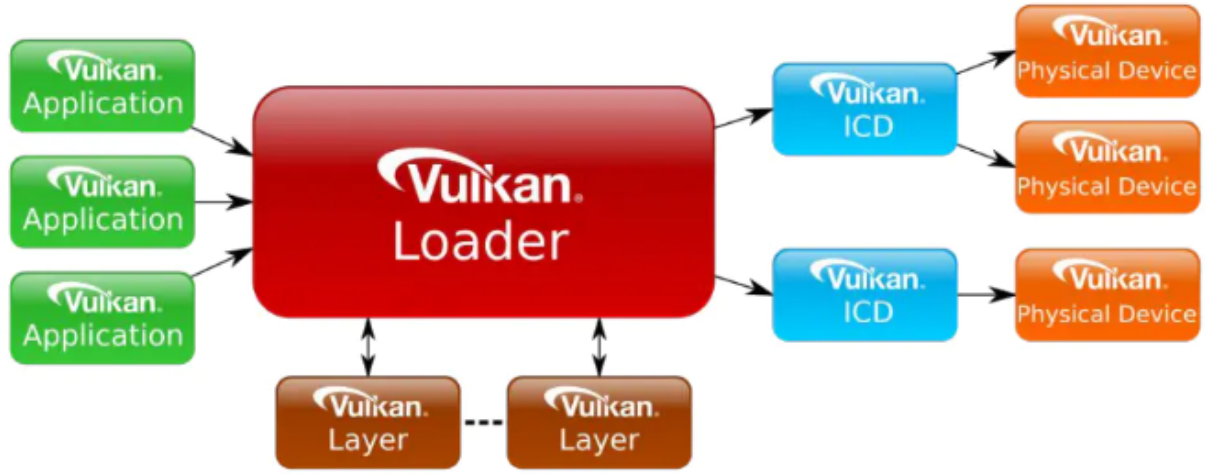
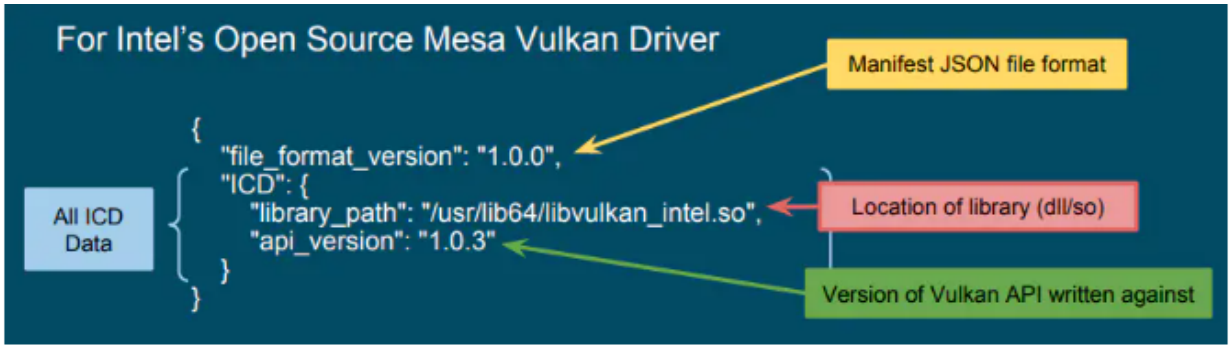
Vulkan Loader



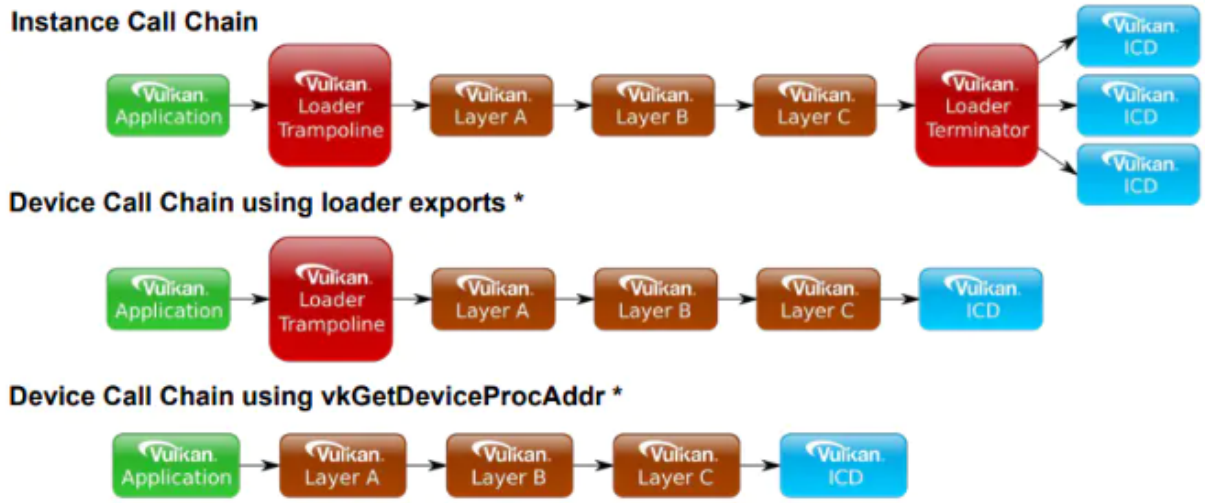
上图给出了vulkan loader在vulkan应用之间的关系，可以看到Vulkan Loader是Vulkan API跟Driver进行交互的过渡层，是其中必不可少的一环；同时，Vulkan Loader还负责不同Vulkan Layer的接入工作，方便了Vulkan根据需要对功能进行扩展。

