
# Performance Profiling

袁亚振


01 Introduction

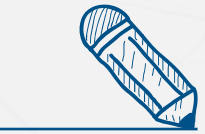
02 Basic GPU Arch

03 GPU Profiling

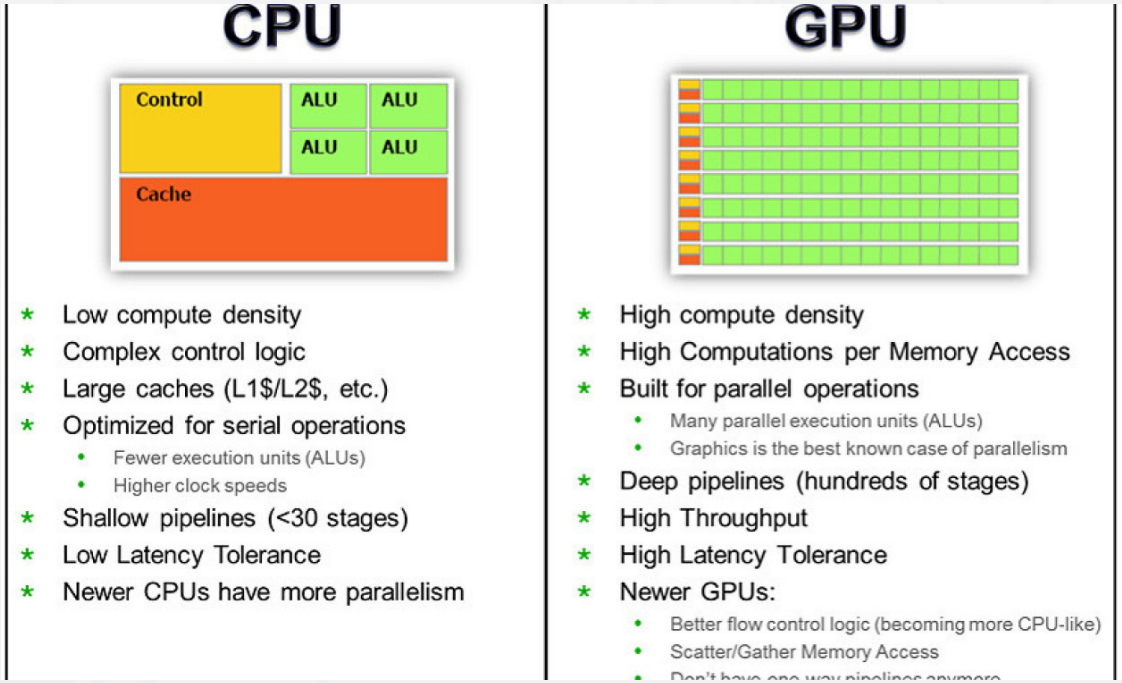
# 为什么需要写出高性能代码



- 帧率
  - FPS(Frames per second)
- 高帧率
  - 更流畅的画面
  - 更低的延迟
- 帧率要求
  - 平台/游戏类型
    - 主机: 60FPS
    - 手机: 30FPS?
    - VR : 90FPS+
  - 平稳



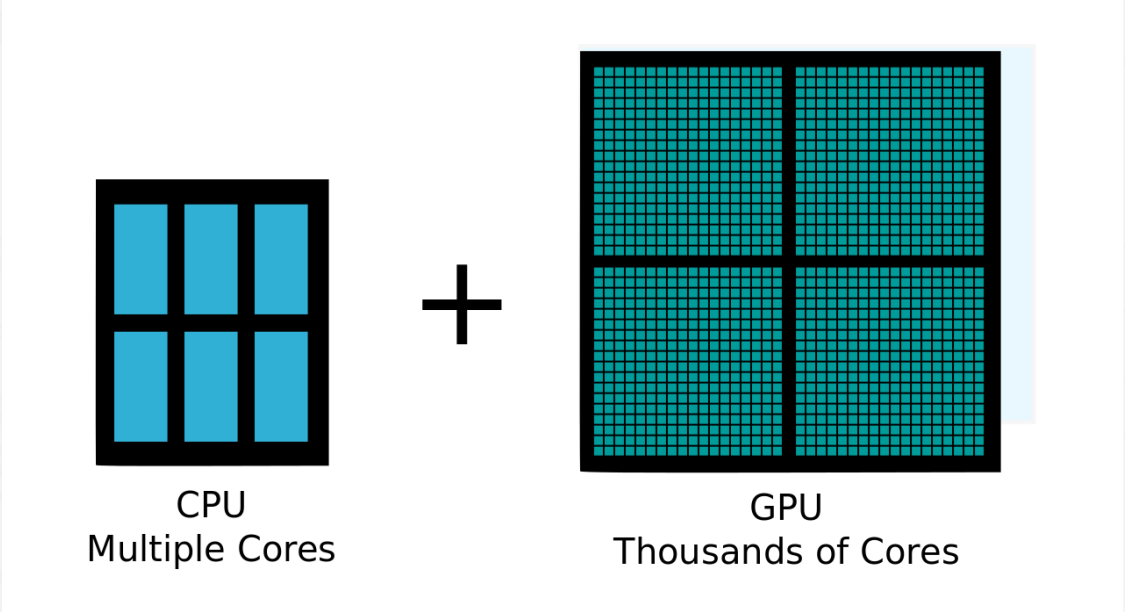
- CPU 与 GPU架构的不同

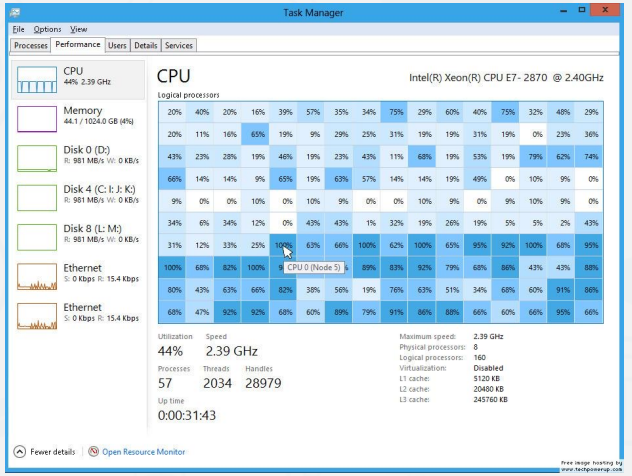


# CPU + GPU



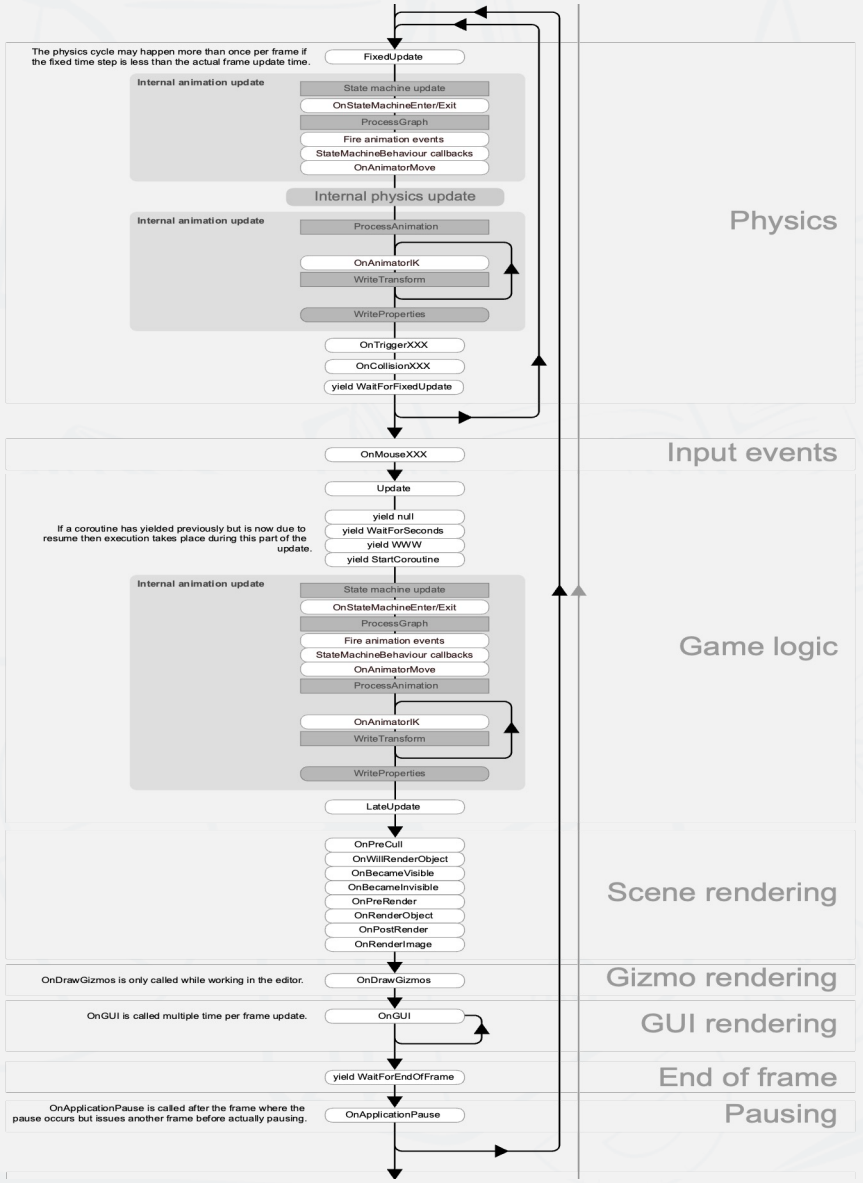
- CPU 和 GPU 需要很好的协作
  - “Batch, Batch, Batch” [GDC2003]
  - CPU的核数也在增长



