Disaggregating Persistent Memory and Controlling Them Remotely: An Exploration of Passive Disaggregated Key-Value Stores

*Shin-Yeh Tsai, **Yizhou Shan**, Yiying Zhang*

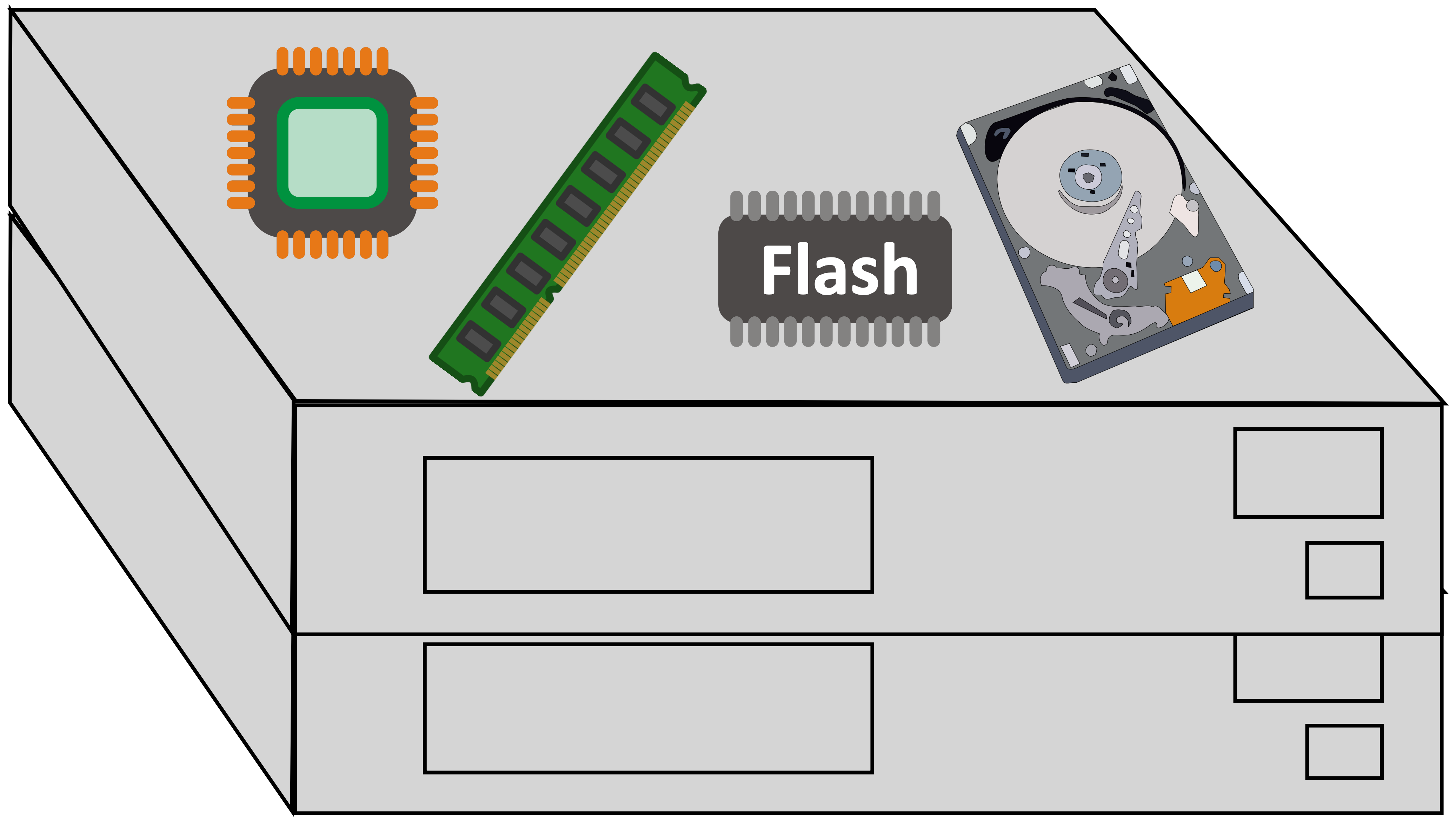
PURDUE UNIVERSITY

WukLab

UC San Diego

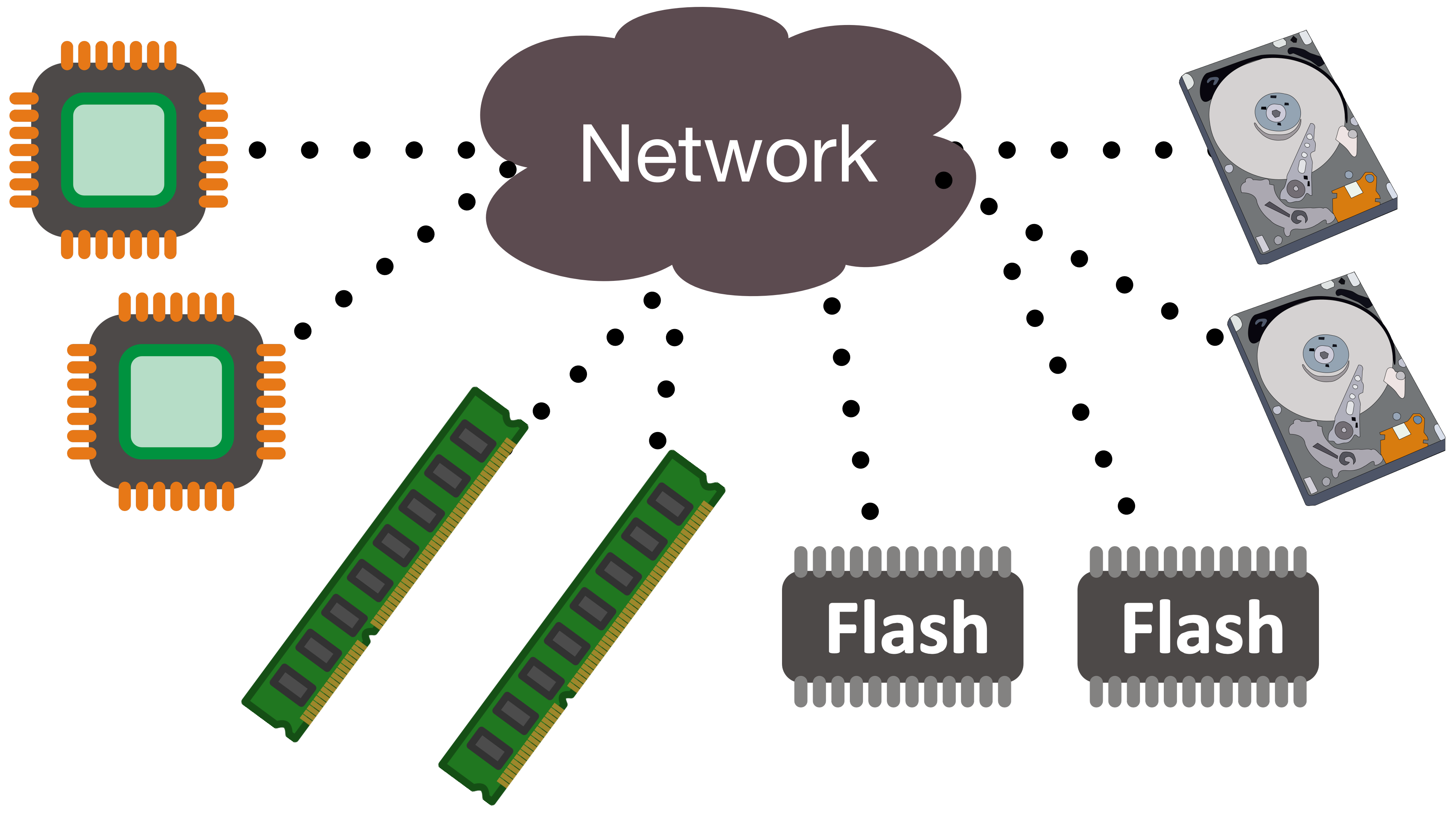
Resource Disaggregation

Break monolithic servers into ***network-attached*** resource pools



Resource Disaggregation

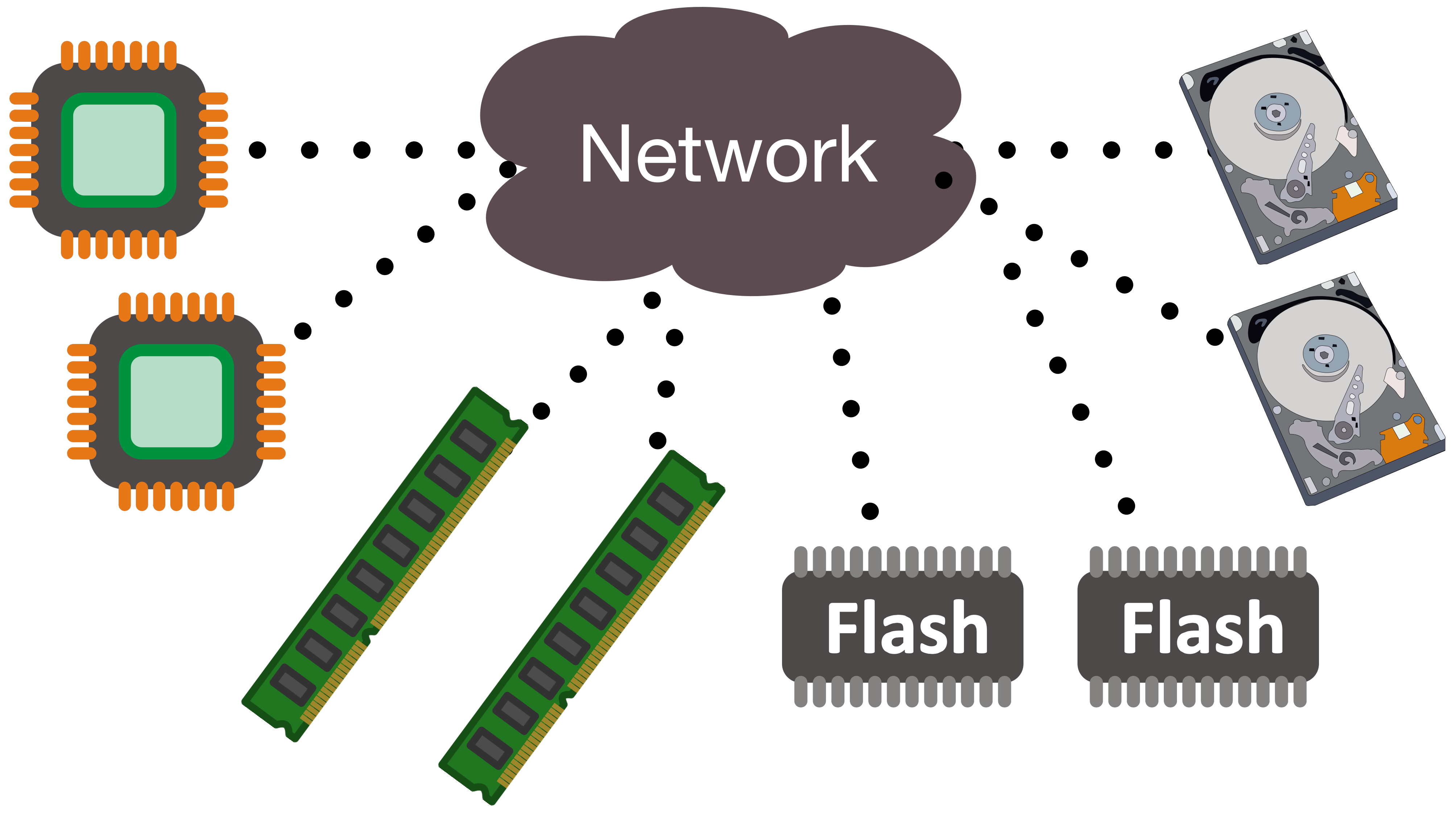
Break monolithic servers into ***network-attached*** resource pools



Resource Disaggregation

Break monolithic servers into ***network-attached*** resource pools

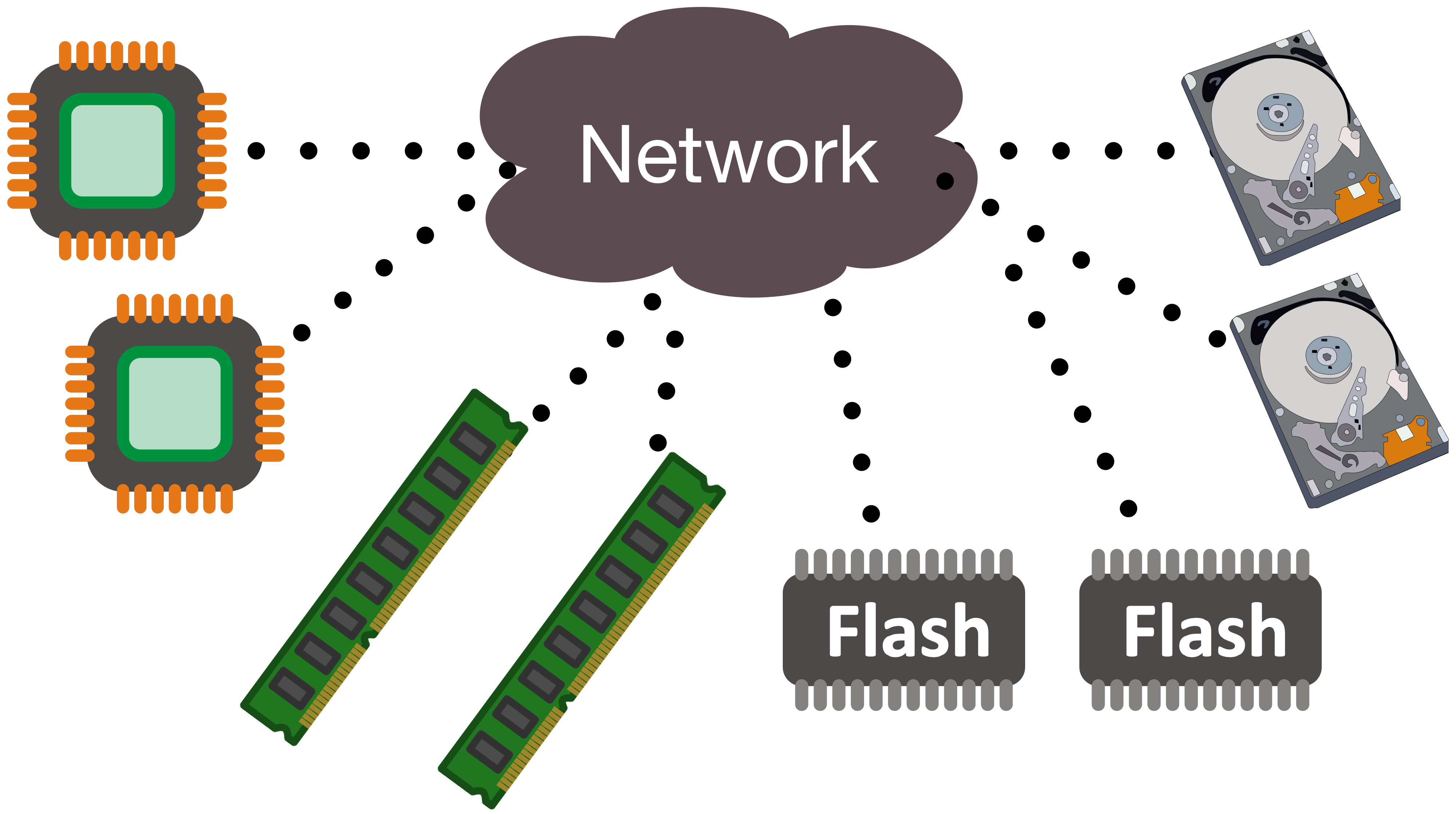
Better manageability, independent scaling, tight resource packing



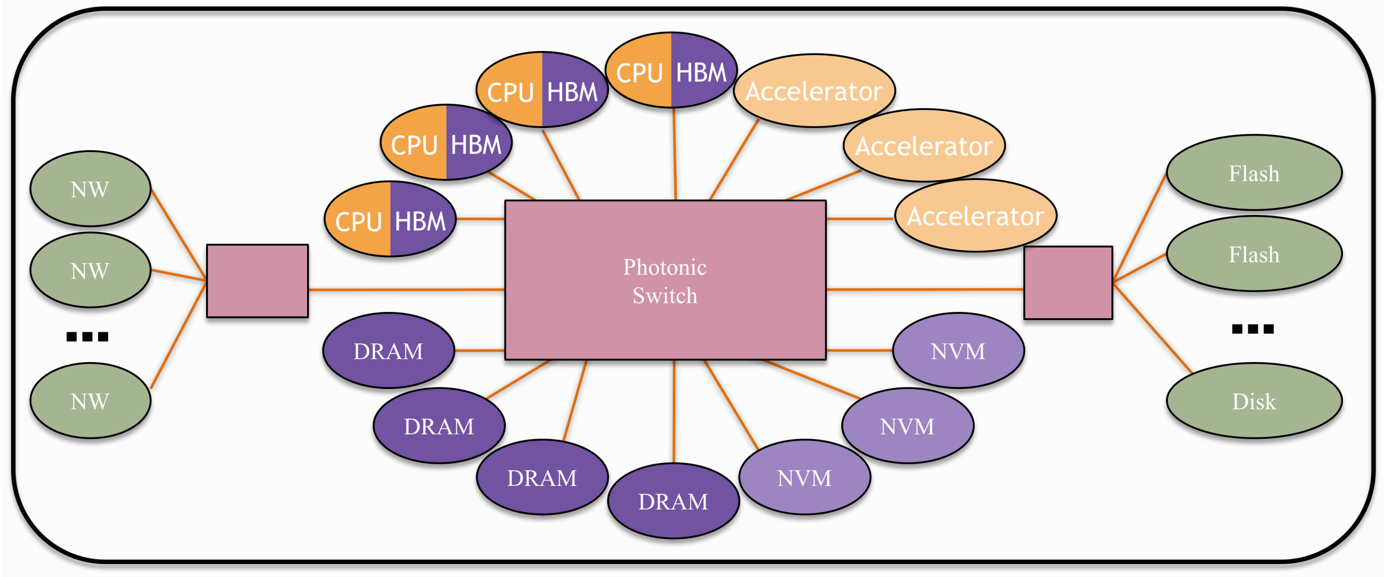
Resource Disaggregation

Break monolithic servers into ***network-attached*** resource pools

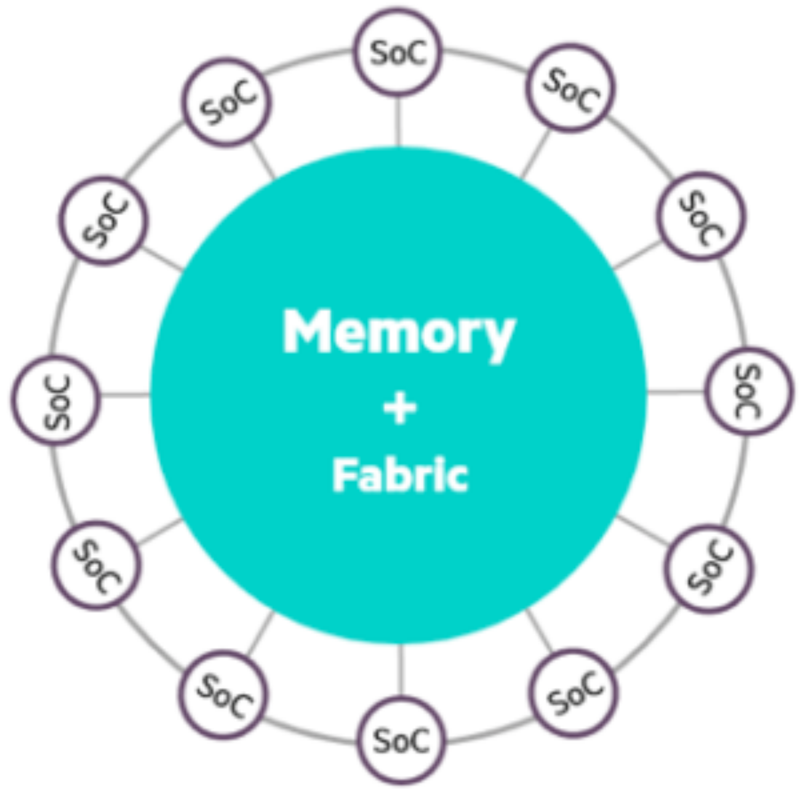
Better manageability, independent scaling, tight resource packing



