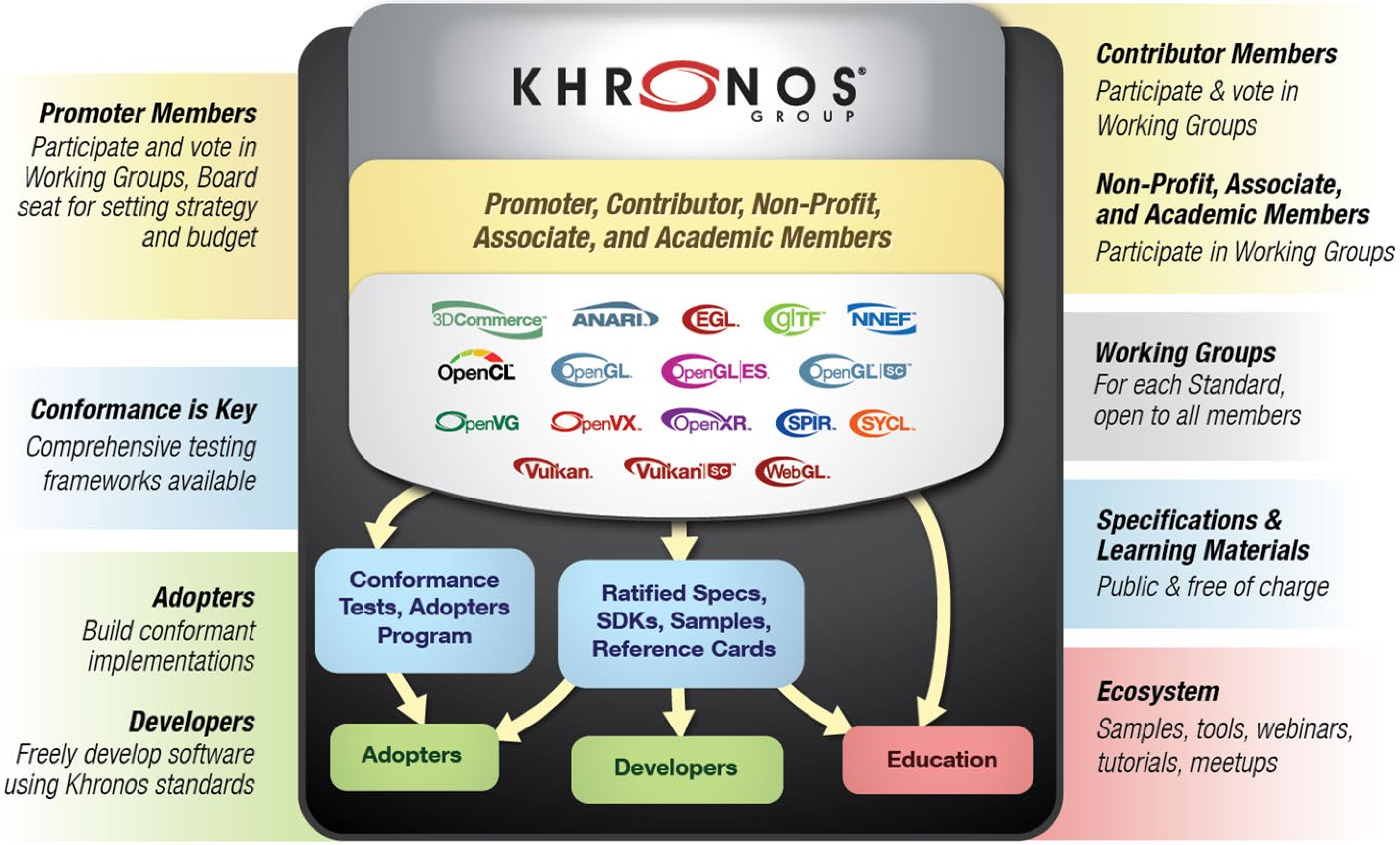
KHRONOS® GROUP

A TOUR OF THE ANARI™ API

Jefferson Amstutz - March 2022

WHAT IS KHRONOS?



Promoter Members
Participate and vote in Working Groups, Board seat for setting strategy and budget

Conformance is Key
Comprehensive testing frameworks available

Adopters
Build conformant implementations

Developers
Freely develop software using Khronos standards

Promoter, Contributor, Non-Profit, Associate, and Academic Members

Conformance Tests, Adopters Program

Ratified Specs, SDKs, Samples, Reference Cards

Adopters

Developers

Education

Contributor Members
Participate & vote in Working Groups

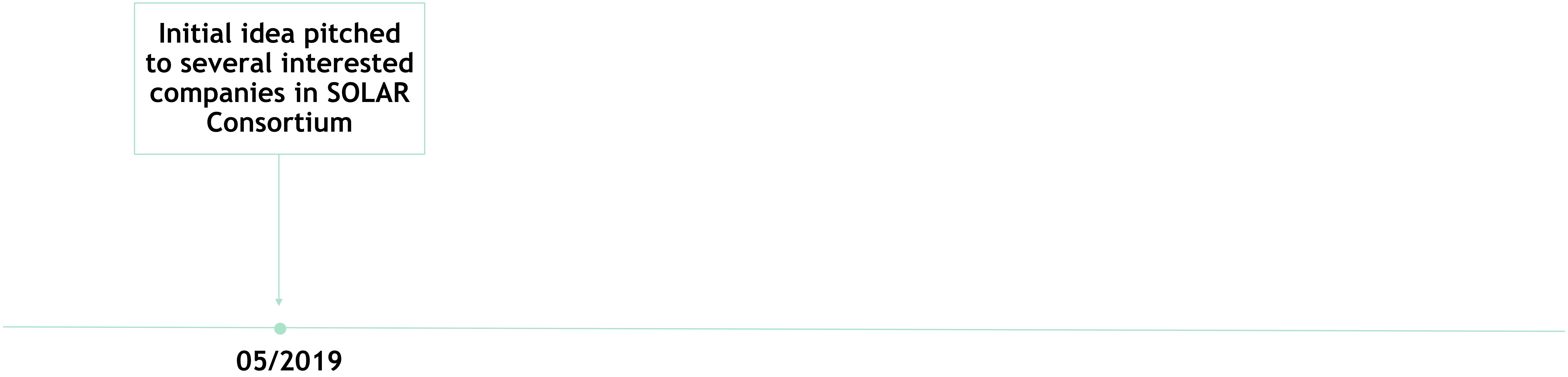
Non-Profit, Associate, and Academic Members
Participate in Working Groups

Working Groups
For each Standard, open to all members

Specifications & Learning Materials
Public & free of charge

Ecosystem
Samples, tools, webinars, tutorials, meetups

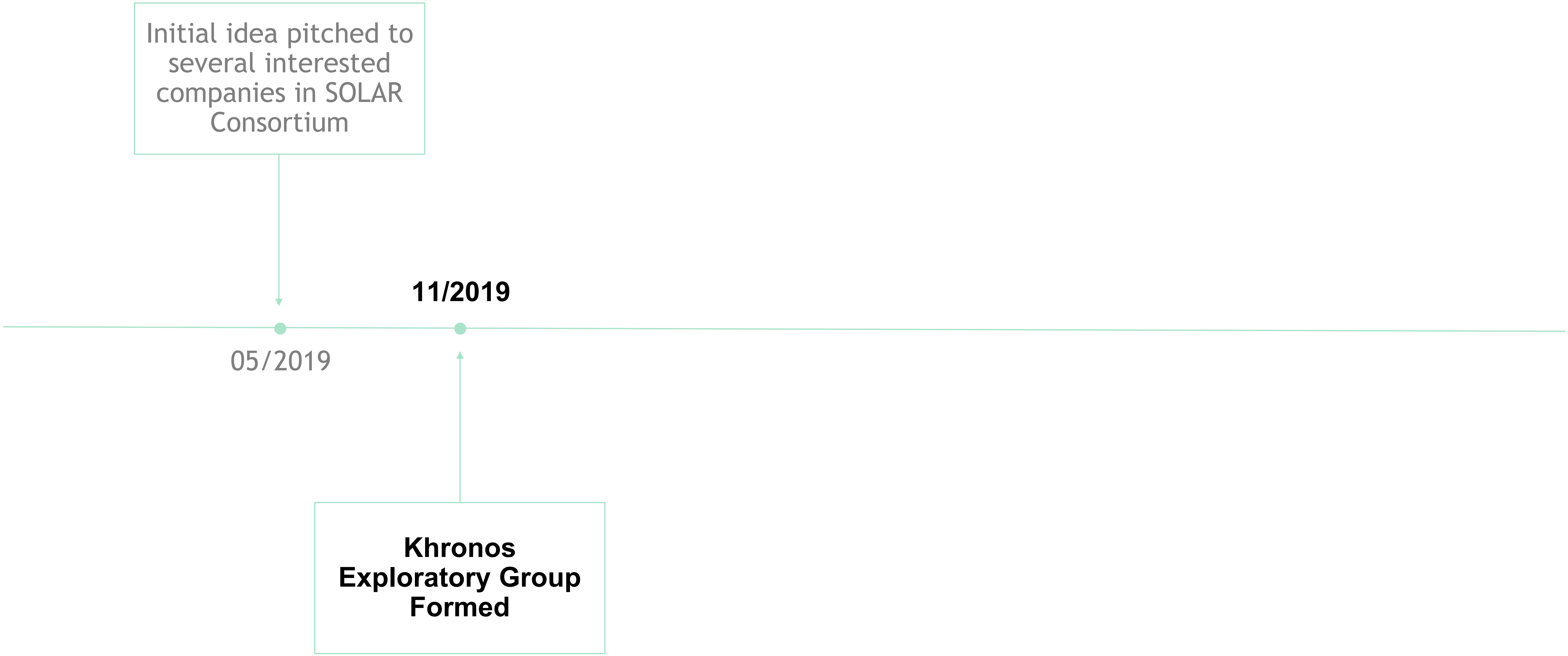
ANARI DEVELOPMENT TIMELINE



Initial idea pitched to several interested companies in SOLAR Consortium

05/2019

ANARI DEVELOPMENT TIMELINE



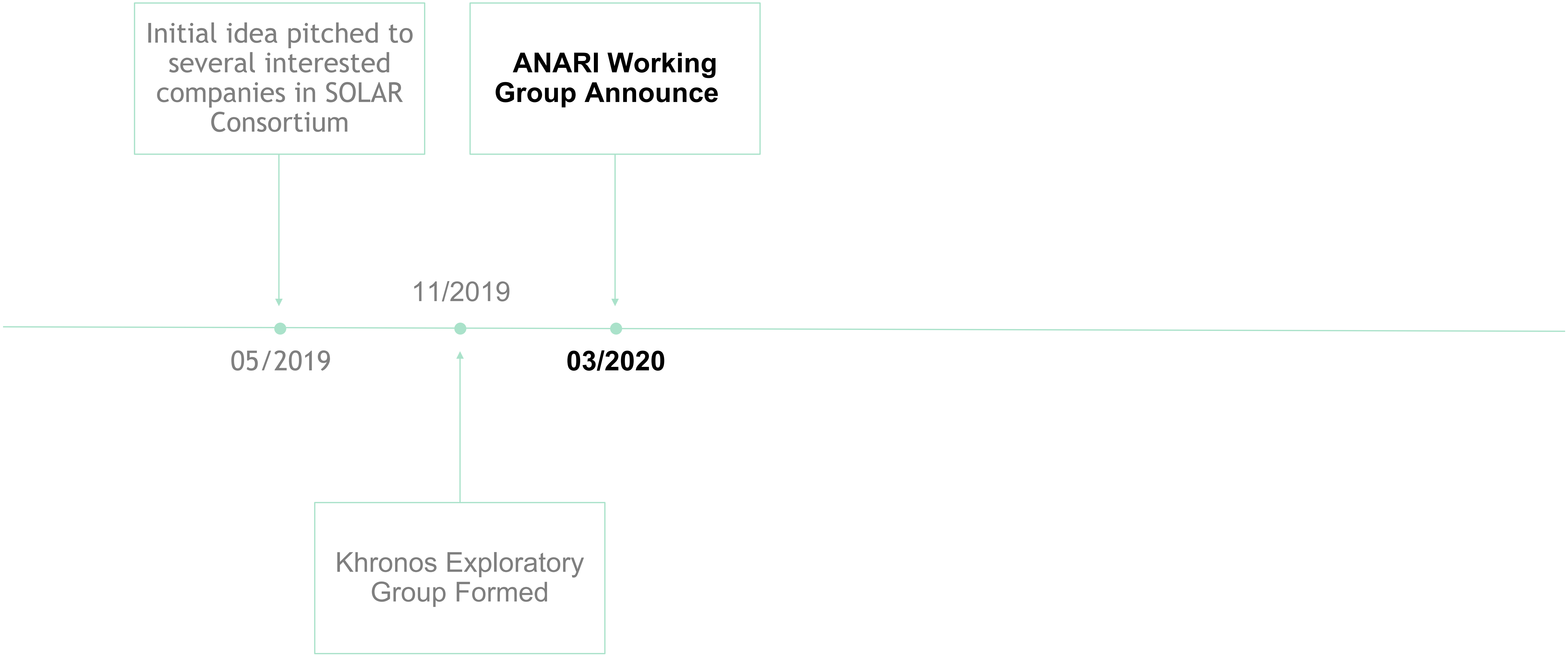
Initial idea pitched to several interested companies in SOLAR Consortium

05/2019

11/2019

Khronos Exploratory Group Formed

ANARI DEVELOPMENT TIMELINE



Initial idea pitched to several interested companies in SOLAR Consortium

05/2019

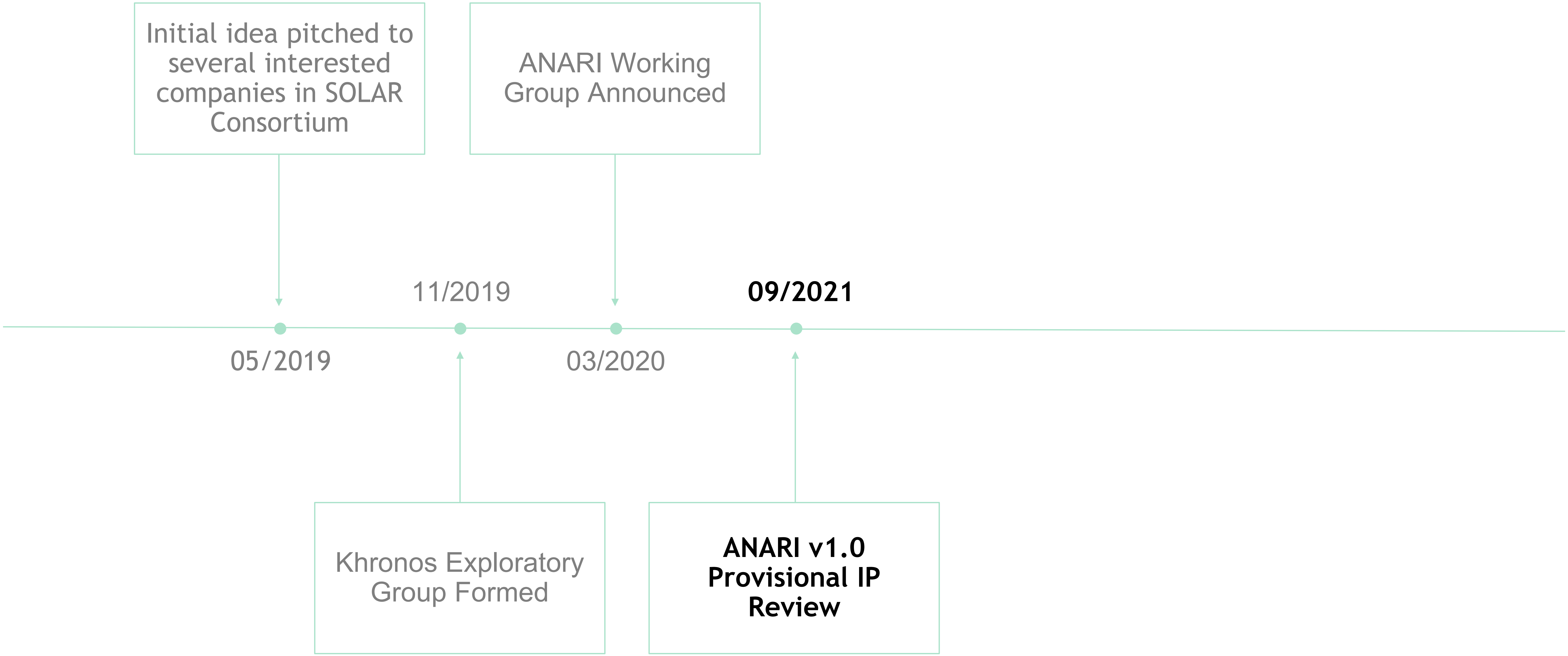
11/2019

Khronos Exploratory Group Formed

ANARI Working Group Announce

03/2020

ANARI DEVELOPMENT TIMELINE



05/2019

Initial idea pitched to several interested companies in SOLAR Consortium

11/2019

Khronos Exploratory Group Formed

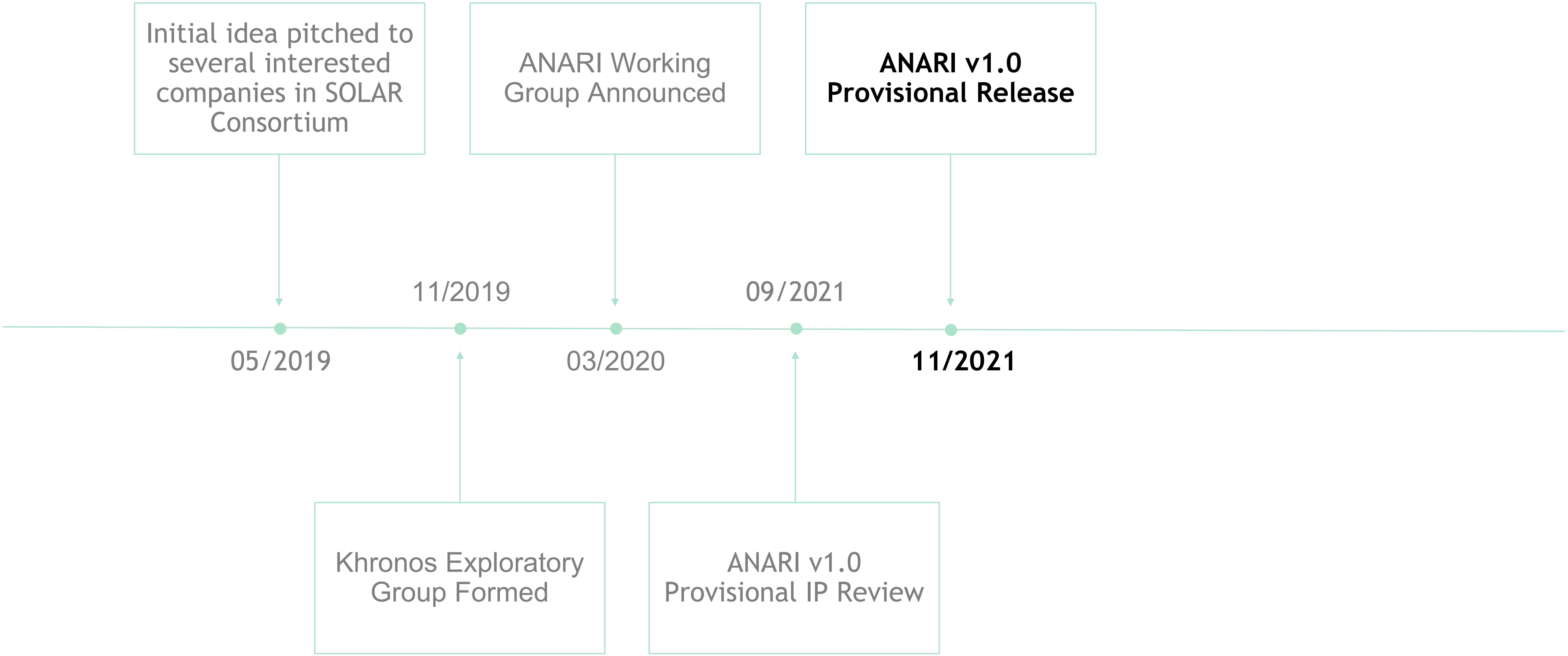
03/2020

ANARI Working Group Announced

09/2021

ANARI v1.0 Provisional IP Review

ANARI DEVELOPMENT TIMELINE



05/2019

Initial idea pitched to several interested companies in SOLAR Consortium

11/2019

Khronos Exploratory Group Formed

03/2020

ANARI Working Group Announced

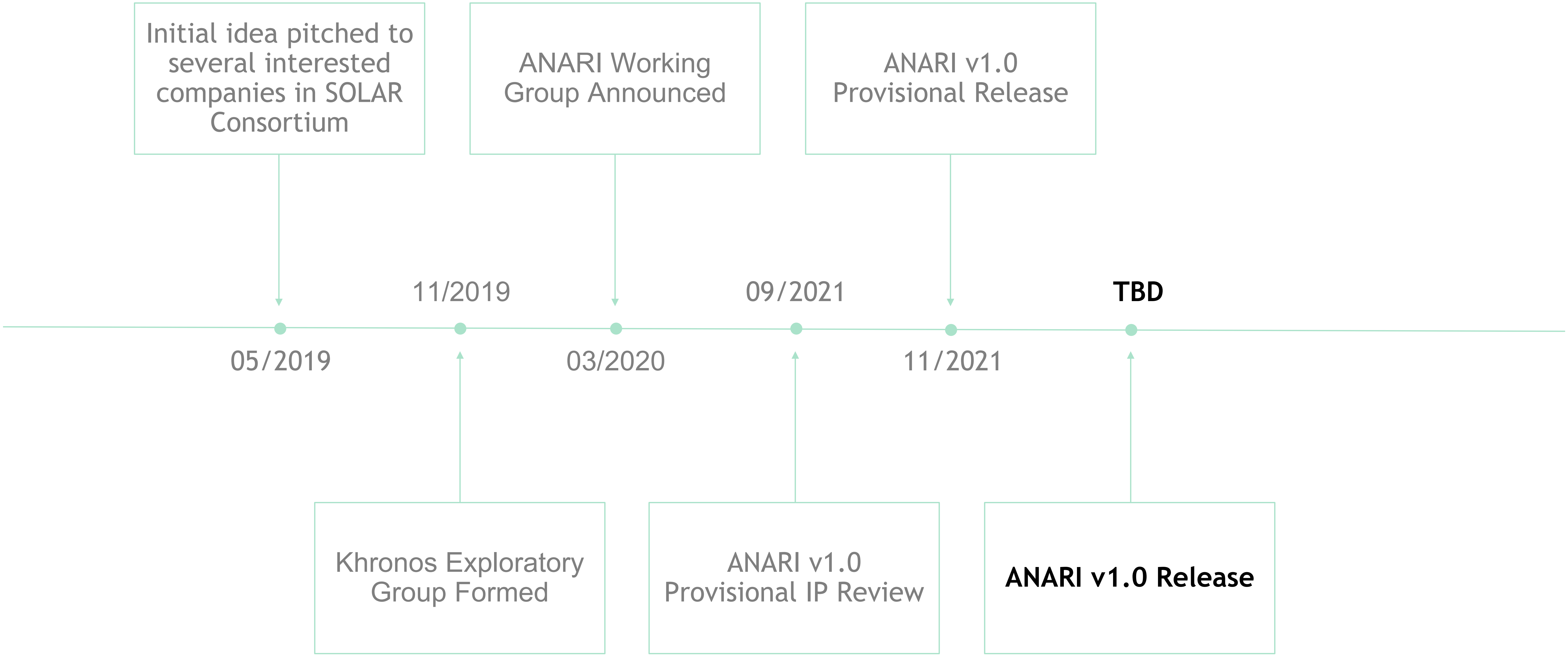
09/2021

ANARI v1.0 Provisional IP Review

11/2021

ANARI v1.0 Provisional Release

ANARI DEVELOPMENT TIMELINE



05/2019

Initial idea pitched to several interested companies in SOLAR Consortium

11/2019

Khronos Exploratory Group Formed

03/2020

ANARI Working Group Announced

09/2021

ANARI v1.0 Provisional IP Review

11/2021

ANARI v1.0 Provisional Release

TBD

ANARI v1.0 Release

WHAT PROBLEM DOES ANARI ADDRESS?

