Capturing and Visualizing RealTime GPU Performance in Mortal Kombat X

Adisak Pochanayon
Principal Software Engineer
NetherRealm Studio

Why make a GPU Visualizer?

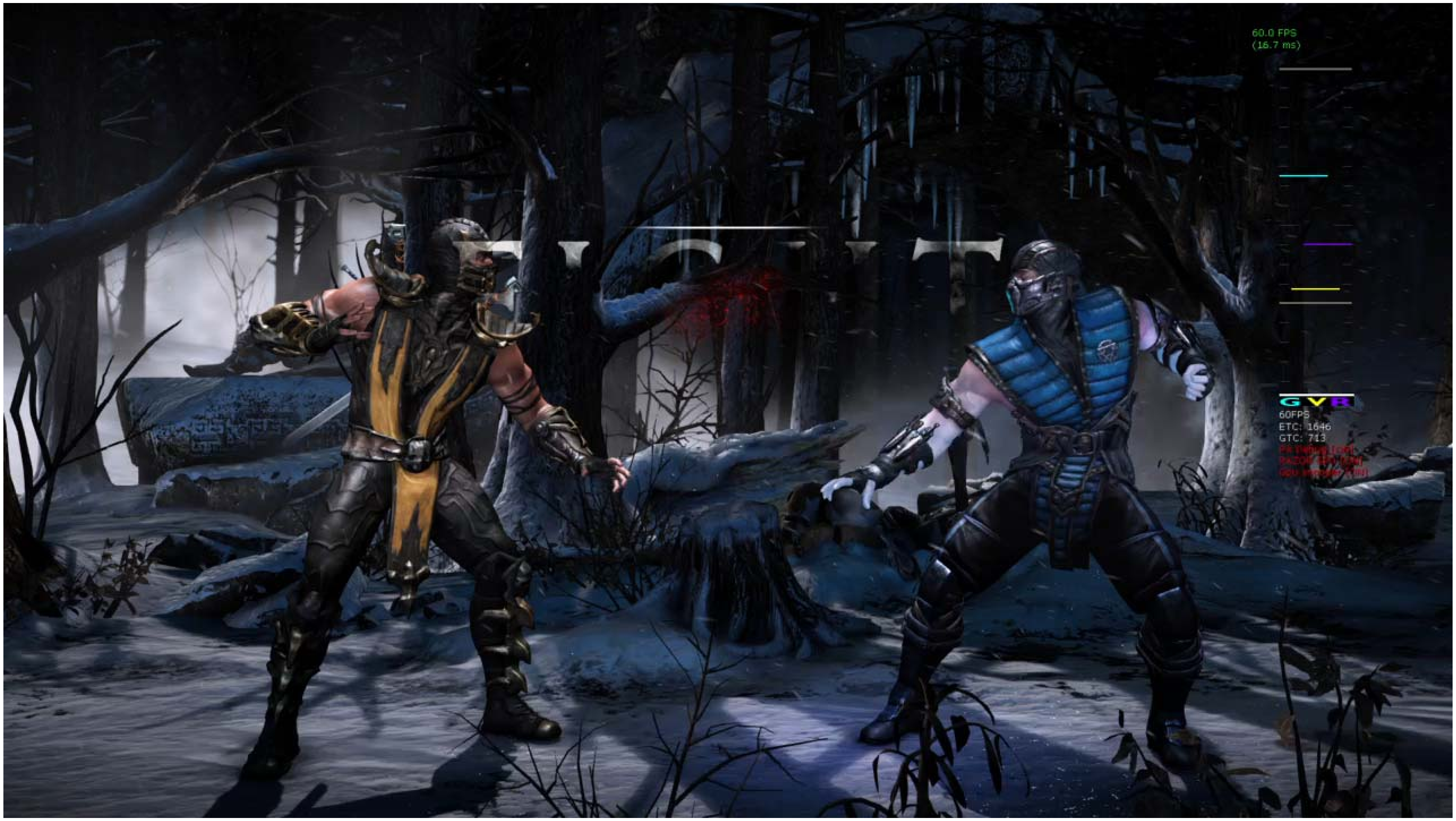
- Pix and Razor Already Exist
  - Great for snapshots / single frames
  - Good for examining low level resources etc
  - Not so good for continuous capture or for investigating spikes (hard to capture spike frame)

GPU Snooper Benefits

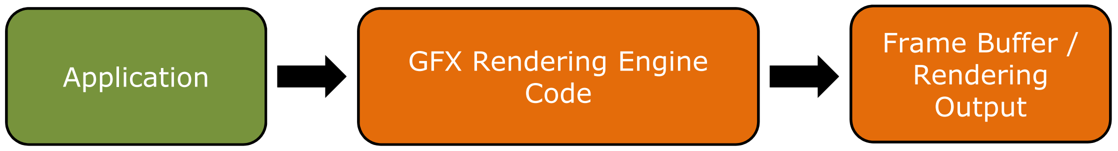
- Realtime Live Profiling (no need to stop game or run analysis app)
- Easy testing of Live scenario changes (recompile and hot-reload pixel shader – immediately see improvements)
- Flipping binary switch in code and looking at live view differences
- Correct Global Hierarchy Level (vs PIX for multiple contexts)

GPU Snooper

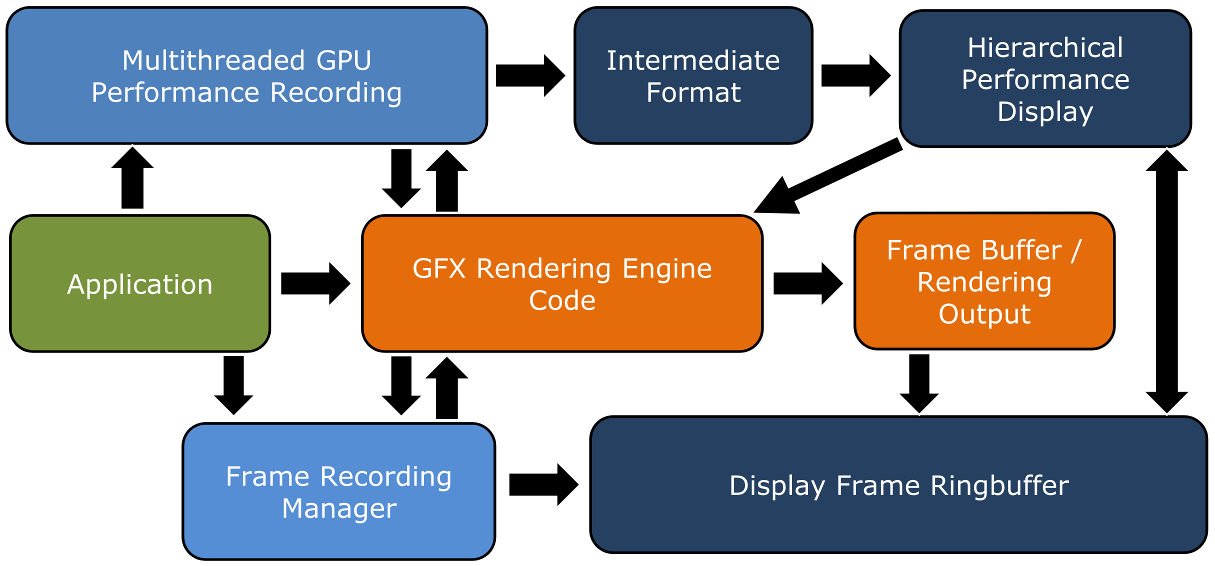
- Quick Architecture Overview
- History
- Implementation
  - Performance and Memory Optimizations
  - Supporting multithreading and multiple contexts
  - Don’t be scared – this is all “easy”*


60.0 FPS
(16.7 ms)
GVR
60FPS
ETC: 1646
GTC: 713

Game Application



NetherRealm Current GPU Profiler Architecture


Multithreaded GPU Performance Recording
Intermediate Format
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output
Frame Recording Manager
Display Frame Ringbuffer

GPU Snooper History as a Debugging Tool

GPU Snooper was a Debug Tool

- GPU Hangs were a Black Box ☹
- Catch GPU Hangs and Stalls / Crashes
  - Originally designed with “printf” debugging early on in the XBOne/PS4 console development cycle before GPU crash dumps and debugging
  - Triggered by code (i.e. timer watchdog) or manually with console command

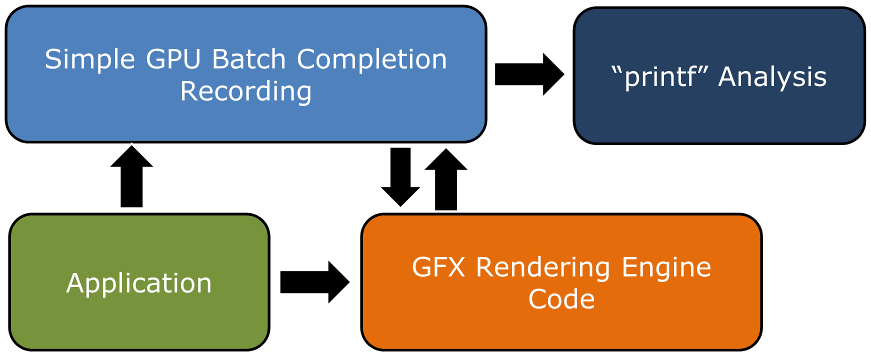
```
GPU exception was detected.
Rendering Thread (or GPU) Possibly hung.  Waiting on label for 123.42 seconds

GPU SNOOPER - BEGIN INVESTIGATING TRACE
BEGIN GPU Marker TRACE at HANG / CRASH
|
Audio
WorldTick
RenderScaleformPrepass
CC Texture Alpha
Scene Captures
Render Scene
    Init Views
    BeginRenderingSceneColor
    PreSkinning
    Build Early Shadow Set
    Init Dynamic Lights
    DPG World
        ClearView
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!                                 !!!!
!!!!  GPU IS HUNG SOMEWHERE IN HERE  !!!!
!!!!                                 !!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        PrePass
            View0
                Dynamic
                PosOnly Opaque
                Opaque
                Masked
        RenderVelocities
            View0
        AmbientOcclusion
            LinearizeDepth
            HBAO
```

```
              . . . Lots of stuff removed to fit . . .
UI
    RenderScaleform
        GFxBeginDisplay
        GFxDrawProcessedPrimitive
        GFxDrawProcessedPrimitive
        GFxEndDisplay
    Deferred Batched Element Rendering
    (Spectator)Deferred Batched Element Rendering
BLOCK ON RT GRAPH
Deferred Batched Element Rendering
(Spectator)Deferred Batched Element Rendering
    Batched Elements

END GPU Marker TRACE at HANG / CRASH
GPU Marker Stack at HANG / CRASH

       !!! WARNING !!! MARKER STACK MAY BE CORRUPT !!!

       Render Scene
       DPG World
       ClearView

Marker (Depth: 3) Text: ClearView
GPU SNOOPER - END INVESTIGATING TRACE
```

Implementing “printf” Debug

- Create GPU Label
  - Simply memory accessible by GPU where we can write a value that is later read by CPU
  - PS4 == context.writeImmediateAtEndOfPipe()
- Save GPU work trace
  - Enter / Exit Markup already done in code for PIX / RAZOR
  - Save “string” and Level
  - Write pointer to string in Label

GPU Snooper "Debugger" (good for catching GPU hangs and crashes)


Simple GPU Batch Completion Recording
"printf" Analysis
Application
GFX Rendering Engine Code

GPU Snooper Transitions into a Profiling Tool

Request for Performance Timing?

