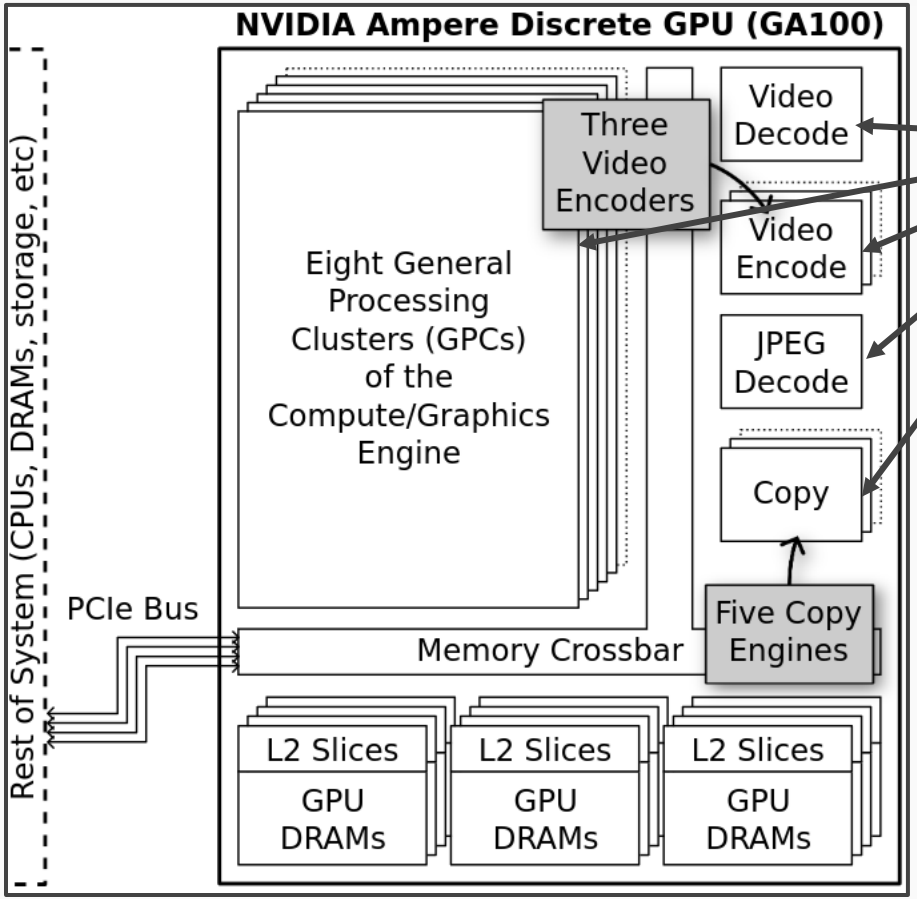
Demystifying NVIDIA GPU Internals to Enable Reliable GPU Management

Joshua Bakita and James H. Anderson

Department of Computer Science
University of North Carolina, Chapel Hill

How can we do more, with less?

Doing More with Less: Leveraging the GPU



Many GPU Engines

My prior work

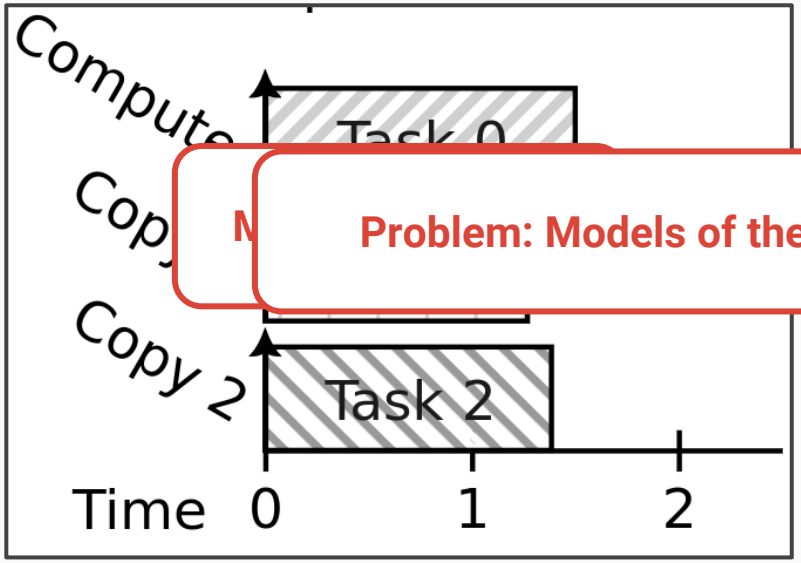
More efficiently using GPU compute cores

This work

More efficiently using multiple GPU engines

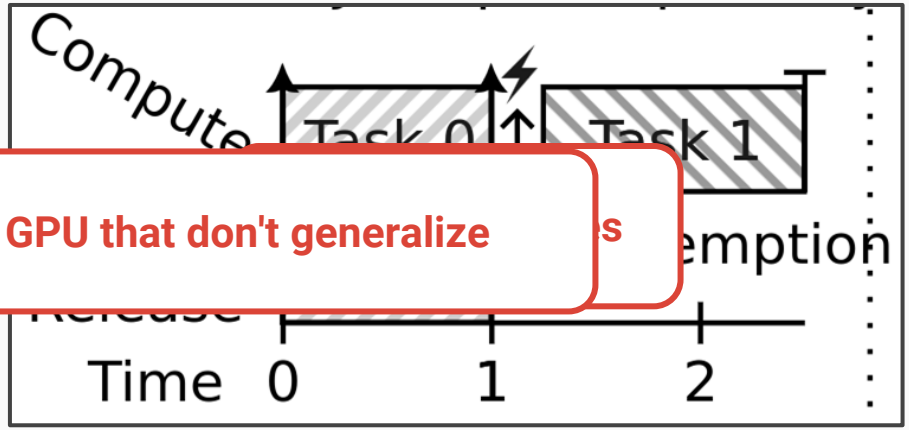
Prior Work on Efficient GPU Engine Use

Use Engines in Parallel, with Locking



Elliot et al. [2]

Use Engines Synchronously, and Preempt



Capodieci et al. [3]

Problem: Models of the GPU that don't generalize

Key Goals

Rules of NVIDIA GPU-internal scheduling that are:

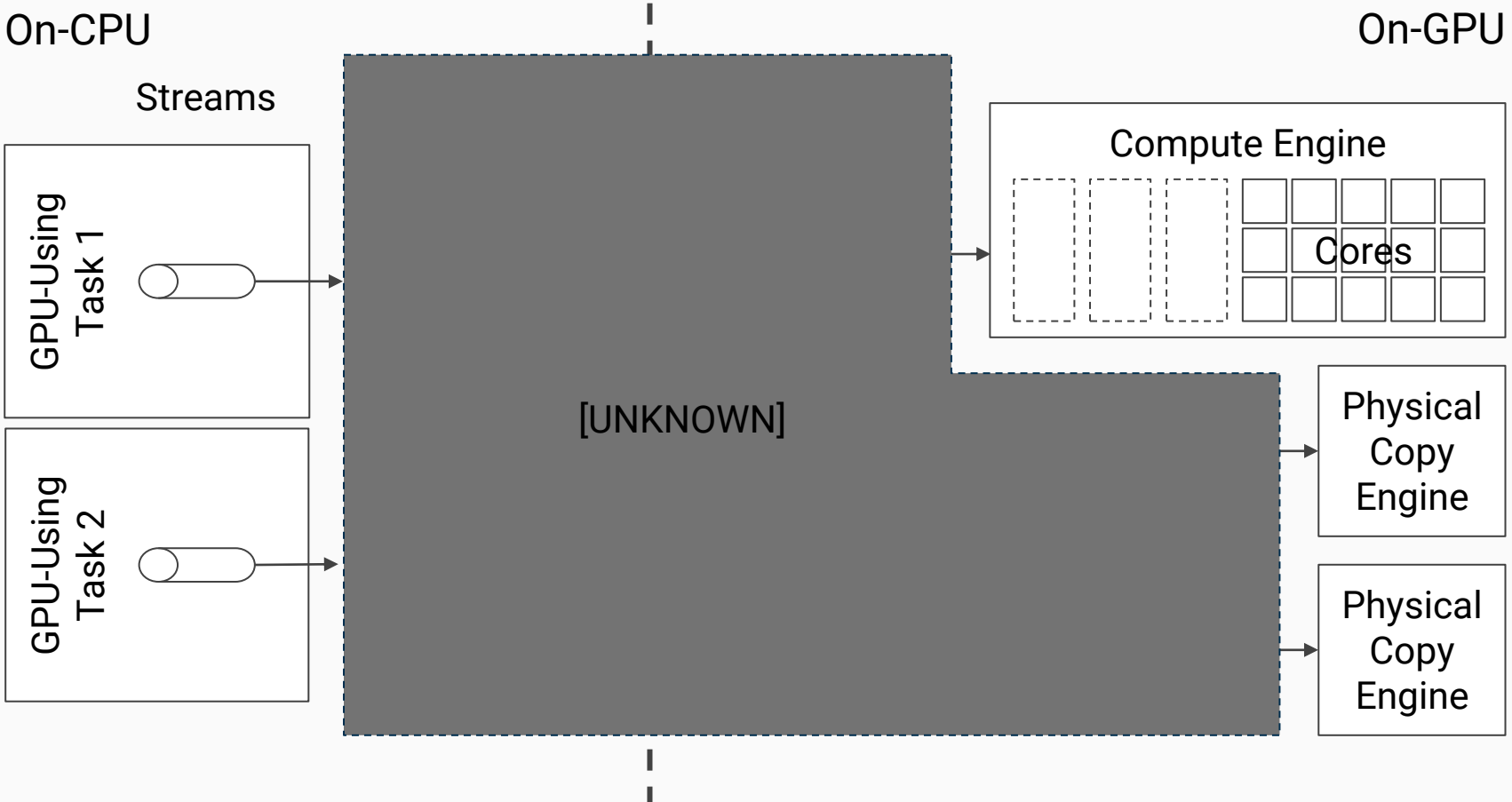
Dependable **Comprehensive** **Necessary**

Via **new experimental tools** and **approaches**, we derive such rules for *any* NVIDIA GPU from the past 8 years.

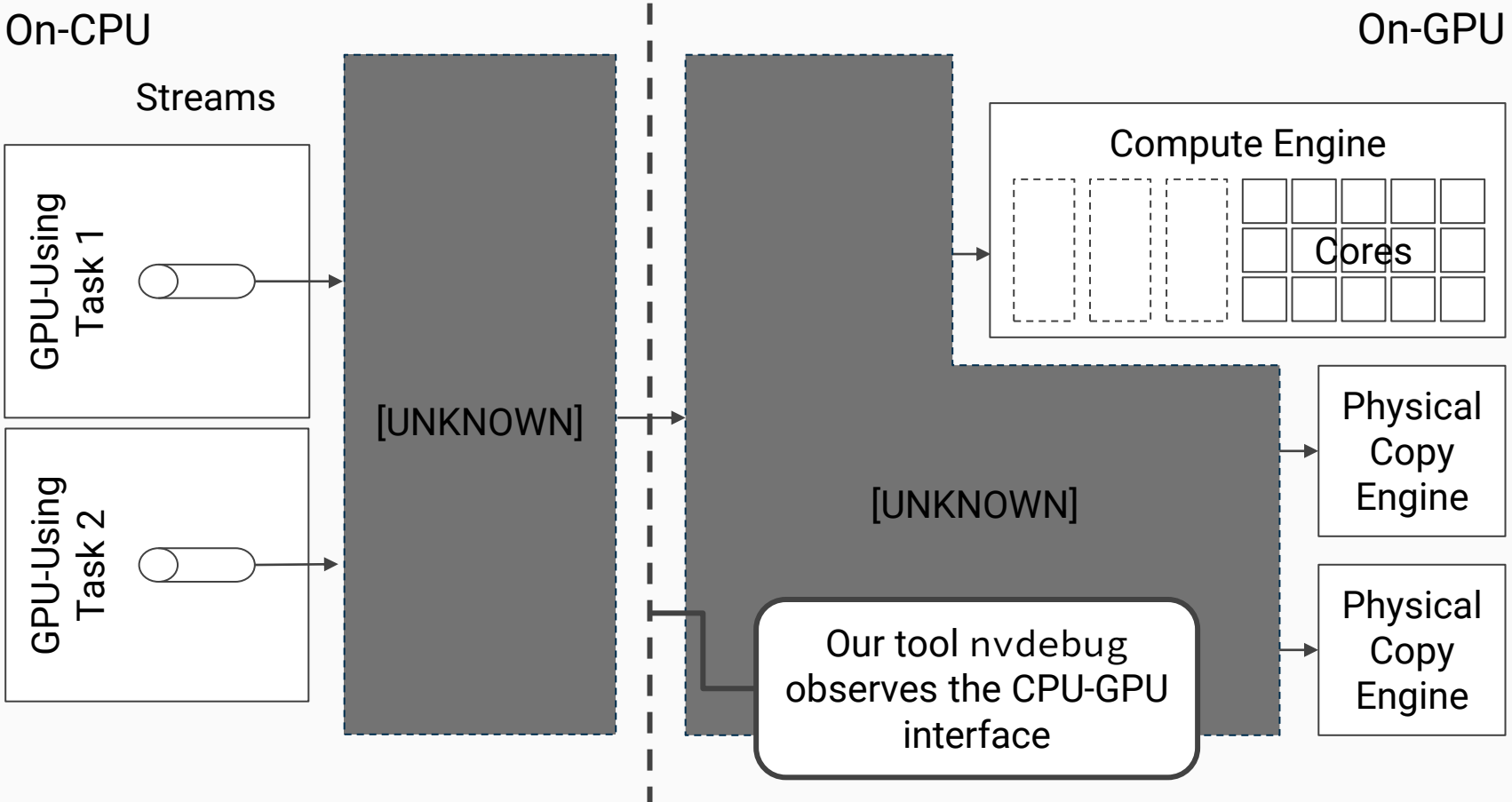
Dependable GPU Scheduling Rules

Goal 1 of 3

Divide and Conquer



Divide and Conquer



Dependable Scheduling Rules

nvdebug

Portable: Works on any NVIDIA GPU of compute capability >3.0 (2013)* and Linux >4.9 (2016)

Key insight: GPU internal scheduling structures are rarely redesigned

Observes GPU state via PCIe and platform registers.

On Linux:

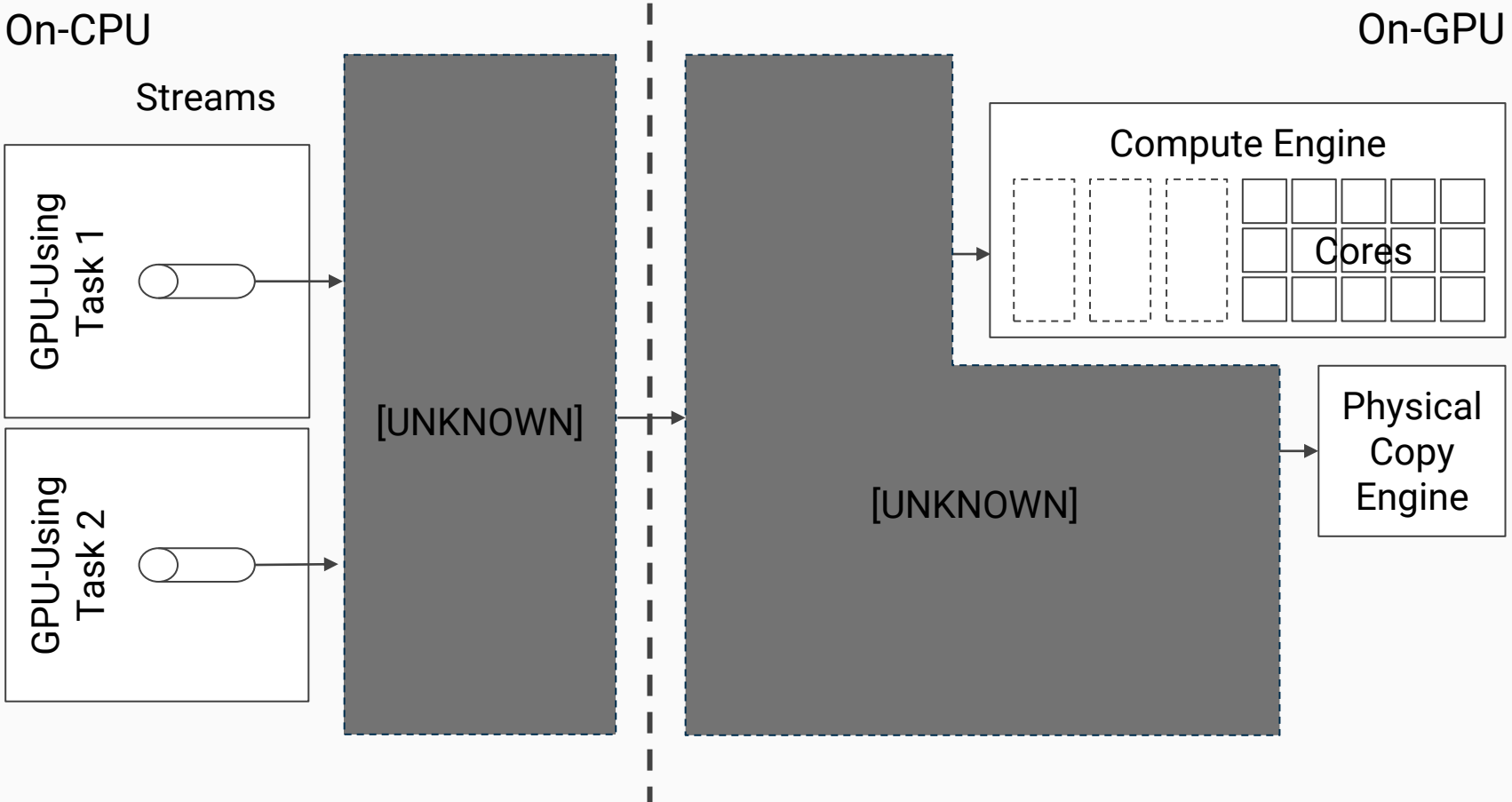
1. Install `gcc` and `linux-headers-generic`
2. Clone `http://rtsrv.cs.unc.edu/cgit/cgit.cgi/nvdebug.git`
3. Run `make` and `sudo insmod nvdebug.ko`

Code is open source and documented. See https://www.cs.unc.edu/~jbakita/rtas24-ae/ to get started.

