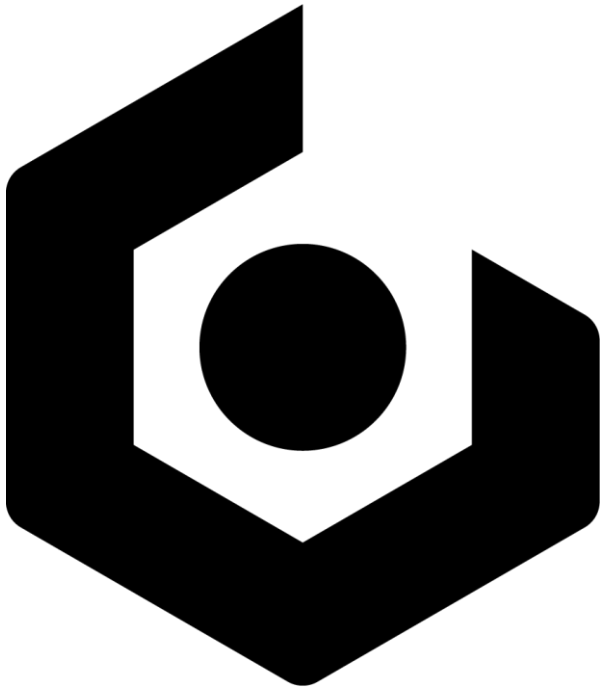
Scope – Hitman & G2

- Internal port
- Live updates - Elusive Targets
- Older branch of G2
- No content changes
- No geometry streaming system
- Aspire to have low maintenance cost

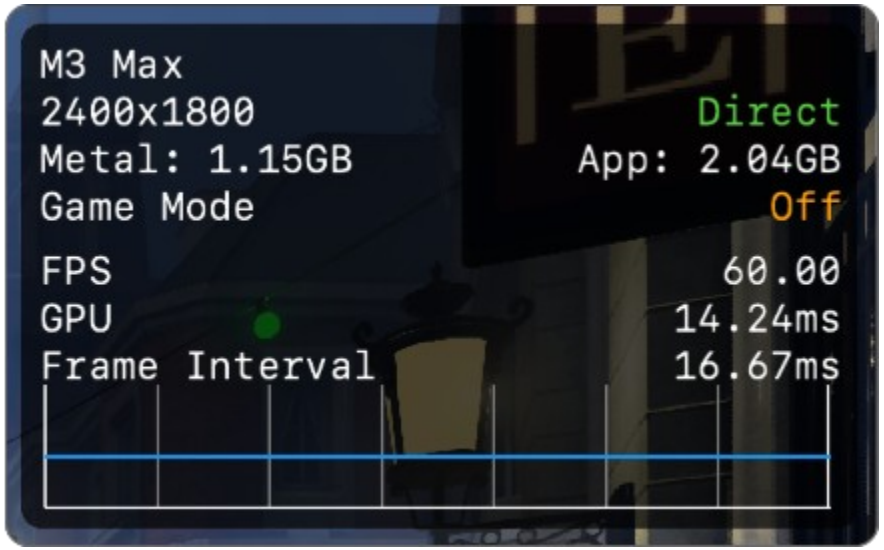


Mobile Platforms – Smartphones & Tablets

- System on Chip (SoC) design
- Unified Memory
- 3-6W Power Budget
- Peak performance is not sustainable
- RAM Access is energy intensive
  - 1W = 150-200MB @ 60fps

Groundwork

- Game Porting Toolkit
  - Trying out a Windows-DX12 build on macOS
- Try to find early failure points
  - Memory Budget
  - Unsupported Features

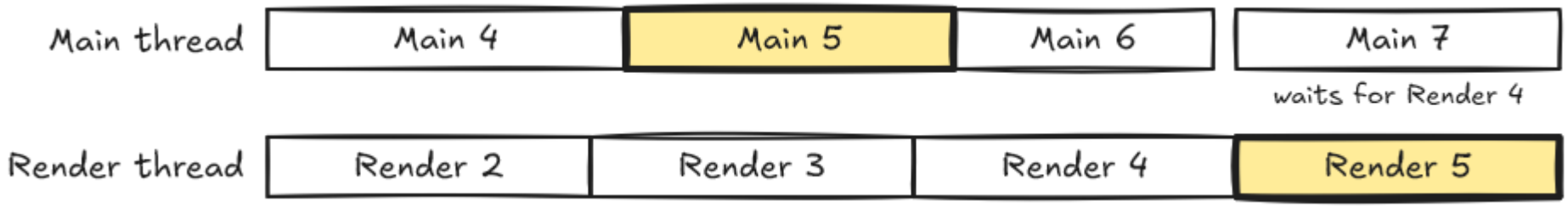


Target

- Min Spec
  - iPhone 15 Pro (A17Pro)
  - 8GB unified memory, ~6GB usable
- iOS 18.0 - “Metal3"
  - Argument Buffers - Tier 2
  - Residency Sets*
- Device Capabilities
  - iphone-performance-gaming-tier
- Entitlements:
  - increased-memory-limit
  - sustained-execution

Render Thread

- Main thread deals with gameplay logic and loading
- Render thread culls and records GPU commands
  - Reflect step copies necessary data from main thread to render thread
- Configurable maximum latency between main and render
  - Used for dealing with main thread spikes
- Not 1:1 with display frames
  - Render thread takes over during loading & sometimes doesn't present a frame



Whose Main Thread Is It Anyway?

