Programming for Hybrid: Untangling your Threads

Leigh Davies

Senior Game/Graphics Application Engineer

03/2023

INTRODUCTION

- Introduction
- OS Scheduling
- Data Capture
- Case Studies
- Closing Thoughts

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

Combines Performance Cores and Efficient Cores

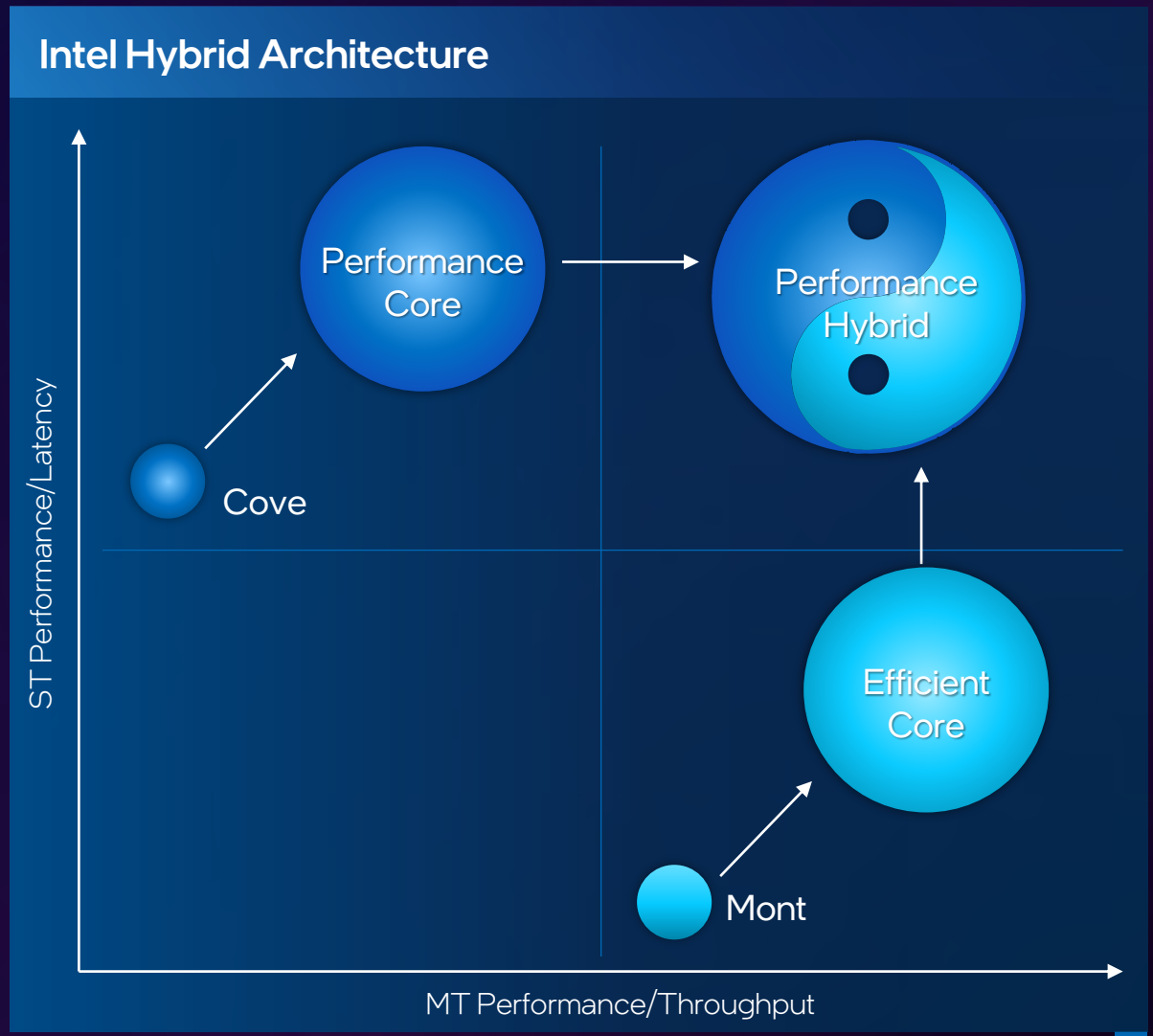
- Two core types with different power and performance characteristics
- Both core types have the same ISA support
  - No AVX512, TSX
  - New AVX-VNNI , UMWAIT/TPAUSE

Performance Cores

- Concentrate on single and limited threading scenarios
- Performance intensive

Efficient Cores

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



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

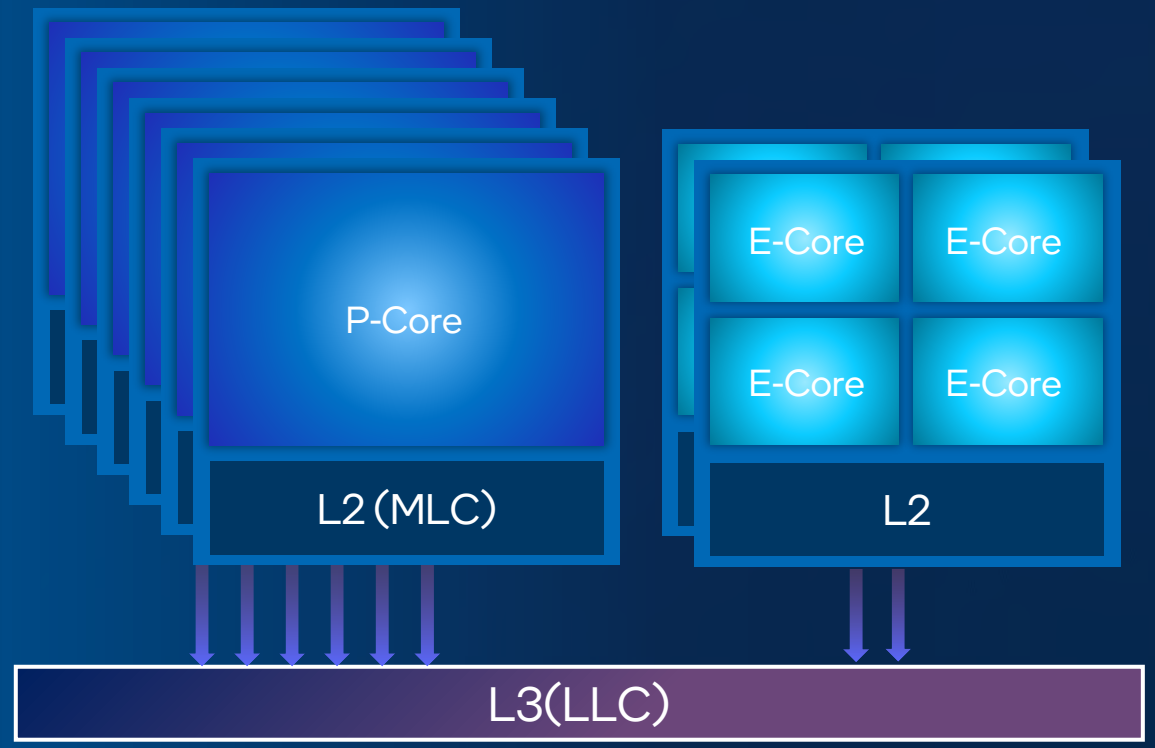
Performance Cores

- 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

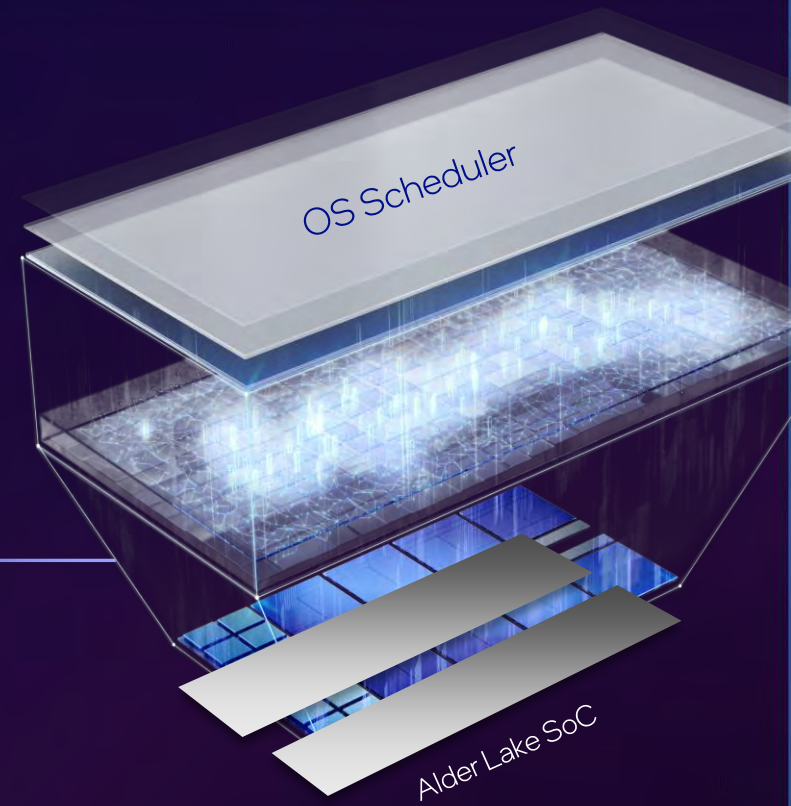
Intel® Thread Director (HGS+)

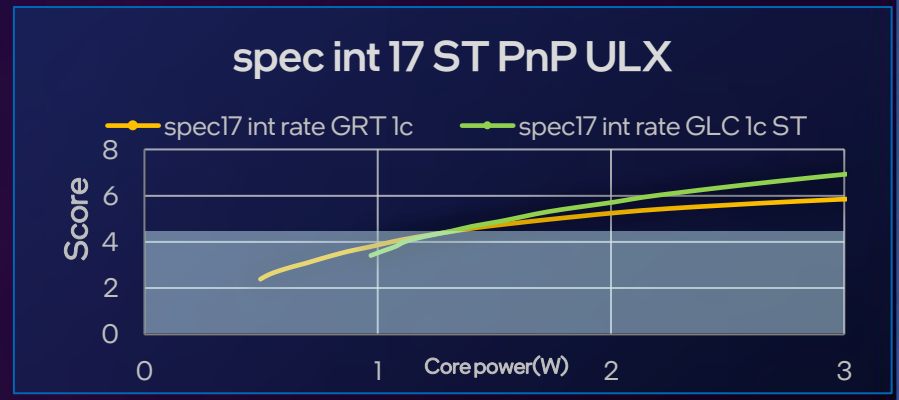
Hardware unit
Intelligence built directly into the core

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



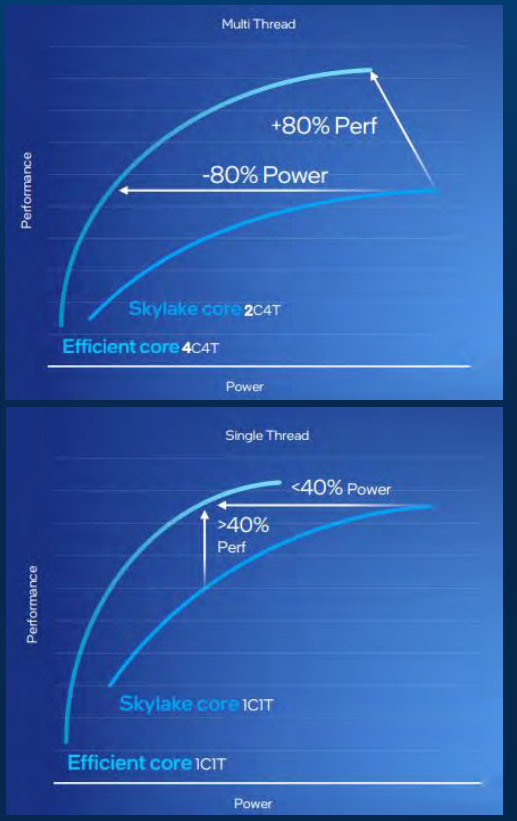


Intel® Thread Director requires 12th gen Intel® Core™ performance hybrid architecture and OS enablement. Available features and functionality will vary by OS

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	✓ 5.8 GHz
L1 D$ / I$	✓ 48 / 32 KB	✓ 48 / 32 KB	✓ 48 / 32 KB
L2 U$	✓ 512 KB	✓ 1280 KB	✓ 2048 KB
L3 U$	✓ 16 MB	✓ 30 MB	✓ 36 MB
E-cores	✓ 0	✓ 8	✓ 16
L2 U$ (4x E-core)	✓ N/A	✓ 2048 KB	✓ 4096 KB
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

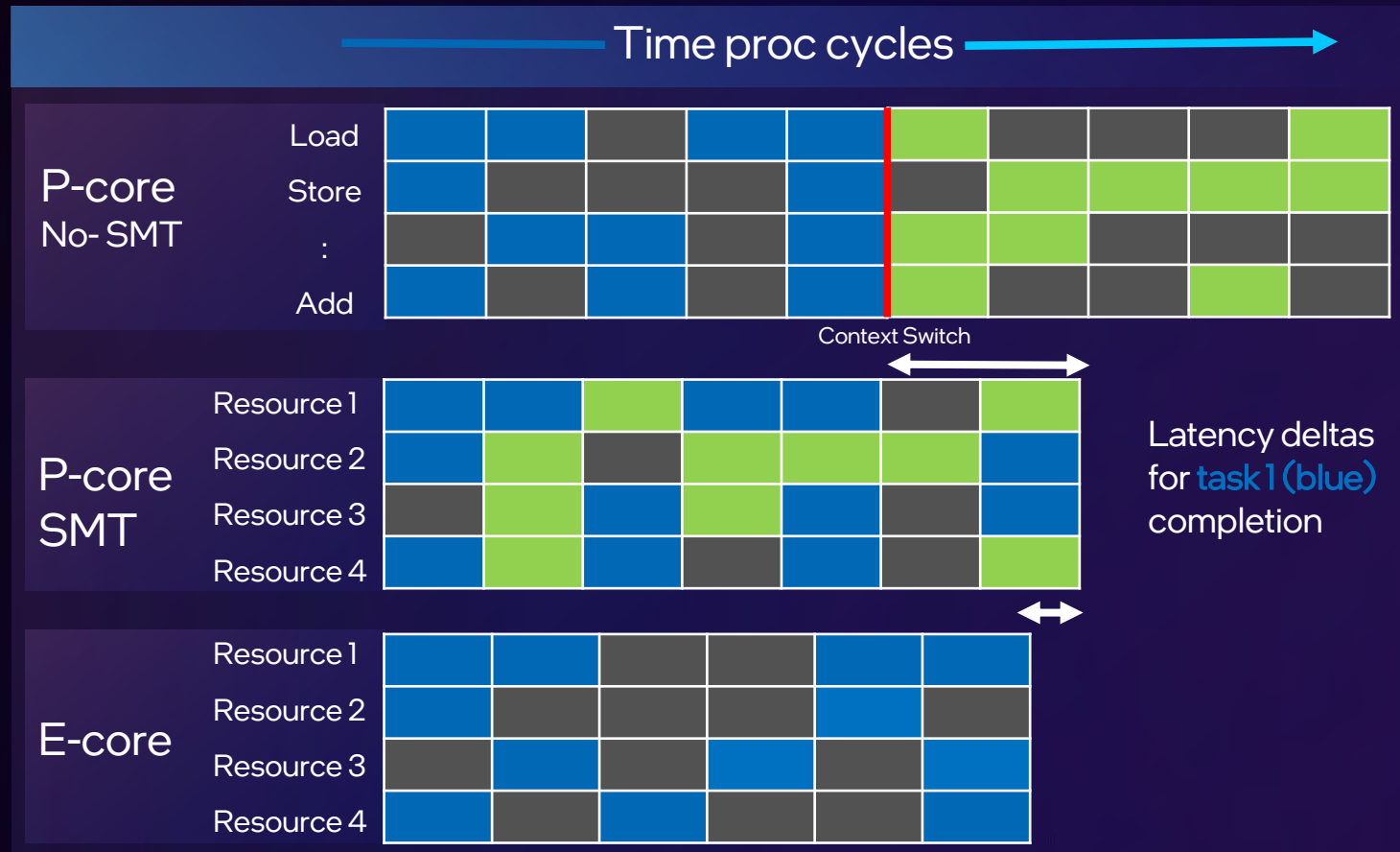
- SPECrate2017_int_base estimates using an open source compiler, iso-binary For workloads and configurations visit www.intel.com/ArchDay21claims. Results may vary.

intel

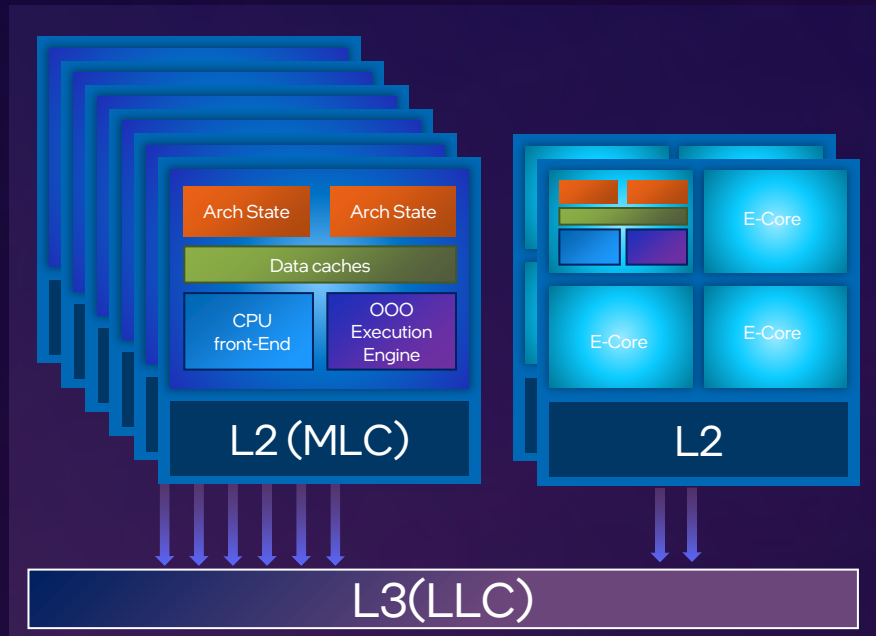
Hyper-Threading Recap

Important for later in the talk.....

Hyper-Threading (Simultaneous Multi-Threading)



Latency deltas for task 1 (blue) completion



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."

Each box represents a processor execution unit

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;

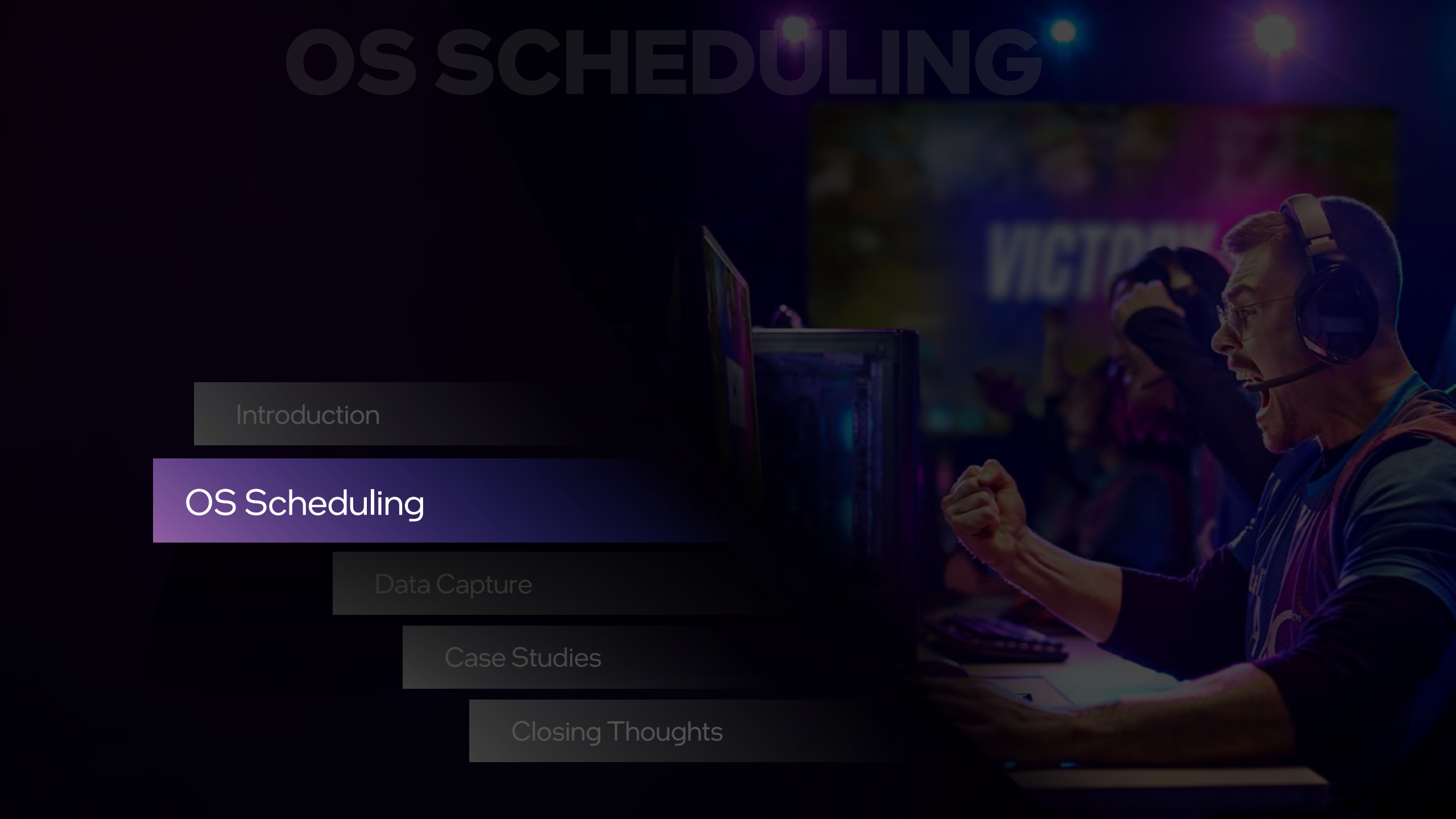
Preferred Enumeration method:
GetLogicalProcessorInformationEx()

- struct _PROCESSOR_RELATIONSHIP:
- Field: EfficiencyClass; << Higher mean more perf
- Note: This is relative to other logical processors in the system.
- For 12th/13th 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* procInfos = (pSLPI_EX) &buffer[i];

