INTRODUCTION TO MONTE CARLO RAY TRACING

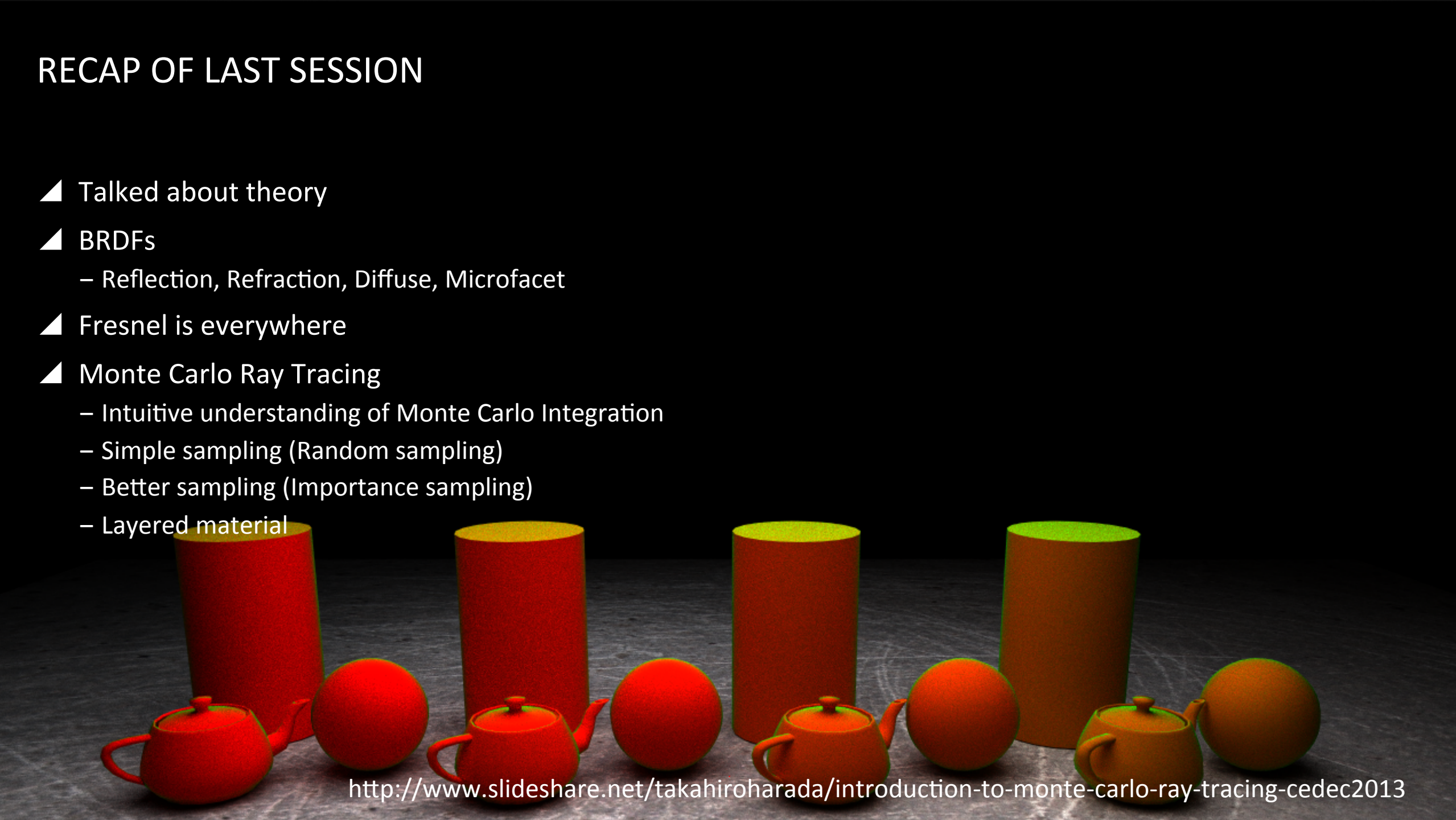
OPENCL IMPLEMENTATION

TAKAHIRO HARADA

9/2014

RECAP OF LAST SESSION

- Talked about theory
- BRDFs
  - Reflection, Refraction, Diffuse, Microfacet
- Fresnel is everywhere
- Monte Carlo Ray Tracing
  - Intuitive understanding of Monte Carlo Integration
  - Simple sampling (Random sampling)
  - Better sampling (Importance sampling)
  - Layered material

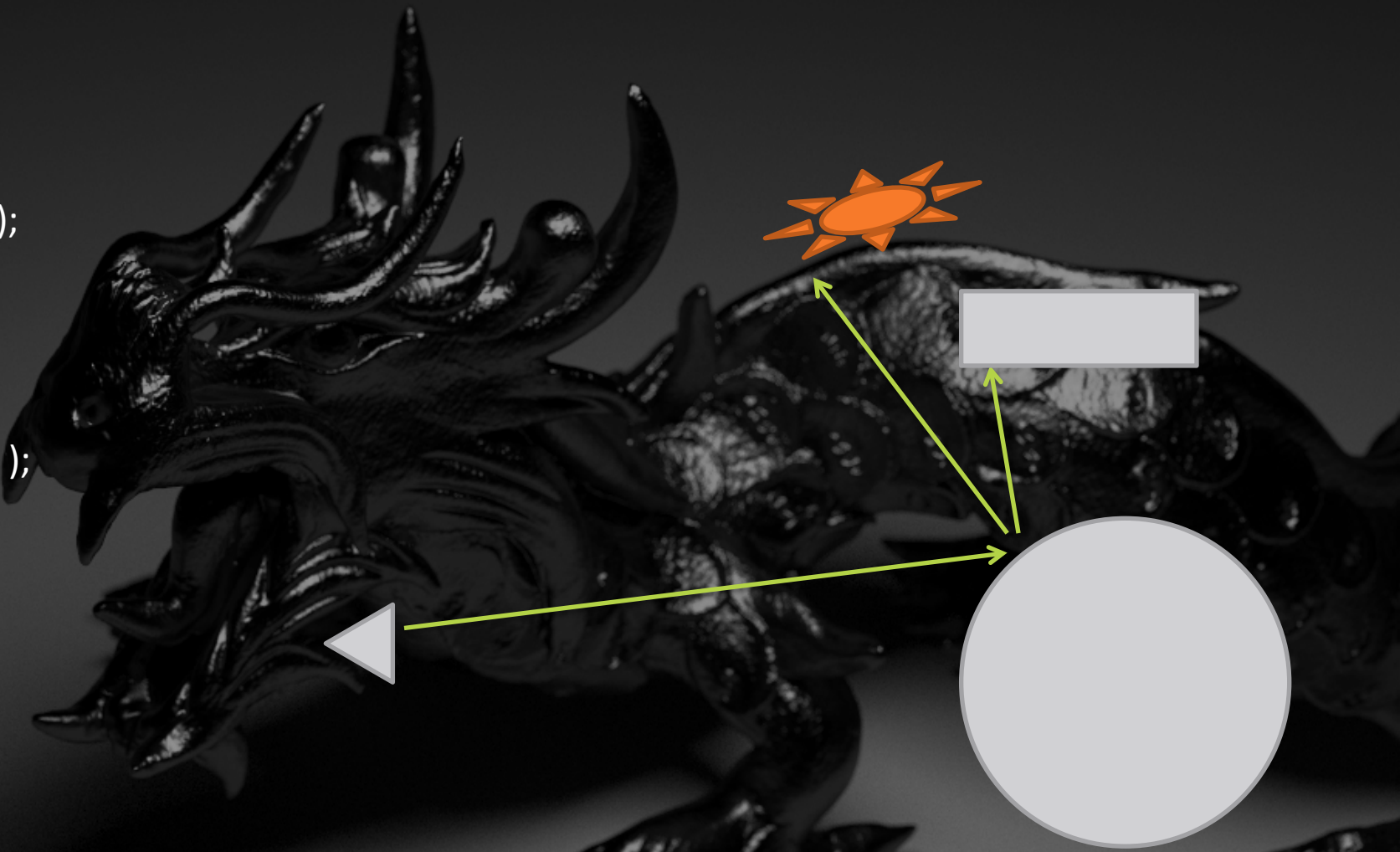


http://www.slideshare.net/takahiroharada/introduction-to-monte-carlo-ray-tracing-cedec2013

REVIEW
SIMPLE CPU MC RAY TRACER

Direct illumination

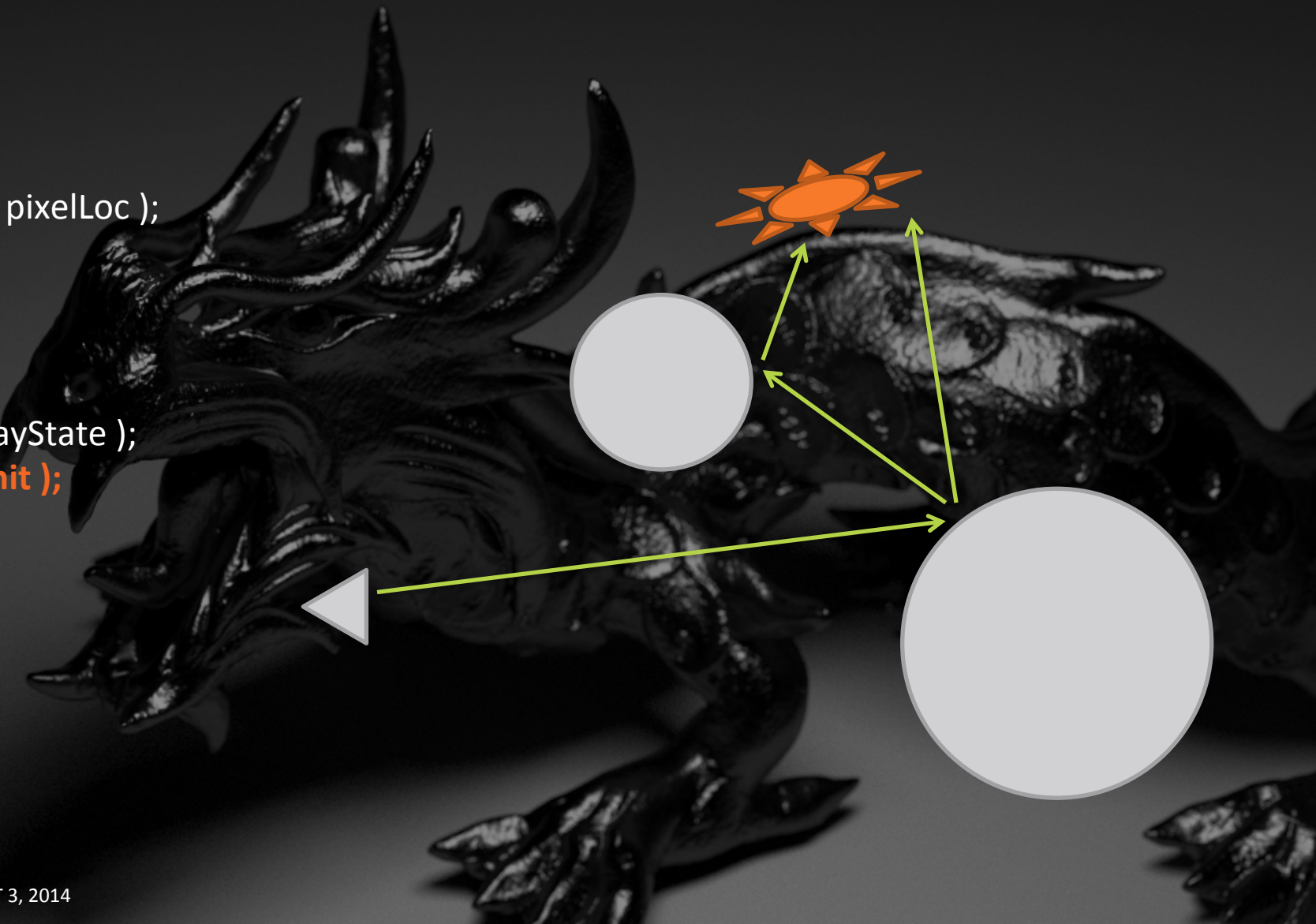
```
for( i, j )
{
   ray = PrimaryRayGen( camera, pixelLoc );

   {
      hit = Trace( ray );
      if( hit )
         fb( pixelLoc ) += EvaluateDI( ray, hit );

   }
}
```



REVIEW
SIMPLE CPU MC RAY TRACER

Indirect illumination

```
for( i, j )
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );
   while(1)
   {
      hit = Trace( ray );
      if( !hit ) break
      fb( pixelLoc ) += EvaluateDI( ray, hit, rayState );
      ray, rayState = sampleNextRay( ray, hit );
   }
}
```



REVIEW

SIMPLE CPU MC RAY TRACER

Direct illumination

```
for( i, j )
{
   ray = PrimaryRayGen( camera, pixelLoc );

   {
      hit = Trace( ray );
      if( hit )
         fb( pixelLoc ) += EvaluateDI( ray, hit );

   }
}
```

Indirect Illumination

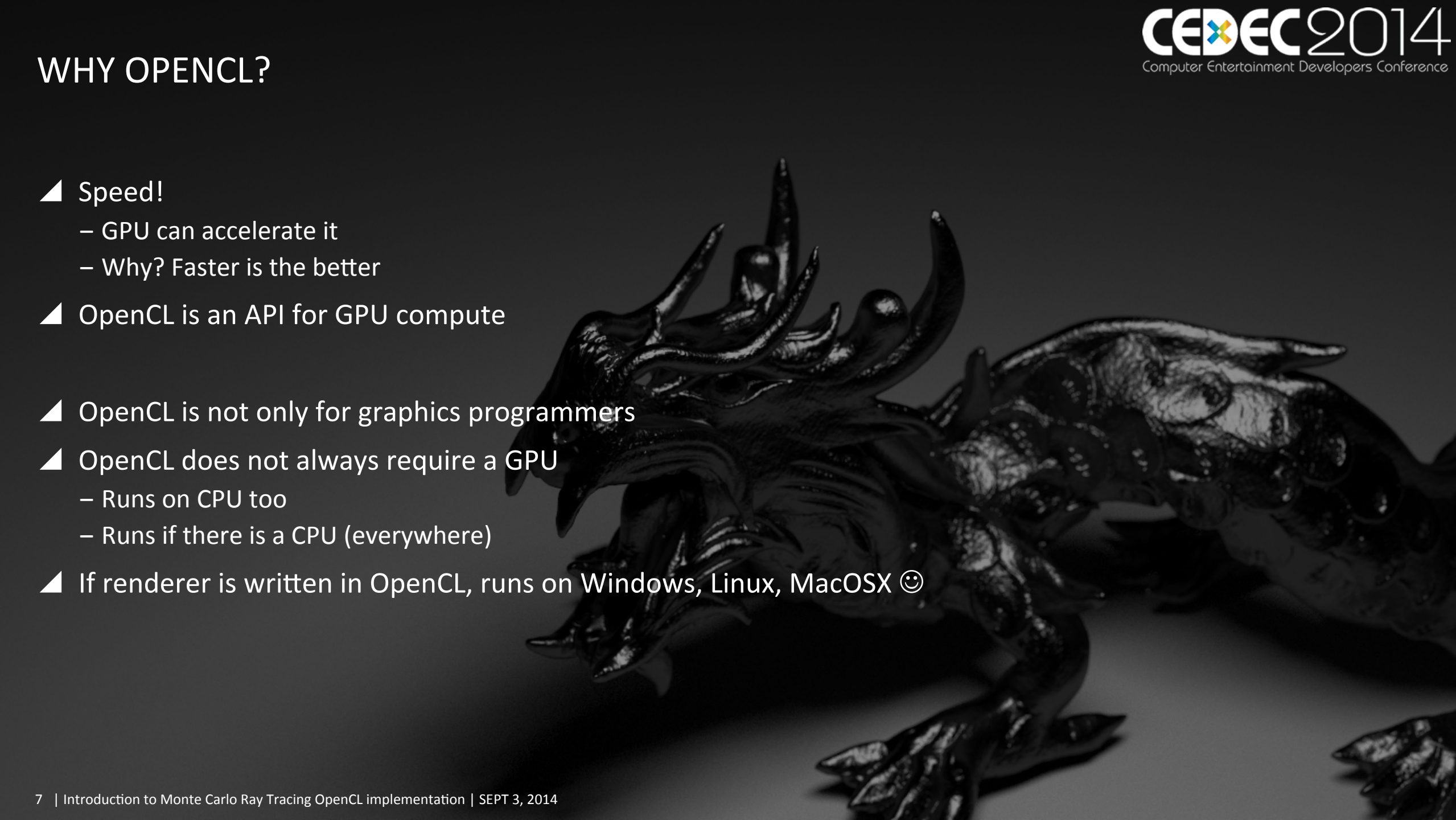
```
for( i, j )
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );
   while(1)
   {
      hit = Trace( ray );
      if( !hit ) break
      fb( pixelLoc ) += EvaluateDI( ray, hit, rayState );
      ray, rayState = sampleNextRay( ray, hit );
   }
}
```

COMPARISON

Direct illumination

Indirect illumination

WHY OPENCL?



- Speed!
  - GPU can accelerate it
  - Why? Faster is the better
- OpenCL is an API for GPU compute

- OpenCL is not only for graphics programmers
- OpenCL does not always require a GPU
  - Runs on CPU too
  - Runs if there is a CPU (everywhere)
- If renderer is written in OpenCL, runs on Windows, Linux, MacOSX ☺


PORTING TO OPENCL
(FIRST ATTEMPT)

THINGS TO BE DONE
DATA STRUCTURE

- No pointer in OpenCL*
- Change pointer to index
- Stored in a flat memory

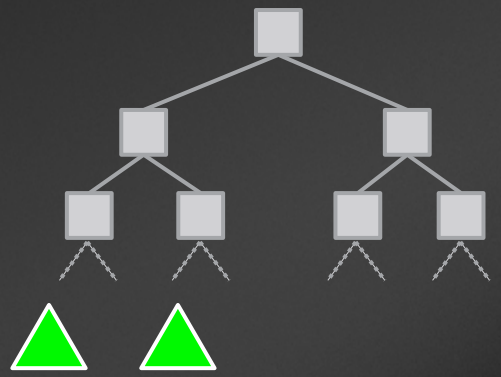
- Not suited for partial update

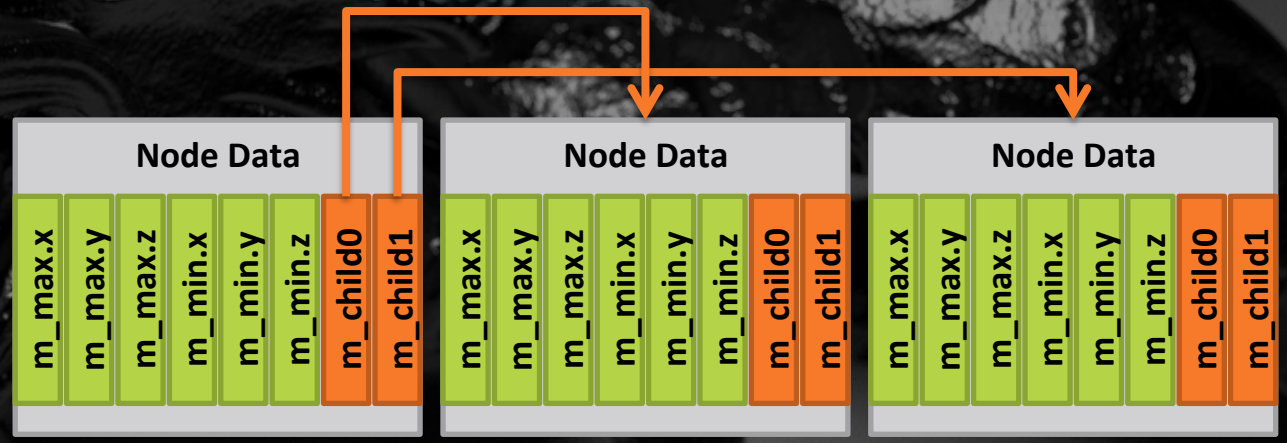
*Shared Virtual Memory (OpenCL 2.0)

THINGS TO BE DONE
DATA STRUCTURE

- Node data for a binary tree
  - Spatial acceleration structure (BVH)
  - Shading network

- Buffer<NodeData> nodeData;



