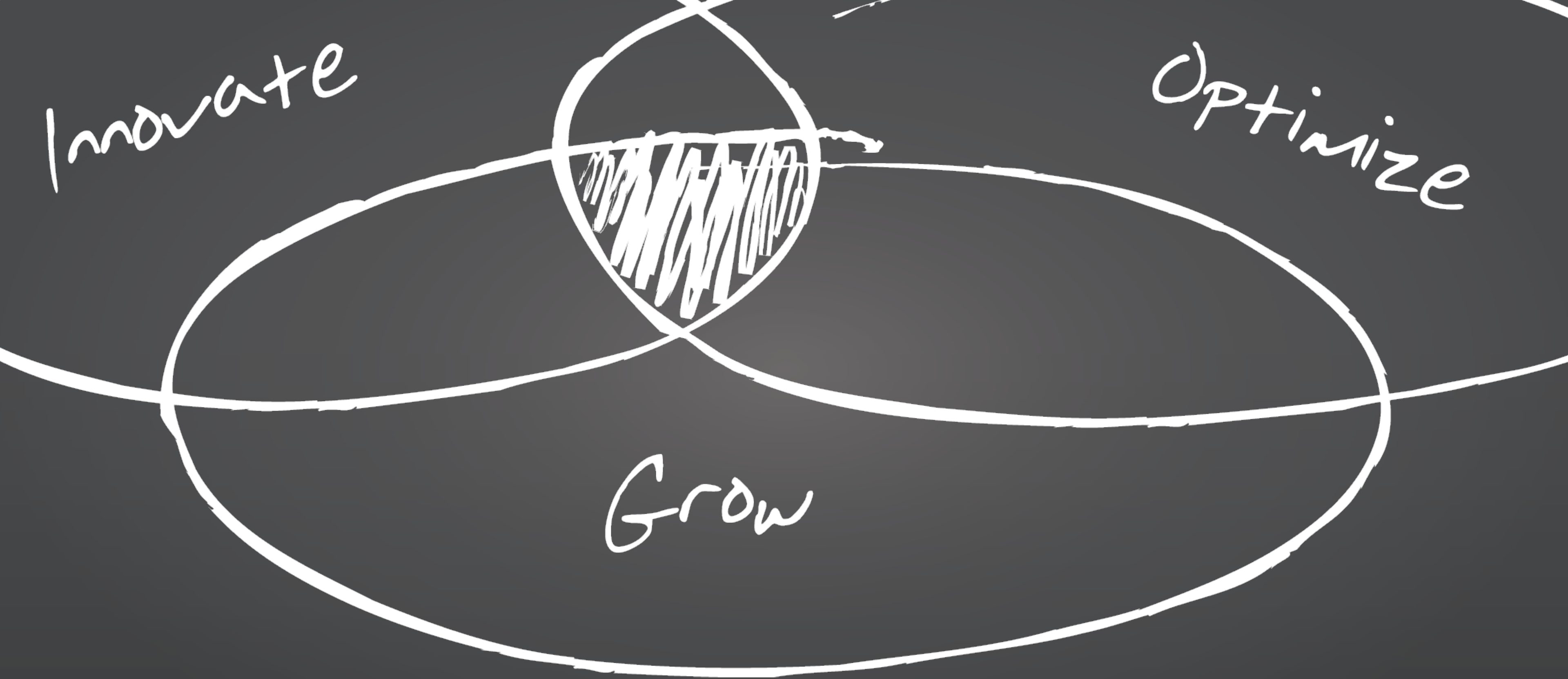
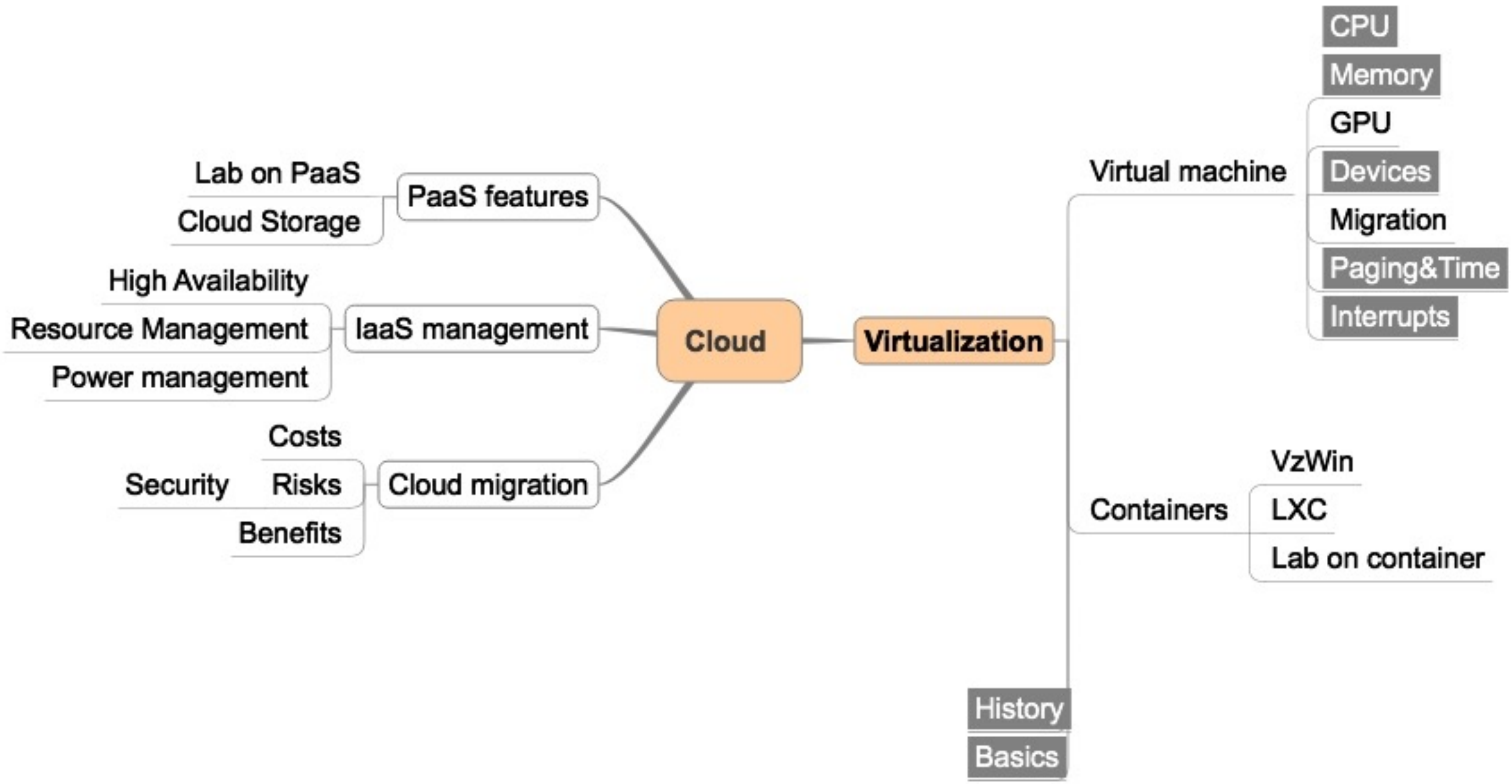