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

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

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

OS SCHEDULING



Introduction

OS Scheduling

Data Capture

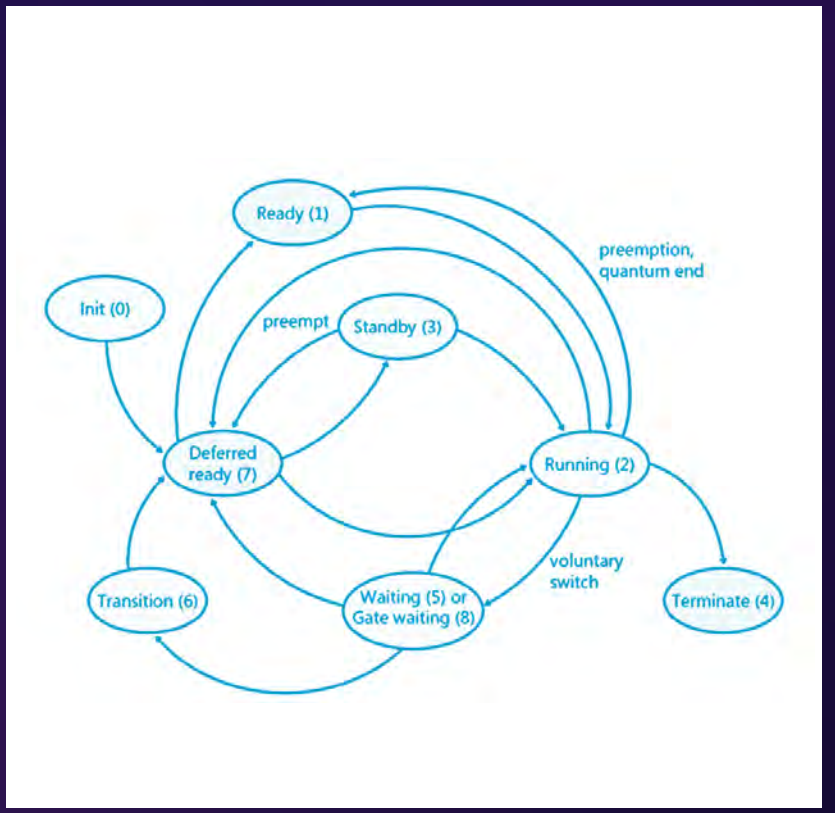
Case Studies

Closing Thoughts

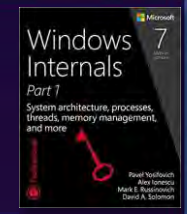
Thread Scheduling

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



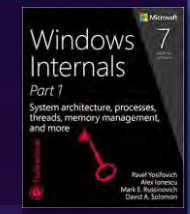
Scheduling Priorities



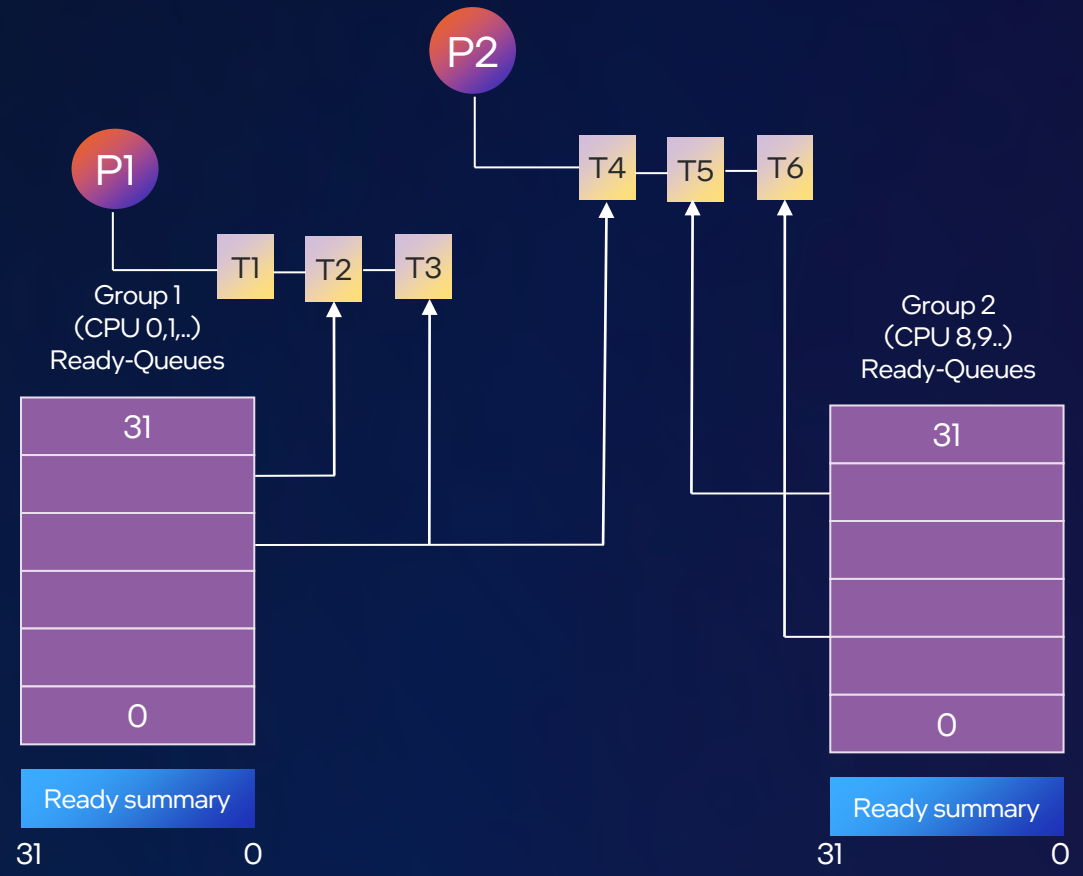
- A process has only a single base priority value
- Each thread has two priority values: current (Dynamic) and base
- Scheduling decisions are made based on the current priority
  - The system under certain circumstances increases the priority of threads in the dynamic range (1 through 15) for brief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



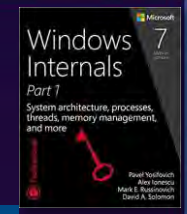
Priority driven, preemptive

- 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

Current Hybrid systems have up to 8 LPs per group

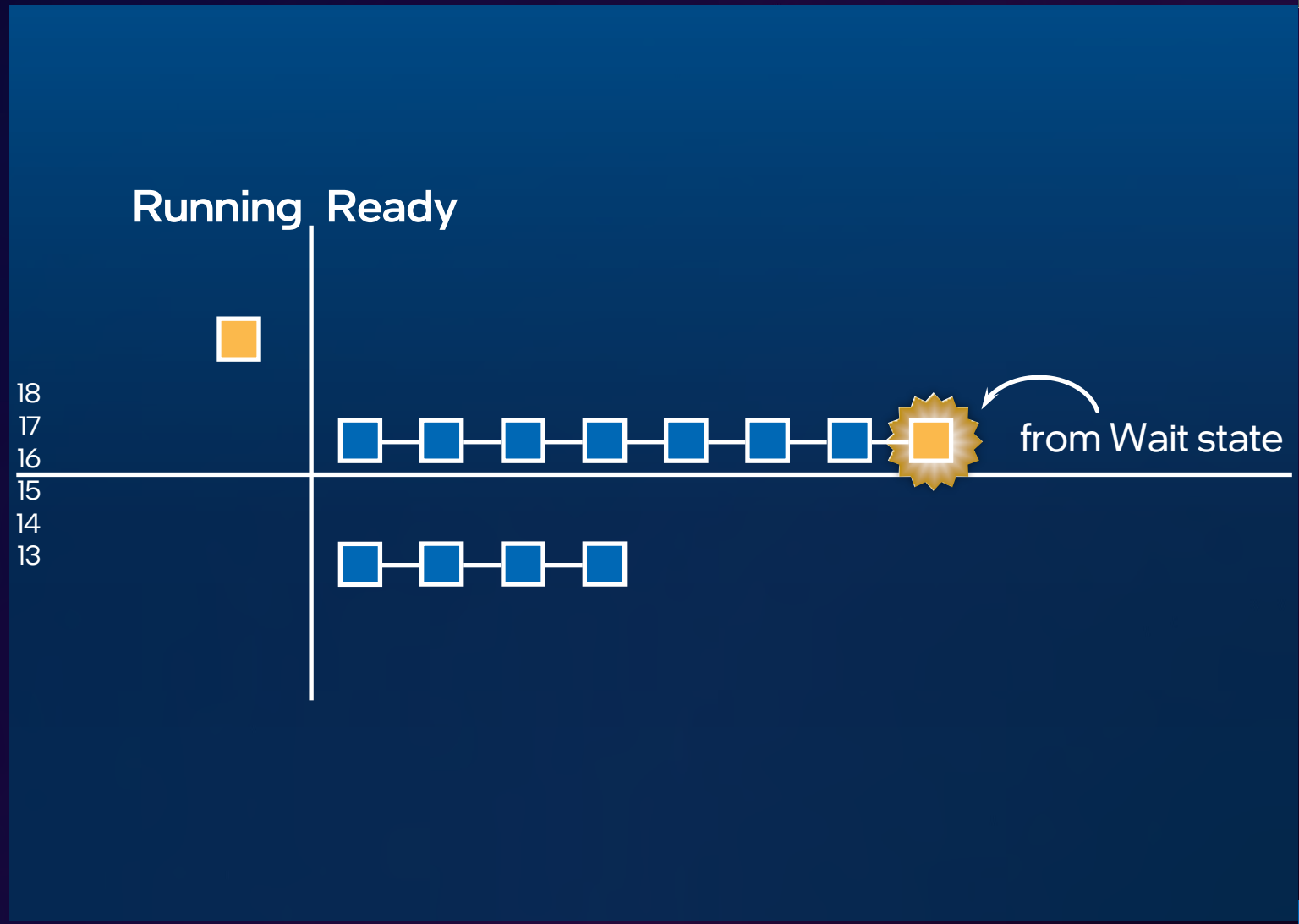
intel

Scheduling Scenarios: On wake



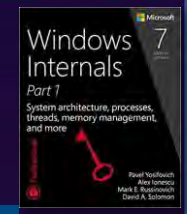
On wake

- 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 >=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



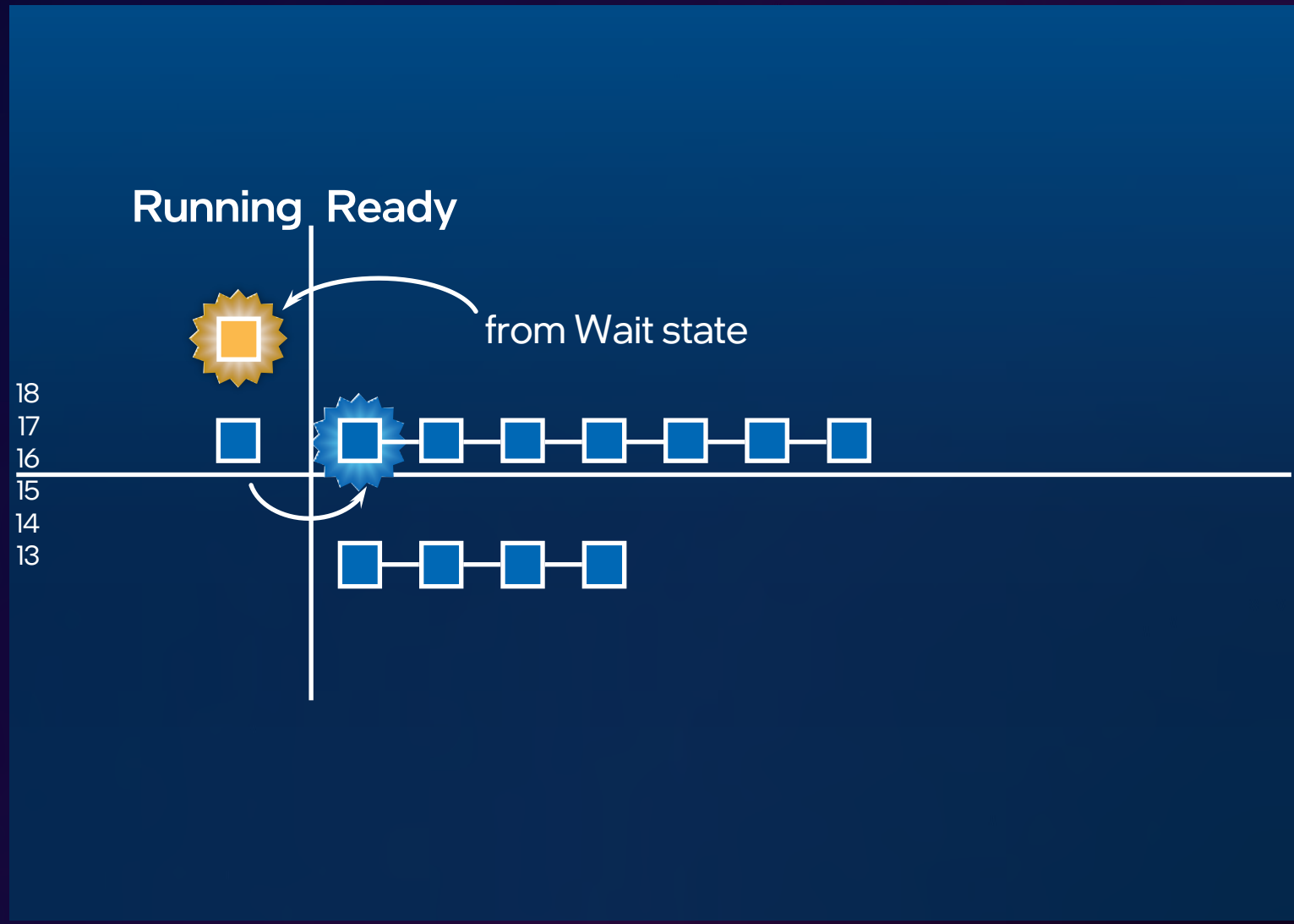
intel

Scheduling Scenarios: Preemption



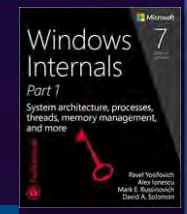
Preemption

- 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



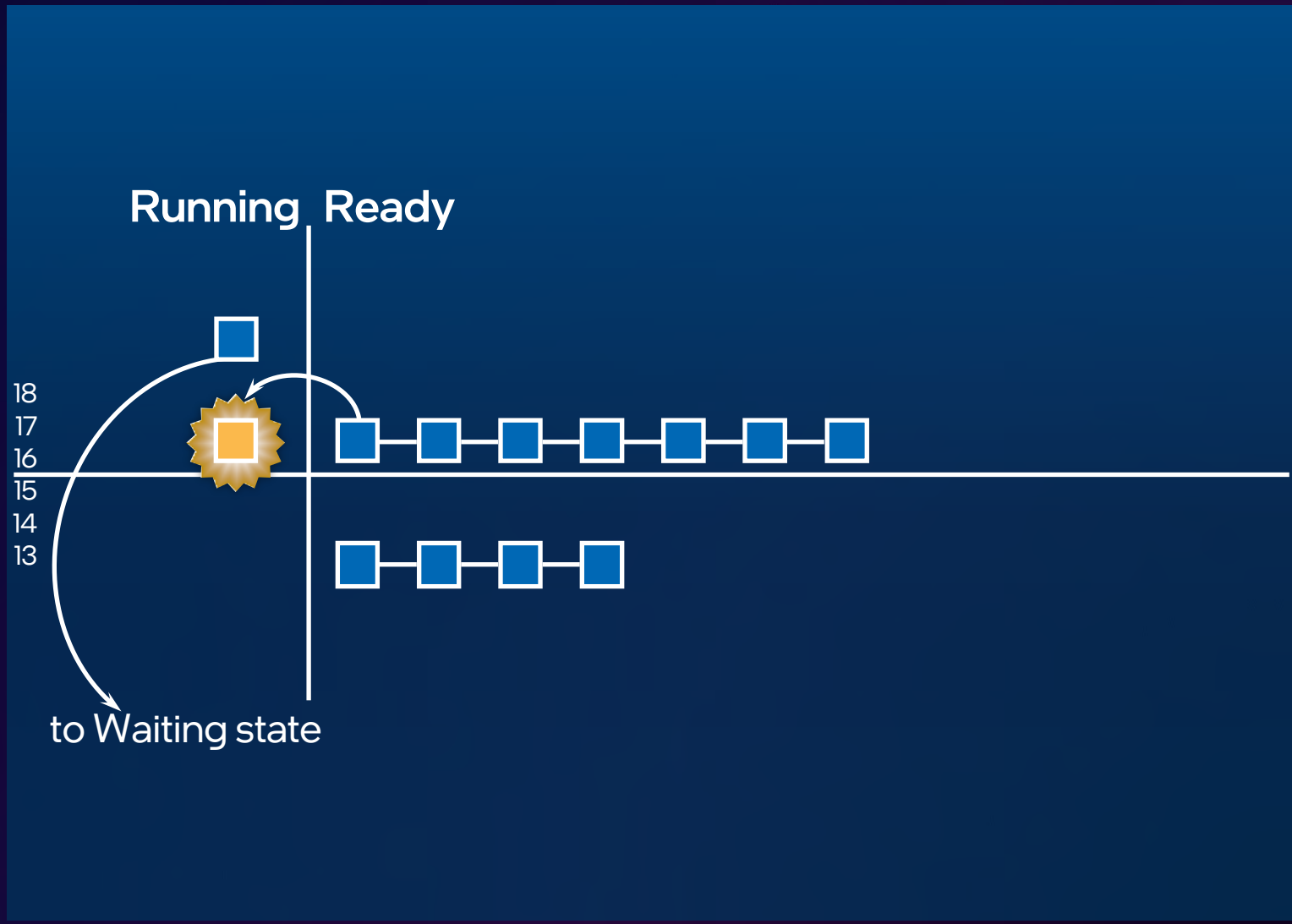
intel

Scheduling Scenarios: Voluntary switch



Voluntary switch

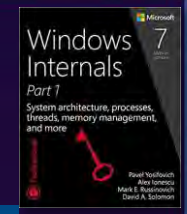
- Waiting on a dispatcher object
- Termination
- Explicit lowering of priority
  - Action: scan for next Ready thread (starting at your priority & down)



Current Hybrid systems have up to 8 LPs per group

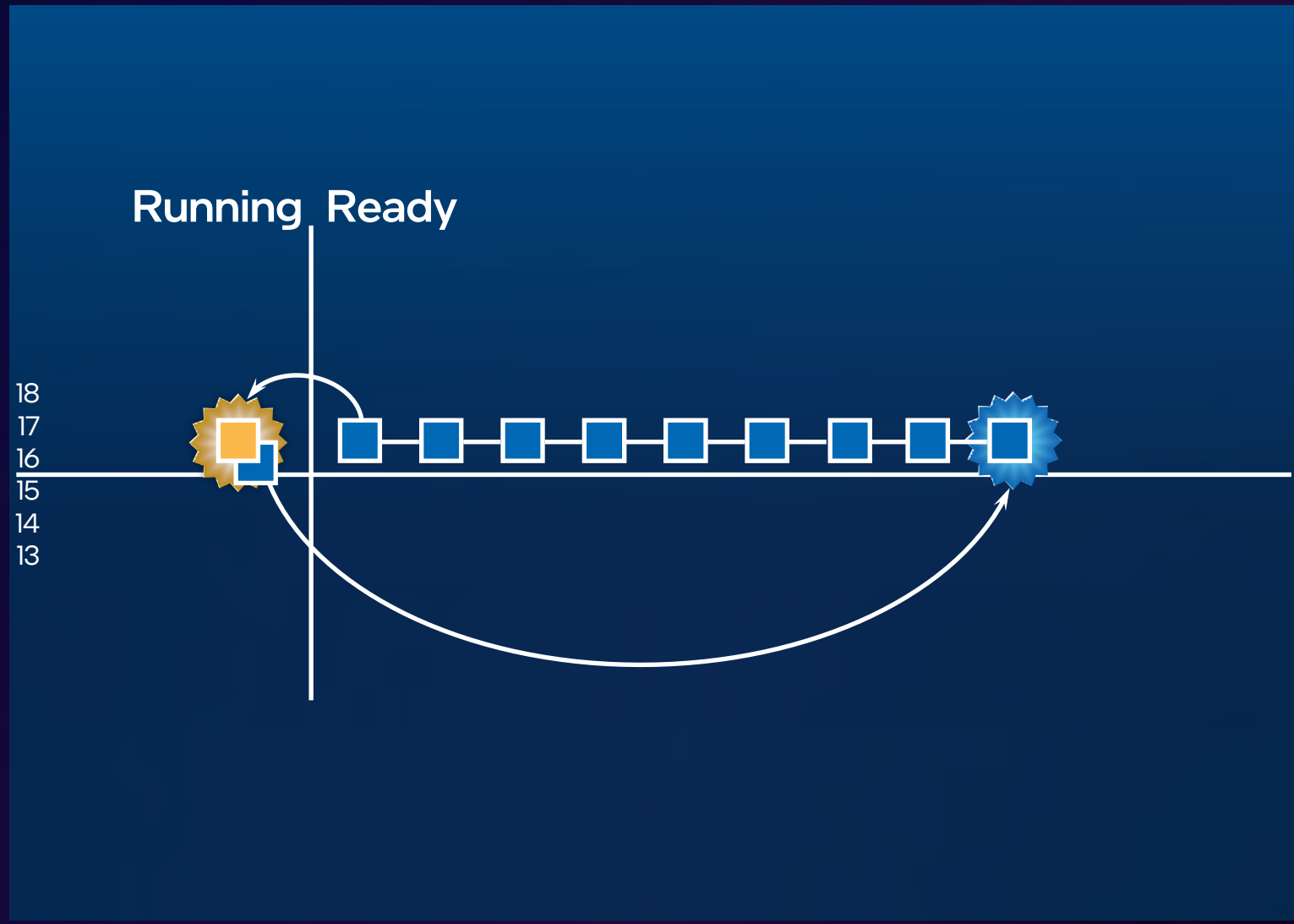
intel

Scheduling Scenarios: Quantum End



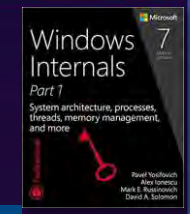
Running thread experiences quantum end

- 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



intel

Priority Boosts



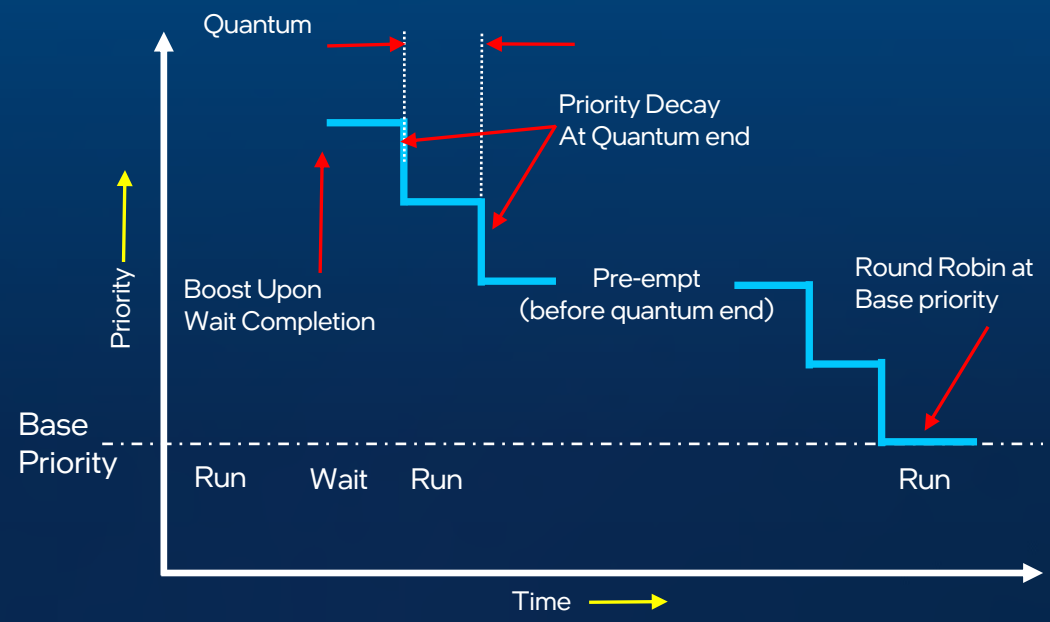
Windows periodically adjusts the current dynamic priority of threads, reasons include:

- 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



		Priority (%)									
ThreadID	TotalCPU%	<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

