Tutorial: Concurrent Data Structures in RDMA

Vitaly Aksenov*, Amanda Baran, Alex Clevenger, Roberto Palmieri,
Yaodong Sheng, Michael Spear

Scalable Systems & Software Research Group
Lehigh University; City, University of London, UK (*)

SCALABLE SYSTEMS
& SOFTWARE
RESEARCH GROUP

The 37th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA ’25)

LEHIGH
UNIVERSITY

The Rise of RDMA

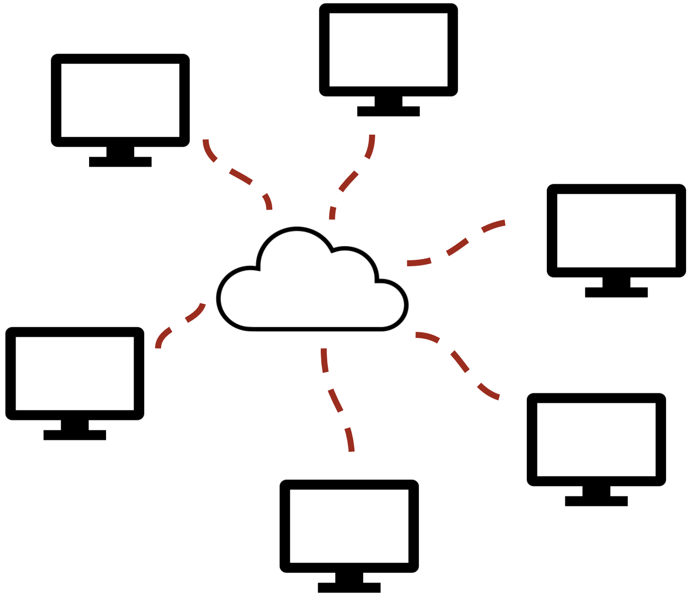
Remote

Direct

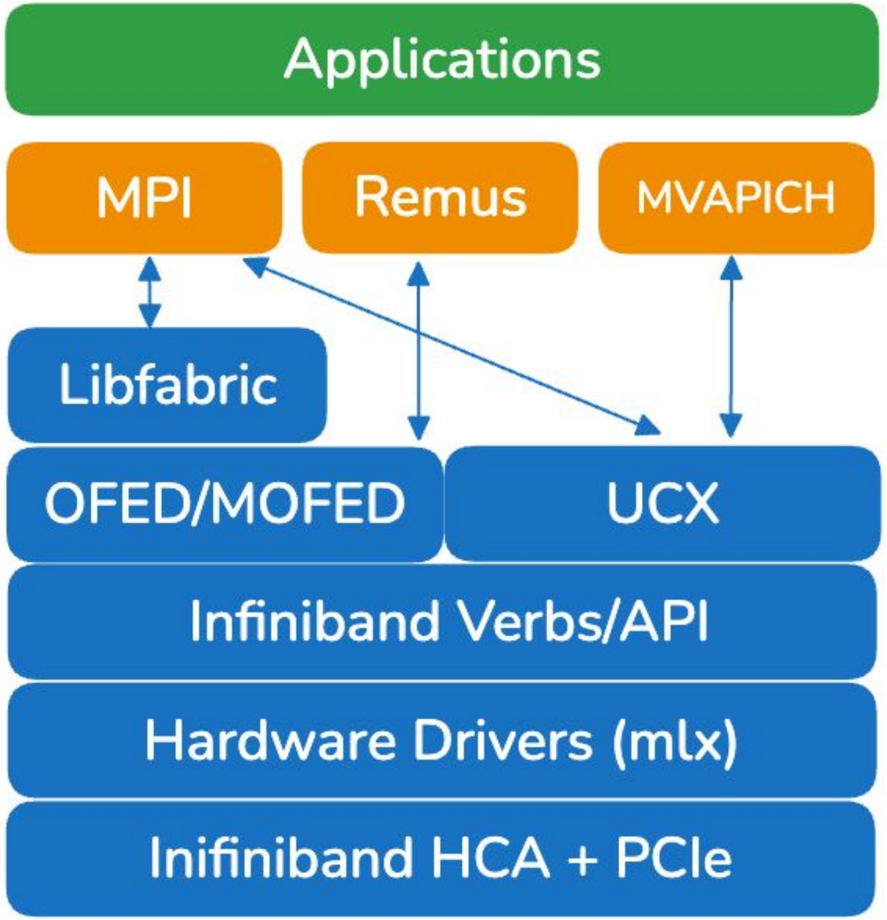
Memory

Access

- Allows a process to directly interact with memory on another node
- Kernel bypass technique
- Sub-microsecond latencies
- > 400 Gbps bandwidth
- GPU integration
- Applications: LLM Inference, HPC, Realtime/Exascale/Datacenter Computing



