FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds

Daehyeok Kim

Tianlong Yu[1], Hongqiang Liu[3], Yibo Zhu[4], Jitu Padhye[2], Shachar Raindel[2]
Chuanxiong Guo[4], Vyas Sekar[1], Srinivasan Seshan[1]

Carnegie Mellon University[1], Microsoft[2], Alibaba group[3], Bytedance[4]

Two Trends in Cloud Applications

Containerization

docker LXC

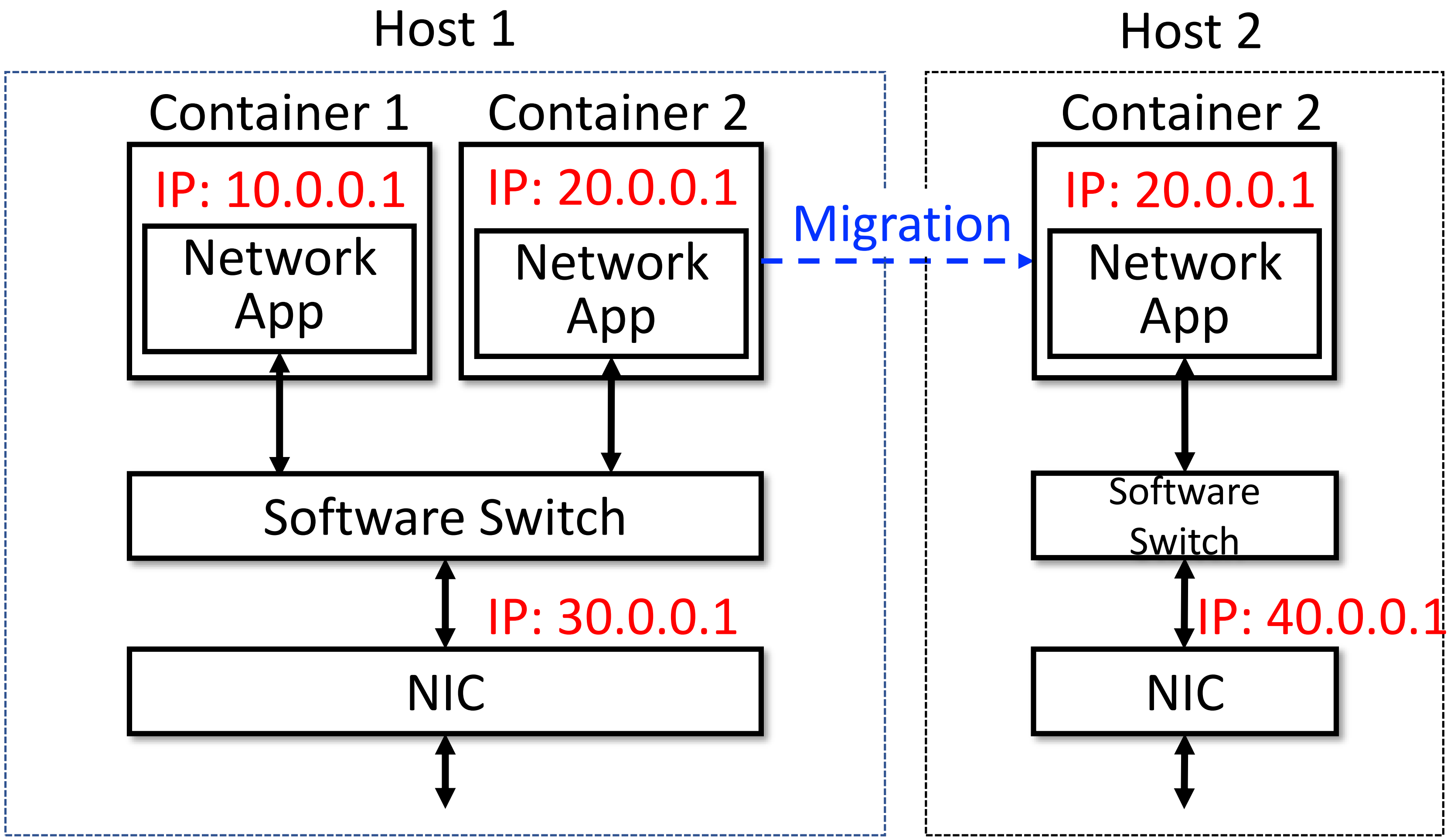
- Lightweight isolation
- Portability

RDMA networking

RoCE™ INFINIBAND℠ TRADE ASSOCIATION

- Higher networking performance

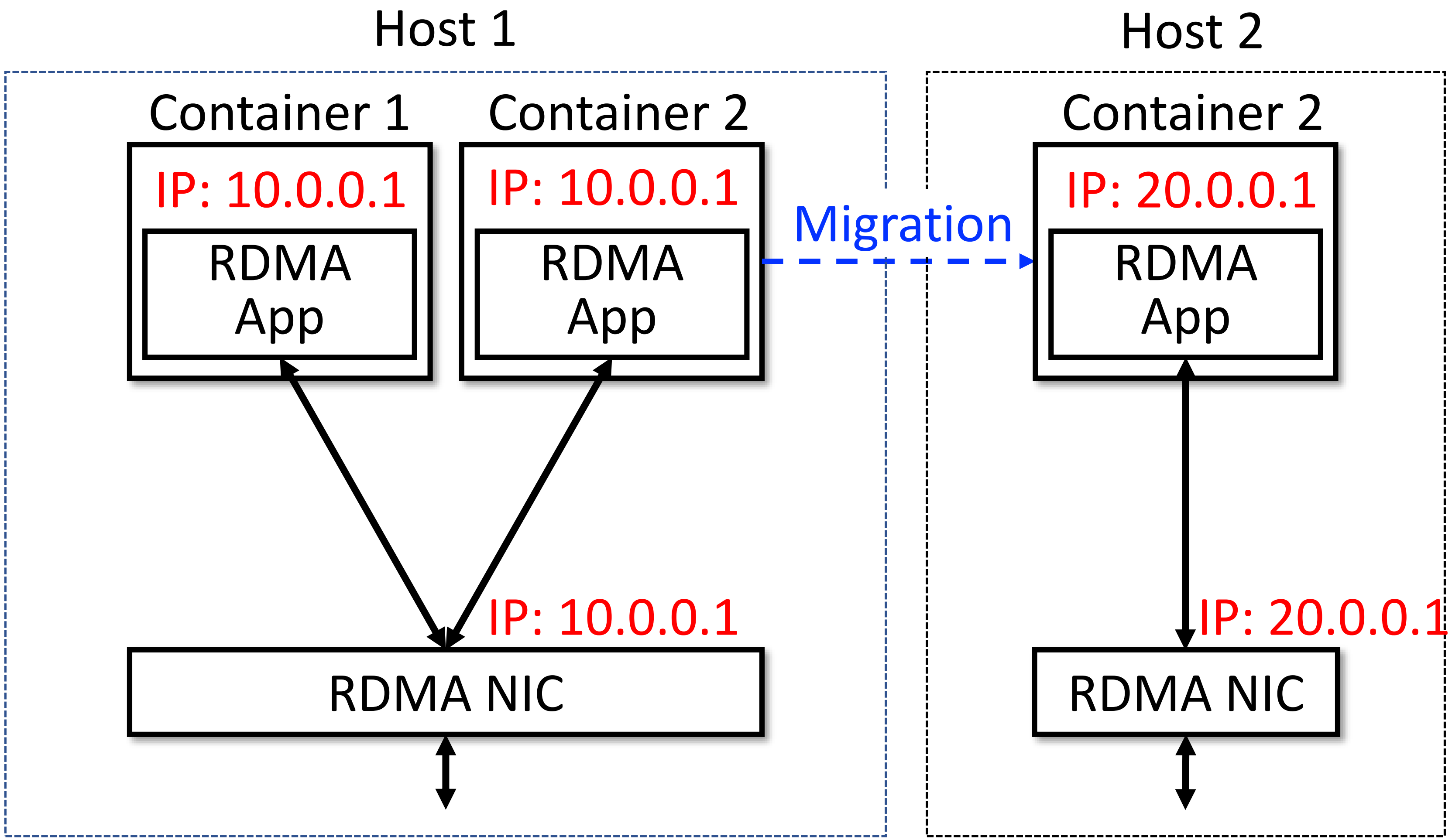
Benefits of Containerization



Namespace Isolation ✓

Portability ✓

Containerization and RDMA are in Conflict!

