Graphics and Games

#WWDC16

What's New in Metal

Part 2

Session 605

Charles Brissart GPU Software Engineer
Dan Omachi GPU Software Engineer
Anna Tikhonova GPU Software Engineer

Metal at WWDC This Year

A look at the sessions

Adopting Metal

Part One

- Fundamental Concepts
- Basic Drawing
- Lighting and Texturing

Part Two

- Dynamic Data Management
- CPU-GPU Synchronization
- Multithreaded Encoding

Metal at WWDC This Year

A look at the sessions

What's New in Metal

Part One

- Tessellation
- Resource Heaps and Memoryless Render Targets
- Improved Tools

Part Two

- Function Specialization and Function Resource Read-Writes
- Wide Color and Texture Assets
- Additions to Metal Performance Shaders

Metal at WWDC This Year

A look at the sessions

Advanced Shader Optimization

- Shader Performance Fundamentals
- Tuning Shader Code

What's New in Metal

What's New in Metal

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

What's New in Metal

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

Function Specialization and
Function Resource Read-Writes

Wide Color and Texture Assets

Additions to Metal Performance Shaders

Function Specialization

Charles Brissart GPU Software Engineer

Function Specialization

Typical pattern

- Write a few complex master functions
- Generate many specialized functions

Function Specialization

Typical pattern

- Write a few complex master functions
- Generate many specialized functions

Advantages

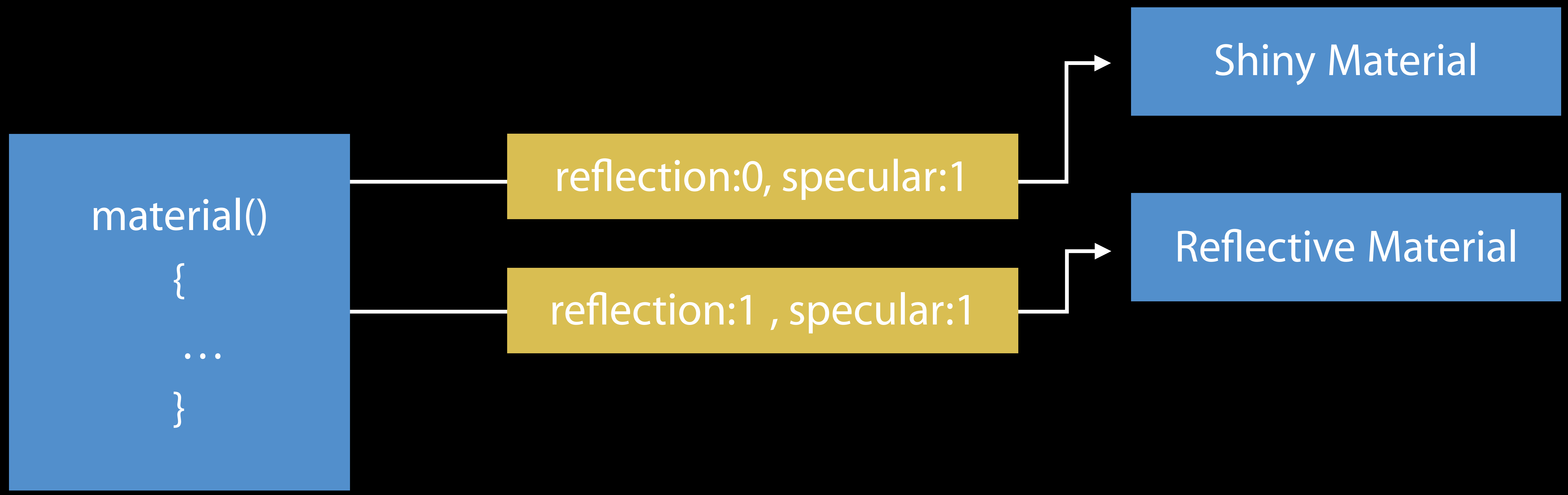
- Smaller functions
- Better performance

Material Function

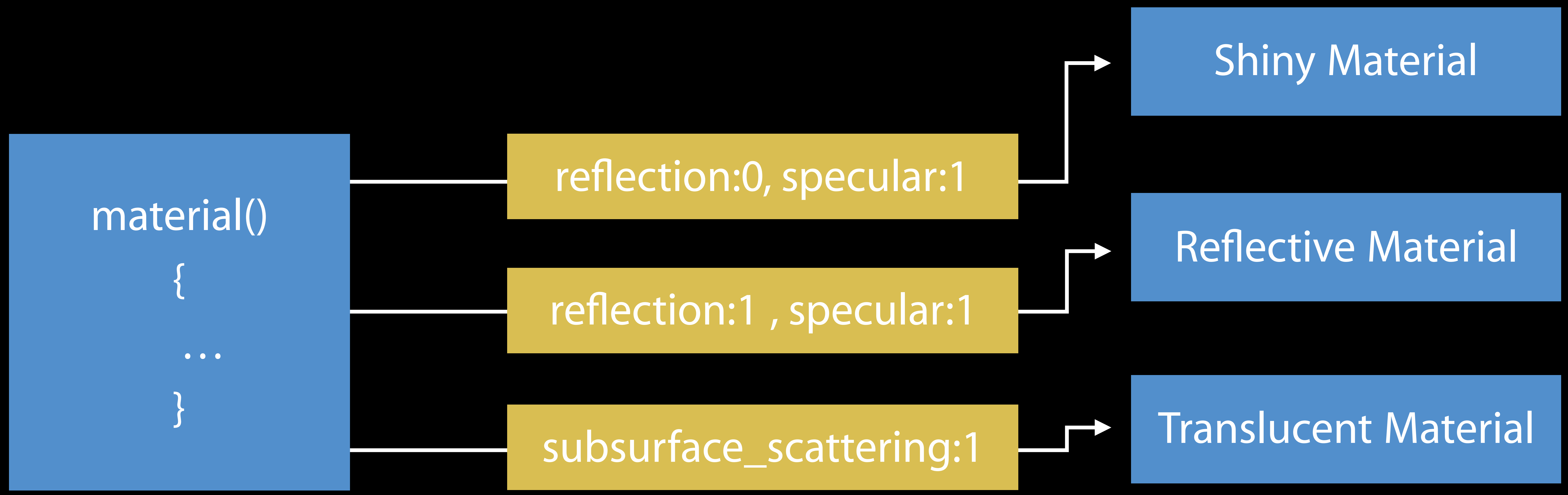
```
material()
{
    …
}
```

Material Function

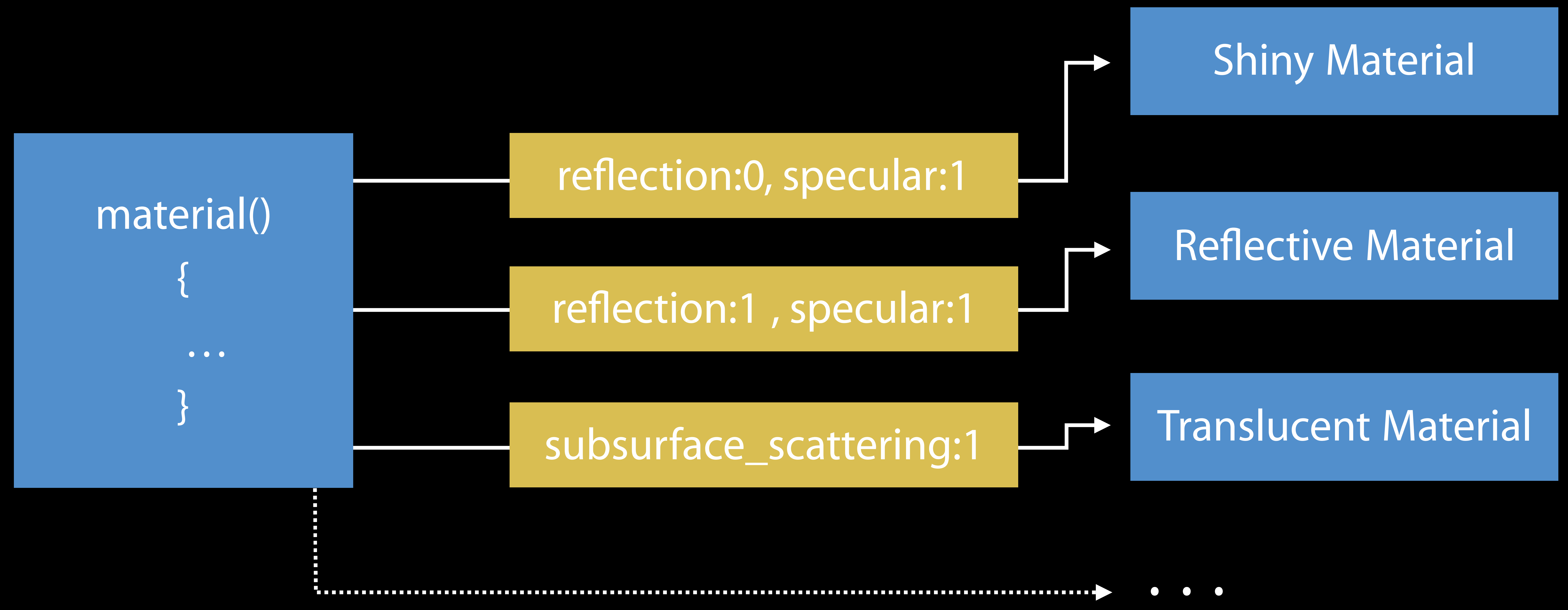
Material Function



Material Function



Material Function



Typical Implementation

Using preprocessor macros (#if, #ifdef)

- Compile at run time
  - Time consuming
- Store every variant at build time
  - Large storage

Using runtime constants

- Values are evaluated during function execution
  - Impacts performance

Function Constants

Global constants defined in the Metal Shading Language

- Compiled into IR
- Values set at run time to create a new function

Function Constants

Global constants defined in the Metal Shading Language

- Compiled into IR
- Values set at run time to create a new function

Advantages

- Master function can be compiled at build time
- Only store the master function in the Metal library
- Unused code is eliminated by a quick optimization phase

```
// Preprocessor







fragment float4 material(…) {
   float4 color = diffuseLighting(…)
   #if SPECULAR_HIGHLIGHT
   color += specularHighlight(…);
   #endif
   #if REFLECTION
   color += calculateReflection(…);
   #endif
   #if SSSCATTERING
   color += calculateSSS(…);
   #endif
   return color;
}
```

```
// Preprocessor




fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    #if SPECULAR_HIGHLIGHT
    color += specularHighlight(…);
    #endif
    #if REFLECTION
    color += calculateReflection(…);
    #endif
    #if SSSCATTERING
    color += calculateSSS(…);
    #endif
    return color;
}
```

```
// Function Constants

constant bool specular [[ function_constant(0) ]];
constant bool reflection [[ function_constant(1) ]];
constant bool scattering [[ function_constant(2) ]];

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    if(specular) {
        color += specularHighlight(…);
    }
    if(reflection) {
        color += calculateReflection(…);
    }
    if(scattering) {
        color += calculateSSS(…);
    }
    return color;
}
```

```
// Preprocessor

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    #if SPECULAR_HIGHLIGHT
    color += specularHighlight(…);
    #endif
    #if REFLECTION
    color += calculateReflection(…);
    #endif
    #if SSSCATTERING
    color += calculateSSS(…);
    #endif
    return color;
}
```

```
// Function Constants

constant bool specular [[ function_constant(0) ]];
constant bool reflection [[ function_constant(1) ]];
constant bool scattering [[ function_constant(2) ]];

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    if(specular) {
        color += specularHighlight(…);
    }
    if(reflection) {
        color += calculateReflection(…);
    }
    if(scattering) {
        color += calculateSSS(…);
    }
    return color;
}
```

```
// Preprocessor

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    #if SPECULAR_HIGHLIGHT
    color += specularHighlight(…);
    #endif
    #if REFLECTION
    color += calculateReflection(…);
    #endif
    #if SSSCATTERING
    color += calculateSSS(…);
    #endif
    return color;
}
```

```
// Function Constants

constant bool specular [[ function_constant(0) ]];
constant bool reflection [[ function_constant(1) ]];
constant bool scattering [[ function_constant(2) ]];

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    if(specular) {
        color += specularHighlight(…);
    }
    if(reflection) {
        color += calculateReflection(…);
    }
    if(scattering) {
        color += calculateSSS(…);
    }
    return color;
}
```

```
// Preprocessor

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    #if SPECULAR_HIGHLIGHT
    color += specularHighlight(…);
    #endif
    #if REFLECTION
    color += calculateReflection(…);
    #endif
    #if SSSCATTERING
    color += calculateSSS(…);
    #endif
    return color;
}
```

```
// Function Constants

constant bool specular [[ function_constant(0) ]];
constant bool reflection [[ function_constant(1) ]];
constant bool scattering [[ function_constant(2) ]];

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    if(specular) {
        color += specularHighlight(…);
    }
    if(reflection) {
        color += calculateReflection(…);
    }
    if(scattering) {
        color += calculateSSS(…);
    }
    return color;
}
```

```
// Preprocessor

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    #if SPECULAR_HIGHLIGHT
    color += specularHighlight(…);
    #endif
    #if REFLECTION
    color += calculateReflection(…);
    #endif
    #if SSSCATTERING
    color += calculateSSS(…);
    #endif
    return color;
}
```

```
// Function Constants

constant bool specular [[ function_constant(0) ]];
constant bool reflection [[ function_constant(1) ]];
constant bool scattering [[ function_constant(2) ]];

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    if(specular) {
        color += specularHighlight(…);
    }
    if(reflection) {
        color += calculateReflection(…);
    }
    if(scattering) {
        color += calculateSSS(…);
    }
    return color;
}
```

```
// Preprocessor

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    #if SPECULAR_HIGHLIGHT
    color += specularHighlight(…);
    #endif
    #if REFLECTION
    color += calculateReflection(…);
    #endif
    #if SSSCATTERING
    color += calculateSSS(…);
    #endif
    return color;
}
```

```
// Function Constants

constant bool specular [[ function_constant(0) ]];
constant bool reflection [[ function_constant(1) ]];
constant bool scattering [[ function_constant(2) ]];

fragment float4 material(…) {
    float4 color = diffuseLighting(…)
    if(specular) {
        color += specularHighlight(…);
    }
    if(reflection) {
        color += calculateReflection(…);
    }
    if(scattering) {
        color += calculateSSS(…);
    }
    return color;
}
```

Setting Constant Values

Create an `MTLFunctionConstantValues` to store the values of the constants:

```
let values = MTLFunctionConstantValues()
```

Set the values of the constants by index or by name:

```
var shadow: uint8 = 1;
values.setConstantValue(&shadow, type: MTLDataType.bool, at: 0)

var aValue: Float = 2.5;
values.setConstantValue(&aValue, type:MTLDataType.float, withName: "value")
```

Setting Constant Values

Create an `MTLFunctionConstantValues` to store the values of the constants:

```
let values = MTLFunctionConstantValues()
```

Set the values of the constants by index or by name:

```
var shadow: uint8 = 1;
values.setConstantValue(&shadow, type: MTLDataType.bool, at: 0)

var aValue: Float = 2.5;
values.setConstantValue(&aValue, type:MTLDataType.float, withName: "value")
```

Setting Constant Values

Create an `MTLFunctionConstantValues` to store the values of the constants:

```
let values = MTLFunctionConstantValues()
```

Set the values of the constants by index or by name:

```
var shadow: uint8 = 1;
values.setConstantValue(&shadow, type: MTLDataType.bool, at: 0)
```

```
var aValue: Float = 2.5;
values.setConstantValue(&aValue, type:MTLDataType.float, withName: "value")
```

Setting Constant Values

Create an `MTLFunctionConstantValues` to store the values of the constants:

```
let values = MTLFunctionConstantValues()
```

Set the values of the constants by index or by name:

```
var shadow: uint8 = 1;
values.setConstantValue(&shadow, type: MTLDataType.bool, at: 0)
```

```
var aValue: Float = 2.5;
values.setConstantValue(&aValue, type:MTLDataType.float, withName: "value")
```

Creating Specialized Functions

Create a specialized function from MTLLibrary:

```
var function : MTLFunction! = nil
do
{
   try function = library.newFunction(withName: "lightingModel", constantValues: values)
} catch let error
{
   print("Error: \(error)")
}
```

Creating Specialized Functions

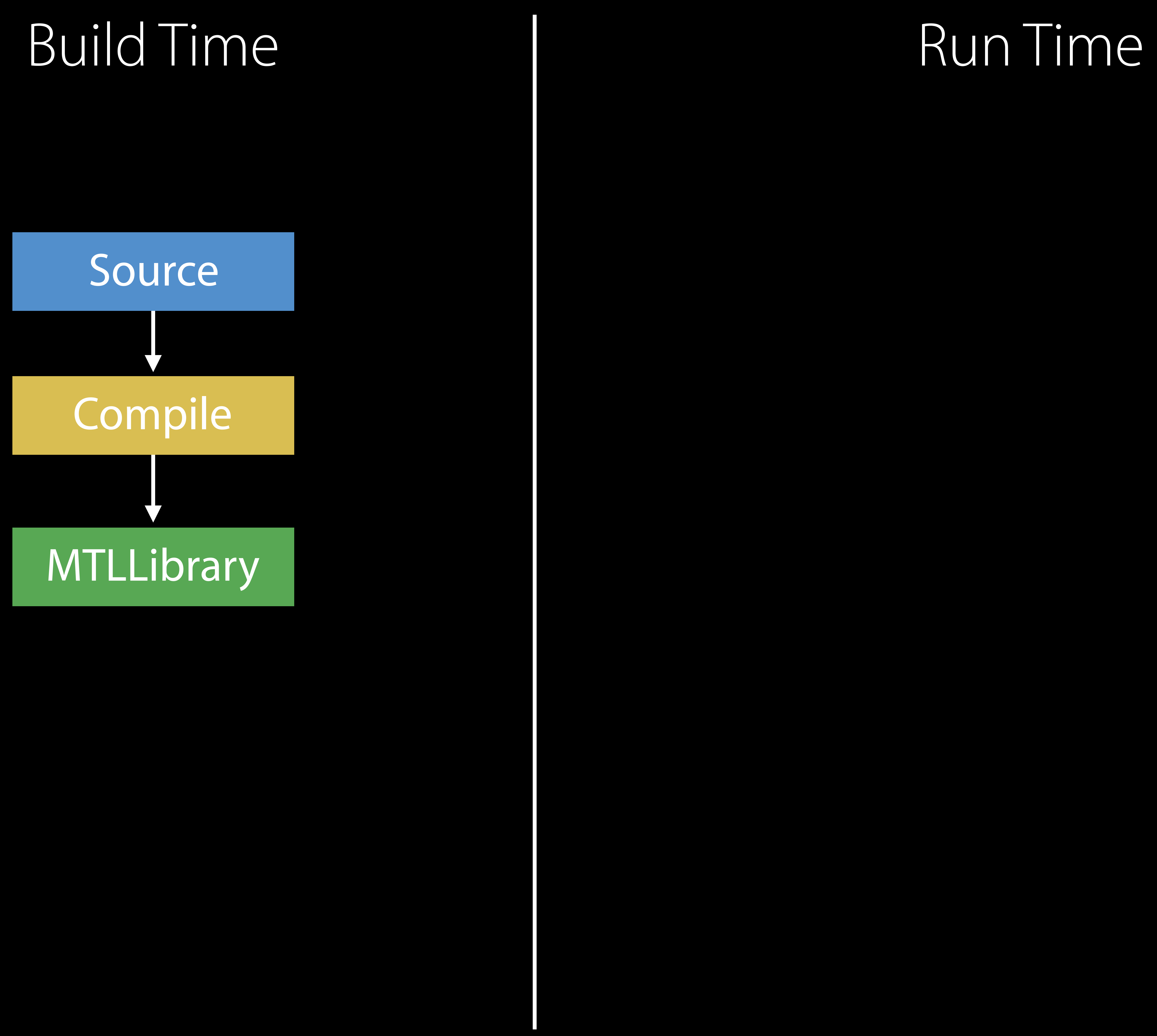
Create a specialized function from MTLLibrary:

```
var function : MTLFunction! = nil
do
{
   try function = library.newFunction(withName: "lightingModel", constantValues: values)
} catch let error
{
   print("Error: \(error)")
}
```

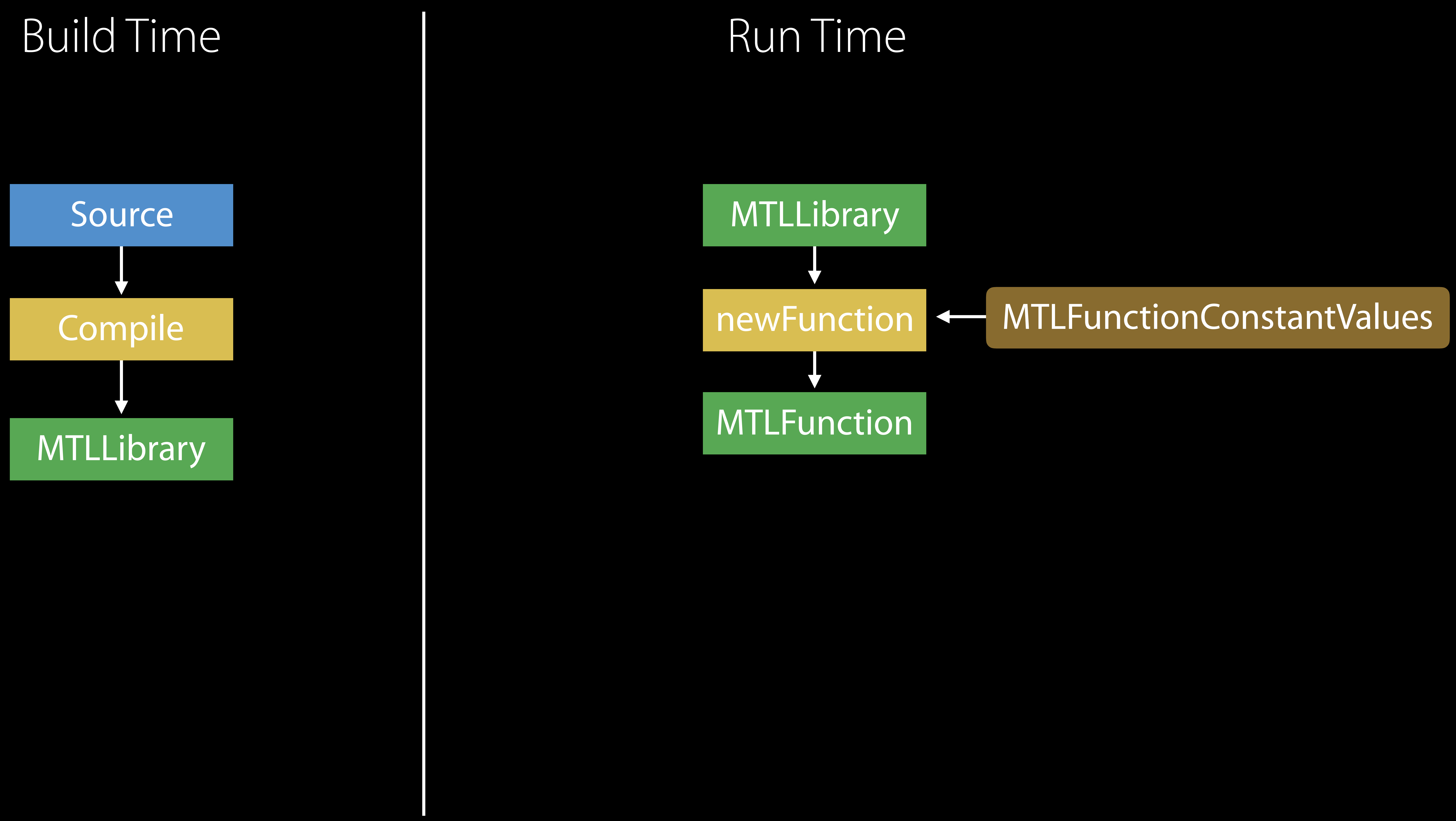
Compiler Pipeline

Build Time	Run Time

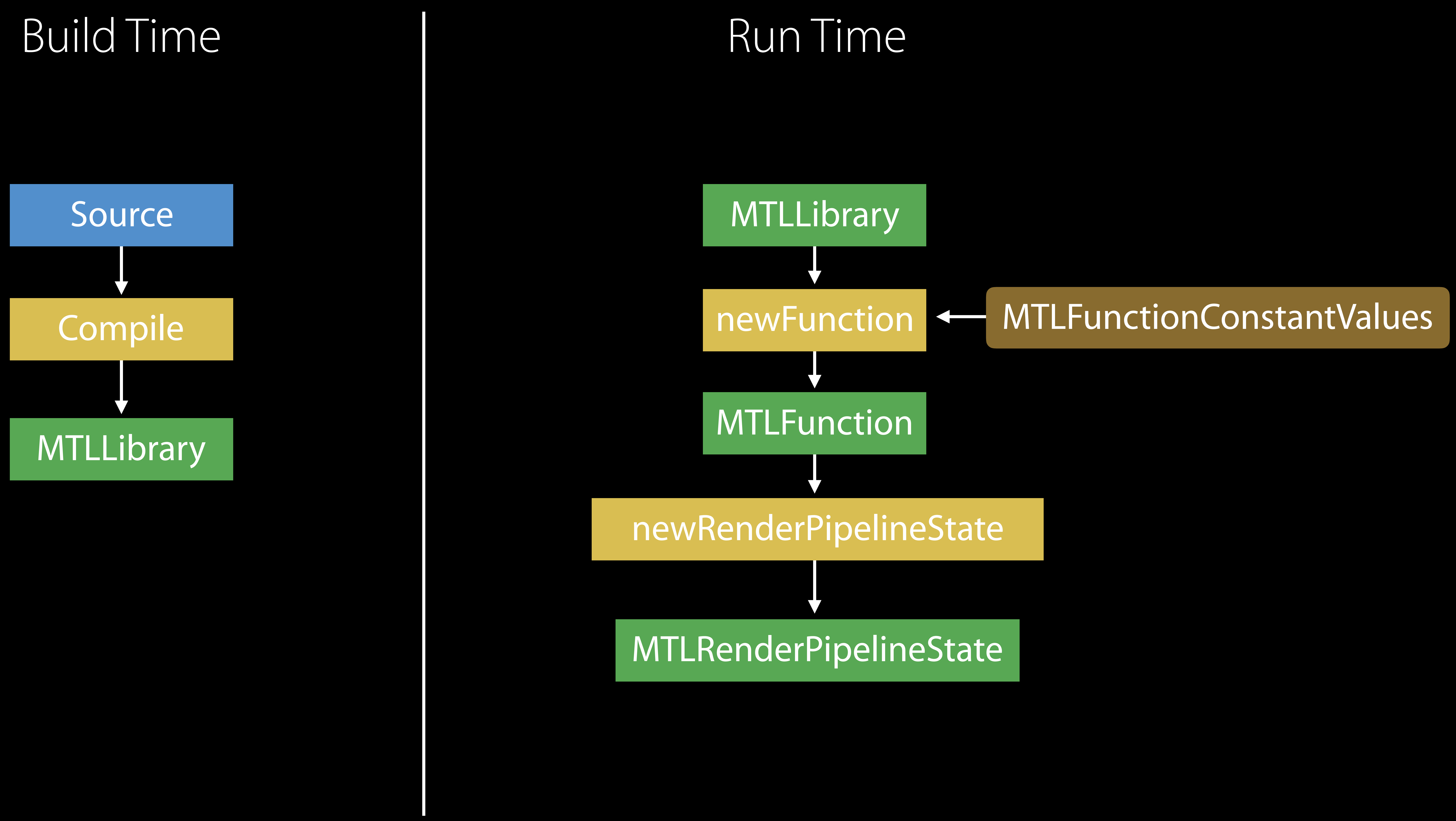
Compiler Pipeline



Compiler Pipeline



Compiler Pipeline



Declaring Constants

Scalar and vector constants of all types (float, half, int, uint, short, …):

```
constant half4 color [[ function_constant(10) ]];
```

Constant defined from other constants:

```
constant bool not_a = !a;
constant float value2 = value * 2.0f + 1.0f;
```

Optional constants (similar to #ifdef):

```
if(is_function_constant_defined(name))
```

Declaring Constants

Scalar and vector constants of all types (float, half, int, uint, short, . . .):

```
constant half4 color [[ function_constant(10) ]];
```

Constant defined from other constants:

```
constant bool not_a = !a;
constant float value2 = value * 2.0f + 1.0f;
```

Optional constants (similar to #ifdef):

```
if(is_function_constant_defined(name))
```

Declaring Constants

Scalar and vector constants of all types (float, half, int, uint, short, . . .):

```
constant half4 color [[ function_constant(10) ]];
```

Constant defined from other constants:

```
constant bool not_a = !a;
constant float value2 = value * 2.0f + 1.0f;
```

Optional constants (similar to #ifdef):

```
if(is_function_constant_defined(name))
```

Declaring Constants

Scalar and vector constants of all types (float, half, int, uint, short, …):

```
constant half4 color [[ function_constant(10) ]];
```

Constant defined from other constants:

```
constant bool not_a = !a;
constant float value2 = value * 2.0f + 1.0f;
```

Optional constants (similar to #ifdef):

```
if(is_function_constant_defined(name))
```

Declaring Constants

Scalar and vector constants of all types (float, half, int, uint, short, . . .):

```
constant half4 color [[ function_constant(10) ]];
```

Constant defined from other constants:

```
constant bool not_a = !a;
constant float value2 = value * 2.0f + 1.0f;
```

Optional constants (similar to #ifdef):

```
if(is_function_constant_defined(name))
```

Function Arguments

Function arguments can be added/removed:

- Avoid binding unused buffers or textures
- Replace an argument with an argument of a different type

```
vertex Output vertexMain(...,
    device float4x4 *matrices [[ buffer(1), function_constant(doSkinning) ]])
{
    ...
    if(doSkinning)
    {
        position = skinPosition(position, matrices, …);
    }
    ...
}
```

Function Arguments

Function arguments can be added/removed:

- Avoid binding unused buffers or textures
- Replace an argument with an argument of a different type

```
vertex Output vertexMain(...,
    device float4x4 *matrices [[ buffer(1), function_constant(doSkinning) ]])
{
    ...
    if(doSkinning)
    {
        position = skinPosition(position, matrices, …);
    }
    ...
}
```

Function Arguments

Function arguments can be added/removed:

- Avoid binding unused buffers or textures
- Replace an argument with an argument of a different type

```
vertex Output vertexMain(...,
    device float4x4 *matrices [[ buffer(1), function_constant(doSkinning) ]])
{
    ...
    if(doSkinning)
    {
        position = skinPosition(position, matrices, …);
    }
    ...
}
```

Function Arguments

Function arguments can be added/removed:

- Avoid binding unused buffers or textures
- Replace an argument with an argument of a different type

```
vertex Output vertexMain(...,
    device float4x4 *matrices [[ buffer(1), function_constant(doSkinning) ]])
{
    ...
    if(doSkinning)
    {
        position = skinPosition(position, matrices, …);
    }
    ...
}
```

[stage_in] Arguments

`[stage_in]` arguments can be added/removed:

```
struct VertexInput
{
    float4 position [[ attribute(0) ]];
    float4 color [[ attribute(1),  function_constant(enable_color) ]];
    half4 lowp_color [[ attribute(1),  function_constant(enable_lowp_color) ]];
}

vertex Output vertexMain(VertexInput in [[stage_in]])
{
}
```

[stage_in] Arguments

`[stage_in]` arguments can be added/removed:

```
struct VertexInput
{
    float4 position [[ attribute(0) ]];
    float4 color [[ attribute(1),  function_constant(enable_color) ]];
    half4 lowp_color [[ attribute(1),  function_constant(enable_lowp_color) ]];
}

vertex Output vertexMain(VertexInput in [[stage_in]])
{
}
```

[stage_in] Arguments

`[stage_in]` arguments can be added/removed:

```
struct VertexInput
{
    float4 position [[ attribute(0) ]];
    float4 color [[ attribute(1),  function_constant(enable_color) ]];
    half4 lowp_color [[ attribute(1),  function_constant(enable_lowp_color) ]];
}

vertex Output vertexMain(VertexInput in [[stage_in]])
{
}
```

Limitations

Structs layouts cannot be modified

But buffer arguments with:

- Same buffer index
- Different types

```
constant bool useConstantsB = !useConstantsA;

vertex Output vertexMain(…,
      ConstantsA *constantsA [[ buffer(1), function_constant(useConstantsA) ]],
      ConstantsB *constantsB [[ buffer(1), function_constant(useConstantsB) ]],
```

Limitations

Structs layouts cannot be modified

But buffer arguments with:

- Same buffer index
- Different types

```
constant bool useConstantsB = !useConstantsA;

vertex Output vertexMain(…,
      ConstantsA *constantsA [[ buffer(1), function_constant(useConstantsA) ]],
      ConstantsB *constantsB [[ buffer(1), function_constant(useConstantsB) ]],
```

Limitations

Structs layouts cannot be modified

But buffer arguments with:

- Same buffer index
- Different types

```
constant bool useConstantsB = !useConstantsA;

vertex Output vertexMain(…,
      ConstantsA *constantsA [[ buffer(1), function_constant(useConstantsA) ]],
      ConstantsB *constantsB [[ buffer(1), function_constant(useConstantsB) ]],
```

Limitations

Structs layouts cannot be modified

But buffer arguments with:

- Same buffer index
- Different types

```
constant bool useConstantsB = !useConstantsA;

vertex Output vertexMain(…,
      ConstantsA *constantsA [[ buffer(1), function_constant(useConstantsA) ]],
      ConstantsB *constantsB [[ buffer(1), function_constant(useConstantsB) ]],
```

Summary

Create specialized functions at run time

- Avoids front end compilation (fast optimization phase instead)
- Compact storage in Metal library
- Ship IR, not source
- Unused code is eliminated for best performance

Function Resource Read-Writes

Overview

Function Buffer Read-Writes:

- Reading and writing to buffers
- Atomic operations

Function Texture Read-Writes

- Reading and writing to textures

Support

	iOS (A9)	macOS
Function Buffer Read-Writes	✓	✓
Function Texture Read-Writes	✗	✓

Function Buffer Read-Writes

Writing to buffers from fragment functions

Atomic operations in vertex and fragment functions

Use cases:

- Order-independent transparency
- Building light lists for tiles
- Debugging

Example

Writing visible fragment positions to a buffer:

```
struct FragmentInput {
   float4 position [[position]];
};

fragment void outputPositions(FragmentInput in [[stage_in]],
      device float2 *outputBuffer [[buffer(0)]],
      device atomic_uint *counter [[buffer(1)]])
{
   uint index = atomic_fetch_add_explicit(counter, 1, memory_order_relaxed);
   outputBuffer[index] = in.position.xy;
}
```

Example

Writing visible fragment positions to a buffer:

```
struct FragmentInput {
   float4 position [[position]];
};

fragment void outputPositions(FragmentInput in [[stage_in]],
      device float2 *outputBuffer [[buffer(0)]],
      device atomic_uint *counter [[buffer(1)]])
{
   uint index = atomic_fetch_add_explicit(counter, 1, memory_order_relaxed);
   outputBuffer[index] = in.position.xy;
}
```

Example

Writing visible fragment positions to a buffer:

```
struct FragmentInput {
   float4 position [[position]];
};

fragment void outputPositions(FragmentInput in [[stage_in]],
      device float2 *outputBuffer [[buffer(0)]],
      device atomic_uint *counter [[buffer(1)]])
{
   uint index = atomic_fetch_add_explicit(counter, 1, memory_order_relaxed);
   outputBuffer[index] = in.position.xy;
}
```

Example

Writing visible fragment positions to a buffer:

```
struct FragmentInput {
   float4 position [[position]];
};

fragment void outputPositions(FragmentInput in [[stage_in]],
      device float2 *outputBuffer [[buffer(0)]],
      device atomic_uint *counter [[buffer(1)]])
{
   uint index = atomic_fetch_add_explicit(counter, 1, memory_order_relaxed);
   outputBuffer[index] = in.position.xy;
}
```

Example

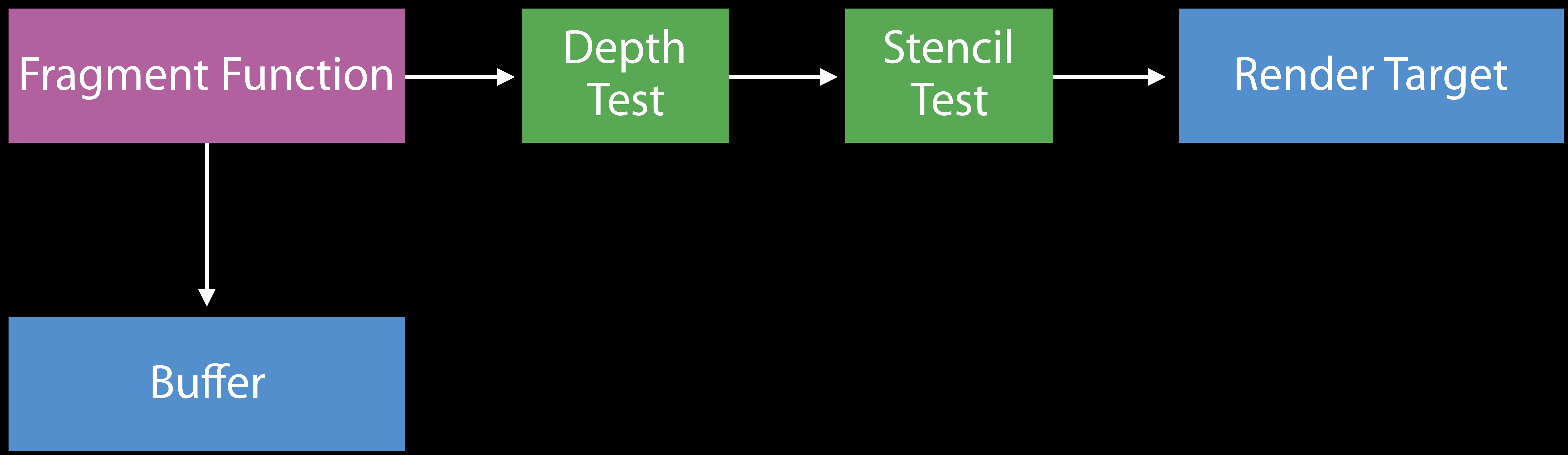
Writing visible fragment positions to a buffer:

```
struct FragmentInput {
   float4 position [[position]];
};

fragment void outputPositions(FragmentInput in [[stage_in]],
      device float2 *outputBuffer [[buffer(0)]],
      device atomic_uint *counter [[buffer(1)]])
{
   uint index = atomic_fetch_add_explicit(counter, 1, memory_order_relaxed);
   outputBuffer[index] = in.position.xy;
}
```

Depth/Stencil Tests

Depth/stencil tests after function execution

Disables early Z optimizations: Impacts performance!

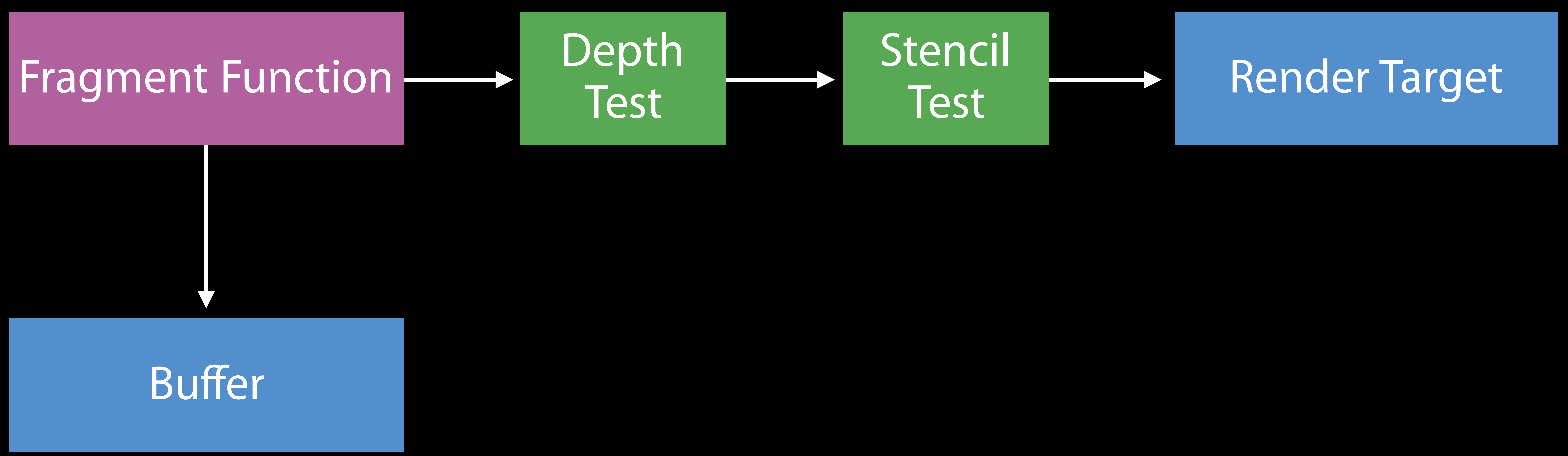


Depth/Stencil Tests

Depth/stencil tests after function execution

Disables early Z optimizations: Impacts performance!

New function qualifier `[[early_fragment_tests]]`

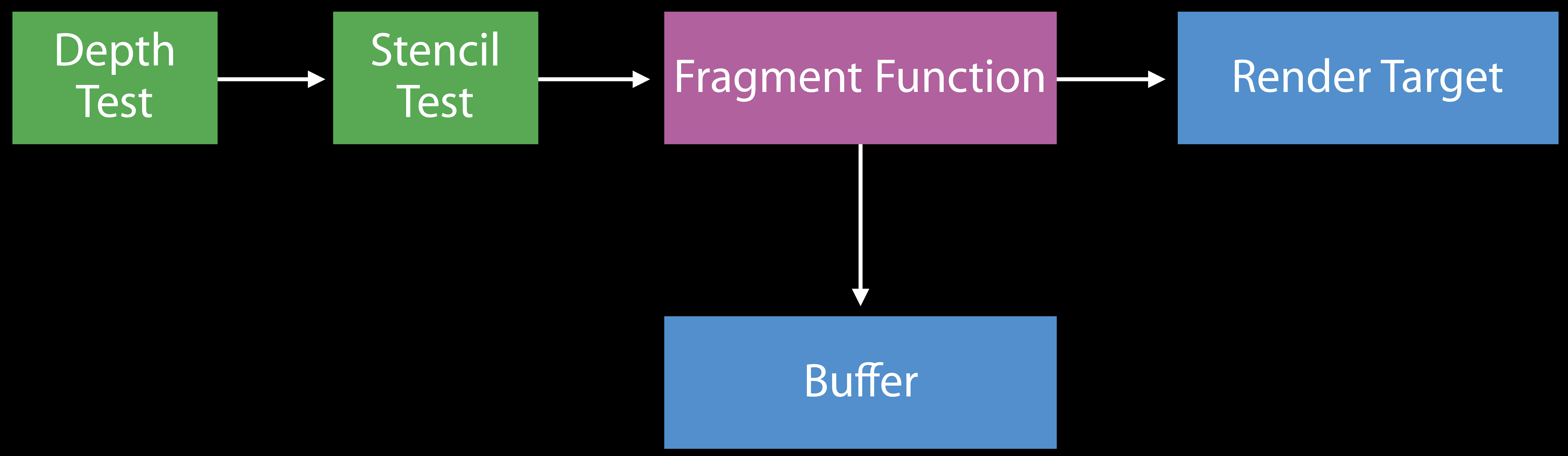


Depth/Stencil Tests

Depth/stencil tests after function execution

Disables early Z optimizations: Impacts performance!

New function qualifier `[[early_fragment_tests]]`

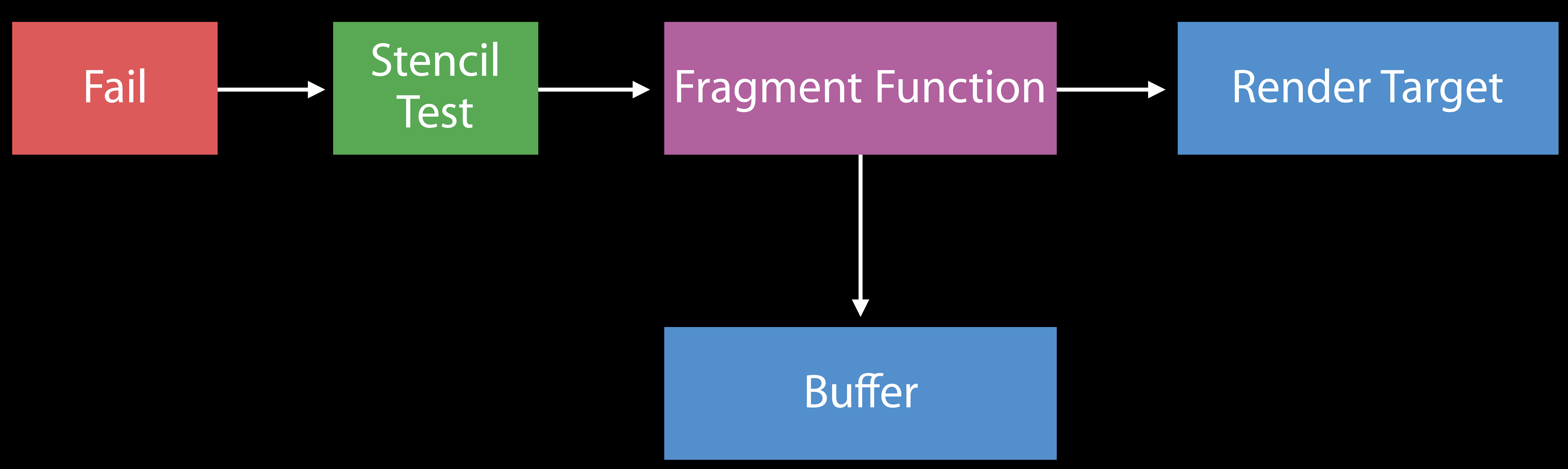


Depth/Stencil Tests

Depth/stencil tests after function execution

Disables early Z optimizations: Impacts performance!

New function qualifier `[[early_fragment_tests]]`

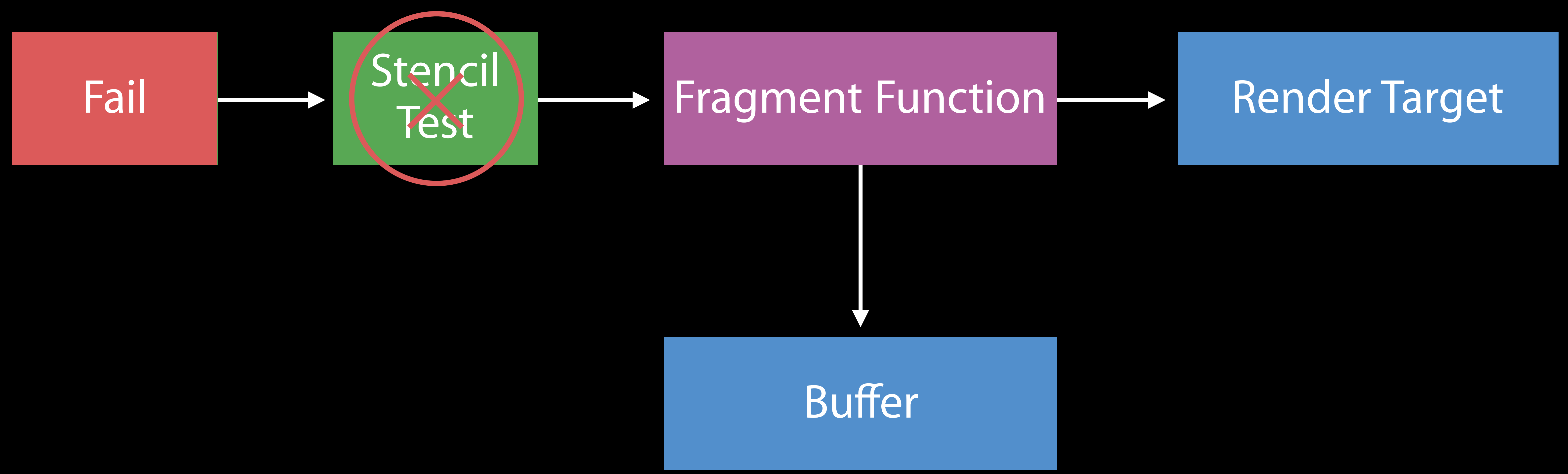


Depth/Stencil Tests

Depth/stencil tests after function execution

Disables early Z optimizations: Impacts performance!

