MUST

NCCLSanitizer: Runtime Correctness Checking Stream-based Communication in NCCL Programs

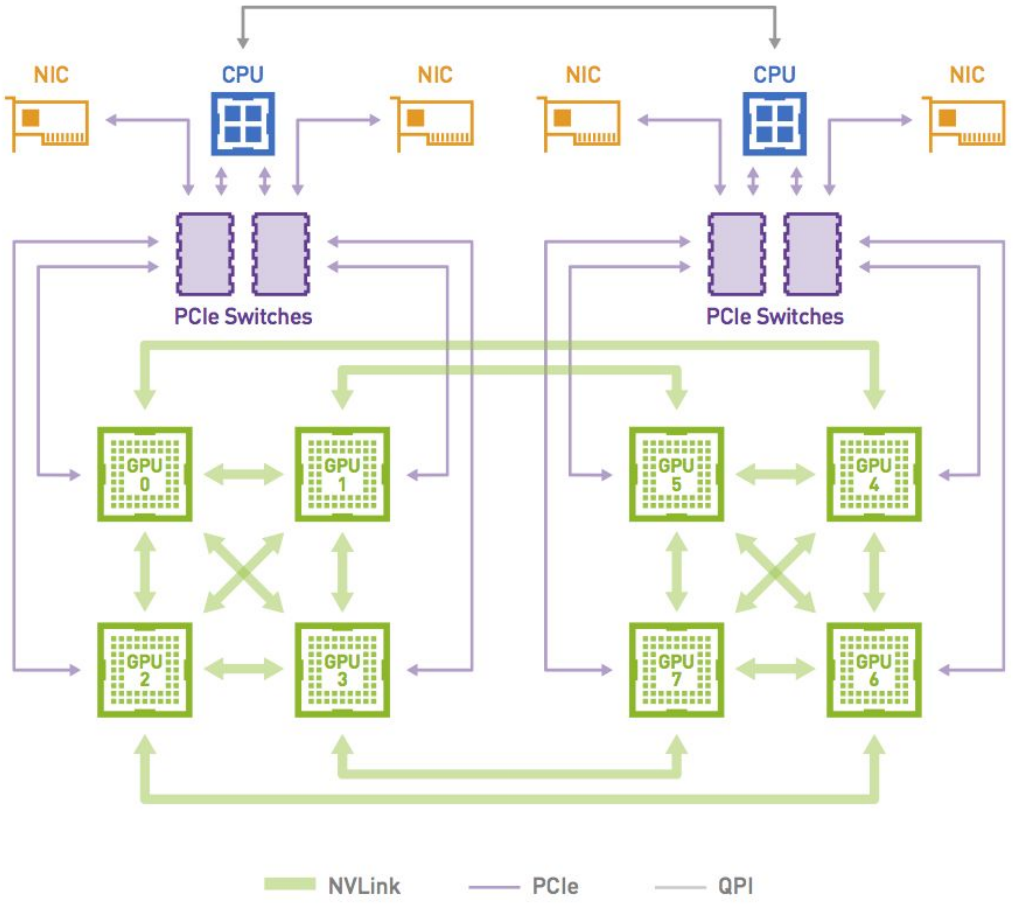
Felix Tomski (tomski@itc.rwth-aachen.de)
Joachim Jenke
Simon Schwitanski

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

NCCL: NVIDIA Collective Communication Library

NCCL

- NVIDIA's collective communication library for **GPU-GPU communication**
- Topology awareness offers high throughput and low latency
- Comparable libraries by other vendors exist (e.g. AMD's RCCL, Intel's oneCCL)
- Predominantly used in ML frameworks (e.g. PyTorch, TensorFlow)
- Rarely in classical HPC applications and libraries (e.g. Gkeyll, PyLops)



[1]

Why a correctness tool for NCCL programs?

- MPI Forum is working on stream extension for MPI standard
  - Multiple prototypes exist that are similar to NCCL's API
- Explore implications of **stream-based communication** semantics for correctness tools
- Assist application developers using NCCL

Stream-based communication in NCCL

```
cudaStream_t stream;
cudaStreamCreate(&stream);
float *d_buf;
cudaMalloc(&d_buf, count);
kernel<<<stream>>>(d_buf);


ncclAllreduce(d_buf, …, stream);
cudaStreamSynchronize(stream);
```

NCCL allows

- Sending from / receiving to device buffer (as GPU-aware MPI)
- Associating communication with device context (stream)
  - Fine-grained control over synchronization between device computation and communication

```
cudaStream_t stream;
cudaStreamCreate(&stream);
float *d_buf;
cudaMalloc(&d_buf, count);
kernel<<<stream>>>(d_buf);
cudaStreamSynchronize(stream);
MPI_Allreduce(d_buf, …); // GPU-aware
```

- Communicator management similar to MPI
  - Each rank in communicator associated with GPU
- Requires another distributed-memory communication protocol for initialization
  - We consider NCCL+MPI programs

Data races in NCCL programs

Data race
- Two execution units
- Unsynchronized access to same memory
- At least one is modifying (a write)

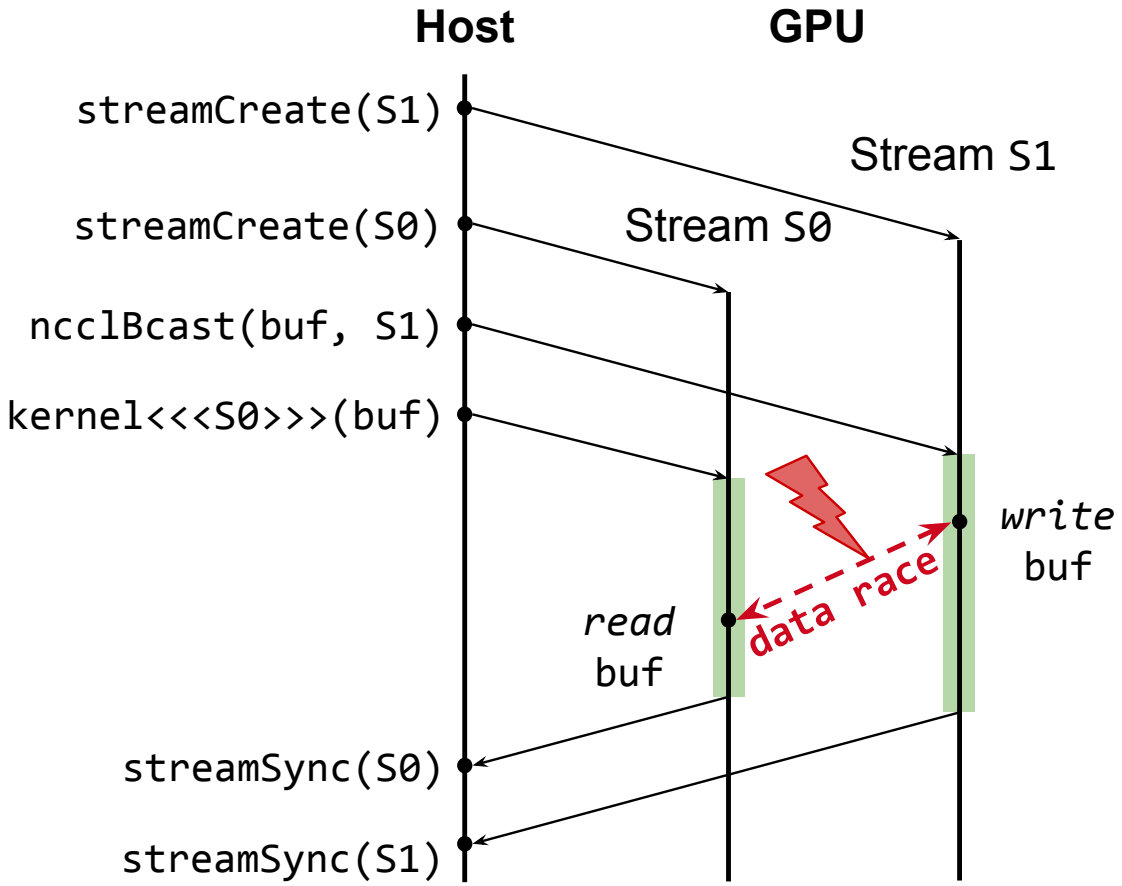
Data races lead to undefined behavior (e.g. crash, corrupted data)

