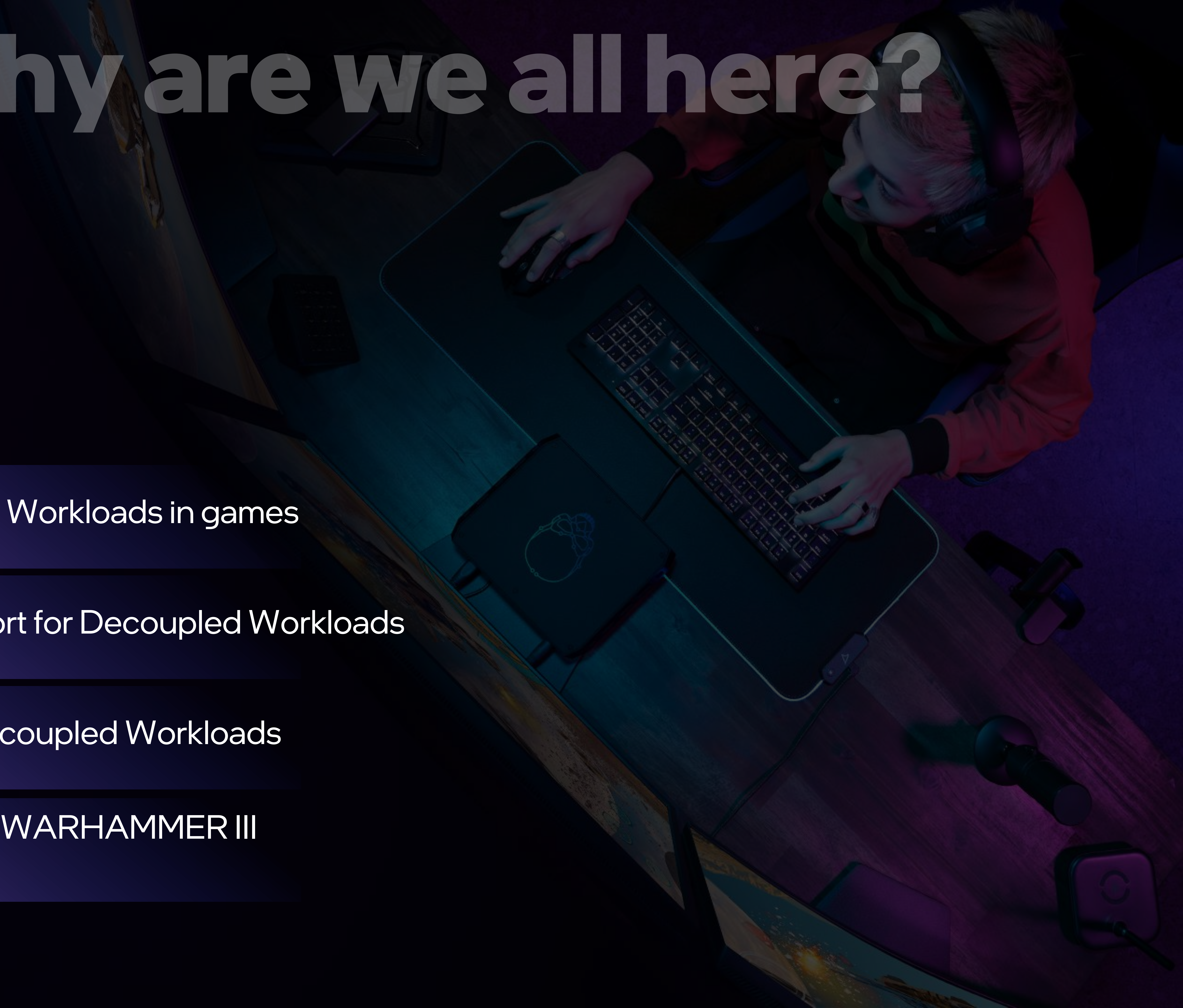
Optimizing for Hybrid Total War: WARHAMMER III

Steve Hughes, Intel
Scott Pitkethly, Creative Assembly

03/2023

Why are we all here?



- Introduction to Decoupled Workloads in games
- Hybrid Architecture support for Decoupled Workloads
- Implementation tips for Decoupled Workloads
- Learnings from Total War: WARHAMMER III by Creative Assembly

Why are we all here?

- Introduction to Decoupled Workloads in games
- Hybrid Architecture support for Decoupled Workloads
- Implementation tips for Decoupled Workloads
- Learnings from Total War: WARHAMMER III by Creative Assembly

Decoupled Workloads 101

Decoupled workload is run on a separate thread AND at a different FPS to the main game.

Key Frame 12 is created by Workload A.

Key Frame 20 is created by Workload B.

Frames 13 to 19 lerp from Key Frame 12 to 20.

Decoupled Workloads 101b

Some game frames run at the same time as the workloads, some not.

Frames 3→6 and 11→14 and 19→20 will execute more quickly due to no overlap with Decoupled Workloads.

Overall, the frame times are still faster.

Ideal Candidate Decoupled Workloads

Discrete subsections of the current frames CPU work running at frame speed...

Generation of animation stacks	Route planning for AI units	Line of sight

Construction of larger assemblies spreading over many frames...

Complex scene element creation	Model and texture streaming	Level loading

Game subsystems...

AI	Physics	Weather & fluid simulation

Decoupled Workload Advantages

Workload does not increase linearly with framerate.

Can easily run single tasks that take multiple frames to complete.

No need for messy RK2/3/4, Midpoint etc to get correct results for AI and Physics - no more “where did 1/2at^2 go???”.

Decoupled Workloads Pitfalls

Can lead to **irregular frame times.**

Can cause game lag if used on the wrong game components.

Can cause stepping for nonlinear acceleration.

For the rest of this talk we are interested in **irregular frame times.**
Let's look more closely at why this happens.

Decoupled Workloads 101c

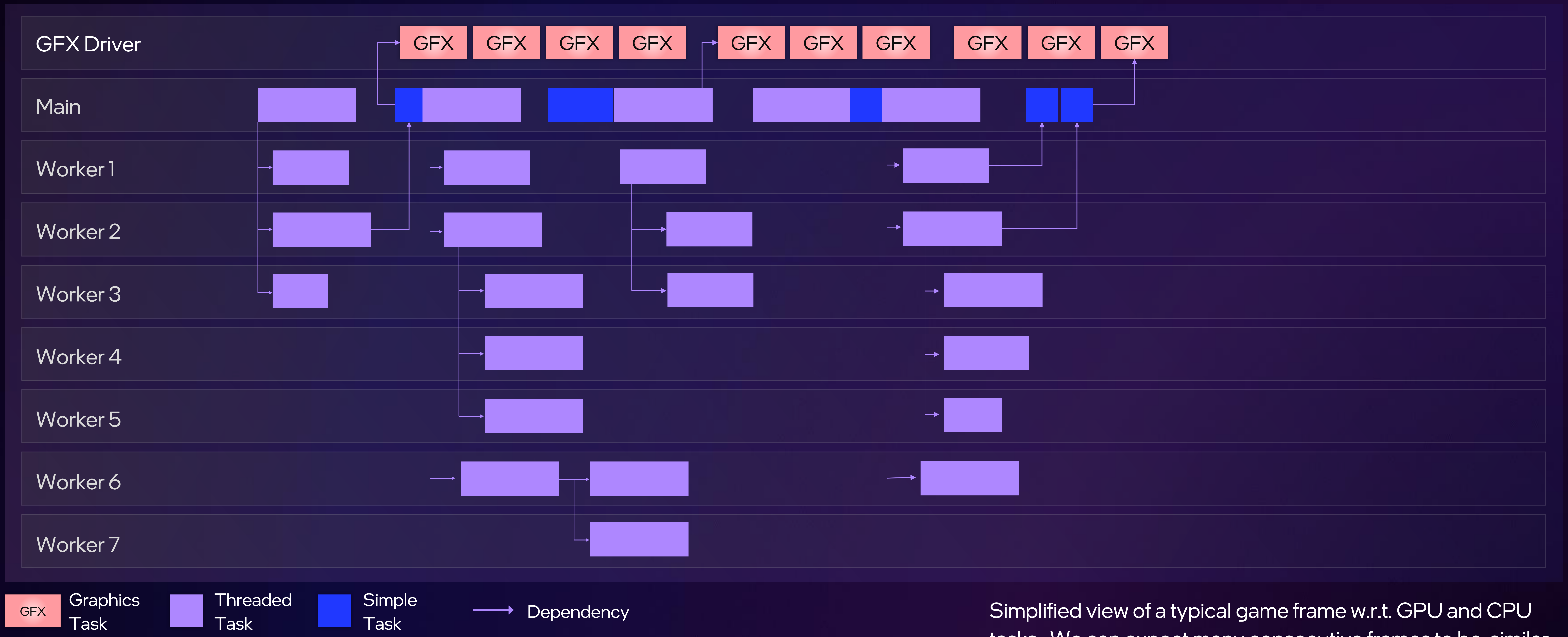
Some frames run at the same time as the Decoupled Workload

All frames that overlap with Decoupled Workload have:

Extra work. | Potential cache pollution.

