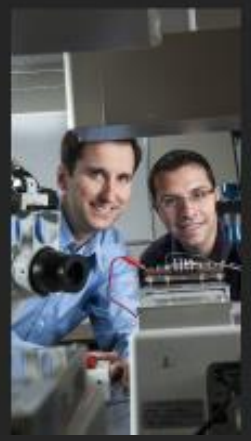





POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

GPU accelerated application tracing

David Couturier
B. Eng.

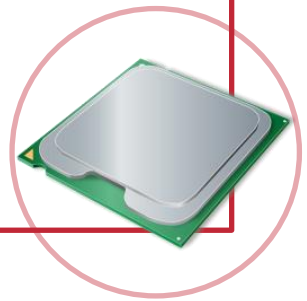
Updated in May 2015

CONTENT

CONTEXT: HETEROGENEOUS HARDWARE

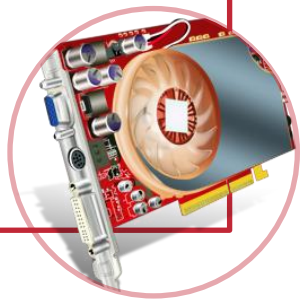
- 1 to 8 physical cores
- High frequency (< ~ 4 GHz)
- Serial computation
- 100s of GFLOPS

CPU



- Up to 3072 computation cores
- Moderated frequency (< ~ 1.2 GHz)
- Parallel computations (SIMD)
- 1000s of GFLOPS

GPU

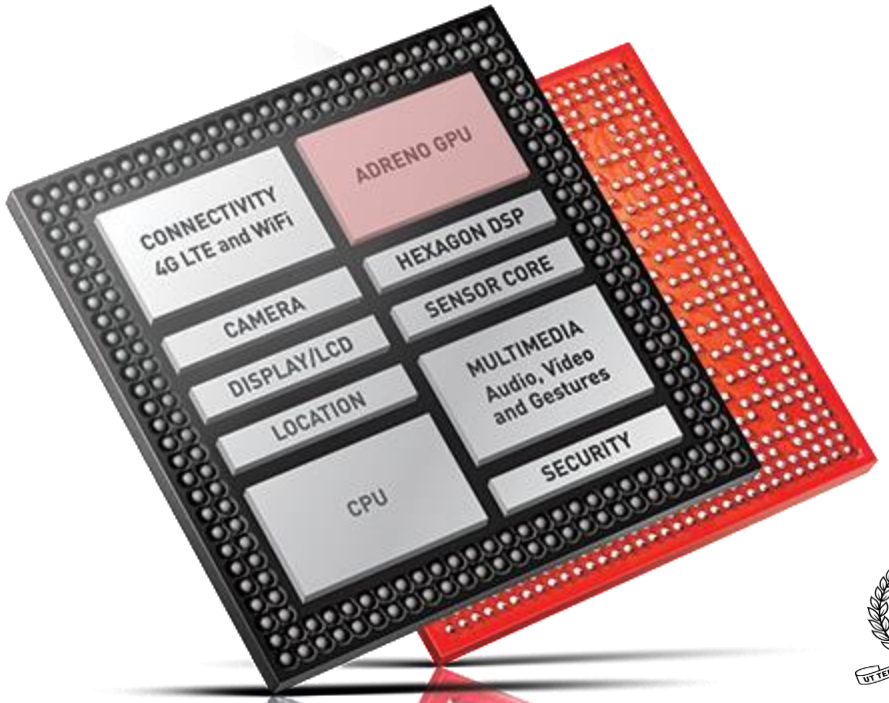


CONTEXT: HETEROGENEOUS HARDWARE

- NVIDIA : Tegra
- Intel : Intel HD Graphics
- Qualcomm : Snapdragon
- AMD : « APU » (Accelerated processing unit)

- Zynq (CPU + FPGA)
- XeonPhi

Snapdragon 810 (Qualcomm)