The total virtualization

Virtualizing graphics

Lecture 8

Course overview



Content

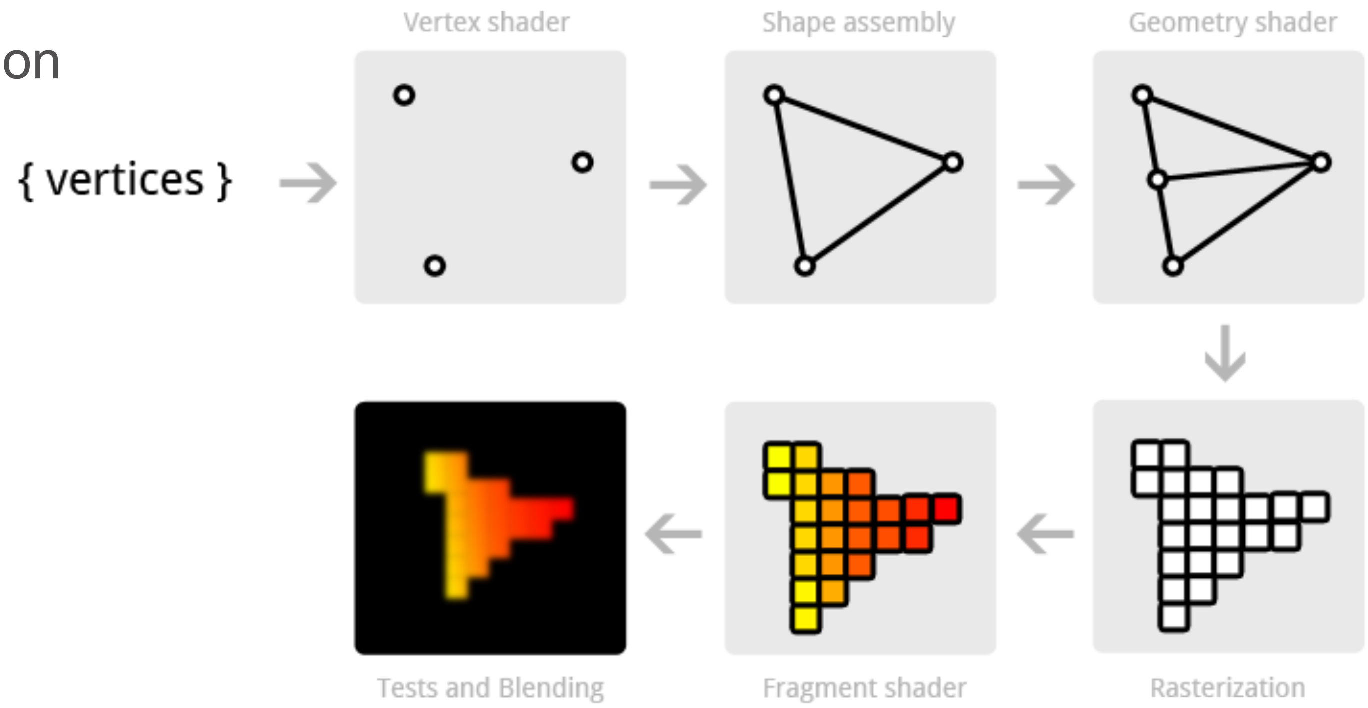
- ✓ Graphics as we know it
- ✓ Virtualizing graphics - a complexity of video system
- ✓ Virtual GPU: approaches

GPU is used for many appliances. Understanding it and its virtualization is vital for big data

Graphics: what do we know about it

Graphics pipeline

- 3D geom primitives (triangles)
- Modeling + transformation
- Camera transformation
- Lighting
- 3D transformation
- Clipping
- Rasterization
- Texturing



Graphics: graphics history (2)

	1996	1997	1998	1999
Application tasks (move objects according to application, move/aim camera)	CPU	CPU	CPU	CPU
Scene level calculations (object level culling, select detail level, create object mesh)	CPU	CPU	CPU	CPU
Transform	CPU	CPU	CPU	GPU
Lighting	CPU	CPU	CPU	GPU
Triangle Setup and Clipping	CPU	GPU	GPU	GPU
Rendering	GPU	GPU	GPU	GPU

3D Application and API

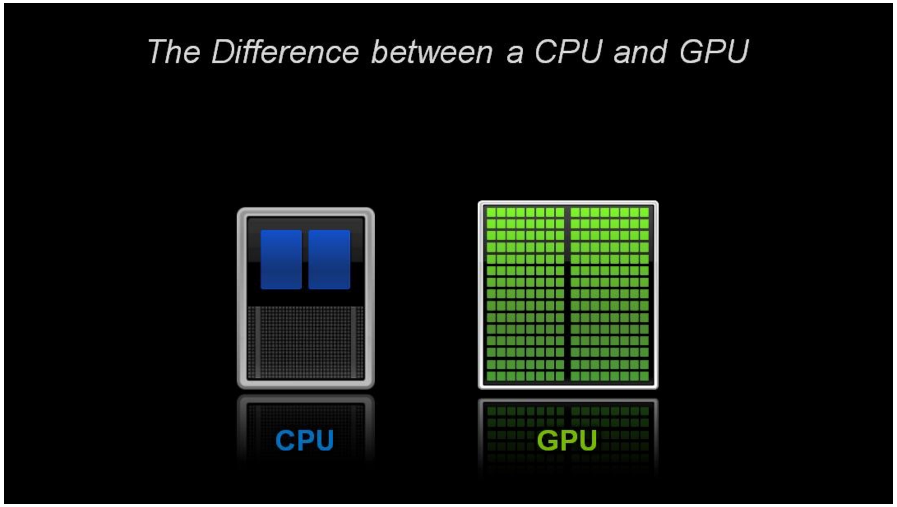
3D Graphics Pipeline

Graphics card: what does it contain?

Graphics card: what does it contain?

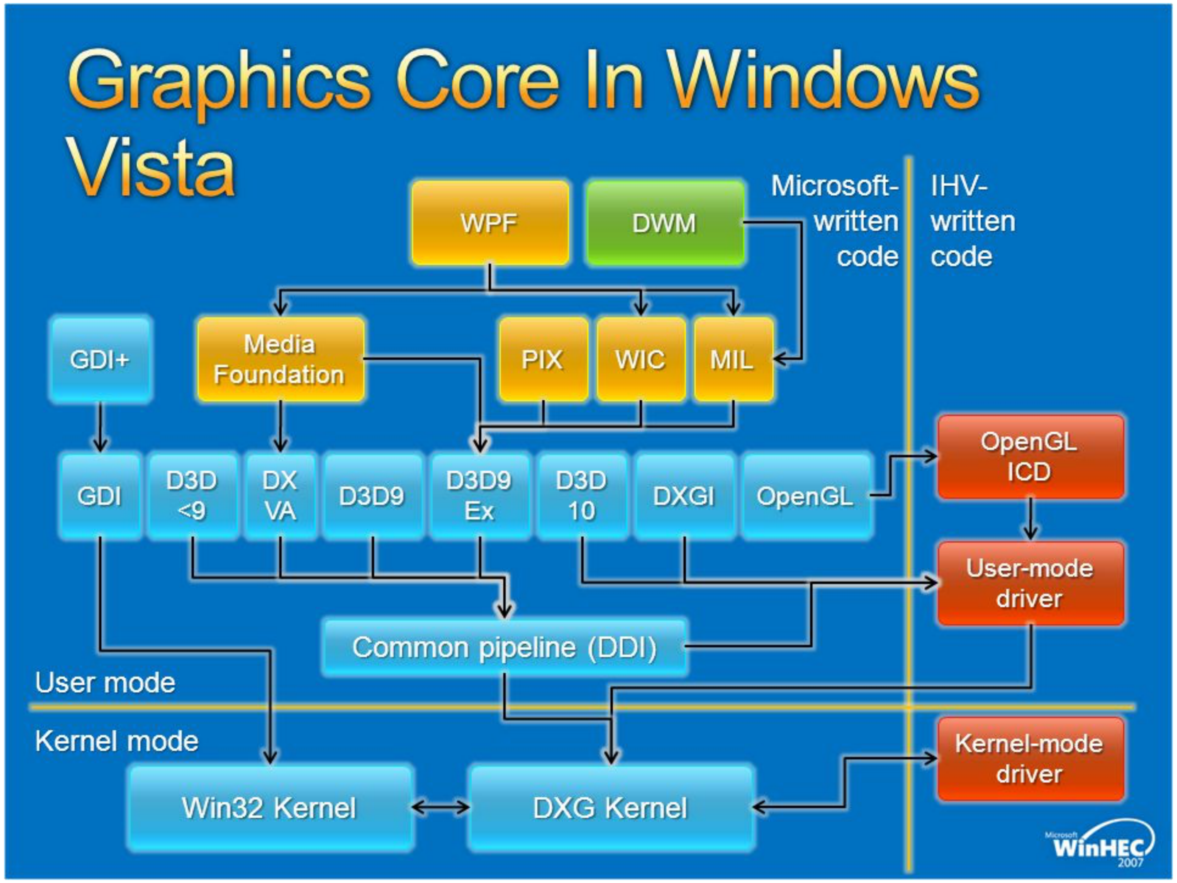
- GPU
- Heat sink
- Video memory
- Video BIOS
- RAMDAC (Random Access Memory Digital-to-Analog Converter)
- Output interfaces (VGA, HDMI, DVI)

