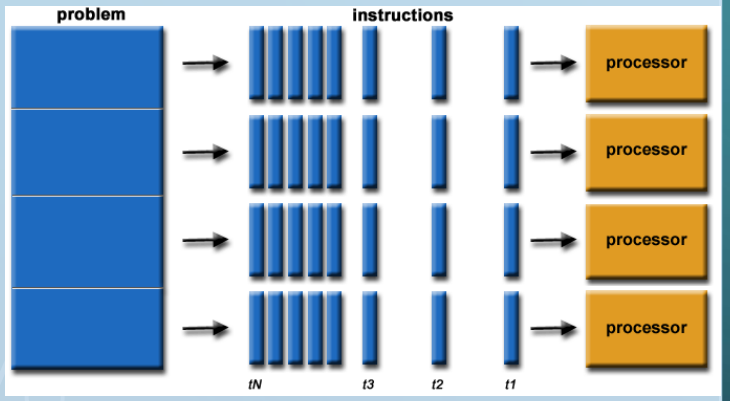
< PARALLEL GAME PROGRAMMING>

< JORDI BACH BALCELLS>

PARALLEL PROGRAMMING

What?



Why?

- Promising performance
- Not used until now
- Parallel hardware
- GPU

How?

- Multithreading
- SIMD
- (GP)GPU
- Platform layer

Microsoft DirectX

MULTITHREADING

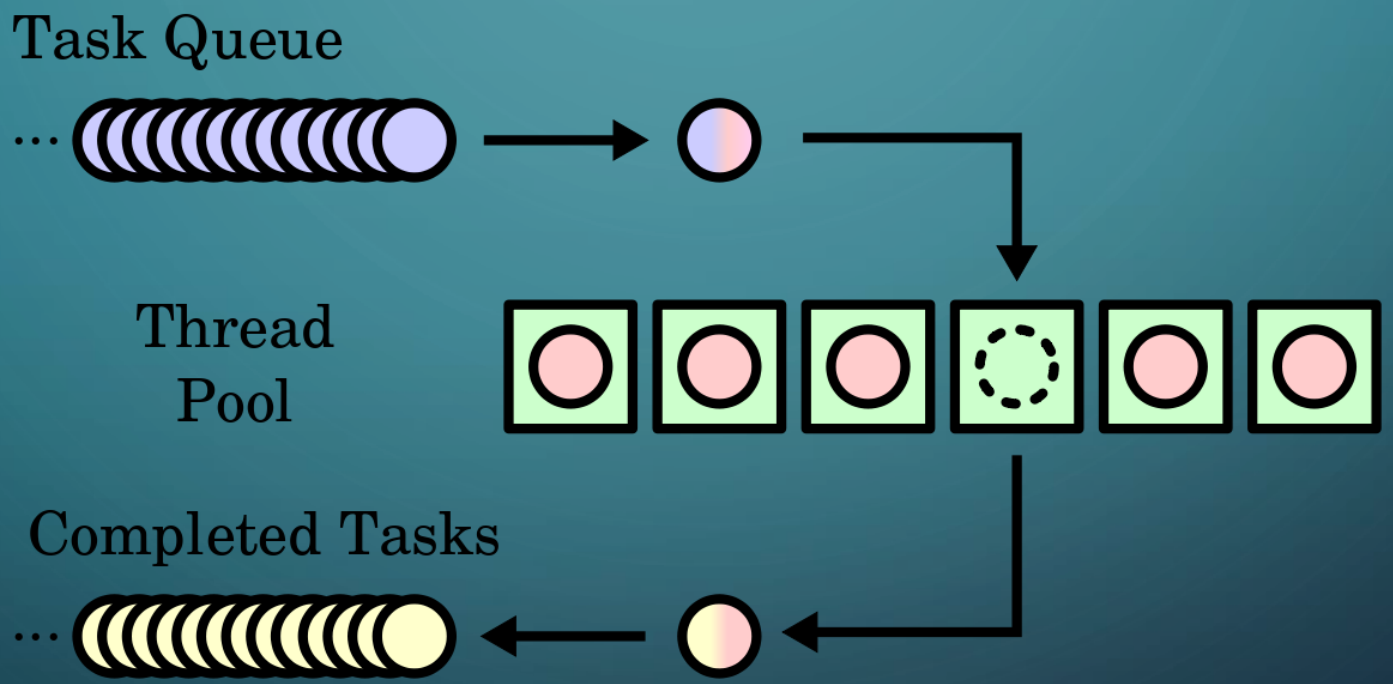
Definition	• Use multiple CPU logical cores
Scheduler	• Preemtive multitasking • Round Robin
Synchronization	• Why needed? • Implementation
Problems	• Overhead / Context switch • Blocked threads / Too much threads

MULTITHREADING

```
unsigned long DoWork(void* params)
{
    //HANDLE hMut = CreateMutex(NULL, FALSE, "Hi");
        WaitForSingleObject(params, INFINITE);
        unsigned int tId = GetCurrentThreadId();
        //unsigned int priorityClass = (unsigned int)GetPriorityClass(GetCurrentProcess());
        //int threadPriority = GetThreadPriority(GetCurrentThread());
        const char* message = "Thread ";
        char integer[11]; char* message2 = integer;
        unsigned char size = UIntToString((unsigned int)tId, &message2);
        const char* message3 = " working !!\n";
        HANDLE hStd = GetStdHandle(STD_OUTPUT_HANDLE);
        WriteConsoleA(hStd, message, strlen(message), NULL, NULL);
        WriteConsoleA(hStd, message2, size, NULL, NULL);
        WriteConsoleA(hStd, message3, strlen(message3), NULL, NULL);
        ReleaseMutex(params);
        //CloseHandle(hMut);
    //}
    return 0;
}
```

```
SYSTEM_INFO info;
DWORD threadId;
HANDLE hThread;
GetSystemInfo(&info);
//HANDLE hEve = CreateEvent(NULL, TRUE, FALSE, NULL);
HANDLE hMut = CreateMutex(NULL, FALSE, "Hi");
const char* message = "main thread!!\n";
HANDLE handles[5];
handles[0] = CreateThread(NULL, info.dwPageSize, DoWork, hMut, 0, &threadId);
handles[1] = CreateThread(NULL, info.dwPageSize, DoWork, hMut, 0, &threadId);
handles[2] = CreateThread(NULL, info.dwPageSize, DoWork, hMut, 0, &threadId);
handles[3] = CreateThread(NULL, info.dwPageSize, DoWork, hMut, 0, &threadId);
handles[4] = CreateThread(NULL, info.dwPageSize, DoWork, hMut, 0, &threadId);
WaitForMultipleObjects(5, handles, true, INFINITE);
WriteConsoleA(GetStdHandle(STD_OUTPUT_HANDLE), message, strlen(message), NULL, NULL);
//ReleaseMutex(hMut);
//SetEvent(hEve);
//CloseHandle(hEve);
CloseHandle(hMut);
return 0;
```

