

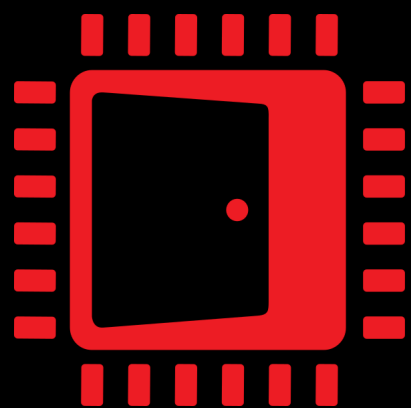
AMD  
EPYC

AMD  
RYZEN

AMD  
RADEON

# THE RENDER PIPELINE SHADERS SDK

ZHUO CHEN, AMD

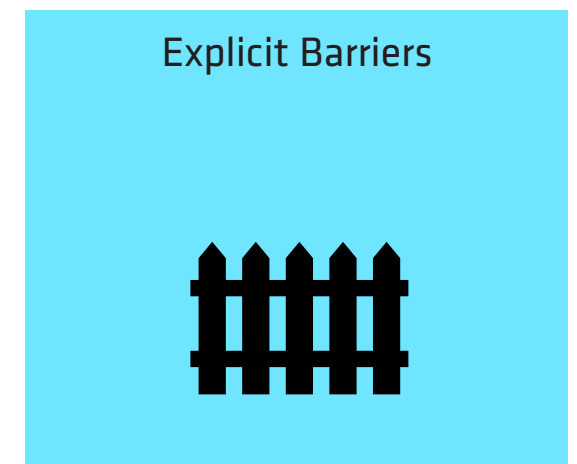
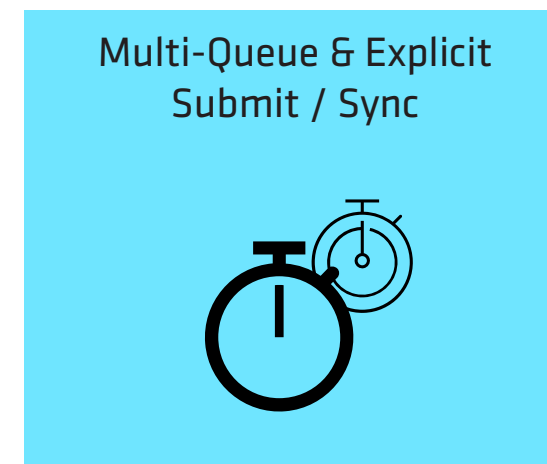
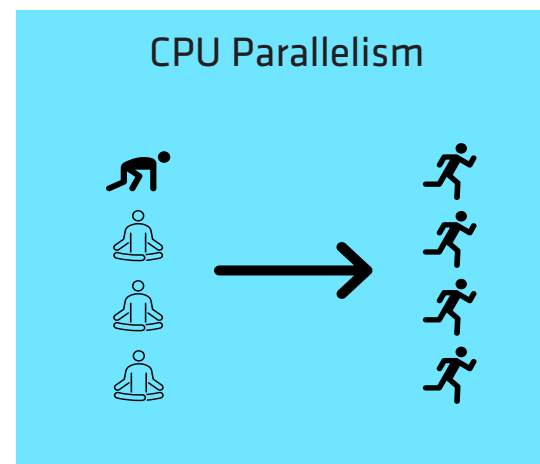


AMD  
GPUOpen

AMD  
together we advance\_

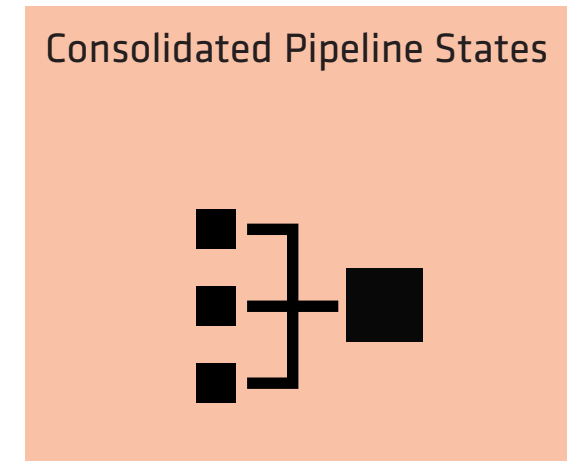
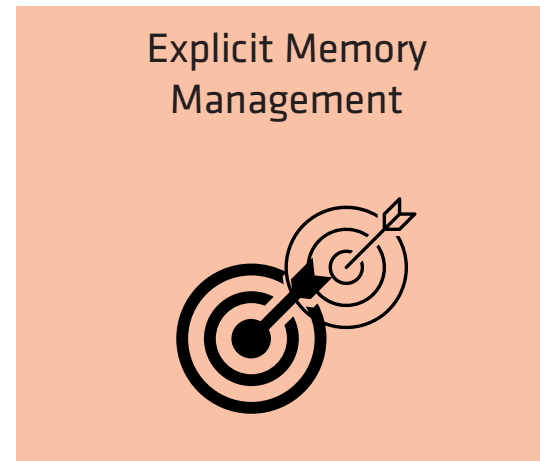
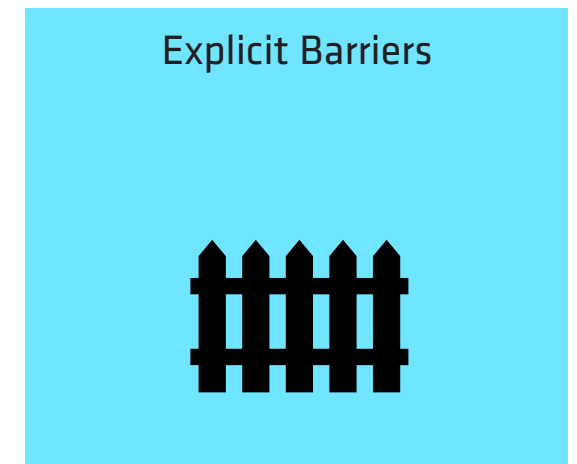
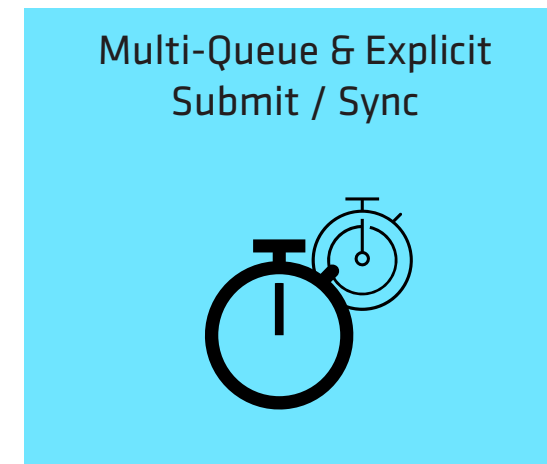
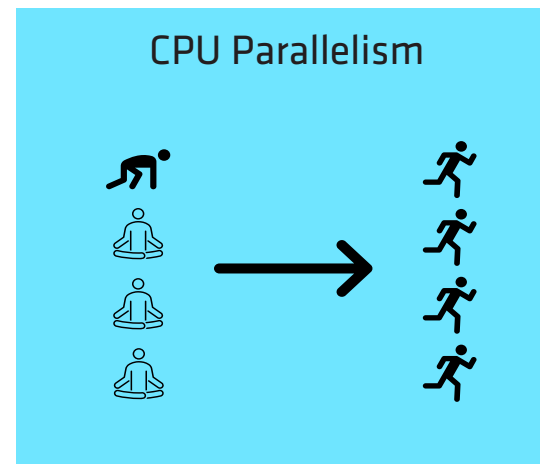
# EXPLICIT APIS: A GREAT LEAP FORWARD

- Unlock Parallelism



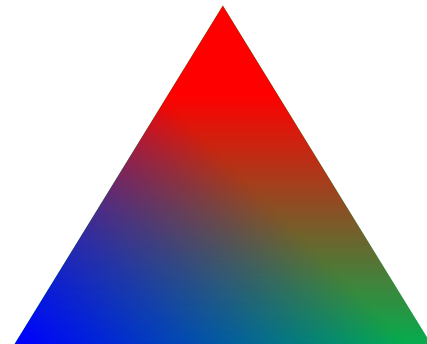
# EXPLICIT APIS: A GREAT LEAP FORWARD

- Unlock Parallelism
- Compute Ahead of Time



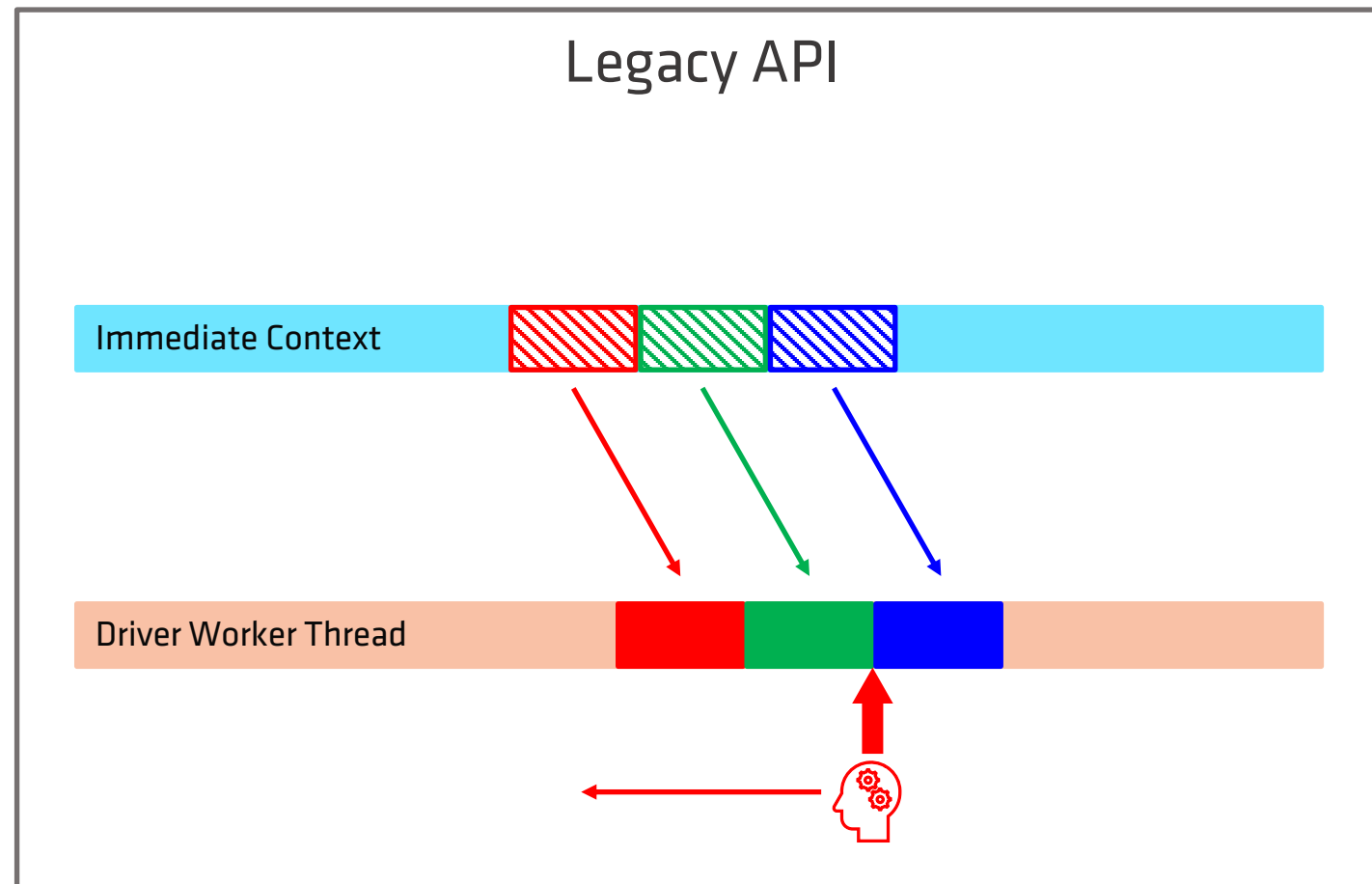
# EXPLICIT APIS: CHALLENGES

Besides “The Lines of Code to the First Triangle”...



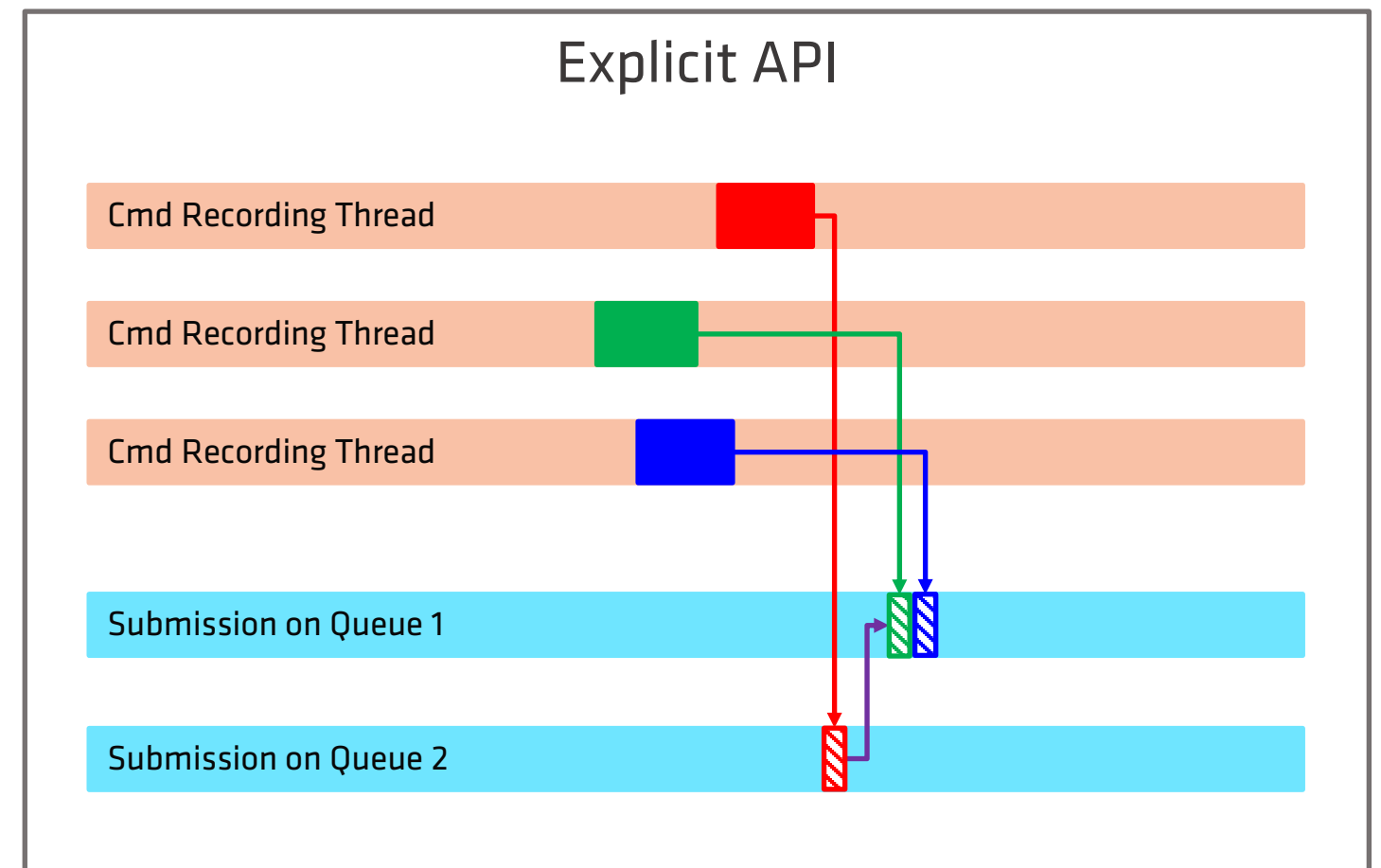
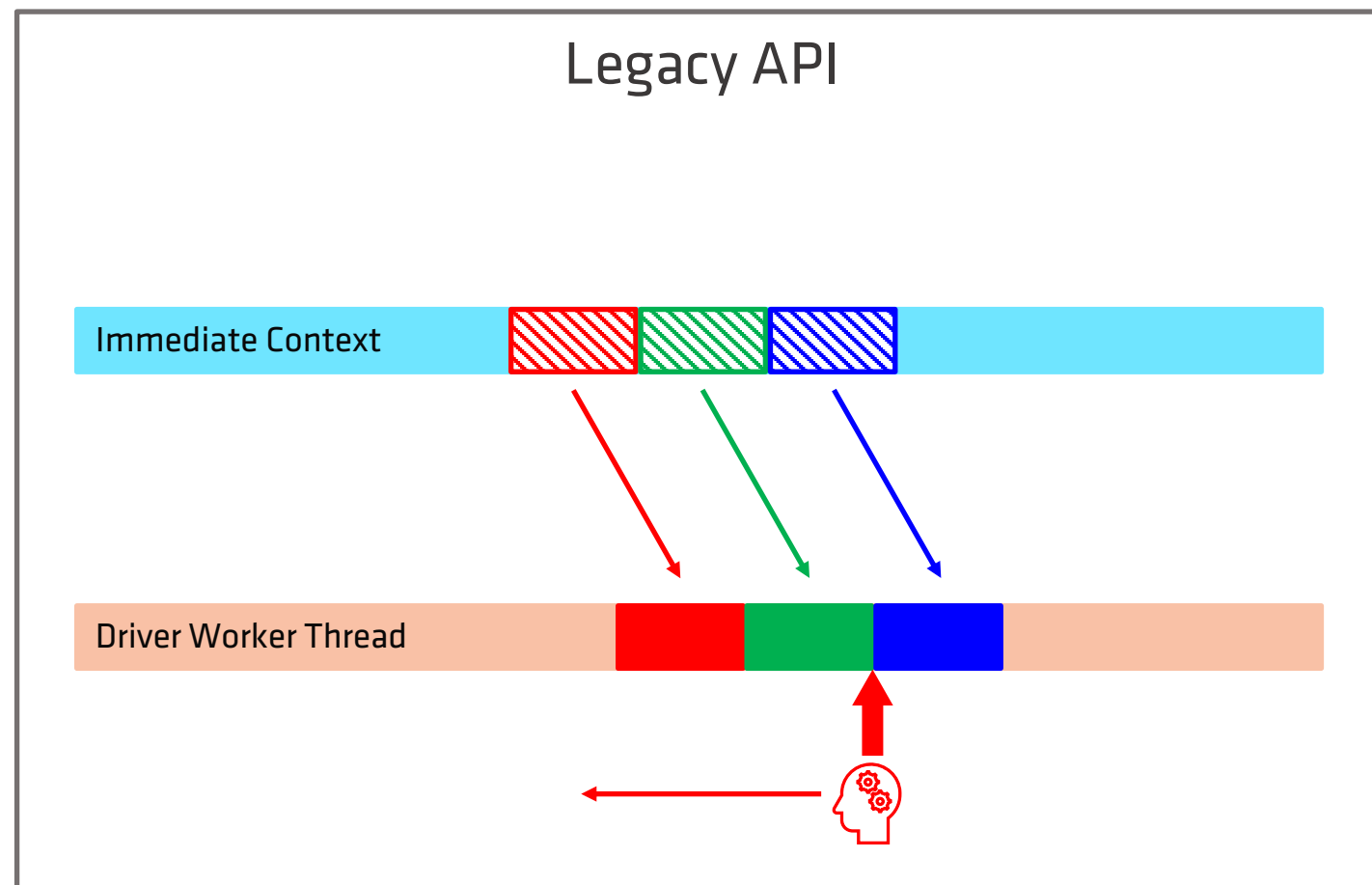
# EXPLICIT APIS: CHALLENGES

Besides “The Lines of Code to the First Triangle”...



