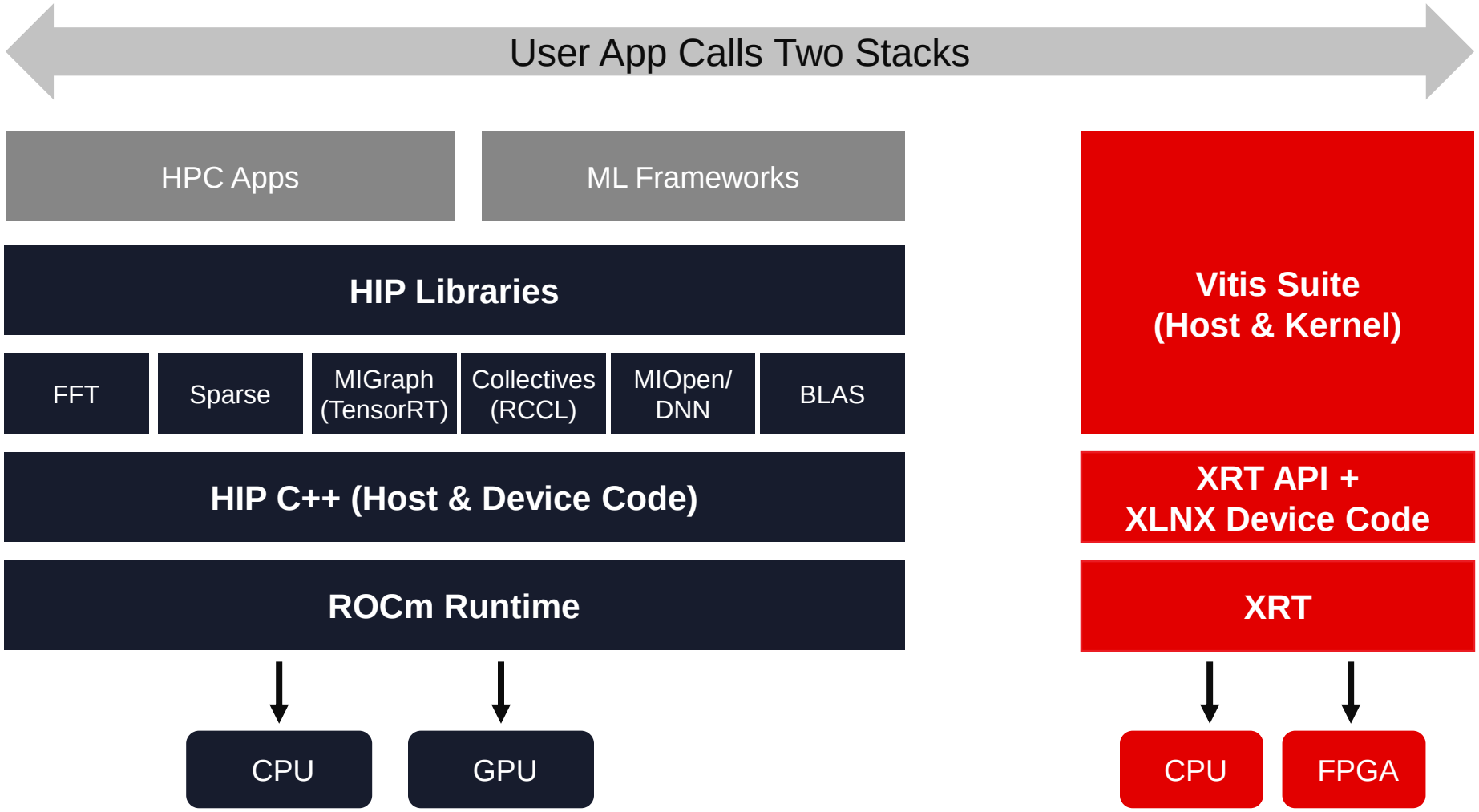
XILINX

AMD + Xilinx Converged Runtime
Technology Demonstration