MULTITHREADING



SIMD

Definition	• Single instruction multiple data
Implementation	• Compiler flag • Intrinsics / Assembly language
Problem	• Portability
Solutions	• SSE4 • Runtime function dispatching

SIMD

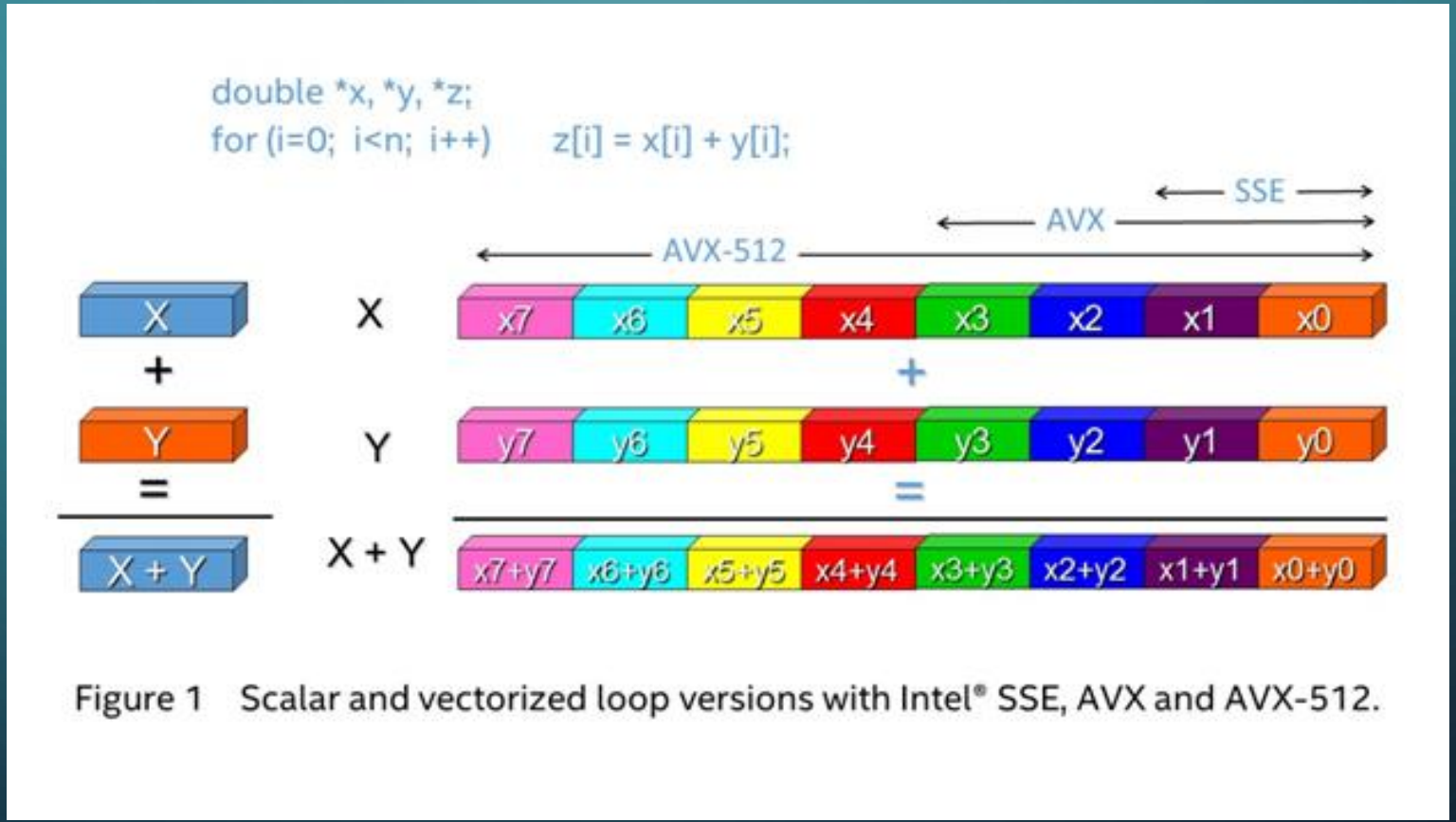
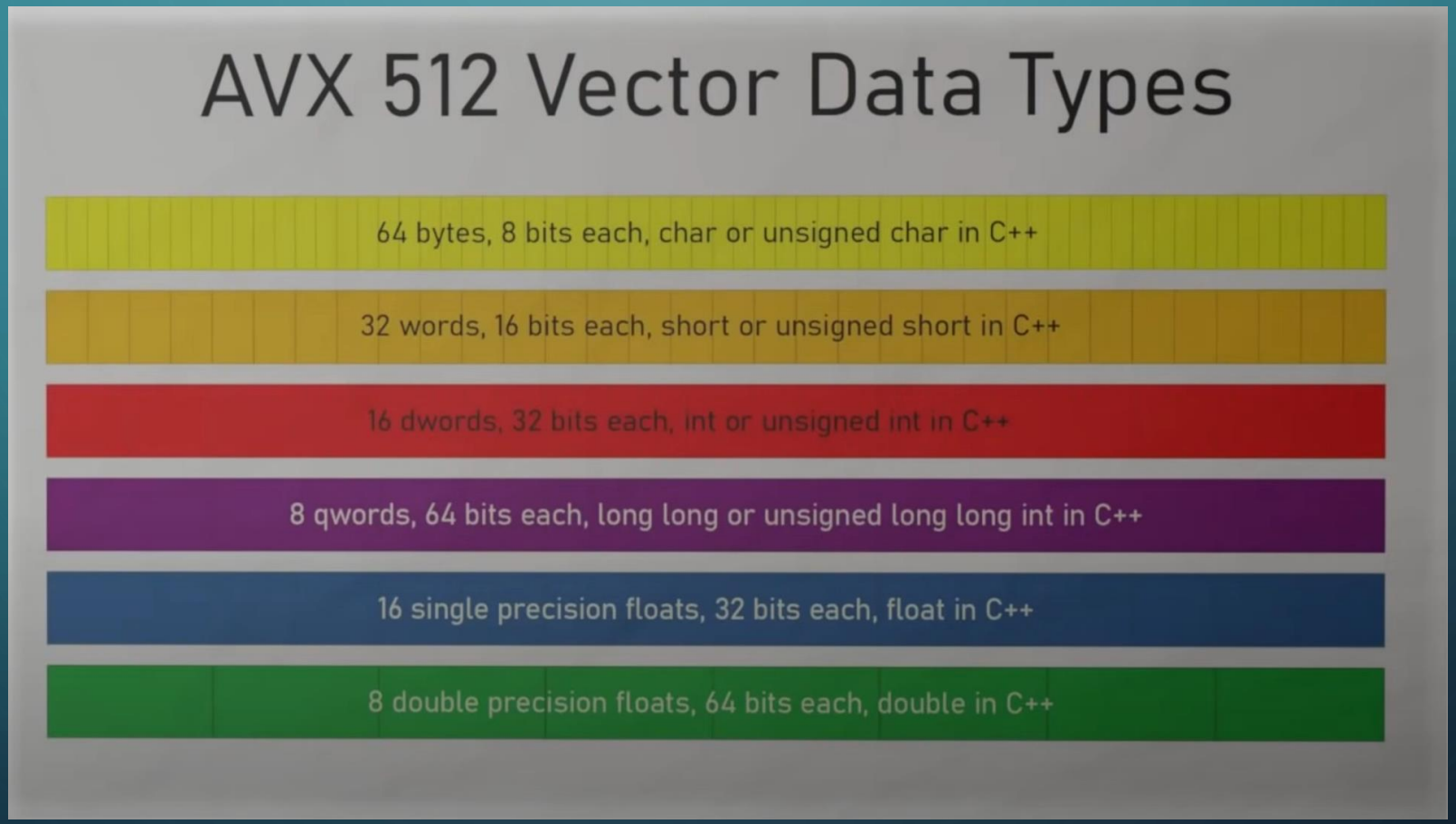


