Standardizing All the Realities: A Look at OpenXR

Kaye Mason

SIGGRAPH, August 2018

A Note on What We'll Cover

A Note on What We'll Cover

- **An Overview of the Spec**
  - **Proviso:** The OpenXR specification is a work in progress.
    Things **will** change between now and the release of the specification
    - Jan 2017 (52pp) :: March 2018 (268pp) :: Aug 2018 (291pp)

    The spec has bugs. Known and Unknown.
  - Talk assumes that you know:
    - Something about AR & VR
    - Nothing about the Specification Process.
  - This talk will not cover the whole spec!
- **Live Demos of OpenXR-backed VR Systems (Starbreeze and Microsoft)**
- **Ample time will be given for Questions at the end!**
  - The spec is long…there may be some questions we can't answer.
  - I can't answer questions about systems that aren't stabilized.
  - I can't tell you when the spec will be released.

A Brief History of the Standard

AMD
antilatency
arm
COLLABORA
EPIC GAMES
Google
HTC

HUAWEI
Imagination
intel
LG
logitech
LUNAR G
magic leap

MEDIATEK
Microsoft
moz://a
NOKIA
NVIDIA
oculus
PICO

pluto
QUALCOMM
RAZER
SAMSUNG
Sony Interactive Entertainment
STARBREEZE STUDIOS
tobii

unity
VALVE
VARJO
VeriSilicon
兆芯
zSpace

KHRONOS GROUP

A Brief History of the Standard

Call for Participation / Exploratory Group Formation -- *Fall F2F, October 2016: Korea*

Statement of Work / Working Group Formation -- *Winter F2F, January 2017: Vancouver*

Specification Work -- *Spring F2F, April 2017: Amsterdam*

Specification Work -- *Interim F2F, July 2017: Seattle*

Defining the MVP -- *Fall F2F, September 2017: Chicago*

Resolving Implementation Blockers -- *Winterim F2F, November 2017: Seattle*

Raising Implementation Issues -- *Winter F2F, January 2018: Taipei*

First Public Information! -- *GDC, March 2018: San Francisco*

Implementation and Refinement -- *Spring F2F, April 2018: Montreal*

Present Day

Updates & First Demonstration! -- *SIGGRAPH, August 2018: Right Here, Right Now!*

Coming Soon

Implementation, Conformance and Refinement -- *Fall F2F, September 2018*

Provisional Release

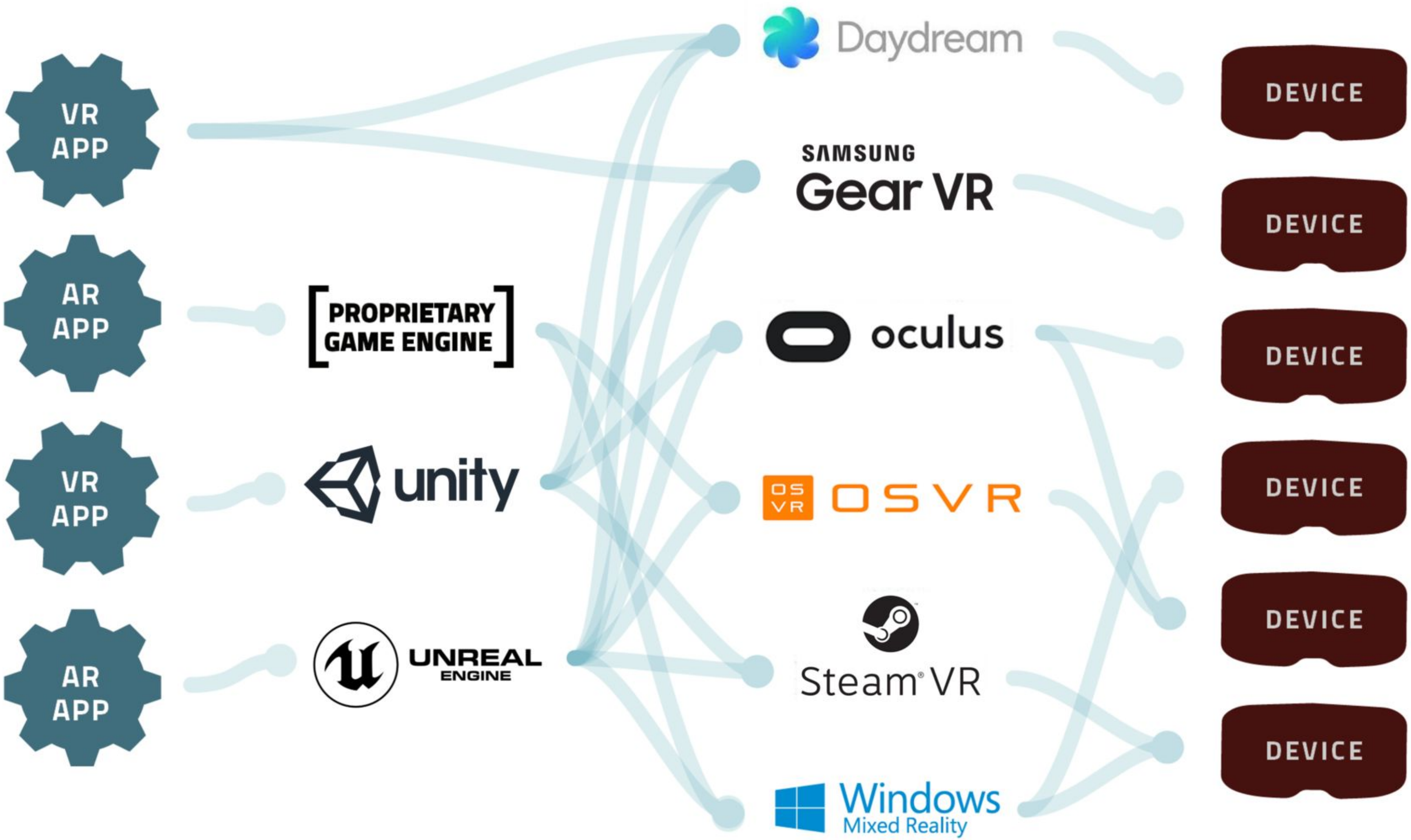
Conformance Testing and Implementation

Ratification and Release

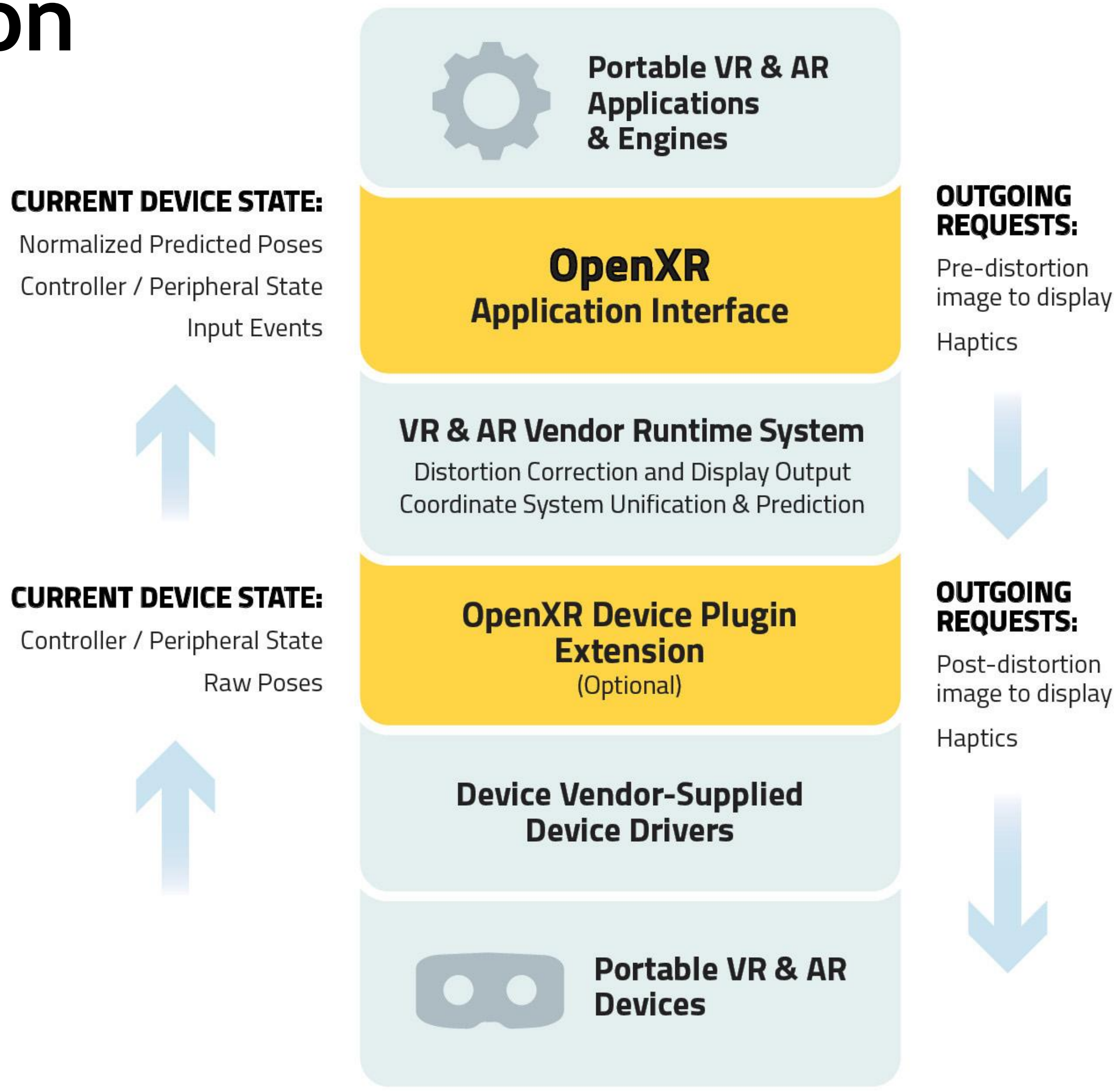
KHRONOS™ GROUP

Goals and Philosophies

The Problem


VR APP
AR APP
VR APP
AR APP
PROPRIETARY GAME ENGINE
unity
UNREAL ENGINE
Daydream
SAMSUNG Gear VR
oculus
OS VR OSVR
Steam VR
Windows Mixed Reality
DEVICE
DEVICE
DEVICE
DEVICE
DEVICE
DEVICE

The Solution



OpenXR Philosophies

1 **Enable both VR and AR applications**

The OpenXR standard unified common VR and AR functionality to streamline software and hardware development for a wide variety of products and platforms

2 **Be future-proof**

While OpenXR 1.0 is focused on enabling the current state-of-the-art, the standard is built around a flexible architecture and extensibility to support rapid innovation in the software and hardware spaces for years to come