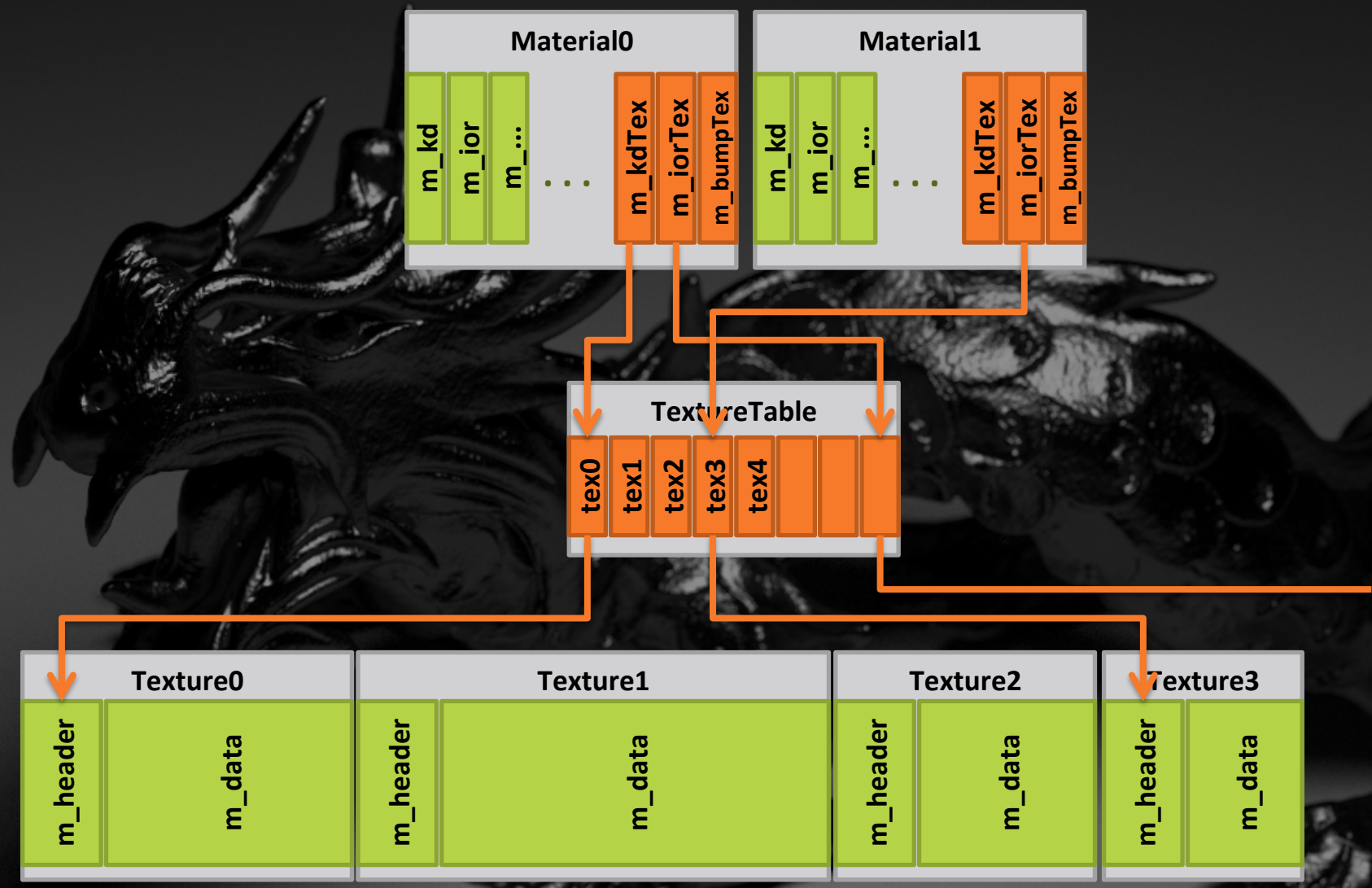


THINGS TO BE DONE

DATA STRUCTURE

- Material
  - Texture entry

- Buffer<Material> material;
- Buffer<char> texData;
- Buffer<uint> texTable;



THINGS TO BE DONE

WRITING OPENCL KERNEL

CPU code

```
for( i, j )
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );
   while(1)
   {
      hit = Trace( ray );
      if(!hit ) break;
      fb( pixelLoc ) += EvaluateDI( ray, hit, rayState );
      ray, rayState = sampleNextRay( ray, hit );
   }
}
```

OpenCL kernel

```
__kernel
void PtKernel(__global ...)
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );
   while(1)
   {
      hit = Trace( ray );
      if(!hit ) return;
      fb( pixelLoc ) += EvaluateDI( ray, hit, rayState );
      ray, rayState = sampleNextRay( ray, hit );
   }
}
```

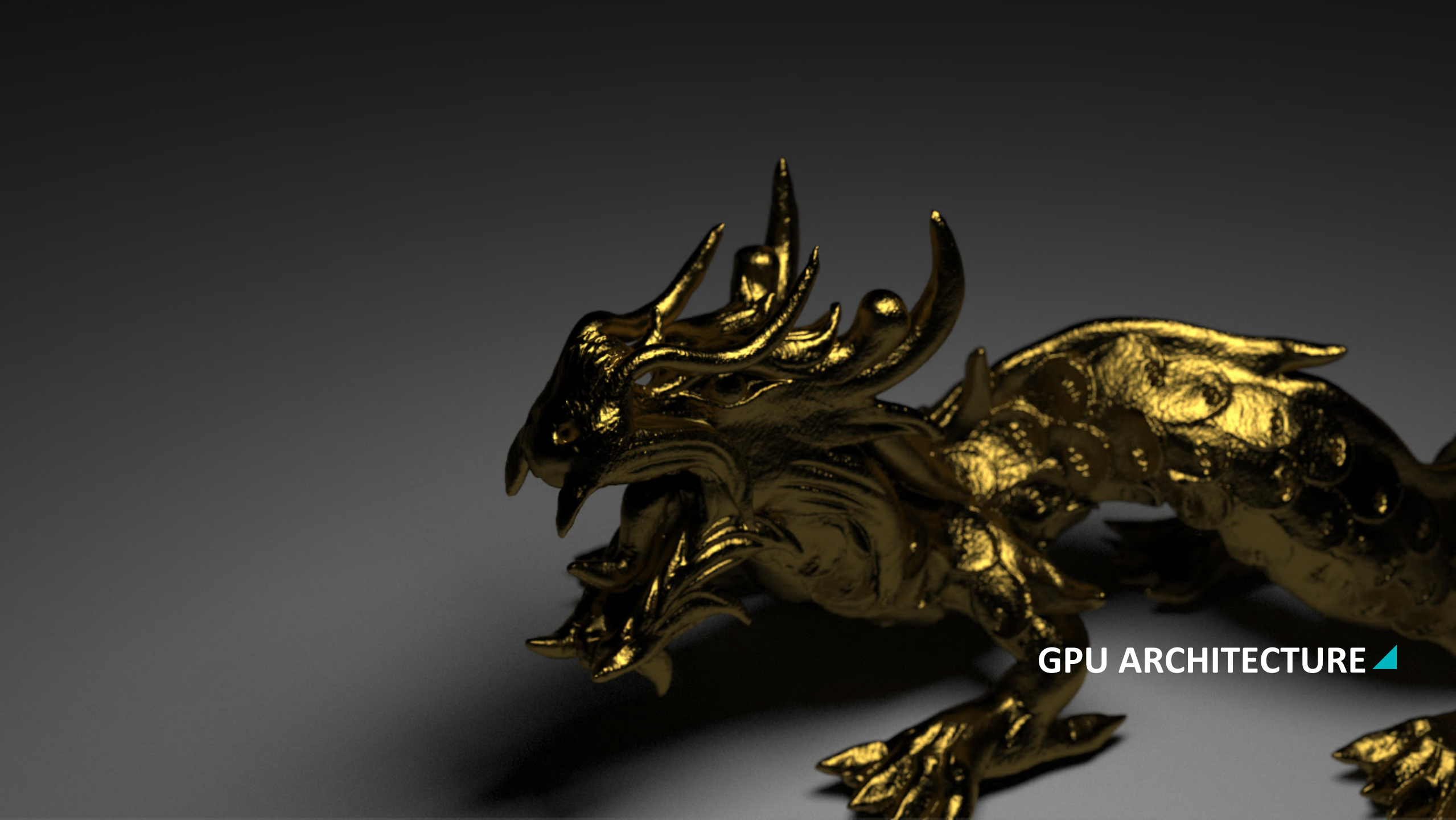
IT WORKS BUT…

- This approach is simple
- But a lot of issues

DRAWBACKS
PERFORMANCE

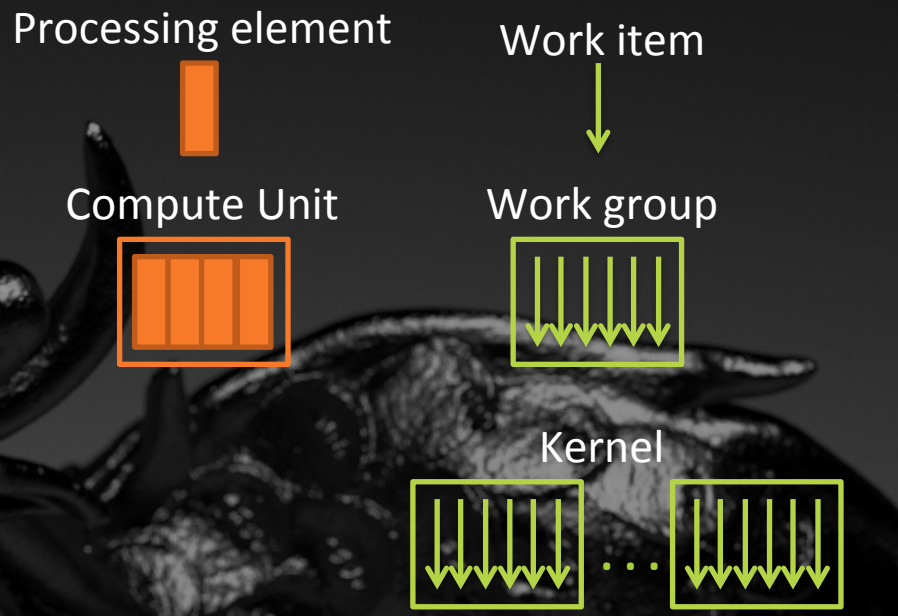
- Likely not utilize hardware efficiently
  - SIMD divergence
  - GPU occupancy (latency)

- Maintainability
- Extendibility, Portability


GPU ARCHITECTURE

OPENCL ON CPU

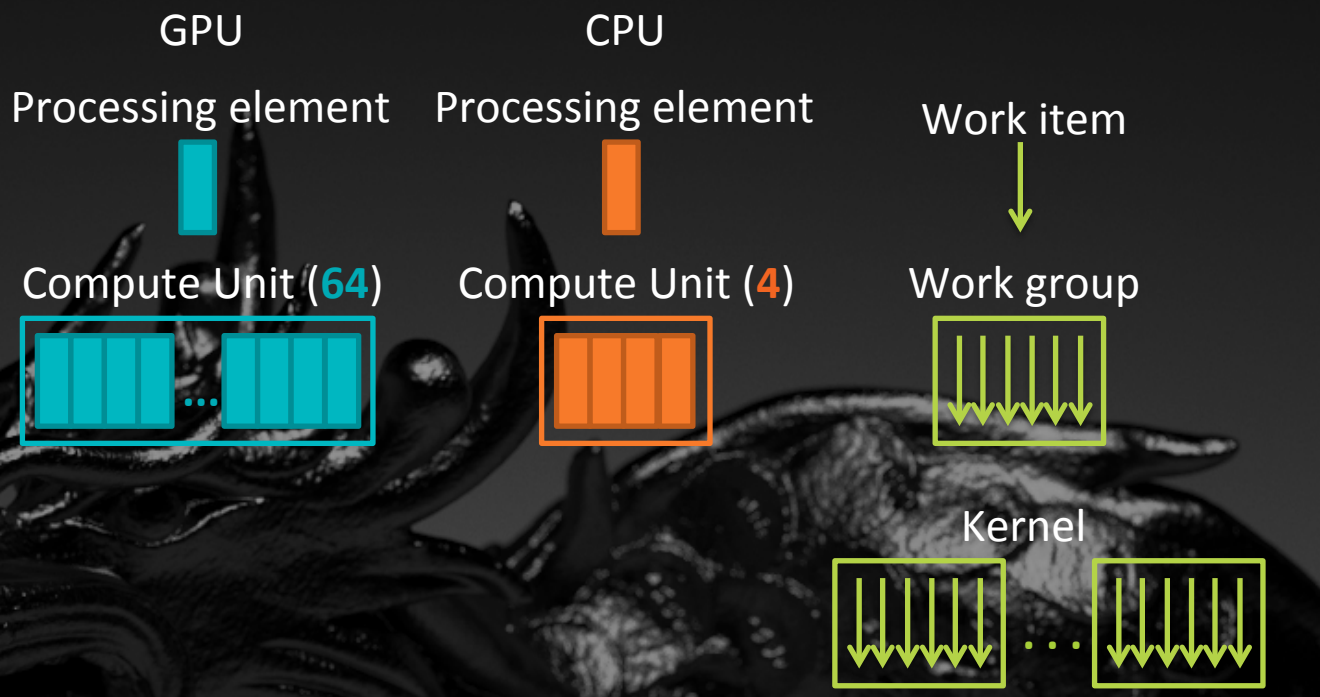
- **Processing element** executes **Work item** (thread)
  - A SIMD lane (4*)
- **Compute unit** executes **Work group** (thread group)
  - A core (8*)
  - # of processing elements != # of work items
- **Compute device** executes **Kernel** (shader)
  - A CPU
  - # of compute units != # of work groups



* On AMD FX-8350

GPU VS CPU

- **Processing element** executes **Work item**
  - A SIMD lane (64*)
- **Compute unit** executes **Work group**
  - A SIMD engine (44x4*)
  - # of processing elements != # of work items
- **Compute device** executes **Kernel**
  - A GPU
  - # of compute units != # of work groups



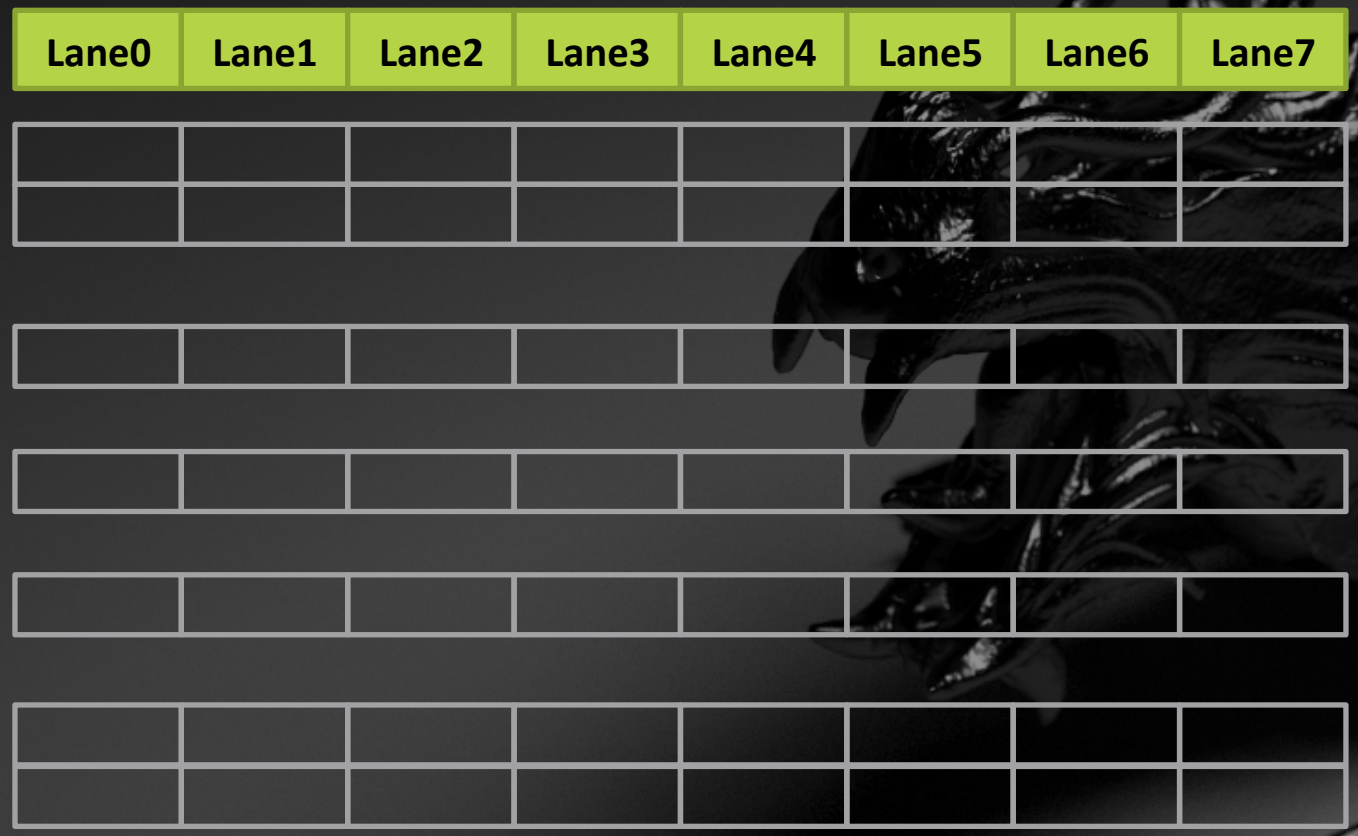
* On AMD Radeon R9 290X

HIGH LEVEL DESCRIPTION

- Today's GPU is similar to a CPU (if you look at very high level)
  - GPU is an extremely wide CPU
  - Many cores
  - Wide SIMD
- AMD Radeon R9 290X GPU
  - **176** = 44x4 SIMD engines (cores)
  - **64** wide SIMD
- But different in
  - SIMD width (very wide)
  - Limited local resources
  - Strategy to hide latency
- Knowing those are the key to exploit the performance

SIMD DIVERGENCE

- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
    int value = 0;
    int a = computeA();
    if( a == 0 )
        value = compute0();
    else if( a == 1 )
        value = compute1();
    else if( a == 2 )
        value = compute2();
    else if( a == 3 )
        value = compute3();
    return value;
}
```

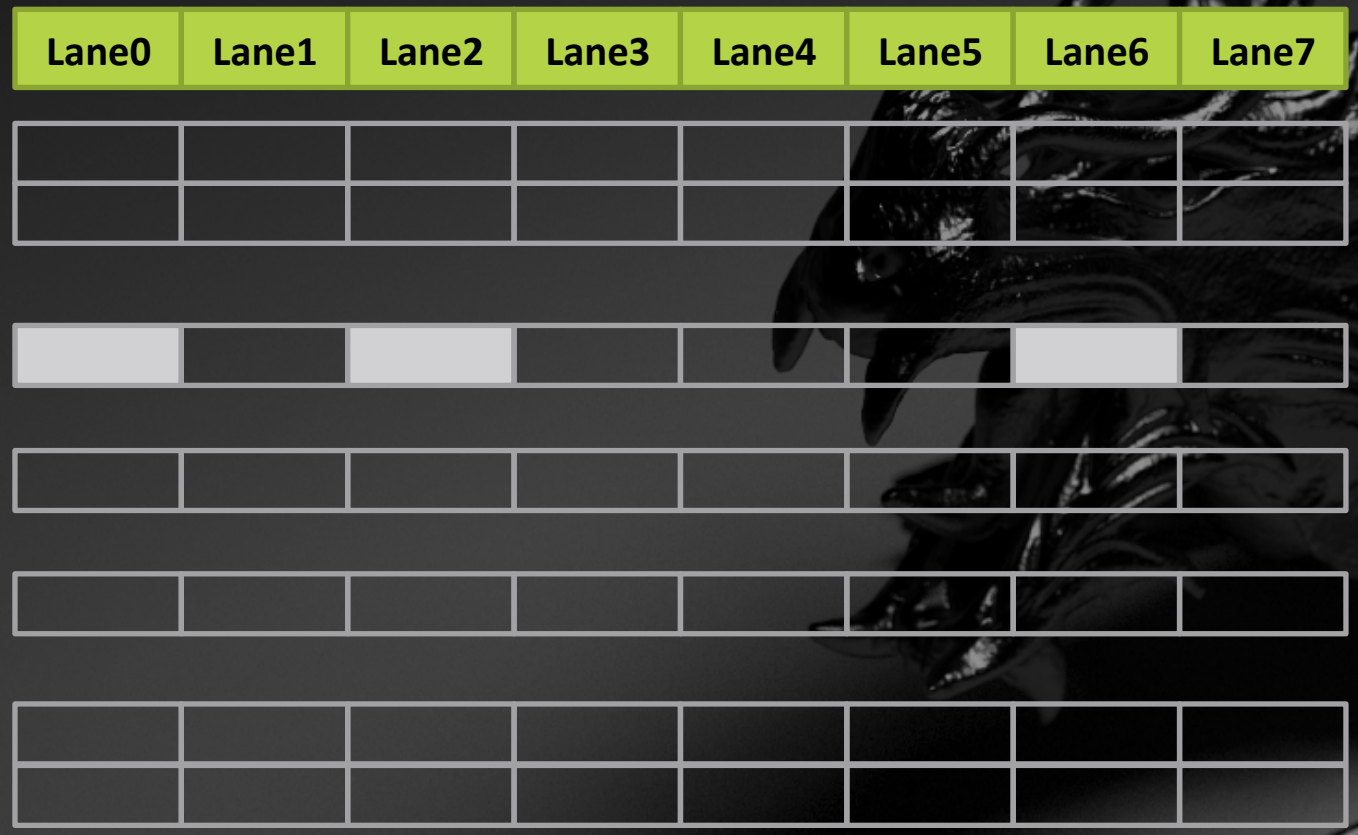
SIMD DIVERGENCE

- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)