Figure 1 Scalar and vectorized loop versions with Intel® SSE, AVX and AVX-512.

SIMD


AVX 512 Vector Data Types
64 bytes, 8 bits each, char or unsigned char in C++
32 words, 16 bits each, short or unsigned short in C++
16 dwords, 32 bits each, int or unsigned int in C++
8 qwords, 64 bits each, long long or unsigned long long int in C++
16 single precision floats, 32 bits each, float in C++
8 double precision floats, 64 bits each, double in C++

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#

SIMD

```
        float arrayy[8000];
#ifdef __AVX2__
        __m256 mult = _mm256_set_ps(2.f,3.f,4.f,5.f,6.f,7.f,8.f,9.f);
        for (int ind = 0; ind < 8000; ind += 8)
        {
            float* ptr = (arrayy + ind);
            __m256 tmp = _mm256_set_ps(ind+7,ind+6,ind+5,ind+4,ind+3,ind+2,ind+1,ind);
            _mm256_storeu_ps(ptr, tmp);
        }
        for (int ind = 0; ind < 8000; ind += 8)
        {
            float* ptr = (arrayy + ind);
            __m256 reg = _mm256_loadu_ps(ptr);
            __m256 tmp = _mm256_mul_ps(reg, mult);
            _mm256_storeu_ps(ptr, tmp);
        }
#else
        for(int i = 0;i<8000;++i)
        {
            arrayy[i] = i;
        }
        for (int i = 0; i < 8000; i+=8)
        {
            arrayy[i+0] *= 9.f;
            arrayy[i+1] *= 8.f;
            arrayy[i+2] *= 7.f;
            arrayy[i+3] *= 6.f;
            arrayy[i+4] *= 5.f;
            arrayy[i+5] *= 4.f;
            arrayy[i+6] *= 3.f;
            arrayy[i+7] *= 2.f;
        }
#endif
```

SIMD

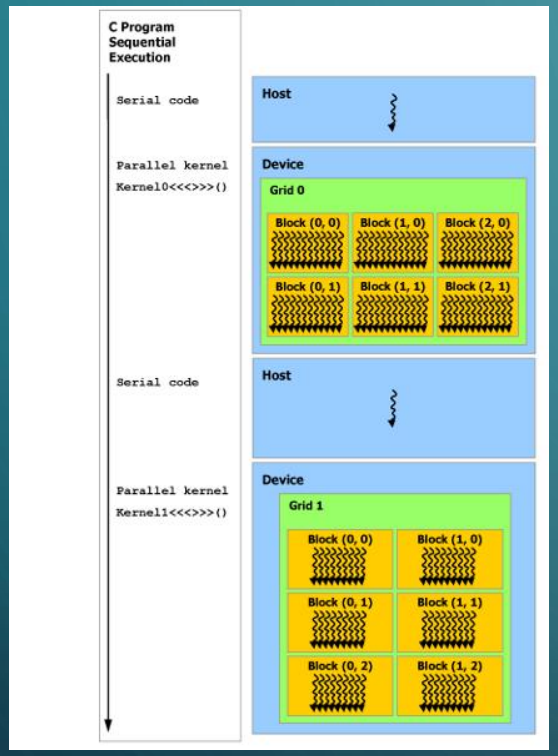
```cpp
#include <iostream>

extern "C" bool AVX512Support();
extern "C" bool AVX256Support();

int main()
{
    if (AVX512Support())
        std::cout << "yay 512" << std::endl;
    else if (AVX256Support())
        std::cout << "yay 256" << std::endl;
    else
        std::cout << "f" << std::endl;

    return 0;
}
```

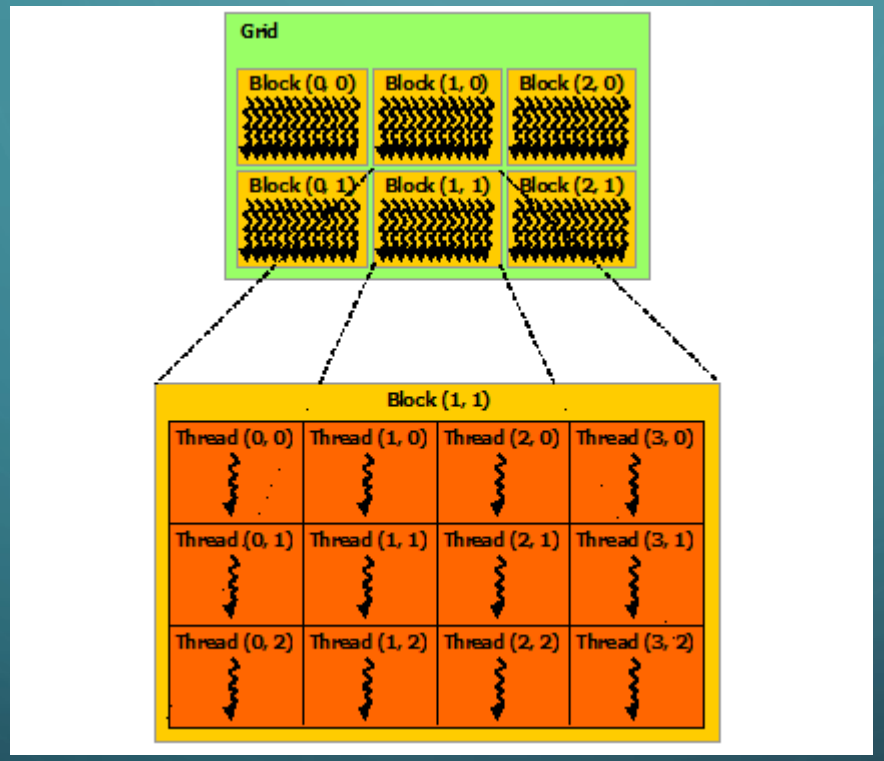
```
.code

    AVX512Support proc
        push rbx

        mov eax, 7
        mov ecx, 0

        cpuid

        shr ebx, 16
        and ebx, 1

        mov eax, ebx

        pop rbx
        ret
    AVX512Support endp

    AVX256Support proc
        push rbx

        mov eax, 7
        mov ecx, 0

        cpuid

        shr ebx, 5
        and ebx, 1

        mov eax, ebx

        pop rbx
        ret
    AVX256Support endp

end
```