WHAT PROBLEM DOES ANARI ADDRESS?

3D APPLICATIONS

ParaView

VisIt

VMD
Visual Molecular Dynamics

...

WHAT PROBLEM DOES ANARI ADDRESS?

3D APPLICATIONS

ParaView

VisIt

VMD
Visual Molecular Dynamics

...

RENDERING ENGINES

Intel® OSPRay

AMD Radeon™ ProRender

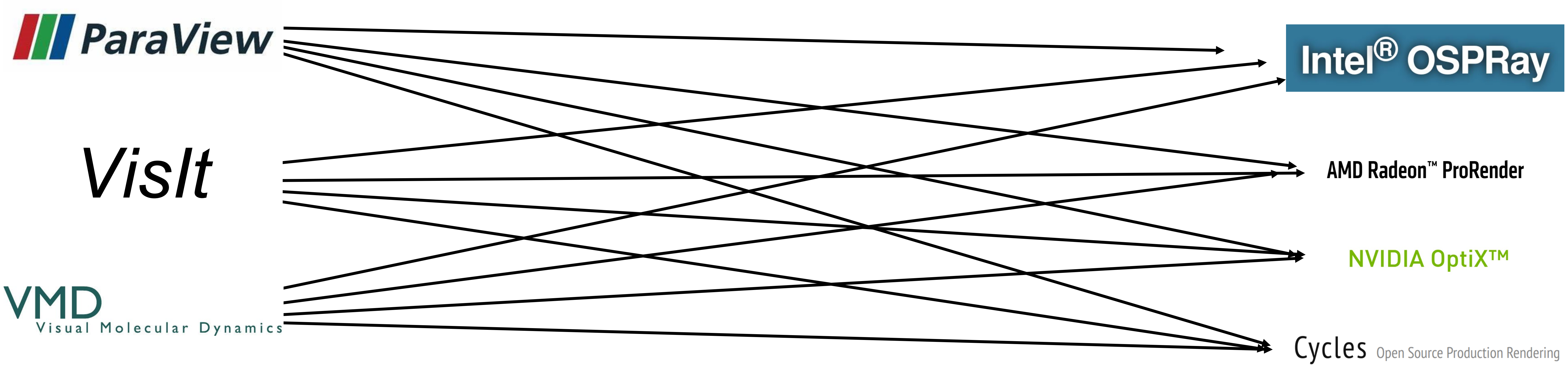
NVIDIA OptiX™

Cycles Open Source Production Rendering

WHAT PROBLEM DOES ANARI ADDRESS?

3D APPLICATIONS

RENDERING ENGINES



ParaView

VisIt

VMD Visual Molecular Dynamics

Intel® OSPRay

AMD Radeon™ ProRender

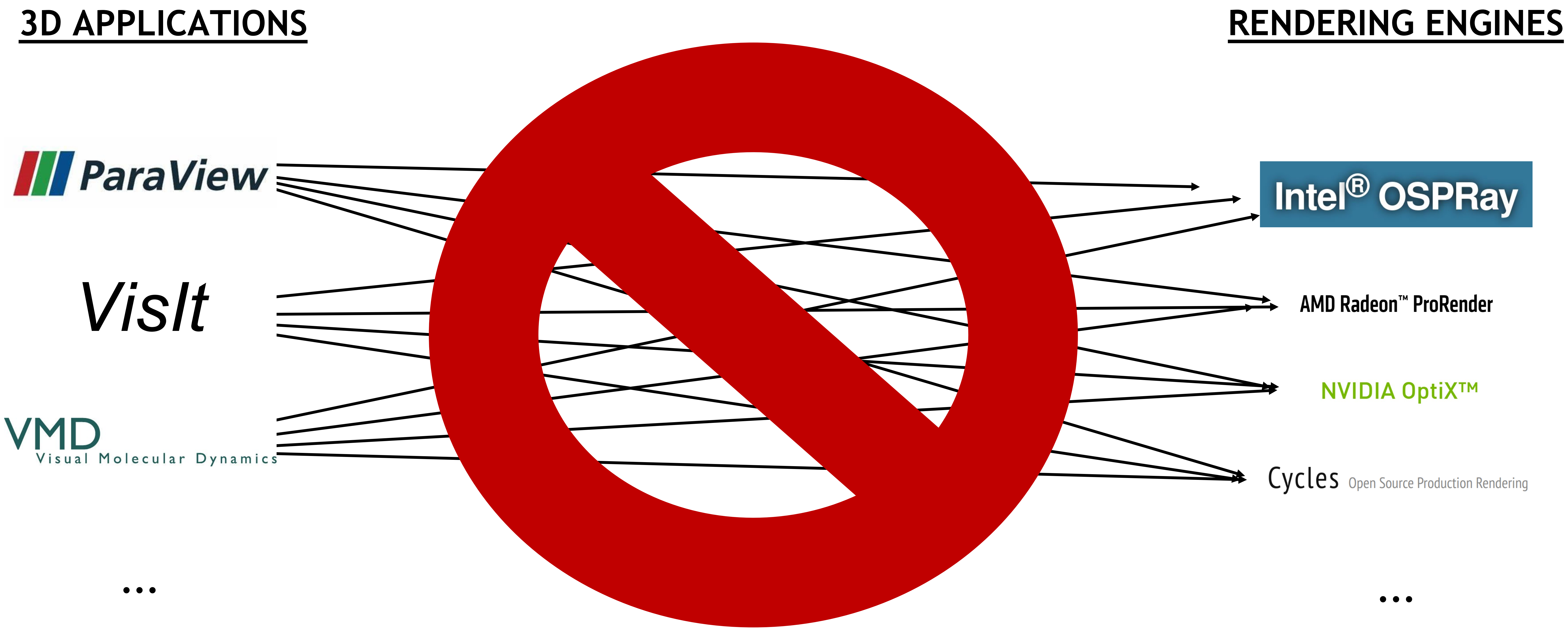
NVIDIA OptiX™

Cycles Open Source Production Rendering

...

...

WHAT PROBLEM DOES ANARI ADDRESS?



3D APPLICATIONS

ParaView

VisIt

VMD Visual Molecular Dynamics

...

RENDERING ENGINES

Intel® OSPRay

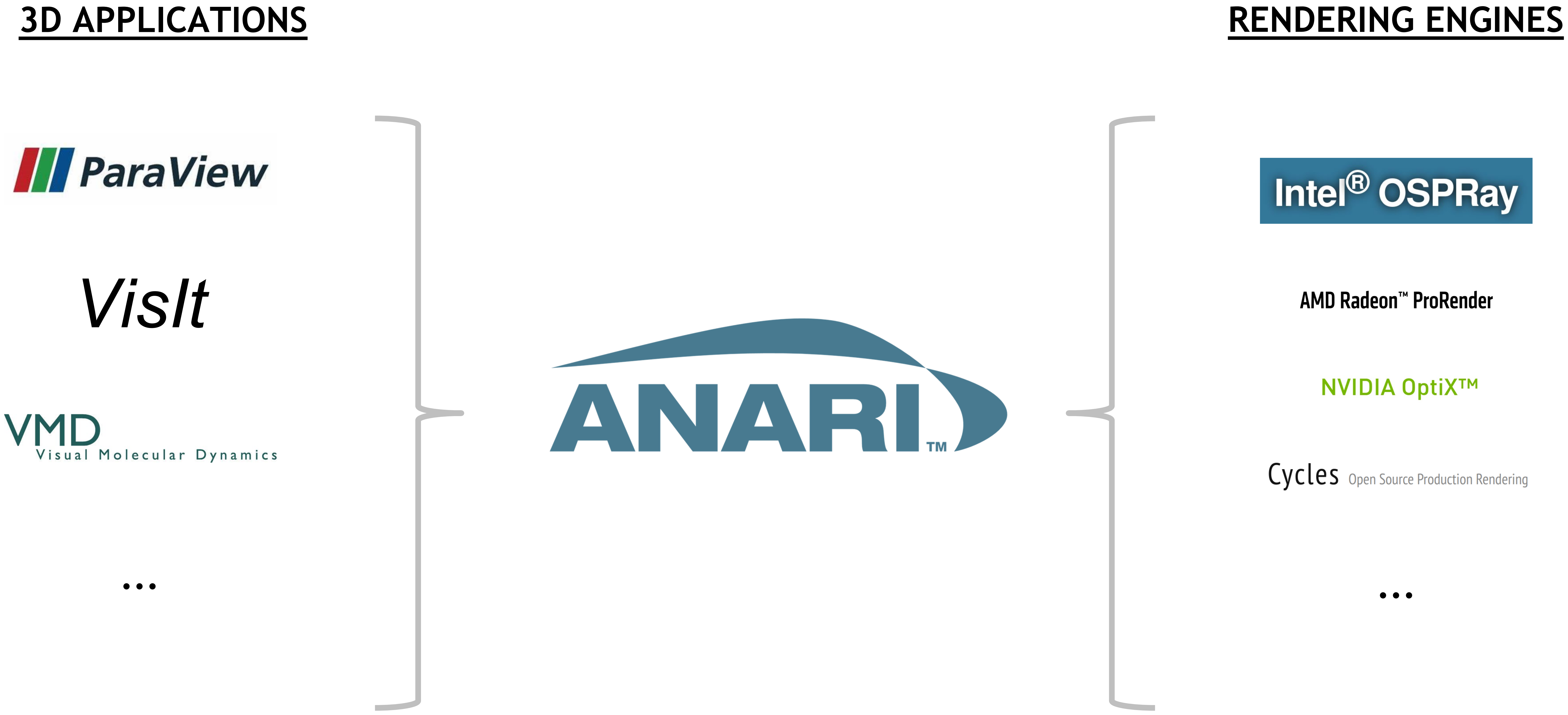
AMD Radeon™ ProRender

NVIDIA OptiX™

Cycles Open Source Production Rendering

...

WHAT PROBLEM DOES ANARI ADDRESS?



WHAT PROBLEM DOES ANARI ADDRESS?

This includes offline (< 5 FPS), interactive (5-30 FPS), and real-time (60+ FPS) rendering applications

WHAT PROBLEM DOES ANARI ADDRESS?

This includes offline (< 5 FPS), interactive (5-30 FPS), and real-time (60+ FPS) rendering applications

ANARI does its best to "get out of the way"

No required infrastructure for applications to use the API, and absolute minimal code required for implementations to hook into the ANARI API front-end library

WHAT PROBLEM DOES ANARI ADDRESS?

This includes offline (< 5 FPS), interactive (5-30 FPS), and real-time (60+ FPS) rendering applications

ANARI does its best to "get out of the way"

- No required infrastructure for applications to use the API, and absolute minimal code required for implementations to hook into the ANARI API front-end library

Implementations can optimize for different things without using different API calls

API BASICS

Software stack

3D Applications

Scene Graphs

API BASICS

Software stack

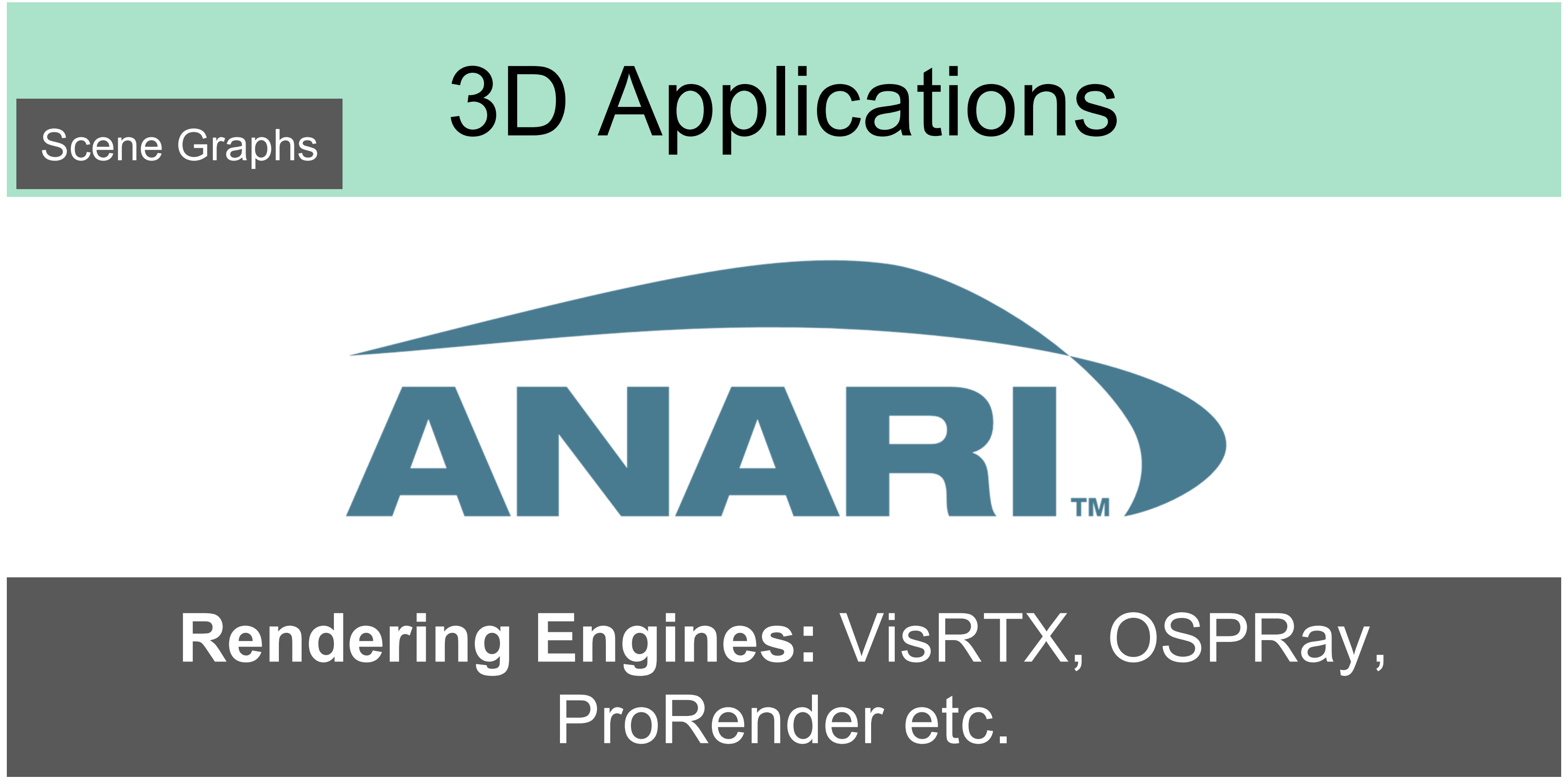
3D Applications

Scene Graphs

ANARI™

API BASICS

Software stack



3D Applications

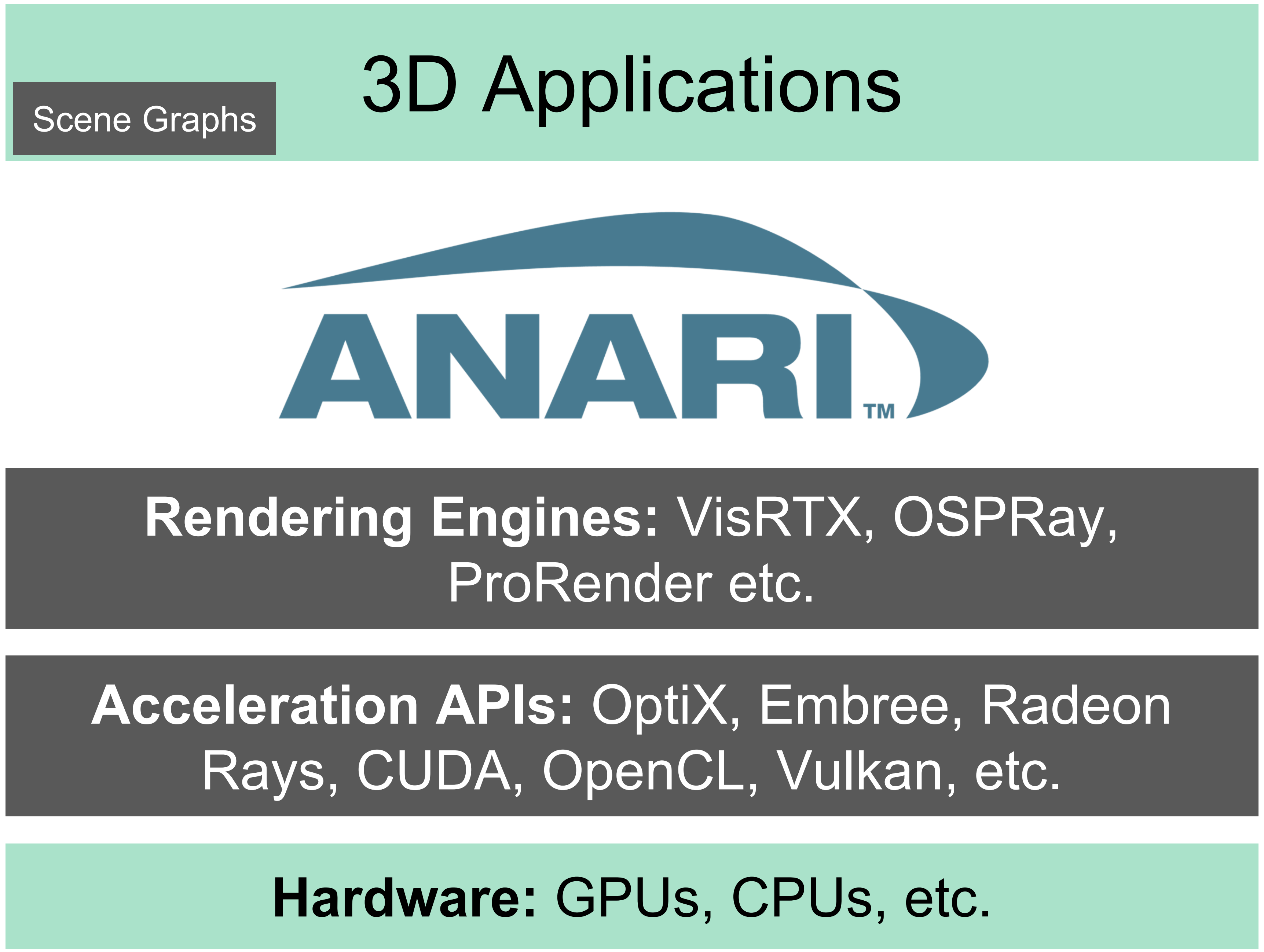
Scene Graphs

ANARI™

Rendering Engines: VisRTX, OSPRay, ProRender etc.

API BASICS

Software stack



3D Applications

Scene Graphs

ANARI™

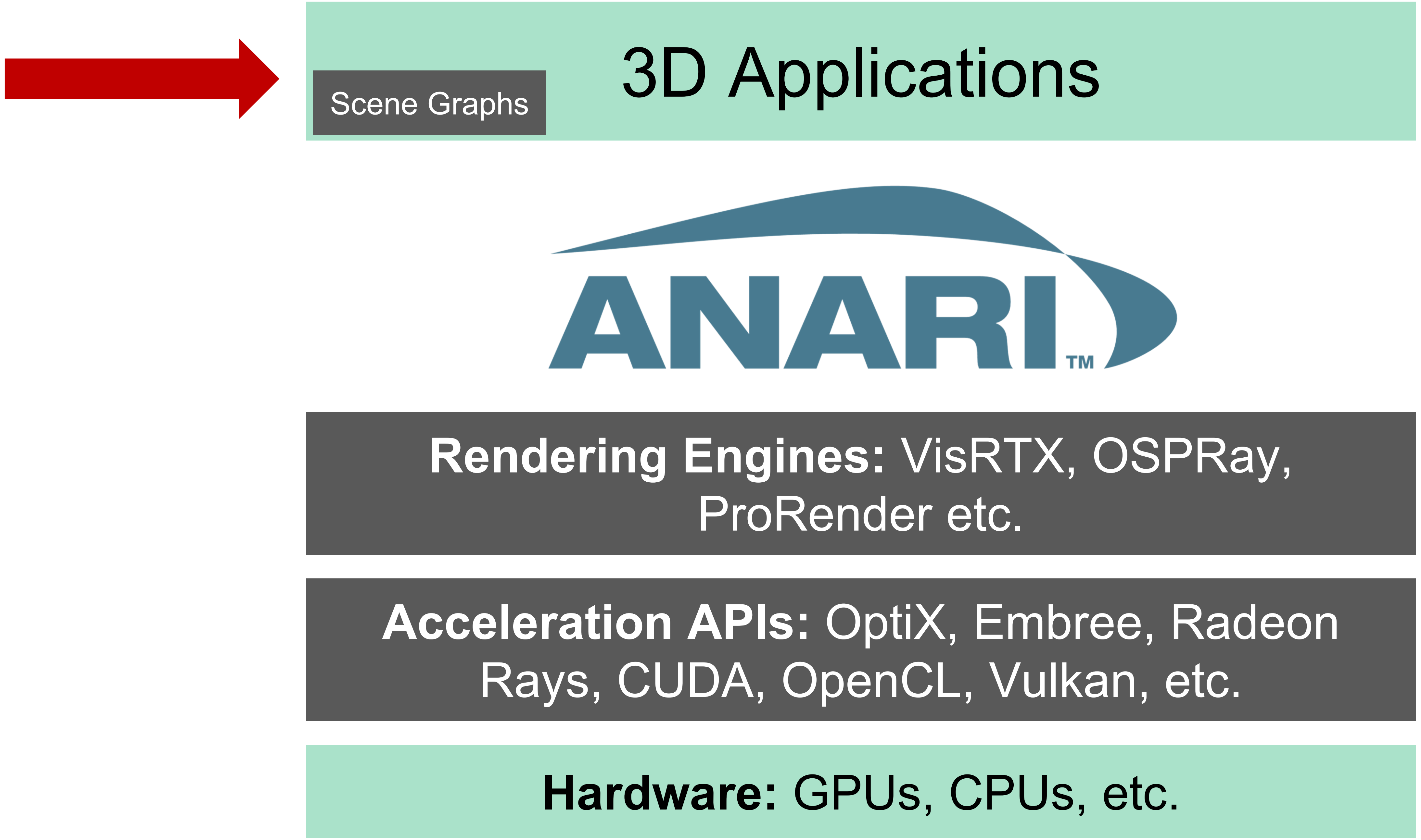
Rendering Engines: VisRTX, OSPRay, ProRender etc.

