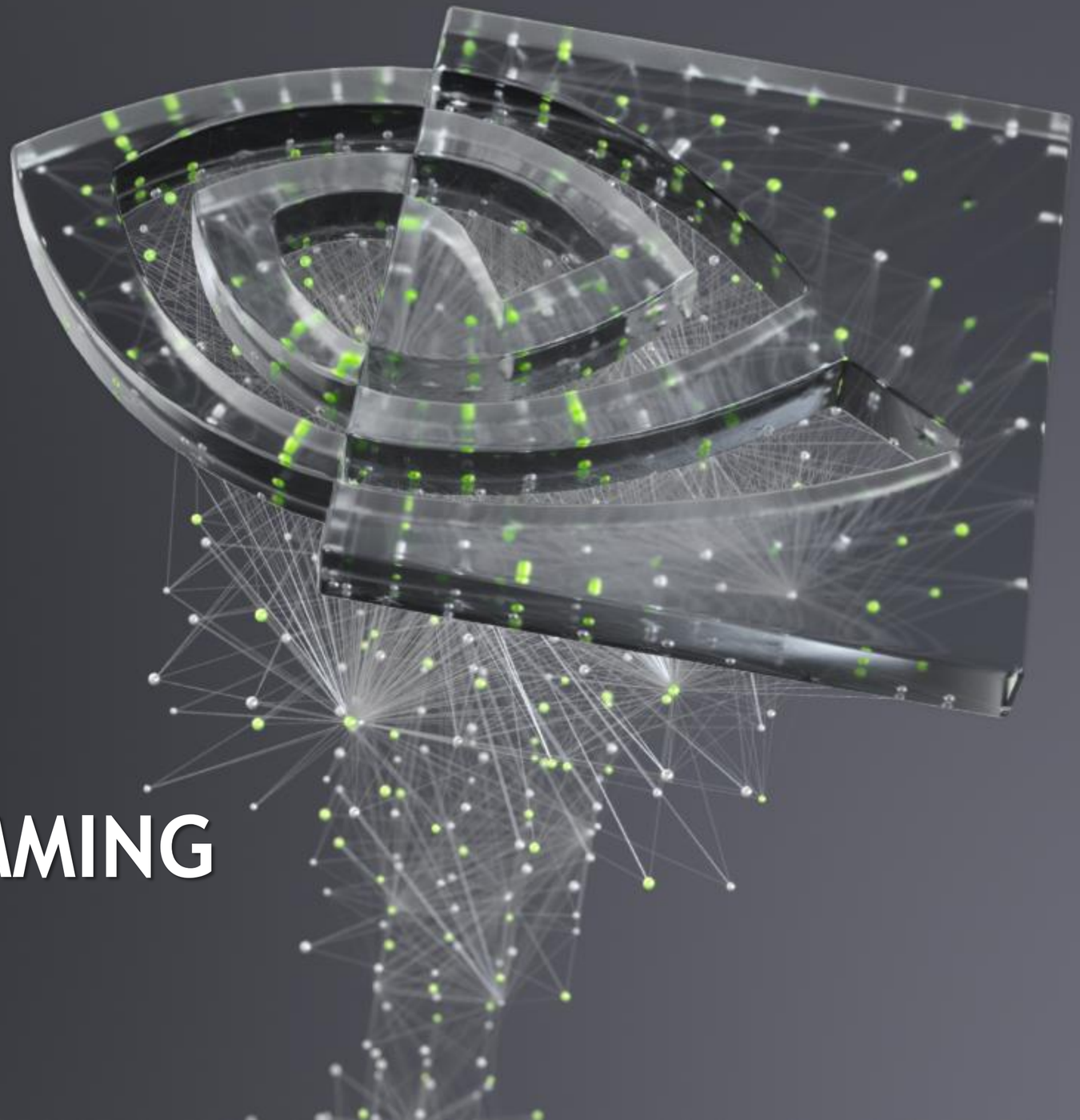


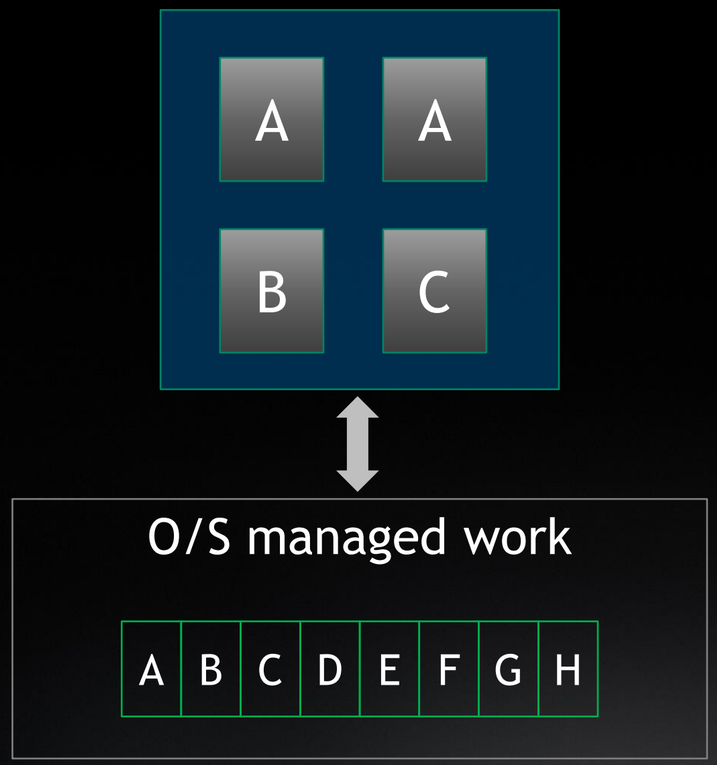
NVIDIA

LAZY GPU PROGRAMMING

Stephen Jones, 10th March 2021

THROUGHPUT MACHINES vs. LATENCY MACHINES

(vs. general purpose machines)



General Purpose System

Primary target: **guaranteed concurrency**

Threads time multiplexed onto cores

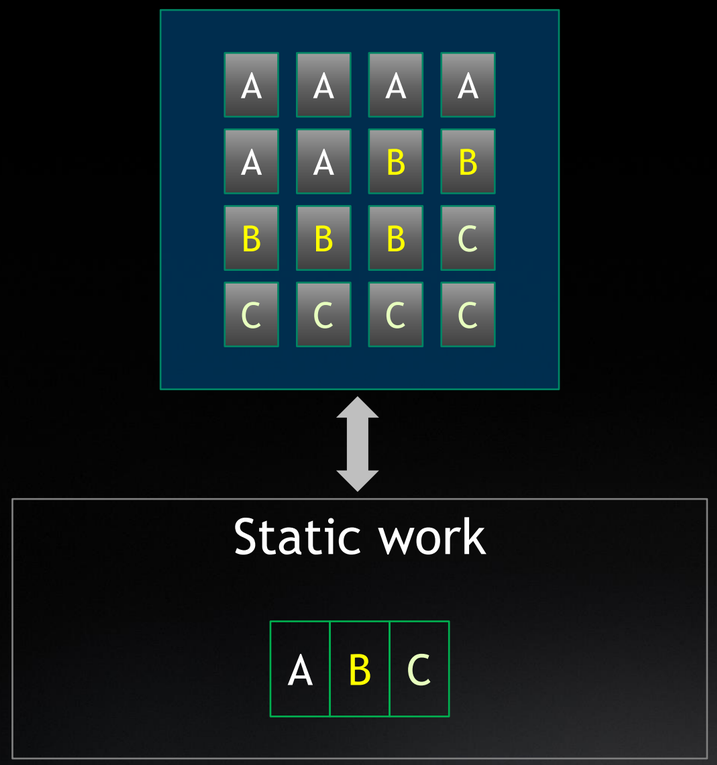
Pre-emptive task switching (arbitrary thread life)

Full concurrency (to limit of memory)

Dynamic scheduling under SW control

THROUGHPUT MACHINES vs. LATENCY MACHINES

(vs. general purpose machines)



Latency System

Primary target: **minimise response time**

Threads statically assigned to resources

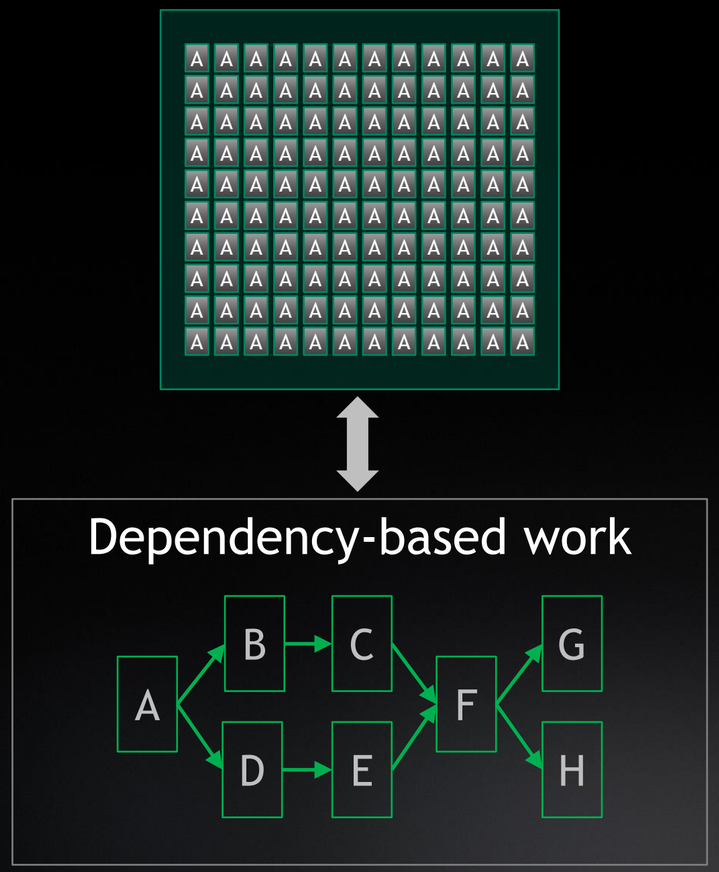
Tasks must run to completion, can block/spin

Concurrency limited to occupancy

SW managed resource allocation

THROUGHPUT MACHINES vs. LATENCY MACHINES

(vs. general purpose machines)



Throughput System

Primary target: **end-to-end runtime**

Threads oversubscribed by design

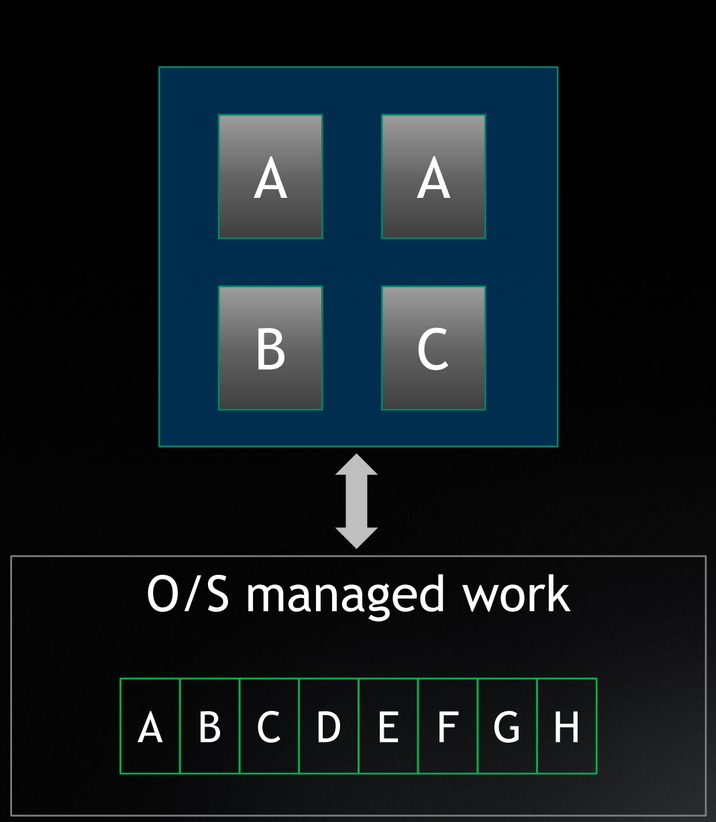
Tasks may never block – forward progress required

