

# Programming for Hybrid: Untangling your Threads

Leigh Davies

Senior Game/Graphics Application Engineer

03/2023

The Intel logo is displayed in white lowercase letters on a dark blue square background. A smaller, lighter blue square is positioned at the bottom left corner of the dark blue square.

intel<sup>®</sup>

# INTRODUCTION

A top-down view of a person with blonde hair wearing large black headphones, sitting at a wooden desk. They are using a computer with a keyboard and mouse. The desk is cluttered with various items including a small black box with a white logo, a smartphone, and a microphone. The scene is dimly lit with a blue and purple color cast.

Introduction

OS Scheduling

Data Capture

Case Studies

Closing Thoughts

# What is a Hybrid SOC (System On a Chip)?

## Combines Performance Cores and Efficient Cores

- Two core types with different power and performance characteristics
- Both core types have the same ISA support
  - No AVX512, TSX
  - New AVX-VNNI, UMWAIT/TPAUSE

## Performance Cores



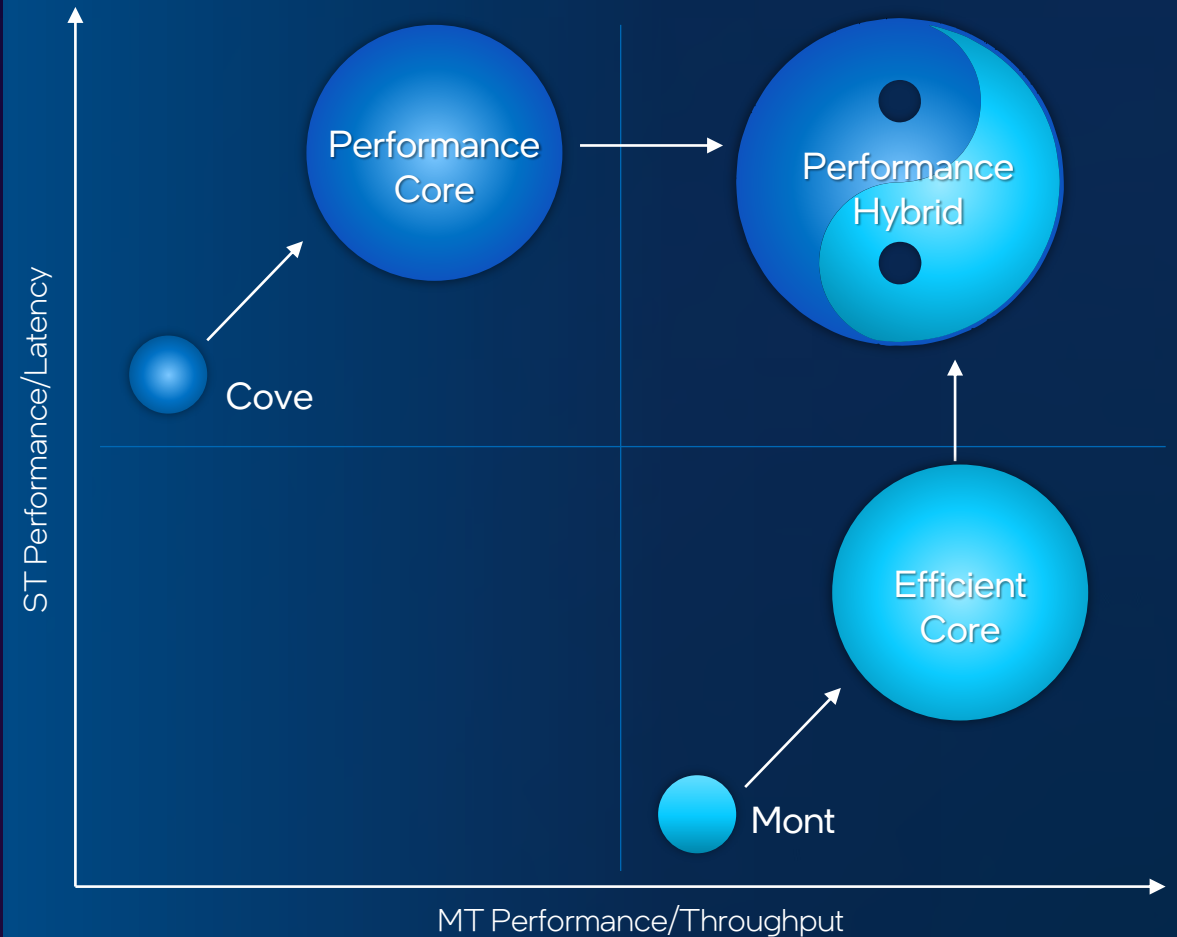
- Concentrate on single and limited threading scenarios
- Performance intensive

## Efficient Cores



- Concentrate on MT throughput and power limited scenarios
- Efficiency focused

## Intel Hybrid Architecture



# What is a Hybrid SOC (System On a Chip)?

## Combines Performance Cores and Efficient Cores

- Two core types with different power and performance characteristics
- Both core types have the same ISA support
  - No AVX512, TSX
  - New AVX-VNNI, UMWAIT/TPAUSE

### Performance Cores



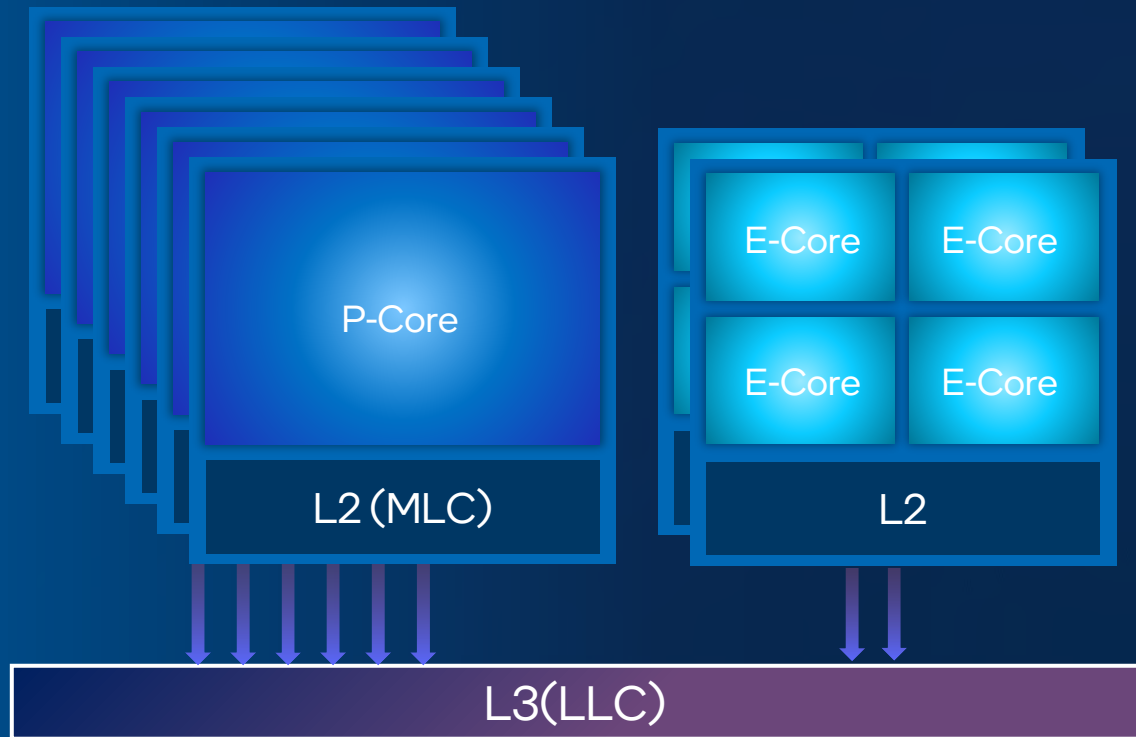
- Concentrate on single and limited threading scenarios
- Performance intensive

### Efficient Cores



- Concentrate on MT throughput and power limited scenarios
- Efficiency focused

## Intel Hybrid Architecture



P-Cores have their own L2 caches, and E-Cores share L2 caches in groups of four.



# Intel® Thread Director (HGS+)

## Hardware unit

Intelligence built directly into the core

## Monitors the runtime instruction mix

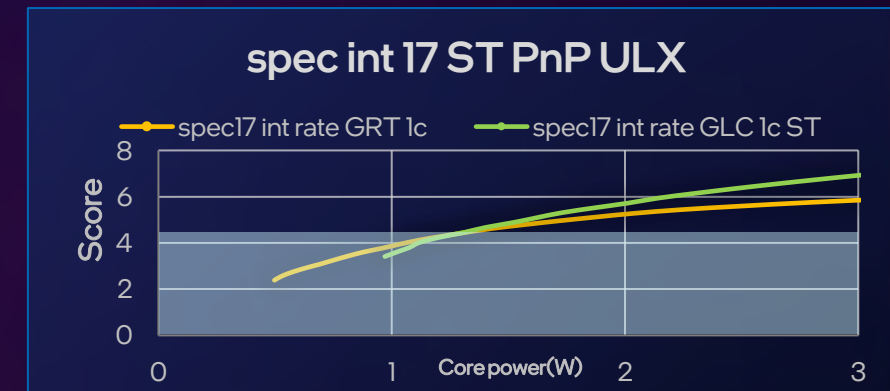
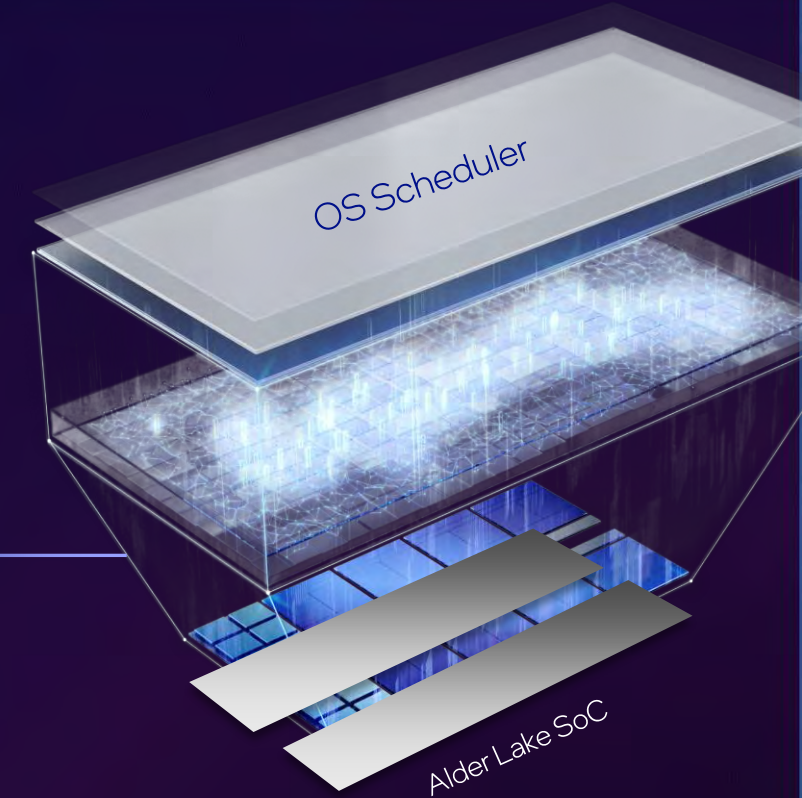
of each thread and as well as the state of each core – with nanosecond precision

## Provides runtime feedback to the OS

to make the optimal scheduling decision for any workload or workflow based on ISA and other inputs

## Dynamically adapts guidance

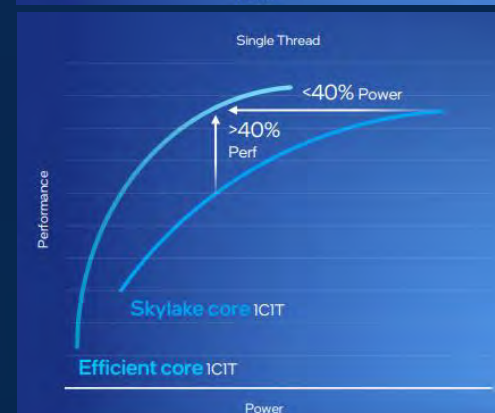
based on the thermal design point, operating conditions, and power settings – without any user input



# CPU Specs Progression

FEATURES	ROCKET LAKE (i9-11900k)	ALDER LAKE (i9-12900k)	RAPTOR LAKE (i9-13900k)
Lithography	✓ 14 nm	✓ Intel 7	✓ Intel 7
P-cores	✓ 8/16	✓ 8/16	✓ 8/16
Base Freq	✓ 3.5 GHz	✓ 2.4 GHz	✓ 3.0 GHz
Max Turbo Freq	✓ 5.3 GHz	✓ 5.2 GHz	✓ <b>5.8 GHz</b>
L1D\$/I\$	✓ 48 / 32 KB	✓ 48 / 32 KB	✓ 48 / 32 KB
L2 U\$	✓ <b>512 KB</b>	✓ <b>1280 KB</b>	✓ <b>2048 KB</b>
L3 U\$	✓ 16 MB	✓ 30 MB	✓ 36 MB
E-cores	✓ <b>0</b>	✓ <b>8</b>	✓ <b>16</b>
L2 U\$ (4x E-core)	✓ <b>N/A</b>	✓ <b>2048 KB</b>	✓ <b>4096 KB</b>
Max Mem Size	✓ 128 GB	✓ 128 GB	✓ 128 GB
Mem Type	✓ DDR4-3200	✓ DDR5-4800	✓ DDR5-5600
Mem B/w	✓ 50 GB/s	✓ 76.8 GB/s	✓ 89.6 GB/s

Efficient Cores are comparable to earlier generation Performance Cores



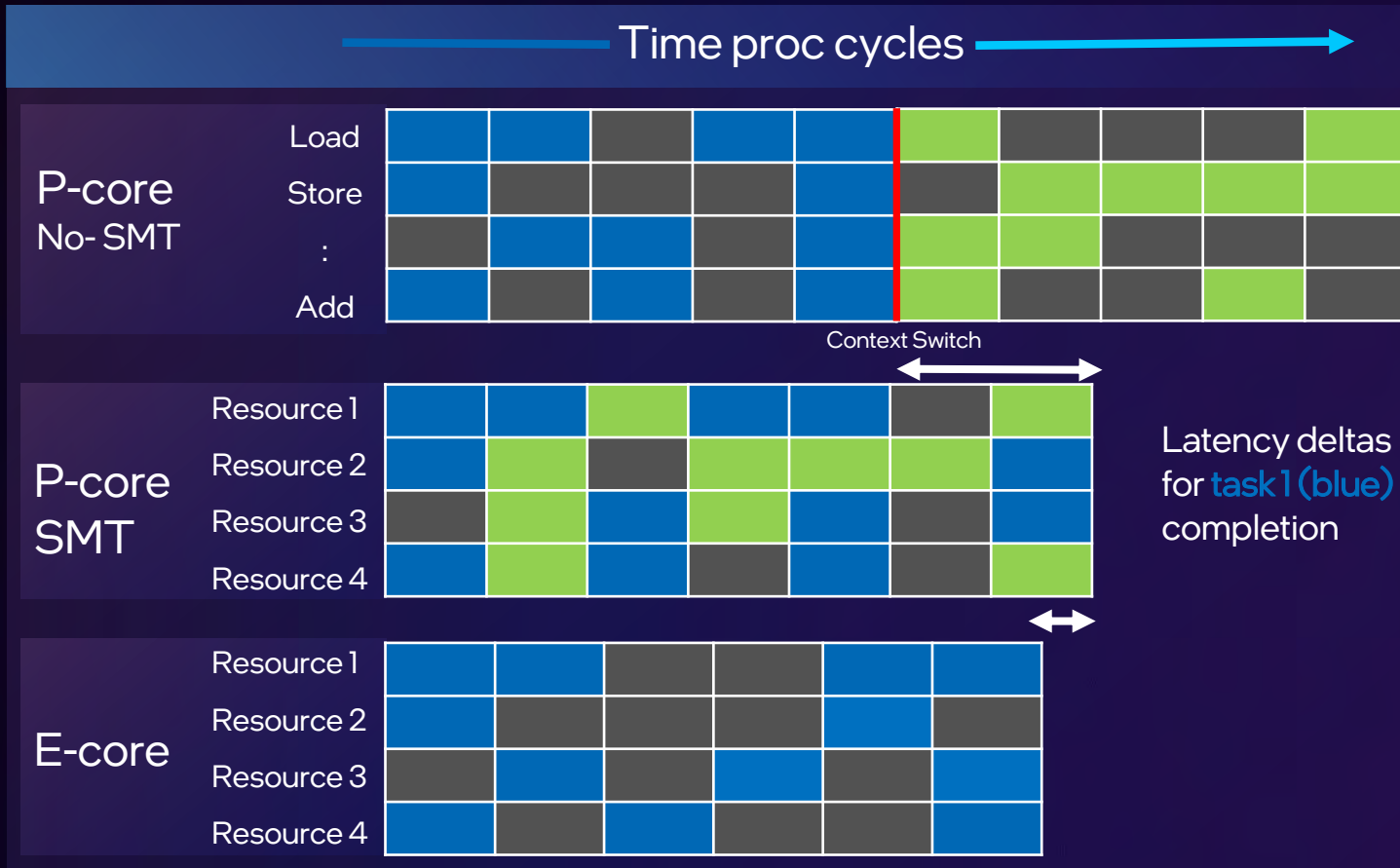
## Disclaimer

- SPECCrate2017\_int\_base estimates using an open source compiler, iso-binary For workloads and configurations visit [www.intel.com/ArchDay21claims](http://www.intel.com/ArchDay21claims). Results may vary.

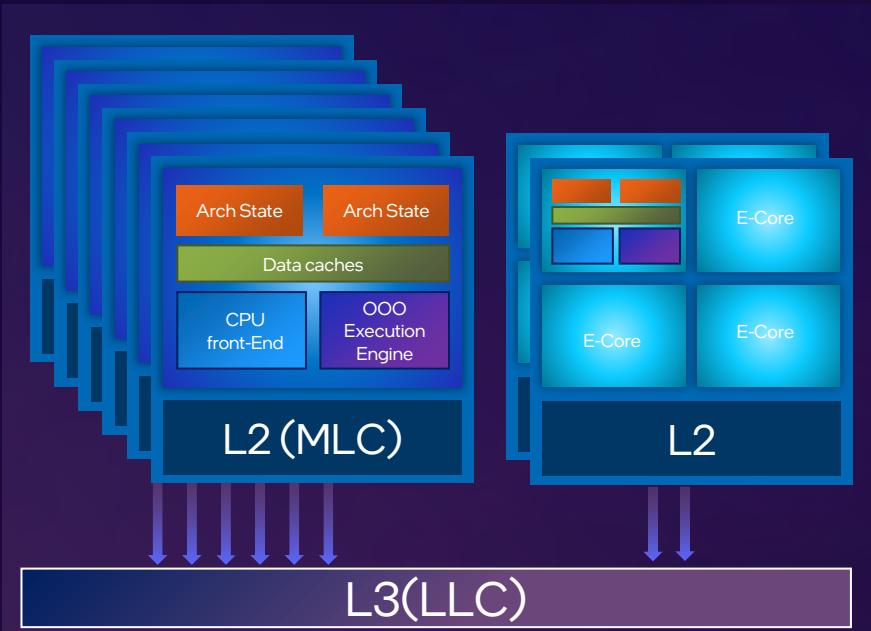
# Hyper-Threading Recap

Important for later in the talk.....

Hyper-Threading ( Simultaneous Multi-Threading)



Each box represents a processor execution unit



## SMT and instruction throughput

- Improves Core CPI (Clockticks Per Instruction)
- Potential degrades Thread CPI

“E-cores are designed to provide better performance than a logical P-core with both hardware sibling hyper-thread busy.”

# Changing Our Assumptions ...

## All cores have the same performance profile

- Significant performance delta between cores
- Same ISA != same throughput

## All cores have the same frequency

- There may be one, two, or more, faster cores
- The fastest core may move around the package

## Hyper Threading doubles the physical core count

- Hyperthreading may be available on only some cores in a package
- Logical core count may not equal 2x physical core count

## Optimizing the CPU only matters if CPU Bound

- Power may be shared between GPU/CPU/Other -> frequency impact



# CPU Topology

All cores exposed to OS as individual Logical Processors using;

## Preferred Enumeration method: GetLogicalProcessorInformationEx()

- struct \_PROCESSOR\_RELATIONSHIP:
- Field: EfficiencyClass; << Higher mean more perf
- Note: This is relative to other logical processors in the system.
- For 12<sup>th</sup>/13<sup>th</sup> Gen Intel Core EfficiencyClass=1 is P-Cores.
  
