OneSidedMW

Managing Disaggregated Memory Efficiently, Flexibly, and Securely with RNIC Offloading

Zixuan Wang, Jinyu Gu, Xingda Wei, Yubin Xia

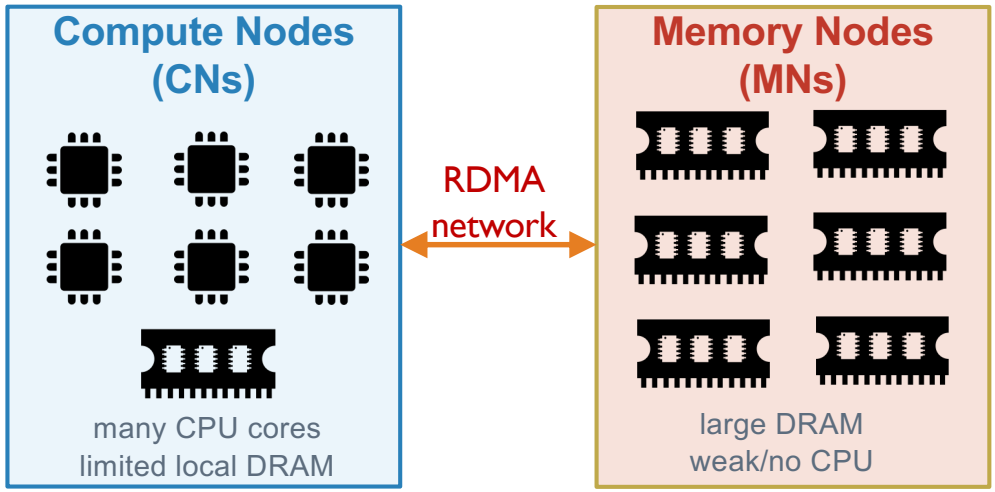
Institute of Parallel and Distributed Systems (IPADS)
Shanghai Jiao Tong University

Background: Disaggregated Memory Architecture

- Separate compute and memory into network-attached pools

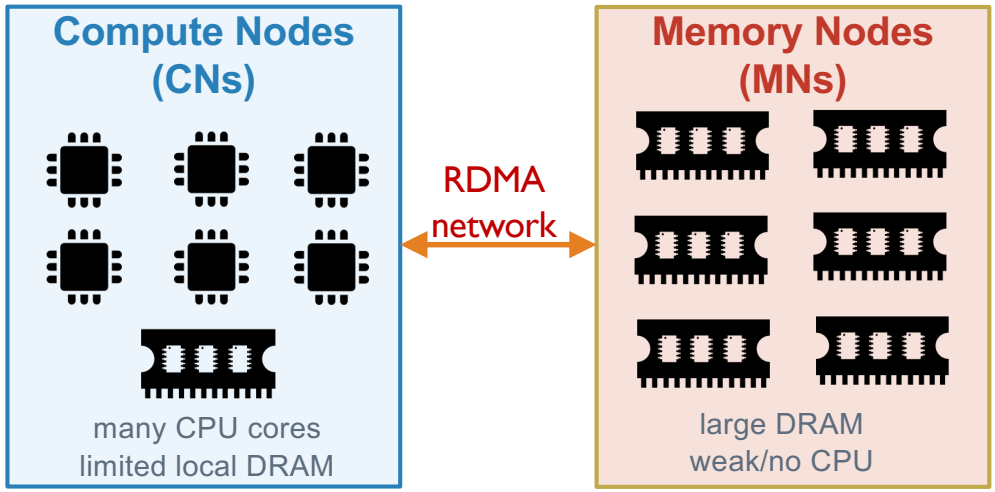
Background: Disaggregated Memory Architecture

- **Separate compute and memory into network-attached pools**
  - Compute Nodes (CNs): strong CPUs, limited local memory
  - Memory Nodes (MNs): abundant DRAM, weak/no CPU
  - CNs access MN memory via RDMA with microsecond-level latency



Background: Disaggregated Memory Architecture

- **Separate compute and memory into network-attached pools**
  - Compute Nodes (CNs): strong CPUs, limited local memory
  - Memory Nodes (MNs): abundant DRAM, weak/no CPU
  - CNs access MN memory via RDMA with microsecond-level latency



Disaggregated Memory Management:

MNs must: (1) handle remote memory allocation and free requests from CNs, and (2) ensure memory isolation among CNs.

Key tradeoff: fine-grained allocation → high remote memory utilization, but high overhead on the weak MN CPU

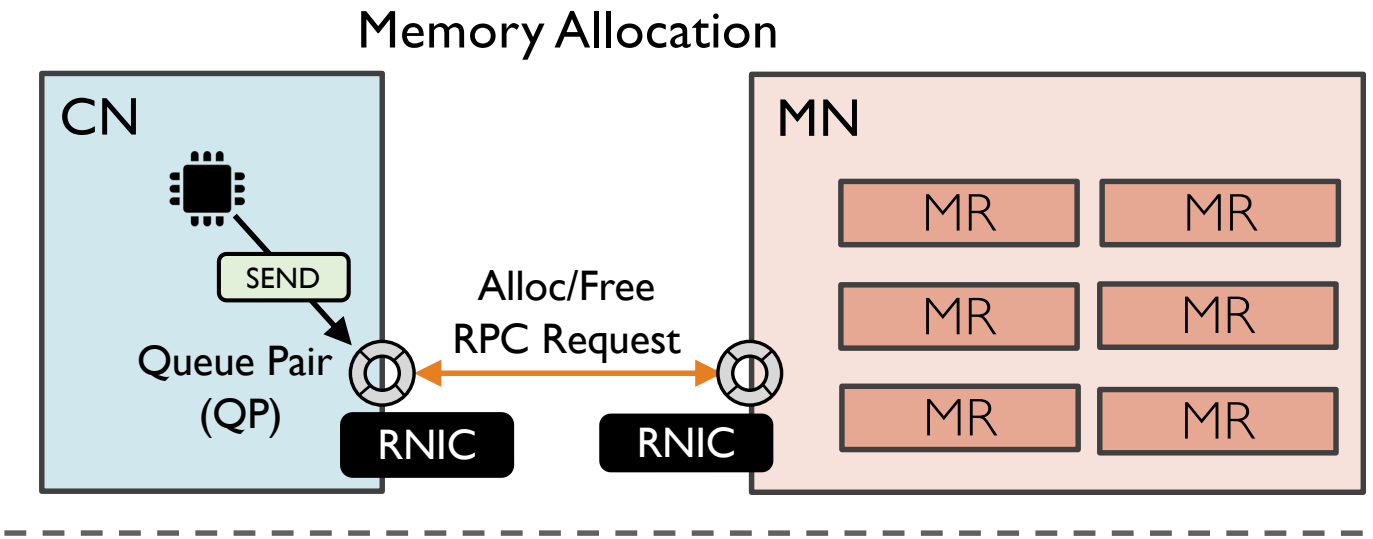
Limitations of Existing Memory Management Approaches

Static-MR

e.g., Fastswap (Eurosys'20)

Technique:

- Memory Registration
- MNs register coarse-grained memory chunks as Memory Regions (i.e., MRs)



Limitations of Existing Memory Management Approaches

Static-MR

e.g., Fastswap (Eurosys'20)

Technique:

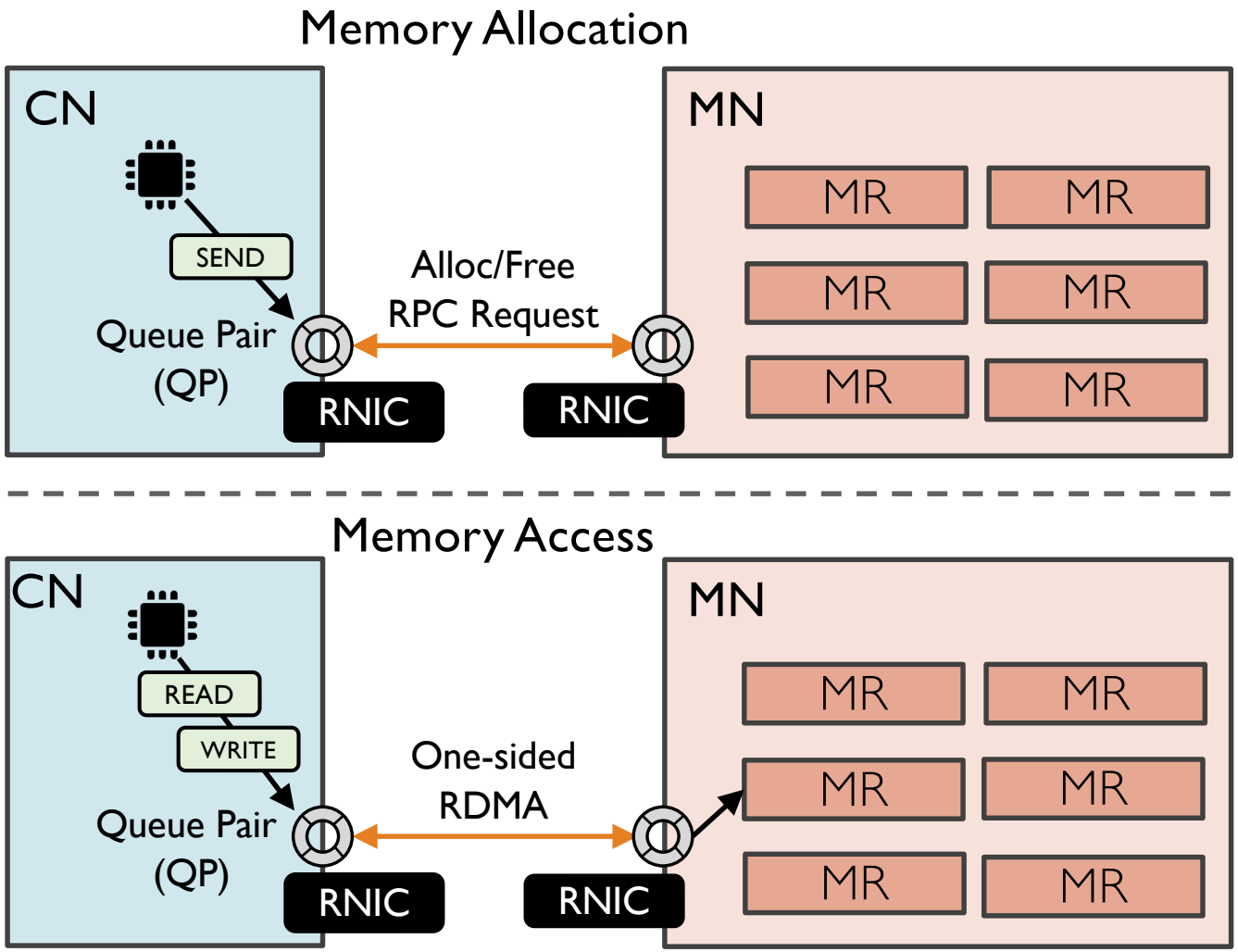
- Memory Registration
- MNs register coarse-grained memory chunks as Memory Regions (i.e., MRs)

