Synchronization using RDMA: High Performance, Programmability, and Scalability

Roberto Palmieri

Scalable Systems & Software Research Group

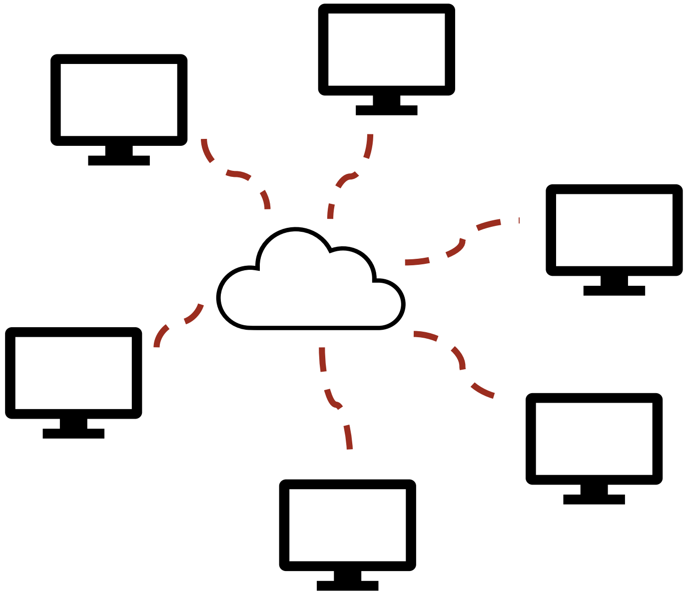
Lehigh University

1st Workshop on Distributed Computing with Emerging Hardware Technology

SCALABLE SYSTEMS & SOFTWARE RESEARCH GROUP

LEHIGH UNIVERSITY

The Rise of RDMA



Remote
Direct
Memory
Access

- Allows a process to directly interact with memory on another node
- Sub-microsecond latencies
- > 200 Gbps bandwidth

TPC/IP

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

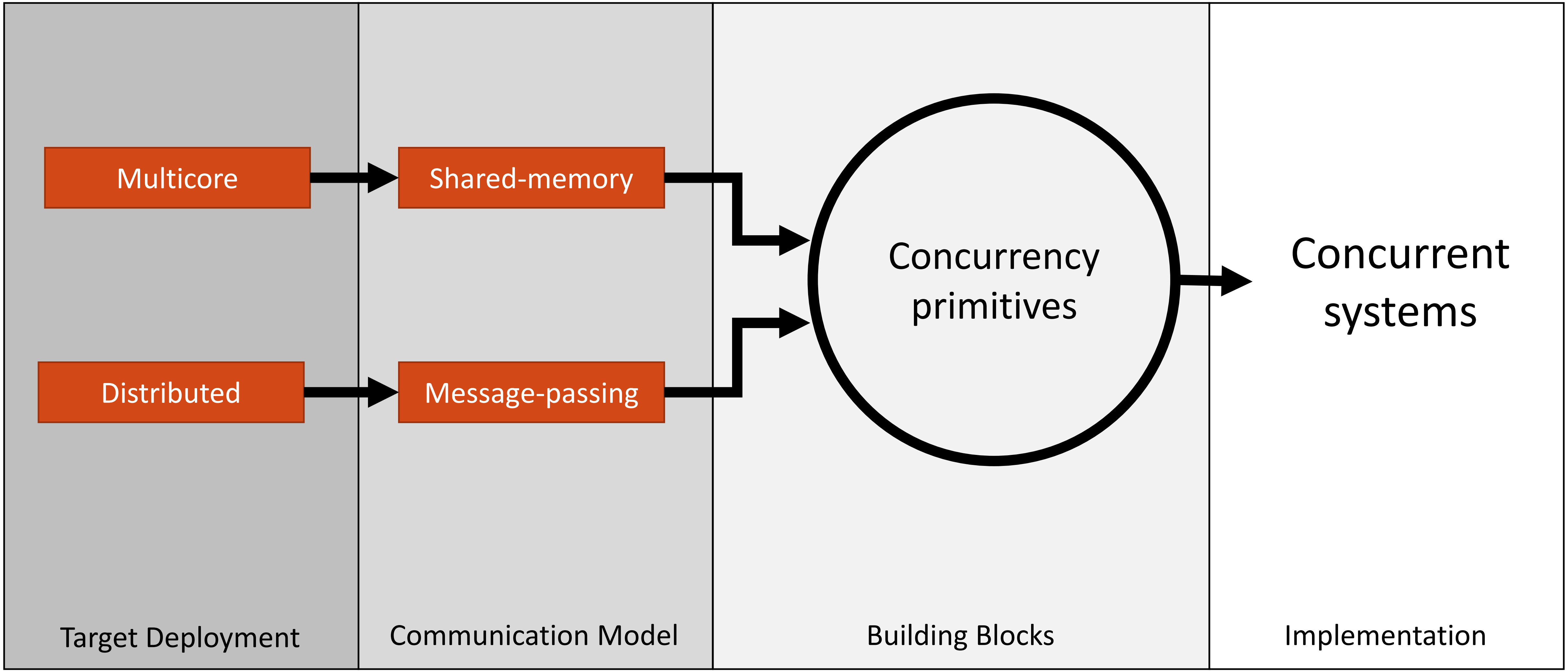
Slow!

