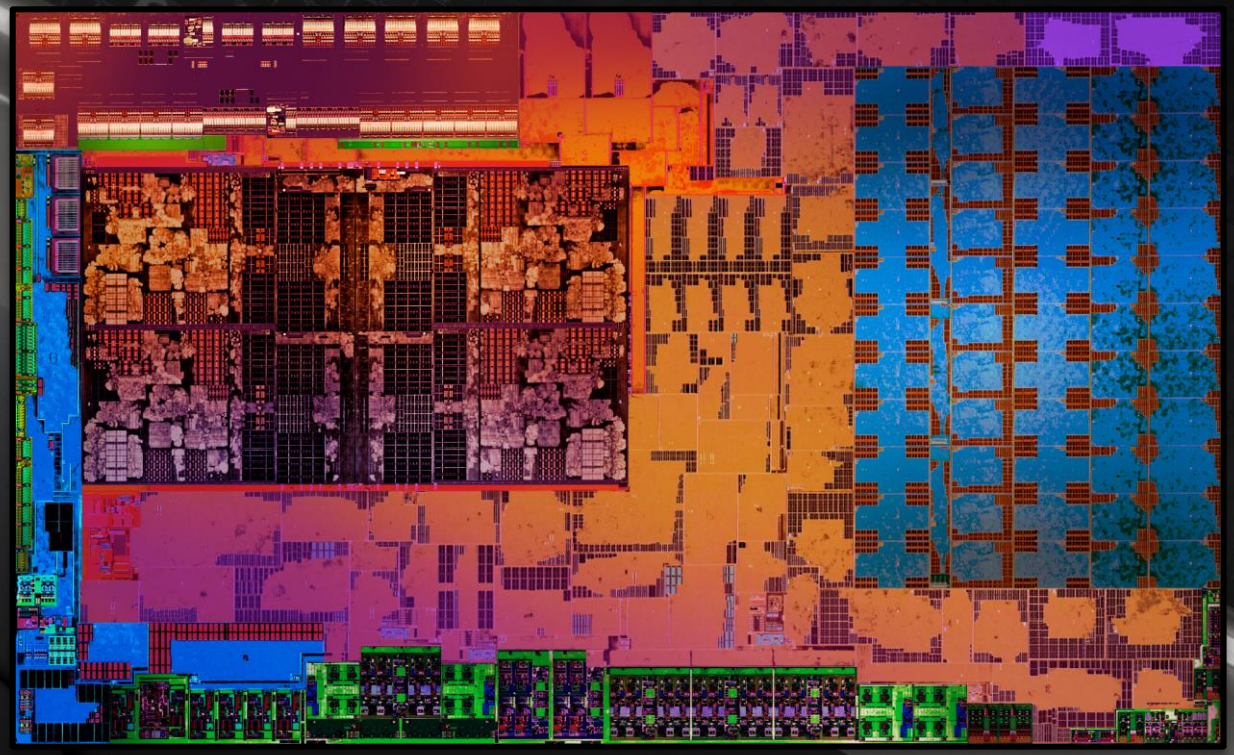
AMD

ENGINE OPTIMIZATION HOT LAP

TIMOTHY LOTTES | GDC 2018



GOALS OF THIS TALK

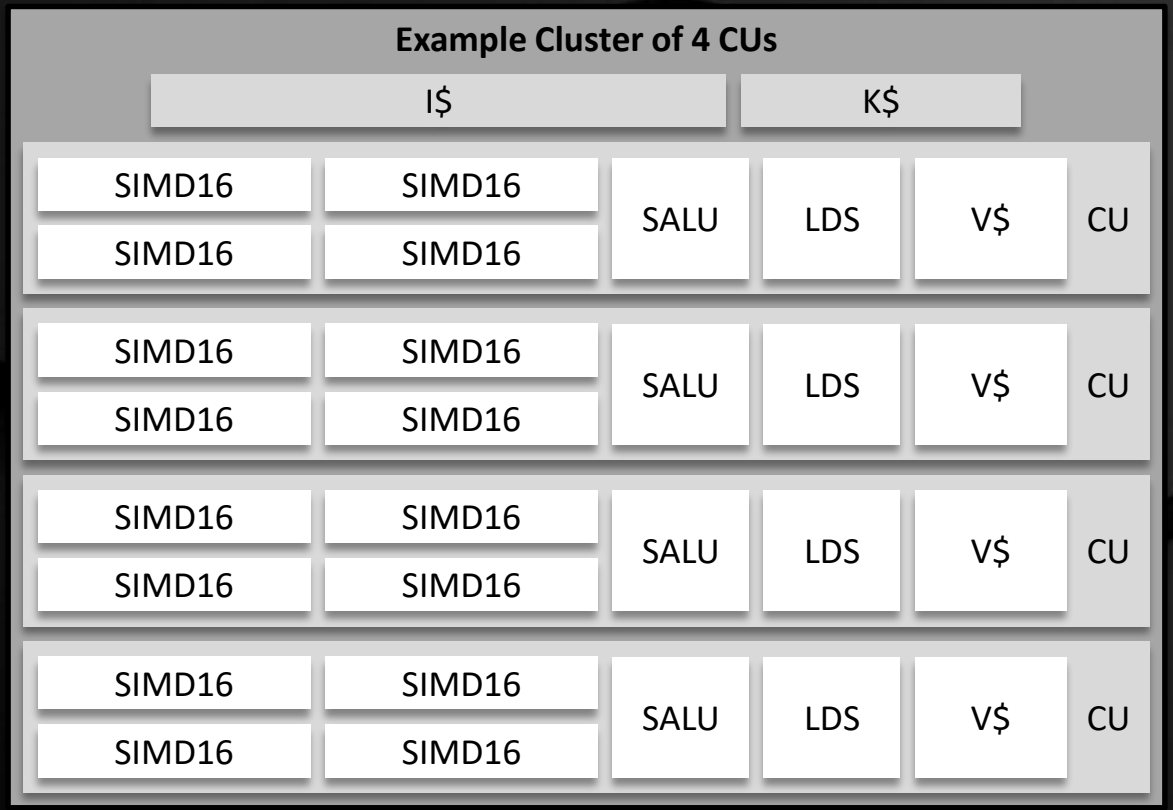
AMD

- Teach you more about how the GPU works closer to the hardware level
- To help you reason about your own GPU workloads and engine design
- Present results of various optimizations
- **And ultimately help you better realize your visual and performance goals as GPUs continue to scale**

GCN REVIEW

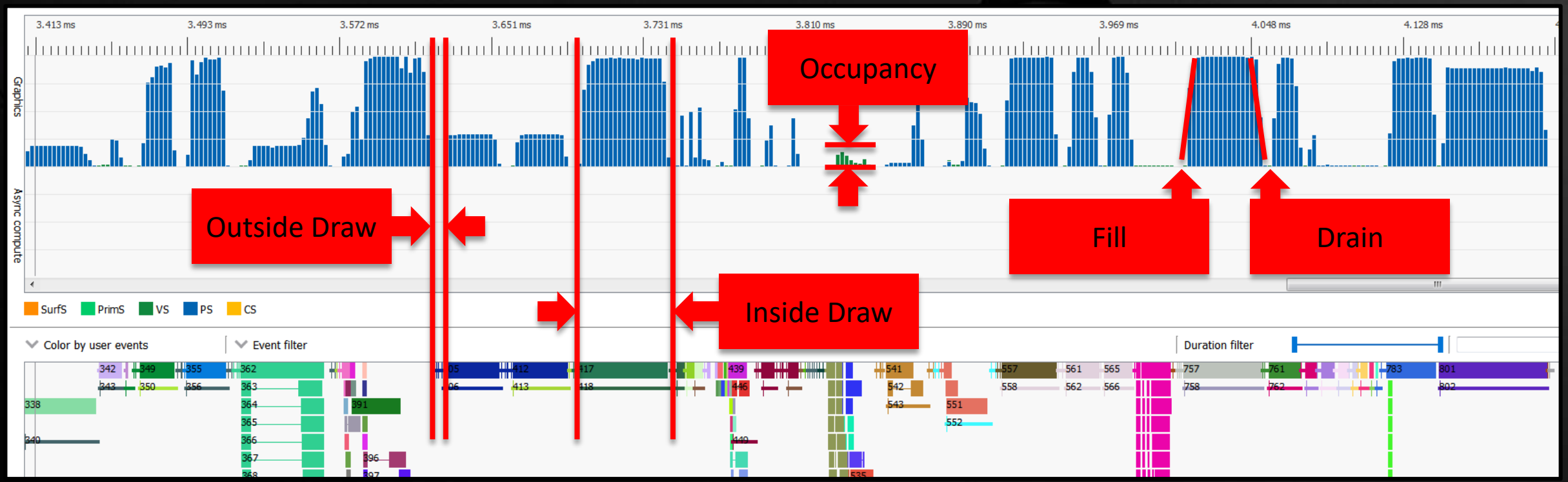
THIS TALK IS A FOLLOW-UP TO **ADVANCED SHADER PROGRAMMING ON GCN** FROM GDC 2017

- SIMD (VALU) = 16-wide vector ALU executes a 64-wide wave across 4 clocks
  - Maximum occupancy 8 waves/SIMD (Polaris) or typically 10 waves/SIMD (non-Polaris)
- CU = Compute Unit includes 4 SIMDs (peak VALU execution throughput of 1 wave/clock)
- SALU = Scalar ALU
- LDS = Local Data Store
- V$ = Vector L1 Cache (VMEM)
- I$ = Instruction L1 Cache
- K$ = Scalar L1 Cache (SMEM)



THINKING OUTSIDE THE DRAW

- Often optimization efforts are narrowly focused on just optimizing the most expensive shaders
- This talk takes a holistic view: **focusing on full-frame runtime inside and outside of all draws/dispatches**

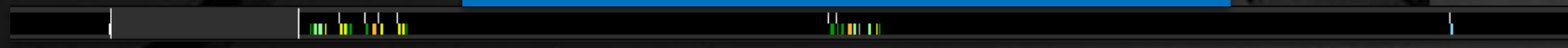


INTRODUCTION TO THE TOOLS USED INSIDE AMD FOR ANALYSIS

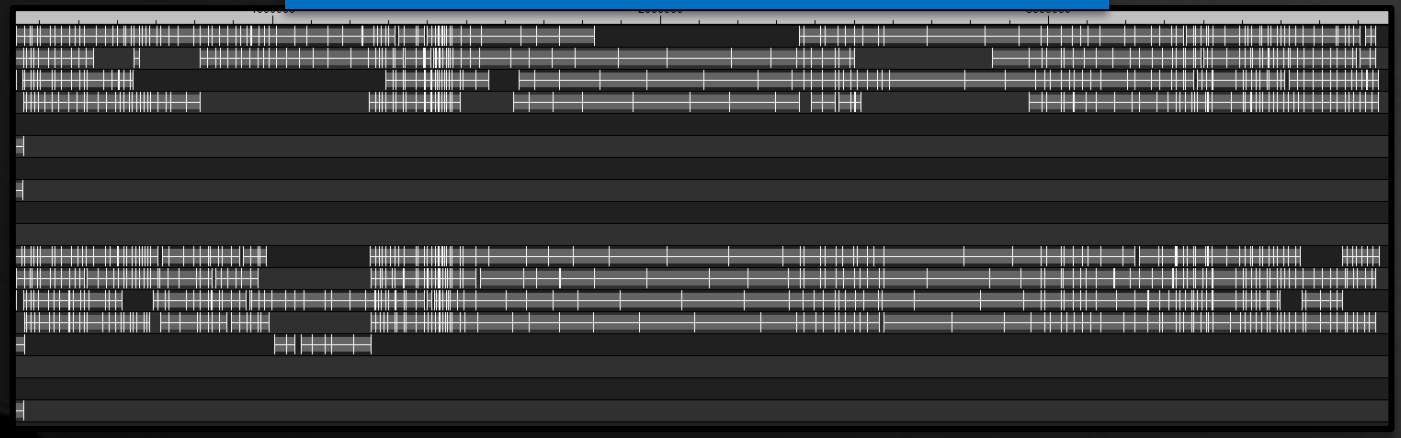
TRANSPARENCY – WITH THIS TALK YOU SEE WHAT WE SEE

- External Tools
  - Radeon GPU Profiler (RGP)
- AMD Internal Tools
  - VK_GPA_interface
  - Driver modifications
  - **Instruction Trace Report**

Wave Execution Detail



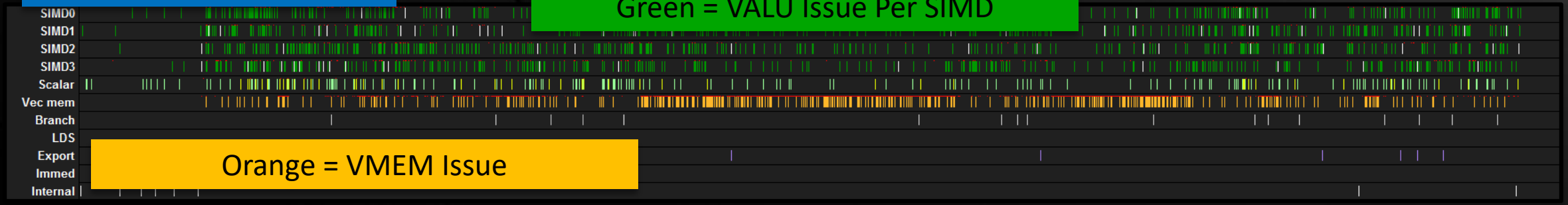
Lifetime of Waves



CU Instruction Summary

Green = VALU Issue Per SIMD

Orange = VMEM Issue



Core Principles

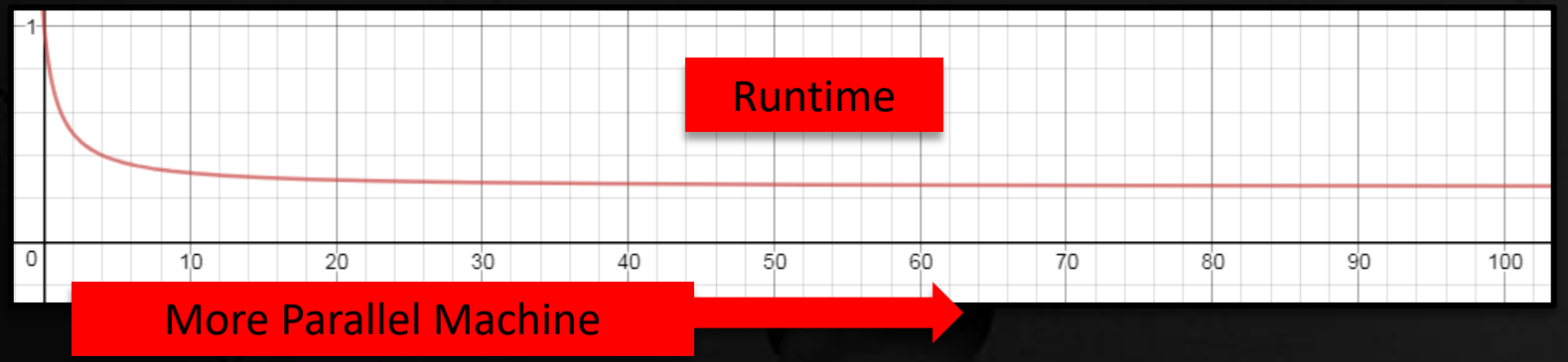
Optimization Theory for Parallel Machines