✓ Pros

- Minimal MN CPU involvement
- Fast access with one-sided RDMA

✗ Cons

- Severe internal fragmentation
- Poor remote memory utilization



Limitations of Existing Memory Management Approaches

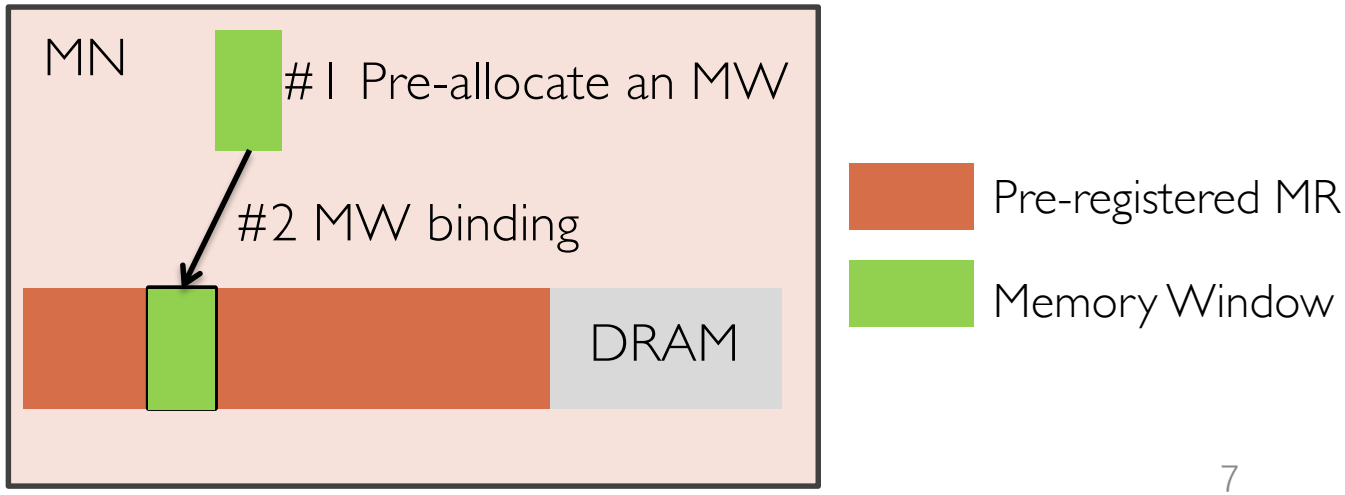
RPC-MW

e.g., Patronus (FAST'23)

Technique:

- Memory Window (MW)
- MNs bind MWs to fine-gained memory chunks and allocate them to CNs on demand.

- **MW – a light-weight memory protection mechanism**
  - MW can be bound to a very ***small*** part of an MR.
  - MW binding is extremely ***fast*** (microsecond-level latency).
  - MW binding ops ***can be performed in user space.***



Limitations of Existing Memory Management Approaches

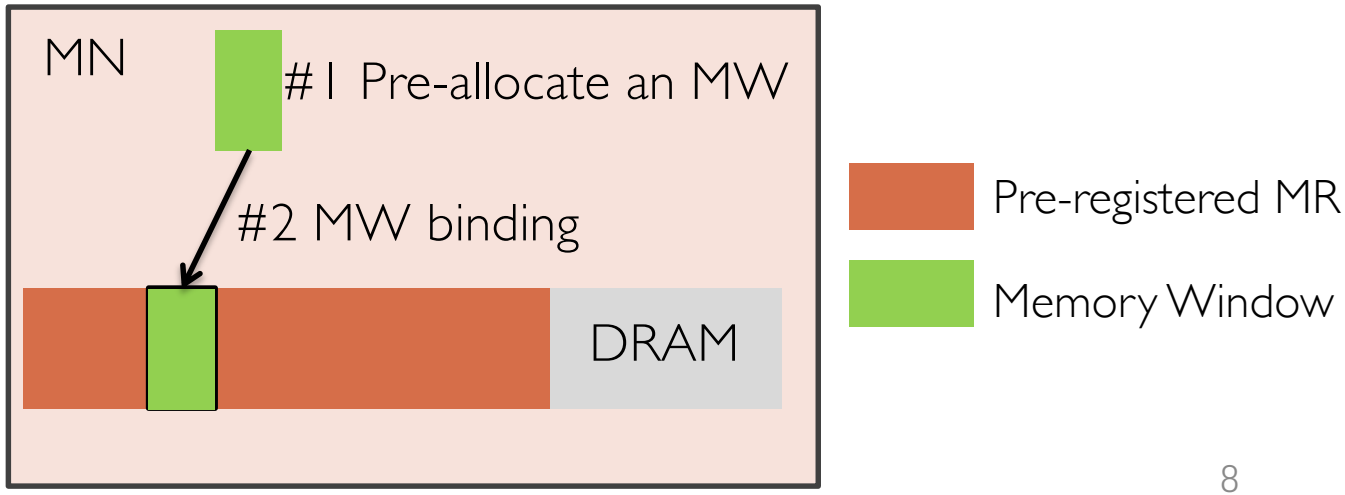
RPC-MW
e.g., Patronus (FAST'23)

Technique:
- Memory Window (MW)
- MNs bind MWs to fine-gained memory chunks and allocate them to CNs on demand.

✓ Pros
- Improved memory utilization
- Fast access with one-sided RDMA

✗ Cons
- Requires MN CPU involvement for every Alloc/Free request
- High performance overhead when MN CPU is overwhelmed

- **MW – a light-weight memory protection mechanism**
  - MW can be bound to a very ***small*** part of an MR.
  - MW binding is extremely ***fast*** (microsecond-level latency).
  - MW binding ops ***can be performed in user space.***

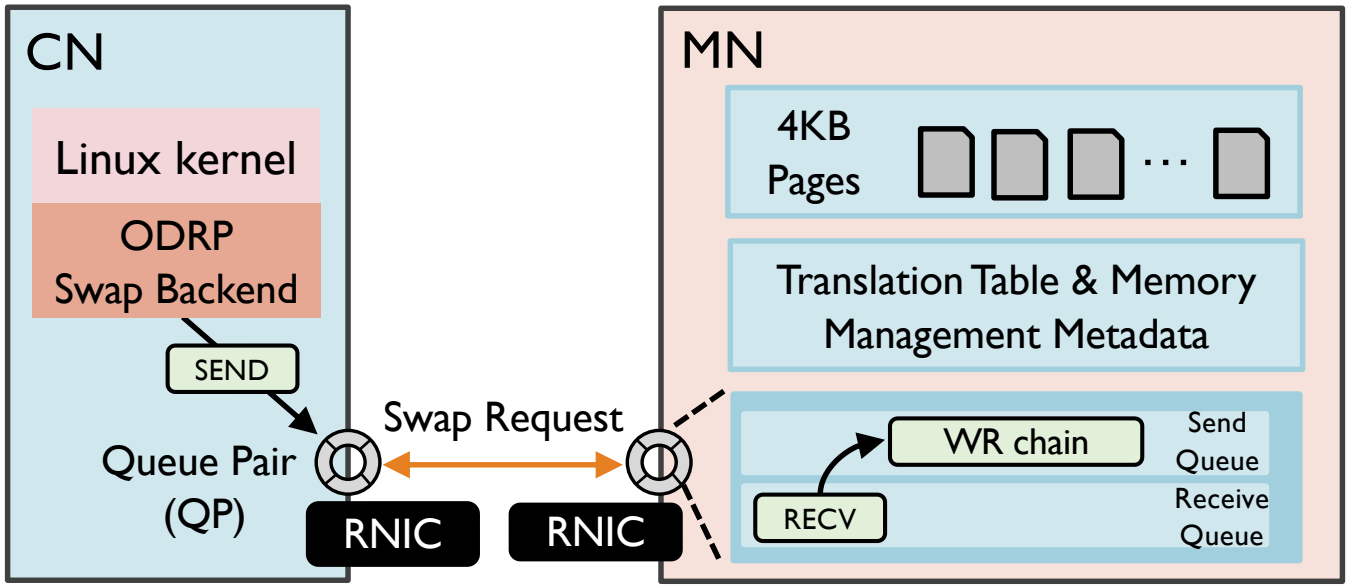


Limitations of Existing Memory Management Approaches

ODRP
NSDI'25

Technique:
- RNIC offloading
- MNs divide memory into 4KB pages
- MNs offload all access & management logic to their RNICs via WR chains

✓ Pros
- No MN CPU involvement
- Optimal memory utilization (page-level allocation)



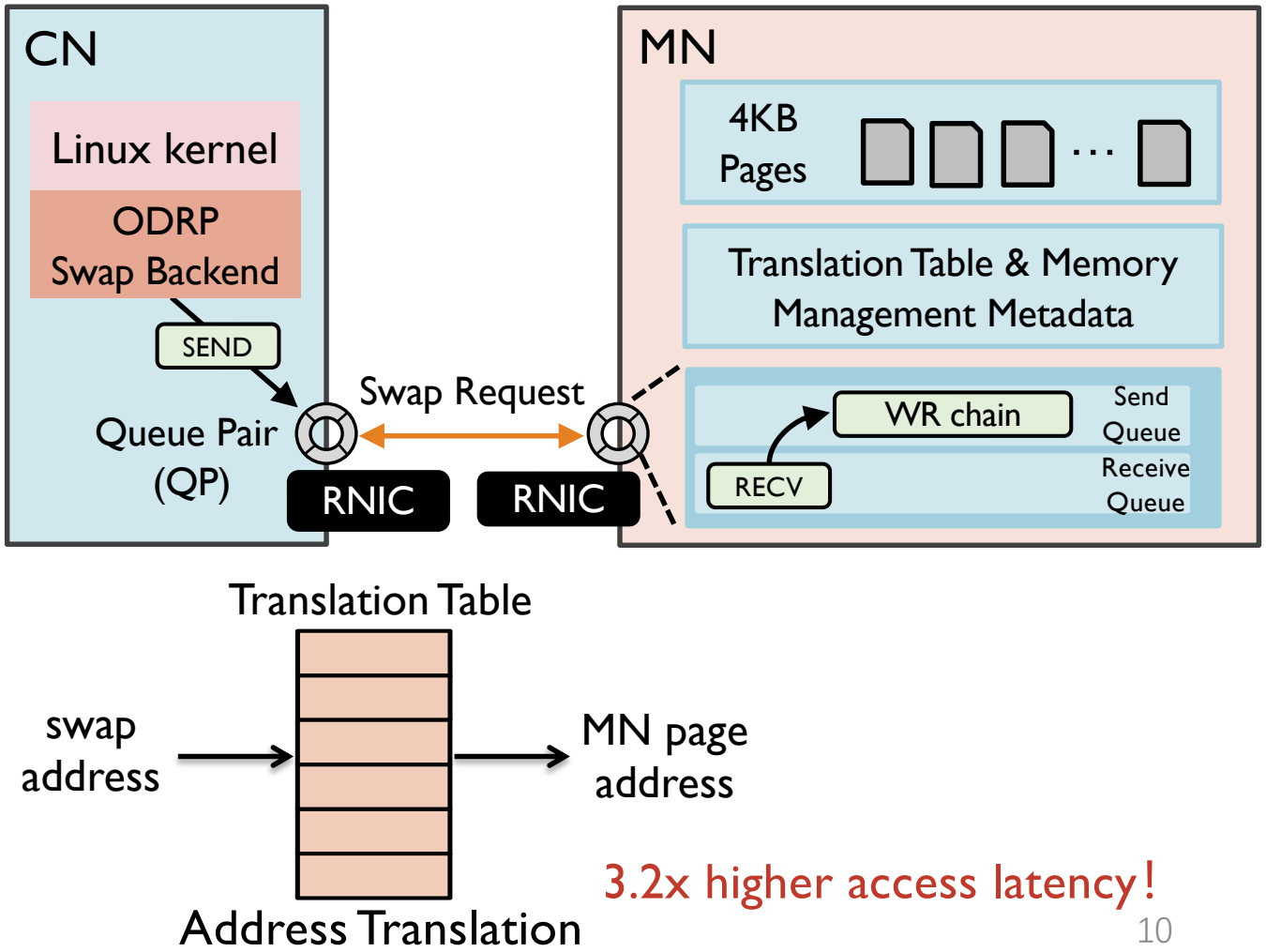
Limitations of Existing Memory Management Approaches

ODRP
NSDI'25

Technique:
- RNIC offloading
- MNs divide memory into 4KB pages
- MNs offload all access & management logic to their RNICs via WR chains

✓ Pros
- No MN CPU involvement
- Optimal memory utilization (page-level allocation)

✗ Cons
- Every access incurs additional address translation overhead
- Fixed 4KB access granularity



Key Insight & Design Goals

None of the existing approaches achieves: high utilization + no MN CPU involvement + high performance + arbitrary access granularity simultaneously.

