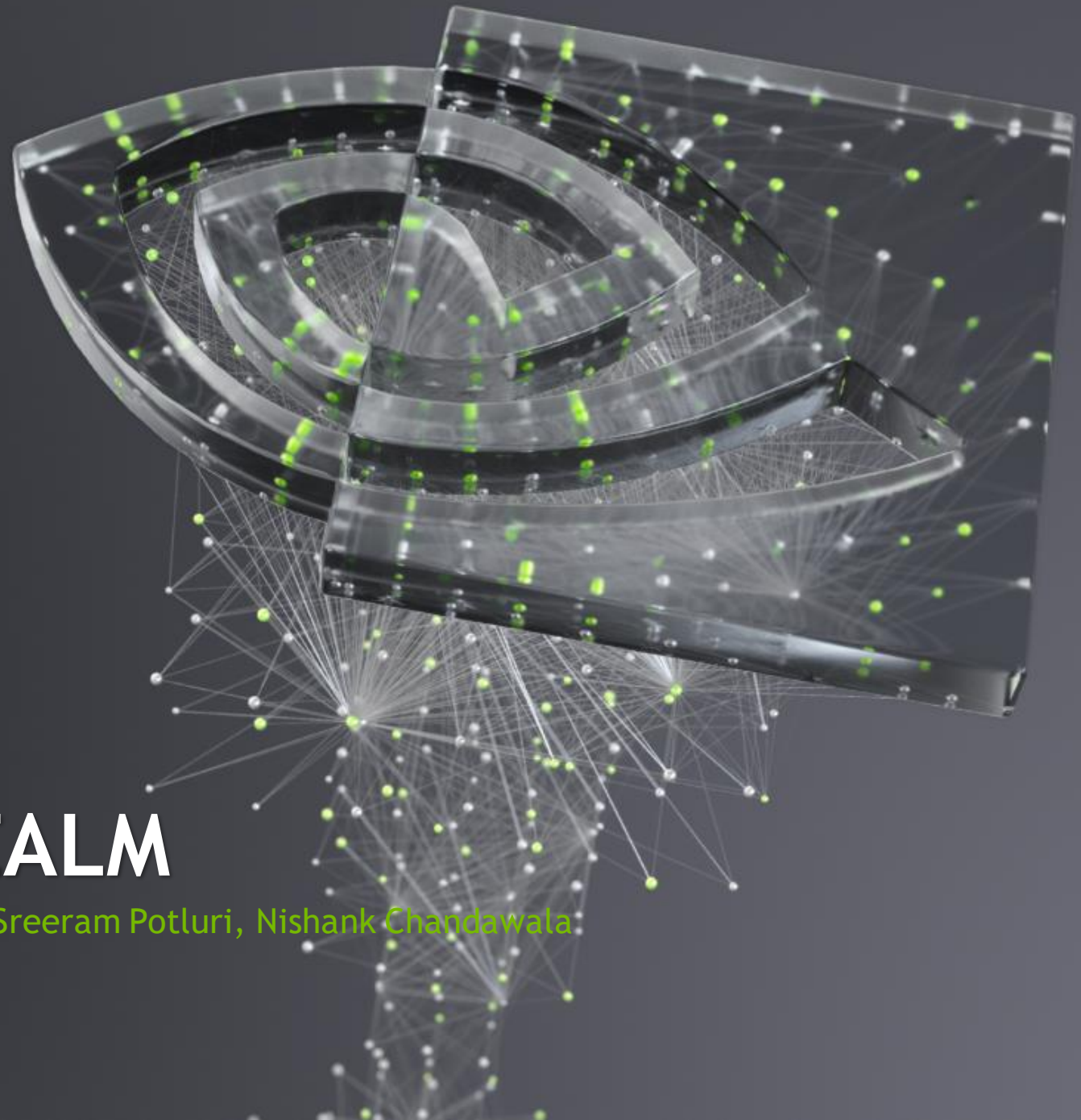


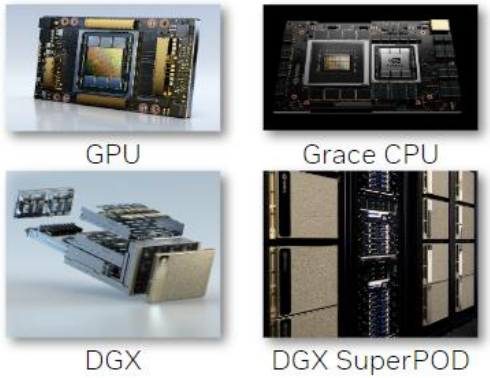
UCX BACKEND IN REALM

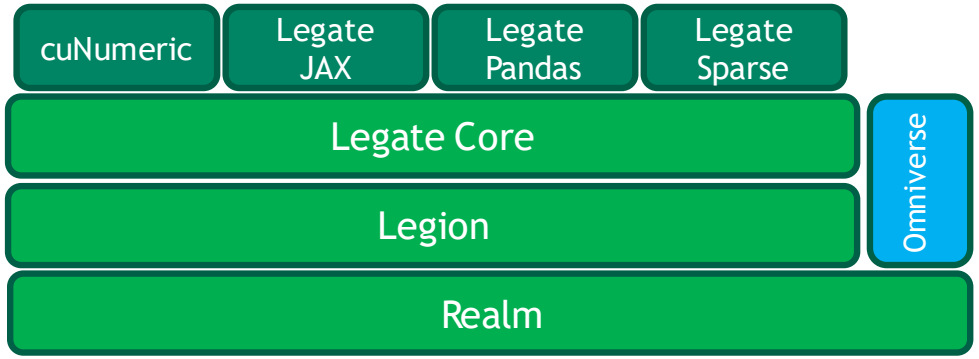
Hessam Mirsadeghi, Akshay Venkatesh, Jim Dinan, Sreeram Potluri, Nishank Chandawala

hmirsadeghi@nvidia.com

REALM SUMMARY

- Asynchronous runtime for heterogeneous distributed memory machines
- Abstract machine model
  - Processors (CPUs and GPUs)
  - Memories
- A key part of the LLR software stack (Legate/Legion/Realm)
  - Transparent scalability (multi-GPU, multi-node)