AMDAHL'S LAW

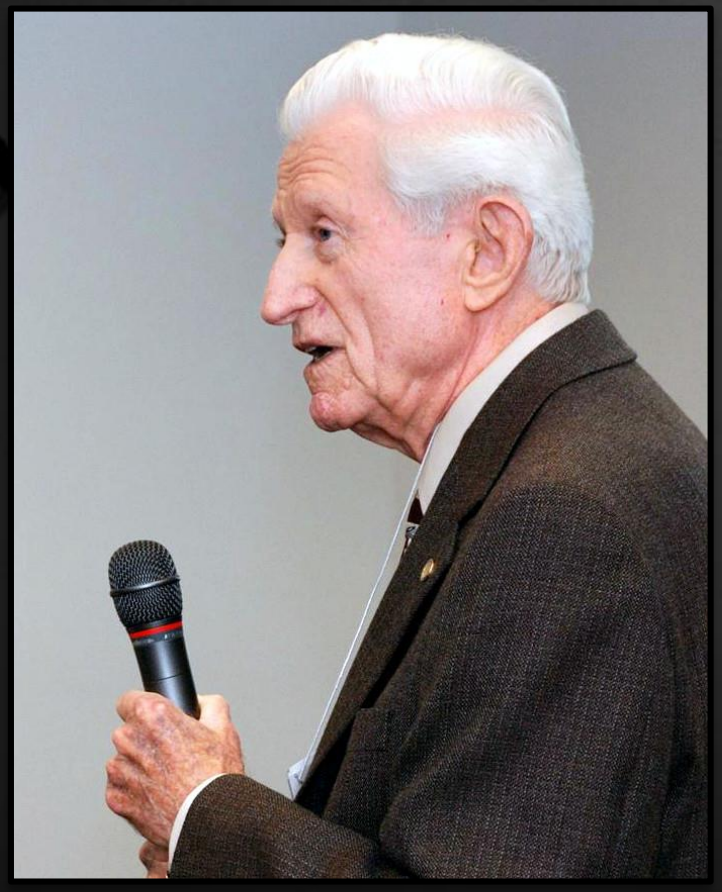
PRESENTED AT THE AFIPS SPRING JOINT COMPUTER CONFERENCE IN 1967

- Performance improvement possible as a machine scales in parallel capacity is proportional to the amount of parallel work

- At the GHz wall for example if only 75% of the work runs in parallel best case runtime will drop to 25% of a non-parallel machine



- Takeaway: **Keep GPU filled with pipelined parallel work!**

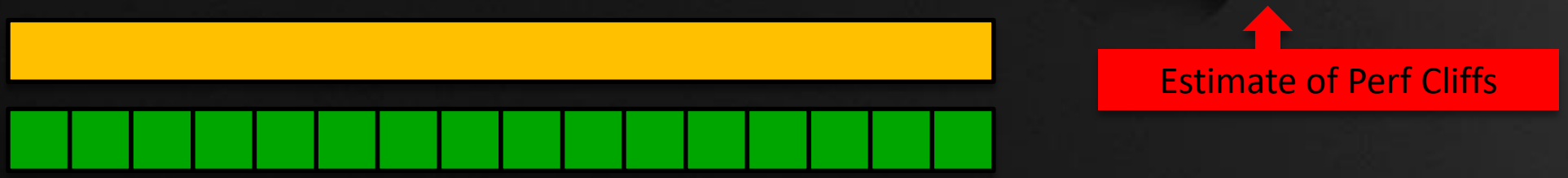


Creative Commons Photo by Pkivolowitz at English Wikipedia

UNDERSTANDING GCN VECTOR MEMORY ACCESS

OPTIMIZATION IS OFTEN ABOUT KEEPING THE GPU MEMORY SYSTEM FILLED PIPELINED PARALLEL WORK

- Average latency can vary by GPU/workload/etc
- Can estimate performance cliffs from synthetic shader latency testing results
  - ~114 clock L1 hit (peak 16 clock/VMEM return rate for image ops) (1 clock/VALU)
  - ~190 clock L2 hit (average ~76 clock extra latency compared to L1 hit …….. 76/16 = 4.75 image L1$ hits)
  - ~350 clock L2 miss (average ~236 clock extra latency compared to L1 hit ……. 236/16 = 14.75 image L1$ hits)

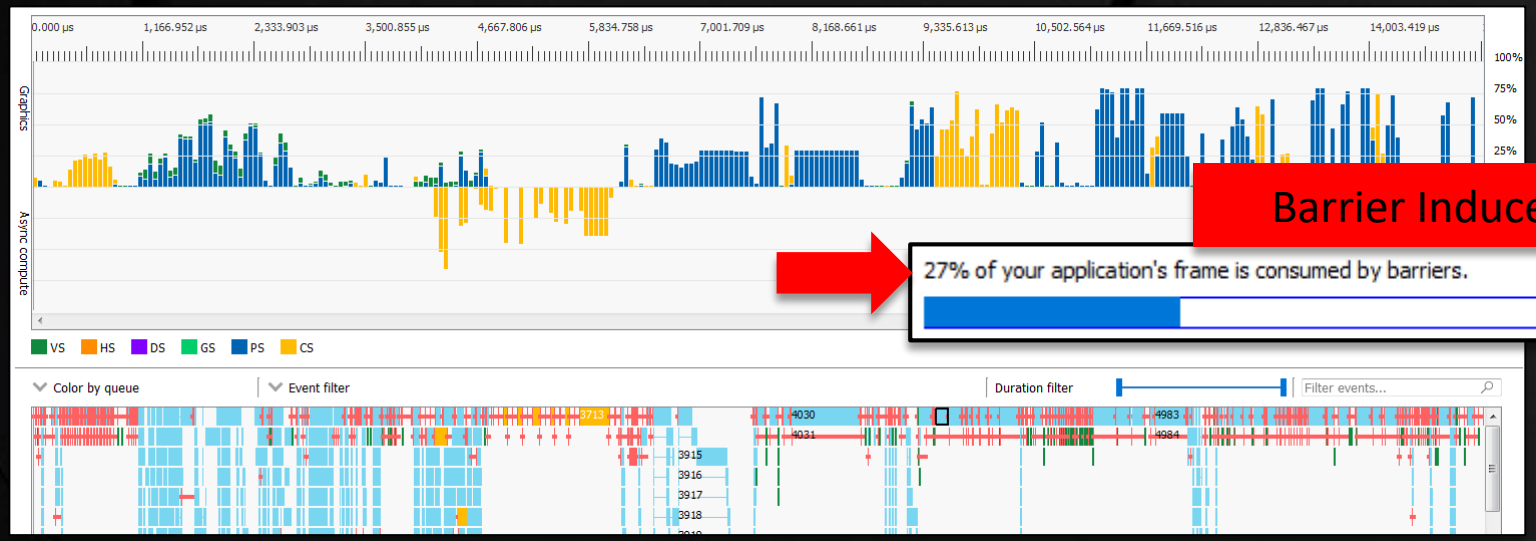


- Things which reduce memory throughput
  - Not sustaining VMEM ops fast enough to keep request queue from draining
    - **Try to sustain a regular flow of memory requests (look for wave occupancy irregularities, etc)**
  - Memory bubbles caused by L1$ misses or L2$ misses
    - **Try to improve data locality spatially and temporally, increase ratio of L1 hits**

Idle Bound

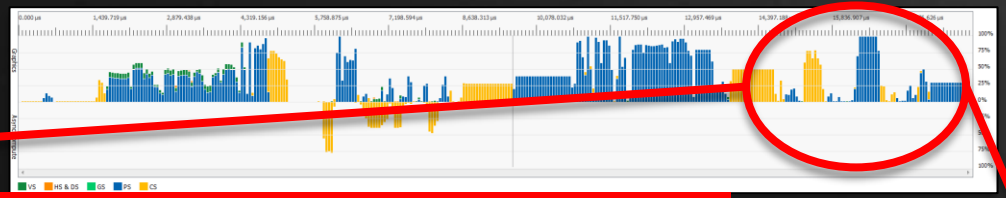
Failure to Keep GPU Fed With Pipelined Work

