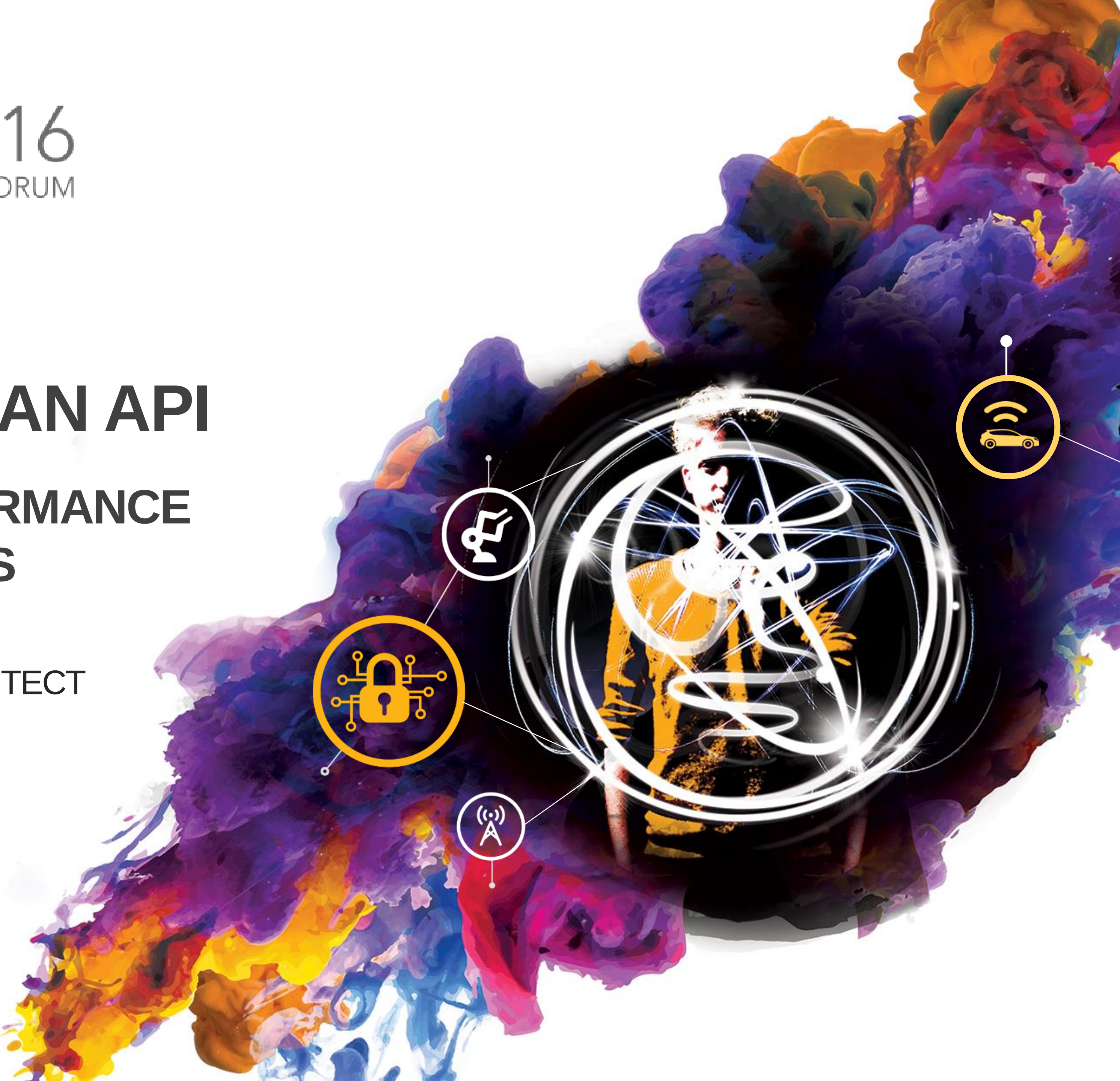
NXP | FTF 2016 TECHNOLOGY FORUM

i.MX 8 AND THE VULKAN API

THE FUTURE OF HIGH PERFORMANCE GRAPHICS IMPLEMENTATIONS

GABRIEL DAGANI
SENIOR GRAPHICS ENGINEER / SOC ARCHITECT
FTF-DES-N1740
MAY 19, 2016



AGENDA

- What is Vulkan?
- Vulkan vs. OpenGL ES and when to use which API
- Using Vulkan with i.MX 8

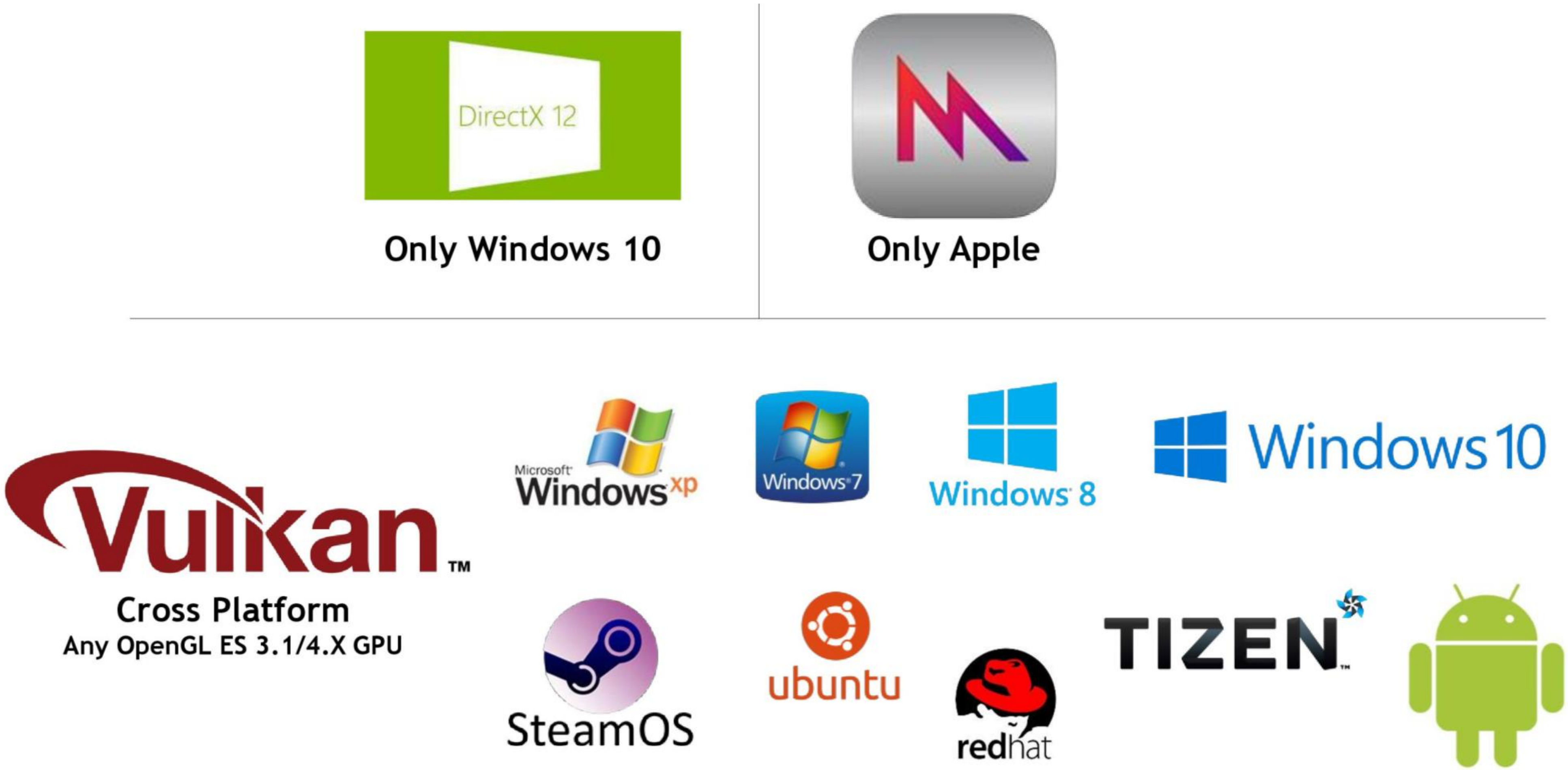
What Is Vulkan?