RDMA Ecosystem & RNIC Performance



Applications

MPI

Remus

MVAPICH

Libfabric

OFED/MOFED

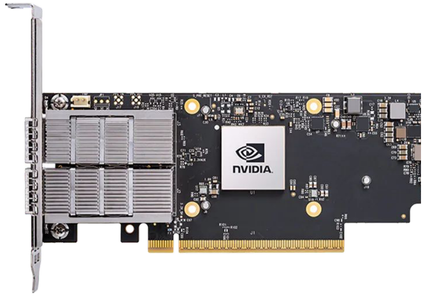
UCX

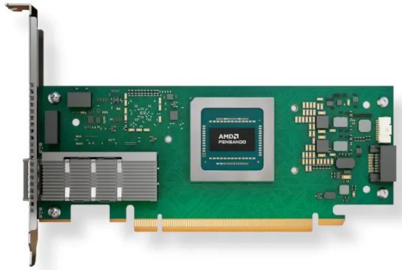
Infiniband Verbs/API

Hardware Drivers (mlx)

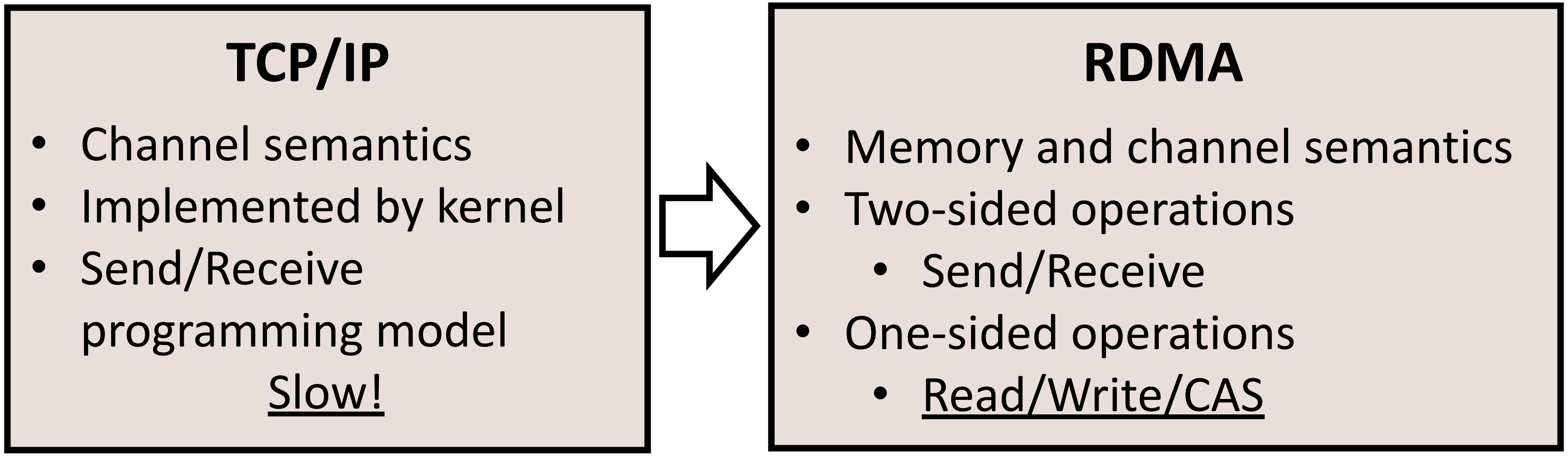
Inifiniband HCA + PCIe

RNIC Name	Ethernet BW	Infiniband BW	PCIe Compatability
ConnectX-3	$\leq$ 40 Gbps	$\leq$ 56 Gbps)	PCIe 3.0
ConnectX-4	$\leq$ 100 Gbps	$\leq$ 100 Gbps	PCIe 3.0/4.0
ConnectX-5	$\leq$ 100 Gbps	$\leq$ 200 Gbps	PCIe 3.0/4.0
ConnectX-6	$\leq$ 200 Gbps	$\leq$ 200 Gbps	PCIe 3.0/4.0
ConnectX-7	$\leq$ 400 Gbps	$\leq$ 400 Gbps	PCIe 5.0
Intel Ethernet 800	$\leq$ 200 Gbps	N/A	PCIe 4.0
Chelsio T6	$\leq$ 100 Gbps	N/A	PCIe 3.0