CUDA streams
- May run concurrently
- Require host-initiated synchronization

NCCL calls
- Nonblocking on the host
- Blocking on the device (stream)

Conflict is always between two kernels, usually on **two unsynchronized streams**
- Kernels may result from: NCCL, MPI, user kernel

Special cases
- Default CUDA stream entails implicit synchronization with other streams
- NCCL groups may lead to races within same stream
  - Similar to nonblocking MPI communication within a stream



Deadlocks in NCCL programs

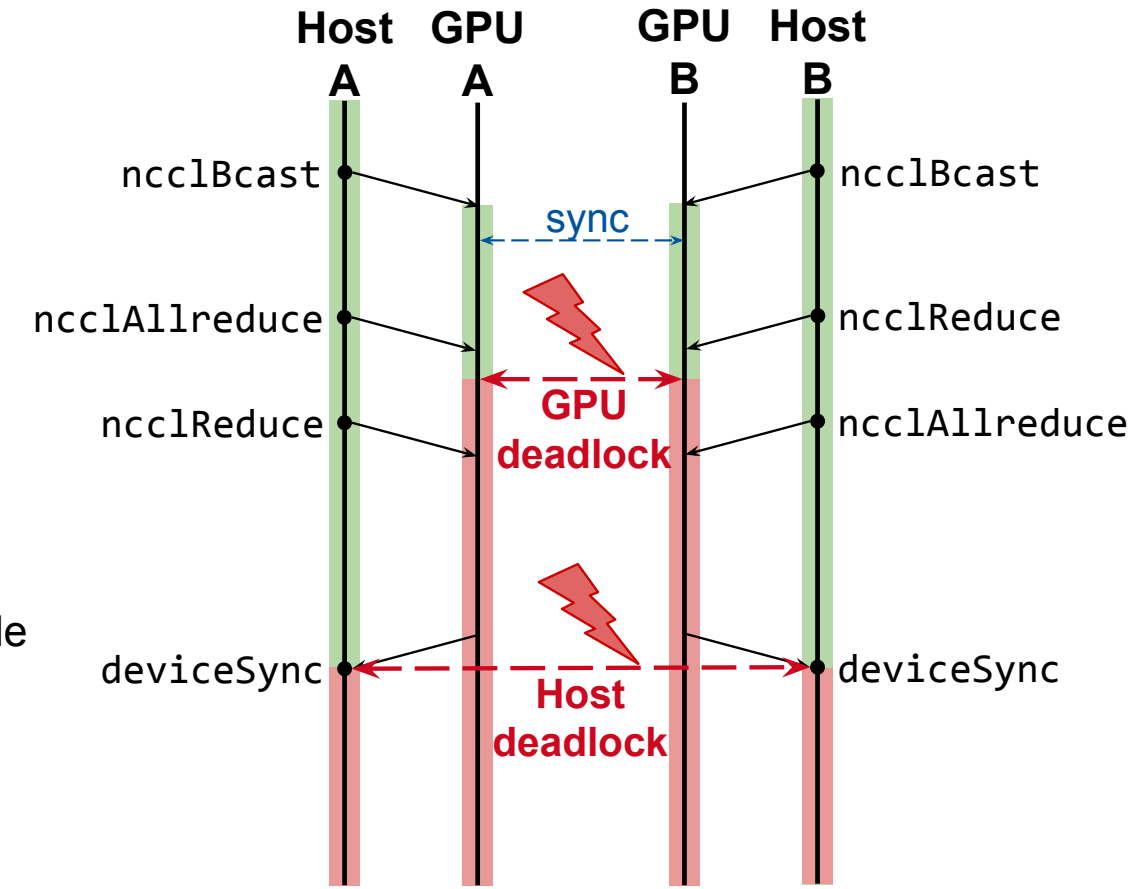
Deadlock
- Execution units are in a circular dependency
- Execution gets stuck

Deadlock may occur if collectives mismatch w.r.t.
- kind (see example)
- datatype
- root rank

NCCL **deadlocks first manifest on GPU**
- **Host blocks on synchronizing with GPU**
  - Similar to MPI blocking completion

Hybrid deadlocks through interleavings of MPI and NCCL are possible
- Host may be blocked due to
  - synchronization with stream that is blocked
  - (non-local, blocking) MPI call
- GPU (stream) may be blocked due to
  - (non-local, blocking) NCCL communication kernel

