UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Quantifying the NUMA Behavior of Partitioned GPGPU Applications

GPGPU-12, April 13, 2019, Providence, RI, USA

Alexander Matz, Holger Fröning
Heidelberg University

Motivation

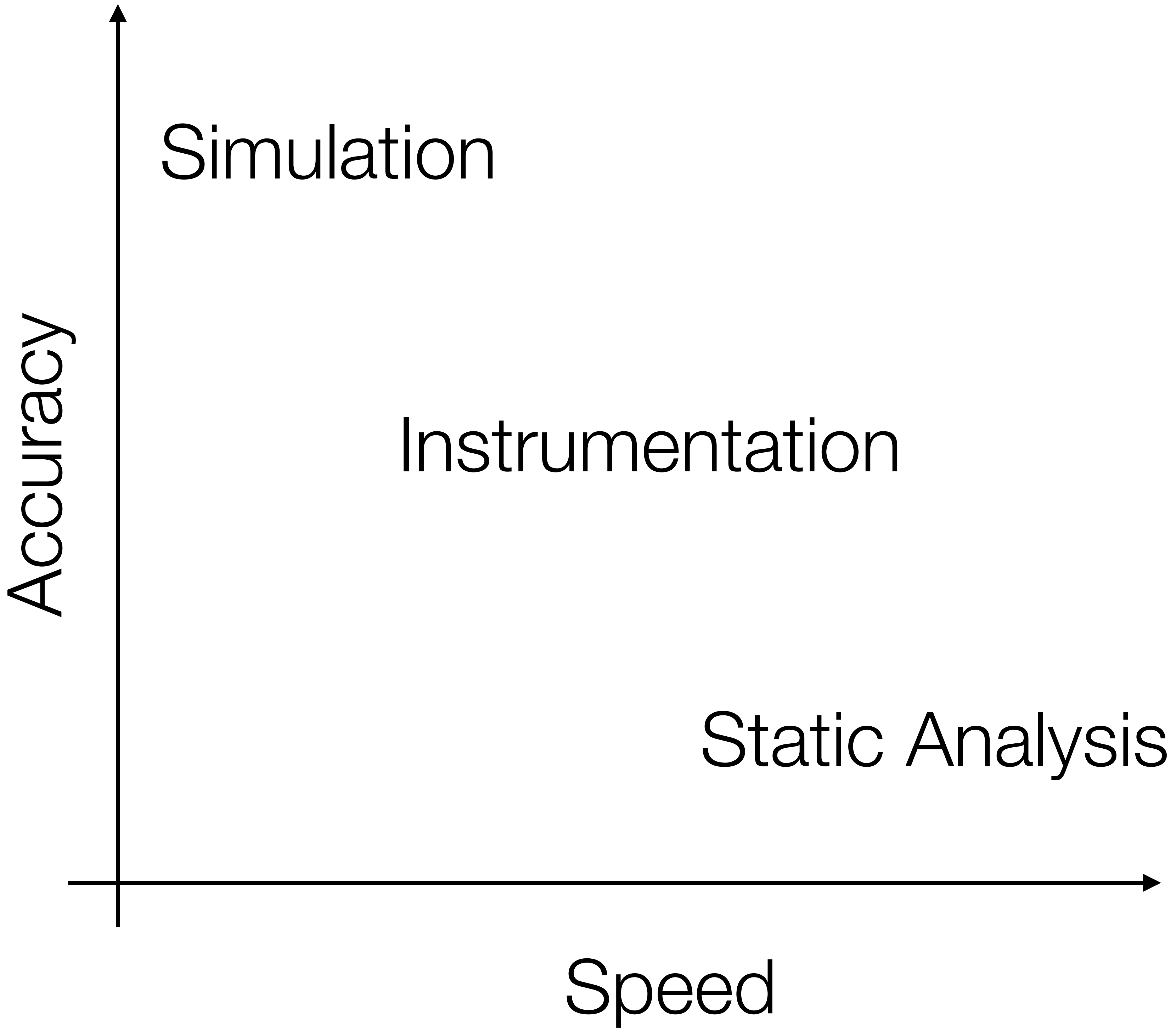
- GPU performance highly susceptible to memory access patterns
- Even stronger in multi-GPU systems (NUMA factor: 10-20x)
- Not much research w.r.t. multi-GPU memory access patterns
- Interesting for multi-GPU ports of single-GPU applications

→ Collect memory traces from single-GPU applications and analyze for virtual multi-GPU behavior

Outline

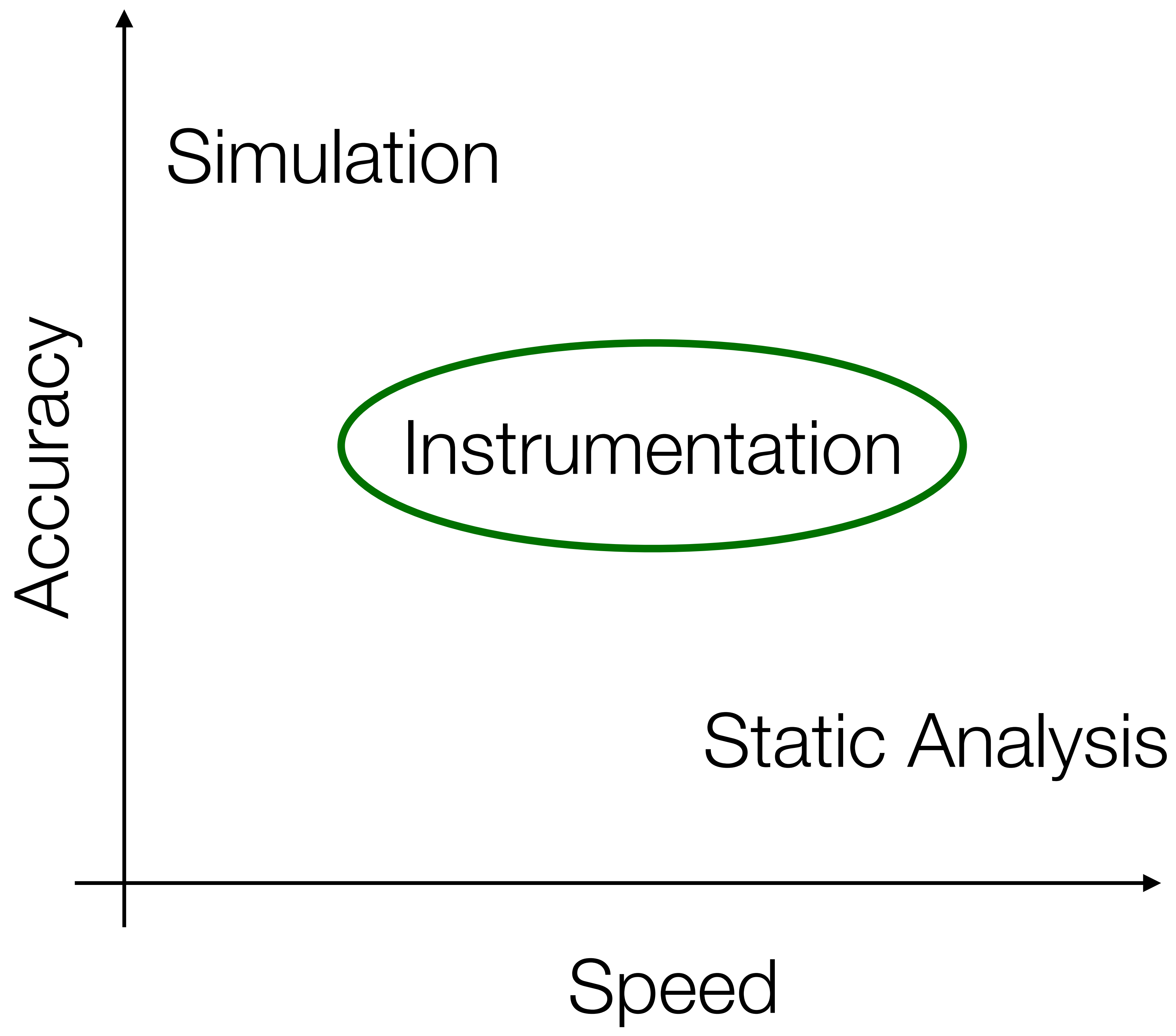
- Trace collection
- Analysis methodology
- Analysis results
- Summary/Outlook