New function qualifier `[[early_fragment_tests]]`

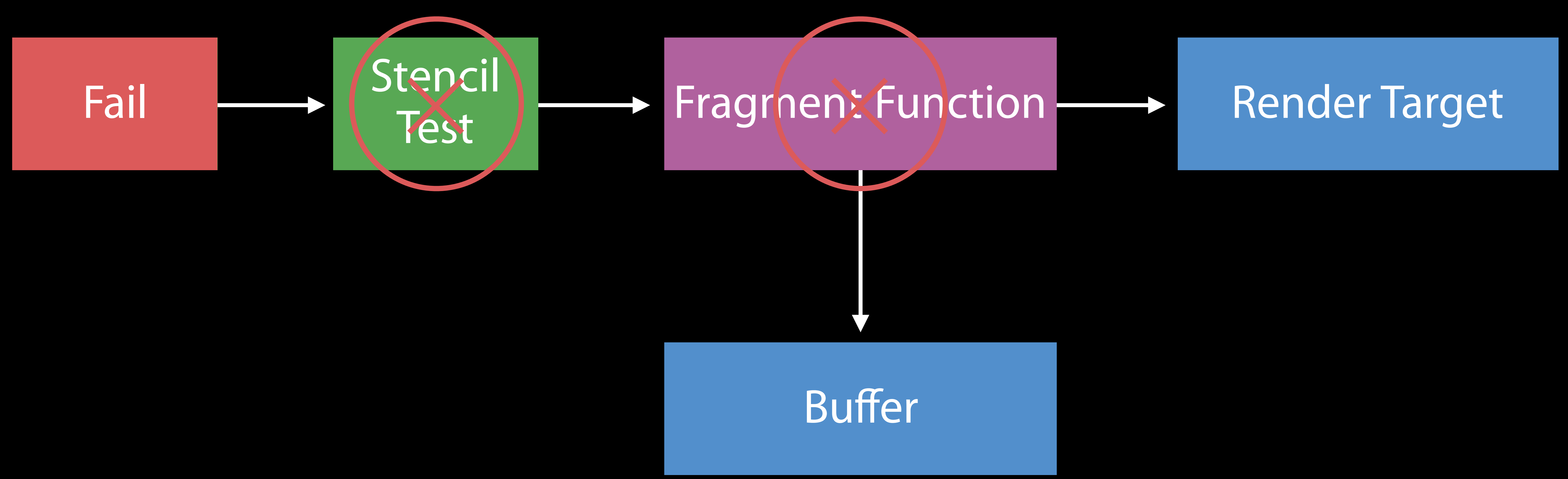


Depth/Stencil Tests

Depth/stencil tests after function execution

Disables early Z optimizations: Impacts performance!

New function qualifier `[[early_fragment_tests]]`

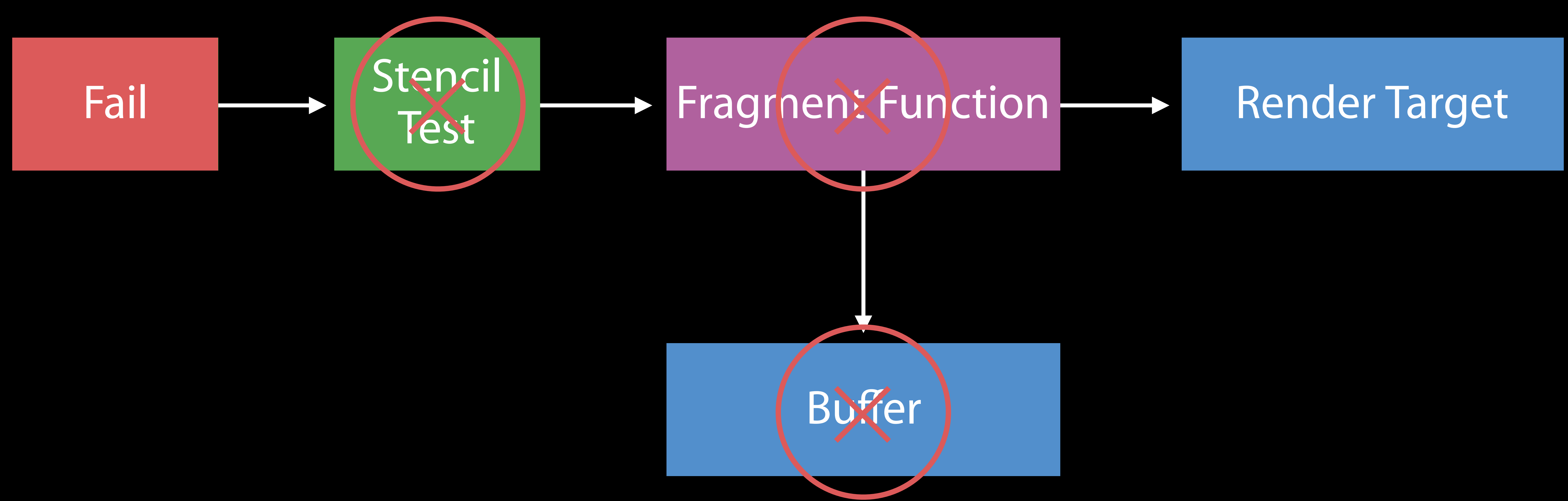


Depth/Stencil Tests

Depth/stencil tests after function execution

Disables early Z optimizations: Impacts performance!

New function qualifier `[[early_fragment_tests]]`



Correct Implementation

Final function with early fragment tests:

```
struct FragmentInput {
   float4 position [[position]];
};

[[early_fragment_tests]] fragment void outputPositions(FragmentInput in [[stage_in]],
      device float2 *outputBuffer [[buffer(0)]],
      device atomic_uint &counter [[buffer(1)]])
{
   uint index = atomic_fetch_add_explicit(counter, 1, memory_order_relaxed);
   outputBuffer[index] = in.position.xy;
}
```

Correct Implementation

Final function with early fragment tests:

```
struct FragmentInput {
   float4 position [[position]];
};

[[early_fragment_tests]] fragment void outputPositions(FragmentInput in [[stage_in]],
      device float2 *outputBuffer [[buffer(0)]],
      device atomic_uint &counter [[buffer(1)]])
{
   uint index = atomic_fetch_add_explicit(counter, 1, memory_order_relaxed);
   outputBuffer[index] = in.position.xy;
}
```

Function Texture Read-Writes

Writing to textures from vertex and fragment functions

Reading and writing to the same texture in a function

Use case:

- Save memory for post processing effects (single texture)

Writing to Textures

Writing to textures from vertex and fragment functions:

```
fragment float4 function(texture2d<float, access::write> tex [[texture(0)]]) {
...
   tex.write(color, texCoord, 0);
...
}
```

Writing to Textures

Writing to textures from vertex and fragment functions:

```
fragment float4 function(texture2d<float, access::write> tex [[texture(0)]]) {
...
   tex.write(color, texCoord, 0);
...
}
```

Writing to Textures

Writing to textures from vertex and fragment functions:

```
fragment float4 function(texture2d<float, access::write> tex [[texture(0)]]) {
...
   tex.write(color, texCoord, 0);
...
}
```

Read-Write Textures

Both reading and writing to a single texture

Supported formats: R32Float, R32Uint, R32Int

```
fragment float4 function(texture2d<float, access::read_write> tex [[texture(0)]]) {
...
   float4 color = tex.read(texCoord);
   ...
   tex.write(color, texCoord, 0);
...
}
```

Read-Write Textures

Both reading and writing to a single texture

Supported formats: R32Float, R32Uint, R32Int

```
fragment float4 function(texture2d<float, access::read_write> tex [[texture(0)]]) {
...
   float4 color = tex.read(texCoord);
   ...
   tex.write(color, texCoord, 0);
...
}
```

Read-Write Textures

Both reading and writing to a single texture

Supported formats: R32Float, R32Uint, R32Int

```
fragment float4 function(texture2d<float, access::read_write> tex [[texture(0)]]) {
...
   float4 color = tex.read(texCoord);
   ...
   tex.write(color, texCoord, 0);
...
}
```

Read-Write Textures

Both reading and writing to a single texture

Supported formats: R32Float, R32Uint, R32Int

```
fragment float4 function(texture2d<float, access::read_write> tex [[texture(0)]]) {
...
   float4 color = tex.read(texCoord);
   ...
   tex.write(color, texCoord, 0);
...
}
```

Texture Fence

Reading after writing to a pixel requires a texture fence

Writing after reading does not require a texture fence

```
tex.write(color, texCoord);
// fence to ensure write becomes visible to later reads by the thread
tex.fence();
// read will see previous write
color = tex.read(texCoord);
```

Texture Fence

Reading after writing to a pixel requires a texture fence

Writing after reading does not require a texture fence

```
tex.write(color, texCoord);
// fence to ensure write becomes visible to later reads by the thread
tex.fence();
// read will see previous write
color = tex.read(texCoord);
```

Texture Fence

Reading after writing to a pixel requires a texture fence

Writing after reading does not require a texture fence

```
tex.write(color, texCoord);
// fence to ensure write becomes visible to later reads by the thread
tex.fence();
// read will see previous write
color = tex.read(texCoord);
```

Texture Fence

Reading after writing to a pixel requires a texture fence

Writing after reading does not require a texture fence

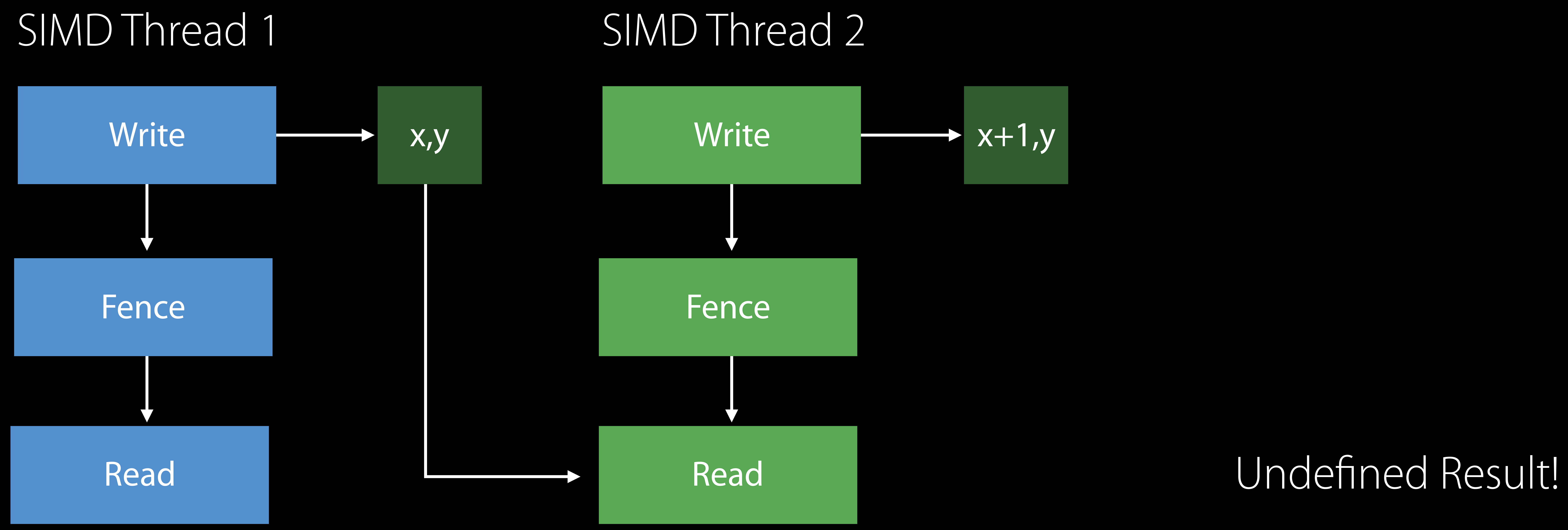
```
tex.write(color, texCoord);
// fence to ensure write becomes visible to later reads by the thread
tex.fence();
// read will see previous write
color = tex.read(texCoord);
```

Texture Fence

Texture fence on single SIMD thread

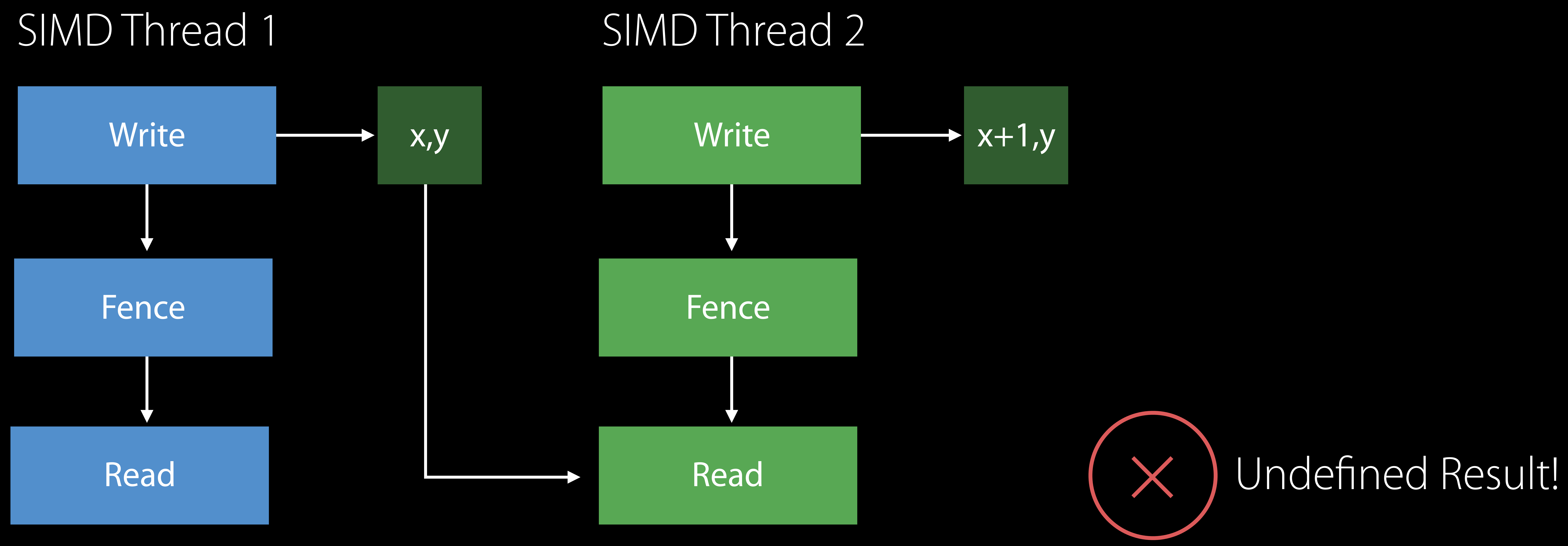
Texture Fence

Texture fence on single SIMD thread



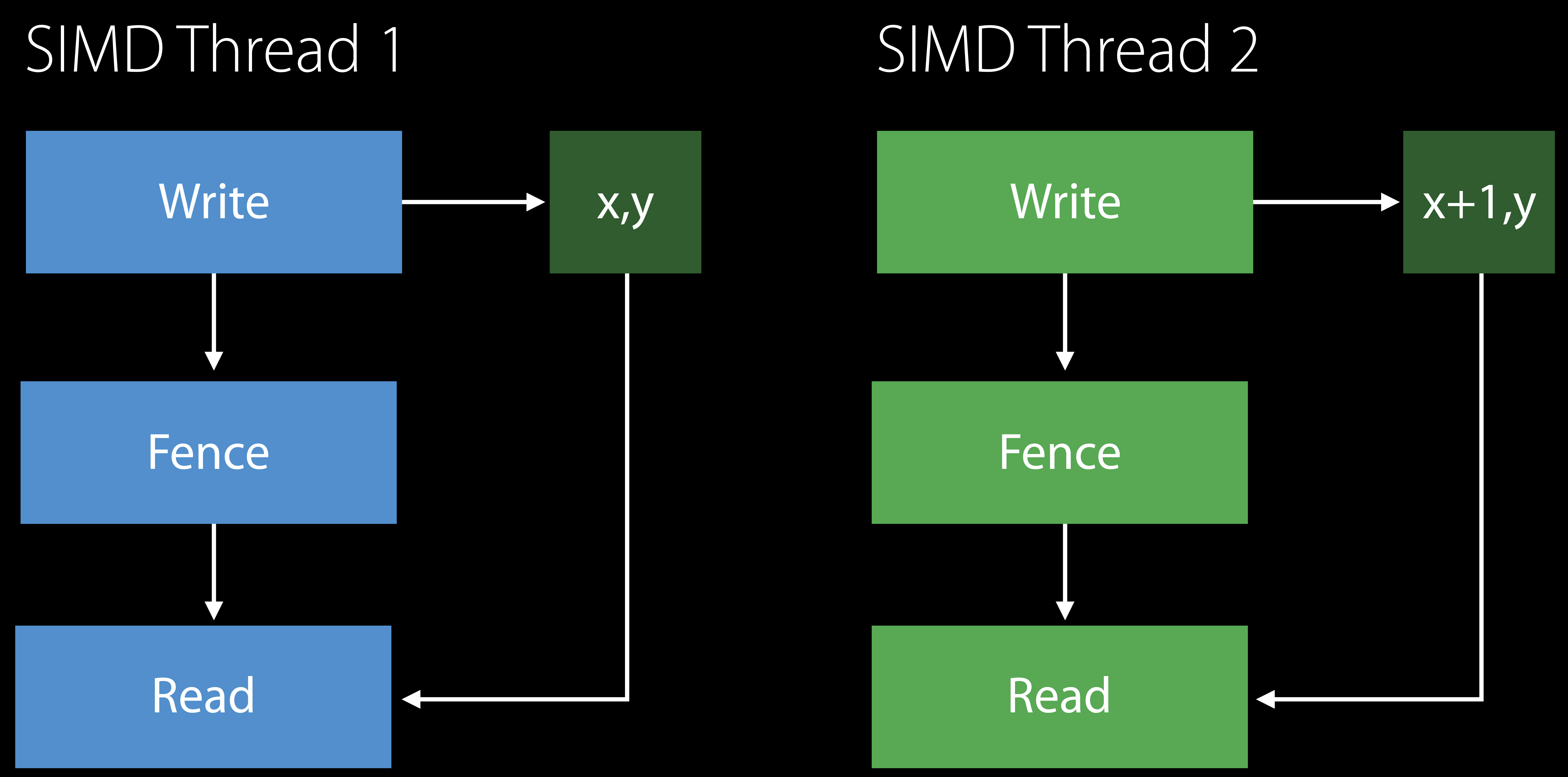
Texture Fence

Texture fence on single SIMD thread



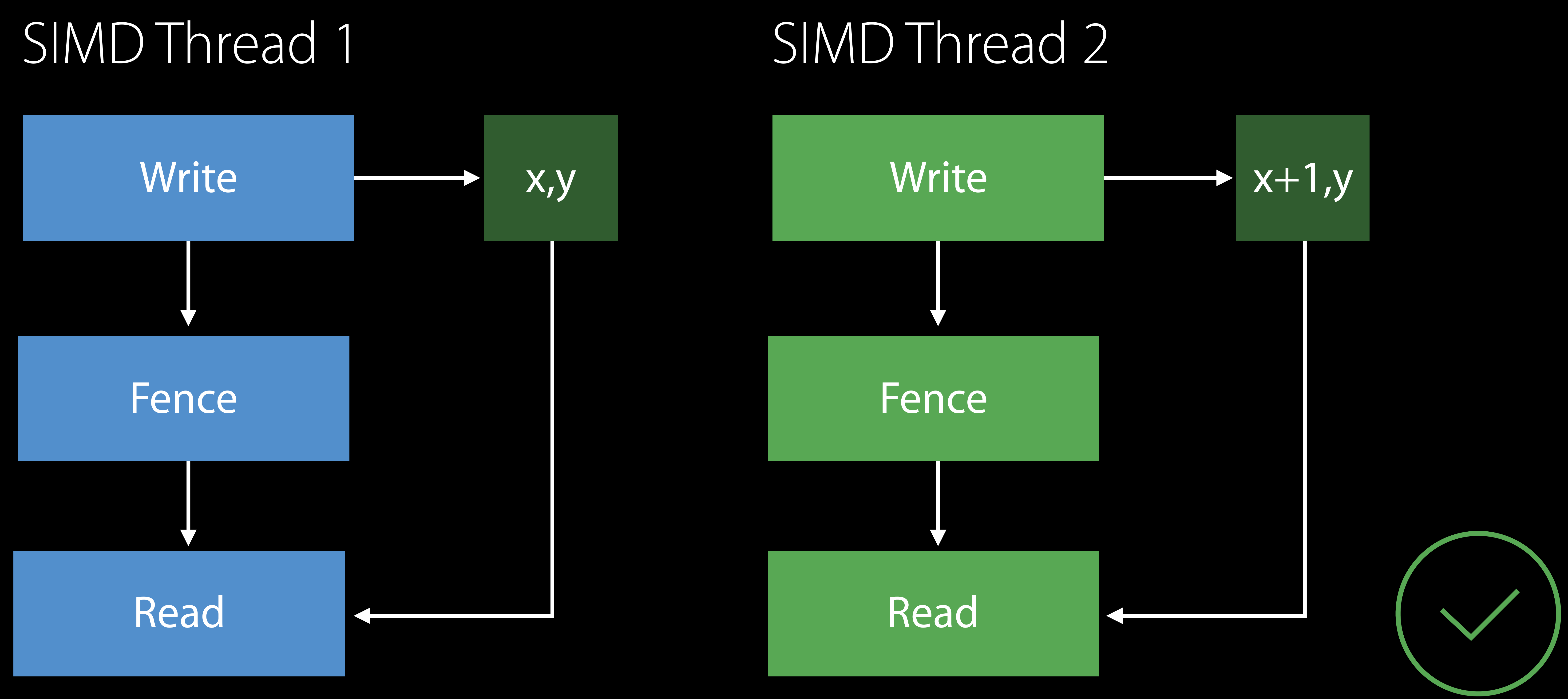
Texture Fence

Texture fence on single SIMD thread



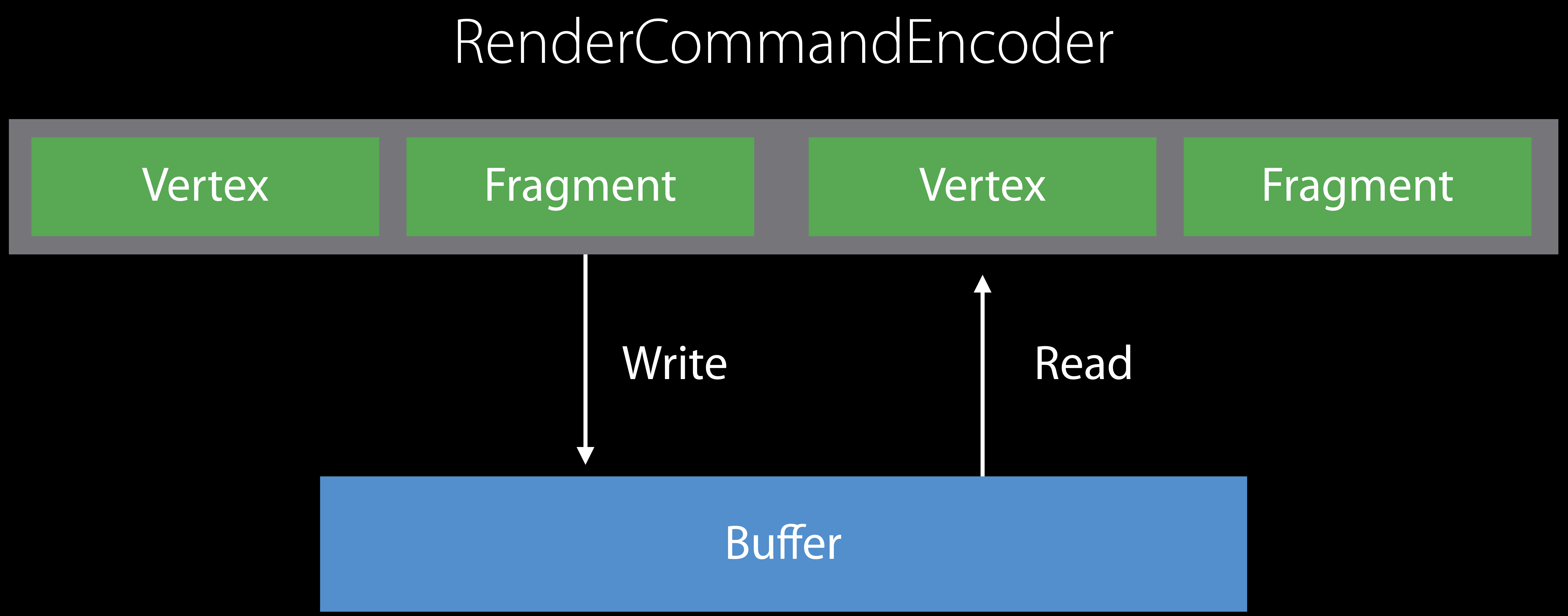
Texture Fence

Texture fence on single SIMD thread



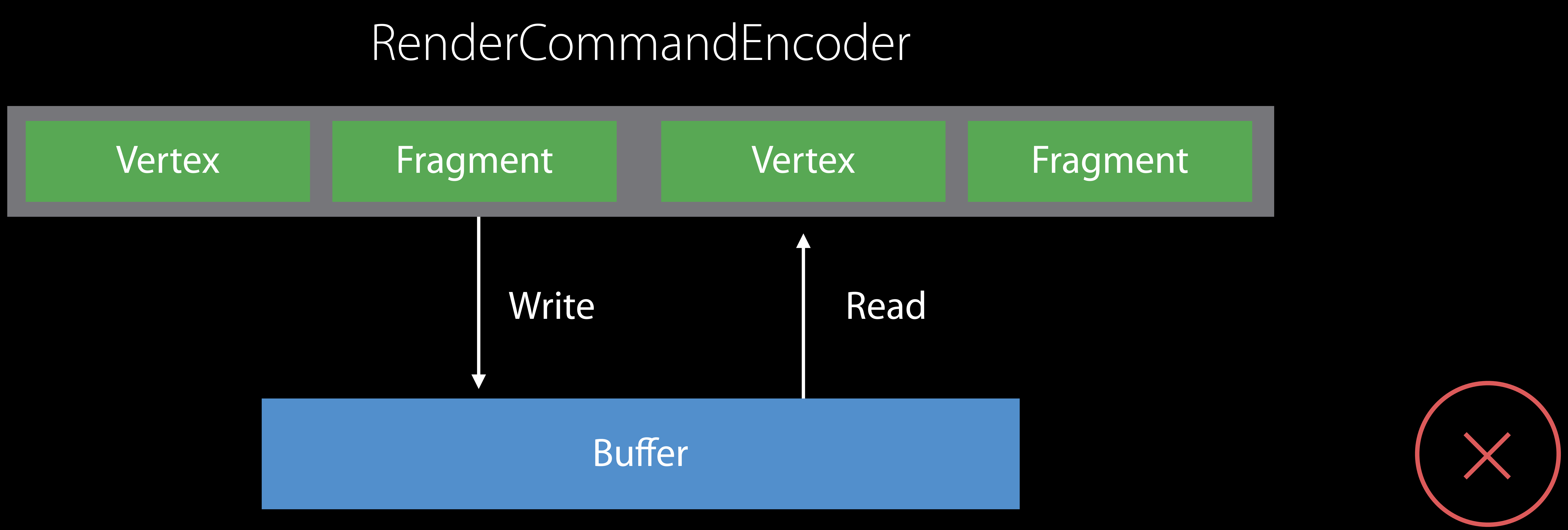
Reading

Reading data in the same RenderCommandEncoder is undefined:



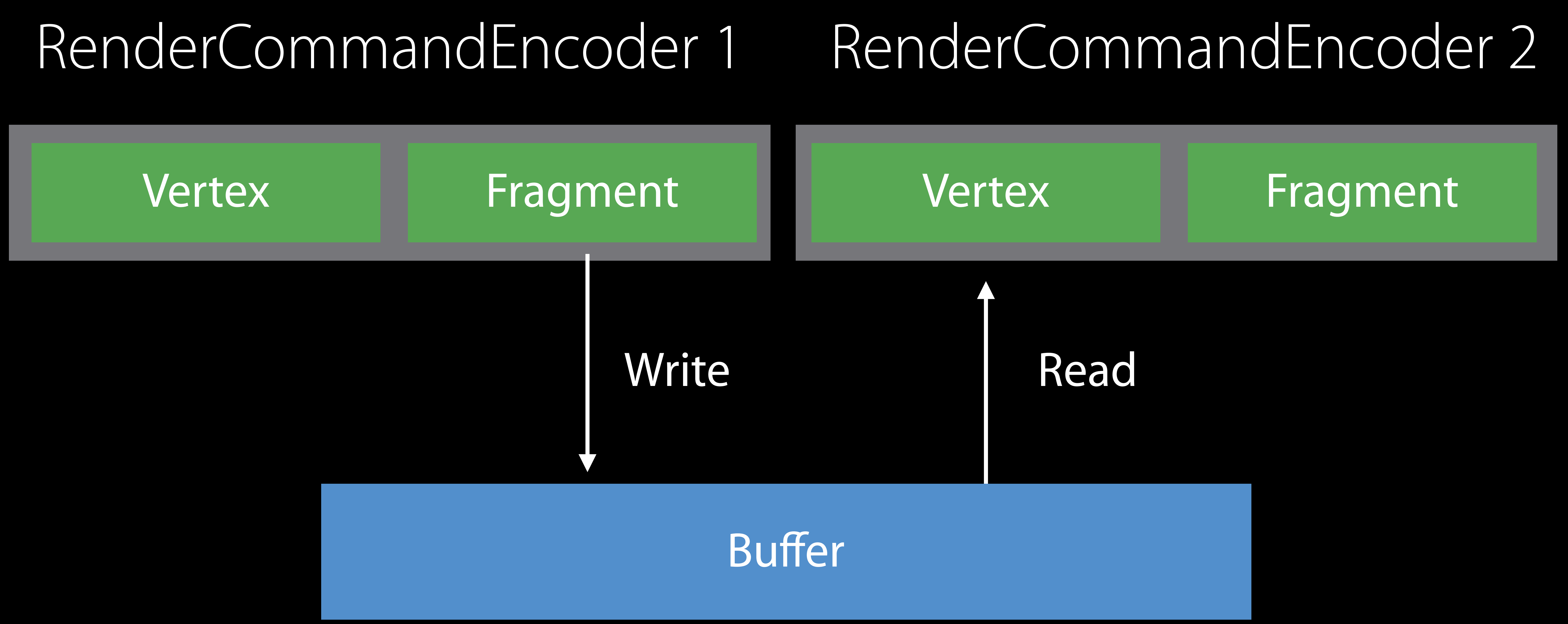
Reading

Reading data in the same RenderCommandEncoder is undefined:



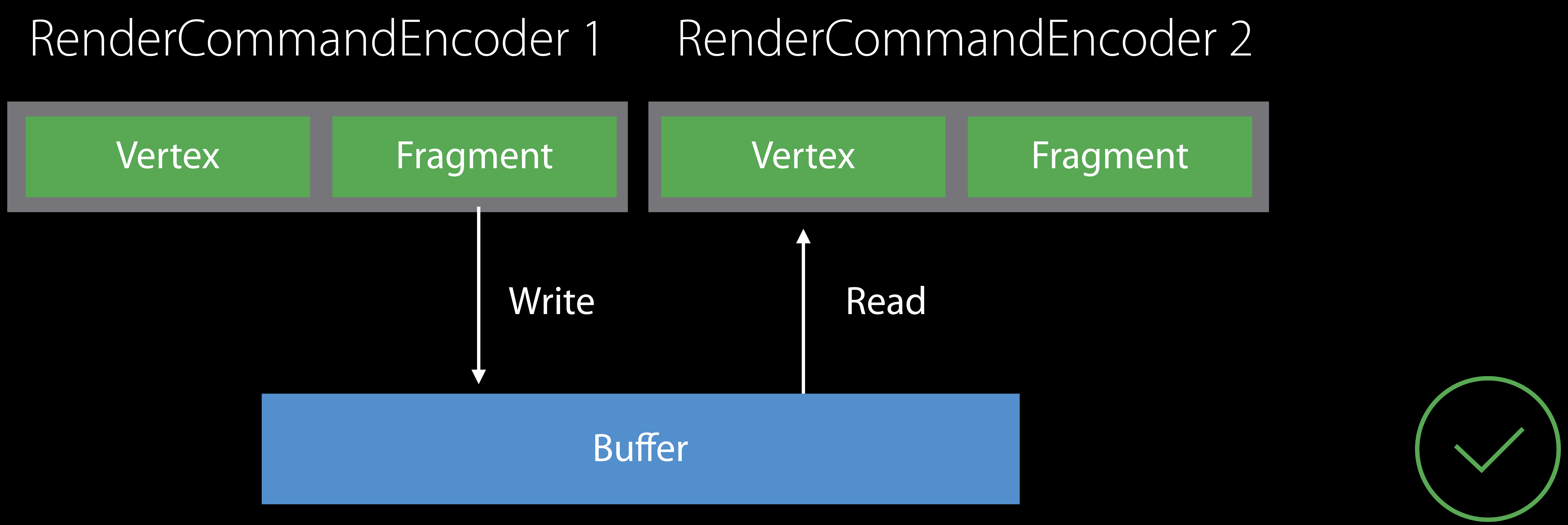
Reading

Reading data in the same RenderCommandEncoder is undefined:



Reading

Reading data in the same RenderCommandEncoder is undefined:



Summary

Function buffer read-writes

Function texture read-writes

Early fragment test

Texture fence

Reading requires new RenderCommandEncoder

Wide Color

Taking advantage of the wide-gamut display

Dan Omachi GPU Software Engineer

Color Management

Caring about color

Color Management

Caring about color

Managing color allows content to look the same regardless of the display

- Rendering should always reflect the artist’s intention

Color Management

Caring about color

Managing color allows content to look the same regardless of the display

- Rendering should always reflect the artist's intention

Apps using high-level frameworks can render in any colorspace

Color Management

Caring about color

Managing color allows content to look the same regardless of the display

- Rendering should always reflect the artist’s intention

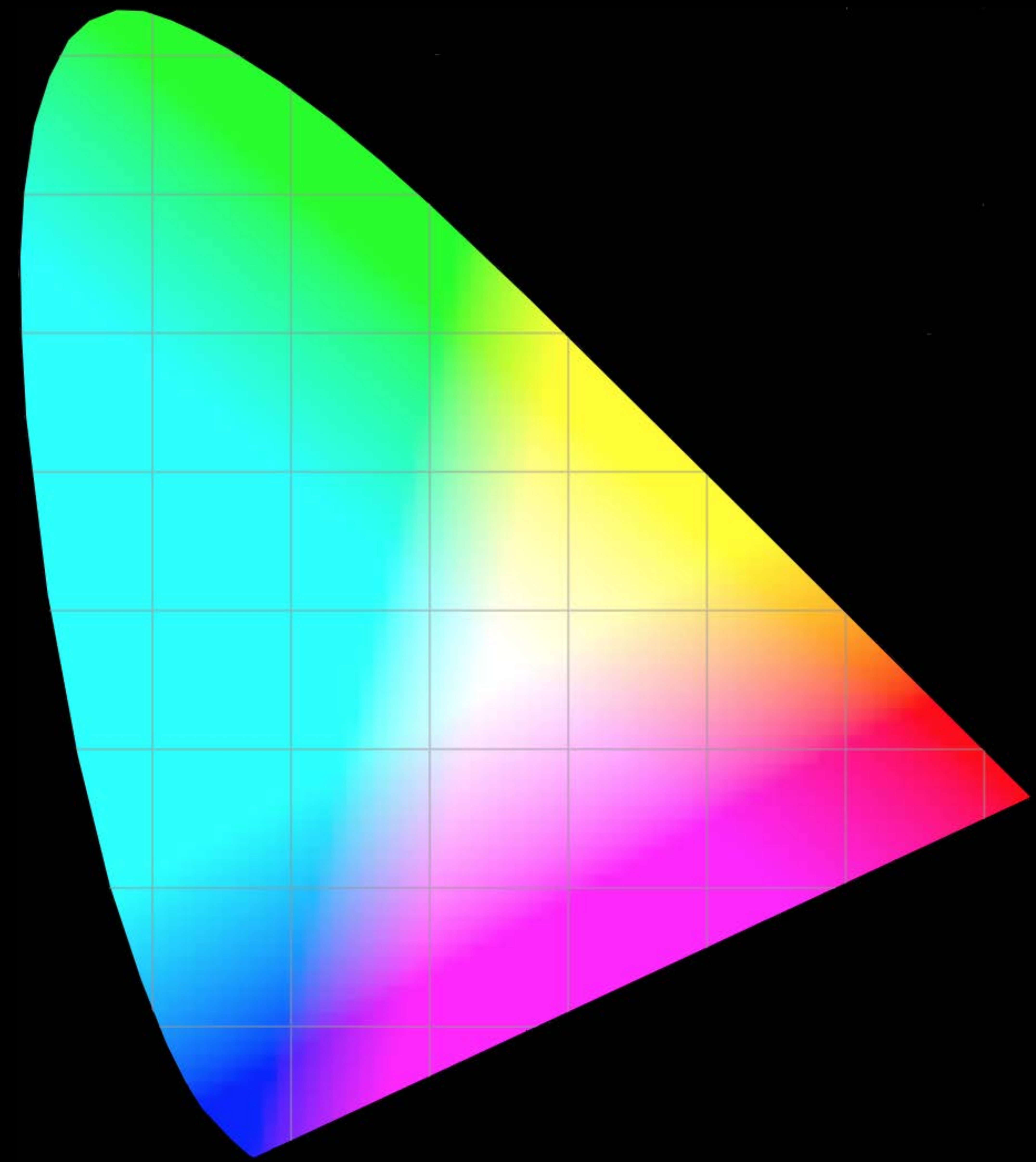
Apps using high-level frameworks can render in any colorspace

Metal is a low-level API

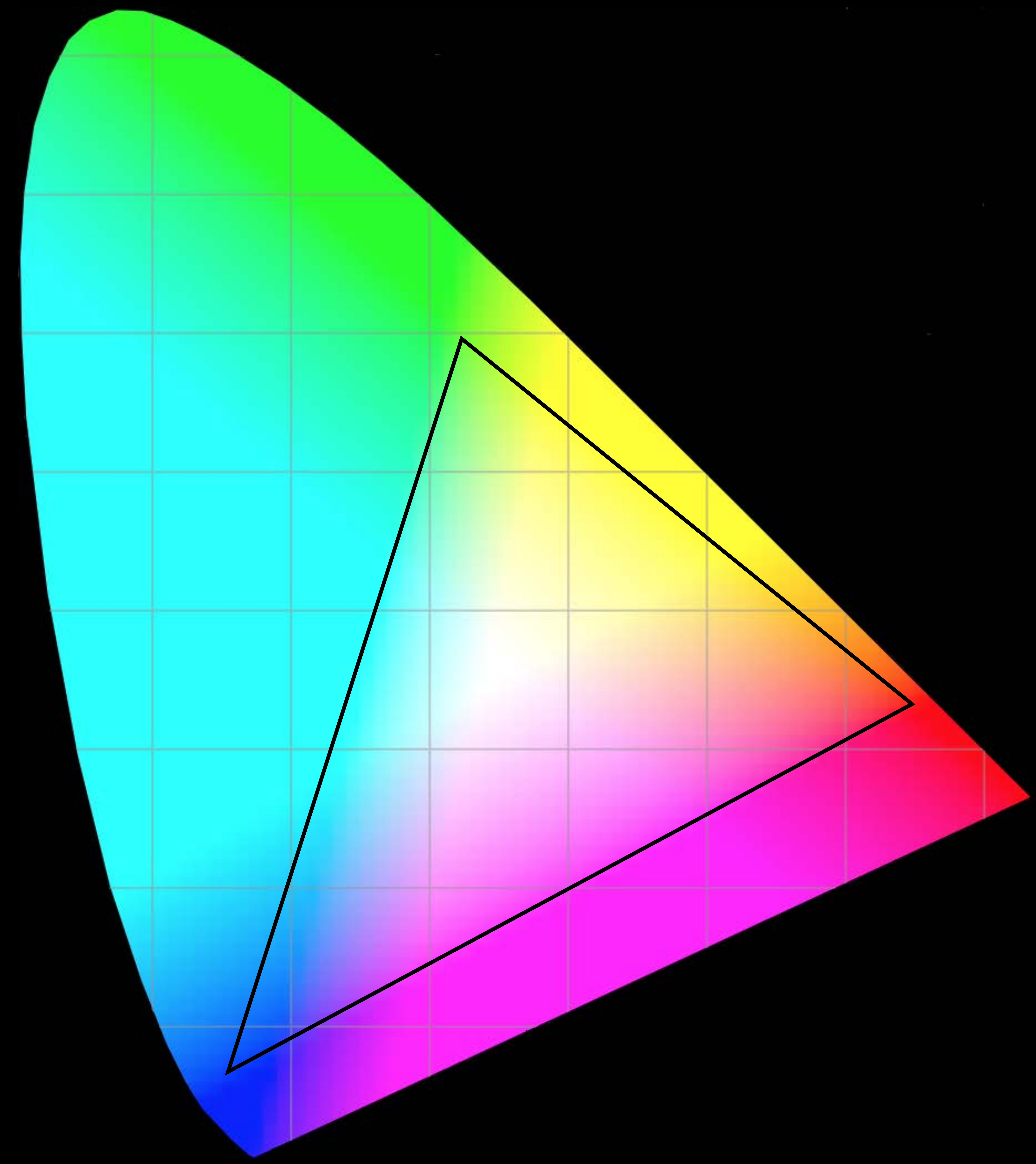
- More consideration required

Why now?

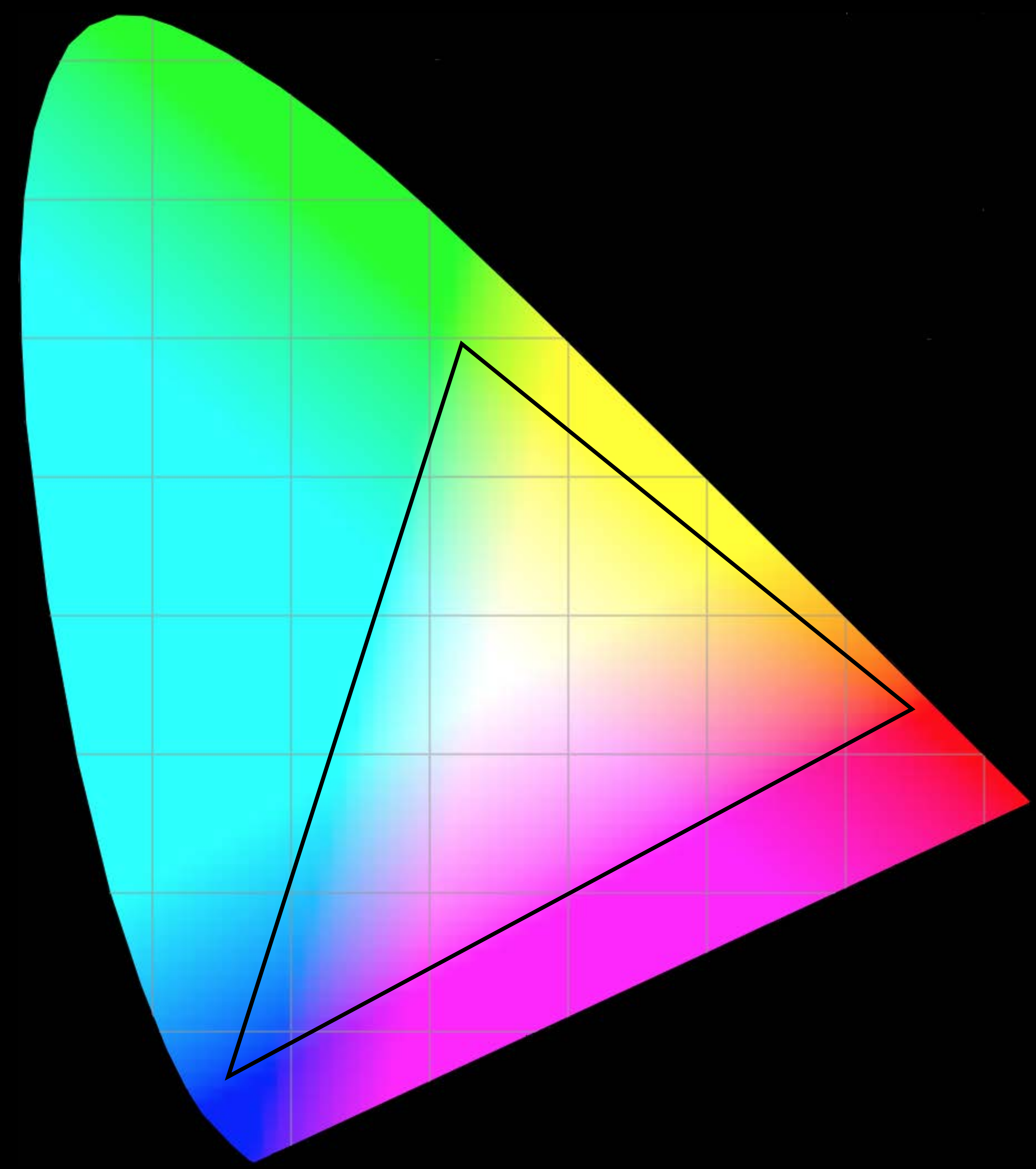
Colorspaces



Colorspaces

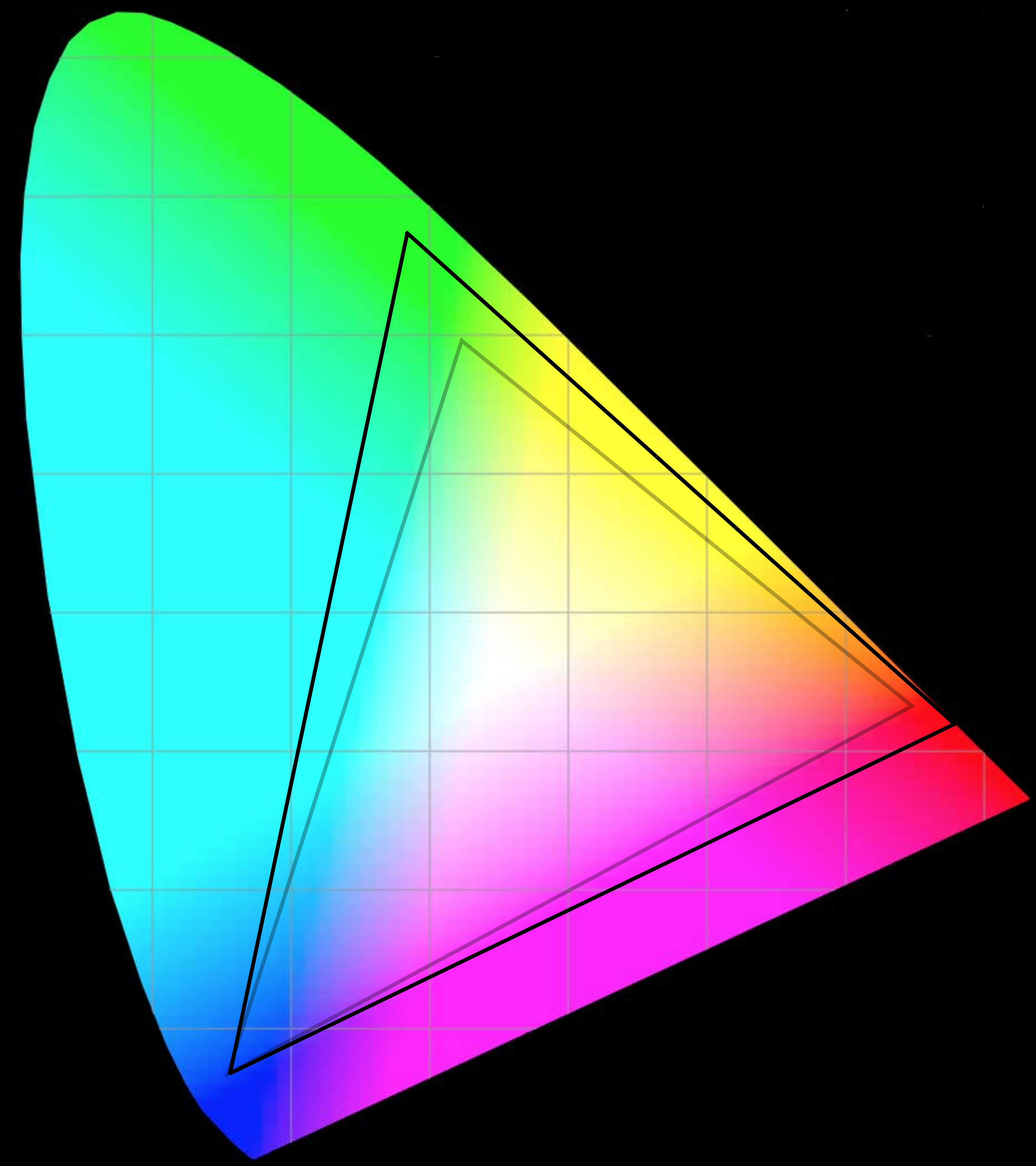


Colorspaces



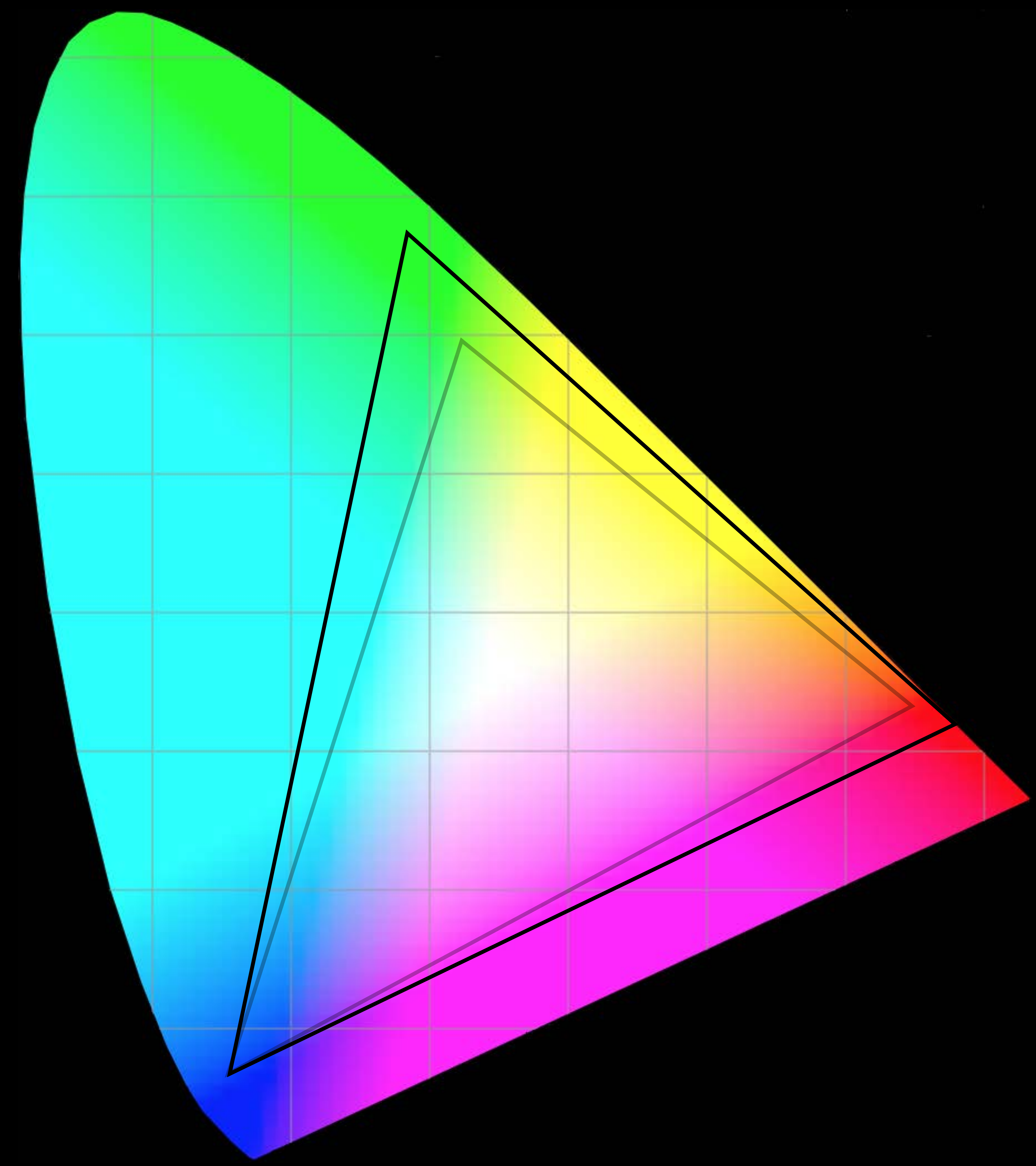
Colorspaces

NEW



Colorspaces

NEW



Color Management on macOS

Color Management on macOS

Render in any colorspace

Color Management on macOS

Render in any colorspace

High-level frameworks perform automatic color matching

- macOS performs conversion to to display colorspace

Color Management on macOS

Render in any colorspace

High-level frameworks perform automatic color matching

- macOS performs conversion to to display colorspace

Metal views, by default, not color managed

- Color match skipped during compositing
    - Offers increased performance

Color Management on macOS

Consequences of ignoring colorspace

Display interprets colors in its own colorspace

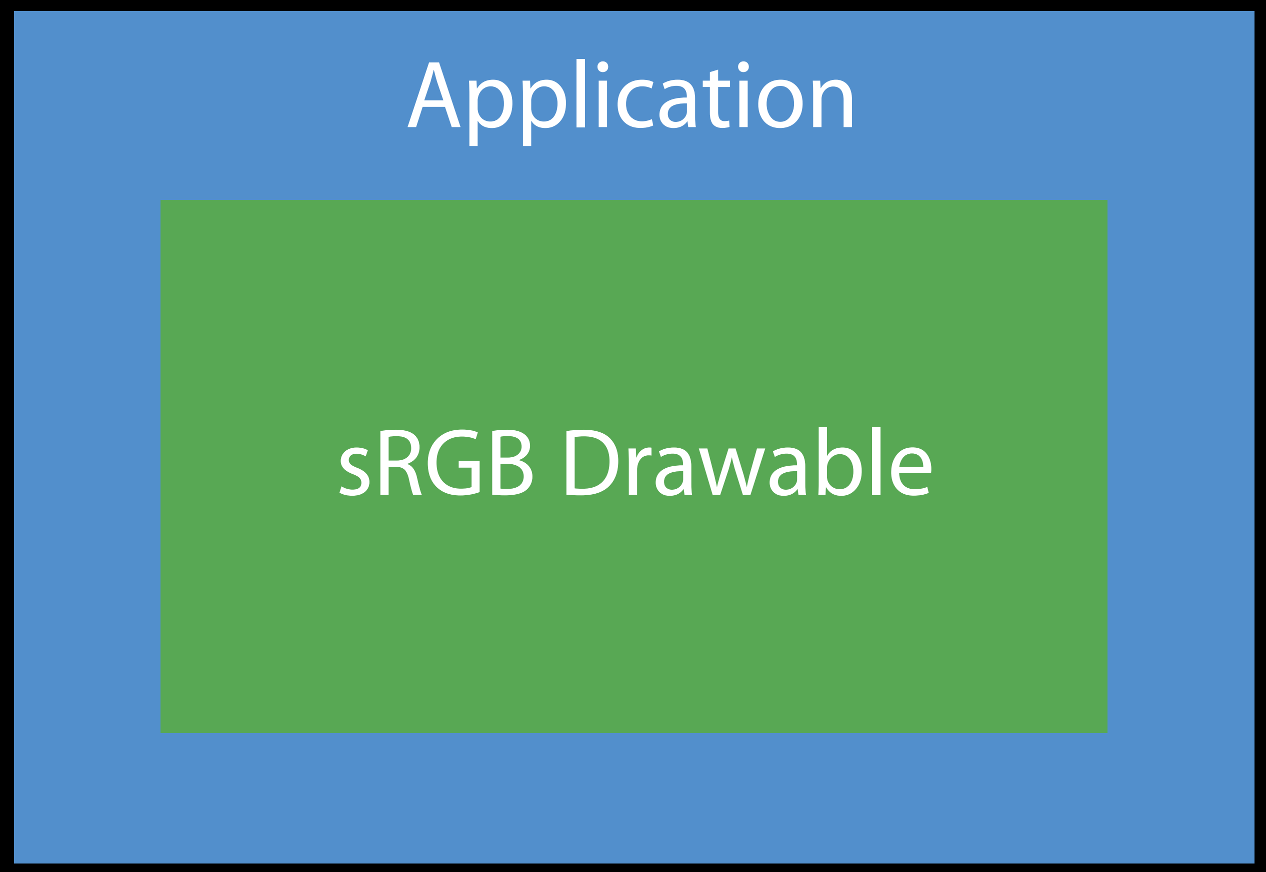
- Rendering inconsistent
- On a P3 display, sRGB colors will be more saturated