上图中的ICD指的是Installable Client Driver，这是由硬件厂商提供的一层驱动（用于实现接口的转换，从而保证上层使用相同的接口调用，底层硬件实现的效果是相同的），每个Physical Device（GPU），都有一个与之对应的ICD，Vulkan Driver具有与之相关联的manifest文件（JSON），这个文件的格式给出如下：



Vulkan设计之初是希望对所有设备而言，都走同一套Loader，**但是最终的结果是PC跟Android分别使用了不同的Loader，不过这两套Loader所使用的的接口是一样的。**

GetProcAddress

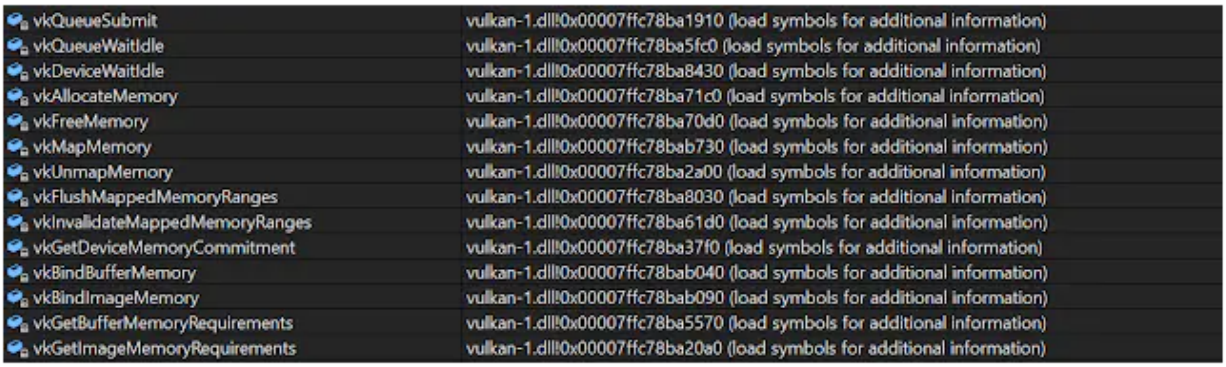


这里给出了Vulkan Call Chain的示意图，这里的Call指的是Vulkan API Call，可以看到，每个API调用的时候，会根据需要一层接一层的添加Layer（比如Validation Layers）对这个API的封装代码；此外，Instance Call需要Vulkan Loader进行ICD Dispatch，而Device Call则直接传导到ICD上，从这个角度来看，后者无疑具有更高性能。

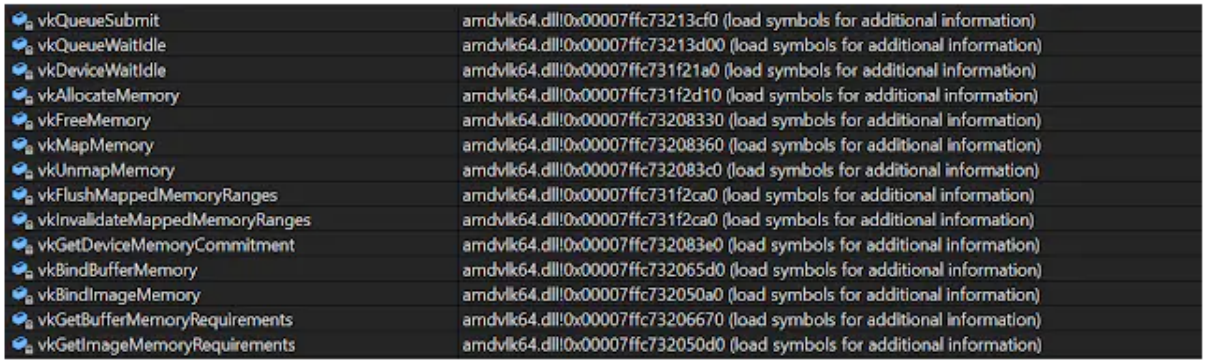
更具体一点，还拿之前的独显跟集显举例子，在一台同时具有独显跟集显的设备上，如果应用最终选定了独显作为渲染的硬件，那么Vulkan Loader就会在这台设备上找到与独显相匹配的ICD，而这个ICD则通过某种方式与GPU以及一些调用函数（call functions，比如vulkan api？）相关联。比如在Windows系统上，我们可以通过GetProcAddress系统函数拿到这些需要调用的API函数的入口，在Linux上则可以通过dlsym函数拿到API函数入口。

实际上，如果我们使用vkGetInstanceProcAddr或者vkGetDeviceProcAddr函数，则可以跳过ICD查找这一步直接获取到Vulkan API的函数入口，从而节省下一定的时间消耗。

下图是通过系统函数获取到API函数入口，可以看到，函数是跟Vulkan Loader相关联的：



而通过vkGetInstanceProcAddr或者vkGetDeviceProcAddr函数，得到的入口则如下图所示，可以看到是直接跟ICD相关联的：



至于vkGetInstanceProcAddr或者vkGetDeviceProcAddr函数两者之间的区别这里也简单介绍下。

vkGetInstanceProcAddr函数返回的接口API会指向转向（dispatch）代码，这段代码会为不同的VkDevice调用不同的API实现，因此会有一定的overhead消耗（参考前面图中的Instance Call Chain），而这个消耗则可以通过vkGetDeviceProcAddr函数（参考前面图中的Device Call Chain）来避免.