TOM CLANCY'S RAINBOW SIX | SIEGE

OPTIMIZING SERVERS FOR THE CLOUD

JALAL EL MANSOURI TECHNICAL ARCHITECT

About Me

- Main contributions on R6 dev done on the architectural side of rendering:
  - GPU driven draw call submission
  - Checkerboard rendering
  - http://www.gdcvault.com/play/1022990/Rendering-Rainbow-Six-Siege
- I'm a systems programmer/architect
- First live game is R6
  - Lots of learnings!

Agenda

- A Year Of Rainbow
- The Cloud Platform
- Siege On The Cloud

01/ A YEAR OF RAINBOW

YEAR ONE ACTIVE PLAYERS

Not Just New Content

- Tech is a focus on R6 post launch
  - Graphics engine refactoring
  - Servers optimizations
  - Pipeline improvements
  - Anti-Cheat improvements
- Tech debt weights in on our tech direction
  - Moving to micro-services based architecture
  - Phasing out P2P systems
- Time to market is really interesting on live games
  - We use it to drive our development

Siege Networking

- Mainly a Client — Server networking model
  - All gameplay is Client - Server
  - In Coop (Terrohunt) a player would be hosting the game
  - Otherwise the game is hosted on dedicated servers
- Some bits are P2P:
  - Party session fully meshed
  - Team session fully meshed

Legacy Reasons

- Ubisoft shared network library built for Peer-2-Peer
  - Implements routing when a link fails between two peers

- Establishing $\frac{n(n-1)}{2}$ connections is still problematic in todays internet.

- Nat traversal, routing, adds additional complexity to the code

-> We are removing all P2P dependencies from our code base to reduce code complexity

Everything On Dedicated

- Man-months on more solid P2P vs man-months on optimizations:
  - Optimizing code vs fighting against the internet nature.
- Better control over the quality of our services
- Streamlining of tech for the benefit of our players
- Security!!!

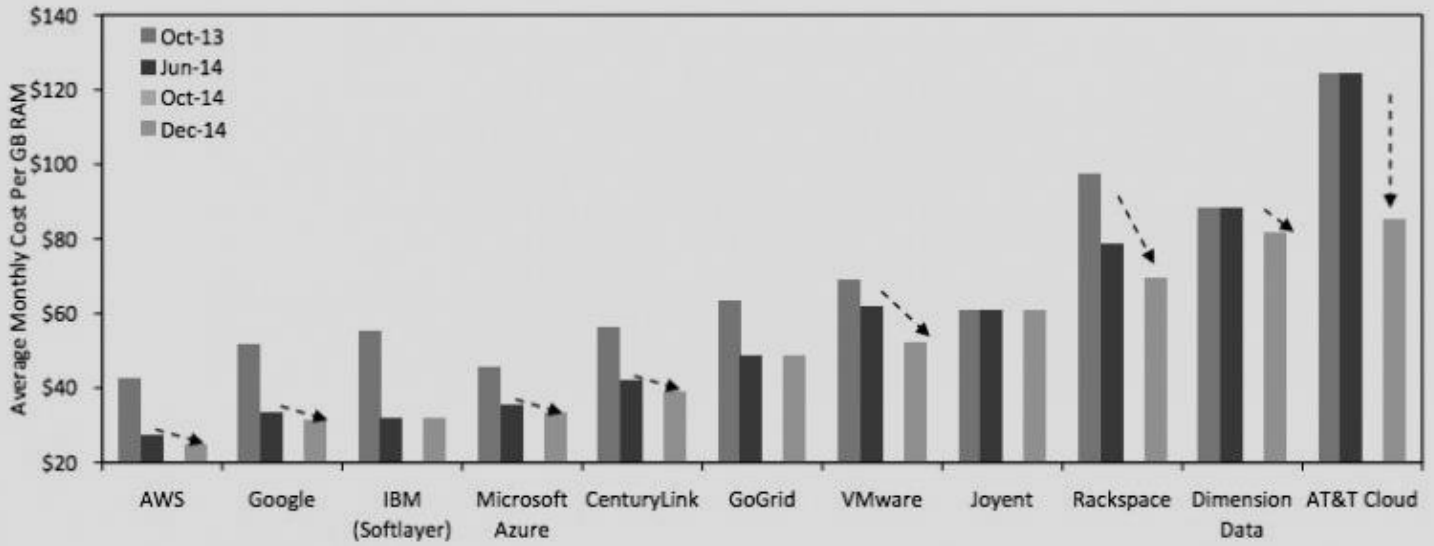
Moving out of Peer-2-Peer

- How about bandwidth costs?
  - Already routing Voice Chat through the game server
  - And a player spends most of their time on a game server currently
- Bandwidth cost is still marginal compared to compute cost