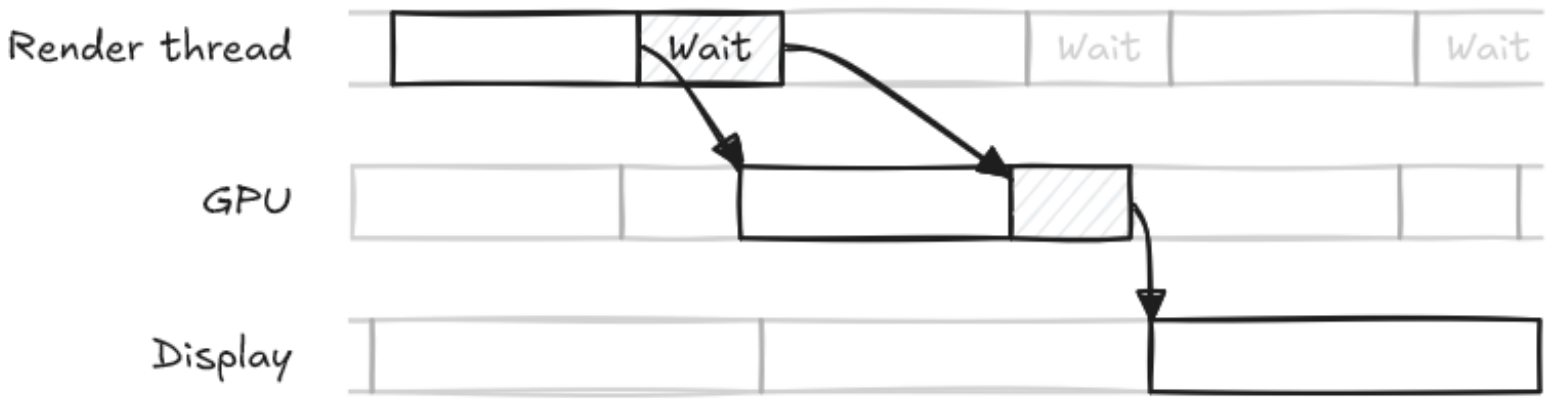
- Apple APIs are based on event driven model
- A lot of them can only run on main thread
  - Runs an event loop for processing I/O and callbacks
  - Avoid blocking as much as possible
- Our main and render threads have their own event loops
  - Needs to do blocking waits for other threads
- Game runs on its own “main” thread
  - Polls *CFRunLoop* to allow system main thread to schedule function
  - Schedules an update function on system main thread every frame

Acquiring a Drawable

- Metal term for back buffer
- Various options available
  - *nextDrawable*, *CADisplayLink*, *CAMetalDisplayLink*
- Threading setup makes things difficult
- Consistency problems
  - Experienced up to 2 ms variability
  - Problematic when GPU downclocks to fit frame in 33 ms

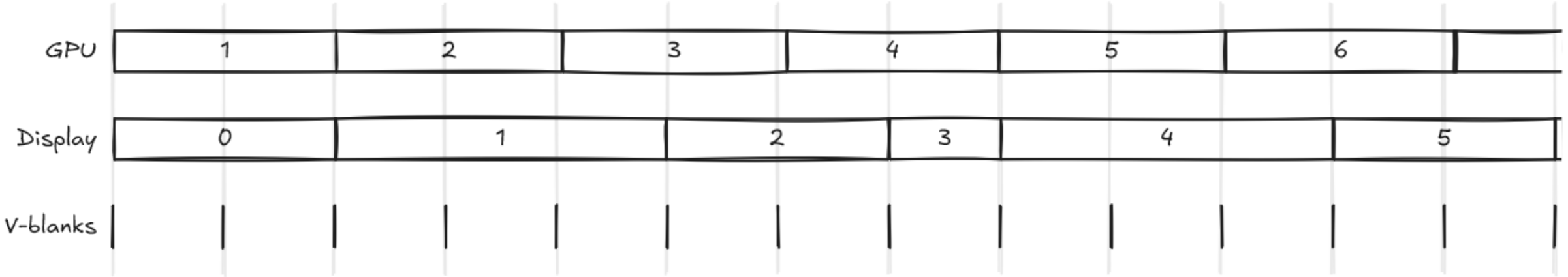
Acquiring a Drawable

- Call *`nextDrawable`* as late as possible
  - Early submit before waiting
- Gives the system a bit of wiggle room

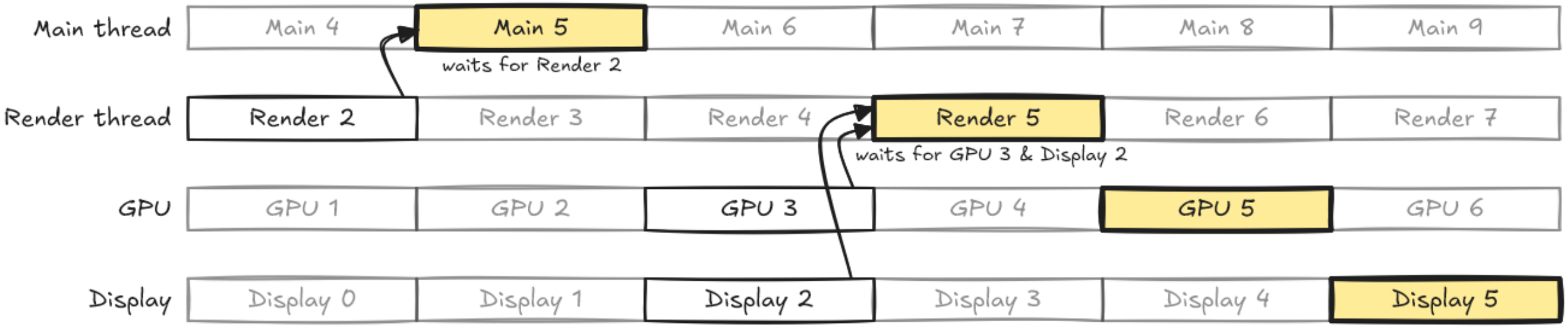


Frame Pacing

- Frames should display the same amount of time
  - Unevenness can cause average FPS to feel lower than it is
- Use *presentAfterMinimumDuration:*
  - Forces the drawable to be on screen for a minimum duration



