Parallel Computing with GPUs

CUDA Streams
Part 2 - CUDA Streams

The
University
Of
Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

❑CUDA Streams

  ❑Demonstrate how to make asynchronous memory calls

  ❑Demonstrate how have *copy* and *compute* concurrency

  ❑Give examples of the issue ordering implications of stream scheduling and concurrency

Opportunities for Device Concurrency

- Most CUDA Devices have an asynchronous Kernel execution and Copy Engine
  - Allows data to be moved at the same time as execution
  - Most device have dual copy engines
    - PCIe upstream (D2H)
    - PCIe downstream (H2D)
  - Ideally we should hide data movement with execution
  - Check your device capability: deviceQuery example “`Concurrent copy and kernel execution:`
- All modern GPU devices are able to execute kernels simultaneously
  - Allows task parallelism on GPU
  - Each kernel represents a different task
  - Very useful for smaller problem sizes

Streams

- ❑CUDA Streams allow operations to be queued for the GPU device
  - ❑All calls are asynchronous by default
  - ❑The host retains control
  - ❑Device takes work from the streams when it is able to do so
- ❑Operations in a stream are ordered and can not overlap (FIFO)
- ❑Operations in different streams can overlap

```
// create a handle for the stream
cudaStream_t stream;
//create the stream
cudaStreamCreate(&stream);

//do some work in the stream ...

//destroy the stream (blocks host until stream is complete)
cudaStreamDestroy(stream);
```

Work Assignment for Streams

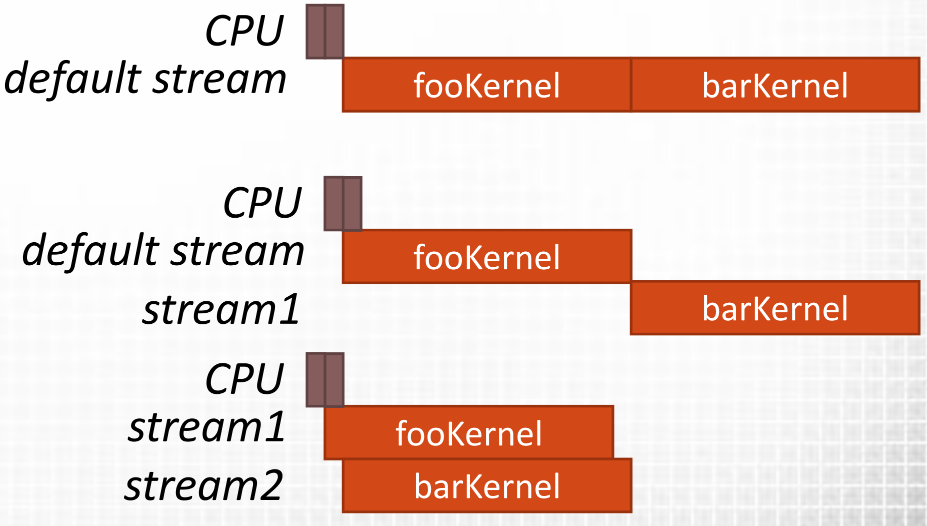
```
//execute kernel on device in specified stream
fooKernel<<<blocks, threads, 0, stream>>>();
```

- ❑Kernel Execution is assigned to streams as 4th parameter of kernel launch
- ❑Care must be taken with the default stream
  - ❑Only stream which is synchronous with others!

```
fooKernel<<<blocks, threads, 0>>>();
barKernel<<<blocks, threads, 0>>>();
```

```
fooKernel<<<blocks, threads, 0>>>();
barKernel<<<blocks, threads, 0, stream>>>();
```

```
fooKernel<<<blocks, threads, 0, stream1>>>();
barKernel<<<blocks, threads, 0, stream2>>>();
```



Asynchronous Memory

- CUDA is able to asynchronously copy data
    - Only if it is Pinned (Page-locked) memory
- Paged Memory
    - Allocated using `malloc(…)` on host and released using `free(…)`
- Pinned Memory
    - Can not be swapped (paged) out by the OS
    - Has higher overhead for allocation
    - Can reach higher bandwidths for large transfers
    - Allocated using `cudaMallocHost(…)` and released using `cudaFreeHost(…)`
    - Can also pin non pinned memory using `cudaHostRegister(…)` / `cudaHostUnregister(…)`
        - Very slow