- struct \_CACHE\_RELATIONSHIP:
- Field: Level << The cache level
- Field: Type << The cache Type (Data, instruction, etc)
- Field: GroupMask.Mask << LP's connected to the cache
  
- Note: Even cores with the same EfficiencyClass can have different cache configurations.

## Pseudo Code

```
typedef pSLPI_EX SYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX*;
uintptr_t affinity;

if (GetLogicalProcessorInformationEx(RelationAll, (pSLPI_EX)&buffer[0], &size))
{
    for (size_t i = 0; i < size;)
    {
        SYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX* proclnfos = (pSLPI_EX) &buffer[i];

        switch (proclnfos->Relationship)
        {
            case RelationProcessorCore:
            {
                for (uint32_t g = 0; g < proclnfos->Processor.GroupCount; ++g)
                {
                    LPNumber = BitScan(proclnfos->Processor.GroupMask[g].Mask);
                    LPClass = proclnfos->Processor.EfficiencyClass;

                }
                break;

            case RelationCache:
            {
                cache.m_ProcessorMask = proclnfos->Cache.GroupMask.Mask;

            }
            break;
        }
        i += proclnfos->Size;
    }
}
```

# OS SCHEDULING



Introduction

OS Scheduling

Data Capture

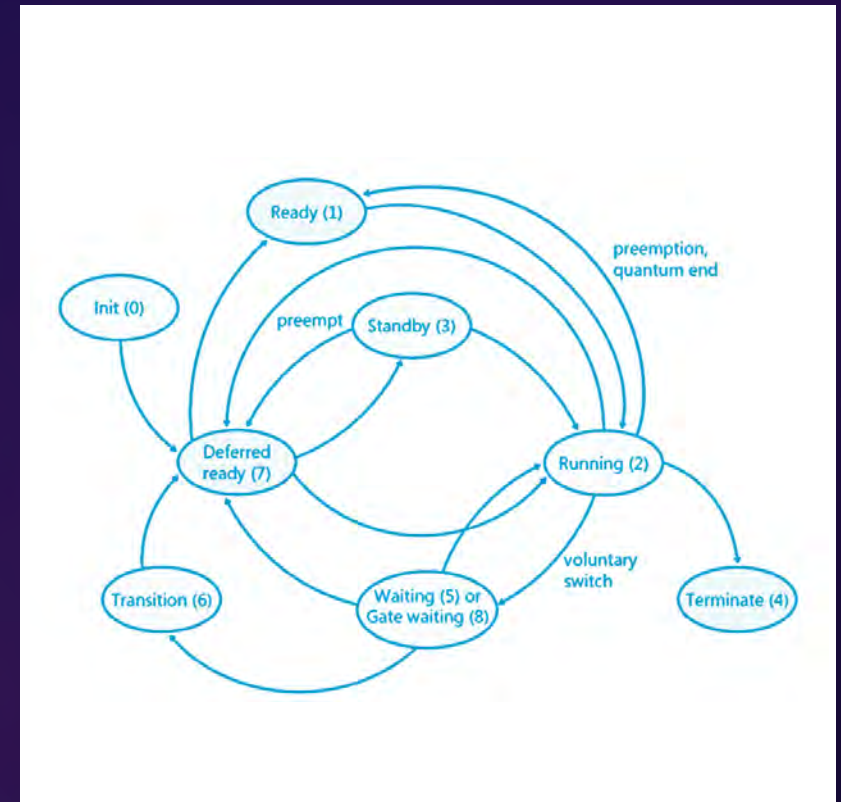
Case Studies

Closing Thoughts

# Thread Scheduling

## No central scheduler

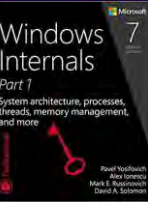
- Scheduling routines are called whenever events occur that change the state of a thread
- Example scheduling events include:
  - A thread becomes ready to execute (newly created or released from wait state)
  - A thread enters a wait state or ends
  - Interval timer interrupts
  - Other hardware interrupts (for I/O wait completion)
  - Quantum End
  - A thread priority is changed
  - Thread QoS changes
  - System concurrency/utilization changes (causing parking)
  - Intel® Thread Director updates



# Scheduling Priorities

- A process has only a single base priority value
- Each thread has two priority values: current (Dynamic) and base
- Scheduling decisions are made based on the current priority
  - The system under certain circumstances increases the priority of threads in the dynamic range (1 through 15) for brief periods

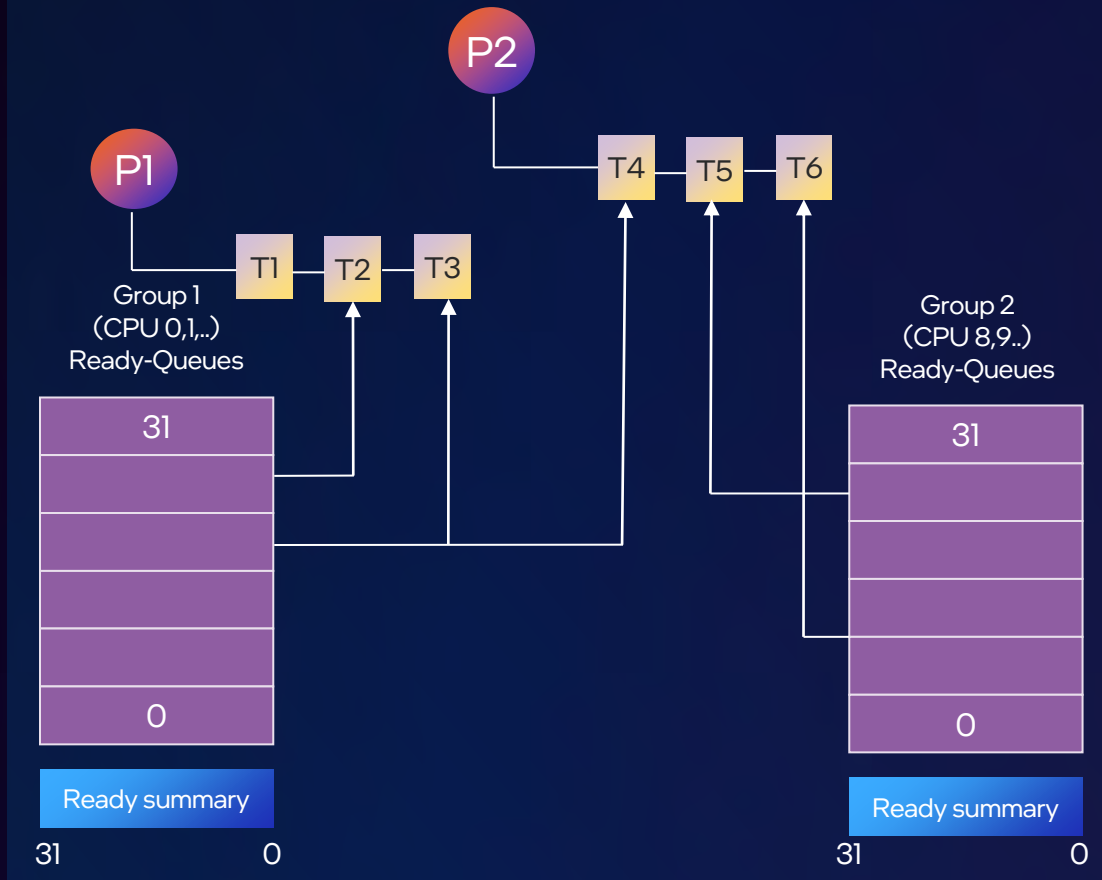
Relative Priorities	Priority Class Relative Priority					
	Real time	High	Above Normal	Normal	Below Normal	idle
THREAD_PRIORITY_TIME_CRITICAL	31	15	15	15	15	15
THREAD_PRIORITY_HIGHEST	26	15	12	10	8	6
THREAD_PRIORITY_ABOVE_NORMAL	25	14	11	9	7	5
THREAD_PRIORITY_NORMAL	24	13	10	8	6	4
THREAD_PRIORITY_BELOW_NORMAL	23	12	9	7	5	3
THREAD_PRIORITY_LOWEST	22	11	8	6	4	2
THREAD_PRIORITY_IDLE	16	1	1	1	1	1





# Thread Scheduling

To improve scalability Windows 8+ added Shared Ready Queues: Reduce contention on Ready Queue.



Current Hybrid systems have up to 8 LPs per group

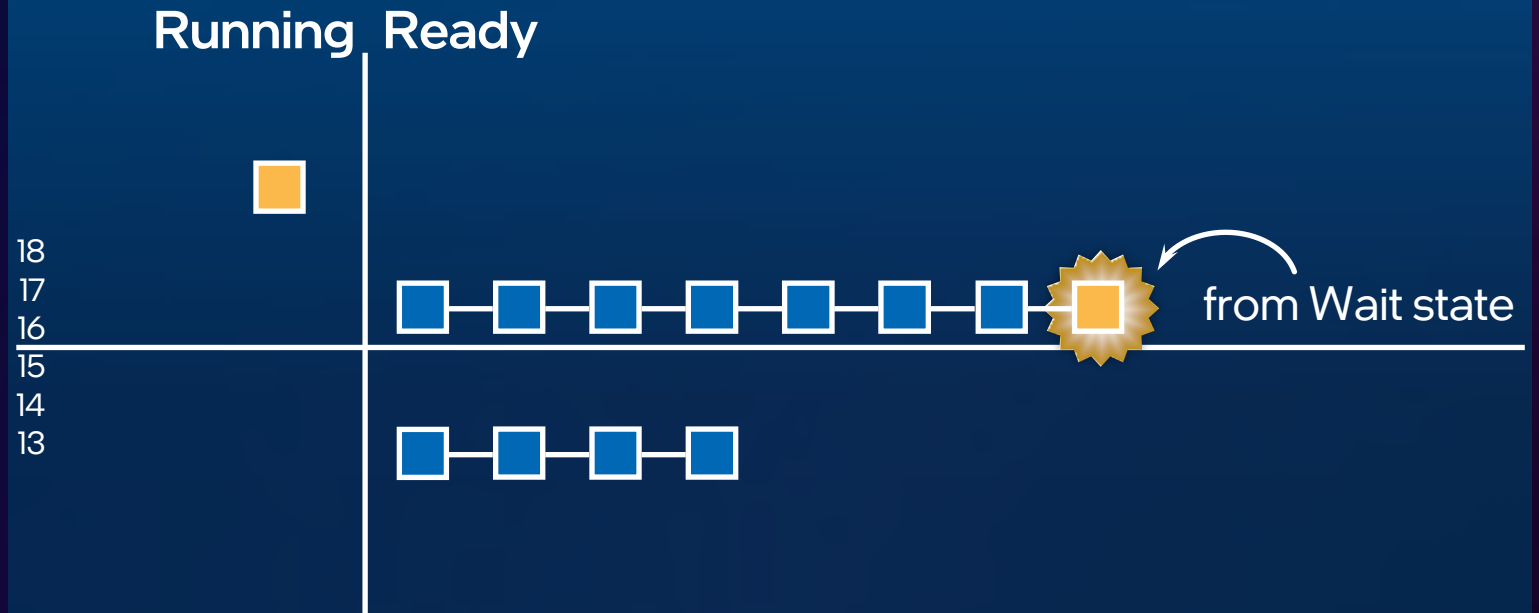
## Priority driven, preemptive

- Ready Queue consist of;
  - 32 queues (FIFO lists) of "ready" threads
  - UP: Highest priority thread always runs
  - MP: One of the highest priority runnable thread will be running somewhere
  - Threads run for an amount of time called a quantum
    - Can be cut short due to preemption by higher priority thread
  - The system treats all threads with the same priority as equal
  - No attempt to share processor(s) "fairly" among processes, only among threads

# Scheduling Scenarios: On wake

## On wake

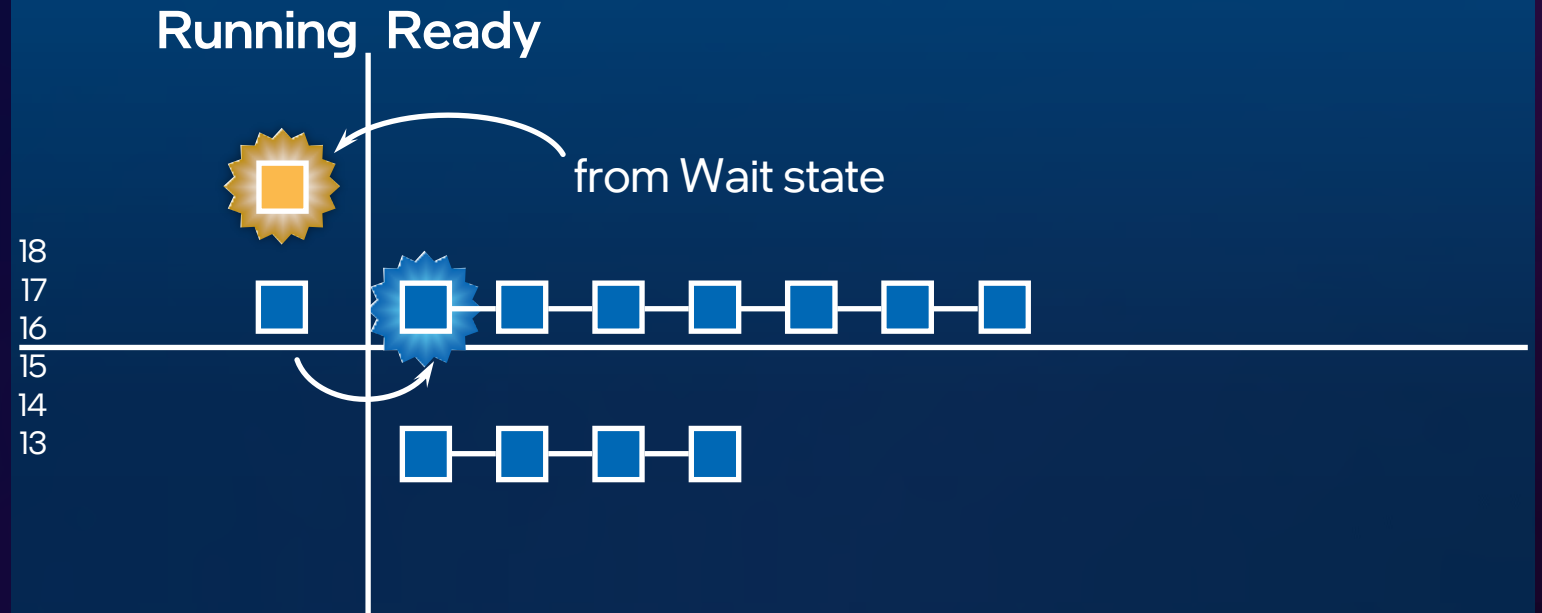
- If newly-Ready thread is not of higher priority than the Running thread...
  - ...it is put at the tail of the Ready queue for its current priority
- If priority  $\geq 14$  quantum is reset .
  - If priority  $< 14$  and you're about to be boosted and didn't already have a boost, quantum is set to process quantum - 1



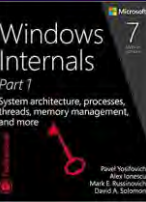
# Scheduling Scenarios: Preemption

## Preemption

- A thread becomes Ready at a higher priority than the running thread and all processors are busy
- Lower-priority *Running* thread is preempted
- Preempted thread goes back to head of its Ready queue
  - **Action:** pick lowest priority thread to preempt

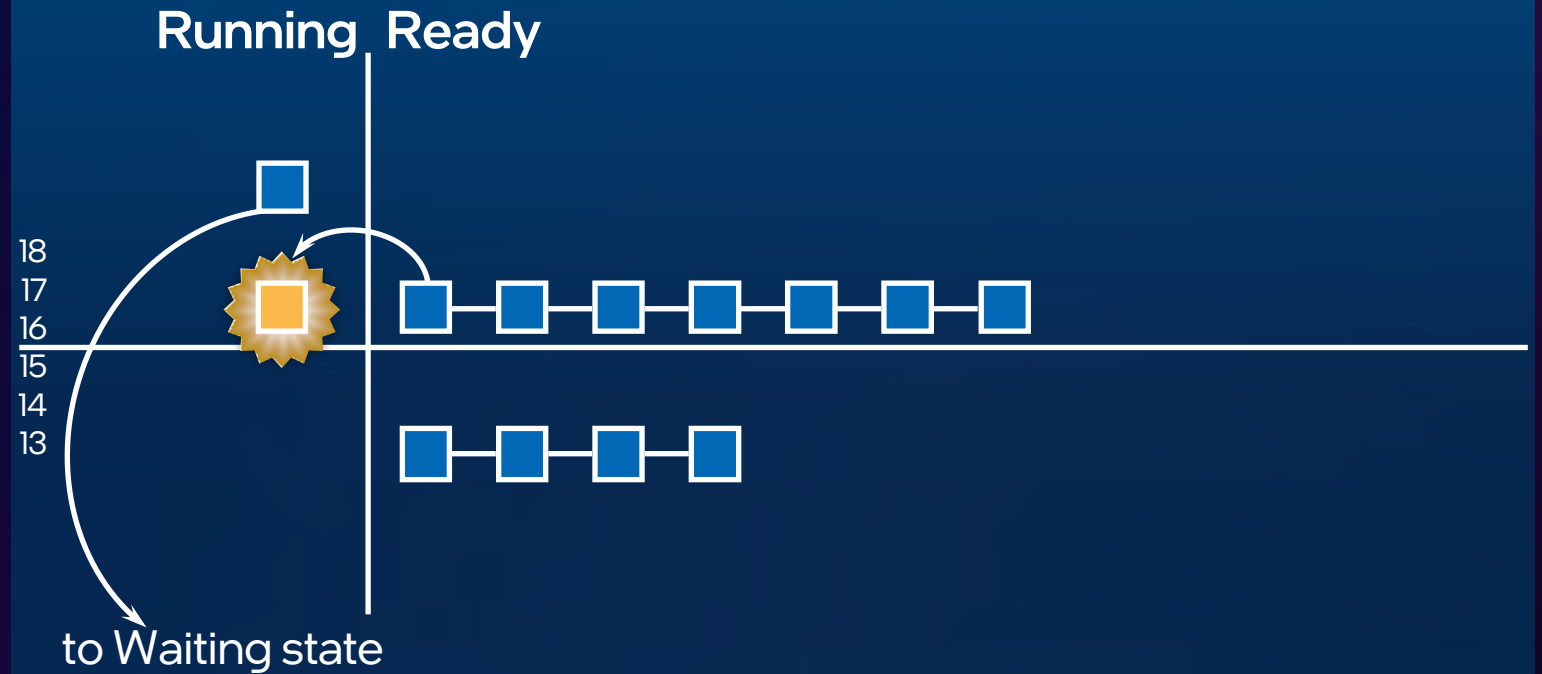


# Scheduling Scenarios: Voluntary switch



## Voluntary switch

- Waiting on a dispatcher object
- Termination
- Explicit lowering of priority
  - **Action:** scan for next Ready thread (starting at your priority & down)



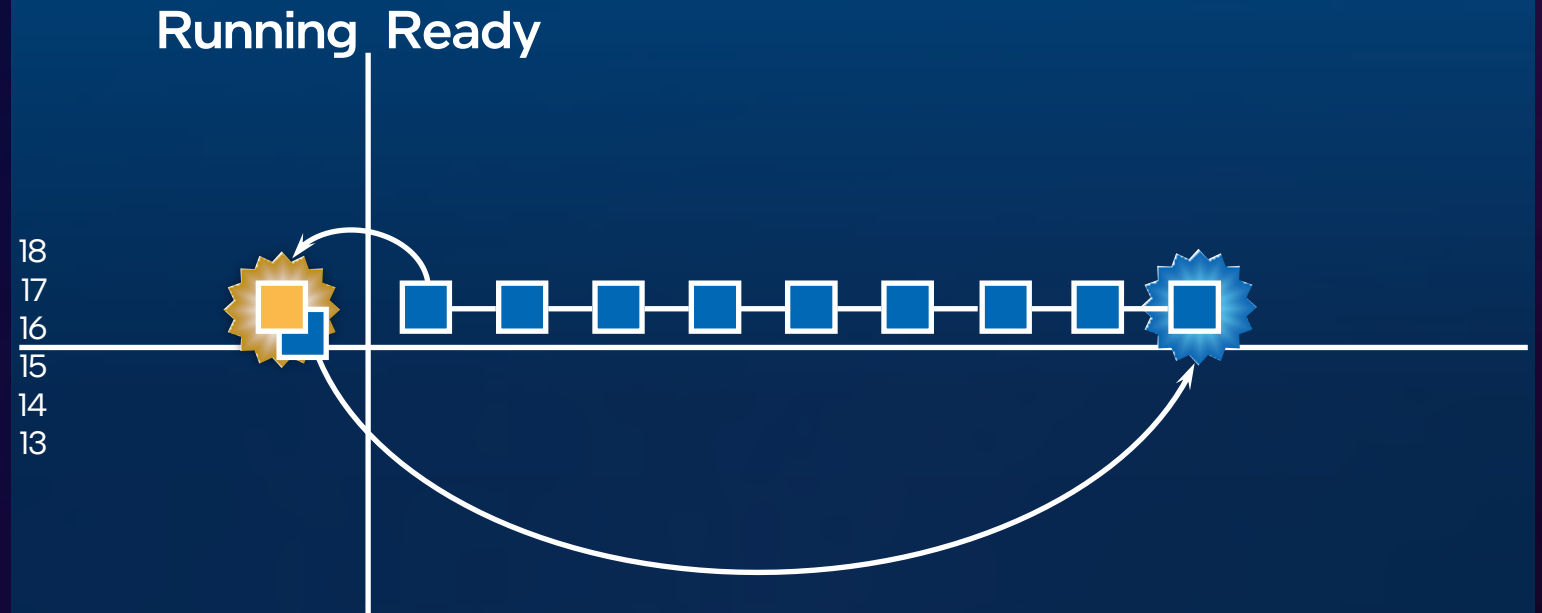
Current Hybrid systems have up to 8 LPs per group