- So… add simple timing
  - Along with capturing strings (and level) also capture start / stop times
- Turns out to be remarkably easy to do

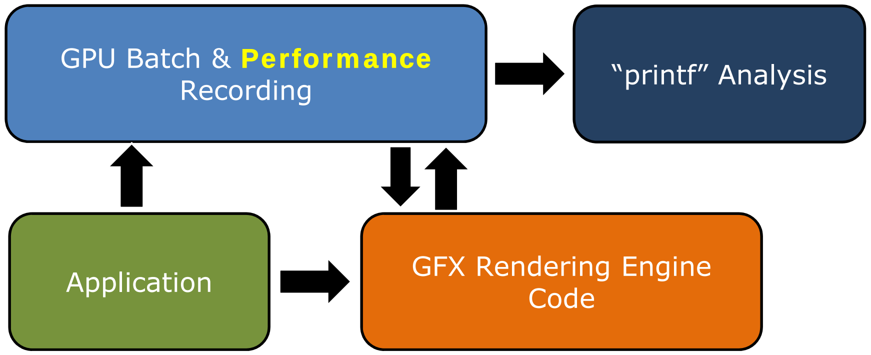
PS4 GPU Timer

```
sourceCPU = sce::Gnm::kEventWriteSourceGlobalClockCounter;
sourceGPU = sce::Gnm::kEventWriteSourceGpuCoreClockCounter;
srcSelector = sourceGPU;

GFXContext->m_dcb.writeAtEndOfPipe(
           sce::Gnm::kEopFlushCbDbCaches,
           sce::Gnm::kEventWriteDestMemory,
           GPUClockTimestamp, // Shared Onion Memory
           srcSelector,
           0,
           sce::Gnm::kCacheActionNone,
           sce::Gnm::kCachePolicyLru
           );
```

XBOne GPU Timer

```
// Create D3D Query
ID3D11Query *timestamp;
D3D11_QUERY_DESC QueryDesc;
appMemzero(&QueryDesc, sizeof(D3D11_QUERY_DESC));
QueryDesc.Query     = D3D11_QUERY_TIMESTAMP;
QueryDesc.MiscFlags = 0;
pD3DDevice->CreateQuery(&QueryDesc, &timestamp);

// Inject Timestamp into GPU Context
pD3DDeviceContext->End(timestamp);
```

First Pass GPU Performance Profiling


GPU Batch & **Performance** Recording
"printf" Analysis
Application
GFX Rendering Engine Code

GPU Profiler ???

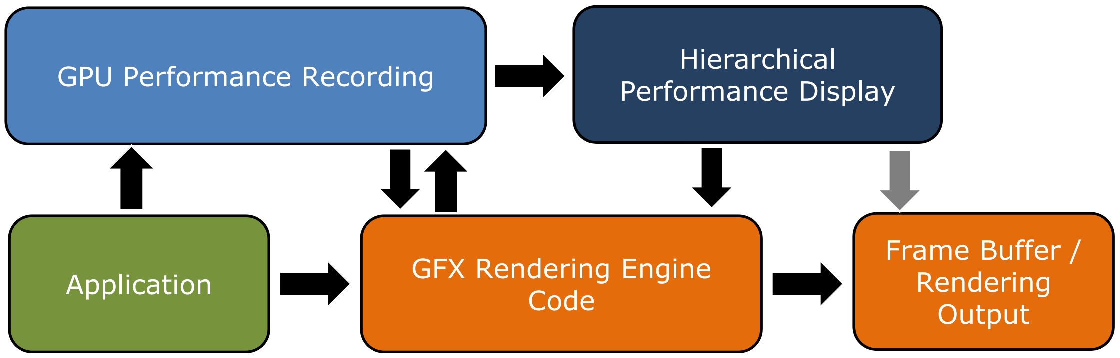
- So now we had timestamps
- We're all done right?
  - By computing delta's we could see time per markup
  - Added timings to printf debugging
- Wrong! Not Done ☹ -- Problems
  - One frame snapshot / Reading printf's is a pain

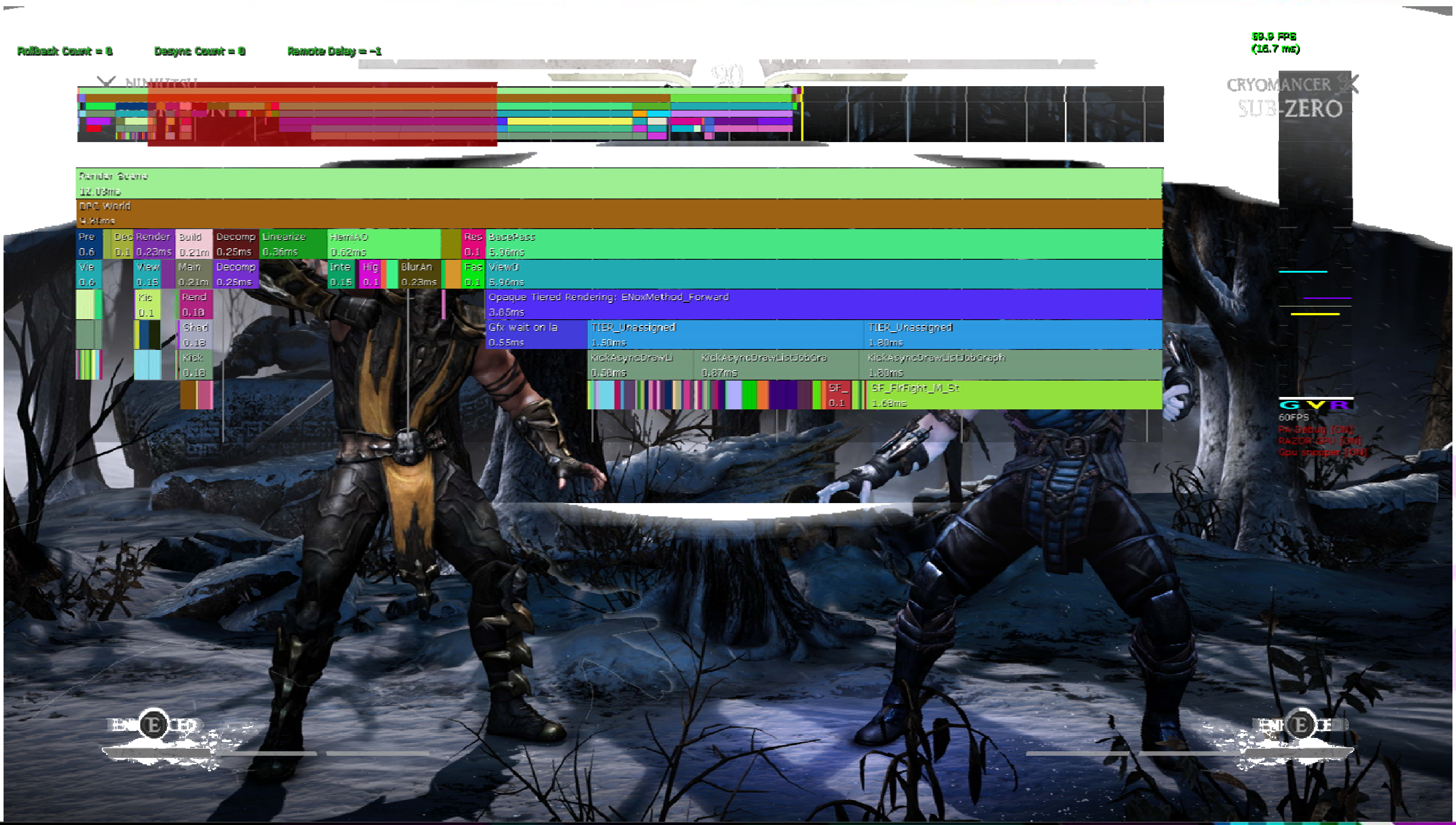
Making GPU Snooper Profiling Useable With Visual Analysis

Solution == GUI Visualizer

- printf’s -> hierarchical time graph
- Simple very portable drawing code
  - Uses only 2 render primitives
    - Translucent Rectangles
    - Text / Strings
    - Optimization: Drawn as two batches / render calls

NetherRealm First-Pass GPU Profiler Display


GPU Performance Recording
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output



```
Rollback Count = 0        Desync Count = 0        Remote Delay = -1

59.9 FPS
(16.7 ms)

Render Scene
12.03ms
DPC World
9.81ms
Pre  Dec  Render   Build   Decomp   Linearize  HemiAO   Res  BasePass
0.6  0.1  0.23ms   0.21m   0.25ms   0.36ms     0.62ms   0.1  5.96ms
Vie       View     Main    Decomp   Inte  Hig  BlurAn   Fas  View0
0.6       0.15     0.21m   0.25ms   0.15  0.1  0.23ms   0.1  5.96ms
          Kic      Rend                                      Opaque Tiered Rendering: ENoxMethod_Forward
          0.1      0.18                                      3.85ms
                   Shad                                      Gfx wait on la  TIER_Unassigned              TIER_Unassigned
                   0.18                                      0.55ms          1.50ms                       1.80ms
                   Kick                                                      KickAsyncDrawLi  KickAsyncDrawListJobGra  KickAsyncDrawListJobGraph
                   0.18                                                      0.58ms           0.87ms                   1.80ms
                                                                                              SF_  SF_FlrFight_M_St
                                                                                              0.1  1.68ms

60FPS
PA Debug [ON]
RAZOR GPU [ON]
Gpu snooper [ON]
```

Hierarchical Chart Drawing

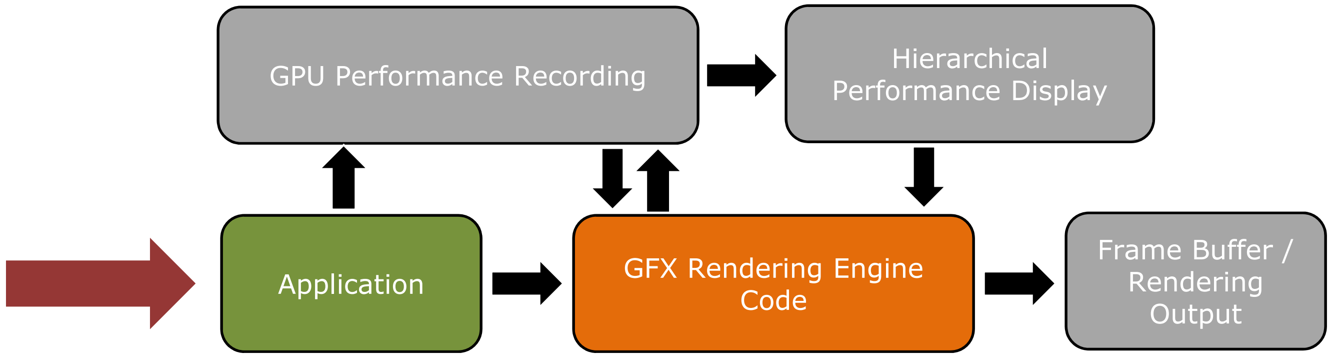
- How did we draw the chart
    - Draw background and time markers for scale
    - Draw blocks that exceed a minimum time
        - Time is X value
        - Level (depth) is Y value
        - Limited depth to a user defined maximum
        - Have a small color separator on blocks
    - Truncate String based on size (minimum 3 characters)
    - Color of block based on quick hash of string

Visualizer == GPU Profiler

- When easy to use, people use tool
  - Mostly GFX Programmers at first for optimization
  - But also artists and designers when looking at bog