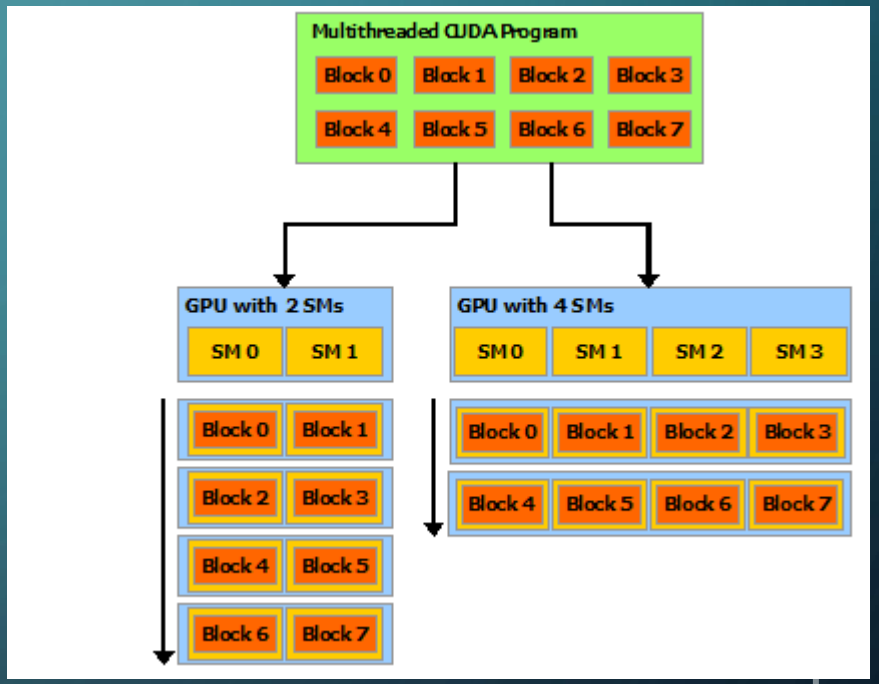
GPU

Definition	• Coprocessor
GPU vs. CPU hardware	• Serial vs Parallel / Latency vs Throughput • Cores / Frequency / Memory bandwidth
GPU software (CUDA)	• Grayscale Image • SIMT / Shared Memory
Optimization	• Memory layout and access patterns • Occupancy & Concurrency

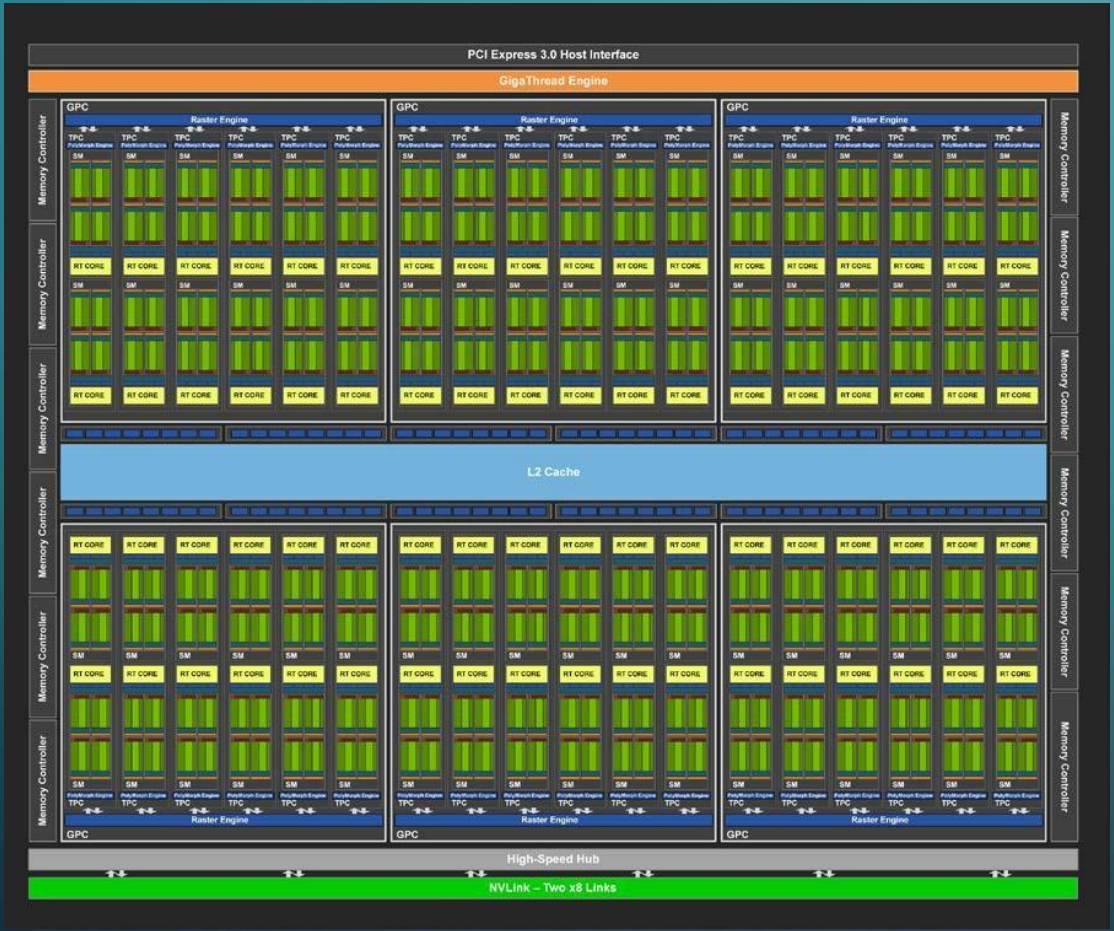
GPU


C Program
Sequential
Execution
Serial code
Parallel kernel
Kernel0<<<>>>()
Serial code
Parallel kernel
Kernel1<<<>>>()
Host
Device
Grid 0
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Host
Device
Grid 1
Block (0, 0)
Block (1, 0)
Block (0, 1)
Block (1, 1)
Block (0, 2)
Block (1, 2)


Grid
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)


Multithreaded CUDA Program
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 2 SMs
SM 0
SM 1
GPU with 4 SMs
SM 0
SM 1
SM 2
SM 3

GPU



1 SM Compute capability 7.5 (Turing):

- 64 FP32 cores
- 32 FP64 cores
- 64 INT32 cores
- 8 Tensor cores
- 16 SFU
- 4 warp schedulers
- 96 KB L1 cache

GPU

CPU loop

```
for (unsigned char* p = img, *pg = gray_img; p != img + img_size; p+= 3, pg += 1)
{
    *pg = (unsigned char)(((*p) * 0.299f + (*(p + 1)) * 0.587f + (*(p + 2)) * 0.114f));
}
```