- **Vulkan is …**
  - Cross platform API
  - Open Source Standard
  - Meant for Graphics & Compute Workloads

- **Vulkan is NOT…**
  - The successor of OpenGL
  - The race of Spock (‘k’ not ‘c’)

HOW TO PERFORM THE VULCAN NERVE PINCH:
1. SNEAK UP BEHIND A "VOLUNTEER"
2. PLACE HAND OVER THE TRAPEZIUS NERVE BUNDLE AT THE BASE OF THE NECK.
3. APPLY GENEROUS AMOUNTS OF PRESSURE UNTIL VOLUNTEER FAINTS.

Next Generation GPU APIs



Only Windows 10

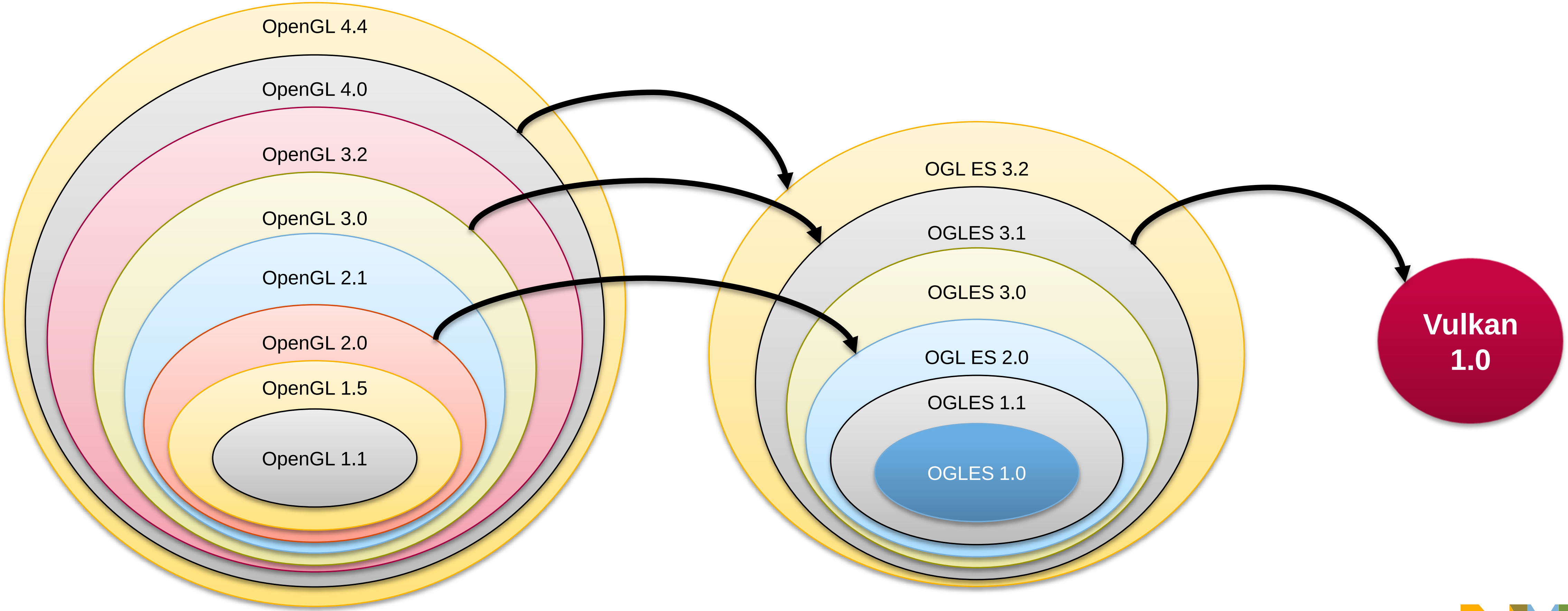
Only Apple

Cross Platform
Any OpenGL ES 3.1/4.X GPU

Source: Khronos Group

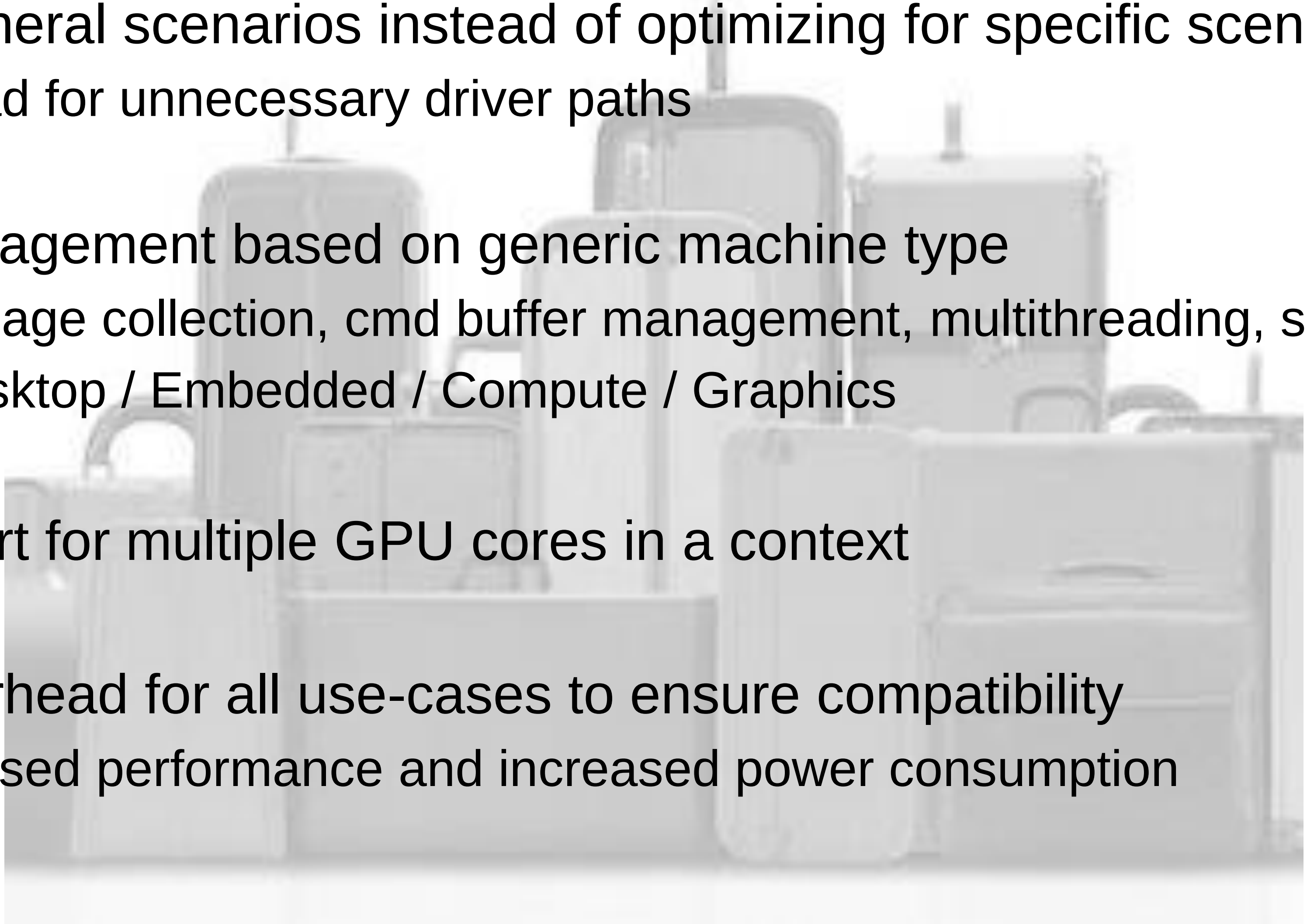
How Did Vulkan Come to Be?

- As graphics APIs evolved, backwards compatibility as a paradigm became unwieldy.
- Embedded Version of OGL emerged to simplify and enhance usage for size and constraints.



Excess Baggage from Backwards Compatibility

- Supporting general scenarios instead of optimizing for specific scenarios
  - Driver overhead for unnecessary driver paths
- Resource management based on generic machine type
  - Prebaked garbage collection, cmd buffer management, multithreading, scheduling, etc.
  - All Modes: Desktop / Embedded / Compute / Graphics
- Lacking support for multiple GPU cores in a context
- Increased overhead for all use-cases to ensure compatibility
  - Implies decreased performance and increased power consumption



The Need for a New Generation GPU API

- Explicit
  - Open up the high-level driver abstraction to give direct, low-level GPU control
- Streamlined
