

rmalloc() and rpipe() - a uGNI-based Distributed Remote Memory Allocator and Access Library for One-sided Messaging

Udayanga Wickramasinghe
Indiana University

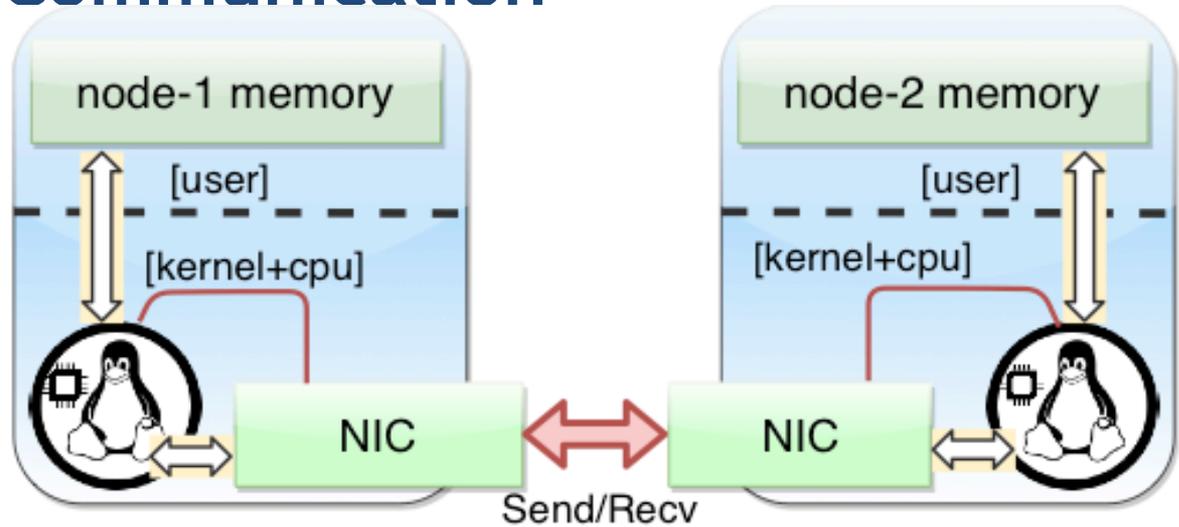
Andrew Lumsdaine
Pacific Northwest National Laboratory

Overview

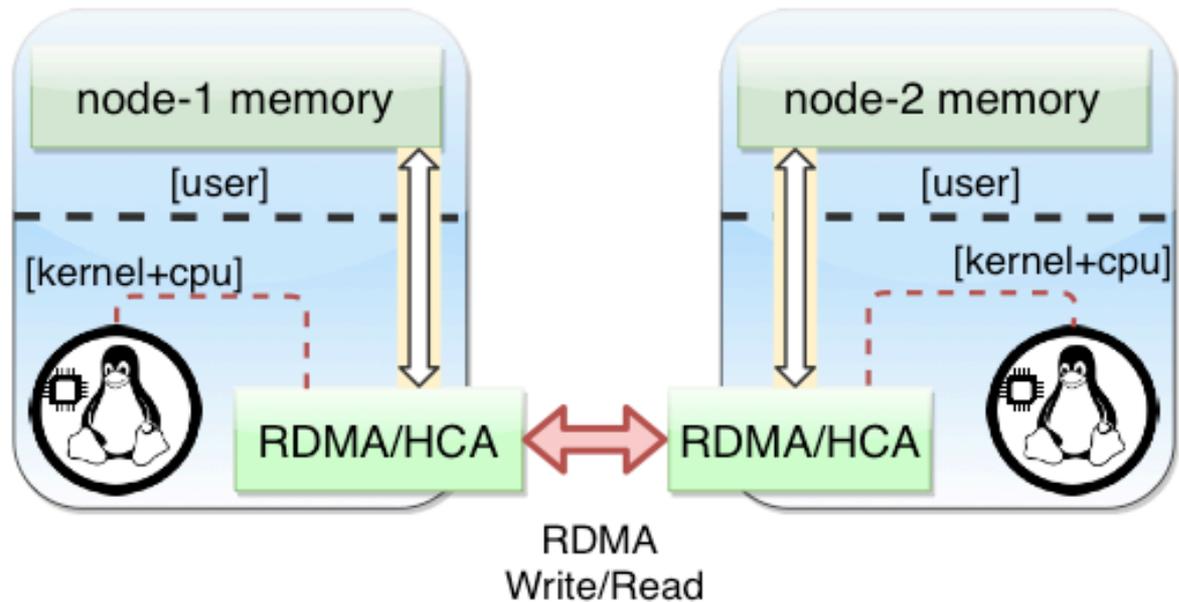
- Motivation
- Design/System Implementation
- Evaluation
- Future Work

RDMA Network Communication

Network Op
Kernel+CPU direct

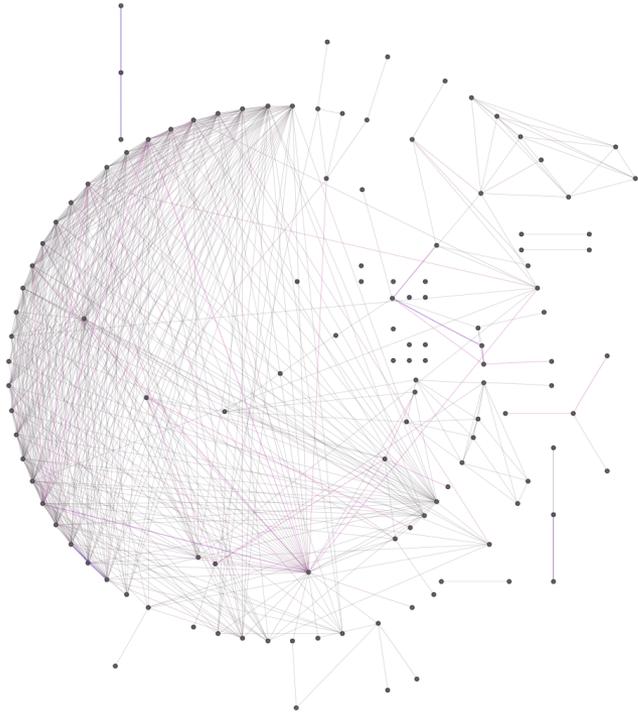


RDMA
Kernel+CPU bypass
Zero Copy



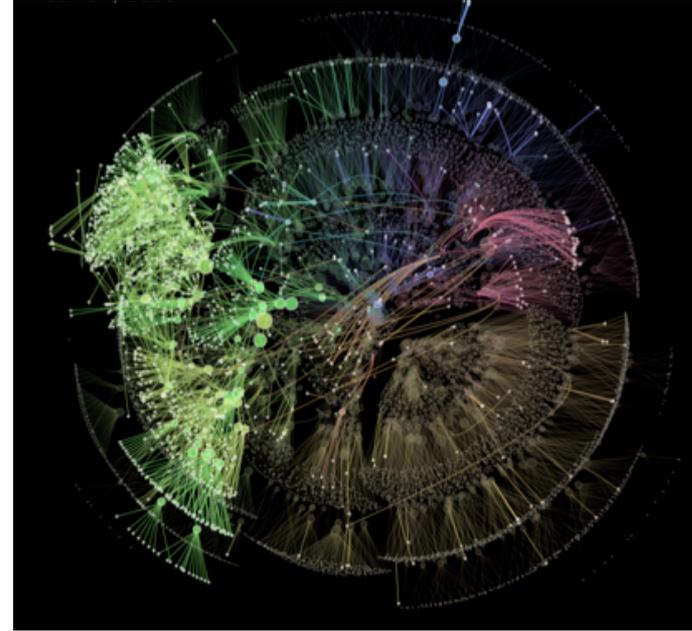
Designed for one-sided communication!!

One-sided Communication



Advantages

- Great for Random Access + Irregular Data patterns
- Less Overhead/High Performance

