Scalable Performance Analysis on Heterogeneous Architectures with HPCToolkit

Milind Chabbi, Karthik Murthy,
Mike Fagan, and John Mellor-Crummey
Rice University

CScADS 2012
Snowbird, Utha
June 26, 2012

HPCToolkit

- Performance measurement using statistical sampling of timers and performance counters
- Attribution to hierarchical calling context
- Works on multilingual, fully-optimized, statically or dynamically linked applications (no source modification)
  - Pthread, OMP, MPI, and any combination
- Low overhead (under 5%) for both profiling and tracing
- Scales to large parallel systems
- Analysis of execution costs, inefficiencies, and scaling characteristics

Supporting Heterogeneity in HPCToolkit

- **Heterogeneous** does't mean just GPU kernel
- Most work on the performance analysis of heterogeneous architectures deals with
  - Identifying GPU-kernel-level issues, and improving via: kernel fusion, unrolling, memory access reordering, etc.
- They ignore other parts of heterogeneous systems viz.
  - Nodes with several GPUs and CPUs, and CPUs with several threads
  - GPUs shared by multiple ranks, and concurrent kernel executions
  - Inter-node, and intra-node communication

Should Measure, Analyze and Present

Performance of

- A standalone GPU kernel
  - Timing, and hardware counter values
- Concurrently executing GPU kernels on multiple graphics cards
  - Challenges: concurrent streams, multiple threads, multiple contexts, GPU sharing between threads and processes
- Data communication between CPUs and GPUs
- Multi-threaded processes
- Multiple MPI processes

And It Should Scale

- Should be able to gather data from thousands of nodes
  - Each with several CPUs, Cores, and multiple GPU cards
- Should not distort original execution overlap
- Should have low runtime overhead
- Should produce manageable profile and trace files

Focus on Resource (under) Utilization

- Heterogeneous systems have multiple resources each with disparate capabilities
- Classical "hot-spot" analysis is insufficient
  - Focuses on "most consumed" resources
  - Provides only symptoms of problems
  - Does not indicate causes of problems
- Key to achieving peek performance on heterogeneous systems is to keep all compute resources working simultaneously
  - Overlap computations on multiple resources

Work Balance Between CPU and GPU

- Offloading entire computation to GPUs wastes CPU compute power
- Offloading entire computation to CPUs wastes GPU compute power

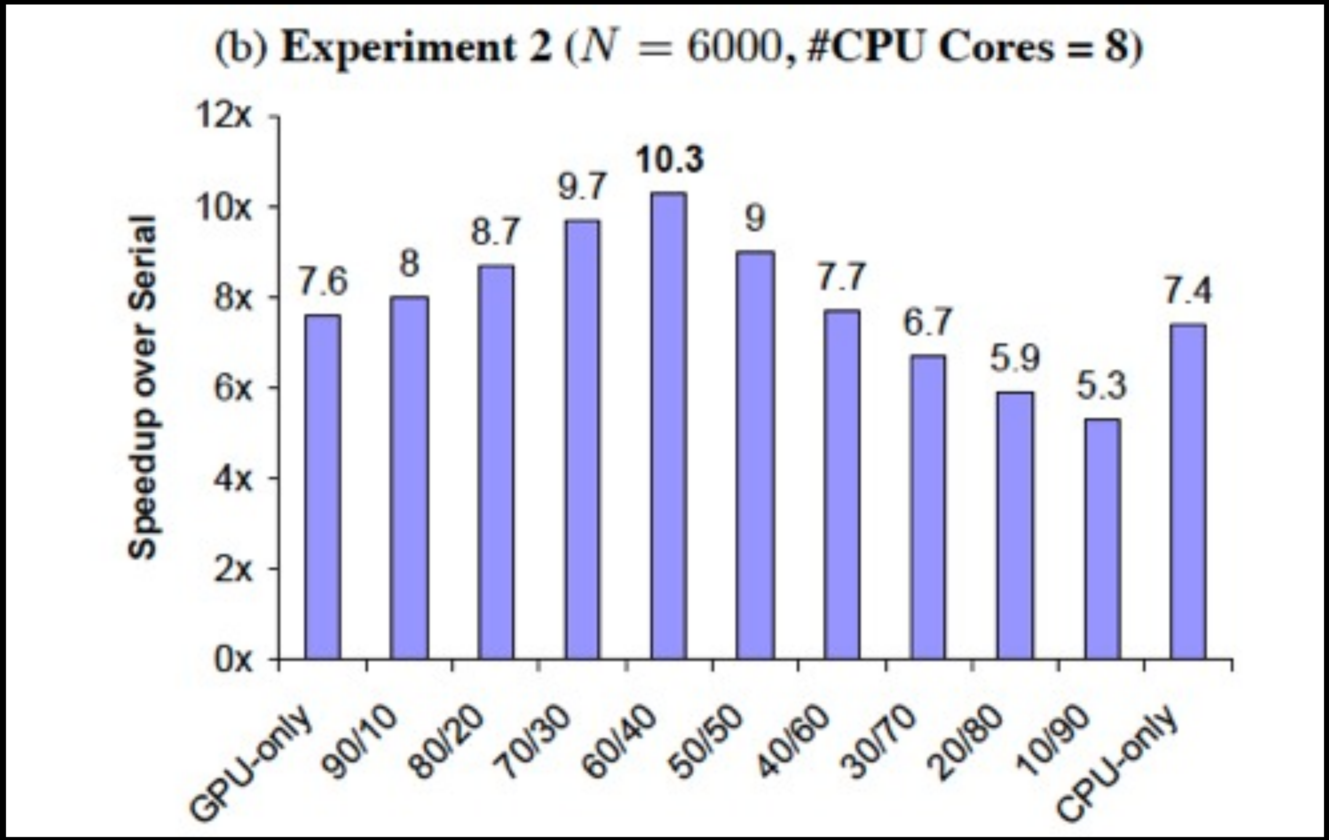


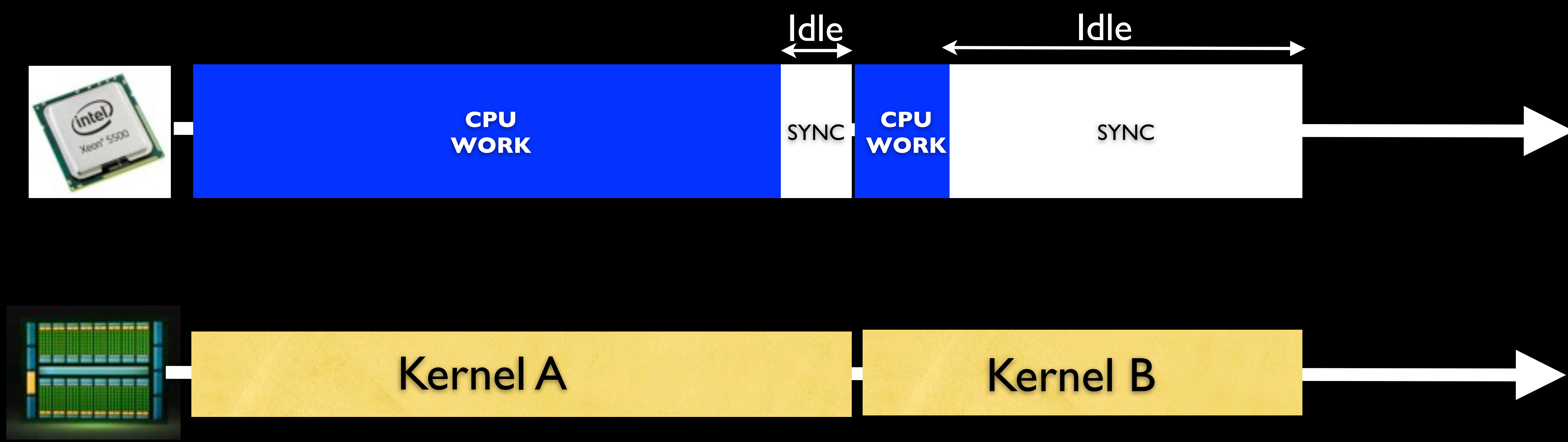
Figure credit: Qilin Exploiting Parallelism on Heterogeneous Multiprocessors

Matrix multiplication on Nvidia 8800 GTX (575 Mhz) and Intel Core2 Quad (2.4Ghz)

Root Cause Analysis with Blame Shifting

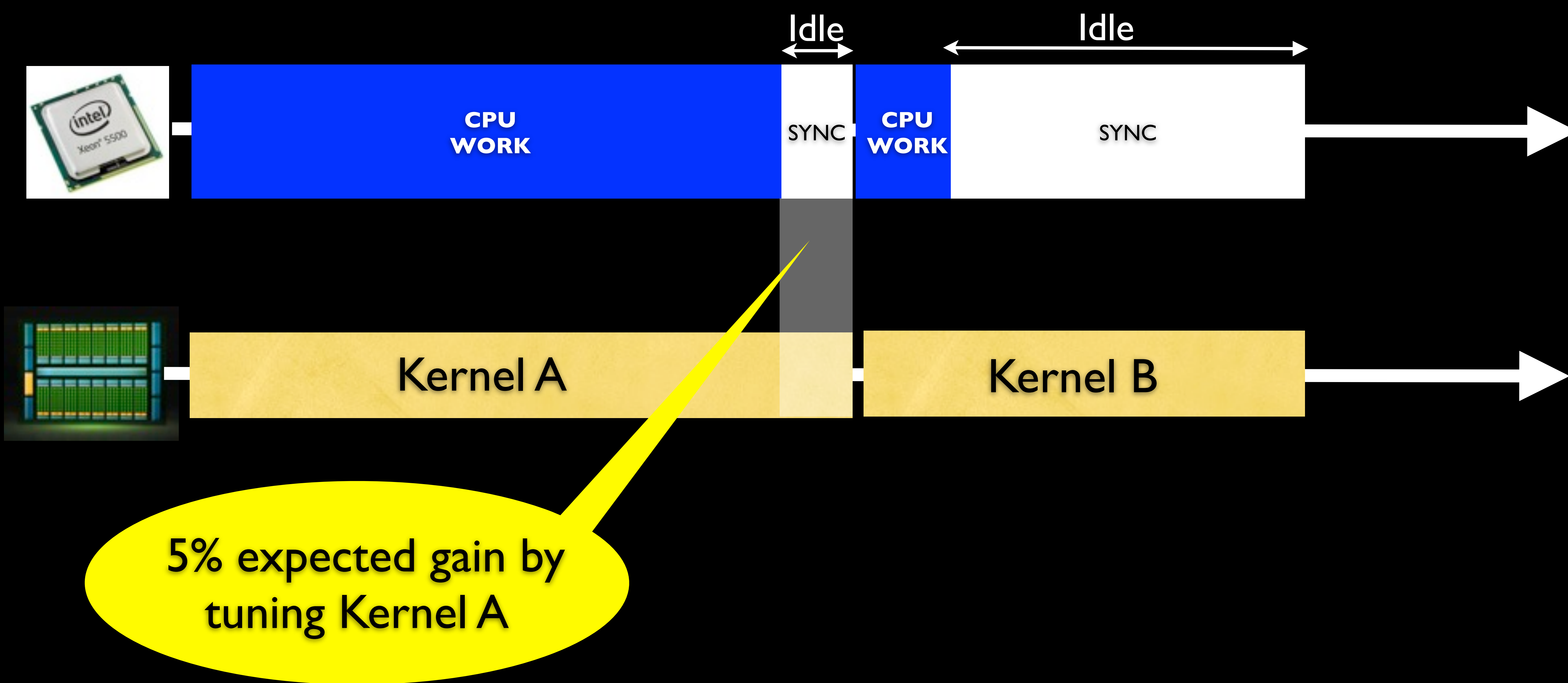
- If GPU is idle, code executing on CPU is responsible for not offloading (enough) work to GPU
  - ✦ Attribute blame to CPU code executing while GPU is idle
- If CPU is idle waiting for GPU kernel(s) to finish, executing GPU kernel(s) are responsible for CPU idleness
  - ✦ Attribute proportional blame to each such kernels
- Credit codes that are well overlapped