CTO Team

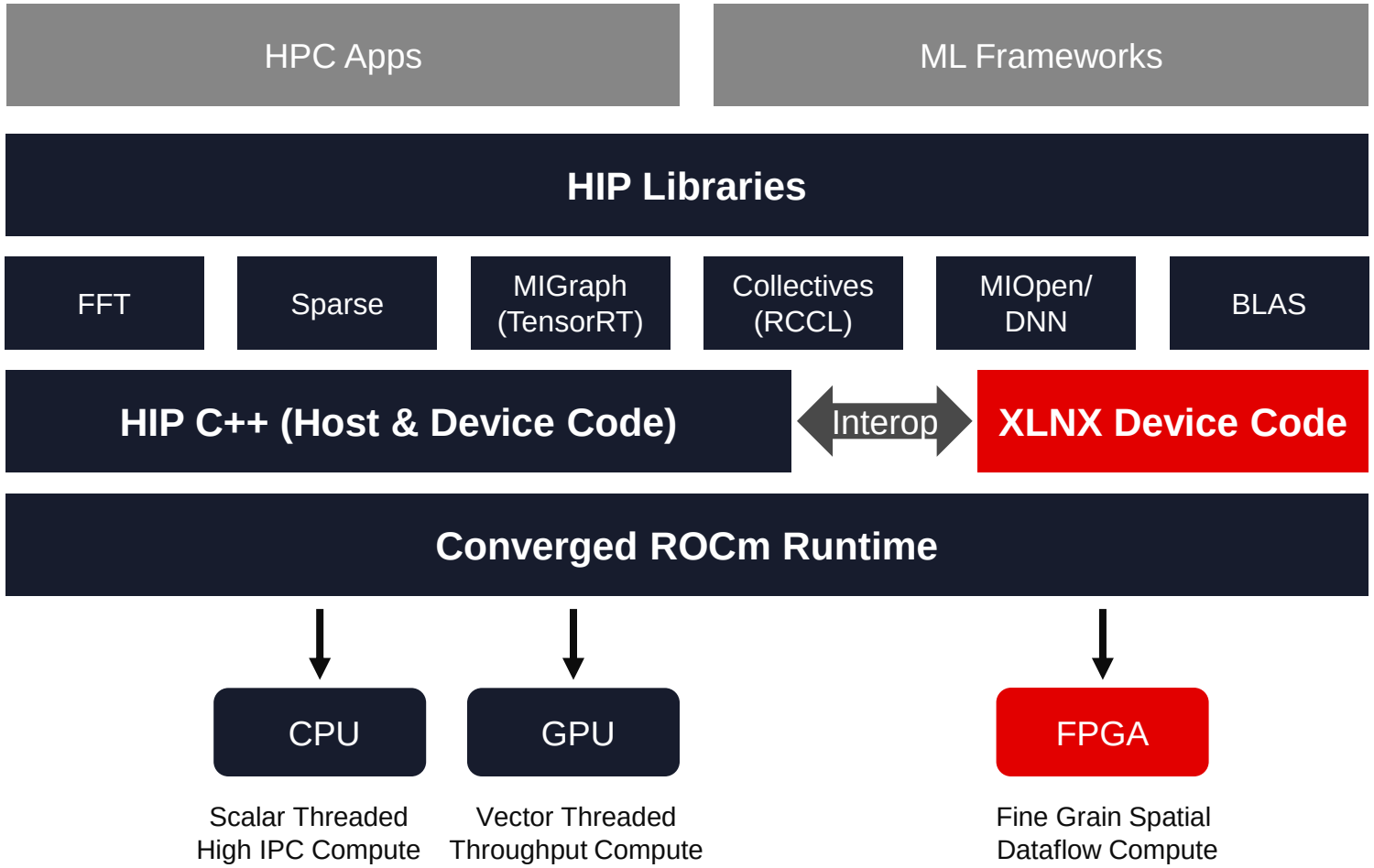
SuperComputing 2020

Current: Two Stacks



XILINX.