EXAMPLES OF BEING IDLE BOUND – SLIDE 1

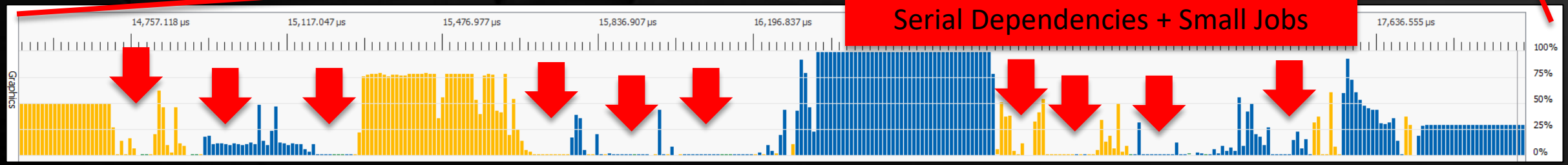


GPU Draining

Barrier Induced Idle

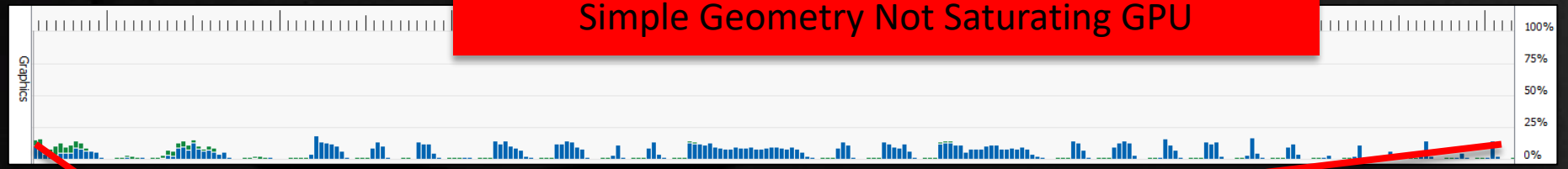


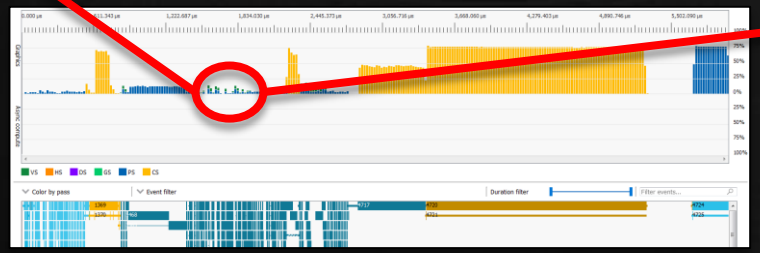
Serial Dependencies + Small Jobs



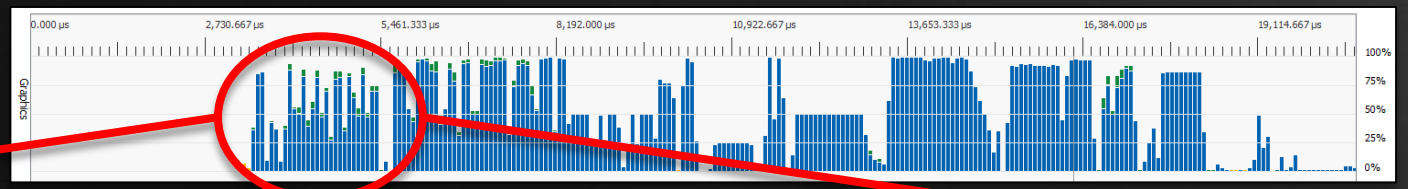
EXAMPLES OF BEING IDLE BOUND – SLIDE 2

Simple Geometry Not Saturating GPU

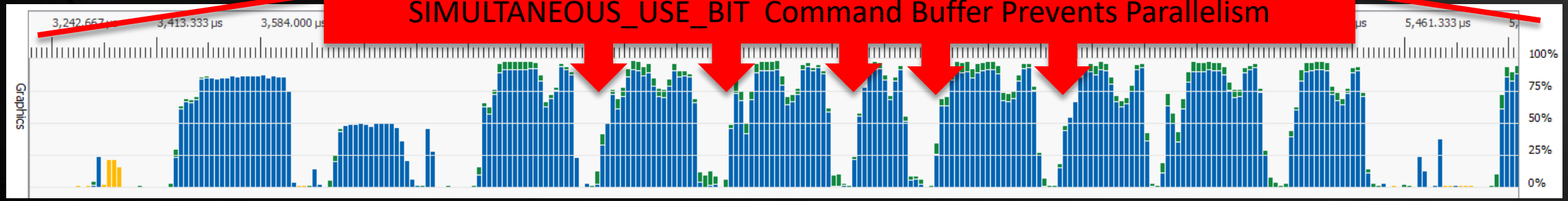




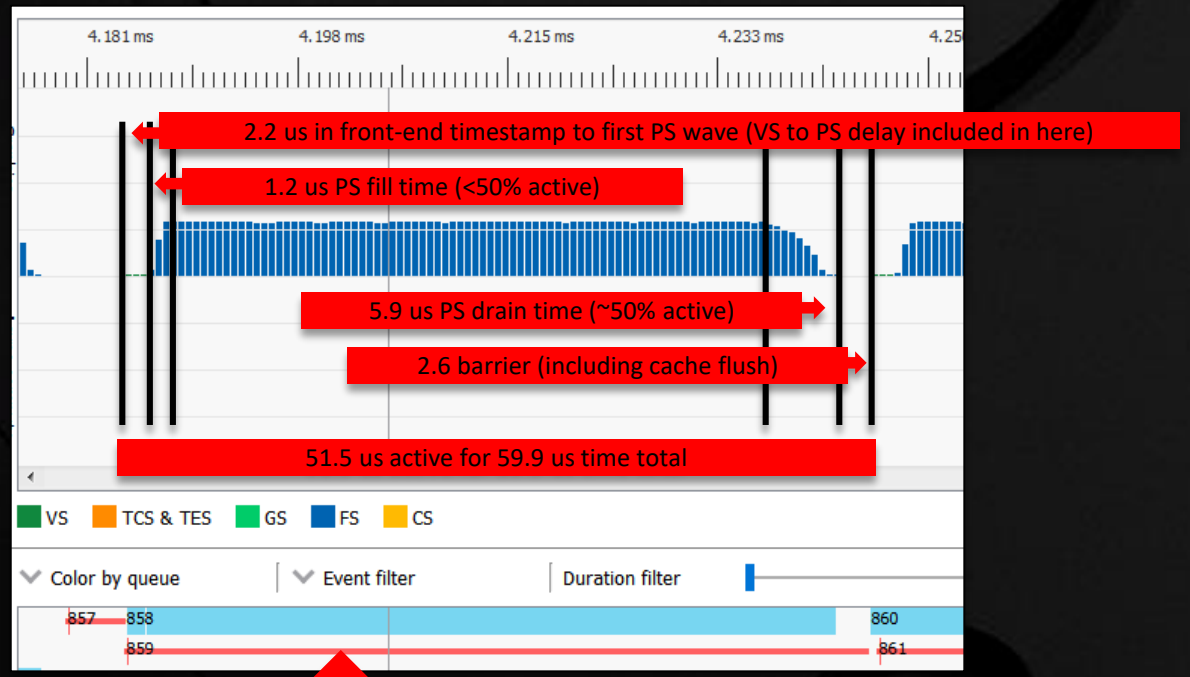
GPU Draining



SIMULTANEOUS_USE_BIT Command Buffer Prevents Parallelism

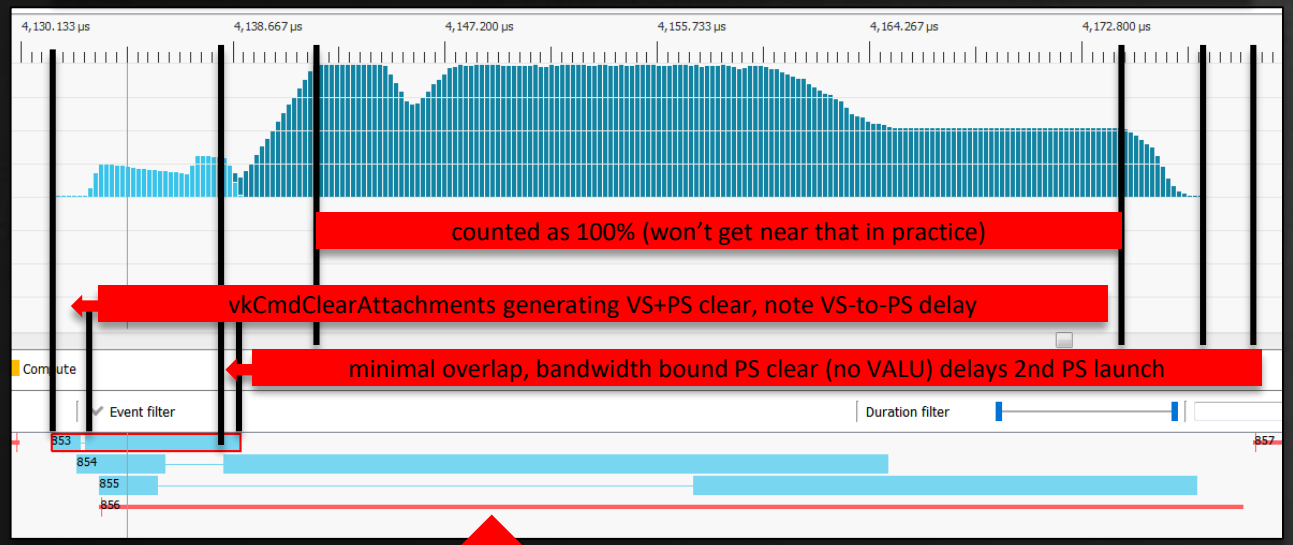


NUMBERS FOR DRAIN AND FILL OF 1/4 RESOLUTION PASSES



GPU at Least 14% VALU Idle in this Post Pass

PS clear in parallel with independent 2nd and 3rd PS draws (35 us VALU work for 48 us time)

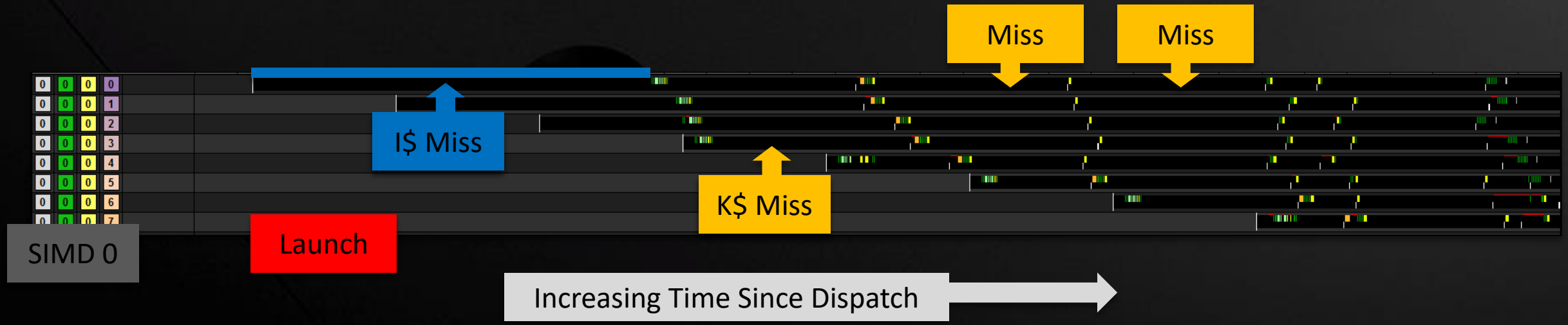


GPU at Least 26% VALU Idle in this Post Pass + Clear

FILLING THE GPU – A STORY OF COLD CACHES

LOOKING AT WAVE LAUNCH FROM AMD-INTERNAL INSTRUCTION TRACE

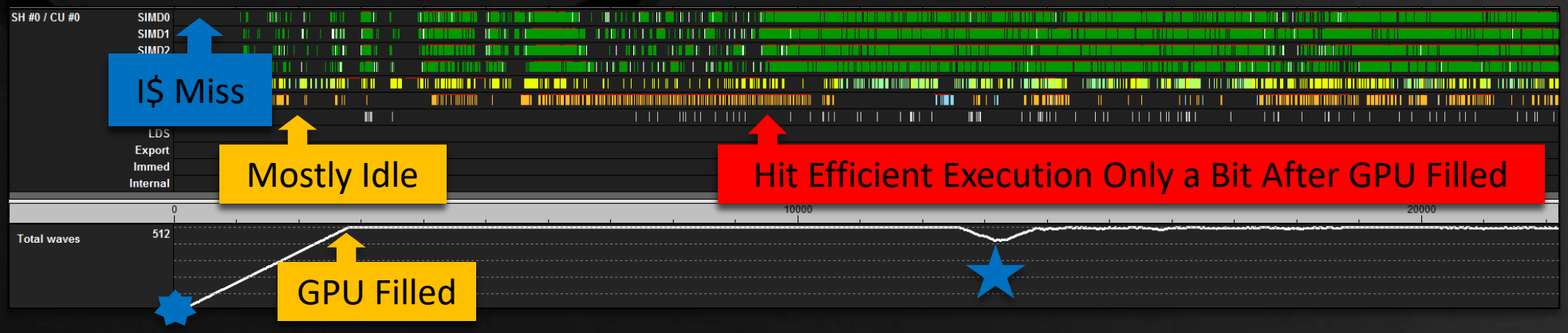
- 64 CUs (Vega) = capacity to launch 2560 waves (~160 Ki work items if at peak occupancy)
- "First-Fill Waves" = waves starting in cold cache conditions
  - Up to 4.4% of a 2560x1440 full-screen pass is first-fill waves
  - Up to 17% of a 1/4 area post pass is first-fill waves (common with DOF/MB/etc)
  - Up to 71% of a 1/16 area post pass is first-fill waves (common with reduction passes)
- Significant amount of GPU time can be spent with cold caches
  - CS dispatch: wave launch rate, I$ missing, K$ miss and dependent fetches
    = **long latency chain which cannot be hidden on idle machine**



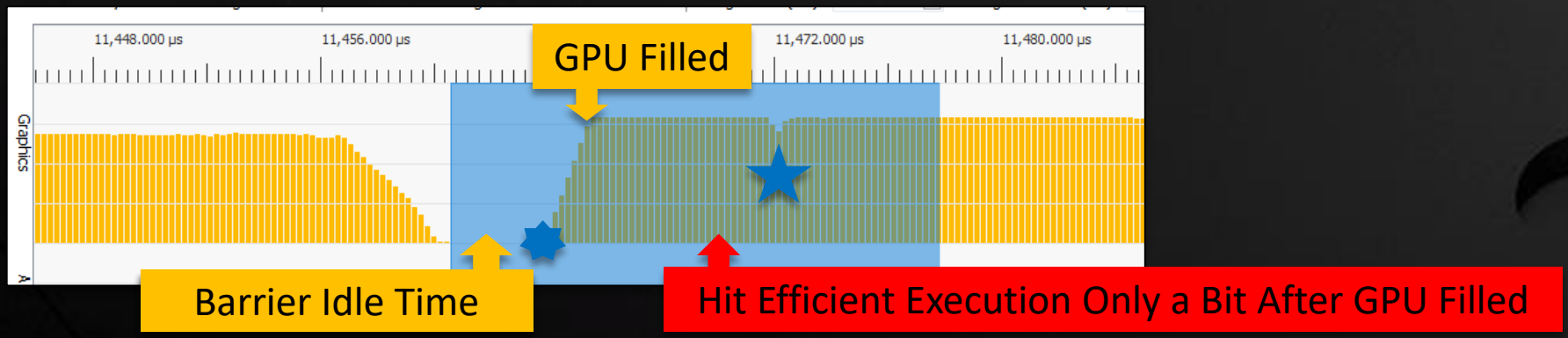
STORY OF COLD CACHES RELATED BACK TO RGP

WAVE OCCUPANCY != A MEASURE OF EXECUTION EFFICIENCY

- AMD-Internal Instruction Trace



- View From RGP



IDLE BOUND – TAKEAWAY

AMD

- **Amdahl's law in practice**
- Time outside the draw matters
- Serializing draws = performance stopped scaling with GPU size
- Time to pipeline

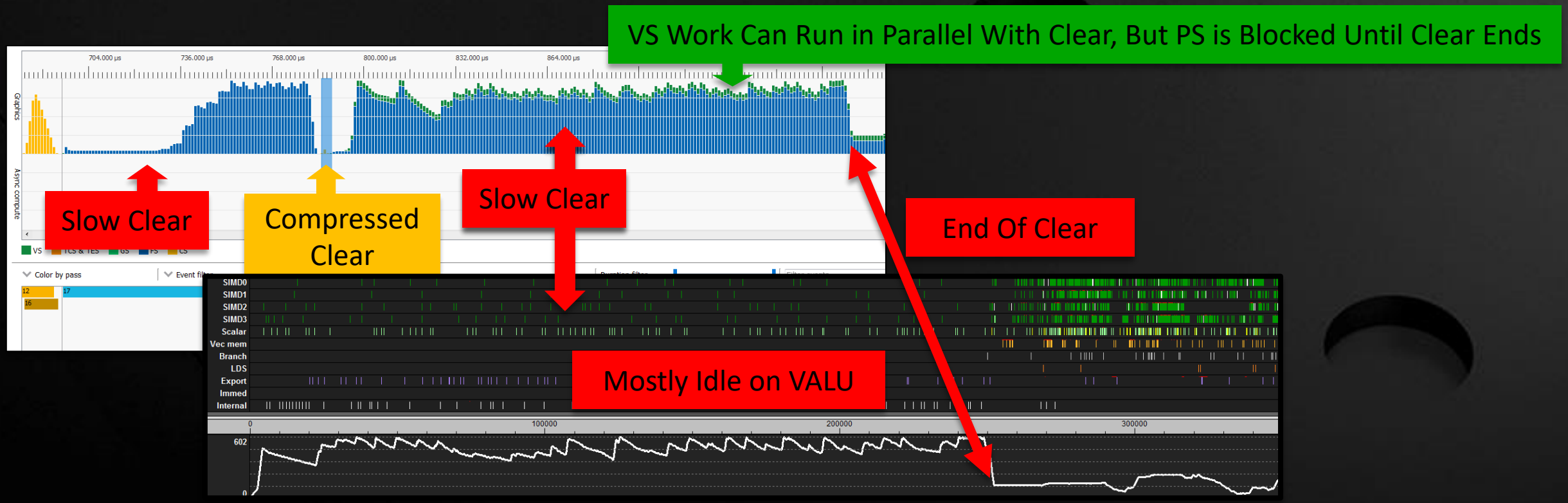
DCC & CS VS PS

Tradeoffs

AVOID SLOW CLEARS

ONE MAJOR MOTIVATION FOR COMPRESSION

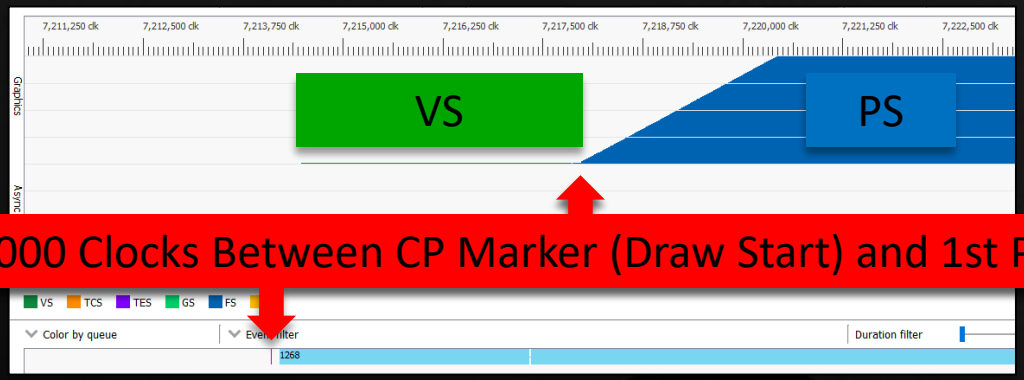
- Avoid clears to images written in CS passes
- Avoid clears to images getting full-screen triangles (aka overwrite majority of image)
- For compressed surfaces advantage via clearing to all 0.0 or 1.0



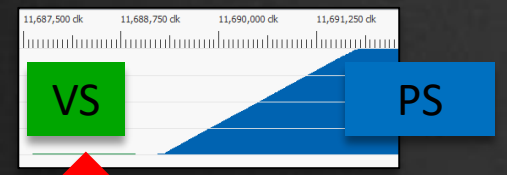
PS AND CS ON GCN

COMPARING PROS AND CONS

- PS – Launches typically between 1-4 waves/V$ before moving to next V$ – **Ordered export**
  - Supports DCC|Z compressed output (note compression can increase latency of loads)
  - Cache flush to read render targets written in prior pass
  - **VS to PS delay can be substantial when VS has long latency chain and cold caches**



Almost 4000 Clocks Between CP Marker (Draw Start) and 1st PS Wave Launch



Less Cache Missing = Lower VS to PS Delay

- CS – Launches workgroup/V$ before moving to next V$ – **No export (unordered)**
  - Can read compressed image written in PS pass, but cannot store to DCC|Z compressed images
  - Does not need a cache flush on shader read to shader write transition
  - Easy to run pipelined in graphics queue or async in CS queue