Dedicated On the Cloud

- Cost scales closer to your usage
- Compute and bandwidth cost only getting down in the future

Exhibit 14: Average Monthly Cost / GB RAM across various RBC Use Cases (excluding support costs)



(excluding optional support costs)

Average Monthly Cost Per GB RAM across various RBC Use Cases

	Oct-13	Jun-14	Dec-14	Change
AWS	$42	$27	$25	-8%
Google	$52	$34	$32	-6%
IBM (Softlayer)	$55	$32	$32	0%
Microsoft Azure	$46	$35	$34	-5%

(including high-touch support)

Average Monthly Cost Per GB RAM across various RBC Use Cases

	Oct-13	Jun-14	Dec-14	Change
AWS	$53	$38	$35	-6%
Google	$93	$69	$68	-2%
IBM (Softlayer)	$56	$32	$32	0%
Microsoft Azure	$70	$60	$58	-3%

Last Year Headlines

Microsoft Azure: Price cuts for virtual machines and storage services

...g price war continues.

Microsoft drops Azure costs by up to 17% in wake of fresh

RELATED STO

VB NEWS EVENTS RESEARCH Search

Caroline Donnelly
Datacentre Editor
15 Jan 2016 12:15

Micro
AWS
price

CLOUD

Microsoft cuts Azure VM prices by up to 50%

JORDAN NOVET @JORDANNOVET OCTOBER 3, 2016 9:56 AM

Cloud As A Lever

- The nature of it gives us more possibilities
  - Memory/compute thread offs by changing machine types

- Plan for the long term
  - Continue optimizing to use less resources
  - Even more important if your game is successful

=> Games are pushing boundaries on synchronous low latency computing

02/ THE CLOUD PLATFORM

Cloud as a Platform

- We consider the cloud a target platform that we optimize for

- The platform consists of the following components
  - Host machine running a host OS powering a hypervisor consisting of
    - Physical cores
    - Logical cores in case of Hyper-Threading

    -> Knowing how the hypervisor operates is key

  - Guest virtual machine running the Guest OS
    - Virtual core that maps to logical cores (n to 1 mapping)

VM sizes and Types

- There are multiple standard sizes to choose from on all cloud providers
  - They mostly differ on how the virtual machine is partitioned over the hardware
    - Number of virtual cores, Size of RAM, Disk size, number of IOPS
    - Exclusivity over virtual core mapping for performance consistency
- Bigger machines with multiple tenants preferred for game servers with small players in a session

Single vs Multiple Tenants

- Single tenant:
  - 1 session = 1 VM
  - Better isolation
  - Simpler machine management (and cost management)
- Multiple tenants:
  - N sessions = 1 VM
  - OS cost amortized
  - Better control over session partitioning over virtual hardware

Evolving Hardware

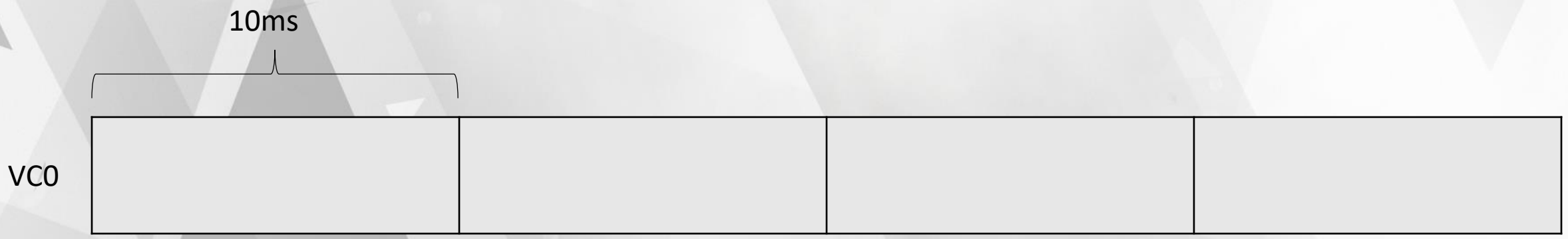
- Assume that you are not targeting fixed hardware
- Gather data on performance and hardware for your game server
- On the Azure A Series for example there is no official mention of underlying hardware
  - You can land on different clusters with different specs
  - Microsoft guarantees performance consistency between the different specs*

Hyper-V

- Hyper-V is the hypervisor used on Azure to run our virtual machines
- Think of it as a scheduler
- Has a base timeslice to schedule the virtual cores over the logical cores