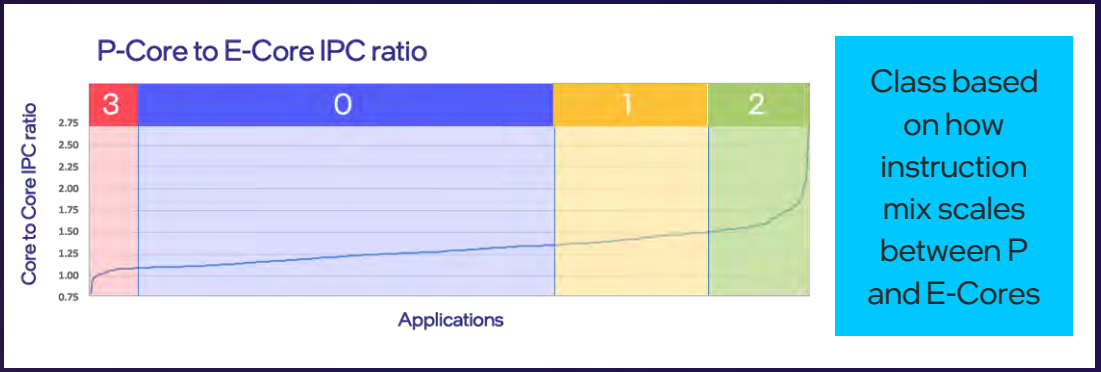
Thread Base Priority

intel

Intel® Thread Director Background

Thread Scheduling Overview

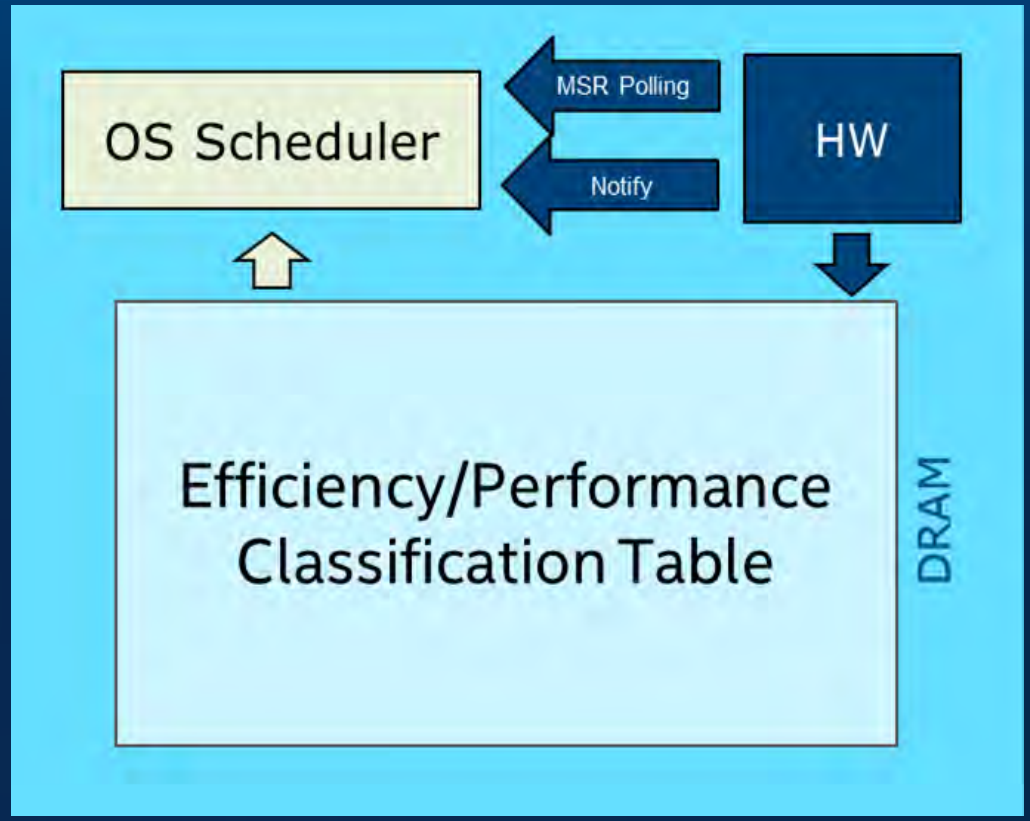
- Processors that support x86 hybrid architecture are categorized on their performance and efficiency.
- Intel Thread Director provides a hint to the os as to the thread that will benefit most from placement on a specific LP



Class based on how instruction mix scales between P and E-Cores

- 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….

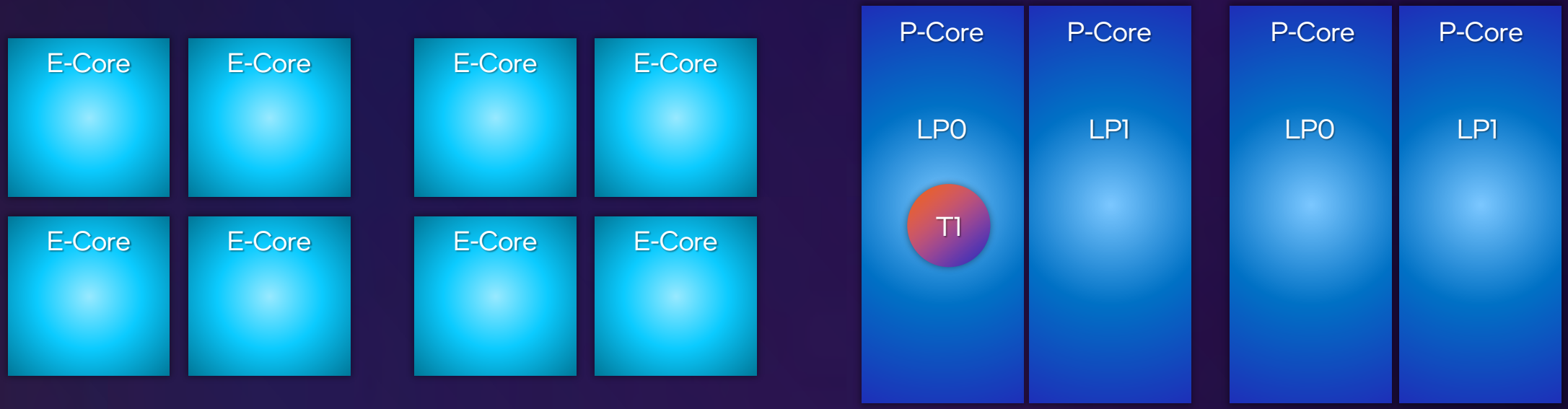
Power Management Settings related to Scheduling / Parking: Varies by power plan.

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.

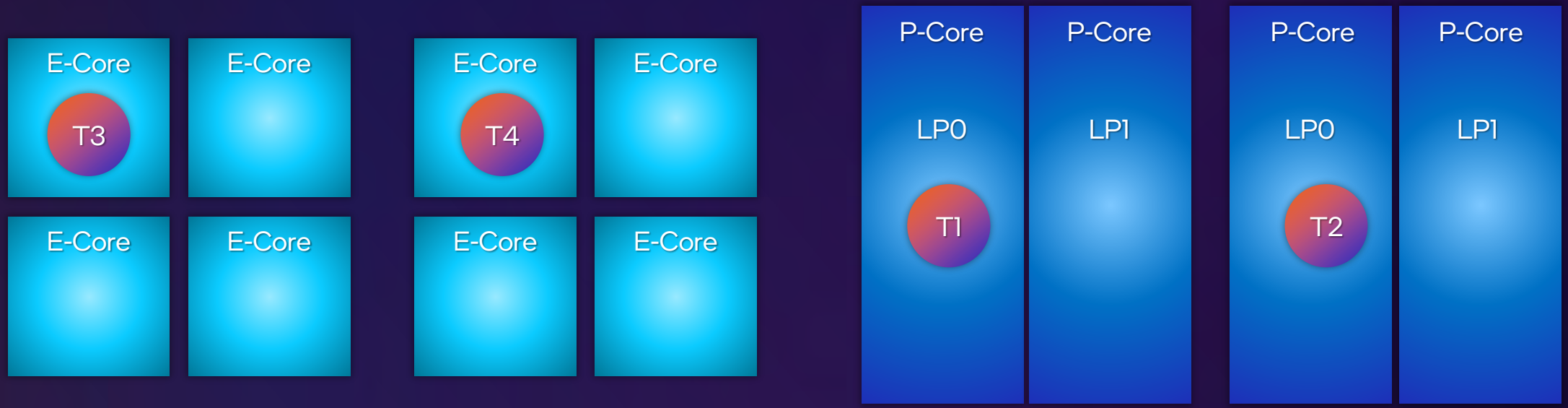
Example combination of P-cores and E-cores



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

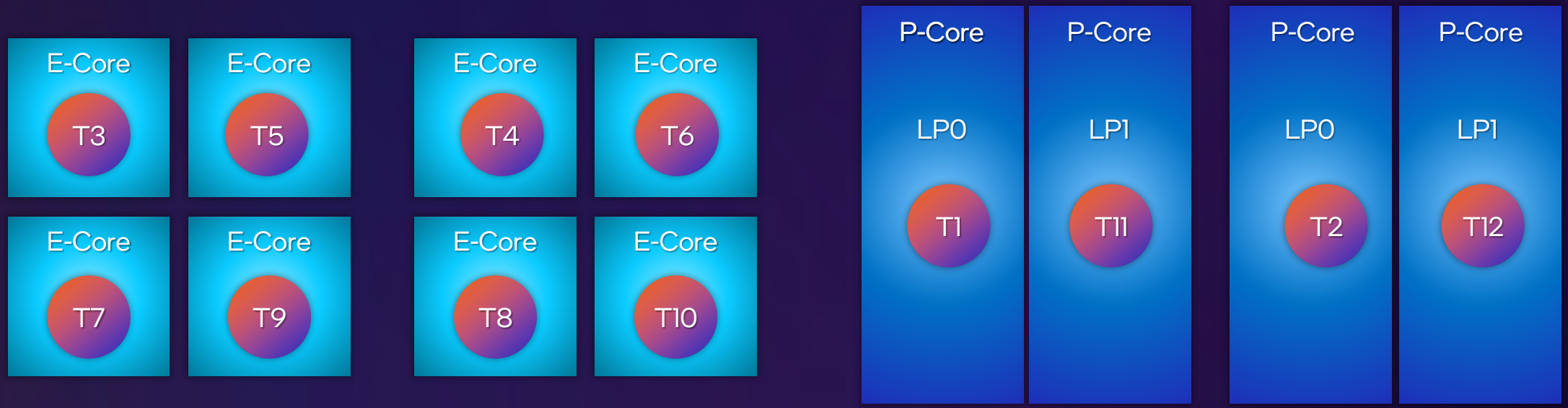
Example combination of P-cores and E-cores



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.

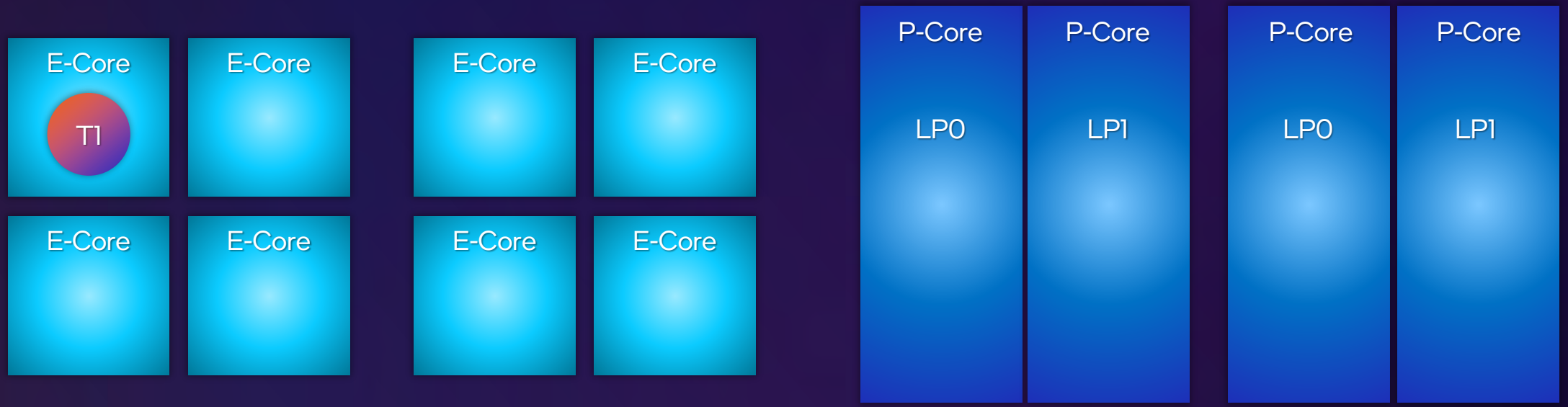
Example combination of P-cores and E-cores



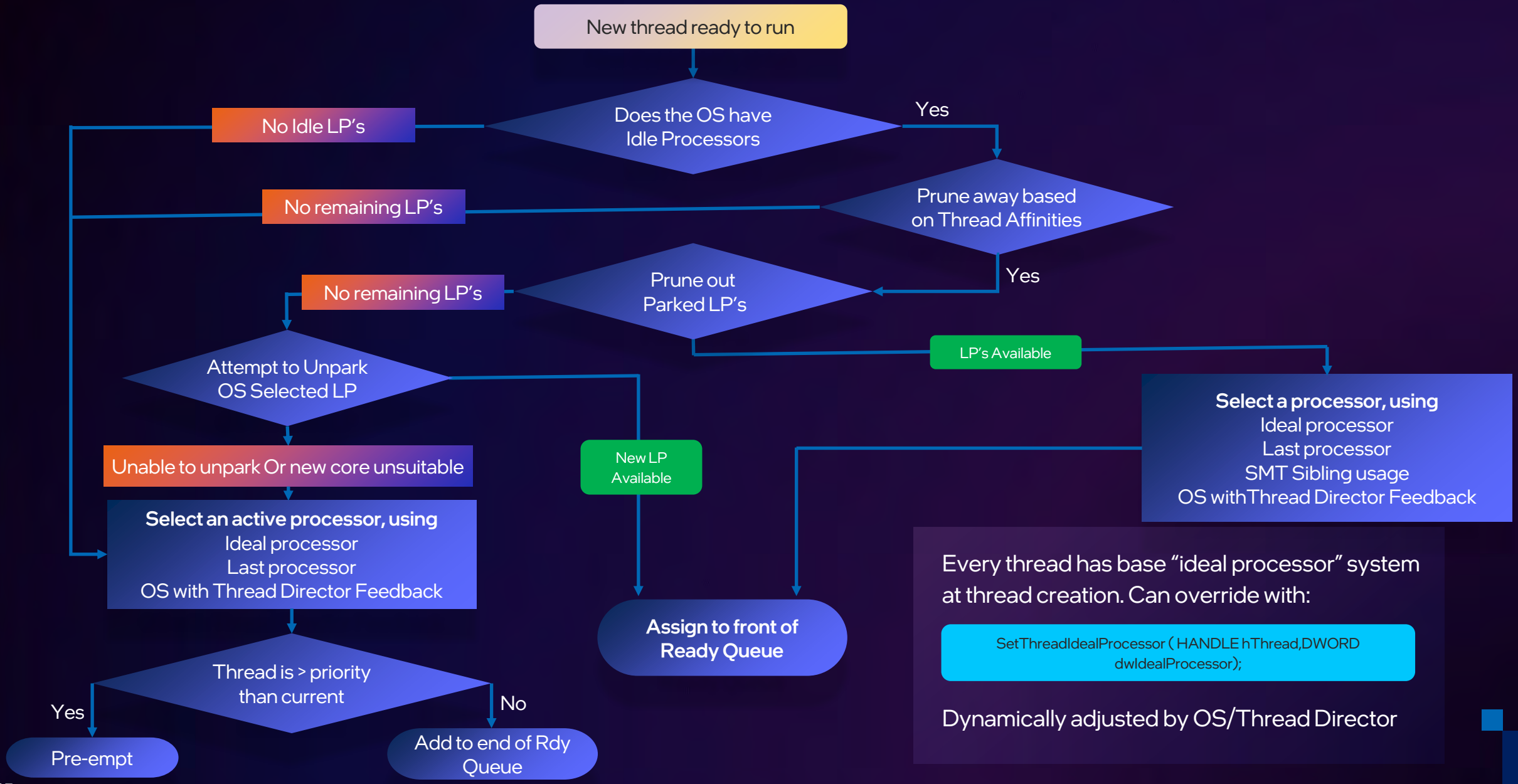
Low Power Scenario

- 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

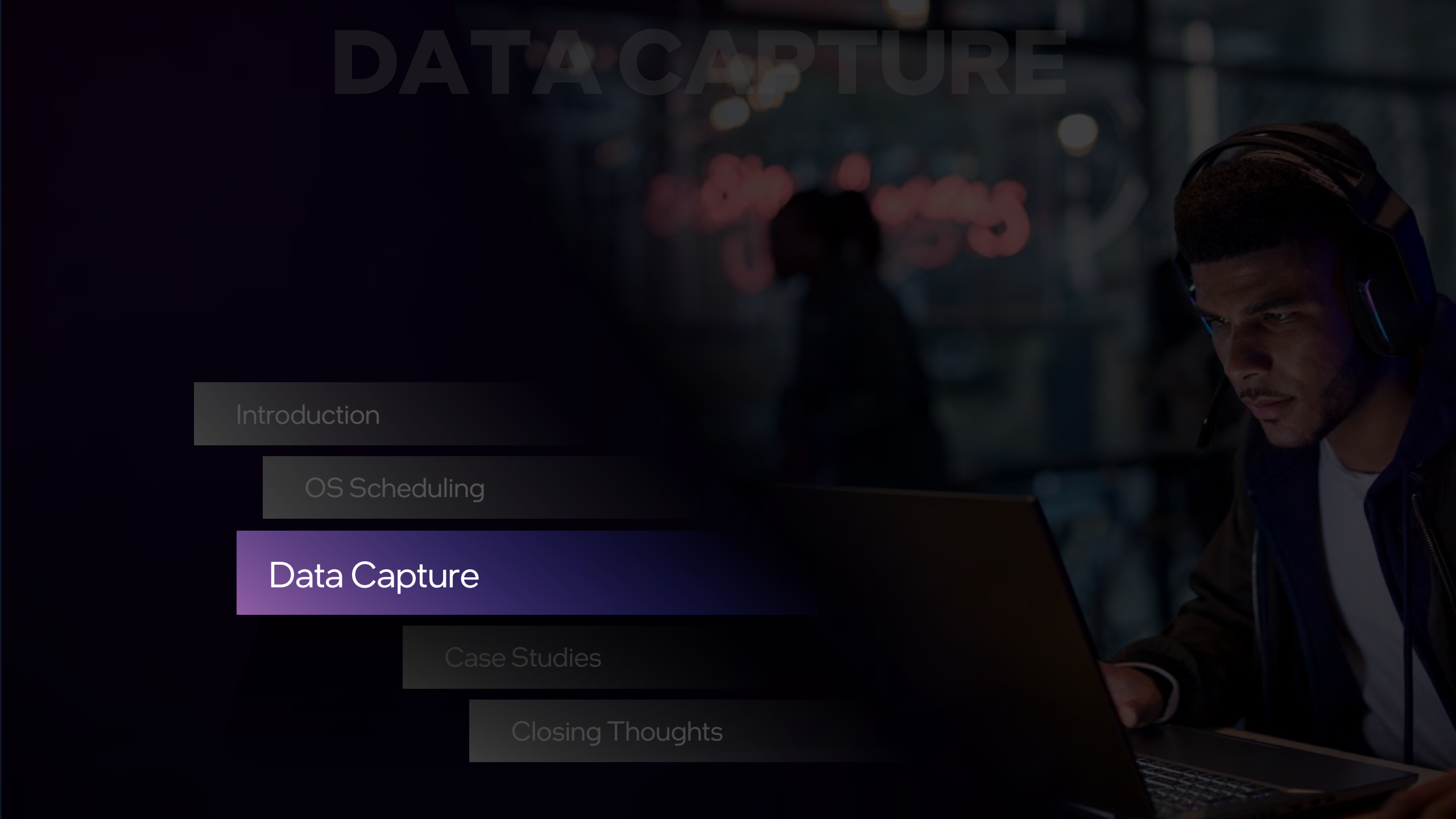
- Use OS Hints for soft affinity
- Any ISV code providing affinity could potentially see perf degrades as OS cannot override the decision
- QoS, SetThreadIdealProcessor help determine where the OS will queue a thread to run.
- CPUSets API. This API takes HGS do not use hints into account and breaks affinity

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

Online resource: Game Dev Guide for 12th Gen Intel® Core™ Processor

DATA CAPTURE



- Introduction
- OS Scheduling
- **Data Capture**
- Case Studies
- Closing Thoughts

Profiling Hybrid Games

FREE

VTune 2022

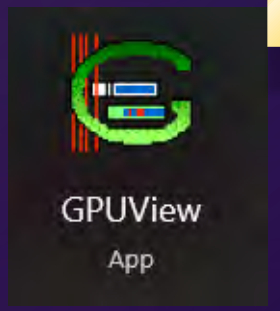
- CPI
- Cache
- Thread Director

VTune Download

WPA

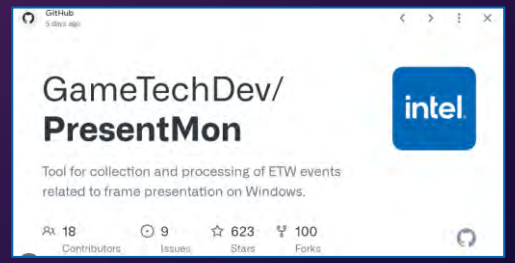
- SMT Usage
- Concurrency
- OS Behaviour