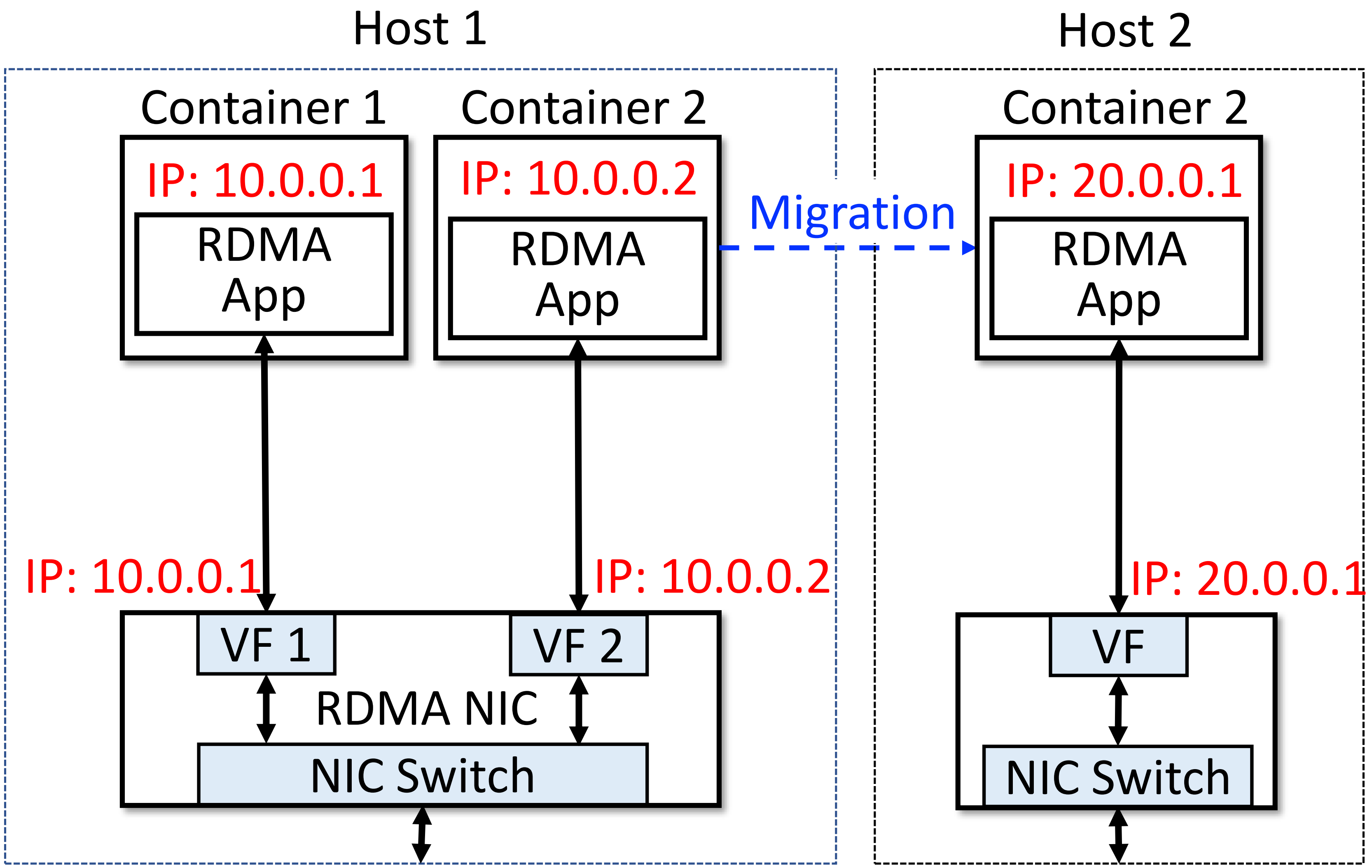


Namespace Isolation ❌

Portability ❌

Existing H/W based Virtualization Isn't Working

Using Single Root I/O Virtualization (SR-IOV)



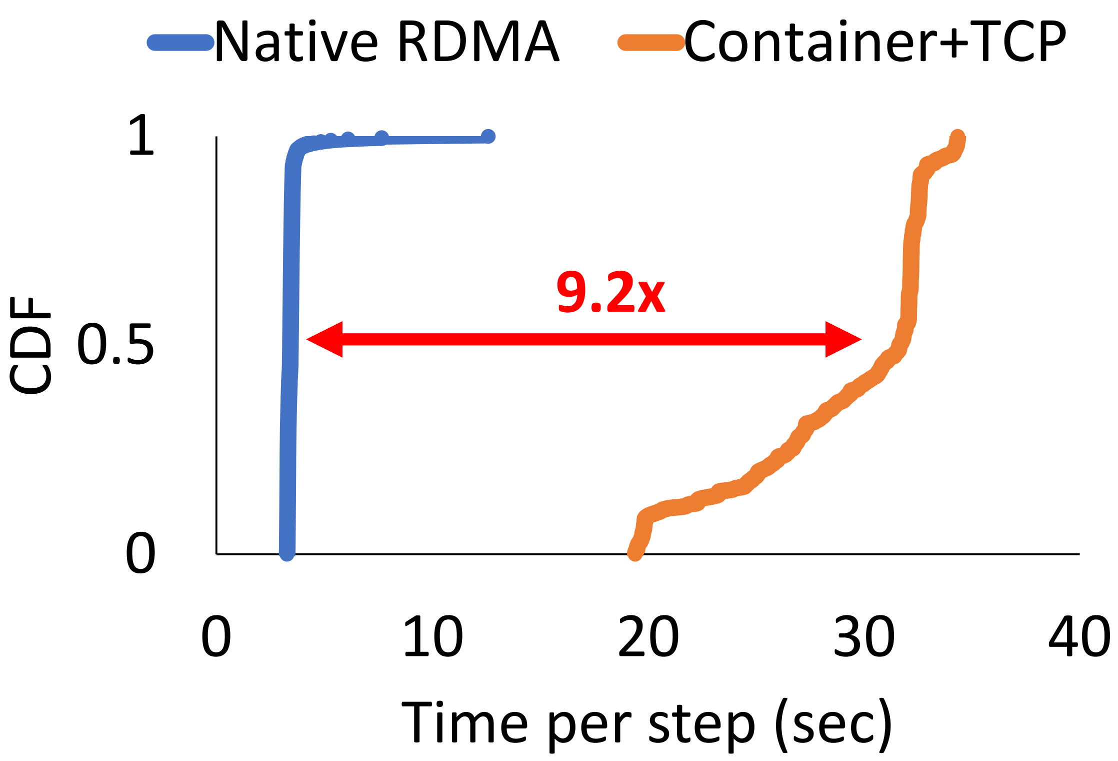
Namespace Isolation ✓

Portability ✗

VF Virtual Function

Sub-optimal Performance of Containerized Apps

RDMA networking can improve the training speed of NN model by ~ 10x !