# Scheduling Scenarios: Quantum End

## Running thread experiences quantum end

- Priority is decremented unless already at thread base priority
- Thread goes to tail of Ready queue for its new priority
- May continue running if no equal or higher-priority threads are Ready
- May migrate to ANOTHER LPs ready queue and start running thereafter
- **Action:** pick next thread at same priority level



# Priority Boosts

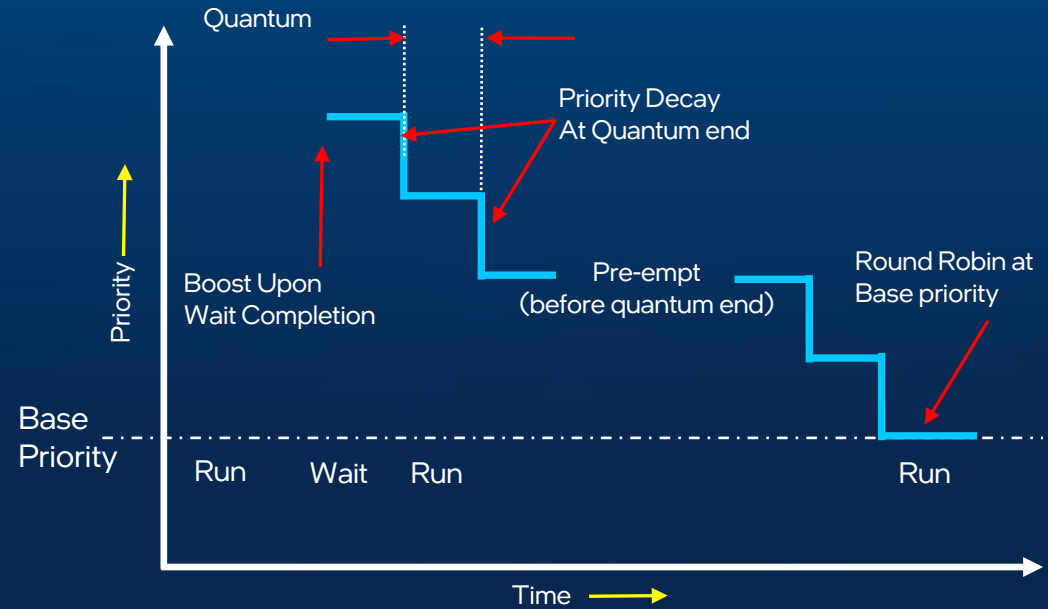
Windows periodically adjusts the current dynamic priority of threads, reasons include:

- Scheduler/dispatcher events:
  - An event is pulsed
  - A mutex/semaphore was released/abandoned
  - A timer was set
  - Other hardware interrupts
  - Has been in the ready queue a long time
  - A thread was alerted/suspended/resumed...

## Behavior of these boosts:

- Applied to thread's current priority will not take you above priority 15
- After a boost, you get one quantum. Then decays 1 level, runs another quantum

## Boost Decay over Time



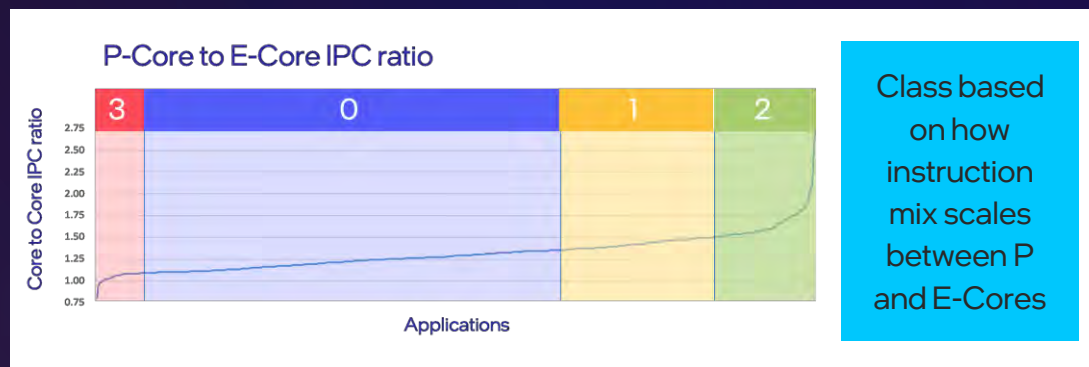
ThreadID	TotalCPU%	Priority (%)									
		<8	8	9	10	11	12	13	14	15	>15
5048	96.61	0	0	0	0	2.46	86.26	0.96	2.93	0	0

▲  
Thread Base Priority

# Intel® Thread Director Background

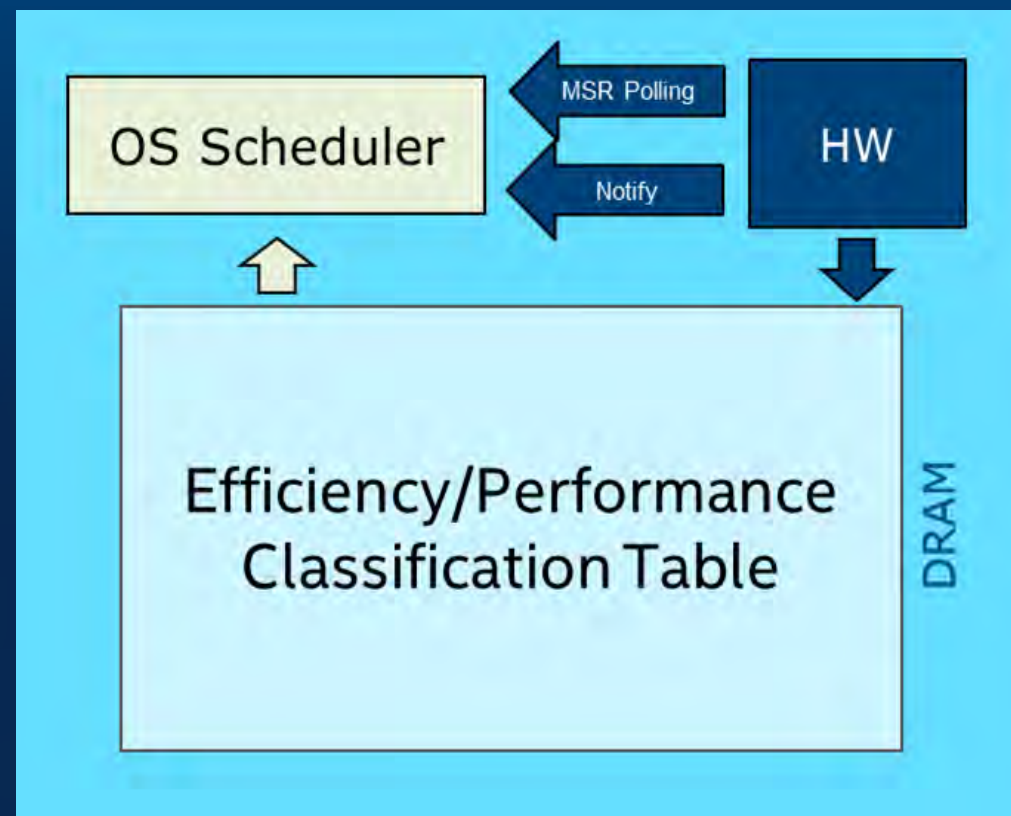
## Thread Scheduling Overview

- Processors that support x86 hybrid architecture are categorized on their performance and efficiency.
- Intel Thread Director provides a hint to the OS as to the thread that will benefit most from placement on a specific LP



- This hint is used within the same or lower QoS/Priority threads.
- HW periodically writes a feedback table (EHFI)

## Windows 11+ Only



# Intel® Core™ Processor Windows Scheduling/Parking Background

## Windows Core Parking Engine

- Makes global scalability decisions about the workload and determines the optimum set of compute cores for execution.
- Max Turbo vs All core frequency
- Enhance battery life
- Prioritize shared resources
- Etc....

## Power Management Settings related to Scheduling / Parking: Varies by power plan.

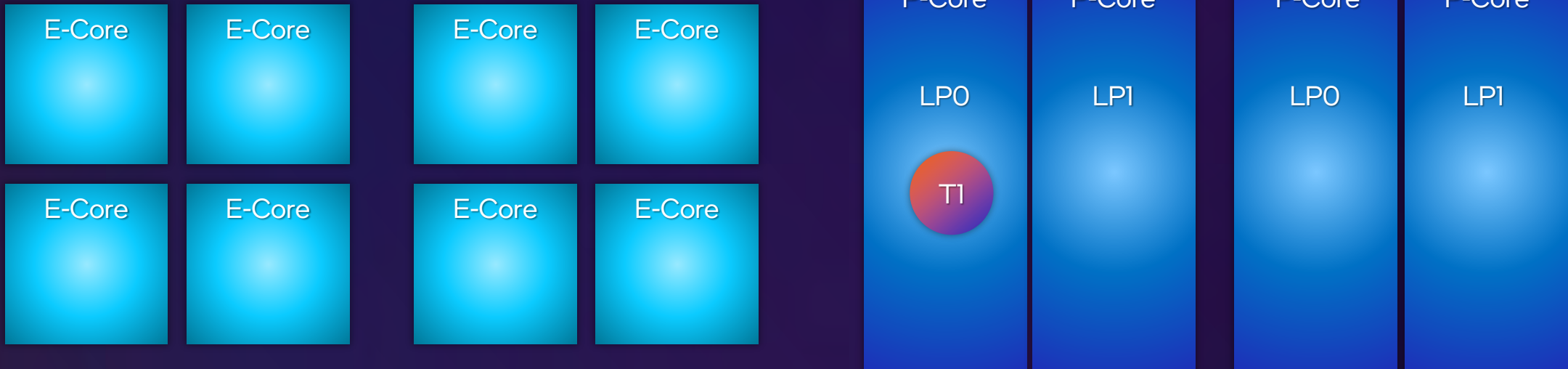
- CPMinCores: Specifies the minimum percentage of logical processors that can be unparked state at any given time.
- CPMaxCores: Specifies the maximum percentage of logical processors that can be unparked state at any given time.
- CPIncreaseTime: The minimum time that must elapse before additional logical processors can be transitioned from the parked to the unparked state.
- CPDecreaseTime: The minimum time that must elapse before additional logical processors can be transitioned from the unparked to the parked state.
- CPHeadroom: Specifies the additional utilization that would cause the core parking engine to unpark an additional parked logical processor



# Single Thread Scenario

- The following example shows Windows leveraging an Intel core for single thread performance.
- This behavior is dynamically achieved when Logical Processor (LP) 0 has the highest performance capability.

Example combination of P-cores and E-cores



# Limited Threaded Scenario

- The following example shows an example scheduling behavior in a limited software thread scenario.
- This behavior is dynamically achieved by the Windows scheduler/parking engine when P-Cores are more performant than the E-Cores. E-Cores are more performant than the SMT sibling of a busy core.
- When the capabilities dynamically change, Windows automatically accounts for this for optimal scheduling
- Favored core priority given to focus application threads, high priority or long duration threads

Example combination of P-cores and E-cores



# Multi Threaded Scenario

- All cores are used by Windows in multithread scenarios
- In power/thermal constraint scenarios, there may be times when all cores aren't used for optimal system performance/efficiency.
- The behavior is dynamically achieved by hardware providing feedback to Windows, and Windows automatically acting on that feedback.

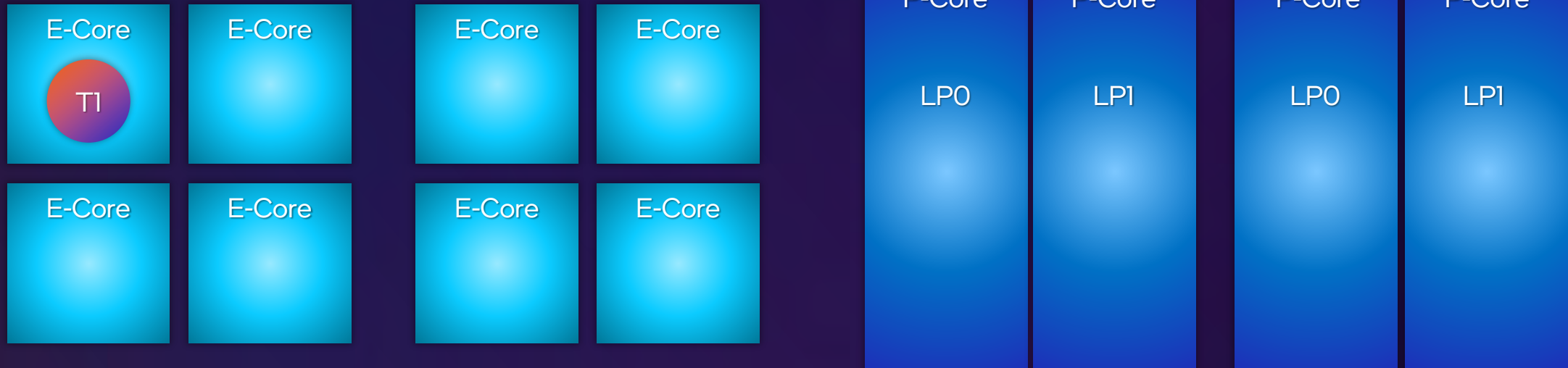
Example combination of P-cores and E-cores



# Low Power Scenario

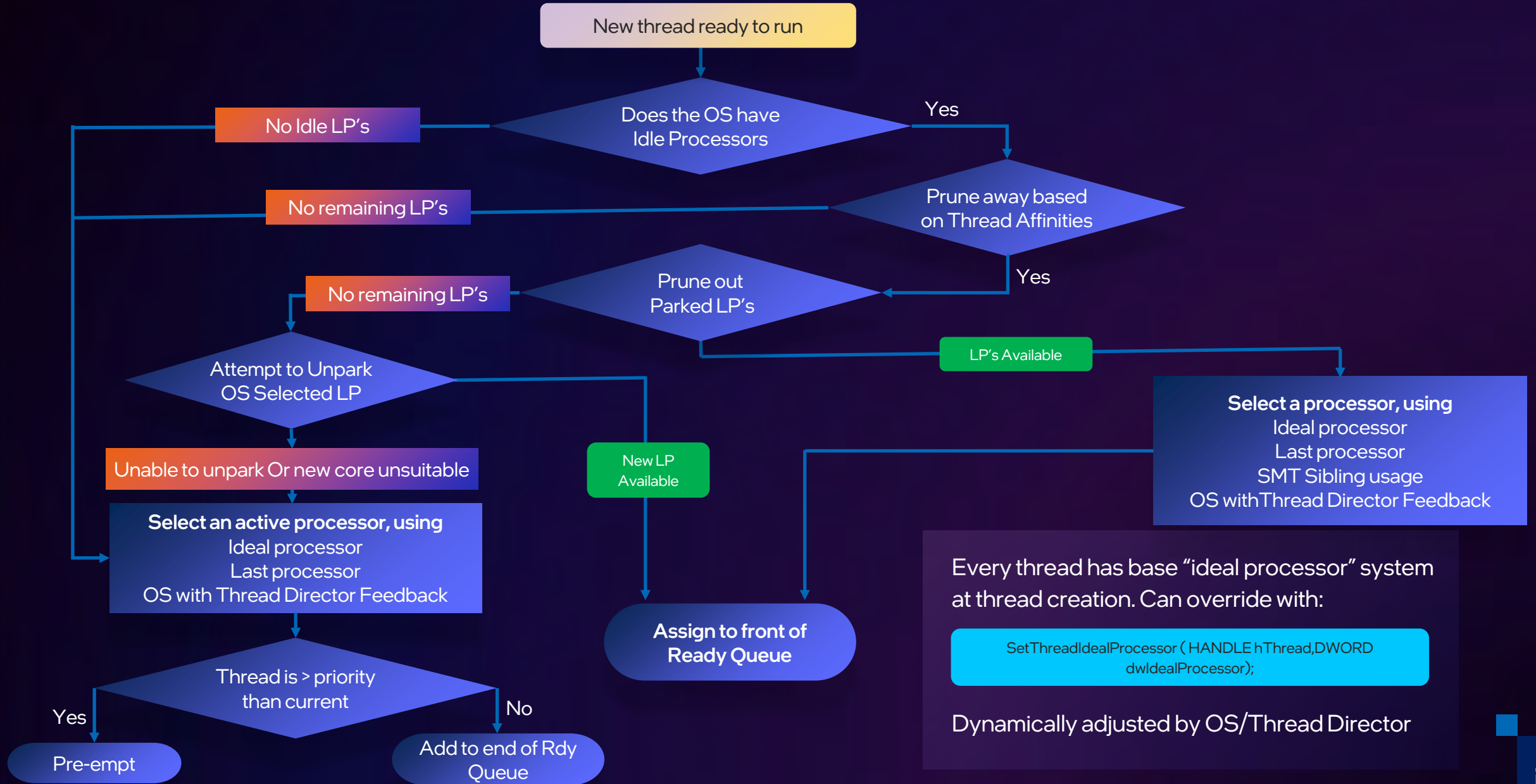
- In certain scenarios like low power envelope SKUs or better battery life goals, it can be more efficient to run low utilization work on cores with higher efficiency capability at efficient frequency

Example combination of P-cores and E-cores





# Simplified Processor/Ready Queue Selection



Every thread has base "ideal processor" system at thread creation. Can override with:

```
SetThreadIdealProcessor (HANDLE hThread, DWORD dwIdealProcessor);
```

Dynamically adjusted by OS/Thread Director

# Software Enabling for Hybrid

OS scheduler will move threads based on their priority, QoS and performance/efficiency HW metrics

## OS scheduler will try to assign work based on:

- Most performant core used first are used first for single-thread & multi-thread performance
- Spill over multi-threaded work uses additional physical for MT-performance
- SMT siblings are used last to avoid any contention impacting performance

## Core Parking

- OS will park inactive and lightly utilized logical processors.
- Saves power, or higher frequencies for running processors.
- Recent changes have made the OS more aggressive at parking for DC scenarios.
- AC scenarios are different due to softparklatency tunings

## Avoid Hard Affinities

- Use OS Hints for soft affinity
- Any ISV code providing affinity could potentially see perf degrades as OS cannot override the decision
- QoS, SetThreadIdealProcessor help determine where the OS will queue a thread to run.
- CPUSets API. This API takes HGS do not use hints into account and breaks affinity

## Workload Scalability

- Not all workloads scale with increased core count
- Increased threads add overhead in context switches/synchronization APIs, reduced cache and shared hardware resources
- Scale thread count based on workload benefits
- GetLogicalProcessorInformationEx
  - To scale application based on best fit to hardware

# DATA CAPTURE

A person wearing a headset is focused on their laptop screen in a dimly lit room. The background is filled with soft, out-of-focus bokeh lights in shades of blue and pink, suggesting a modern, possibly server or data center environment. The overall mood is professional and tech-oriented.