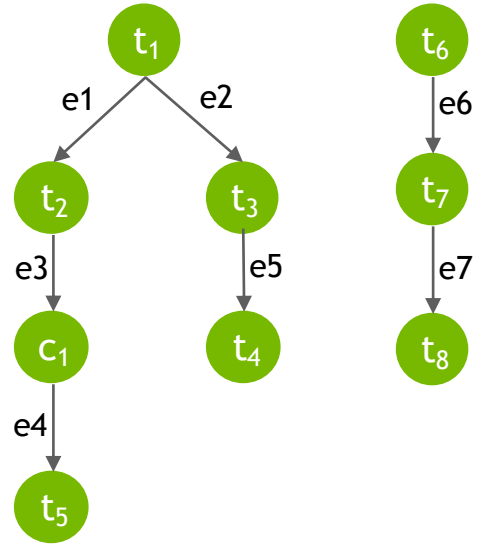


https://legion.stanford.edu/

REALM SUMMARY

How is it different from MPI?

- Realm is an explicit representation runtime
- Parallel application expressed in terms of an operations graph
  - Nodes: tasks, data copies
  - Edges: events representing ordering and dependences
  - Generated dynamically at runtime
- Direct access to the graph
  - Realm does all the synchronization and scheduling (not the programmer)
  - Recognize and exploit operations overlap opportunities