Key Insight

Retrofit RNIC offloading to perform MW binding/unbinding operations remotely → fine-grained remote memory allocation/deallocation without any MN CPU involvement.

⚡ No MN CPU Involvement

- Offloads memory allocation/deallocation logic to the MN's RNIC
- Bypasses the wimpy MN CPU entirely

💾 High Memory Utilization

- Fine-grained allocation with configurable chunk size (e.g., 1MB for swap, 16KB for KVS)

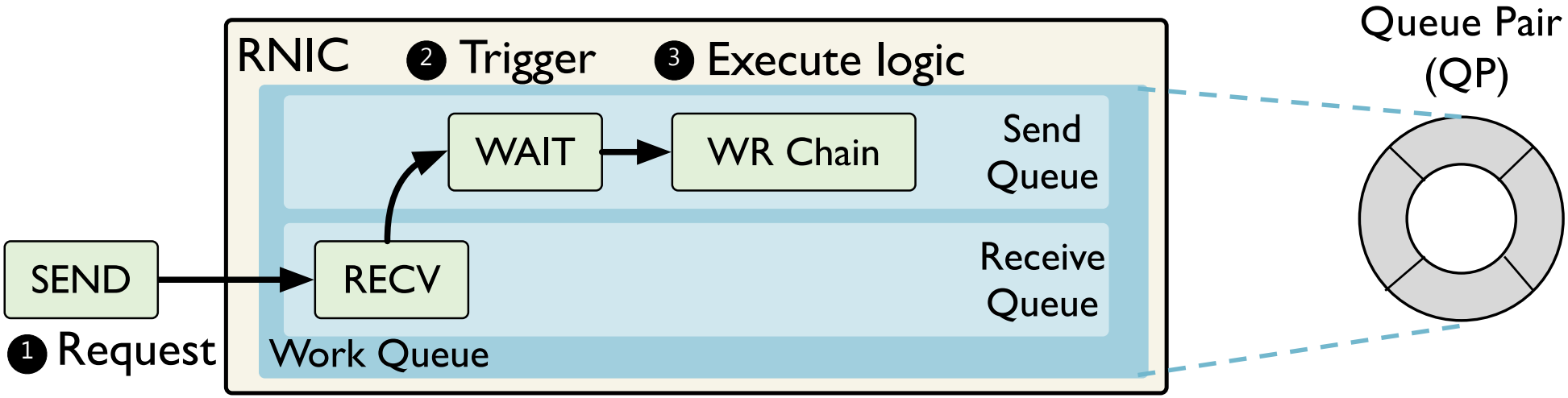
🚀 High Performance

- Highly efficient remote memory allocation
- Memory access via native one-sided RDMA

🔒 Strong Isolation & Flexible I/O

- Supports arbitrary I/O sizes
- Enforces hardware-level isolation with Type-2 MW

Quick Recap on RNIC Offloading



Invocable: Clients trigger logic via RDMA

- RDMA WAIT suspends execution until a SEND arrives
- Incoming SEND triggers the pre-programmed WR chain on the RNIC
- Logic executes entirely on the RNIC — no CPU involvement

Expressive: Native RDMA primitives suffice

- Chain basic WRs (READ, WRITE, FAA/CAS) into WR chains to express complex logic
- WR chains are Turing-complete — no custom hardware needed (RedN NSDI'22)

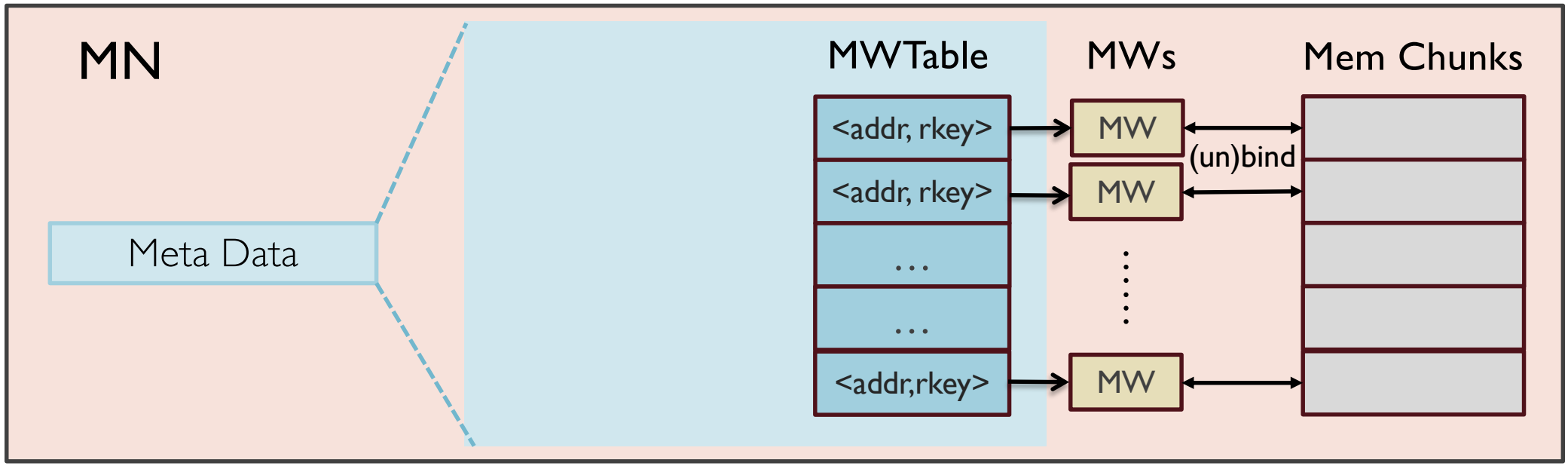
OneSidedMW Design

#1 The MN organizes its memory into fixed-size chunks (configurable granularity).

OneSidedMW Design

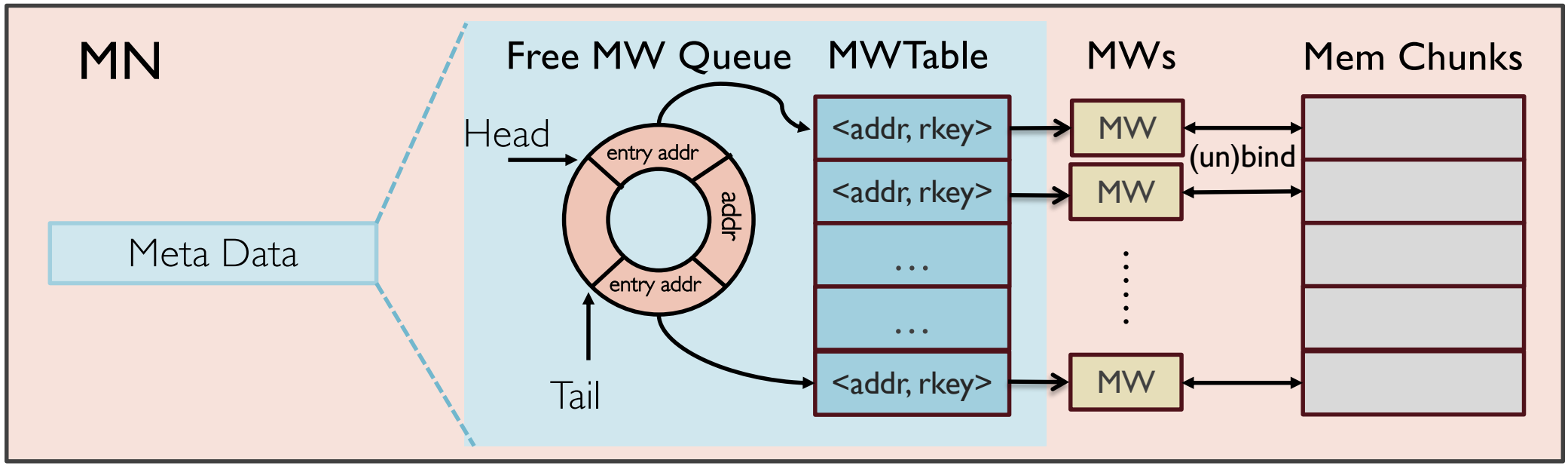
#2 Pre-allocates an unbound MW for each chunk.

OneSidedMW Design



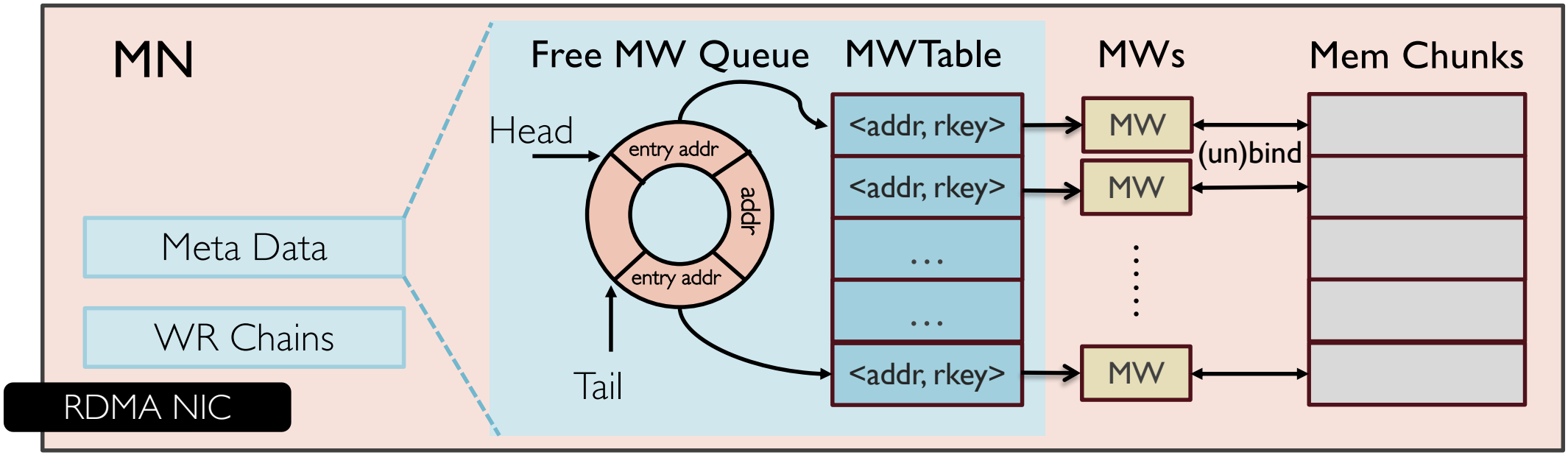
#3 Maintains an MWTable storing each chunk's base address and MW rkey.