  - Faster performance, lower overhead, less latency
- Portable
  - Cloud, desktop, console, mobile and embedded
- Extensible
  - Platform for rapid innovation

OpenGL has evolved over 25 years and continues to meet industry needs – but there is a need for a complementary API approach

GPUs are increasingly programmable and compute capable + platforms are becoming mobile, memory-unified and multi-core

GPUs will accelerate graphics, compute, vision and deep learning across diverse platforms: FLEXIBILITY and PORTABILITY are key

Source: Khronos Group

What Developers Have Been Asking for…

… at least developers that need and can benefit from explicit control over GPU operation

Leading Edge Graphics Functionality
Equivalent to OpenGL in V1.0

General Purpose Compute
Graphics AND compute queues in a single API

Precompiled Shaders
SPIR-V for shading language flexibility including C++ Programming (future)

FUNCTIONALITY

Efficient Multi-threading
Use multiple CPU cores to create command buffers in parallel

Low Driver Overhead
Thinner, simpler driver reduces CPU bottlenecks and latency

PERFORMANCE

Same API for mobile, desktop, console and embedded
Defined 'Feature Sets' per platform
No need for 'Vulkan ES'

Explicit API
Direct control over GPU and memory allocation for less hitches and surprises

Clean, Streamlined API
Easier to program, implement and test for cross-vendor consistency

PORTABILITY

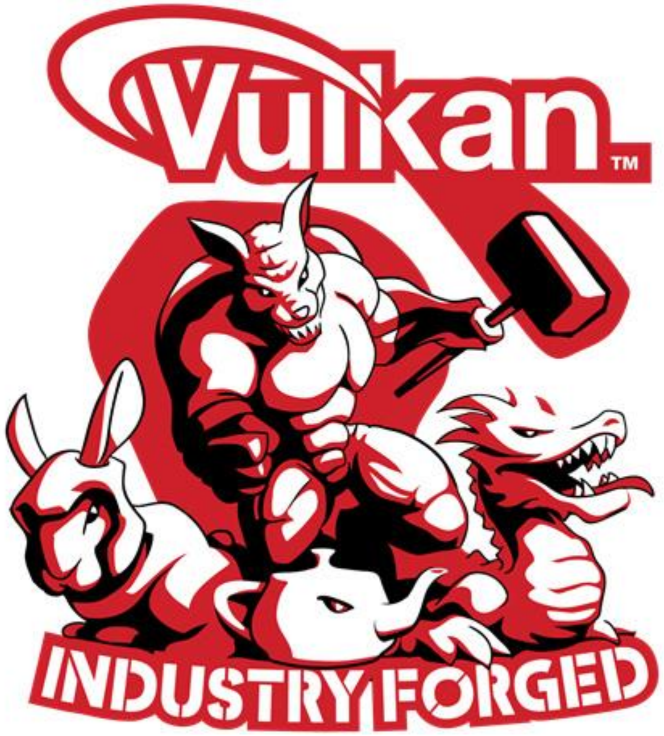
Source: Khronos Group

How Will Vulkan Help Embedded Graphics?