Speech recognition RNN training

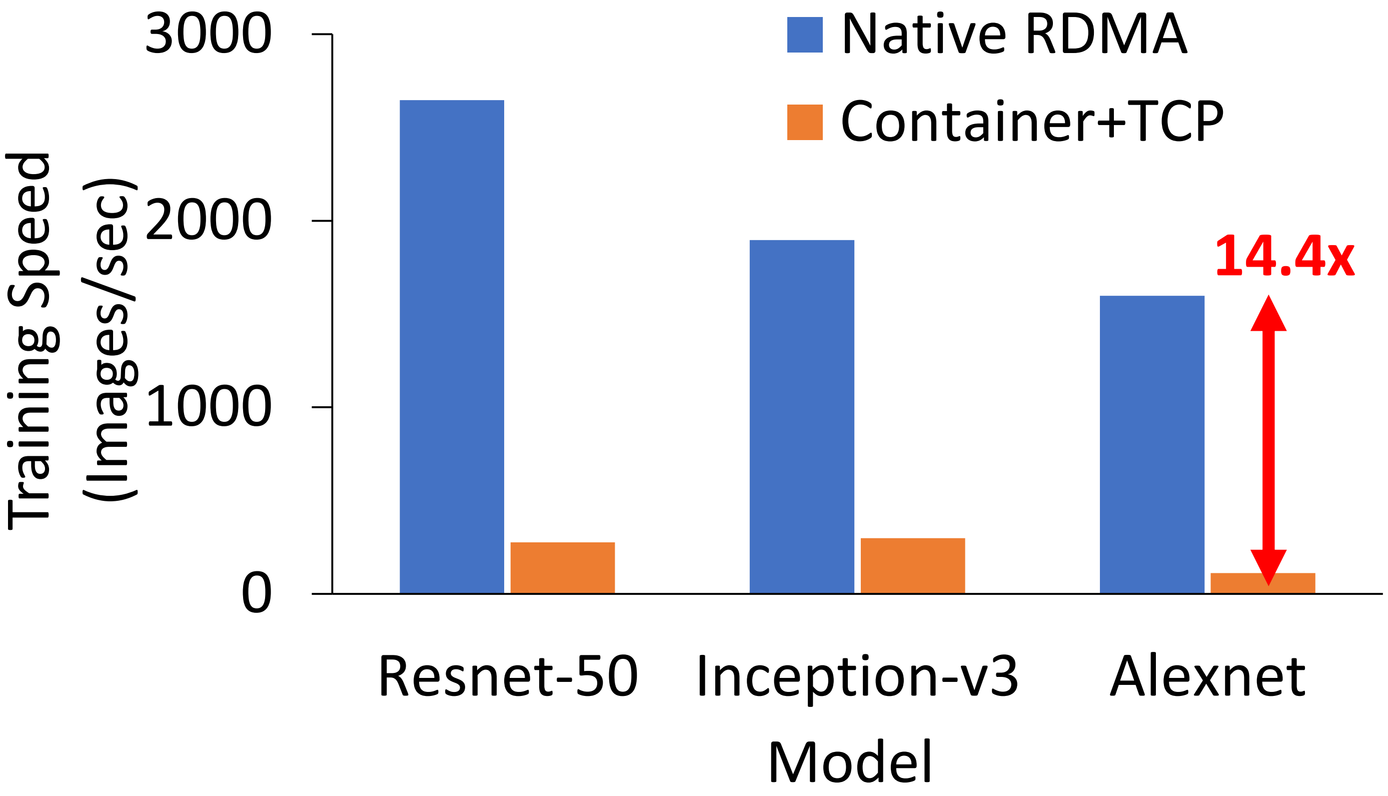


Image classification CNN training

Our Work: FreeFlow

- Enable high speed RDMA networking capabilities for containerized applications

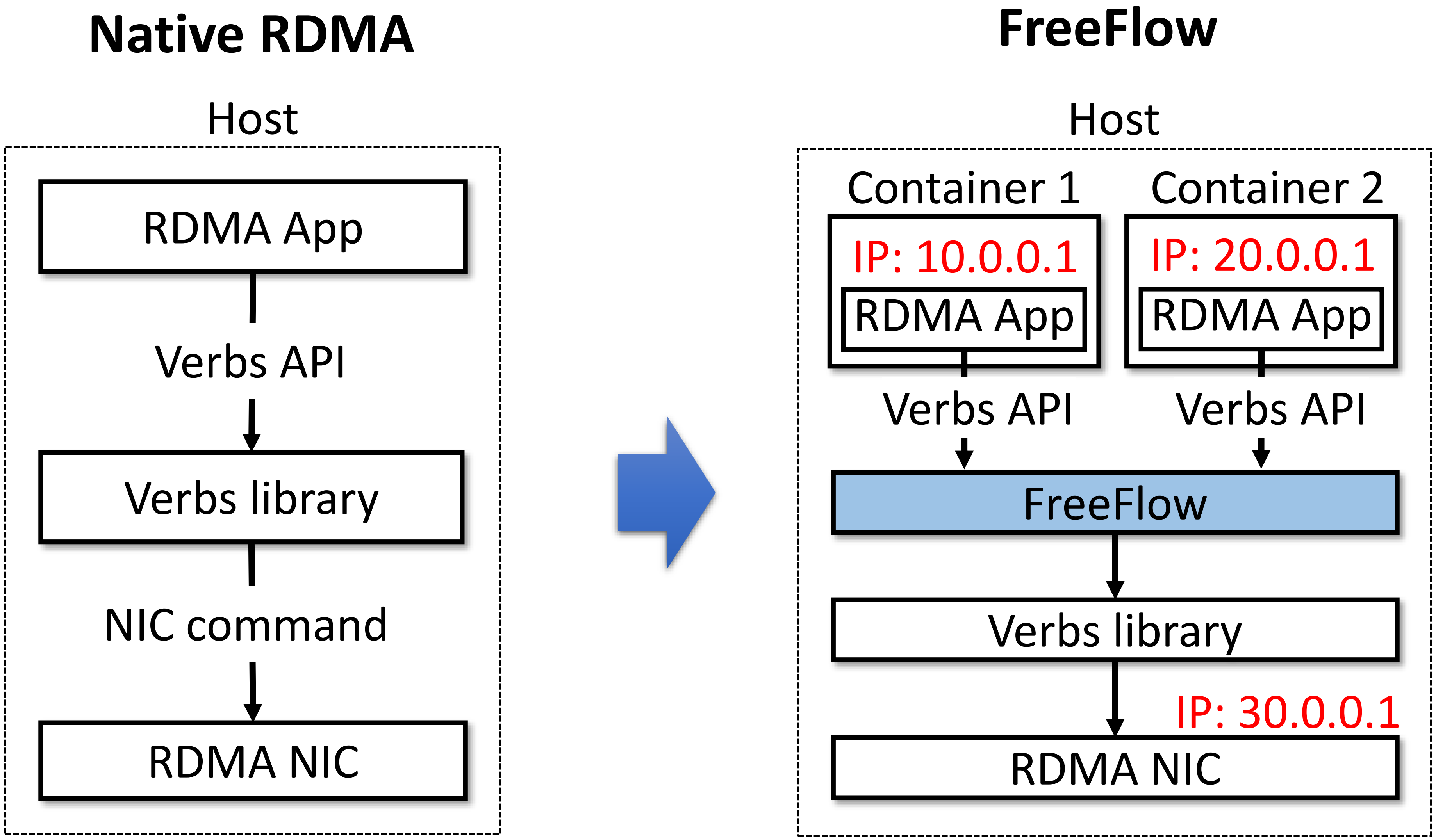
- Compatible with existing RDMA applications

- Close to native RDMA performance
    - Evaluation with real-world data-intensive applications

Outline

- Motivation
- FreeFlow Design
- Implementation and Evaluation

FreeFlow Design Overview



Native RDMA

Host

RDMA App

Verbs API

Verbs library

NIC command

RDMA NIC

FreeFlow

Host

Container 1

IP: 10.0.0.1

RDMA App

Container 2

IP: 20.0.0.1

RDMA App

Verbs API

Verbs API

FreeFlow

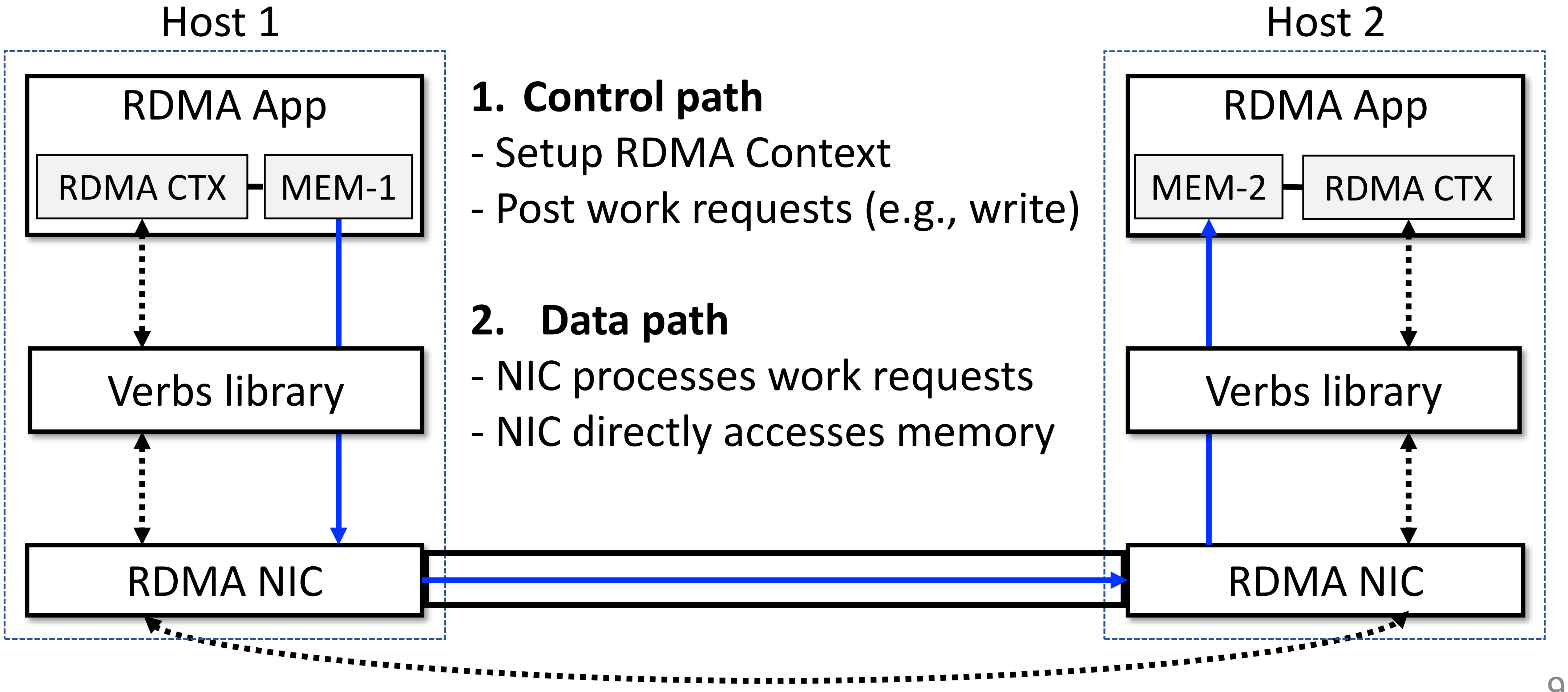
Verbs library

IP: 30.0.0.1

RDMA NIC

Background on RDMA

“Host 1 wants to write contents in MEM-1 to MEM-2 on Host 2”