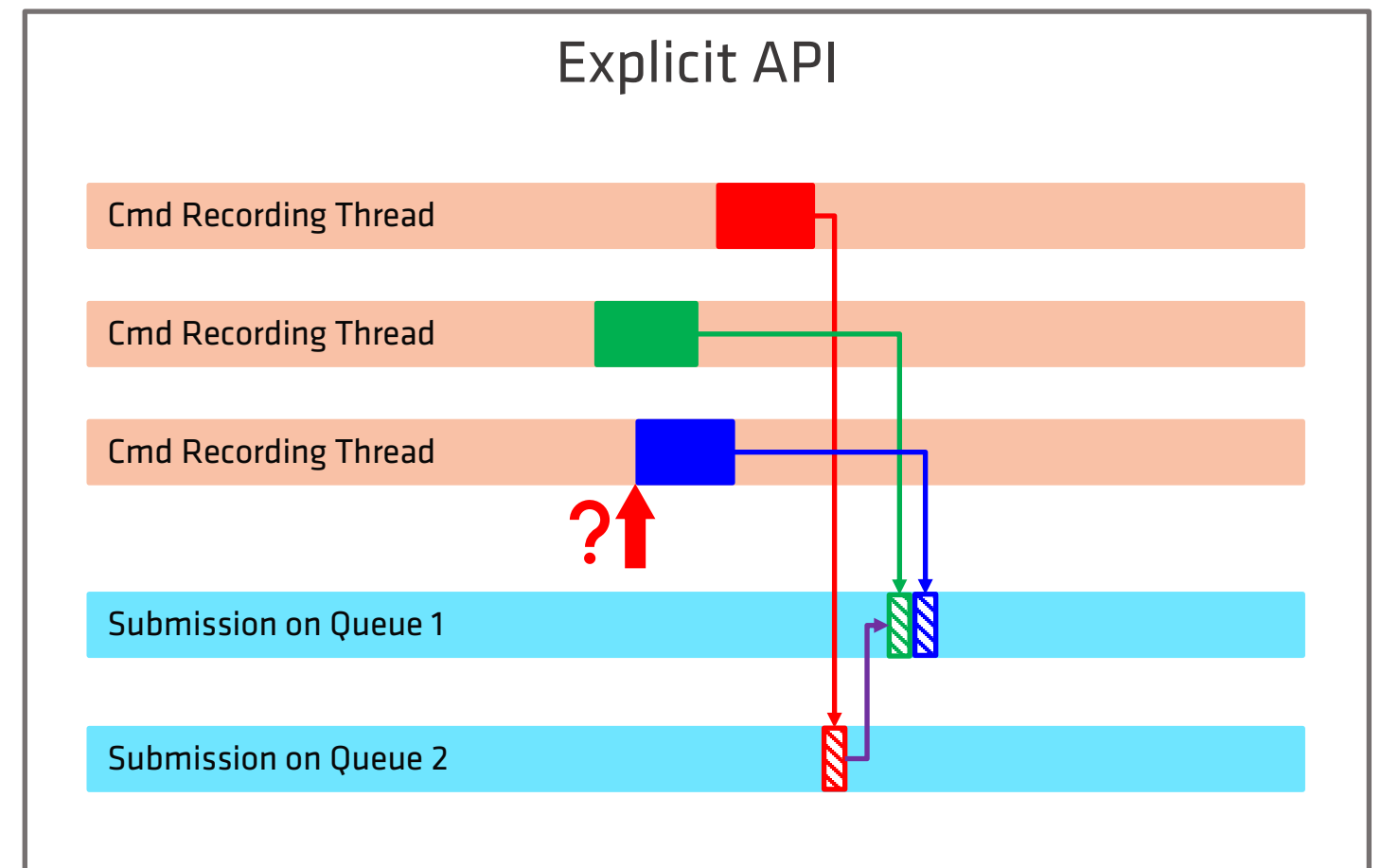
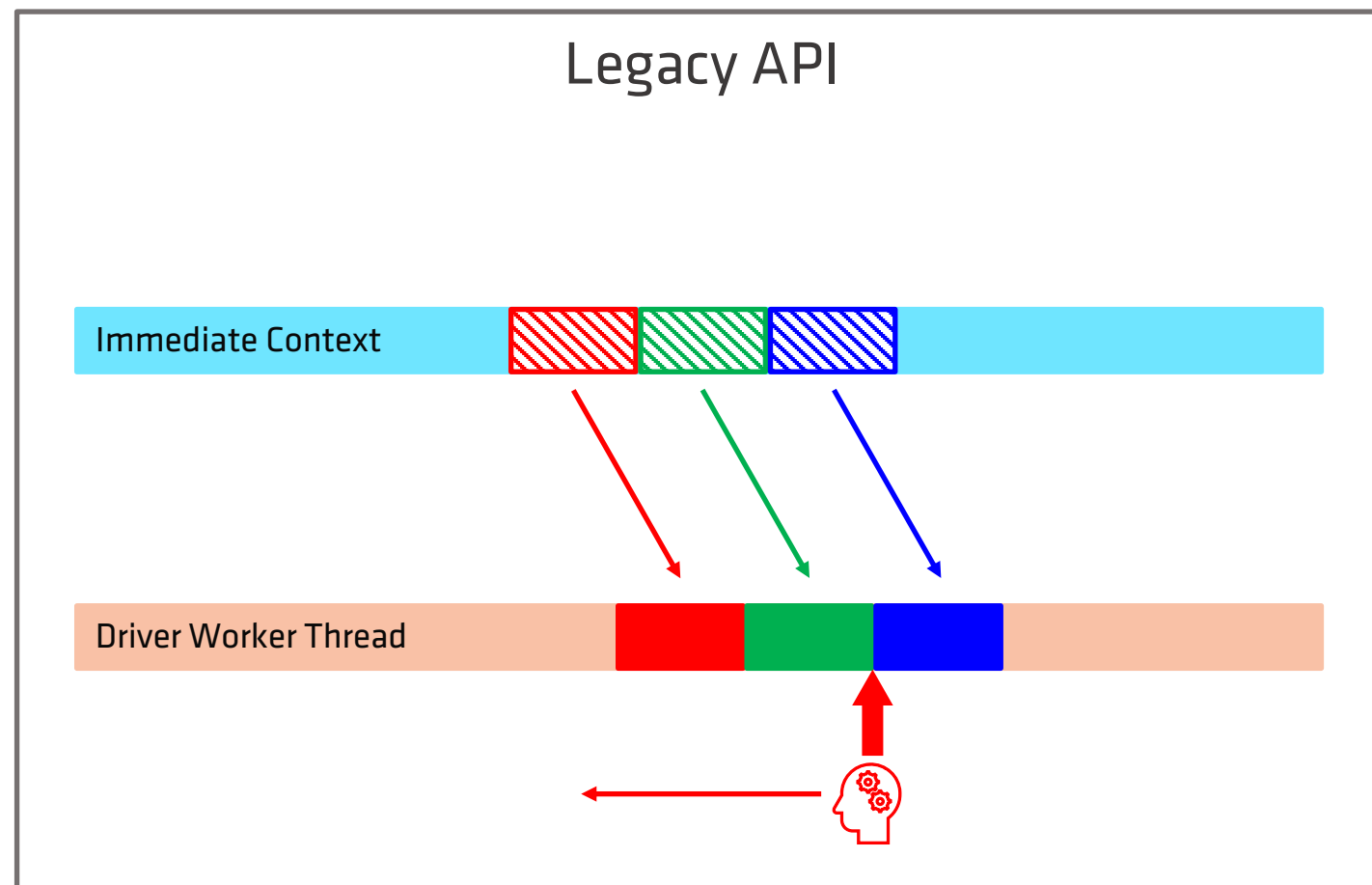
# EXPLICIT APIS: CHALLENGES

Besides “The Lines of Code to the First Triangle” ...



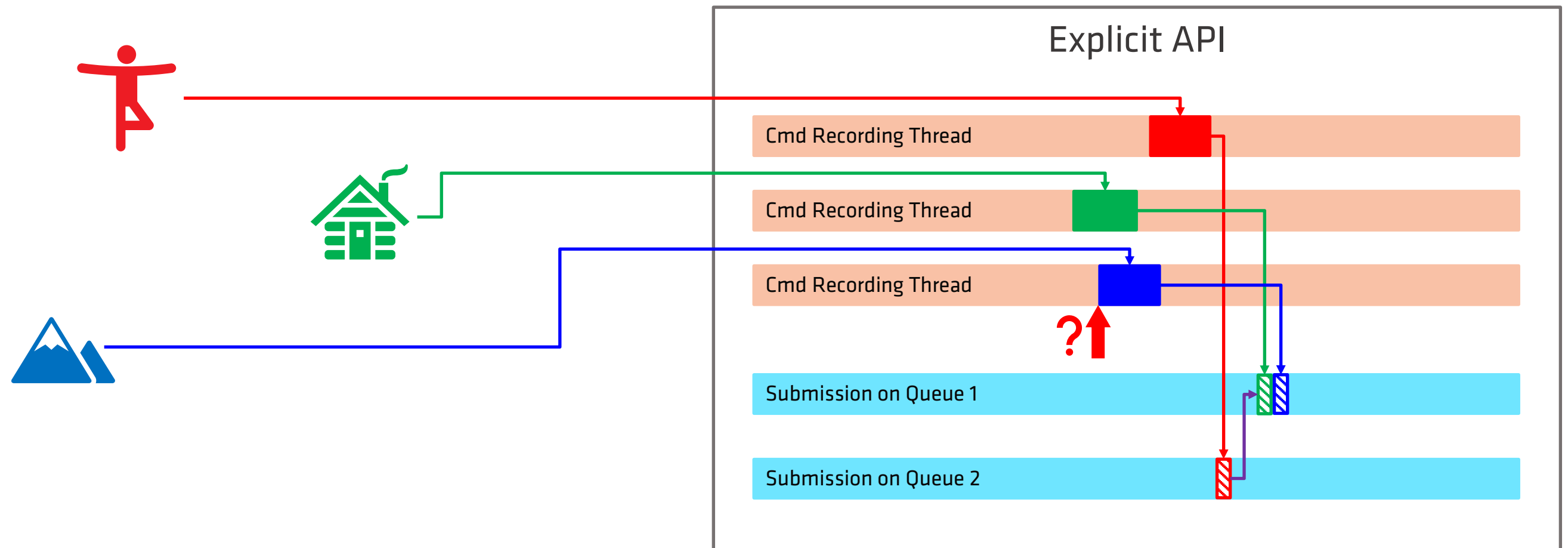
# EXPLICIT APIS: CHALLENGES

Besides “The Lines of Code to the First Triangle” ...



# EXPLICIT APIS: CHALLENGES

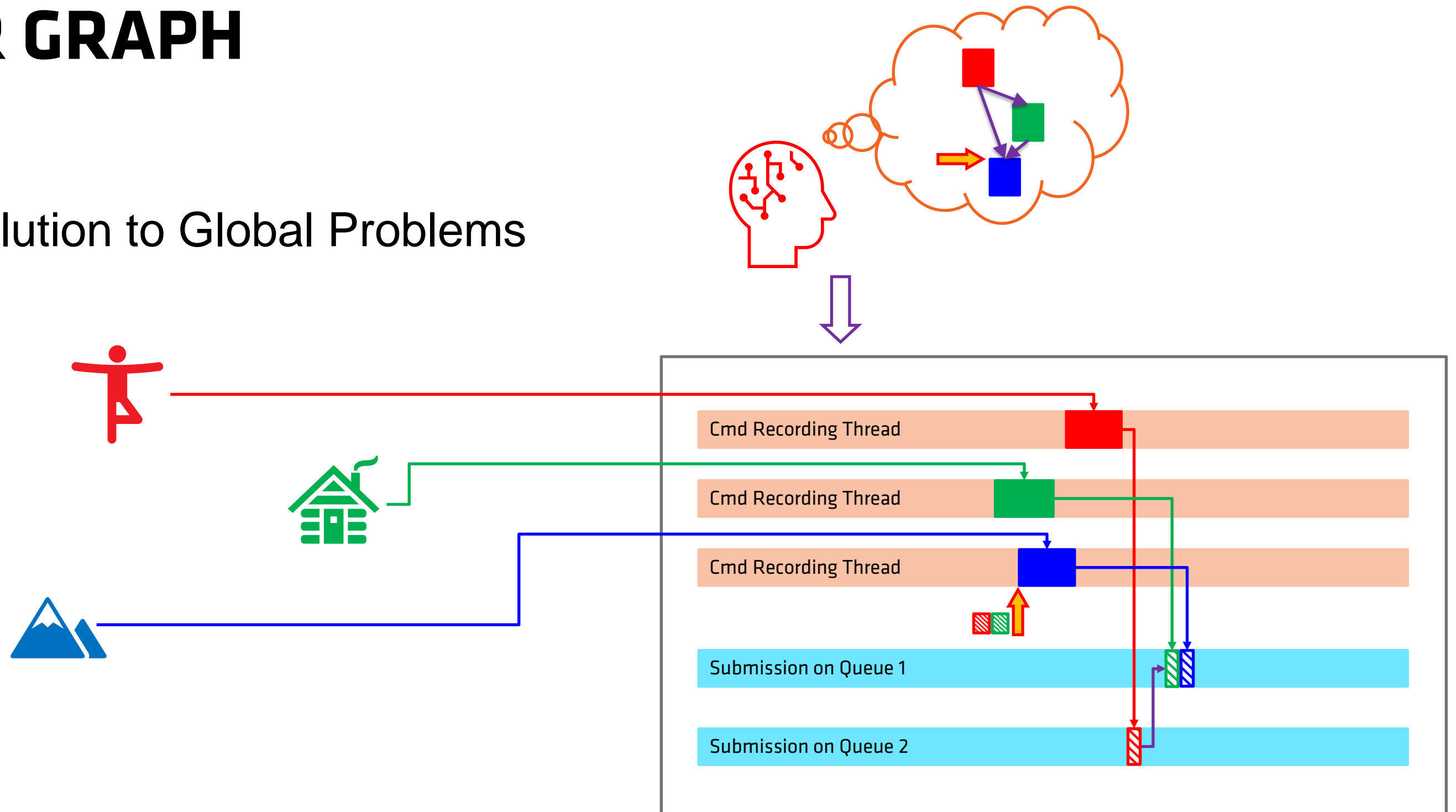
Besides “The Lines of Code to the First Triangle” ...





# RENDER GRAPH

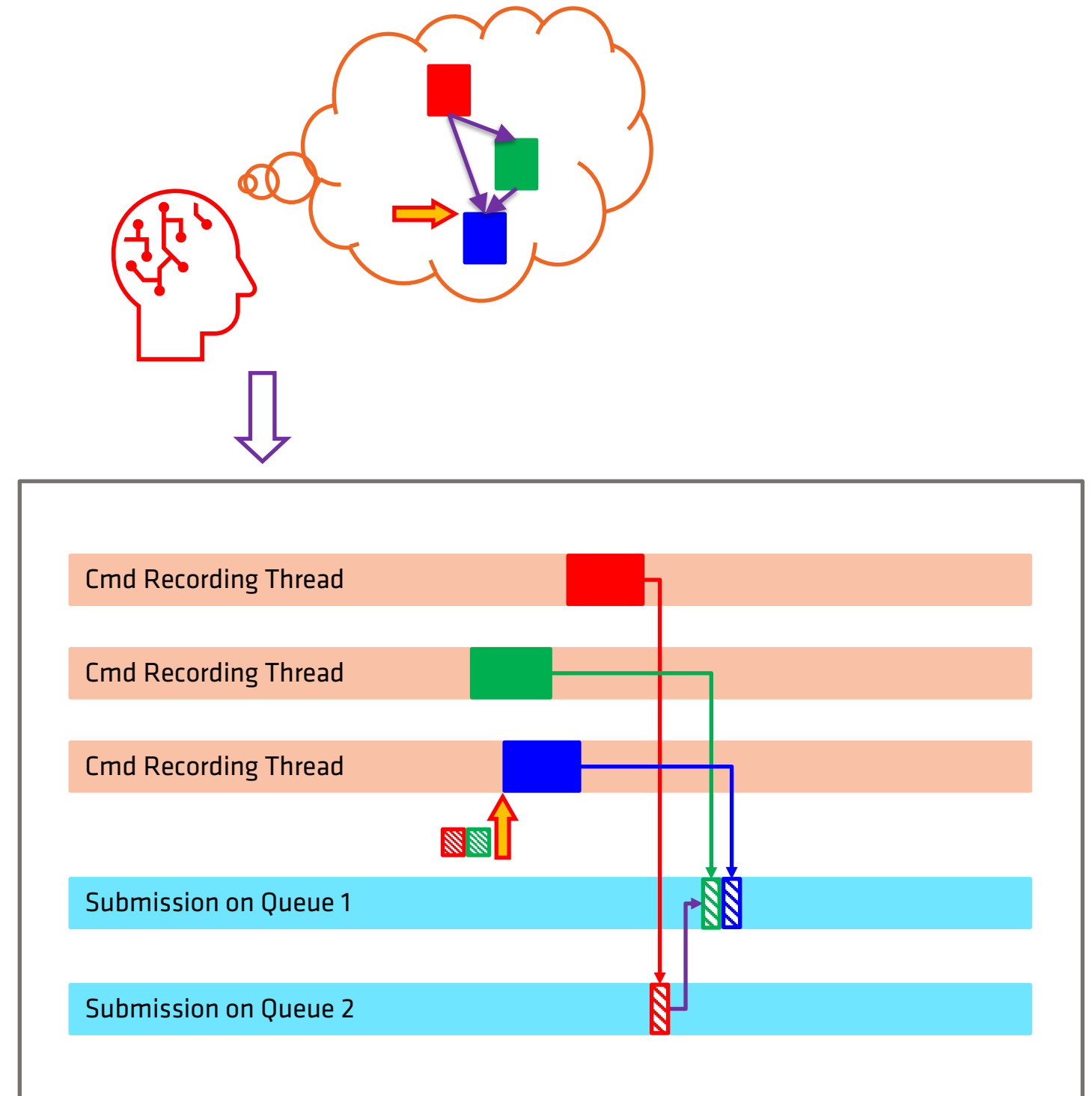
A Global Solution to Global Problems



# RENDER GRAPH

## A Global Solution to Global Problems

- High level view of data flow & dependencies
- Schedule resources & barriers globally
- Compiler-like architecture



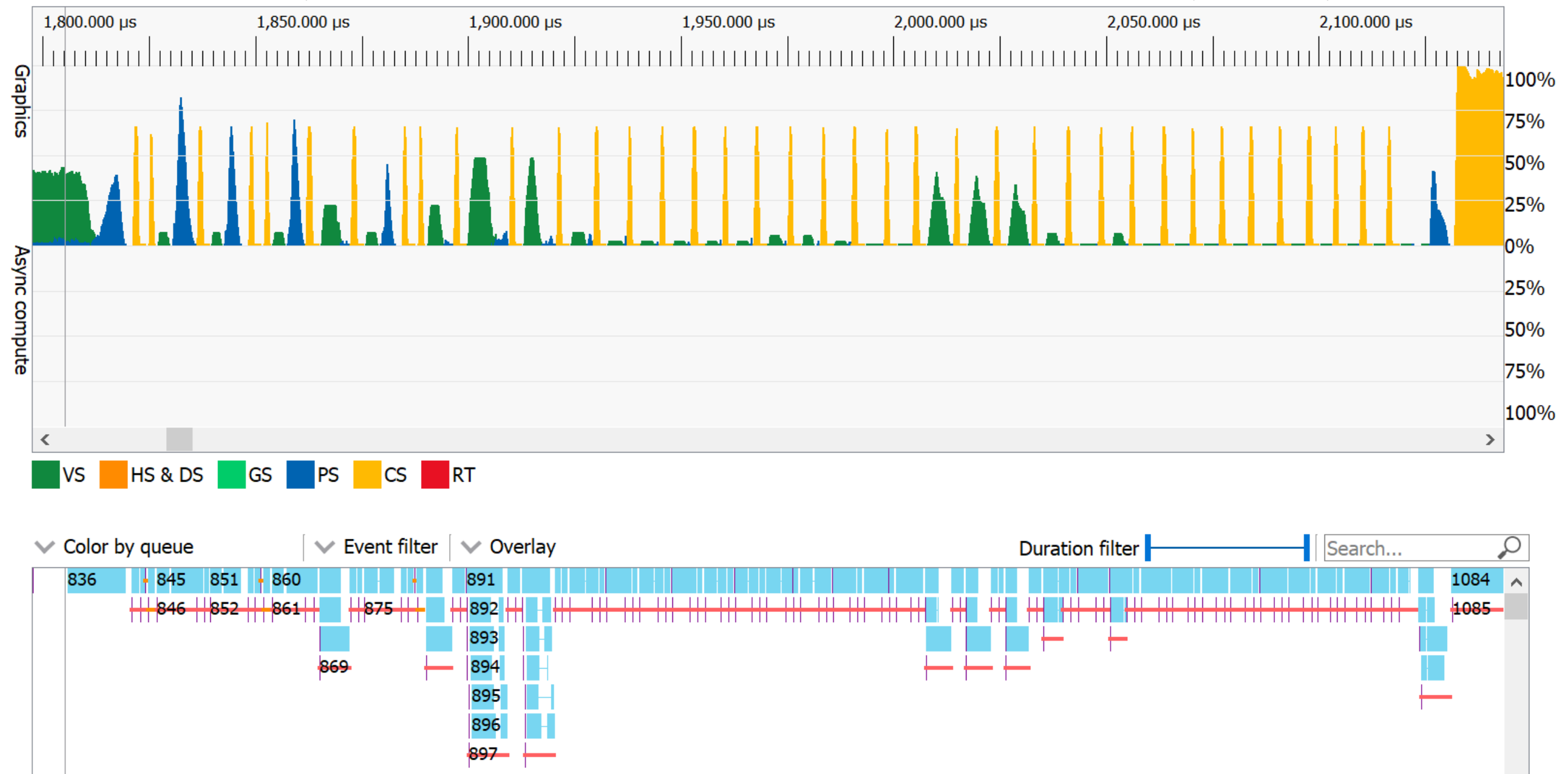
# RENDER GRAPH: CHALLENGES



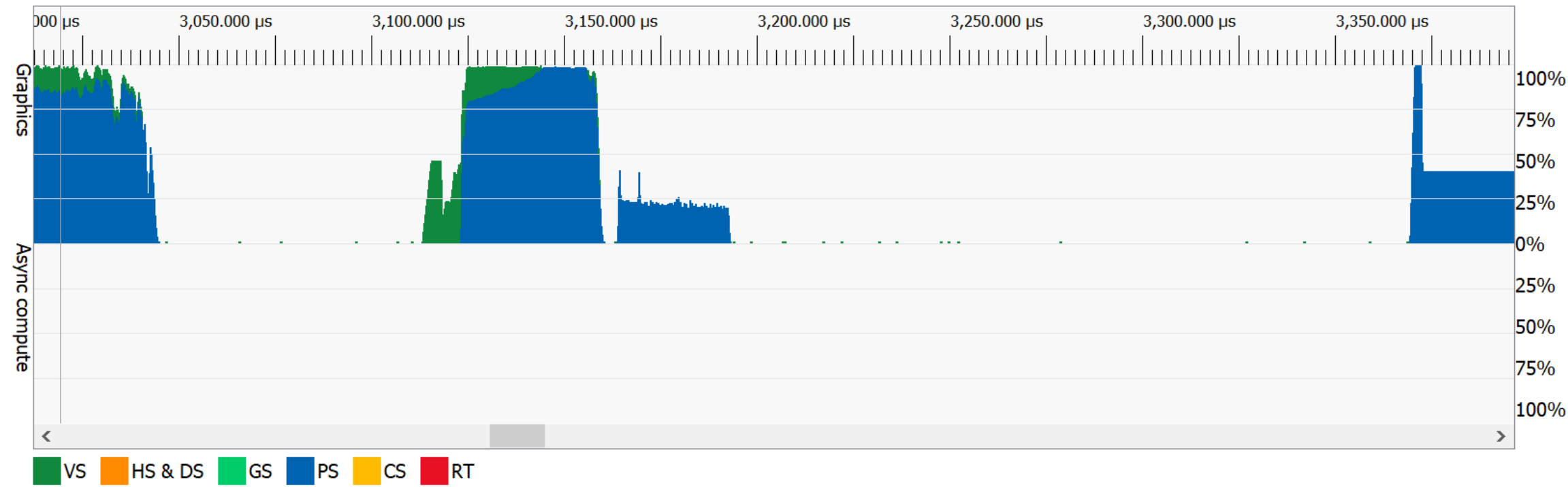
# RENDER GRAPH: CHALLENGES

- Engine-Specific
  - Significant initial investment
  - Hard to share / reuse
  - Still many engines / titles can't enjoy the benefits