The Full Picture

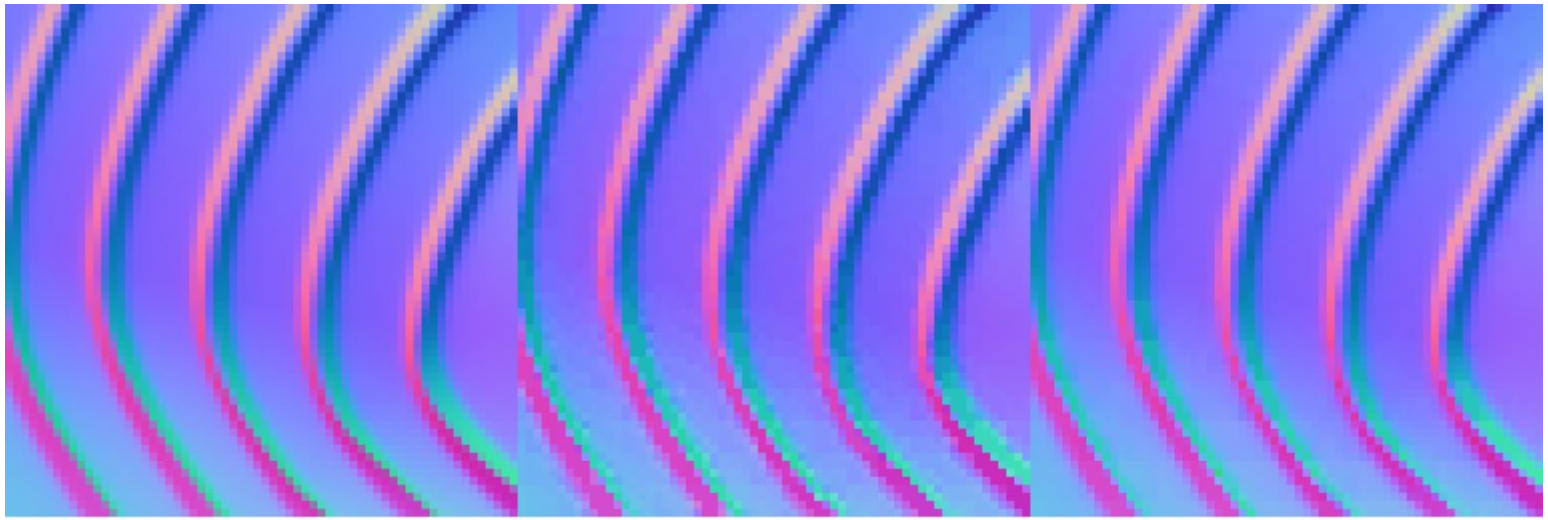


Resource Pipeline

- Dedicated resource server
- Spawns custom resource packers
- Runs on windows and serves data across the network
- Separate blob for retail builds

Texture Packing

- ASTC: more mobile friendly
- Block size heuristic for BCn->ASTC
- Apple don't expose GPU swizzling
- Can be very slow to compress



From: nVidia Developer Blog

Shader Compilation

- We author in HLSL
- Thin macro layer on top to express techniques
- Premade "shader nodes" that allow TAs to edit visually
- Small offline tool to generate cbuffer structs

```
CBUFFER_SHARED_BEGIN(cbColorCorrection, 5)
    FLOAT(fWeight0)
    FLOAT(fWeight1)
    INT(nGamma)
CBUFFER_SHARED_END(cbColorCorrection)

CBUFFER_SHARED_BEGIN(cbColorCorrectionResolve, 5)
    FLOAT(fNeutralLUT)
CBUFFER_SHARED_END(cbColorCorrectionResolve)
```

CPP

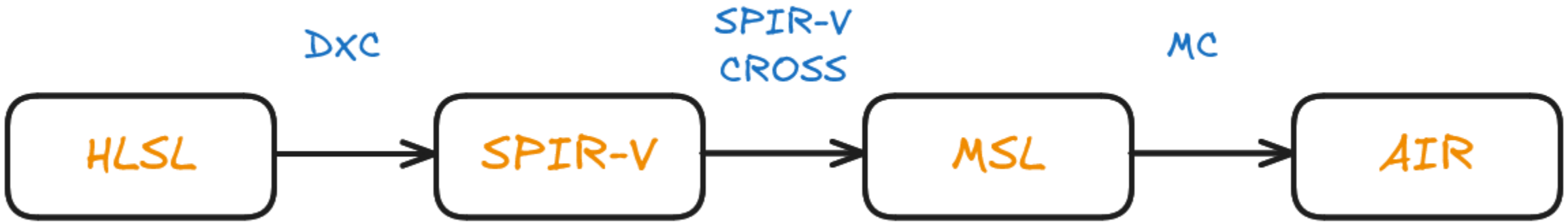
```
enum
{
    CBUFFER_CBCOLORCORRECTION_SLOT = 5,
    CBUFFER_CBCOLORCORRECTIONRESOLVE_SLOT = 5,
};

enum
{
    CBUFFER_CBCOLORCORRECTIONINSTANCE_COUNT = 4096,
    CBUFFER_CBCOLORCORRECTIONRESOLVEINSTANCE_COUNT = 4096,
};

struct S_cbColorCorrection
{
    float         fWeight0;
    float         fWeight1;
    int           nGamma;
    float           __pad3;
};

struct S_cbColorCorrectionResolve
{
    float         fNeutralLUT;
    float           __pad1;
    float           __pad2;
    float           __pad3;
};
```

