GPU 虚拟化

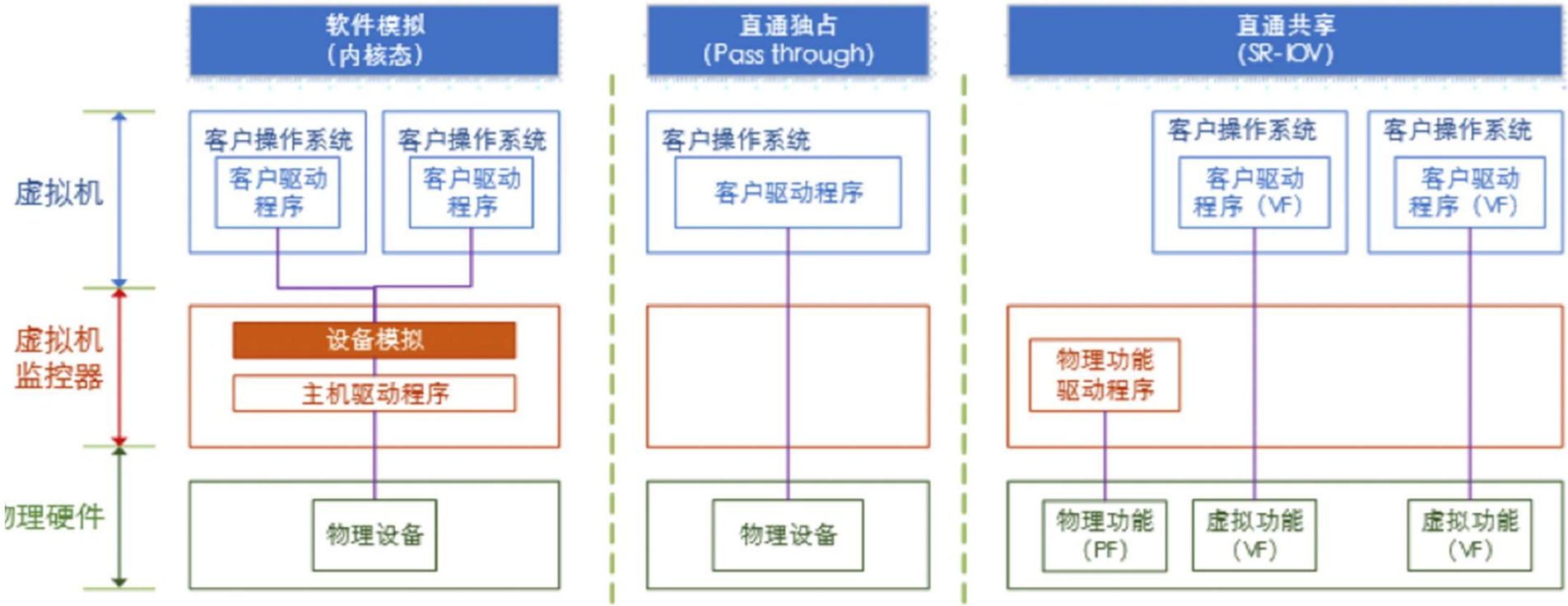
管海兵　汤冬劼

2020.10.12

大纲

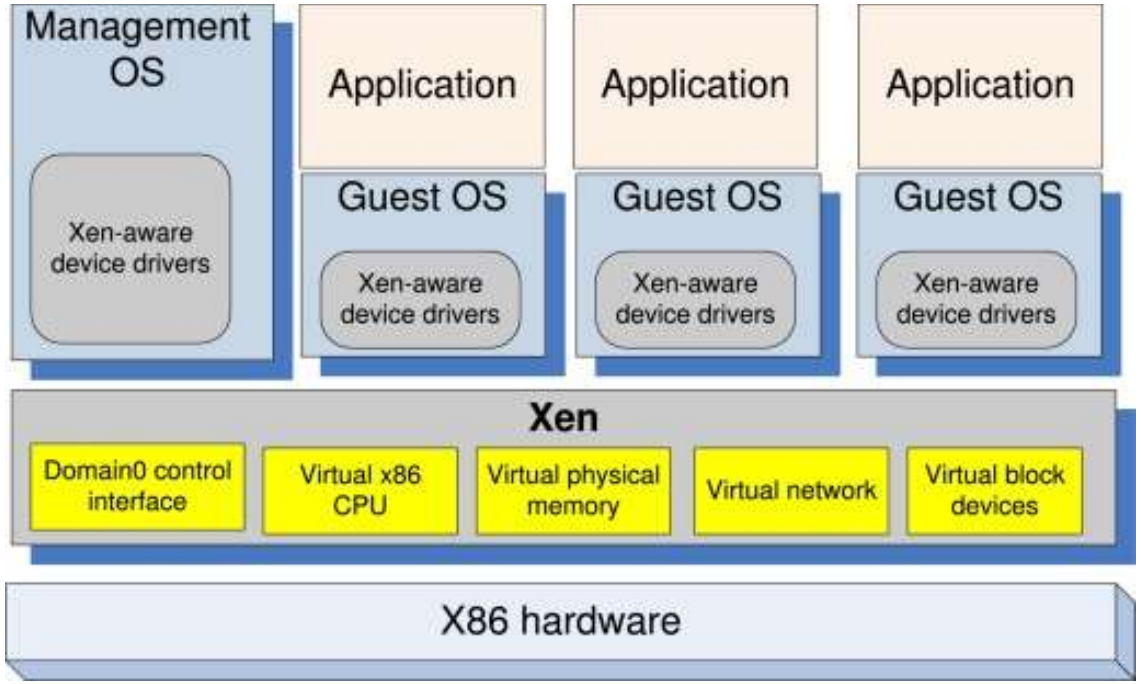
- GPU 虚拟化的起源
- GPU虚拟化的发展
- GPU虚拟化的挑战
- 新型硬件架构的初想

GPU 虚拟化的起源



1. **软件模拟：VMware SVGA 3D software renderer，Citrix GPU accelerator**
2. **直通独占：VMware Virtual Dedicated Graphics Acceleration (vDGA)**
3. **直通共享：SR-IOV，gvirt**

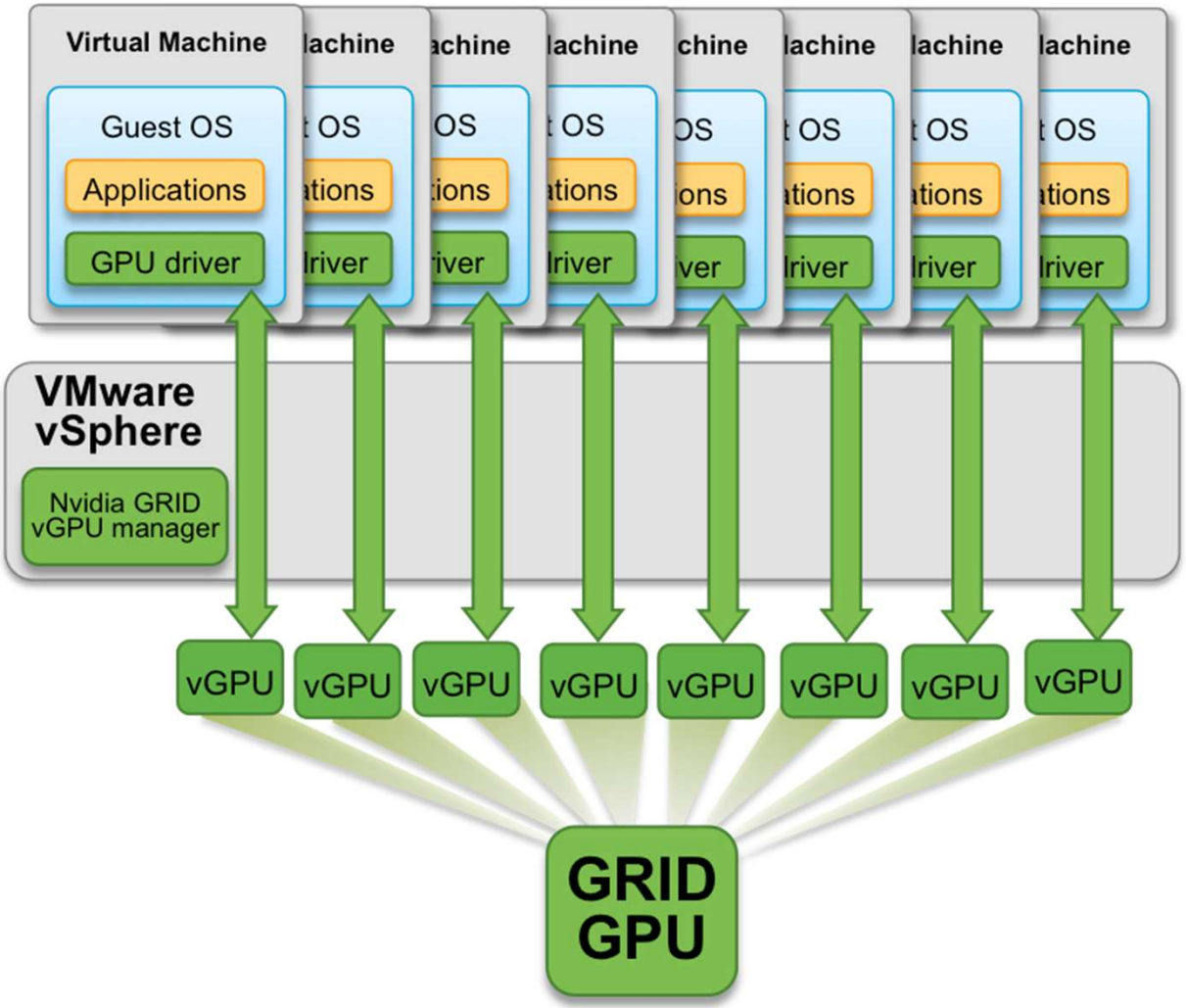
软件模拟(废弃)



1. 做GPU的任务无需真正的硬件
2. 通过xen-aware device 驱动来完成模拟
3. 降低成本，提高CPU使用范围

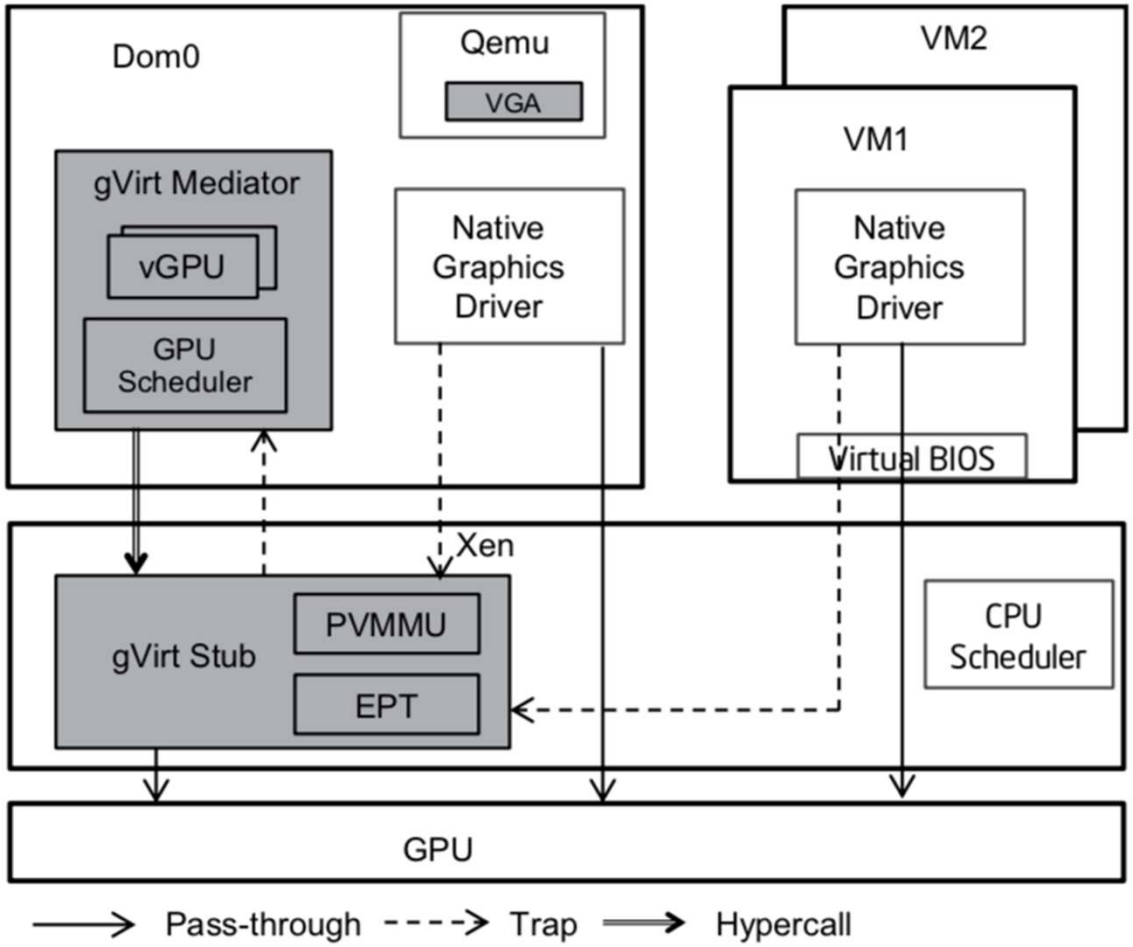
- 软件模拟，无法真正继承GPU的优点
- 指令集更新， xen 驱动无法自适应
- 能耗高，内存高，CPU无法适应
- 性能低，3D模拟无法达到应用的QoS

直通独占(废弃)



- 允许一个虚拟机直接独占一个GPU硬件
- 直接可以享受硬件加速
- 无法提高硬件利用率
- 安全隐患

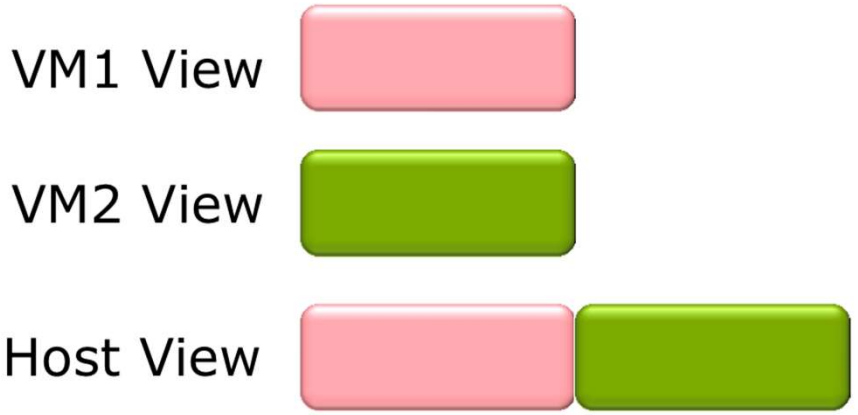
直通共享（共享GPU）



- **设置了一个gvirt stub，将所有的GPU请求送入 gvirt stub**
- **通过pvmmu和ept，对一些指令和数据进行筛选，保证安全。**
- **gvirt mediator管理vGPU之间的切换**
- **GPU scheduler负责调度。**