GPU kernel

```
__global__ void RGBToGreyScale2D(const uchar3* const colorPixels, unsigned char* greyPixels, int rowElements, int columnElements)
{
    const int y = threadIdx.y + blockIdx.y * blockDim.y;
    const int x = threadIdx.x + blockIdx.x * blockDim.x;
    if (y < columnElements && x < rowElements)
    {
        int index = y * rowElements + x;
        greyPixels[index] = 0.299f * colorPixels[index].x + 0.587f * colorPixels[index].y + 0.114f * colorPixels[index].z;
    }
}
```

```
//cacheSize = blockDim.x*4*3
__global__ void CachedRGB12ToGreyScale(const uchar4* const colorPixels, uchar4* greyPixels, unsigned int numThreads)
{
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    if (index < numThreads)
    {
        const unsigned int offset = 2 * blockDim.x * blockIdx.x;
        extern __shared__ uchar4 shared[];
        shared[threadIdx.x] = colorPixels[index + offset];
        shared[threadIdx.x + blockDim.x] = colorPixels[index + offset + blockDim.x];
        shared[threadIdx.x + 2*blockDim.x] = colorPixels[index + offset + 2*blockDim.x];
        __syncthreads();
        const uchar4 threadDataBlock1 = shared[threadIdx.x * 3];
        const uchar4 threadDataBlock2 = shared[threadIdx.x * 3 + 1];
        const uchar4 threadDataBlock3 = shared[threadIdx.x * 3 + 2];
        uchar4 result;
        result.x = 0.299f * threadDataBlock1.x + 0.587f * threadDataBlock1.y + 0.114f * threadDataBlock1.z;
        result.y = 0.299f * threadDataBlock1.w + 0.587f * threadDataBlock2.x + 0.114f * threadDataBlock2.y;
        result.z = 0.299f * threadDataBlock2.z + 0.587f * threadDataBlock2.w + 0.114f * threadDataBlock3.x;
        result.w = 0.299f * threadDataBlock3.y + 0.587f * threadDataBlock3.z + 0.114f * threadDataBlock3.w;
        greyPixels[index] = result;
    }
}
```