HLSL

```
//*************************************************************************
//    cbColorCorrection
//*************************************************************************

struct S_cbColorCorrection
{
    float         fWeight0;
    float         fWeight1;
    int           nGamma;
};

CONSTANT_BUFFER(cbColorCorrection, 5, S_cbColorCorrection);

#define CBCOLORCORRECTION_FWEIGHT0_OFFSET 0
#define CBCOLORCORRECTION_FWEIGHT1_OFFSET 4
#define CBCOLORCORRECTION_NGAMMA_OFFSET 8

//*************************************************************************
//    cbColorCorrectionResolve
//*************************************************************************

struct S_cbColorCorrectionResolve
{
    float         fNeutralLUT;
};

CONSTANT_BUFFER(cbColorCorrectionResolve, 5, S_cbColorCorrectionResolve);

#define CBCOLORCORRECTIONRESOLVE_FNEUTRALLUT_OFFSET 0
```

Shader Compilation

- Classic approach: SPV-Cross

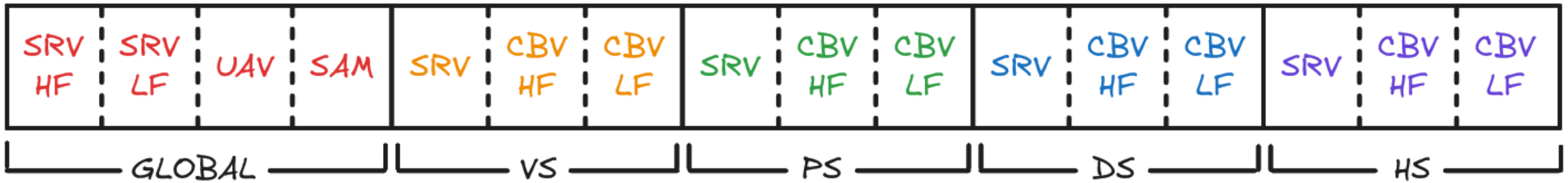


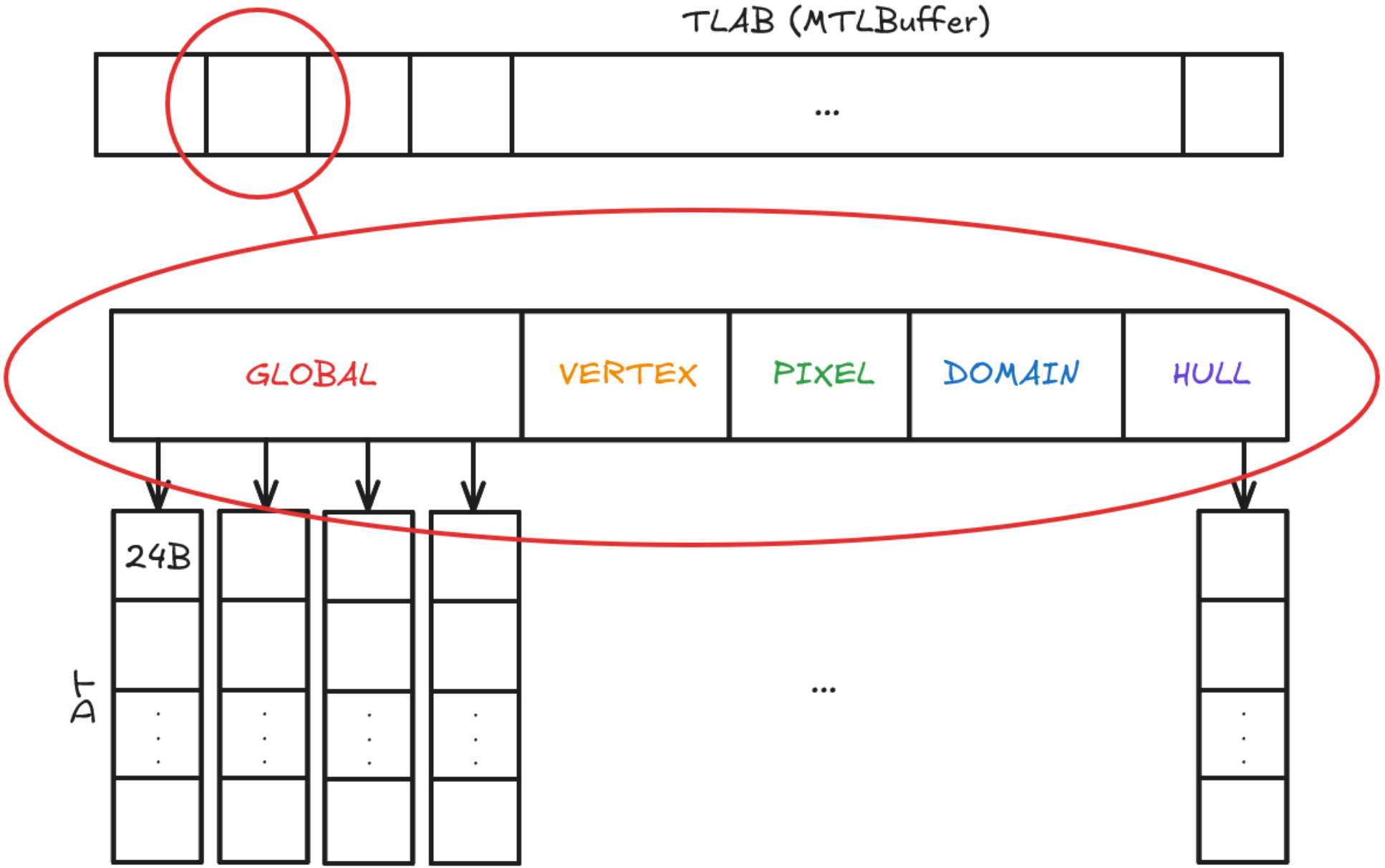
Shader Compilation

- New approach: Metal Shader Converter
  - Official support from Apple
  - Can always mix in with MSL shaders
  - Exposes FB-Fetch
  - Embeds shader source (HLSL)