Trace Collection



- Multiple options to extract memory accesses
- Instrumentation good compromise between speed and accuracy
- Host AND GPU are instrumented, connected via shared queue

Trace Collection



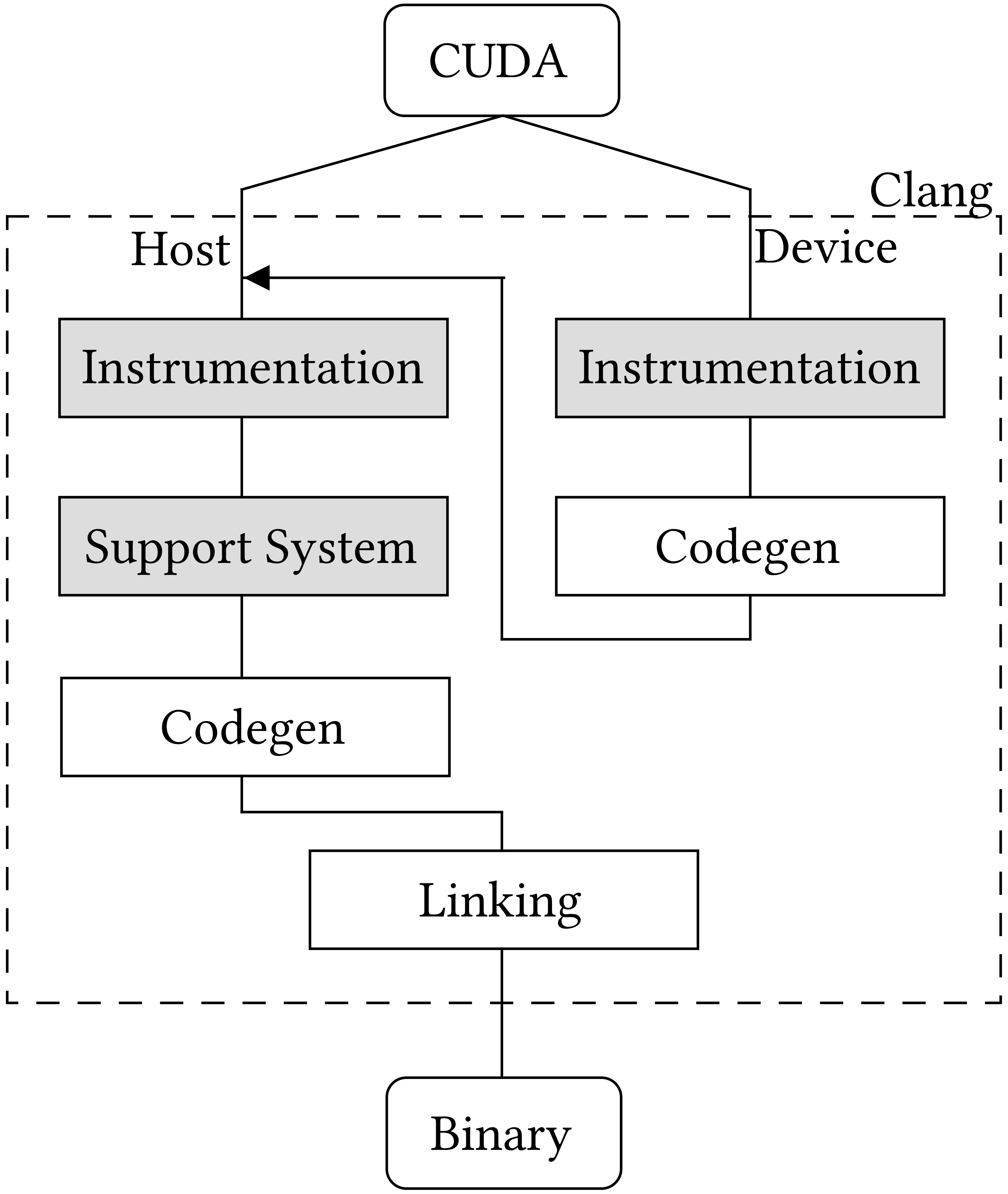
- Multiple options to extract memory accesses

- Instrumentation good compromise between speed and accuracy

- Host AND GPU are instrumented, connected via shared queue

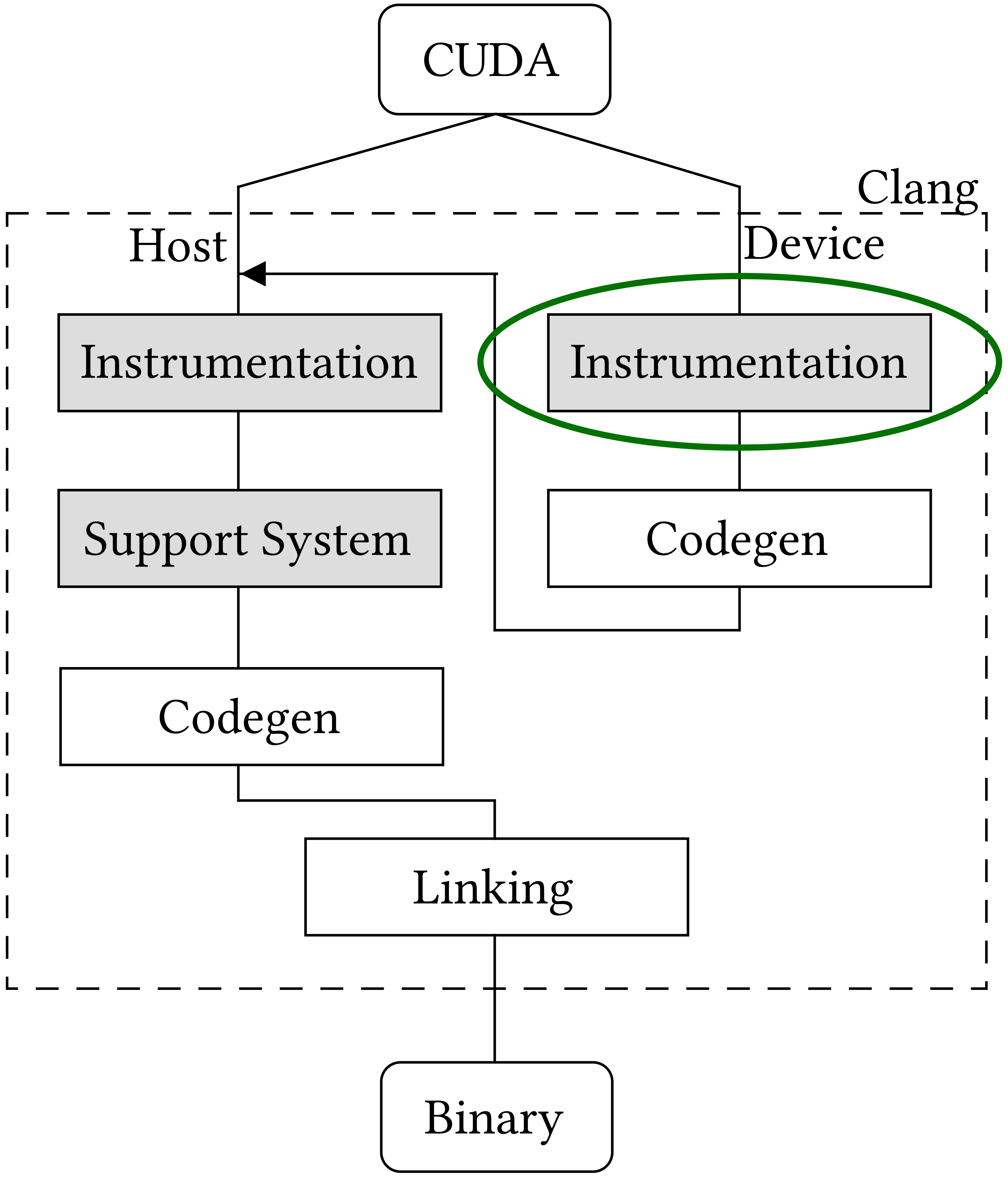
Trace Collection - Overview

- Instrumenting CUDA applications
- Open source plugin (Github) based on Clang
- Plugin consists of three parts
  - Device instrumentation
  - Host instrumentation
  - Shared queue