GPU kernel
40% faster

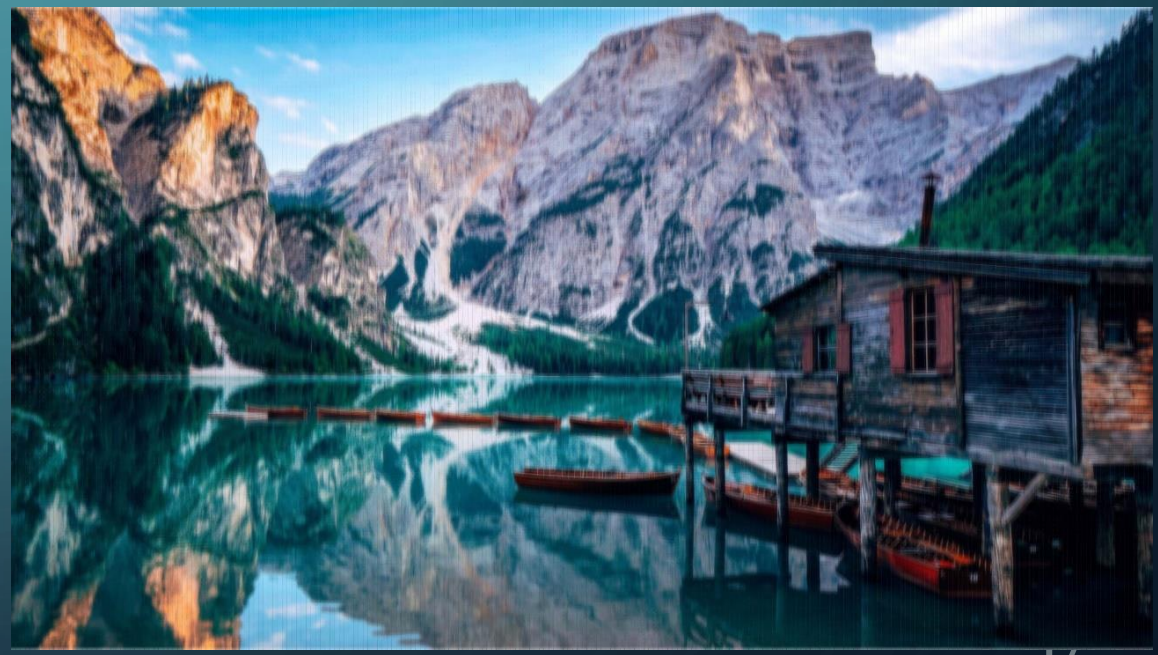
GPU

4k

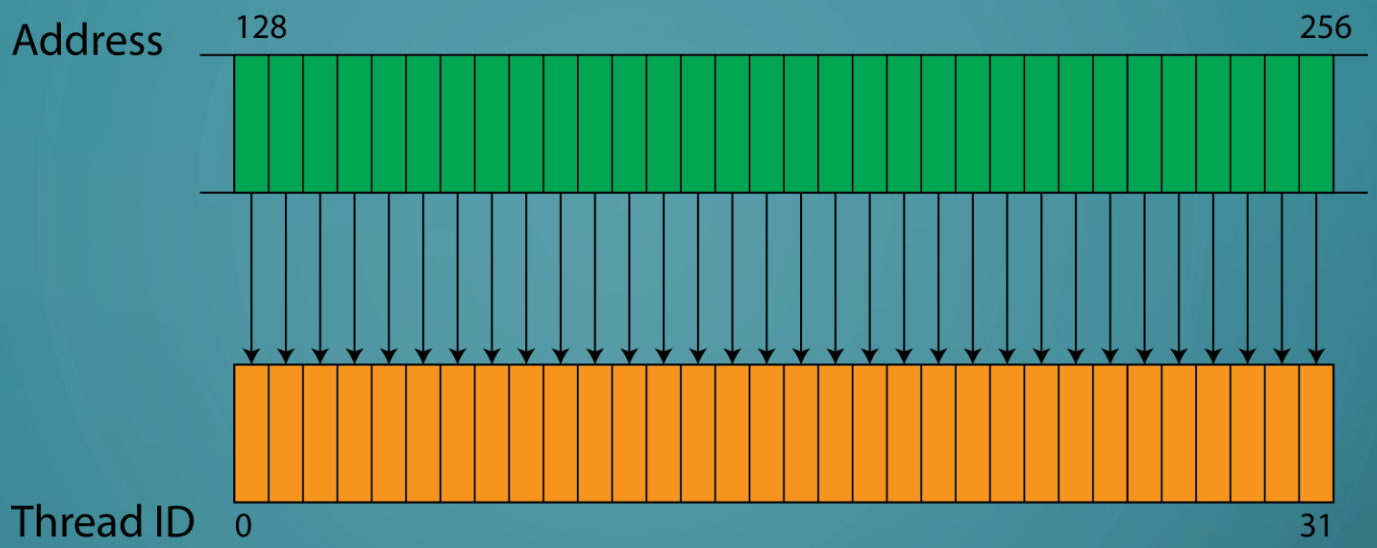
CPU: 13 ms
GPU: 0.3 ms

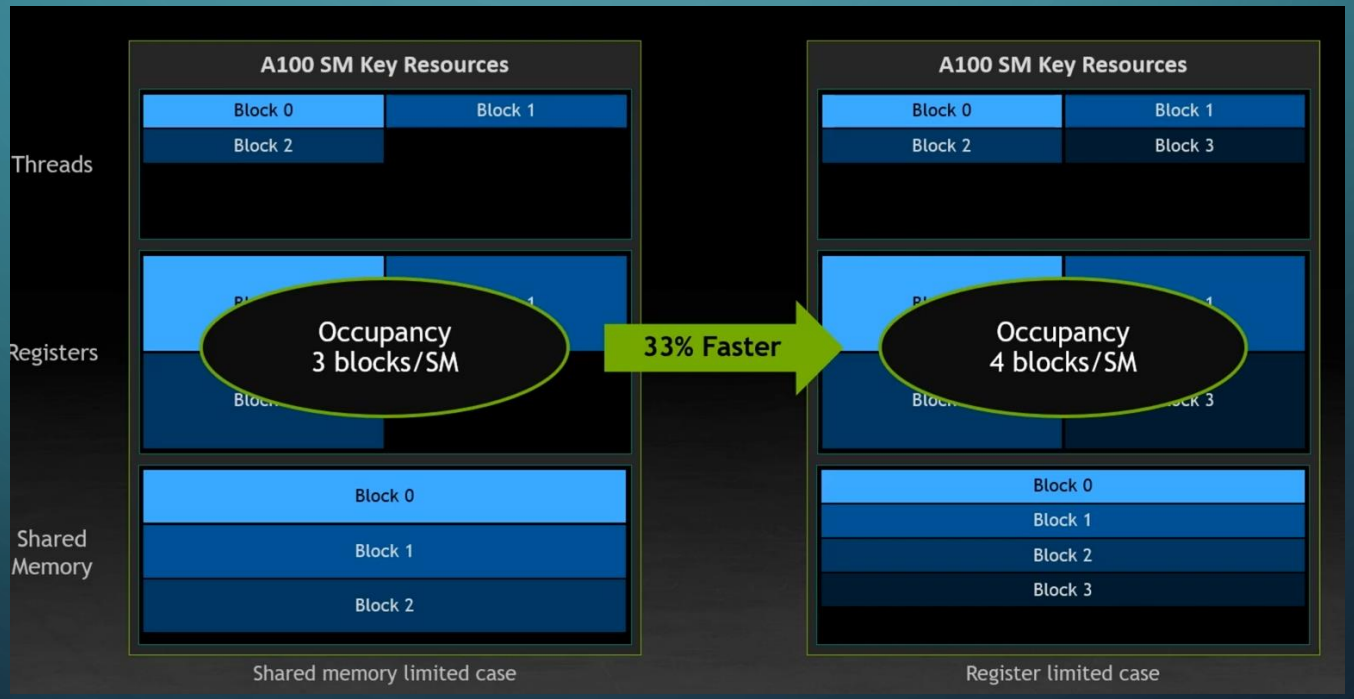
8k

CPU: 53 ms
GPU: 0.62 ms



GPU

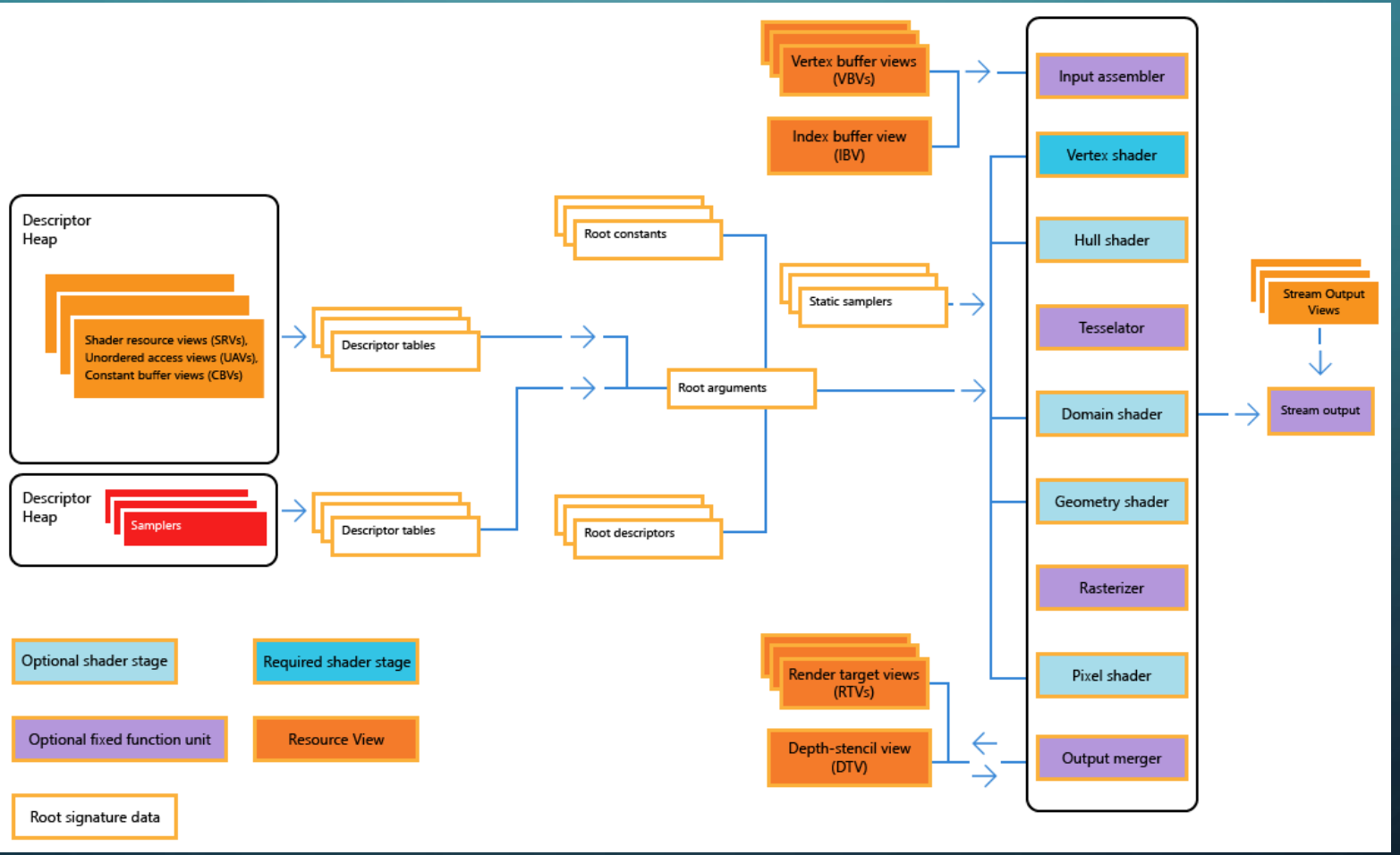

Address
128
256
Thread ID
0
31


Threads
Registers
Shared Memory
A100 SM Key Resources
Block 0
Block 1
Block 2
Occupancy
3 blocks/SM
Block 0
Block 1
Block 2
Shared memory limited case
33% Faster
A100 SM Key Resources
Block 0
Block 1
Block 2
Block 3
Occupancy
4 blocks/SM
Block 0
Block 1
Block 2
Block 3
Register limited case