Graphics card: GPU vs CPU

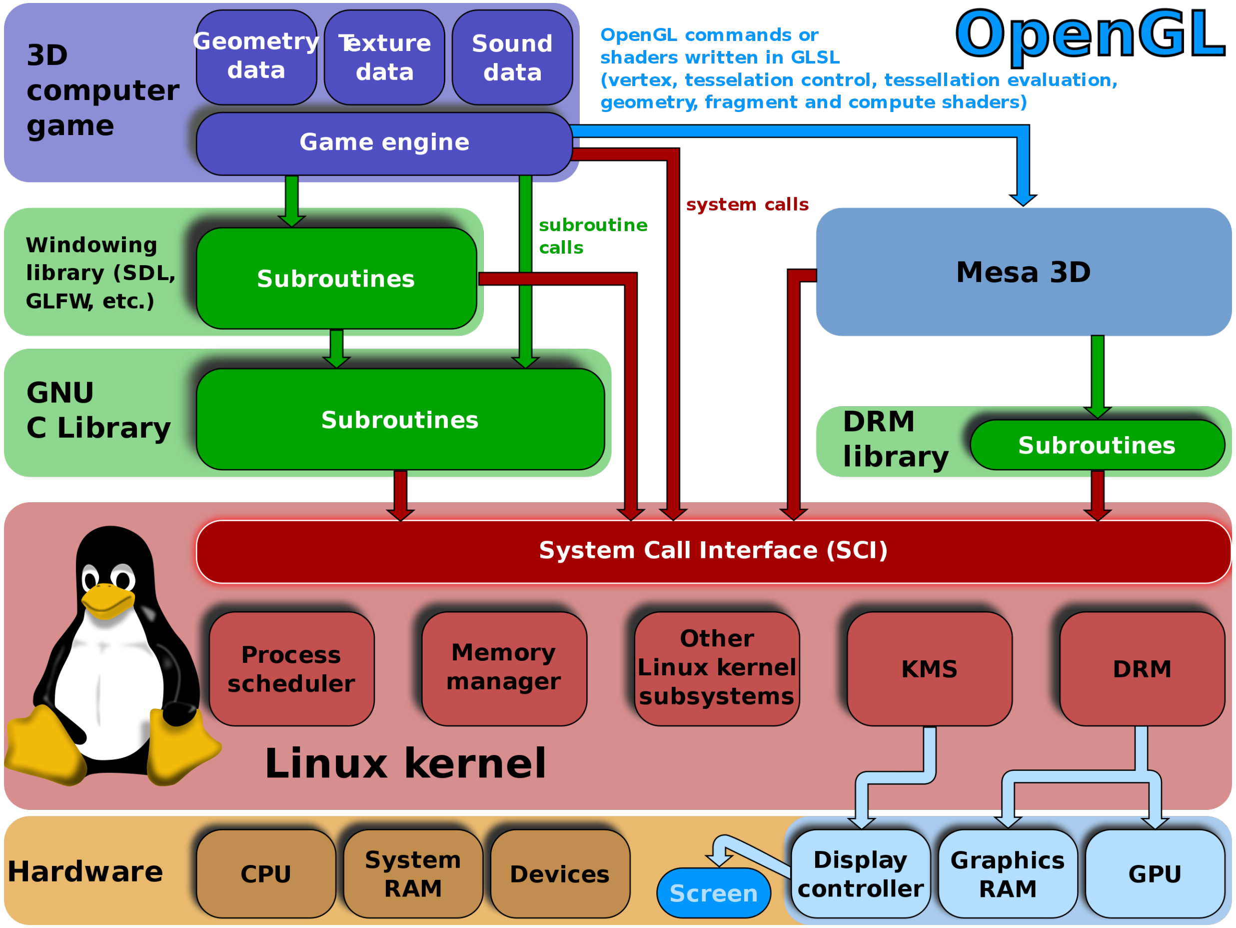


- Multiple GPU architectures
- Proprietary incompatible hardware API

Graphics card: OpenGL/DirectX/Vulkan vs GPU



Graphics card: OpenGL/DirectX/Vulkan vs GPU



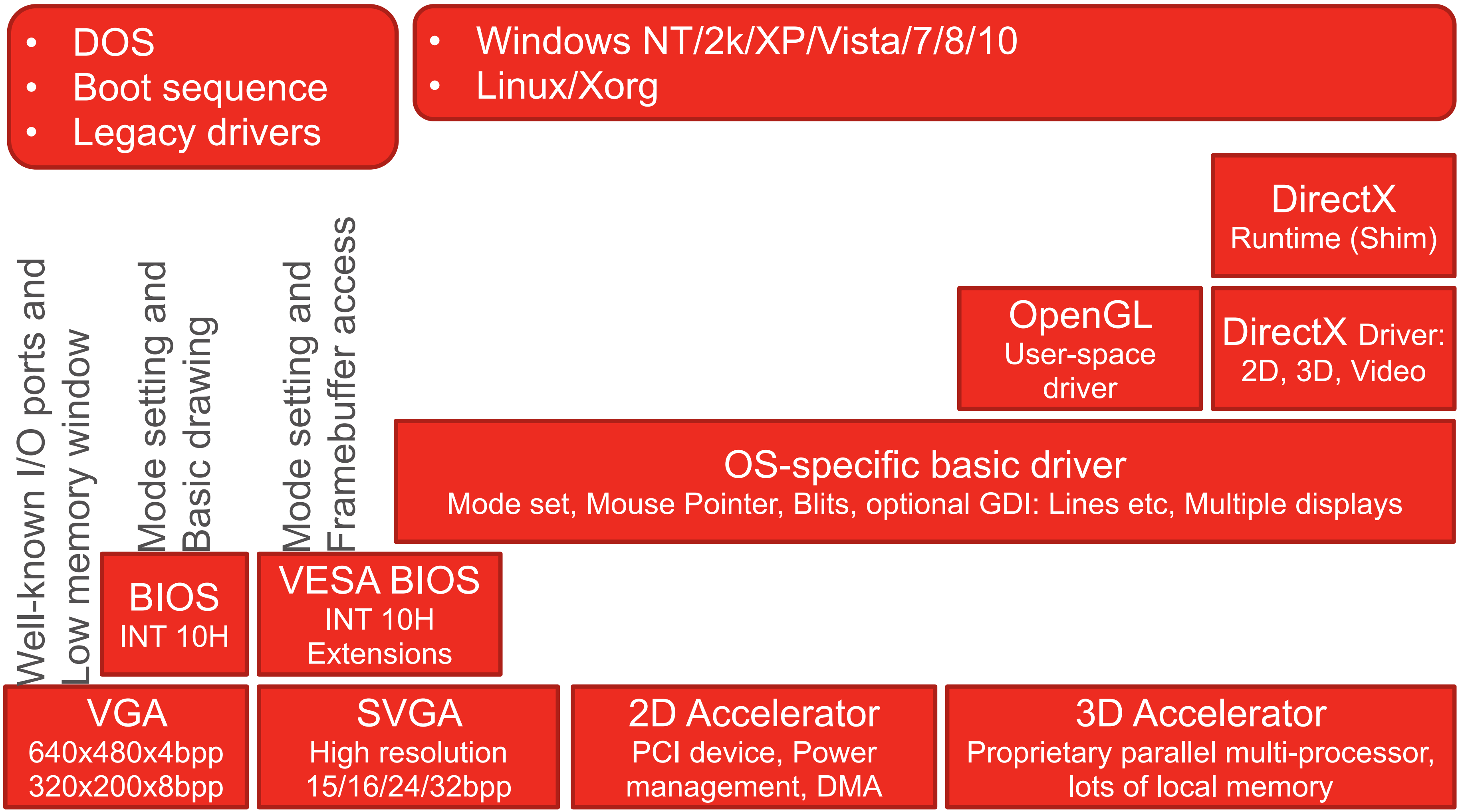
Graphics card: OpenGL/DirectX/Vulkan vs GPU