Preview: ROCm Runtime Integration

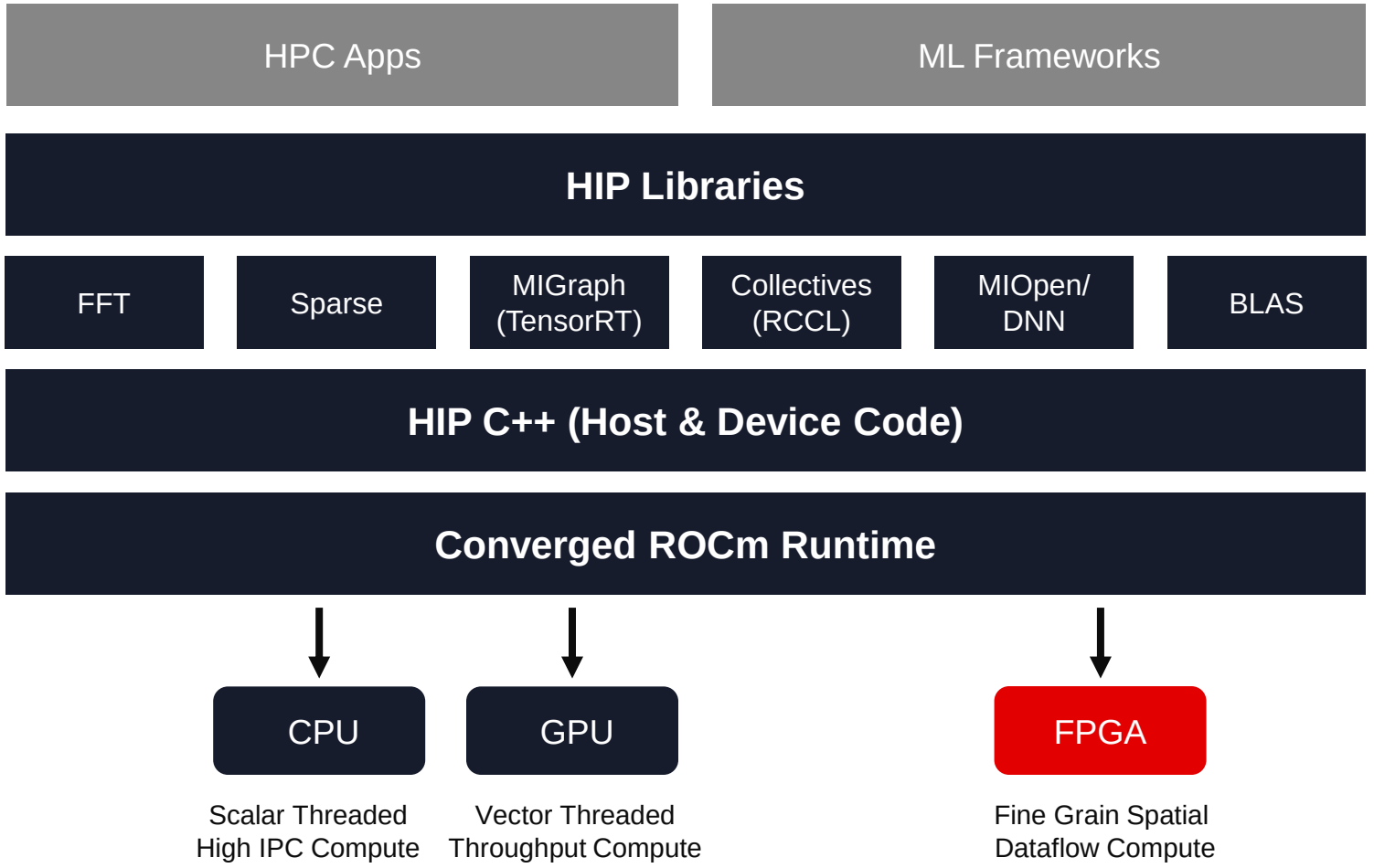


ROCm

- **ROCm Runtime Integration**
  - Unified Resource Discovery
  - Unified Dispatch & Sync
  - Shared Virtual Memory
- **XLNX Device Languages**
  - HLS C, AIE C

XILINX.

In Development: Integrated SW Stack



ROCm

- **ROCm Runtime Integration**
  - Unified Resource Discovery
  - Unified Dispatch & Sync
  - Shared Virtual Memory
- **Unified Device Languages**
  - Thread, Pipeline, Spatial
- **ROCm Library Integration**
  - Harmonized Library APIs
  - Harmonized Framework Plugins

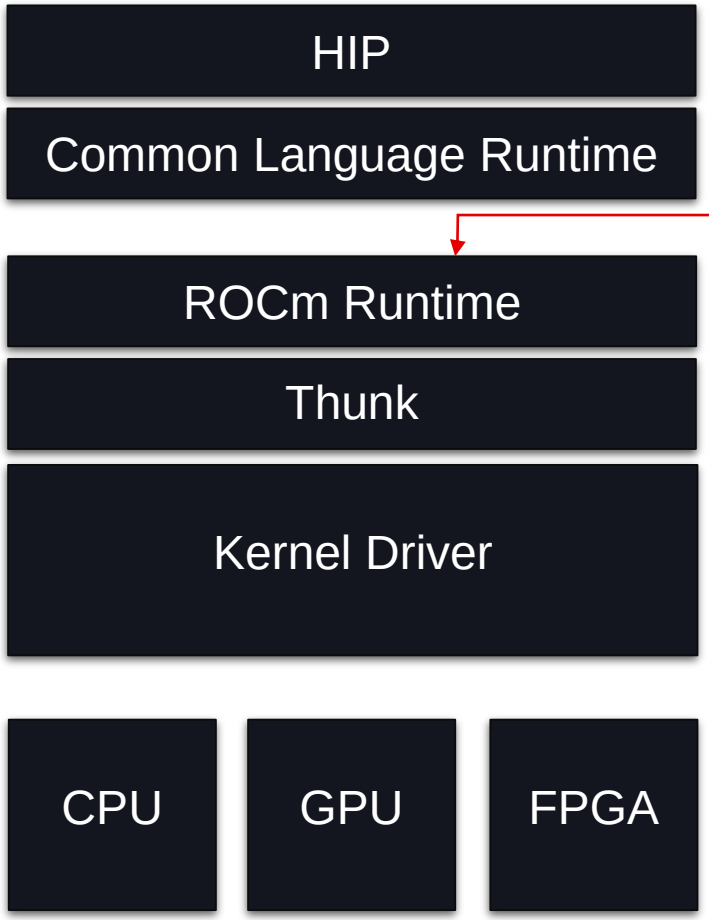
Unified ROCm Runtime for GPUs and FPGAs

- **Unified Device and Topology Discovery**
  - Runtime resource reservation
  - Scale-out
- **Unified Dispatch and Synchronization**
  - Common user-mode queuing architectural queuing model
  - Peer-to-peer events
- **Shared Virtual Memory**
  - Common virtual address space
  - Visibility of FPGA buffers from GPU and CPU