LegoOS



Berkeley Firebox



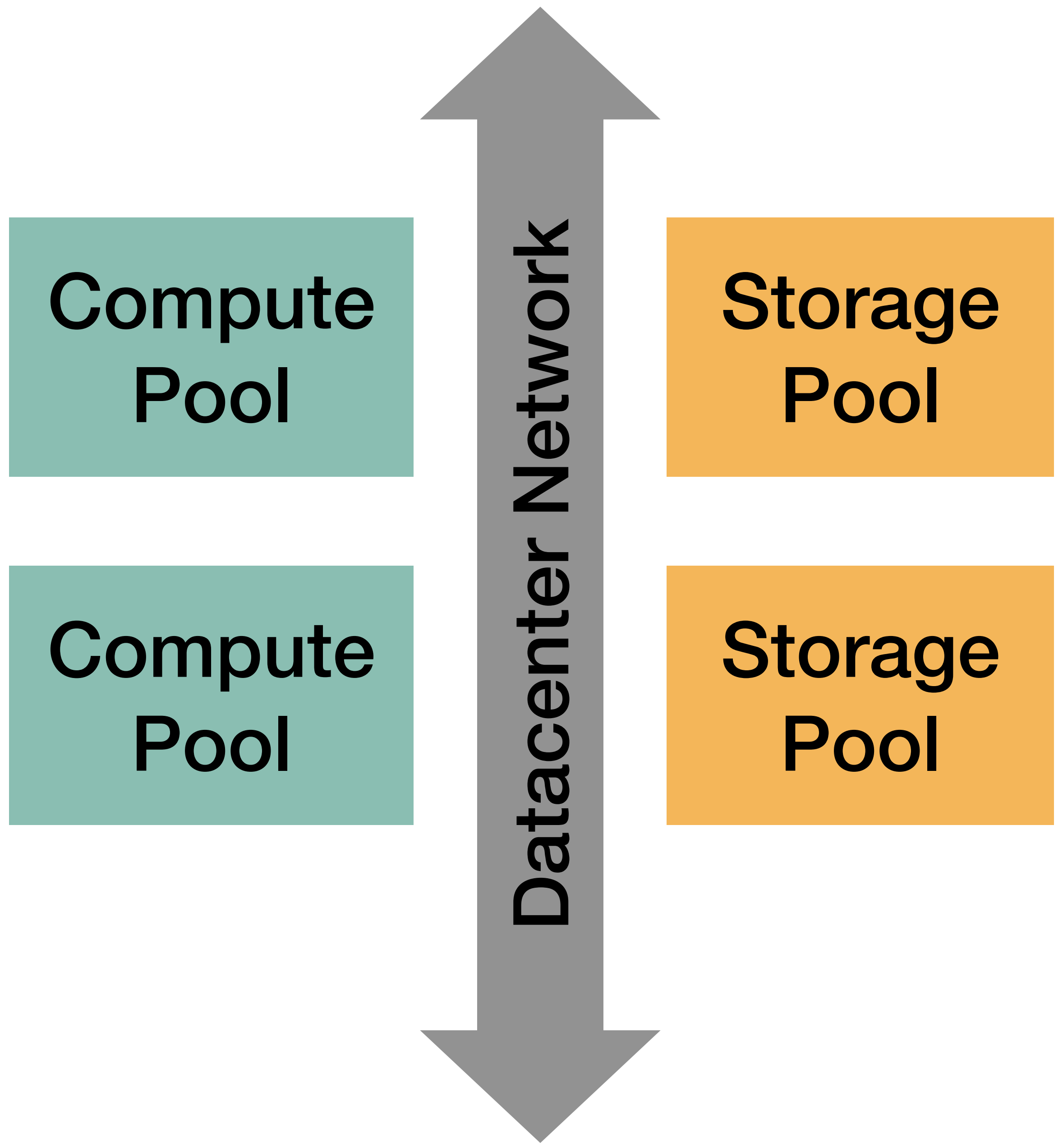
Hewlett Packard Enterprise

Disaggregated Storage

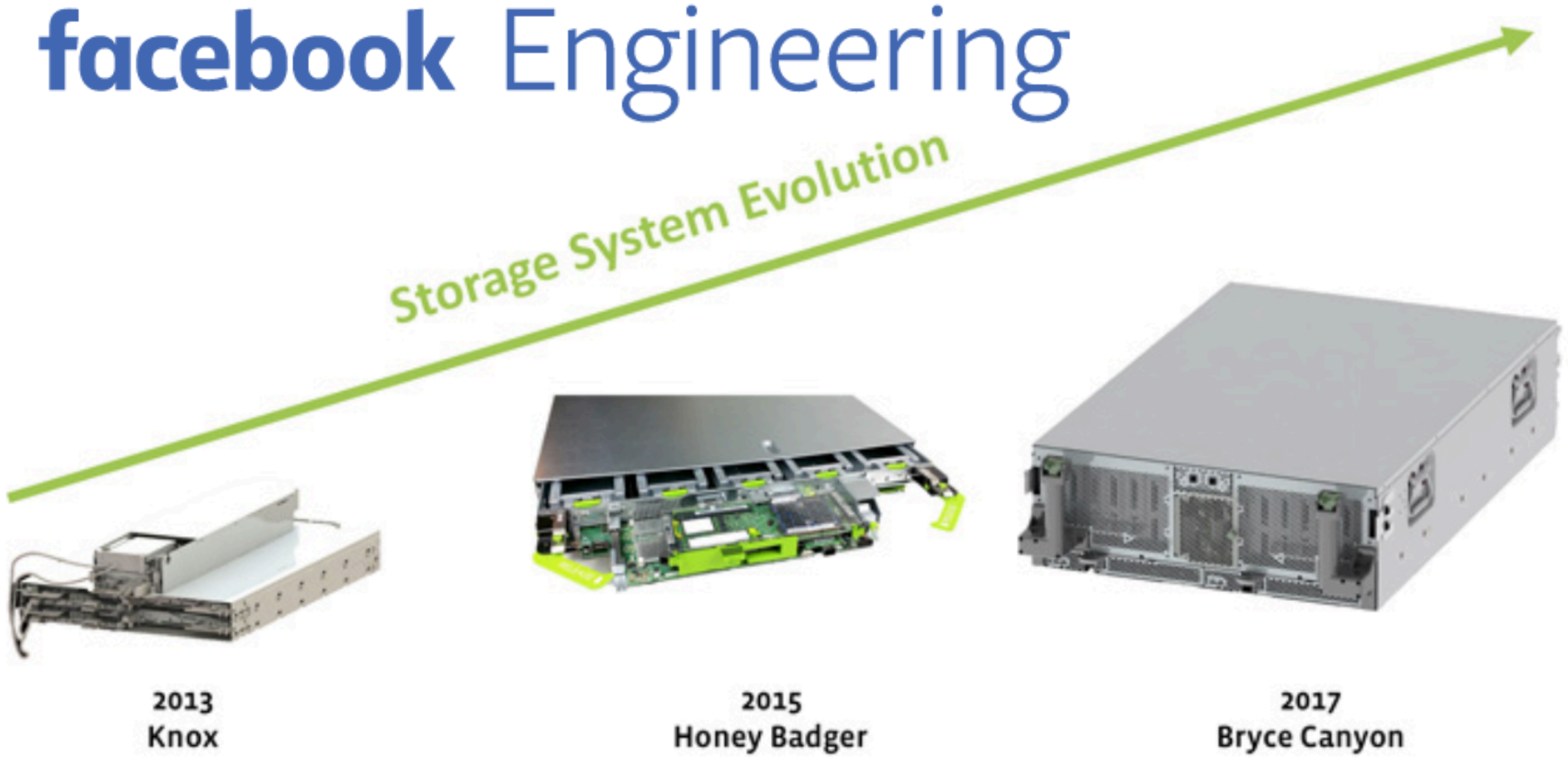
Separate compute and storage pools

- Manage and scale independently

A common practice in datacenters and clouds



facebook Engineering



Alibaba Singles' Day 2019 had a Record Peak Order Rate of 544,000 per Second

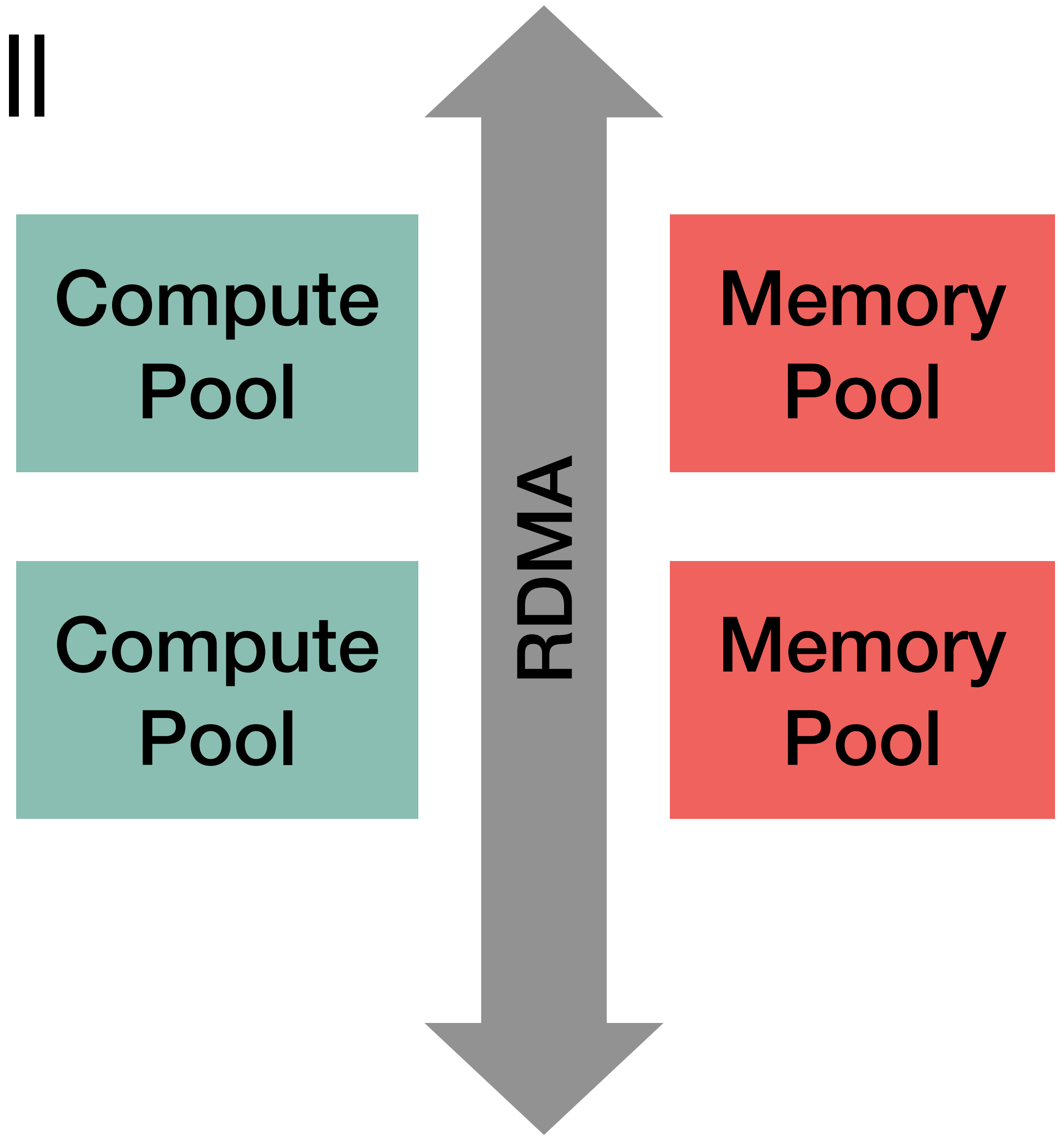
Yash Wate

Disaggregated Memory

- Network is getting faster (e.g., 200 Gbps, sub-600 ns)
- Application need for large memory + memory-capacity wall

➡ Remote/disaggregated memory

- Applications access (large) non-local memory

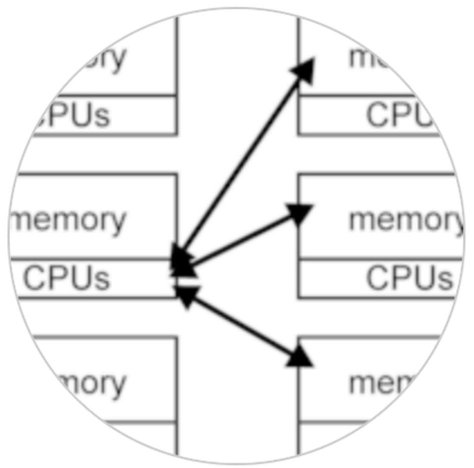


Microsoft

FaRM

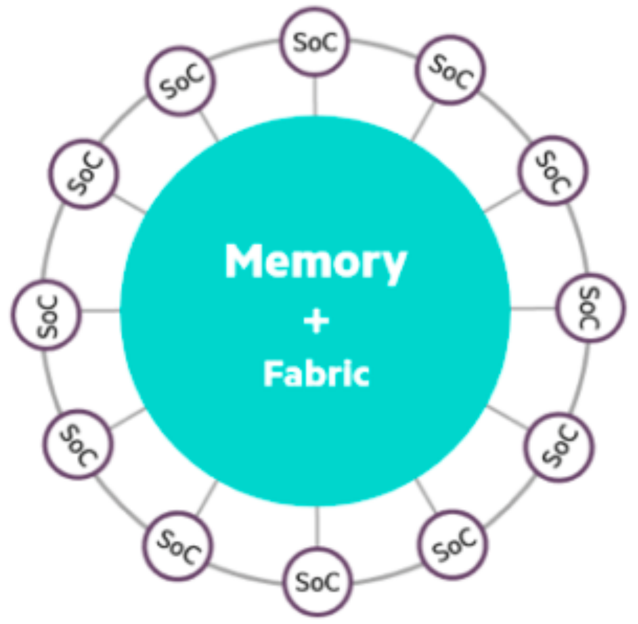
vmware

Remote memory



HUAWEI

HTC-DC



Hewlett Packard Enterprise

Memory Blades, ISCA'09
NAM-DB, VLDB'17
ZombieLand, EuroSys'18
StRoM, EuroSys'20

Disaggregated Persistent Memory?

Disaggregated Persistent Memory?

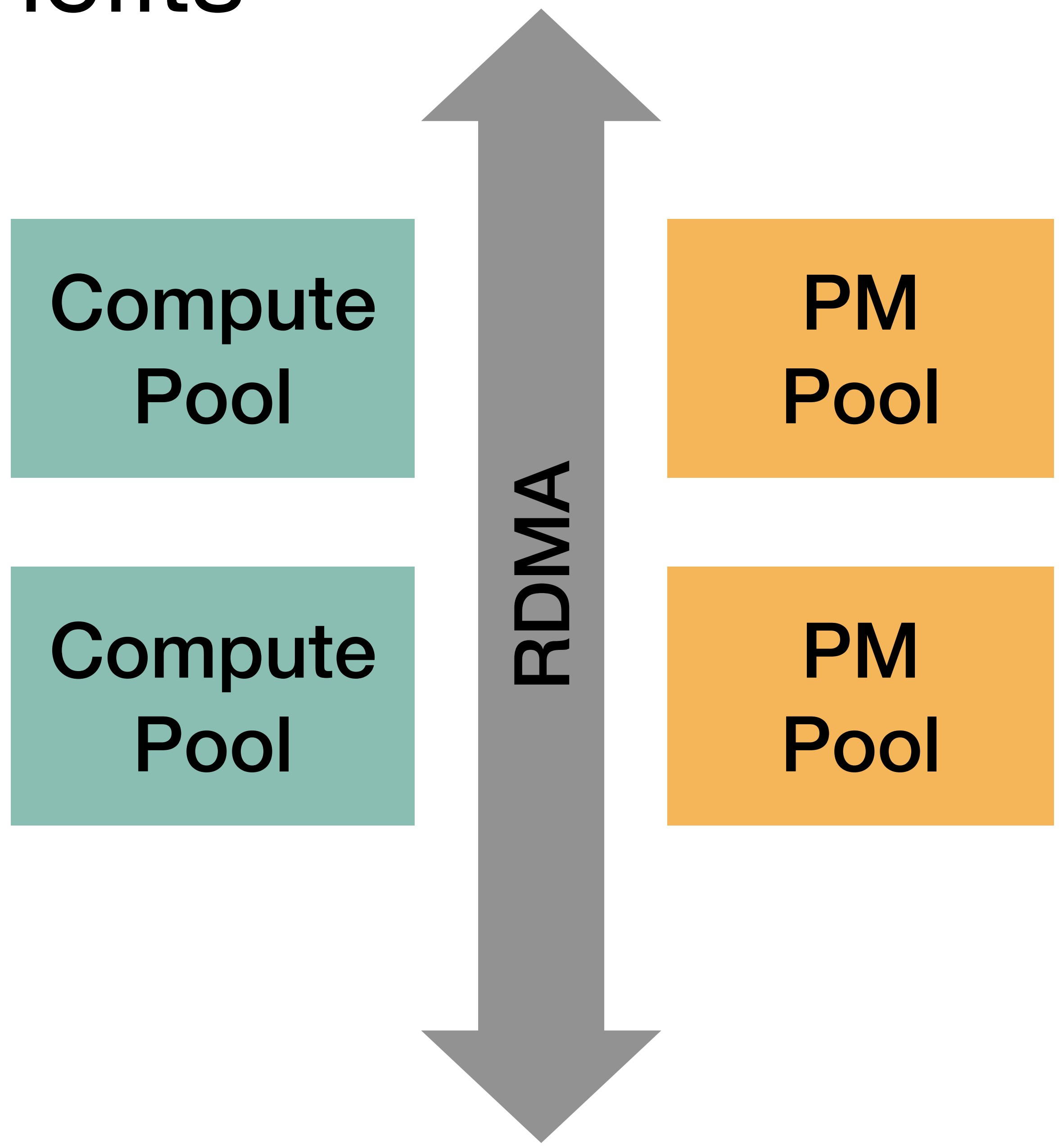
PM: byte-addressable, persistent, memory-like perf

Disaggregated Persistent Memory?

PM: byte-addressable, persistent, memory-like perf

Disaggregating PM (DPM)

- Enjoy disaggregation's management, scalability, utilization benefits
- Easy way to integrate PM into current datacenters



Disaggregated Persistent Memory?

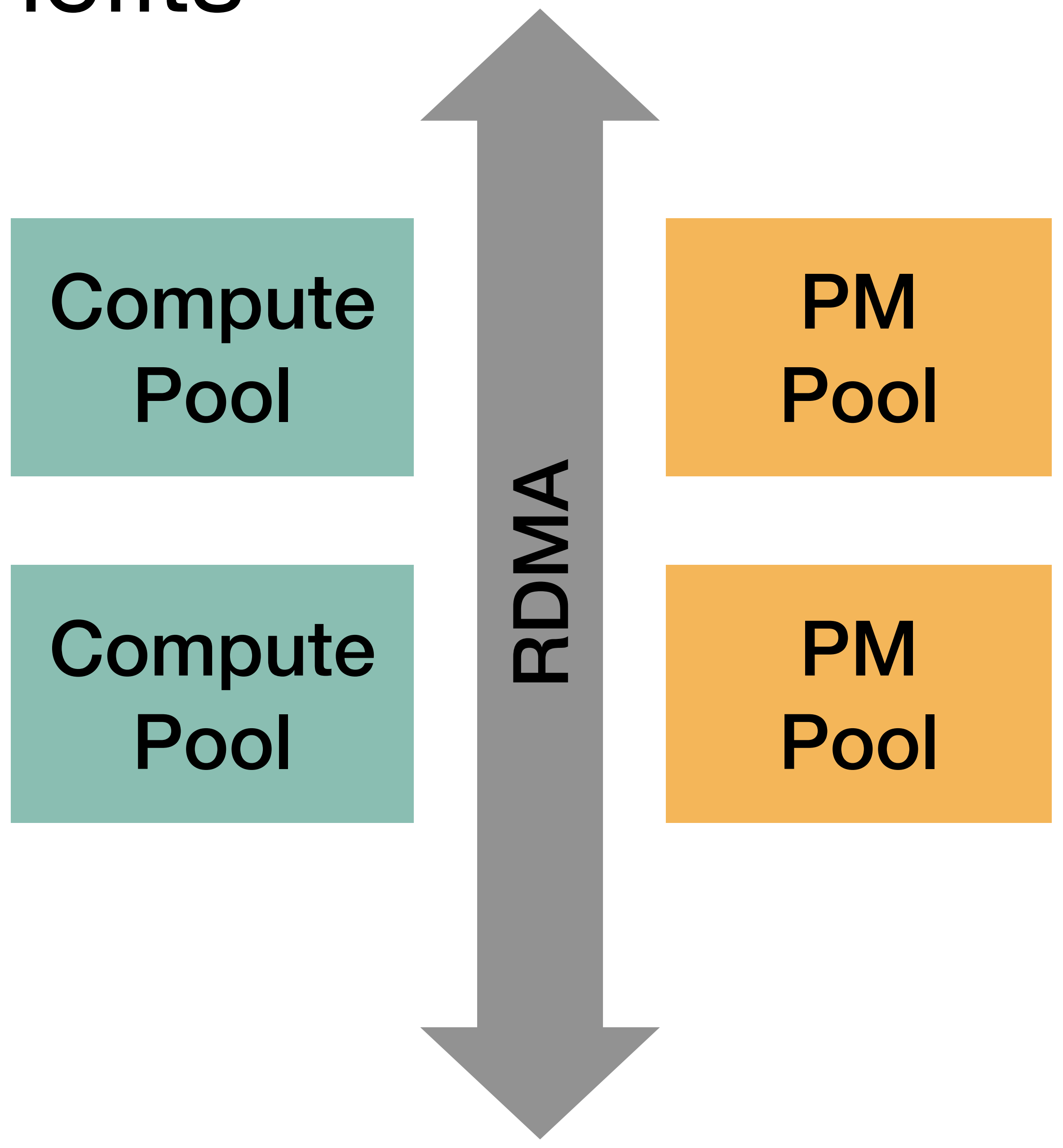
PM: byte-addressable, persistent, memory-like perf

Disaggregating PM (DPM)

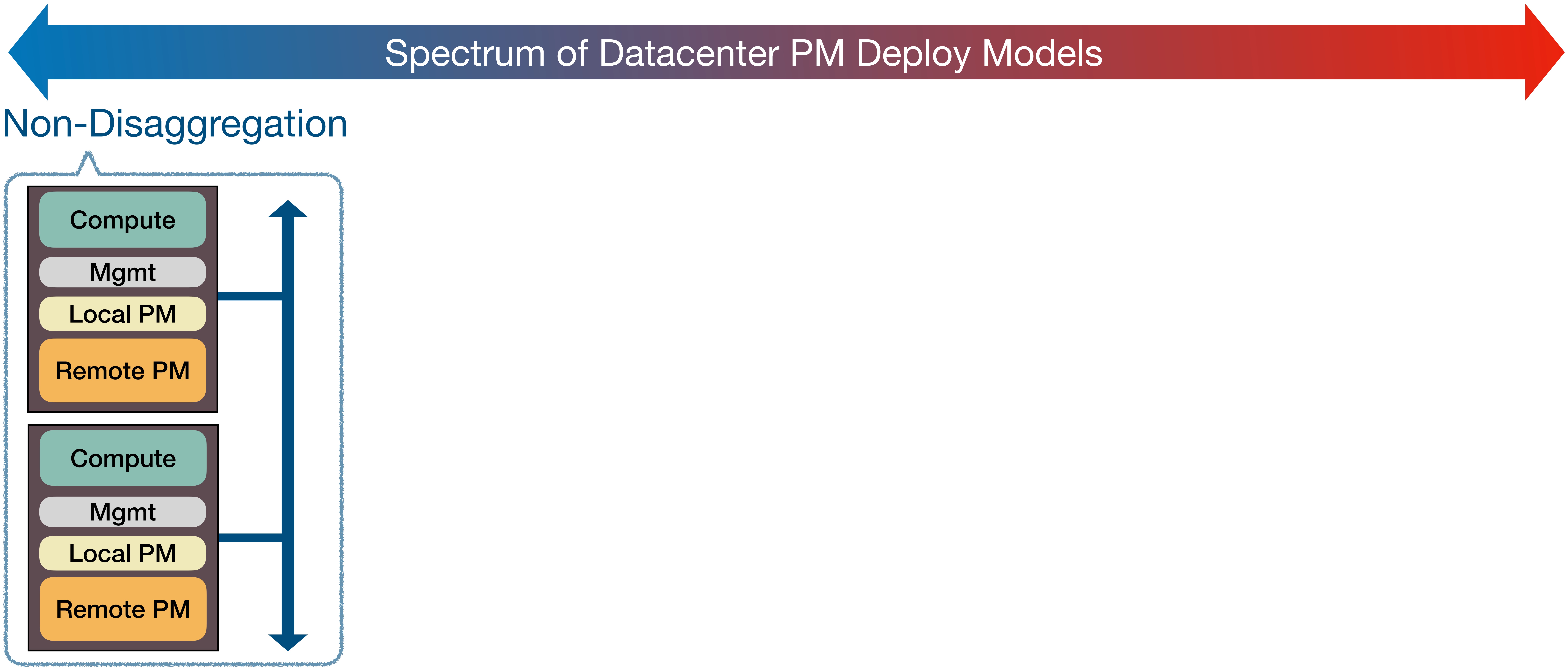
- Enjoy disaggregation's management, scalability, utilization benefits
- Easy way to integrate PM into current datacenters

Use existing disaggregated systems for DPM?

