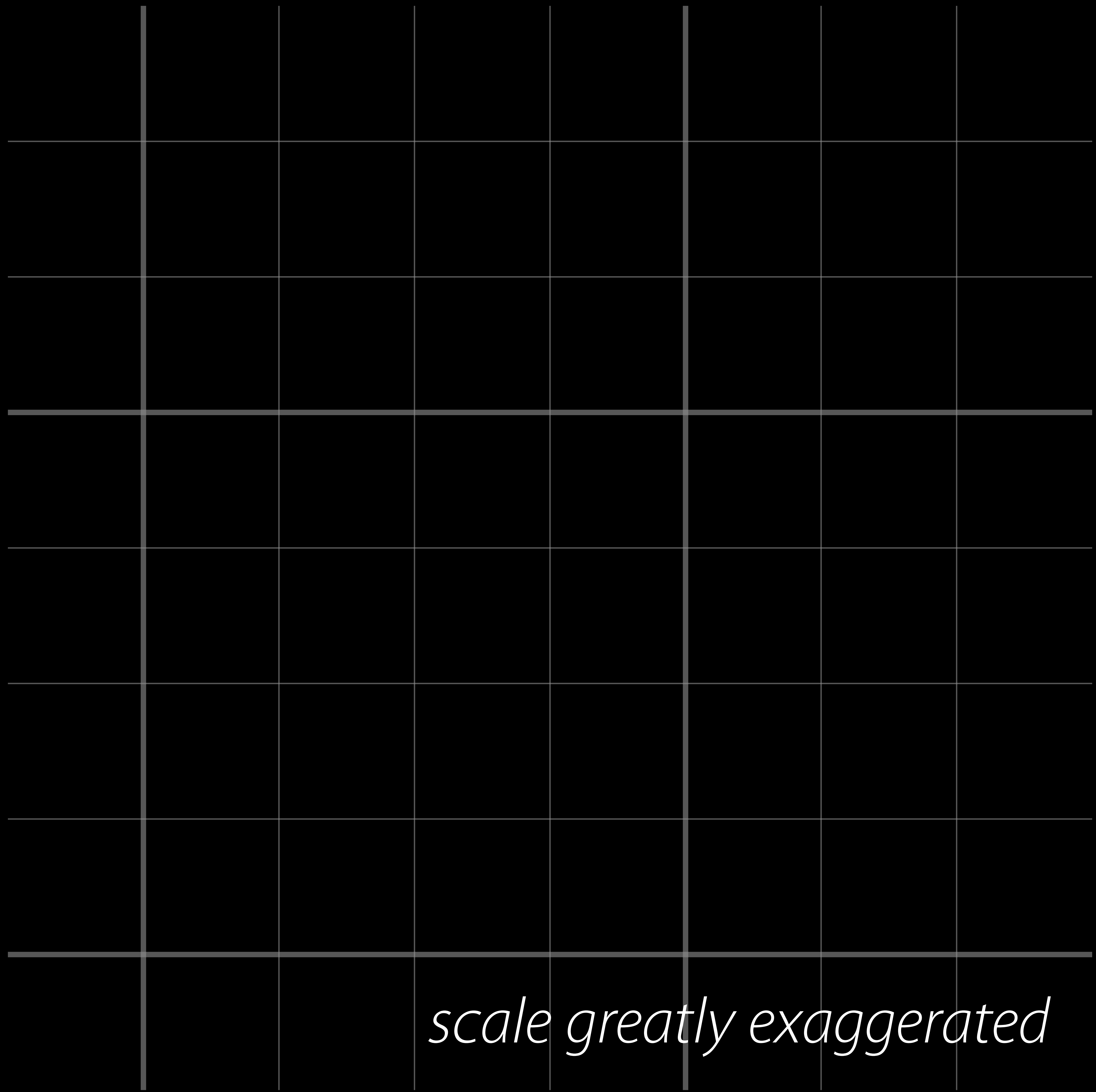
Orientation is numerically *unstable*

When points are nearly collinear, usual algorithms produce garbage results

Numerical Stability

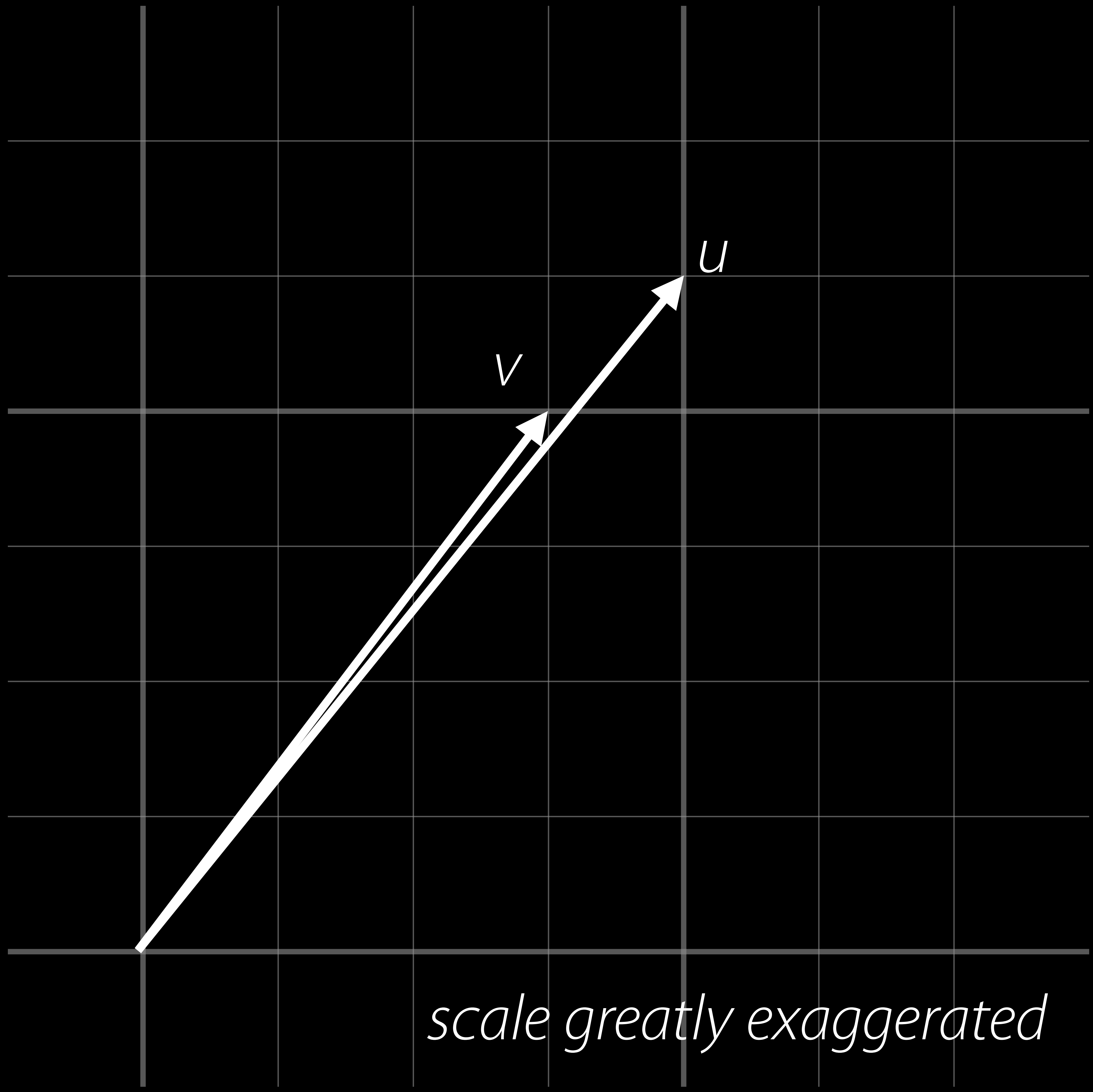
```swift
import simd

let tiny = Float(1).ulp
let u = float2(1, 1+tiny)
let v = float2(1-tiny, 1)

let m = float2x2([u, v])
matrix_determinant(m.cmatrix)
```

```
1.192093e-07
float2(1.0,1.0)
float2(1.0,1.0)

float2x2([[…
0
```