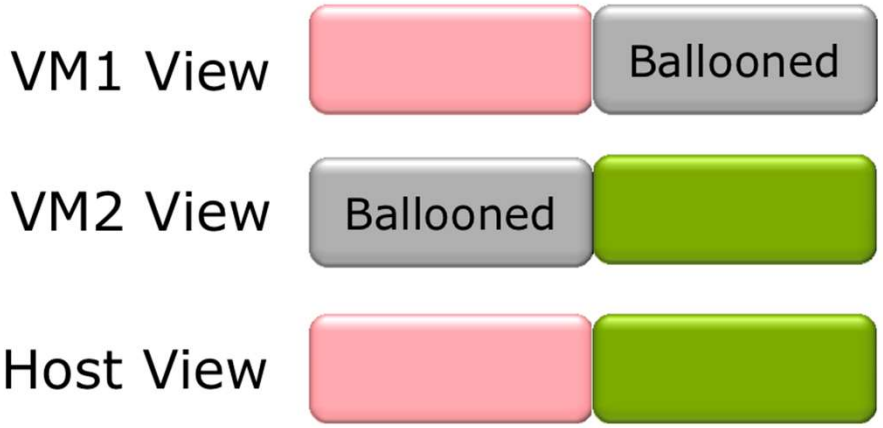
gVirt（ATC14)

直通共享（共享内存）



(虚拟机的角度)

- 少地址空间
- 每个地址都从0开始



(虚拟机的角度)

- 正常地址空间
- 每个地址都从原始位开始

直通共享（成就）

- **实验配置**

- 硬件： 第四代Intel® CoreTM Processor

  - 4 CPU cores (2.4Ghz)、8GB system memory、256GB Intel® 520 series SSD

  - Intel® Processor Graphics 、2GB global graphics memory、Multiple 2GB local graphics memory

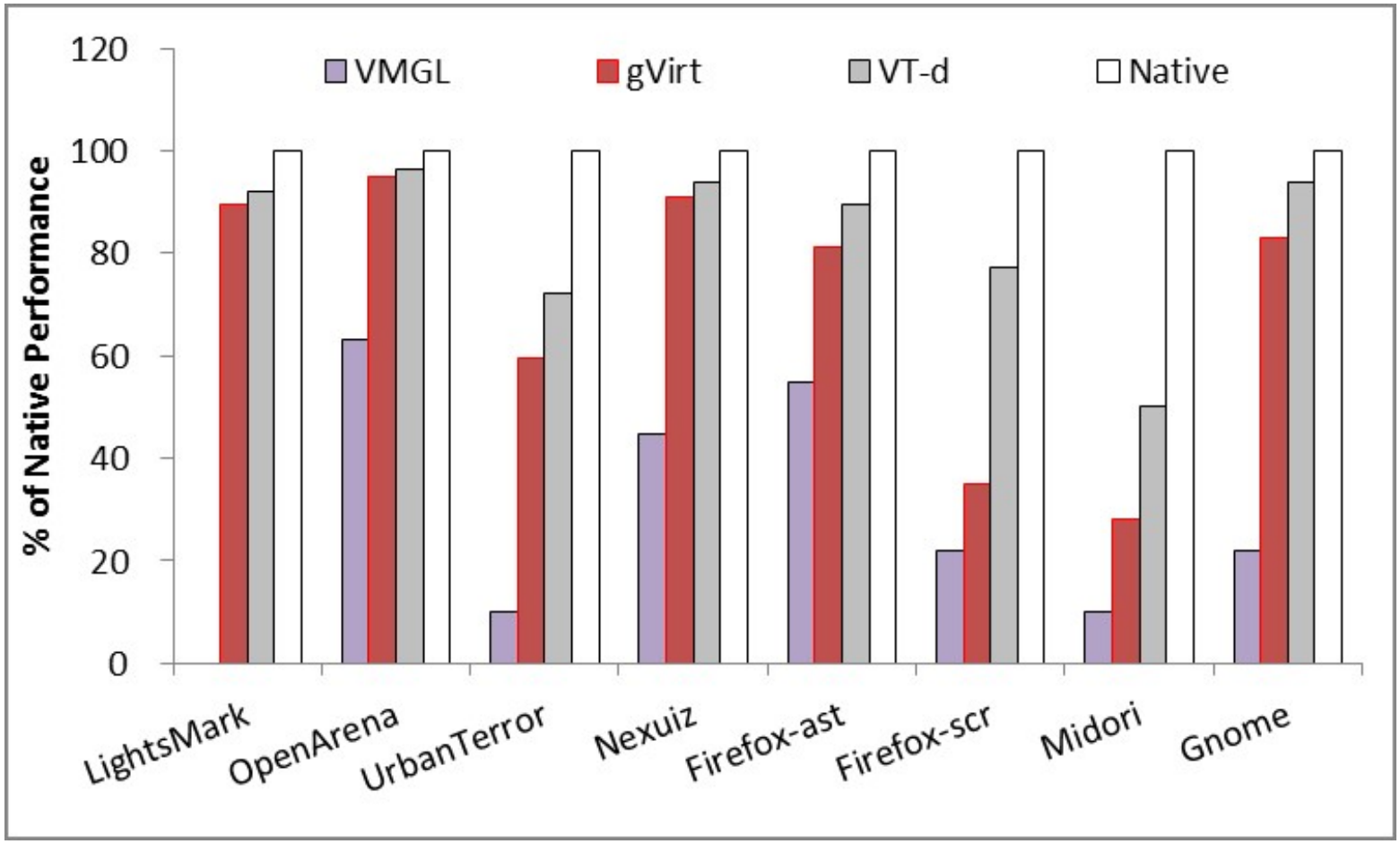
- 软件

  - Dom0/Linux VM: 64bit Ubuntu 12.04 (3.8 kernel)， Windows VM: 64bit Win7 Xen: 4.3

- VM configuration： 4 VCPUs and 2GB system memory

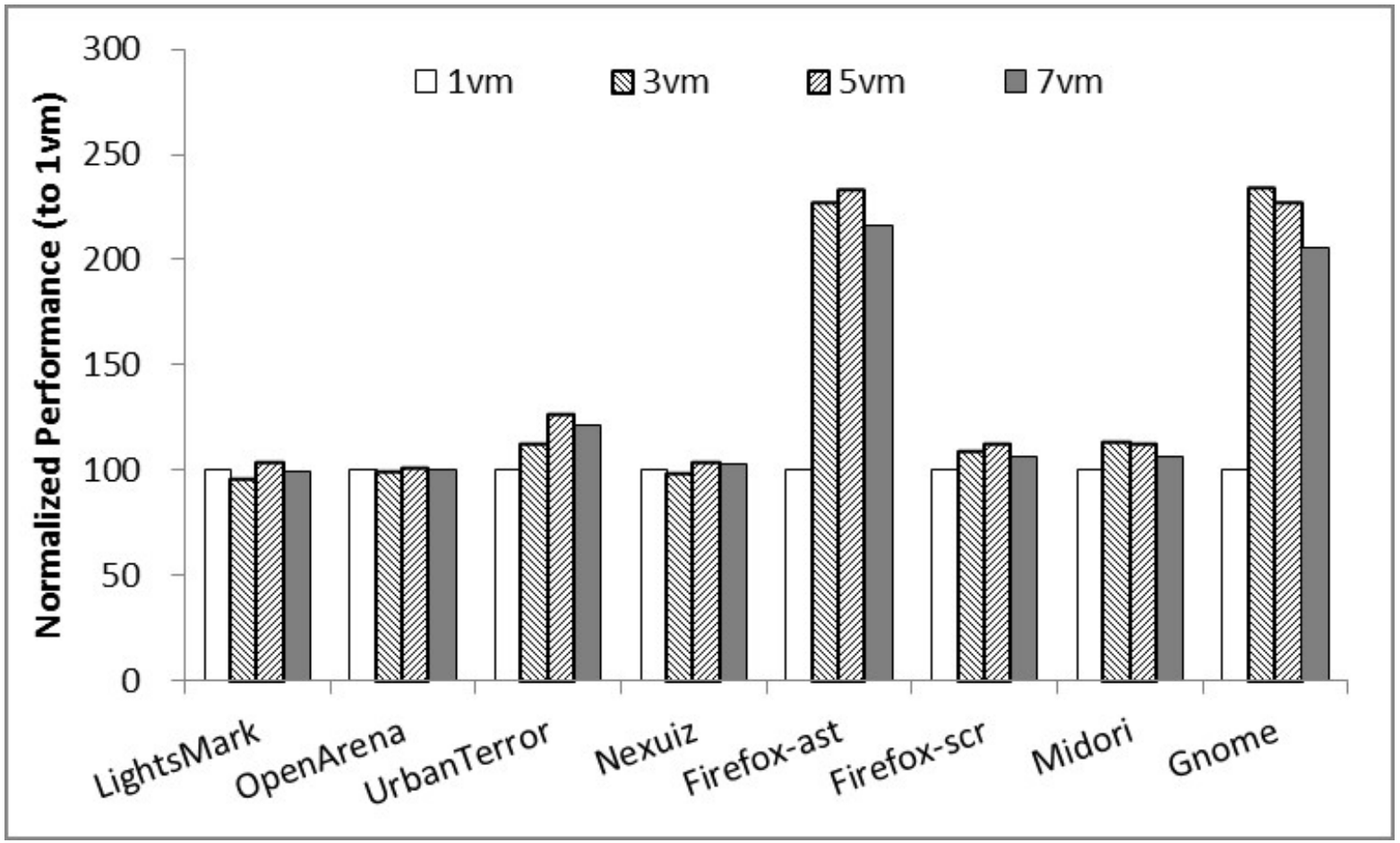
  - Evenly partitioned global graphics memory (e.g. 512MB per VM in a 3- VM configuration)

直通共享（成就）



小于50%的性能损失

直通共享（成就）



可以扩展出7个虚拟机

总结

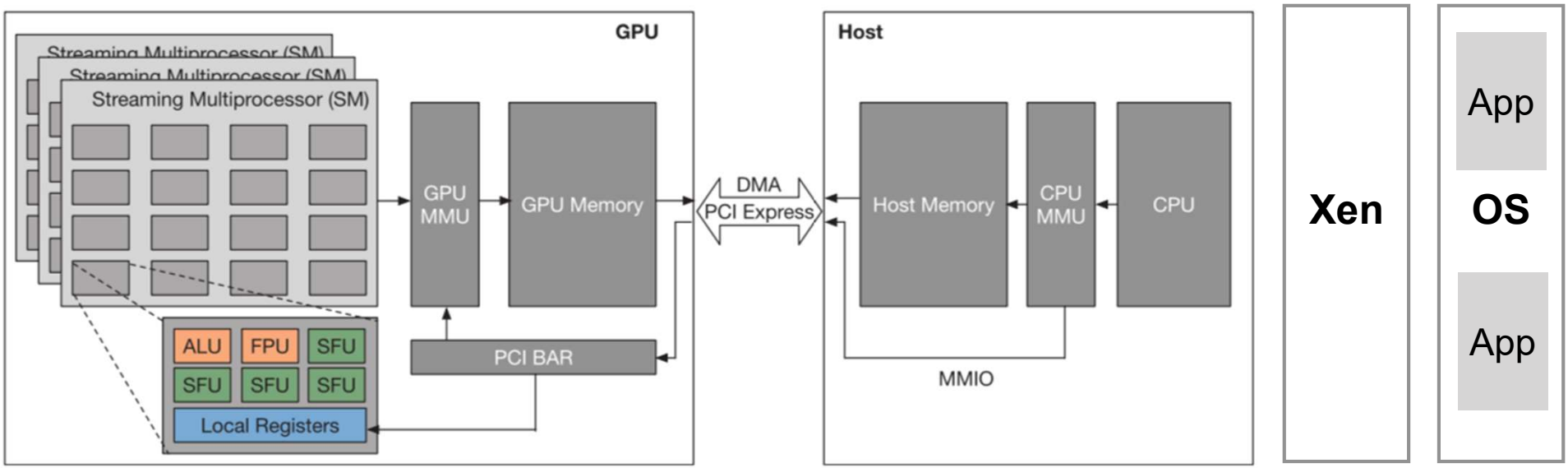
- **上述的方法除了直通共享，其余的已经废弃。**
- **对于硬件有依赖性。**
- **现存直通共享技术需要对GPU内存有访问和修改权限**
- **上述GPU虚拟化秉持一个核心原则：扩大GPU能力。**
- **上述GPU虚拟化前提条件：GPU相较于CPU短缺。**

总结

- 上述的方法除了直通共享，其余的已经废弃。
- 对于硬件有依赖性
- 现存直通共享技术需要对GPU内存有访问和修改权限
- 上述GPU虚拟化秉持一个核心原则：扩大GPU能力。
- 上述GPU虚拟化前提条件：GPU相较于CPU短缺。

GPU虚拟化发展

- **CPU & GPU （独显）**

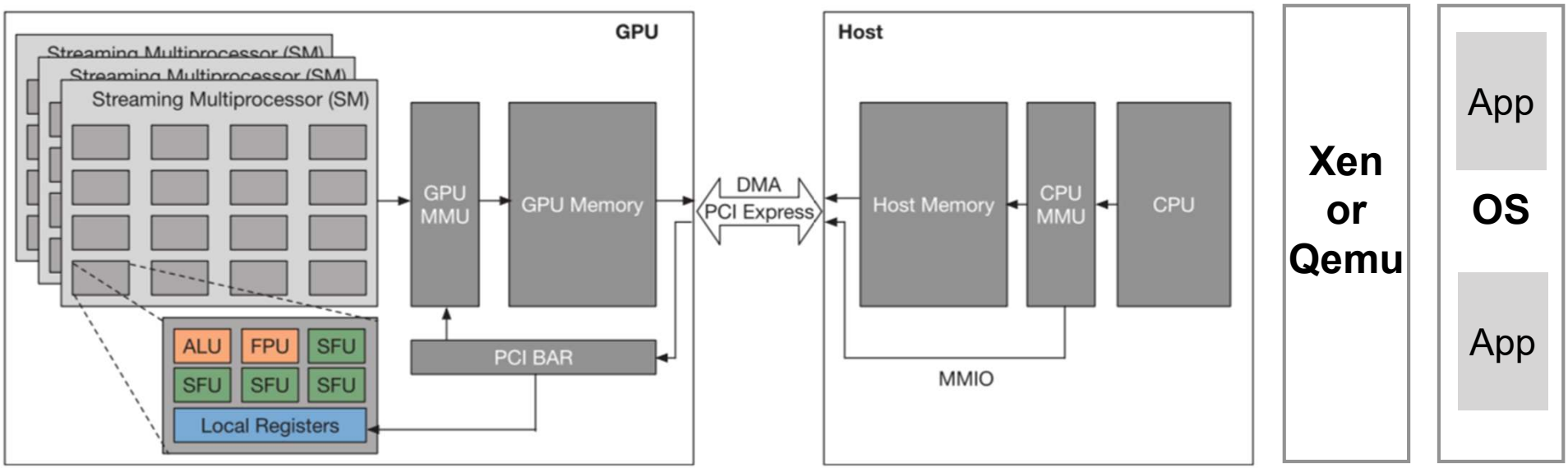


1. **用户只和CPU进行交流**
2. **通过CPU编译将指令通过pcie和dma传输给GPU**

GPU虚拟化发展

gVirt
不适用独显

- **CPU & GPU （独显）**



1. **用户只和CPU进行交流**
2. **通过CPU编译将指令通过pcie和dma传输给GPU**

总结

- 上述的方法除了直通共享，其余的已经废弃。
- 硬件需要跟系统有接触。
- 现存直通共享技术需要对GPU内存有访问和修改权限
- 上述GPU虚拟化秉持一个核心原则：扩大GPU能力。
- 上述GPU虚拟化前提条件：GPU相较于CPU短缺。