Demonstration of Capabilities

Demo 1: Device Discovery

ROCm Runtime discovers CPUs, GPUs and FPGAs in consistent way




```
36  static hsa_status_t show_name(hsa_agent_t agent, void *data) {
37    char name[64] = { 0 };
38    hsa_status_t status = hsa_agent_get_info(agent, HSA_AGENT_INFO_NAME, name);
39    printf("Found agent %s\n", name);
40    return status;
41  }
42
43  int main(int argc, char ** argv) {
44
45    hsa_status_t err = hsa_init();
46
47    printf("Initialized\n");
48
49    // Display the agents found in the system
50    hsa_agent_t agent;
51    err = hsa_iterate_agents(show_name, &agent);
52
```

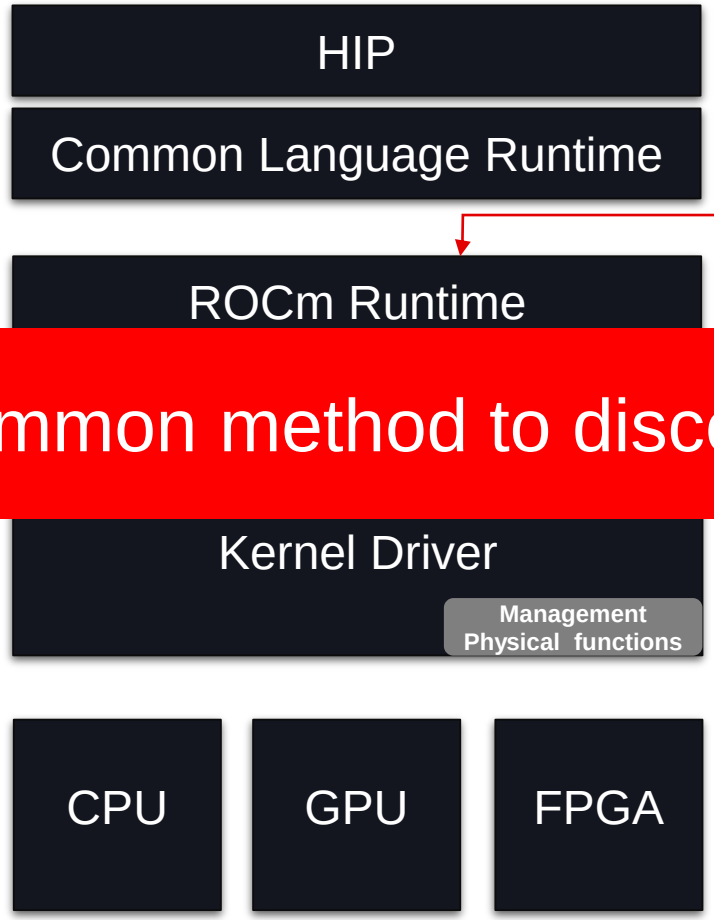
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Initialized
Found agent AMD Ryzen 9 3900X 12-Core Processor
Found agent gfx803
Found agent Xilinx U280
```


XILINX.

Demo 1: Device Discovery

ROCm Runtime discovers CPUs, GPUs and FPGAs in consistent way




```
36  static hsa_status_t show_name(hsa_agent_t agent, void *data) {
37    char name[64] = { 0 };
38    hsa_status_t status = hsa_agent_get_info(agent, HSA_AGENT_INFO_NAME, name);
39    printf("Found agent %s\n", name);
40    return status;
41  }
42
43  int main(int argc, char ** argv) {
```

Common method to discover and reserve CPU, GPU and FPGA resources

```
49    // Display the agents found in the system
50    hsa_agent_t agent;
51    err = hsa_iterate_agents(show_name, &agent);
52
```

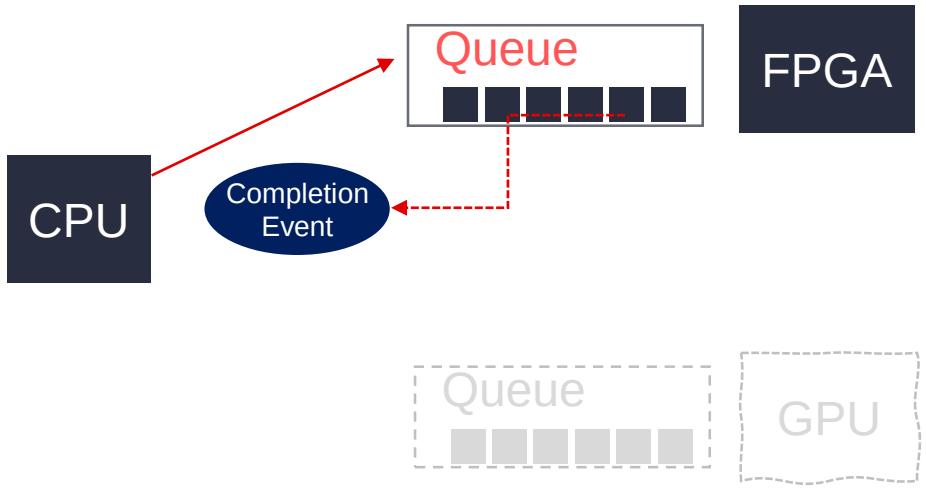
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Initialized
Found agent AMD Ryzen 9 3900X 12-Core Processor
Found agent gfx803
Found agent Xilinx U280
```


XILINX.