1. **Control path**

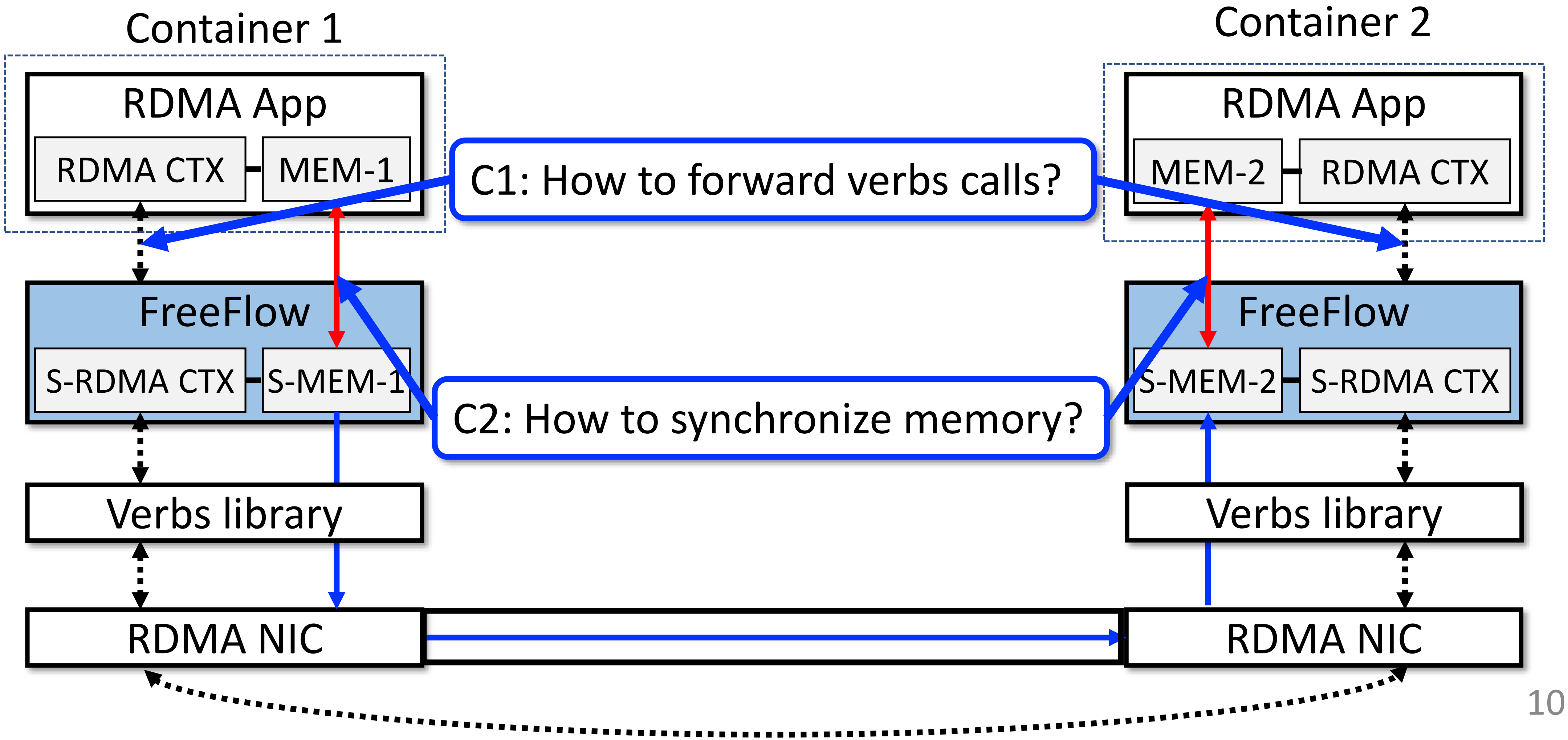
- Setup RDMA Context
- Post work requests (e.g., write)

2. **Data path**

- NIC processes work requests
- NIC directly accesses memory

FreeFlow in the Scene

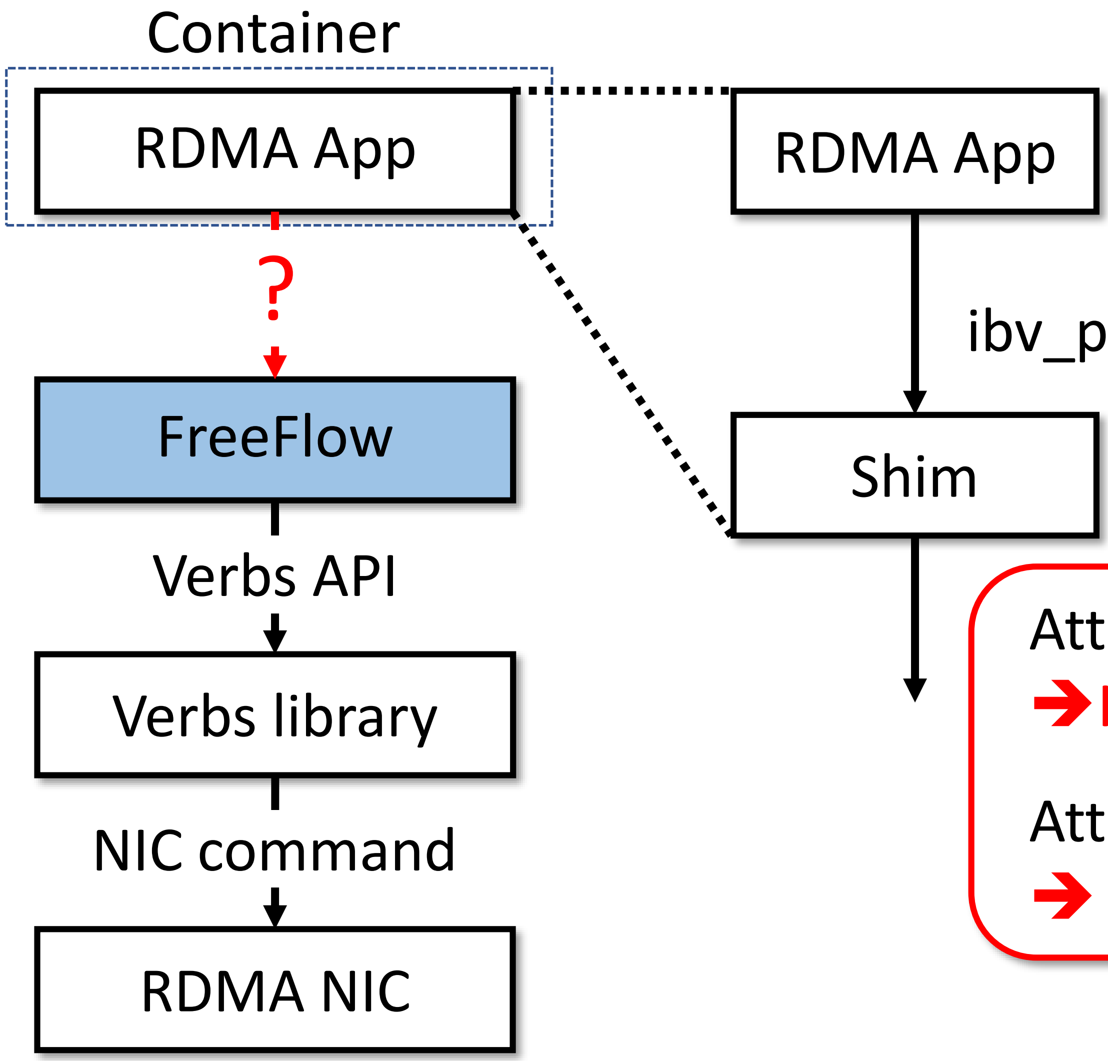
“Container 1 wants to write contents in MEM-1 to MEM-2 on Container 2”



C1: How to forward verbs calls?

C2: How to synchronize memory?