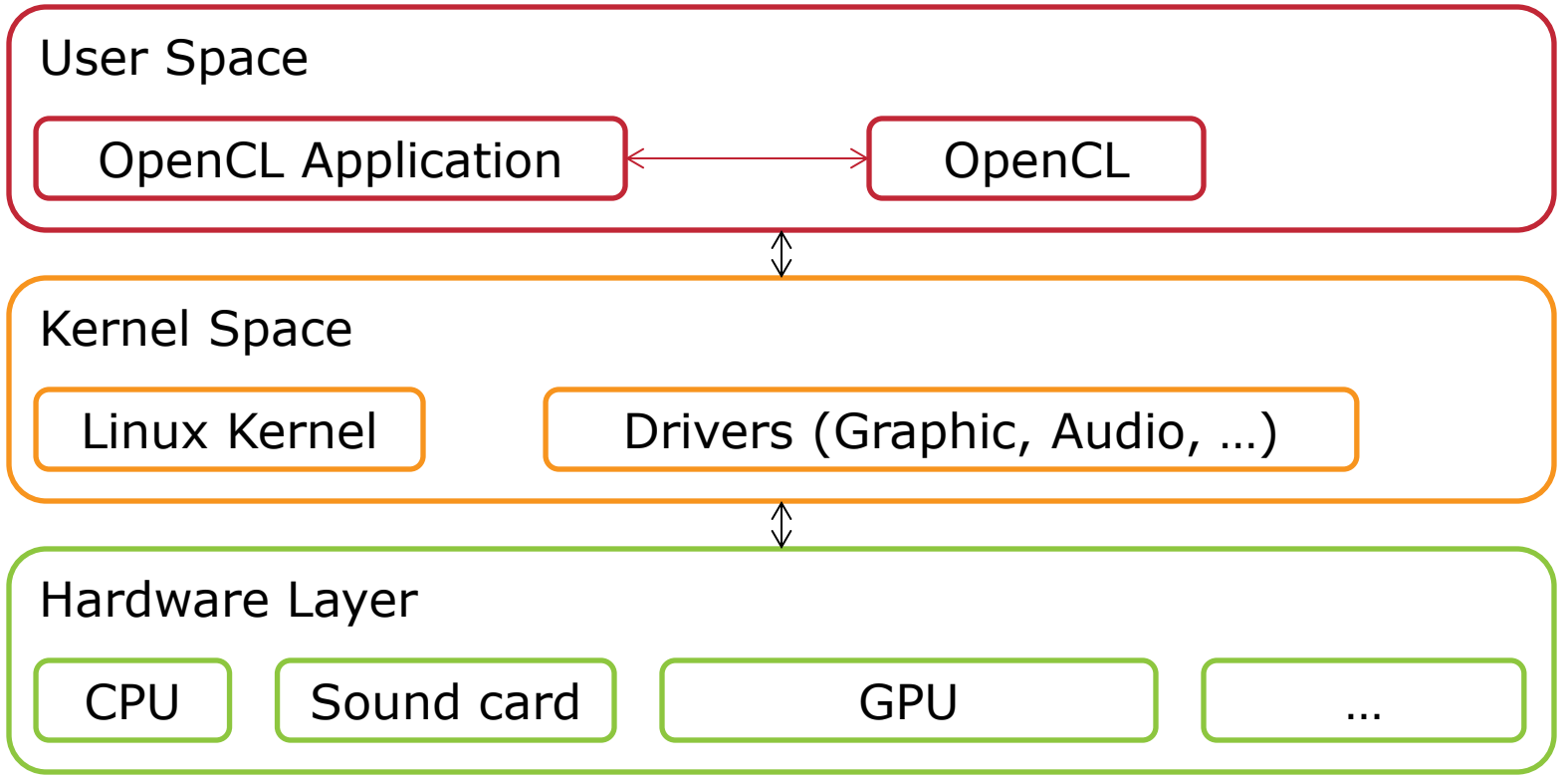


Source: qualcomm.com

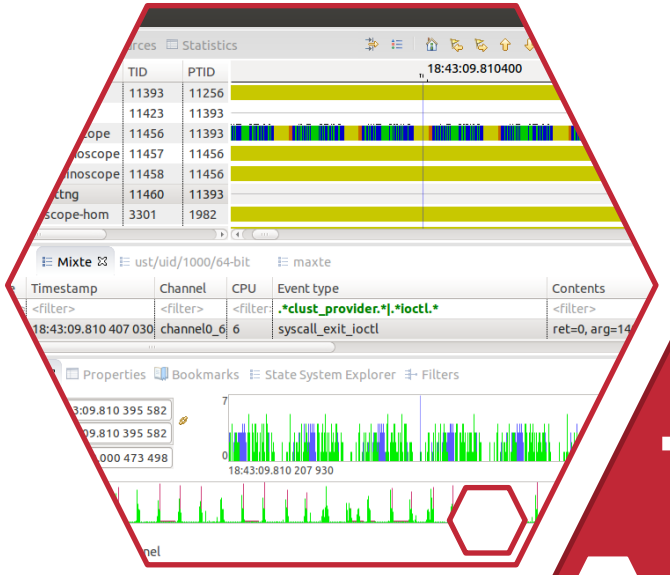
CONTEXT: ARCHITECTURE

Source: nvidia.com

CONTEXT: OPENCL

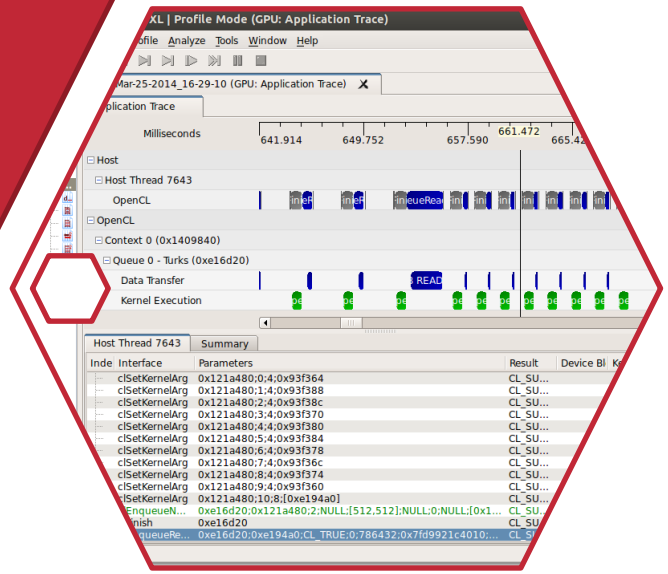


CONTEXT: TRACING



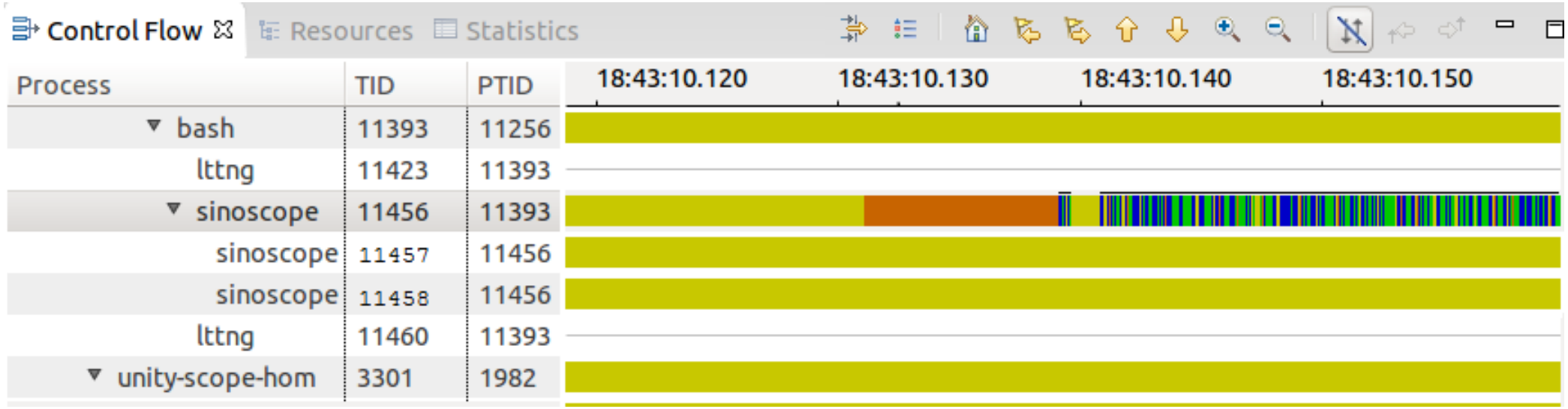
Trace CPU

Trace GPU



CONTEXT: CPU TRACE OF OPENCL APPLICATION

Trace Compass:

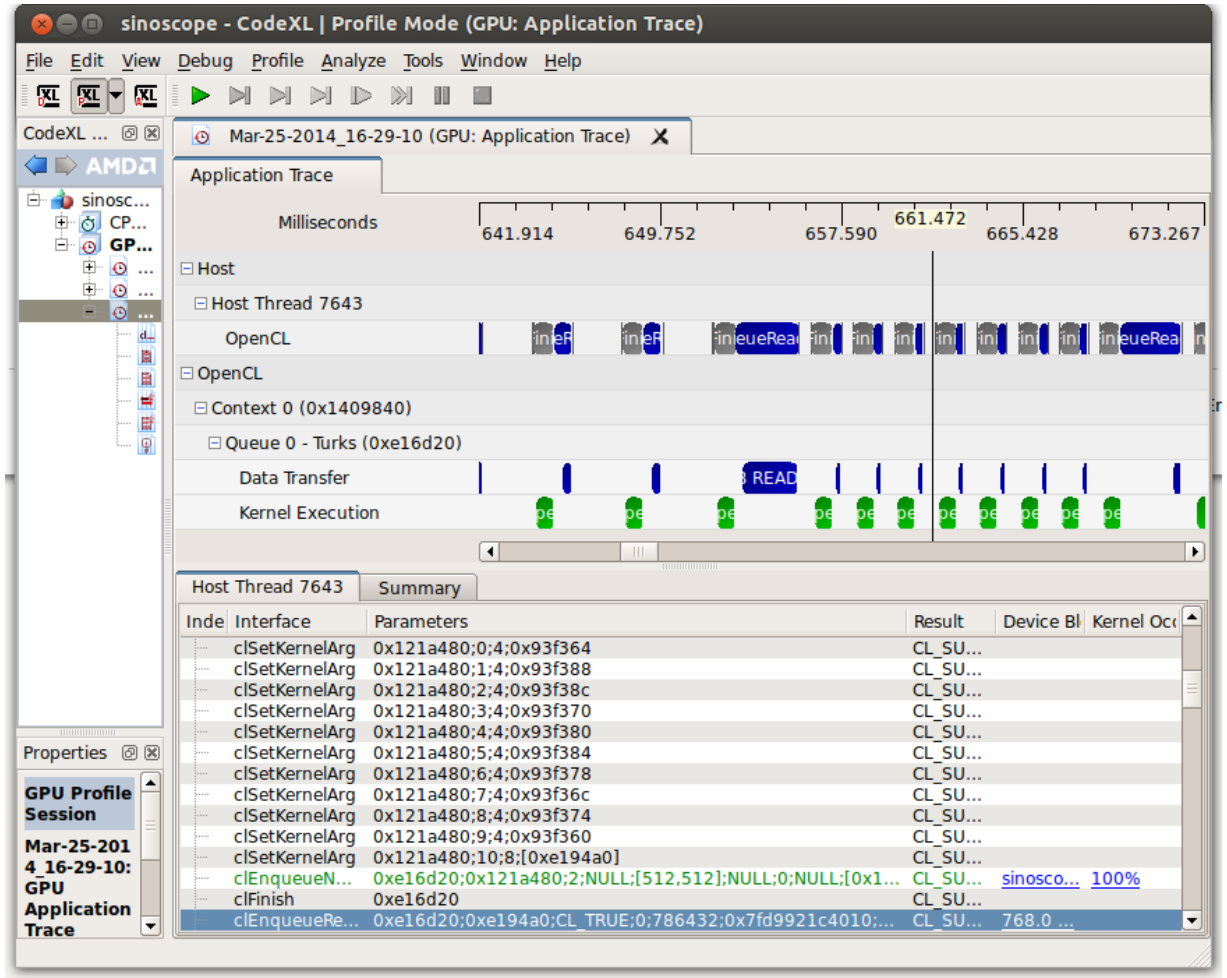


CONTEXT: GPU TRACE

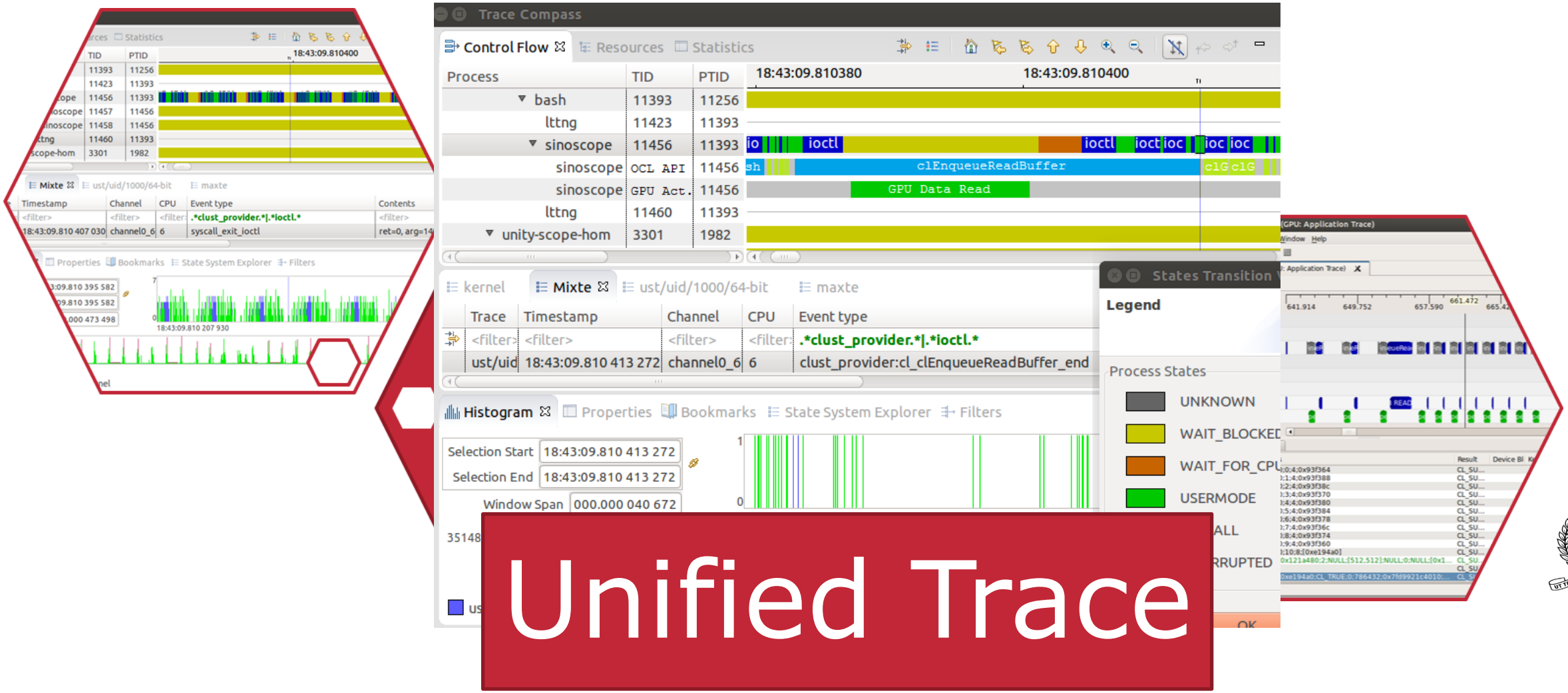
CodeXL (AMD's GPU tracing tool)

