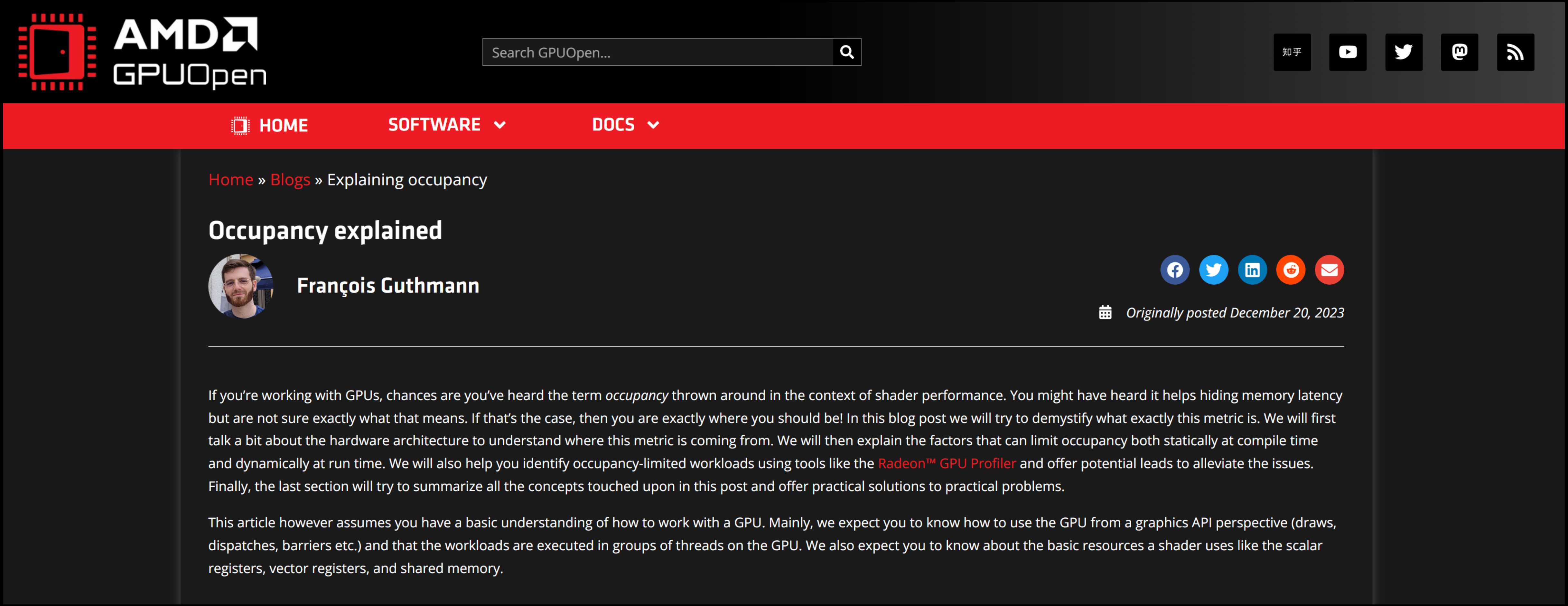
Occupancy explained through the AMD RDNA™ architecture

François Guthmann

Graphics Programming Conference

GPUOPEN



Home » Blogs » Explaining occupancy

Occupancy explained

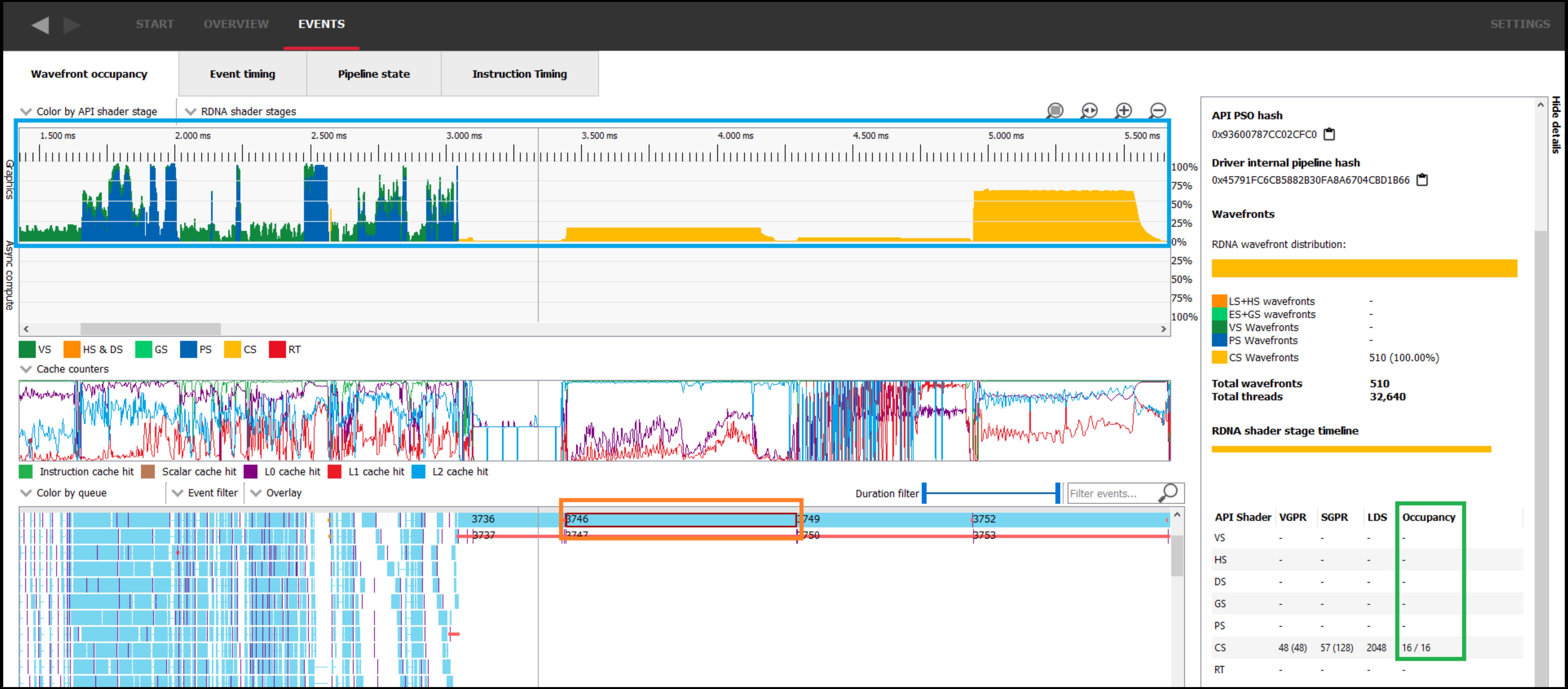
François Guthmann

Originally posted December 20, 2023

If you're working with GPUs, chances are you've heard the term *occupancy* thrown around in the context of shader performance. You might have heard it helps hiding memory latency but are not sure exactly what that means. If that's the case, then you are exactly where you should be! In this blog post we will try to demystify what exactly this metric is. We will first talk a bit about the hardware architecture to understand where this metric is coming from. We will then explain the factors that can limit occupancy both statically at compile time and dynamically at run time. We will also help you identify occupancy-limited workloads using tools like the Radeon™ GPU Profiler and offer potential leads to alleviate the issues. Finally, the last section will try to summarize all the concepts touched upon in this post and offer practical solutions to practical problems.

This article however assumes you have a basic understanding of how to work with a GPU. Mainly, we expect you to know how to use the GPU from a graphics API perspective (draws, dispatches, barriers etc.) and that the workloads are executed in groups of threads on the GPU. We also expect you to know about the basic resources a shader uses like the scalar registers, vector registers, and shared memory.

OCCUPANCY?



API PSO hash

0x93600787CC02CFC0

Driver internal pipeline hash

0x45791FC6CB5882B30FA8A6704CBD1B66

Wavefronts

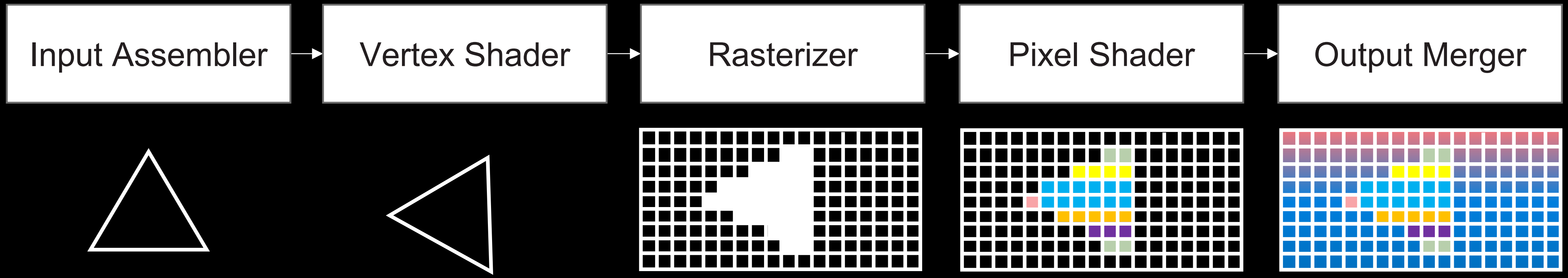
RDNA wavefront distribution:

🟧 LS+HS wavefronts	-
🟩 ES+GS wavefronts	-
🟩 VS Wavefronts	-
🟦 PS Wavefronts	-
🟨 CS Wavefronts	510 (100.00%)
Total wavefronts	**510**
Total threads	**32,640**

RDNA shader stage timeline

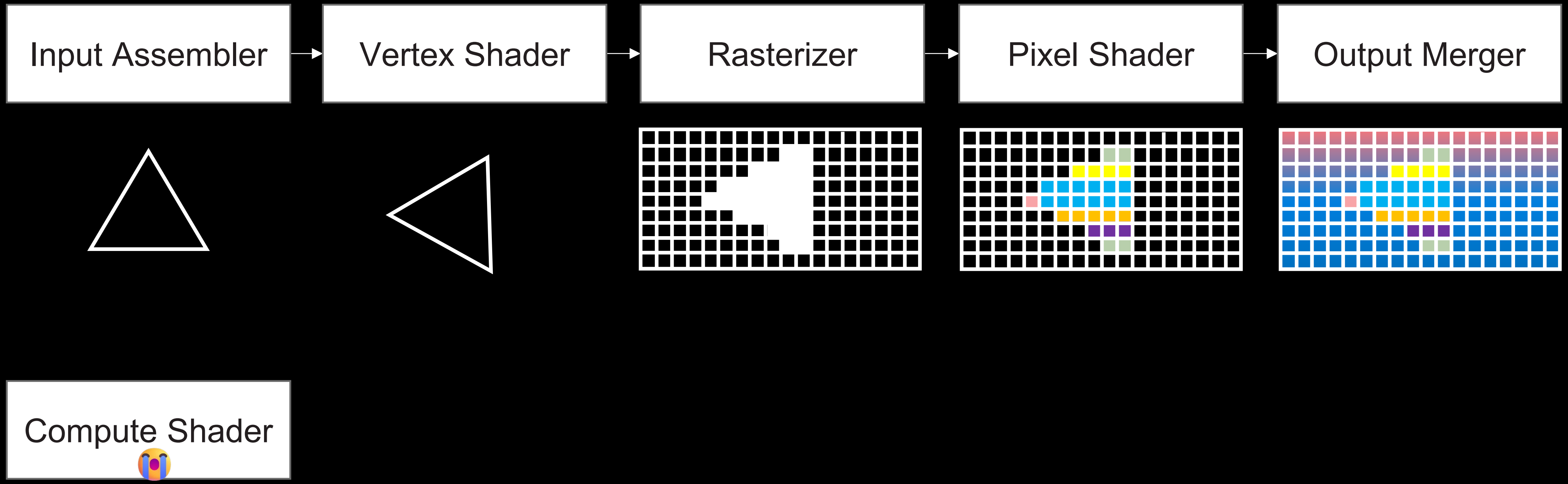
API Shader	VGPR	SGPR	LDS	Occupancy
VS	-	-	-	-
HS	-	-	-	-
DS	-	-	-	-
GS	-	-	-	-
PS	-	-	-	-
CS	48 (48)	57 (128)	2048	16 / 16
RT	-	-	-	-

LOGICAL GRAPHICS PIPELINE



Input Assembler → Vertex Shader → Rasterizer → Pixel Shader → Output Merger

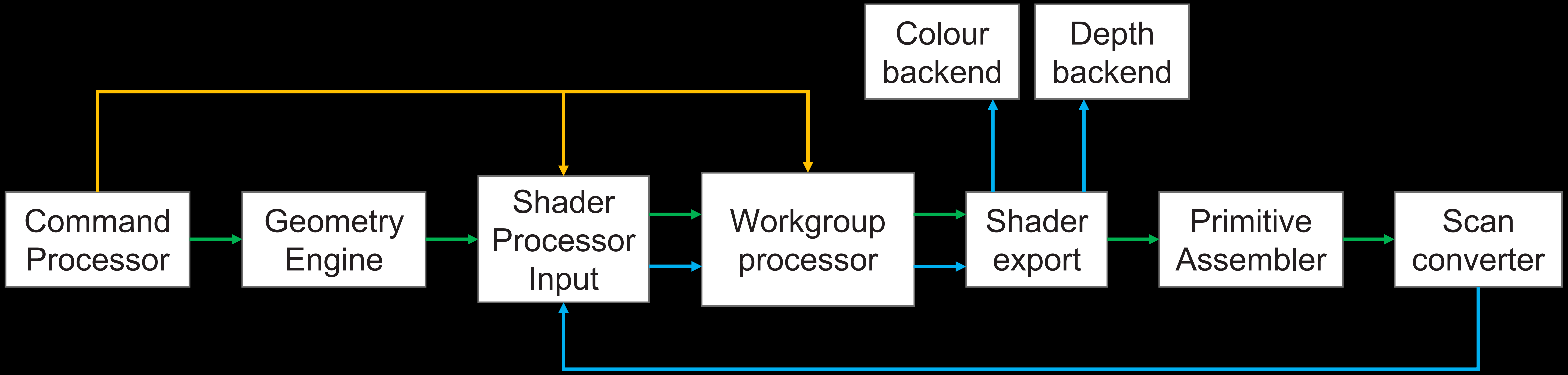
LOGICAL GRAPHICS PIPELINE



Input Assembler → Vertex Shader → Rasterizer → Pixel Shader → Output Merger

Compute Shader 😭

HARDWARE GRAPHICS PIPELINE

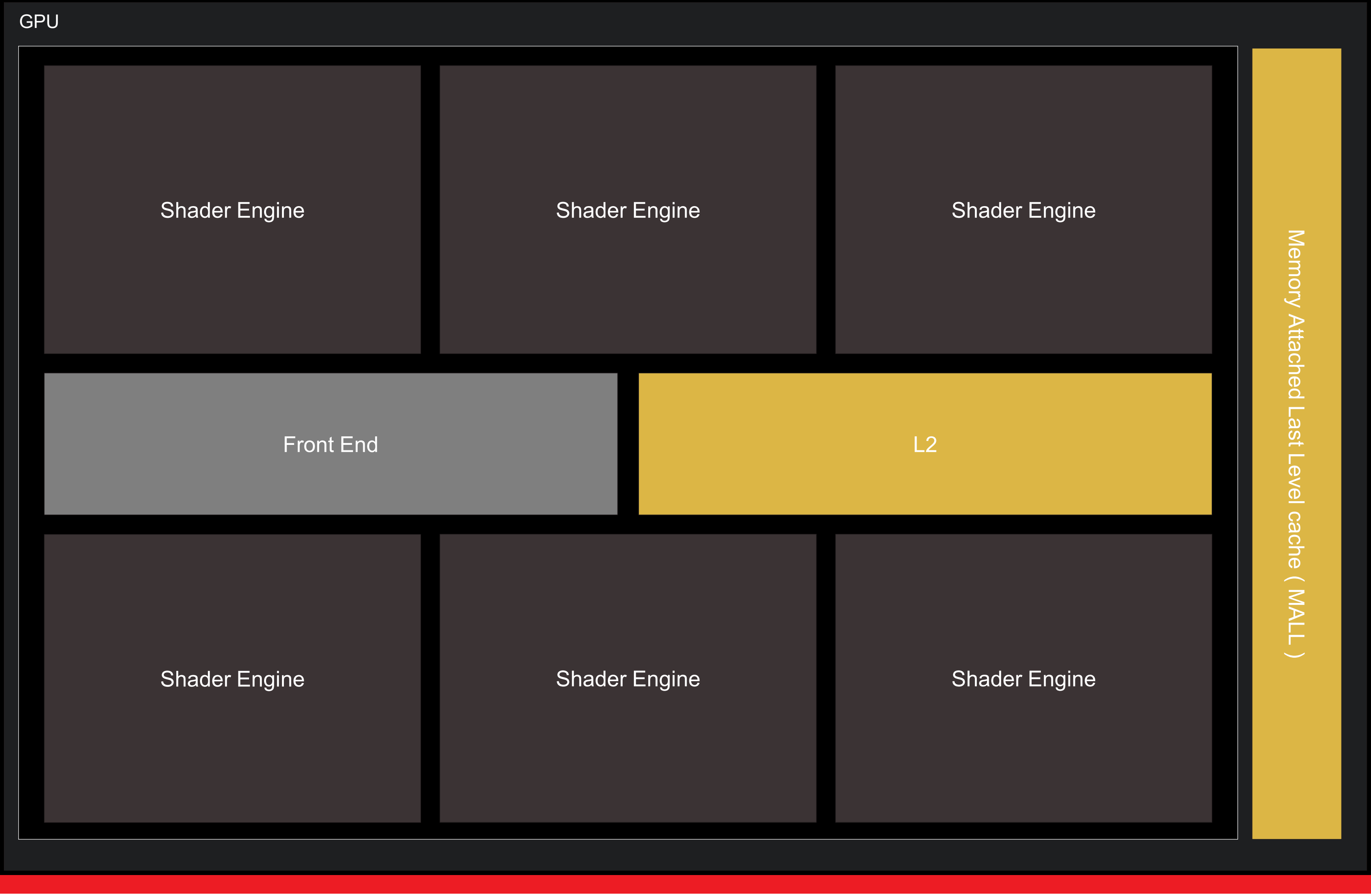


🟩 Vertex path

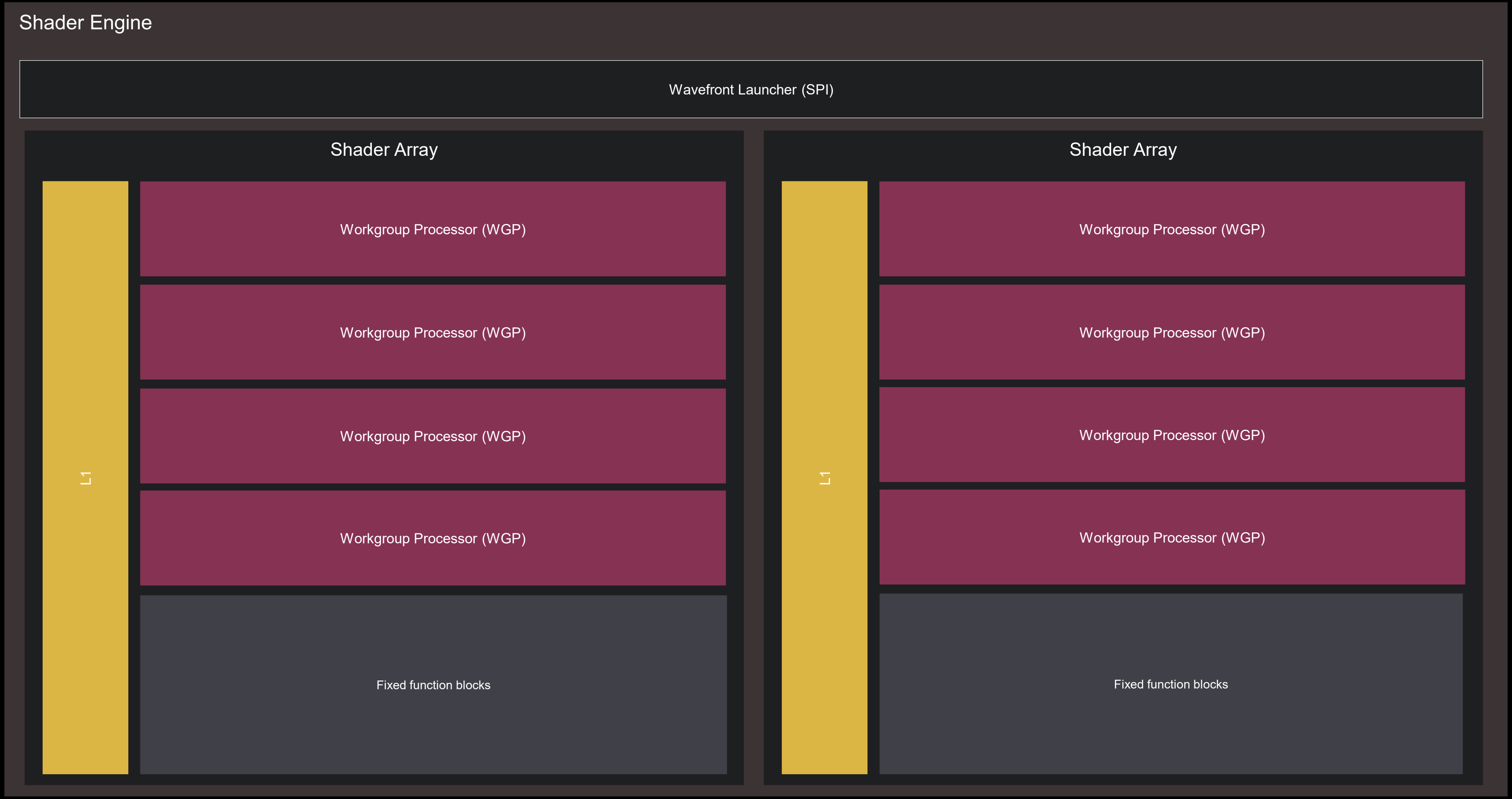
🟦 Fragment path

🟨 Compute path

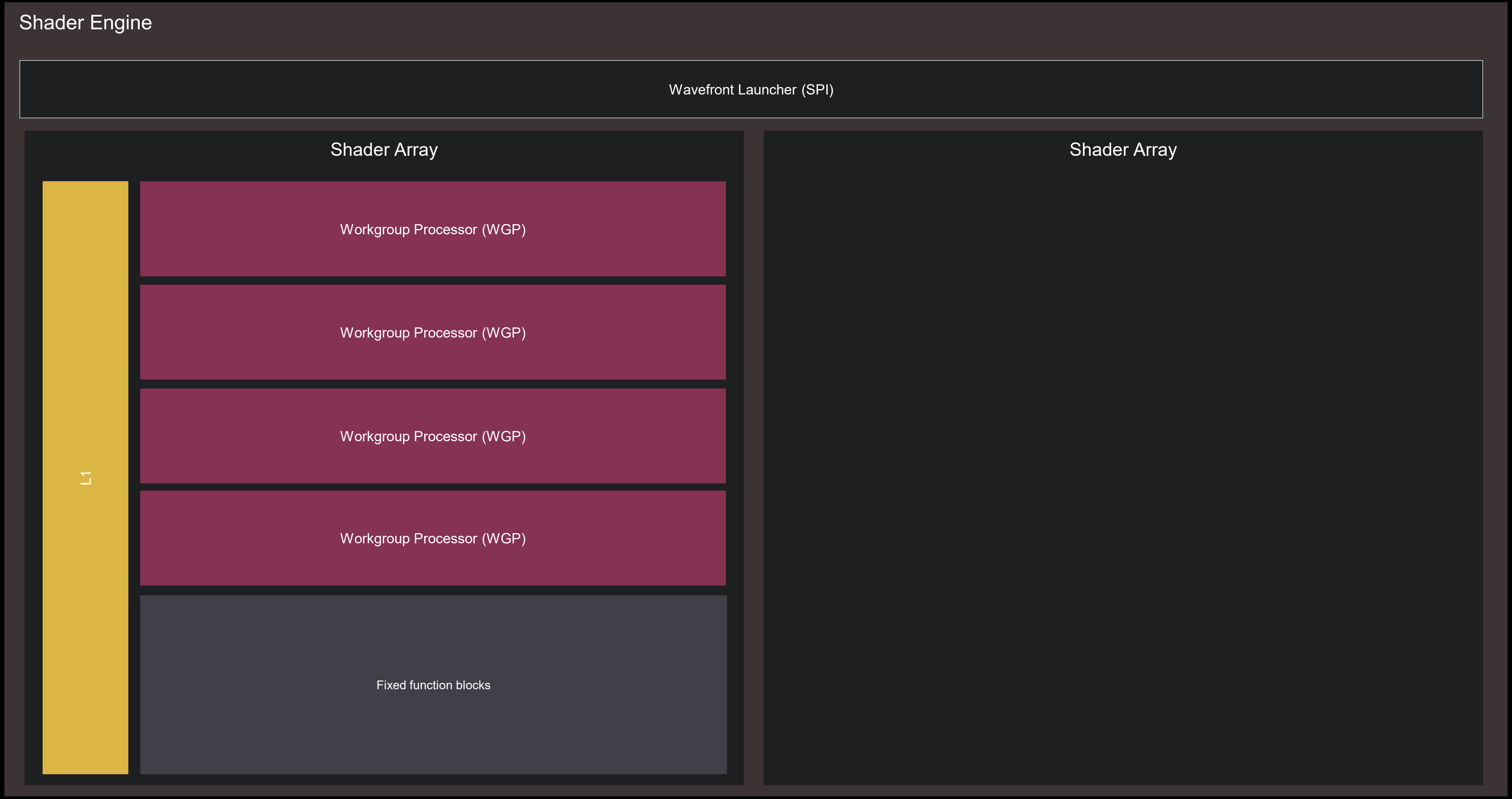
HIGH LEVEL OVERVIEW



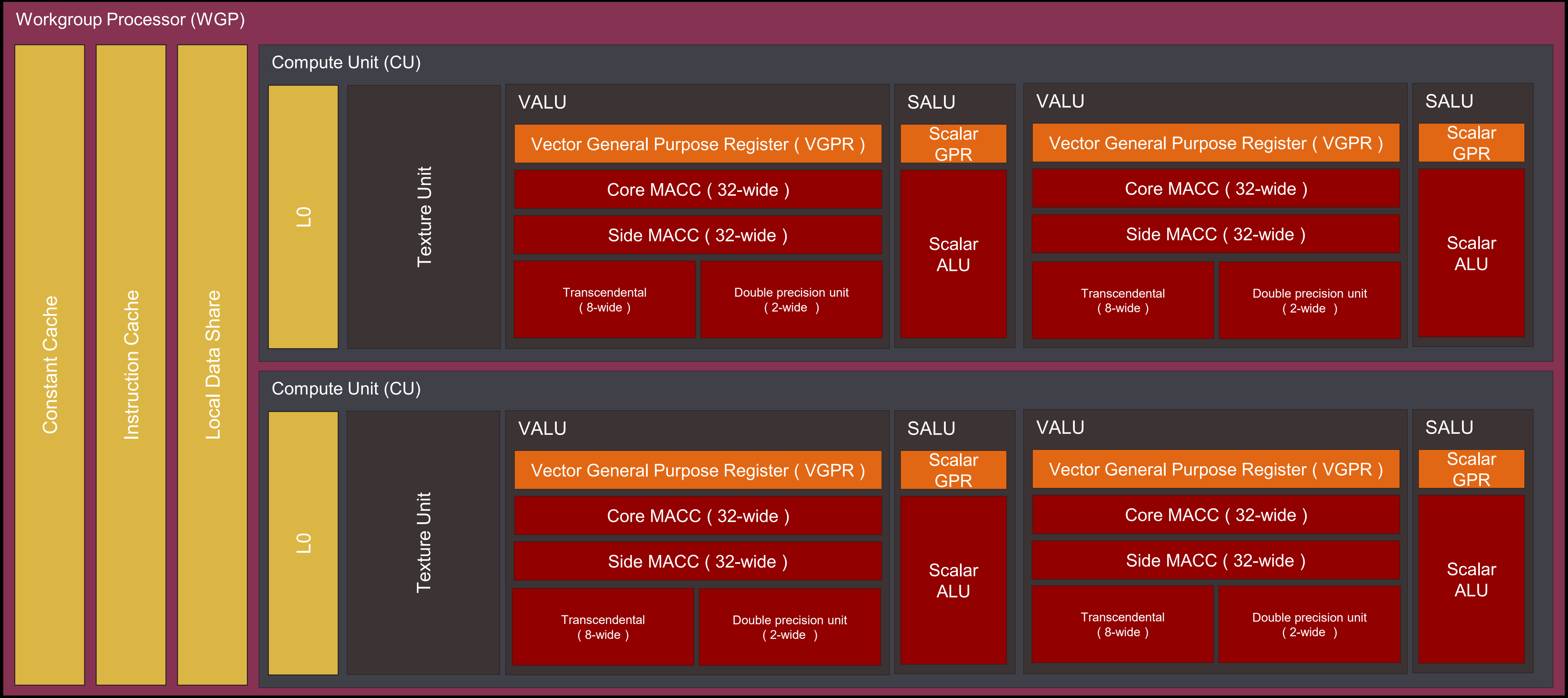
SHADER ENGINE


Shader Engine
Wavefront Launcher (SPI)
Shader Array
L1
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Fixed function blocks
Shader Array
L1
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Fixed function blocks

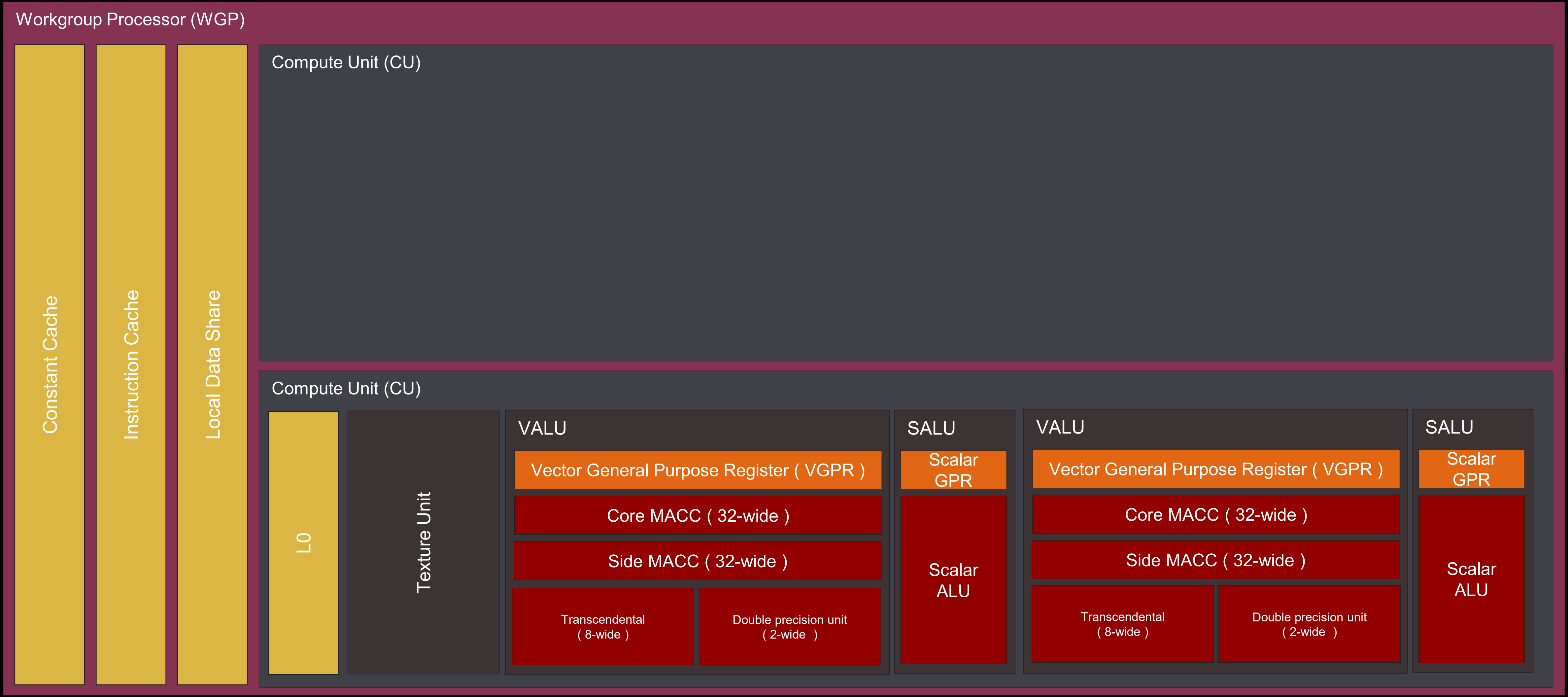
SHADER ENGINE


Shader Engine
Wavefront Launcher (SPI)
Shader Array
Shader Array
L1
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Workgroup Processor (WGP)
Fixed function blocks

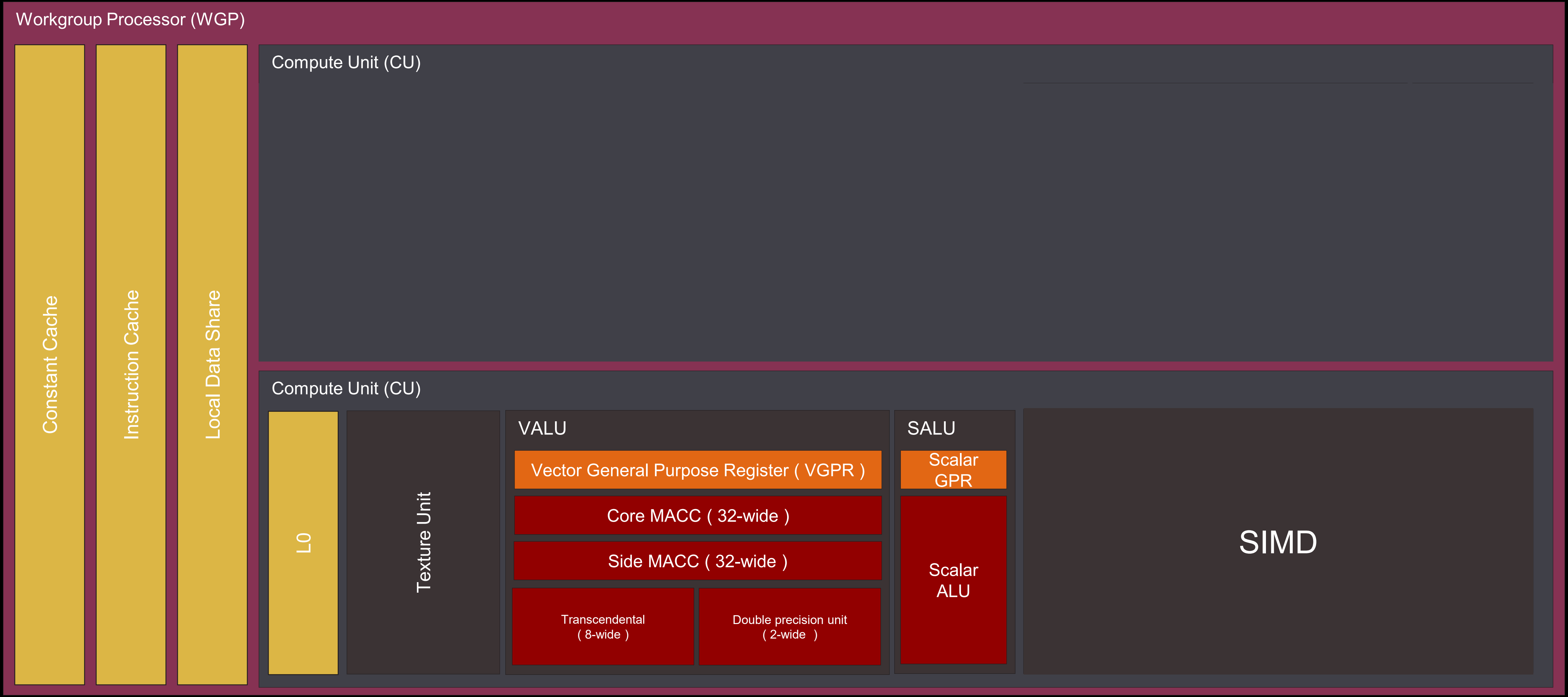
WGP


Workgroup Processor (WGP)
Constant Cache
Instruction Cache
Local Data Share
Compute Unit (CU)
L0
Texture Unit
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU
Compute Unit (CU)
L0
Texture Unit
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU

WGP

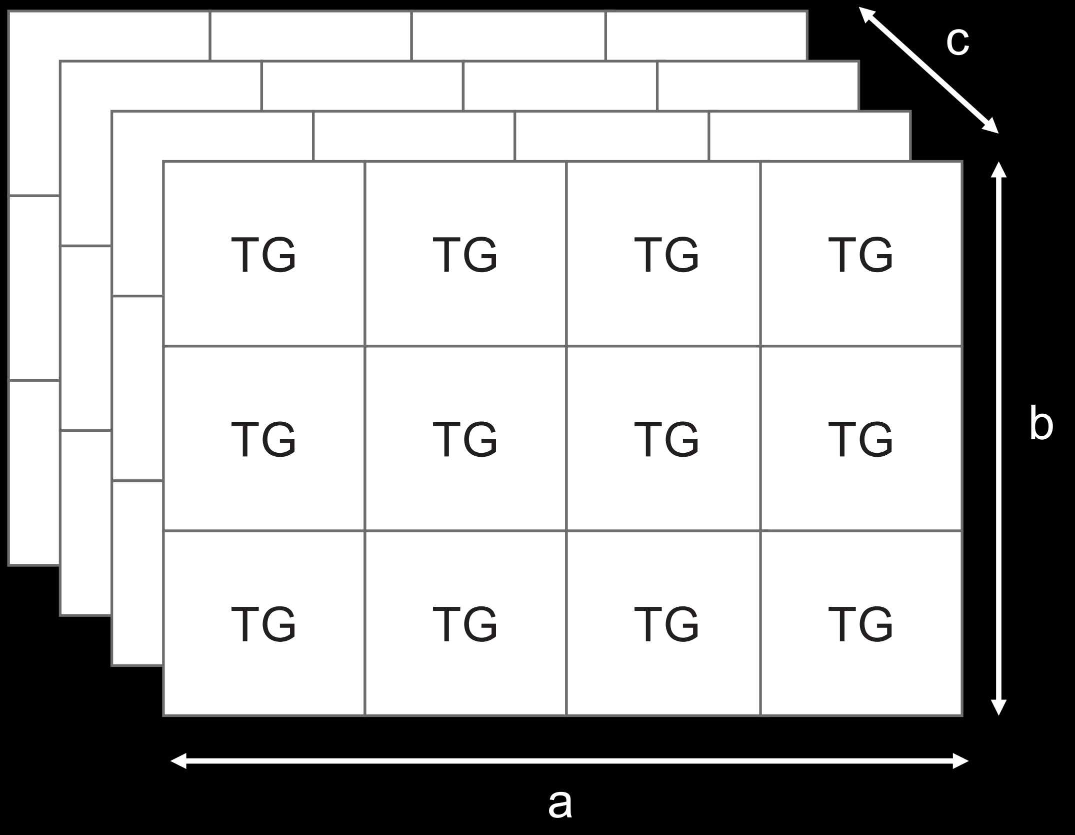

Workgroup Processor (WGP)
Constant Cache
Instruction Cache
Local Data Share
Compute Unit (CU)
Compute Unit (CU)
L0
Texture Unit
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU

WGP

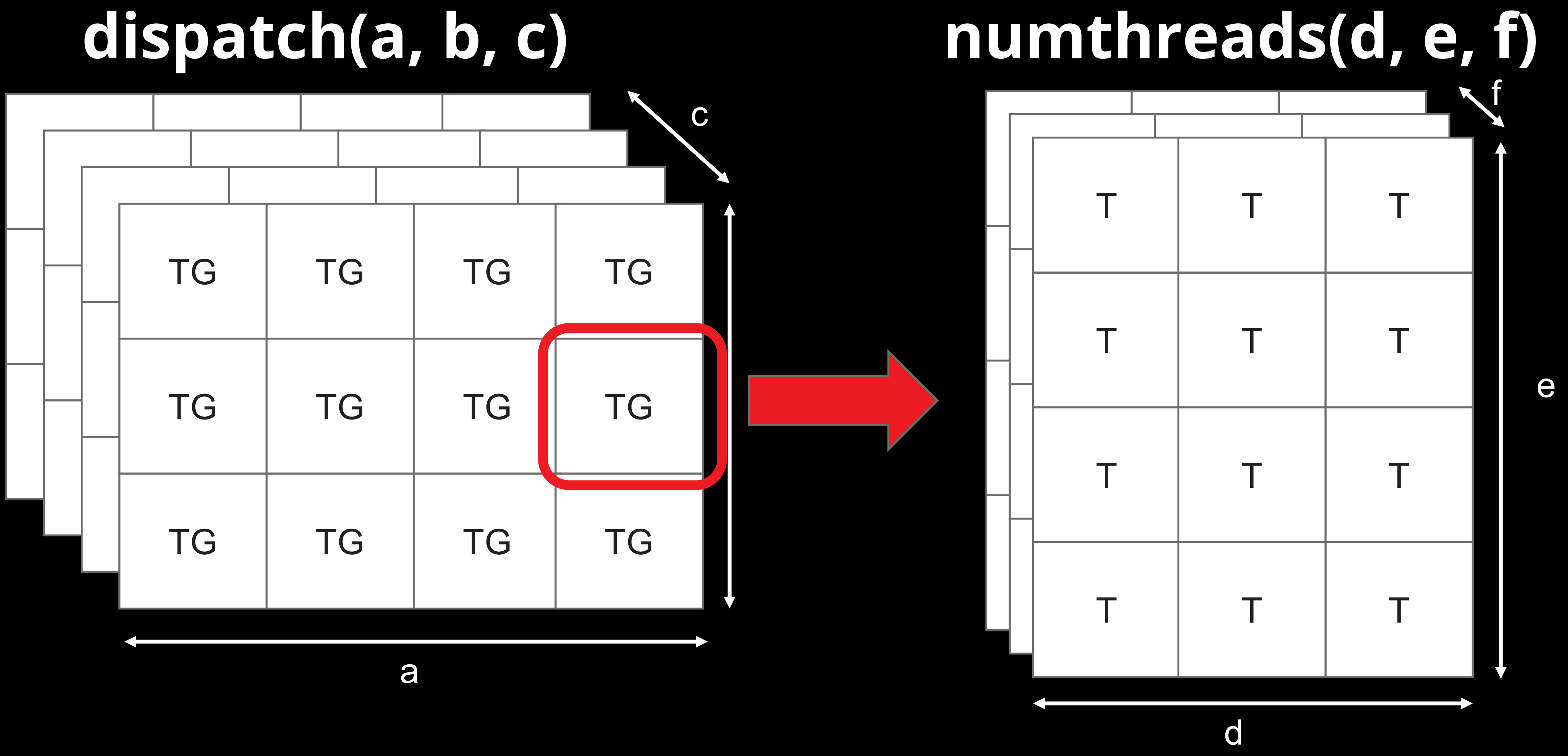

Workgroup Processor (WGP)
Constant Cache
Instruction Cache
Local Data Share
Compute Unit (CU)
Compute Unit (CU)
L0
Texture Unit
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU
SIMD

COMPUTE & THREADGROUPS

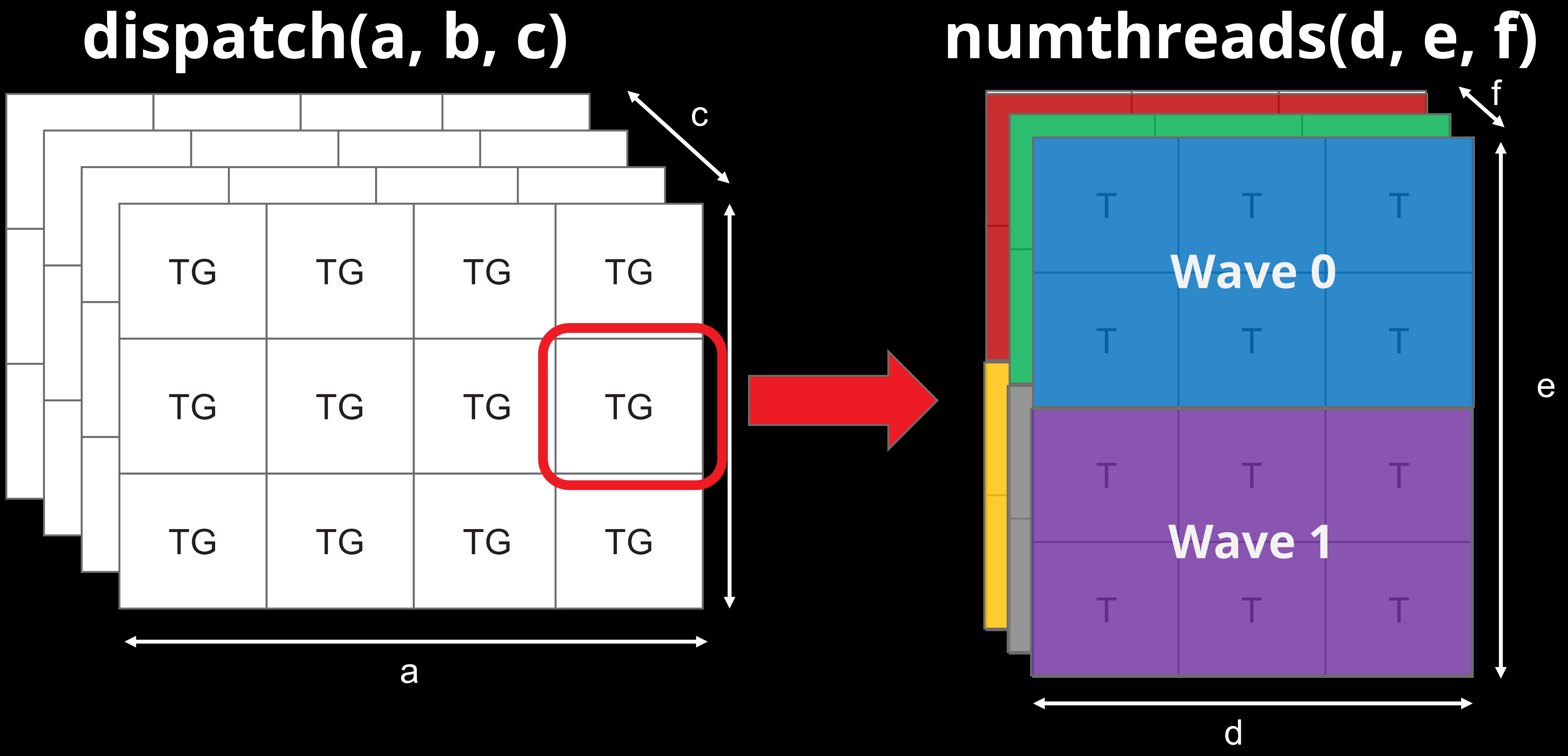
dispatch(a, b, c)



COMPUTE & THREADGROUPS

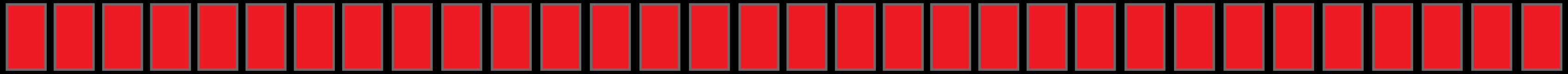


WAVEFRONTS



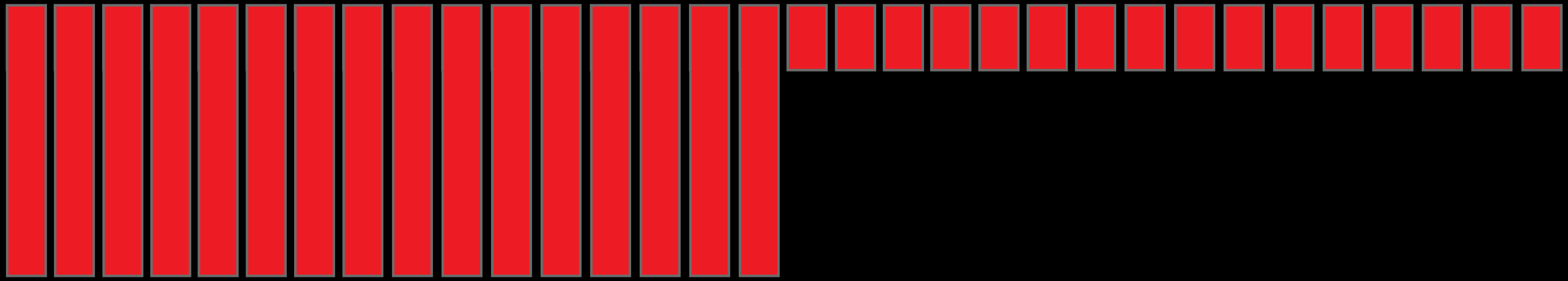
LOCKSTEP EXECUTION

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```



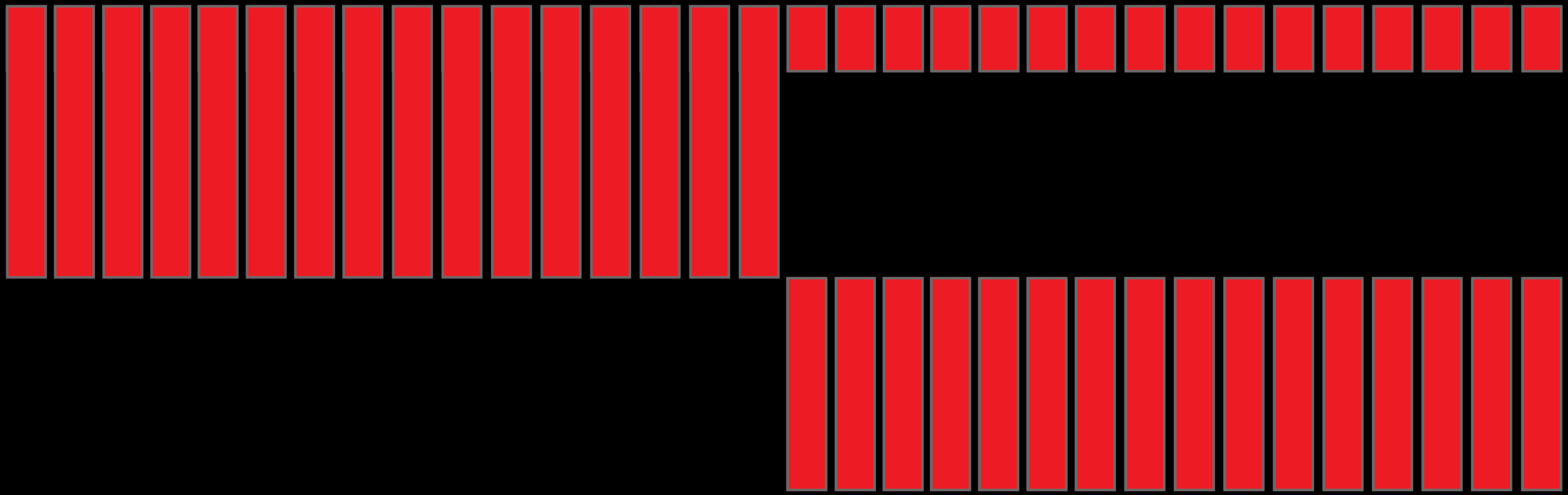
LOCKSTEP EXECUTION

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```



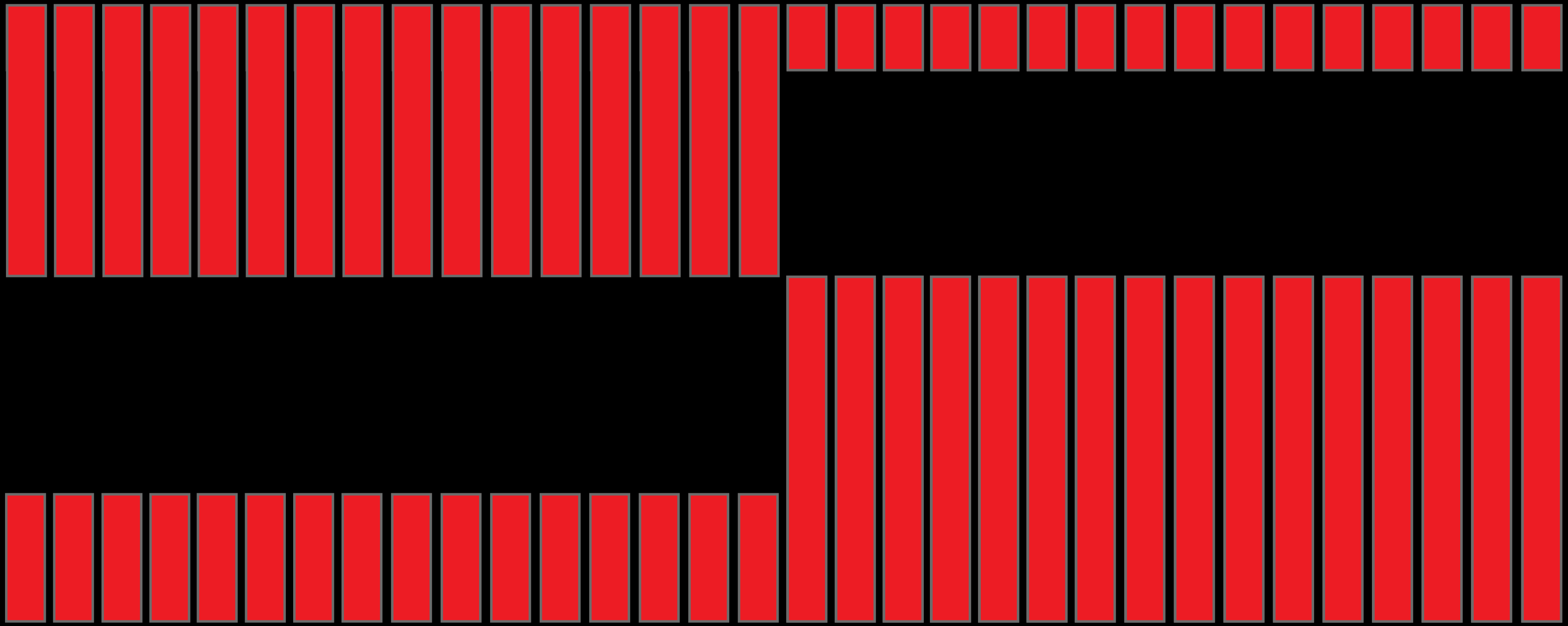
LOCKSTEP EXECUTION

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```



LOCKSTEP EXECUTION

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```

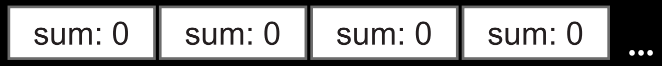


SIMD & LOCKSTEP

```
cbuffer input : register(b0){ int data; };
RWBuffer<int> output : register(u0);

[numthreads(32, 1, 1)]
void CSMain( uint threadIndex : SV_DispatchThreadID )
{
    int sum = 0;
    sum += threadIndex;
    sum += data;
    output[threadIndex] = sum;
}
```

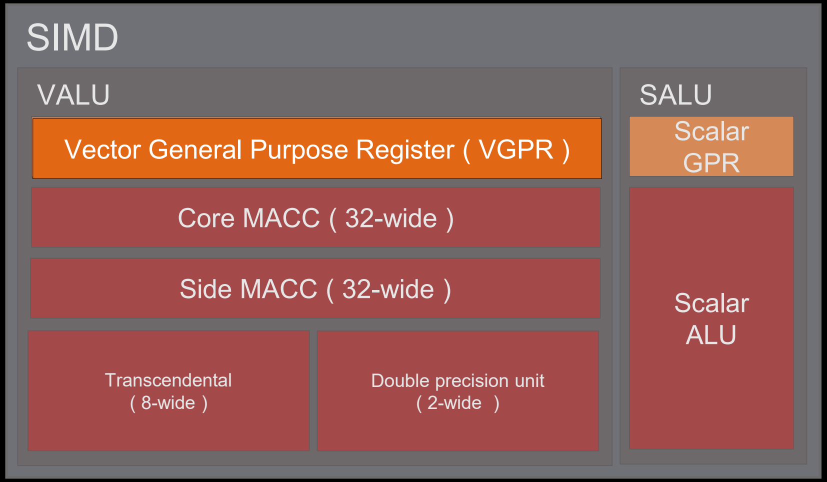
SIMD & LOCKSTEP

```
cbuffer input : register(b0){ int data; };
RWBuffer<int> output : register(u0);

[numthreads(32, 1, 1)]
void CSMain( uint threadIndex : SV_DispatchThreadID )
{
    int sum = 0;
    sum += threadIndex;
    sum += data;
    output[threadIndex] = sum;
}
```



sum: 0 | sum: 0 | sum: 0 | sum: 0 …

SIMD & LOCKSTEP

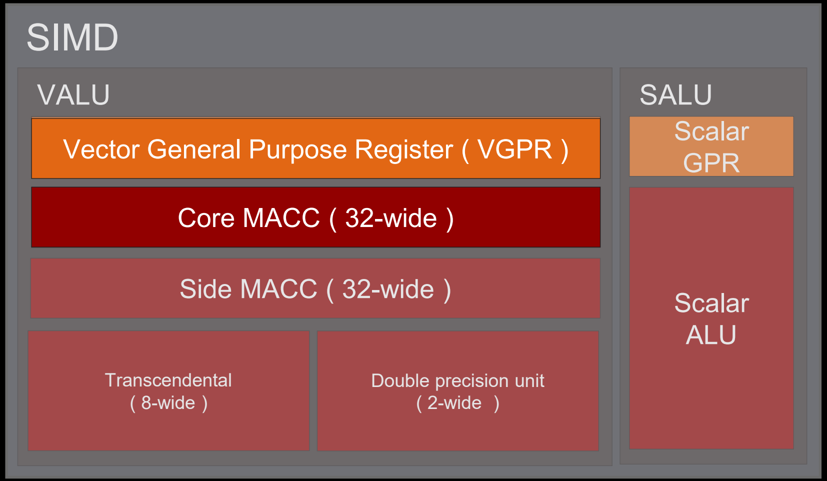


```
cbuffer input : register(b0){ int data; };
RWBuffer<int> output : register(u0);

[numthreads(32, 1, 1)]
void CSMain( uint threadIndex : SV_DispatchThreadID )
{
    int sum = 0;
    sum += threadIndex;
    sum += data;
    output[threadIndex] = sum;
}
```

VGPR

sum: 0	sum: 0	sum: 0	sum: 0	…

SIMD & LOCKSTEP



SIMD

VALU

Vector General Purpose Register (VGPR)

Core MACC (32-wide)

Side MACC (32-wide)

Transcendental (8-wide)

Double precision unit (2-wide)

SALU

Scalar GPR

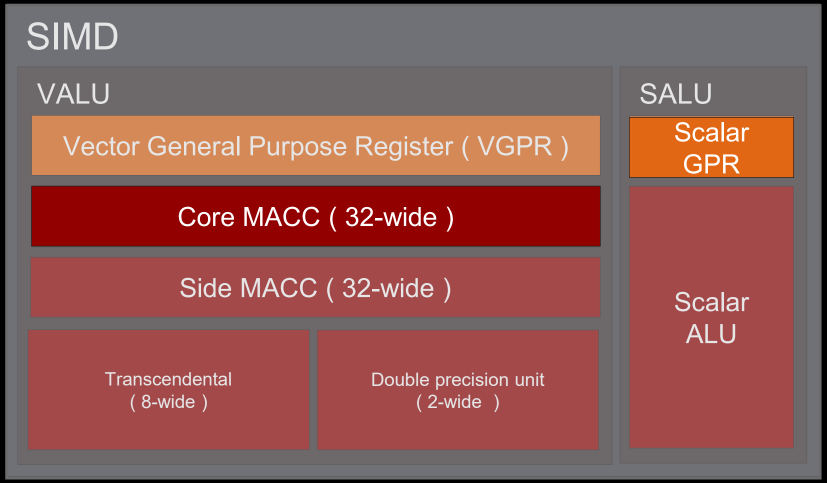
Scalar ALU

```
cbuffer input : register(b0){ int data; };
RWBuffer<int> output : register(u0);

[numthreads(32, 1, 1)]
void CSMain( uint threadIndex : SV_DispatchThreadID )
{
    int sum = 0;
    sum += threadIndex;
    sum += data;
    output[threadIndex] = sum;
}
```

VGPR

sum: 0	sum: 0	sum: 0	sum: 0	…
sum: 0	sum: 1	sum: 2	sum: 3	…

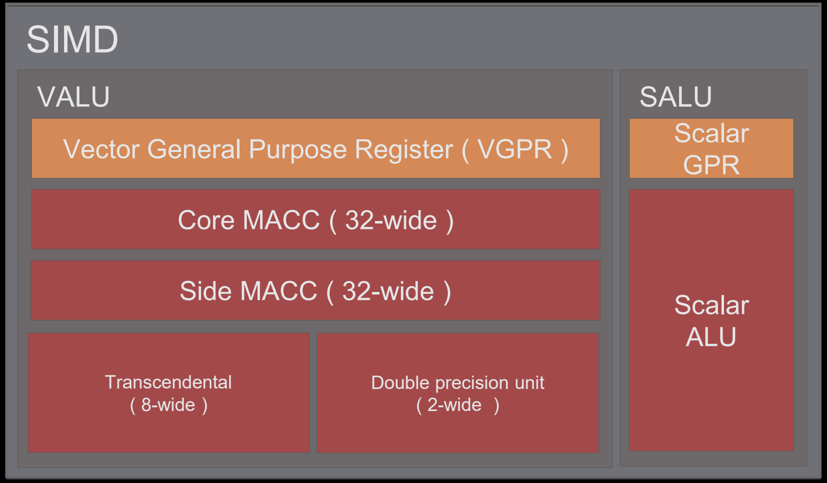
SIMD & LOCKSTEP


SIMD
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU

```
cbuffer input : register(b0){ int data; };
RWBuffer<int> output : register(u0);

[numthreads(32, 1, 1)]
void CSMain( uint threadIndex : SV_DispatchThreadID )
{
    int sum = 0;
    sum += threadIndex;
    sum += data;
    output[threadIndex] = sum;
}
```

VGPR					SGPR
sum: 0	sum: 0	sum: 0	sum: 0	…	data: 5
sum: 0	sum: 1	sum: 2	sum: 3	…	data: 5
sum: 5	sum: 6	sum: 7	sum: 8	…	data: 5

SIMD & LOCKSTEP


SIMD
VALU
Vector General Purpose Register (VGPR)
Core MACC (32-wide)
Side MACC (32-wide)
Transcendental (8-wide)
Double precision unit (2-wide)
SALU
Scalar GPR
Scalar ALU

```
cbuffer input : register(b0){ int data; };
RWBuffer<int> output : register(u0);

[numthreads(32, 1, 1)]
void CSMain( uint threadIndex : SV_DispatchThreadID )
{
    int sum = 0;
    sum += threadIndex;
    sum += data;
    output[threadIndex] = sum;
}
```

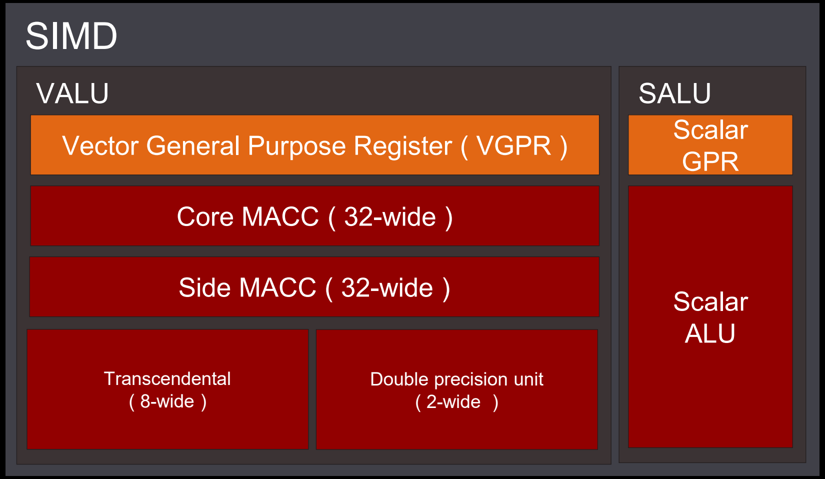
VGPR

sum: 0	sum: 0	sum: 0	sum: 0	…
sum: 0	sum: 1	sum: 2	sum: 3	…
sum: 5	sum: 6	sum: 7	sum: 8	…
sum: 5	sum: 6	sum: 7	sum: 8	…

SGPR

data: 5
data: 5
data: 5
data: 5

SIMD & LOCKSTEP



SIMD

VALU

Vector General Purpose Register (VGPR)

Core MACC (32-wide)

Side MACC (32-wide)

Transcendental
(8-wide)

Double precision unit
(2-wide)

SALU

Scalar
GPR

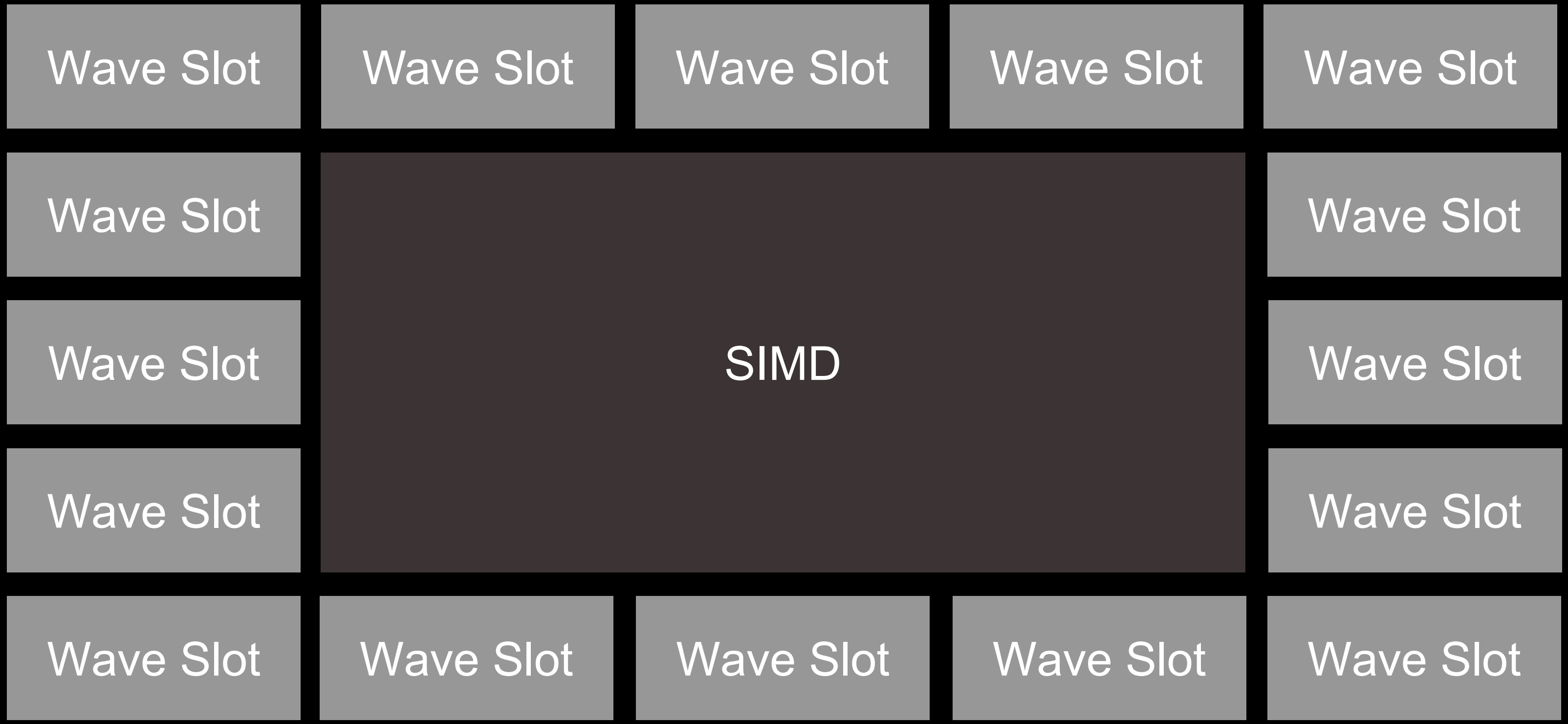
Scalar
ALU

```
cbuffer input : register(b0){ int data; };
RWBuffer<int> output : register(u0);

[numthreads(32, 1, 1)]
void CSMain( uint threadIndex : SV_DispatchThreadID )
{
    int sum = 0;
    sum += threadIndex;
    sum += data;
    output[threadIndex] = sum;
}
```

VGPR

sum: 0	sum: 0	sum: 0	sum: 0	…	data: 5
sum: 0	sum: 1	sum: 2	sum: 3	…	data: 5
sum: 5	sum: 6	sum: 7	sum: 8	…	data: 5
sum: 5	sum: 6	sum: 7	sum: 8	…	data: 5

