Standardizing All the Realities: A Look at OpenXR

Nick Whiting

GDC, March 2018

Agenda

- A Note on What We'll Cover
- A Brief History of the Standard
- Goals and Philosophies
- The Standard Itself
  - API Conventions and Primitives
  - The Instance, the System, and the Session
  - Input and Haptics
  - Rendering and Frame Timing
  - Layers and Viewports
  - Device Plugin Extension
  - Extensions
- Where Do We Go From Here
- Questions

KHRONOS™ GROUP

A Note on What We'll Cover

KHRONOS GROUP

A Note on What We'll Cover

- **A Quick Introduction to the Spec**
  - **Proviso:** This is still a work in progress!
    Things might change between now and the release of the spec.
  - Talk assumes that you know:
    - A little bit about VR and AR
    - A little bit about programming, and very basic real-time rendering
    - Nothing about the specification process
    - Nothing about any other Khronos specifications
  - This talk will not cover the whole spec!
    - We'll focus on what is (mostly) well-defined at this point
    - There are entire systems that aren't ready to be covered yet
- **Ample time will be given for Questions at the end!**
  - The spec is long…there may be some questions I can't answer
  - I can't answer questions about systems that aren't stabilized
  - No, you can't ask when it'll be released.

KHRONOS GROUP

A Brief History of the Standard

KHRONOS GROUP

KHRONOS
GROUP

AMD
antilatency
arm

COLLABORA

EPIC
GAMES

Google
htc

HUAWEI

Imagination

intel

LG
logitech

LUNAR

MEDIATEK

Microsoft

moz://a

NOKIA

NVIDIA

oculus

Pico
pluto

QUALCOMM

RAZER
SAMSUNG

sensics

Sony
Interactive
Entertainment

tobii

unity

VALVE

VeriSilicon

兆芯

zSpace

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: Washington

Defining the MVP
Fall F2F, September 2017: Chicago

Resolving Implementation Blockers
Winterim F2F, November 2017: Washington

Raising Implementation Issues
Winter F2F, January 2018: Taipei

Present Day

First Public Information!
GDC, March 2018: Right Here, Right Now!

Coming Soon

Provisional Release

Conformance Testing and Implementation

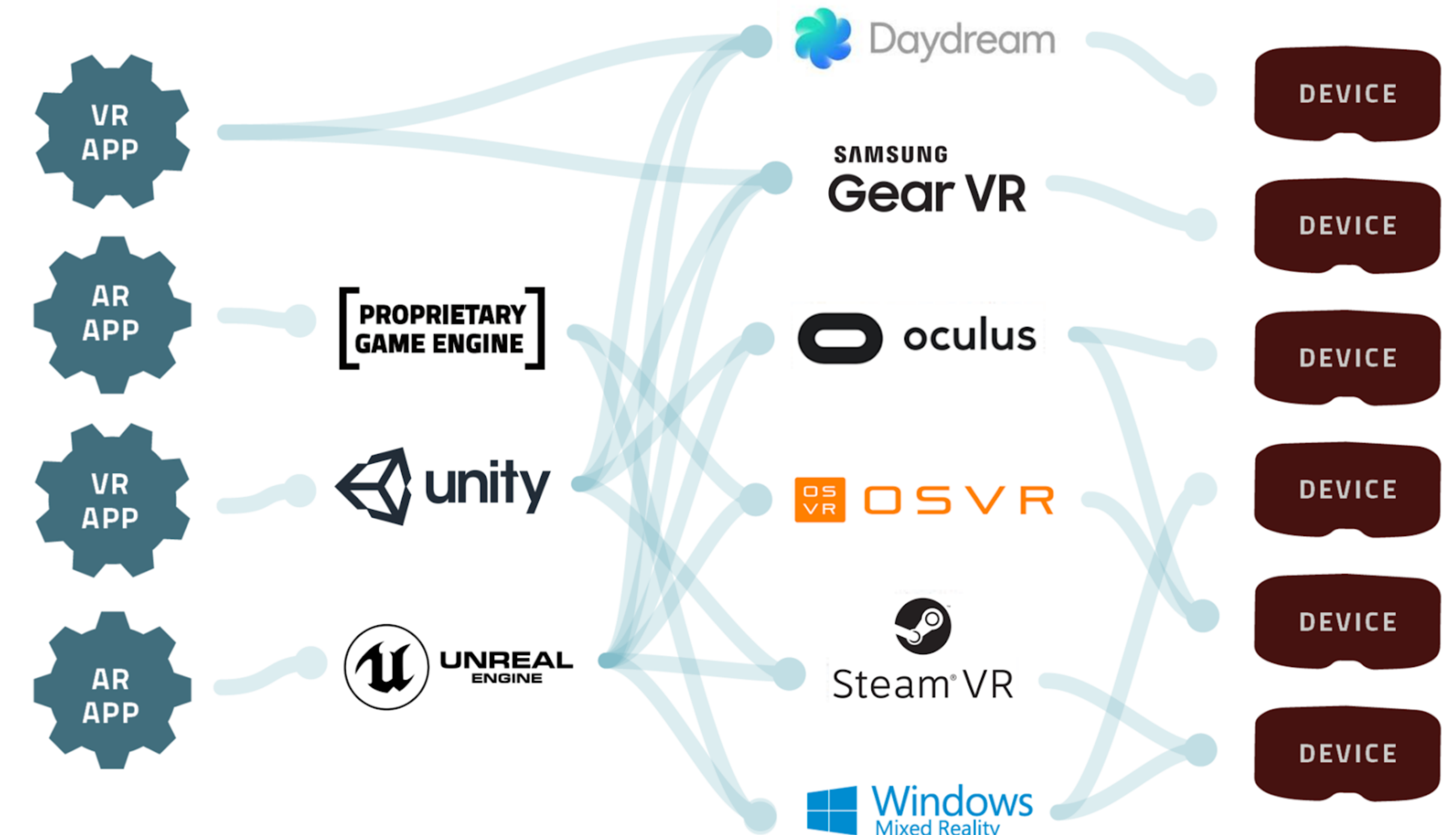
Ratification and Release

KHRONOS GROUP

Goals and Philosophies

KHRONOS GROUP

The Problem



KHRONOS GROUP

The Solution

Portable VR & AR Applications & Engines

CURRENT DEVICE STATE:

Normalized Predicted Poses

Controller / Peripheral State

Input Events

OpenXR
Application Interface

OUTGOING REQUESTS:

Pre-distortion image to display

Haptics

VR & AR Vendor Runtime System

Distortion Correction and Display Output

Coordinate System Unification & Prediction

CURRENT DEVICE STATE:

Controller / Peripheral State

Raw Poses

OpenXR Device Plugin Extension
(Optional)

OUTGOING REQUESTS:

Post-distortion image to display

Haptics

Device Vendor-Supplied Device Drivers

Portable VR & AR Devices

KHRONOS™ GROUP

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

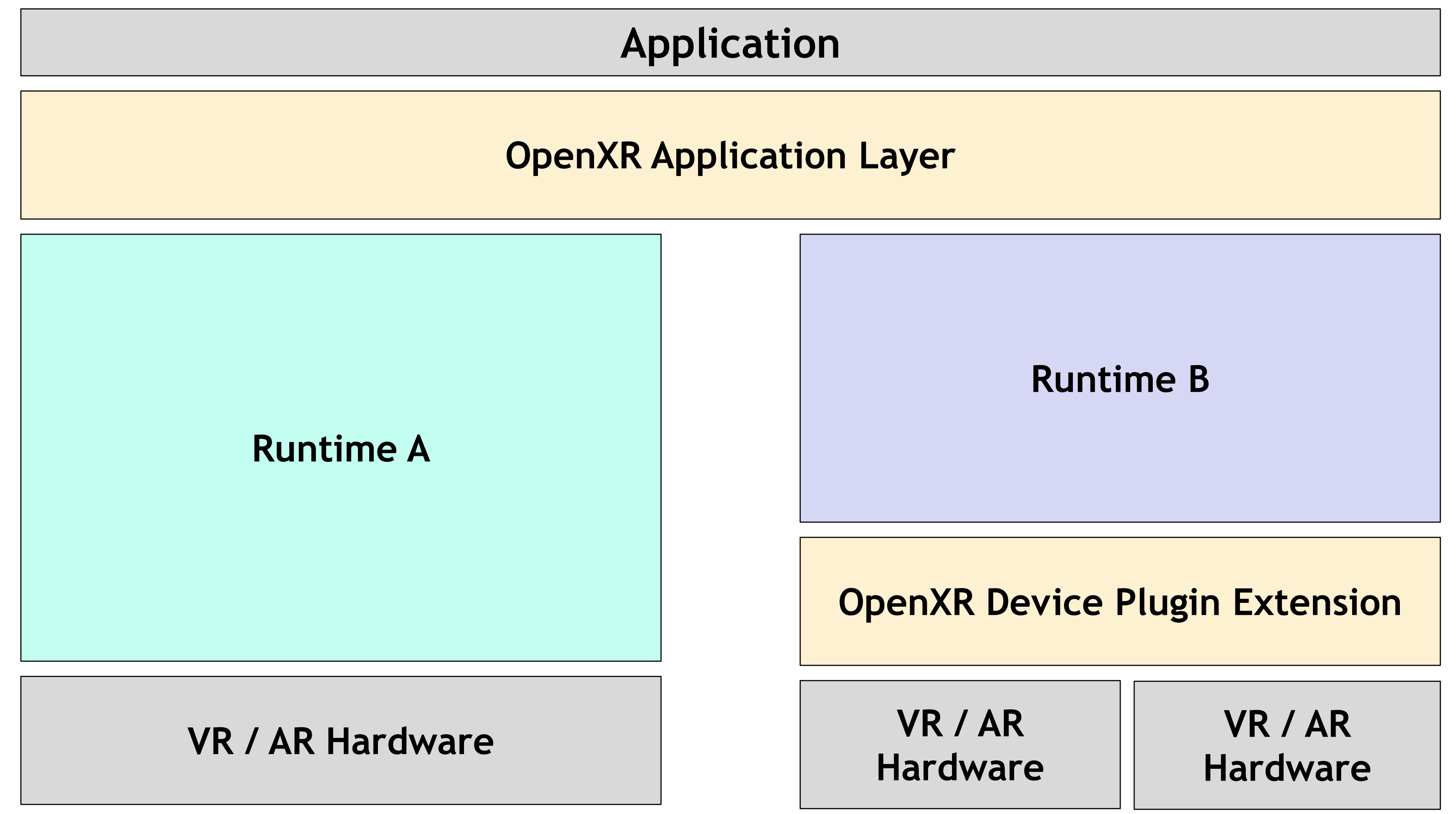
4

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

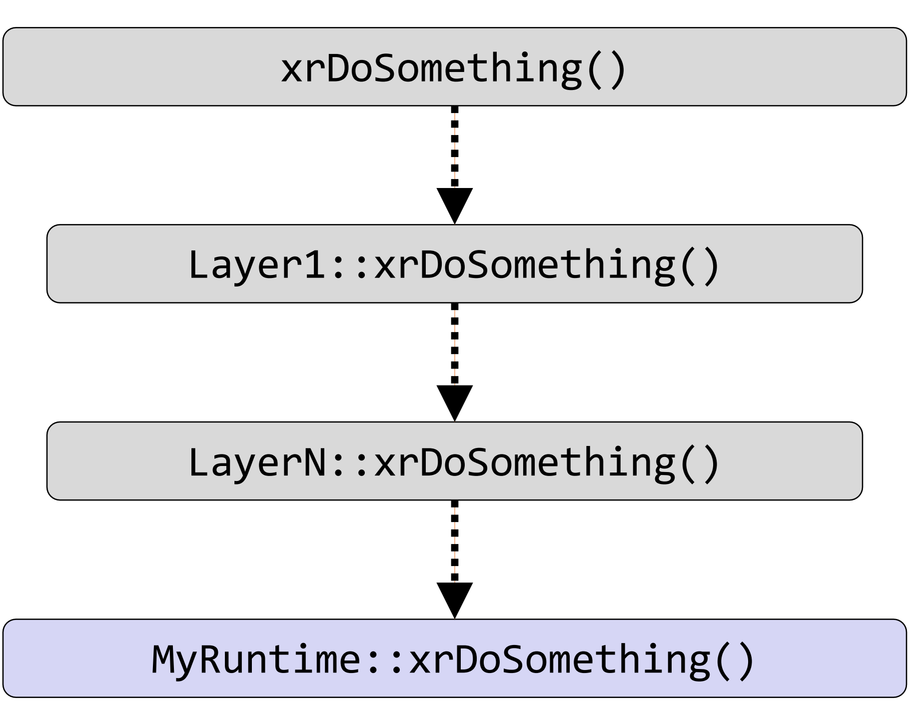
KHRONOS GROUP™

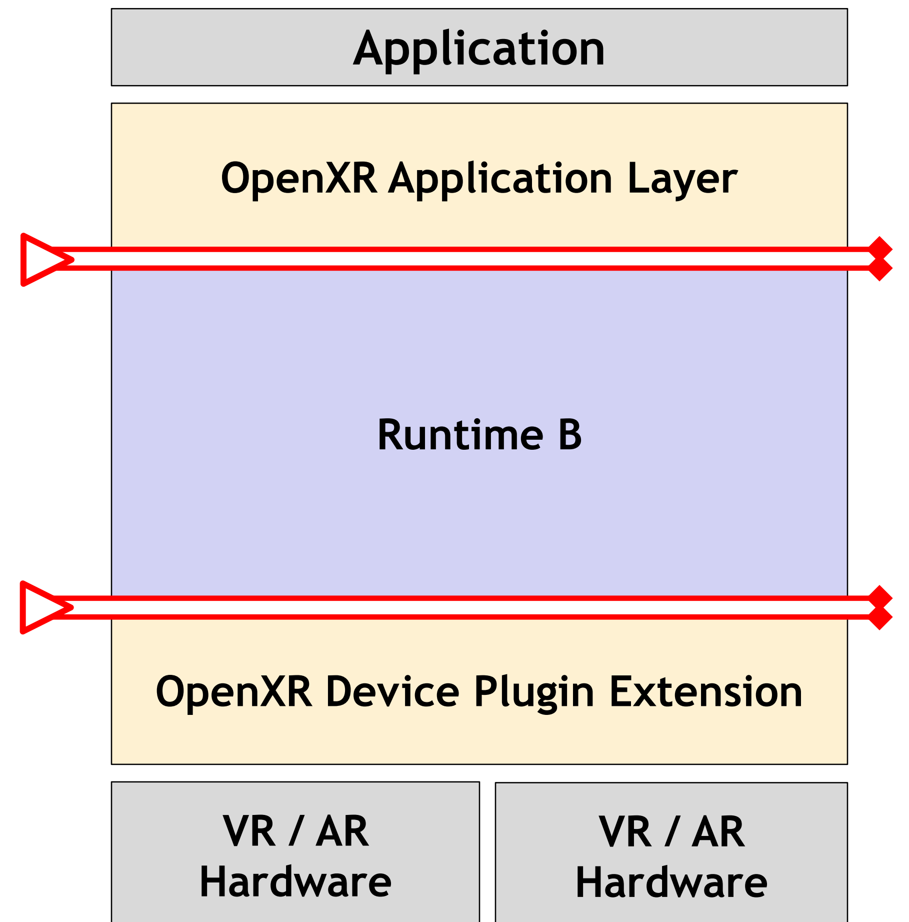
The Structure



KHRONOS GROUP

Layered API





Copyright ©2018 The Khronos™ Group Inc. - Page 13

Standard Overview

KHRONOS GROUP

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
- Requests for the same underlying object will return the same handle
- Hierarchical
  - E.g. To create an `XrSystem` handle, you must pass in a parent `XrInstance`
  - Handles for children are only valid within their direct parent's scope
- Typed using `XrObjectType` enumeration

KHRONOS GROUP

Copyright ©2018 The Khronos™ Group Inc.