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

intel

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	Applicatic	ProcessID	SwapChai	Runtime	SyncInterv	PresentFl	Dropped	TimeInSe	msInPrese	msBetwee	AllowsTe	PresentM	msUntilRe	msUntilDi	msBetwee	msUntilRe	msGPUAct
2	engine.ex	13724	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	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	0x000000(	DXGI	0	512	0	0.082237	0.1837	4.6587	1	Hardware	2.9332	2.9332	4.5989	-1.6657	4.0976

msBetweenPresents = CPUtimgs

msUntilDisplayed = Display Latency

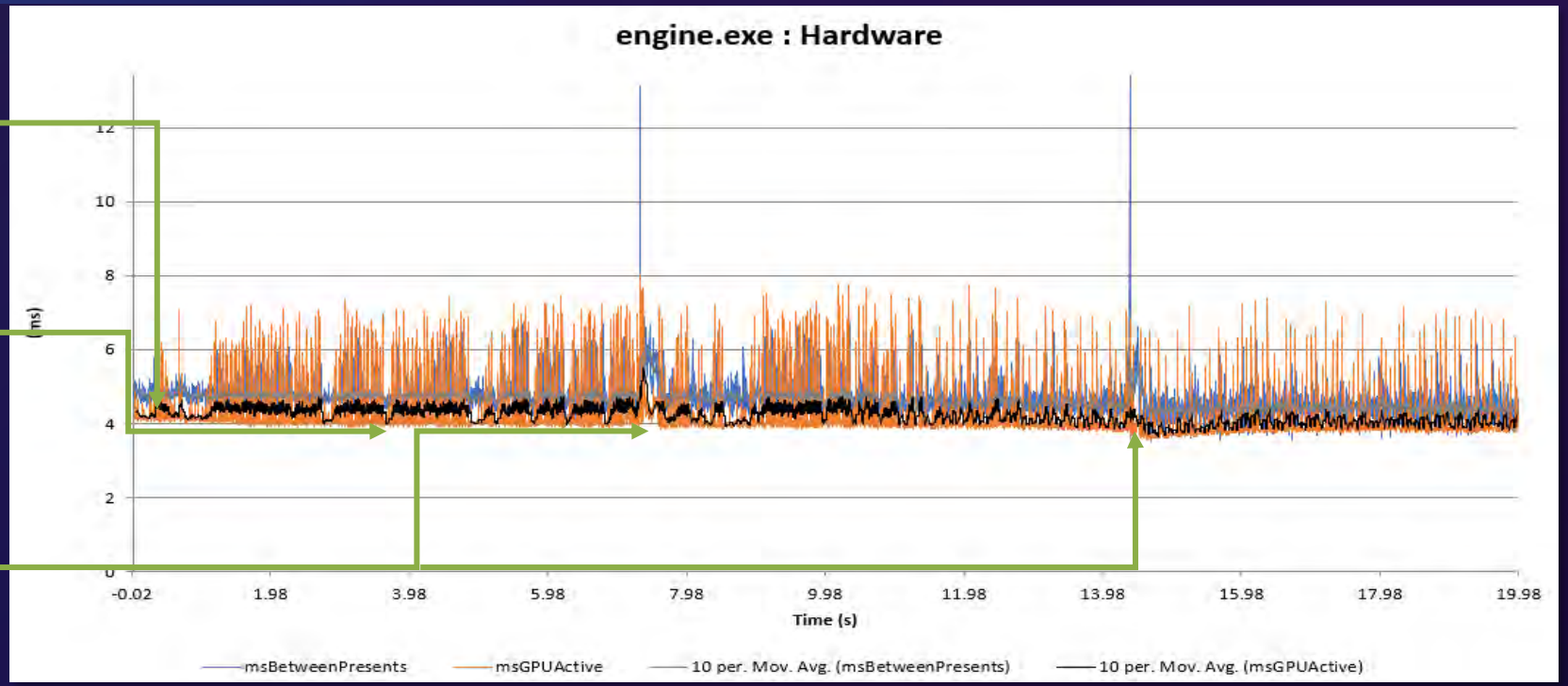
msGPUActive = GPU Timing

intel

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

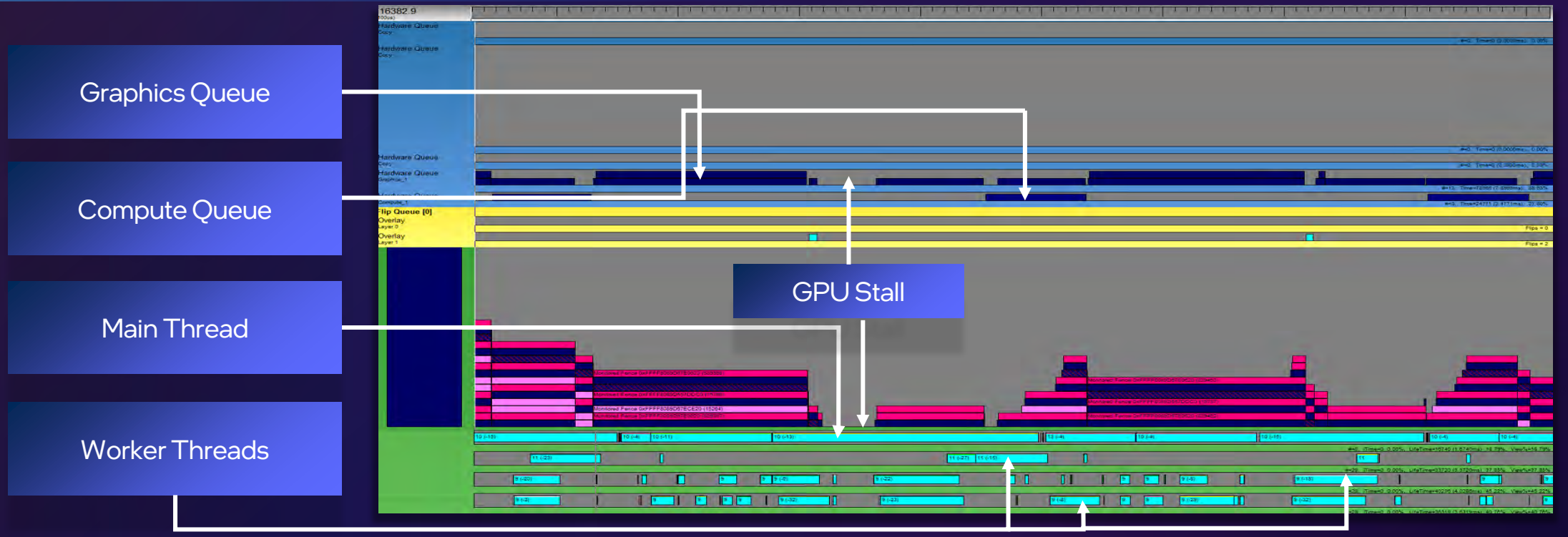


intel

Viewing GPU Data

GPUView
App

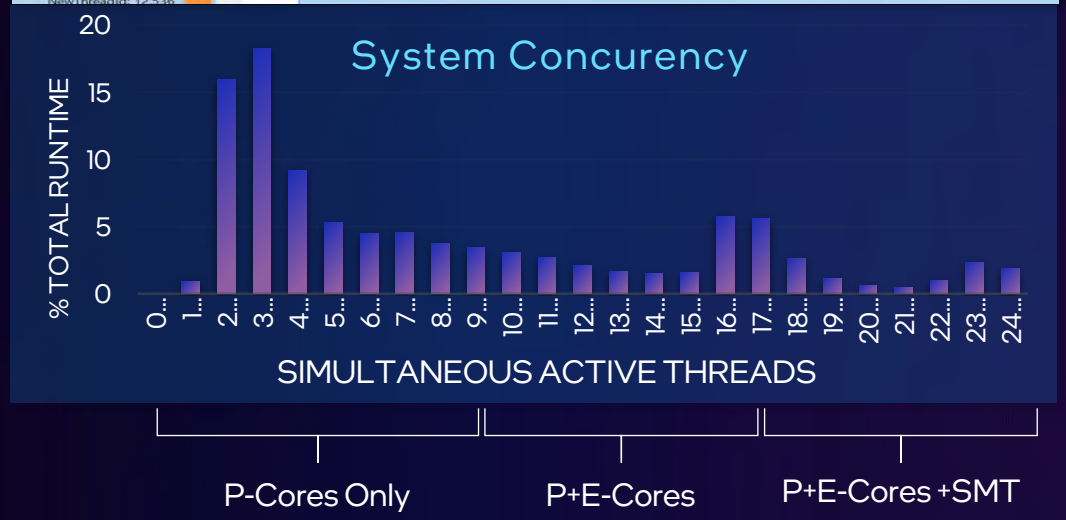
GPUView



intel

Understanding Concurrency

WPA: Timeline by Process, Thread



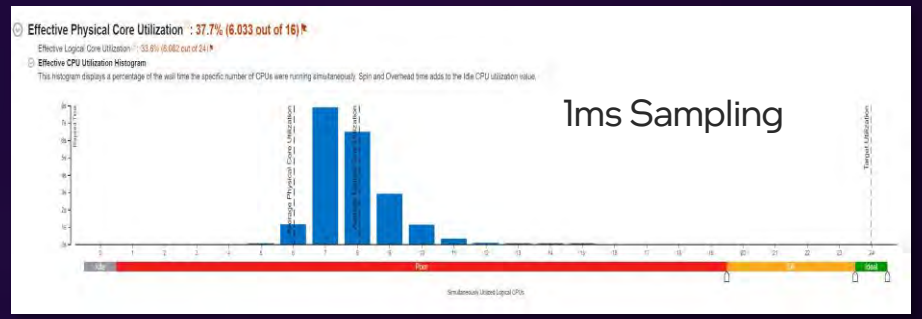
CPU Concurrency:

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

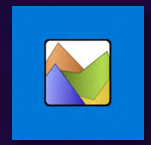
Xperf/WPA:

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

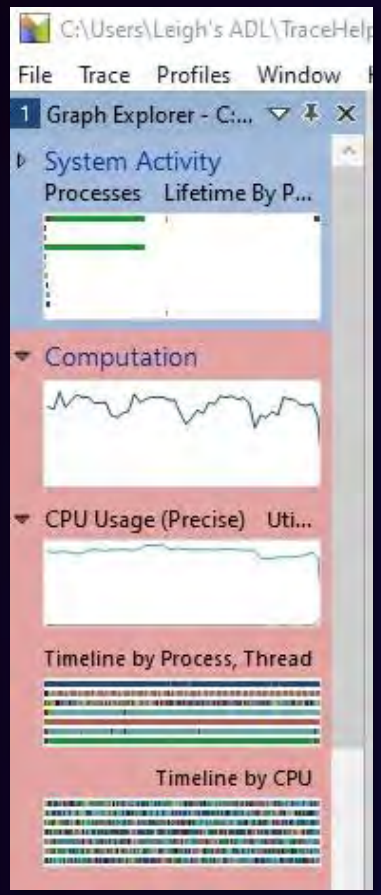
Sampled concurrency views like VTune don't provide enough detail on concurrency at the OS event level.



Understanding WPA (1/5)

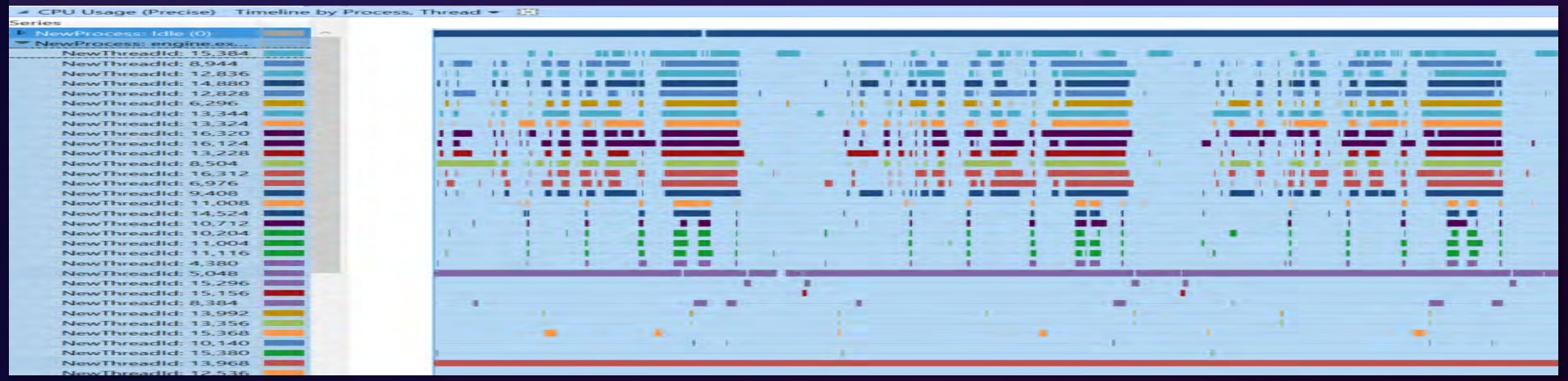


A lot of customisable views into OS/hardware level data



- Computation
  - CPU Usage
    - Timeline by Process, Thread

(Precise) ← Track events rather than using sampling

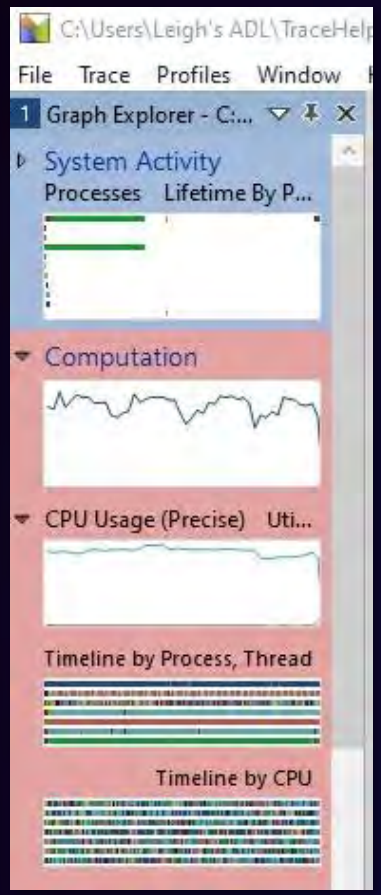


Thread-level timings

intel

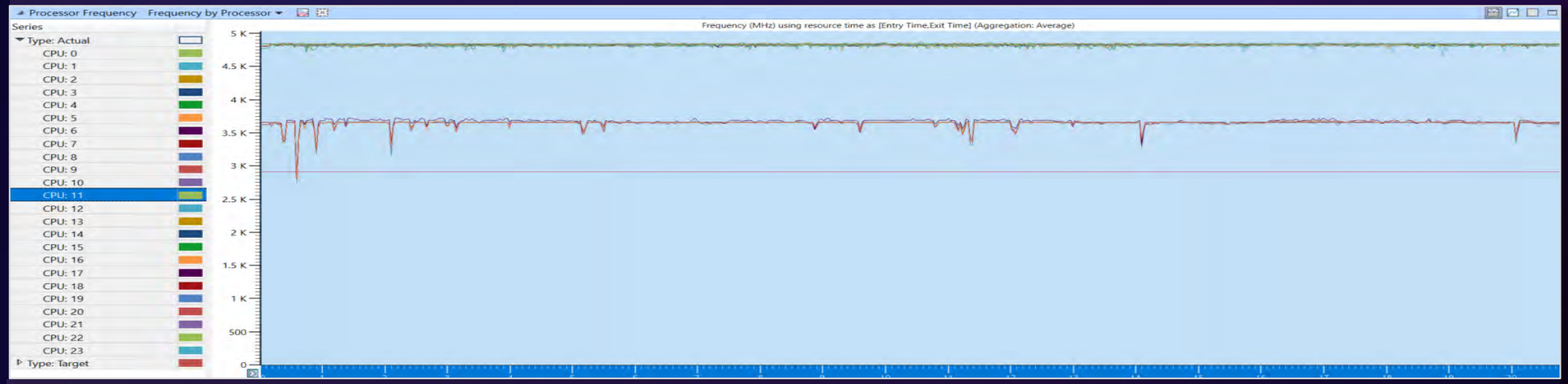
Understanding WPA (2/5)

A lot of customisable views into OS/hardware level data



- Power
  - Processor Utility
    - Processor Frequency

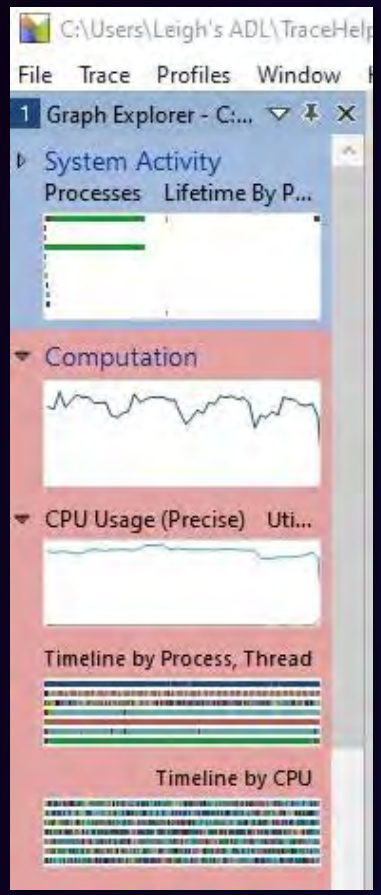
View Type Actual



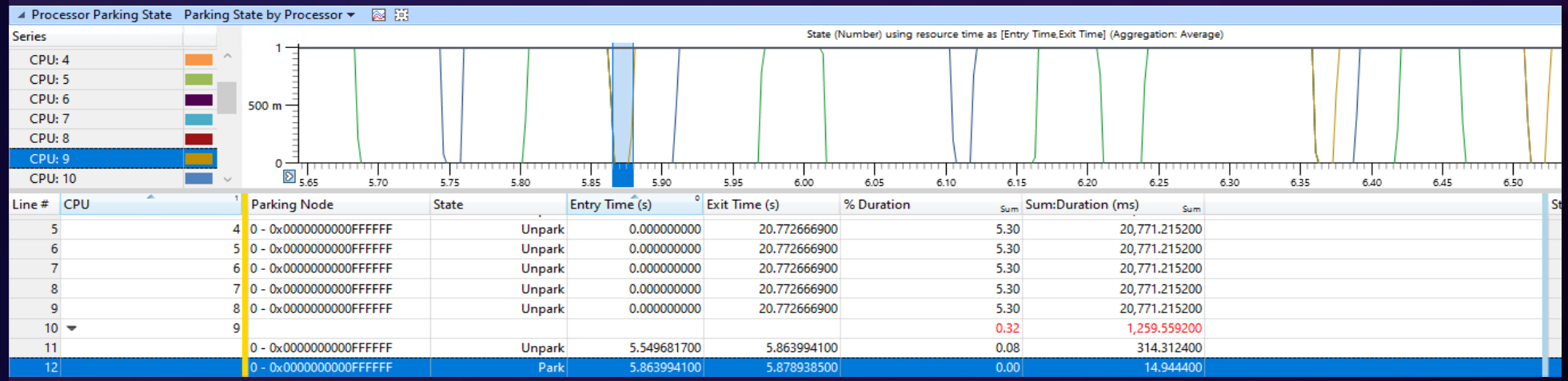
Hardware frequency including turbo

Understanding WPA (3/5)

A lot of customisable views into OS/hardware level data



- Power
  - Processor Parking State

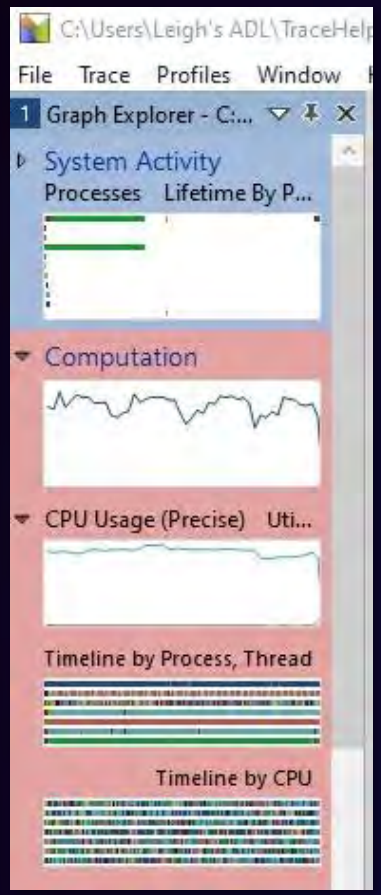


Line #	CPU	Parking Node	State	Entry Time (s)	Exit Time (s)	% Duration (Sum)	Sum:Duration (ms) (Sum)
5	4	0 - 0x0000000000FFFFFF	Unpark	0.000000000	20.772666900	5.30	20,771.215200
6	5	0 - 0x0000000000FFFFFF	Unpark	0.000000000	20.772666900	5.30	20,771.215200
7	6	0 - 0x0000000000FFFFFF	Unpark	0.000000000	20.772666900	5.30	20,771.215200
8	7	0 - 0x0000000000FFFFFF	Unpark	0.000000000	20.772666900	5.30	20,771.215200
9	8	0 - 0x0000000000FFFFFF	Unpark	0.000000000	20.772666900	5.30	20,771.215200
10	9					0.32	1,259.559200
11		0 - 0x0000000000FFFFFF	Unpark	5.549681700	5.863994100	0.08	314.312400
12		0 - 0x0000000000FFFFFF	Park	5.863994100	5.878938500	0.00	14.944400

Logical processor state

Understanding WPA (4/5)

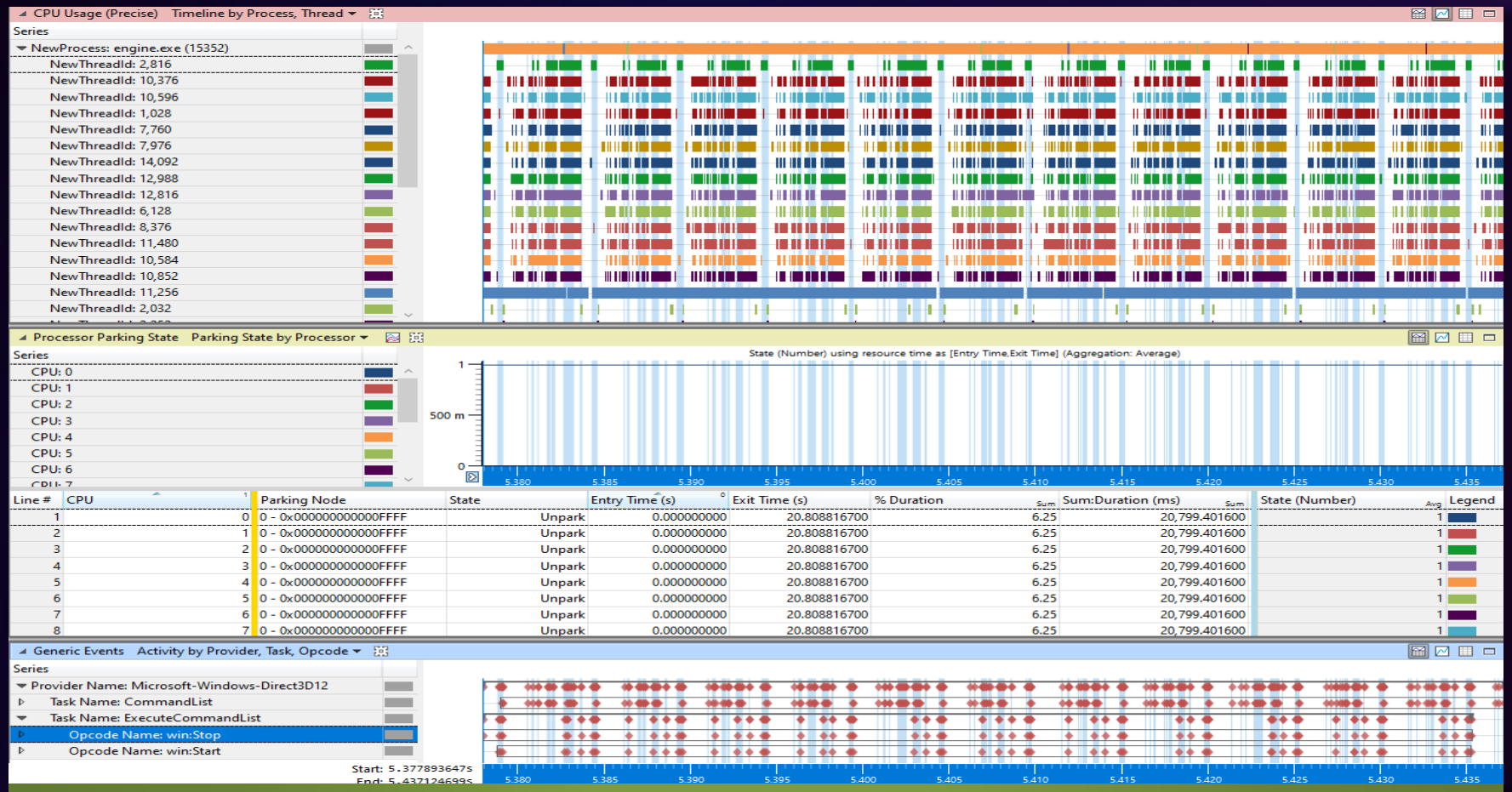
A lot of customisable views into OS/hardware level data