- Abstracting video from video provider (proprietary code)
- Allows to execute operations if video provider is unable to do that

Graphics card: CUDA

```
void main(){
    float *a, *b, *out;
    float *d_a;

    a = (float*)malloc(sizeof(float) * N);

    // Allocate device memory for a
    cudaMalloc((void**)&d_a, sizeof(float) * N);

    // Transfer data from host to device memory
    cudaMemcpy(d_a, a, sizeof(float) * N, cudaMemcpyHostToDevice);

    …
    vector_add<<<1,1>>>(out, d_a, b, N);
    …

    // Cleanup after kernel execution
    cudaFree(d_a);
    free(a);
}
```

Video device API legacy stack



Virtualizing graphics: why do we need?

Virtualizing graphics: why do we need?

- ✓ CAD
- ✓ CAE
- ✓ CAM
- ✓ Games
- ✓ CUDA/OpenCL (3D Computation)

Now Amazon supplies G2 instances that are GPU As A Service

Virtualizing graphics: principles

- ✓ Performance is more important then accuracy
- ✓ Software compatibility is more important then features

Virtualizing graphics: approaches

Virtualizing graphics: approaches

- ✓ Software emulation
- ✓ Paravirtualization (front-end/back-end pair)
- ✓ Dedicated card (passthrough)
- ✓ GPU virtualization

Virtualizing graphics: software emulation

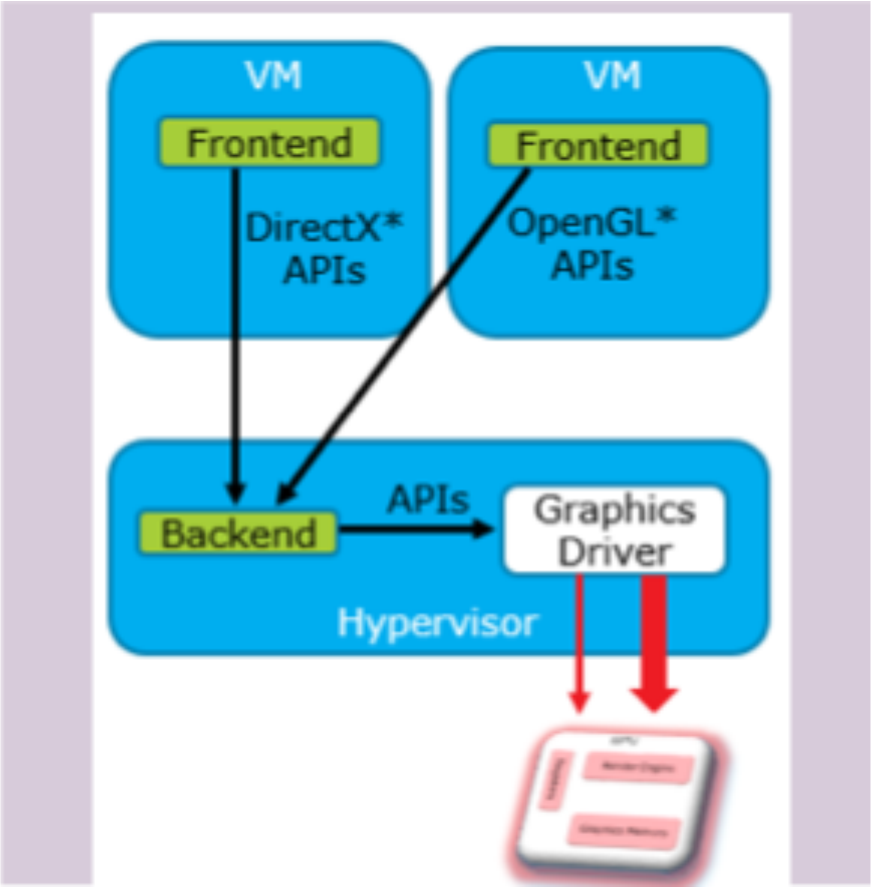
- ✓ Hardware and software agnostic

- ➢ High complexity
  - VGA
  - 2D only
- ➢ Poor performance

Virtualizing graphics: Paravirtualization (a.k.a. API Forwarding)