Color Management on macOS

Consequences of ignoring colorspace

Display interprets colors in its own colorspace

- Rendering inconsistent
- On a P3 display, sRGB colors will be more saturated

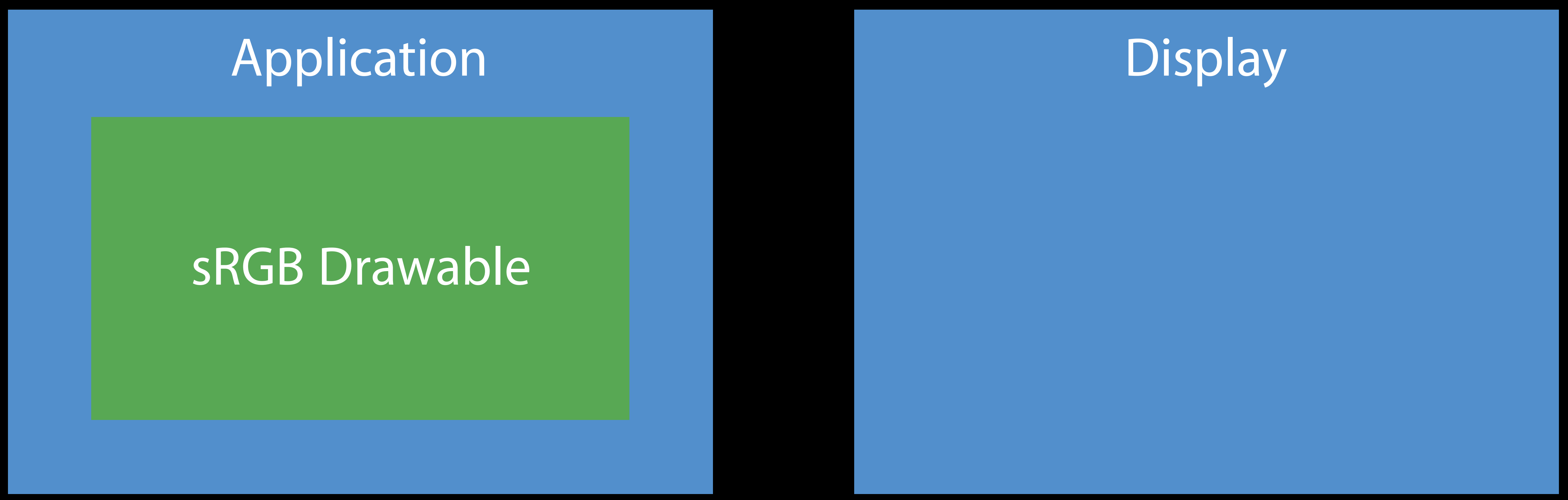


Color Management on macOS

Consequences of ignoring colorspace

Display interprets colors in its own colorspace

- Rendering inconsistent
- On a P3 display, sRGB colors will be more saturated

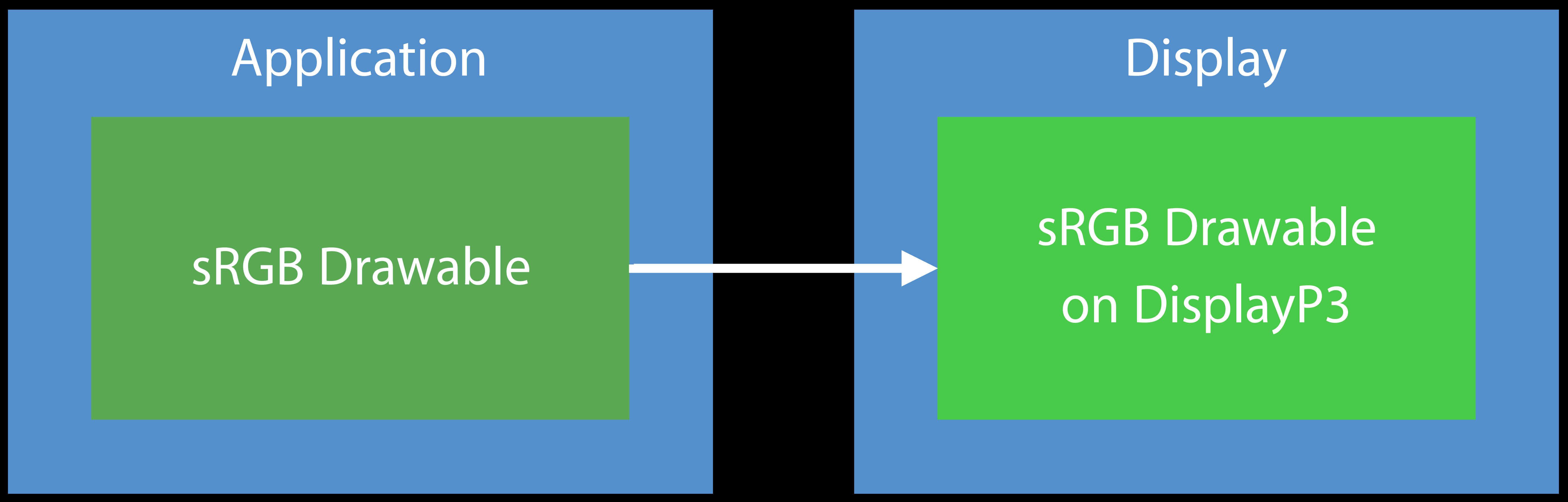


Color Management on macOS

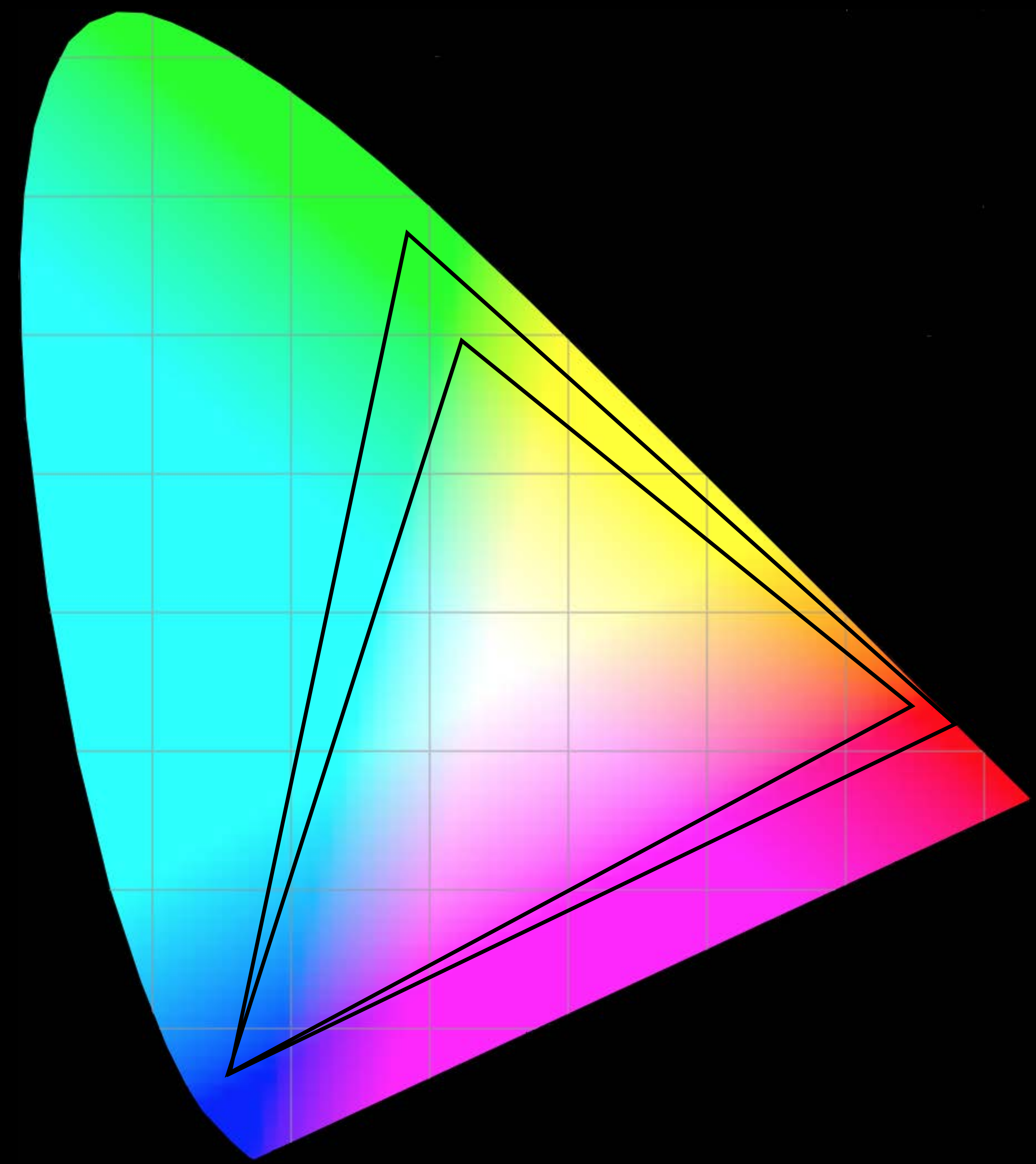
Consequences of ignoring colorspace

Display interprets colors in its own colorspace

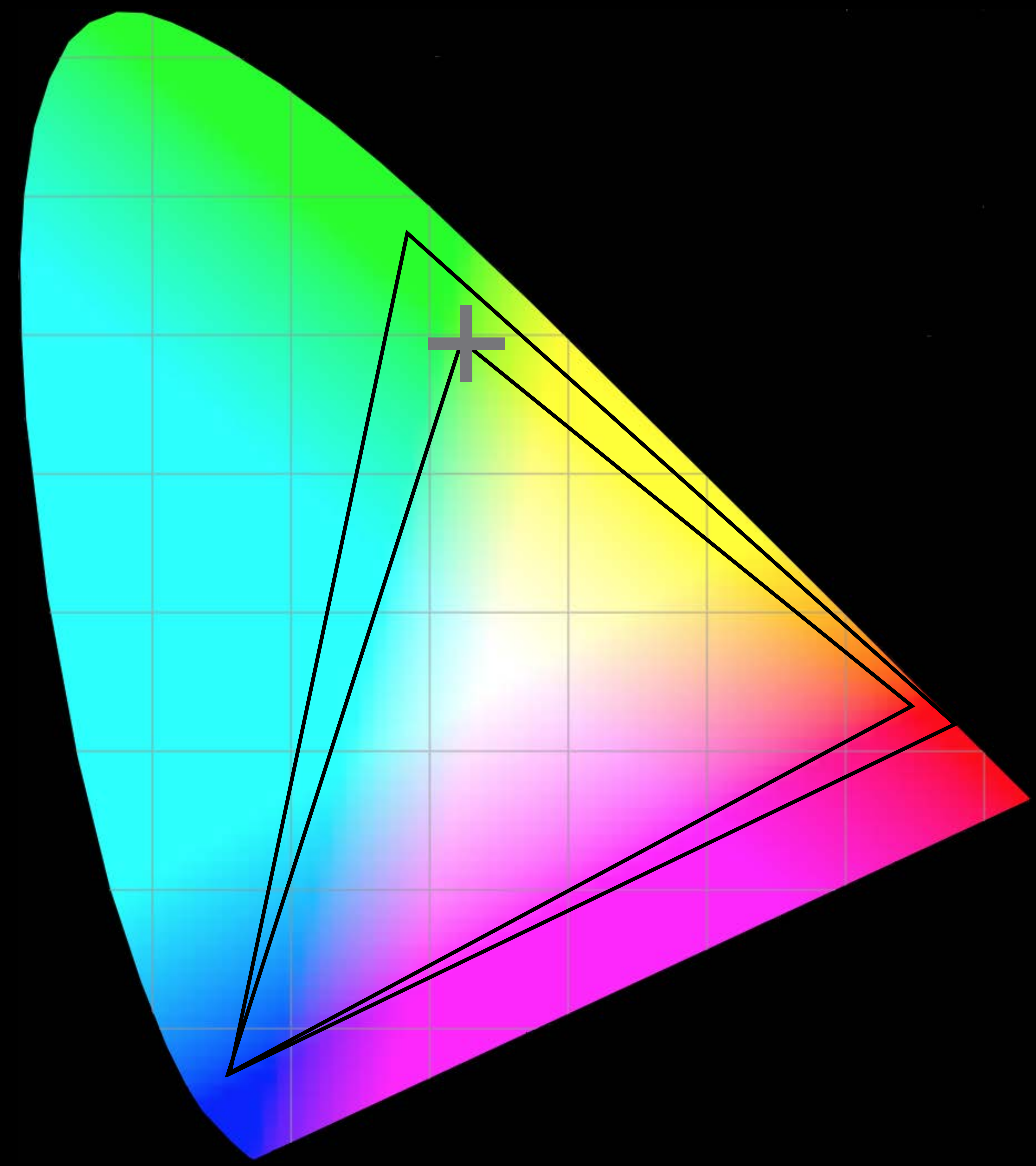
- Rendering inconsistent
- On a P3 display, sRGB colors will be more saturated



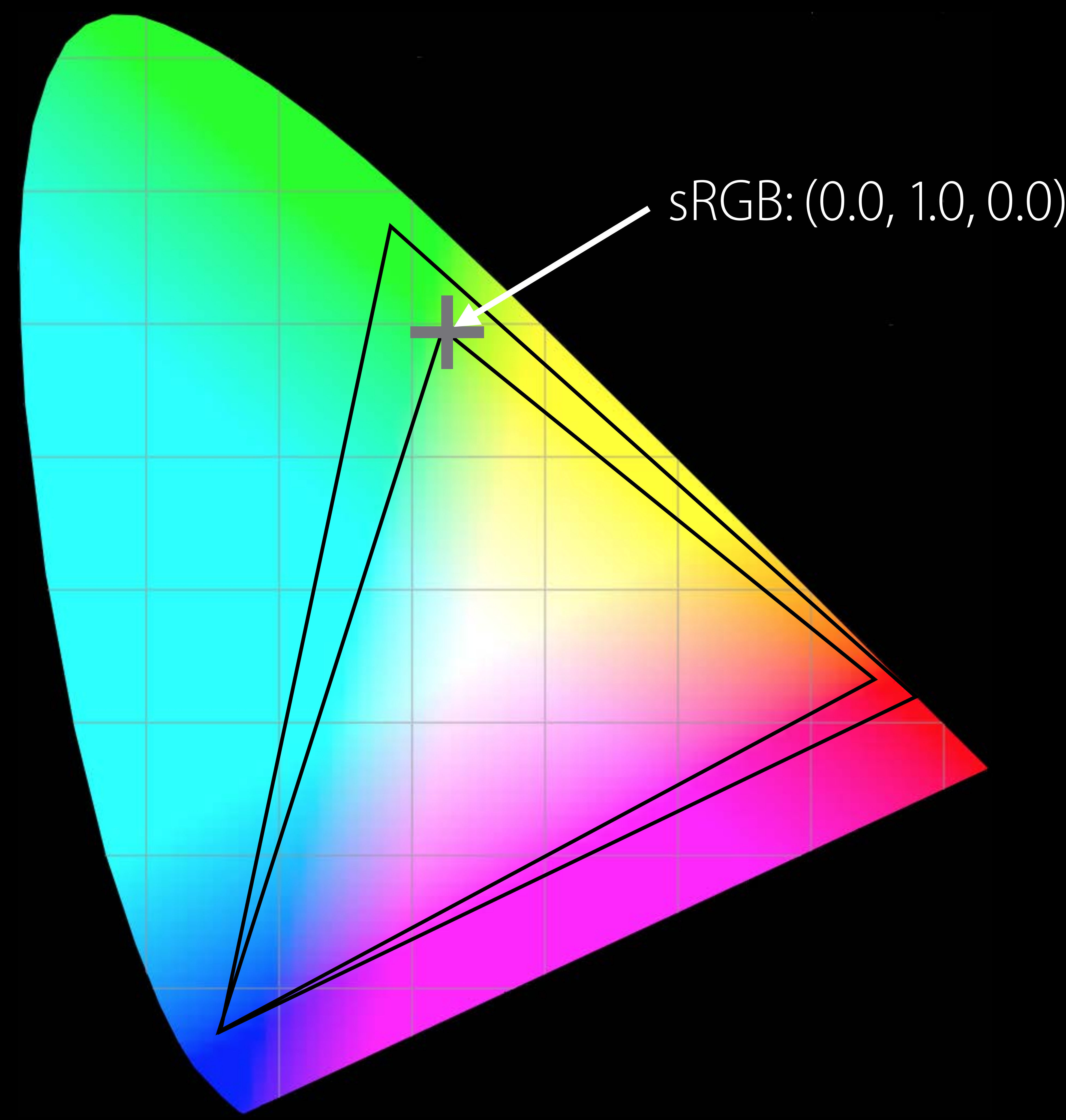
Color Management on macOS



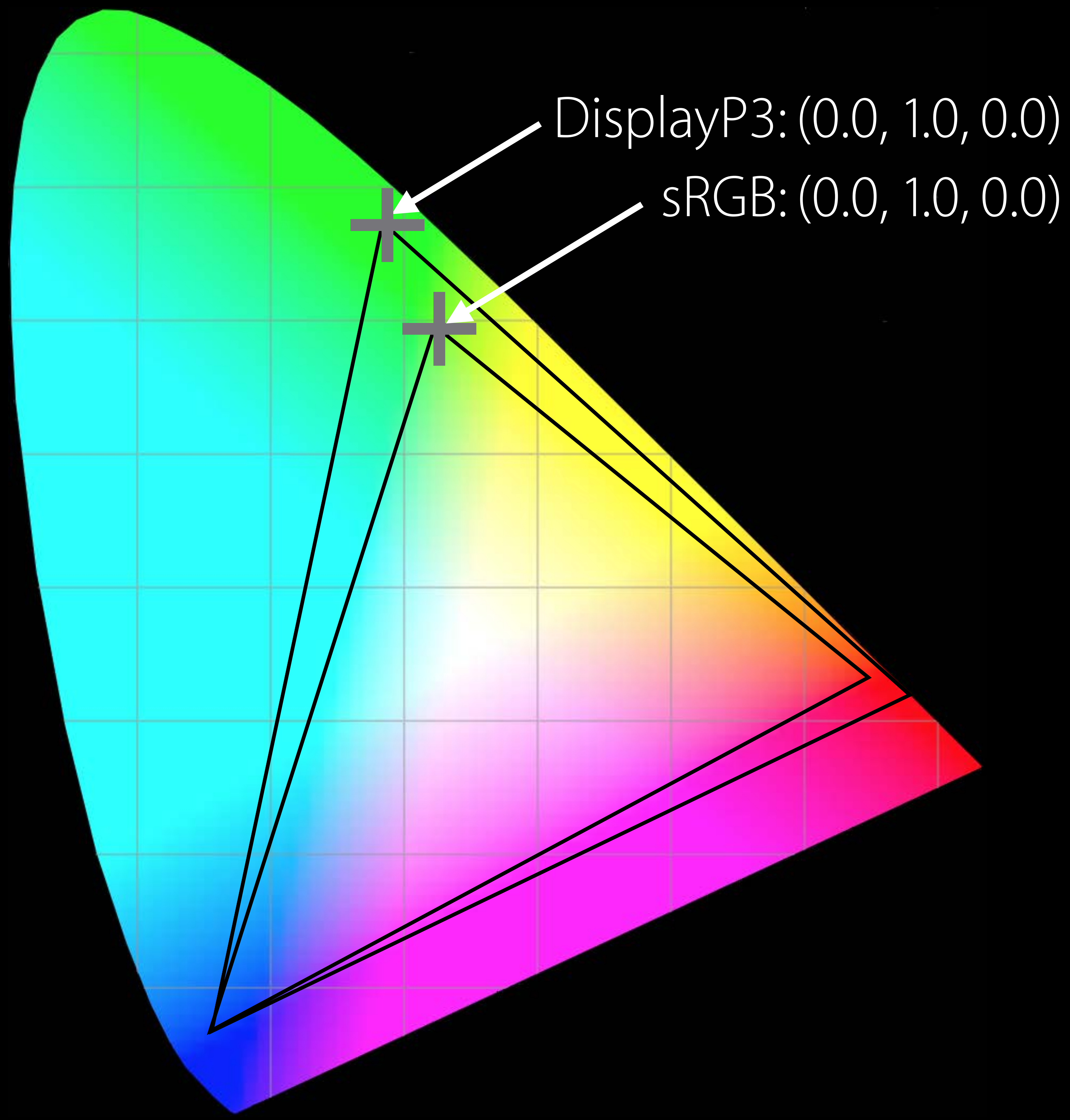
Color Management on macOS



Color Management on macOS



Color Management on macOS



Color Management on macOS

Enable automatic color management

- Set `colorspace` property in either `NSWindow` or `CAMetalLayer`

OS performs color match

- Part of window server's normal compositing pass

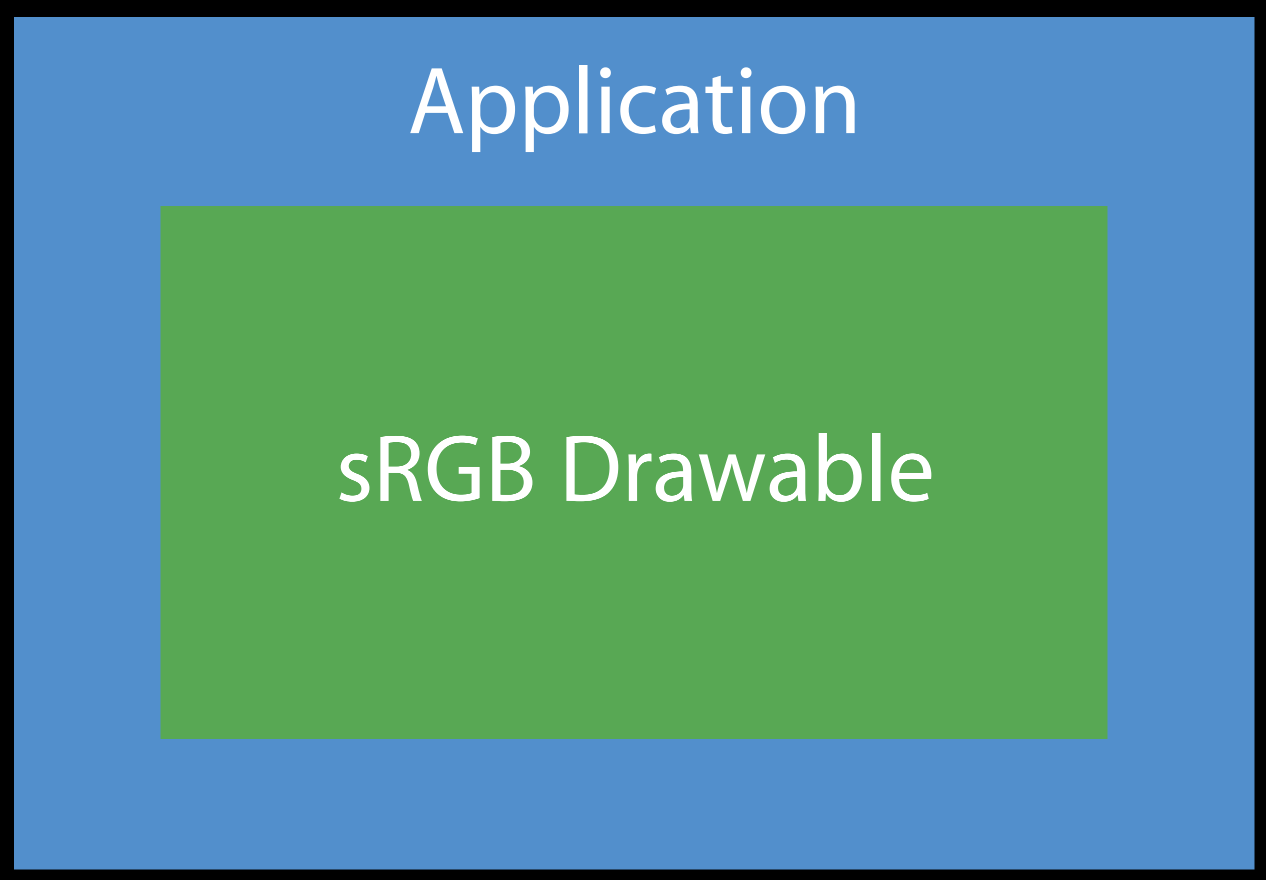
Color Management on macOS

Enable automatic color management

- Set `colorspace` property in either `NSWindow` or `CAMetalLayer`

OS performs color match

- Part of window server's normal compositing pass



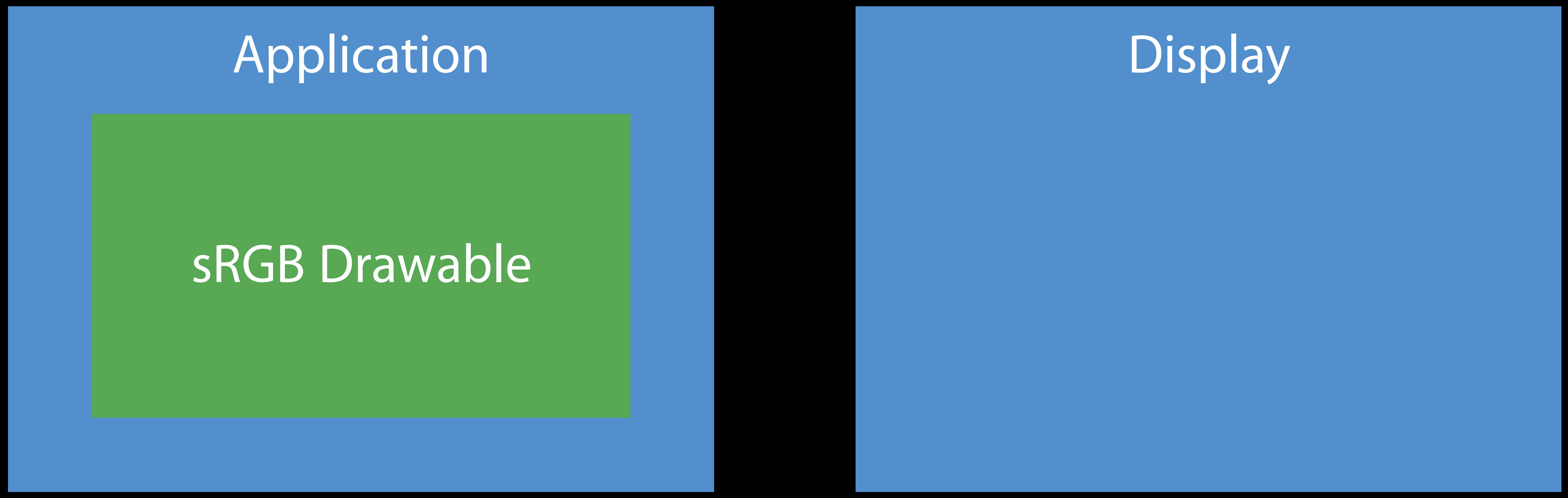
Color Management on macOS

Enable automatic color management

- Set `colorspace` property in either `NSWindow` or `CAMetalLayer`

OS performs color match

- Part of window server's normal compositing pass



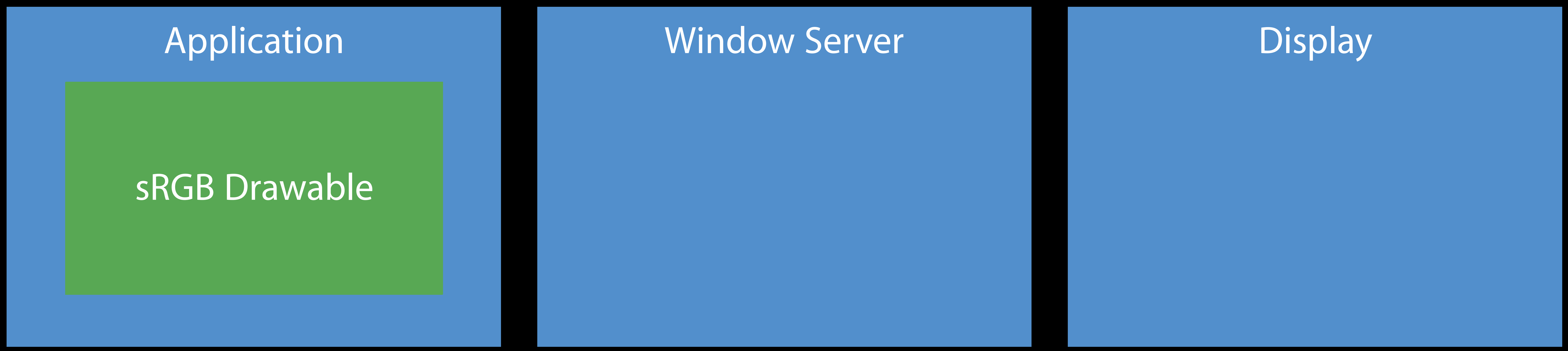
Color Management on macOS

Enable automatic color management

- Set `colorspace` property in either `NSWindow` or `CAMetalLayer`

OS performs color match

- Part of window server's normal compositing pass



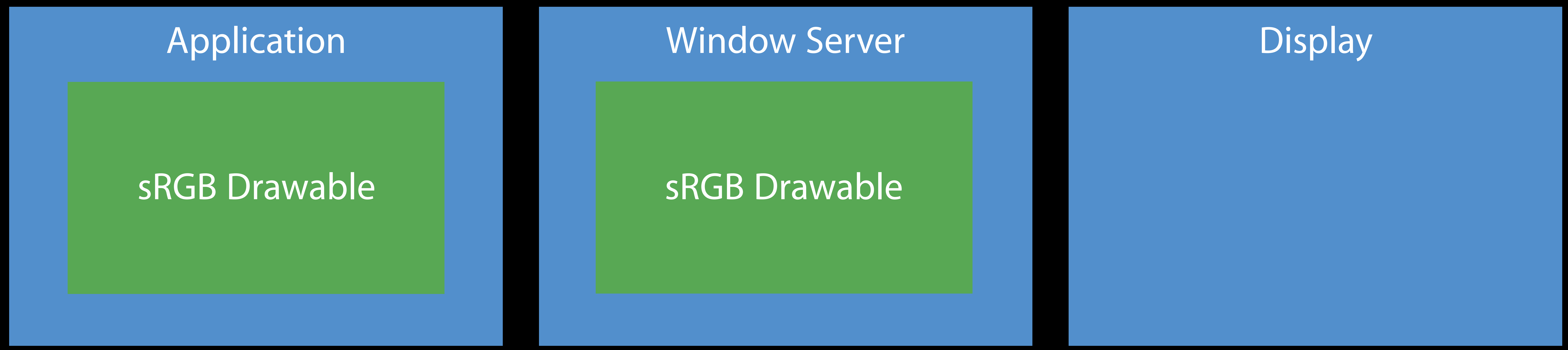
Color Management on macOS

Enable automatic color management

- Set `colorspace` property in either `NSWindow` or `CAMetalLayer`

OS performs color match

- Part of window server's normal compositing pass



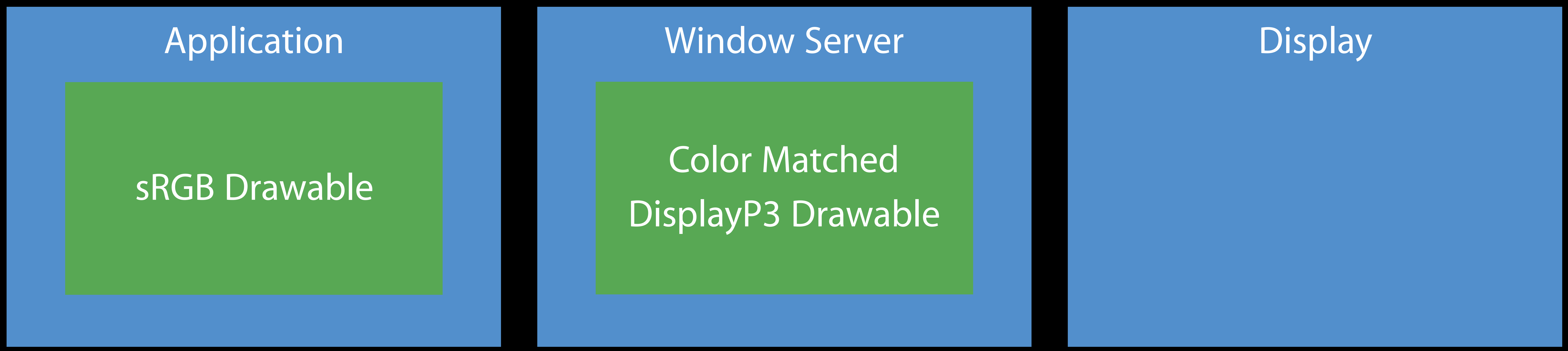
Color Management on macOS

Enable automatic color management

- Set `colorspace` property in either `NSWindow` or `CAMetalLayer`

OS performs color match

- Part of window server's normal compositing pass



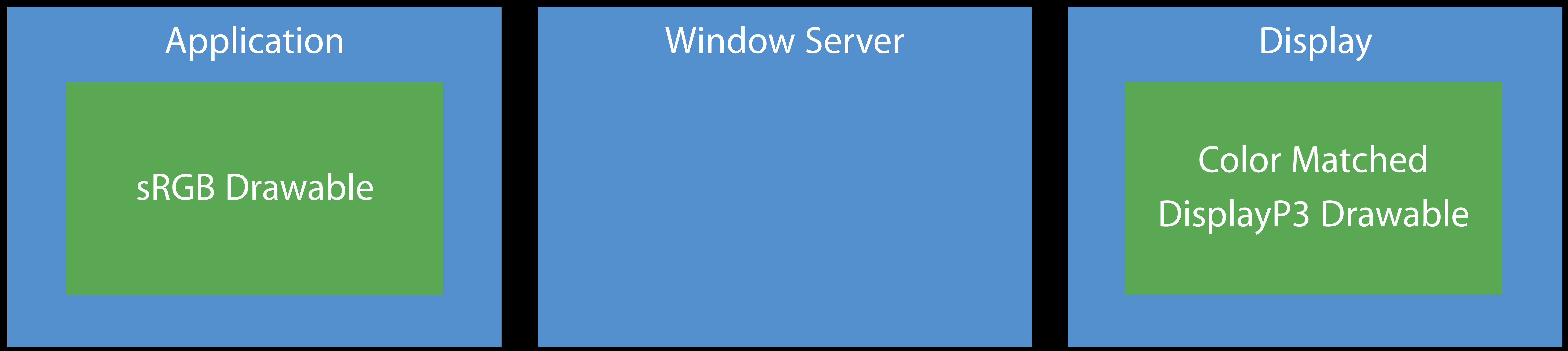
Color Management on macOS

Enable automatic color management

- Set `colorspace` property in either `NSWindow` or `CAMetalLayer`

OS performs color match

- Part of window server's normal compositing pass



Adopting Wide Color on macOS

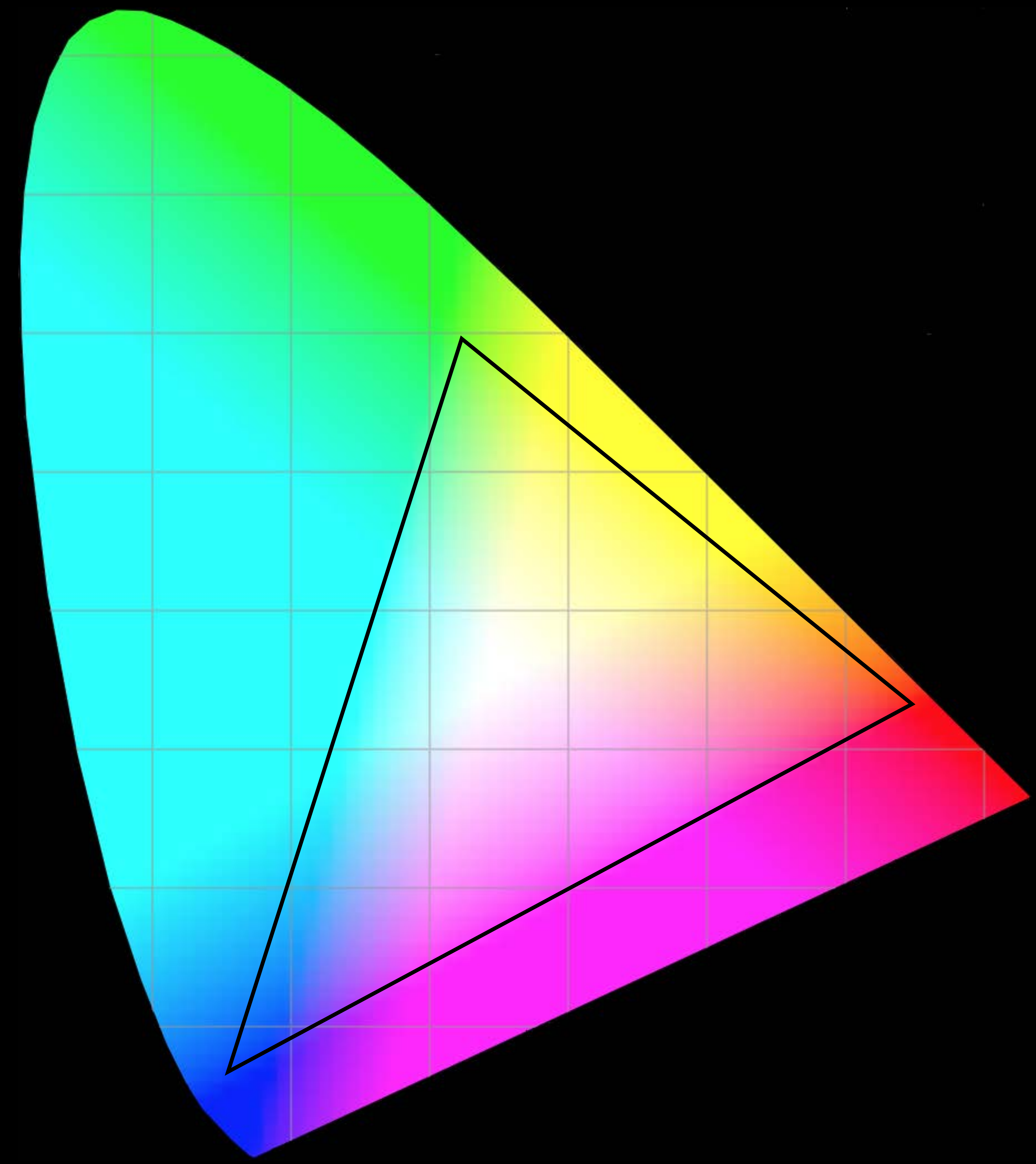
Taking advantage of wide gamut

Create content with wider colors

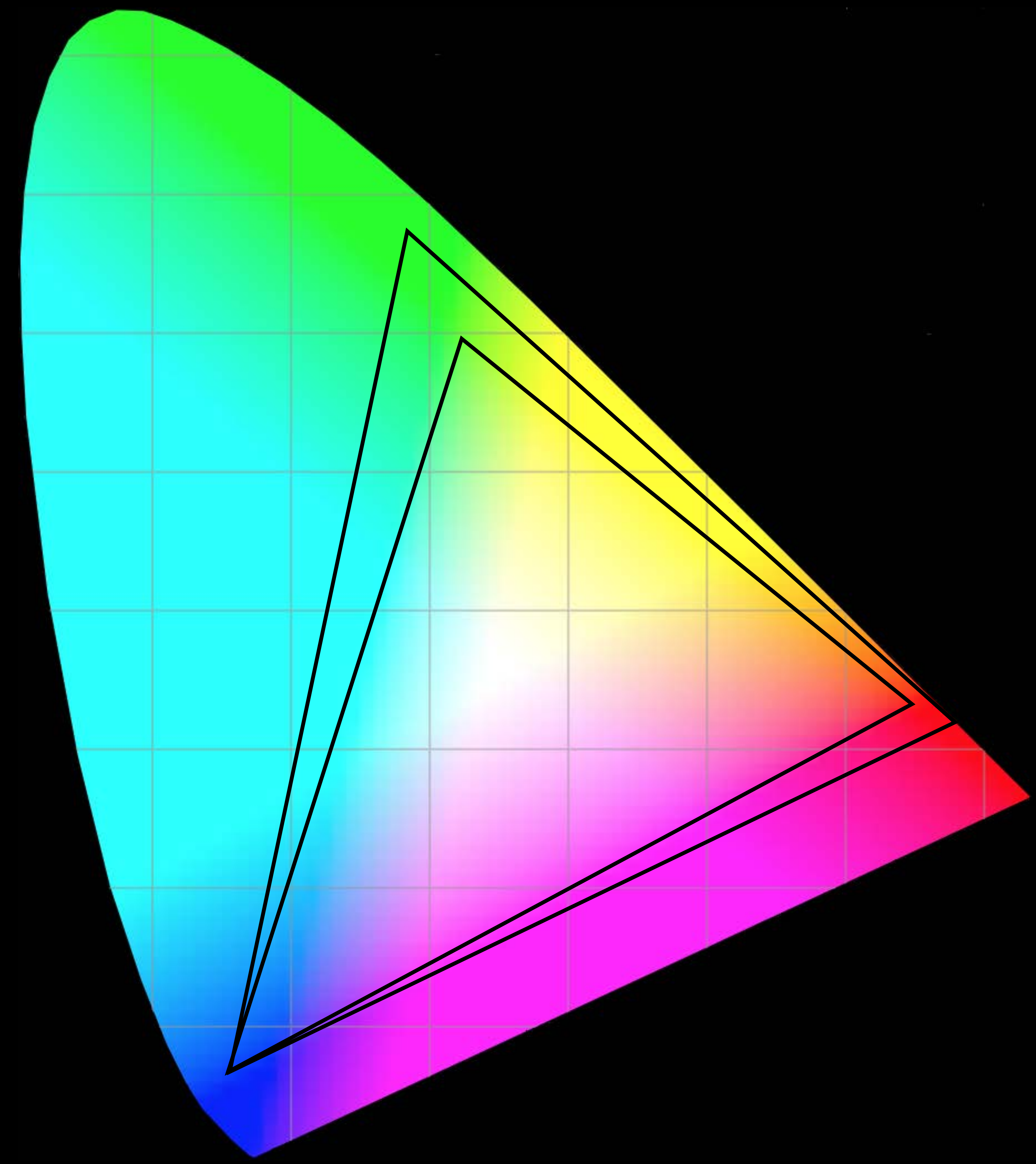
Using Extended Range sRGB simplest option

- Existing assets and pipelines "just work"
- New wide color assets provide more intense colors