Performance Expectations for Heterogeneous Systems with Blame Shifting



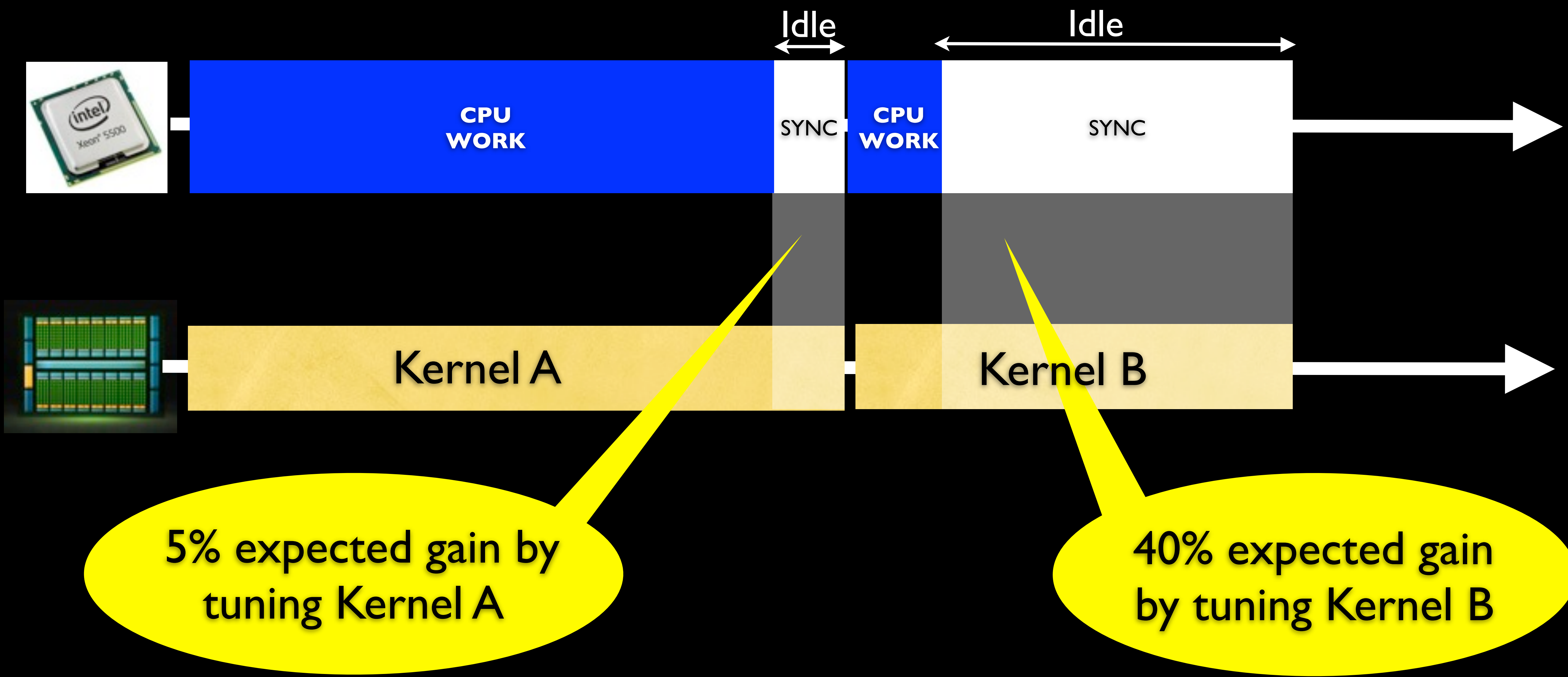
Top GPU-kernel may not be the best candidate for tuning

Performance Expectations for Heterogeneous Systems with Blame Shifting



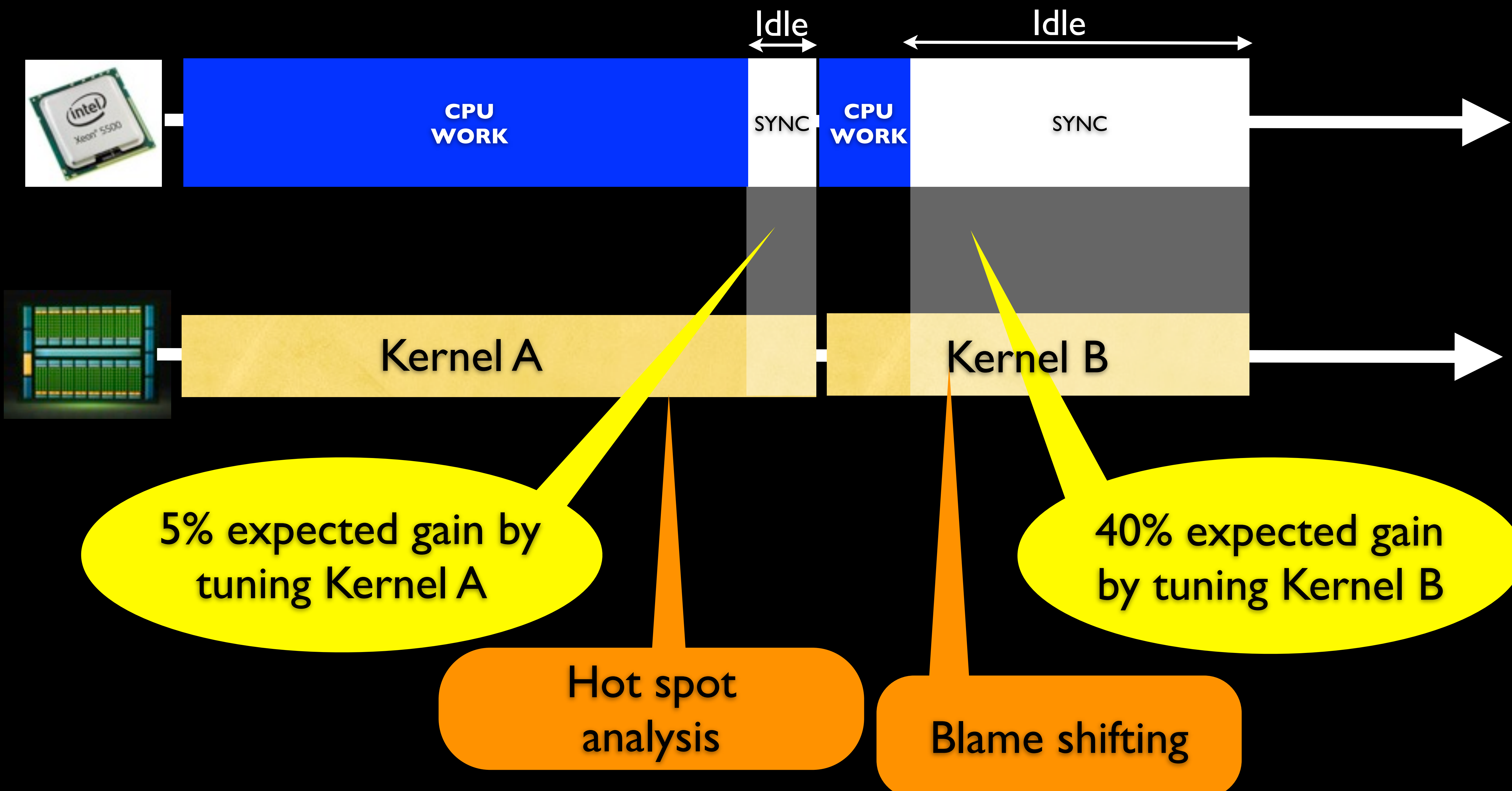
Top GPU-kernel may not be the best candidate for tuning

Performance Expectations for Heterogeneous Systems with Blame Shifting



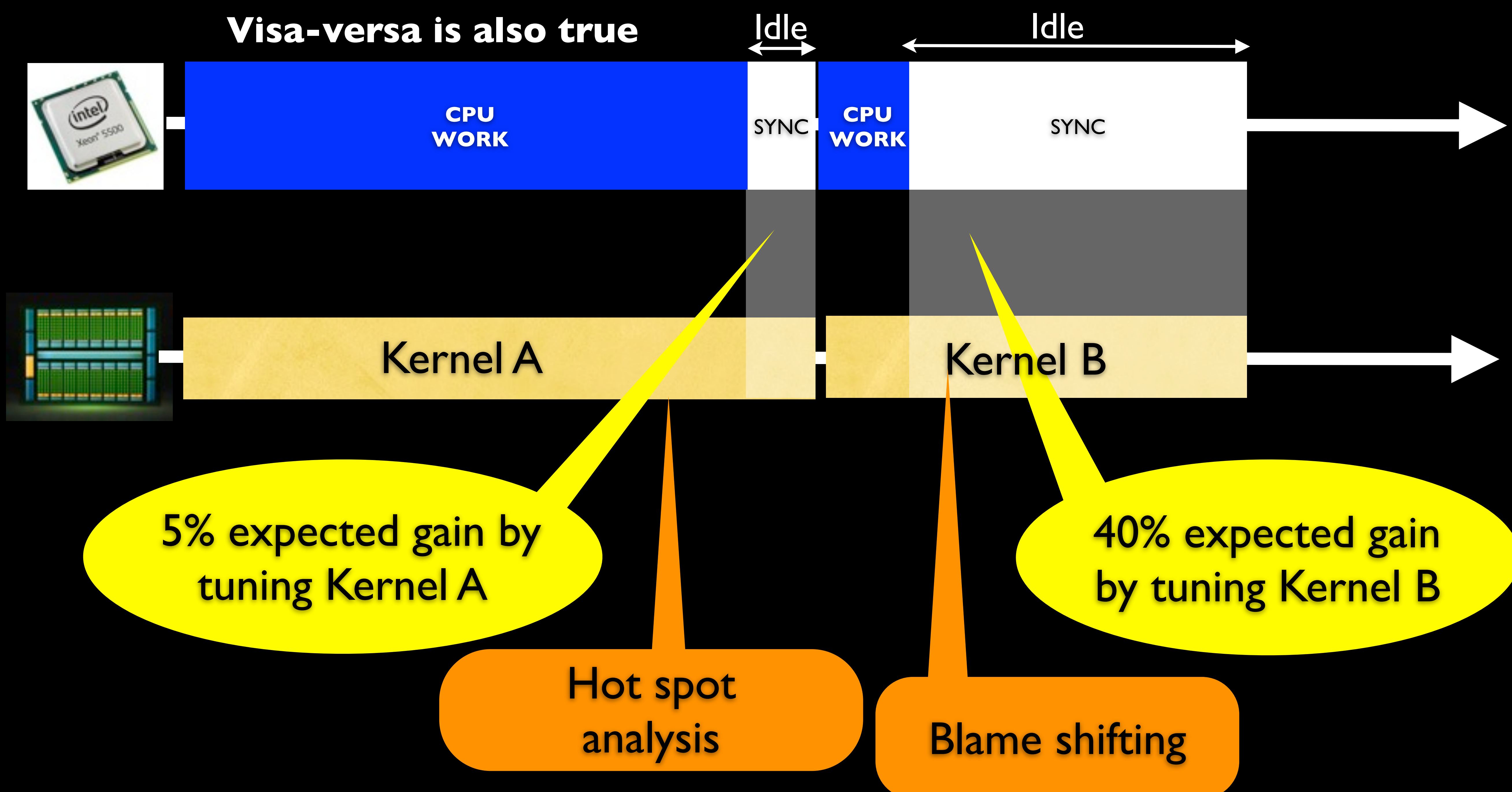
Top GPU-kernel may not be the best candidate for tuning

Performance Expectations for Heterogeneous Systems with Blame Shifting



Top GPU-kernel may not be the best candidate for tuning

Performance Expectations for Heterogeneous Systems with Blame Shifting



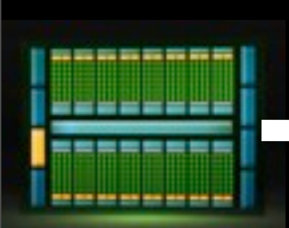
Top GPU-kernel may not be the best candidate for tuning

Advantages of Blame Shifting on Heterogeneous Systems

- Pinpoints codes (both GPU kernels and CPU contexts) that benefit most from tuning
  - Improves developer productivity
  - Full calling context to distinguish same kernel, different callpath
- Provides an expectation for the upper bound of performance gain when tuning
- Sampling-based approach keeps overhead low and provides scalability
- Extends naturally to any shared resource
  - GPU, communication network, I/O network