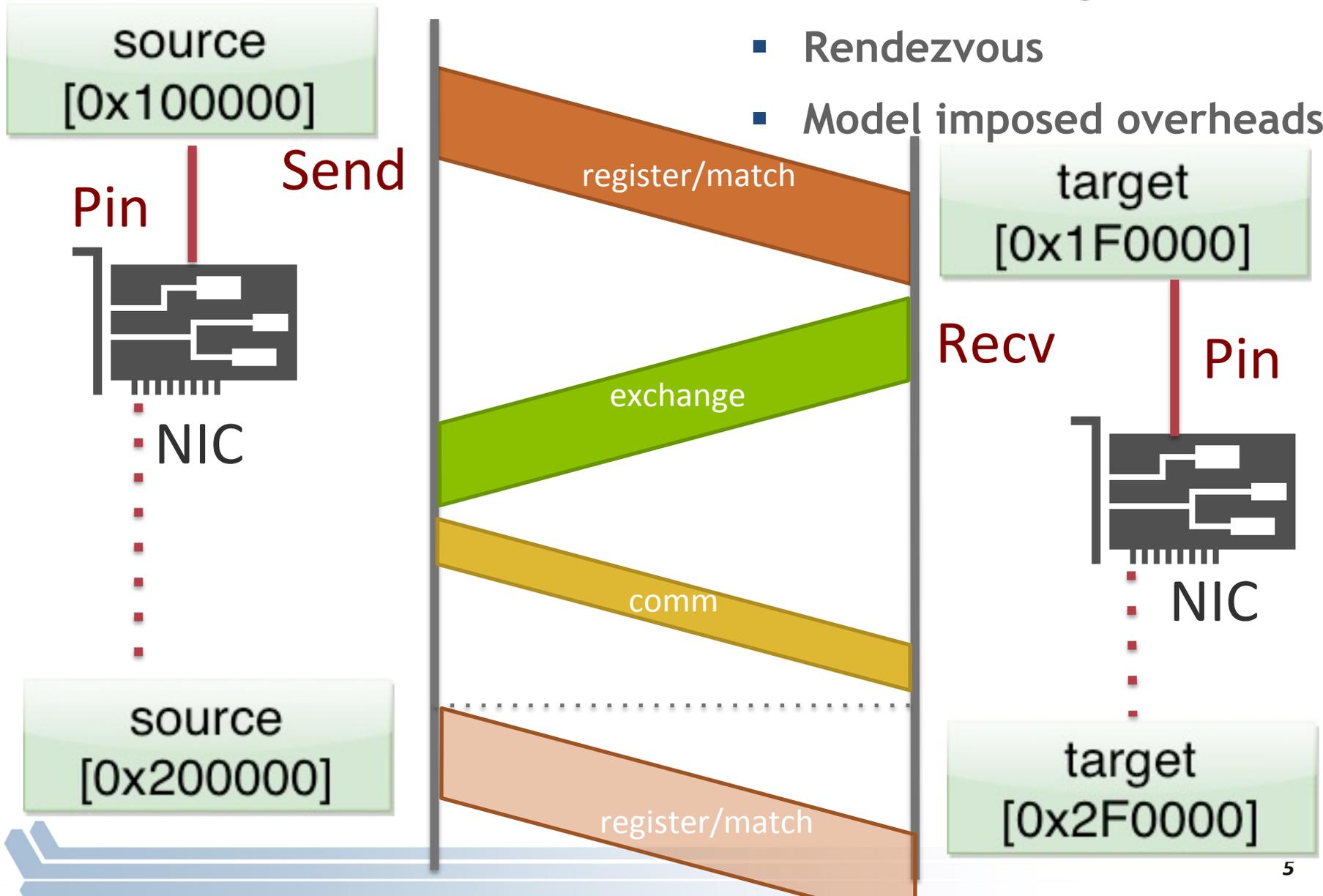


Disadvantages

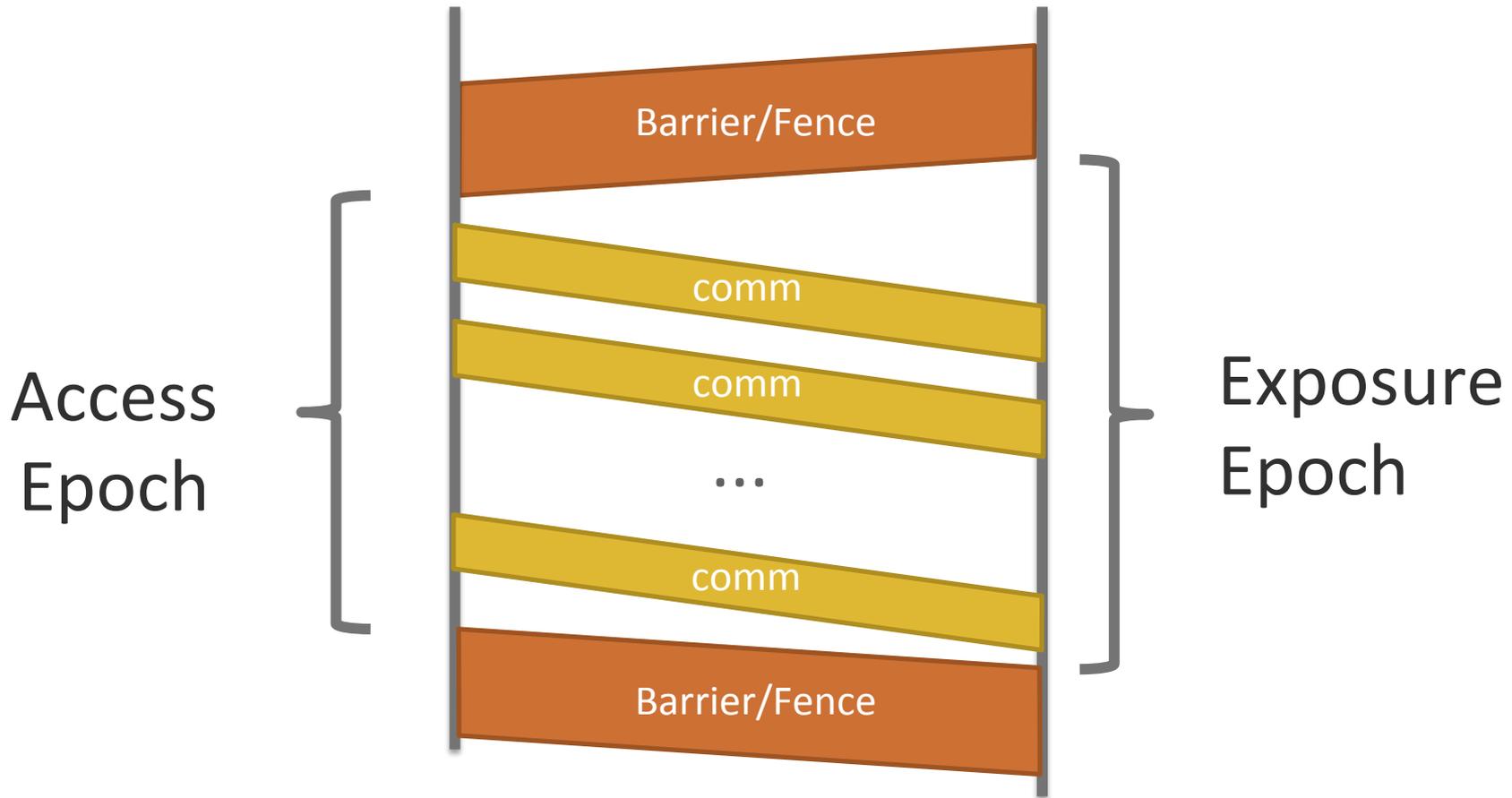
- Explicit Synchronization – **separate from data-path!!**

RDMA Challenges – Communication

- Buffer Pin/Registration
- Rendezvous
- Model imposed overheads

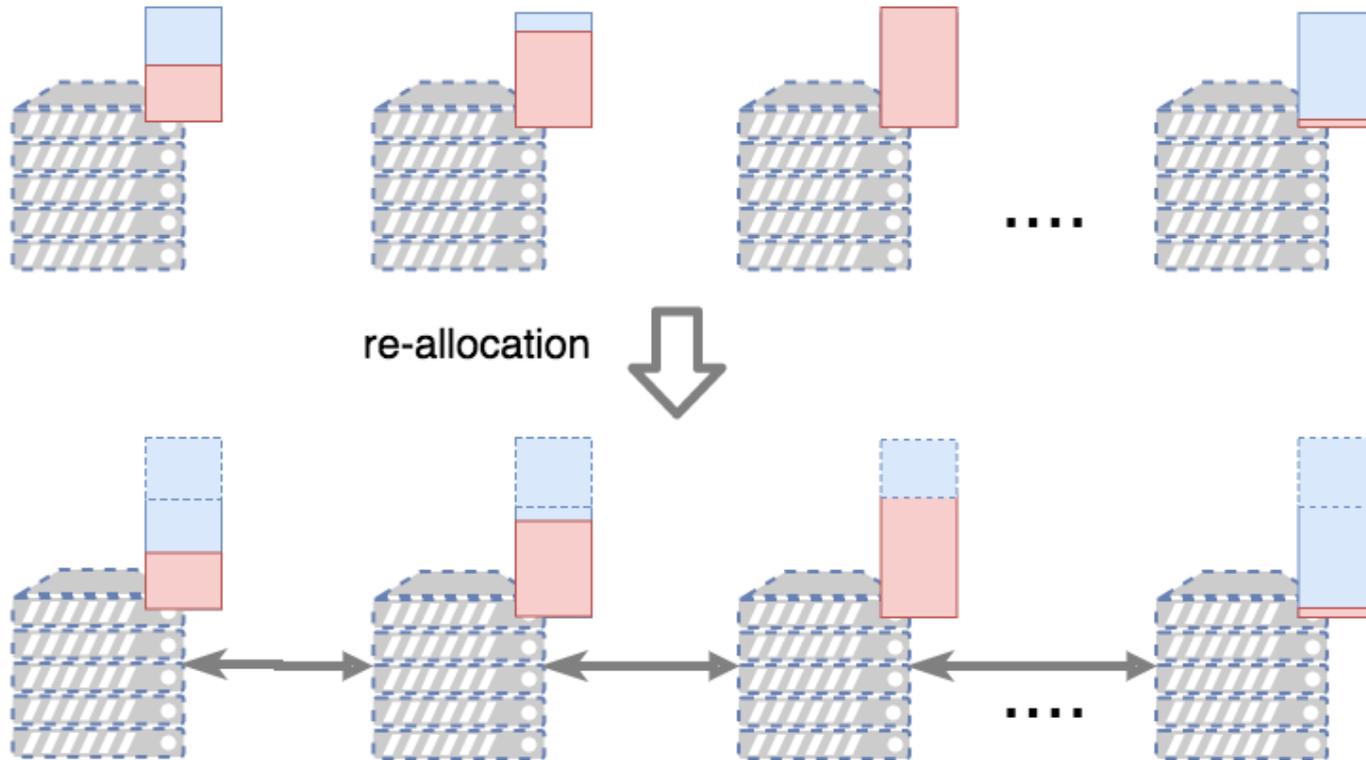
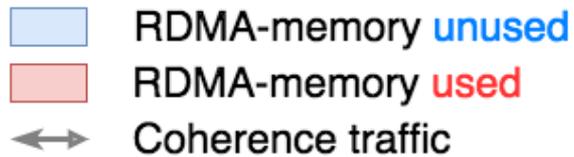


RDMA Challenges – Synchronization



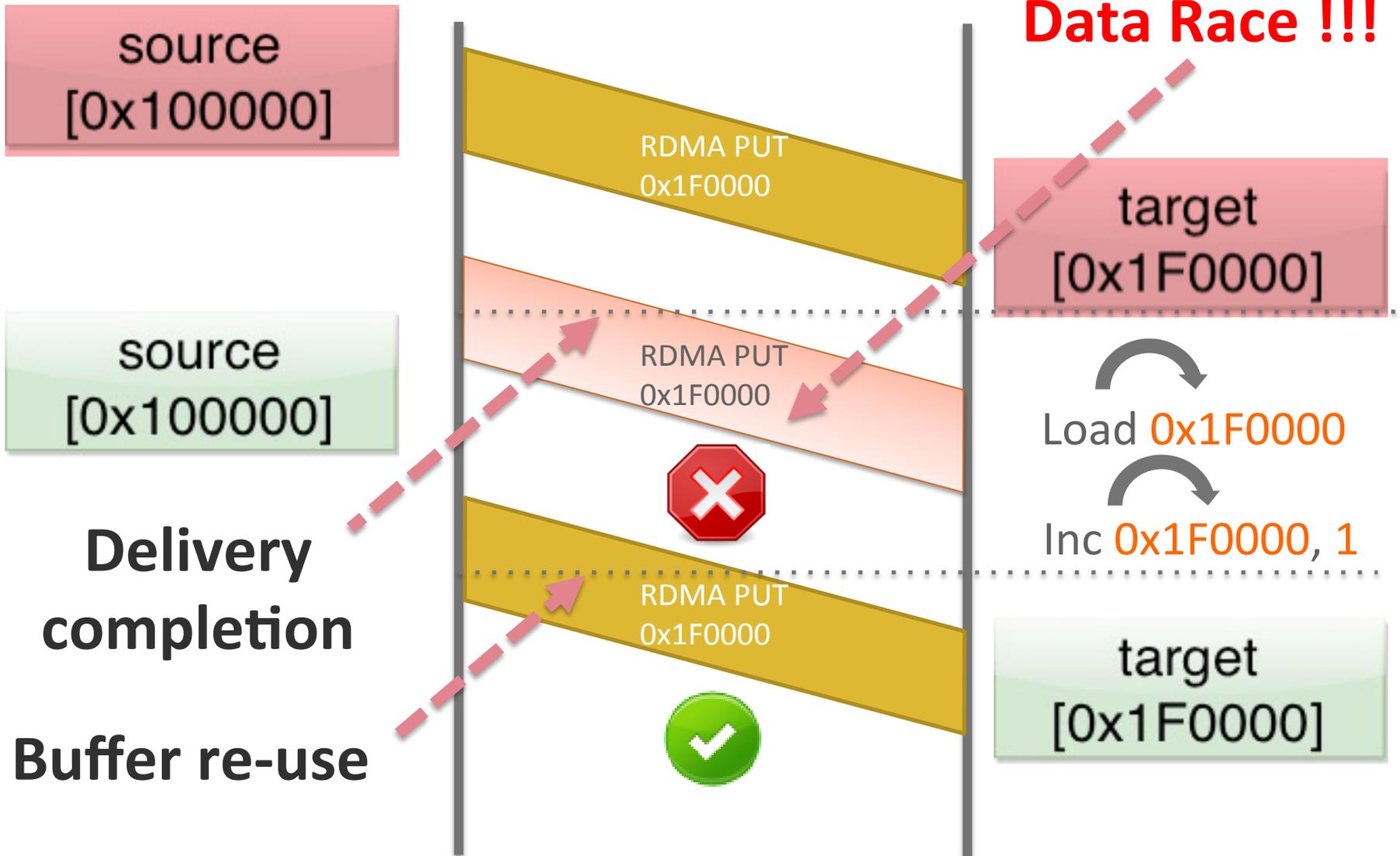
How to make reads and updates visible ? “in-use”/”re-use”

RDMA Challenges – Dynamic Memory Management



Cluster wide allocations → costly in a dynamic context i.e. PGAS

RDMA Challenges – Programming



Challenges – Programming

- Enforcing “in-use”/”re-use” semantics
 - Flow Control – Credit based, Counter based, polling (CQ based)
- Enforcing Completion semantics
 - MPI 3.0 Active/Passive – **barriers, fence, lock, unlock, flush**
 - GAS/PGAS based (SHMEM, X10, Titanium) – **futures, barriers, locks, actions**
 - GASNet like (RDMA) Libraries – **user has to implement**
- **Explicit and Complex to implement for applications !!**

Challenges – Summary

- Low overhead, high-throughput communication?
 - Eliminate unnecessary **overheads**.
- Dynamic On-demand RDMA Memory?
 - Allocate/de-Allocate with heuristics support.
 - Less **coherence** Traffic and may be better **utilization**
- Scalable Synchronization?
 - Completion and Buffer **in-use/re-use**.
- RDMA Programming abstractions for applications?
 - No **explicit** synchronization – Let middleware **transparently** handle it.
 - Expose **light-weight** RDMA ready memory and operations.