GPU虚拟化发展


鲲鹏云手机面临的挑战和机遇
• ARM GPU生态还不够成熟
› A厂只有开源驱动，编码性能无法正常发挥
› N厂的开源驱动性能欠佳，官方对社区无明确支持
– Nouveau开源驱动，N厂正在逐步支持ARM生态，比如将CUDA应用到基于ARM的HPC场景
› 无法支持虚拟化直通、VGPU
› 服务器专业级GPU对OpenGL ES的支持，不像移动设备GPU做了针对性的优化
– 导致画面色彩失真、花屏
• 业界对仿真手机的认知还不统一，也存在客观的困难
› 反模拟器检测
• 是挑战，也是机遇

GPU虚拟化发展

1. **AMD GPU 驱动有官方闭源**
2. **AMD 的mesa驱动不纯净也不全面**
3. **GPU 驱动不责编码译码。**

GPU虚拟化发展

gVirt
不适用独显

GPU虚拟化发展

- **API 转发 (Thin-Client sosp15)**

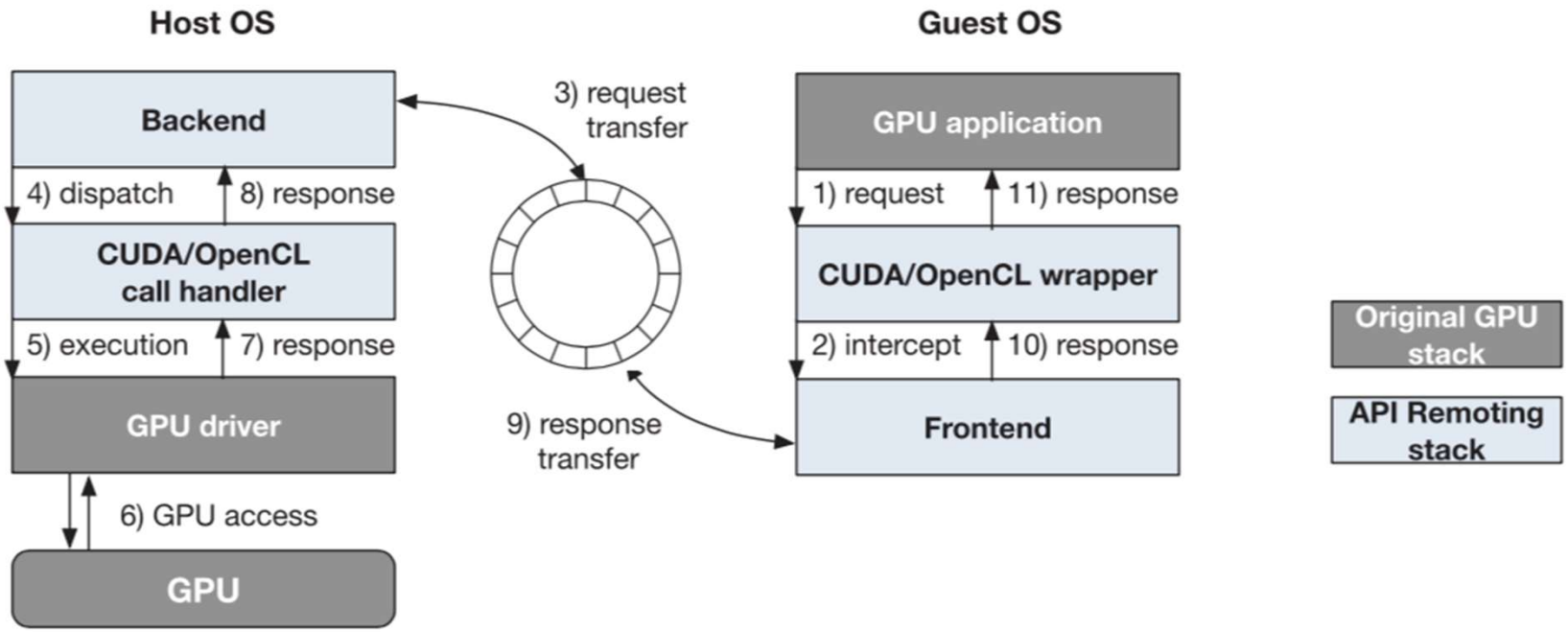
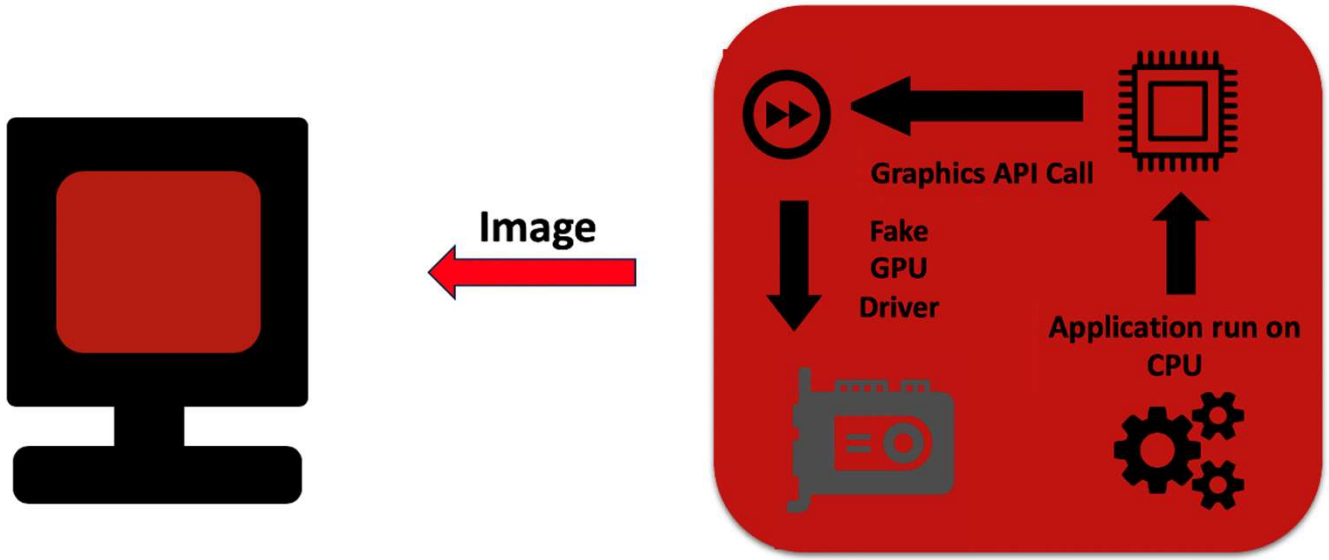


Fig. 2. Architecture of the API remoting approach.

GPU虚拟化发展

- **云游戏 （瘦客户端）**



Xbox Play Anywhere. *https://www.xbox.com/xbox-play-anywhere*

NVIDIA Geforce Now. *https://www.nvidia.com/en-us/geforce-now/*

GPU虚拟化发展

- **远程API 转发 （远程虚拟化）**

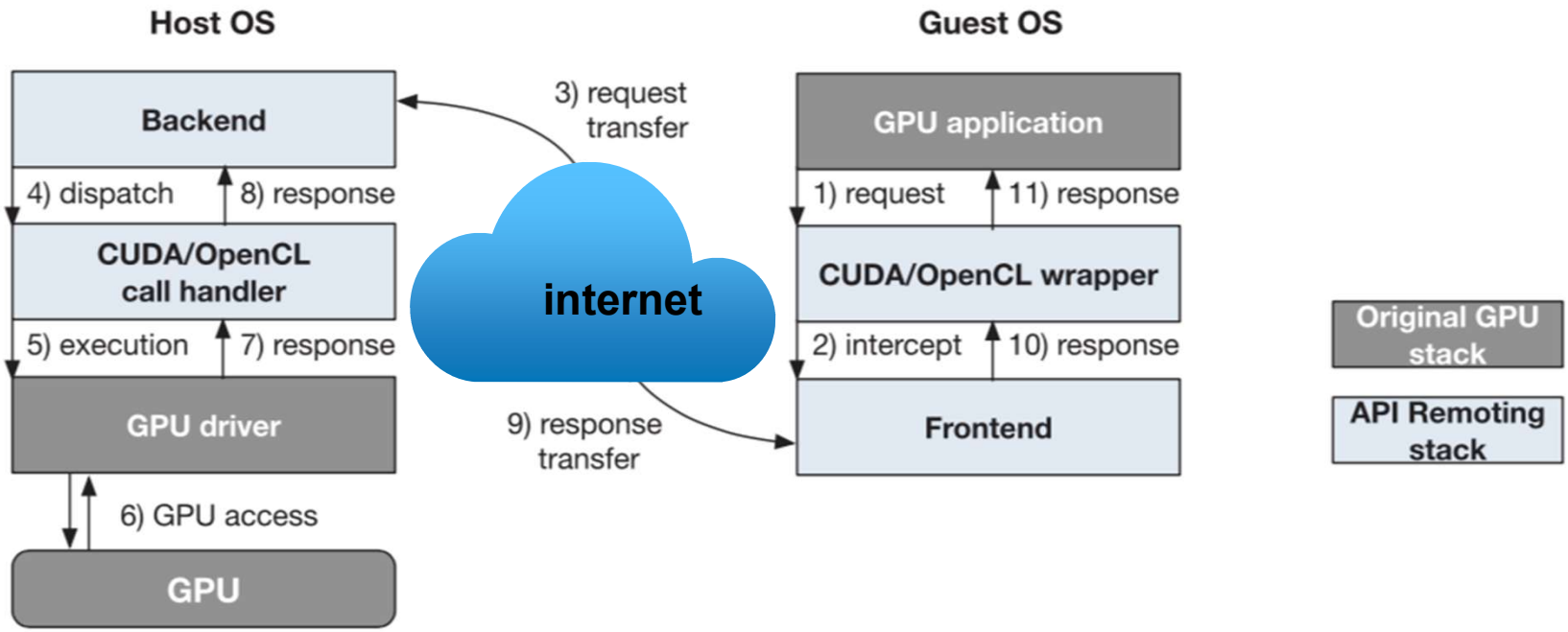
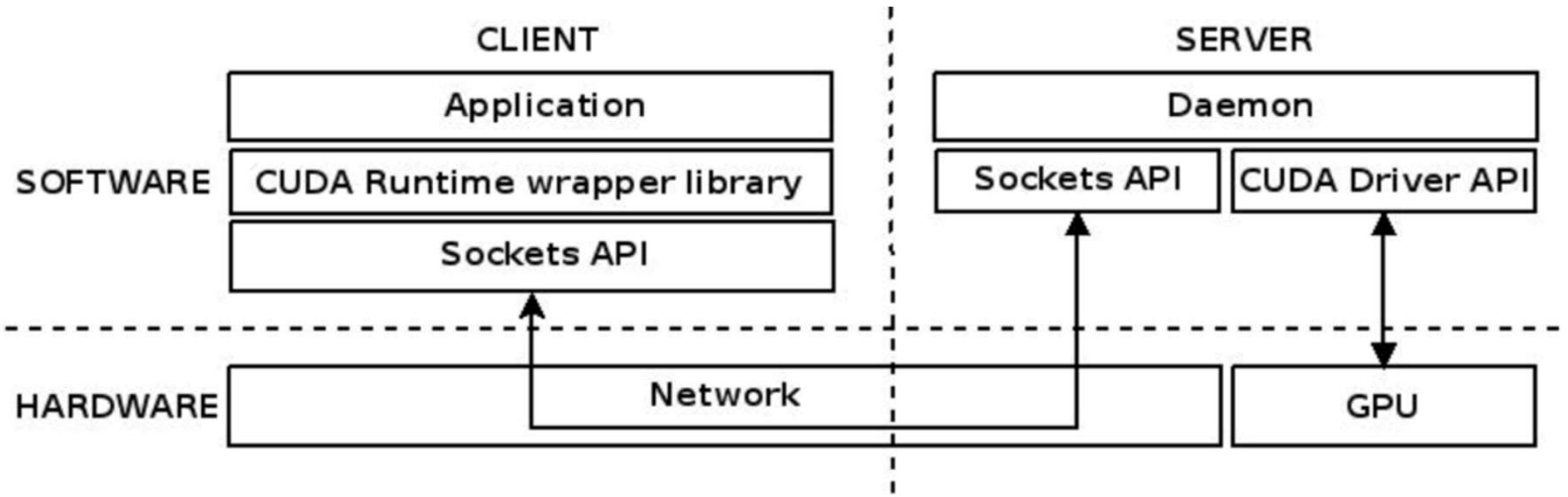


Fig. 2. Architecture of the API remoting approach.