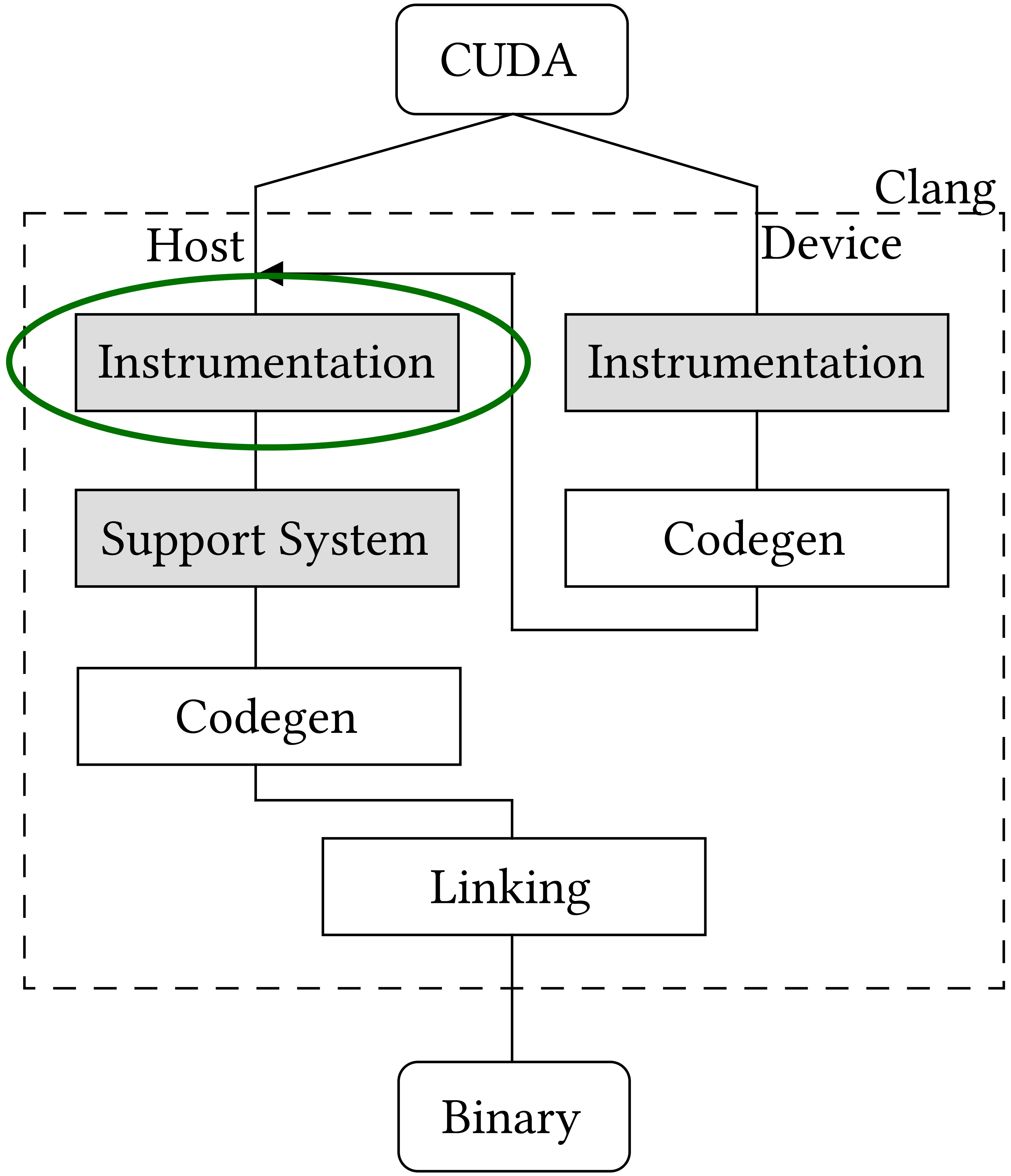
Trace Collection - Overview

- Instrumenting CUDA applications
- Open source plugin (Github) based on Clang
- Plugin consists of three parts
    - Device instrumentation
    - Host instrumentation
    - Shared queue