Numerical Stability

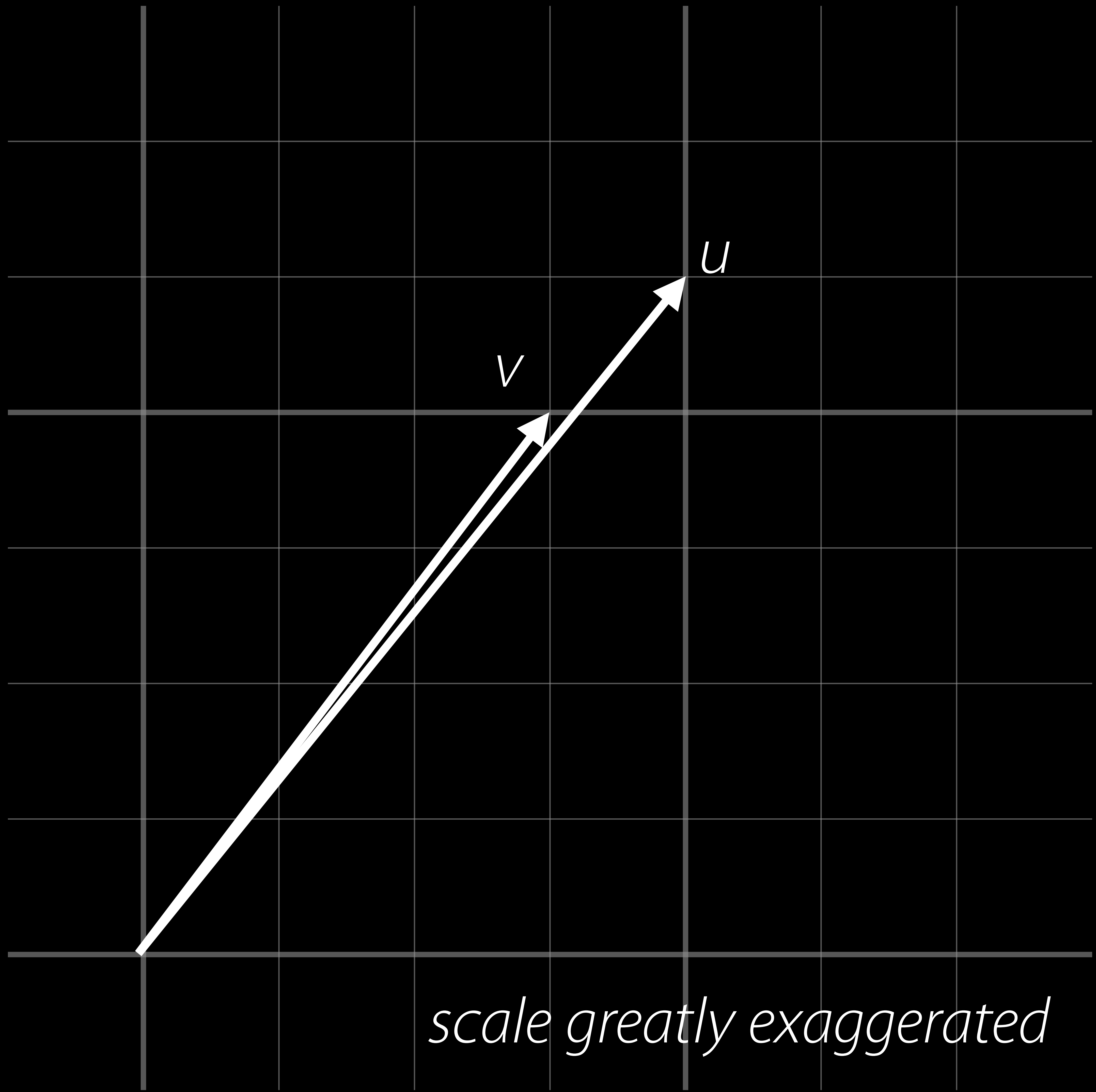
```
import simd

let tiny = Float(1).ulp                    1.192093e-07
let u = float2(1, 1+tiny)                  float2(1.0,1.0)
let v = float2(1-tiny, 1)                  float2(1.0,1.0)

let m = float2x2([u, v])                   float2x2([[…
matrix_determinant(m.cmatrix)              0
```