# Common Problem: Excessive barriers & Suboptimal scheduling



# Common Problem: Suboptimal resource state / image layout



## 3054 CmdDepthStencilResummarize()

Launched from Direct queue

Start time 3,046.759 μs  
 End time 3,061.448 μs  
 Duration 14.688 μs  
 Hardware context 1 rolled

### Parent Barrier

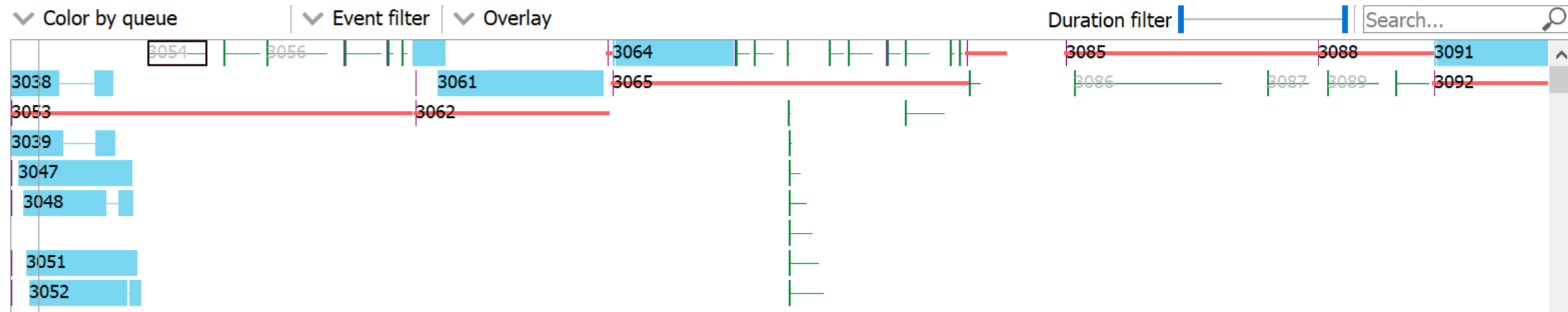
3053 ResourceBarrier()

### Wavefronts

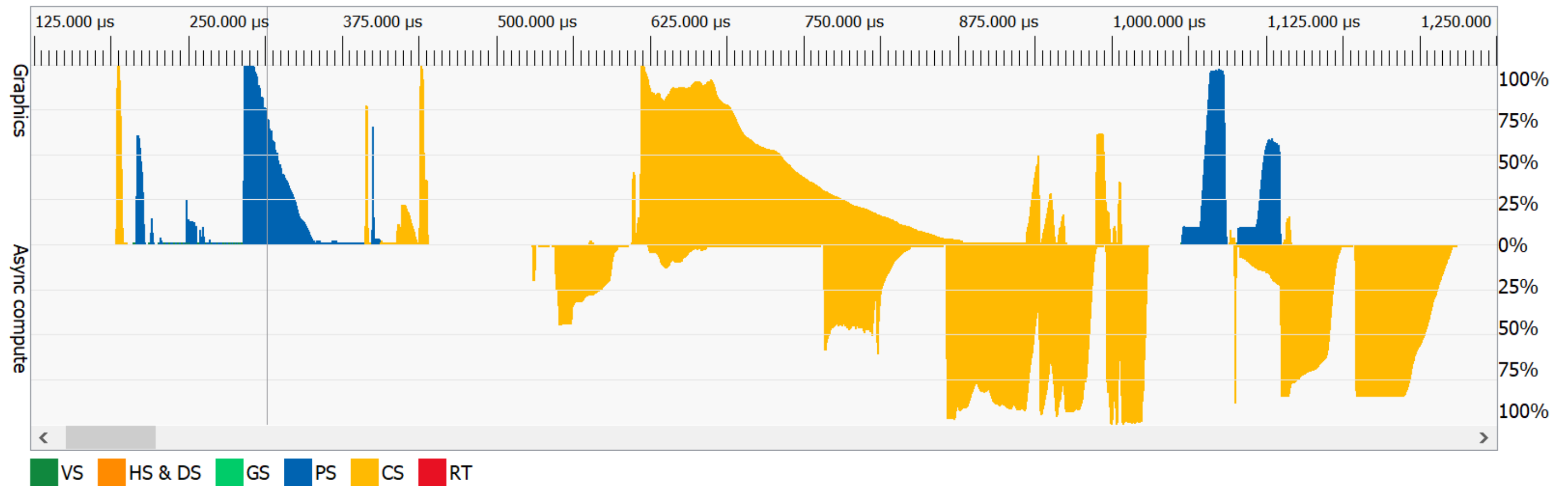
SurfS wavefronts 1 (100.00%)  
 PrimS wavefronts -  
 VS Wavefronts {0}  
 PS Wavefronts -  
 CS Wavefronts -

Total wavefronts 1

Total threads 1

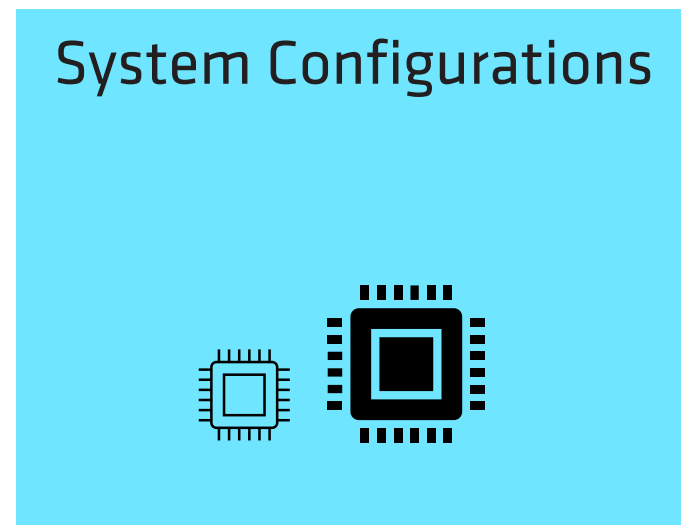


# Common Problem: Multi-Queue scheduling overhead & alignment



# RENDER GRAPH: CHALLENGES

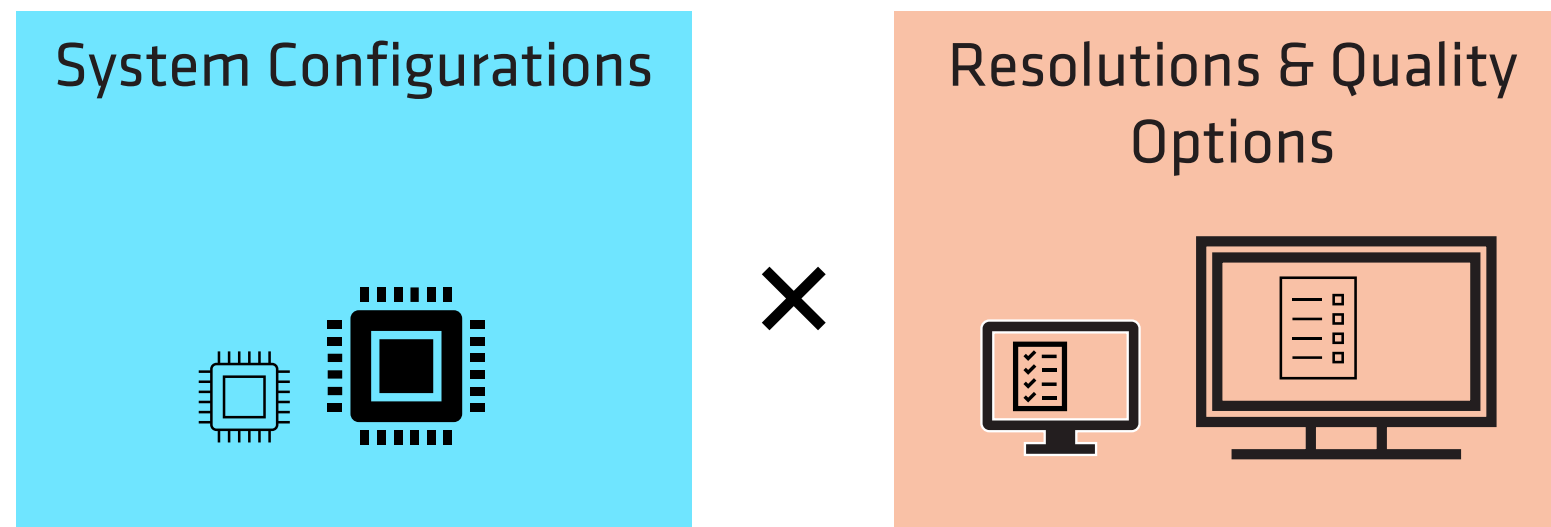
- Low-level optimizations are often scenario-dependent





# RENDER GRAPH: CHALLENGES

- Low-level optimizations are often scenario-dependent



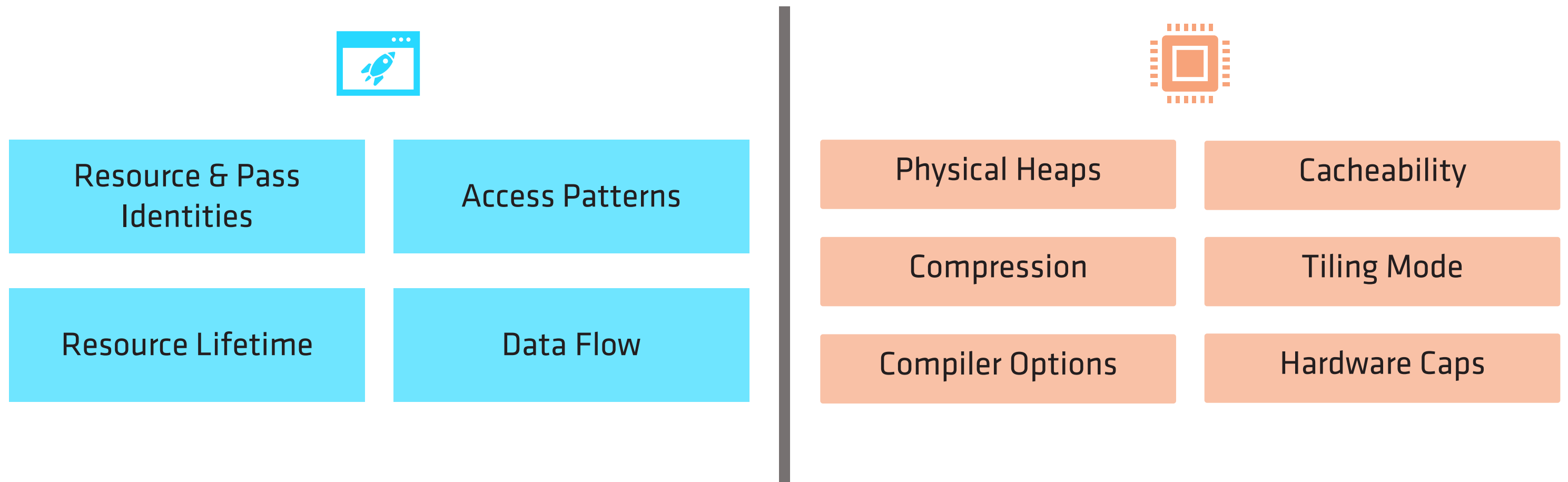
# RENDER GRAPH: CHALLENGES

- Low-level optimizations are often scenario-dependent



# RENDER GRAPH: CHALLENGES

- Useful information lost in translation



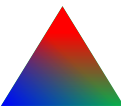
# GOALS

- Make Render Graphs easier & more accessible
  - Generally optimal barrier / resource scheduling by default
  - Easier componentization: move, copy & share!

# GOALS

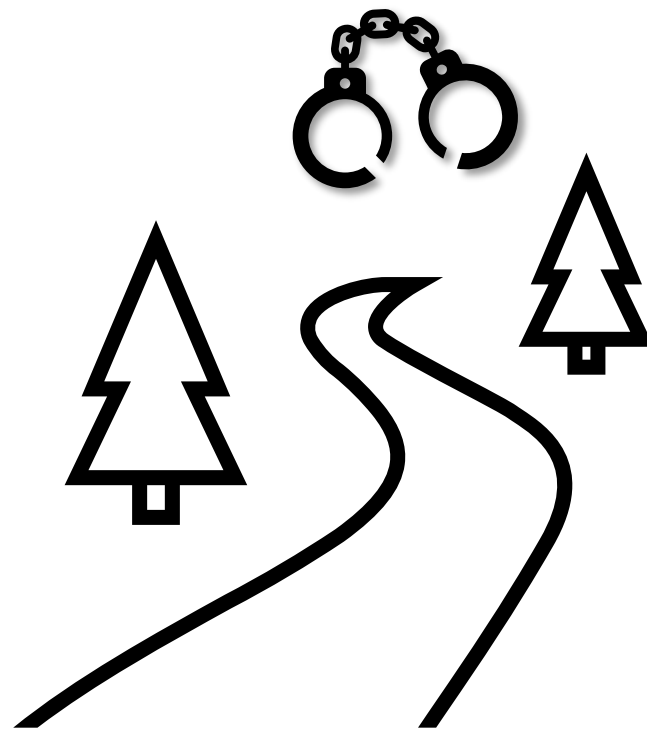
- Make Render Graphs easier & more accessible
  - Generally optimal barrier / resource scheduling by default
  - Easier componentization: move, copy & share!
- Squeeze more performance
  - Convey more usage & identity info to lower-level of the software stack
  - Simplify scenario-specific micro-optimizations at scale

# GOALS

- Make Render Graphs easier & more accessible
  - Generally optimal barrier / resource scheduling by default
  - Easier componentization: move, copy & share!
- Squeeze more performance
  - Convey more usage & identity info to lower-level of the software stack
  - Simplify scenario-specific micro-optimizations at scale
- Bonus point: Simplify the API
  - LoC to 
  - More static analysis
  - Less repetition

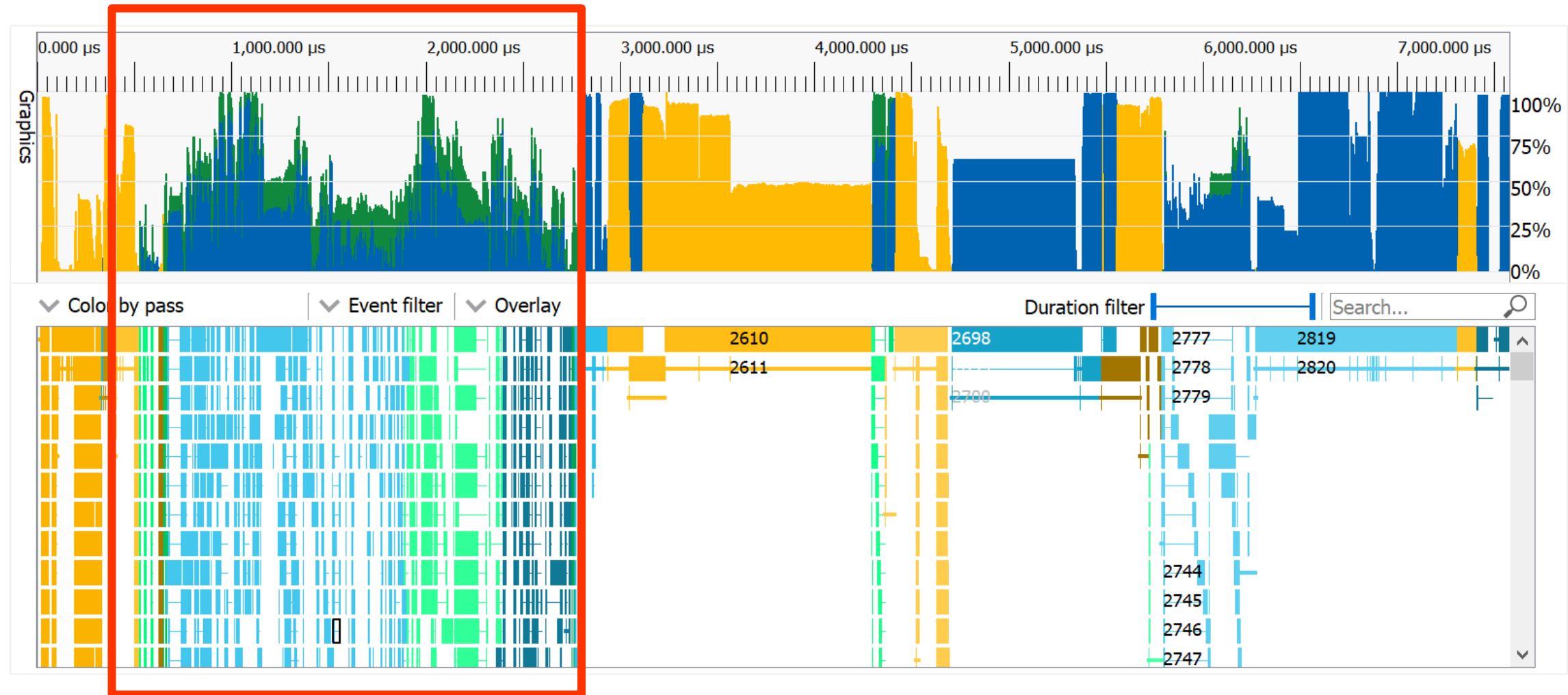
# QUESTIONS REMAINING

- Aren't we cycling back to the old, serialized API full of choke points?
- Aren't we losing controllability and predictability?



The Road (back) to Serfdom?