Concurrent Copies in Streams

- ❑Memory copies can be replaced with `cudaMemcpyAsync()`
  - ❑Requires an extra argument (a stream)
  - ❑Places transfer into the stream and returns control to host
  - ❑Conditions of use
    - ❑Must be pinned memory
    - ❑Must be in the non-default stream

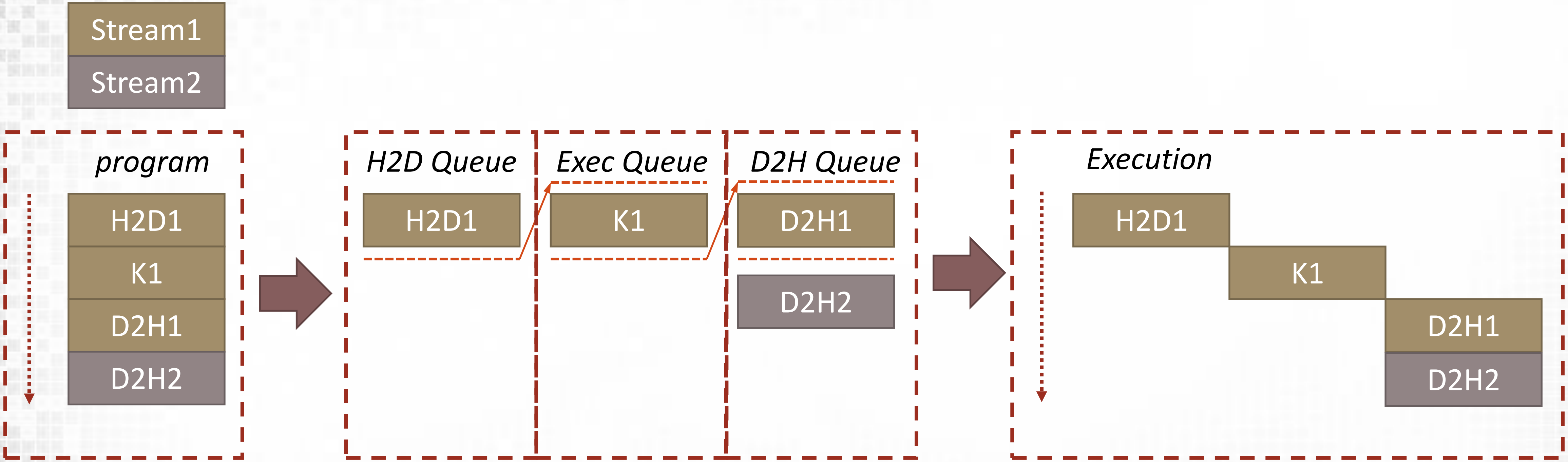
```
int *h_A, *d_A;
cudaStream_t stream1;

cudaStreamCreate(&stream1);
cudaMallocHost(&h_A, SIZE);
cudaMalloc(&d_A, SIZE);
initialiseA(h_A);

cudaMemcpyAsync(d_A, h_A, SIZE, cudaMemcpyHostToDevice, stream1);

//work in other streams ...

cudaStreamDestroy(stream1);
```

Stream Scheduling

- ❑CUDA operations dispatched to hardware in sequence that they were issued
  - ❑Hence issue order is important (FIFO)
- ❑Kernel and Copy Engine (x2) have different queues

- ❑Operations are de-queued if
  1. Preceding call in the same stream have completed
  2. Preceding calls in the same queue have been dispatched, and
  3. Resources are available
     - ❑i.e. kernels can be concurrently executed if in different streams
- ❑Blocking operations (e.g. `cudaMemcpy` will block all streams)