Proxy Sampling of GPU Activities

CPU
thread

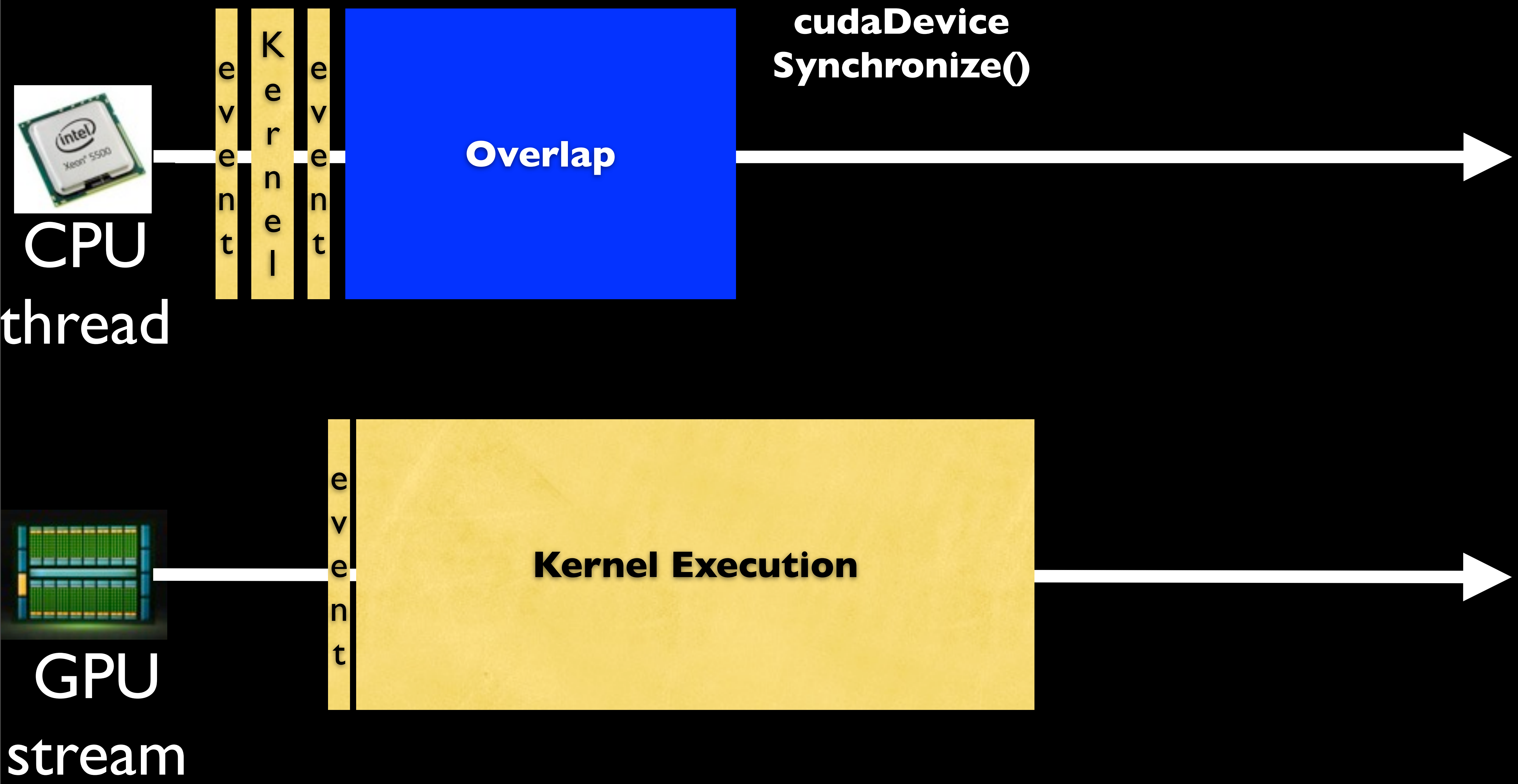


GPU
stream

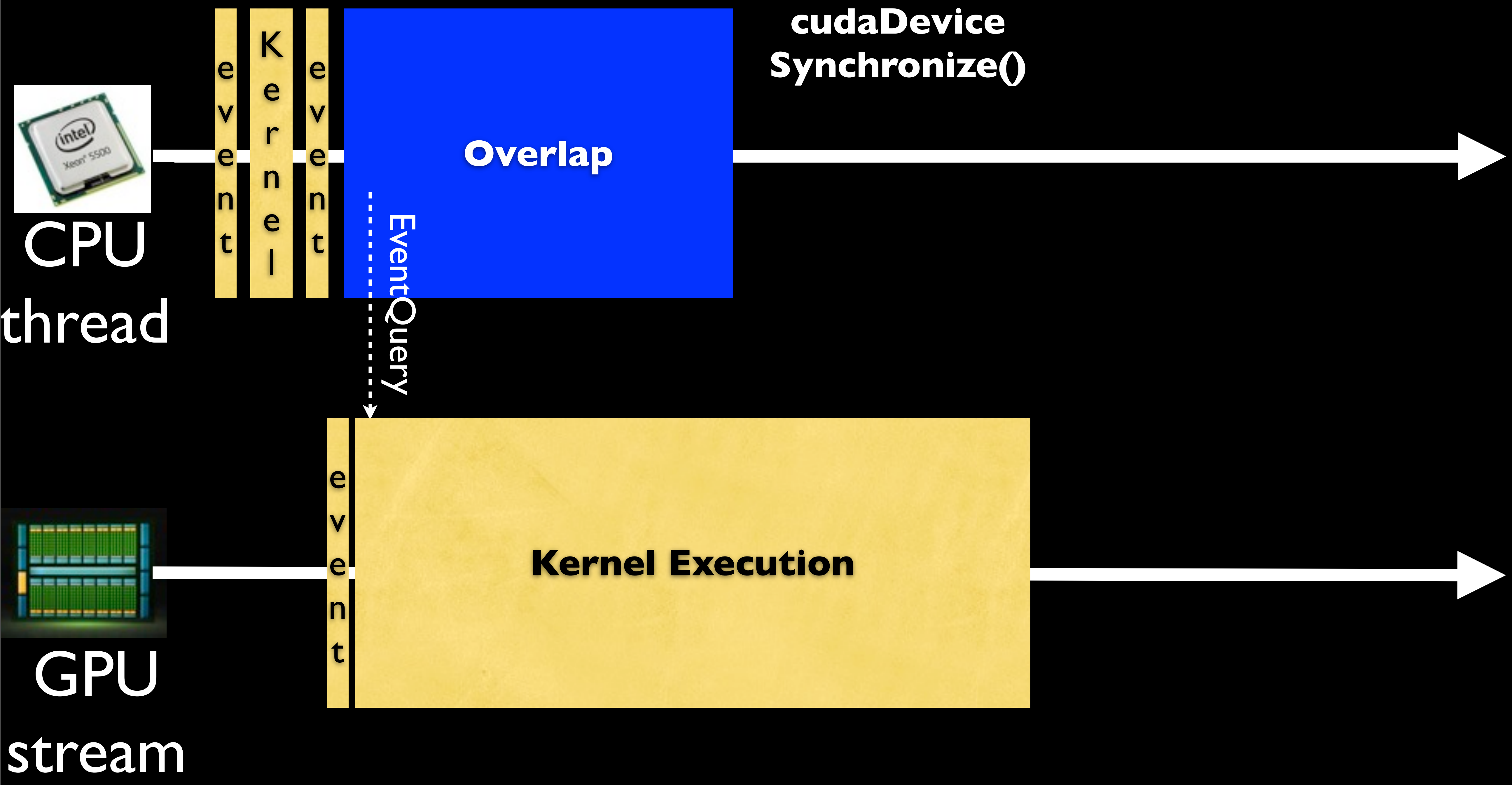
Proxy Sampling of GPU Activities

Proxy Sampling of GPU Activities

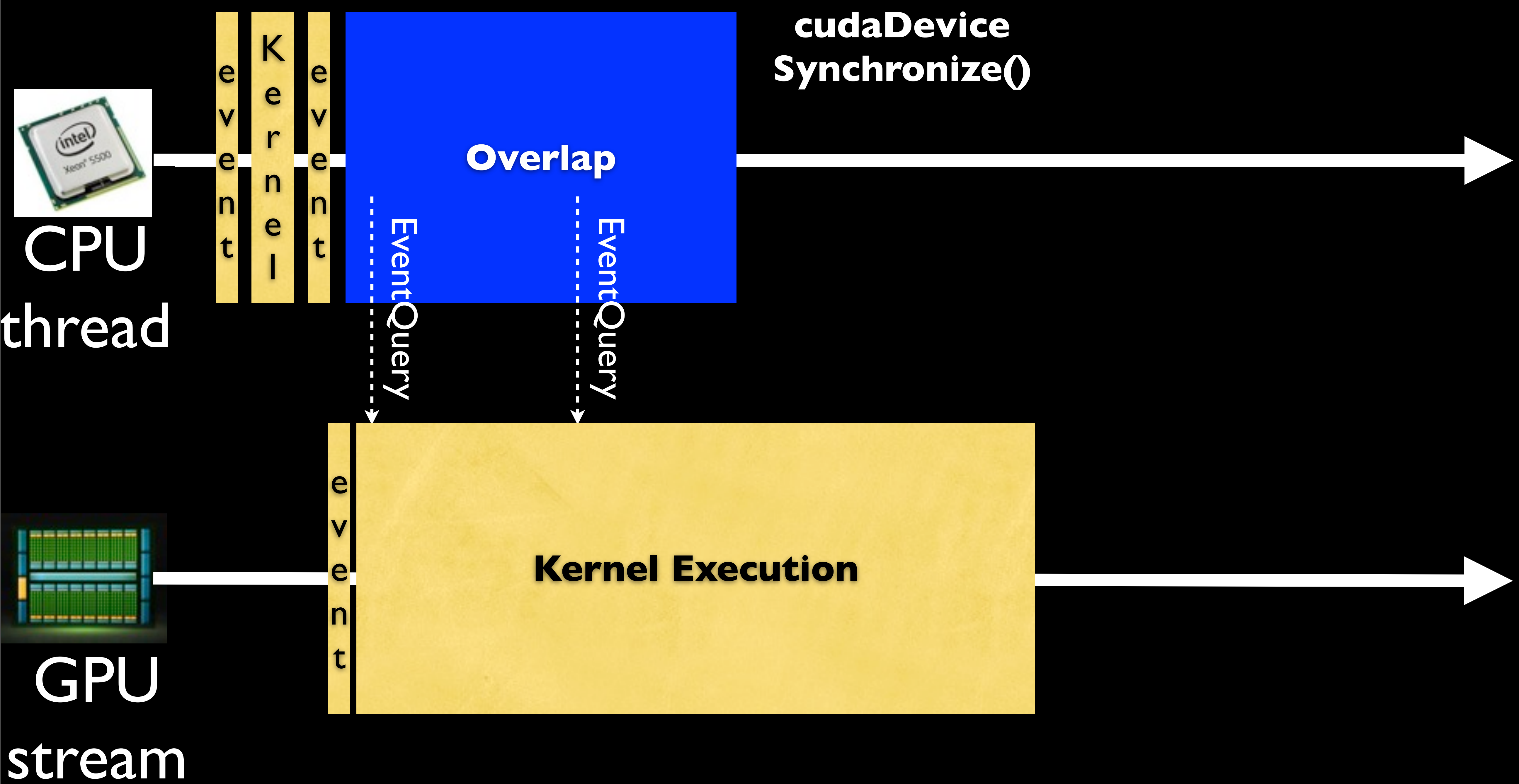
Proxy Sampling of GPU Activities



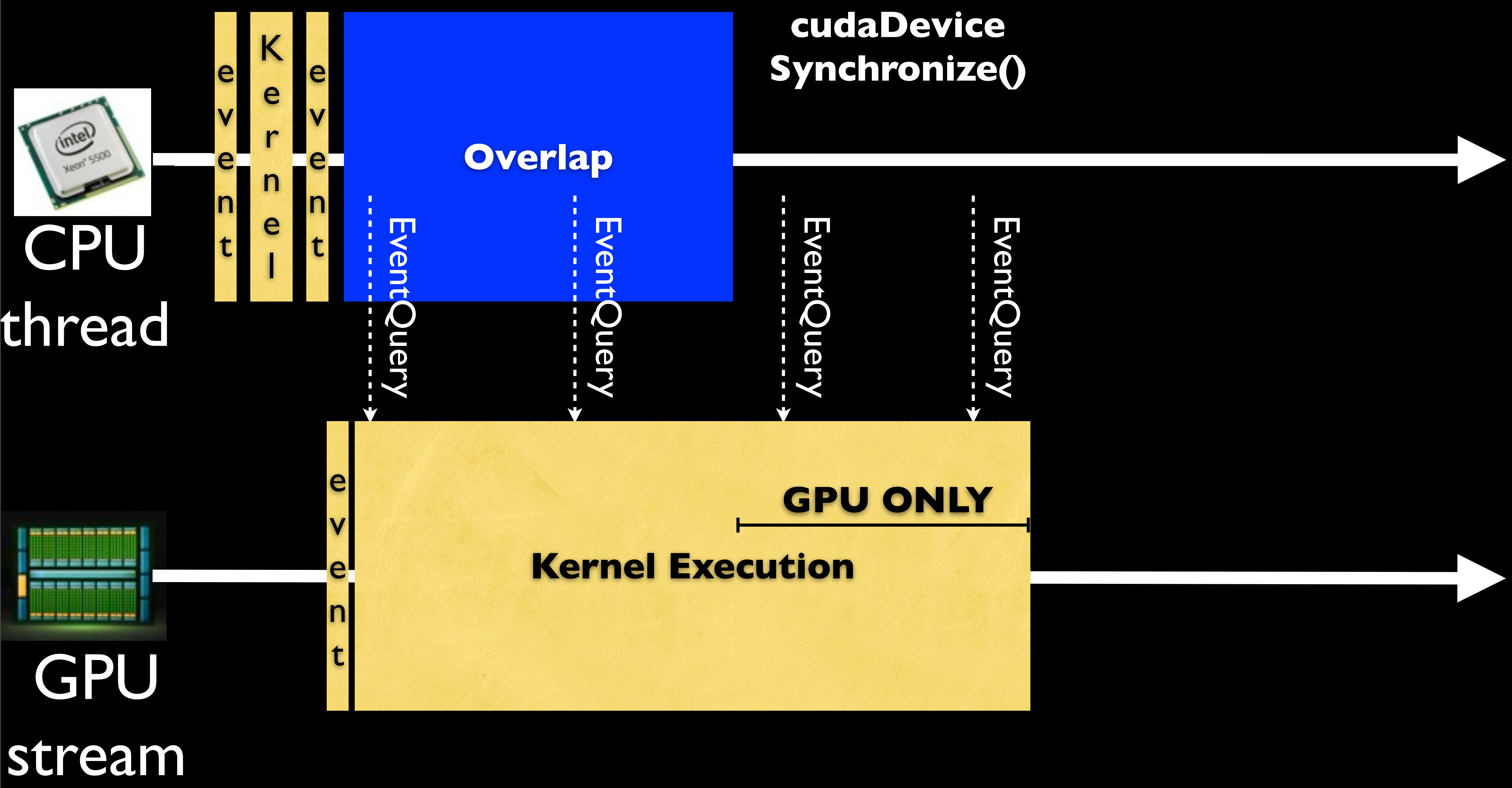
Proxy Sampling of GPU Activities



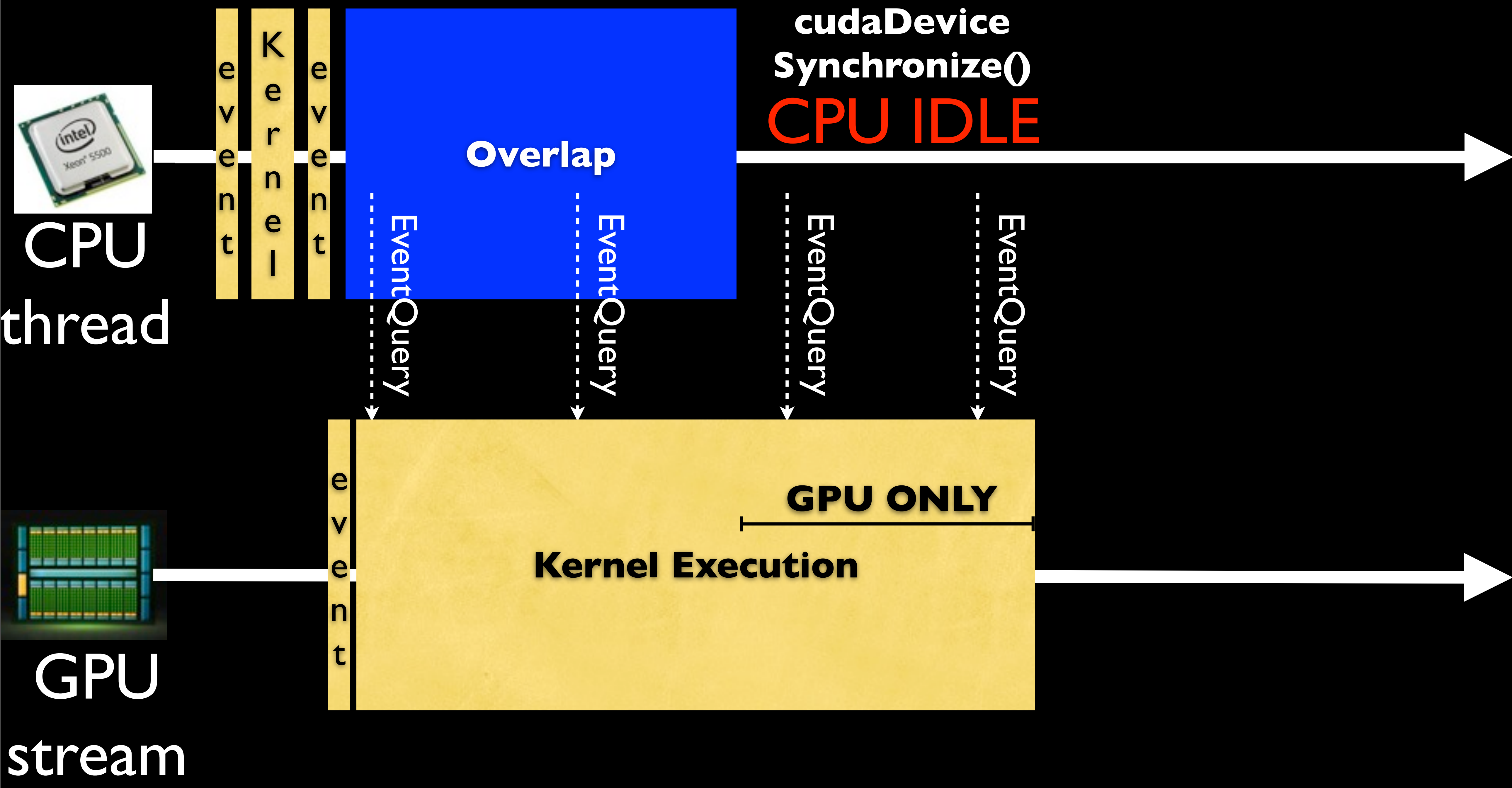
Proxy Sampling of GPU Activities



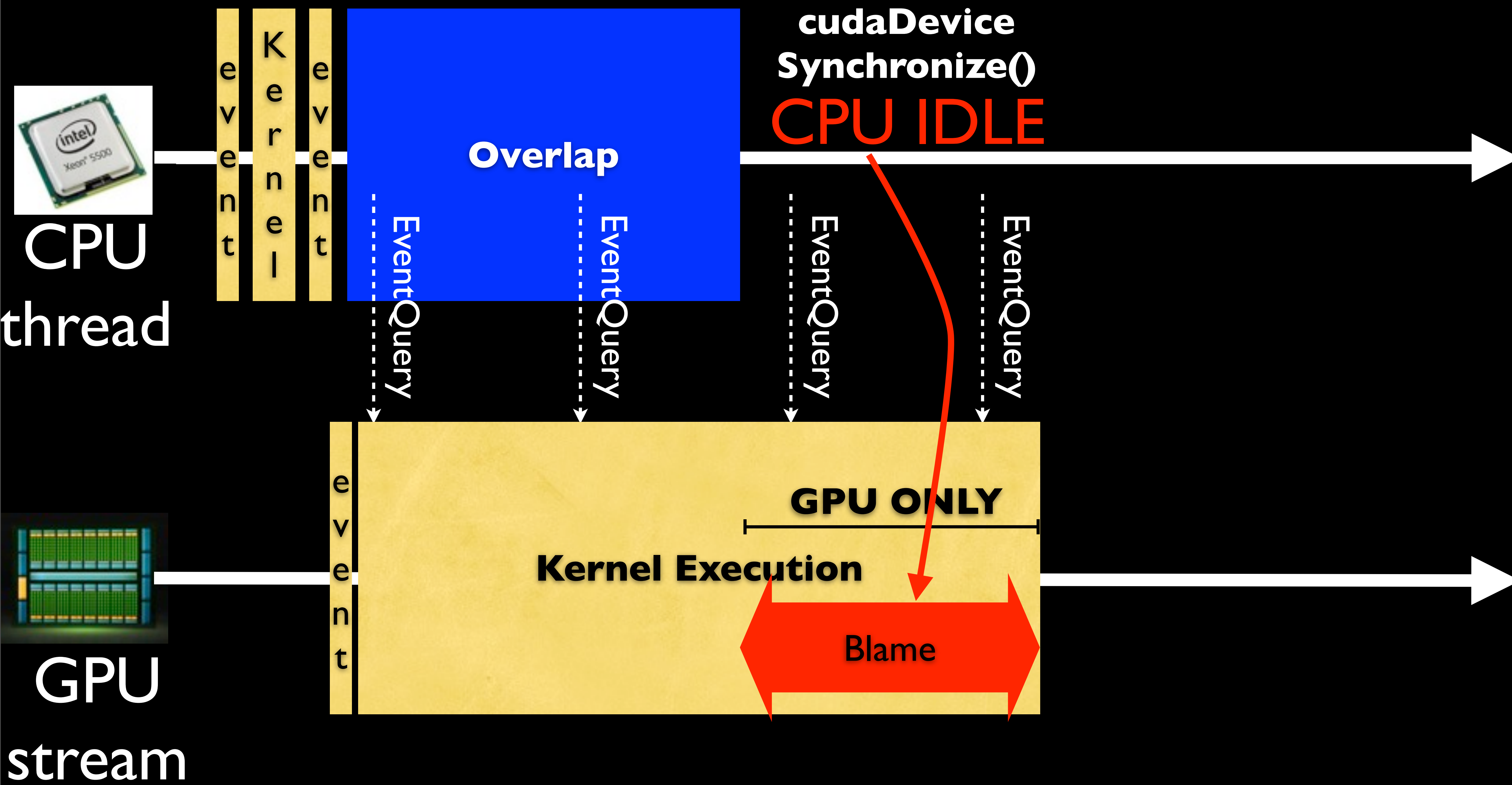
Proxy Sampling of GPU Activities