Challenge 1: Verbs forwarding in Control Path



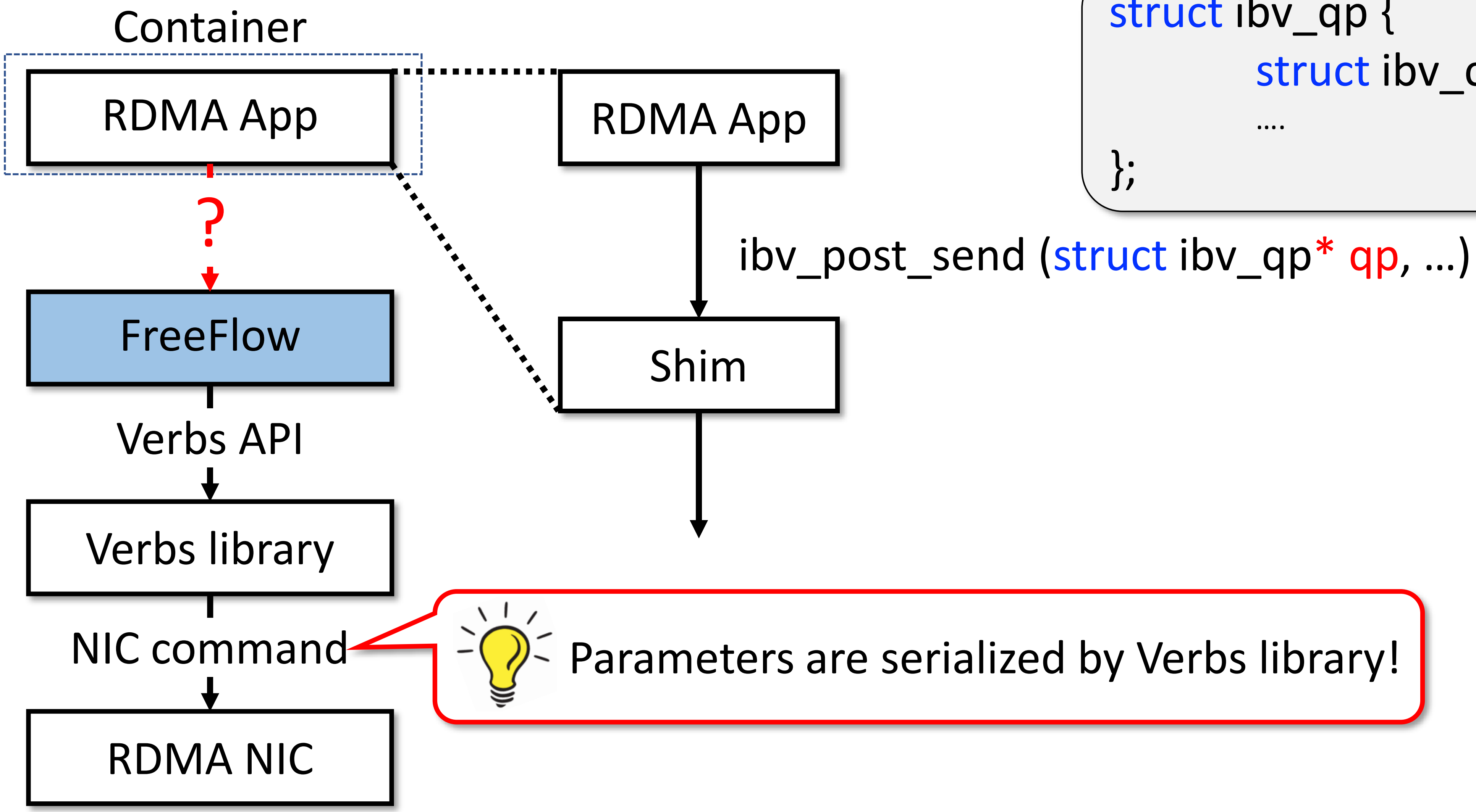
```
struct ibv_qp {
          struct ibv_context *context;
          ....
};
```

ibv_post_send (struct ibv_qp* qp, ...)

Attempt 1: Forward "as it is"
➔Incorrect

Attempt 2: "Serialize" and forward
➔ Inefficient

Internal Structure of Verbs Library

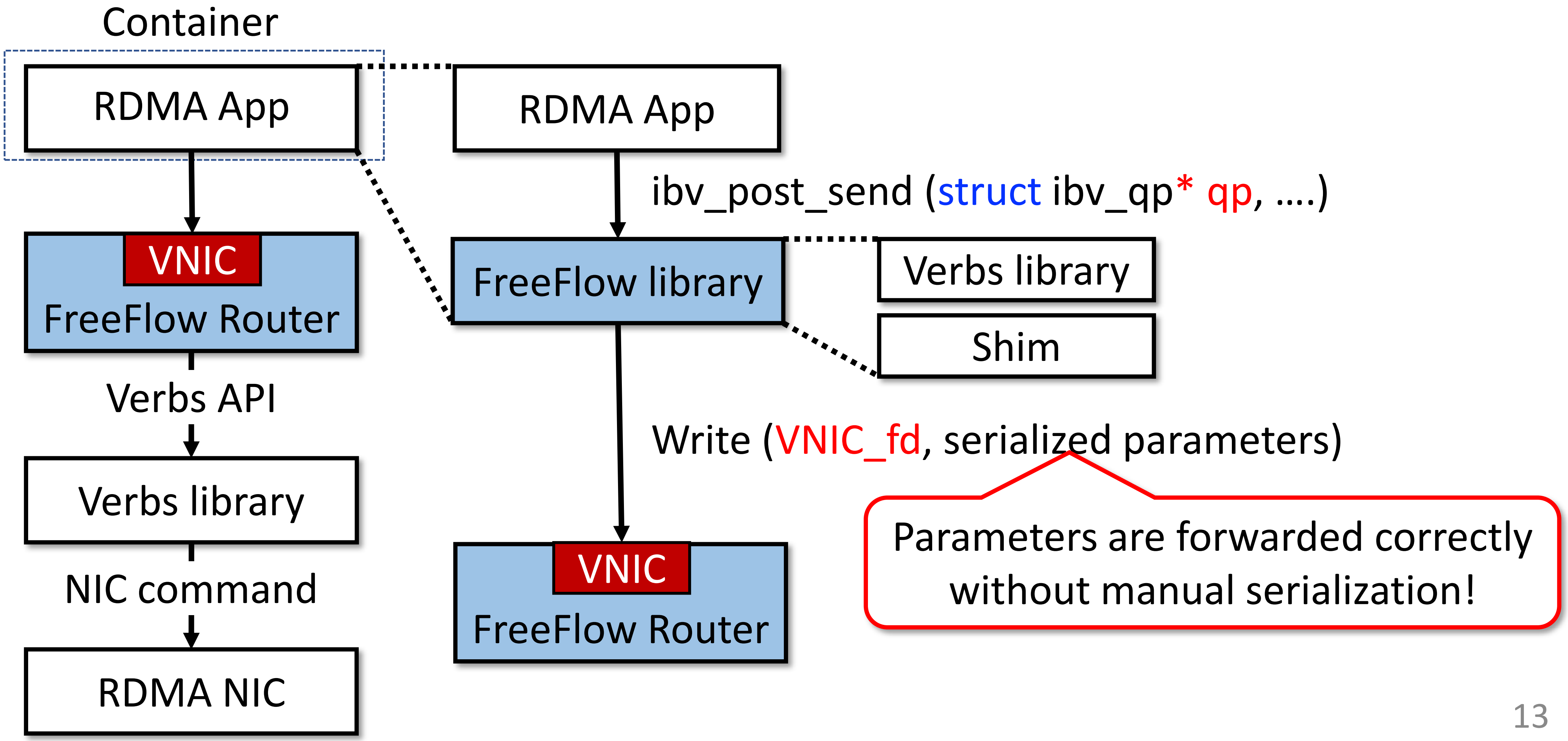


```
struct ibv_qp {
        struct ibv_context *context;
        ….
};
```

Parameters are serialized by Verbs library!

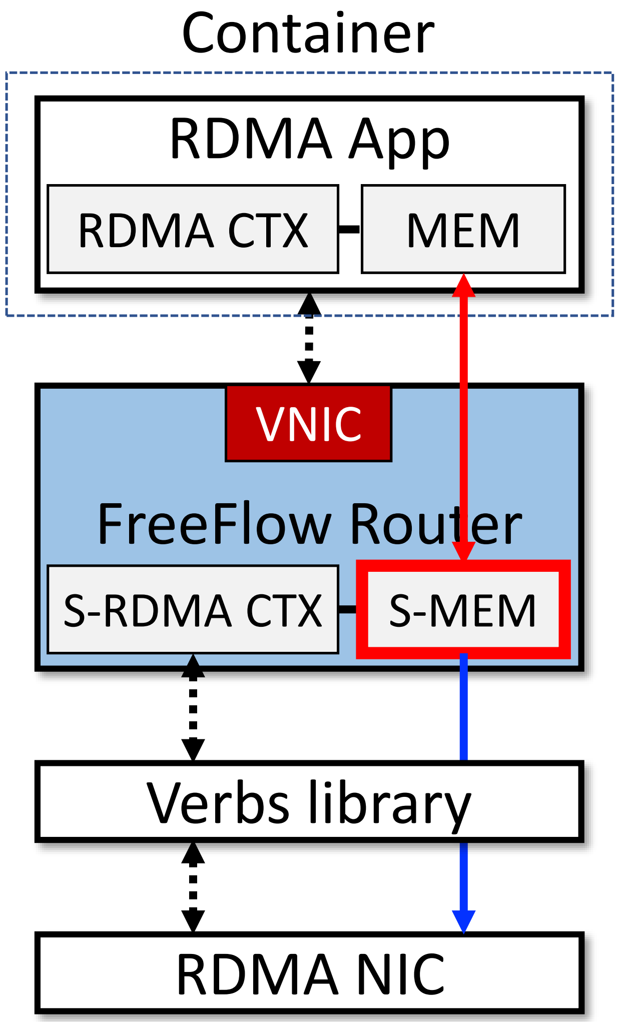
FreeFlow Control Path Channel

Idea: Leveraging the serialized output of verbs library



Parameters are forwarded correctly without manual serialization!