# A TALE OF TWO DRAWS





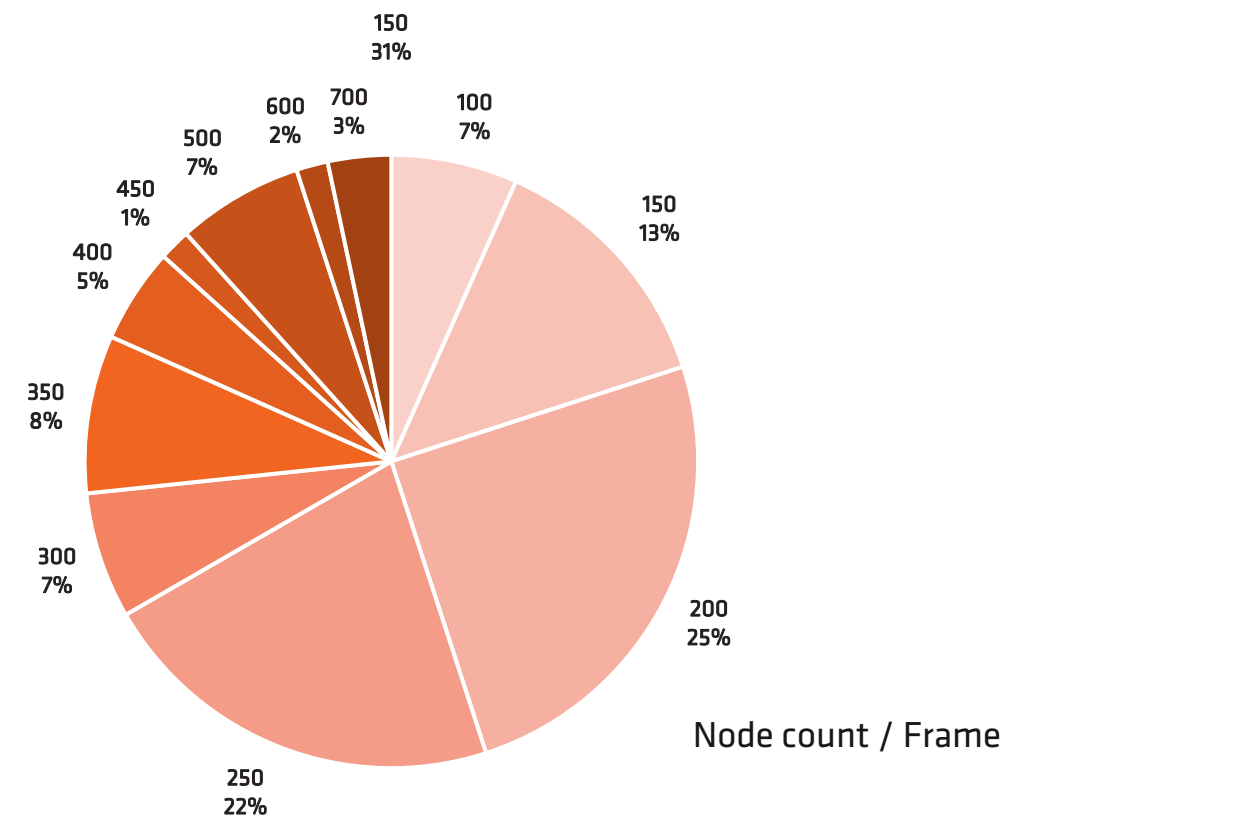
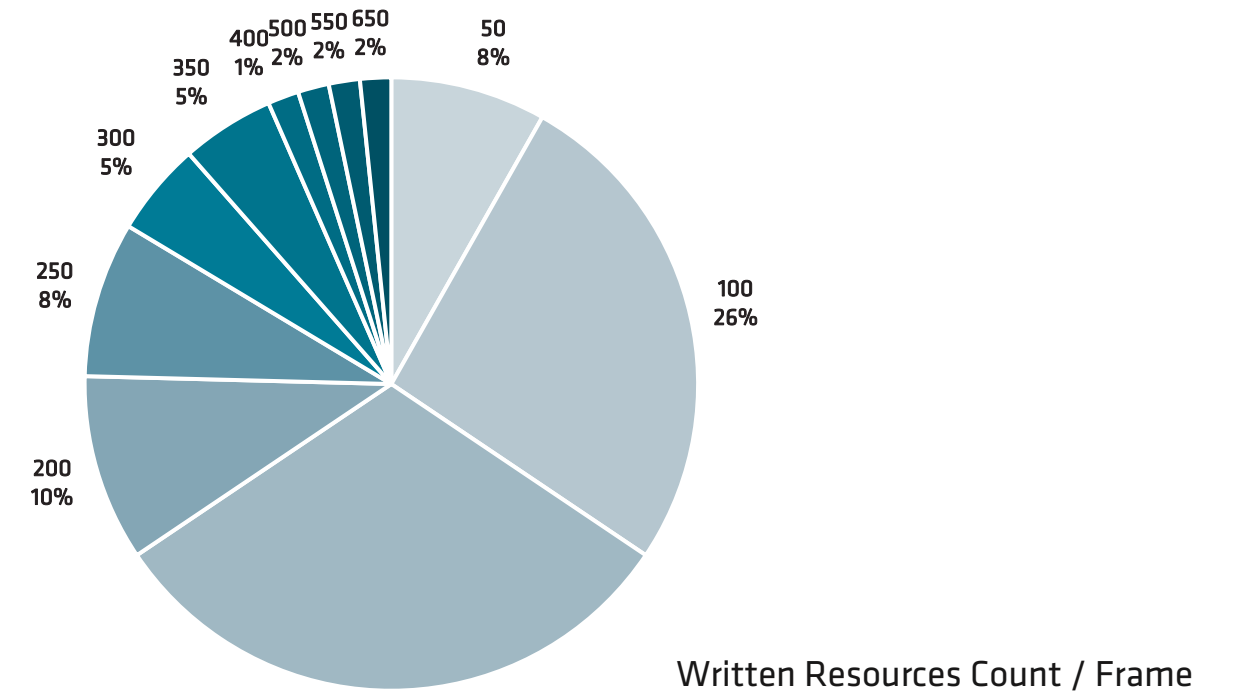
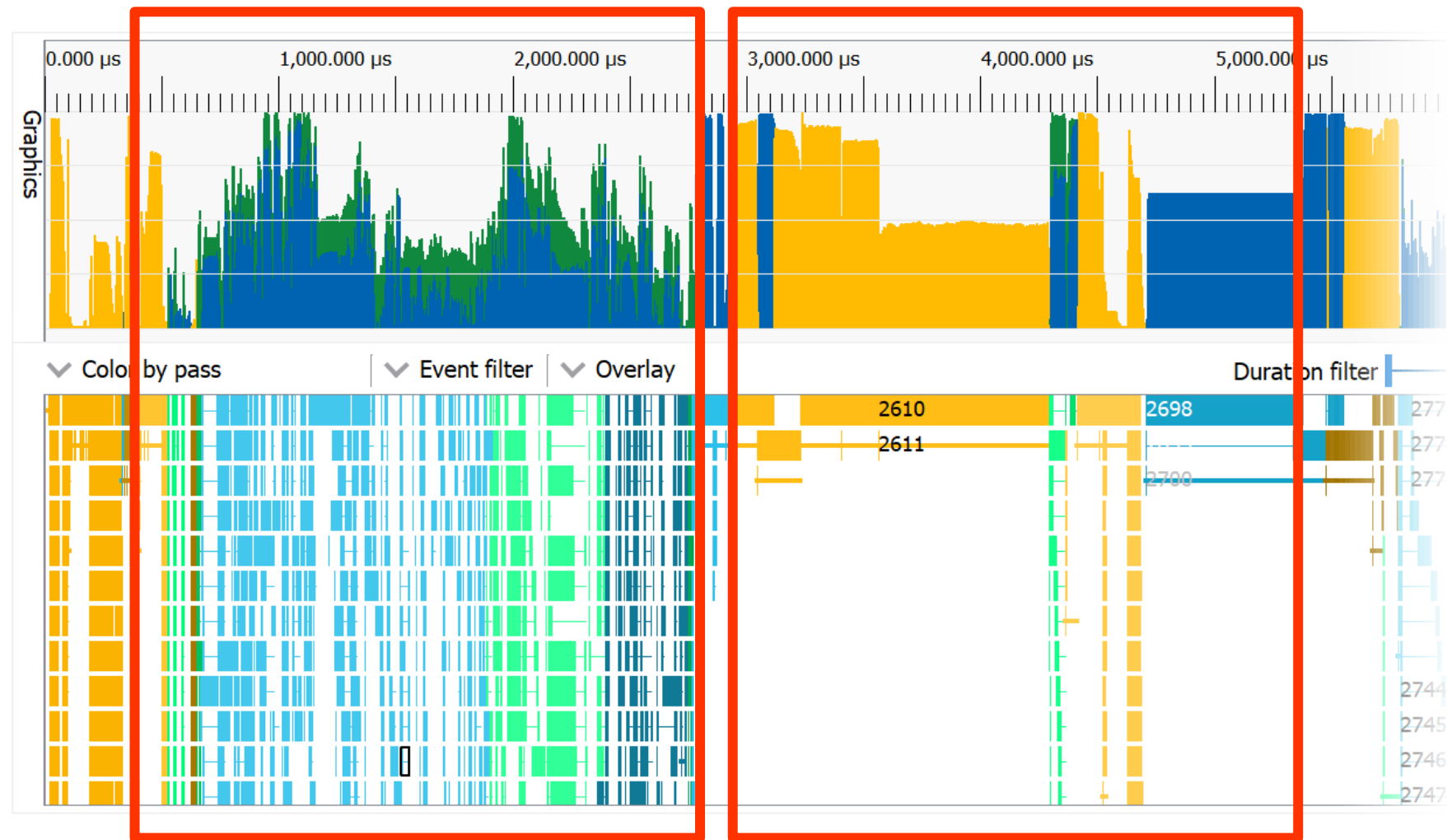
# A TALE OF TWO DRAWS



# A TALE OF TWO DRAWS



# SIZE OF THE PROBLEM



Source: AMD Engineering using RenderDoc Captures + RPS Replay Tool, as of Nov 2022.

# FINDING THE RECYCLABLES

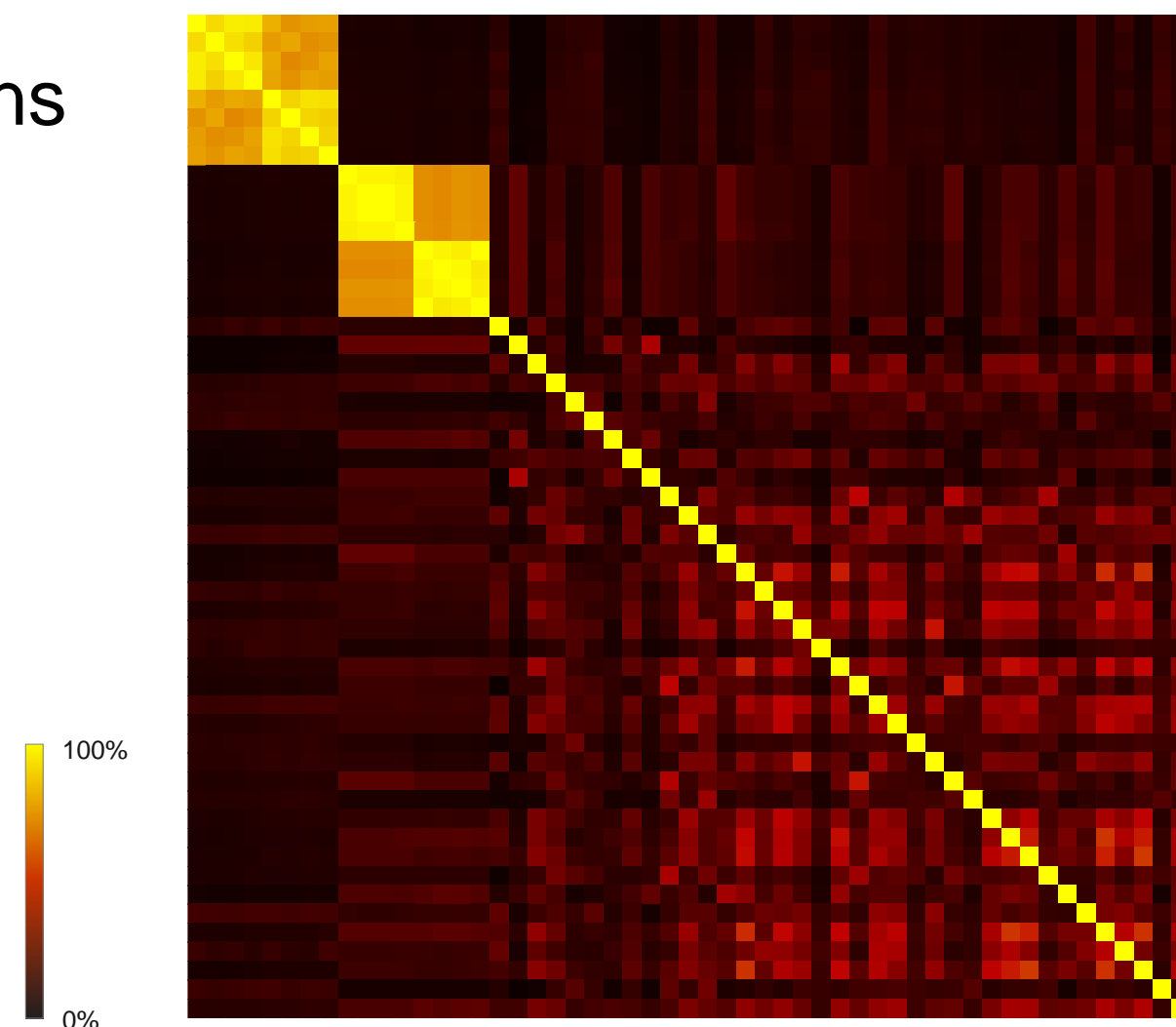
- Frame structure is stable while not fully static
  - Enables deterministic resource reuse
  - Enables iterative profiling-guided optimizations

Frame to Frame API sequence similarity  
(excluding geometry passes):

Consecutive 4 frames: ~92-98%

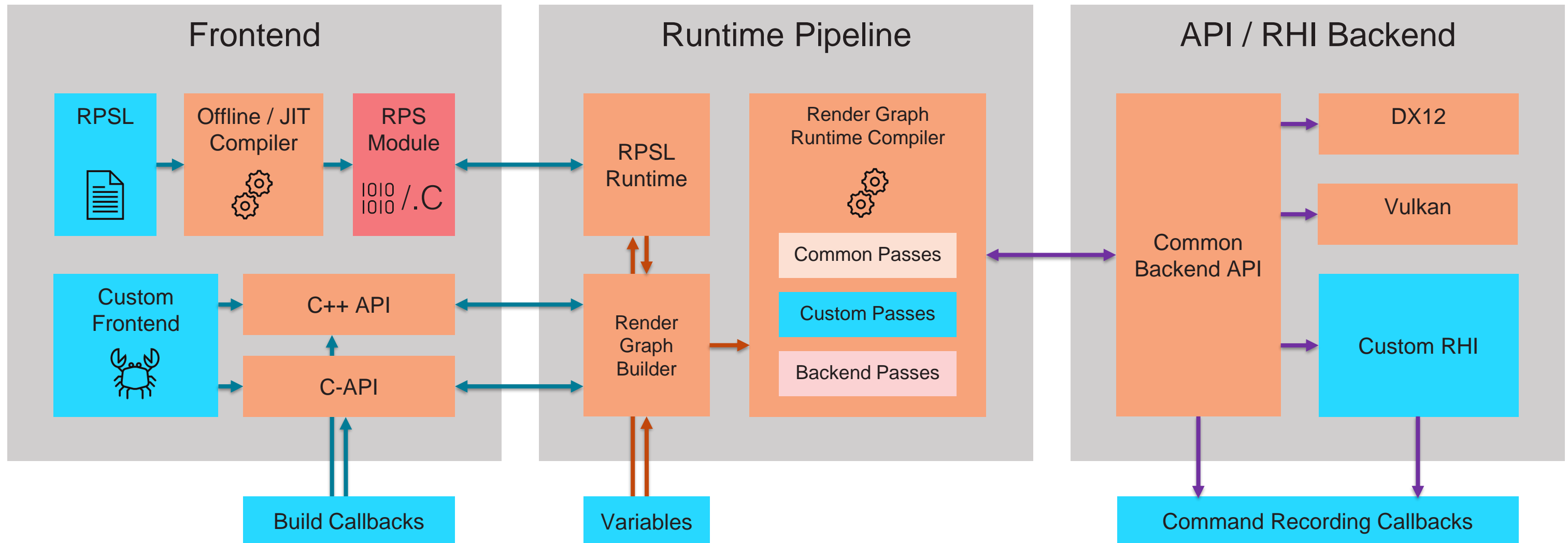
Same title, two scenes: ~80%

Different games: ~22%

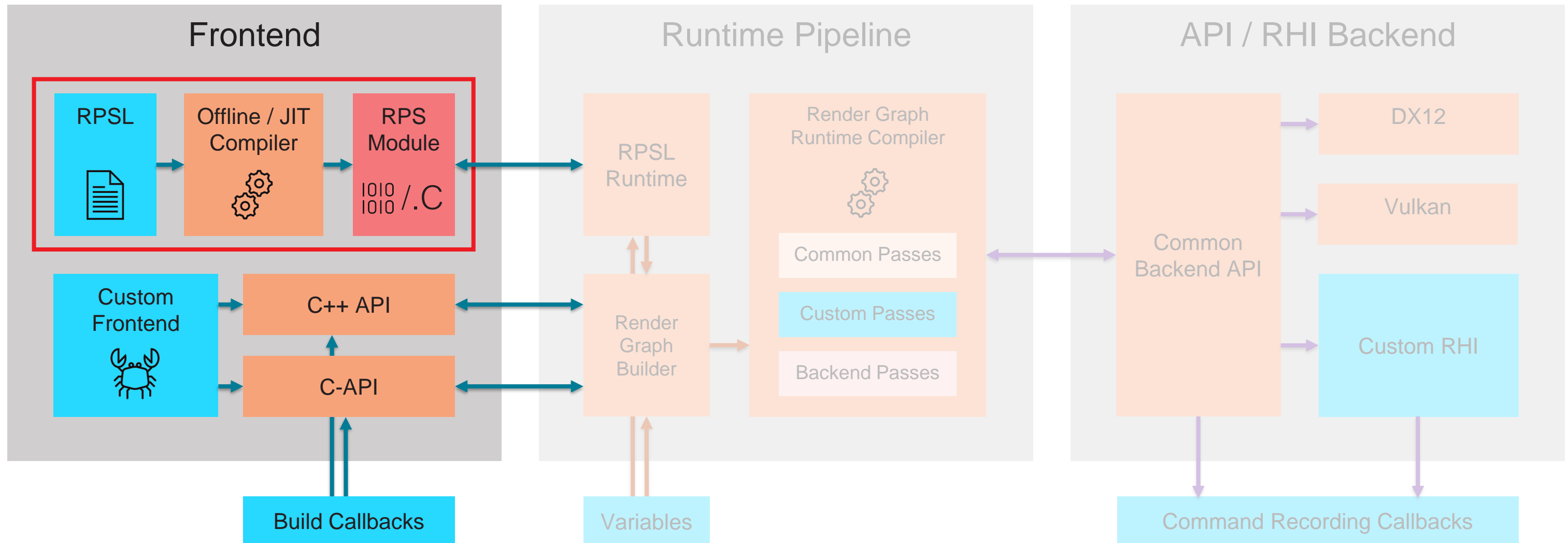


Source: AMD Engineering using RenderDoc Captures + RPS Replay Tool, as of Nov 2022.

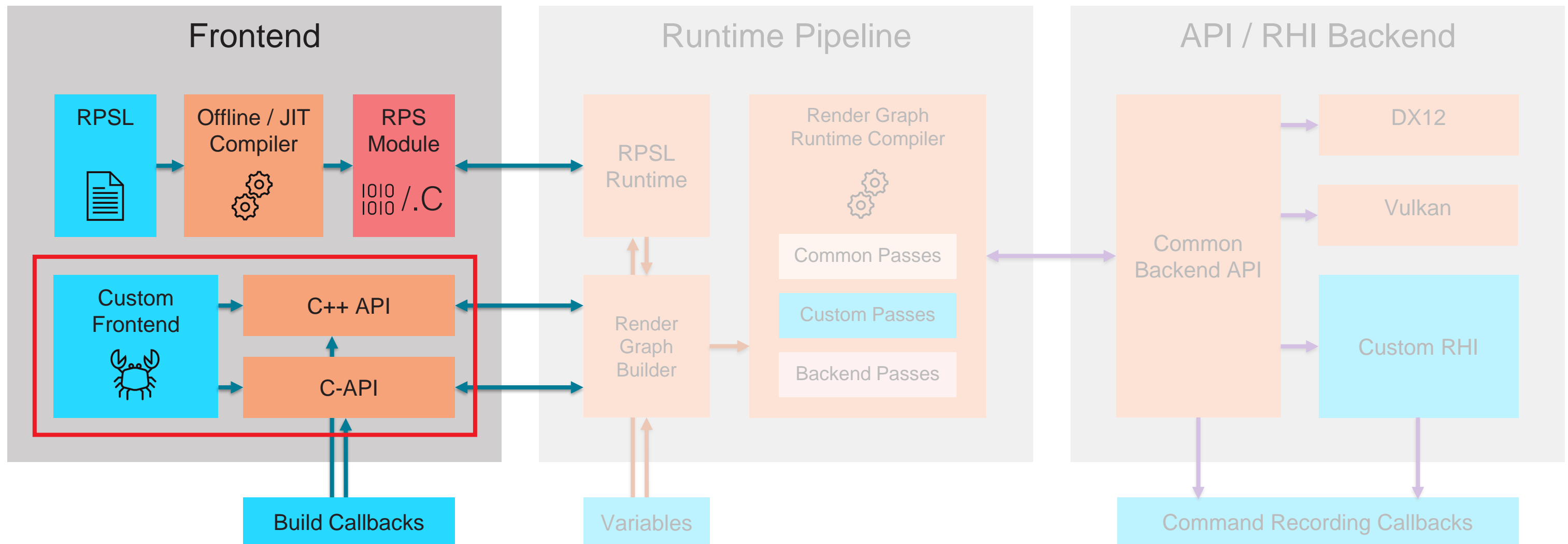
# THE SDK



# THE FRONTEND - RPSL TOOLCHAIN



# THE FRONTEND - C/C++ API



```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale([readwrite(renderTarget)] texture dest : SV_Target0,
7             [readonly(ps)] texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     // Declare a transient texture resource
13     const ResourceDesc backBufferDesc = backBuffer.desc();
14     texture offscreen = create_tex2d(backBufferDesc.Format,
15                                     uint(backBufferDesc.Width) / 2,
16                                     uint(backBufferDesc.Height) / 2);
17
18     // Built-in clear node
19     clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     // Render to offscreen texture
22     // with user defined "Triangle" node
23     Triangle(offscreen);
24
25     // Blt offscreen to backbuffer
26     // with user defined "Upscale" node
27     Upscale(backBuffer, offscreen);
28 }
```



```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale([readwrite(renderTarget)] texture dest : SV_Target0,
7             [readonly(ps)] texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     // Declare a transient texture resource
13     const ResourceDesc backBufferDesc = backBuffer.desc();
14     texture offscreen = create_tex2d(backBufferDesc.Format,
15                                     uint(backBufferDesc.Width) / 2,
16                                     uint(backBufferDesc.Height) / 2);
17
18     // Built-in clear node
19     clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     // Render to offscreen texture
22     // with user defined "Triangle" node
23     Triangle(offscreen);
24
25     // Blt offscreen to backbuffer
26     // with user defined "Upscale" node
27     Upscale(backBuffer, offscreen);
28 }
```

```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale([readwrite(renderTarget)] texture dest : SV_Target0,
7             [readonly(ps)] texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     // Declare a transient texture resource
13     const ResourceDesc backBufferDesc = backBuffer.desc();
14     texture offscreen = create_tex2d(backBufferDesc.Format,
15                                     uint(backBufferDesc.Width) / 2,
16                                     uint(backBufferDesc.Height) / 2);
17
18     // Built-in clear node
19     clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     // Render to offscreen texture
22     // with user defined "Triangle" node
23     Triangle(offscreen);
24
25     // Blt offscreen to backbuffer
26     // with user defined "Upscale" node
27     Upscale(backBuffer, offscreen);
28 }
```

hello\_triangle.rpsl X

□ ...

```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale([readwrite(renderTarget)] texture dest : SV_Target0,
7 [readonly(ps)] texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     // Declare a transient texture resource
13     const ResourceDesc backBufferDesc = backBuffer.desc();
14     texture offscreen = create_tex2d(backBufferDesc.Format,
15     uint(backBufferDesc.Width) / 2,
16     uint(backBufferDesc.Height) / 2);
17
18     // Built-in clear node
19     clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     // Render to offscreen texture
22     // with user defined "Triangle" node
23     Triangle(offscreen);
24
25     // Blt offscreen to backbuffer
26     // with user defined "Upscale" node
27     Upscale(backBuffer, offscreen);
28 }
```

```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale([readwrite(renderTarget)] texture dest : SV_Target0,
7             [readonly(ps)] texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     // Declare a transient texture resource
13     const ResourceDesc backBufferDesc = backBuffer.desc();
14     texture offscreen = create_tex2d(backBufferDesc.Format,
15                                     uint(backBufferDesc.Width) / 2,
16                                     uint(backBufferDesc.Height) / 2);
17
18     // Built-in clear node
19     clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     // Render to offscreen texture
22     // with user defined "Triangle" node
23     Triangle(offscreen);
24
25     // Blt offscreen to backbuffer
26     // with user defined "Upscale" node
27     Upscale(backBuffer, offscreen);
28 }
```

```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale([readwrite(renderTarget)] texture dest : SV_Target0,
7             [readonly(ps)] texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     ... // Declare a transient texture resource
13     ... const ResourceDesc backBufferDesc = backBuffer.desc();
14     ... texture offscreen = create_tex2d(backBufferDesc.Format,
15     ...                                     uint(backBufferDesc.Width) / 2,
16     ...                                     uint(backBufferDesc.Height) / 2);
17
18     ... // Built-in clear node
19     ... clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     ... // Render to offscreen texture
22     ... // with user defined "Triangle" node
23     ... Triangle(offscreen);
24
25     ... // Blt offscreen to backbuffer
26     ... // with user defined "Upscale" node
27     ... Upscale(backBuffer, offscreen);
28 }
```

```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale([readwrite(renderTarget)] texture dest : SV_Target0,
7             [readonly(ps)] texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     // Declare a transient texture resource
13     const ResourceDesc backBufferDesc = backBuffer.desc();
14     texture offscreen = create_tex2d(backBufferDesc.Format,
15                                     uint(backBufferDesc.Width) / 2,
16                                     uint(backBufferDesc.Height) / 2);
17
18     // Built-in clear node
19     clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     // Render to offscreen texture
22     // with user defined "Triangle" node
23     Triangle(offscreen);
24
25     // Blt offscreen to backbuffer
26     // with user defined "Upscale" node
27     Upscale(backBuffer, offscreen);
28 }
```



```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale ([readwrite(renderTarget)] texture dest : SV_Target0,
7 |.....|.....|.....| [readonly(ps)] ..... texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     ....// Declare a transient texture resource
13     ....const ResourceDesc backBufferDesc = backBuffer.desc();
14     ....texture offscreen = create_tex2d(backBufferDesc.Format,
15 |.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
16 |.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
17 |.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
18     ....// Built-in clear node
19     ....clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     ....// Render to offscreen texture
22     ....// with user defined "Triangle" node
23     ....Triangle(offscreen);
24
25     ....// Blt offscreen to backbuffer
26     ....// with user defined "Upscale" node
27     ....Upscale(backBuffer, offscreen);
28 }
```

```
1 // Sample RPSL code
2
3 // Node declarations
4 node Triangle([readwrite(renderTarget)] texture renderTarget : SV_Target0);
5
6 node Upscale ([readwrite(renderTarget)] texture dest : SV_Target0,
7 |.....|.....|.....| [readonly(ps)] ..... texture source);
8
9 // Render Graph entry point
10 export void hello_rpsl([readonly(present)] texture backBuffer)
11 {
12     ....// Declare a transient texture resource
13     ....const ResourceDesc backBufferDesc = backBuffer.desc();
14     ....texture offscreen = create_tex2d(backBufferDesc.Format,
15 |.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
16 |.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
17 |.....|.....|.....|.....|.....|.....|.....|.....|.....|.....|
18     ....// Built-in clear node
19     ....clear(offscreen, float4(0.0, 0.2, 0.4, 1.0));
20
21     ....// Render to offscreen texture
22     ....// with user defined "Triangle" node
23     ....Triangle(offscreen);
24
25     ....// Blt offscreen to backbuffer
26     ....// with user defined "Upscale" node
27     ....Upscale(backBuffer, offscreen);
28 }
```