Limitations:

- OpenCL application has to be launched by CodeXL
- Recording trace performance for large traces
- AMD GPU only



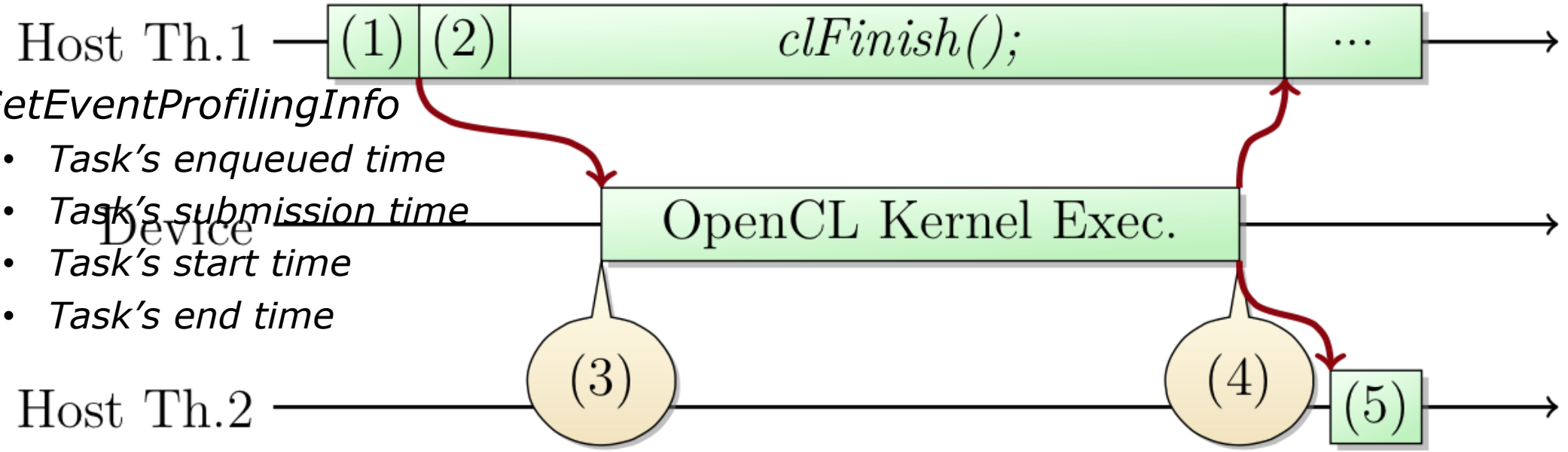
OBJECTIVE: UNIFIED TRACE



SECONDARY OBJECTIVES

- Performance
  - Minimal overhead on the system
- Problem solving

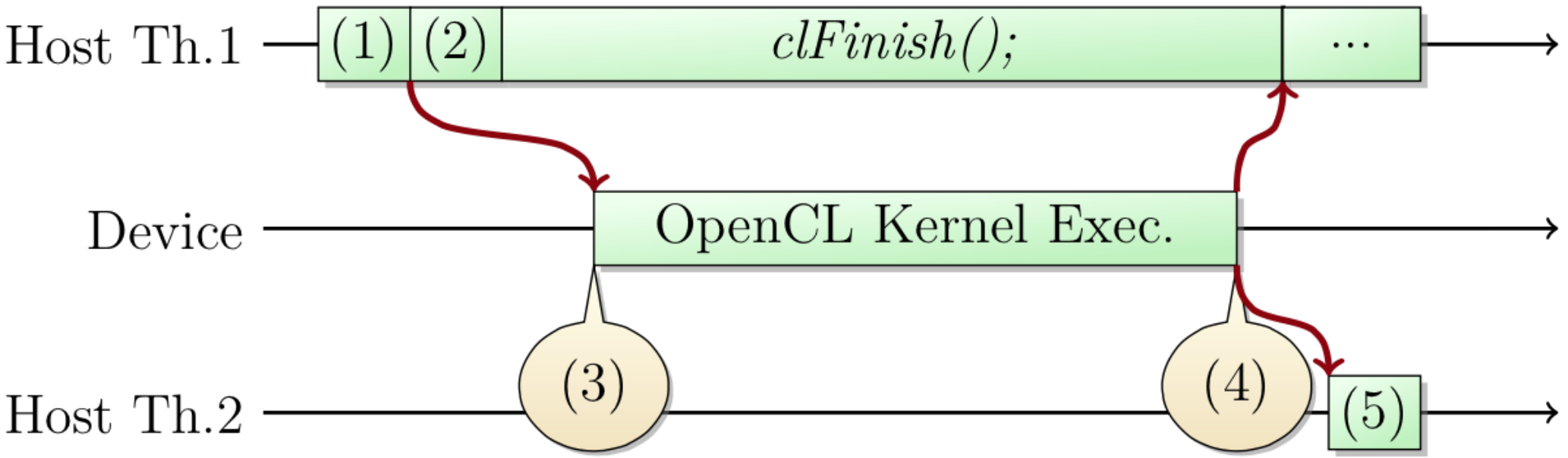
SOLUTION: OPENCL PROFILING



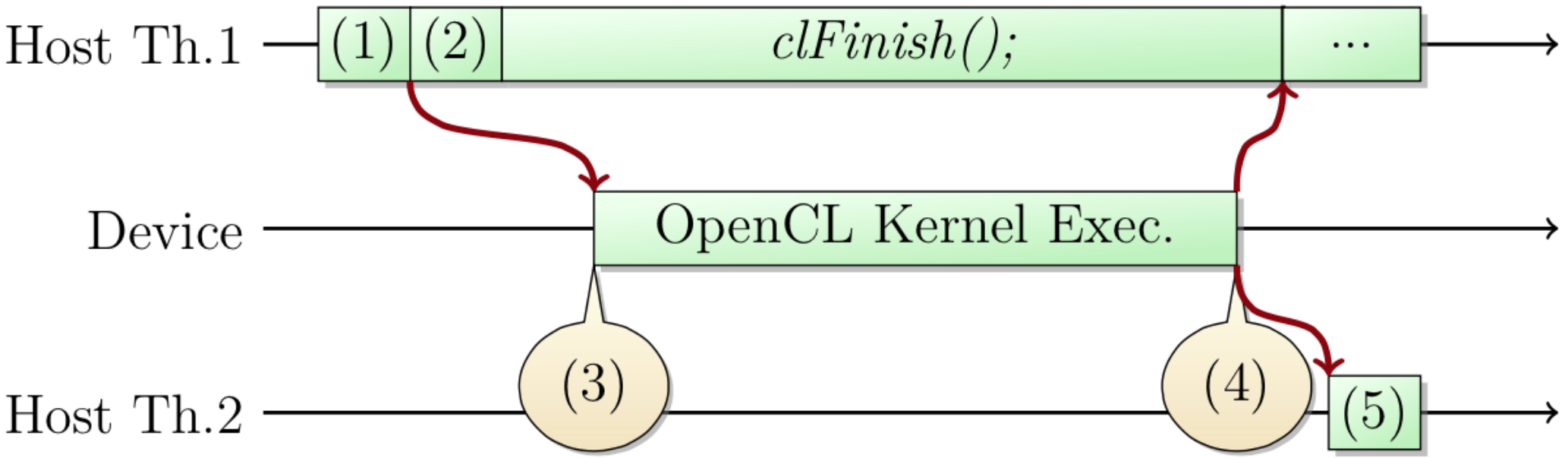
clGetEventProfilingInfo

- *Task's enqueued time*
- *Task's submission time*
- *Task's start time*
- *Task's end time*