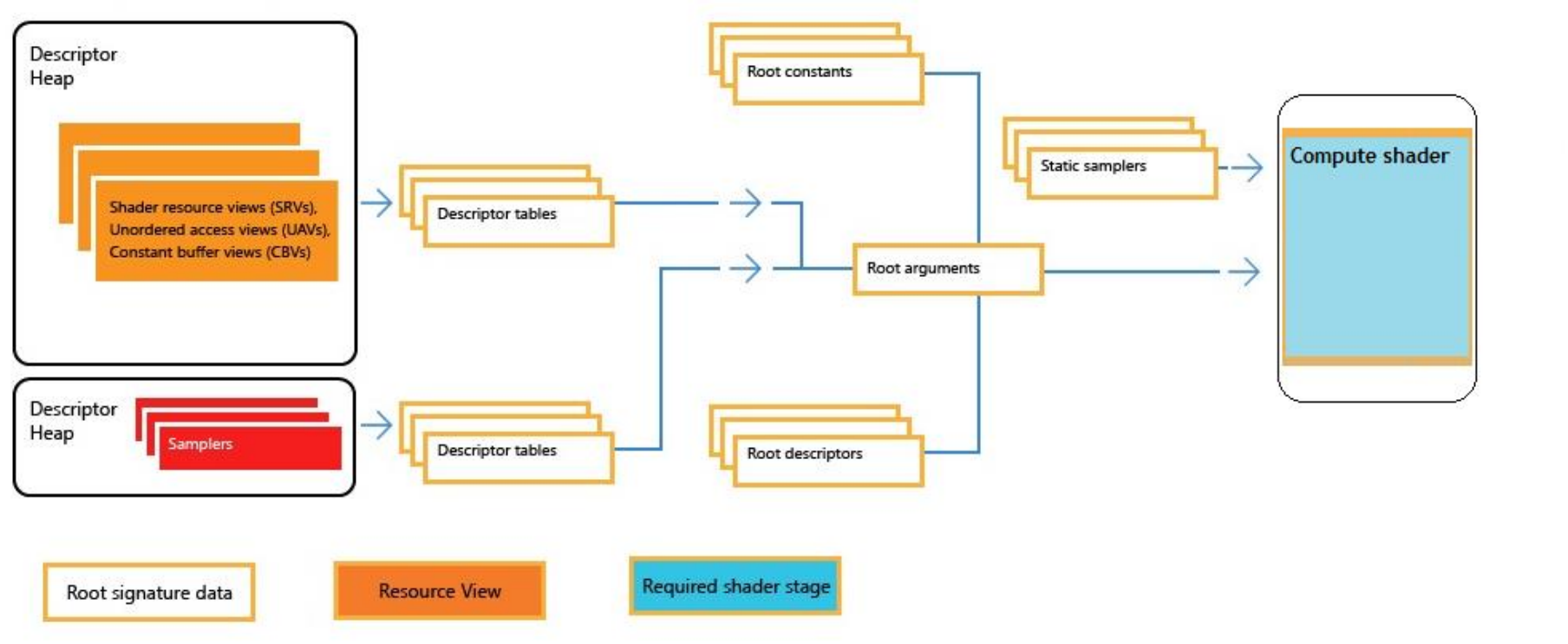
DIRECTX12

DirectX12	• CUDA not portable • Shaders / Pipelines
Low-Level renderer	• Memory residency / Resource binding • GPU work submission
GPU-CPU synchronization	• Application responsability
Compute pipeline (GPGPU)	• Mipmapping

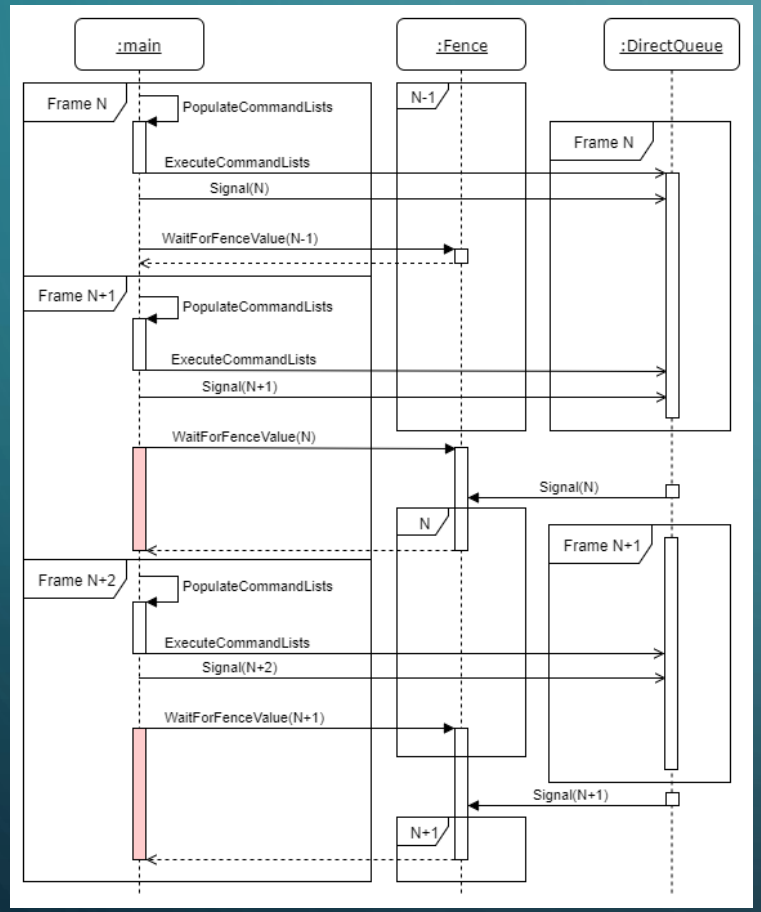
DIRECTX12


Vertex buffer views (VBVs)
Index buffer view (IBV)
Input assembler
Vertex shader
Hull shader
Tesselator
Domain shader
Geometry shader
Rasterizer
Pixel shader
Output merger
Stream Output Views
Stream output
Descriptor Heap
Shader resource views (SRVs), Unordered access views (UAVs), Constant buffer views (CBVs)
Descriptor tables
Root constants
Static samplers
Root arguments
Descriptor Heap
Samplers
Descriptor tables
Root descriptors
Render target views (RTVs)
Depth-stencil view (DTV)
Optional shader stage
Required shader stage
Optional fixed function unit
Resource View
Root signature data