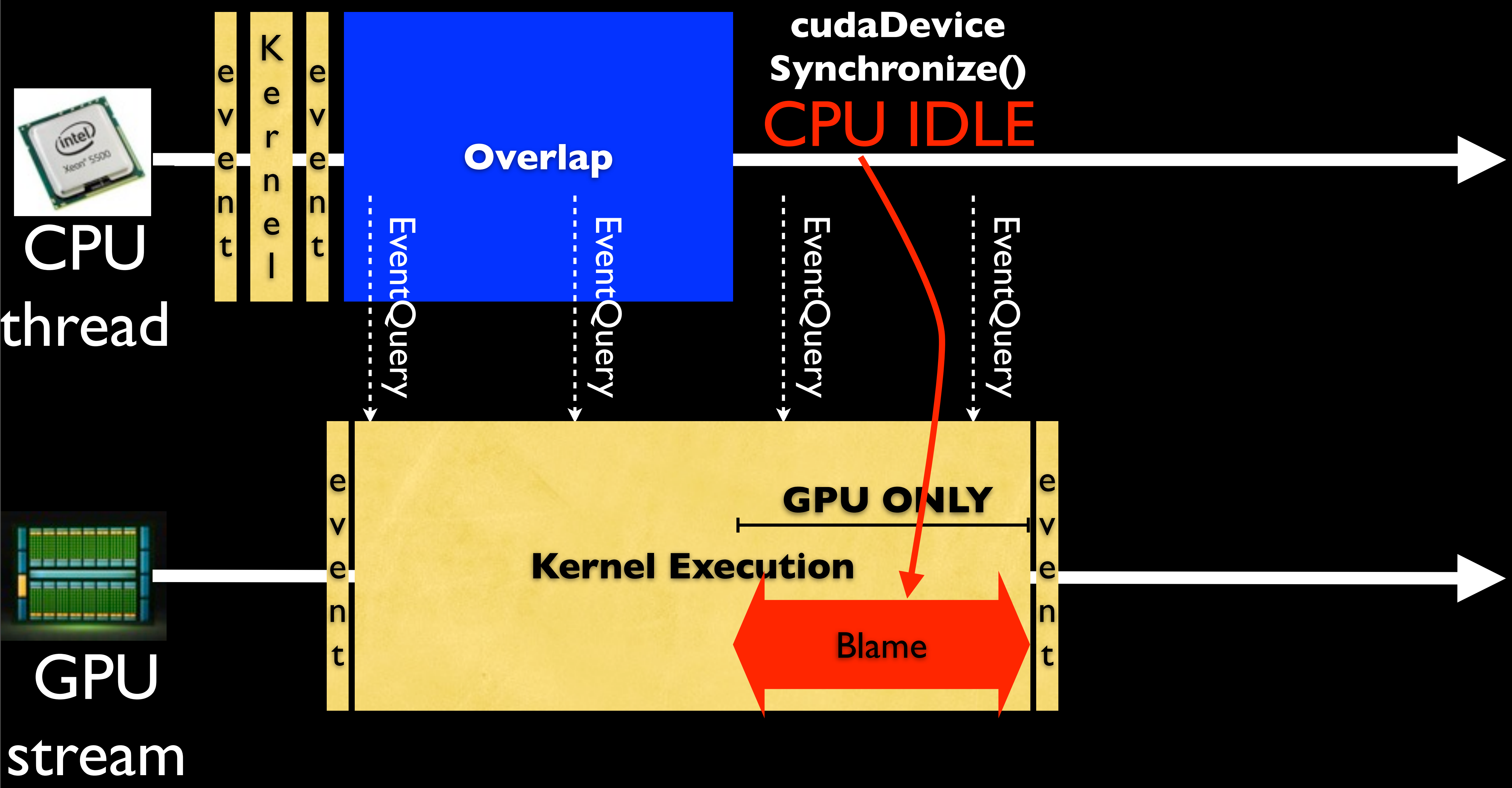
Proxy Sampling of GPU Activities



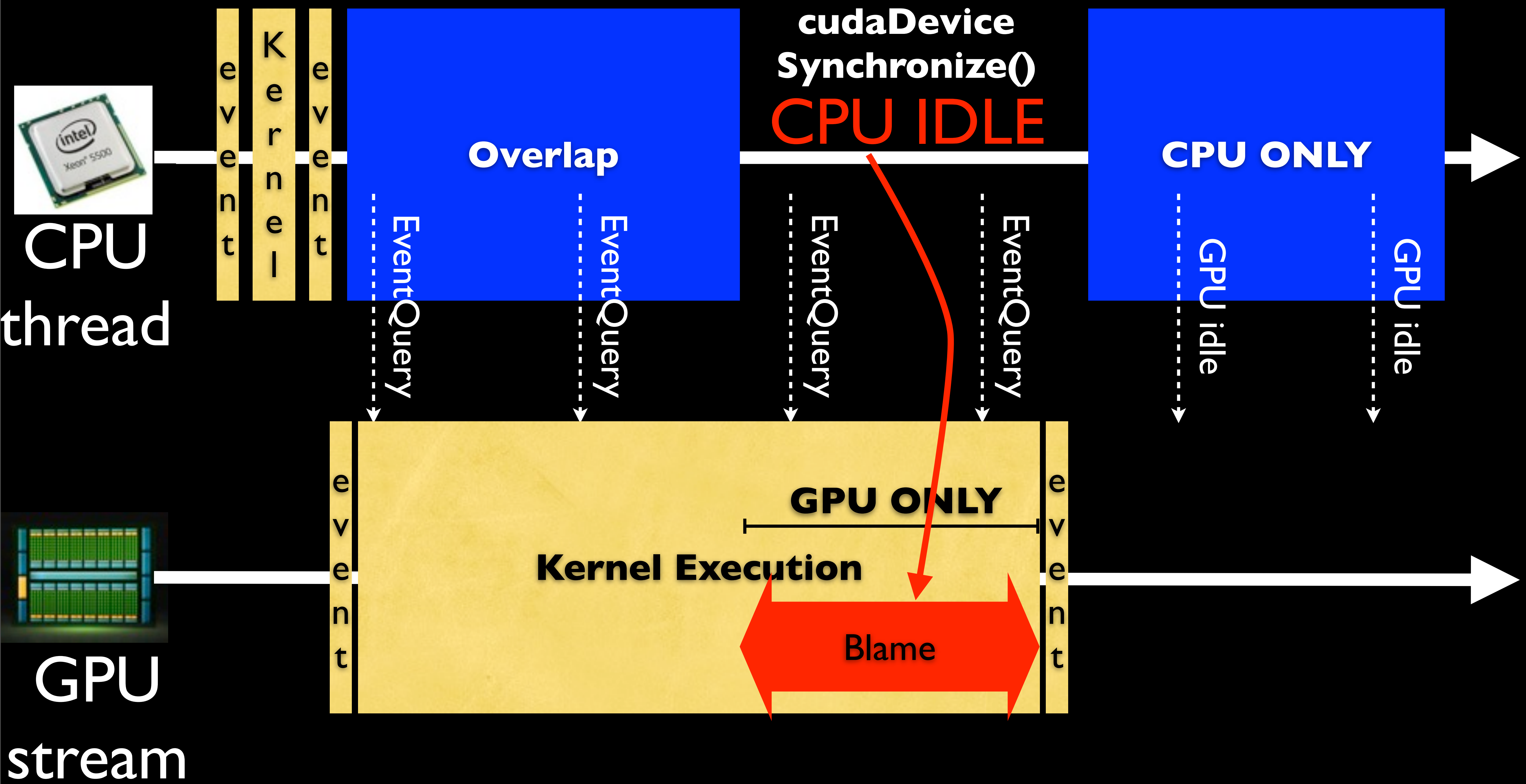
Proxy Sampling of GPU Activities



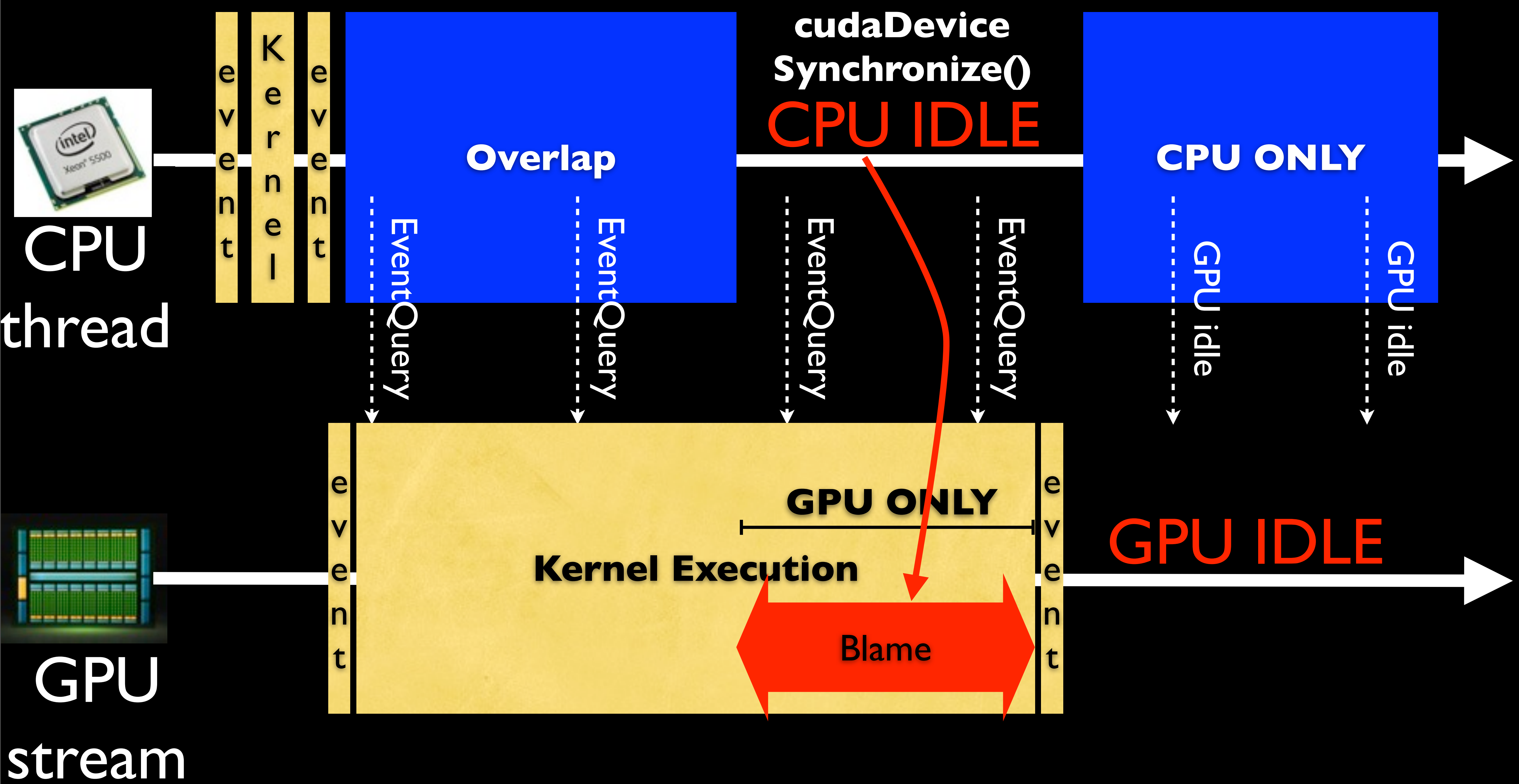
Proxy Sampling of GPU Activities



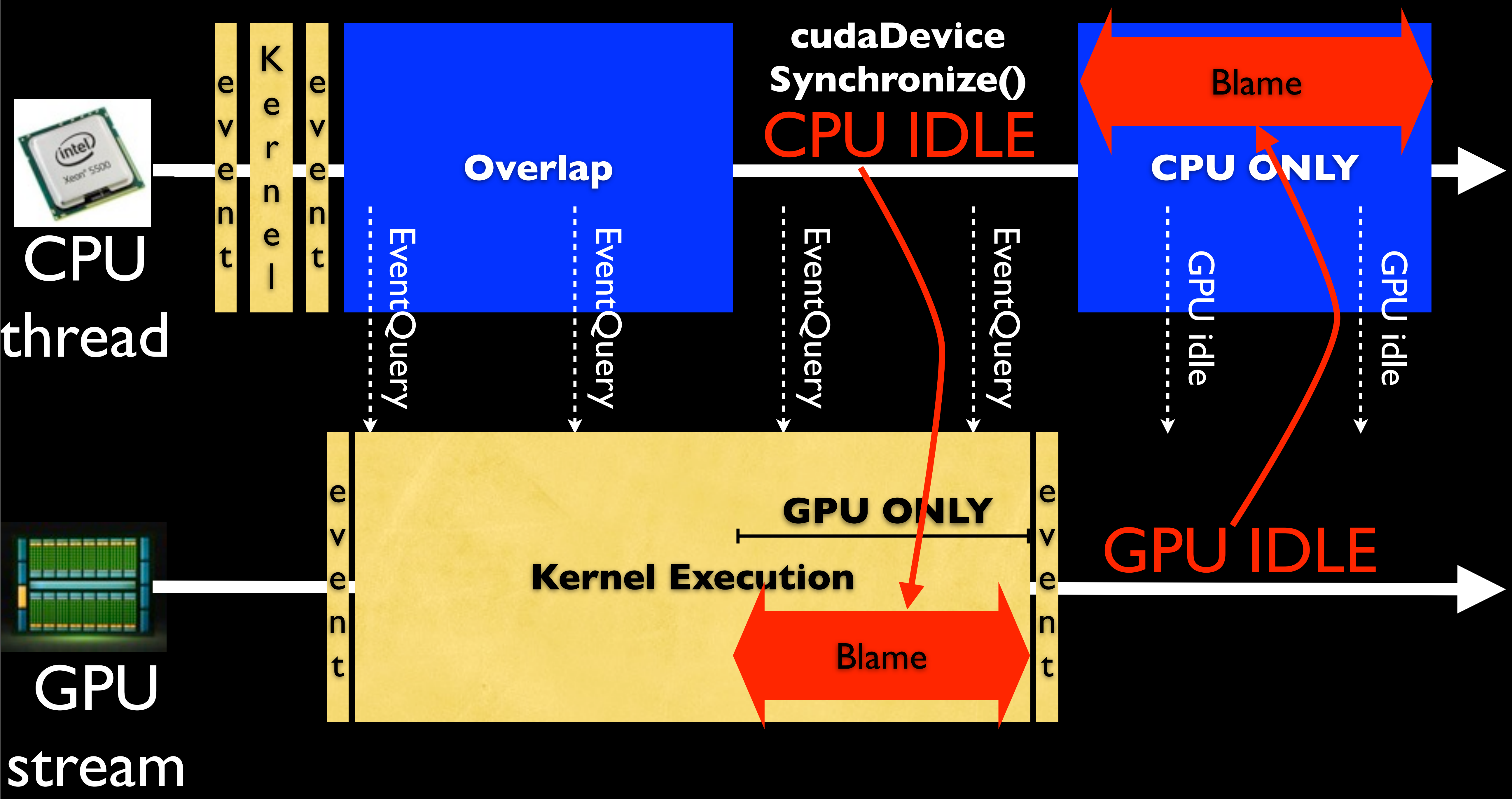
Proxy Sampling of GPU Activities



Proxy Sampling of GPU Activities



Proxy Sampling of GPU Activities



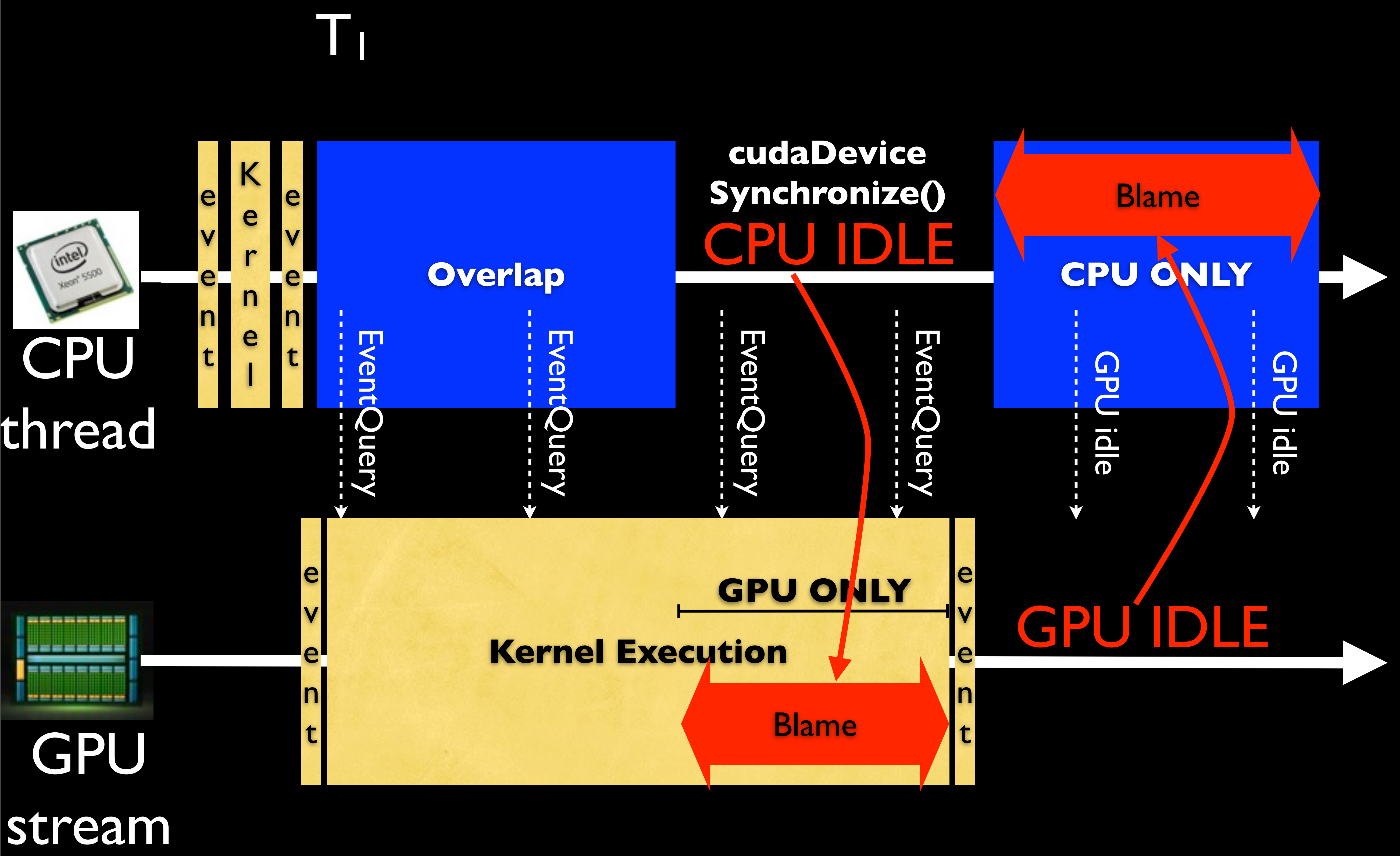
Implementation Challenges

- No sampling support from GPUs
  - ✦ Would have liked timer/counter-based signals from GPUs
- CUPTI has several limitations (some fixed in 5.0RC)
  - ✦ Kernel serialization when using CUPTI
  - ✦ Serialization of CPU threads simultaneously using CUPTI
  - ✦ Activity API is more tracing style, not suitable for profiling
- CUDA limitations (supposed to be fixed in Kepler 2)
  - ✦ Kernel serialization when using events for querying/timing
- Can't poke GPU with cudaEventQuery() from a signal handler when thread is inside a CUDA API call