API Conventions and Primitives

Semantic Paths

Semantic paths (`XrPath`) are a hierarchical string representation of *something* in the OpenXR ecosystem:

- Spaces
- Devices and their Input Sources
- Viewport Configurations

Examples:

```
/devices/my_vendor/my_device
/user/hand/left
/user/hand/primary
/user/head
/devices/my_vendor/my_controller/input/thumbstick/x
/viewport_configuration/vr
```

KHRONOS™ GROUP

API Conventions and Primitives

Semantic Paths

Properties of `XrPaths`:

- Hierarchical
- Can be aliased
  - `/user/hand/left == /user/hand/primary`
- Stored in a string table
- Human-readable
- Can be pre-defined (reserved) or application-defined
- Handles
- CaSe InSeNsItIvE
- `[A-Z, a-z,0-9,-,_,.,/]`
- Null terminated
- Not file paths!
  - Can't use ./ or ../ pathing

KHRONOS GROUP

Copyright ©2018 The Khronos™ Group Inc.

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
/devices/<vendor_name>/<device_name>
/devices/<vendor_name>/<device_name>/<identifier>/<component>
```

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

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

KHRONOS GROUP

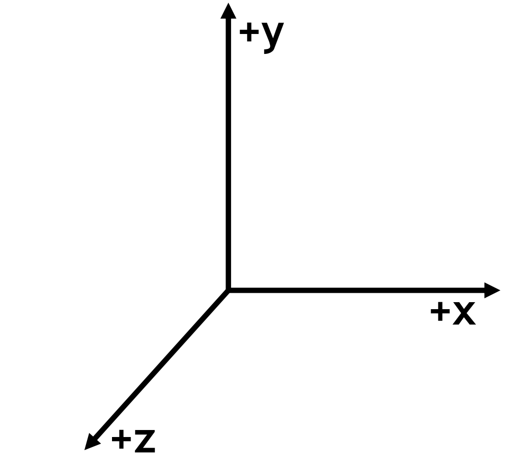
Copyright ©2018 The Khronos™ Group Inc. -

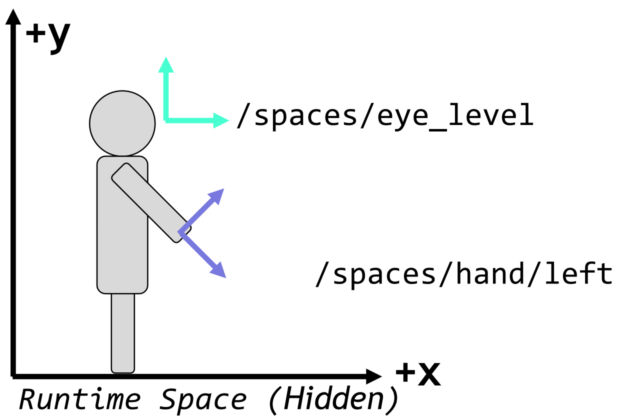
API Conventions and Primitives

XrSpace

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