Comprehensive GPU Scheduling Rules

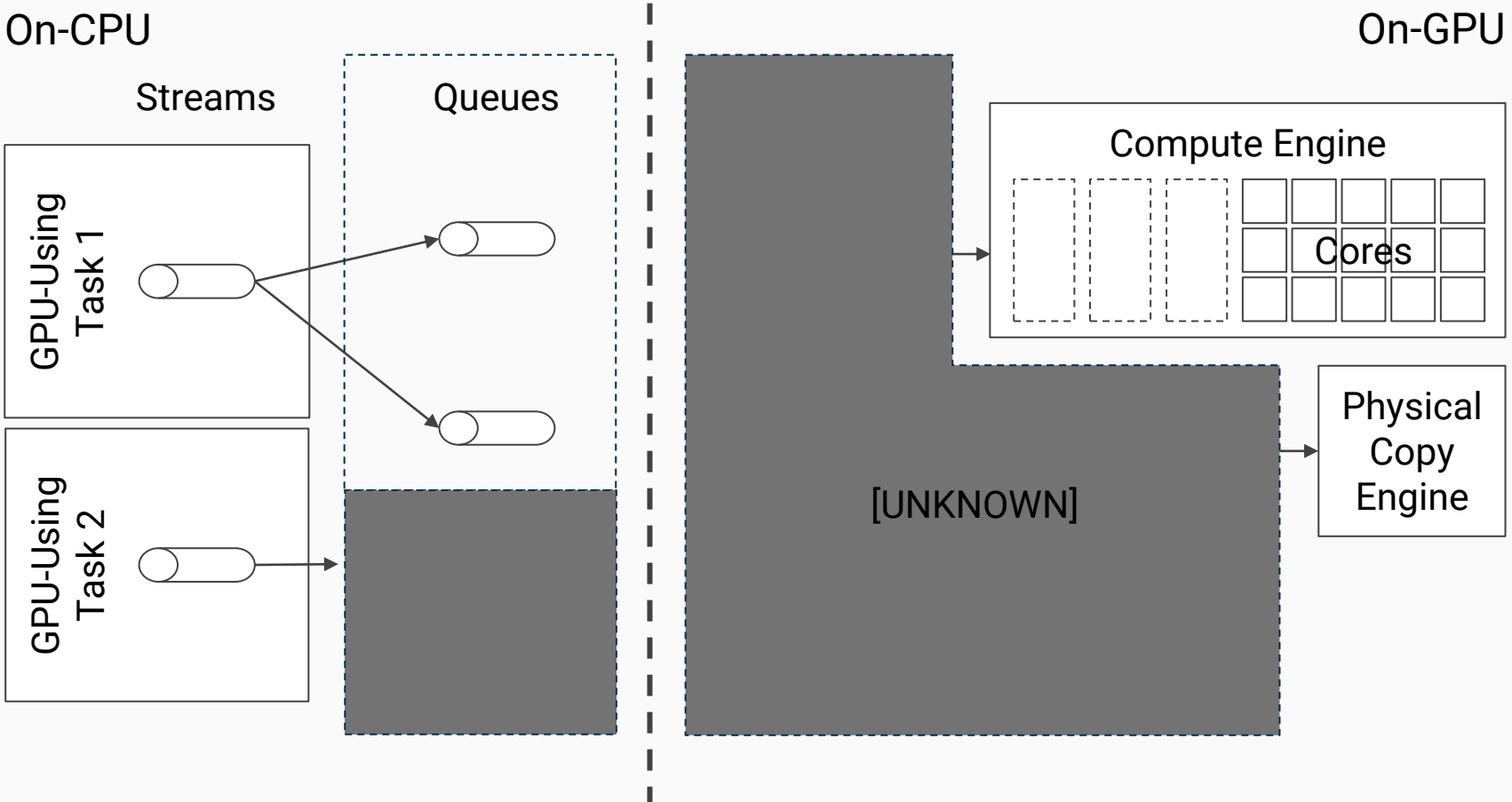
Goal 2 of 3

Dependable Scheduling Rules



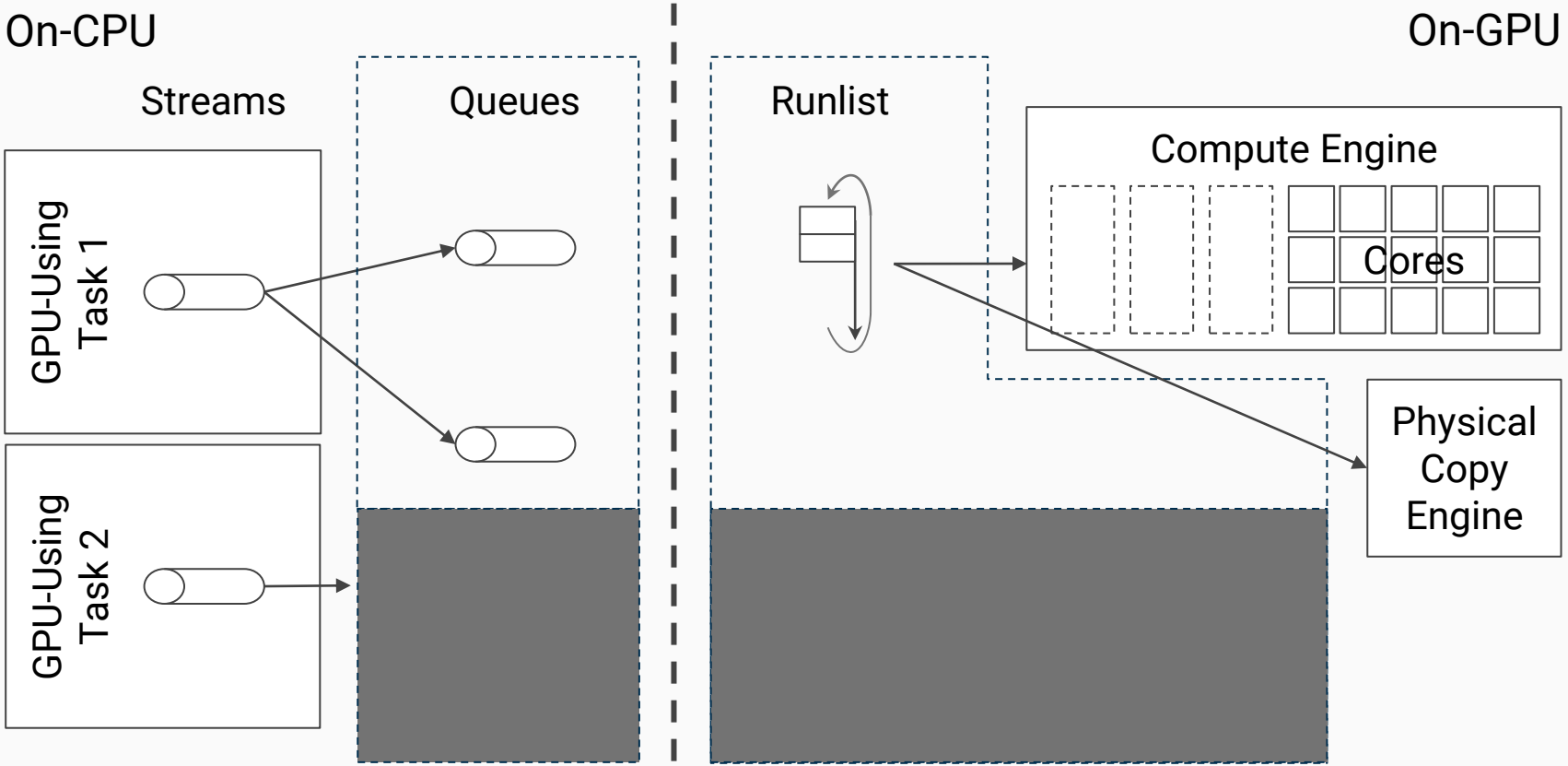
Comprehensive Scheduling Rules

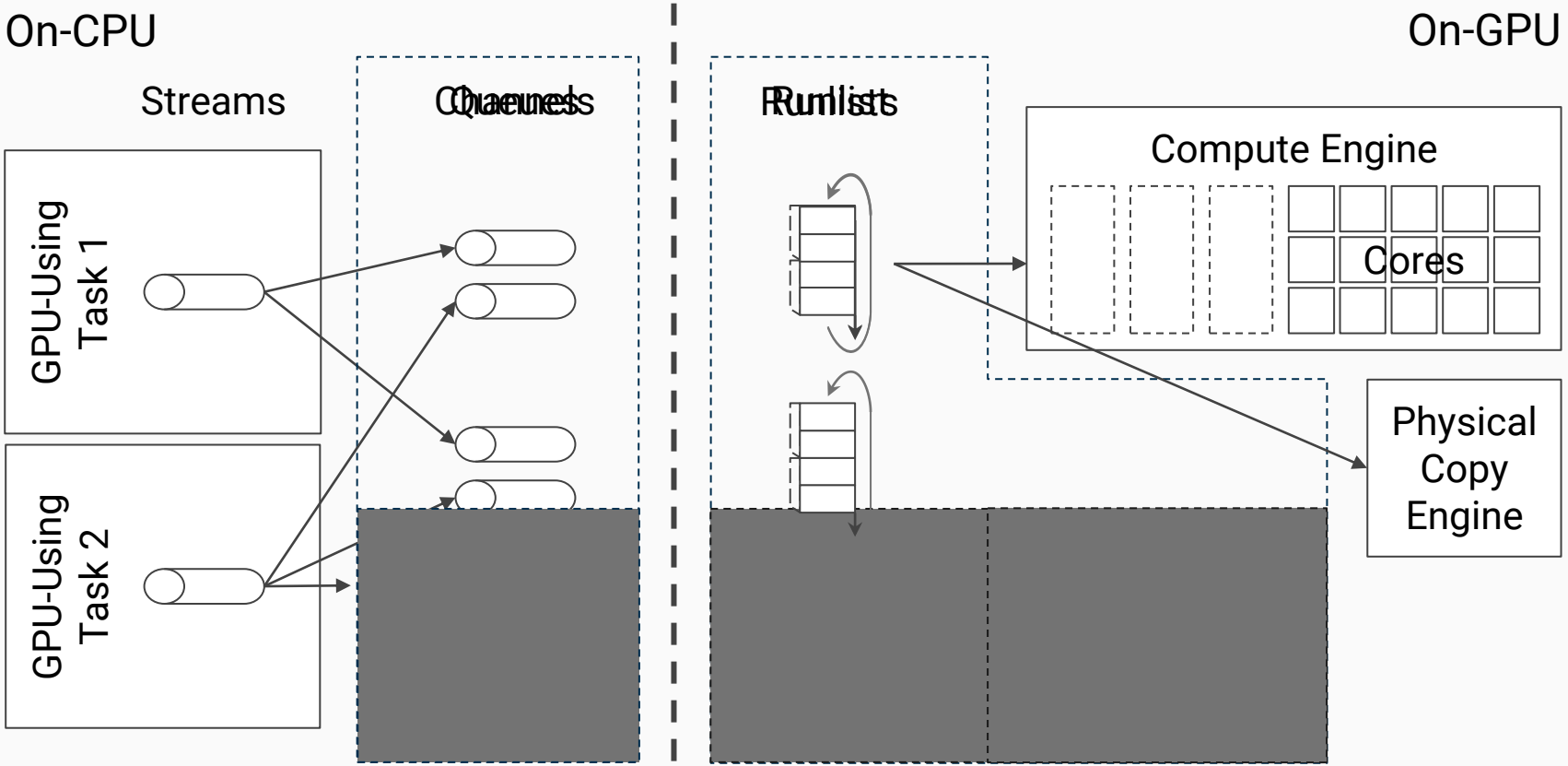
Prior Work [5] (Amert, 2017)