Demo 2: Kernel Dispatch and Synchronization
Usermode Queues and Events

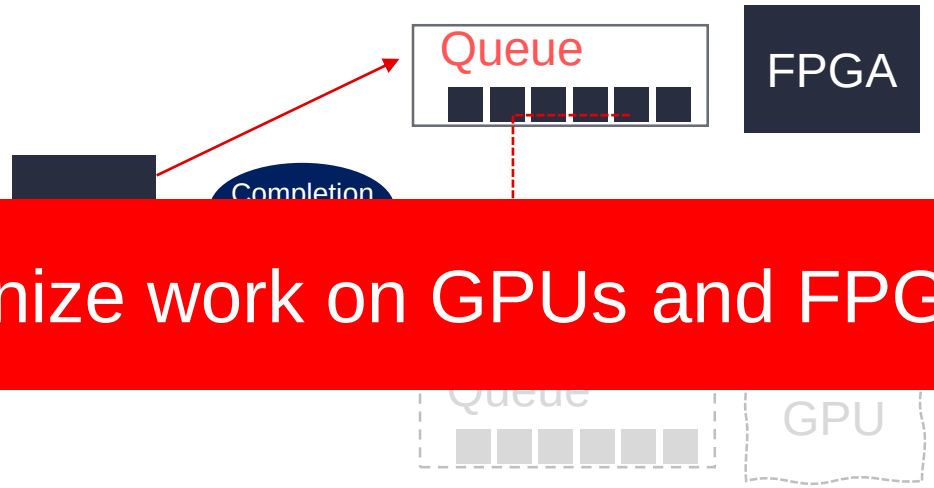
```
11  int main(int argc, char ** argv) {
12
13    hsa_status_t status = hsa_init();
14
15    hsa_agent_t fpga_agent[1];
16    status = hsa_iterate_agents(get_fpga_agent, fpga_agent);
17
18    hsa_queue_t * queue = malloc(sizeof(hsa_queue_t));
19
20    // create a user mode queue on FPGA
21    status = hsa_queue_create(
22        fpga_agent[0], 256,
23        HSA_QUEUE_TYPE_SINGLE,0,0,0,0,
24        &queue );
25
26    hsa_signal_t completion_signal;
27    status = hsa_signal_create(1 /* Initial Value */ ,1, fpga_agent, &completion_signal);
28
29    uint64_t index = hsa_queue_load_write_index_relaxed(queue);
30    const uint32_t mask = queue->size -1;
31    hsa_agent_dispatch_packet_t * packet =
32        &(((hsa_agent_dispatch_packet_t*)(queue->base_address))[index & mask]);
33
34      // set parameters in packet
35    packet->completion_signal = completion_signal;
36    hsa_queue_store_write_index_relaxed(queue, index+1);
37
38    hsa_signal_wait_scacquire(completion_signal,
39                              HSA_SIGNAL_CONDITION_EQ, 0, 2000,
40                              HSA_WAIT_STATE_ACTIVE);
41  printf("Success\n");
42  }
```



XILINX.

Demo 2: Kernel Dispatch and Synchronization
Usermode Queues and Events

```
11  int main(int argc, char ** argv) {
12
13    hsa_status_t status = hsa_init();
14
15    hsa_agent_t fpga_agent[1];
16    status = hsa_iterate_agents(get_fpga_agent, fpga_agent);
17
18    hsa_queue_t * queue = malloc(sizeof(hsa_queue_t));
19
20    // create a user mode queue on FPGA
21    status = hsa_queue_create(
28
29    uint64_t index = hsa_queue_load_write_index_relaxed(queue);
30    const uint32_t mask = queue->size -1;
31    hsa_agent_dispatch_packet_t * packet =
32        &(((hsa_agent_dispatch_packet_t*)(queue->base_address))[index & mask]);
33
34      // set parameters in packet
35    packet->completion_signal = completion_signal;
36    hsa_queue_store_write_index_relaxed(queue, index+1);
37
38    hsa_signal_wait_scacquire(completion_signal,
39                              HSA_SIGNAL_CONDITION_EQ, 0, 2000,
40                              HSA_WAIT_STATE_ACTIVE);
41  printf("Success\n");
42  }
```