P2P communication may be part of circular dependency

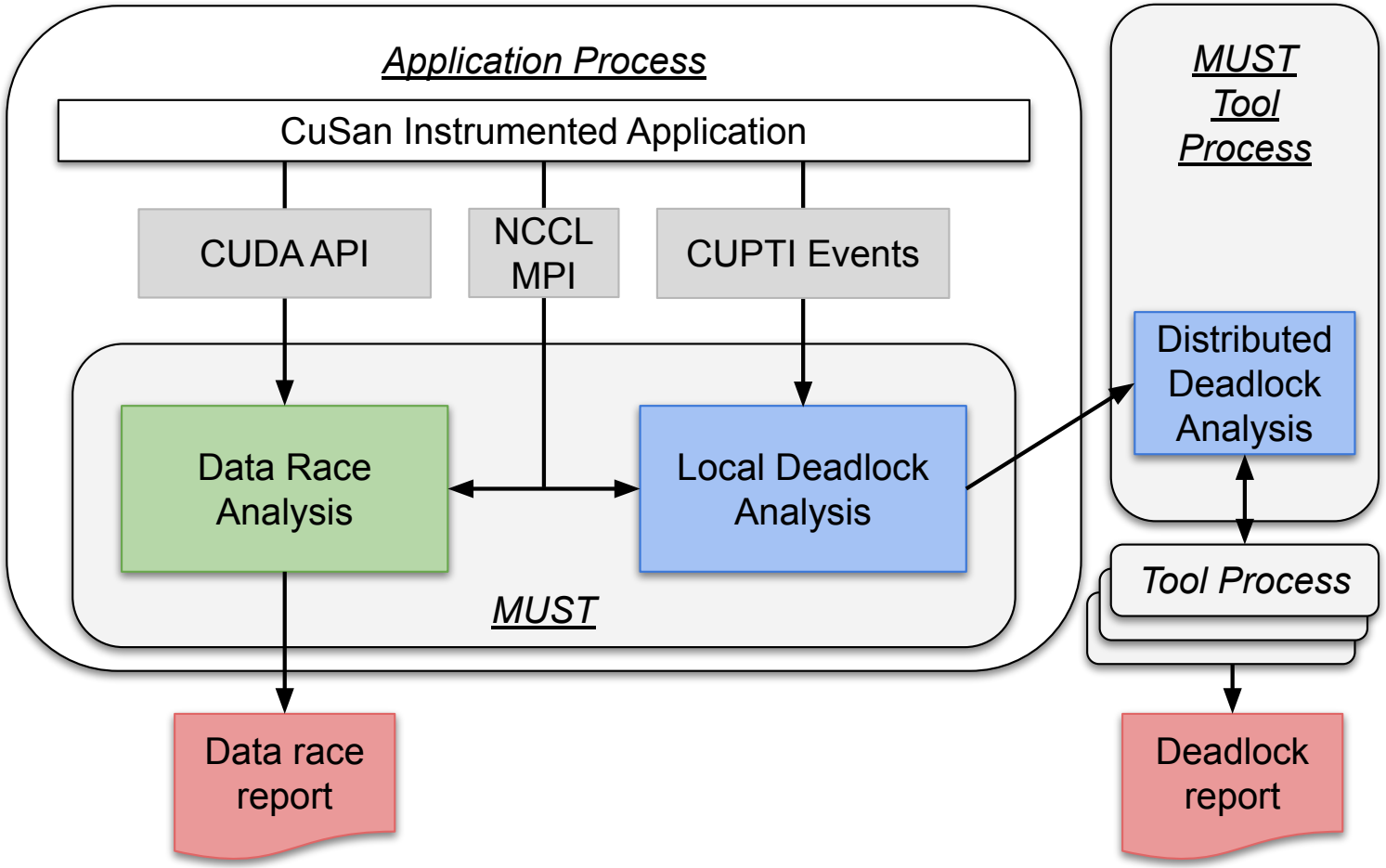


NCCLSanitizer

- Compiler-aided runtime correctness tool for NCCL+MPI programs
- Builds upon MPI correctness tool MUST and MPI+CUDA data race detector CuSan

- **MUST**
  - Deadlocks in MPI programs
  - Data races in nonblocking MPI communication and MPI+OpenMP programs

- **CuSan**
  - Data races in GPU-aware MPI communication



NCCLSanitizer: Detecting data races

Requirements

- Track synchronization and memory accesses
  - Accesses come from NCCL calls and CUDA kernels
  - Synchronization from CUDA API calls
  - Stream activation/switches

CuSan compiler wrapper

- Instrument memory accesses in kernels
- Instrument stream synchronization and switches

CuSan runtime

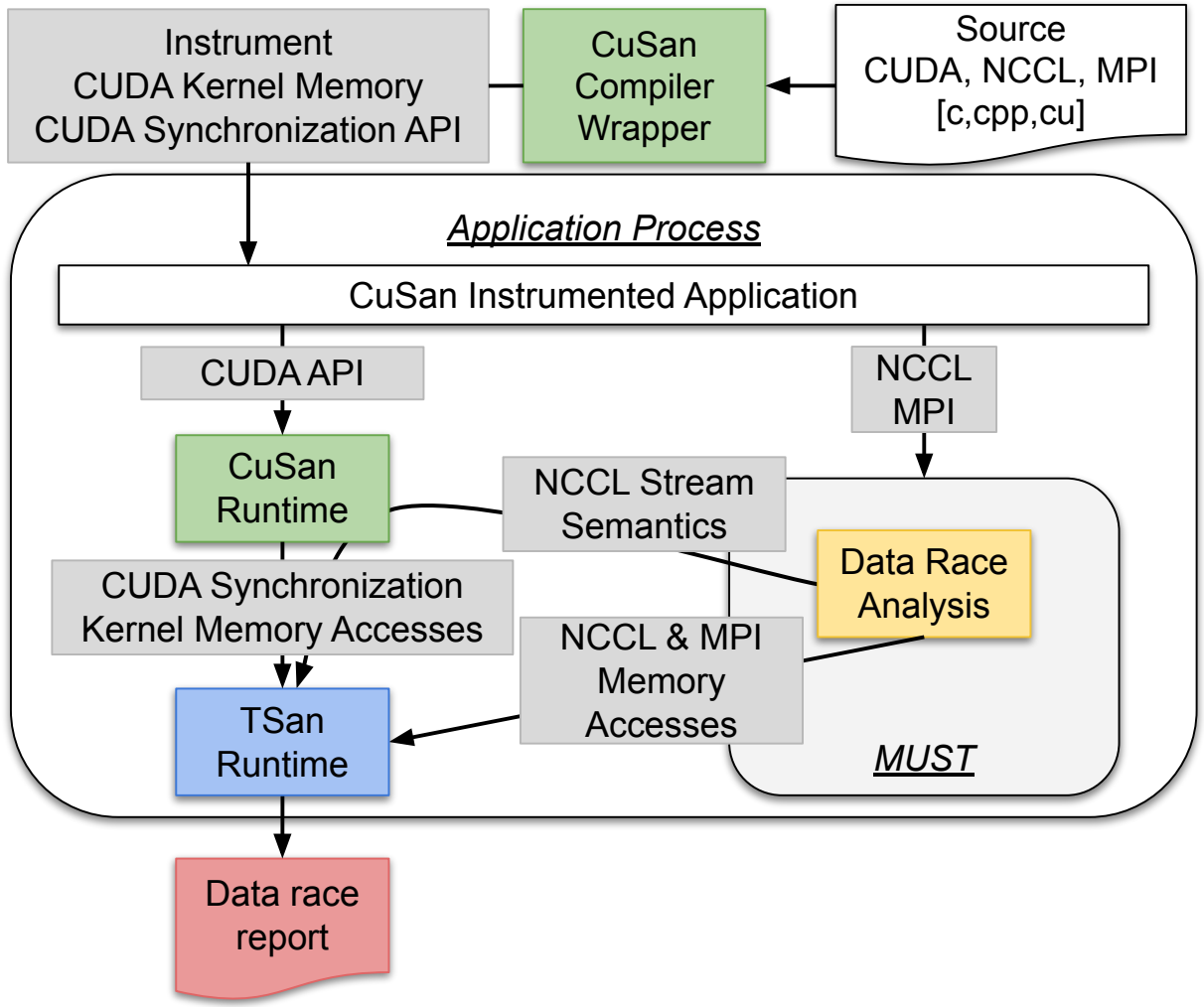
- Propagate kernel memory accesses and stream synchronization+switches to TSan runtime