Numerical Stability

```
import simd

let tiny = Float(1).ulp
let u = float2(1, 1+tiny)
let v = float2(1-tiny, 1)

let m = float2x2([u, v])
matrix_determinant(m.cmatrix)
```

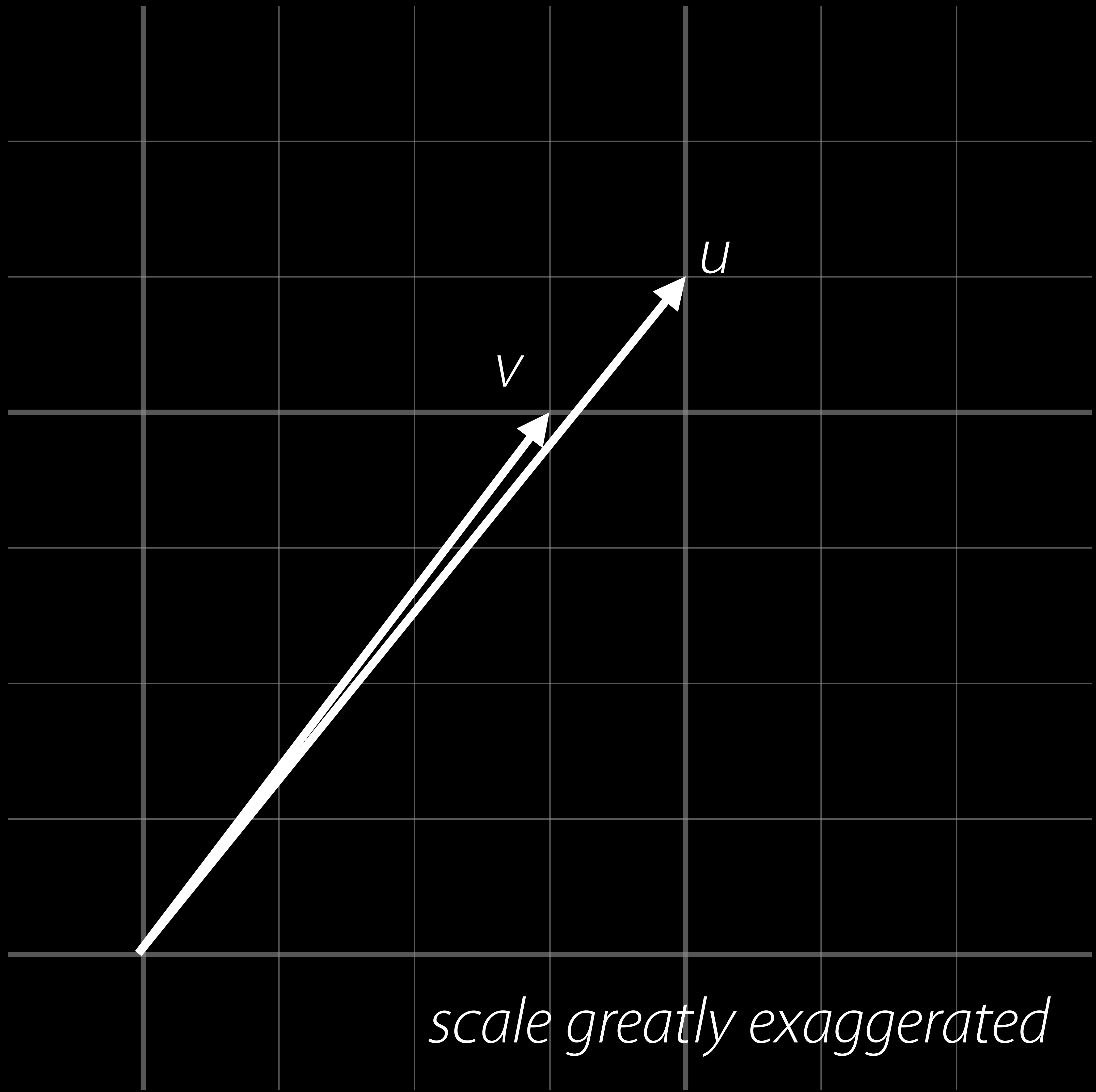
```
1.192093e-07
float2(1.0,1.0)
float2(1.0,1.0)

float2x2([[…
0
```



Numerical Stability

```
import simd

let tiny = Float(1).ulp
let u = float2(1, 1+tiny)
let v = float2(1-tiny, 1)

simd_orient(u, v)
```