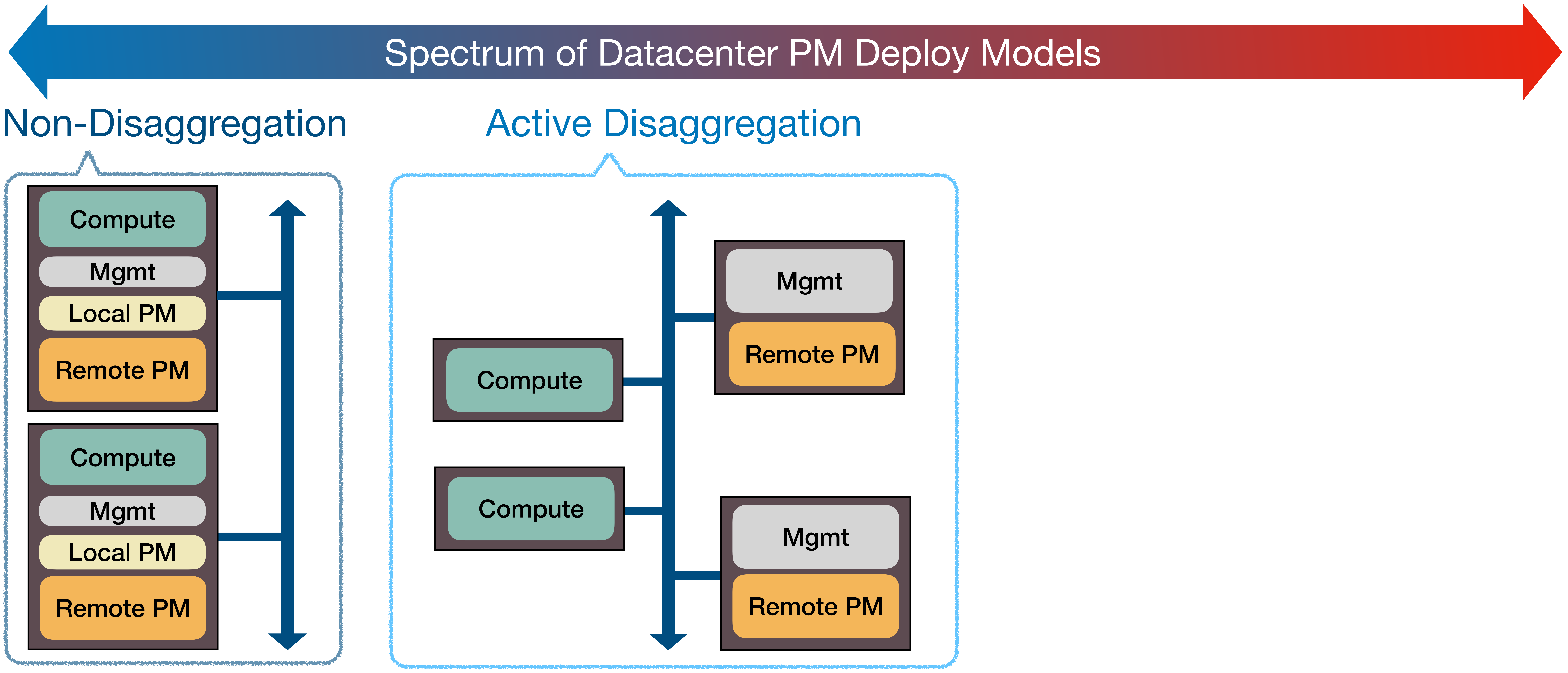
- Disaggregated storage: software stack too slow for fast PM
- Disaggregated memory: do not provide data reliability



Spectrum of Datacenter PM Deploy Models



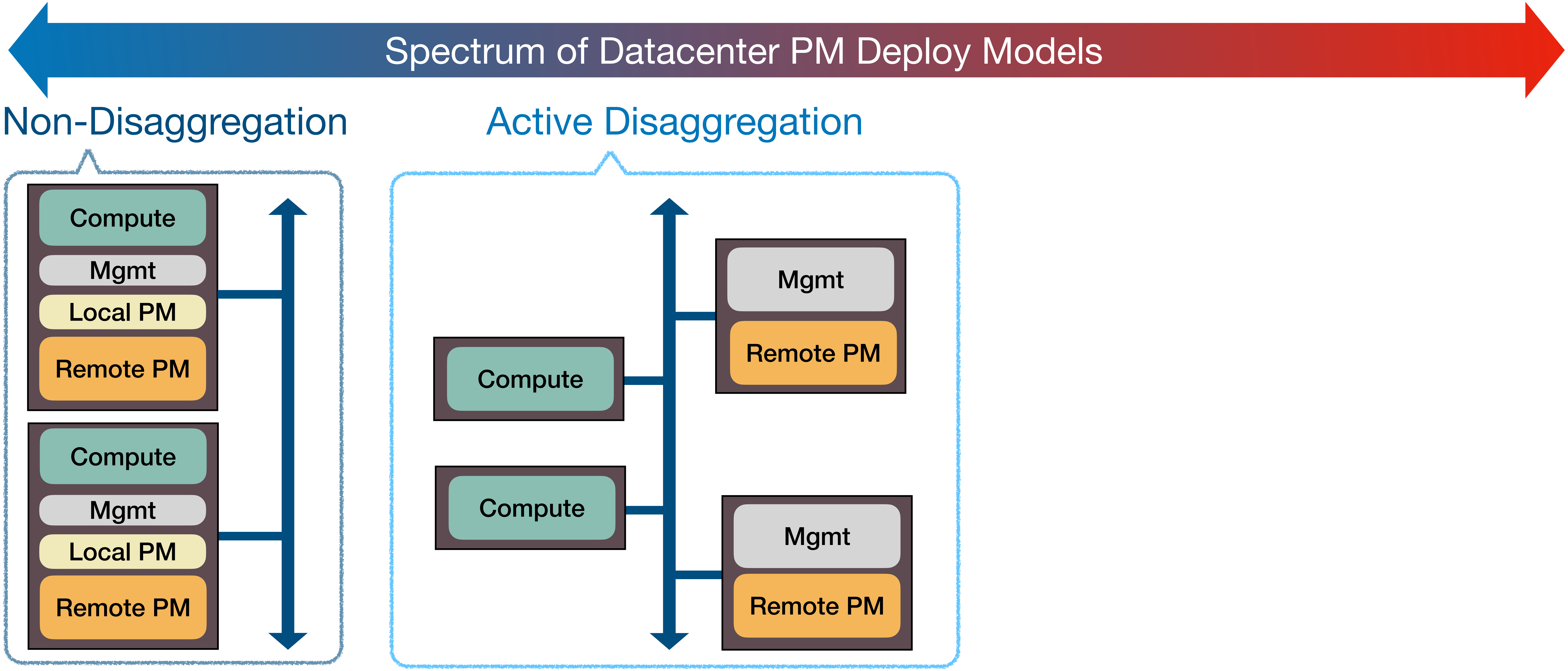
Hotpot, SoCC’17
Octopus, ATC’17
Remote Regions, ATC’18



Hotpot, SoCC'17
Octopus, ATC'17
Remote Regions, ATC'18

HERD, SIGCOMM'14
Decibel, NSDI'17
HyperLoop, SIGCOMM'18
Snowflake, NSDI'20

Traditional Storage Systems

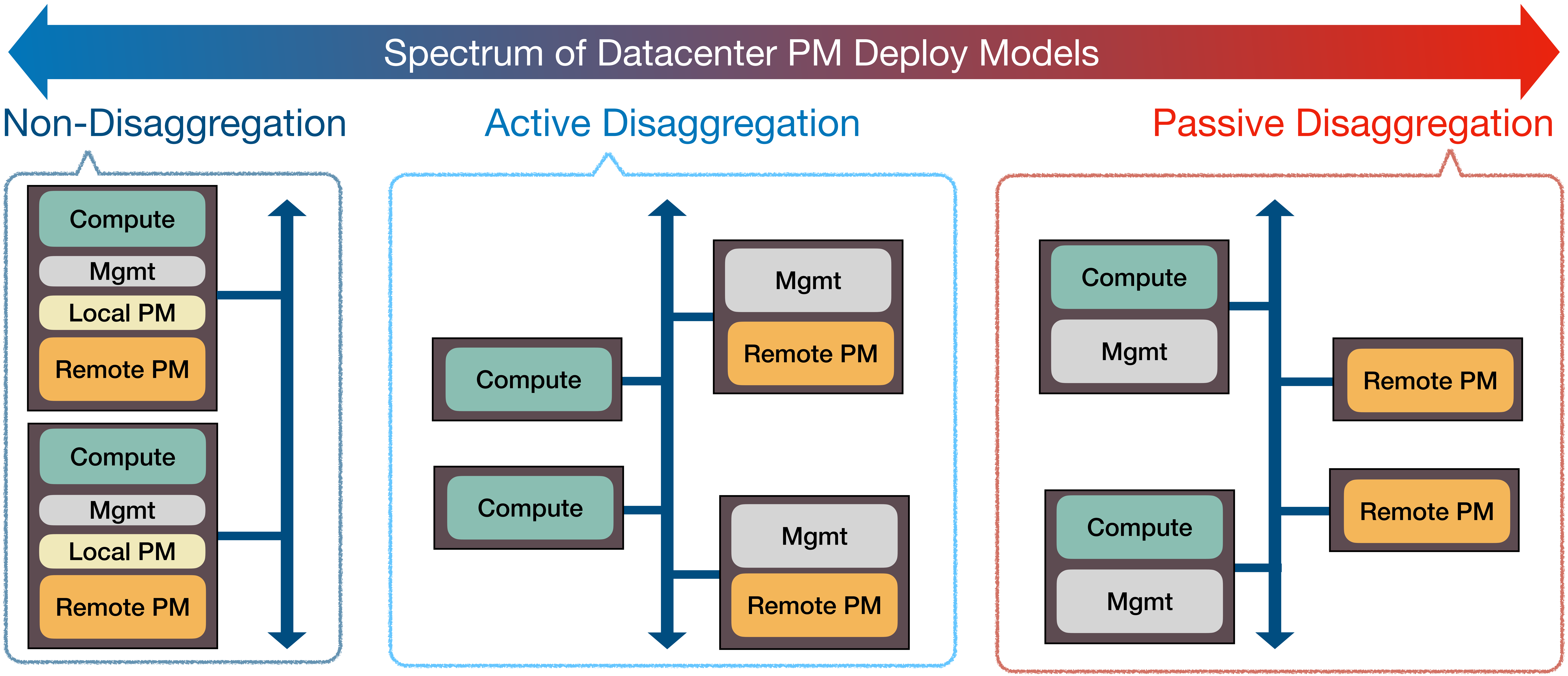


Hotpot, SoCC’17
Octopus, ATC’17
Remote Regions, ATC’18

HERD, SIGCOMM’14
Decibel, NSDI’17
HyperLoop, SIGCOMM’18
Snowflake, NSDI’20

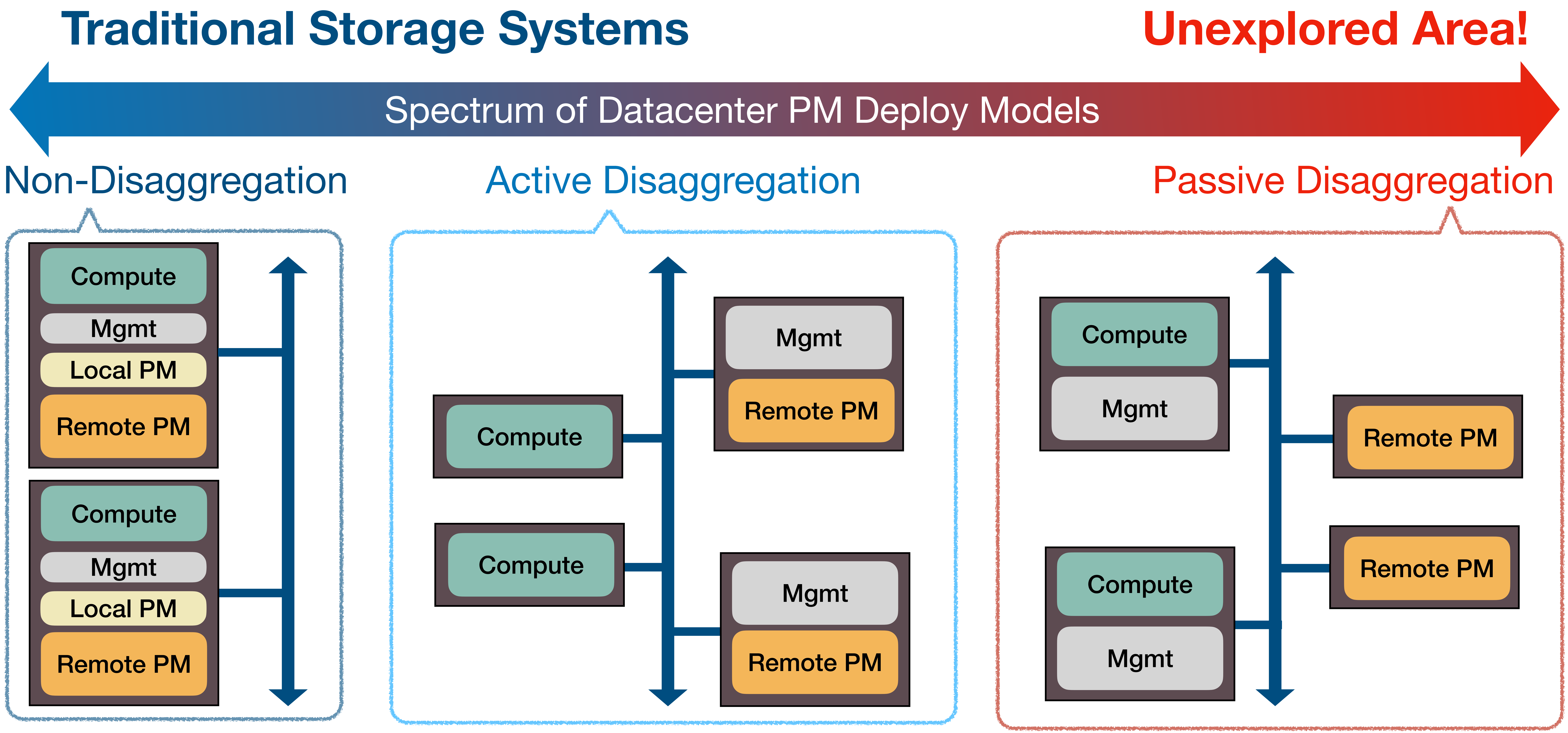
Traditional Storage Systems

Unexplored Area!

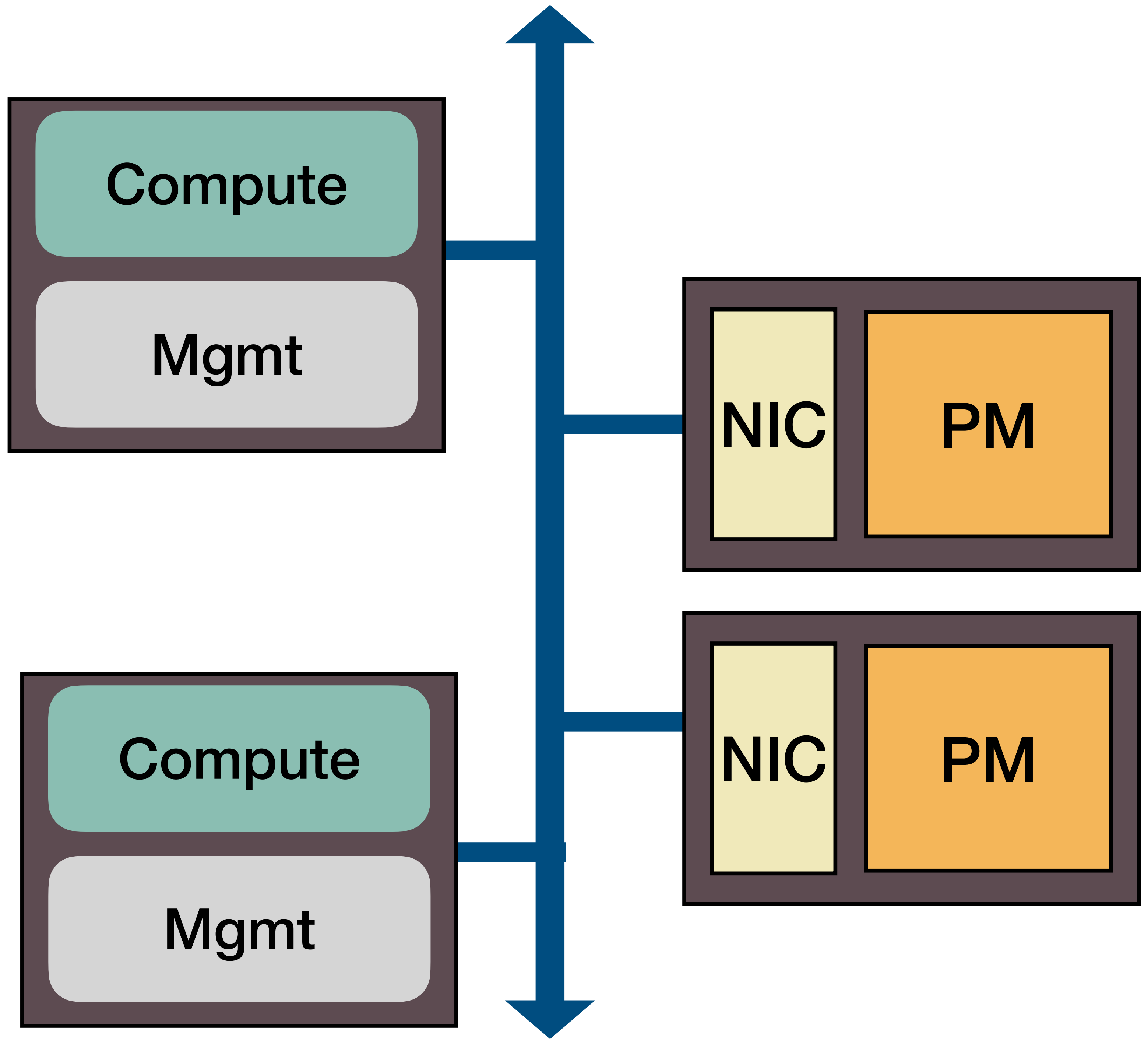


Hotpot, SoCC'17
Octopus, ATC'17
Remote Regions, ATC'18

HERD, SIGCOMM'14
Decibel, NSDI'17
HyperLoop, SIGCOMM'18
Snowflake, NSDI'20


Traditional Storage Systems
Unexplored Area!
Spectrum of Datacenter PM Deploy Models
Non-Disaggregation
Active Disaggregation
Passive Disaggregation
Compute
Mgmt
Local PM
Remote PM
Compute
Mgmt
Local PM
Remote PM
Compute
Compute
Mgmt
Remote PM
Mgmt
Remote PM
Compute
Mgmt
Remote PM
Remote PM
Compute
Mgmt