Binding Model

- DX12 Root Signatures
- MSC + DXC Reflection
- 2 Level Argument Buffers
  - Linear Allocator per frame in flight



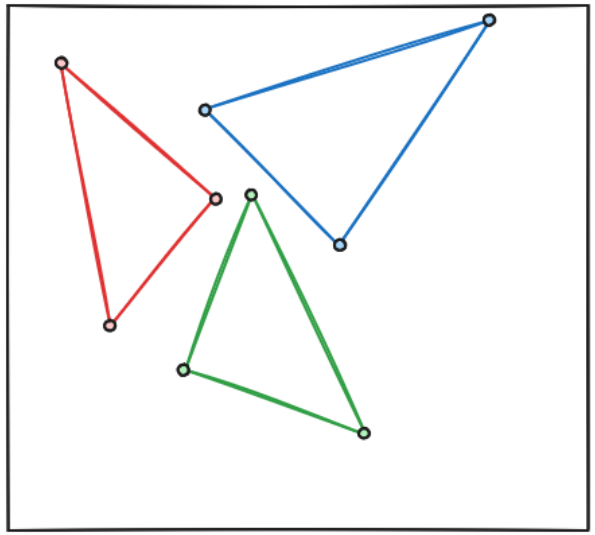

TLAB (MTLBuffer)
...
GLOBAL
VERTEX
PIXEL
DOMAIN
HULL
24B
DT
...

Resource Binding

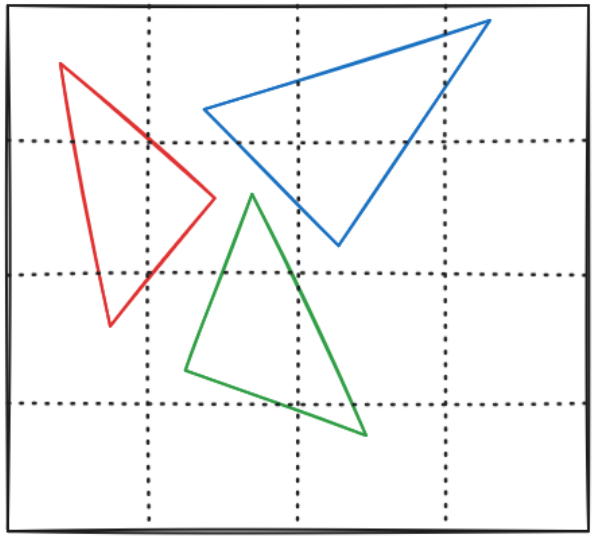
- Null Buffers & Dummy Textures are required
- OOB Reads & Writes are not discarded
- MSC forces texture array views (prior to version 3)
- MSC Runtime header adds redundant calls
- You can use *setBytes:* to provide data inline

TB;DR

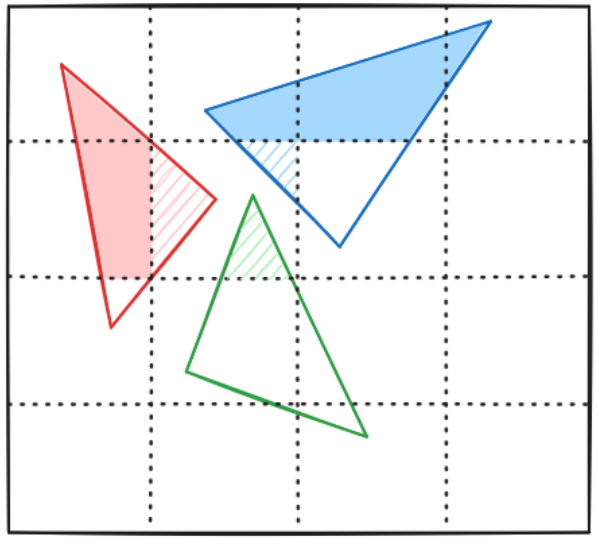
- Tile Based Deferred Rendering
  - Not related to Deferred Rendering
- Use fast local memory to significantly reduce bandwidth



Position-only vertex shading



Triangles are binned to tiles



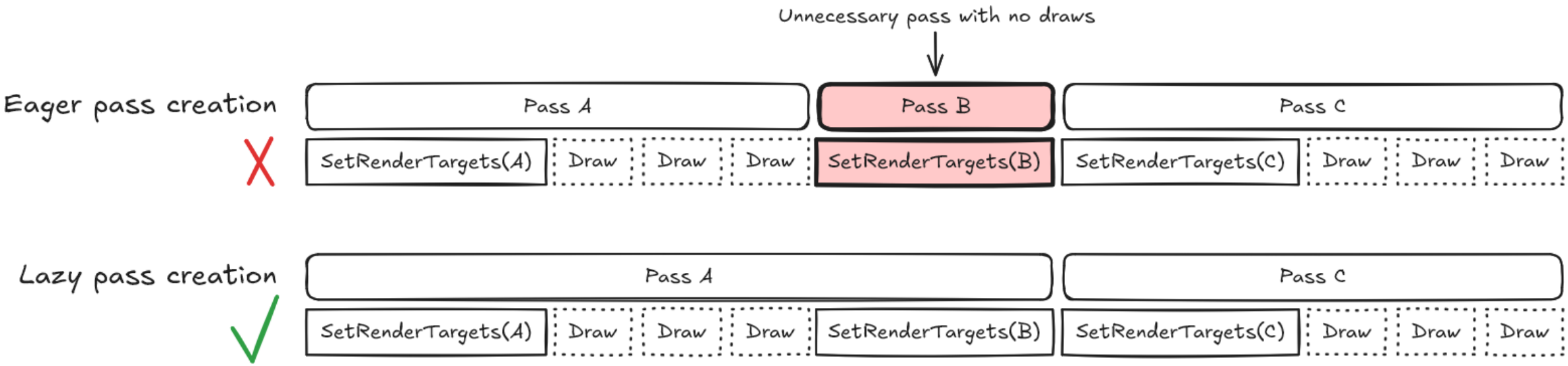
Full shading locally in tiles

Render Passes

- Needs to be created explicitly
- Frame graph ideal for this
  - Unfortunately not a thing in HITMAN
  - Too much work to refactor codebase
- Create on-the-fly instead