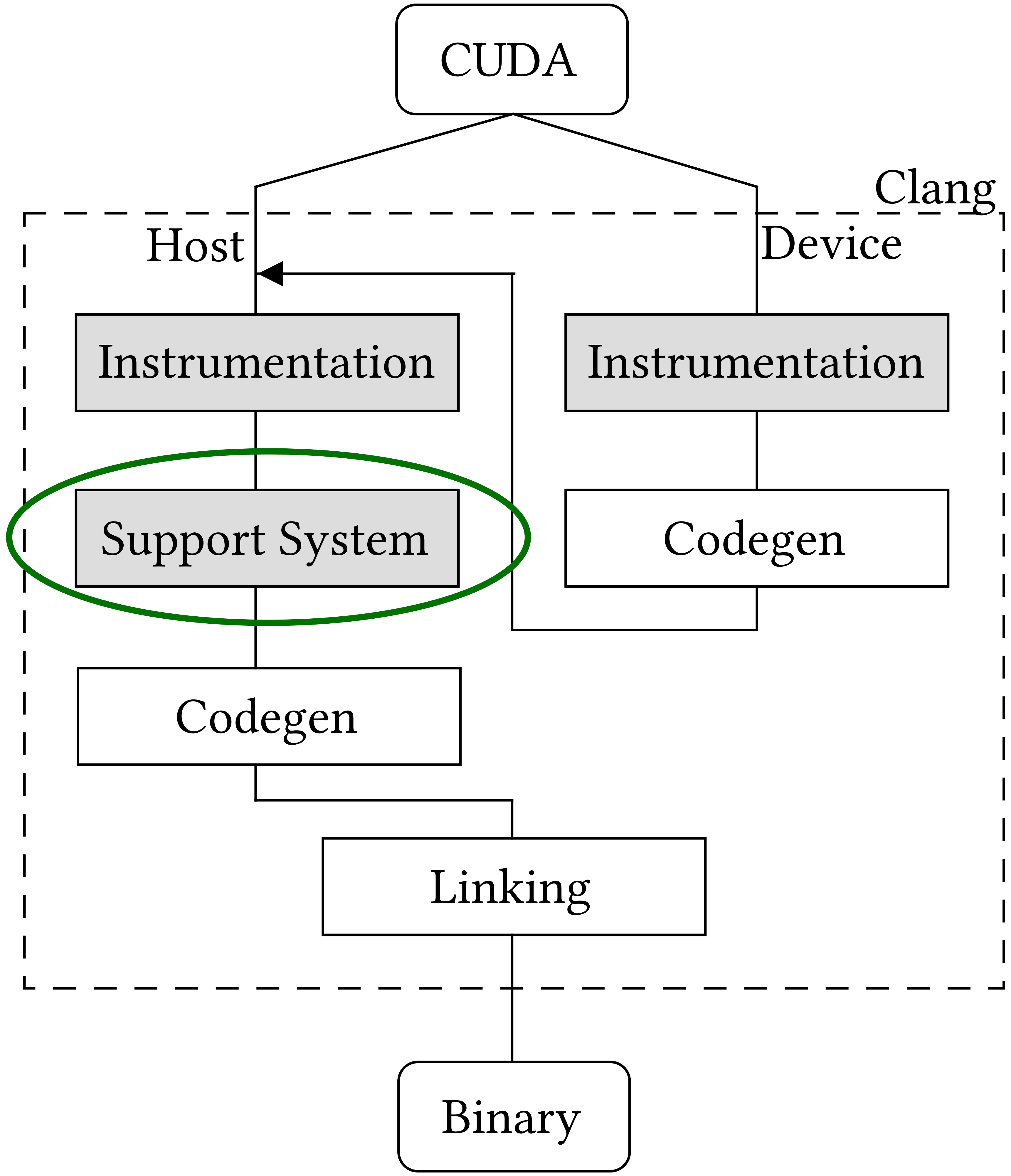
Trace Collection - Overview

- Instrumenting CUDA applications
- Open source plugin (Github) based on Clang
- Plugin consists of three parts
  - Device instrumentation
  - Host instrumentation
  - Shared queue



Trace Collection - Overview

- Instrumenting CUDA applications
- Open source plugin (Github) based on Clang
- Plugin consists of three parts
  - Device instrumentation
  - Host instrumentation
  - Shared queue



Trace Collection - Device

- Locate queue
- Locate global memory accesses
- Instrument memory accesses

```
__global__ K(float* arr) {
  __shared__ float shmem[512];
  shmem[address_a] = f();

  arr[address_b] = g();
}
```

→

```
__device__ uint8_t *queue;

__global__ K(float* arr) {
  __shared__ float shmem[512];
  shmem[address_a] = f();
  trace(queue, address_b, sizeof(float));
  arr[address_b] = g();
}
```

Trace Collection - Host

- Locate kernel launch
- Prepare queue
- Queue consumer thread management

```
K <<<..., stream>>> (...);
```

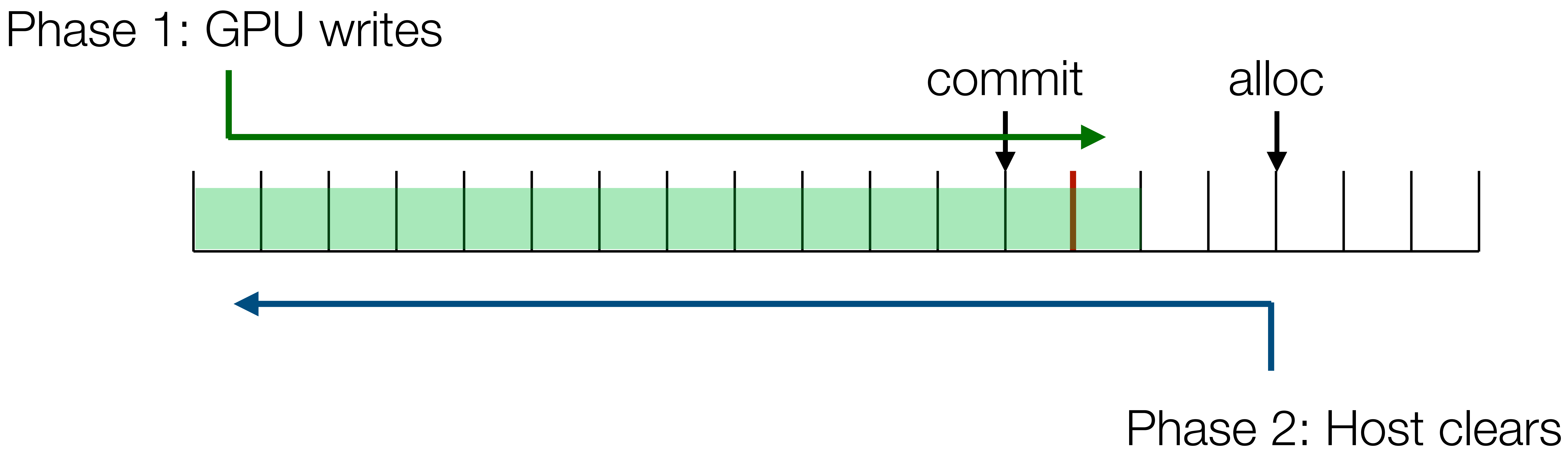
→

```
prepare_queue (stream, "K");
cudaStreamAddCallback (stream, start_q, "K");
K <<<..., stream>>> (...);
cudaStreamAddCallback (stream, stop_q, NULL);
```

Trace Collection - Queue

Phase 1 - GPU: writes to queue until alloc hits watermark
Host: wait for commit to hit watermark

Phase 2 - Host: read data and reset commit and alloc
GPU: wait for alloc to reset below watermark

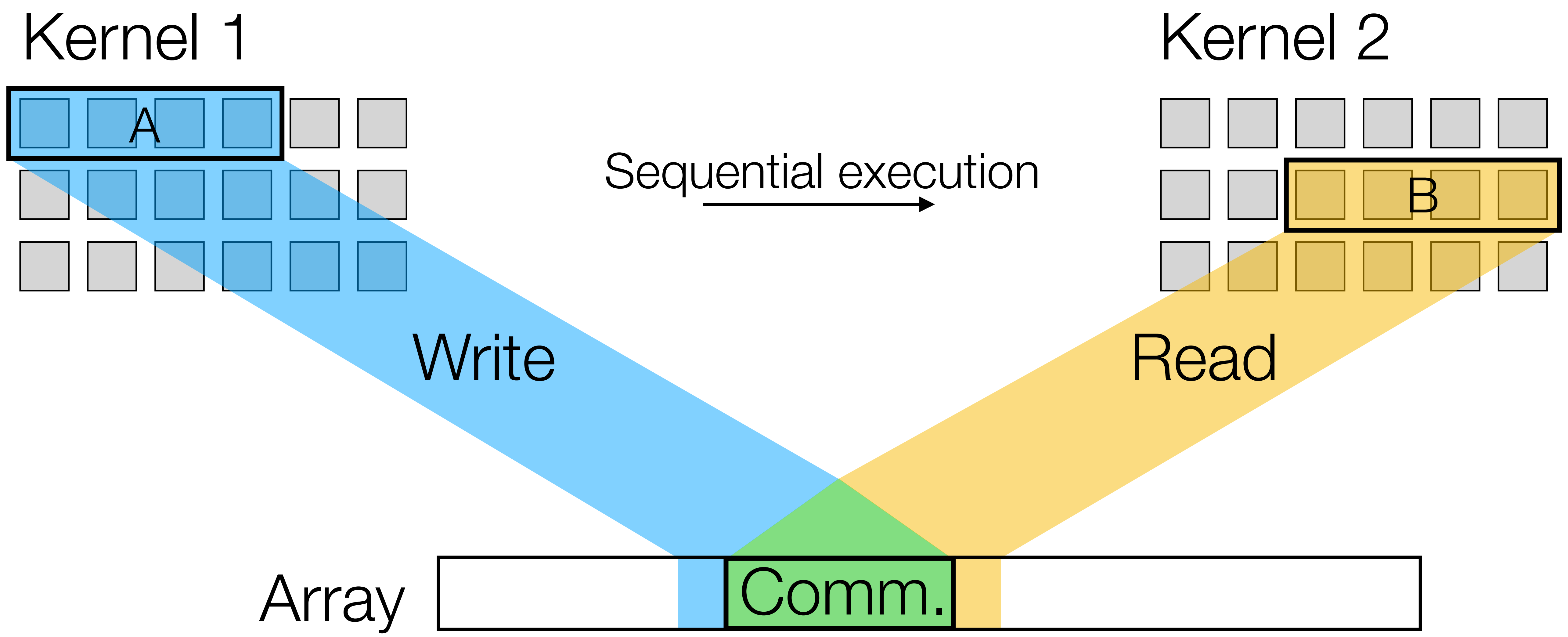


Methodology - Communication

- Intersection between writes of thread block set A and reads of thread block set B