```
int funcA()
{
  int value = 0;
  int a = computeA();
  if( a == 0 )
    value = compute0();
  else if( a == 1 )
    value = compute1();
  else if( a == 2 )
    value = compute2();
  else if( a == 3 )
    value = compute3();
  return value;
}
```

SIMD DIVERGENCE

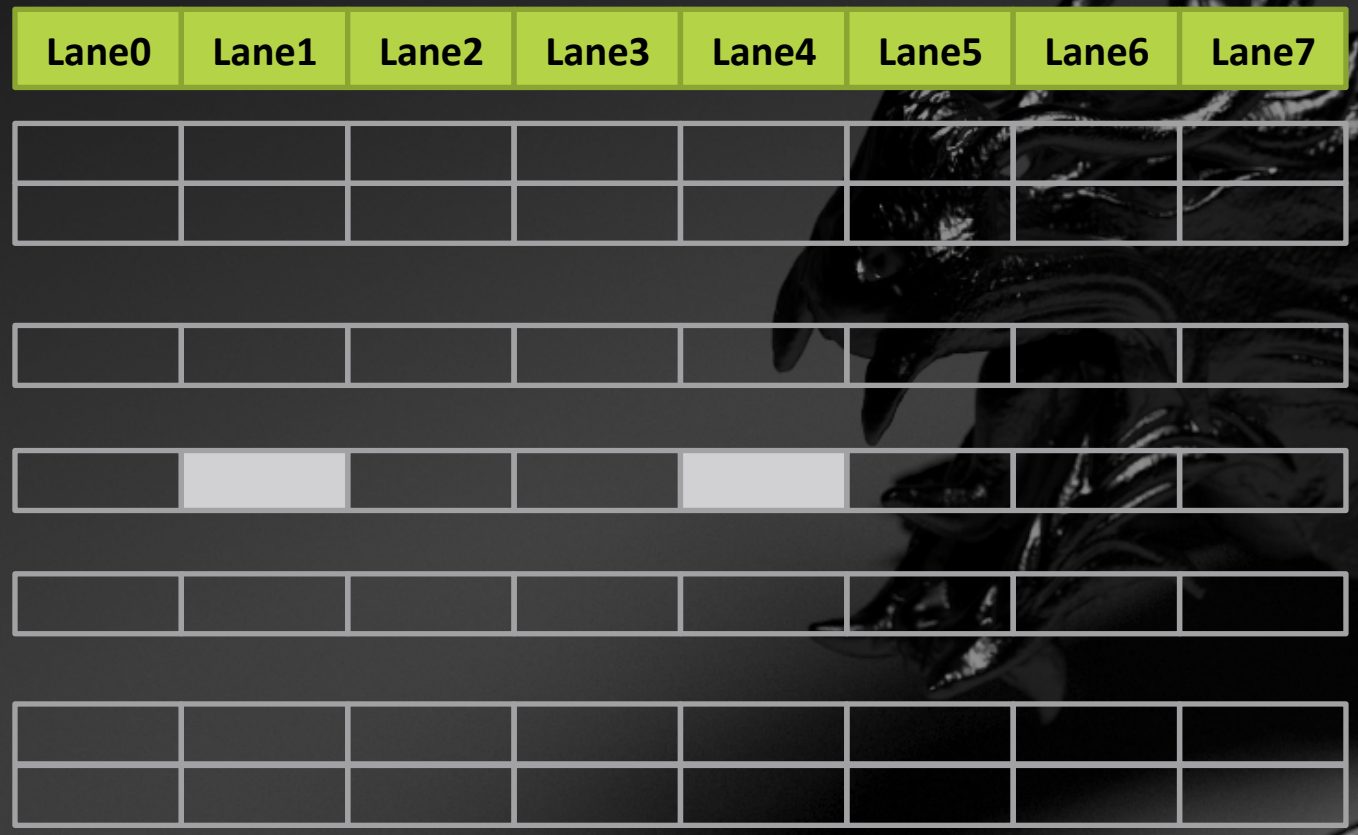
- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
  int value = 0;
  int a = computeA();
  if( a == 0 )
    value = compute0();
  else if( a == 1 )
    value = compute1();
  else if( a == 2 )
    value = compute2();
  else if( a == 3 )
    value = compute3();
  return value;
}
```

SIMD DIVERGENCE

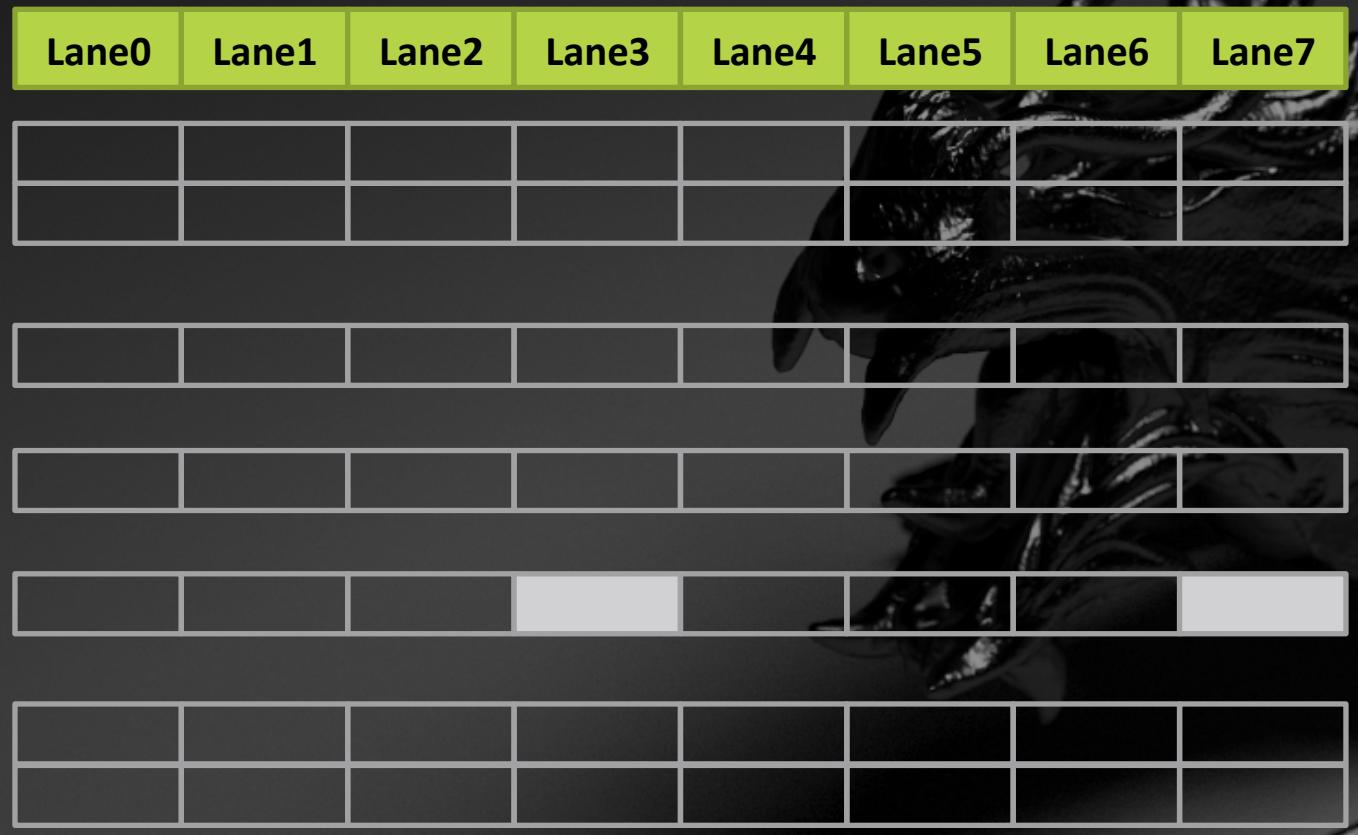
- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
  int value = 0;
  int a = computeA();
  if( a == 0 )
    value = compute0();
  else if( a == 1 )
    value = compute1();
  else if( a == 2 )
    value = compute2();
  else if( a == 3 )
    value = compute3();
  return value;
}
```

SIMD DIVERGENCE

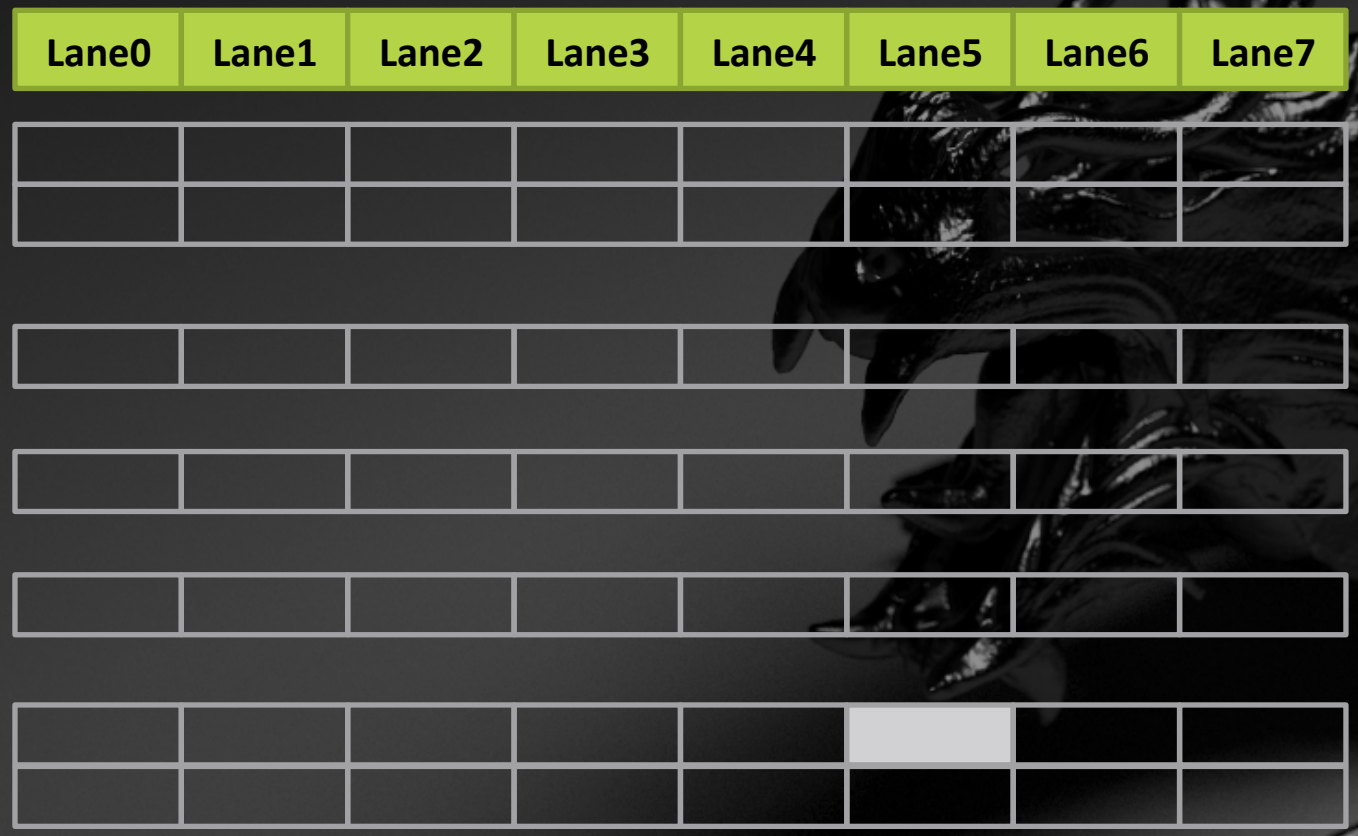
- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
  int value = 0;
  int a = computeA();
  if( a == 0 )
    value = compute0();
  else if( a == 1 )
    value = compute1();
  else if( a == 2 )
    value = compute2();
  else if( a == 3 )
    value = compute3();
  return value;
}
```

SIMD DIVERGENCE

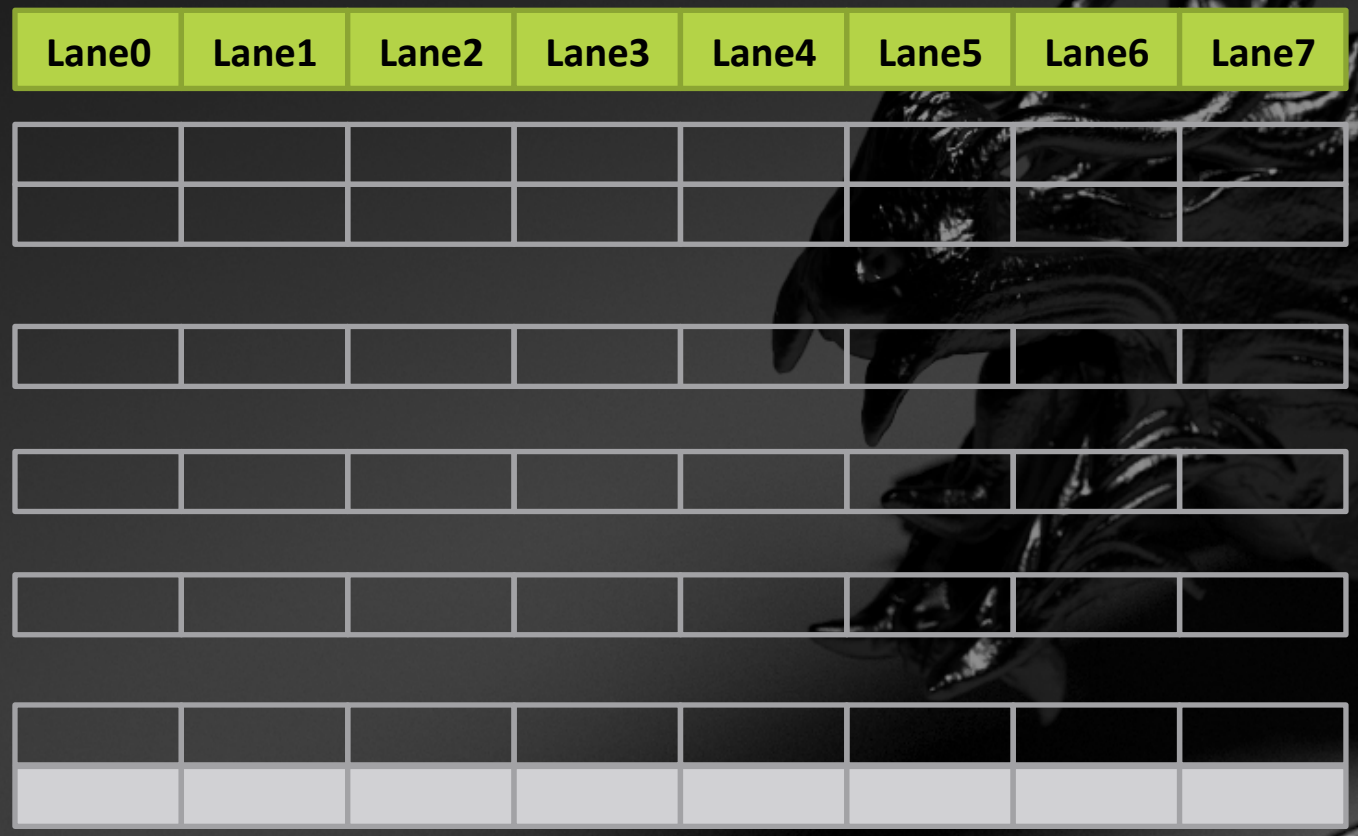
- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
  int value = 0;
  int a = computeA();
  if( a == 0 )
    value = compute0();
  else if( a == 1 )
    value = compute1();
  else if( a == 2 )
    value = compute2();
  else if( a == 3 )
    value = compute3();
  return value;
}
```

SIMD DIVERGENCE

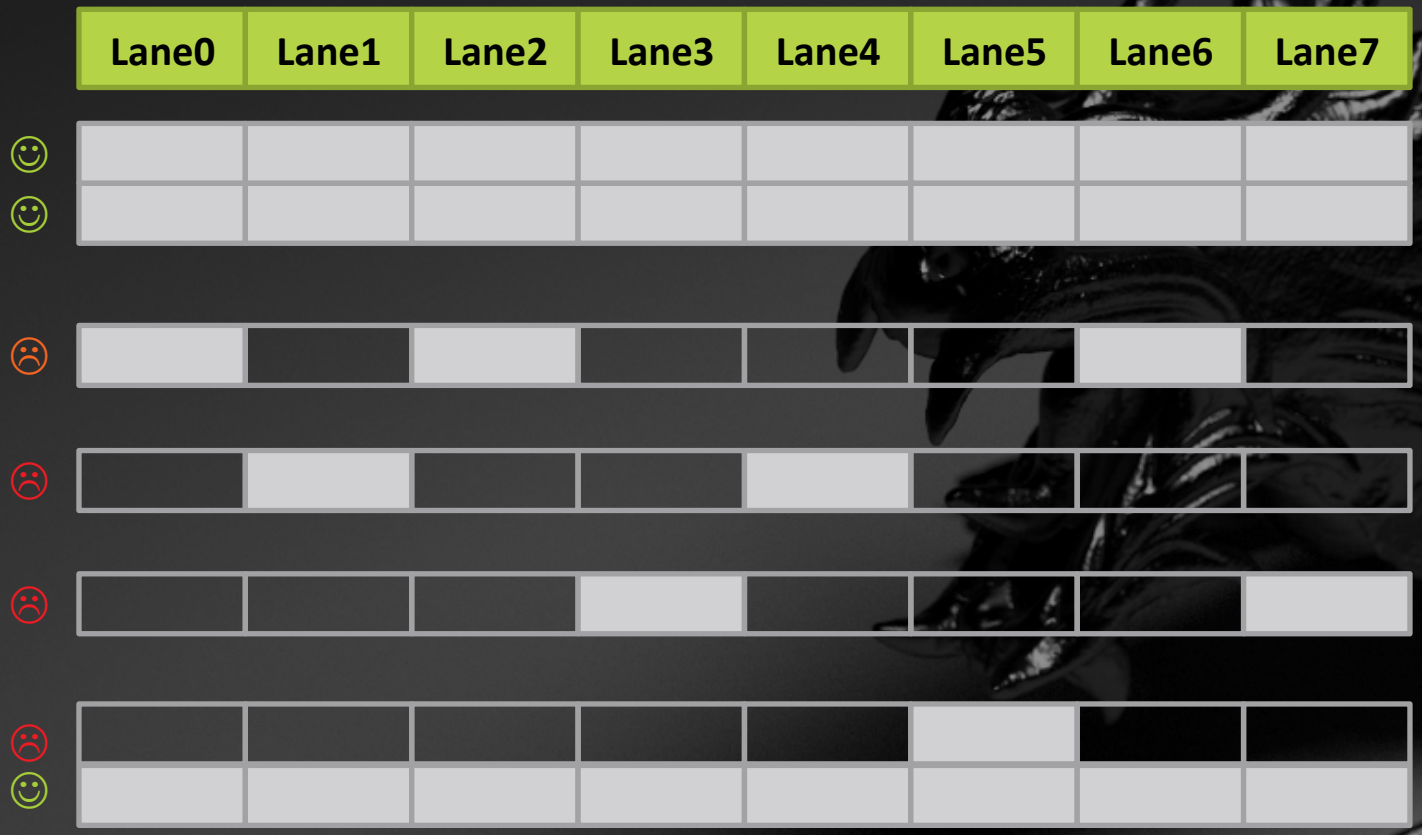
- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
  int value = 0;
  int a = computeA();
  if( a == 0 )
    value = compute0();
  else if( a == 1 )
    value = compute1();
  else if( a == 2 )
    value = compute2();
  else if( a == 3 )
    value = compute3();
  return value;
}
```