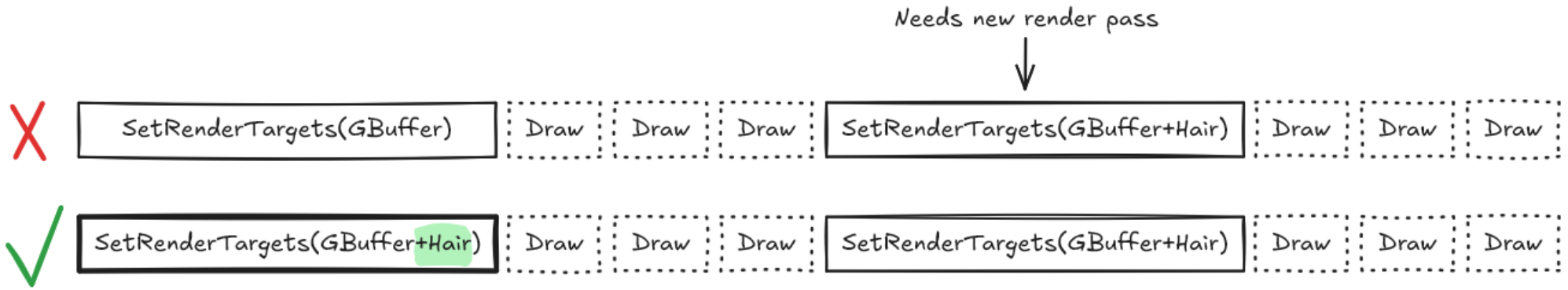
Render Passes

- Set dirty bit when render targets change
- If set when making a draw call, compare current & desired RTs



Render Pass Setup

- Minimal changes made to bind textures early
  - Mask out writes for RTs that aren't actually bound for the draw call
  - Allows more passes to be merged

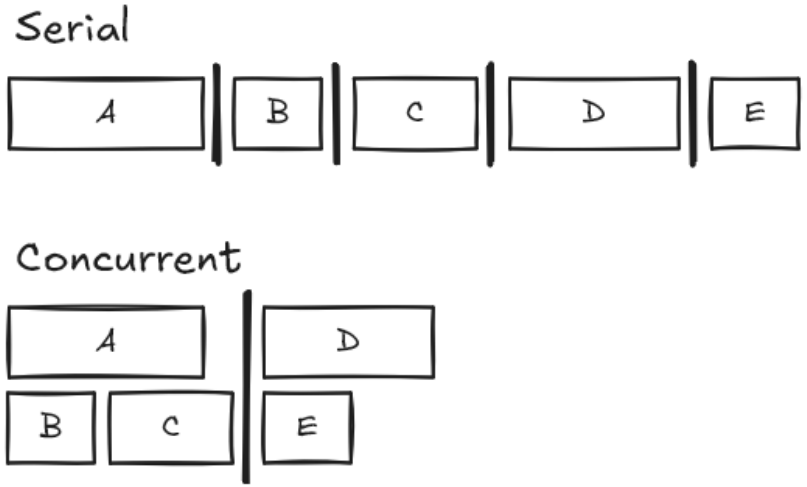


Render Pass Setup

- Naïve approach, but has worked out mostly well
  - Helps that the high-level rendering code is already well structured
- Our load/store actions are very conservative
  - Most stores end up being used though
  - Special handling for certain cases that didn't

Compute Pass Setup

- Metal exposes serial and concurrent compute passes
  - Serial puts a barrier between all dispatches in pass
  - Concurrent requires manual barriers
- We use concurrent only
  - Allows for overlapping dispatches
  - Our deferred shading dispatches are meant to run in parallel
  - Less driver overhead than 1 pass per dispatch



Compute Pass Setup

- Resources are tracked between dispatches
  - Fixed size buffer storing up to 32 resources needing a barrier
- Flush buffer on dispatch or if we hit the limit
  - *memoryBarrierWithResources:count:*
  - A bit conservative, but works well enough
  - Barrier only blocks within the same compute pass

Command Buffer Handling

- Command buffers in a ring buffer
  - Unwrapped index used as fence value
  - Used as synchronization primitive in engine for waiting on GPU
- Ring tracks currently submitted, free and completed index
  - Stored as unwrapped uint64 values
- Maximum of 4 active command buffers
  - Helps keep memory usage down

GPU Synchronization

- We rely on the automatic hazard tracking in Metal
  - Would like to do our own, but bigger fish to fry
  - It's a lot of work to move away from
  - Ended up good enough
- Tried implementing async compute
  - Didn't see performance gain
  - Metal already runs passes in parallel

Resource Allocators: Buffers

- Read-Only: suballocate from large *MTLBuffer*
- Split between per-frame and persistent
- Writeable: Individual allocation to avoid over-synchronization
- All buffers are shared resources
- Use *setBufferOffset:* methods as much as possible