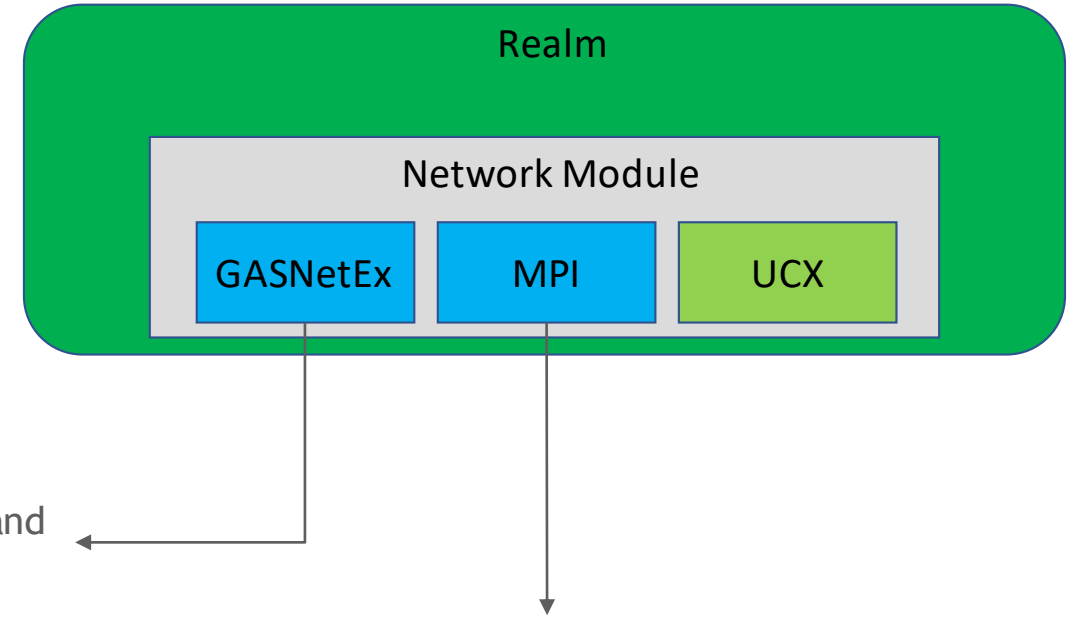


Essential to lower runtime overheads
Communication cost in distributed memory machines

REALM MULTI-NODE SUPPORT

Network Module

- Implemented by communication backends
- Active Message API
  - Data copies
  - Control messages



Networking middleware layer that provides AM, RMA, and collective operations https://gasnet.lbl.gov/

PoC backend for Realm; not performant

REALM UCX BACKEND

Why a UCX backend?

- Latest hardware/software features (RDMA, GPUDirect, DPU)
  - New NVIDIA networking innovations are exposed through UCX first
- Best performance for both CPU and GPU point-to-point communications
  - NVIDIA efforts for CUDA-aware communications are funneled through UCX
- Fewer external dependencies, more unified software stack
  - NVIDIA is the main contributor to UCX
- Better support for future Realm requirements
  - Elasticity, fault tolerance

REALM UCX BACKEND

Realm requirements

- Active messages
- Non-blocking operations
- Communication progress must be explicit
  - No internal progress threads
- Multi-threaded support
- Pre-registration of buffers
- Efficient zero-copy transfer of pre-registered buffers
- Efficient one-sided data transfer
- Performant GPU-to-GPU communications
- Fault tolerance
- Elasticity
  - Add/remove nodes on demand
  - Dynamic creation/removal of end points