A shift from legacy network programming



TCP/IP

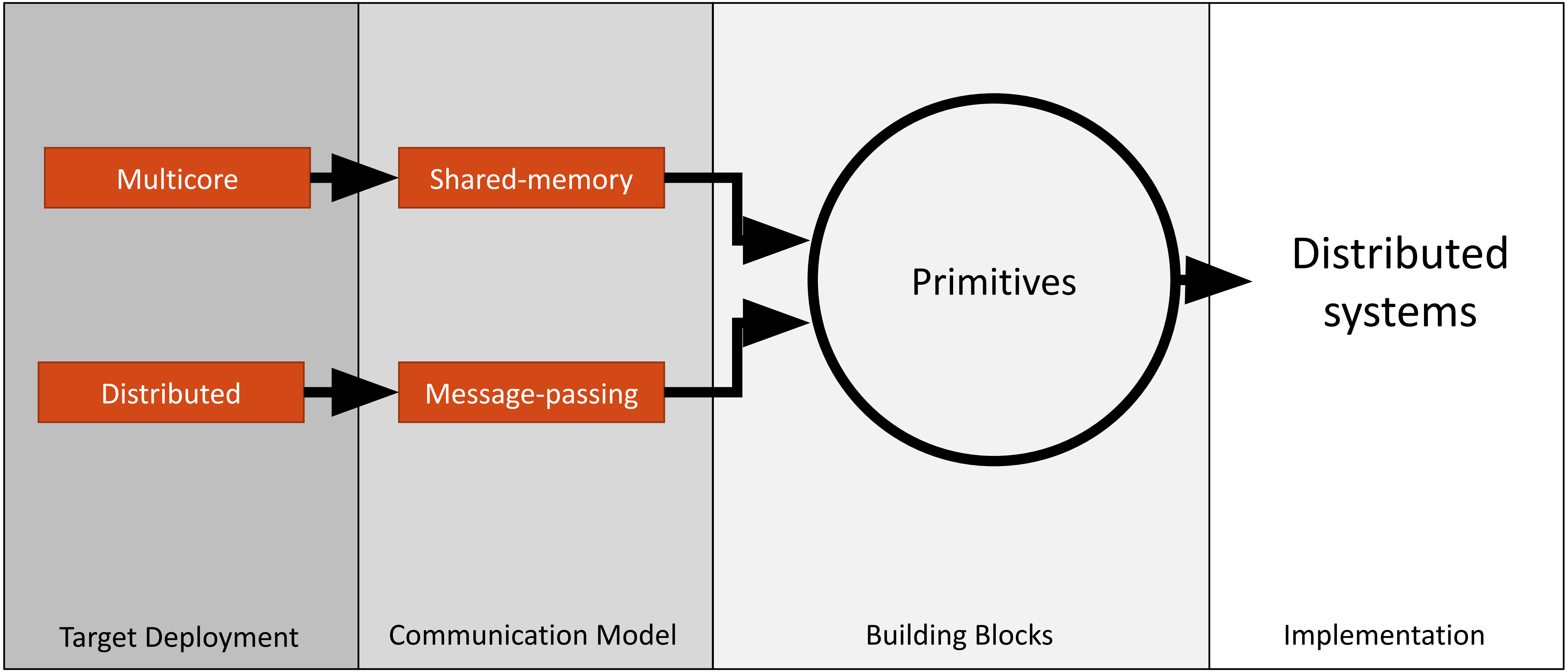
- Channel semantics
- Implemented by kernel
- Send/Receive programming model

Slow!

RDMA

- Memory and channel semantics
- Two-sided operations
  - Send/Receive
- One-sided operations
  - Read/Write/CAS