GPU虚拟化发展

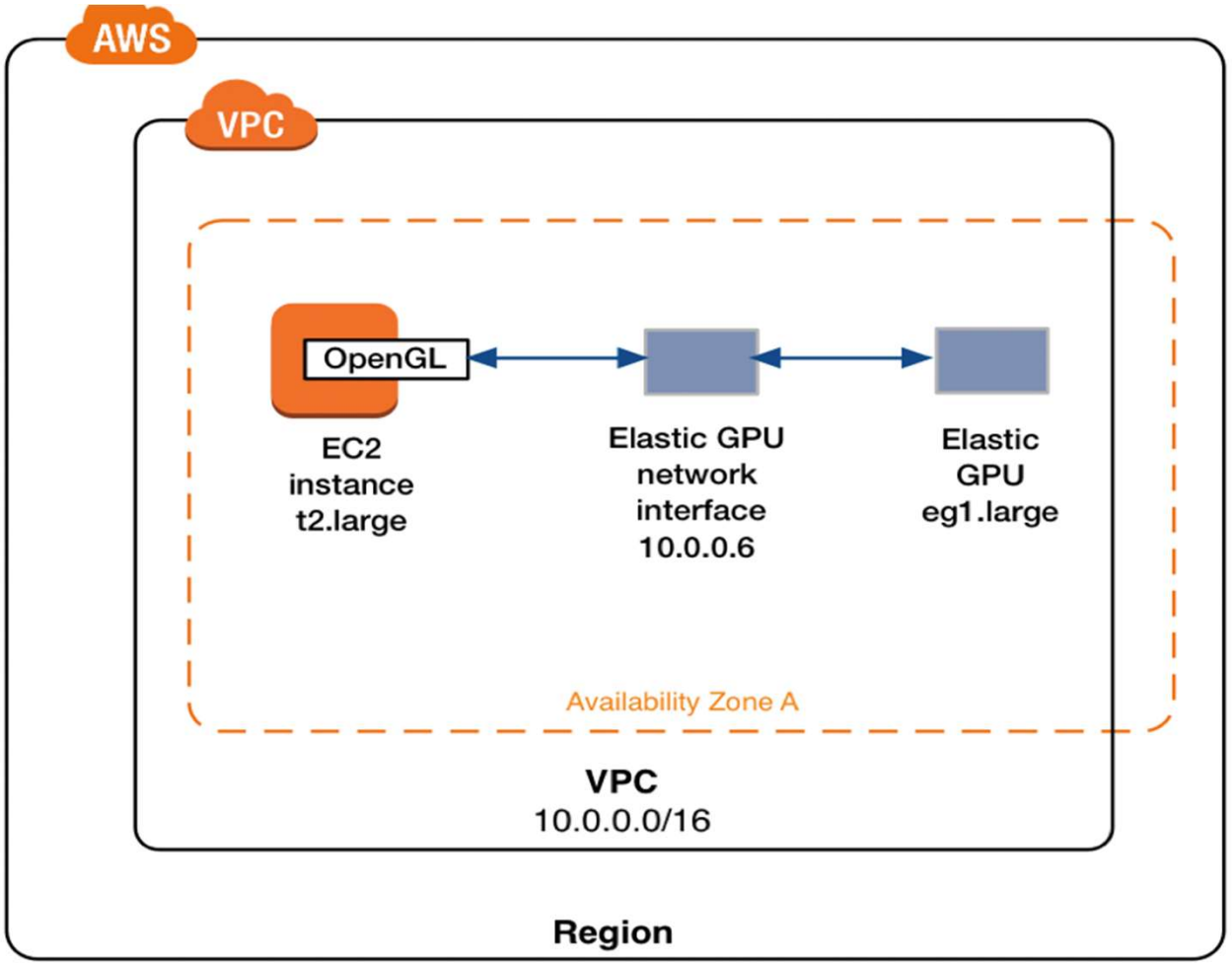
- **远程虚拟化 （rCUDA hpcs10）**



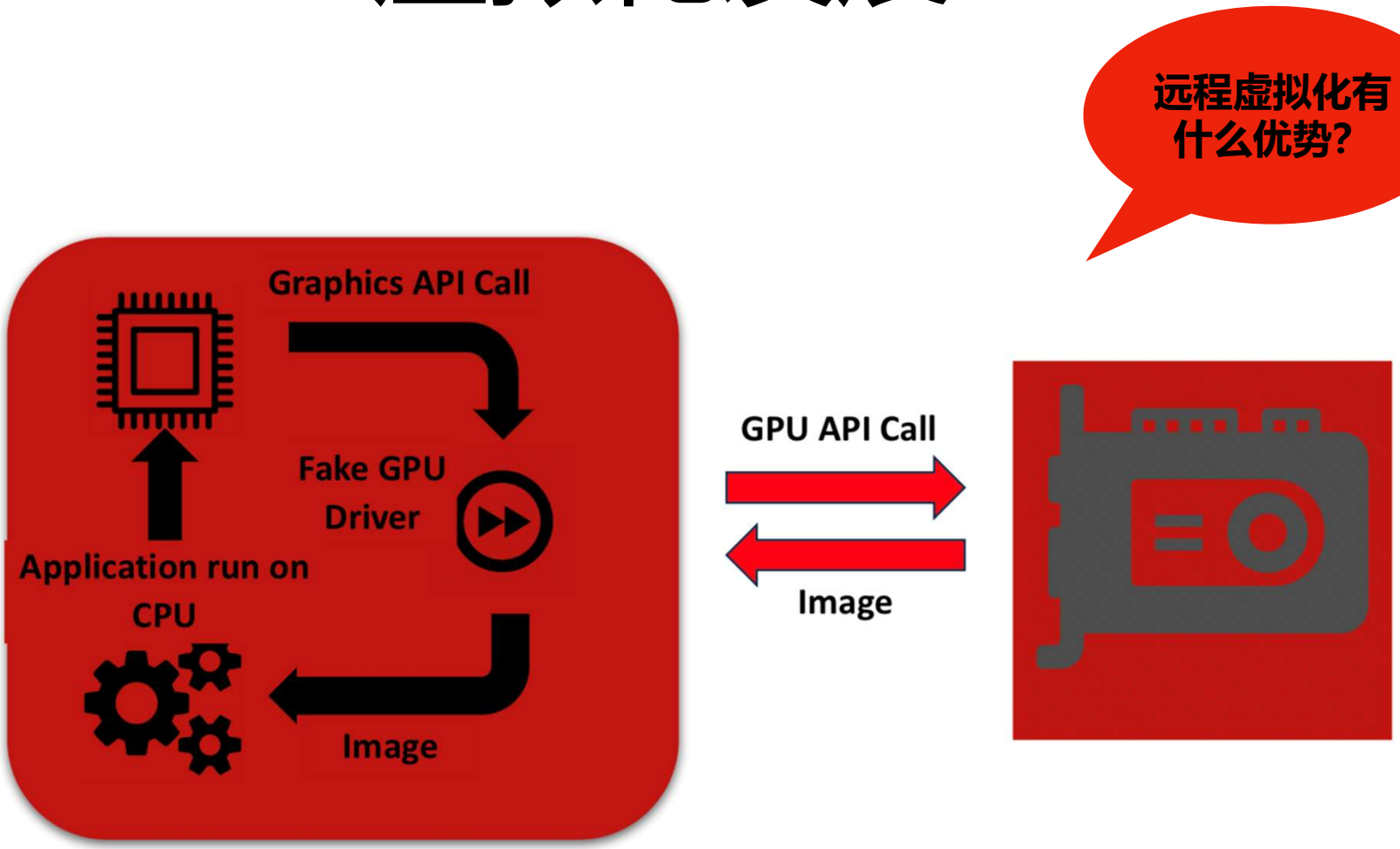
1. **通过wrapper library 截获相关GPU指令**
2. **通过sockets 将其送往远端GPU**
3. **享受到远端硬件加速**
4. **将结果通过socket传回**

GPU虚拟化发展

- **远程虚拟化（Amazon Elastic GPU）**



GPU虚拟化发展


远程虚拟化有什么优势?
Graphics API Call
Fake GPU Driver
Application run on CPU
Image
GPU API Call
Image

GPU虚拟化发展

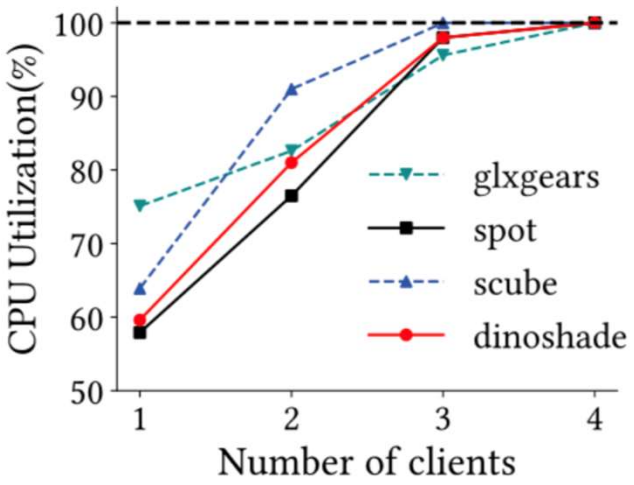
- **远程虚拟化的优势**

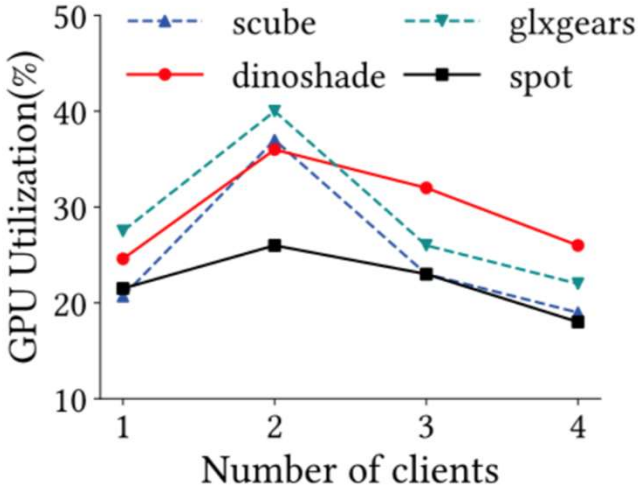
1. **扩大应用范围，不再局限于本地机器，远端机器也可以享用硬件加速（rCUDA）**
2. **相较于瘦客户端， 远程虚拟化可以零成本提高硬件配置 （gremote）**
3. **提高云内的服务器利用率。 （microsoft xcloud）**

GPU虚拟化发展

- **GPU vs CPU**

```
System_call_IOProcessing();
TransCommands_to_GPU();
Context_establish_Initialization();
while true do
    PipelineCreated_FrameSetup();
    GPURendering_BufferReady();
    Present();
end
```



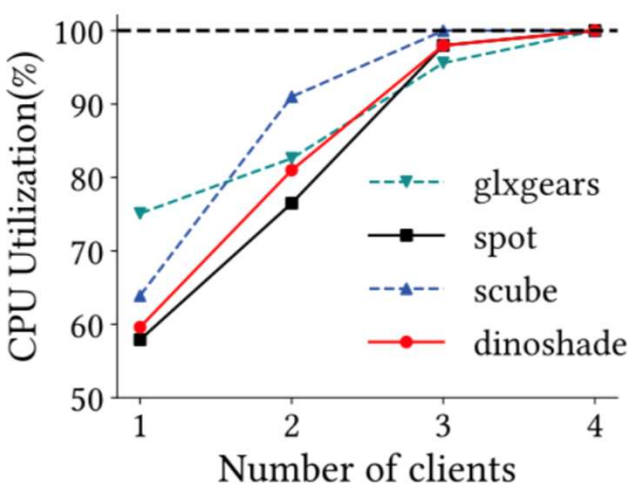


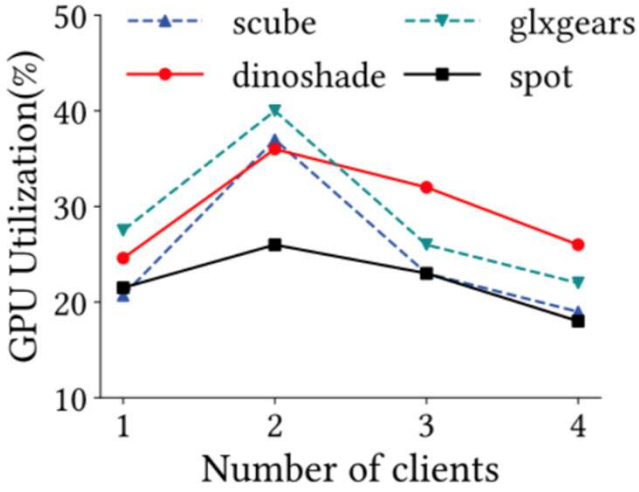
Intel core i5, Nvidia 1060 running on Xen SR-IOV with VM

GPU虚拟化发展

- **GPU vs CPU (CPU bottleneck)**

```
System_call_IOProcessing();
TransCommands_to_GPU();
Context_establish_Initialization();
while true do
    PipelineCreated_FrameSetup();
    GPURendering_BufferReady();
    Present();
end
```





CPU and GPU usage of different clients under the intra-cloud environment connected through Ethernet

总结

- 上述的方法除了直通共享，其余的已经废弃。
- 硬件需要跟系统有一定接触。
- 现存直通共享技术需要对GPU内存有访问和修改权限
- 上述GPU虚拟化秉持一个核心原则：扩大GPU能力。
- 上述GPU虚拟化前提条件：GPU相较于CPU短缺。

总结

传统虚拟化不再适用于当代硬件

- 上述的方法除了直通共享，其余的已经废弃。
- 硬件需要跟系统有一定接触。
- 现存直通共享技术需要对GPU内存有访问和修改权限
- 上述GPU虚拟化秉持一个核心原则：扩大GPU能力。
- 上述GPU虚拟化前提条件：GPU相较于CPU短缺。

GPU虚拟化发展

- **解决CPU 短缺的方案**

  - ***gRemote (HPDC'20) expands CPU resources from only server-side to both sides***

  - ***DroidCloud (MM'20) replace VM with container***

  - ***Pliant (HPCA'19) lowers data precision to save CPU resources***

GPU虚拟化发展

- **gRemote (CPU bottleneck solutions)**

```
glXChooseFBConfig(dpy = 0xcce9e0, screen = 0, attribList..., );
glXGetVisualFromFBConfig(dpy = 0xcce9e0, config = 0xd34b20);
glXCreateContext(dpy = 0xcce9e0, vis = &visual = 0xcda350, visualid = 36...);
glXMakeCurrent(dpy = 0xcce9e0, drawable = 81788930, ctx = 0xdb67f8);
glViewport(x = 0, y = 0, width = 2560, height = 1440);
glScissor(x = 0, y = 0, width = 2560, height = 1440);
glXDestroyContext(dpy = 0xcce9e0, ctx = 0xdb67f8);
glXCreateContextAttribsARB(dpy = 0xcce9e0, config = 0xd34b20, share_context
  =NULL...);
glXMakeCurrent(dpy = 0xcce9e0, drawable = 81788930, ctx = 0xee7648);
glXSwapIntervalEXT(dpy = 0xcce9e0, drawable = 81788930, interval = 0);
glDepthFunc(func = GL_LEQUAL);
glEnable(cap = GL_DEPTH_TEST);
glViewport(x = 0, y = 0, width = 2560, height = 1440);
...
glGenTextures();
```