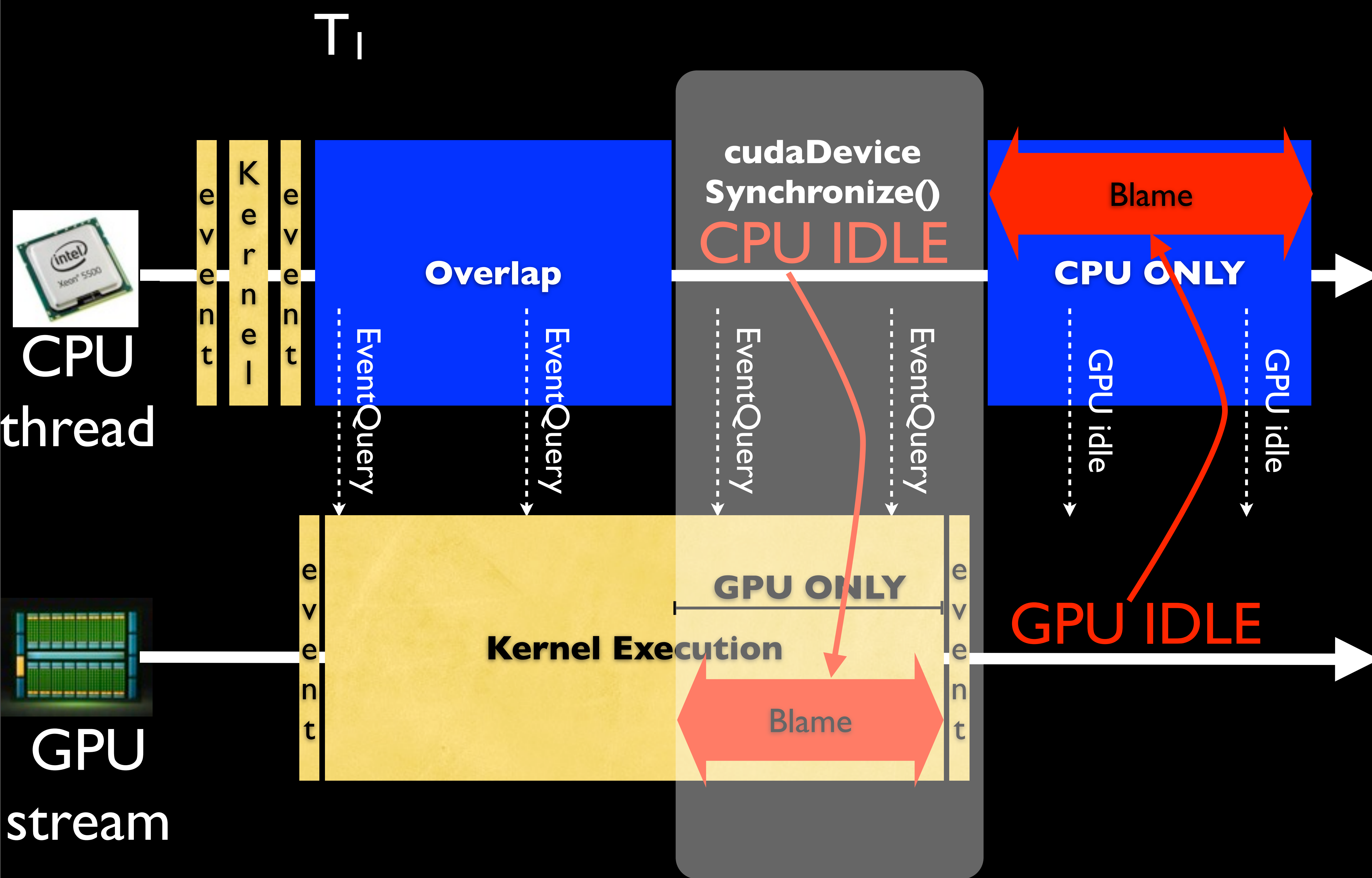
Workarounds

- CUDA Function wrapping to inject events
  - ✦ Eliminates CPU threads serialization
  - ✦ Waiting for Kepler-2 to fix kernel serialization when using events
- Disable calling cudaEventQuery() from signal handler when CPU is inside CUDA API
  - ✦ Deferred blaming of kernels