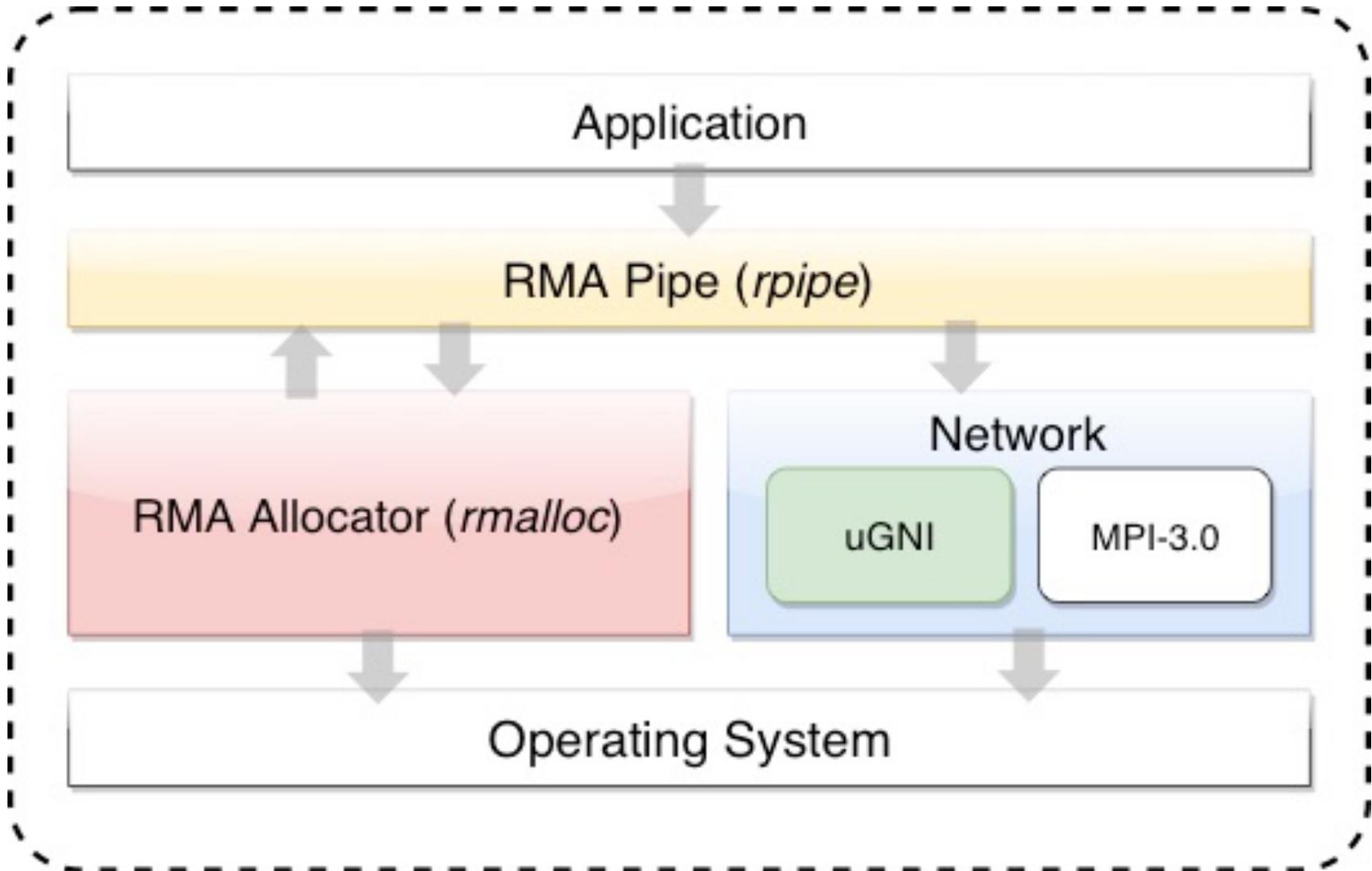
How `rmalloc()/rpipe()` meets these Challenges ?

Problem	Key Idea
Low Communication Overhead	Fast Path(MMIO vs Doorbell) Network Operation (in <code>uGNI</code>) with synchronized updates.
Dynamic RDMA Memory Mgmt	Per endpoint RDMA Dynamic Heap → Heuristics + Asymmetric Allocation
Synchronization	Notification Flags with Polling (<code>NFP</code>)
Programmability	A familiar Two-level Abstraction → allocator (<code>rmalloc</code>) + stream like channel(<code>rpipe</code>) → No explicit synchronization

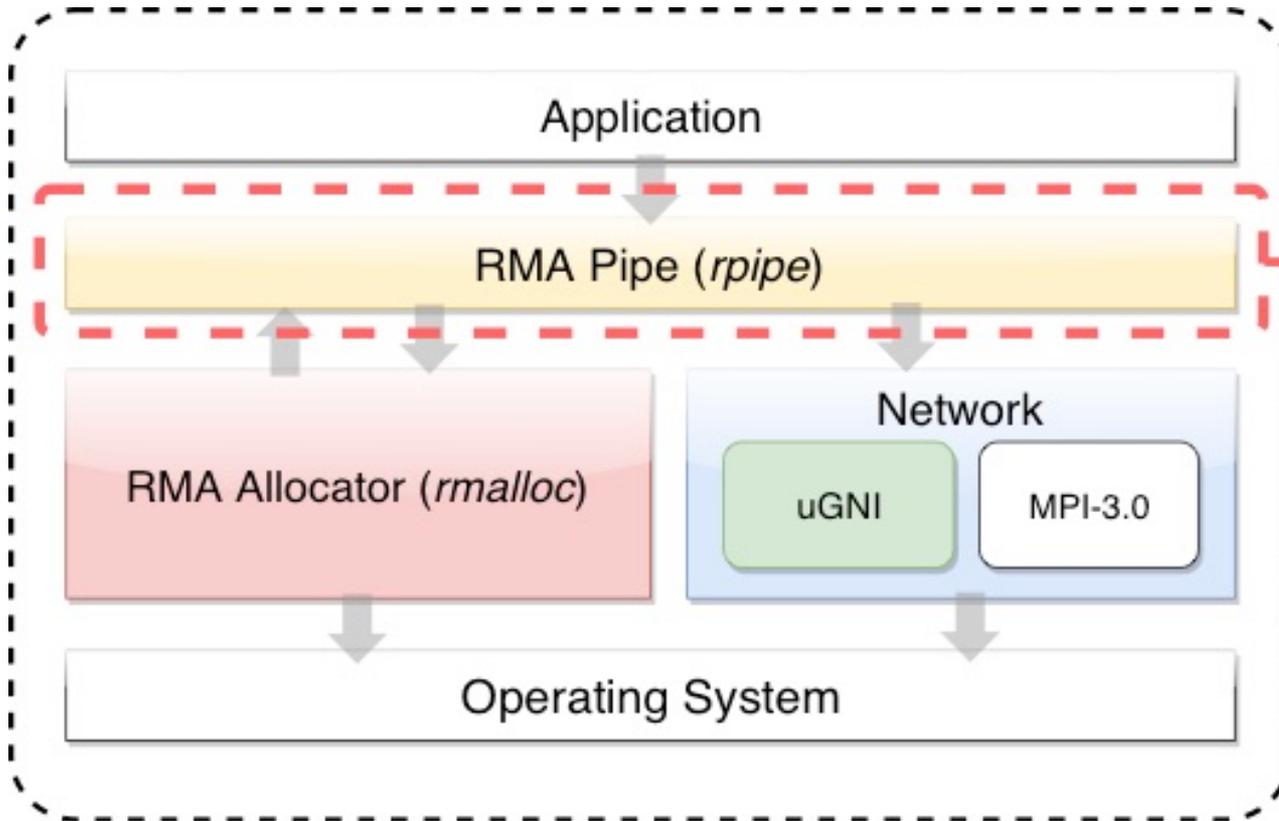
Overview

- Motivation
- Design/System Implementation
- Evaluation
- Future Work

System Overview



System Overview



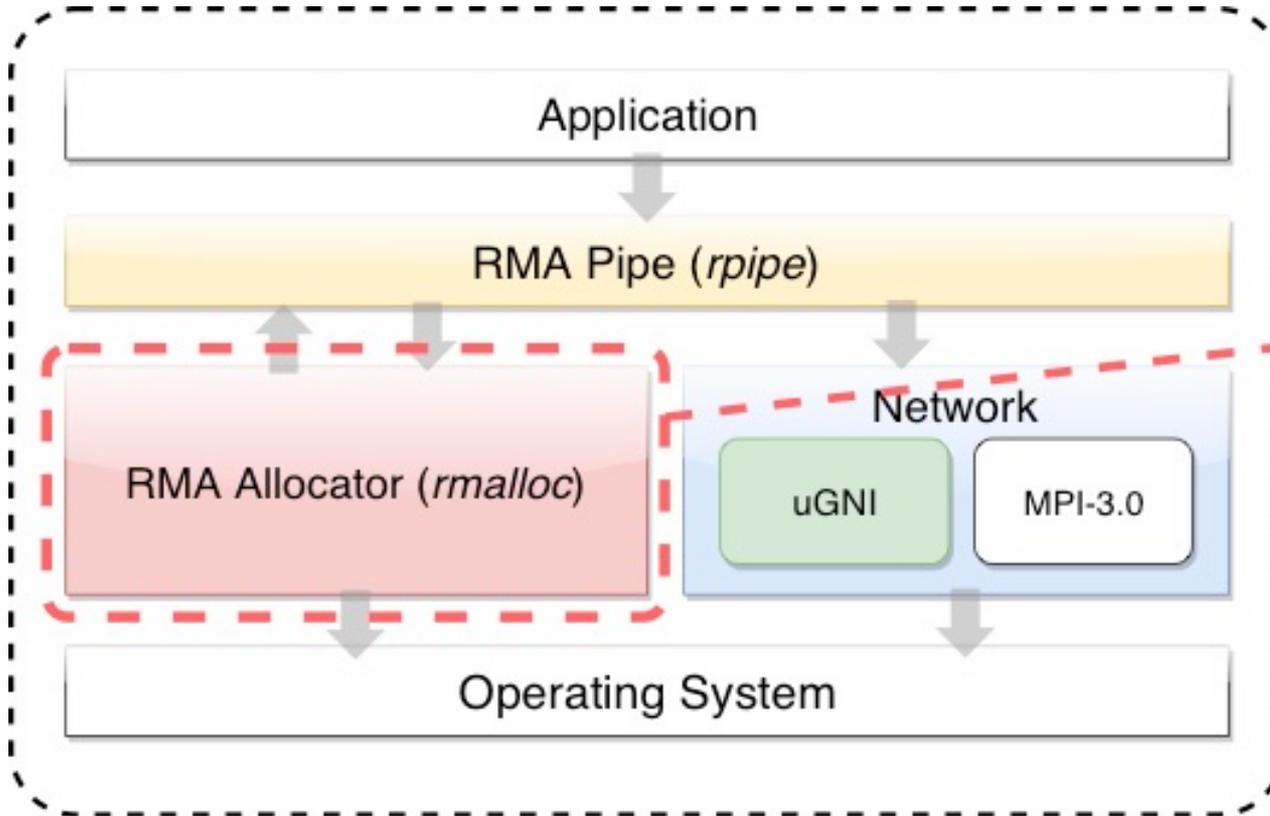
High Performance RDMA Channel

- Expose Zero-copy RDMA ops
- Interface/s
 - `rread()`
 - `rrwrite()`

Enable Implicit Synchronization

- NFP (Notified Flags with Polling)