- Kernel of A must precede kernel of B (relaxed consistency)

- Thread block sets can have arbitrary shape → partition = thread block set



Methodology - Partitioning

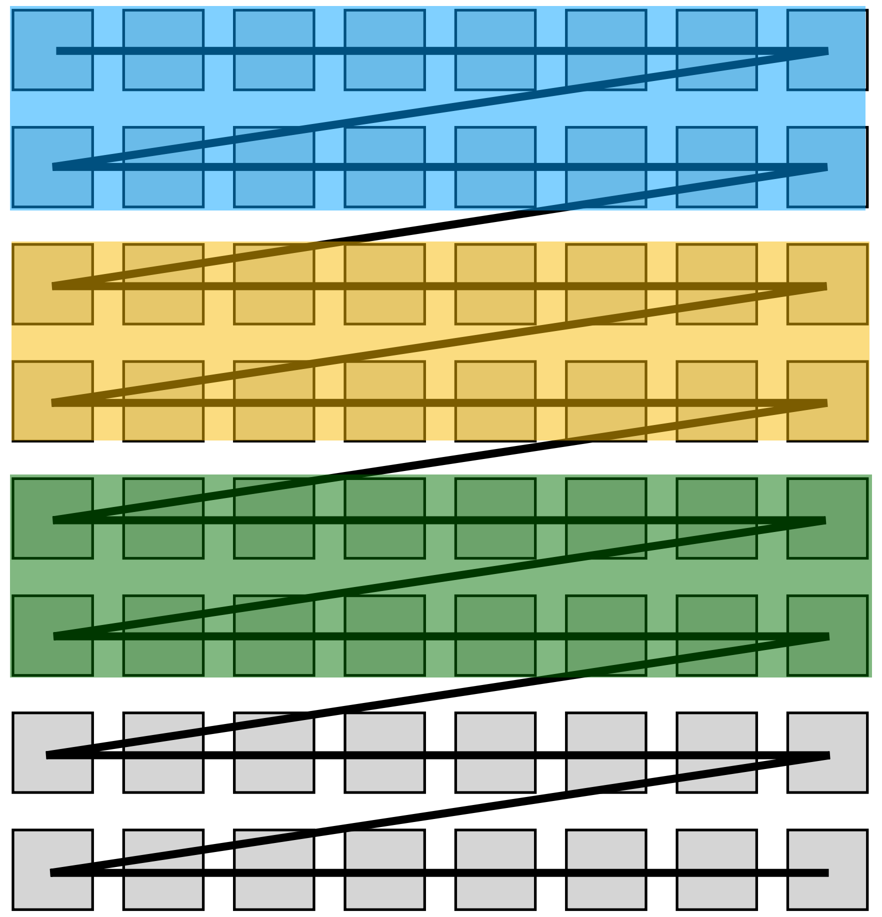
Wanted: (x, y, z) → {0, ..., n-1}
Mapping from Thread Block IDs to partition

- Consistent results for different thread grid sizes and dimensionalities
- Spatial locality of thread blocks' accesses → difficult
- Spatial locality of thread blocks' IDs → feasible

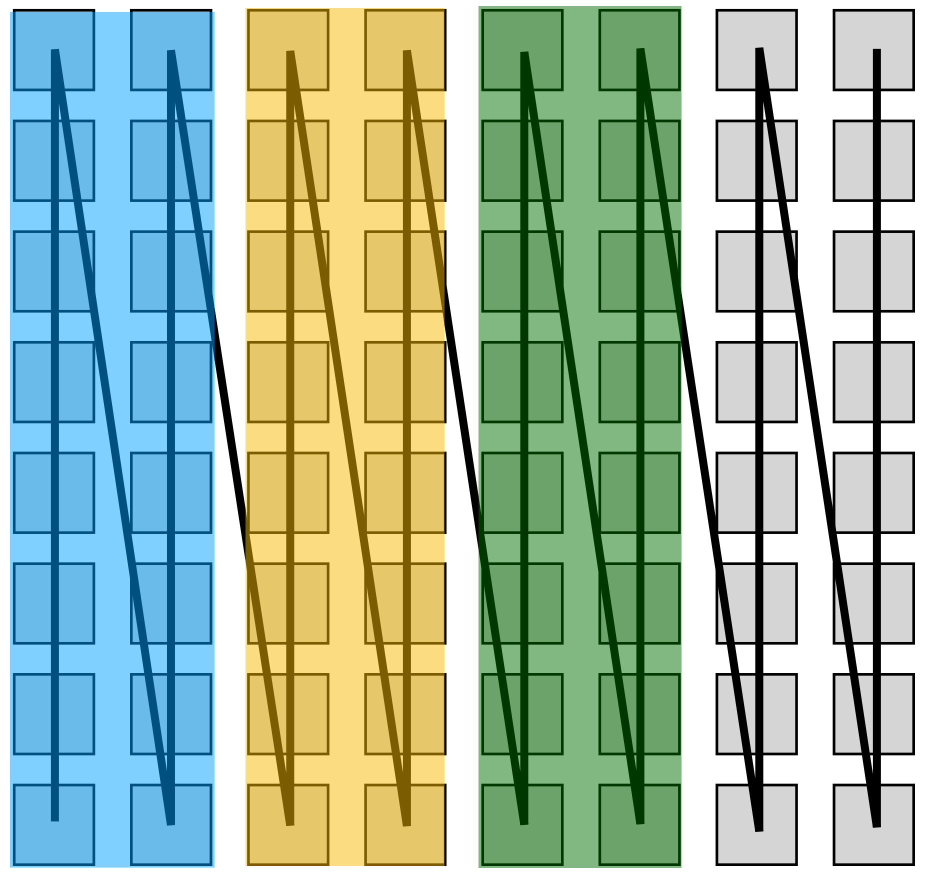
Methodology - Partitioning

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

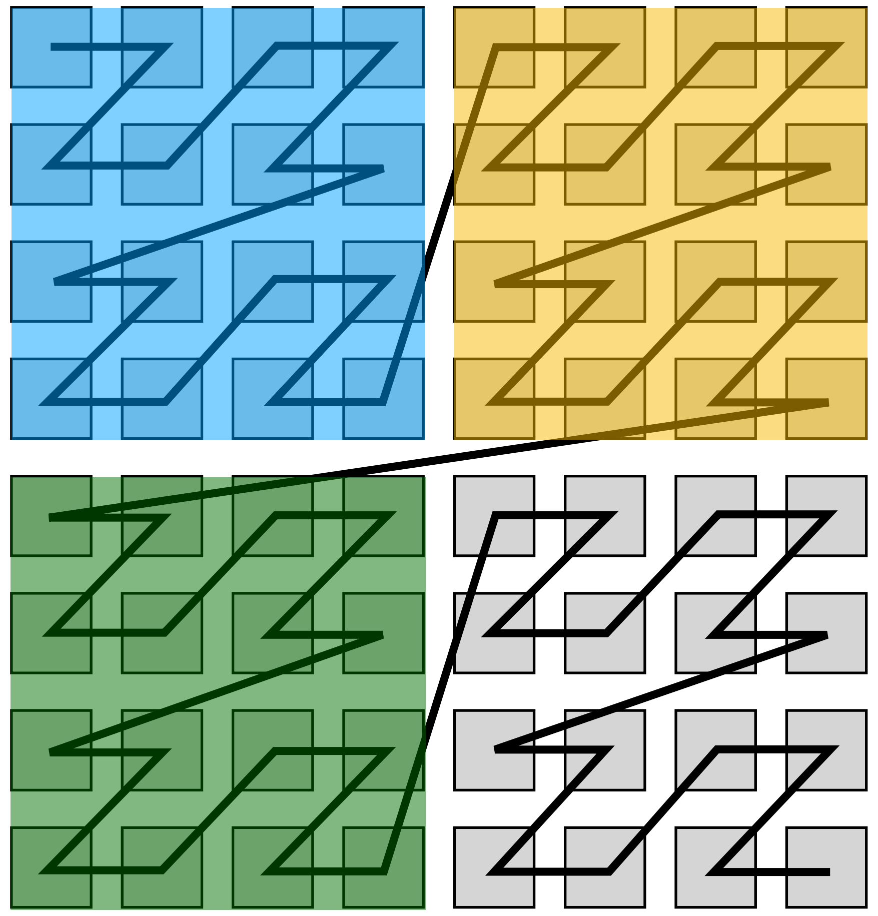
Lexicographic "row-major order"