Virtual Core Time Slicing

- A virtual core will get CPU time each 10ms

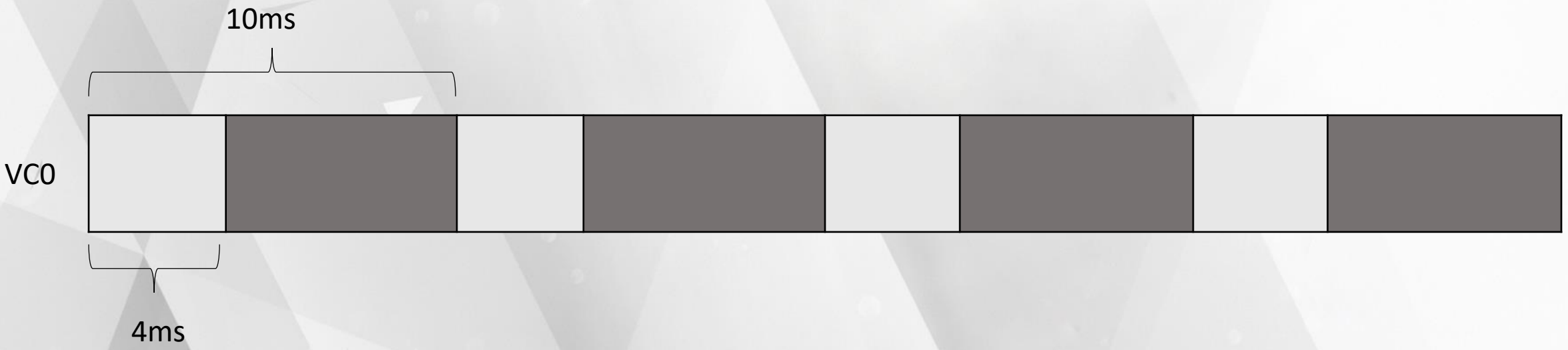


Virtual Core Time Slicing

- On older hardware generations you get 10ms out of 10ms
  - Thus enjoying a full logical core

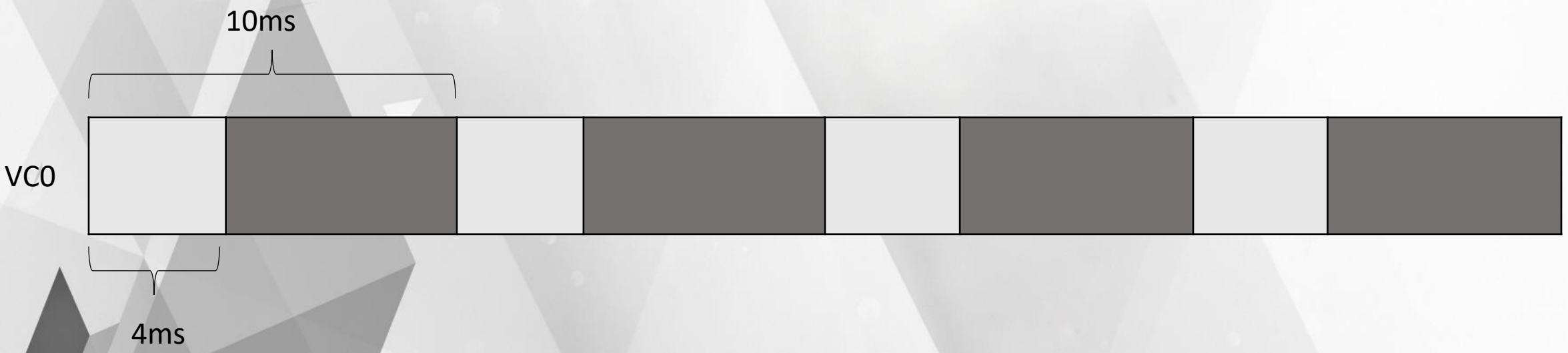
Virtual Core Time Slicing

- On new hardware generation you get a chunk of the 10ms
  - We most of the time observe 4ms our of 10ms