Acceleration APIs: OptiX, Embree, Radeon Rays, CUDA, OpenCL, Vulkan, etc.

Hardware: GPUs, CPUs, etc.

API BASICS

Software stack



Scene Graphs

3D Applications

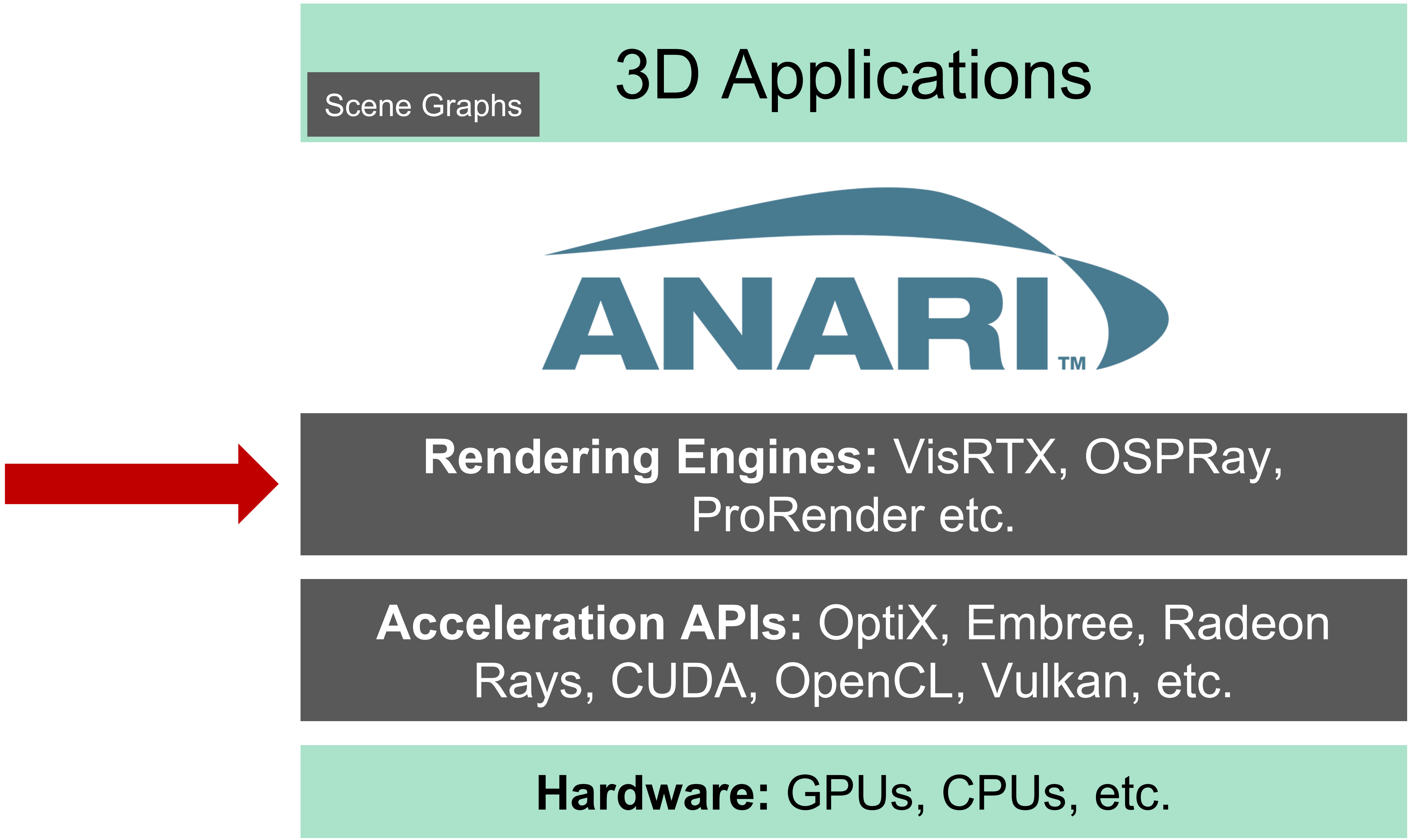
Rendering Engines: VisRTX, OSPRay, ProRender etc.

Acceleration APIs: OptiX, Embree, Radeon Rays, CUDA, OpenCL, Vulkan, etc.

Hardware: GPUs, CPUs, etc.

API BASICS

Software stack



3D Applications

Scene Graphs

ANARI™

Rendering Engines: VisRTX, OSPRay, ProRender etc.

Acceleration APIs: OptiX, Embree, Radeon Rays, CUDA, OpenCL, Vulkan, etc.

Hardware: GPUs, CPUs, etc.

API BASICS

Design choices

C99

API BASICS

Design choices

C99

Common front-end library

API BASICS

Design choices

C99

Common front-end library

SDK for "quality-of-life" extras: C++ bindings, debug tools, tests, etc.

API BASICS

Design choices

C99

Common front-end library

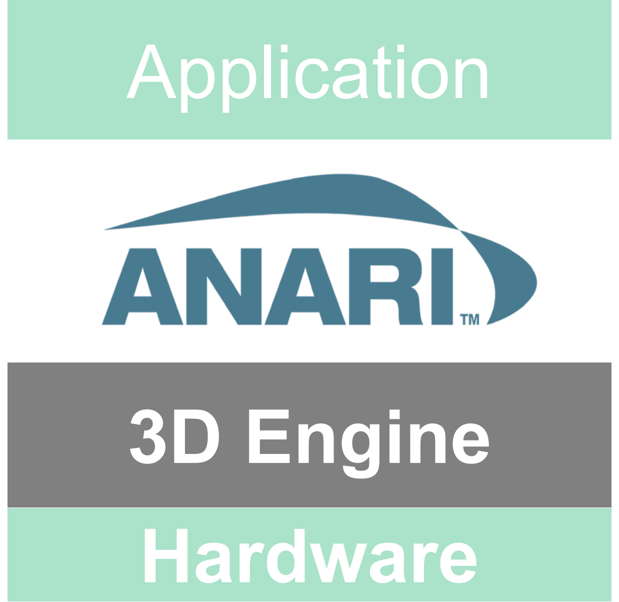
SDK for "quality-of-life" extras: C++ bindings, debug tools, tests, etc.

Single API to handle both local and distributed rendering (mobile up to clusters)

API BASICS

Design choices

Local Rendering



Application

ANARI

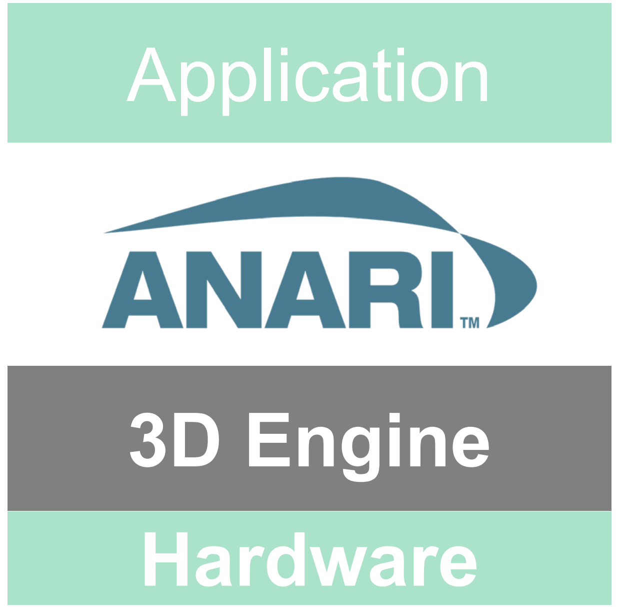
3D Engine

Hardware

API BASICS

Design choices

Local Rendering



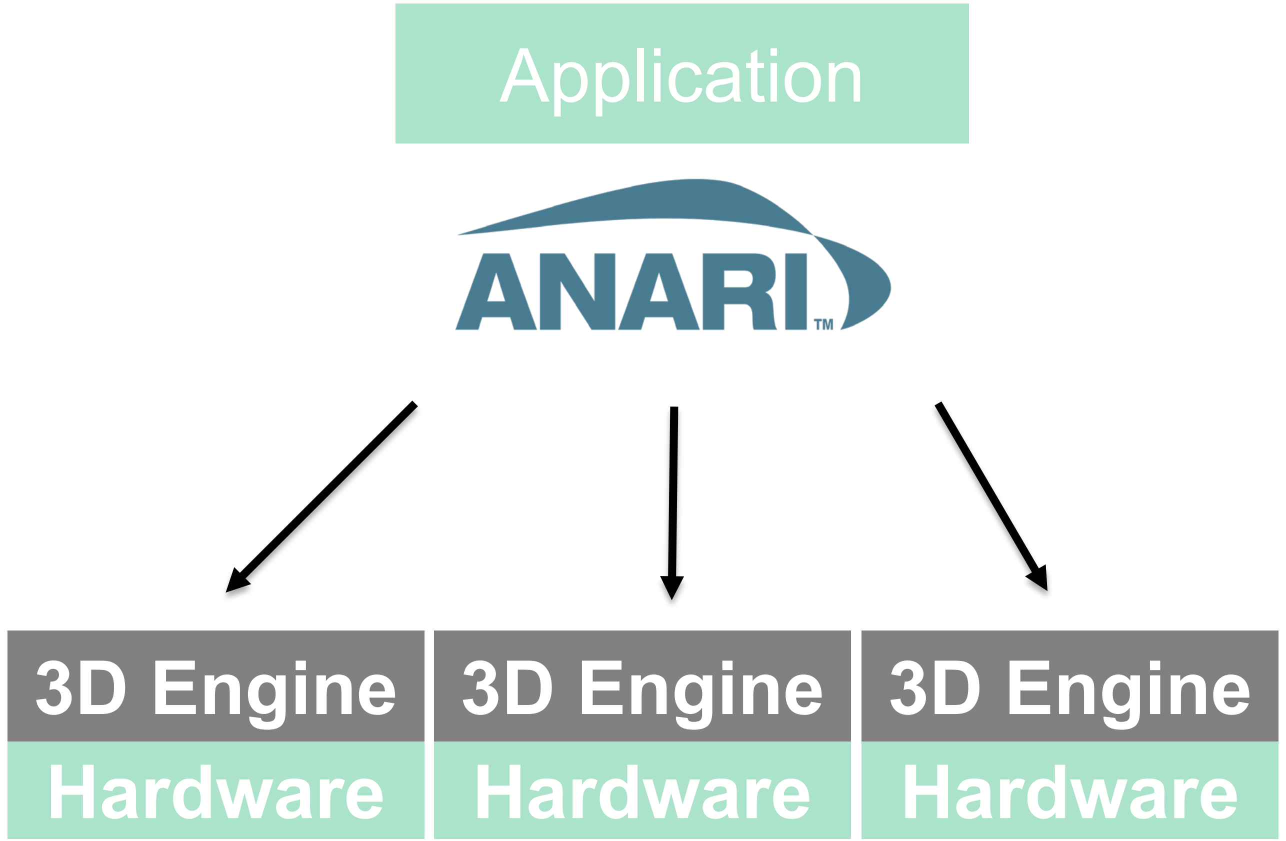
Application

ANARI

3D Engine

Hardware

Offload Rendering



Application

ANARI

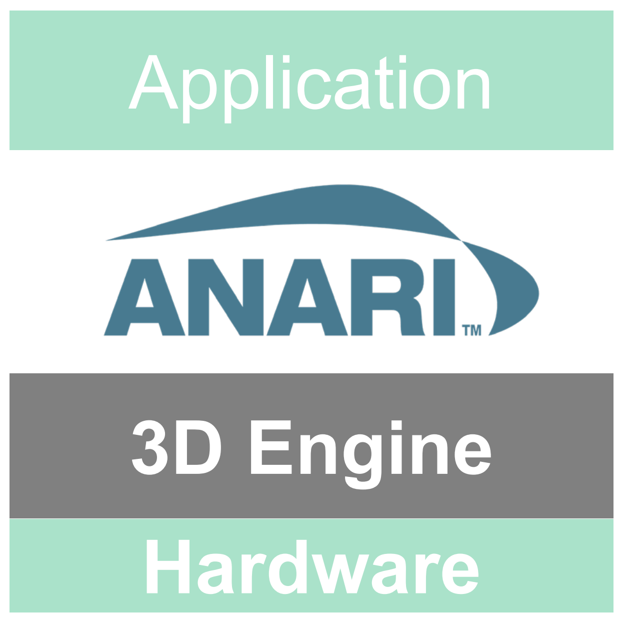
3D Engine | 3D Engine | 3D Engine

Hardware | Hardware | Hardware

API BASICS

Design choices

Local Rendering



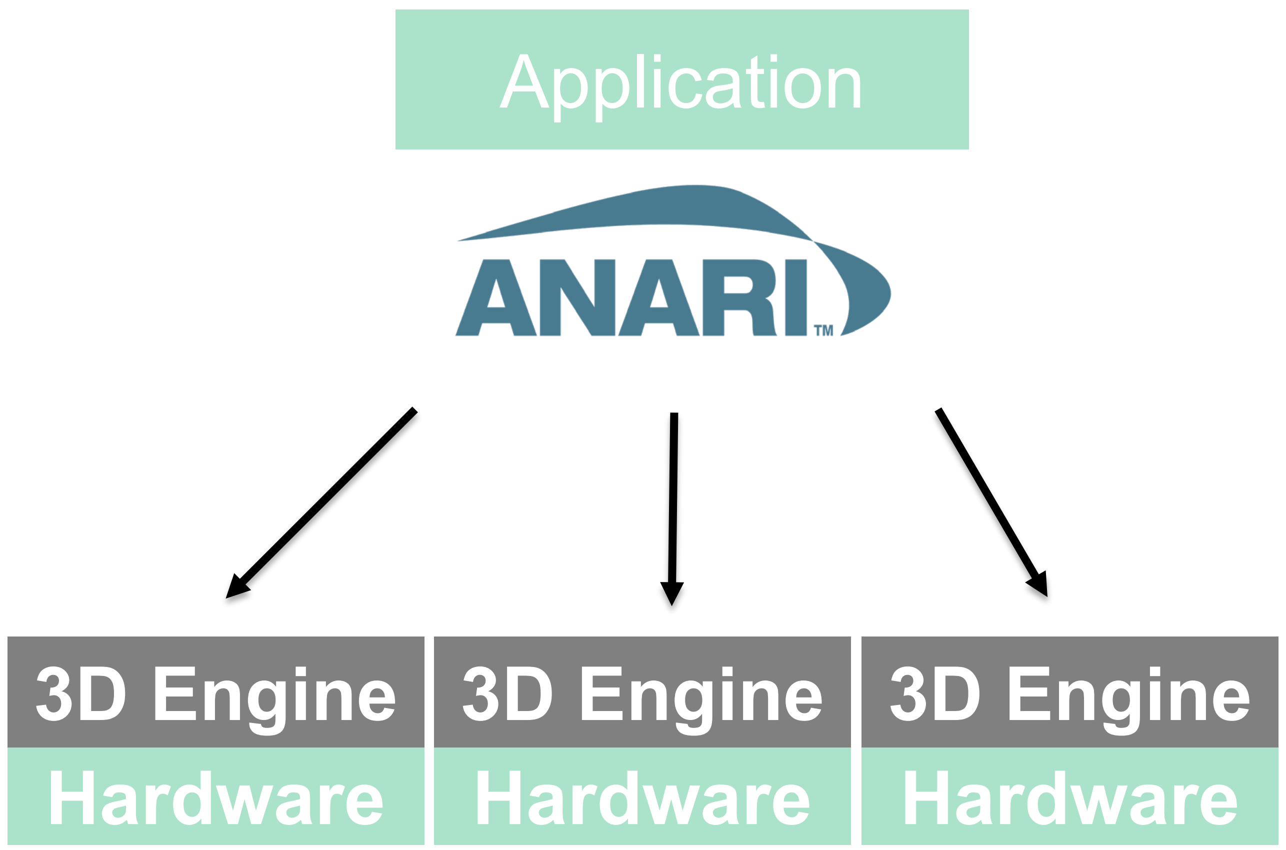
Application

ANARI

3D Engine

Hardware

Offload Rendering

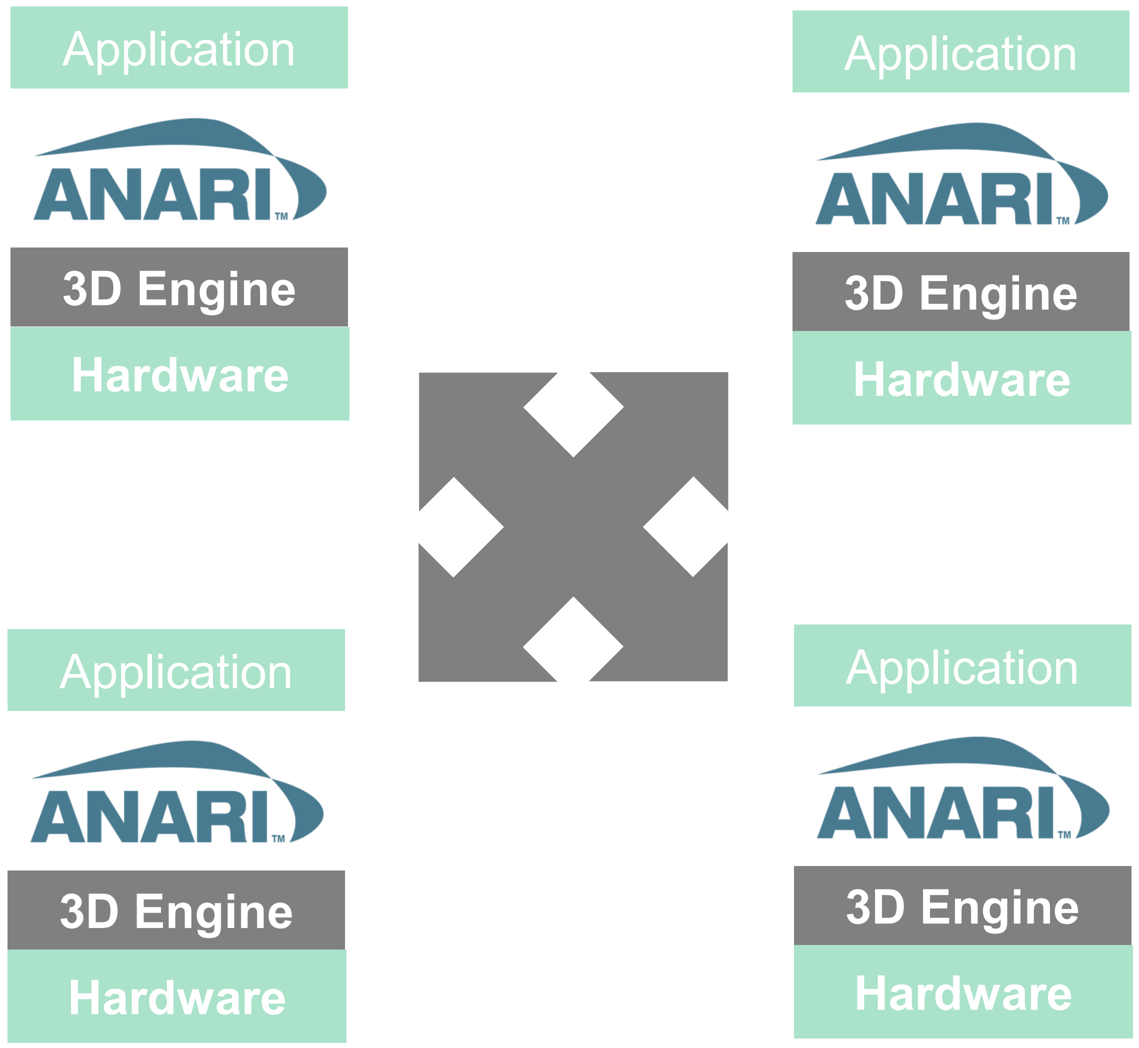


Application

ANARI

3D Engine	3D Engine	3D Engine
Hardware	Hardware	Hardware

Distributed Rendering



Application	Application
ANARI	ANARI
3D Engine	3D Engine
Hardware	Hardware
Application	Application
ANARI	ANARI
3D Engine	3D Engine
Hardware	Hardware