This extends their execution time.

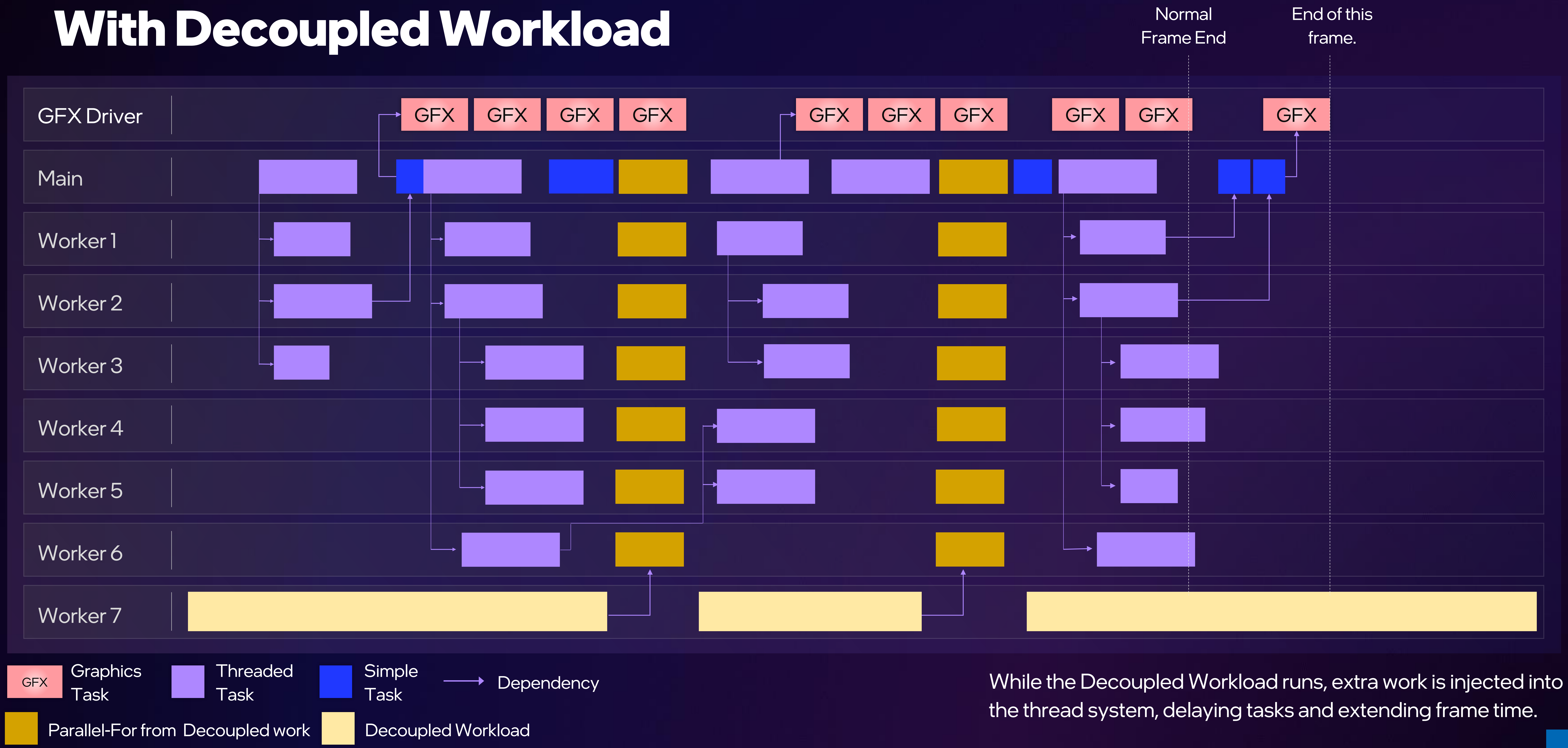
Functional View of a typical frame



Simplified view of a typical game frame w.r.t. GPU and CPU tasks. We can expect many consecutive frames to be similar.

Functional View of a typical frame

With Decoupled Workload



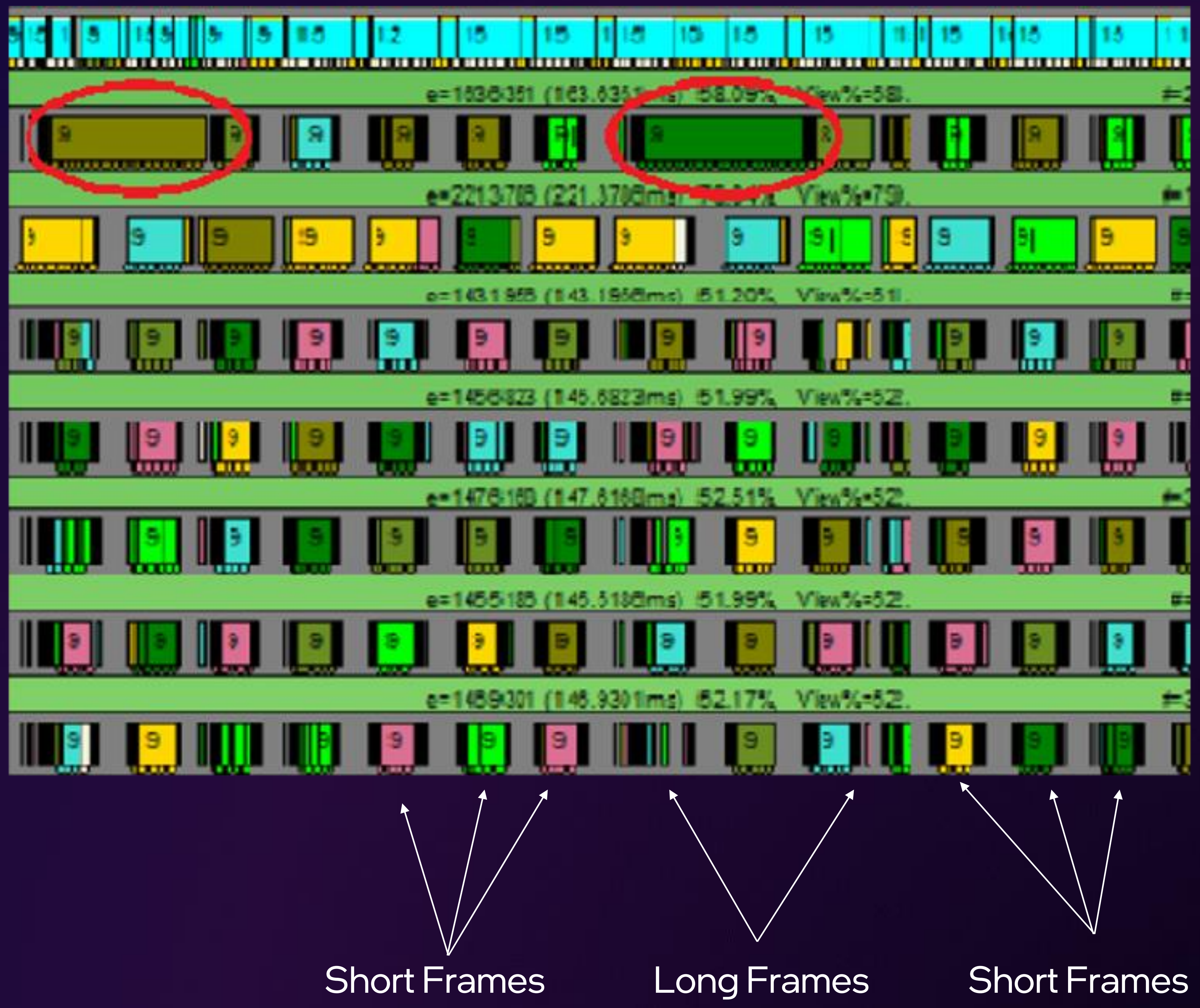
While the Decoupled Workload runs, extra work is injected into the thread system, delaying tasks and extending frame time.