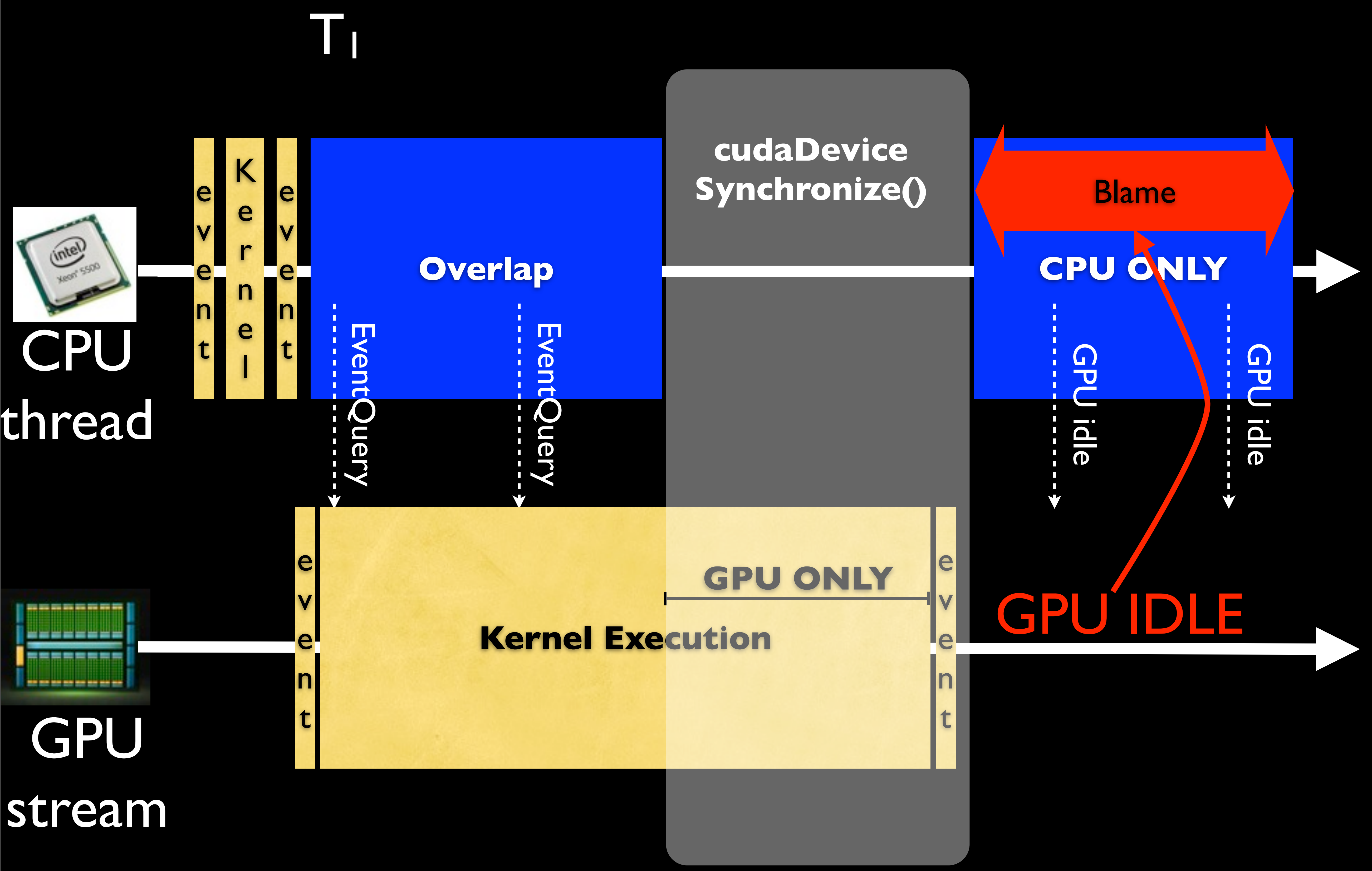
Workarounds: Deferred Blaming



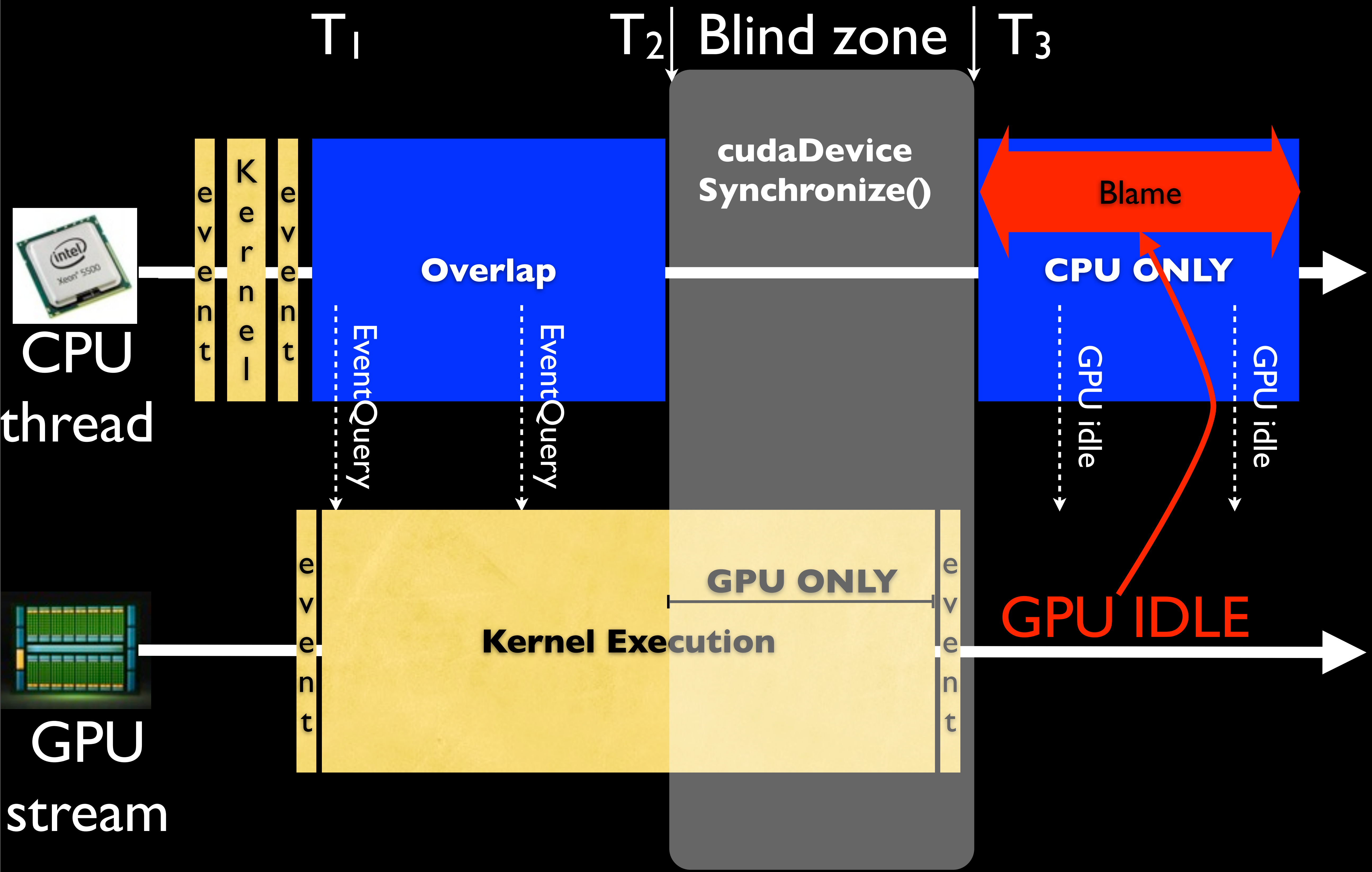
Workarounds: Deferred Blaming



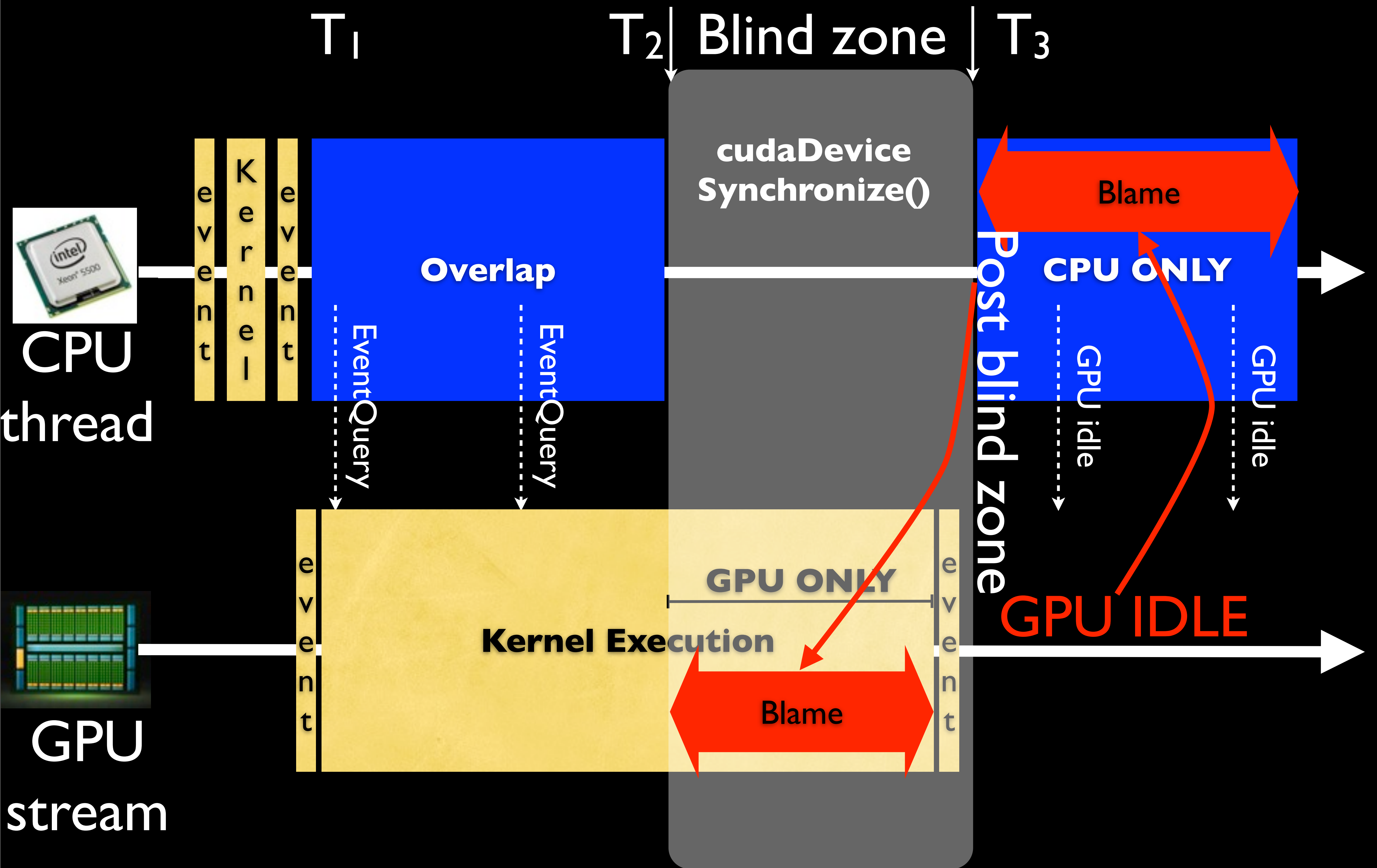
Workarounds: Deferred Blaming



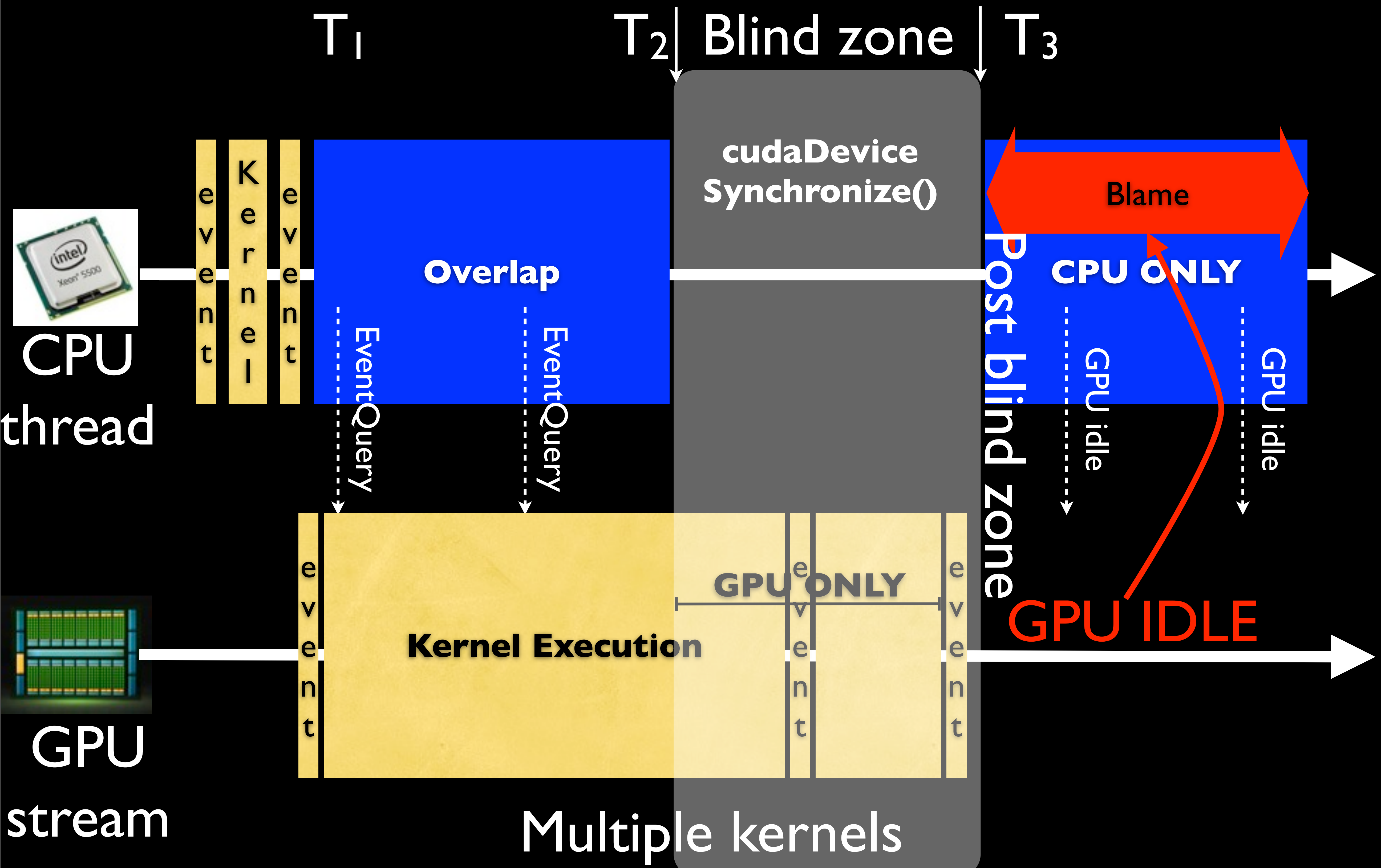
Workarounds: Deferred Blaming



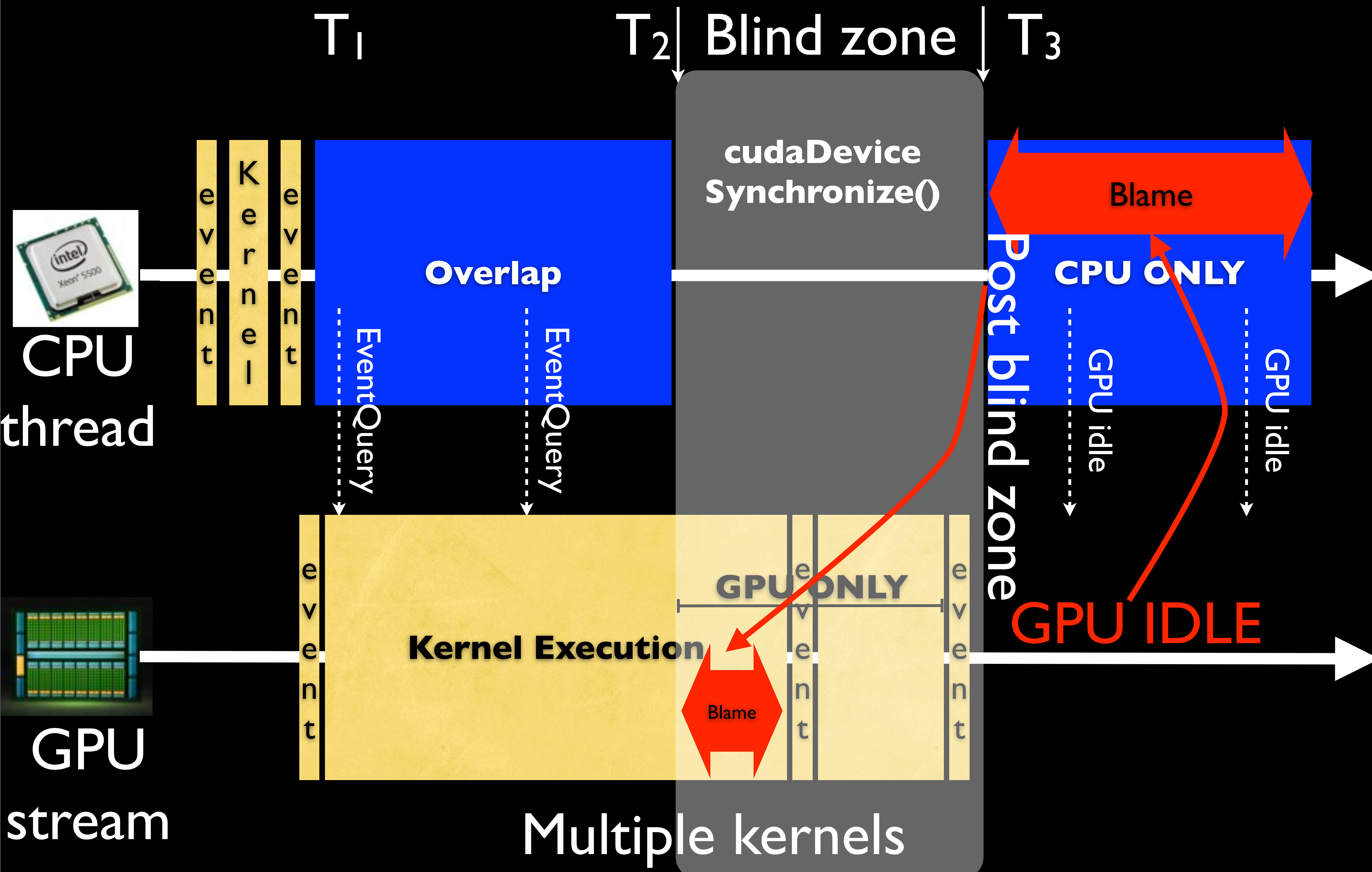
Workarounds: Deferred Blaming



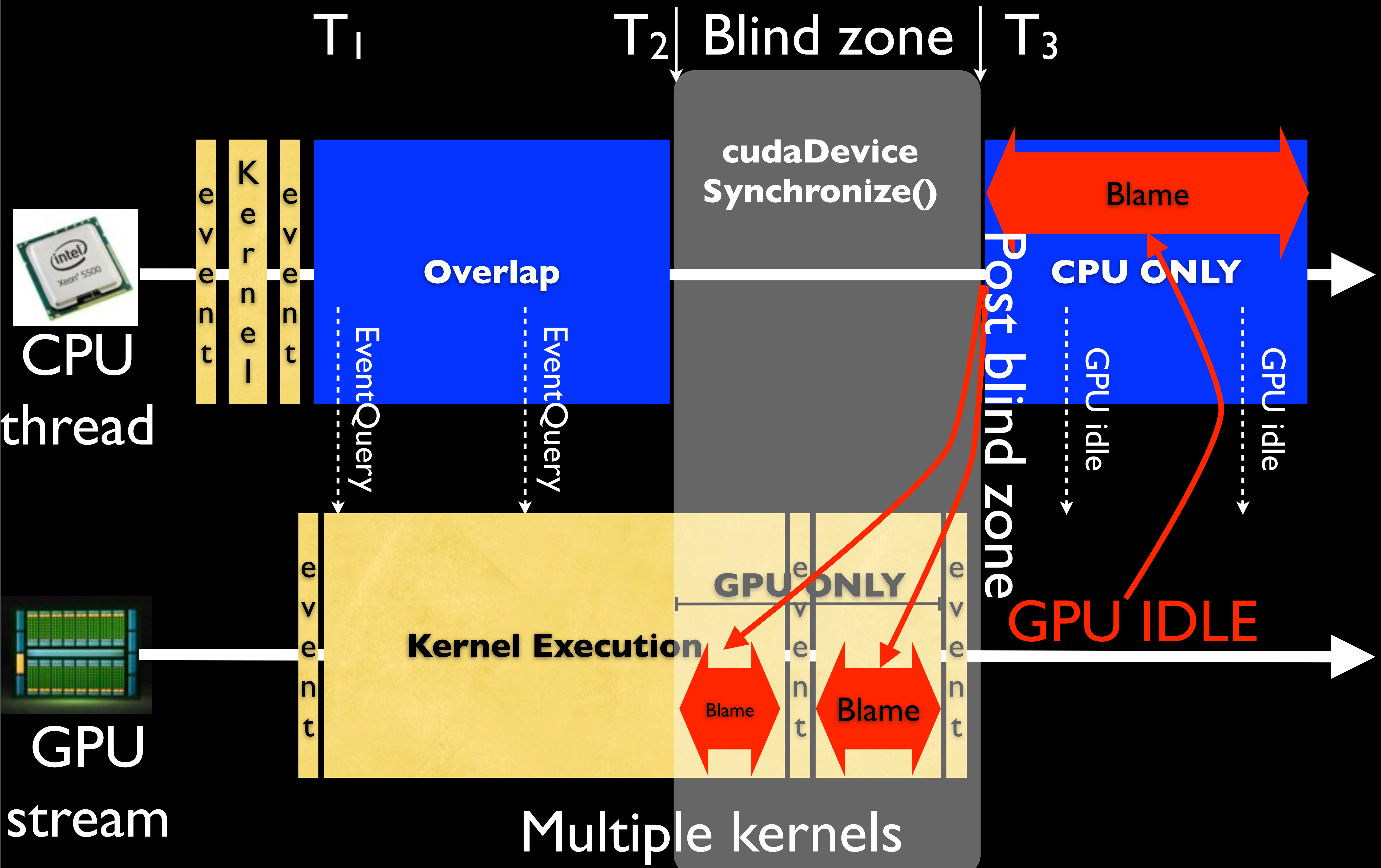
Workarounds: Deferred Blaming



Workarounds: Deferred Blaming



Workarounds: Deferred Blaming



HPCToolkit vs. TAU & Vampir
Time Overhead

Keeneland : Intel Westmere hex-core CPUs@2.8GHz, 24GB,
NVIDIA 6GB Tesla M 2090 GPUs, and a Qlogic QDR InfiniBand interconnect

Program	Base runtime	HPCToolkit		TAU		Vampir
		Profiling	Tracing	Profiling	Tracing	Tracing
LAMMPS rhodopsin protein in solvated lipid bilaye (32procs, 6 nodes, 6ppn, 3 gpu/node)	26.8264 sec	**8.9%** (29.2059s)	**10%** (29.5081s)	**3.1x** (83.6458s)	**3.3x** (89.5835s)	**156x** (4182.72s)
LULESH (1 node, 1proc, 1 gpu)	17.4887 sec	**4.1%** (18.2031s)	**5.8%** (18.5003s)	**47%** (25.7486s)	**44%** (25.1442s)	**5.2X** **(90.8506s)**

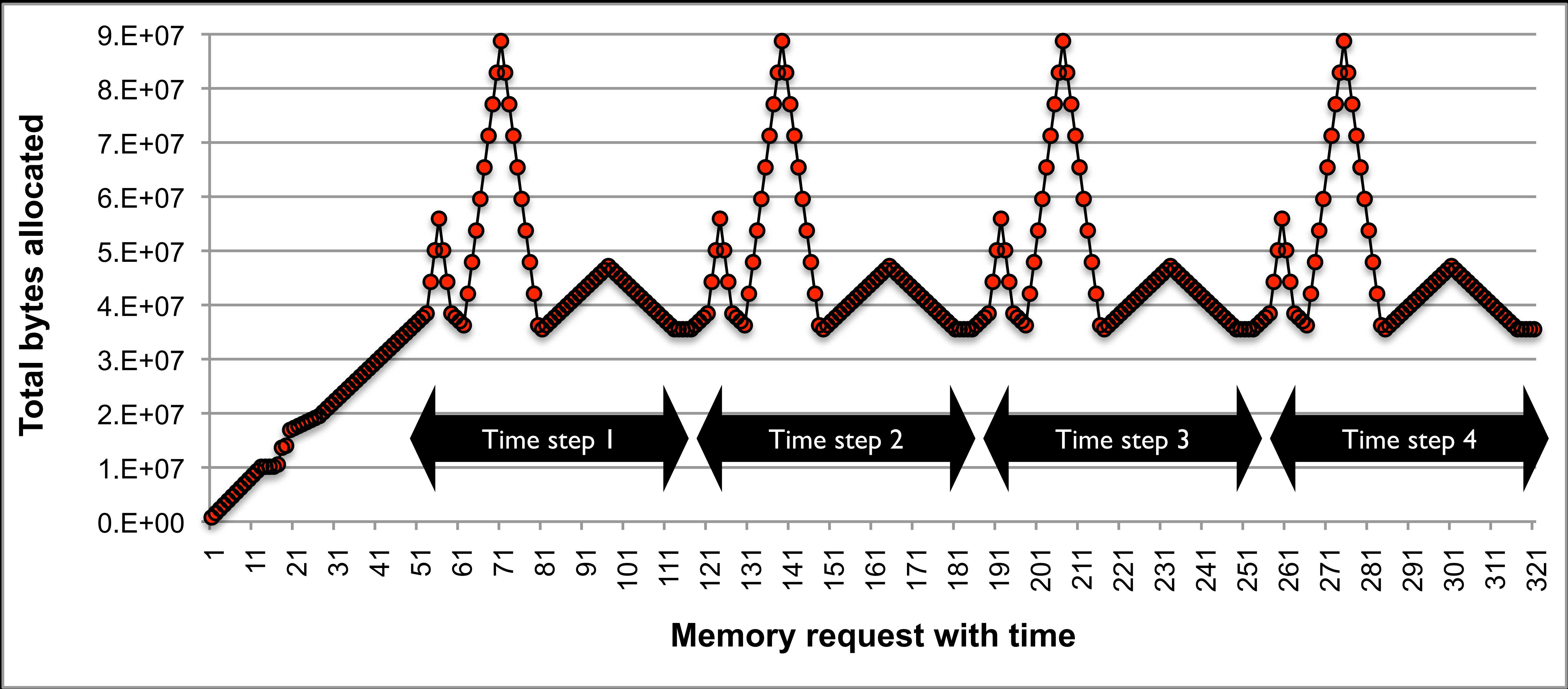
HPCToolkit vs. TAU & Vampir
Data Volume

Program	HPCToolkit		TAU		Vampir
	Profiling	Tracing	Profiling	Tracing	Tracing
LAMMPS (32procs, 6 nodes, 6ppn, 3 gpu/node)	16MB	57MB	**43x** (693MB)	**216x** (12GB)	**1491x** (83GB)
LULESH	268KB	4MB	**3.5x** (948KB)	**42.8x** (171MB)	**140.25x** (561MB)

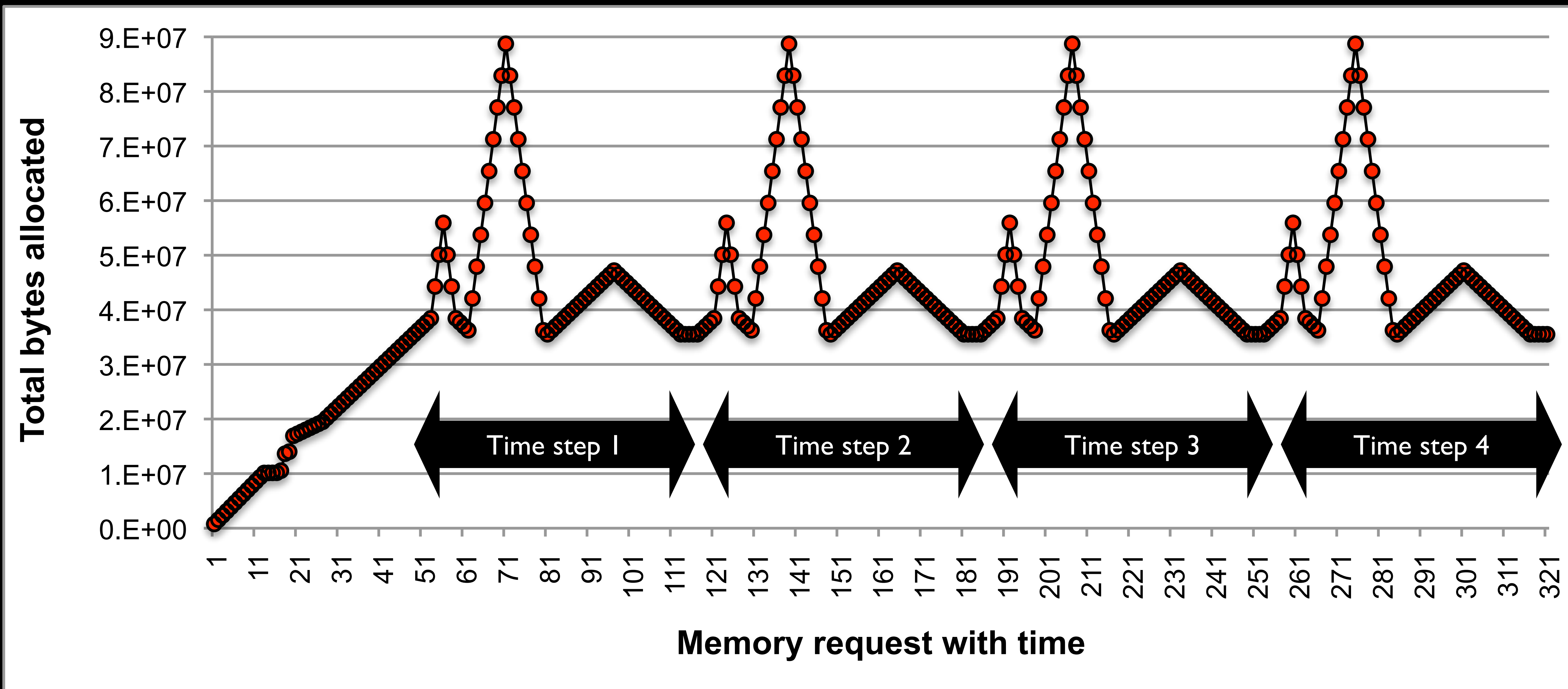
Insights via Blame Shifting in LULESH CUDA

- Simulations involving complex multi-material motion are one of the the most CPU time consuming applications
  - ✦ LULESH: classic hydro-dynamics code, solves Sedov blast wave problem with “leap frog” time integration scheme
- CUDA version available from LLNL
- DEMO

LULESH CUDA Memory Allocation