System Overview



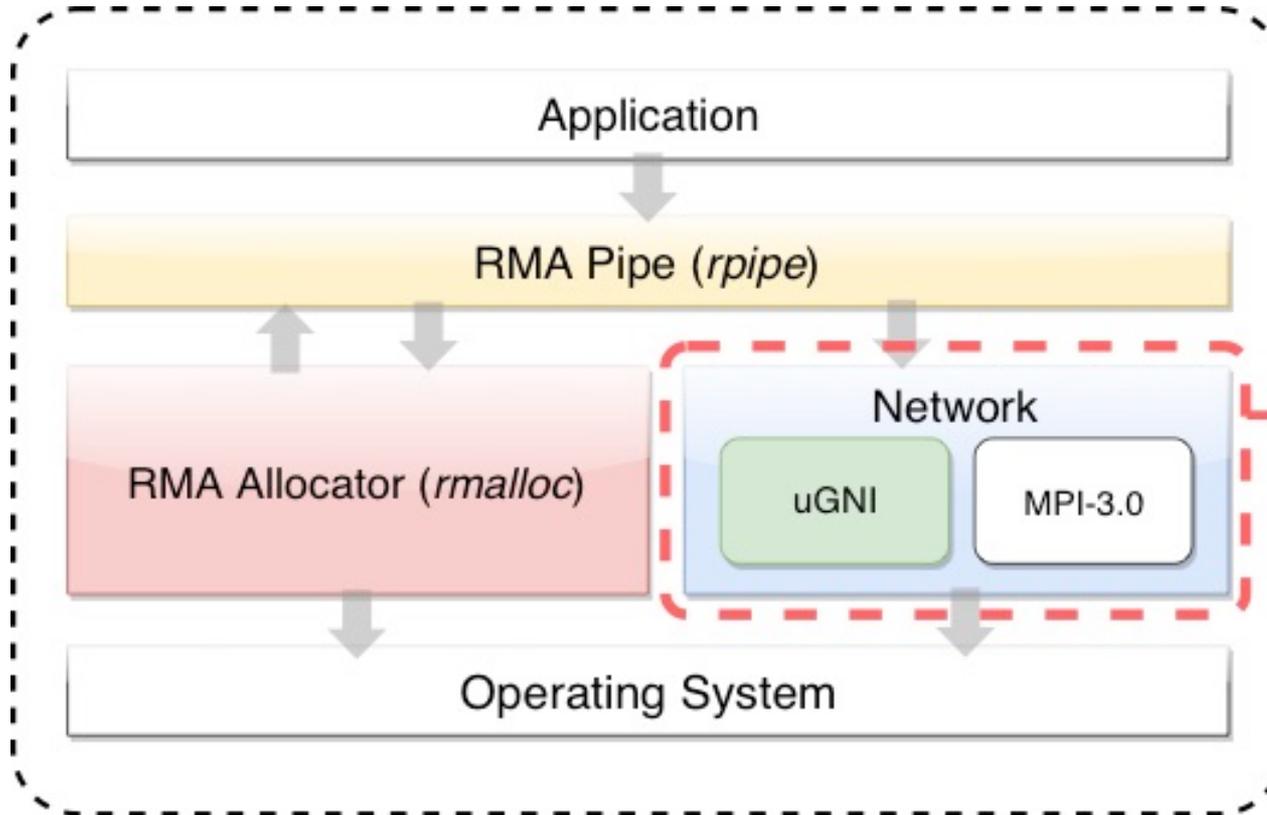
Allocates RDMA memory

- Returns Network Compatible Memory
- **Dynamic Asymmetric Heap for RDMA**
- Interface/s
 - `rmalloc()`

Allocation policies

- Next-fit, First-fit

System Overview



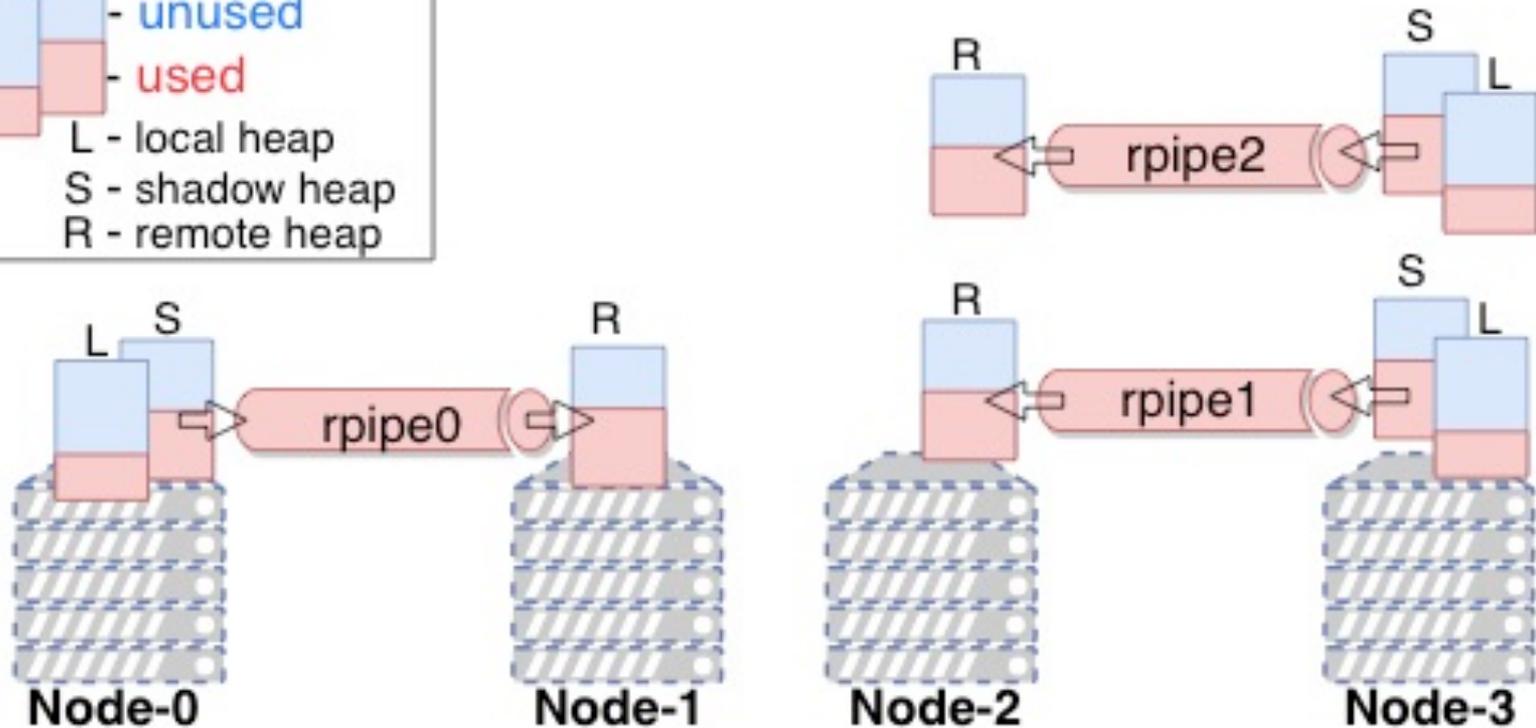
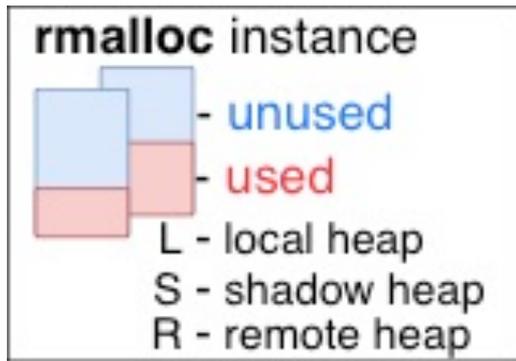
Network Backend

- Cray specific – uGNI
- MPI 3.0 based (portability layer)

Cray uGNI

- FMA/BTE Support
- Memory Registration
- CQ handling

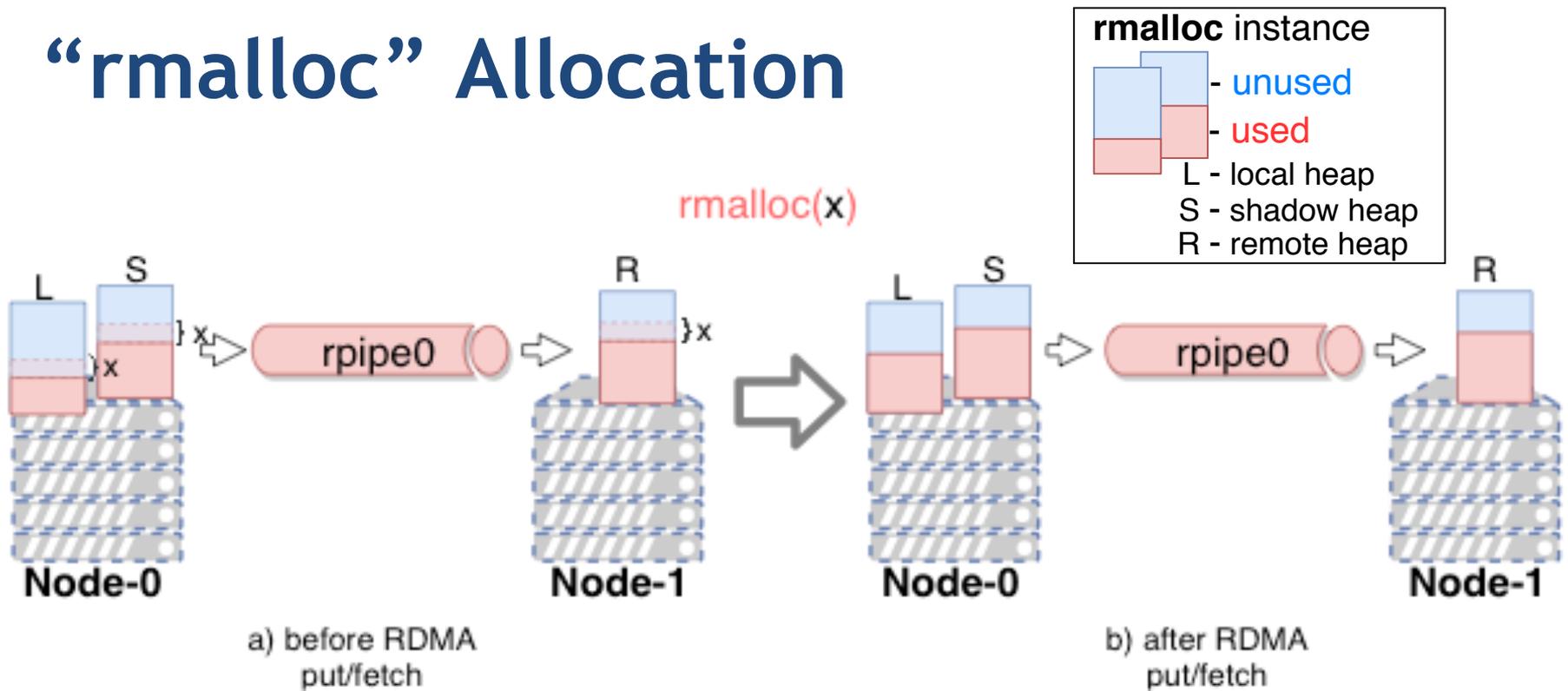
“rmalloc”