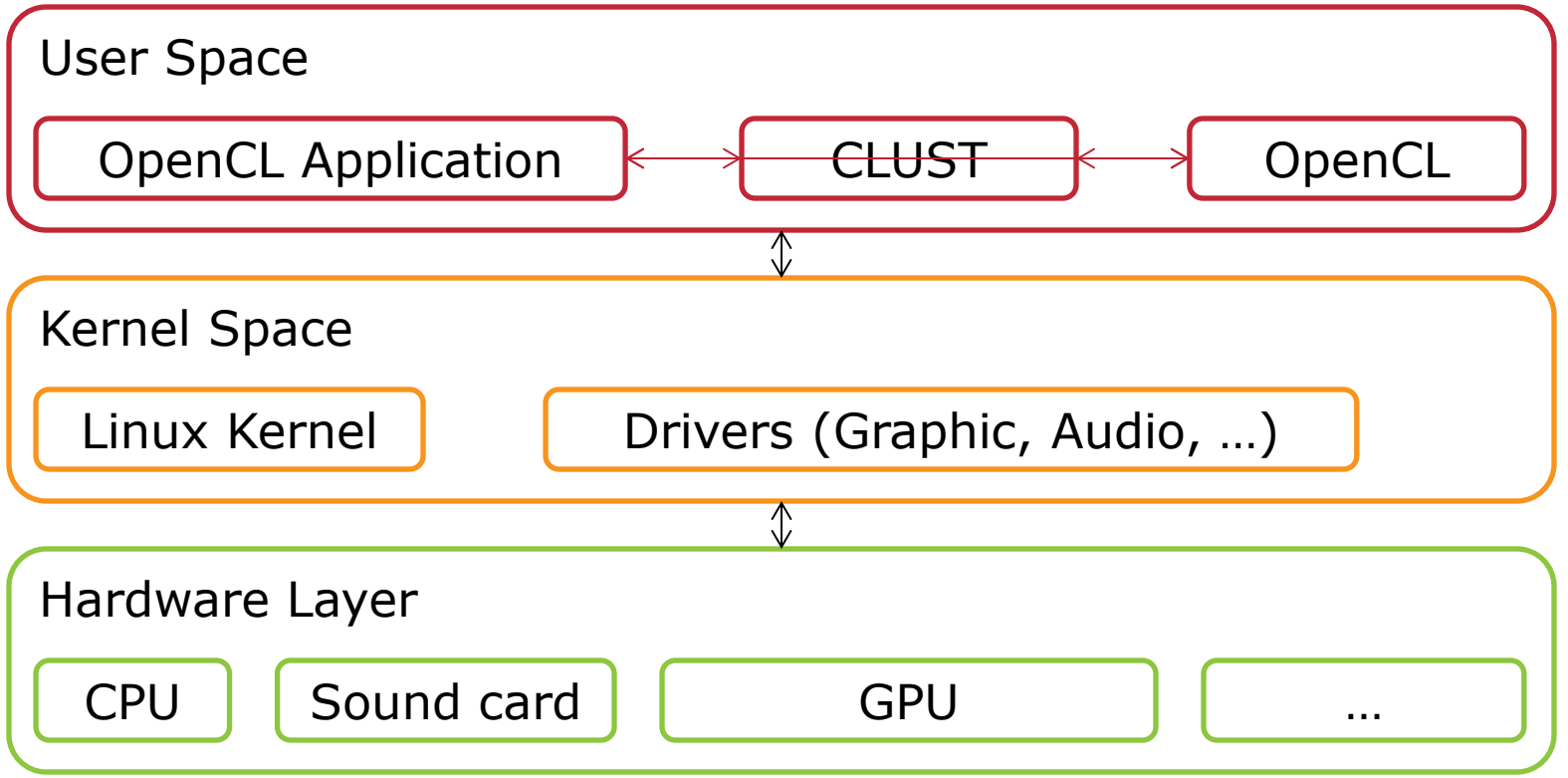
SOLUTION: NON MONOTONIC PROFILING



SOLUTION: HOST-DEVICE SYNCHRONIZATION



SOLUTION: DYNAMIC SYMBOL OVERLOADING



SOLUTION: SYNCHRONOUS API CALL

```
cl_int clGetPlatformIDs(cl_uint num_entries, cl_platform_id * platforms, cl_uint * num_platforms)  {
        tracepoint(clust_provider, cl_clGetPlatformIDs_start);
        cl_int ret = reallib_clGetPlatformIDs(num_entries, platforms, num_platforms);
        tracepoint(clust_provider, cl_clGetPlatformIDs_end);
        return ret;
}
```