LULESH CUDA Memory Allocation



Replaced repeated memory allocation/free with a global allocation: **30%** running time improvement

LAMMPS on LJ

- Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS): Classical molecular dynamics code
- Two CUDA versions
  - ✦ GPU
    - ★ Designed to exploit common GPU hardware configurations wAtom-based data (e.g. coordinates, forces) moves back-and-forth between the CPU(s) and GPU every timestep.
    - ★ Neighbor lists can be constructed on the CPU or on the GPU
    - ★ The charge assignment and force interpolation portions of PPPM can be run on the GPU. The FFT portion runs on the CPU.
    - ★ Asynchronous force computations can be performed simultaneously on the CPU(s) and GPU.
  - ✦ USER-CUDA (all on GPU)
    - ★ Many timesteps, to run entirely on the GPU

Conclusions

- Hybrid CPU/GPU blame shifting with HPCToolkit
  - ✦ Provides novel and practical technique for performance analysis of heterogeneous systems
  - ✦ Pinpoints code fragments (CPU and GPU) worth tuning
    - ★ Improves developer productivity
  - ✦ Provides scalable performance measurement and analysis with low space and time overhead compared to state-of-the-art tools
- Several implementation challenges
  - ✦ Better API/hardware support from vendor can eliminate workarounds in all tools

Conclusions

- Hybrid CPU/GPU blame shifting with HPCToolkit
  - ✦ Provides novel and practical technique for performance analysis of heterogeneous systems
  - ✦ Pinpoints code fragments (CPU and GPU) worth tuning
    - ⋆ Improves developer productivity
  - ✦ Provides scalable performance measurement and analysis with low space and time overhead compared to state-of-the-art tools
- Several implementation challenges
  - ✦ Better API/hardware support from vendor can eliminate workarounds in all tools