A Decoupled Workload in the Wild

GPUView

- Allows us to visualize our CPU use.
- Rows are our threads.
- Boxes are tasks or groups of tasks.
- Colors represent CPU cores.

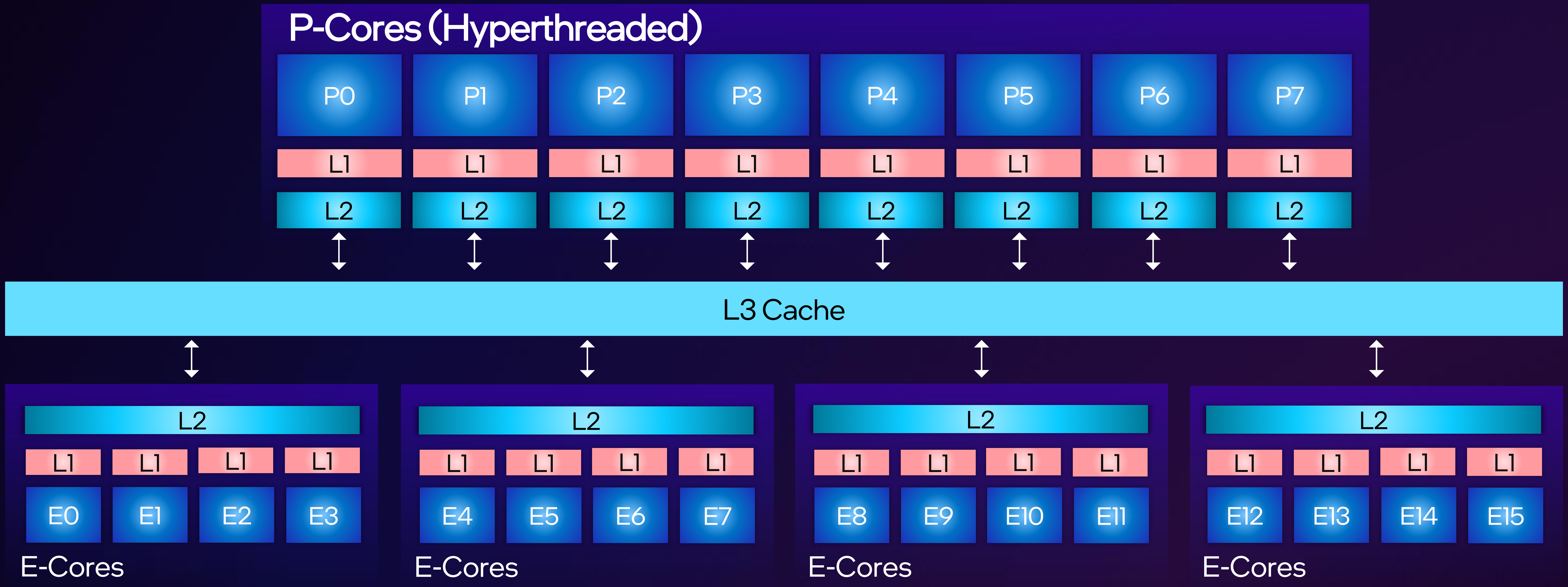
During the Decoupled Workload, frames are 2-3ms slower



Why are we all here?

- Introduction to Decoupled Workloads in games
- **Hybrid Architecture support for Decoupled Workloads**
- Implementation tips for Decoupled Workloads
- Learnings from Total War: WARHAMMER III by Creative Assembly

What is a Hybrid Processor (10000ft view)



P0 thru P8 → Performance Cores

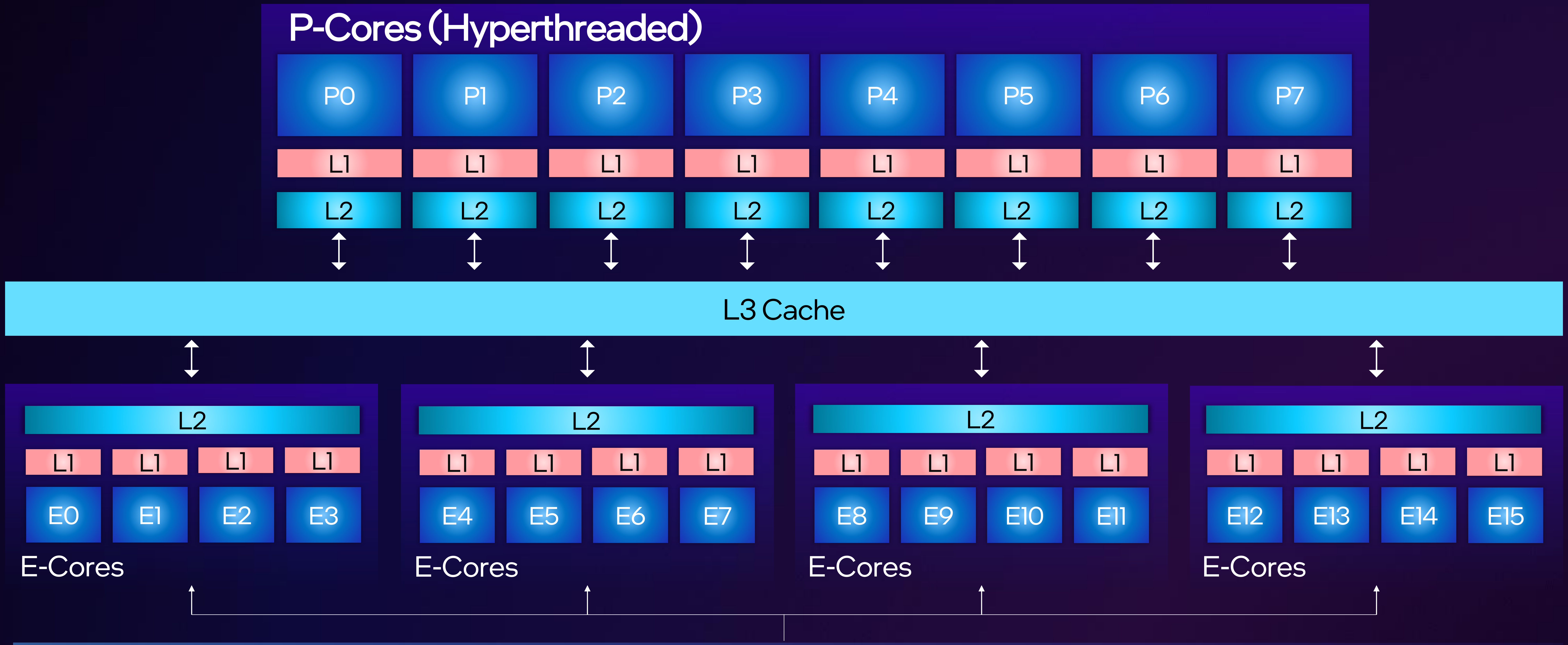
E0 thru E16 → Efficient Cores

L1 First level cache

L2 Second level cache

Each E-Core is equivalent to a Skylake core from a few years back!

Realization



Any of these groups of 4 E-Cores would be a great place to run a Decoupled Workload:

- Won't steal CPU time from P-Cores
- Less tendency to pollute cache

Why are we all here?