- **Benefits Performance**
  - Render more with the same GPU
  - Render “the same” with a lower cost GPU
  - Less latency for all use cases

- **Thin simple driver**
  - More robust
  - Smaller code footprint
  - Explicit GPU control and feature implementation

- **Naturally Cross platform**
  - Shares all same code – except windowing - on simulation and target

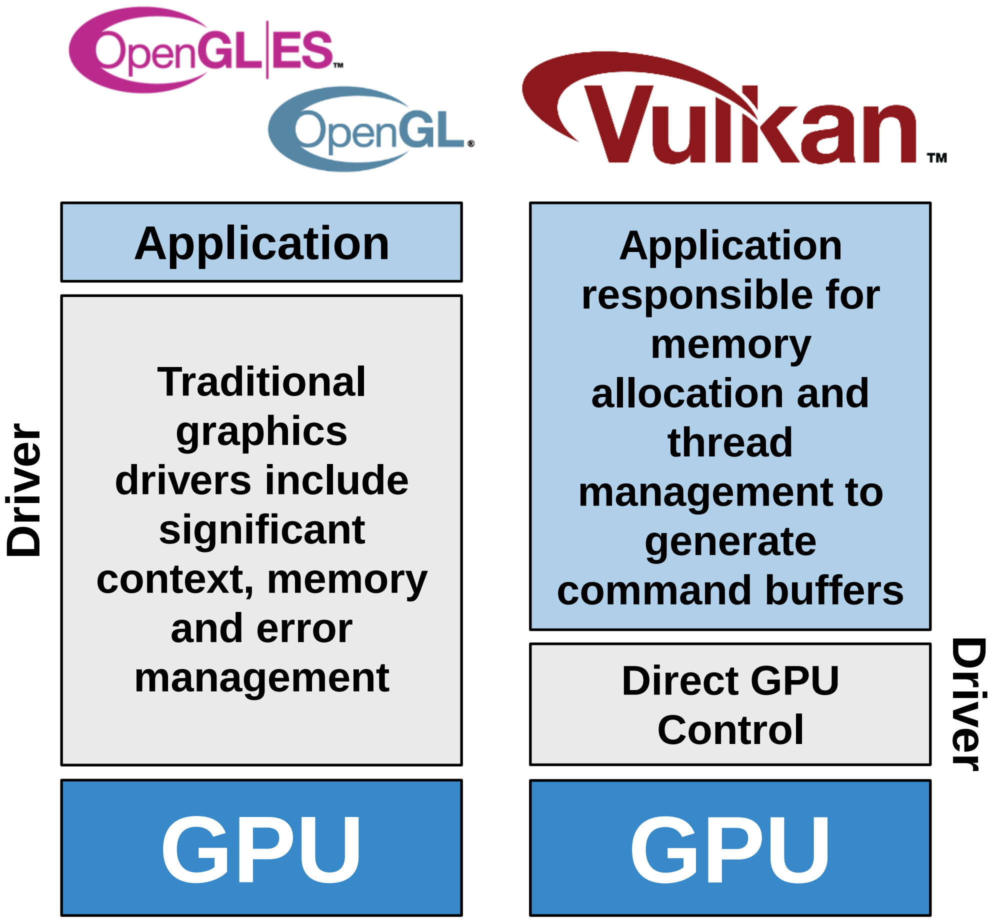


Vulkan Explicit GPU Control

Complex drivers lead to driver overhead and cross vendor unpredictability

Error management is always active

Driver compiles full shading language source



OpenGL ES

OpenGL

Application

Driver

Traditional graphics drivers include significant context, memory and error management

GPU

Vulkan

Application responsible for memory allocation and thread management to generate command buffers

Direct GPU Control

Driver

GPU

Simpler drivers for low-overhead efficiency and cross vendor consistency

Layered architecture so validation and debug layers can be loaded only when needed

Run-time only has to ingest SPIR-V intermediate language

Source: Khronos Group

What Are the Implications of a Thin Driver?

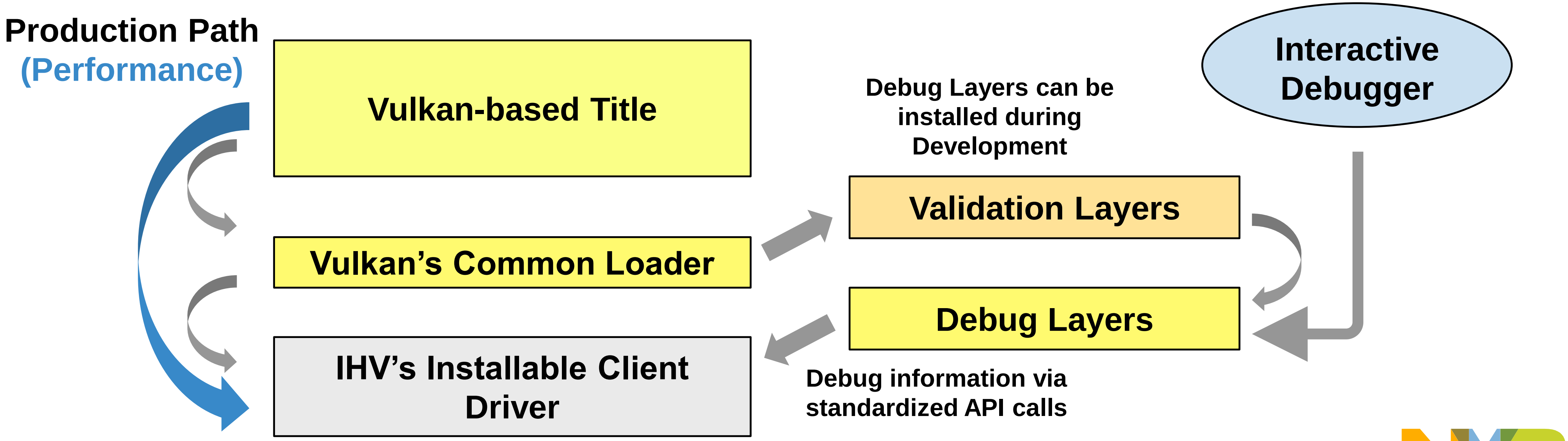
- **More Flexibility:**
  - All functions can and need to be re-invented and customized.
  - Features can be left out or simplified based on your use case.
  - Vulkan customized feature re-usability will be very important to users

- **More User Validation:**
  - Vulkan by itself does not validate or error check at any point for maximum performance
  - Developer responsible for robust application checking
    - Validation and debug layers available
    - Debug / Validation Layers can be removed at run-time

> “With Great Power Comes Great Responsibility”
>
> Spider-Man
>
> Saturday - Nov 10, 2012(2:00 am)

Vulkan Tools Architecture

- Layered design for cross-vendor tools innovation and flexibility
  - IHVs plug into a common, extensible architecture for code validation, debugging and profiling during development without impacting production performance
- Khronos Open Source Loader enables use of tools layers during debug
  - Finds and loads drivers, dispatches API calls to correct driver and layers


Production Path
(Performance)
Vulkan-based Title
Vulkan's Common Loader
IHV's Installable Client Driver
Debug Layers can be installed during Development
Interactive Debugger
Validation Layers
Debug Layers
Debug information via standardized API calls