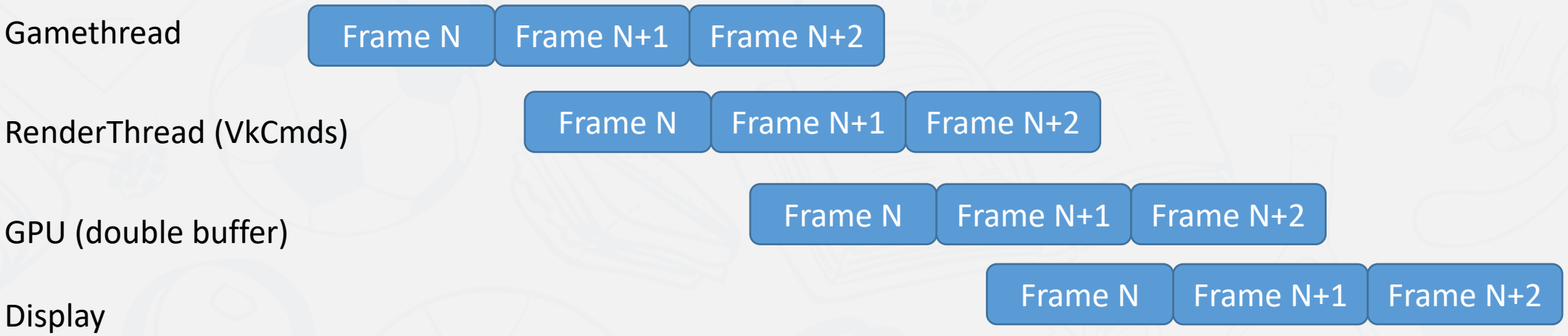
# Game Loop




The physics cycle may happen more than once per frame if the fixed time step is less than the actual frame update time.
FixedUpdate
Internal animation update
State machine update
OnStateMachineEnter/Exit
ProcessGraph
Fire animation events
StateMachineBehaviour callbacks
OnAnimatorMove
Internal physics update
Internal animation update
ProcessAnimation
OnAnimatorIK
WriteTransform
WriteProperties
OnTriggerXXX
OnCollisionXXX
yield WaitForFixedUpdate
Physics
Input events
OnMouseXXX
Update
If a coroutine has yielded previously but is now due to resume then execution takes place during this part of the update.
yield null
yield WaitForSeconds
yield WWW
yield StartCoroutine
Internal animation update
State machine update
OnStateMachineEnter/Exit
ProcessGraph
Fire animation events
StateMachineBehaviour callbacks
OnAnimatorMove
ProcessAnimation
OnAnimatorIK
WriteTransform
WriteProperties
Game logic
LateUpdate
OnPreCull
OnWillRenderObject
OnBecameVisible
OnBecameInvisible
OnPreRender
OnRenderObject
OnPostRender
OnRenderImage
Scene rendering
OnDrawGizmos is only called while working in the editor.
OnDrawGizmos
Gizmo rendering
OnGUI is called multiple time per frame update.
OnGUI
GUI rendering
yield WaitForEndOfFrame
End of frame
OnApplicationPause is called after the frame where the pause occurs but issues another frame before actually pausing.
OnApplicationPause
Pausing

# Gamethread vs Renderthread





# Gamethread vs Renderthread

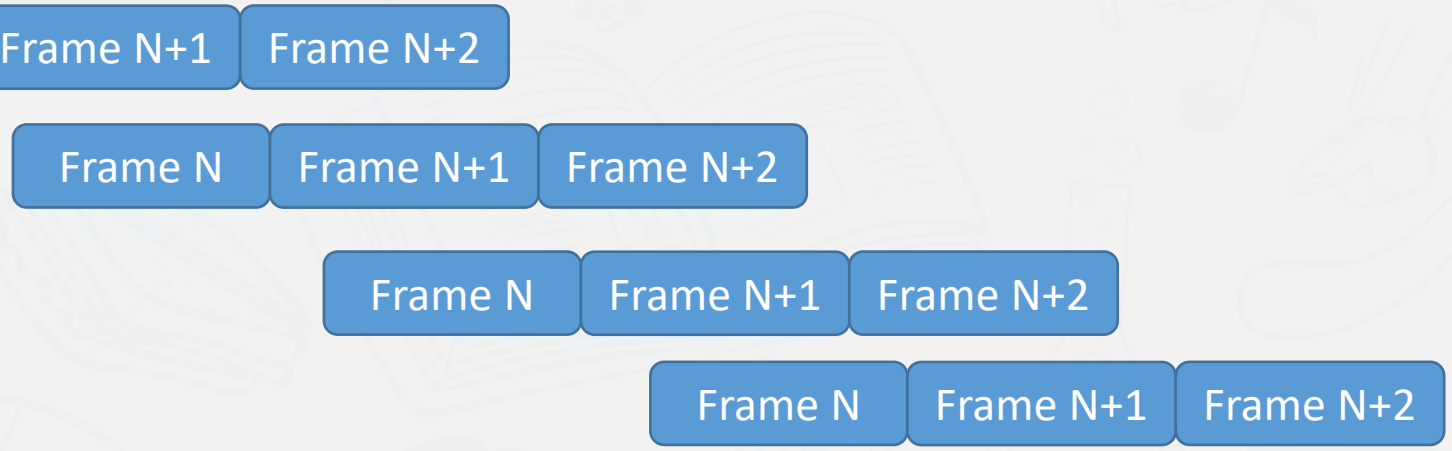


Gamethread

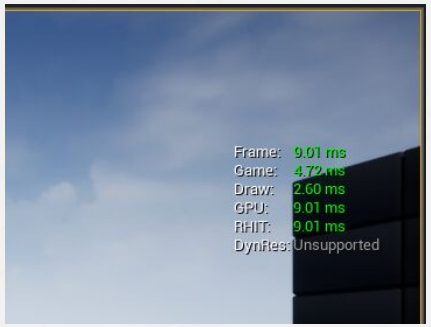
RenderThread (VkCmds)

GPU (double buffer)