MUST

- Propagate NCCL memory accesses to TSan runtime
- Propagate NCCL stream switches to CuSan

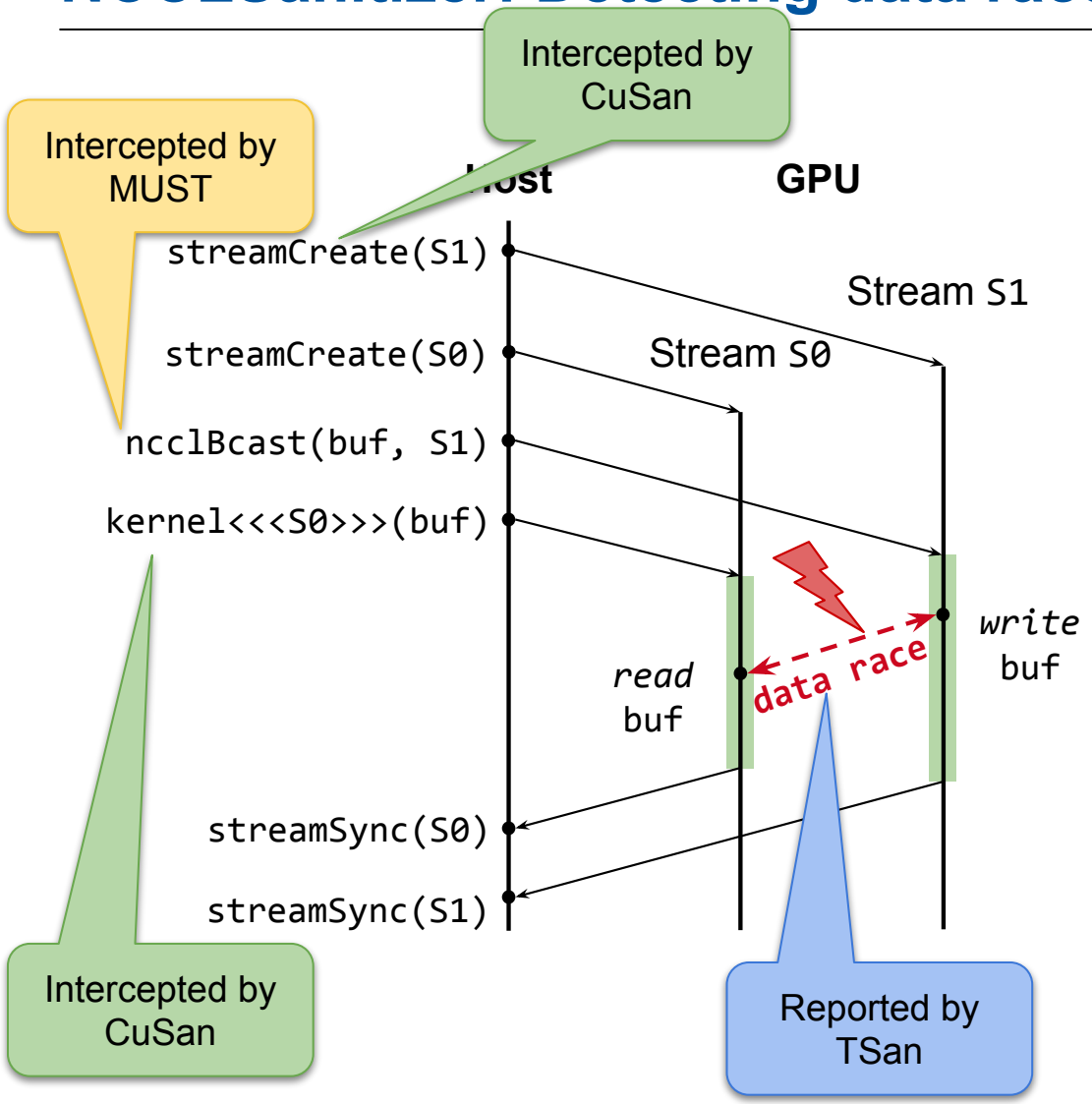
ThreadSanitizer (TSan) runtime

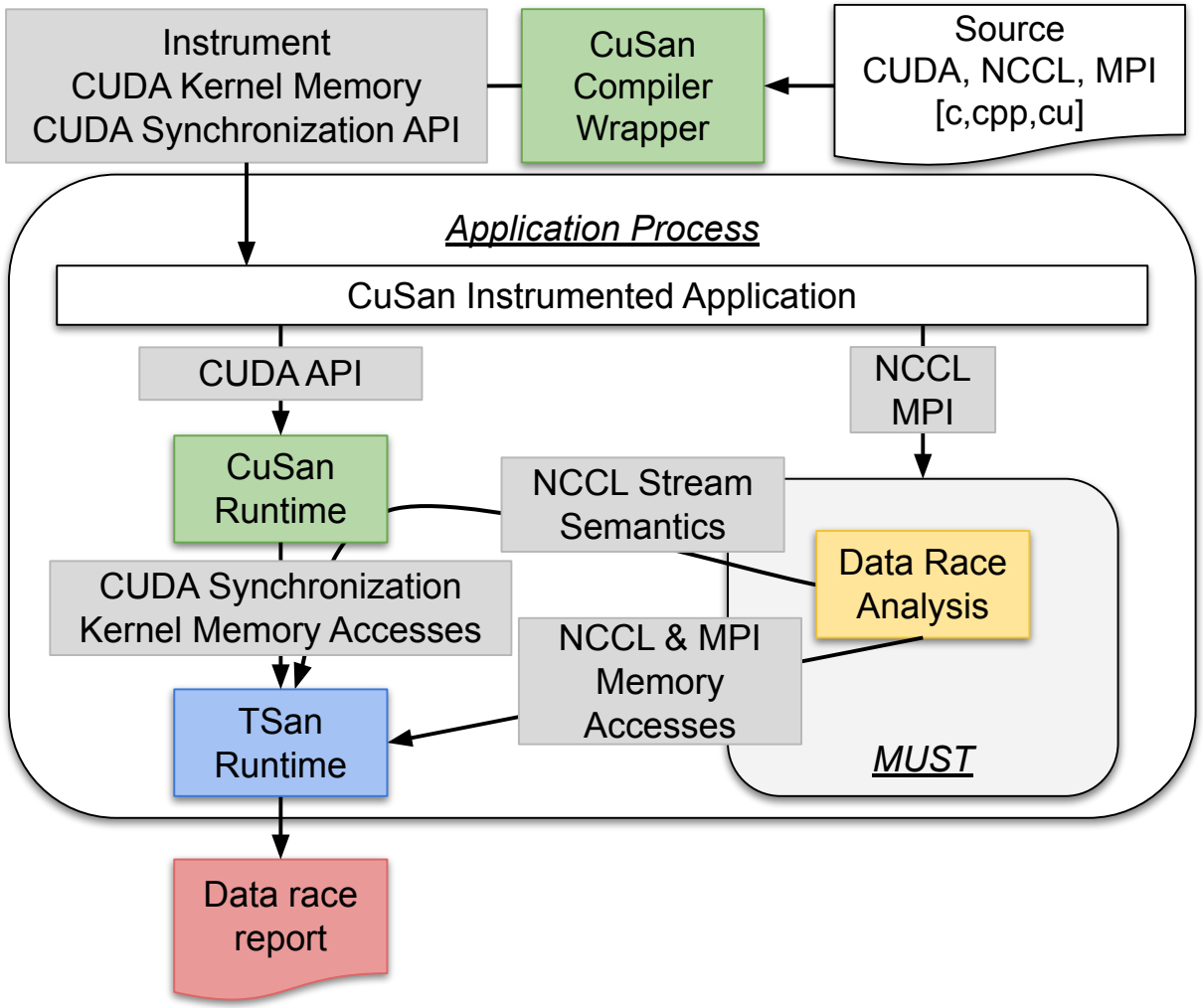
- Perform actual data race detection



NHR4CES NHR for Computational Engineering Science
Performance Optimisation and Productivity
A Centre of Excellence in HPC
i12 High Performance Computing