→

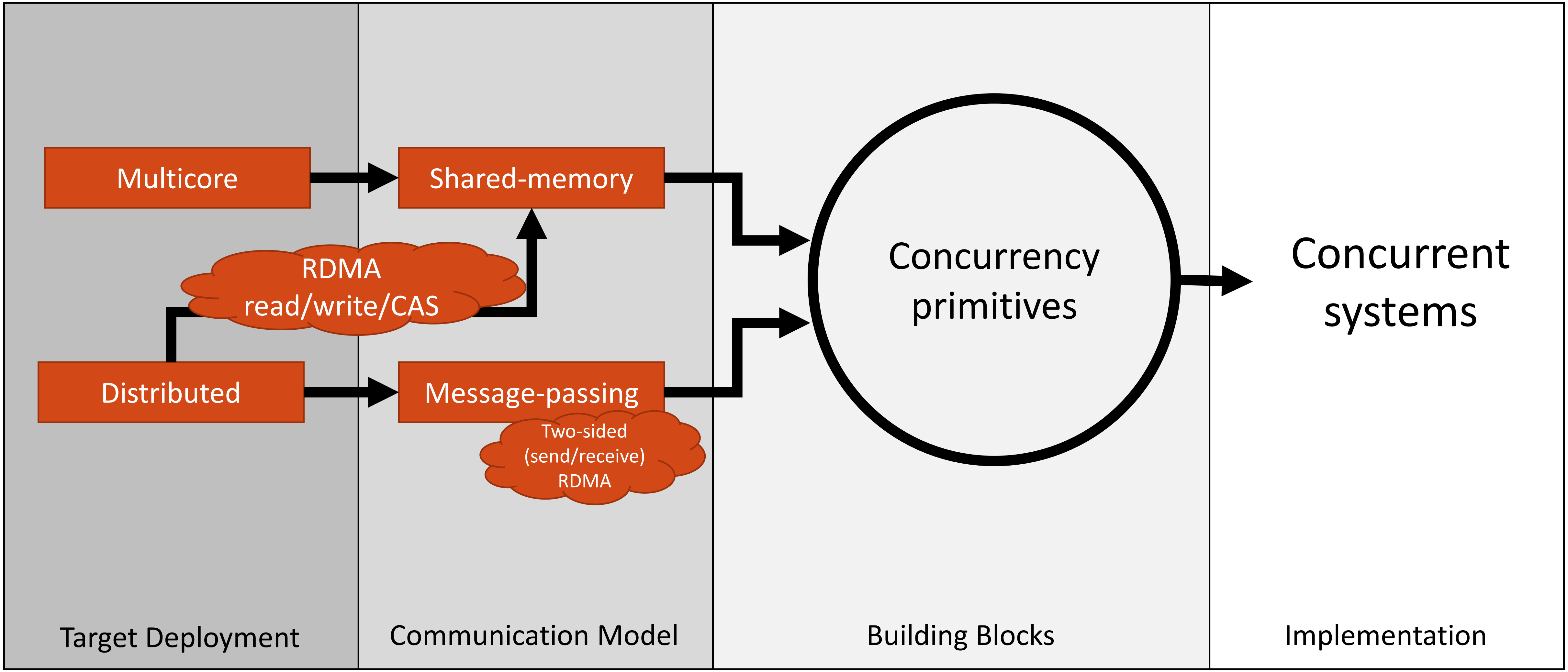
RDMA

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

A Traditional View of Concurrent Systems



A Modern View of Concurrent Systems



Problem solved!