Issue Ordering



Stream1

Stream2

program

H2D1

K1

D2H1

D2H2

H2D Queue

H2D1

Exec Queue

K1

D2H Queue

D2H1

D2H2

Execution

H2D1

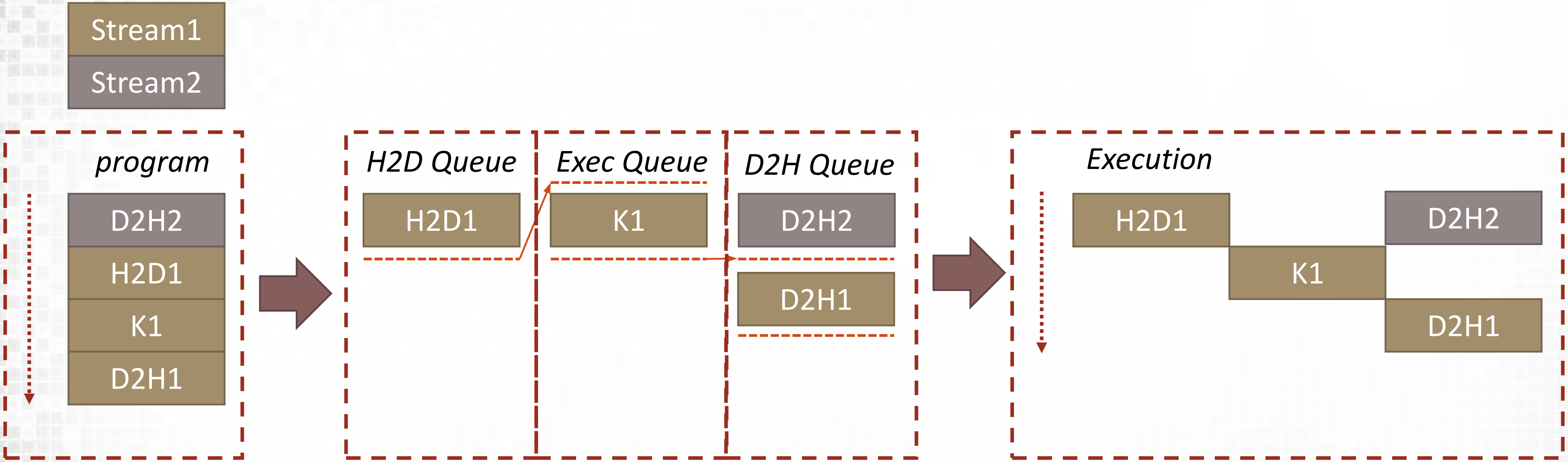
K1

D2H1

D2H2

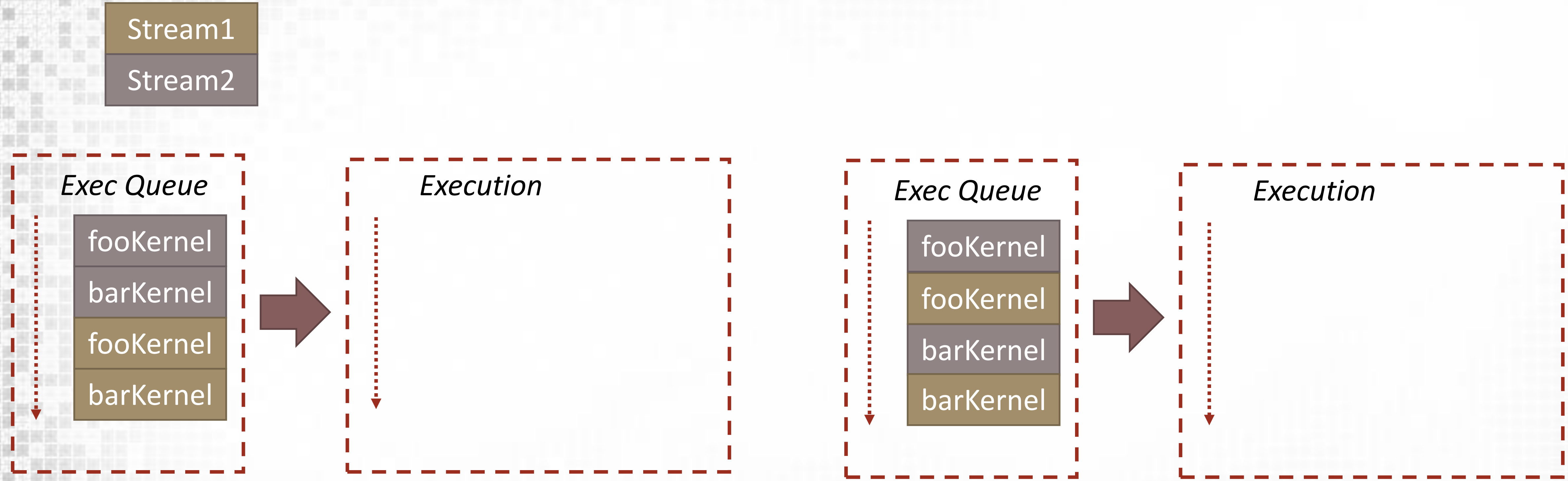
- No Concurrency of D2H2
- Blocked by D2H1
  - Issued first (FIFO)