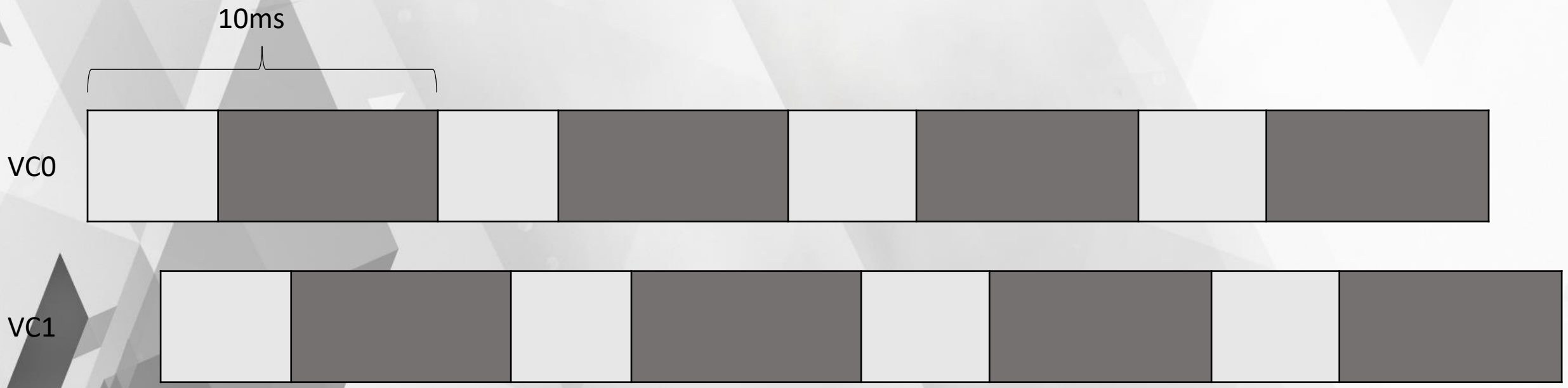
Virtual Core Time Slicing

- The dark area is occupied by another virtual core either belonging to the same VM or another VM



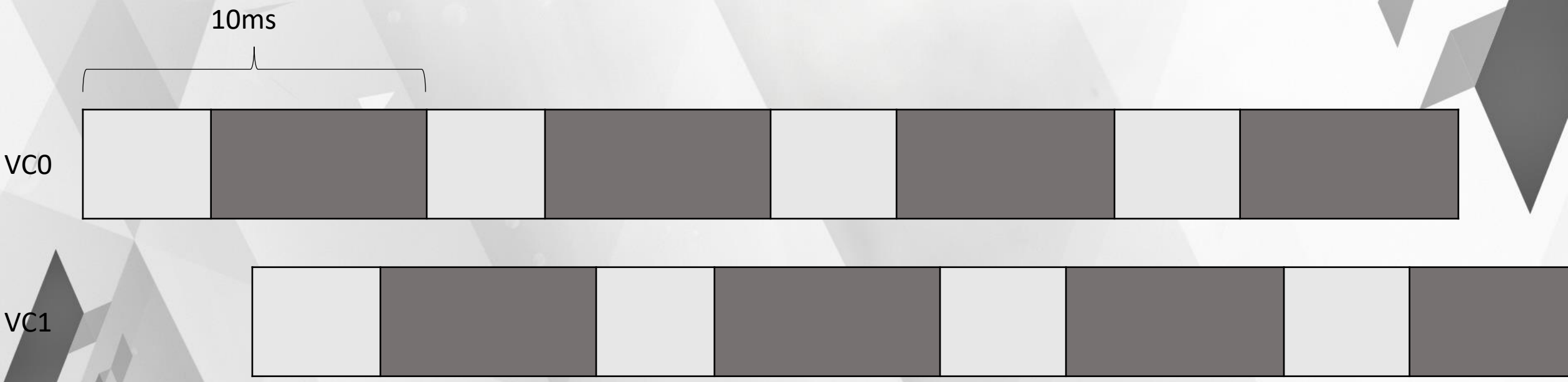
Virtual Core Time Slicing

- Each core works on its own timeline
  - Beware when heavy synching between threads



Virtual Core Time Slicing

- Virtual cores will also not overlap when they are mapped to the same logical core



Virtual Core Time Slicing

- What could go wrong?

Virtual Core Time Slicing

- Out of sync virtual cores will cause synchronisation issues when dealing with shared resources
- Main example is locks

Virtual Core Time Slicing

- Shared lock
  - Green is an an acquired lock
  - Orange is waiting for an ownership transfer

Virtual Core Time Slicing

- Shared spinlock
  - Green is an an acquired lock
  - Orange is waiting to acquire the lock

Single Virtual Core Machine

- Engines now assume to be running on multiple core
  - PCs minimum specs are 2 cores minimum
  - Consoles have 7—ish cores
- We usually abuse spinning synchronisation primitives
  - Scheduling on consoles more consistent
  - Spinlock on one core is a waste of a full quantum
- Based on the same scheduler and code base you need to target this unusual case

Operating System Time

- Operating system will need each share of the CPU too
- If you starve the OS its threads priority will be boosted
- On a single virtual core machine, the recommendation leaving 50% CPU time to the OS