API BASICS

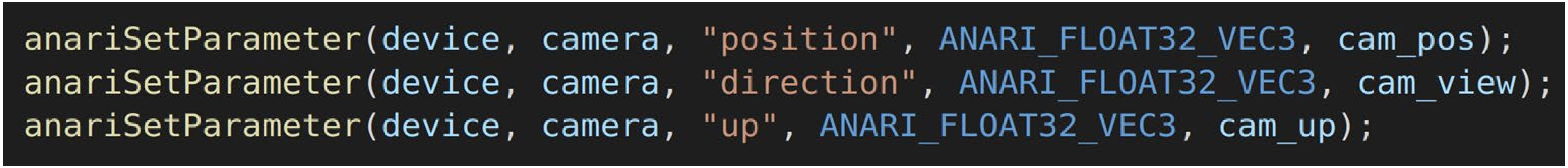
Devices

ANARI uses "software devices" to handle all API calls

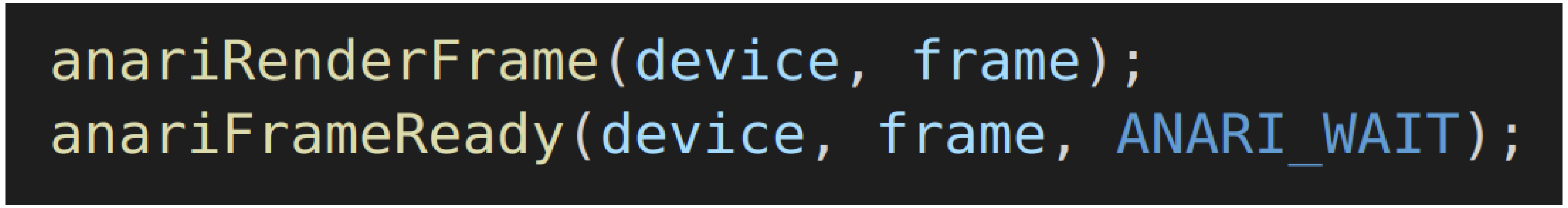
API BASICS

Devices

ANARI uses "software devices" to handle all API calls



```
anariSetParameter(device, camera, "position", ANARI_FLOAT32_VEC3, cam_pos);
anariSetParameter(device, camera, "direction", ANARI_FLOAT32_VEC3, cam_view);
anariSetParameter(device, camera, "up", ANARI_FLOAT32_VEC3, cam_up);
```



```
anariRenderFrame(device, frame);
anariFrameReady(device, frame, ANARI_WAIT);
```

API BASICS

Device extensions

ANARI extensions are optional features for a device to implement:

API BASICS

Device extensions

ANARI extensions are optional features for a device to implement:

- Object subtypes
- Extra object parameters and/or properties
- Enhanced core API semantics (e.g., thread safety)
- (rare) Extra API functions

API BASICS

Device extensions

ANARI extensions are optional features for a device to implement:

- Object subtypes
- Extra object parameters and/or properties
- Enhanced core API semantics (e.g., thread safety)
- (rare) Extra API functions

"Core extensions" exist in the specification

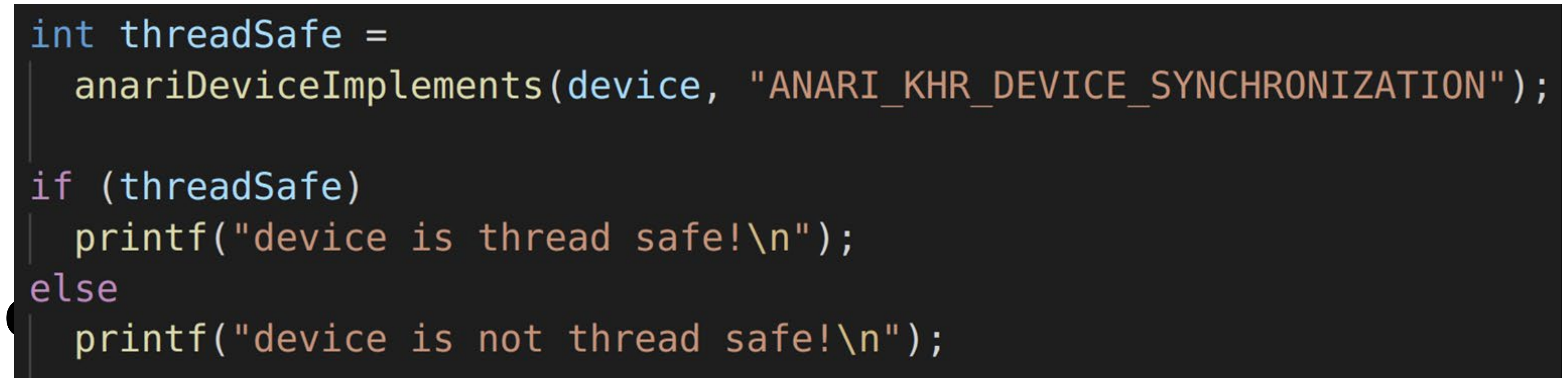
"Vendor extensions" are documented by adopters only

API BASICS

Device extensions

ANARI extensions are optional features for a device to implement:

- Object subtypes
- Extra object parameters and/or properties
- Enhanced core API semantics (e.g., thread safety)
- (rare) Extra API functions



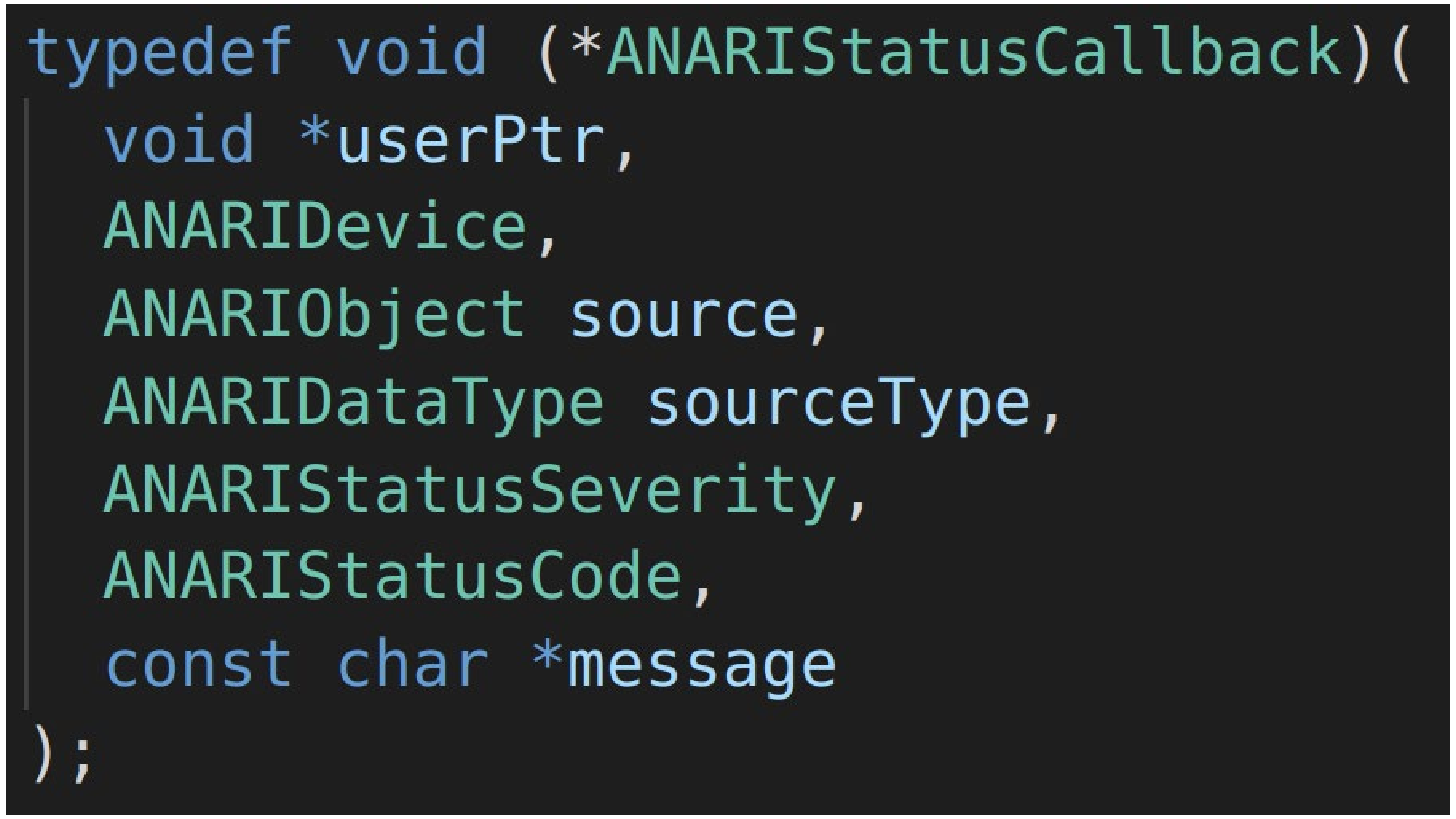
```
int threadSafe =
  anariDeviceImplements(device, "ANARI_KHR_DEVICE_SYNCHRONIZATION");

if (threadSafe)
  printf("device is thread safe!\n");
else
  printf("device is not thread safe!\n");
```

"Core extensions" exist in the specificati

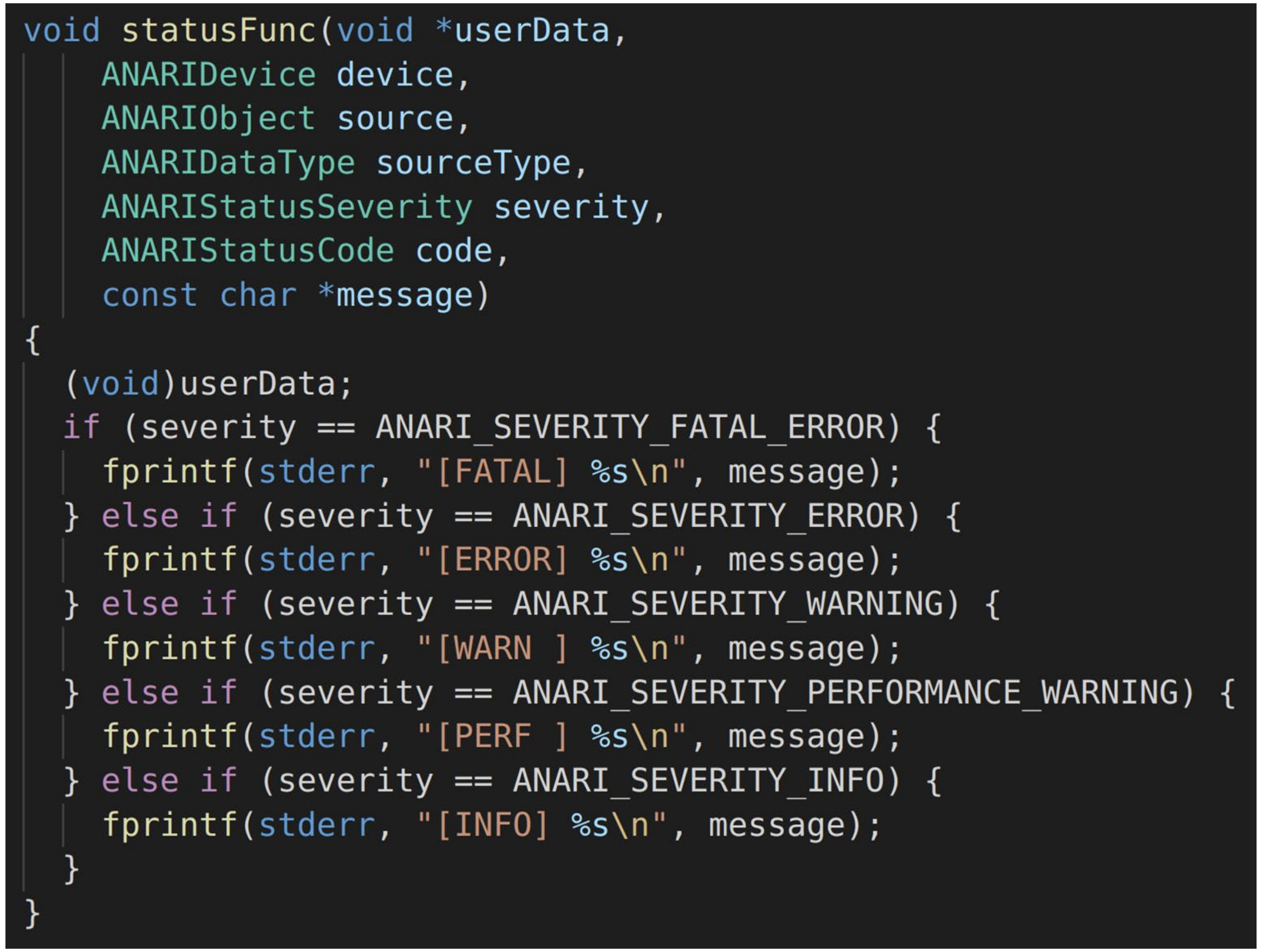
"Vendor extensions" are documented by adopters only

API BASICS

Error handling



```c
typedef void (*ANARIStatusCallback)(
  void *userPtr,
  ANARIDevice,
  ANARIObject source,
  ANARIDataType sourceType,
  ANARIStatusSeverity,
  ANARIStatusCode,
  const char *message
);
```



```c
void statusFunc(void *userData,
    ANARIDevice device,
    ANARIObject source,
    ANARIDataType sourceType,
    ANARIStatusSeverity severity,
    ANARIStatusCode code,
    const char *message)
{
  (void)userData;
  if (severity == ANARI_SEVERITY_FATAL_ERROR) {
    fprintf(stderr, "[FATAL] %s\n", message);
  } else if (severity == ANARI_SEVERITY_ERROR) {
    fprintf(stderr, "[ERROR] %s\n", message);
  } else if (severity == ANARI_SEVERITY_WARNING) {
    fprintf(stderr, "[WARN ] %s\n", message);
  } else if (severity == ANARI_SEVERITY_PERFORMANCE_WARNING) {
    fprintf(stderr, "[PERF ] %s\n", message);
  } else if (severity == ANARI_SEVERITY_INFO) {
    fprintf(stderr, "[INFO] %s\n", message);
  }
}
```

API BASICS

Handles and objects

Objects are characterized as:

API BASICS

Handles and objects

Objects are characterized as:

1. Represented by an opaque handle

2. Can take parameters

3. Can publish properties

4. Lifetime controlled by retain/release

API BASICS

Handles and objects

Objects are characterized as:

1. Represented by an opaque handle

2. Can take parameters

3. Can publish properties

4. Lifetime controlled by retain/release

Objects represent all scene "actors":

- geometry, materials, and surfaces
- spatial fields and volumes
- lights
- cameras
- renderers
- instances
- ...

API BASICS

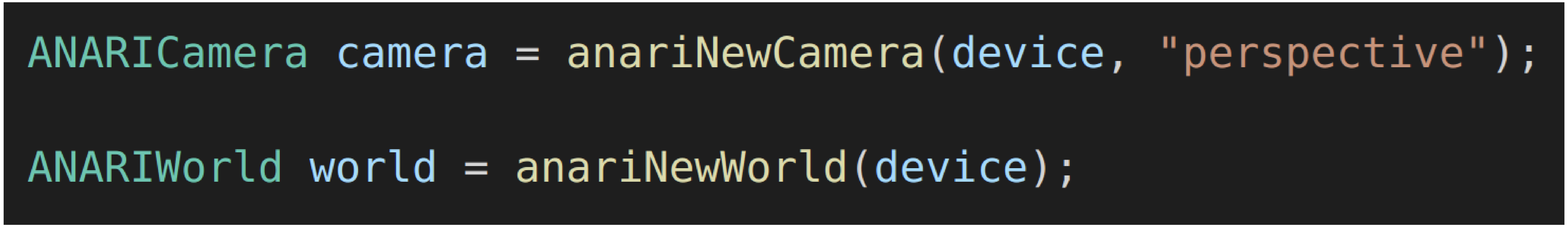
Creating objects + object lifetime

Objects are created with "anariNew" functions, sometimes with a subtype

API BASICS

Creating objects + object lifetime

Objects are created with "anariNew" functions, sometimes with a subtype

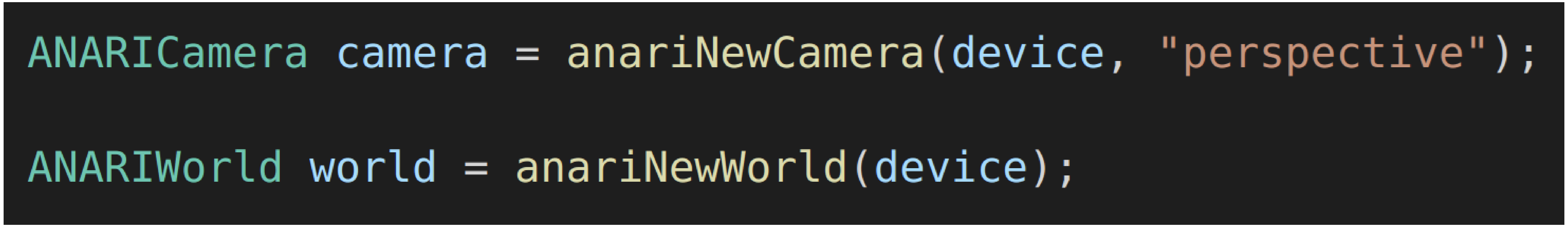


```
ANARICamera camera = anariNewCamera(device, "perspective");

ANARIWorld world = anariNewWorld(device);
```

API BASICS

Creating objects + object lifetime

Objects are created with "anariNew" functions, sometimes with a subtype



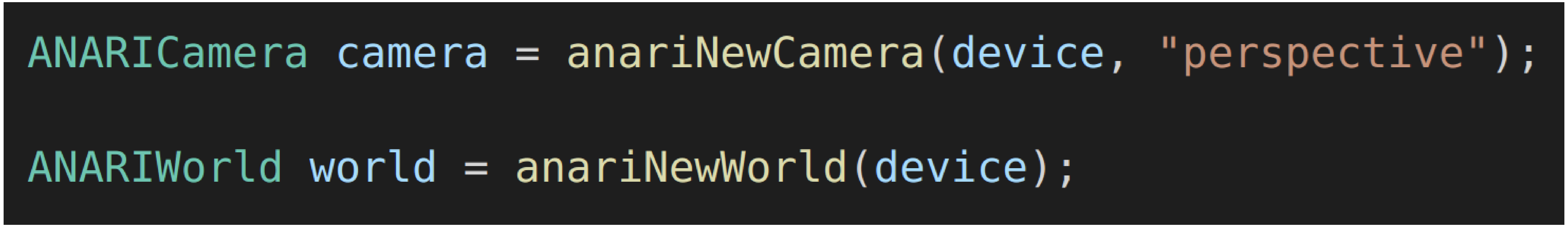
```
ANARICamera camera = anariNewCamera(device, "perspective");

ANARIWorld world = anariNewWorld(device);
```

Object lifetime is tracked by reference count, which is modified by anariRelease() and anariRetain()

API BASICS

Creating objects + object lifetime

Objects are created with "anariNew" functions, sometimes with a subtype



```
ANARICamera camera = anariNewCamera(device, "perspective");

ANARIWorld world = anariNewWorld(device);
```

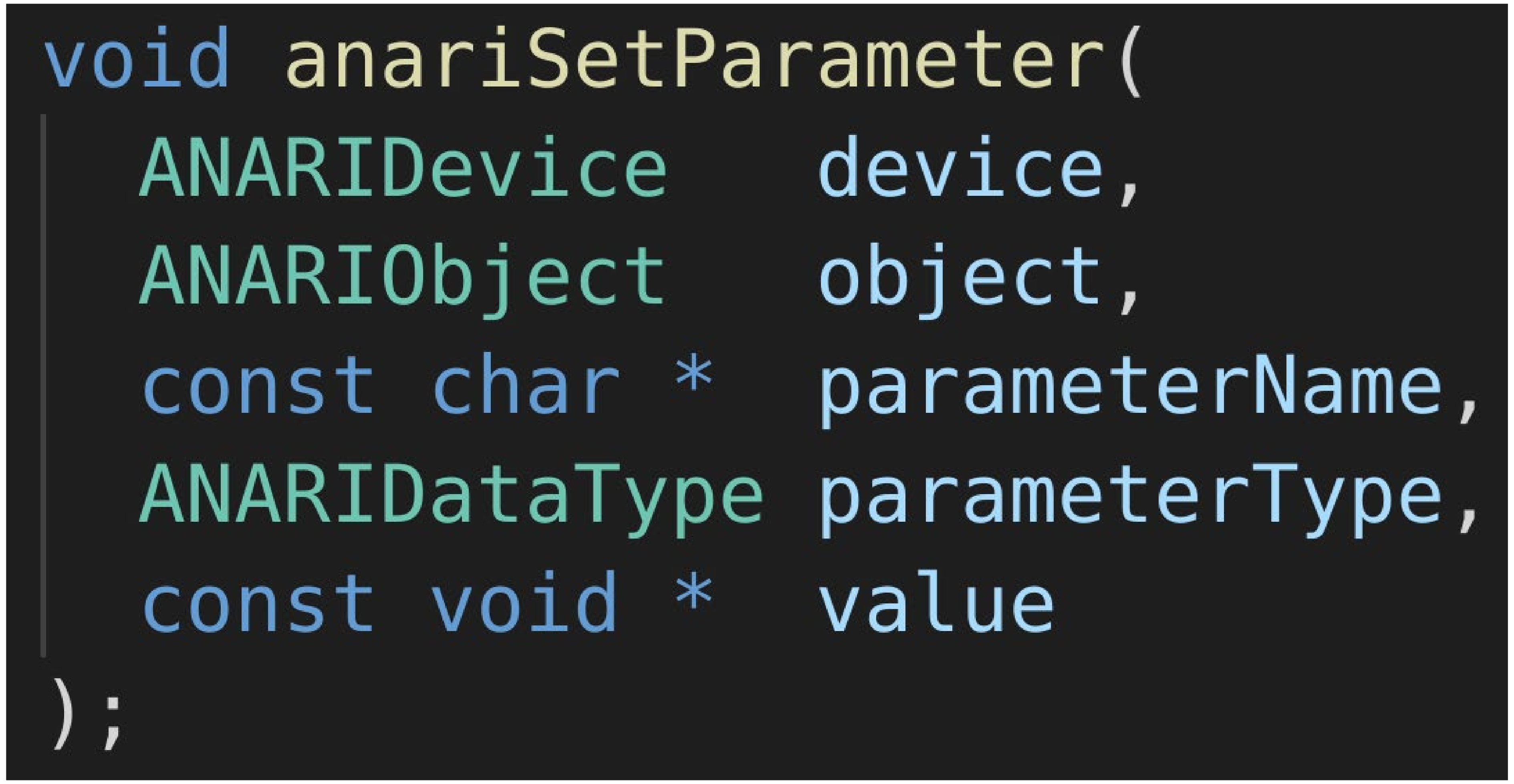
Object lifetime is tracked by reference count, which is modified by anariRelease() and anariRetain()