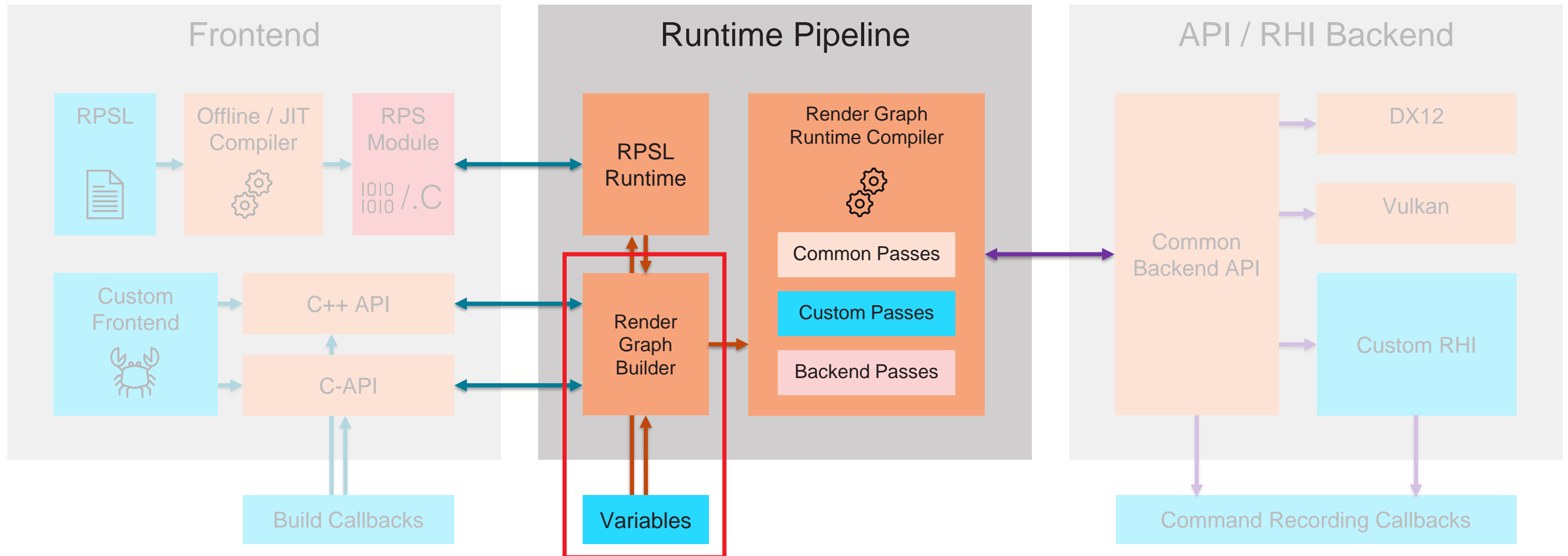




```
1 |
2 ; Note: shader requires additional functionality:
3 ; ..... Use native low precision
4 ;
5 ; shader debug name: 7bdb85ed37c900f0c60dd53676a42afc.pdb
6 ; shader hash: 7bdb85ed37c900f0c60dd53676a42afc
7 ;
8 ; Buffer Definitions:
9 ;
10 ;
11 ; Resource Bindings:
12 ;
13 ; Name ..... Type ..... Format ..... Dim ..... ID ..... HLSL Bind ..... Count
14 ; -----
15 ;
16 target datalayout = "e-m:e-p:32:32-i1:32-i8:8-i16:16-i32:32-i64:64-f16:16-f32:32-f64:64-n8:16:32:64"
17 target triple = "dxil-ms-dx"
18
19 %0 = type { i32, i32, i32, i32 }
20 %__rpsl_node_info_struct = type <{ i32, i32, i32, i32, i32 }>
21 %__rpsl_entry_desc_struct = type <{ i32, i32, i32, i32, i8*, i8* }>
22 %__rpsl_type_info_struct = type <{ i8, i8, i8, i8, i32, i32, i32 }>
23 %__rpsl_params_info_struct = type <{ i32, i32, i32, i32, i32, i16, i16 }>
24 %__rpsl_shader_ref_struct = type <{ i32, i32, i32, i32 }>
25 %__rpsl_pipeline_info_struct = type <{ i32, i32, i32, i32 }>
26 %__rpsl_pipeline_field_info_struct = type <{ i32, i32, i32, i32, i32, i32, i32, i32 }>
27 %__rpsl_pipeline_res_binding_info_struct = type <{ i32, i32, i32, i32 }>
28 %__rpsl_module_info_struct = type <{ i32, i32, i32, i32, i32, i32, i32, i32, i32, i32, i32, i32, [108 x i8]*, [4 x %__rpsl_node_
```

```
40  @@@rps_Str0" = private unnamed_addr constant [10 x i8] c"offscreen\00"
41  @__rpsl_nodedefs_hello_triangle = private constant [4 x %__rpsl_node_info_struct] [%__rpsl_node_info_struct <{ i32 0, i32 57, i32 0,
42  @__rpsl_entries_hello_triangle = private constant [2 x %__rpsl_entry_desc_struct] [%__rpsl_entry_desc_struct <{ i32 0, i32 86, i32 5,
43  @__rpsl_types_metadata_hello_triangle = private constant [3 x %__rpsl_type_info_struct] [%__rpsl_type_info_struct <{ i8 6, i8 0, i8 0,
44  @__rpsl_params_metadata_hello_triangle = private constant [7 x %__rpsl_params_info_struct] [%__rpsl_params_info_struct <{ i32 25, i32
45  @__rpsl_shader_refs_hello_triangle = private constant [1 x %__rpsl_shader_ref_struct] zeroinitializer, align 4
46  @__rpsl_pipelines_hello_triangle = private constant [1 x %__rpsl_pipeline_info_struct] zeroinitializer, align 4
47  @__rpsl_pipeline_fields_hello_triangle = private constant [1 x %__rpsl_pipeline_field_info_struct] zeroinitializer, align 4
48  @__rpsl_pipeline_res_bindings_hello_triangle = private constant [1 x %__rpsl_pipeline_res_binding_info_struct] zeroinitializer, align
49  @__rpsl_string_table_hello_triangle = constant [108 x i8] c"offscreen\00hello_triangle\00t\00data\00renderTarget\00dest\00source\00clear\00"
50  @__rpsl_module_info_hello_triangle = dlllexport constant %__rpsl_module_info_struct <{ i32 1297305682, i32 3, i32 9, i32 10, i32 108, i
51  @@@rps_Str1" = private unnamed_addr constant [12 x i8] c"clear_color\00"
52  @@@rps_Str2" = private unnamed_addr constant [2 x i8] c"t\00"
53  @@@rps_ParamAttr3" = private constant %0 { i32 805306496, i32 0, i32 0, i32 0 }, align 4
54  @@@rps_Str4" = private unnamed_addr constant [5 x i8] c"data\00"
55  @@@rps_ParamAttr5" = private constant %0 { i32 0, i32 0, i32 27, i32 0 }, align 4
56  @@@rps_ParamDescArray6" = private constant [2 x %struct.RpsParameterDesc] [%struct.RpsParameterDesc { %struct.RpsTypeInfo { i16 36, i16
57  @@@rps_Str7" = private unnamed_addr constant [9 x i8] c"Triangle\00"
58  @@@rps_Str8" = private unnamed_addr constant [13 x i8] c"renderTarget\00"
59  @@@rps_ParamAttr9" = private constant %0 { i32 128, i32 0, i32 35, i32 0 }, align 4
60  @@@rps_ParamDescArray10" = private constant [1 x %struct.RpsParameterDesc] [%struct.RpsParameterDesc { %struct.RpsTypeInfo { i16 36, i16
61  @@@rps_Str11" = private unnamed_addr constant [8 x i8] c"Upscale\00"
62  @@@rps_Str12" = private unnamed_addr constant [5 x i8] c"dest\00"
63  @@@rps_ParamAttr13" = private constant %0 { i32 128, i32 0, i32 35, i32 0 }, align 4
64  @@@rps_Str14" = private unnamed_addr constant [7 x i8] c"source\00"
65  @@@rps_ParamAttr15" = private constant %0 { i32 16, i32 2, i32 0, i32 0 }, align 4
66  @@@rps_ParamDescArray16" = private constant [2 x %struct.RpsParameterDesc] [%struct.RpsParameterDesc { %struct.RpsTypeInfo { i16 36, i16
67  @NodeDecls_hello_triangle = dlllexport constant [3 x %struct.RpsNodeDesc] [%struct.RpsNodeDesc { i32 1, i32 2, %struct.RpsParameterDesc*
```

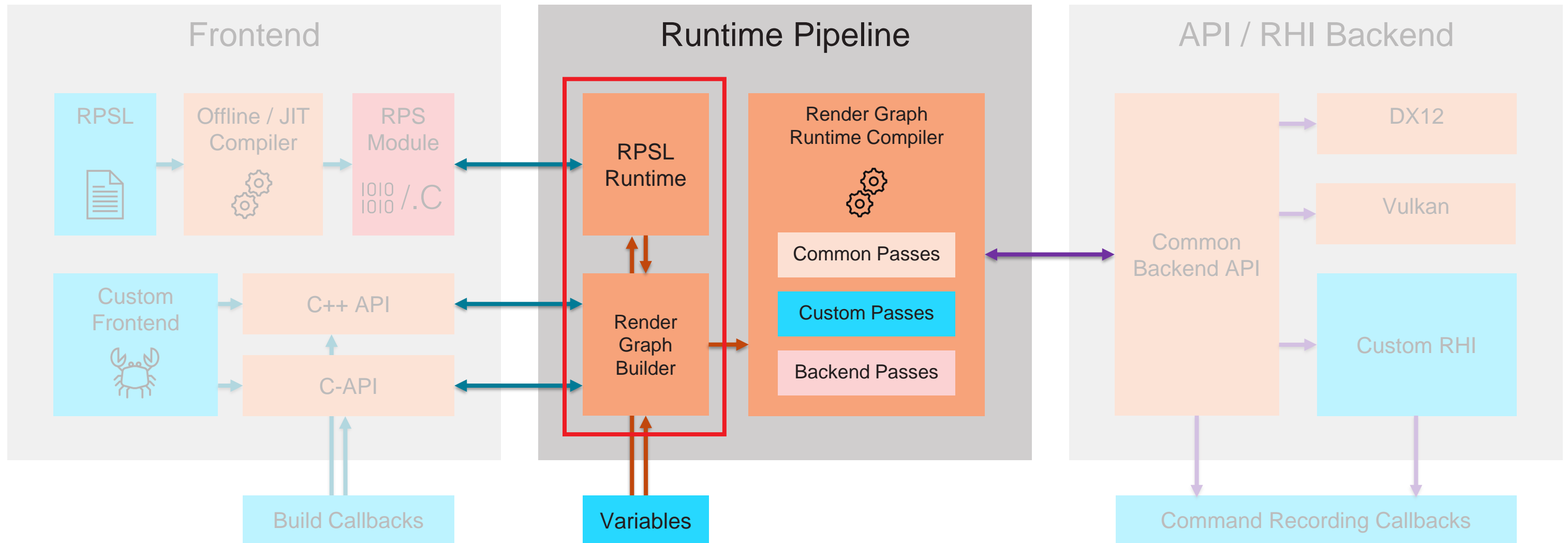
# THE RUNTIME



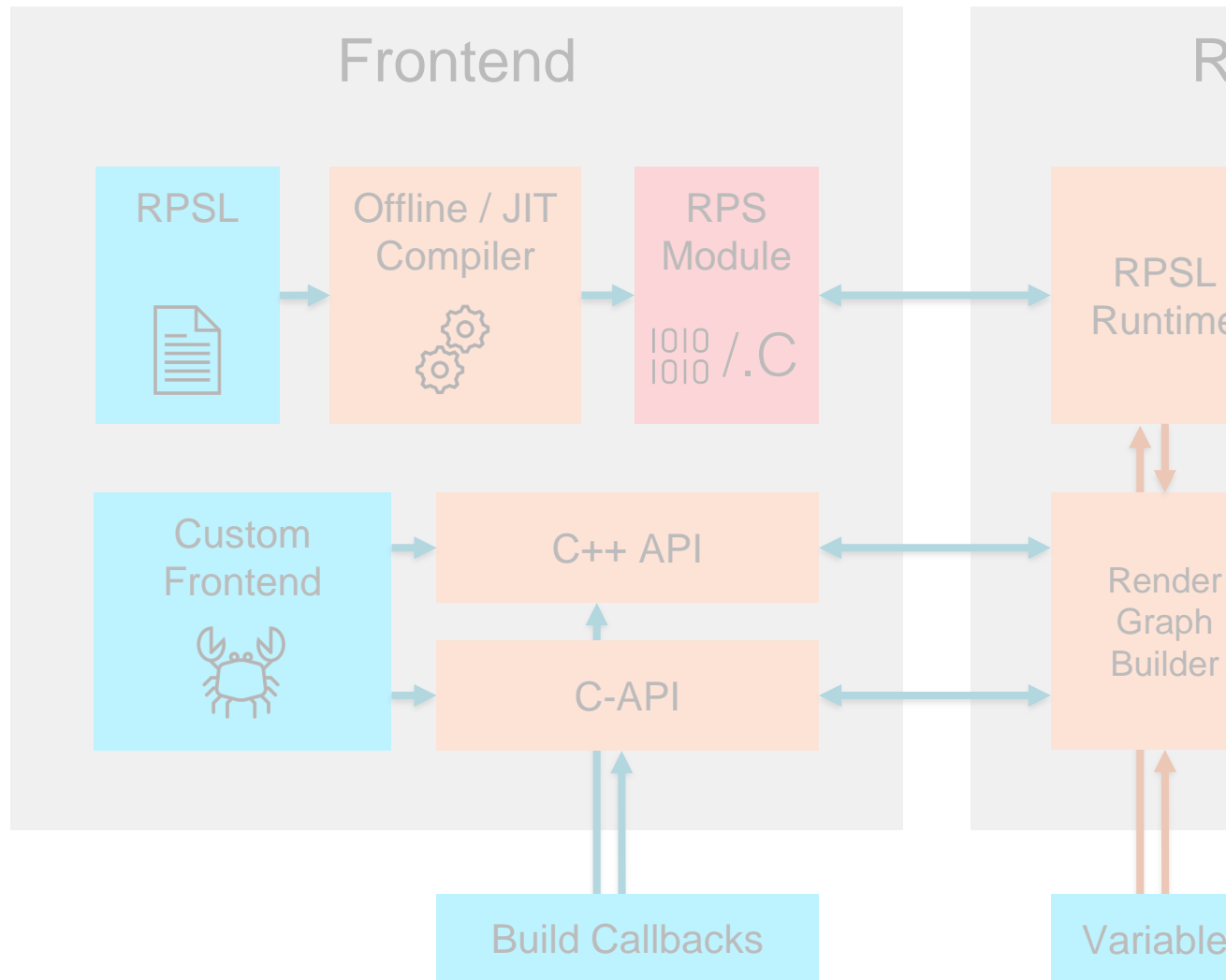


```
Cauldron.rpsl X
1 #define RPSL_SHADER
2 #include "RenderSettings.h"
3 #include "RenderUtil.h"
4 #include "../libs/cauldron/src/common/rps/RpsLib.h"
5
6 export void CauldronFrame([readonly(present)] texture backBuffer, FrameConstants settings)
7 {
8     texture shadowAtlas = null;
9     texture shadowMask = null;
10
11     texture backBufferRtv = backBuffer.format( settings.backBufferFormat );
12
13     texture depthBuffer = create_tex2d(settings.depthFormat, settings.width, settings.height);
14     texture colorBuffer = create_tex2d(settings.colourFormat, settings.width, settings.height);
15     texture motionVectorBuffer = null;
16
17     if(settings.bUseMotionVectors)
18     {
19         motionVectorBuffer = create_tex2d(settings.motionVectorsFormat, settings.width, settings.height);
20         clear(motionVectorBuffer, float4(0.0f, 0.0f, 0.0f, 0.0f));
21     }
22
23     // Always clear depth
24     clear_depth( depthBuffer, 1.0f );
25
26     // Clear color if no skybox
27     if( settings.skyboxType > 1 )
28     {
29         clear(colorBuffer, float4(0.0f.xxx, 1.0f));
30     }
31
32     node sortGeoNode = SortGeometry();
33
34     // ZPrePass
35
RenderSettings.h X
1 #pragma once
2
3 enum SkyBoxType
4 {
5     SkyBoxType_Procedural = 0,
6     SkyboxType_Cubemap,
7     SkyboxType_Clear,
8 };
9
10 struct FrameConstants
11 {
12     RpsBool bUseZPrePass;
13     RpsBool bUseMotionVectors;
14     RpsBool bDrawGeometry;
15
16     RpsBool bDrawBoundingBoxes;
17     RpsBool bDrawLightFrustums;
18
19     RpsBool bUseShadowMask;
20     RpsBool bUseBloom;
21     RpsBool bUseTAA;
22     RpsBool bTakeScreenshot;
23
24     SkyBoxType skyboxType;
25     int numShadowedLights;
26     int maxBloomMipLevels;
27     int maxDownsamples;
28
29     uint32_t shadowMapWidth;
30     uint32_t shadowMapHeight;
31     RpsFormat shadowDepthAtlasFormat;
32     RpsFormat shadowMaskFormat;
33
34     uint32_t width;
35     uint32_t height;
```