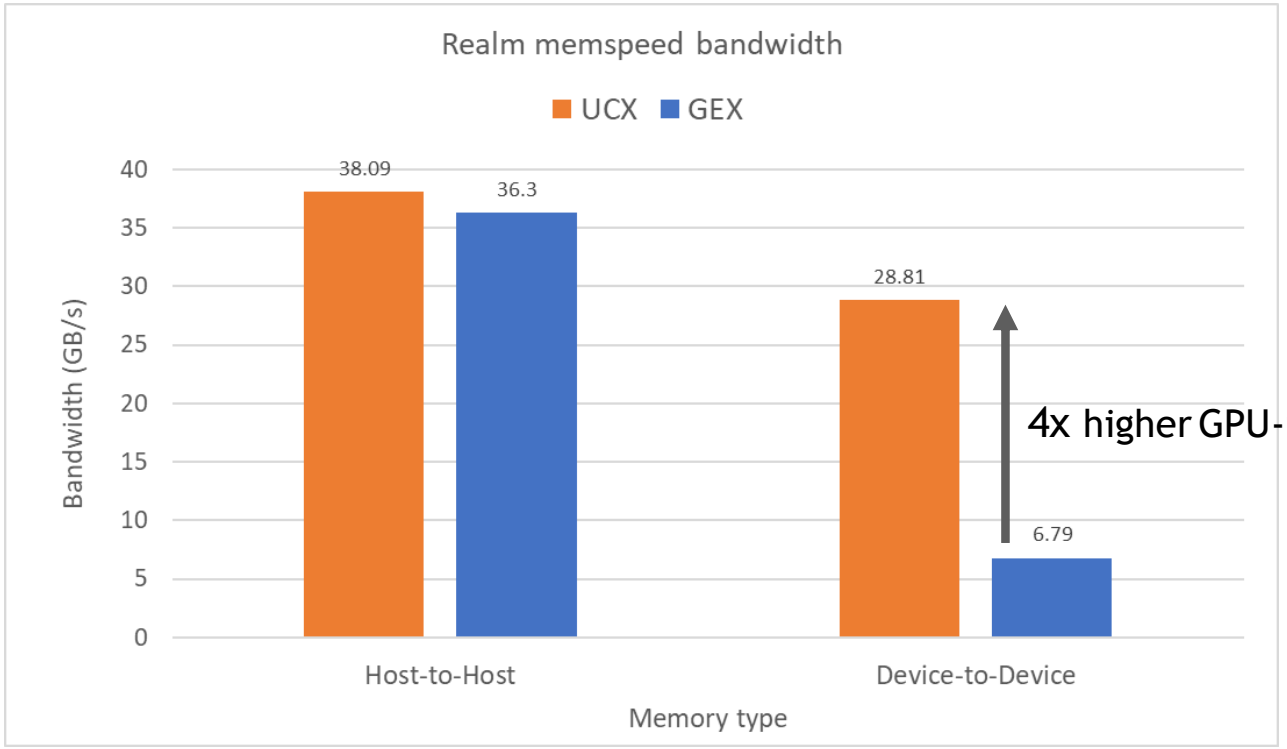
UCX features

- UCP active message API
- UCP provides non-blocking communication operations
- Progress is done by explicitly calling `ucp_worker_progress()`
- UCP worker created with `UCS_THREAD_MODE_MULTI`
- `ucp_mem_map()`
- Registration cache + zero-copy protocols
- UCP one-sided put/get operations
- GPU awareness, GPUDirect RDMA, topology awareness (GPU-NIC affinity)
- Isolated error handling
- Simply create a new UCP end point using remote worker/IP address
  - Client-server API facilitates this further