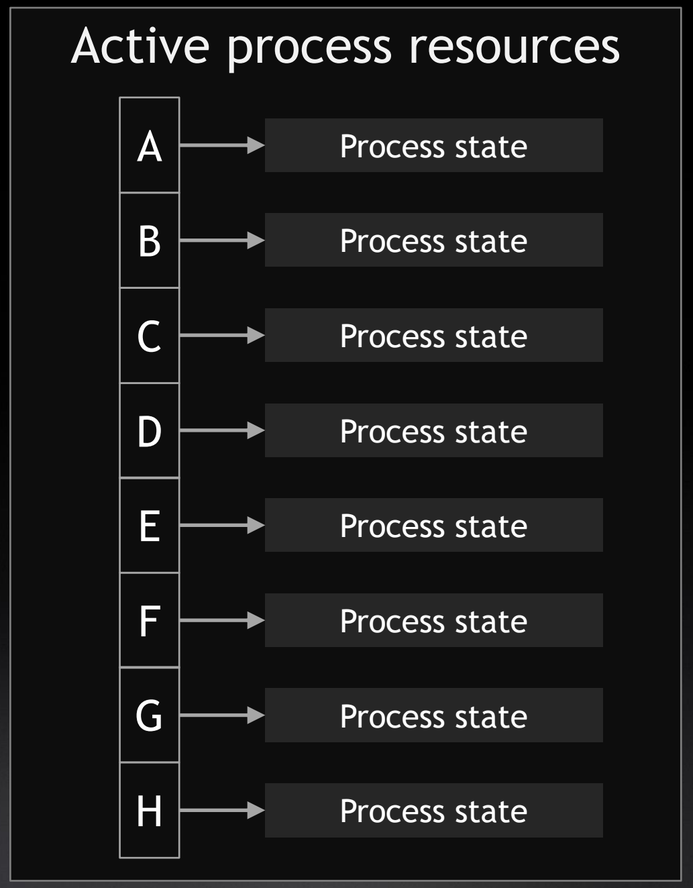
No concurrency guarantees

Work ordered by dependency task graph

CONCURRENCY vs. PARALLELISM

Concurrency means **eventual** forward progress for all [concurrent] work





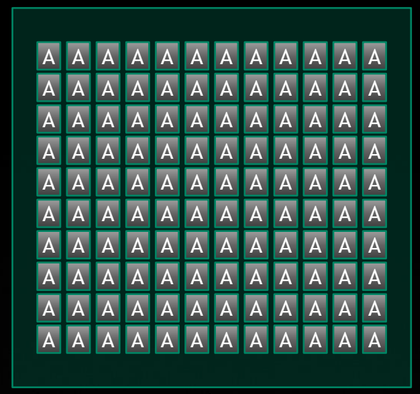
Process state

- Primarily stack space
- Size is unbounded
- Thread count unbounded
- Memory use unbounded

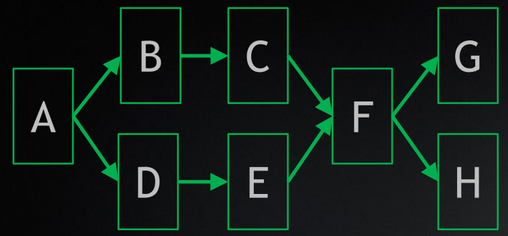
Entire state for **all** processes must be live to ensure concurrency

CONCURRENCY vs. PARALLELISM

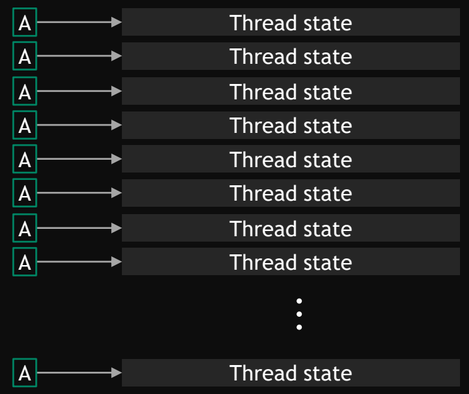
Parallelism means **simultaneous** forward progress for all [parallel] work



Dependency-based work



Active thread resources



Only live threads have state

Total threads >> live threads

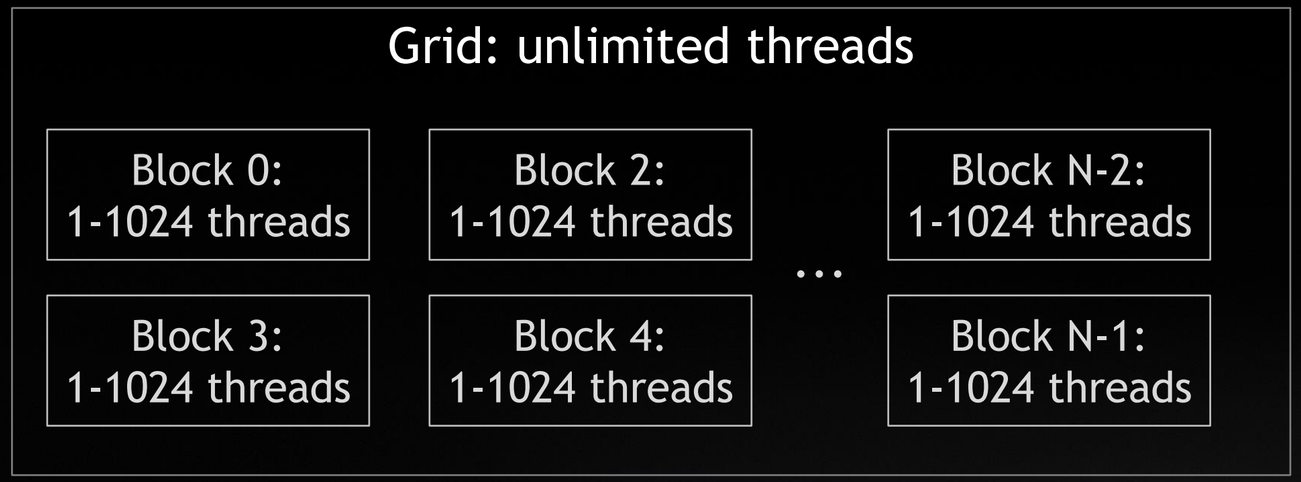
Thread state

- A100: around 3k/thread
- 221,184 threads on A100
- No ordering guarantees
- Hence no concurrency

Threads are never concurrent, so **must not** wait, block or synchronize

CUDA'S HIERARCHICAL PARALLELISM

Mix of concurrency and parallelism



CUDA's parallelism model

Threads within the same block are guaranteed co-scheduled

Can wait & synchronize **only** with other threads in the same block

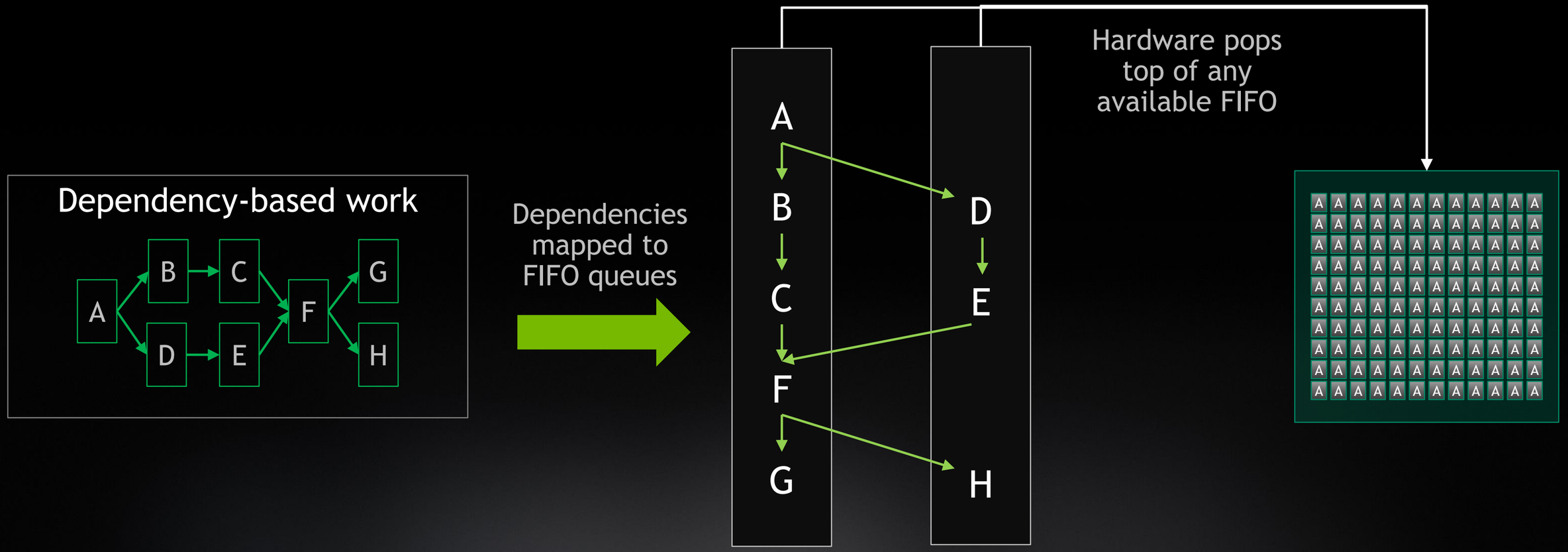
All blocks are the same size

Blocks may run in any order, including sequentially

Blocks may **never** synchronize with each other

GPU FIFO-BASED SCHEDULING

Hardware drains multiple (<=32) FIFO queues of work

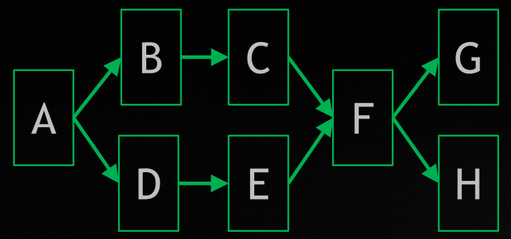


CUDA STREAMS

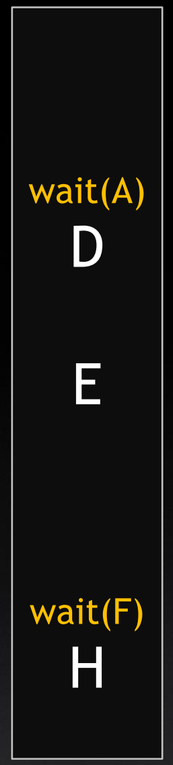
CUDA maps software “stream” to hardware FIFO