Source: Khronos Group

Vulkan Layers Will Enhance the Eco-System

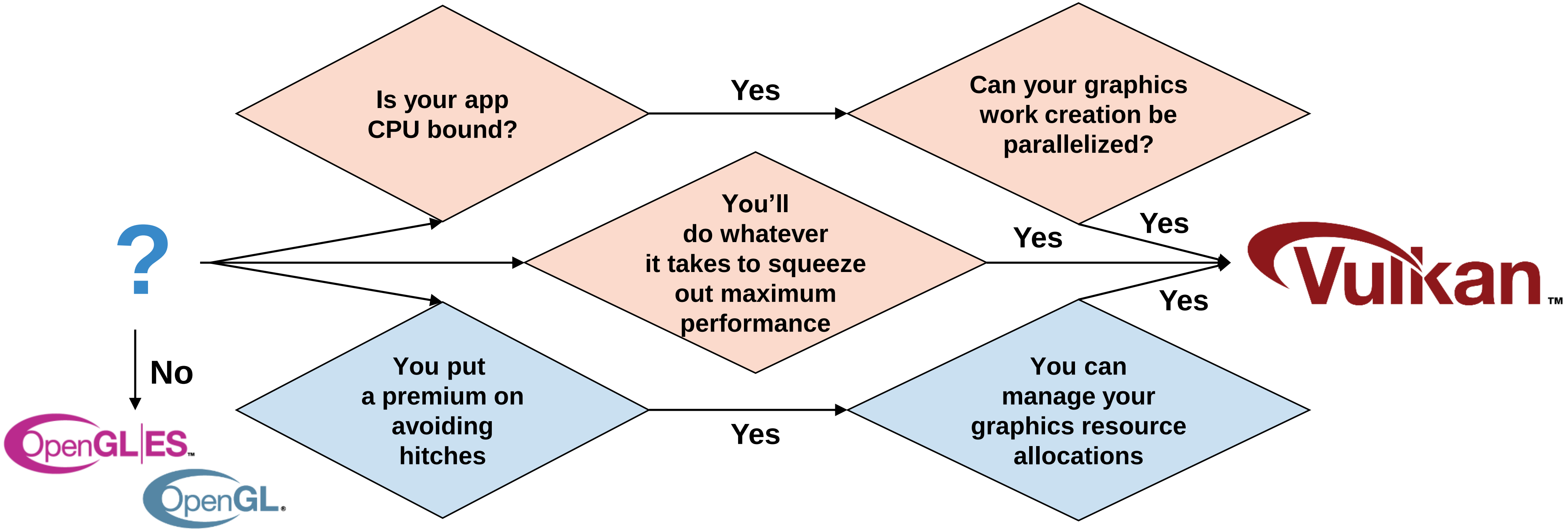
- **Layers can be enabled in production code – with performance tradeoff**

- **Software App Developers can develop and integrate their own layers**
  - E.g. special validation for safety applications
  - Driver Source doesn’t have to be available

- **Public layers will emerge**
  - Validation and Debug Layers
  - API Insertion layers
  - API Tracing Layers
  - Extra-functional Layers for System Integration

Which Developers Should Use Vulkan?

- Vulkan puts more work and responsibility into the application
  - Not every developer will need or want to make that extra investment
- For many developers OpenGL and OpenGL ES will remain the most effective API
  - Khronos actively evolving OpenGL and OpenGL ES in parallel with Vulkan


?
Is your app CPU bound?
Yes
Can your graphics work creation be parallelized?
You'll do whatever it takes to squeeze out maximum performance
Yes
Yes
Yes
Vulkan
You put a premium on avoiding hitches
Yes
You can manage your graphics resource allocations
No
OpenGL ES
OpenGL

Vulkan provides more choice to developers and can be used to create new classes of end-user experience

Source: Khronos Group

Vulkan vs. OpenGL ES (Use Case Dependent)

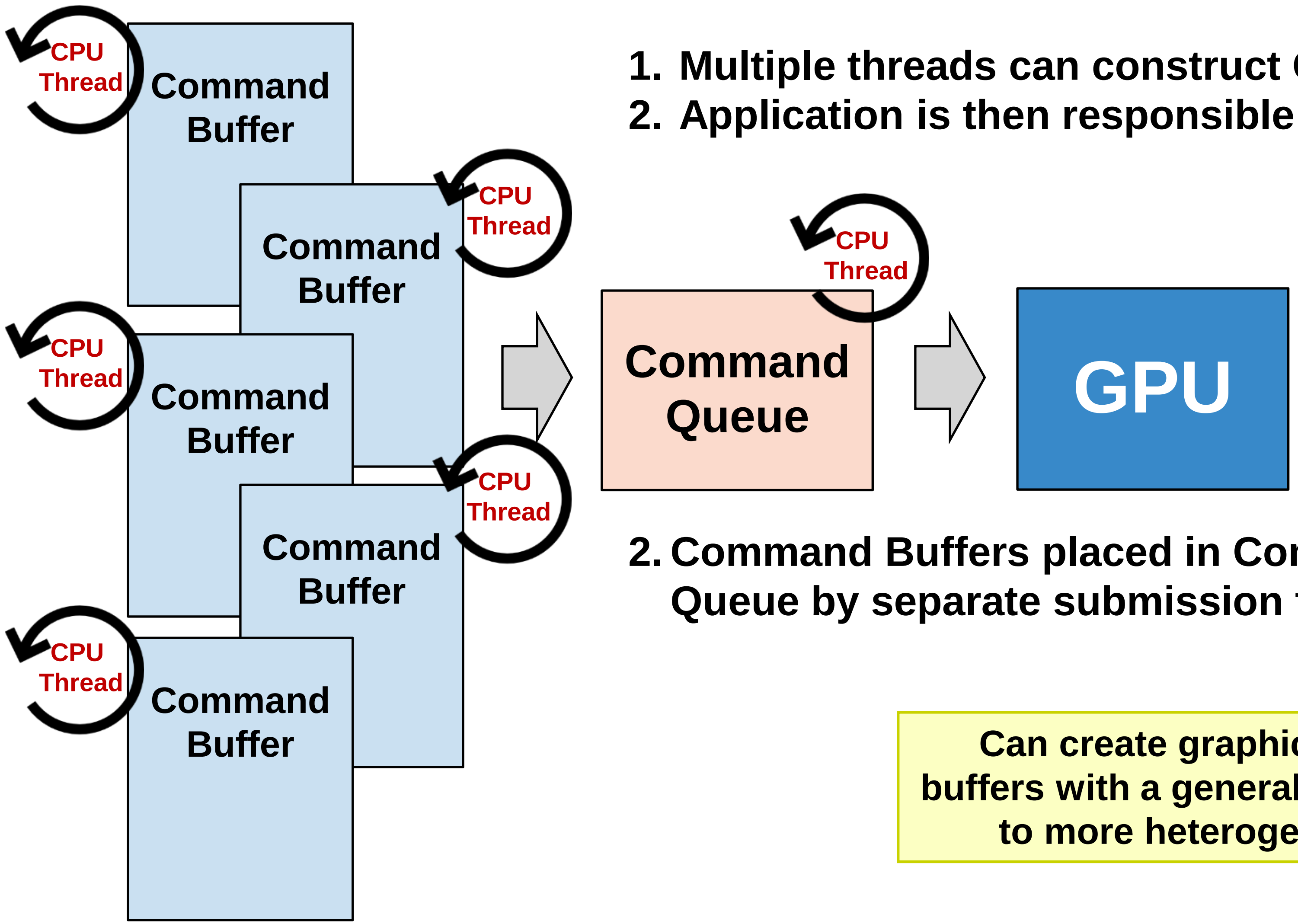
- **Vulkan is code hungry**
  - Simple Triangle Application
    - OpenGL ES : 50 lines of code
    - Vulkan: 500 lines of code