Shared-memory application

RDMA one-sided operations: READ/WRITE/CAS

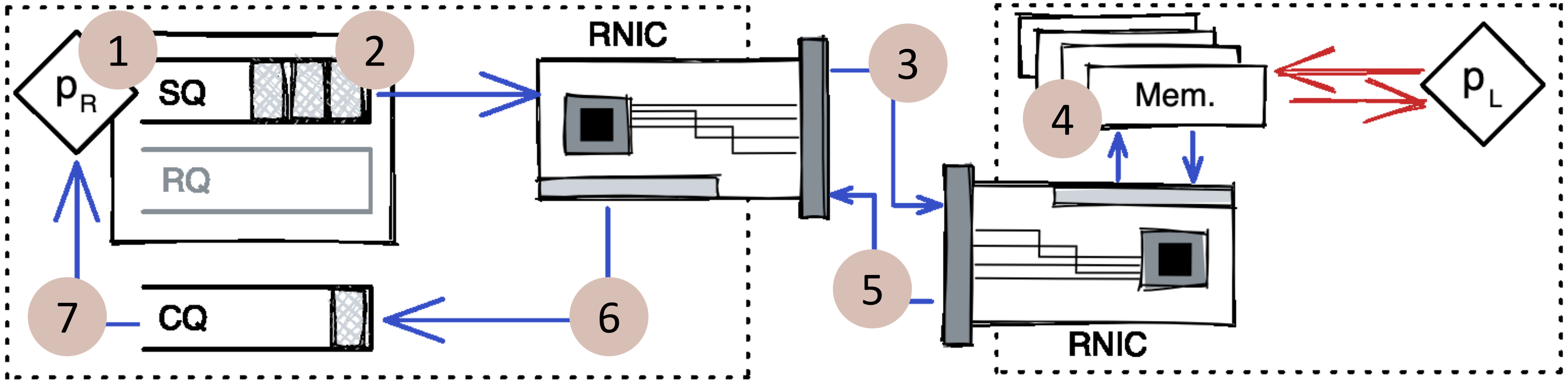
Distributed application!

Problem !solved

- What if the system size grows beyond tens of nodes?
- Is topology important?
- What if processes access RDMA memory with shared-memory API?
- What's the NUMA effect on RDMA operations?
- 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?
- How do we reclaim remote memory efficiently?

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)



$\boldsymbol{p_R}$: Remote process using RDMA
$\boldsymbol{p_L}$: Local process using native access

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

Our TO-DO list

- Studying the performance implications of RDMA and NUMA [SRDS'15]
- A new lock primitive to synchronize local (shared-memory) processes and remote (RDMA) processes [SPAA'24]
- An open-source library for programmers to develop RDMA-enabled applications and systems that use one-sided operations.
  - Remus (*https://github.com/sss-lehigh*)
- A new RDMA-aware object that enabled non-blocking remote traversals [BA SPAA'24]

In progress…

- Studying the impact of network topology on RDMA scalability
- Designing RDMA-aware directory for memory object moment
- RDMA and GPUs, a unified framework to develop distributed heterogeneous data structures

SCALABLE SYSTEMS & SOFTWARE RESEARCH GROUP

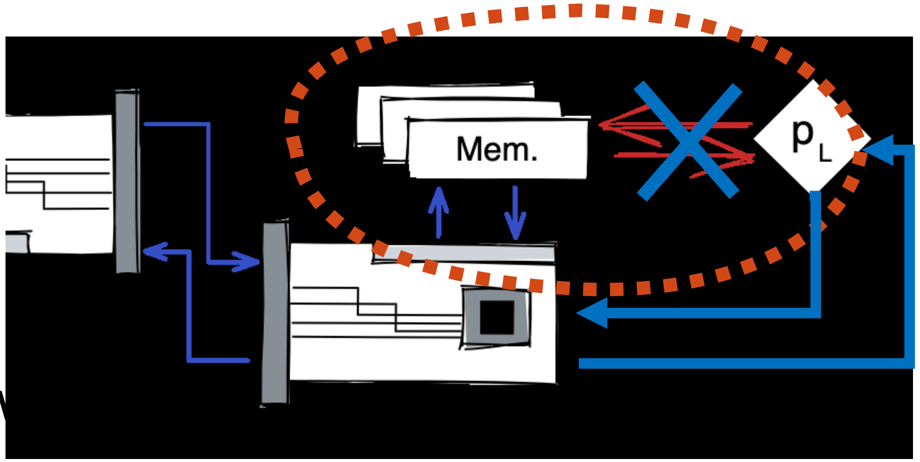
ALock: Asymmetric Lock Primitive for RDMA Systems