Introduction to Decoupled Workloads in games

Hybrid Architecture support for Decoupled Workloads

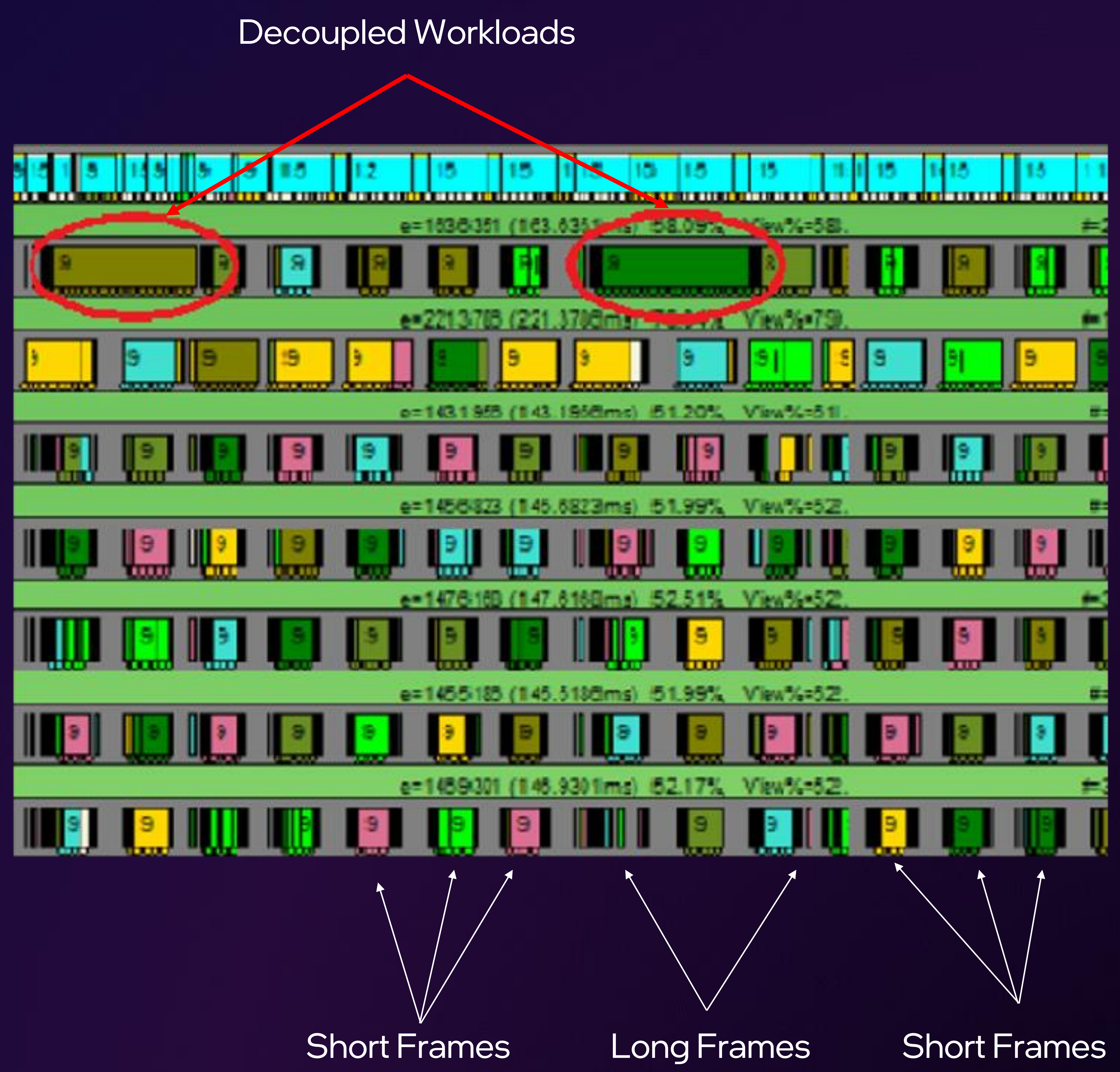
Implementation tips for Decoupled Workloads