- **Vulkan is Error Intolerant**
  - Without layers there is no offline or real-time error checking

- **Vulkan is Hardware Agnostic**
  - Program will crash based on missing hardware features
  - Up to the application developer to check hardware restrictions

- **Recommended**
  - For simple graphics (or performance insensitive apps) use OpenGL ES
  - Otherwise:
    - Implement own or use public abstraction layer
    - Enable and implement only the layers which are needed for your application

Vulkan Multi-threading Efficiency



1. Multiple threads can construct Command Buffers in parallel
2. Application is then responsible for thread management and sync-ing

2. Command Buffers placed in Command Queue by separate submission thread

Can create graphics, compute and DMA command buffers with a general queue model that can be extended to more heterogeneous processing in the future

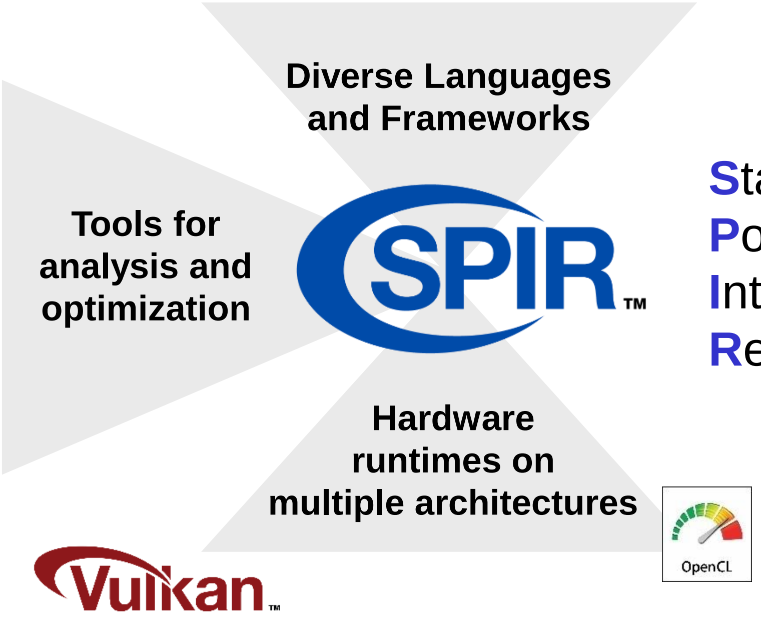
Source: Khronos Group

SPIR-V Transforms the Language Ecosystem

- First multi-API, intermediate language for parallel compute and graphics
  - Native representation for Vulkan shader and OpenCL kernel source languages
  - https://www.khronos.org/registry/spir-v/papers/whitepaper.pdf
- Cross vendor intermediate representation
  - Language front-ends can easily access multiple hardware run-times
  - Acceleration hardware can leverage multiple language front-ends
  - Encourages tools for program analysis and optimization in SPIR form

Multiple Developer Advantages

Same front-end compiler for multiple platforms
Reduces runtime kernel compilation time
Don't have to ship shader/kernel source code
Drivers are simpler and more reliable


Diverse Languages and Frameworks
Tools for analysis and optimization
SPIR™
Hardware runtimes on multiple architectures
Vulkan
OpenCL

Standard
Portable
Intermediate
Representation

Source: Khronos Group

How to Use Vulkan API – Shaders

- Vulkan uses SPIR-V as shader language
  - GLSL is not available out of box

- Cross Compilers already exist, to build SPIR-V out of GLSL and save on porting effort
  - Offline compiler: generates SPIR-V bytecode

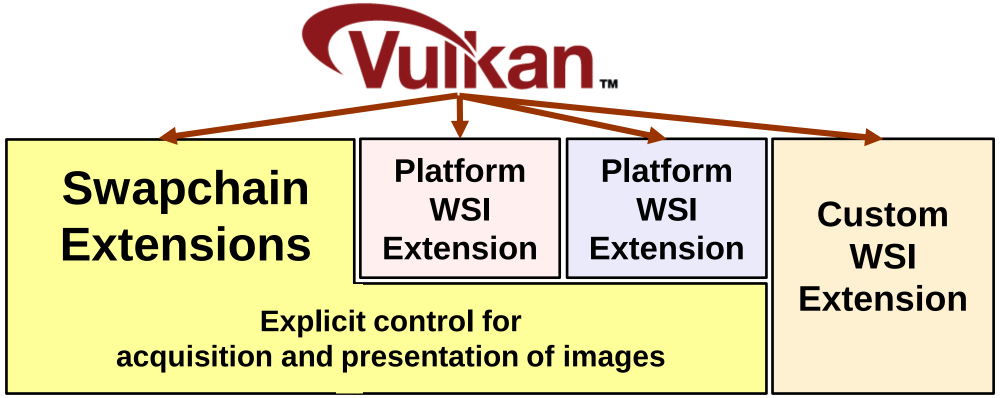
- Smaller SPIR-V to GPU specific instructions set compiler
  - Much smaller footprint than GLSL compiler

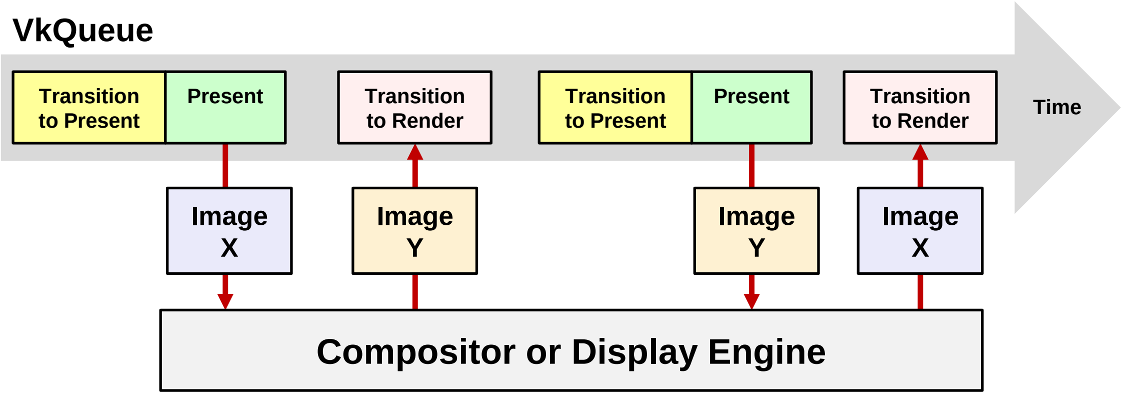
- SPIR-V is well defined:
  - Possible to implement e.g. HLSL to SPIR-V compiler