- ✓ Hardware agnostic
- ✓ Easy to implement

- ➢ API compatibility
- ➢ CPU overhead

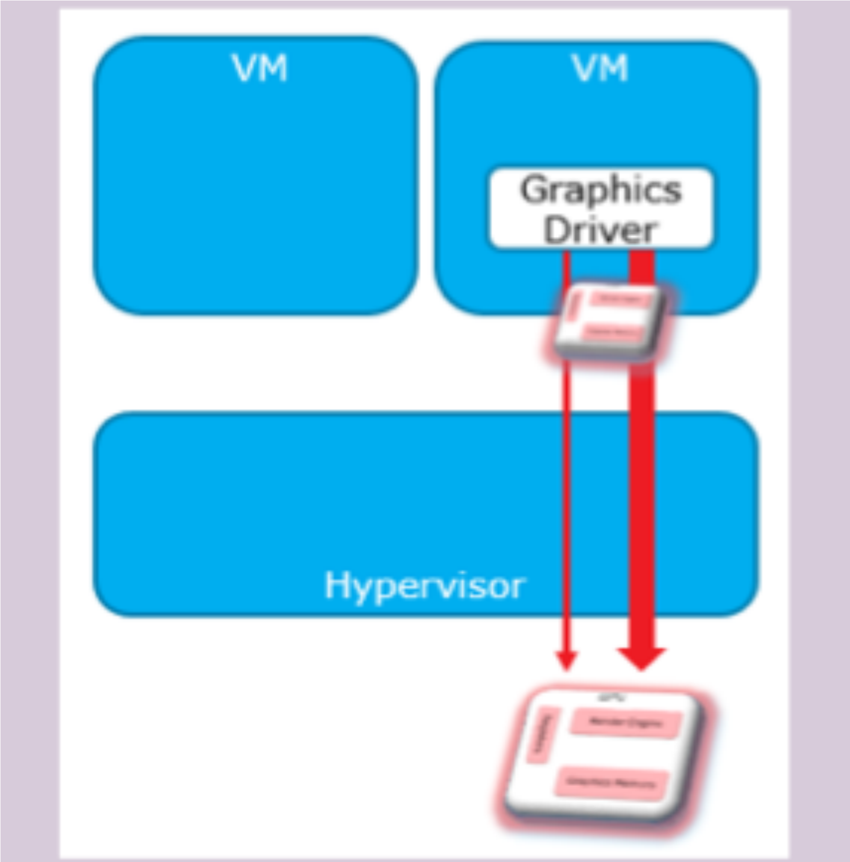


Virtualizing graphics: Pass-through

✓ The near-native performance

➢ Only for specific hardware

➢ No multiplexing



Virtualizing graphics: GPU virtualization

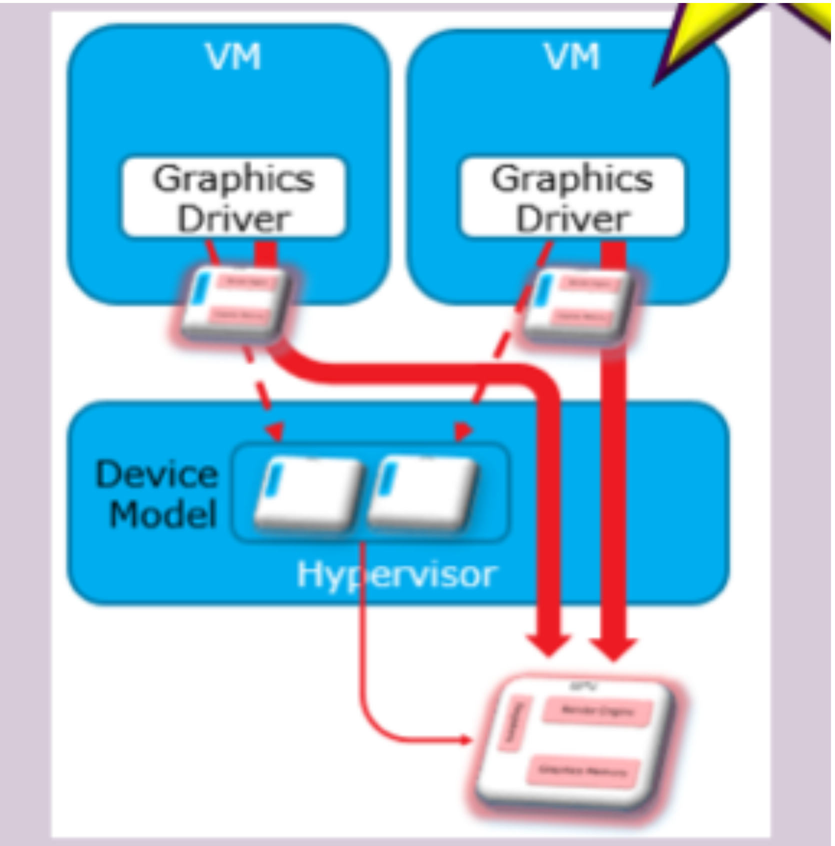
✓ Good performance

✓ Multiplexing

➢ Only for specific hardware

➢ Complex solution

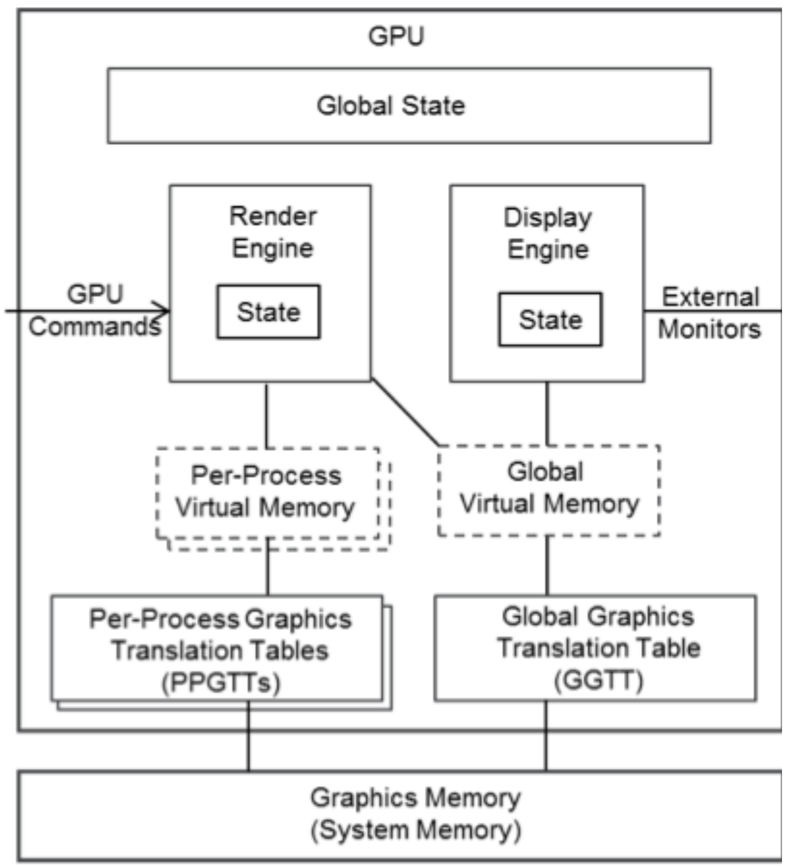
➢ Requires access to GPU spec



Graphics card: what is about CUDA?

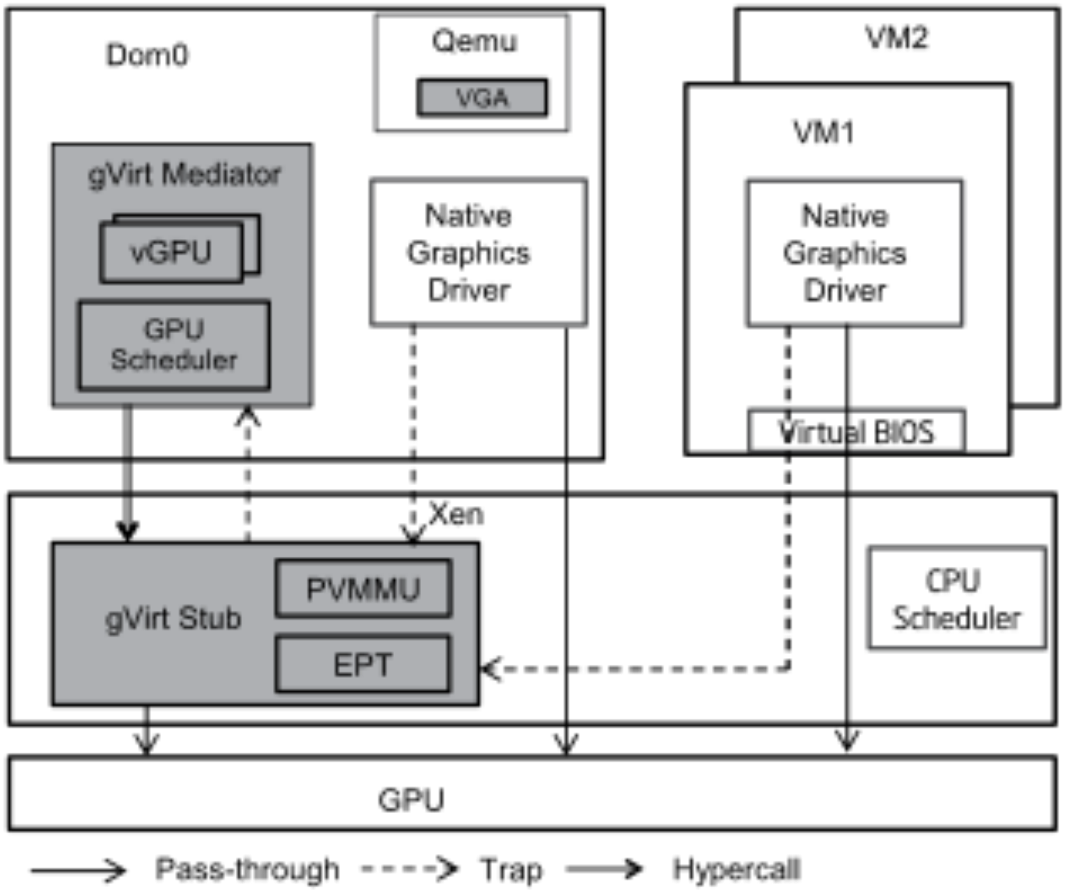
```
void main(){
    float *a, *b, *out;
    float *d_a;

    a = (float*)malloc(sizeof(float) * N);

    // Allocate device memory for a
    cudaMalloc((void**)&d_a, sizeof(float) * N);

    // Transfer data from host to device memory
    cudaMemcpy(d_a, a, sizeof(float) * N, cudaMemcpyHostToDevice);

    …
    vector_add<<<1,1>>>(out, d_a, b, N);
    …

    // Cleanup after kernel execution
    cudaFree(d_a);
    free(a);
```