ASSIGNED WAVEFRONTS

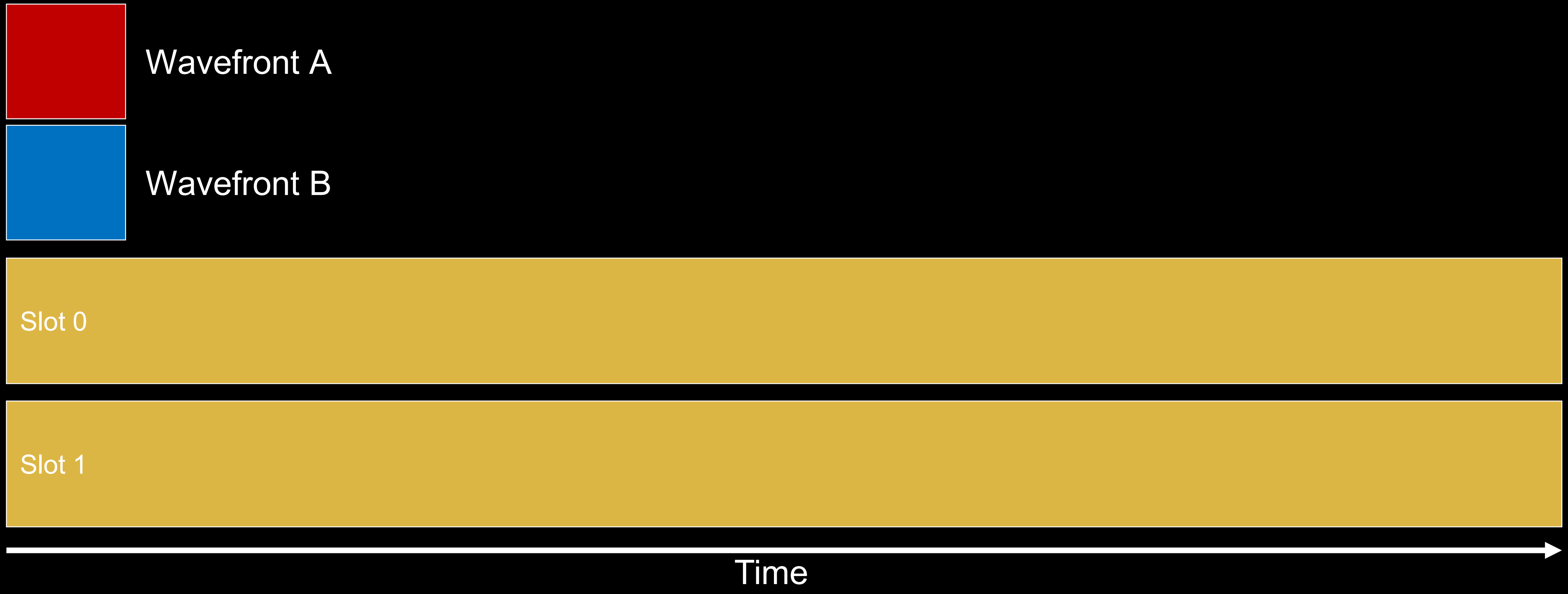


ASSIGNED WAVEFRONTS

Wavefronts don’t have to be executed in order

Wavefronts execution can be interrupted and resumed at any time

WAVEFRONT SCHEDULING & LATENCY HIDING



🟥 Wavefront A

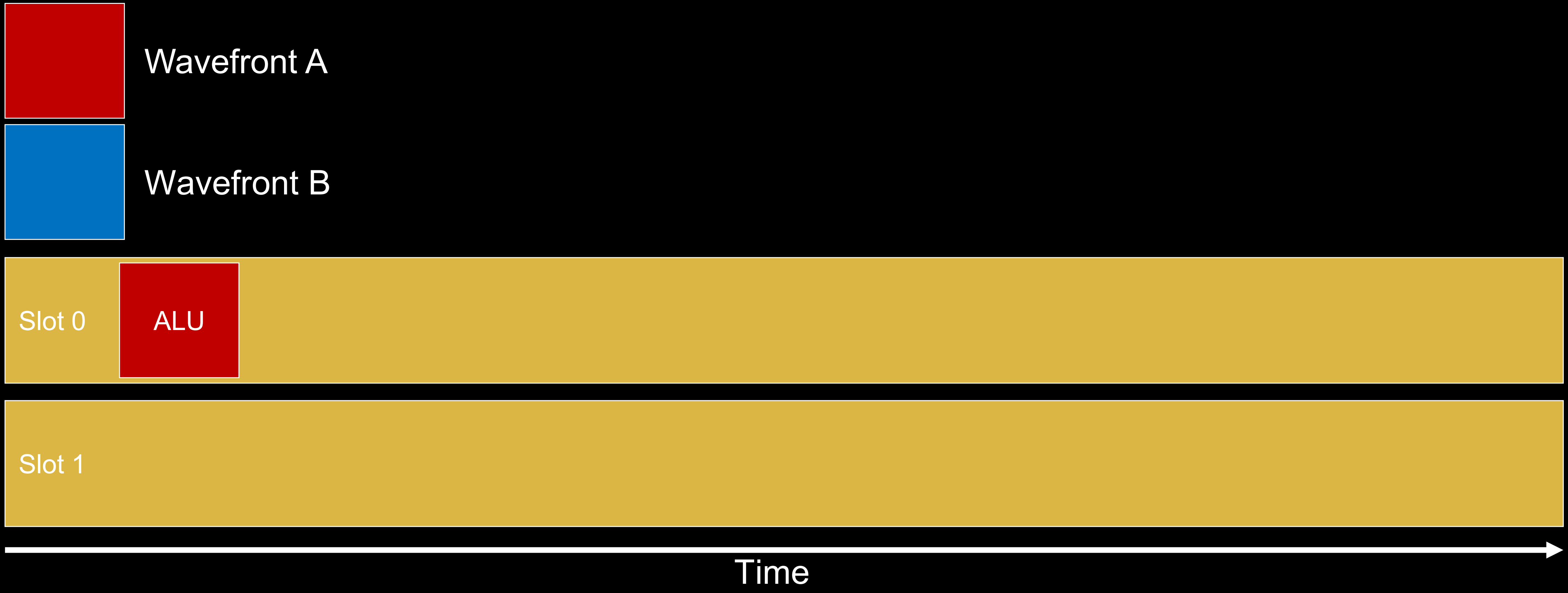
🟦 Wavefront B

Slot 0

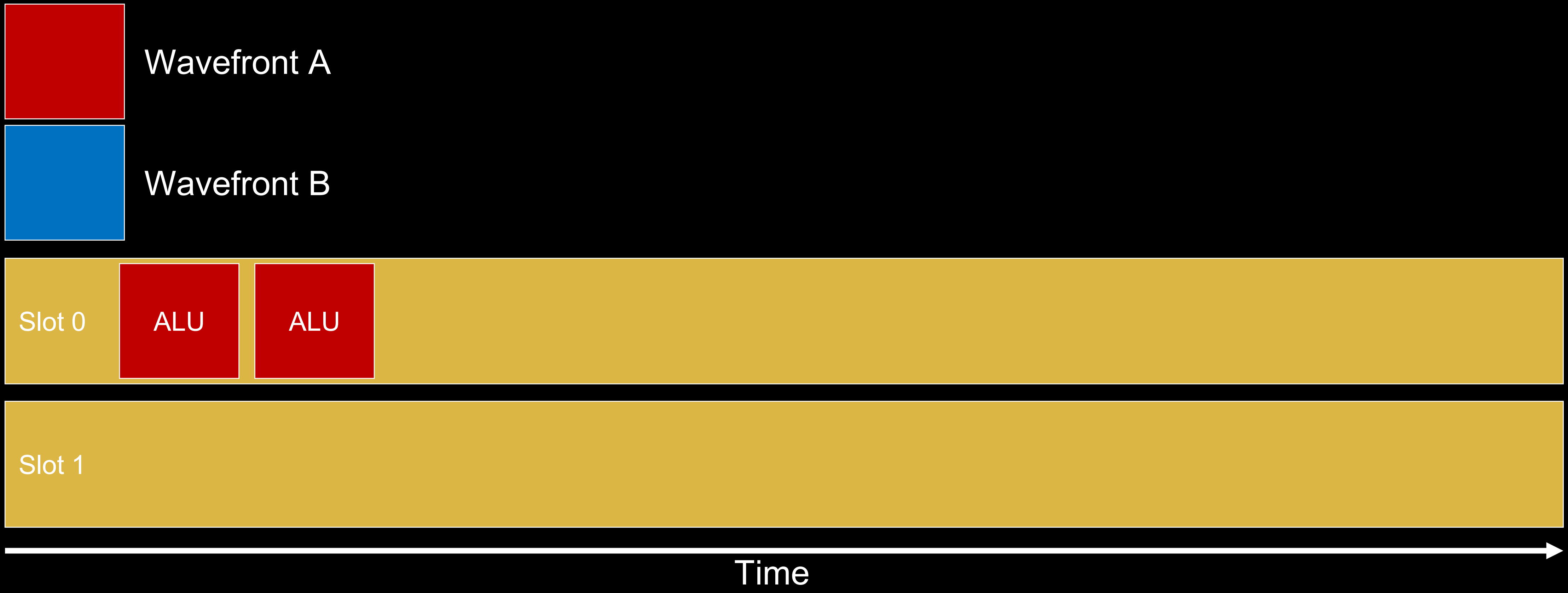
Slot 1

Time

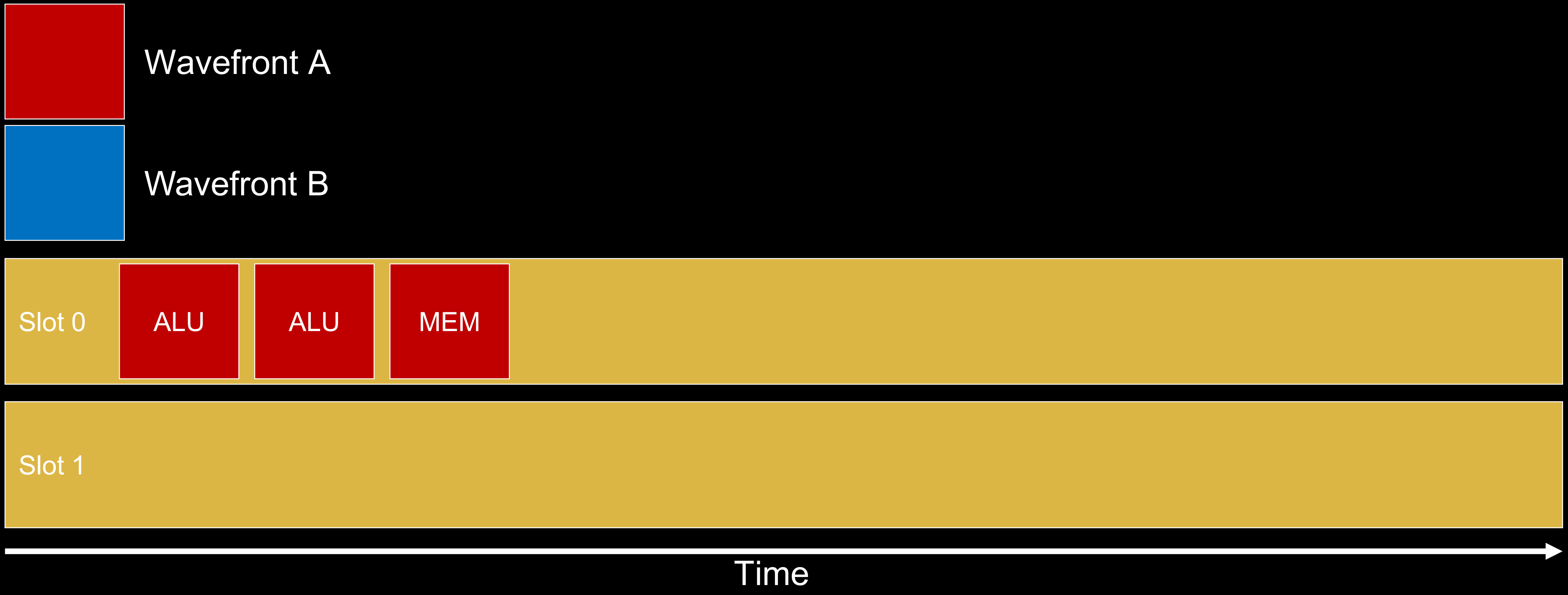
WAVEFRONT SCHEDULING & LATENCY HIDING



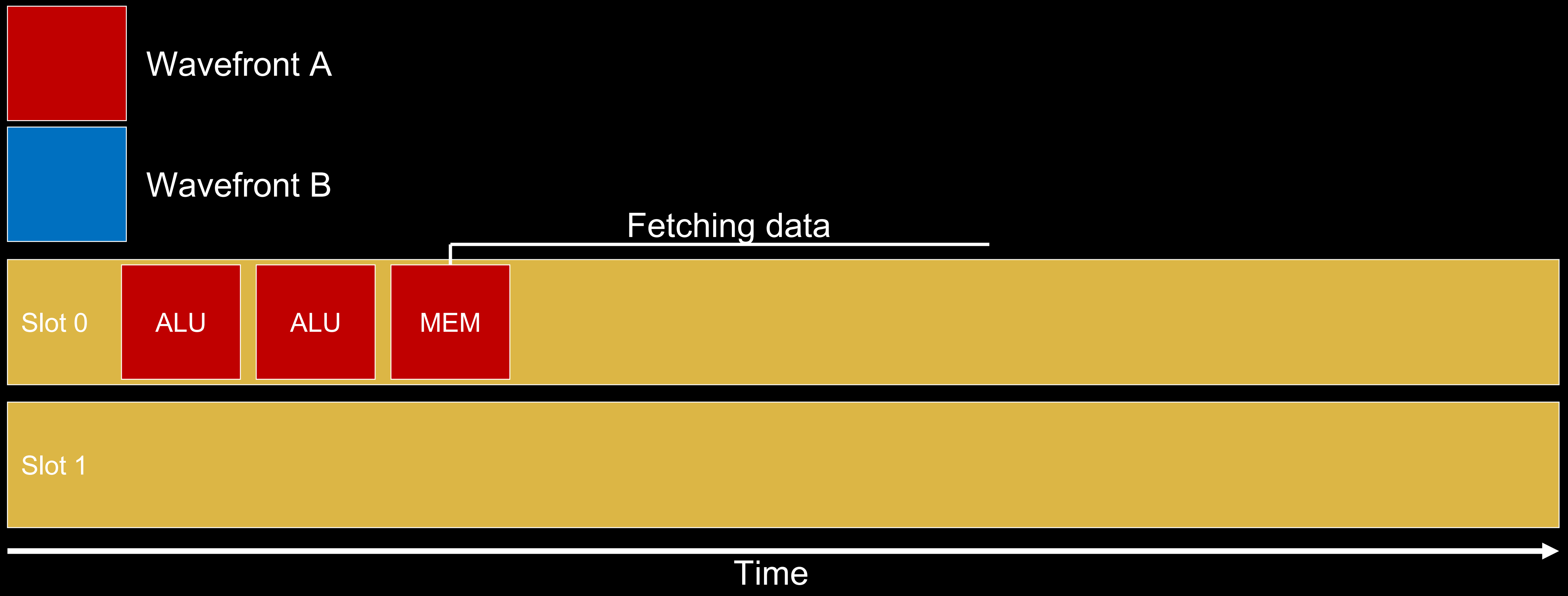
WAVEFRONT SCHEDULING & LATENCY HIDING



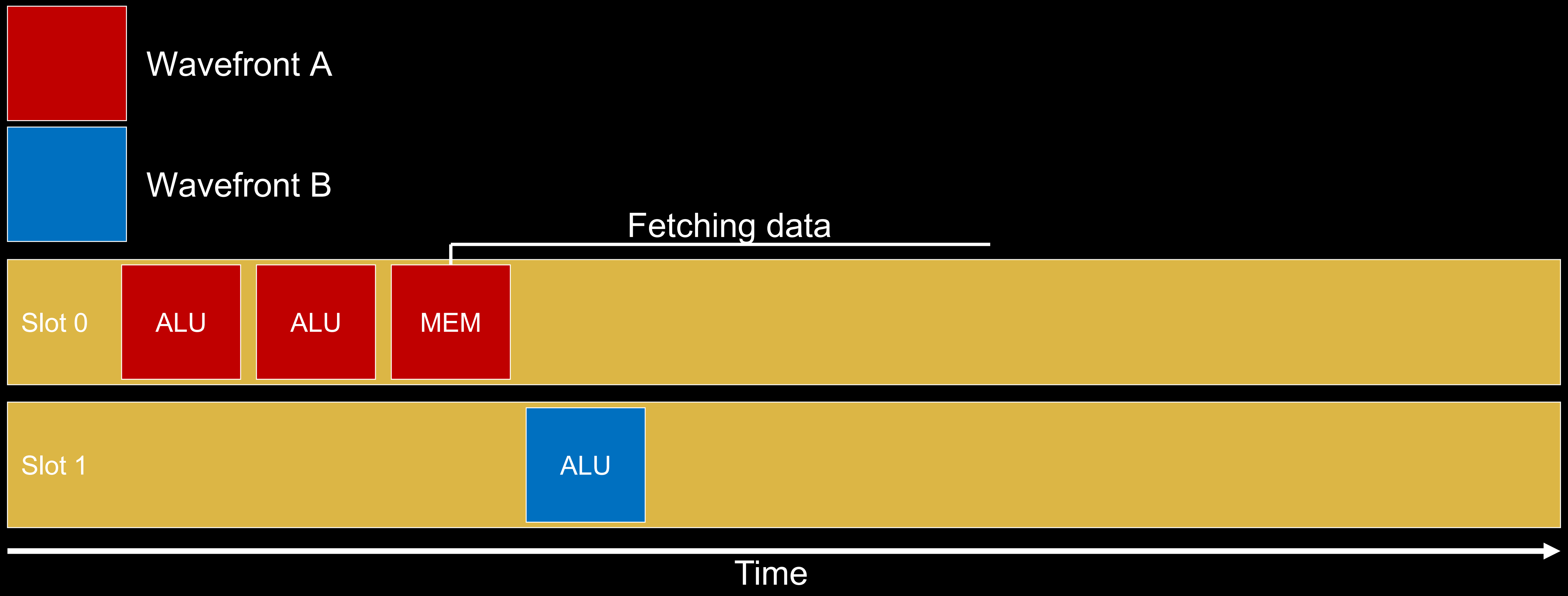
WAVEFRONT SCHEDULING & LATENCY HIDING



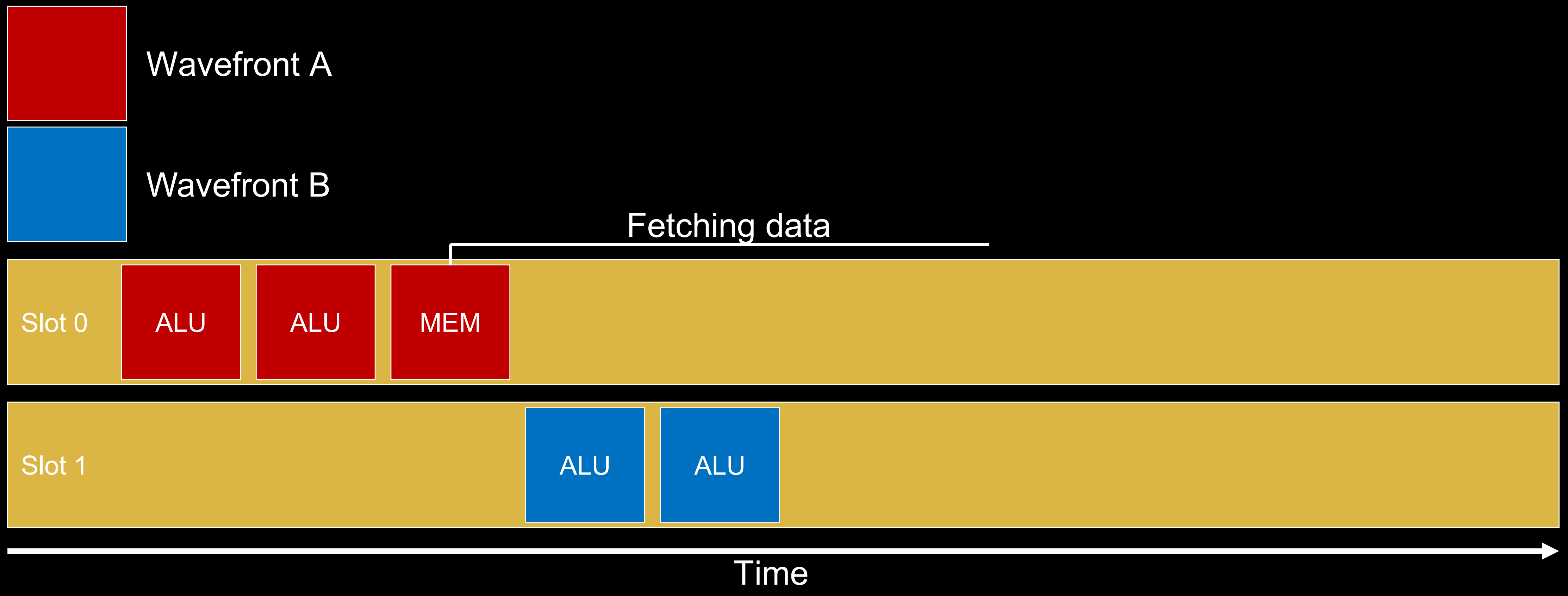
WAVEFRONT SCHEDULING & LATENCY HIDING



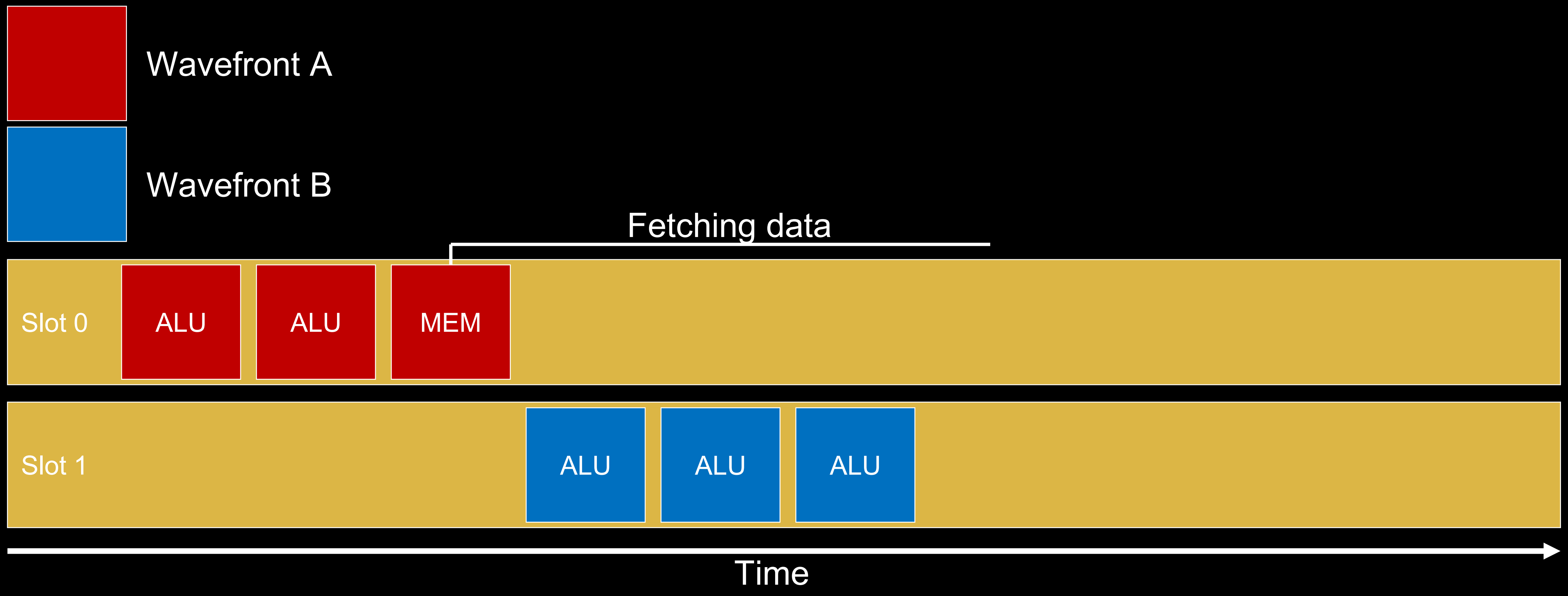
WAVEFRONT SCHEDULING & LATENCY HIDING



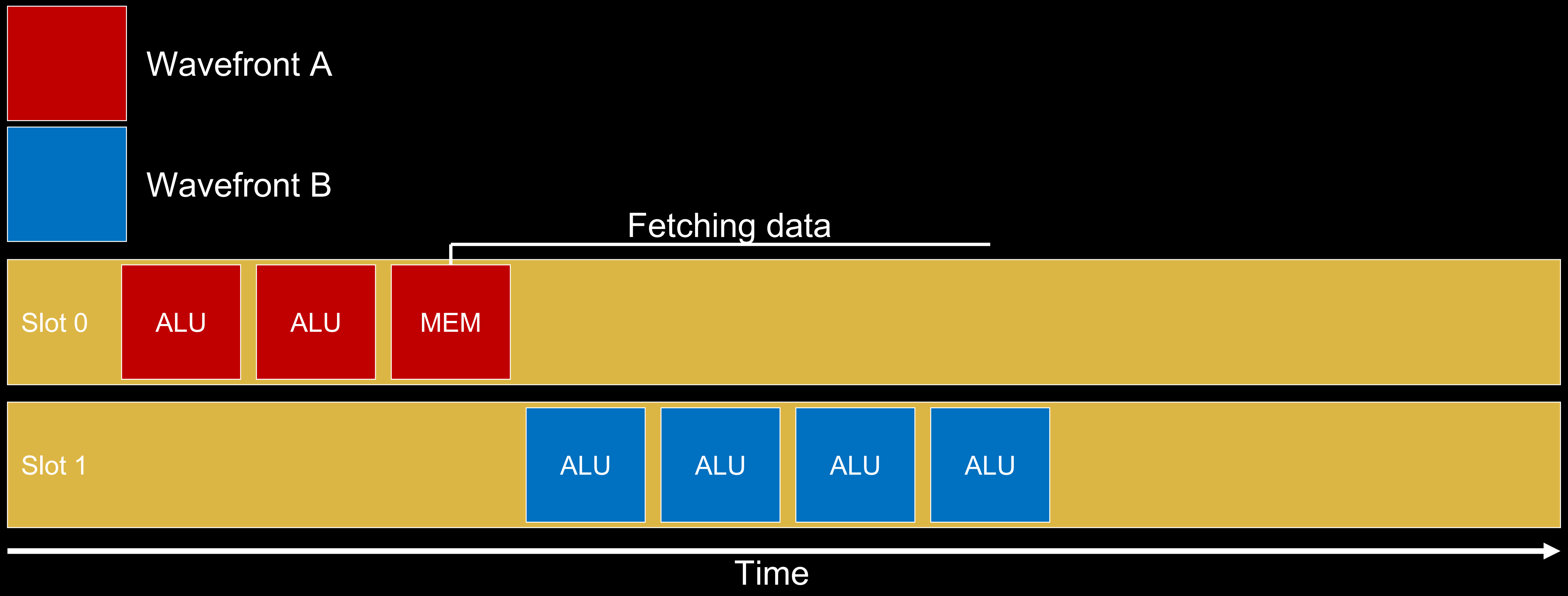
WAVEFRONT SCHEDULING & LATENCY HIDING



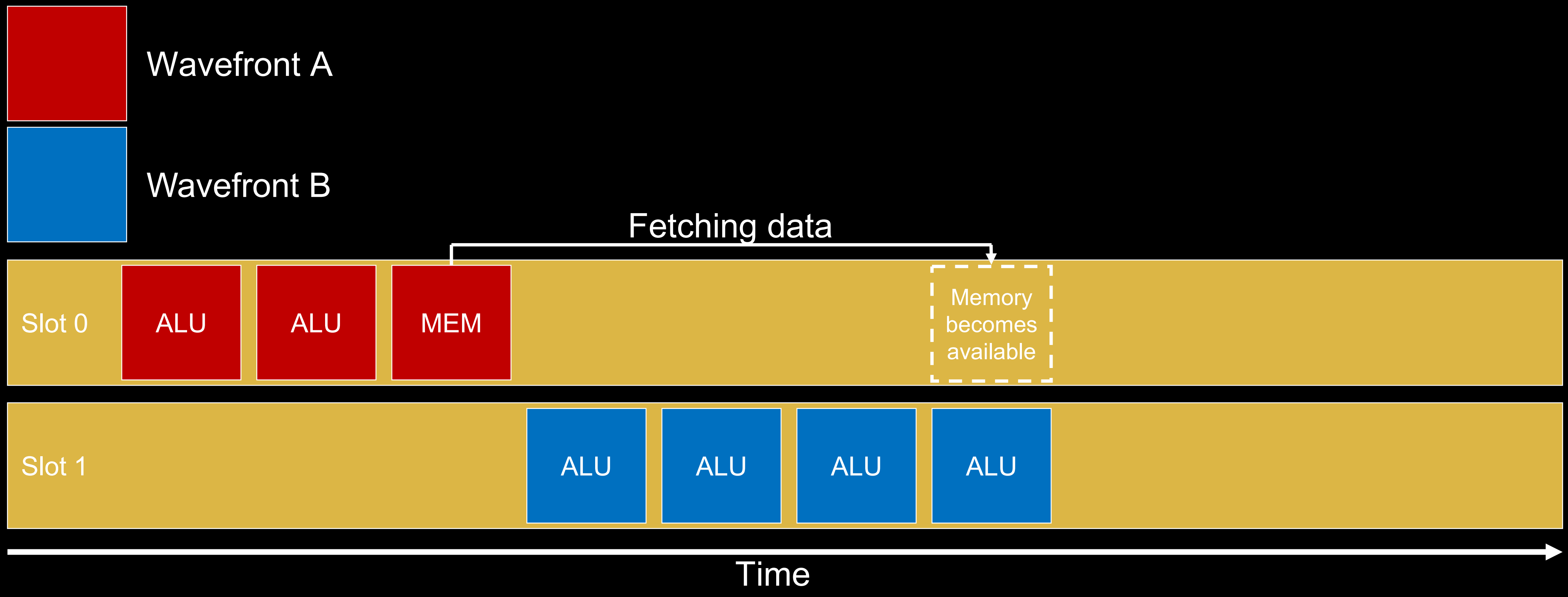
WAVEFRONT SCHEDULING & LATENCY HIDING



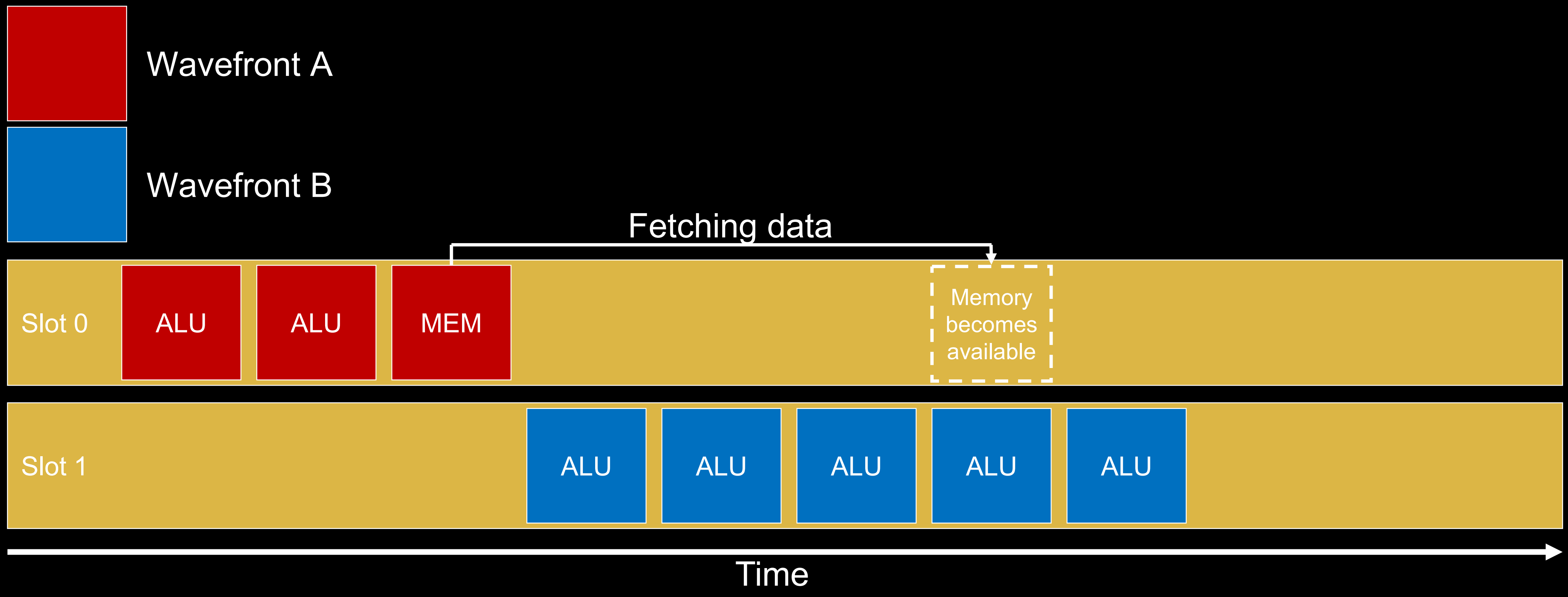
WAVEFRONT SCHEDULING & LATENCY HIDING



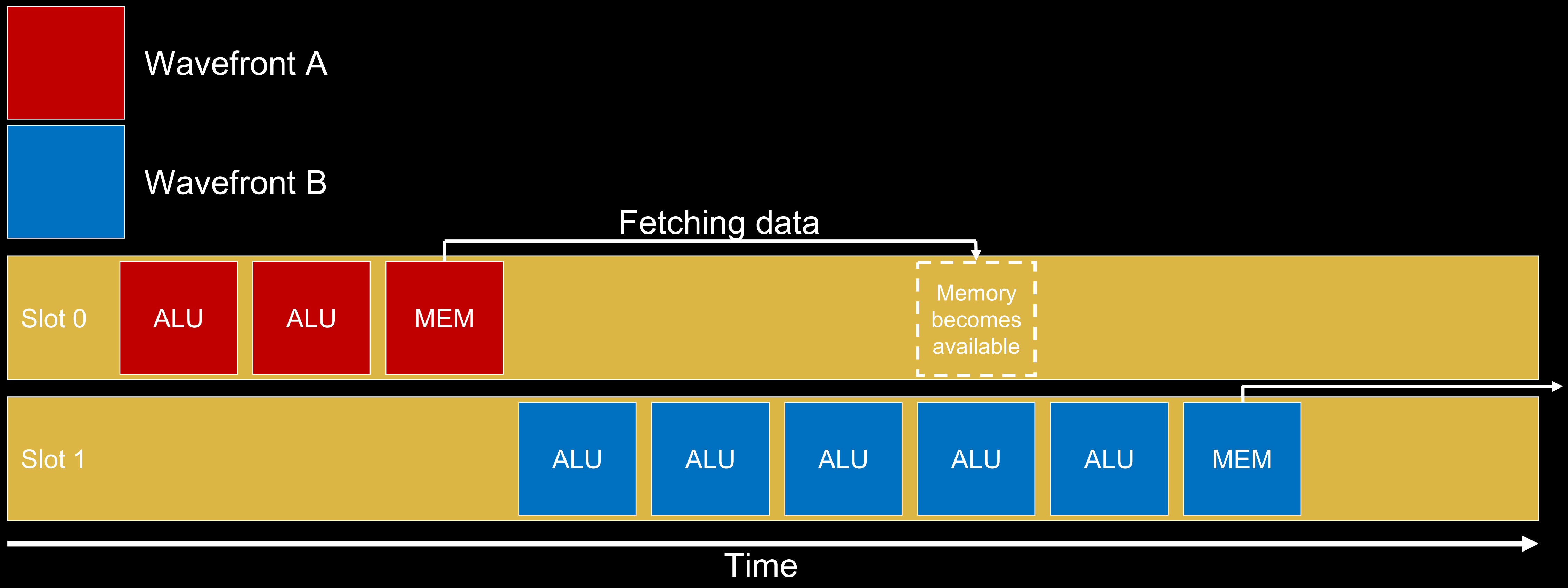
WAVEFRONT SCHEDULING & LATENCY HIDING



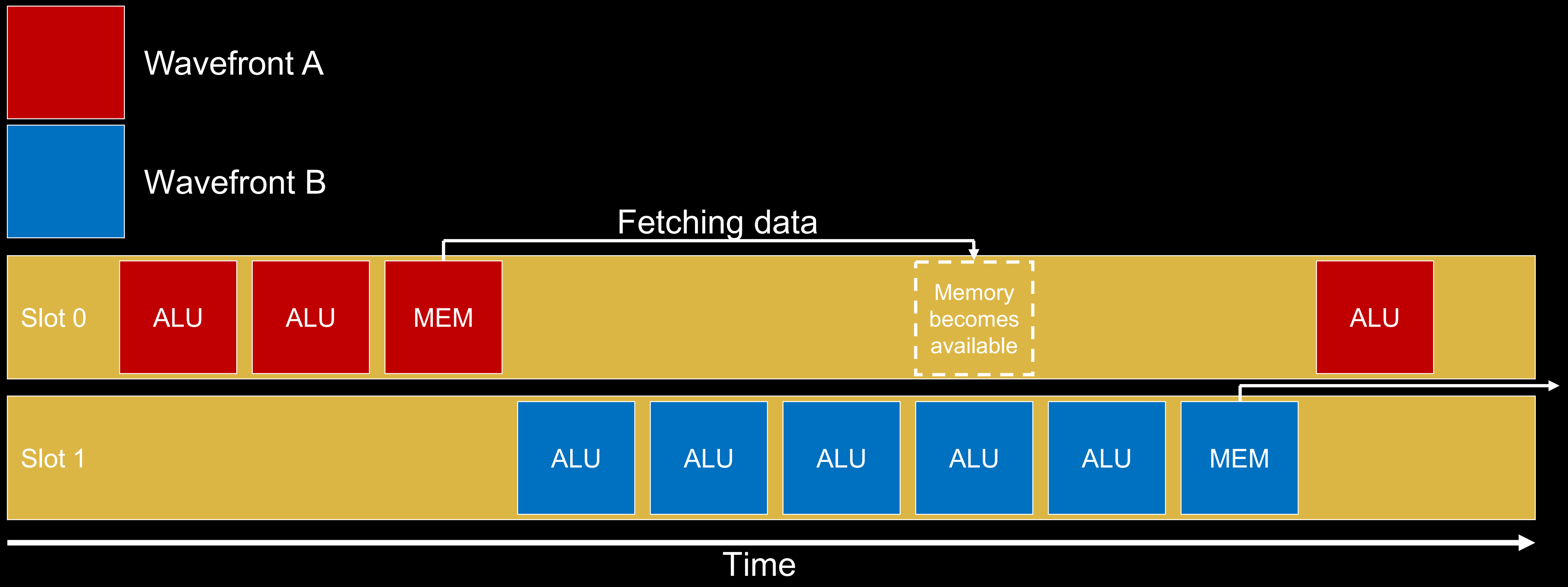
WAVEFRONT SCHEDULING & LATENCY HIDING



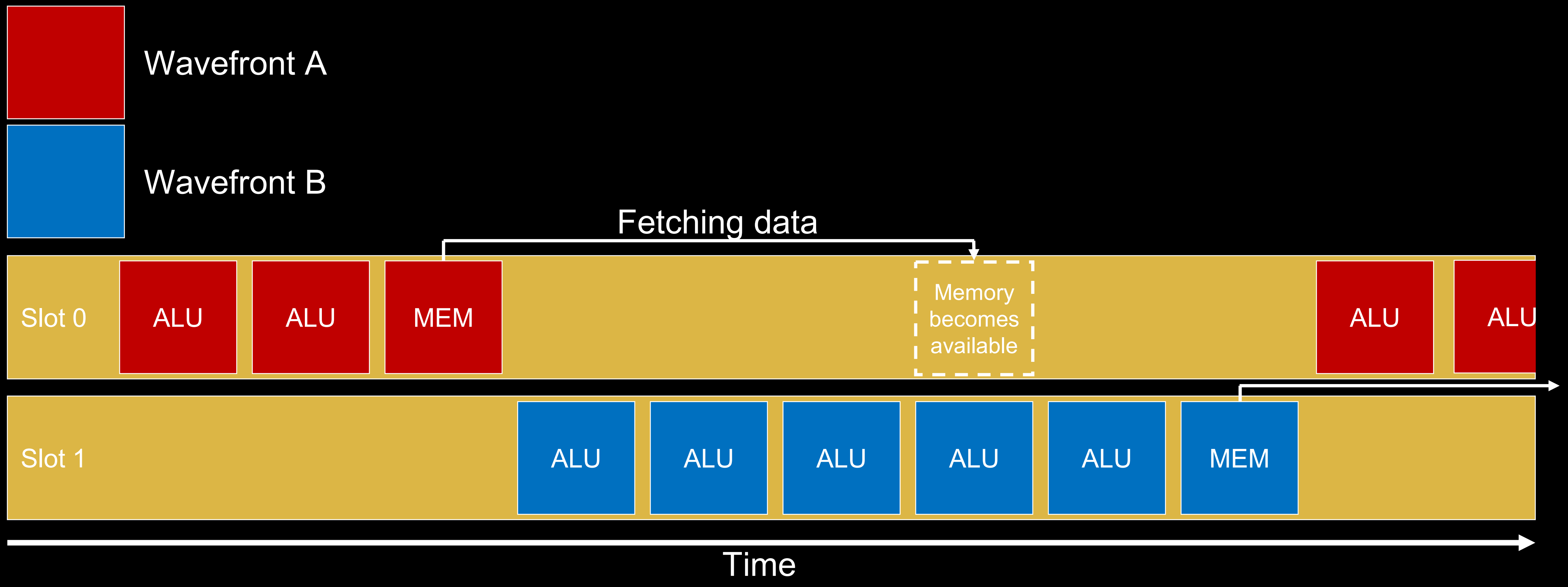
WAVEFRONT SCHEDULING & LATENCY HIDING



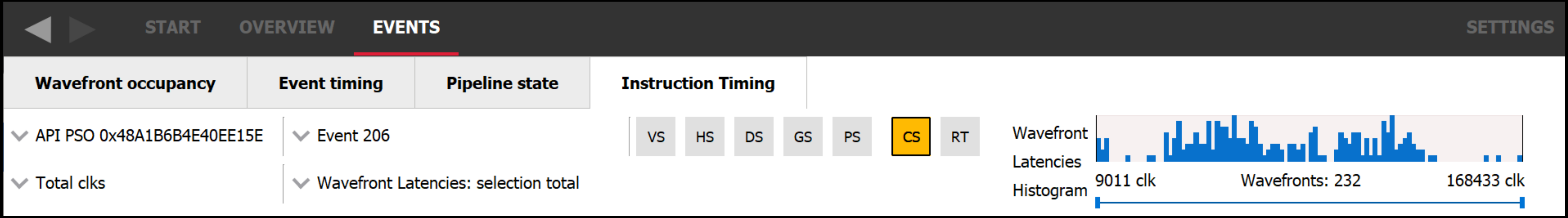
WAVEFRONT SCHEDULING & LATENCY HIDING



WAVEFRONT SCHEDULING & LATENCY HIDING



LATENCY HIDING IN RGP

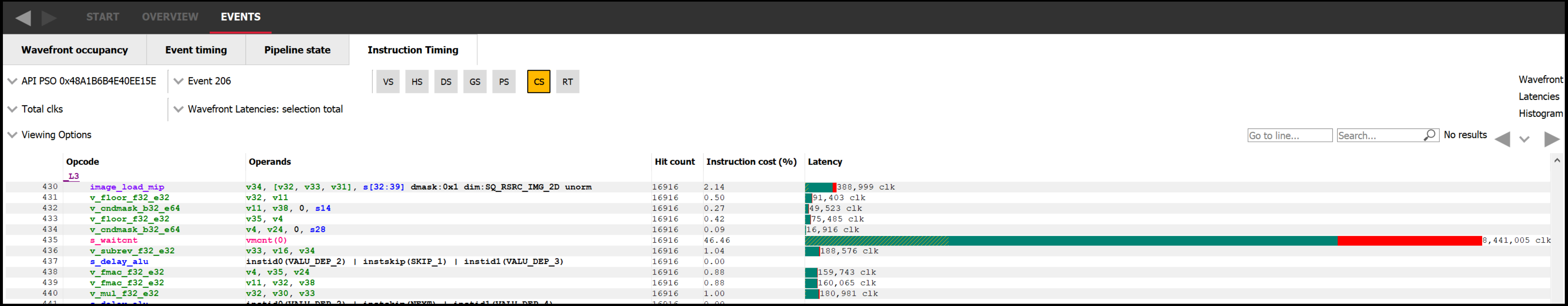


API PSO 0x48A1B6B4E40EE15E

Event 206

Total clks

Wavefront Latencies: selection total



API PSO 0x48A1B6B4E40EE15E

Event 206

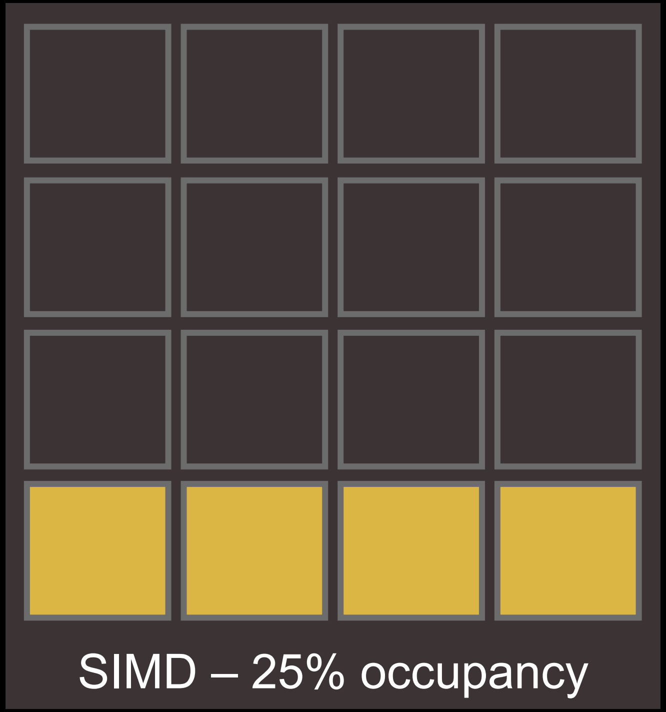
Total clks

Wavefront Latencies: selection total

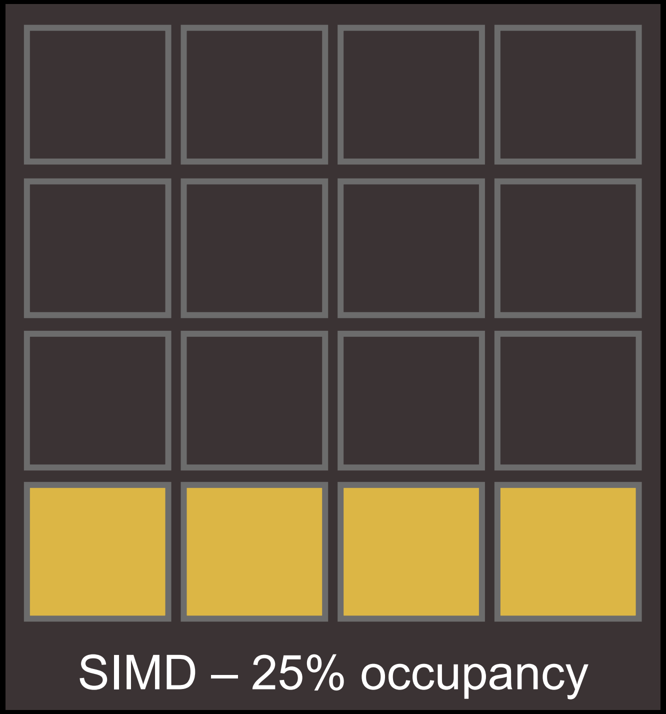
Viewing Options

	Opcode	Operands	Hit count	Instruction cost (%)	Latency
	_L3				
430	image_load_mip	v34, [v32, v33, v31], s[32:39] dmask:0x1 dim:SQ_RSRC_IMG_2D unorm	16916	2.14	388,999 clk
431	v_floor_f32_e32	v32, v11	16916	0.50	91,403 clk
432	v_cndmask_b32_e64	v11, v38, 0, s14	16916	0.27	49,523 clk
433	v_floor_f32_e32	v35, v4	16916	0.42	75,485 clk
434	v_cndmask_b32_e64	v4, v24, 0, s28	16916	0.09	16,916 clk
435	s_waitcnt	vmcnt(0)	16916	46.46	8,441,005 clk
436	v_subrev_f32_e32	v33, v16, v34	16916	1.04	188,576 clk
437	s_delay_alu	instid0(VALU_DEP_2) | instskip(SKIP_1) | instid1(VALU_DEP_3)	16916	0.00	
438	v_fmac_f32_e32	v4, v35, v24	16916	0.88	159,743 clk
439	v_fmac_f32_e32	v11, v32, v38	16916	0.88	160,065 clk
440	v_mul_f32_e32	v32, v30, v33	16916	1.00	180,981 clk
441	s_delay_alu	instid0(VALU_DEP_2) | instskip(NEXT) | instid1(VALU_DEP_4)	16916	0.00	

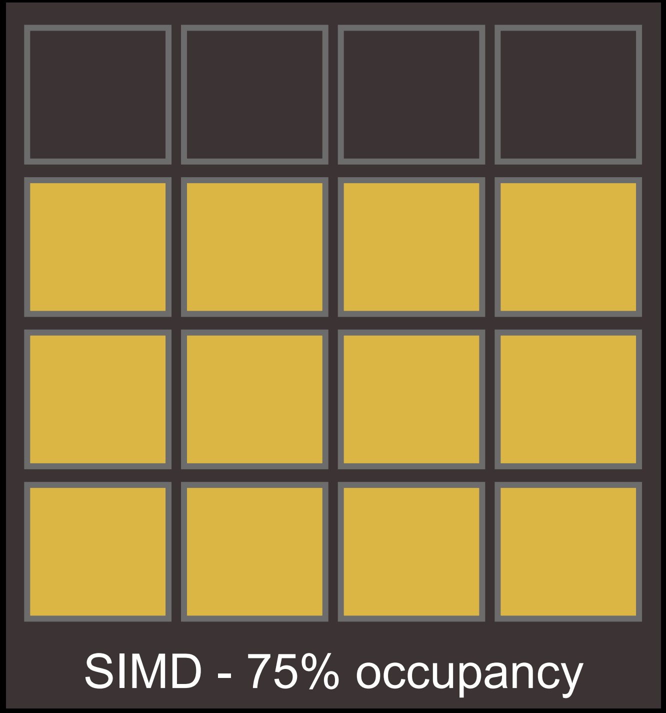
Occupancy is the ratio of assigned wavefronts to the maximum available slots


SIMD – 25% occupancy

Occupancy is the ratio of assigned wavefronts to the maximum available slots

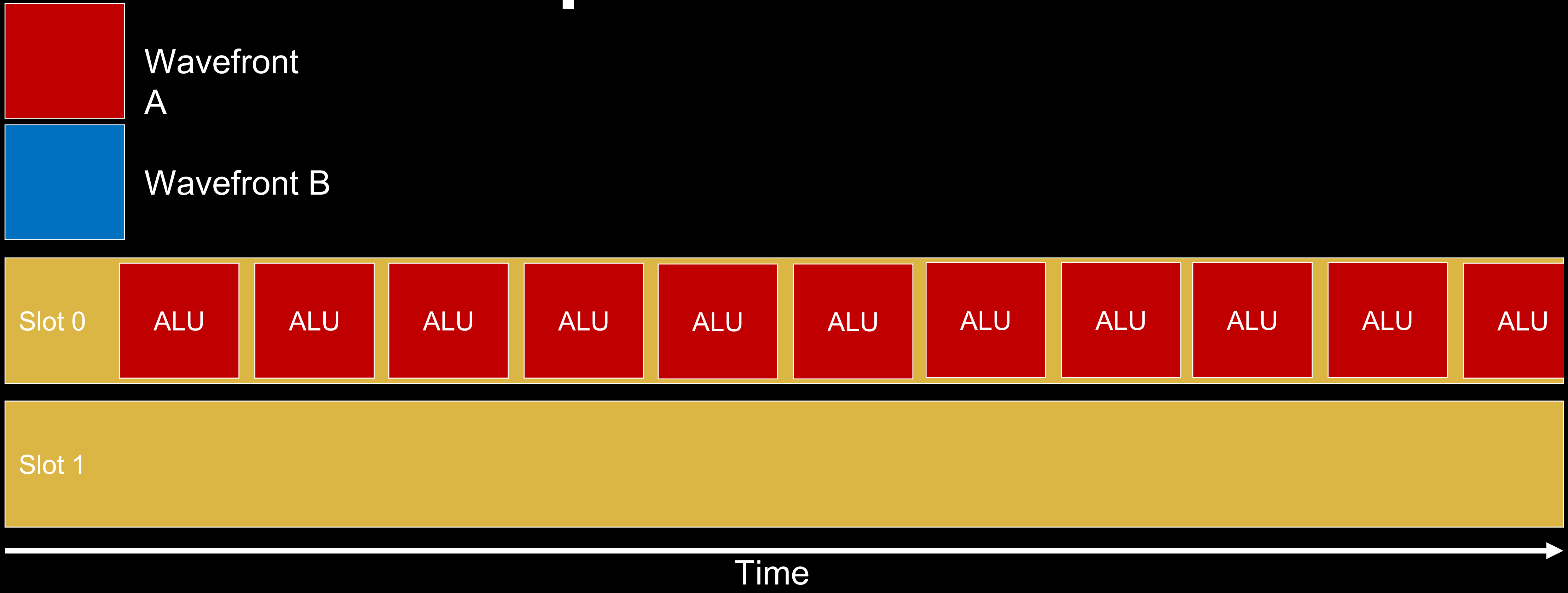


SIMD – 25% occupancy

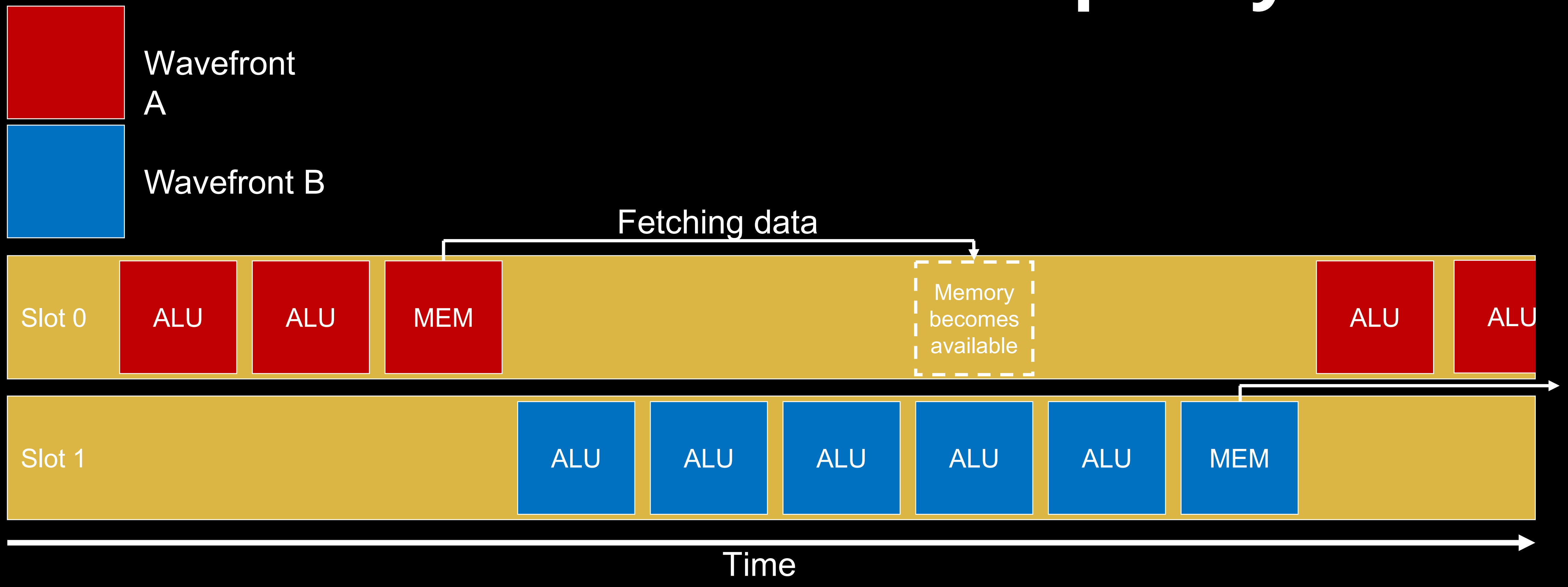


SIMD - 75% occupancy