Streams have **implicit** submission-order dependencies

Dependency-based work

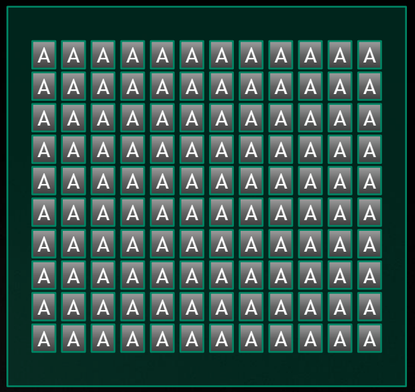


Dependencies **expressed** as CUDA streams



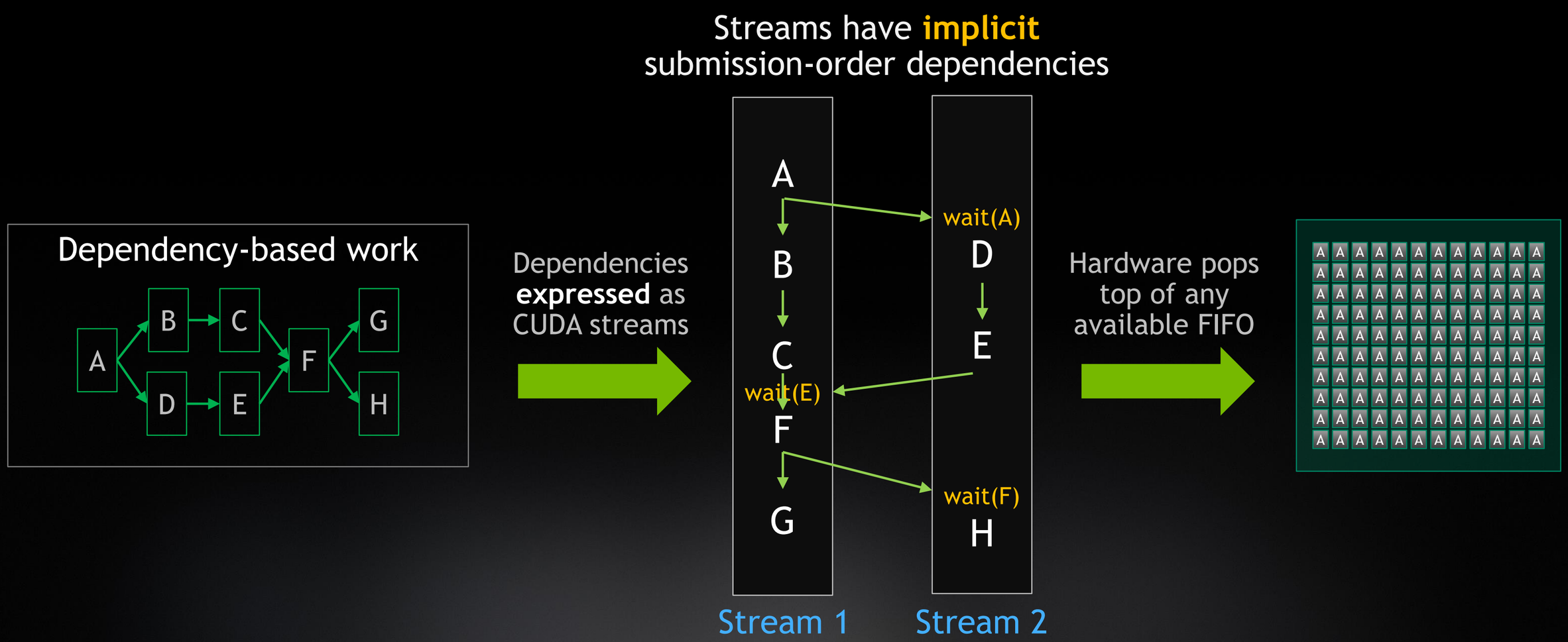
Stream 1 Stream 2

Hardware pops top of any available FIFO



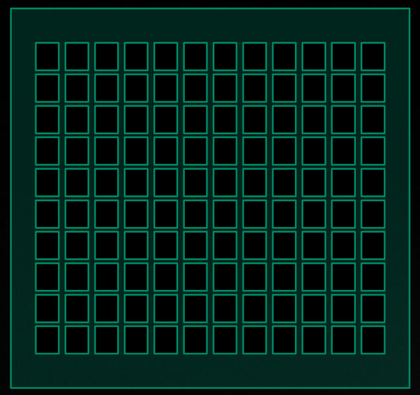
CUDA STREAMS

CUDA maps software “stream” to hardware FIFO



SYNCHRONOUS LAUNCHES ARE INEFFICIENT

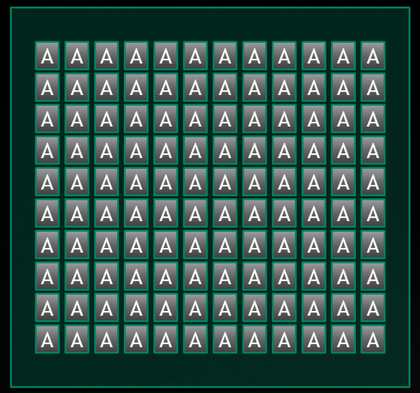
The GPU depends on a constant flow of work for efficiency

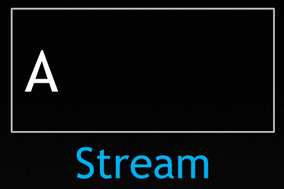



```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

The GPU depends on a constant flow of work for efficiency

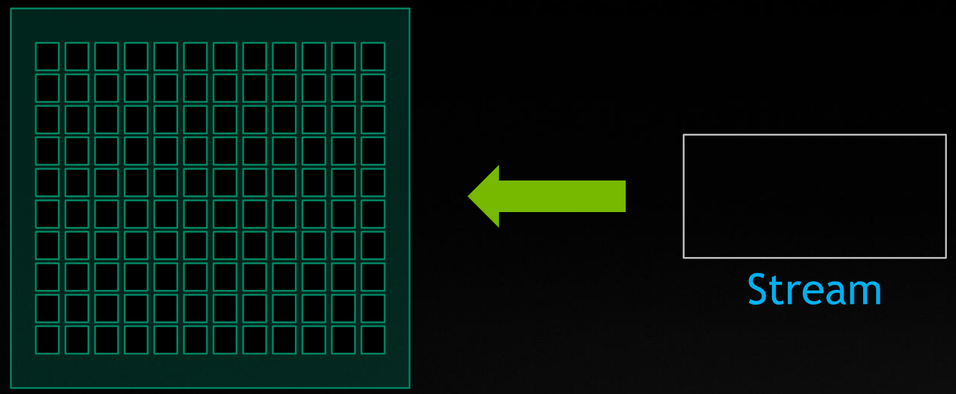





```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

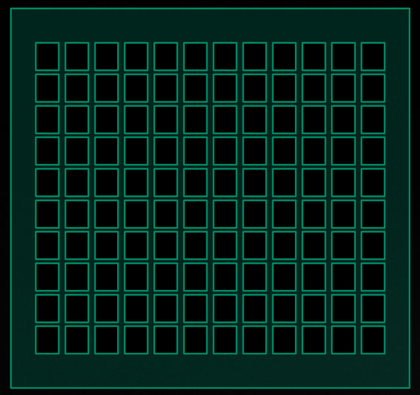
The GPU depends on a constant flow of work for efficiency




```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

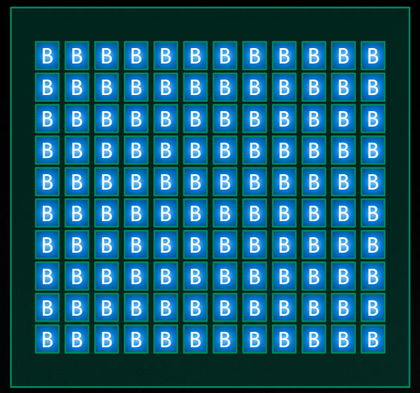
The GPU depends on a constant flow of work for efficiency




```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

The GPU depends on a constant flow of work for efficiency



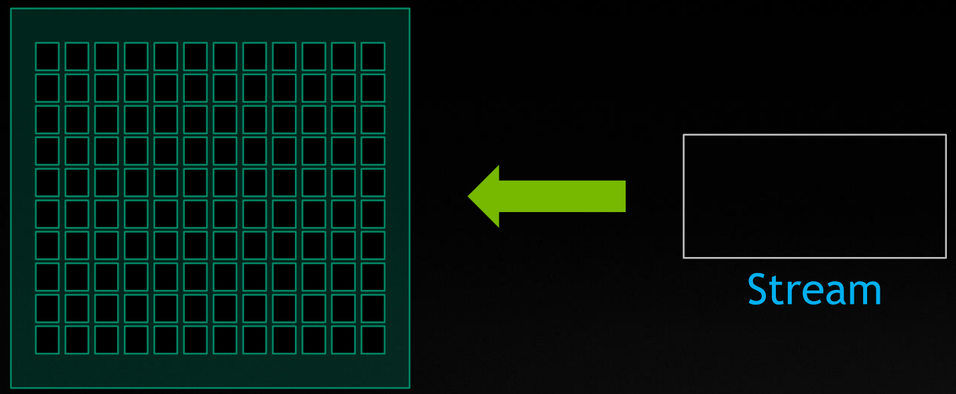
B

Stream

```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

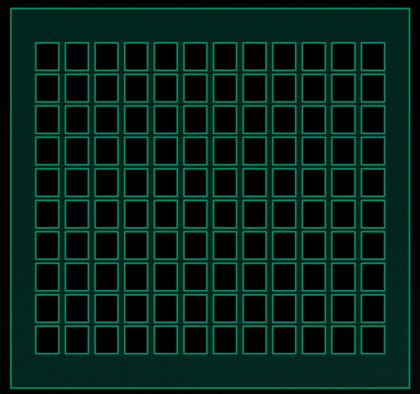
The GPU depends on a constant flow of work for efficiency




```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

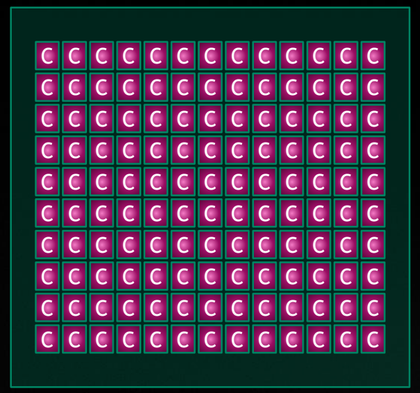
The GPU depends on a constant flow of work for efficiency




```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

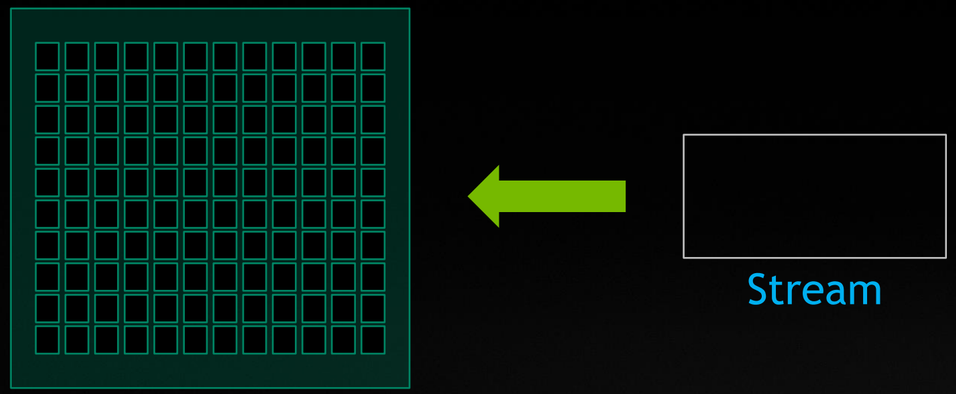
The GPU depends on a constant flow of work for efficiency




```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

SYNCHRONOUS LAUNCHES ARE INEFFICIENT

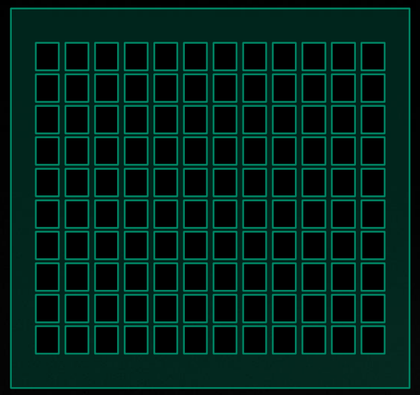
The GPU depends on a constant flow of work for efficiency




```
Launch(A);
Launch(B);
Launch(C);
Launch(D);
Launch(E);
```

ASYNCHRONOUS EXECUTION: KEEPING THE GPU BUSY

The GPU depends on a constant flow of work for efficiency

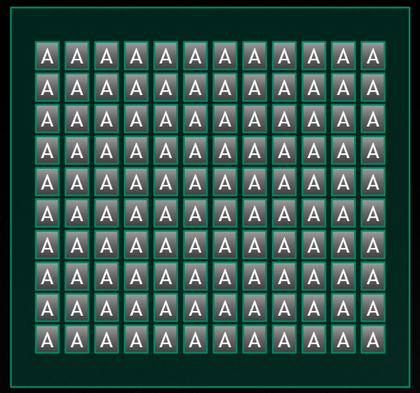


stream

```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

ASYNCHRONOUS EXECUTION: KEEPING THE GPU BUSY

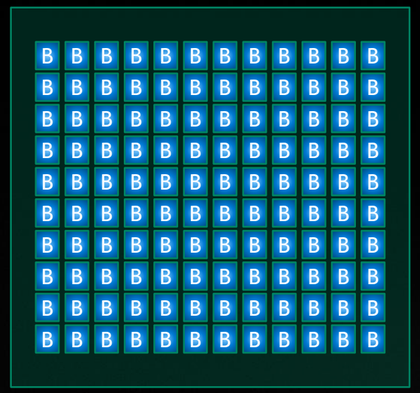
The GPU depends on a constant flow of work for efficiency




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

ASYNCHRONOUS EXECUTION: KEEPING THE GPU BUSY

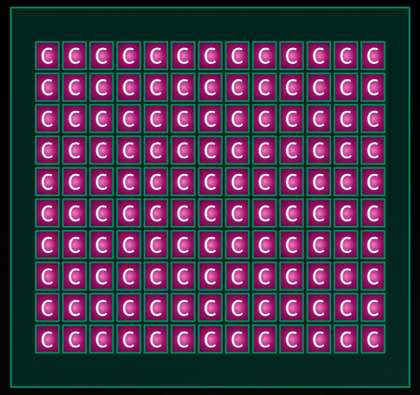
The GPU depends on a constant flow of work for efficiency




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

ASYNCHRONOUS EXECUTION: KEEPING THE GPU BUSY

The GPU depends on a constant flow of work for efficiency



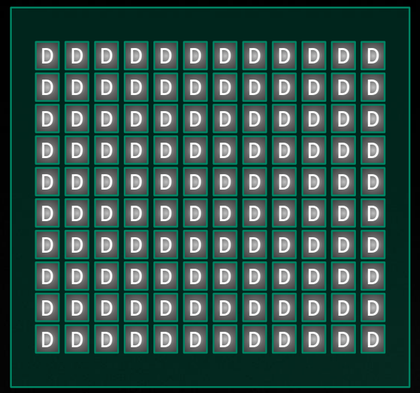
C D E

stream