- SPIR-V to LLVM and back with no loss guaranteed
  - SPIR-V can be optimized with LLVM tools

Vulkan Window System Integration (WSI)

- Explicit control for acquisition and presentation of images
  - Designed to fit the Vulkan API and today's compositing window systems
  - Cleanly separates device creation from window system
- Platform provides an array of persistent presentable images = Vulkan Swapchain
  - Device exposes which queues support presentation
  - Application explicitly controls which image to render and present
- Standardized extensions - unified API for multiple window systems
  - Works across Android, Mir, Windows (Vista and up), Wayland and X (with DRI3)
  - Platforms can extend functionality, define custom WSI stack, or have no display at all





Source: Khronos Group

Vulkan API and Windowing

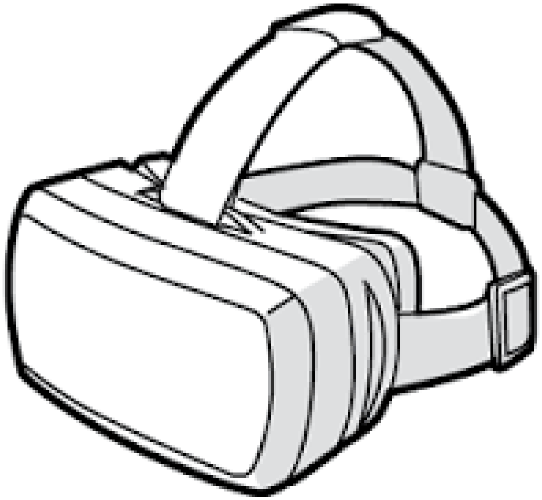
- Core Vulkan has no window system specified
  - Vulkan can run in “console”

- Windowing specified in extension
  - Android, Linux (X11, XLIB, Mir and Wayland) and Windows

- Like any other Vulkan extension, simple to query and enable

Vulkan vs. OpenGL ES and When to Use Which API

- Reduce Render Latency
  - See GDC 2016 presentation “Performance Lessons from Porting Source 2 to Vulkan”
  - https://www.khronos.org/assets/uploads/developers/library/2016-gdc/khronos-vulkan-sessions-part%20ii_gdc_mar16.pdf
    - Latency from Frame End to Beginning went from 3.8ms (DX9) to 0.4ms (Vulkan)

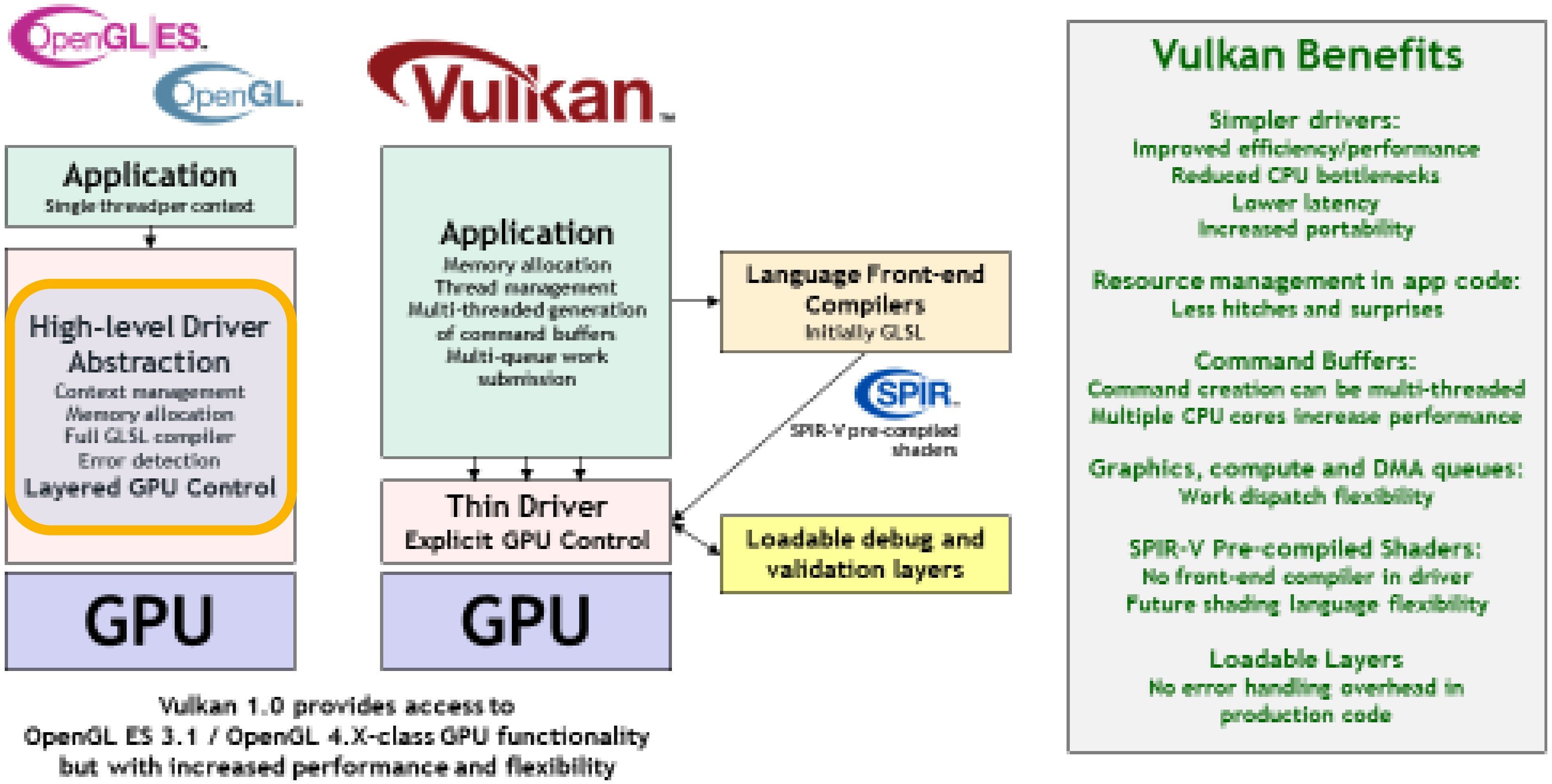


- Low Latency Display – the next killer App
  - Low Latency between GPU and Display is critical for VR and AR Applications

Using Vulkan with i.MX 8

Vulkan explicit GPU control

Application has to implement functions, what driver did before



Application
Single thread per context

High-level Driver Abstraction
Context management
Memory allocation
Full GLSL compiler
Error detection
Layered GPU Control

GPU

Application
Memory allocation
Thread management
Multi-threaded generation of command buffers
Multi-queue work submission

Language Front-end Compilers
Initially GLSL

SPIR-V pre-compiled shaders

Thin Driver
Explicit GPU Control

Loadable debug and validation layers

GPU

Vulkan 1.0 provides access to
OpenGL ES 3.1 / OpenGL 4.X-class GPU functionality
but with increased performance and flexibility