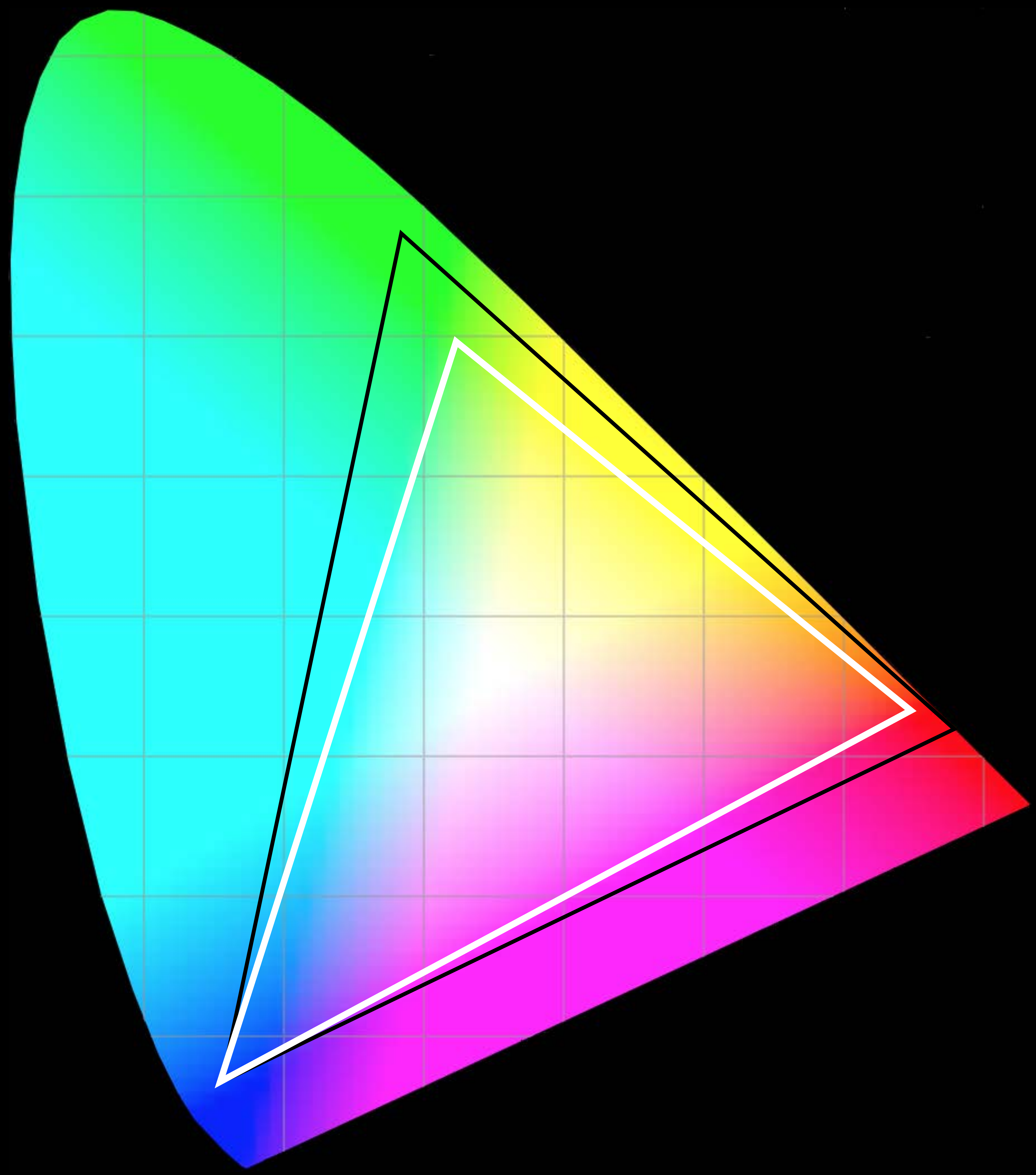
Extended Range sRGB



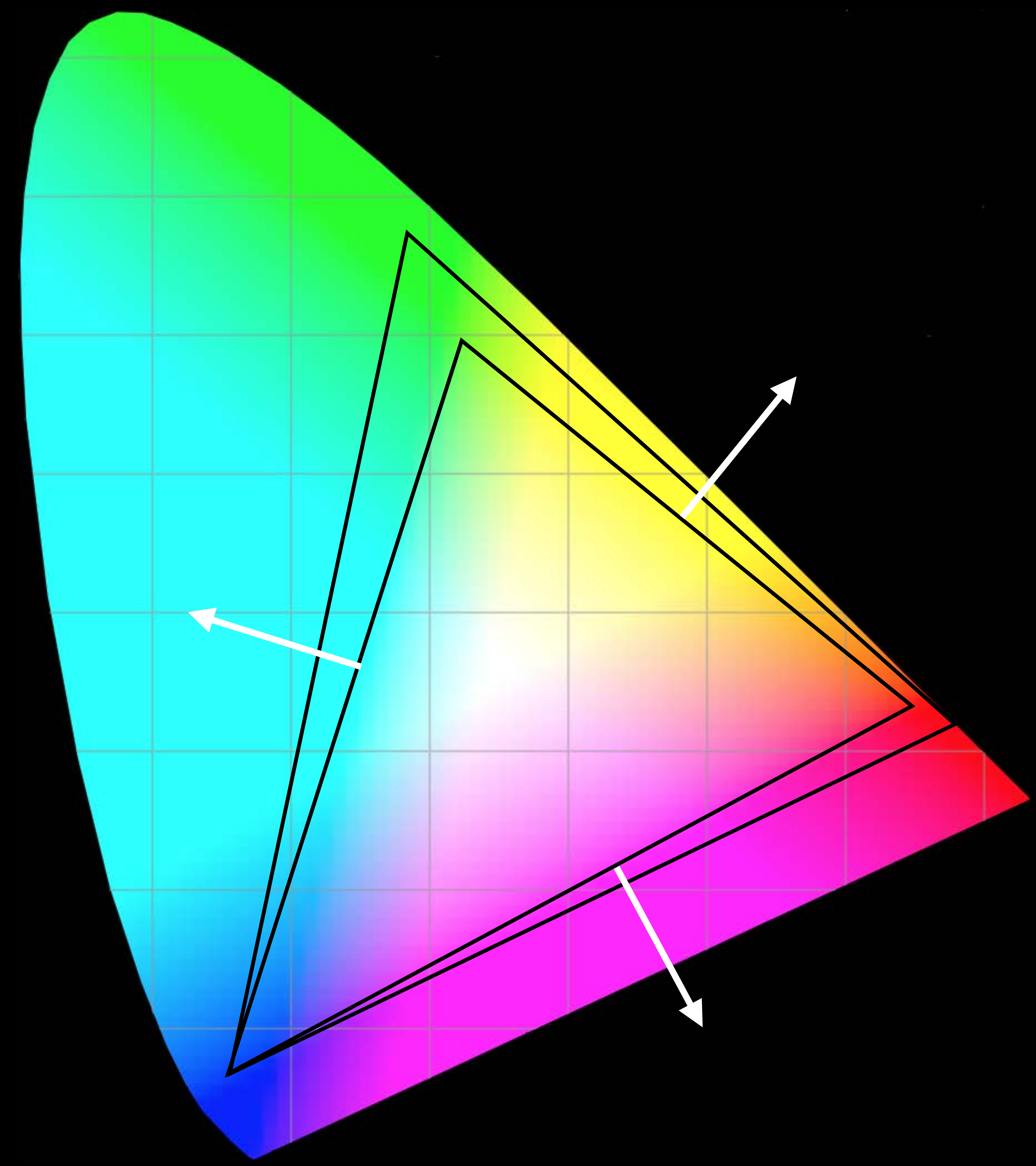
Extended Range sRGB



Extended Range sRGB



Extended Range sRGB



Adopting Wide Color on macOS

Adopting Wide Color on macOS

Use floating-point pixel formats to express values outside 0.0–1.0

Adopting Wide Color on macOS

Use floating-point pixel formats to express values outside 0.0–1.0

Recommendations for source textures

- `MTLPixelFormatBC6H_RGBFloat`
- `MTLPixelFormatRG11B10Float`
- `MTLPixelFormatRGB9E5Float`

Adopting Wide Color on macOS

Use floating-point pixel formats to express values outside 0.0–1.0

Recommendations for source textures

- `MTLPixelFormatBC6H_RGBFloat`
- `MTLPixelFormatRG11B10Float`
- `MTLPixelFormatRGB9E5Float`

Recommendations for render destinations

- `MTLPixelFormatRG11B10Float`
- `MTLPixelFormatRGBA16Float`

Color Management on iOS

Color Management on iOS

Always render in the sRGB colorspace

Color Management on iOS

Always render in the sRGB colorspace

Even when targeting devices with a P3 Display

- Colors automatically matched with no performance penalty

Color Management on iOS

Always render in the sRGB colorspace

Even when targeting devices with a P3 Display

- Colors automatically matched with no performance penalty

Use new pixel formats to take advantage of wide color

- Natively readable by display
- Can be gamma encoded for Extended Range sRGB
- Efficient for use with source textures

Extended Range sRGB Formats

NEW

`MTLPixelFormatBGR10_XR_sRGB`

- Three 10-bit channels packed into 32-bits

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10-bit Extended Range Red										10-bit Extended Range Green										10-bit Extended Range Blue										Pad	

`MTLPixelFormatBGRA10_XR_sRGB`

- Four 10-bit channels packed into 64-bits

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10-bit Extended Range Green										Pad						10-bit Extended Range Blue										Pad					

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
10-bit Extended Range Alpha										Pad						10-bit Extended Range Red										Pad					

Authoring Content

Tools and API

Wide color content creation

- Author using an image editor on macOS supporting P3 displays
- Save as 16-bit per channel PNG or JPEG using the DisplayP3 profile

Authoring Content

Tools and API

Building wide-gamut source textures

Authoring Content

Tools and API

Building wide-gamut source textures

- Build your own tool using ImageIO or vImage frameworks
  - macOS: 16-Bit DisplayP3 => `ExtendedSRGB` Float => Floating-point pixel format
  - iOS: 16-Bit DisplayP3 => `ExtendedSRGB` Float => Metal XR sRGB PixelFormat

Authoring Content

Tools and API

Building wide-gamut source textures

- Build your own tool using ImageIO or vImage frameworks
  - macOS: 16-Bit DisplayP3 => `ExtendedSRGB` Float => Floating-point pixel format
  - iOS: 16-Bit DisplayP3 => `ExtendedSRGB` Float => Metal XR sRGB PixelFormat
- Xcode asset catalogs
  - Automatically creates Extended Range sRGB textures for devices with a P3 Display

Texture Assets

Working with textures in Xcode

Asset Catalogs

What do they offer?

Create specialized asset versions based on device capabilities

Download and install only the single version made the device

Compression over the wire and on the device

Efficient access by applications

Asset Catalog Texture Sets

NEW

What do they offer?

Storage for mipmap levels

- Offline mipmap generation

Automatic color matching

- Match to sRGB or ExtendedSRGB

Optimal pixel format selection

- Compressed texture formats
- Wide color formats

Asset Catalog Texture Sets

NEW

Basic workflow

Create a texture set in an asset catalog

- Assign a name to the set
- Indicate how the texture will be used by your app
- Create assets with the Xcode UI or programmatically

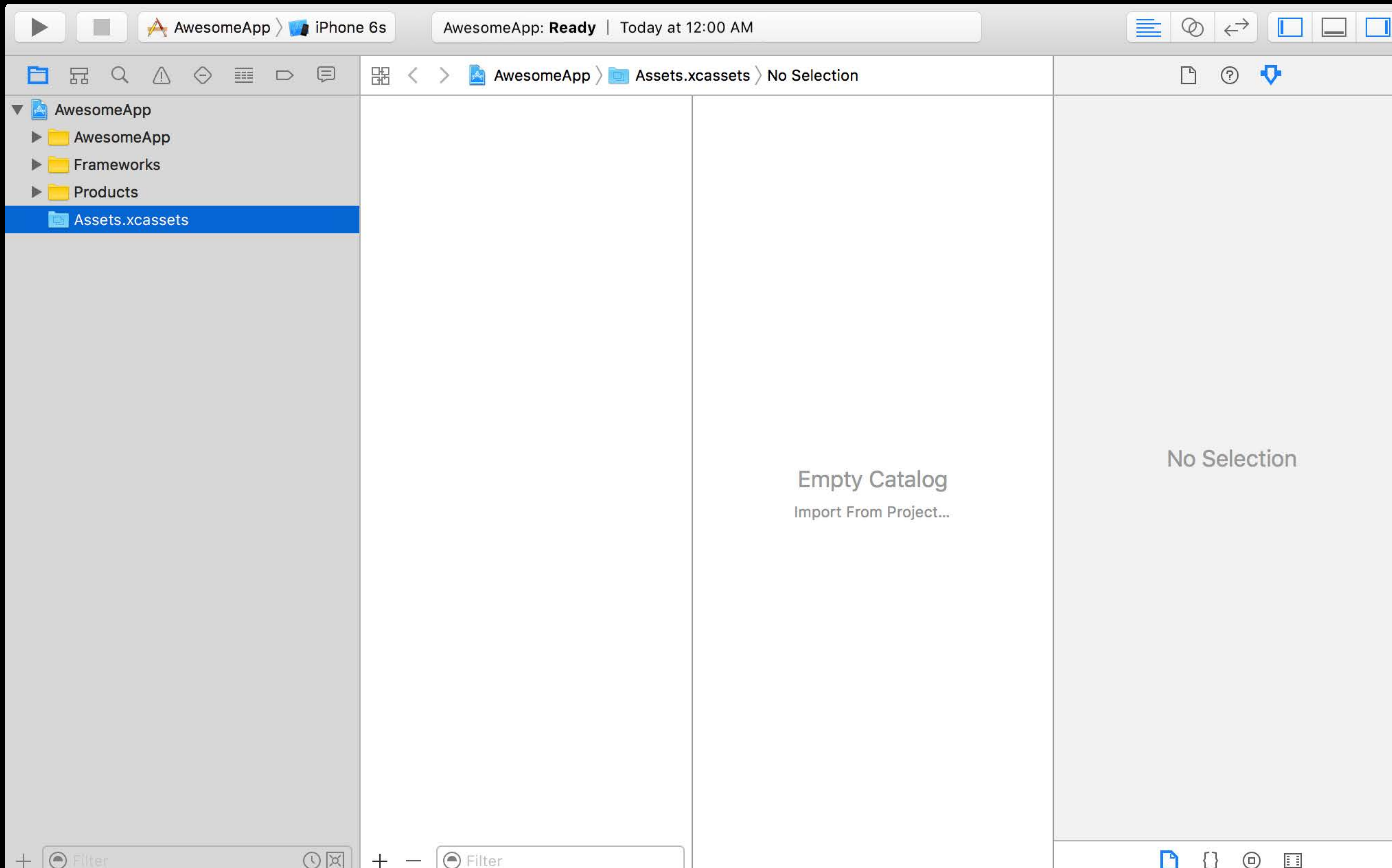
Asset Catalog Texture Sets

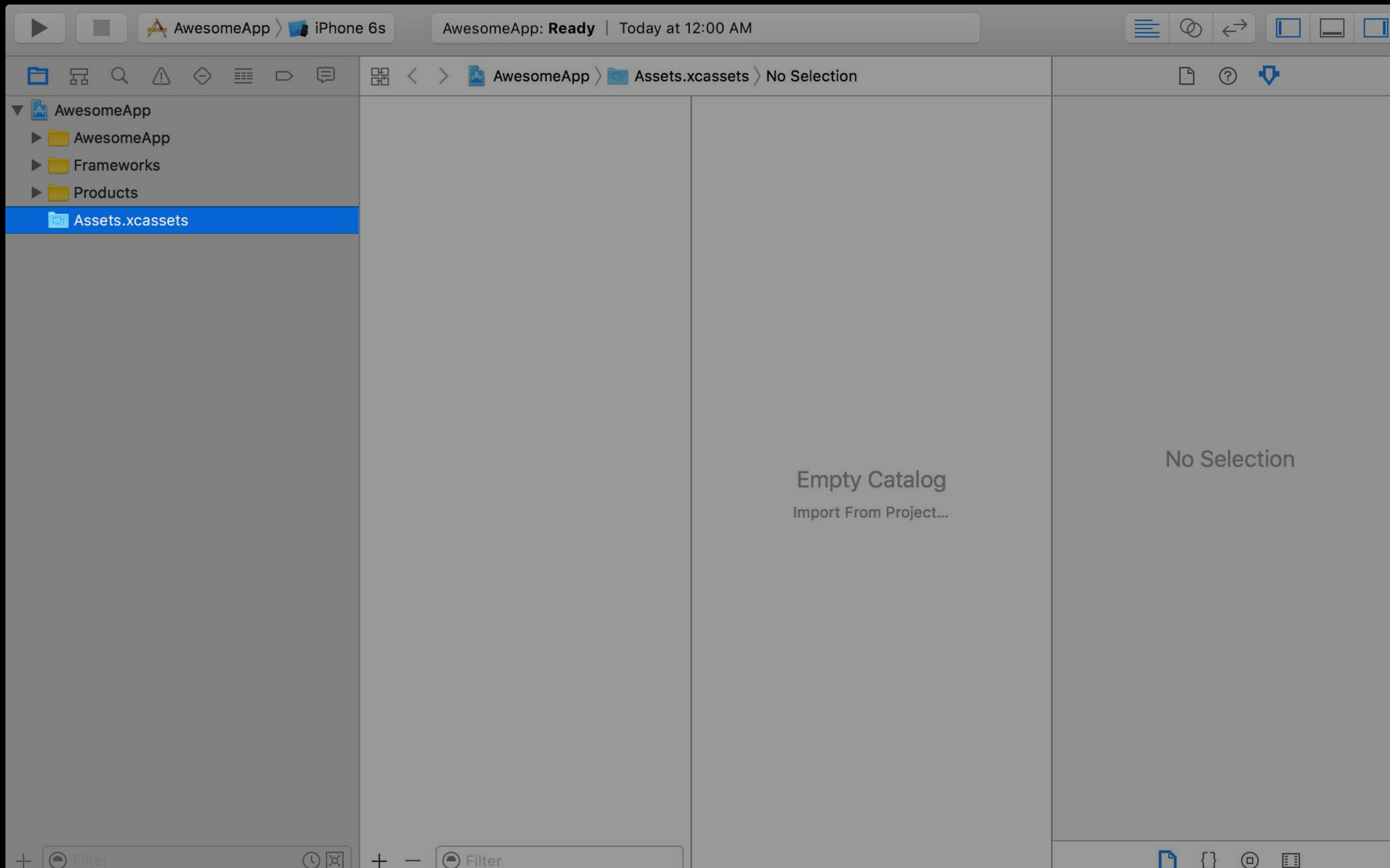
NEW

Basic workflow

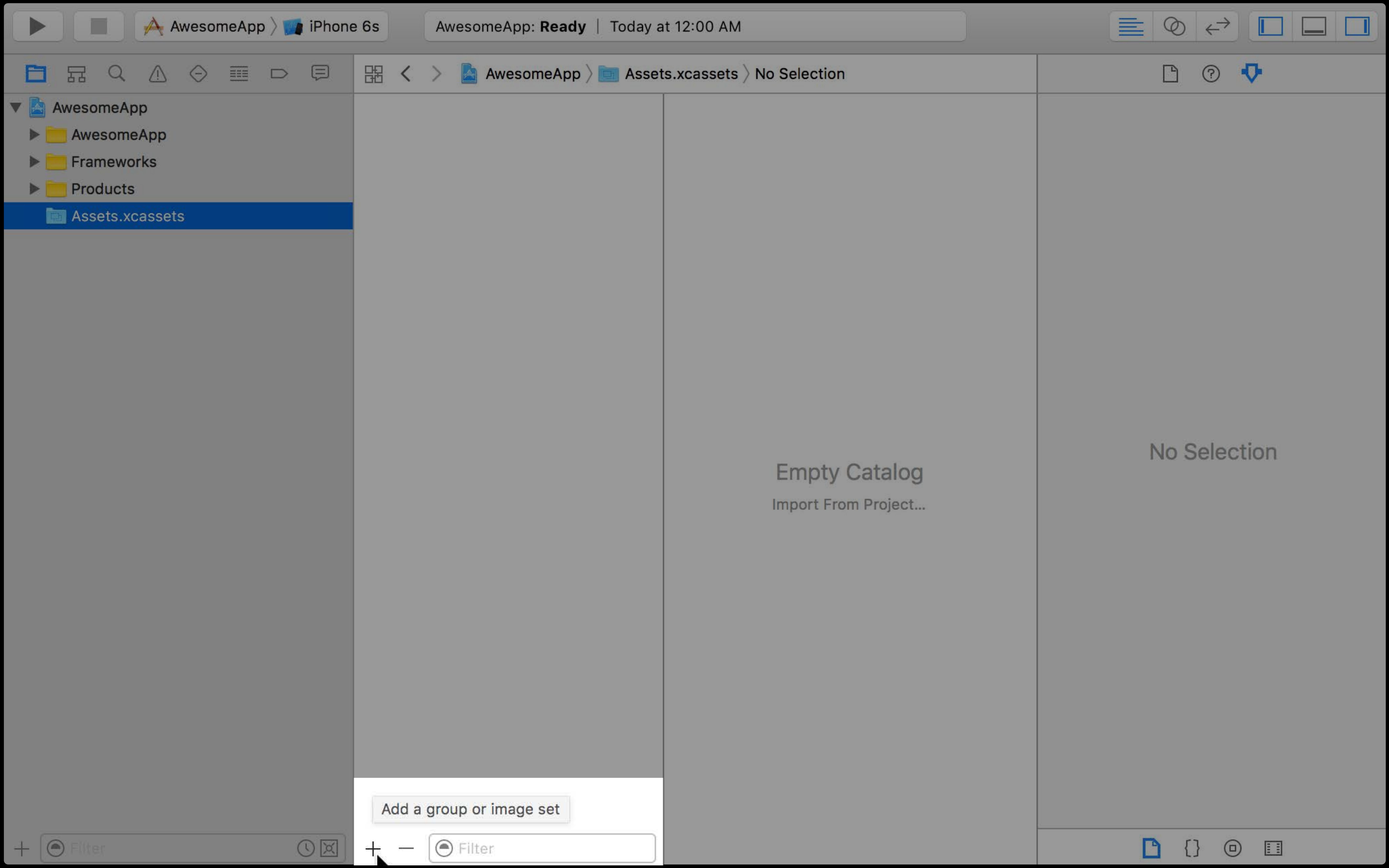
Load the texture on device

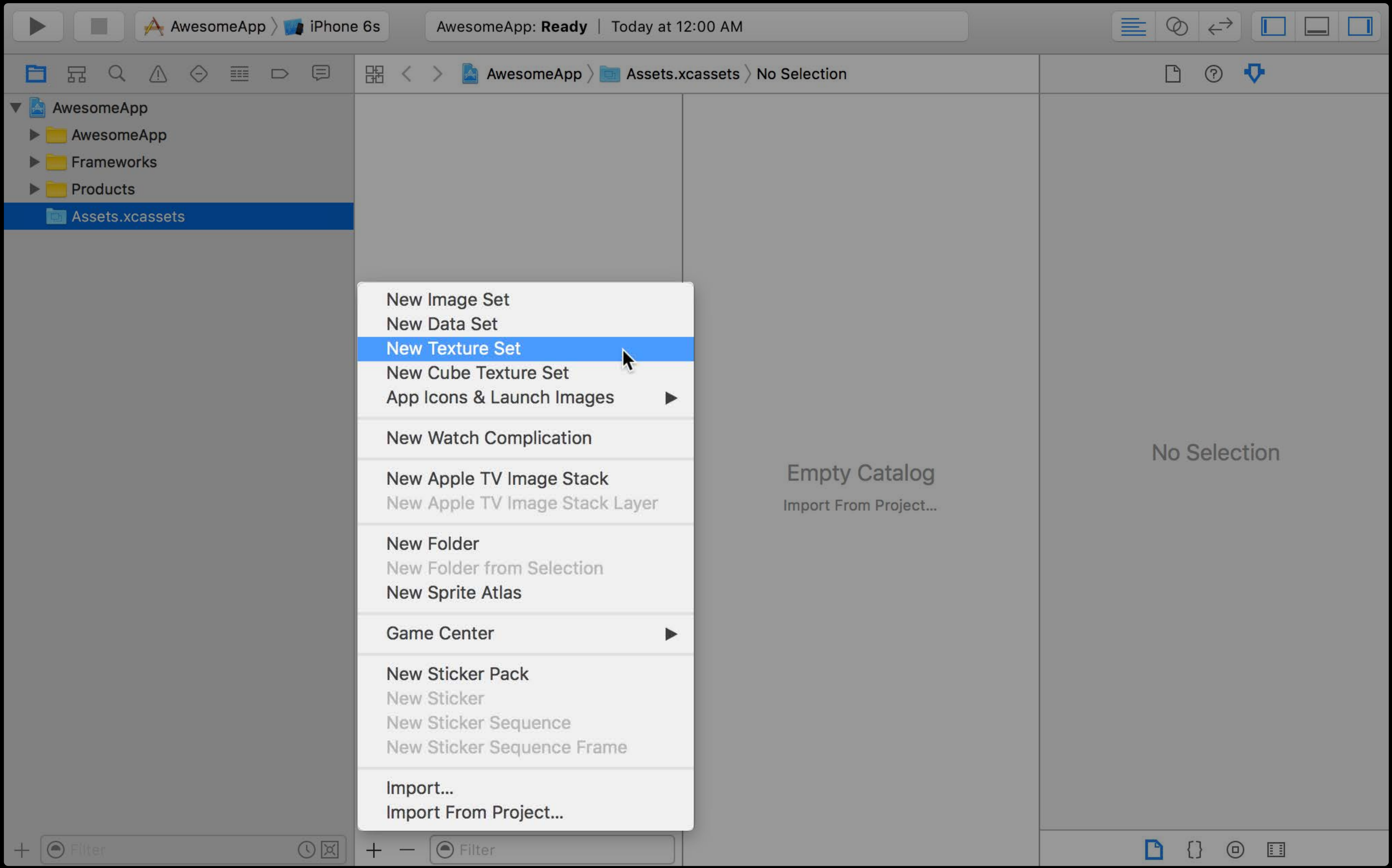
- Supply the name to MetalKit
- MetalKit creates a Metal texture from the set
- Uses data of the specialized version created for the device

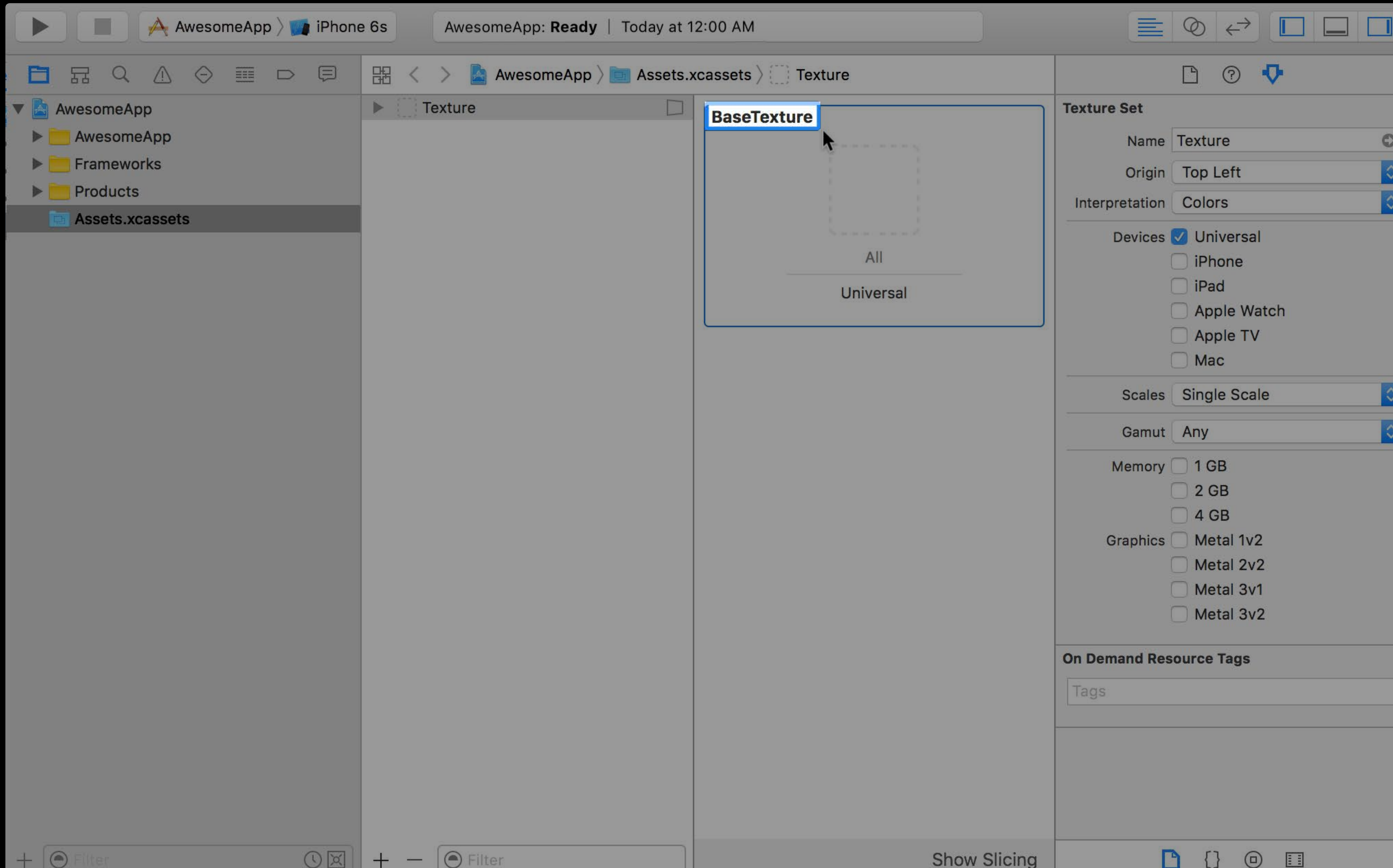

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
No Selection
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Empty Catalog
Import From Project...
No Selection
Filter
Filter

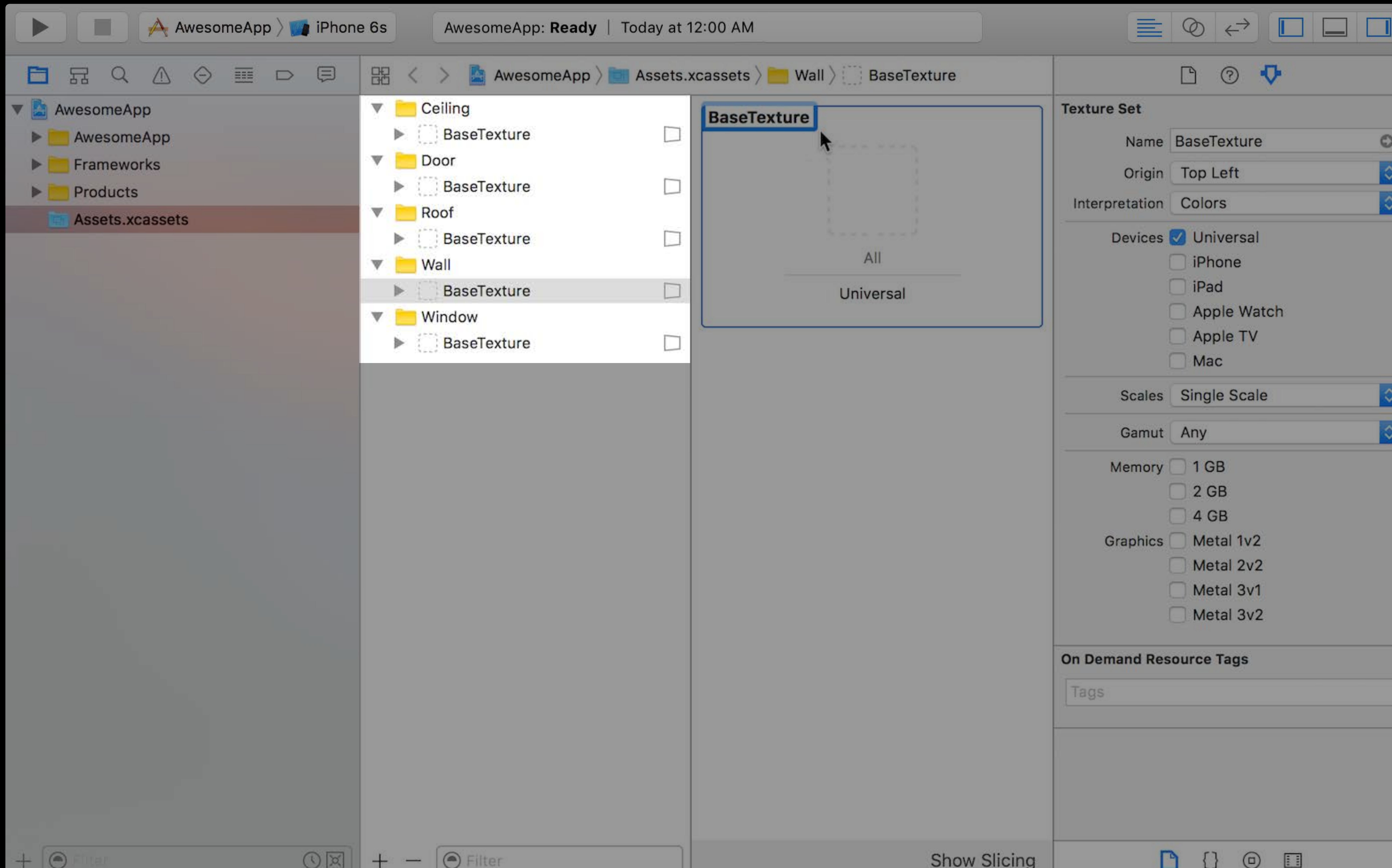


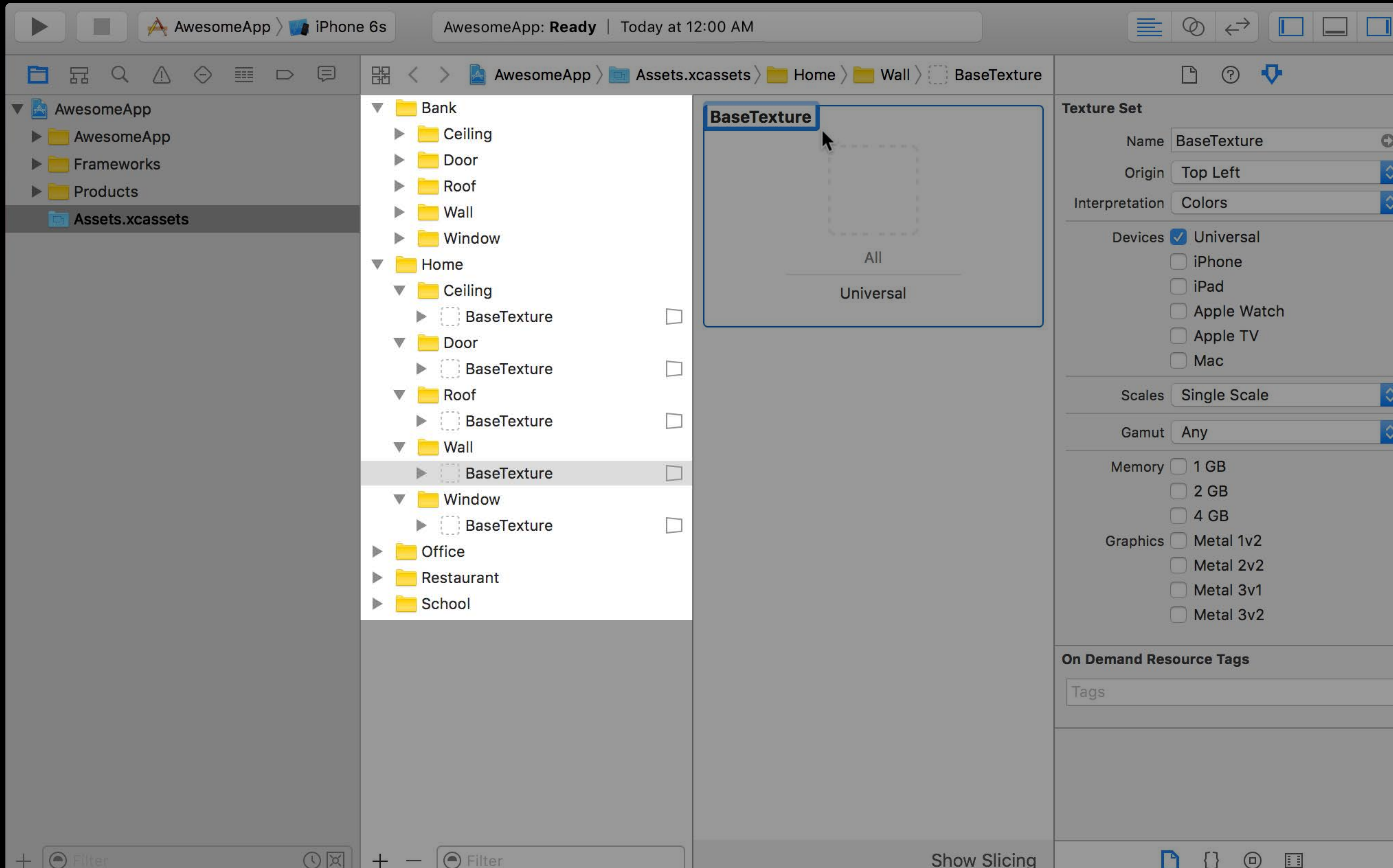
AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
No Selection
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Empty Catalog
Import From Project...
No Selection
Filter
Filter

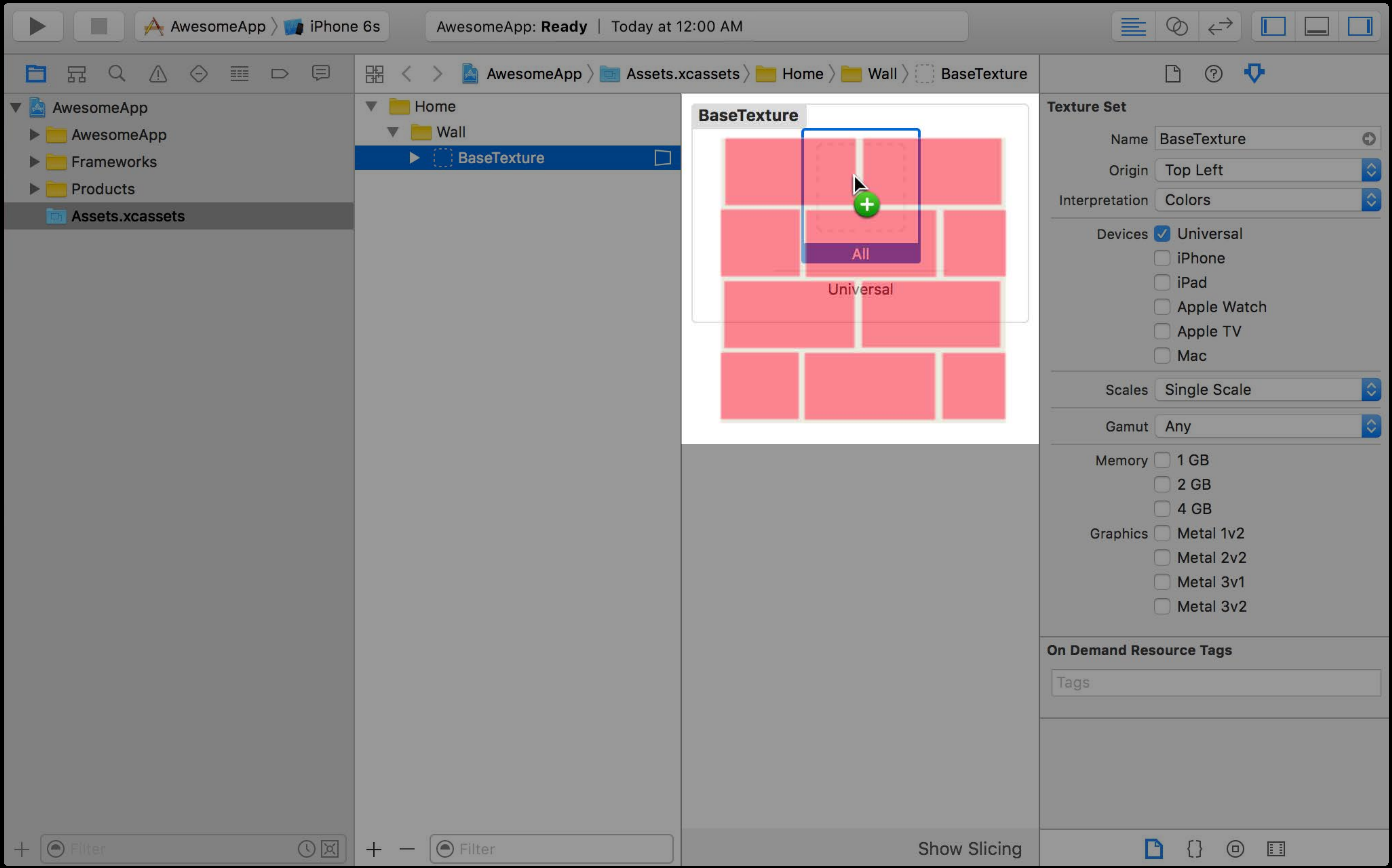

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
AwesomeApp
Assets.xcassets
No Selection
Empty Catalog
Import From Project...
No Selection
Add a group or image set
Filter


AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
No Selection
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
New Image Set
New Data Set
New Texture Set
New Cube Texture Set
App Icons & Launch Images
New Watch Complication
New Apple TV Image Stack
New Apple TV Image Stack Layer
New Folder
New Folder from Selection
New Sprite Atlas
Game Center
New Sticker Pack
New Sticker
New Sticker Sequence
New Sticker Sequence Frame
Import...
Import From Project...
Empty Catalog
Import From Project...
No Selection
Filter
Filter

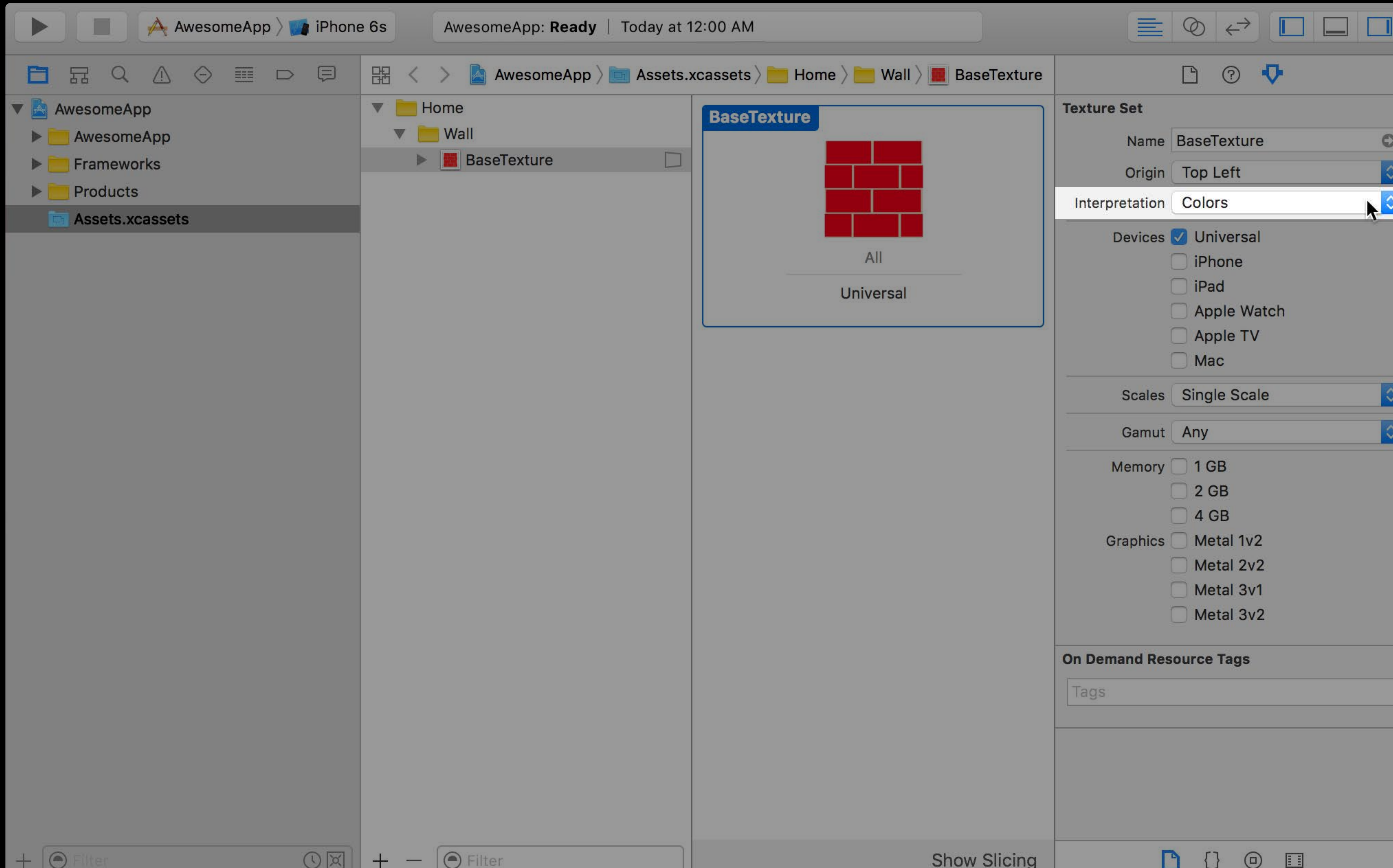

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
AwesomeApp
Assets.xcassets
Texture
Texture
BaseTexture
All
Universal
Texture Set
Name
Texture
Origin
Top Left
Interpretation
Colors
Devices
Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales
Single Scale
Gamut
Any
Memory
1 GB
2 GB
4 GB
Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
On Demand Resource Tags
Tags
Filter
Filter
Show Slicing

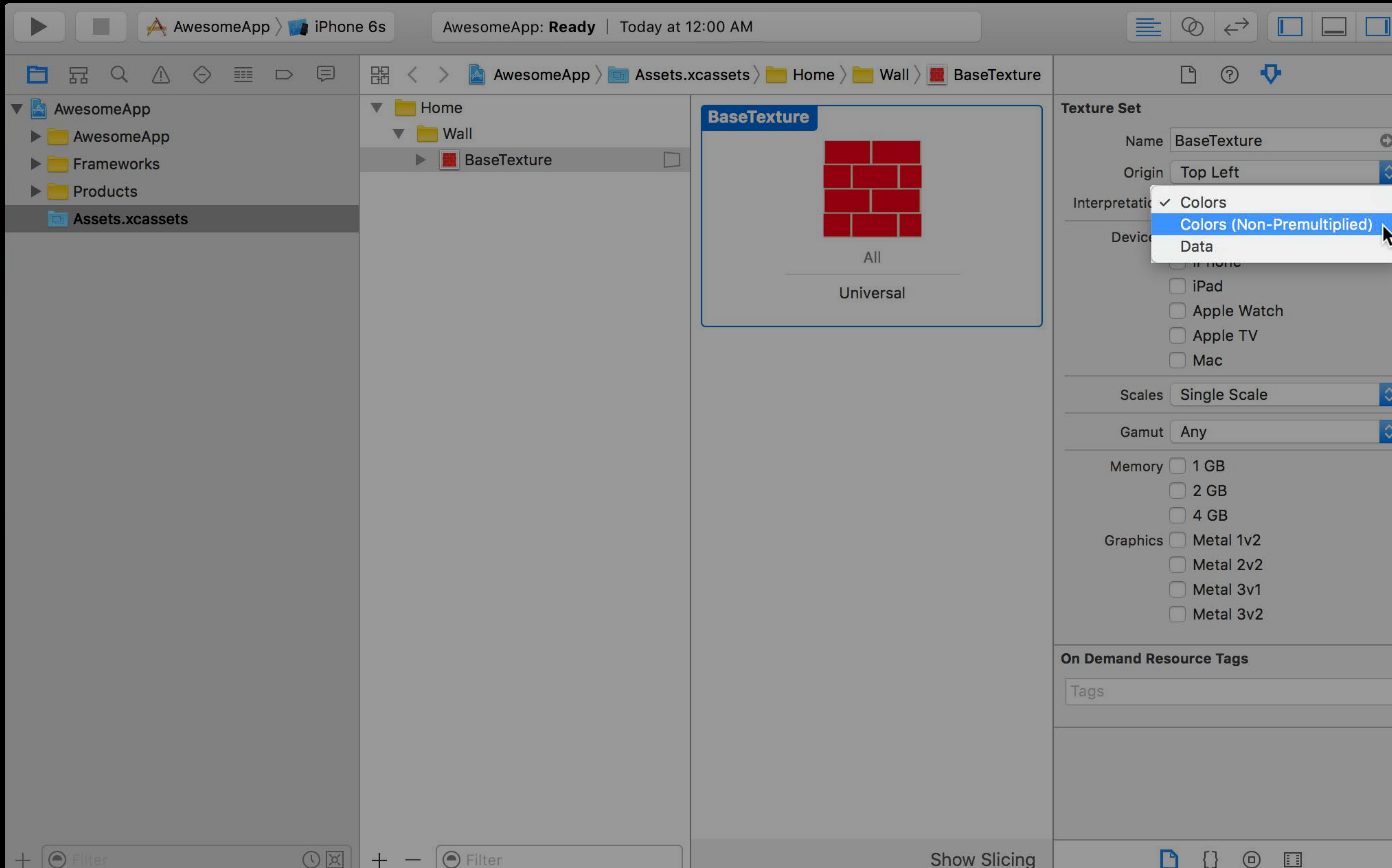

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
Wall
BaseTexture
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Ceiling
BaseTexture
Door
BaseTexture
Roof
BaseTexture
Wall
BaseTexture
Window
BaseTexture
BaseTexture
All
Universal
Texture Set
Name
BaseTexture
Origin
Top Left
Interpretation
Colors
Devices
Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales
Single Scale
Gamut
Any
Memory
1 GB
2 GB
4 GB
Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
On Demand Resource Tags
Tags
Filter
Show Slicing