```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

ASYNCHRONOUS EXECUTION: KEEPING THE GPU BUSY

The GPU depends on a constant flow of work for efficiency

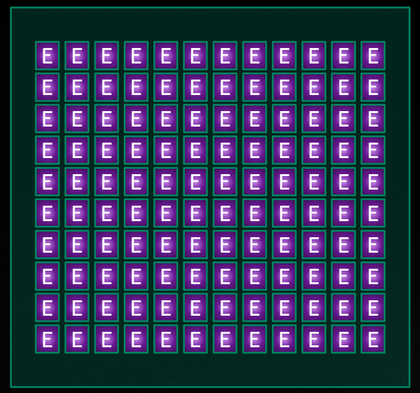


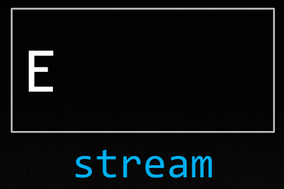
stream

```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

ASYNCHRONOUS EXECUTION: KEEPING THE GPU BUSY

The GPU depends on a constant flow of work for efficiency

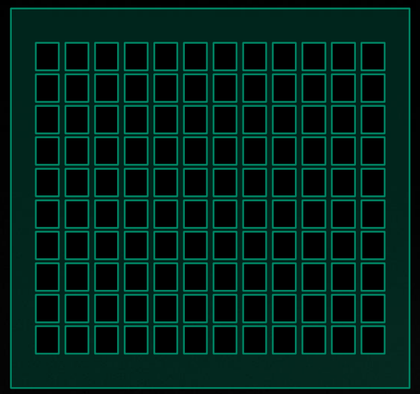





```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

ASYNCHRONOUS EXECUTION: KEEPING THE GPU BUSY

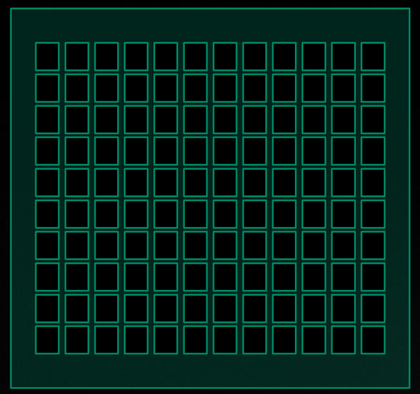
The GPU depends on a constant flow of work for efficiency




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

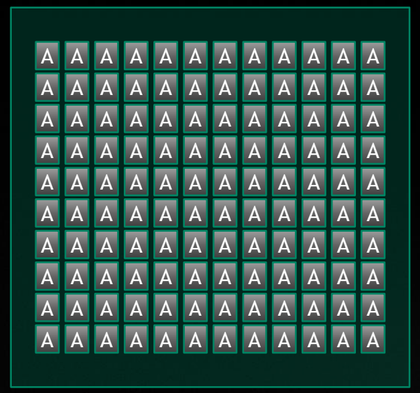
CPU-side operations will interrupt the GPU workflow




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

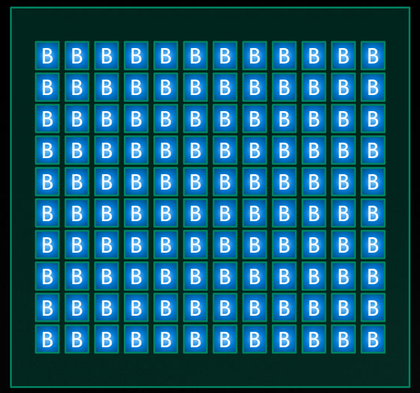
CPU-side operations will interrupt the GPU workflow




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

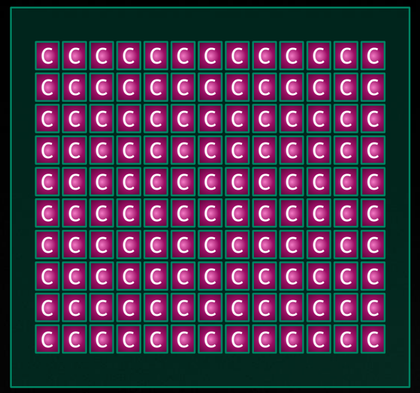
CPU-side operations will interrupt the GPU workflow




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

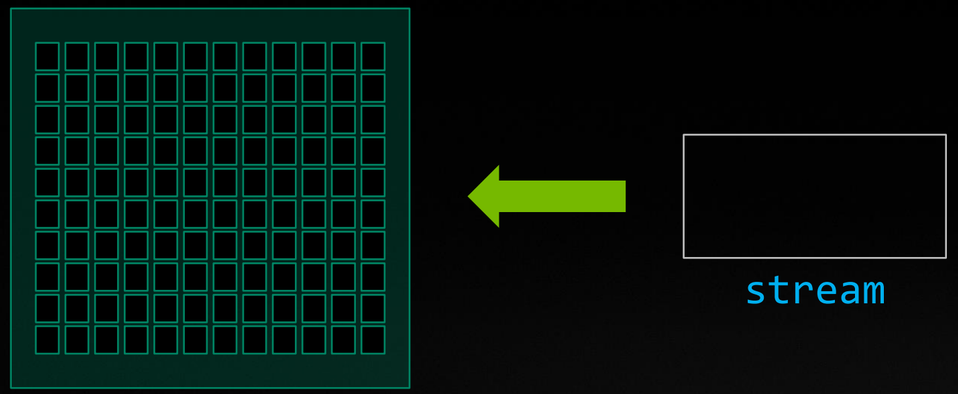
CPU-side operations will interrupt the GPU workflow




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

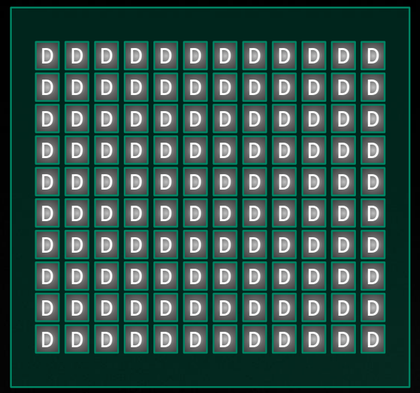
CPU-side operations will interrupt the GPU workflow




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

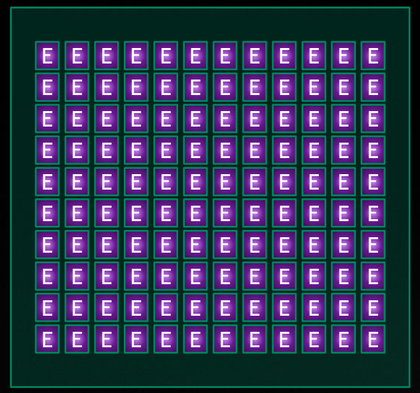
CPU-side operations will interrupt the GPU workflow

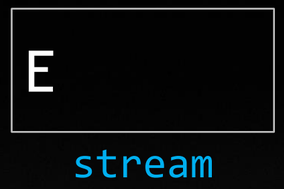



```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

CPU-side operations will interrupt the GPU workflow

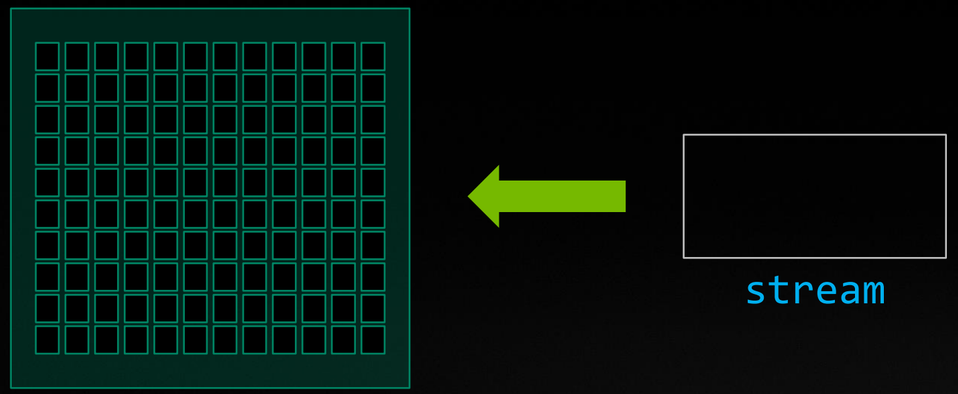





```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

CPU-DEPENDENT WORK

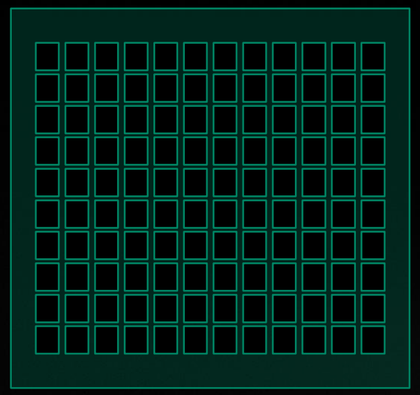
CPU-side operations will interrupt the GPU workflow




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
cudaStreamSynchronize();
MPI_Isend( data_from_C );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

ASYNCHRONOUS CPU WORK

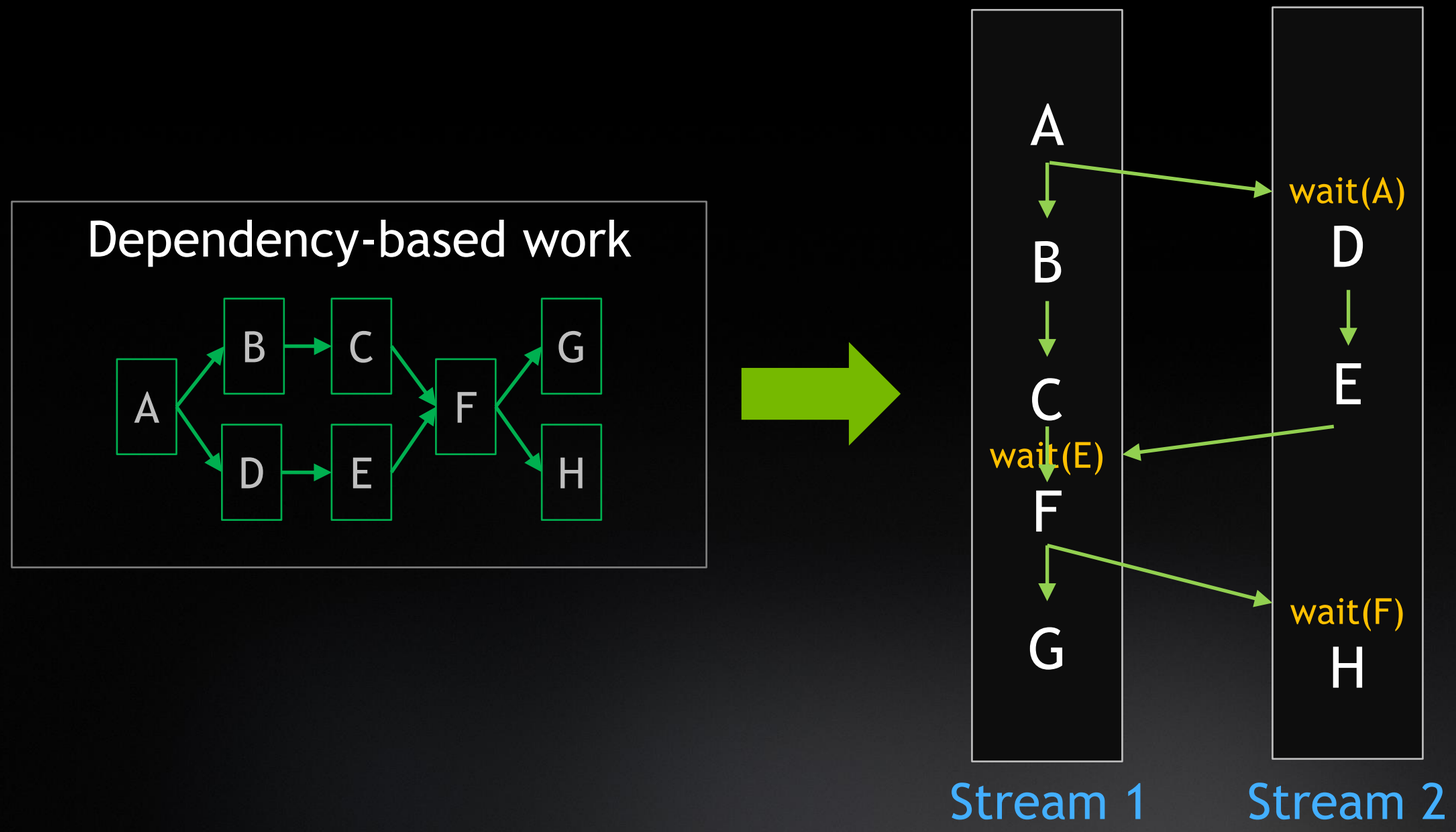
Trigger CPU operation without blocking GPU flow