OneSidedMW Design



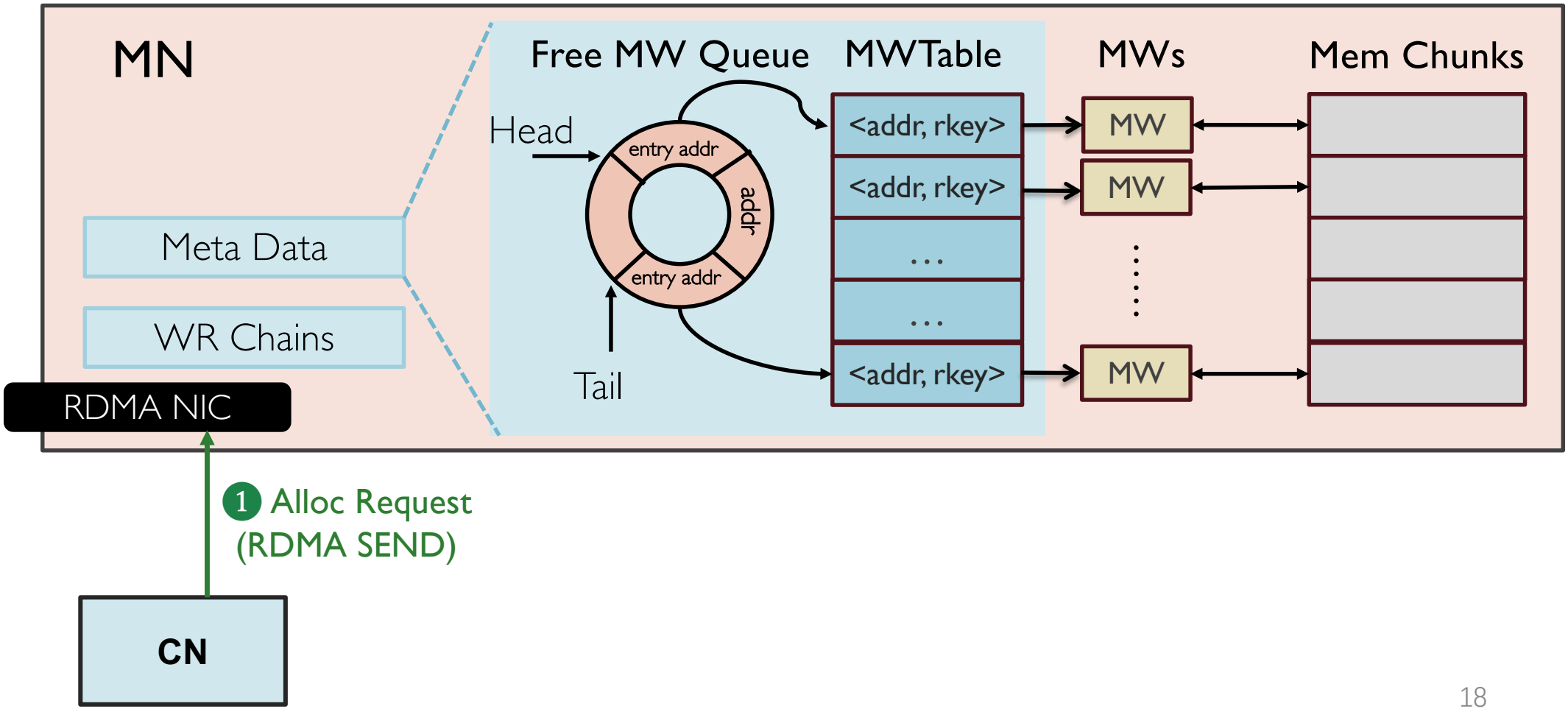
#4 Uses a free MW queue to track available chunks; each entry contains the address of an MWTable record.

OneSidedMW Design

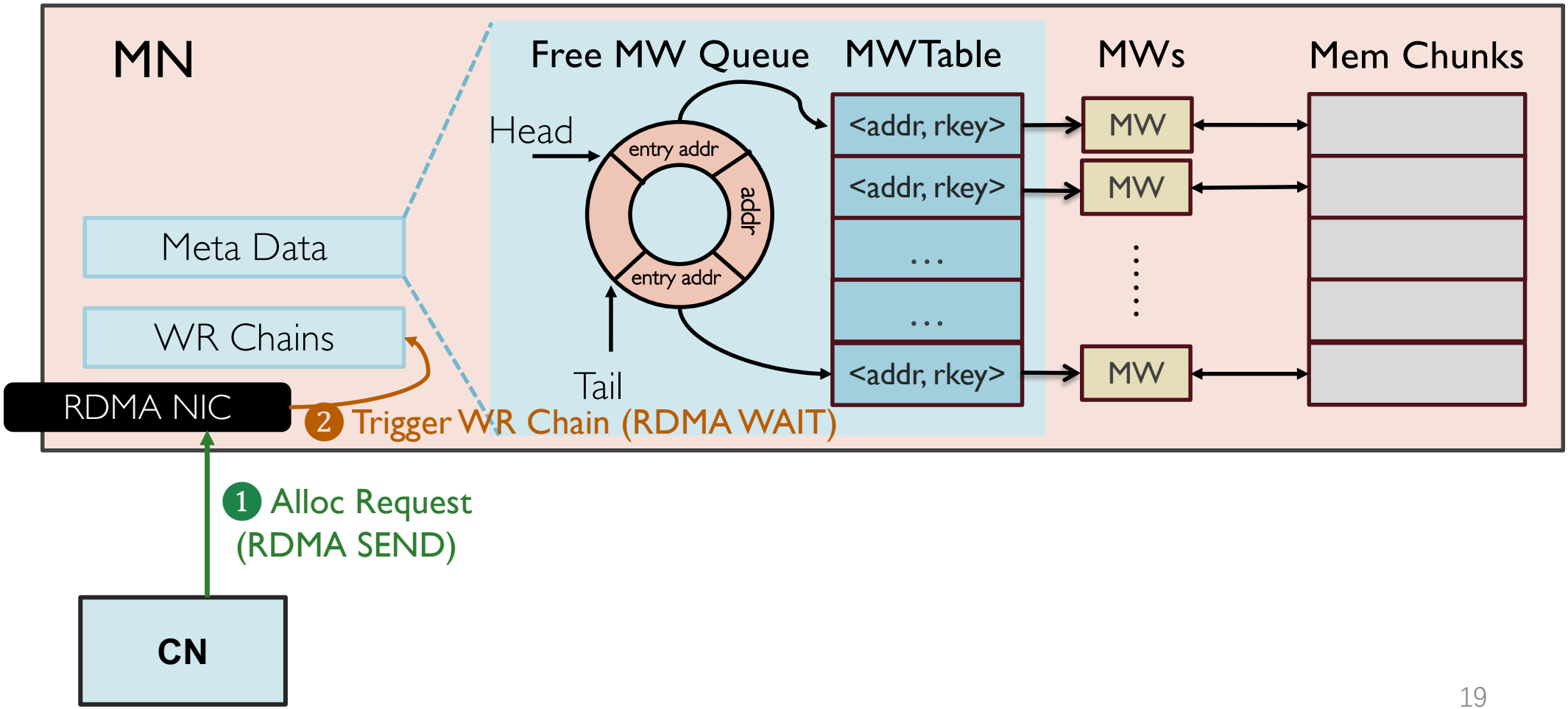


#5 Offloads WR chains to the RNIC to handle memory (de)allocation and MW (un)binding — no MN CPU involved