Commands after fake GPU driver

GPU虚拟化发展

- **gRemote (CPU bottleneck solutions)**

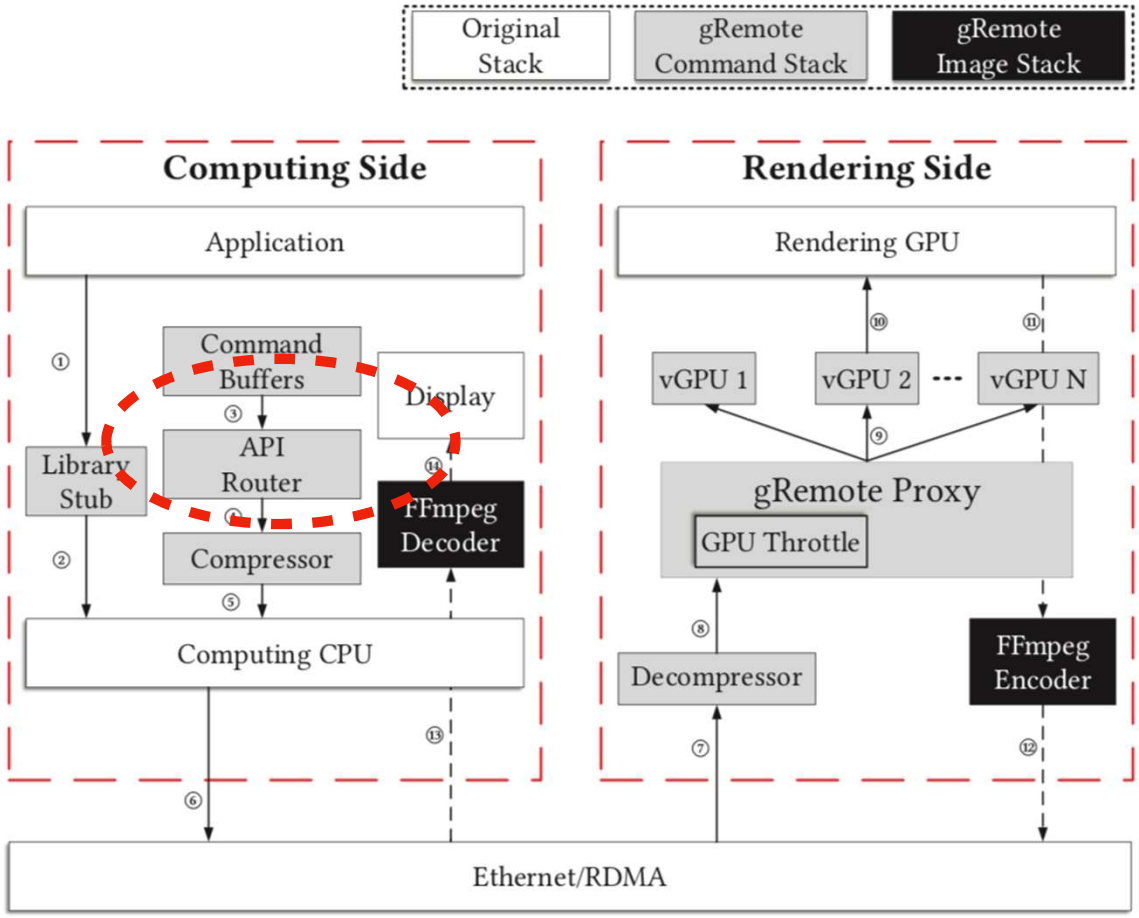


Fig. 3. **The detailed architecture of gRemote**

GPU虚拟化发展

- **gRemote (CPU bottleneck solutions)**

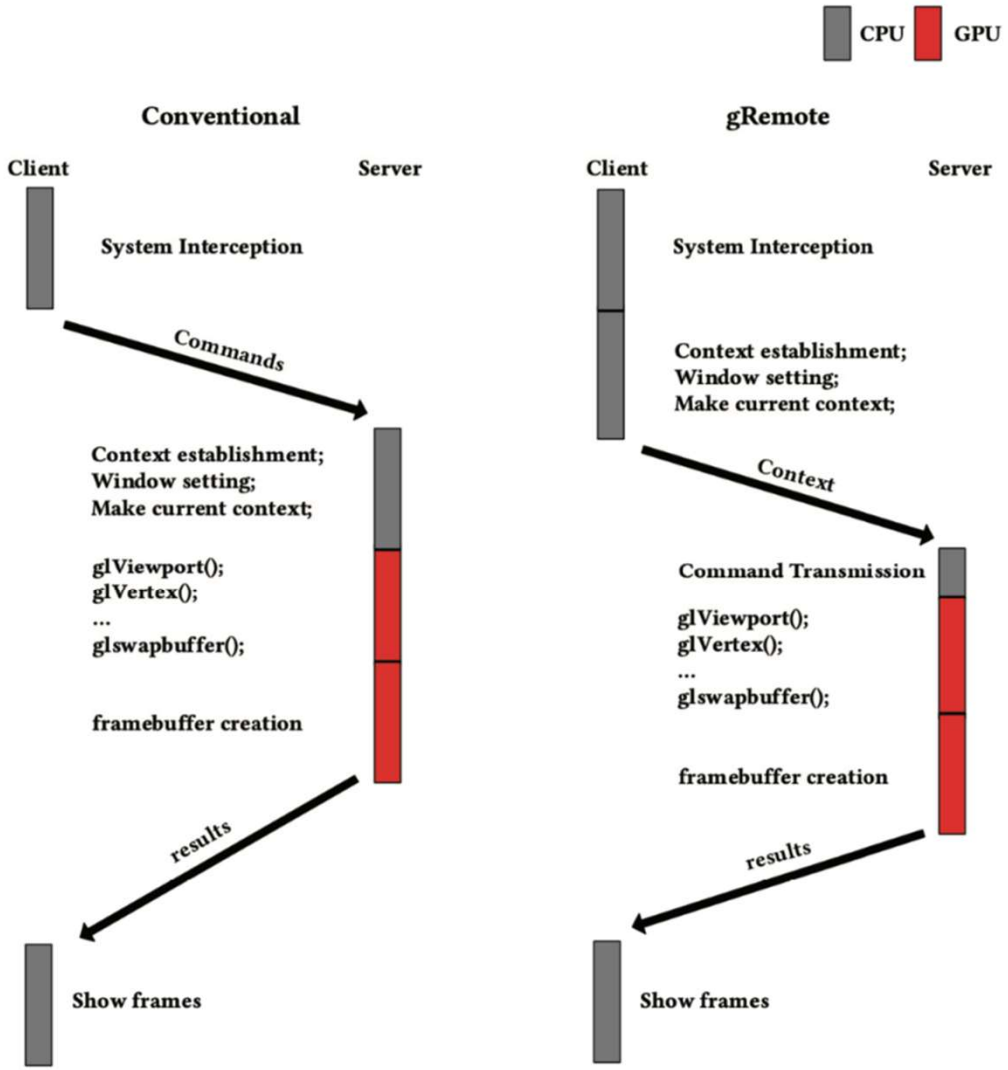
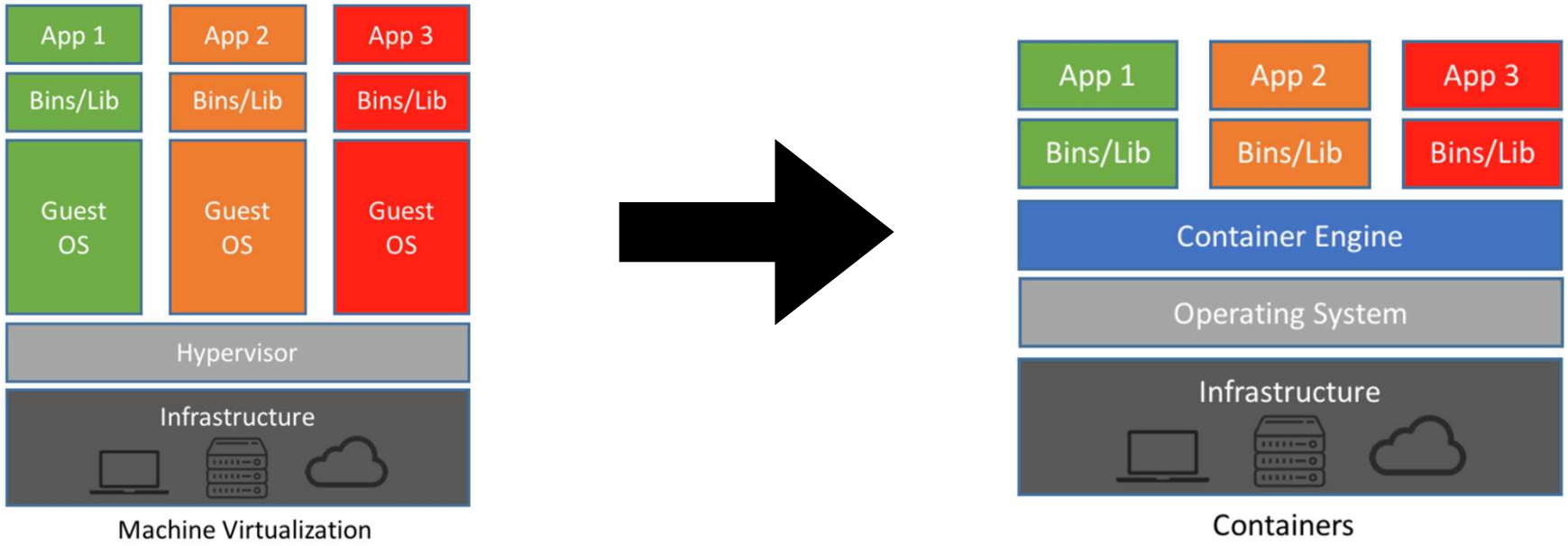


Fig. 4. **Workload division in gRemote**

GPU虚拟化发展

- **DroidCloud (CPU bottleneck solutions)**



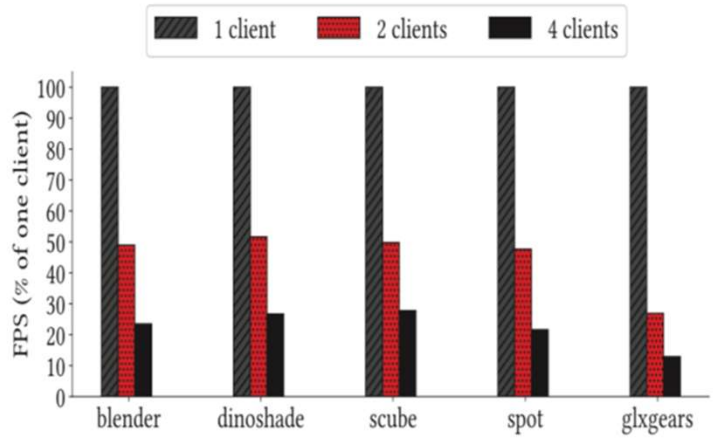
- **隔离单元从虚拟机变成容器**
- **隔离单元变得很轻量化从而节省硬件资源**
- **GPU和系统交互有限，因此guest os功能冗余**

GPU虚拟化挑战

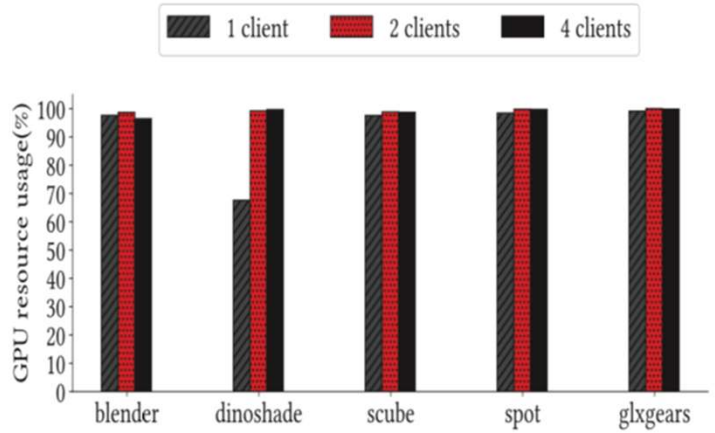
- **GPU资源共享与隔离**
- **远程虚拟化网络带宽问题**
- **GPU的迁移问题**