Challenge 2: Synchronizing Memory for Data Path



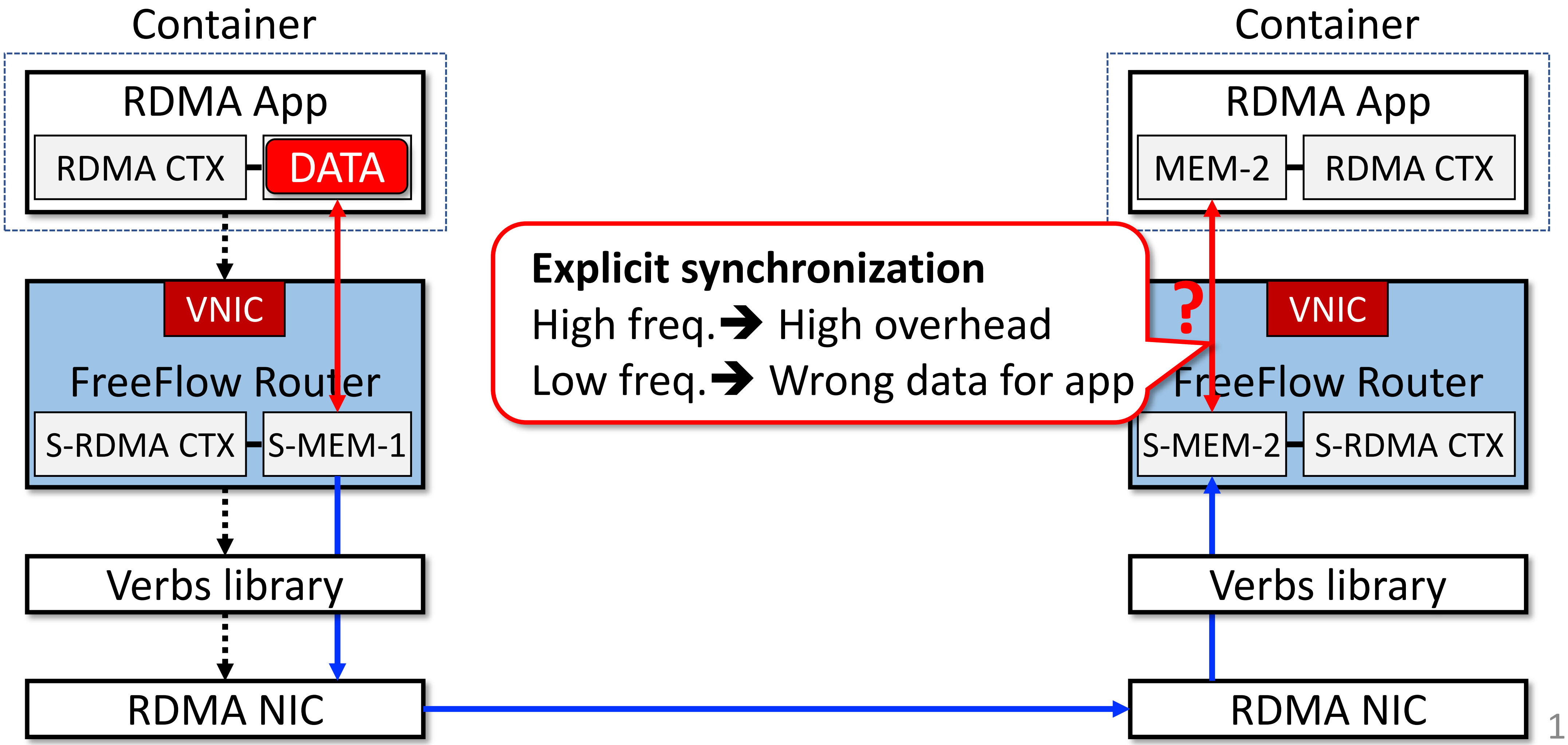
- Shadow memory in FreeFlow router
  - A copy of application’s memory region
  - Directly accessed by NICs

- S-MEM and MEM must be synchronized.

- How to synchronize S-MEM and MEM?

Strawman Approach for Synchronization

“Container 1 wants to write contents in MEM-1 to MEM-2 on Container 2”