Passive Disaggregated PM (pDPM)



pDPM

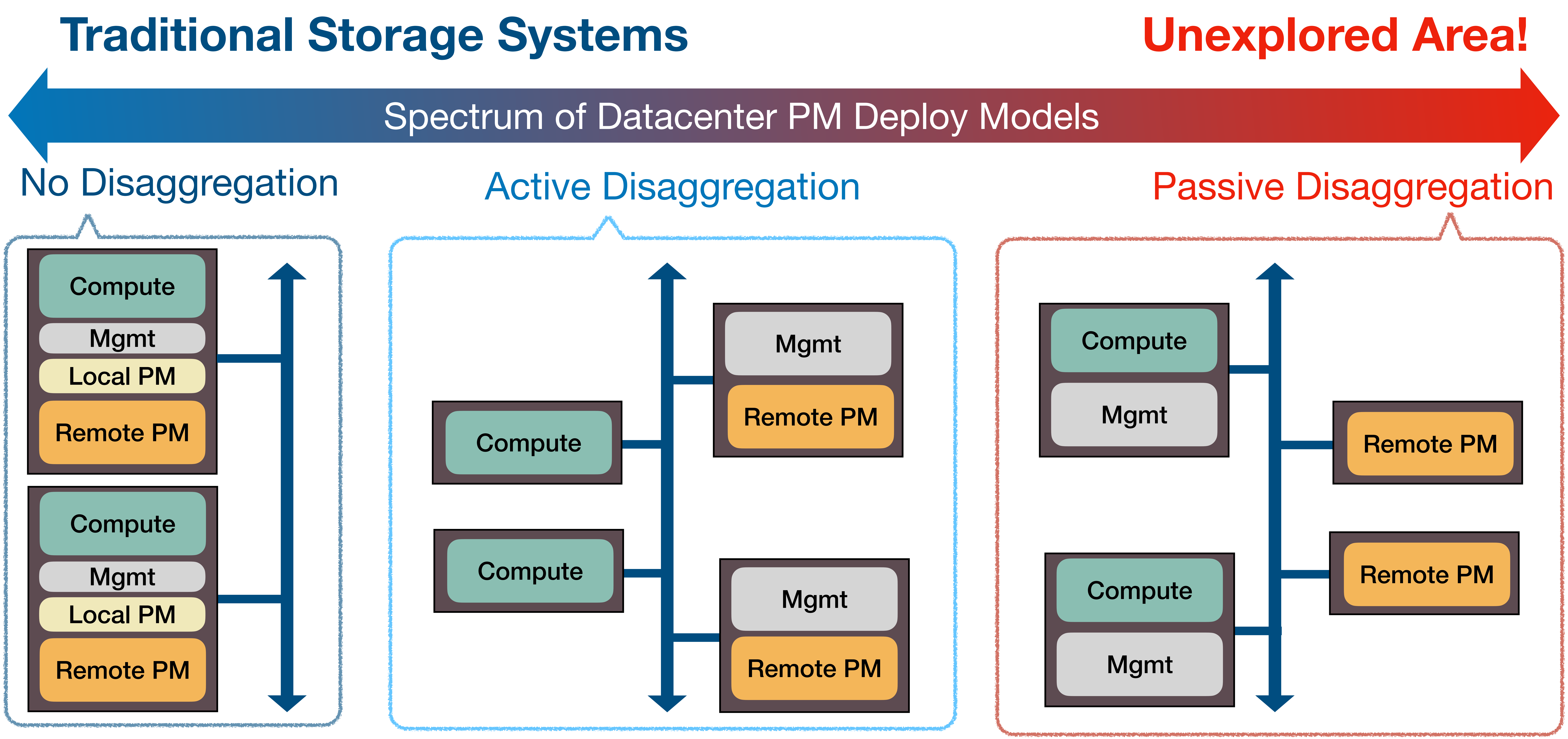
- Passive PM devices with NIC and PM
- Accessible only via network

Why pDPM?

- Low CapEx and OpEx
- Easy to add, remove, and change
- No scalability bottleneck at storage nodes
- Research value in exploring new design area

Why possible now? Fast RDMA network + CPU bypassing

Without processing power at PM,
where to process and manage data?


Traditional Storage Systems
Unexplored Area!
Spectrum of Datacenter PM Deploy Models
No Disaggregation
Active Disaggregation
Passive Disaggregation
Compute
Mgmt
Local PM
Remote PM
Compute
Mgmt
Local PM
Remote PM
Mgmt
Remote PM
Compute
Compute
Mgmt
Remote PM
Compute
Mgmt
Remote PM
Remote PM
Compute
Mgmt

Spectrum of Datacenter PM Deploy Models

Non Disaggregation

Active Disaggregation

Passive Disaggregation

Spectrum of Datacenter PM Deploy Models

Non Disaggregation

Active Disaggregation

Passive Disaggregation

Where to process and manage data?

Spectrum of Datacenter PM Deploy Models

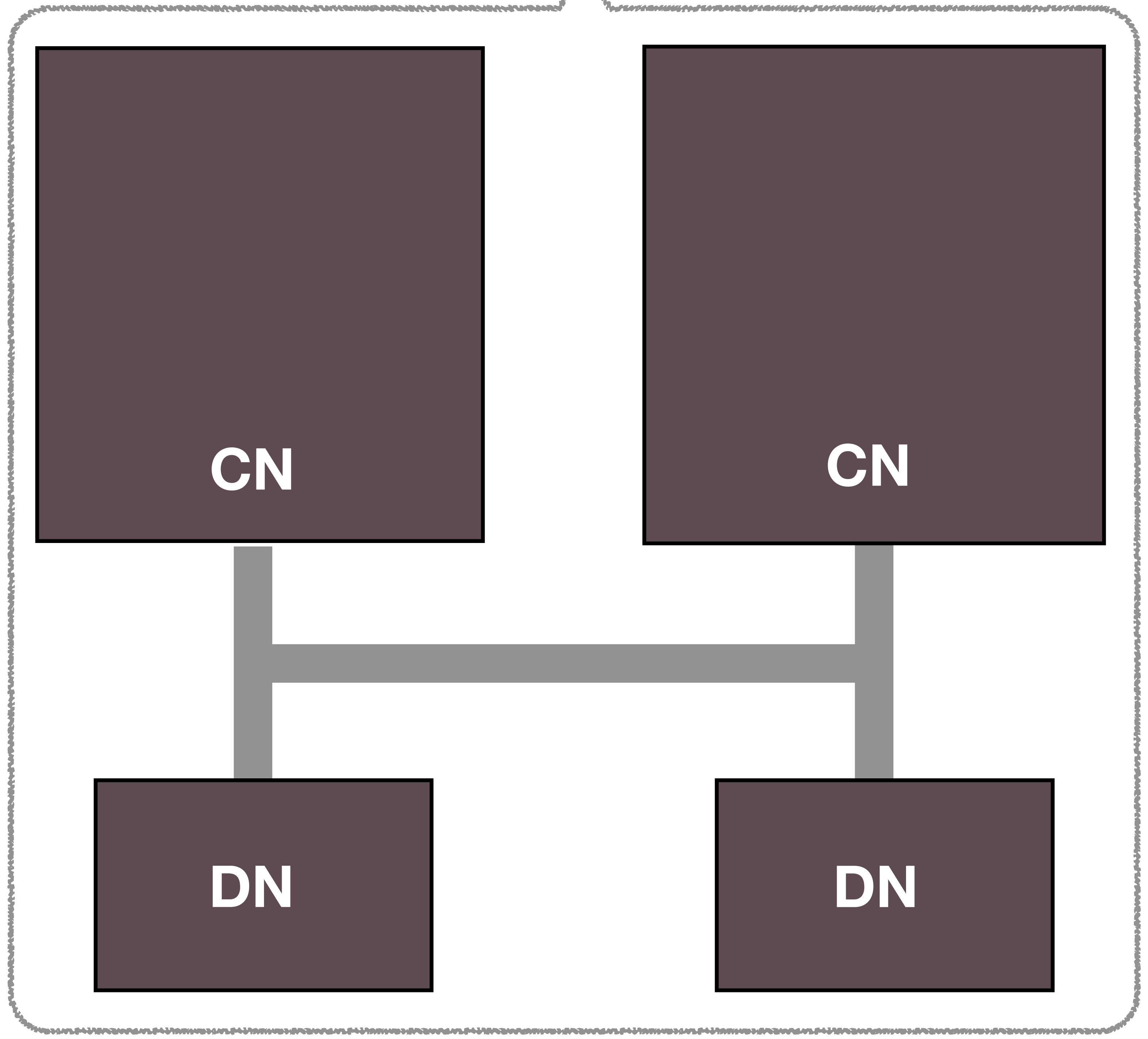
Non Disaggregation

Active Disaggregation

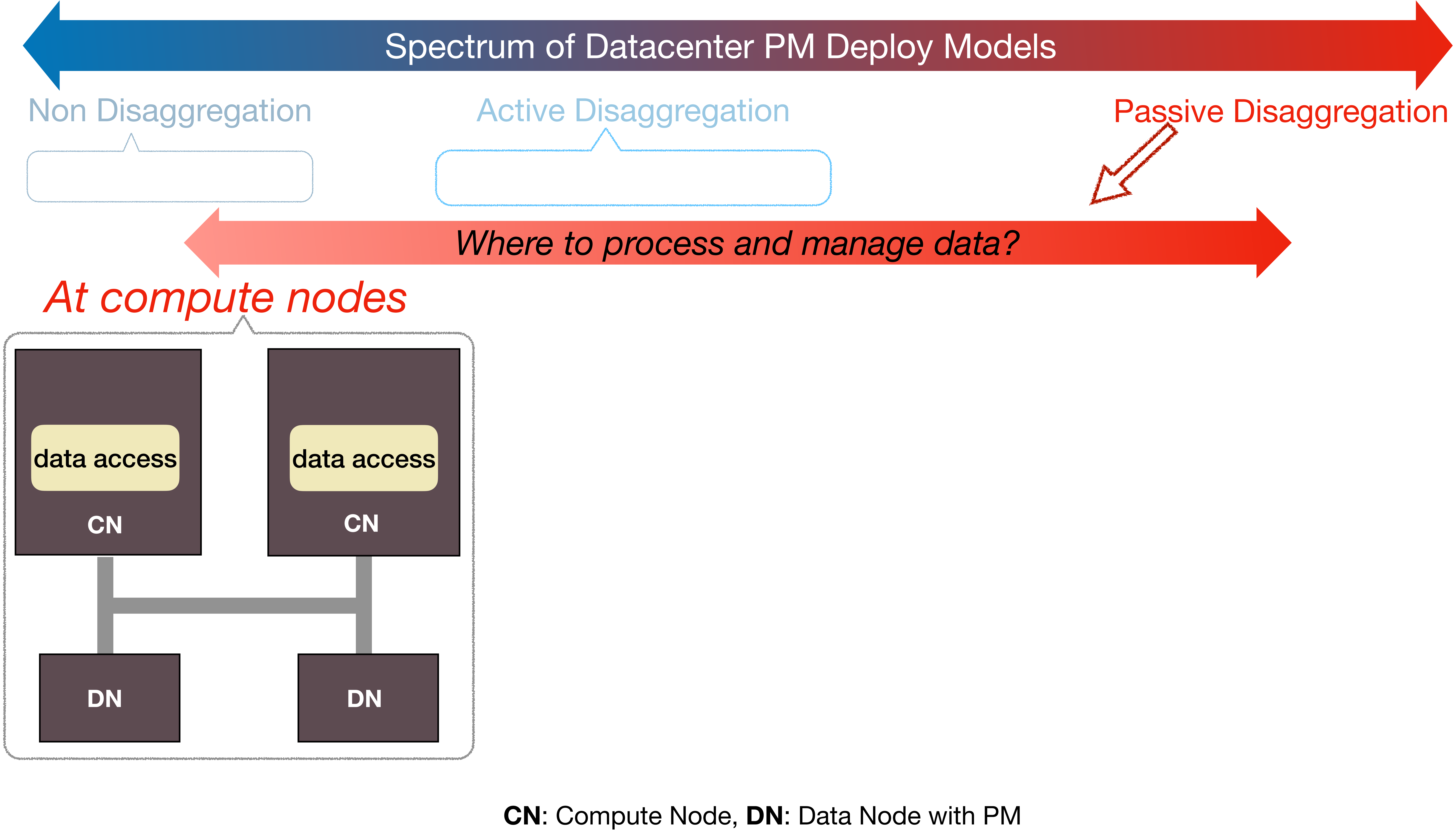
Passive Disaggregation

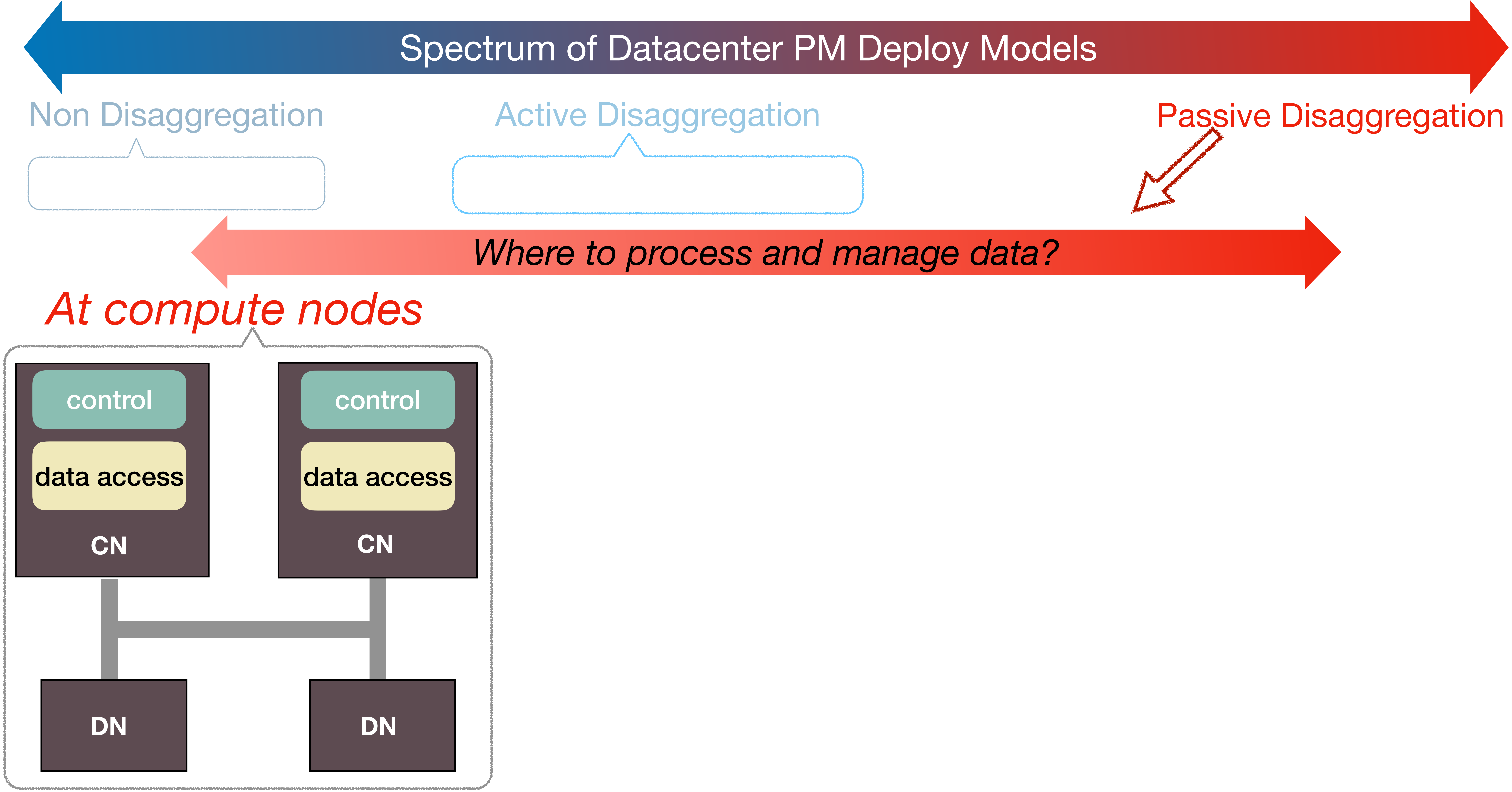
Where to process and manage data?

At compute nodes



CN: Compute Node, **DN**: Data Node with PM


Spectrum of Datacenter PM Deploy Models
Non Disaggregation
Active Disaggregation
Passive Disaggregation
Where to process and manage data?
At compute nodes
data access
CN
data access
CN
DN
DN
CN: Compute Node, DN: Data Node with PM



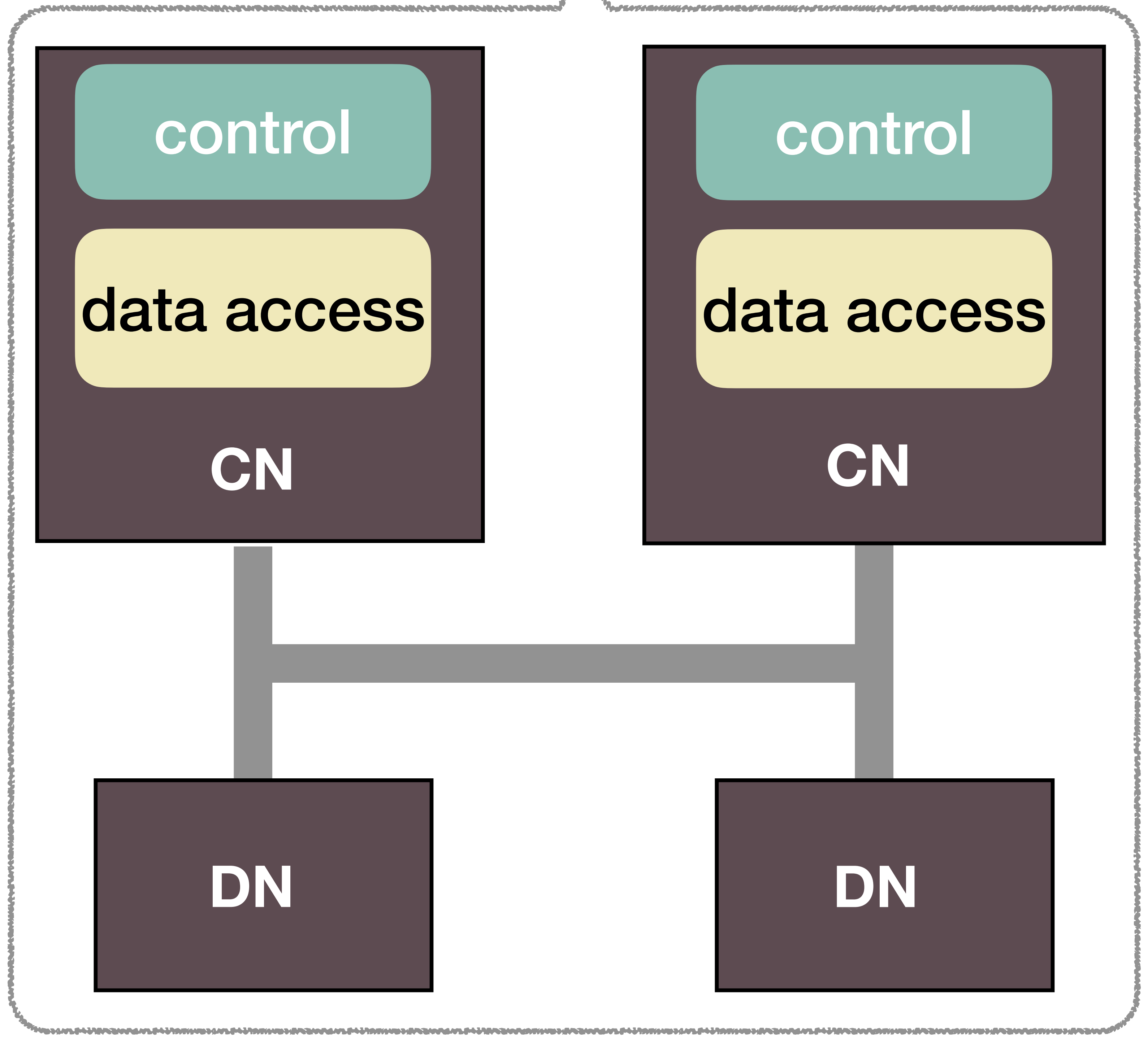
CN: Compute Node, **DN**: Data Node with PM