GPU虚拟化挑战

- **GPU资源共享与隔离 (motivation)**



(a) Performance variation for different applications

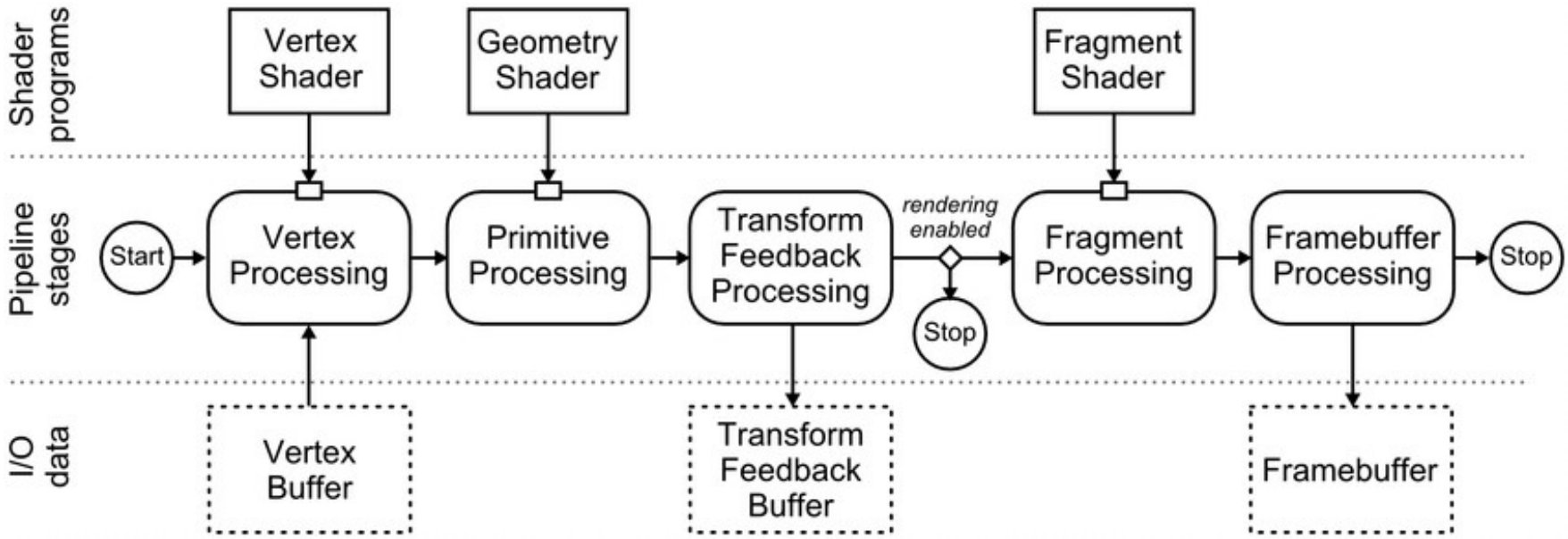


(b) GPU resource ratio taken by different applications

Intel core i5, Nvidia 1060 on the bare metal

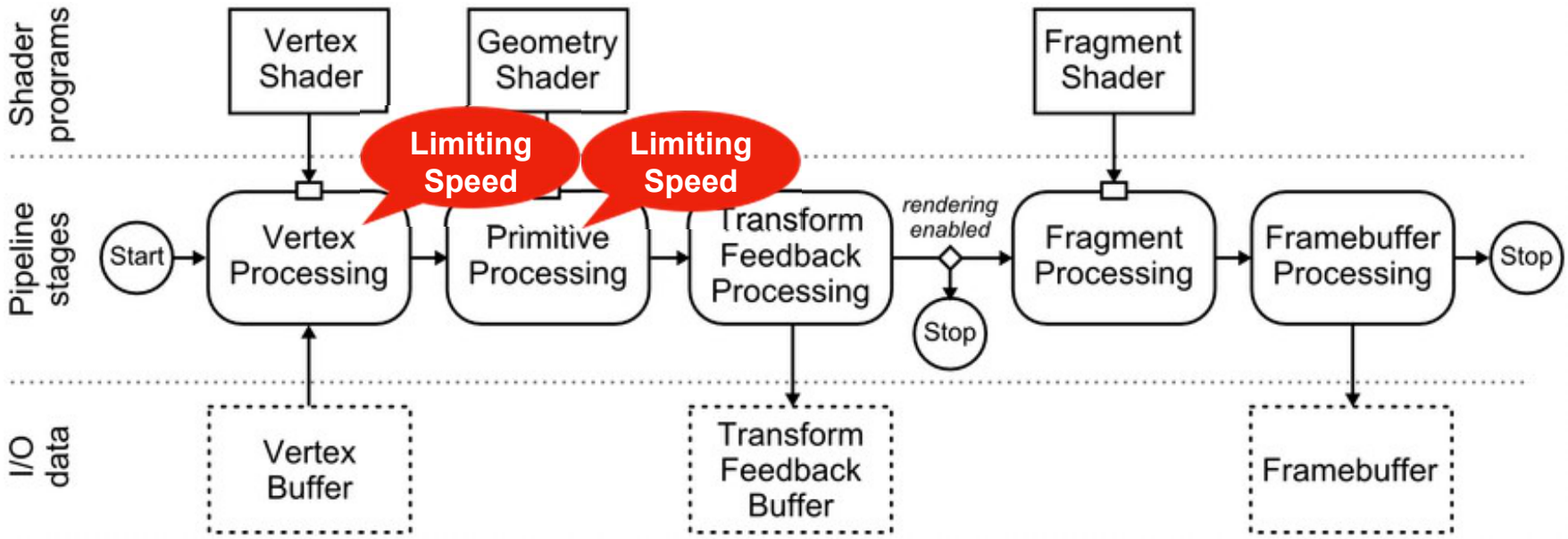
GPU虚拟化挑战

- **GPU throttle (gRemote)**



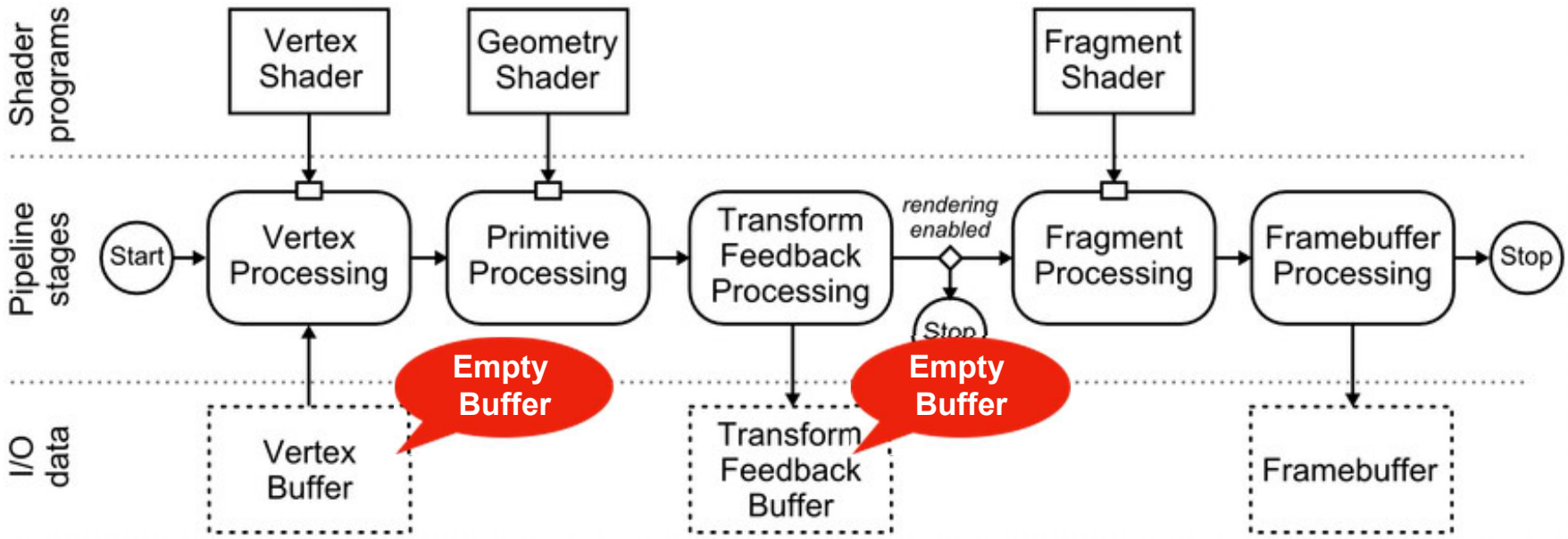
GPU虚拟化挑战

- **GPU throttle (gRemote)**



GPU虚拟化挑战

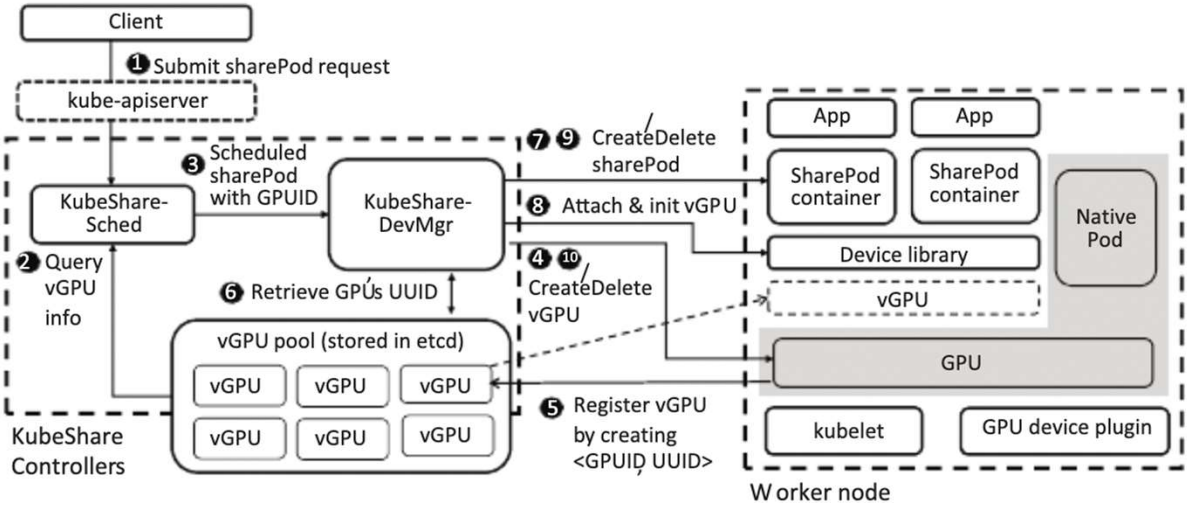
- **GPU throttle (gRemote)**



GPU虚拟化挑战

- **Kubershare**

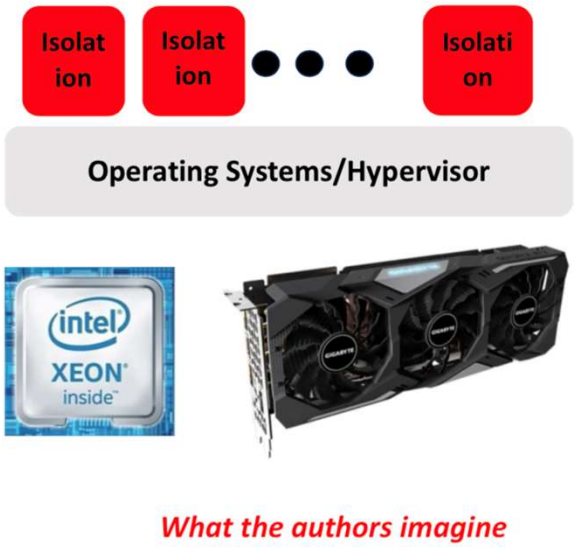
```
apiVersion: kubeshare/v1
kind: SharePod
metadata:
  name: pod1
  annotations:
    "kubeshare/gpu_request": "0.5"
    "kubeshare/gpu_limit": "0.8"
    "kubeshare/gpu_mem": "0.5"
    "kubeshare/sched_affinity": "green"
    "kubeshare/sched_anti-affinity": "red"
    "kubeshare/sched_exclusion": "blue"
    "kubeshare/GPUID": "xyz"
spec: #PodSpec
  nodeName: node1
  containers:
  - name: example
    image: nvidia/cuda:9.0-base
    command: ["nvidia-smi", "-L"]
```



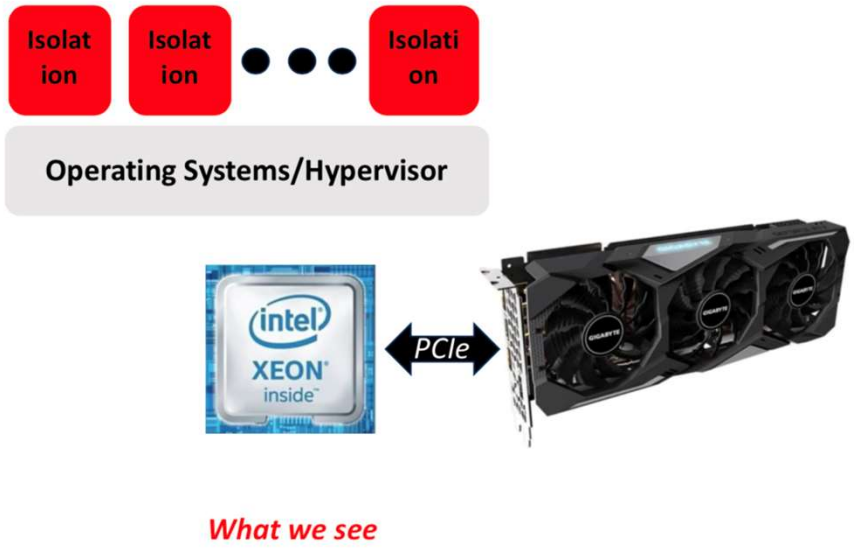
cuMemAlloc, cuArrayCreate, cuLaunchKernel, cuLaunchGrid

GPU虚拟化挑战

- **资源共享瓶颈**



What the authors imagine



What we see

1. **所有方案间接作用于GPU，有overhead.**
2. **功能有限。**
3. **增加CPU的负担**

GPU虚拟化挑战

- **网络带宽消耗（motivation）**

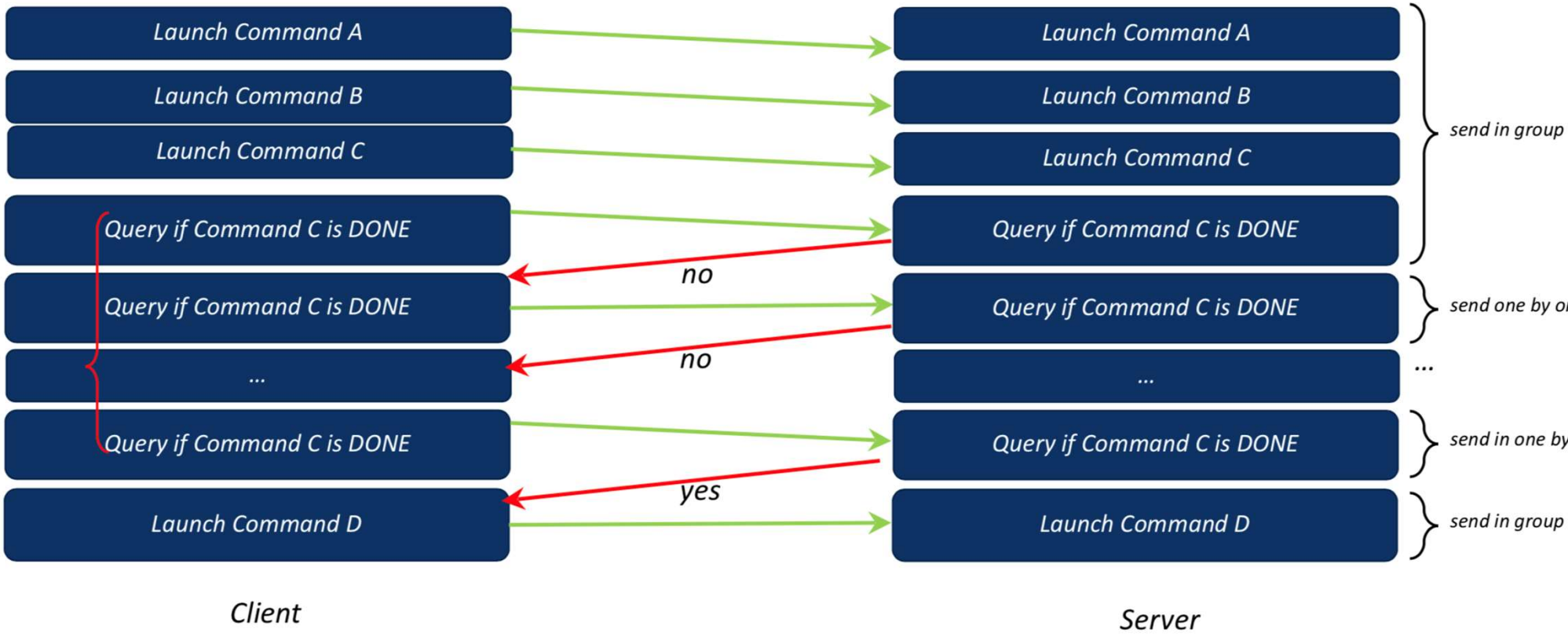
	FPS	Bandwidth (MiB/s)
blender	56	39.6
dinoshade	55	28.6
scube	60	34.2
spot	61	37.8
glxgears	58	36.5

PS. One command in google earth will bring more than 10 MB data.

glVertexAttribPointer in Uginine Heaven will bring nearly 100 MB data.