Issue Ordering



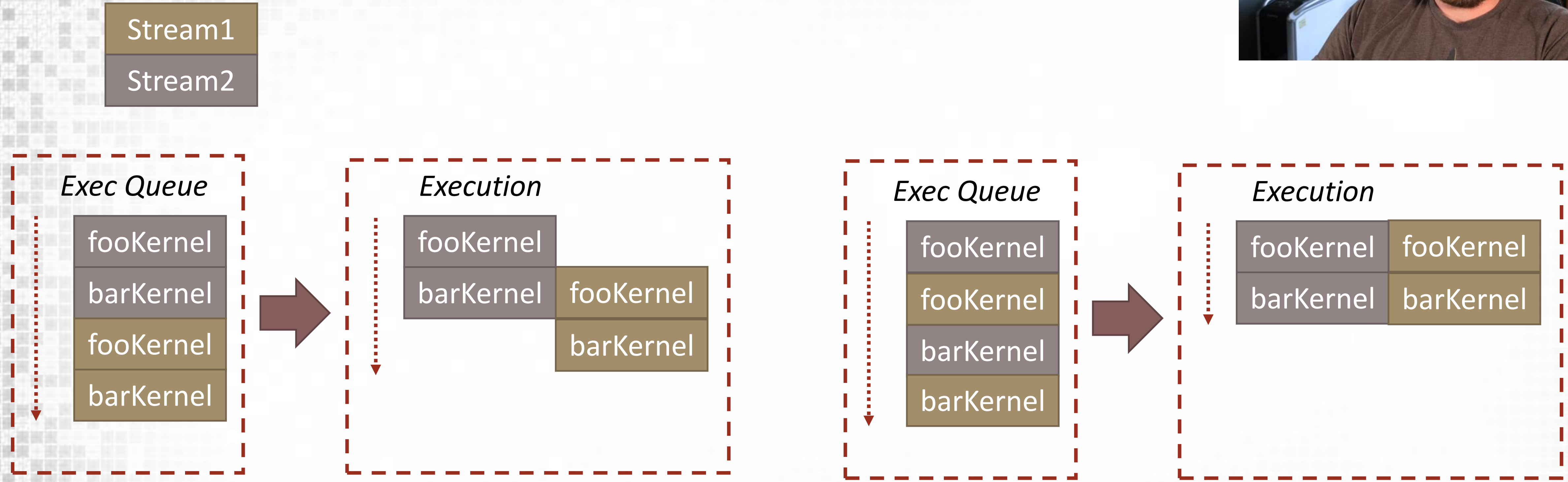
❑Concurrency of D2H2 and H2D1

Issue Ordering (Kernel Execution)



❑Which has best Asynchronous execution?

Issue Ordering (Kernel Execution)