WIDE SIMD EXECUTION

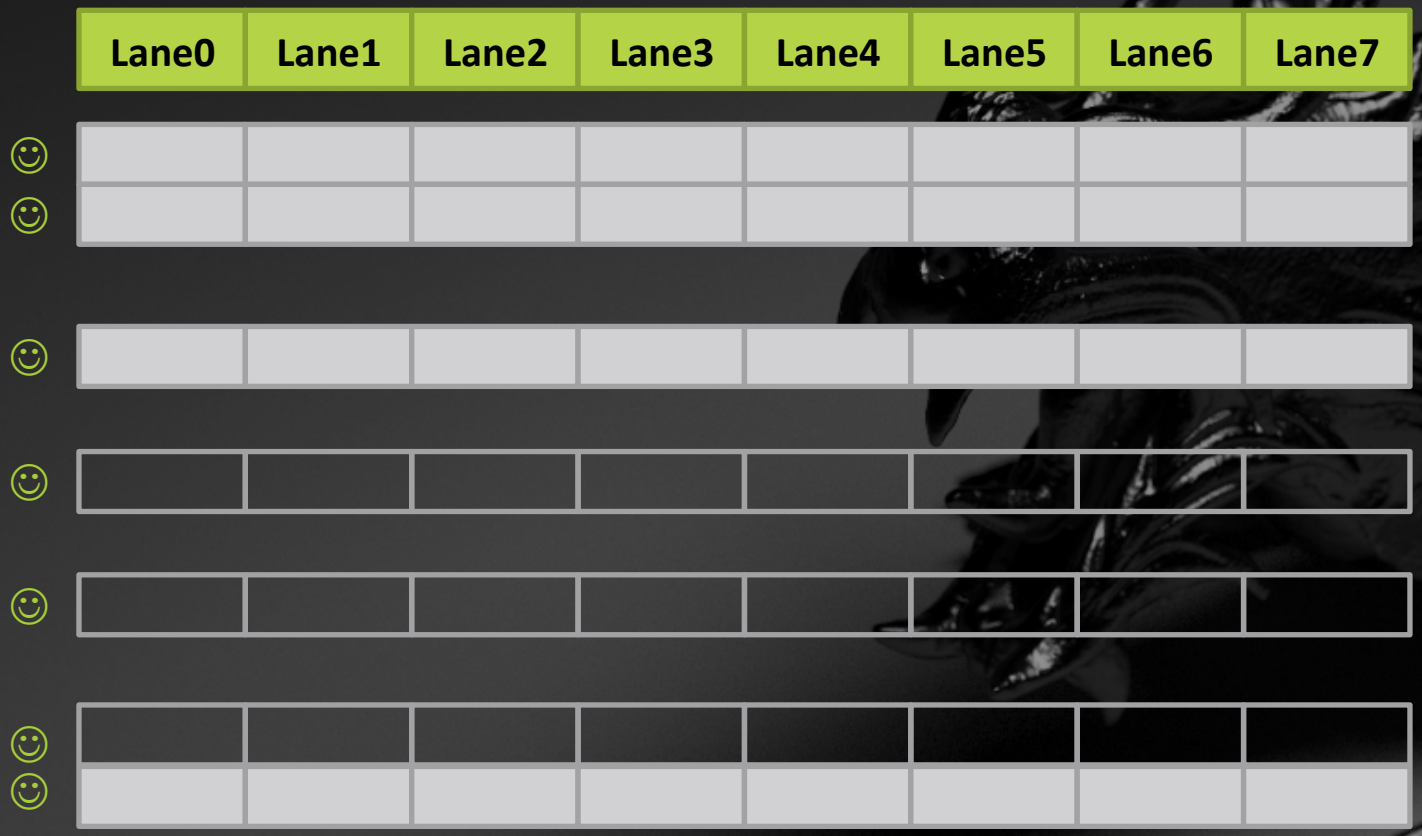
- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
    int value = 0;
    int a = computeA();
    if( a == 0 )
        value = compute0();
    else if( a == 1 )
        value = compute1();
    else if( a == 2 )
        value = compute2();
    else if( a == 3 )
        value = compute3();
    return value;
}
```

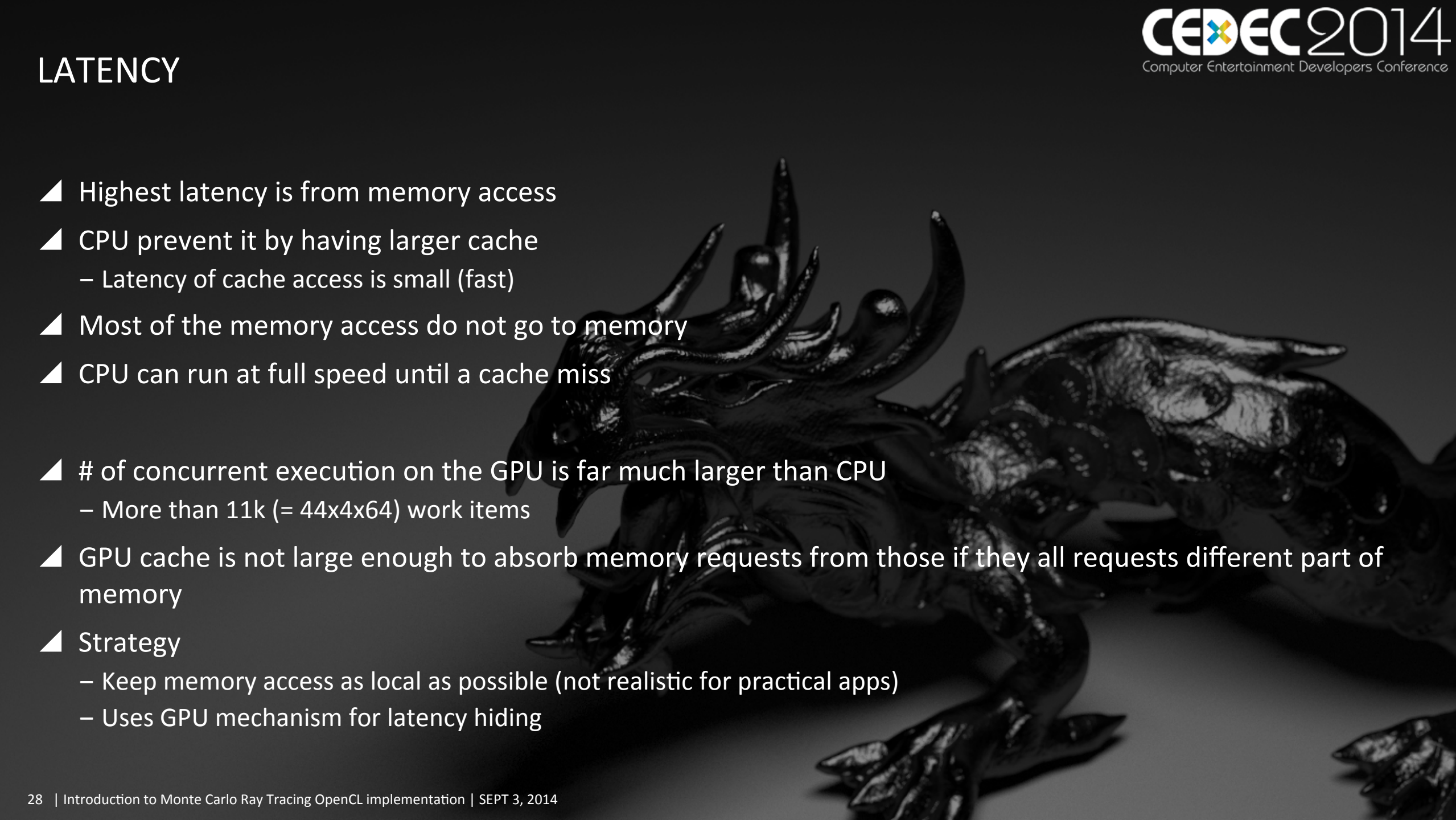
SIMD DIVERGENCE

- SIMD execution = Program counter is shared among SIMD lanes
- If it diverges in branches, HW utilization decreases a lot (Gets easier to diverge on wide SIMD)




```
int funcA()
{
    int value = 0;
    int a = computeA();
    if( a == 0 )
        value = compute0();
    else if( a == 1 )
        value = compute1();
    else if( a == 2 )
        value = compute2();
    else if( a == 3 )
        value = compute3();
    return value;
}
```

LATENCY



- Highest latency is from memory access
- CPU prevent it by having larger cache
  - Latency of cache access is small (fast)
- Most of the memory access do not go to memory
- CPU can run at full speed until a cache miss

- # of concurrent execution on the GPU is far much larger than CPU
  - More than 11k (= 44x4x64) work items
- GPU cache is not large enough to absorb memory requests from those if they all requests different part of memory
- Strategy
  - Keep memory access as local as possible (not realistic for practical apps)
  - Uses GPU mechanism for latency hiding

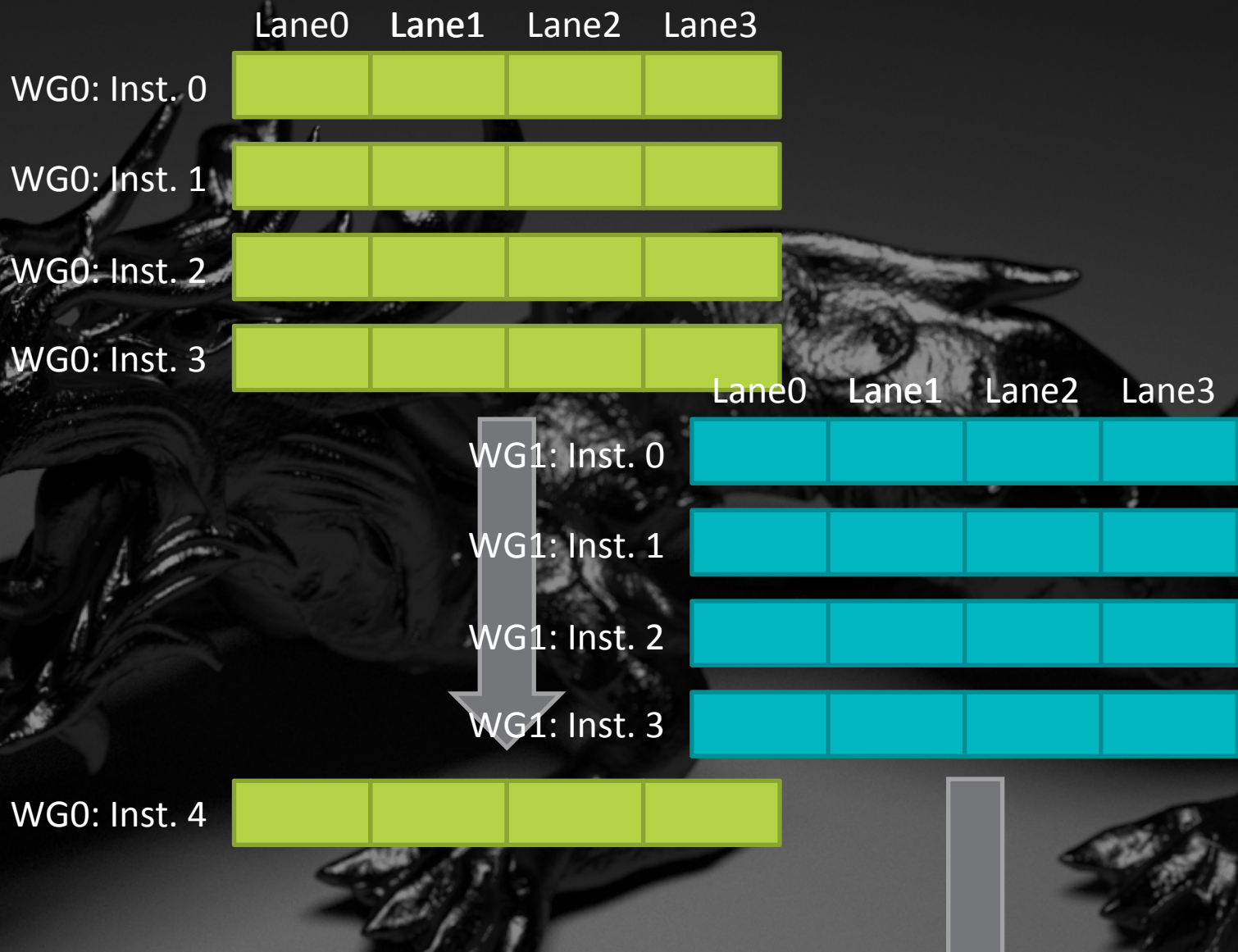
GPU LATENCY HIDING

- GPU can execute at full speed if there are only ALU instructions (Inst. 0 - 2) *
- Stalls on memory access instruction (Inst. 3)

* Can hide latency using logical vector

GPU LATENCY HIDING

- When stalled, switch to another work group
- Could fill the stall with instructions from WG1
- A SIMD of GPU needs to process multiple WGs at the same time to hide latency (or maximize its throughput)



HOW MANY WGS CAN WE EXECUTE PER SIMD

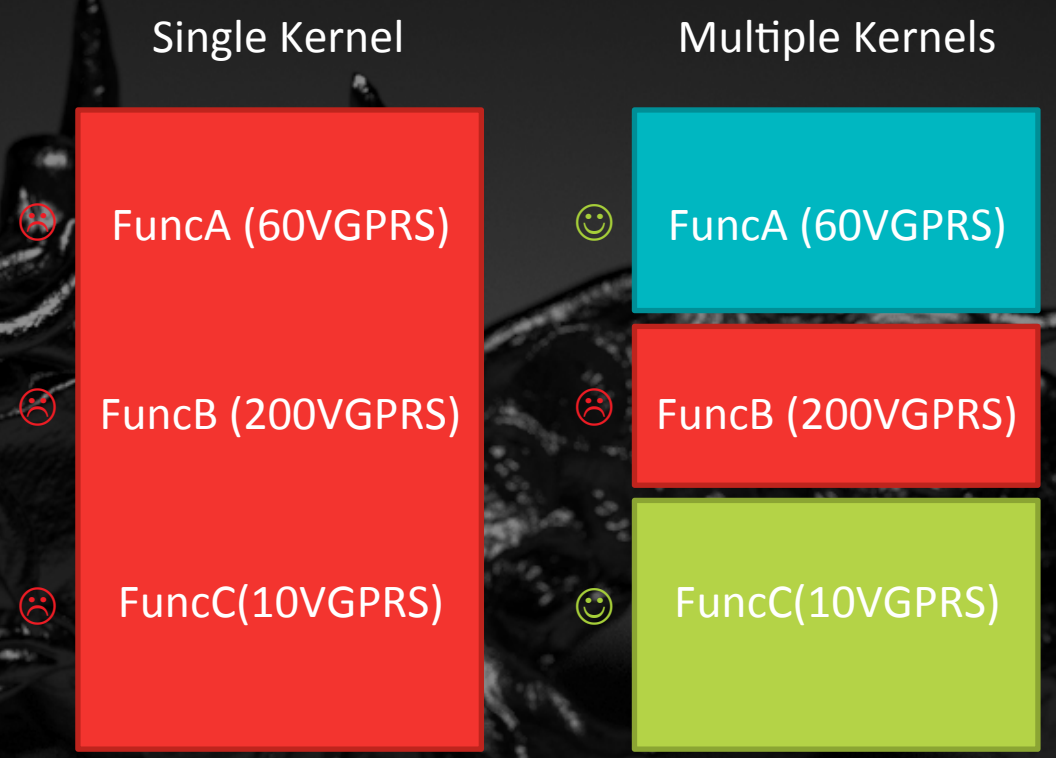
- 10 wavefronts (64WIs) per SIMD is the max
- It depends on local resource usage of the kernel
- VGPR usage is often the problem
- Share 256 VGPRs among n work groups
  - 1 wavefront, 256VGPRs ☹☹
  - 2 wavefronts, 128VGPRs
  - 4 wavefronts, 64VGPRs ☺
  - 10 wavefronts, 25VGPRs
- Share 16KB LDS among n work groups
  - 1 work group, 16KB ☹☹
  - 2 work group, 8KB
  - 4 work group, 4KB ☺

- VGPRs
  - Registers used by vector ALUs
  - 64KB/SIMD
  - 256 VGPRs/SIMD lane (= 64KB/64/4)
- LDS (Local data share)
  - 64KB/CU (CU == 4SIMD)
  - 32KB/SIMD

ADVICE TO REDUCE VGPR PRESSURE

GET MORE PERFORMANCE FROM GPU

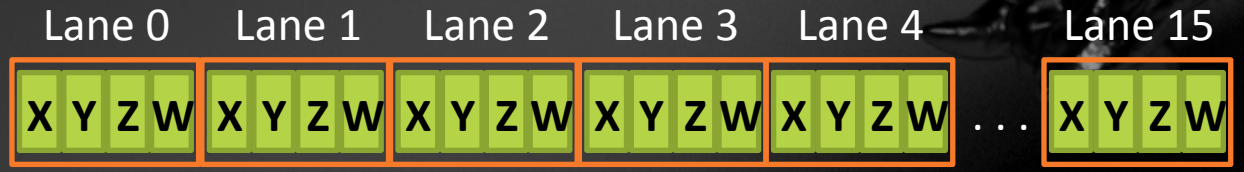
- Don’t write a large kernel
- If the program can be split into several pieces, split them into several kernels
  - Single kernel approach
    - VGPR usage of the kernel is 200 = max(60, 200, 10)
    - 1 wavefront per SIMD
    - Bad for latency hiding
  - Multiple kernel approach
    - FuncA: 4 wavefronts per SIMD
    - FuncB: 1 wavefronts per SIMD
    - FuncC: 10 wavefronts per SIMD
    - FuncB is bad, but FuncA, FuncC runs fast
- Helps compiler too



WHAT IS THE SCALAR ARCHITECTURE?

VLIW4 (NI), VLIW5 (EG)

- Best for vector computation
- Low efficiency on scalar computation
- Physical concurrent execution
  - 16 work items
  - 4 ALU operations each
  - Total: 16x4 ALU operations
- 1 SIMD is running in a CU
- Difficult to fill xyzw
- If not filled, we waste HW cycle



Scalar Architecture (SI, CI)

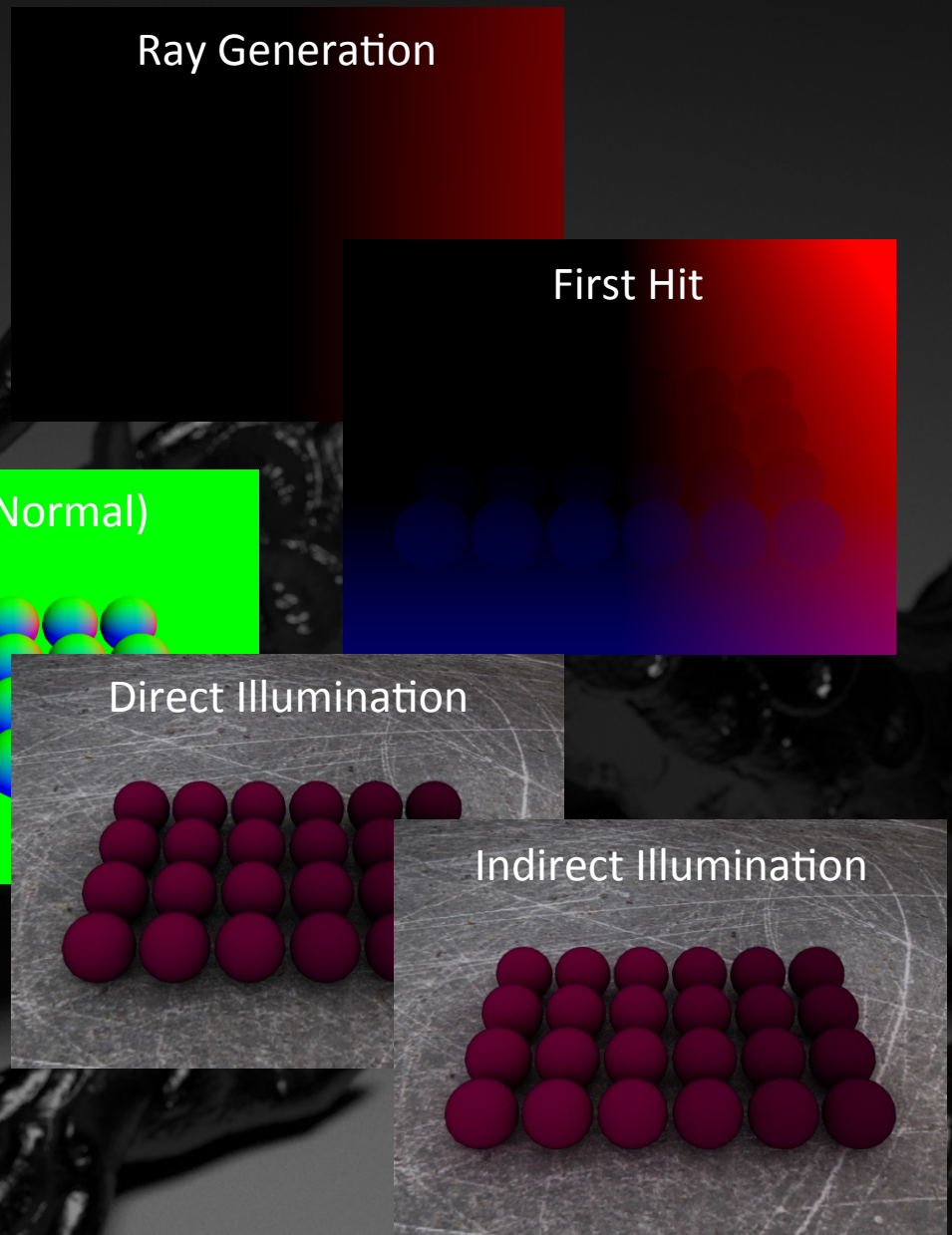
- Good for both vector and scalar computation
- Need more work groups to fill GPU
- Physical concurrent execution
  - 16x4 work items
  - 1 ALU operation each
  - Total: 16x4 ALU operations
- 4 SIMDs are running in a CU
- 4x more work groups are necessary to fill HW

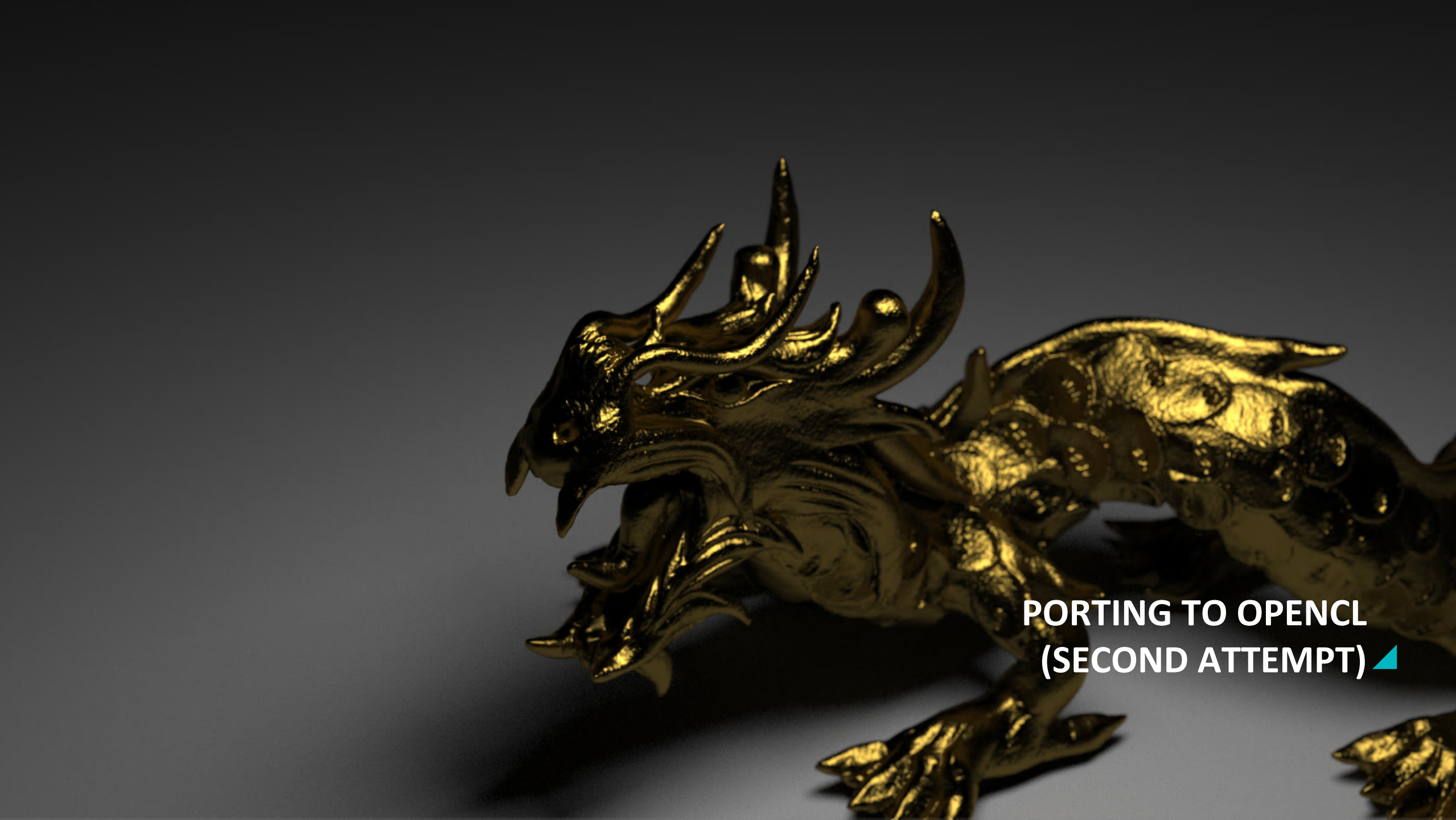
ANOTHER SOLUTION

- Splitting computation into multiple kernels
  - Primary ray gen kernel
  - Trace kernel
  - Evaluate DI kernel
  - Etc
- Better HW utilization
  - Less divergence
  - Higher HW occupancy

OTHER BENEFITS?

- Maintainability
  - Debug is not as easy as we do on C, C++
  - Not all compilers are mature
    - Can hit to a compiler bug, which is hard to debug
    - Helps compiler
  - By splitting kernels, we can isolate the issue
  - If the code is developed by many people, this is important
- Extendibility, Portability
  - Easy to extend features
  - Primary Ray Gen Kernel
    - Add another camera projection
  - Ray Casting Kernel
    - Easy to add another primitives (e.g., vector displacement)
    - Take it out for physics ray casting queries




PORTING TO OPENCL
(SECOND ATTEMPT)

SPLITTING KERNELS
TRANSFORMING CPU CODE

Naïve CPU implementation