RWTH AACHEN UNIVERSITY

NCCLSanitizer: Detecting data races - Example





NCCLSanitizer: Detecting deadlocks

Requirements

- Track blocking state of host and GPU streams
  - Host blocked from
    - MPI calls
    - Synchronization with GPU
  - GPU stream blocked from
    - NCCL calls

CUPTI adapter

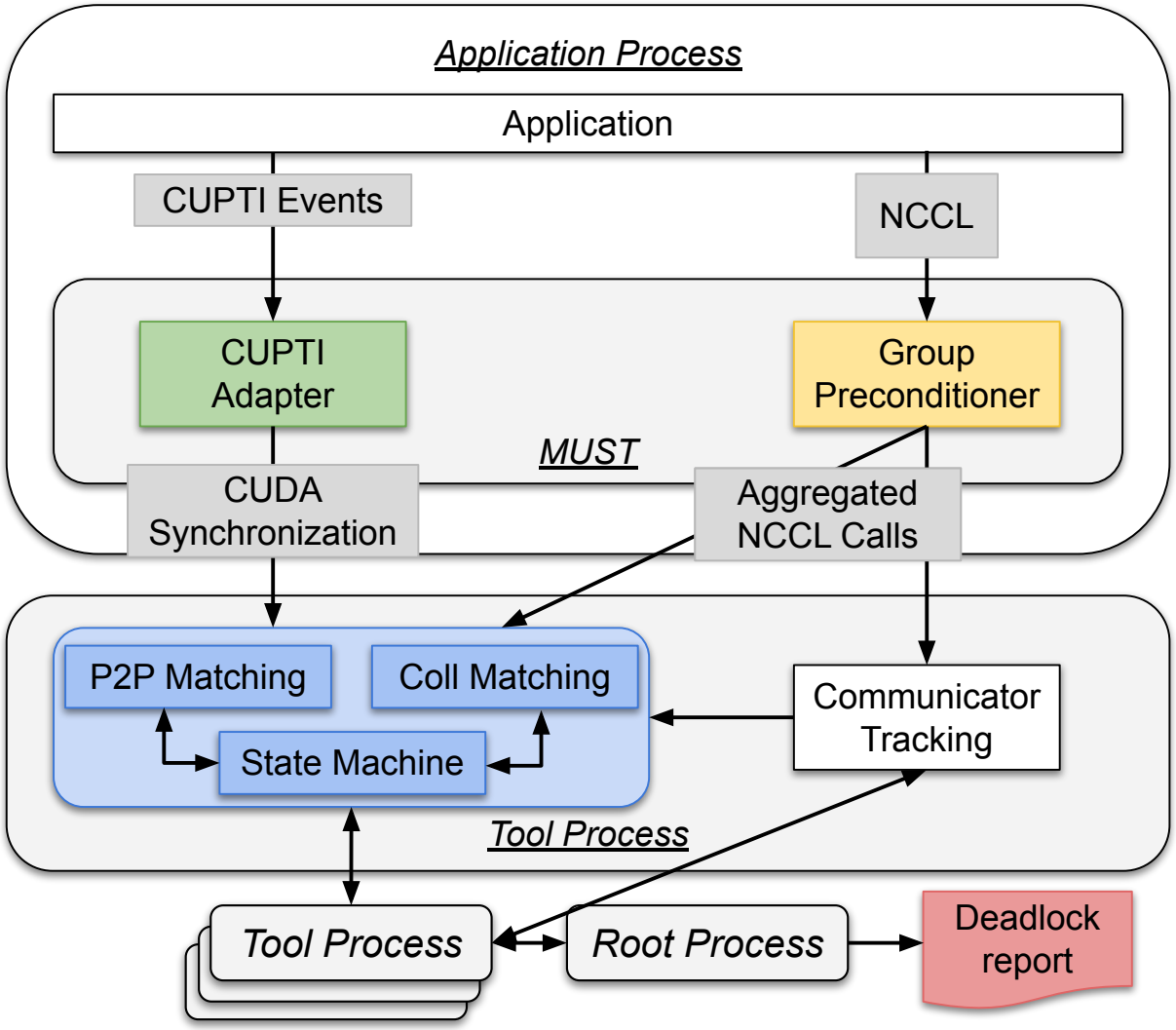
- Track host-GPU synchronization, per
  - device
  - stream