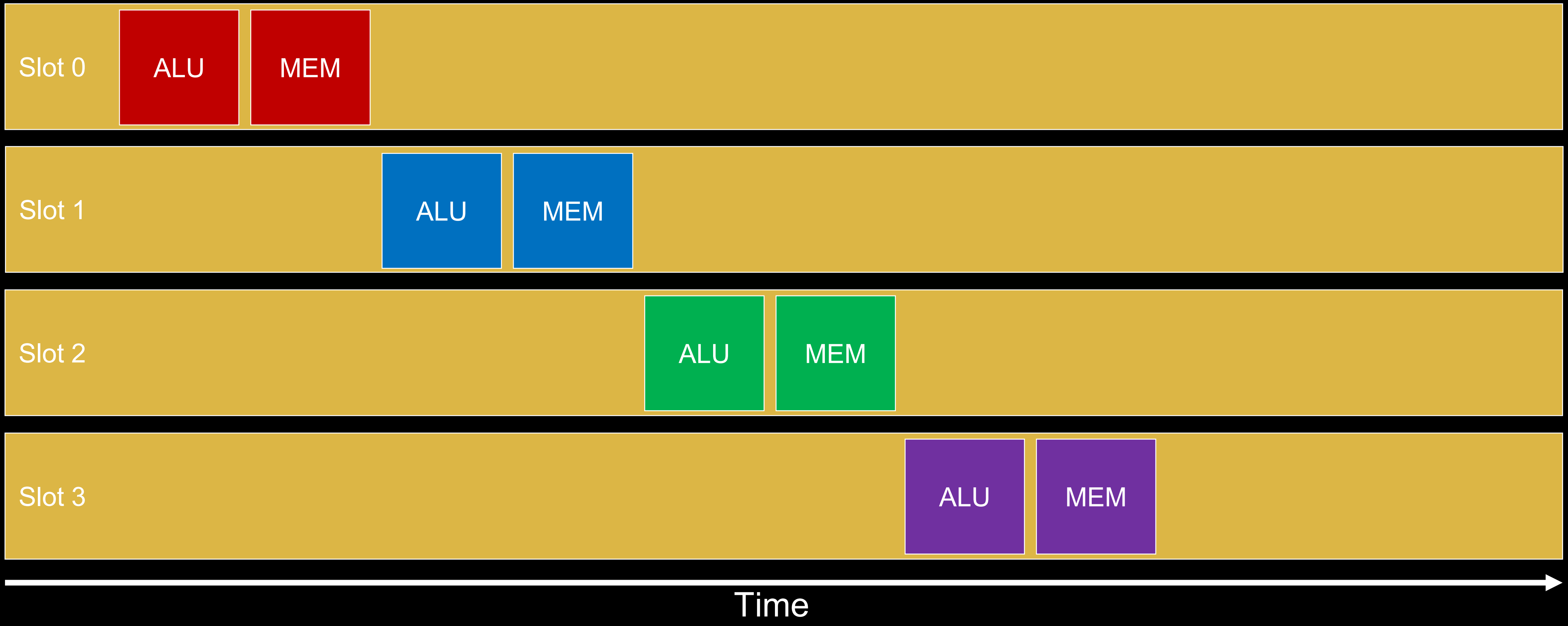
Better occupancy doesn't mean better performance !



Latency bound workloads *might* benefit from increased occupancy



In memory bound scenarios, increasing occupancy might thrash the caches



THEORETICAL OCCUPANCY – GPRS

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :
        SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```

THEORETICAL OCCUPANCY – GPRS

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :
       SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```

```
shader main
  asic(GFX10_3)
  type(CS)
  sgpr_count(6)
  vgpr_count(8)
  wave_size(32)
  s_version       UC_VERSION_GFX10 | UC_VERSION_W32_BIT
  s_inst_prefetch 0x0003
  s_getpc_b64     s[0:1]
  s_mov_b32       s0, s2
  s_load_dwordx4  s[4:7], s[0:1], null
  v_mad_u32_u24   v1, s3, 32, v0
  v_cmp_gt_u32    vcc_lo, 16, v1
  v_cndmask_b32   v2, 2, 1, vcc_lo
  v_mov_b32       v3, v2
  v_mov_b32       v4, v2
  v_mov_b32       v5, v2
  s_waitcnt       lgkmcnt(0)
  buffer_store_format_xyzw  v[2:5], v1, s[4:7], 0 idxen glc
  s_endpgm
```

THEORETICAL OCCUPANCY – GPRS

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :
        SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```

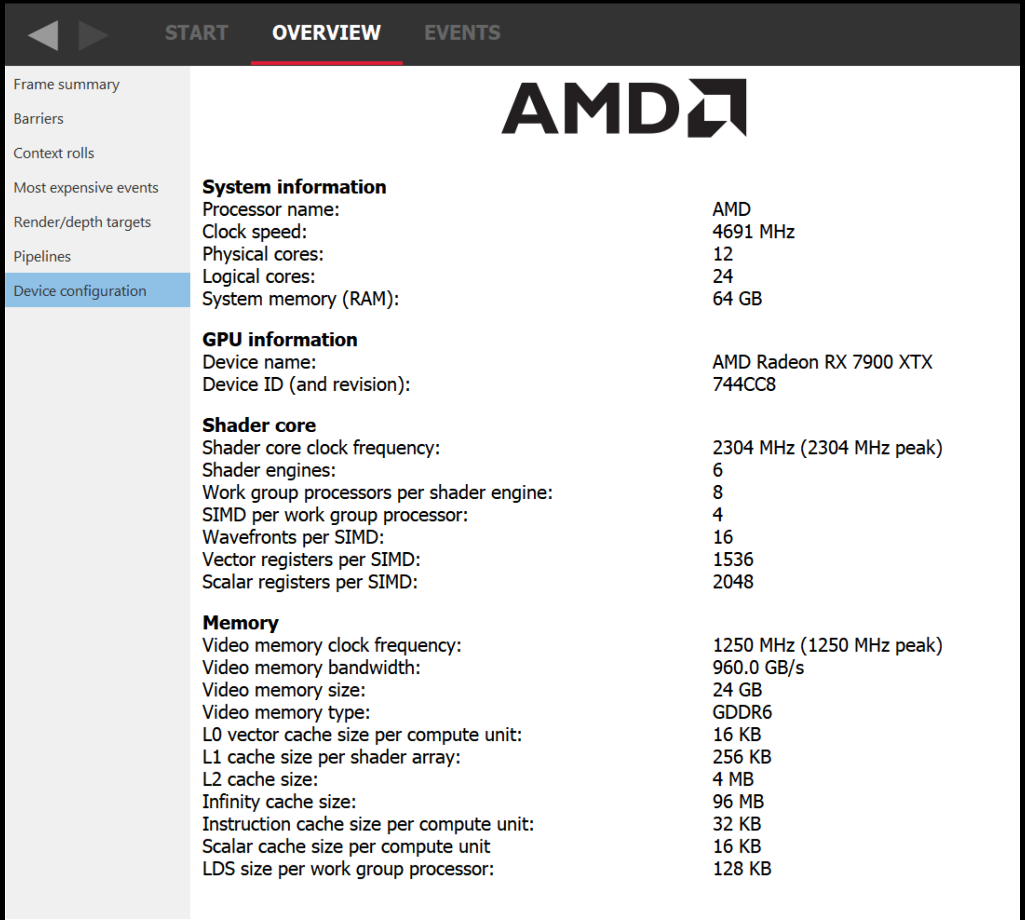
```
shader main
  asic(GFX10_3)
  type(CS)
  sgpr_count(6)
  vgpr_count(8)
  wave_size(32)
  s_version         UC_VERSION_GFX10 | UC_VERSION_W32_BIT
  s_inst_prefetch 0x0003
  s_getpc_b64       s[0:1]
  s_mov_b32         s0, s2
  s_load_dwordx4    s[4:7], s[0:1], null
  v_mad_u32_u24     v1, s3, 32, v0
  v_cmp_gt_u32      vcc_lo, 16, v1
  v_cndmask_b32     v2, 2, 1, vcc_lo
  v_mov_b32         v3, v2
  v_mov_b32         v4, v2
  v_mov_b32         v5, v2
  s_waitcnt         lgkmcnt(0)
  buffer_store_format_xyzw  v[2:5], v1, s[4:7], 0 idxen glc
  s_endpgm
```

THEORETICAL OCCUPANCY – GPRS

```
[numthreads(32, 1, 1)]
void CSMain( uint threadIndex :
        SV_DispatchThreadID )
{
    int sum = 0;
    if(threadIndex < 16)
    {
        sum += 1;
    }
    else
    {
        sum += 2;
    }

    data[threadIndex] = sum;
}
```

```
shader main
  asic(GFX10_3)
  type(CS)
  sgpr_count(6)
  vgpr_count(8)
  wave_size(32)
  s_version        UC_VERSION_GFX10 | UC_VERSION_W32_BIT
  s_inst_prefetch 0x0003
  s_getpc_b64      s[0:1]
  s_mov_b32        s0, s2
  s_load_dwordx4   s[4:7], s[0:1], null
  v_mad_u32_u24    v1, s3, 32, v0
  v_cmp_gt_u32     vcc_lo, 16, v1
  v_cndmask_b32    v2, 2, 1, vcc_lo
  v_mov_b32        v3, v2
  v_mov_b32        v4, v2
  v_mov_b32        v5, v2
  s_waitcnt        lgkmcnt(0)
  buffer_store_format_xyzw  v[2:5], v1, s[4:7], 0 idxen glc
  s_endpgm
```

RADEON™ GPU PROFILER TO THE RESCUE



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

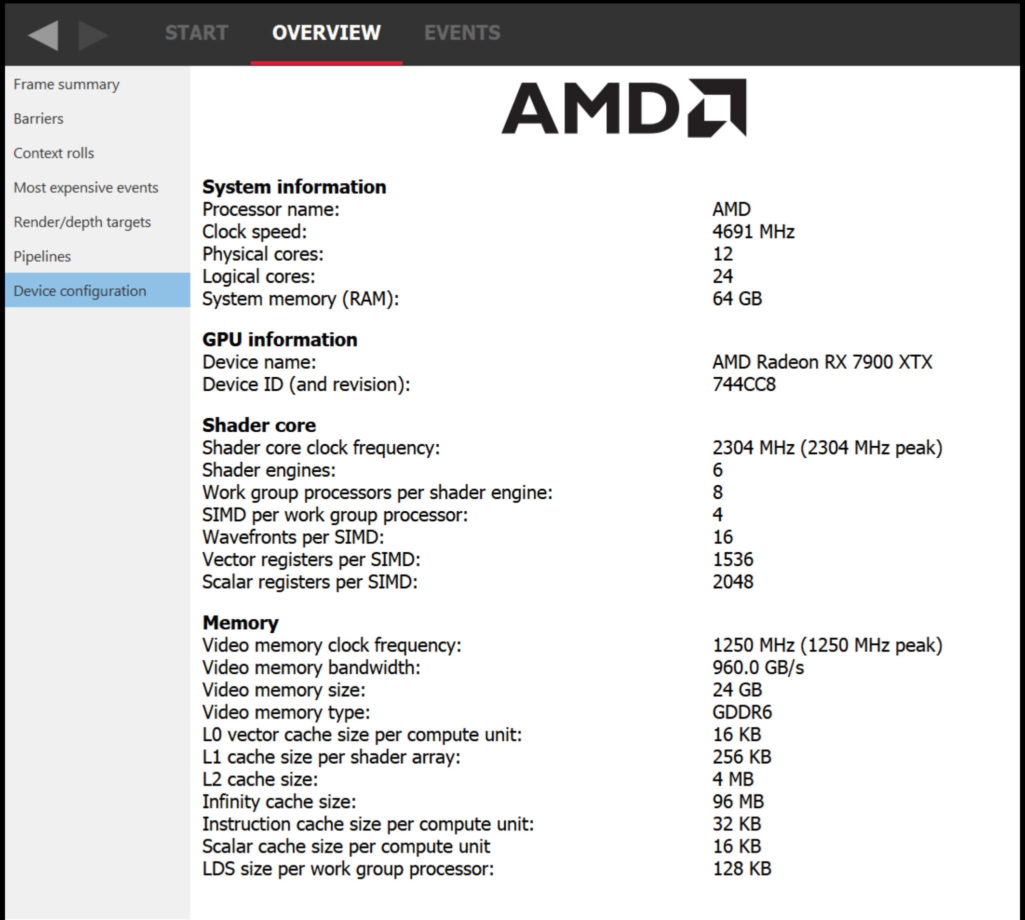
Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

For max occupancy

- Wave32 - 1536 / 16 = 96 VGPR per wave
- Wave64 - 1536 / 2 / 16 = 48 VGPR per wave