Implementation Details: Game Side

Implementation – Game Side

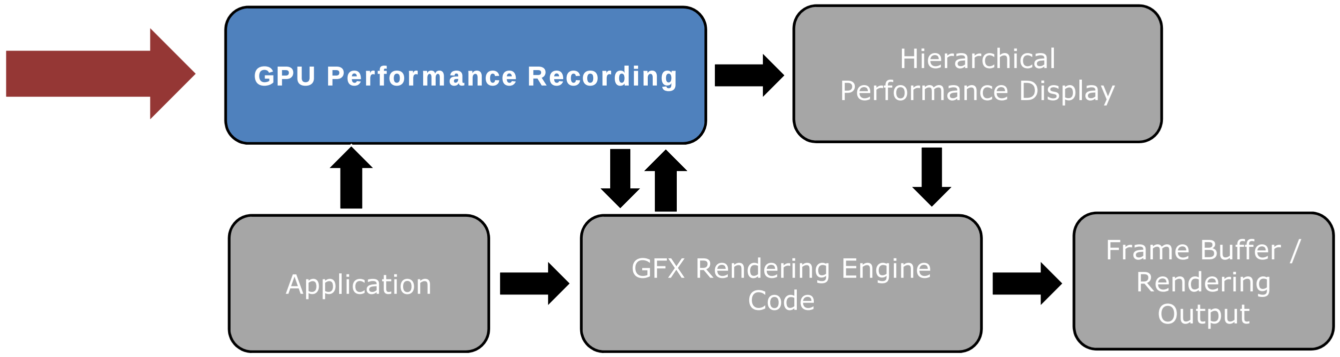

GPU Performance Recording
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output

Game and Engine Modifications

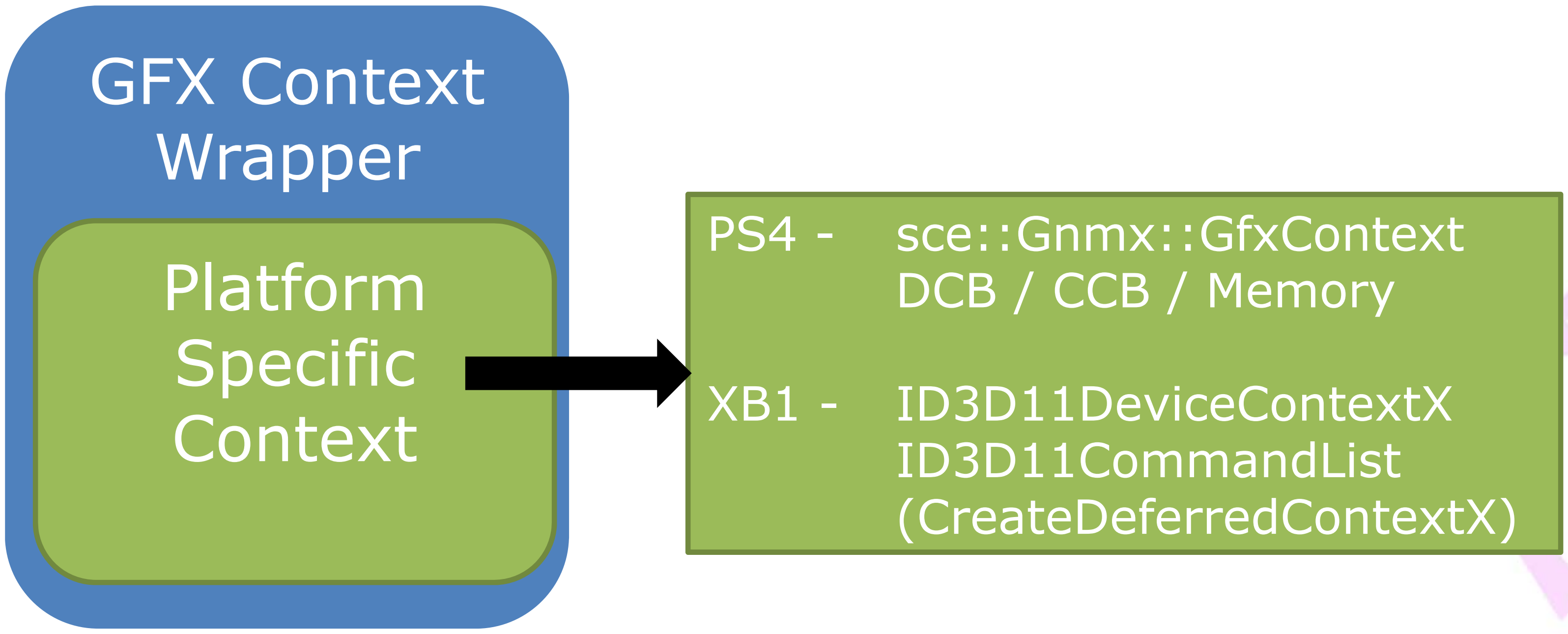
- Profiling Requires Manual Instrumentation
  - 99% chance you're already doing this
    - PS4 / Razor – pushMarker / popMarker
    - XB1 / PIX – BeginDrawEvent / EndDrawEvent
- We use a scoped wrapper…
  - C++ object pushes marker in constructor / pops in destructor
  - Compiles out completely from retail builds

Implementation Details: Performance Recording

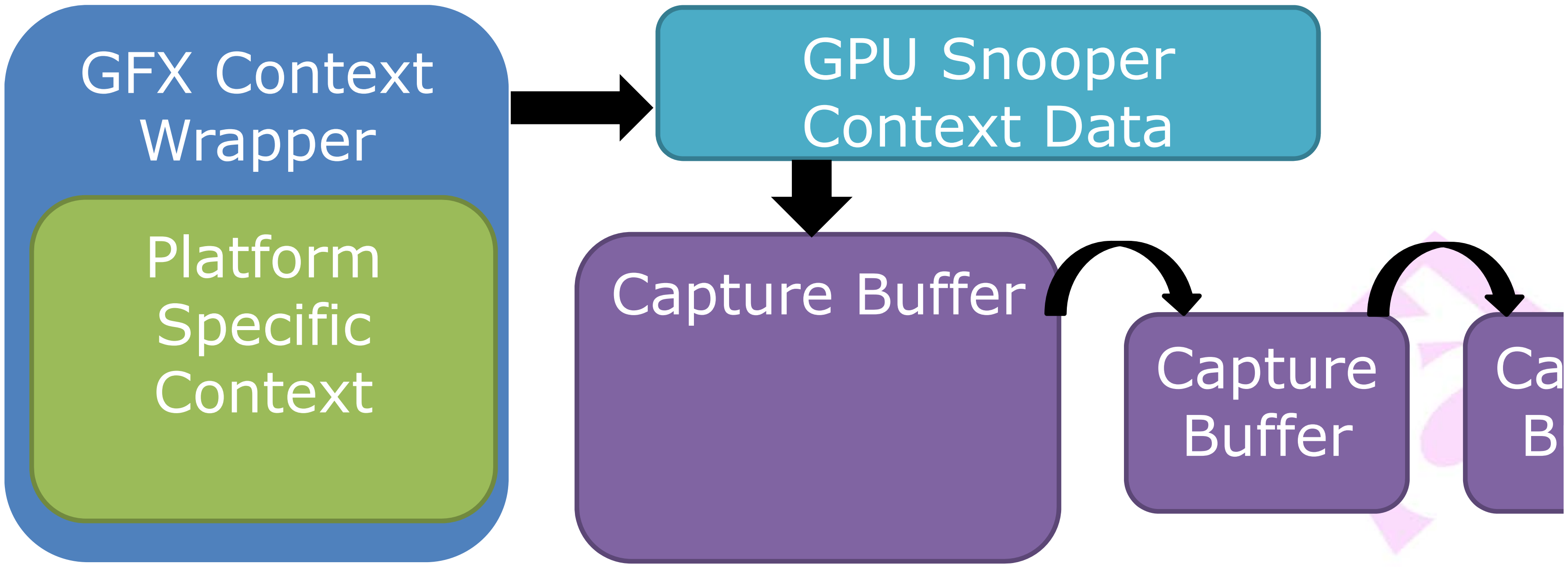
GPU Recording Implementation


GPU Performance Recording
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output

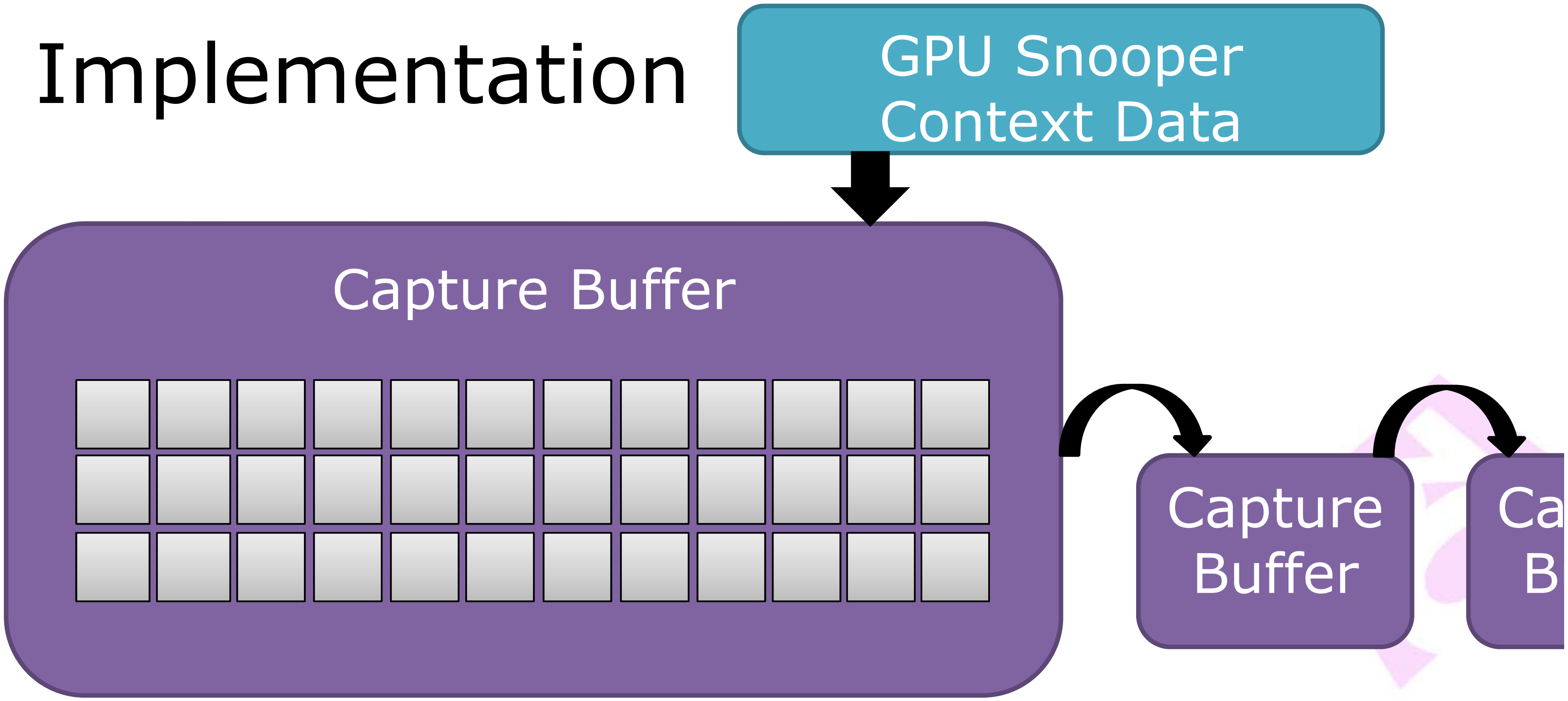
Implementation


GFX Context Wrapper
Platform Specific Context
PS4 - sce::Gnmx::GfxContext
DCB / CCB / Memory
XB1 - ID3D11DeviceContextX
ID3D11CommandList
(CreateDeferredContextX)