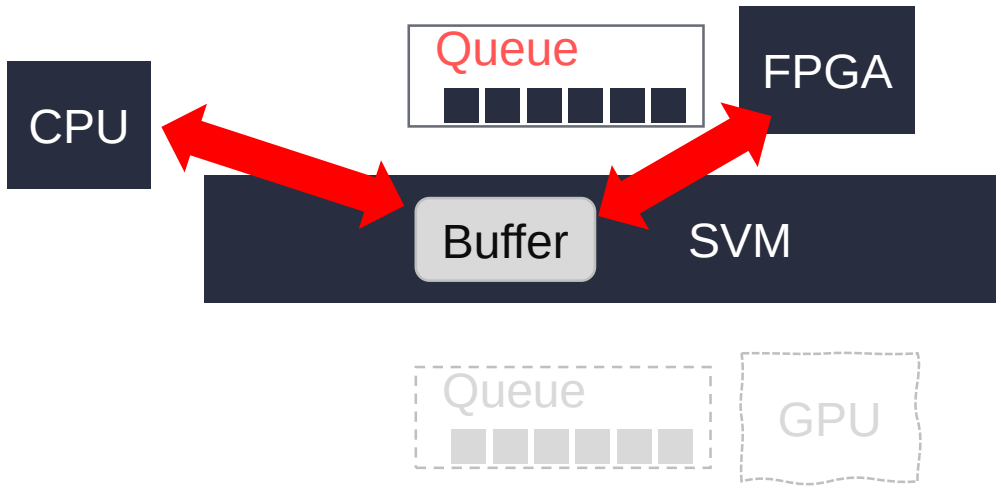


Common method to dispatch and synchronize work on GPUs and FPGAs

XILINX.

Demo 3: Kernel

Queues and Events using Shared Virtual Memory

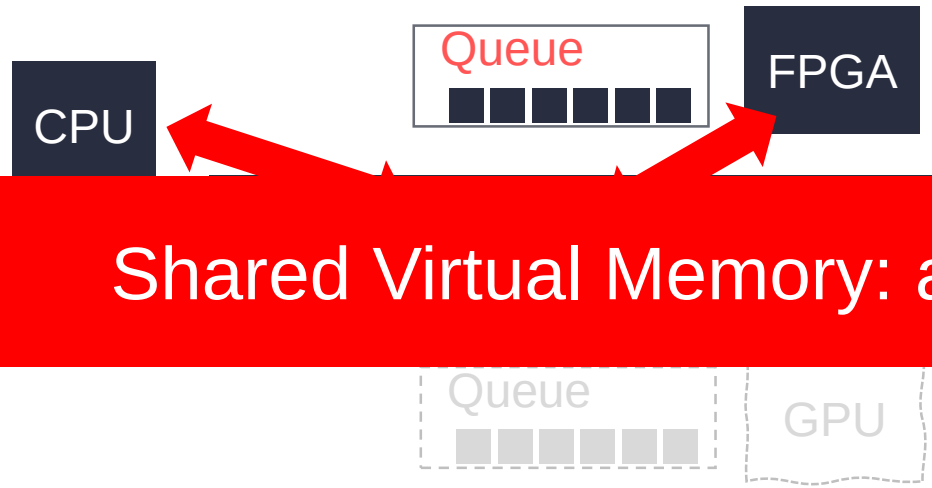



```
33  void * input_address, *output_address;
34  hsa_agent_iterate_regions(fpga_agent[0], get_karg_region, &karg_region);
35  // create a user mode queue on FPGA
36  status = hsa_queue_create(
37      fpga_agent[0],
38      256,
39      HSA_QUEUE_TYPE_SINGLE,
40      0,0,0,0,
41      &queue);
42
43  status = hsa_signal_create(1 /* Initial Value */ ,1, fpga_agent, &completion_signal);
44
45  /* Allocate FPGA-visible input buffer on the host */
46  hsa_memory_allocate(karg_region, 512, &input_address);
47  /* Allocate FPGA-visible output_buffer on the host */
48  hsa_memory_allocate(karg_region, 512, &output_address);
49
50   uint64_t index = hsa_queue_load_write_index_relaxed(queue);
51   const uint32_t mask = queue->size -1;
52   hsa_agent_dispatch_packet_t * packet =
53       &(((hsa_agent_dispatch_packet_t*)(queue->base_address))[index & mask]);
54
55   // set parameters in packet
56   packet->completion_signal = completion_signal;
57   packet->arg[0] = (uint64_t)input_address;
58   packet->arg[1] = (uint64_t)output_address;
59   packet->header = HSA_PACKET_TYPE_AGENT_DISPATCH << HSA_PACKET_HEADER_TYPE;
60   hsa_queue_store_write_index_relaxed(queue, index+1);
61
62   hsa_signal_wait_scacquire(completion_signal,
63                 HSA_SIGNAL_CONDITION_EQ, 0,
64                 2000,
65                 HSA_WAIT_STATE_ACTIVE);
66   printf("Success\n");
67  }
```


XILINX.