- 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

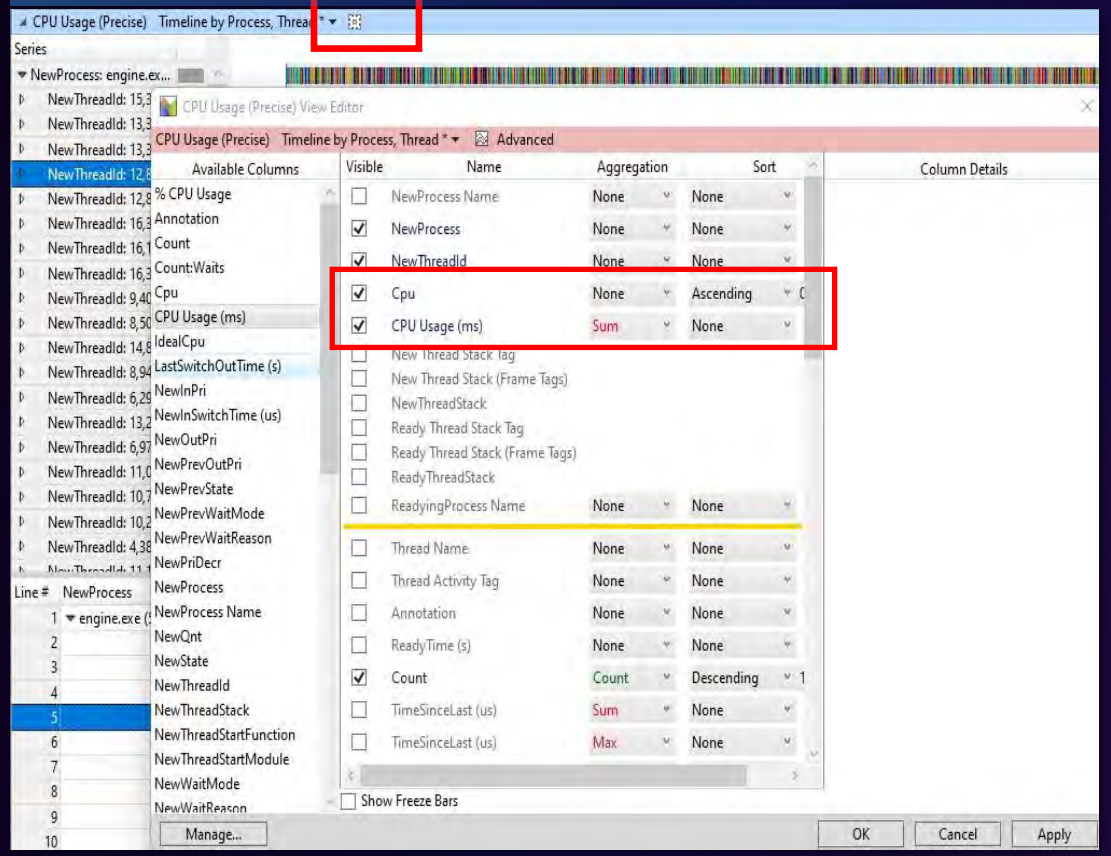
DX12 Events

Multiple views use same zoom/easy to align data

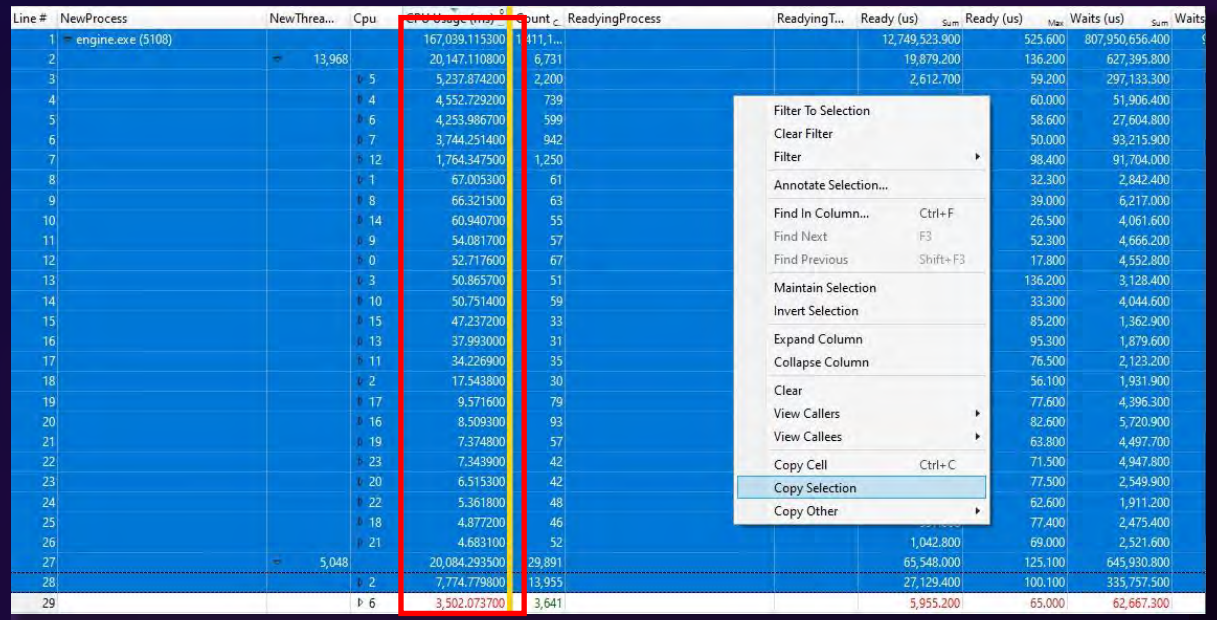
Thread Execution

CPU :Timeline by Process, Thread

Add Cpu + CPU Usage(ms) Sum



Sort By CPU Usage (Sum)



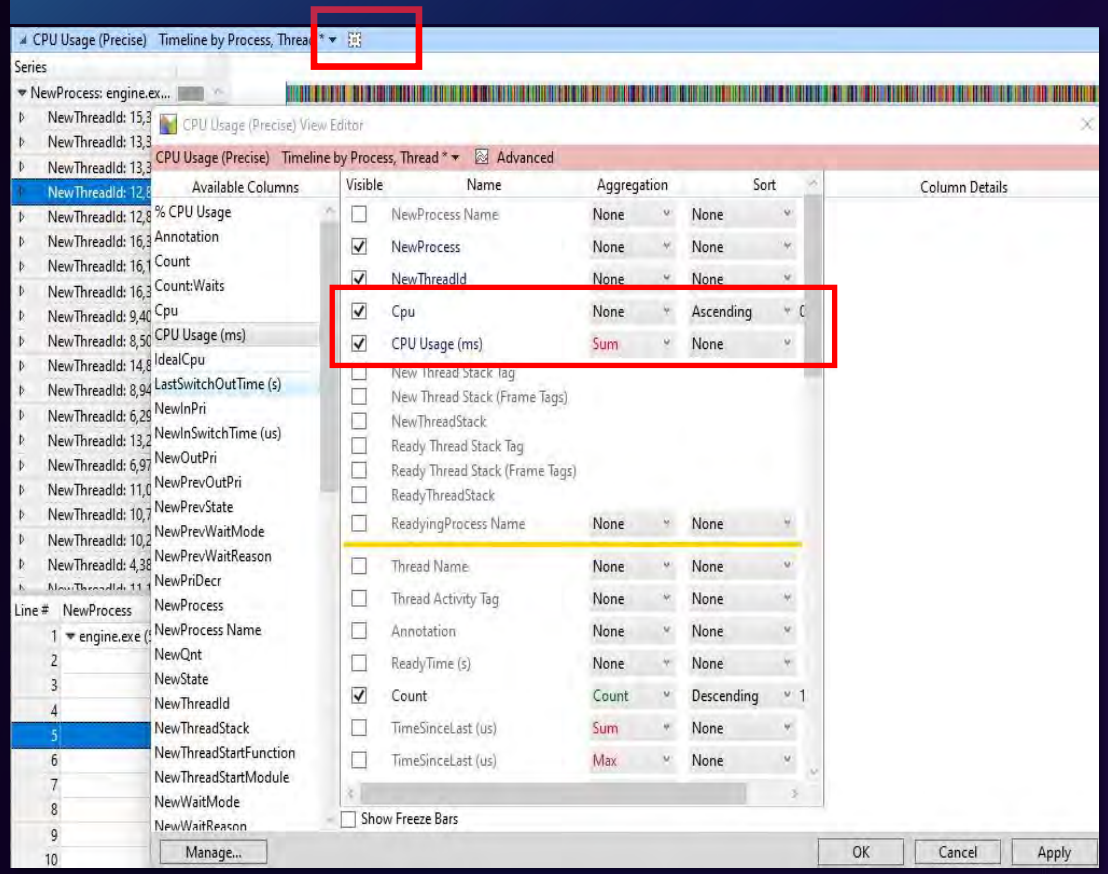
Breakdown thread execution time by Core type
Long duration threads should favour Performance cores
WPA Tabled can be copied to Excel and graphed.

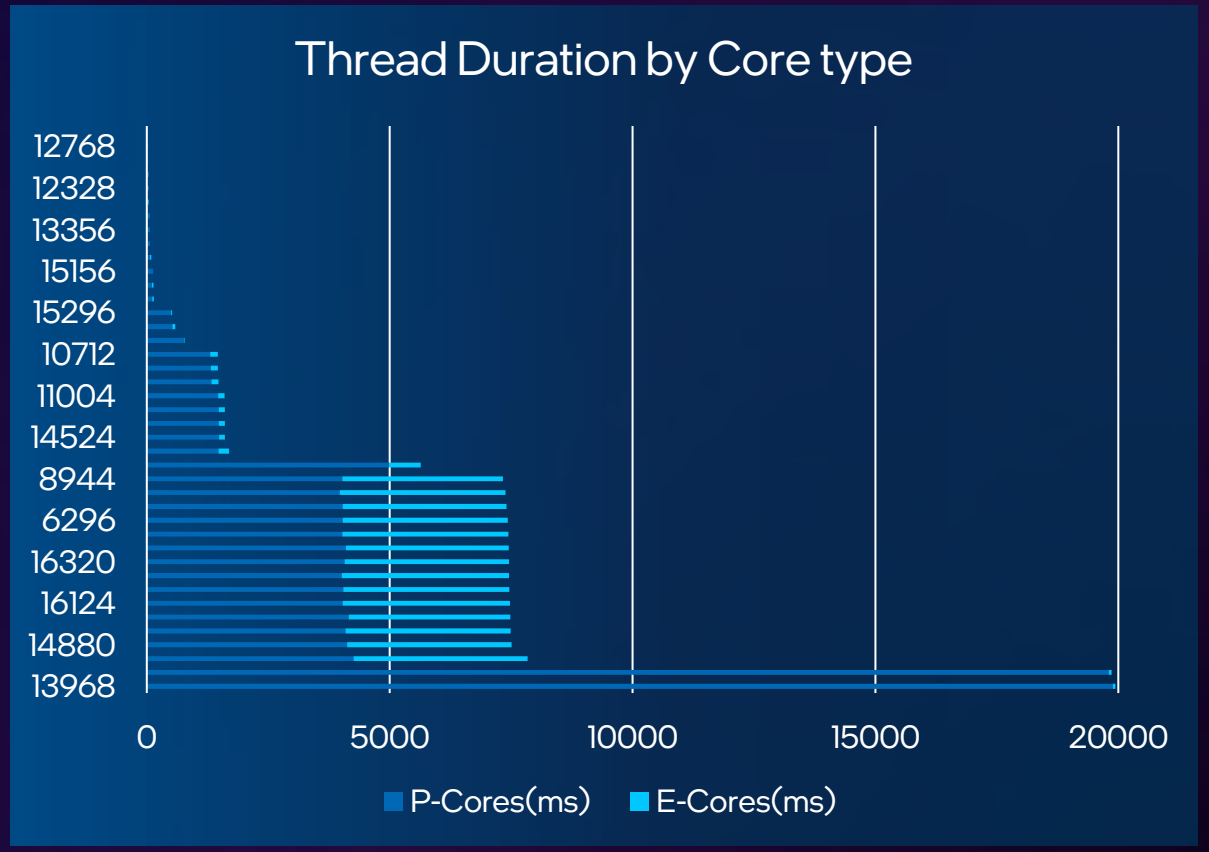
https://devblogs.microsoft.com/performance-diagnostics/wpa-intro/

Thread Execution

CPU :Timeline by Process, Thread

Add Cpu + CPU Usage(ms) Sum



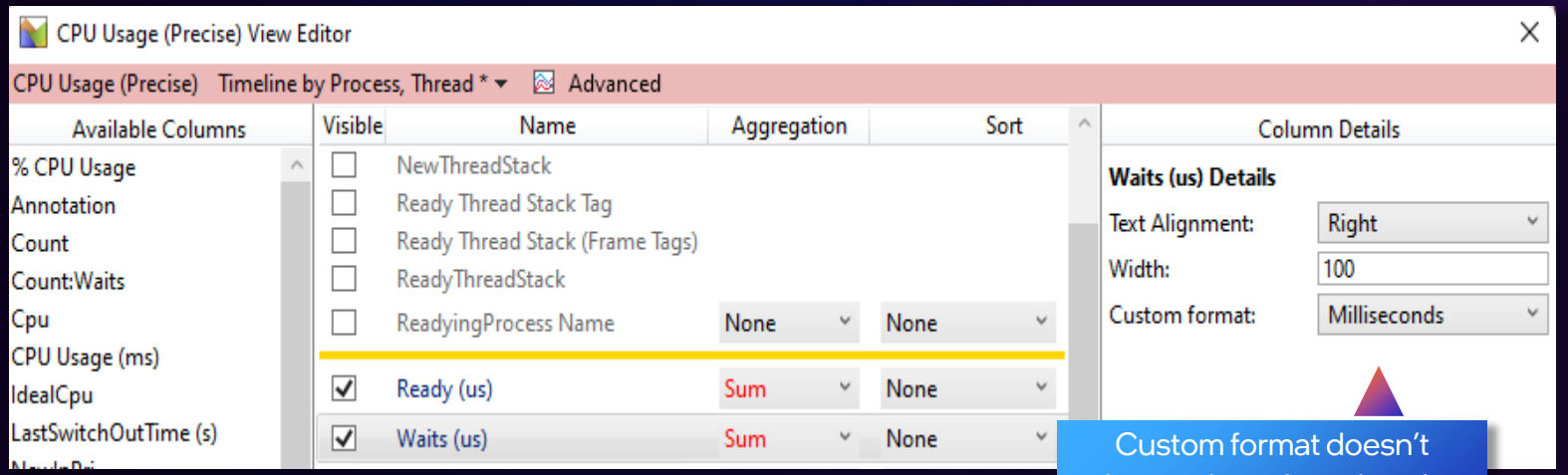


https://devblogs.microsoft.com/performance-diagnostics/wpa-intro/

Thread Ready/Wait times

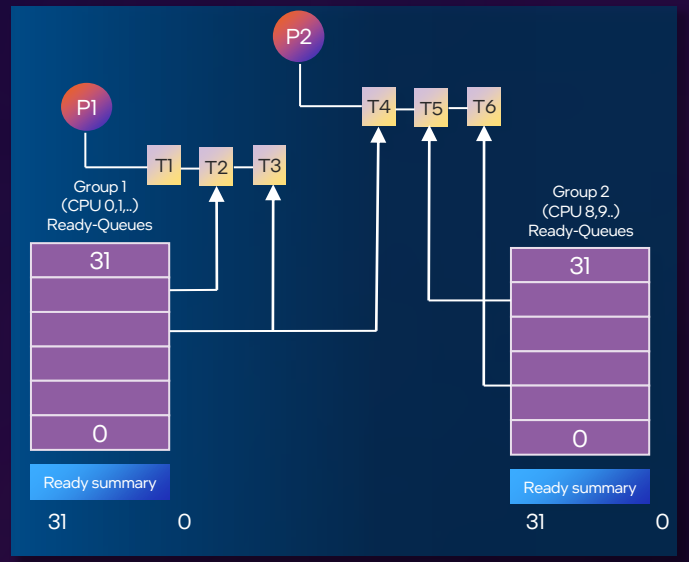
Are threads efficiently scheduled?

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



Custom format doesn't change the column header on some versions of WPA

Line #	NewProcess	NewThreadId	CPU Usage (ms)	Ready (us) Sum	Waits (us) Sum	Count	ReadyingProcess	ReadyingT...	Ready (us) Max	Waits (us) Max	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	5,911.210400	266.858500	14,588.815000	76,090			1,477.600	8,317.200	
3		14,648	7,838.370800	584.434500	12,354.585600	65,518			1,440.100	9,053.000	
4		13,148	7,661.142100	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

Time in Shared Ready queue

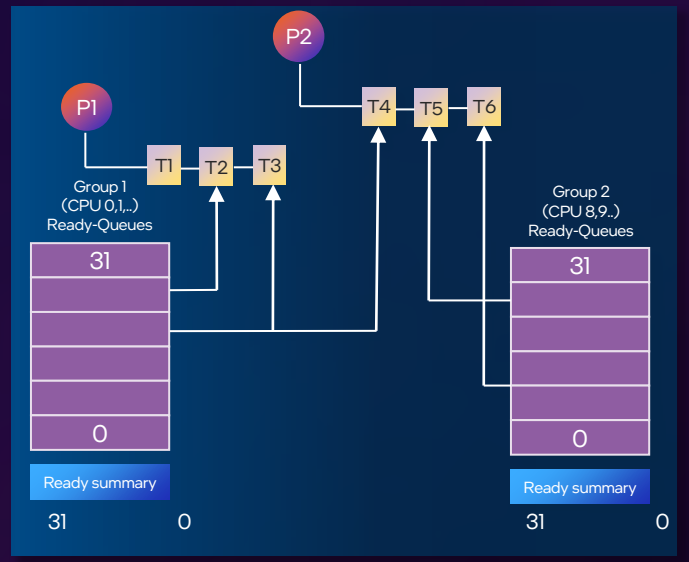
Time thread is in wait state

Max Ready Queue

Longest wait

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

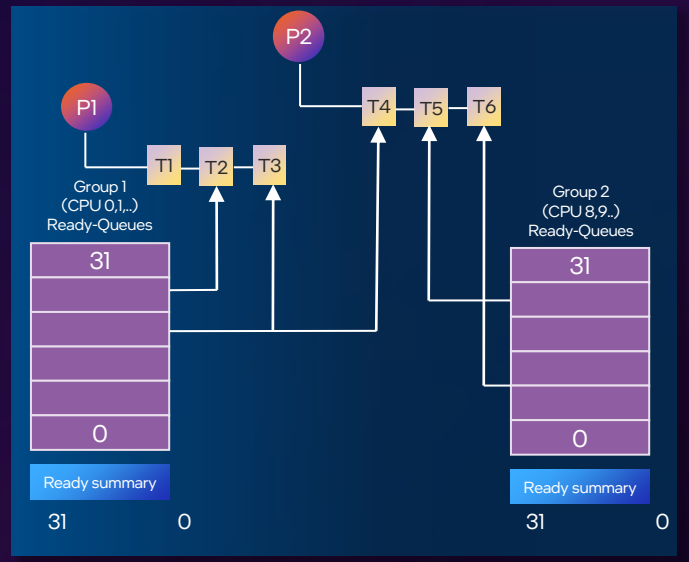
Thread Job	New ThreadId	CPU Usage (ms)	Ready (ms) Sum	Waits (ms) Sum	Count	Ready (us) Max	Waits (us) Max	Count: Waits
Render	7444	20125.96	43.94	749.32	9226	381.3	50467.6	5044
Game	11256	19926.66	62.12	831.01	37873	426	3319.6	27374
Worker 1	10376	8285.756	240.84	12276.28	75037	692.7	10014.7	71946
Worker 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