Display

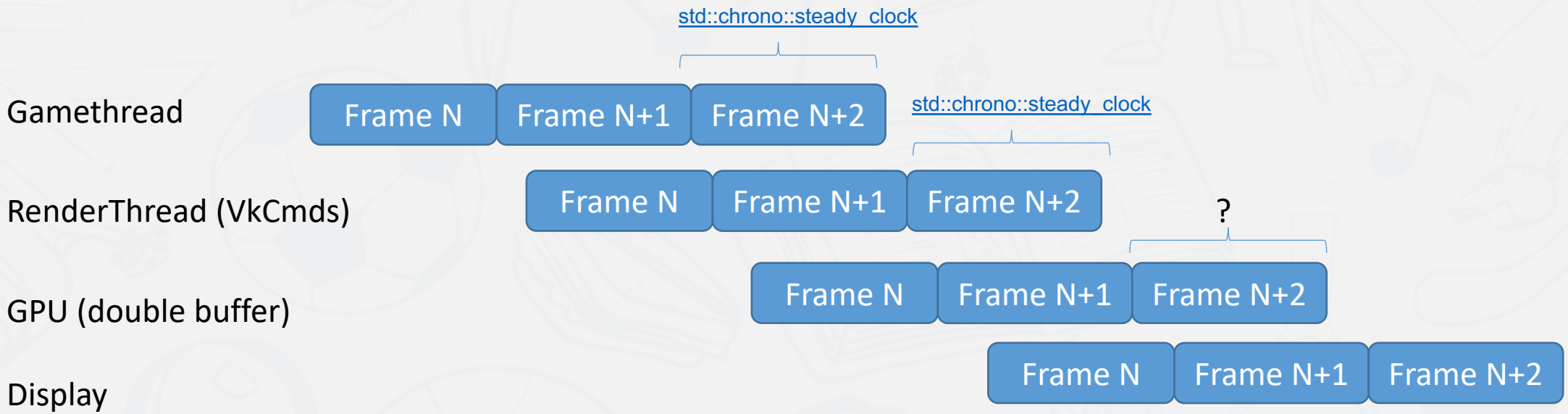


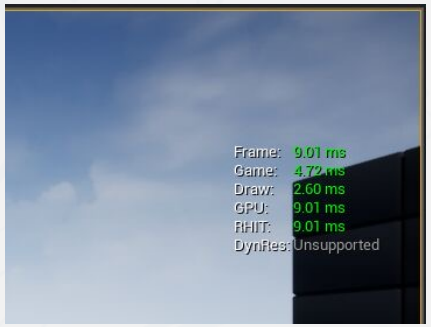

*最慢的一环决定了最终的FPS*



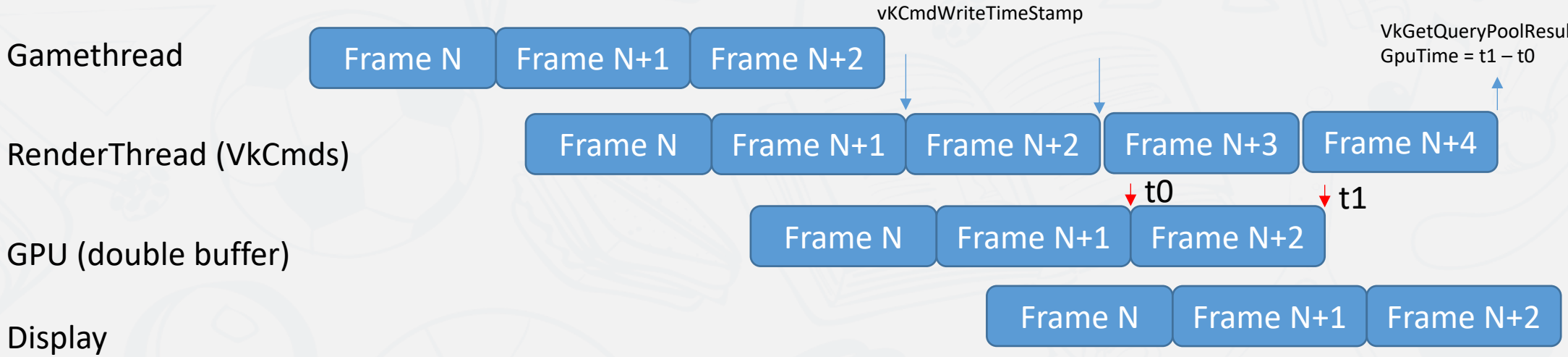
# Gamethread vs Renderthread

- Measure Time



- Measure Time




```
// Provided by VK_VERSION_1_0
void vkCmdWriteTimestamp(
    VkCommandBuffer                             commandBuffer,
    VkPipelineStageFlagBits                     pipelineStage,
    VkQueryPool                                 queryPool,
    uint32_t                                    query);
```

```
// Provided by VK_VERSION_1_0
VkResult vkGetQueryPoolResults(
    VkDevice                                    device,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount,
    size_t                                      dataSize,
    void*                                       pData,
    VkDeviceSize                                stride,
    VkQueryResultFlags                          flags);
```