Asymmetric heaps across cluster

- 0 or more for each endpoint pair
- dynamically created

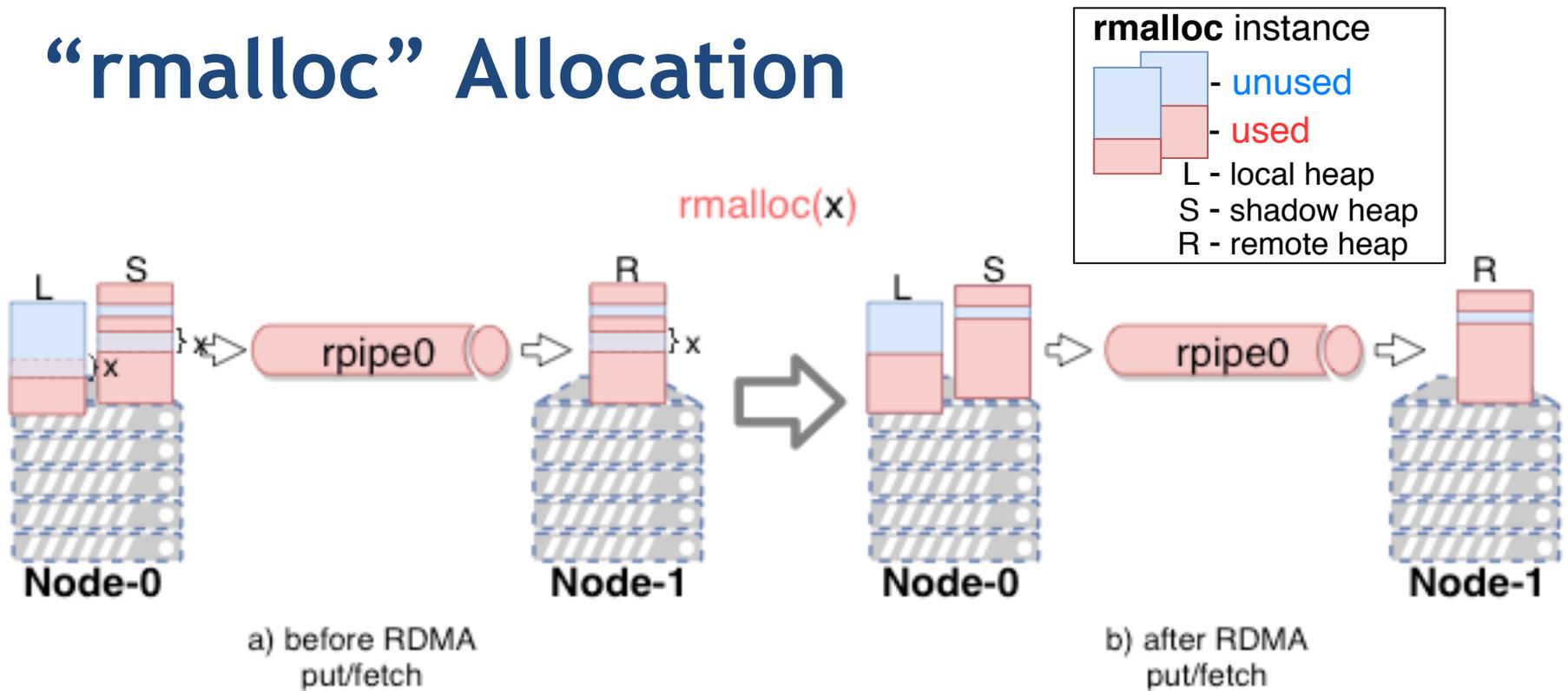
“rmalloc” Allocation



Next-fit heuristic – return **next** available RDMA heap segment

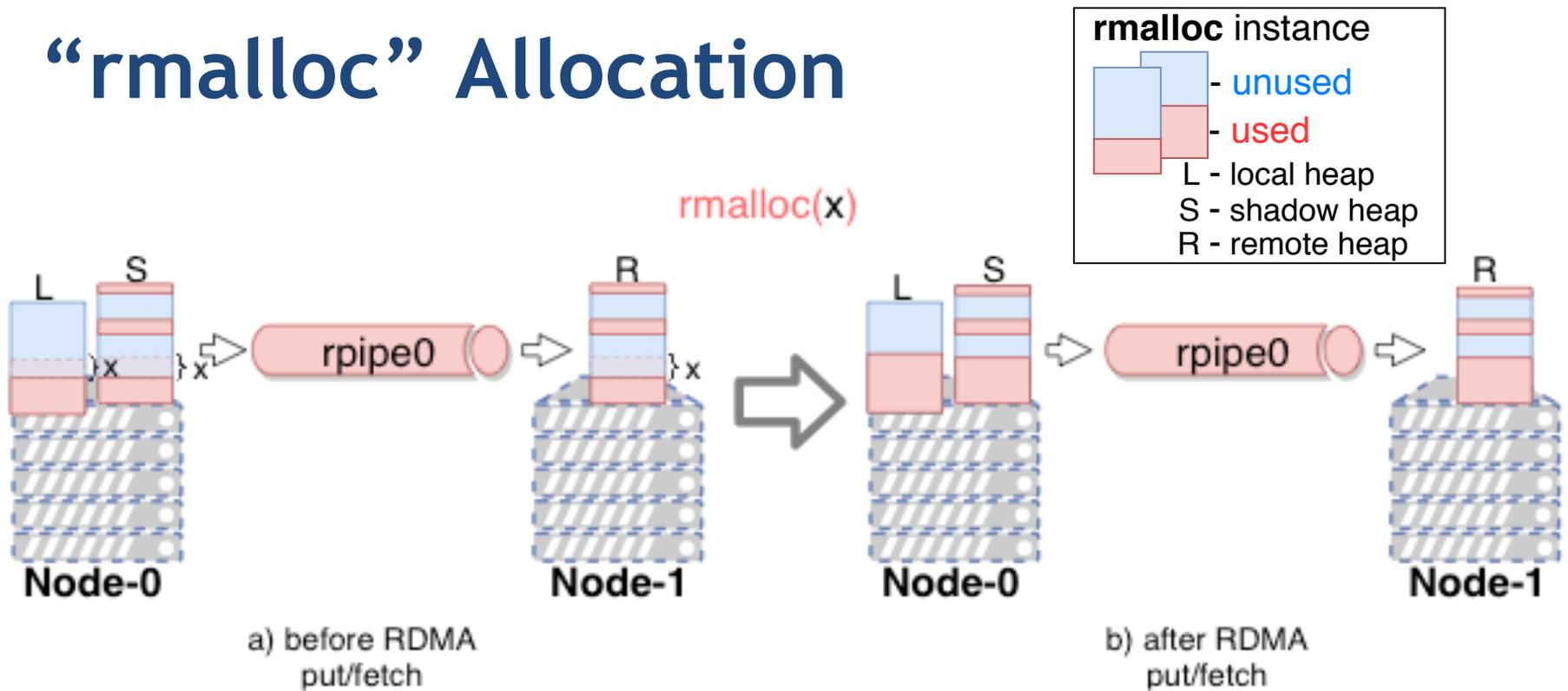
Synchronization → a special **bootstrap** rpipe

“rmalloc” Allocation



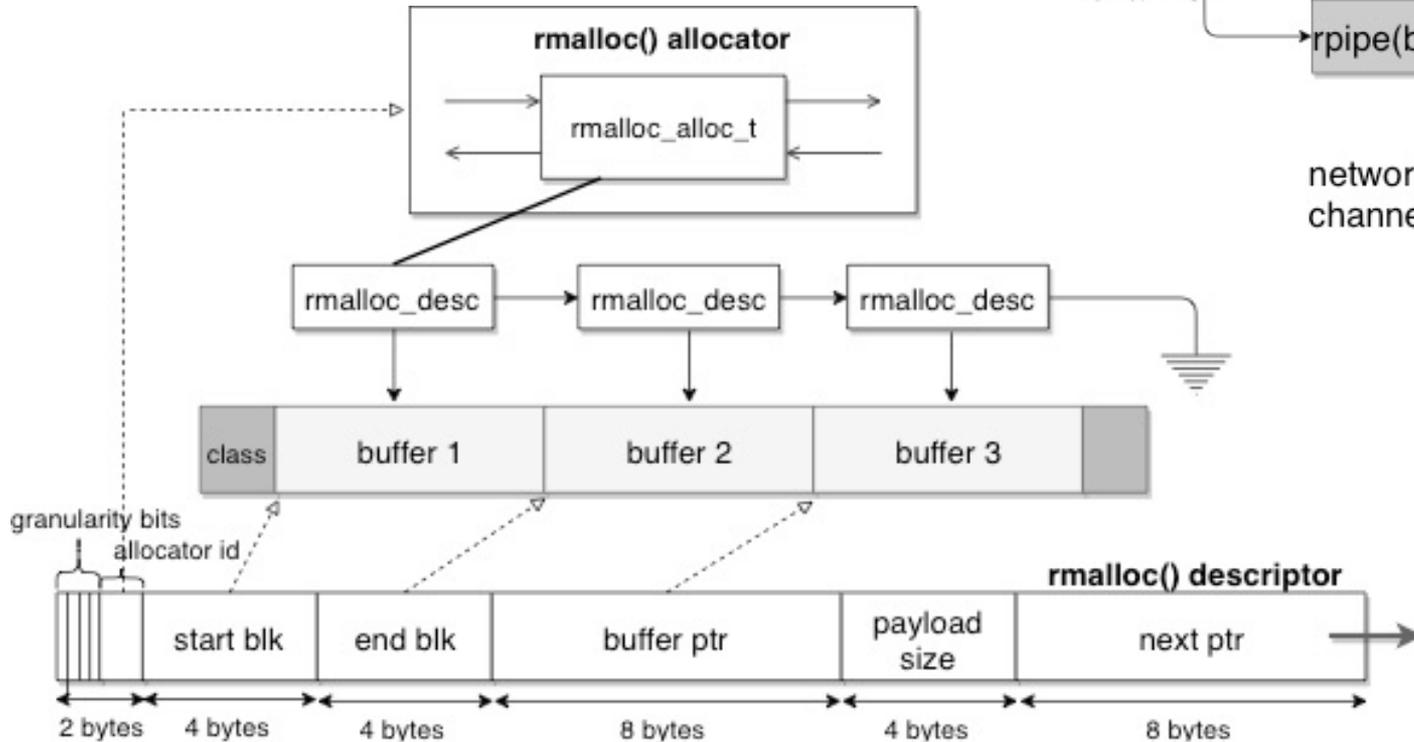
best-fit heuristic – find **smallest** possible RDMA heap segment

“rmalloc” Allocation



worst-fit heuristic – find **largest** possible RDMA heap segment

“rmalloc” Implementation



rmalloc_descriptor → manages local and remote virtual memory

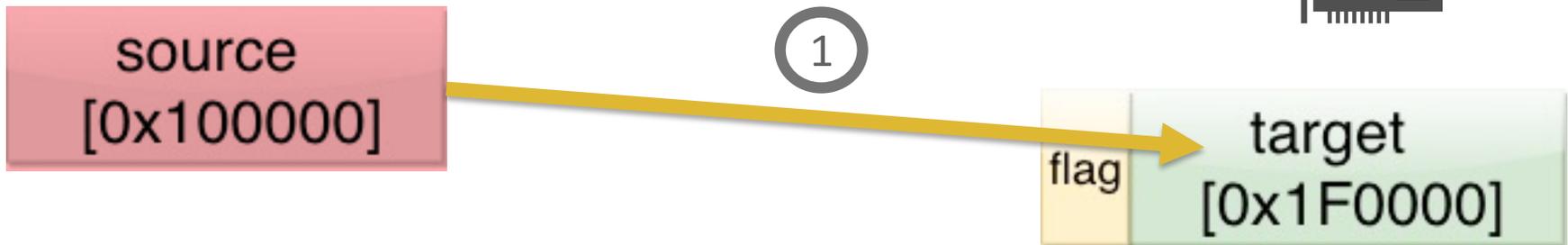
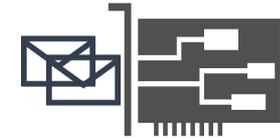
rfree()/rmalloc() synchronization

- When to synchronize ? Buffer “in-use/re-use”
 - Two options, use both for different allocation modes
 - At allocation time → latency (i.e. rmalloc())
 - At de-allocation time → throughput (i.e. rfree())
- *Deferred* synchronization by **rfree()** → next-fit
 - Coalesce tags from a sorted free list
 - rmalloc updates state by RDMA into coalesced tag list in the remote
- *Immediate* synchronization by **rmalloc()** → best-fit
OR worst-fit
 - Using a special *bootstrap* rpipe to synchronize at each allocated memory

“rpipe” – rwrite()

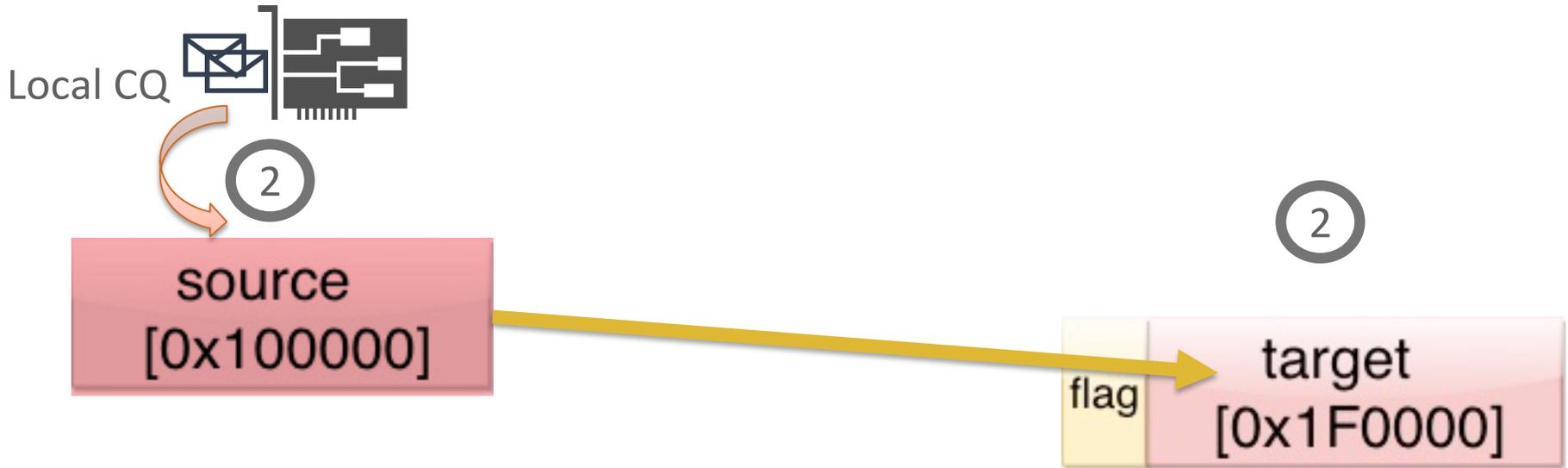


- Completion Queue (CQ)
(Light weight events by NIC/HCA)