Hybrid System

	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

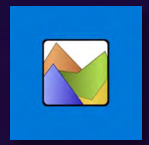
Smaller max. Ready time



Ideally compare against 2 systems

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

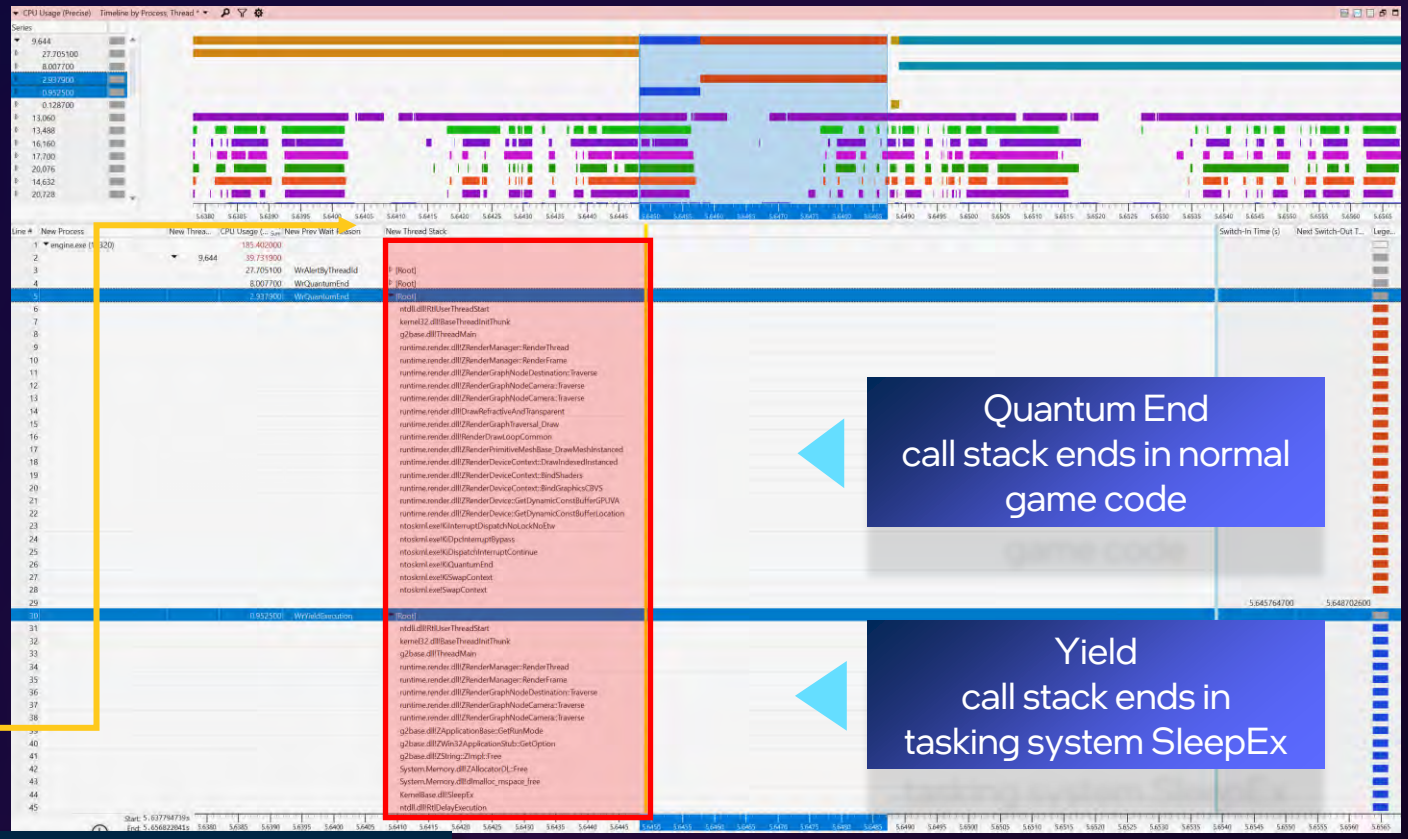
Understanding Context switches



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

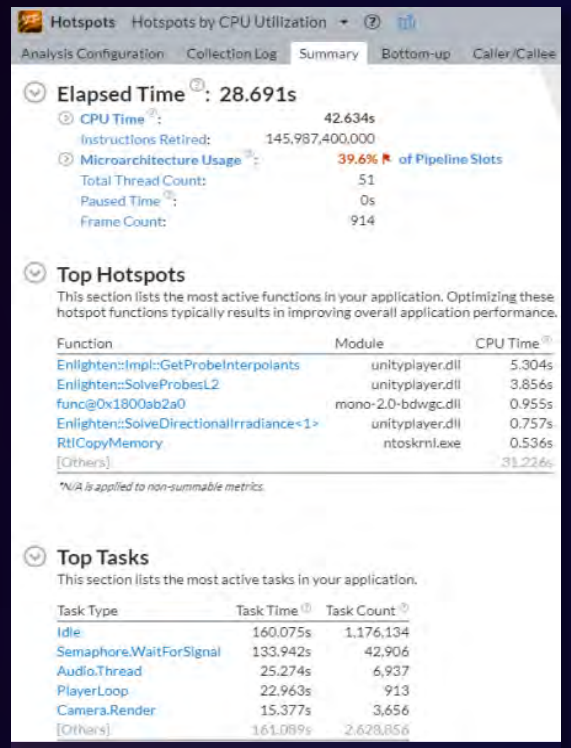
NewThread Stack

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

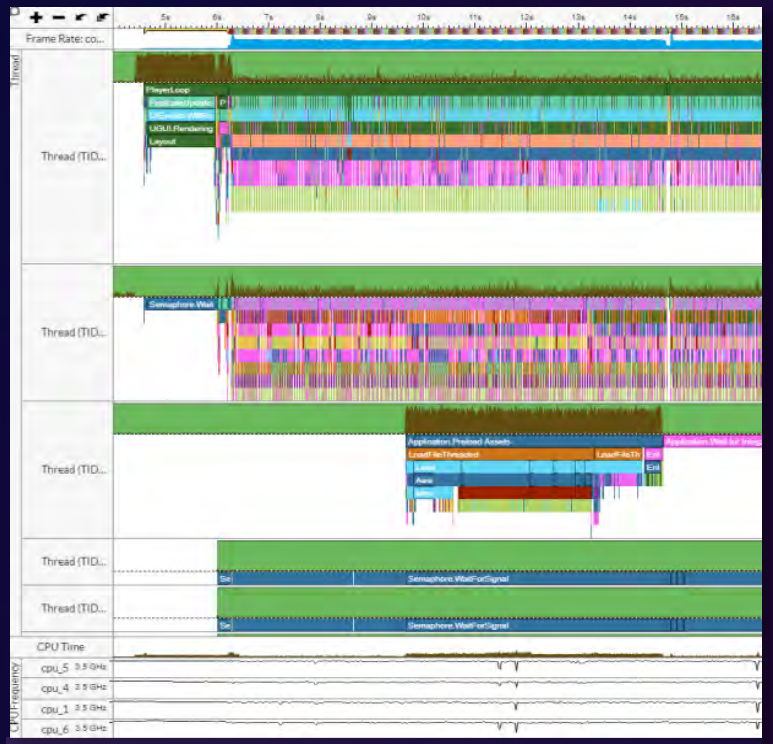
intel

Intel® VTune™ Profiler

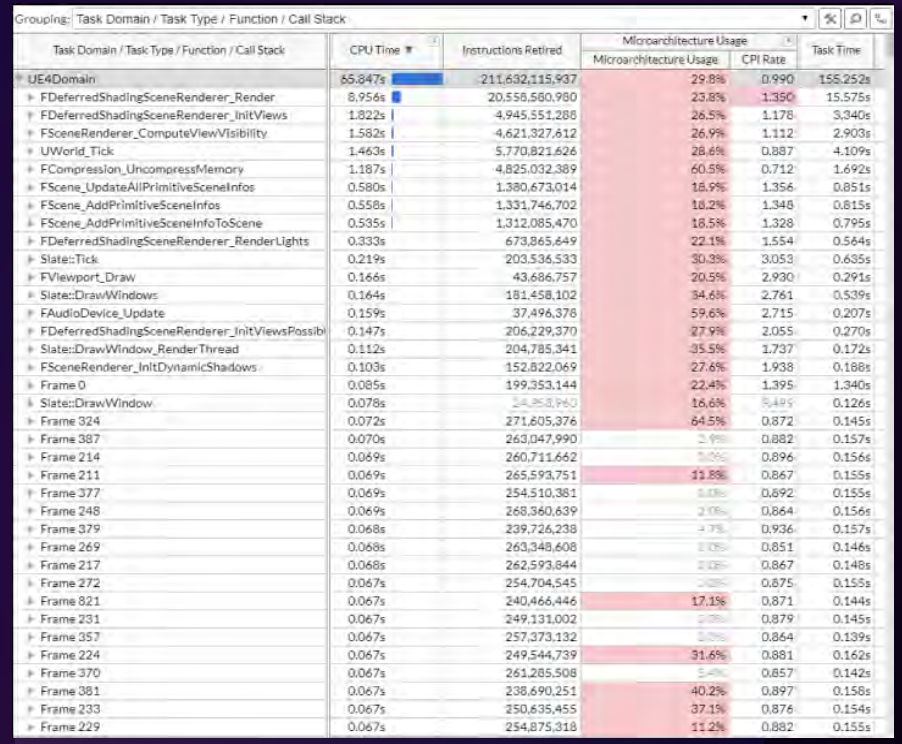
Advanced sampling profiler allows you to quickly identify CPU bottlenecks causing slow frames and tasks.



Hotspot Analysis: Identifies functions consuming the most CPU time



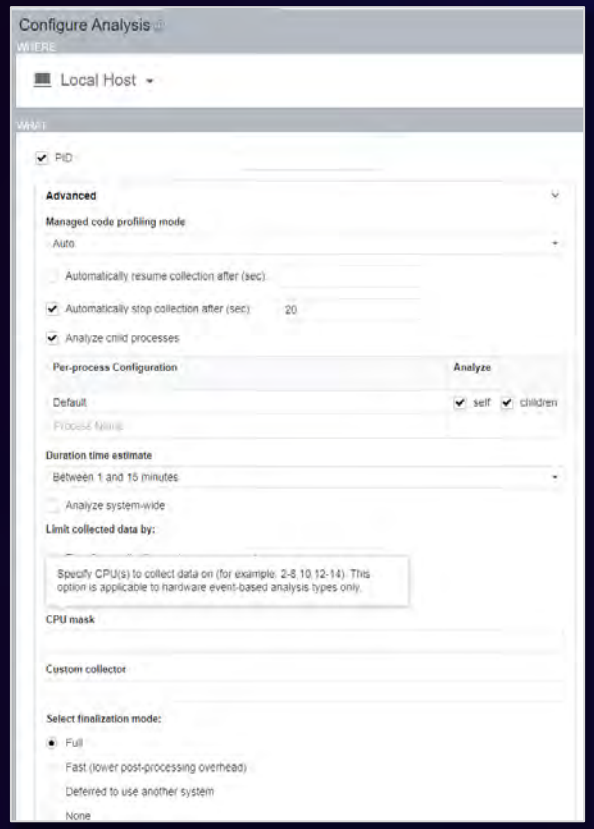
Thread Performance: Visualizes thread behavior to quickly identify concurrency problems

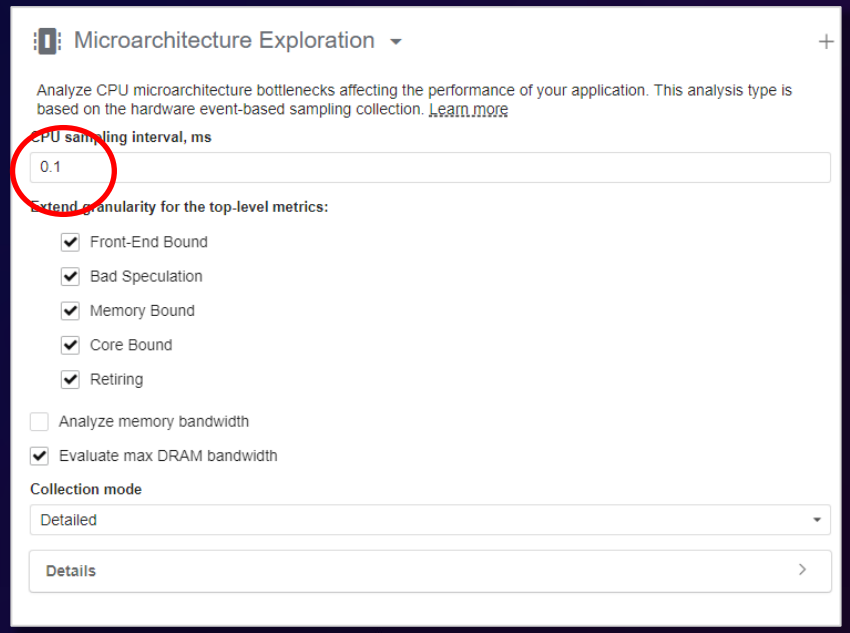


Instrumentation API: Extensive API enables frame and task markup for better results

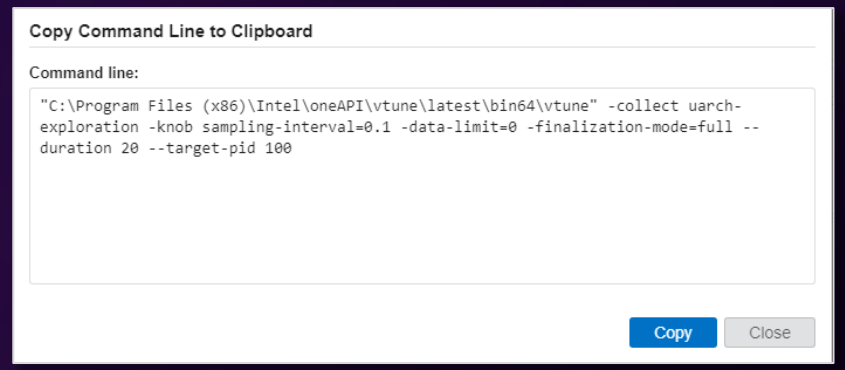
Finding Architectural Issues

Configure VTune™ for microarchitecture analysis: Small sampling internal.





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

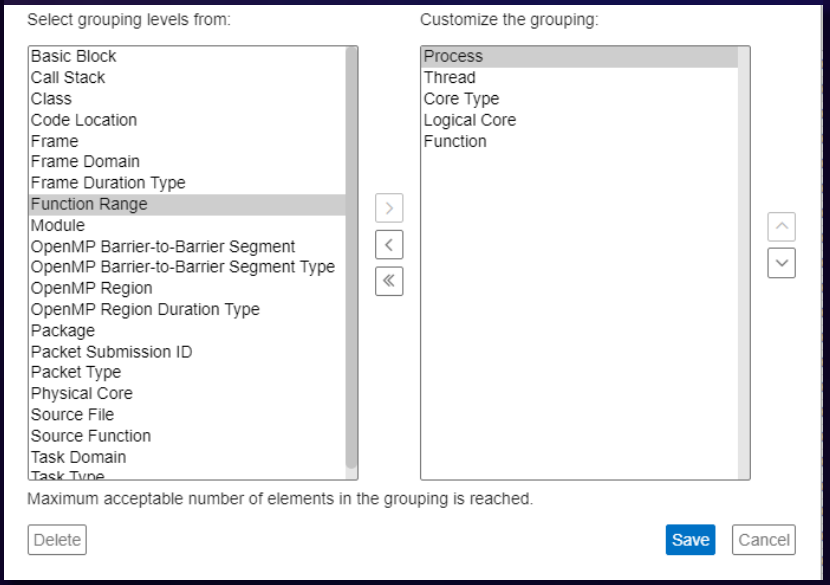
Metric	Description
CPI Rate	Cycles per Instruction Retired, or CPI, how much time each executed instruction took, in units of cycles. Modern superscalar processors issue up to four instructions per cycle, suggesting a theoretical best CPI of 0.25.
Cache Bound	This metric shows how often the machine was 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.

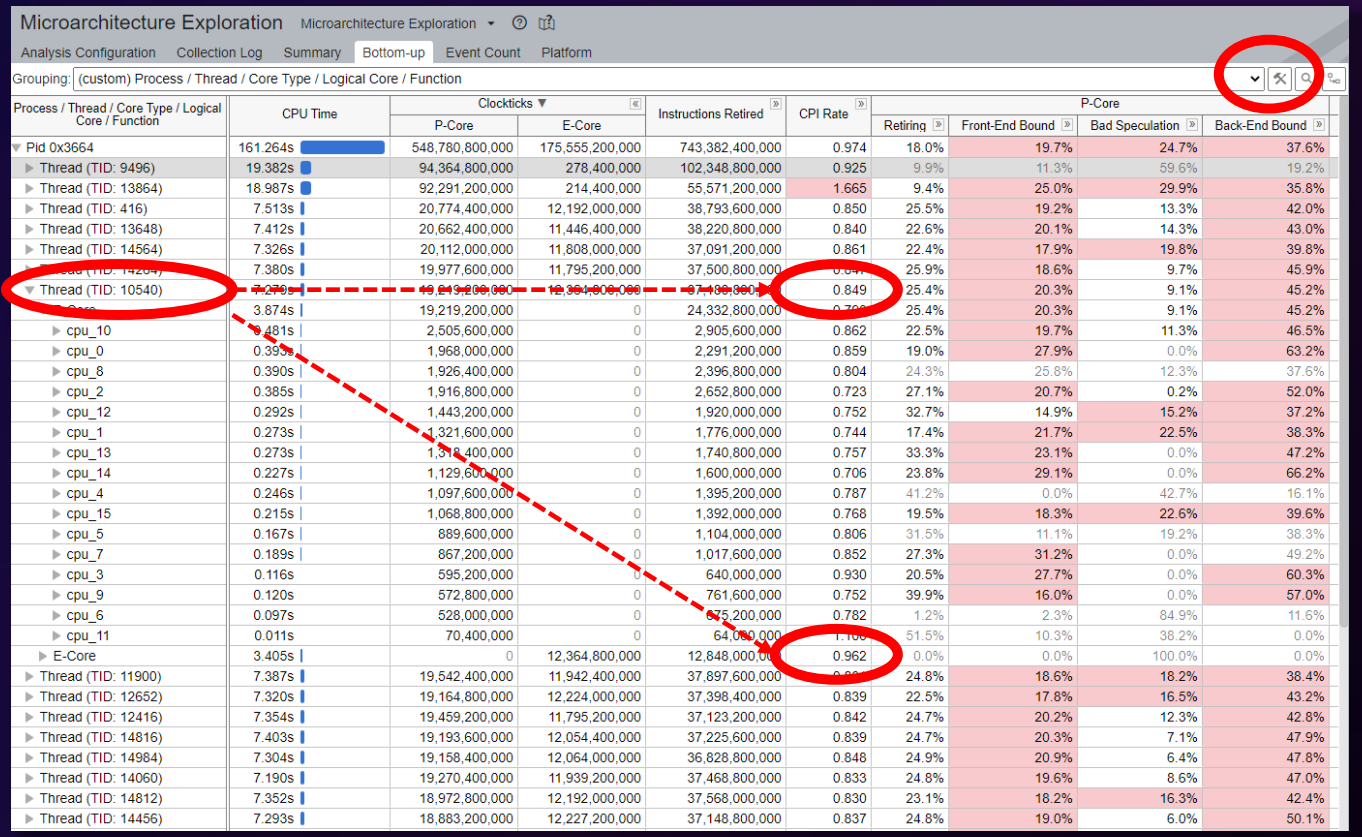
Microarchitecture by Core Type

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

Bottom-up:

- Custom grouping ( )





Relative Thread Perf. by Core Type

Thread	Thread ID	P-Core CPI	e-Core CPI	P-Core Million Instructions/Second	E-Core Million Instructions/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

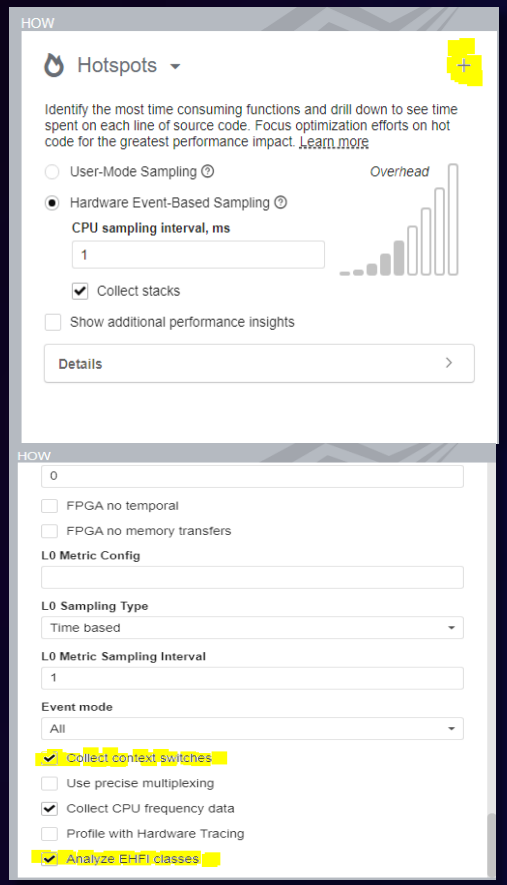
Potentially unfair to compare P- and E-Cores: E-Cores are lowering sibling activity.
(see slide 8 - Hyper-Threading Recap)

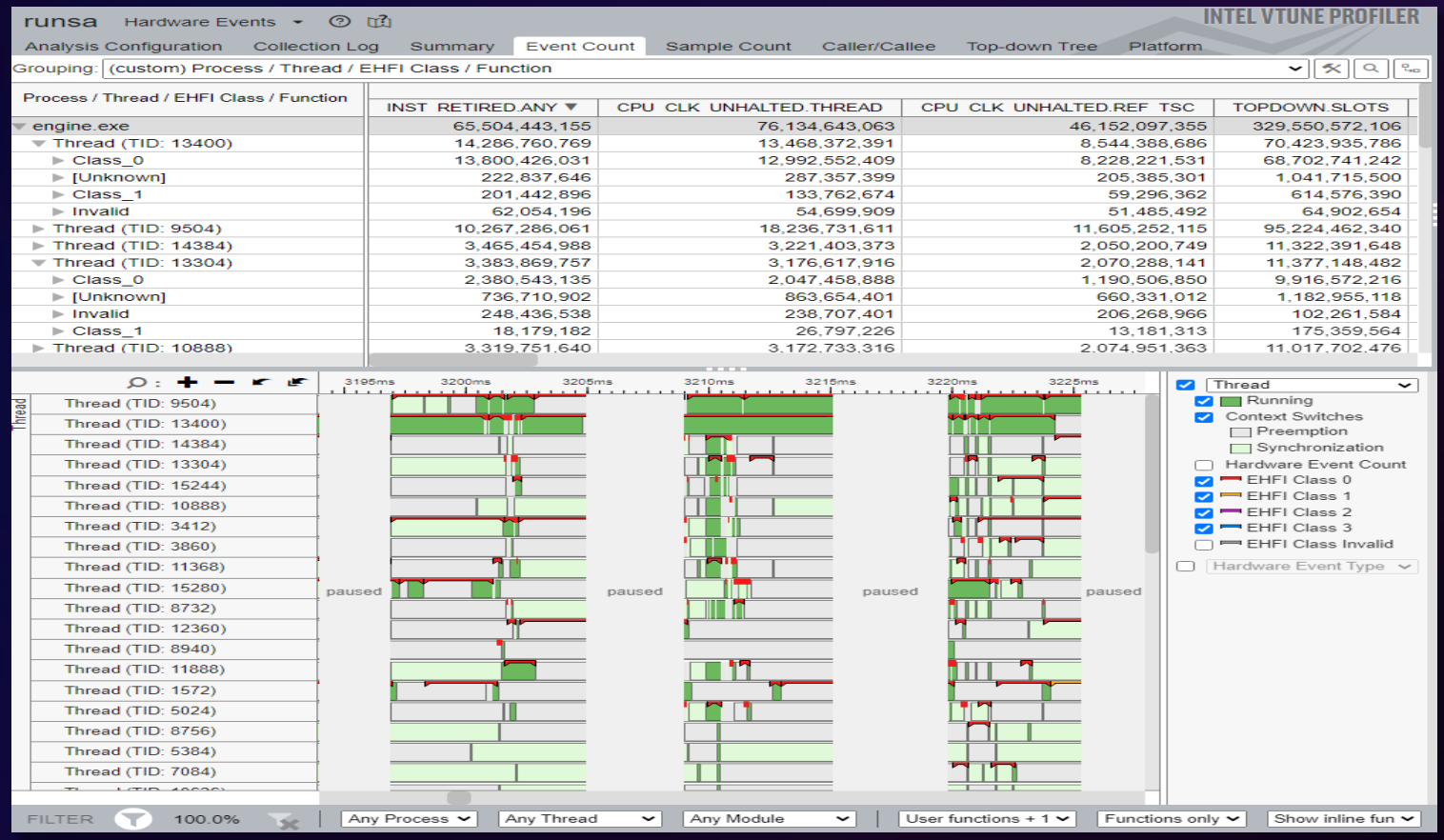
*****SMT Statistics*****			*****SMT Statistics*****		
	8C/16T			Hybrid	
Core ID	Both Siblings Idle	Both Siblings Active	Core ID	Both Siblings Idle	Both Siblings Active
	Percentage	Percentage		Percentage	Percentage
LP0 & LP1	44.94	29.75	LP0 & LP1	45.44	20.34
LP2 & LP3	43.19	34.73	LP2 & LP3	31.47	23.08
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.