DELTA COLOR COMPRESSION (DCC) ON/OFF

AMD

- DCC on PC GPUs has possible advantages and disadvantages
  - Possible reduction in bandwidth for an increase in latency (need to fetch meta-data)

- **Best to profile to choose when to use and not use DCC**
  - Can disable DCC on a render target by using STORAGE or UAV usage flags
  - Better to try individual render targets instead of just all on vs all off

- Example performance delta with DCC enabled for some game passes (Vega numbers)
  - -5.0% for deferred shading (slower)
  - -3.1% for screen space reflections
  - 1.8% for screen space occlusion
  - 2.2% for compositing pass
  - 3.8% for post processing (faster)

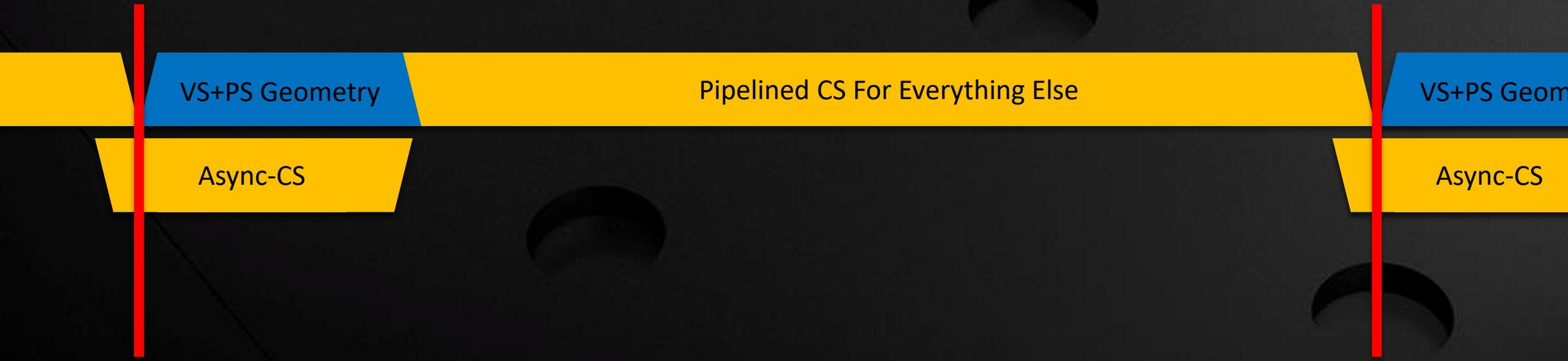
Pipelining With Events

Keeping the Machine Busy

AUTHOR'S OPINION OF IDEAL TRIANGLE-BASED ENGINE

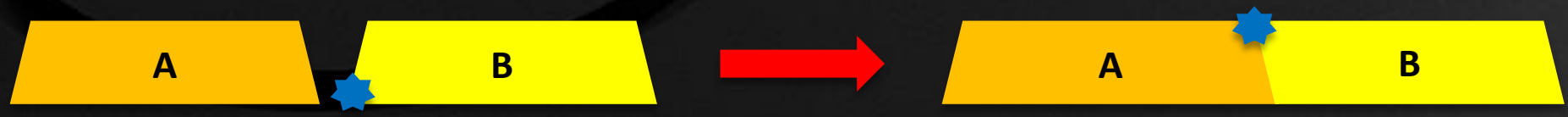
WHEN TO USE CS AND VS+PS

- VS+PS only when rendering geometry, CS for everything else
- **CS maximizes ability to easily pipeline**



PIPELINING

- Hide the latency of cold caches during fill in the drain of the prior work

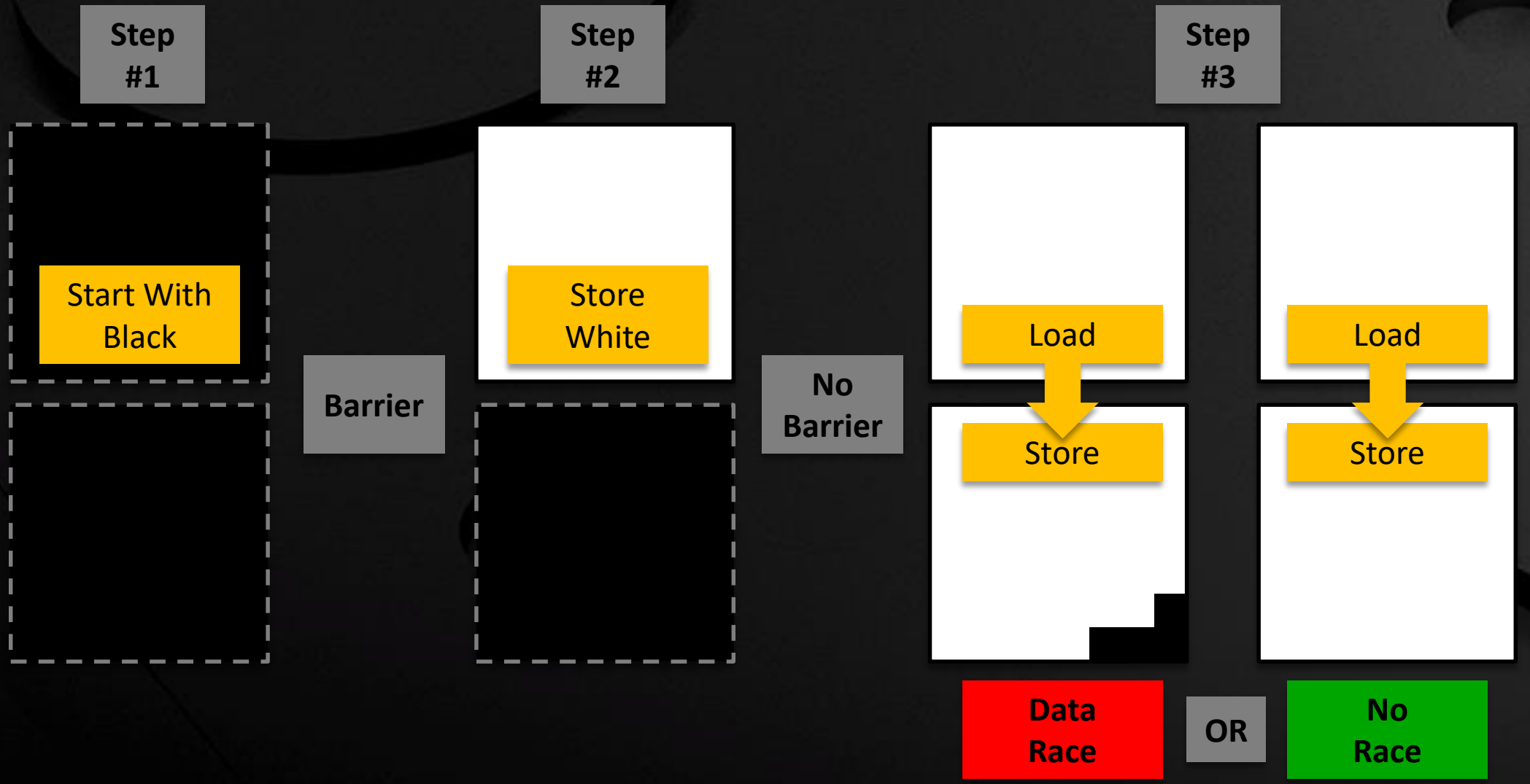


- Going to explore an API-illegal technique to pipeline first: remove the barrier and test for data race
  - Useful for understanding hardware
  - Useful to try and estimate performance upper bound to test against for API-legal solution
  - Useful to prove some kind of execution dependency is required

- Following up with an API-legal solution in later slides

SIMPLE DATA RACE TESTER

TWO 32-BIT/PIXEL IMAGES, THREE COMPUTE DISPATCHES

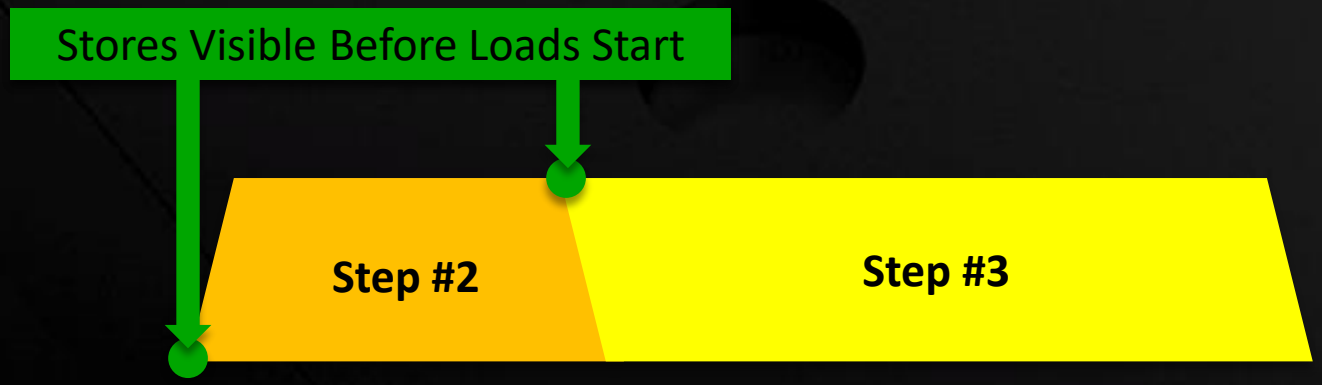


SIMPLE DATA RACE TESTER : RESULTS VALIDATE DATA RACE IS POSSIBLE

TWO 32-BIT/PIXEL IMAGES, THREE COMPUTE DISPATCHES

- Results running various image sizes on an RX 580
  - No Race : 256x256 and Larger
  - **Data Race : 128x128 and Smaller – Validation that some form of execution dependency is actually required**

- Combination of factors make data race not probable in some cases
  - Non-coherent (GLC=0) stores still write-through to coherent L2$
  - **Step #2** waves need to launch before **Step #3** waves (API submission-order launch, out-of-order completion)
  - GPU's loose workgroup launch order for dispatch (roughly right to left, then top to bottom)



Cannot Depend On This Behavior
GPU Size and Workload
Can Adjust Probability
Of Getting Data Race!

LEVERAGING COHERENT MEMORY QUALIFIER IN GLSL

TOOL TO REMOVE THE NECESSITY TO FLUSH CACHES FOR BUFFER/IMAGE STORE TO LOAD TRANSITION

AMD

- Coherent translates to GLC=1 store on GCN which bypasses L1$ and stores to a coherent L2$
  - Example: layout(set=0, binding=5, rgba8) uniform **coherent** image2D imgA;

- ARB_shader_image_load_store – *"Buffer object or texture image memory accessed through such [coherent] variables may be cached only if caches are automatically updated due to stores issued by any other shader invocation"*
  - Explicitly states that coherent stores need to bypass non-coherent caches

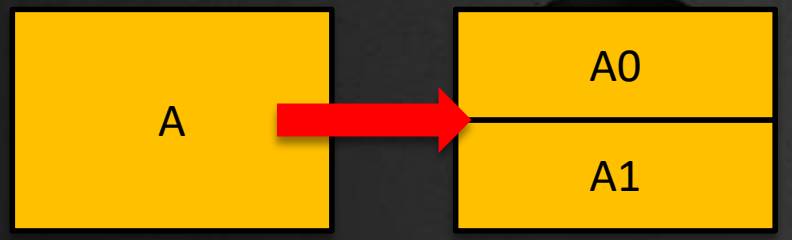
- **Crucial observation**
  Loads need not use coherent qualifier and can thus be cached under the following usage model
  - Beginning/end of frame cache flush to guarantee fresh caches
  - Write cachelines only once with coherent stores before any read (only need execution dependency, aka vkEvent)
  - Read cachelines via non-coherent caches (aka highest performance) any number of times

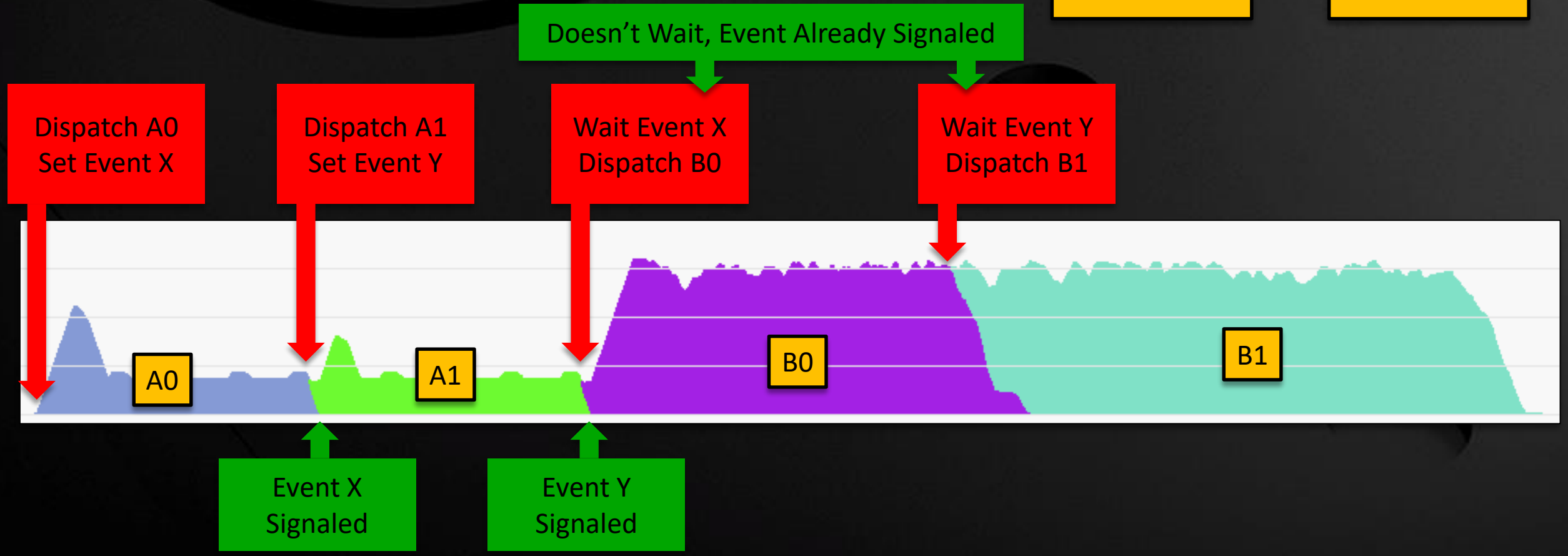
The Perfect Tool For Pipelining In Vulkan®

MECHANICS OF PIPELINING WITH EVENTS

EVEN SERIALLY DEPENDENT PASSES CAN OFTEN LEVERAGE EXPLICIT SPATIAL INDEPENDENCE

- Manually split dispatch into two dispatches
  - Splits will have to vary based on maximum filter kernel window size, etc