Group preconditioner

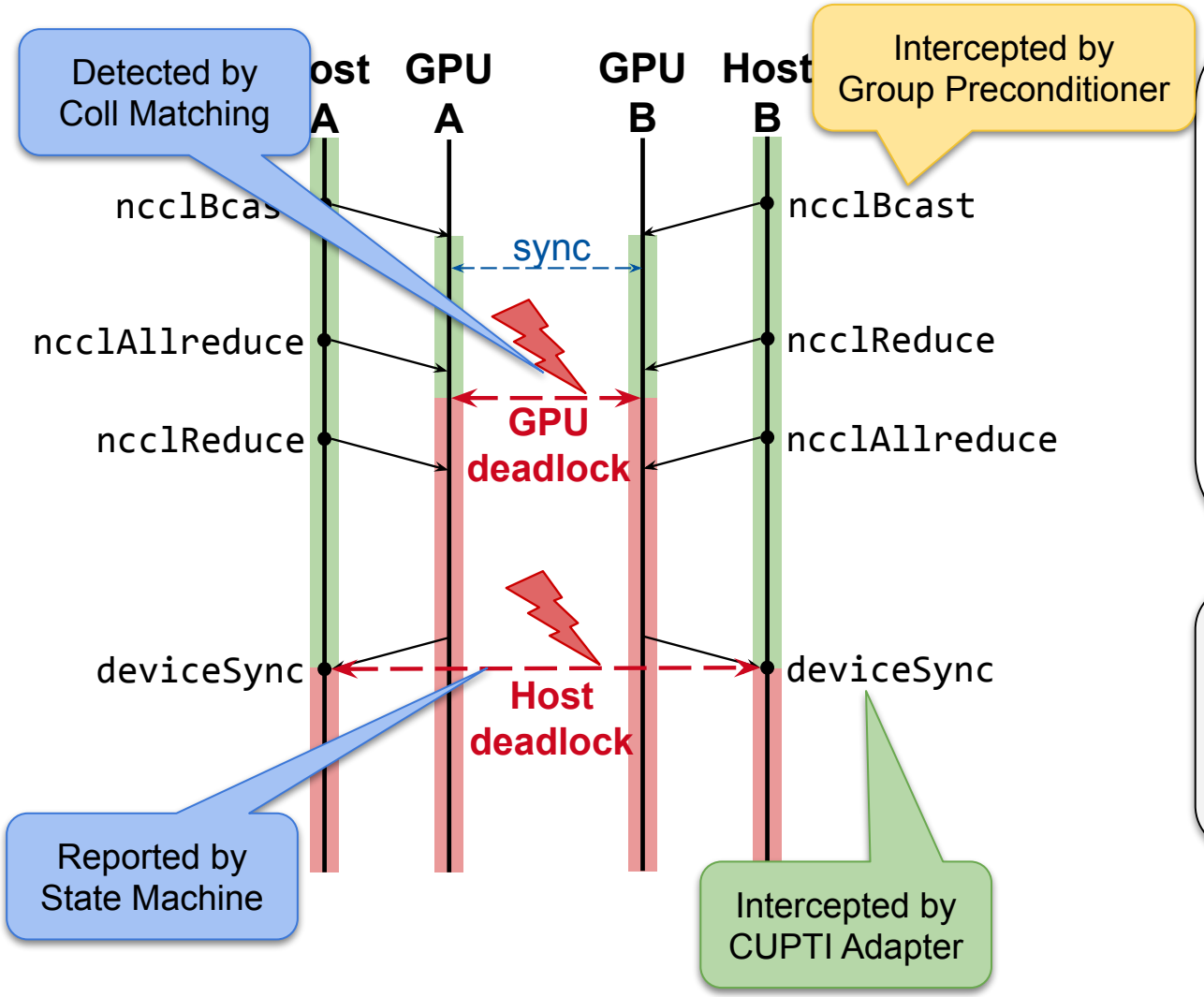
- Aggregate NCCL calls per group
  - Similar to nonblocking MPI calls + Waitall

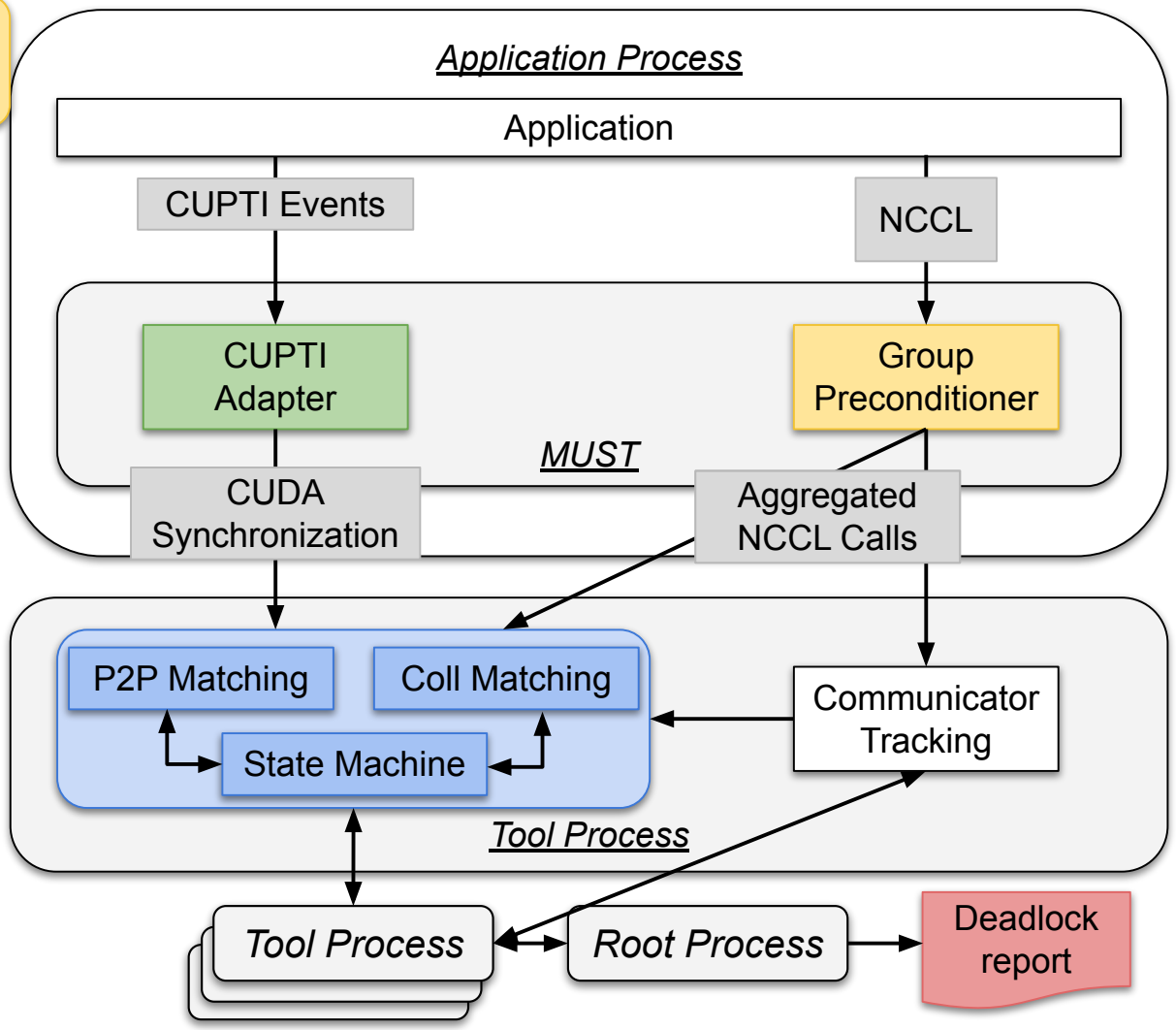
Distributed deadlock analysis

- Keep track of all processes blocking state
  - If all ranks are blocked →deadlock
- Also detects collective mismatches



NCCLSanitizer: Detecting deadlocks - Example





Evaluation: Configurations

baseline
- Application without any analyses

cusan compiled
- CuSan's race analysis (user kernels only)

must clean
- MUST without any NCCL analyses

must + cusan
- MUST with enabled NCCL analyses (race & deadlock) and CuSan's race analysis

Preliminary results
- Deadlock not independent of CuSan's instrumentation
- Will be updated for final paper