Thread Director Uncovered





- Analyse EHFI classes as part of a hotspot VTune collection.
- Most game code will be class 0 :- and will target the P-Cores by default.*

* When not power constrained

CASE STUDIES

Introduction

OS Scheduling

Data Capture

Case Studies

Closing Thoughts

Case Study Background

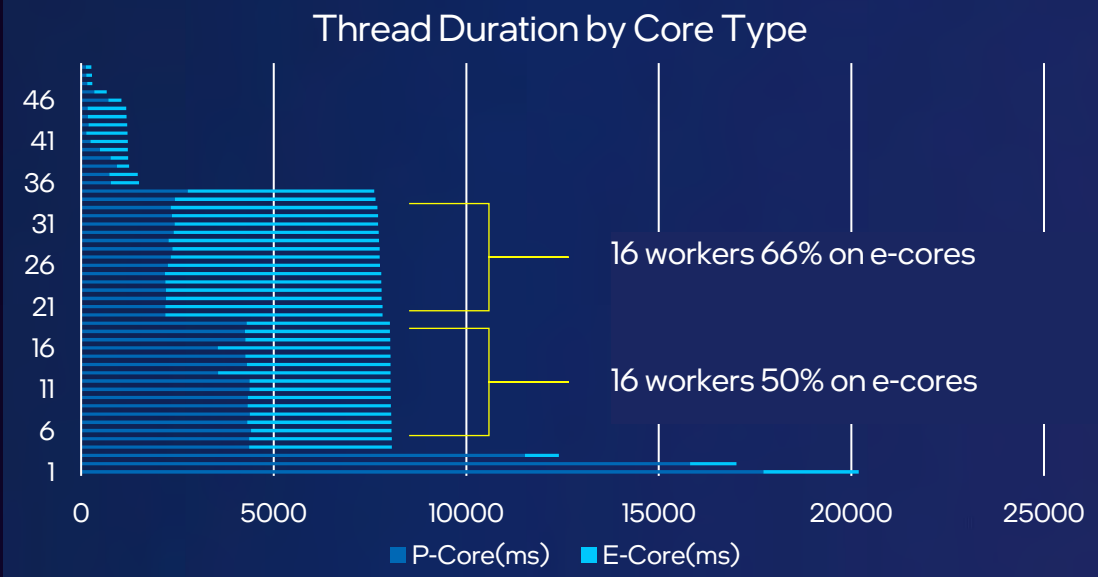
- Titles used are anonymous
- All data taken from titles un-optimised for Hybrid
- Data gathered during platform validation
- All titles give a good user experience on Hybrid
- Used purely to illustrate OS behaviour

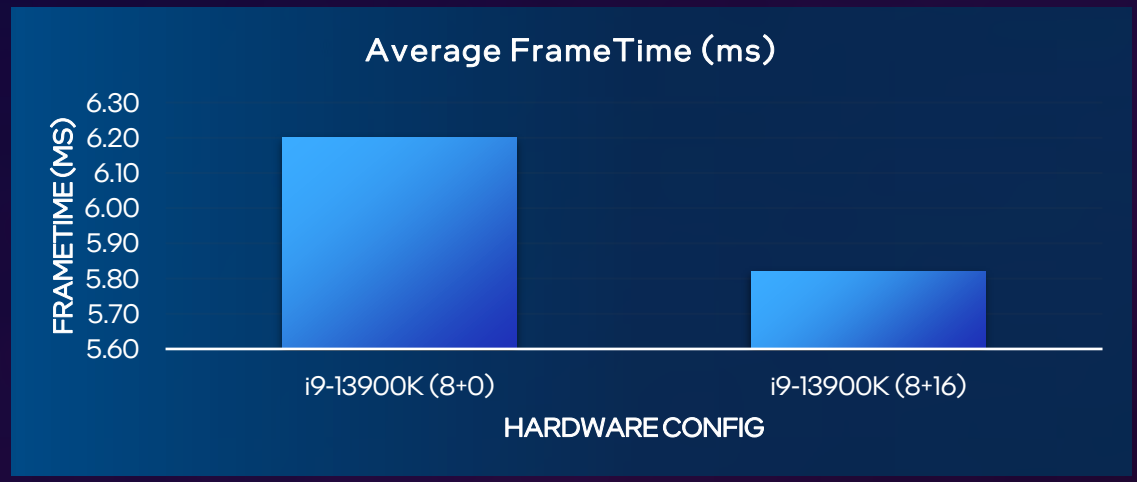
Case Study 1: Worker Threads on E-Cores

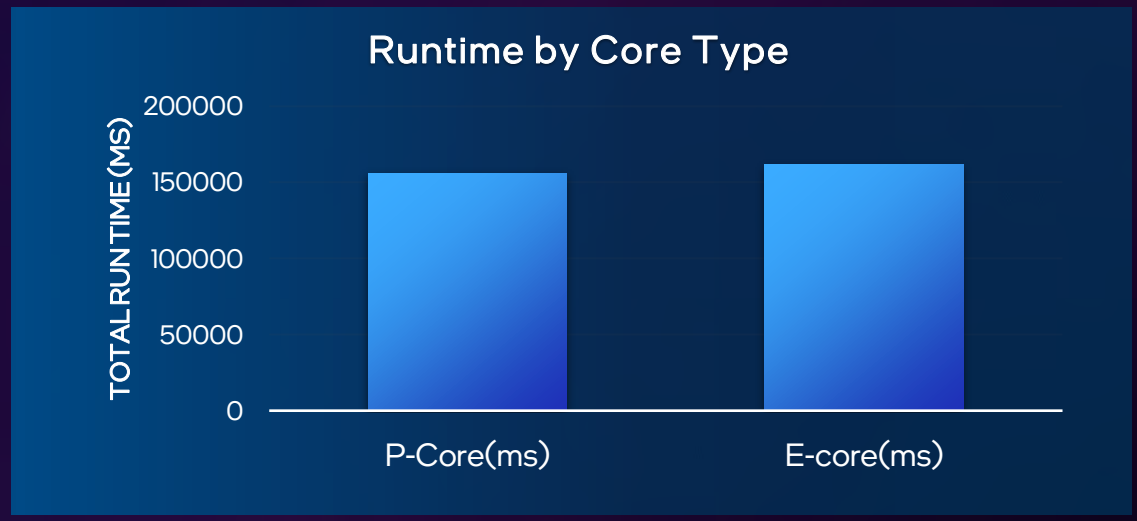
Problem statement:

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

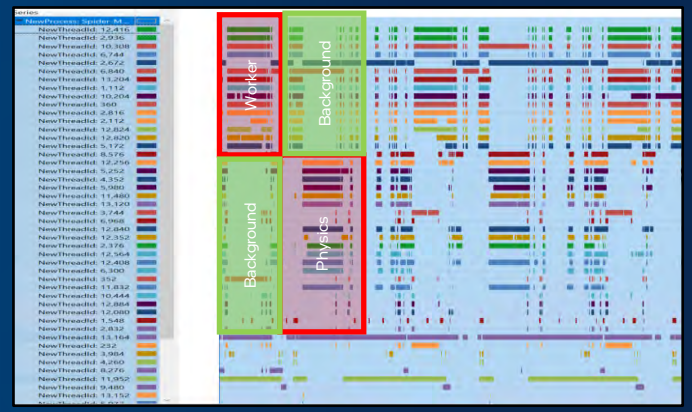
intel

Case Study 1: Non-Hybrid Behaviour

Engine created three thread-pools

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

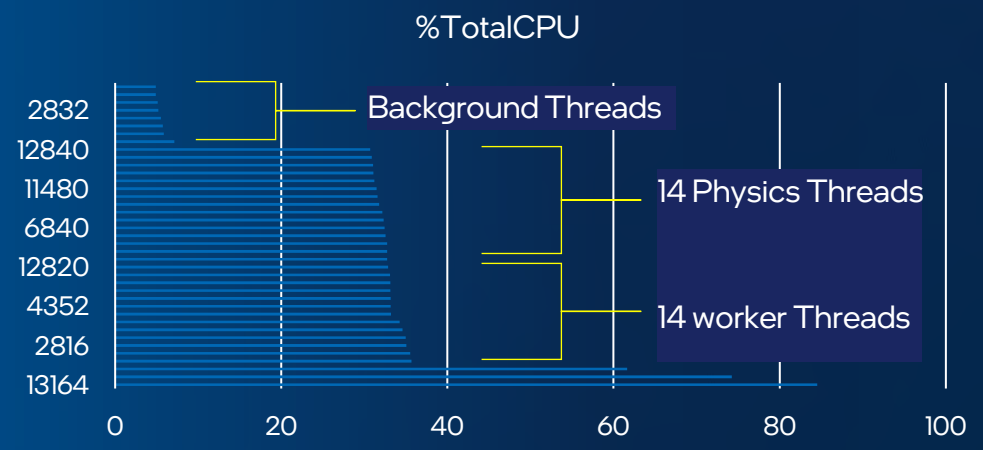
Worker threads run serially with physics, background fills in idle time.



Managed by thread priority.

Thread concurrency of 16:

- 14 Worker Threads + 2 Main threads



Case Study 1: Over reliance on Priority

GPUView App

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	9548.69	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	

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

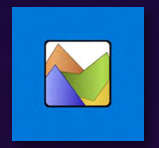
5,252

12,416

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

56

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

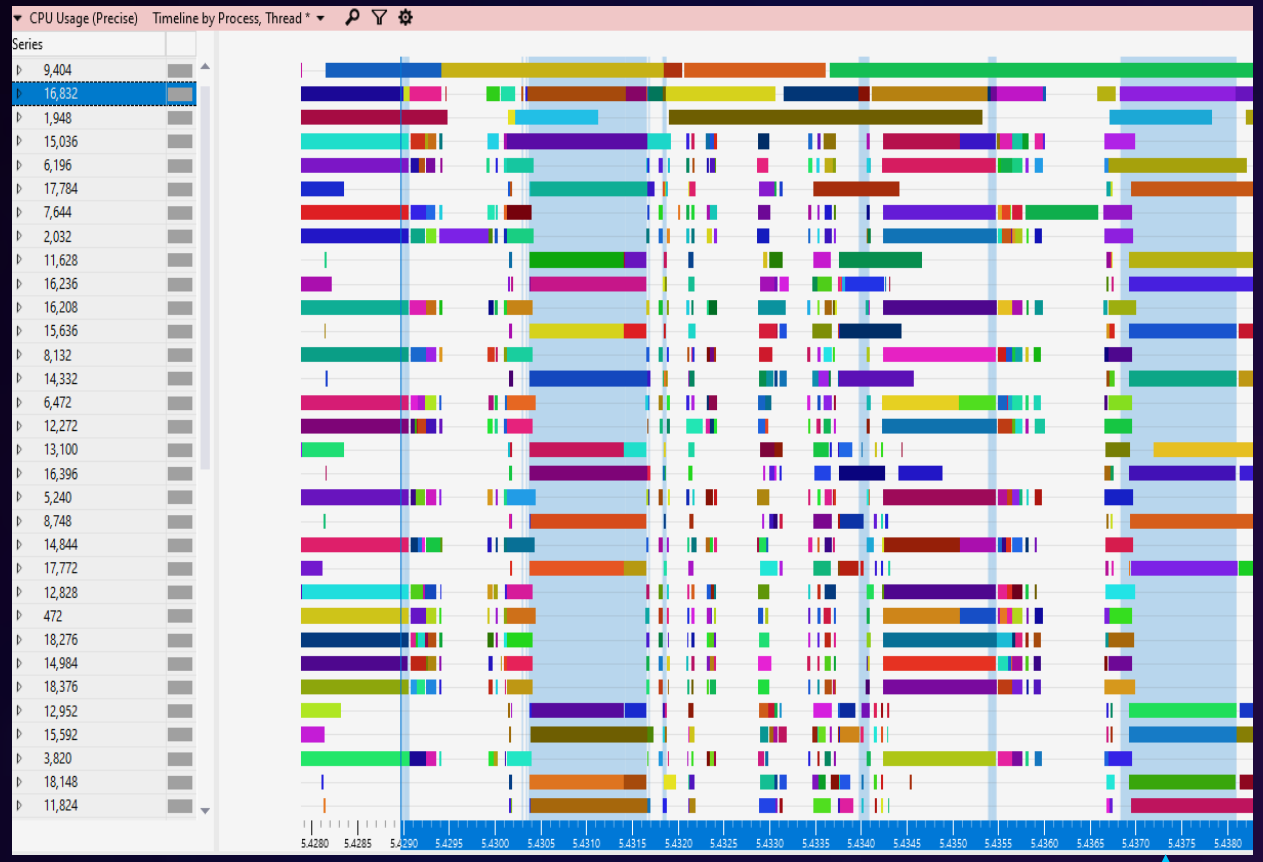


- Priority 9 Ready time drops 10x.
- Low priority threads don't have to wait.

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

P9 Threads

Case Study 1: Summary



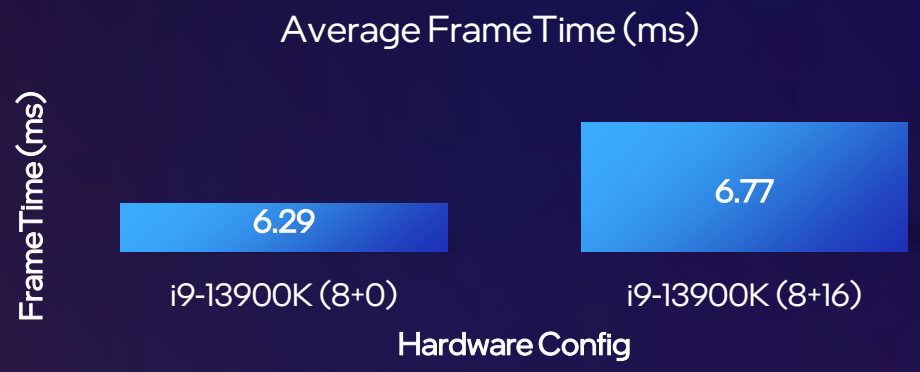
- Low priority work, runs in parallel with high priority work.
- High priority, long running threads, run on E-Cores when previous lower priority work is already in-flight on P-Cores.

- Could defer scheduling of priority 10 threads until after priority have started running.
- Could move background threads on to EcoQos

Thread 16,832 running on e-core

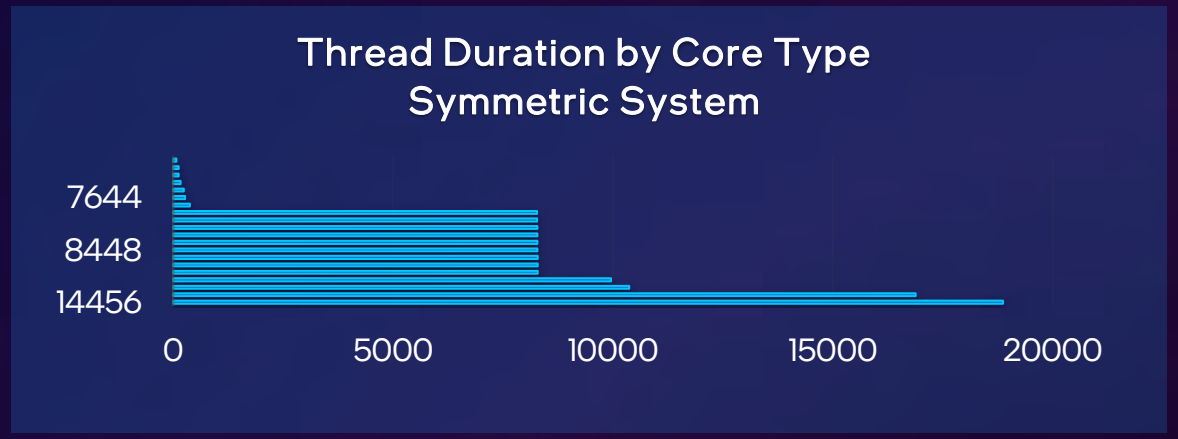
Case Study 2: Unclear Critical Path / Poor Multi-Threaded Scaling

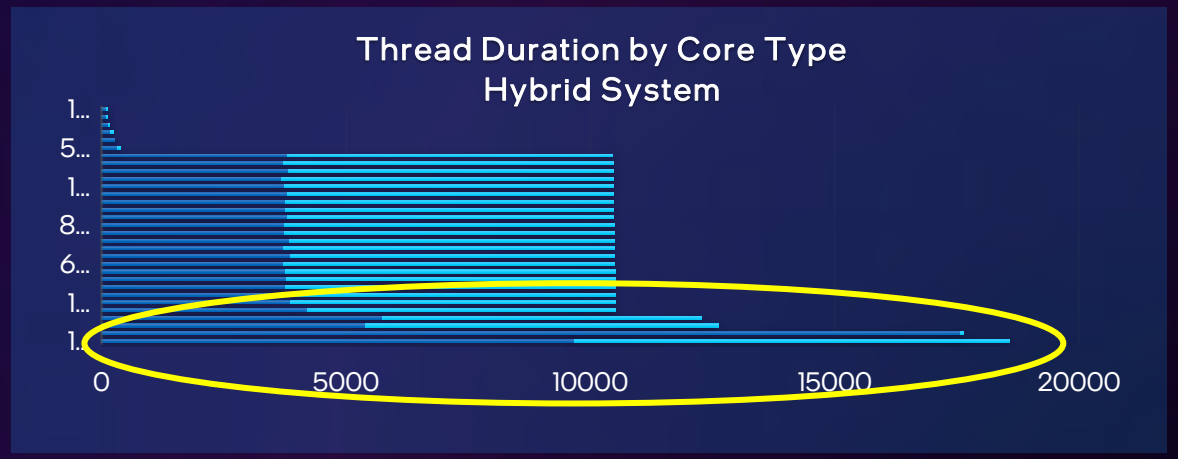
8% slower with E-Cores enabled



Two issues to investigate:

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





Thread creation based on logical processor count

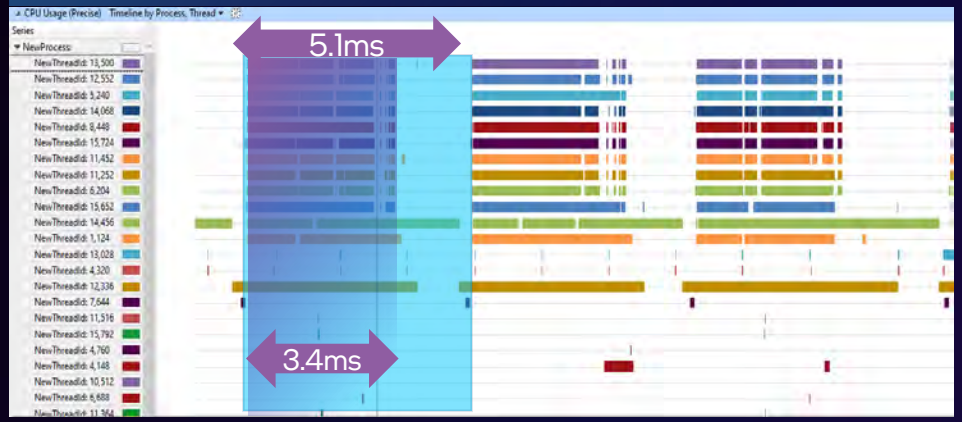
Disclaimer

- 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

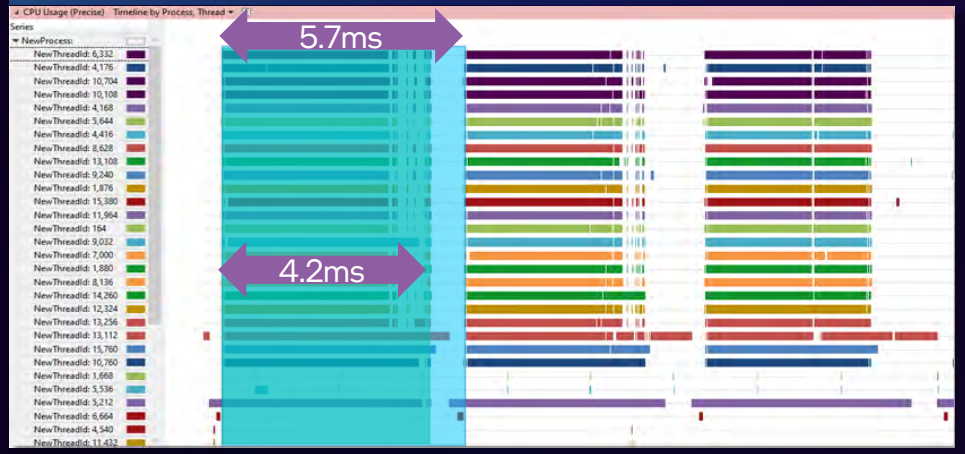
intel

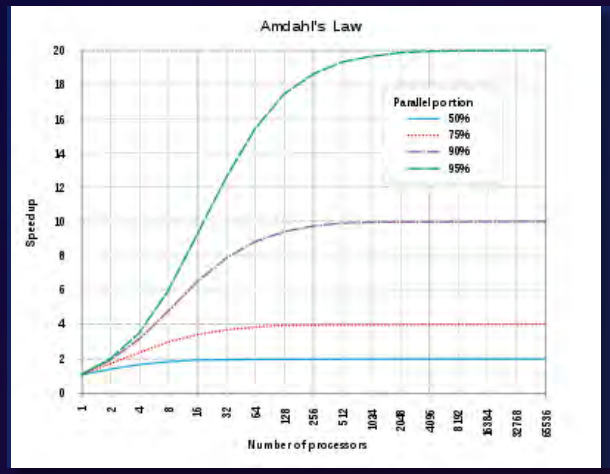
Case Study 2: Poor Multi-Threaded Scaling

Non-Hybrid System



Hybrid System





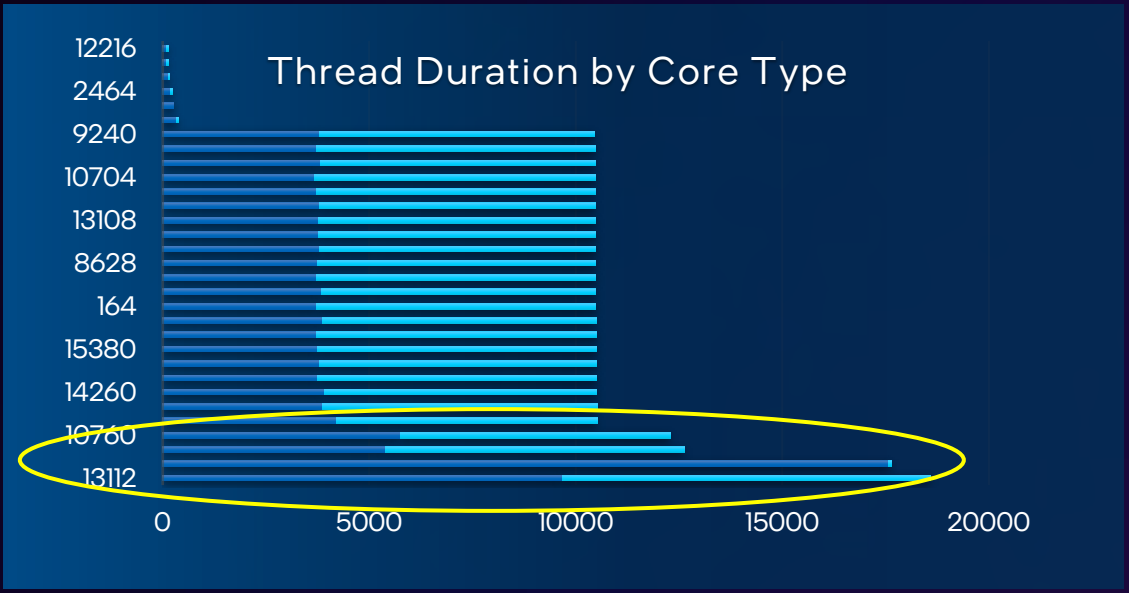
Amdahl's Law: 2x increase in cores should halve wall time.

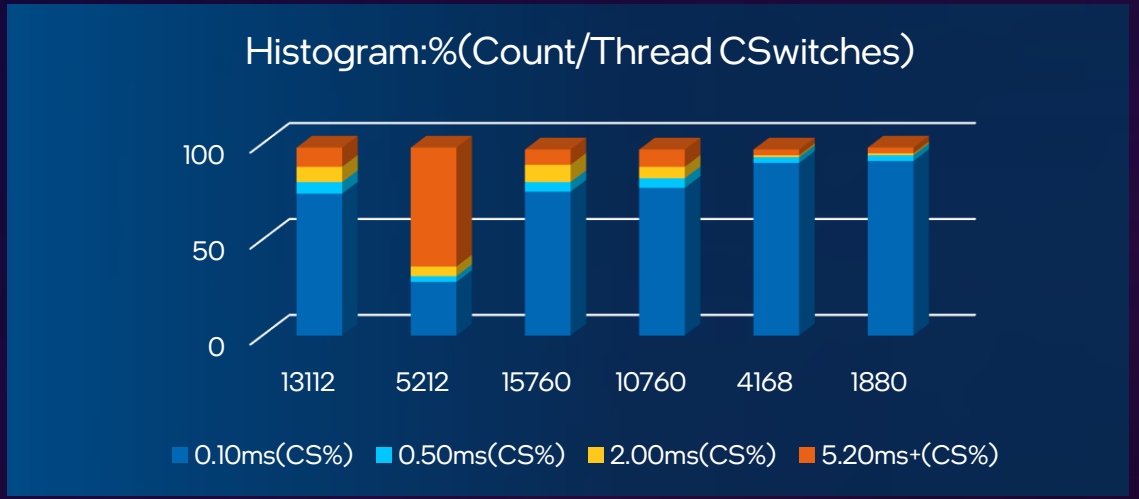
- But: 22 worker threads is slower than 10 worker threads.