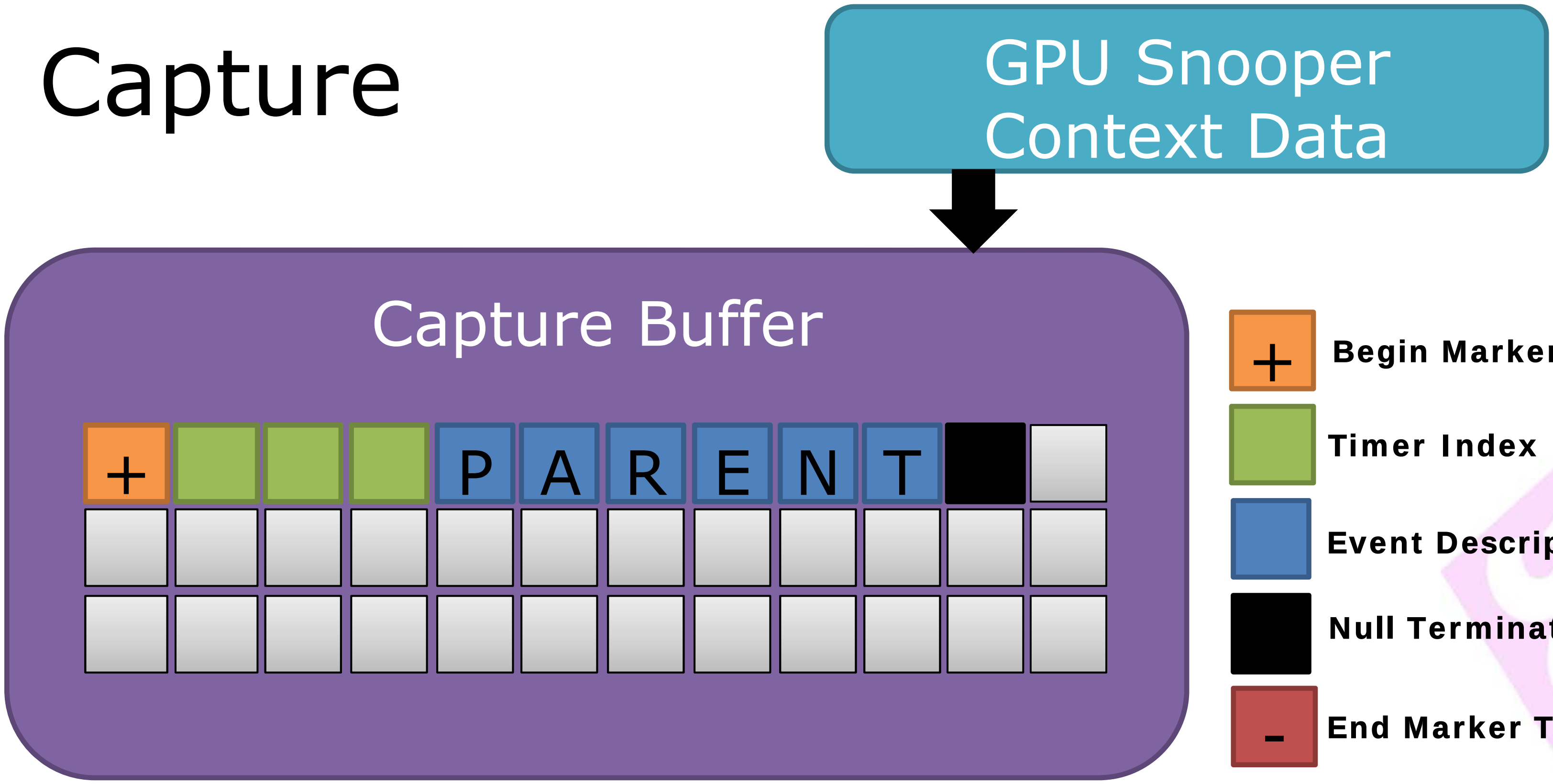
Implementation (Single Threaded)



Implementation

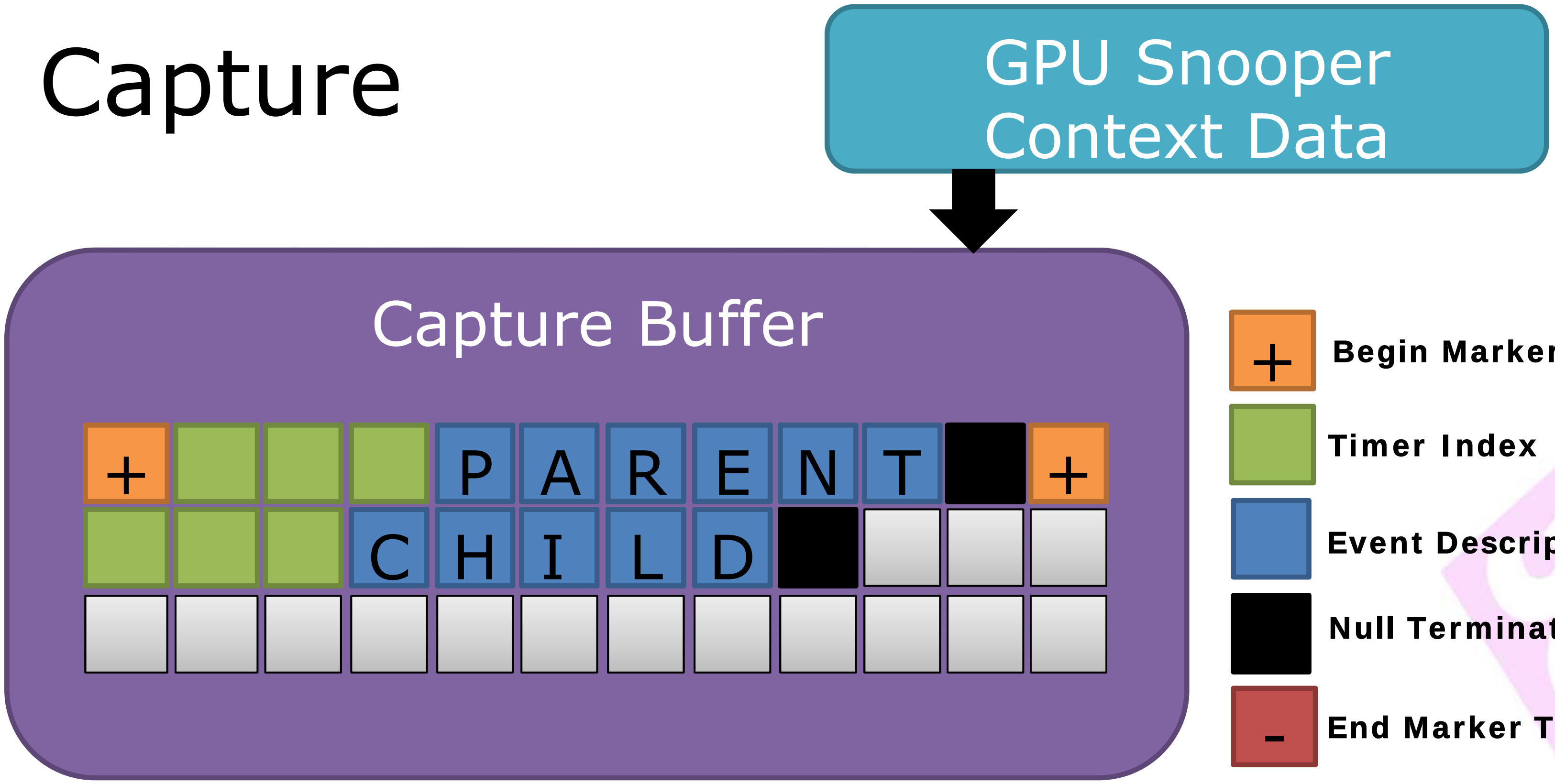


Capture



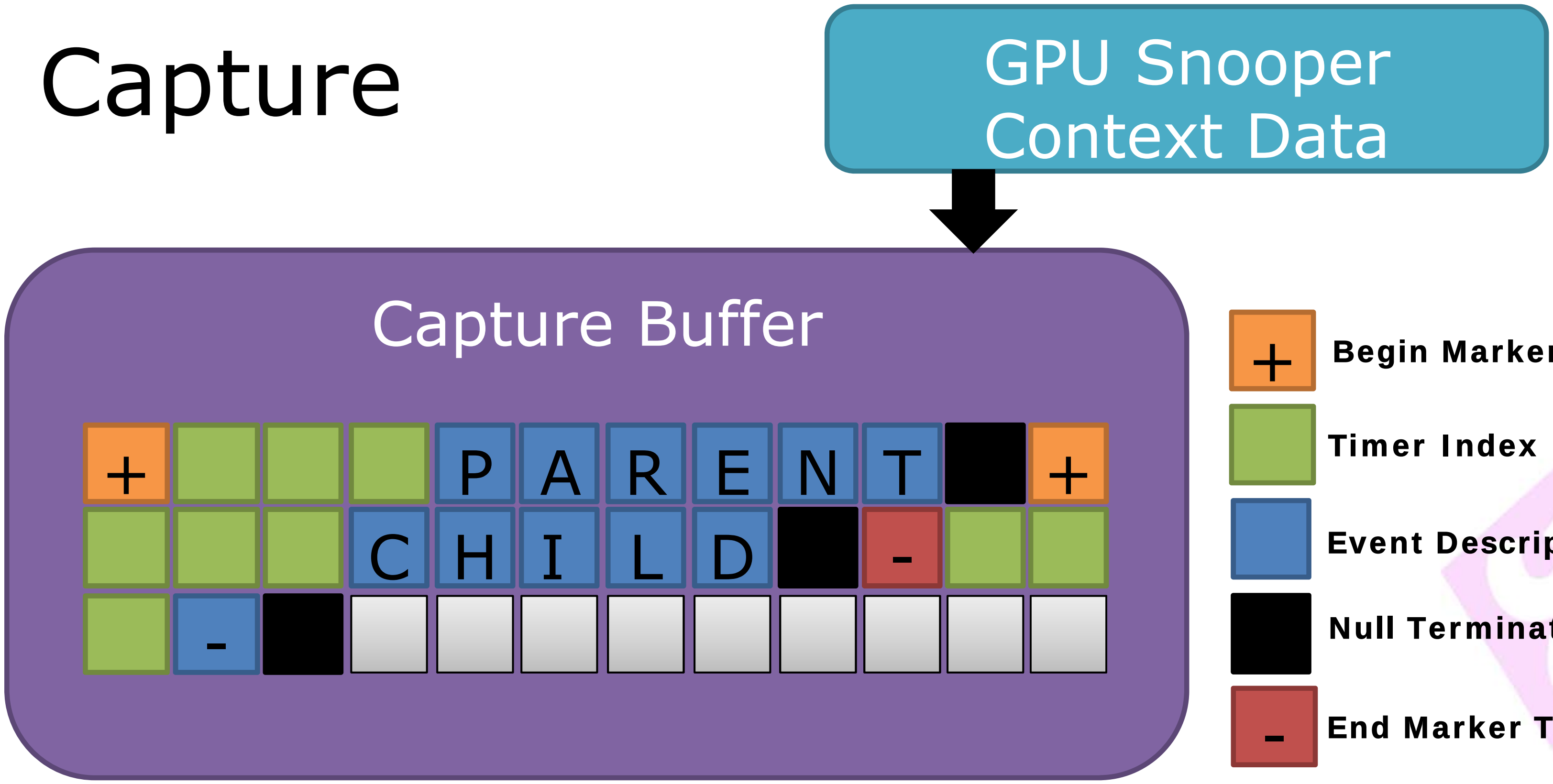
🟧 +	Begin Marker Tag
🟩	Timer Index
🟦	Event Description
⬛	Null Terminator
🟥 -	End Marker Tag