RADEON™ GPU PROFILER TO THE RESCUE



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

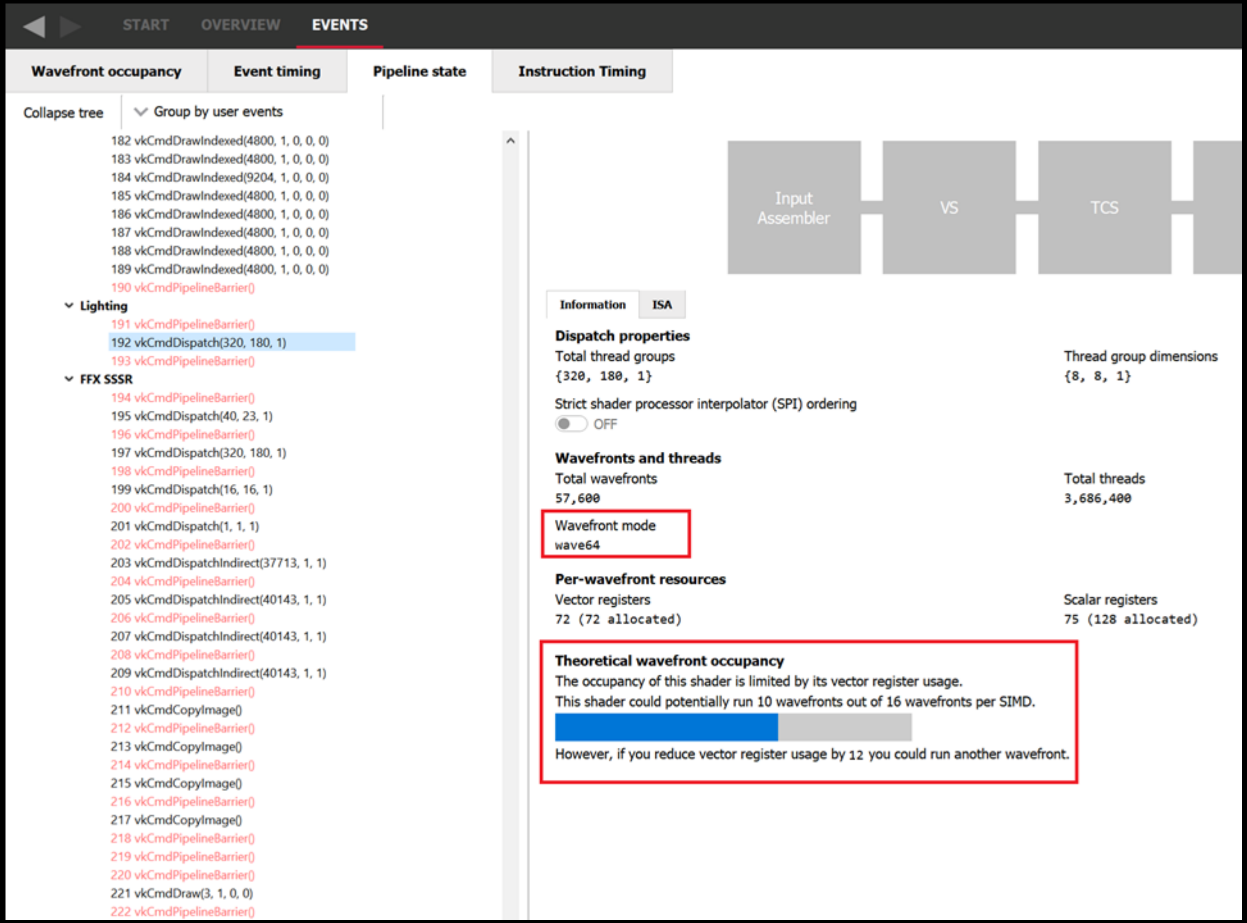
Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB



```
    182 vkCmdDrawIndexed(4800, 1, 0, 0, 0)
    183 vkCmdDrawIndexed(4800, 1, 0, 0, 0)
    184 vkCmdDrawIndexed(9204, 1, 0, 0, 0)
    185 vkCmdDrawIndexed(4800, 1, 0, 0, 0)
    186 vkCmdDrawIndexed(4800, 1, 0, 0, 0)
    187 vkCmdDrawIndexed(4800, 1, 0, 0, 0)
    188 vkCmdDrawIndexed(4800, 1, 0, 0, 0)
    189 vkCmdDrawIndexed(4800, 1, 0, 0, 0)
    190 vkCmdPipelineBarrier()
Lighting
    191 vkCmdPipelineBarrier()
    192 vkCmdDispatch(320, 180, 1)
    193 vkCmdPipelineBarrier()
FFX SSSR
    194 vkCmdPipelineBarrier()
    195 vkCmdDispatch(40, 23, 1)
    196 vkCmdPipelineBarrier()
    197 vkCmdDispatch(320, 180, 1)
    198 vkCmdPipelineBarrier()
    199 vkCmdDispatch(16, 16, 1)
    200 vkCmdPipelineBarrier()
    201 vkCmdDispatch(1, 1, 1)
    202 vkCmdPipelineBarrier()
    203 vkCmdDispatchIndirect(37713, 1, 1)
    204 vkCmdPipelineBarrier()
    205 vkCmdDispatchIndirect(40143, 1, 1)
    206 vkCmdPipelineBarrier()
    207 vkCmdDispatchIndirect(40143, 1, 1)
    208 vkCmdPipelineBarrier()
    209 vkCmdDispatchIndirect(40143, 1, 1)
    210 vkCmdPipelineBarrier()
    211 vkCmdCopyImage()
    212 vkCmdPipelineBarrier()
    213 vkCmdCopyImage()
    214 vkCmdPipelineBarrier()
    215 vkCmdCopyImage()
    216 vkCmdPipelineBarrier()
    217 vkCmdCopyImage()
    218 vkCmdPipelineBarrier()
    219 vkCmdPipelineBarrier()
    220 vkCmdPipelineBarrier()
    221 vkCmdDraw(3, 1, 0, 0)
    222 vkCmdPipelineBarrier()
```

Dispatch properties

Total thread groups
`{320, 180, 1}`

Thread group dimensions
`{8, 8, 1}`

Strict shader processor interpolator (SPI) ordering
○ OFF

Wavefronts and threads

Total wavefronts
`57,600`

Total threads
`3,686,400`

Wavefront mode
`wave64`

Per-wavefront resources

Vector registers
`72 (72 allocated)`

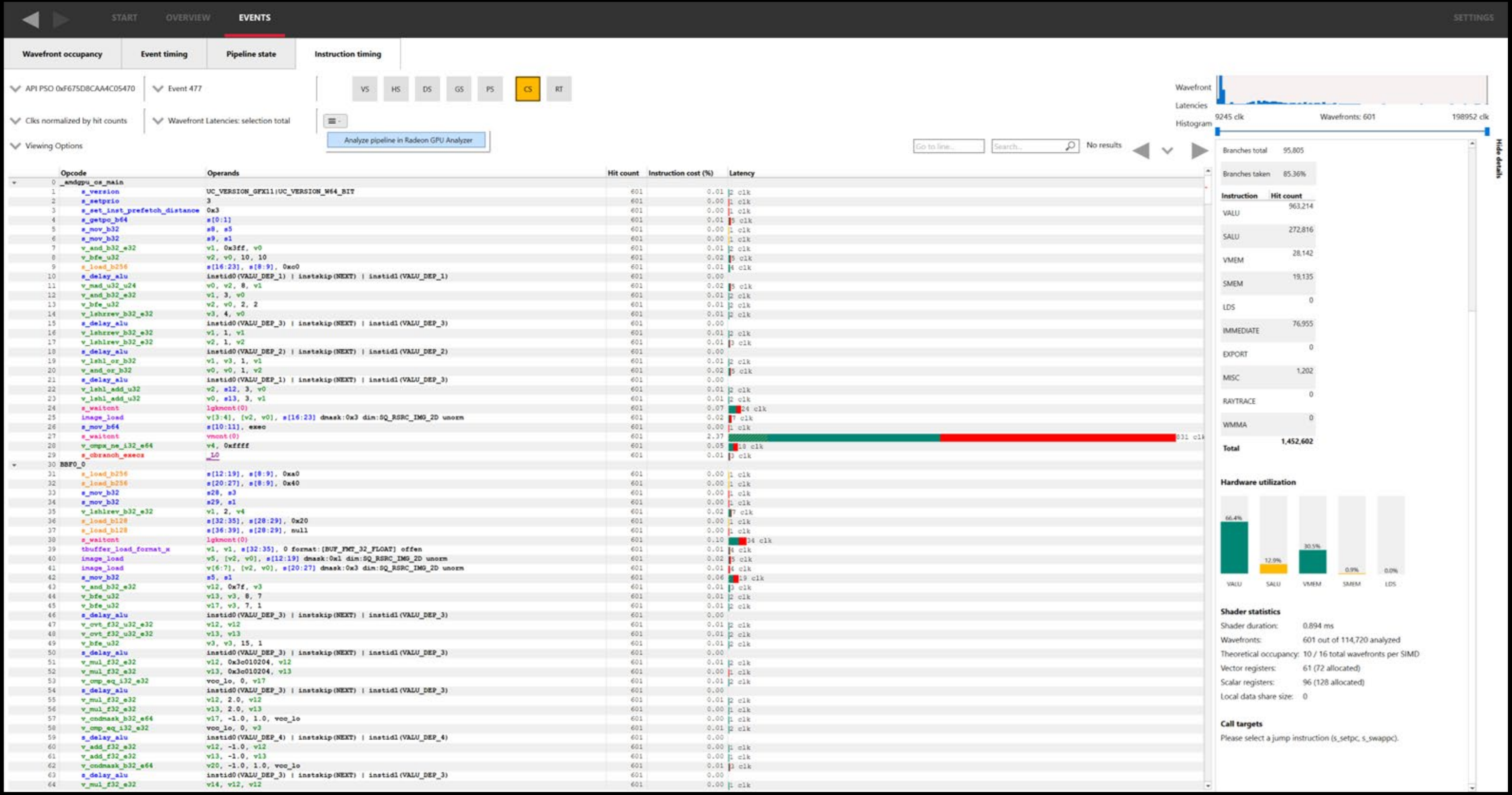
Scalar registers
`75 (128 allocated)`

Theoretical wavefront occupancy

The occupancy of this shader is limited by its vector register usage.
This shader could potentially run 10 wavefronts out of 16 wavefronts per SIMD.

However, if you reduce vector register usage by 12 you could run another wavefront.

RADEON™ GPU ANALYZER TO THE RESCUE



```
    Opcode                      Operands                                                              Hit count  Instruction cost (%)  Latency
 0  _amdgpu_cs_main
 1      s_version               UC_VERSION_GFX11|UC_VERSION_W64_BIT                                      601        0.01              2 clk
 2      s_setprio               3                                                                        601        0.00              1 clk
 3      s_set_inst_prefetch_distance 0x3                                                                 601        0.00              1 clk
 4      s_getpc_b64             s[0:1]                                                                   601        0.01              5 clk
 5      s_mov_b32               s8, s5                                                                   601        0.00              1 clk
 6      s_mov_b32               s9, s1                                                                   601        0.00              1 clk
 7      v_and_b32_e32           v1, 0x3ff, v0                                                            601        0.01              2 clk
 8      v_bfe_u32               v2, v0, 10, 10                                                           601        0.02              5 clk
 9      s_load_b256             s[16:23], s[8:9], 0xc0                                                   601        0.01              4 clk
10      s_delay_alu             instid0(VALU_DEP_1) | instskip(NEXT) | instid1(VALU_DEP_1)               601        0.00
11      v_mad_u32_u24           v0, v2, 8, v1                                                            601        0.02              5 clk
12      v_and_b32_e32           v1, 3, v0                                                                601        0.01              2 clk
13      v_bfe_u32               v2, v0, 2, 2                                                             601        0.01              2 clk
14      v_lshrrev_b32_e32       v3, 4, v0                                                                601        0.01              2 clk
15      s_delay_alu             instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)               601        0.00
16      v_lshrrev_b32_e32       v1, 1, v1                                                                601        0.01              2 clk
17      v_lshlrev_b32_e32       v2, 1, v2                                                                601        0.01              3 clk
18      s_delay_alu             instid0(VALU_DEP_2) | instskip(NEXT) | instid1(VALU_DEP_2)               601        0.00
19      v_lshl_or_b32           v1, v3, 1, v1                                                            601        0.01              2 clk
20      v_and_or_b32            v0, v0, 1, v2                                                            601        0.02              5 clk
21      s_delay_alu             instid0(VALU_DEP_1) | instskip(NEXT) | instid1(VALU_DEP_3)               601        0.00
22      v_lshl_add_u32          v2, s12, 3, v0                                                           601        0.01              2 clk
23      v_lshl_add_u32          v0, s13, 3, v1                                                           601        0.01              2 clk
24      s_waitcnt               lgkmcnt(0)                                                               601        0.07              24 clk
25      image_load              v[3:4], [v2, v0], s[16:23] dmask:0x3 dim:SQ_RSRC_IMG_2D unorm            601        0.02              7 clk
26      s_mov_b64               s[10:11], exec                                                           601        0.00              1 clk
27      s_waitcnt               vmcnt(0)                                                                 601        2.37              831 clk
28      v_cmpx_ne_i32_e64       v4, 0xffff                                                               601        0.05              18 clk
29      s_cbranch_execz         _L0                                                                      601        0.01              3 clk
30  BBF0_0
31      s_load_b256             s[12:19], s[8:9], 0xa0                                                   601        0.00              1 clk
32      s_load_b256             s[20:27], s[8:9], 0x40                                                   601        0.00              1 clk
33      s_mov_b32               s28, s3                                                                  601        0.00              1 clk
34      s_mov_b32               s29, s1                                                                  601        0.00              1 clk
35      v_lshlrev_b32_e32       v1, 2, v4                                                                601        0.02              7 clk
36      s_load_b128             s[32:35], s[28:29], 0x20                                                 601        0.00              1 clk
37      s_load_b128             s[36:39], s[28:29], null                                                 601        0.00              1 clk
38      s_waitcnt               lgkmcnt(0)                                                               601        0.10              34 clk
39      tbuffer_load_format_x   v1, v1, s[32:35], 0 format:[BUF_FMT_32_FLOAT] offen                      601        0.01              4 clk
40      image_load              v5, [v2, v0], s[12:19] dmask:0x1 dim:SQ_RSRC_IMG_2D unorm                601        0.02              5 clk
41      image_load              v[6:7], [v2, v0], s[20:27] dmask:0x3 dim:SQ_RSRC_IMG_2D unorm            601        0.01              4 clk
42      s_mov_b32               s5, s1                                                                   601        0.06              19 clk
43      v_and_b32_e32           v12, 0x7f, v3                                                            601        0.01              3 clk
44      v_bfe_u32               v13, v3, 8, 7                                                            601        0.01              2 clk
45      v_bfe_u32               v17, v3, 7, 1                                                            601        0.01              2 clk
46      s_delay_alu             instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)               601        0.00
47      v_cvt_f32_u32_e32       v12, v12                                                                 601        0.01              2 clk
48      v_cvt_f32_u32_e32       v13, v13                                                                 601        0.01              2 clk
49      v_bfe_u32               v3, v3, 15, 1                                                            601        0.01              2 clk
50      s_delay_alu             instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)               601        0.00
51      v_mul_f32_e32           v12, 0x3c010204, v12                                                     601        0.01              2 clk
52      v_mul_f32_e32           v13, 0x3c010204, v13                                                     601        0.00              1 clk
53      v_cmp_eq_i32_e32        vcc_lo, 0, v17                                                           601        0.01              2 clk
54      s_delay_alu             instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)               601        0.00
55      v_mul_f32_e32           v12, 2.0, v12                                                            601        0.01              2 clk
56      v_mul_f32_e32           v13, 2.0, v13                                                            601        0.00              1 clk
57      v_cndmask_b32_e64       v17, -1.0, 1.0, vcc_lo                                                   601        0.00              1 clk
58      v_cmp_eq_i32_e32        vcc_lo, 0, v3                                                            601        0.01              2 clk
59      s_delay_alu             instid0(VALU_DEP_4) | instskip(NEXT) | instid1(VALU_DEP_4)               601        0.00
60      v_add_f32_e32           v12, -1.0, v12                                                           601        0.00              1 clk
61      v_add_f32_e32           v13, -1.0, v13                                                           601        0.00              1 clk
62      v_cndmask_b32_e64       v20, -1.0, 1.0, vcc_lo                                                   601        0.01              3 clk
63      s_delay_alu             instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)               601        0.00
64      v_mul_f32_e32           v14, v12, v12                                                            601        0.00              1 clk
```

Branches total	95,805
Branches taken	85.36%

Instruction	Hit count
VALU	963,214
SALU	272,816
VMEM	28,142
SMEM	19,135
LDS	0
IMMEDIATE	76,955
EXPORT	0
MISC	1,202
RAYTRACE	0
WMMA	0
Total	**1,452,602**

Hardware utilization

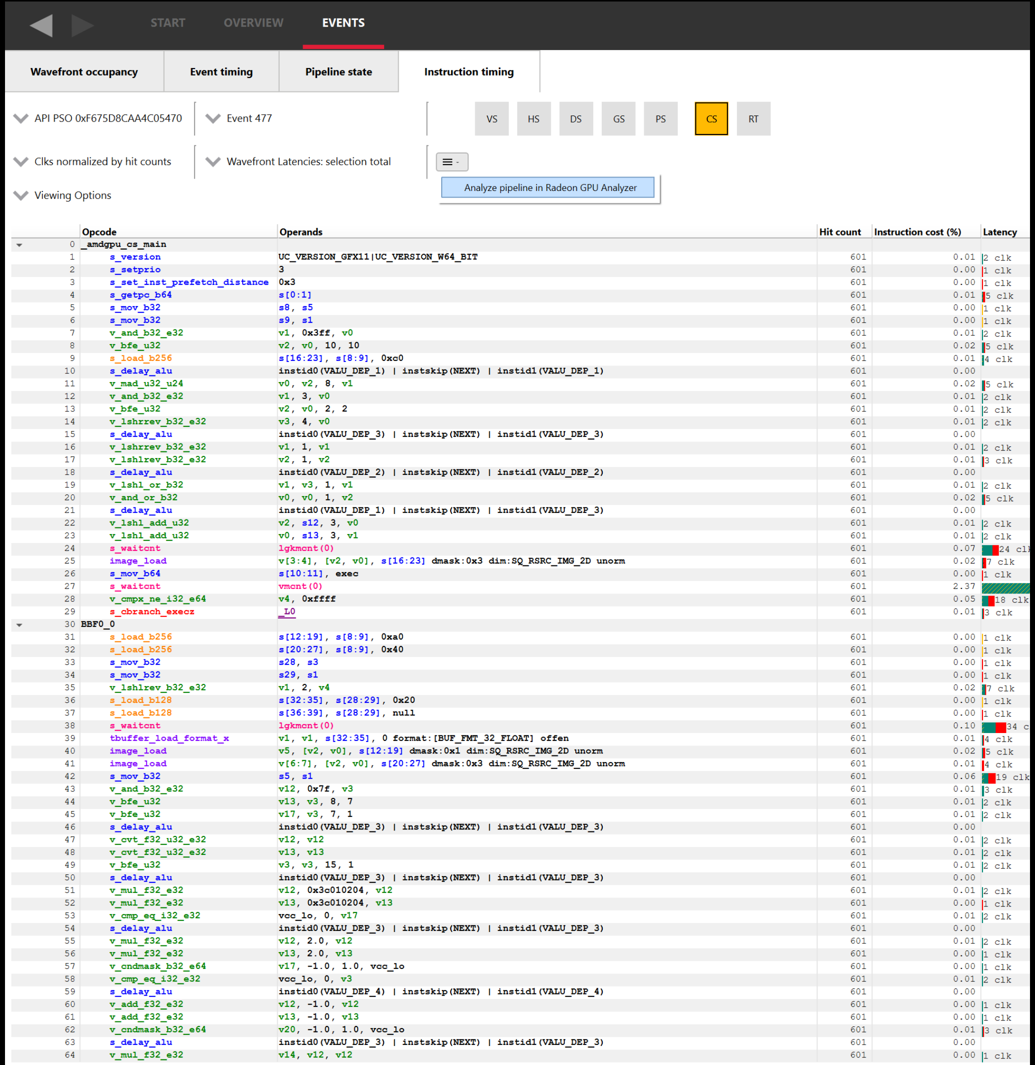
Shader statistics

Shader duration: 0.894 ms
Wavefronts: 601 out of 114,720 analyzed
Theoretical occupancy: 10 / 16 total wavefronts per SIMD
Vector registers: 61 (72 allocated)
Scalar registers: 96 (128 allocated)
Local data share size: 0

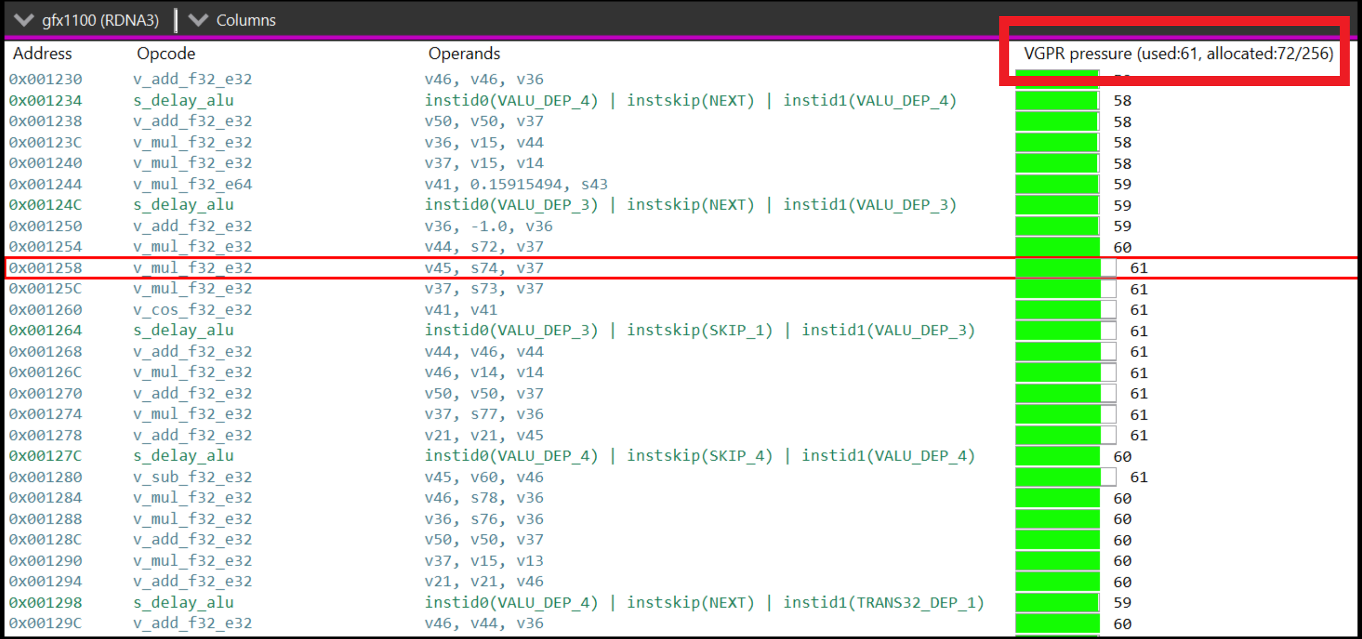
Call targets

Please select a jump instruction (s_setpc, s_swappc).

RADEON™ GPU ANALYZER TO THE RESCUE

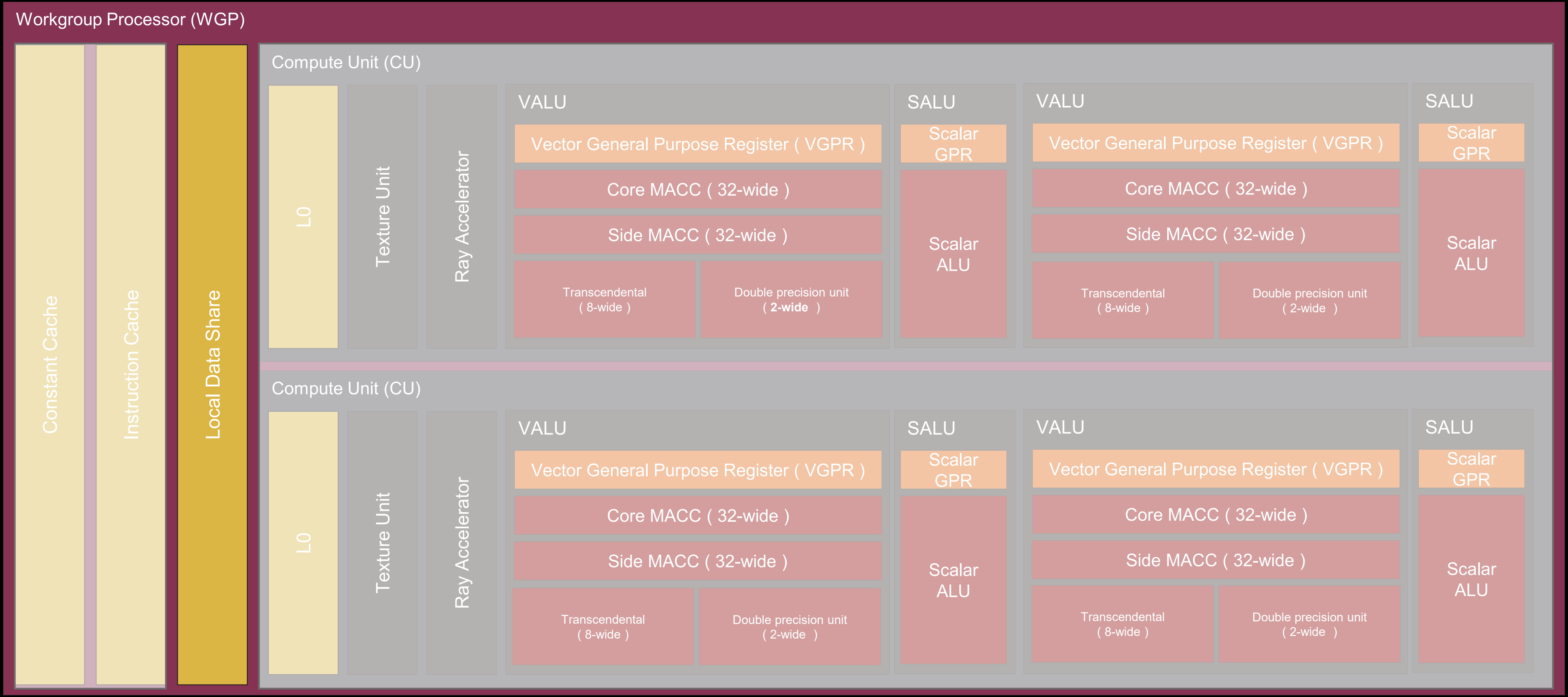

START
OVERVIEW
EVENTS
Wavefront occupancy
Event timing
Pipeline state
Instruction timing
API PSO 0xF675D8CAA4C05470
Event 477
VS
HS
DS
GS
PS
CS
RT
Clks normalized by hit counts
Wavefront Latencies: selection total
Analyze pipeline in Radeon GPU Analyzer
Viewing Options

	Opcode	Operands	Hit count	Instruction cost (%)	Latency
0	_amdgpu_cs_main				
1	s_version	UC_VERSION_GFX11|UC_VERSION_W64_BIT	601	0.01	2 clk
2	s_setprio	3	601	0.00	1 clk
3	s_set_inst_prefetch_distance	0x3	601	0.00	1 clk
4	s_getpc_b64	s[0:1]	601	0.01	5 clk
5	s_mov_b32	s8, s5	601	0.00	1 clk
6	s_mov_b32	s9, s1	601	0.00	1 clk
7	v_and_b32_e32	v1, 0x3ff, v0	601	0.01	2 clk
8	v_bfe_u32	v2, v0, 10, 10	601	0.02	5 clk
9	s_load_b256	s[16:23], s[8:9], 0xc0	601	0.01	4 clk
10	s_delay_alu	instid0(VALU_DEP_1) | instskip(NEXT) | instid1(VALU_DEP_1)	601	0.00	
11	v_mad_u32_u24	v0, v2, 8, v1	601	0.02	5 clk
12	v_and_b32_e32	v1, 3, v0	601	0.01	2 clk
13	v_bfe_u32	v2, v0, 2, 2	601	0.01	2 clk
14	v_lshrrev_b32_e32	v3, 4, v0	601	0.01	2 clk
15	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)	601	0.00	
16	v_lshrrev_b32_e32	v1, 1, v1	601	0.01	2 clk
17	v_lshlrev_b32_e32	v2, 1, v2	601	0.01	3 clk
18	s_delay_alu	instid0(VALU_DEP_2) | instskip(NEXT) | instid1(VALU_DEP_2)	601	0.00	
19	v_lshl_or_b32	v1, v3, 1, v1	601	0.01	2 clk
20	v_and_or_b32	v0, v0, 1, v2	601	0.02	5 clk
21	s_delay_alu	instid0(VALU_DEP_1) | instskip(NEXT) | instid1(VALU_DEP_3)	601	0.00	
22	v_lshl_add_u32	v2, s12, 3, v0	601	0.01	2 clk
23	v_lshl_add_u32	v0, s13, 3, v1	601	0.01	2 clk
24	s_waitcnt	lgkmcnt(0)	601	0.07	24 cl
25	image_load	v[3:4], [v2, v0], s[16:23] dmask:0x3 dim:SQ_RSRC_IMG_2D unorm	601	0.02	7 clk
26	s_mov_b64	s[10:11], exec	601	0.00	1 clk
27	s_waitcnt	vmcnt(0)	601	2.37	
28	v_cmpx_ne_i32_e64	v4, 0xffff	601	0.05	18 clk
29	s_cbranch_execz	_L0	601	0.01	3 clk
30	BBF0_0				
31	s_load_b256	s[12:19], s[8:9], 0xa0	601	0.00	1 clk
32	s_load_b256	s[20:27], s[8:9], 0x40	601	0.00	1 clk
33	s_mov_b32	s28, s3	601	0.00	1 clk
34	s_mov_b32	s29, s1	601	0.00	1 clk
35	v_lshlrev_b32_e32	v1, 2, v4	601	0.02	7 clk
36	s_load_b128	s[32:35], s[28:29], 0x20	601	0.00	1 clk
37	s_load_b128	s[36:39], s[28:29], null	601	0.00	1 clk
38	s_waitcnt	lgkmcnt(0)	601	0.10	34 c
39	tbuffer_load_format_x	v1, v1, s[32:35], 0 format:[BUF_FMT_32_FLOAT] offen	601	0.01	4 clk
40	image_load	v5, [v2, v0], s[12:19] dmask:0x1 dim:SQ_RSRC_IMG_2D unorm	601	0.02	5 clk
41	image_load	v[6:7], [v2, v0], s[20:27] dmask:0x3 dim:SQ_RSRC_IMG_2D unorm	601	0.01	4 clk
42	s_mov_b32	s5, s1	601	0.06	19 clk
43	v_and_b32_e32	v12, 0x7f, v3	601	0.01	3 clk
44	v_bfe_u32	v13, v3, 8, 7	601	0.01	2 clk
45	v_bfe_u32	v17, v3, 7, 1	601	0.01	2 clk
46	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)	601	0.00	
47	v_cvt_f32_u32_e32	v12, v12	601	0.01	2 clk
48	v_cvt_f32_u32_e32	v13, v13	601	0.01	2 clk
49	v_bfe_u32	v3, v3, 15, 1	601	0.01	2 clk
50	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)	601	0.00	
51	v_mul_f32_e32	v12, 0x3c010204, v12	601	0.01	2 clk
52	v_mul_f32_e32	v13, 0x3c010204, v13	601	0.00	1 clk
53	v_cmp_eq_i32_e32	vcc_lo, 0, v17	601	0.01	2 clk
54	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)	601	0.00	
55	v_mul_f32_e32	v12, 2.0, v12	601	0.01	2 clk
56	v_mul_f32_e32	v13, 2.0, v13	601	0.00	1 clk
57	v_cndmask_b32_e64	v17, -1.0, 1.0, vcc_lo	601	0.00	1 clk
58	v_cmp_eq_i32_e32	vcc_lo, 0, v3	601	0.01	2 clk
59	s_delay_alu	instid0(VALU_DEP_4) | instskip(NEXT) | instid1(VALU_DEP_4)	601	0.00	
60	v_add_f32_e32	v12, -1.0, v12	601	0.00	1 clk
61	v_add_f32_e32	v13, -1.0, v13	601	0.00	1 clk
62	v_cndmask_b32_e64	v20, -1.0, 1.0, vcc_lo	601	0.01	3 clk
63	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)	601	0.00	
64	v_mul_f32_e32	v14, v12, v12	601	0.00	1 clk

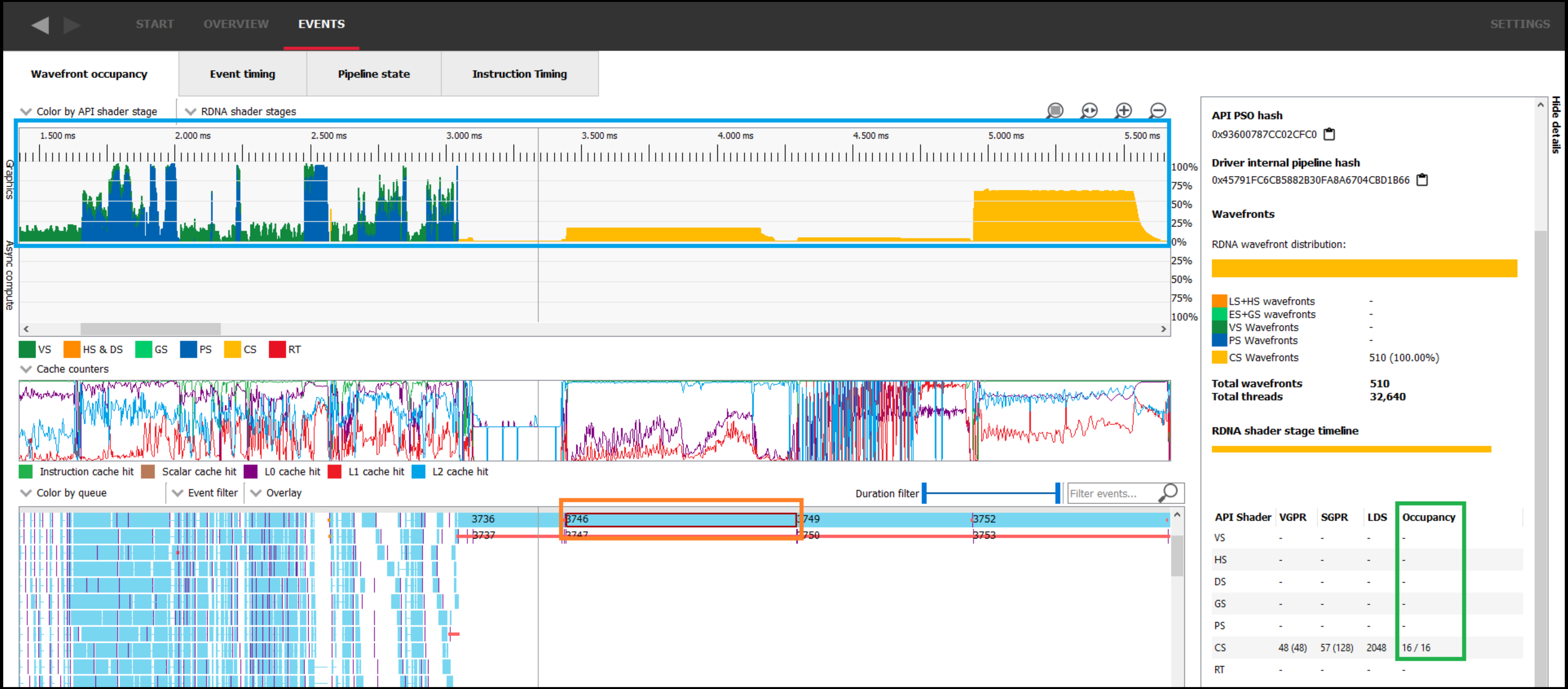


Address	Opcode	Operands	VGPR pressure (used:61, allocated:72/256)
0x001230	v_add_f32_e32	v46, v46, v36	[illegible]
0x001234	s_delay_alu	instid0(VALU_DEP_4) | instskip(NEXT) | instid1(VALU_DEP_4)	58
0x001238	v_add_f32_e32	v50, v50, v37	58
0x00123C	v_mul_f32_e32	v36, v15, v44	58
0x001240	v_mul_f32_e32	v37, v15, v14	58
0x001244	v_mul_f32_e64	v41, 0.15915494, s43	59
0x00124C	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)	59
0x001250	v_add_f32_e32	v36, -1.0, v36	59
0x001254	v_mul_f32_e32	v44, s72, v37	60
0x001258	v_mul_f32_e32	v45, s74, v37	61
0x00125C	v_mul_f32_e32	v37, s73, v37	61
0x001260	v_cos_f32_e32	v41, v41	61
0x001264	s_delay_alu	instid0(VALU_DEP_3) | instskip(SKIP_1) | instid1(VALU_DEP_3)	61
0x001268	v_add_f32_e32	v44, v46, v44	61
0x00126C	v_mul_f32_e32	v46, v14, v14	61
0x001270	v_add_f32_e32	v50, v50, v37	61
0x001274	v_mul_f32_e32	v37, s77, v36	61
0x001278	v_add_f32_e32	v21, v21, v45	61
0x00127C	s_delay_alu	instid0(VALU_DEP_4) | instskip(SKIP_4) | instid1(VALU_DEP_4)	60
0x001280	v_sub_f32_e32	v45, v60, v46	61
0x001284	v_mul_f32_e32	v46, s78, v36	60
0x001288	v_mul_f32_e32	v36, s76, v36	60
0x00128C	v_add_f32_e32	v50, v50, v37	60
0x001290	v_mul_f32_e32	v37, v15, v13	60
0x001294	v_add_f32_e32	v21, v21, v46	60
0x001298	s_delay_alu	instid0(VALU_DEP_4) | instskip(NEXT) | instid1(TRANS32_DEP_1)	59
0x00129C	v_add_f32_e32	v46, v44, v36	60