Objects which refer to other objects may keep them around if necessary

API BASICS

Setting parameters

Parameters are all set via one API call



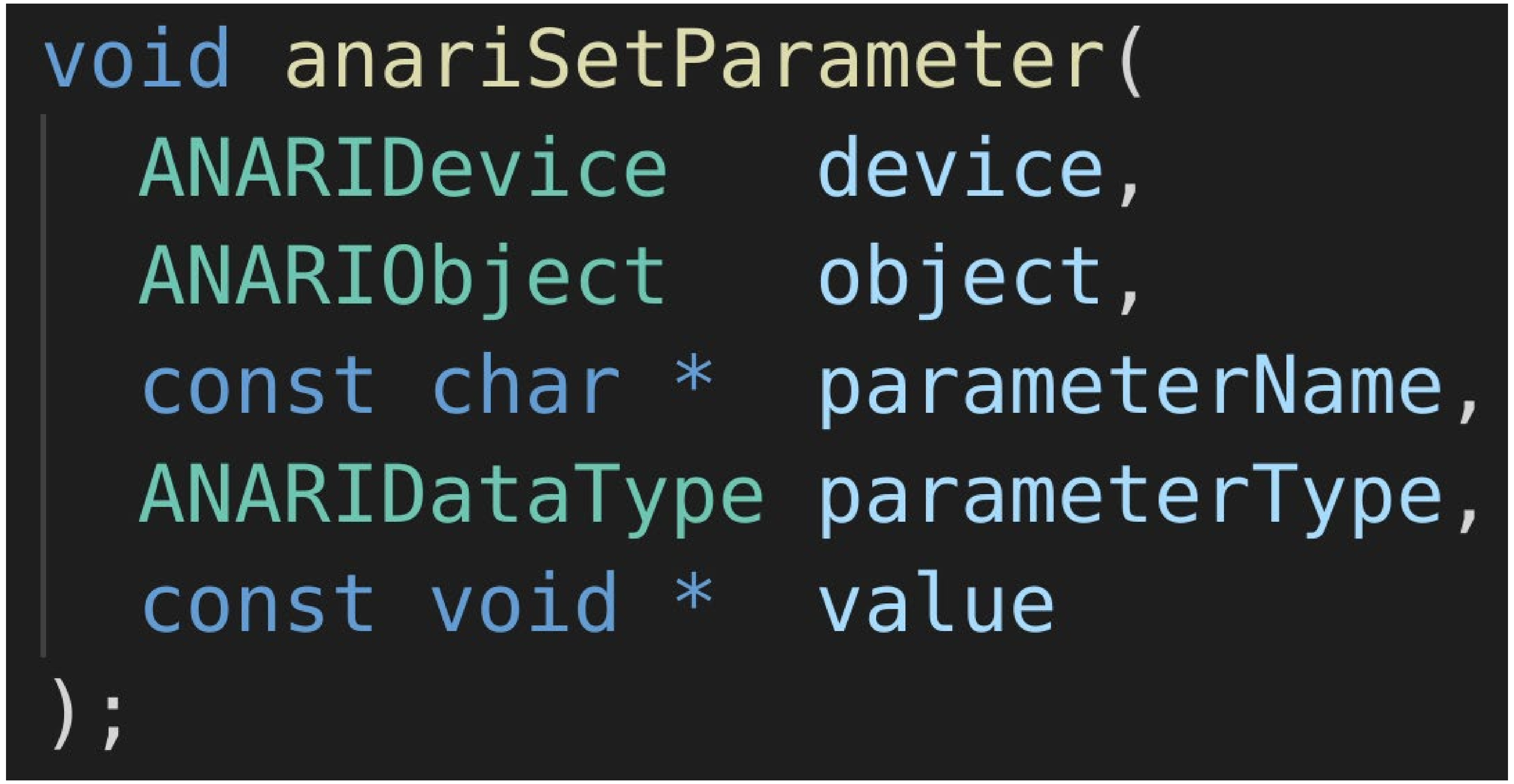
```
void anariSetParameter(
   ANARIDevice    device,
   ANARIObject    object,
   const char *   parameterName,
   ANARIDataType  parameterType,
   const void *   value
);
```

API BASICS

Setting parameters

Parameters are all set via one API call

All parameters are uniquely identified with a string name/value pair



```
void anariSetParameter(
  ANARIDevice    device,
  ANARIObject    object,
  const char *   parameterName,
  ANARIDataType  parameterType,
  const void *   value
);
```

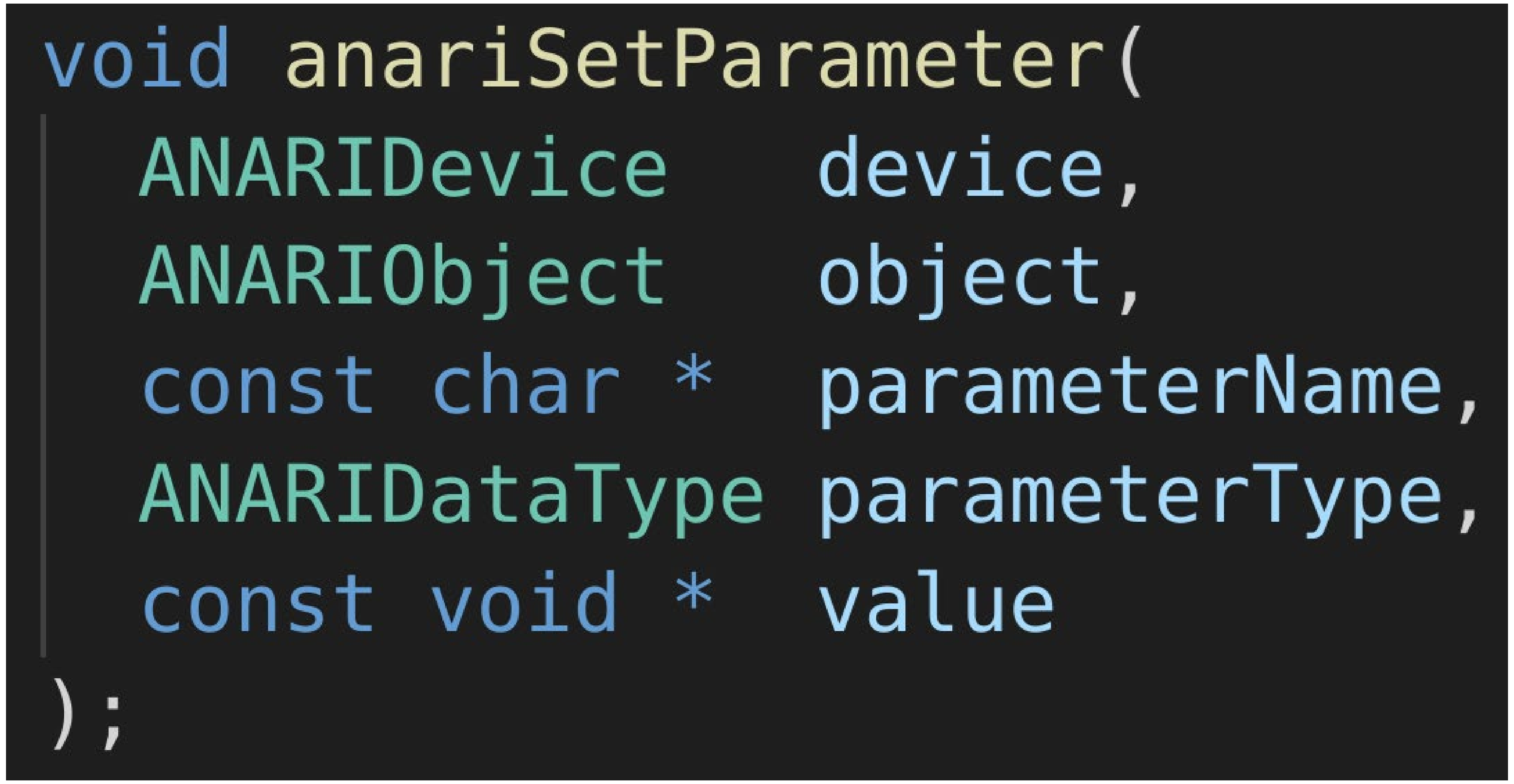
API BASICS

Setting parameters

Parameters are all set via one API call

All parameters are uniquely identified with a string name/value pair

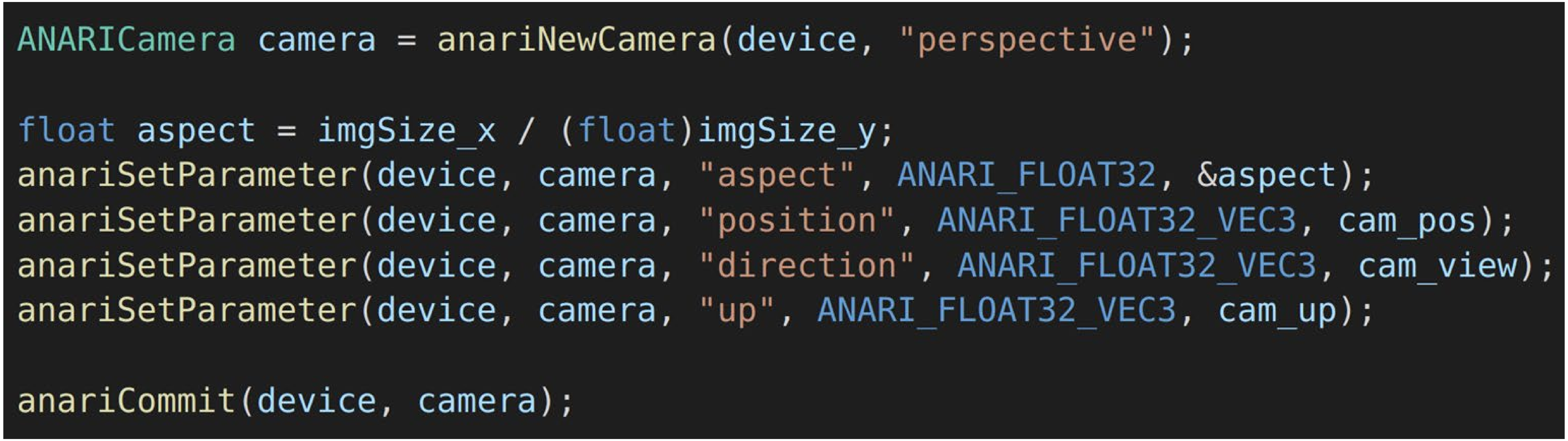
Parameters which are not used are ignored (warnings may be emitted)



```
void anariSetParameter(
  ANARIDevice    device,
  ANARIObject    object,
  const char *   parameterName,
  ANARIDataType  parameterType,
  const void *   value
);
```

API BASICS

Setting parameters



```
ANARICamera camera = anariNewCamera(device, "perspective");

float aspect = imgSize_x / (float)imgSize_y;
anariSetParameter(device, camera, "aspect", ANARI_FLOAT32, &aspect);
anariSetParameter(device, camera, "position", ANARI_FLOAT32_VEC3, cam_pos);
anariSetParameter(device, camera, "direction", ANARI_FLOAT32_VEC3, cam_view);
anariSetParameter(device, camera, "up", ANARI_FLOAT32_VEC3, cam_up);

anariCommit(device, camera);
```

API BASICS

Committing parameters

Staged values

Name	Type	Value
up	FLOAT32_VEC3	(0, 1, 0)
direction	FLOAT32_VEC3	(1, 0, 0)
position	FLOAT32_VEC3	(0, 0, 0)

Live values

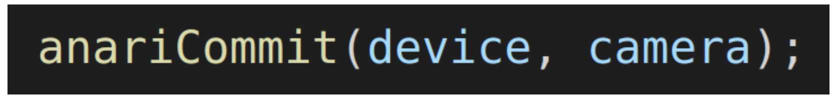
Name	Type	Value
up	FLOAT32_VEC3	
direction	FLOAT32_VEC3	
position	FLOAT32_VEC3	

API BASICS

Committing parameters

Staged values

Name	Type	Value
up	FLOAT32_VEC3	(0, 1, 0)
direction	FLOAT32_VEC3	(1, 0, 0)
position	FLOAT32_VEC3	(0, 0, 0)



```
anariCommit(device, camera);
```

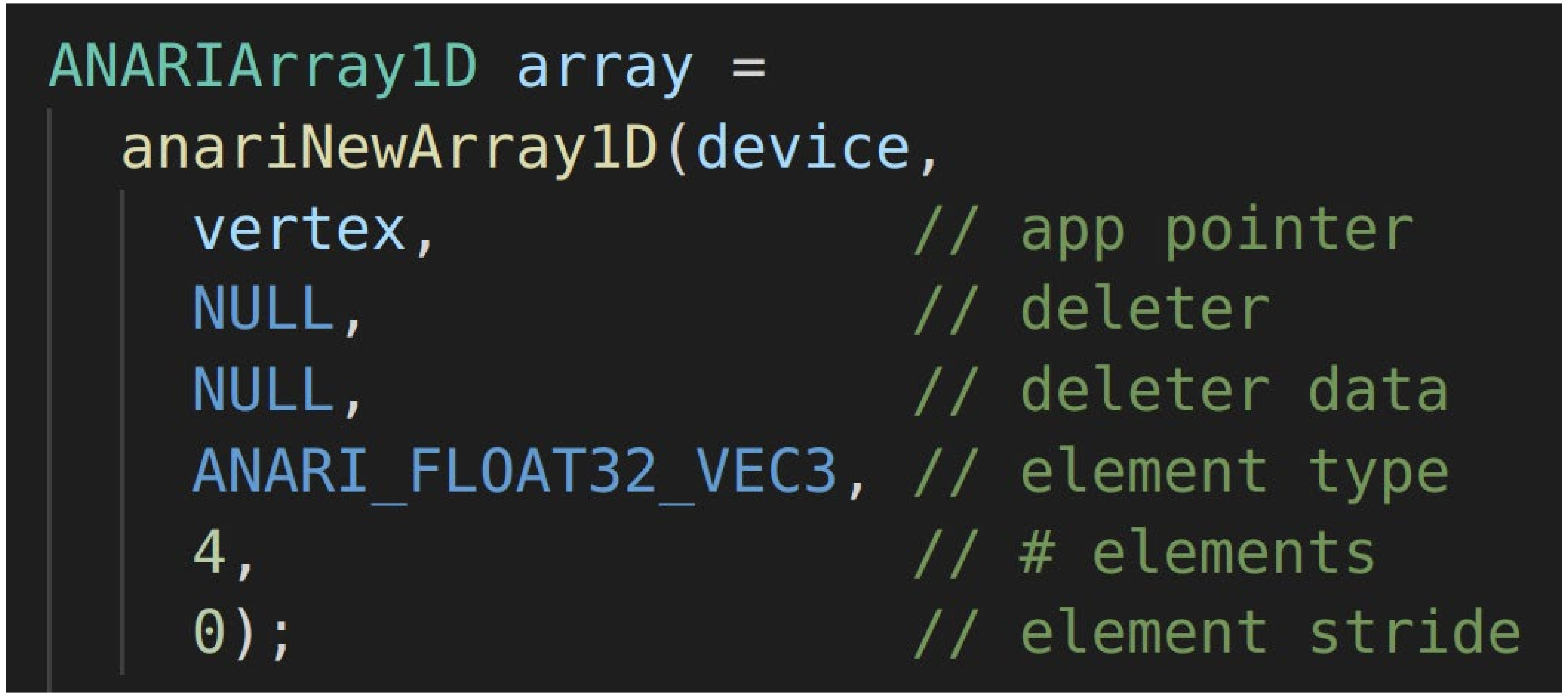
Live values

Name	Type	Value
up	FLOAT32_VEC3	(0, 1, 0)
direction	FLOAT32_VEC3	(1, 0, 0)
position	FLOAT32_VEC3	(0, 0, 0)

API BASICS

Arrays

Arrays are described by objects



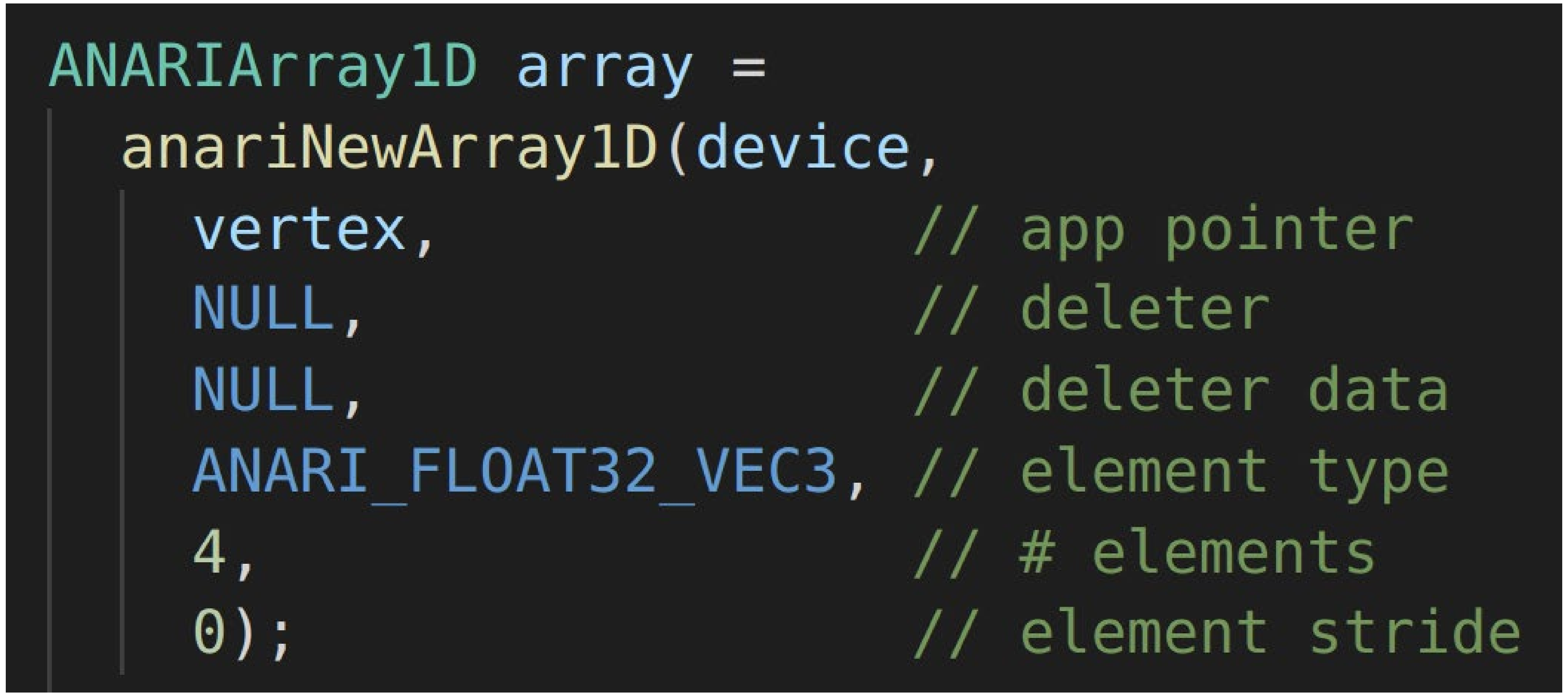
```
ANARIArray1D array =
  anariNewArray1D(device,
    vertex,             // app pointer
    NULL,               // deleter
    NULL,               // deleter data
    ANARI_FLOAT32_VEC3, // element type
    4,                  // # elements
    0);                 // element stride
```

API BASICS

Arrays

Arrays are described by objects

Can have shared ownership with the application or be opaquely handled by the ANARI device



```
ANARIArray1D array =
  anariNewArray1D(device,
    vertex,             // app pointer
    NULL,               // deleter
    NULL,               // deleter data
    ANARI_FLOAT32_VEC3, // element type
    4,                  // # elements
    0);                 // element stride
```

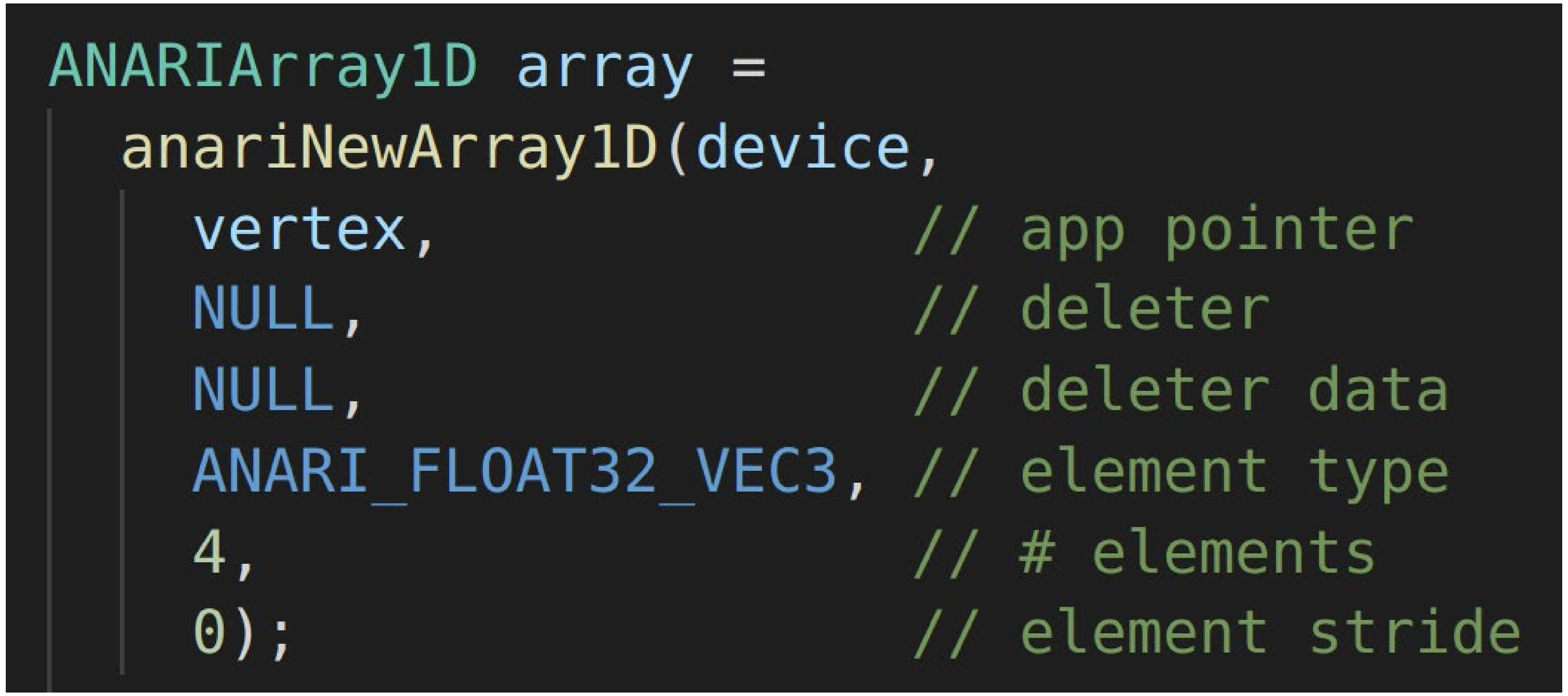
API BASICS

Arrays

Arrays are described by objects

Can have shared ownership with the application or be opaquely handled by the ANARI device