- XrSpaces define meaningful spaces within the environment, which can be related to one another, and used as a basis for functions that return spatial values
- The Runtime can hold any representation it wants internally, and can adjust them as new data is acquired
- XrSpaces can also track dynamic objects, such as motion controllers, for ease of reference



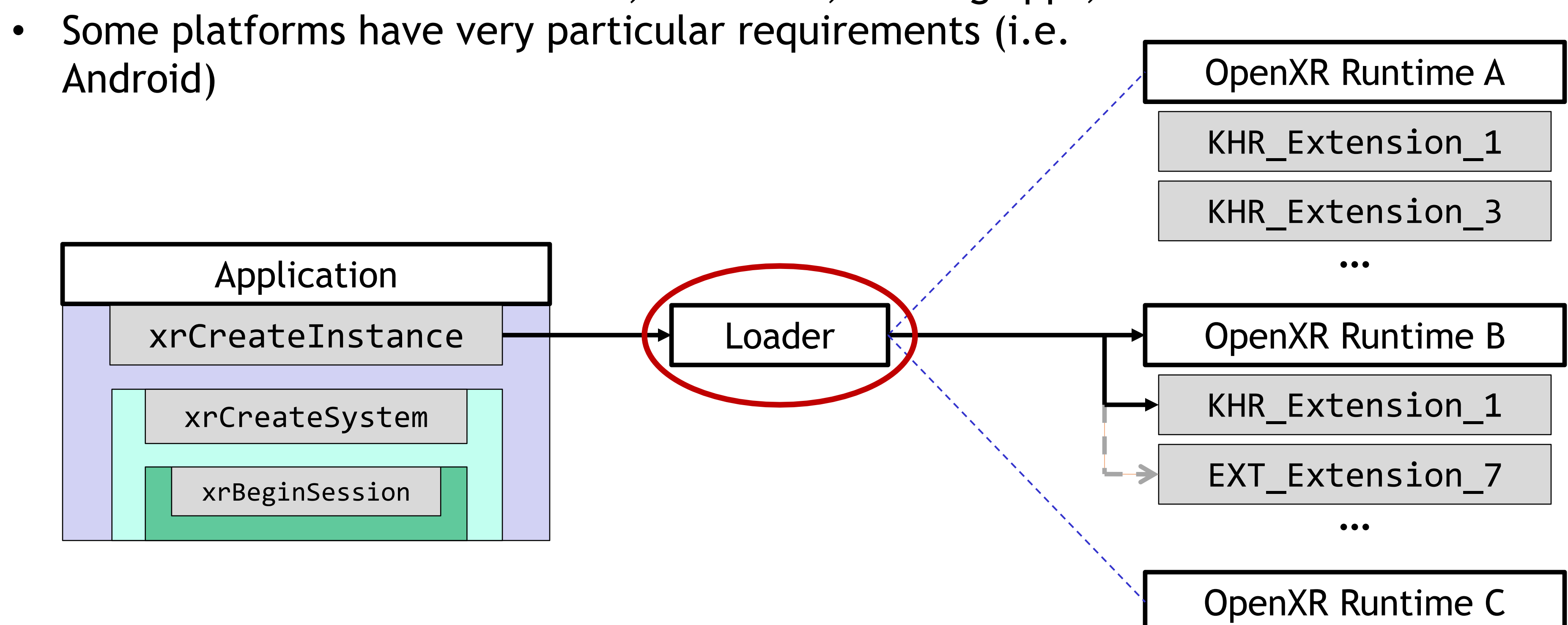


KHRONOS GROUP

The Instance, the System, and the Session

Loader:

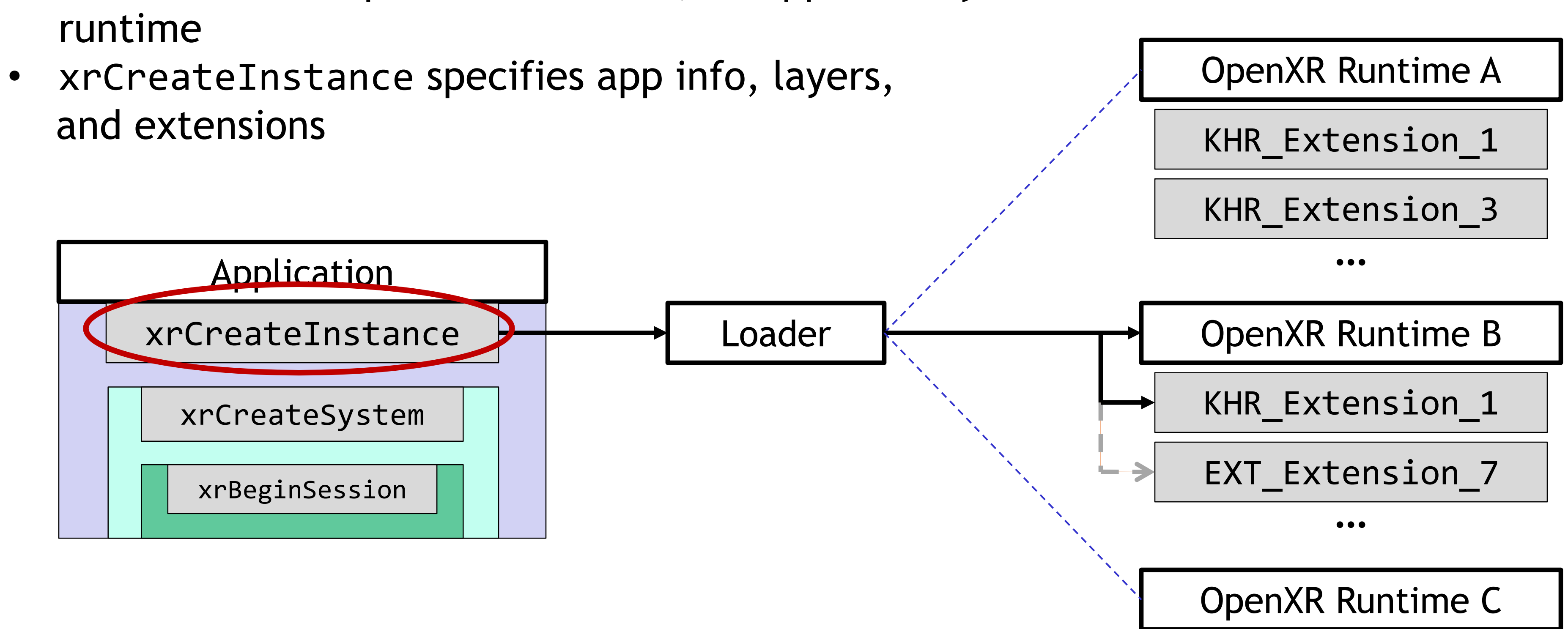
- Not required to use the OpenXR loader, but we have one
- Complexity can vary from “just pick one” to more intelligent decisions based on user factors, hardware, running apps, etc
- Some platforms have very particular requirements (i.e. Android)



The Instance, the System, and the Session

XrInstance:

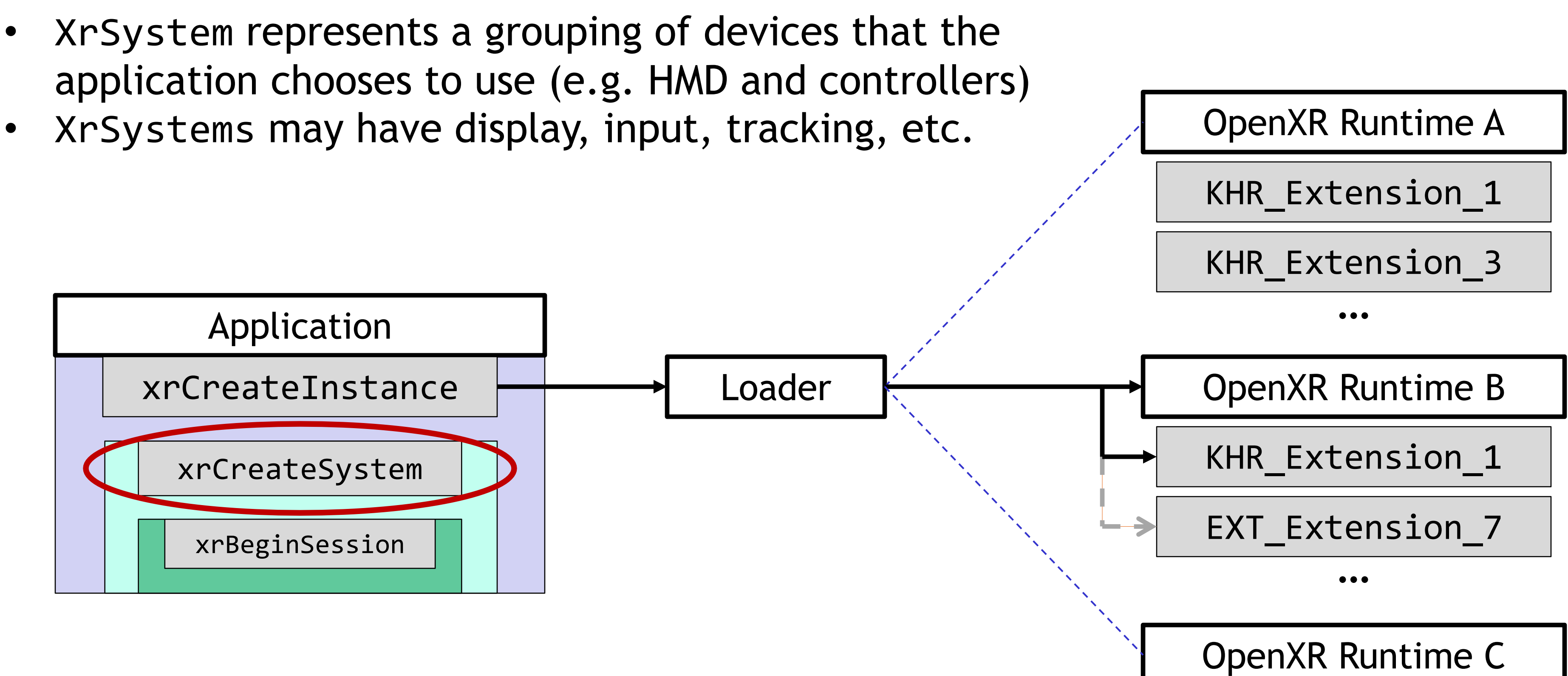
- The XrInstance is basically the application's representation of the OpenXR runtime
- Can create multiple XrInstances, if supported by the runtime
- xrCreateInstance specifies app info, layers, and extensions



The Instance, the System, and the Session

XrSystem:

- OpenXR groups physical devices into logical systems of related devices
- XrSystem represents a grouping of devices that the application chooses to use (e.g. HMD and controllers)