03/ SIEGE ON THE CLOUD

Rainbow Six Cloud Platform

- R6 is running on Azure on the following environment
  - Different generation hardware powered by the Hyper-V hypervisor
  - Standard Tier A1 with Windows Server 2012 R2 VM
    - 1 Virtual Core
    - 1.75 GiB Of RAM
    - 70 GiB HDD
    - 2x500 IOPS

Headless Engine

- Headless engine is a simplified version of a client with the following components stripped out:
  - Graphics
  - Sound
  - Inputs system
  - UI
  - All FX scripts
  - Animation code that doesn't modify hit boxes
- Additional validation/anti-cheat code is running
- Physics and animation are the most CPU intensive tasks on the server currently

Headless Engine

- Stub low level APIs, code that should never be reached should be asserted
  - Graphics & input & sound good candidate for this
- Operate on the scheduler level
  - FX tasks not pushed in the first place
- The cleaner the dependencies on your engine the easier is to strip out component
  - Still >2500 ifdeffed chuncks on R6 code, was >3000 a year ago

Scheduler Logic

- N worker threads
- Workers subscribe to Queues where pending jobs are pushed
  - Order of subscription defines priority
  - Exemple queues: GraphicsQueue, PhysicsQueue, ...
- System designed to be lightweight, low cost very in single core scenarios
  - Already pushing >1000 of jobs in the client per frame

Server Scheduling

- R6 used to runs on A2
- A2 and A1 share actually the same scheduling logic
- We use two worker threads
  - First is used for frame blocking queues with higher priority, then async queues
  - Second worker used exclusively for async queues
- Before optimization push for A1, second worker was empty most of the time during gameplay
  - Ideal to give OS enough CPU time

Spinning Logic

- Any spin is a waste of the rest of a quantum on a 1 core machine
- Windows mutexes don't spin in a one core machine
- Engine's system library implements
  - Adaptive locks (spin first then use a system lock)
  - Spin locks

  =>Typedeffed to no spin in all cases

Spinning Logic

- Some code is not using the system defined spin locks but still spins
  - Hard to go through in a 10 year old codebase
- Any leaf profiling scope that takes more than half a quantum is logged on a special version
  - Excluding context switches from the computed time
  - QC would go through a build and give me the log file
- Quantum on a desktop windows is bigger (95ms) than on a server (16ms)

Profiling On The Dev Machine

- Start your windows machine with one core
  - Don't do that ☺
- If using windows 10 spawn a local VM using Hyper-V
  - Ideal to get a reliable environment
  - Want to experiment with docker for windows to speed up tests
- Setting the process affinity to one core
  - Have a commandline to do so
  - Works out well to bring out spin lock issues

Profiling On The Dev Machine

- The VM is going to be sharing a physical core and a logical core sometimes
  - Simulated through launching a second process on the paired logical core

```
constexpr int kArrayLength = 0x2000000;  // 32 MB (must be power of 2)
constexpr int kCacheLineSize = 64;
constexpr int kStep = kCacheLineSize / sizeof(int);
int main(int argc, char* argv[]) {
   SetProcessAffinityMask(GetCurrentProcess(), 1 << 7); // GS is running on SetProcessAffinityMask(GetCurrentProcess(), 1 << 6)
   int* mem = new int[kArrayLength];
   float fpAccu = 0.f;
   __m128 sseAccu = { 0.f };
   while (1) {
      for (int ofs = 0; ofs < kArrayLength; ofs += kStep) {
         mem[ofs]++;
         fpAccu = FPCode(fpAccu);
         sseAccu = SSECode(fpAccu, sseAccu);
      }
```

Profiling On The Dev Machine

- The VM is going to be sharing a physical core and a logical core sometimes
  - Simulated through launching a second process on the paired logical core
  - Having same kind of code run after each engine loop to trash the cash
- Usually exhibits more inconsistencies in tick rates

Synchronizing With The Hypervisor

- Game code is running synched to a tick rate
  - 30 ticks per secs
  - Code takes 13ms to run fitting in 4 slices
  - Going take an effective 37ms

Synchronizing With The Hypervisor

- Ideally you would run at a multiple of the hypervisor resolution
- We need to wait for the hypervisor slice to begin
  - We don't have a primitive for that
- We implemented our own hypervisor suspension detection
  - Actually quite simple