A Traditional View of Distributed Systems



Multicore

Distributed

Shared-memory

Message-passing

Primitives

Distributed systems

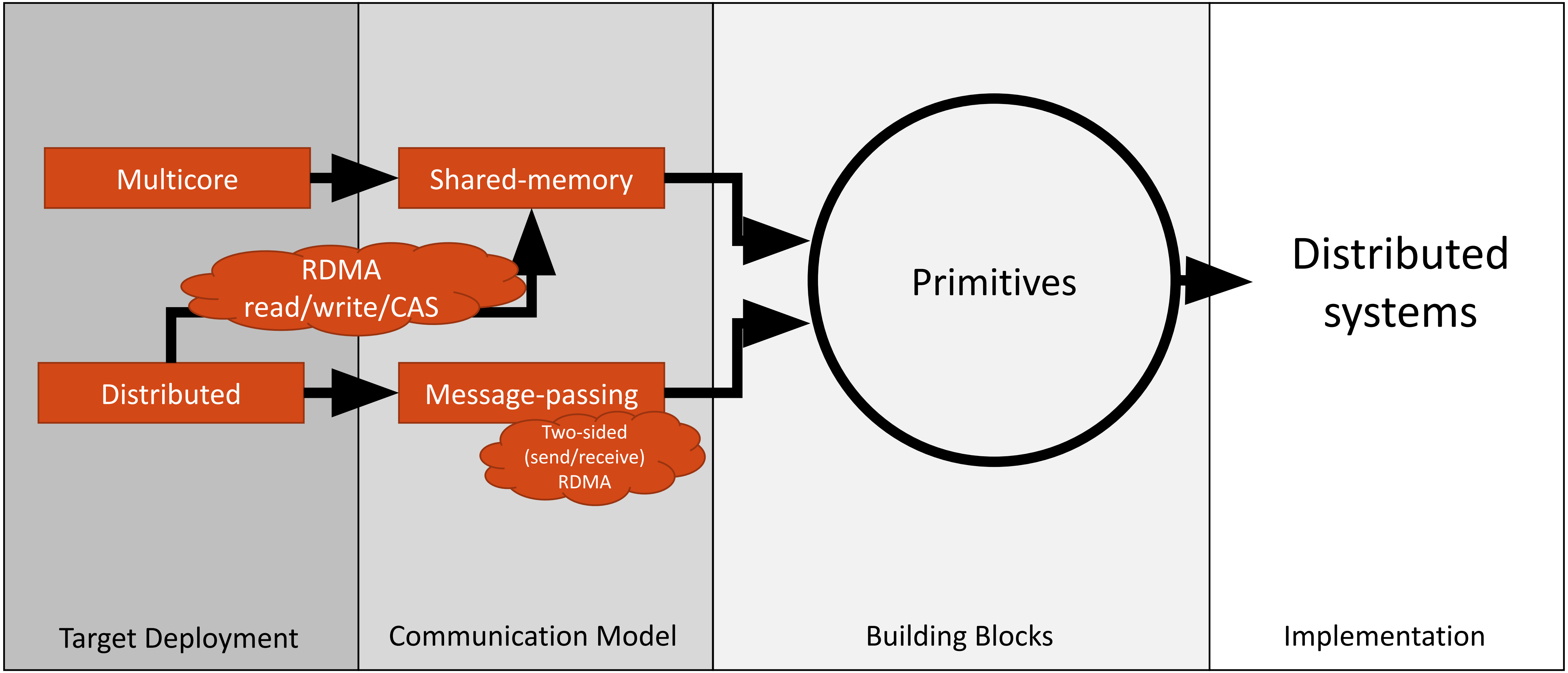