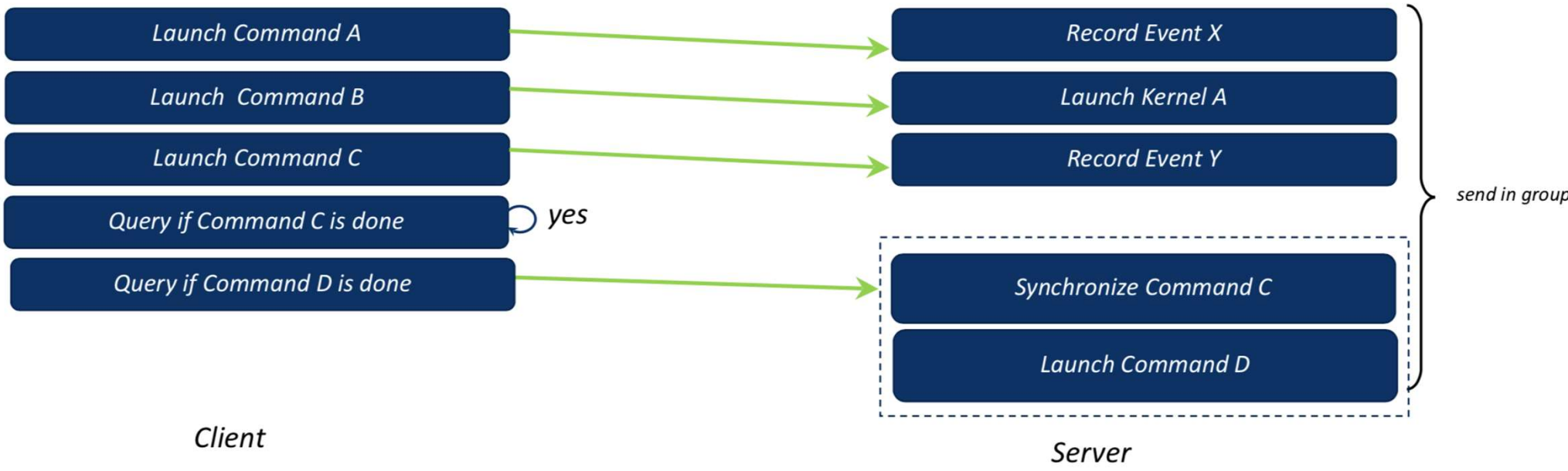
GPU虚拟化挑战

- **gremote （CCO）**



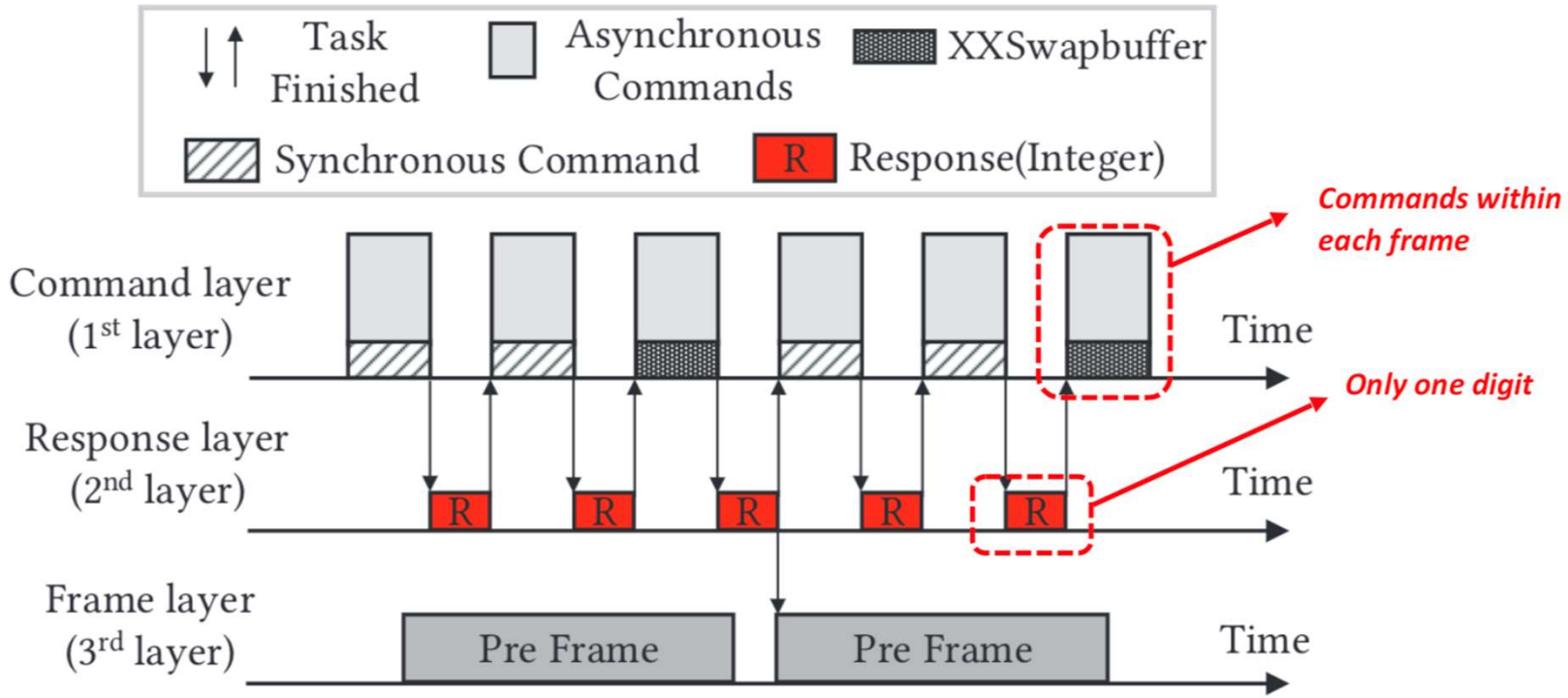
GPU虚拟化挑战

- **gremote （CCO）**



GPU虚拟化挑战

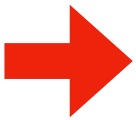
- **gremote （CCO）**



GPU虚拟化挑战

- **gremote（cco，统一数据格式，对其数据）**

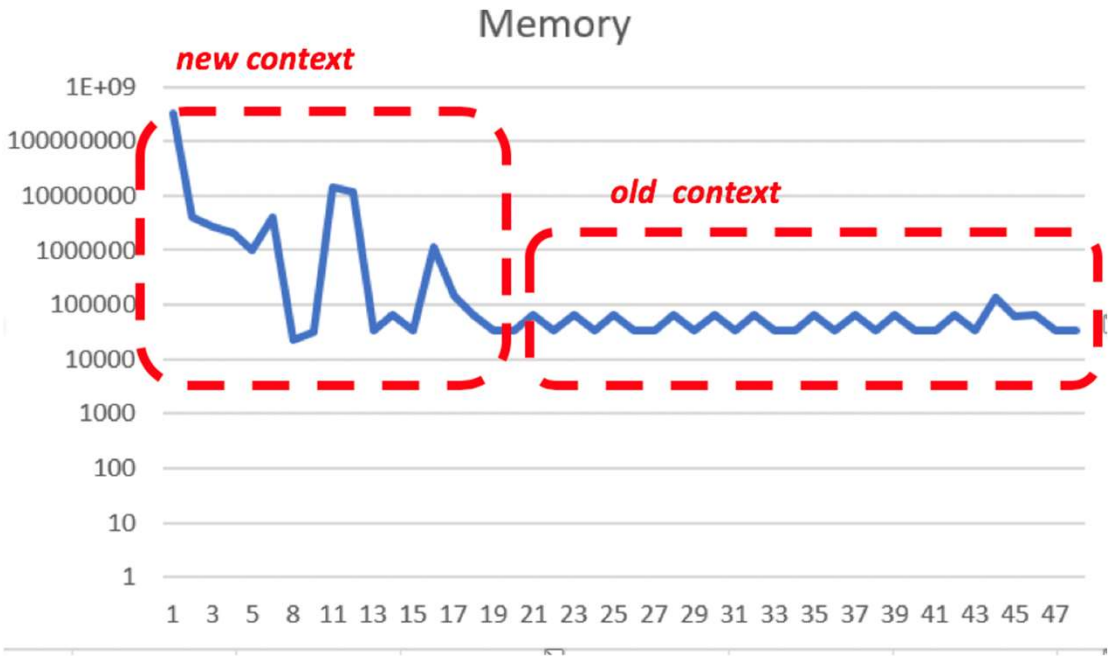
```
glXChooseFBConfig(dpy = 0xcce9e0, screen = 0, attribList..., );
glXGetVisualFromFBConfig(dpy = 0xcce9e0, config = 0xd34b20);
glXCreateContext(dpy = 0xcce9e0, vis = &visual = 0xcda350, visualid = 36, screen = 0...);
glXMakeCurrent(dpy = 0xcce9e0, drawable = 81788930, ctx = 0xdb67f8);
glViewport(x = 0, y = 0, width = 2560, height = 1440);
glScissor(x = 0, y = 0, width = 2560, height = 1440);
glXDestroyContext(dpy = 0xcce9e0, ctx = 0xdb67f8);
glXCreateContextAttribsARB(dpy = 0xcce9e0, config = 0xd34b20, share_context =NULL...);
glXMakeCurrent(dpy = 0xcce9e0, drawable = 81788930, ctx = 0xee7648);
glXSwapIntervalEXT(dpy = 0xcce9e0, drawable = 81788930, interval = 0);
glDepthFunc(func = GL_LEQUAL);
glEnable(cap = GL_DEPTH_TEST);
glViewport(x = 0, y = 0, width = 2560, height = 1440);
...
glGenTextures();
```



```
(28 7A 2A 5E 2C 5D 2E 4A 30 43 32
(28 65 2A 44 2C 4A 2E 46 30 5F 32
(28 65 2A 44 2C 4A 2E 46 30 5F 32
(28 7A 2A 5E 2C 5D 2E 4A 30 43 32
(1F 42 21 56 23 4A 25 69 27 63 )
(1F 6F 21 69 )
(1F 42 21 56 23 4A 25 65 27 49 29
(1F 63 21 43 23 4A 25 45 27 4D 29
(1F 4C 21 43 23 46 25 43 27 44 29
(1F 75 21 51 23 41 25 54 27 66 29
(1F 4C 21 43 23 46 25 43 27 44 29
(1F 70 21 43 23 57 25 55 27 5F 29
(1F 54 21 5A 23 50 25 73 27 5B 29
(1F 54 21 5A 23 50 25 76 27 49 29
(1F 46 21 50 23 49 25 6A 27 47 29
```

GPU虚拟化挑战

- **网络带宽损耗（瓶颈）**

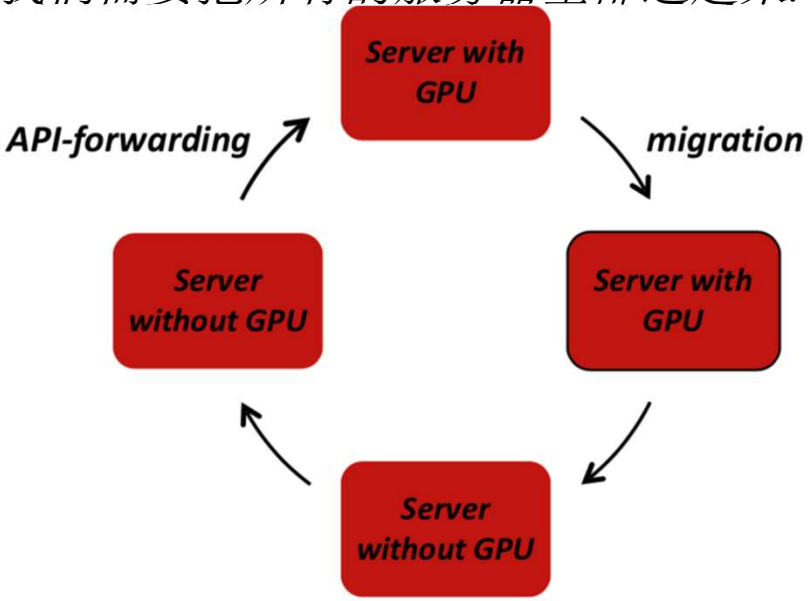


对于大应用而言，大数据量往往发生在新的状态的第一帧或前几帧；往后，数据量会接近于平稳，而对于很多方式方法，都是在状态平稳后做文章。

GPU虚拟化挑战

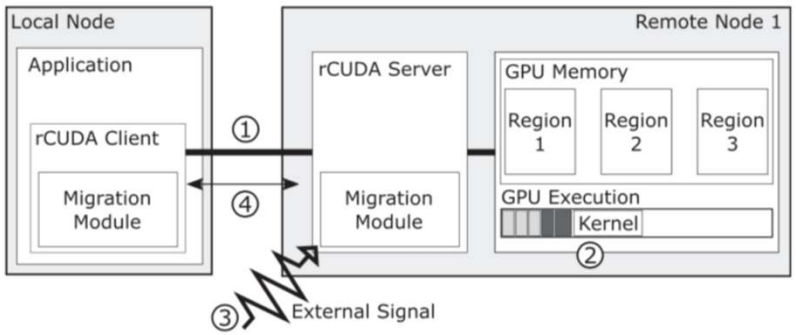
- **GPU迁移（motivation）**

- *每个GPU应用对于硬件资源的需求都是动态的。 (DCUDA (SOCC'20), gRemote (HPDC'20))*
- *系统不在真正作用于一个服务器上，而是一个集群(rCUDA (tpds'19))*
- *在这个集群中，我们需要把所有的服务器全部连起来. (Microsoft XCloud)*

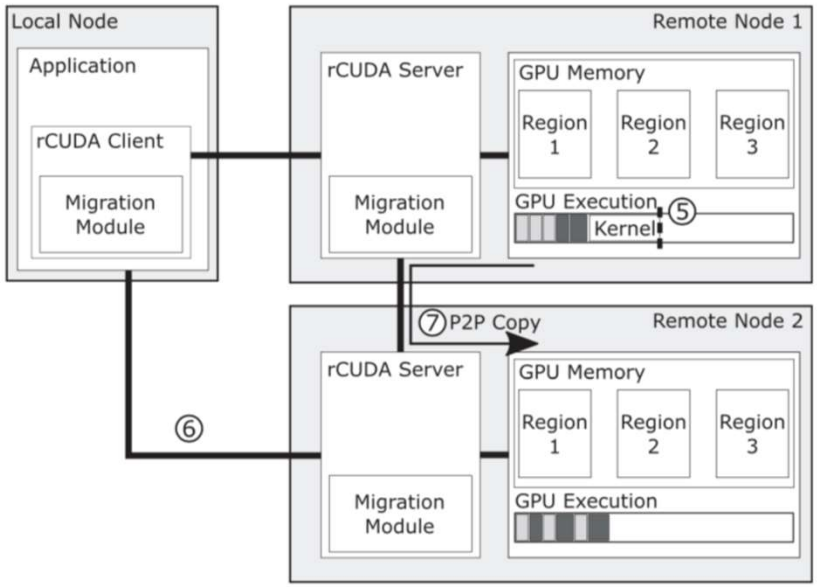


GPU虚拟化挑战

- **GPU迁移（rCUDA）**



(a) The application starts execution and, at some point in time, the migration signal, triggered by the resource scheduler, arrives at the rCUDA server.



(b) The memory is copied from source GPU to destination GPU in another node of the cluster.

GPU虚拟化挑战

- **GPU迁移瓶颈**

1. **迁移时间过长，所有的技术基本上都在用retrace 和 replay**
2. **整个迁移时间是秒级别的，比起cpu迁移的ms级别相差很多**
3. **因此，对于GPU的迁移，目前没人说是实时迁移。**