SOLUTION: ASYNCHRONOUS API CALL

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read, size_t offset, size_t cb, void
        const bool trace = __tracepoint_clust_provider___clust_device_event.state;
        bool toDelete = false;
        if(caa_unlikely(trace)) {
                if(event == NULL) {
                        event = malloc(sizeof(cl_event));
                        toDelete = true;
                }
        }

        tracepoint(clust_provider, cl_clEnqueueReadBuffer_start);
        cl_int ret = reallib_clEnqueueReadBuffer(command_queue, buffer, blocking_read, offset, cb, ptr, num_events_in_wait_list
        tracepoint(clust_provider, cl_clEnqueueReadBuffer_end);

        if(caa_unlikely(trace)) {
                int r = reallib_clSetEventCallback(*event, CL_COMPLETE, &eventCompleted, (toDelete)?&ev_delete:&ev_keep);
                if(r != CL_SUCCESS) fprintf(stderr, "CLUST::clEnqueueReadBuffer->clSetEventCallback:error->%d\n", r);
        }

        return ret;
}
```

SOLUTION: ASYNCHRONOUS API CALL & CALLBACK FUNCTION

```
// Get event start time
cl_int ret = reallib_clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_START, sizeof(cl_ulong), &start, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUST::eventCompleted:error->CL_PROFILING_COMMAND_START returned %d\n", ret);
// Get event end time
ret = reallib_clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_END, sizeof(cl_ulong), &end, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUST::eventCompleted:error->CL_PROFILING_COMMAND_END returned %d\n", ret);
// Get event enqueue time
ret = reallib_clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_QUEUED, sizeof(cl_ulong), &queued, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUST::eventCompleted:error->CL_PROFILING_COMMAND_QUEUED returned %d\n", ret);
// Get event submit time
ret = reallib_clGetEventProfilingInfo(event, CL_PROFILING_COMMAND_SUBMIT, sizeof(cl_ulong), &submit, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUST::eventCompleted:error->CL_PROFILING_COMMAND_SUBMIT returned %d\n", ret);
// Get event command name (CL_COMMAND_NDRANGE_KERNEL, CL_COMMAND_WRITE_BUFFER, ...)
ret = reallib_clGetEventInfo(event,CL_EVENT_COMMAND_TYPE,sizeof(cl_command_type), &command, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUST::eventCompleted:error->CL_EVENT_COMMAND_TYPE returned %d\n", ret);
// Get event queue id
ret = reallib_clGetEventInfo(event,CL_EVENT_COMMAND_QUEUE,sizeof(cl_command_queue), &queue, NULL);
if(ret != CL_SUCCESS) fprintf(stderr, "CLUST::eventCompleted:error->CL_EVENT_COMMAND_QUEUE returned %d\n", ret);
// Record with UST tracepoint
tracepoint(clust_provider, clust_device_event, (ulong)queue, command, queued, submit, start, end);
if(*releaseEvent == ev_delete) {
        reallib_clReleaseEvent(event);
}
```

METHODOLOGY
CONFIGURATION

- Intel i7-4770 with HD Graphics 4600 integrated graphics
- 32 GB DDR3 RAM
- Ubuntu 14.04 (Kernel 3.18.4 + patch)
- Beignet v1.0.2 OpenCL drivers + patch
- LTTng v2.6.0-rc1

- Monotonic clock

- 1000s of iterations measured many times to acquire statistics

METHODOLOGY
MEASURING SYNCHRONOUS FUNCTION OVERHEAD

```
clGetPlatformIDs(num_entries, platforms, num_platforms);
```

METHODOLOGY
MEASURING ASYNCHRONOUS FUNCTION OVERHEAD