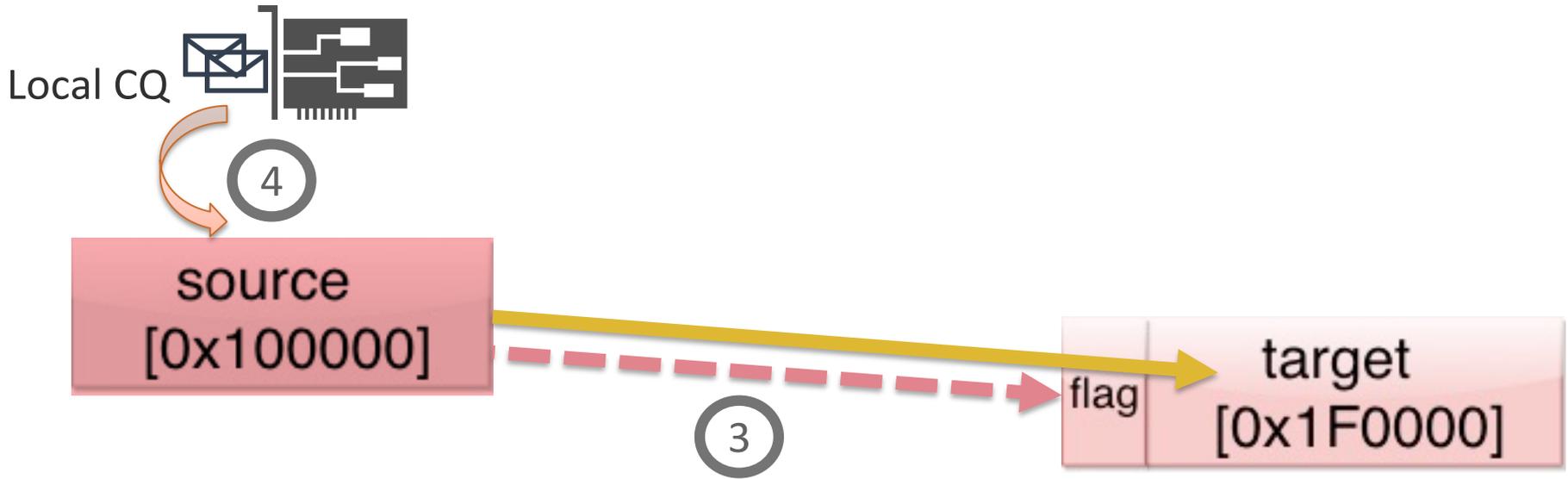
1. Initiate RDMA Write.
– Source buffer → “in-use”

“rpipe” – rwrite()



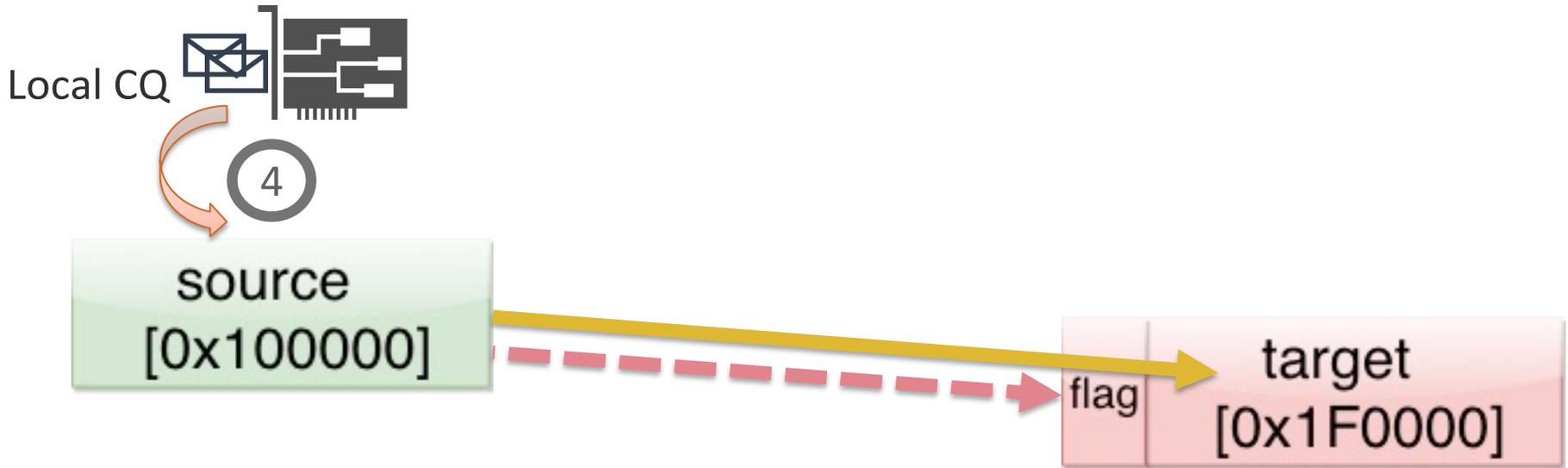
2. **Probe** Local CQ for completion.
Zero-copy source data to target.

“rpipe” – rwrite()



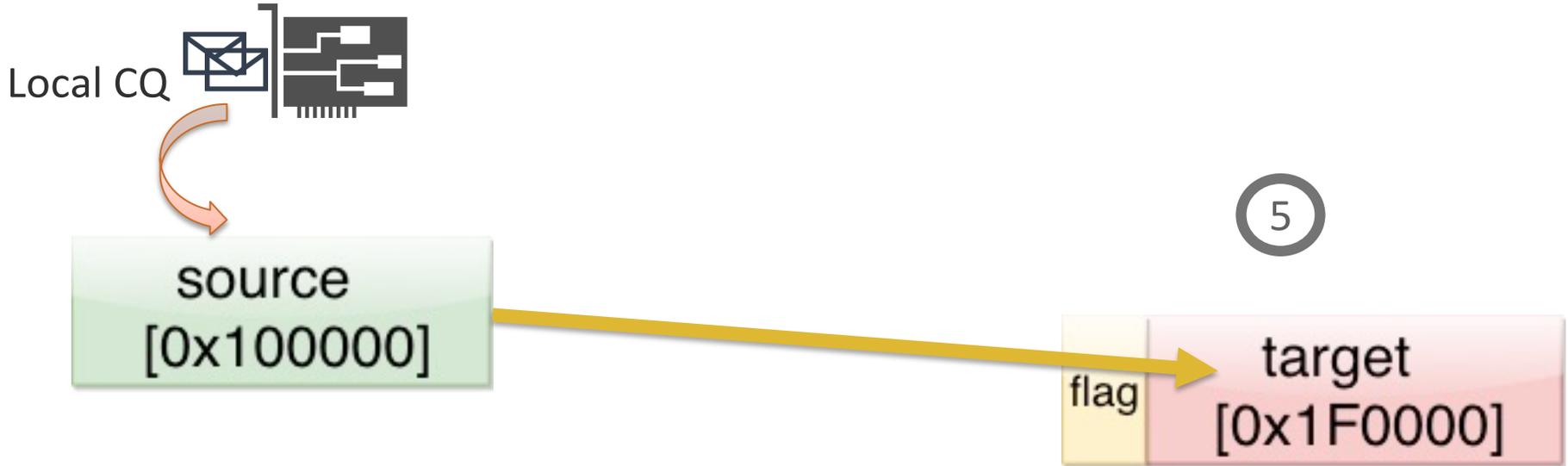
3. Write to **flag** just **after** data.

“rpipe” – rwrite()



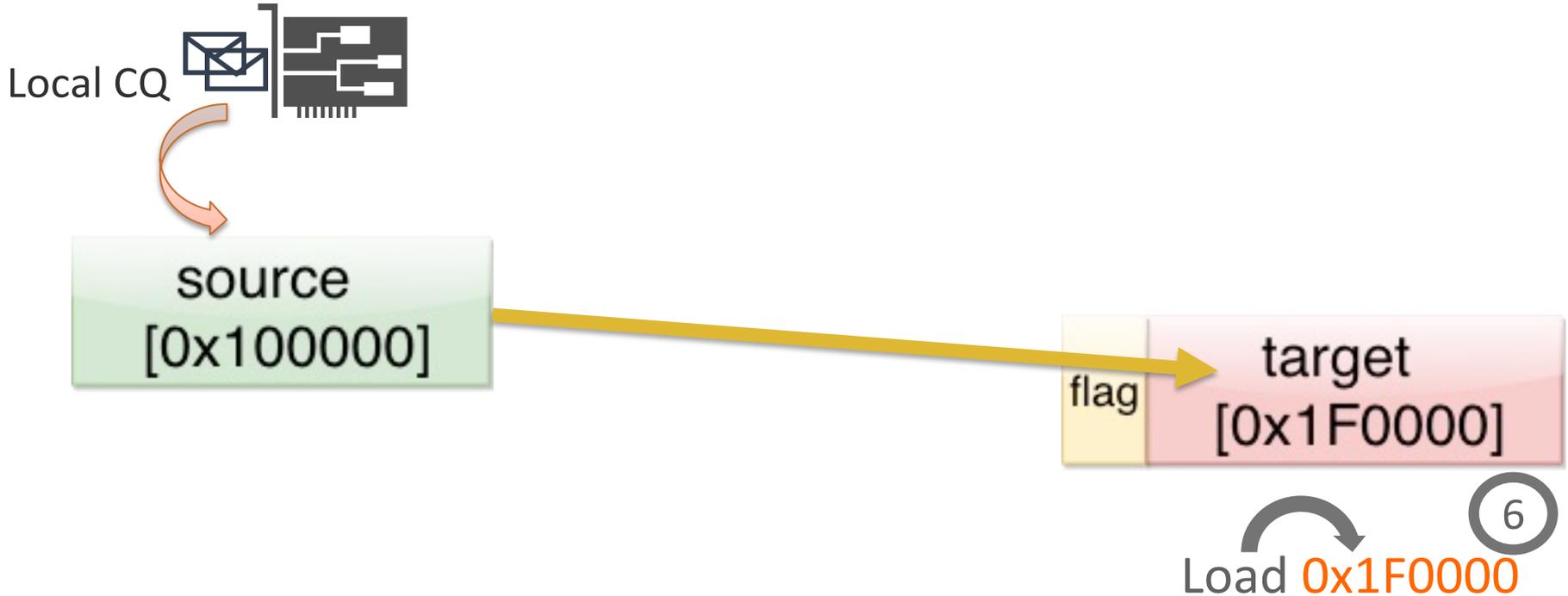
4. **Probe** Local CQ success.
Source buffer → “re-use”

“rpipe” – rwrite()



5. **Probe** flag success.
target buffer is **ready** to load/ops.

“rpipe” – rwrite()



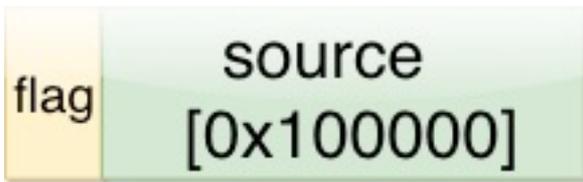
6. remote host consumes data.
Source yet to know buffer → `rfree()`

“rpipe” – rread()



1

Store 0x1F0000, val

A diagram showing a circular arrow with the number 1 inside a circle, indicating a store operation.

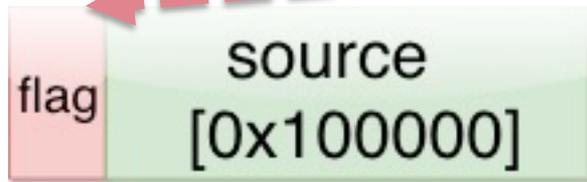
1. Store data into **target**.
 - Target buffer → “in-use”.

“rpipe” – rread()



2

Store **0x1F0000**, val
rfree()



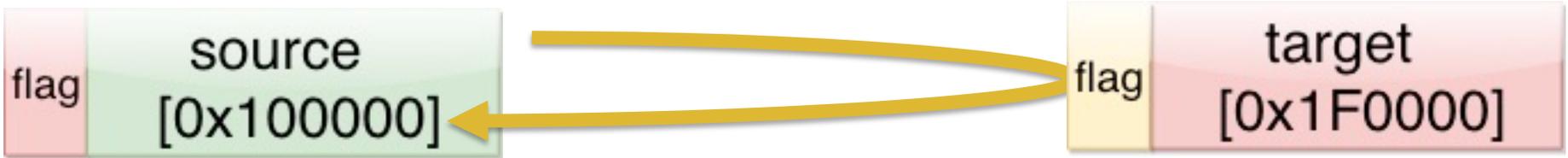
2. Write to source **flag**.