Capture



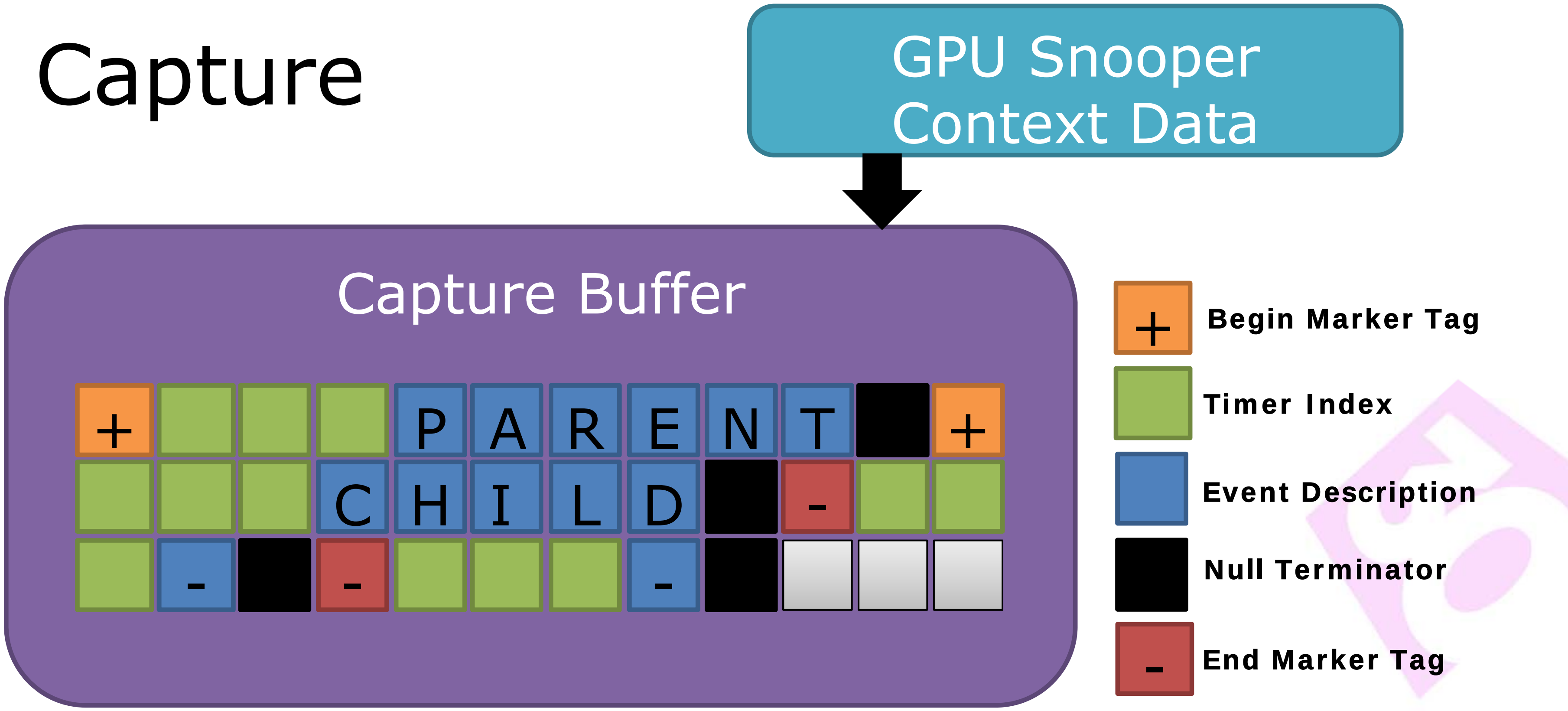
🟧 +	Begin Marker Tag
🟩	Timer Index
🟦	Event Description
⬛	Null Terminator
🟥 -	End Marker Tag

Capture



🟧 +	Begin Marker Tag
🟩	Timer Index
🟦	Event Description
⬛	Null Terminator
🟥 -	End Marker Tag

Capture


GPU Snooper Context Data
Capture Buffer
+
P
A
R
E
N
T
+
C
H
I
L
D
-
-
-
-
+
Begin Marker Tag
Timer Index
Event Description
Null Terminator
-
End Marker Tag

Implementation Details:
Capture Analysis

Analysis of the Data

- Parse Capture Buffer List for Events
- Read Associated Timer for Start / Stop
- Determine Hierarchy Level by Nesting
- Transfer the data to a PerfSnapshot struct for visualization rendering routines

Implementation Details: Multithreading

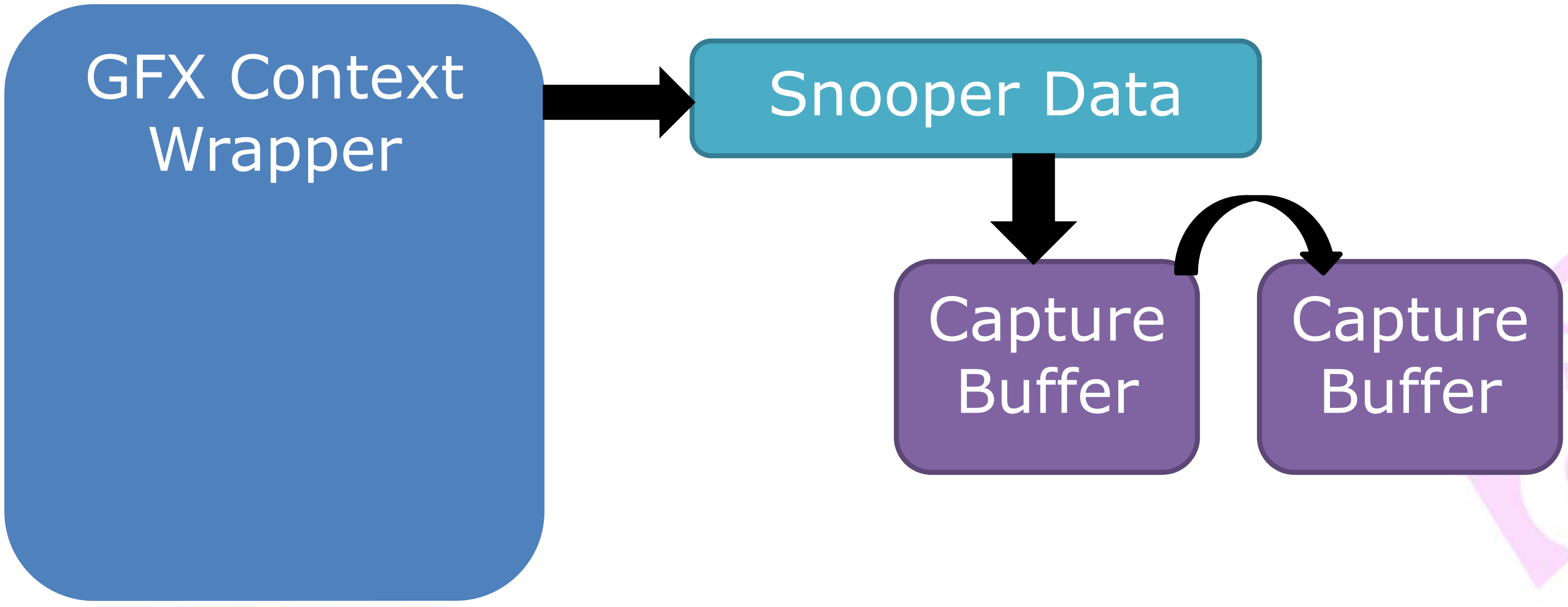
Multithreaded Implementation

- Parallel context structures
  - Per thread GFX contexts
  - Single threaded because each thread gets their own version
- Lock-Free Pool (SList) for Timers
- Lock Accessed list of Snooper Data Structs

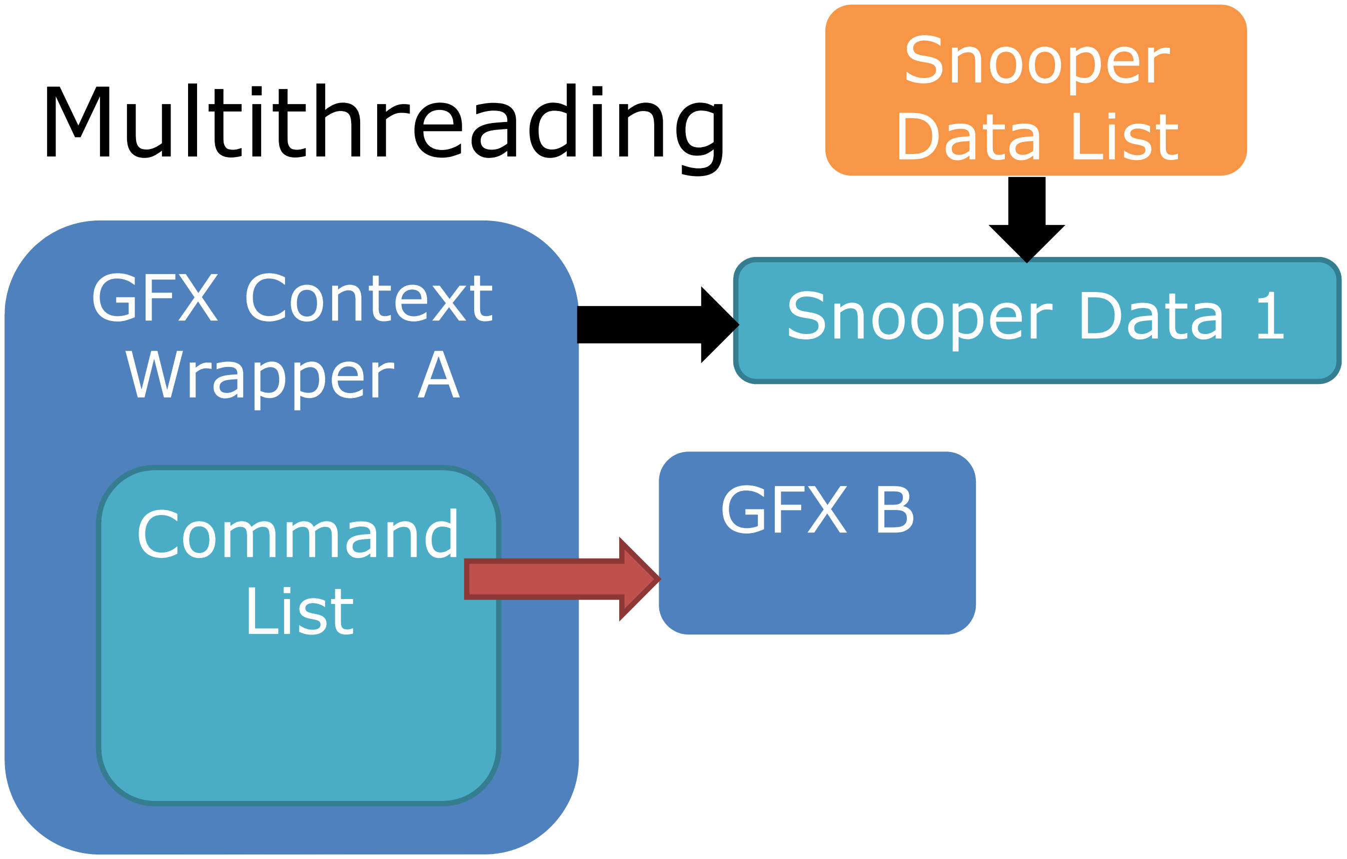
Multithreading

- Simplest version is just one main context calling deferred sub-contexts in a row and keeping a list of them.
- But how do you handle allowing any context to call a deferred sub-context at any time?