THEORETICAL OCCUPANCY – LDS & THREADGROUP_SIZE



MEASURED OCCUPANCY



API PSO hash

0x93600787CC02CFC0

Driver internal pipeline hash

0x45791FC6CB5882B30FA8A6704CBD1B66

Wavefronts

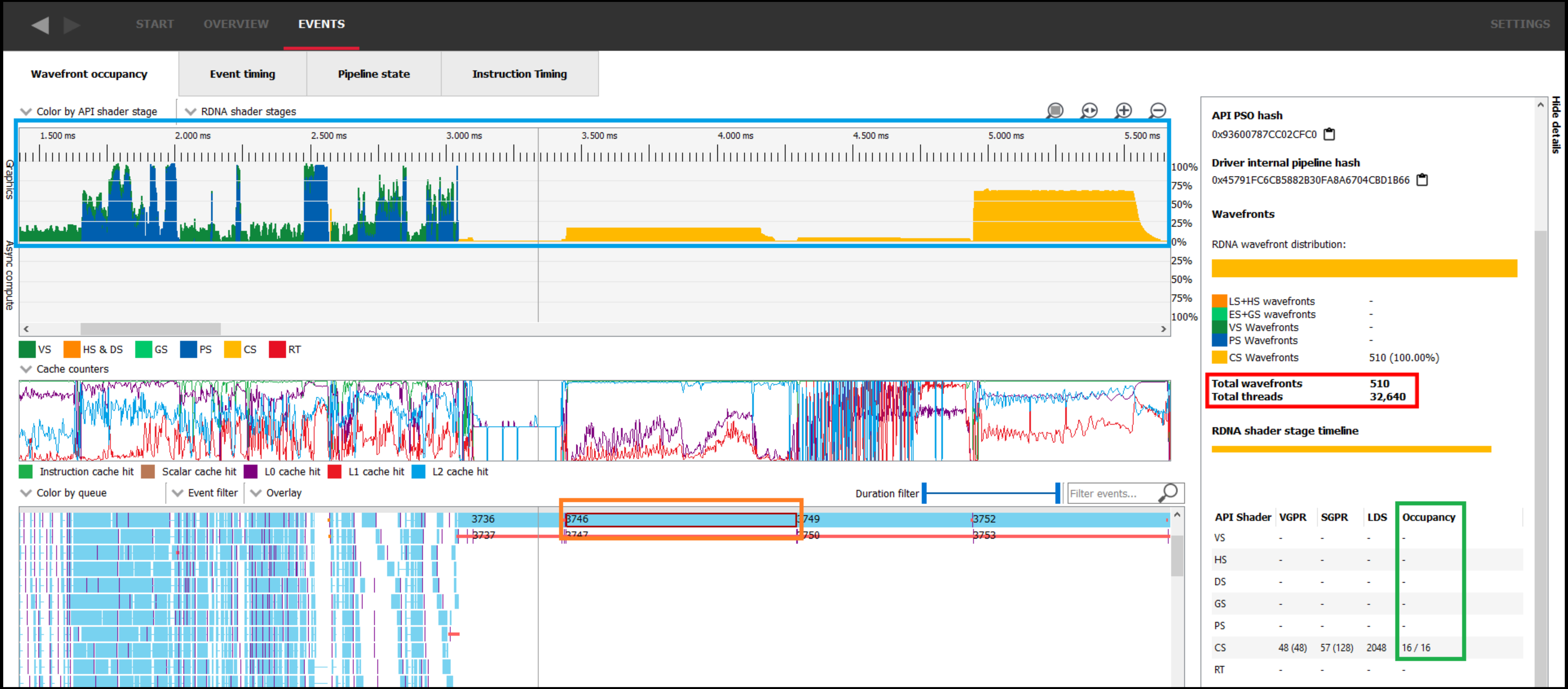
RDNA wavefront distribution:

🟧 LS+HS wavefronts	-
🟩 ES+GS wavefronts	-
🟩 VS Wavefronts	-
🟦 PS Wavefronts	-
🟨 CS Wavefronts	510 (100.00%)
Total wavefronts	**510**
Total threads	**32,640**

RDNA shader stage timeline

API Shader	VGPR	SGPR	LDS	Occupancy
VS	-	-	-	-
HS	-	-	-	-
DS	-	-	-	-
GS	-	-	-	-
PS	-	-	-	-
CS	48 (48)	57 (128)	2048	16 / 16
RT	-	-	-	-

MEASURED OCCUPANCY



API PSO hash

0x93600787CC02CFC0

Driver internal pipeline hash

0x45791FC6CB5882B30FA8A6704CBD1B66

Wavefronts

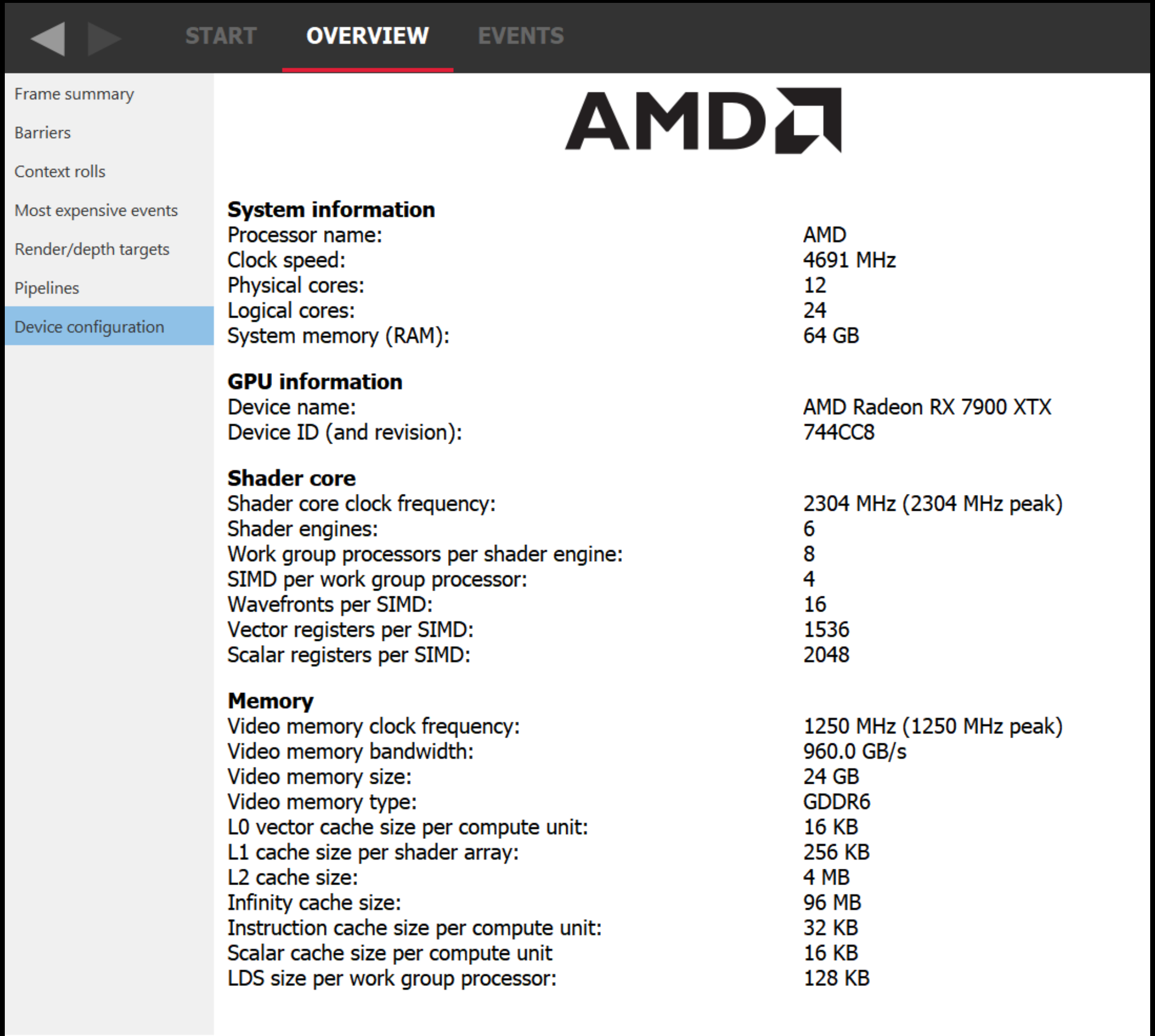
RDNA wavefront distribution:

🟧 LS+HS wavefronts	-
🟩 ES+GS wavefronts	-
🟩 VS Wavefronts	-
🟦 PS Wavefronts	-
🟨 CS Wavefronts	510 (100.00%)
Total wavefronts	**510**
Total threads	**32,640**

RDNA shader stage timeline

API Shader	VGPR	SGPR	LDS	Occupancy
VS	-	-	-	-
HS	-	-	-	-
DS	-	-	-	-
GS	-	-	-	-
PS	-	-	-	-
CS	48 (48)	57 (128)	2048	16 / 16
RT	-	-	-	-

LACK OF WORK LIMITED OCCUPANCY



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

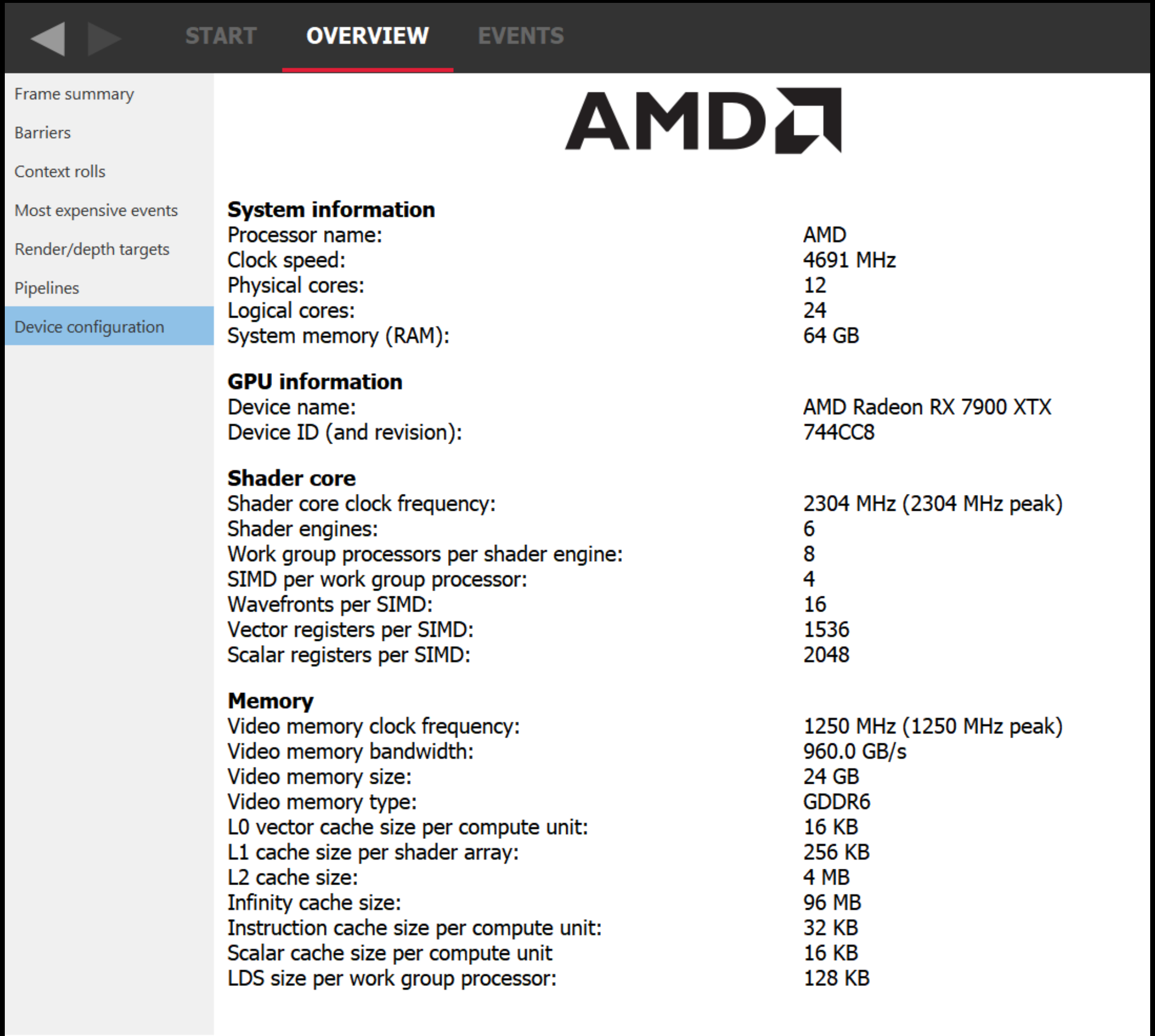
Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6

LACK OF WORK LIMITED OCCUPANCY



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

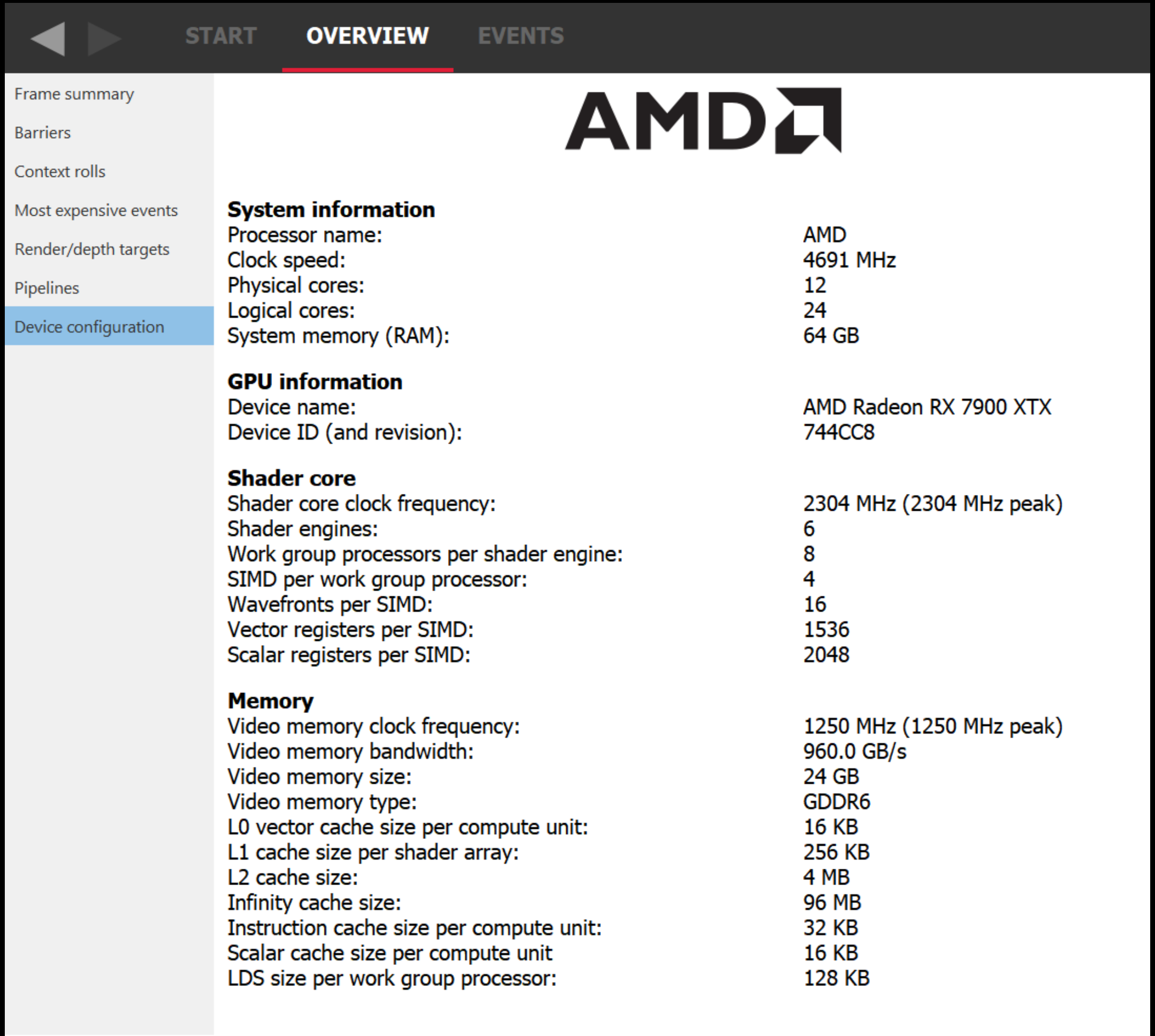
Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6
WorkGroup Processors (WGP) / SE: 8

LACK OF WORK LIMITED OCCUPANCY



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

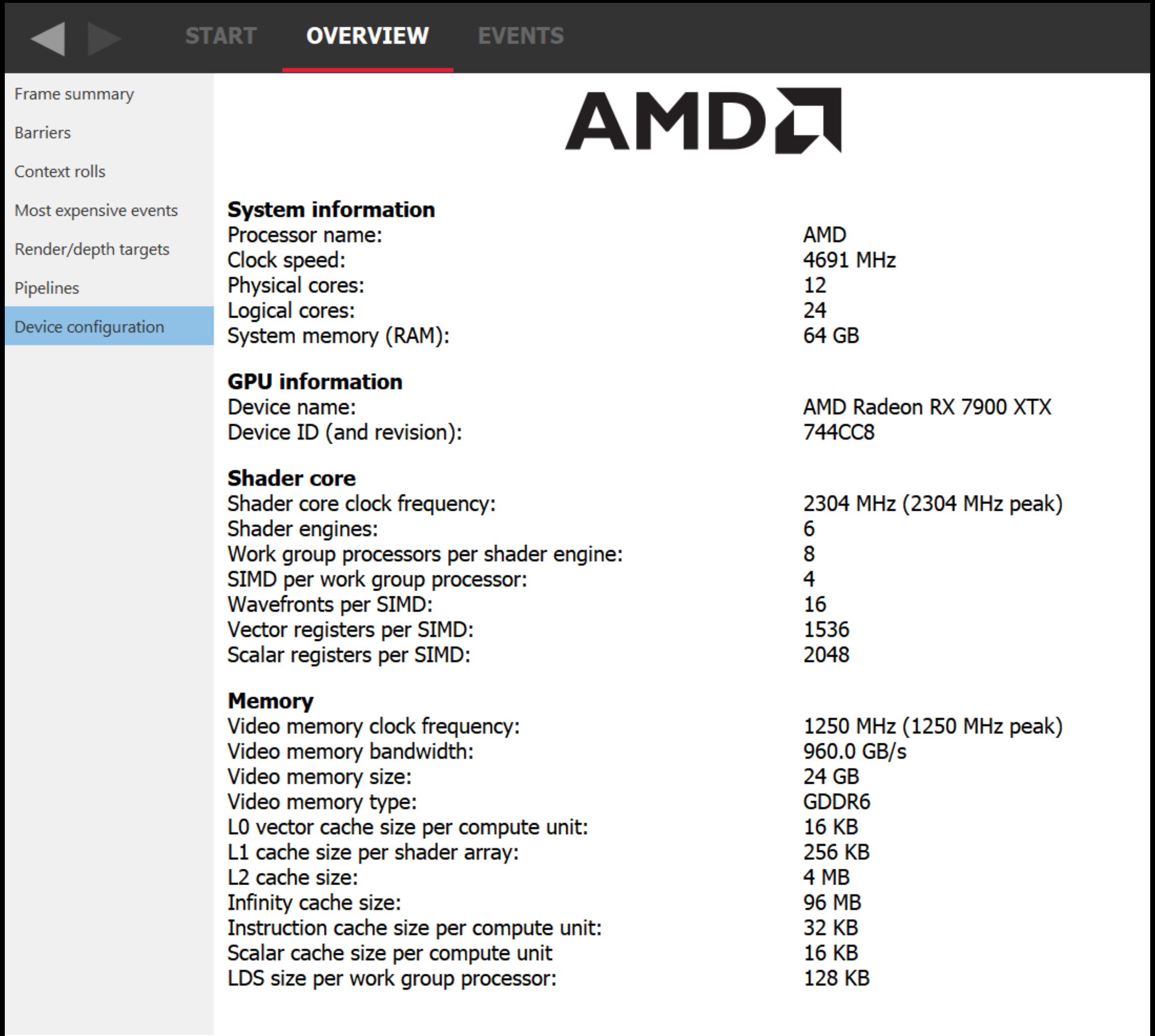
Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6
WorkGroup Processors (WGP) / SE: 8
Total WGP: 6 * 8 = 48

LACK OF WORK LIMITED OCCUPANCY



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

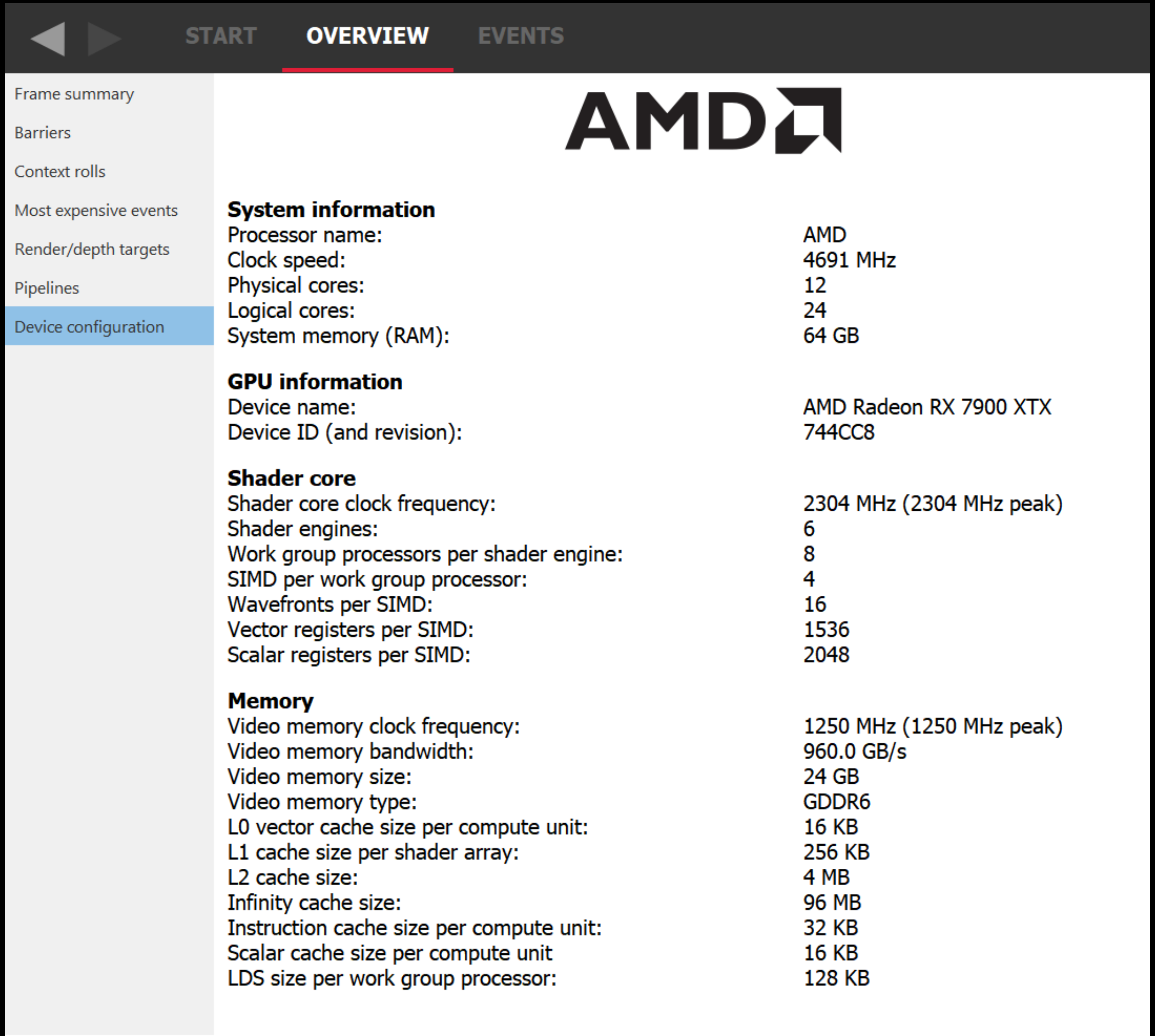
Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6
WorkGroup Processors (WGP) / SE: 8
Total WGP: 6 * 8 = 48
SIMD per WGP: 4

LACK OF WORK LIMITED OCCUPANCY



System information	
Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

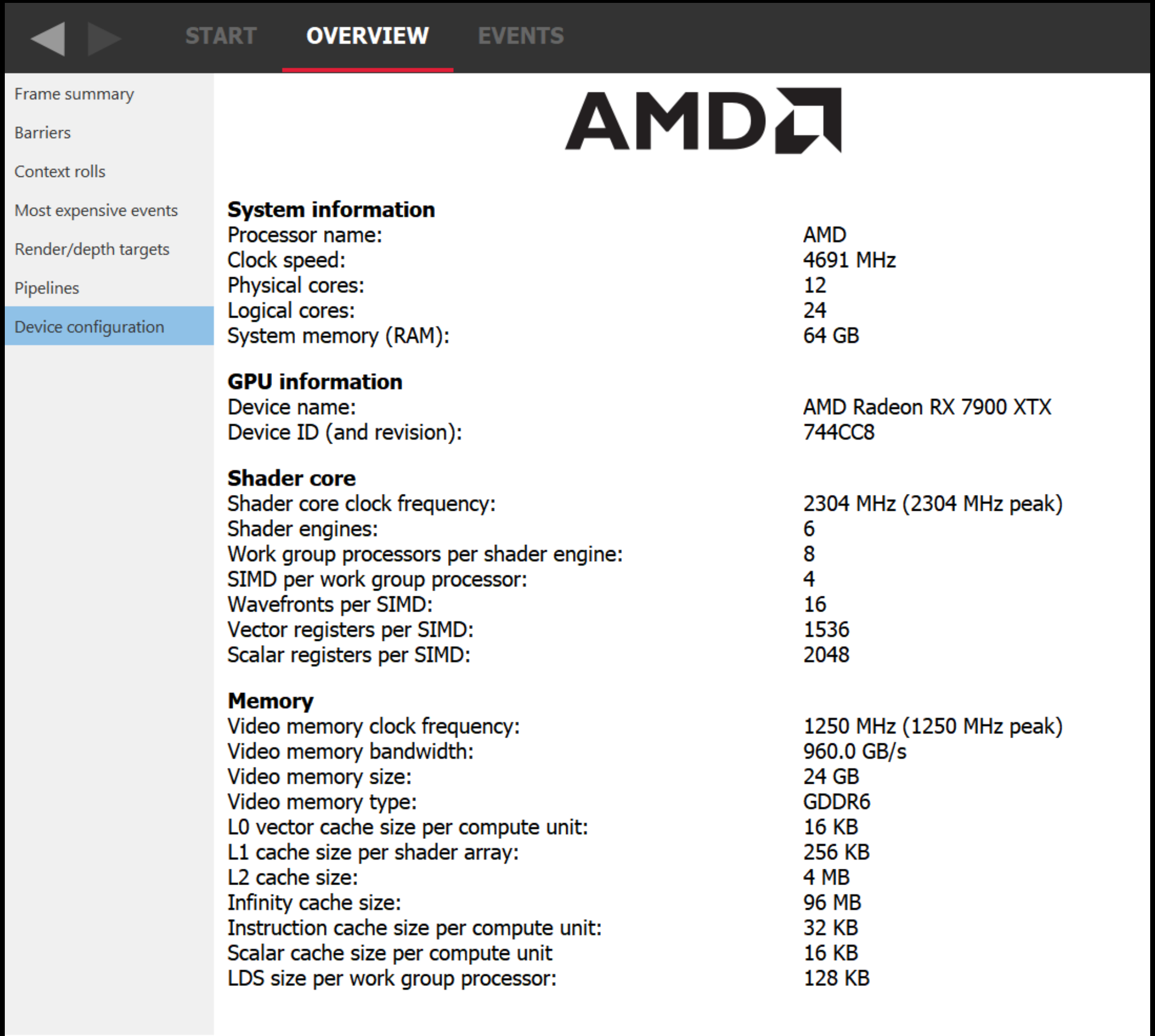
GPU information	
Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core	
Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

Memory	
Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6
WorkGroup Processors (WGP) / SE: 8
Total WGP: 6 * 8 = 48
SIMD per WGP: 4
Total SIMD: 4 * 48 = 192

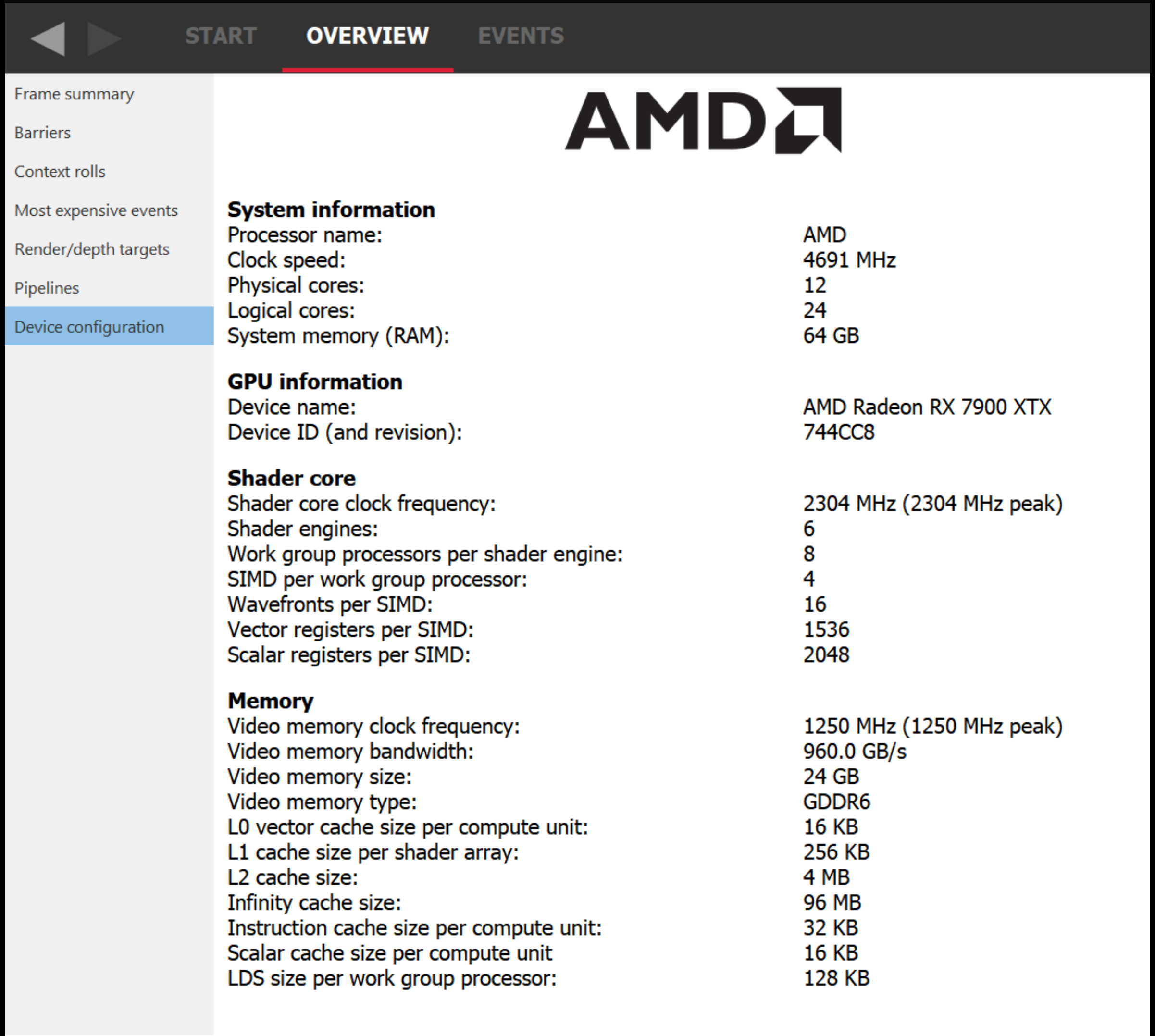
LACK OF WORK LIMITED OCCUPANCY



System information	
Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB
GPU information	
Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8
Shader core	
Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048
Memory	
Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6
WorkGroup Processors (WGP) / SE: 8
Total WGP: 6 * 8 = 48
SIMD per WGP: 4
Total SIMD: 4 * 48 = 192
Wave slots per SIMD: 16