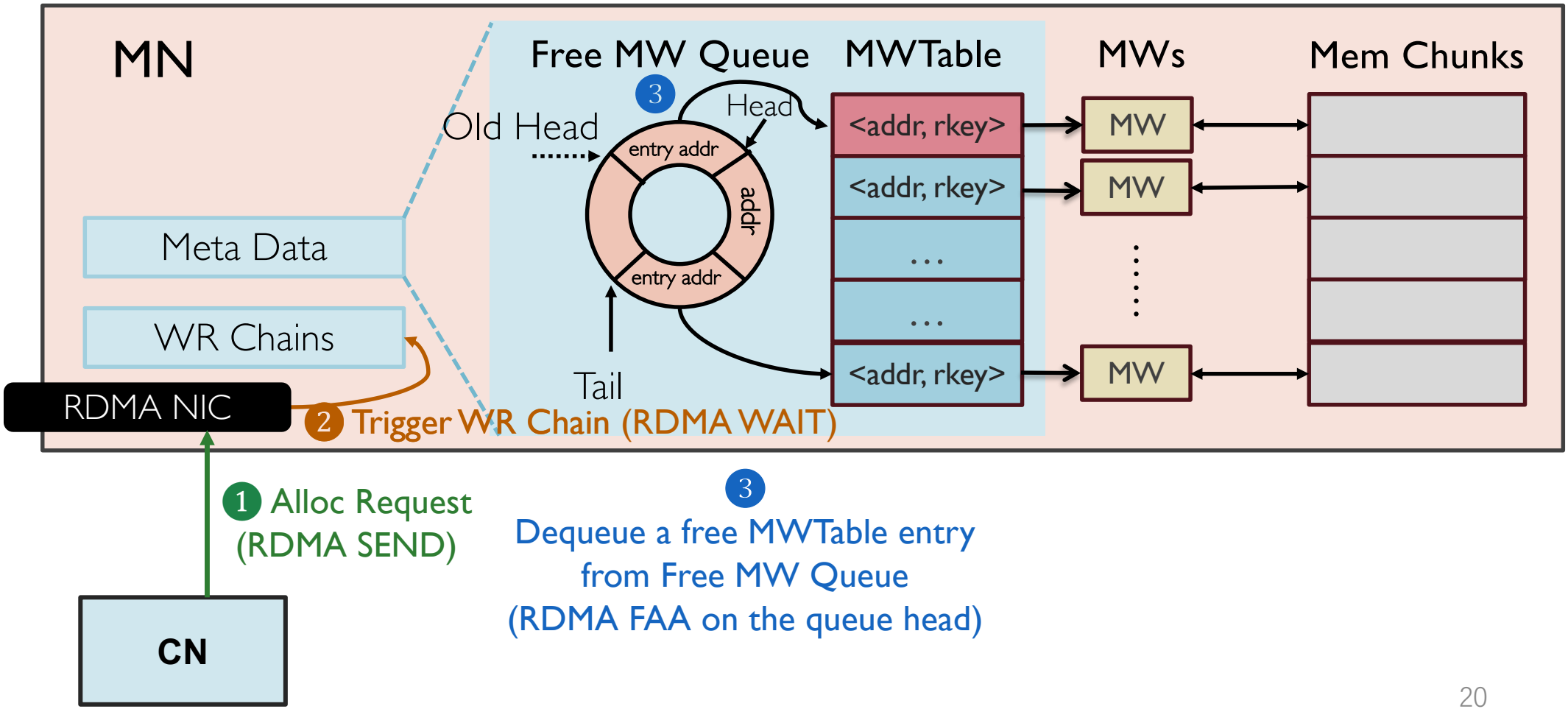
Remote Memory Allocation Workflow



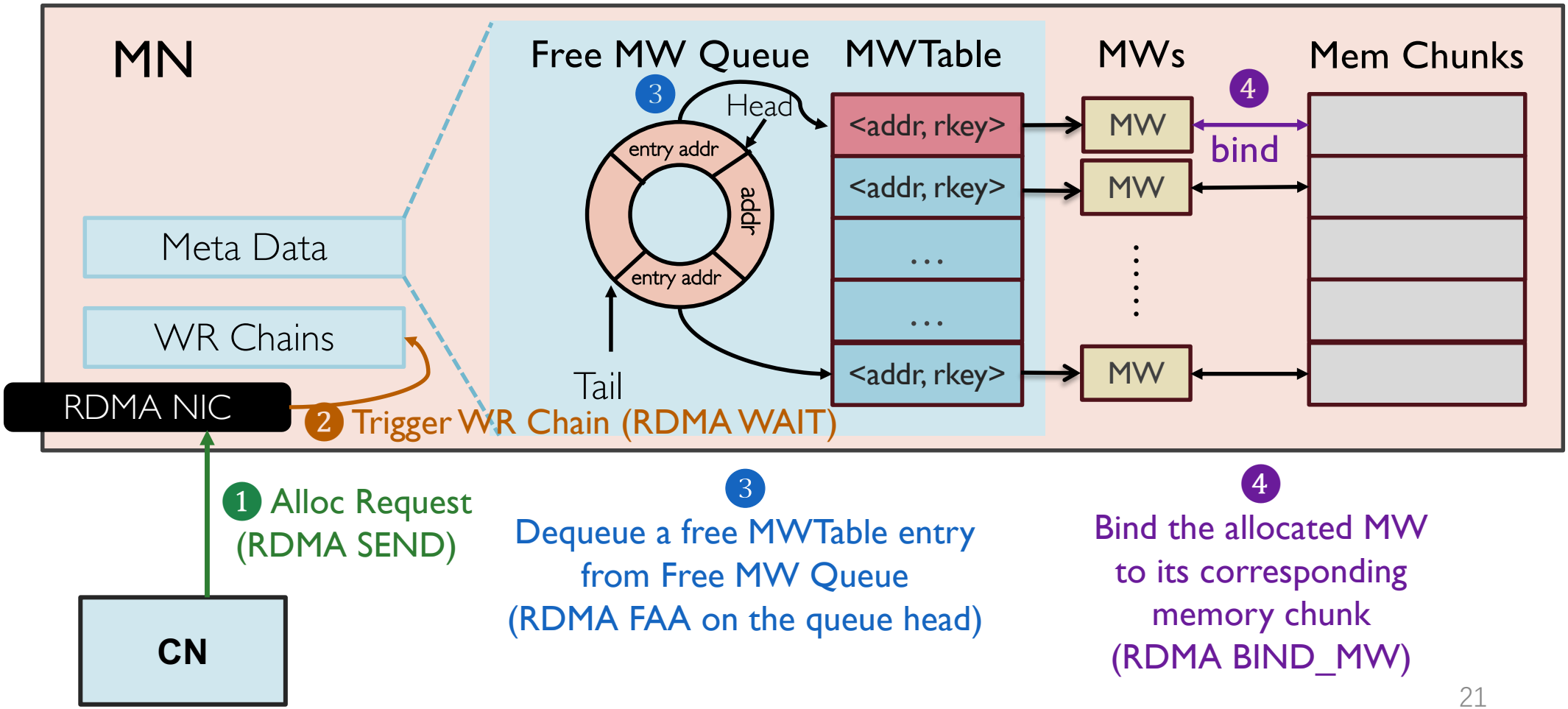
Remote Memory Allocation Workflow



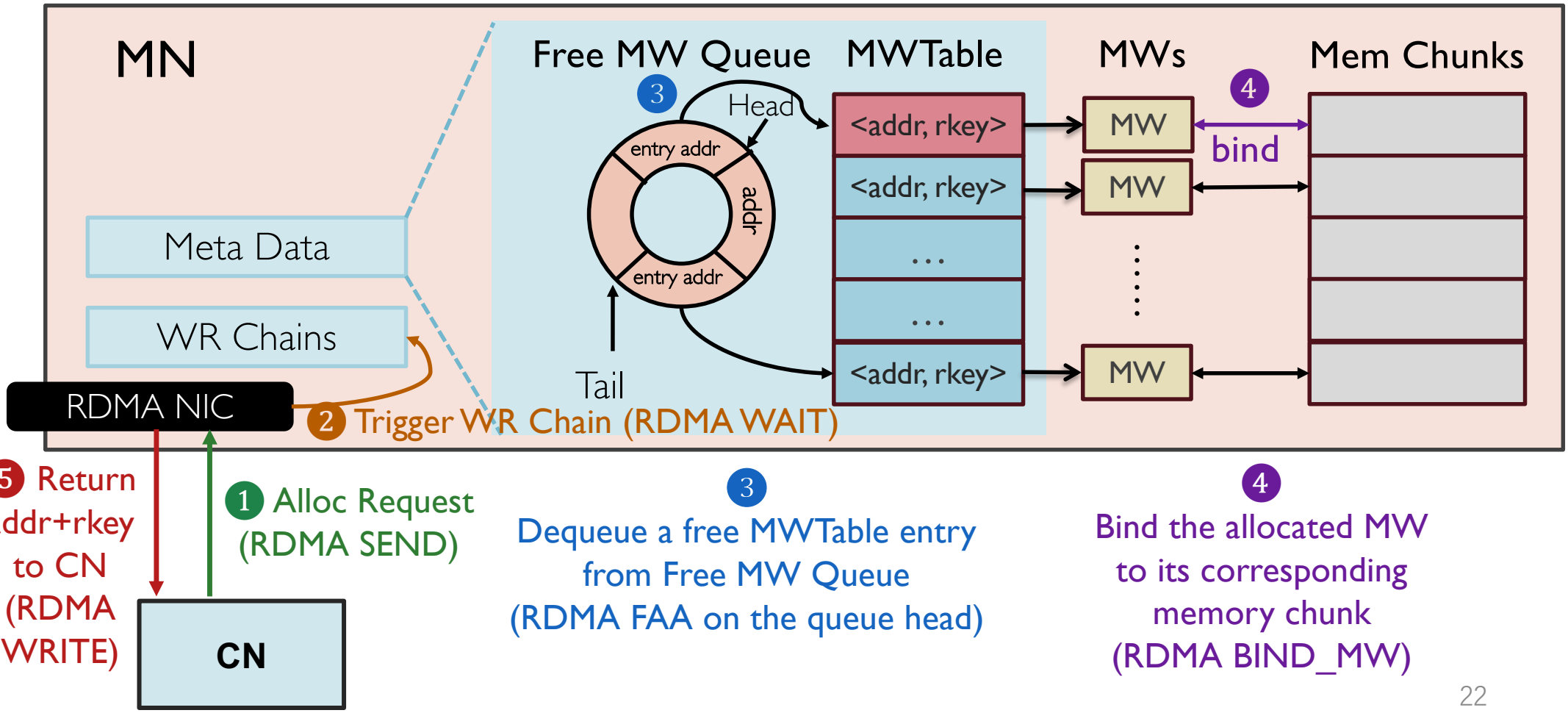
Remote Memory Allocation Workflow



Remote Memory Allocation Workflow



Remote Memory Allocation Workflow

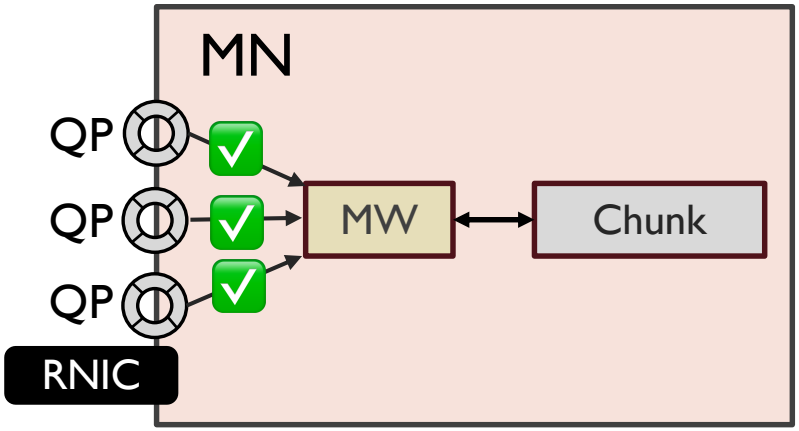


Type-1 MW vs. Type-2 MW

- Why Not Type-1 MWs? — A Security Vulnerability

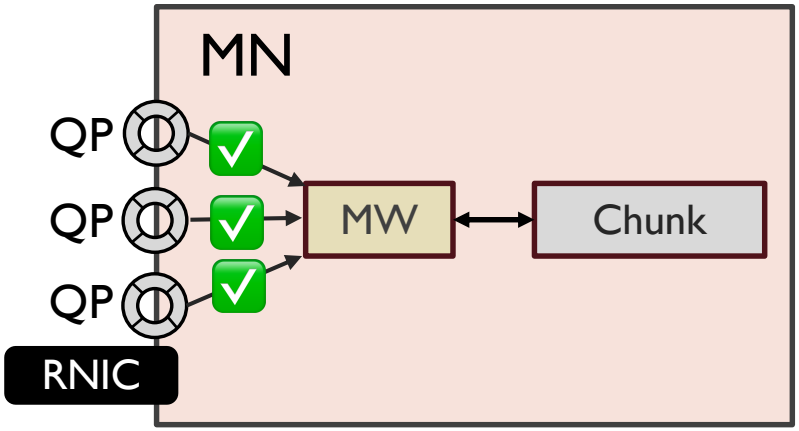
Type-1 MW vs. Type-2 MW

- **Why Not Type-1 MWs? — A Security Vulnerability**
  - After MW binding, **any QP** can access the memory area as long as it provides the correct **rkey.**



Type-1 MW vs. Type-2 MW

- **Why Not Type-1 MWs? — A Security Vulnerability**
  - After MW binding, **any QP** can access the memory area as long as it provides the correct **rkey.**
  - After rebinding, only the **lower 8 bits** of the rkey change — the upper 24 bits remain unchanged (used by RNIC hardware to index MW metadata).



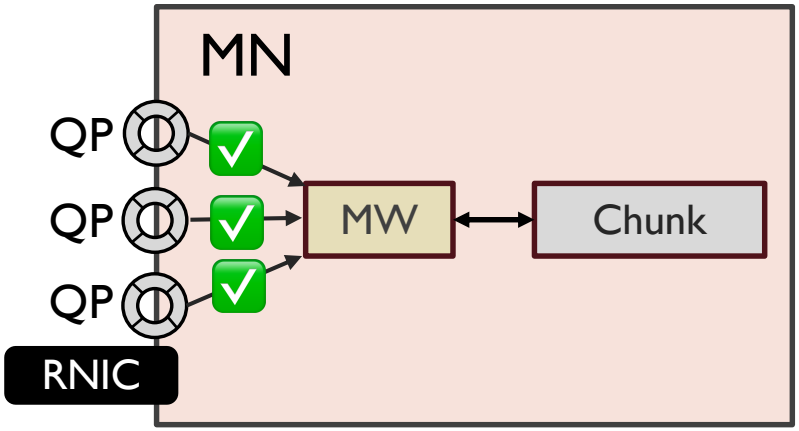
rkey structure (32 bits):

Upper 24 bits — unchanged after rebinding (RNIC hardware index, same across reallocations)	Lower 8 bits changes on rebind

→ Only 256 possible values: easily brute-forced!