3 **Do not try to predict the future of XR technology**

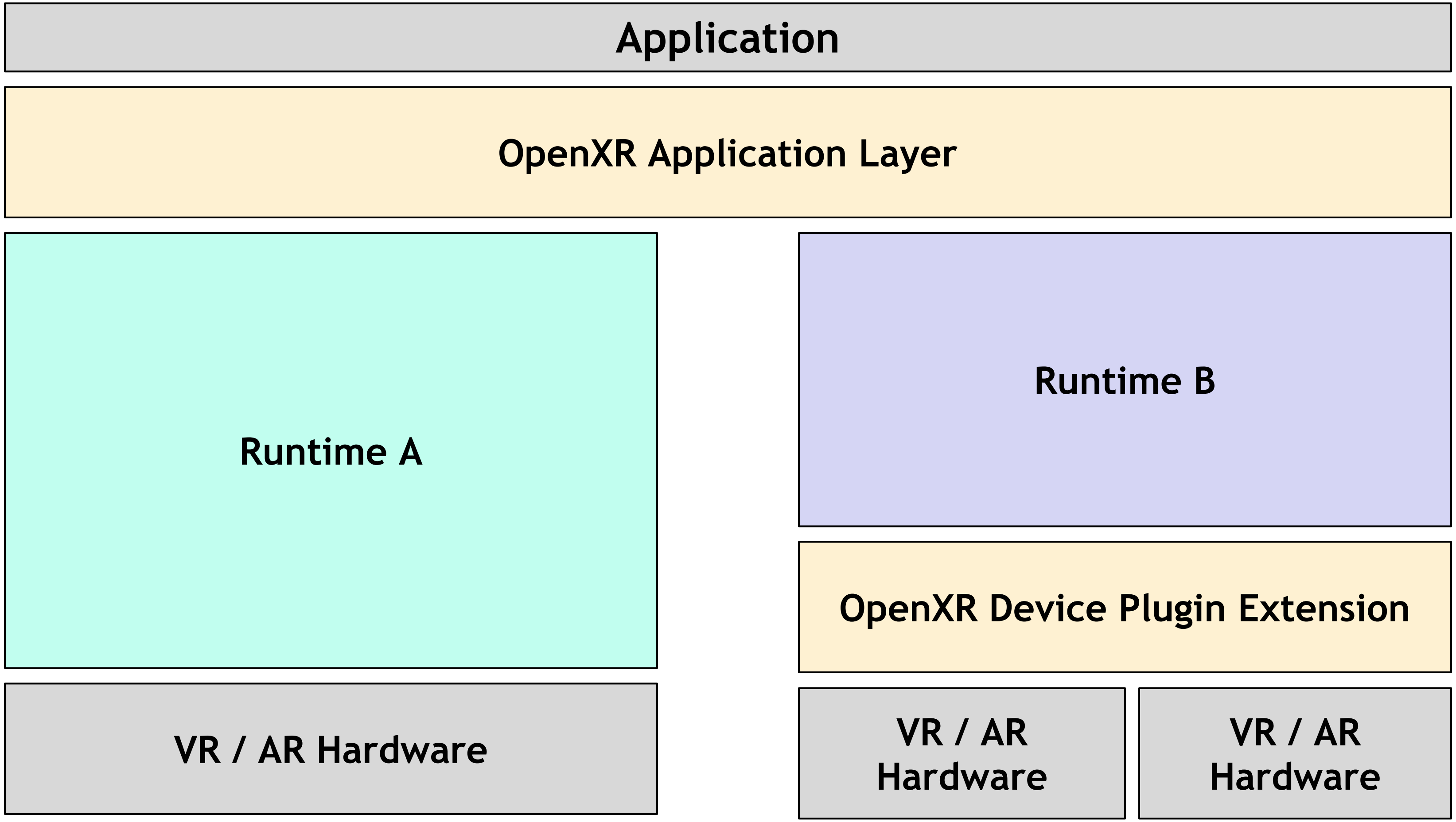
While trying to predict the future details of XR would be foolhardy, OpenXR uses forward-looking API design techniques to enable designers to easily harness new and emerging technologies

4 **Unify performance-critical concepts in XR application development**

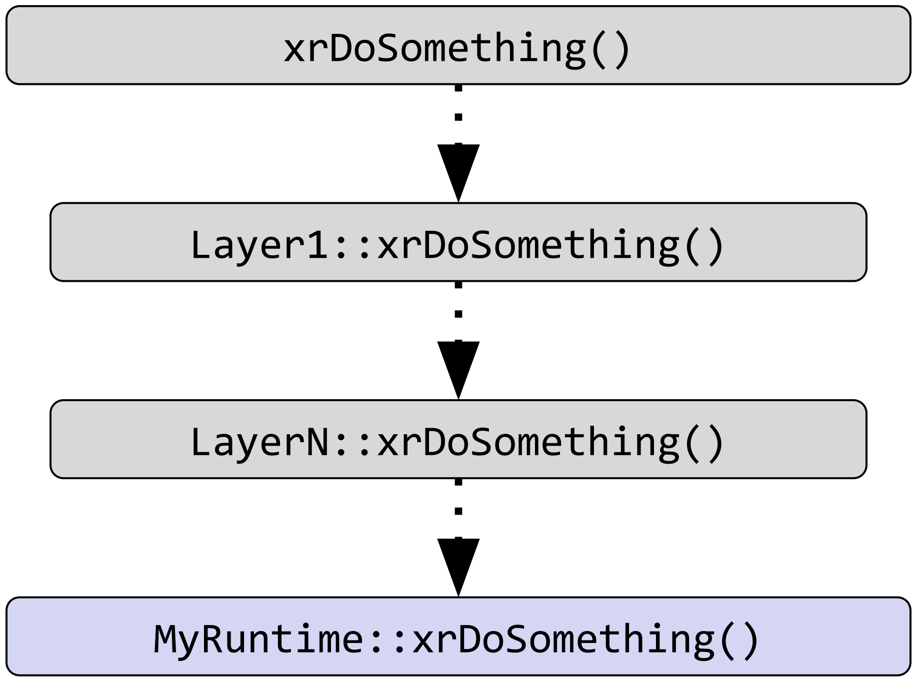
Developers can optimize to a single, predictable, universal target rather than add application complexity to handle a variety of target platforms

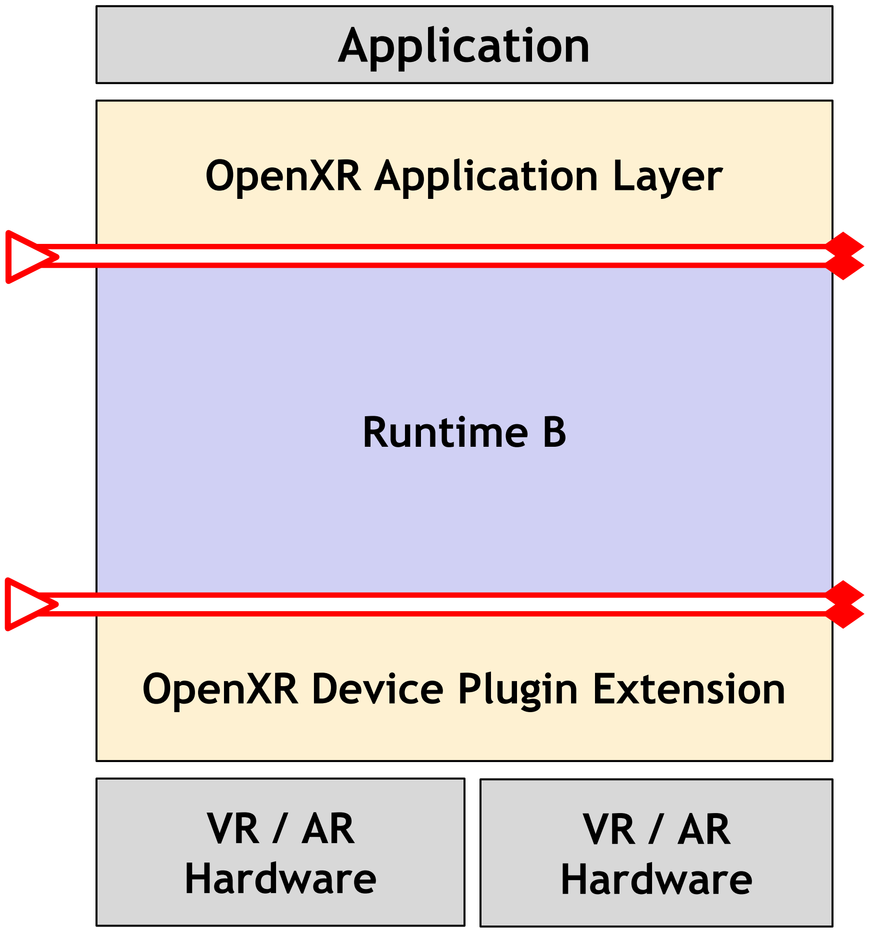
KHRONOS GROUP

The Structure



Layered API


xrDoSomething()
Layer1::xrDoSomething()
LayerN::xrDoSomething()
MyRuntime::xrDoSomething()


Application
OpenXR Application Layer
Runtime B
OpenXR Device Plugin Extension
VR / AR Hardware
VR / AR Hardware

Architecture Overview

API Conventions and Primitives

Handles

Objects which are allocated by the runtime on behalf of the application are represented by handles

Handles are:

- Opaque identifiers to the underlying object
- Lifetime generally managed by `xrCreate*` and `xrDestroy*` functions
- Hierarchical
  - E.g. To create an `XrSession` handle, you must pass in a parent `XrInstance`
  - Handles for children are only valid within their direct parent's scope

API Conventions and Primitives

Semantic Paths

Properties of `XrPaths`:

- Hierarchical
- Stored in a string table
- Human-readable
- Can be pre-defined (reserved) or user-defined
- Handles
- `[a-z,0-9,-,_,.,/]`
- Null terminated
- Not file paths!
  - Can’t use ./ or ../ for pathing

API Conventions and Primitives

Semantic Paths

Some paths are reserved by the specification for special purposes:

```
/user/hand/left, user/hand/right
/user/hand/primary, user/hand/secondary
/user/head
/space/head
/space/hand/left/grip
/devices/<vendor_name>/<unique_identifier>
/devices/<vendor_name>/<unique_identifier>/<type>/<component>
```

where <type> is: `thumbstick`, `trigger`, `system`, etc.

and <component> is: `click`, `touch`, `value`, `delta_x`, etc.