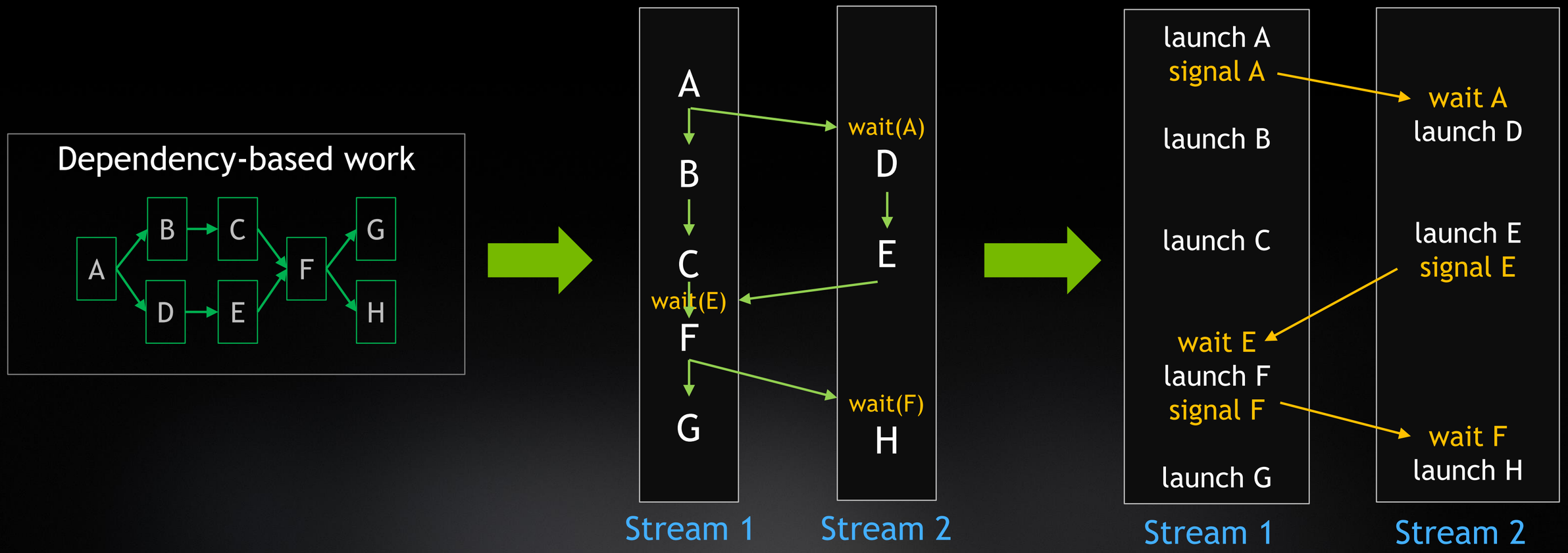
stream

```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
MPI_Isend_on_stream( data_from_C, stream );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

HOW STREAMS REALLY WORK

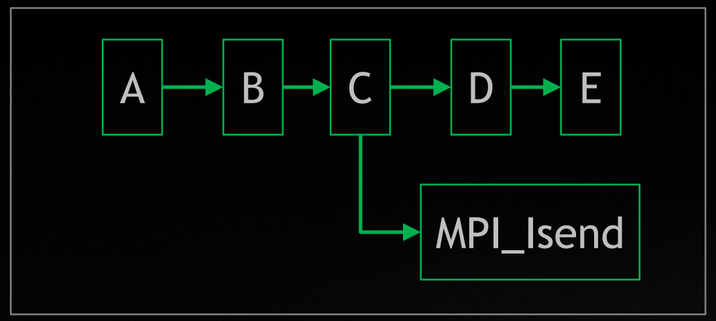


HOW STREAMS REALLY WORK



TRIGGERING EXTERNAL WORK

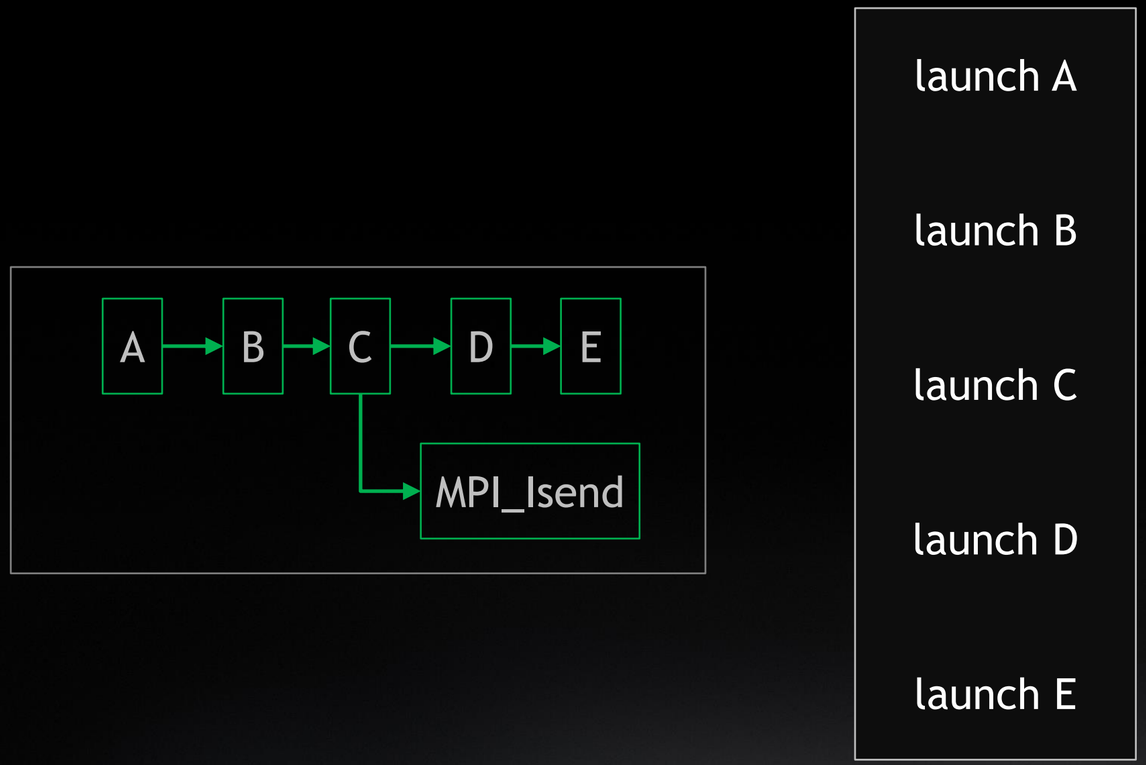
Triggering a CPU operation without blocking GPU flow




```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
MPI_Isend_on_stream( data_from_C, stream );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

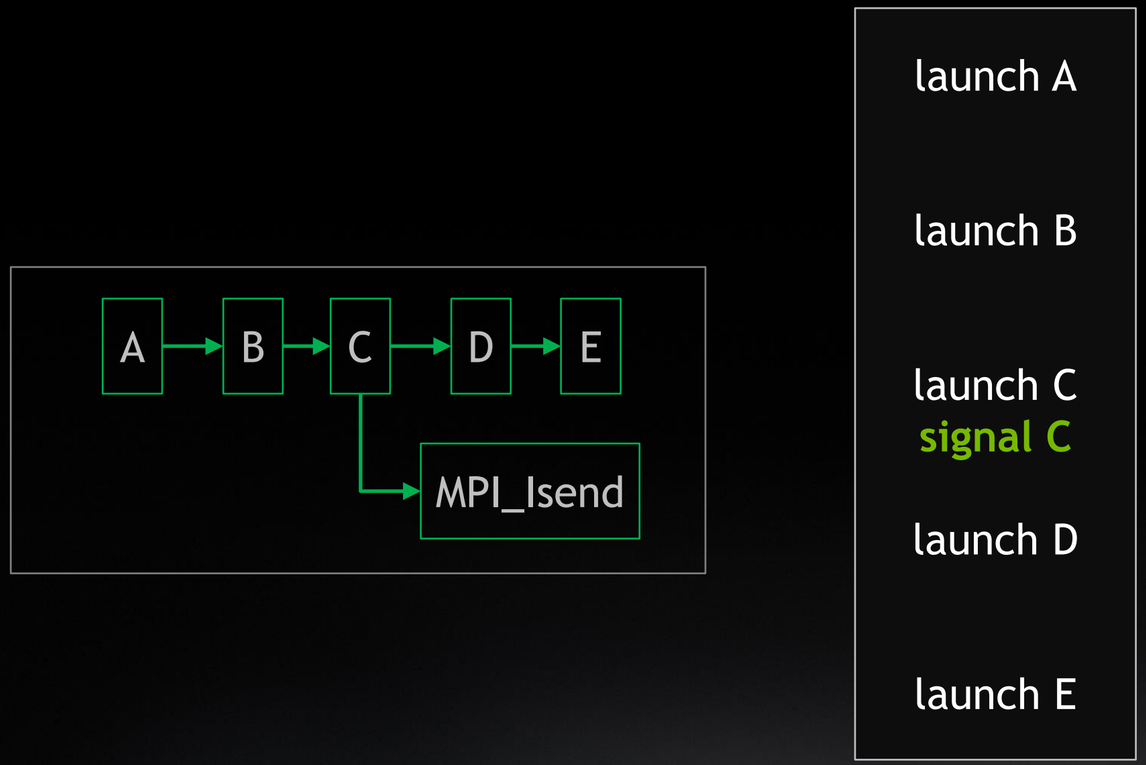
TRIGGERING EXTERNAL WORK

Triggering a CPU operation without blocking GPU flow



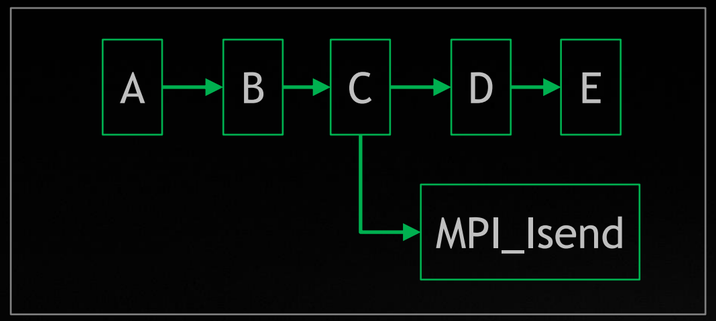
TRIGGERING EXTERNAL WORK

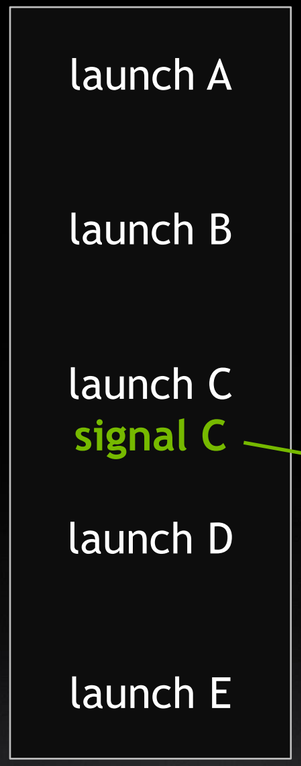
Triggering a CPU operation without blocking GPU flow