Learnings from Total War: WARHAMMER III
by Creative Assembly

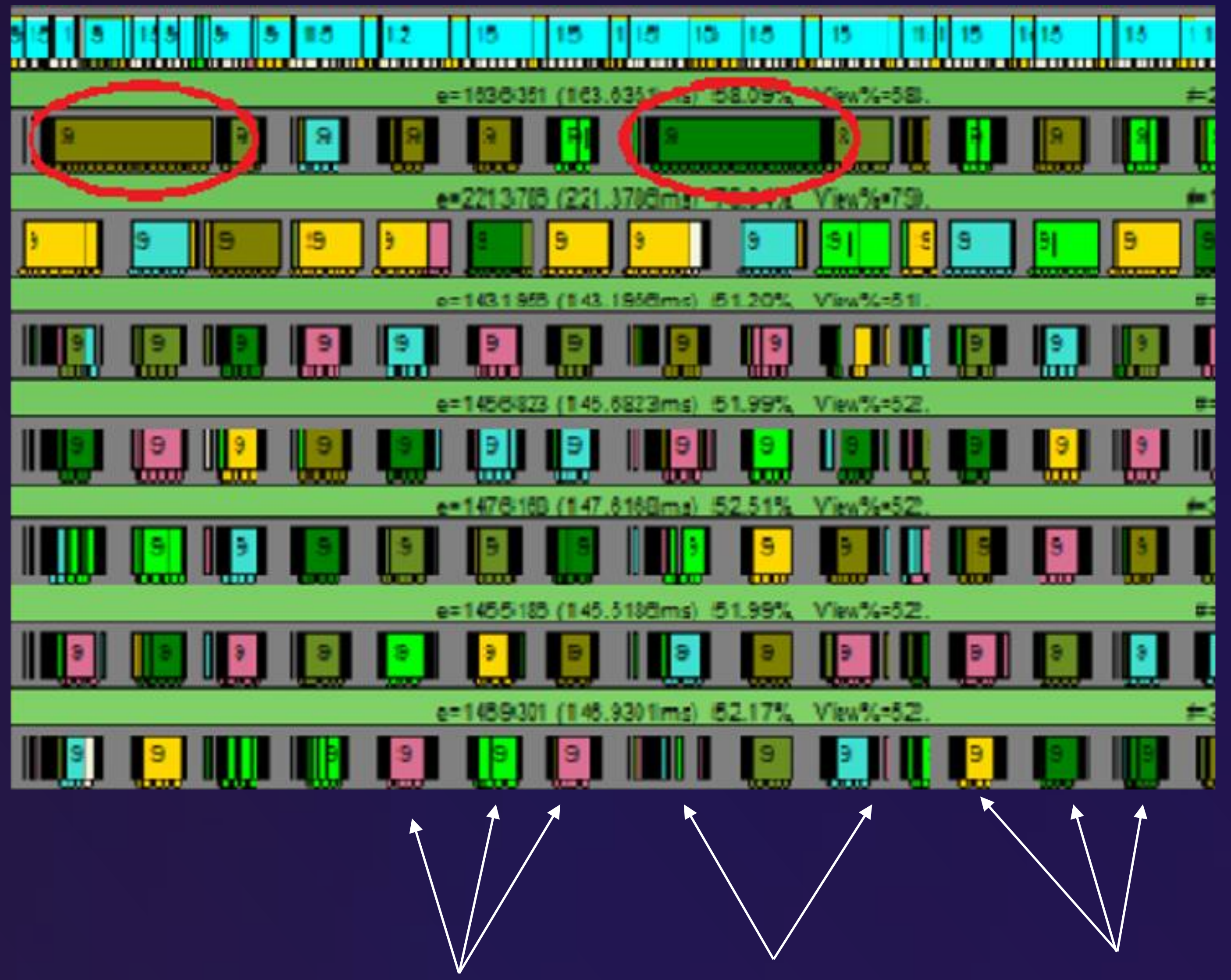
A Decoupled Workload in the Wild

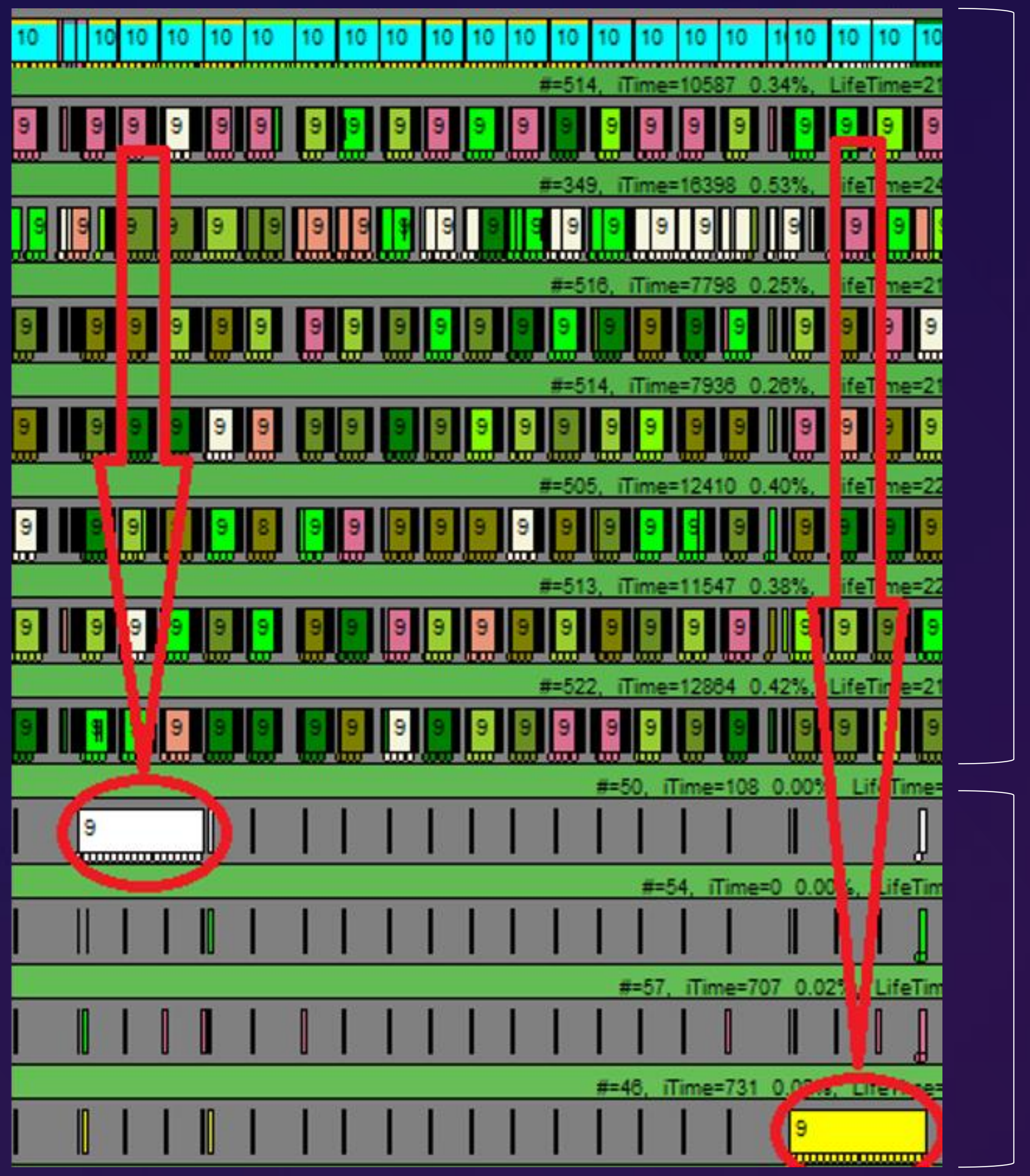
GPUView

- Allows us to visualize our CPU use.
- Rows are our threads.
- Boxes are tasks or groups of tasks.
- Colors represent CPU cores.



Moving a P-Core Decoupled Workload to E-Cores





Decoupled Workloads were causing irregular frame rates due to:

- Supporting code in other threads.
- Multithreaded flow interference.
- Cache interference.

Choosing our method

There are 2 ways we can implement this solution where we restrict which threads can run on which cores:

- We can use Affinity or,
- We can use CPUSets.

To use affinity, we call SetThreadAffinityMask(). We pass in the thread handle and a bit vector with 1 bit for each core. Set the bit to one if you want the thread to run on that core, zero if you don't.

- This is a Hard restriction on the OS.
- It is not compliant with power management schemes.
- Reduces the OS' ability to manage workloads.

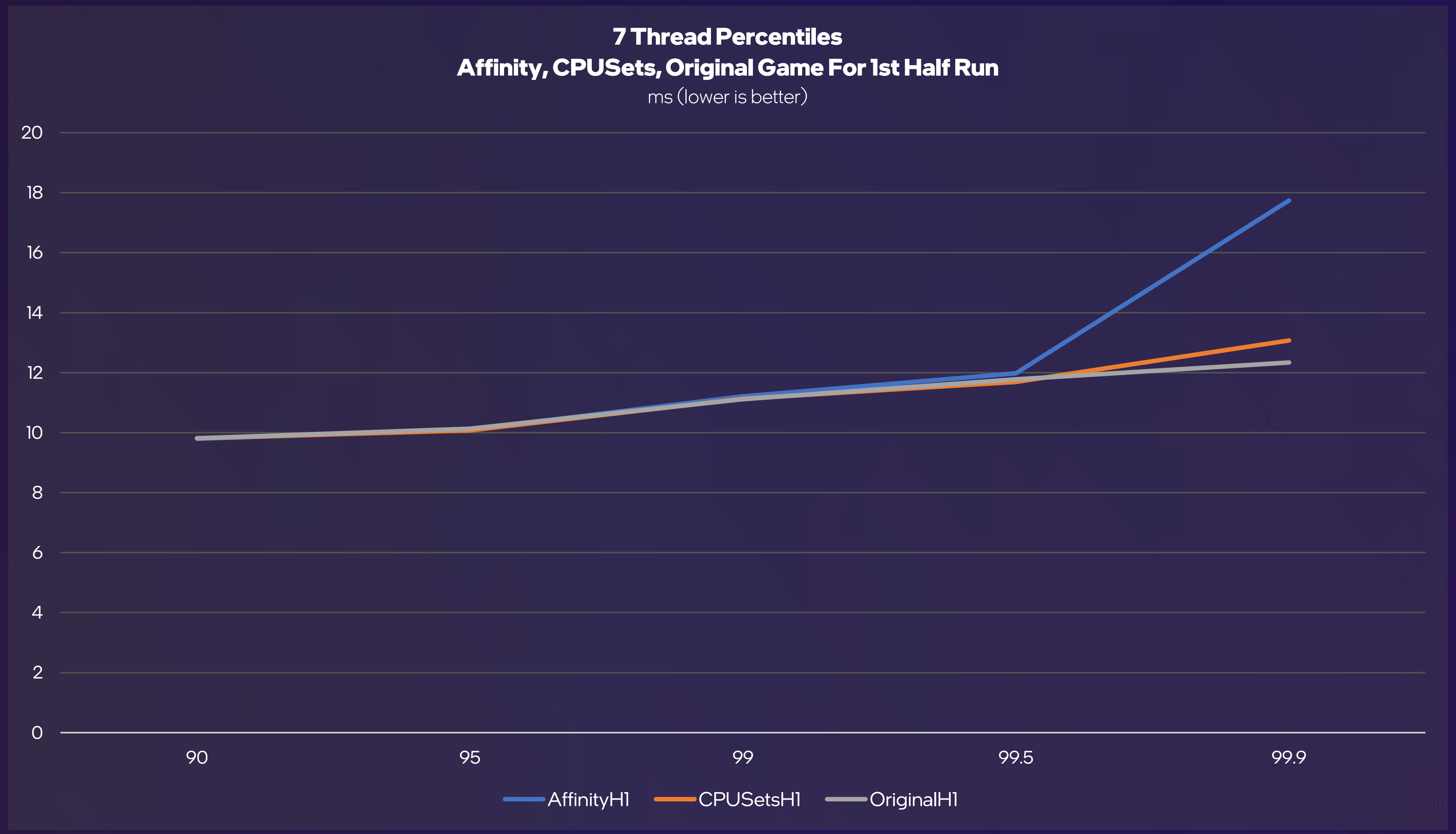
CPUSets is a relatively new API. To use it, we give it similar info to SetThreadAffinityMask() using SetThreadSelectedCPUSets().