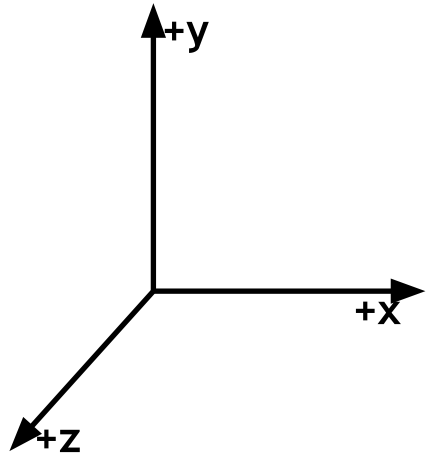
KHRONOS GROUP

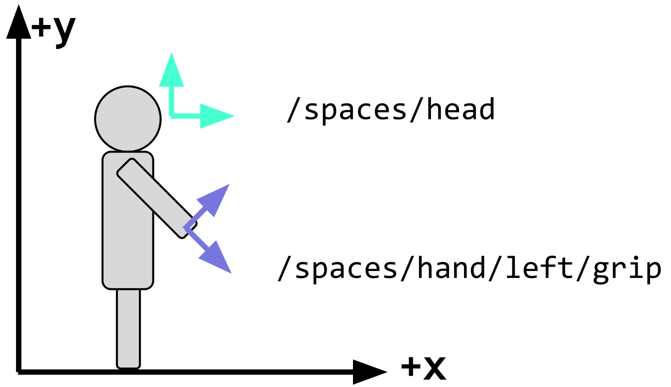
API Conventions and Primitives

XrSpace

XrSpace is one of the fundamental concepts used throughout the API to help with making a generalized understanding of the physical tracking environment.

- The Runtime can hold any representation it wants internally.
- XrSpaces are independent coordinate systems tracked by the runtime, which can be related to one another, and used as a basis for functions that return spatial values
- In certain cases, such as motion controllers, XrSpaces can be attached to tracked objects for ease of reference

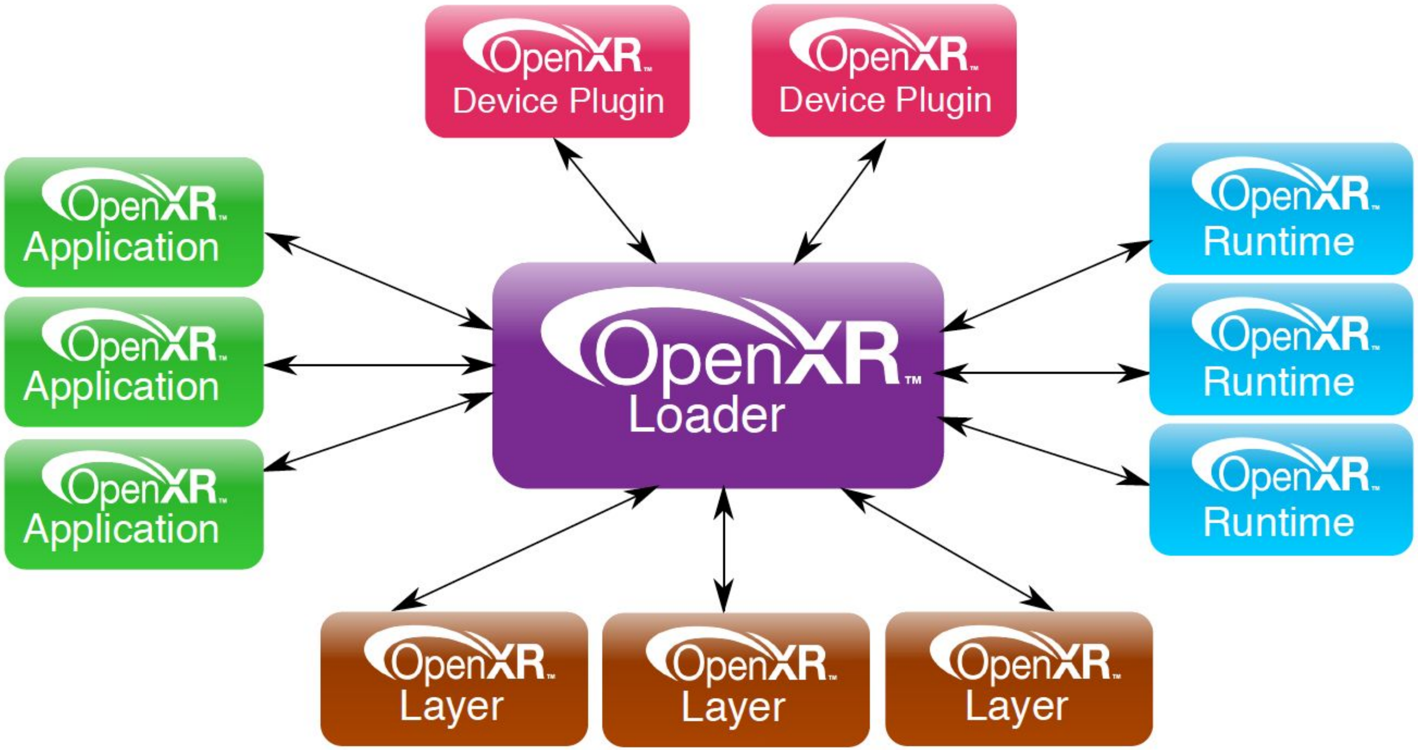




The Loader, Extensions and Layers

Loader:

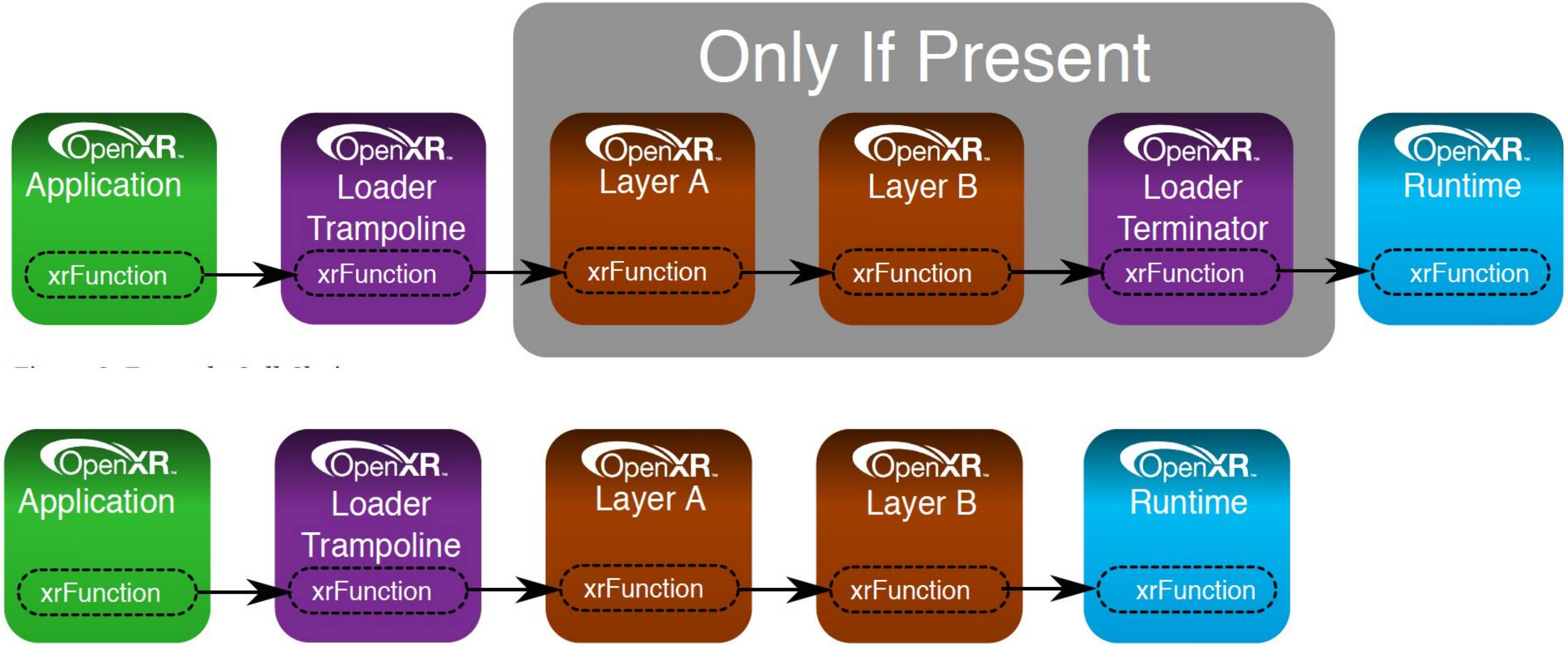
- Not *required*
- Complexity can vary
- Some platforms have strict requirements (i.e., mobile)



The Loader, Extensions and Layers

Loader:

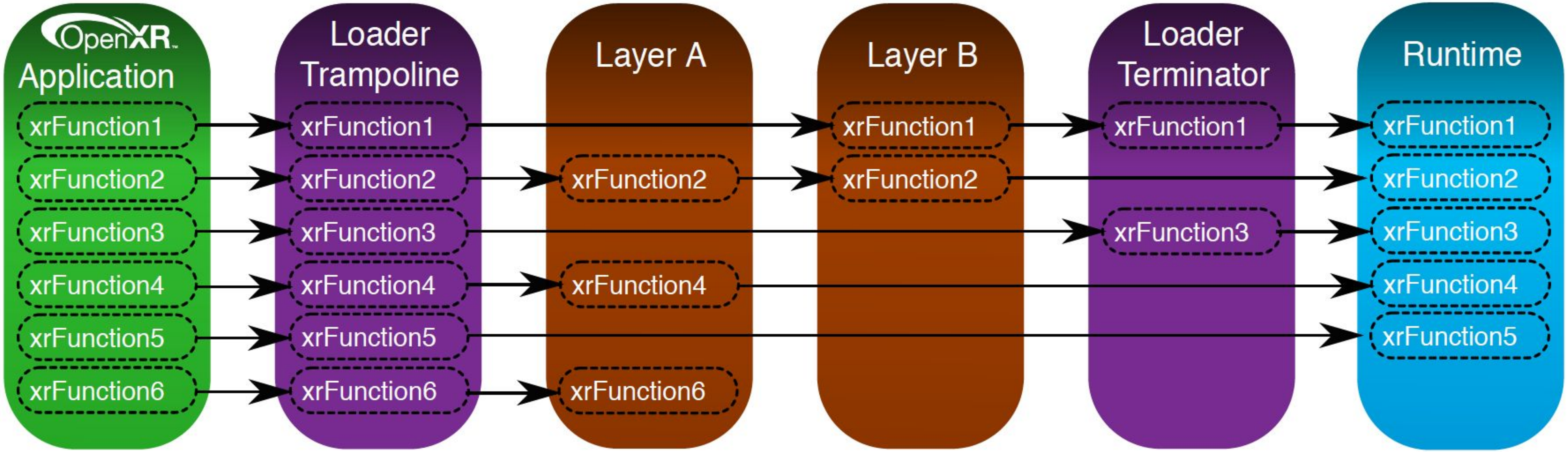
- Loader Trampoline and Terminator Patterns