TRIGGERING EXTERNAL WORK

Triggering a CPU operation without blocking GPU flow

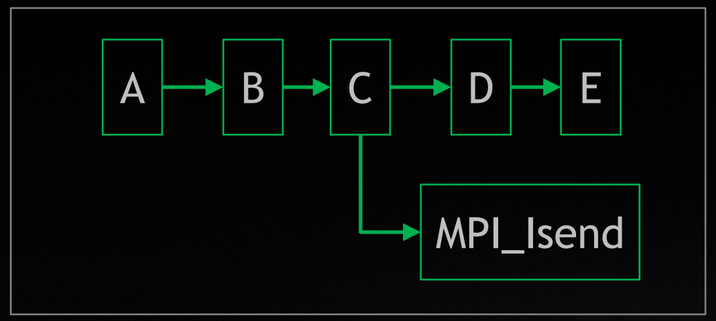


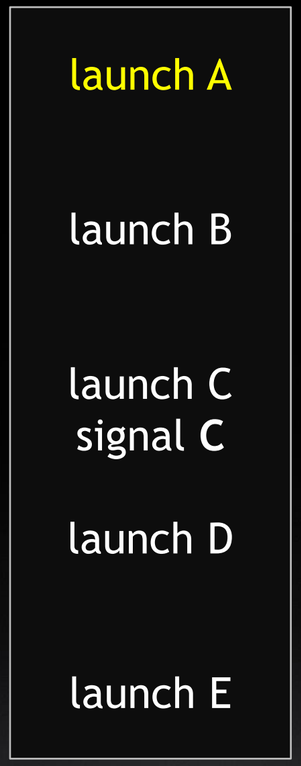



```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
MPI_Isend_on_stream(ptr, stream) {
     cuda_allocate_signal_mem(C);
     cuda_insert_signal(C, stream);
     wait(C);
     MPI_Isend(ptr);
}
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

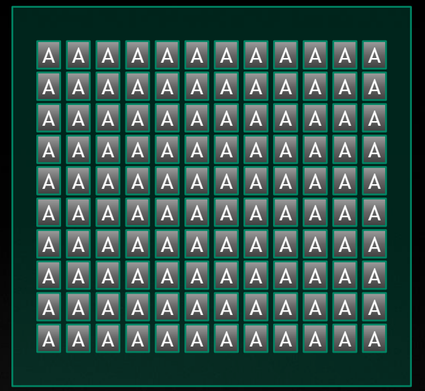
TRIGGERING EXTERNAL WORK

Triggering a CPU operation without blocking GPU flow



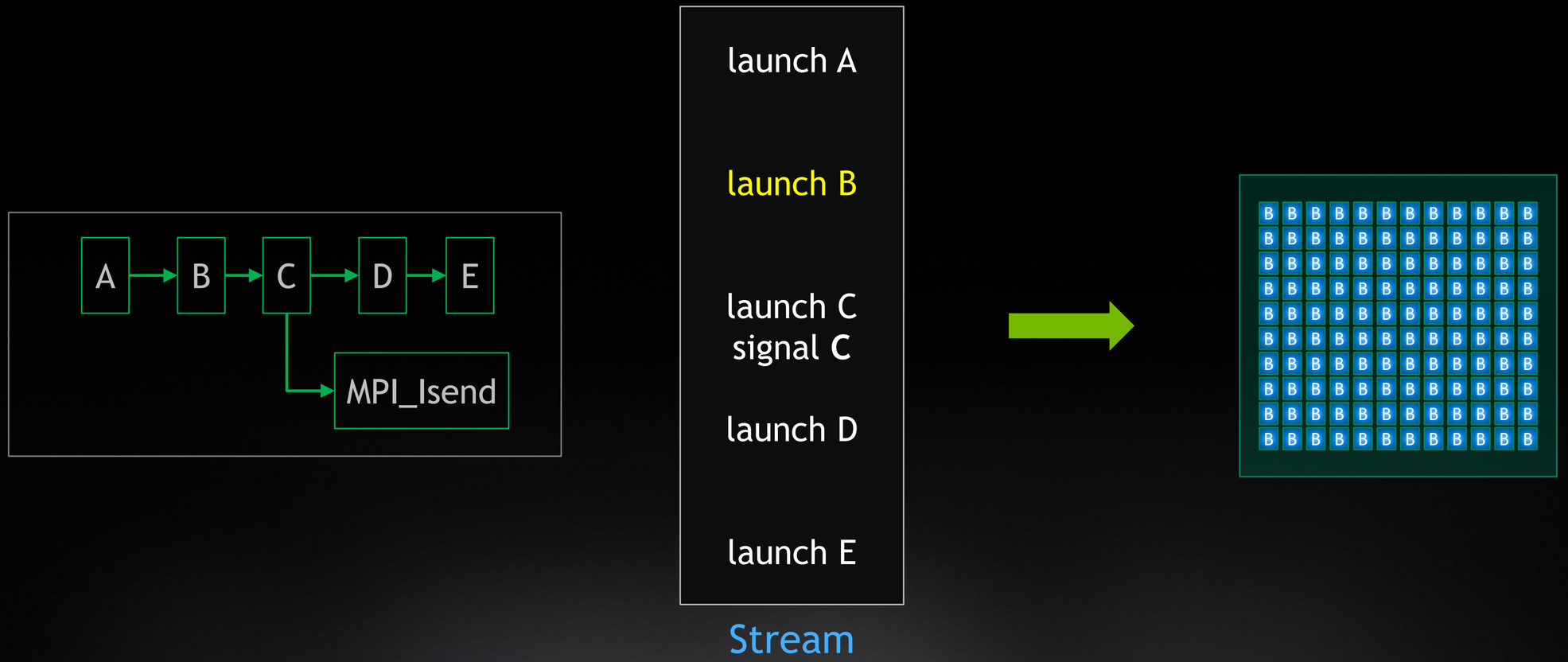


Stream



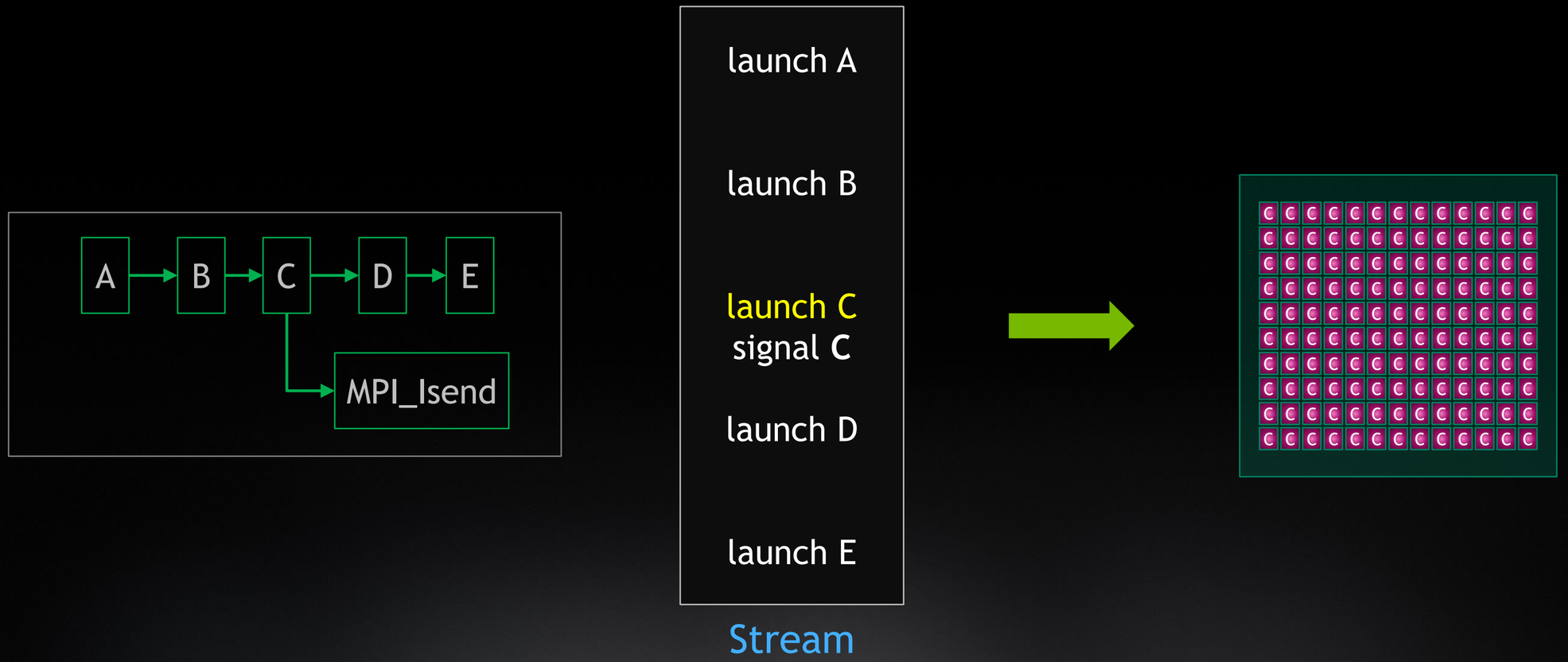
TRIGGERING EXTERNAL WORK

Triggering a CPU operation without blocking GPU flow



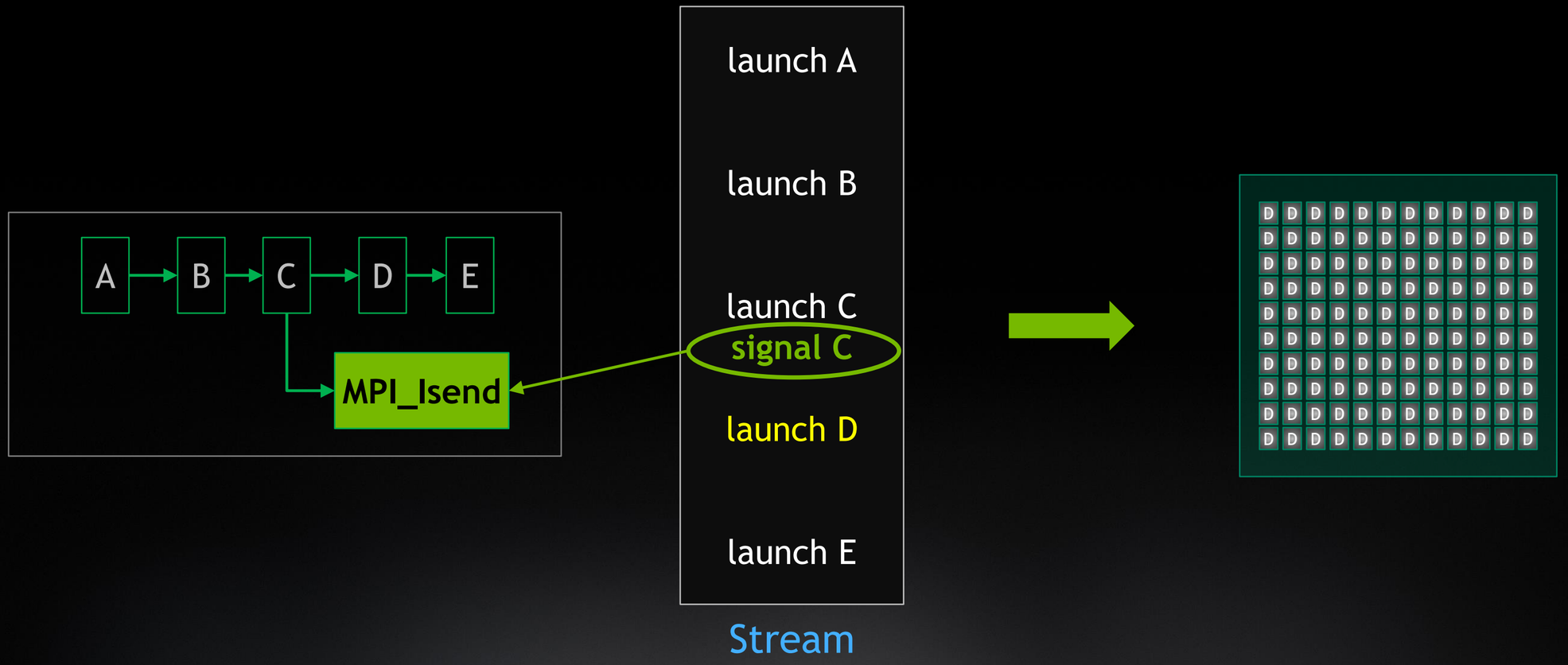
TRIGGERING EXTERNAL WORK

Triggering a CPU operation without blocking GPU flow



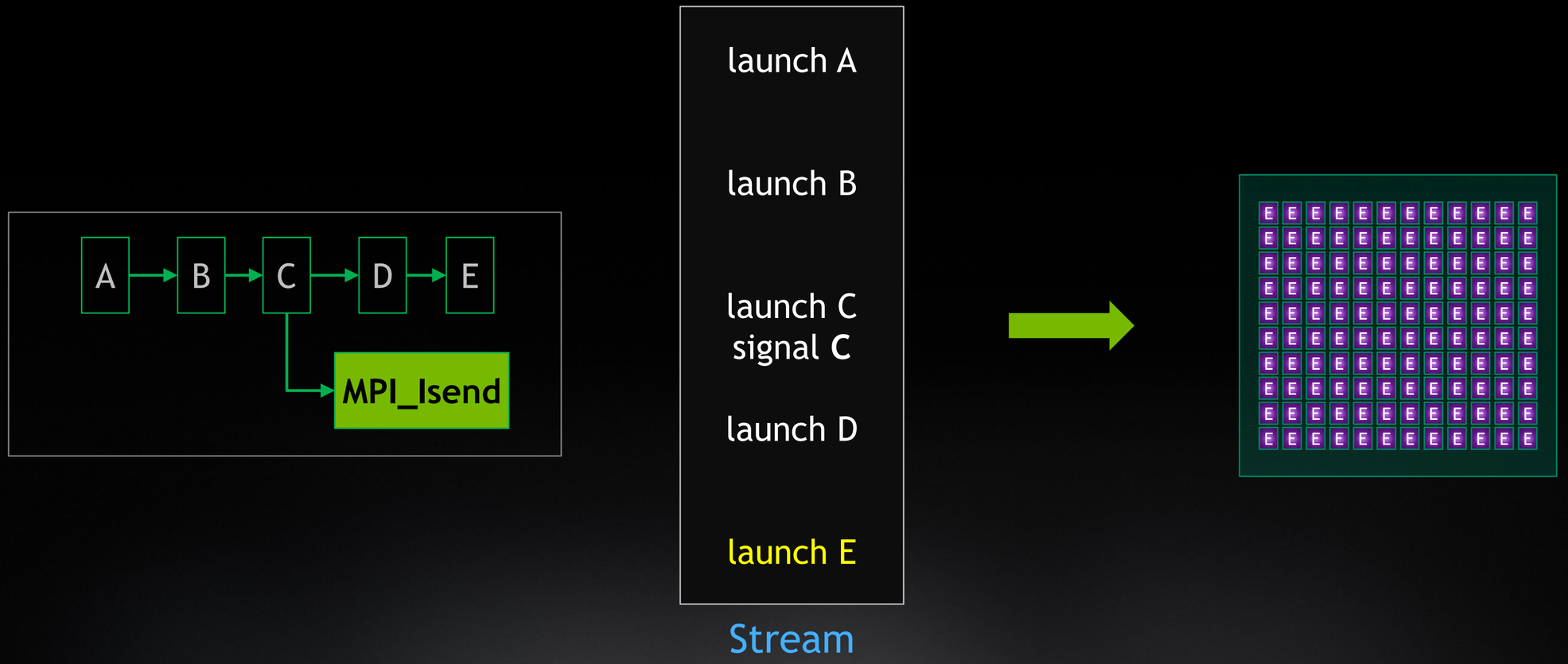
TRIGGERING EXTERNAL WORK