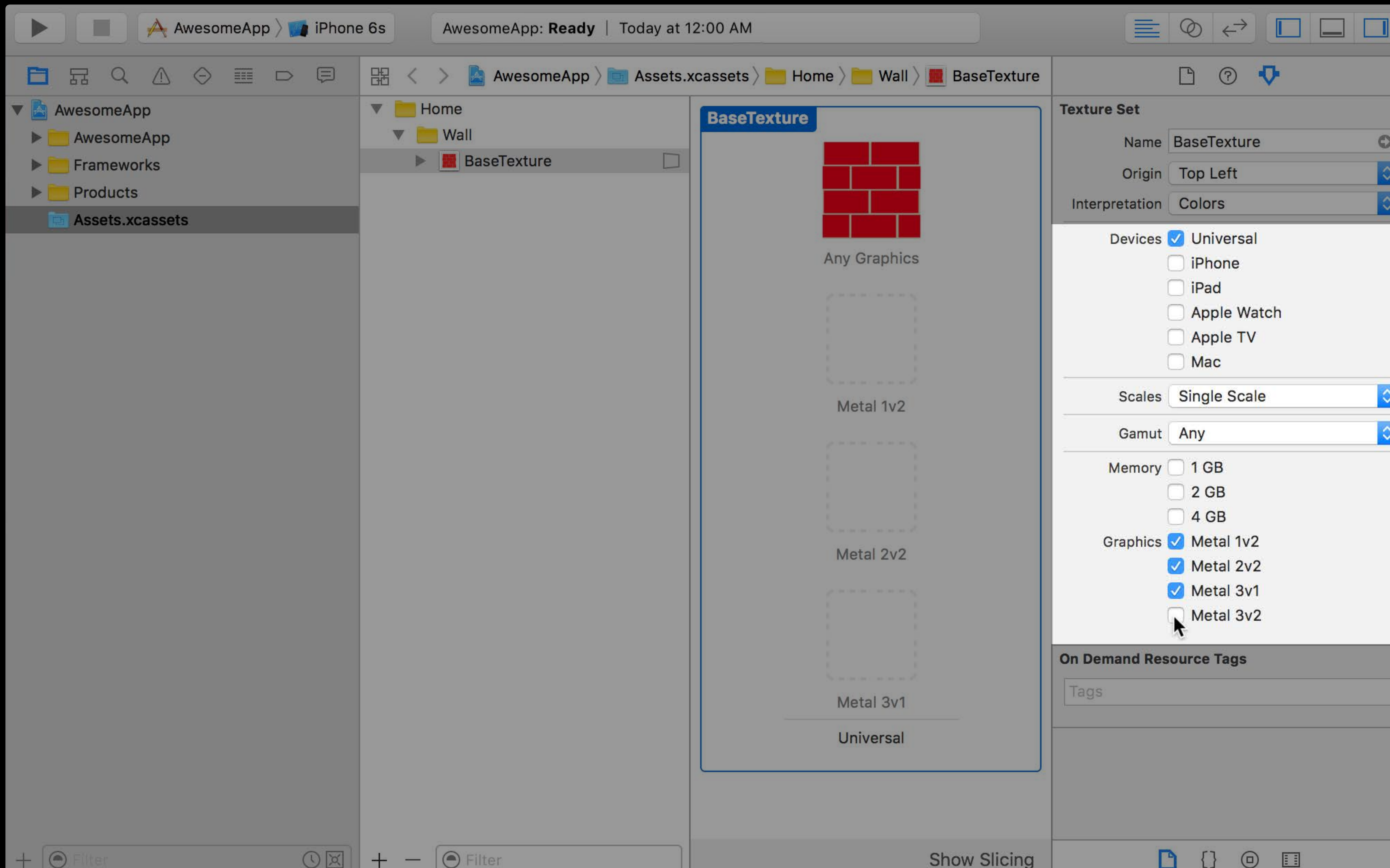

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
AwesomeApp
Assets.xcassets
Home
Wall
BaseTexture
Bank
Ceiling
Door
Roof
Wall
Window
Home
Ceiling
BaseTexture
Door
BaseTexture
Roof
BaseTexture
Wall
BaseTexture
Window
BaseTexture
Office
Restaurant
School
Filter
BaseTexture
All
Universal
Show Slicing
Texture Set
Name BaseTexture
Origin Top Left
Interpretation Colors
Devices Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales Single Scale
Gamut Any
Memory 1 GB
2 GB
4 GB
Graphics Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
On Demand Resource Tags
Tags

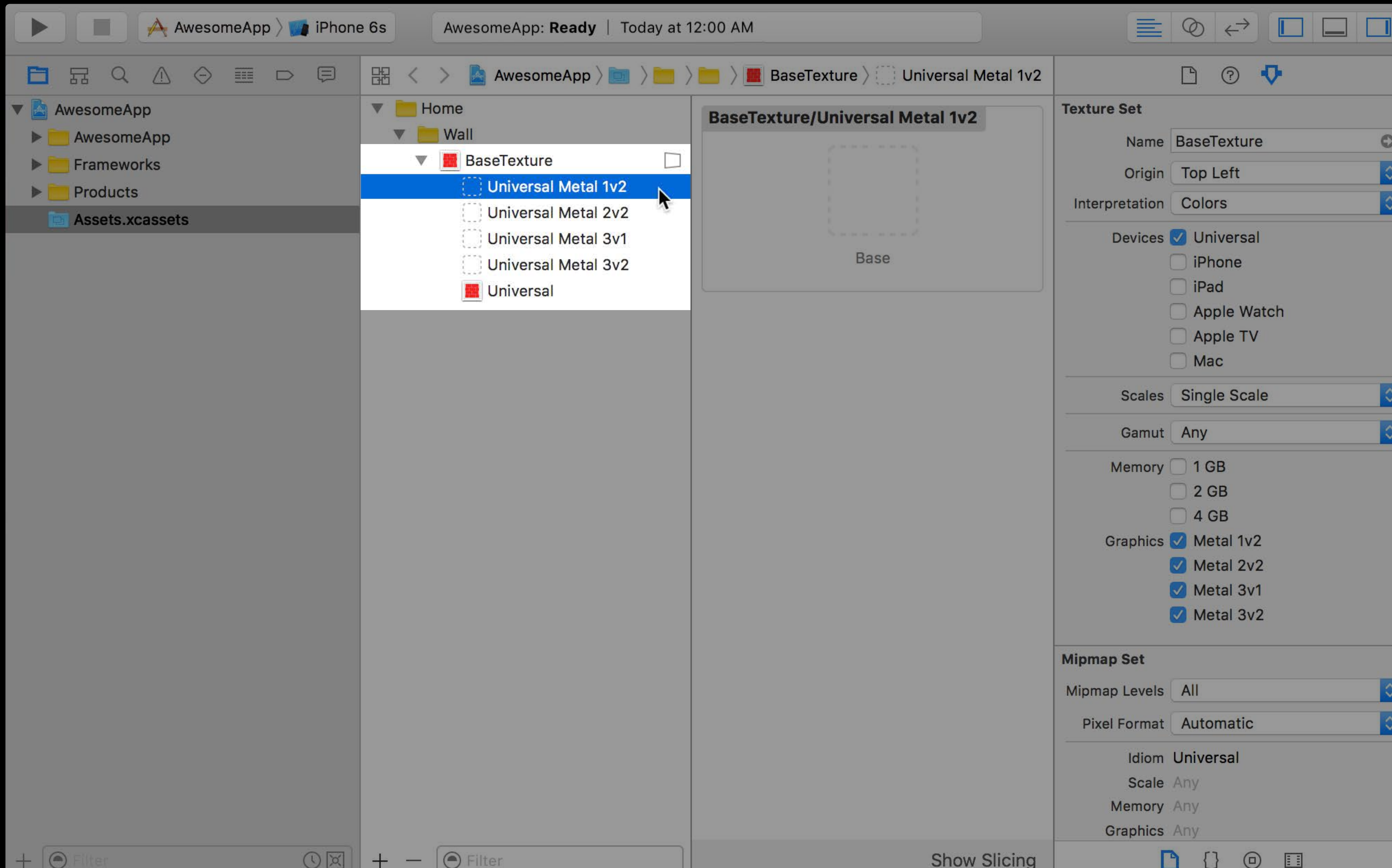


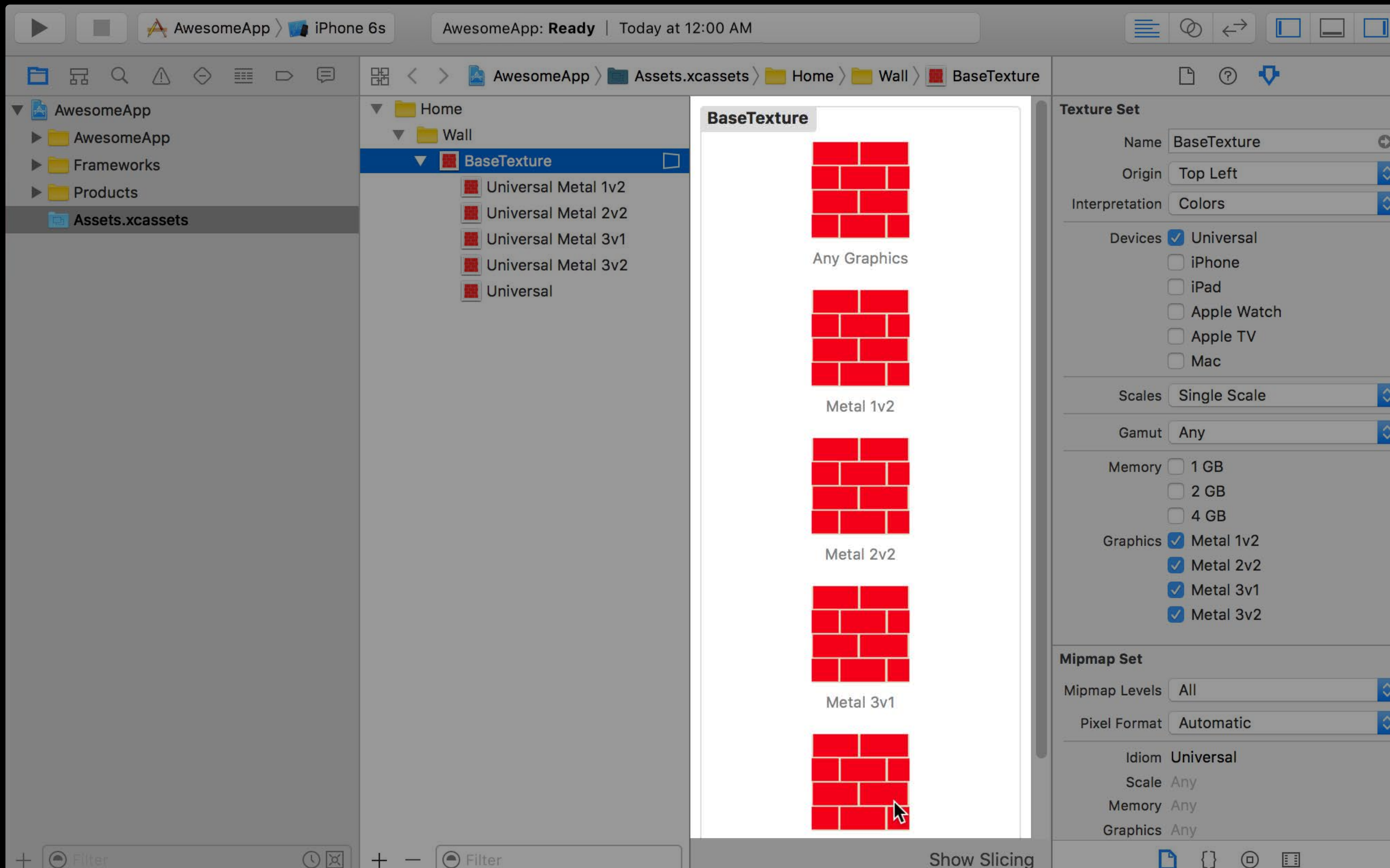
AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
Home
Wall
BaseTexture
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Home
Wall
BaseTexture
BaseTexture
All
Universal
Texture Set
Name BaseTexture
Origin Top Left
Interpretation Colors
Devices Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales Single Scale
Gamut Any
Memory 1 GB
2 GB
4 GB
Graphics Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
On Demand Resource Tags
Tags
Filter
Filter
Show Slicing

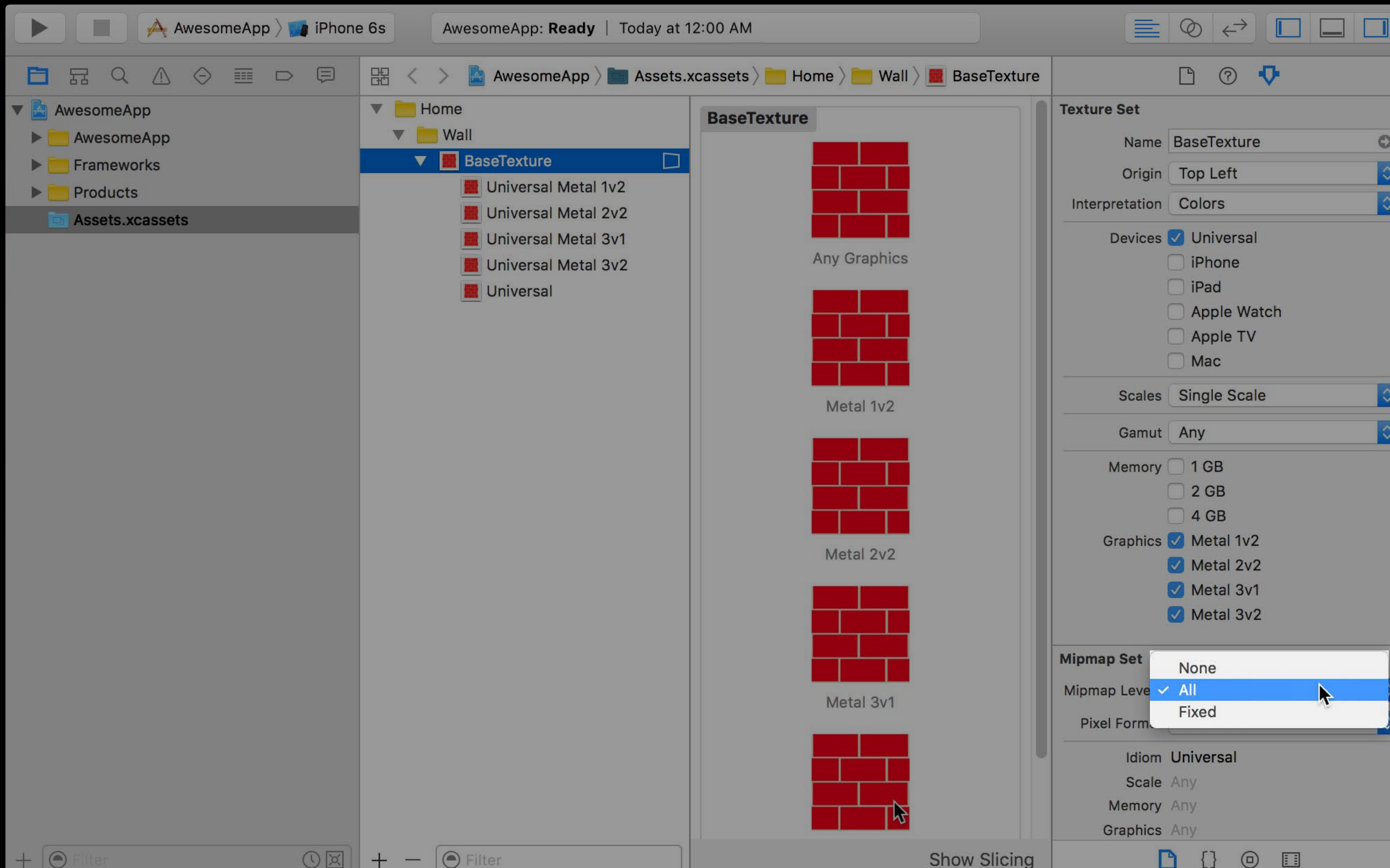

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
Home
Wall
BaseTexture
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Home
Wall
BaseTexture
BaseTexture
All
Universal
Texture Set
Name BaseTexture
Origin Top Left
Interpretation Colors
Devices Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales Single Scale
Gamut Any
Memory 1 GB
2 GB
4 GB
Graphics Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
On Demand Resource Tags
Tags
Filter
Filter
Show Slicing

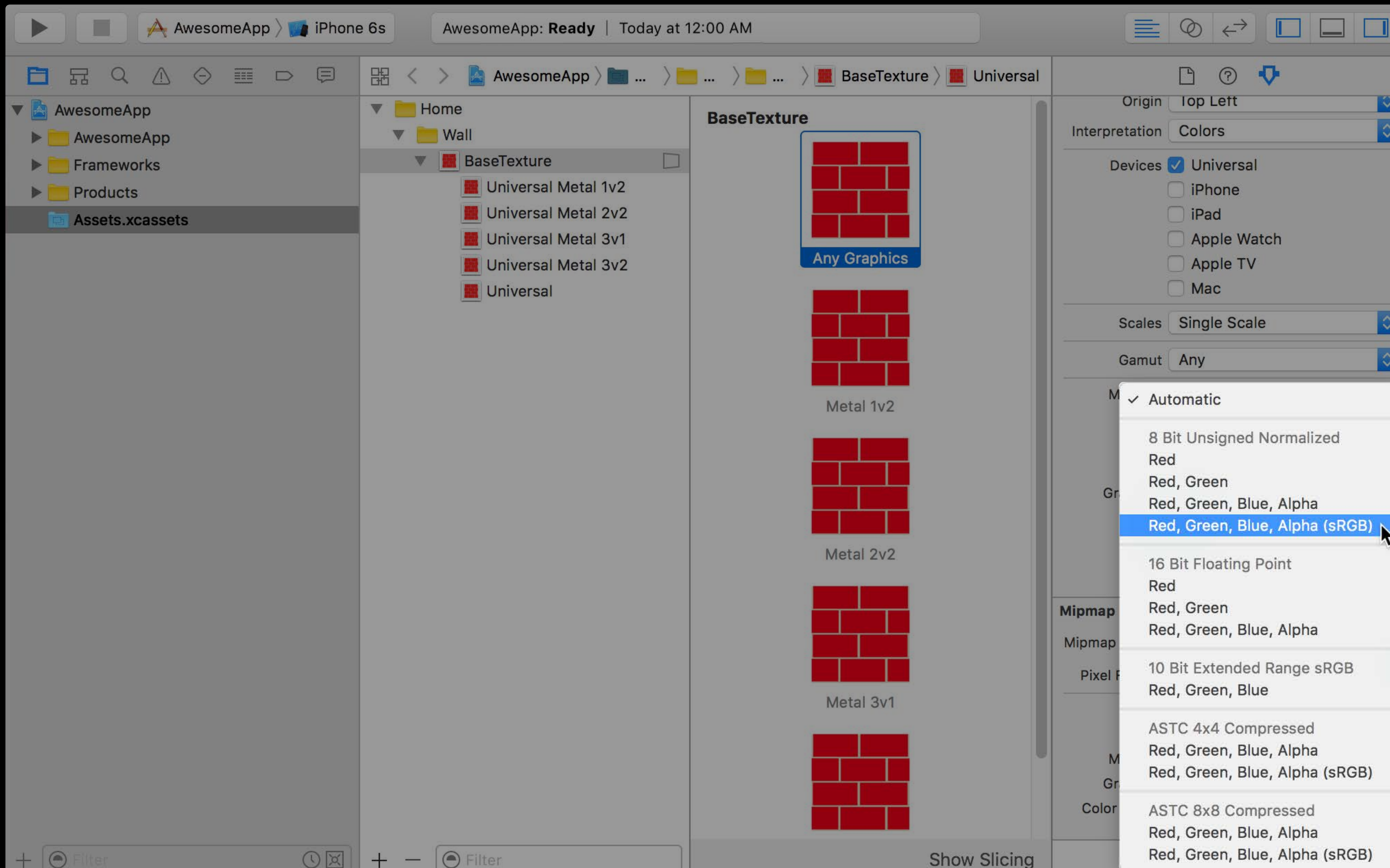

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
Home
Wall
BaseTexture
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Home
Wall
BaseTexture
BaseTexture
All
Universal
Texture Set
Name
BaseTexture
Origin
Top Left
Interpretatio
Colors
Colors (Non-Premultiplied)
Data
Devic
iPad
Apple Watch
Apple TV
Mac
Scales
Single Scale
Gamut
Any
Memory
1 GB
2 GB
4 GB
Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
On Demand Resource Tags
Tags
Filter
Filter
Show Slicing


AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
AwesomeApp
Assets.xcassets
Home
Wall
BaseTexture
Home
Wall
BaseTexture
BaseTexture
Any Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Universal
Texture Set
Name
BaseTexture
Origin
Top Left
Interpretation
Colors
Devices
Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales
Single Scale
Gamut
Any
Memory
1 GB
2 GB
4 GB
Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
On Demand Resource Tags
Tags
Filter
Filter
Show Slicing

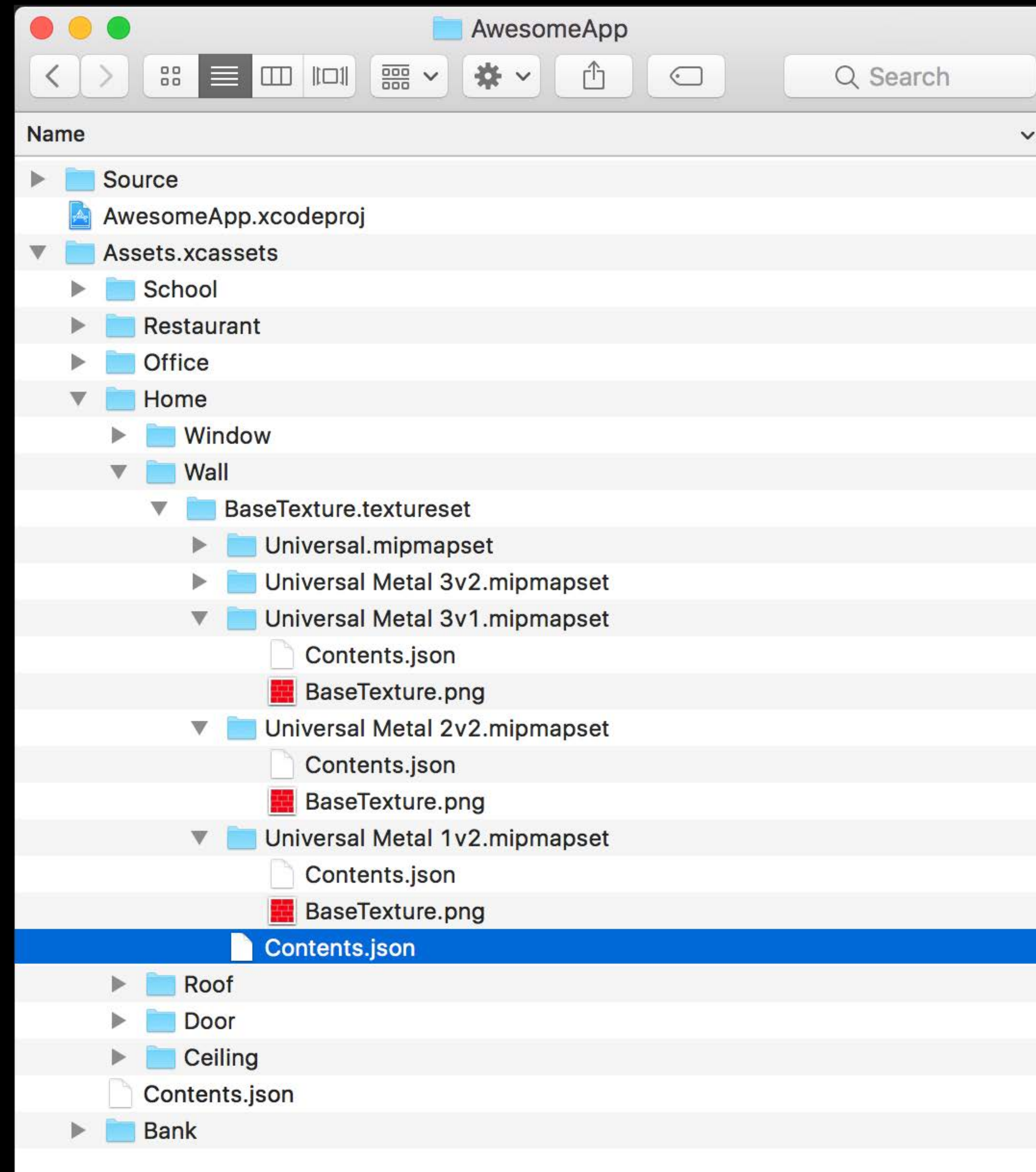

AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
AwesomeApp
BaseTexture
Universal Metal 1v2
Home
Wall
BaseTexture
Universal Metal 1v2
Universal Metal 2v2
Universal Metal 3v1
Universal Metal 3v2
Universal
BaseTexture/Universal Metal 1v2
Base
Texture Set
Name BaseTexture
Origin Top Left
Interpretation Colors
Devices Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales Single Scale
Gamut Any
Memory 1 GB
2 GB
4 GB
Graphics Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
Mipmap Set
Mipmap Levels All
Pixel Format Automatic
Idiom Universal
Scale Any
Memory Any
Graphics Any
Filter
Filter
Show Slicing


AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
Home
Wall
BaseTexture
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Home
Wall
BaseTexture
Universal Metal 1v2
Universal Metal 2v2
Universal Metal 3v1
Universal Metal 3v2
Universal
BaseTexture
Any Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Filter
Filter
Show Slicing
Texture Set
Name BaseTexture
Origin Top Left
Interpretation Colors
Devices Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales Single Scale
Gamut Any
Memory 1 GB
2 GB
4 GB
Graphics Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
Mipmap Set
Mipmap Levels All
Pixel Format Automatic
Idiom Universal
Scale Any
Memory Any
Graphics Any


AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
Assets.xcassets
Home
Wall
BaseTexture
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Home
Wall
BaseTexture
Universal Metal 1v2
Universal Metal 2v2
Universal Metal 3v1
Universal Metal 3v2
Universal
BaseTexture
Any Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Texture Set
Name
BaseTexture
Origin
Top Left
Interpretation
Colors
Devices
Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales
Single Scale
Gamut
Any
Memory
1 GB
2 GB
4 GB
Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Metal 3v2
Mipmap Set
None
All
Fixed
Idiom
Universal
Scale
Any
Memory
Any
Graphics
Any
Filter
Filter
Show Slicing



AwesomeApp
iPhone 6s
AwesomeApp: Ready | Today at 12:00 AM
AwesomeApp
BaseTexture
Universal
AwesomeApp
AwesomeApp
Frameworks
Products
Assets.xcassets
Home
Wall
BaseTexture
Universal Metal 1v2
Universal Metal 2v2
Universal Metal 3v1
Universal Metal 3v2
Universal
BaseTexture
Any Graphics
Metal 1v2
Metal 2v2
Metal 3v1
Show Slicing
Filter
Filter
Origin
Top Left
Interpretation
Colors
Devices
Universal
iPhone
iPad
Apple Watch
Apple TV
Mac
Scales
Single Scale
Gamut
Any
Mipmap
Automatic
8 Bit Unsigned Normalized
Red
Red, Green
Red, Green, Blue, Alpha
Red, Green, Blue, Alpha (sRGB)
16 Bit Floating Point
Red
Red, Green
Red, Green, Blue, Alpha
10 Bit Extended Range sRGB
Red, Green, Blue
ASTC 4x4 Compressed
Red, Green, Blue, Alpha
Red, Green, Blue, Alpha (sRGB)
ASTC 8x8 Compressed
Red, Green, Blue, Alpha
Red, Green, Blue, Alpha (sRGB)


AwesomeApp
Search
Name
Source
AwesomeApp.xcodeproj
Assets.xcassets
School
Restaurant
Office
Home
Window
Wall
BaseTexture.textureset
Universal.mipmapset
Universal Metal 3v2.mipmapset
Universal Metal 3v1.mipmapset
Contents.json
BaseTexture.png
Universal Metal 2v2.mipmapset
Contents.json
BaseTexture.png
Universal Metal 1v2.mipmapset
Contents.json
BaseTexture.png
Contents.json
Roof
Door
Ceiling
Contents.json
Bank

Contents.json

Contents.json 〉 No Selection


```json
{
  "properties" : {
    "interpretation" : "non-premultiplied-colors"
  },
  "info" : {
    "version" : 1,
    "author" : "xcode"
  },
  "textures" : [
    {
      "pixel-format" : "rbg-10-extended-range-sRGB",
      "idiom" : "universal",
      "filename" : "Universal Metal 1v2.mipmapset",
      "graphics-feature-set" : "metal1v2"
    },
    {
      "pixel-format" : "rgba-16-float",
      "idiom" : "universal",
      "filename" : "Universal Metal 2v2.mipmapset",
      "graphics-feature-set" : "metal2v2"
    },
    {
      "pixel-format" : "astc-4x4-sRGB",
      "idiom" : "universal",
      "filename" : "Universal Metal 3v1.mipmapset",
      "graphics-feature-set" : "metal3v1"
    },
    {
      "pixel-format" : "astc-8x8-sRGB",
      "idiom" : "universal",
      "filename" : "Universal Metal 3v2.mipmapset",
      "graphics-feature-set" : "metal3v2"
    },
    {
      "idiom" : "universal",
```

Loading the Texture Asset

Create the Metal texture using its name in MTKTextureLoader:

```
let textureLoader = MTKTextureLoader.init(device: device)


var wallTexture : MTLTexture! = nil

do {
   try wallTexture = textureLoader.newTexture(withName:"Home/Wall/baseTexture",
                                              scaleFactor: scaleFactor,
                                              bundle:nil,
                                              options:nil)

} catch let error {

   print("Error: \(error)")

}
```

Loading the Texture Asset

Create the Metal texture using its name in MTKTextureLoader:

```
let textureLoader = MTKTextureLoader.init(device: device)


var wallTexture : MTLTexture! = nil
do {
   try wallTexture = textureLoader.newTexture(withName:"Home/Wall/baseTexture",
                                              scaleFactor: scaleFactor,
                                              bundle:nil,
                                              options:nil)
} catch let error {
   print("Error: \(error)")
}
```

Loading the Texture Asset

Create the Metal texture using its name in MTKTextureLoader:

```
let textureLoader = MTKTextureLoader.init(device: device)

var wallTexture : MTLTexture! = nil
do {
   try wallTexture = textureLoader.newTexture(withName:"Home/Wall/baseTexture",
                                              scaleFactor: scaleFactor,
                                              bundle:nil,
                                              options:nil)
} catch let error {
   print("Error: \(error)")
}
```

Summary

Pay attention to colorspace

Take advantage of wide-gamut displays

Asset catalogs can help you target wide color

Get the best texture for every device

Additions to Metal Performance Shaders

Anna Tikhonova GPU Software Engineer

Metal Performance Shaders (MPS)

Recap

A framework of data-parallel algorithms for the GPU

Optimized for iOS

Designed to integrate easily into your Metal applications

As simple as calling a library function

Metal Performance Shaders (MPS)

Supported image operations

Convolution: General, Gaussian Blur, Box, Tent, and Sobel

Morphology: Min, Max, Dilate, and Erode

Lanczos Resampling

Histogram, Equalization, and Specification

Median Filter

Thresholding

Image Integral

Metal Performance Shaders (MPS)

NEW

New operations

Wide Color Conversion

Metal Performance Shaders (MPS)

NEW

New operations

Wide Color Conversion

Gaussian Pyramid

Metal Performance Shaders (MPS)

NEW

New operations

Wide Color Conversion

Gaussian Pyramid

Convolutional Neural Networks (CNNs)

Deep Learning

Can a machine do the same task a human can do?

Deep Learning

Can a machine do the same task a human can do?

Deep Learning

Can a machine do the same task a human can do?

Amber and Julie are skateboarding at the beach.

Deep Learning

Can a machine do the same task a human can do?

Deep Learning

Can a machine do the same task a human can do?

Some Applications of Deep Learning

Images

- Object detection and recognition, image classification and segmentation

Audio

- Speech recognition and translation

Haptics

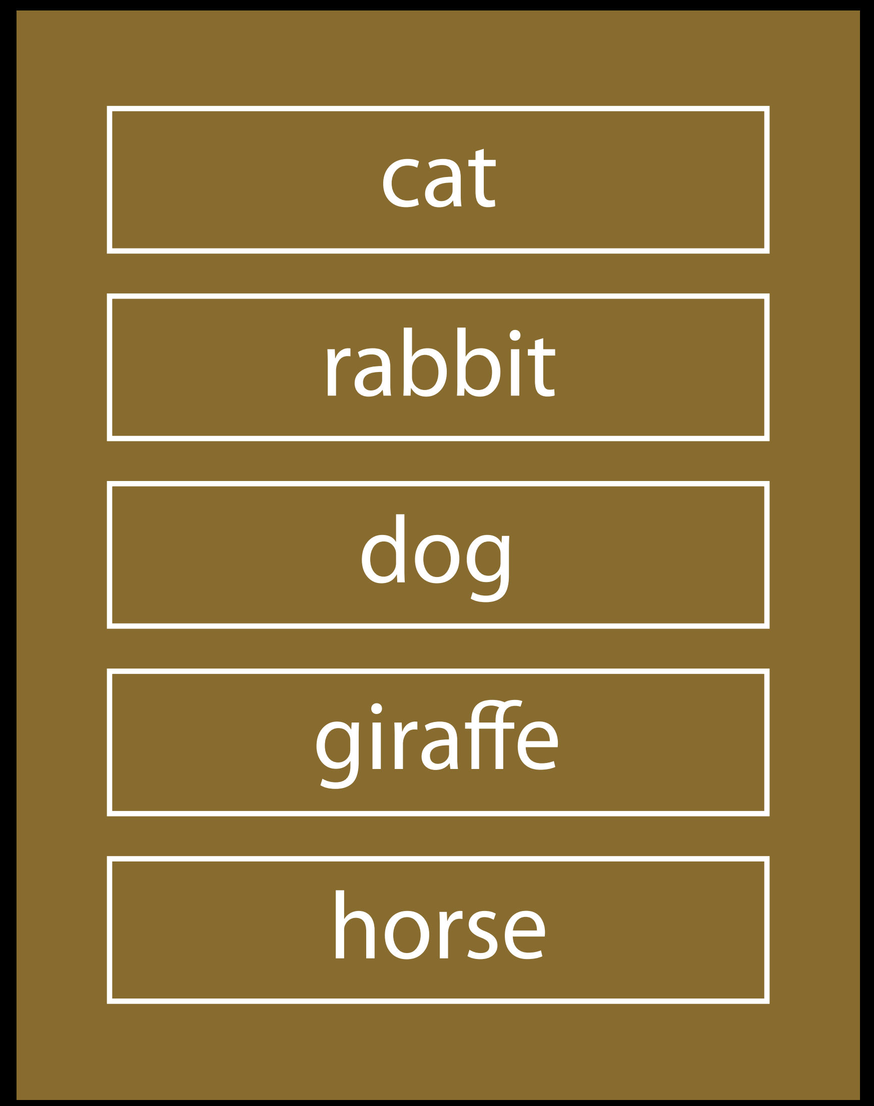
- Sense of touch

Training

First phase

Training

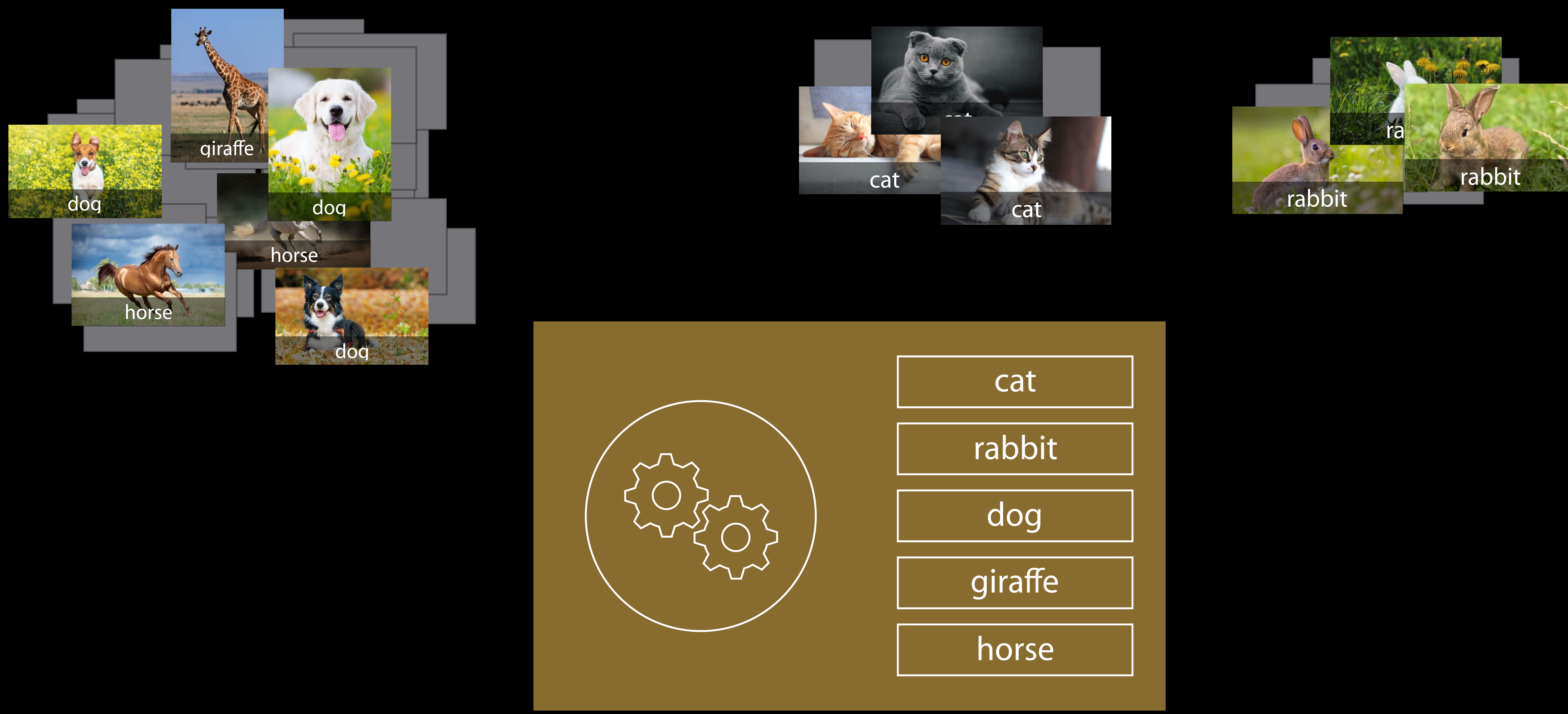
First phase



Training to Classify Images

Training

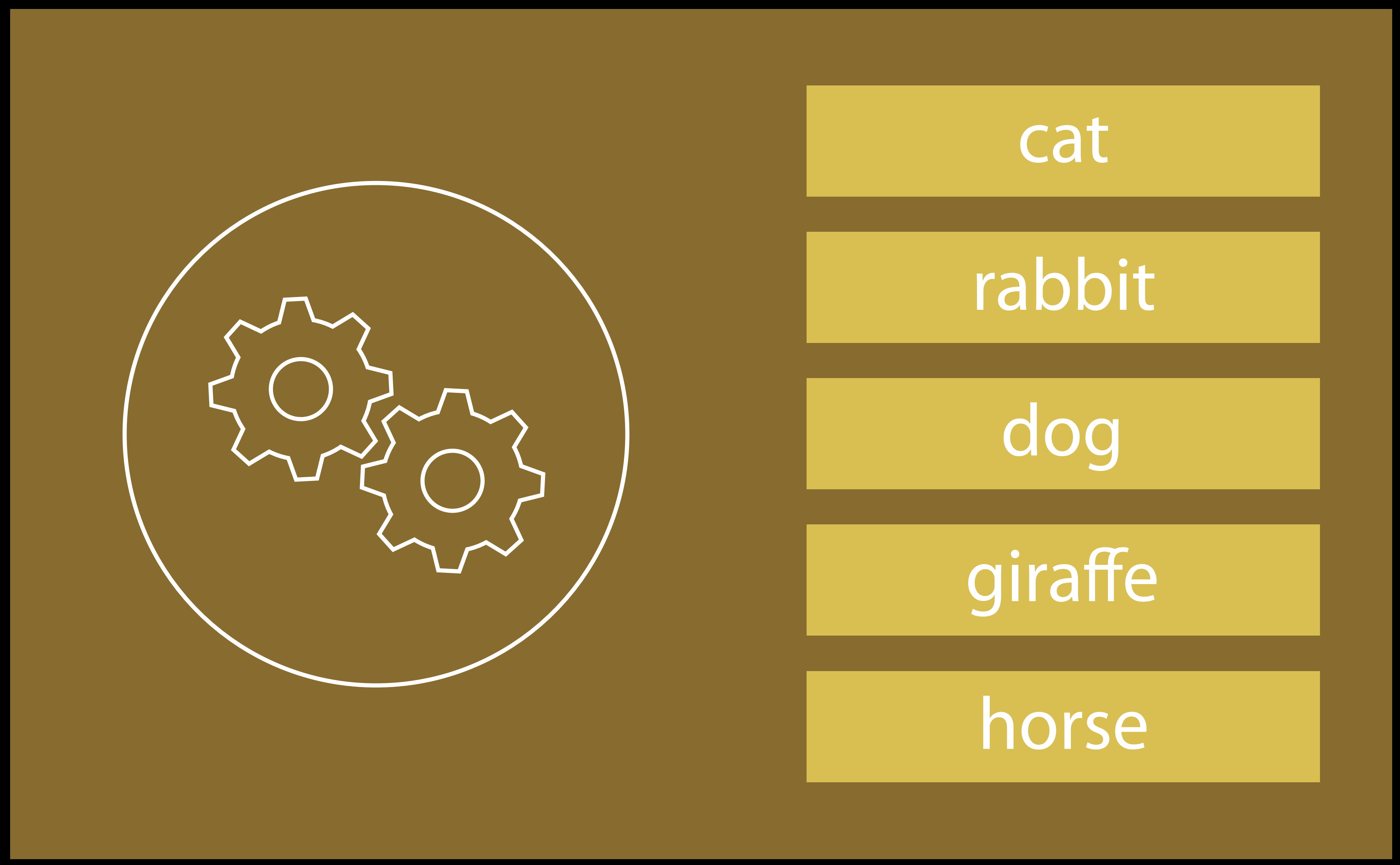
First phase



Training to Classify Images

Training

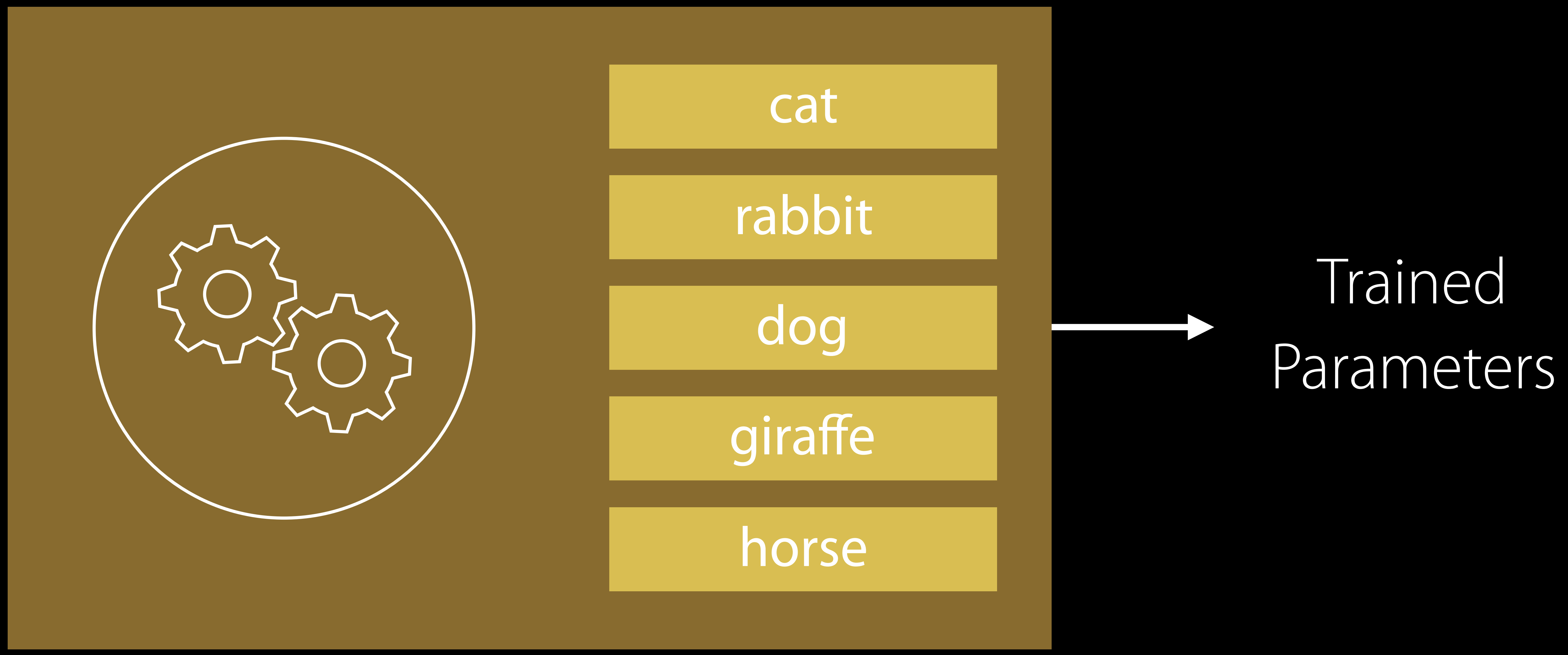
First phase



Training to Classify Images

Training

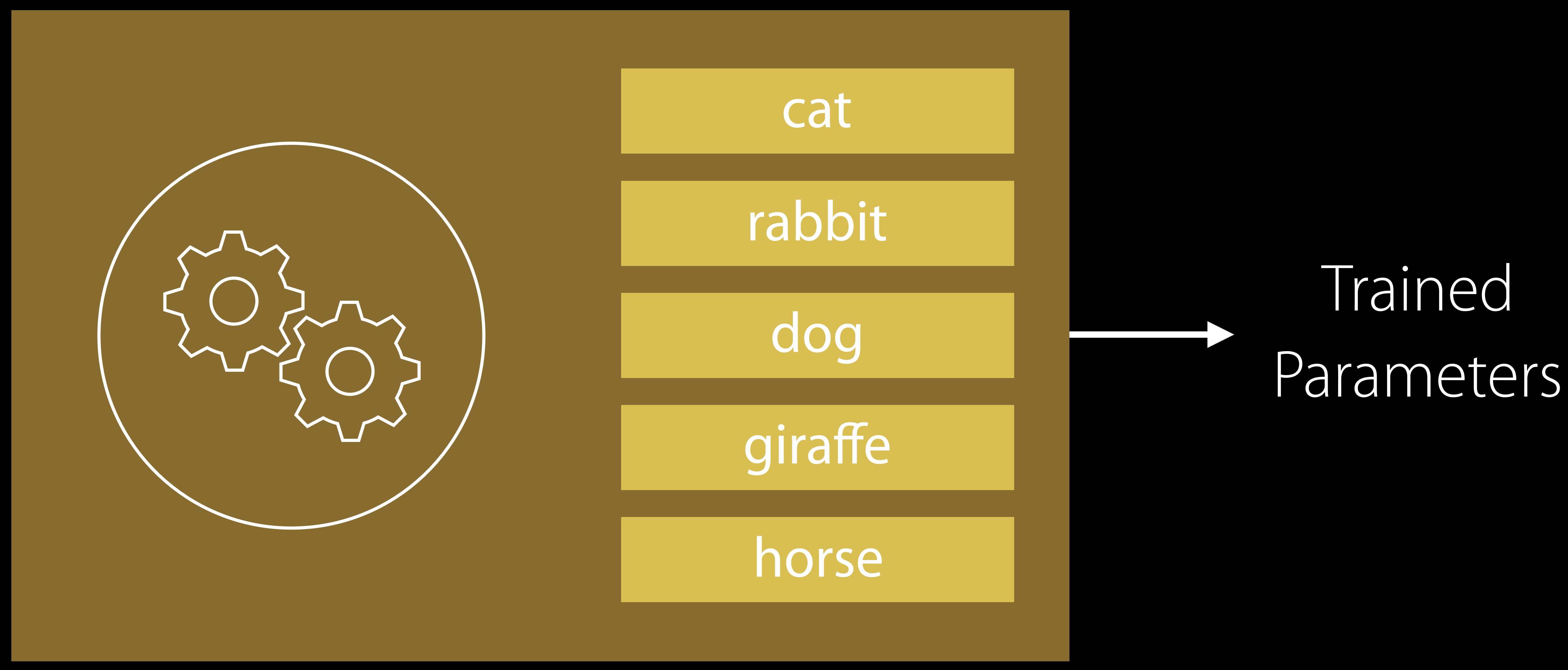
First phase



Training to Classify Images

Training

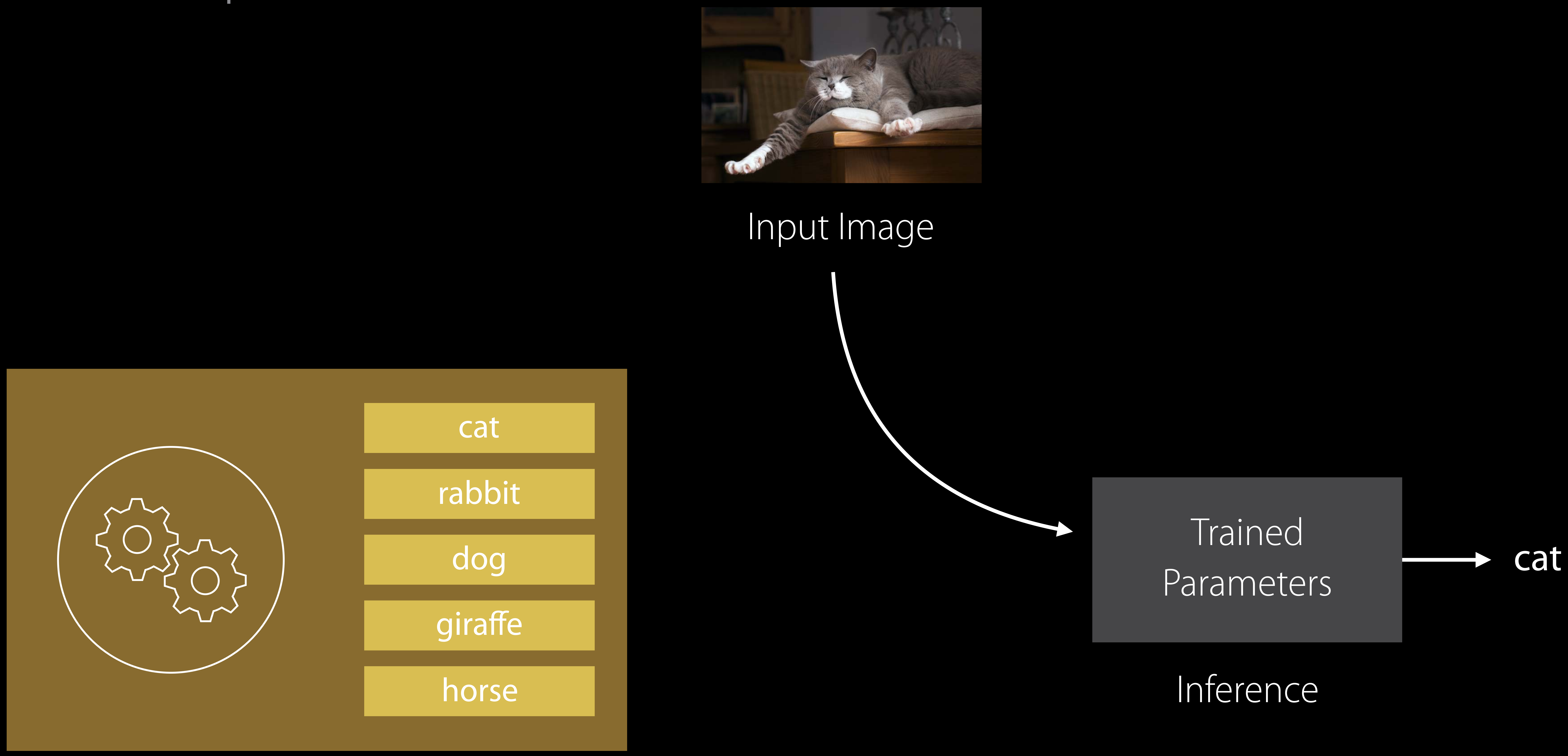
Second phase



Training to Classify Images

Training

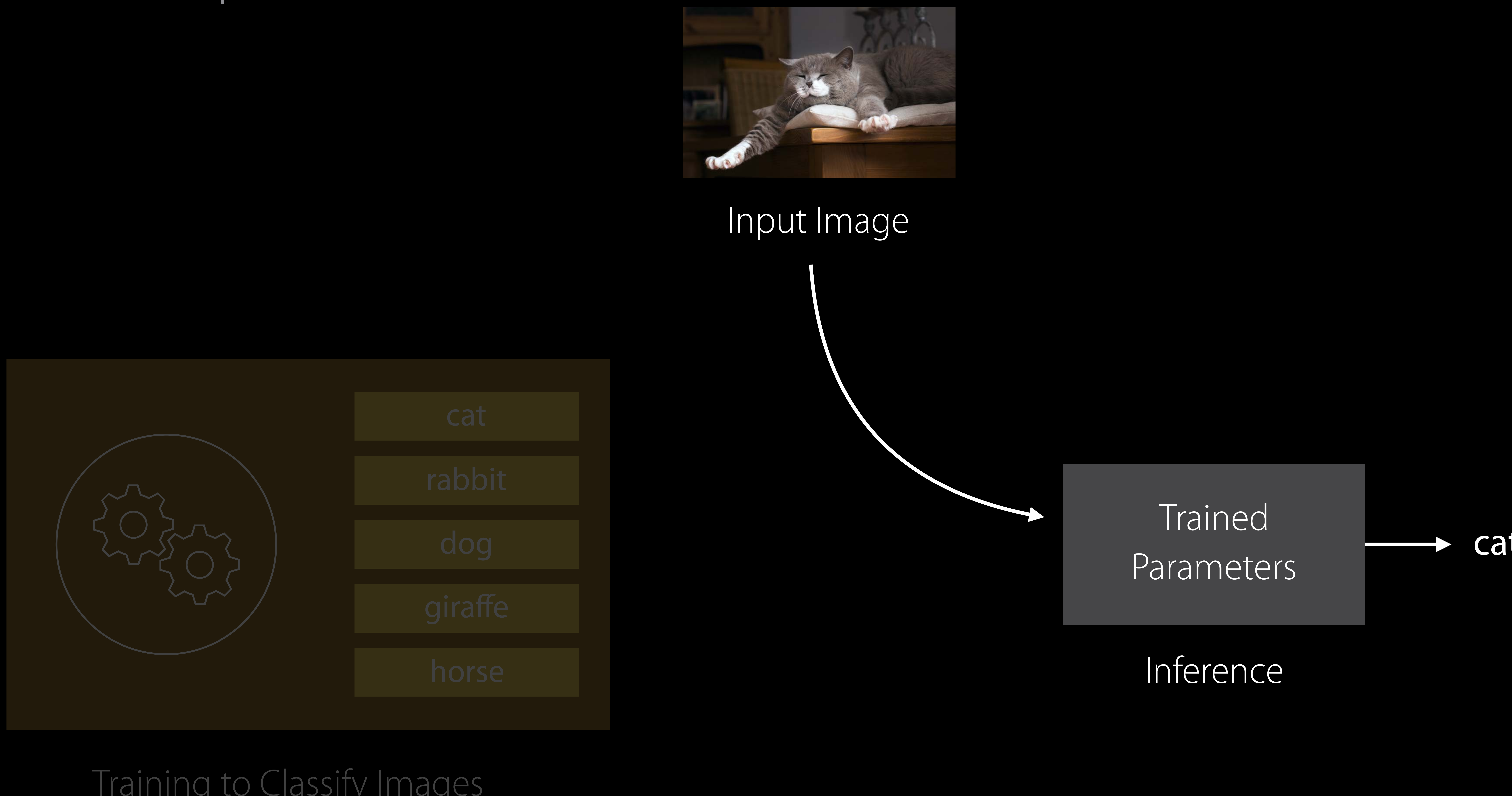
Second phase



Training to Classify Images

Training

Second phase



Convolutional Neural Networks (CNNs)