Synchronizing With The Hypervisor

- Loop for 50ms if a step takes more than a ms return the time taken:

```
HFTimer testTimer;
HFTimer innerLoopTimer;
HFTimer outerLoopTimer;
testTimer.Start();
outerLoopTimer.Start();
do {
    innerLoopTimer.Start();
    if (outerLoopTimer.GetElapsedTime() > 1000) { // 1ms
        return outerLoopTimer.GetElapsedTime();
    }
    outerLoopTimer.Start();
    DoSomeStuffFunction();
    if (innerLoopTimer.GetElapsedTime() > 1000) { // 1ms
        return innerLoopTimer.GetElapsedTime();
    }
} while (testTimer.GetElapsedTime() < 50000); // 50ms
return 0;
```

Synchronizing With The Hypervisor

- Sleep before running this code to reset priority boosts
- Bump the priority of the thread running this code to time critical
- Run the code for each core
- Reset priority and sleep

Synchronizing With The Hypervisor

- Code returns at the beginning of a time slice for each virtual core
- We use it to set our waitable timers so they start at the beginning
- We adjust usable game time to give some CPU time to the OS

Tick Rate on Azure

- Getting % of time over 10ms time slice
  - Going above 100 ticks per sec unrealistic
- On multiple cores you lose on synchronicity
- On a single core you need to leave time to the OS
  - One slice for you, one for the OS

Tick Rate On Azure

- timeBeginPeriod needs to be set to 10ms/5ms/2ms/1ms
  - Default resolution is 15.66ms. So not compatible with the hypervisor sync
- We use the OS waitable timers
  - One for each supported sync period (10ms/20ms/30ms/50ms)
- We run by default with the 20ms timer
  - 50 Ticks per second