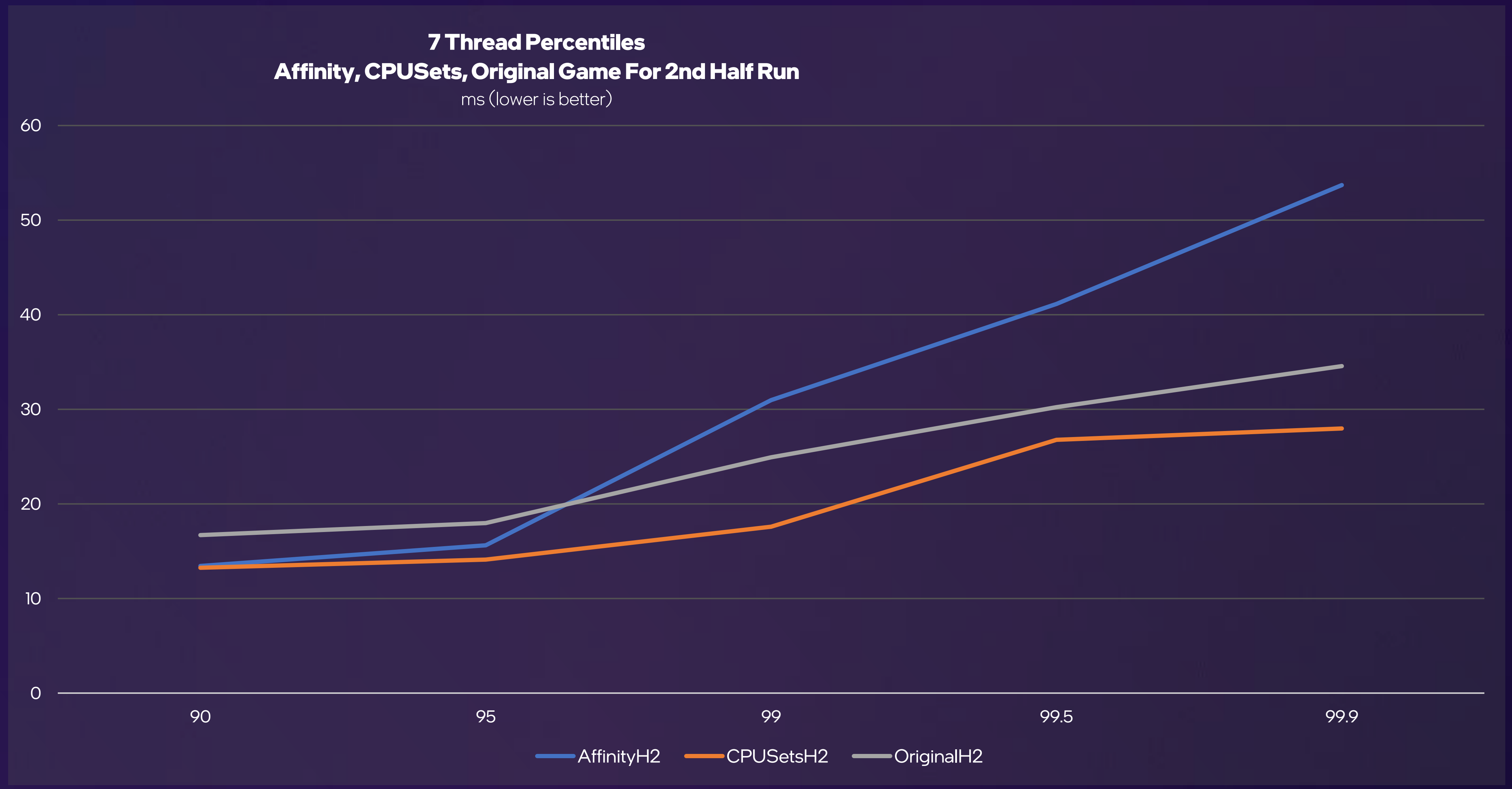
- Sets a Soft affinity.
- OS can balance work more easily.
- The full API is well documented on MSDN.

Findings

We implemented both the Affinity and the CPUSets methods to implement Decoupled Workloads on the E-Cores and compare the results.

- Both methods would cause a small improvement in framerate ~5%*.
- CPUSets gave us the predicted improvement in framerate smoothness.
- Affinity reduced the framerate smoothness at higher workload.





*Results from internal binary which may not exactly match final released version of the game.

intel

Conclusion

- E-Cores are an ideal fit to execute Decouple Workloads.
- E-Cores are great to run extra content that would normally not be enabled due to performance concerns.
- As the developer, you have the best context on your workloads and so are best placed to judge the benefits of decoupling in your game.

Why are we all here?

- Introduction to Decoupled Workloads in games
- Hybrid Architecture support for Decoupled Workloads
- Implementation tips for Decoupled Workloads
- **Learnings from Total War: WARHAMMER III by Creative Assembly**

OPTIMISING FOR HYBRID CPUS:

THEORY TO PRACTICE

INTRODUCTION

- Scott Pitkethly
- Technical Director on Total War : Warhammer 3
- Been working on battles for more than 2 decades
- Typing this out puts things into perspective!

TOTAL WAR - REAL-TIME BATTLES
OVERVIEW

- Turn-based campaign with real-time battles
- 1000s of entities updated in real-time
- Projectiles in game also fully simulated
- Combat, line of sight calculations, animation constraints, calculated for every entity each tick
- The logical state of the battle we refer to as the Battle Model

TOTAL WAR - REAL-TIME BATTLES
OVERVIEW - BATTLE MODEL

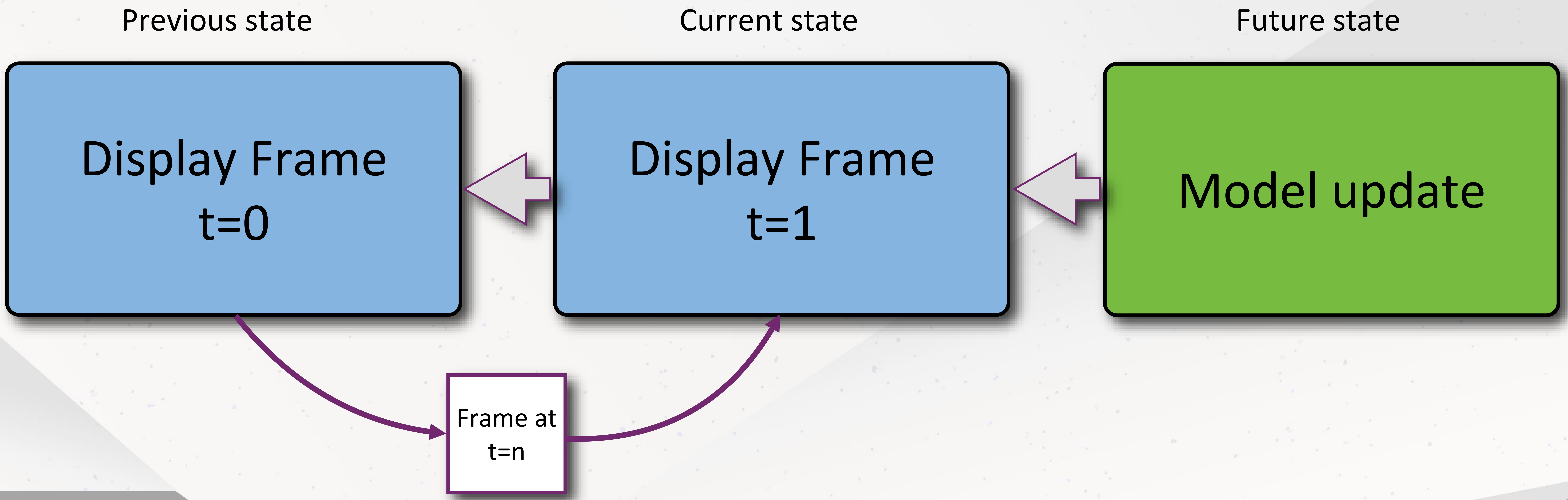
- Can’t ‘lod’ this update in order to preserve deterministic state
- Our multi-player model requires us to enforce a deterministic state on all machines
  - Only the player orders are broadcast
  - This means we can’t reduce the complexity of logical updates based on camera position etc
- Key point here is we update the logical state (or battle model) at a lower frequency than the frame rate

BATTLE MODEL (LOGIC)