Resource Allocators: Textures

- Read-Only: *MTLHeap*
  - Balance number of heaps: num residency calls vs upfront allocation size
- Read-Only: Individual allocation + Global *MTLResidencySet*
  - Single make resident call per command buffer / queue
- Writeable: Individual allocation
  - Pool resources
  - Make resident with *useResource:stages:*
    - Make sure to call the stages variant, allows for potential VS-PS overlap
- Upload data via Ring Buffer

Resource Allocators: Textures

Lossless Compression:

- *MTLTextureUsagePixelFormatView*
- Linear textures

Buffer textures:

- No depth formats
- No texture arrays
- No mips

Upscaling

- MetalFX Spatial
  - Very easy to integrate
  - Over sharpens the image
- MetalFX Temporal
  - Integrates well with our SSAA pipeline setup
  - Better image quality, but can look soft at lower resolutions
- Native Resolution UI
  - We draw our UI as part of the spatial upscaling pass
  - No extra bandwidth cost, since it’s needed anyway
  - Allows for very crisp UI on top of decoupled render resolution


Show Menu


Show Menu


Show Menu

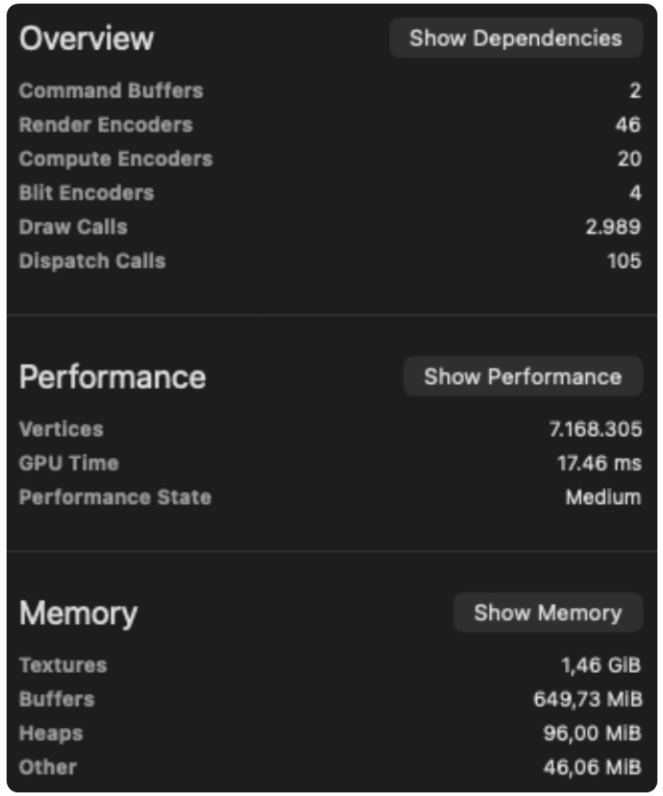

Show Menu

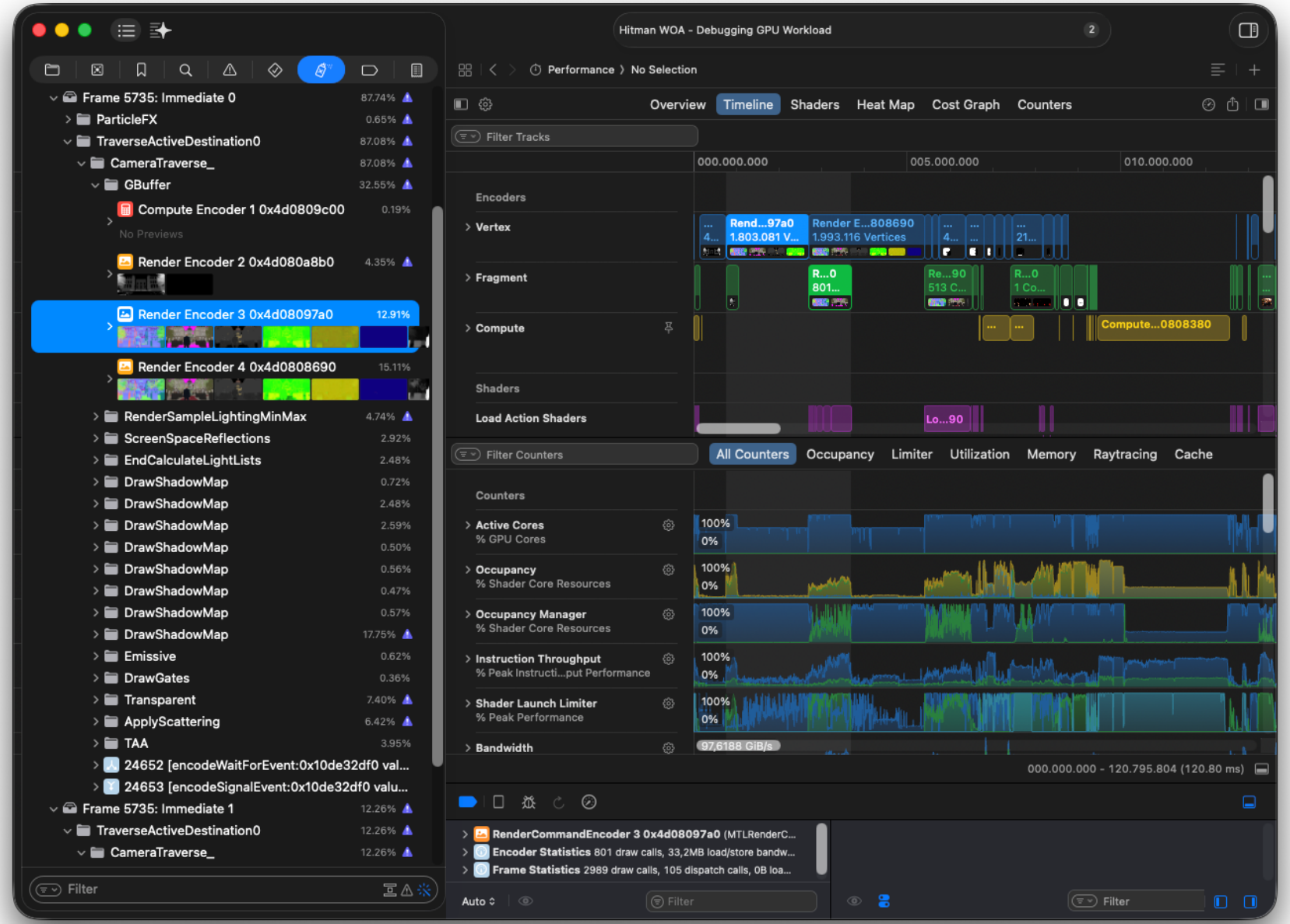
PSO Caching: Shader Prewarming

- Gather PSO data during gameplay
- Create helper state objects at engine startup
- Create PSOs at level load time
- Requires QA pass to gather all data

```
union
{
    struct
    {
        uint64 vertexShader : RENDER_SHADER_BITS;                  // 14
        uint64 pixelShader : RENDER_SHADER_BITS;                   // 14
        uint64 inputLayout : RENDER_INPUT_LAYOUT_BITS;             // 8
        uint64 rasterizerState : RENDER_RASTERIZER_STATE_BITS;     // 7
        uint64 blendState : RENDER_BLEND_STATE_BITS;               // 8
        uint64 depthStencilState : RENDER_DEPTHSTENCIL_STATE_BITS; // 8
        uint64 topologyType : RENDER_TOPOLOGY_TYPE_BITS;           // 4
        uint64 _pad0 : 1;                                          // 1
        uint64 renderTargetFormat : RENDER_TARGET_FORMAT_BITS;     // 7
        uint64 domainShader : RENDER_SHADER_BITS;                  // 14
        uint64 hullShader : RENDER_SHADER_BITS;                    // 14
        uint64 computeShader: RENDER_SHADER_BITS;                  // 14
        uint64 _pad1 : 128 - 113;
    };

    struct
    {
        uint64 hash0;
        uint64 hash1;
    };
};
```

What Does It Look Like?


Overview
Show Dependencies
Command Buffers 2
Render Encoders 46
Compute Encoders 20
Blit Encoders 4
Draw Calls 2.989
Dispatch Calls 105
Performance
Show Performance
Vertices 7.168.305
GPU Time 17.46 ms
Performance State Medium
Memory
Show Memory
Textures 1,46 GiB
Buffers 649,73 MiB
Heaps 96,00 MiB
Other 46,06 MiB