DIRECTX12


Descriptor Heap
Shader resource views (SRVs), Unordered access views (UAVs), Constant buffer views (CBVs)
Descriptor Heap
Samplers
Descriptor tables
Descriptor tables
Root constants
Root arguments
Static samplers
Root descriptors
Compute shader
Root signature data
Resource View
Required shader stage

DIRECTX12

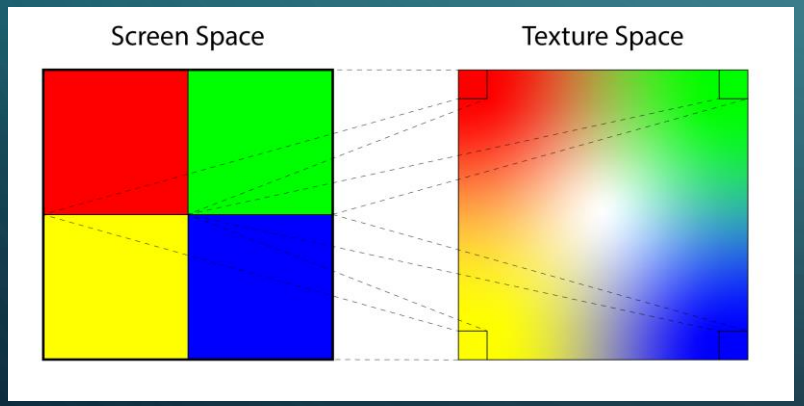


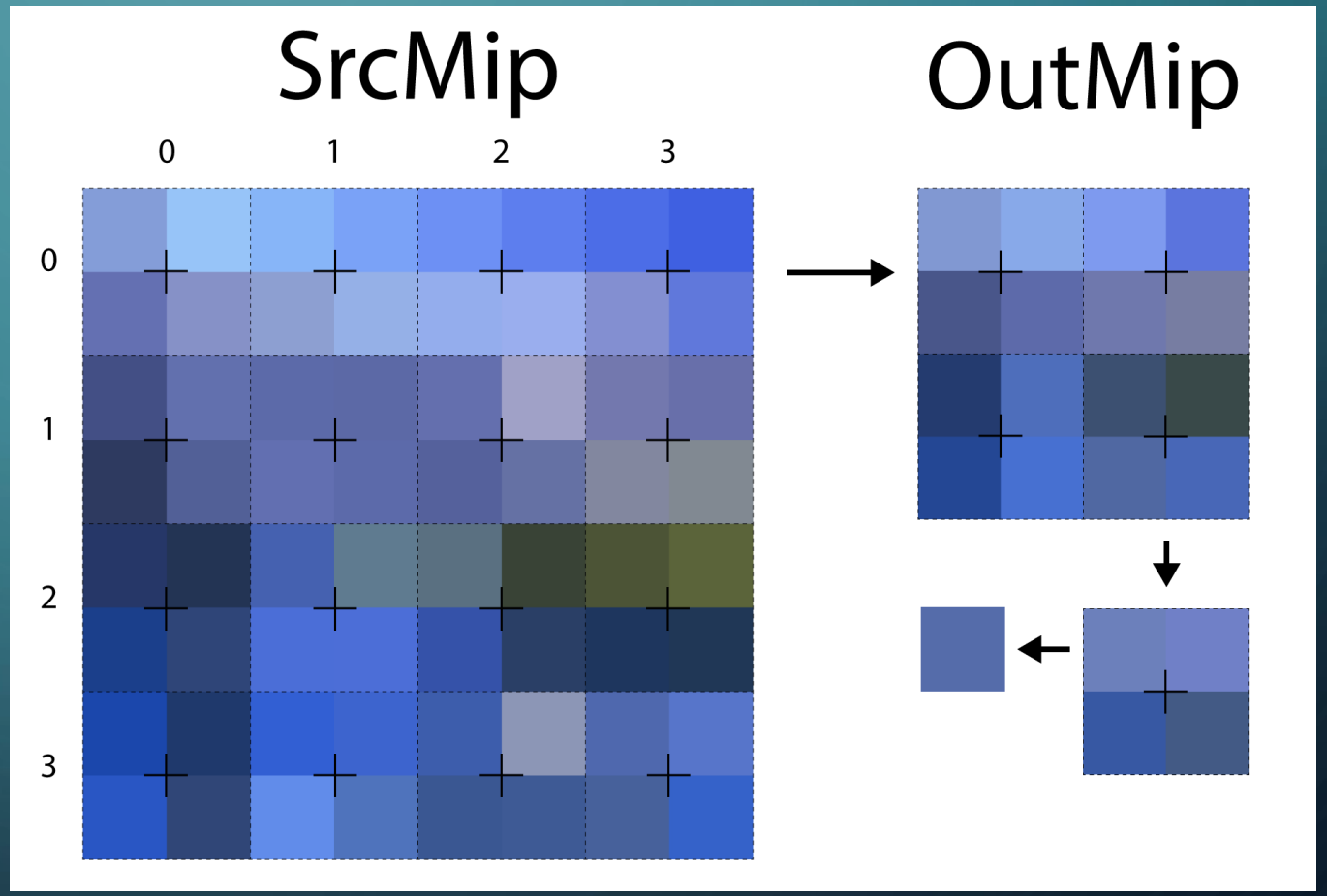

```
cList->Close();
ID3D12CommandList* const ppCommandLists[] = {cList};
renderer.cQueue.iCQueue->ExecuteCommandLists(1, ppCommandLists);
renderer.rtvFenceValues[currentBackBufferIndex] = ++cQueue.fence.value;
cQueue.iCQueue->Signal(cQueue.fence.iFence, cQueue.fence.value);
cQueue.cAllocators.Push({ cAllocator, cQueue.fence.value });
cQueue.cLists.Push(cList);
//present the frame
unsigned int sync = renderer.vSync ? 1 : 0;
unsigned int presentFlags = (renderer.tearingSupport && !renderer.vSync) ? DXGI_PRESENT_ALLOW_TEARING : 0;
renderer.sChain->Present(sync, presentFlags);
currentBackBufferIndex = renderer.sChain->GetCurrentBackBufferIndex();
unsigned long long fValue = renderer.rtvFenceValues[currentBackBufferIndex];
//wait for fence value
if (cQueue.fence.iFence->GetCompletedValue() < fValue)
{
    cQueue.fence.iFence->SetEventOnCompletion(fValue, cQueue.fence.event);
    WaitForSingleObject(cQueue.fence.event, INFINITE);
    PIXNotifyWakeFromFenceSignal(cQueue.fence.event);
}
```

DIRECTX12

DIRECTX12


tom's hardware guide
No Mipmapping
With Mipmapping


Screen Space
Texture Space


SrcMip
OutMip
0
1
2
3
0
1
2
3