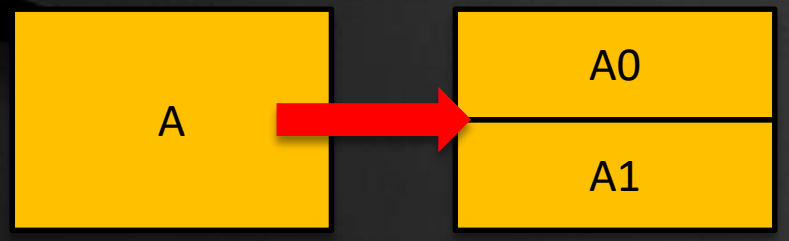




WRITE ONCE COHERENT BEFORE ANY READ, THEN READ CACHED

CONSIDERATIONS FOR SAFE USAGE

- API does not expose GPU cacheline size and hardware is free to change cacheline size
- As of March 2018, there is no API interface to report “safe” split line granularity
  - To make sure split does not straddle a cacheline boundry
- Current workaround is to greatly overestimate required alignment
  - Split offset should be a large powers of two
  - 32-bit/texel mixed with one hardware swizzle pattern on 64-byte cacheline might require 4 texel alignment
  - Another image with 16-bit/texel on a machine with 128-byte cacheline might require 8 texel alignment
  - **Better to pad to say 64 texel alignment – leaves flexibility for change in cacheline size and swizzle patterns**

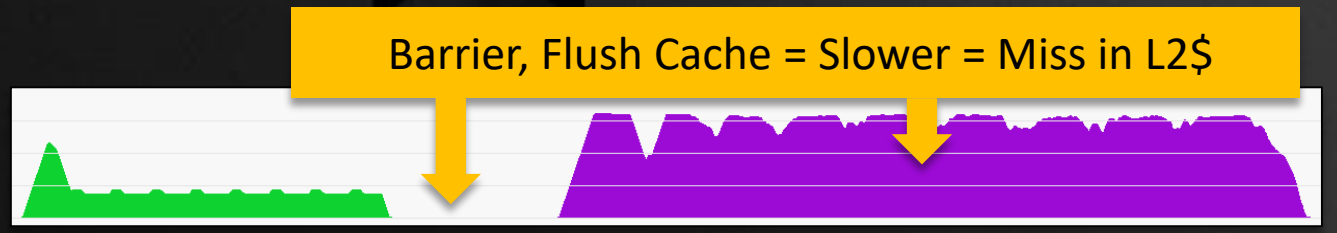


SIMPLE DATA RACE TESTER : VARIATIONS AT 1152 X 1152 ON VEGA

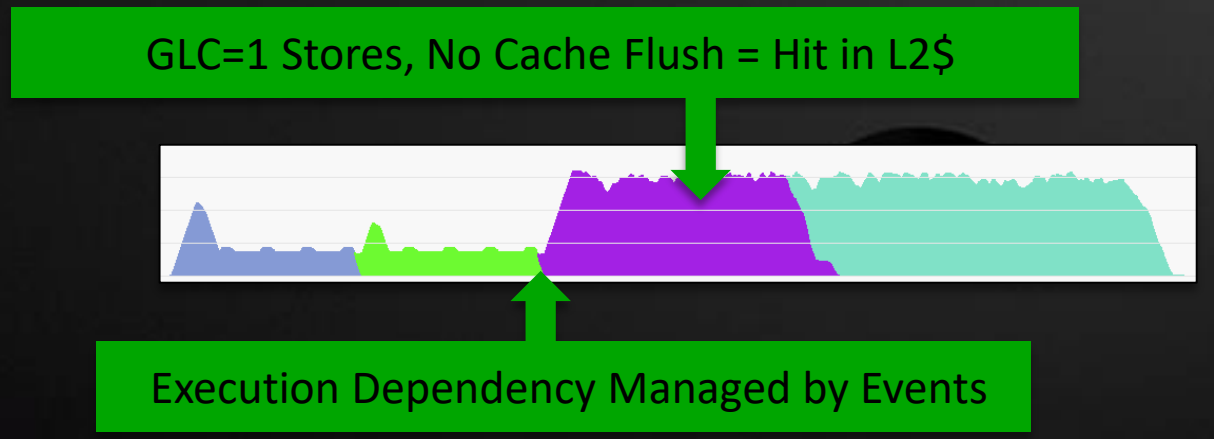
MEASURED NORMALIZED TIME (FIRST WAVE TO END OF LAST WAVE)

- **1.00 time : Data Race Tester (Not API-Legal)**
  - Possible upper bound on limit of manual pipelining

- **1.31 time : Re-Introduce The Barrier**
  - API-legal common practice (time includes barrier)
  - Overly conservative driver flushed L2$
    - Or maybe barrier with usage requiring L2$ flush



- **1.03 time : Manually Split Pipeline Via Events**
  - Use “coherent” GLC=1 stores instead of image barrier
  - **Suggested “safe” and API-legal solution**



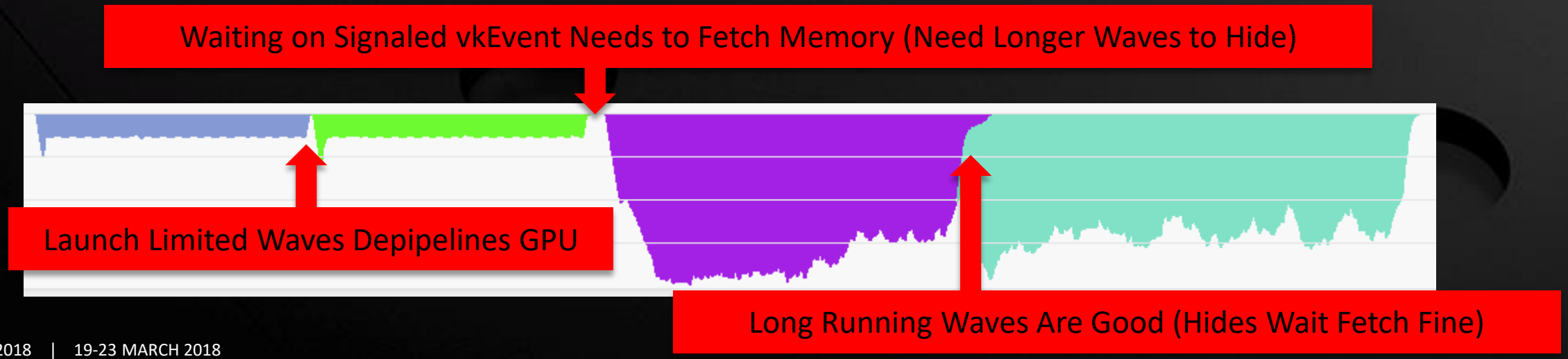
DIFFERENCES BETWEEN CS AND GFX QUEUE

LOOKING AT SAME OPERATIONS IN DIFFERENT QUEUES

AMD

- Graphics queue is better at pipelining front-end operations (often see 40+ draws in parallel)
  - Often better to pipeline small jobs in the graphics queue

- Compute queue needs longer running waves to hide front-end overheads

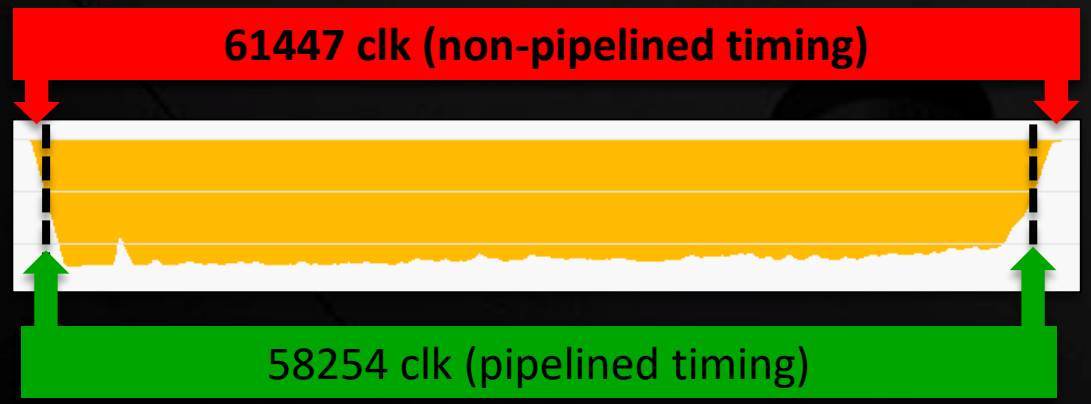


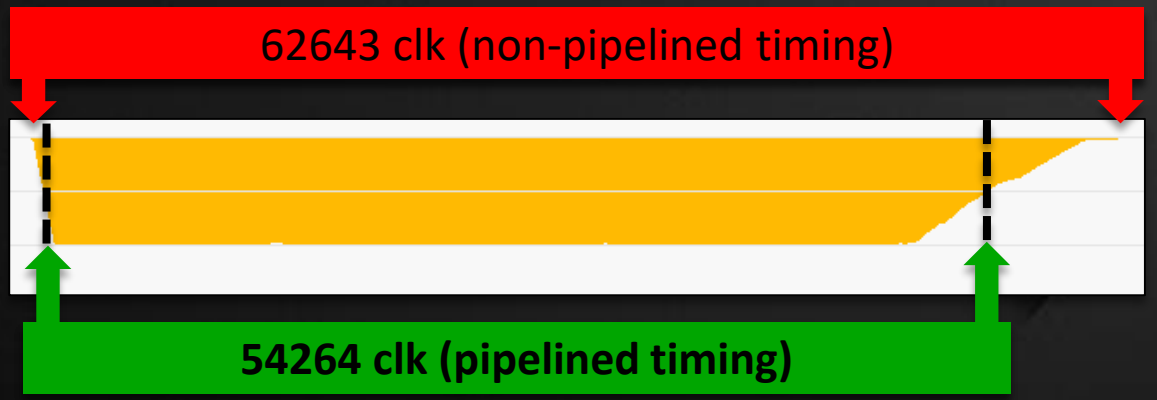
MEASURING ISOLATED TIMING IN A PIPELINED WORLD

THE CHALLENGES OF SHORT DISPATCHES, LONG RUNNING WAVES, AND LARGE GPUS

- **Old non-pipelined world focuses on minimizing first wave launch to last wave exit time**
  - Which implies minimizing fill and drain time, which implies avoiding long running waves (due to long drain)!
- **New pipelined world focuses on minimizing average "wave run-time / wave occupancy"**
  - Changes what to optimize for (not interested in minimizing drain time)
  - Changes what to measure for shader timed in isolation (take drain and fill into account)

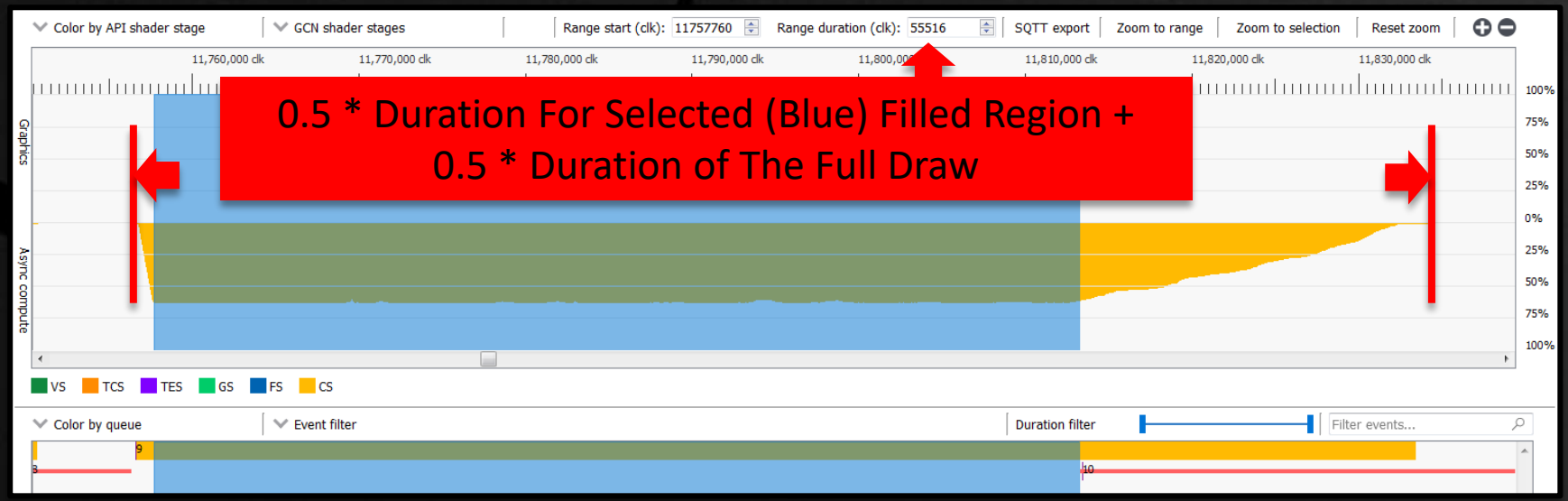
- Which shader is **faster**? (depends on if dispatch is to be pipelined or not)



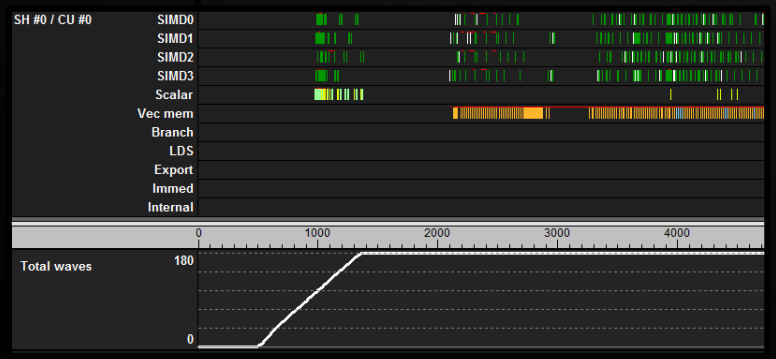


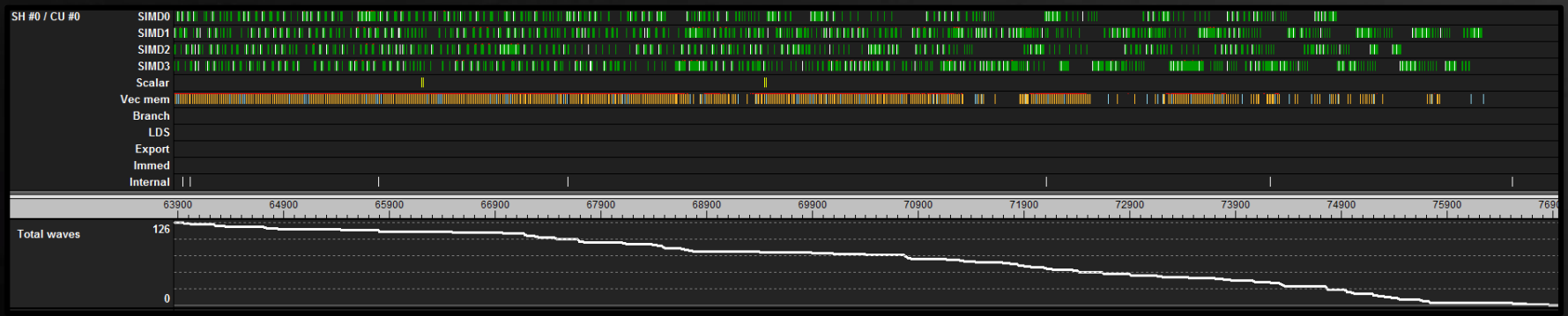
ESTIMATING PIPELINED TIMING IN RGP – ROUGH BUT USEFUL

USING BALLPARK ESTIMATE OF 50% UTILIZATION IN DRAIN AND FILL REGIONS



- Motivation of 50% utilization estimate in drain and fill from ramp of utilization see in instruction traces