[SPAA ’24]

Amanda Baran, Jacob Nelson-Slivon, Lewis Tseng, Roberto Palmieri

Mutual Exclusion in the absence of global atomicity

- Shared-memory CAS and RDMA CAS are not atomic with each other!
- **Solution:**
  - Local workload uses RDMA loopback
- **Problem:**
  - Saturation of RDMA loopback
    - side-effect -> whole distributed system can slow



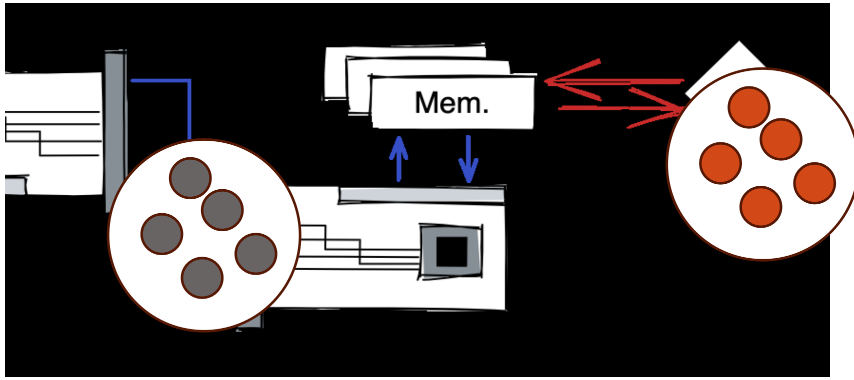
Access (8B)		Remote (RDMA)		
		Read	Write	CAS
Local	Read	Yes	Yes	Yes
	Write	Yes	Yes	No
	RMW	Yes	Yes	No

Goal:

- Local processes should use shared memory operations
- Remote processes should limit the number of RDMA operations

Inspiration

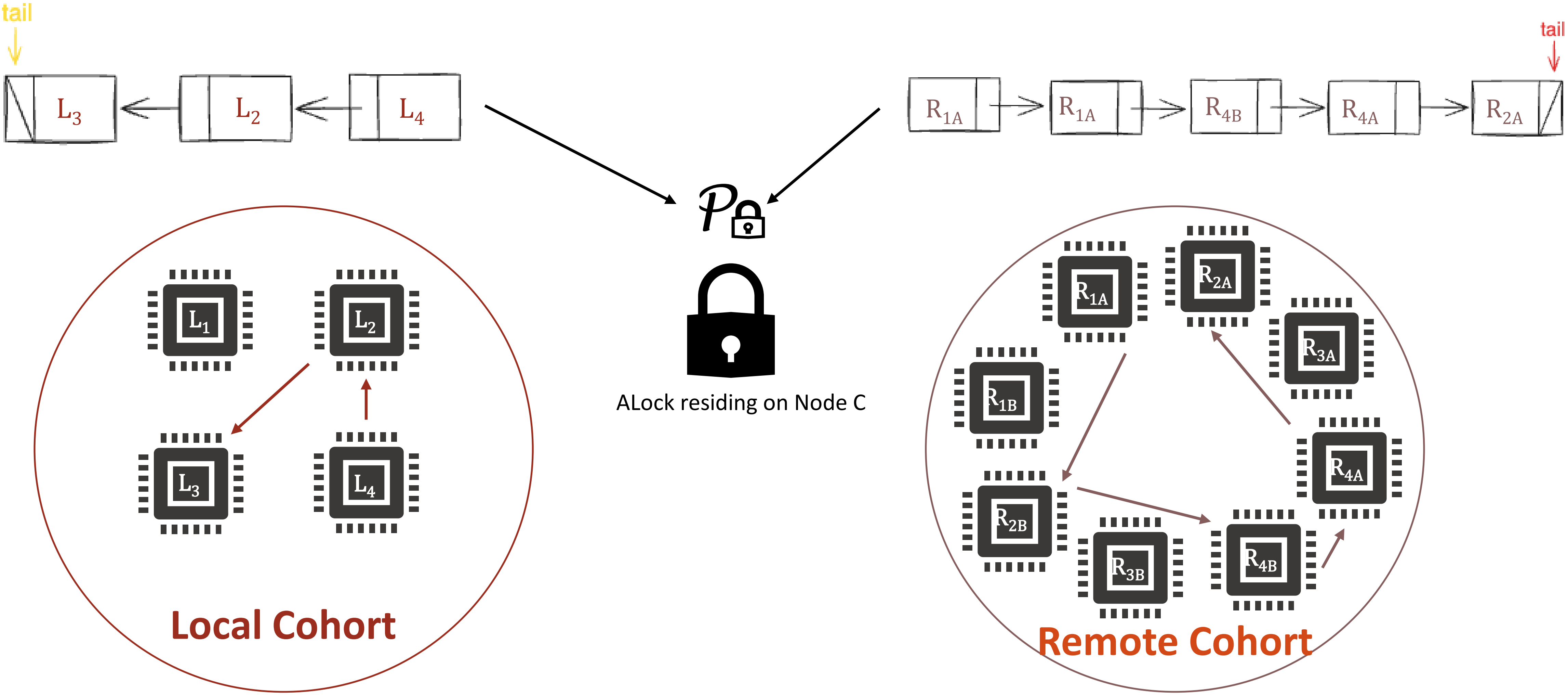
- Lock Cohorting
  - Hierarchical lock was originally used in systems with NUMA-like behaviors
  - Processes who behave similarly (a cohort) compete amongst themselves first
  - Leaders of each cohort compete for the global lock