- XrSystems may have display, input, tracking, etc.



Copyright ©2018 The Khronos™ Group Inc.

The Instance, the System, and the Session

Session:

- A session is how an application indicates it wants to render and output VR / AR frames
- An app tells the runtime it wants to enter an interactive state by beginning a session with `xrBeginSession` and ending it with `xrBeginSession`
- No session, no frames.

KHRONOS™ GROUP

Copyright ©2018 The Khronos™ Group Inc.

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 yield to another application
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

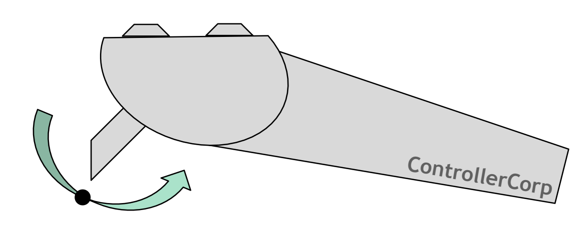
KHRONOS™ GROUP

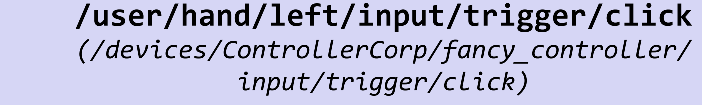
Copyright ©2018 The Khronos™ Group Inc.

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 in the runtime’s UI, etc)

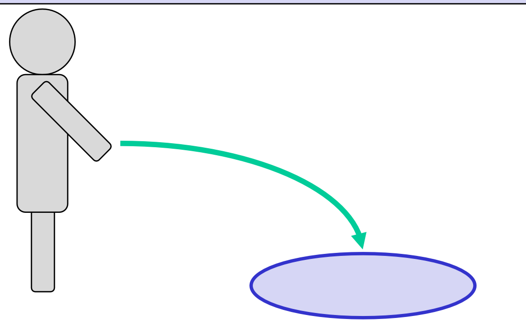




OpenXR Runtime

.../input/button_a/click	Explode
.../input/trigger/click	Teleport
.../input/grip/value	SpawnKittens
⋮	

XrAction: “Teleport”



KHRONOS™ GROUP

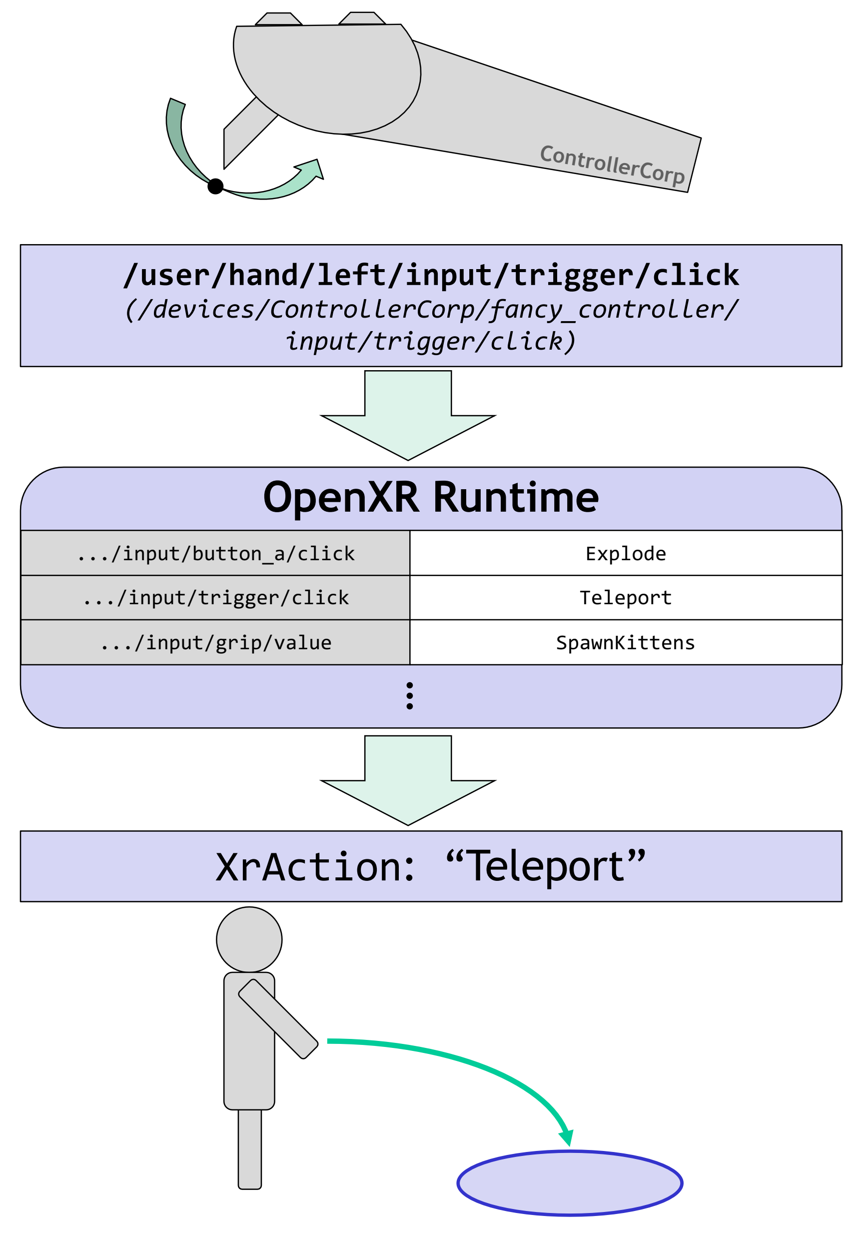
Input and Haptics

Forcing applications through this indirection has several advantages:

- Greater future-proofing as improvements to hardware, new form factors, 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



KHRONOS GROUP

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