```
clEnqueueReadBuffer(command_queue, buffer, blocking_read, offset, cb, ptr, num_events_in_wait_list
```

RESULTS: CLUST OVERHEAD
SYNCHRONOUS FUNCTIONS

Qty. (iterations)	Reference (ns)	Preload (ns)	UST Tracing (ns)	Preload Overhead (ns)	UST Tracing Overhead (ns)
10^0	16	18	383	2	367
10^1	5.2	7.8	366.5	2.6	361.3
10^2	4.64	6.66	365.68	2.02	361.04
10^3	4.291	6.058	365.168	1.767	360.877
10^4	4.277	6.283	359.780	2.006	355.503
10^5	4.526	6.484	359.379	1.958	354.853
10^6	4.531	6.467	363.313	1.936	358.782
10^7	4.537	6.499	361.145	1.962	356.608
10^8	4.535	6.460	361.108	1.925	356.573

→ ~ 1 ns overhead per inactive UST tracepoint
→ ~ 180 ns overhead per active UST tracepoint

RESULTS: CLUST OVERHEAD
ASYNCHRONOUS FUNCTIONS

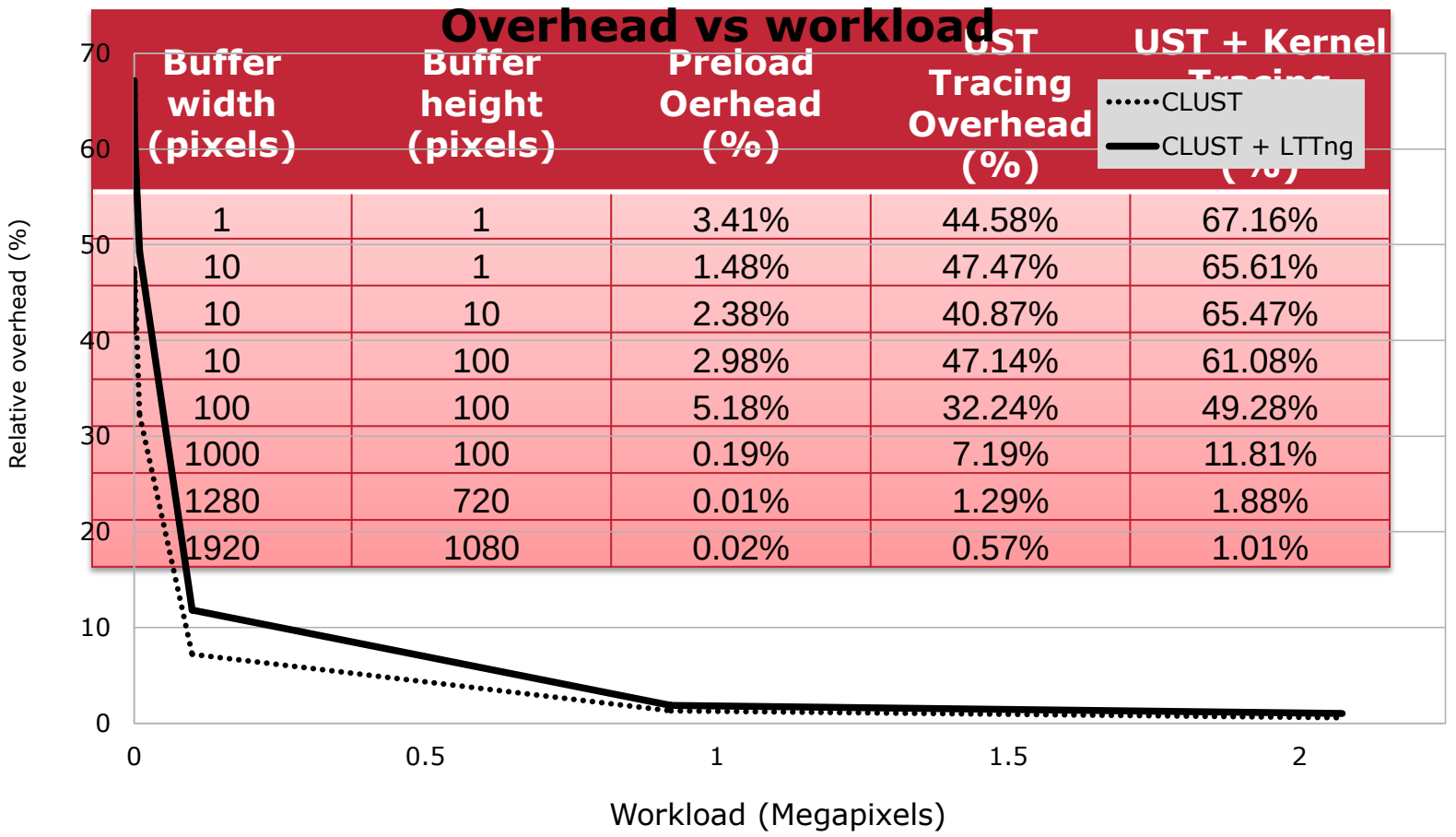
Buffer size (byte)	Reference (ns)	Preload (ns)	UST Tracing (ns)	Preload Overhead (ns)	Tracing overhead (ns)
$4x10^0$	149.51	164.7	7000.6	15.2	6851.1
$4x10^1$	158.99	168.7	7026.8	9.7	6867.8
$4x10^2$	156.15	174.7	7269.3	18.5	7113.2
$4x10^3$	188.44	226.7	7043.6	38.3	6855.2
$4x10^4$	1499.76	1503.3	8393.0	3.6	6893.3
$4x10^5$	17805.67	17862.1	25404.7	56.4	7599.0

→ [3.6, 56.4] ns overhead per « preloaded » asynchronous call*
→ ~ 7030 ns overhead per traced asynchronous call

RESULTS: CLUST OVERHEAD
REAL OPENCL APPLICATION

Buffer width (pixels)	Buffer height (pixels)	Reference (ns)	Preload (ns)	UST Tracing (ns)	UST + Kernel Tracing (ns)