Hitman WOA - Debugging GPU Workload
Performance 〉 No Selection
Overview
Timeline
Shaders
Heat Map
Cost Graph
Counters
Filter Tracks
Encoders
Vertex
Fragment
Compute
Shaders
Load Action Shaders
Filter Counters
All Counters
Occupancy
Limiter
Utilization
Memory
Raytracing
Cache
Counters
Active Cores
% GPU Cores
Occupancy
% Shader Core Resources
Occupancy Manager
% Shader Core Resources
Instruction Throughput
% Peak Instructi...put Performance
Shader Launch Limiter
% Peak Performance
Bandwidth
Frame 5735: Immediate 0
ParticleFX
TraverseActiveDestination0
CameraTraverse_
GBuffer
Compute Encoder 1 0x4d0809c00
Render Encoder 2 0x4d080a8b0
Render Encoder 3 0x4d08097a0
Render Encoder 4 0x4d0808690
RenderSampleLightingMinMax
ScreenSpaceReflections
EndCalculateLightLists
DrawShadowMap
Emissive
DrawGates
Transparent
ApplyScattering
TAA
Frame 5735: Immediate 1
RenderCommandEncoder 3 0x4d08097a0
Encoder Statistics 801 draw calls, 33,2MB load/store bandw...
Frame Statistics 2989 draw calls, 105 dispatch calls, 0B loa...
Filter

Rapid Fire

- Aggressive downclocking
  - Hard to see performance improvements - device might decide to run at lower frequency
  - Makes dynamic resolution scaling hard to implement
- CPU-GPU Load balancing - GPU saves can help CPU
  - Also makes performance improvements hard to see directly
- Limit number of command buffers per frame
  - Dispatching each has a big overhead

Rapid Fire

- Use Instruments.app
- Metal Frame Capture
  - Disables GPU downclock
- Do not spin lock
  - Consumes battery
  - OS gives the thread more priority
- Do not sleep
  - OS deprioritizes the thread, can result in ~10ms wake up delay
  - Rely on wake ups from other threads instead

Rapid Fire

- Clearing memory can be expensive
- Prefer direct release over autorelease for Objective-C objects
- Follow Cocoa memory management policy
- System Clock Time
  - CLOCK_MONOTONIC queries time of day to increment while system sleeps
    - Fairly costly when used a lot
  - CLOCK_UPTIME_RAW does not

Rapid Fire

- Debug Groups / Signposts
  - *MTLCommandBuffer* and *MTLCommandEncoder* has *push/popDebugGroup*
  - But we don’t know where our passes will begin and end in the future
  - Ended up just putting first two levels of debug groups on the command buffer
- Thin LTO / Compilation Time Comparison
  - Usually HITMAN compiles with unity builds, but we don’t have that working for Darwin toolchain
  - Thin LTO didn’t add too much extra time, and made a huge performance difference
  - Just add *-flto=thin*

Conclusions

- Shipped running at 30 FPS
  - iPhone 15 Pro and newer
- Overall fun project to work on
  - A healthy dose of pain
  - Many interesting technical challenges
  - Learned a ton of things doing it ourselves

Thank you!


Q&A