```
forAll()
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );
   while(1)
   {
      hit = Trace( ray );
      if( !hit )
         break;
      fb( pixelLoc ) += EvaluateDI( ray, hit, rayState );
      ray, rayState = sampleNextRay( ray, hit );
   }
}
```

Preparing for OpenCL implementation

```
{
   forAll() PrimaryRayGenKernel();
   while(1)
   {
      forAll() TraceKernel();
      if( !any( hits ) )
         break;
      forAll() SampleLightKernel();
      forAll() TraceKernel();
      forAll() AccumulateDIKernel();
      forAll() SampleNextRayKernel();
   }
}
```

Each for loop => A kernel execution

SPLITTING KERNELS

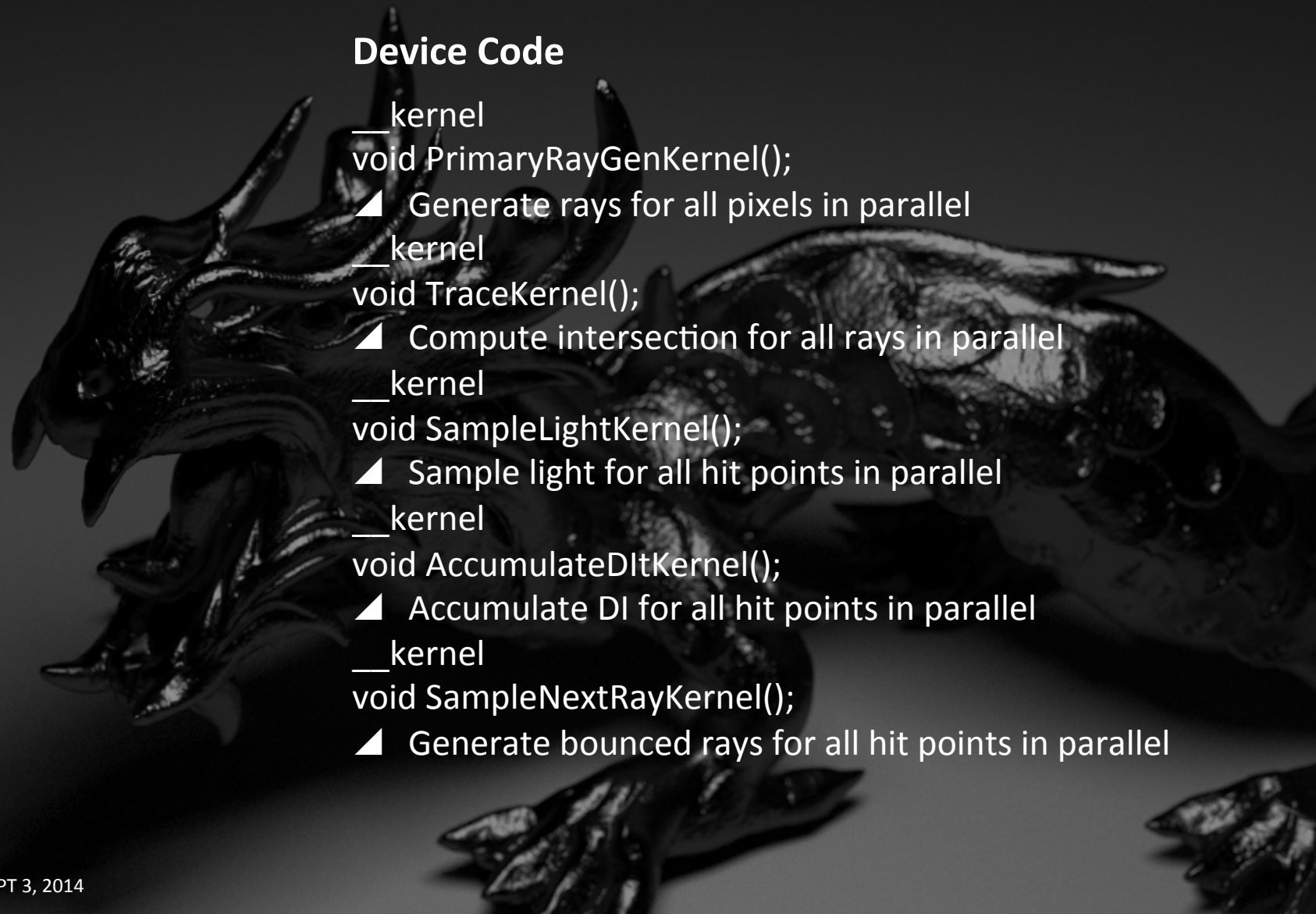
CPU implementation

```
forAll()
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );
   while(1)
   {
      hit = Trace( ray );
      if( !hit )
         break;
      fb( pixelLoc ) += EvaluateDI( ray, hit, rayState );
      ray, rayState = sampleNextRay( ray, hit );
   }
}
```

Host code

```
{
   launch( PrimaryRayGenKernel );
   while(1)
   {
      launch( TraceKernel );
      if( !any( hits ) )
         break;
      launch( SampleLightKernel );
      launch( TraceKernel );
      launch( AccumulateDIKernel );
      launch( SampleNextRayKernel );
   }
}
```

SPLITTING KERNELS



Host Code

```
{
   launch( PrimaryRayGenKernel );
   while(1)
   {
      launch( TraceKernel );
      if( !any( hits ) )
         break;
      launch( SampleLightKernel );
      launch( TraceKernel );
      launch( AccumulateDIKernel );
      launch( SampleNextRayKernel );
   }
}
```

Device Code

```
__kernel
void PrimaryRayGenKernel();
```

◢ Generate rays for all pixels in parallel

```
__kernel
void TraceKernel();
```

◢ Compute intersection for all rays in parallel

```
__kernel
void SampleLightKernel();
```

◢ Sample light for all hit points in parallel

```
__kernel
void AccumulateDItKernel();
```

◢ Accumulate DI for all hit points in parallel

```
__kernel
void SampleNextRayKernel();
```

◢ Generate bounced rays for all hit points in parallel

DESIGN
LOCALIZE BRANCH

Camera Type

```
{
   launch( PrimaryRayGenKernel );
   while(1)
   {
      launch( TraceKernel );
      if( !any( hits ) )
         break;
      launch( SampleLightKernel );
      launch( TraceKernel );
      launch( AccumulateDIKernel );
      launch( SampleNextRayKernel );
   }
}
```

Brdf Type

```
{
   launch( PrimaryRayGenKernel );
   while(1)
   {
      launch( TraceKernel );
      if( !any( hits ) )
         break;
      launch( SampleLightKernel );
      launch( TraceKernel );
      launch( AccumulateDIKernel );
      launch( SampleNextRayKernel );
   }
}
```

DESIGN
RAY STATE

- Cannot keep state between kernels
- Ray state needs to be saved to/restored from global memory

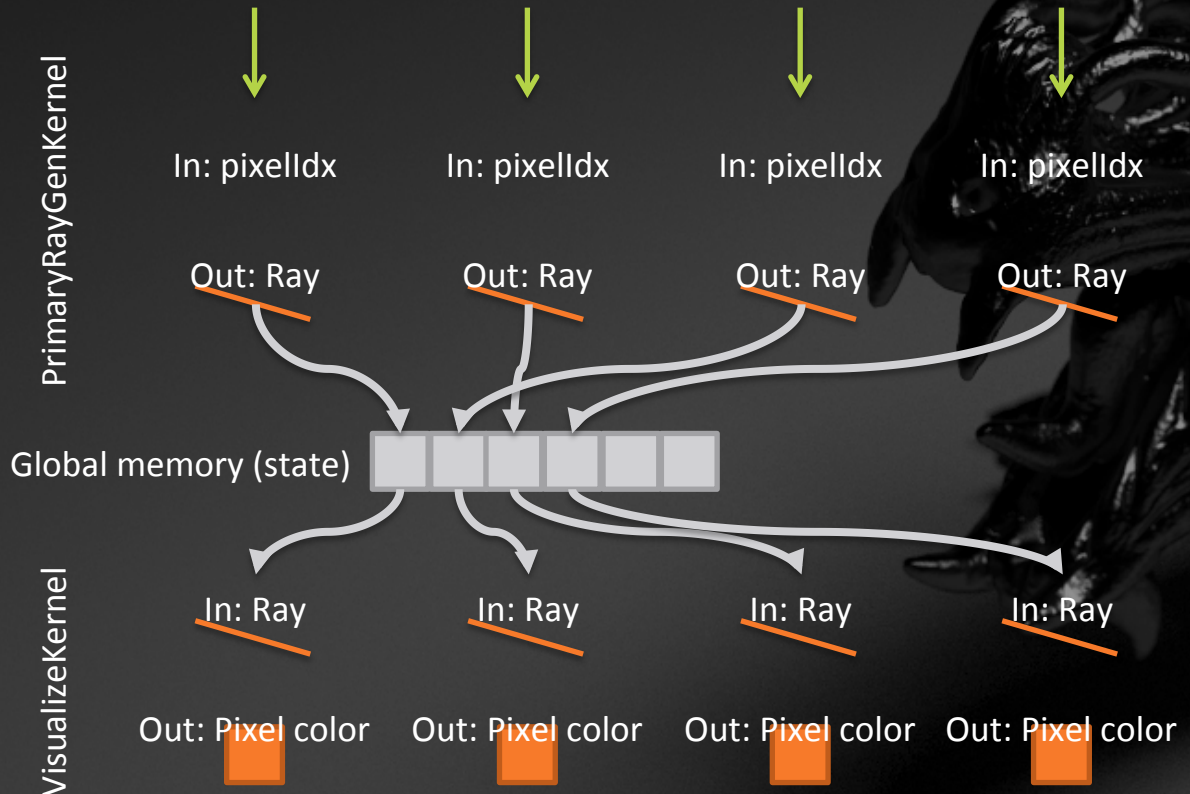
```
struct RayState
{
   float4 m_throughput;
   int2 m_randomNumber;
   int m_pixelIdx;
};
```

- Example
  - Ray generation + ray direction visualization

```
__kernel
void PrimaryRayGenKernel(__global ...)
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );

   int dst = atom_inc( &gRayCount );
   gRay[dst] = ray;
   gRayState[dst] = rayState; // save
}

__kernel
void VisualizeRayKernel(__global ...)
{
   RayState s = gRayState[get_global_id(0)]; // restore
   Ray ray = gRay[get_global_id(0)];

   gFb[s.m_pixelIdx] = Ray_getDir( ray );
}
```

DESIGN
RAY STATE

- Cannot keep state between kernels
- Ray state needs to be saved to/restored from global memory

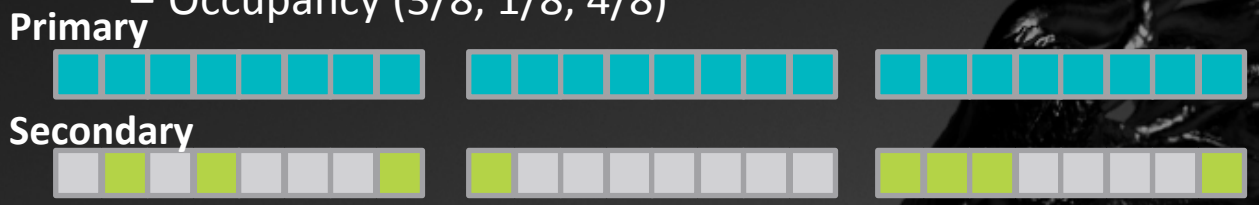


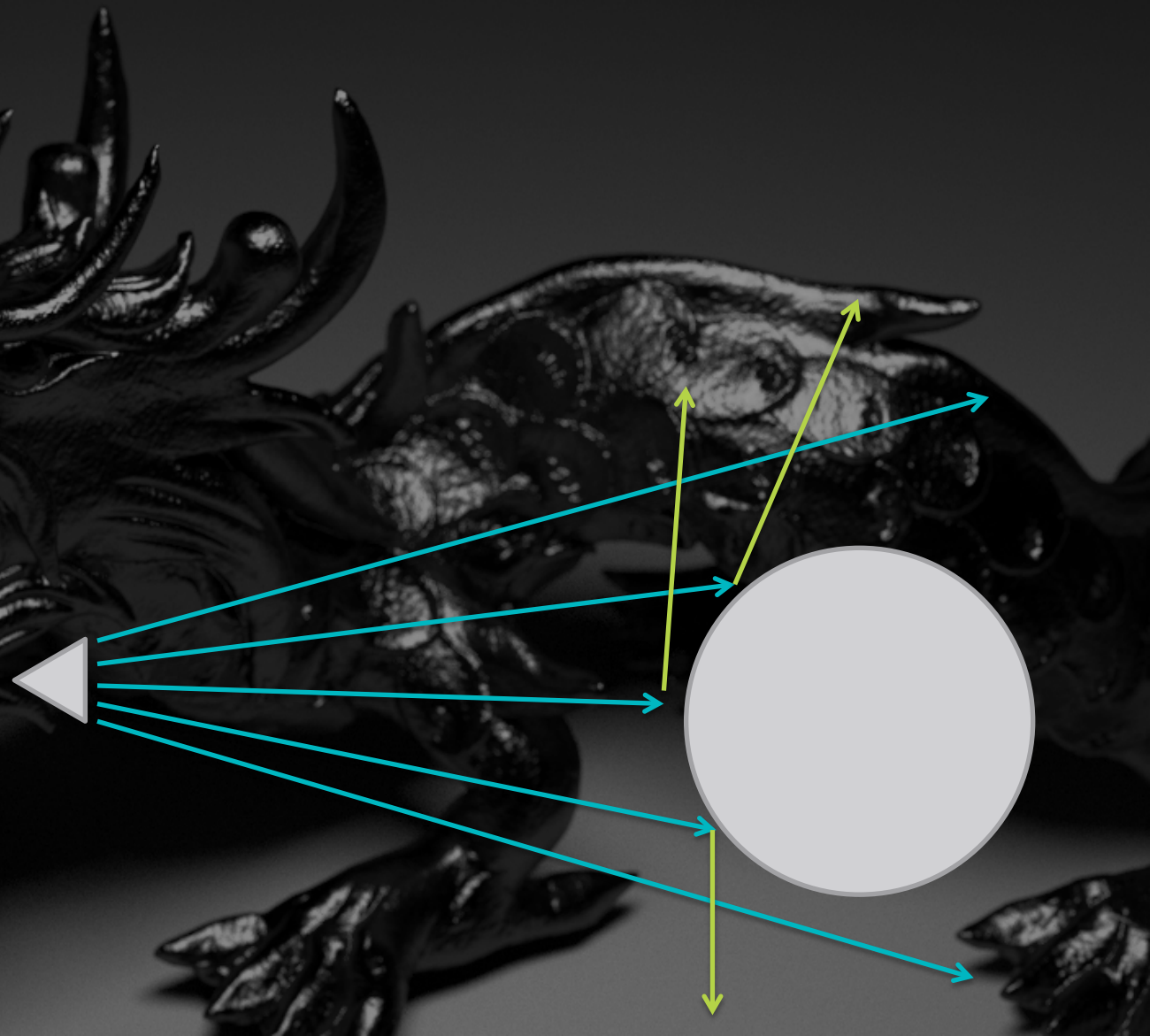
- Example
  - Ray generation + ray direction visualization