Evaluation: Runtime overhead analysis - NCCL Broadcast

Benchmark: NVIDIA's NCCL Tests

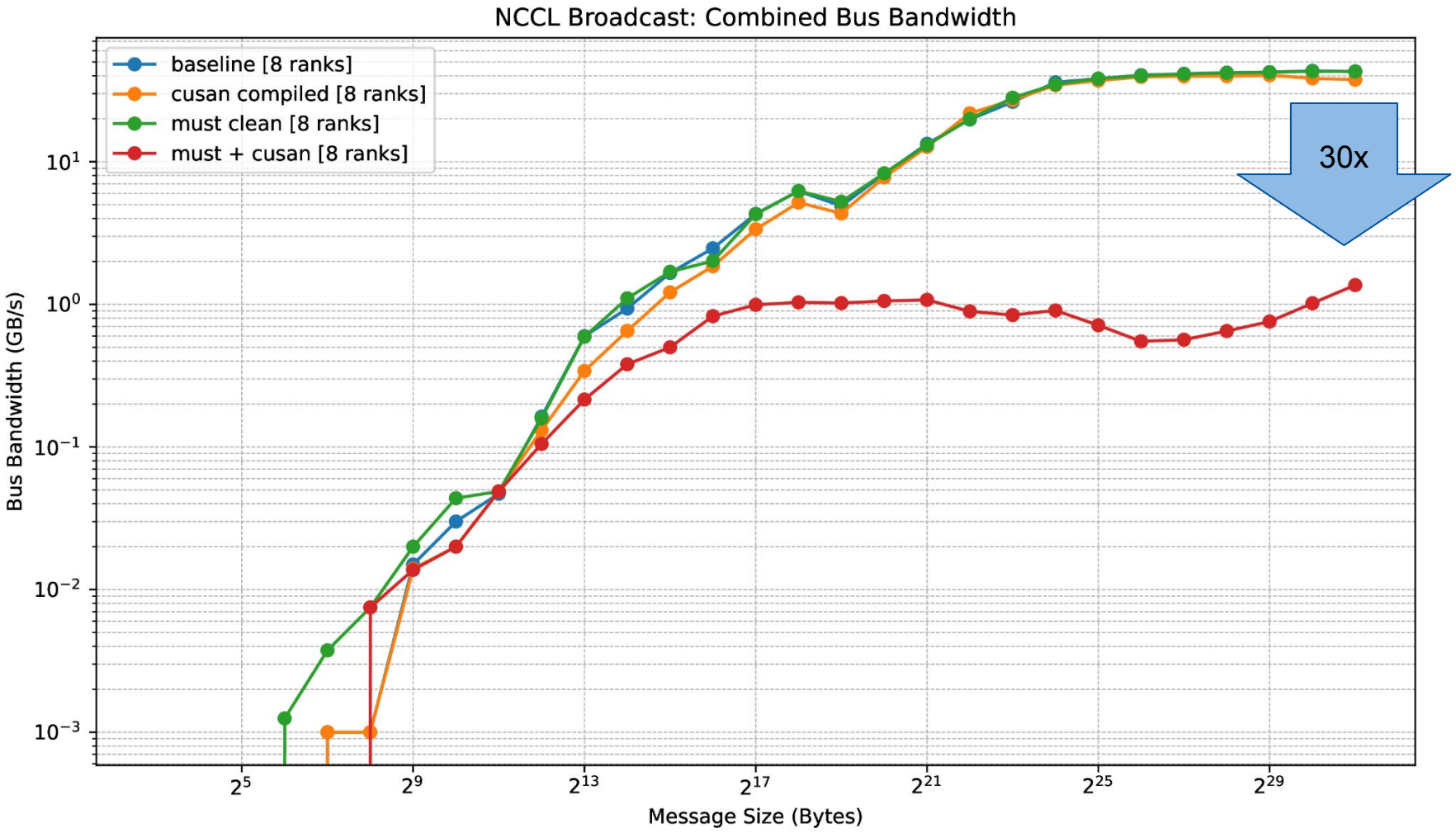
- Only communication
- No computation
- Only NCCL kernels
- No user kernels

Setup: 8 H100 across two nodes

- InfiniBand 50GB/s
- One rank per GPU
- MUST executions use additional processes on second node

Observations

- Further analysis revealed most overhead results from data race analysis



NHR4CES NHR for Computational Engineering Science
Performance Optimisation and Productivity A Centre of Excellence in HPC
i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Evaluation: Runtime overhead analysis - Jacobi kernel

Benchmark: Jacobi kernel

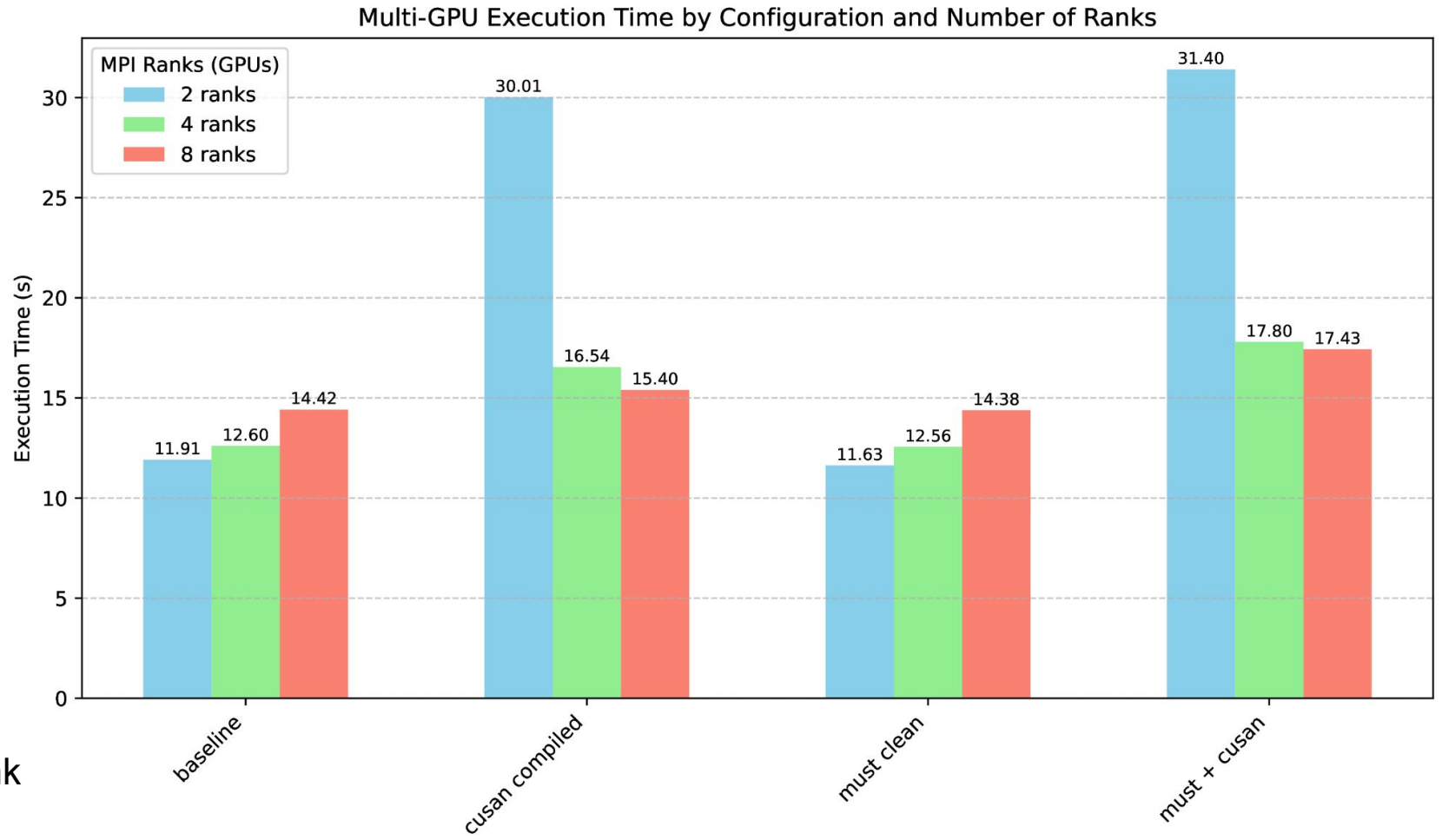
- NCCL P2P communication
- Computation CUDA kernels