GPU virtualization: GPU model



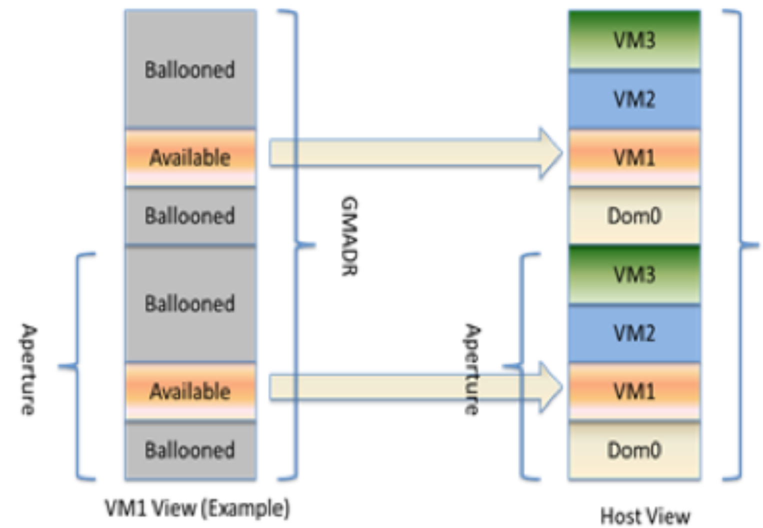
- Graphics memory
  - Virtual memory address spaces
    - A single global virtual memory (GVM) space
    - Multiple per-process virtual memory (PPVM) spaces
  - Backed by system memory through GTTs
- Render engine
  - Fulfill the acceleration capability through fixed pipelines and execution units
- Display engine
  - Route date from graphics memory to external monitors
- Global state
  - Represent remaining circuits, including initialization, PM, etc.

GPU virtualization: Common view



GPU virtualization: memory model (1)

- The single GVM space is partitioned
  - Access to VM’s own GVM region is **passed through**
  - Classical memory virtualization challenge
    - Host view vs. guest view
  - Address space ballooning with driver cooperation

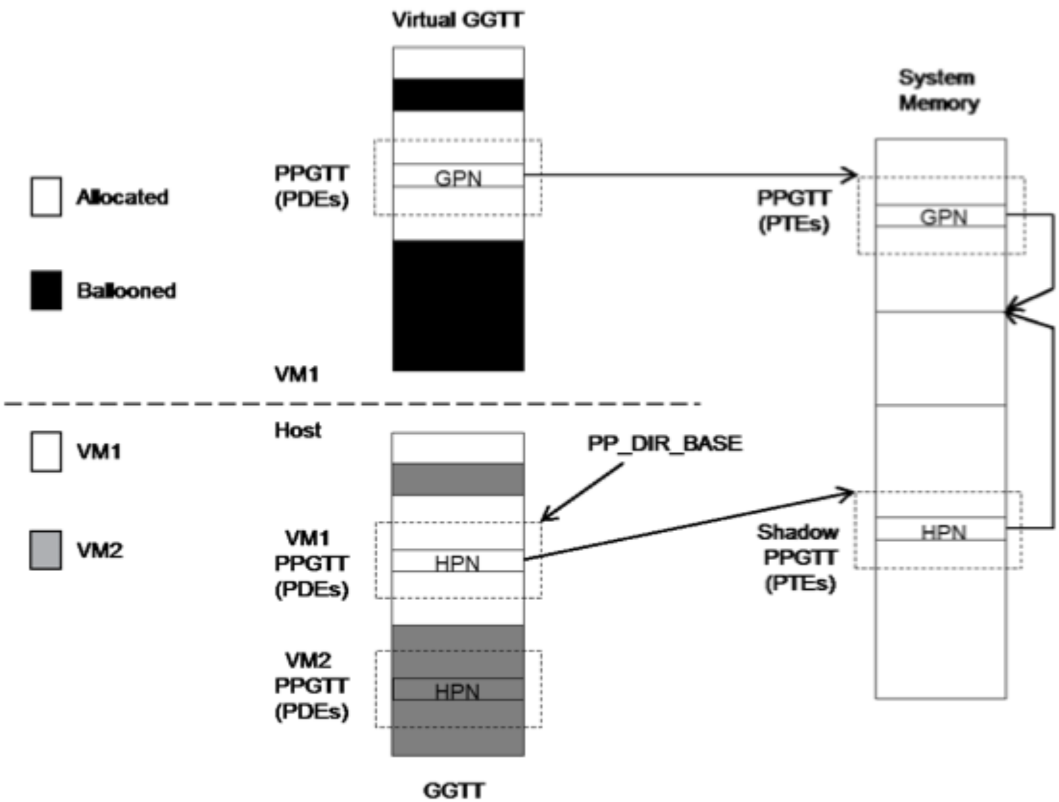


- GGTT accesses are **mediated**
  - Access to its own GGTT entries is translated
    - GPFN <-> MFN
  - Access to others’ entries is virtualized

GPU virtualization: memory model(2)

Per-Process Virtual Memory Spaces

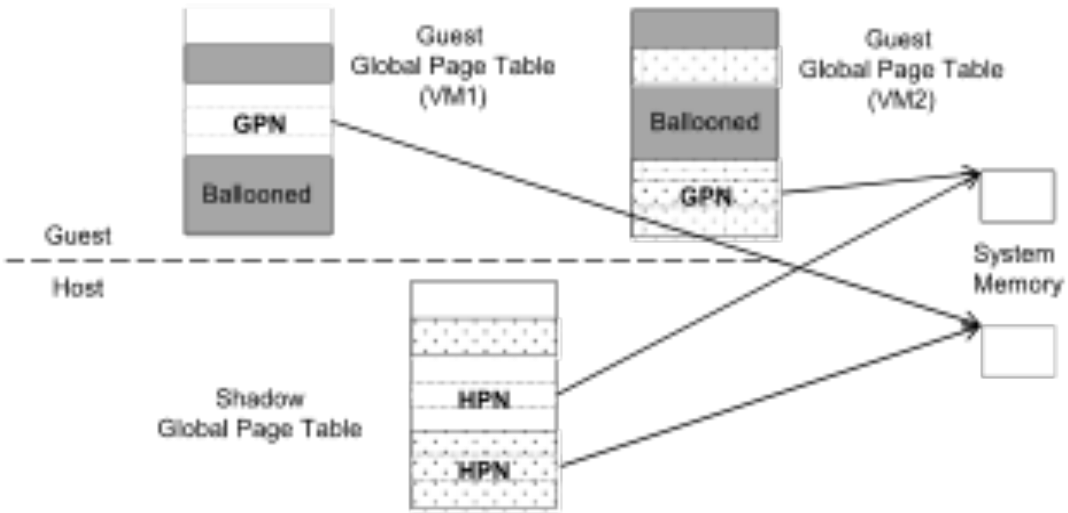
- Each VM manages its own PPVM spaces
  - Active space pointed by PP_DIR_BASE
  - Accesses are **passed through**
- PPGTT accesses are **write-protected**
  - Shadow PPGTT table
  - Switch PP_DIR_BASE at render context switch



GPU virtualization: memory model(3)

PPGTT accesses are write-protected:

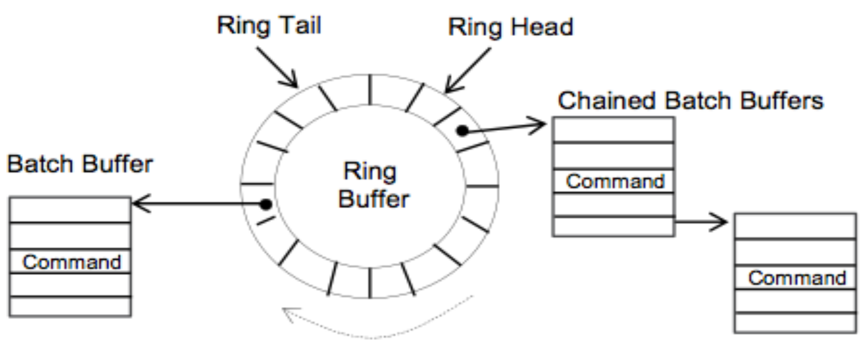
- Shadow PPGTT table
- Switch PP_DIR_BASE at render context switch