Type-1 MW vs. Type-2 MW

- **Why Not Type-1 MWs? — A Security Vulnerability**
  - After MW binding, **any QP** can access the memory area as long as it provides the correct **rkey.**
  - After rebinding, only the **lower 8 bits** of the rkey change — the upper 24 bits remain unchanged (used by RNIC hardware to index MW metadata).
  - A malicious CN can enumerate all **256 (2^8)** possible rkeys to gain unauthorized access to memory reallocated to another CN.



rkey structure (32 bits):

Upper 24 bits — unchanged after rebinding (RNIC hardware index, same across reallocations)	Lower 8 bits changes on rebind

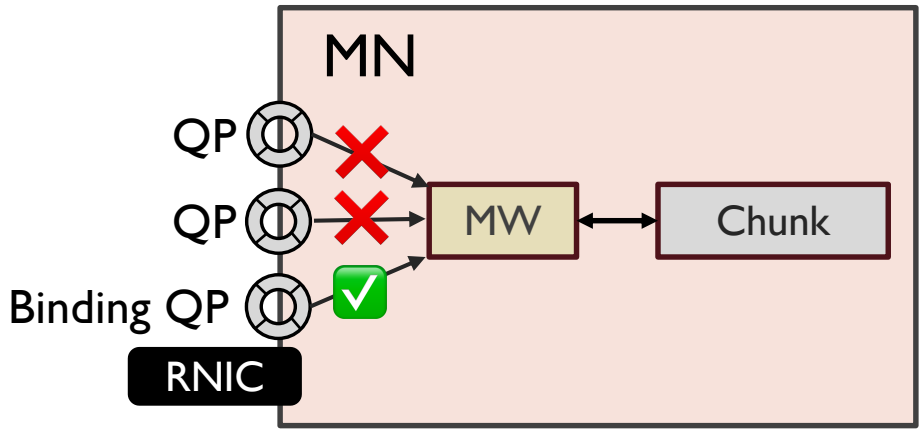
→ *Only 256 possible values: easily brute-forced!*

Type-1 MW vs. Type-2 MW

- Type-2 MWs for Strong Isolation

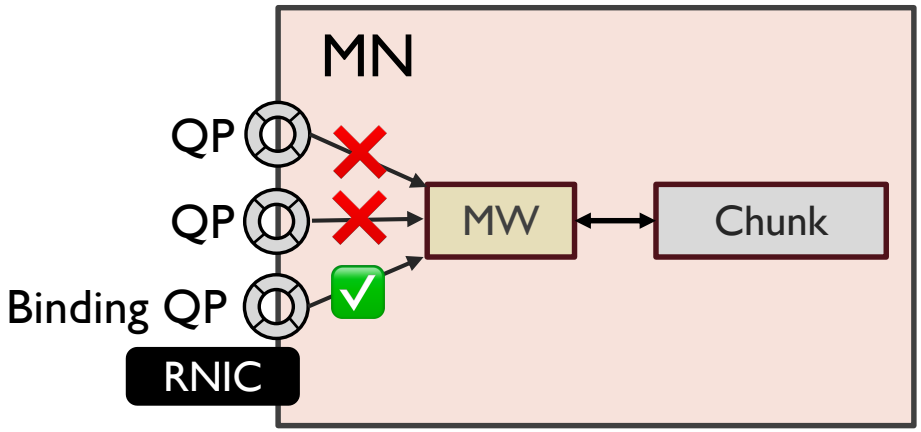
Type-1 MW vs. Type-2 MW

- **Type-2 MWs for Strong Isolation**
  - **Exclusive Assignment:** after MW binding, the MW is **exclusively assigned to the QP that performed the RDMA BIND_MW operation** — only this QP can access the memory area or unbind the MW.



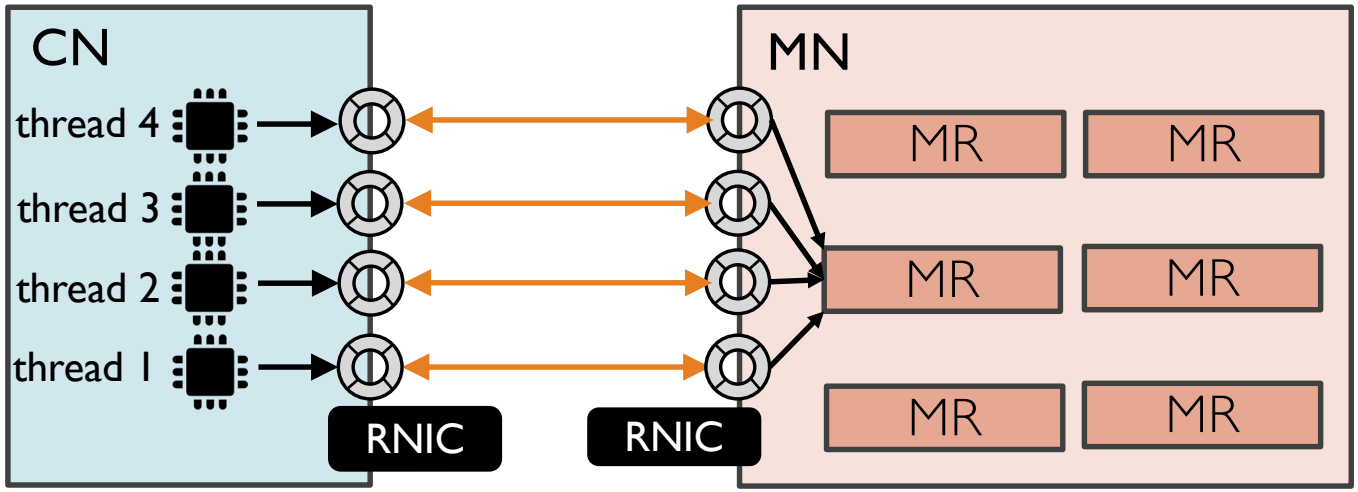
Type-1 MW vs. Type-2 MW

- **Type-2 MWs for Strong Isolation**
  - **Exclusive Assignment:** after MW binding, the MW is **exclusively assigned to the QP that performed the RDMA BIND_MW operation** — only this QP can access the memory area or unbind the MW.
  - Access from any other QP is **rejected** by RNIC hardware.



Type-2 MW: Strong Isolation, but at a Cost

- **Multiple QP Connections for High-Performance Memory Access**
  - CNs typically establish multiple QP connections with the MN — one per CPU core or thread
  - **Benefit #1:** Multiple QP connections enable concurrent memory access, **fully utilizing the RNIC's internal parallelism**
  - **Benefit #2:** Requests can be routed to different QPs to **prevent interference** — e.g., latency-sensitive requests are isolated from latency-insensitive requests

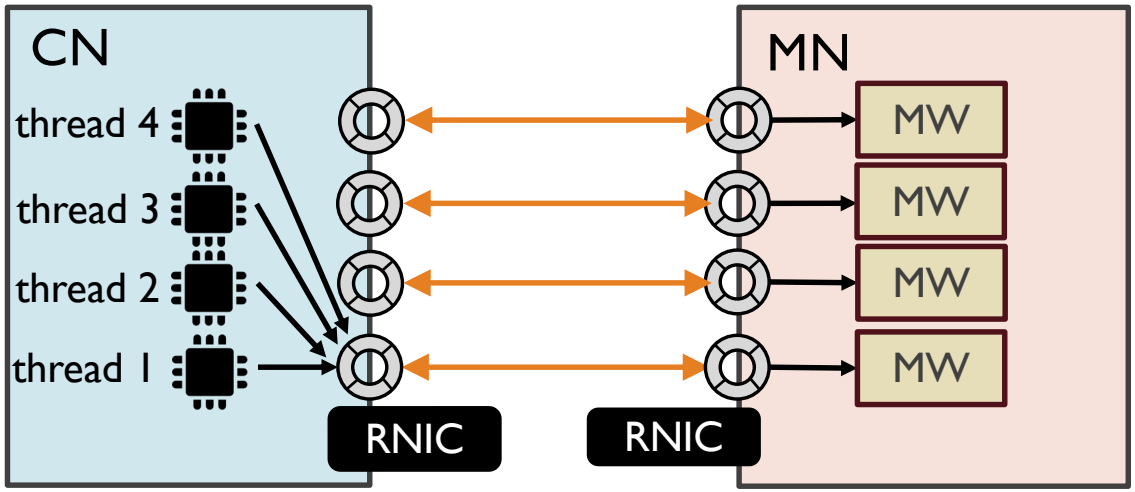


Type-2 MW: Strong Isolation, but at a Cost

- **Challenge #1: Performance interference within a QP**
  - A type-2 MW can only be accessed via the QP that performed its binding

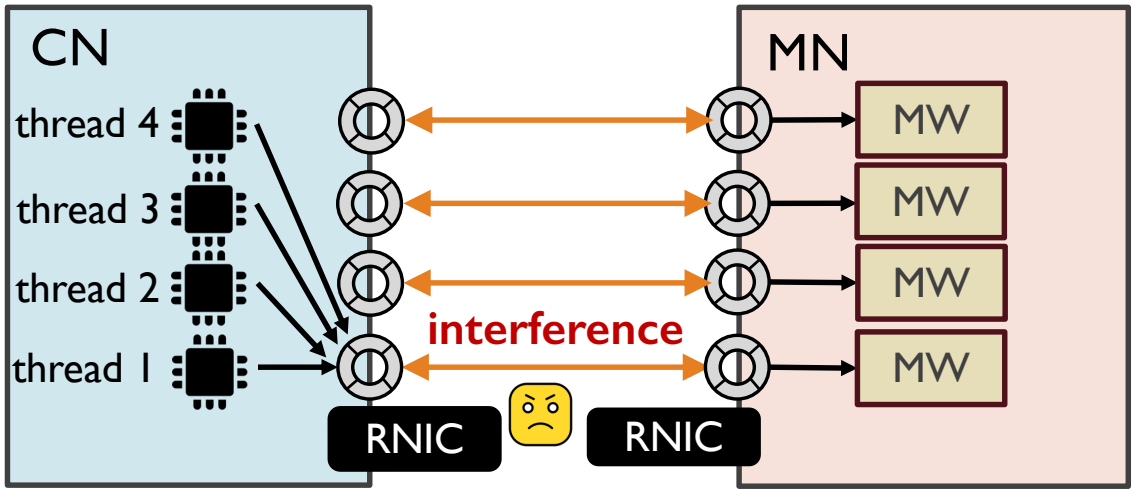
Type-2 MW: Strong Isolation, but at a Cost

- **Challenge #1: Performance interference within a QP**
  - A type-2 MW can only be accessed via the QP that performed its binding
  - All access requests to the same chunk must **share the same QP**



Type-2 MW: Strong Isolation, but at a Cost

- **Challenge #1: Performance interference within a QP**
  - A type-2 MW can only be accessed via the QP that performed its binding
  - All access requests to the same chunk must **share the same QP**
  - Latency-sensitive requests (e.g., swap-in) can be **delayed** by latency-insensitive ones (e.g., page prefetch)