Demo 3: Kernel

Queues and Events using Shared Virtual Memory




```
33  void * input_address, *output_address;
34  hsa_agent_iterate_regions(fpga_agent[0], get_karg_region, &karg_region);
35  // create a user mode queue on FPGA
36  status = hsa_queue_create(
37      fpga_agent[0],
38      256,
39      HSA_QUEUE_TYPE_SINGLE,
40      0,0,0,0,
41      &queue);
42
43  status = hsa_signal_create(1 /* Initial Value */ ,1, fpga_agent, &completion_signal);
44
45  /* Allocate FPGA-visible input buffer on the host */
```

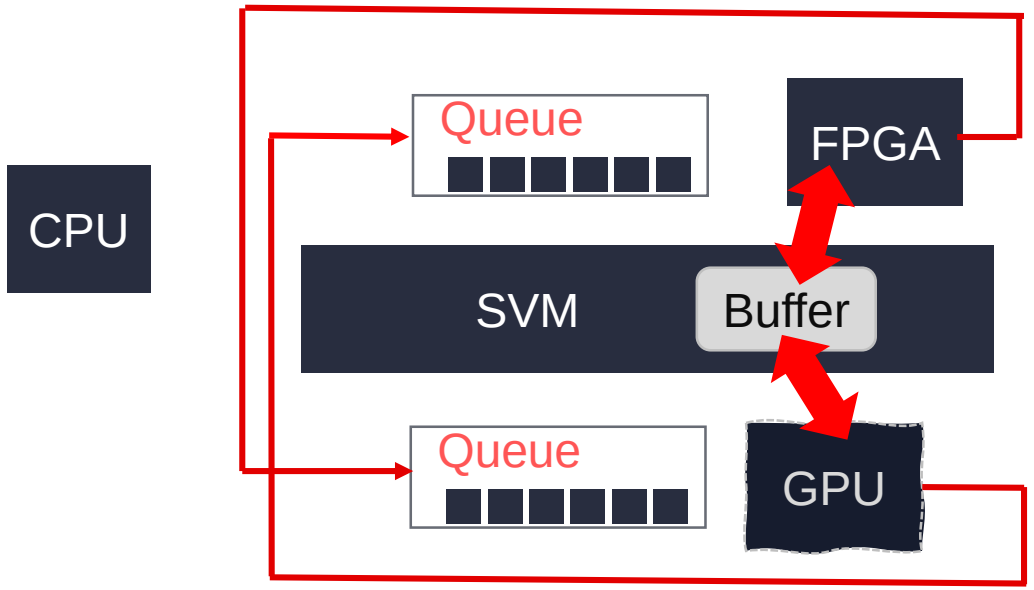
```
53      &(((hsa_agent_dispatch_packet_t*)(queue->base_address))[index & mask]);
54
55  // set parameters in packet
56  packet->completion_signal = completion_signal;
57  packet->arg[0] = (uint64_t)input_address;
58  packet->arg[1] = (uint64_t)output_address;
59  packet->header = HSA_PACKET_TYPE_AGENT_DISPATCH << HSA_PACKET_HEADER_TYPE;
60  hsa_queue_store_write_index_relaxed(queue, index+1);
61
62  hsa_signal_wait_scacquire(completion_signal,
63                HSA_SIGNAL_CONDITION_EQ, 0,
64                2000,
65                HSA_WAIT_STATE_ACTIVE);
66  printf("Success\n");
67  }
```

Shared Virtual Memory: access buffers from CPU, GPU, and FPGA

XILINX.