Introduction

OS Scheduling

Data Capture

Case Studies

Closing Thoughts

# Profiling Hybrid Games

FREE

VTune 2022



- CPI
- Cache
- Thread Director

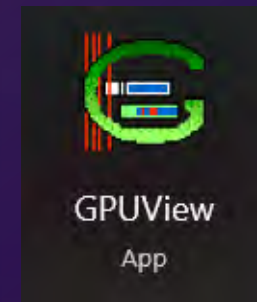
[VTune Download](#)

WPA



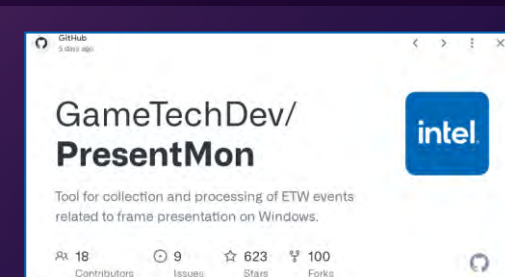
- SMT Usage
- Concurrency
- OS Behaviour

[Windows SDK](#)



[Windows SDK](#)

GPU/CPU  
Concurrency



[Github://Presentmon](#)

<https://learn.microsoft.com/en-us/windows-hardware/drivers/display/using-gpuview>

<https://developer.amd.com/wordpress/media/2012/10/Using%20GPUView%20to%20Understand%20your%20DirectX%2011%20Game.pps>



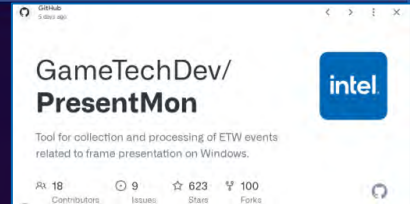
# Thanks to IO Interactive

- Example data collected from Hitman 3.
- Reproduced with permission from IO Interactive.
- Used to show profiling data only, title interacts with well with the OS.





# CPU or GPU Bound?



GPU bound will affect timings of CPU threads

- When idling waiting on the GPU the CPU will drop into lower C-states: Lowers CPU performance
- Changes thread concurrency
- Frame latency hides scheduling issues

## Capture timings with PresentMon:

```
> < presentmon -track_gpu -captureall -multi_csv -timed 20 -terminate_after_timed >
```

Or

- Capture with <Windows Performance Toolkit/GPUView/log.cmd light >
- View merged.etl in GPUView
- Post process with < presentmon -track\_gpu -etl\_file merged.etl -multi-csv >

## PresentMon output

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Applicatio	ProcessID	SwapChai	Runtime	SyncInter	PresentFl	Dropped	TimeInSec	msInPres	msBetwe	AllowsTe	PresentM	msUntilRe	msUntilDi	msBetwe	msUntilRe	msGPUAc
2	engine.ex	13724	0x000000C	DXGI	0	512	0	0.009755	0.3022	3.5081	1	Hardware	4.1077	4.1077	3.9256	0.1821	3.9256
3	engine.ex	13724	0x000000C	DXGI	0	512	0	0.013463	0.1472	3.7082	1	Hardware	4.3738	4.3738	3.9743	0.3995	3.9743
4	engine.ex	13724	0x000000C	DXGI	0	512	0	0.017083	0.1789	3.6194	1	Hardware	5.0149	5.0149	4.2605	0.7544	4.2605
5	engine.ex	13724	0x000000C	DXGI	0	512	0	0.021378	0.1583	4.2951	1	Hardware	5.6613	5.6613	4.9415	0.7198	4.9415
6	engine.ex	13724	0x000000C	DXGI	0	512	0	0.026208	0.2023	4.83	1	Hardware	4.7731	4.7731	3.9418	0.8313	3.9418
7	engine.ex	13724	0x000000C	DXGI	0	512	0	0.030531	0.1615	4.3226	1	Hardware	4.4052	4.4052	3.9547	0.4505	3.9547
8	engine.ex	13724	0x000000C	DXGI	0	512	0	0.035185	0.1769	4.6541	1	Hardware	3.702	3.702	3.9509	-0.2489	3.9509
9	engine.ex	13724	0x000000C	DXGI	0	512	0	0.039992	0.166	4.8078	1	Hardware	2.8466	2.8466	3.9524	-1.1058	3.946
10	engine.ex	13724	0x000000C	DXGI	0	512	0	0.044595	0.1789	4.6027	1	Hardware	2.9354	2.9354	4.6915	-1.7561	4.1153
11	engine.ex	13724	0x000000C	DXGI	0	512	0	0.049307	0.1798	4.7115	1	Hardware	2.7933	2.7933	4.5694	-1.7761	4.1034
12	engine.ex	13724	0x000000C	DXGI	0	512	0	0.054221	0.1599	4.9143	1	Hardware	3.5637	3.5637	5.6847	-2.121	4.8955
13	engine.ex	13724	0x000000C	DXGI	0	512	0	0.058722	0.1463	4.5014	1	Hardware	3.1657	3.1657	4.1034	-0.9377	4.0963
14	engine.ex	13724	0x000000C	DXGI	0	512	0	0.063595	0.1695	4.8724	1	Hardware	2.7361	2.7361	4.4428	-1.7067	4.1992
15	engine.ex	13724	0x000000C	DXGI	0	512	0	0.068466	0.1854	4.8711	1	Hardware	2.7998	2.7998	4.9348	-2.135	4.1242
16	engine.ex	13724	0x000000C	DXGI	0	512	0	0.072951	0.1581	4.4855	1	Hardware	2.7594	2.7594	4.4451	-1.6857	4.1327
17	engine.ex	13724	0x000000C	DXGI	0	512	0	0.077578	0.1589	4.6265	1	Hardware	2.993	2.993	4.8601	-1.8671	4.3796
18	engine.ex	13724	0x000000C	DXGI	0	512	0	0.082237	0.1837	4.6587	1	Hardware	2.9332	2.9332	4.5989	-1.6657	4.0976

msBetweenPresents  
= CPU timings

msUntilDisplayed  
= Display Latency

msGPUActive  
= GPU Timing

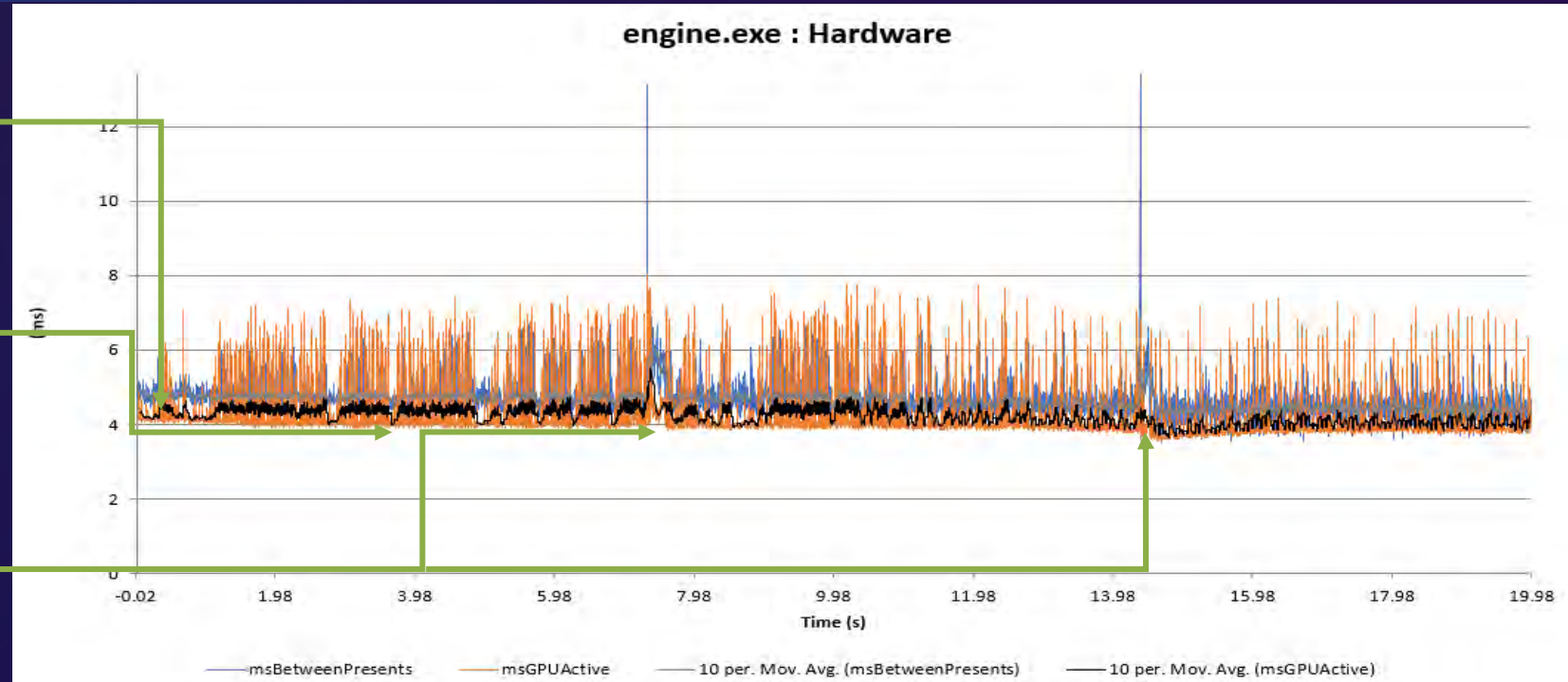
# Viewing GPU Data

Simple to plot timings from PresentMon

GPU 1ms faster than CPU

Big CPU frame time variations

Areas of interest: Possible IO stalls, memory paging

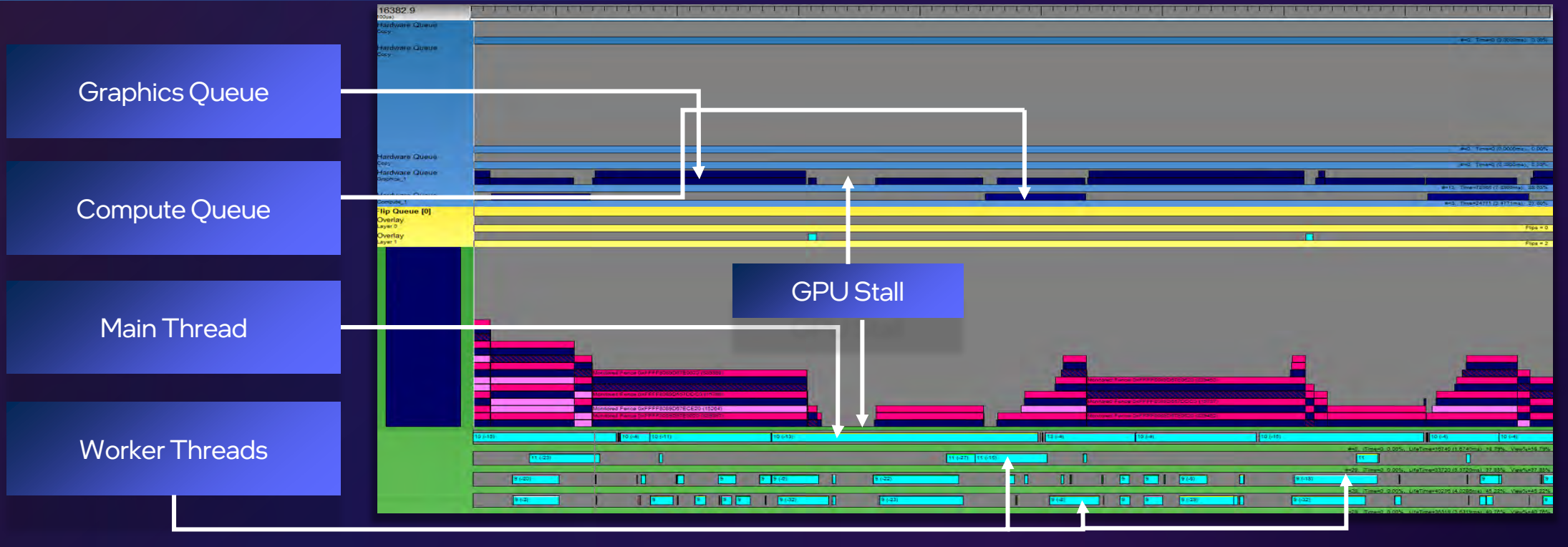


# Viewing GPU Data



GPUView  
App

## GPUView





# Understanding Concurrency



## WPA: Timeline by Process, Thread



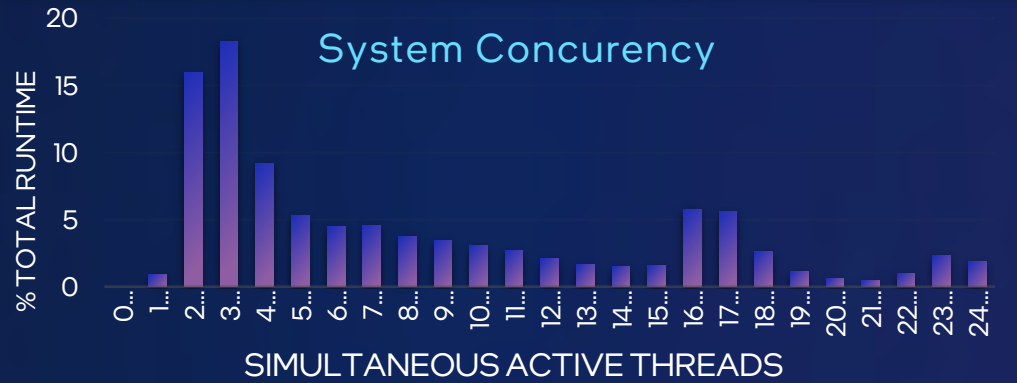
## CPU Concurrency:

Number of simultaneous active threads at and point in a frame.

## Xperf/WPA:

Precise view, tracks kernel events in ETL files. Provides a fine-grained view of individual threads.

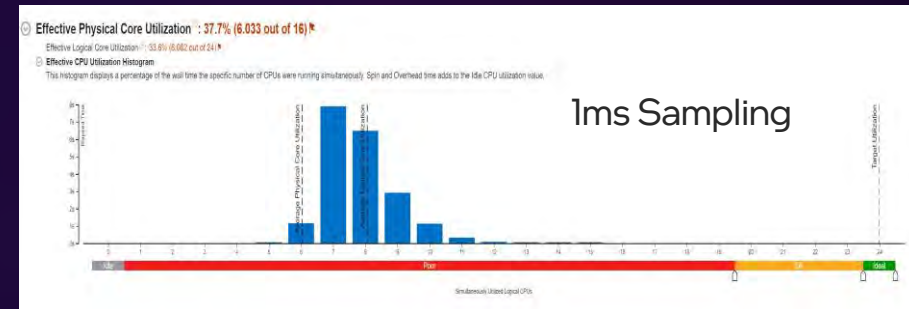
Sampled concurrency views like VTune don't provide enough detail on concurrency at the OS event level.



P-Cores Only

P+E-Cores

P+E-Cores + SMT



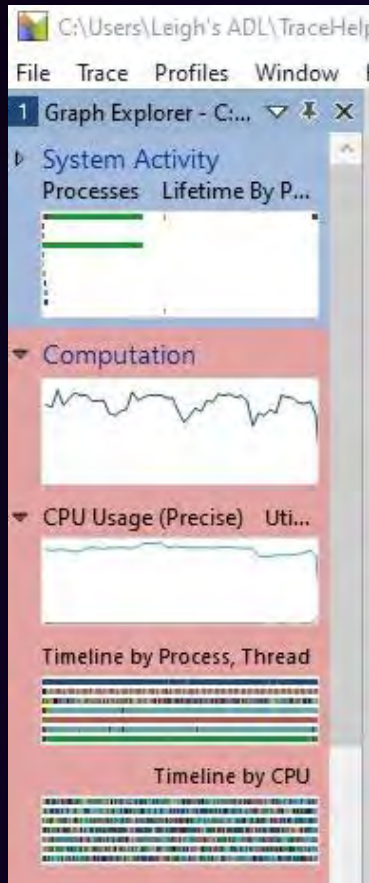


# Understanding WPA (1/5)



A lot of customisable views into OS/hardware level data

- Computation
  - CPU Usage
    - Timeline by Process, Thread  
(Precise) ← Track events rather than using sampling



Thread-level timings

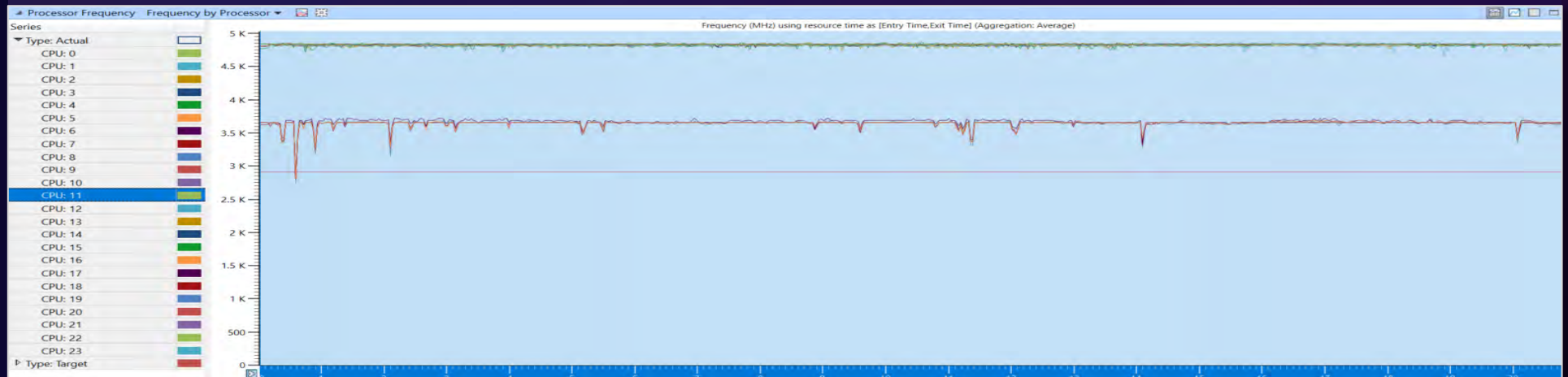
# Understanding WPA (2/5)



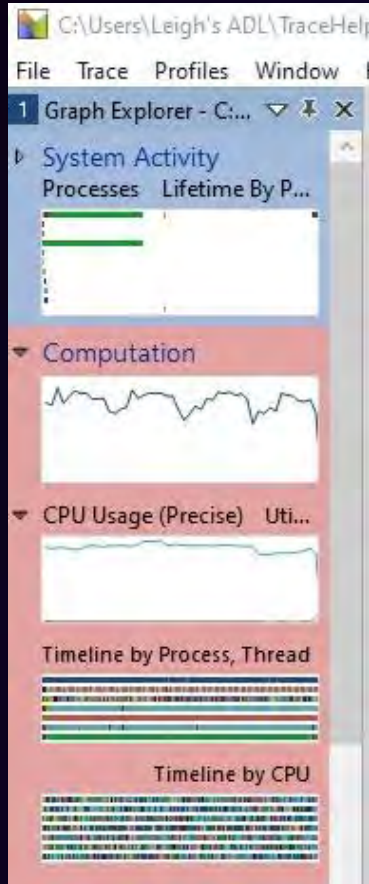
A lot of customisable views into OS/hardware level data

- Power
  - Processor Utility
  - Processor Frequency

View Type Actual



Hardware frequency including turbo

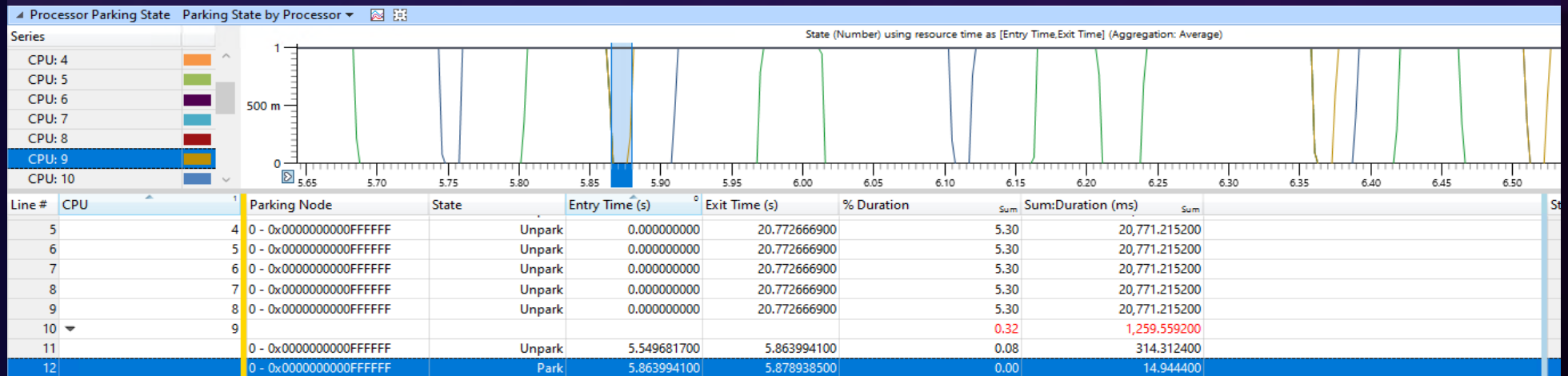
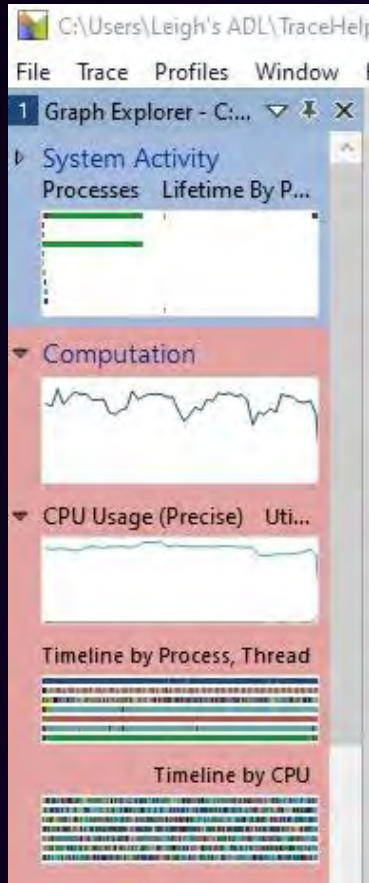