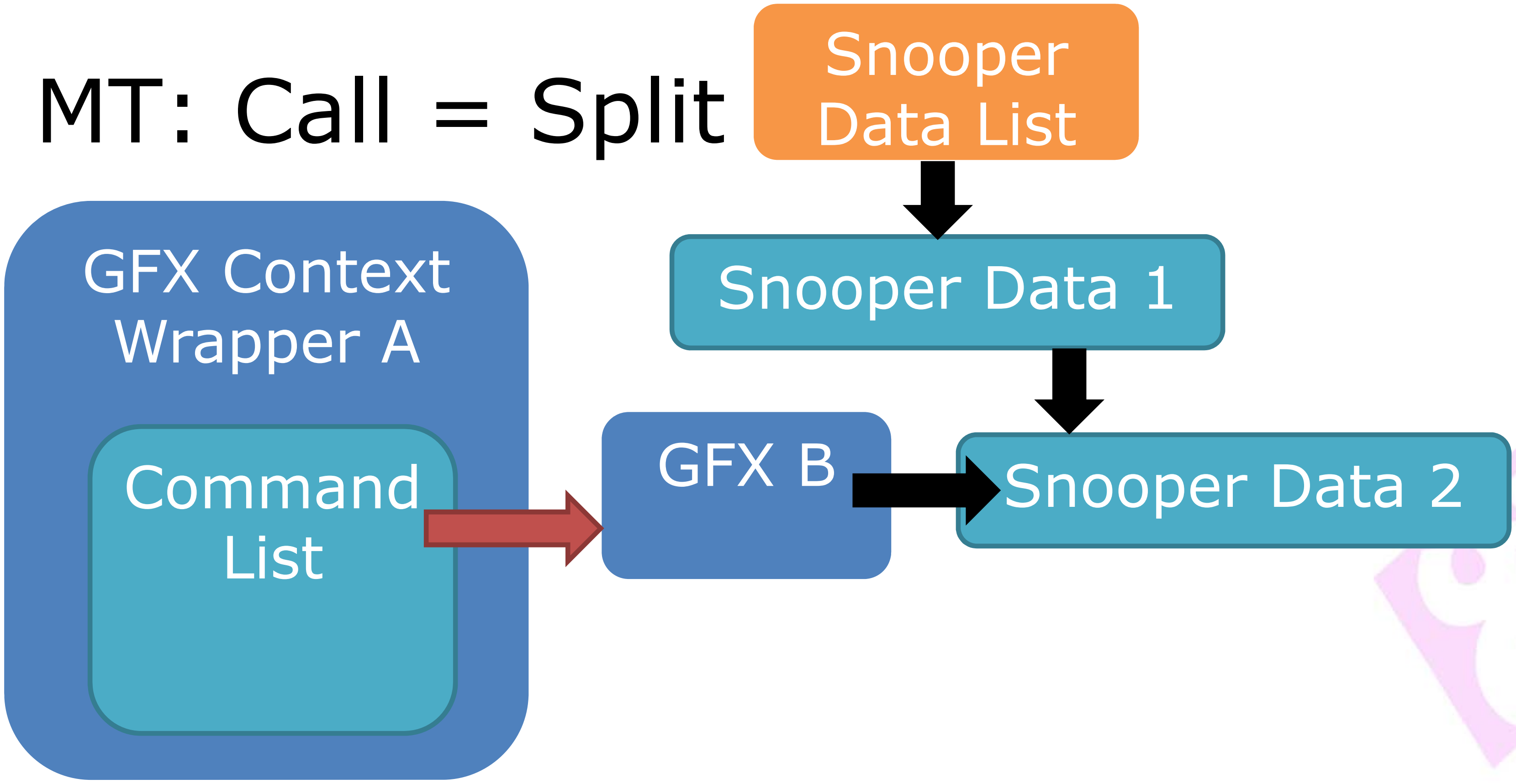
Multithreading GFX Contexts



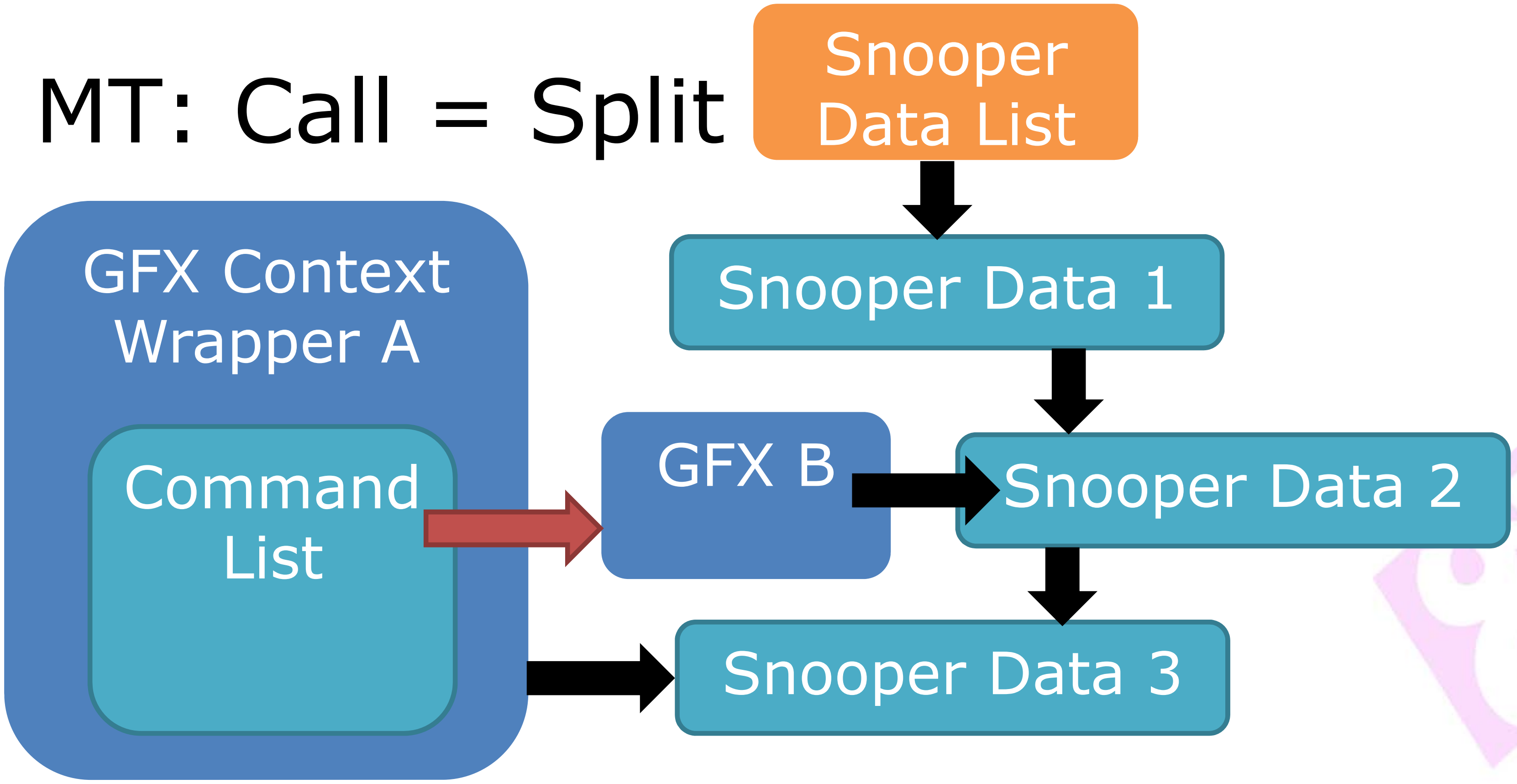
Multithreading



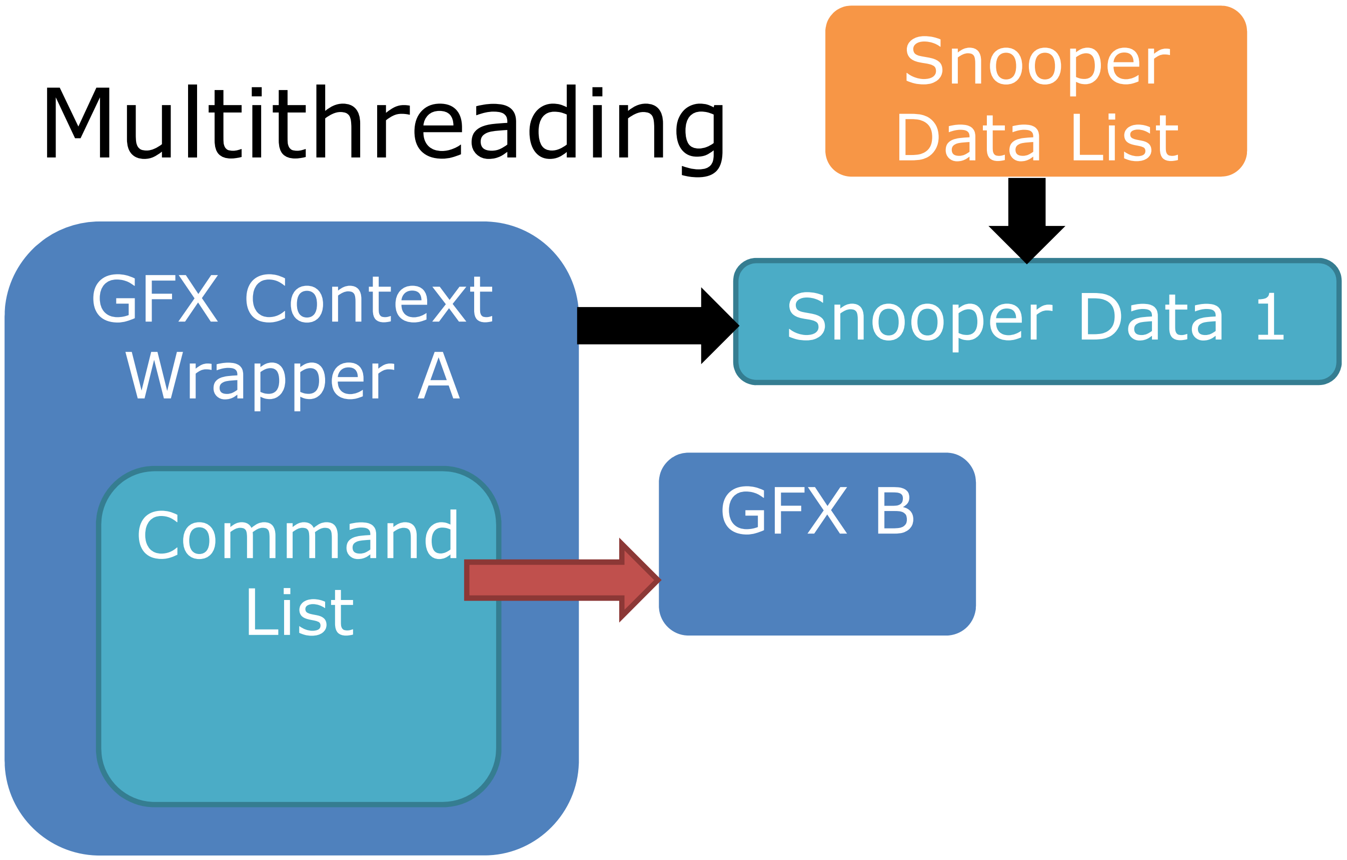
MT: Call = Split



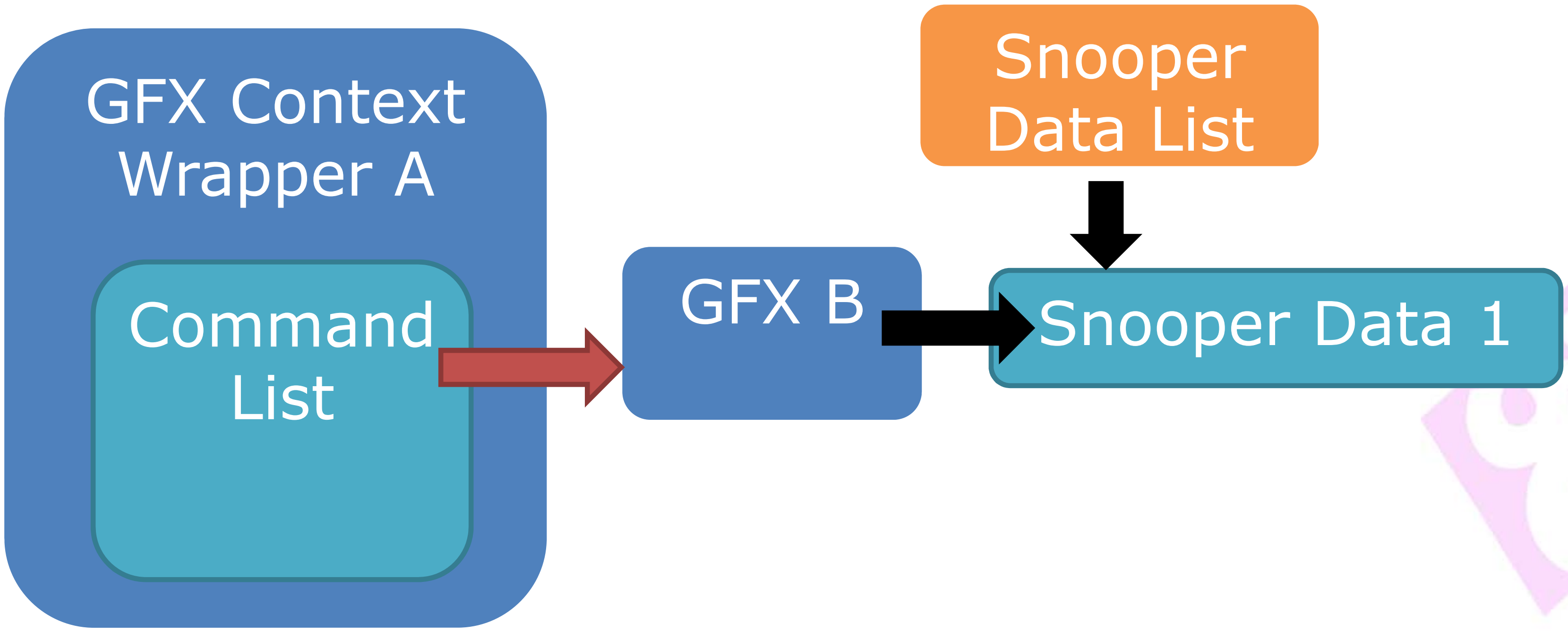
MT: Call = Split



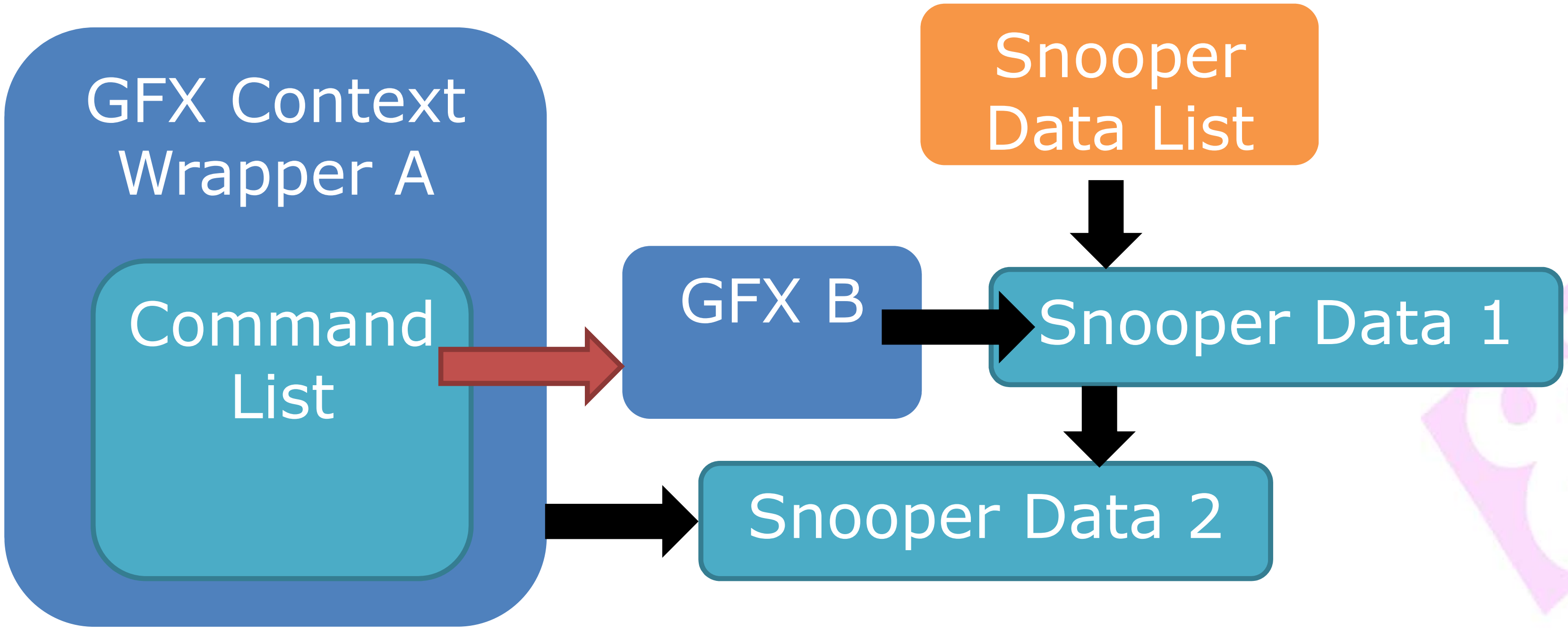
Multithreading



MT: Call = Transfer



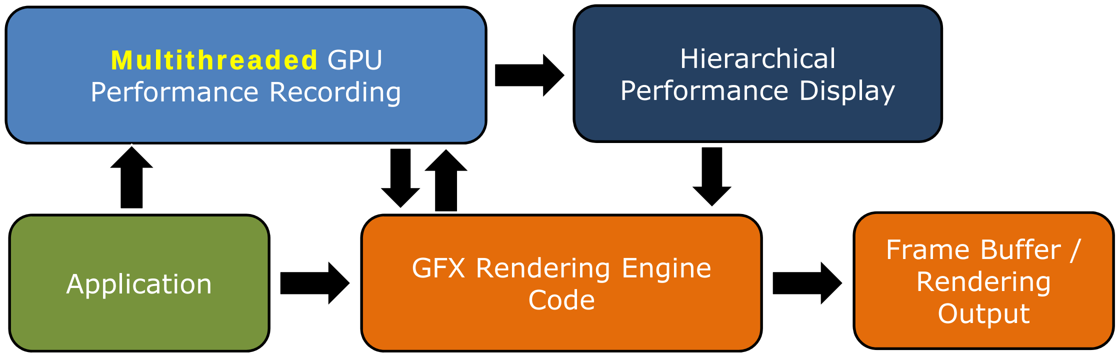
MT: Call = Transfer



Multithreading

- At frame submission time, we will have an list of GPU Snooper Data structures that we can process (in order)
- Analysis is exactly the same for MT as ST just potentially split across multiple "chunks"

NetherRealm Second-Pass GPU Profiler Display


Multithreaded GPU Performance Recording
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output

Implementation Details: Miscellaneous

Optimizations

- GPU performance impact very minimal