Target Deployment

Communication Model

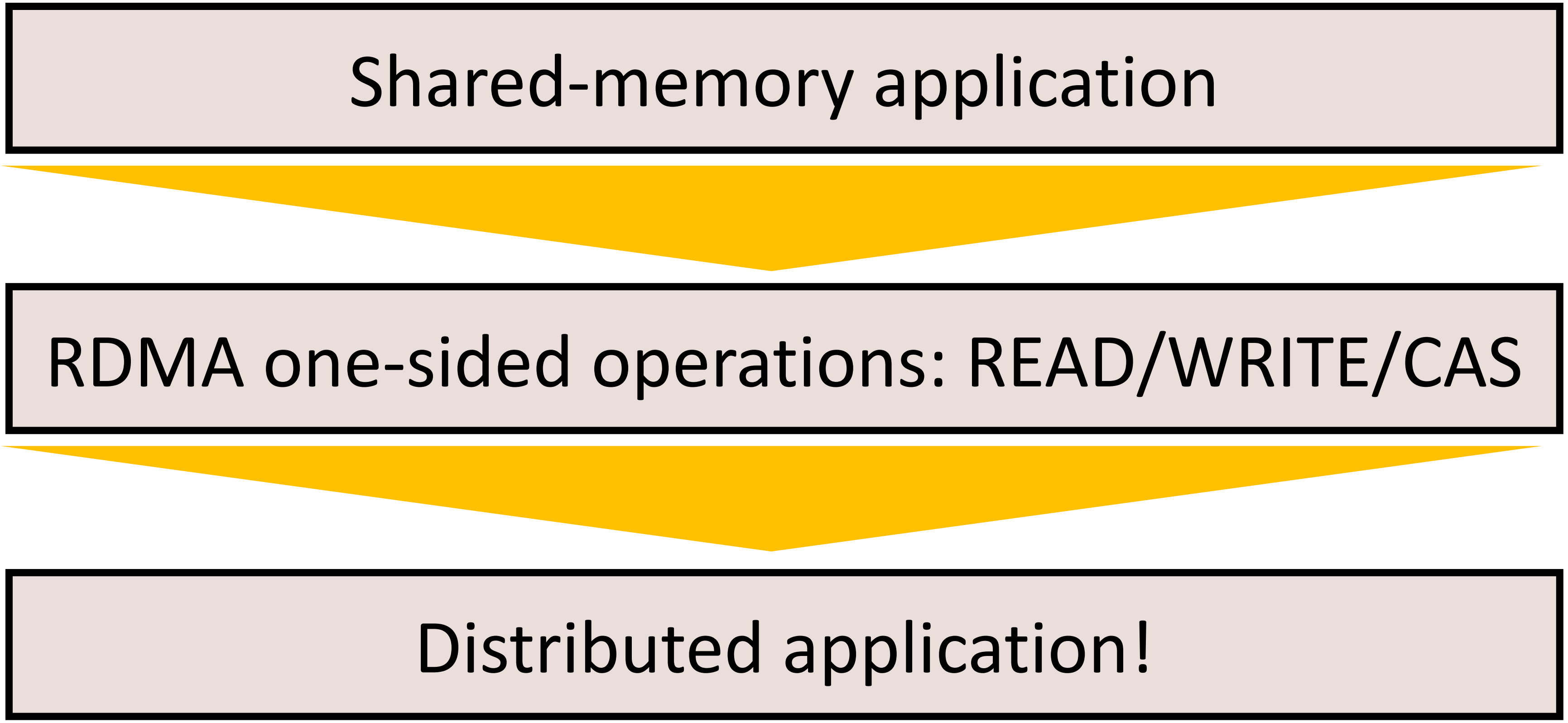
Building Blocks

Implementation

A Modern View of Distributed Systems



Problem solved!