```
__kernel
void PrimaryRayGenKernel(__global ...)
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );

   int dst = atom_inc( &gRayCount );
   gRay[dst] = ray;
   gRayState[dst] = rayState; // save
}

__kernel
void VisualizeRayKernel(__global ...)
{
   RayState s = gRayState[get_global_id(0)]; // restore
   Ray ray = gRay[get_global_id(0)];

   gFb[s.m_pixelIdx] = Ray_getDir( ray );
}
```

RAY COMPACTION

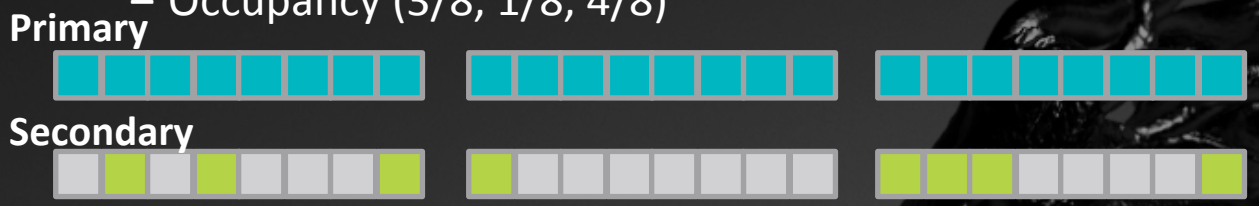
- Sparse data lowers SIMD utilization
- Without compaction
  - 3 SIMD executions
  - Occupancy (3/8, 1/8, 4/8)



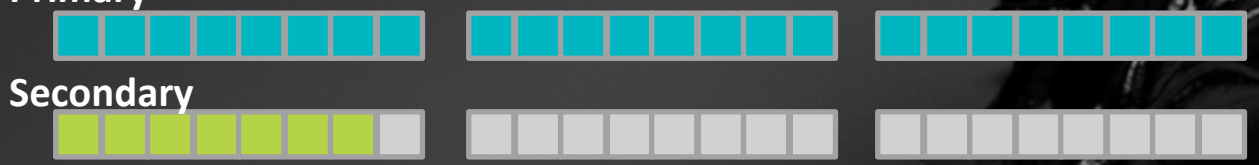


RAY COMPACTION

- Sparse data lowers SIMD utilization
- Without compaction
  - 3 SIMD executions
  - Occupancy (3/8, 1/8, 4/8)



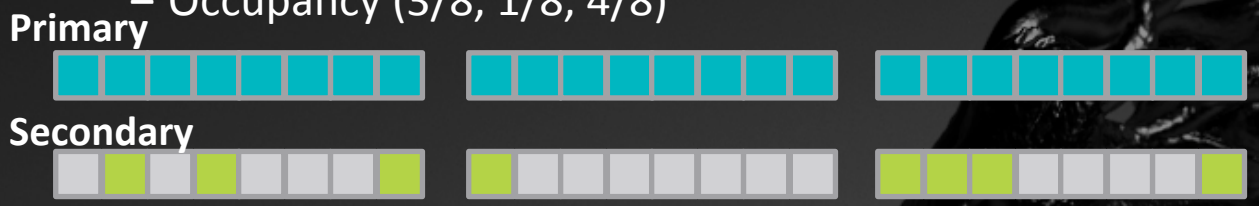
- With compaction
  - 1 SIMD execution
  - Occupancy (7/8)



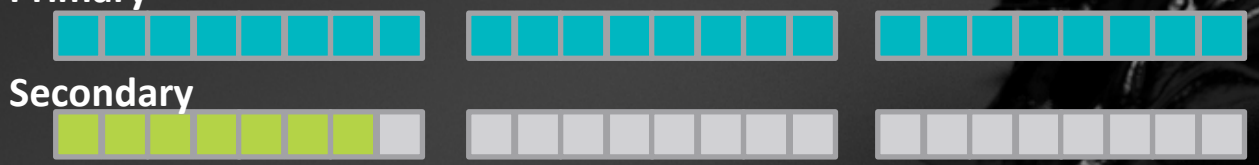
*When rays are created for all pixels, this is not necessary

RAY COMPACTION

- Sparse data lowers SIMD utilization
- Without compaction
  - 3 SIMD executions
  - Occupancy (3/8, 1/8, 4/8)



- With compaction
  - 1 SIMD execution
  - Occupancy (7/8)



*When rays are created for all pixels, this is not necessary

- No need to write a compaction kernel
- Can compact using global atomics
  - Prepare a counter (gRayCount)
  - Perform atomic increment to reserve memory
  - Better to do atomics in WG first, then do an atomic add per WG

```
__kernel
void PrimaryRayGenKernel(__global ...)
{
   ray, rayState = PrimaryRayGen( camera, pixelLoc );

   int dst = atom_inc( &gRayCount );
   gRay[dst] = ray;
   gRayState[dst] = rayState;
}
```

DIRECT ILLUMINATION COMPUTATION

- SampleLightKernel
  - Want to keep the work uniform
    - Different # of light sample per ray isn't good
  - Compute contribution from one point on a light
- Simple approach
  - Select a light
  - Select a point on a light
  - Compute DI without occlusion term
- More sophisticated light sampling
  - Using potential contribution for PDF
  - Forward+ style light culling

```
__kernel
void SampleLightKernel(__global ...)
{
    RayState s = gRayState[GIDX];
    Ray ray = gRay[GIDX];

    shadowRay, lfnDotV = Light_Sample( ray, s );

    gShadowRay[GIDX] = shadowRay;
    gDi[GIDX] = lfnDotV;
    gRayState[GIDX] = s;
}
```

DIRECT ILLUMINATION COMPUTATION

- SampleLightKernel
- TraceRayKernel
  - Check if the point on the light is visible or not
  - Reuse code
- AccumulateDIKernel
  - If the ray is not blocked, accumulate the result

```
__kernel
void SampleLightKernel(__global ...)
{
   RayState s = gRayState[GIDX];
   Ray ray = gRay[GIDX];

   shadowRay, lfnDotV = Light_Sample( ray, s );

   gShadowRay[GIDX] = shadowRay;
   gDi[GIDX] = lfnDotV;
   gRayState[GIDX] = s;
}

__kernel
void AccumulateDIKernel(__global ...)
{
   Hit shadowHit = gShadowHit[GIDX];
   float4 di = gDi[GIDX];

   if( !shadowHit )
      gFb[GIDX] += di;
}
```

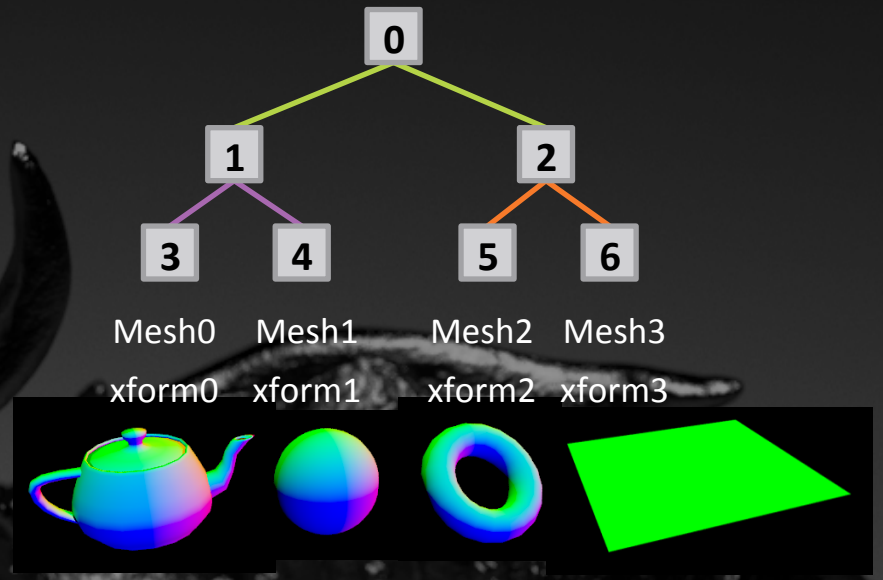
SAMPLE NEXT RAY

- Compute next ray by sampling BRDF
- Store ray and ray state

```
__kernel
void SampleNextRayKernel(__global ...)
{
    RayState s = gRayState[GIDX];
    Ray ray = gRay[GIDX];
    Hit hit = gHit[GIDX];

    if( !hit )
        return;

    nextRay, s = Brdf_Sample( ray, s );