PIPELINING WITH EVENTS – TAKEAWAY

AMD

- Vulkan®
  - **Pipelined CS works great now via vkEvents and “coherent” keyword**
  - Good gains possible from avoiding cache flushes
  - Good gains possible from avoiding draining the GPU
  - As of this talk (March 2018)
    - Barriers : Possibly overly conservative cache flushes in the driver, we are looking to improve

- Direct3D® 12
  - Pair independent work to avoid barriers and leverage Async-CS to fill graphics drains
  - The HLSL version of “coherent” is “globallycoherent”
  - As of this talk (March 2018)
    - Split Barriers : Possibly overly conservative, we are looking at options to improve

GPU Utilization

Visual Review of Instruction Traces

COLLECTION OF INSTRUCTION TRACES FROM VARIOUS WORKLOADS

ONE IMPORTANT TREND – **NONE OF THESE ARE FULLY VALU BOUND, MOST HAVE A LOT OF VALU IDLE TIME**

Down Sample

Motion Blur

Draw GUI

Light Occlusion

Horz Blur

SSR

G-Buffer Fill

SSAO

Deferred Shading

TAA

Tonemap & More

4 Almost Empty Green SIMD Rows
Approaching VALU Idle

Red Arrows
Fully Memory Bound

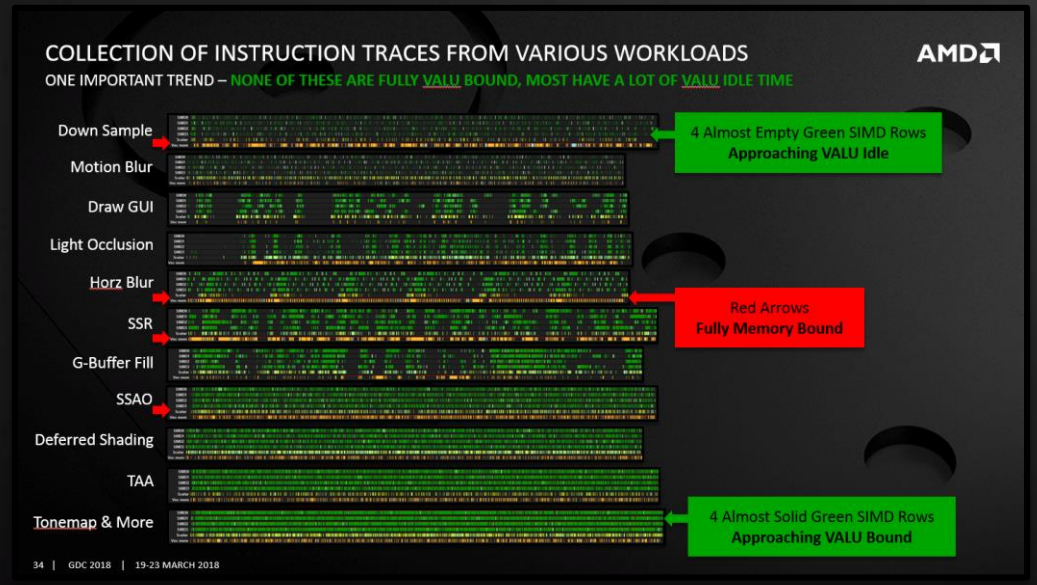
4 Almost Solid Green SIMD Rows
Approaching VALU Bound

UTILIZATION TRENDS FROM “INSIDE THE DRAW/DISPATCH”

AND BASIC THEORY ON HOW TO OPTIMIZE

- **The GPU is often VALU idle = Huge untapped capacity to do work!**
  - Engineer to avoid being bottlenecked by fixed function limits
  - Merge passes to avoid being memory bound (for instance down-sample in one pass)
  - Leverage async-compute
  - . . .

- **Some passes are fully saturating the VMEM request queue**
  - Yet can still be optimized
  - Focus on improving cache locality
  - By improving work to V$ mapping
  - By improving scheduling of work
  - By improving access patterns
  - . . .



Wave Launch Rate

Larger GPUs Prefer Longer PS Waves

GCN WAVE LAUNCH RATE VARIES ACROSS GPUS

ONE OF THE CORE DIFFERENCES BETWEEN LARGE PC GPUS AND CONSOLES

AMD

- SE = Shader Engines – Each SE can launch 1 wave every 4 clocks
- CU = Compute Units – Number of CUs determines the size of the GPU

- SE:CU ratio determines how fast the machine can fill itself
  - **1:16 – Fury X, Nano, Vega 64**
  - 1:14 – Fury, Vega 56
  - 1:11 – R9 390X
  - 1:10 – XBox 1X
  - 1:9 – PS4, PS4 Pro, RX 480, RX 580
  - 1:8 – R9 380X, RX 470, RX 560, RX 570
  - 1:7 – RX 460
  - **1:6 – XBox 1**

Larger GPUs Take Longer to Fill, Prefer Longer Running Waves

Smaller GPUs Fill Faster

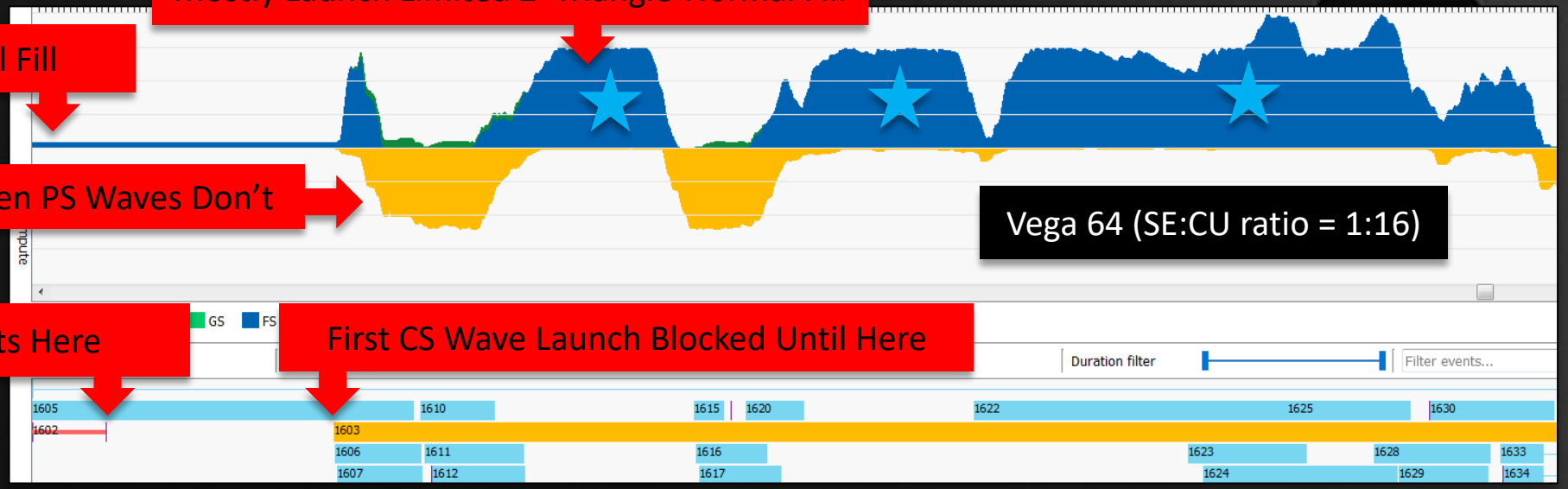
SHORT RUNNING WAVES CAN BOTTLENECK LARGER GPUS

EXAMPLE OF WAVE LAUNCH FOR ASYNC-COMPUTE GETTING BLOCKED BY WAVE-LAUNCH-LIMITED PS DRAWS

Mostly Launch Limited Z+Triangle-Normal Fill

Wave Launch Limited Stencil Fill

Async CS Wave Can Only Launch When PS Waves Don't

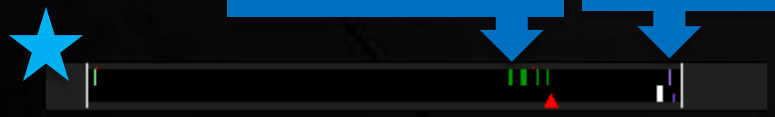


CS Dispatch Starts Here

First CS Wave Launch Blocked Until Here

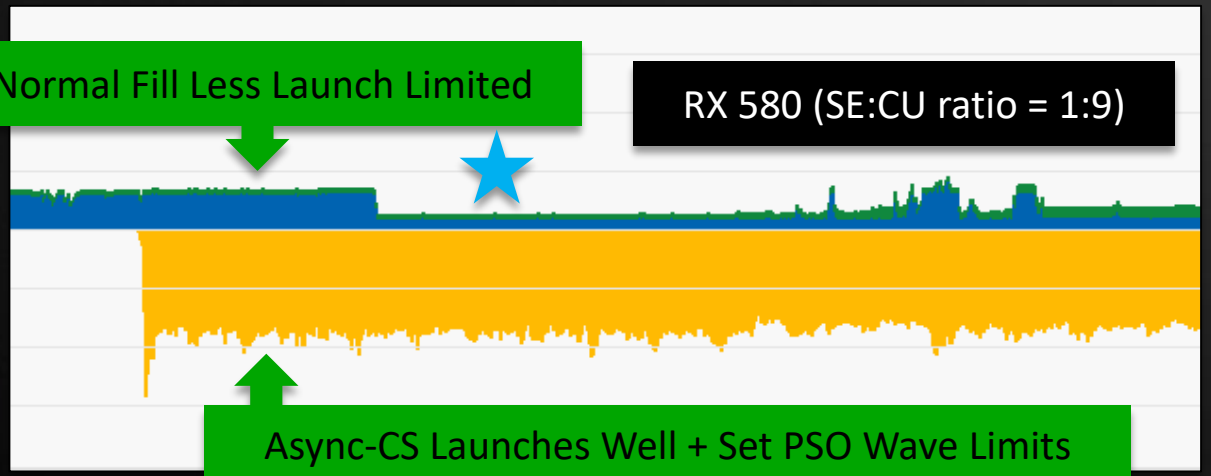
A Few VALU ops

Export Z+Tri Norm



PS Wave Instruction Trace : Nearly Empty and No Loads

Z+Triangle-Normal Fill Less Launch Limited



Async-CS Launches Well + Set PSO Wave Limits

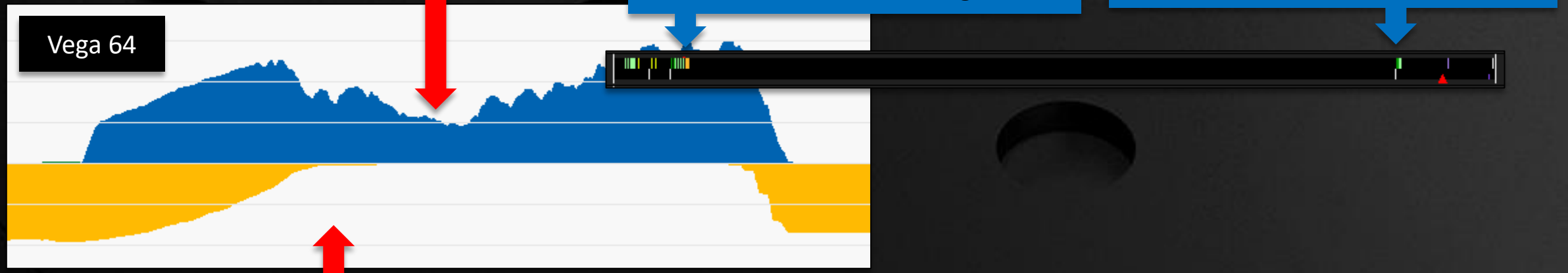
ANOTHER EXAMPLE OF SHORT RUNNING PS STALLING ASYNC-COMPUTE

BEST TO HAVE LONG RUNNING PS OR NO PS (LIKE Z-ONLY RENDERING FOR Z-PRE-PASS OR SHADOWS)

Simple PS With One Fetch

Instruction Trace Shows Single Fetch

Wait on Fetch, Some Math, Export



PS Stalls 4-Wave Workgroup Async-CS (Watch Out For Workgroup Sizing Issues)

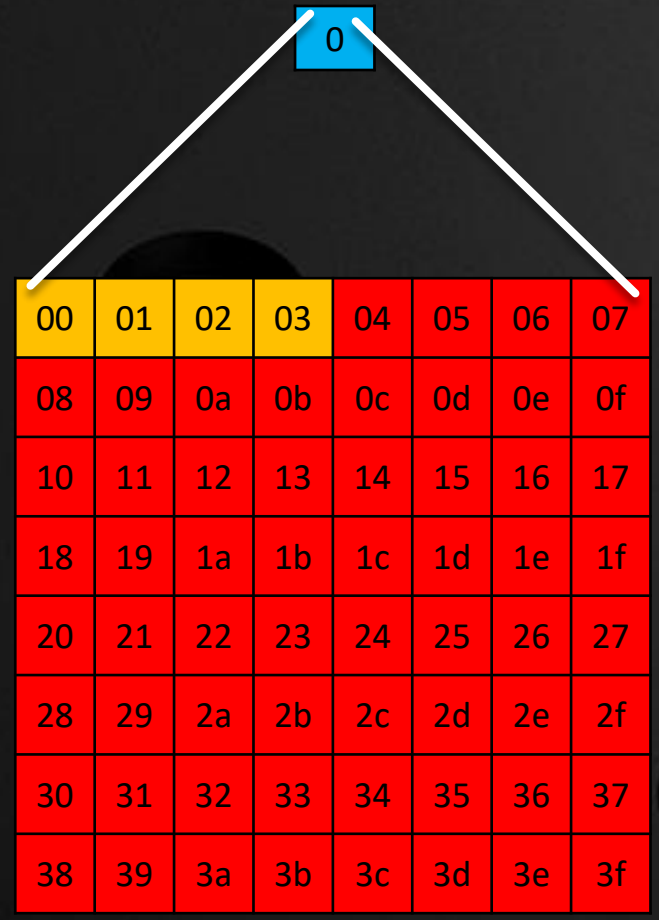
Same Async-CS Dispatch Overlaps Good With Shadow Rendering (No PS To Stall Async-CS)