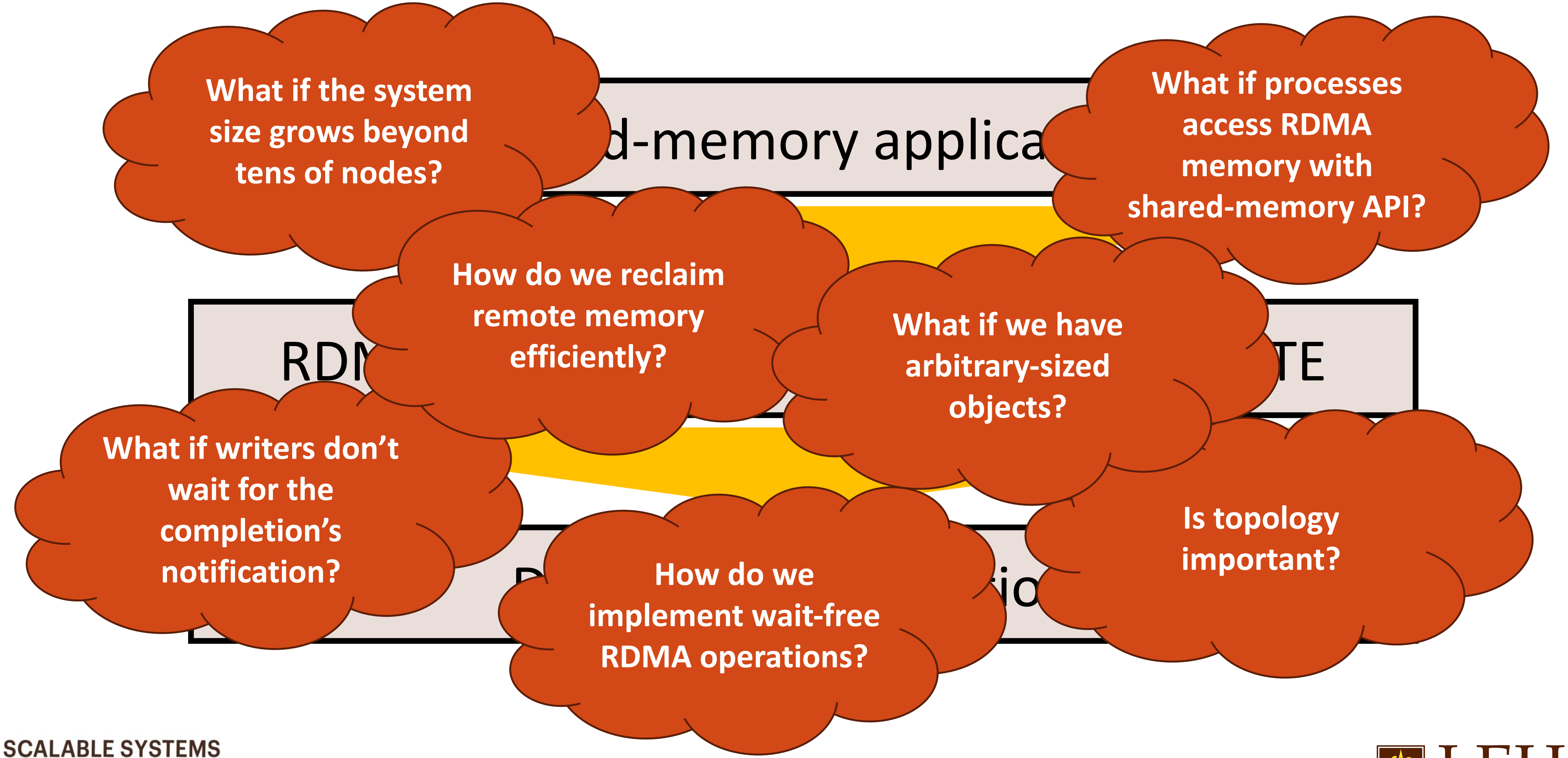


Shared-memory application

RDMA one-sided operations: READ/WRITE/CAS

Distributed application!

Problem !solved



What if the system size grows beyond tens of nodes?

What if processes access RDMA memory with shared-memory API?

How do we reclaim remote memory efficiently?

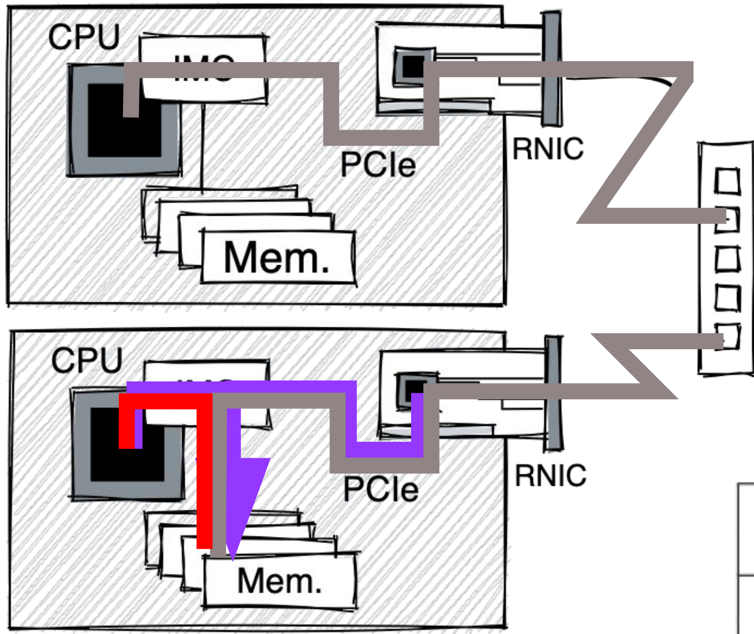
What if we have arbitrary-sized objects?

What if writers don't wait for the completion's notification?

How do we implement wait-free RDMA operations?

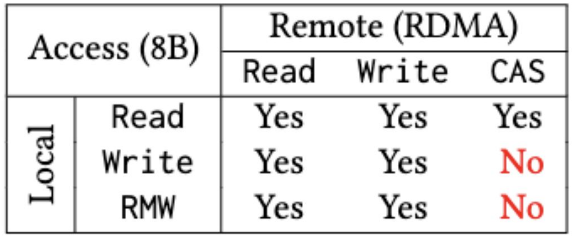
Is topology important?

Process Roles in the RDMA Model



Network fabric
(e.g., RoCE, InfiniBand)