KHRONOS™ GROUP

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, using the specified `XrSpace` as a basis:

- 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`

KHRONOS GROUP

Copyright ©2018 The Khronos™ Group Inc.

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

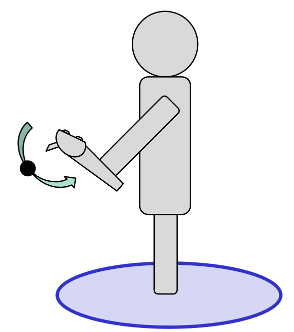
KHRONOS GROUP

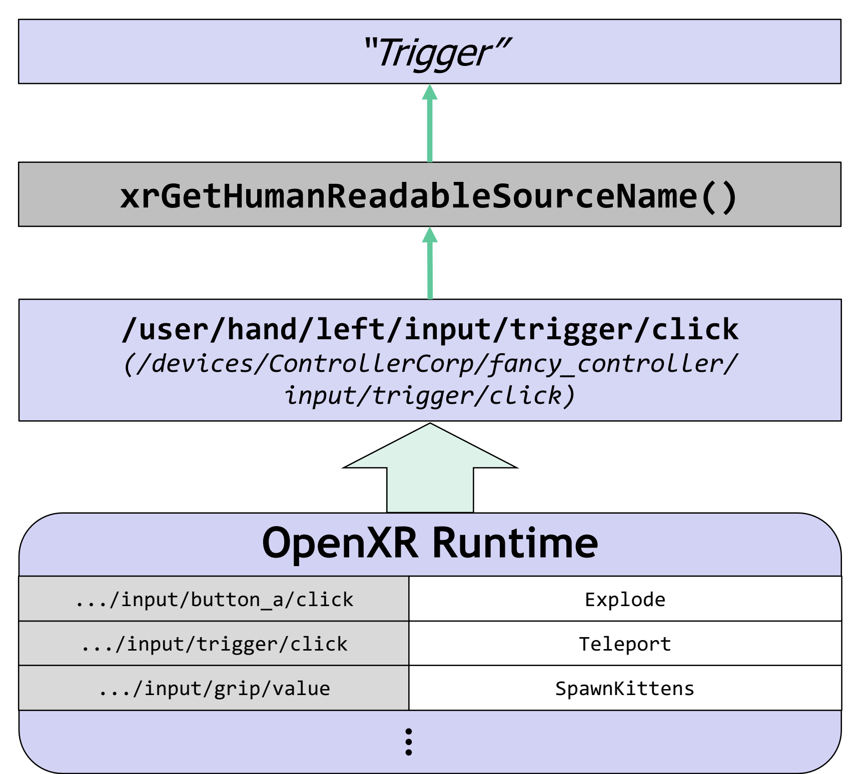
Copyright ©2018 The Khronos™ Group Inc.

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 "Pull the Trigger to Teleport!"

Pull the Trigger to Teleport!





KHRONOS™ GROUP

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

KHRONOS GROUP

Frame Timing

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

xrBeginFrame:

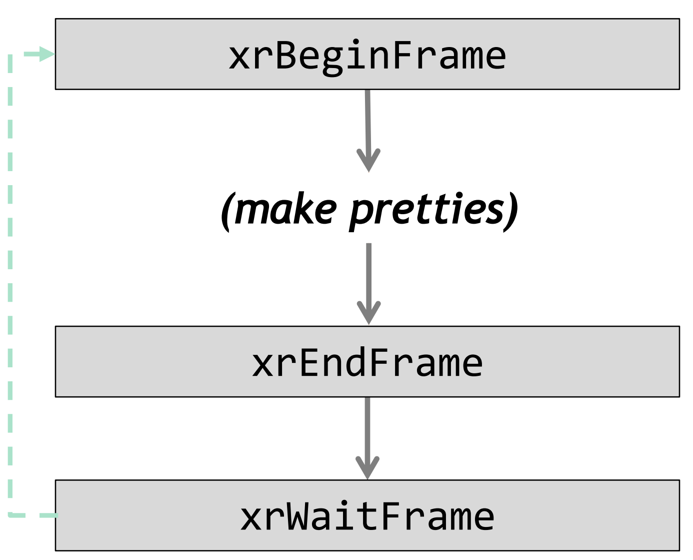
Signals that we’re ready to begin rendering pixels to the active image in our swap chain

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



Copyright ©2018 The Khronos™ Group Inc. -

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

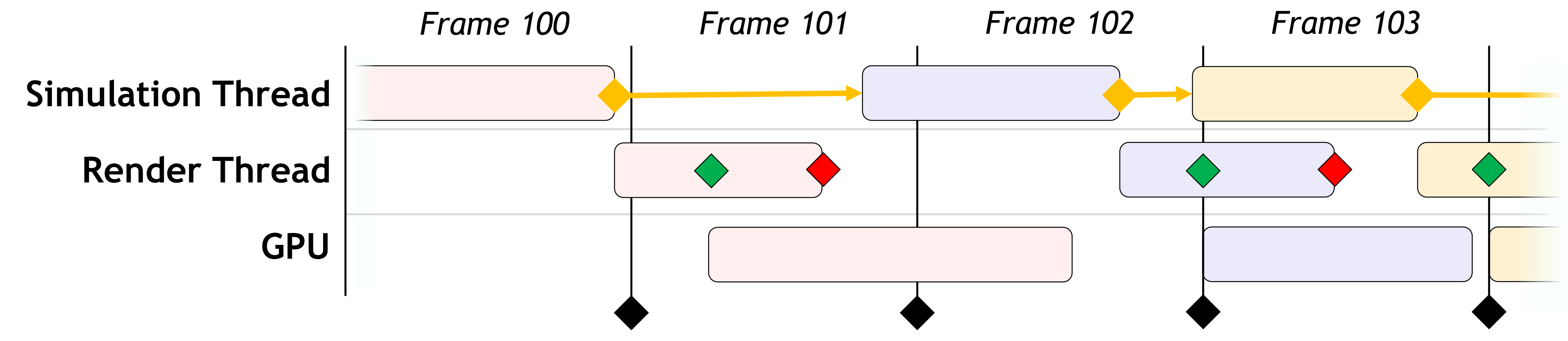
To see what this means, let's take a look at a slightly more complex multi-threaded engine example…

KHRONOS™ GROUP

Copyright ©2018 The Khronos™ Group Inc.

Frame Timing

Simple Multithreaded Example
(DX11, OpenGL)



`xrWaitFrame`

`xrBeginFrame`

`xrEndFrame`

Compositor Frame Hook