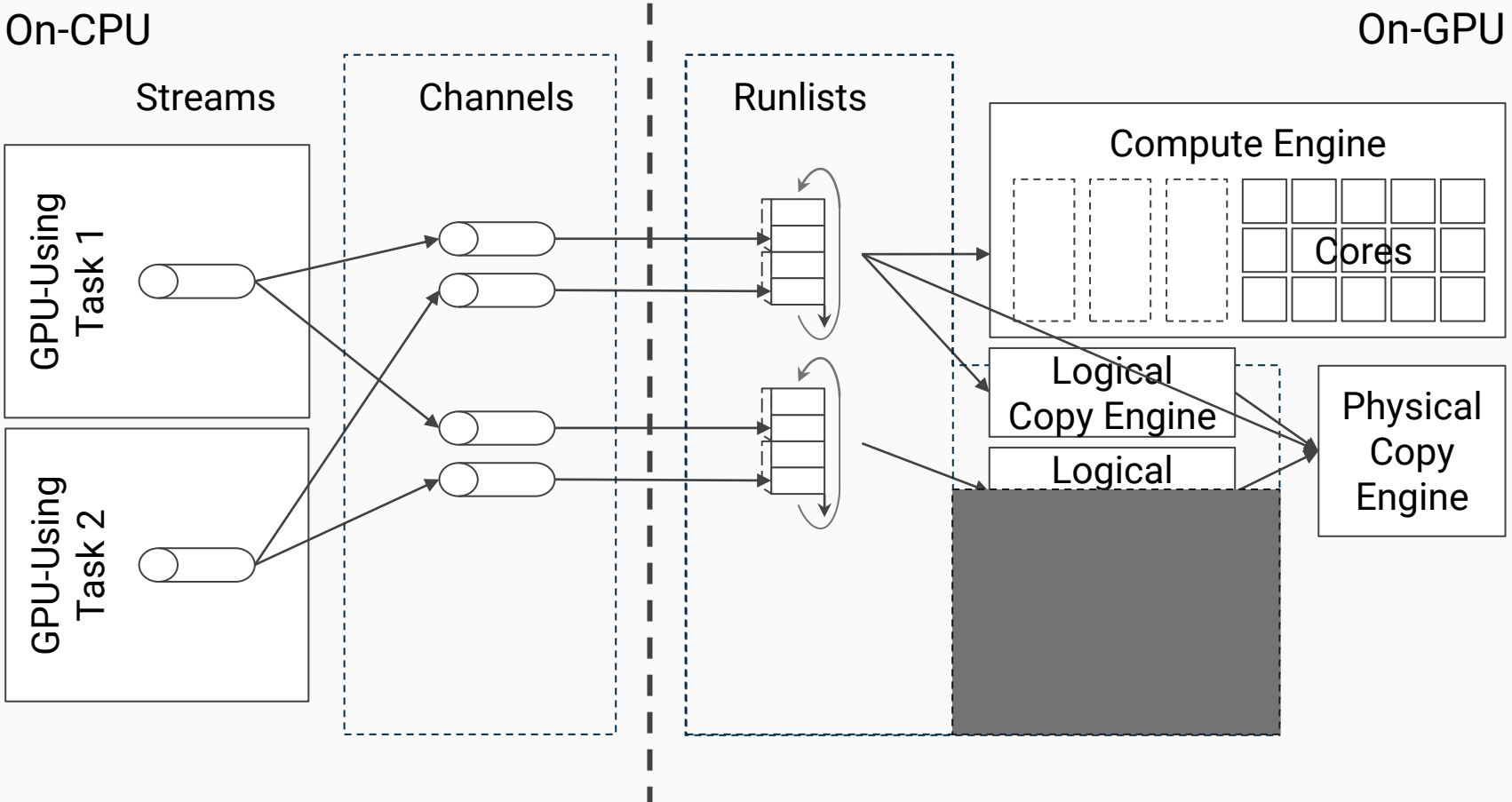


Comprehensive Scheduling Rules

Prior Work [3] (Capodieci, 2018)

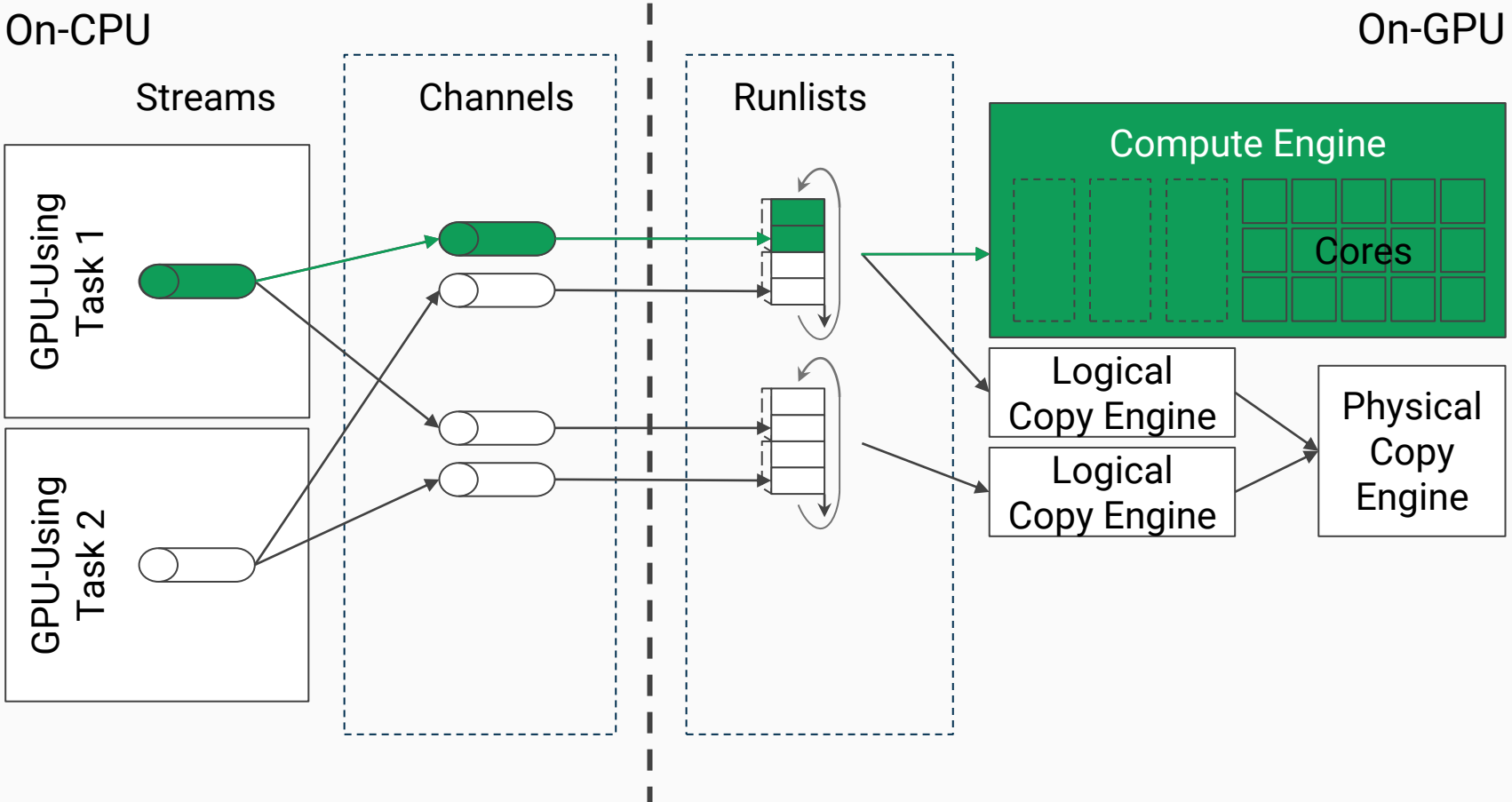



On-CPU
On-GPU
Streams
Channels
Runlists
GPU-Using Task 1
GPU-Using Task 2
Compute Engine
Cores
Physical Copy Engine


On-CPU
On-GPU
Streams
Channels
Runlists
GPU-Using Task 1
GPU-Using Task 2
Compute Engine
Cores
Logical Copy Engine
Logical
Physical Copy Engine