# 常见瓶颈



**GameThread**

- 脚本
- 物理
- 网络
- …

**RenderThread**

- 动态物体过多
- 灯的数量过多
- DrawCall 过多
- …

**GPU**

- 三角形数量
- 贴图数量
- 分辨率
- Raytracing 过于复杂
- 显存
- …

# Profiling/Debug Tools



**GameThread**

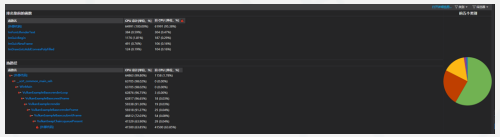
- 脚本
- 物理
- 网络
- …

**RenderThread**

- 动态物体过多
- 灯的数量过多
- DrawCall 过多
- …

**GPU**

- 三角形数量
- 贴图数量
- 分辨率
- Raytracing 过多
- 显存
- …



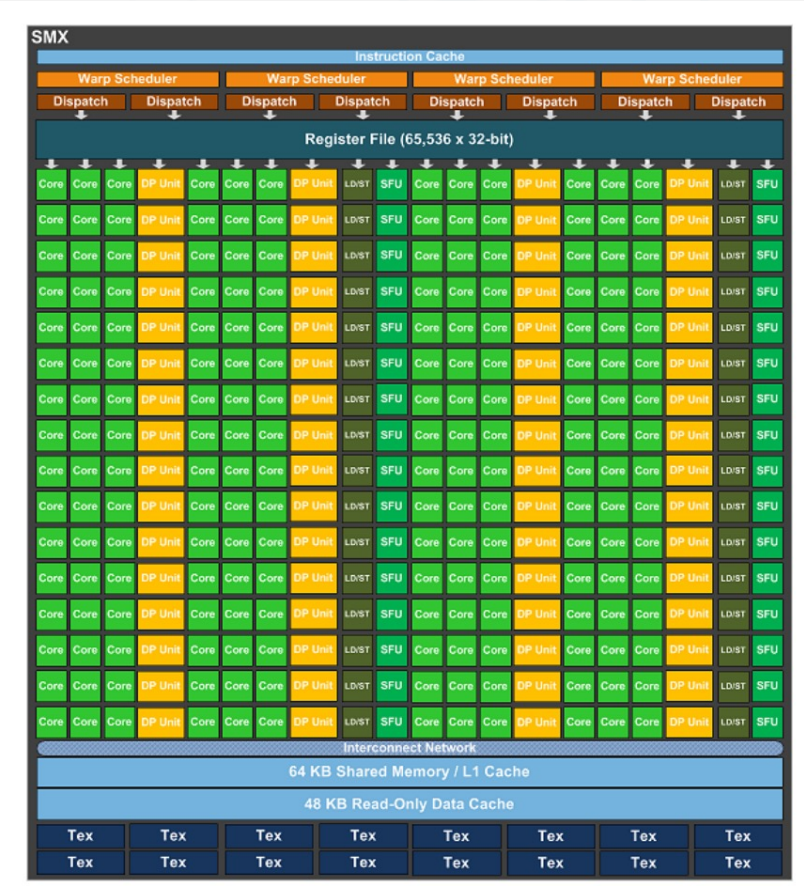
Visual studio Profiler

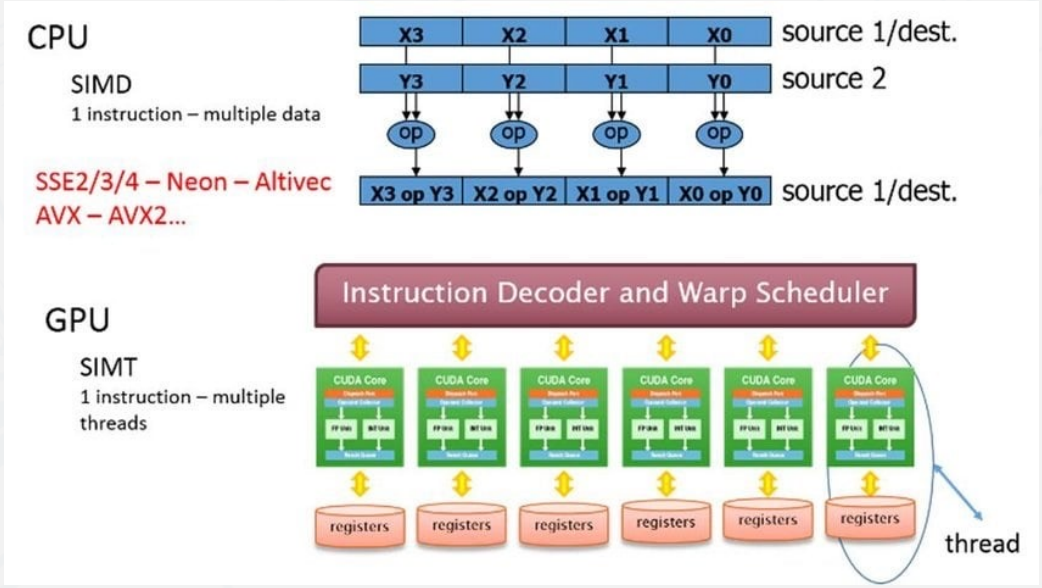

RenderDoc
Nsight
Pix

# Basic GPU Arch

# GPU Architecture

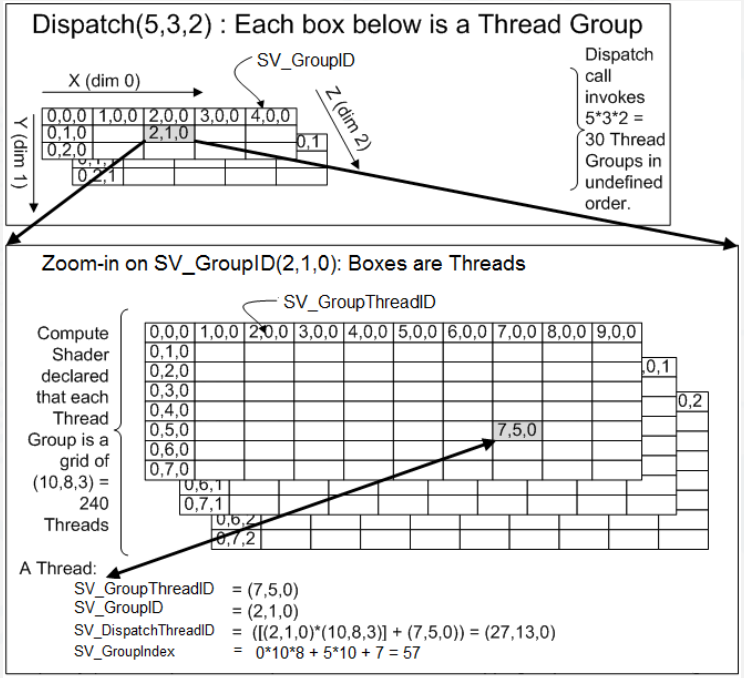
## Basic GPU Arch



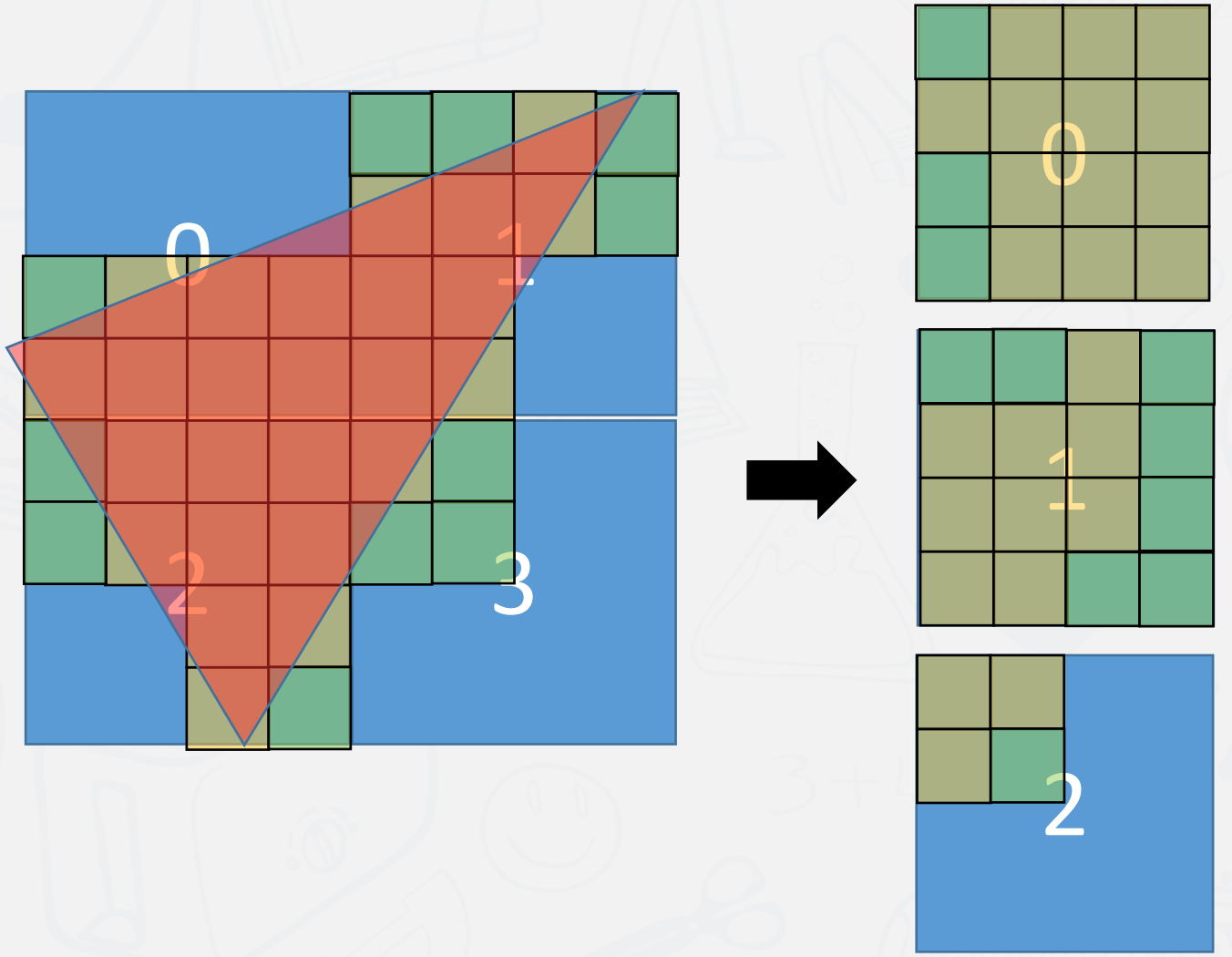
# Warp & Wavefront

- Warp & Wavefront
  - 一次指令发射多个线程一起执行
  - Warp：32 线程，NVIDIA
  - Wavefront ： 64线程 AMD
- SIMT 执行
  - 编程者只用关心自己的线程
- Compute Shader
  - 线程组中warp数量 ： （thread + 32-1 ）/32
  - Numthread(10,8,3) : 8 warp
  - Inactive Thread/lanes: 8*32 -240 = 16



# Warp & Wavefront

- Warp & Wavefront
- SIMT 执行
- Compute Shader
- Vertex Shader
  - Batch Processing
  - Post-Transform Vertex Cache
- Pixel Shader
  - Quad 为单位
  - Tile Walk & PS Wave pack
  - Helper lane/inactive thread




[Drobot. et al 2020]

# 分支



- 以Warp为单位执行指令
  - Case 1: Warp 内走的是不同分支
  - Case 2: Warp 内走的是一样的分支

Case 1

```
int tid = get_local_id(0)
if ( tid % 2 == 0) //Even Work Items
        DoSomeWork()
else
        DoSomeWork2()
```

Conditional – With divergence

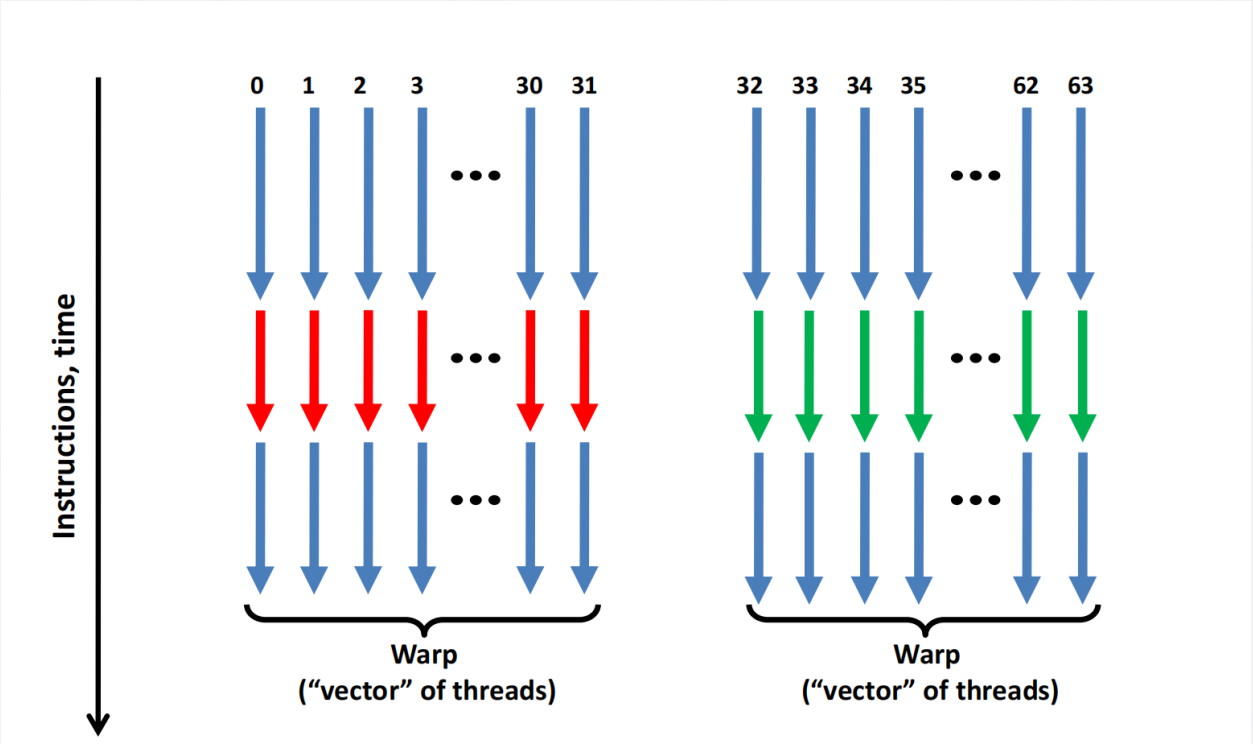
Case 2

```
int tid = get_local_id(0)
if ( tid / 64 == 0)  //Full First Wavefront
        DoSomeWork()
else if (tid /64 == 1) //Full Second  Wavefront
        DoSomeWork2()
```