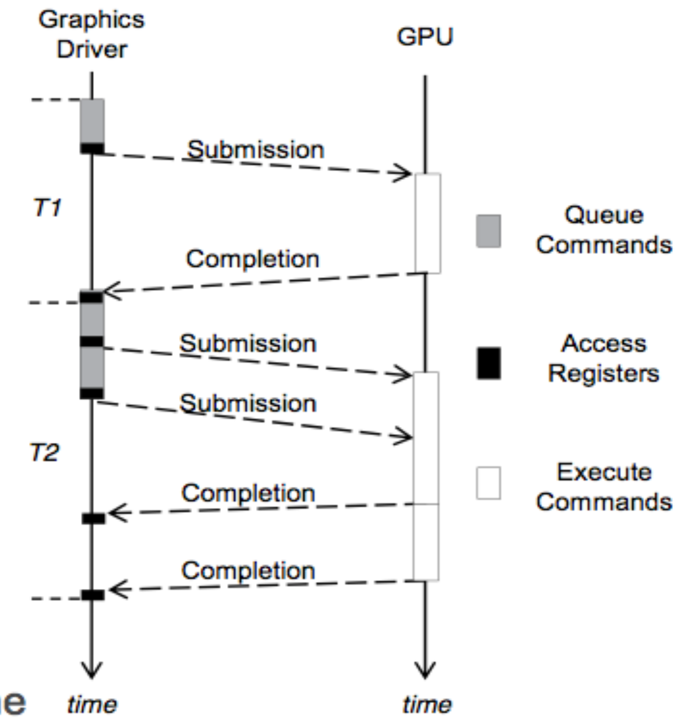


GPU virtualization: command buffer

Command Buffers

- Command buffer access is **passed through**
  - Reside in virtual memory spaces





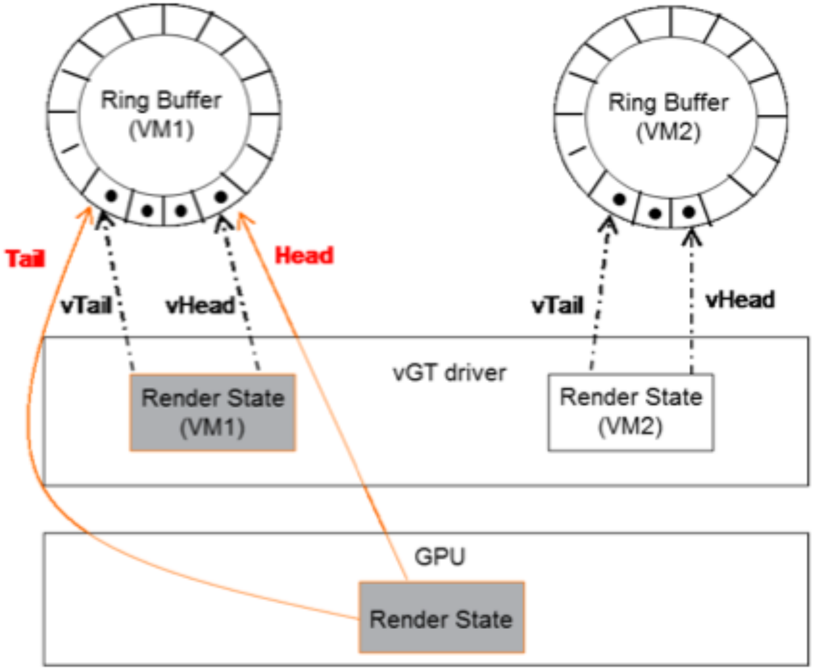
- Command submission request is **mediated**
  - Through MMIO register (ring tail)
  - Render scheduler makes the decision
    - Render owner request is submitted to render engine
    - Non-render owner request is blocked

Render Engine Sharing

- A simple round-robin scheduler
  - In 16ms epoch
- Render owner access is trap-and-forwarded to the render engine
- Non-render owner access is trap-and-emulated

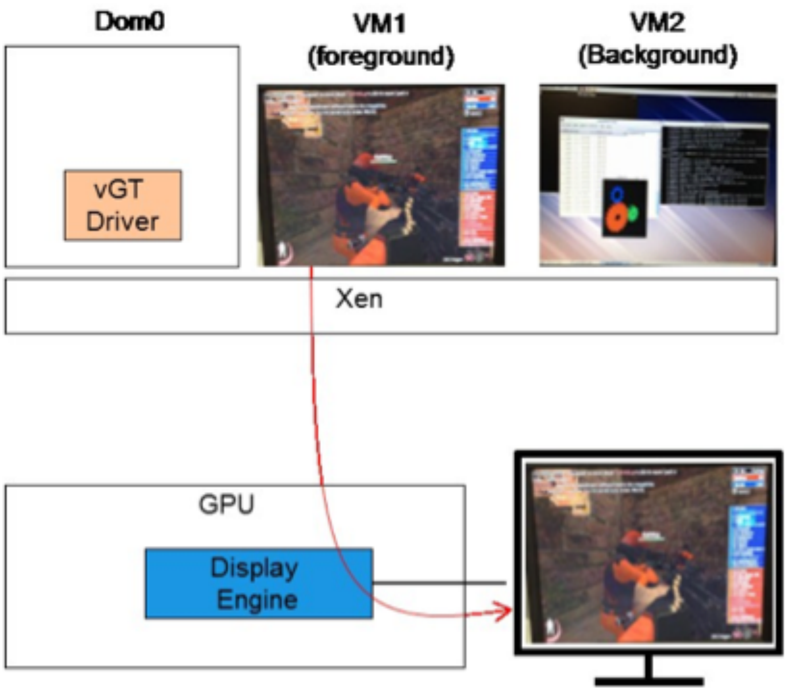
Render context switch flow

1. Wait VM1 ring buffer becoming empty
2. Save render MMIO registers for VM1
3. Flush internal TLB/caches
4. Hardware context switch
5. Restore render MMIO registers for VM2
6. Submit previously queued commands

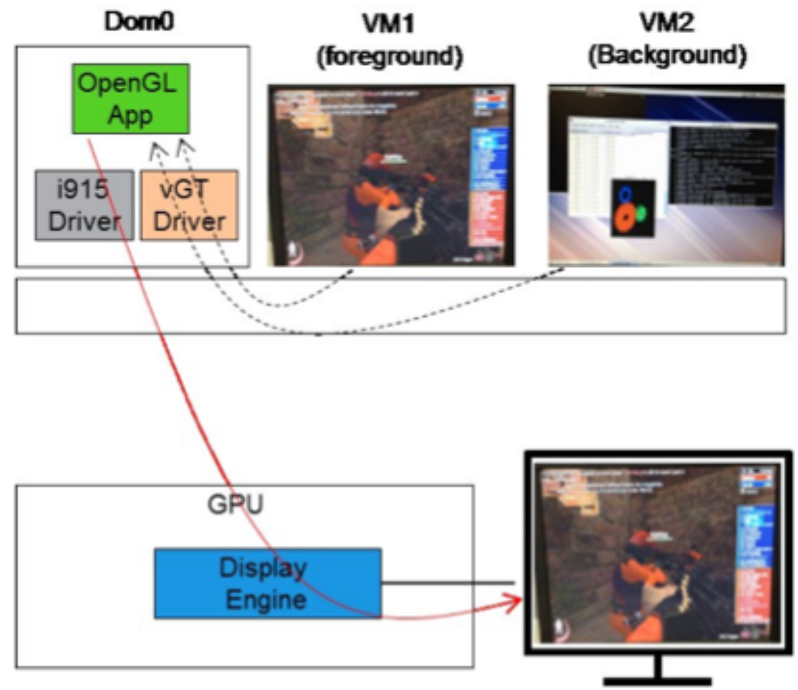


Display Engine Sharing

Direct display model
- Display engine points to the frame buffer of the foreground VM
- vGT driver configures display engine for foreground/background switch