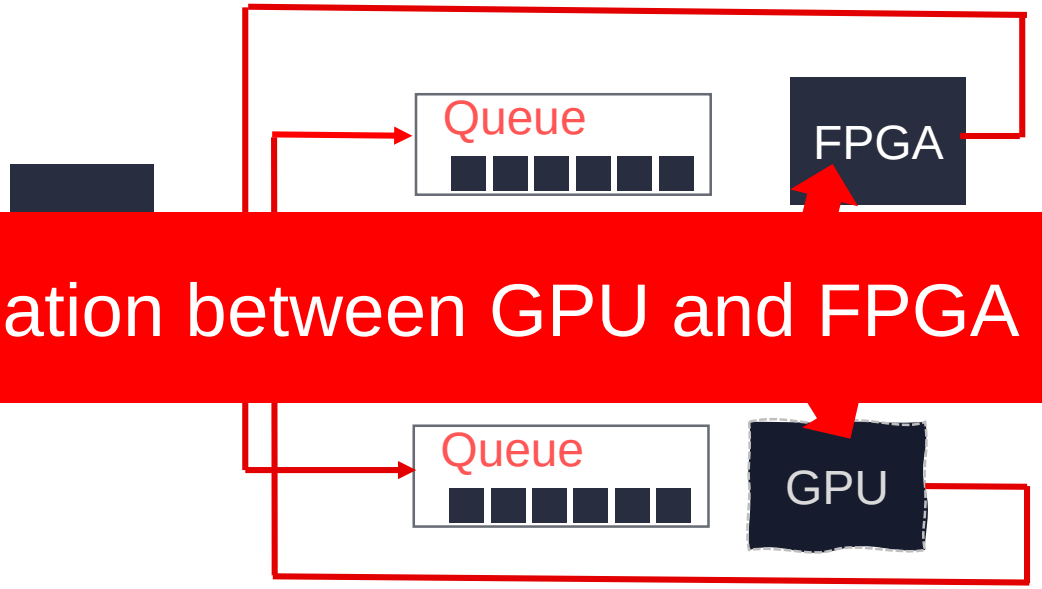
Demo 4: Interop with GPU

```
66  uint64_t fpga_qidx = hsa_queue_load_write_index_relaxed(fpga_queue);
67  const uint32_t fpga_mask = fpga_queue->size -1;
68  hsa_agent_dispatch_packet_t * fpga_packet =
69    &(((hsa_agent_dispatch_packet_t*)(fpga_queue->base_address))[fpga_qidx & fpga_mask]);
70
71  // set parameters in packet
72  fpga_packet->completion_signal = fpga_completion_signal;
73  fpga_packet->arg[0] = (uint64_t)input_address;
74  fpga_packet->arg[1] = (uint64_t)output_address;
75  fpga_packet->header = HSA_PACKET_TYPE_AGENT_DISPATCH << HSA_PACKET_HEADER_TYPE;
76  /* Dispatch Kernel to FPGA */
77  hsa_queue_store_write_index_relaxed(fpga_queue, ++fpga_qidx);
78
79  uint64_t gpu_qidx = hsa_queue_load_write_index_relaxed(gpu_queue);
80  const uint32_t gpu_mask = fpga_queue->size -1;
81  hsa_barrier_and_packet_t * barrier_packet =
82    &(((hsa_barrier_and_packet_t*)(gpu_queue->base_address))[gpu_qidx & gpu_mask]);
83
84  memset(barrier_packet, 0, sizeof(hsa_barrier_and_packet_t));
85  barrier_packet->dep_signal[0] = fpga_completion_signal;
86  barrier_packet->header = HSA_PACKET_TYPE_BARRIER_AND << HSA_PACKET_HEADER_TYPE;
87  /* Write Barrier packet into GPU queue */
88  hsa_queue_store_write_index_relaxed(gpu_queue, ++gpu_qidx);
89  hsa_kernel_dispatch_packet_t * gpu_packet =
90    &(((hsa_kernel_dispatch_packet_t*)(gpu_queue->base_address))[gpu_qidx & gpu_mask]);
91  /* Sets up kernel dispatch and makes packet visible */
92  populate_gpu_kernel_packet(gpu_packet, &gpu_completion_signal);
93  /* Write GPU kernel packet into GPU queue, will only start when FPGA task is complete */
94  hsa_queue_store_write_index_relaxed(gpu_queue, ++gpu_qidx);
95  hsa_signal_wait_scacquire(gpu_completion_signal,
96                HSA_SIGNAL_CONDITION_EQ, 0, 2000,
97                HSA_WAIT_STATE_ACTIVE);
98  printf("Success\n");
99  }
```



Demo 4: Interop with GPU

```
66  uint64_t fpga_qidx = hsa_queue_load_write_index_relaxed(fpga_queue);
67  const uint32_t fpga_mask = fpga_queue->size -1;
68  hsa_agent_dispatch_packet_t * fpga_packet =
69    &(((hsa_agent_dispatch_packet_t*)(fpga_queue->base_address))[fpga_qidx & fpga_mask]);
70
71  // set parameters in packet
72  fpga_packet->completion_signal = fpga_completion_signal;
73  fpga_packet->arg[0] = (uint64_t)input_address;
74  fpga_packet->arg[1] = (uint64_t)output_address;
75  fpga_packet->header = HSA_PACKET_TYPE_AGENT_DISPATCH << HSA_PACKET_HEADER_TYPE;
76  /* Dispatch Kernel to FPGA */
83
84  memset(barrier_packet, 0, sizeof(hsa_barrier_and_packet_t));
85  barrier_packet->dep_signal[0] = fpga_completion_signal;
86  barrier_packet->header = HSA_PACKET_TYPE_BARRIER_AND << HSA_PACKET_HEADER_TYPE;
87  /* Write Barrier packet into GPU queue */
88  hsa_queue_store_write_index_relaxed(gpu_queue, ++gpu_qidx);
89  hsa_kernel_dispatch_packet_t * gpu_packet =
90    &(((hsa_kernel_dispatch_packet_t*)(gpu_queue->base_address))[gpu_qidx & gpu_mask]);
91  /* Sets up kernel dispatch and makes packet visible */
92  populate_gpu_kernel_packet(gpu_packet, &gpu_completion_signal);
93  /* Write GPU kernel packet into GPU queue, will only start when FPGA task is complete */
94  hsa_queue_store_write_index_relaxed(gpu_queue, ++gpu_qidx);
95  hsa_signal_wait_scacquire(gpu_completion_signal,
96                HSA_SIGNAL_CONDITION_EQ, 0, 2000,
97                HSA_WAIT_STATE_ACTIVE);
98  printf("Success\n");
99  }
```



Peer-to-peer data sharing and synchronization between GPU and FPGA

XILINX.

Xilinx Mission

Building the Adaptable, Intelligent World