Colexicographic "column-major order"



Z-Order curve Recursive Zs



Partitions	1	2	3	4

Results - Workloads

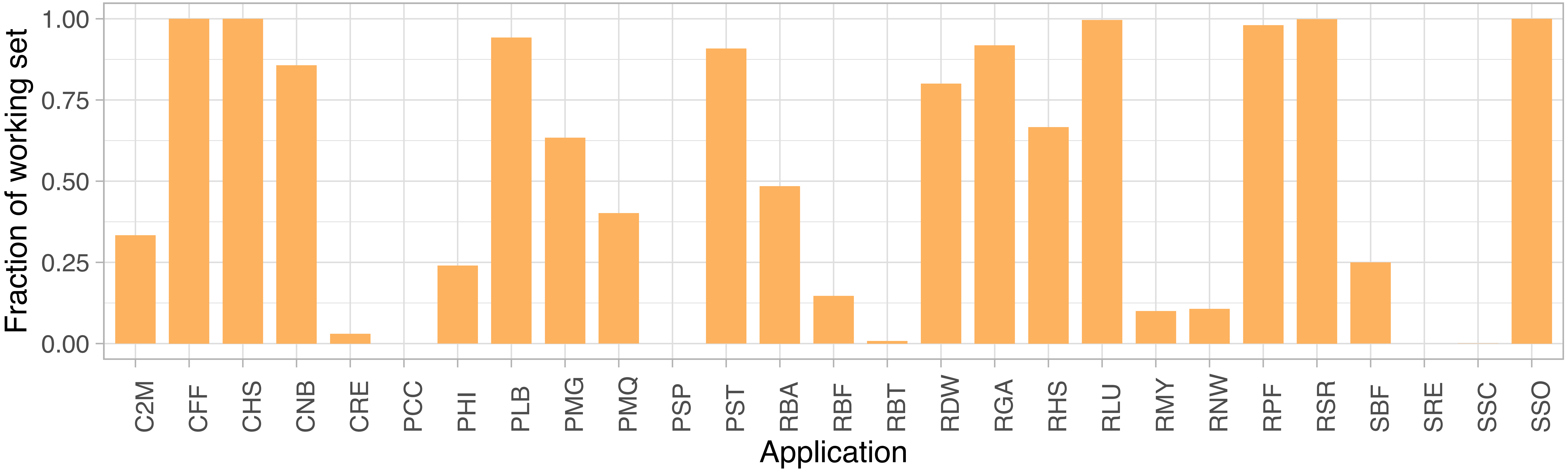
- Workloads
  - Parboil - 7/11
  - Rodinia - 11/23
  - SHOC - 4/9
  - Custom applications - 5/5

- Reasons for exclusion
  - Unsupported features
  - Tracing incompatibilities
  - Excessive trace size
  - Single Kernel launch

Results - Data Origin

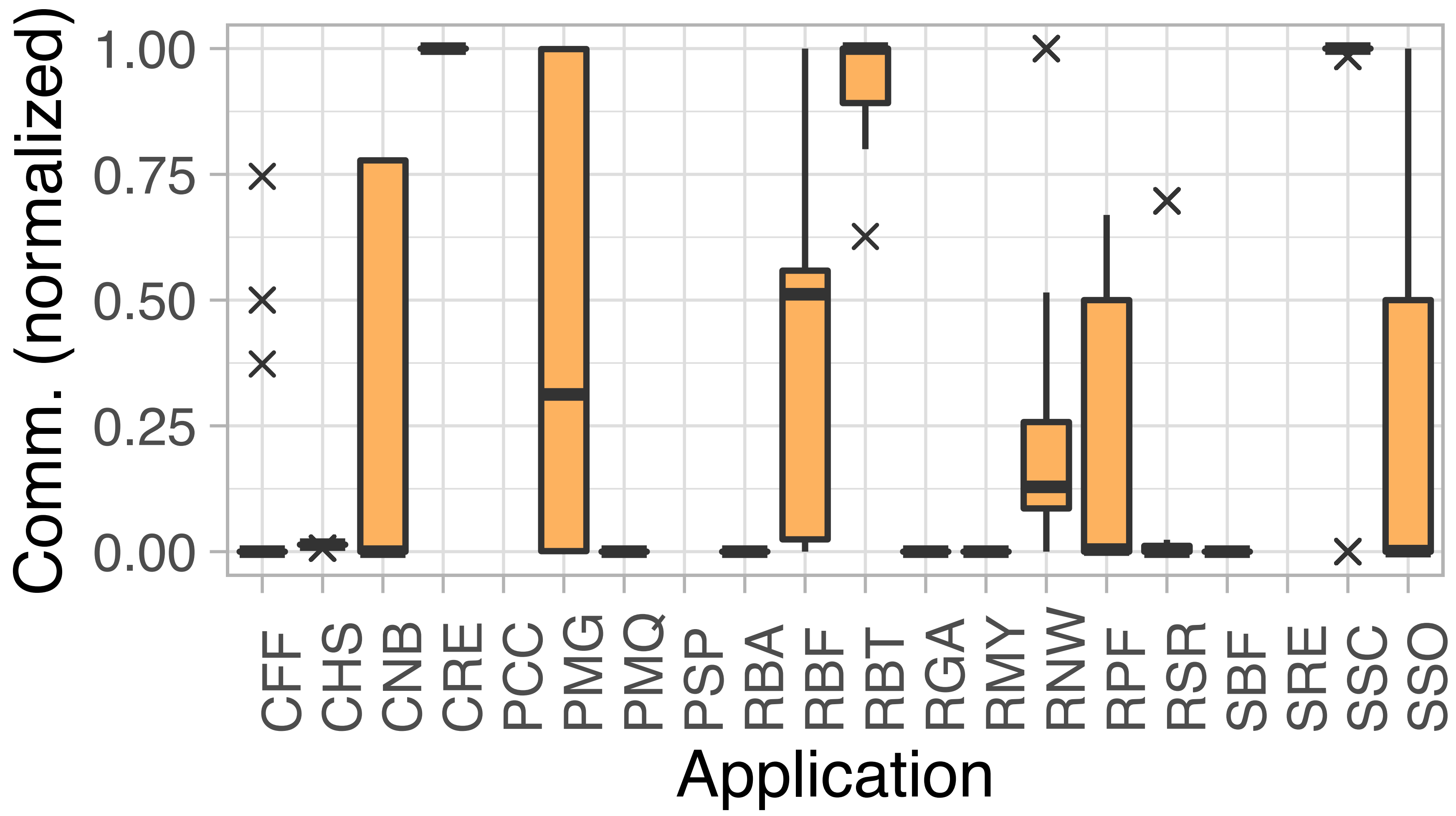
- Data read **from the GPU** at least once, normalized to full working set