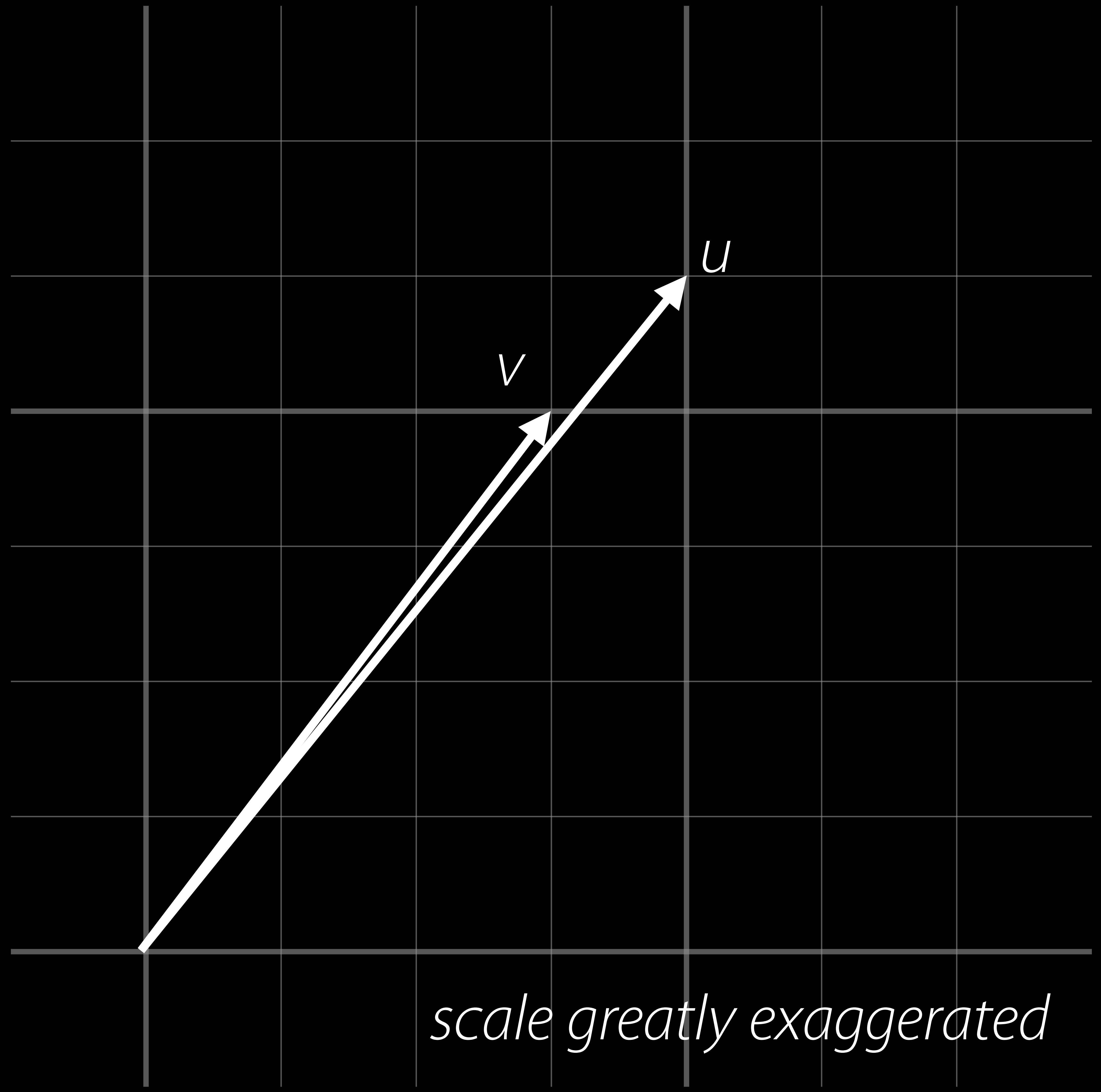
```
1.192093e-07
float2(1.0,1.0)
float2(1.0,1.0)

1.421085e-14
```



Numerical Stability

```
import simd

let tiny = Float(1).ulp
let u = float2(1, 1+tiny)
let v = float2(1-tiny, 1)

simd_orient(u, v)
```

```
1.192093e-07
float2(1.0,1.0)
float2(1.0,1.0)

1.421085e-14
```