- Optimizations are primarily on the CPU side (performance and memory)
  - Single-threaded code when possible (per context)
  - Lock Free Pools for Multithreaded Allocations
  - Byte-Packed Capture Buffers
  - Timers addressed by Index rather than Pointer

Usability notes

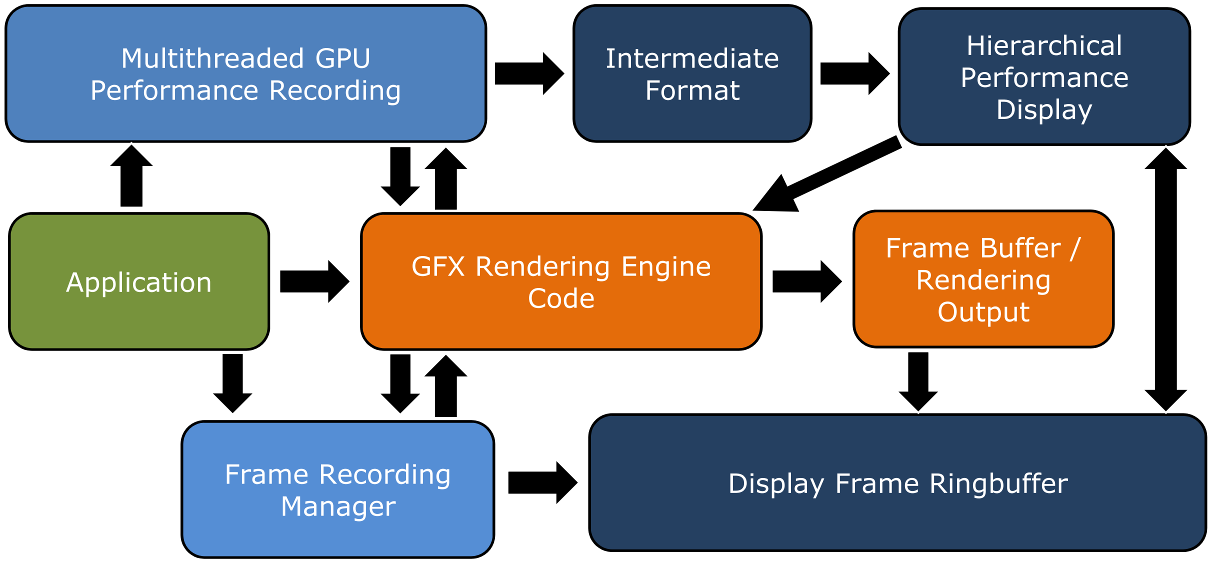
- Performed a couple days of tuning controls and GUI Visualizer
  - Getting zoom, movement, etc to feel natural
    - Example, original zoom was painful
    - Scale pan so it moves the same pixel speed regardless of zoom
  - Added “Key Chart”

Implementation Details: Continuous Frame Recording

Frame & Performance Recording

- Keep around more Intermediate Performance Records (i.e. larger N)
- Record Framebuffer Output
  - Simple Memory Ring Buffer of Scaled Output Frames
  - Can tie GPU Snooper capture to a frame
  - Allows freezing and scanning back & forth in time