The Loader, Extensions and Layers

Loader:

- Loader Trampoline and Terminator Patterns



Core and Extensions

Core Standard	Core concepts that are fundamental to the specification for all use cases *Examples: Instance management, tracking*
KHR Extensions	Functionality that a large classes of runtimes will likely implement *Examples: Platform support, Device Plugin, Headless, Tracking Bounds, Vulkan Extensions*
EXT Extensions	Functionality that a few runtimes might implement *Examples: Performance Settings, Thermals, Debug Utils*
Vendor Extensions	Functionality that is limited to a specific vendor *Examples: Device specific functionality*

Layers

We already saw how layers work with the loader. Some possible example layers:

Validation	Push detailed validation of API usage into a layer so it can be turned off in production.
Platform App Quality	Yes, OpenXR allows you to do that, but on our platform, it's not smart. *Examples: Specialized hardware.*
Debug Panels	Capture information and display it. *Examples: Frame rate, frame drops, latency*

Application Lifecycles

- Operating Systems can have very different application lifecycles.
- Here are two examples cases:

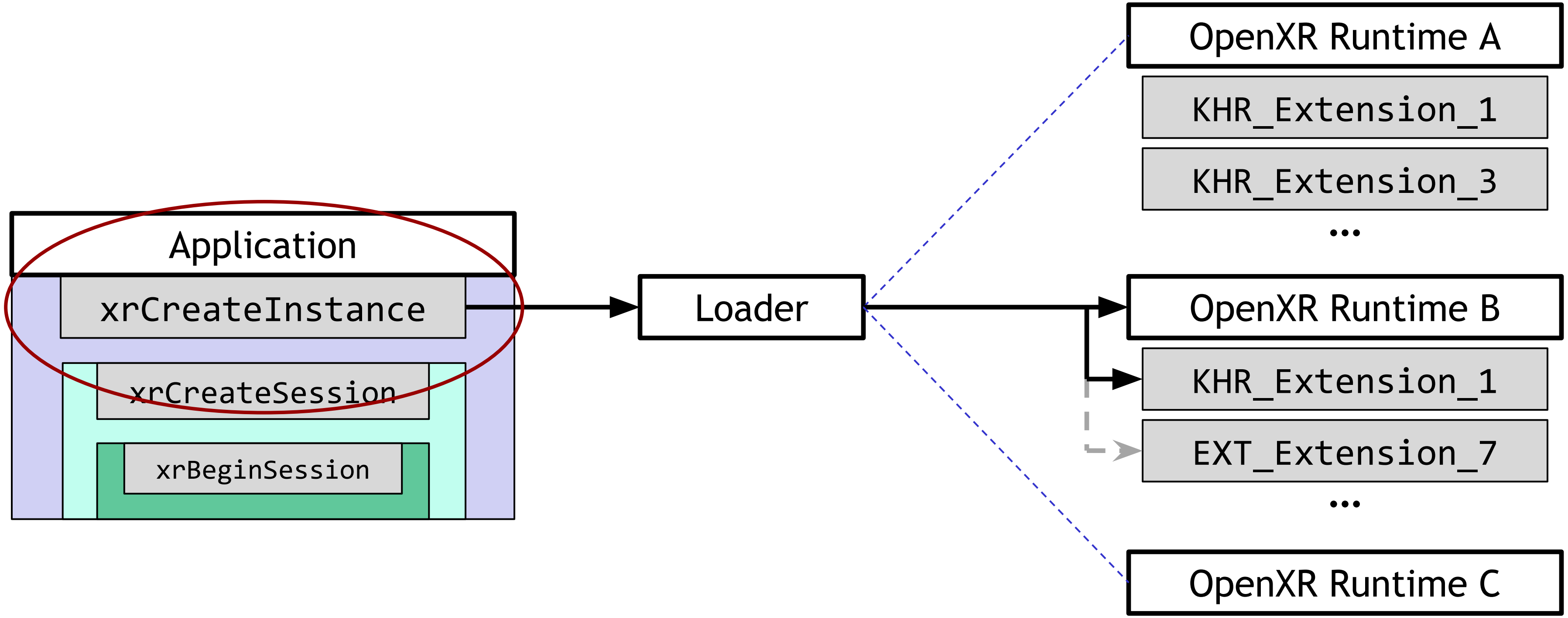
Android
(Launch)
OnCreate
OnStart
OnResume
(Running)
OnPause
OnStop
(Shut Down)

Windows 10
Not Running
(Activated)
Running In Foreground
(Leaving Foreground)
Running in Background
(Entering Foreground)
(Suspending)
Suspended
(Resuming)

Lifecycles: the Instance and the Session

XrInstance:

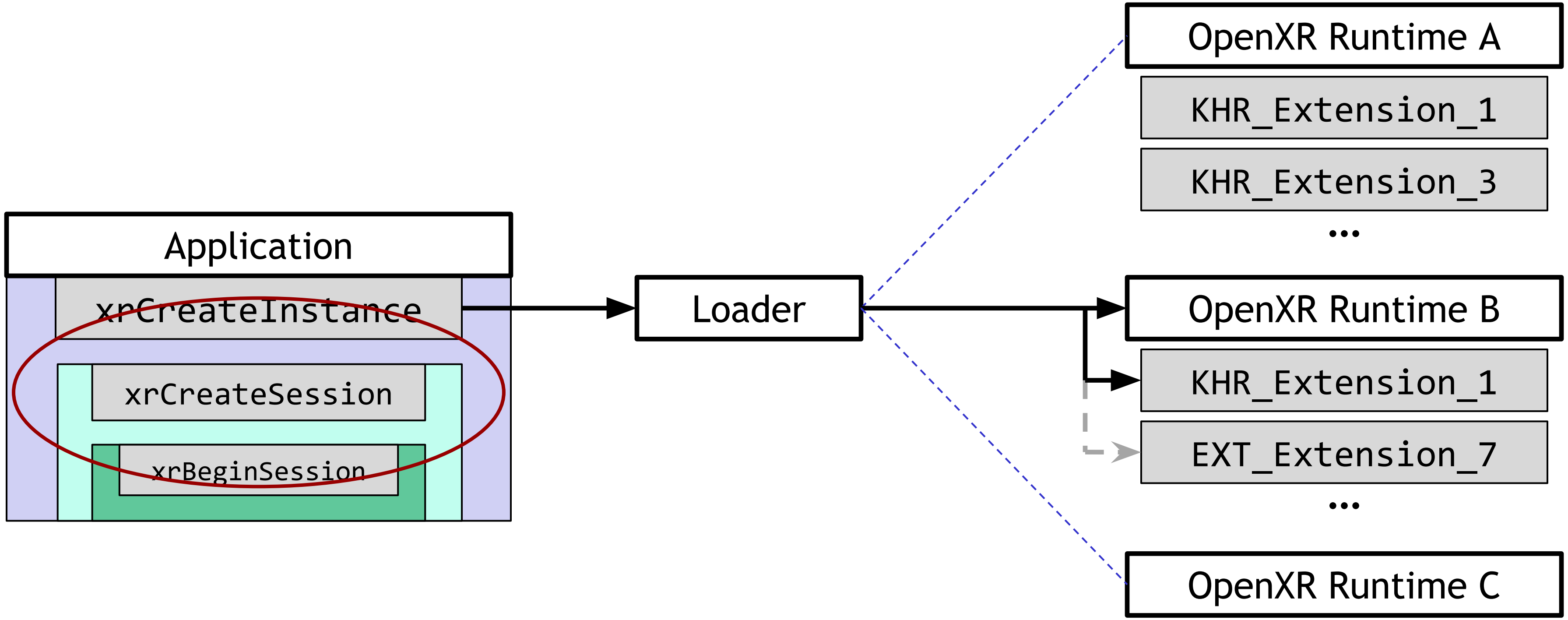
- The XrInstance is basically the application's representation of the OpenXR runtime
- Can create multiple XrInstances, if supported by the runtime
- xrCreateInstance specifies app info, layers, and extensions for the instance.



Lifecycles: the Instance and the Session

XrSession:

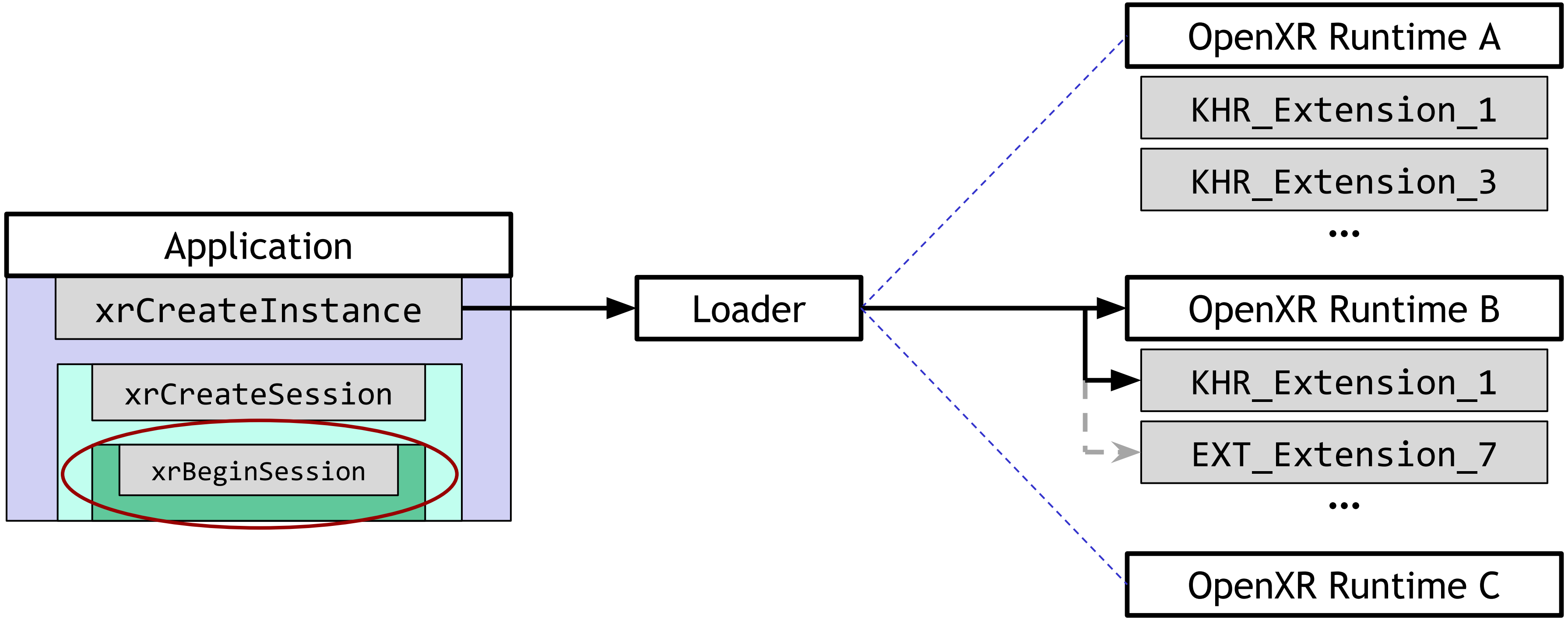
- `XrSession` represents an active interaction between the application and the runtime.
- `XrSession` can now have its own extensions.
- Swapchain management is done here.