Numerical Stability

Numerical Stability

These geometric predicates use *adaptive precision*

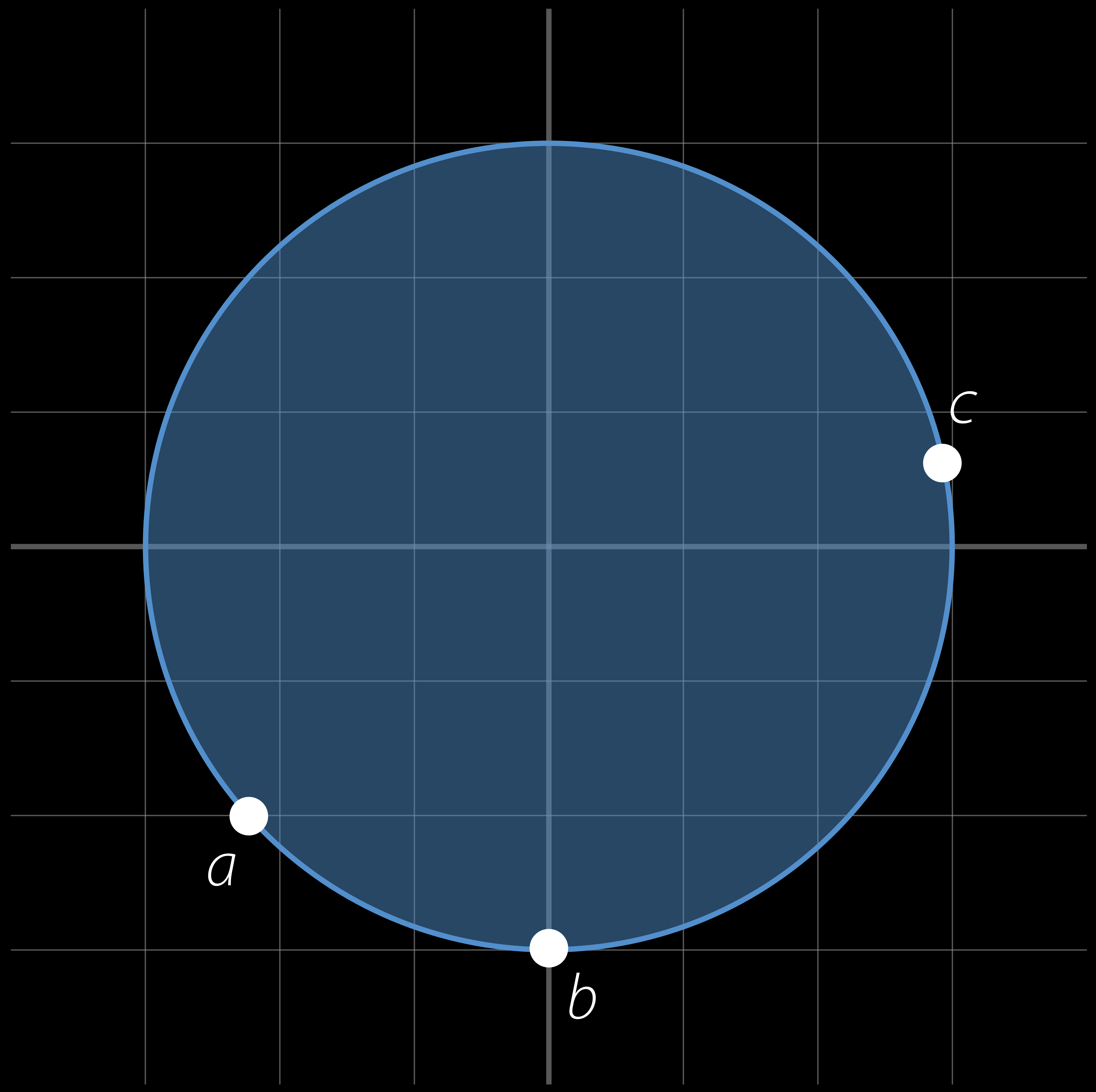
Numerical Stability

These geometric predicates use *adaptive precision*

Computation uses as many bits as needed to produce the correct result

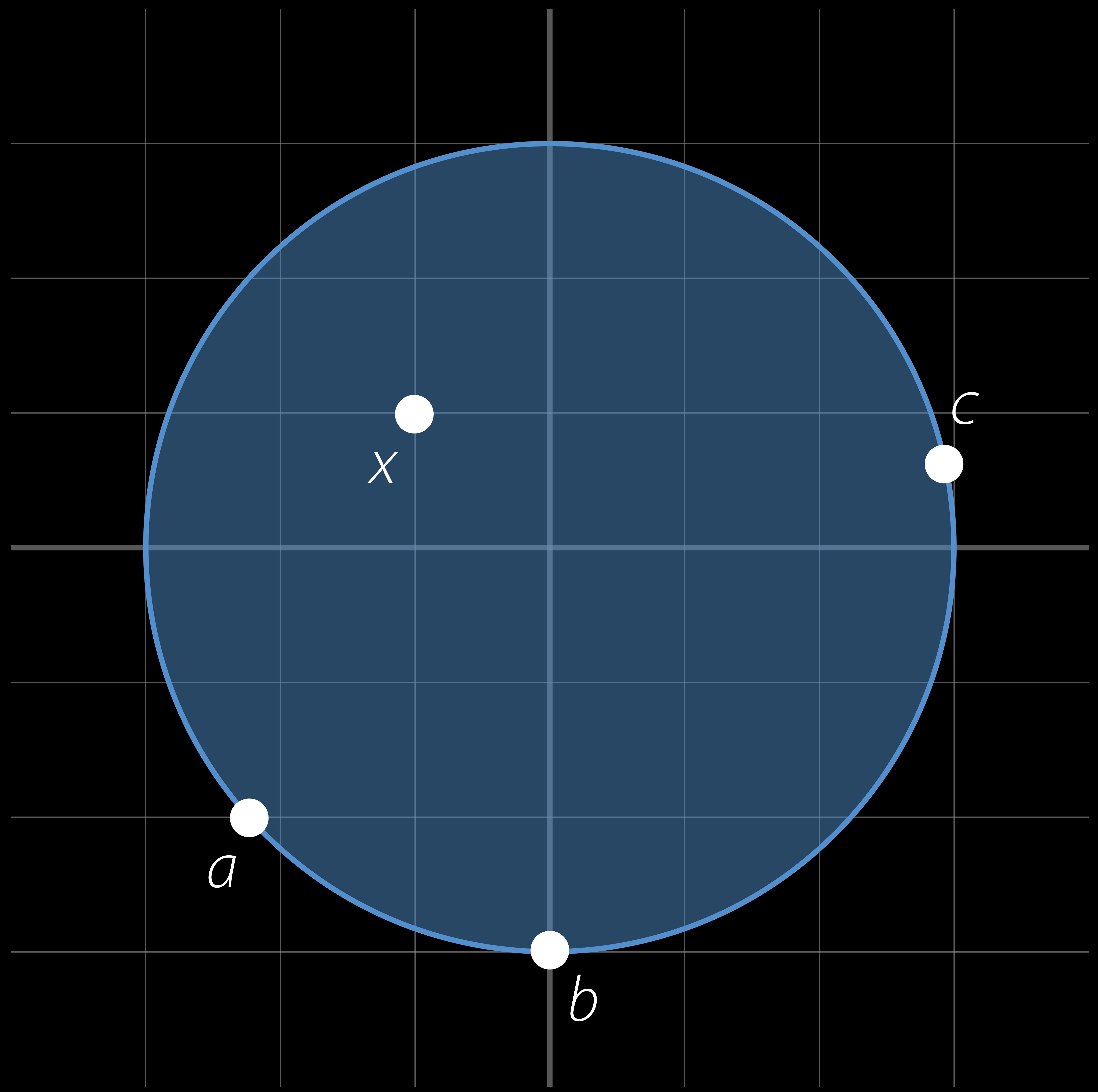
incircle

Three points (a, b, c) determine a circle



incircle

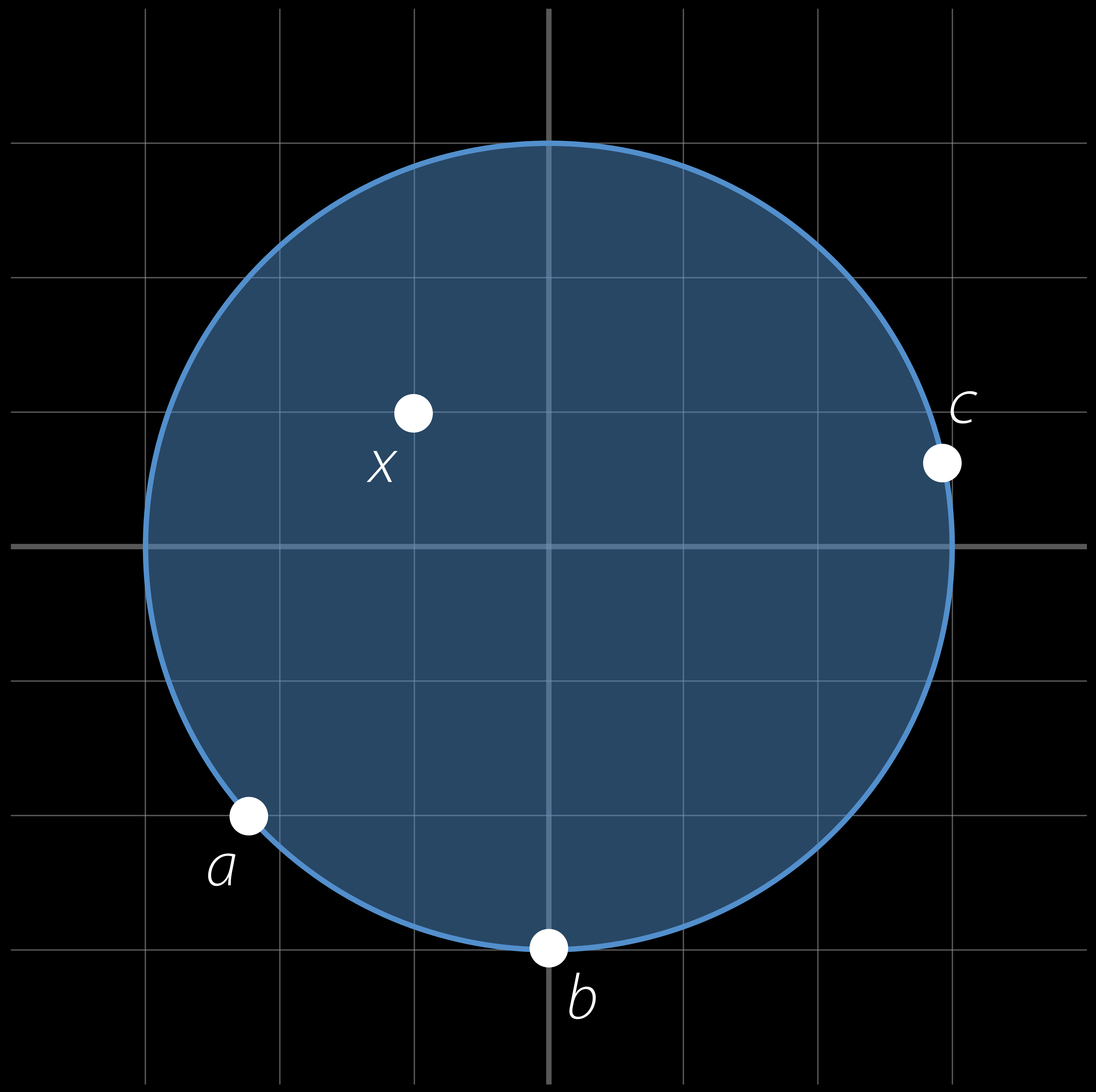
simd_incircle(x, a, b, c)