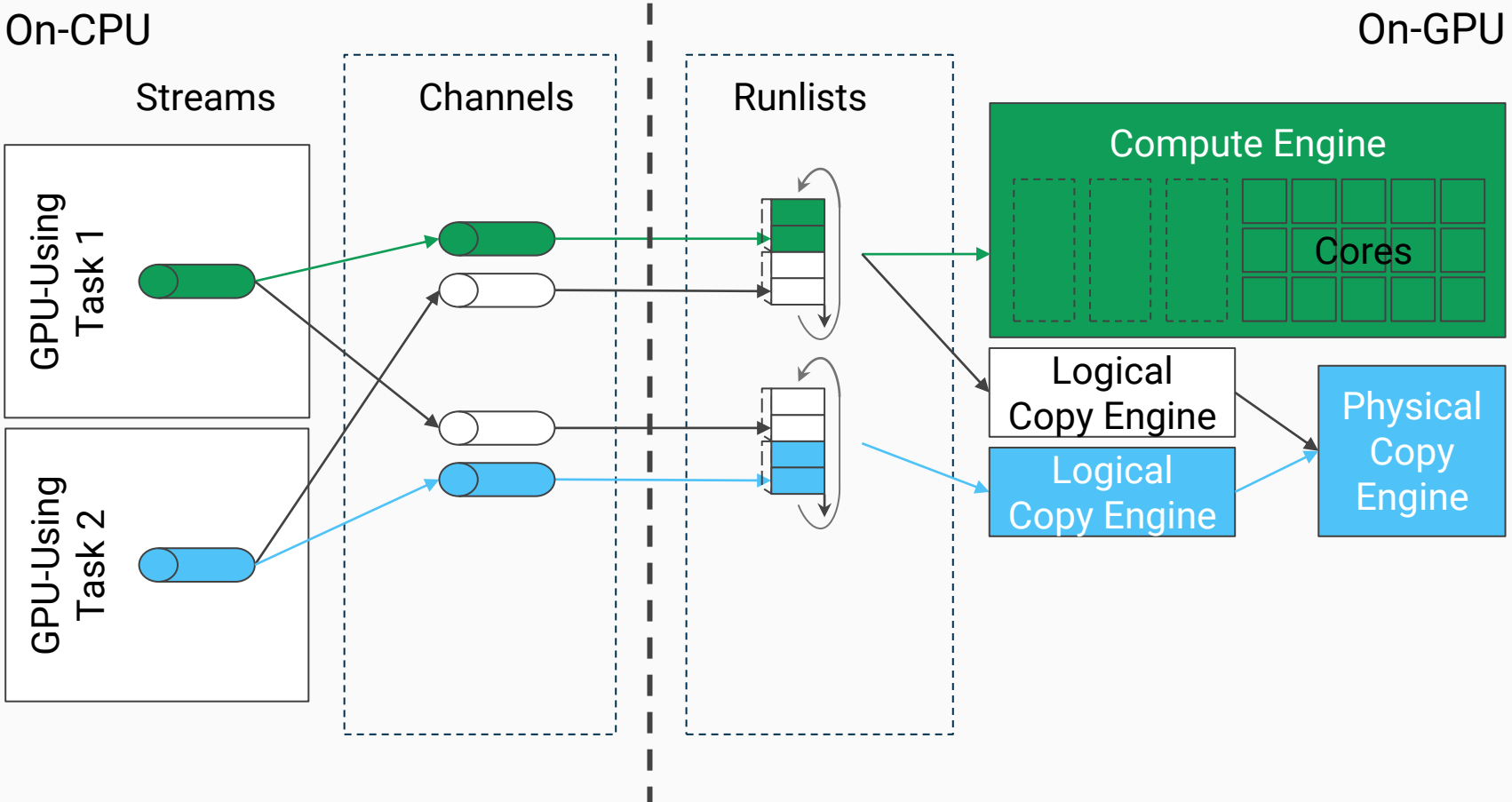
Comprehensive Scheduling Rules

Example



Comprehensive Scheduling Rules

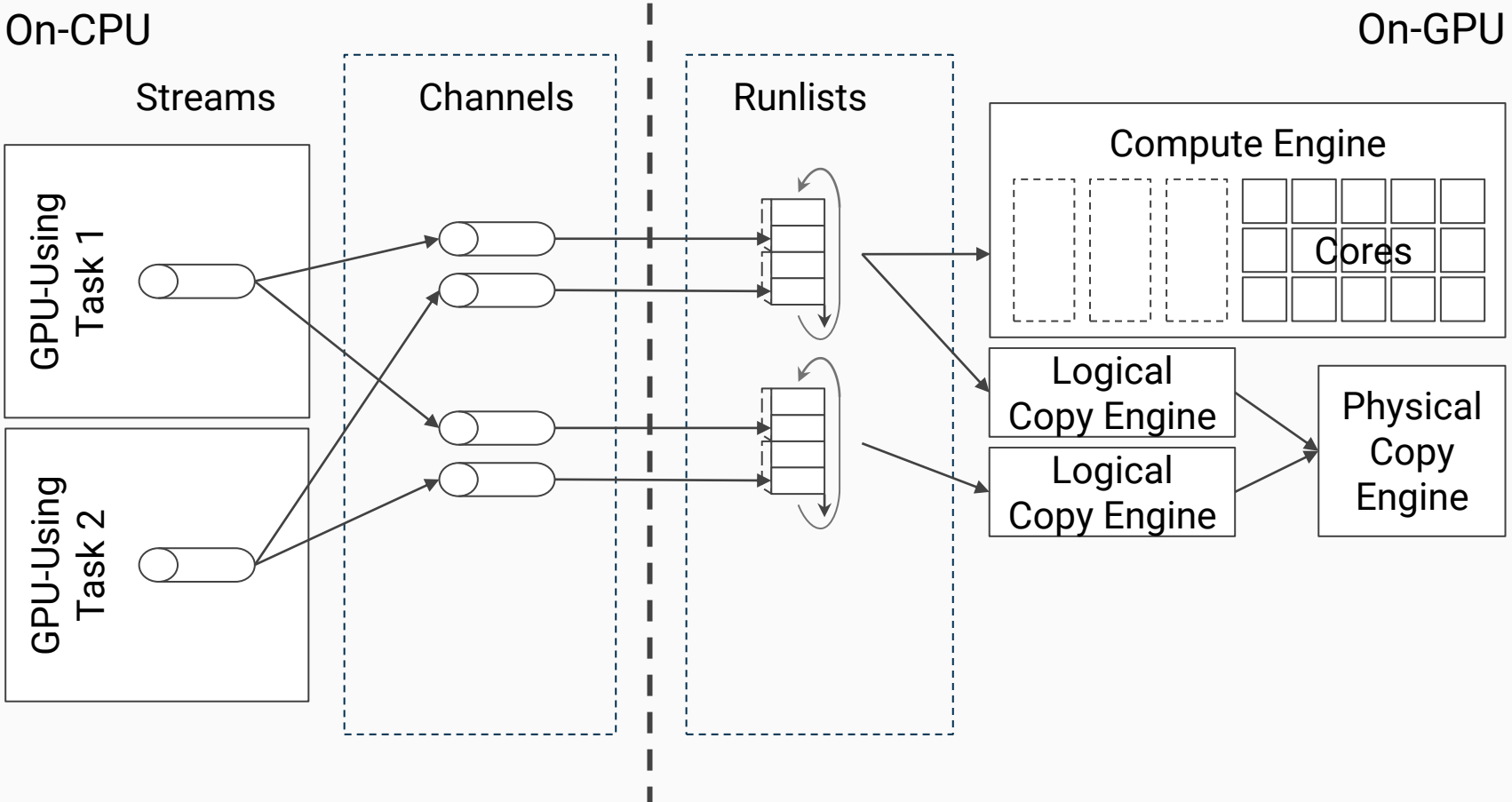
Example



Necessary GPU Scheduling Rules

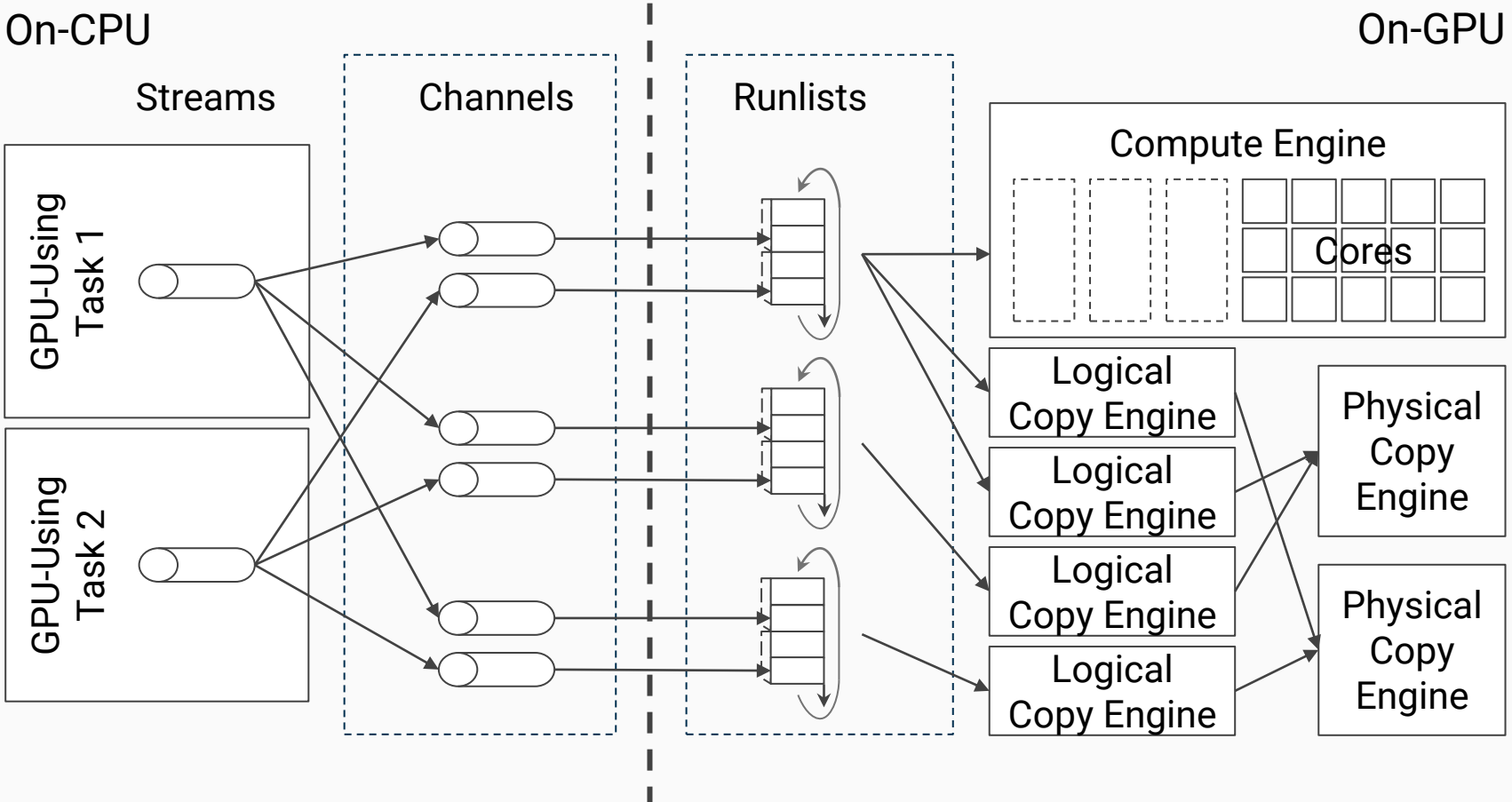
Goal 3 of 3

Necessary Scheduling Rules



Necessary Scheduling Rules

Actual GPU: RTX 6000 Ada

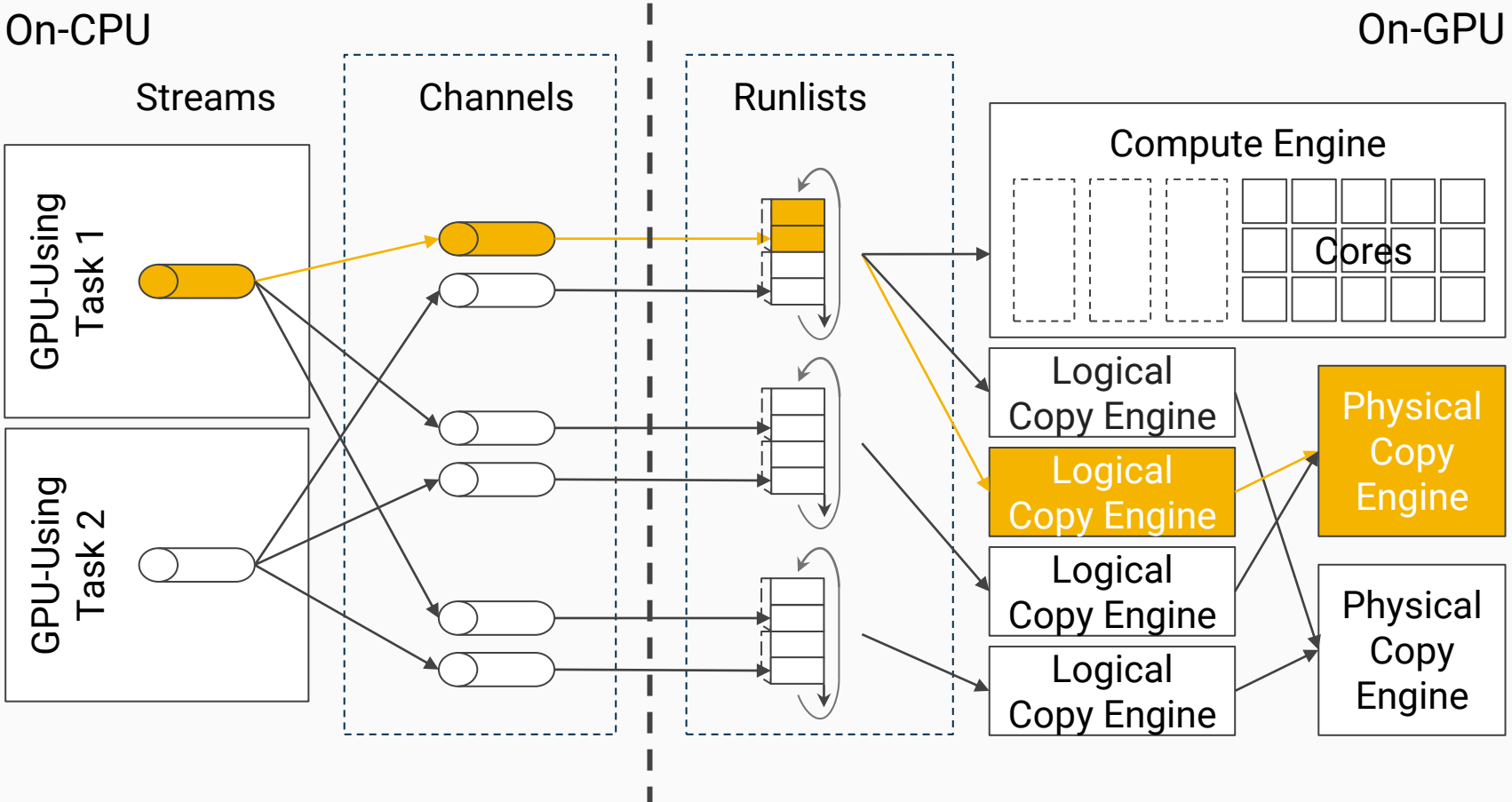


Necessary Scheduling Rules

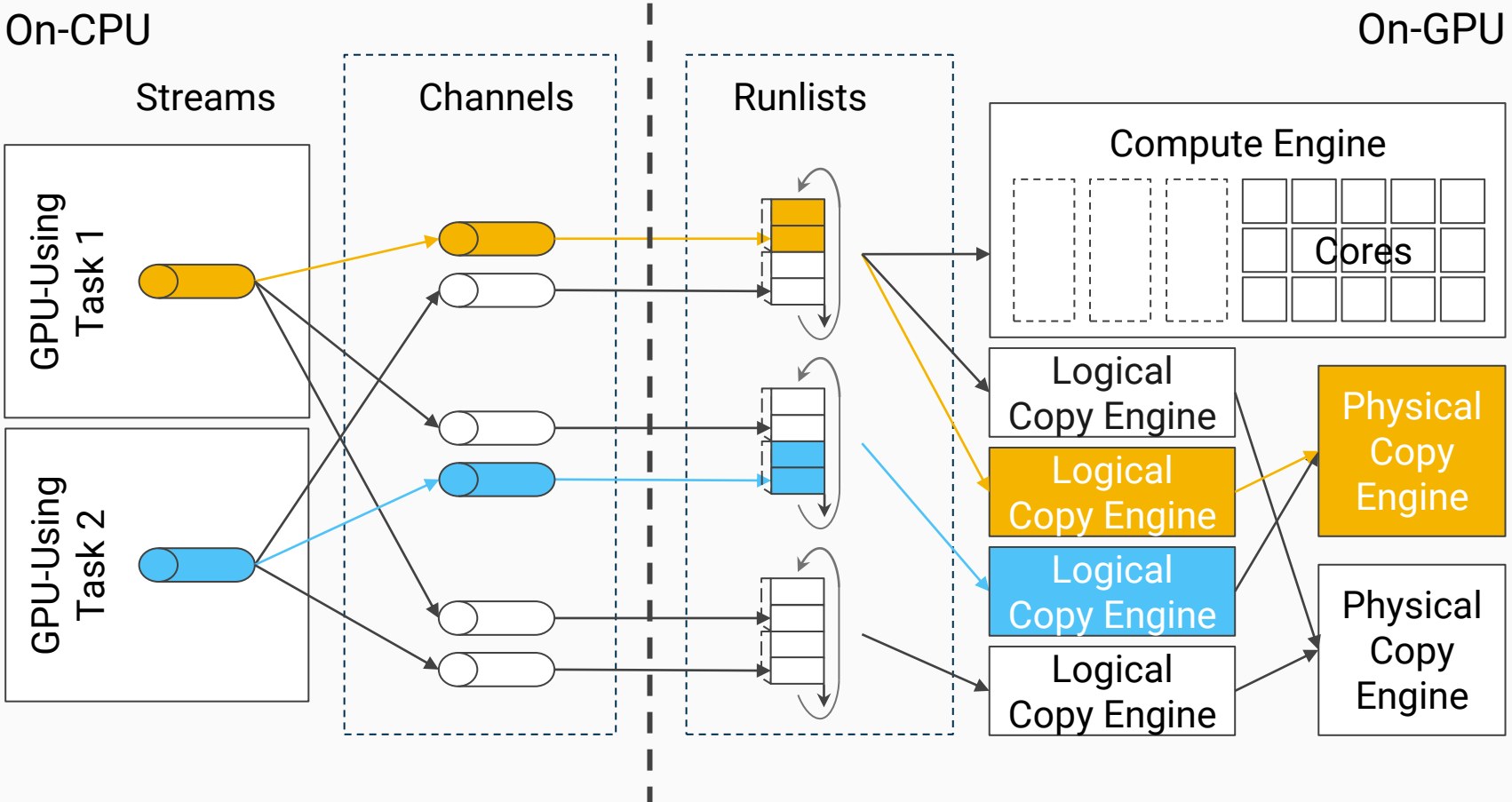
Per-Engine Locking

Assume each engine can be used simultaneously.
Create one lock per engine.
To use an engine, obtain the lock.

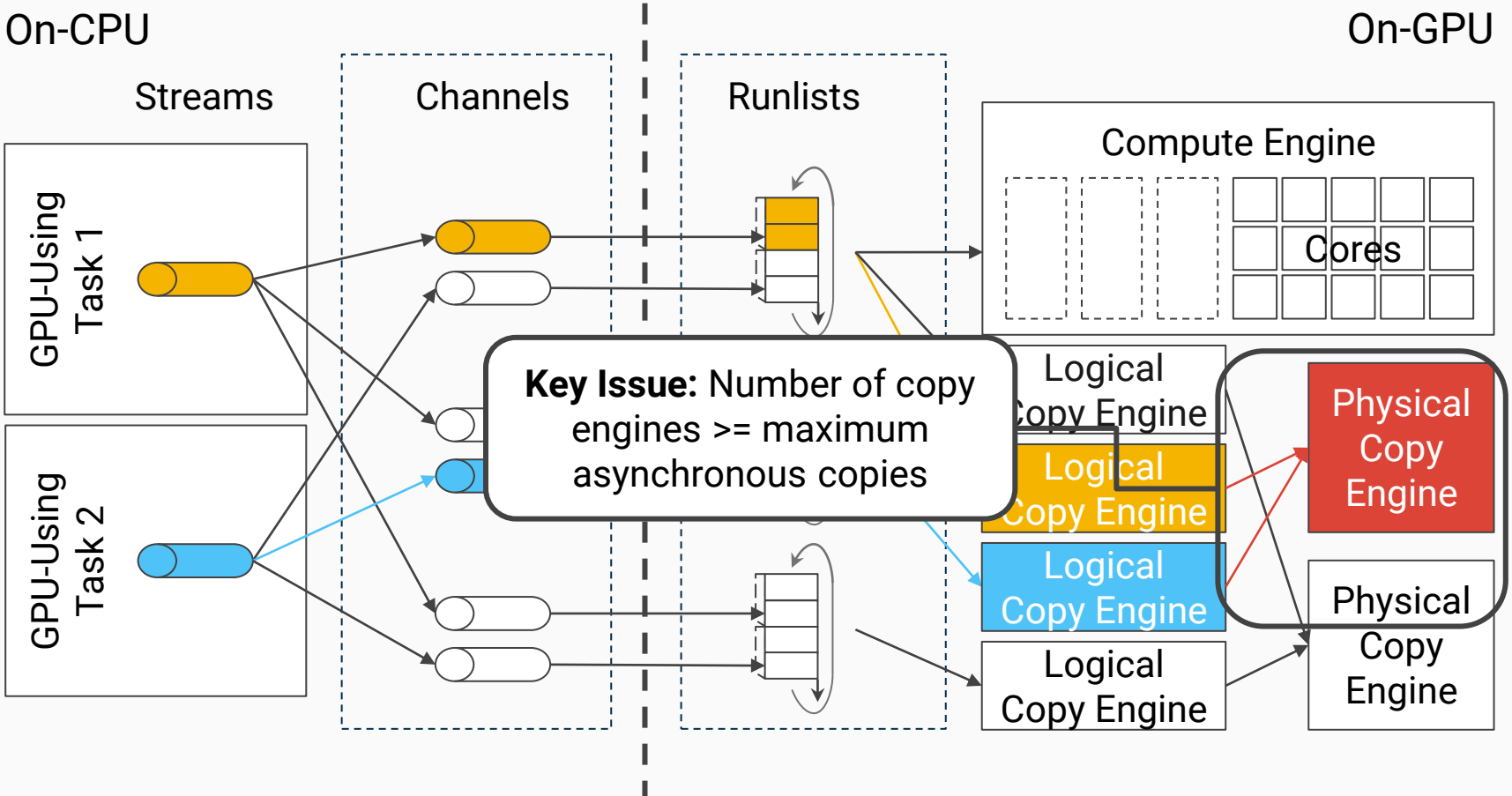
Problems with Per-Engine Locking



Problems with Per-Engine Locking



Problems with Per-Engine Locking