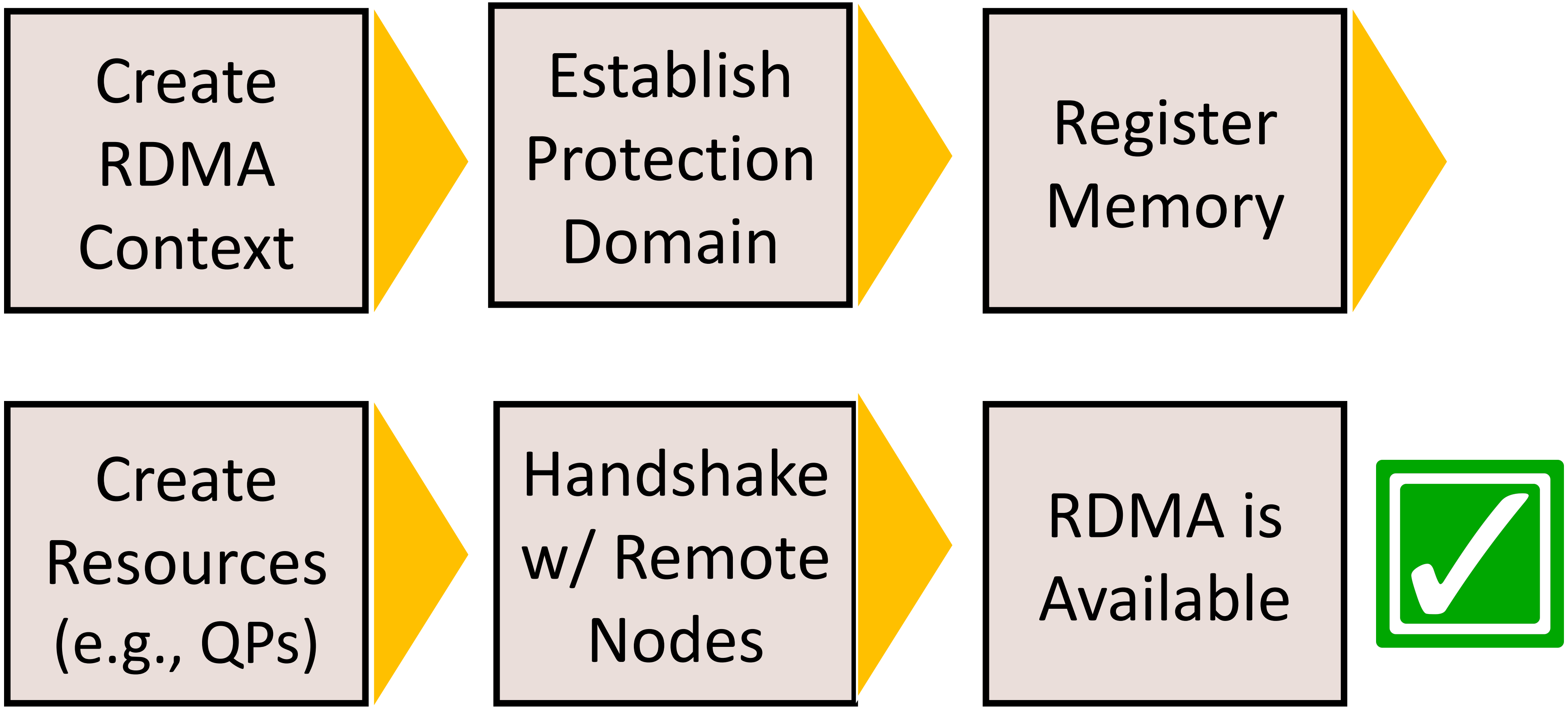
IMC: Integrated memory controller
PCIe: Peripheral Component Interconnect Express
RNIC: RDMA-capable network interface controller
RoCE: RDMA over Converged Ethernet



Access (8B)		Remote (RDMA)		
		Read	Write	CAS
Local	Read	Yes	Yes	Yes
	Write	Yes	Yes	No
	RMW	Yes	Yes	No

- **Local processes** access memory using the underlying memory subsystem (native access)
- **RDMA loopback** allows local processes to access memory through the RNIC (remote access)
- **Remote processes** utilize network + RNIC + PCIe bus + mem. subsystem (remote access)
- Kernel bypass

Setting up programs using RDMA



Create RDMA Context

Establish Protection Domain

Register Memory

Create Resources (e.g., QPs)