PERFORMANCE RESULTS

memspeed bandwidth, UCX vs. GASNetEx

2 Nodes
8 IB HDR 200 Gbps NICs per node
8 NVIDIA A100 GPUs per node



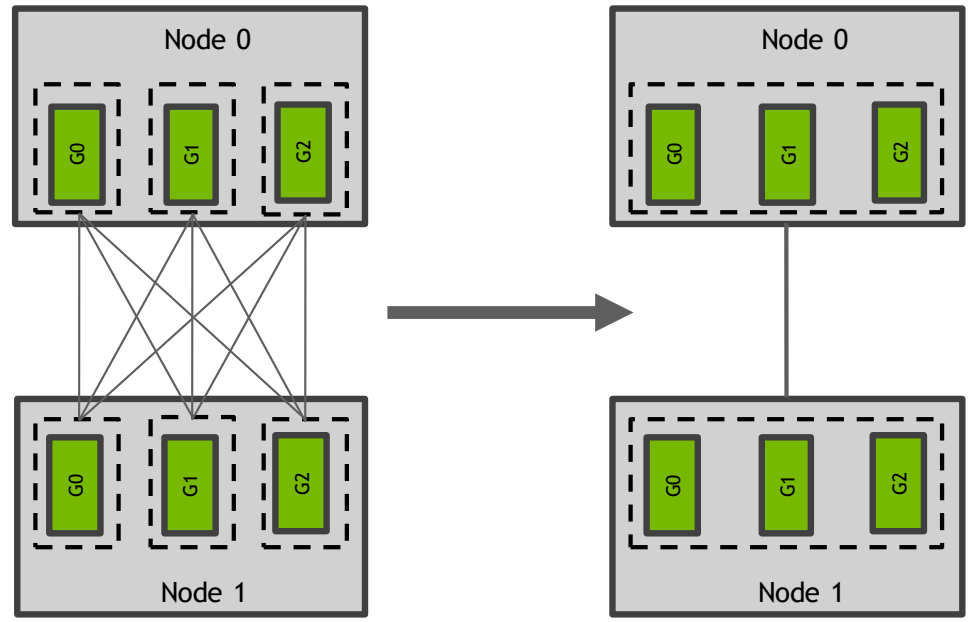
4x higher GPU-to-GPU bandwidth with UCX

CHALLENGES AND FEATURE GAPS

CHALLENGES

Multi-GPU support per process

- Realm: multi-GPU per process usage model
- UCX: single-GPU per process usage model
- Workaround
  - Creating separate UCX contexts, one per GPU
    - Set the corresponding GPU context before `ucp_init()`
    - Push/pop the corresponding GPU contexts
  - Caveat: extra end points → extra communication progress overhead