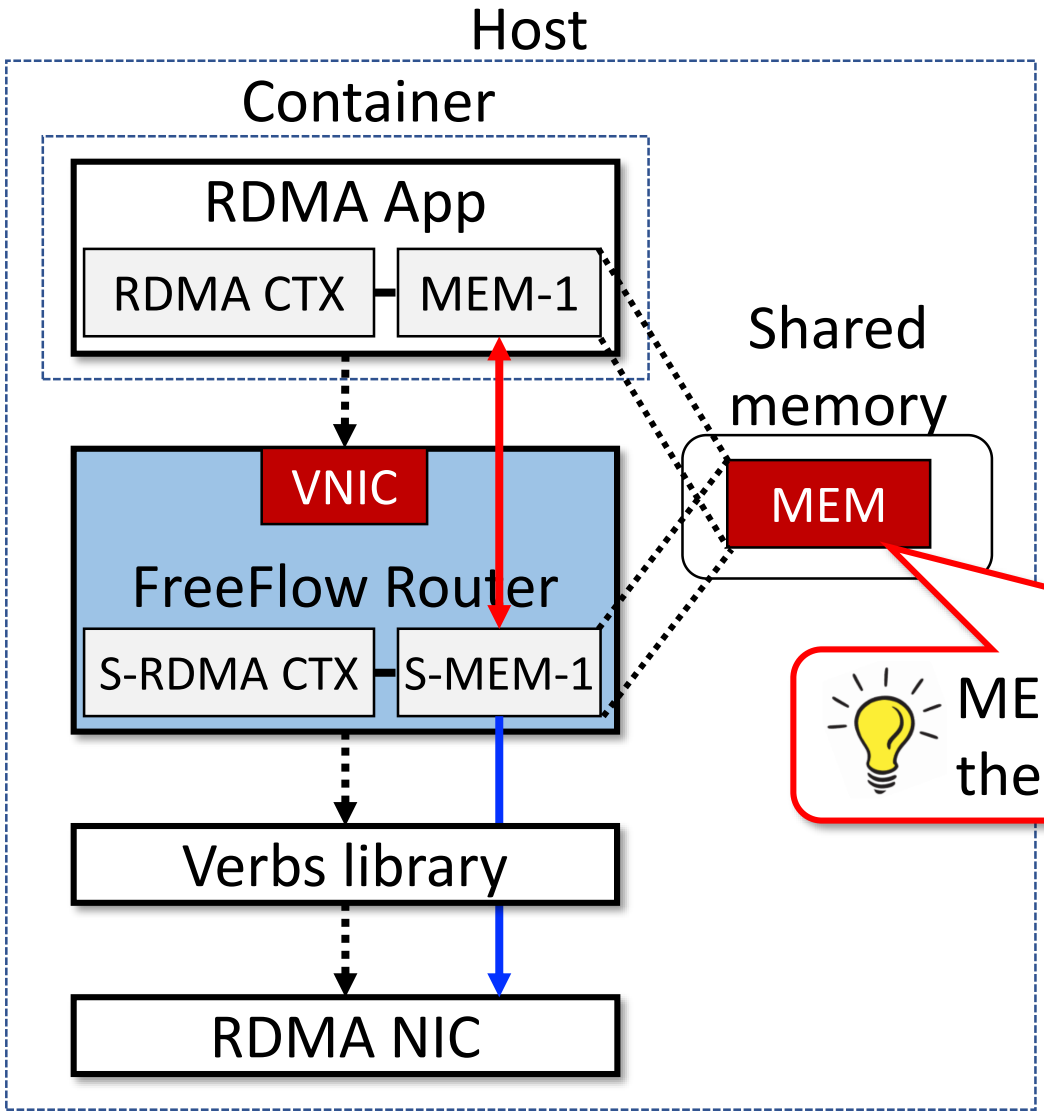


Explicit synchronization

High freq.➔ High overhead

Low freq.➔ Wrong data for app

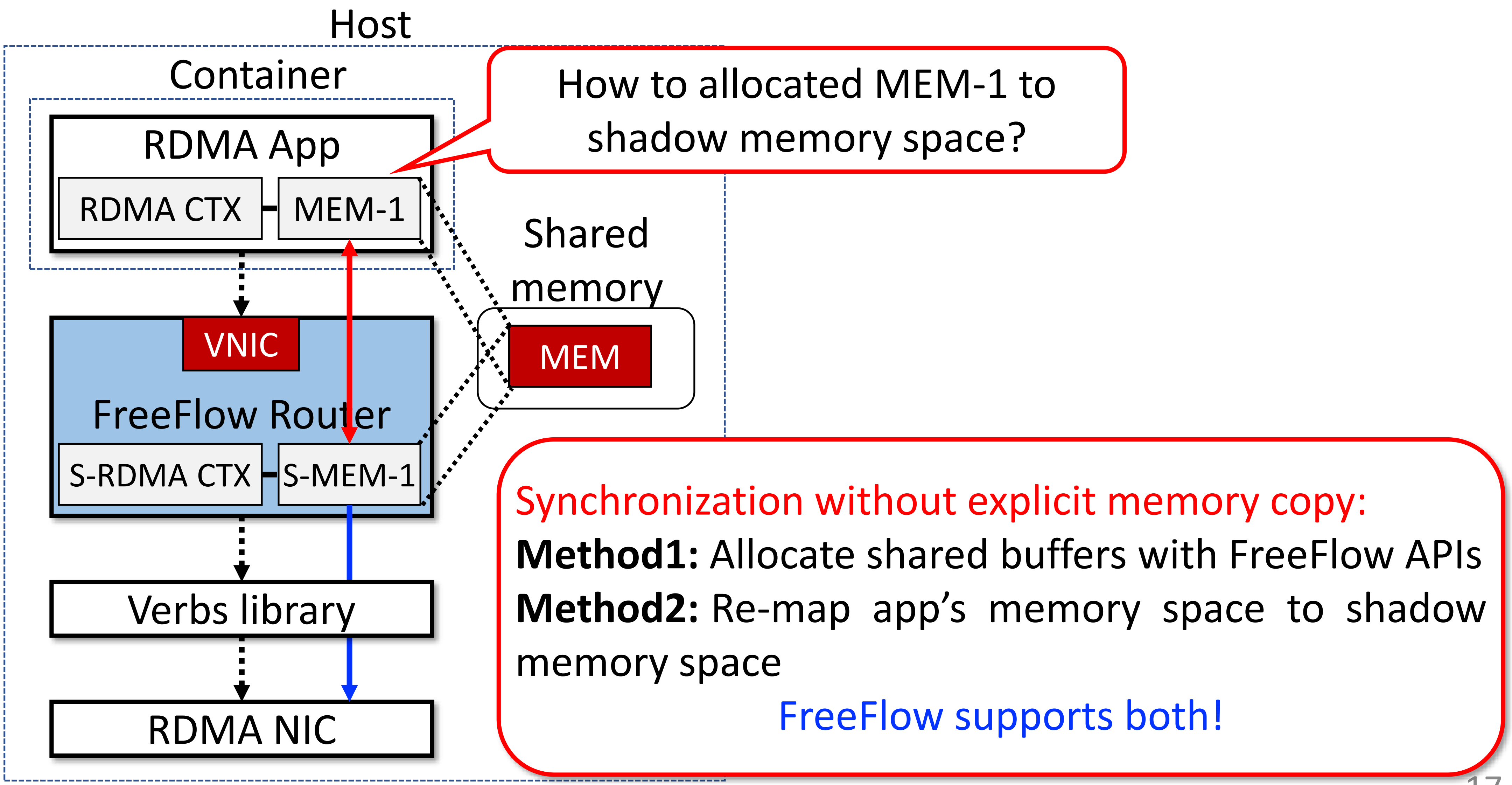
Containers can Share Memory Regions



- FreeFlow router is running in a container

MEM and S-MEM can be located on the same physical memory region

Zero-copy Synchronization in Data Path



How to allocated MEM-1 to shadow memory space?

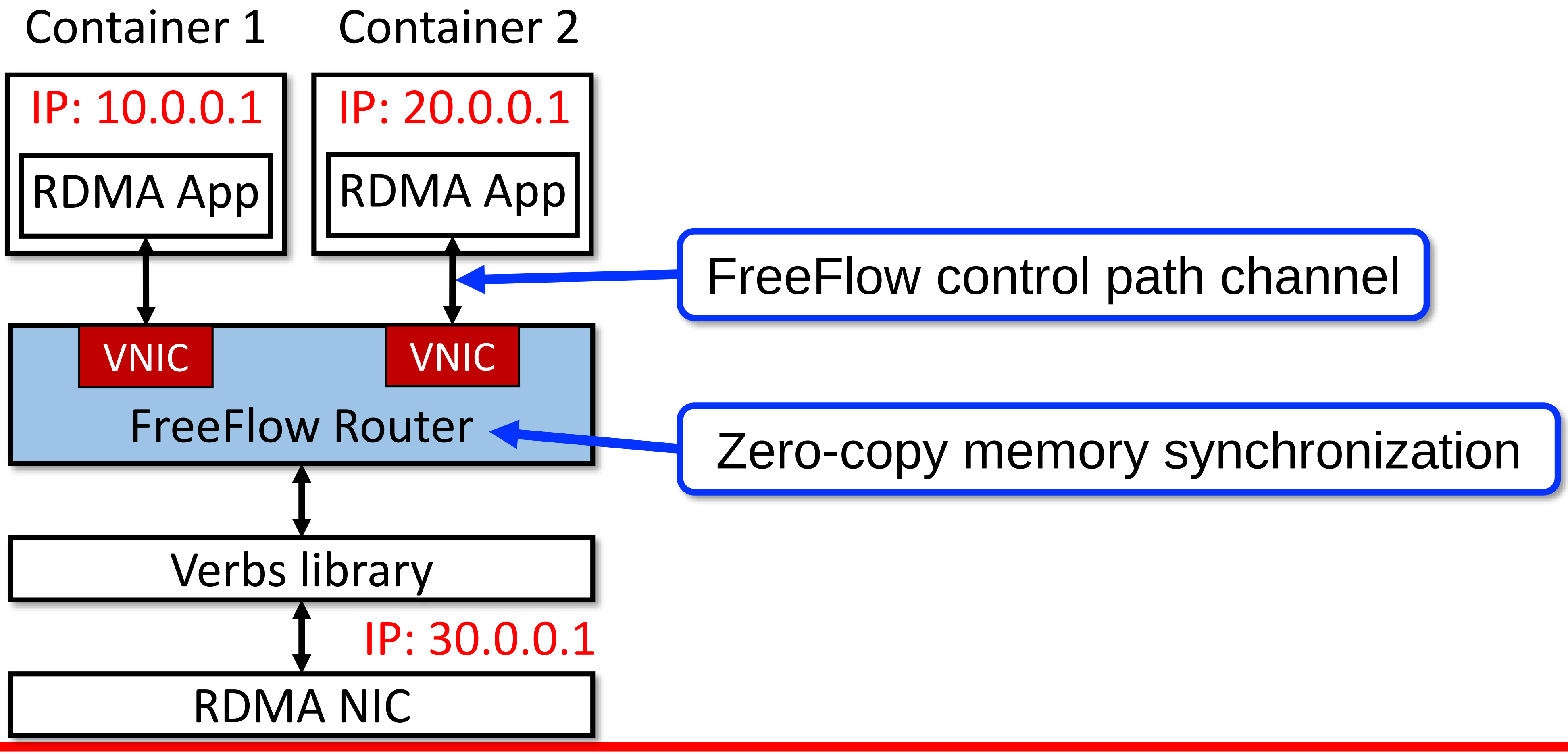
Synchronization without explicit memory copy:

Method1: Allocate shared buffers with FreeFlow APIs

Method2: Re-map app’s memory space to shadow memory space

FreeFlow supports both!

FreeFlow Design Summary



FreeFlow control path channel

Zero-copy memory synchronization

FreeFlow provides near native RDMA performance for containers!