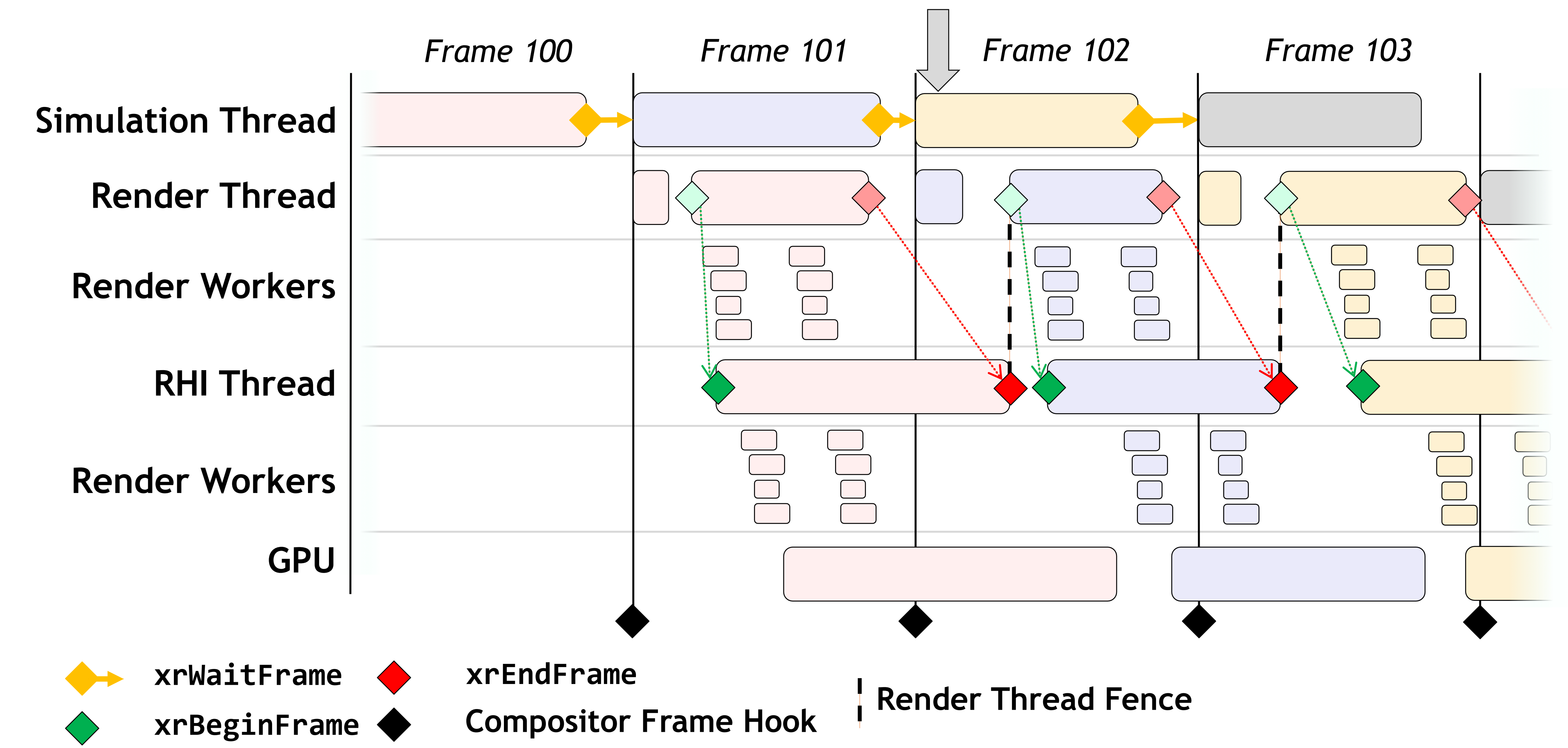
Frame 100: Late, so we hold *Frame 101* until `xrBeginFrame` can kick off right after the Compositor Frame Hook

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

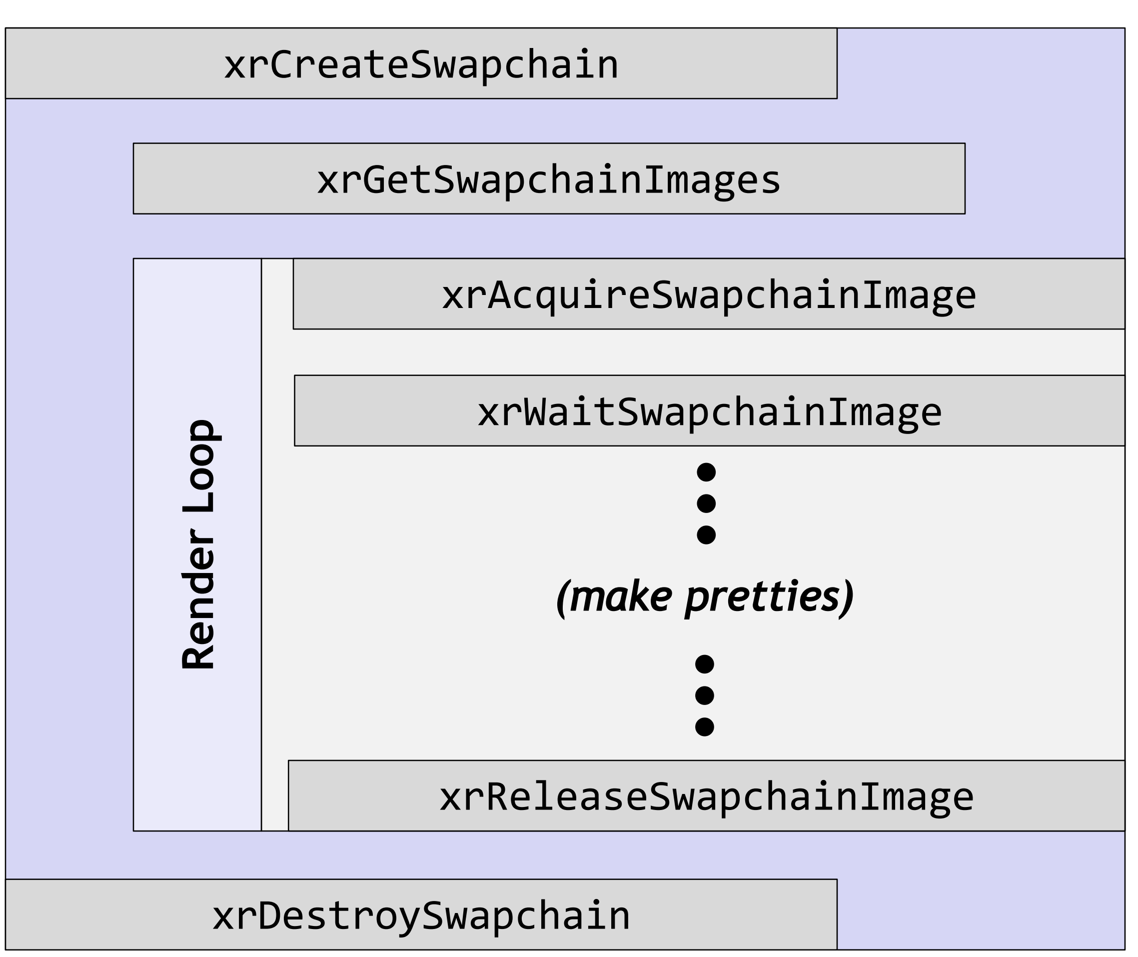
Deeply Pipelined Multithreaded Example

(Unreal Engine 4 with Vulkan, DX12, Metal)



KHRONOS GROUP

Swap Chains and Rendering



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

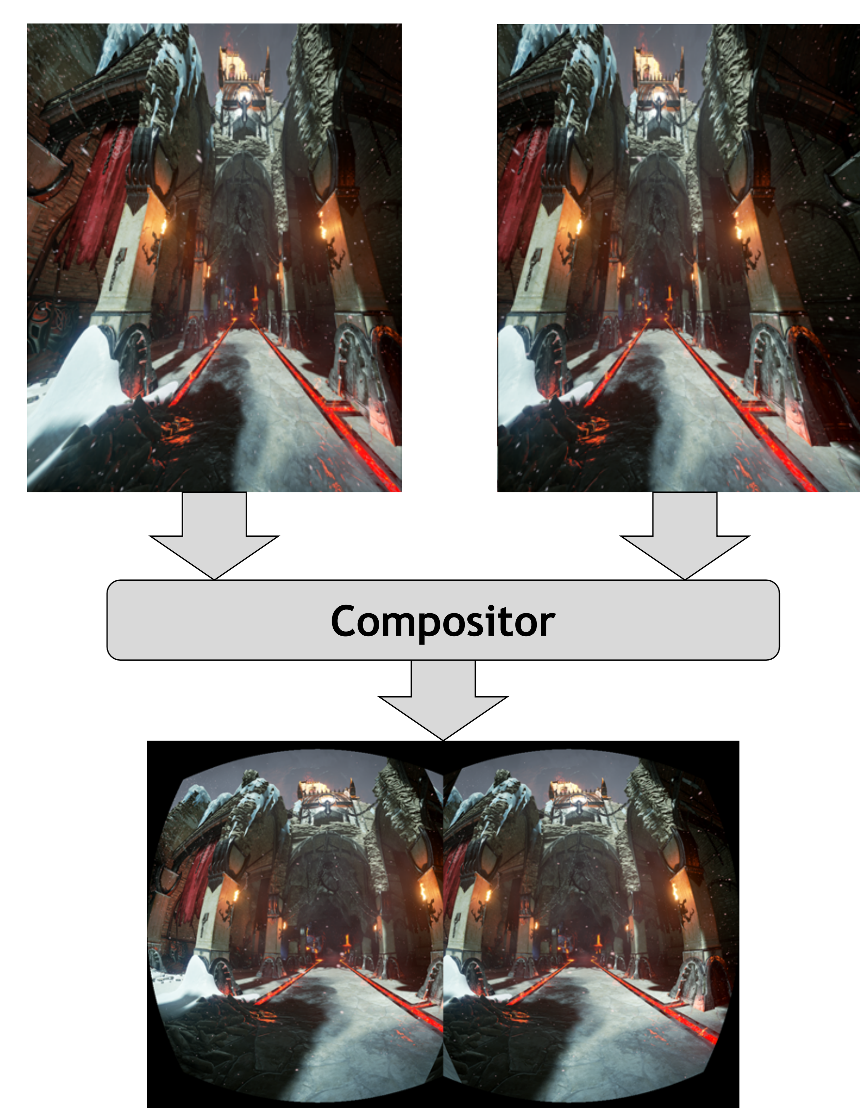
- Usage Flags
- Format
- Width
- Height
- Swap chain length

Copyright ©2018 The Khronos™ Group Inc.

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 `XrCompositionData`:
  - Type, display time, eye, and `XrSpace`
- Have `XrCompositionLayerData`:
  - Swap chain, and current index



KHRONOS™ GROUP

Compositor Layers

`XrCompositorLayerMultiprojection:`

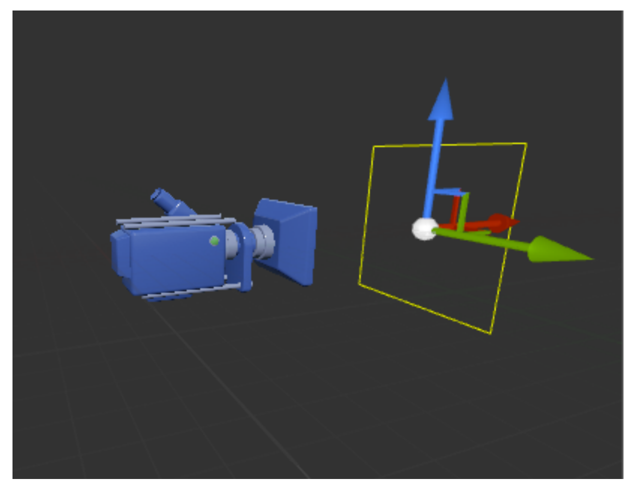
Most common type of Layer. This is the classic “eye” layer, with each eye represented by a standard perspective projection matrix

`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

Photo Credit: Dave Pape

Camera Passthrough AR	Stereoscopic VR / AR	Projection CAVE
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

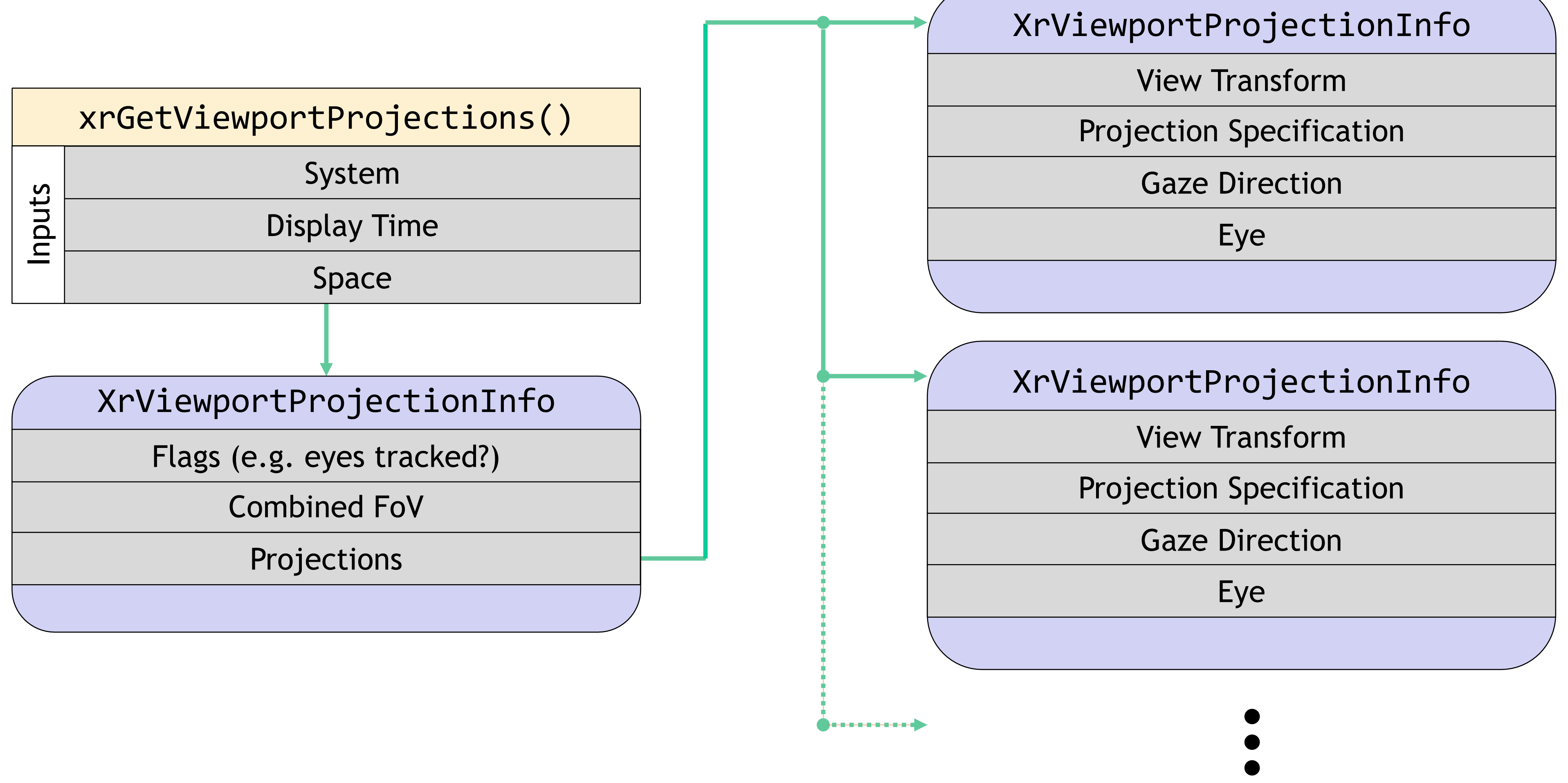
Applications can:

- Query the active `XrSystem` for its supported Viewport Configurations