Workgroup Optimization

2D Locality vs Workgroup Launch Rate

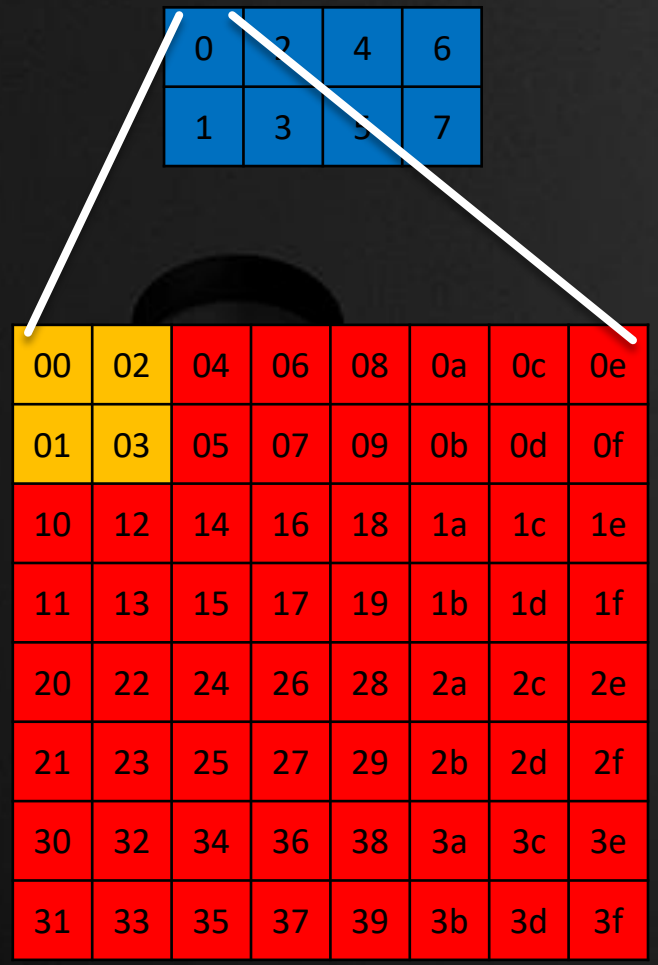
TYPICAL PRACTICE OF PS TO CS CONVERSION

- Move PS to a {8,8,1} workgroup in CS
  - 1 wave/V$ for locality (can be worse than PS)
  - GCN launches one workgroup per CU (one V$/CU)
    Before moving to the next CU
- Lane to 8x8 tile mapping is “linear block”
  - Possible poor {4x1} pattern for texture fetch quad
  - Possible V$ bank conflicts
  - GCN sampling works in groups of 4 lanes



EXAMPLE “OPTIMIZED”*** CONFIGURATION

- {512,1,1} workgroup configured to {32,16,1}
  - 8 wave/V$ for locality
  - Each wave is a 8x8 tile
  - 8 waves organized into 4x2 collection of 8x8 tiles
- Lane to 8x8 tile mapping is “swizzled block linear”
  - Good {2x2} pattern for texture fetch quad
- *** Optimized for a specific shader
  - Which is highly dependent on 2D locality for cache hits
  - And less dependent on wave launch issues



{512,1,1} TO {32,16,1} RECONFIGURATION CODE FOR PRIOR SLIDE

GLSL CODE

```
// 5 ops on GCN of extra VALU overhead (minimal)
uvec2 Remap(uint a) {
     uint y = bitfieldExtract(a,3,4); // v_bfe_u32 ---> {...0,y3,y2,y1,x2}
     y = bitfieldInsert(y,a,0,1);     // v_bfi_b32 ---> {...0,y3,y2,y1,y0}
     uint x = bitfieldExtract(a,1,3); // v_bfe_u32 ---> {...0,x2,x1,x0}
     a = bitfieldExtract(a,4,5);      // v_bfe_u32 ---> {...0,x4,x3,y3,y2,y1}
     x = bitfieldInsert(a,x,0,3);     // v_bfi_b32 ---> {...0,x4,x3,x2,x1,x0}
     return uvec2(x, y); }
```

```
// usage in shader
uvec2 xy = Remap(gl_LocalInvocationID.x);
// 2 ops on Vega
xy.x += gl_WorkGroupID.x << 5; // v_lshl_add_u32
xy.y += gl_WorkGroupID.y << 4; // v_lshl_add_u32
```

```
REMAP

876543210 <- input bits
---------
  yyy   y <- output bits
  321   0
xx   xxx
43   210
```

CS SSAO MEASURED PERFORMANCE FOR DIFFERENT CONFIGURATIONS

3440X1440 RESOLUTION

AMD

- Optimized drops 43% off runtime (yellow cases)
  - **{64 ,1 ,1} remap to {8, 8, 1} : 1.93 ms (base but with swizzled block)**
  - {128 ,1 ,1} remap to {8, 16, 1} : 1.69 ms
  - {8 ,32,1} no-remap : 1.57 ms (linear block)
  - {256 ,1 ,1} remap to {8, 32, 1} : 1.36 ms
  - {512 ,1 ,1} remap to {16, 32, 1} : 1.24 ms
  - {1024,1 ,1} remap to {8, 128, 1} : 1.22 ms (workgroup too big for non-pinned workload)
  - {512 ,1 ,1} remap to {8, 64, 1} : 1.15 ms (easier VALU conversion, but worse L1$ perf)
  - **{512 ,1 ,1} remap to {32, 16, 1} : 1.09 ms (using transform from prior slide)**

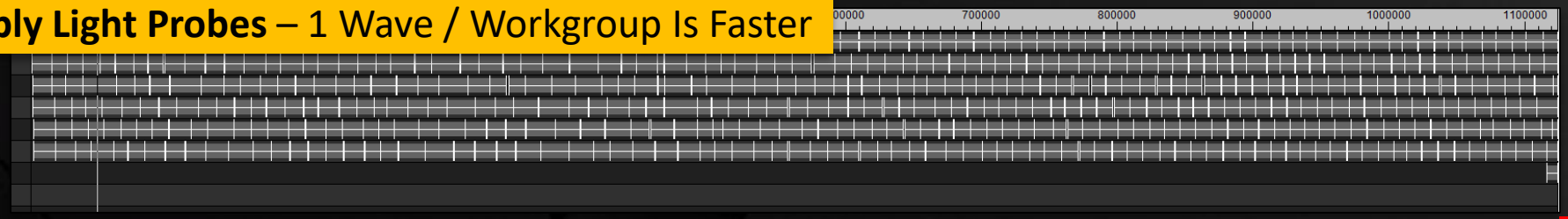
- **But large workgroups don't always yield these kinds of gains**
  - Especially with async-CS work
  - Diving deeper into the challenges in the next slide . . .

DYNAMICS GCN WORKGROUP LAUNCH

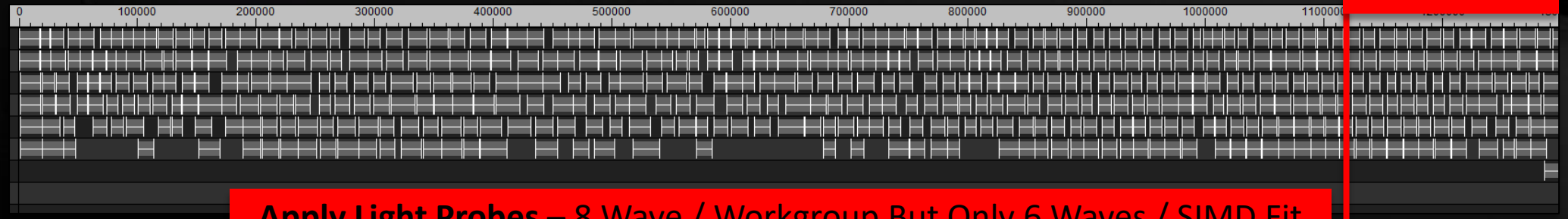
TWO COMPETING FACTORS – GETTING GOOD CACHE LOCALITY VS KEEPING MACHINE FILLED OR NOT

- Workgroups launched one per V$ (one per CU), round robin across CUs which can dispatch work
  - Thus two primary options for V$ cache locality: larger workgroups or **more work in a wave**
- For each SE, a wave can launch every 4 clocks, but waves can exit much faster than that
  - Not always possible re-fill the machine fast
- Workgroups need a contiguous block of LDS and waves need contiguous block of VGPRs/SGPRs to launch
  - Smaller workgroups = easier to launch, irregularity in {size, wave run-time, etc} can yield run-time gaps

Apply Light Probes – 1 Wave / Workgroup Is Faster



Extra Runtime



Apply Light Probes – 8 Wave / Workgroup But Only 6 Waves / SIMD Fit

Semi-Persistent Waves

Keeping GPU Filled With Good Data Locality

SEMI-PERSISTENT WAVE GOALS

ANOTHER INTERESTING OPTIMIZATION TOOL

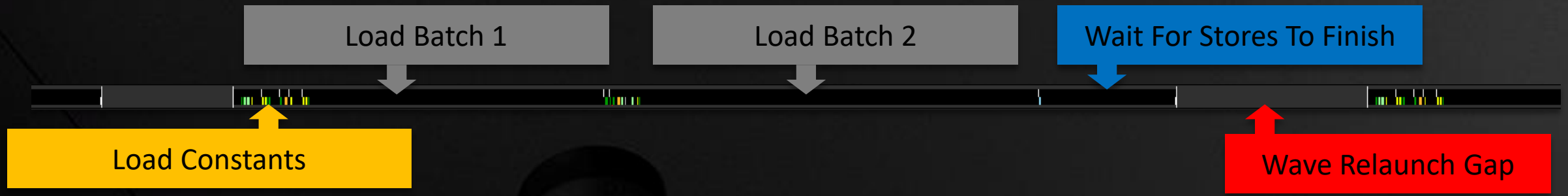
AMD

- Often using larger CS workgroups results in launch issues (especially with async compute)
- **Would like another way to gain memory locality and keep with single wave workgroup**
  - Will use wave-sized {64,1,1} to 8x8 remap for good 2x2 quads from prior section
- Current PC graphics APIs lack a function to dispatch to fill machine only once
- So fully persistent workgroups is not yet practical
- However semi-persistent waves can be an interesting middle ground

ANATOMY OF A SHORT SHADER AT SEMI-STEADY-STATE RUN-TIME

VIEW FROM INSTRUCTION TRACE

- Wave relaunch
  - Breaks ability to factor work (for instance would like to reuse instead of reload constants)
  - Breaks ability to hide latency at end of shader
  - Relaunch can be a significant amount of time where wave resources are not used

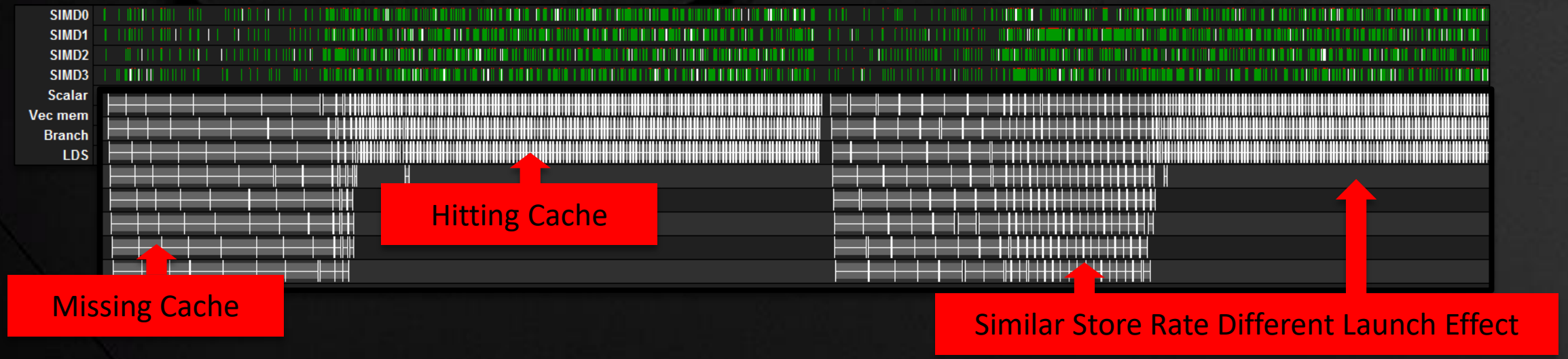


- Semi-persistent wave – can instead loop through N instances of the shader in one wave . . .
  - Amortize overheads of wave start/end/relaunch

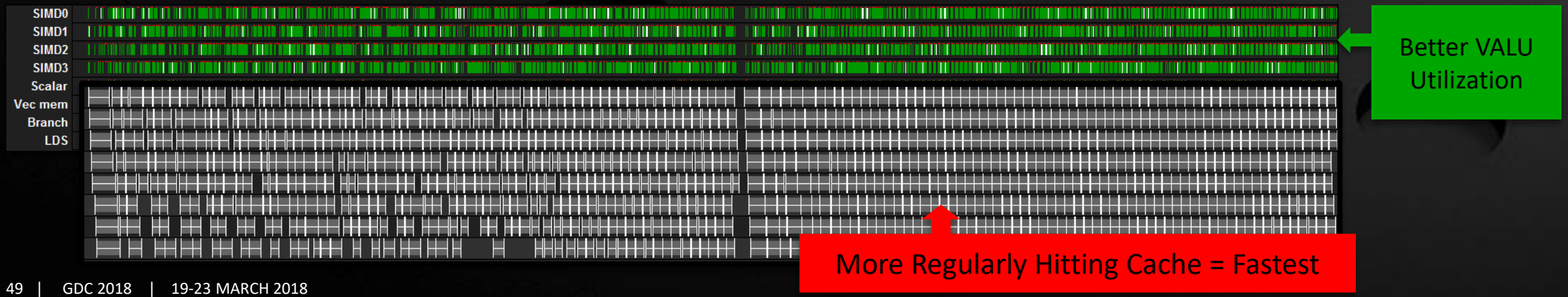
NOT LEAVING DATA LOCALITY AND LAUNCH REGULARITY UP TO CHANCE

WANT TO AVOID DEPENDING ON ACCIDENTAL DATA LOCALITY DUE TO EFFECT OF HOW GPU LAUNCHES WORKGROUPS

- {8,8,1} workgroup, horizontal then vertical kernel, later execution happens to hit more in cache



- {64,1,1} remap to 8x8, same kernels same occupancy slightly different execution, better cache rate