Biologically-inspired, resemble the visual cortex

Convolutional Neural Networks (CNNs)

Biologically-inspired, resemble the visual cortex

- Organized into layers of neurons
- Trained to recognize increasingly complex features

Convolutional Neural Networks (CNNs)

Biologically-inspired, resemble the visual cortex

- Organized into layers of neurons
- Trained to recognize increasingly complex features
  - First layers trained to recognize low-level features

Convolutional Neural Networks (CNNs)

Biologically-inspired, resemble the visual cortex

- Organized into layers of neurons
- Trained to recognize increasingly complex features
  - First layers trained to recognize low-level features
  - Subsequent layers trained to recognize higher-level features

Convolutional Neural Networks (CNNs)

Biologically-inspired, resemble the visual cortex

- Organized into layers of neurons
- Trained to recognize increasingly complex features
  - First layers trained to recognize low-level features
  - Subsequent layers trained to recognize higher-level features
  - Last layers combine all generated information to produce output

Building Blocks

NEW

Data

- MPSImage and MPSTemporaryImage

Building Blocks

NEW

Data

- MPSImage and MPSTemporaryImage

Layers

- Convolution
- Pooling
- Fully-Connected
- Neuron
- SoftMax
- Normalization

Building Blocks

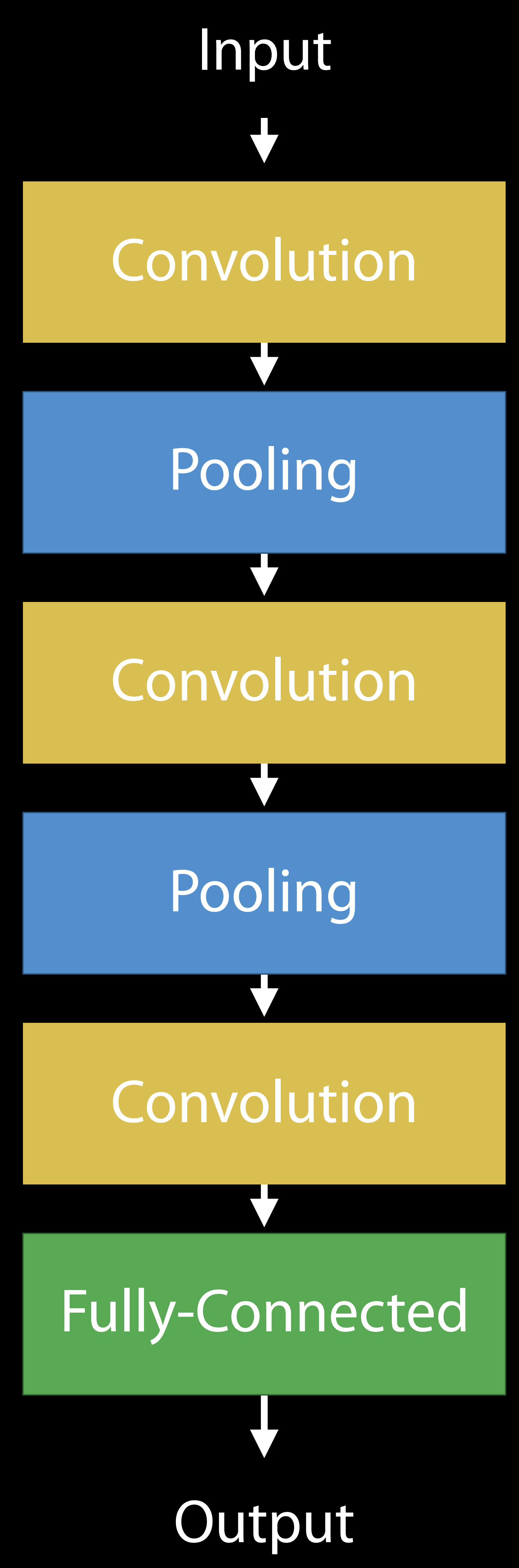
NEW

Data

- MPSImage and MPSTemporaryImage

Layers

- Convolution
- Pooling
- Fully-Connected
- Neuron
- SoftMax
- Normalization



Demo

Detecting a smile

Convolution Layer

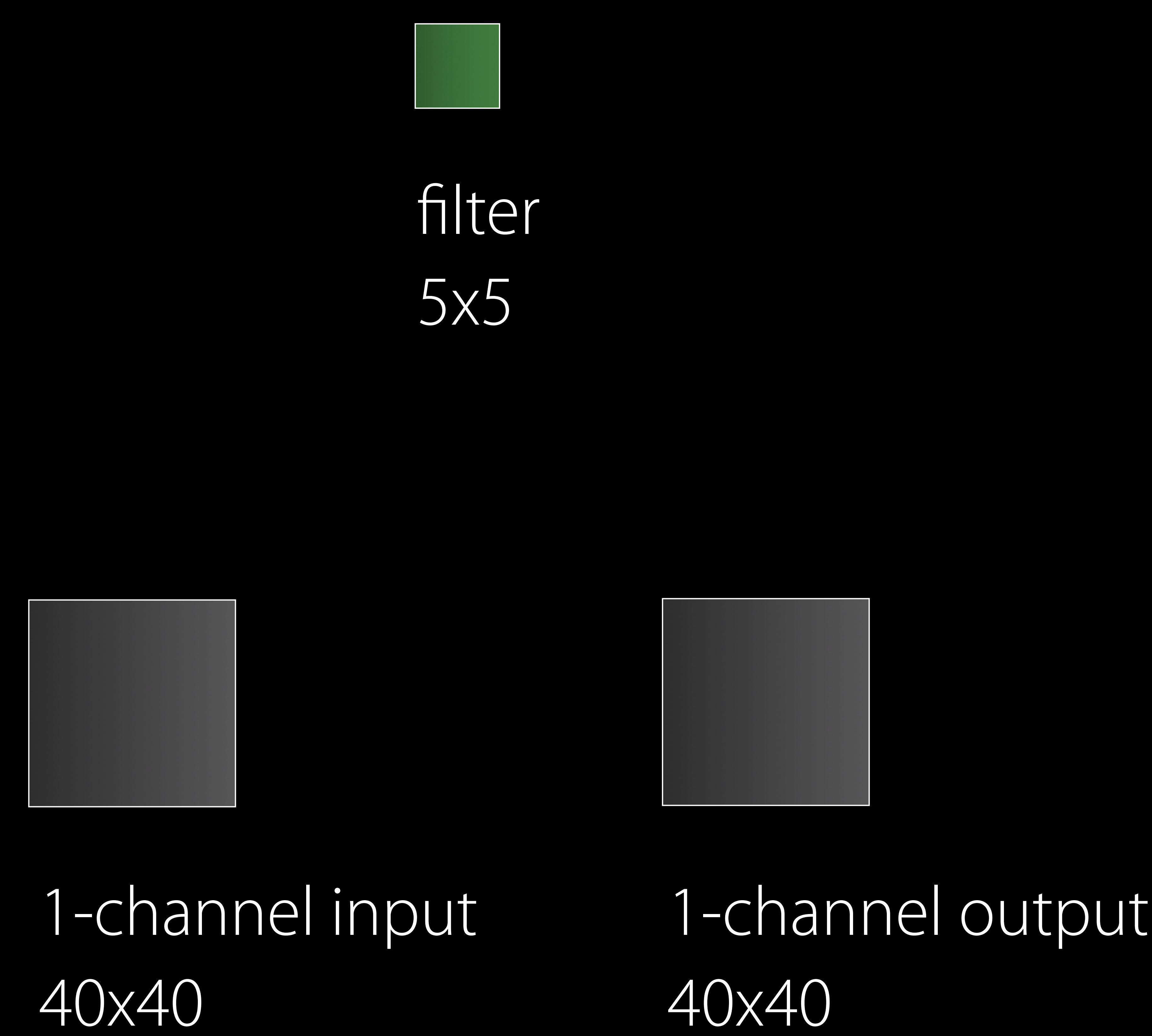
Definition

Core building block

Recognizes features in input

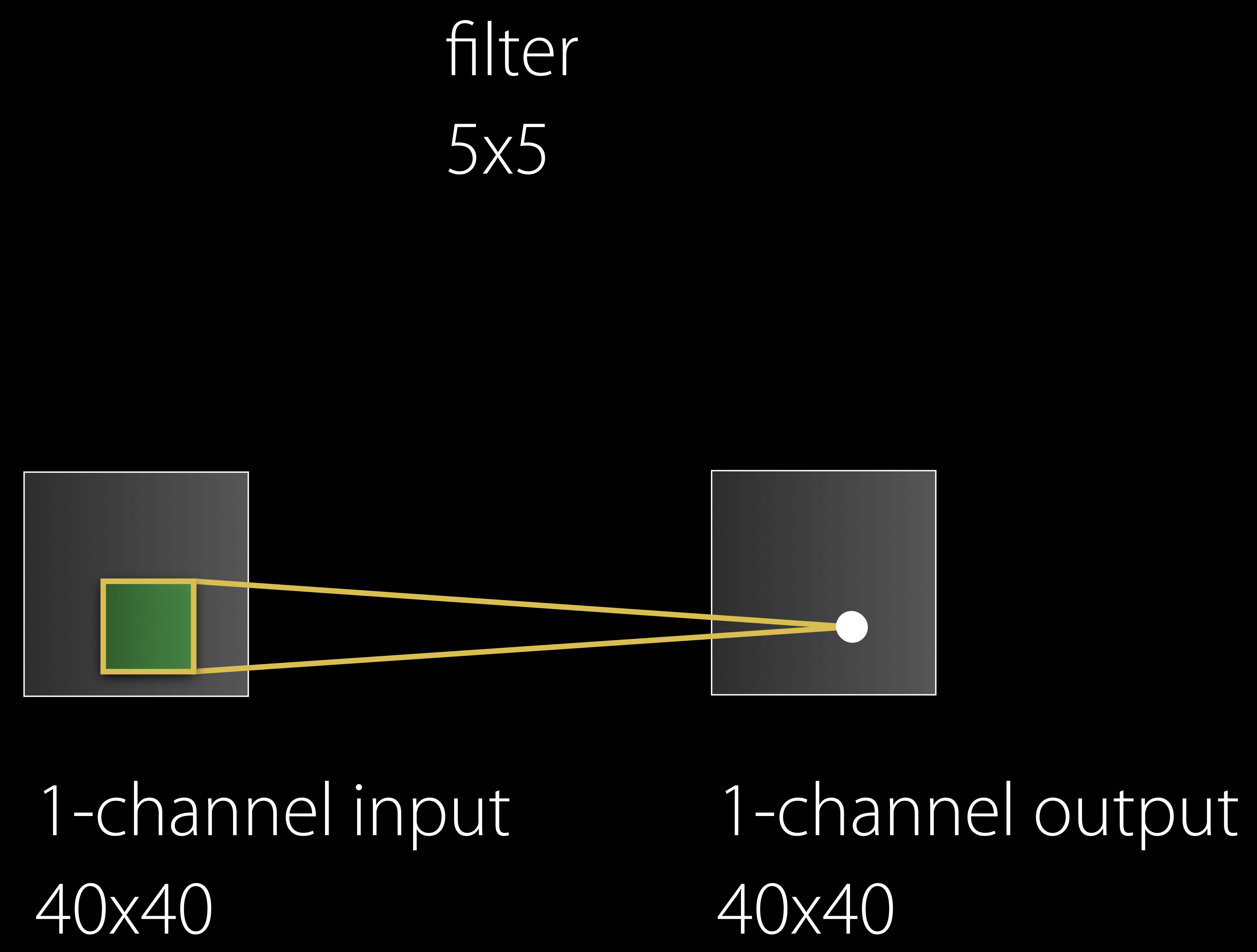
Convolution Layer

How it works



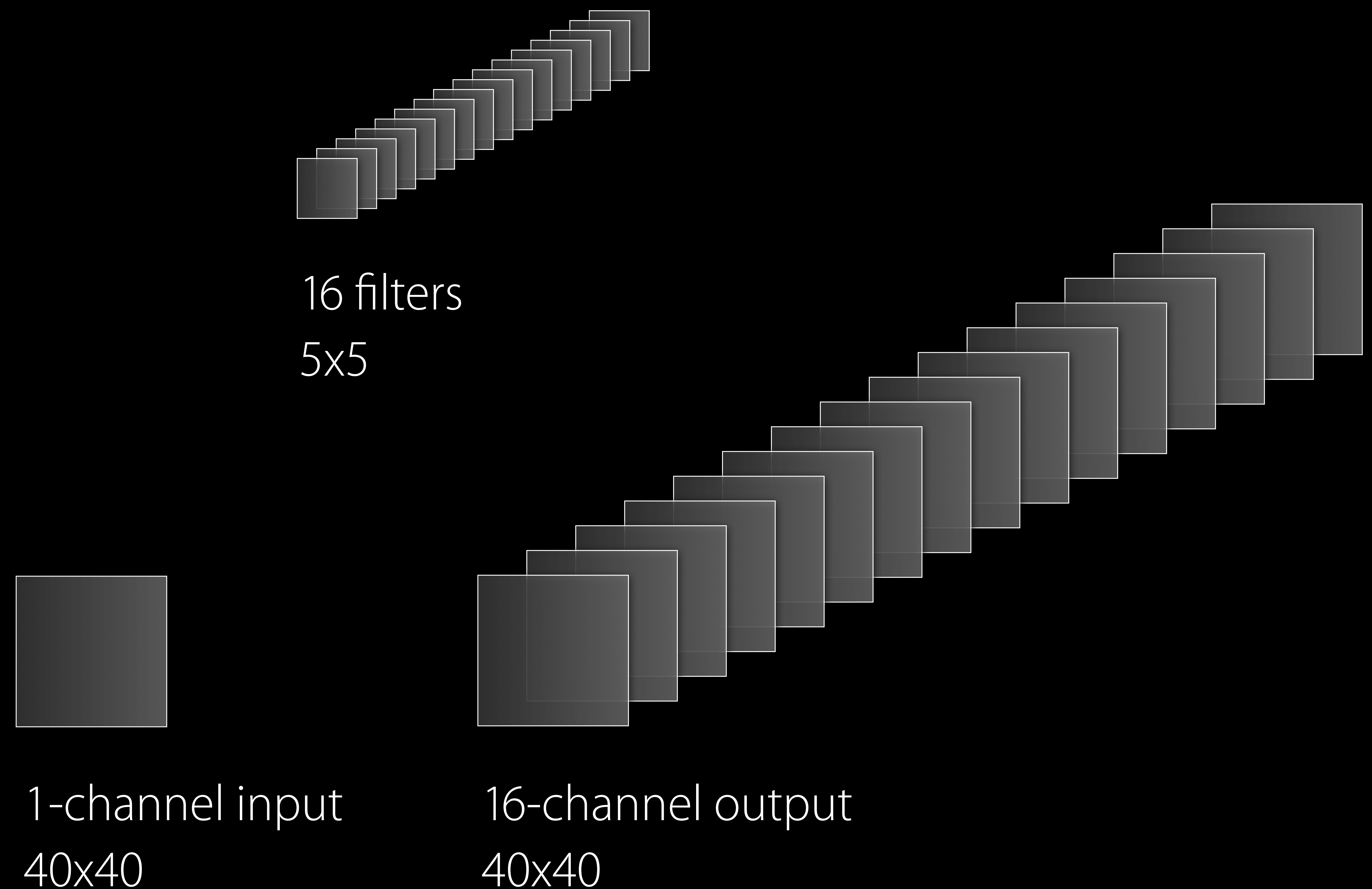
Convolution Layer

How it works



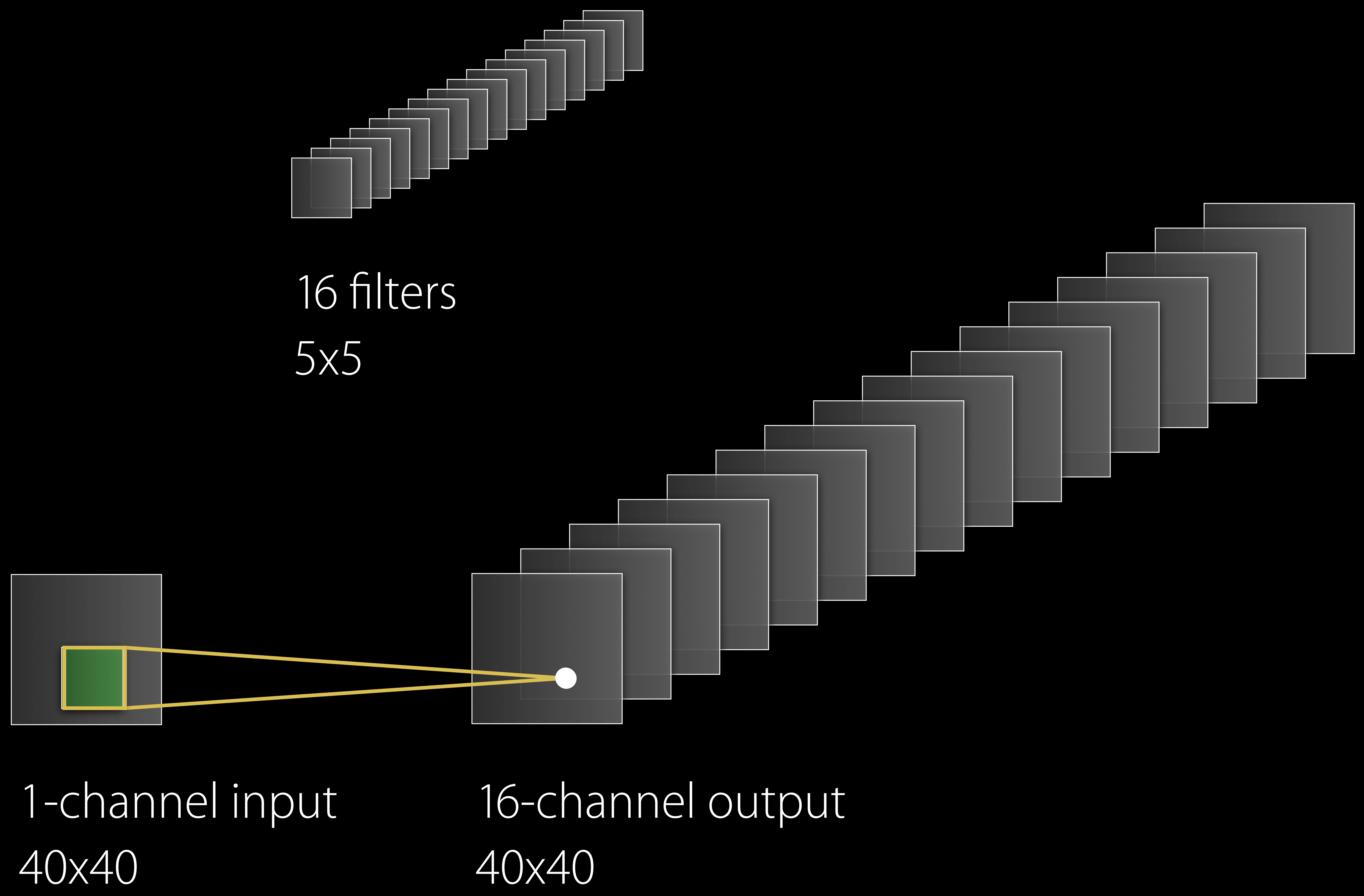
Convolution Layer

How it works



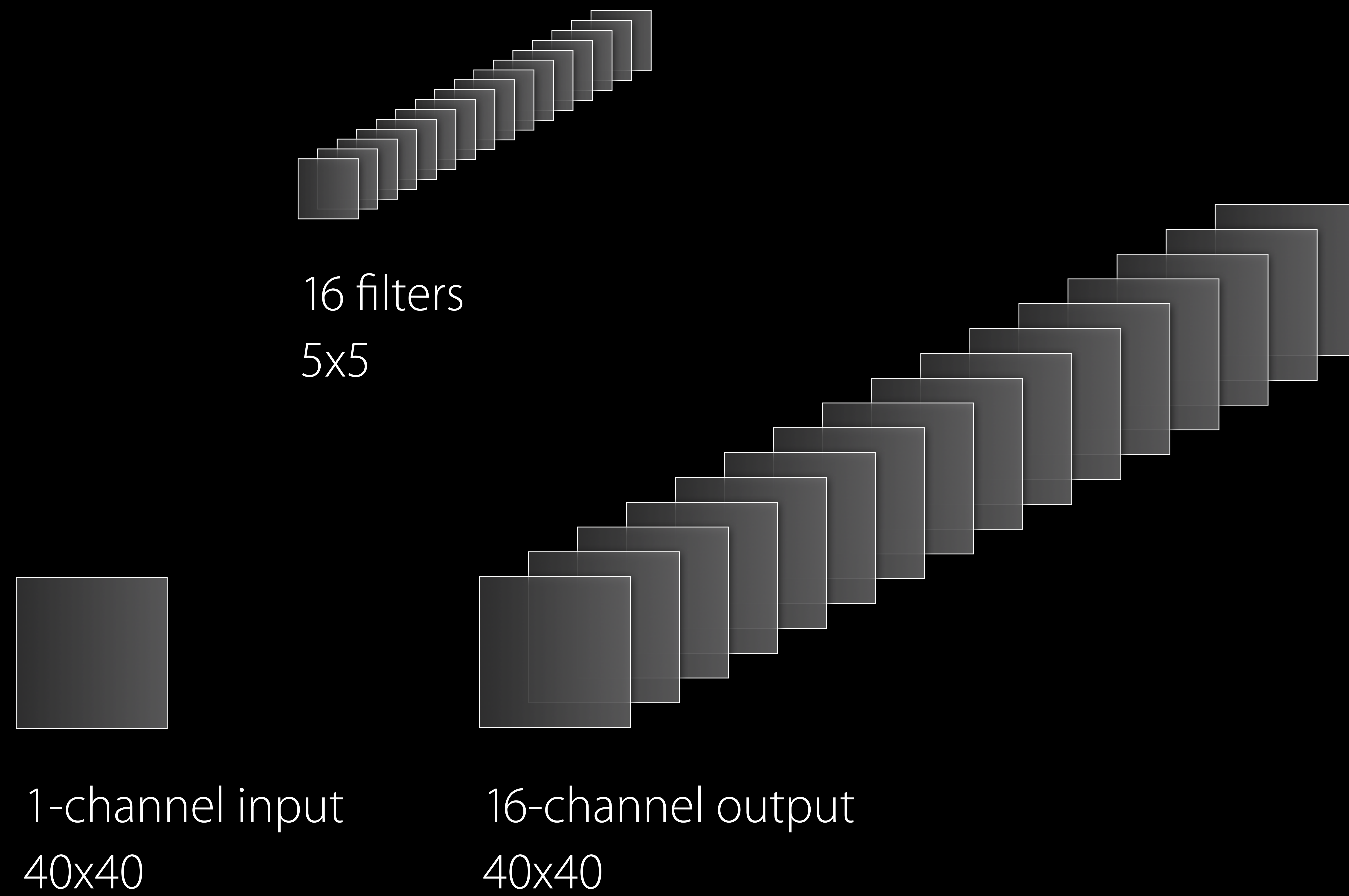
Convolution Layer

How it works



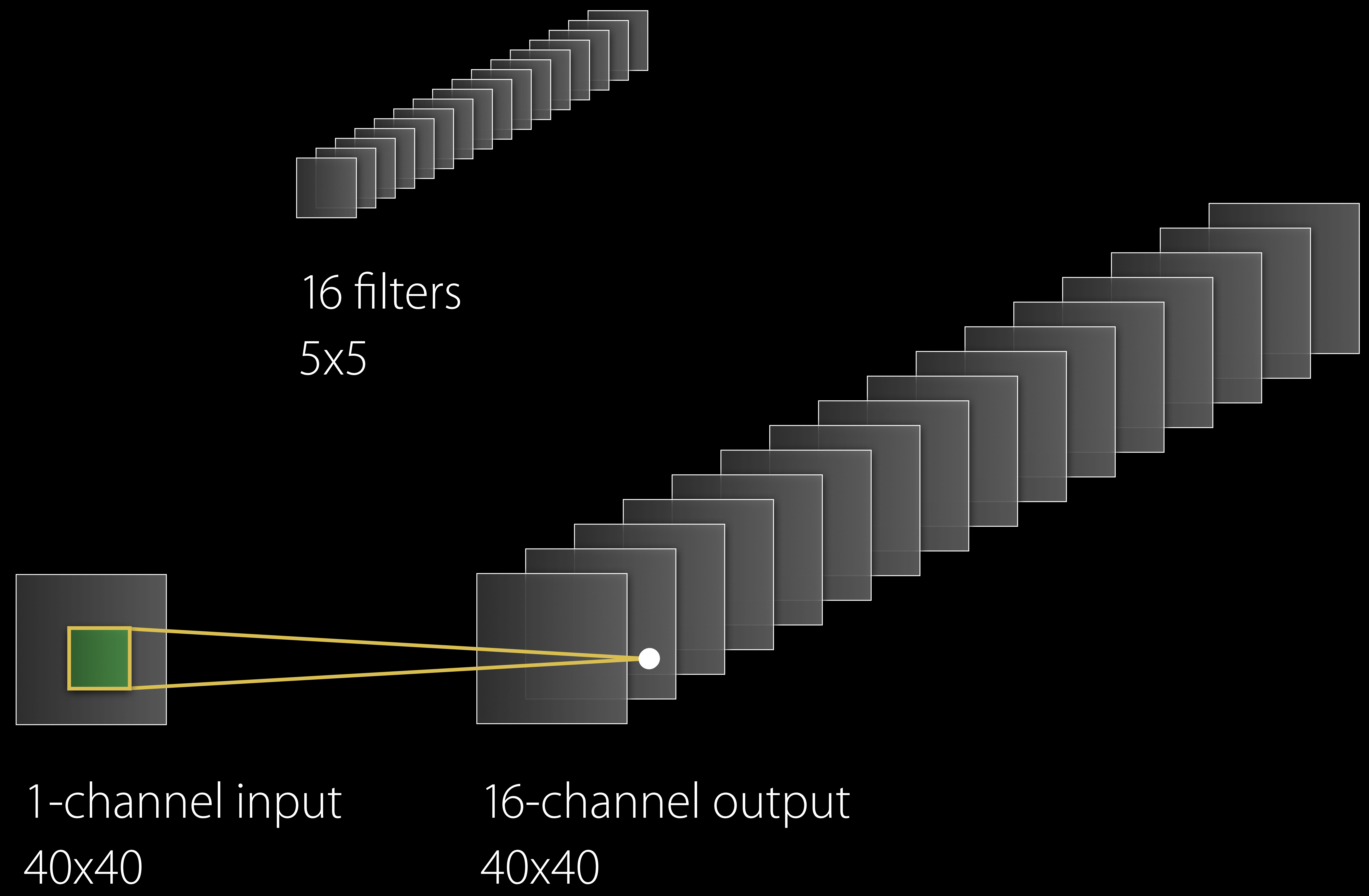
Convolution Layer

How it works



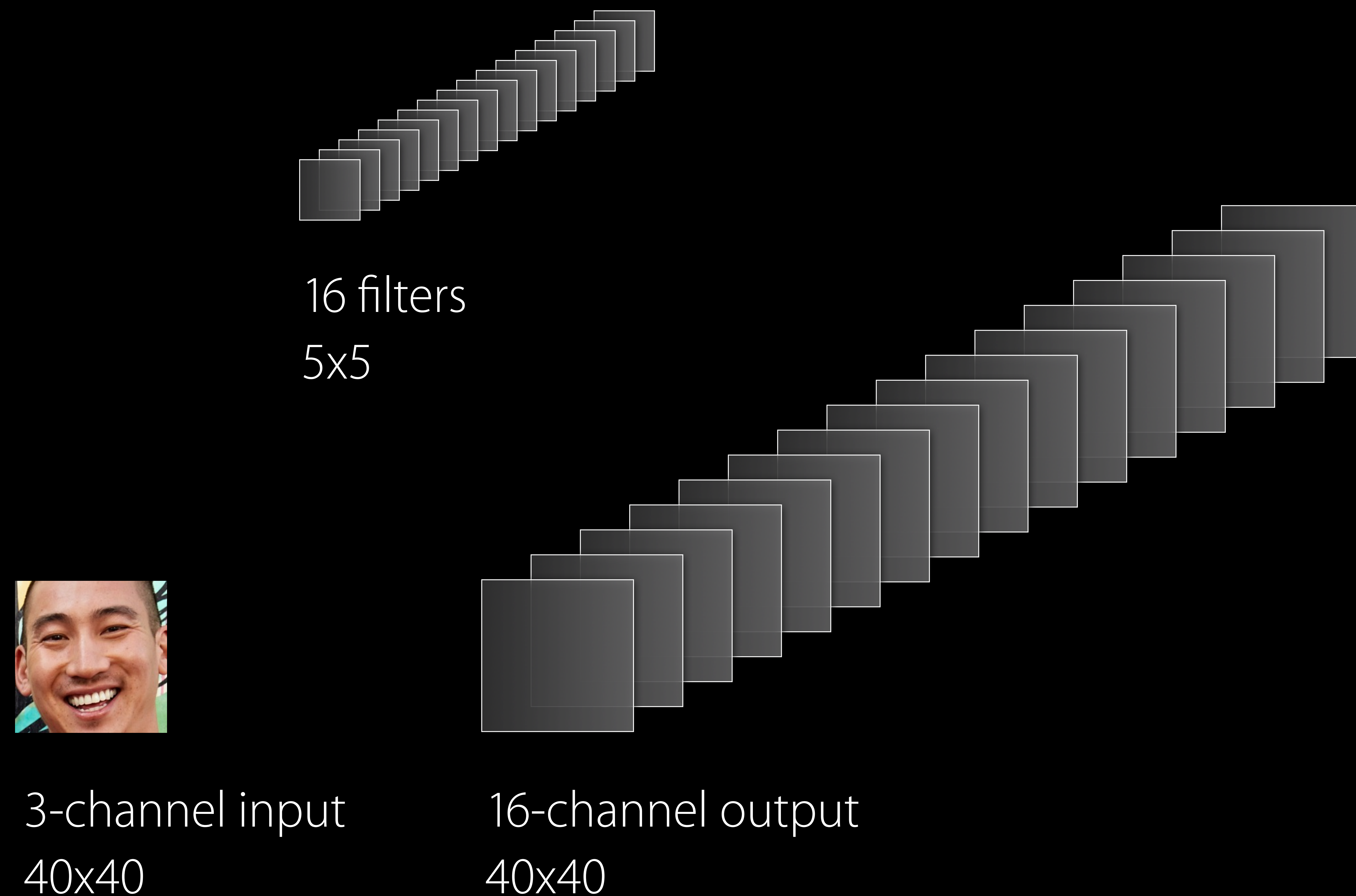
Convolution Layer

How it works



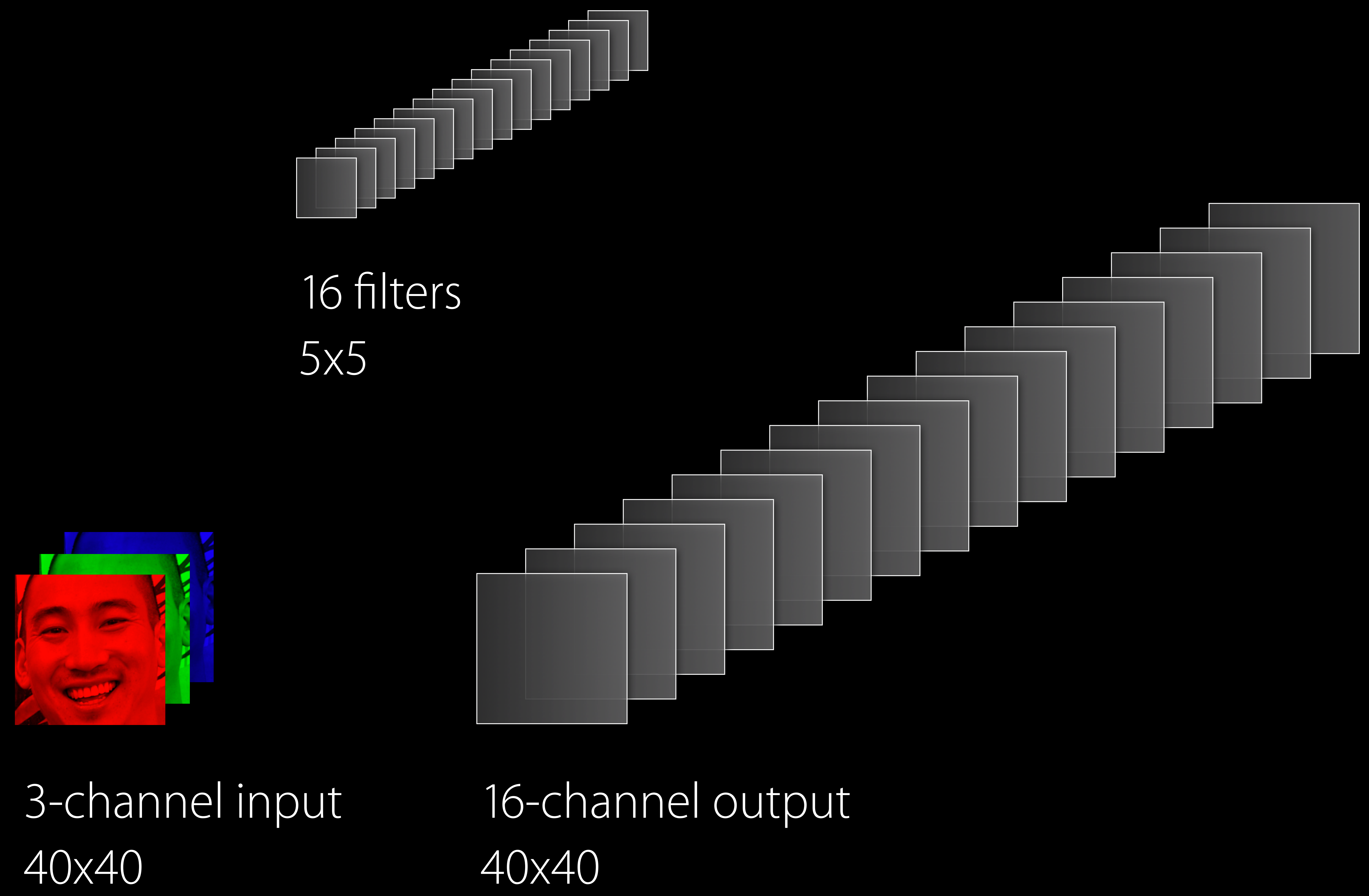
Convolution Layer

How it works



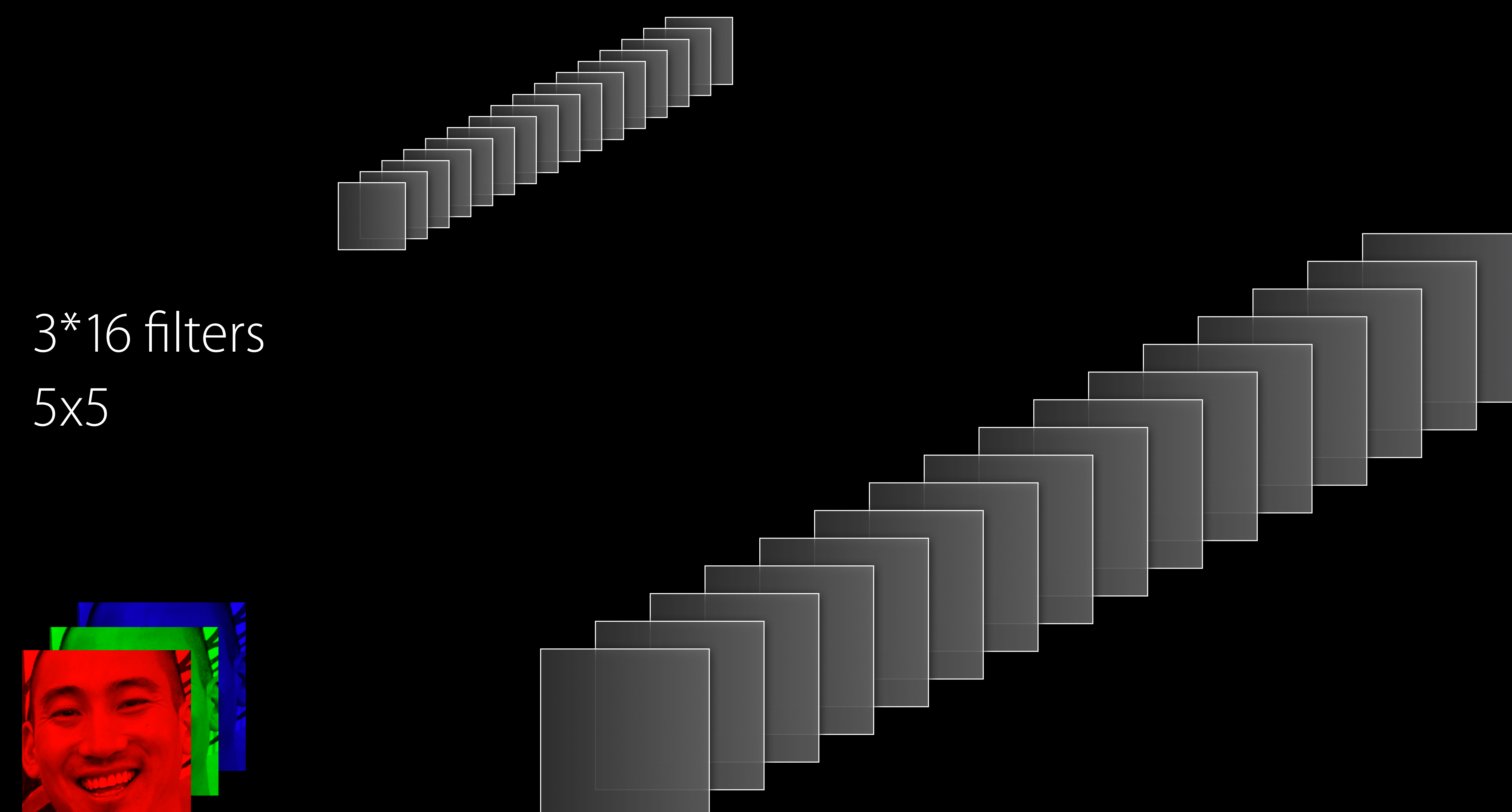
Convolution Layer

How it works



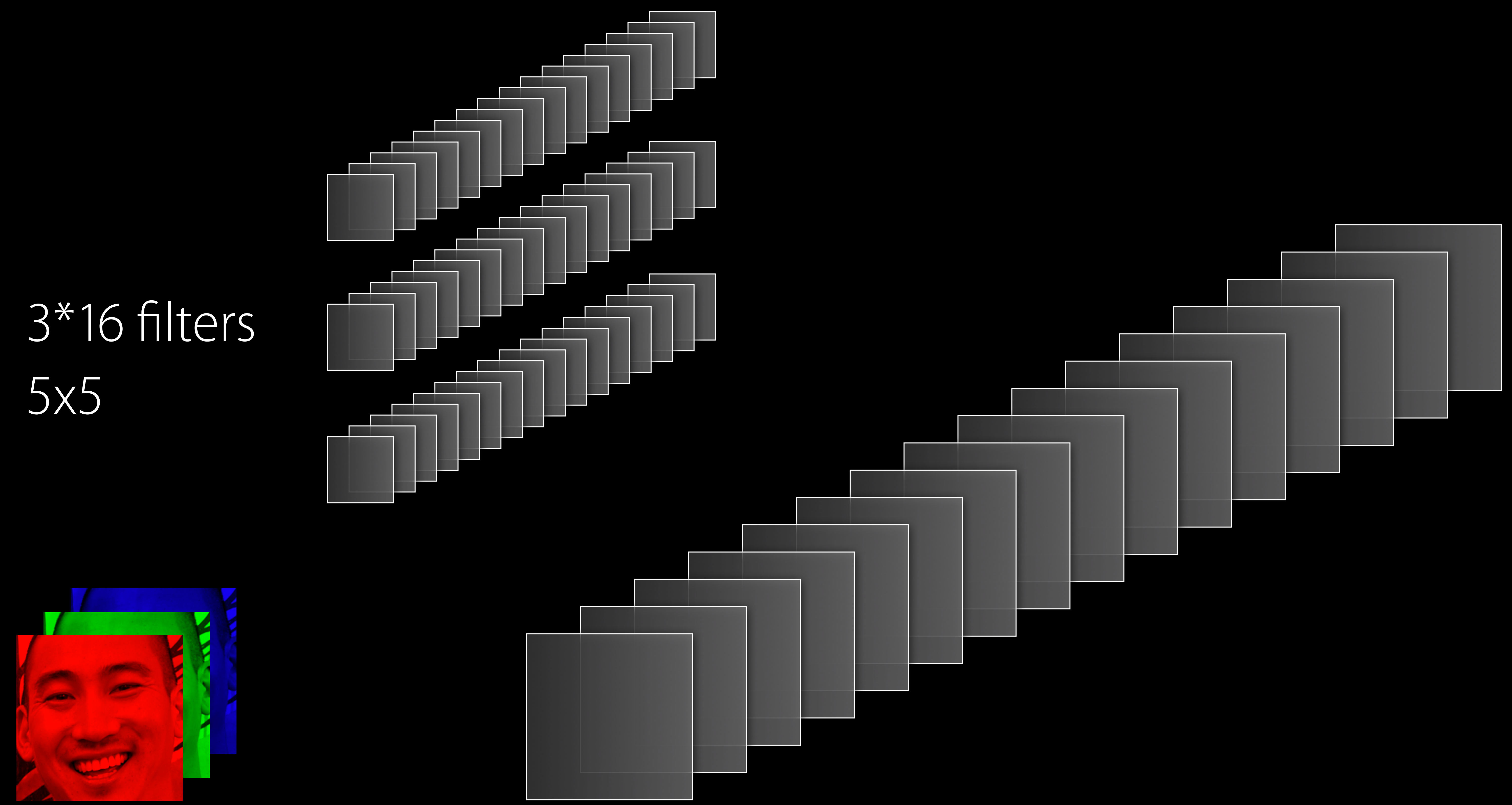
Convolution Layer

How it works



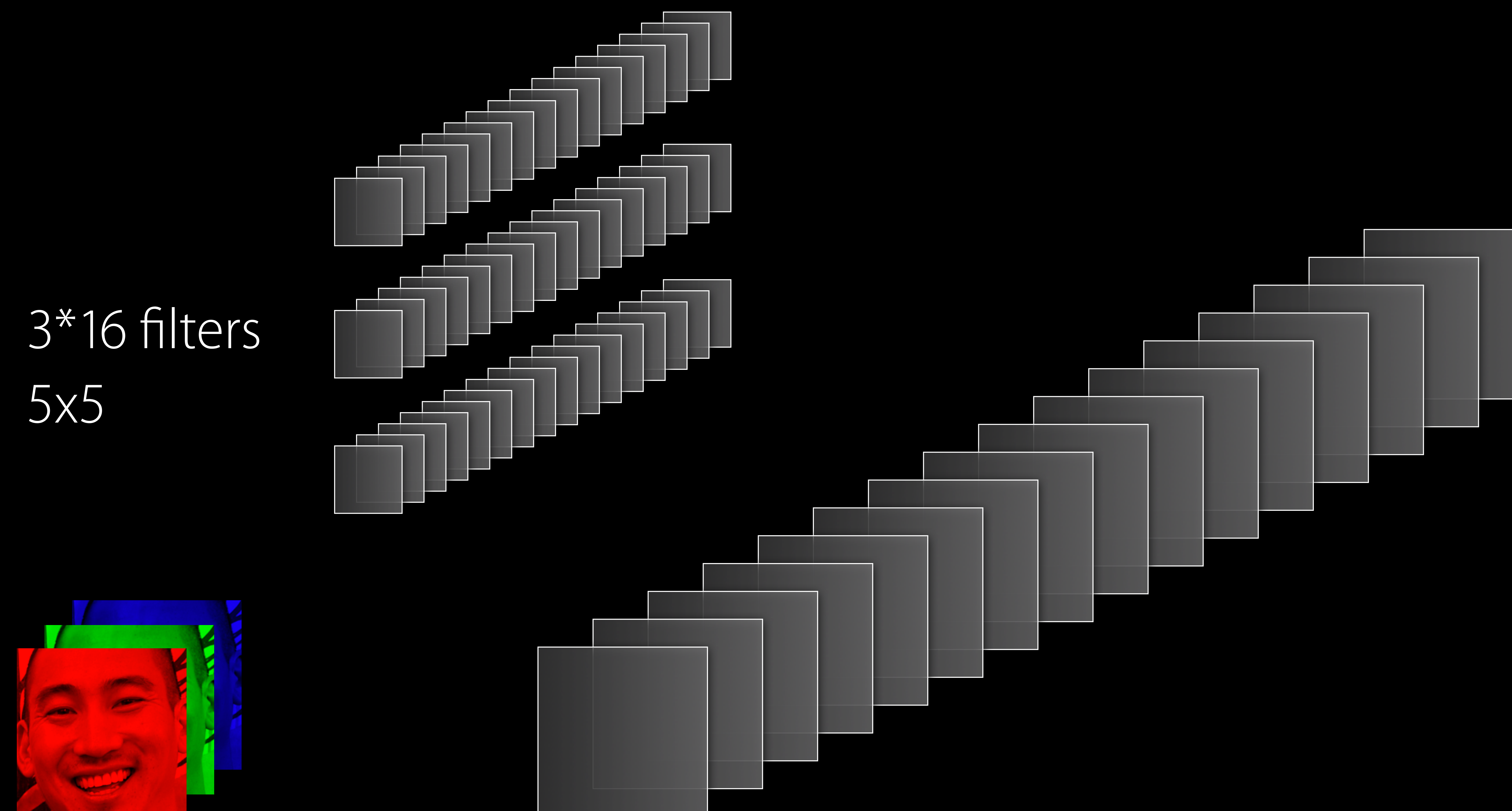
Convolution Layer

How it works



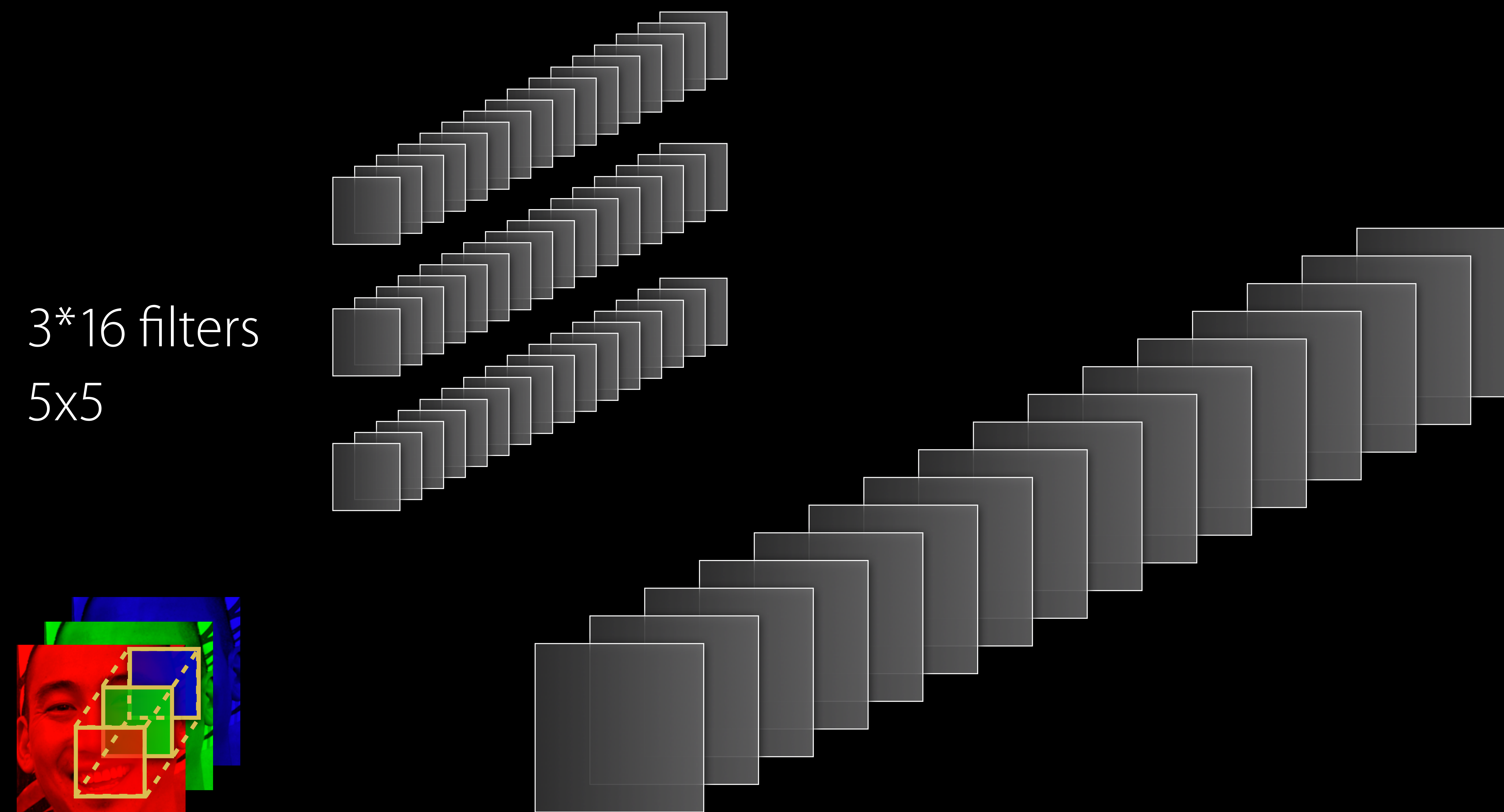
Convolution Layer

How it works



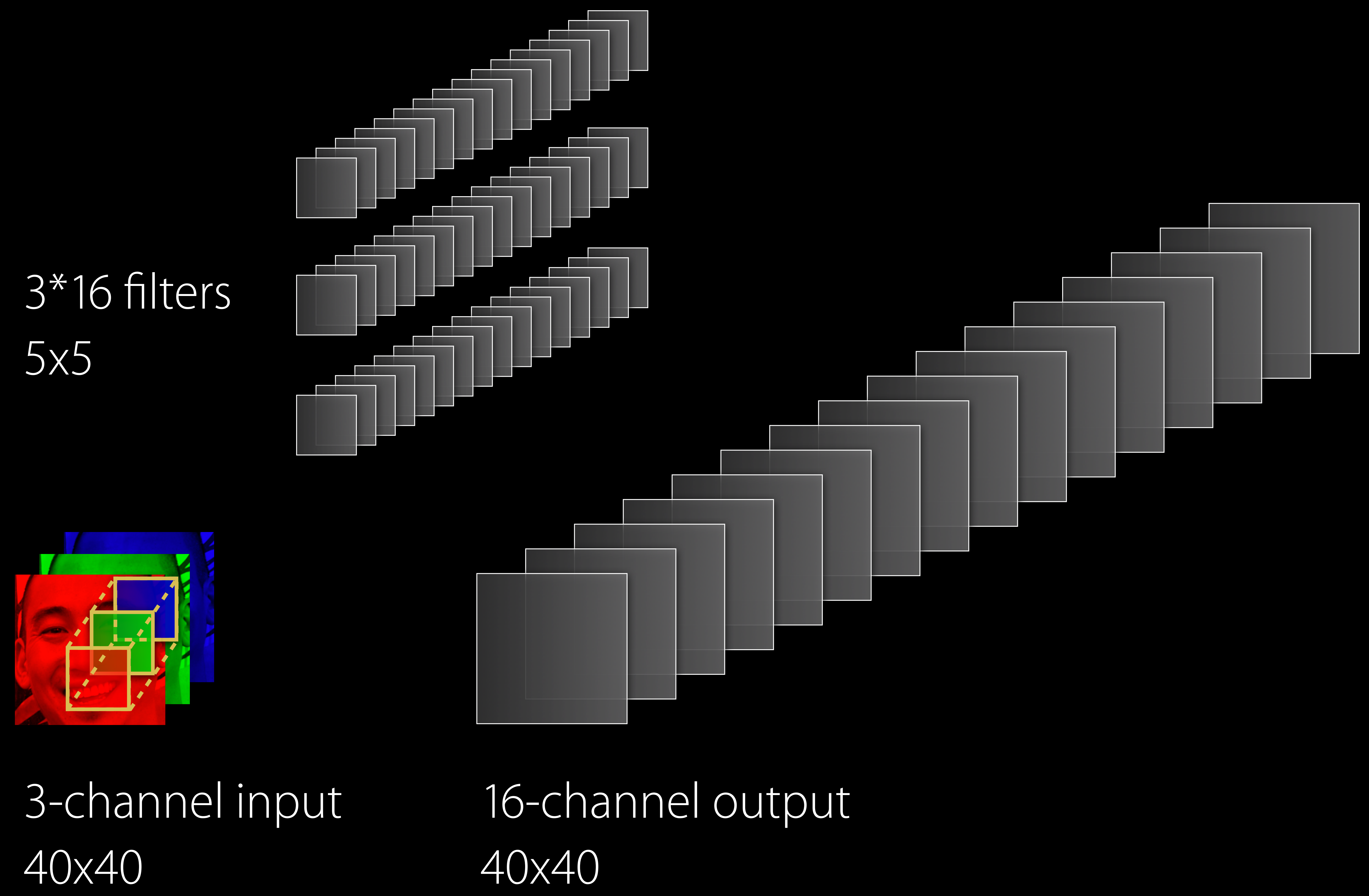
Convolution Layer

How it works



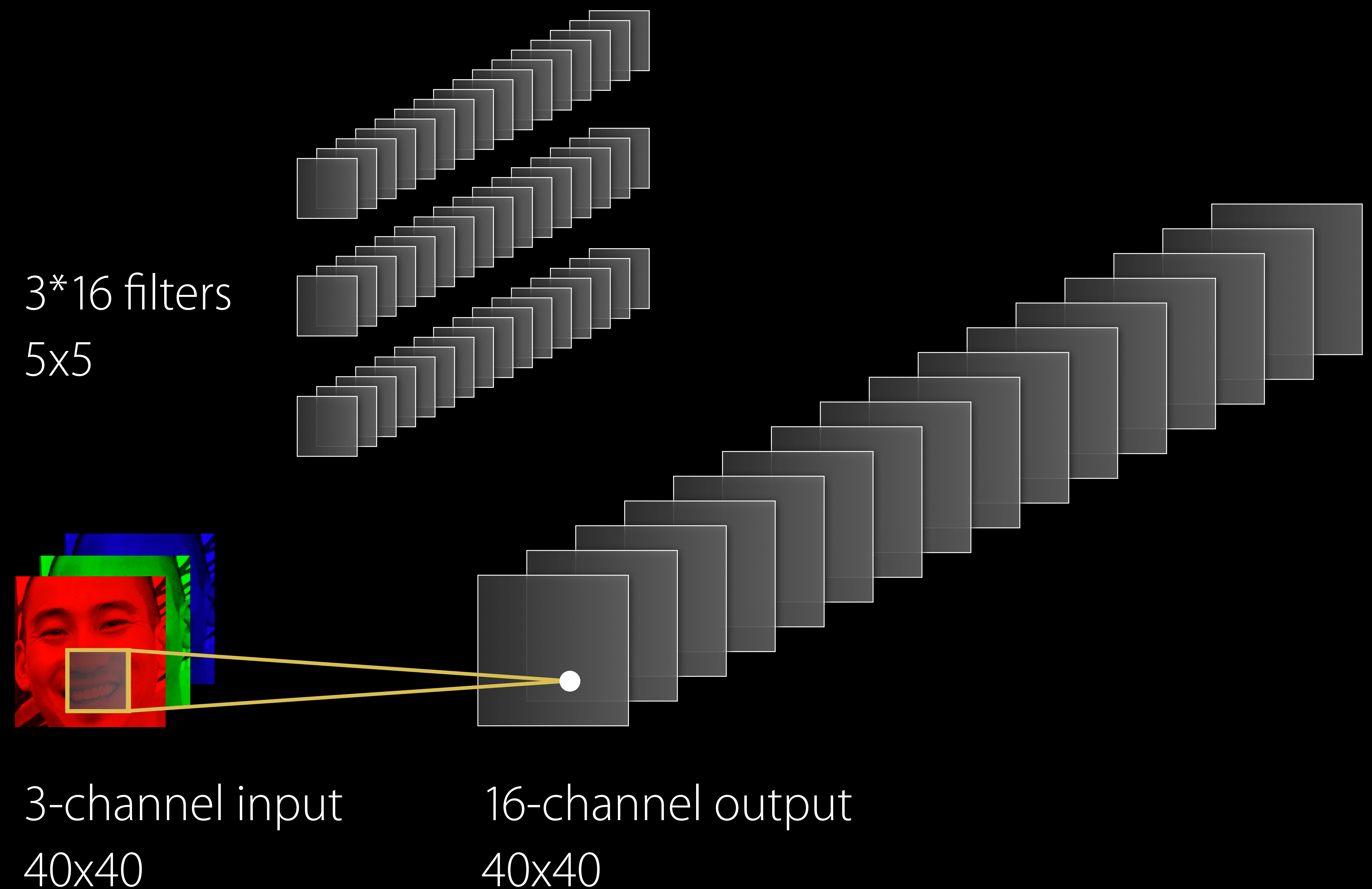
Convolution Layer

How it works



Convolution Layer

How it works



```swift
// Convolution Layer
// Code

let convDesc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
   inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
var conv = MPSCNNConvolution(device: device, convolutionDescriptor: convDesc,
   kernelWeights: featureFilters, biasTerms: convBias, flags: MPSCNNConvolutionFlags.none)
```

```swift
// Convolution Layer
// Code

let convDesc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
   inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
var conv = MPSCNNConvolution(device: device, convolutionDescriptor: convDesc,
   kernelWeights: featureFilters, biasTerms: convBias, flags: MPSCNNConvolutionFlags.none)