End points per node: $O(NG^2)$

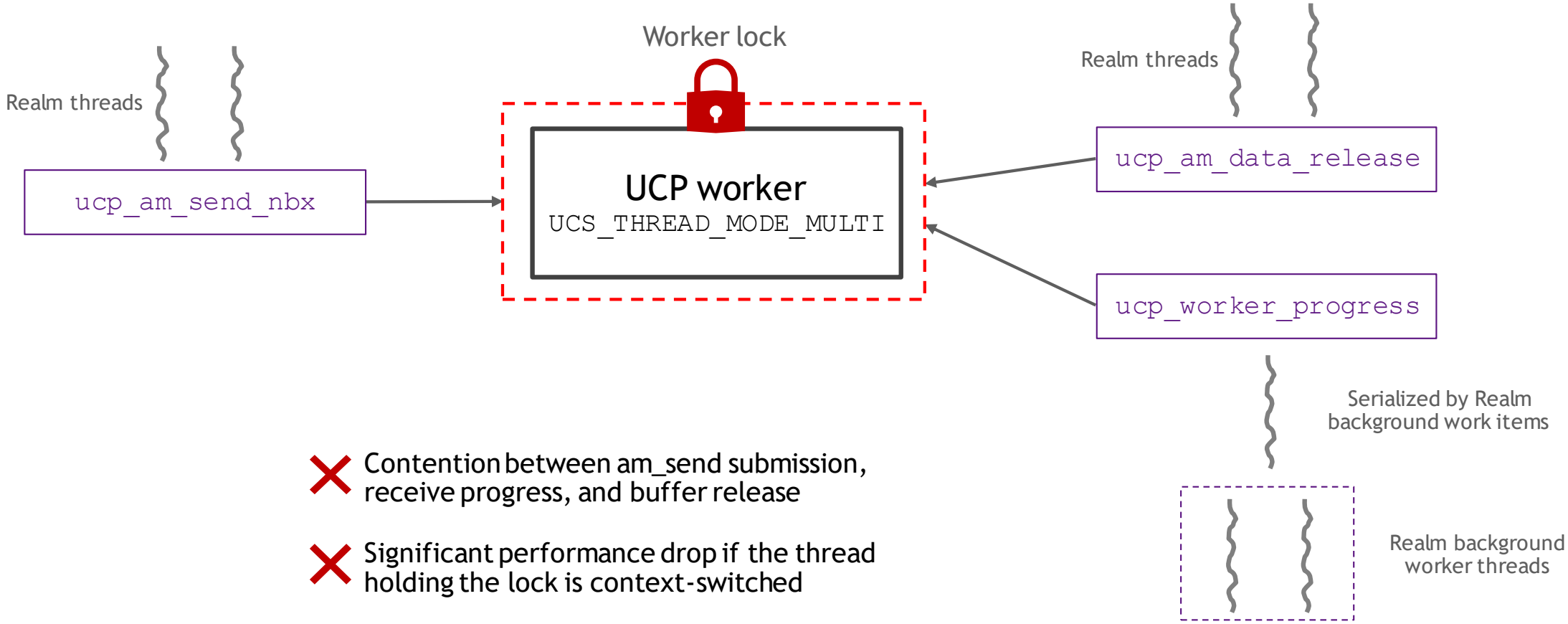
End points per node: $O(N)$

N = number of nodes
G = number of GPUs per node

Desired solution:
Adding native multi-GPU per process support in UCX

CHALLENGES

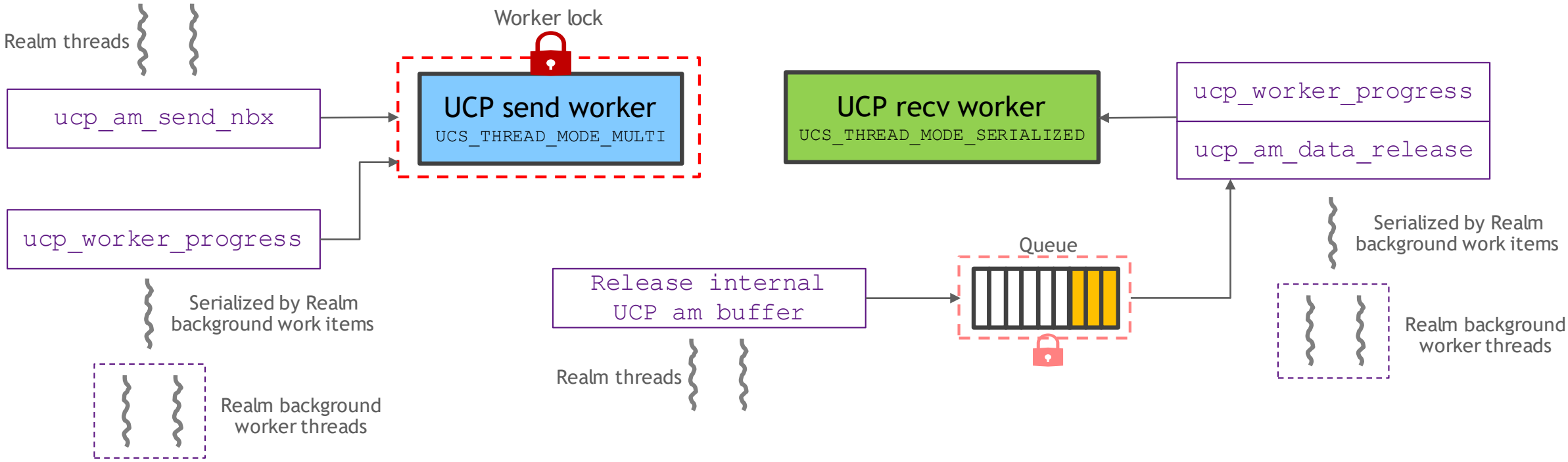
Coarse-grained UCP worker lock



- ✗ Contention between am_send submission, receive progress, and buffer release
- ✗ Significant performance drop if the thread holding the lock is context-switched

CHALLENGES

Coarse-grained UCP worker lock – Separate send recv workers



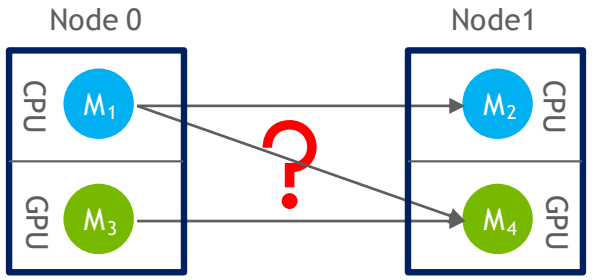
- ✓ No contention between send and receive
- ✓ No lock for the receive worker (serialized mode)
- ✓ Small-scope queue lock