Array data can be updated through mapping

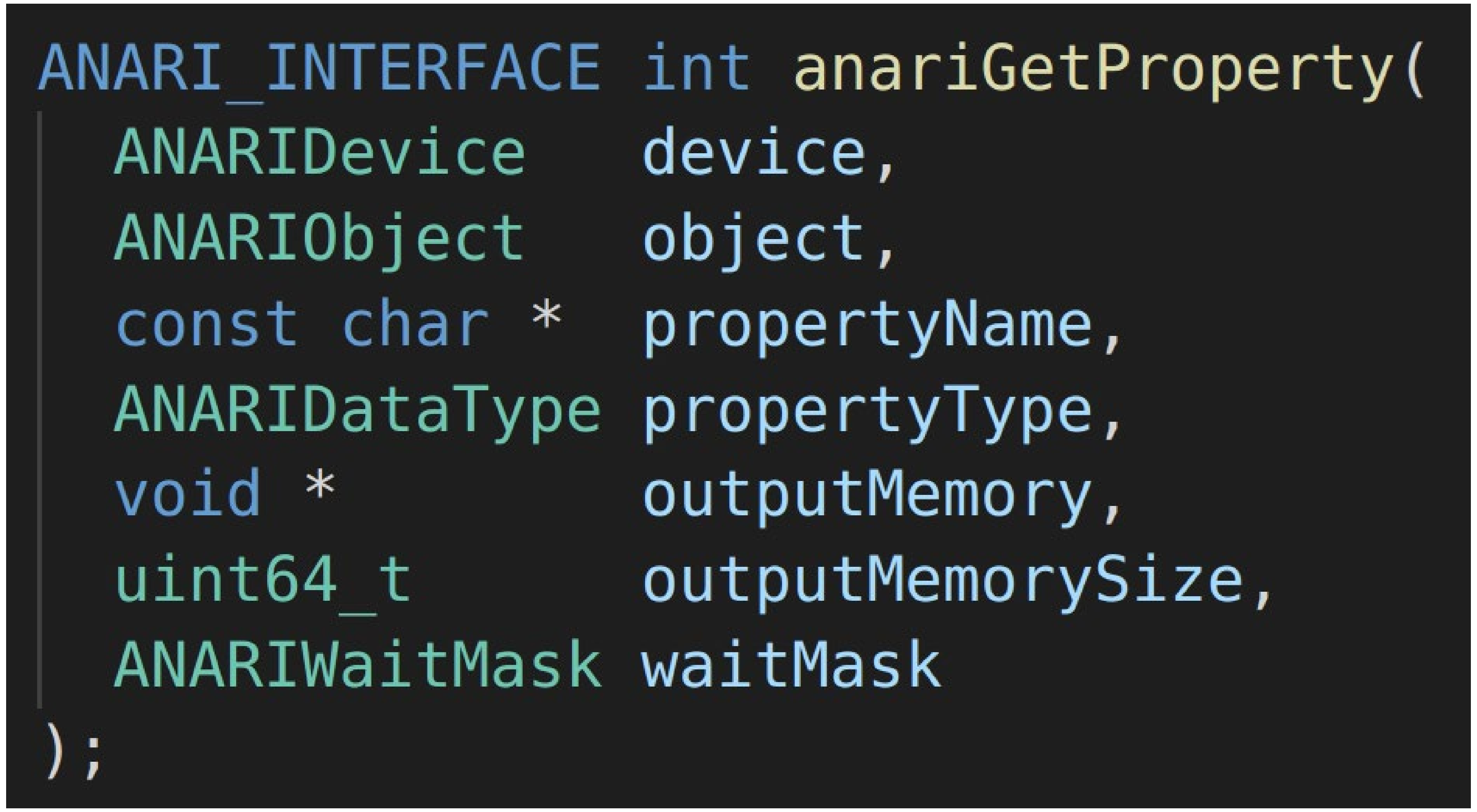


```
ANARIArray1D array =
  anariNewArray1D(device,
    vertex,            // app pointer
    NULL,              // deleter
    NULL,              // deleter data
    ANARI_FLOAT32_VEC3, // element type
    4,                 // # elements
    0);                // element stride
```

API BASICS

Properties

Properties represent published values an application can read



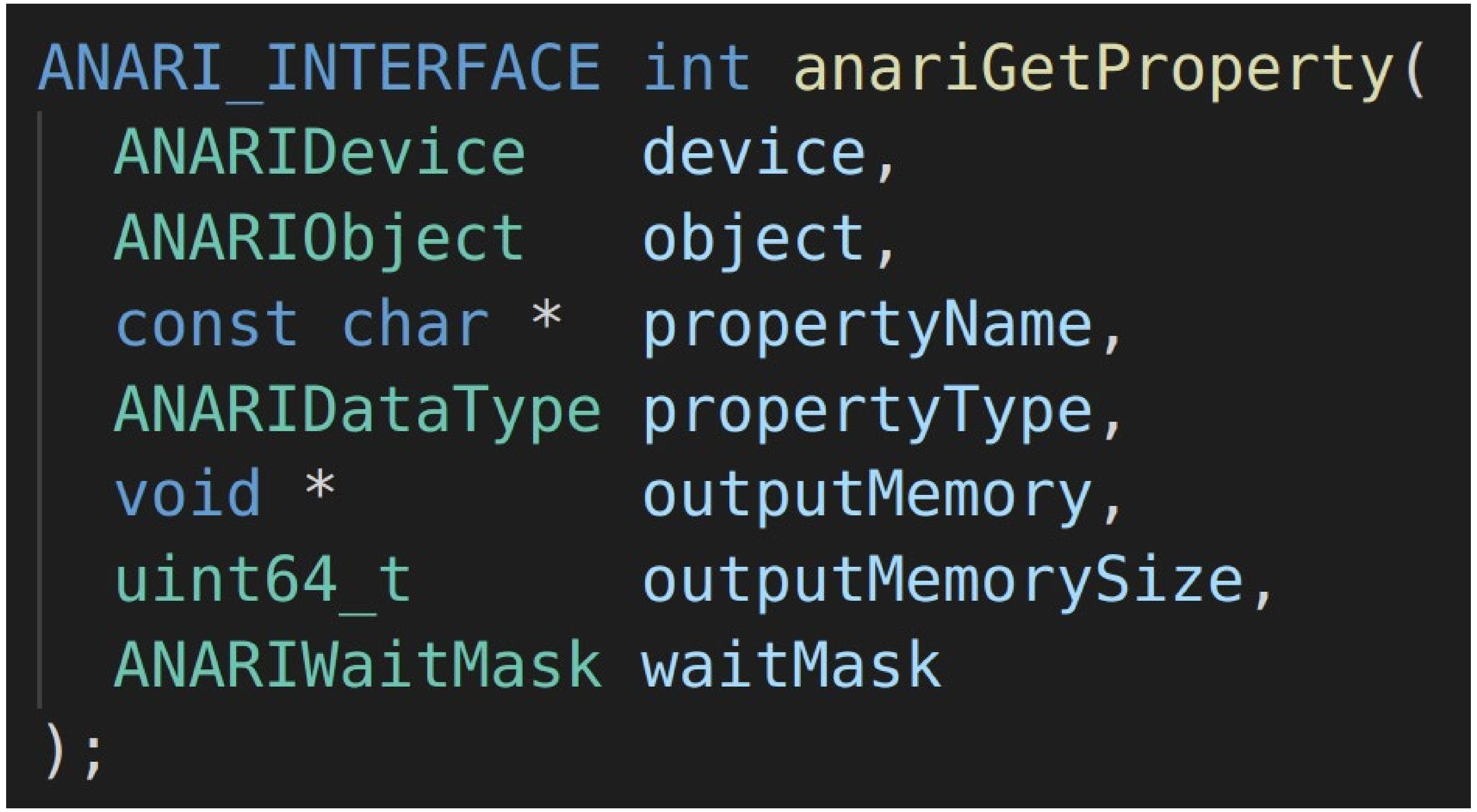
```
ANARI_INTERFACE int anariGetProperty(
  ANARIDevice   device,
  ANARIObject   object,
  const char *  propertyName,
  ANARIDataType propertyType,
  void *        outputMemory,
  uint64_t      outputMemorySize,
  ANARIWaitMask waitMask
);
```

API BASICS

Properties

Properties represent published values an application can read

Properties are not intrinsically tied to parameters



```
ANARI_INTERFACE int anariGetProperty(
  ANARIDevice    device,
  ANARIObject    object,
  const char *   propertyName,
  ANARIDataType propertyType,
  void *         outputMemory,
  uint64_t       outputMemorySize,
  ANARIWaitMask waitMask
);
```

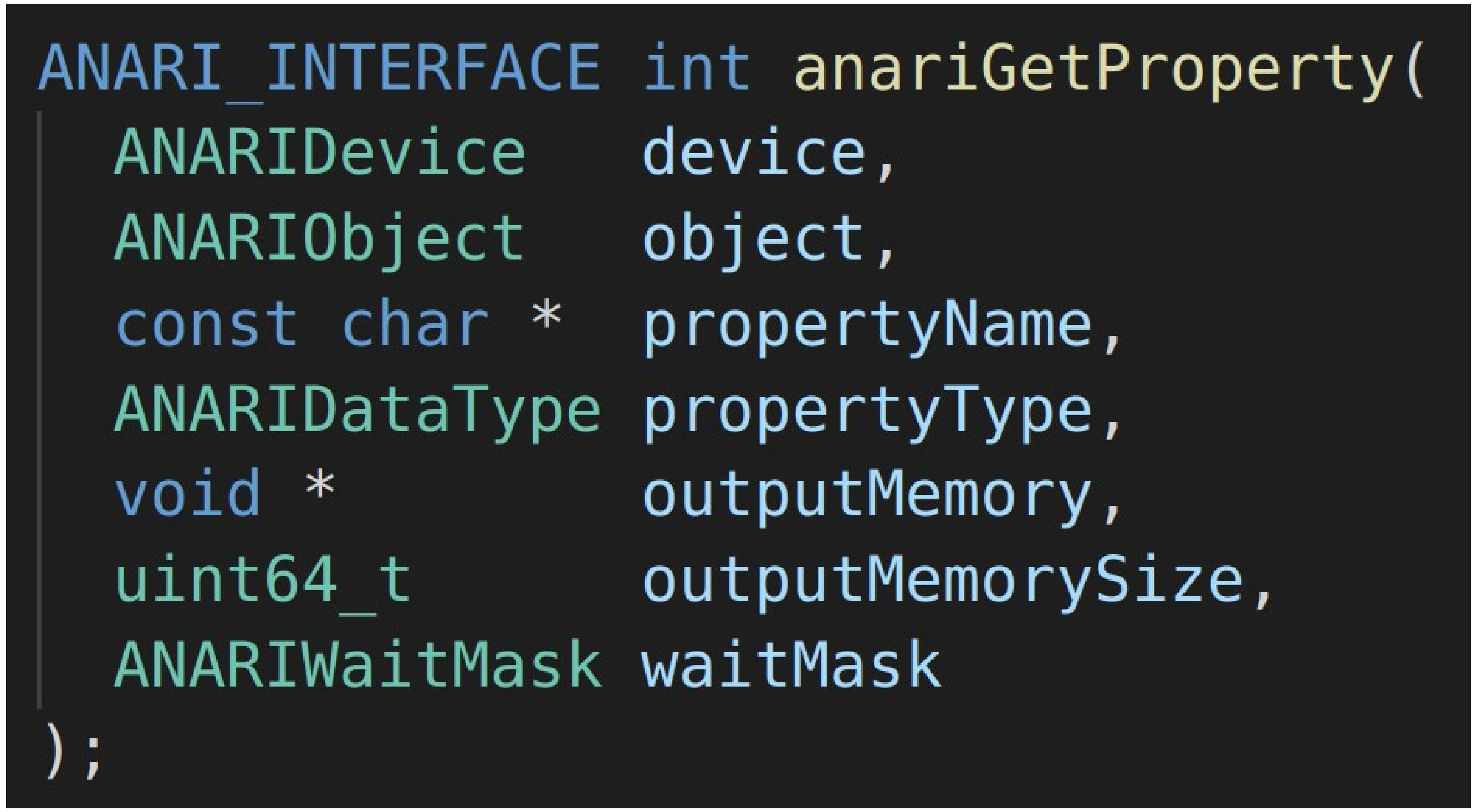
API BASICS

Properties

Properties represent published values an application can read

Properties are not intrinsically tied to parameters

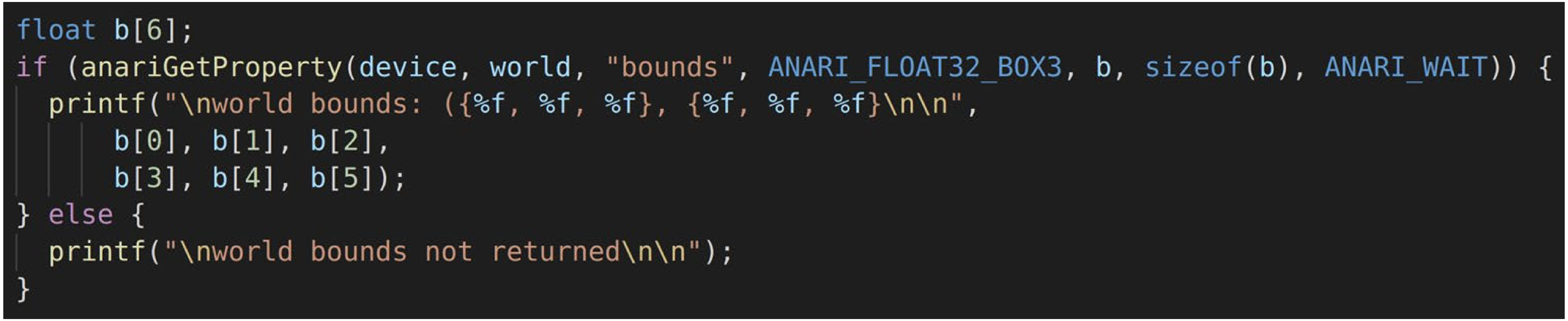
Property queries can be asynchronous



```
ANARI_INTERFACE int anariGetProperty(
  ANARIDevice   device,
  ANARIObject   object,
  const char *  propertyName,
  ANARIDataType propertyType,
  void *        outputMemory,
  uint64_t      outputMemorySize,
  ANARIWaitMask waitMask
);
```

API BASICS

Property query example



```
float b[6];
if (anariGetProperty(device, world, "bounds", ANARI_FLOAT32_BOX3, b, sizeof(b), ANARI_WAIT)) {
  printf("\nworld bounds: ({%f, %f, %f}, {%f, %f, %f}\n\n",
      b[0], b[1], b[2],
      b[3], b[4], b[5]);
} else {
  printf("\nworld bounds not returned\n\n");
}
```

OBJECT OVERVIEW

Object types (1)

ANARIDevice - implementation object

ANARIFrame - top-level object holding everything necessary to render an image

ANARICamera - view projection object

ANARIRenderer - rendering algorithm condfigured by its parameters

ANARIWorld - top-level object holding all objects which can be "seen"

ANARIGroup - a collection of lights, surfaces, and volumes which share an object coordinate system

ANARIInstance - transform ANARIGroup into world-space

ANARIArray - describes an array of values: element type, number of elements, and memory ownership

OBJECT OVERVIEW

Object types (2)

ANARIGeometry - the mathmatical 3D definition of a viewable surface object (+ its data) in a local coordinate system

ANARIMaterial - the parameterized "look" of a surface

ANARISampler - maps data on an ANARIGeometry into the inputs of ANARIMaterial

ANARISurface - concretely ties together ANARIGeometry and ANARIMaterial

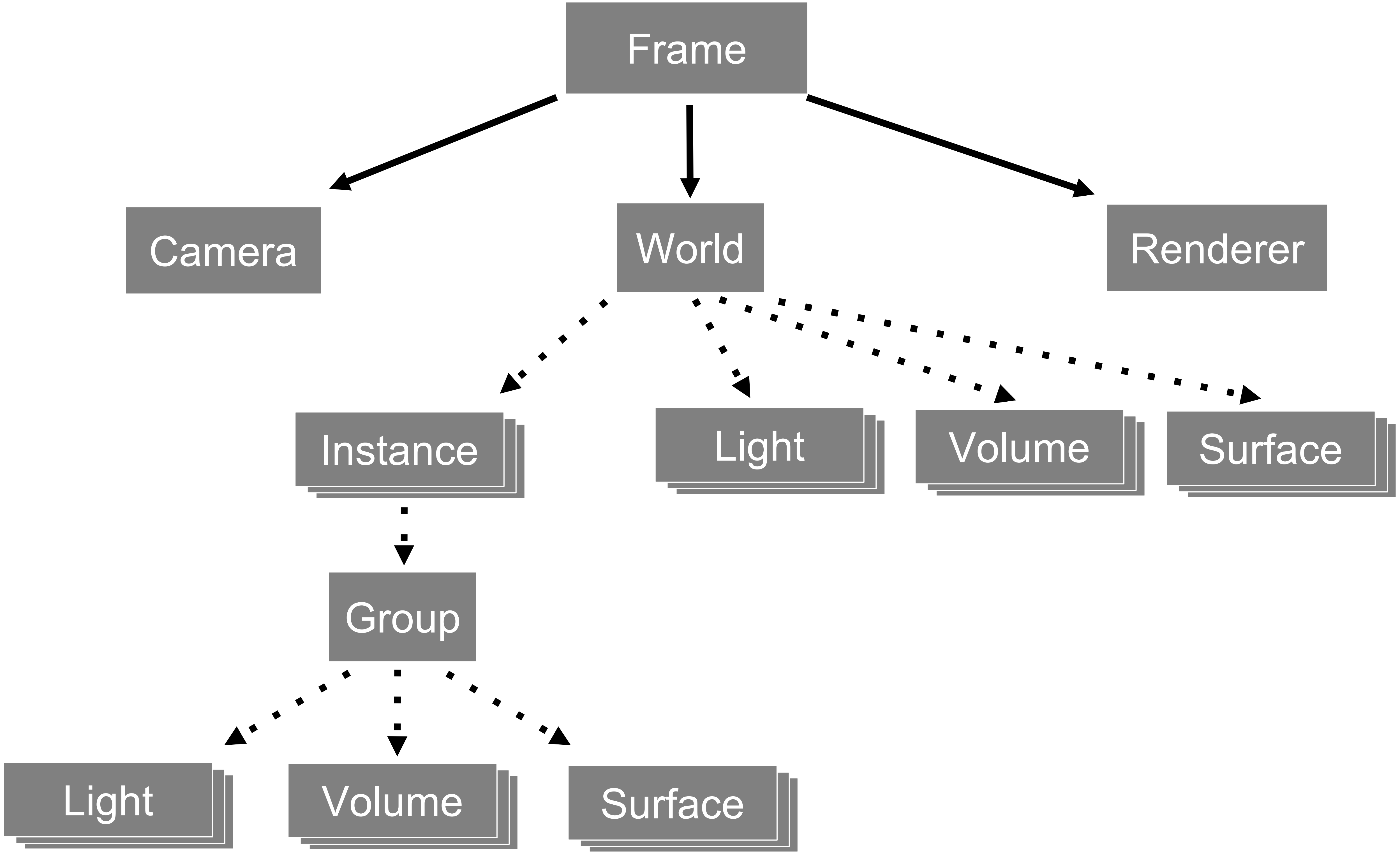
ANARISpatialField - a collection of values which can be sampled within a common local coordinate system

ANARIVolume - the parameterized "look" of a volumetric object using one or more ANARISpatialField objects as input