LACK OF WORK LIMITED OCCUPANCY



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

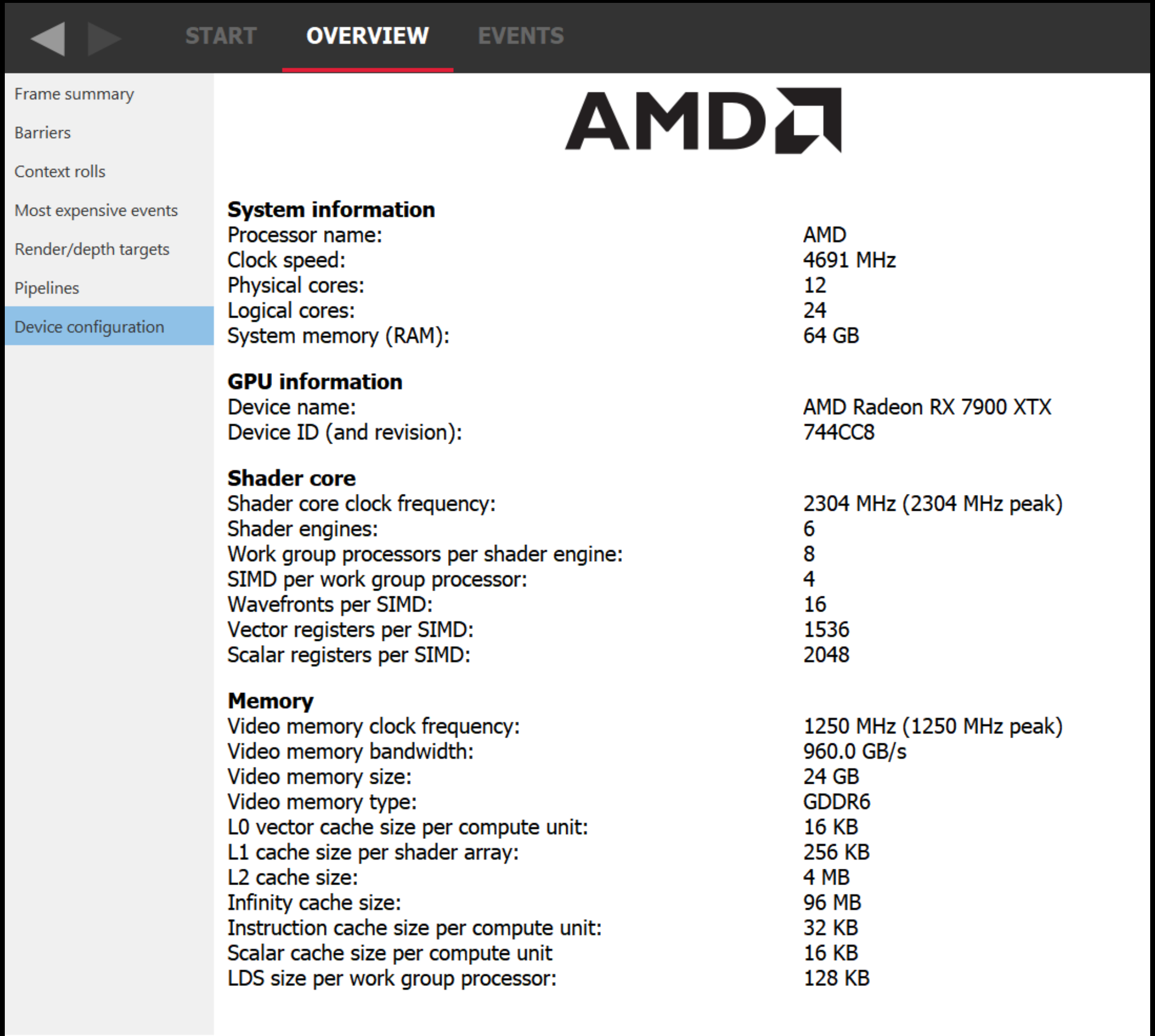
Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6
WorkGroup Processors (WGP) / SE: 8
Total WGP: 6 * 8 = 48
SIMD per WGP: 4
Total SIMD: 4 * 48 = 192
Wave slots per SIMD: 16
Wave slots: 16 * 192 = 3072

LACK OF WORK LIMITED OCCUPANCY



System information

Processor name:	AMD
Clock speed:	4691 MHz
Physical cores:	12
Logical cores:	24
System memory (RAM):	64 GB

GPU information

Device name:	AMD Radeon RX 7900 XTX
Device ID (and revision):	744CC8

Shader core

Shader core clock frequency:	2304 MHz (2304 MHz peak)
Shader engines:	6
Work group processors per shader engine:	8
SIMD per work group processor:	4
Wavefronts per SIMD:	16
Vector registers per SIMD:	1536
Scalar registers per SIMD:	2048

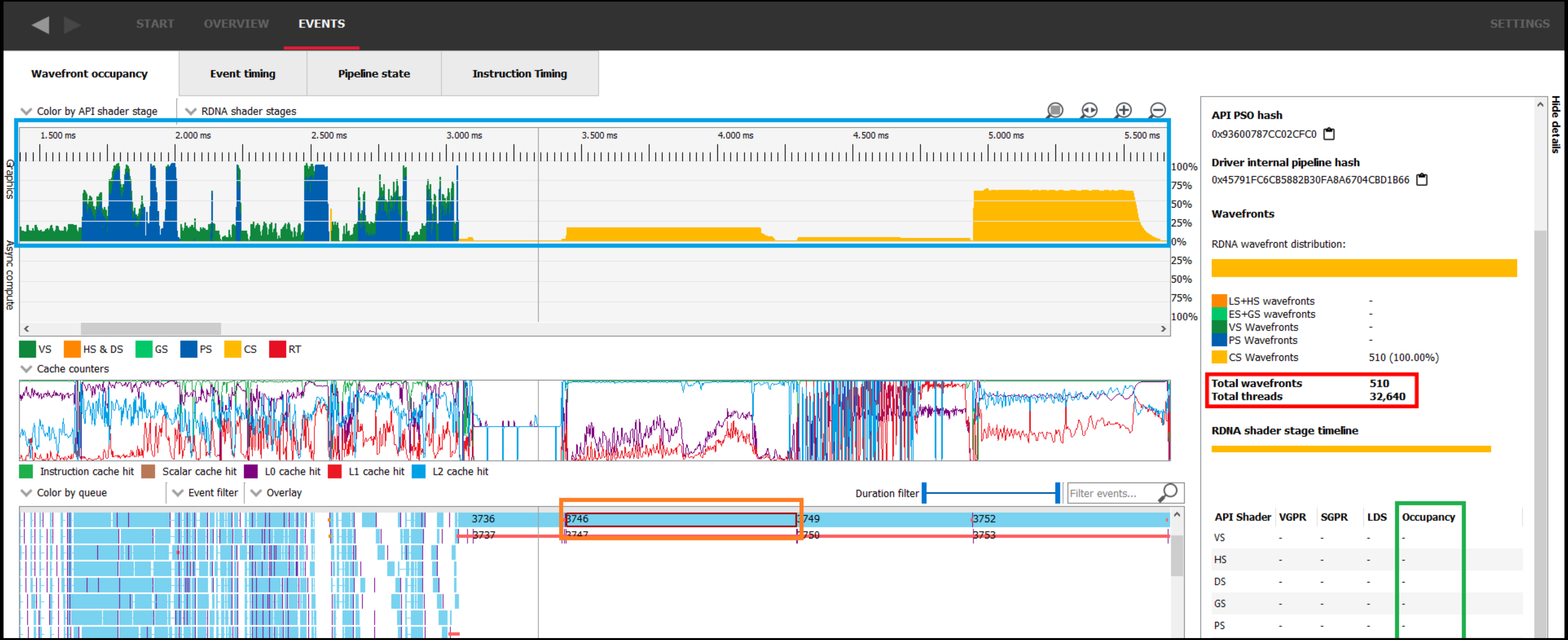
Memory

Video memory clock frequency:	1250 MHz (1250 MHz peak)
Video memory bandwidth:	960.0 GB/s
Video memory size:	24 GB
Video memory type:	GDDR6
L0 vector cache size per compute unit:	16 KB
L1 cache size per shader array:	256 KB
L2 cache size:	4 MB
Infinity cache size:	96 MB
Instruction cache size per compute unit:	32 KB
Scalar cache size per compute unit	16 KB
LDS size per work group processor:	128 KB

Shader Engines (SE): 6
WorkGroup Processors (WGP) / SE: 8
Total WGP: 6 * 8 = 48
SIMD per WGP: 4
Total SIMD: 4 * 48 = 192
Wave slots per SIMD: 16
Wave slots: 16 * 192 = 3072

Max occupancy
510 / 3072 = 16.6%

FILL THE GPU WITH ENOUGH WORK



API PSO hash
0x93600787CC02CFC0

Driver internal pipeline hash
0x45791FC6CB5882B30FA8A6704CBD1B66

Wavefronts

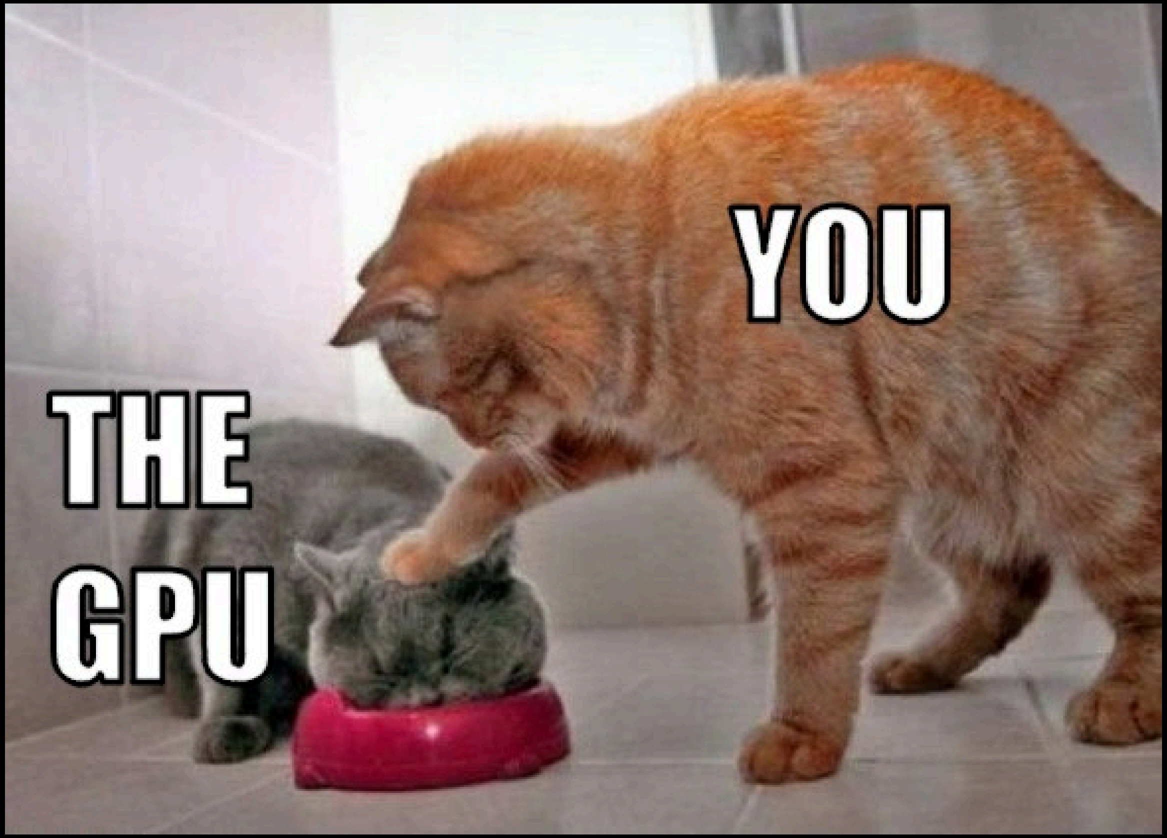
RDNA wavefront distribution:

LS+HS wavefronts	-
ES+GS wavefronts	-
VS Wavefronts	-
PS Wavefronts	-
CS Wavefronts	510 (100.00%)
Total wavefronts	**510**
Total threads	**32,640**

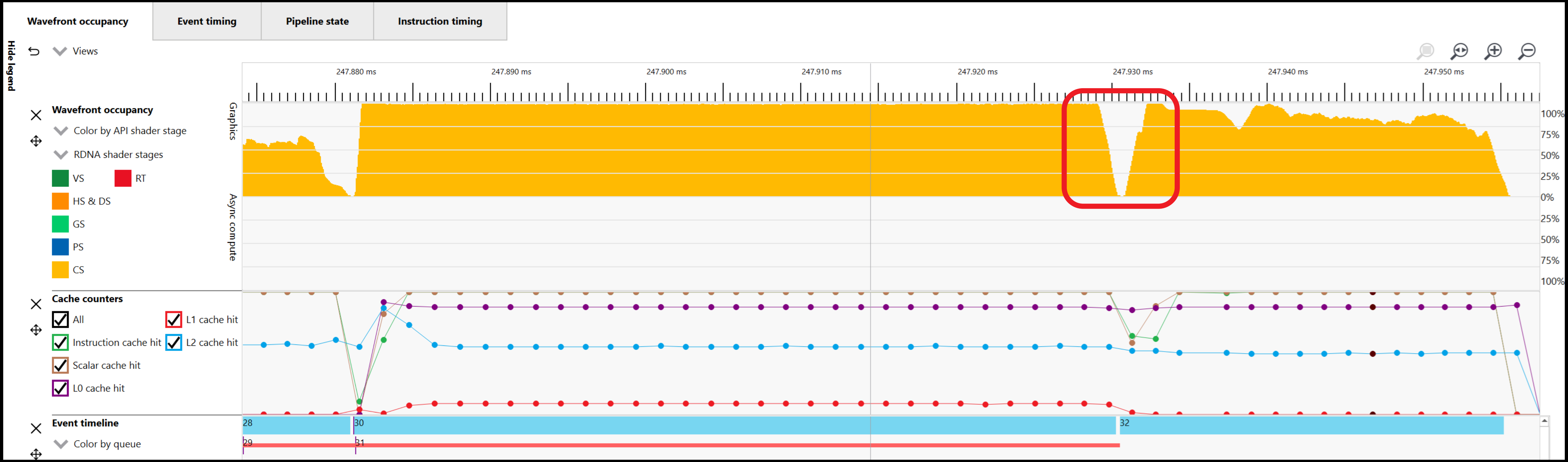
RDNA shader stage timeline

API Shader	VGPR	SGPR	LDS	Occupancy
VS	-	-	-	-
HS	-	-	-	-
DS	-	-	-	-
GS	-	-	-	-
PS	-	-	-	-

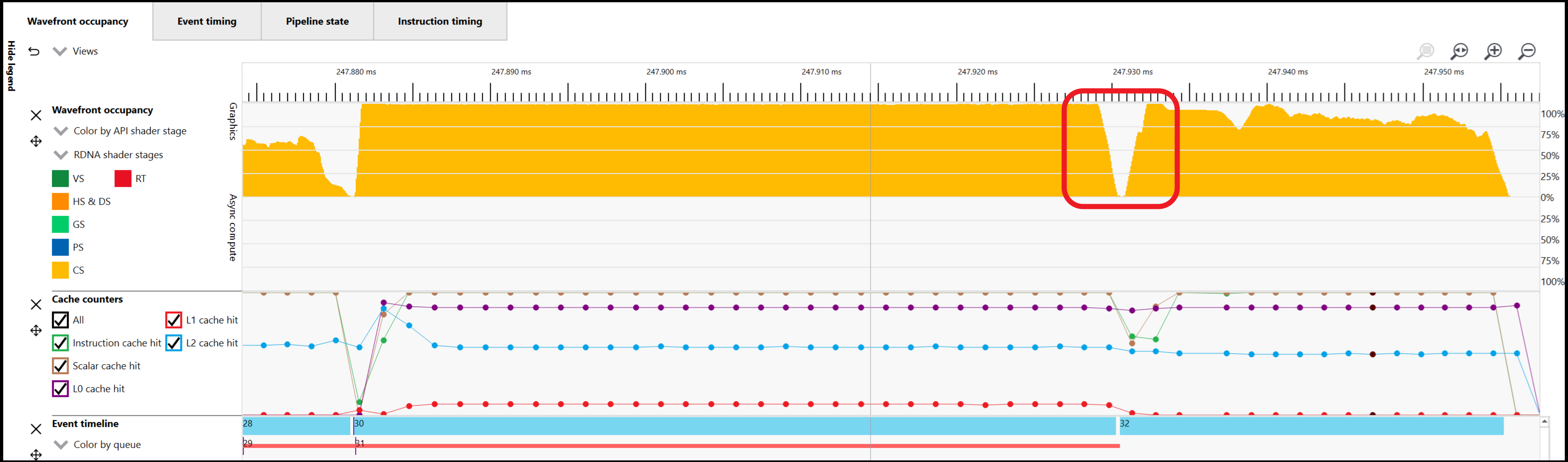
FEED DEM GPUs

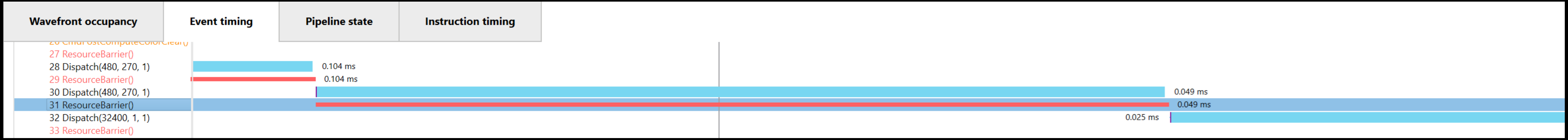


OCCUPANCY GAP



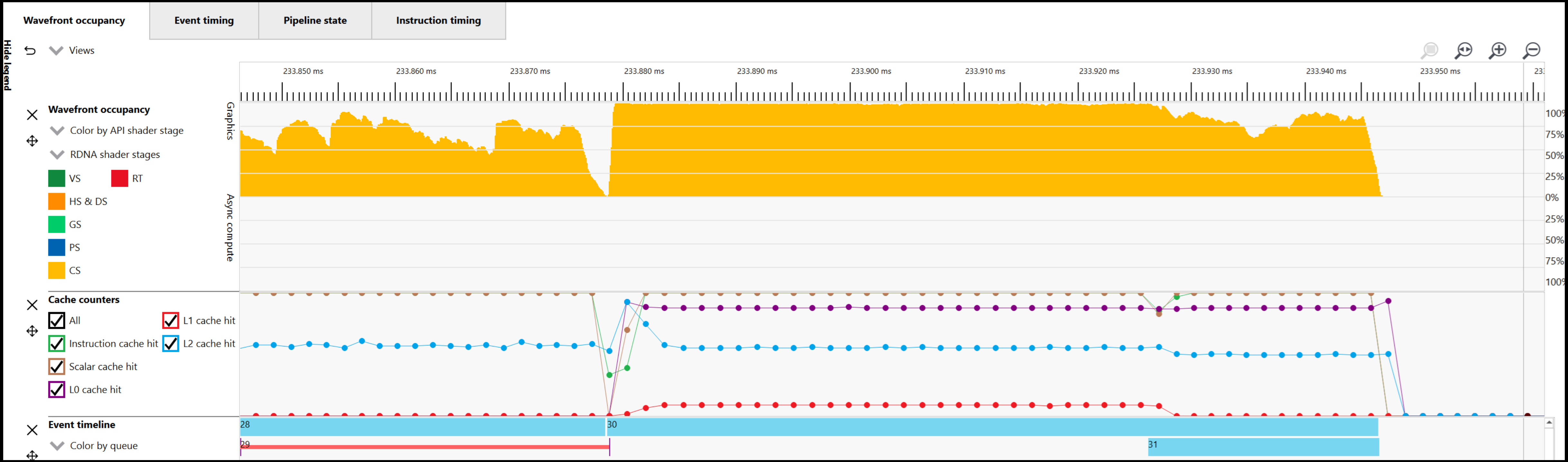
OCCUPANCY GAP





27 ResourceBarrier()
28 Dispatch(480, 270, 1) 0.104 ms
29 ResourceBarrier() 0.104 ms
30 Dispatch(480, 270, 1) 0.049 ms
31 ResourceBarrier() 0.049 ms
32 Dispatch(32400, 1, 1) 0.025 ms
33 ResourceBarrier()

LET YOUR WORKLOADS OVERLAP



Wavefront occupancy

Color by API shader stage

RDNA shader stages

🟩 VS 🟥 RT

🟧 HS & DS

🟩 GS

🟦 PS

🟨 CS

Cache counters

☑ All ☑ L1 cache hit

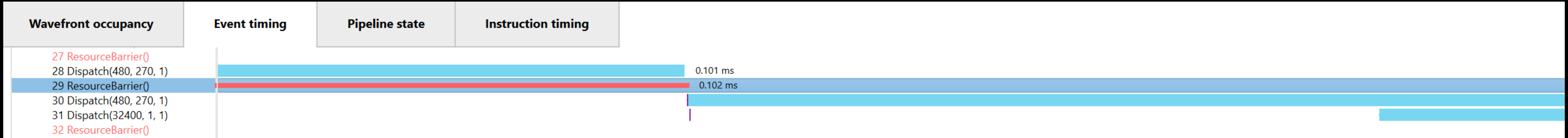
☑ Instruction cache hit ☑ L2 cache hit

☑ Scalar cache hit

☑ L0 cache hit

Event timeline

Color by queue



27 ResourceBarrier()

28 Dispatch(480, 270, 1) 0.101 ms

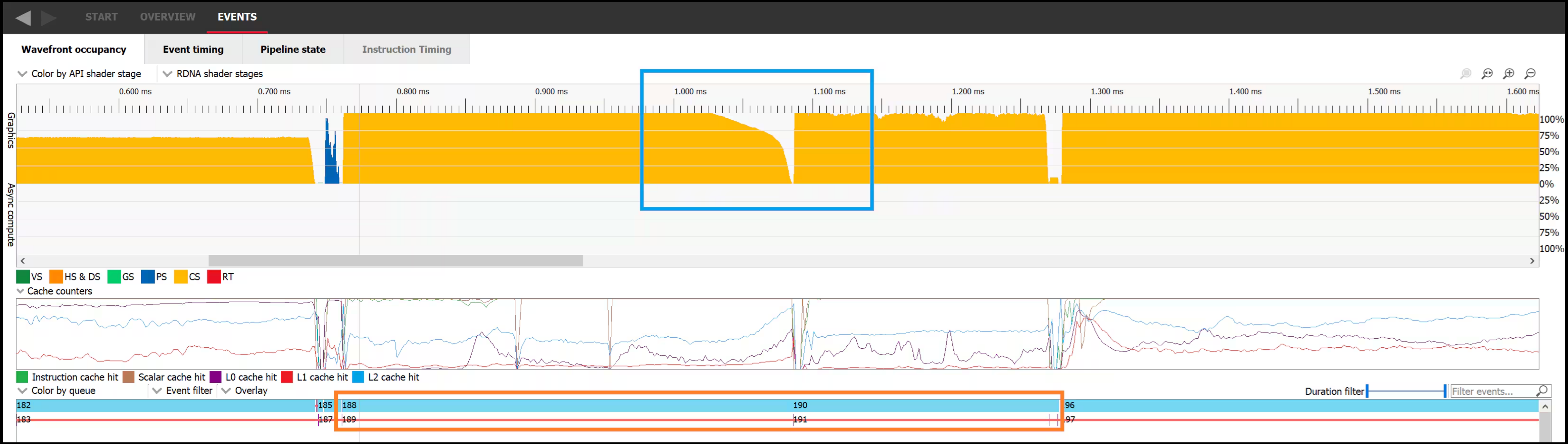
29 ResourceBarrier() 0.102 ms

30 Dispatch(480, 270, 1)

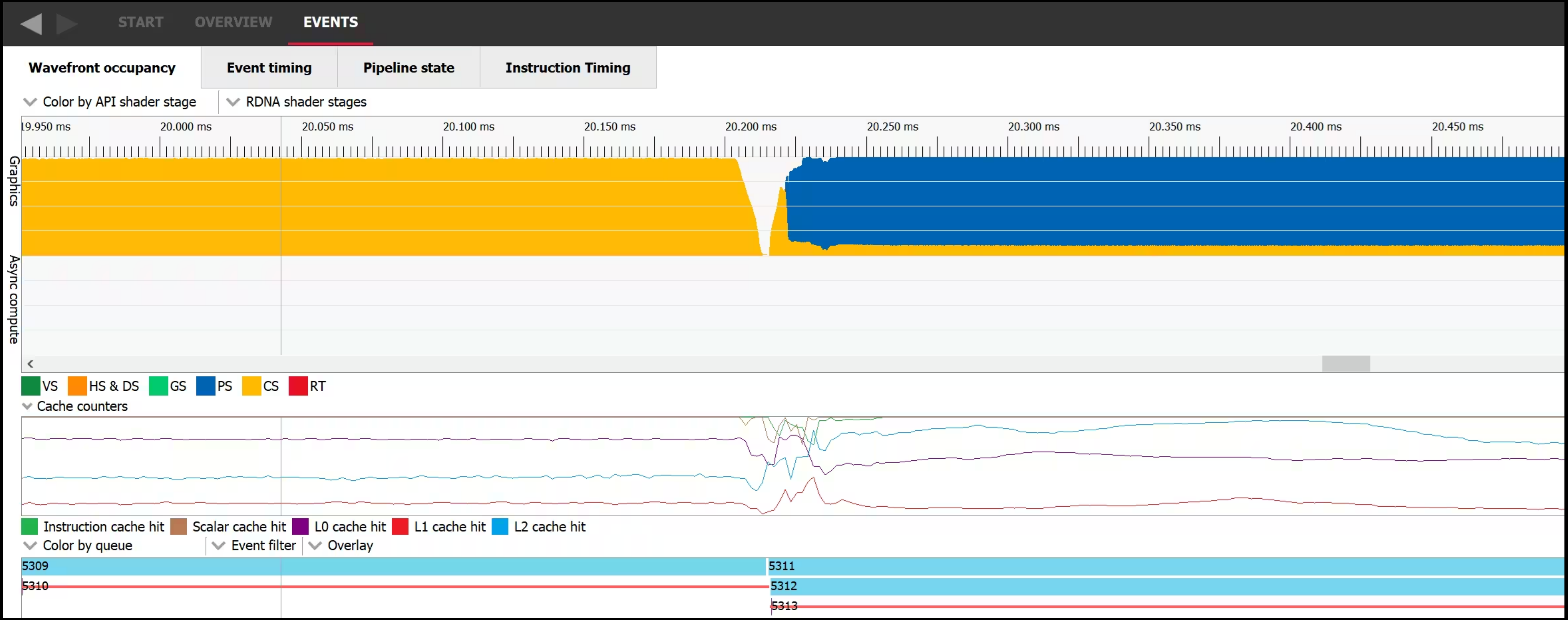
31 Dispatch(32400, 1, 1)