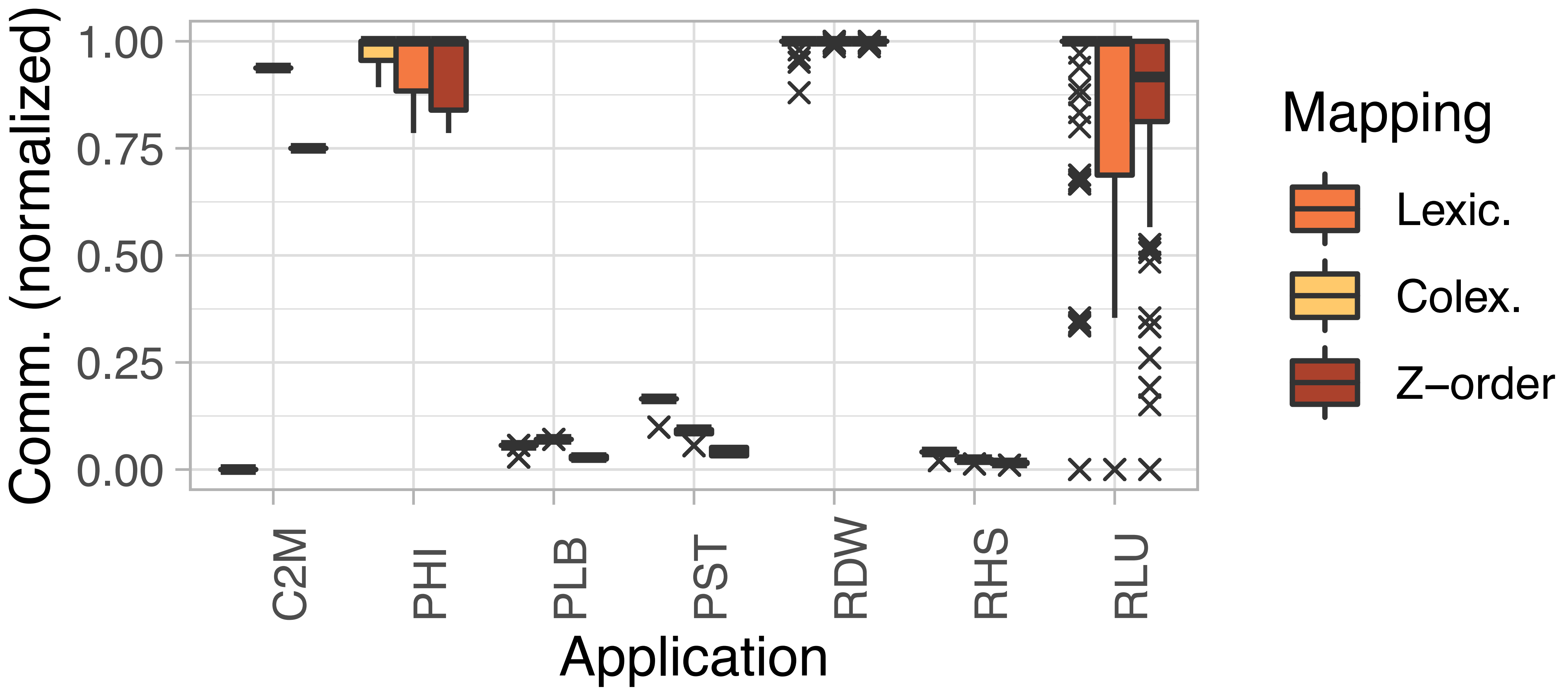
- More than 50% for 13 / 27



Results - Remote Loads

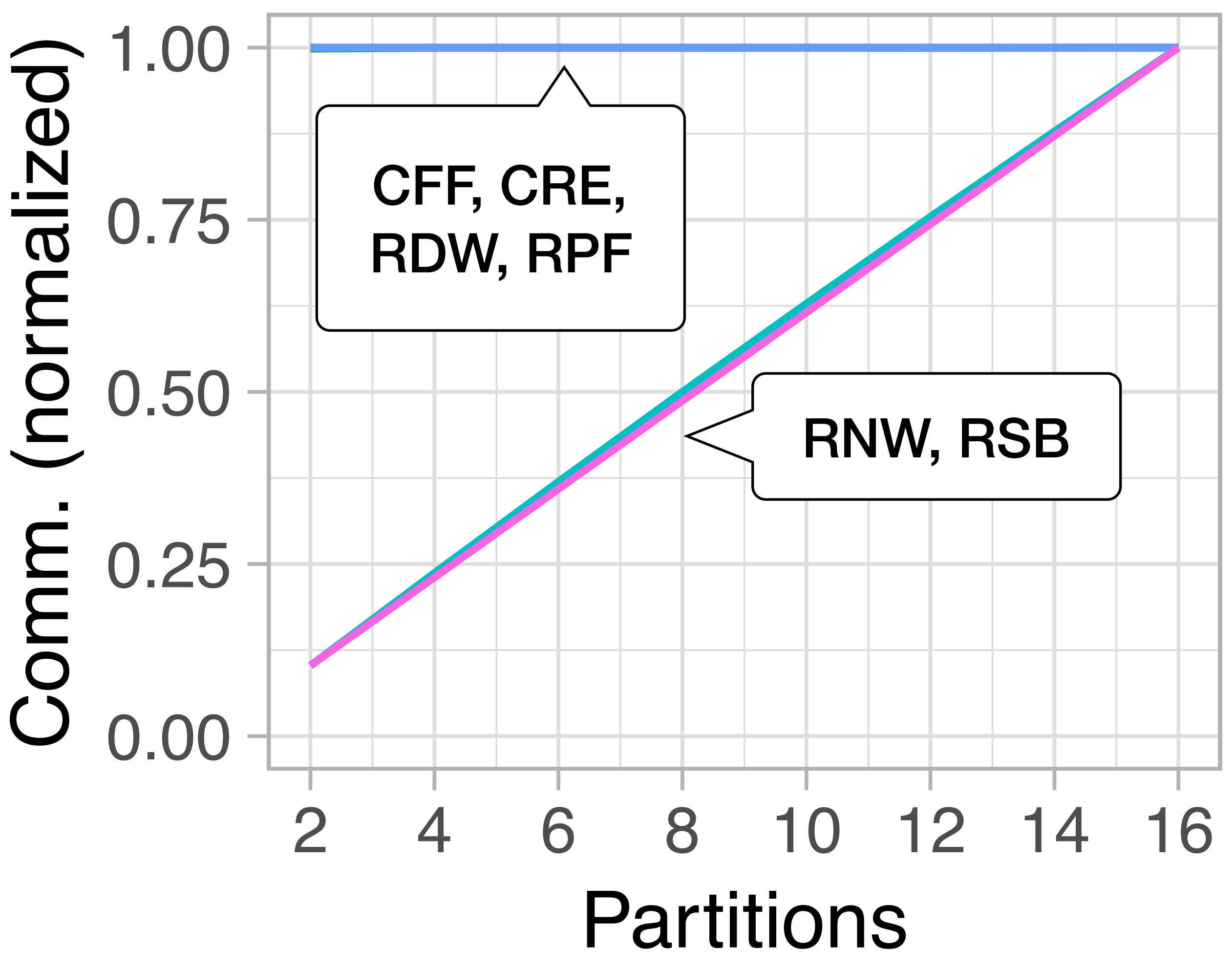
- Data read **from other partitions**, normalized to full working set (16 partitions)
- Identical for most applications (one-dimensional kernels)
- Otherweise Z-Order has best median for all but two application