Data is now **ready** for **rread()!!**

“rpipe” – rread()



3

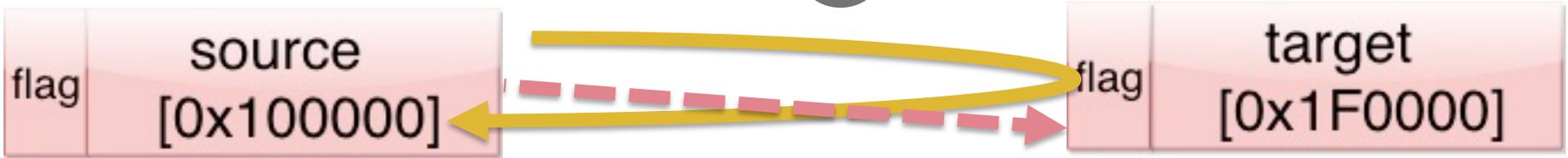


3. RDMA **Zero-Copy** to source.

“rpipe” – rread()

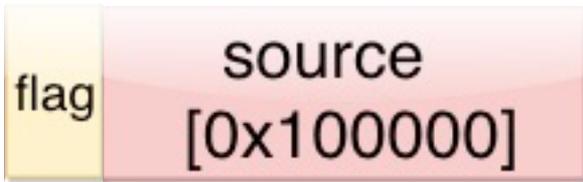


4



4. Write to **flag** just **after** data.

“rpipe” – rread()



5. **Probe** Local CQ for completion.

Implementing rpipe(), rwrite() and rread()

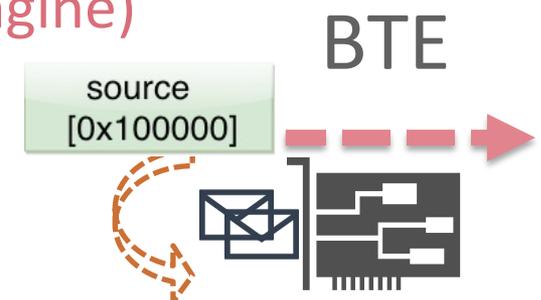
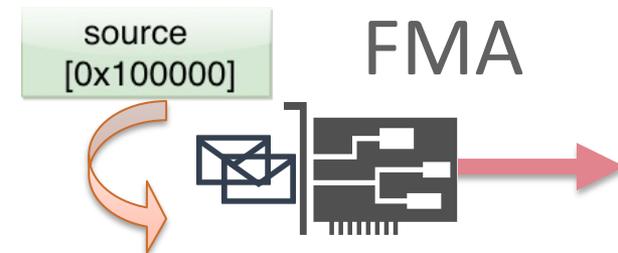
- A rpipe is created between two endpoints.
 - A uGNI based Control Message (FMA Cmsg) network to lazy initialize rpipe i.e. GNI_CqCreate, GNI_EpCreate, GNI_EpBind

- Implements rwrite(), rread() in uGNI

- Small/medium messages – **FMA (Fast Memory Access)**
- Large messages – **BTE (Byte Transfer Engine)**

- MPI portability Layer

- rpipe with MPI-3.0 windows + passive RMA



Overview

- Motivation
- Design/System Implementation
- Evaluation
- Future Work

rpipe programming

```
int main(){
#define PIPE_WIDTH 8

rpipe_t rp;
rinit(&rank, NULL);
// create a Half Duplex RMA pipe
rpipe(rp, peer, iswriter, PIPE_WIDTH, HD_PIPE);

raddr_t addr;
int *ptr;

if (iswriter) {
    addr = rmalloc(rp, sizeof(int));
    ptr = rmem(rp, addr);
    *ptr = SEND_VAL;
    rwrite(rp, addr);
} else {
    rread(rp, addr, sizeof(int));
    ptr = rmem(rp, addr);

    rfree(addr);
}
```

Remote allocate

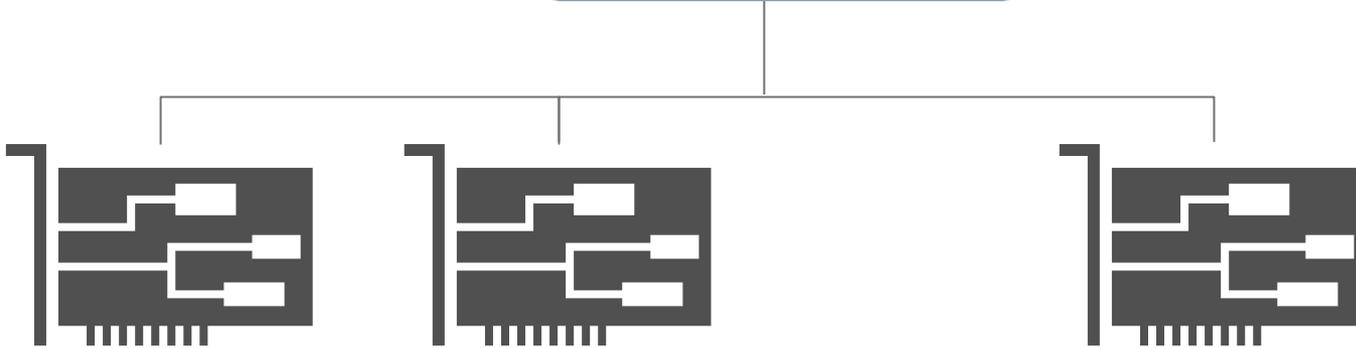
Rpipe ops

Free rem memory

*Release immediately
after use !!*

Experimentation Setup

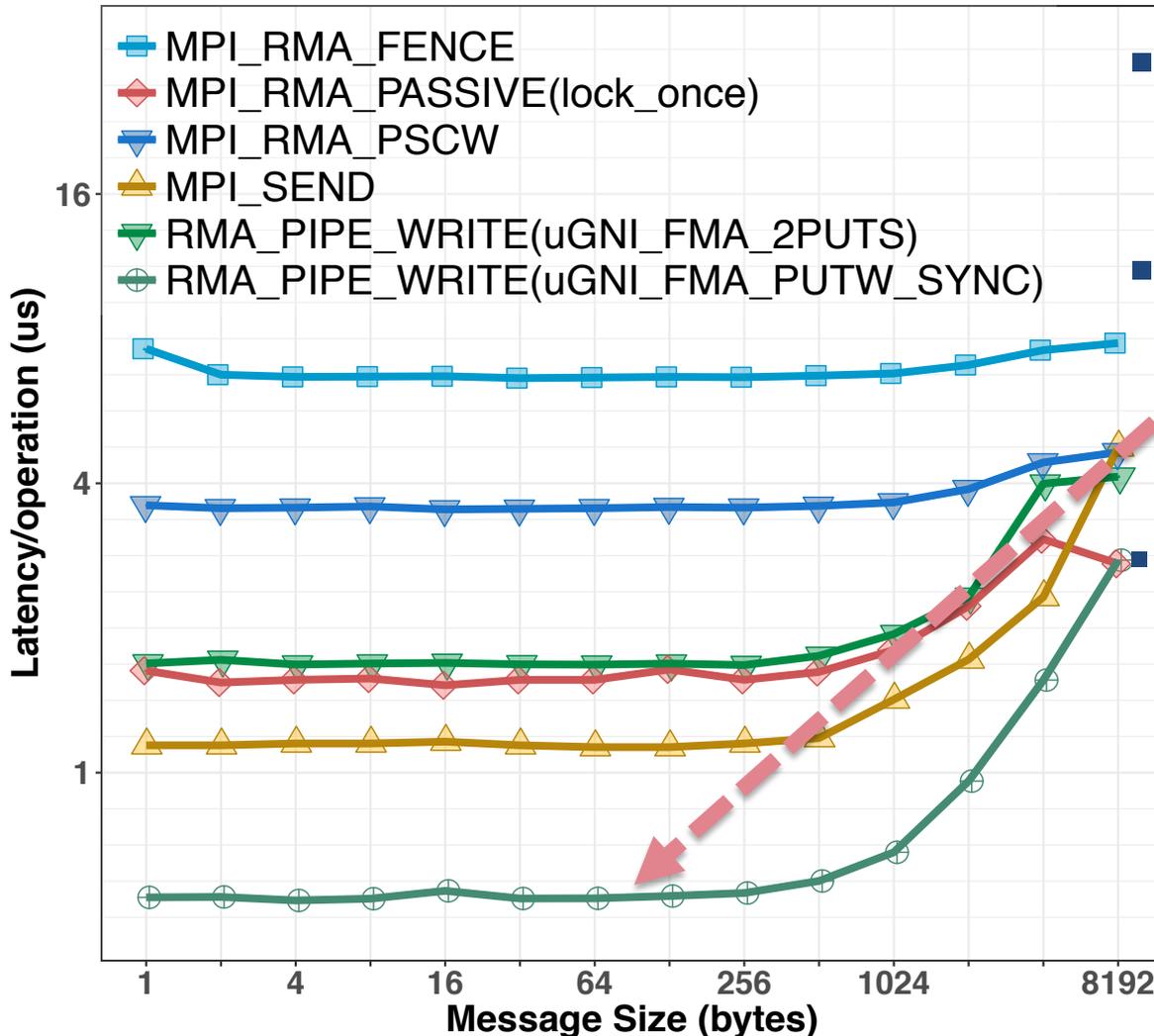
Cray XC30[Aries]/
Dragon Fly



BigredII+ 550 nodes/ Rpeak 280 Tflops
— 10GB/s Uni-directional 15GB/s Bi-directional BW

Perf baseline → MPI/OSU Benchmark

Small/Medium Message Latency Comparison



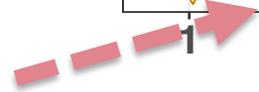
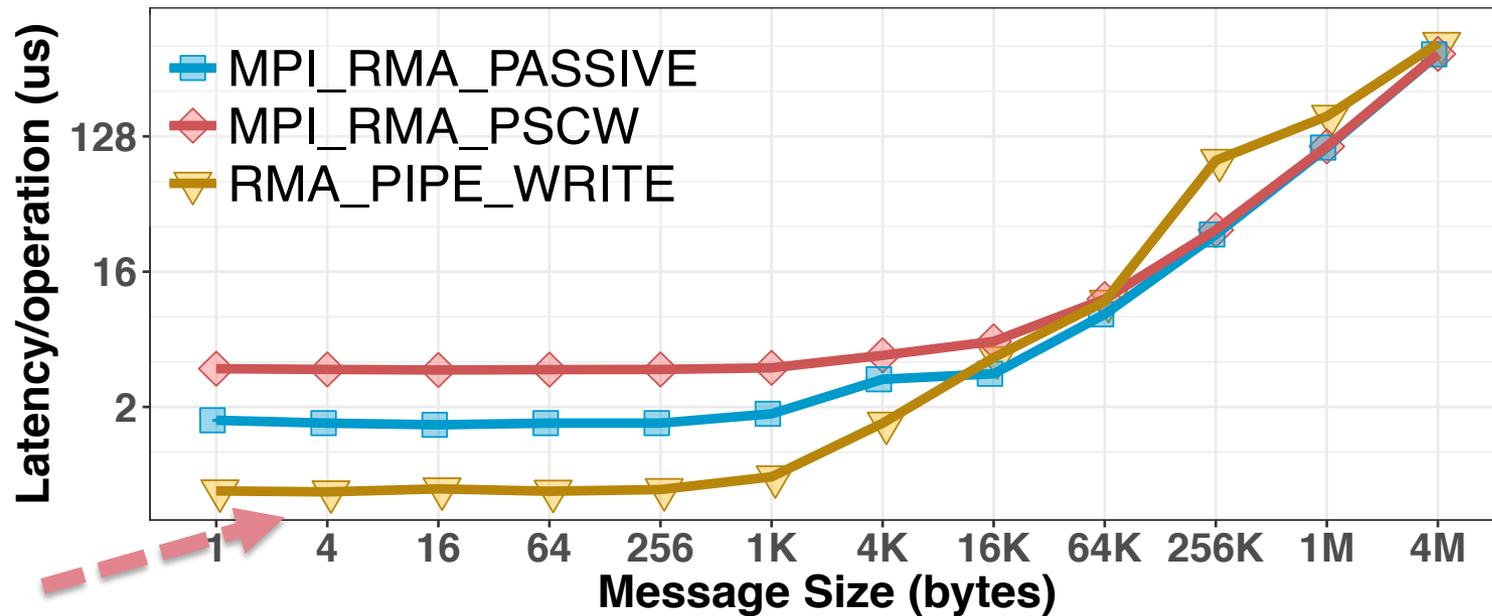
Default Alloc = Next-Fit