Dependable Scheduling Rules

Problems with other management

Management-free analysis via a large number of streams [4]

Key Issue:
Cannot use more streams than there are channels.

Preemptive scheduling via resetting the runlist [3]

Key Issue:
Only preempts tasks on the first runlist

Conclusions

We provide rules of NVIDIA GPU-internal scheduling that are:

Dependable	**Comprehensive**	**Necessary**
Do GPUs change too much?	Can it describe the path from CUDA to the GPU?	Are the rules needed for safe GPU management?
No. Our nvdebug tool shows that internal structures rarely change.	Yes! We fill in all previously unknown gaps in the pipeline.	Yes. Their absence results in a loss of generalizability.

What you have to read the paper for…

Evaluation:

- A detailed theoretical analysis of how prior GPU management approaches can break down

Tooling:

- Details on the use and features of `nvdebug`
- Our new microsecond-accurate GPU microbenchmark suite `gpu-microbench`

Regarding rules:

- Eight detailed rules, covering tasks to channels, channels to runlists, and runlists to engines
- Detailed microbenchmark experiments to justify and demonstrate each rule

\+ More details and background on everything covered in this presentation

Thanks!
Questions?

Contact:
Email: jbakita@cs.unc.edu
Twitter: @JJBakita
Web: https://cs.unc.edu/~jbakita

Old Well, University of North Carolina at Chapel Hill, Winter 2017