Spectrum of Datacenter PM Deploy Models

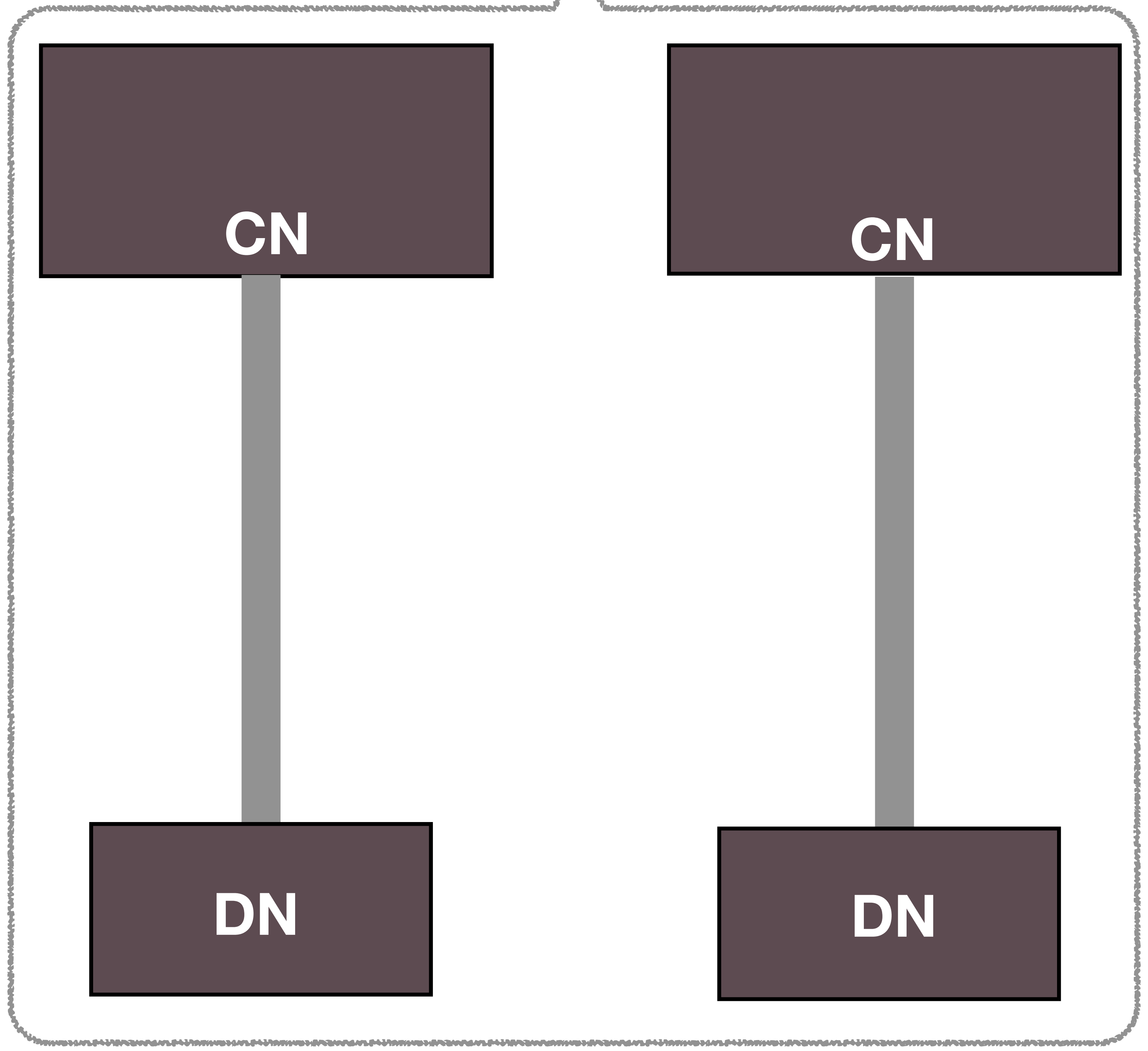
Non Disaggregation | Active Disaggregation | Passive Disaggregation

Where to process and manage data?

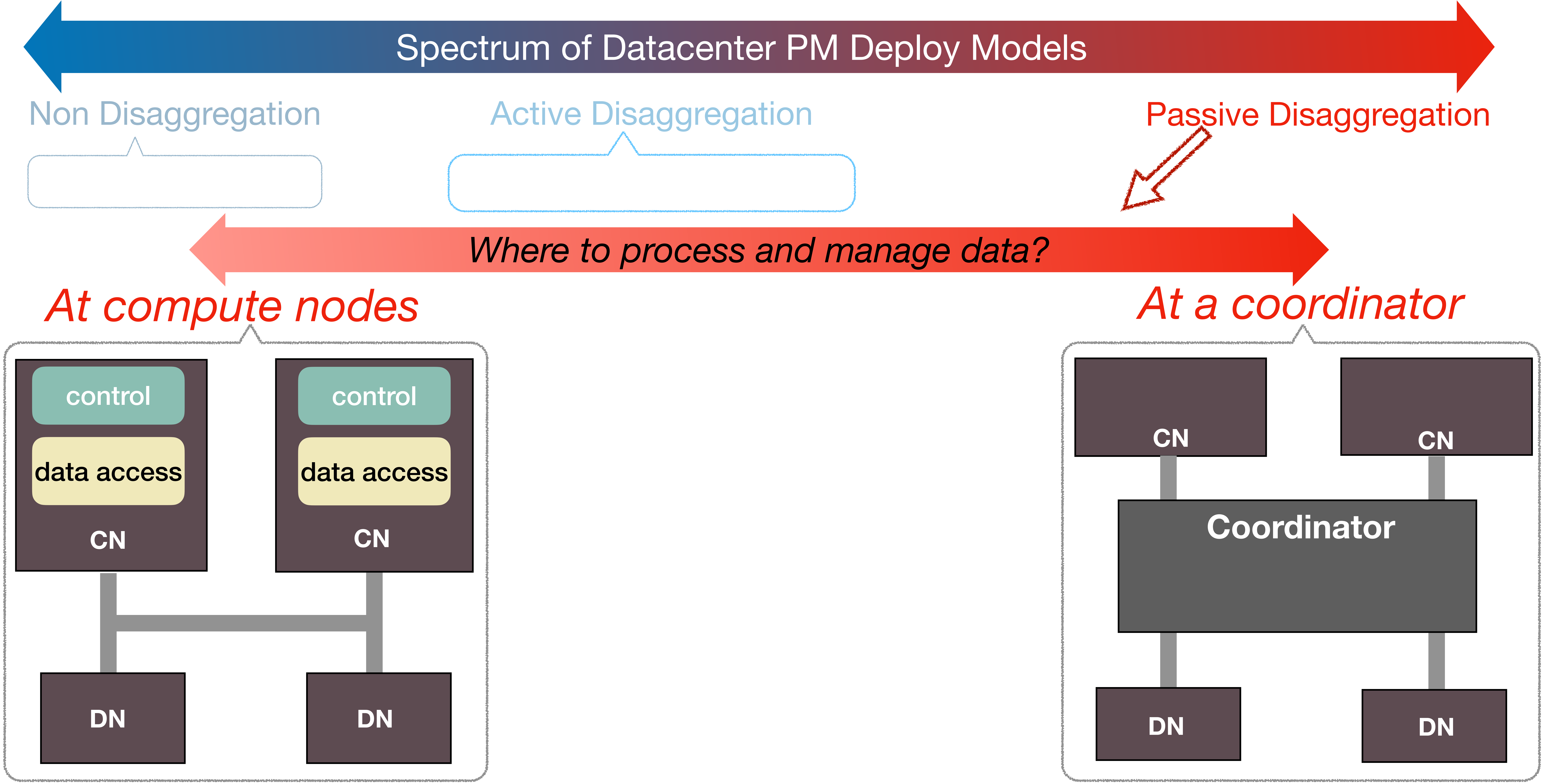
At compute nodes



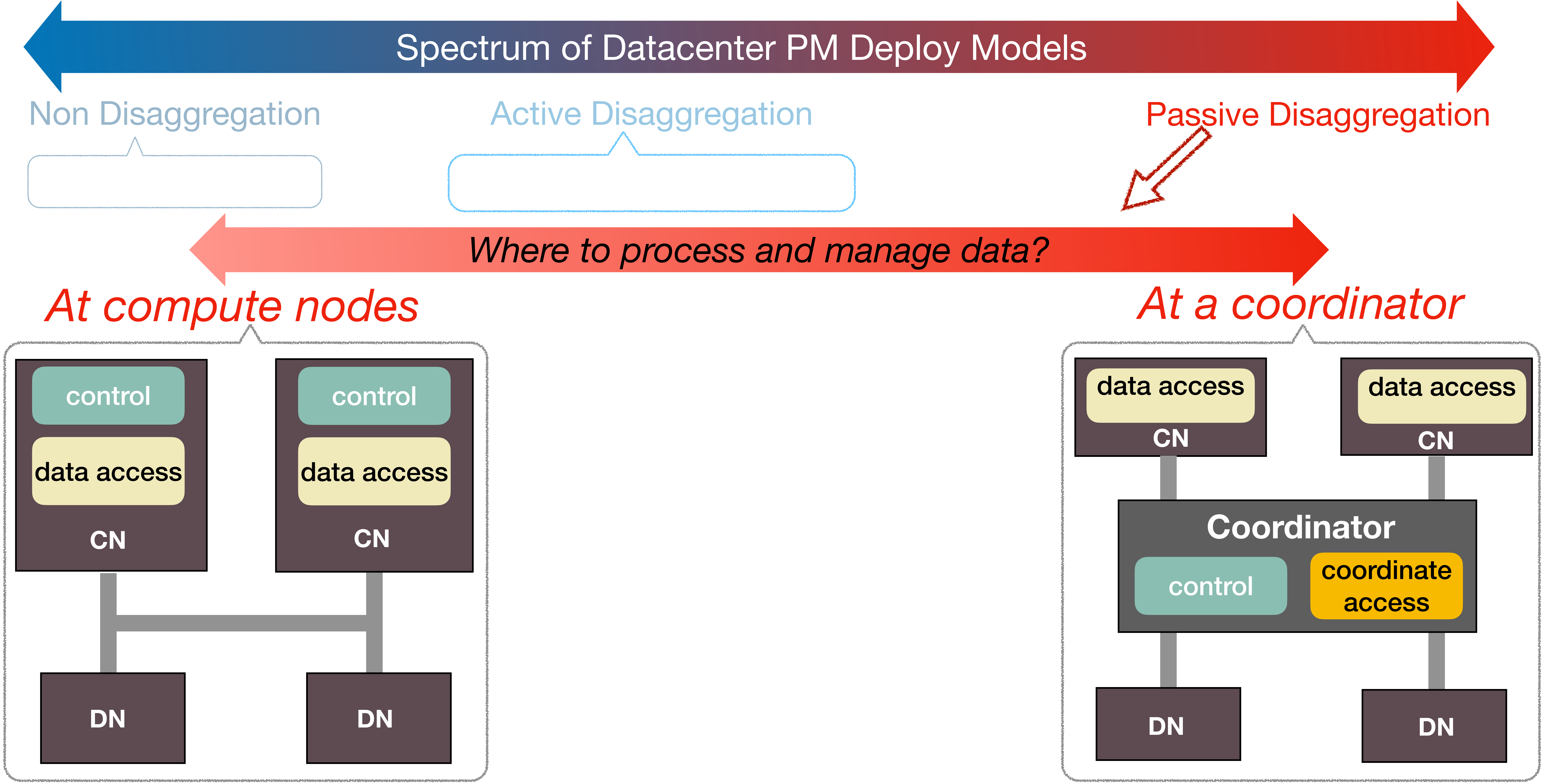
At a coordinator



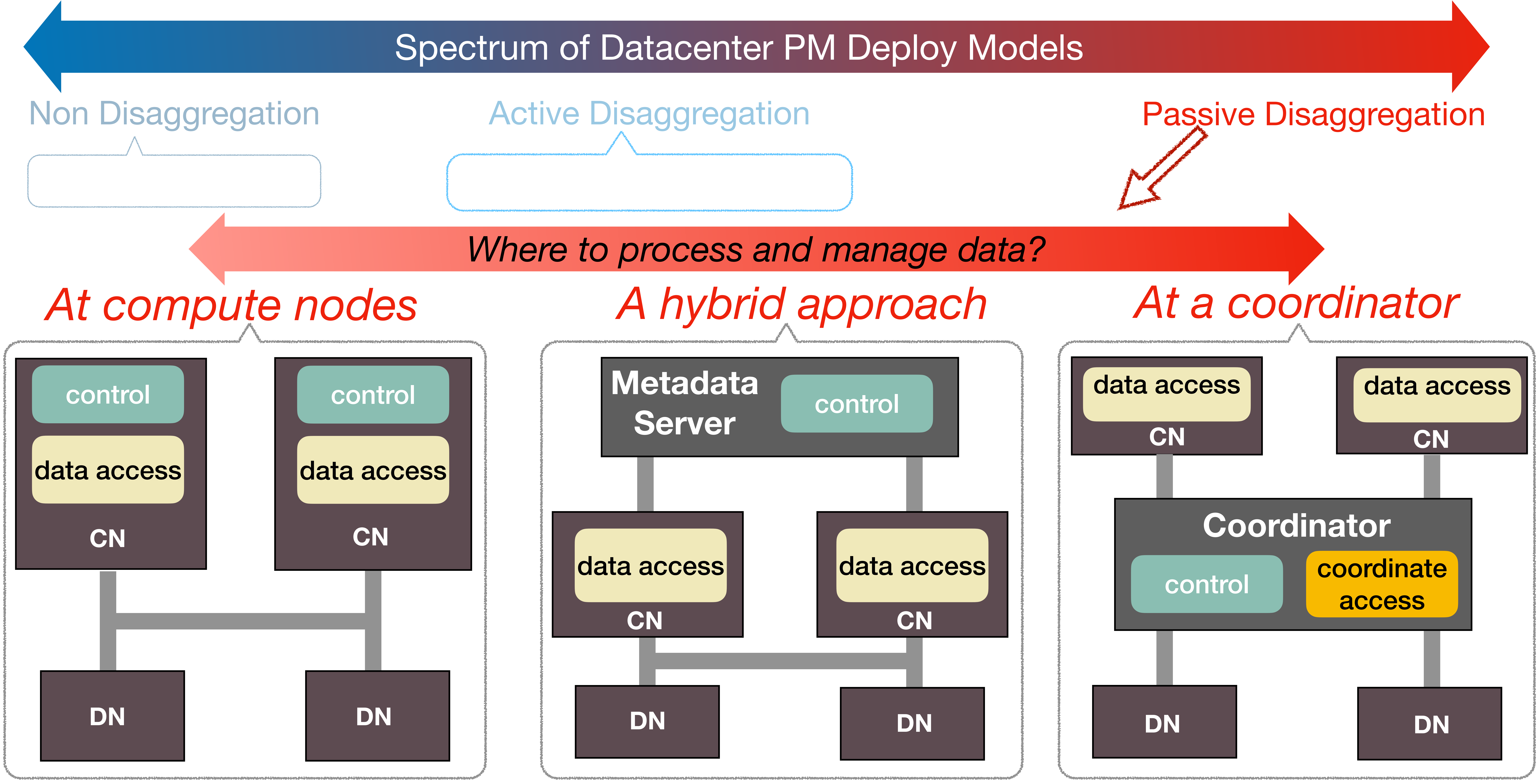
CN: Compute Node, **DN**: Data Node with PM



CN: Compute Node, **DN**: Data Node with PM



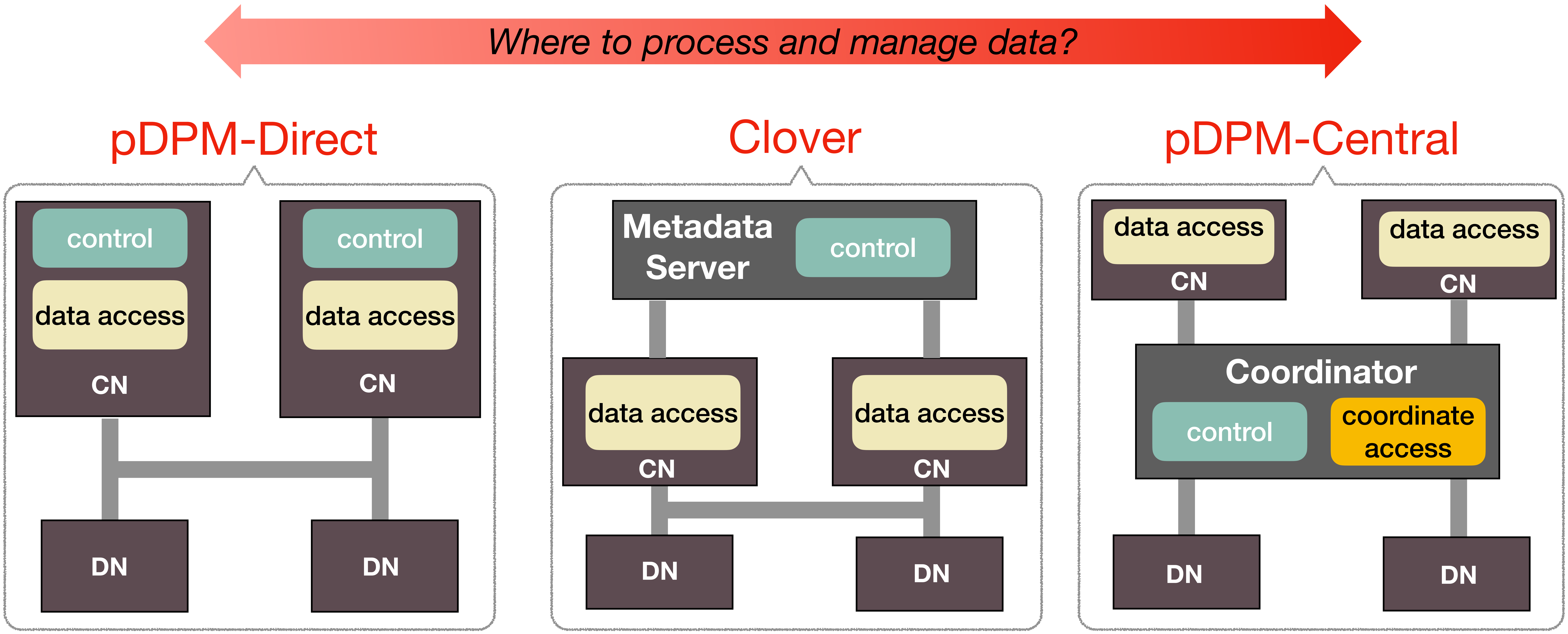
CN: Compute Node, **DN**: Data Node with PM



CN: Compute Node, **DN**: Data Node with PM

Passive Disaggregated PM (pDPM) Systems

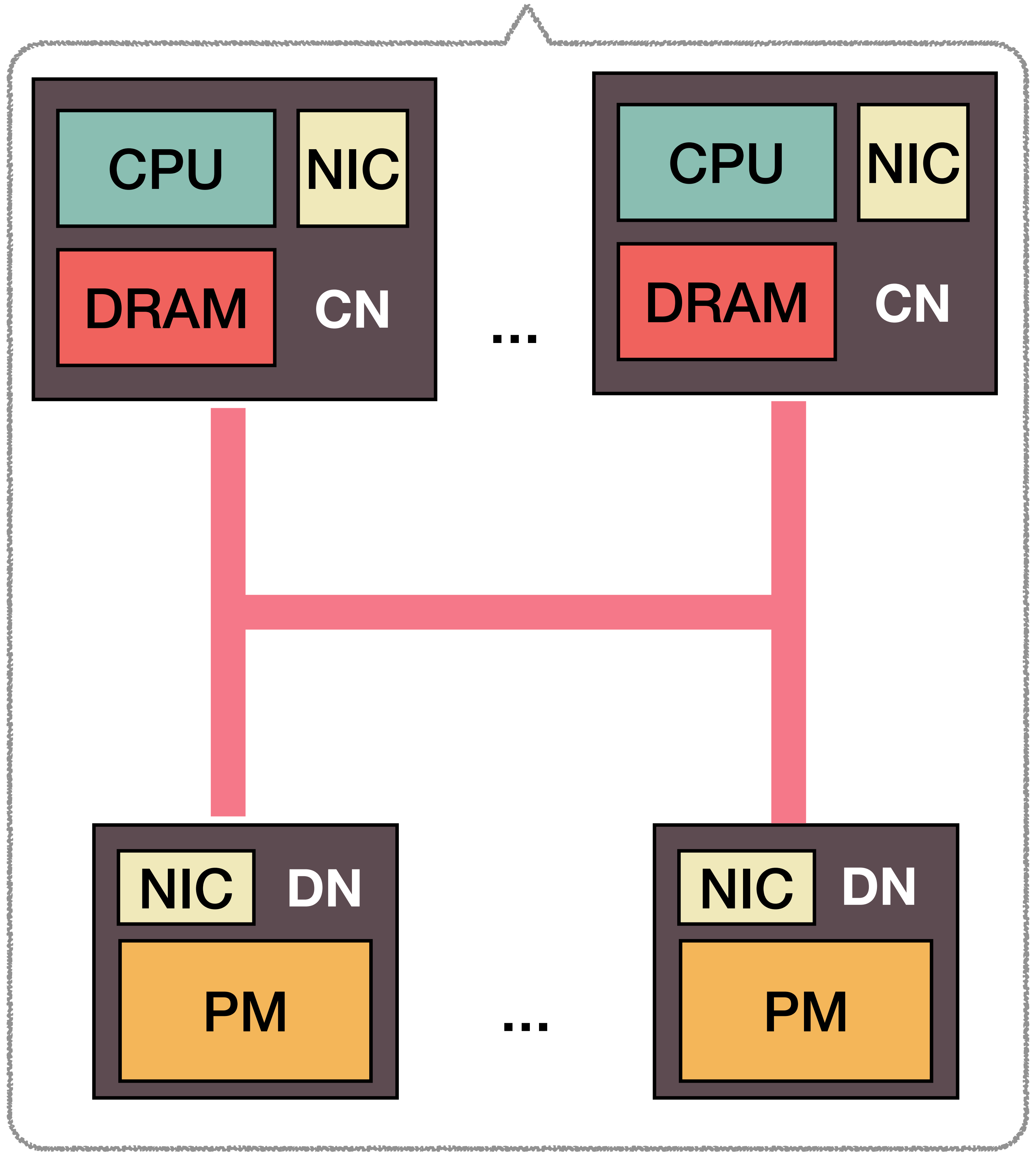
- We design and implement three pDPM key-value stores
  - At computer nodes → **pDPM-Direct**
  - At global coordinator → **pDPM-Central**
  - A hybrid approach → **Clover**
- Carry out extensive experiments: performance, scalability, costs
- Clover is the best pDPM model: perf similar to active DPM, but lower costs
- Discovered tradeoffs between passive and active DPMs