# THE RUNTIME



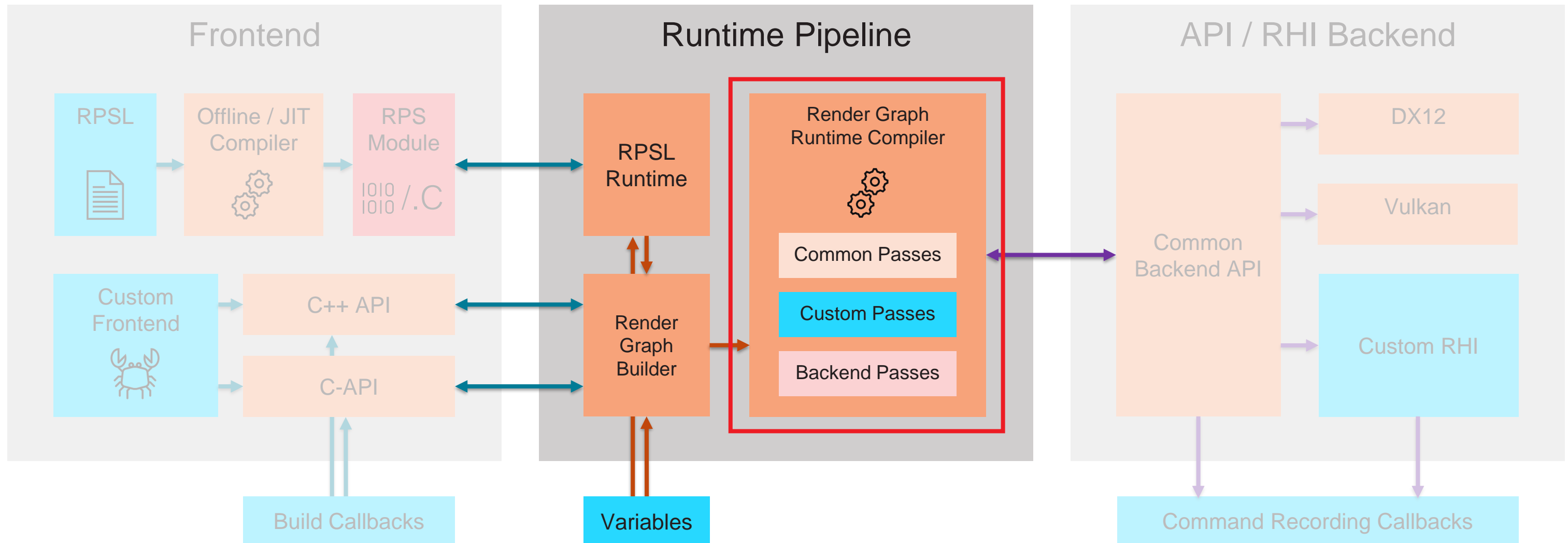
# THE RUNTIME



```
Resource Declarations:
-- 0 : backBuffer
-- type : tex2D( 1280 x 720 ), -fmt : R8G8B8A8_UNORM, -array : 1, -mip : 1, -samples : 1, -flags(NONE)
-- 1 : depthBuffer
-- type : tex2D( 1280 x 720 ), -fmt : D32_FLOAT, -array : 1, -mip : 1, -samples : 1
-- 2 : colorBuffer
-- type : tex2D( 1280 x 720 ), -fmt : R11G11B10_FLOAT, -array : 1, -mip : 1, -samples : 1
-- 3 : downSampleOutput
-- type : tex2D( 640 x 360 ), -fmt : R11G11B10_FLOAT, -array : 1, -mip : 5, -samples : 1
-- 4 : blurTemp
-- type : tex2D( 640 x 360 ), -fmt : R11G11B10_FLOAT, -array : 1, -mip : 5, -samples : 1
-- 5 : motionVectorBuffer
-- type : tex2D( 1280 x 720 ), -fmt : R16G16_FLOAT, -array : 1, -mip : 1, -samples : 1
-- 6 : shadowAtlas
-- type : tex2D( 2048 x 2048 ), -fmt : D32_FLOAT, -array : 1, -mip : 1, -samples : 1
-- 7 : TAABuffer
-- type : tex2D( 1280 x 720 ), -fmt : R11G11B10_FLOAT, -array : 1, -mip : 1, -samples : 1
-- 8 : historyBuffer
-- type : tex2D( 1280 x 720 ), -fmt : R11G11B10_FLOAT, -array : 1, -mip : 1, -samples : 1, -flags(PERSISTENTNONE)
-- 9 : shadowMask
-- type : tex2D( 1280 x 720 ), -fmt : R8G8B8A8_UNORM, -array : 1, -mip : 1, -samples : 1

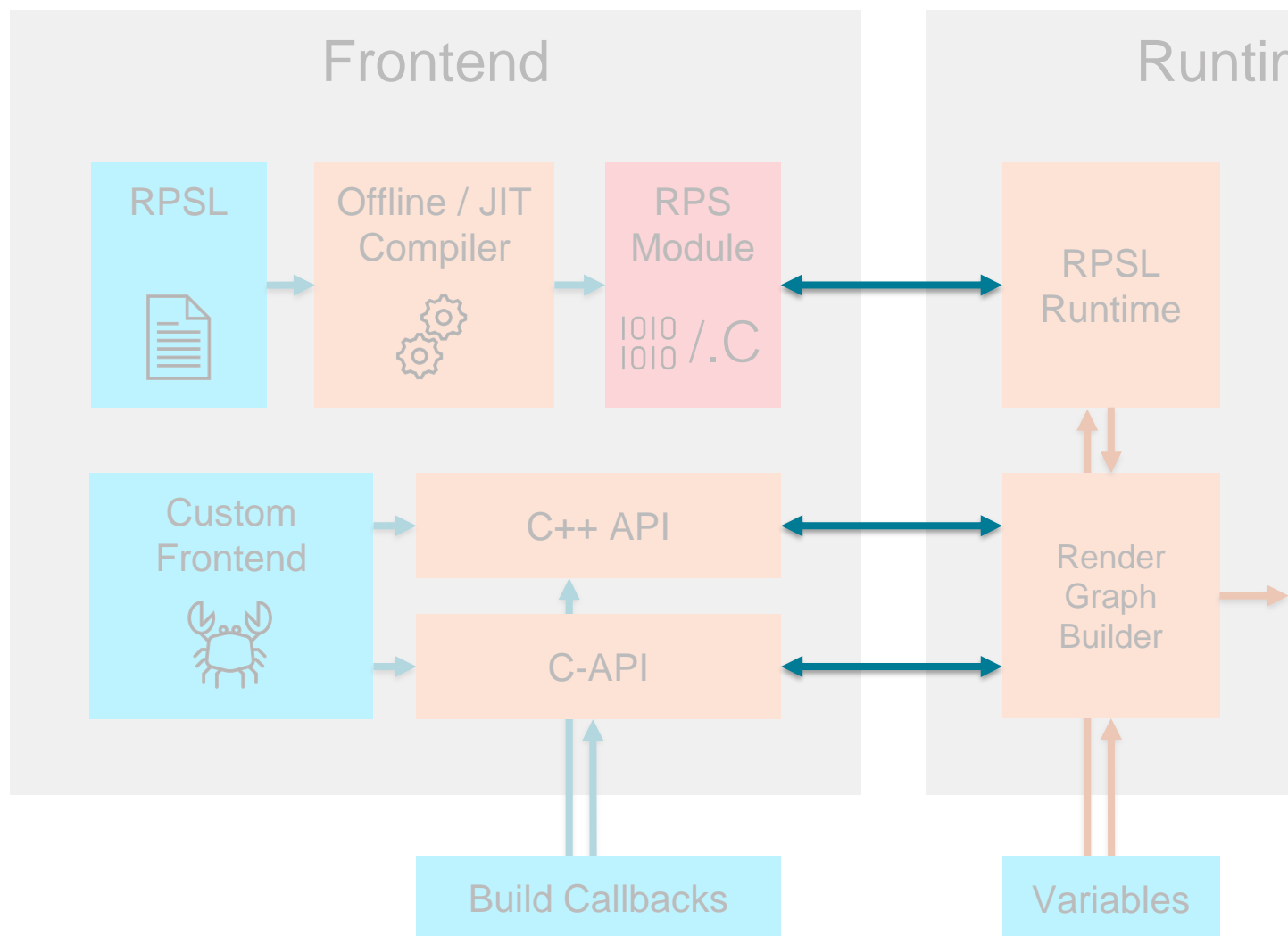
Commands:
-- 0 : clear_color_0( |
-- | t [motionVectorBuffer : (color, clear, discard)] )
-- 1 : clear_depth_stencil_1(
-- | t [depthBuffer : (depth_write, stencil_write, clear, discard)] )
-- 2 : SortGeometry_2( )
-- 3 : ZPrePass_3(
-- | gBuffDepth [depthBuffer : (depth_write, stencil_write)] : SV_DepthStencil[0] )
-- 4 : clear_depth_stencil_4(
-- | t [shadowAtlas : (depth_write, stencil_write, clear, discard)] )
-- 5 : Shadows_5(
-- | shadowAtlas [shadowAtlas : (depth_write, stencil_write)] : SV_DepthStencil[0] )
-- 6 : ResolveShadows_6(
-- | shadowMap [shadowAtlas : (srv(cs))],
-- | depthBuffer [depthBuffer : (srv(cs))],
-- | shadowBuffer [shadowMask : (uav(ps, cs))] )
-- 7 : PBROpaqueForwardAfterZPrePass_7(
-- | gBuffColor [colorBuffer : (color, render_pass)] : SV_Target[0],
-- | gBuffMotionVectors [motionVectorBuffer : (color, render_pass)] : SV_Target[1],
-- | gBuffDepth [depthBuffer : (depth_read, stencil_write)] : SV_DepthStencil[0],
-- | shadowAtlas [shadowMask : (srv(ps))] )
```

# THE RUNTIME



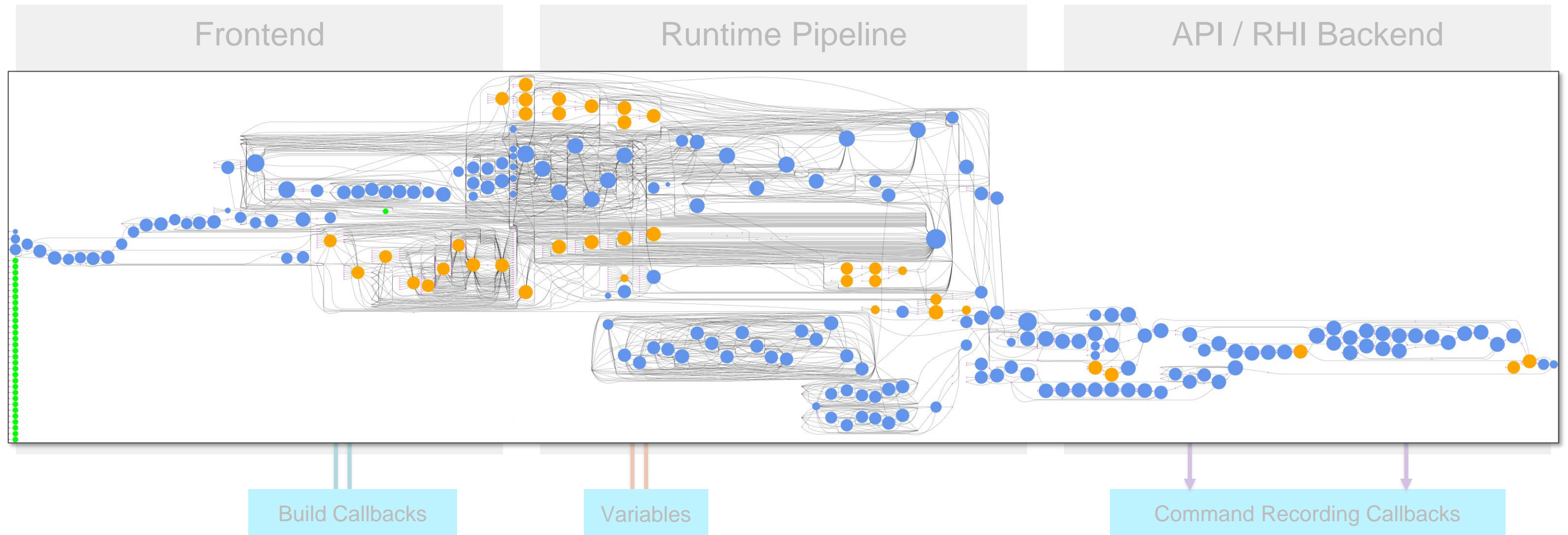


# THE RUNTIME



```
digraph G {
  graph [ size = "128,64" ];
  edge [ style = bold ];
  node [ shape = polygon, sides = 4, color = magenta, style = filled, orientation = "45.0" ];
  t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
  t_9 t_10 t_11 t_12 t_13 t_14 t_15 t_16
  t_17 t_18 t_19 t_20 t_21 t_22 t_23 t_24
  t_25 t_26 t_27 t_28 t_29 t_30 t_31 t_32
  t_33 t_34 t_35 t_36 t_37 t_38 t_39 t_40
  t_41 t_42 t_43 t_44 t_45 t_46 t_47 t_48
  t_49 t_50 t_51 t_52 t_53 t_54 t_55 t_56
  t_57 t_58 t_59;
  node [ shape = circle, color = cyan, style = filled ];
  clear_color_0 clear_depth_stencil_1 SortGeometry_2 ZPrePass_3 clear_depth_stencil_4 Shadows_5 PBROp
  SkyboxDeferred_8 PBRTransparentForward_9 Downsample_10 Downsample_11 Downsample_12 Downsamp
  BlurVertical_16 Upscale_17 BlurHorizontal_18 BlurVertical_19 Upscale_20 BlurHorizontal_21 B
  BlurHorizontal_24 BlurVertical_25 Upscale_26 BlurHorizontal_27 BlurVertical_28 Upscale_29 To
  ;
  node [ shape = circle, color = orange, style = filled ];
  ResolveShadows_6 TAA_30 Sharpen_31;
  node [ shape = circle, color = lime, style = filled ];
  node [ shape = circle, color = gray, style = filled ];
  t_1 -> clear_color_0;
  t_2 -> clear_depth_stencil_1;
  clear_depth_stencil_1 -> ZPrePass_3;
  t_2 -> ZPrePass_3;
  t_3 -> clear_depth_stencil_4;
  clear_depth_stencil_4 -> Shadows_5;
  t_3 -> Shadows_5;
  t_4 -> ResolveShadows_6;
  t_5 -> ResolveShadows_6;
  t_6 -> ResolveShadows_6;
  SortGeometry_2 -> PBROpaqueForwardAfterZPrePass_7;
  t_7 -> PBROpaqueForwardAfterZPrePass_7;
  clear_color_0 -> PBROpaqueForwardAfterZPrePass_7;
  t_1 -> PBROpaqueForwardAfterZPrePass_7;
  t_8 -> PBROpaqueForwardAfterZPrePass_7;
  t_9 -> PBROpaqueForwardAfterZPrePass_7;
  PBROpaqueForwardAfterZPrePass_7 -> SkyboxDeferred_8;
  t_7 -> SkyboxDeferred_8;
  PBROpaqueForwardAfterZPrePass_7 -> SkyboxDeferred_8;
  t_8 -> SkyboxDeferred_8;
```

# THE RUNTIME



# SCHEDULING

- Scoring based on various factors:
  - Barrier batching, Memory usage & footprint, Program order, Queue Switches, Overlapping Opportunity, ...
- Customizable:
  - Global flags
  - Intrinsic & attributes
  - User defined passes

```
enum RpsSchedulerFlagBits
{
    RPS_SCHEDULER_FLAG_KEEP_PROGRAM_ORDER,
    RPS_SCHEDULER_FLAG_PREFER_BARRIER_BATCHING,
    RPS_SCHEDULER_FLAG_PREFER_MEMORY_SAVING,
    ...
};
```

```
[subgraph(sequential)]
void Foo()
{
    node1();
    node2();
};
```

# SCHEDULING

- Scoring based on various
  - Barrier batching, Memory Queue Switches, Overlap
- Customizable:
  - Global flags
  - Intrinsic & attributes
  - User defined passes

```
enum RpsSchedulerFlagBits
{
    RPS_SCHEDULER_FLAG_KEEP_PROGRAM_ORDER,
    RPS_SCHEDULER_FLAG_PREFER_BARRIER_BATCHING,
    RPS_SCHEDULER_FLAG_PREFER_MEMORY_SAVING,
    ...
};
```

```
Schedule:
Batch 0 Queue 0:
0 : <preamble>
1 : t_2 <depthBuffer> : (*) => (depth_write, stencil_write, clear, discard)
2 : t_3 <shadowAtlas> : (*) => (depth_write, stencil_write, clear, discard)
3 : clear_depth_stencil_1(
    t [depthBuffer : (depth_write, stencil_write, clear, discard)] )
4 : ZPrePass_3(
    gBuffDepth [depthBuffer : (depth_write, stencil_write)] : SV_DepthStencil[0] )
5 : clear_depth_stencil_4(
    t [shadowAtlas : (depth_write, stencil_write, clear, discard)] )
6 : Shadows_5(
    shadowAtlas [shadowAtlas : (depth_write, stencil_write)] : SV_DepthStencil[0] )
7 : t_5 <shadowAtlas> : (depth_write, stencil_write, clear, discard) => (srv(cs))
8 : t_6 <depthBuffer> : (depth_write, stencil_write, clear, discard) => (srv(cs))
9 : t_4 <shadowMask> : (*) => (uav(ps, cs))
10 : ResolveShadows_6(
    shadowMap [shadowAtlas : (srv(cs))],
    depthBuffer [depthBuffer : (srv(cs))],
    shadowBuffer [shadowMask : (uav(ps, cs))] )
11 : t_8 <depthBuffer> : (srv(cs)) => (depth_read, stencil_write)
12 : t_9 <shadowMask> : (uav(ps, cs)) => (srv(ps))
13 : t_1 <motionVectorBuffer> : (*) => (color, render_pass, clear, discard)
14 : t_7 <colorBuffer> : (*) => (color, render_pass)
15 : clear_color_0(
    t [motionVectorBuffer : (color, clear, discard)] )
16 : SortGeometry_2( )
17 : PBROpaqueForwardAfterZPrePass_7(
    gBuffColor [colorBuffer : (color, render_pass)] : SV_Target[0],
    gBuffMotionVectors [motionVectorBuffer : (color, render_pass)] : SV_Target[1],
    gBuffDepth [depthBuffer : (depth_read, stencil_write)] : SV_DepthStencil[0],
    shadowAtlas [shadowMask : (srv(ps))] )
18 : SkyboxDeferred_8(
    gBuffColor [colorBuffer : (color, render_pass)] : SV_Target[0],
    gBuffDepth [depthBuffer : (depth_read, stencil_write)] : SV_DepthStencil[0] )
19 : PBRTransparentForward_9(
    gBuffColor [colorBuffer : (color, render_pass)] : SV_Target[0],
    gBuffMotionVectors [motionVectorBuffer : (color, render_pass)] : SV_Target[1],
    gBuffDepth [depthBuffer : (depth_read, stencil_write)] : SV_DepthStencil[0],
    shadowAtlas [shadowMask : (srv(ps))],
    gBufferDepthSrv [depthBuffer : (srv(ps))] )
20 : t_11 <colorBuffer> : (color, render_pass) => (srv(ps))
```



# SCHEDULING

- Scoring based on various
  - Barrier batching, Memory Queue Switches, Overlap
- Customizable:
  - Global flags
  - Intrinsic & attributes
  - User defined passes

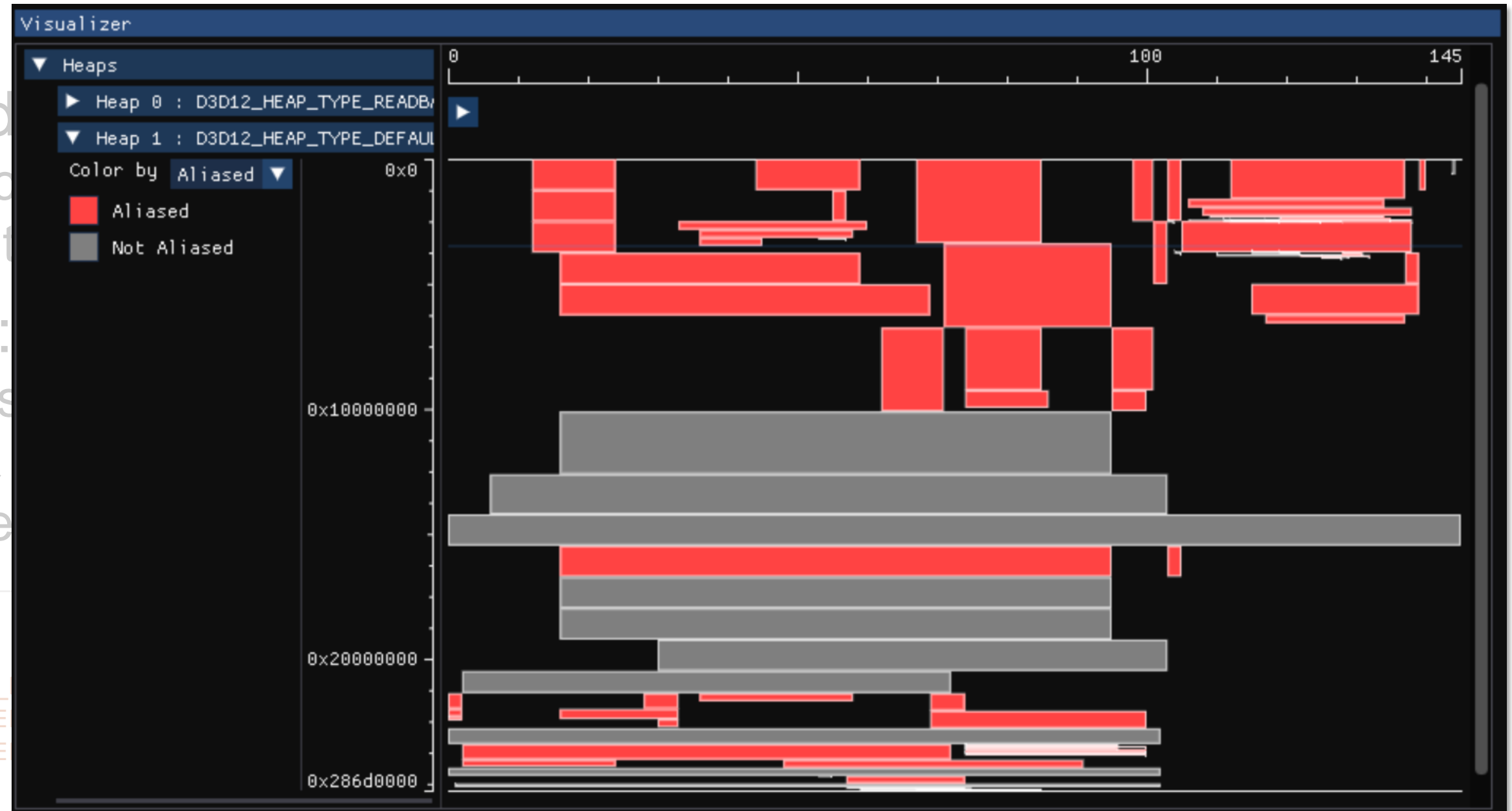
```
enum RpsSchedulerFlagBits
{
    RPS_SCHEDULER_FLAG_KEEP_PROGRAM_ORDER,
    RPS_SCHEDULER_FLAG_PREFER_BARRIER_BATCHING,
    RPS_SCHEDULER_FLAG_PREFER_MEMORY_SAVING,
    ...
};
```

```
Schedule:
Batch 0 Queue 0:
0 : <preamble>
1 : t_2 <depthBuffer> : (*) => (depth_write, stencil_write, clear, discard)
2 : t_3 <shadowAtlas> : (*) => (depth_write, stencil_write, clear, discard)
3 : clear_depth_stencil_1(
    t [depthBuffer : (depth_write, stencil_write, clear, discard)] )
4 : ZPrePass_3(
    gBuffDepth [depthBuffer : (depth_write, stencil_write)] : SV_DepthStencil[0] )
5 : clear_depth_stencil_4(
    t [shadowAtlas : (depth_write, stencil_write, clear, discard)] )
6 : Shadows_5(
    shadowAtlas [shadowAtlas : (depth_write, stencil_write)] : SV_DepthStencil[0] )
7 : t_5 <shadowAtlas> : (depth_write, stencil_write, clear, discard) => (srv(cs))
8 : t_6 <depthBuffer> : (depth_write, stencil_write, clear, discard) => (srv(cs))
9 : t_4 <shadowMask> : (*) => (uav(ps, cs))
10 : ResolveShadows_6(
    shadowMap [shadowAtlas : (srv(cs))],
    depthBuffer [depthBuffer : (srv(cs))],
    shadowBuffer [shadowMask : (uav(ps, cs))] )
11 : t_8 <depthBuffer> : (srv(cs)) => (depth_read, stencil_write)
12 : t_9 <shadowMask> : (uav(ps, cs)) => (srv(ps))
13 : t_1 <motionVectorBuffer> : (*) => (color, render_pass, clear, discard)
14 : t_7 <colorBuffer> : (*) => (color, render_pass)
15 : clear_color_0(
    t [motionVectorBuffer : (color, clear, discard)] )
16 : SortGeometry_2( )
17 : PBROpaqueForwardAfterZPrePass_7(
    gBuffColor [colorBuffer : (color, render_pass)] : SV_Target[0],
    gBuffMotionVectors [motionVectorBuffer : (color, render_pass)] : SV_Target[1],
    gBuffDepth [depthBuffer : (depth_read, stencil_write)] : SV_DepthStencil[0],
    shadowAtlas [shadowMask : (srv(ps))] )
18 : SkyboxDeferred_8(
    gBuffColor [colorBuffer : (color, render_pass)] : SV_Target[0],
    gBuffDepth [depthBuffer : (depth_read, stencil_write)] : SV_DepthStencil[0] )
19 : PBRTransparentForward_9(
    gBuffColor [colorBuffer : (color, render_pass)] : SV_Target[0],
    gBuffMotionVectors [motionVectorBuffer : (color, render_pass)] : SV_Target[1],
    gBuffDepth [depthBuffer : (depth_read, stencil_write)] : SV_DepthStencil[0],
    shadowAtlas [shadowMask : (srv(ps))],
    gBufferDepthSrv [depthBuffer : (srv(ps))] )
20 : t_11 <colorBuffer> : (color, render_pass) => (srv(ps))
```