- Peterson’s Algorithm
  - Two-process mutual exclusion algorithm using only atomic read/write operations

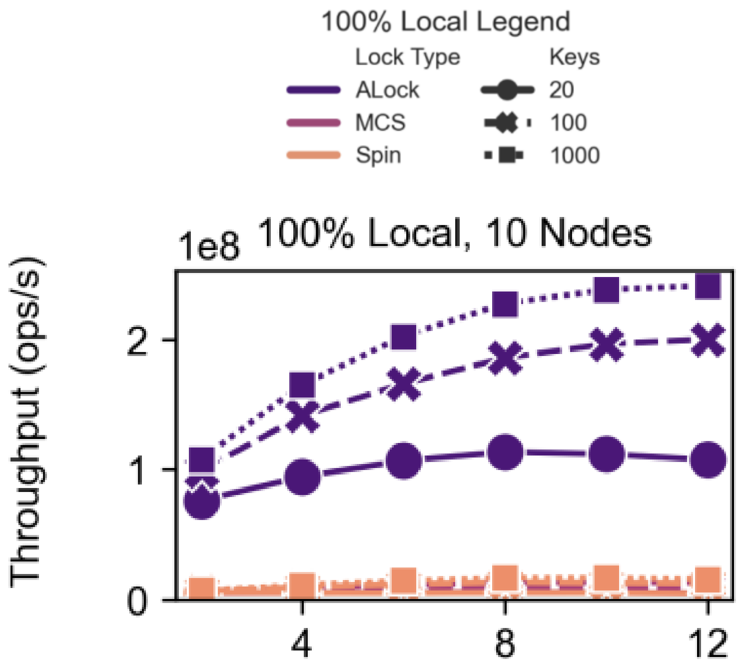
Access (8B)		Remote (RDMA) Read	Remote (RDMA) Write	Remote (RDMA) CAS
Local	Read	Yes	Yes	Yes
Local	Write	Yes	Yes	No
Local	RMW	Yes	Yes	No