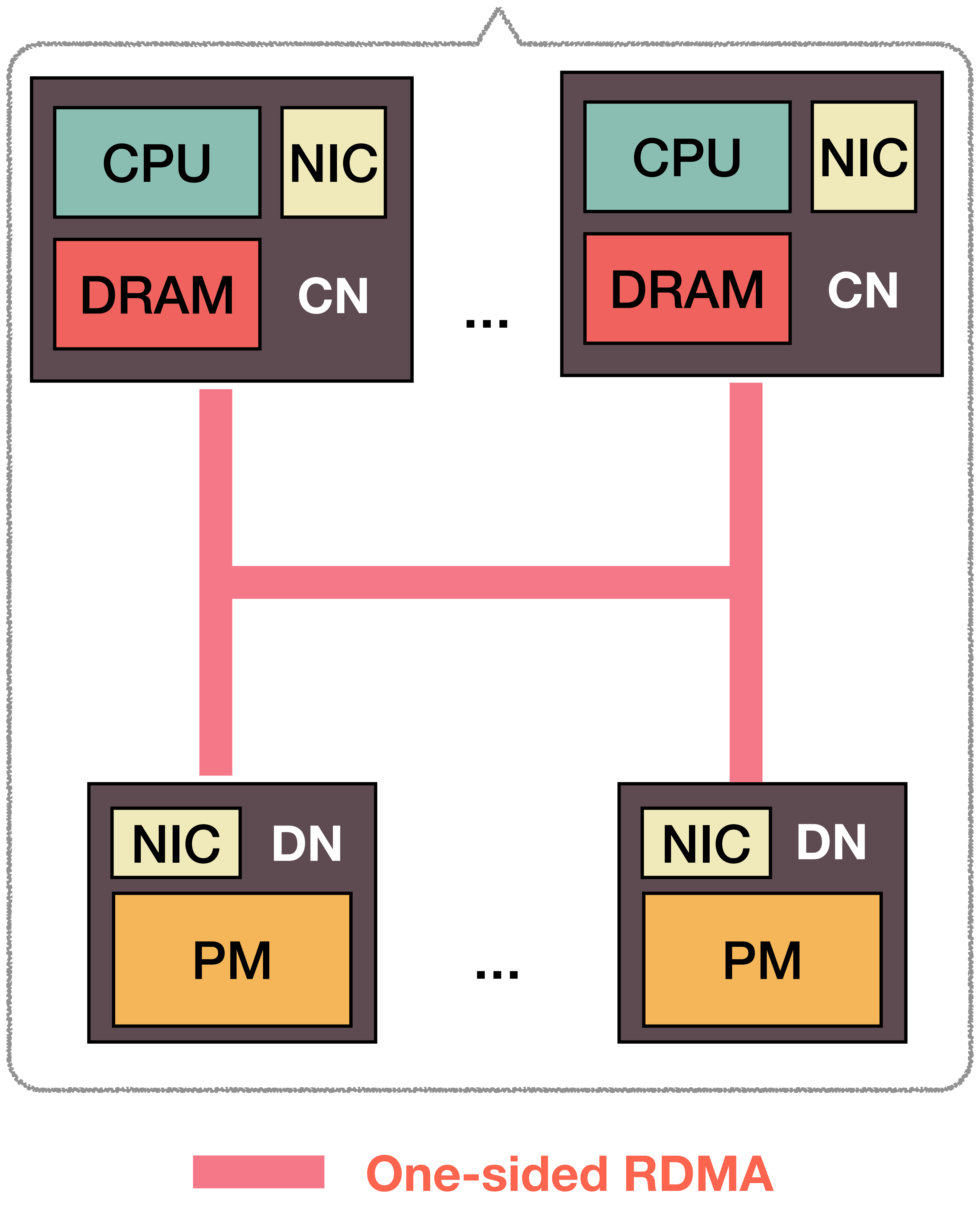
Where to process and manage data?
pDPM-Direct
control
data access
CN
control
data access
CN
DN
DN
Clover
Metadata Server
control
data access
CN
data access
CN
DN
DN
pDPM-Central
data access
CN
data access
CN
Coordinator
control
coordinate access
DN
DN

Where to process and manage data?
pDPM-Direct
control
data access
CN
control
data access
CN
DN
DN
Clover
Metadata Server
control
data access
CN
data access
CN
DN
DN
pDPM-Central
data access
CN
data access
CN
Coordinator
control
coordinate access
DN
DN

pDPM-Direct: Directly Access and Manage DNs from CNs



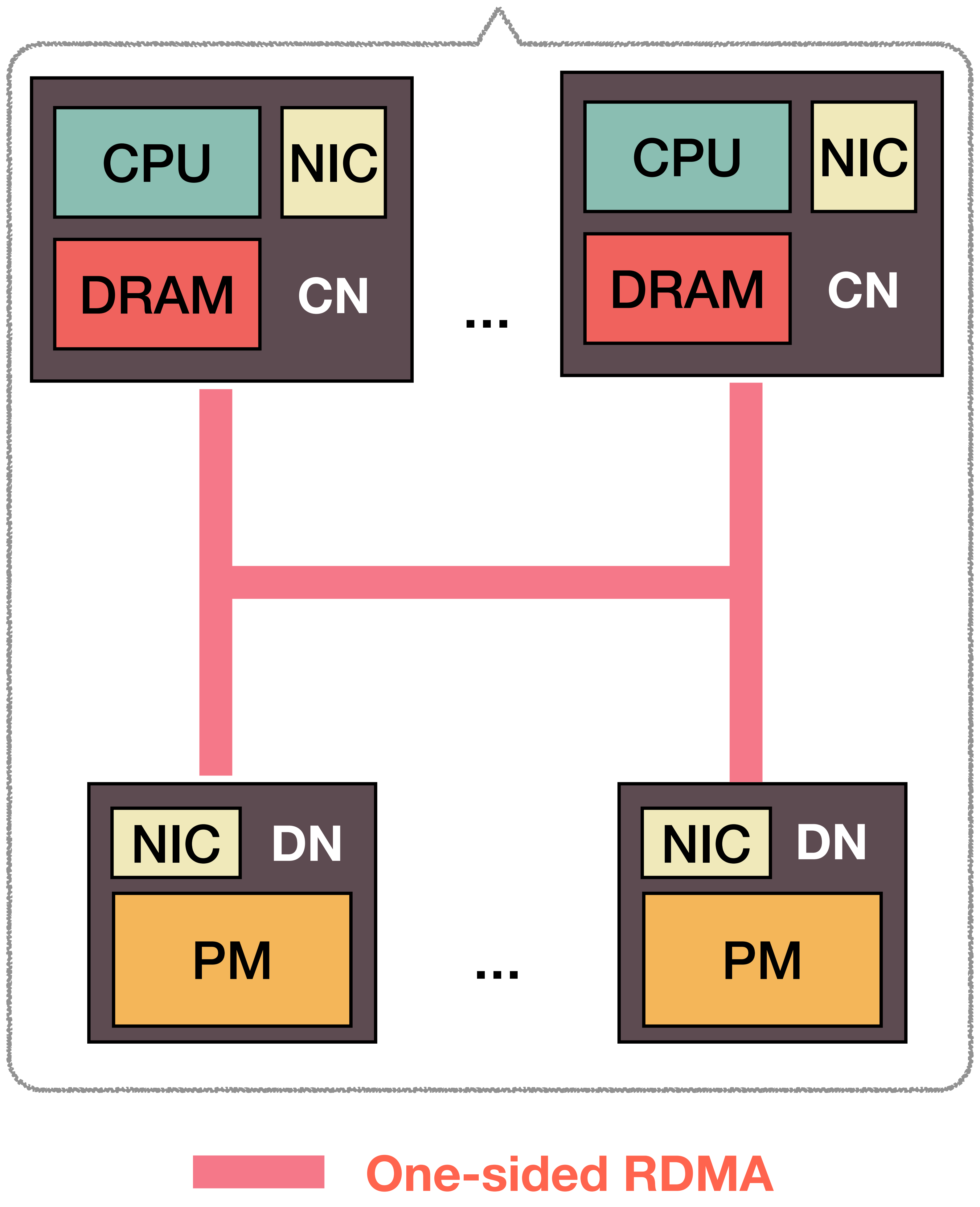
pDPM-Direct: Directly Access and Manage DNs from CNs



Overall Architecture

- CNs access and manage DNs directly via one-sided RDMA
- Both data and control planes run within CNs

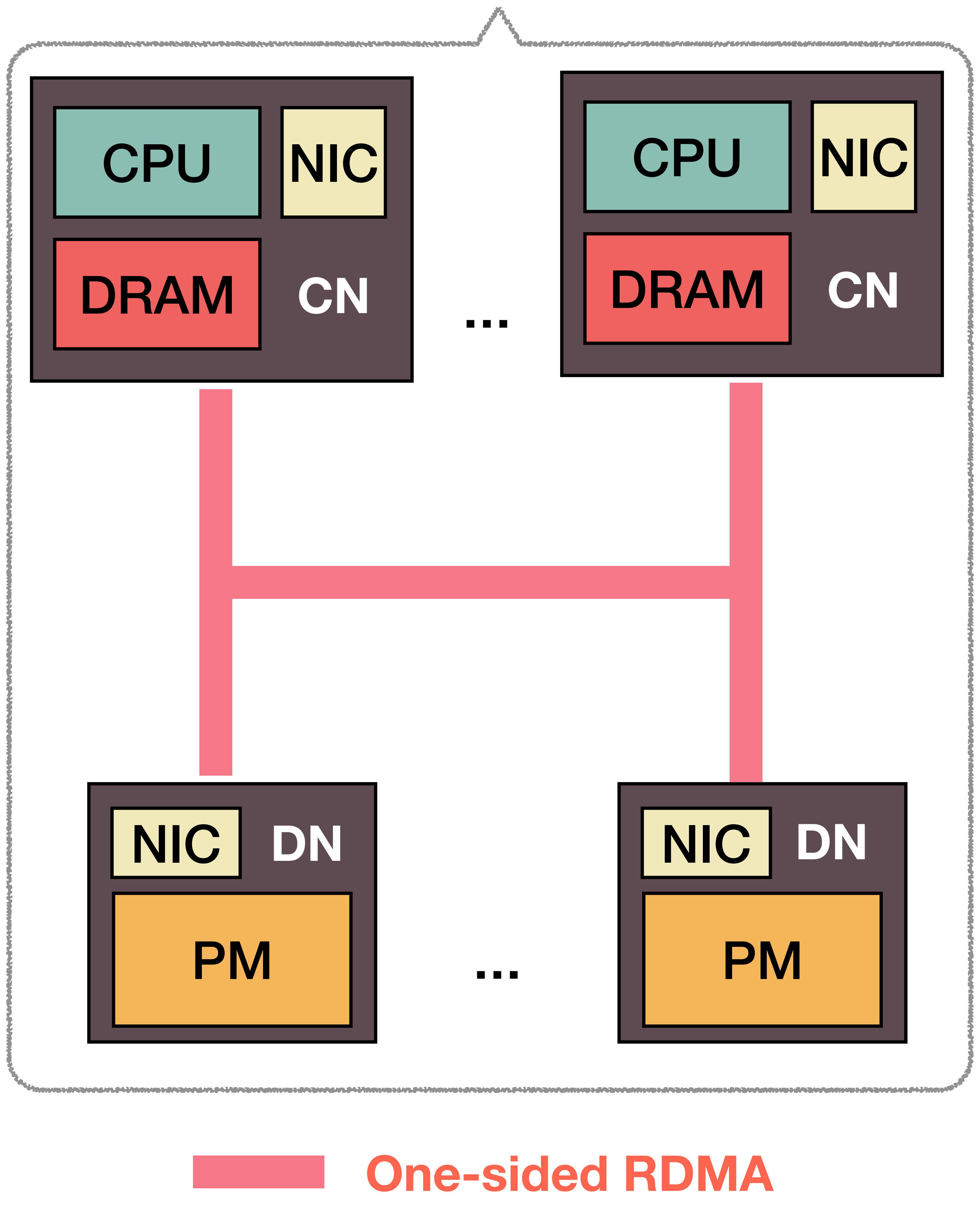
pDPM-Direct: Directly Access and Manage DNs from CNs



Overall Architecture

- CNs access and manage DNs directly via one-sided RDMA
- Both data and control planes run within CNs

pDPM-Direct: Directly Access and Manage DNs from CNs



Overall Architecture

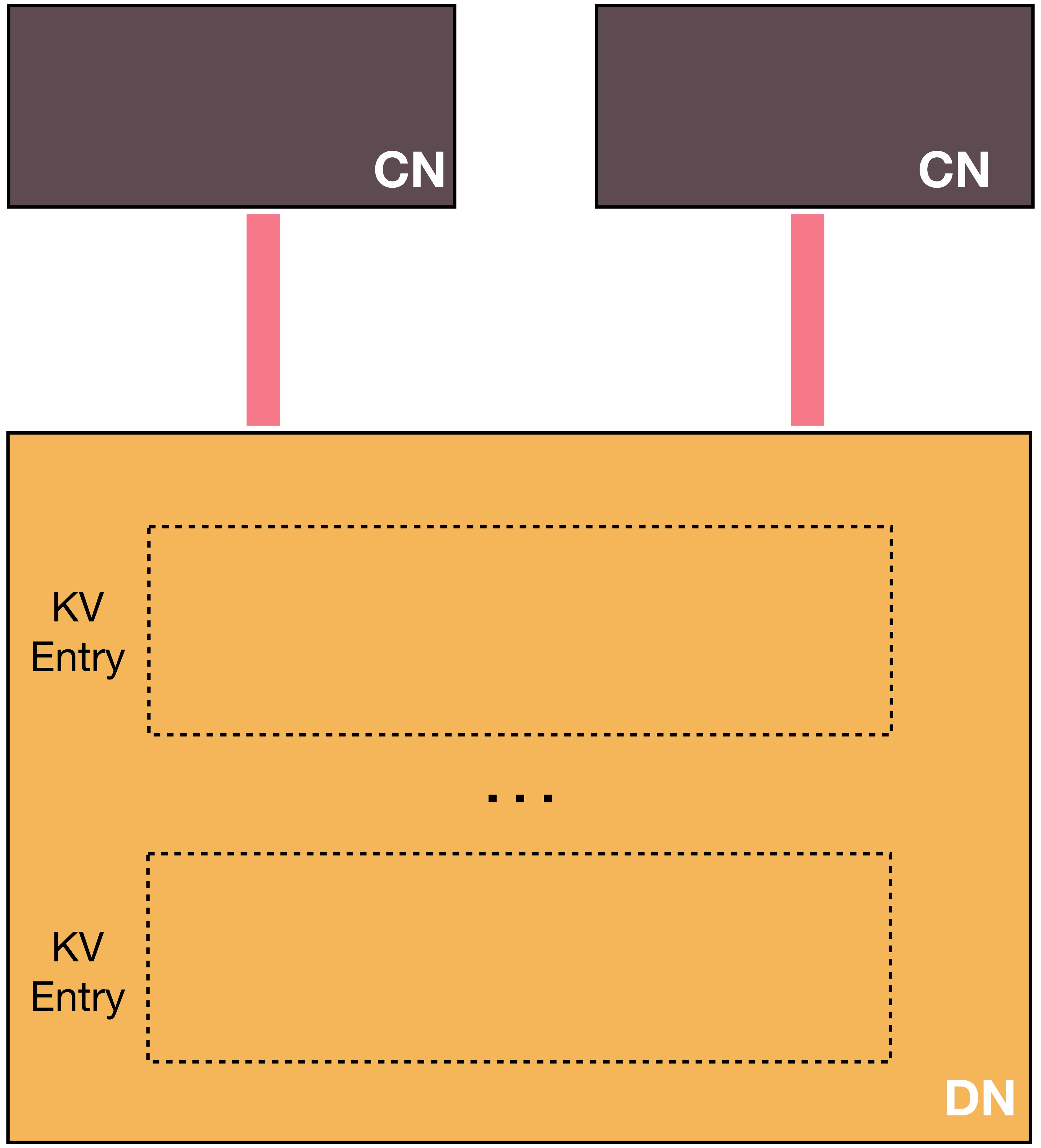
- CNs access and manage DNs directly via one-sided RDMA
- Both data and control planes run within CNs

Challenges

- How to manage DN space?
- How to coordinate concurrent reads/writes across CNs?