Conditional – No divergence

# 分支

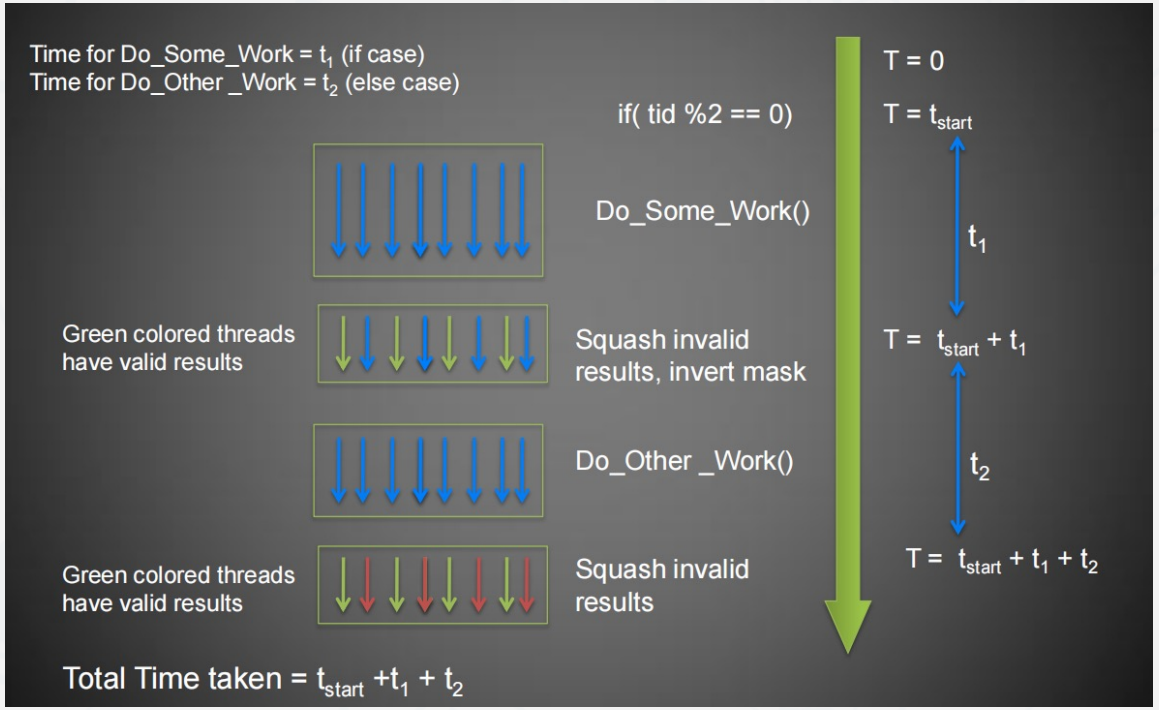
- Case 1
  - 没有性能负担
  - 比如uniform




Case 1

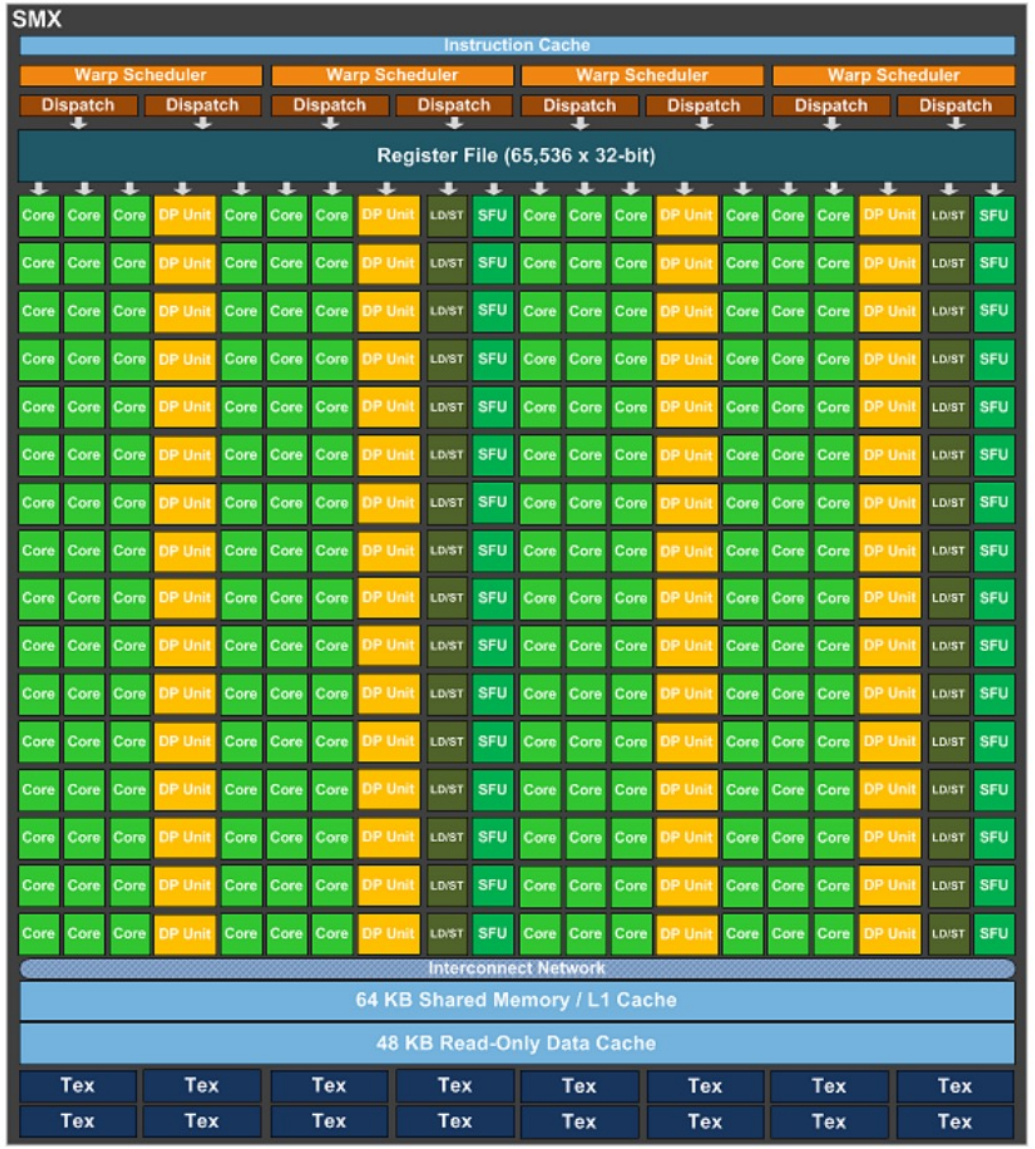
# 分支

- Case 2
  - 两个分支都会走
  - 算术指令都会执行
  - 访存: 读和写都不会发生



# Warp Scheduler

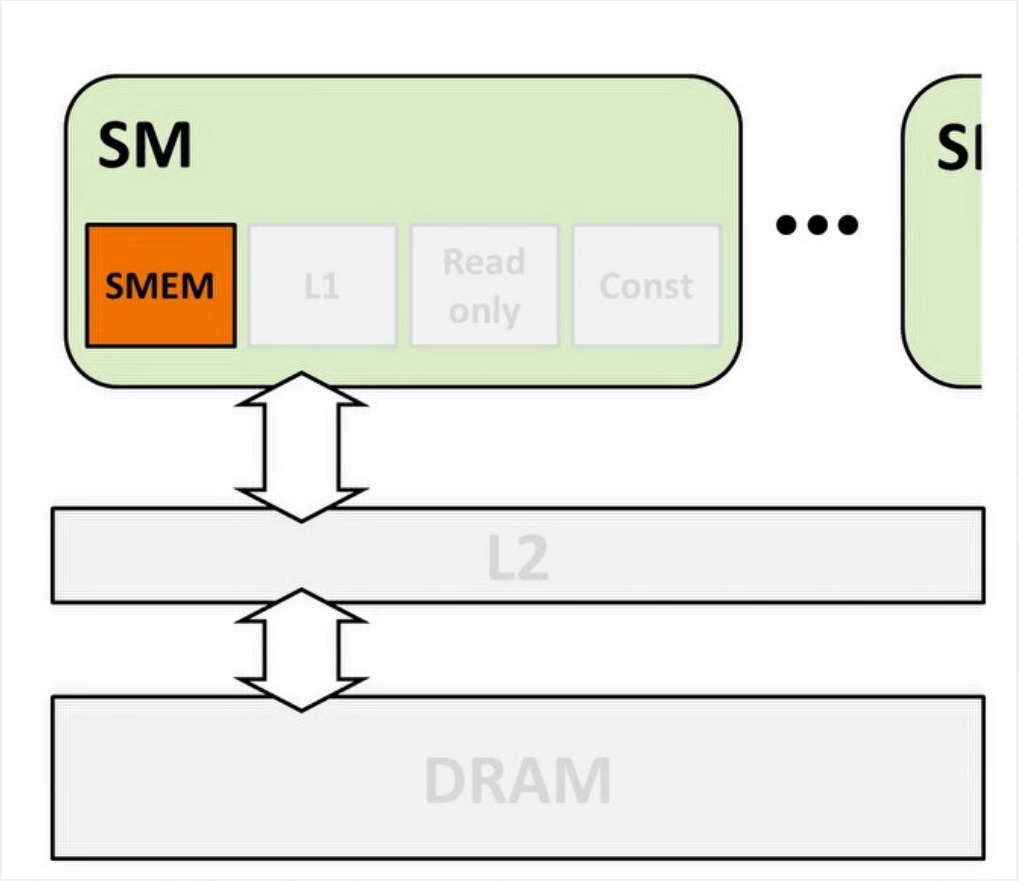
- 4个Warp Scheduler，双发射
- Instruction Latency
- Warp Switching
  - 没有Overhead
- 寄存器/SMEM资源是预先分配的
  - Threadblock
  - Occupancy