```

```swift
// Convolution Layer
// Code

let convDesc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
   inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
var conv = MPSCNNConvolution(device: device, convolutionDescriptor: convDesc,
   kernelWeights: featureFilters, biasTerms: convBias, flags: MPSCNNConvolutionFlags.none)
```

```swift
// Convolution Layer
// Code

let convDesc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
   inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
var conv = MPSCNNConvolution(device: device, convolutionDescriptor: convDesc,
   kernelWeights: featureFilters, biasTerms: convBias, flags: MPSCNNConvolutionFlags.none)
```

Pooling Layer

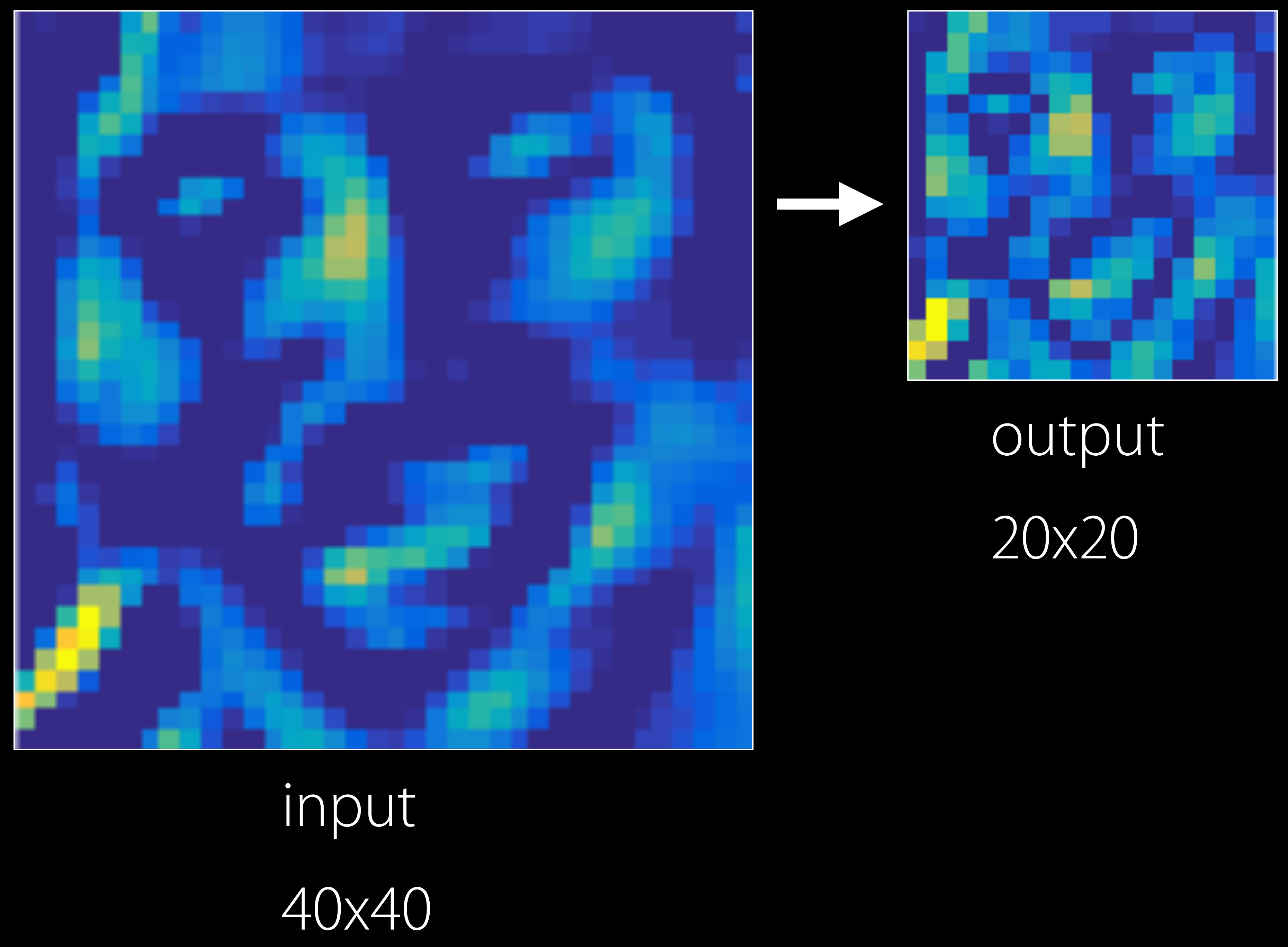
Definition

Reduces spatial size

Condenses information in a region of an image

Pooling operations

- Maximum
- Average



Pooling Layer

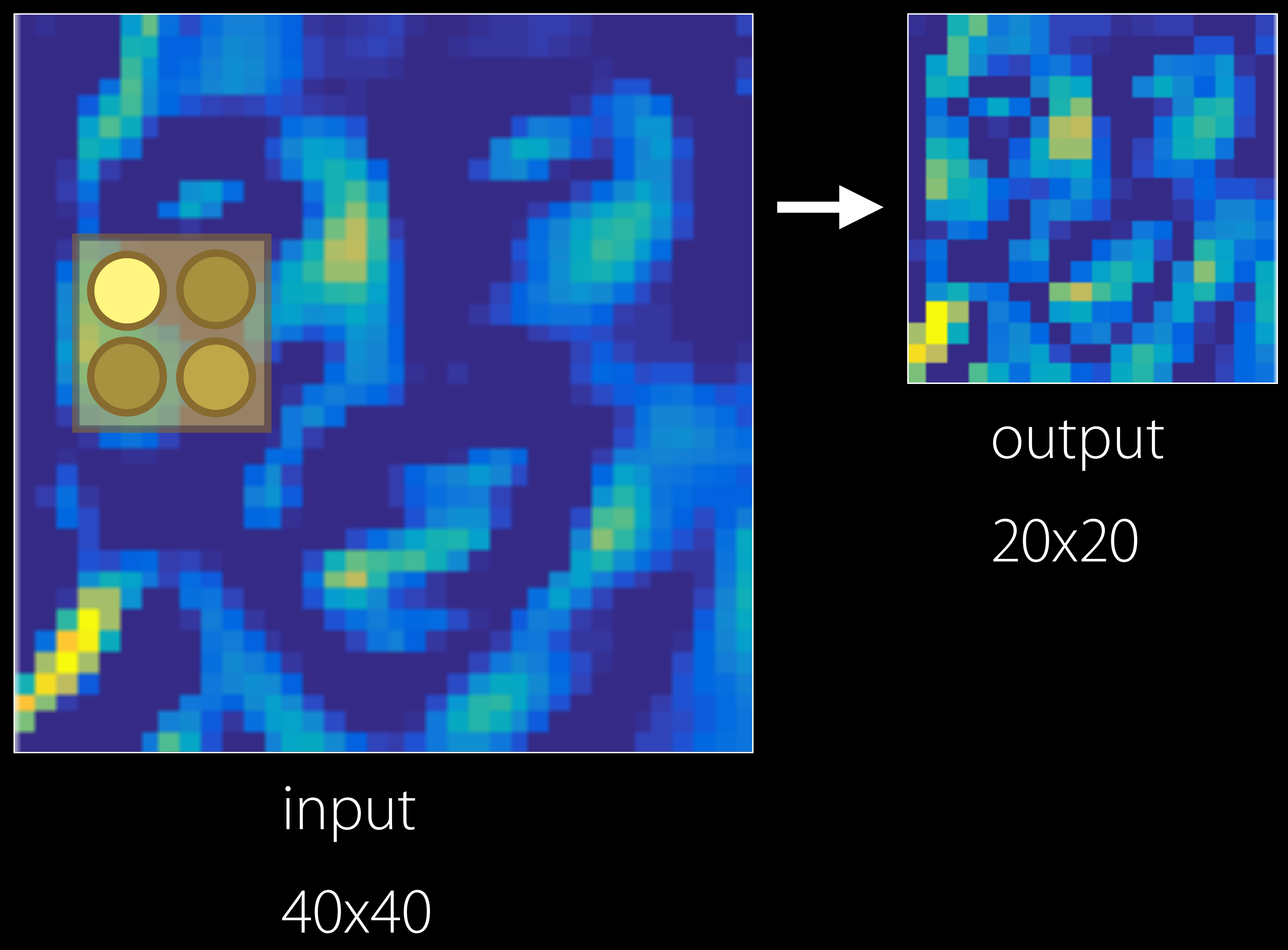
Definition

Reduces spatial size

Condenses information in a region of an image

Pooling operations

- Maximum
- Average



Pooling Layer

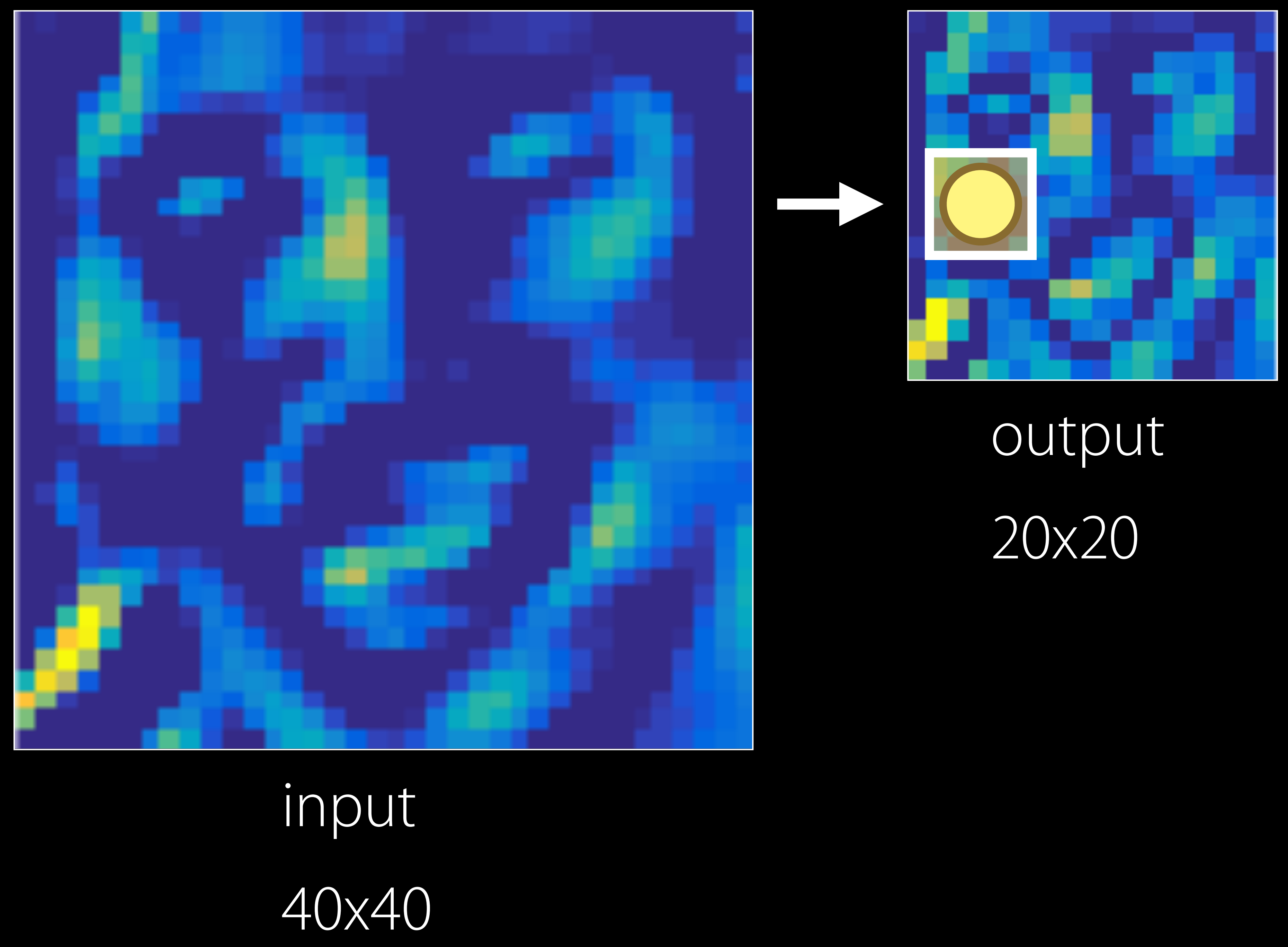
Definition

Reduces spatial size

Condenses information in a region of an image

Pooling operations

- Maximum
- Average



```swift
// Pooling Layer
// Code

var pool = MPSCNNPoolingMax(device: device, kernelWidth: 2, kernelHeight: 2,
   strideInPixelsX: 2, strideInPixelsY: 2)
```

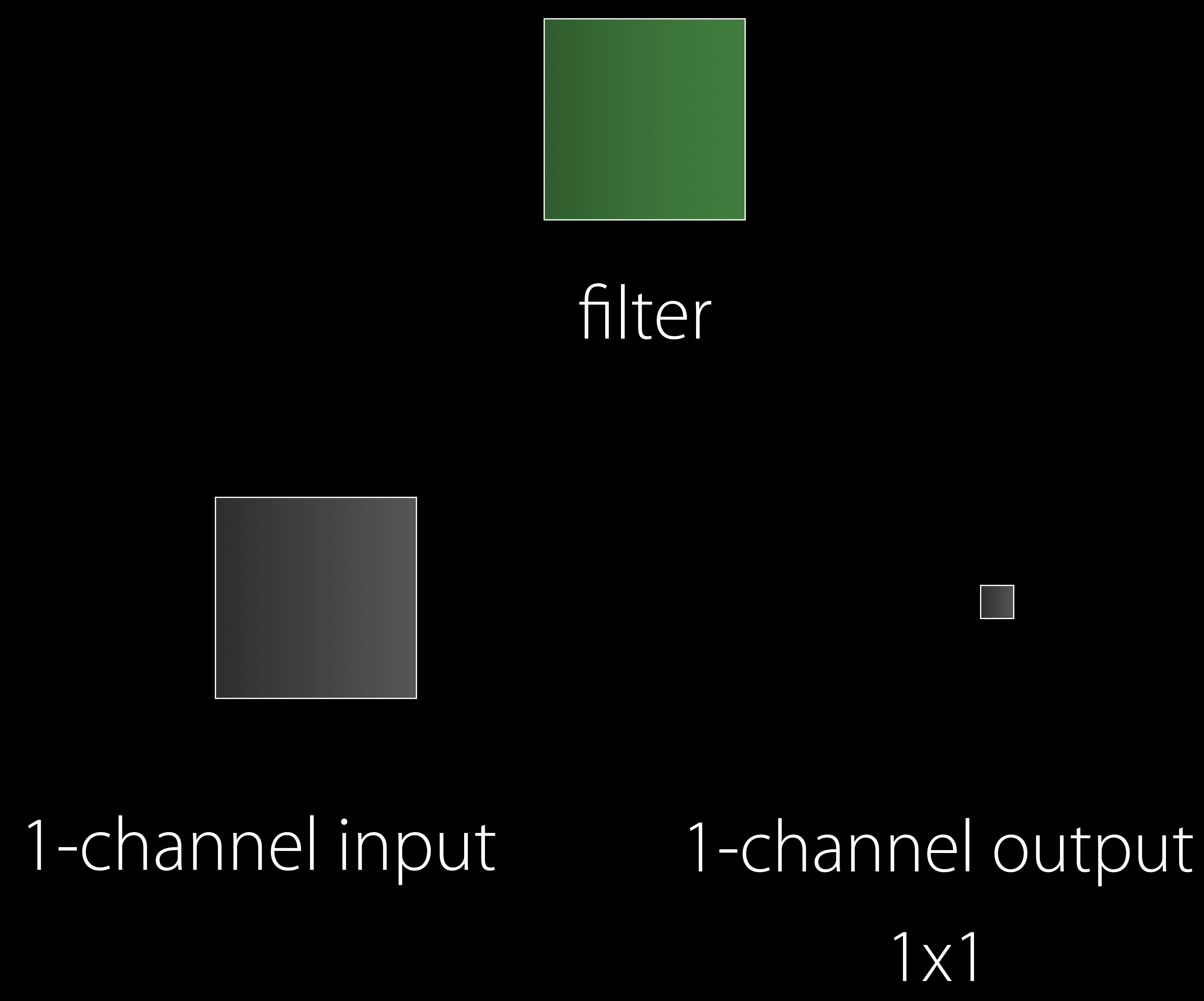
```swift
// Pooling Layer
// Code

var pool = MPSCNNPoolingMax(device: device, kernelWidth: 2, kernelHeight: 2,
   strideInPixelsX: 2, strideInPixelsY: 2)
```

```swift
// Pooling Layer
// Code

var pool = MPSCNNPoolingMax(device: device, kernelWidth: 2, kernelHeight: 2,
   strideInPixelsX: 2, strideInPixelsY: 2)
```

Fully-Connected Layer

Definition

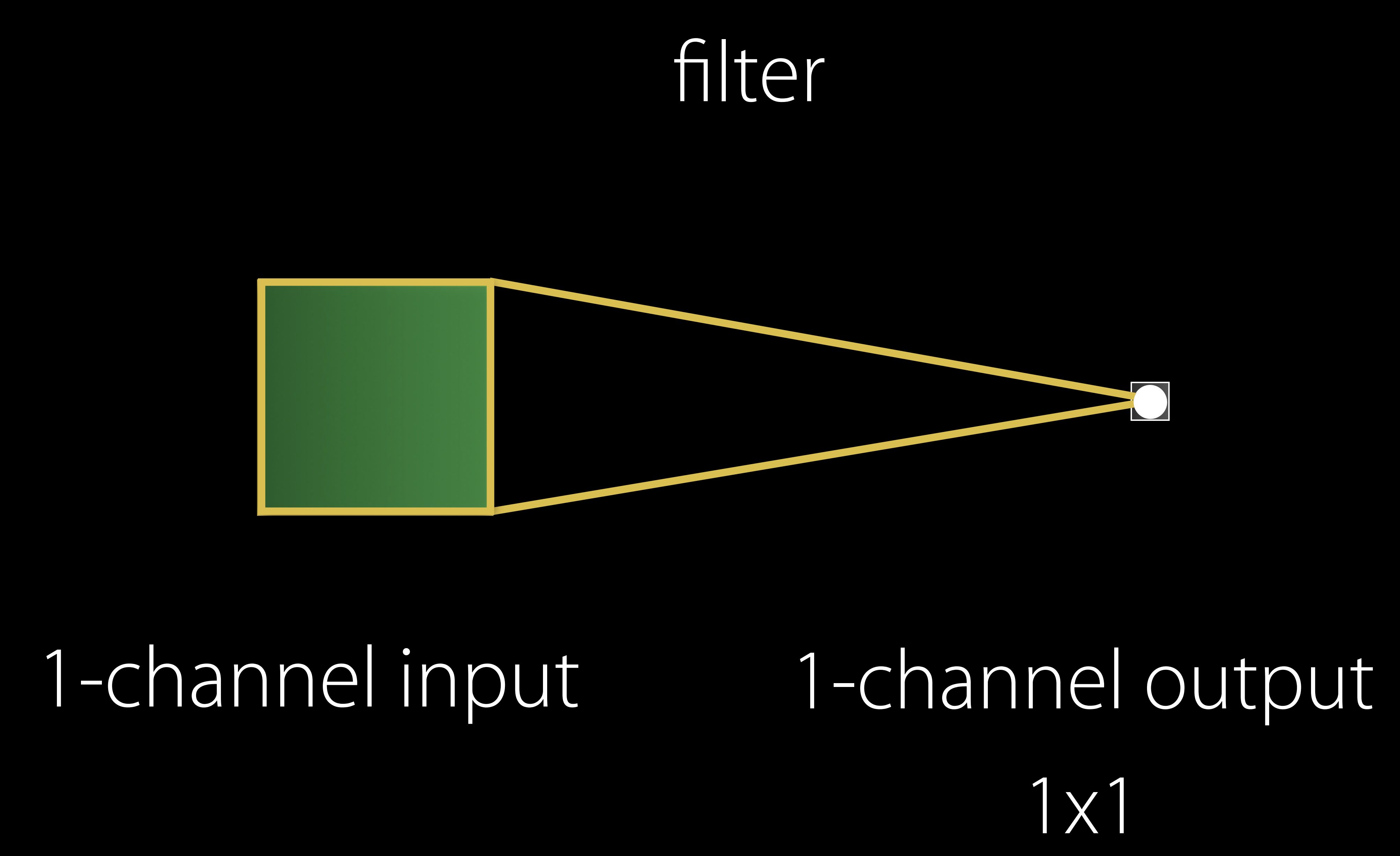
Convolution layer, where filter size == input size



Fully-Connected Layer

Definition

Convolution layer, where filter size == input size



```swift
// Fully-Connected Layer
// Code

let fcDesc = MPSCNNConvolutionDescriptor(kernelWidth: inputWidth, kernelHeight: inputHeight,
   inputFeatureChannels: 256, outputFeatureChannels: 1)
var fc = MPSCNNFullyConnected(device: device, convolutionDescriptor: fcDesc,
   kernelWeights: fcFeatureFilters, biasTerms: fcBias, flags: MPSCNNConvolutionFlags.none)
```

```swift
// Fully-Connected Layer
// Code

let fcDesc = MPSCNNConvolutionDescriptor(kernelWidth: inputWidth, kernelHeight: inputHeight,
   inputFeatureChannels: 256, outputFeatureChannels: 1)
var fc = MPSCNNFullyConnected(device: device, convolutionDescriptor: fcDesc,
   kernelWeights: fcFeatureFilters, biasTerms: fcBias, flags: MPSCNNConvolutionFlags.none)
```

```swift
// Fully-Connected Layer
// Code

let fcDesc = MPSCNNConvolutionDescriptor(kernelWidth: inputWidth, kernelHeight: inputHeight,
   inputFeatureChannels: 256, outputFeatureChannels: 1)
var fc = MPSCNNFullyConnected(device: device, convolutionDescriptor: fcDesc,
   kernelWeights: fcFeatureFilters, biasTerms: fcBias, flags: MPSCNNConvolutionFlags.none)
```

Additional Layers

Neuron

SoftMax

Normalization

MPSImage

What is it?

MPSImage

What is it?

MTLTexture

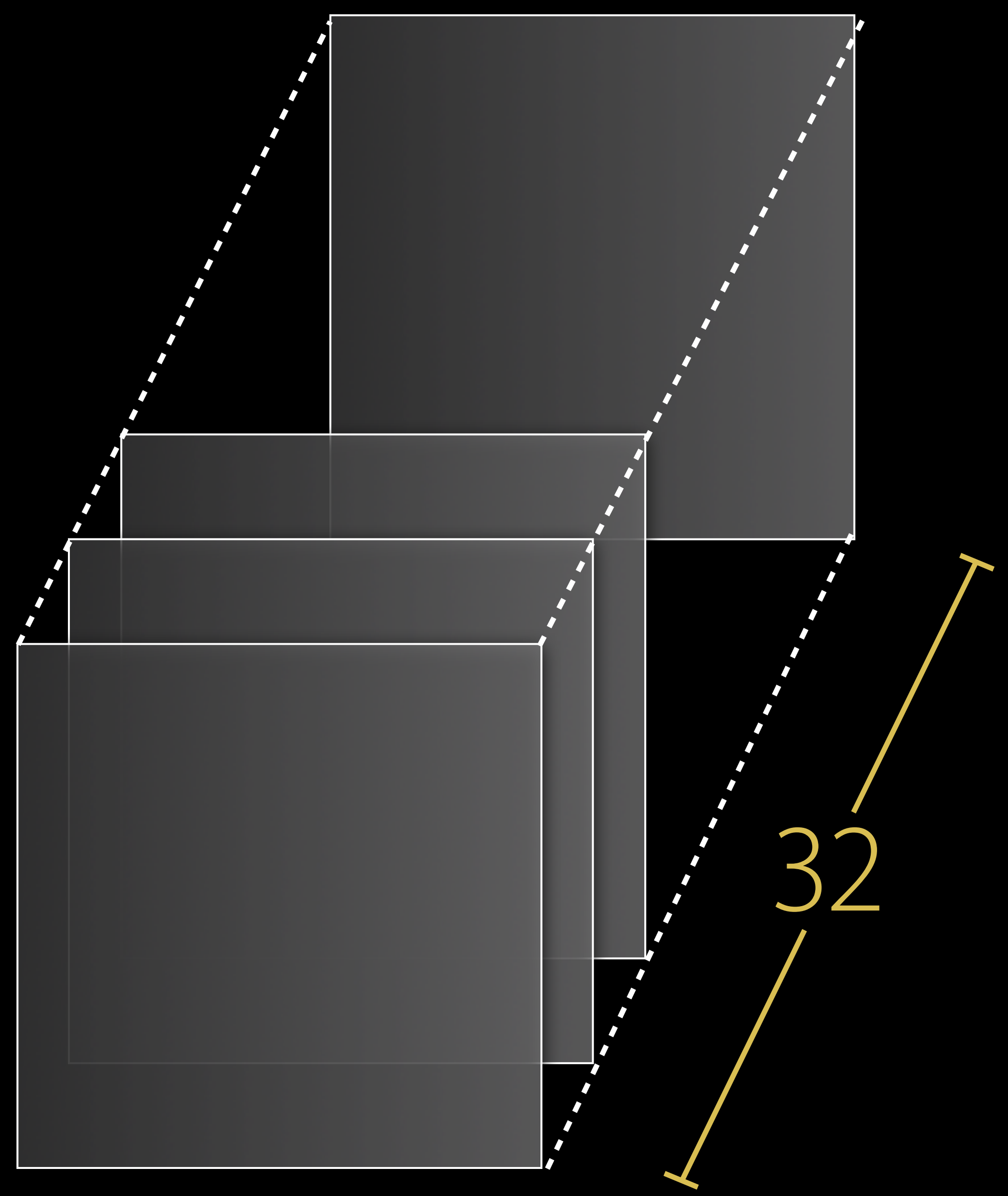
RGBA, 4 channels

MPSImage

What is it?

MTLTexture

RGBA, 4 channels



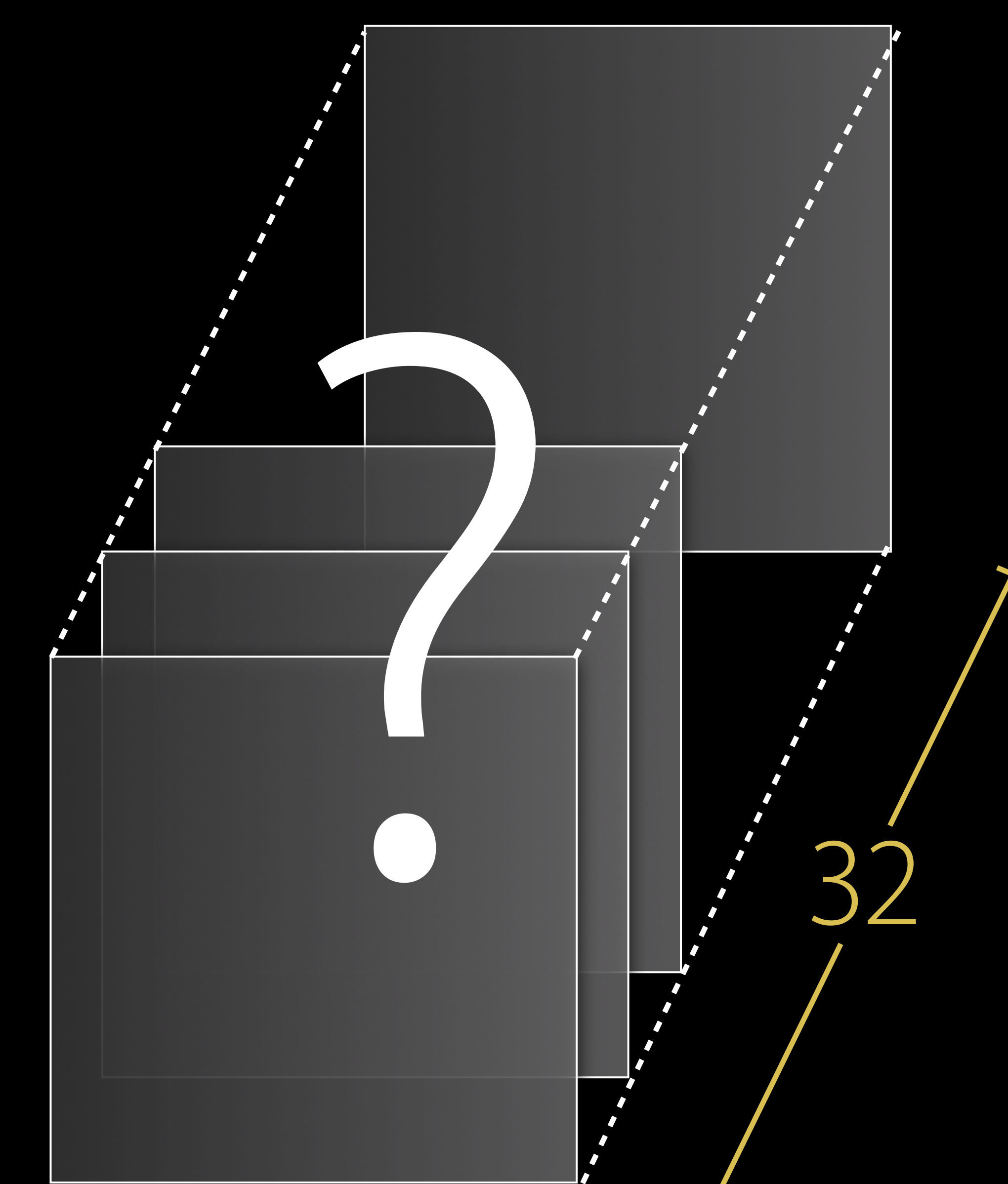
32 channels

MPSImage

What is it?