- Applications can then set the Viewport Configurations that they plan to use
- Select and change their active configuration over the lifetime of the session

Runtimes can:

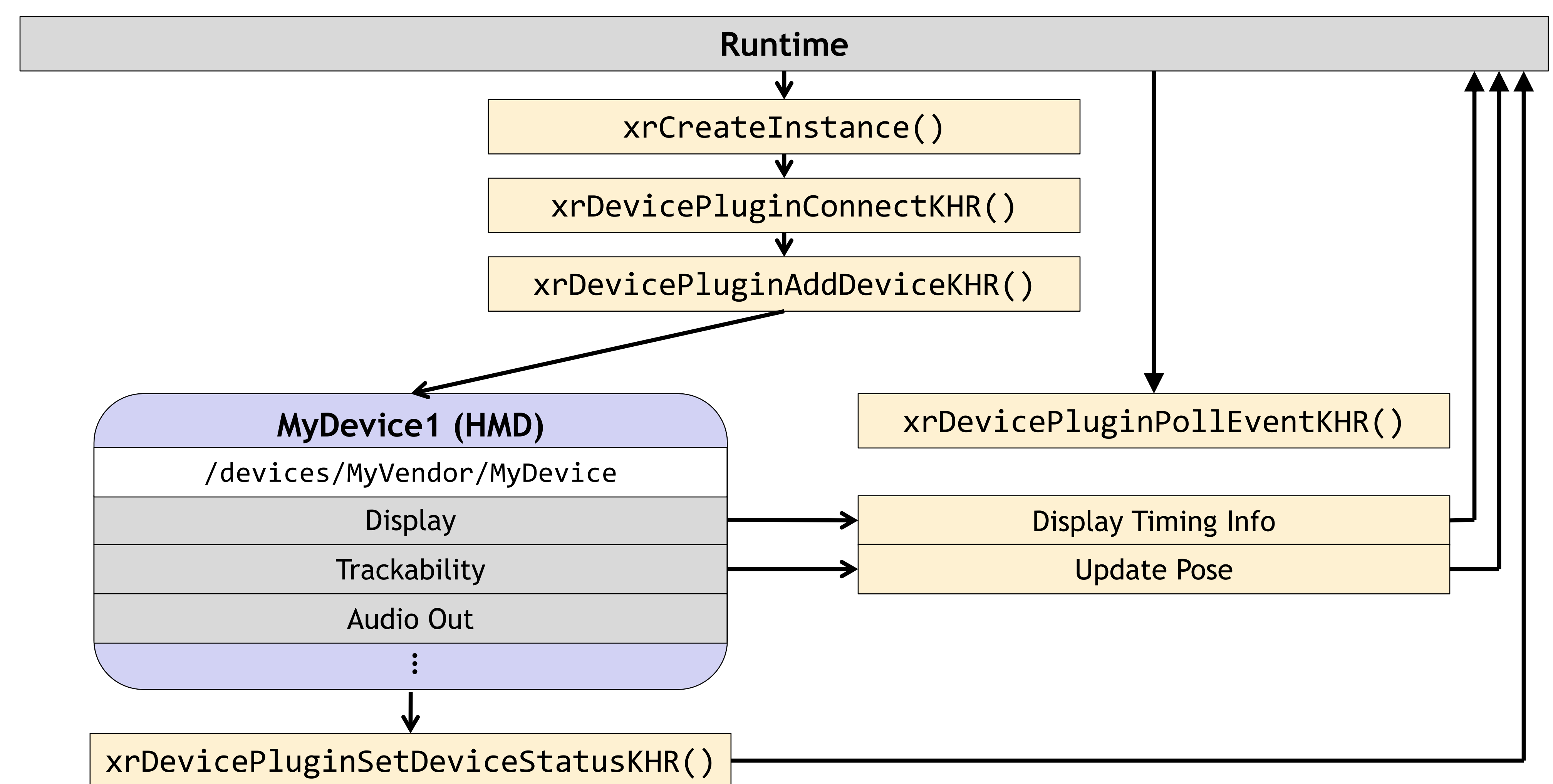
- Request the application change configuration, but app is not required to comply

Viewport Projections



Copyright ©2018 The Khronos™ Group Inc.

Device Plugin



Copyright ©2018 The Khronos™ Group Inc.

Extensions

Core Standard	Core concepts that are fundamental to the specification for all use cases *Examples: Instance management, tracking, frame timing*
KHR Extensions	Functionality that a large class of runtimes will likely implement *Examples: Platform support , Graphic API Extensions, Device Plugin, Headless, Tracking Bounds*
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*

Copyright ©2018 The Khronos™ Group Inc.

Where Do We Go From Here?

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: Washington

Defining the MVP
Fall F2F, September 2017: Chicago

Resolving Implementation Blockers
Winterim F2F, November 2017: Washington

Raising Implementation Issues
Winter F2F, January 2018: Taipei

Present Day

First Public Information!
GDC, March 2018: Right Here, Right Now!

Coming Soon

Provisional Release

Conformance Testing and Implementation

Ratification and Release

Copyright ©2018 The Khronos™ Group Inc.