Indirect display model
- vGT driver provides interface to decode VM frame buffer location/format
- An OpenGL app composites VM frame buffers



Video basic also

- 3D coordinates vertices -> 2D
- Trianglization
- Textures based on per-pixel coordinates
- Pixel’s color is calculated from texture, lightening and others
- As many triangles as possible

- Varying number of vertex properties
- Various texture and data formats
- Various ways to feed vertex data
- Configurable/programmable vertex transformations and pixel calculations

- A few major APIs: OpenGL, DirectX/Direct3D
- Applications: games, CAD/CAM, animation, desktop effects

Video device emulation approach

- VGA: full I/O emulation in VMM records state
- VGA: VM polls for mode and frame buffer changes

- SVGA: I/O emulation in VMM enough for BIOS, signals mode changes
- SVGA: VM polls for frame buffer changes based on PTE D-bits

- 2D/PCI device: minimum I/O emulation to support DMA-based protocol
- 2D/PCI device: guest drivers talk directly to VM backend

- 3D: uses the same DMA-based protocol + paged memory BAR
- 3D: the problem is now software-only: just follow the APIs
- 3D: remote OpenGL/DirectX efficiently
- 3D: convert Direct3D to OpenGL (in OS X)
- 3D: mix 2D and 3D effectively
- 3D: grasp all the detail of 2 decades of 3D APIs evolution
- 3D: work around host deficiencies and bugs

Video device emulation components

- Small portion of VMM, mostly VGA
- Dedicated thread in VM backend handles DMA requests
- 2D and 3D command stream parser
- OpenGL execution
- Direct3D translation and execution

- VGA/VESA BIOS

- Windows 9x basic driver
- Windows NT 4 basic driver
- Windows 2k/XP (XPDM) driver w/ DX9/OpenGL support
- Windows Vista/7/8 (WDDM 1.1) driver w/ DX9/OpenGL support
- Linux/Xorg driver, OpenGL library

Video paravirtualization: guest OpenGL on host OS X

- Straightforward approach: pass guest command stream to host as is
- OpenGL was client-server from the start, only need simple API marshaler
- Serialized calls are packed into DMA buffer, sent to host when need response
- Host unpacks parameters, makes actual GL calls
- Integration with window system: guest windows are PBuffers on host

- Marshaler code is autogenerated via PHP script from annotated gl.h/glext.h
- Annotation allows inserting hand-crafted code

- Performance is dealt with on per-case basis
- Most often slowdown is caused by guest application requesting GL response
- Autogenerated code grew by incorporating portions of GL state tracker

- OS X bugs/features leak into guest
- Suspend/resume of VM looses OpenGL state completely

Video paravirtualization: guest DirectX on host OS X(1)

- Implementation started with Direct3D 7 for Windows XP
- Got temporary boost by optional use of WINE libraries, dropped quickly
- Handily, XPDM provides the driver with already serialized command stream
- Resource management commands are separate from command stream
- Direct3D 7 is fixed function: convert state to GL and draw
- DirectDraw is a pain: requires direct access to surfaces

- Direct3D 8 introduces shaders: MS specified assembly-like byte-code
- Need to convert shaders to GLSL
- Fortunately, Shader Model 1/2/3 only have two types: float and boolean
- Mixing fixed function with shaders calls for shader generator on host

- Direct3D 9 introduces more smaller features and details, evolves from D3D8
- More surface formats, new shader instructions, events/queries

- Direct3D emulation evolved along with OpenGL and H/W growth in OS X

Video paravirtualization: DirectX on OS X performance

- Switching from guest to host once per frame is perfect
- Follow MS design, behave more like actual hardware
- Some optional DX/driver features better be supported, or else
- Data transfer efficiency matters, avoid copies, avoid pixel readback
- Minimise OpenGL state changes, expensive updates

- Analysis approach: look for idling pipeline stages, optimise where load is
- Use performance tools: profiler is king

- Hardware specific optimisations are unavoidable: NVIDIA vs ATI

Video paravirtualization: DirectX on OS X conformance

- Direct3D does not match OpenGL exactly, intentional
- Pixel center, resource formats and limits, shader language boundary cases
- OpenGL is mostly upload/readback, Direct3D is direct access (Lock())

- Direct3D API semantics are vast, not completely documented
- Capability bits is a pain to get right
- Microsoft DCT tests are necessary but insufficient
- In many cases applications define behavior, even exact
- Some applications expect either NVIDIA or ATI, fail otherwise
- Lots of specific scenarios

- Avoid OpenGL implementation corner cases and rarely used features
- OS X OpenGL bugs

Video paravirtualization in a perfect world

- 3D Hardware vendors could leverage their experience by providing VM drivers
- These drivers might talk directly to host drivers or even hardware
- Could take hardware capabilities into account, use advanced features
- Eliminate user-space API translation, save shader recompilation
- Hardware may have virtualization aides built in
- Overall the best approach from performance standpoint
- Take off work from VM developers

- There would be complications integrating video device with VM services
- VM looses hardware agnosticism
- VM developer looses part of control

Conclusions

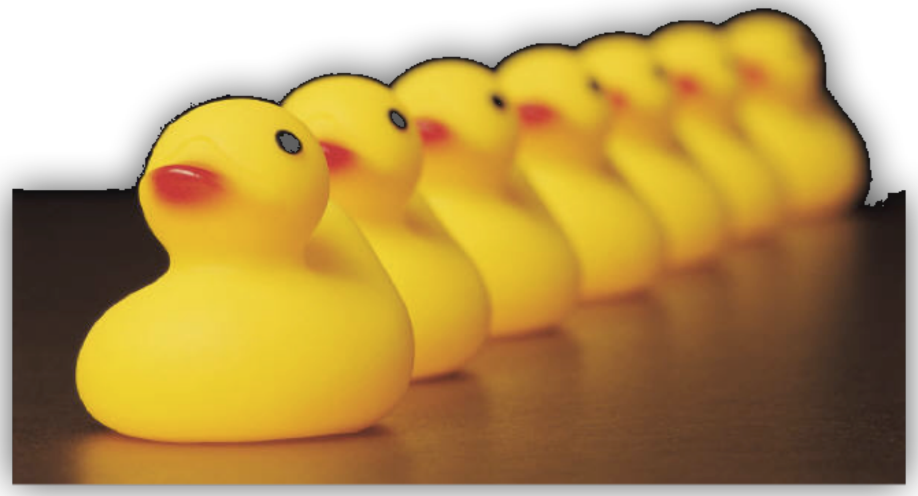
GPU allows parallel execution on the cost of memory. 2 best approaches to GPU support: GPU virtualization and graphics paravirtualization

Questions?

Assignment 7(common)

What kind of GPU virtualization approaches is supported by listed vendors:

- ✓ Vmware
- ✓ Xen
- ✓ Microsoft HyperV
- ✓ Parallels Desktop
- ✓ Linux KVM (+Qemu)



Projects

Project. GPU virtualization solutions overview

I've mentioned that Amazon supplies G2 instances. What are other examples of GPU virtualization solutions? And in Russia?

Project. GPU: API forwarding

Suppose you need to translate GL ES to OpenGL. Investigate the interface stability issues, estimate the project size (man-months), suppose the basic principles

Based on materials

- Chris McClanahan “History and Evolution of GPU Architecture”
- http://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/
- Lectures by Sergey Viktorov, Parallels
- Materials of open.gl
- http://www.linux-kvm.org/wiki/images/f/f3/01x08b-KVMGT-a.pdf
- Dowty “GPU Virtualization on VMware’s Hosted I/O Architecture”
- Kun Tian, “A Full GPU Virtualization Solution with Mediated Pass-Through”