incircle

simd_incircle(x, a, b, c)

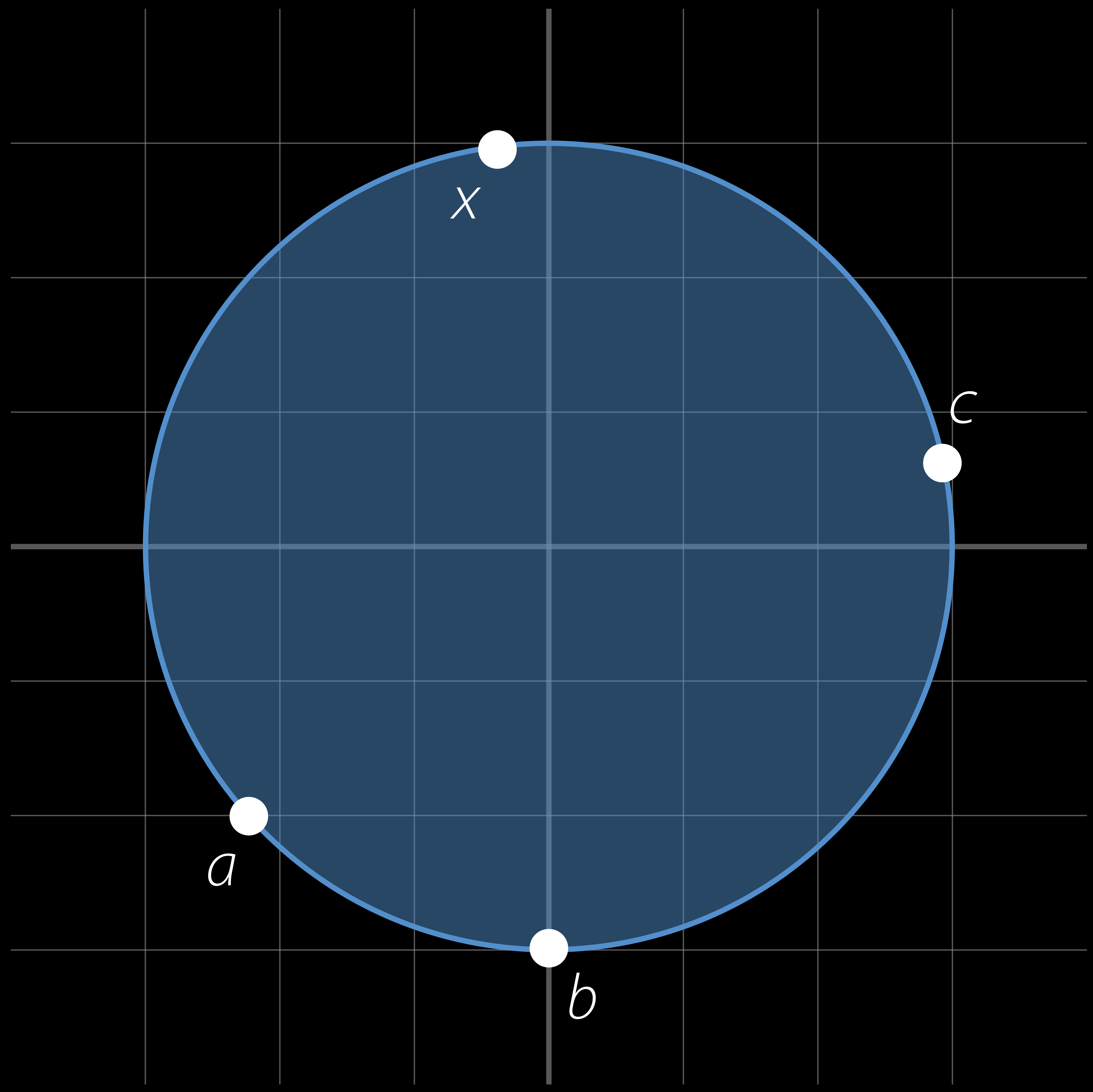
- Positive if x is inside the circle



incircle

simd_incircle(x, a, b, c)