Triggering a CPU operation without blocking GPU flow

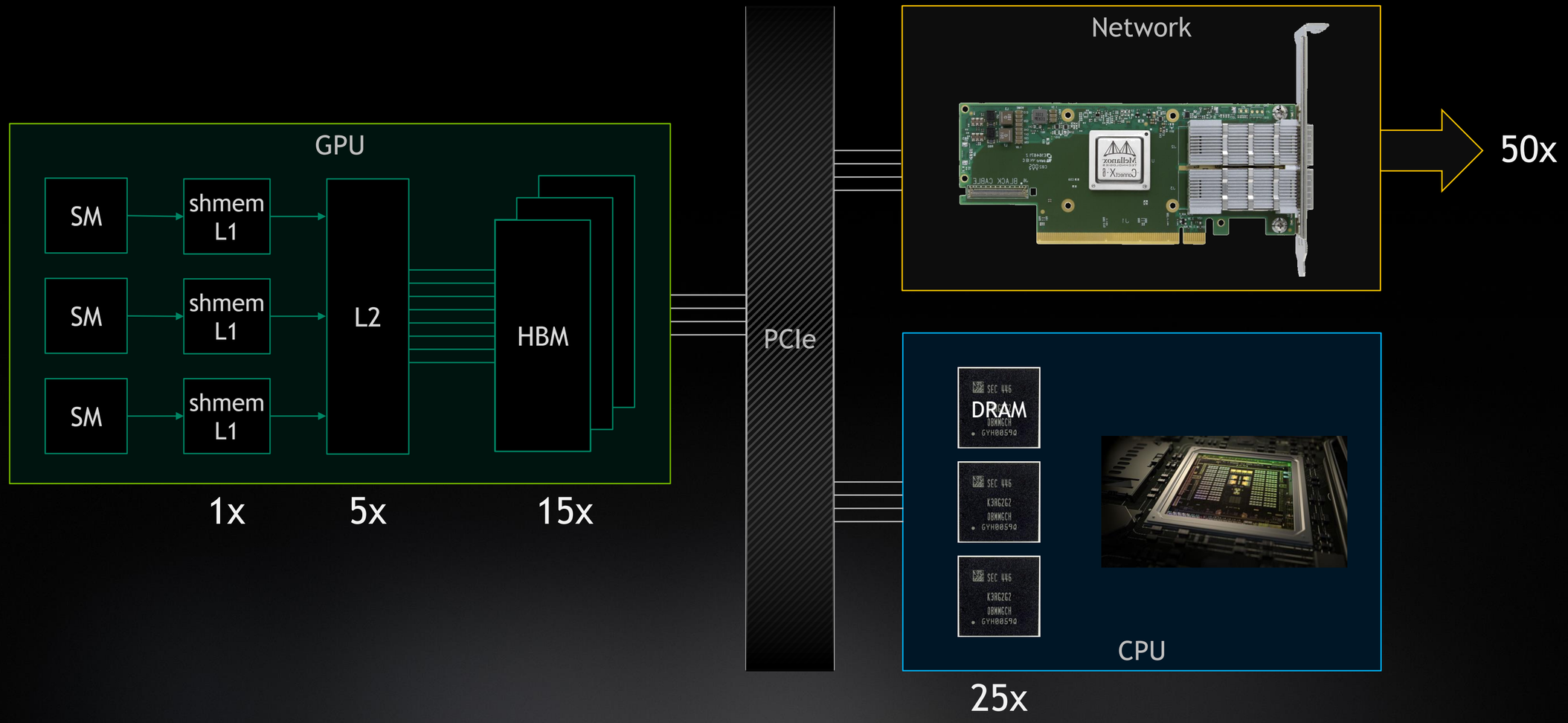


TRIGGERING EXTERNAL WORK

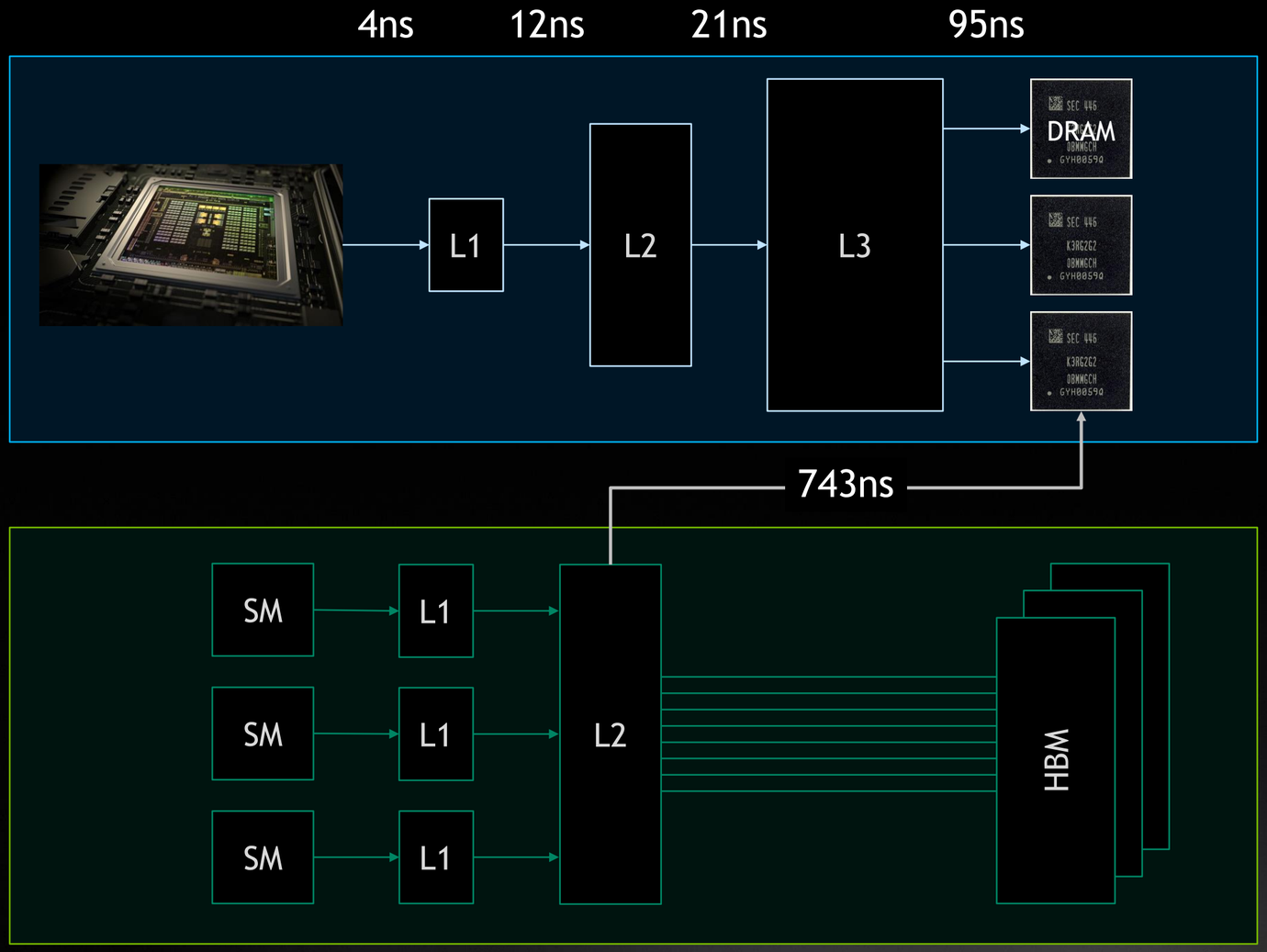
Triggering a CPU operation without blocking GPU flow



GPU EXECUTION: HIERARCHY OF LATENCIES



COMPARING MEMORY LATENCIES



Memory latencies are the best possible delay to exchange signals

Synchronizing the GPU costs between 10us & 50us, or more

An asynchronous signal is therefore 10x to 100x faster than stop-sync-restart

CPU: Intel Xeon E5-2699v4 + DDR4-2400
GPU: NVIDIA GV100 @ 1.5GHz

LAZY EXECUTION: THE PENALTY

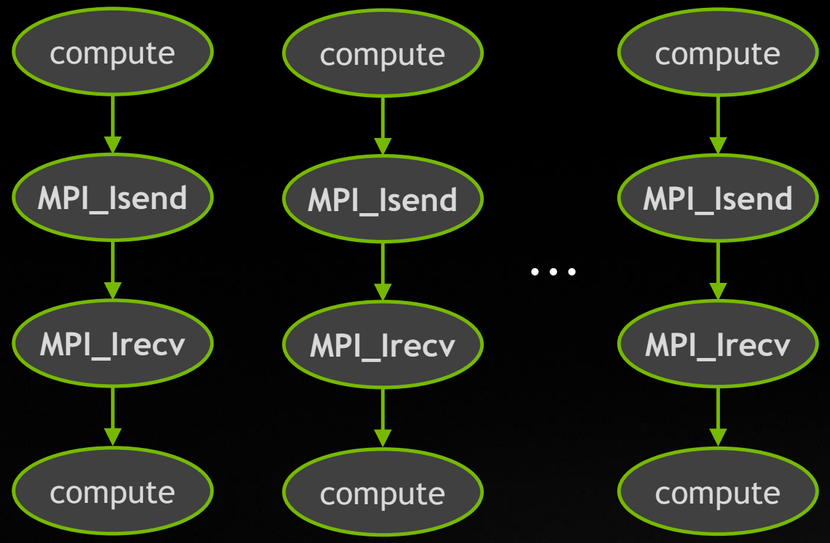
Asynchronous (deferred) execution must

- Record the parameters of the call for later execution
- Manage execution & memory dependencies so that *data_from_c* is visible at the right time
- Set up an efficient callback mechanism to invoke the deferred function