# Memory model

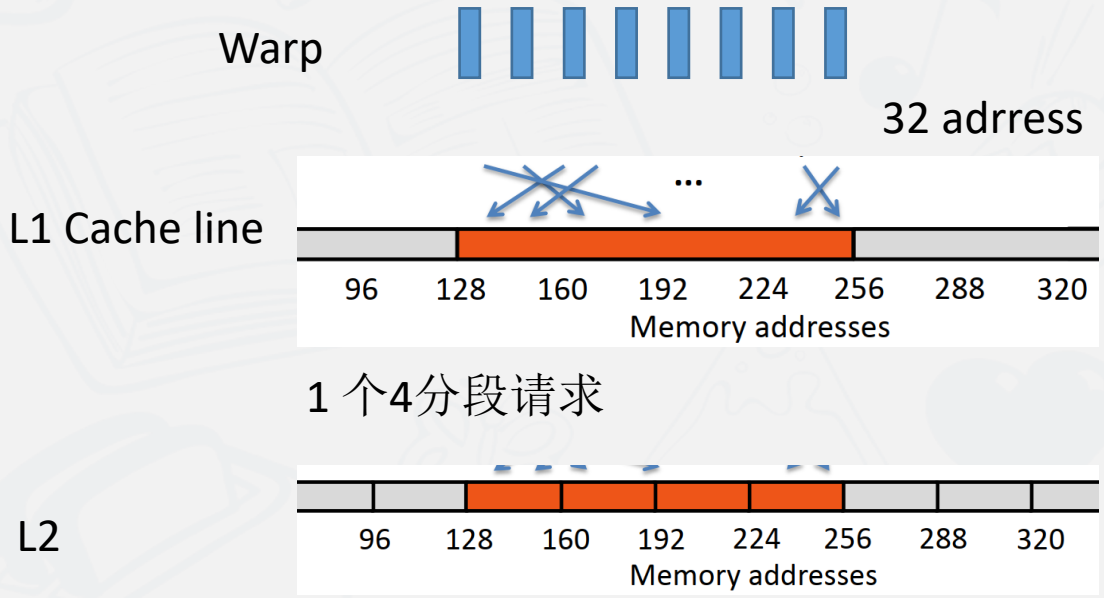


- DRAM
  - Read /write to Video RAM
- L2
- L1
- Read Only
  - Textures
- Const
  - unforms
- Shared Memory



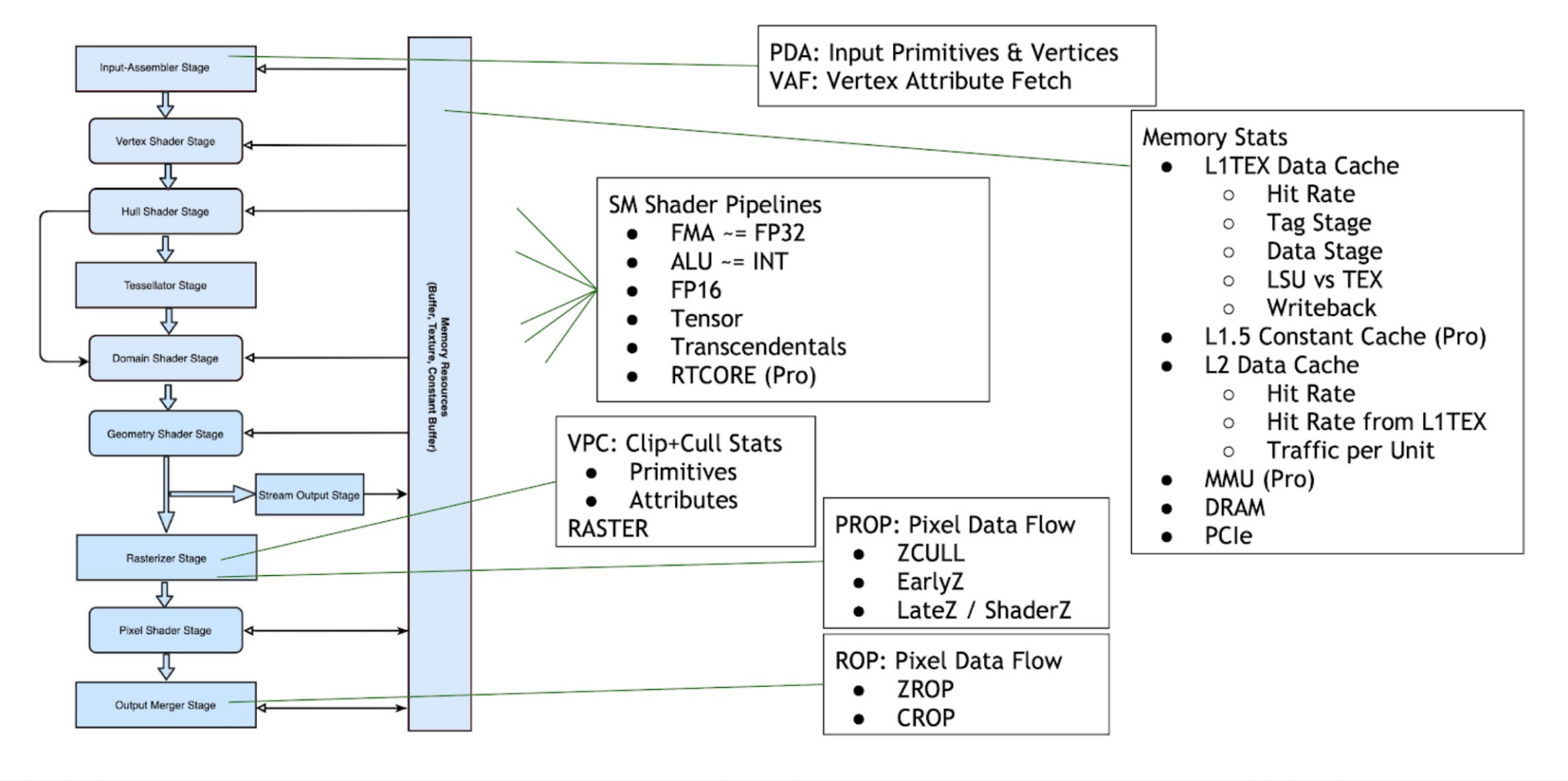
# Memory Access

- Cache Line :128 bytes
- Segments: 32bytes
- Warp 提供32个地址
  - 硬件整合成请求
  - Texture是4个thread为单位