Lifecycles: the Instance and the Session

`XrSession`:

- Beginning an `XrSession` is how an application indicates it wants to render stuff.
- Applications use this to tell the runtime what to render and how.



Events

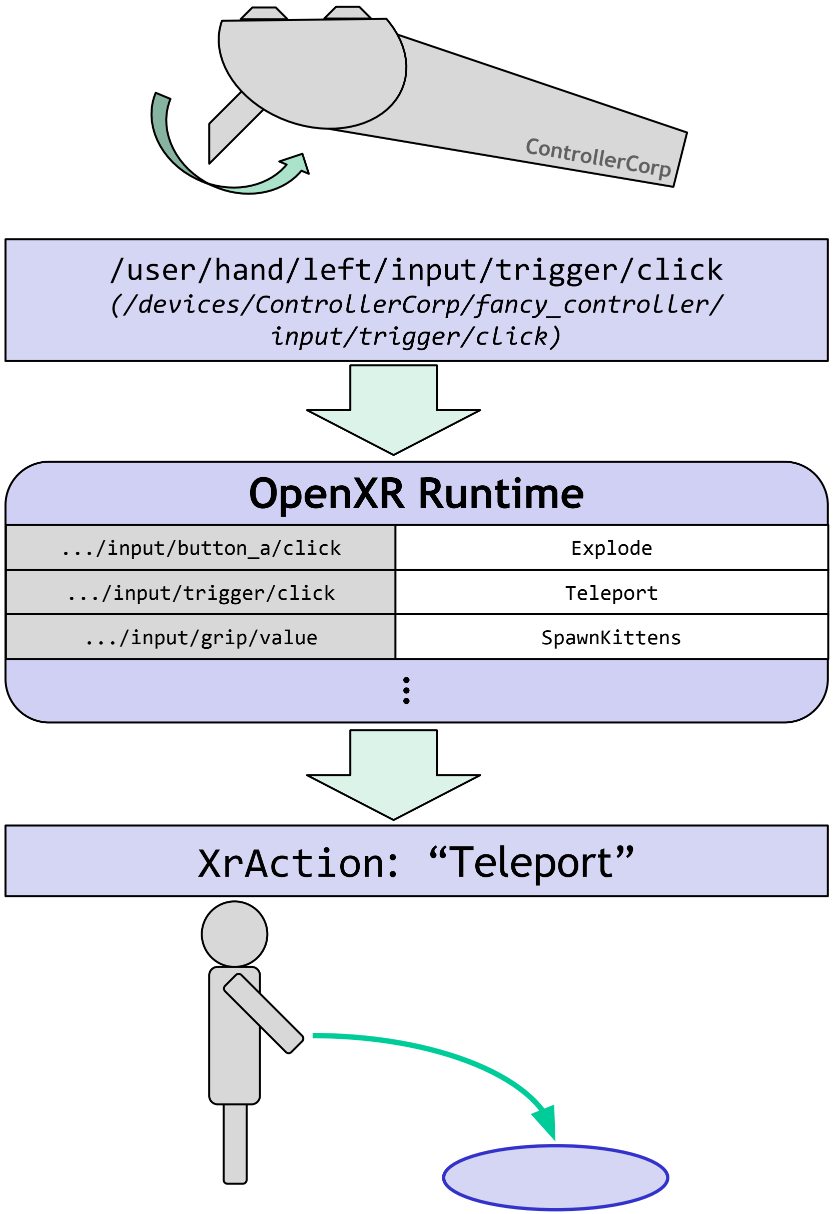
Events are messages sent from the runtime to the application. They're put into a queue by the runtime, and read from that queue by the application by `xrPollEvent`

Visibility Changed	Whether or not the application is visible on the device
Focus Changed	Whether or not the application is receiving input from the system
Request End Session	Runtime wants the application to exit
Request End Instance	Call `xrDestroyInstance`, because the runtime needs to update
Availability Changed	Device attached or lost
Engagement Changed	Device is put on or taken off

Input and Haptics

Input in OpenXR goes through a layer of abstraction built around Input Actions (`XrActions`). These allow application developers to define input based on resulting action (*e.g. "Move," "Jump," "Teleport"*) rather than explicitly binding controls

While the application can suggest recommended bindings, it is ultimately up to the runtime to bind input sources to actions as it sees fit (application's recommendation, user settings, etc.)

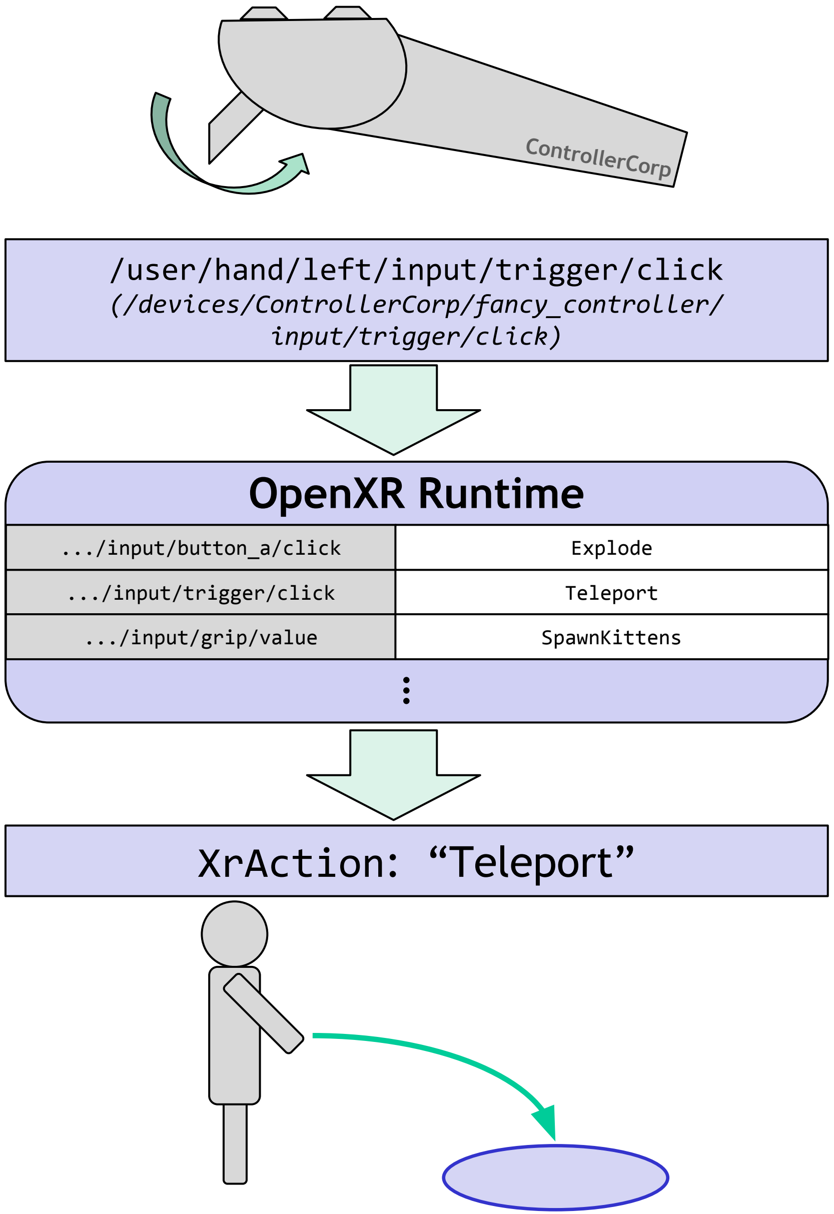


Input and Haptics

Forcing applications through this indirection has several advantages:

- Greater future-proofing as improvements to hardware and runtimes come out

  "Dev teams are ephemeral, platforms are forever"
- Allows for runtimes to "mix-and-match" multiple input sources
- Easy optional feature support (e.g. body tracking)
- Allows hardware manufacturers a pool of existing content to use with their new devices



Input and Haptics

`XrActions` are created with the following information:

- **Action Name:** A name to reference the action by (e.g. "Teleport")
- **Localized Name:** A human-readable description of the action, localized to the system's current locale
- **Action Set:** The logical grouping of actions this action belongs to (NULL for global)
- **Suggested Binding:** Optional, but suggests which bindings for known devices the application developer recommends
- **Action Type:**

Suggested Binding Restrictions

XR_INPUT_ACTION_TYPE_BOOLEAN	If path is a scalar value, a threshold must be applied. If not a value, needs to be bound to …/click
XR_INPUT_ACTION_TYPE_VECTOR1F	If path is a scalar value, then input is directly bound. If the bound value is boolean, the runtime must supply a 0.0 or 1.0 as the conversion
XR_INPUT_ACTION_TYPE_VECTOR2F	Path must refer to parent with child values …/x and …/y
XR_INPUT_ACTION_TYPE_VECTOR3F	Path must refer to parent with child values …/x, …/y, and …/z

Input and Haptics

There is another type of `XrInputAction`, `XR_TYPE_ACTION_STATE_POSE`, which allows for adding new tracked devices into the scene

`xrGetActionStatePose` allows the application to get the following information in the specified `XrSpace`:

- Pose (position and orientation)
- Linear Velocity (m/s^2)
- Angular Velocity
- Linear Acceleration
- Angular Acceleration

For some devices, not all data is available
Validity can be checked using XrTrackerPoseFlags

Input and Haptics

`XrActions` can be grouped into `XrActionSets` to reflect different input modalities within the application

For example, in *Kitten Petter VR*, you might be in kitty petting mode, or in UI mode, and have different input actions for each:

XrActionSet: Kitten_Petting	
.../input/button_a/click	SpawnYarnBall
.../input/trigger/click	Teleport
.../input/grip/value	SpawnKittens
⋮	

XrActionSet: UI_Mode	
.../input/button_a/click	SelectItem
.../input/trigger/click	ChangeMenu
.../input/trackpad/delta_y	ScrollMenu
⋮	