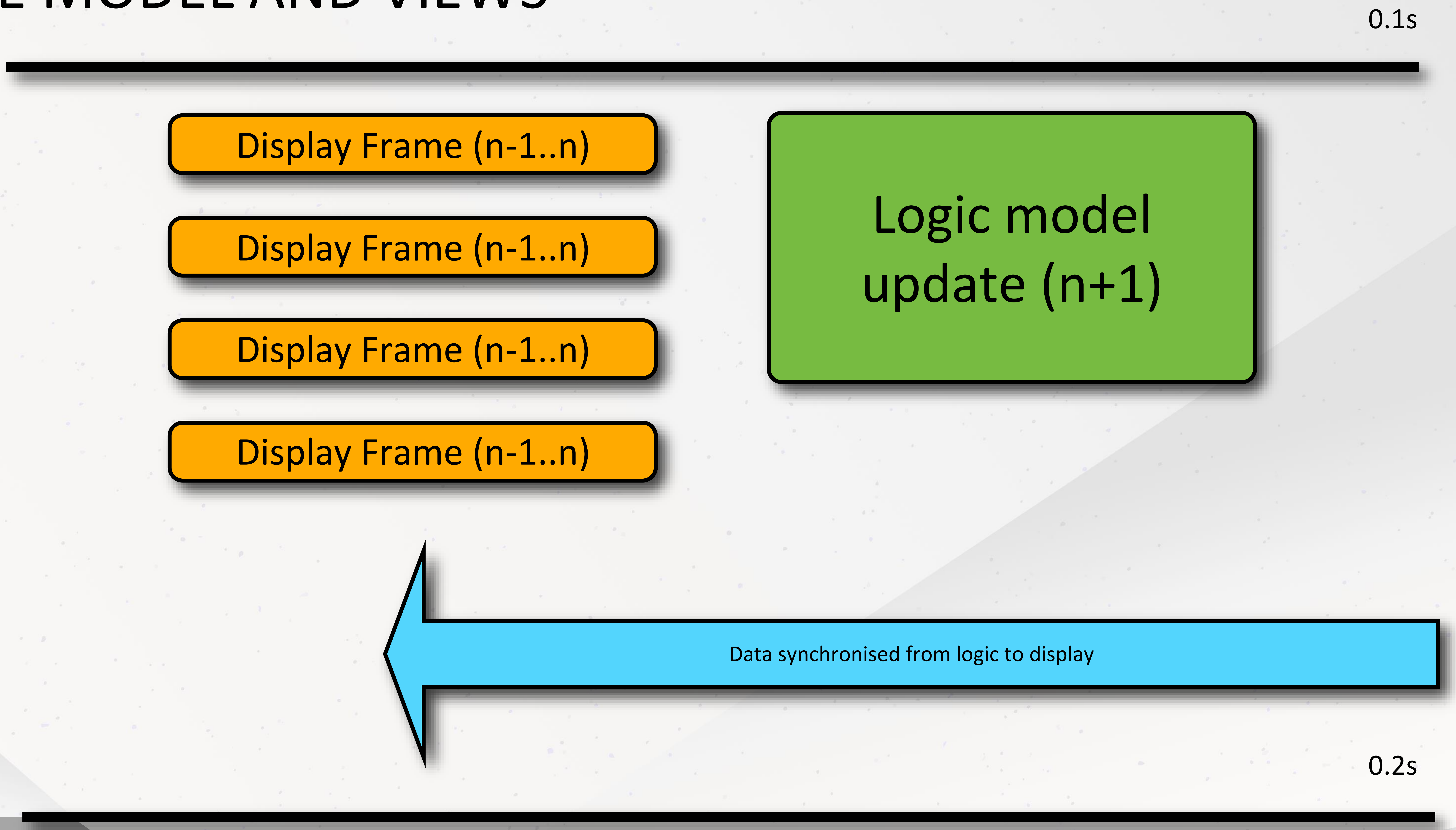
- Has a ‘slow’ update rate between 5-10fps (dependent on game type)
- No motivation to complete this update faster than tick time. Only go wide when necessary, otherwise you are robbing CPU time from views
- Seemed like a good initial candidate to run on the e-cores
- Keep the p-cores freed up for per-frame calculations
- Conserve energy

BATTLE MODEL (LOGIC)

Display interpolates between previous state to current state whilst the Battle Model creates the state for the future frame



BATTLE MODEL AND VIEWS



TASK SYSTEM

- Bespoke task system using fibres
- Create worker threads based on number of cores (one per physical core on standard architectures)
- Each worker thread has its own task queue
- Worker threads can steal tasks from each others queues when idle
- Workers sleep when no tasks available
- Battle model tick is a task. Create child tasks via parallel_for (when necessary)
- Internal affinity to stop large tasks (render thread task and battle model) ending up on the same thread

QUERYING THE CPU

- Use CPUSets to determine CPU topology and detect hybrid architectures
- CPUSets windows API - `GetSystemCpuSetInformation()`
- EfficiencyClass - this is the value that groups e/p-cores

```
typedef struct _SYSTEM_CPU_SET_INFORMATION {
  DWORD                     Size;
  CPU_SET_INFORMATION_TYPE Type;
  union {
    struct {
      DWORD   Id;
      WORD    Group;
      BYTE    LogicalProcessorIndex;
      BYTE    CoreIndex;
      BYTE    LastLevelCacheIndex;
      BYTE    NumaNodeIndex;
      BYTE    EfficiencyClass;
      union {
        BYTE AllFlags;
        struct {
          BYTE Parked : 1;
          BYTE Allocated : 1;
          BYTE AllocatedToTargetProcess : 1;
          BYTE RealTime : 1;
          BYTE ReservedFlags : 4;
        } DUMMYSTRUCTNAME;
      } DUMMYUNIONNAME2;
      union {
        DWORD Reserved;
        BYTE  SchedulingClass;
      };
      DWORD64 AllocationTag;
    } CpuSet;
  } DUMMYUNIONNAME;
} SYSTEM_CPU_SET_INFORMATION, *PSYSTEM_CPU_SET_INFORMATION;
```

- Can query cache to keep worker threads from jumping onto another cpu with a separate L2 cache
  - `GetLogicalProcessorInformationEx()`
  - `RelationCache - enum _LOGICAL_PROCESSOR_RELATIONSHIP`

INCORPORATING HYBRID ARCHITECTURES
MODIFYING THE TASK SYSTEM

- Separate worker threads created for the e-cores and p-cores (again based on number available)
- Used CPUSets to tell the OS what cores to run them on
- Modified our task system
  - Worker threads have a type (either High Performance or High Efficiency)
  - Tasks tagged when spawned with desired type (default is High Performance)
  - Workers can only steal tasks from the same type

INCORPORATING HYBRID ARCHITECTURES
FINDINGS

- E-cores are surprisingly fast
  - Around 60-70% of speed running battle model
  - Probably due to random-access nature of model code, and memory bus is shared
- Could run model on the e-cores without any noticeable affect on frame rate, and in some cases gave a boost (as more p-cores available)

- Once we got the battle model running on the e-cores we introduced other systems:
    - Character cloth
    - Async Rag-doll
    - Vortex effects

PROBLEMS ENCOUNTERED

- Stuttering issue - some users with hybrid cores found the game could stutter. Occasional frames would take much longer than expected.
- Took a lot of investigation - most profiling tools don’t show you what core a thread is running on. GPUView and Windows Performance Analyser (WPA) can give you this information
- Very difficult to observe over a remote connection!
- Results different between Windows 10 and Windows 11
- We noticed in the profiler that the bottleneck was threads taking a long time (>30ms) to wake up and service new tasks

PROBLEMS ENCOUNTERED

PROBLEMS ENCOUNTERED

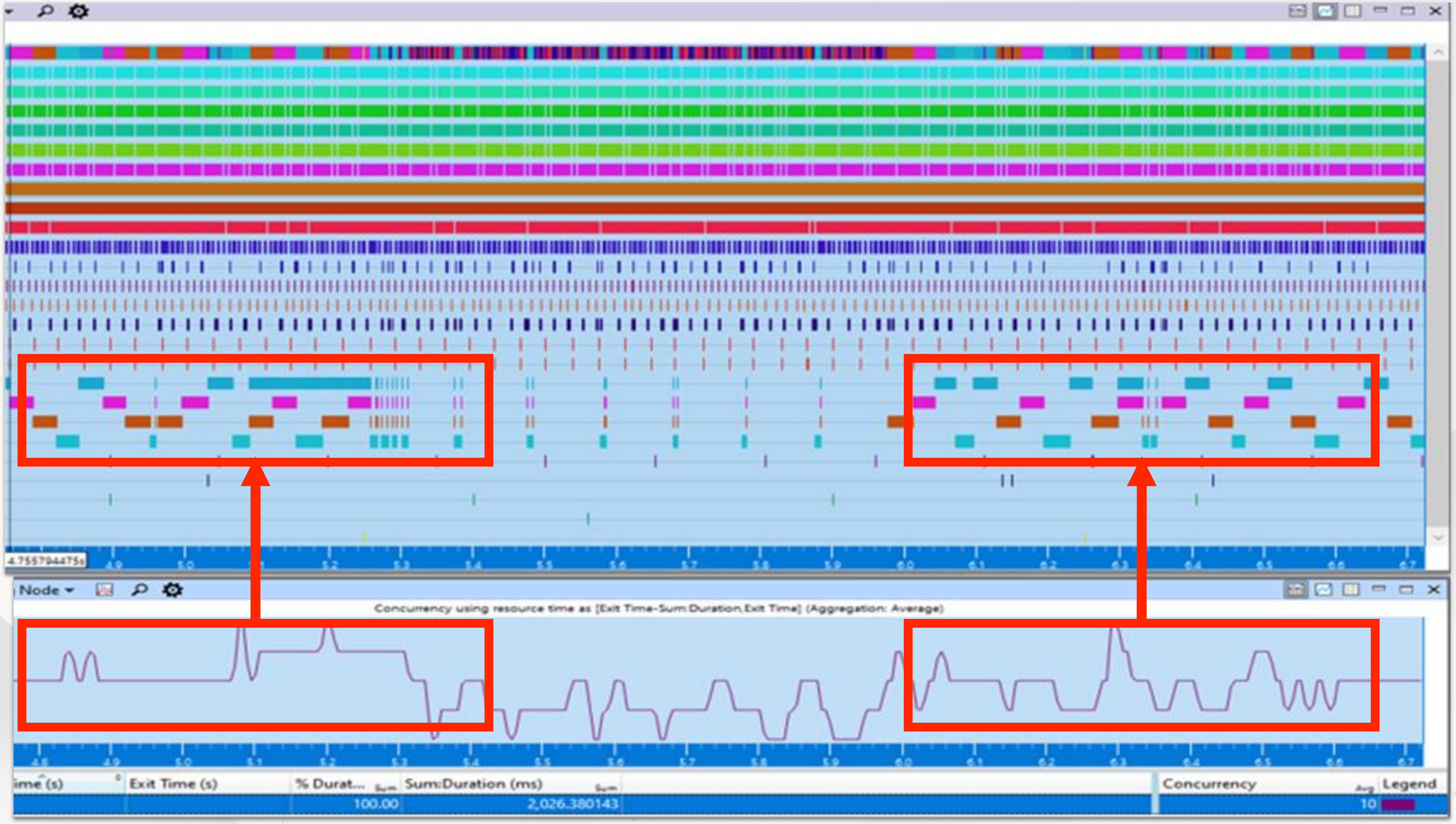
- Observed that stuttering occurred on worker thread wake-up (see screen shot)
- Initially mitigated the stuttering issue by disabling sleeping in our task system.
- This led to CPU temperature complaints due to thread idling
- PAUSE intrinsic - used in idle loop
- Still hadn’t got a handle on the real cause