ANARILight - casts illumination into the scene

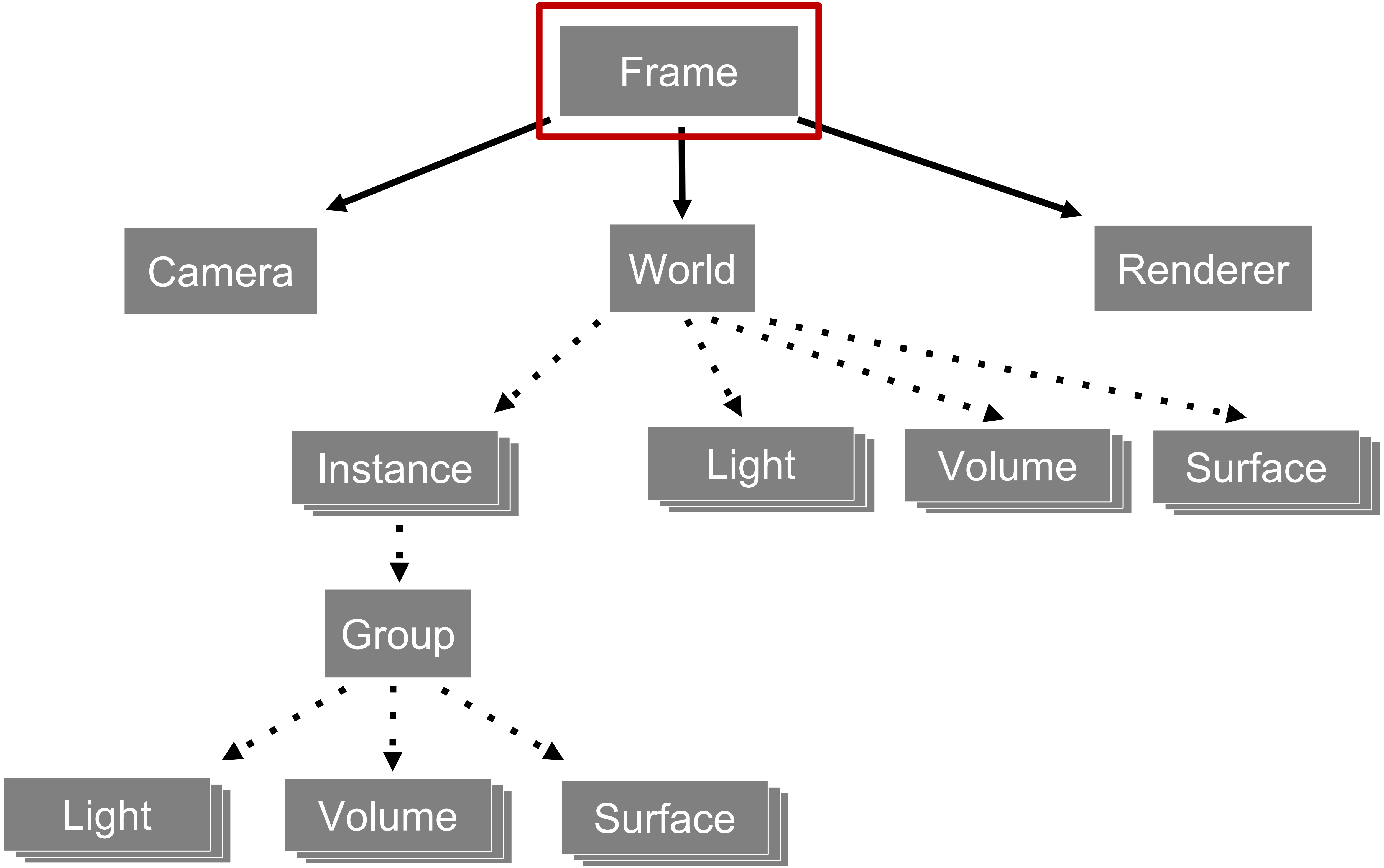
OBJECT OVERVIEW

Object hierarchy



OBJECT OVERVIEW

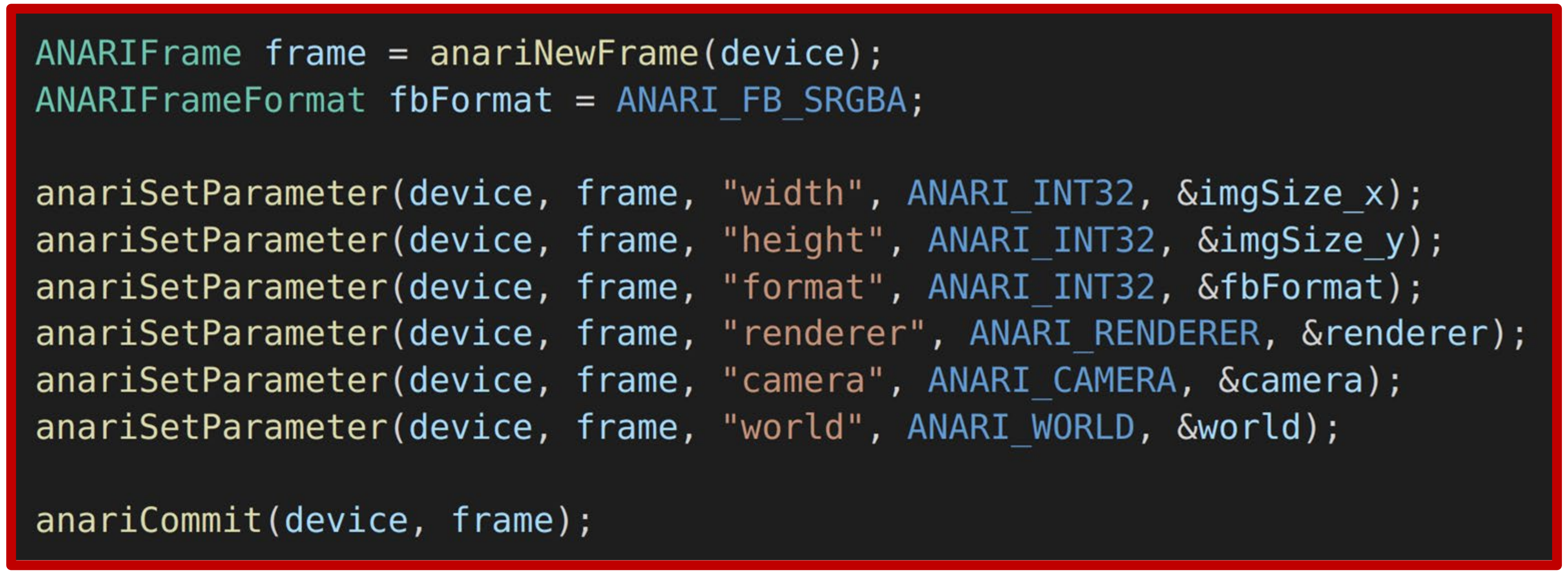
Object hierarchy



OBJECT OVERVIEW

ANARIFrame

ANARIFrame represents the top-level object in the object hierarchy

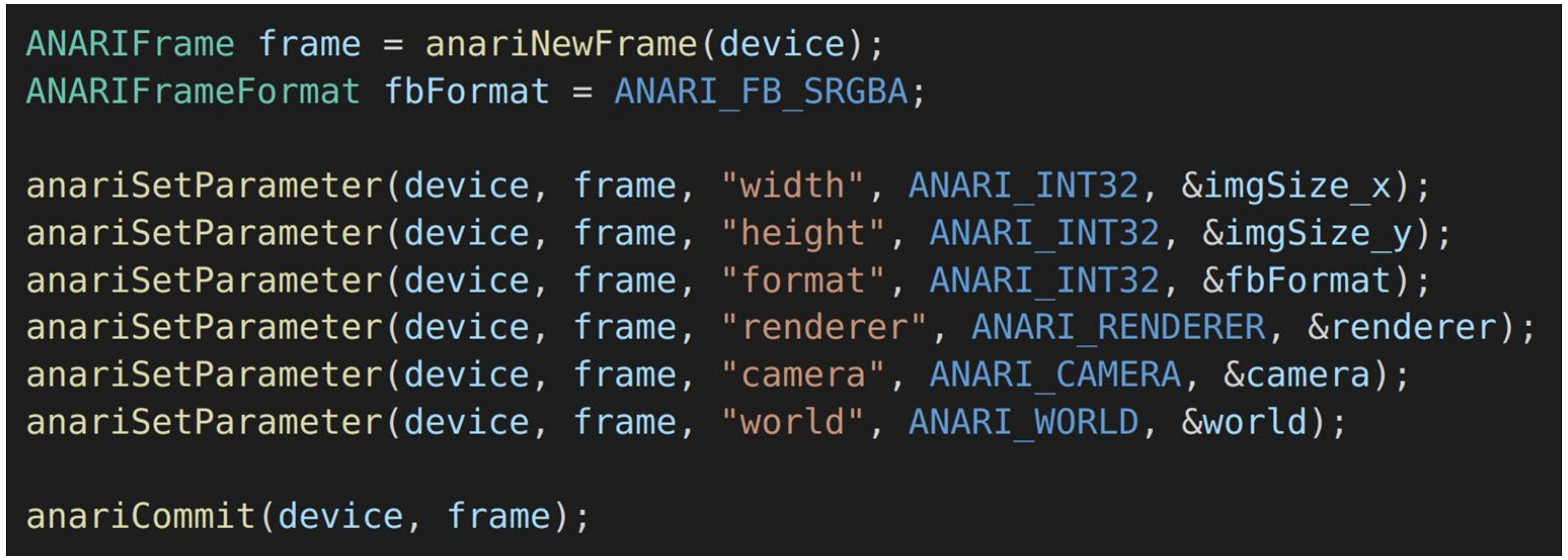


```
ANARIFrame frame = anariNewFrame(device);
ANARIFrameFormat fbFormat = ANARI_FB_SRGBA;

anariSetParameter(device, frame, "width", ANARI_INT32, &imgSize_x);
anariSetParameter(device, frame, "height", ANARI_INT32, &imgSize_y);
anariSetParameter(device, frame, "format", ANARI_INT32, &fbFormat);
anariSetParameter(device, frame, "renderer", ANARI_RENDERER, &renderer);
anariSetParameter(device, frame, "camera", ANARI_CAMERA, &camera);
anariSetParameter(device, frame, "world", ANARI_WORLD, &world);

anariCommit(device, frame);
```

OBJECT OVERVIEW

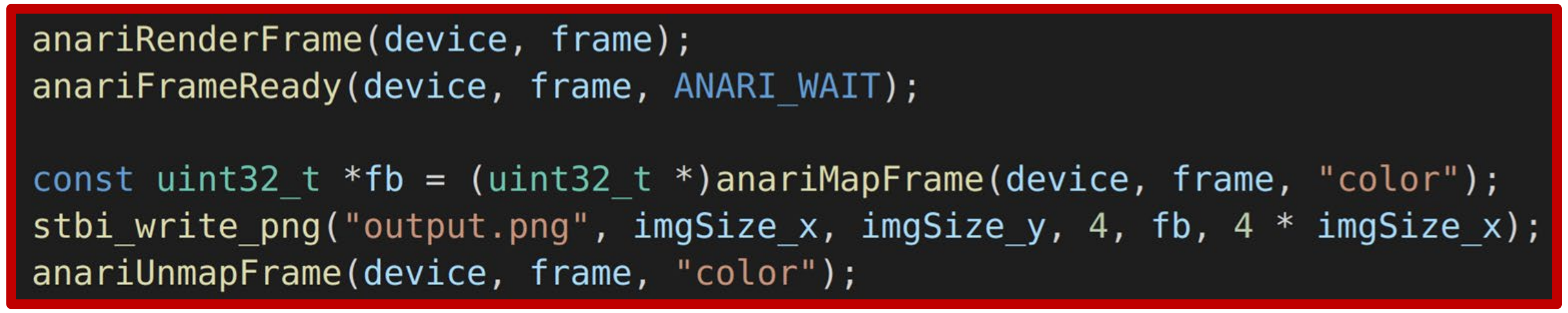
ANARIFrame

ANARIFrame represents the top-level object in the object hierarchy



```
ANARIFrame frame = anariNewFrame(device);
ANARIFrameFormat fbFormat = ANARI_FB_SRGBA;

anariSetParameter(device, frame, "width", ANARI_INT32, &imgSize_x);
anariSetParameter(device, frame, "height", ANARI_INT32, &imgSize_y);
anariSetParameter(device, frame, "format", ANARI_INT32, &fbFormat);
anariSetParameter(device, frame, "renderer", ANARI_RENDERER, &renderer);
anariSetParameter(device, frame, "camera", ANARI_CAMERA, &camera);
anariSetParameter(device, frame, "world", ANARI_WORLD, &world);

anariCommit(device, frame);
```

Frames are rendered asynchronously

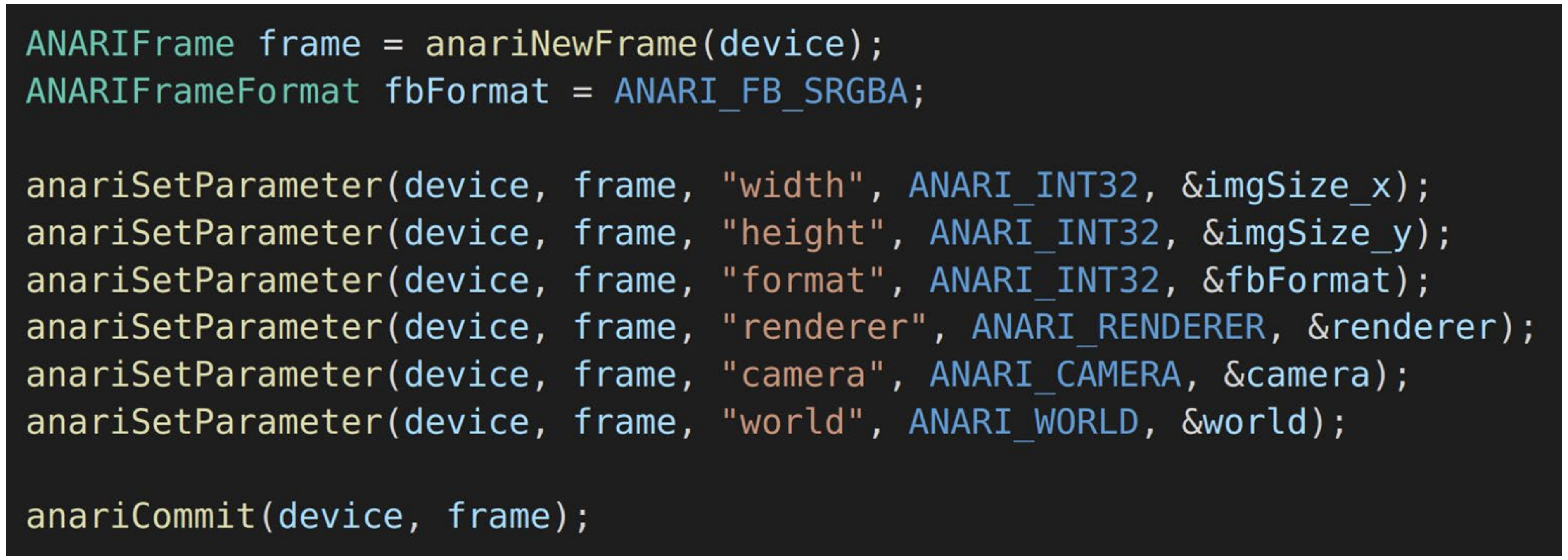


```
anariRenderFrame(device, frame);
anariFrameReady(device, frame, ANARI_WAIT);

const uint32_t *fb = (uint32_t *)anariMapFrame(device, frame, "color");
stbi_write_png("output.png", imgSize_x, imgSize_y, 4, fb, 4 * imgSize_x);
anariUnmapFrame(device, frame, "color");
```

OBJECT OVERVIEW

ANARIFrame

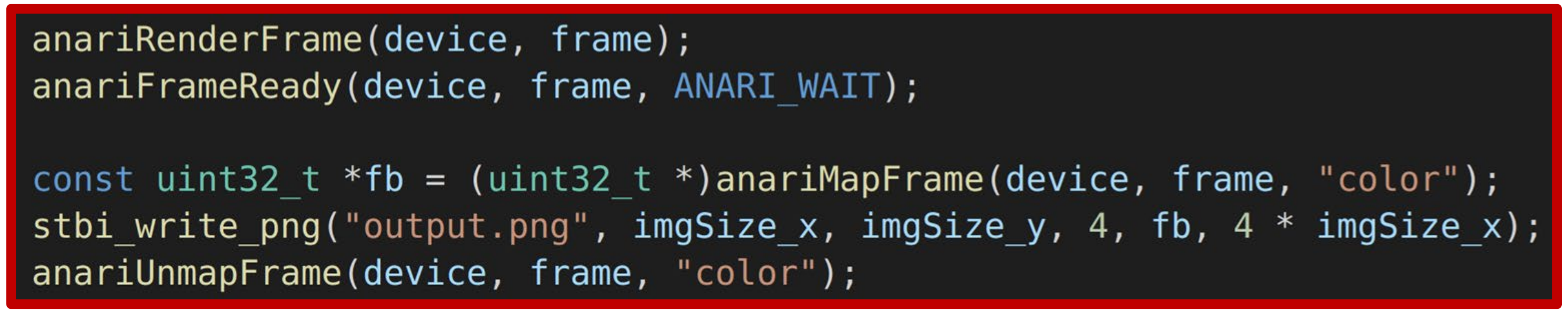
ANARIFrame represents the top-level object in the object hierarchy



```
ANARIFrame frame = anariNewFrame(device);
ANARIFrameFormat fbFormat = ANARI_FB_SRGBA;

anariSetParameter(device, frame, "width", ANARI_INT32, &imgSize_x);
anariSetParameter(device, frame, "height", ANARI_INT32, &imgSize_y);
anariSetParameter(device, frame, "format", ANARI_INT32, &fbFormat);
anariSetParameter(device, frame, "renderer", ANARI_RENDERER, &renderer);
anariSetParameter(device, frame, "camera", ANARI_CAMERA, &camera);
anariSetParameter(device, frame, "world", ANARI_WORLD, &world);

anariCommit(device, frame);
```

Frames are rendered asynchronously

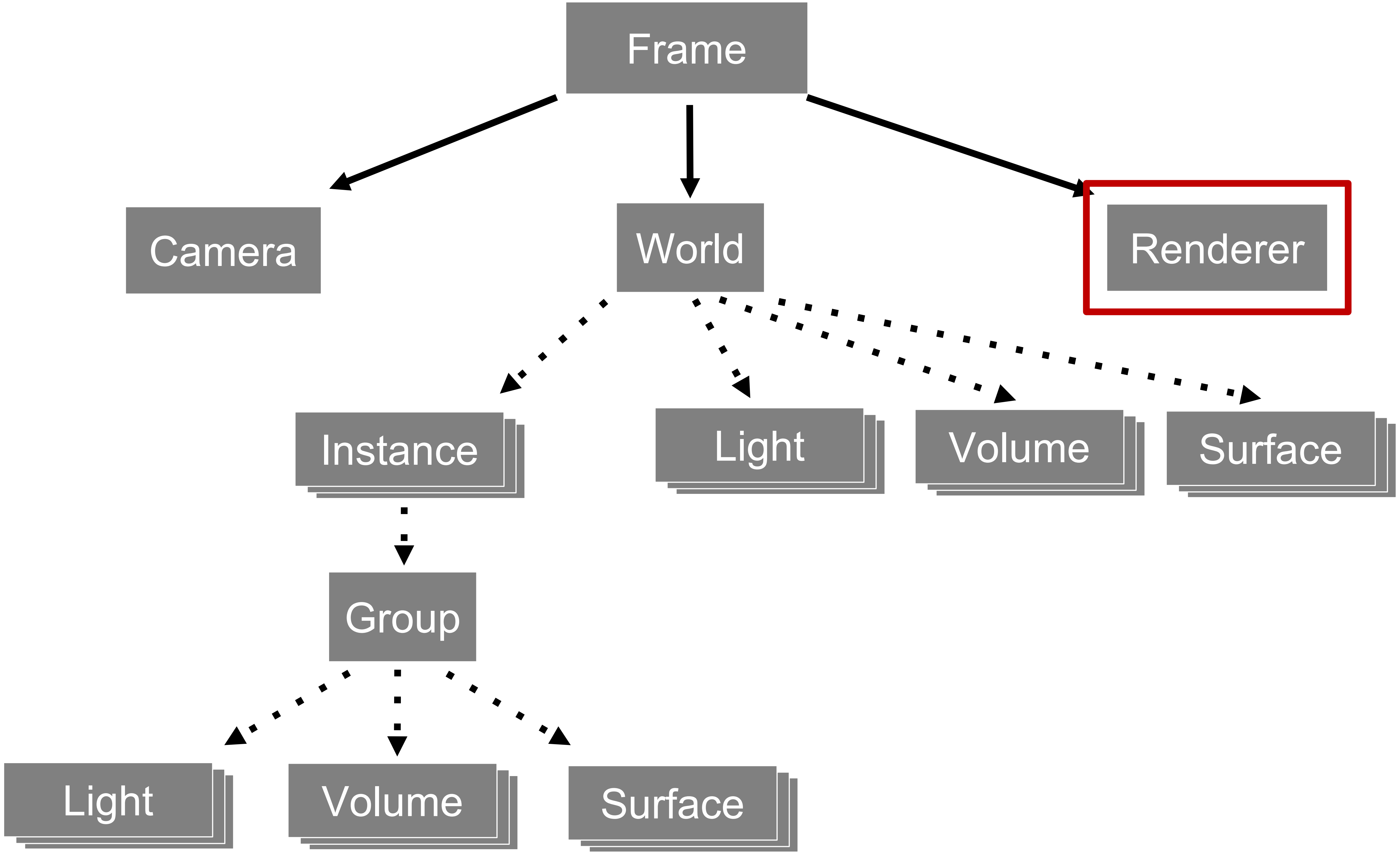
Frames hold frame buffer results formatted according to parameters



```
anariRenderFrame(device, frame);
anariFrameReady(device, frame, ANARI_WAIT);

const uint32_t *fb = (uint32_t *)anariMapFrame(device, frame, "color");
stbi_write_png("output.png", imgSize_x, imgSize_y, 4, fb, 4 * imgSize_x);
anariUnmapFrame(device, frame, "color");
```