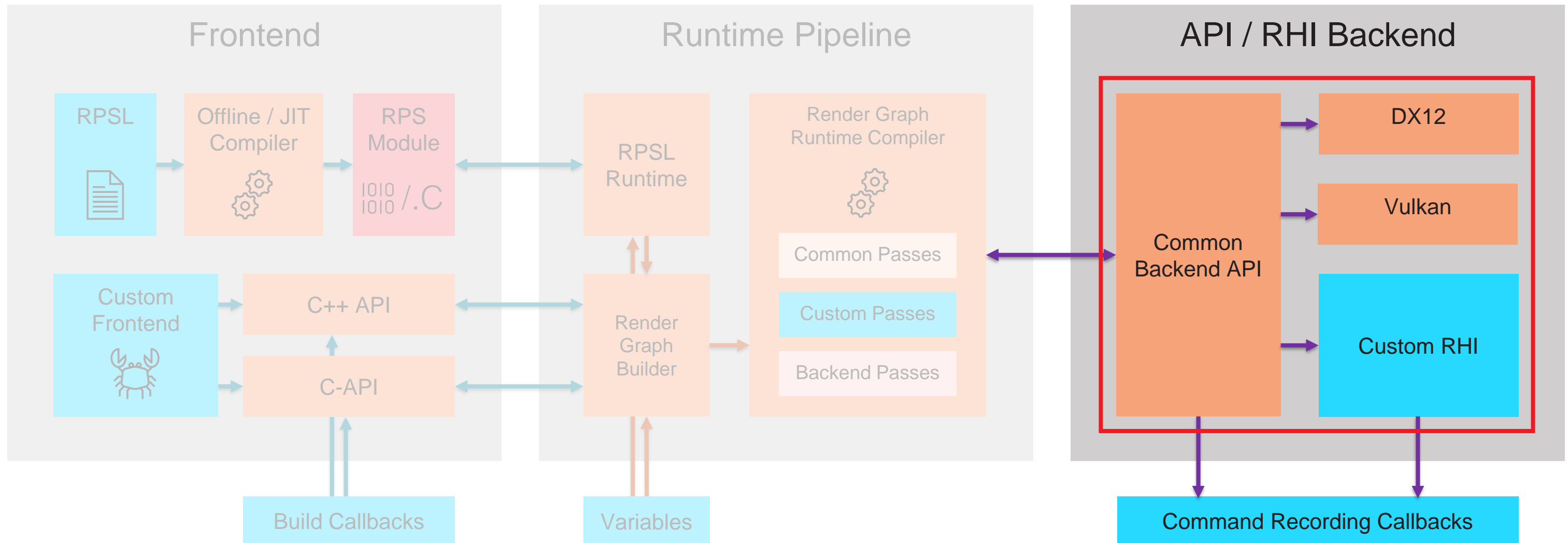
# SCHEDULING

- Scoring based
  - Barrier batch
  - Queue Switch
- Customizable:
  - Global flags
  - Intrinsic &
  - User define

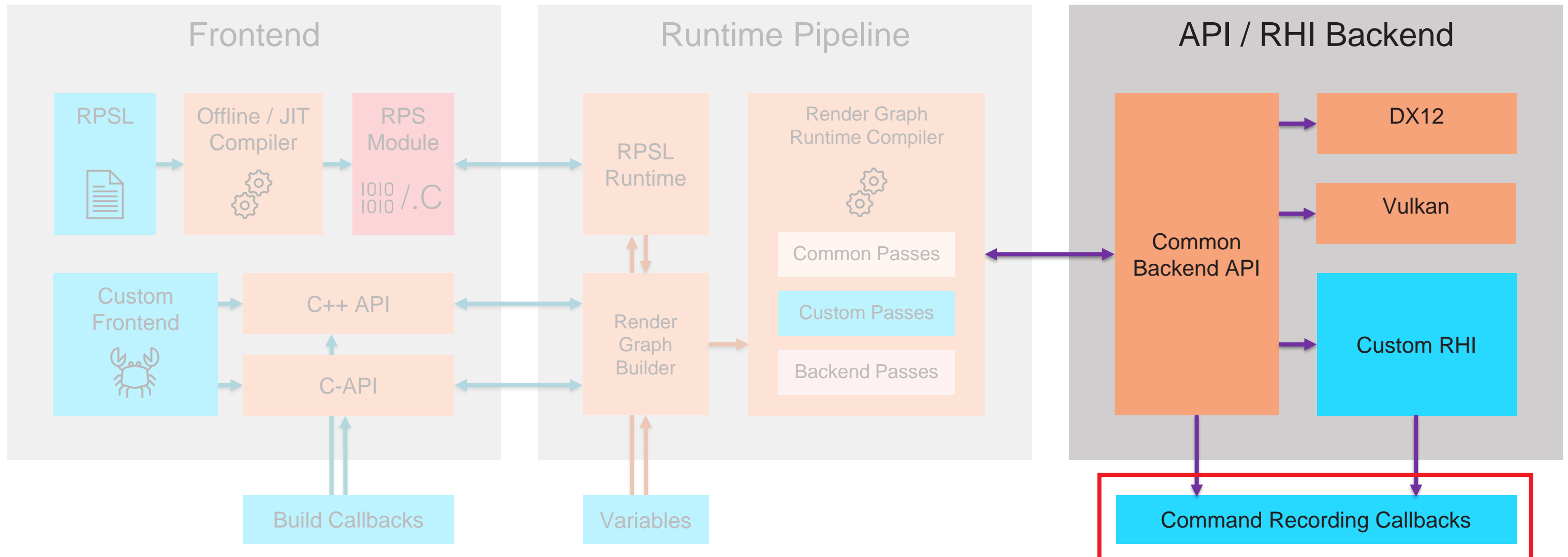
```
enum RpsSchedulerFlagBits
{
    RPS_SCHEDULER_FLAG_KEEP_
    RPS_SCHEDULER_FLAG_PREFERE
    RPS_SCHEDULER_FLAG_PREFERE
    ...
};
```



# THE BACKEND



# THE BACKEND





# HELLO RPS CALLBACKS

```
// Sample RPSL code
node Upscale ([readwrite(renderTarget)] texture dest : SV_Target0,
             [readonly(ps)] texture source);
```

```
// Sample C++ code
// Command Recording Callback
void Upscale(const RpsCmdCallbackContext* pContext, RpsUnusedArg dest, D3D12_CPU_DESCRIPTOR_HANDLE source)
{
    ID3D12GraphicsCommandList* pCmdList = rpsD3D12GraphicsCommandListFromHandle(pContext->hCommandBuffer);

    pCmdList->SetGraphicsRootSignature(m_upscaleRootSignature.Get());
    pCmdList->SetPipelineState(m_upscalePSO.Get());

    D3D12_GPU_DESCRIPTOR_HANDLE srvTable = CopyToShaderVisibleDescriptorRing(
        D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV, &offscreenRTSrv, 1);

    BindDescriptorHeaps(pCmdList);
    pCmdList->SetGraphicsRootDescriptorTable(0, srvTable);
    pCmdList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
    pCmdList->DrawInstanced(3, 1, 0, 0);
}
```

# HELLO RPS CALLBACKS

```
// Sample RPSL code
node Upscale ([readwrite(rendertarget)] texture dest : SV_Target0,
             [readonly(ps)] texture source);
```

```
// Sample C++ code
// Command Recording Callback
void Upscale(const RpsCmdCallbackContext* pContext, RpsUnusedArg dest, D3D12_CPU_DESCRIPTOR_HANDLE source)
{
    ID3D12GraphicsCommandList* pCmdList = rpsD3D12GraphicsCommandListFromHandle(pContext->hCommandBuffer);

    pCmdList->SetGraphicsRootSignature(m_upscaleRootSignature.Get());
    pCmdList->SetPipelineState(m_upscalePSO.Get());

    D3D12_GPU_DESCRIPTOR_HANDLE srvTable = CopyToShaderVisibleDescriptorRing(
        D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV, &offscreenRTSrv, 1);

    BindDescriptorHeaps(pCmdList);
    pCmdList->SetGraphicsRootDescriptorTable(0, srvTable);
    pCmdList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
    pCmdList->DrawInstanced(3, 1, 0, 0);
}
```

# HELLO RPS CALLBACKS

```
// Sample RPSL code
node Upscale ([readwrite(rendertarget)] texture dest : SV_Target0,
             [readonly(ps)] texture source);
```

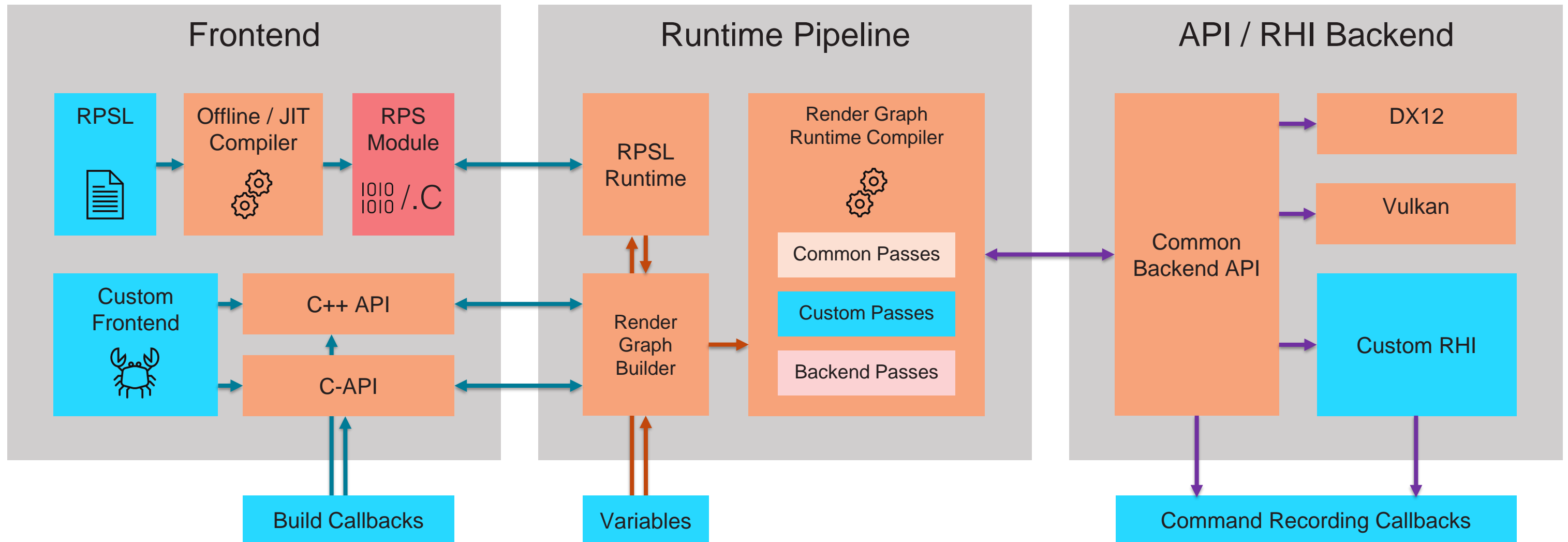
```
// Sample C++ code
// Command Recording Callback
void Upscale(const RpsCmdCallbackContext* pContext, RpsUnusedArg dest, D3D12_CPU_DESCRIPTOR_HANDLE source)
{
    ID3D12GraphicsCommandList* pCmdList = rpsD3D12GraphicsCommandListFromHandle(pContext->hCommandBuffer);

    pCmdList->SetGraphicsRootSignature(m_upscaleRootSignature.Get());
    pCmdList->SetPipelineState(m_upscalePSO.Get());

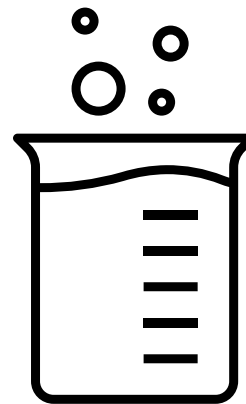
    D3D12_GPU_DESCRIPTOR_HANDLE srvTable = CopyToShaderVisibleDescriptorRing(
        D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV, &offscreenRTSrv, 1);

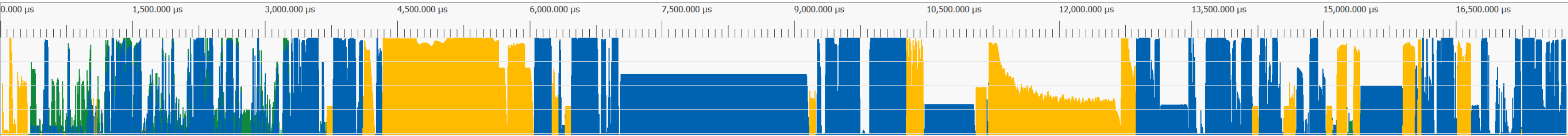
    BindDescriptorHeaps(pCmdList);
    pCmdList->SetGraphicsRootDescriptorTable(0, srvTable);
    pCmdList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
    pCmdList->DrawInstanced(3, 1, 0, 0);
}
```

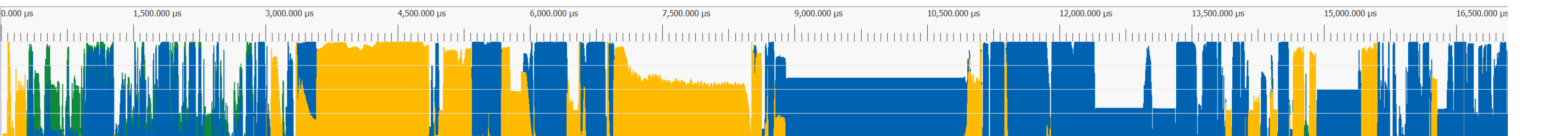
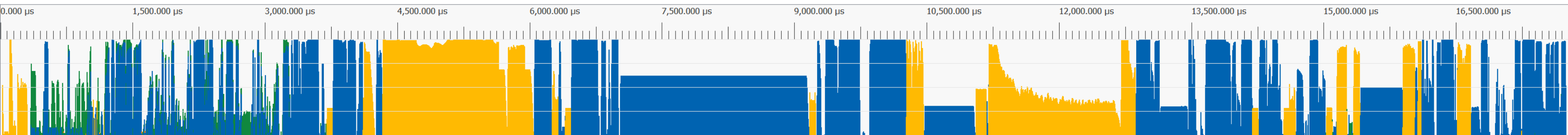
# THE SDK

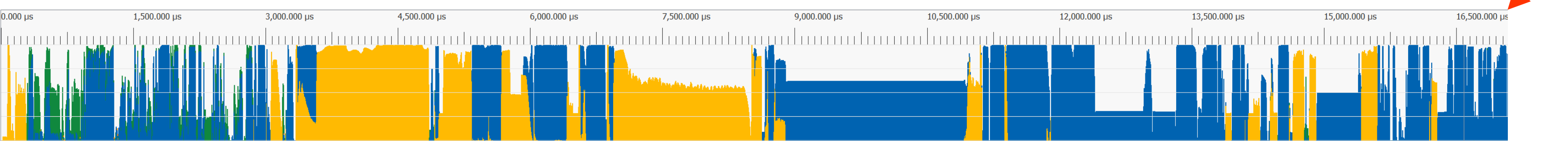
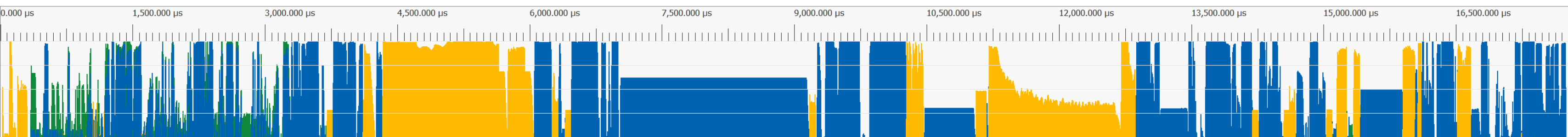


# RESULTS WITH GAME TRACES

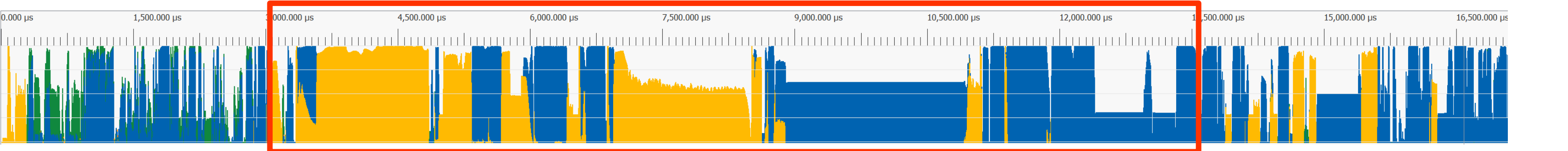
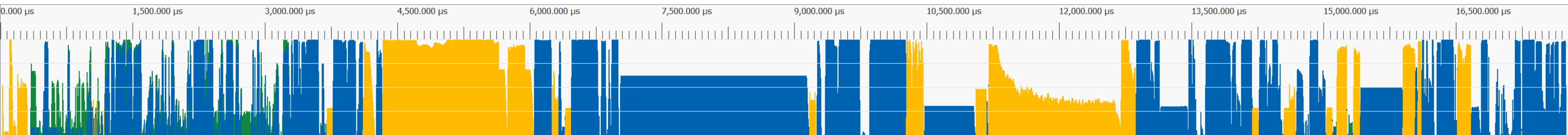


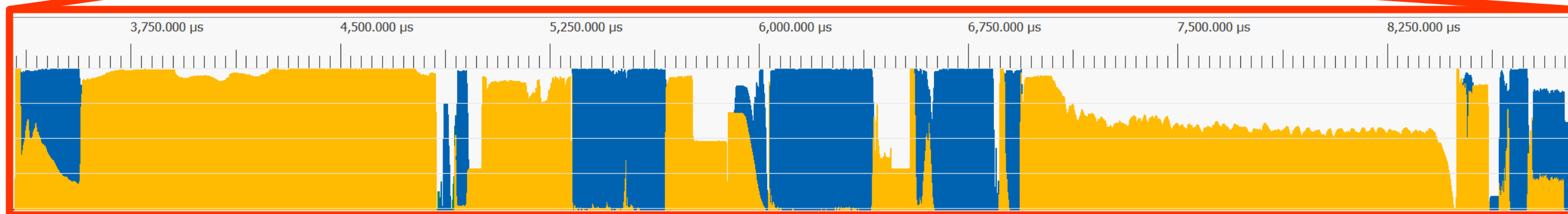
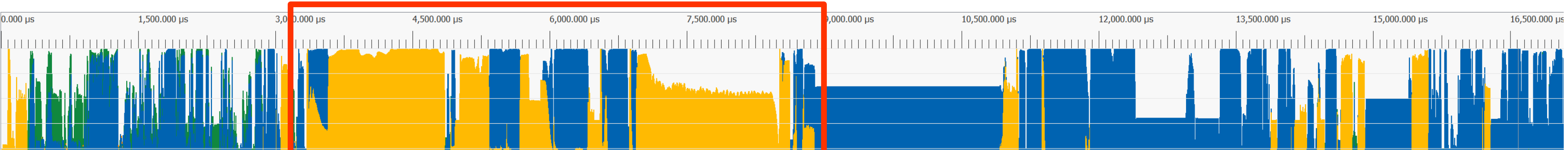
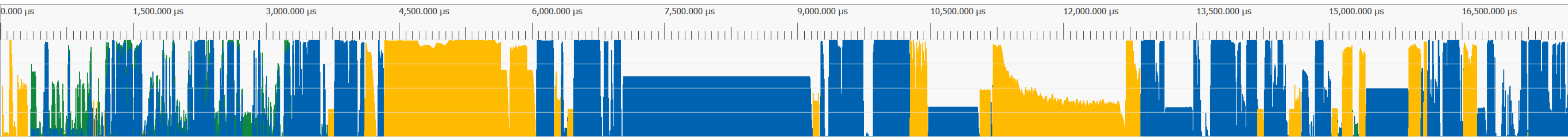