# Understanding WPA (3/5)



A lot of customisable views into OS/hardware level data

- Power
  - Processor Parking State



Logical processor state

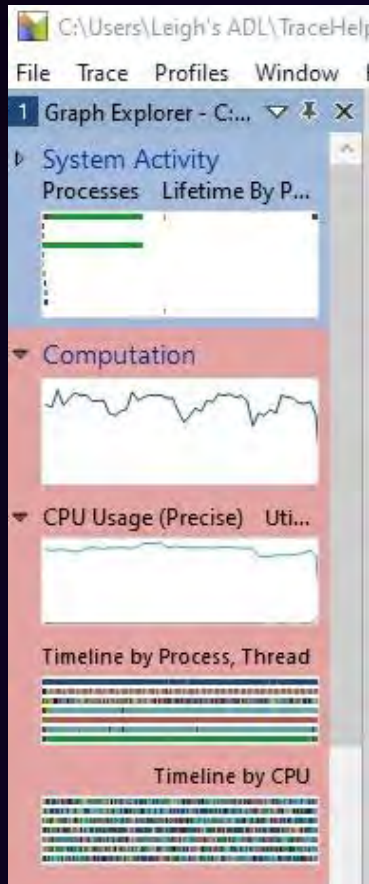


# Understanding WPA (4/5)



A lot of customisable views into OS/hardware level data

- Generic Events
- Sorted by Service Provider
  - Microsoft-Windows-DirectD3D12
  - Microsoft-Windows-Kernel-Processor-Power
  - Microsoft-Windows-DXGI



Useful to track where GPU commands are issued



# Understanding WPA (5/5)



Thread Timeline

Core Parking

DX12 Events

Multiple views use same zoom/easy to align data





# Thread Execution

## CPU :Timeline by Process, Thread

### Add Cpu + CPU Usage(ms) Sum

The screenshot shows the 'Advanced' column configuration dialog in Windows Performance Monitor. The 'Available Columns' list on the left includes 'Cpu' and 'CPU Usage (ms)'. The 'Visible' list on the right shows 'Cpu' and 'CPU Usage (ms)' checked. The 'Aggregation' for 'CPU Usage (ms)' is set to 'Sum', and the 'Sort' is set to 'Ascending'.

### Sort By CPU Usage (Sum)

Line #	NewProcess	NewThrea...	Cpu	CPU Usage (ms)	Count	ReadyingProcess	ReadyingT...	Ready (us)	Sum	Ready (us)	Max	Waits (us)	Sum	Waits
1	engine.exe (5108)		5	167,039.115300	411,1...			12,749,523.900	525.600	807,950,656.400				
2		13,968	5	20,147.110800	6,731			19,879.200	136.200	627,395.800				
3			5	5,237.874200	2,200			2,612.700	59.200	297,133.300				
4			4	4,552.729200	739				60.000	51,906.400				
5			6	4,253.986700	599				58.600	27,604.800				
6			7	3,744.251400	942				50.000	93,215.900				
7			12	1,764.347500	1,250				98.400	91,704.000				
8			1	67,005300	61				32.300	2,842.400				
9			8	66,321500	63				39.000	6,217.000				
10			14	60,940700	55				26.500	4,061.600				
11			9	54,081700	57				52.300	4,666.200				
12			0	52,717600	67				17.800	4,552.800				
13			3	50,865700	51				136.200	3,128.400				
14			10	50,751400	59				33.300	4,044.600				
15			15	47,237200	33				85.200	1,362.900				
16			13	37,993000	31				95.300	1,879.600				
17			11	34,226900	35				76.500	2,123.200				
18			2	17,543800	30				56.100	1,931.900				
19			17	9,571600	79				77.600	4,396.300				
20			16	8,509300	93				82.600	5,720.900				
21			19	7,374800	57				63.800	4,497.700				
22			23	7,343900	42				71.500	4,947.800				
23			20	6,515300	42				77.500	2,549.900				
24			22	5,361800	48				62.600	1,911.200				
25			18	4,877200	46				77.400	2,475.400				
26			21	4,683100	52				1,042.800	69.000			2,521.600	
27		5,048	20	20,084.293500	29,891			65,548.000	125.100	645,930.800				
28			2	7,774.779800	13,955			27,129.400	100.100	335,757.500				
29			6	3,502.073700	3,641			5,955.200	65.000	62,667.300				

The screenshot shows a context menu over the table data. The 'Copy Selection' option is highlighted, indicating that the data can be copied to Excel for further analysis.

Breakdown thread execution time by Core type  
 Long duration threads should favour Performance cores  
 WPA Tabled can be copied to Excel and graphed.





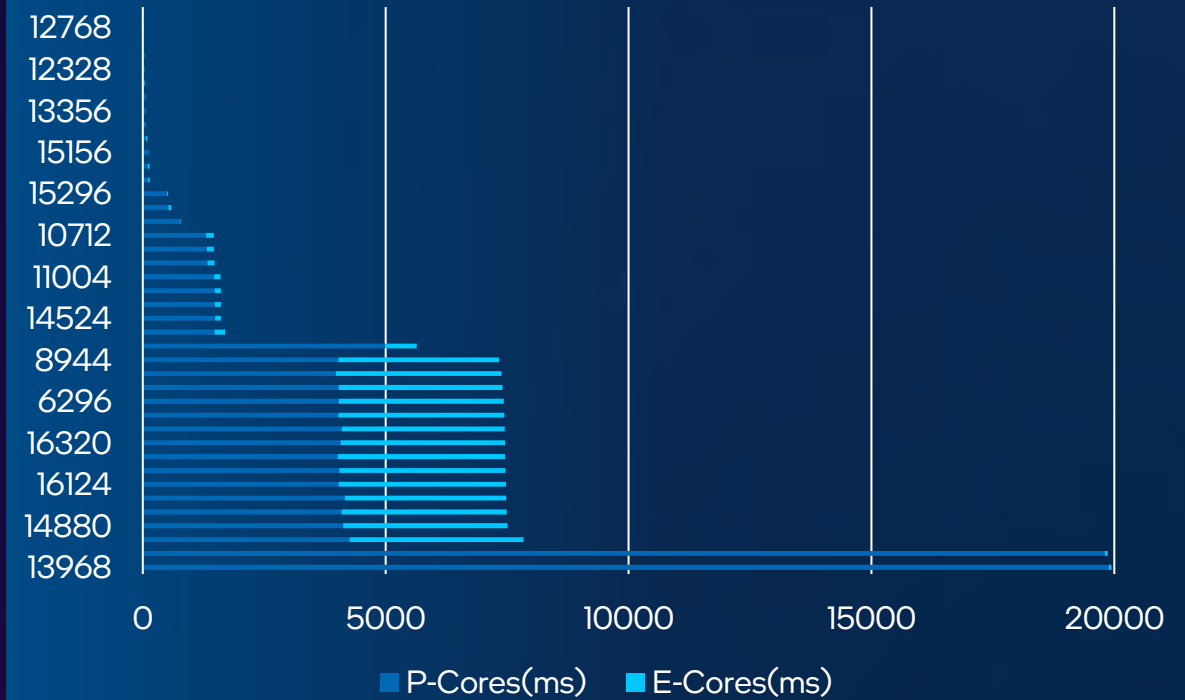
# Thread Execution

## CPU :Timeline by Process, Thread

### Add Cpu + CPU Usage(ms) Sum

Available Columns	Visible	Name	Aggregation	Sort
<input type="checkbox"/> % CPU Usage	<input type="checkbox"/>	NewProcess Name	None	None
<input type="checkbox"/> Annotation	<input checked="" type="checkbox"/>	NewProcess	None	None
<input type="checkbox"/> Count	<input checked="" type="checkbox"/>	NewThreadId	None	None
<input type="checkbox"/> Count:Waits	<input checked="" type="checkbox"/>	Cpu	None	Ascending
<input type="checkbox"/> Cpu	<input checked="" type="checkbox"/>	CPU Usage (ms)	Sum	None
<input type="checkbox"/> CPU Usage (ms)	<input type="checkbox"/>	New Thread Stack Tag		
<input type="checkbox"/> IdealCpu	<input type="checkbox"/>	New Thread Stack (Frame Tags)		
<input type="checkbox"/> LastSwitchOutTime (s)	<input type="checkbox"/>	NewThreadStack		
<input type="checkbox"/> NewInPri	<input type="checkbox"/>	Ready Thread Stack Tag		
<input type="checkbox"/> NewInSwitchTime (us)	<input type="checkbox"/>	Ready Thread Stack (Frame Tags)		
<input type="checkbox"/> NewOutPri	<input type="checkbox"/>	ReadyThreadStack		
<input type="checkbox"/> NewPrevOutPri	<input type="checkbox"/>	ReadyingProcess Name	None	None
<input type="checkbox"/> NewPrevState	<input type="checkbox"/>	Thread Name	None	None
<input type="checkbox"/> NewPrevWaitMode	<input type="checkbox"/>	Thread Activity Tag	None	None
<input type="checkbox"/> NewPrevWaitReason	<input type="checkbox"/>	Annotation	None	None
<input type="checkbox"/> NewPriDecr	<input type="checkbox"/>	ReadyTime (s)	None	None
<input type="checkbox"/> NewProcess	<input checked="" type="checkbox"/>	Count	Count	Descending
<input type="checkbox"/> NewProcess Name	<input type="checkbox"/>	TimeSinceLast (us)	Sum	None
<input type="checkbox"/> NewQnt	<input type="checkbox"/>	TimeSinceLast (us)	Max	None
<input type="checkbox"/> NewState	<input type="checkbox"/>			
<input type="checkbox"/> NewThreadId	<input type="checkbox"/>			
<input type="checkbox"/> NewThreadStack	<input type="checkbox"/>			
<input type="checkbox"/> NewThreadStartFunction	<input type="checkbox"/>			
<input type="checkbox"/> NewThreadStartModule	<input type="checkbox"/>			
<input type="checkbox"/> NewWaitMode	<input type="checkbox"/>			
<input type="checkbox"/> NewWaitReason	<input type="checkbox"/>			

### Thread Duration by Core type



<https://devblogs.microsoft.com/performance-diagnostics/wpa-intro/>





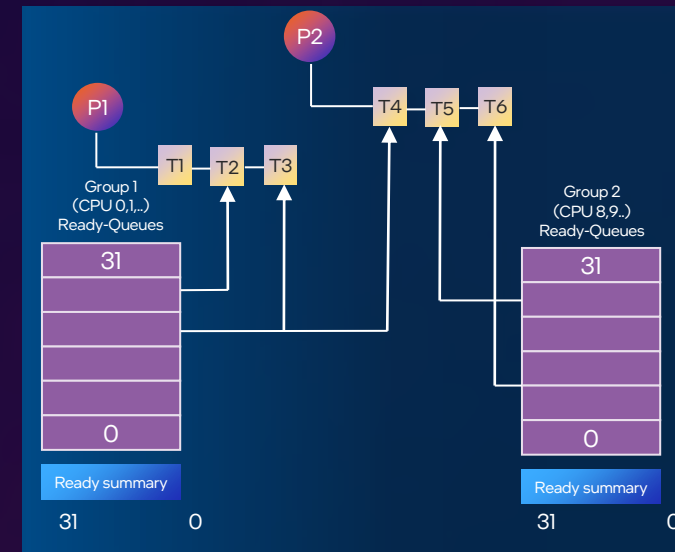
# Thread Ready/Wait times

Are threads efficiently scheduled?

How long do they wait and is the OS able to schedule them?

The screenshot shows the 'CPU Usage (Precise) View Editor' window. The 'Available Columns' list includes 'Ready (us)' and 'Waits (us)', both of which are checked. The 'Column Details' for 'Waits (us)' are shown on the right, with 'Text Alignment' set to 'Right', 'Width' set to '100', and 'Custom format' set to 'Milliseconds'. A blue callout box points to the 'Custom format' dropdown with the text: 'Custom format doesn't change the column header on some versions of WPA'.

Line #	NewProcess	NewThreadId	CPU Usage (ms)	Ready (us)	Waits (us)	Count	ReadyingProcess	ReadyingT...	Ready (us)	Waits (us)	Count
1	engine.exe (11484)		166,635.621200	14,115.191600	814,821.719900	1,390,7...			4,894.500	1,000,929.800	1,3...
2			9,836	266,858500	14,588.815000	76,090			1,477.600	8,317.200	
3			14,648	584,434500	12,354.585600	65,518			1,440.100	9,053.000	
4			13,148	699,908600	12,404.492500	65,338			1,390.500	9,050.400	



## CPU Usage (Precise)

- Add CPU usage to table
- Move *Ready* & *Wait* up the table
- Reformat units

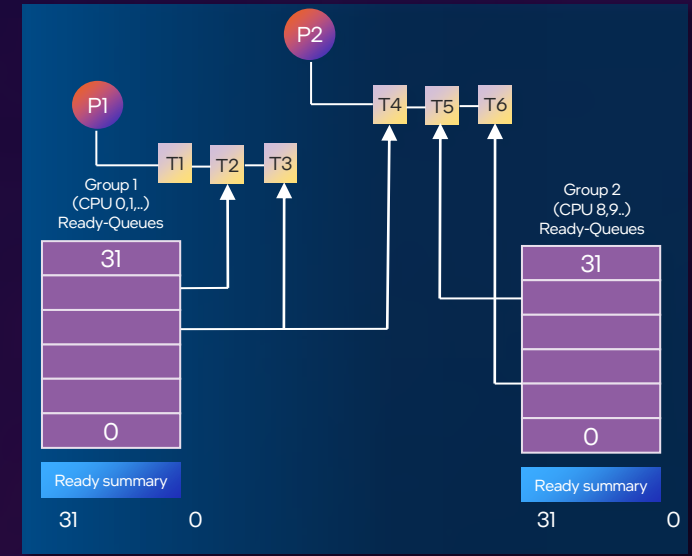


# Thread Ready/Wait times



Thread Job	New ThreadId	CPU Usage (ms)	Ready (ms) Sum	Waits (ms) Sum	Count	Ready (us) Max	Waits (us) Max	Count: Waits
Render	7444	20125.96	43.94	749.32	9226	381.3	50467.6	5044
Game	11256	19926.66	62.12	831.01	37873	426	3319.6	27374
Worker 1	10376	8285.756	240.84	12276.28	75037	692.7	10014.7	71946
Worker 2	10852	8233.223	237.80	12327.32	76566	762.7	10017.6	73448
Worker 3	7976	8213.791	254.07	12326.72	75698	526.4	10013.5	72006

Non Hybrid System



# Thread Ready/Wait times



Hybrid system has:

- Smaller Wait time for main 2 threads
- Less Ready time on Render thread

Less context switches for all main threads

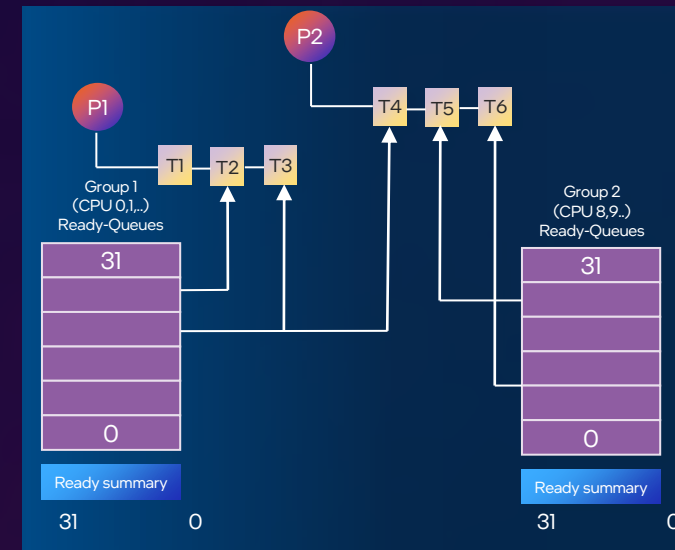
Thread Job	New ThreadId	CPU Usage (ms)	Ready (ms) Sum	Waits (ms) Sum	Count	Ready (us) Max	Waits (us) Max	Count: Waits
Render	7444	20125.96	43.94	749.32	9226	381.3	50467.6	5044
Game	11256	19926.66	62.12	831.01	37873	426	3319.6	27374
Worker 1	10376	8285.756	240.84	12276.28	75037	692.7	10014.7	71946
Worker 2	10852	8233.223	237.80	12327.32	76566	762.7	10017.6	73448
Worker 3	7976	8213.791	254.07	12326.72	75698	526.4	10013.5	72006

Non Hybrid System

Smaller max. Ready time

Hybrid System

Thread Job	New ThreadId	CPU Usage (ms)	Ready (ms) Sum	Waits (ms) Sum	Count	Ready (us) Max	Waits (us) Max	Count: Waits
Render	13968	20147.11	19.88	627.40	6731	136.2	6593.4	5225
Game	5048	20084.29	65.55	645.93	29891	125.1	2785.3	26094
Worker 1	13324	7927.543	489.20	12406.89	67058	344.7	18072.9	63135
Worker 2	14880	7589.991	714.03	12529.15	65649	334.2	17040.8	62649
Worker 3	16312	7570.553	707.25	12547.99	66215	259.9	18143.7	63147



Ideally compare against 2 systems

- i.e. Intel i9-12900K
- e-cores on vs off in bios

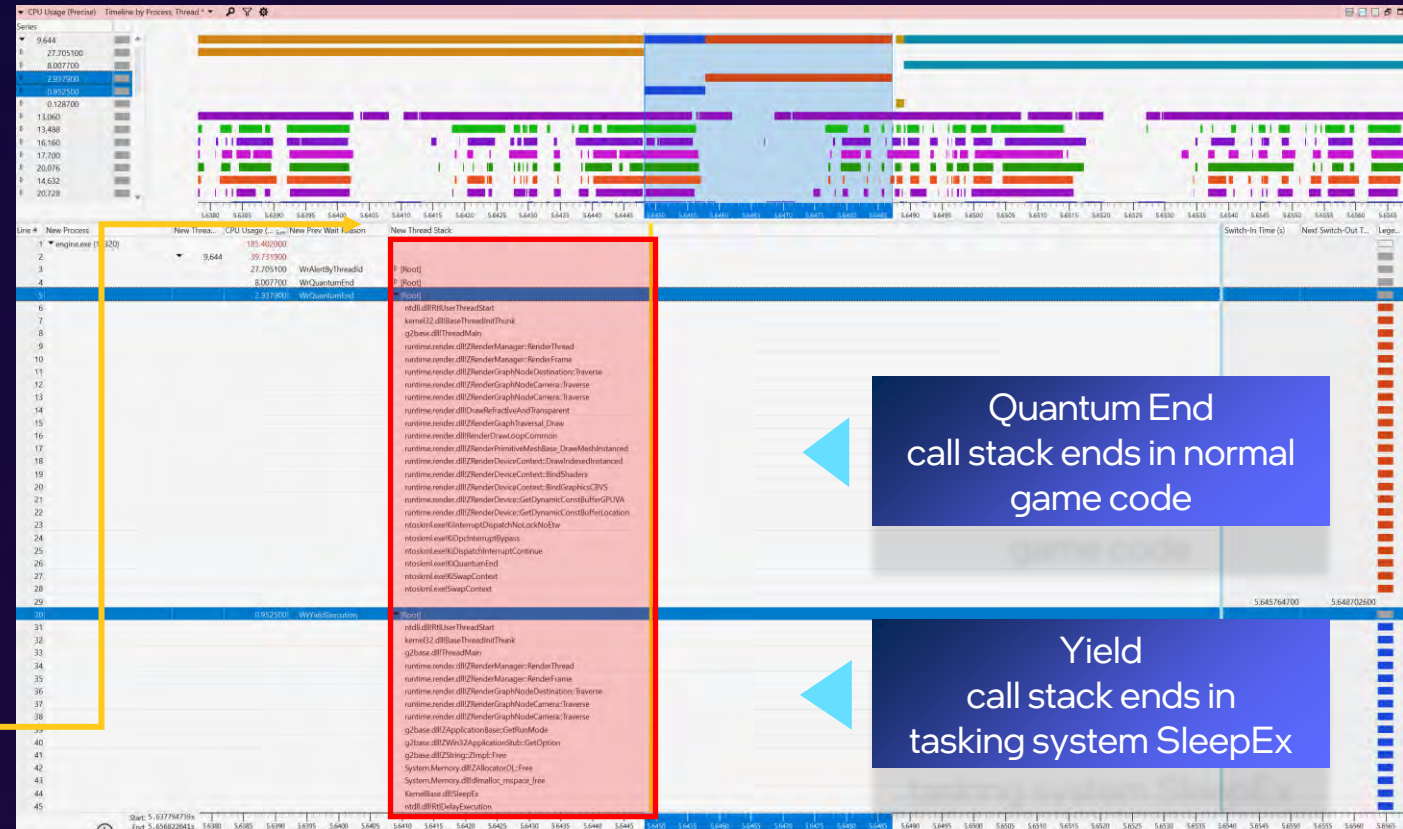
# Understanding Context switches



Context Switch is the process of changing the active thread on a processor.