Setup: *n* H100 across two nodes

- InfiniBand 50GB/s
- One rank per GPU
- MUST executions use additional processes on second node

Observations

- Single rank per node CuSan's race analysis dominates
- More ranks
  - →Less computation per rank
  - →Less analysis overhead



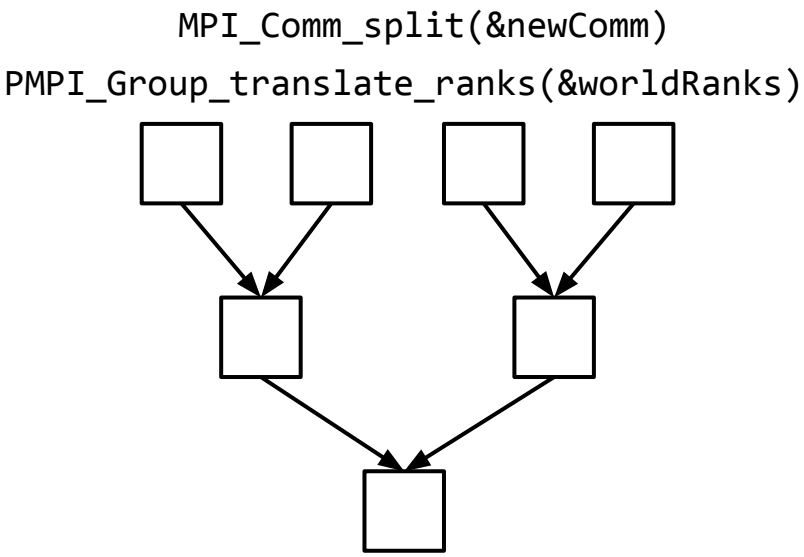
NHR4CES NHR for Computational Engineering Science
Performance Optimisation and Productivity
A Centre of Excellence in HPC
i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Tool development challenges

- MPI offers convenient functions to retrieve information on resources (communicators, groups, etc.)
  - E.g. world ranks in derived communicator
    - Needed for global view in deadlock detection

- NCCL only offers small set of getter functions
  - E.g. no way to deduce the global ranks of `ncclCommSplit` locally
    - Need to mimic logic in NCCLSanitizer
    - Implemented through global reduction in NCCLSanitizer
  - Additional implementation effort for tool developers
  - Additional runtime overhead (through communication & synchronization)

- One rank may manage multiple GPUs
  - Rank has multiple communicator handles for same communicator, each addressing a specific GPU



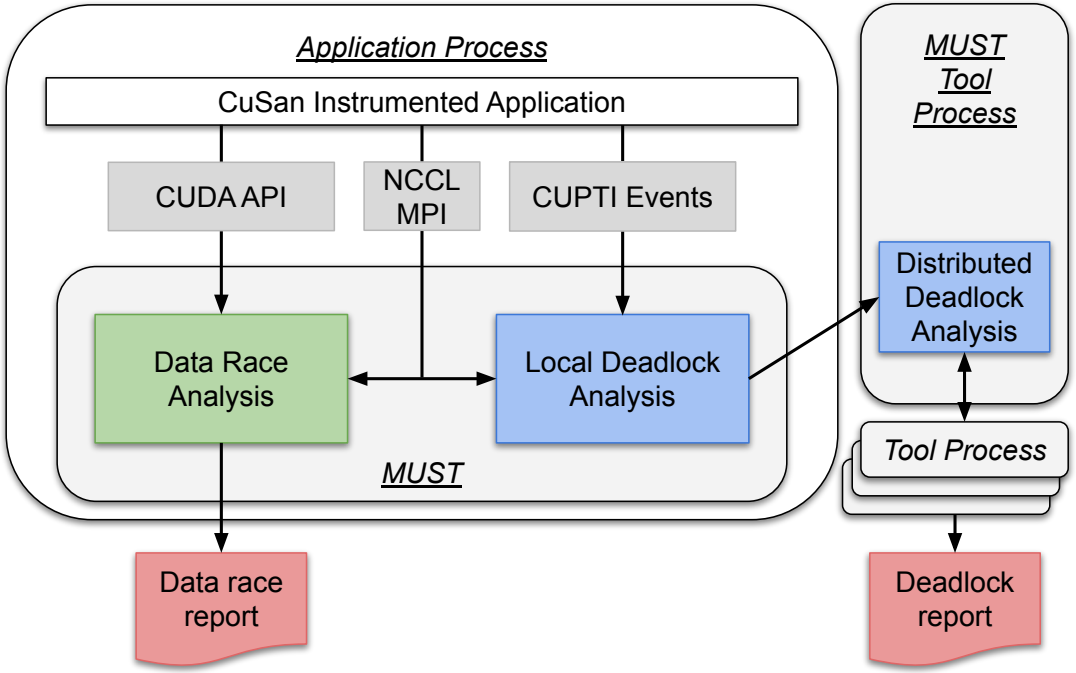
Summary

NCCL
- Collective Communication Library for GPU-GPU communication
- Allows stream-based communication
- Data races may occur between concurrent streams
- Deadlocks delayed observable at host

NCCLSanitizer
- Dynamic correctness tool for NCCL+MPI programs
- Supports data race and deadlock detection
  - GPU streams treated similar to threads in a single process
- Extends MUST and CuSan, uses PMPI and CUPTI

Runtime overhead
- Data race analysis dominates overhead
- Runtime slowdown of up to 30x for worst case
- Under 3x for kernel including computation

Limitations & Future work
- CUDA Graphs and Events
- AMD's RCCL+ROCm and other vendor's libraries



Thank you for your attention!
Questions?