NetherRealm Current GPU Profiler Architecture


Multithreaded GPU Performance Recording
Intermediate Format
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output
Frame Recording Manager
Display Frame Ringbuffer

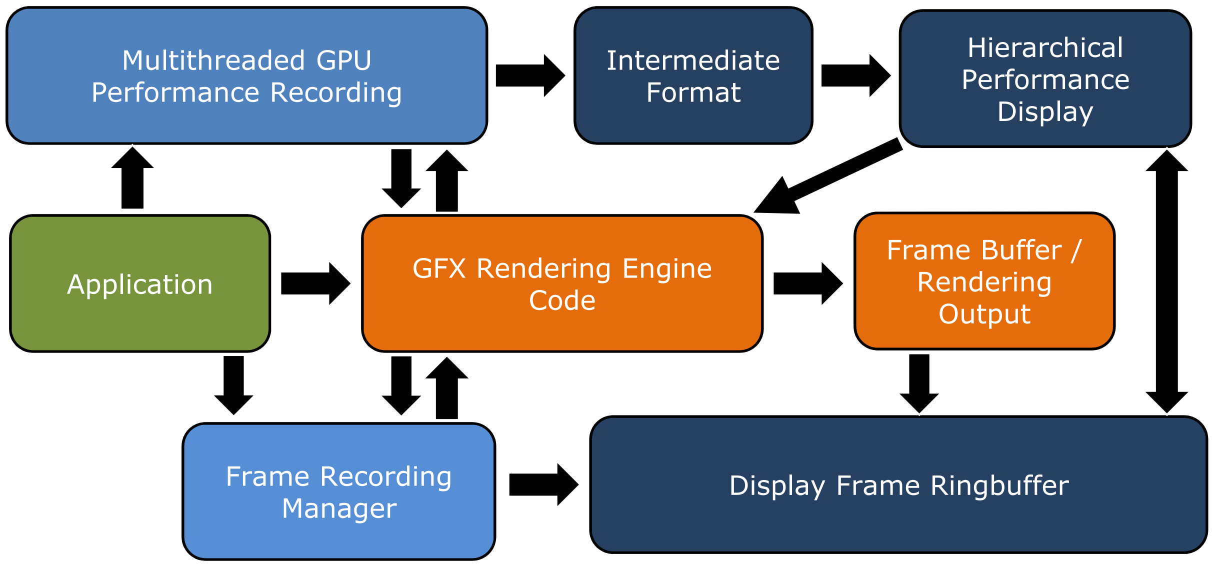
59.9 FPS
(16.7 ms)
NINJUTSU
SCORPION
90
R1
R1
CRYOMANCER
SUB-ZERO
GVR
60FPS
ETC: 8882
GTC: 2276
P4 Debug [ON]
RAZOR GPU [ON]
ENHANCED
E

GPU Snooper:
The Future ???

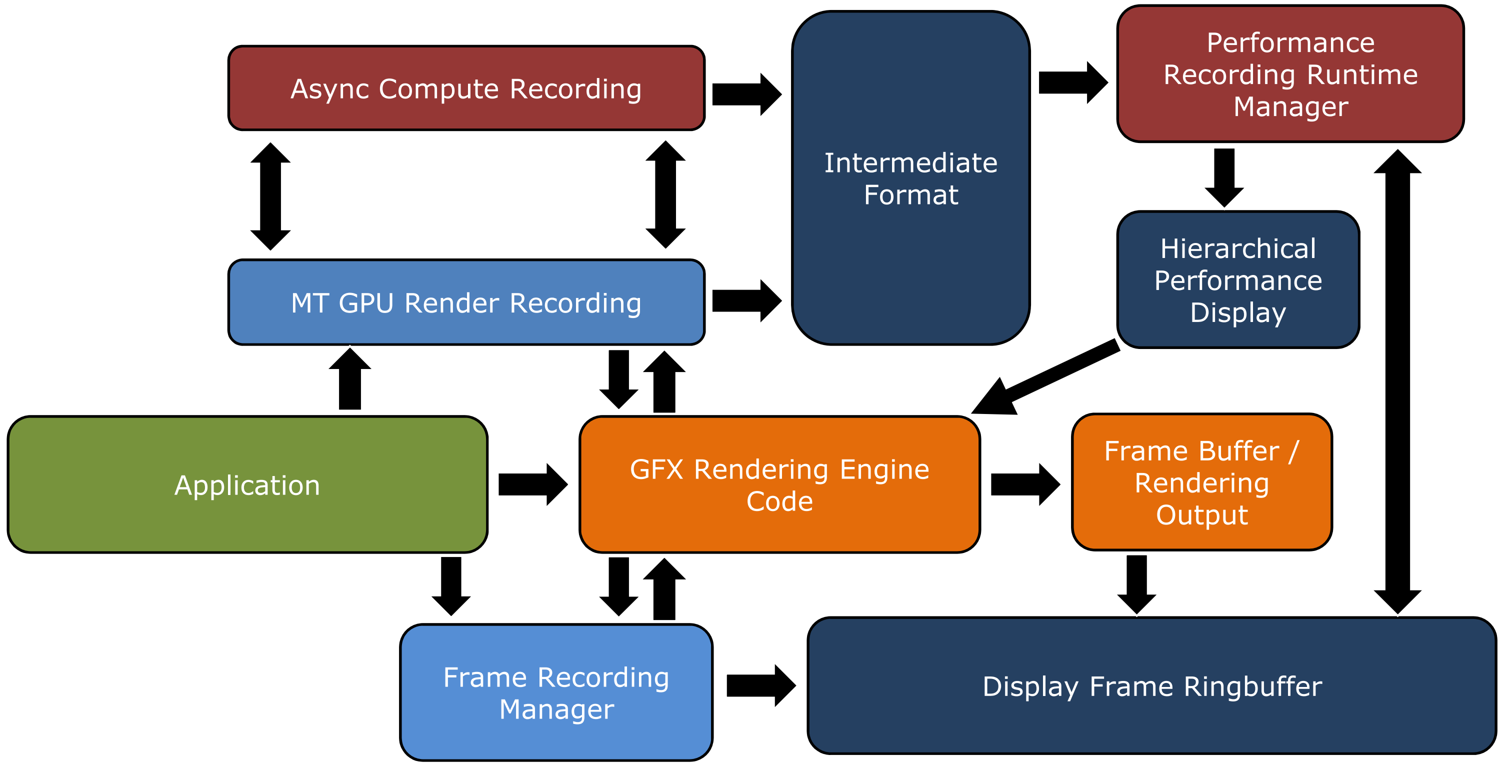
Where do we go from here

- Some future work already planned / started
- Additional ideas

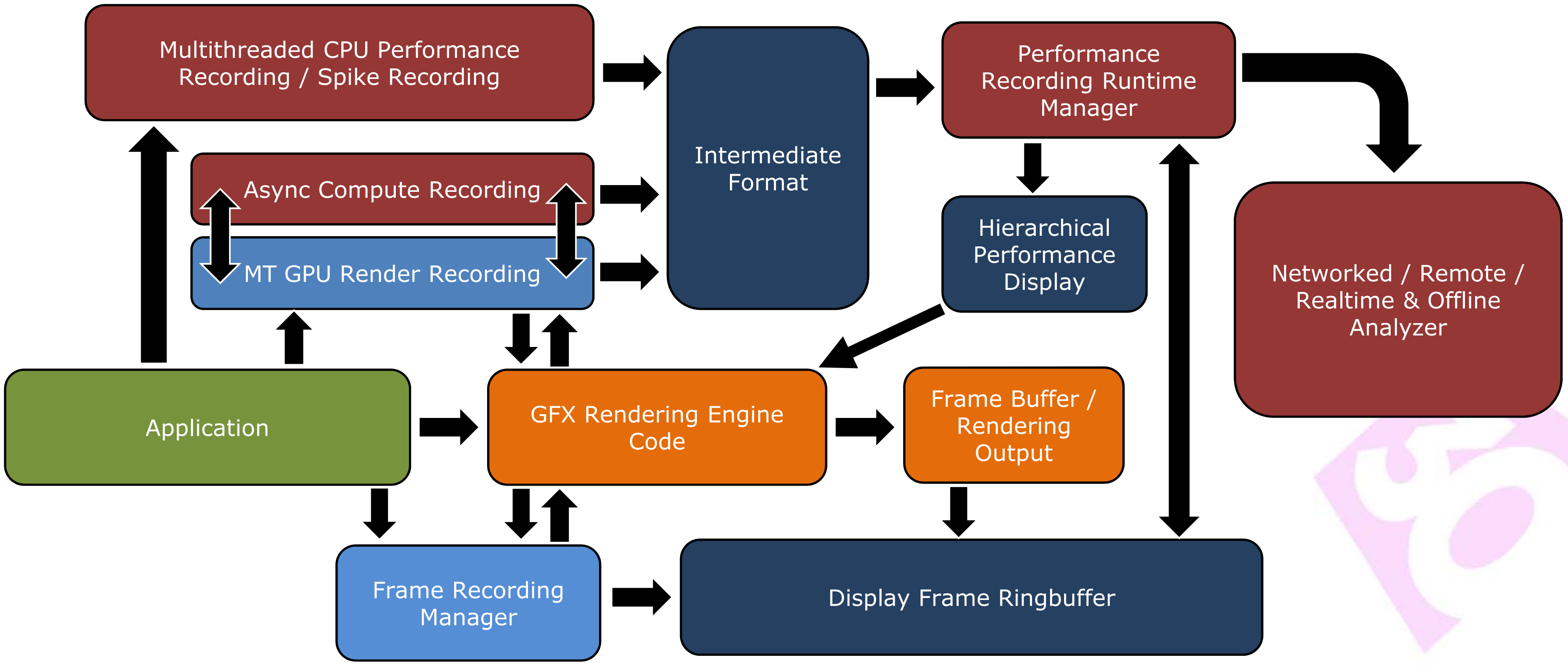
NetherRealm Current GPU Profiler Architecture


Multithreaded GPU Performance Recording
Intermediate Format
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output
Frame Recording Manager
Display Frame Ringbuffer

NetherRealm Profiler Architecture **Future Goals?**


Async Compute Recording
Intermediate Format
Performance Recording Runtime Manager
MT GPU Render Recording
Hierarchical Performance Display
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output
Frame Recording Manager
Display Frame Ringbuffer

NetherRealm Profiler Architecture **Future Goals?**


Multithreaded CPU Performance Recording / Spike Recording
Async Compute Recording
MT GPU Render Recording
Intermediate Format
Performance Recording Runtime Manager
Hierarchical Performance Display
Networked / Remote / Realtime & Offline Analyzer
Application
GFX Rendering Engine Code
Frame Buffer / Rendering Output
Frame Recording Manager
Display Frame Ringbuffer

Contact Info

Adisak Pochanayon

adisak@wbgames.com

Twitter: @adisak

Rollback Count = 0
Desync Count = 0
Remote Delay = -1
59.9 FPS
(16.7 ms)
CRYOMANCER
SUB-ZERO
Render Scene
12.03ms
DPC World
4.41ms
Opaque Tiered Rendering: ENoxMethod_Forward
3.85ms
BasePass
5.96ms
View0
5.90ms
Gfx wait on la
0.55ms
TIER_Unassigned
1.50ms
TIER_Unassigned
1.80ms
KickAsyncDrawListJobGraph
1.80ms
SF_FirFight_M_St
1.68ms
60FPS

QUESTIONS ???

- Click to edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level