UCX FEATURE GAPS FOR REALM

Providing Realm with what it needs to build its DMA graph

Realm DMA graph

- Nodes: memory regions
- Edges: communication channels



Minimal UCP feature

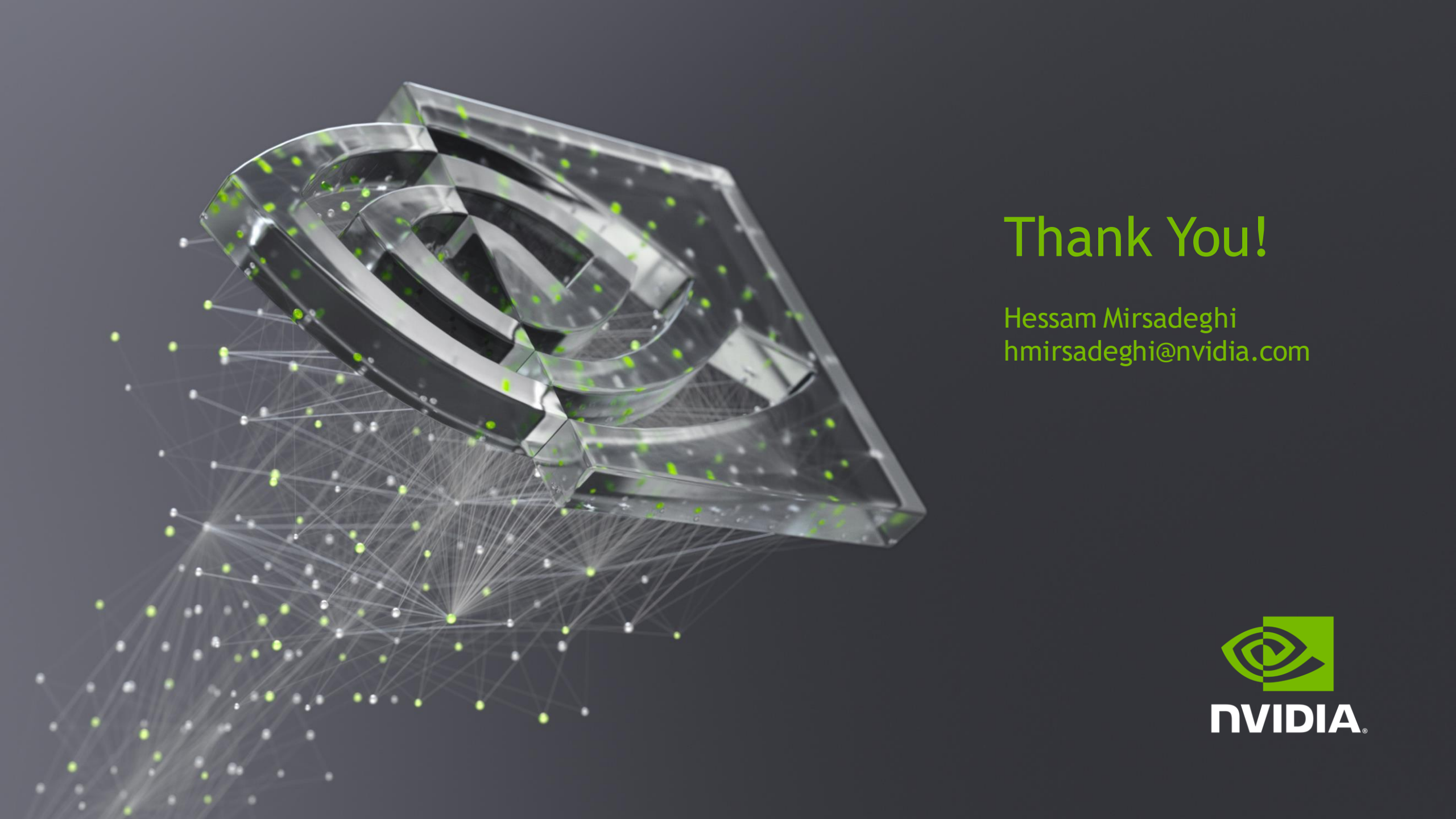
1. Disable copy-based protocols
2. Query if we can transfer from/to a given buffer
   - If so → zcopy transfer → add an edge in the graph

Ideal UCP feature

1. Query UCP about protocols details
   - What are the possible protocols?
   - What is the latency and bandwidth of each protocol?
2. Enforce a specific protocol per operation
   - Realm's global vs. UCX's local view of communications

UCX FEATURE GAPS FOR REALM

- Query UCP about the maximum single-fragment message size
  - Avoid malloc, copy and reassemble on the receiving side
  - Realm already does fragmentation
- Prioritized communications
  - Enforce high/low priorities per operation
    - Critical small control messages vs. large data transfers
- Enforcing order between active messages and RMA operations
  - `ucp_put` the payload followed by a header-only active message



Thank You!

Hessam Mirsadeghi
hmirsadeghi@nvidia.com

NVIDIA