# GPU Components



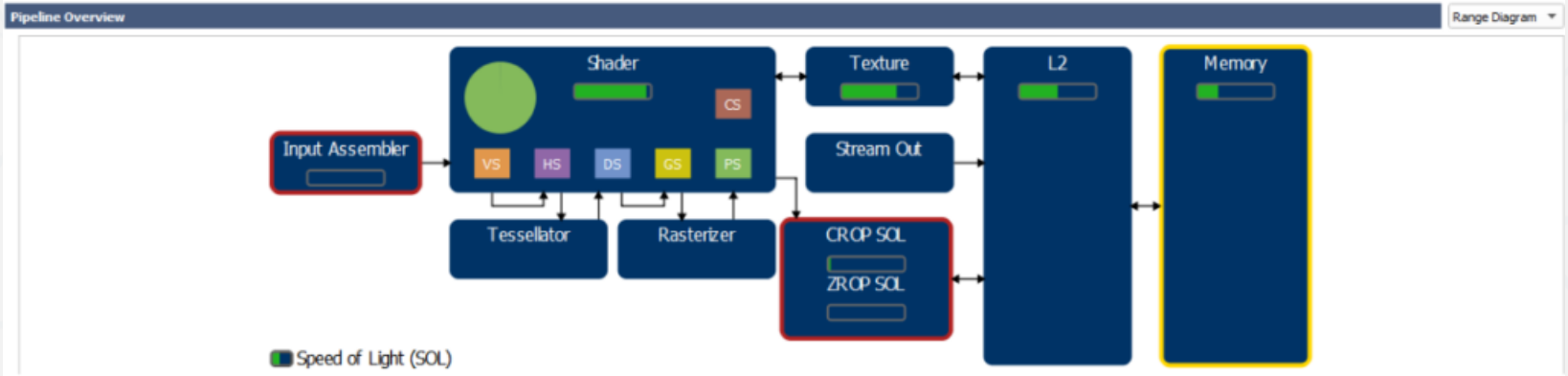


# Nsight GPU Profiling

# Nsight Graphics



- Nvidia GPU Metrics
- Throughput based
  - SOL

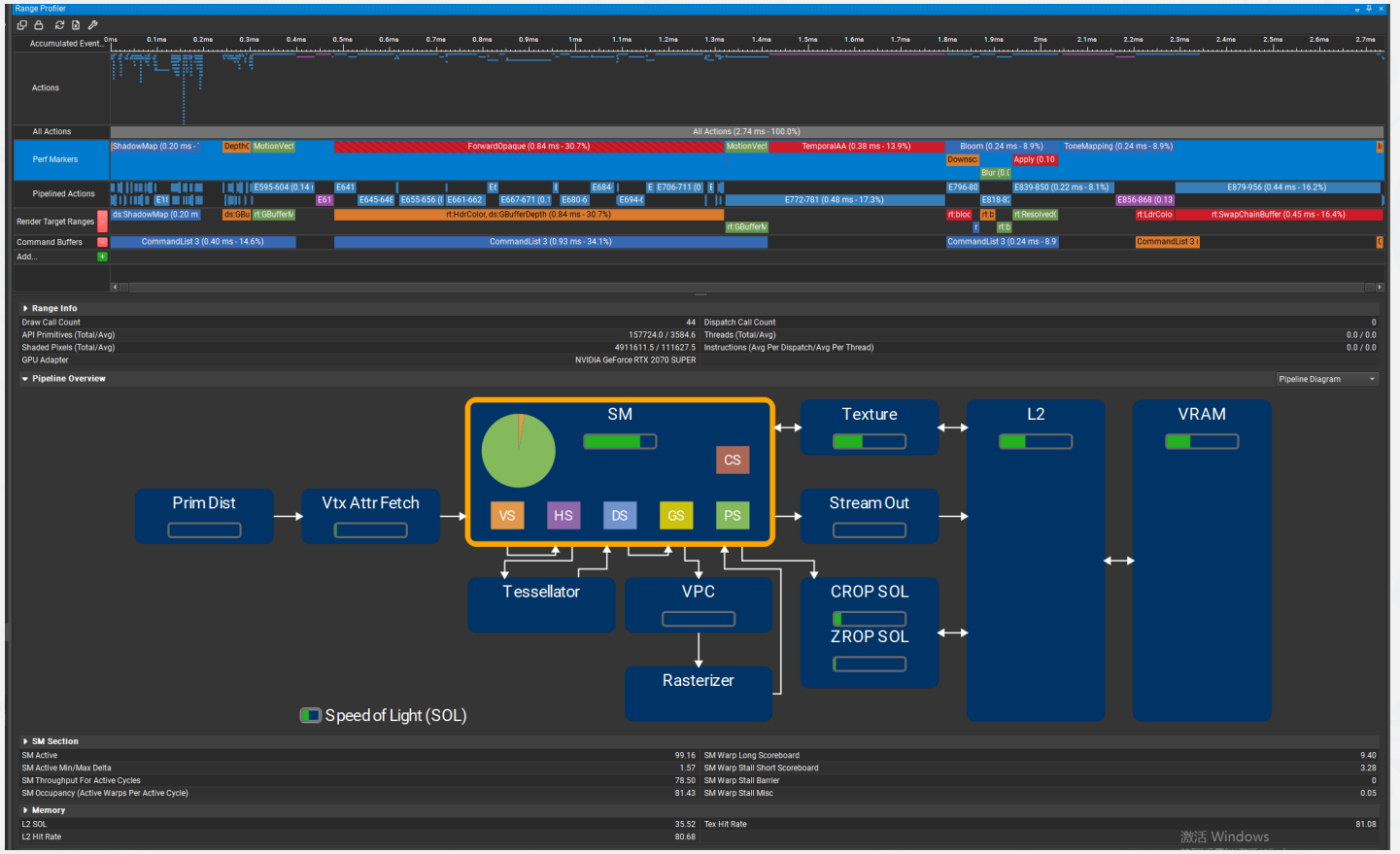



*"The Peak-Performance-Percentage Analysis Method for Optimizing Any GPU Workload"*

# Nsight Frame Profiler



- 基于指令重放
- 可以分析每一个GPU Event的具体情况

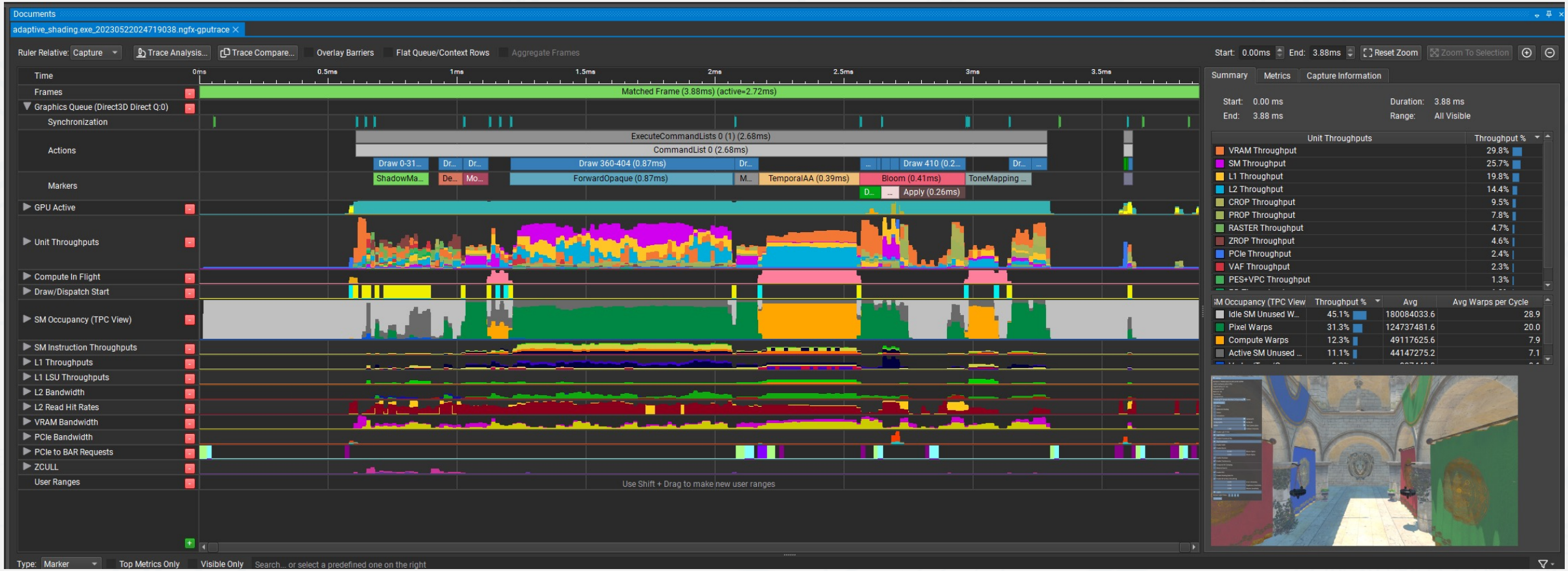


# Nsight GPU Trace



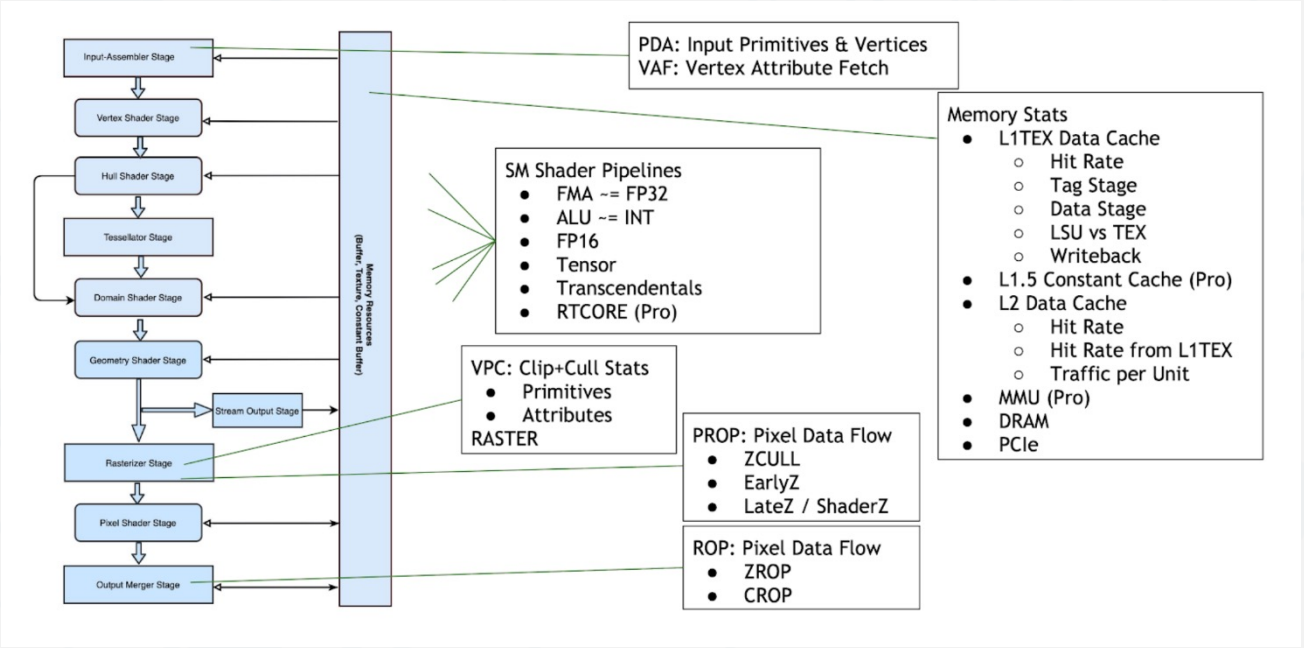


- 即时的结果，实时捕获，时间值上更准确。



# Details

- VAF
- PD
- PES+VPC
- RASTER
- SM
- L1TEX
- L2
- VRAM
- ZROP
- CROP



# SM Occupancy



- 一个SM上最多活跃多少个Warp
  - Hide latency
- Register Number
  - GPU Profiler->Linked Program