```
A<<< stream >>>();
B<<< stream >>>();
C<<< stream >>>();
MPI_Isend_on_stream( data_from_C, stream );
D<<< stream >>>();
E<<< stream >>>();
cudaStreamSynchronize();
```

LAZY EXECUTION: SOME BENEFITS

Asynchronous independent concurrency



Neighbours run independently
from a single execution thread

```
for(i=0; i<26; i++) {
     compute_neighbour<<< stream[i] >>>(n[i]);
     MPI_Isend_on_stream( n[i], stream );
     MPI_Irecv_on_stream( h[i], stream );
     compute_from_neighbour<<< stream >>>(h[i]);
}
```

LAZY EXECUTION: SOME BENEFITS

Asynchronous composability

```
main_loop(...) {
    do_work(stream);
    multi_gpu_fft(stream);
    do_more_work(stream);
}
```

```
multi_gpu_fft(stream) {
    fft_X(stream);
    MPI_Isend_on_stream( ..., stream );
    MPI_Irecv_on_stream( ..., stream );
    fft_Y(stream);
    ...
}
```

Function does not have to sync in order to send

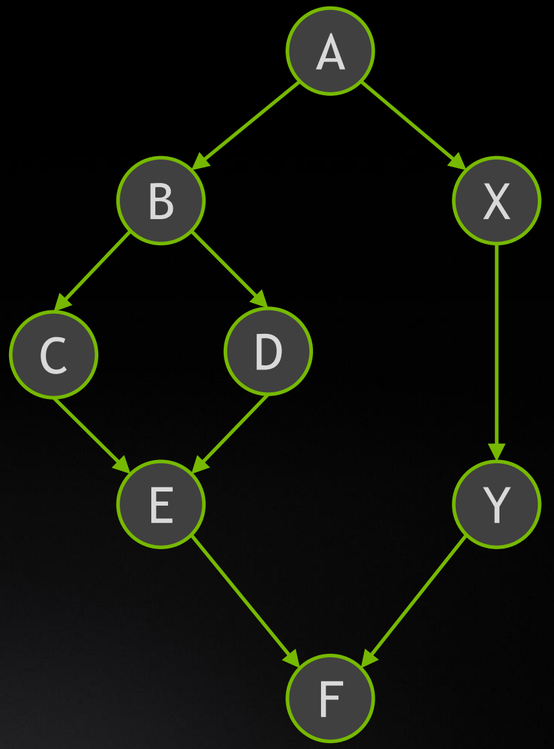
ASYNCHRONOUS TASK GRAPHS

A Graph Node Is A CUDA Operation

Sequence of operations, connected by dependencies

Operations are one of:

Kernel Launch	CUDA kernel running on GPU
CPU Node	Callback function on CPU
Memcopy/Memset	GPU data management
External Dependency	Graphs can depend on or trigger external work



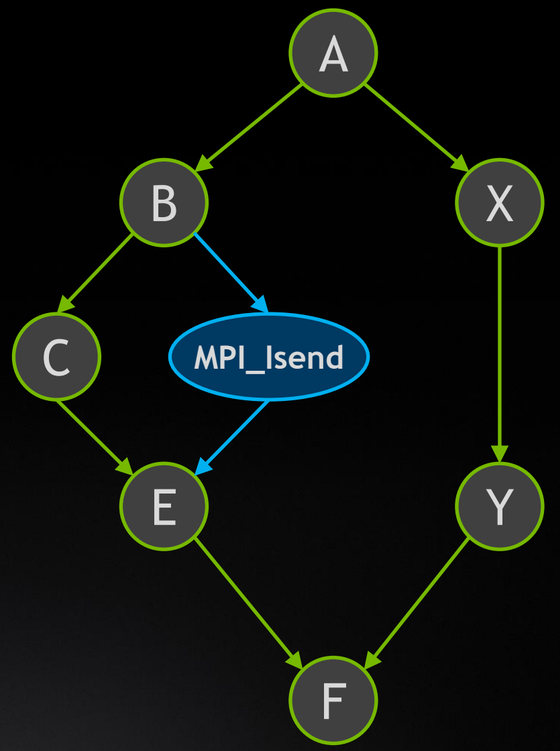
ASYNCHRONOUS TASK GRAPHS

A Graph Node Is A CUDA Operation

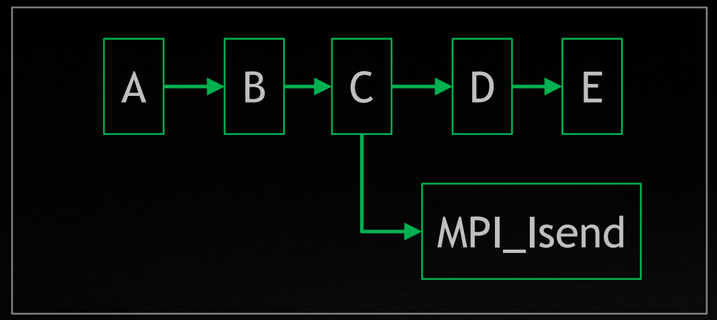
Sequence of operations, connected by dependencies

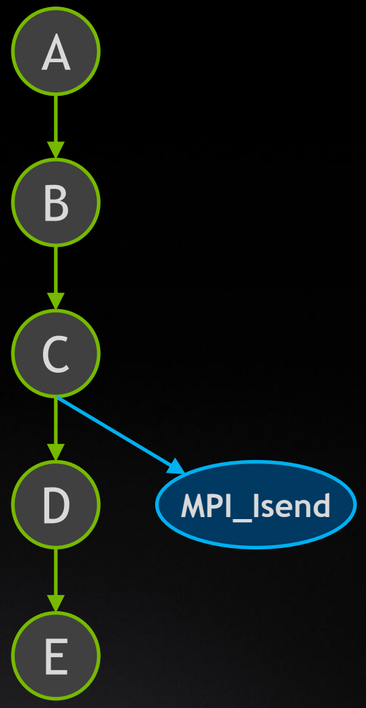
Operations are one of:

Kernel Launch	CUDA kernel running on GPU
CPU Node	**Callback function on CPU**
Memcopy/Memset	GPU data management
Sub-Graph	Graphs are hierarchical



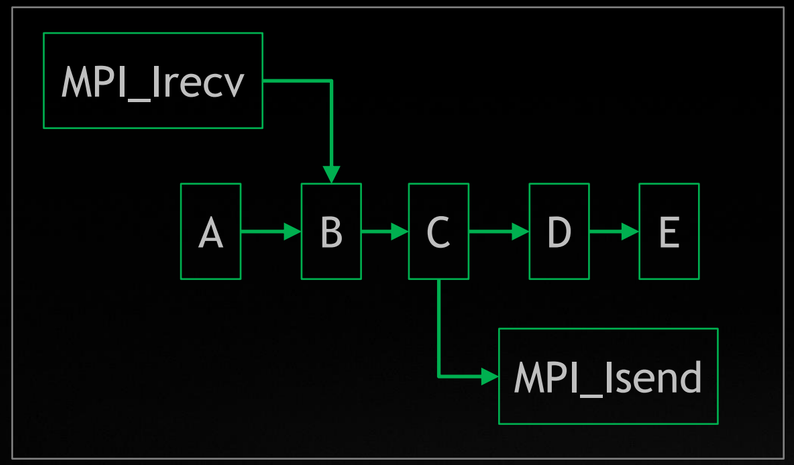
A CUDA GRAPH OF OUR TASK GRAPH

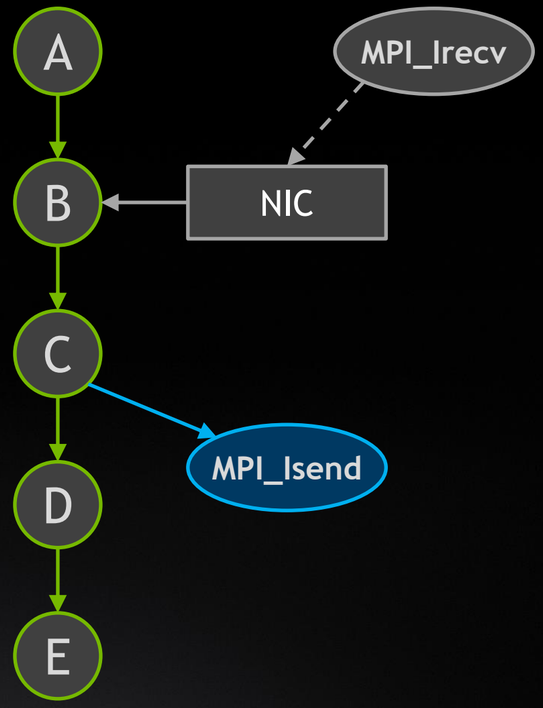




TASK GRAPHS CAN DEPEND ON INCOMING SYNC AS WELL

The obvious way to compose receive as well as send...





IN CONCLUSION

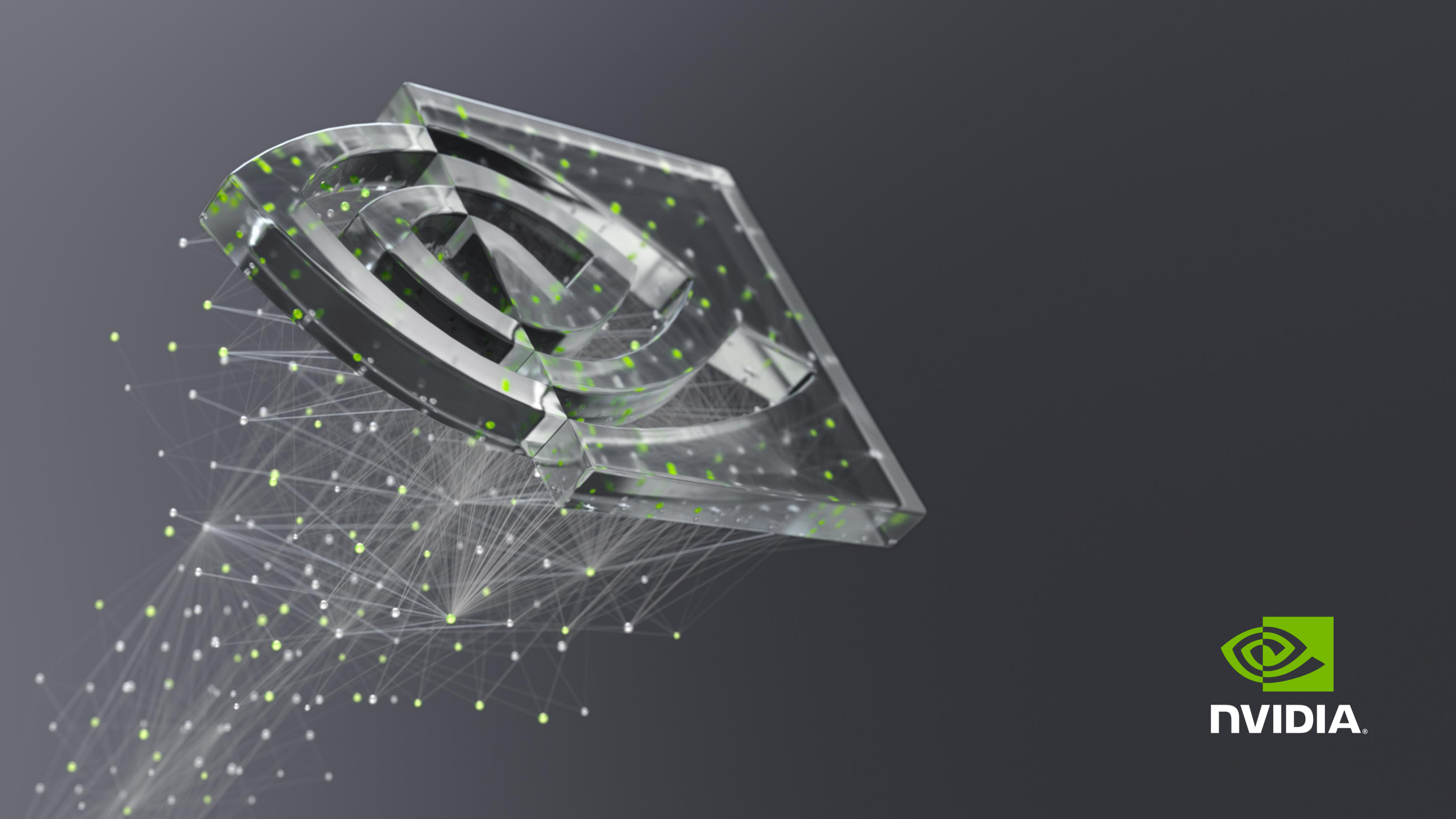
Asynchronous CPU/GPU interaction is as important as asynchronous I/O

Eager execution prevents this – inputs must be consumed at time of call

Deferring execution via external sync can be both general and 10-100x faster

CUDA’s streams support external sync mechanisms today

CUDA’s task graphs explicitly span heterogeneous boundaries to make this easier


NVIDIA