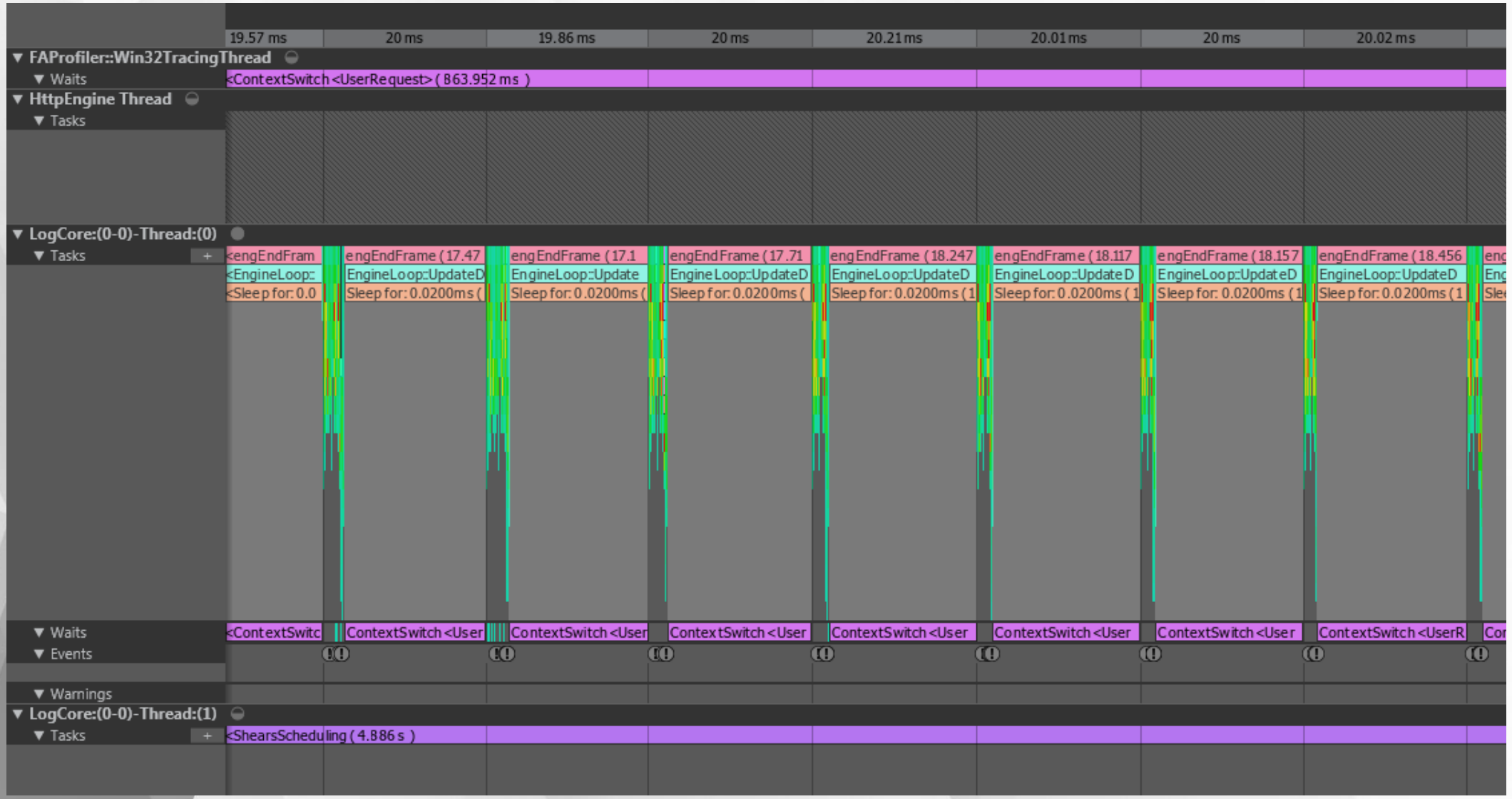
Tick Rate On Azure

- Tick rate is adaptative
  - Between gameplay phases we take more CPU time
- If OS we start starving the OS we move to the next sync period
  - 33 ticks per seconds
- Using timeslice detection results we adapt the threshold for all hardware generations

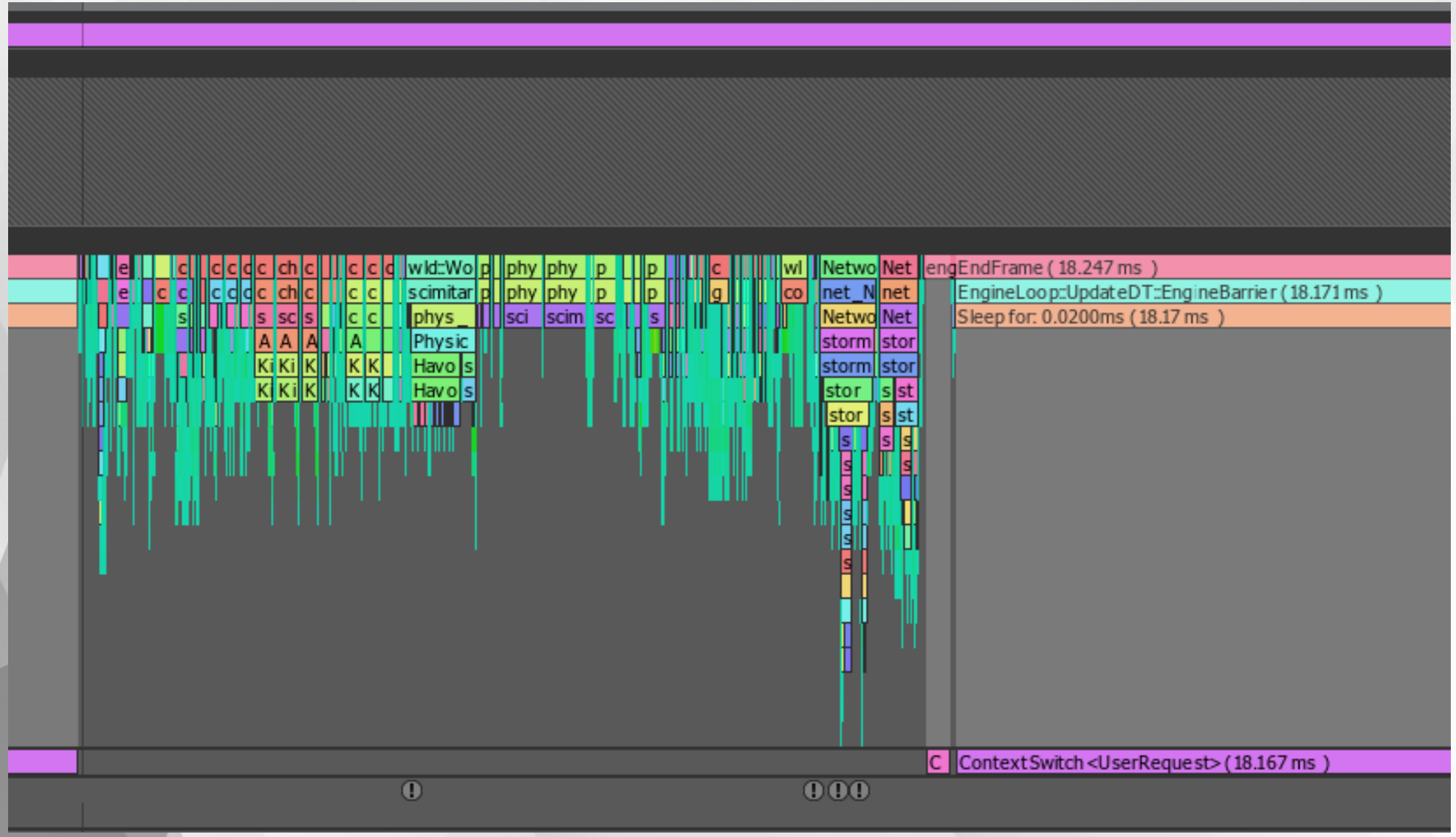
Don't Forget About Memory

- Our profiling shows context switch reasons:
  - We noticed page swaps during gameplay
- With 1.75 of RAM and the OS taking up to 40% we were swapping pages in/out quite frequently
- Keeping an eye on memory is quite important, most of our other targets have lower memory requirement