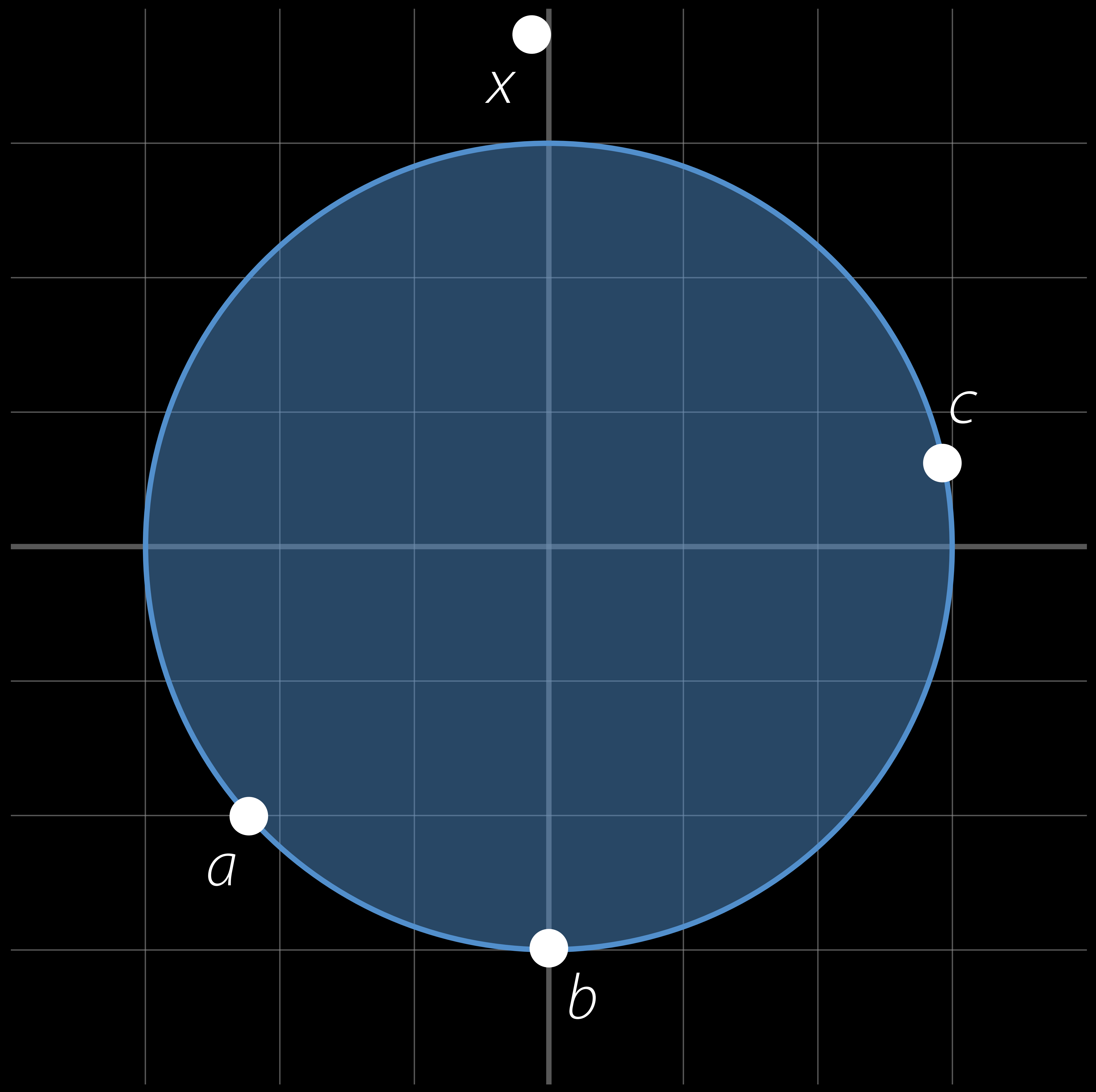
- Positive if x is inside the circle
- Zero if x is on the circle



incircle

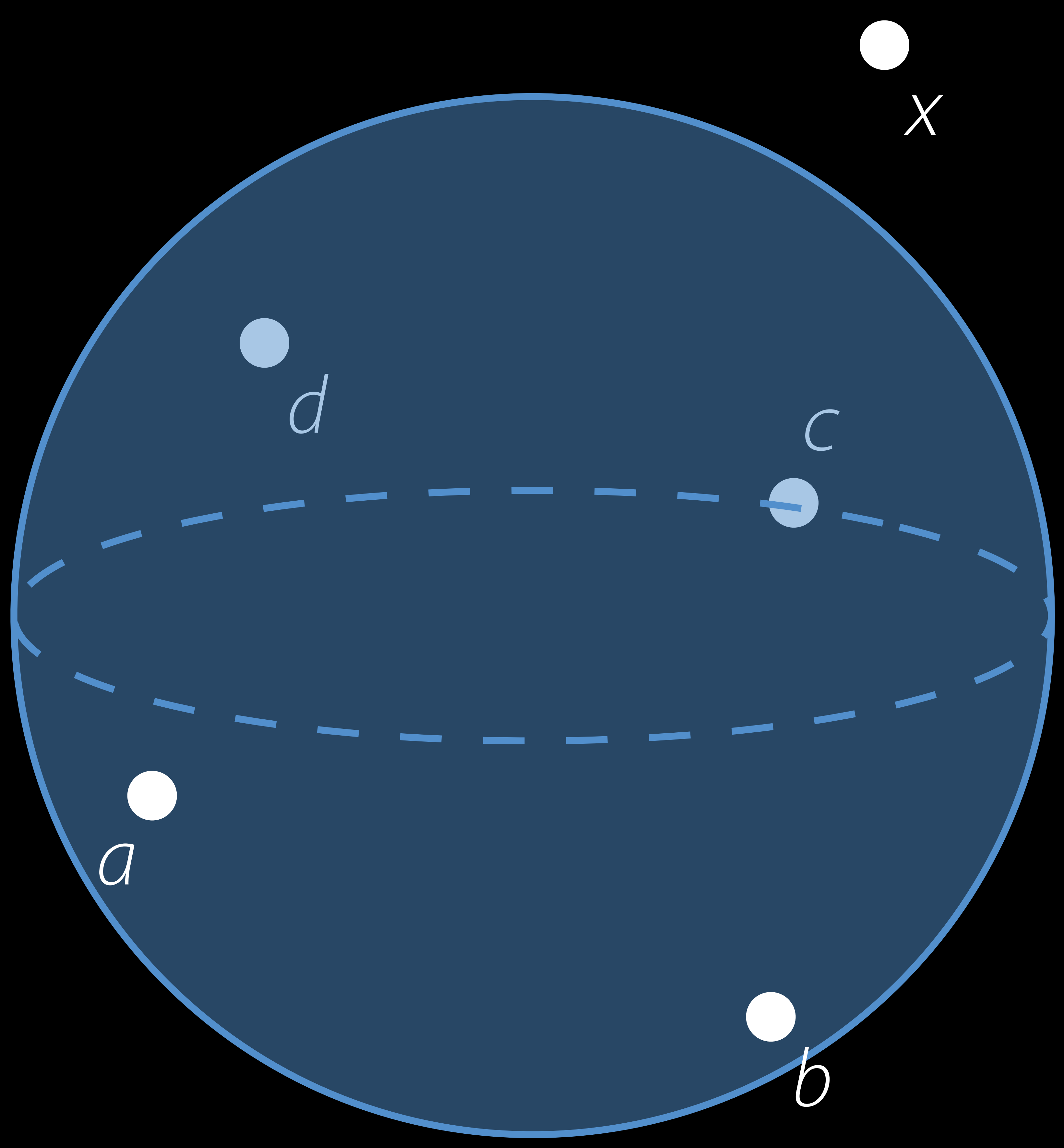
simd_incircle(x, a, b, c)