Overhead of changing architecture state

- Capture with `log.cmd` normal
- Load symbols (MSFT symbol server)
- Add `NewThreadStack` to table view



Quantum End  
call stack ends in normal  
game code

Yield  
call stack ends in  
tasking system SleepEx



**New Thread Stack**  
The stack of the new thread when it is switched in. Usually indicates what the thread was blocked or waiting on.



# Intel® VTune™ Profiler



Advanced sampling profiler allows you to quickly identify CPU bottlenecks causing slow frames and tasks.

**Hotspots** Hotspots by CPU Utilization

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee

Elapsed Time: 28.691s

- CPU Time: 42.634s
- Instructions Retired: 145,987,400,000
- Microarchitecture Usage: 39.6% of Pipeline Slots
- Total Thread Count: 51
- Paused Time: 0s
- Frame Count: 914

**Top Hotspots**

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time
Enlighten:Impl:GetProbeInterpolants	unityplayer.dll	5.304s
Enlighten:SolveProbesL2	unityplayer.dll	3.856s
func@0x1800ab2a0	mono-2.0-bdwcg.dll	0.955s
Enlighten:SolveDirectionalIrradiance<1>	unityplayer.dll	0.757s
RtlCopyMemory	ntoskrnl.exe	0.536s
[Others]		31.226s

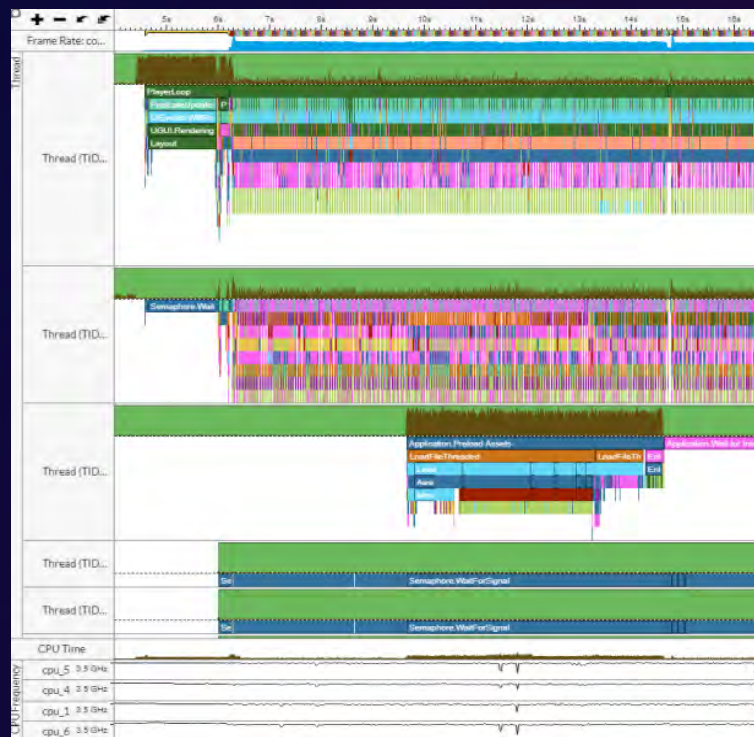
*\*N/A is applied to non-summable metrics.*

**Top Tasks**

This section lists the most active tasks in your application.

Task Type	Task Time	Task Count
Idle	160.075s	1,176,134
Semaphore.WaitForSignal	133.942s	42,906
Audio.Thread	25.274s	6,937
PlayerLoop	22.963s	913
Camera.Renderer	15.377s	3,656
[Others]	161.089s	2,628,856

Hotspot Analysis: Identifies functions consuming the most CPU time



Thread Performance: Visualizes thread behavior to quickly identify concurrency problems

Grouping: Task Domain / Task Type / Function / Call Stack

Task Domain / Task Type / Function / Call Stack	CPU Time	Instructions Retired	Microarchitecture Usage		Task Time
			Microarchitecture Usage	CPI Rate	
UE4Domain	65.847s	211,632,115,937	29.8%	0.990	155.252s
FDeferredShadingSceneRenderer_Render	8.954s	20,558,560,980	23.8%	1.350	15.575s
FDeferredShadingSceneRenderer_InitViews	1.822s	4,945,551,288	26.5%	1.178	3.340s
FSceneRenderer_ComputeViewVisibility	1.582s	4,621,327,612	26.9%	1.112	2.903s
UWorld_Tick	1.463s	5,770,821,626	28.6%	0.887	4.109s
FCompression_UncompressMemory	1.187s	4,825,032,389	60.5%	0.712	1.692s
FScene_UpdateAllPrimitiveSceneInfos	0.580s	1,380,673,014	18.5%	1.356	0.851s
FScene_AddPrimitiveSceneInfos	0.558s	1,331,746,702	18.2%	1.348	0.815s
FScene_AddPrimitiveSceneInfoToScene	0.535s	1,312,085,470	18.5%	1.328	0.795s
FDeferredShadingSceneRenderer_RenderLights	0.333s	673,865,649	22.1%	1.554	0.564s
Slate:Tick	0.219s	203,536,533	30.3%	3.053	0.635s
FViewport_Draw	0.166s	43,686,757	20.5%	2.930	0.291s
Slate:DrawWindows	0.164s	181,458,102	34.6%	2.761	0.539s
FAudioDevice_Update	0.159s	37,496,378	59.6%	2.715	0.207s
FDeferredShadingSceneRenderer_InitViewsPossib	0.147s	206,229,370	27.9%	2.055	0.270s
Slate:DrawWindow_Render Thread	0.112s	204,785,341	35.5%	1.737	0.172s
FSceneRenderer_InitDynamicShadows	0.103s	152,822,069	27.6%	1.938	0.188s
Frame 0	0.085s	199,353,144	22.4%	1.395	1.340s
Slate:DrawWindow	0.078s	24,238,960	16.6%	5.495	0.126s
Frame 324	0.072s	271,605,376	64.5%	0.872	0.145s
Frame 387	0.070s	263,047,990	2.9%	0.882	0.157s
Frame 214	0.069s	260,711,662	30.9%	0.896	0.156s
Frame 211	0.069s	265,593,751	11.8%	0.867	0.155s
Frame 377	0.069s	254,510,381	0.0%	0.892	0.155s
Frame 248	0.069s	268,360,839	2.0%	0.864	0.156s
Frame 379	0.068s	239,726,238	4.7%	0.936	0.157s
Frame 269	0.068s	263,348,608	0.0%	0.851	0.146s
Frame 217	0.068s	262,593,844	2.0%	0.867	0.148s
Frame 272	0.067s	254,704,545	0.0%	0.875	0.153s
Frame 821	0.067s	240,466,446	17.1%	0.871	0.144s
Frame 231	0.067s	249,131,002	0.0%	0.879	0.145s
Frame 357	0.067s	257,373,132	0.0%	0.864	0.139s
Frame 224	0.067s	249,544,739	31.6%	0.881	0.162s
Frame 370	0.067s	261,285,508	5.4%	0.857	0.142s
Frame 381	0.067s	238,690,251	40.2%	0.897	0.158s
Frame 233	0.067s	250,635,455	37.1%	0.876	0.154s
Frame 229	0.067s	254,875,318	11.2%	0.882	0.155s

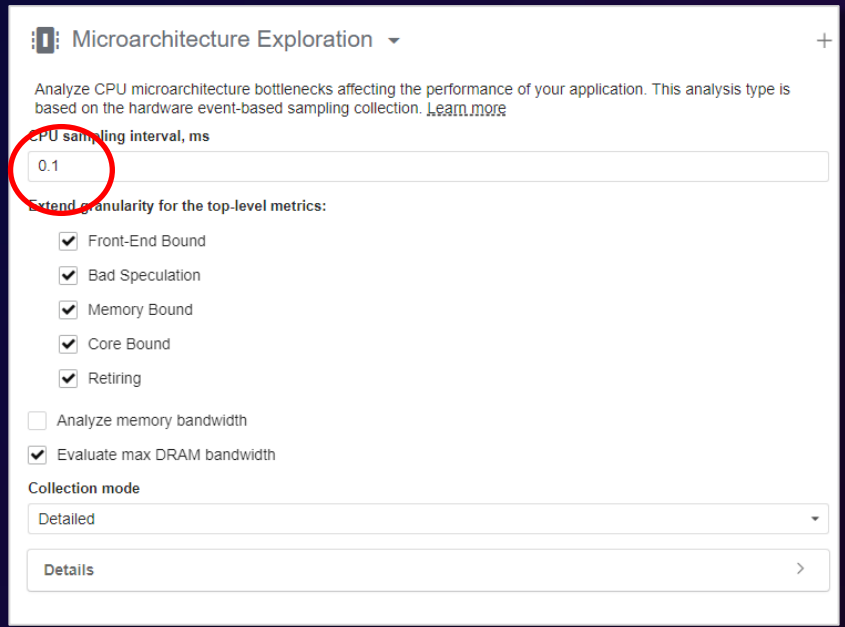
Instrumentation API: Extensive API enables frame and task markup for better results



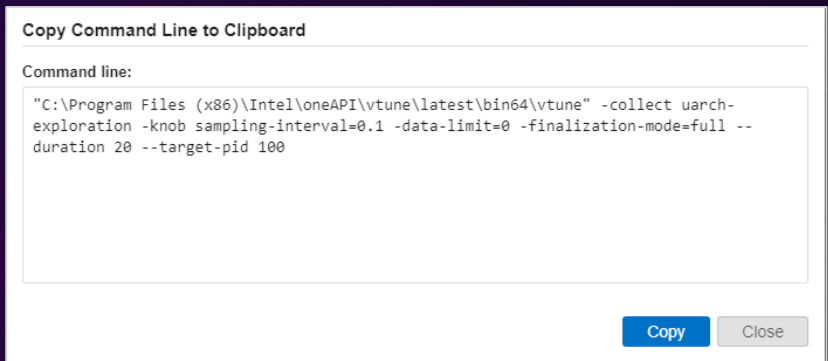
# Finding Architectural Issues



Configure VTune™ for microarchitecture analysis: Small sampling interval.



Can be run from command-line if preferred, minimal overheads. Embed into application using a hotkey?



Virtualization based security limits VTune™ collection, disable for collection of microarchitecture events.

# Useful VTune Metrics



Metric	Description
CPI Rate	Cycles per Instruction Retired, or CPI, how much time each executed instruction took, in units of cycles. Modern superscalar processors issue up to four instructions per cycle, suggesting a theoretical best CPI of 0.25.
Cache Bound	This metric shows how often the machine was stalled on L1, L2 and L3 caches. This metric also includes coherence penalties for shared data.
Contested Accesses	Contested accesses occur when data written by one thread is read by another thread on a different core. Examples of contested accesses include synchronizations such as locks, true data sharing such as modified locked variables, and false sharing.

# Microarchitecture by Core Type

Import results into VTune UI, use a custom grouping to sort thread activity into core type.

Bottom-up:

- Custom grouping (🔧)

Select grouping levels from:

- Basic Block
- Call Stack
- Class
- Code Location
- Frame
- Frame Domain
- Frame Duration Type
- Function Range
- Module
- OpenMP Barrier-to-Barrier Segment
- OpenMP Barrier-to-Barrier Segment Type
- OpenMP Region
- OpenMP Region Duration Type
- Package
- Packet Submission ID
- Packet Type
- Physical Core
- Source File
- Source Function
- Task Domain
- Task Type

Customize the grouping:

- Process
- Thread
- Core Type
- Logical Core
- Function

Maximum acceptable number of elements in the grouping is reached.

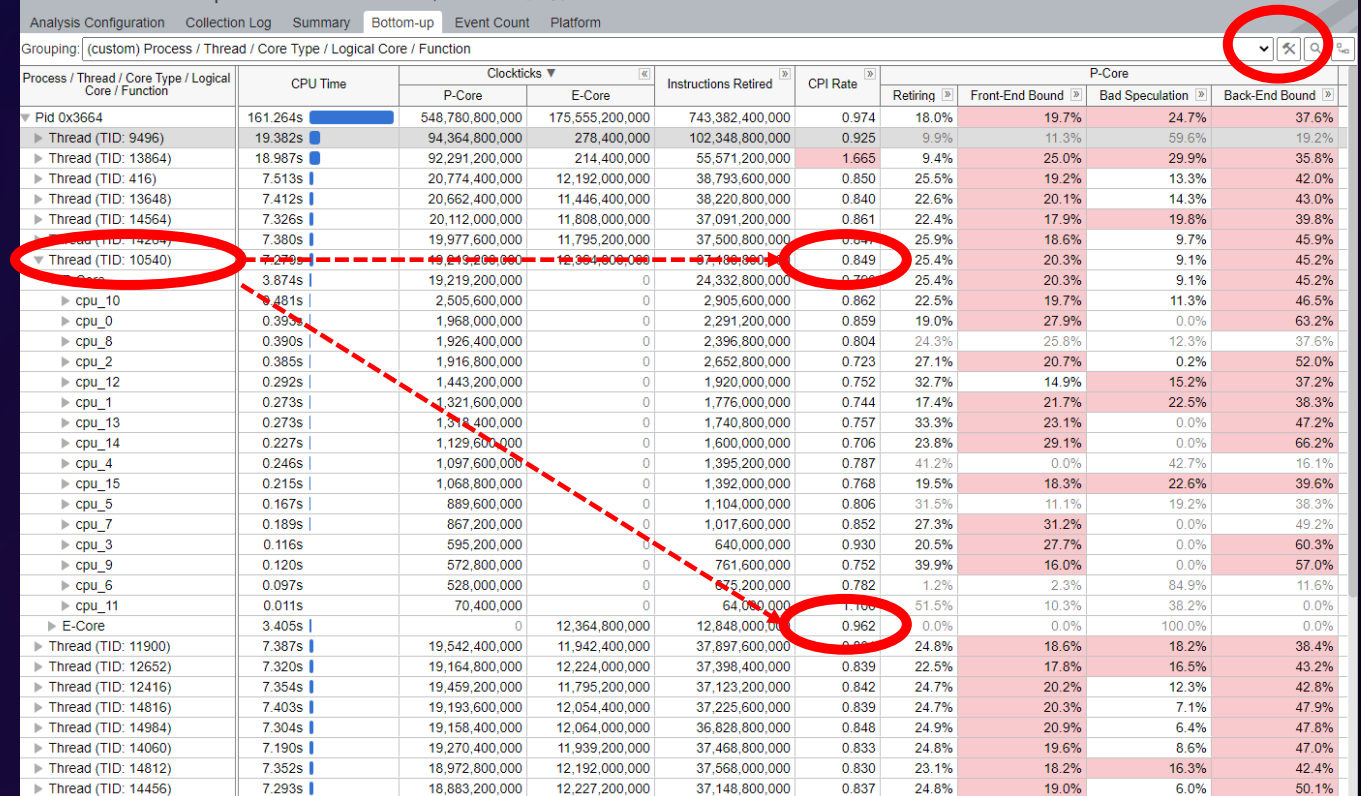
Buttons: Delete, Save, Cancel

Microarchitecture Exploration

Analysis Configuration | Collection Log | Summary | Bottom-up | Event Count | Platform

Grouping: (custom) Process / Thread / Core Type / Logical Core / Function

Process / Thread / Core Type / Logical Core / Function	CPU Time	Clockticks		Instructions Retired	CPI Rate	P-Core			
		P-Core	E-Core			Retiring	Front-End Bound	Bad Speculation	Back-End Bound
▼ Pid 0x3664	161.264s	548,780,800,000	175,555,200,000	743,382,400,000	0.974	18.0%	19.7%	24.7%	37.6%
▶ Thread (TID: 9496)	19.382s	94,364,800,000	278,400,000	102,348,800,000	0.925	9.9%	11.3%	59.6%	19.2%
▶ Thread (TID: 13864)	18.987s	92,291,200,000	214,400,000	55,571,200,000	1.665	9.4%	25.0%	29.9%	35.8%
▶ Thread (TID: 416)	7.513s	20,774,400,000	12,192,000,000	38,793,600,000	0.850	25.5%	19.2%	13.3%	42.0%
▶ Thread (TID: 13648)	7.412s	20,662,400,000	11,446,400,000	38,220,800,000	0.840	22.6%	20.1%	14.3%	43.0%
▶ Thread (TID: 14564)	7.326s	20,112,000,000	11,808,000,000	37,091,200,000	0.861	22.4%	17.9%	19.8%	39.8%
▶ Thread (TID: 14264)	7.380s	19,977,600,000	11,795,200,000	37,500,800,000	0.847	25.9%	18.6%	9.7%	45.9%
▶ Thread (TID: 10540)	7.279s	19,812,000,000	12,364,000,000	37,104,800,000	0.849	25.4%	20.3%	9.1%	45.2%
▶ Thread (TID: 11900)	7.387s	19,542,400,000	11,942,400,000	37,897,600,000	0.839	24.8%	18.6%	18.2%	38.4%
▶ Thread (TID: 12652)	7.320s	19,164,800,000	12,224,000,000	37,398,400,000	0.839	22.5%	17.8%	16.5%	43.2%
▶ Thread (TID: 12416)	7.354s	19,459,200,000	11,795,200,000	37,123,200,000	0.842	24.7%	20.2%	12.3%	42.8%
▶ Thread (TID: 14816)	7.403s	19,193,600,000	12,054,400,000	37,225,600,000	0.839	24.7%	20.3%	7.1%	47.9%
▶ Thread (TID: 14984)	7.304s	19,158,400,000	12,064,000,000	36,828,800,000	0.848	24.9%	20.9%	6.4%	47.8%
▶ Thread (TID: 14060)	7.190s	19,270,400,000	11,939,200,000	37,468,800,000	0.833	24.8%	19.6%	8.6%	47.0%
▶ Thread (TID: 14812)	7.352s	18,972,800,000	12,192,000,000	37,568,000,000	0.830	23.1%	18.2%	16.3%	42.4%
▶ Thread (TID: 14456)	7.293s	18,883,200,000	12,227,200,000	37,148,800,000	0.837	24.8%	19.0%	6.0%	50.1%



# Relative Thread Perf. by Core Type

Thread	Thread ID	P-Core CPI	e-Core CPI	P-Core Million Instruction s/Second	E-Core Million Instruction s/Second	Relative instructions per second
Render	9496	0.93	2.18	5210.81	1664.37	0.32
Game	13964	1.67	2.68	2894.89	1350.75	0.47
Worker 1	416	0.79	0.99	6140.13	3663.97	0.60
Worker 2	13648	0.79	0.95	6093.55	3826.64	0.63
Streamer	14092	1.38	1.25	3487.70	2907.63	0.83
Audio	13640	1.43	1.23	3370.63	2935.93	0.87

Frequency\*  
 $\frac{1}{CPI}$

Threads more efficient on P-Cores

Threads slightly more efficient on P-Cores

Memory limited, core type doesn't matter



# Relative Thread Perf. by Core Type

Potentially unfair to compare P- and E-Cores: E-Cores are lowering sibling activity.  
(see slide 8 - Hyper-Threading Recap)

*****SMT Statistics*****			*****SMT Statistics*****		
	8C/16T			Hybrid	
Core ID	Both Siblings Idle	Both Siblings Active	Core ID	Both Siblings Idle	Both Siblings Active
	Percentage	Percentage		Percentage	Percentage
LP0 & LP1	44.94	29.75	LP0 & LP1	45.44	20.34
LP2 & LP3	43.19	34.73	LP2 & LP3	31.47	23.08
LP4 & LP5	42.18	36.61	LP4 & LP5	43.44	22.66
LP6 & LP7	43.11	36.66	LP6 & LP7	45.92	23.2
LP8 & LP9	1.99	45.28	LP8 & LP9	11.46	28.22
LP10 & LP11	44.01	37.17	LP10 & LP11	46.6	23.71
LP12 & LP13	2.14	45.93	LP12 & LP13	13.15	27.92
LP14 & LP15	43.63	35.79	LP14 & LP15	46.38	23.17
Average		37.74			24.0375

	Thread ID	Hybrid P-Core CPI	Symmetric P-Core CPI	Hybrid vs Symmetric
Render	9496	0.925	0.991	1.07
Game	13964	1.665	1.714	1.03
Worker 1	416	0.785	0.888	1.13
Worker 2	13648	0.791	0.904	1.14
Streamer	14092	1.382	1.415	1.02
Audio	13640	1.43	1.395	0.98

- 33% reduction in SMT work.
- 3-14% improvement in P-Core SMT.