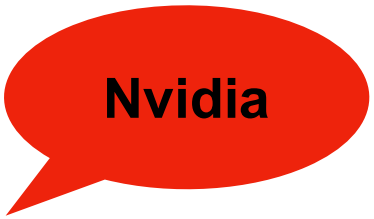
GPU虚拟化挑战

- **造成技术瓶颈的原因： GPU和系统的闭源**

https://www.zhihu.com/zvideo/1288795168036892672?utm_source=wechat_session&ut

GPU虚拟化挑战

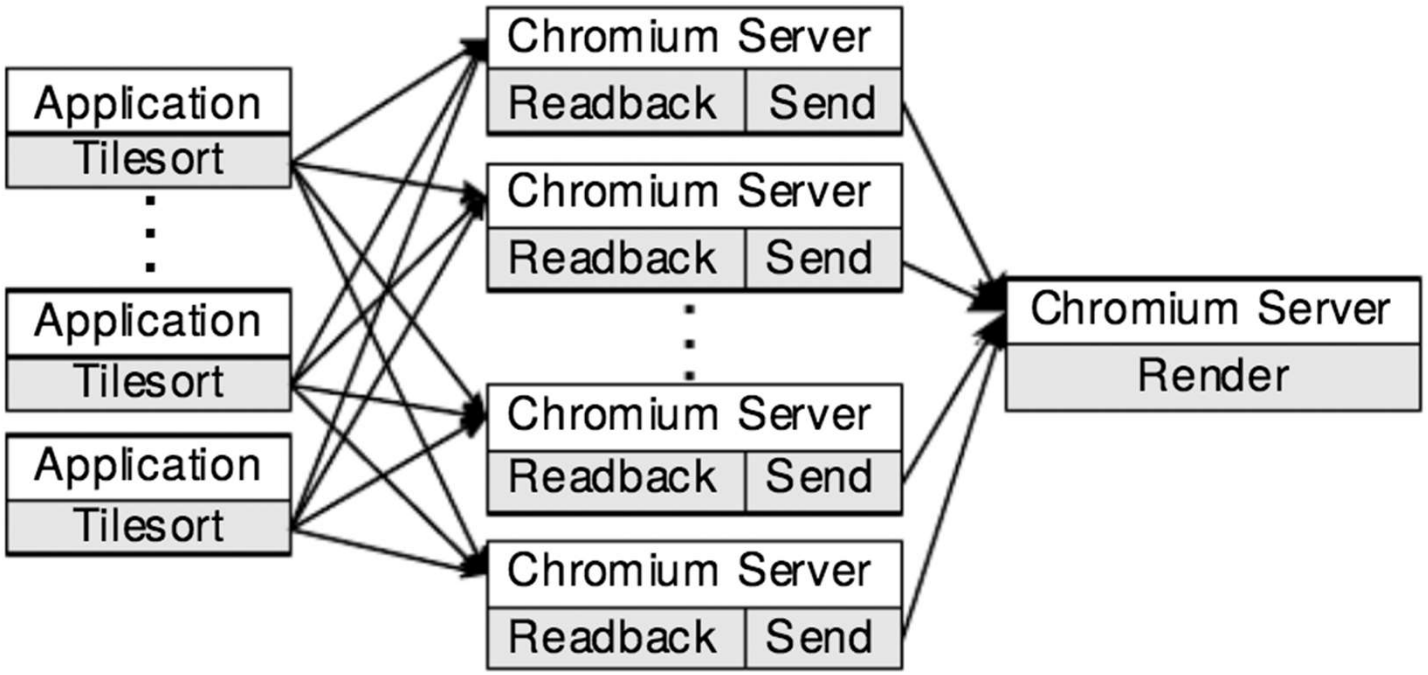
- **造成技术瓶颈的原因： GPU和系统的闭源**



https://www.zhihu.com/zvideo/1288795168036892672?utm_source=wechat_session&ut

新型硬件架构的初想

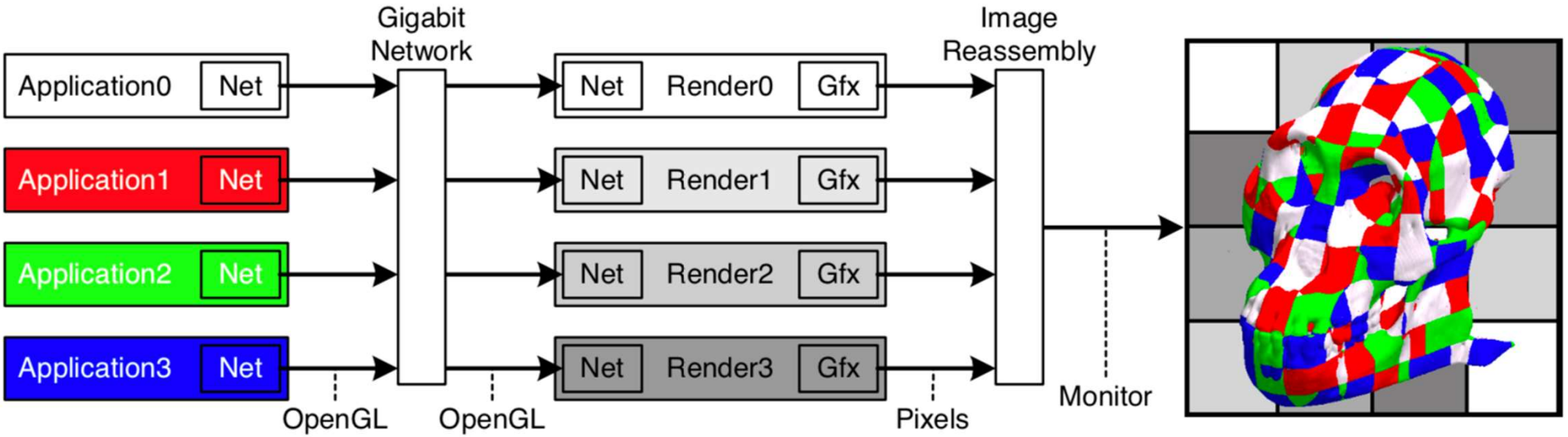
- **扩展性GPU**



1. **Chromium 发在 siggraph 2003**
2. **应用可以跑在多个或一个GPU上面**

新型硬件架构的初想

- **扩展性GPU**



1. **按颜色分类和object分类**
2. **然后跑在不同的硬件上面**

新型硬件架构的初想

- **扩展性GPU**
  - **2.0时代处理逻辑**

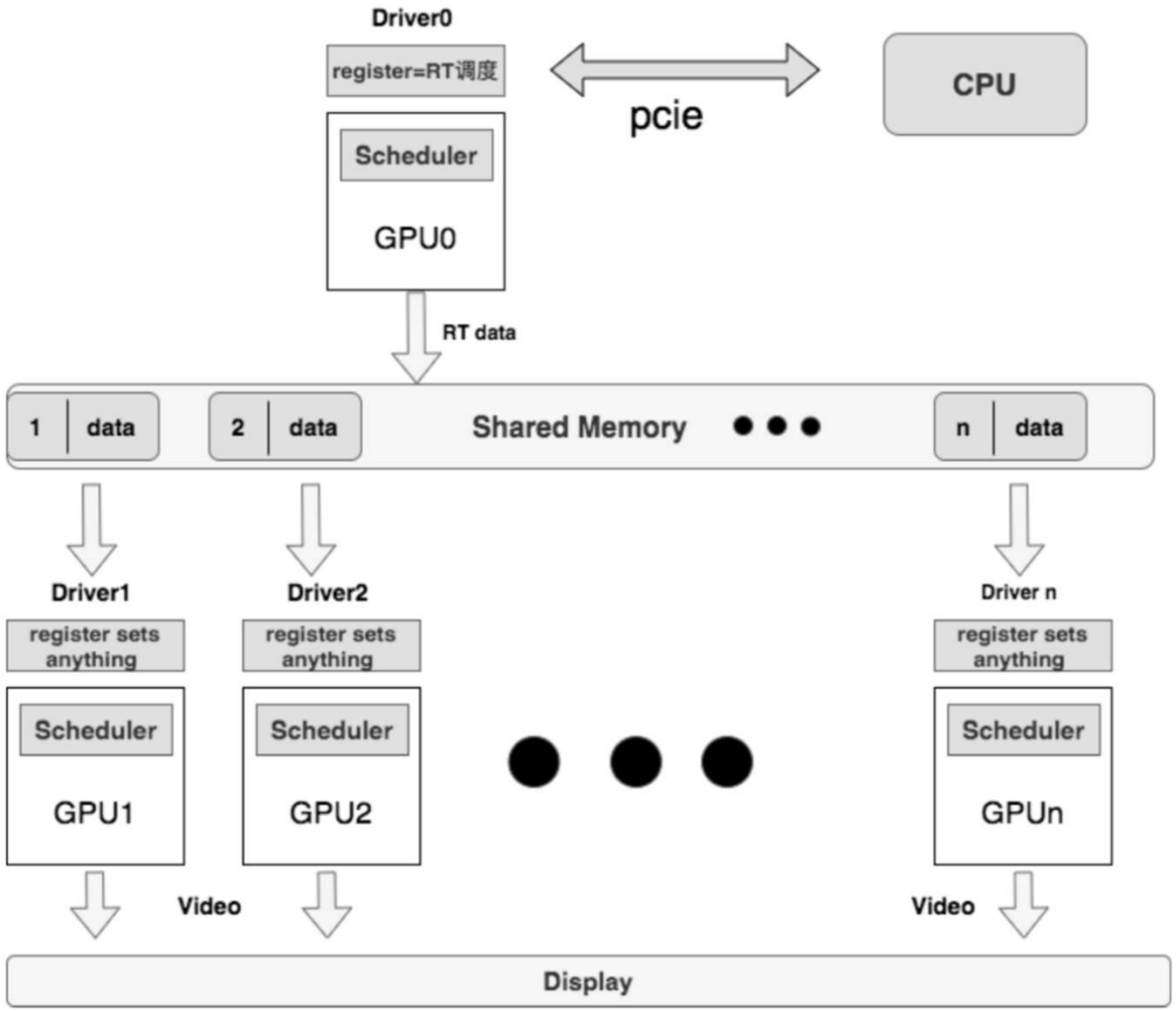
```
glBegin(GL_QUADS);
da = 2.0 * M_PI / teeth / 4.0;
for (i = 0; i < teeth; i++) {
angle = i * 2.0 * M_PI / teeth;
glVertex3f(r1 * cos(angle), r1 * sin(angle), width *
0.5);
glVertex3f(r2 * cos(angle + da), r2 * sin(angle + da),
width * 0.5);
glVertex3f(r2 * cos(angle + 2 * da), r2 * sin(angle + 2
* da),  width * 0.5);
glVertex3f(r1 * cos(angle + 3 * da), r1 * sin(angle + 3
* da),width * 0.5);
 }
 glEnd();
```

  - 当代处理逻辑

```
glCompileShader(shader = 3)
glCreateShader(type = GL_FRAGMENT_SHADER) = 4
glShaderSource(shader = 4, count = 1, string =
&"#version 150
uniform sampler2DShadow s_texture_0;
in vec4 s_texcoord;
in vec4 s_color;
out vec4 s_frag_data_0;
out vec4 s_frag_data_1;
void main() {
s_frag_data_0 =
texture(s_texture_0,vec3(s_texcoord.xy,0.99)) *
s_color;
s_frag_data_1 = s_frag_data_0;
}
", length = NULL)
glBufferData (...., buffer = data_array)
glBindAttrib  (....)
```

新型硬件架构的初想

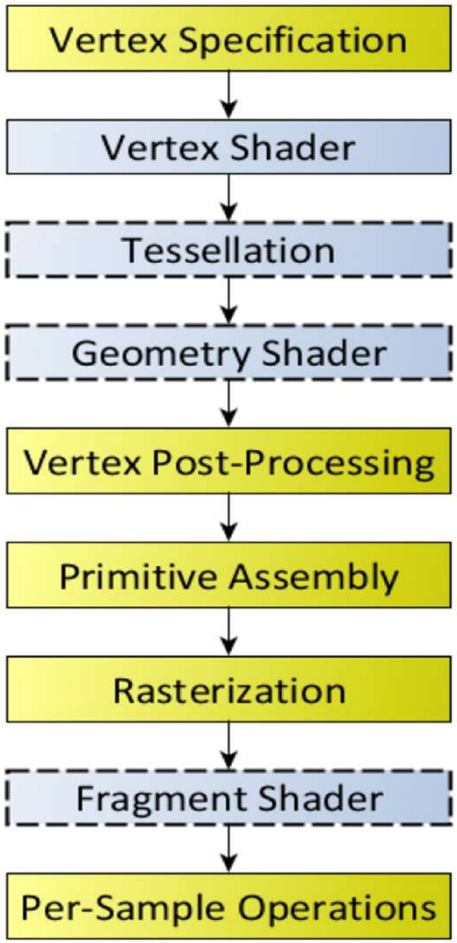
- **扩展性GPU**



新型硬件架构的初想

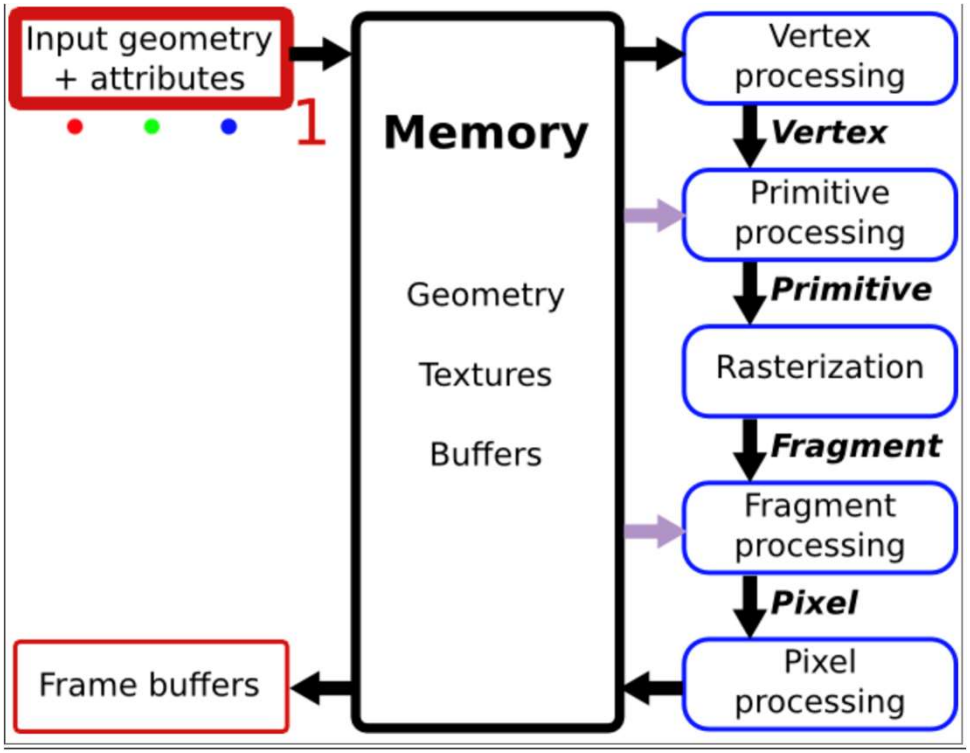
- **扩展性GPU**

NOW.



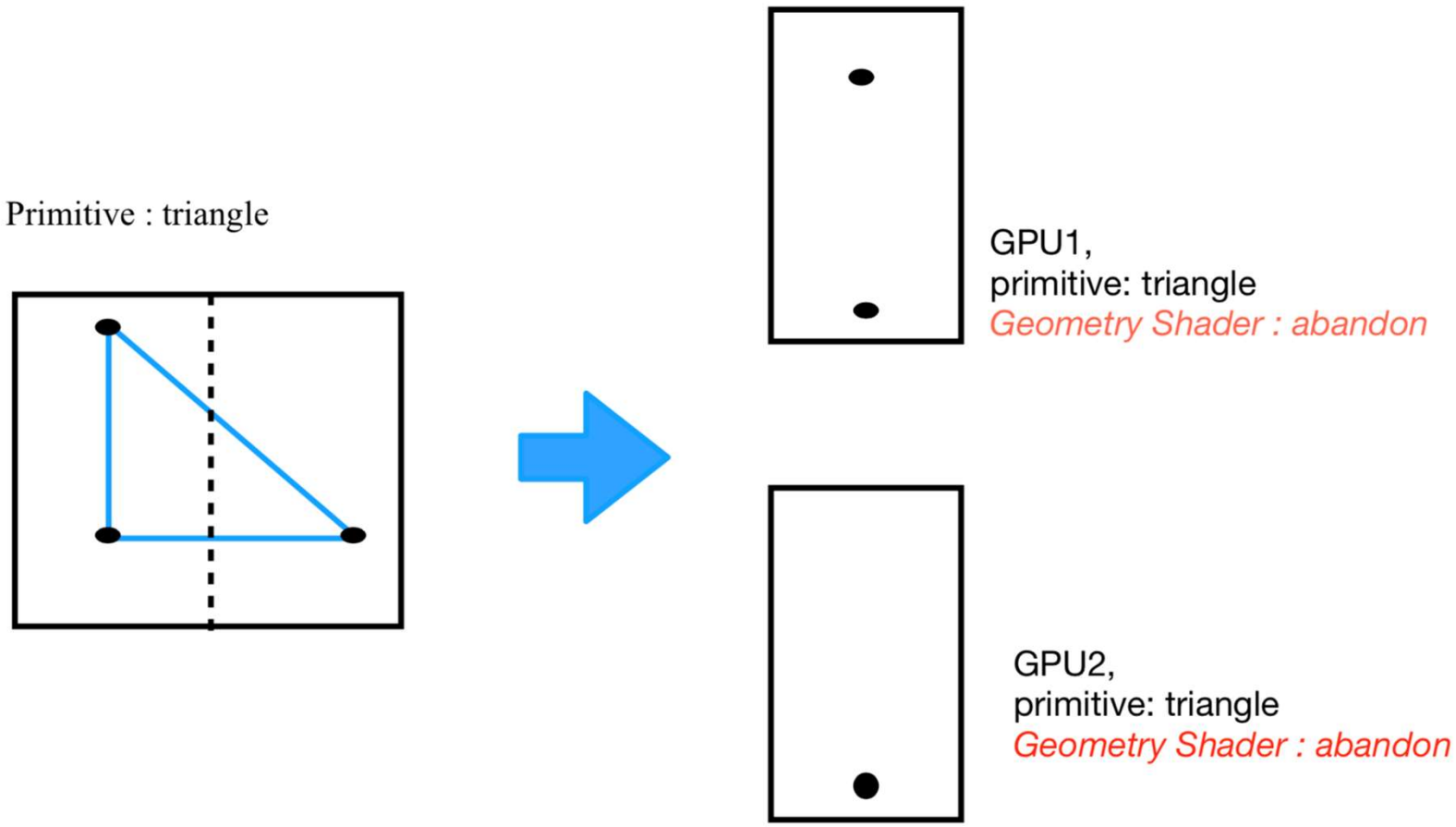
VS

1.0



新型硬件架构的初想

- **扩展性GPU**



新型硬件架构的初想

- **扩展性GPU**

- 应用环境： 可以接触gpu driver；且每一个硬件单元只有GPU组成，CPU和GPU的数量处于一对多的状态

- 限制： 需要提供完整纯净的GPU驱动（mesa不完整也不纯净），对驱动进行大量的魔改，所以需要对GPU架构，驱动全占有着深入且全面的了解（我们对此一无所知），操作难度相对大，可能会加速CPU的bottleneck

- 优点：overhead相对小。如果成功，可以为主打vulkan的可扩展性GPU（nvidia的短板）的进步增添力量。

总结

- **GPU虚拟化需要走很长的路（从生态到架构）**
- **相比于CPU, GPU架构和虚拟化处于初级状态**
- **打破英伟达垄断，是GPU发展的目标**
- **欢迎大家了解并真正认识GPU**