Handshake w/ Remote Nodes

RDMA is Available ✓

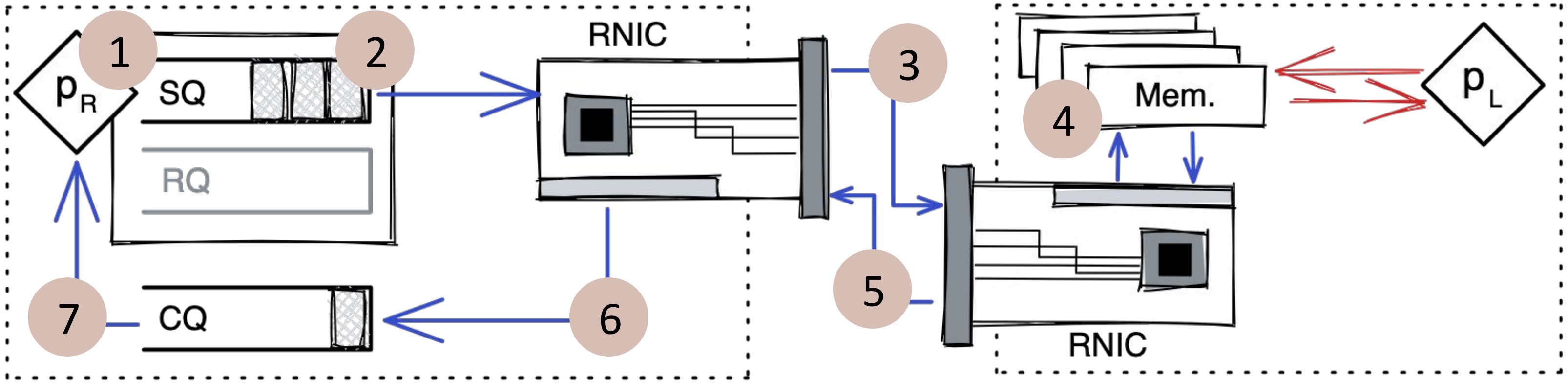
Handling RDMA One-sided Operations

1 p_R posts to its SQ to initiate RDMA request

2 Local RNIC fetches req. from memory and 3 issues it

4 Remote RNIC processes req. directly in memory and 5 responds

6 Local RNIC notifies p_R of result through CQ 7



p_R: Remote process using RDMA
p_L: Local process using native access

SQ: Send queue
RQ: Receive queue (unused)
CQ: Completion queue

What's Remus and why we need it

Without Remus

```
bool postWRandPollCQ(const std::vector<struct ibv_mr*>& sge, struct ibv_cq* cq) {
  struct ibv_send_wr rdma_wr, send_wr, *bad_wr = nullptr;
  memset(&send_wr, 0, sizeof(send_wr));
  rdma_wr.wr.rdma.remote_addr = peer_memory_region->addr;
  rdma_wr.wr.rdma.rkey = peer_memory_region->rkey;
  rdma.wr.opcode = IBV_WR_RDMA_READ;
  struct ibv_sge* send_sge = calloc(sizeof(struct ibv_sge), sge.size());
  for (int i = 0; i < sge.size(); i++) {
    send_sge[i].addr = (uintptr_t) sge[i].addr;
    send_sge[i].length = sge[i].length;
    send_sge[i].lkey = sge[i].lkey;
  }
  send_wr.sg_list = send_sge;
  send_wr.num_sge = sge.size();
  send_wr.wr_id = 200;

  // Post send WR with desired op code (READ/WRITE/SEND/RECV)
  send_wr.opcode = IBV_WR_RDMA_READ;
  send_wr.send_flags = IBV_SEND_SIGNALED;
  send_wr.next = nullptr;
  auto result = ibv_post_send(queue_pair_, &send_wr, &bad_wr);
  free(send_sge);

  // Poll CQ
  struct ibv_wc wc;
  int result;
  do {
      result = ibv_poll_cq(cq, 1, &wc);
  } while (result == 0);
  if (result > 0 && wc.status == ibv_wc_status::IBV_WC_SUCCESS) {
    return true;
  }
  printf("Poll failed with status %s (work request ID: %llu)\n",
ibv_wc_status_str(wc.status), wc.wr_id);
  return false;
}
```

With Remus

```
cThread->Read(ptr, obj);
```

Open Distributed Computing Questions

- Caching (hardware cache won't help as it does on shared memory)
- Synchronization (in the absence of global atomicity)
- Fault tolerance
- Security
- Topology Policies (how to build & how to use)
- Memory Allocation (& allocation policies)
- And many more :)

Our RDMA Contributions

- [CGO 2019] *Understanding RDMA Behavior in NUMA Systems*, J. Nelson and R. Palmieri
- [ICDCS 2020] *On the Performance Impact of NUMA on One-sided RDMA Interactions*, J. Nelson and R. Palmieri
- [SPAA 2024] *Brief Announcement: ROMe: Wait-free Objects for RDMA*, J. Nelson-Slivon, R. Yankovich, A. Hassan, and R. Palmieri
- [SPAA 2024] *ALock: Asymmetric Lock Primitive for RDMA Systems*, A. Baran, J. Nelson-Slivon, L. Tseng, and R. Palmieri
- [SRDS 2025] *On Designing High-Performance Distributed Shared Memory Systems with RDMA,* A. Baran and R. Palmieri
- More in progress :)

Thanks! & Questions?

https://sss.cse.lehigh.edu/

https://github.com/sss-lehigh