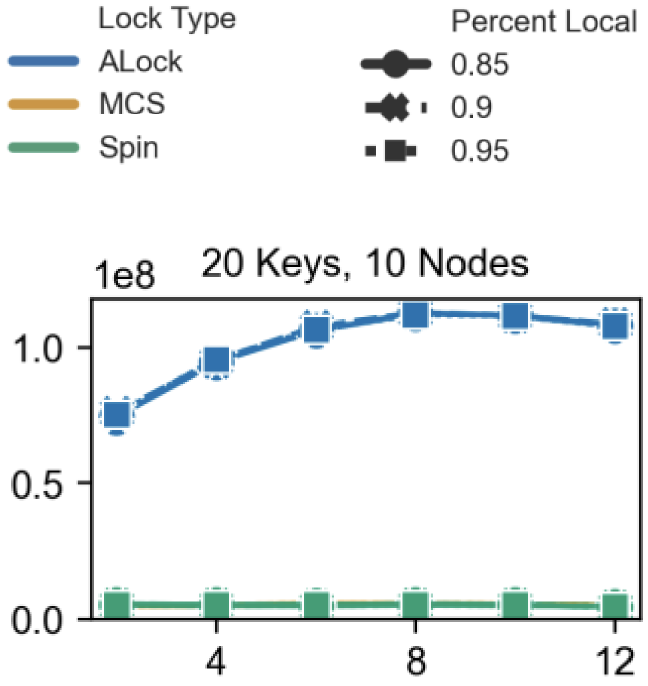
Lock Cohorting + Peterson's Algorithm



ALock Performance

- High performance in both high locality and with high contention





ROMe: Remote Object Memory

[BA SPAA'24]

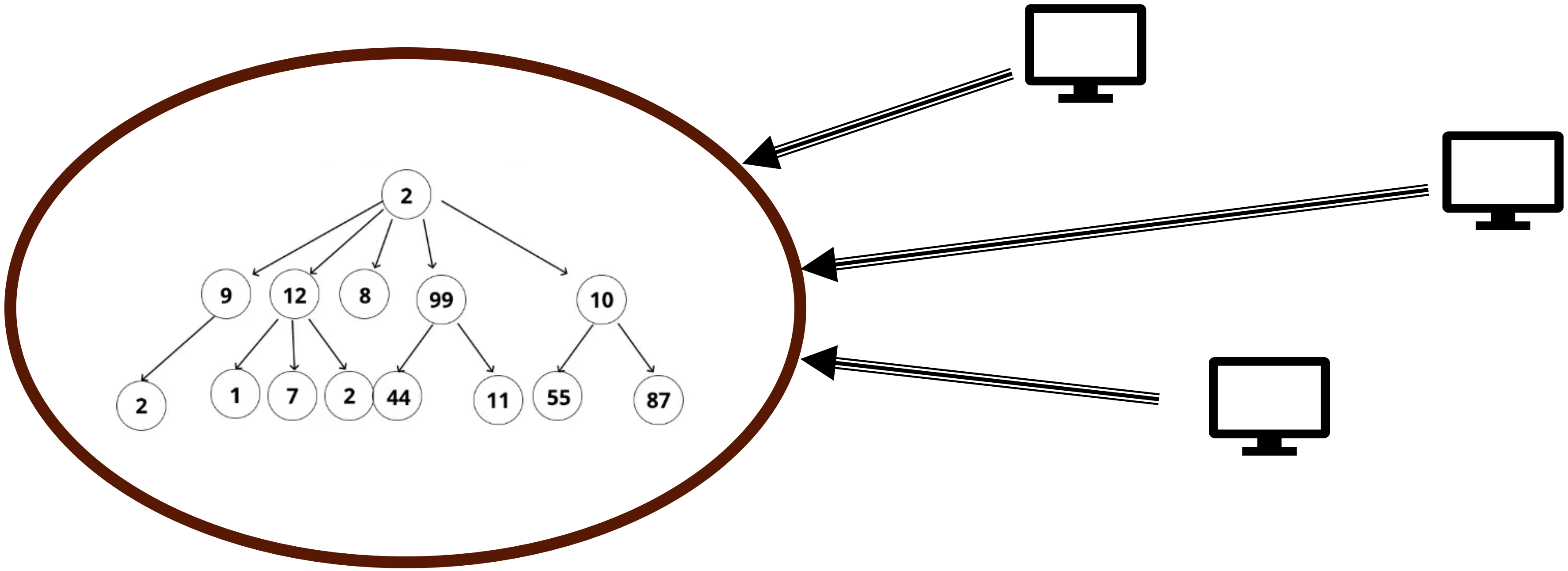
Jacob Nelson-Slivon, Reilly Yankovich, Ahmed Hassan, Roberto Palmieri