FMA_PUT_W_SYNC

Upto 6X speedup MPI
RMA

$rpipe\ PUT_W_sync(S) <$
 $rpipe\ 2PUT\ (s)$

Large Message Latency Comparison – rwrite()



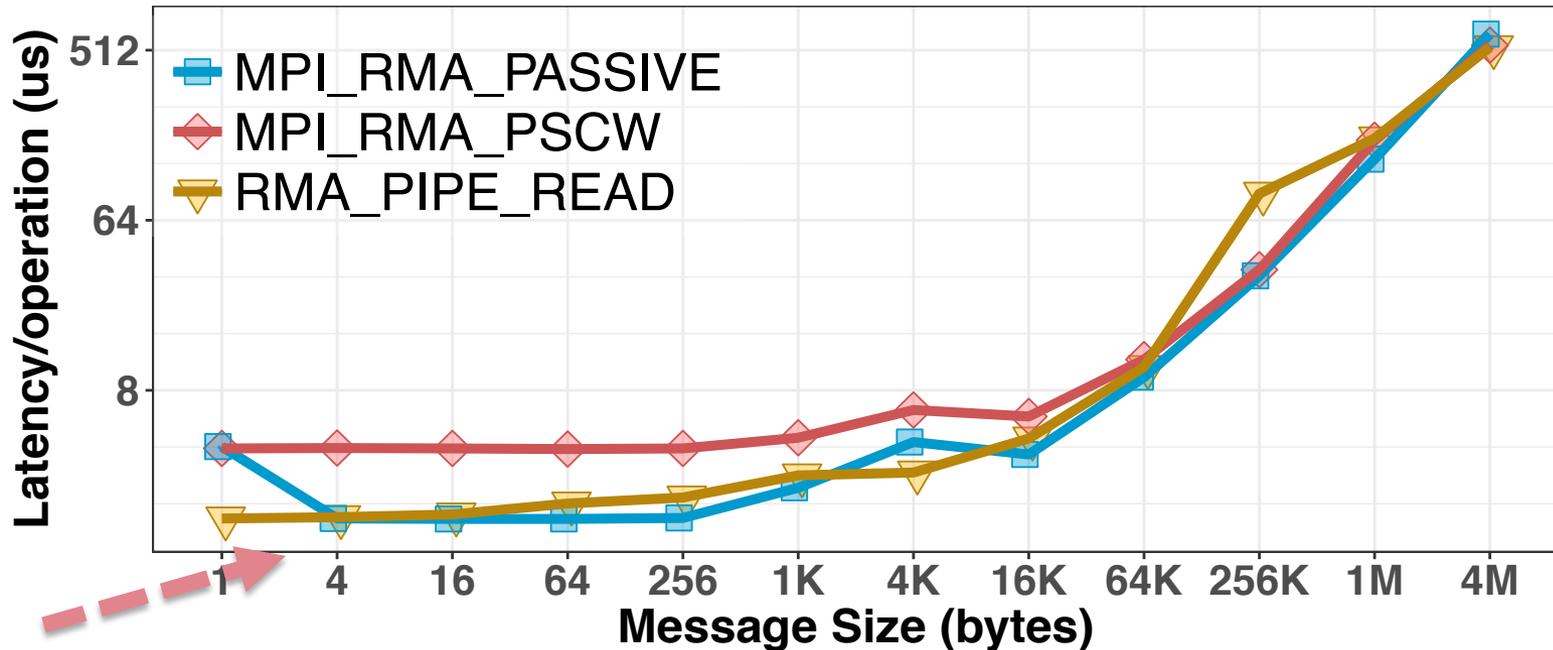
small/medium

0.65us

■ *rpipe* uGNI(s) \approx *rpipe* MPI(s) when $s > 4K$

- $S \geq 4K \rightarrow$ FMA to BTE switch

Large Message Latency Comparison – rread()



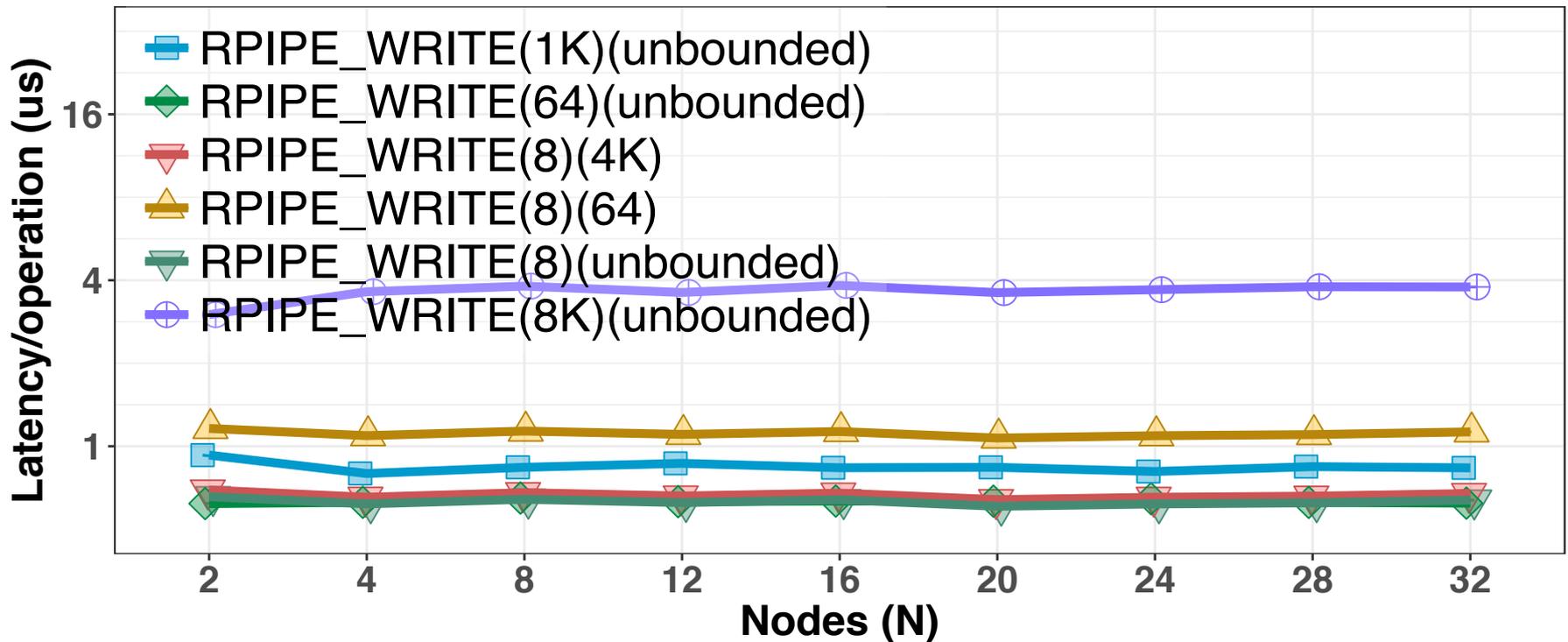
small/medium

2.14us

▪ $rpipe\ uGNI(s) \approx rpipe\ MPI(s)$ when $s > 1K$

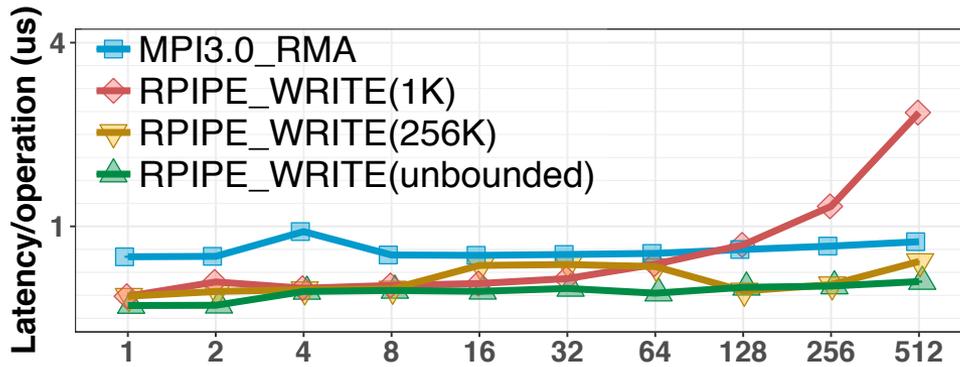
- $S < 4b \rightarrow FMA_FETCH$ Atomic (AMO)
- $S < 1K \rightarrow FMA_FETCH + PSYNC$
- $S \geq 1K \rightarrow FMA$ to BTE switch (BTE_FETCH + FMA_PSYNC)

Rpipe Scales ...



- “*unbounded*” → allocator has full rpipe available for all Zero-copy operations
- *Scaling upto 32 nodes – randomized rwrite()*
 - 0.65 – 3.8us avg latency

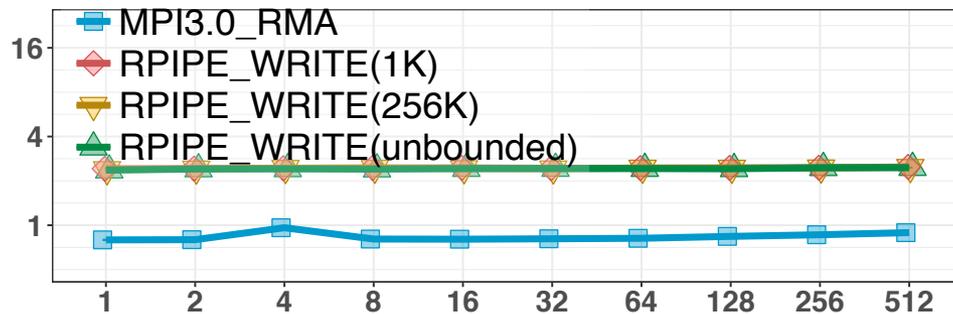
Allocation Algorithms



Next-fit

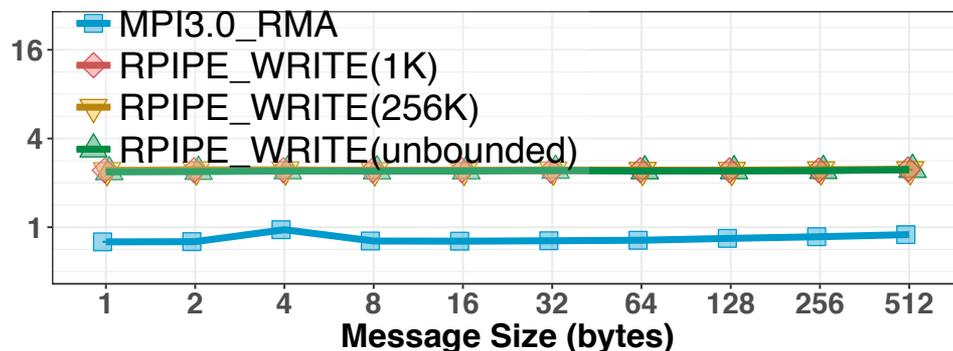
Zero-copy write vs Heuristics

- Next-fit allocator has better performance



Best-fit

- 1X – 3.5X slowdown for Best/Worst-fit



Worst-fit

L = Latency

$L[\text{Next-fit}] < L[\text{MPI}] < L[\text{Worst-fit}]$

Overview

- Motivation
- Design/System Implementation
- Evaluation
- Future Work

Future Work

- Platform Support/Automated synchronization

- High performance RMA Kernels
 - Active messages/ Neighbor/collective communication

- Aggregated rpipes
 - Leverage Zero copy/Eliminate hidden buffers
 - i.e. Collectives
 - Possible throughput, memory utilization gains

- Irregular RMA and memory disaggregation



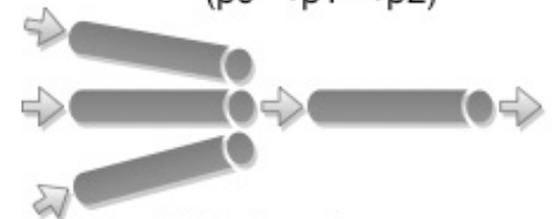
a) Half Duplex pipe



b) Full Duplex pipe



c) Redirection pipe
(p0 → p1 → p2)



d) Redux pipe

Questions?

Thank You!