MTLTexture

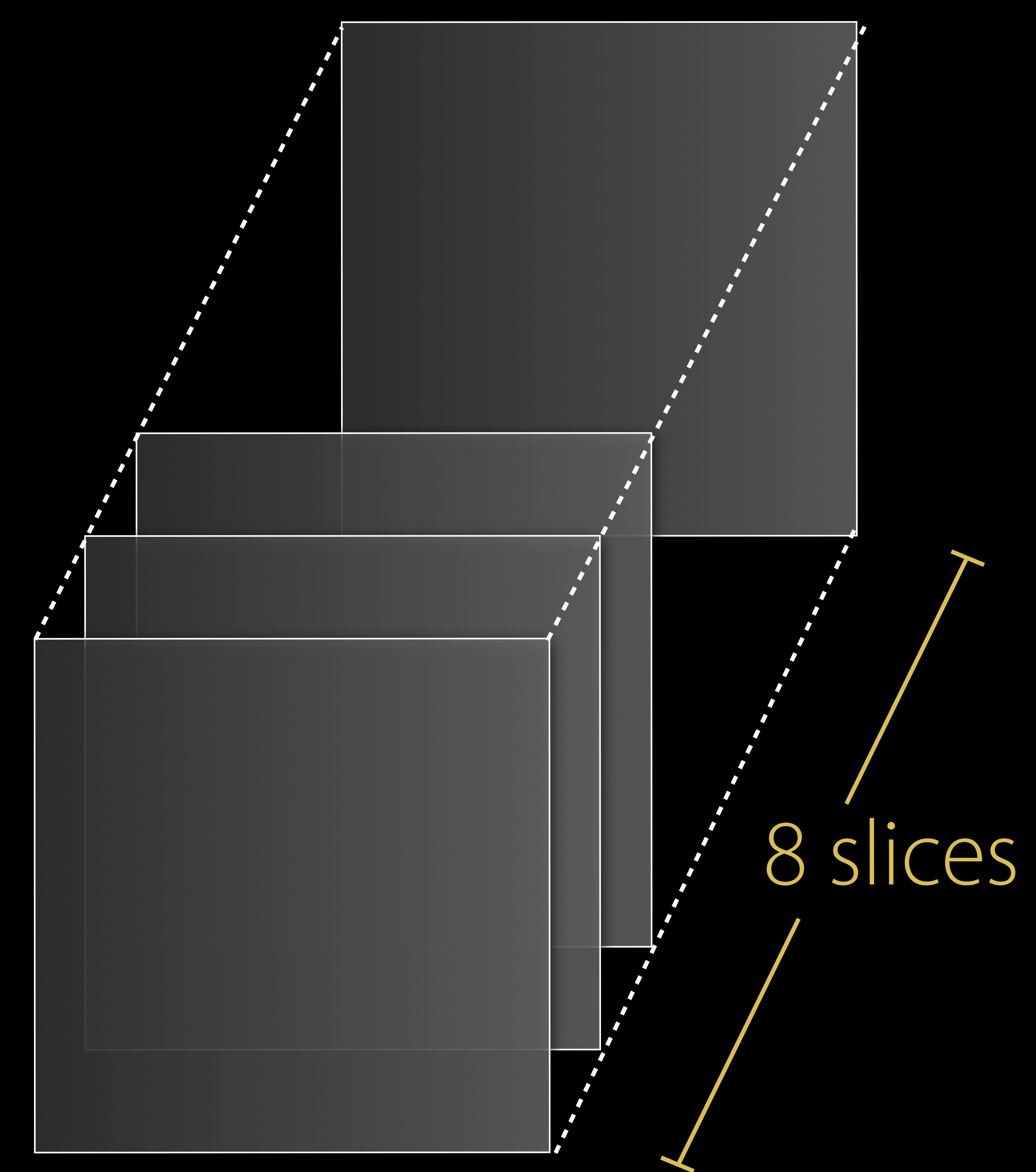
RGBA, 4 channels



32 channels

MPSImage

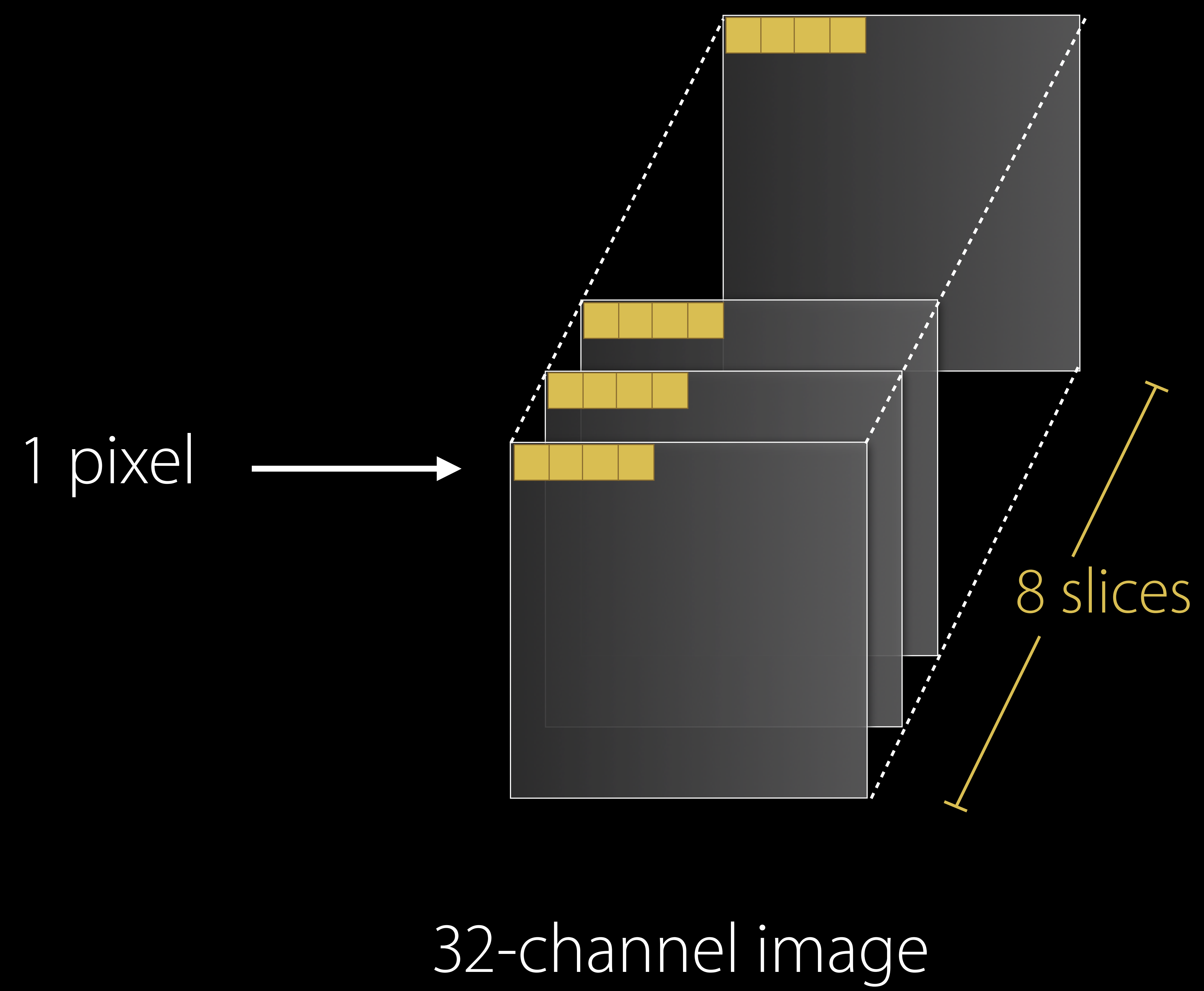
Data layout



32-channel image

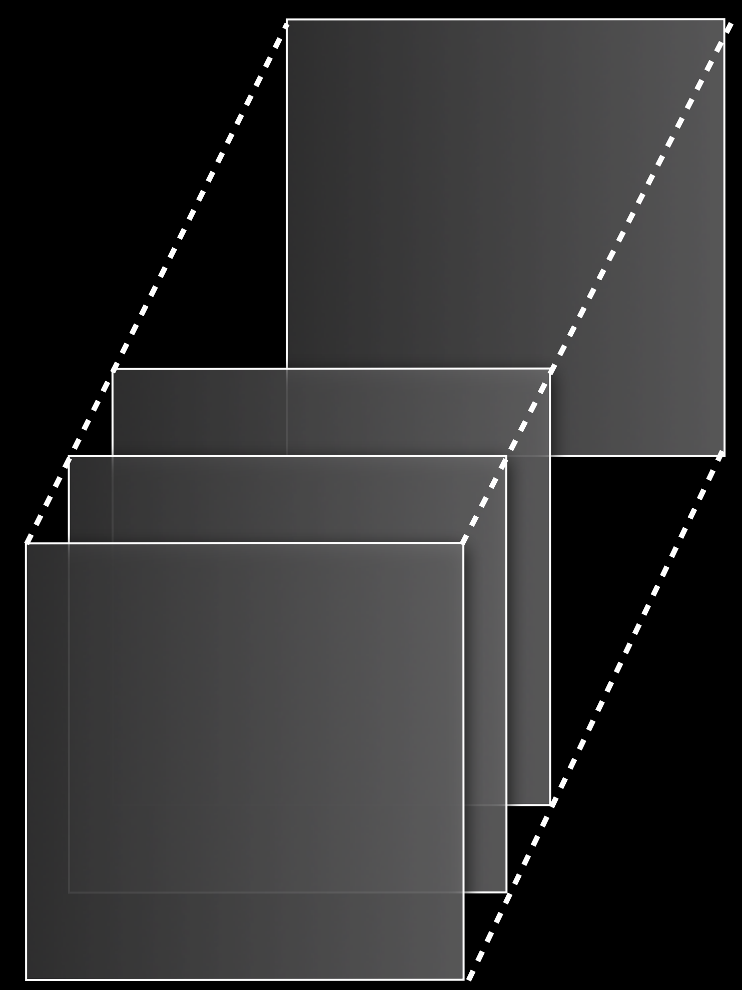
MPSImage

Data layout



MPSImage

Code

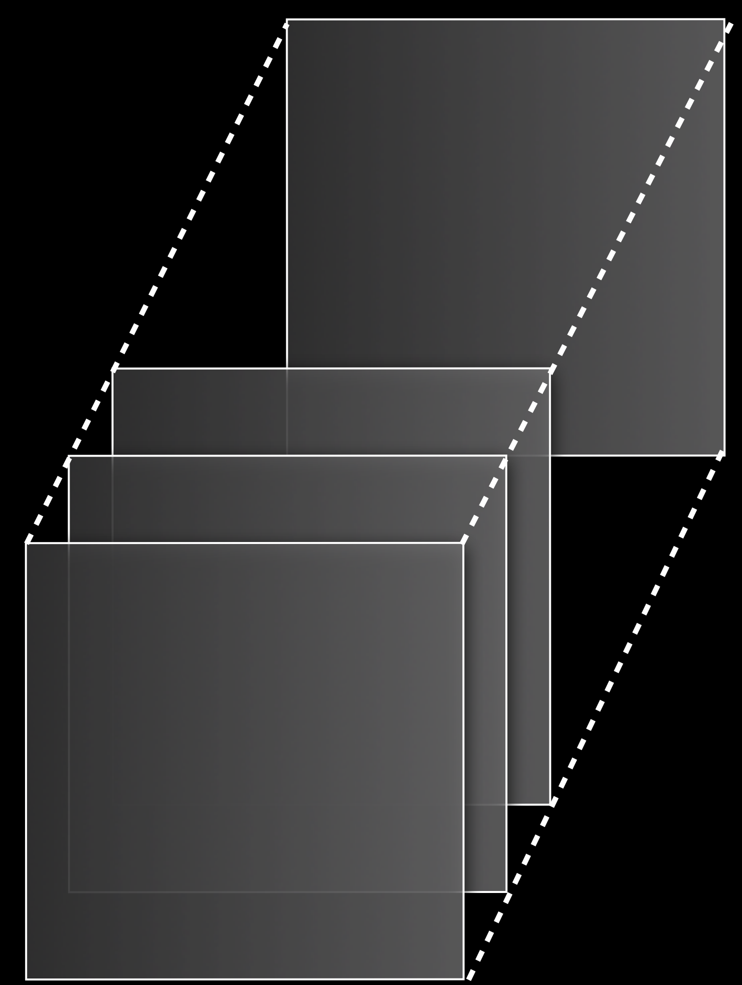


32-channel image

```
let imgDesc = MPSImageDescriptor(channelFormat: MPSImageFeatureChannelFormat.float16,
   width: width, height: height, featureChannels: 32)
var img = MPSImage(device: device, imageDescriptor: imgDesc)
```

MPSImage

Code

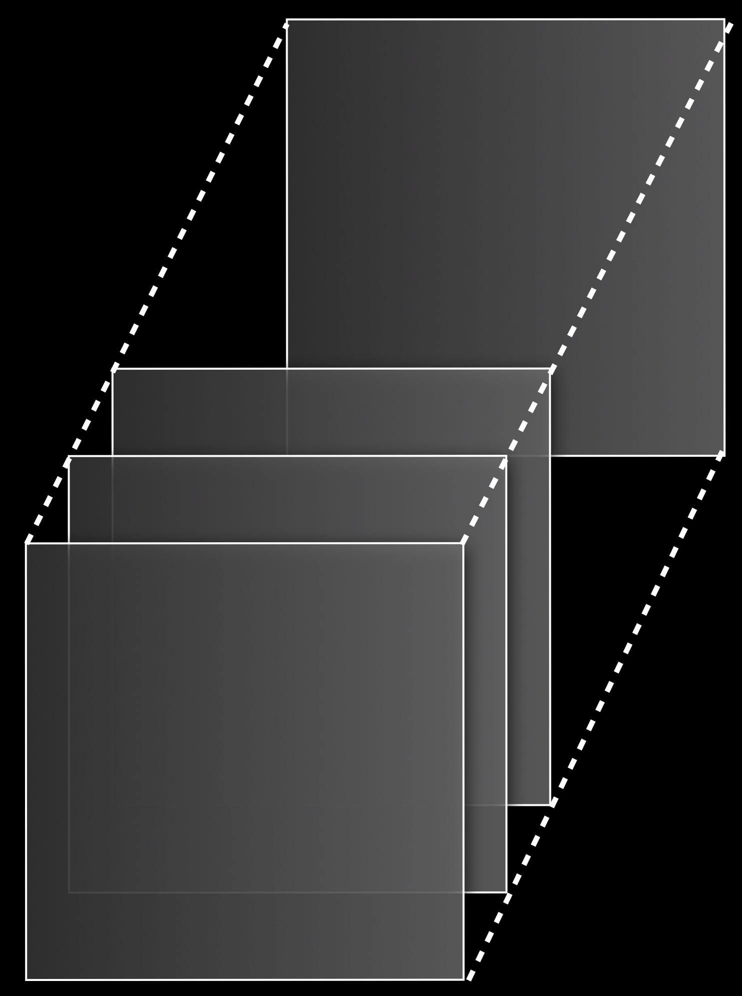


32-channel image

```
let imgDesc = MPSImageDescriptor(channelFormat: MPSImageFeatureChannelFormat.float16,
   width: width, height: height, featureChannels: 32)
var img = MPSImage(device: device, imageDescriptor: imgDesc)
```

MPSImage

Code

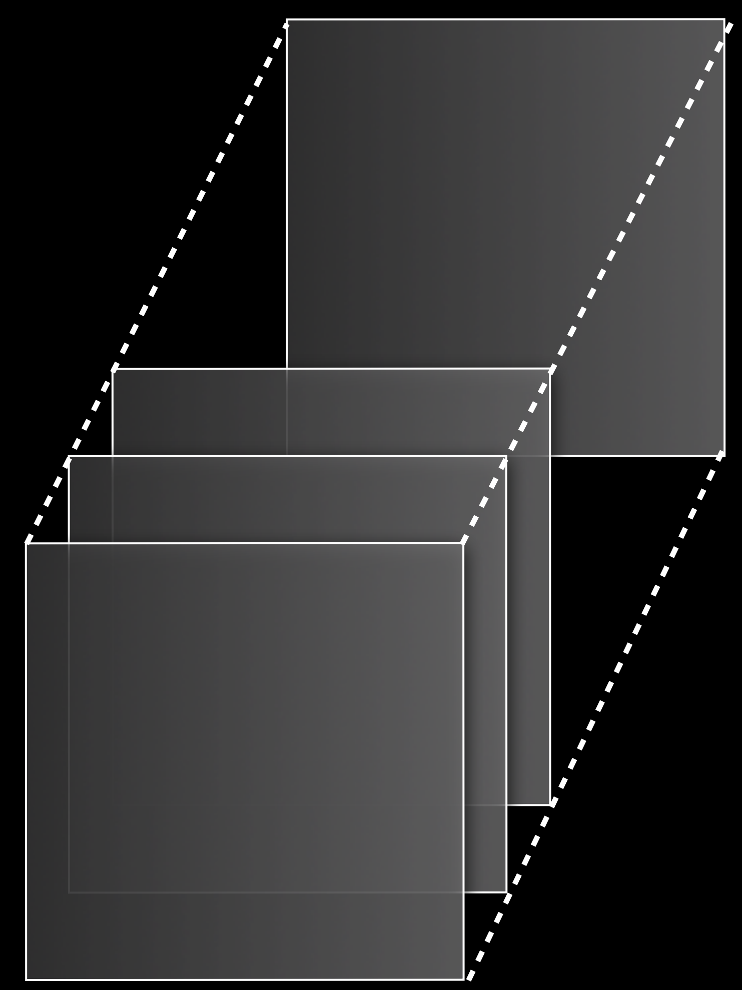


32-channel image

```
let imgDesc = MPSImageDescriptor(channelFormat: MPSImageFeatureChannelFormat.float16,
    width: width, height: height, featureChannels: 32)
var img = MPSImage(device: device, imageDescriptor: imgDesc)
```

MPSImage

Code

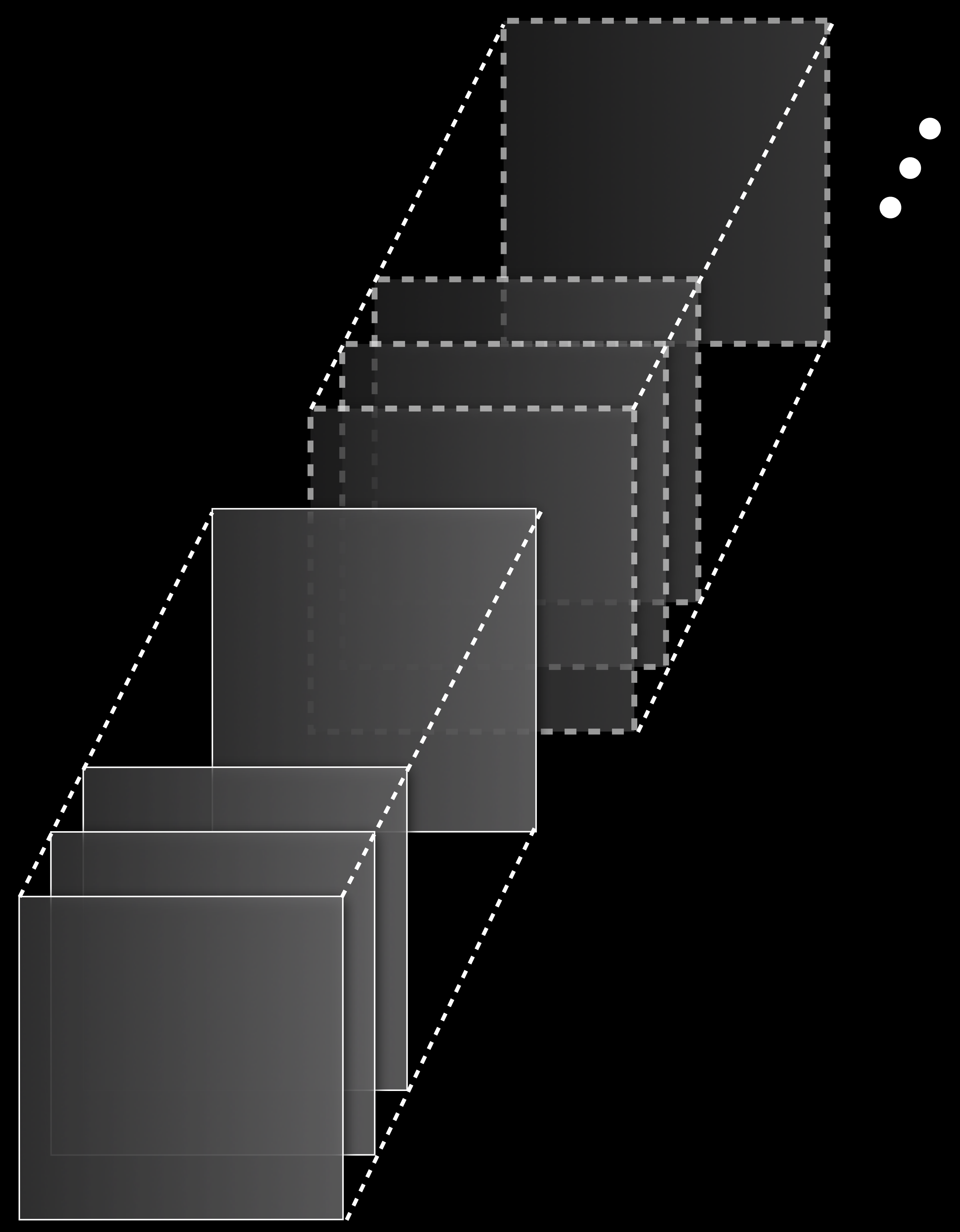


32-channel image

```
let imgDesc = MPSImageDescriptor(channelFormat: MPSImageFeatureChannelFormat.float16,
   width: width, height: height, featureChannels: 32)
var img = MPSImage(device: device, imageDescriptor: imgDesc)
```

MPSImage

Batch processing

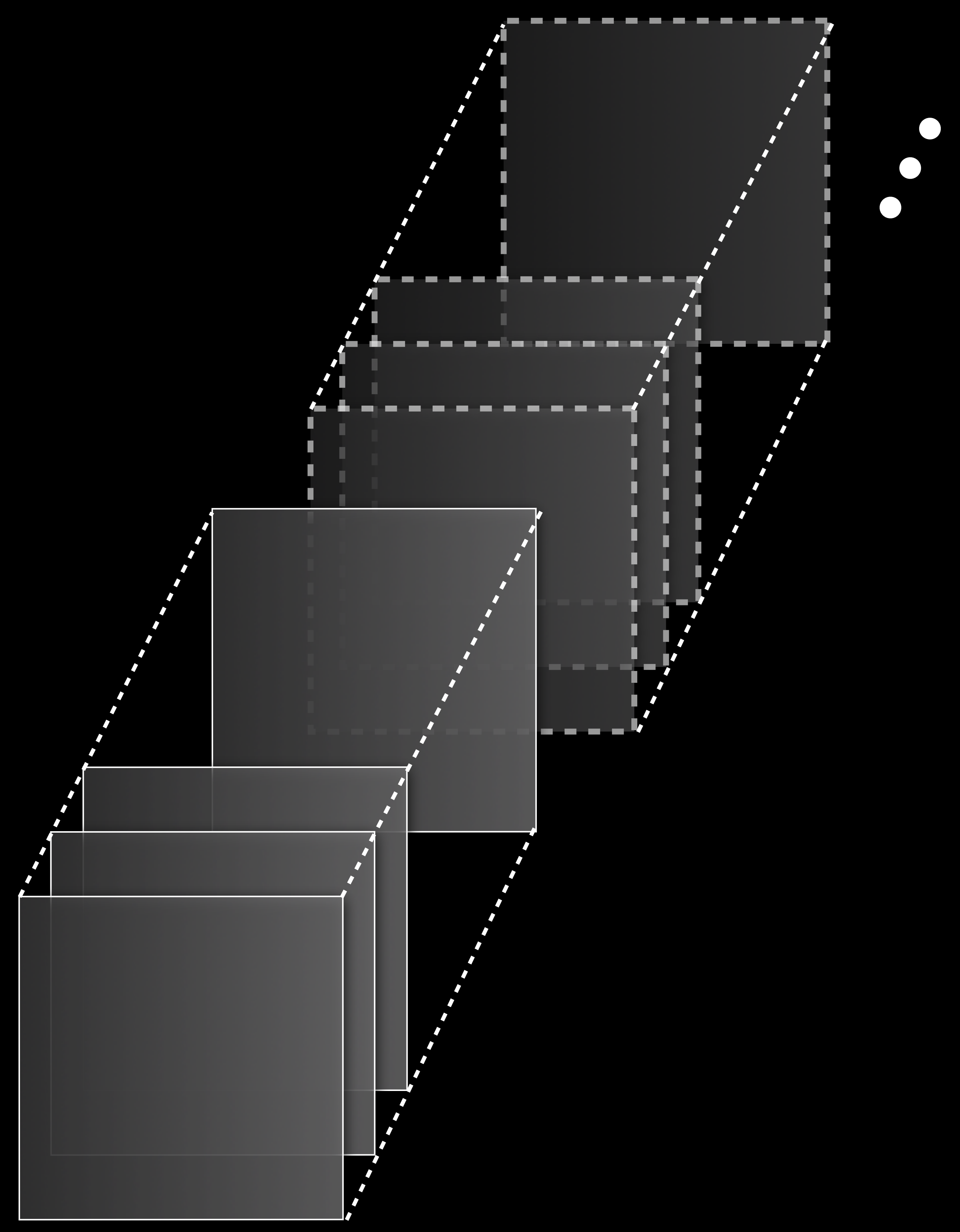


100 32-channel image

```
let imgDesc = MPSImageDescriptor(channelFormat: MPSImageFeatureChannelFormat.float16,
   width: width, height: height, featureChannels: 32 numberOfImages: 100)
var img = MPSImage(device: device, imageDescriptor: imgDesc)
```

MPSImage

Batch processing



100 32-channel image

```
let imgDesc = MPSImageDescriptor(channelFormat: MPSImageFeatureChannelFormat.float16,
   width: width, height: height, featureChannels: 32 numberOfImages: 100)
var img = MPSImage(device: device, imageDescriptor: imgDesc)
```

Detecting a Smile

Inference

3
channels

40 x 40

input

Detecting a Smile

Inference

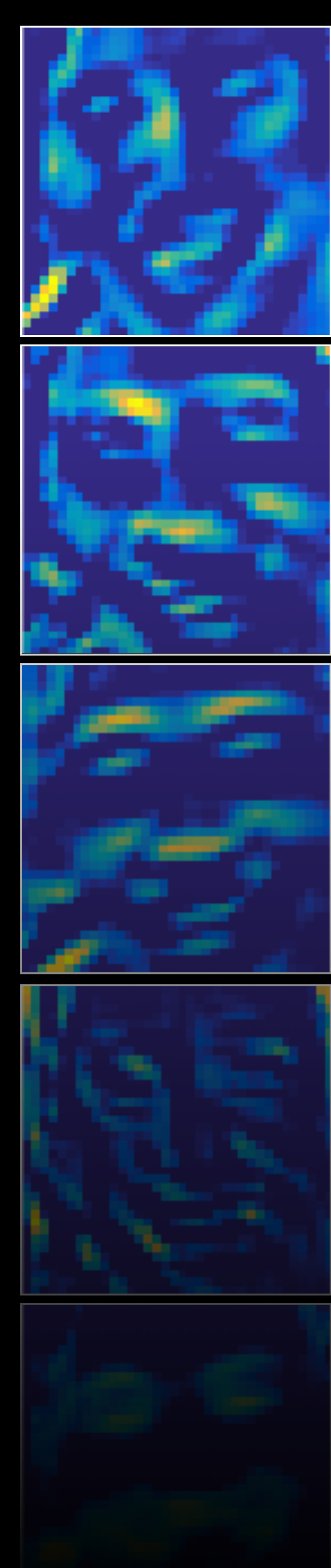
conv
layer 1

3
channels

40 x 40

input

16
channels



Detecting a Smile

Inference

conv
layer 1

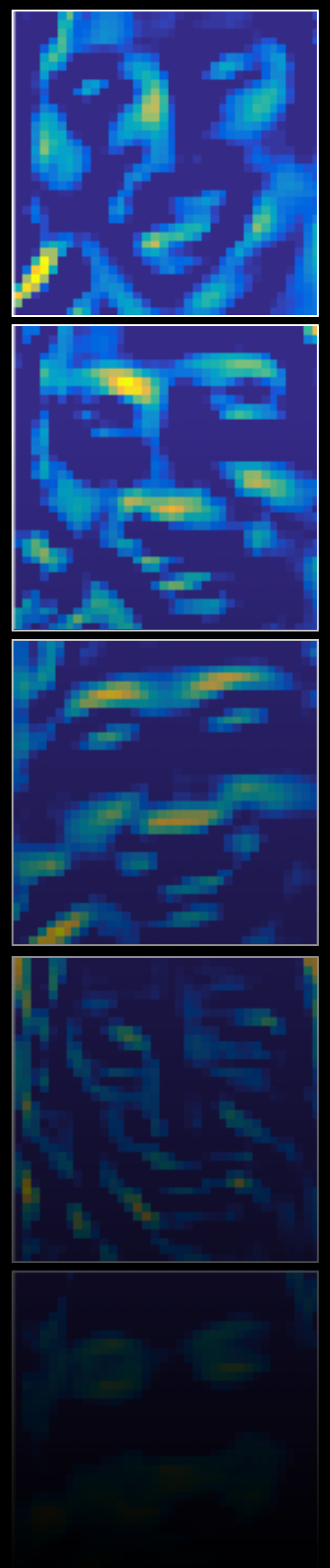
pool
layer 1

3
channels

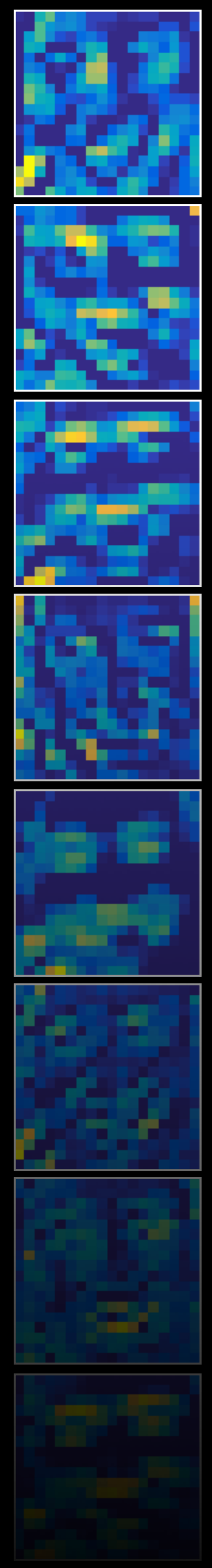
40 x 40

input

16
channels



16
channels



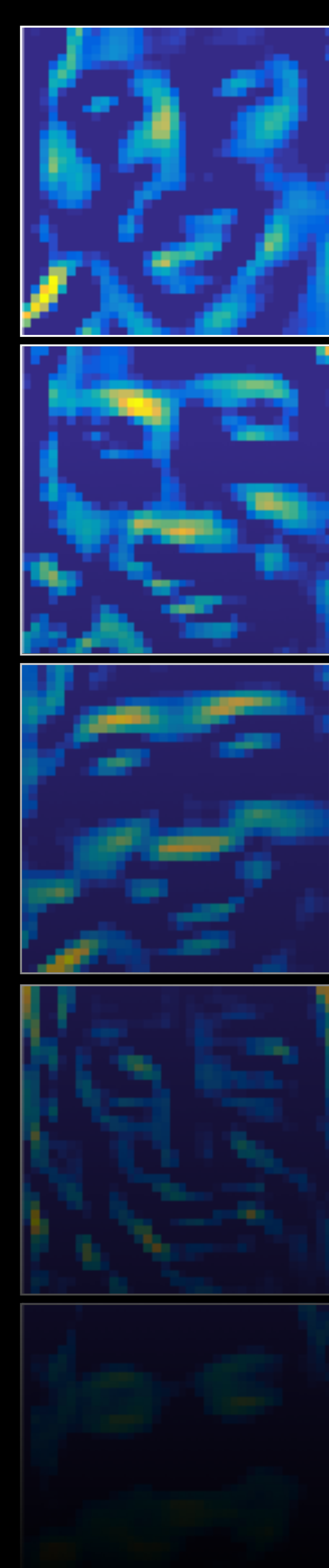
Detecting a Smile

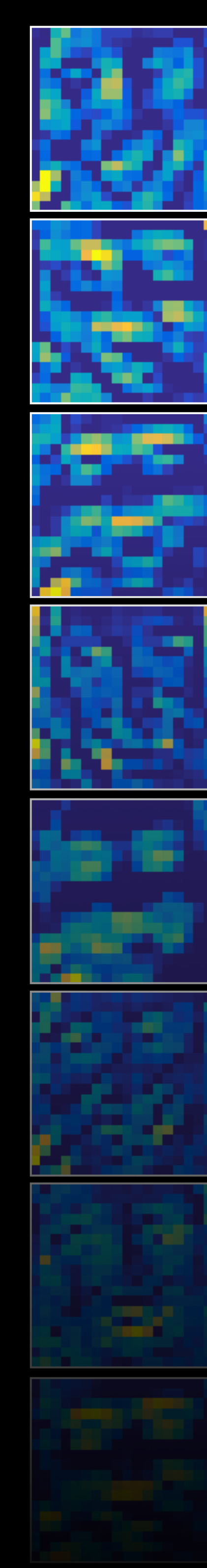
Inference

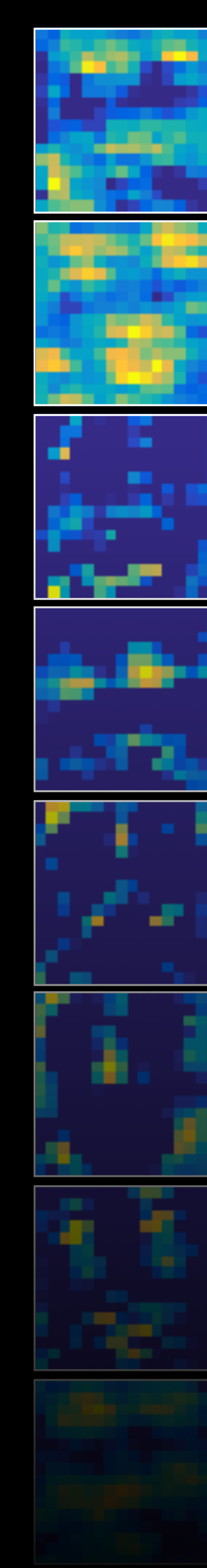
conv layer 1	pool layer 1	conv layer 2

3 channels	16 channels	16 channels	48 channels
40 x 40			

input







Detecting a Smile

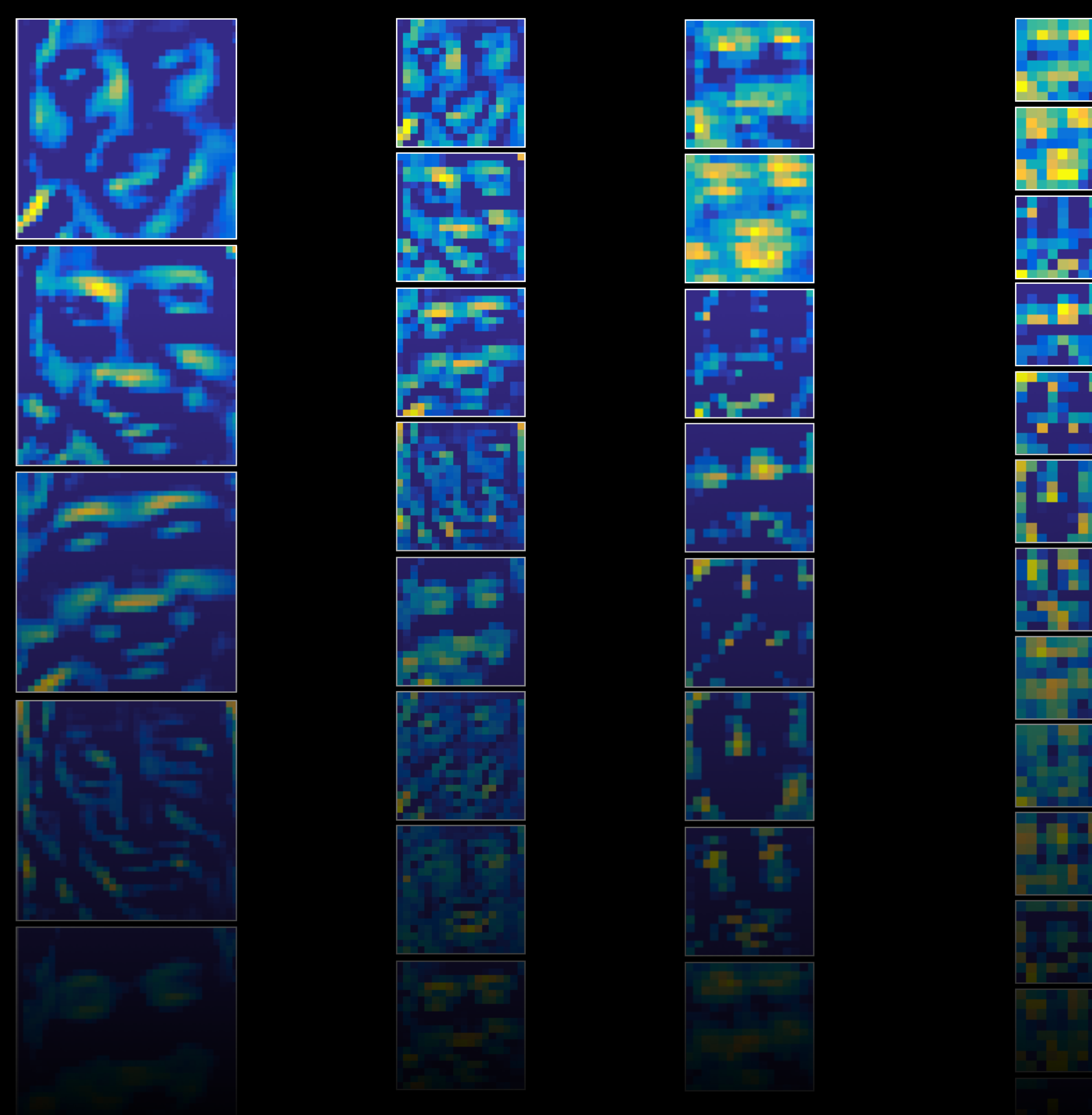
Inference

conv layer 1 | pool layer 1 | conv layer 2 | pool layer 2

3 channels | 16 channels | 16 channels | 48 channels | 48 channels

40 x 40

input



Detecting a Smile

Inference

```swift
// Code Sample: Detecting a Smile Using CNN
// Create layers
var conv1, conv2, conv3, conv4: MPSCNNConvolution
let conv1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
conv1 = MPSCNNConvolution(device: device, convolutionDescriptor: conv1Desc,
   kernelWeights: conv1Filters, biasTerms: conv1Bias, flags: MPSCNNConvolutionFlags.none)
...

var pool: MPSCNNPoolingMax
pool = MPSCNNPoolingMax(device: device, kernelWidth: 2, kernelHeight: 2,
   strideInPixelsX: 2, strideInPixelsY: 2)

var fc1, fc2: MPSCNNFullyConnected
let fc1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 2, kernelHeight: 2,
inputFeatureChannels: 64, outputFeatureChannels: 256, neuronFilter: nil)
conv1 = MPSCNNFullyConnected(device: device, convolutionDescriptor: fc1Desc,
   kernelWeights: fc1Feature, biasTerms: fc1Bias, flags: MPSCNNConvolutionFlags.none)
...
```

```swift
// Code Sample: Detecting a Smile Using CNN
// Create layers
var conv1, conv2, conv3, conv4: MPSCNNConvolution
let conv1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
conv1 = MPSCNNConvolution(device: device, convolutionDescriptor: conv1Desc,
   kernelWeights: conv1Filters, biasTerms: conv1Bias, flags: MPSCNNConvolutionFlags.none)
...

var pool: MPSCNNPoolingMax
pool = MPSCNNPoolingMax(device: device, kernelWidth: 2, kernelHeight: 2,
   strideInPixelsX: 2, strideInPixelsY: 2)

var fc1, fc2: MPSCNNFullyConnected
let fc1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 2, kernelHeight: 2,
inputFeatureChannels: 64, outputFeatureChannels: 256, neuronFilter: nil)
conv1 = MPSCNNFullyConnected(device: device, convolutionDescriptor: fc1Desc,
   kernelWeights: fc1Feature, biasTerms: fc1Bias, flags: MPSCNNConvolutionFlags.none)
...
```

```swift
// Code Sample: Detecting a Smile Using CNN
// Create layers
var conv1, conv2, conv3, conv4: MPSCNNConvolution
let conv1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
conv1 = MPSCNNConvolution(device: device, convolutionDescriptor: conv1Desc,
   kernelWeights: conv1Filters, biasTerms: conv1Bias, flags: MPSCNNConvolutionFlags.none)
...

var pool: MPSCNNPoolingMax
pool = MPSCNNPoolingMax(device: device, kernelWidth: 2, kernelHeight: 2,
   strideInPixelsX: 2, strideInPixelsY: 2)

var fc1, fc2: MPSCNNFullyConnected
let fc1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 2, kernelHeight: 2,
inputFeatureChannels: 64, outputFeatureChannels: 256, neuronFilter: nil)
conv1 = MPSCNNFullyConnected(device: device, convolutionDescriptor: fc1Desc,
   kernelWeights: fc1Feature, biasTerms: fc1Bias, flags: MPSCNNConvolutionFlags.none)
...
```

```swift
// Code Sample: Detecting a Smile Using CNN
// Create layers
var conv1, conv2, conv3, conv4: MPSCNNConvolution
let conv1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 5, kernelHeight: 5,
inputFeatureChannels: 3, outputFeatureChannels: 16, neuronFilter: nil)
conv1 = MPSCNNConvolution(device: device, convolutionDescriptor: conv1Desc,
   kernelWeights: conv1Filters, biasTerms: conv1Bias, flags: MPSCNNConvolutionFlags.none)
...

var pool: MPSCNNPoolingMax
pool = MPSCNNPoolingMax(device: device, kernelWidth: 2, kernelHeight: 2,
   strideInPixelsX: 2, strideInPixelsY: 2)

var fc1, fc2: MPSCNNFullyConnected
let fc1Desc = MPSCNNConvolutionDescriptor(kernelWidth: 2, kernelHeight: 2,
inputFeatureChannels: 64, outputFeatureChannels: 256, neuronFilter: nil)
conv1 = MPSCNNFullyConnected(device: device, convolutionDescriptor: fc1Desc,
   kernelWeights: fc1Feature, biasTerms: fc1Bias, flags: MPSCNNConvolutionFlags.none)
...
```

```swift
// Create MPSImages to Hold Input and Output
var input, output : MPSImage
input = MPSImage(texture: inputTexture, featureChannels: 3) // 40 x 40
output = MPSImage(texture: outputTexture, featureChannels: 1) // 1 x 1
```

```swift
// Create MPSImages to Hold Input and Output
var input, output : MPSImage
input = MPSImage(texture: inputTexture, featureChannels: 3) // 40 x 40
output = MPSImage(texture: outputTexture, featureChannels: 1) // 1 x 1
```

```swift
// Create MPSImages to Hold Input and Output
var input, output : MPSImage
input = MPSImage(texture: inputTexture, featureChannels: 3) // 40 x 40
output = MPSImage(texture: outputTexture, featureChannels: 1) // 1 x 1
```

```swift
// Encode Layers

conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```
// Encode Layers

conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```
// Encode Layers

conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)
fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage



conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage

conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)

pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: ...)

conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)

pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: ...)

conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)

pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: ...)

conv2.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

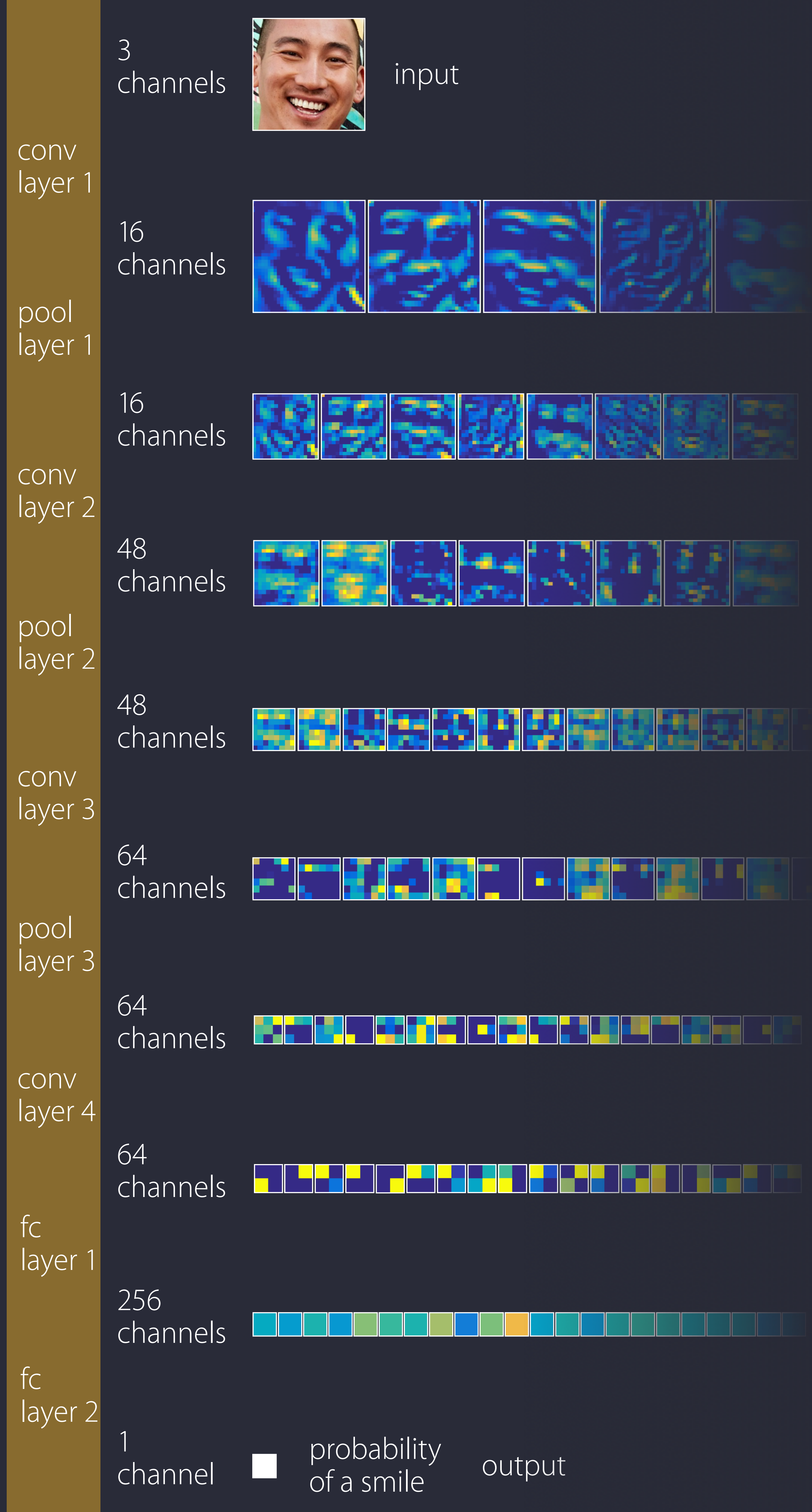
```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)
img2 = ...
pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: img2)

conv2.encode(to: commandBuffer, sourceImage: img2, destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)
img2 = ...
pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: img2)

conv2.encode(to: commandBuffer, sourceImage: img2, destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv3.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

pool.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

conv4.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc1.encode(to: commandBuffer, sourceImage: ..., destinationImage: ...)

fc2.encode(to: commandBuffer, sourceImage: ..., destinationImage: output)
```

```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)
img2 = ...
pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: img2)

conv2.encode(to: commandBuffer, sourceImage: img2, destinationImage: img3)

pool.encode(to: commandBuffer, sourceImage: img3, destinationImage: img4)

conv3.encode(to: commandBuffer, sourceImage: img4, destinationImage: img5)

pool.encode(to: commandBuffer, sourceImage: img5, destinationImage: img6)

conv4.encode(to: commandBuffer, sourceImage: img6, destinationImage: img7)

fc1.encode(to: commandBuffer, sourceImage: img7, destinationImage: img8)

fc2.encode(to: commandBuffer, sourceImage: img8, destinationImage: output)
```

```swift
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)
img2 = ...
pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: img2)

conv2.encode(to: commandBuffer, sourceImage: img2, destinationImage: img3)

pool.encode(to: commandBuffer, sourceImage: img3, destinationImage: img4)

conv3.encode(to: commandBuffer, sourceImage: img4, destinationImage: img5)

pool.encode(to: commandBuffer, sourceImage: img5, destinationImage: img6)

conv4.encode(to: commandBuffer, sourceImage: img6, destinationImage: img7)

fc1.encode(to: commandBuffer, sourceImage: img7, destinationImage: img8)

fc2.encode(to: commandBuffer, sourceImage: img8, destinationImage: output)
```

```
// Encode Layers
var img1, img2, img3, img4, img5, img6, img7, img8 : MPSTemporaryImage
let img1Desc = MPSImageDescriptor(channelFormat: float16,
   width: 40, height: 40, featureChannels: 16)
img1 = MPSTemporaryImage(device: device, imageDescriptor: img1Desc)
conv1.encode(to: commandBuffer, sourceImage: input, destinationImage: img1)
img2 = ...
pool.encode(to: commandBuffer, sourceImage: img1, destinationImage: img2)

conv2.encode(to: commandBuffer, sourceImage: img2, destinationImage: img3)

pool.encode(to: commandBuffer, sourceImage: img3, destinationImage: img4)

conv3.encode(to: commandBuffer, sourceImage: img4, destinationImage: img5)

pool.encode(to: commandBuffer, sourceImage: img5, destinationImage: img6)

conv4.encode(to: commandBuffer, sourceImage: img6, destinationImage: img7)

fc1.encode(to: commandBuffer, sourceImage: img7, destinationImage: img8)

fc2.encode(to: commandBuffer, sourceImage: img8, destinationImage: output)
```



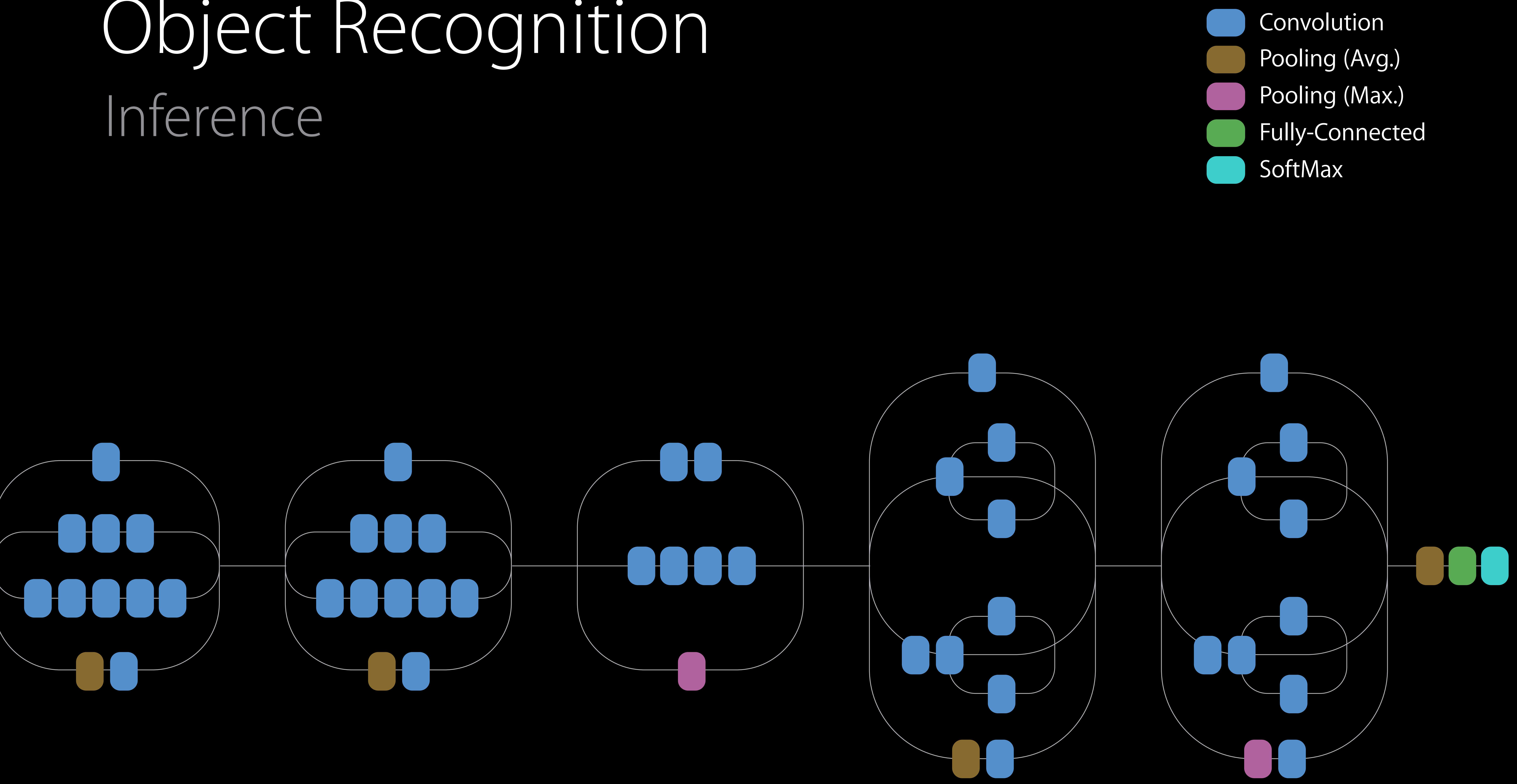
Object Recognition

Inference

- Convolution
- Pooling (Avg.)
- Pooling (Max.)
- Fully-Connected
- SoftMax

Object Recognition

Inference



Demo

Object recognition

Memory Savings with MPSTemporaryImages

Object recognition

74 MPSImages (83.8MB) needed for intermediate images

Replaced MPSImages with MPSTemporaryImages

- Reduced CPU cost: time and energy
- Automatic reduction of 74 images to five underlying allocations (20.5MB)

76%

Memory savings!

Summary

Metal Performance Shaders framework provides complete support for building Convolutional Neural Networks for inference on the GPU

What's New in Metal

Summary

What's New in Metal

Summary

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

What's New in Metal

Summary

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

Function Specialization and
Function Resource Read-Writes

Wide Color and Texture Assets

Additions to Metal Performance Shaders

More Information

https://developer.apple.com/wwdc16/605

Related Sessions

Adopting Metal, Part 1	Nob Hill	Tuesday 1:40PM
Adopting Metal, Part 2	Nob Hill	Tuesday 3:00PM
What's New in Metal, Part 1	Pacific Heights	Wednesday 11:00AM
Advanced Metal Shader Optimization	Nob Hill	Wednesday 3:00PM
Working with Wide Color	Mission	Thursday 1:40PM
Neural Networks and Accelerate	Nob Hill	Thursday 4:00PM

Labs

Color Lab	Frameworks Lab A	Wednesday 1:00PM
Metal Lab	Graphics, Games, and Media Lab A	Thursday 12:00PM
macOS Graphics and Games Lab	Graphics, Games, and Media Lab B	Thursday 12:00PM
Color Lab	Graphics, Games, and Media Lab C	Friday 4:00PM

WWDC16