FINDINGS
PARKING

- OS wants to conserve power and 'park' cores
- Parking is essentially powering down the core
- Discovered if you disable parking in Windows the stuttering goes away - needed to understand why!
- Thanks to Steve and Leigh at Intel for this discovery and the following profile capture!

ETL trace viewed in WPA (from Intel)

PROBLEMS ENCOUNTERED



High Performance Workers

High Efficiency Workers

Number of cores parked

FINDINGS
PARKING

- Stuttering directly linked to number of parked cores
- When we understood the ‘parking problem’ things became clear
- OS is parking e-cores as they are often idle for many milliseconds (between model ticks)
- This highlighted an issue with our task system
  - We have internal affinity in our task system to stop multiple large tasks ending up on the same thread
  - The Battle Model was forced to run on a specific worker (via our task affinity), but if multiple e-cores are parked, this specific worker might be queued along with multiple other workers on a single core

FINDINGS

PARKING

Battle Model Task Affinity - Worker 0

Worker 0

```
worker_thread_func()

 task = get_task()
 if(task)
    execute(task)
 else if(total_tasks==0)
    sleep()
```

Worker 1

```
worker_thread_func()

 task = get_task()
 if(task)
    execute(task)
 else if(total_tasks==0)
    sleep()
```

Worker n..

```
worker_thread_func()

 task = get_task()
 if(task)
    execute(task)
 else if(total_tasks==0)
    sleep()
```

FINDINGS

PARKING

Battle Model Task Affinity - Worker 0

Worker 0

Worker 1

Worker n..

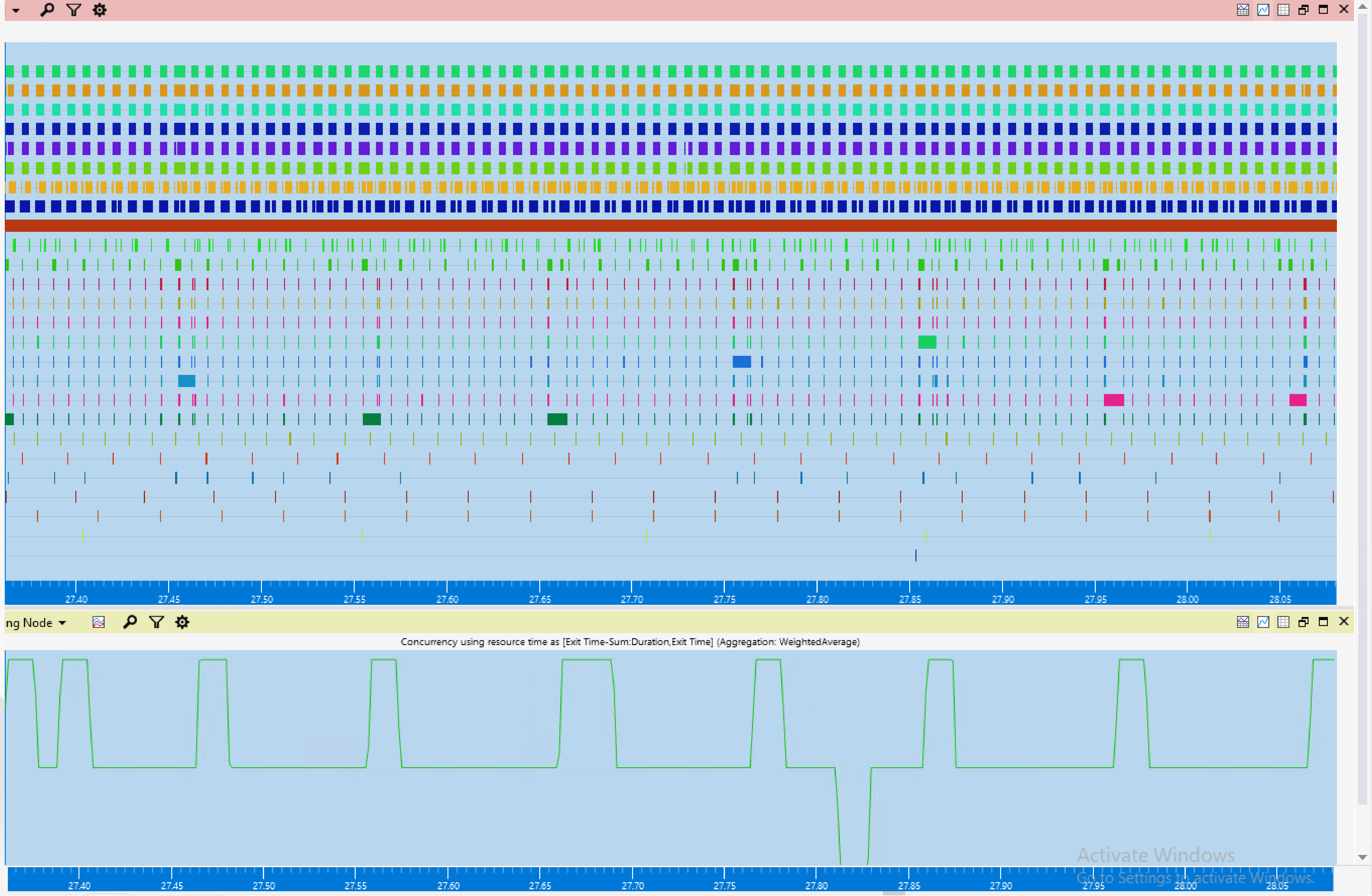
```
worker_thread_func()

 task = get_task()
 if(task)
    execute(task)
 else if(total_tasks==0)
    sleep()
```

Many e-cores are parked, so the worker threads are queued on a single core. The first worker thread can't pull a task but there are still tasks in the queue so it doesn't sleep and spins until timeout

ETL trace viewed in WPA (from Intel)

STUTTER RESOLVED!



} High Performance Workers

} High Efficiency Workers

} Number of cores parked

FINDINGS
CONCLUSIONS AND SUGGESTIONS

- Use CPUSets to establish CPU topology
- Use CPUSets to bind your threads to e-cores/p-cores
  - Use profiling tools that allow you to track which core a thread is running on (GPUView, WPA) to identify bottlenecks
- Set an IdealProcessor for your threads, helps to keep them on the same core when waking up
- Ensure your task system supports stealing, so if a worker thread does get stalled by being on a parked CPU it can be executed by other workers

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

intel®

intel ARC™