Our solution

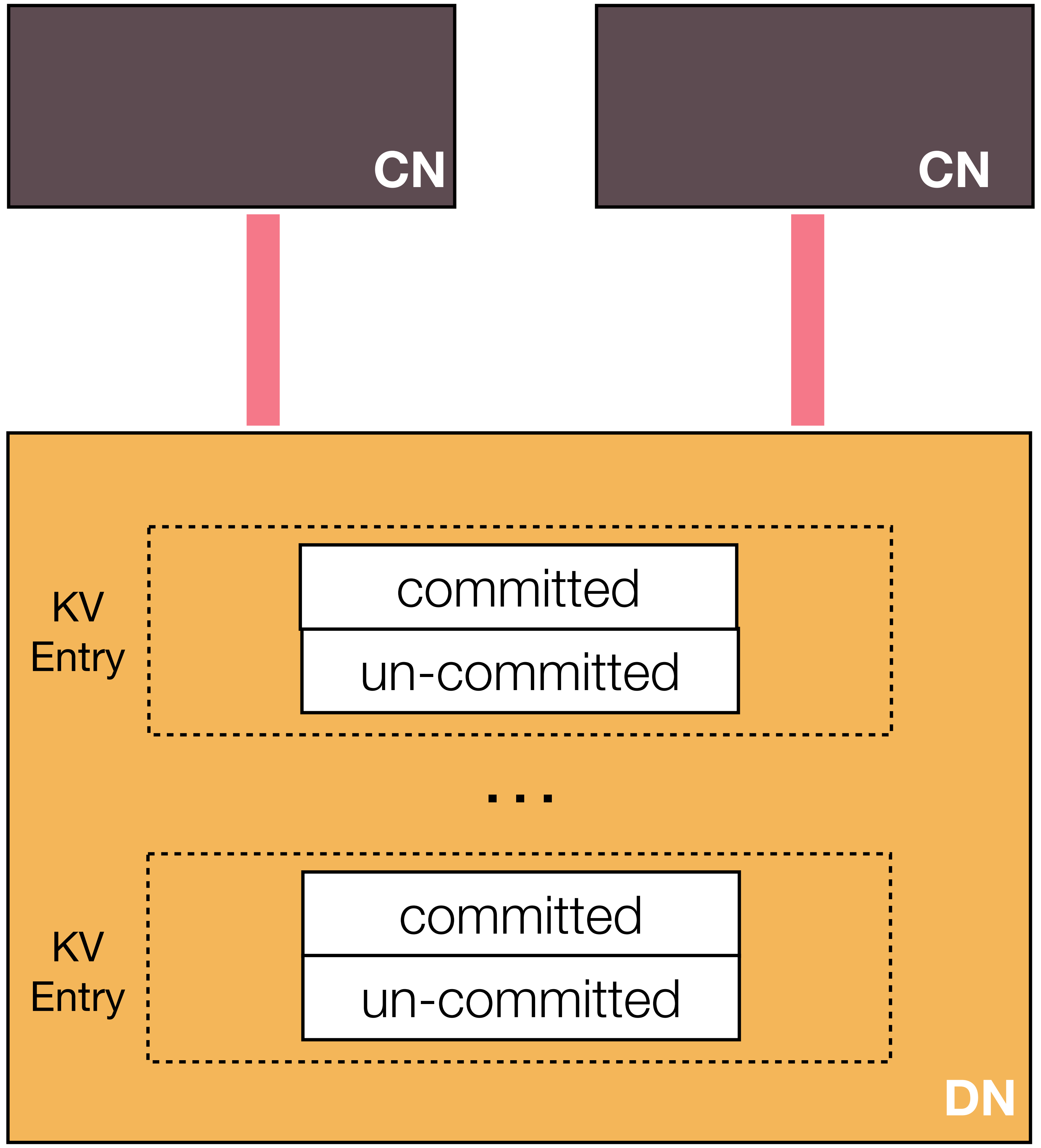
pDPM-Direct



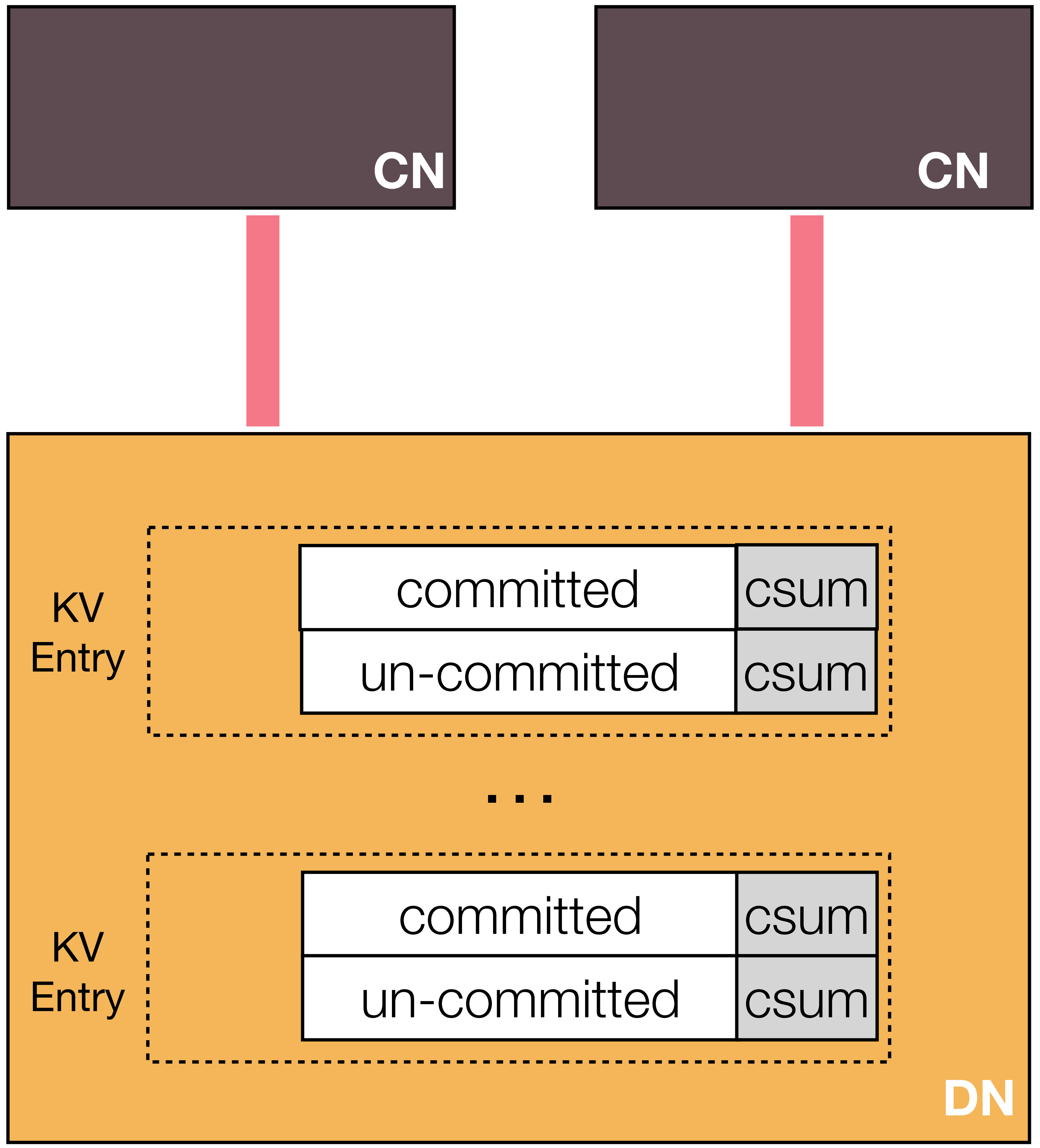
Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)

pDPM-Direct



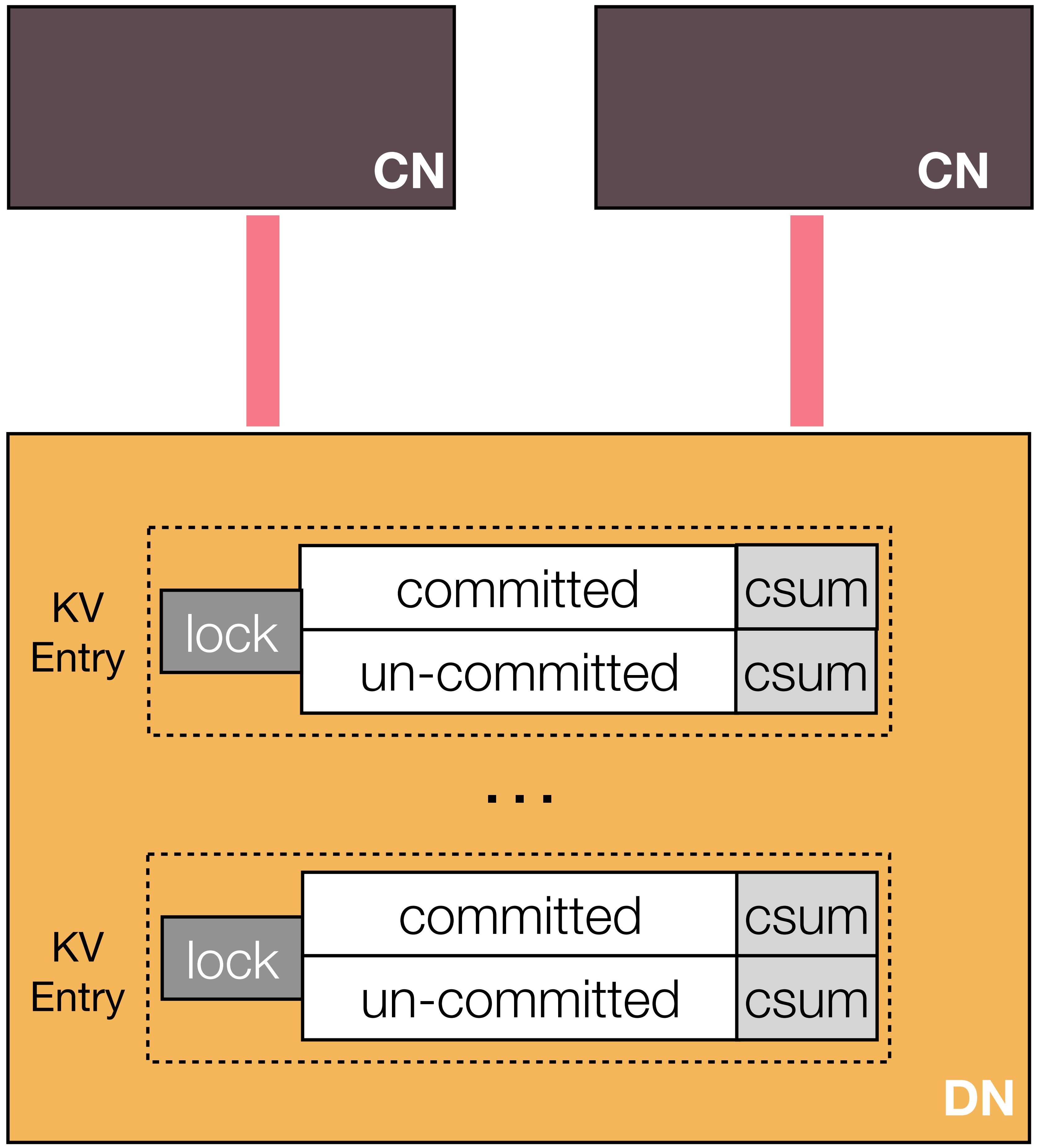
pDPM-Direct



Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)

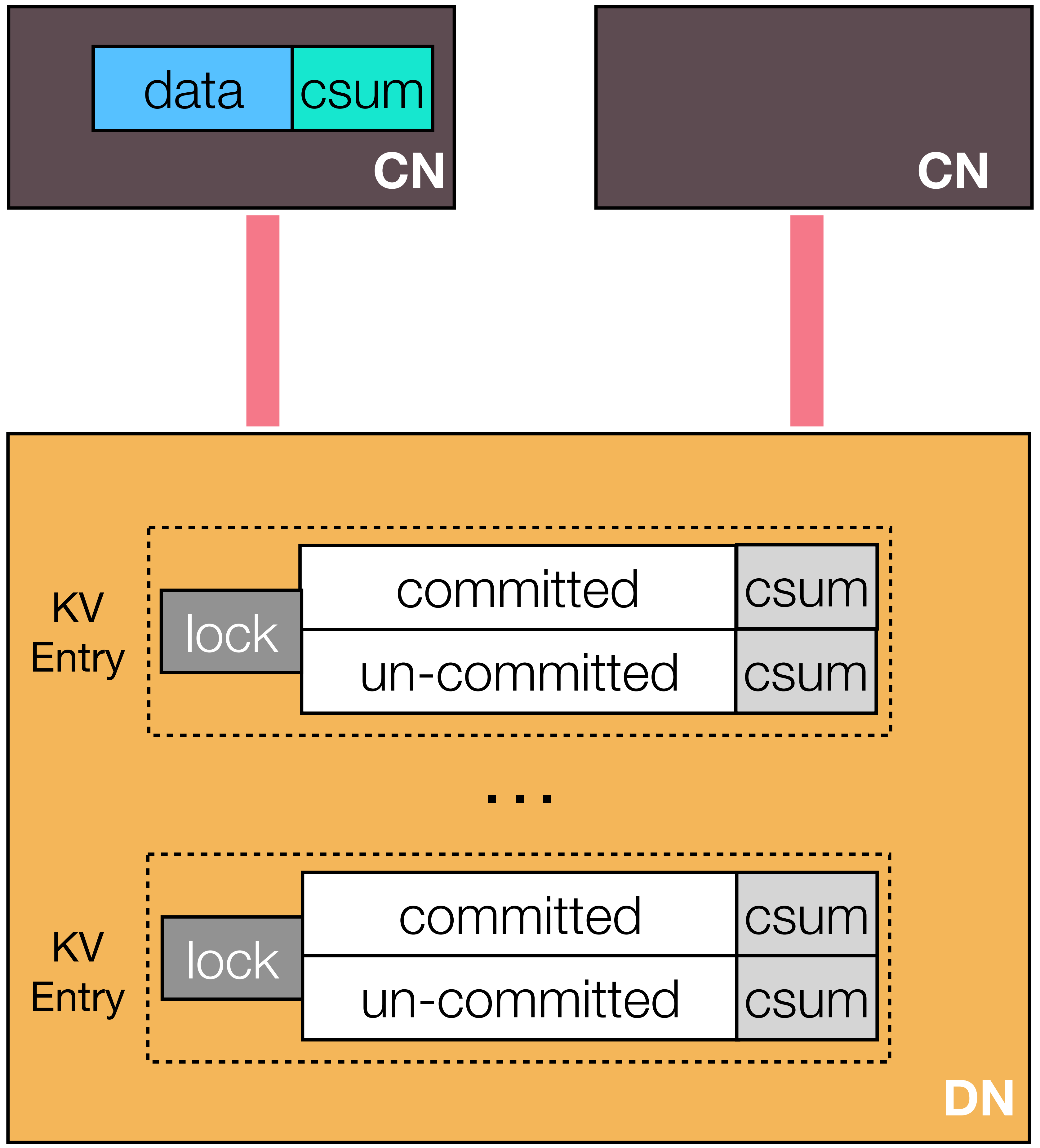
pDPM-Direct



Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

pDPM-Direct



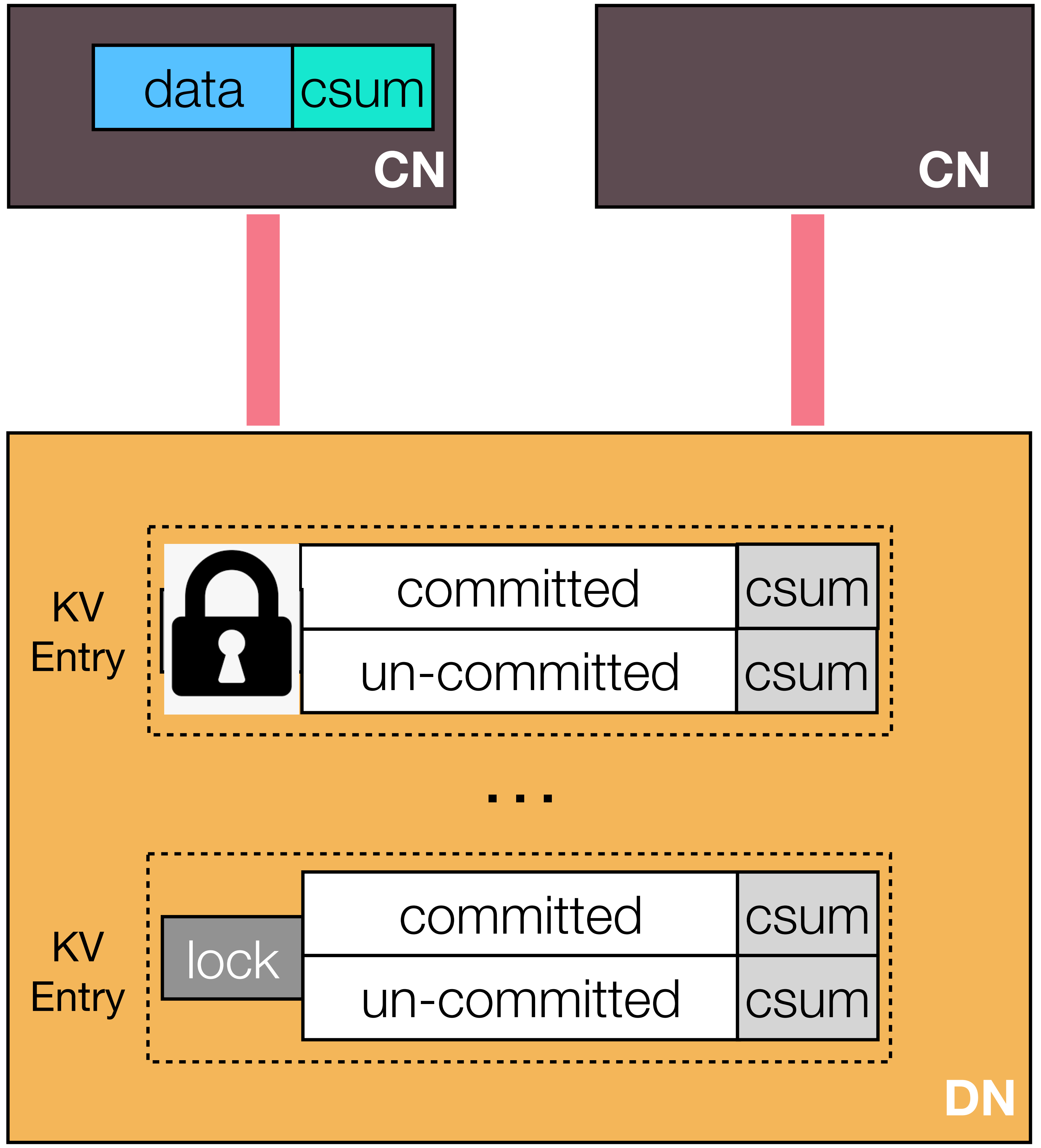
Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

pDPM-Direct



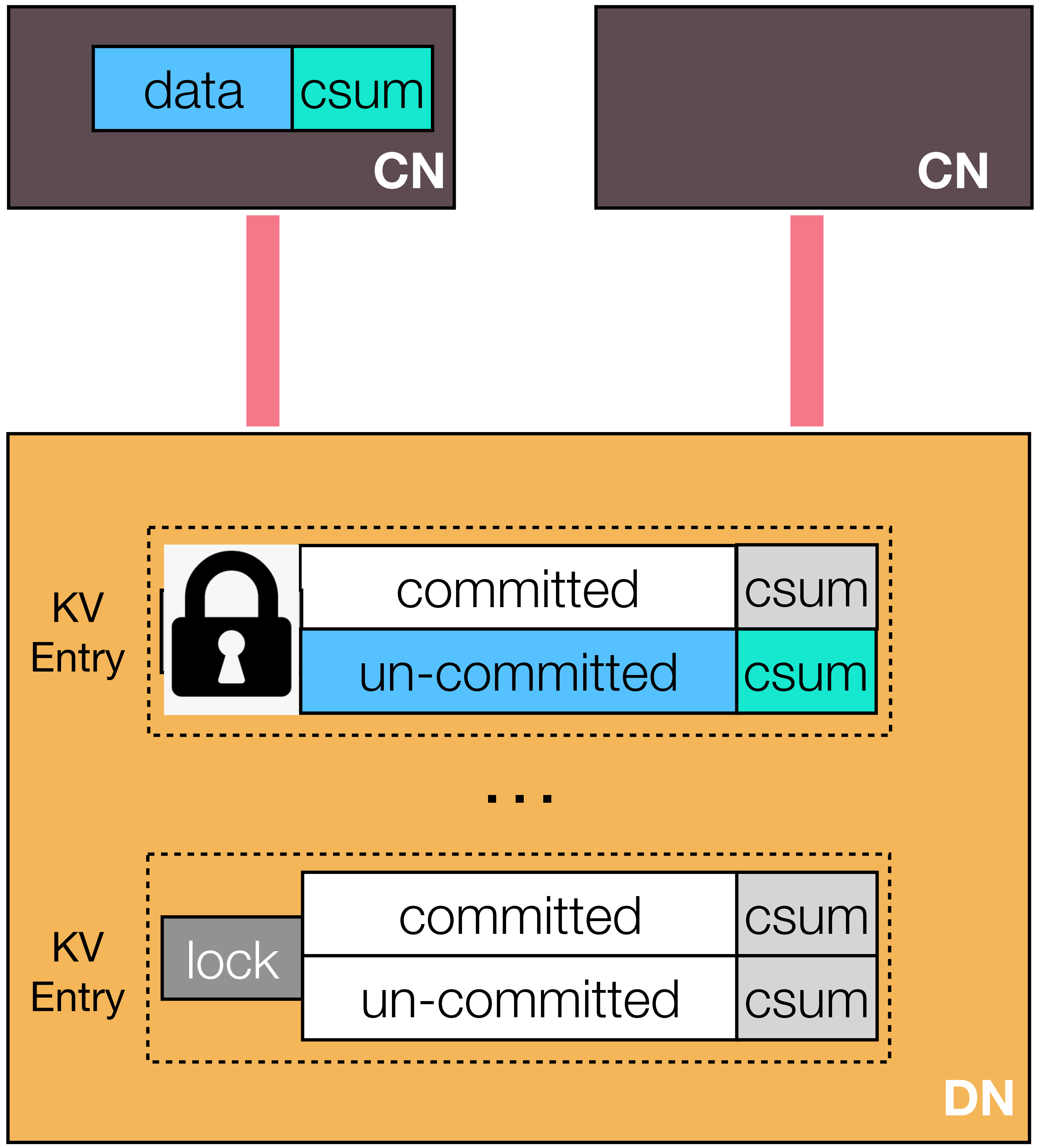
Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

pDPM-Direct



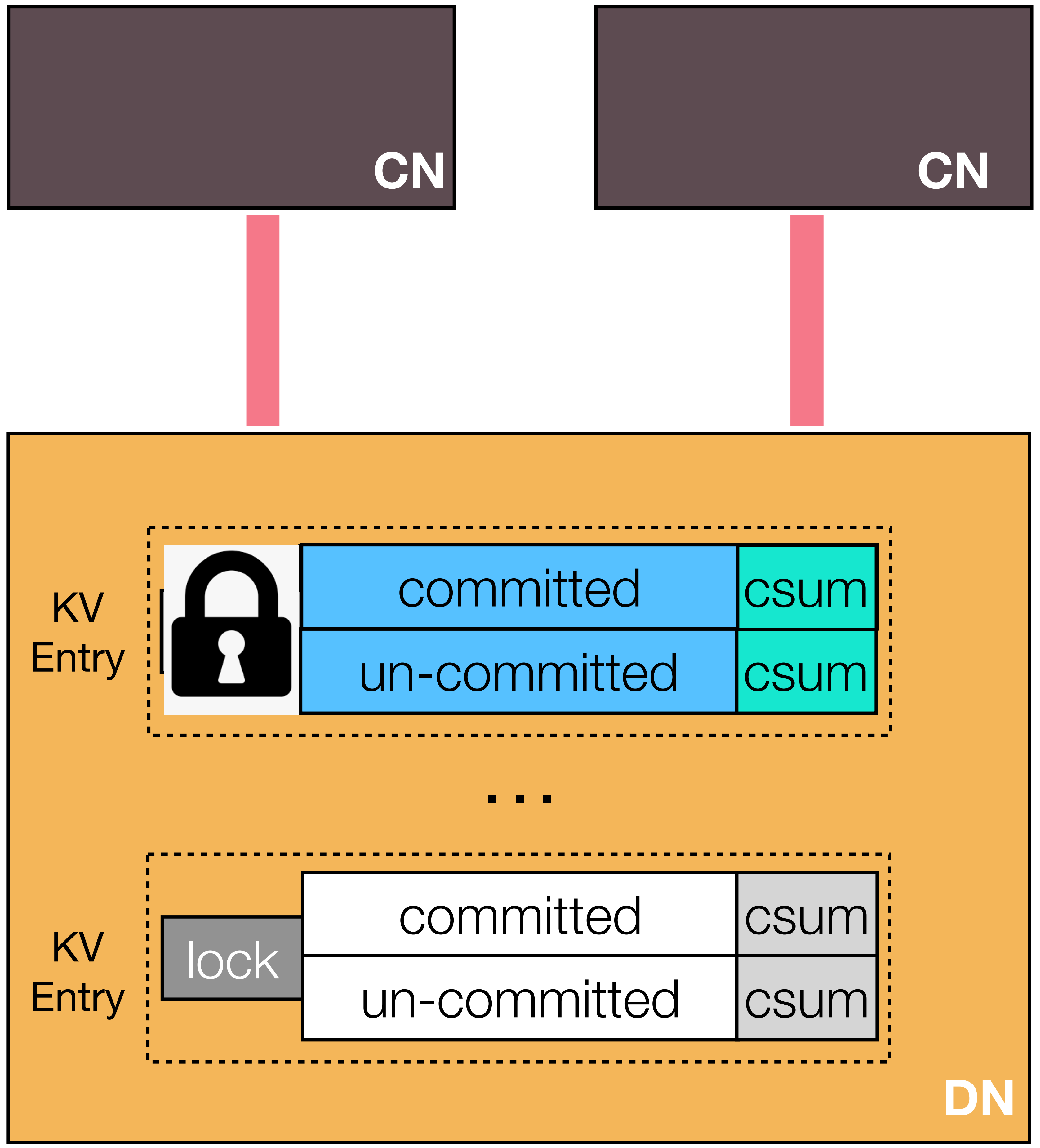
Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