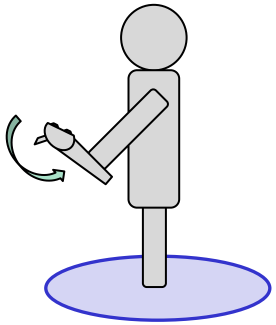
The application can then swap between which `XrActionSet` (or Sets) when it syncs action state in `xrSyncActionData`

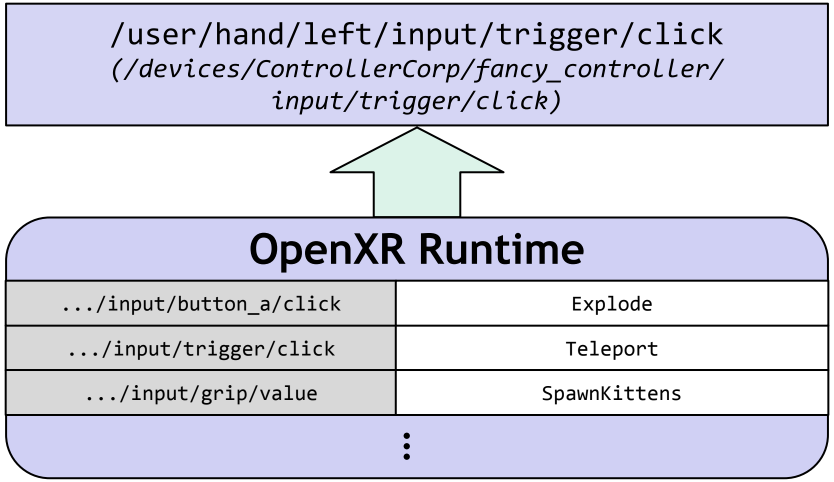
Input and Haptics

We can also flip things, and figure out what device input that a particular `XrAction` is bound to

This is useful for prompts like “Activate the Trigger to Teleport!”

Activate the Trigger to Teleport!





Input and Haptics

Haptics build upon the same `XrAction` system, and have their own Action Type: `XR_HAPTIC_VIBRATION`. Just like other XrActions, they can be used with `XrActionSets`, but unlike inputs, they are activated with `xrApplyHapticFeedback`

Currently, only `XrHapticVibration` is supported:

- Start Time
- Duration (s)
- Frequency (Hz)
- Amplitude (0.0 – 1.0)

We expect that many more haptic types will be added through extensions as the technology develops

Frame Timing

Let's examine frame timing first in the simplest case of a single-threaded render loop

`xrBeginFrame:`

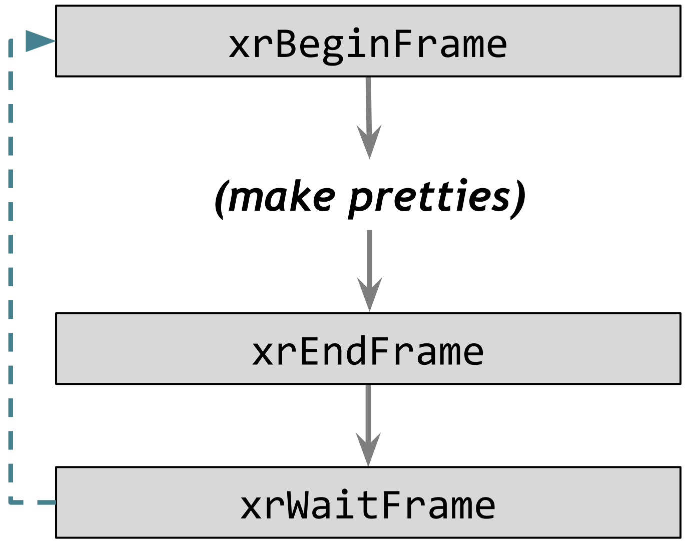
Signals that we're ready to begin rendering pixels to the active image in our swap chain

`xrEndFrame:`

We're finished rendering, and now are ready to hand off the compositor for presentation. Takes a predicted display time, and layers to present

`xrWaitFrame:`

Called before we begin simulation of the next frame. This is responsible for throttling



Frame Timing

Digging into `xrWaitFrame` a bit more…

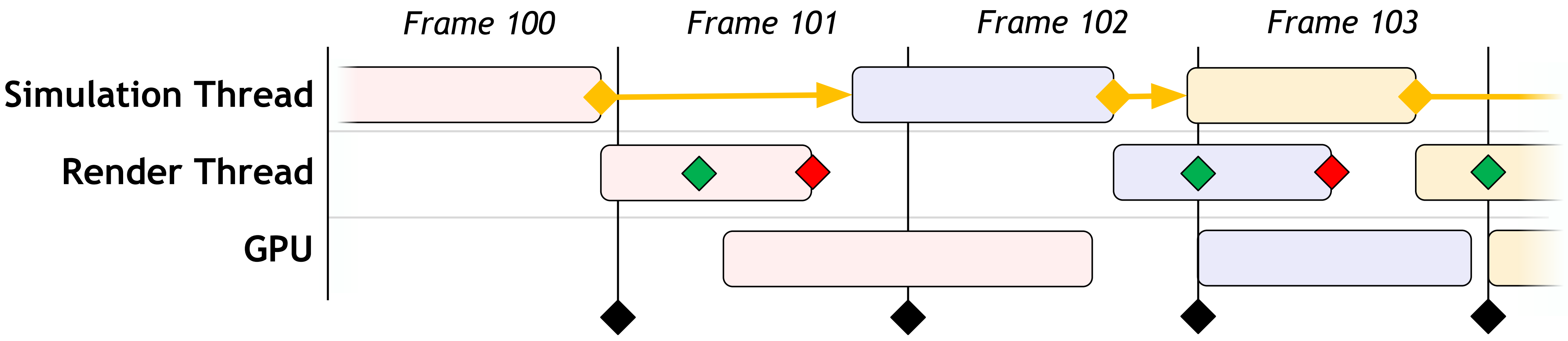
Blocks on two factors:

- Swap Interval, as requested as part of XrWaitFrameDescription, which is passed in
  - **Swap Interval = 1:** `xrWaitFrame` returns when it determines the application should start drawing for the next frame *at the display's native refresh cycle*
  - **Swap Interval = 2:** `xrWaitFrame` skips a refresh cycle before returning
  - **Swap Interval = 0:** Invalid, would rip a hole in space and time
- Throttling of the application by the runtime, in order to try and align GPU work with the compositor hook time

To see what this means, let's take a look at a slightly more complex multi-threaded engine example…

Frame Timing

Simple Multithreaded Example
(DX11, OpenGL)



- xrWaitFrame
- xrBeginFrame
- xrEndFrame
- Compositor Frame Hook

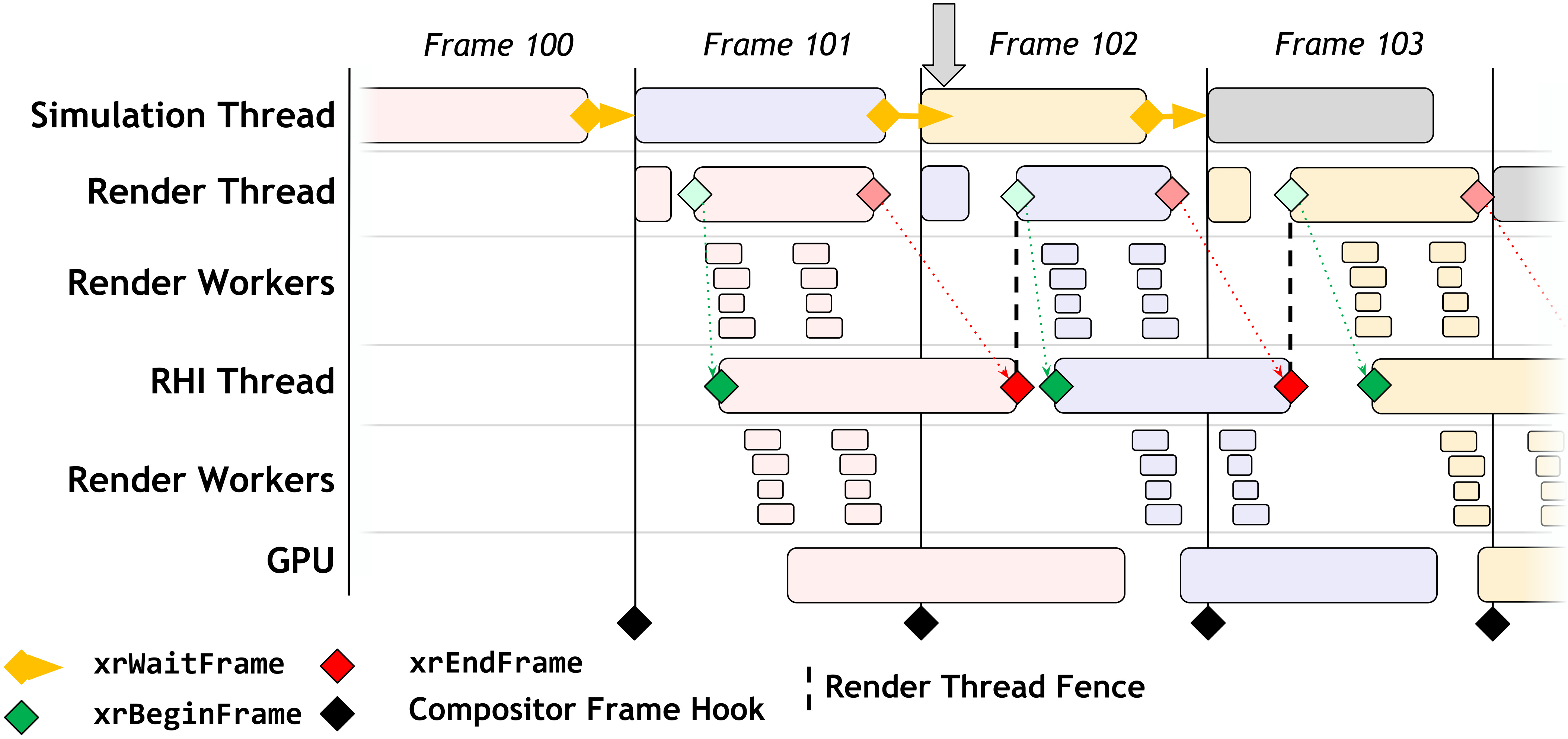
Frame 100: Late, so we hold *Frame 101* until `xrBeginFrame` can kick off right after the Compositor Frame Hook

Frame 101: Ideally scheduled. `xrBeginFrame` happens right after Compositor Hook for the previous frame, and GPU work finishes in time for the next Compositor Hook