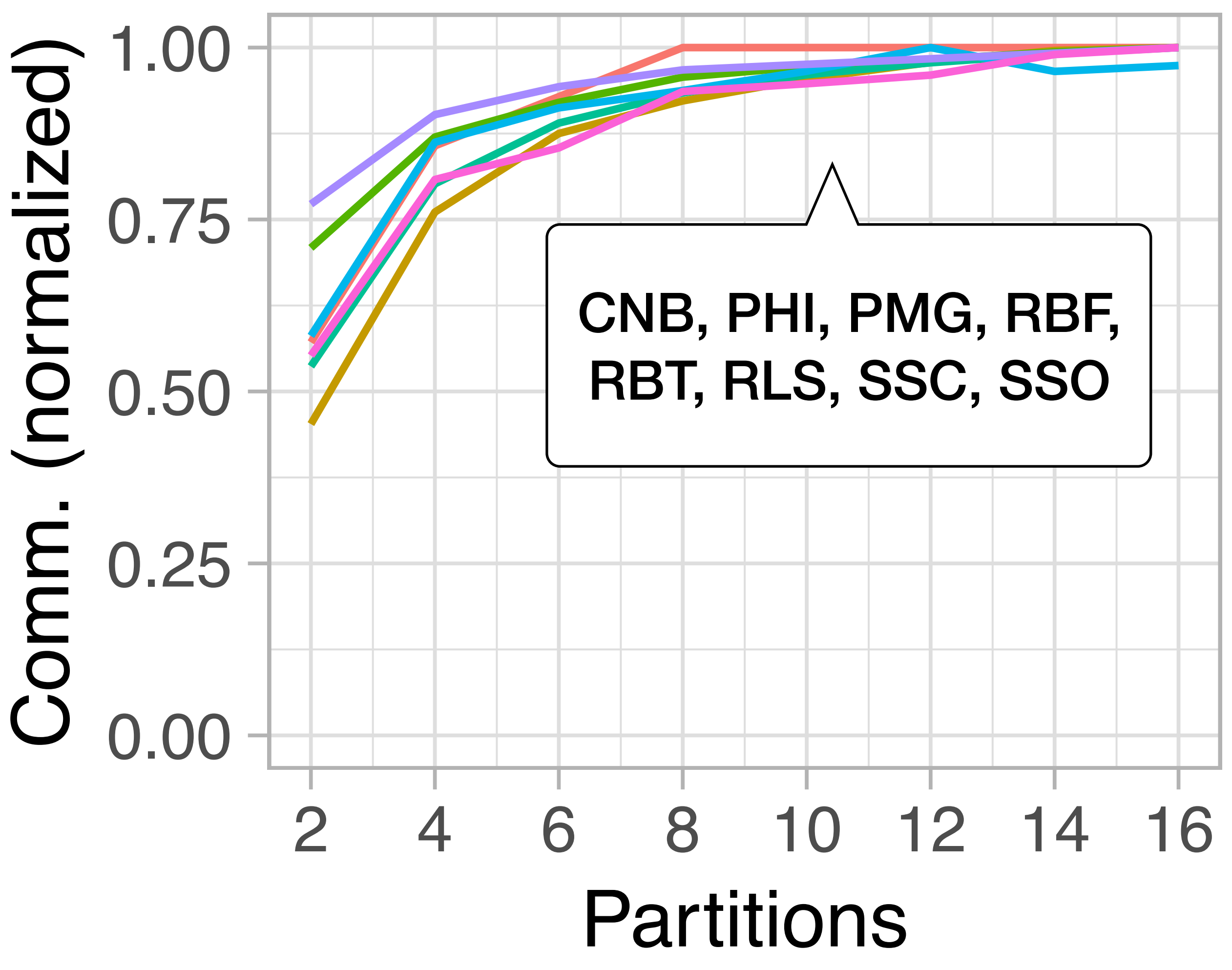


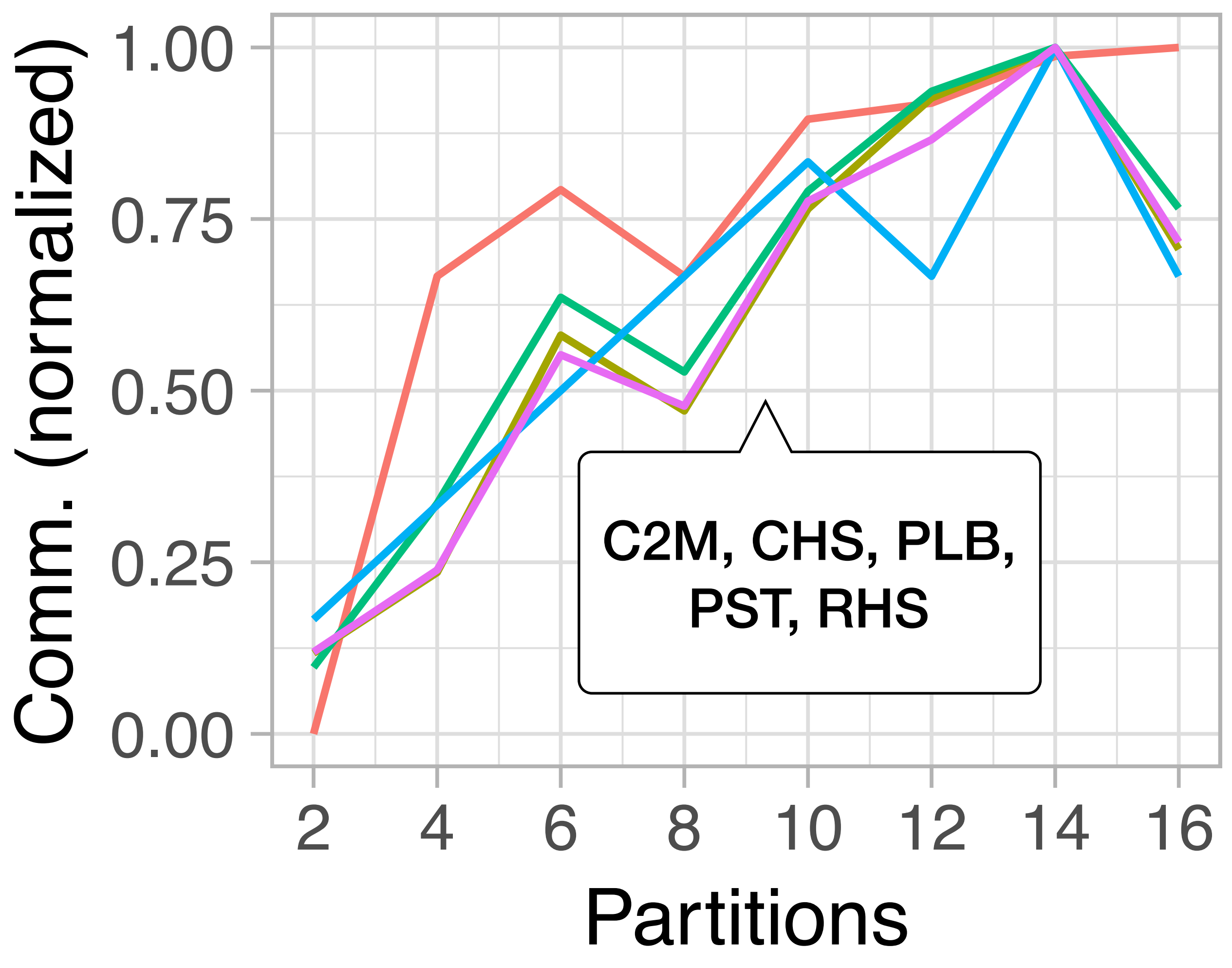


Results - Scaling

- **Summed communication** between partitions, normalized to full working set
- Z-Order partitioning only, 16 partitions
- Sensitivity for thread distribution varies significantly







Summary & Outlook

- Open Source CUDA instrumentation for memory accesses
- Analysis methodology works → more partitioning mappings needed
- High diversity throughout applications → analysis reveals expected traffic

Time for Questions

Thank you for your attention