Vulkan Benefits

Simpler drivers:
Improved efficiency/performance
Reduced CPU bottlenecks
Lower latency
Increased portability

Resource management in app code:
Less hitches and surprises

Command Buffers:
Command creation can be multi-threaded
Multiple CPU cores increase performance

Graphics, compute and DMA queues:
Work dispatch flexibility

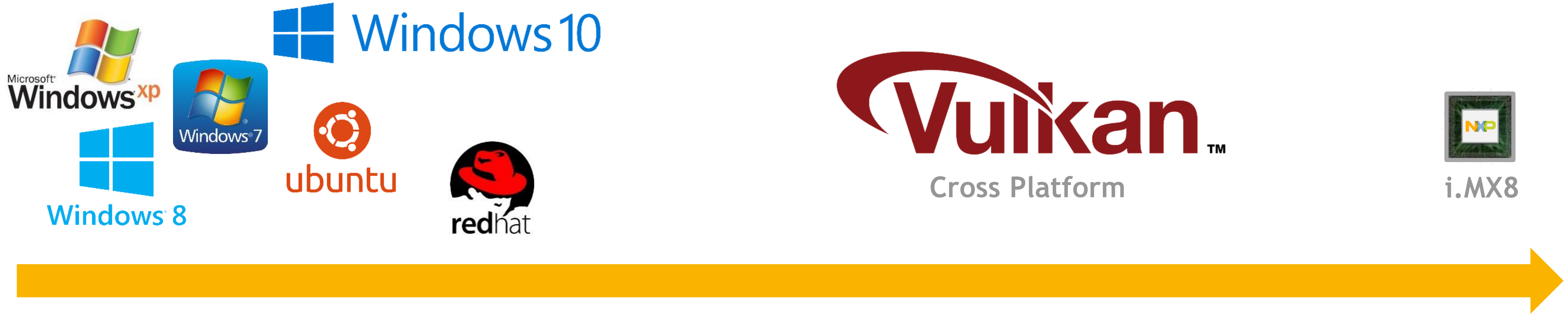
SPIR-V Pre-compiled Shaders:
No front-end compiler in driver
Future shading language flexibility

Loadable Layers
No error handling overhead in production code

Source: Khronos Group

Using Vulkan with i.MX 8

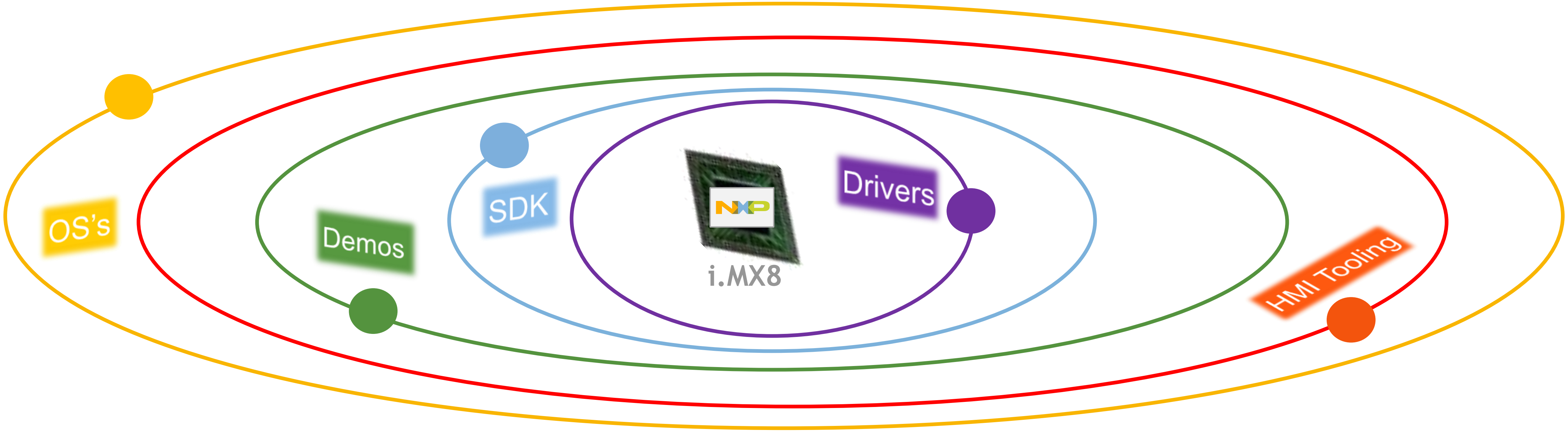
- Development and testing
  - No more embedded or desktop profiles
    - Allows better simulation and testing on desktop



Optimizing i.MX 8 with Vulkan

- Optimization
  - Implementation workload
    - Only implement and use Vulkan “fast path” on i.MX 8
      - E.g. Image handling
  - Code Footprint
    - Application only has to implement graphical features, which are needed
      - E.g. Pure 2D instrument cluster does not need a mip-map generator.
- Flexibility
  - Specific driver extensions can be designed and implemented by application programmer

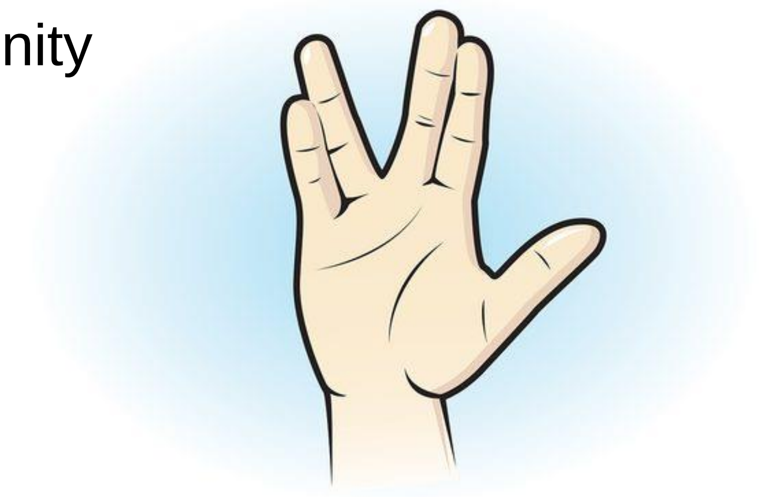
i.MX 8 and Vulkan Enablement



NXP is committed to the education and equipping of Vulkan to our partners

Summary

- Vulkan is a new bare-metal API for graphics and compute
- Vulkan and OpenGL (ES) will coexist
  - Vulkan has momentum in the graphics and compute community
- i.MX 8 will support Vulkan when launched
- NXP is working with OS Vendors, HMI tool providers, and Engine developers on enabling Vulkan support



NXP

SECURE CONNECTIONS
FOR A SMARTER WORLD

ATTRIBUTION STATEMENT