Type-2 MW: Strong Isolation, but at a Cost

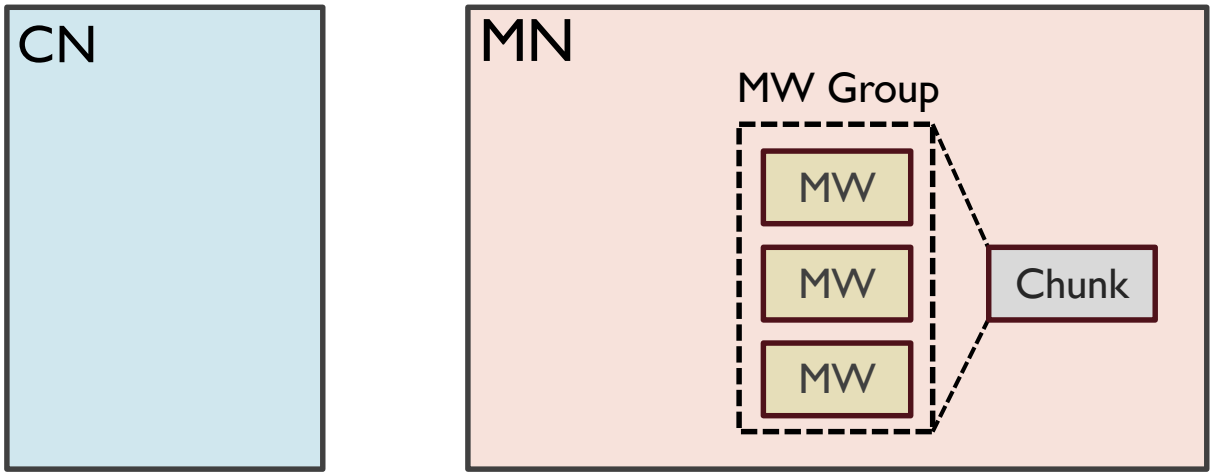
- **Challenge #1: Performance interference within a QP**
- **Challenge #2: Offloading overhead with multiple QPs**
  - Type-2 MW's exclusive assignment requires each QP to have its own offloaded WR chain to perform MW binding independently
  - Too many QPs with offloaded **logic degrades one-sided RDMA performance**
  - Up to **12.6% higher RDMA READ latency** with 96 QPs

Technique #1: QP and MW Grouping

- **Observation:** different MWs can be bound to overlapping memory areas

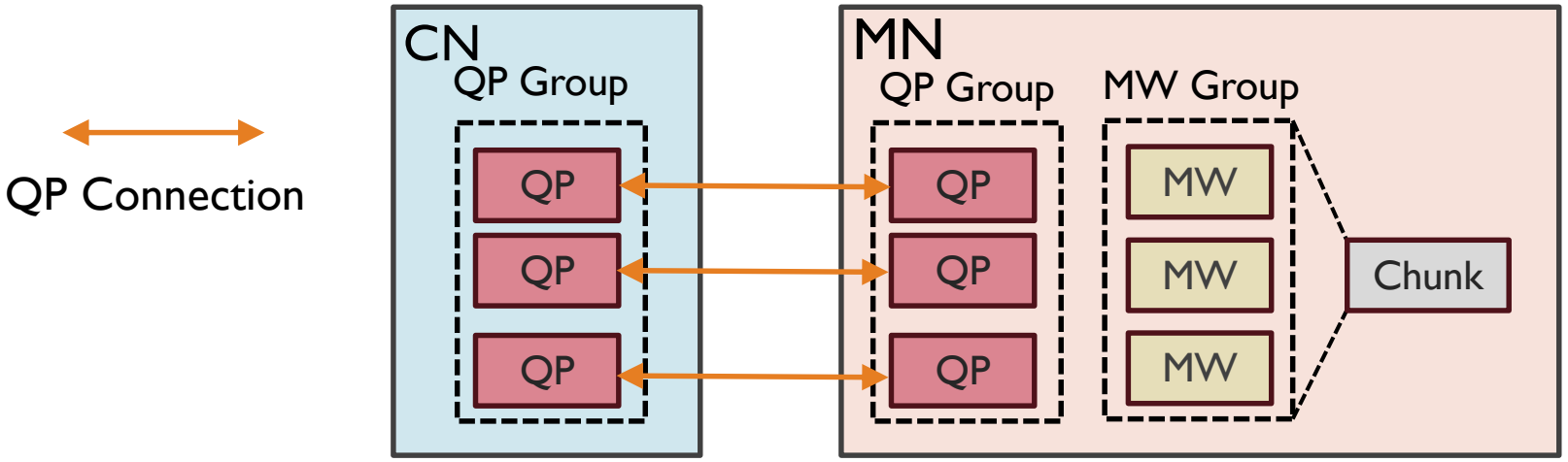
Technique #1: QP and MW Grouping

- **Observation: different MWs can be bound to overlapping memory areas**
- **Group-based allocation**
  - Pre-allocate multiple MWs (an MW group) for each memory chunk



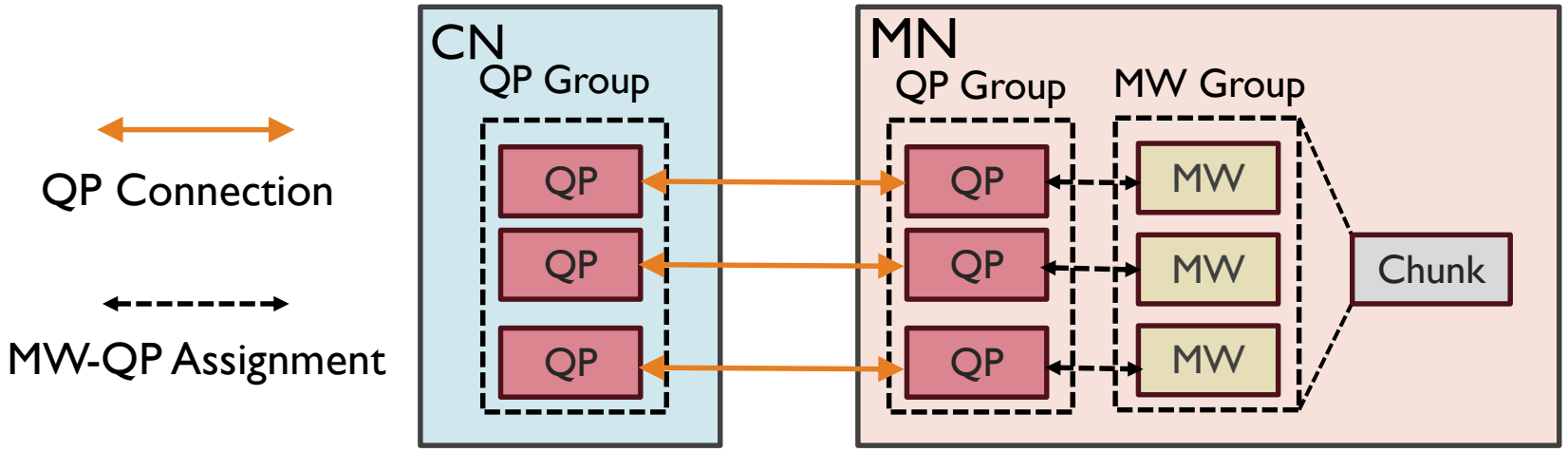
Technique #1: QP and MW Grouping

- **Observation: different MWs can be bound to overlapping memory areas**
- **Group-based allocation**
  - Pre-allocate multiple MWs (an MW group) for each memory chunk
  - Organize QP connections into QP groups



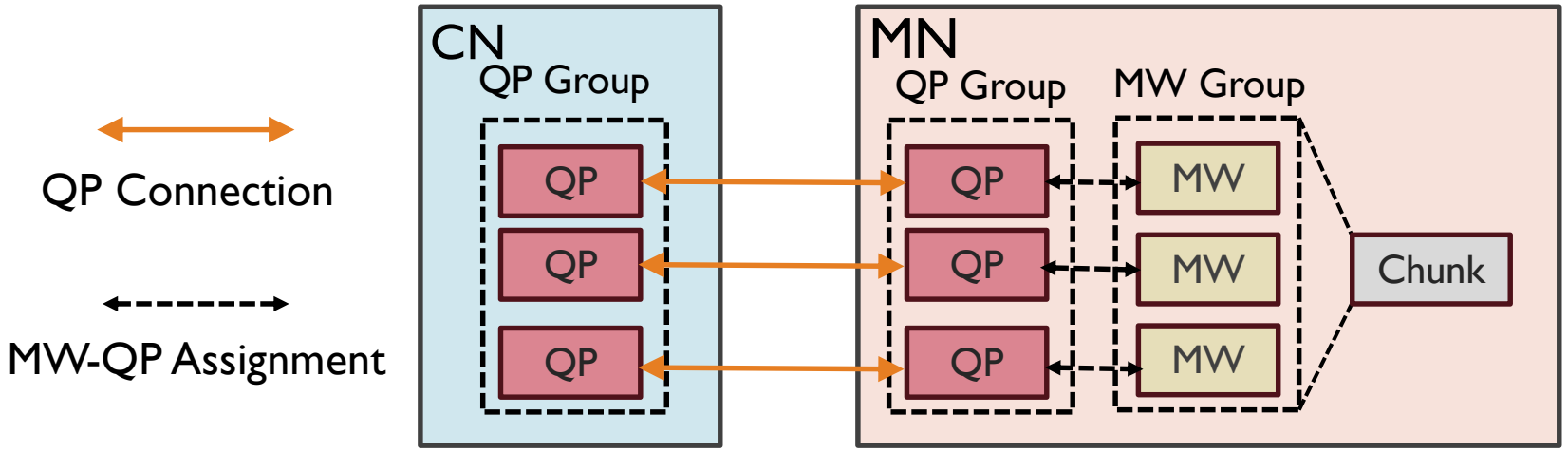
Technique #1: QP and MW Grouping

- **Observation: different MWs can be bound to overlapping memory areas**
- **Group-based allocation**
  - Pre-allocate multiple MWs (an MW group) for each memory chunk
  - Organize QP connections into QP groups
  - Bind each MW in the group to the same memory chunk, assigning each to a distinct QP



Technique #1: QP and MW Grouping

- **Observation: different MWs can be bound to overlapping memory areas**
- **Group-based allocation**
- **Access through multiple QPs**
  - Access the same memory chunk through through any QP in the QP group
  - Eliminate performance interference

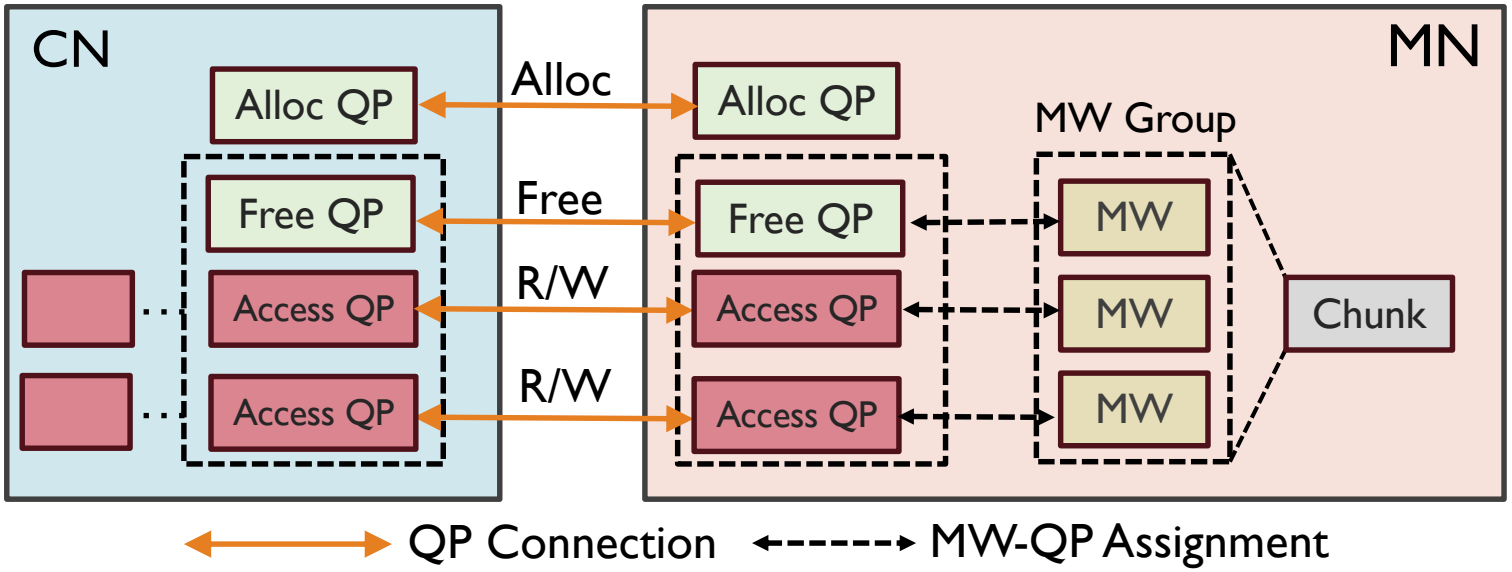


Technique #2: Management-Access QP Separation

- **Observation:** alloc/free operations happen less frequently than access operations