1	1	35198	36399	50890	58838
10	1	35183	35702	51883	58265
10	10	36031	36890	50758	59619
10	100	37937	39067	55820	61108
100	100	56770	59709	75073	84746
1000	100	250694	251165	268726	280299
1280	720	1951826	1951965	1976916	1988445
1920	1080	4466096	4466777	4491589	4511394

RESULTS: CLUST OVERHEAD
REAL OPENCL APPLICATION (…)



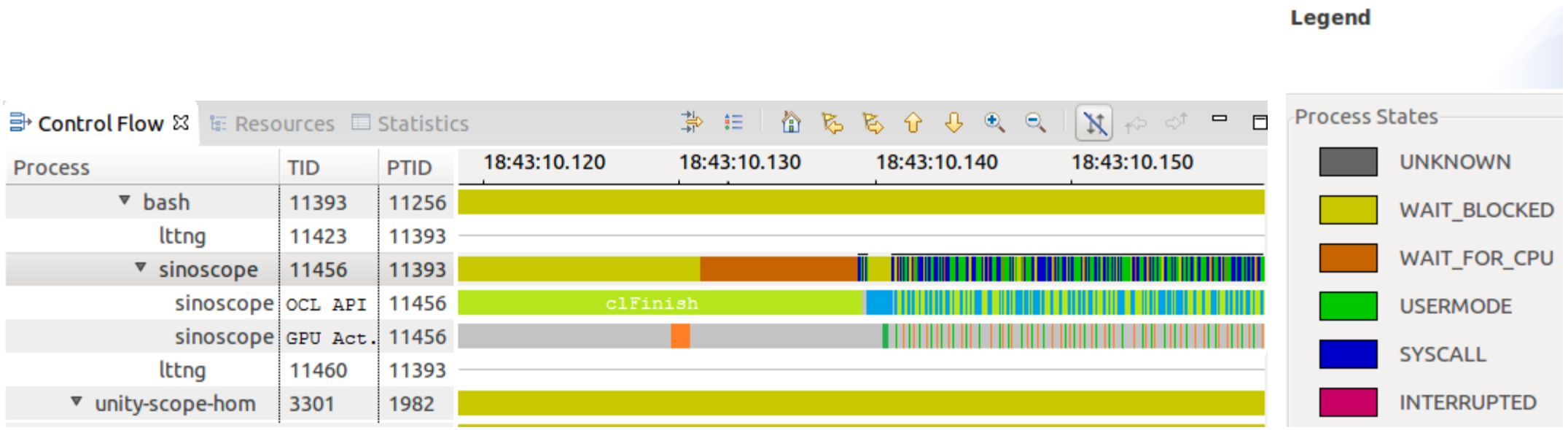
Buffer width (pixels)	Buffer height (pixels)	Preload Oerhead (%)	UST Tracing Overhead (%)	UST + Kernel Tracing (%)
1	1	3.41%	44.58%	67.16%
10	1	1.48%	47.47%	65.61%
10	10	2.38%	40.87%	65.47%
10	100	2.98%	47.14%	61.08%
100	100	5.18%	32.24%	49.28%
1000	100	0.19%	7.19%	11.81%
1280	720	0.01%	1.29%	1.88%
1920	1080	0.02%	0.57%	1.01%

USE CASES

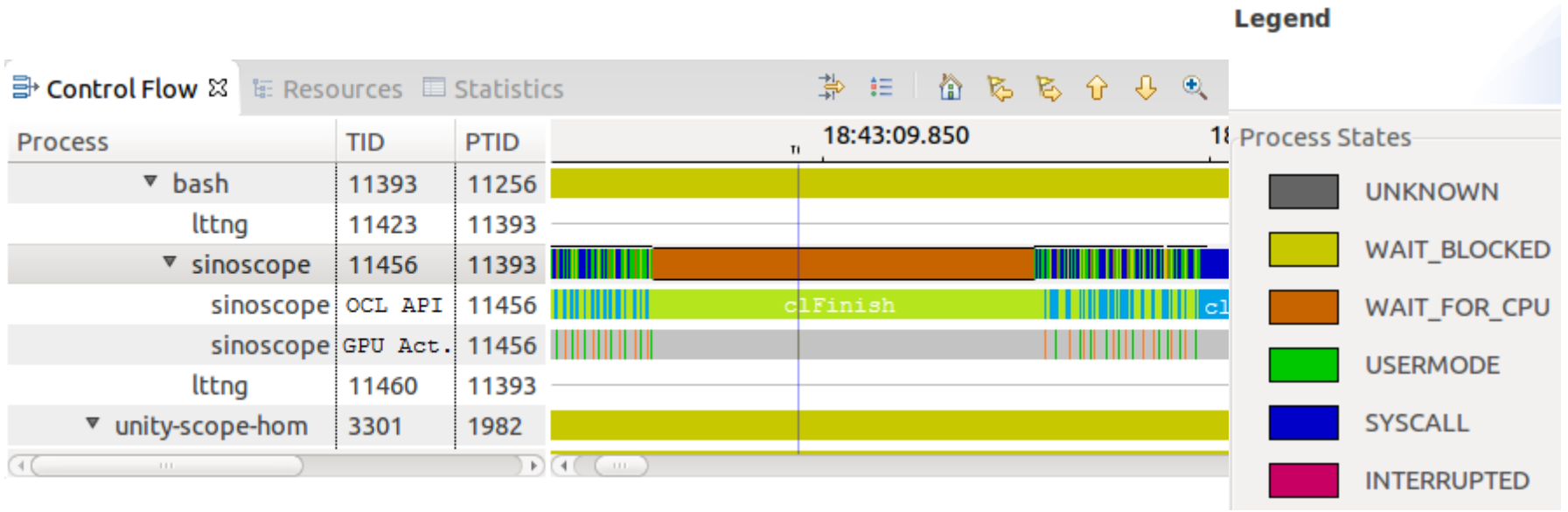
- System-wide unified tracing
- Flight recording mode
  - 24/7 volatile recording
  - Hard drive dump when required

- OpenCL application optimization
- OpenCL application debugging

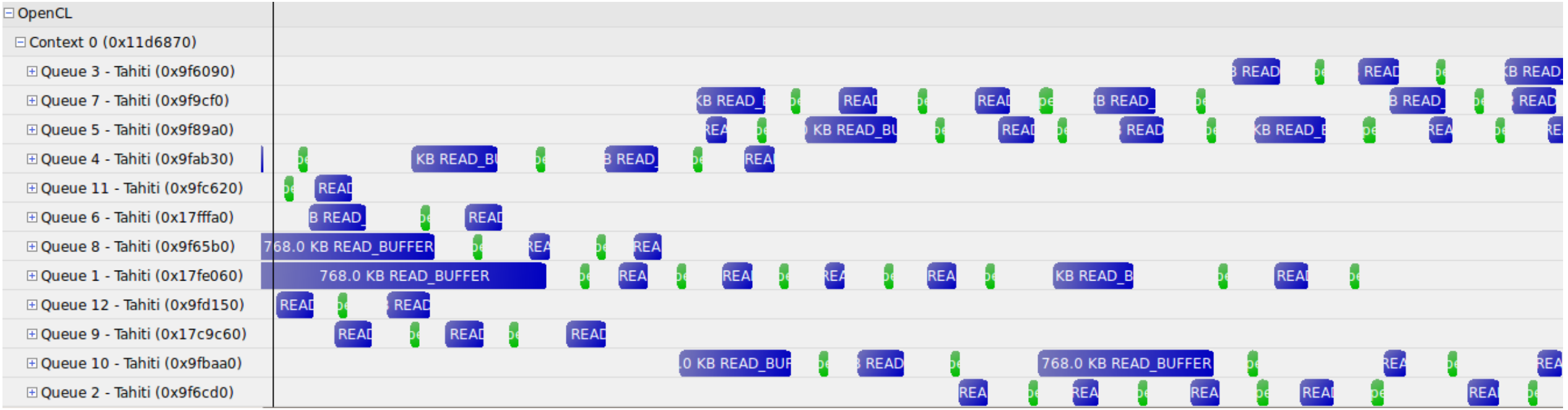
USE CASE: RESSOURCE SHARING



USE CASE: CPU PREEMPTION



USE CASE: OPENCL PIPELINE USAGE MAXIMISATION



FUTURE WORK

- Trace analysis utilities
- Hardware performance counter data acquisition
- OpenCL 2.0 support
- OpenGL tracing
  - Vulkan API
- Heterogeneous computing framework
  - CPU-GPU
  - CPU-DSP
  - …







POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Questions?







POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Thank you!