    int dst = atom_inc( &gRayCount );
    gRayNext[dst] = nextRay;
    gRayStateNext[dst] = s;
}
```

TRACE KERNEL

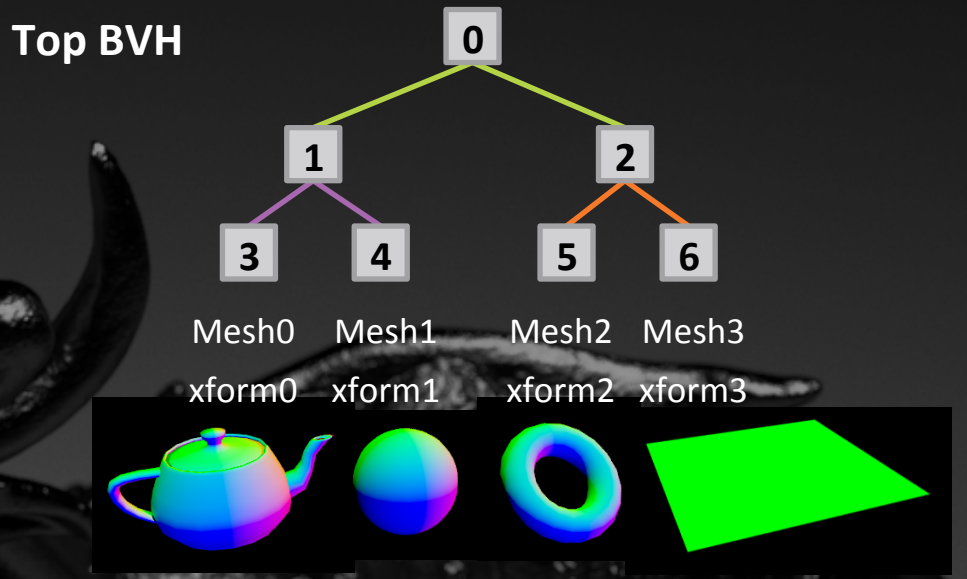
- BVH is used for acceleration structure
  - Index is used to describe hierarchy structure (no pointer)

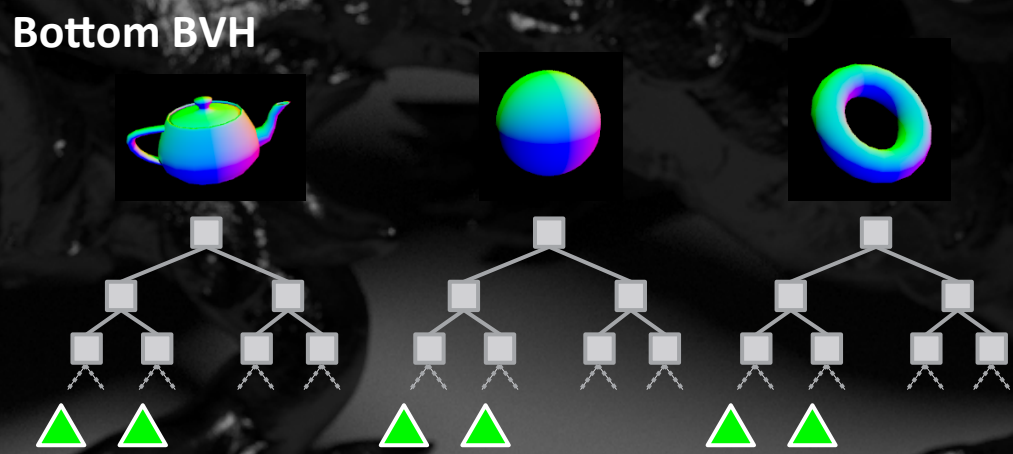


TRACE KERNEL

- BVH is used for acceleration structure
  - Index is used to describe hierarchy structure (no pointer)

- 2 level BVH
  - Top: stores an object in a leaf (object index, transform)
  - Bottom: stores a primitive (triangle, quad) in a leaf

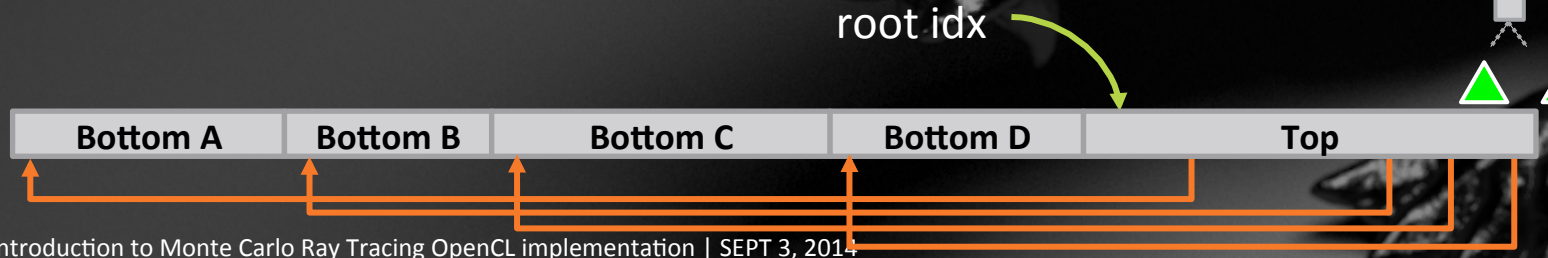


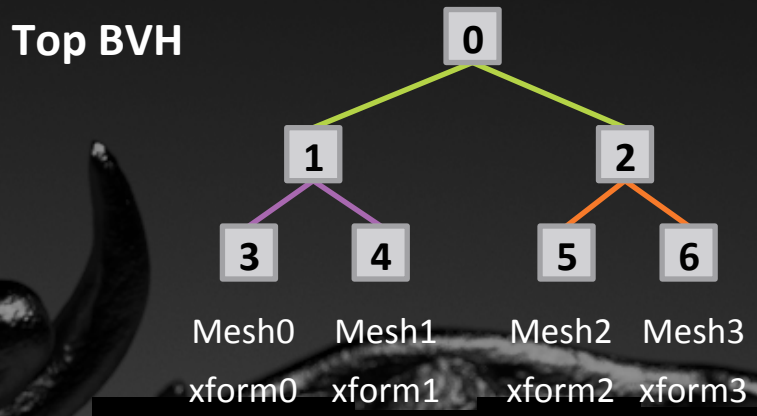


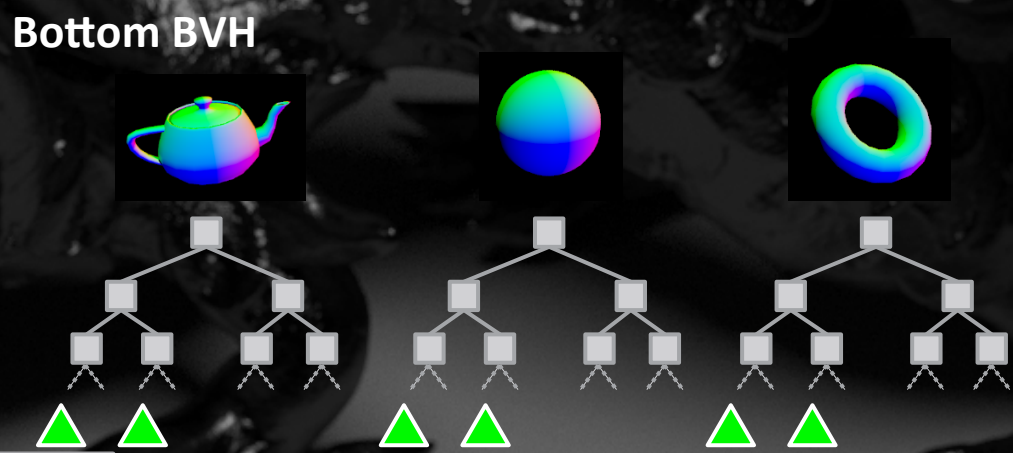
TRACE KERNEL

- BVH is used for acceleration structure
  - Index is used to describe hierarchy structure (no pointer)

- 2 level BVH
  - Top: stores an object in a leaf (object index, transform)
  - Bottom: stores a primitive (triangle, quad) in a leaf
- Store those BVHs in a single memory
  - Traverse top tree
  - Hit a leaf, transform the ray into object space
  - Traverse bottom tree
  - On exit, transform the ray back to world space



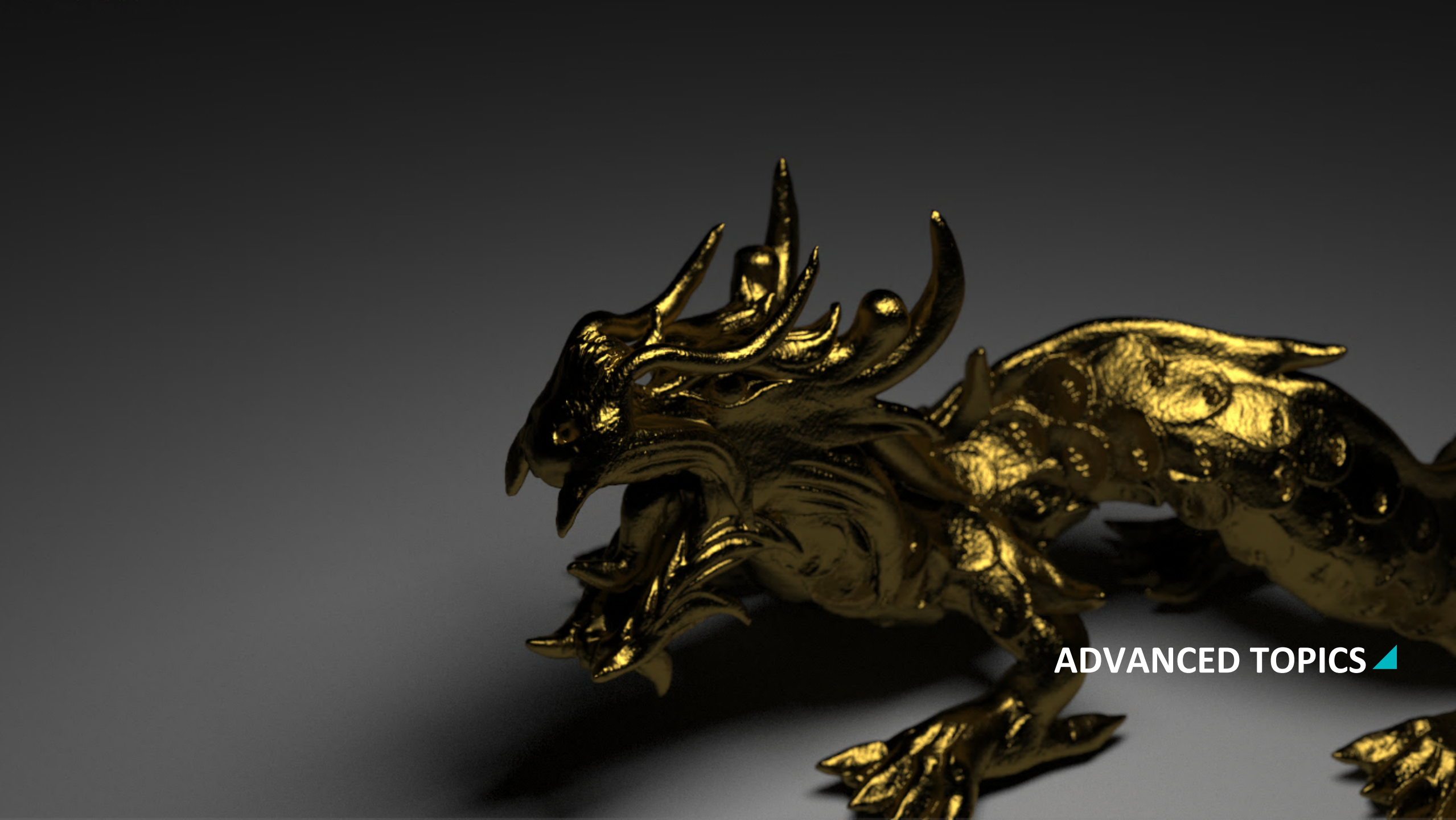




SO FAR

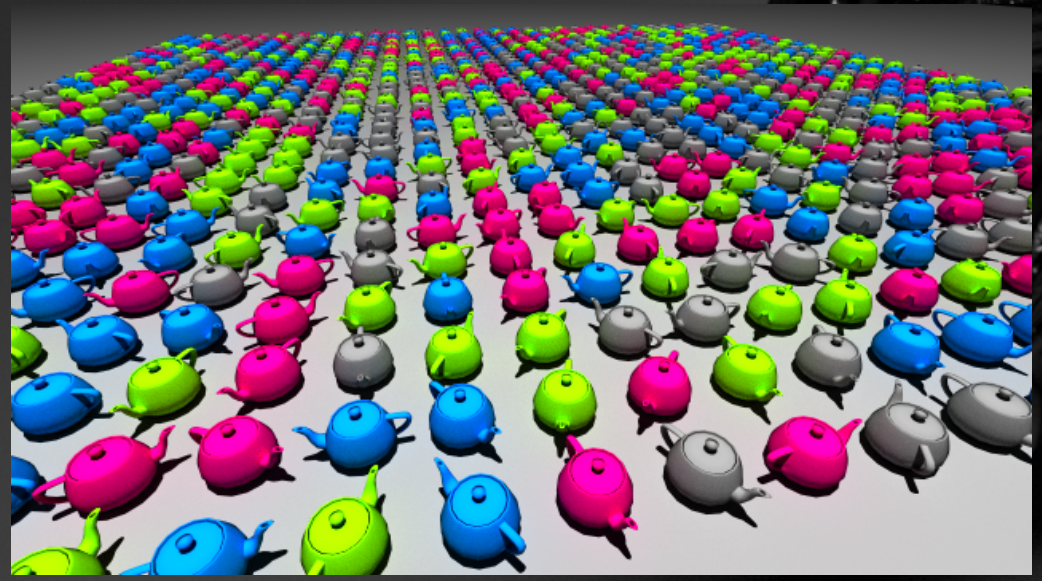
- Explained an OpenCL implementation of a simple path tracer
- Easy to extend from here

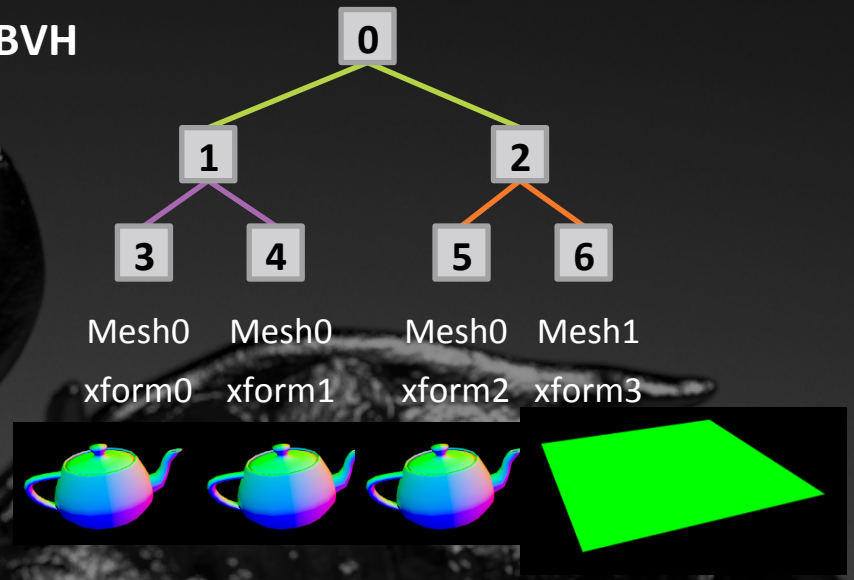
- Extension can be done by swapping one or two kernels
  - Material system, Shader
  - Light sampling
  - Support for different type of primitives
  - Ray caster + spatial acceleration structure


ADVANCED TOPICS

INSTANCING

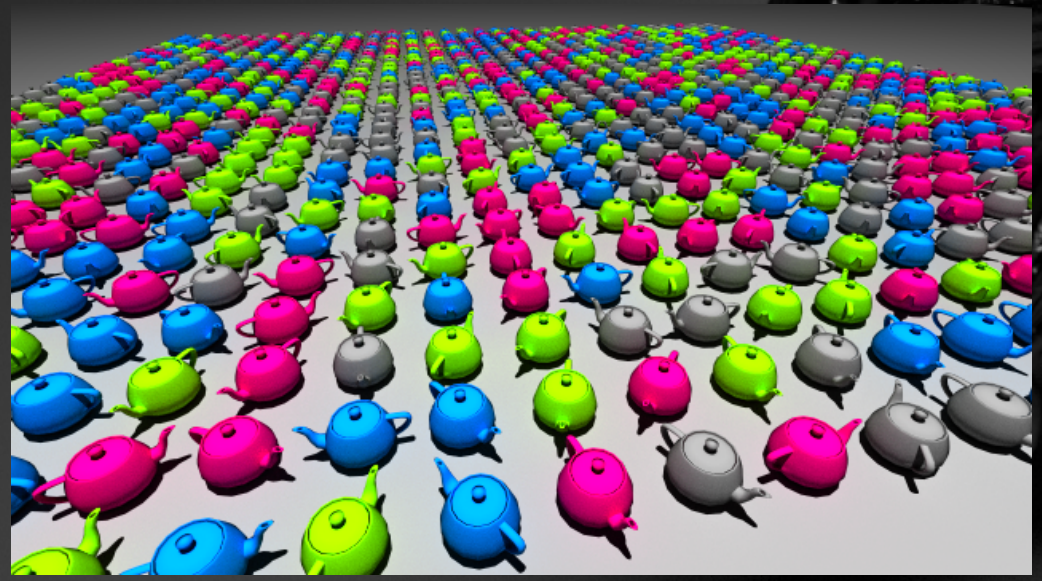
- Powerful technique to increase geometric complexity
- Small memory overhead
  - Shares geometric information (vertex, normal etc)
  - Shares BVH
  - Stores object transform

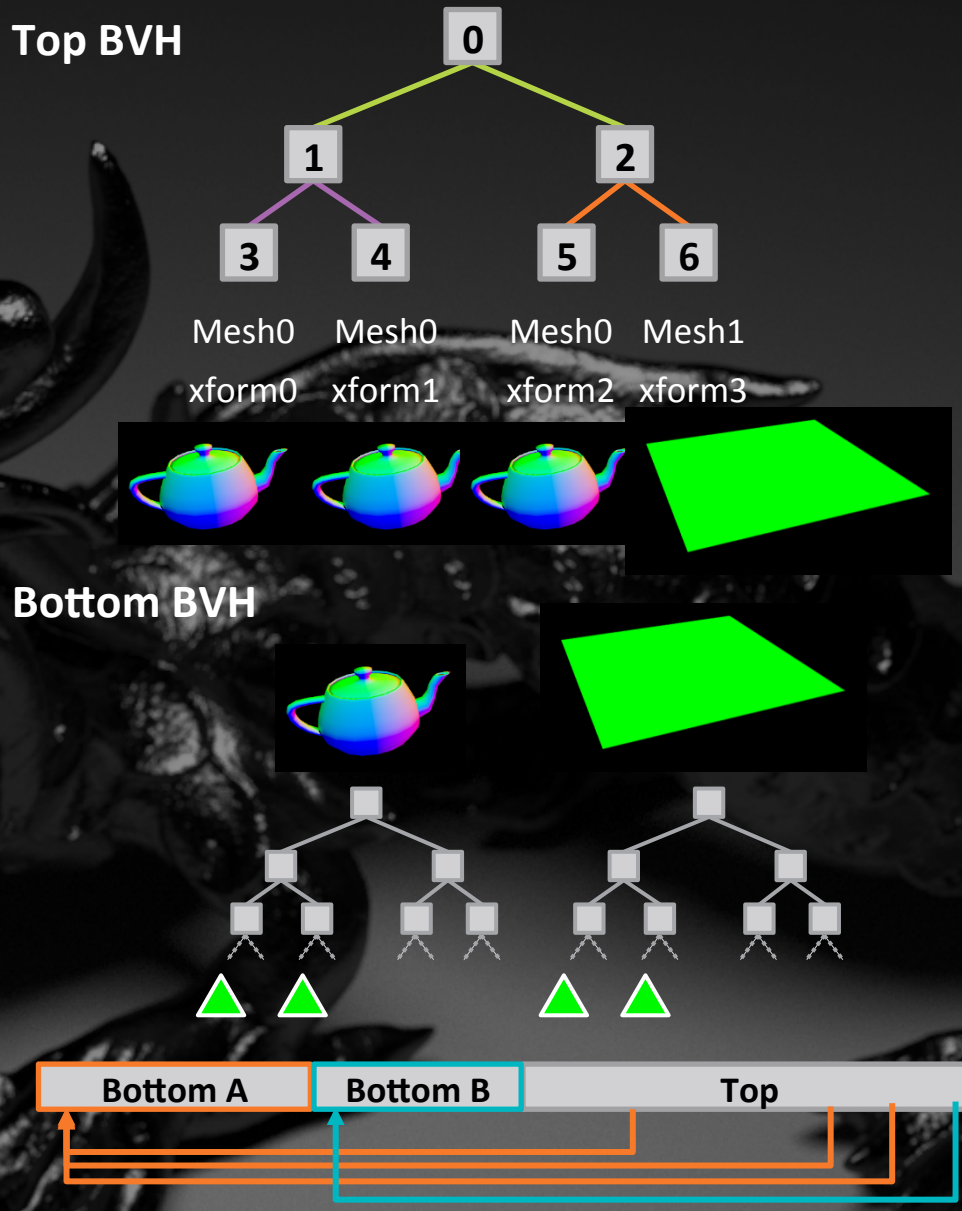




INSTANCING

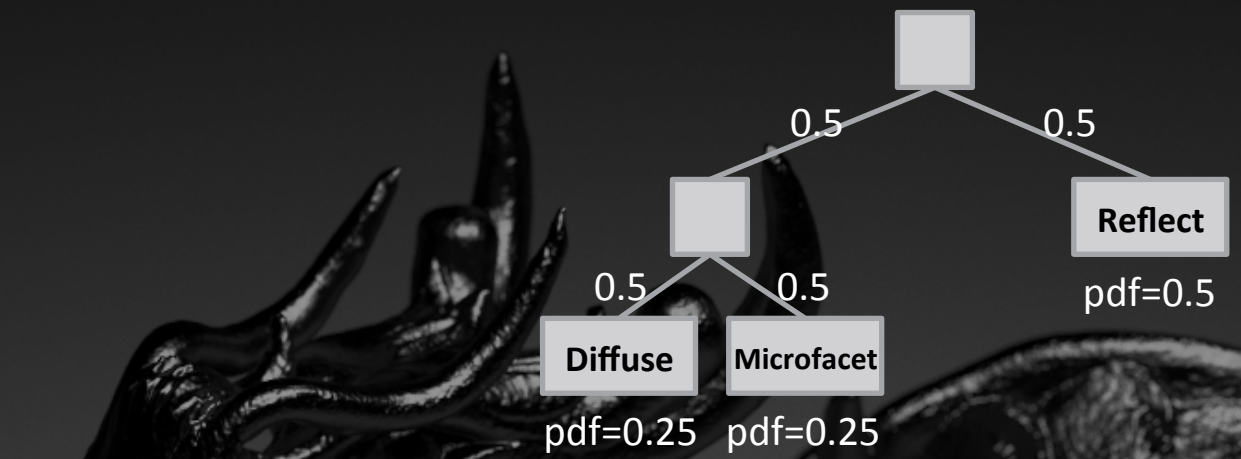
- Powerful technique to increase geometric complexity
- Small memory overhead
  - Shares geometric information (vertex, normal etc)
  - Shares BVH
  - Stores object transform





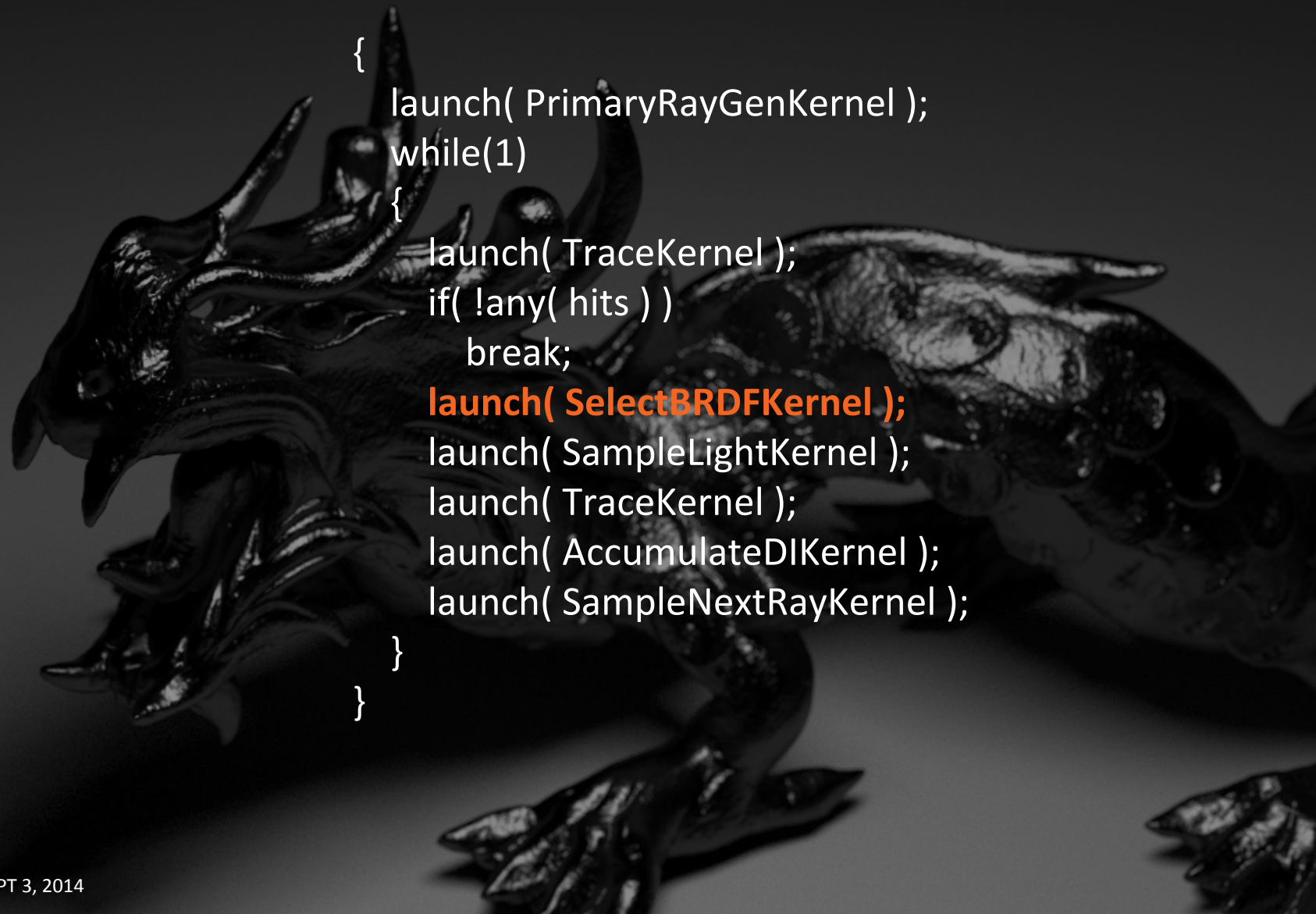
LAYERED MATERIAL

- Binary tree of BRDFs
- Leaf node
  - BRDF
- Internal node
  - Blend function
    - Fresnel blend, Linear blend
- Evaluate one BRDF at a time
  - Traverse binary tree
  - Random sampling at internal node



LAYERED MATERIAL

- Binary tree of BRDFs
- Leaf node
  - BRDF
- Internal node
  - Blend function
    - Fresnel blend, Linear blend
- Evaluate one BRDF at a time
  - Traverse binary tree
  - Random sampling at internal node

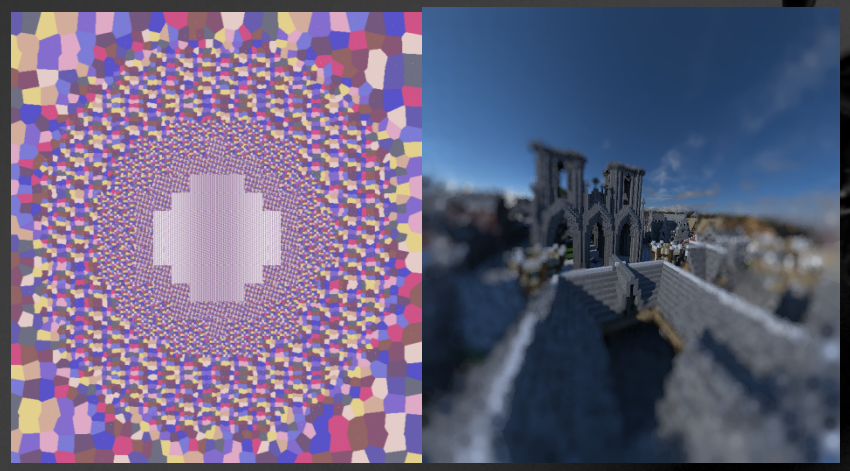


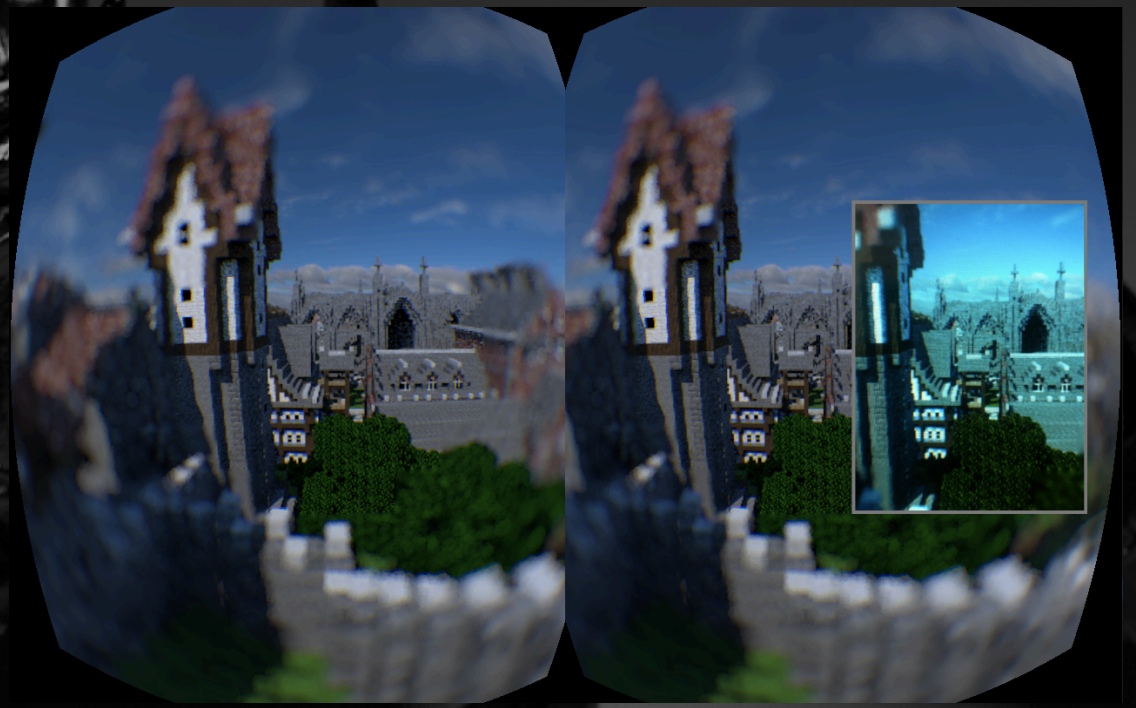
```
{
    launch( PrimaryRayGenKernel );
    while(1)
    {
        launch( TraceKernel );
        if( !any( hits ) )
            break;
        launch( SelectBRDFKernel );
        launch( SampleLightKernel );
        launch( TraceKernel );
        launch( AccumulateDIKernel );
        launch( SampleNextRayKernel );
    }
}
```


MORE ADVANCED TOPICS

VR

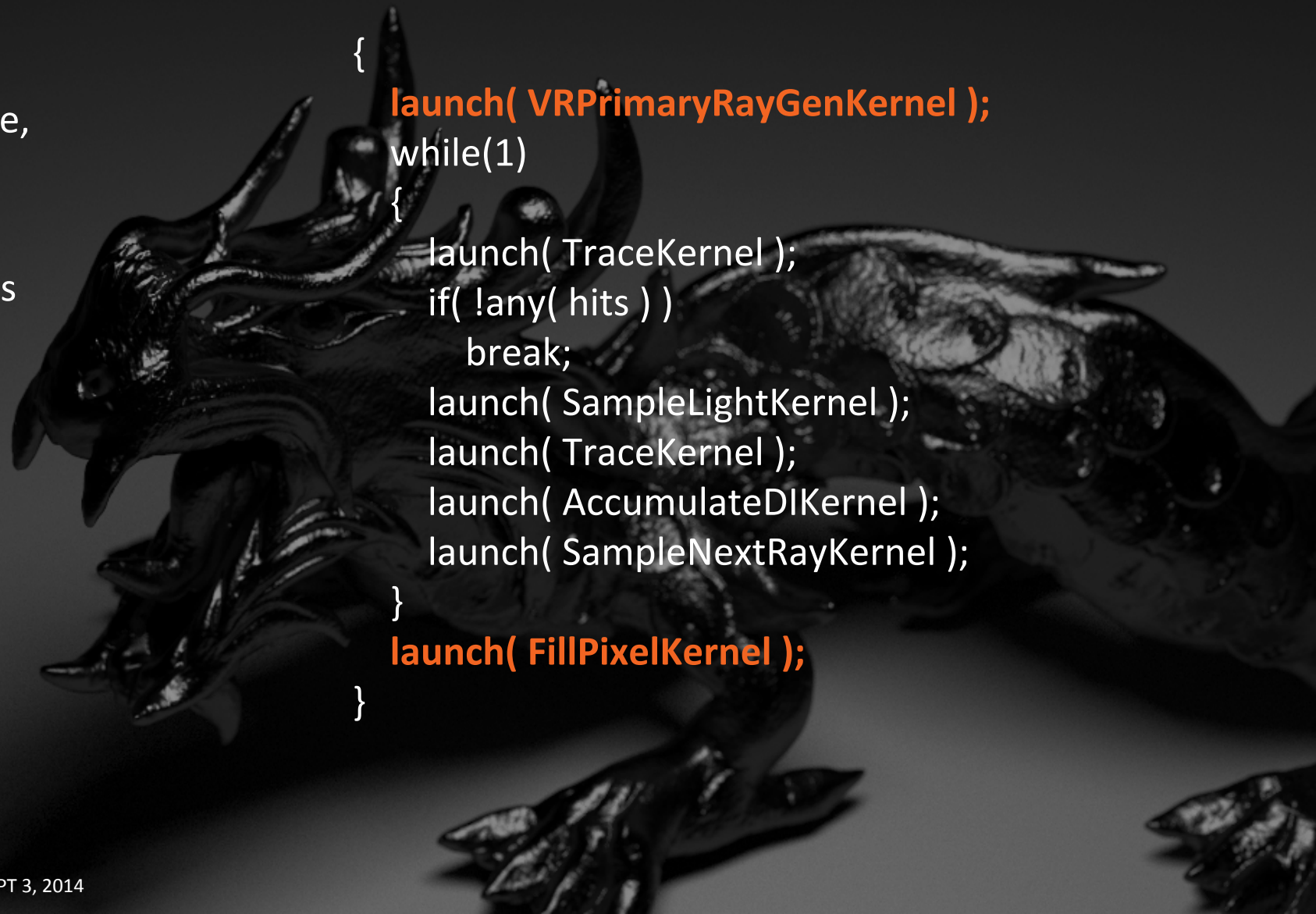
- Latency is super important
- To improve a frame rendering time,
  - Used multiple GPUs
  - Foveated rendering
- More than 60fps on 4 Hawaii GPUs
  - 6M triangles
  - 32 shadow rays/sample
  - 2 AA rays/sample





VR

- Latency is important
- To improve a frame rendering time,
  - Used multiple GPUs
  - Foveated rendering
- More than 60fps on 4 Hawaii GPUs
  - 6M triangles
  - 32 shadow rays/sample
  - 2 AA rays/sample