Technique #2: Management-Access QP Separation

- **Observation: alloc/free operations happen less frequently than access operations**
- **Management-Access QP Separation**
  - Designate a **small number of dedicated Alloc/Free QPs** (with offloaded WR chains) to handle memory (de)allocation
  - Set the qpn field in RDMA BIND_MW to **assign the allocated MWs to Access QPs**



Experimental Setup

- **Hardware setup**
  - **CPU:** 12-Core Intel Xeon E5-2650 CPU
  - **DRAM:** 224 GB DDR4 RAM
  - **RNIC:** 100 Gbps Mellanox ConnectX-5 RNIC
  - **Cluster:** 6 CNs, 1 MN (only use one CPU core)
- **Other baselines**
  - Static-MR[1,2]: pre-registering course-grained MRs
  - RPC-MW[3]: using RPC and MW binding for fine-grained memory allocation
  - ODRP[4] : offloading on-demand paging logic on the MN's RNIC

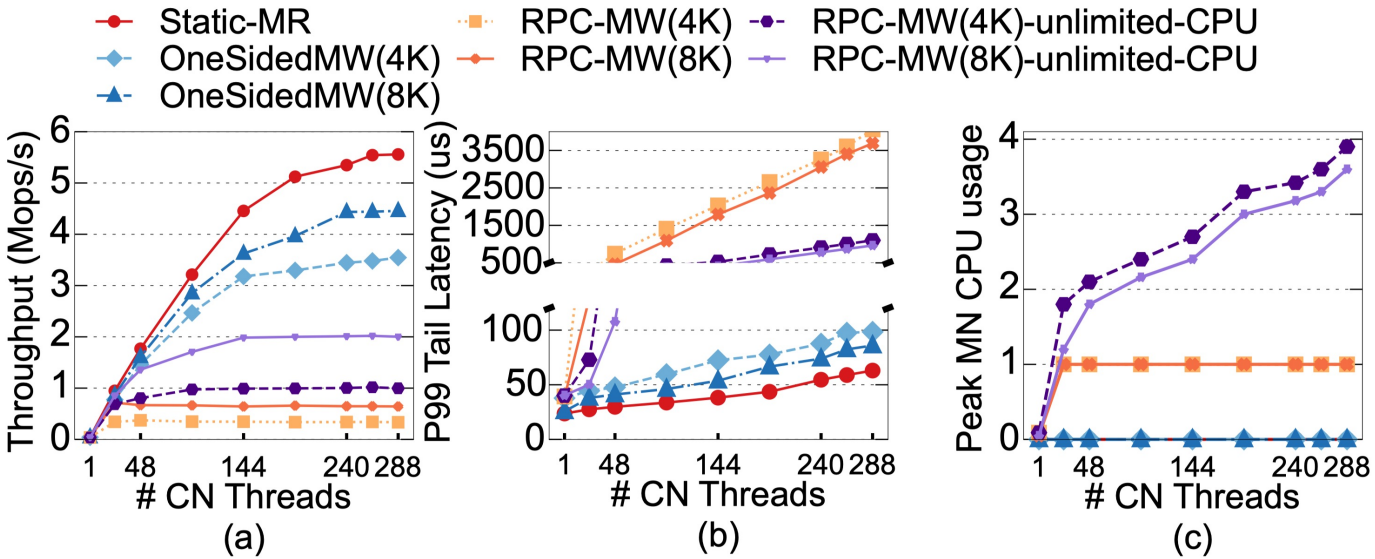
[1] One-sided rdma-conscious extendible hashing for disaggregated memory (ATC'21)
[2] Can far memory improve job throughput? (Eurosys'20)
[3] Patronus: High-performance and protective remote memory (FAST'23)
[4] ODRP: On-Demand Remote Paging with Programmable RDMA (NSDI'25)

Results – Disaggregated KVS

- Integration with Disaggregated Key-Value Store (RACE Hashing)
  - MN divides memory into **fine-grained chunks** for key-value storage
  - Each CN allocates remote memory chunks and organizes them into **size classes** to minimize internal fragmentation
  - For each insert/write, the CN allocates memory from the smallest size class that fits the value

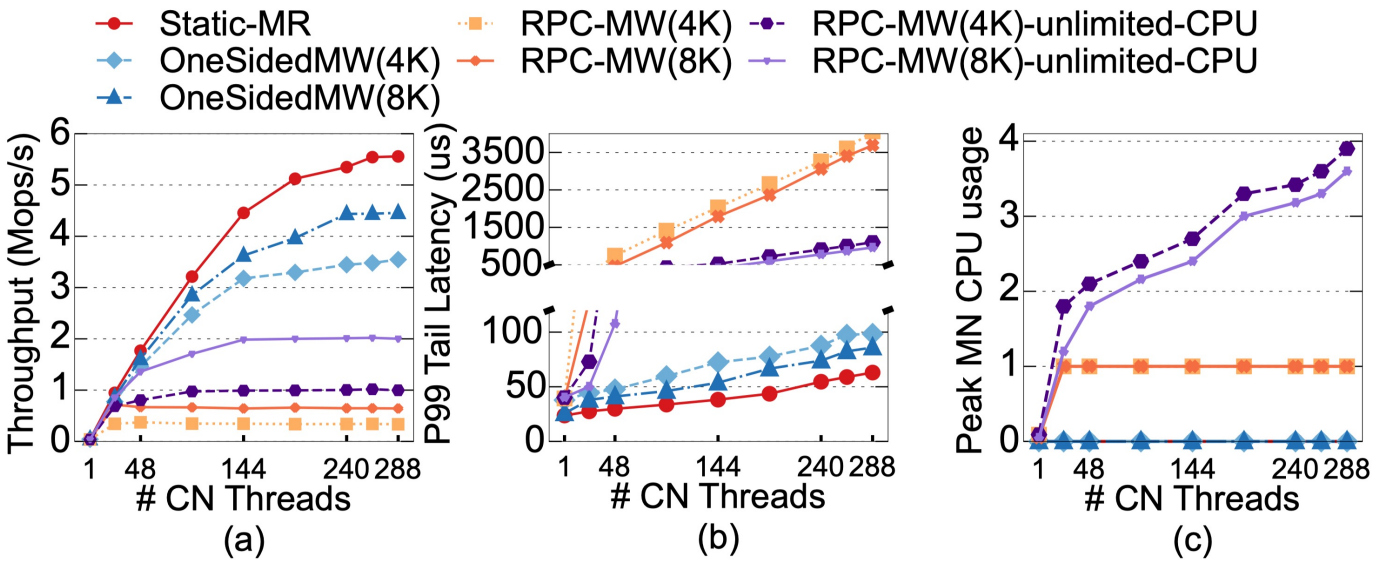
Results – Disaggregated KVS

- Insert-only workload, followed by randomly deleting 90% of the inserted key-value pairs.



Results – Disaggregated KVS

- Insert-only workload, followed by randomly deleting 90% of the inserted key-value pairs.



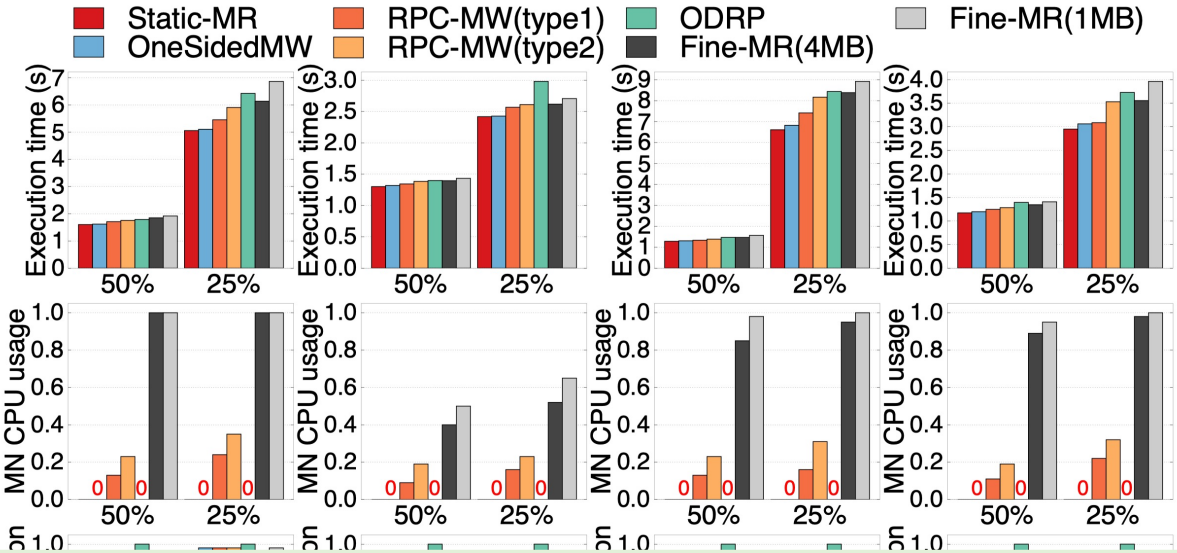
OneSidedMW (8KB allocation size) provides efficient memory management:
- achieves up to 10.6× better throughput than RPC-MW
- saves 43% remote memory with ≤19.8% overhead compared to Static-MR

Results – Kernel Swap

- Integration with Swap-based System (Fastswap)
  - MN partitions memory into **1 MB chunks**
  - Each CN divides its swap space into 1 MB segments and maps them to remote memory on demand

Results – Kernel Swap

- Run real-world cloud applications on 6 CNs.



≤ 3.8% overhead compared to Static-MR

No MN CPU usage

OneSidedMW

- achieves 1.5x to 2.4x higher memory utilization compared to Static-MR
- delivers a 19.7% performance improvement compared to RPC-MW
- delivers a 25.8% performance improvement compared to ODRP

Conclusion

- We leverage RNIC offloading and Memory Window to build an efficient **one-sided remote memory allocation** primitive for disaggregated memory architecture.
- OneSidedMW adopts type-2 MW for strong isolation and proposes two software techniques to solve the performance challenges introduced by type-2 MW.
- OneSidedMW achieves:
  - High memory utilization
  - High allocation & access performance
  - Zero MN CPU usage
  - Arbitrary I/O granularity
  - Strong isolation
- More details and security analysis in the paper.

Thanks!