# CASE STUDIES

A hand is holding an Intel ARC graphics card in front of a server rack. The server rack is filled with various components, including cables and a cooling fan. The Intel ARC logo is visible on the graphics card. The background is dark with blue and purple lighting.

Introduction

OS Scheduling

Data Capture

Case Studies

Closing Thoughts

# Case Study Background

- Titles used are anonymous
- All data taken from titles un-optimised for Hybrid
- Data gathered during platform validation
- All titles give a good user experience on Hybrid
- Used purely to illustrate OS behaviour

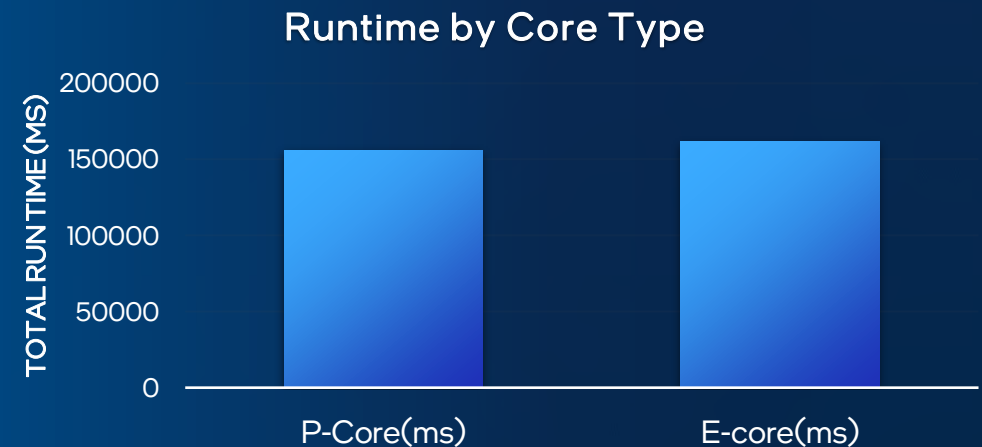
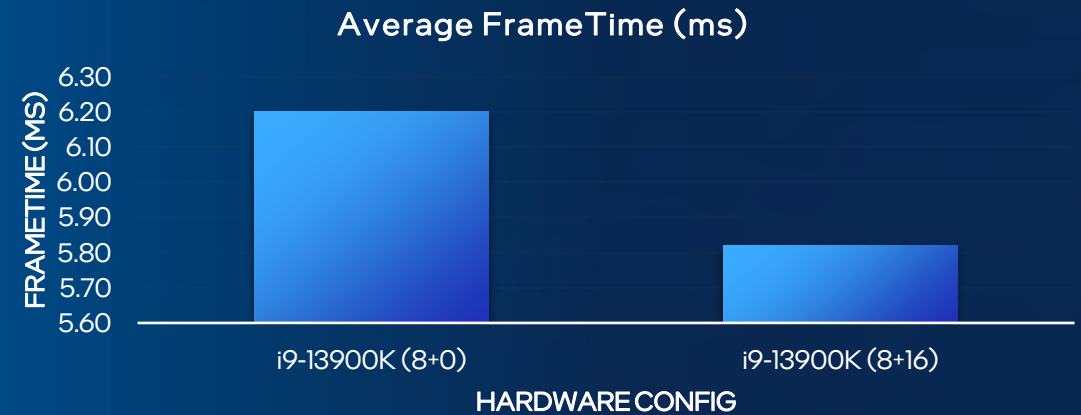
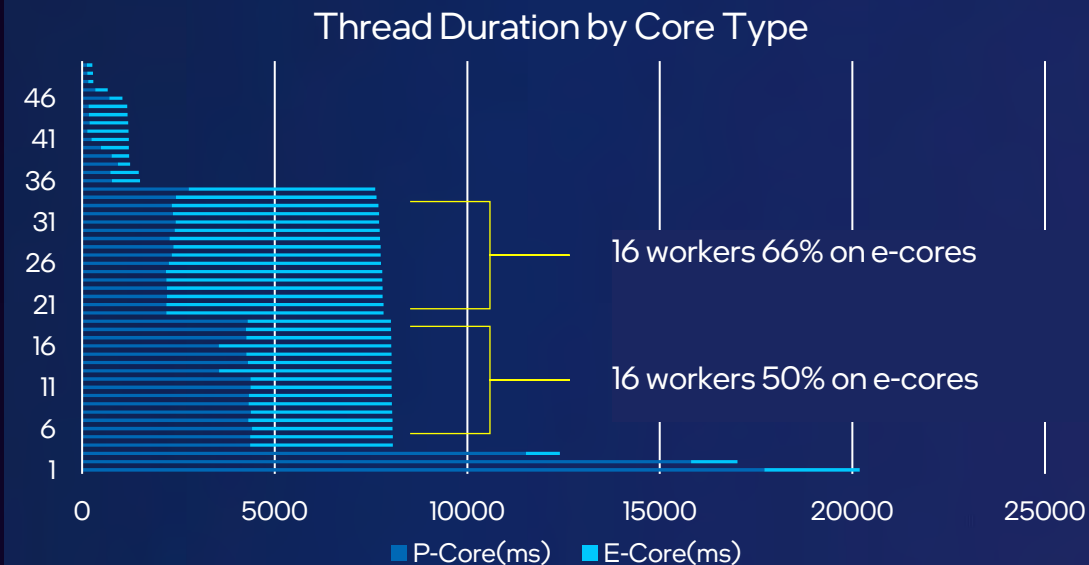


# Case Study 1: Worker Threads on E-Cores

## Problem statement:

Title scales on Hybrid but...

- Very high E-Core utilisation
- Critical threads on E-Cores



#### Disclaimer

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Configurations used for test and this perf data: Intel® i9-12900K + NVIDIA 3090
- All testing was performed at Intel® Munich. Numbers may differ based on actual hardware used and/or based on how the benchmark is written. Intel® makes no guarantee on the specific numbers and it is intended for providing reference
- The above is for reference and work in progress data and software

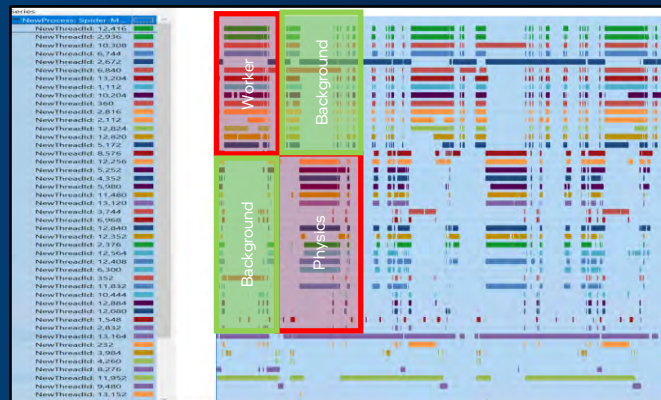
# Case Study 1: Non-Hybrid Behaviour



## Engine created three thread-pools

- Worker threads \* X
- Background threads \* X
- Physics threads \* X

Worker threads run serially with physics, background fills in idle time.

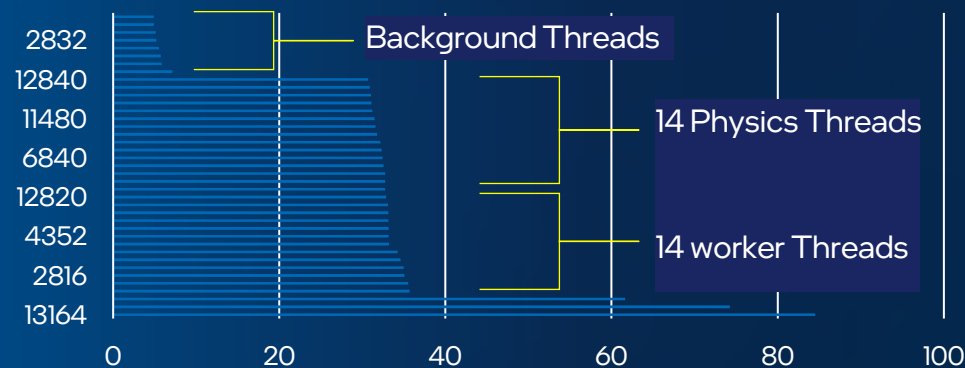


## Managed by thread priority.

Thread concurrency of 16:

- 14 Worker Threads + 2 Main threads

%TotalCPU



# Case Study 1: Over reliance on Priority

Thread 5,252 sits in a ready state while 12,416 is running

NewThreadId	CPU Usage (ms)	Ready (ms)	
13164	26262.94	170.62	15
2672	21733.46	8170.25	9
11952	18384.48	52.02	15
13204	10747.57	445.80	11
1112	10723.61	445.45	11
10308	10716.05	604.76	11
12416	10691.85	514.57	11
2816	10687.33	519.11	11
2936	10521.04	739.96	11
12820	9885.51	1284.05	11
5172	9859.60	1322.32	11
10204	9853.41	1363.99	11
360	9842.28	1402.56	11
2112	9831.45	1436.90	11
12824	9830.03	1324.76	11
6840	9815.88	1403.71	11
6744	9718.37	1439.86	11
12408	9993.18	9532.86	9
11832	9956.15	9536.22	9
5252	9947.14	9480.11	9
4352	9943.25	9549.69	9
12256	9939.59	9580.09	9
13120	9933.24	9538.27	9
12564	9601.82	9952.32	9
12352	9550.92	10004.38	9
6300	9492.72	10081.98	9
11480	9482.79	10129.44	9
5980	9407.38	10187.29	9
8576	9398.60	10206.77	9
2376	9344.28	10255.47	9
12840	9334.23	10271.34	9
232	2122.71	4278.37	
352	2026.23	4438.73	
1548	1780.68	132.23	



5,252

12,416

On symmetric system, priority 9 threads spend 50% of their time in Ready Queue





# Case Study 1: Priority Does Not Block Background Thread on Hybrid

- Priority 9 Ready time drops 10x.
- Low priority threads don't have to wait.

P9 Threads

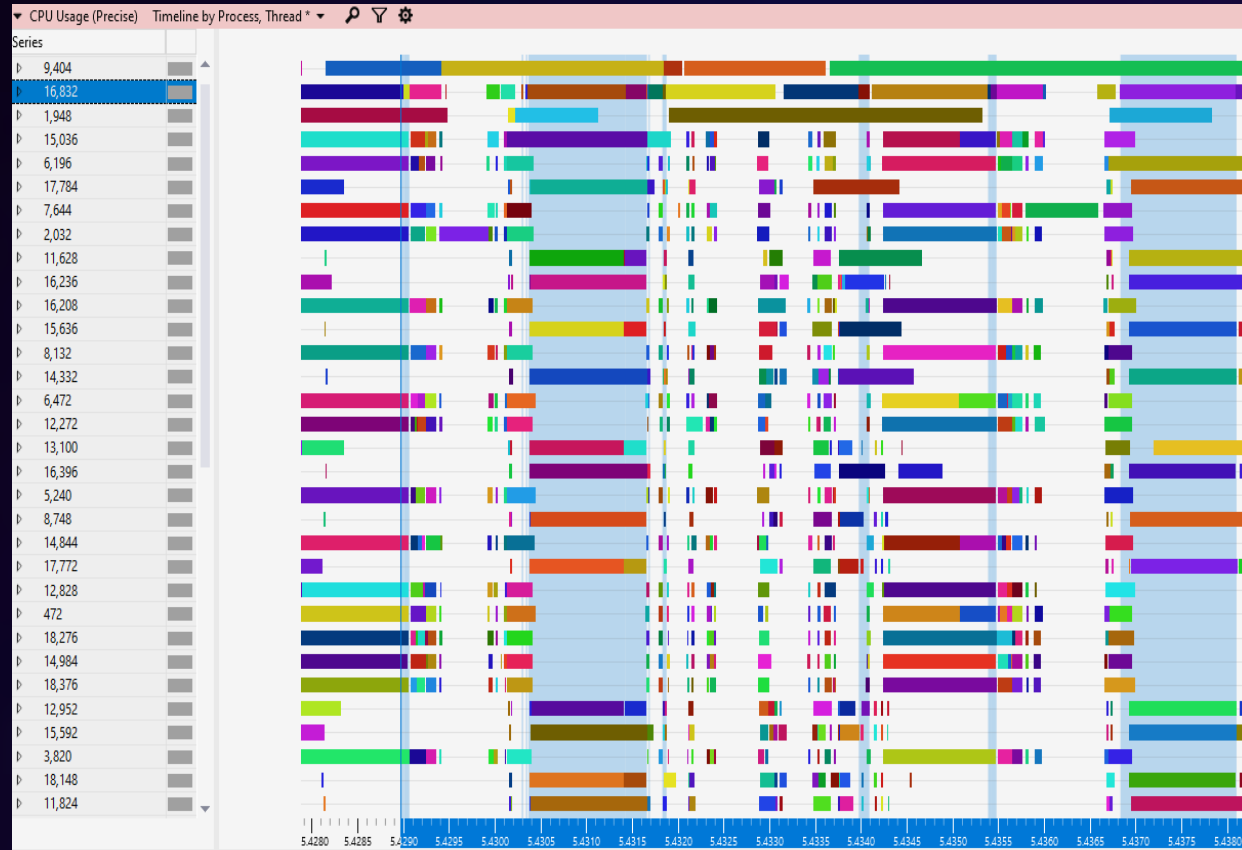
NewThreadId	CPU Usage (ms)	Ready (ms)	Waits (ms)
9404	29561.60	407.07	19.29
16832	26209.49	15.69	6116.78
1948	18290.85	152.10	11550.94
472	11923.43	376.37	17822.46
:	:	:	:
6196	11894.83	390.61	17877.16
17784	11625.97	1006.57	17534.10
15592	11622.22	1006.30	17515.31
11824	11615.71	1020.29	17517.20



80%+ CPU Utilisation



# Case Study 1: Summary

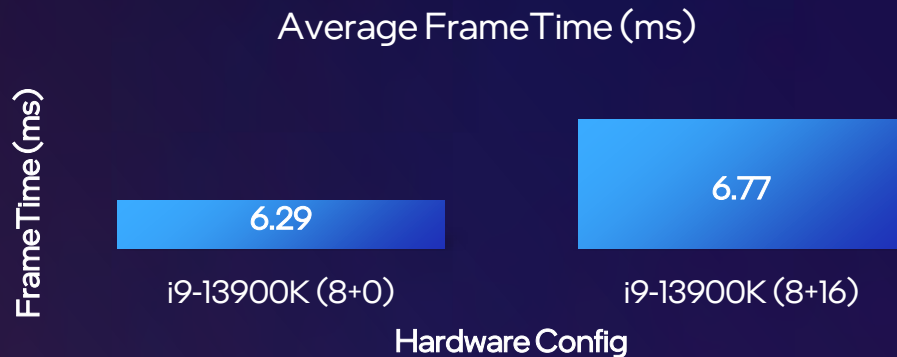


Thread 16,832 running on e-core

- Low priority work, runs in parallel with high priority work.
- High priority, long running threads, run on E-Cores when previous lower priority work is already in-flight on P-Cores.
- Could defer scheduling of priority 10 threads until after priority have started running.
- Could move background threads on to EcoQos

# Case Study 2: Unclear Critical Path / Poor Multi-Threaded Scaling

8% slower with E-Cores enabled



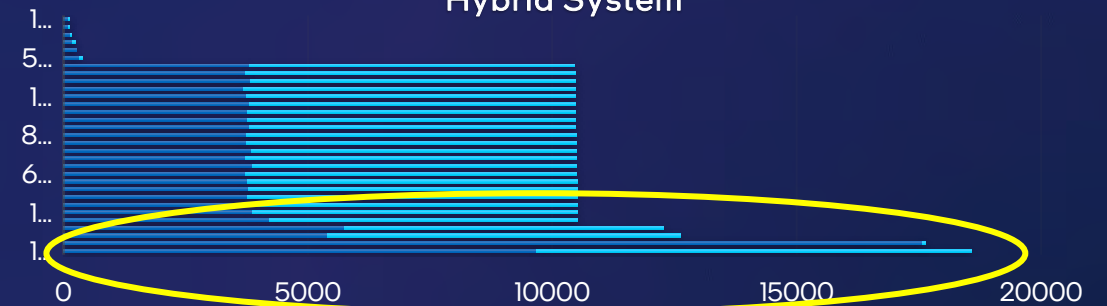
Two issues to investigate:

- Long running thread on E-Cores 50% of the time
- Thread-pool wall time increasing on Hybrid

Thread Duration by Core Type  
Symmetric System



Thread Duration by Core Type  
Hybrid System



Thread creation based on logical processor count

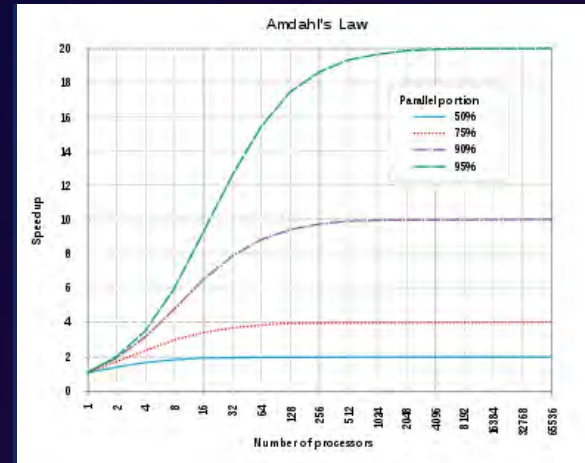
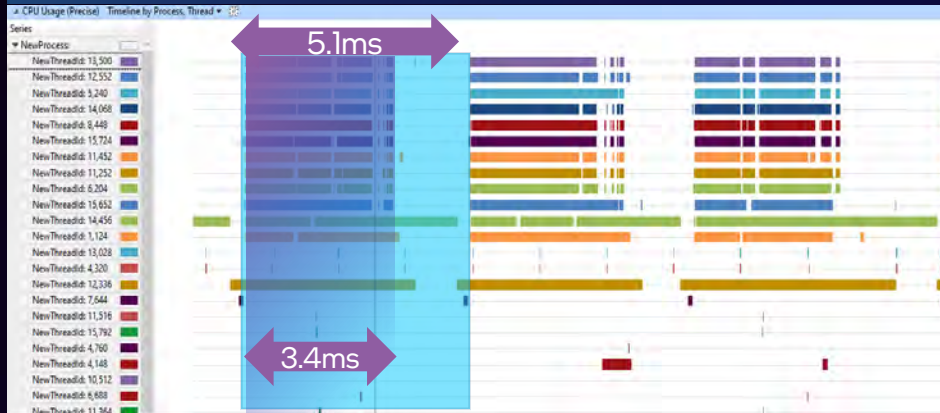
## Disclaimer

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Configurations used for test and this perf data: Intel® i9-12900K + NVIDIA 3090
- All testing was performed at Intel® Munich. Numbers may differ based on actual hardware used and/or based on how the benchmark is written. Intel® makes no guarantee on the specific numbers and it is intended for providing reference
- The above is for reference and work in progress data and software

# Case Study 2: Poor Multi-Threaded Scaling



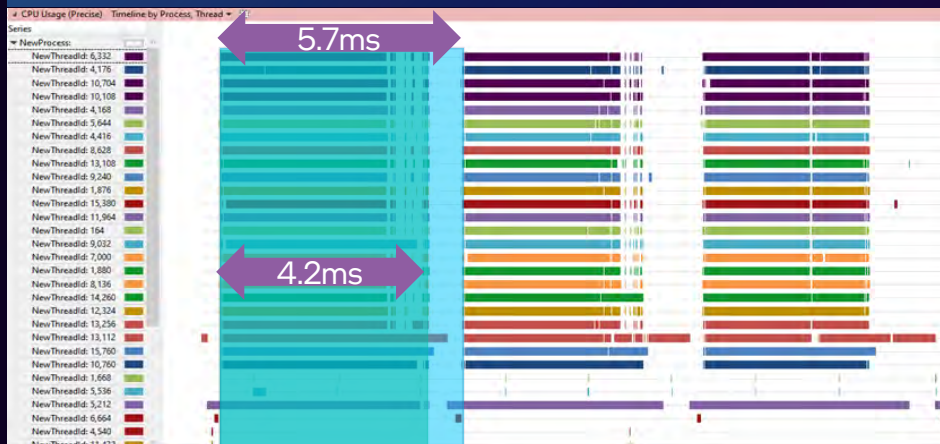
## Non-Hybrid System



Amdahl's Law: 2x increase in cores should halve wall time.