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

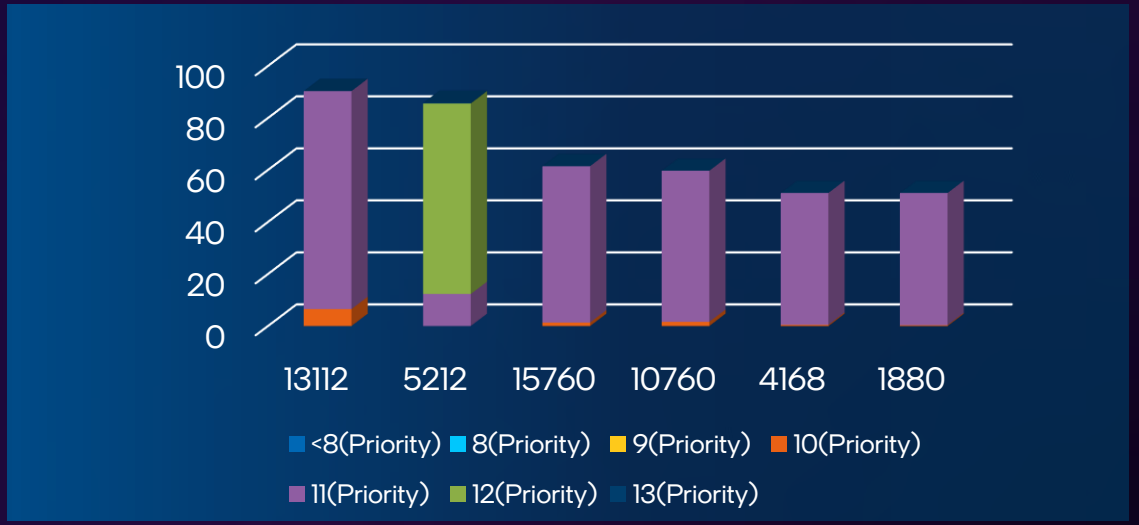
	Hybrid	Non Hybrid
metric_CPU operating frequency(GHz)	5.2460	5.3889
metric_CPI	2.3365	1.5610
metric_TMA_Backend_Bound(%)	77.0914	54.9688
metric_TMA_..Memory_Bound(%)	66.0355	42.5052
metric_TMA_....L3_Bound(%)	48.6328	23.5164

Case Study 2: Unclear Critical Path



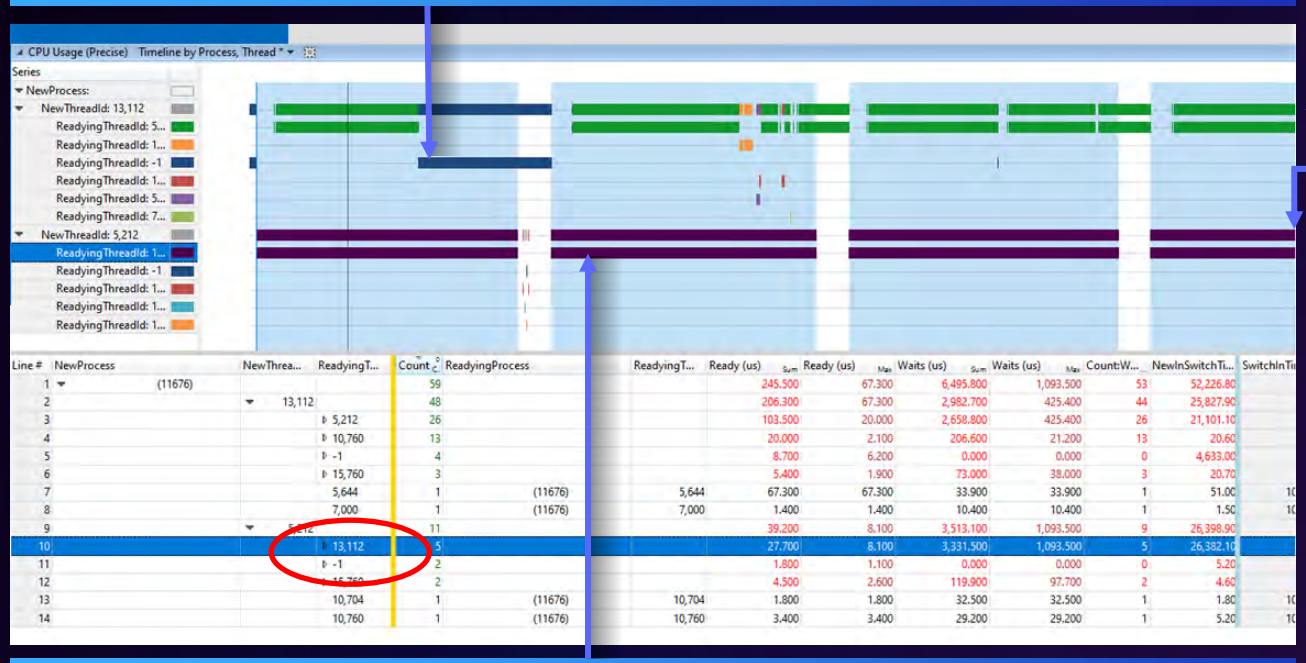


Thread 5,212 is high priority and stays on P-Cores.
Thread 13,112 looks the same as worker threads from the OS level.
Same priority 11 as workers.
80% short run time on thread wake up.



Case Study 2: Summary

Thread 13,112 context switches mid frame



Thread 5,212 waits on thread 13,112

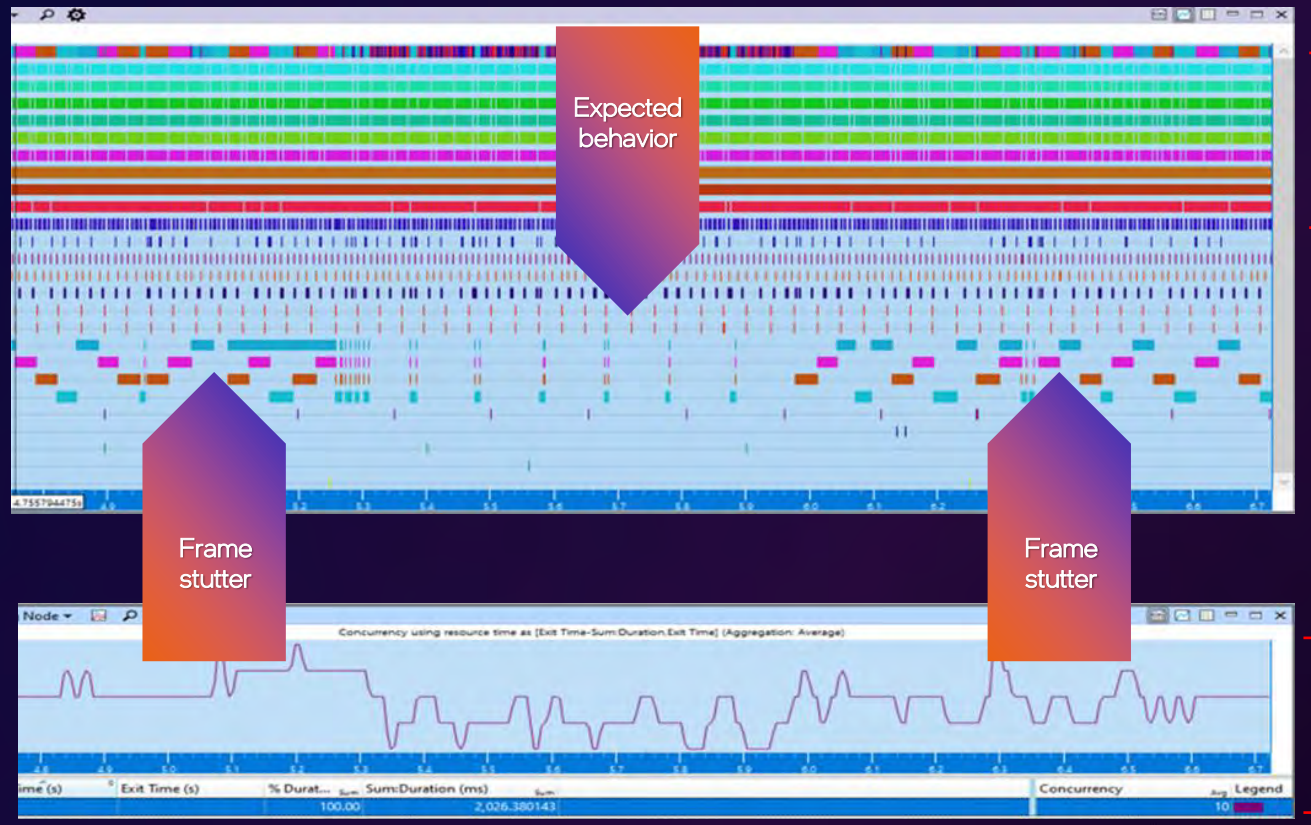
Two long running threads with hard dependency between them
context switch while being highly subscribed → high chance to schedule on an E-Core.
Time spent on E-Core is part of the critical path.

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

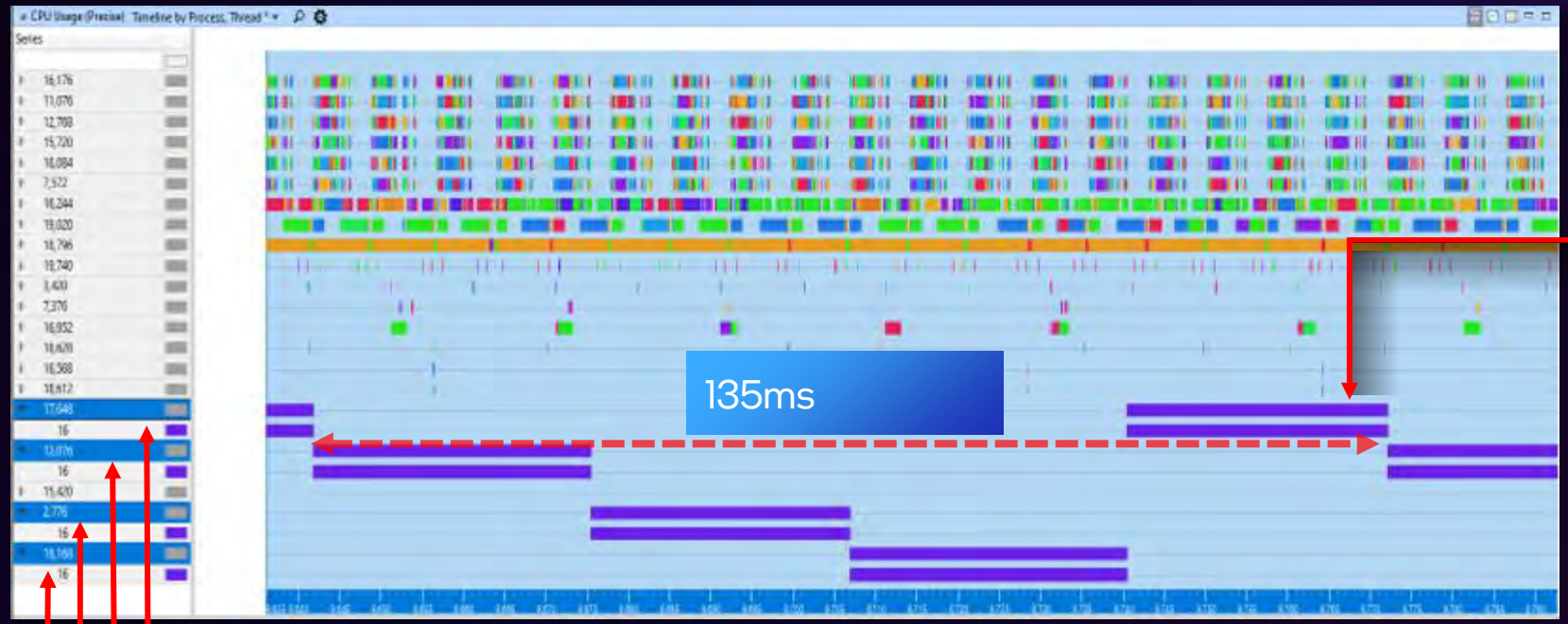
Case Study 3: Erratic Behaviour Over Time

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



Case Study 3: OS Forced Serialization

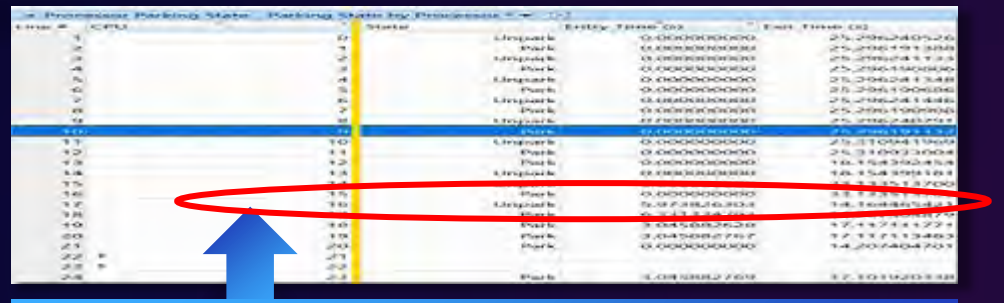


Why 35ms?

Thread switch at quantum end →
threads never yield

Producer and consumer threads serialised onto core 16

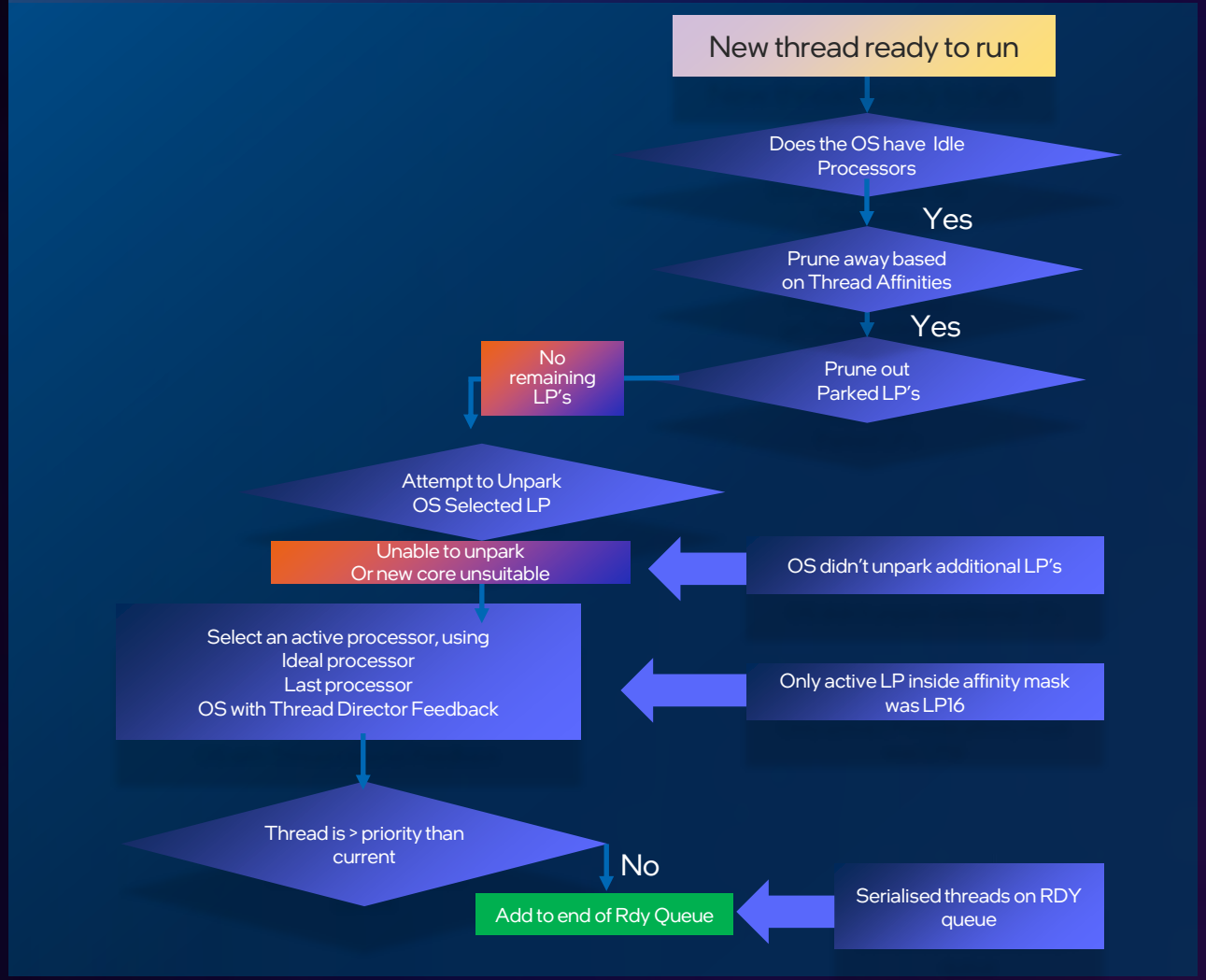
Line #	New Process	New Threa...	New Wait Reason	CPU
17		18,612	DelayExecution	
18		2,776	WrQuantumEnd	16
19		17,648	WrQuantumEnd	16
20				
21				
22		8,384	DelayExecution	0
23		15,420	UserRequest	8
24		18,168	WrQuantumEnd	16
25		12,076	WrQuantumEnd	16



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

Case Study 3: Summary

Remember this??



Frame rate stutter linked to core parking

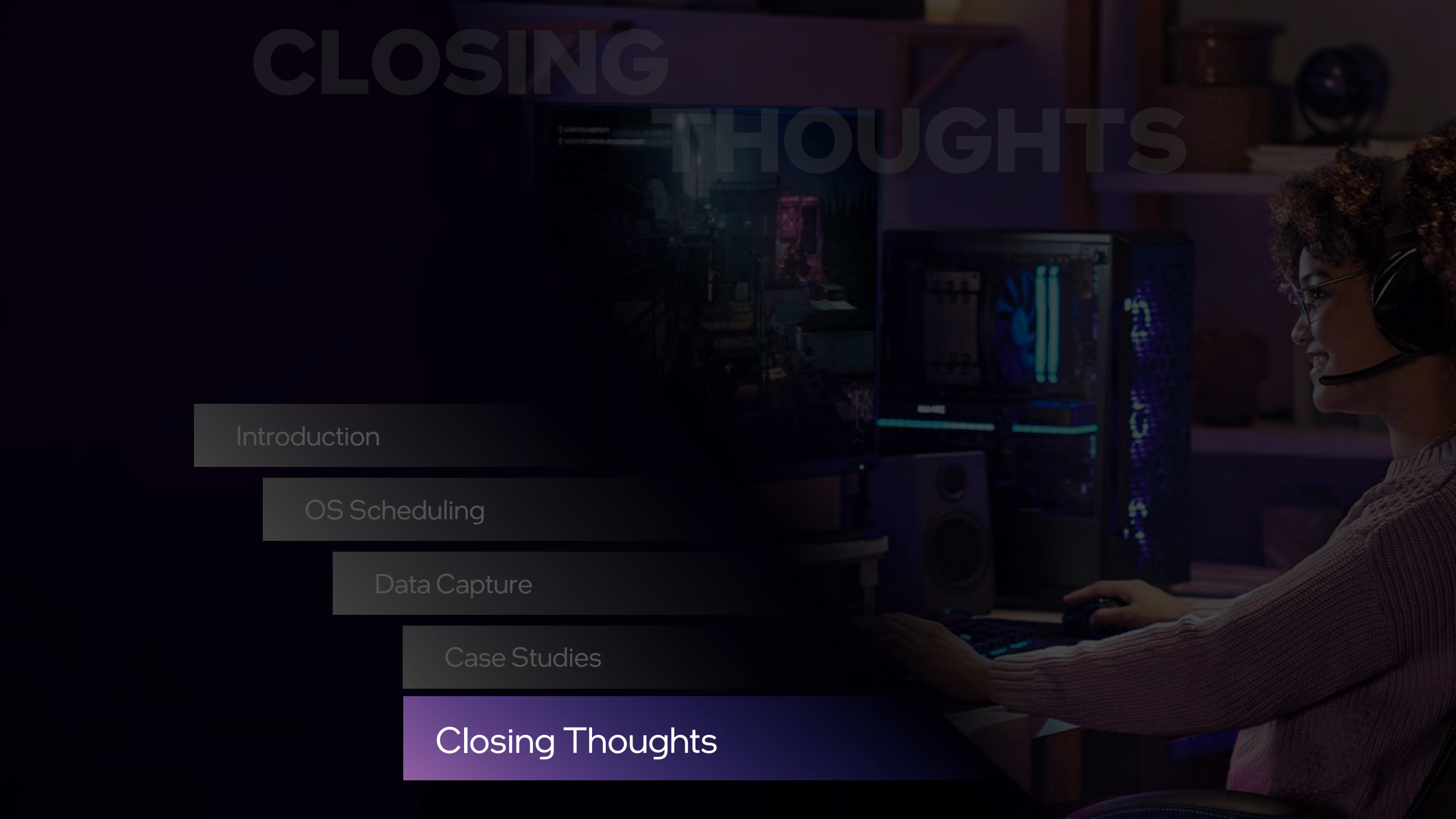
Thread's Ideal Processor was outside the thread affinity mask.

- 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

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

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

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

Avoid pinning threads to a single logical processor

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

Understand how your middleware uses threads

Thank you

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at 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.

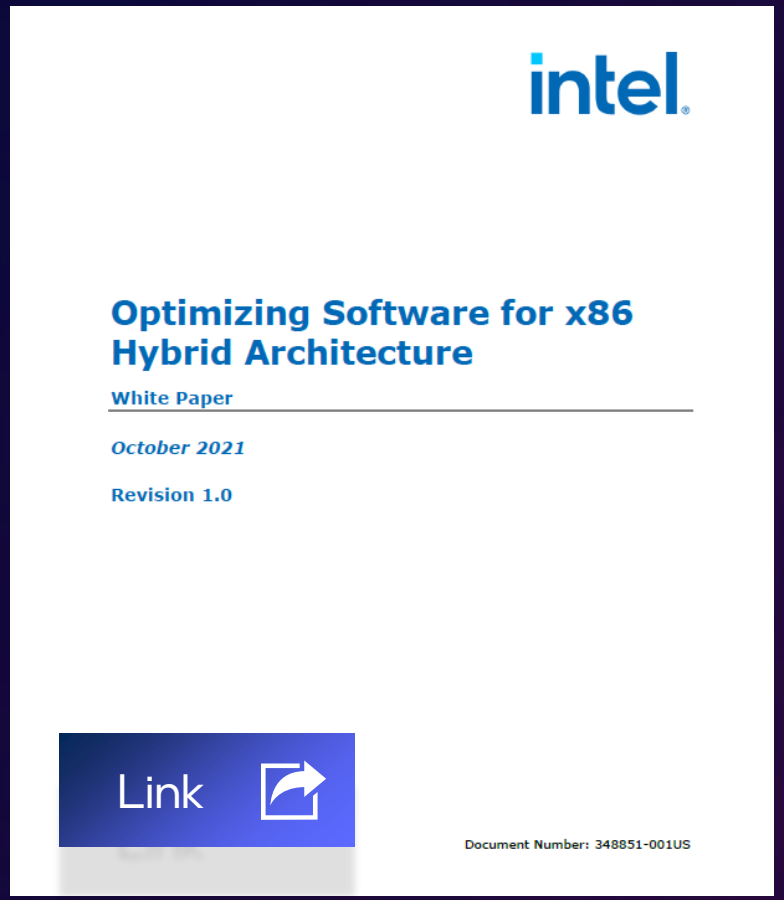
Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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.


intel
Optimizing Software for x86 Hybrid Architecture
White Paper
October 2021
Revision 1.0
Link
Document Number: 348851-001US


intel