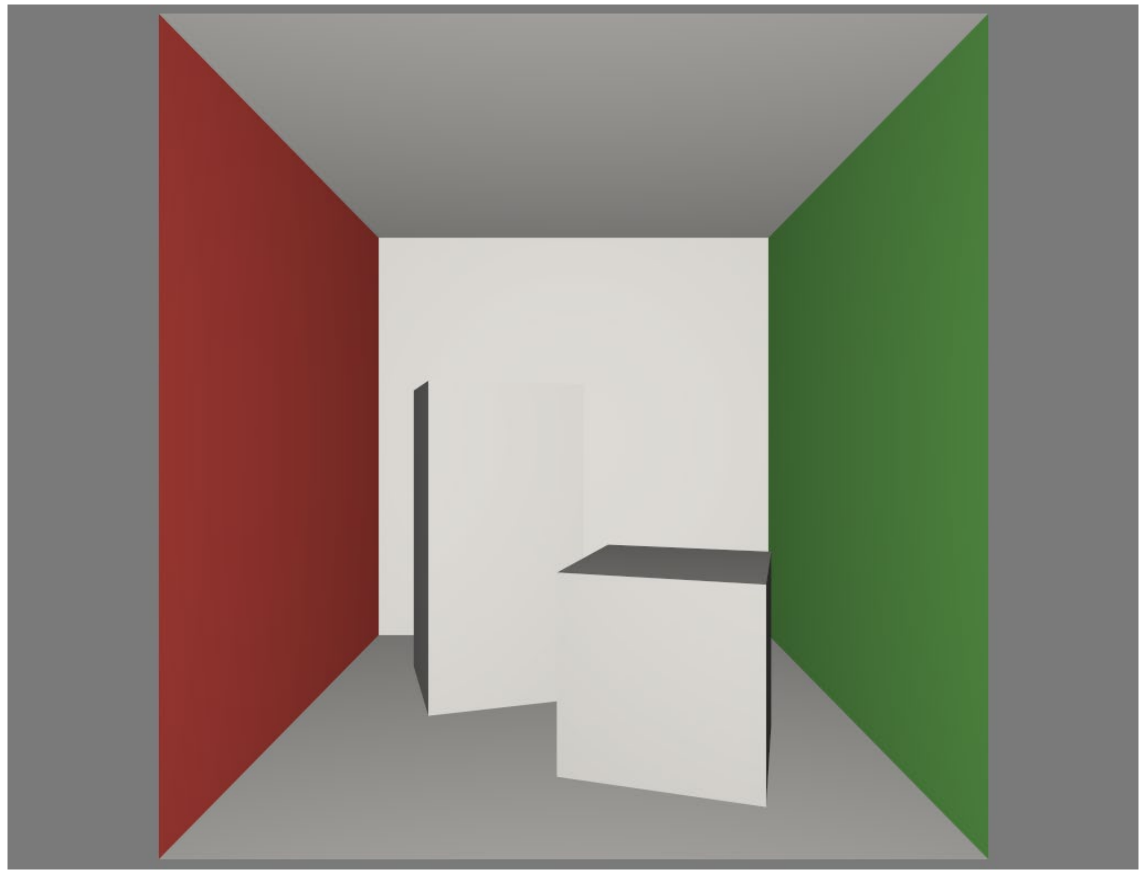
OBJECT OVERVIEW

Object hierarchy

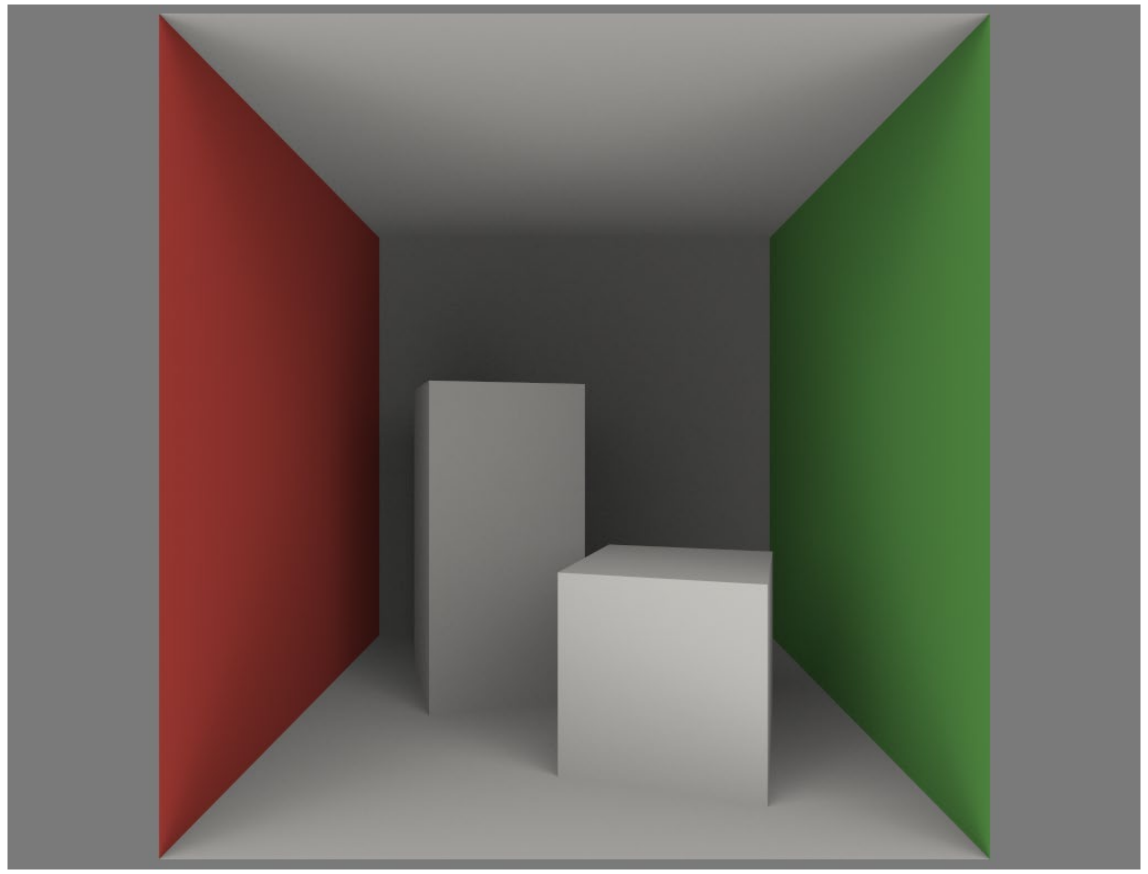


OBJECT OVERVIEW

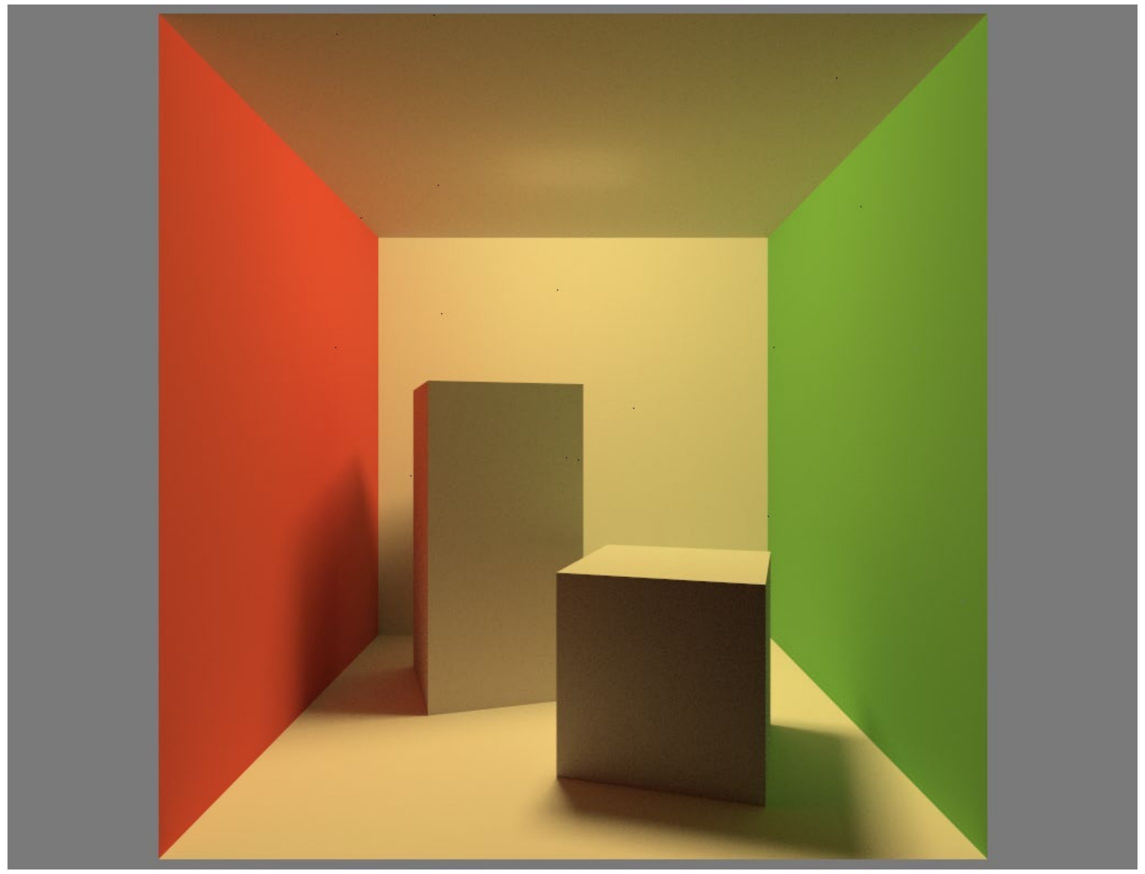
ANARIRenderer



"raycast"



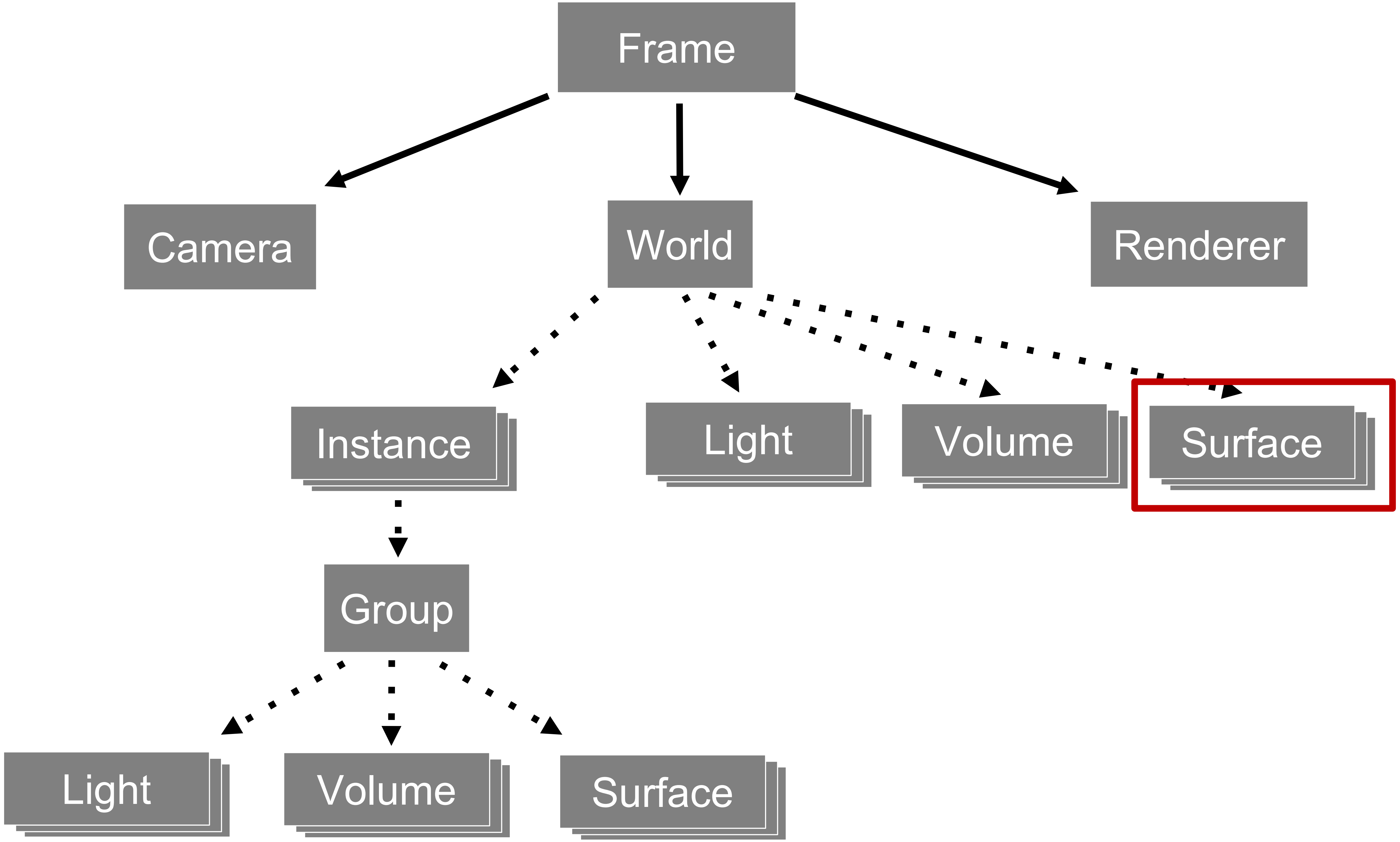
"ao"



"pathtracer"

OBJECT OVERVIEW

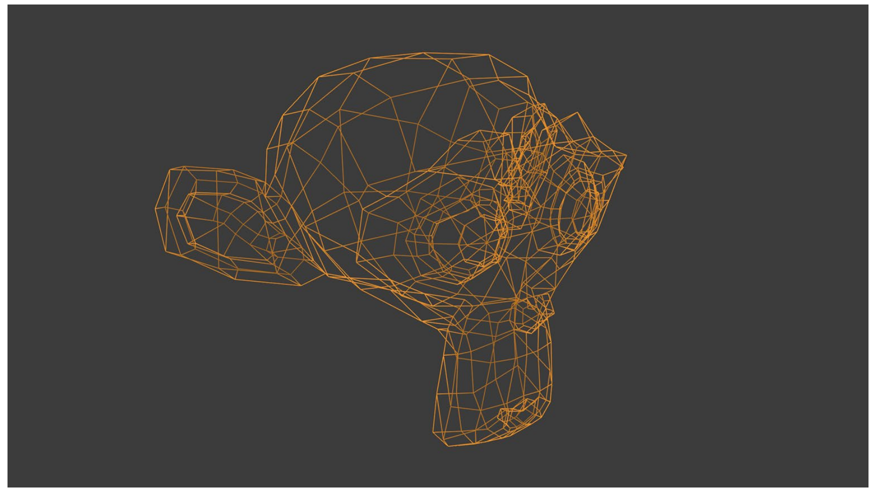
Object hierarchy



OBJECT OVERVIEW

ANARISurface

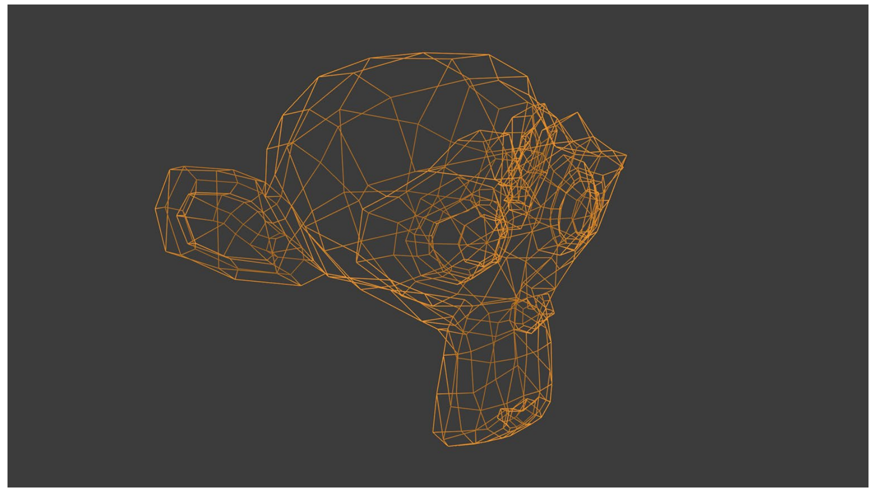
Geometry

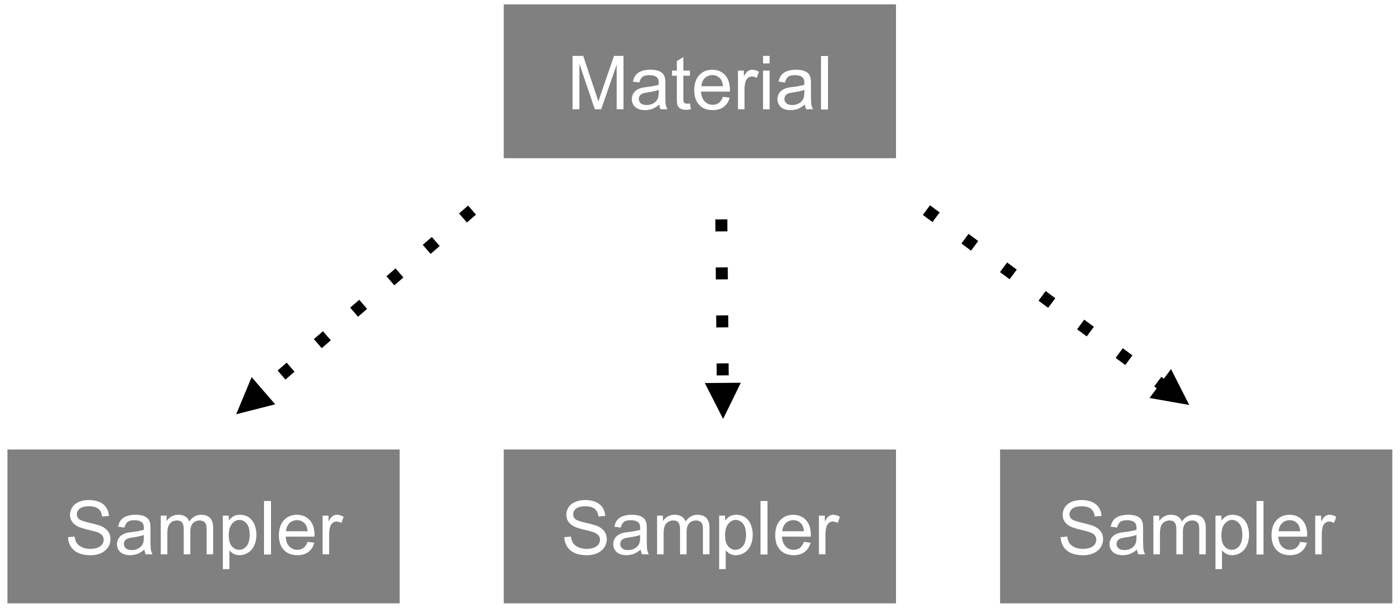


OBJECT OVERVIEW

ANARISurface

Geometry





Material

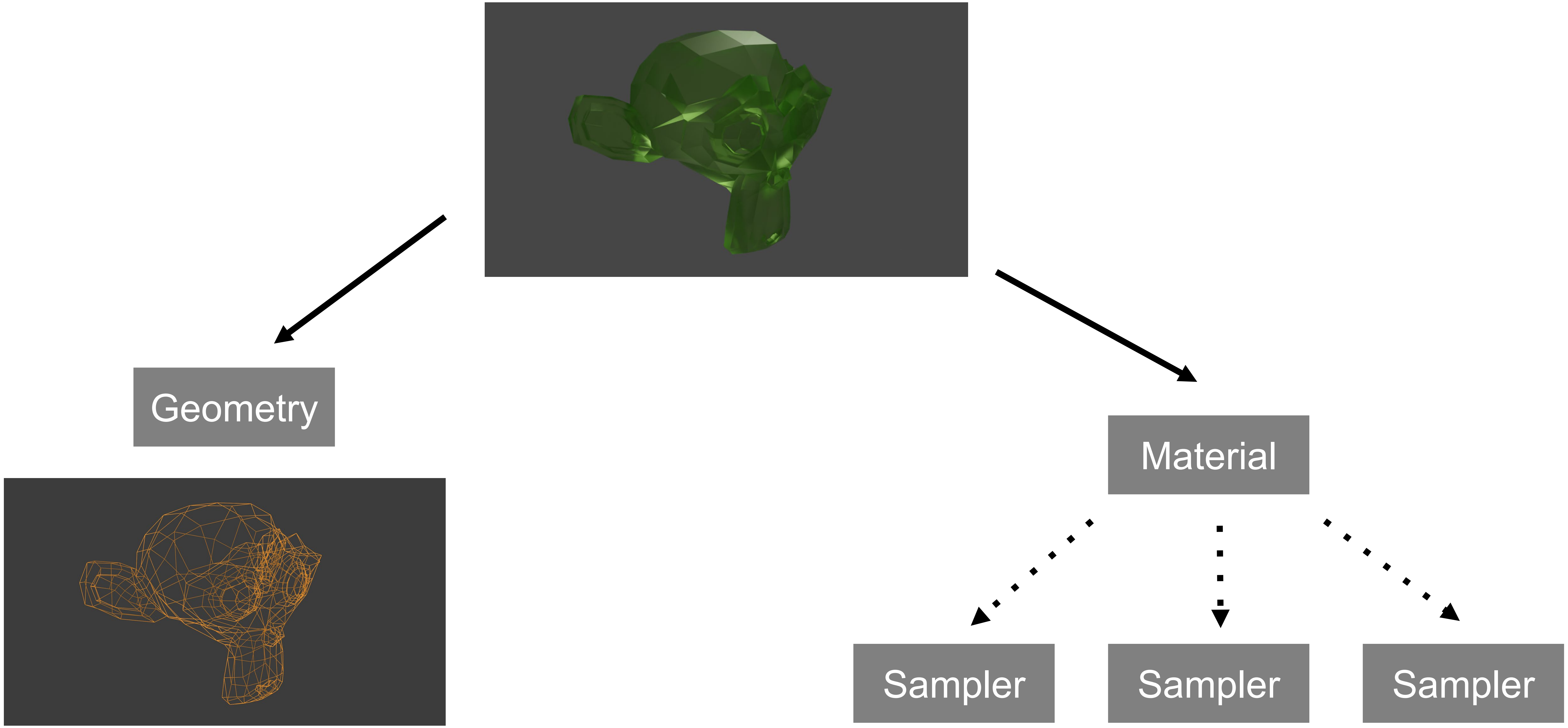
Sampler

Sampler

Sampler

OBJECT OVERVIEW

ANARISurface



Geometry

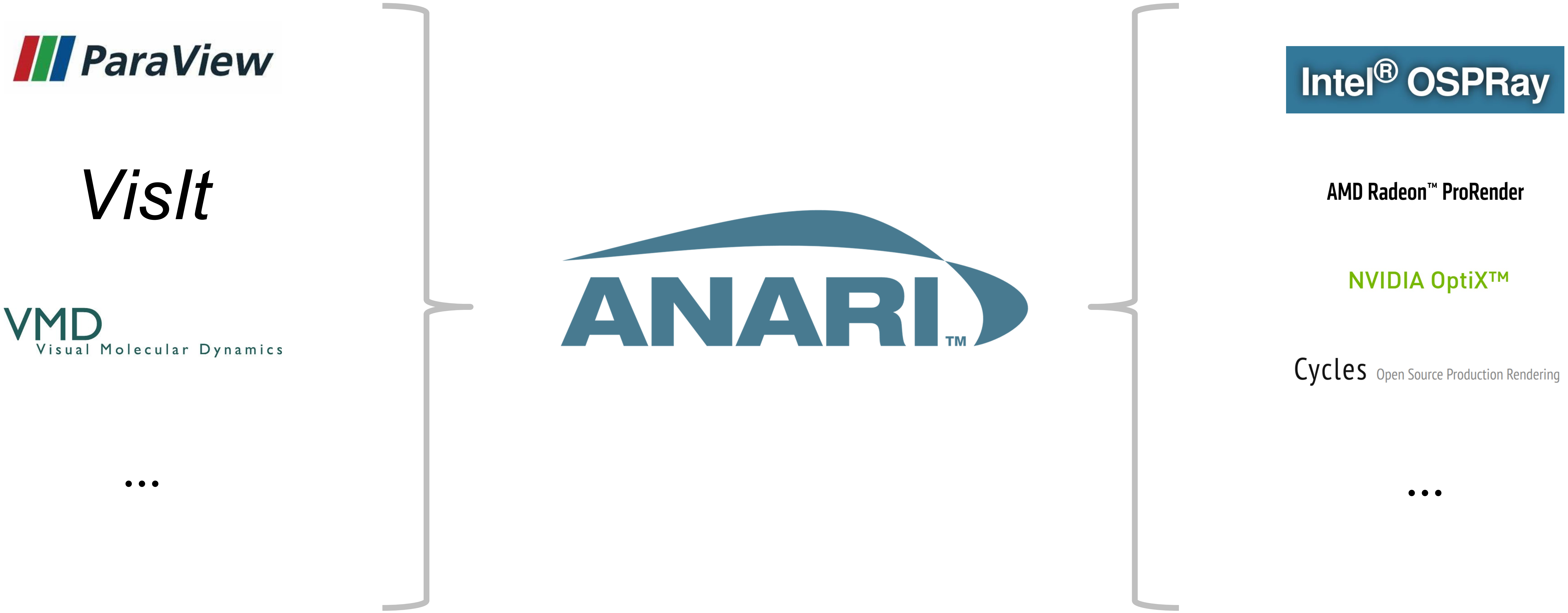
Material

Sampler

Sampler

Sampler

Wrap Up



ParaView

VisIt

VMD Visual Molecular Dynamics

...

ANARI™

Intel® OSPRay

AMD Radeon™ ProRender

NVIDIA OptiX™

Cycles Open Source Production Rendering

...