SCALABLE SYSTEMS
& SOFTWARE
RESEARCH GROUP

LEHIGH UNIVERSITY

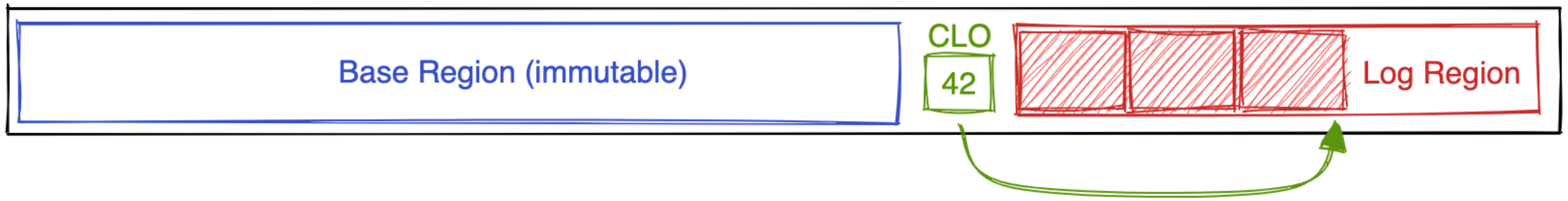
Motivating question

- Can we design a semantic object to allow remote threads to perform consistent non-blocking range queries?



- Problem:
  - Local and remote writes/reads are consistent <u>only within one cache line</u>

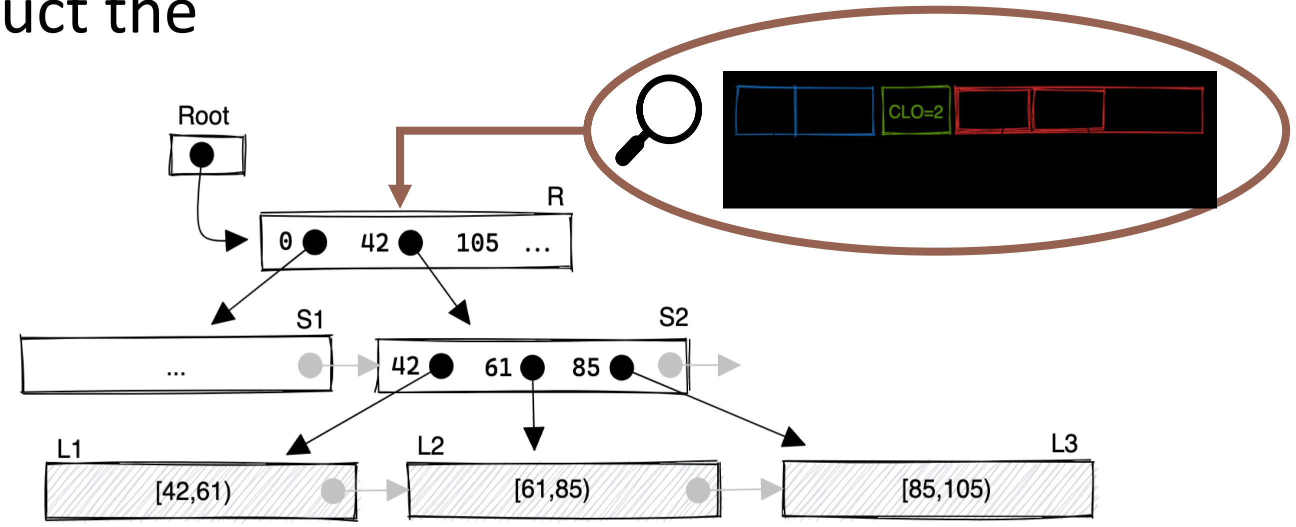
The ROMe object



- Base Region
  - Immutable memory region representing initial state
- Active Region
  - Current Log offset (CLO) and other user-defined metadata to be updated in place
- Log region
  - Updates to base region
- Supplemental region
  - Any additional metadata required for synchronization