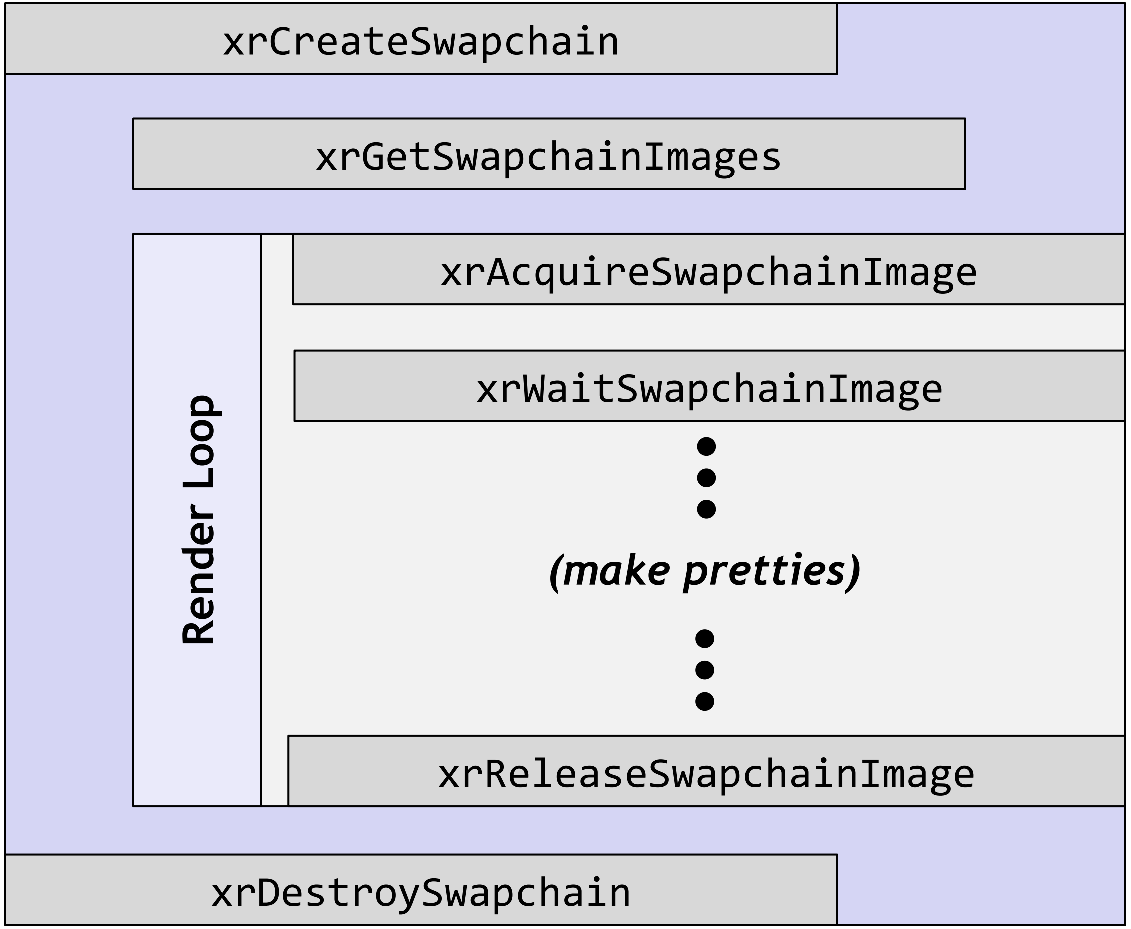
Frame Timing

Deeply Pipelined Multithreaded Example

(Unreal Engine 4 with Vulkan, DX12, Metal)



Swap Chains and Rendering



XrSwapchains:

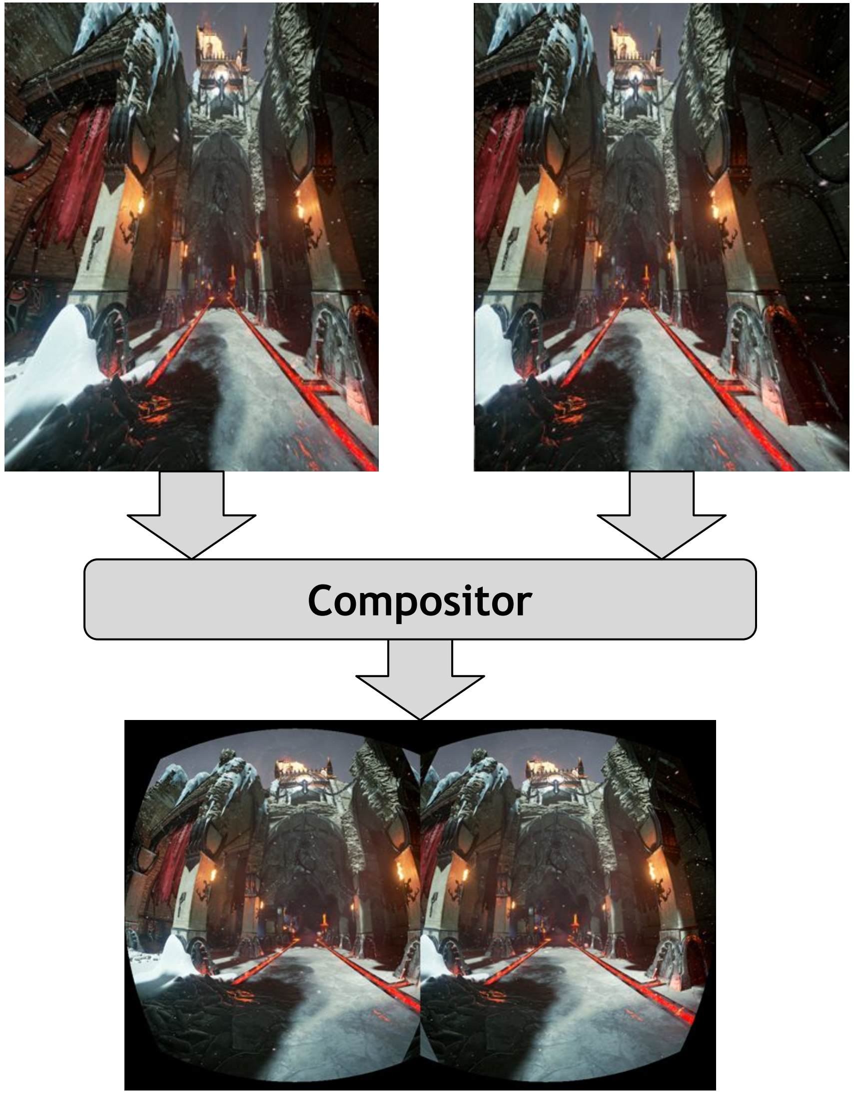
XrSwapchains are limited by the capabilities of the XrSession that they are being created for, and can be customized on creation based on application needs

- Usage Flags
- Format
- Width
- Height
- Swap chain length

Compositor Layers

The Compositor is responsible for taking all the Layers, reprojecting and distorting them, and displaying them to the device

- Layers are aggregated by the Compositor in `xrEndFrame` for display
- You can use multiple, up to the limit of the runtime
- Have `XrCompositionLayerData`:
  - Swap chain, and current index
  - Type, display time, eye, and `XrSpace`



Compositor Layers

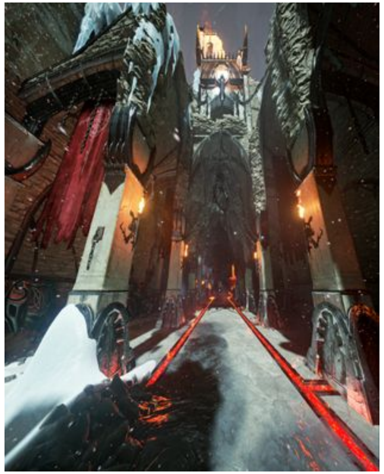
`XrCompositorLayerMultiProjection:`

Most common type of Layer. This is the classic “eye” layer, with each eye represented by a standard perspective projection matrix

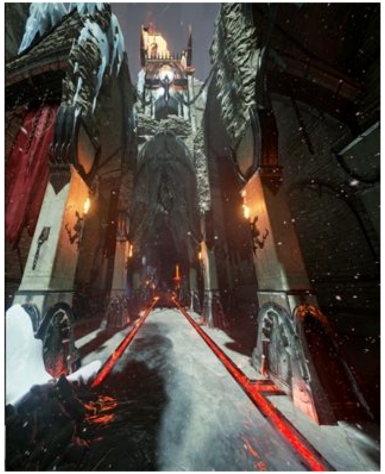
`XrCompositorLayerQuad:`

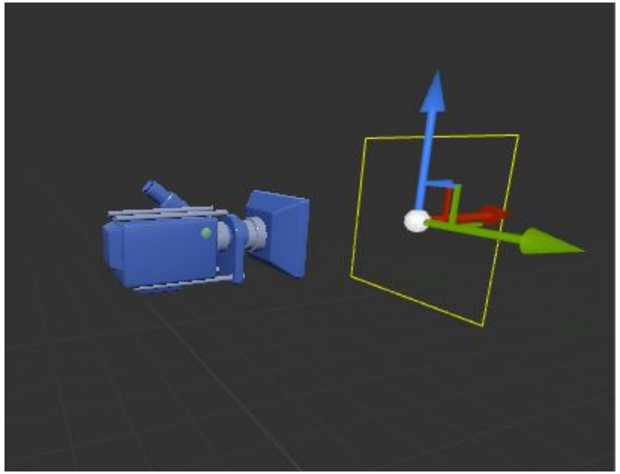
Quad layers are common for UI elements, or videos or images represented in the virtual world on a quad in virtual world space

XR_EYE_LEFT



XR_EYE_RIGHT





Viewport Configurations

Camera Passthrough AR	Stereoscopic VR	Projection CAVE
		 *Photo Credit: Dave Pape*
One Viewport	Two Viewports (one per eye)	Twelve Viewports (six per eye)
`/viewport_configuration/ar_mono/magic_window`	`/viewport_configuration/vr/hmd`	`/viewport_configuration/vr_cube/cave_vr`