Outline

- Motivation

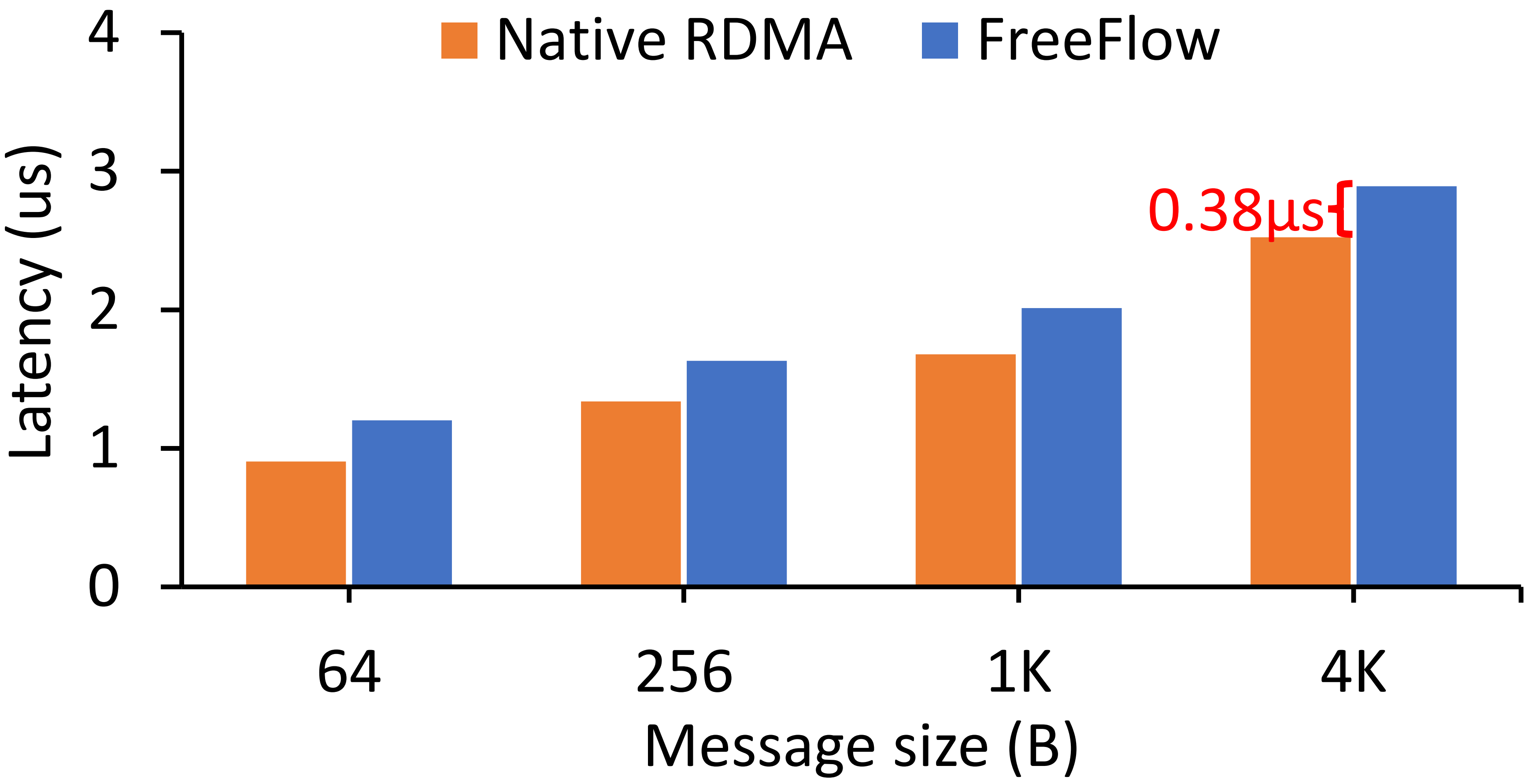
- FreeFlow Design

- Implementation and Evaluation

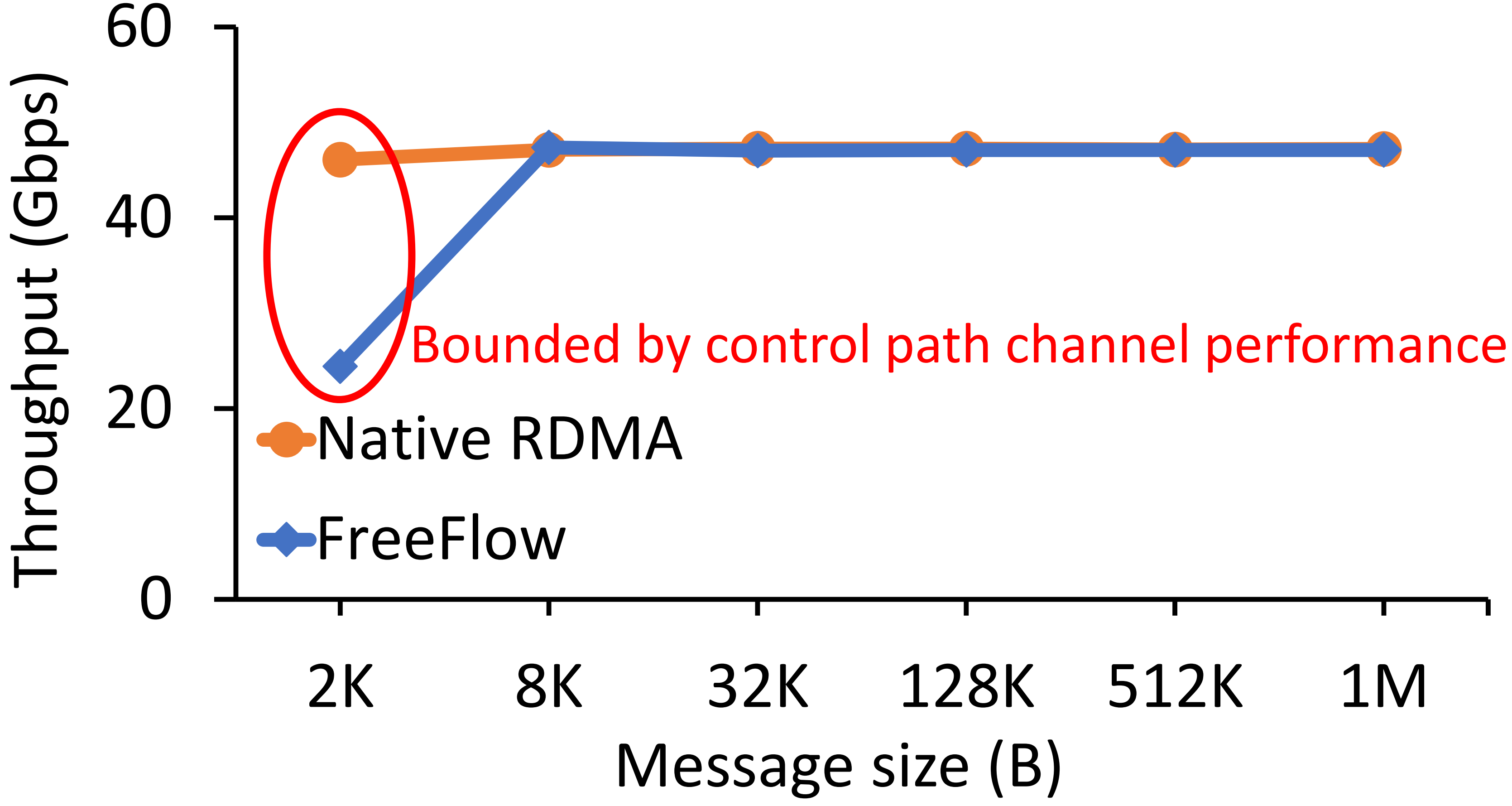
Implementation and Experimental Setup

- FreeFlow Library
  - Add 4000 lines in C to libibverbs and libmlx4.
- FreeFlow Router
  - 2000 lines in C++
- Testbed setup
  - Two Intel Xeon E5-2620 8-core CPUs, 64 GB RAM
  - 56 Gbps Mellanox ConnectX-3 NICs
  - Docker containers

Does FreeFlow Support Low Latency?

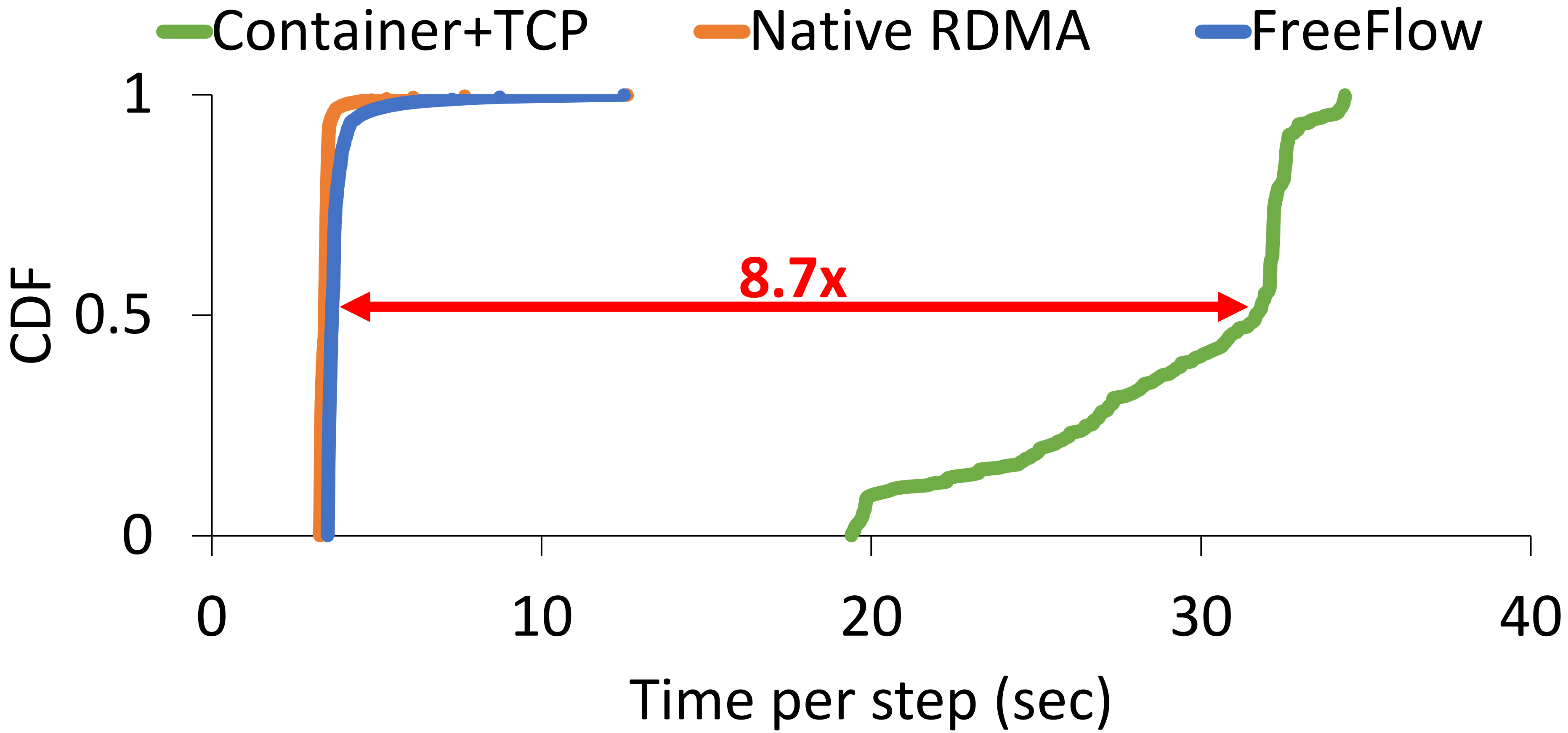


Does FreeFlow Support High Throughput?



Bounded by control path channel performance

Do Applications Benefit from FreeFlow?



Summary

- Containerization today can’t benefit from speed of RDMA.
- Existing solutions for NIC virtualization don’t work (e.g., SR-IOV).

- FreeFlow enables containerized apps to use RDMA.
- Challenges and Key Ideas
  - Control path: Leveraging Verbs library structure for efficient Verbs forwarding
  - Data path: Zero-copy memory synchronization
- Performance close to native RDMA

github.com/microsoft/freeflow