```
{
    launch( VRPrimaryRayGenKernel );
    while(1)
    {
        launch( TraceKernel );
        if( !any( hits ) )
            break;
        launch( SampleLightKernel );
        launch( TraceKernel );
        launch( AccumulateDIKernel );
        launch( SampleNextRayKernel );
    }
    launch( FillPixelKernel );
}
```

DISPLACEMENT MAPPING

- Powerful technique to increase geometric complexity
- Pre tessellation
  - Required memory is too large
  - GPU memory is too small
- Direct ray tracing
  - When hit a patch, tessellate and displace

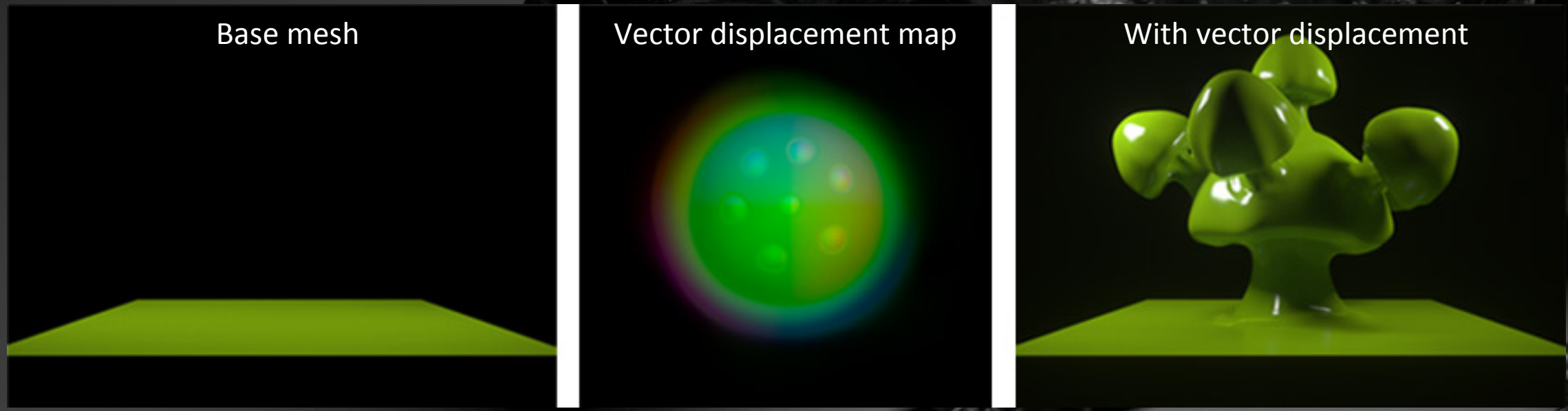


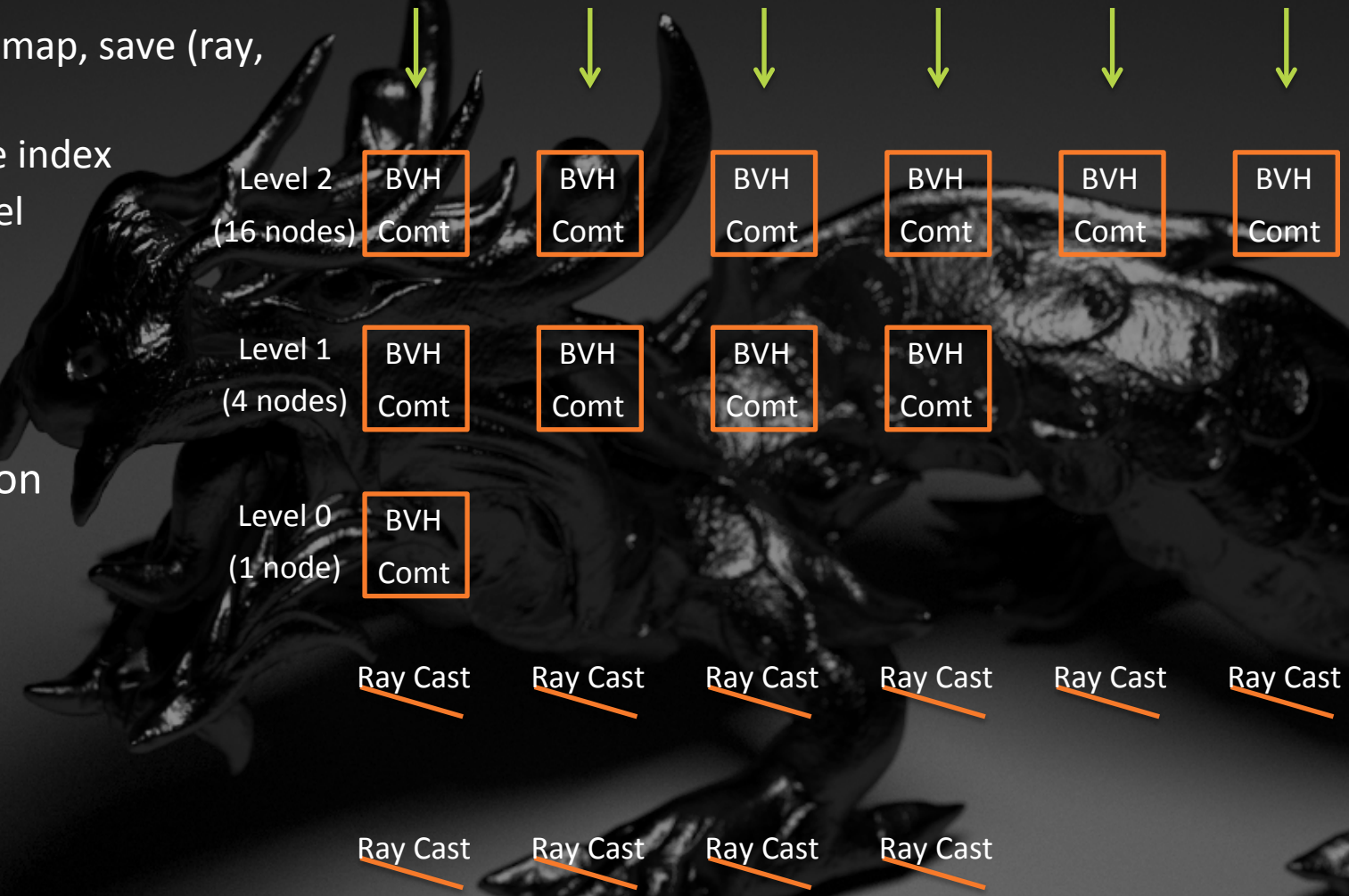
Fig. from http://support.nextlimit.com/display/mxdocsv3/Displacement+component

DISPLACEMENT MAPPING

- Powerful technique to increase geometric complexity
- Pre tessellation
  - Required memory is too large
  - GPU memory is too small
- Direct ray tracing
  - When hit a patch, tessellate and displace
- To amortize tessellation, displacement cost, batch ray intersection
- Need to change TraceKernel

DISPLACEMENT MAPPING

- TraceKernel
  - If a ray hit a quad with displacement map, save (ray, primitive) to a buffer
  - Sort (ray, primitive) pairs by primitive index
  - Process primitives in the list in parallel
- For each patch
  - Build quad BVH in parallel
  - Cast rays in parallel
- Key is work buffer memory allocation



VECTOR DISPLACEMENT IN ACTION

Base mesh

Vector displacement

52GB memory if pre tessellation is used

OPEN SHADING LANGUAGE

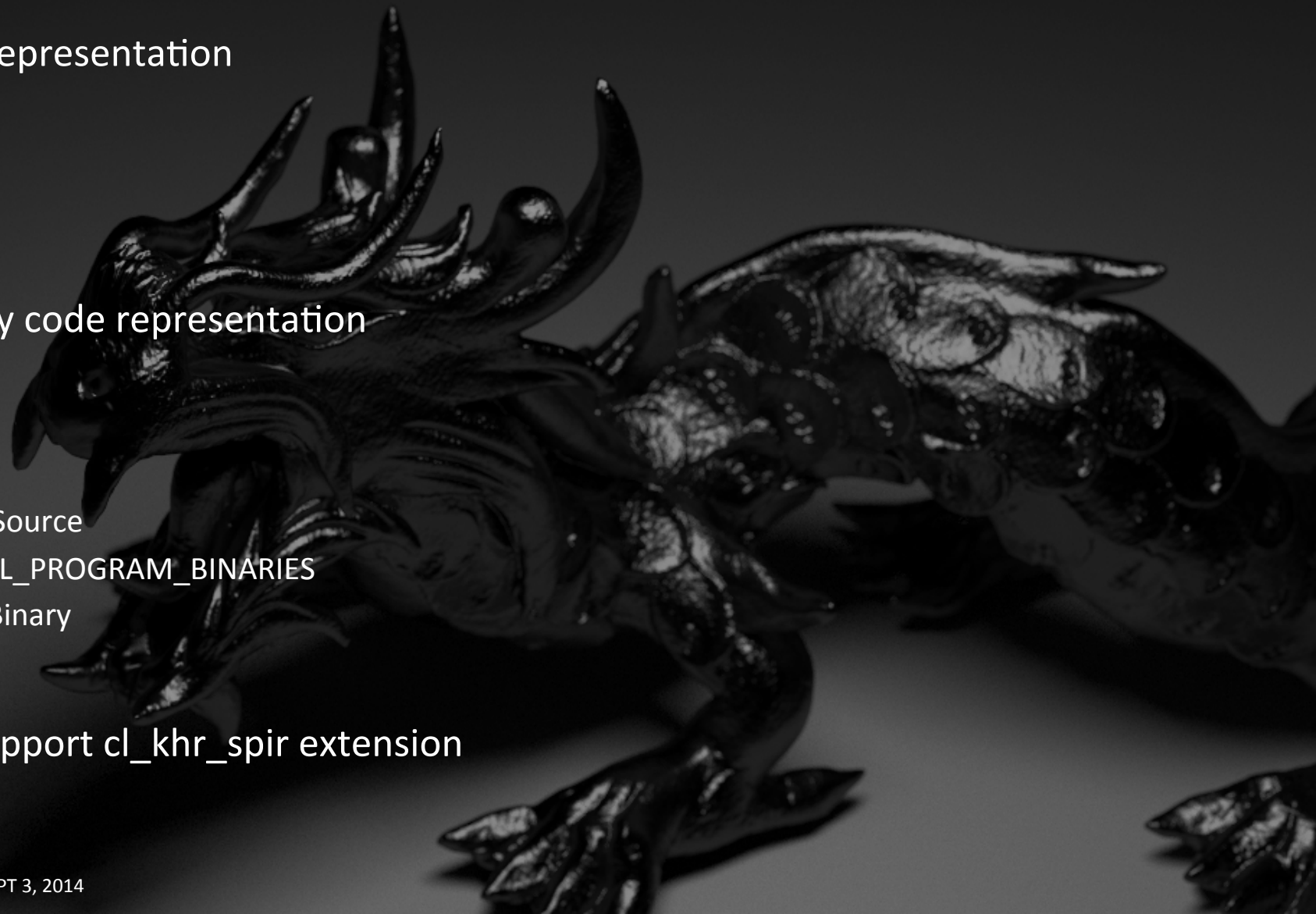
- OSL itself has nothing to do with OpenCL
- Many use cases

- Using OSL in OCL renderer
  - Translate OSL to
    - OCL kernel
    - SPIR
  - Feed those to OCL runtime
    - clBuildProgram
    - clCreateKernel

- OSL example

```
surface
matte [[ string description = "Lambertian diffuse material" ]]
(float Kd = 1 [[float UImin = 0, float UIsoftmax = 1 ]],
   color Cs = 1 [[float UImin = 0, float UImax = 1 ]],
   string texname = “diffuse.tex” [[int texture_slot = 1]]   )
{
        Ci = Kd * Cs * noise(5.0 * P) * diffuse (N);
}
```

SPIR



- Standard Portable Intermediate Representation
- Based on LLVM IR (32, 64)
- Useful to ship OpenCL Apps
- Device independent
- OpenCL did not have usable binary code representation
  - Binary for each device x driver
  - Combination explode
  - Embed kernel as string
    - Load source, clCreateProgramWithSource
    - Dump binary, clGetProgramInfo + CL_PROGRAM_BINARIES
    - Load binary, clCreateProgramWithBinary
- OpenCL implementation has to support cl_khr_spir extension
  - Works on AMD, Intel (OpenCL 1.2)
  - SPIR 2.0 is coming with OpenCL 2.0

SPIR

CREATE SPIR BINARY

- Offline compiler
  - clang-spir* -cc1 -emit-llvm-bc -triple spir-unknown-unknown -cl-spir-compile-options ”-x spir" -include <opencl_spir.h> -o <output> <input>
  - clBuildProgram with “-x spir -spir-std=CL1.2”
- Use host OpenCL API
  - clCompileProgram + Option
  - clGetProgramInfo + CL_PROGRAM_BINARIES

*https://github.com/KhronosGroup/SPIR


Autodesk Maya 2015 - Not for Resale: C:\share\12_scene10.mb* --- top
File Edit Modify Create Display Window Assets Select Mesh Edit Mesh Mesh Tools Normals Color Create UVs Edit UVs Fire Render Muscle XGen Pipeline Cache Bifrost Help
General Curves Surfaces Polygons Deformation Animation Dynamics Rendering PaintEffects Toon Muscle Fluids Fur nHair nCloth Custom XGen
View Shading Lighting Show Renderer Panels
Attribute Editor
List Selected Focus Attributes Show Help
FireRenderGlobals1
FireRenderGlobals: FireRenderGlobals1
Focus
Presets
Show Hide
Fire Render Controls
Enable Render
Pixel Filter Type Mitchell
Ray Number Per Pixel 3000
Max Ray Depth 6
Pixel Grid Size 4
Gamma 2.200
Texture Gamma 2.200
Enable HUD
HUDText Color
HUDValue Color
Render Mode globalIllumination
Node Behavior
Extra Attributes
Notes: FireRenderGlobals1
Select
Load Attributes
Copy Tab
Channel Box / Layer Editor
Attribute Editor
0.877403
MEL
Start