32 ResourceBarrier()

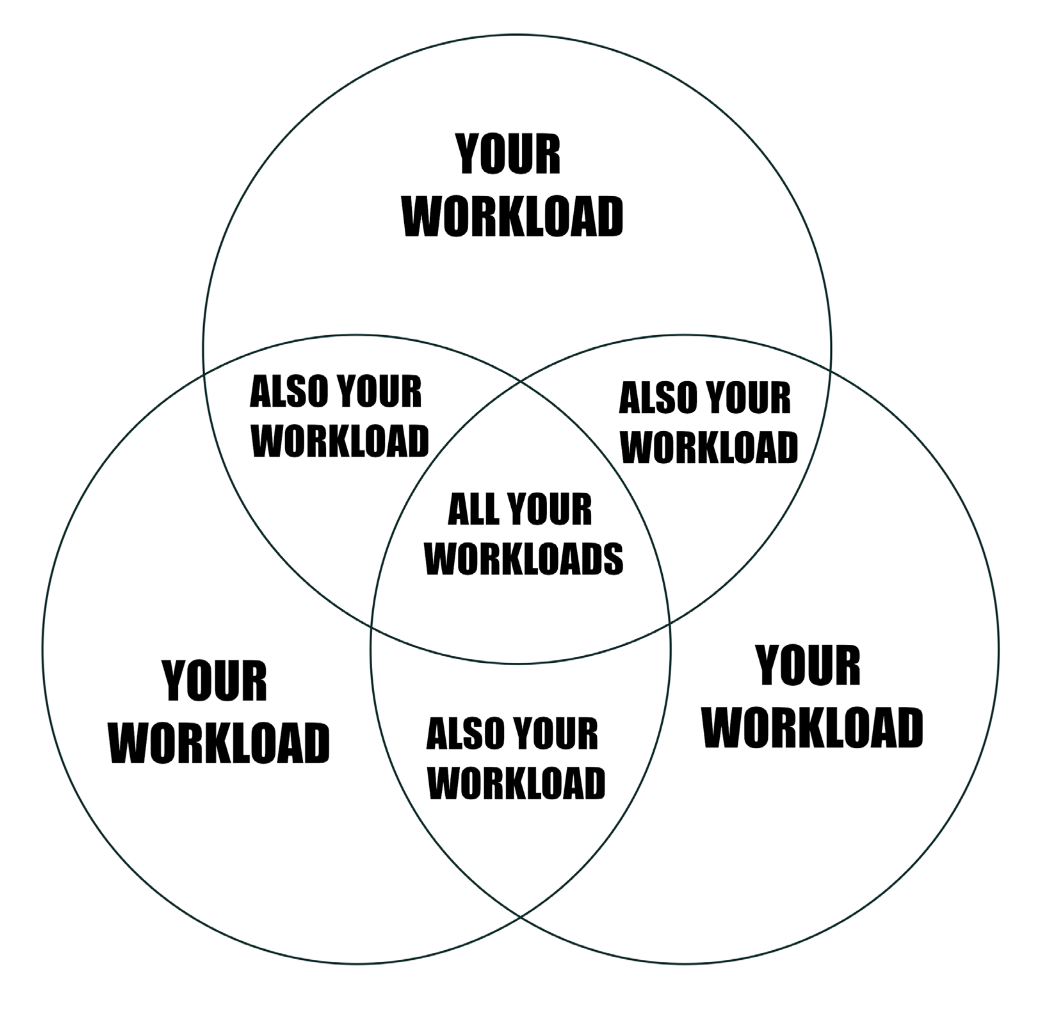
LET YOUR WORKLOADS OVERLAP



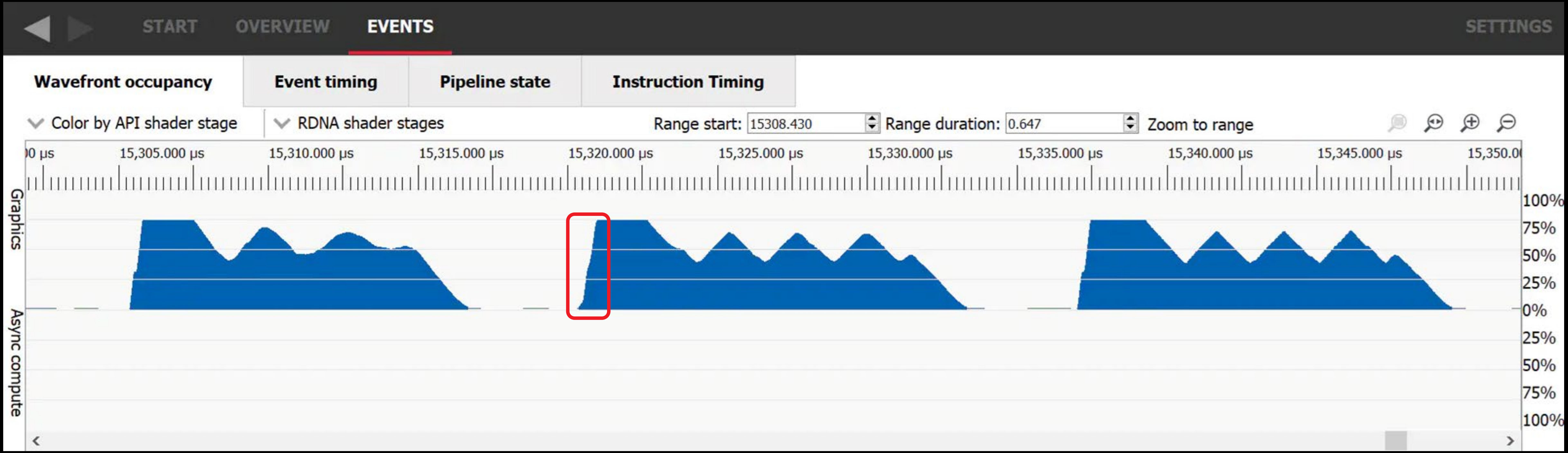
LET YOUR WORKLOADS OVERLAP



LET YOUR WORKLOADS OVERLAP



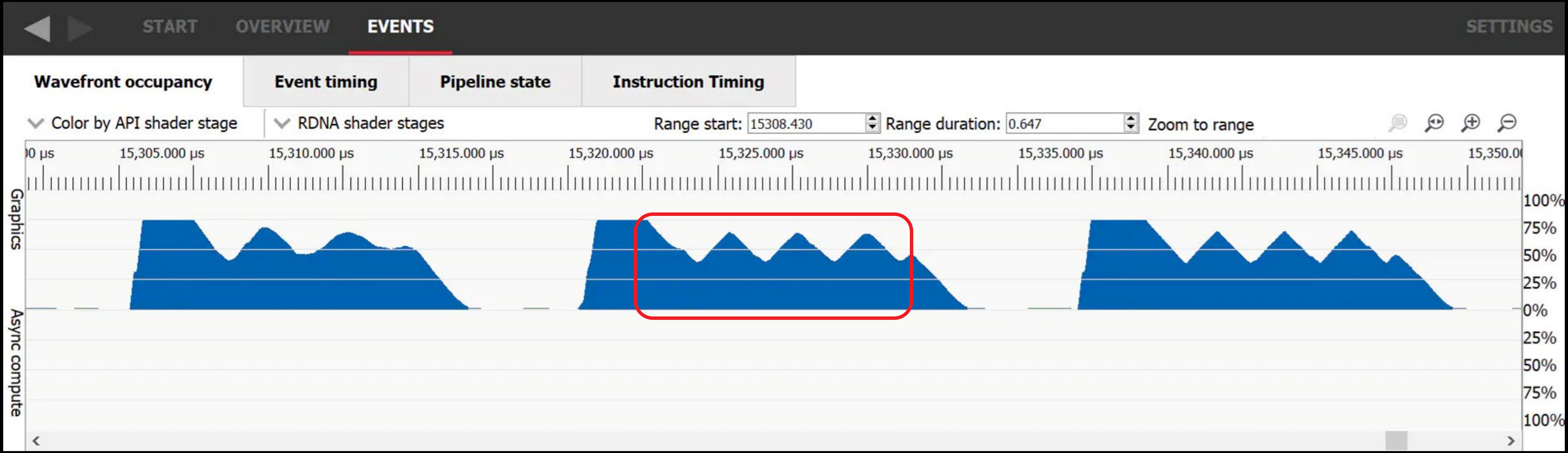
LAUNCH RATE LIMITED WORKLOAD



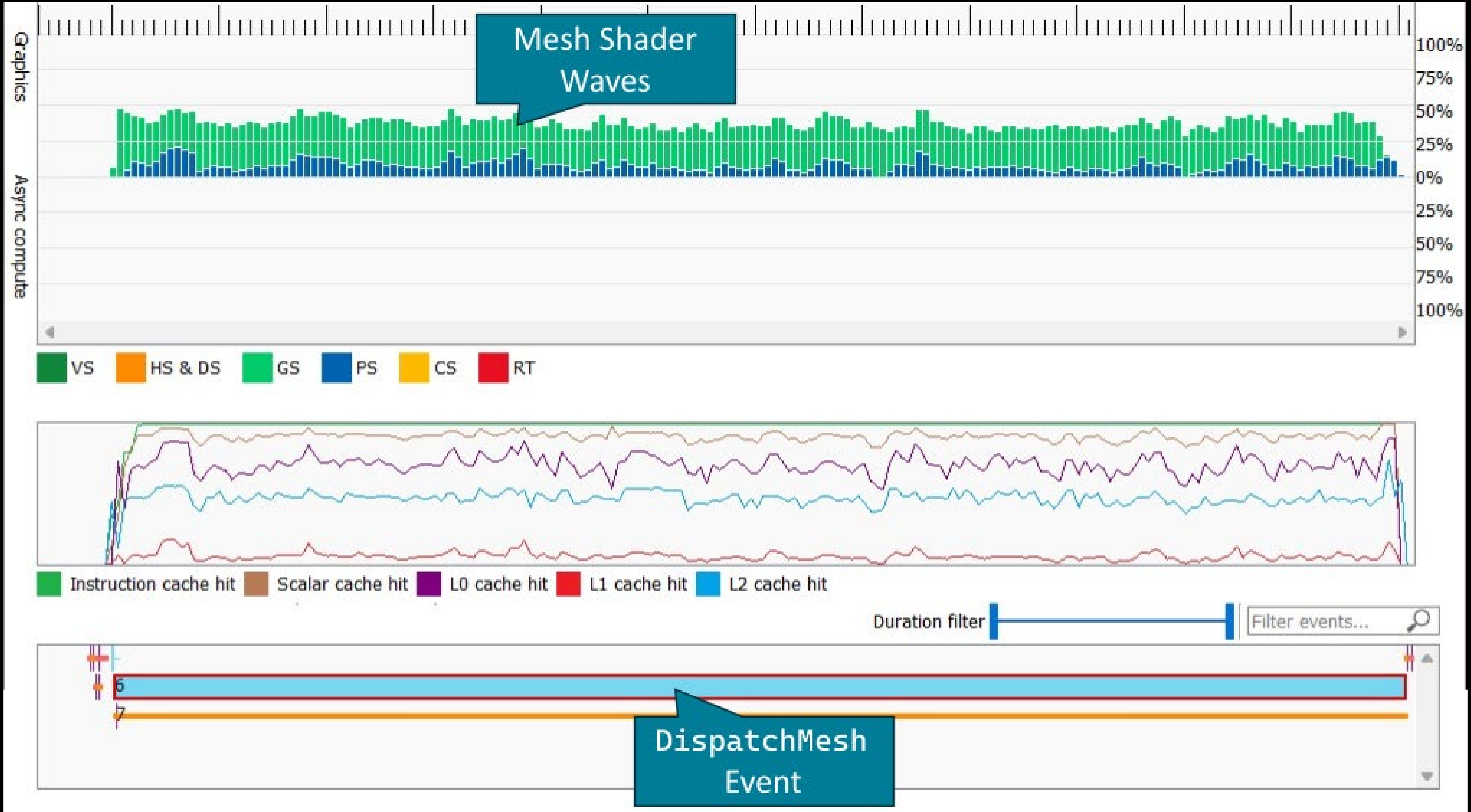
Range start: 15308.430

Range duration: 0.647

LAUNCH RATE LIMITED WORKLOAD



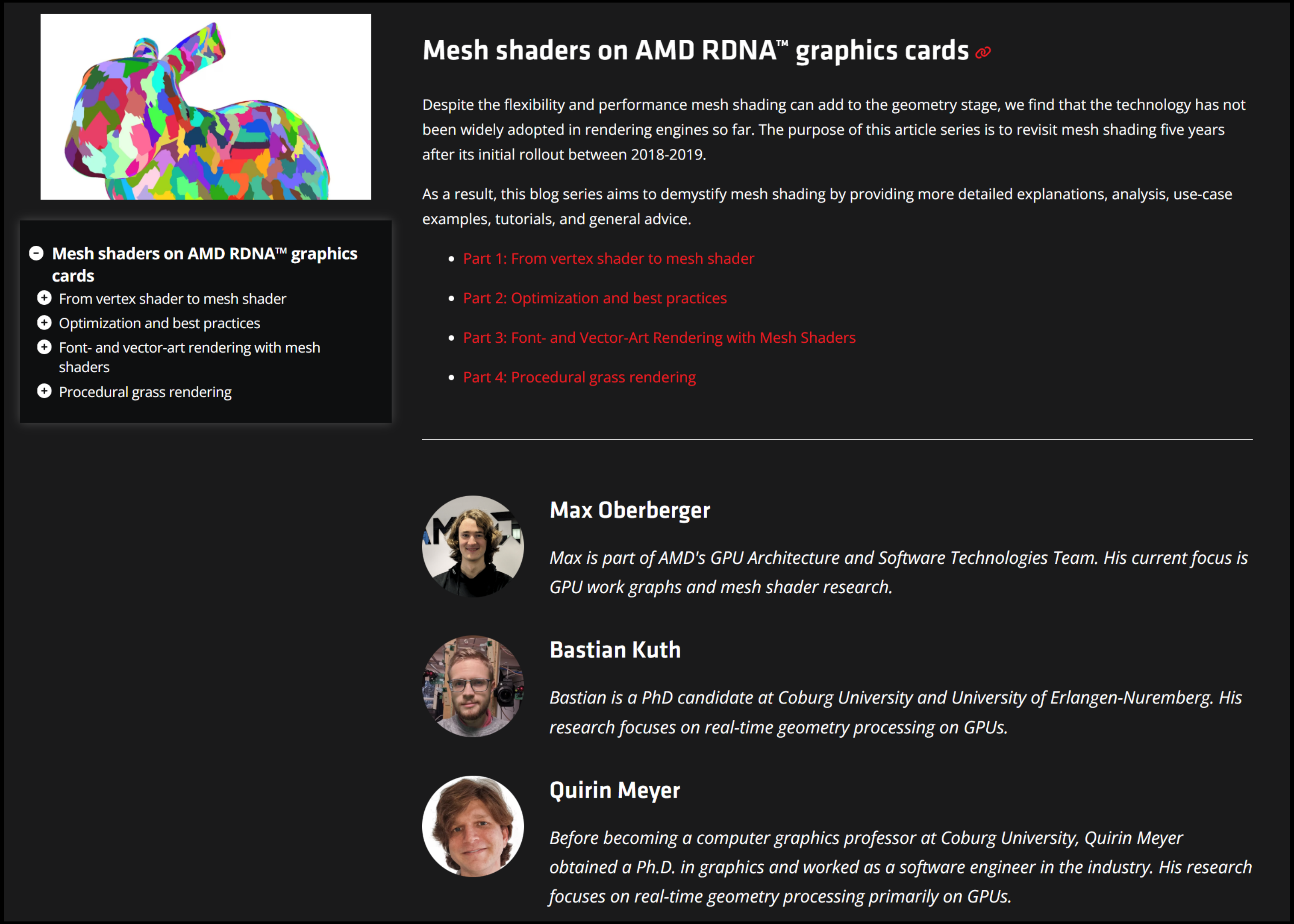
GEOMETRY WORKLOADS



Mesh Shader Waves

DispatchMesh Event

GEOMETRY WORKLOADS



- Mesh shaders on AMD RDNA™ graphics cards
  - From vertex shader to mesh shader
  - Optimization and best practices
  - Font- and vector-art rendering with mesh shaders
  - Procedural grass rendering

Mesh shaders on AMD RDNA™ graphics cards

Despite the flexibility and performance mesh shading can add to the geometry stage, we find that the technology has not been widely adopted in rendering engines so far. The purpose of this article series is to revisit mesh shading five years after its initial rollout between 2018-2019.

As a result, this blog series aims to demystify mesh shading by providing more detailed explanations, analysis, use-case examples, tutorials, and general advice.

- Part 1: From vertex shader to mesh shader
- Part 2: Optimization and best practices
- Part 3: Font- and Vector-Art Rendering with Mesh Shaders
- Part 4: Procedural grass rendering

Max Oberberger

Max is part of AMD's GPU Architecture and Software Technologies Team. His current focus is GPU work graphs and mesh shader research.

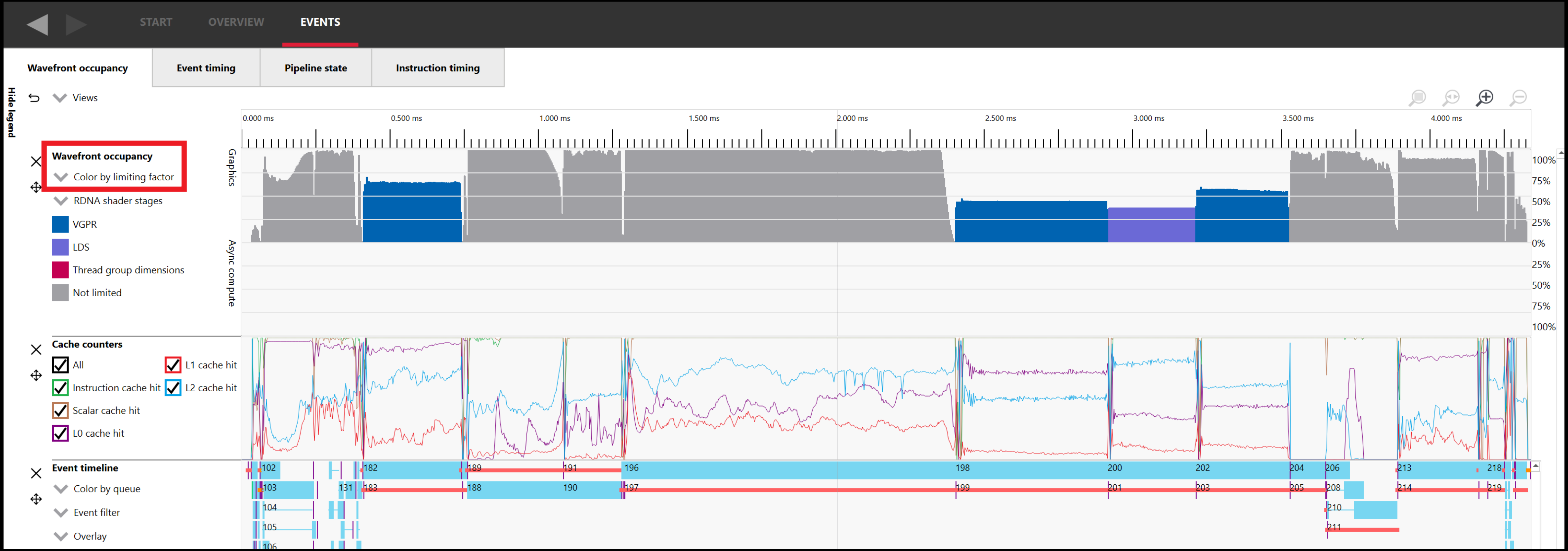
Bastian Kuth

Bastian is a PhD candidate at Coburg University and University of Erlangen-Nuremberg. His research focuses on real-time geometry processing on GPUs.

Quirin Meyer

Before becoming a computer graphics professor at Coburg University, Quirin Meyer obtained a Ph.D. in graphics and worked as a software engineer in the industry. His research focuses on real-time geometry processing primarily on GPUs.

OCCUPANCY LIMITERS

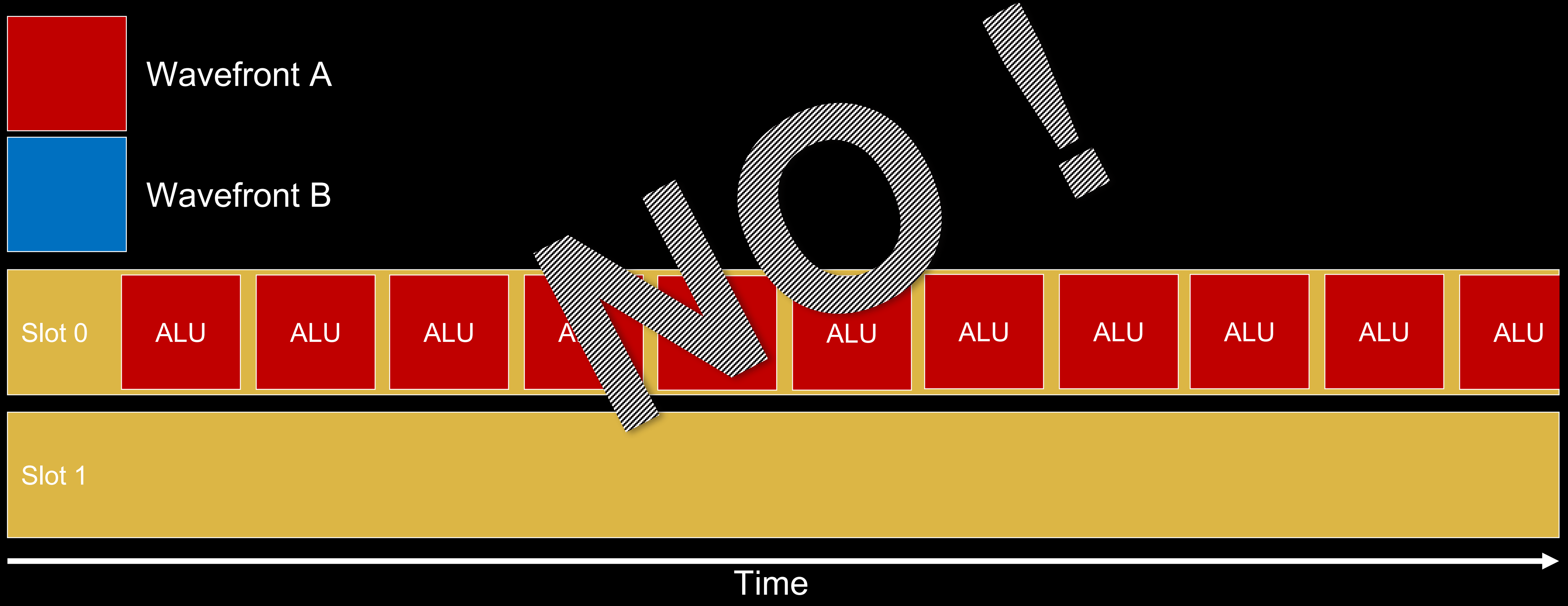


Wavefront occupancy
Color by limiting factor
RDNA shader stages
🟦 VGPR
🟪 LDS
🟥 Thread group dimensions
⬜ Not limited

Cache counters
☑ All
☑ L1 cache hit
☑ Instruction cache hit
☑ L2 cache hit
☑ Scalar cache hit
☑ L0 cache hit

Q: Does better occupancy necessarily mean better performance?

Q: Does better occupancy necessarily mean better performance?



🟥 Wavefront A

🟦 Wavefront B

Slot 0: ALU ALU ALU A… ALU ALU ALU ALU ALU ALU

Slot 1

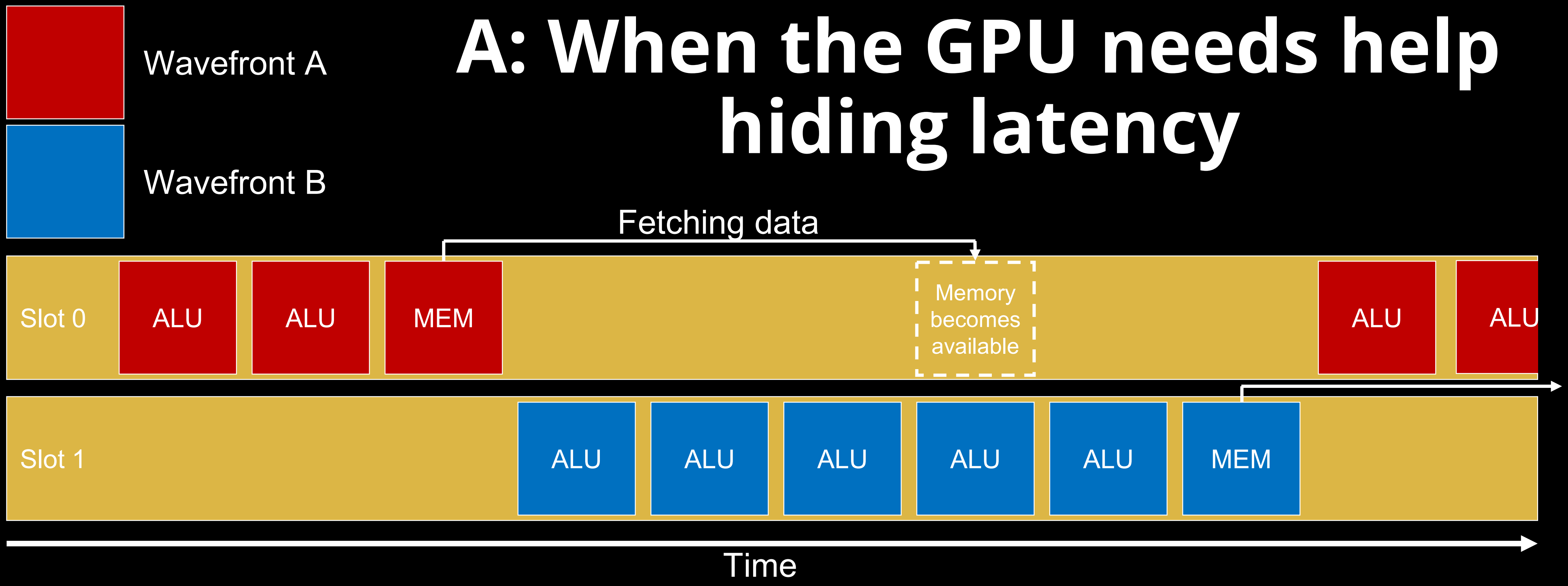
Time

NO !

Q: When should I care about occupancy?

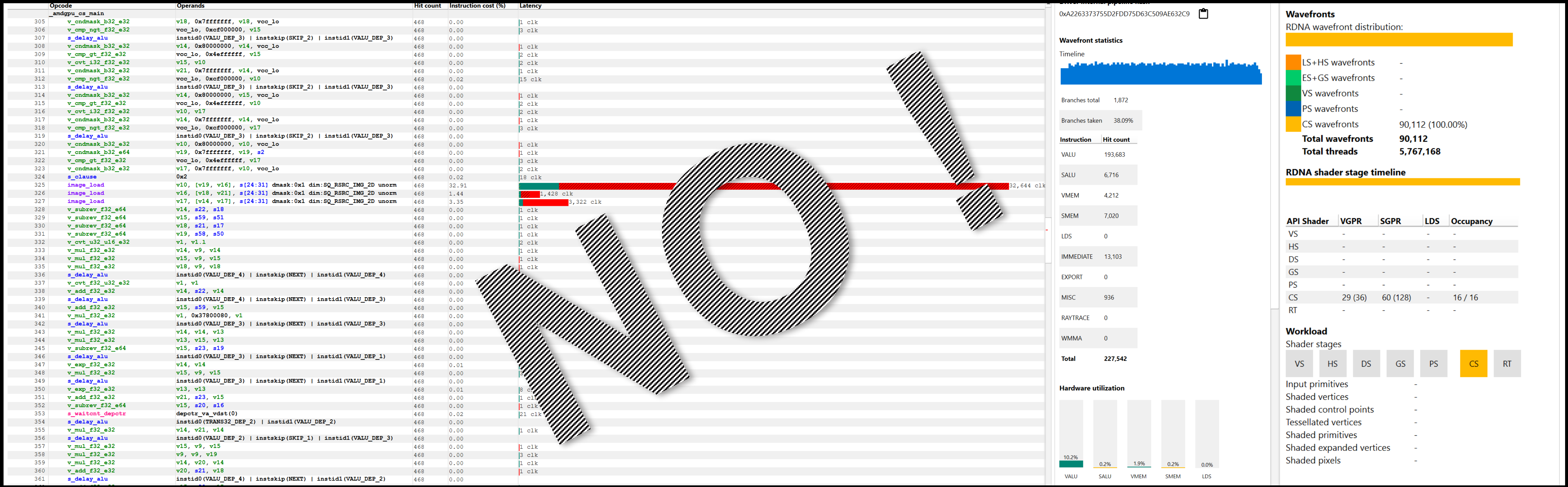
Q: When should I care about occupancy?

A: When the GPU needs help hiding latency



Q: Does maximum occupancy mean that all the memory access latency from my shader is hidden?

Q: Does maximum occupancy mean that all the memory access latency from my shader is hidden?



	Opcode	Operands	Hit count	Instruction cost (%)	Latency
	_amdgpu_cs_main				
305	v_cndmask_b32_e32	v18, 0x7fffffff, v18, vcc_lo	468	0.00	1 clk
306	v_cmp_ngt_f32_e32	vcc_lo, 0xcf000000, v15	468	0.00	3 clk
307	s_delay_alu	instid0(VALU_DEP_3) | instskip(SKIP_2) | instid1(VALU_DEP_3)	468	0.00	
308	v_cndmask_b32_e32	v14, 0x80000000, v14, vcc_lo	468	0.00	1 clk
309	v_cmp_gt_f32_e32	vcc_lo, 0x4effffff, v15	468	0.00	2 clk
310	v_cvt_i32_f32_e32	v15, v10	468	0.00	2 clk
311	v_cndmask_b32_e32	v21, 0x7fffffff, v14, vcc_lo	468	0.00	1 clk
312	v_cmp_ngt_f32_e32	vcc_lo, 0xcf000000, v10	468	0.02	15 clk
313	s_delay_alu	instid0(VALU_DEP_3) | instskip(SKIP_2) | instid1(VALU_DEP_3)	468	0.00	
314	v_cndmask_b32_e32	v14, 0x80000000, v15, vcc_lo	468	0.00	1 clk
315	v_cmp_gt_f32_e32	vcc_lo, 0x4effffff, v10	468	0.00	2 clk
316	v_cvt_i32_f32_e32	v10, v17	468	0.00	2 clk
317	v_cndmask_b32_e32	v14, 0x7fffffff, v14, vcc_lo	468	0.00	1 clk
318	v_cmp_ngt_f32_e32	vcc_lo, 0xcf000000, v17	468	0.00	3 clk
319	s_delay_alu	instid0(VALU_DEP_3) | instskip(SKIP_2) | instid1(VALU_DEP_3)	468	0.00	
320	v_cndmask_b32_e32	v10, 0x80000000, v10, vcc_lo	468	0.00	1 clk
321	v_cndmask_b32_e64	v19, 0x7fffffff, v19, s2	468	0.00	1 clk
322	v_cmp_gt_f32_e32	vcc_lo, 0x4effffff, v17	468	0.00	3 clk
323	v_cndmask_b32_e32	v17, 0x7fffffff, v10, vcc_lo	468	0.00	2 clk
324	s_clause	0x2	468	0.02	18 clk
325	image_load	v10, [v19, v16], s[24:31] dmask:0x1 dim:SQ_RSRC_IMG_2D unorm	468	32.91	32,644 clk
326	image_load	v16, [v18, v21], s[24:31] dmask:0x1 dim:SQ_RSRC_IMG_2D unorm	468	1.44	1,428 clk
327	image_load	v17, [v14, v17], s[24:31] dmask:0x1 dim:SQ_RSRC_IMG_2D unorm	468	3.35	3,322 clk
328	v_subrev_f32_e64	v14, s22, s18	468	0.00	1 clk
329	v_subrev_f32_e64	v15, s59, s51	468	0.00	1 clk
330	v_subrev_f32_e64	v18, s21, s17	468	0.00	1 clk
331	v_subrev_f32_e64	v19, s58, s50	468	0.00	1 clk
332	v_cvt_u32_u16_e32	v1, v1.1	468	0.00	2 clk
333	v_mul_f32_e32	v14, v9, v14	468	0.00	1 clk
334	v_mul_f32_e32	v15, v9, v15	468	0.00	1 clk
335	v_mul_f32_e32	v18, v9, v18	468	0.00	1 clk
336	s_delay_alu	instid0(VALU_DEP_4) | instskip(NEXT) | instid1(VALU_DEP_4)	468	0.00	
337	v_cvt_f32_u32_e32	v1, v1	468	0.00	[illegible]
338	v_add_f32_e32	v14, s22, v14	468	0.00	[illegible]
339	s_delay_alu	instid0(VALU_DEP_4) | instskip(NEXT) | instid1(VALU_DEP_3)	468	0.00	
340	v_add_f32_e32	v15, s59, v15	468	0.00	[illegible]
341	v_mul_f32_e32	v1, 0x37800080, v1	468	0.00	[illegible]
342	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_3)	468	0.00	
343	v_mul_f32_e32	v14, v14, v13	468	0.00	[illegible]
344	v_mul_f32_e32	v13, v15, v13	468	0.00	[illegible]
345	v_subrev_f32_e64	v15, s23, s19	468	0.00	[illegible]
346	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_1)	468	0.00	
347	v_exp_f32_e32	v14, v14	468	0.01	[illegible]
348	v_mul_f32_e32	v15, v9, v15	468	0.00	[illegible]
349	s_delay_alu	instid0(VALU_DEP_3) | instskip(NEXT) | instid1(VALU_DEP_1)	468	0.00	
350	v_exp_f32_e32	v13, v13	468	0.01	8 clk
351	v_add_f32_e32	v21, s23, v15	468	0.00	1 clk
352	v_subrev_f32_e64	v15, s20, s16	468	0.00	1 clk
353	s_waitcnt_depctr	depctr_va_vdst(0)	468	0.02	21 clk
354	s_delay_alu	instid0(TRANS32_DEP_2) | instid1(VALU_DEP_2)	468	0.00	
355	v_mul_f32_e32	v14, v21, v14	468	0.00	1 clk
356	s_delay_alu	instid0(VALU_DEP_2) | instskip(SKIP_1) | instid1(VALU_DEP_3)	468	0.00	
357	v_mul_f32_e32	v15, v9, v15	468	0.00	1 clk
358	v_mul_f32_e32	v9, v9, v19	468	0.00	3 clk
359	v_mul_f32_e32	v14, v20, v14	468	0.00	1 clk
360	v_add_f32_e32	v20, s21, v18	468	0.00	1 clk
361	s_delay_alu	instid0(VALU_DEP_4) | instskip(NEXT) | instid1(VALU_DEP_2)	468	0.00	

0xA2263373755D2FDD75D63C509AE632C9

Wavefront statistics

Timeline

Branches total	1,872
Branches taken	38.09%

Instruction	Hit count
VALU	193,683
SALU	6,716
VMEM	4,212
SMEM	7,020
LDS	0
IMMEDIATE	13,103
EXPORT	0
MISC	936
RAYTRACE	0
WMMA	0
Total	**227,542**

Hardware utilization

Wavefronts

RDNA wavefront distribution:

LS+HS wavefronts	-
ES+GS wavefronts	-
VS wavefronts	-
PS wavefronts	-
CS wavefronts	90,112 (100.00%)
Total wavefronts	**90,112**
Total threads	**5,767,168**

RDNA shader stage timeline

API Shader	VGPR	SGPR	LDS	Occupancy
VS	-	-	-	-
HS	-	-	-	-
DS	-	-	-	-
GS	-	-	-	-
PS	-	-	-	-
CS	29 (36)	60 (128)	-	16 / 16
RT	-	-	-	-

Workload

Shader stages

VS HS DS GS PS CS RT

Input primitives	-
Shaded vertices	-
Shaded control points	-
Tessellated vertices	-
Shaded primitives	-
Shaded expanded vertices	-
Shaded pixels	-

Q: Is lower theoretical occupancy always bad for performance?

Q: Is lower theoretical occupancy always bad for performance?

NO !

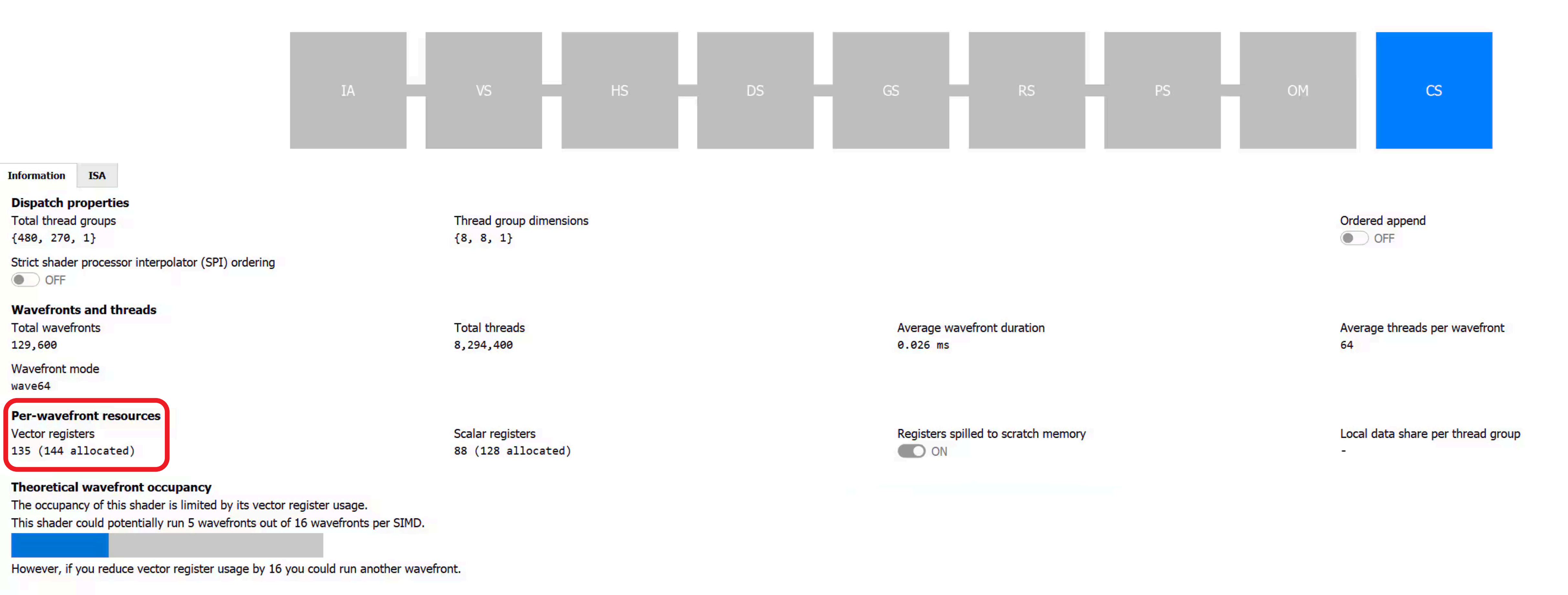
Q: Is lower theoretical occupancy always bad for performance?

NO !

BUT ALSO ,

YES?

REGISTER SPILLING



Dispatch properties

Total thread groups
`{480, 270, 1}`

Thread group dimensions
`{8, 8, 1}`

Ordered append
OFF

Strict shader processor interpolator (SPI) ordering
OFF

Wavefronts and threads

Total wavefronts
`129,600`

Total threads
`8,294,400`

Average wavefront duration
`0.026 ms`

Average threads per wavefront
`64`

Wavefront mode
`wave64`

Per-wavefront resources

Vector registers
`135 (144 allocated)`

Scalar registers
`88 (128 allocated)`

Registers spilled to scratch memory
ON

Local data share per thread group
-

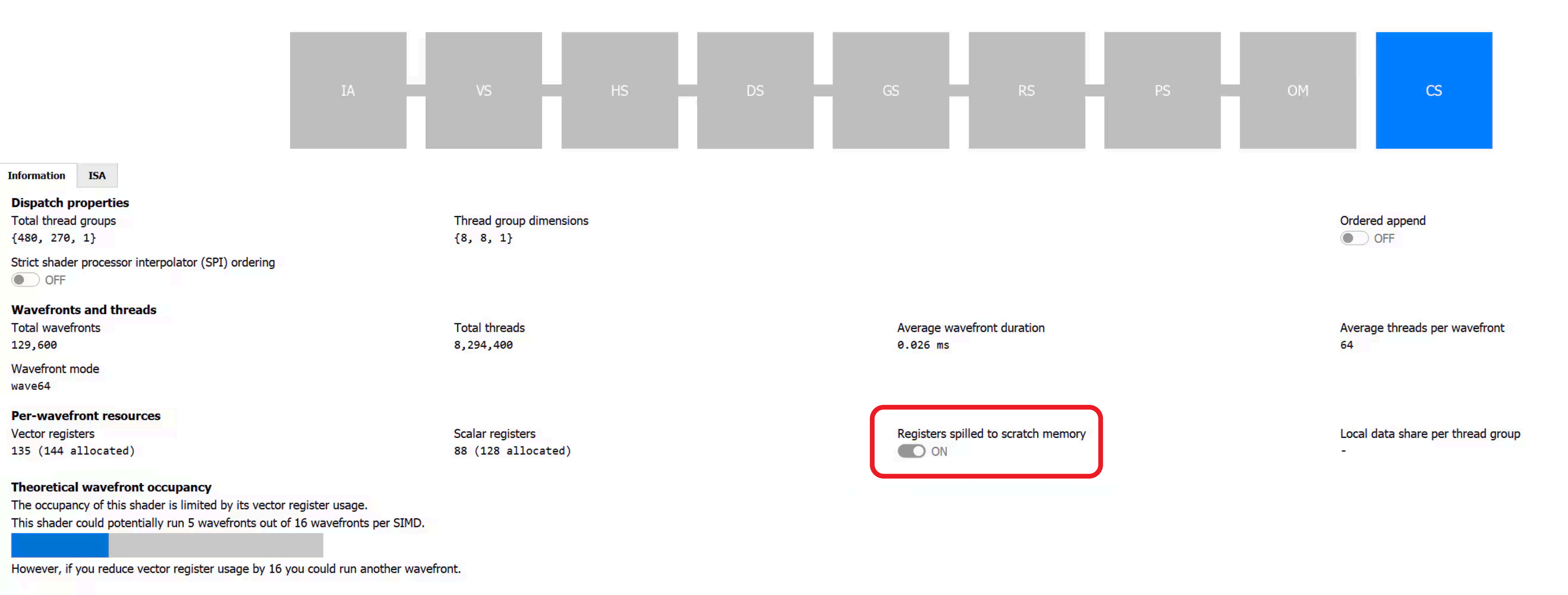
Theoretical wavefront occupancy

The occupancy of this shader is limited by its vector register usage.

This shader could potentially run 5 wavefronts out of 16 wavefronts per SIMD.

However, if you reduce vector register usage by 16 you could run another wavefront.

REGISTER SPILLING



Dispatch properties

Total thread groups
`{480, 270, 1}`

Thread group dimensions
`{8, 8, 1}`

Ordered append
OFF

Strict shader processor interpolator (SPI) ordering
OFF

Wavefronts and threads

Total wavefronts
`129,600`

Total threads
`8,294,400`

Average wavefront duration
`0.026 ms`

Average threads per wavefront
`64`

Wavefront mode
`wave64`

Per-wavefront resources

Vector registers
`135 (144 allocated)`

Scalar registers
`88 (128 allocated)`

Registers spilled to scratch memory
ON

Local data share per thread group
-

Theoretical wavefront occupancy

The occupancy of this shader is limited by its vector register usage.
This shader could potentially run 5 wavefronts out of 16 wavefronts per SIMD.

However, if you reduce vector register usage by 16 you could run another wavefront.

COPYRIGHT AND DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THANK YOU !

✉️ - francois.guthmann@amd.com

🦋 / X - @frguthmann

AMD
GPUOpen