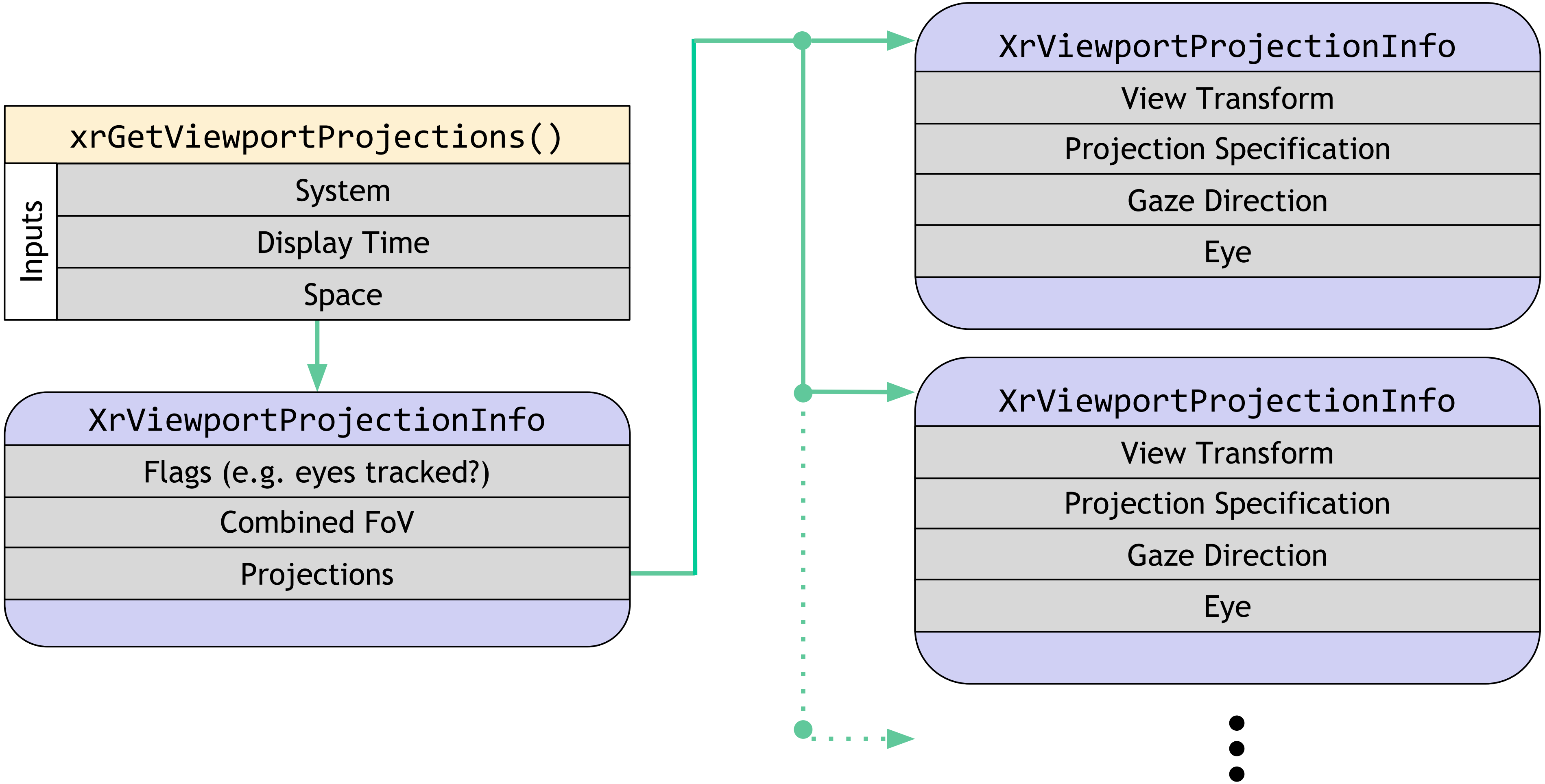
Applications can:

- Query the active `XrSystemId` for its supported Viewport Configurations
- Applications can then set the Viewport Configurations that they plan to use
- Select/change aspects of their active configuration over the lifetime of the `XrSession`

Runtimes can:

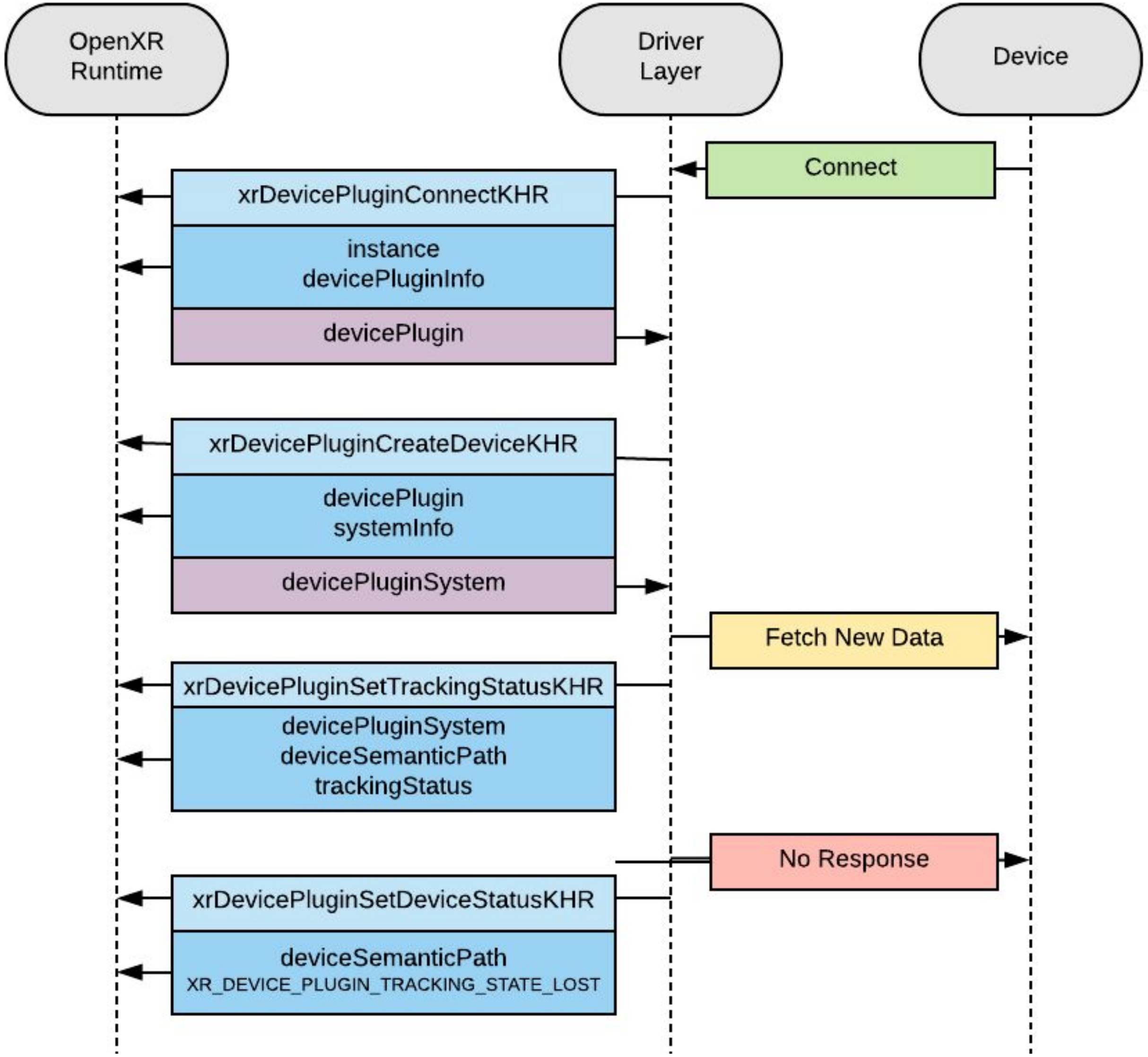
- Request the application change configuration, but app is not required to comply

Viewport Projections



Device Plugin

The Device device plugin allows a standard API for device manufacturers to communicate with OpenXR Runtimes.



Where Do We Go From Here?

A Brief History of the Standard

Call for Participation / Exploratory Group Formation -- *Fall F2F, October 2016: Korea*

Statement of Work / Working Group Formation -- *Winter F2F, January 2017: Vancouver*

Specification Work -- *Spring F2F, April 2017: Amsterdam*

Specification Work -- *Interim F2F, July 2017: Seattle*

Defining the MVP -- *Fall F2F, September 2017: Chicago*

Resolving Implementation Blockers -- *Winterim F2F, November 2017: Seattle*

Raising Implementation Issues -- *Winter F2F, January 2018: Taipei*

First Public Information! -- *GDC, March 2018: San Francisco*

Implementation and Refinement -- *Spring F2F, April 2018: Montreal*

Present Day

Updates & First Demonstration! -- *SIGGRAPH, August 2018: Right Here, Right Now!*

Coming Soon

Implementation, Conformance and Refinement -- *Fall F2F, September 2018*

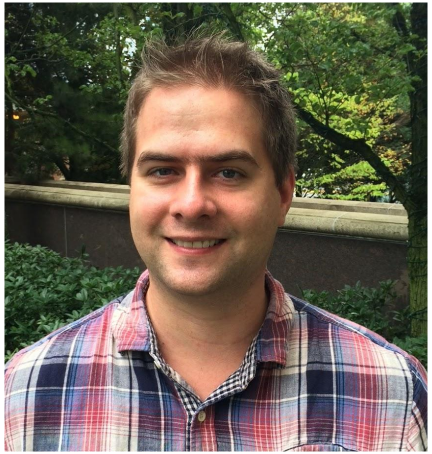
Provisional Release

Conformance Testing and Implementation

Ratification and Release

KHRONOS GROUP

Demos



Dr. Nick Whiting is currently Technical Director the of the award-winning Unreal Engine 4's virtual / augmented reality and machine learning efforts, including shipping the recent "Robo Recall", “Bullet Train,” "Thief in the Shadows," "Showdown," and "Couch Knights" VR titles. Previously, he has helped shipped titles in the blockbuster "Gears of War" series, including "Gears of War 3" and "Gears of War: Judgment." He is also currently serving as the chair of the Khronos OpenXR initiative, working to create a standard for VR and AR platforms and applications.

KHRONOS™ GROUP

The Structure

Unreal Showdown Demo

Unreal Engine

OpenXR Application Layer

Windows Mixed Reality Runtime
(inside-out head tracking)

StarVR Runtime
(quad viewport / large FOV)

Samsung Odyssey HMD

StarVR One HMD

Demos

Dr. Rémi Arnaud serves as Chief Architect Officer at Starbreeze, leading developments such as the StarVR SDK. Involved early on with real-time image generation in Paris where he did his Ph.D., he then relocated to California and since has worked on many projects including Silicon Graphics IRIS Performer, Keyhole's Earth Viewer, Intrinsic Graphics' Alchemy, Sony's PS3 SDK, Intel's Larrabee Game Engine, Screampoint's 5D City, Fl4re's game engine. Collaborated to various Khronos groups including OpenGL ES, COLLADA, glTF, webGL, webCL, and OpenXR.

Alex Turner is a Principal Program Manager at Microsoft, leading API design for the world's first mixed reality development platform that spans both holographic and immersive headsets! Before this, he was a PM on the Managed Languages team, where he drove the C#/VB Compiler team to ship Dynamic, Async and Windows 8, as well as Analyzers support as part of the .NET Compiler Platform ("Roslyn") project. Alex graduated with an MS in Computer Science from Stony Brook University and has spoken at GDC, BUILD, PDC, TechEd, TechDays and MIX.

Questions?