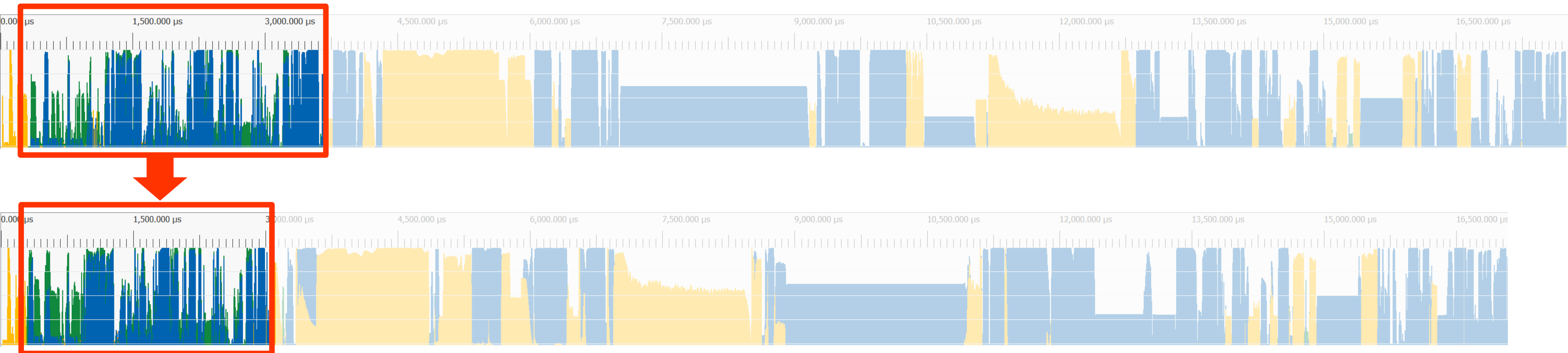




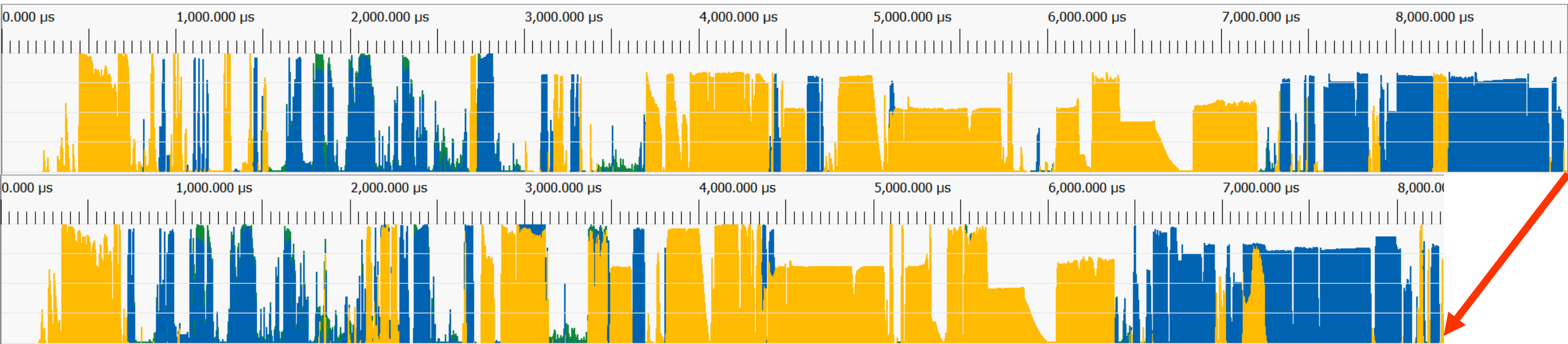


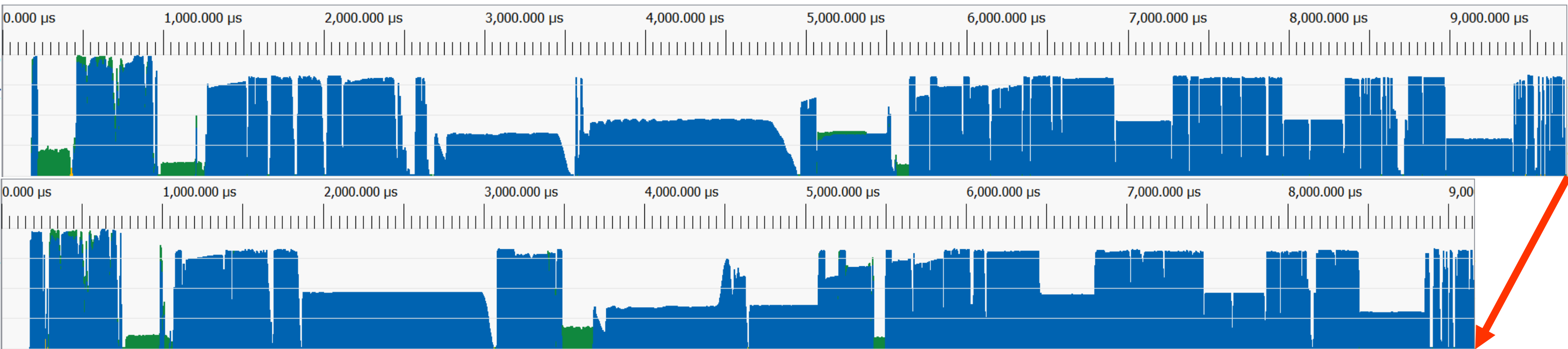






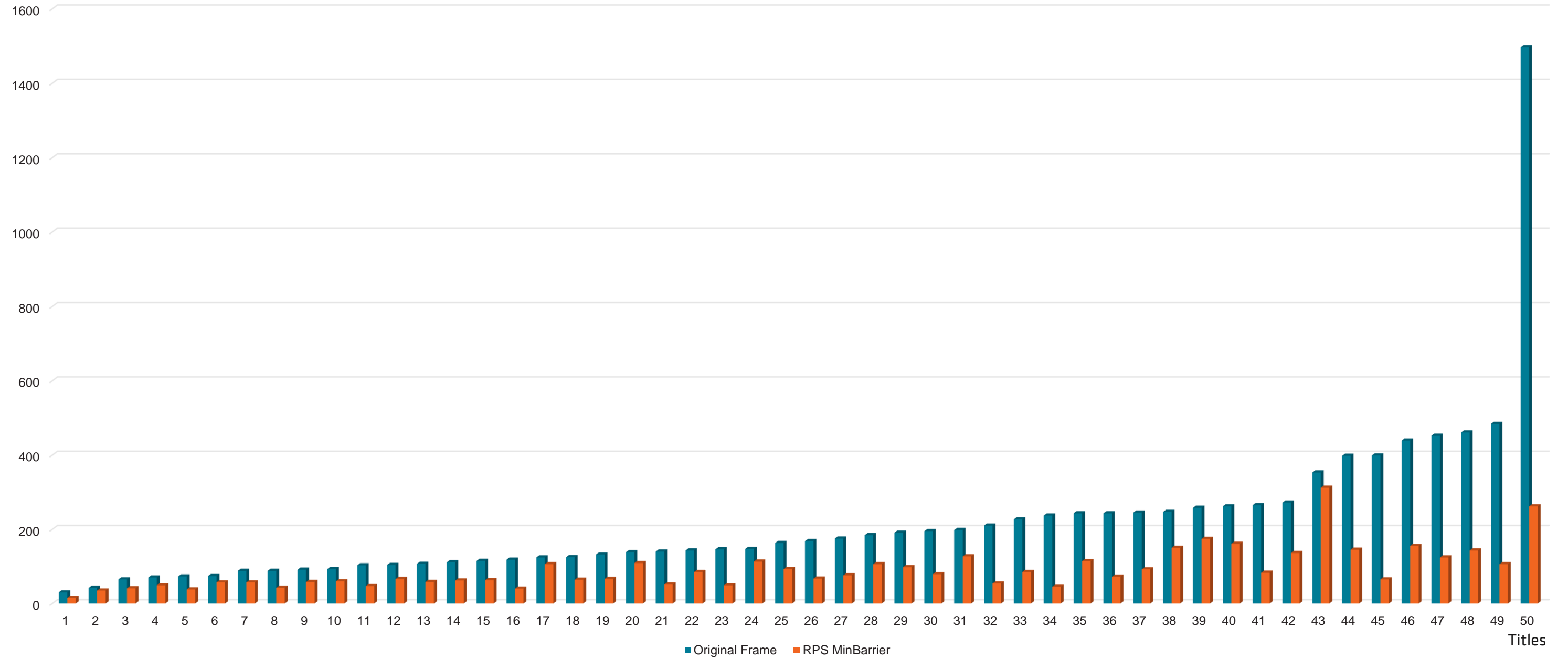






Barrier API Calls

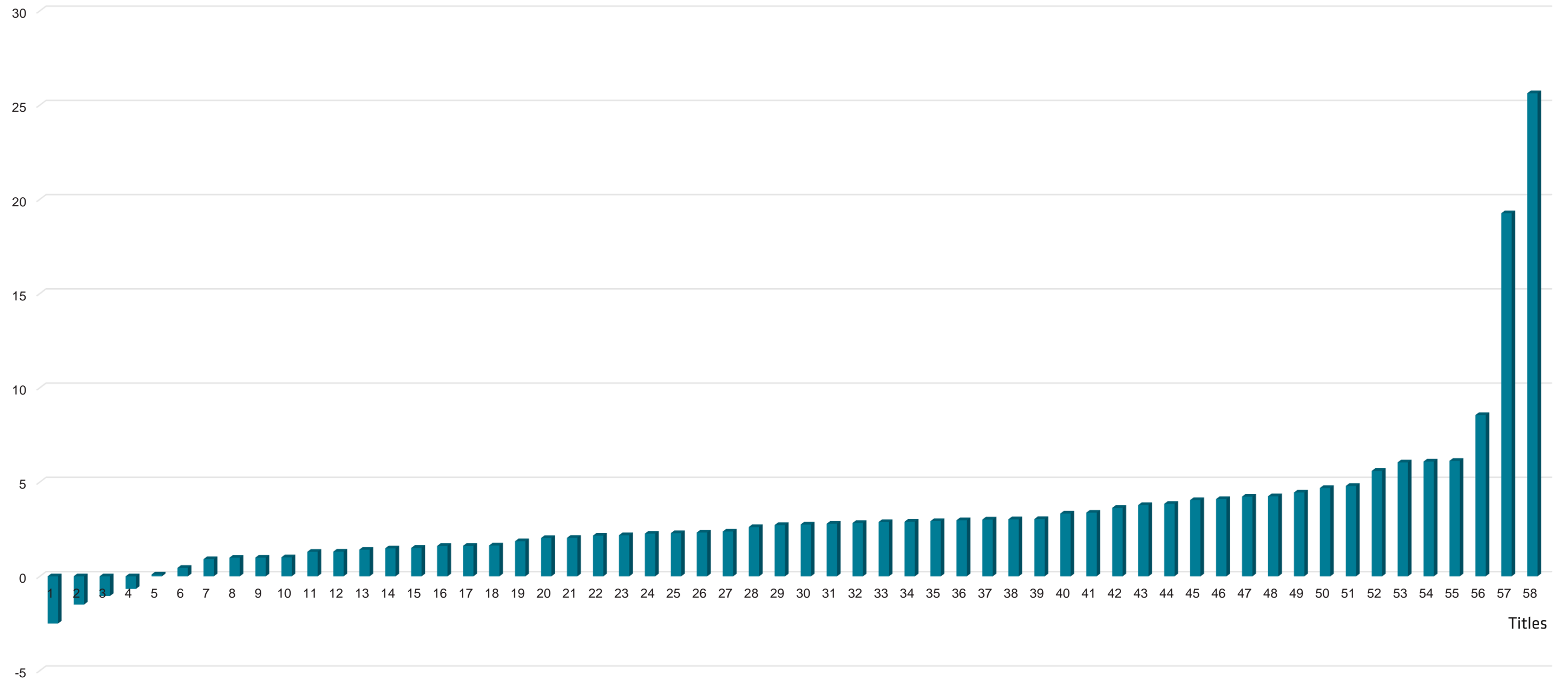
### Number of ResourceBarrier API calls



Source: AMD Engineering using RenderDoc Captures + RPS Replay Tool, as of Dec 2022.

GPU Time Reduction %

### RPS Frame Time Reduction % (Radeon RX 7900XTX)

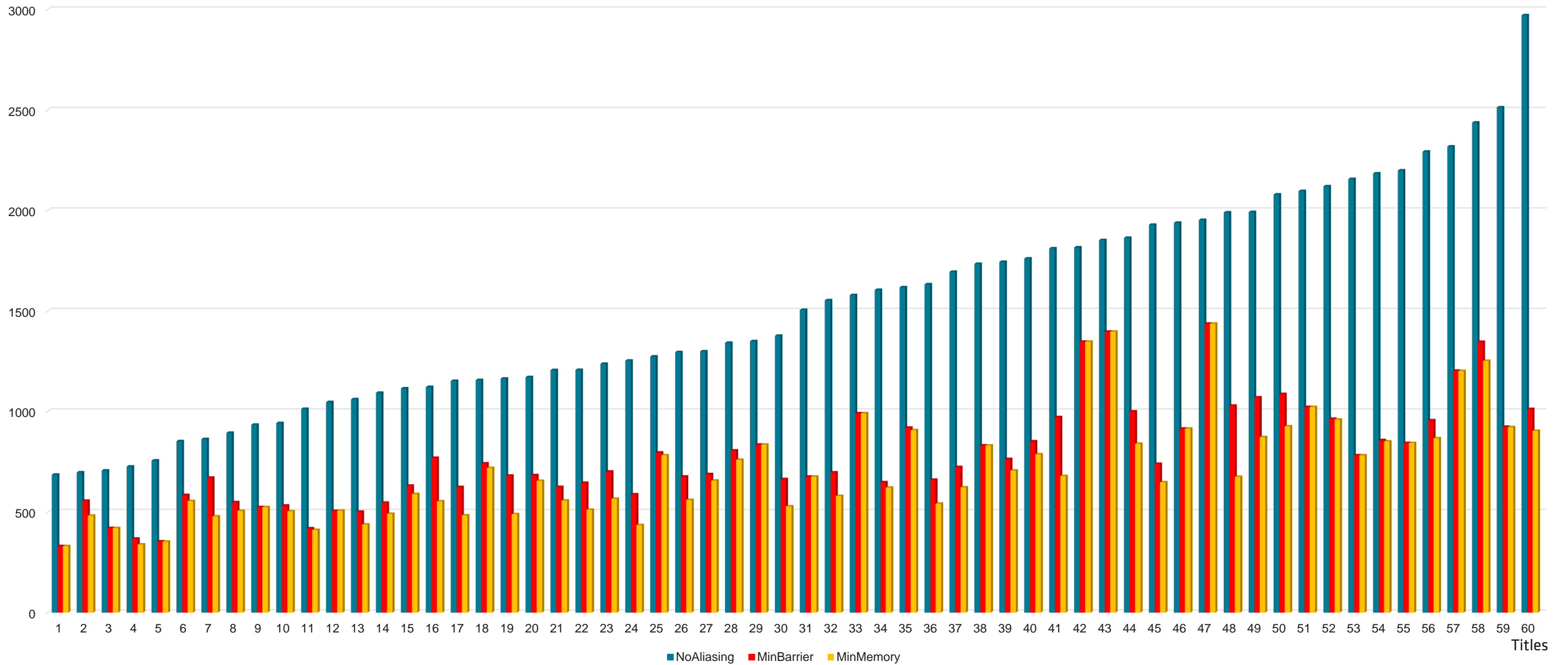


Source: AMD Engineering using RenderDoc Captures + RPS Replay Tool, as of Dec 2022.



Local Video Memory Usage (MiB)

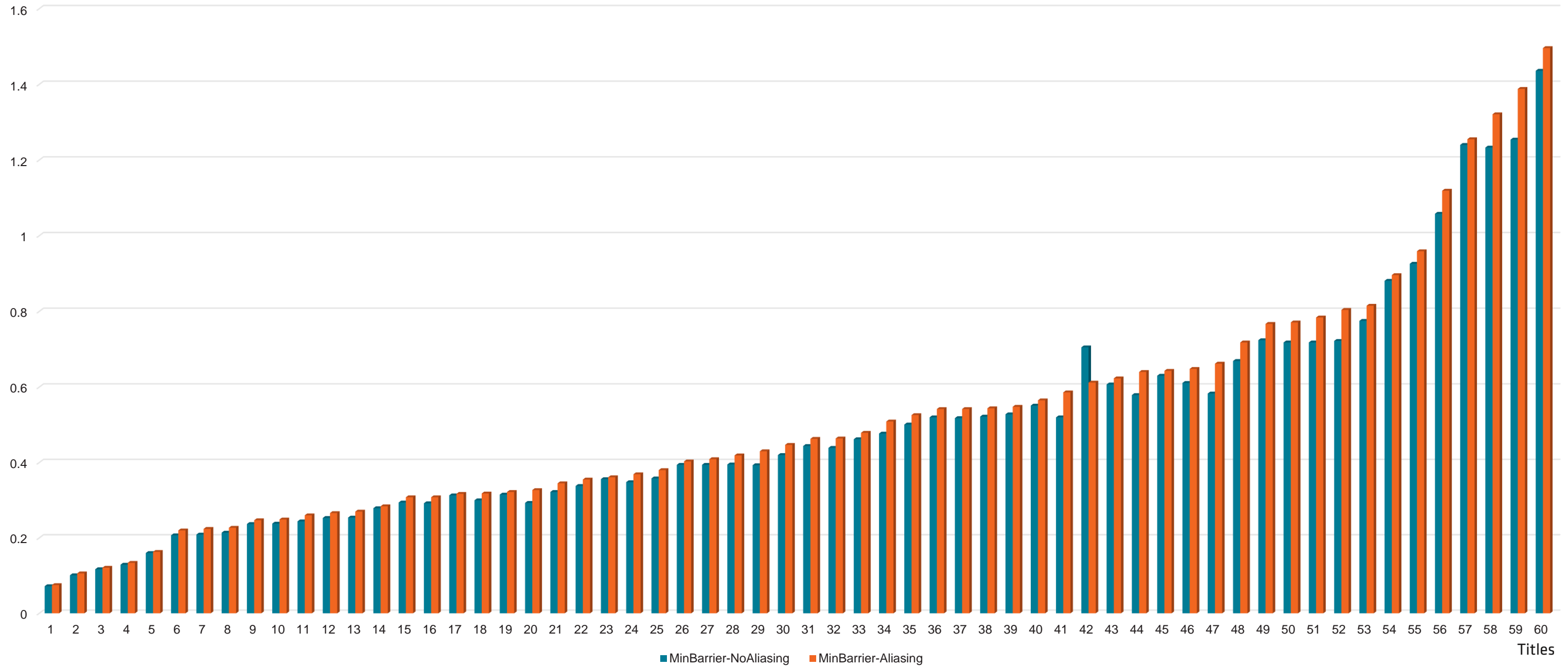
GPU Memory Usage (MiB)



Source: AMD Engineering using RenderDoc Captures + RPS Replay Tool, as of Dec 2022.

CPU time (ms)

### Frame Update CPU Time (Ryzen 9 3900, Single thread, ms)



Source: AMD Engineering using RenderDoc Captures + RPS Replay Tool, as of Dec 2022.

# RECAP & WHAT'S NEXT

- The RPS SDK today
  - A comprehensive & extensible Render Graph solution
  - Helps simplifying explicit API usage
    - Render Graph construction
    - Resource & Descriptor creation
    - Resource Barriers
    - Memory Aliasing
    - Single / Multi-Queue scheduling
    - Non-shader resource binding
    - ...

# RECAP & WHAT'S NEXT

- What's next
  - Areas of improvement
    - Tooling
    - More caching
    - Scheduling algorithms
    - Shader resource bindings
  - GPU shader interoperability
  - Your feedback! 😊

# THANKS

Steven Tovey

Dave Oldcorn

Florian Herick

Noah Cabral

Gareth Thomas

Jordan Logan

Justin Fyfe

Nicolas Thibieroz

Mike Smith

Rys Sommefeldt

David Ziman

Chas Boyd

Rosanna Ashworth-jones

Agustin Palacio

All our ISV partners who provided invaluable feedback!

# REFERENCES

- [0] Dave Oldcorn et al. “Right On Queue,” GDC 2016
- [1] Yuriy O’Donnell, “Frame Graph: Extensible Rendering Architecture in Frostbite,” GDC 2017
- [2] Tiago Rodrigues, “Moving To DirectX 12: Lessons Learned,” GDC 2017
- [3] Matthäus Chajdas, “D3D12 & Vulkan: Lessons Learned,” GDC 2017
- [4] Nicolas Guillemot, “Design Patterns for Low-Level Real-Time Rendering,” CppCon 2017
- [5] Pavlo Muratov, “Organizing GPU Work with Directed Acyclic Graphs,” [levelup.gitconnected.com](https://levelup.gitconnected.com), 2020, <https://levelup.gitconnected.com/organizing-gpu-work-with-directed-acyclic-graphs-f3fd5f2c2af3> (Accessed Feb 1, 2022)
- [6] Pavlo Muratov, “GPU Memory Aliasing,” [levelup.gitconnected.com](https://levelup.gitconnected.com), 2020, <https://levelup.gitconnected.com/gpu-memory-aliasing-45933681a15e> (Accessed Feb 1, 2022)

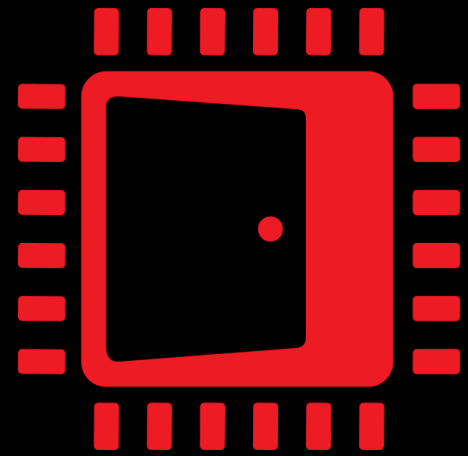
## DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. AMD, the AMD Arrow logo, [insert all other AMD trademarks used in the material here per AMD's Guidelines on Using Trademark Notice and Attribution] and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

OpenGL® and the oval logo are trademarks or registered trademarks of Hewlett Packard Enterprise in the United States and/or other countries worldwide. DirectX is a registered trademark of Microsoft Corporation in the US and other jurisdictions. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

© 2022 Advanced Micro Devices, Inc. All rights reserved.



AMD   
GPU Open

**AMD**   
together we advance\_

**AMD**   
EPYC

**AMD**   
RYZEN

**AMD**   
RADEON