- Positive if x is inside the circle
- Zero if x is on the circle
- Negative if x is outside the circle



insphere

simd_insphere(x, a, b, c, d) is the same thing in three dimensions



```swift
import simd

/// Simple struct representing a triangle in 3 dimensions.
struct Triangle {

    var vertices: (float3, float3, float3)

    /// True iff `self` faces towards `camera`.
    func isFacing(camera: float3) -> Bool {
        //  Vector normal to front face of triangle.
        let normal = cross(vertices.0 - vertices.2, vertices.1 - vertices.2)
        //  Vector from triangle to camera.
        let toCamera = camera - vertices.2
        //  If dot product is positive, the triangle faces the camera.
        return dot(toCamera, normal) > 0
    }
}
```

```swift
import simd

/// Simple struct representing a triangle in 3 dimensions.
struct Triangle {

    var vertices: (float3, float3, float3)

    /// True iff `self` faces towards `camera`.
    func isFacing(camera: float3) -> Bool {
        //  Vector normal to front face of triangle.
        let normal = cross(vertices.0 - vertices.2, vertices.1 - vertices.2)
        //  Vector from triangle to camera.
        let toCamera = camera - vertices.2
        //  If dot product is positive, the triangle faces the camera.
        return dot(toCamera, normal) > 0
    }
}
```

```swift
import simd

/// Simple struct representing a triangle in 3 dimensions.
struct Triangle {

    var vertices: (float3, float3, float3)

    /// True iff `self` faces towards `camera`.
    func isFacing(camera: float3) -> Bool {
        //  Vector normal to front face of triangle.
        let normal = cross(vertices.0 - vertices.2, vertices.1 - vertices.2)
        //  Vector from triangle to camera.
        let toCamera = camera - vertices.2
        //  If dot product is positive, the triangle faces the camera.
        return dot(toCamera, normal) > 0
    }
}
```

```swift
import simd

/// Simple struct representing a triangle in 3 dimensions.
struct Triangle {

    var vertices: (float3, float3, float3)

    /// True iff `self` faces towards `camera`.
    func isFacing(camera: float3) -> Bool {
        return simd_orient(camera, vertices.0, vertices.1, vertices.2) > 0
    }
}
```

```swift
import simd

/// Simple struct representing a triangle in 3 dimensions.
struct Triangle {

    var vertices: (float3, float3, float3)

    /// True iff `self` faces towards `camera`.
    func isFacing(camera: float3) -> Bool {
        return simd_orient(camera, vertices.0, vertices.1, vertices.2) > 0
    }
}
```

```swift
import simd

/// Simple struct representing a triangle in 3 dimensions.
struct Triangle {

    var vertices: (float3, float3, float3)

    /// True iff `self` faces towards `camera`.
    func isFacing(camera: float3) -> Bool {
        return simd_orient(camera, vertices.0, vertices.1, vertices.2) > 0
    }
}
```

What’s New?

What’s New?

New libraries

What’s New?

New libraries

- BNNS

What's New?

New libraries

- BNNS
- Quadrature

What’s New?

New libraries

- BNNS
- Quadrature

New features

What's New?

New libraries

- BNNS
- Quadrature

New features

- Orientation and Incircle

What's New?

New libraries

- BNNS
- Quadrature

New features

- Orientation and Incircle

All added in response to feature requests!

OK, but What *Else* Is New?

OK, but What *Else* Is New?

vImage geometry operations for interleaved chroma planes

OK, but What *Else* Is New?

vImage geometry operations for interleaved chroma planes

Expanded supported formats for vImage conversion

OK, but What *Else* Is New?

vImage geometry operations for interleaved chroma planes

Expanded supported formats for vImage conversion

Improved performance for interleaved complex formats in vDSP

OK, but What *Else* Is New?

vImage geometry operations for interleaved chroma planes

Expanded supported formats for vImage conversion

Improved performance for interleaved complex formats in vDSP

Improved performance of level 2 BLAS operations

OK, but What *Else* Is New?

vImage geometry operations for interleaved chroma planes

Expanded supported formats for vImage conversion

Improved performance for interleaved complex formats in vDSP

Improved performance of level 2 BLAS operations

…

Summary

Summary

Single-stop shopping for computational operations

Summary

Single-stop shopping for computational operations

- Correct

Summary

Single-stop shopping for computational operations

- Correct
- Fast

Summary

Single-stop shopping for computational operations

- Correct
- Fast
- Energy efficient

Summary

Single-stop shopping for computational operations

- Correct
- Fast
- Energy efficient

Keep the feature requests coming!

More Information

https://developer.apple.com/wwdc16/715

Related Sessions

What's New in Metal, Part 2	Pacific Heights	Wednesday 1:40PM
Advanced Metal Shader Optimization	Nob Hill	Wednesday 3:00PM
Increase Usage of Your App with Proactive Suggestions	Mission	Friday 1:40PM

Labs

Accelerate Lab	Graphics, Games, and Media Lab D	Thursday 12:00PM
Accelerate Lab	Fort Mason	Thursday 5:00PM

WWDC16