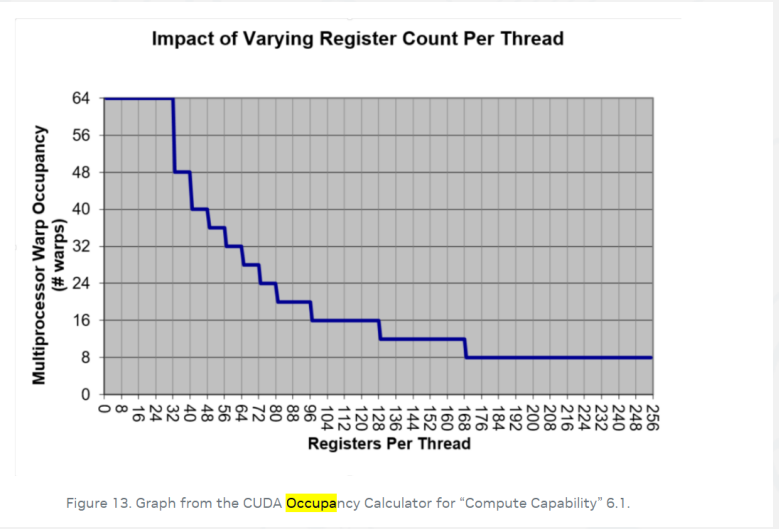
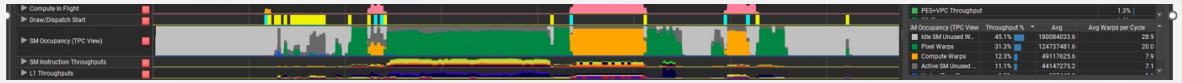
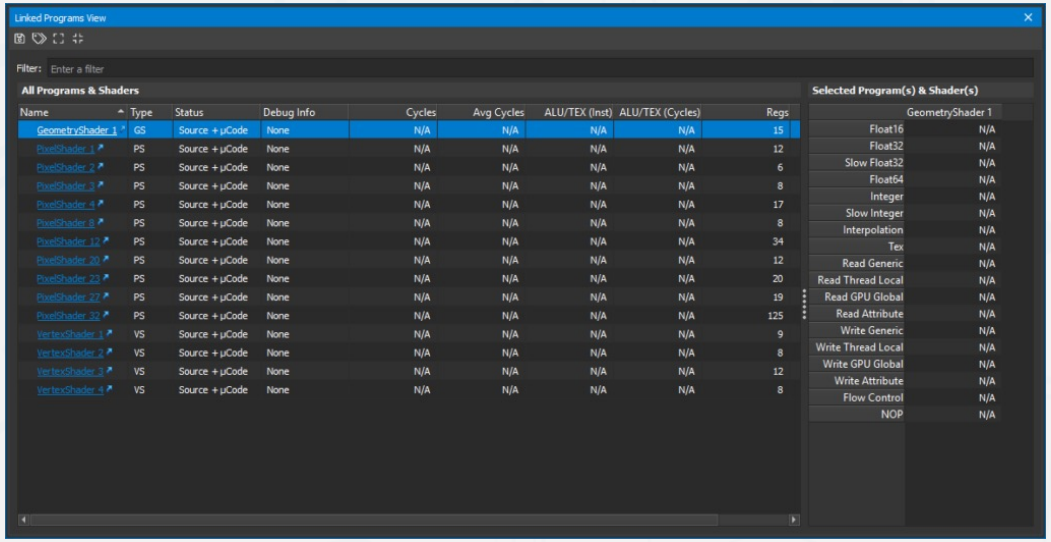
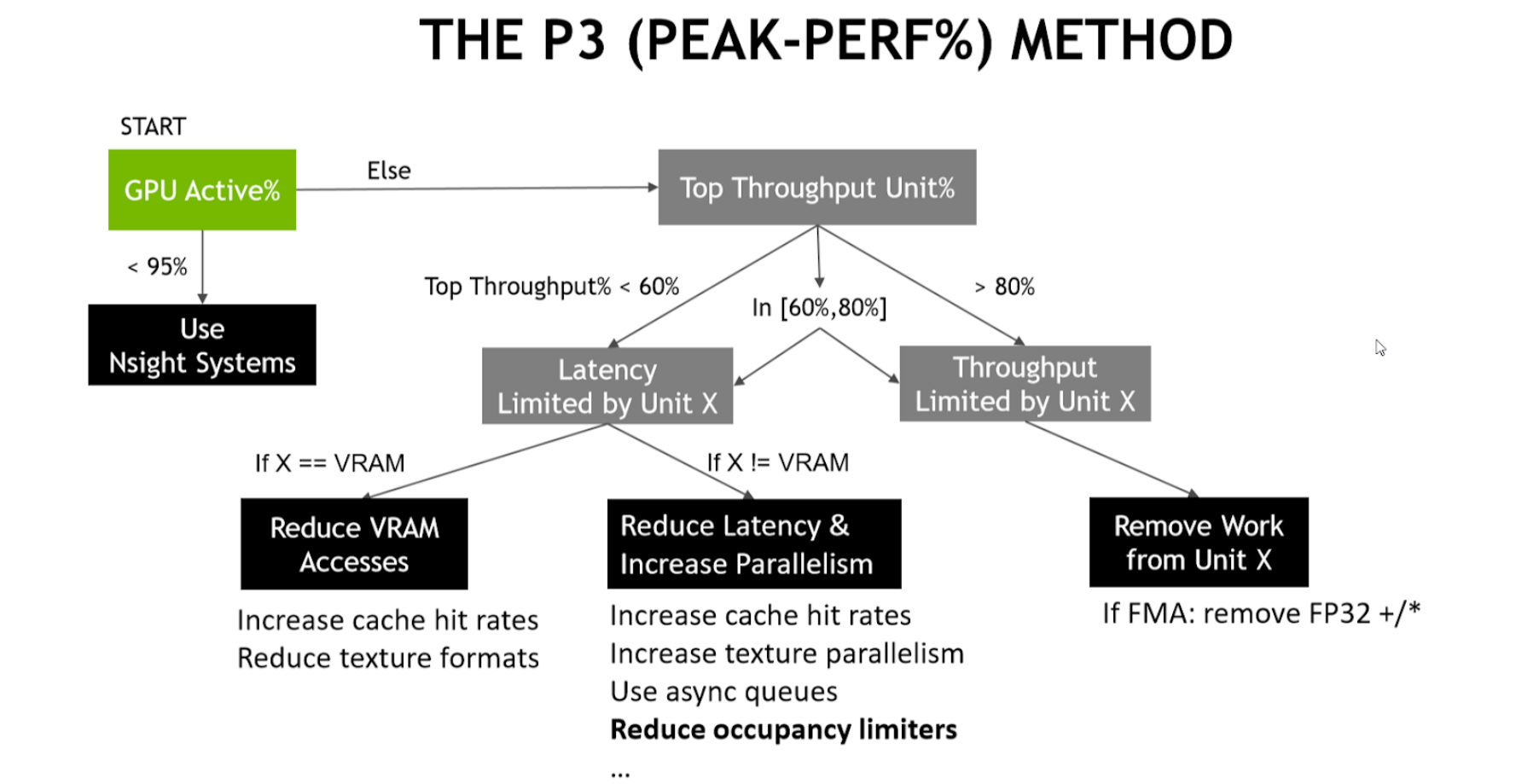



Figure 13. Graph from the CUDA Occupancy Calculator for "Compute Capability" 6.1.



SM Occupancy






*不同的平台瓶颈可能不一样，以实测为基准！*

# 后续课程



- VAF
- PD
- PES+VPC
- RASTER
- SM
- L1TEX
- L2
- VRAM
- ZROP
- CROP

Geometric LOD

Multi Rate Shading

Shader LOD

Texture LOD

# Reference

- https://developer.nvidia.com/blog/the-peak-performance-analysis-method-for-optimizing-any-gpu-workload/
- https://on-demand.gputechconf.com/gtc/2013/presentations/S3466-Programming-Guidelines-GPU-Architecture.pdf
- https://docs.nvidia.com/nsight-graphics/UserGuide/index.html