ROMe-KV

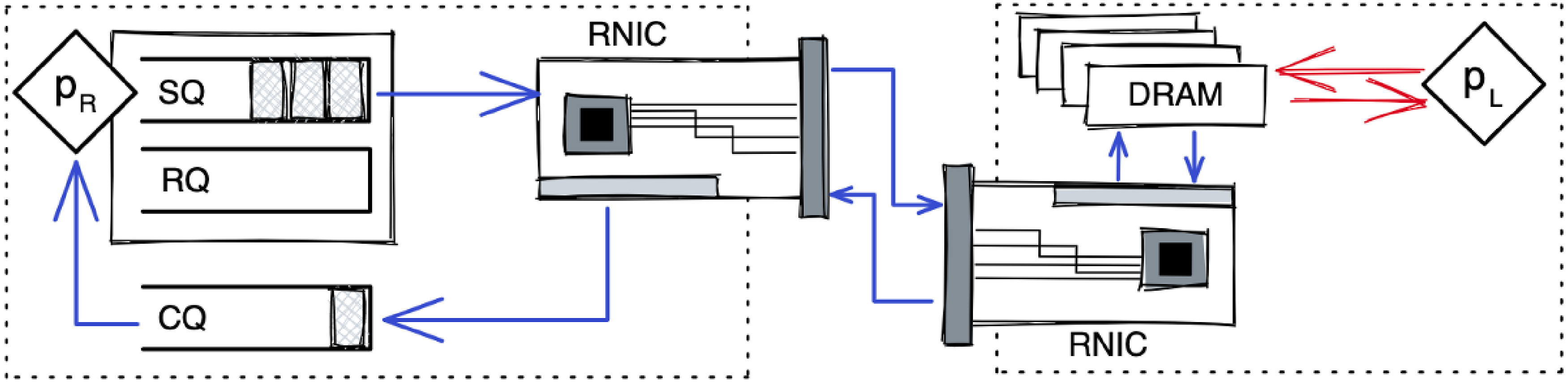
- Writes
  - Remote writes are sent to the host machine, which performs them locally
  - Set metadata for reads to reconstruct the current state of memory
- Reads
  - Remote reads use RDMA
  - Local reads avoid RDMA entirely
  - It requires exactly two RDMA read operations regardless of size
    - One to read the metadata set (must fit one cache line) by the writers
    - One to read the object itself



What about RDMA scalability?

Queue Pair (QP) Thrashing

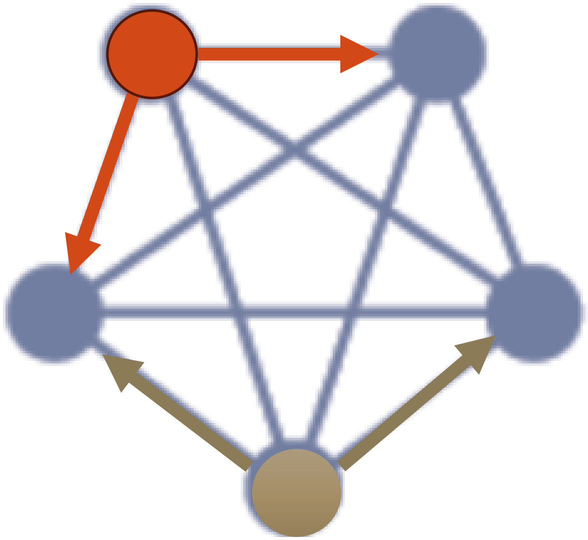
- QP information is needed to serve RDMA requests
- RNIC has a relatively small on-card memory
  - RNIC must fetch QP info from memory if not cached
  - If no room, then RNIC must evict something → thrashing



Performance degrades even further with QP sharing!

What if we use a preferred topology instead?

- Processes can still connect to all but they are likely to access memory allocated on a subset of nodes
- This set of *preferred nodes* per node will likely be cached in the RDMA internal memory
- The system size can grow, as long as the set of preferred nodes stays small



- The system should be designed so that:
  - Threads could use any QP to issue RDMA operations
  - BUT they should be accessing memory only from the preferred node for the most part

= NUMA?

NU(R)MA = Non-Uniform Remote Memory Access

- Extends (?) the idea of NUMA to include remote memory accesses
- NU(R)MA-aware programming means tailoring remote accesses to minimize the number of QPs a node needs to communicate
- Upon a memory access
  - If local
    - Follow NUMA-aware design
    - Use shared-memory APIs to synchronize (e.g., ALock)
  - If remote
    - Use RDMA one-sided operations
    - Memory should be on a preferred node
      - If not, a more expensive remote operation should be performed

Thanks! & Questions?

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

GitHub repo:
https://github.com/sss-lehigh