In Practice


19.57 ms
20 ms
19.86 ms
20 ms
20.21 ms
20.01 ms
20 ms
20.02 ms
FAProfiler::Win32TracingThread
Waits
<ContextSwitch <UserRequest> (863.952 ms)
HttpEngine Thread
Tasks
LogCore:(0-0)-Thread:(0)
Tasks
<engEndFram
engEndFrame (17.47
engEndFrame (17.1
engEndFrame (17.71
engEndFrame (18.247
engEndFrame (18.117
engEndFrame (18.157
engEndFrame (18.456
<EngineLoop::
EngineLoop::UpdateD
EngineLoop::Update
EngineLoop::UpdateD
EngineLoop::UpdateD
EngineLoop::UpdateD
EngineLoop::UpdateD
EngineLoop::UpdateD
<Sleep for: 0.0
Sleep for: 0.0200ms (
Sleep for: 0.0200ms (
Sleep for: 0.0200ms (
Sleep for: 0.0200ms (1
Sleep for: 0.0200ms (1
Sleep for: 0.0200ms (1
Sleep for: 0.0200ms (1
Waits
<ContextSwitc
ContextSwitch <User
ContextSwitch <User
ContextSwitch <User
ContextSwitch <User
ContextSwitch <User
ContextSwitch <User
ContextSwitch <UserR
Events
Warnings
LogCore:(0-0)-Thread:(1)
Tasks
<ShearsScheduling (4.886 s)

In Practice


engEndFrame (18.247 ms)
EngineLoop::UpdateDT::EngineBarrier (18.171 ms)
Sleep for: 0.0200ms (18.17 ms)
ContextSwitch <UserRequest> (18.167 ms)

In Practice

10ms time slice

10ms

~6ms suspend

Hardware Generations In Practice

- We are running on different generation of HW:
  - On hardware with 10ms slices (continuous time):
    - median tick time (including wait): 19.998
    - average tick time (including wait) : 20.044
    - median update time : 12.56
    - average update time : 12.86
  - On hardware with 4ms slices:
    - median tick time (including wait) : 19.999
    - average tick time (including wait) : 20.092
    - median update time (< timeslice) : 4.13
    - average update time (< timeslice) : 4.056
    - median update time (> timeslice) : 11.18
    - average update time (> timeslice) : 10.775

Hardware Generations In Practice

- Newer generations gives us more inconsistencies:
  - Worst case on hardware with 4ms slices:
    - median tick time (including wait) : 23.95
    - average tick time (including wait) : 24.62
  - Update can take up to 3x 4x slices where it's a rarity on HW with 10ms slices.

Wrap Up

- The Cloud opens up more possibilities
  - Cost is the limit
- Running synchronous code at high frequency needs work
  - Necessary investment

Special Thanks

- Andrew Farrier
- Catalin Arsenescu
- Jeff Preshing
- Lindsey Lachance
- Lorenzo Aurea
- Vincent Jouault
- Yasser Rihan

THANKS FOR LISTENING

References

- The Online Tech of Respawn’s Titanfall, Jon Shiring 2014
  - http://www.slothy.com/Rackspace-talk-slides.pdf
- Dedicated Servers in Gears of War 3, Michael Weilbacher
  - http://www.gdcvault.com/play/1015337/Dedicated-Servers-In-Gears-of
- Azure A-SERIES, D-SERIES and G-SERIES: Consistent Performances and Size Change Considerations
  - https://blogs.msdn.microsoft.com/igorpag/2014/11/11/azure-a-series-d-series-and-g-series-consistent-performances-and-size-change-considerations/
- Sizes for Cloud Services
  - https://azure.microsoft.com/en-us/documentation/articles/cloud-services-sizes-specs/
- Generational Performance Comparison: Microsoft Azure’s ASeries and D-Series
  - http://cloudspectator.com/wp-content/uploads/report/generational-performance-comparison-microsoft-azures-a-series-and-d-series.pdf