- But: 22 worker threads is slower than 10 worker threads.

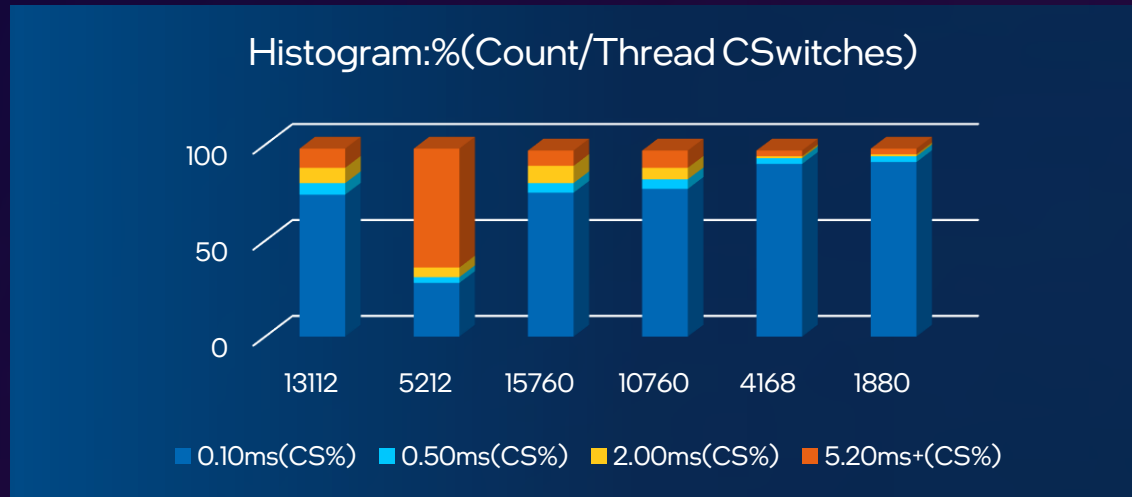
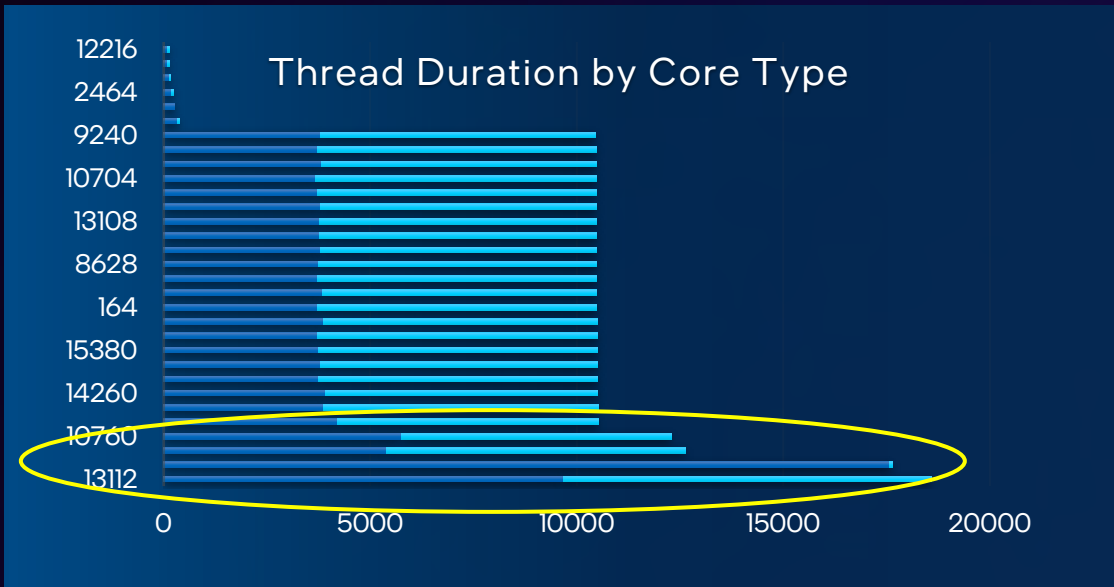
## Hybrid System



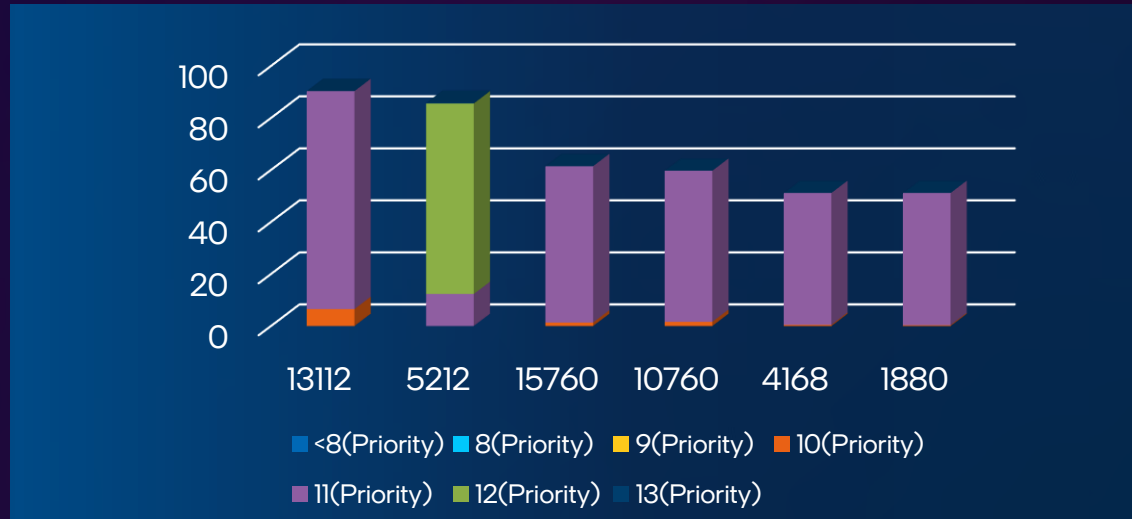
Additional threads show high CPI on hybrid. L3 boundedness increased by 2x.

	Hybrid	Non Hybrid
metric_CPU operating frequency(GHz)	5.2460	5.3889
metric_CPI	2.3365	1.5610
metric_TMA_Backend_Bound(%)	77.0914	54.9688
metric_TMA_...Memory_Bound(%)	66.0355	42.5052
metric_TMA_...L3_Bound(%)	48.6328	23.5164

# Case Study 2: Unclear Critical Path



Thread 5,212 is high priority and stays on P-Cores.  
 Thread 13,112 looks the same as worker threads from the OS level.  
 Same priority 11 as workers.  
 80% short run time on thread wake up.

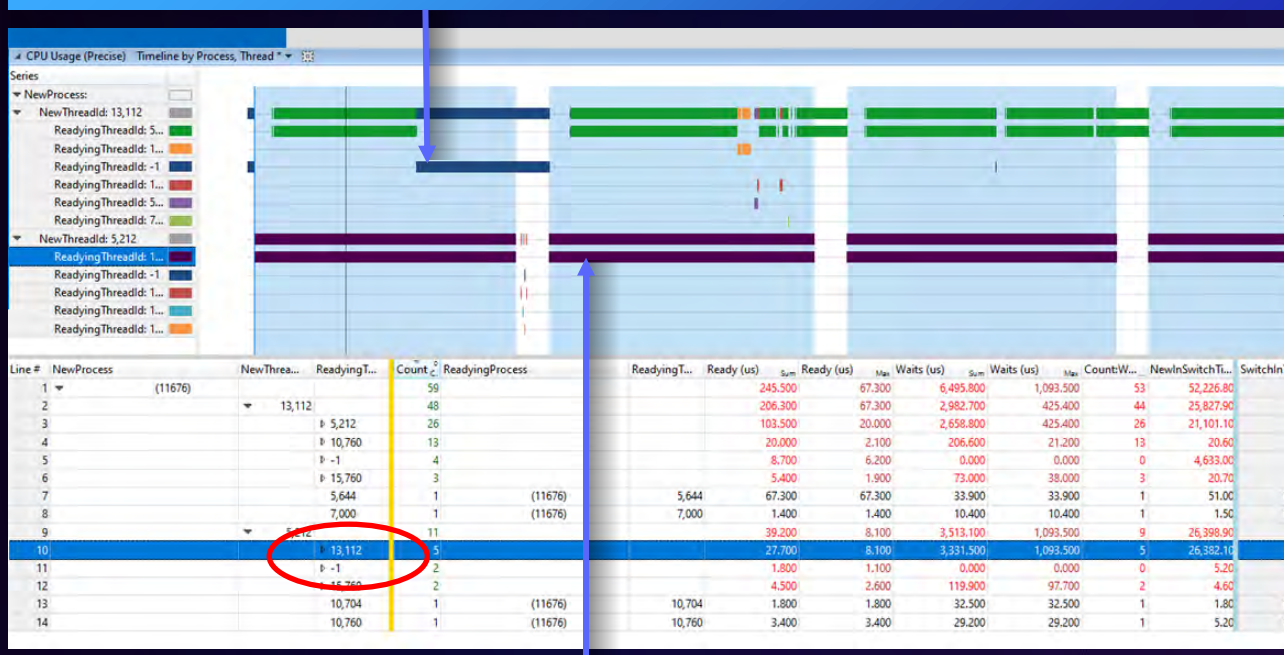




# Case Study 2: Summary



Thread 13,112 context switches mid frame



Thread 5,212 waits on thread 13,112



Two long running threads with hard dependency between them context switch while being highly subscribed → high chance to schedule on an E-Core. Time spent on E-Core is part of the critical path.

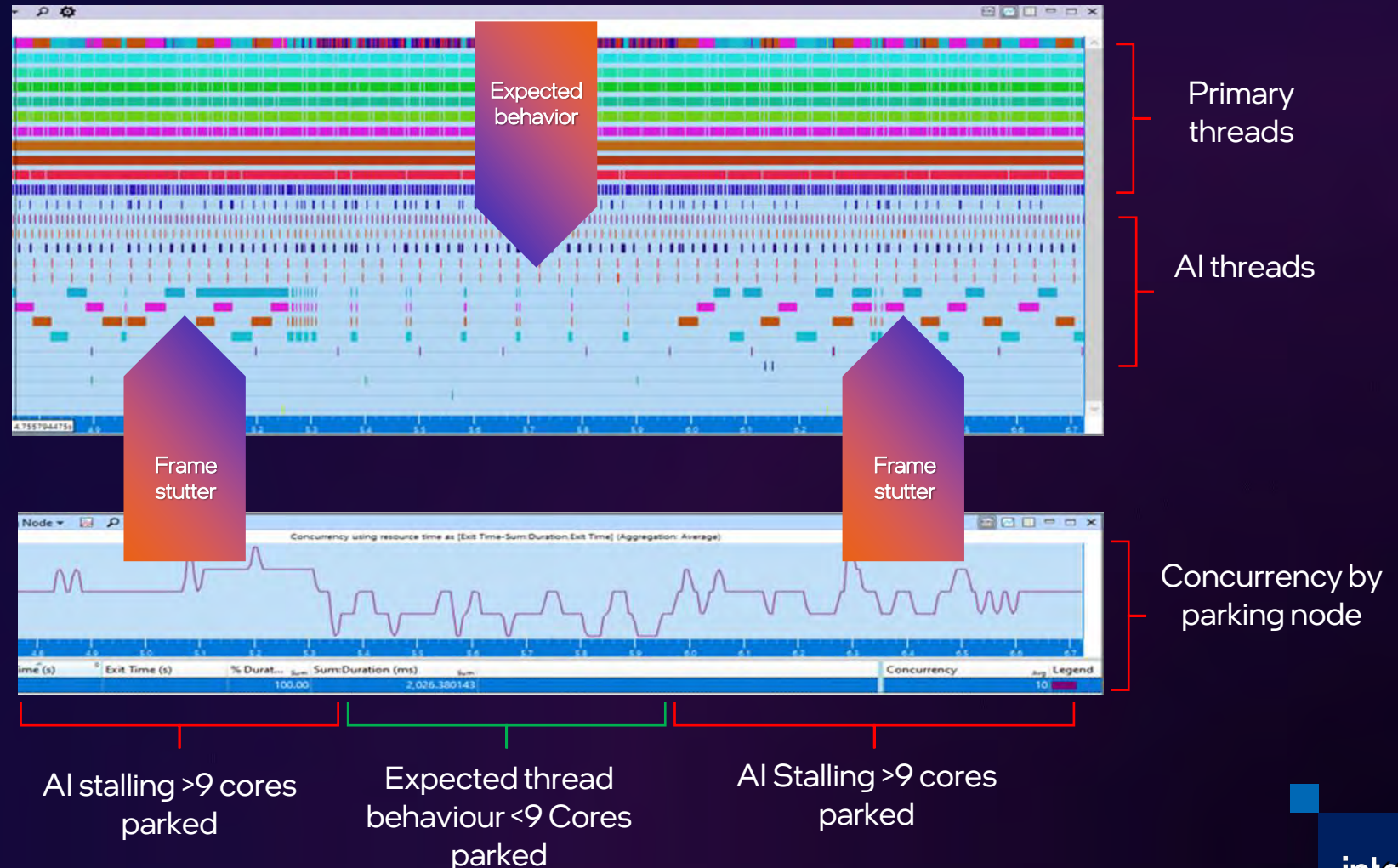
- Increase priority of critical path thread
- Reduce number of worker threads to reduce memory contention

# Case Study 3: Erratic Behaviour Over Time



When processor selection goes bad...

- Application stutters during gameplay
- AI called on separate threads decoupled from primary task system
- Threads doing AI change behaviour over time
- Stutter coincides with higher core parking and long running AI threads



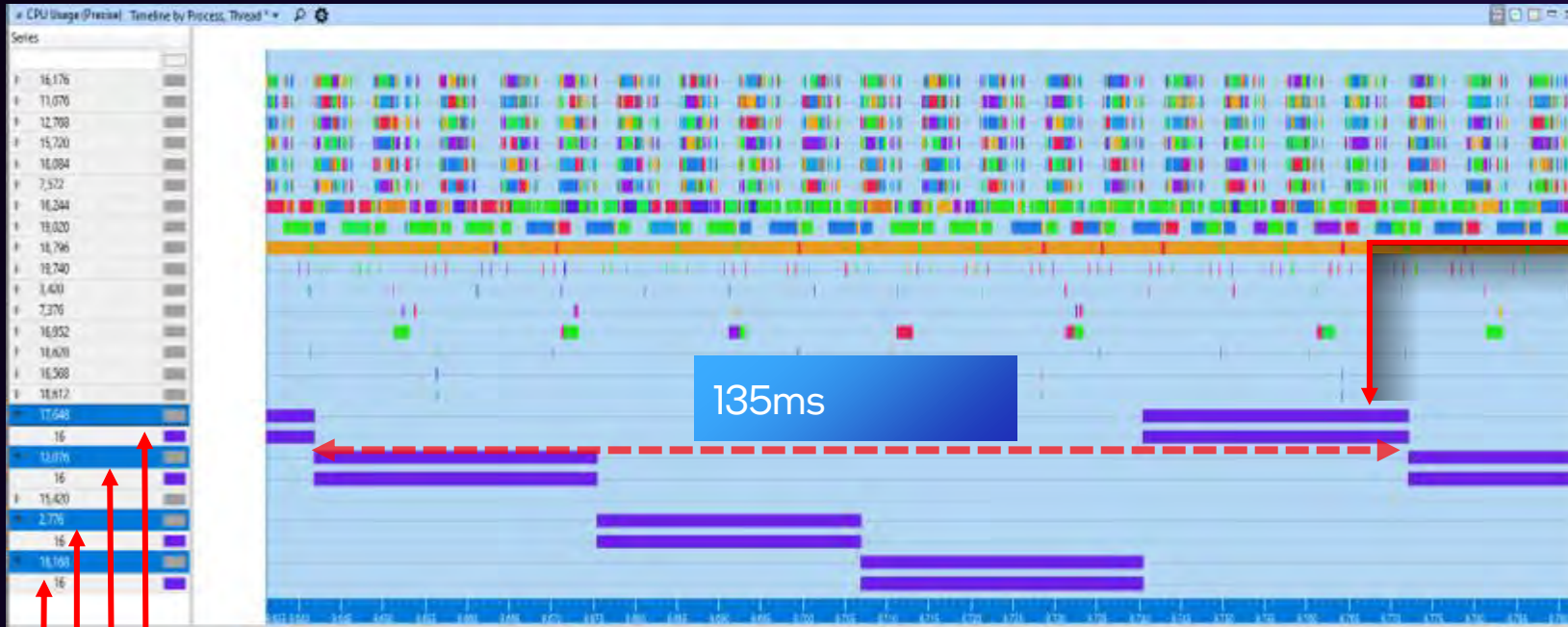
AI stalling >9 cores parked

Expected thread behaviour <9 Cores parked

AI Stalling >9 cores parked



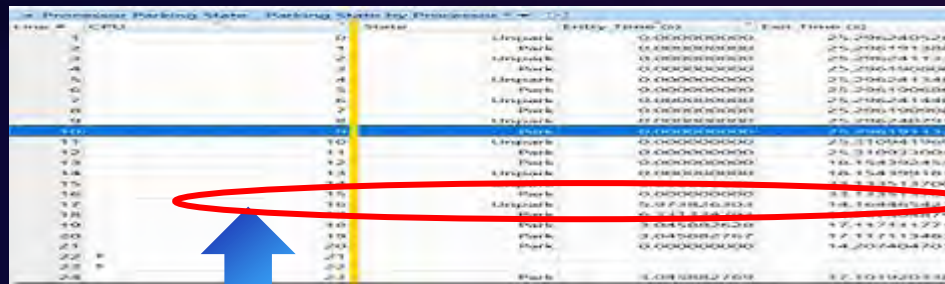
# Case Study 3: OS Forced Serialization



Why 35ms?

Thread switch at quantum end → threads never yield

Producer and consumer threads serialised onto core 16



Games is affinitizing to e-cores for AI  
Most E-Cores are parked Only core 16 is fully unparked

Line #	New Process	New Threa...	New Wait Reason	CPU
17		18,612	DelayExecution	
18		2,776	WrQuantumEnd	16
19		17,648	WrQuantumEnd	16
20				
21				
22		8,384	DelayExecution	0
23		15,420	UserRequest	8
24		18,168	WrQuantumEnd	16
25		12,076	WrQuantumEnd	16

# Case Study 3: Summary

Remember this??



Frame rate stutter linked to core parking

Thread's Ideal Processor was outside the thread affinity mask.

- Therefore used last used core (16)

Data contention resulted in blocked thread progress until quantum end

- Fixed with `SetThreadIdealProcessor`
- Removed data contention



# CLOSING THOUGHTS



Introduction

OS Scheduling

Data Capture

Case Studies

Closing Thoughts

# Hybrid CPU Best Practices

## Profile your workload

- Use `QueryPerformanceCounter()` for micro-benchmarking
- Use Intel® VTune™ Profiler for in-depth CPU performance analysis

Avoid pinning threads to a single logical processor

## Don't oversubscribe your thread pool

- Don't use hyperthread cores if your workload can't benefit from hyper-threading
- Avoid unnecessary context switches and cache flushes

Avoid scheduling lower priority tasks on the same cores as your critical path

## Use Quality of Service APIs for OS and Intel® Thread Director optimizations

- QoS APIs can be used in combination with Static Partitioning APIs based on application architecture

Understand how your middleware uses threads

## Avoid static partitioning; allow cores to steal work from other cores

- Work stealing allows idle threads to take tasks from cores that may be overworked, increasing throughput



Thank you

# Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at [www.intel.com/PerformanceIndex](http://www.intel.com/PerformanceIndex) (graphics and accelerators).

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. No product or component can be absolutely secure.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Intel technologies may require enabled hardware, software or service activation.

All product plans and roadmaps are subject to change without notice.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Statements that refer to future plans or expectations are forward-looking statements. These statements are based on current expectations and involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in such statements. For more information on the factors that could cause actual results to differ materially, see our most recent earnings release and SEC filings at [www.intc.com](http://www.intc.com).

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.





## Optimizing Software for x86 Hybrid Architecture

White Paper

---

October 2021

Revision 1.0

Link



Document Number: 348851-001US

intel®