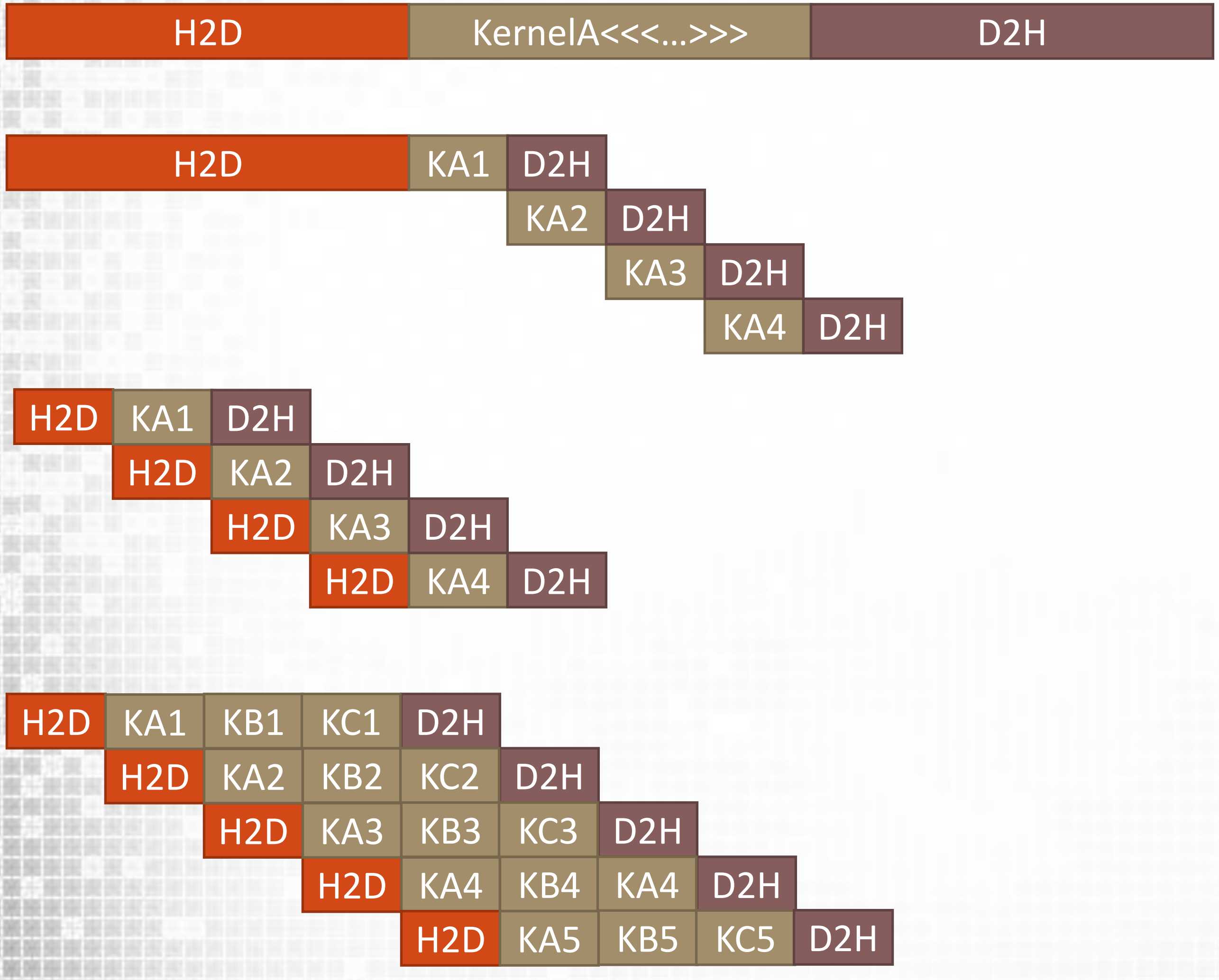
❑barKernel can't be removed from queue until fooKernel has completed

❑Blocks fooKernel

❑Both fooKernels can be concurrently executed

❑Both barKernels concurrently executed

Levels of Concurrency



Fully Synchronous (Serial Execution)

2-way Concurrency

- ❑ H2D and D2H not concurrent

3-way Concurrency

- ❑ Both Copy Engines active
- ❑ Execution Engine active
  - ❑ May or may not be fully utilised

5-way Concurrency

- ❑ Both Copy Engines active
- ❑ Execution Engine active
  - ❑ Higher independent workload
  - ❑ Better chance of 100% utilisation
- ❑ What about Host?

The University Of Sheffield.

Summary

- CUDA Streams
  - Demonstrate how to make asynchronous memory calls
  - Demonstrate how have *copy* and *compute* concurrency
  - Give examples of the issue ordering implications of stream scheduling and concurrency

- Next Lecture: Synchronisation