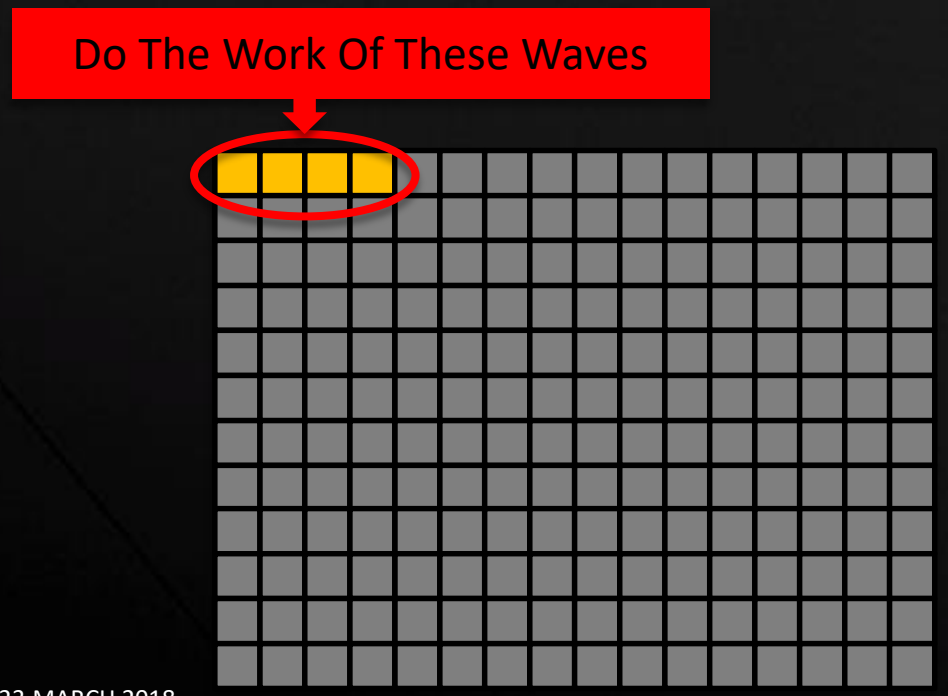


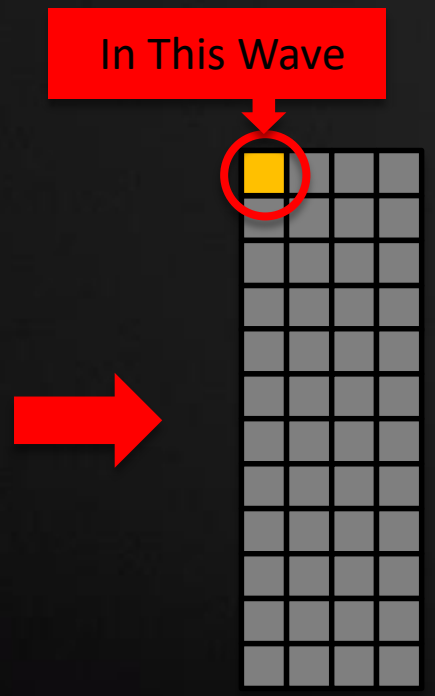
SEMI-PERSISTENT WAVE LAUNCH

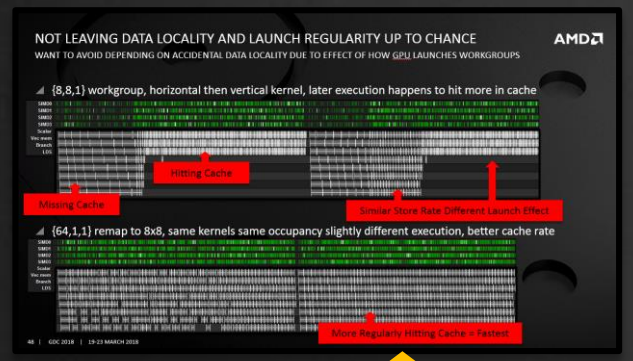
USING WAVE-SIZED WORKGROUP TO BEST KEEP MACHINE FILLED

- Convert multi-wave workgroup into single wave-sized workgroup via loop unrolling
  - **Group dimension setup to maximize data locality (ie use vertical grouping for vertical blur)**
  - Reduce dispatch size

- Example for a horizontal blur shader







Less Random
Data Locality

Explicit Data Locality
By Unrolling

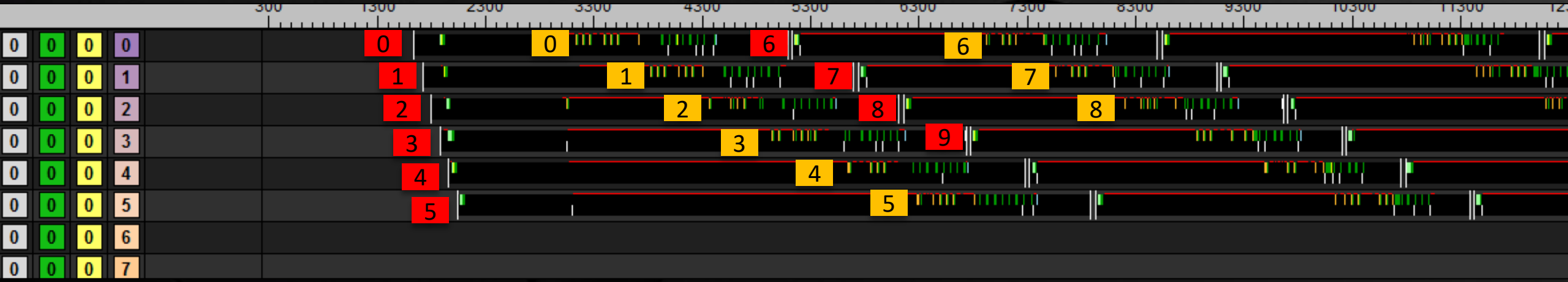
UNDERSTANDING WAVE EXECUTION AND SCHEDULING

VIA INSTRUCTION TRACE OF SIMPLE SHADER AT LAUNCH (LOOKING AT ONE SIMD)

- Showing effects of oldest wave first scheduling priority

0 0

- Highlighting **wave launch ordering** and **when the majority of texture fetches happen**



- Oldest wave first scheduling priority attempts to maximize opportunity for a wave to get cache reuse
  - **Good pairing for the unrolling for semi-persistent workgroups**

SIMPLE HORIZONTAL BLUR EXAMPLE

TESTING BASIC THEORY ON AMORTIZING WORK BY UNROLLING

AMD

- 8x8 tile doing a simple 9 tap box blur + 1 store

- Semi-persistent waves unrolling **share filter tap** – compiler optimized this avoiding refetching

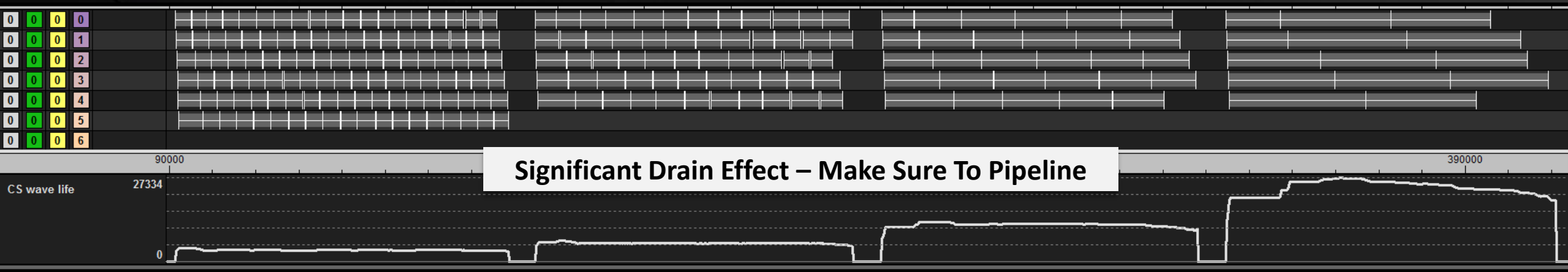
- Can estimate initial scaling by the number of VMEM ops (ignoring possible cache hit rate changes)
  - 1.00 ops : 8x8 unroll 8 times = 10 VMEM ops + 9 VMEM ops * 7 times = 73 VMEM ops / 8 tiles = 9.125 ops/tile
  - 1.01 ops : 8x8 unroll 4 times = 10 VMEM ops + 9 VMEM ops * 3 times = 37 VMEM ops / 4 tiles = 9.250 ops/tile
  - 1.04 ops : 8x8 unroll 2 times = 10 VMEM ops + 9 VMEM ops * 1 times = 19 VMEM ops / 2 tiles = 9.500 ops/tile
  - 1.10 ops : 8x8 no unroll . . . . = 10 VMEM ops . . . . . = 10 VMEM ops / 1 tiles = 10.000 ops/tile

SIMPLE HORIZONTAL BLUR EXAMPLE – MEASURED RUNTIME

NORMALIZED RX 580 MEASURED PIPELINED TIMING ALONG WITH PREDICTED SCALING BASED ON VMEM OPS

- **Semi-persistent is doing better than expected based on number of VMEM ops alone**
  - Suggests successful increase in cache hit rate

- 1.00 time : 1.00 ops : Semi-persistent 8x8 running 64x8 (aka unroll 8x)
- 1.04 time : 1.01 ops : Semi-persistent 8x8 running 32x8 (aka unroll 4x)
- 1.10 time : 1.04 ops : Semi-persistent 8x8 running 16x8 (aka unroll 2x)
- 1.22 time : 1.10 ops : Classic 8x8 linear



SIMPLE HORIZONTAL BLUR EXAMPLE – WAVE LIMITED RESULTS RX 580

RUNNING SOME CACHE/WAVE OPTIONS TO FIND OPTIMAL CONFIGURATIONS

AMD

- Semi-persistent wave example
  - Faster for either non-pipelined
  - Or optimal pipelined execution

CONFIG	WAVE/ SIMD	CLOCKS NON-PIPELINED	CLOCKS PIPELINED
8x8 unroll x8	3	74184	68483
8x8 unroll x4	3	72092	69520
8x8 unroll x2	3	73905	72272
8x8 linear	3	83508	82850
8x8 unroll x8	4	74241	66600
8x8 unroll x4	**4**	**71811**	**68222**
8x8 unroll x2	4	73376	71525
8x8 linear	4	82583	81491
8x8 unroll x8	**5**	**75571**	**64628**
8x8 unroll x4	5	73129	67426
8x8 unroll x2	5	73384	70679
8x8 linear	5	82427	80571
8x8 linear	**6**	**80971**	**78527**

Fastest For Non-Pipelined

Fastest For Pipelined Execution

Unoptimized

WORKGROUP OPTIMIZATION + SEMI-PERSISTENT WAVES – TAKEAWAY

AMD

- Two useful tools for optimization
- **Can provide gains even in fully memory bound situations!**
- Un-pinning the work-item to lane mapping via semi-persistent waves has advantages beyond what we have time to dive into today . . .

The End

Coding [koh-ding]

n. the act of transmuting caffeine into sarcasm with binary side effects

XOR RAX,RAX; XOR [RAX],RAX;

AMD

- Post-talk follow-up: timothy.lottes@amd.com
- **Want to join this team: AMD's Graphics Performance R&D Team is hiring**

- Special thanks to
  - Guennadi along with the AMD Tools Team – For giving us eyes to see with
  - AMD Vulkan Driver Team – For helping bypass the rev limiter
  - Jordan Logan – For extra help with run-time testing
  - Dustin Land – For proving it is not always hard to port to Vulkan
  - Sebastian Aaltonen and many others – For realizing cool stuff on the GPU
  - . . .

DISCLAIMER & ATTRIBUTION

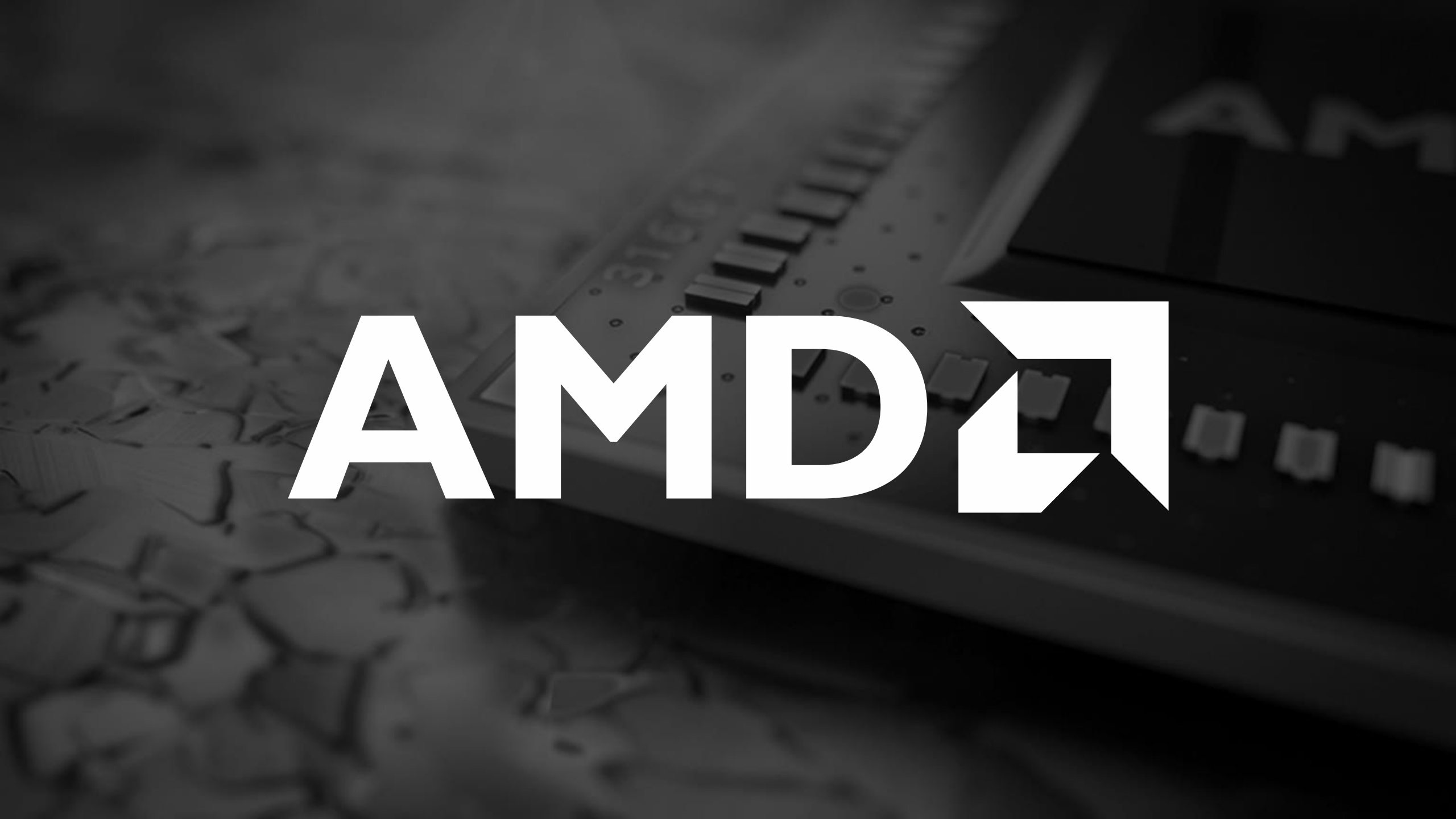
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. 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.

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 MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY 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.

ATTRIBUTION


AMD