pDPM-Direct

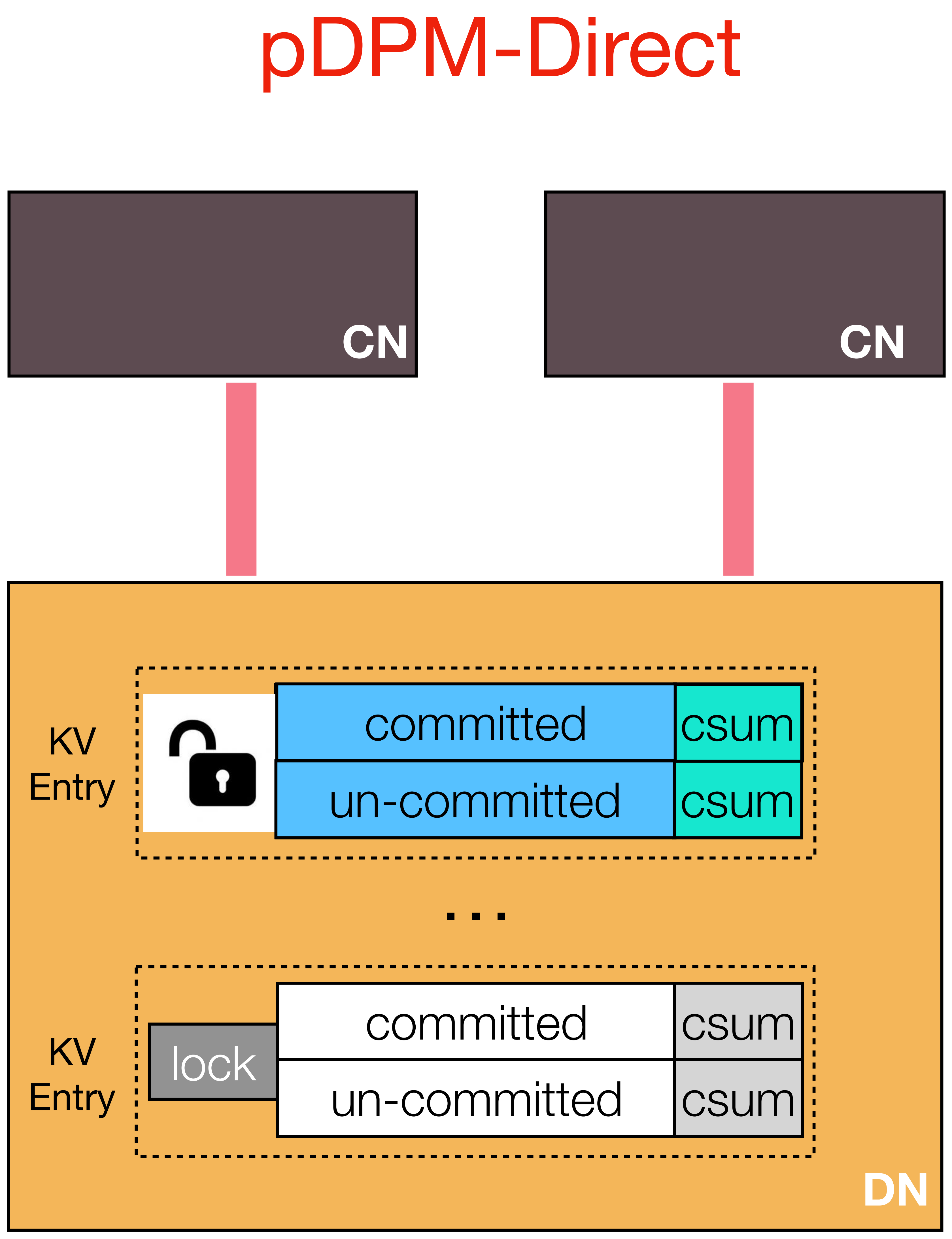


Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock



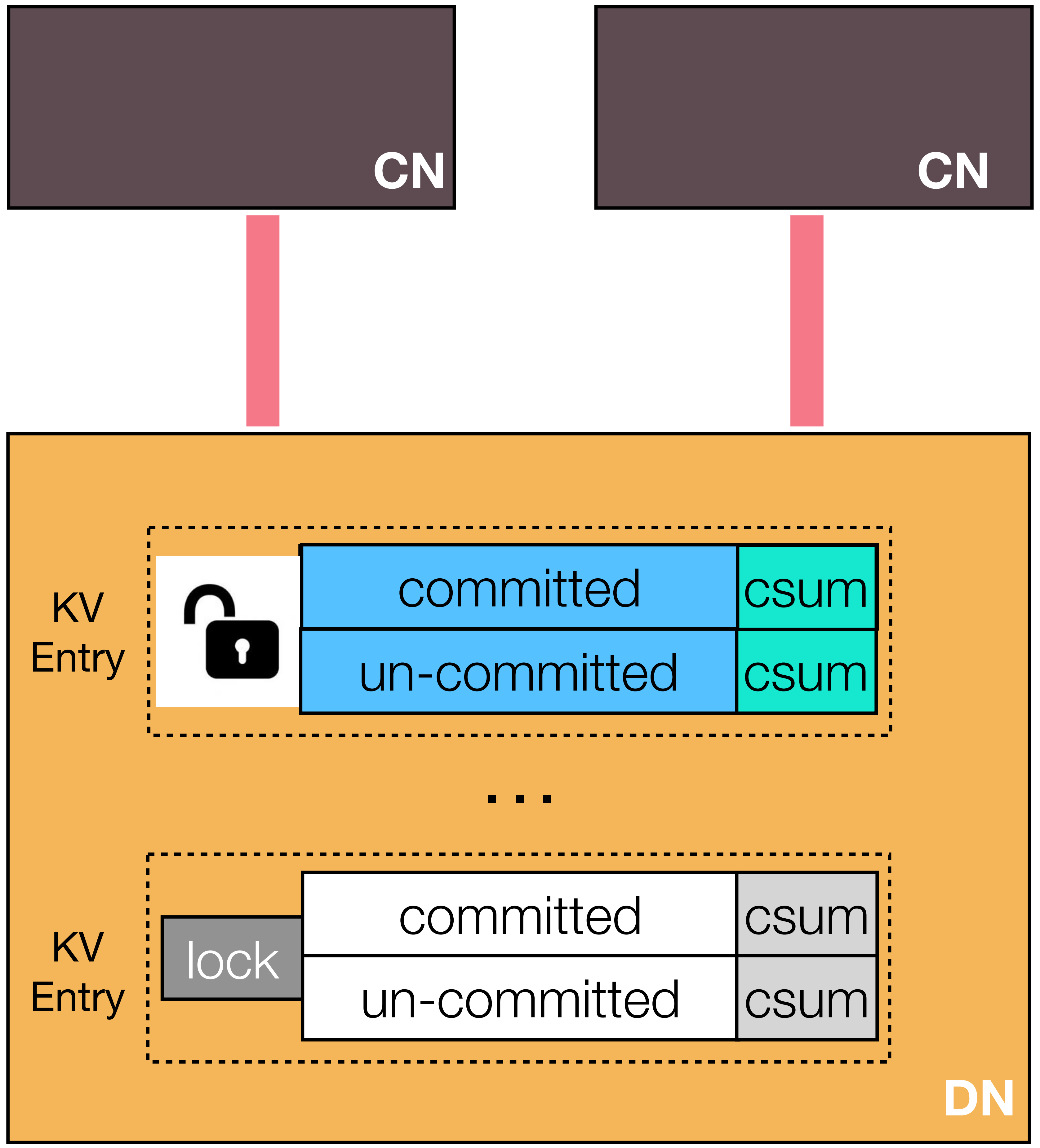
Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

pDPM-Direct



Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

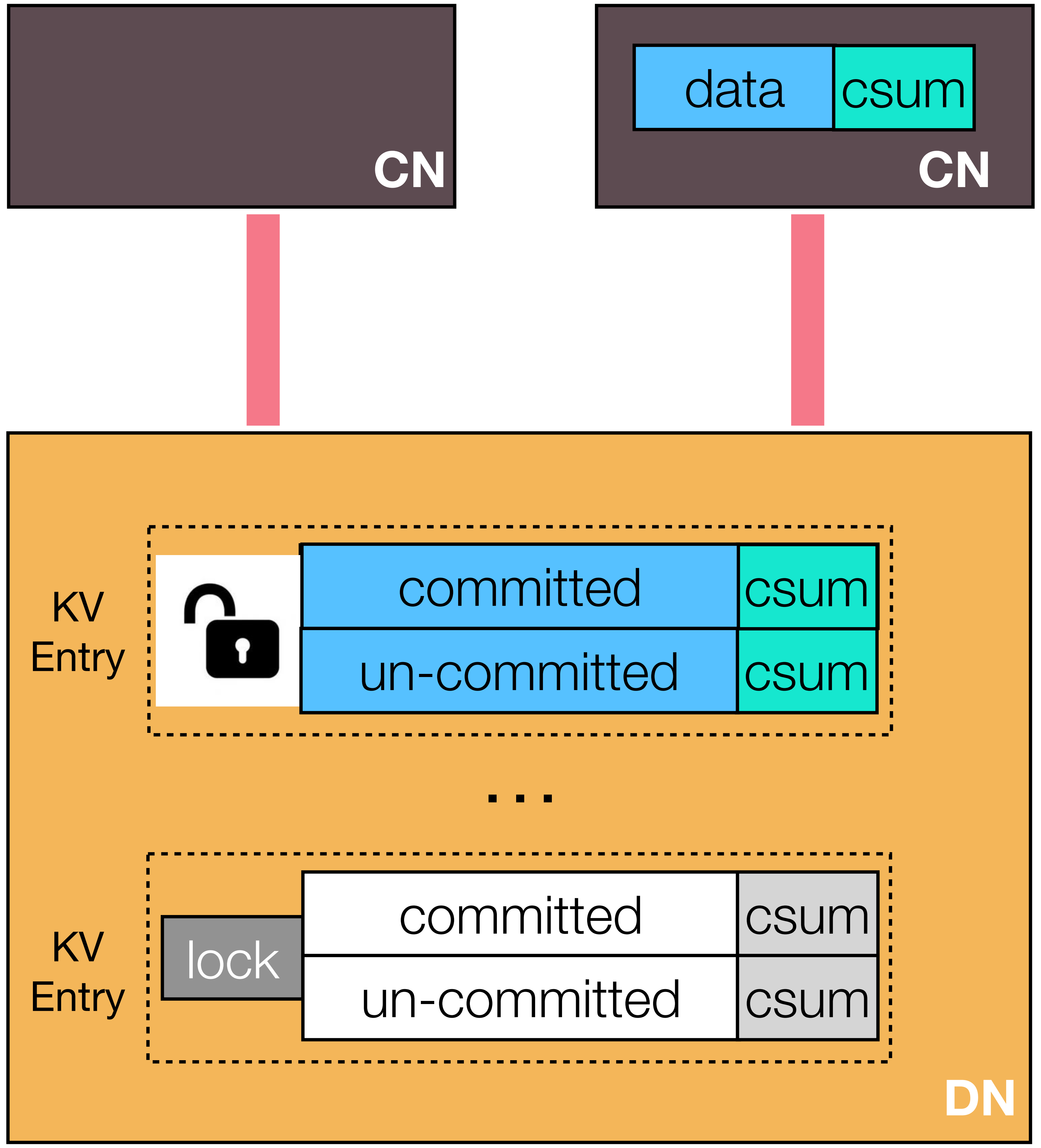
Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

Read Flow

- CN reads committed data and CRC
- CN checks if CRC match. If mismatch, retry

pDPM-Direct



Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

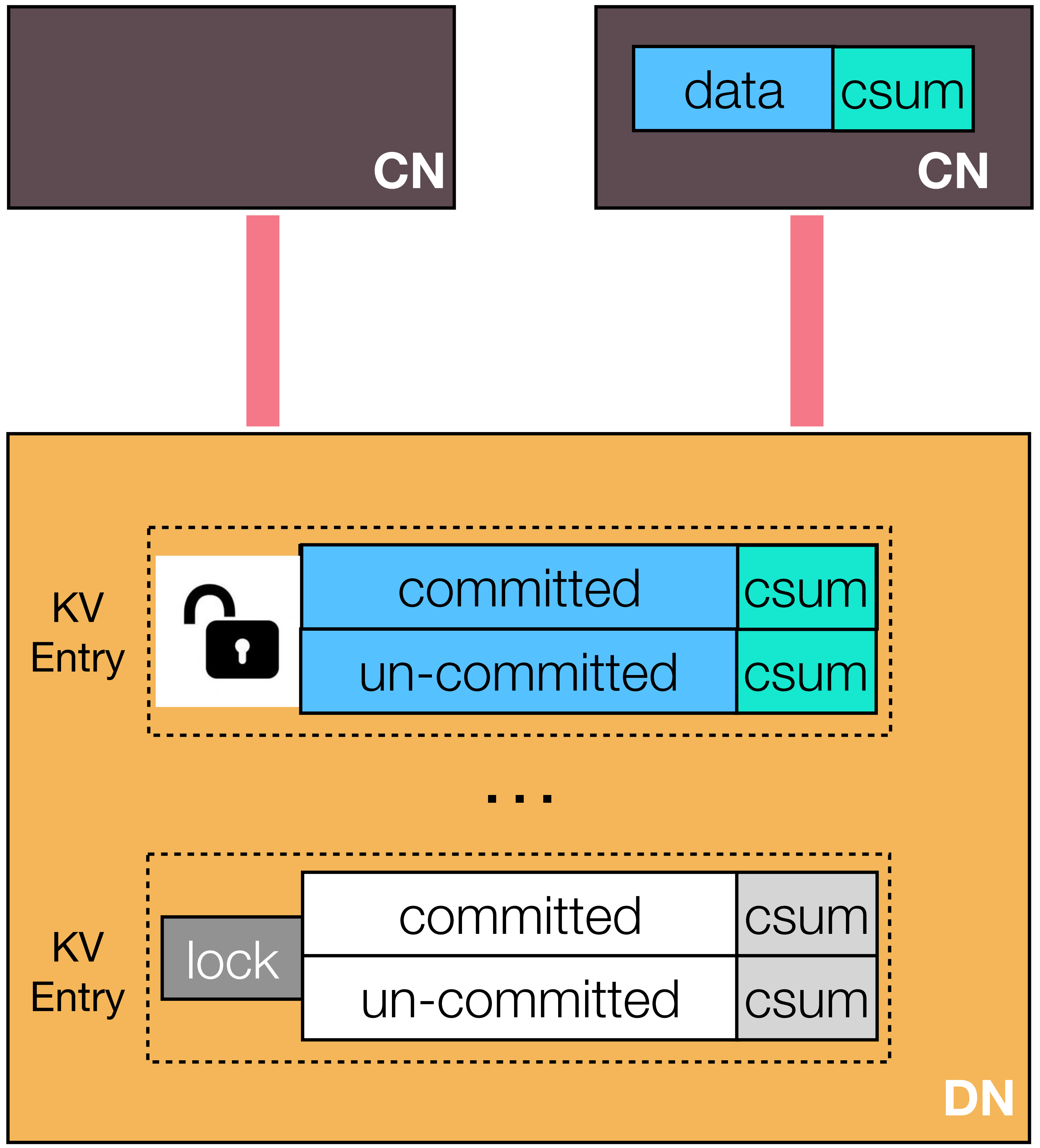
Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

Read Flow

- CN reads committed data and CRC
- CN checks if CRC match. If mismatch, retry

pDPM-Direct



Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

Write Flow

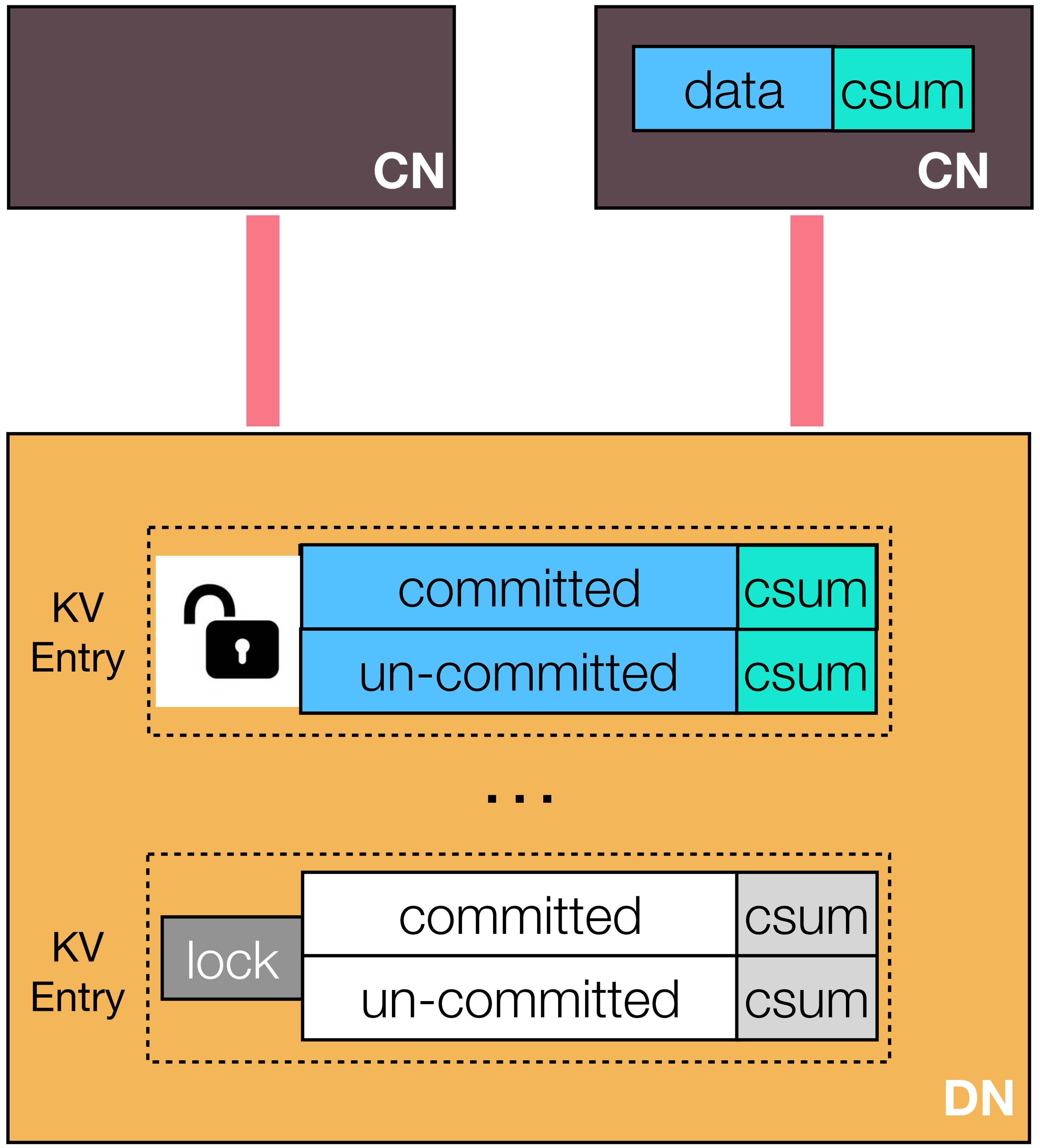
- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

Read Flow

- CN reads committed data and CRC
- CN checks if CRC match. If mismatch, retry

Best case
Write: 4 RTT + csum calc
Read: 1 RTT + csum calc

pDPM-Direct



Our solution

- Pre-assign two spaces for each KV entry (`committed+uncommitted`)
- Lock-free, checksum-based read (`csum`)
- RDMA c&s-based write lock (`lock`)

Write Flow

- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

Read Flow

- CN reads committed data and CRC
- CN checks if CRC match. If mismatch, retry

Best case
Write: 4 RTT + csum calc
Read: 1 RTT + csum calc

Slow write
Slow with large data
Poor scalability under concurrent accesses

☹


Where to process and manage data?
pDPM-Direct
control
data access
CN
control
data access
CN
DN
DN
Clover
Metadata Server
control
data access
CN
data access
CN
DN
DN
pDPM-Central
data access
CN
data access
CN
Coordinator
control
coordinate access
DN
DN

- Slow write
- Slow for large data

Distributed data & metadata planes

- Slow write
- Slow for large data

Distributed data & metadata planes

Centralized data & metadata planes

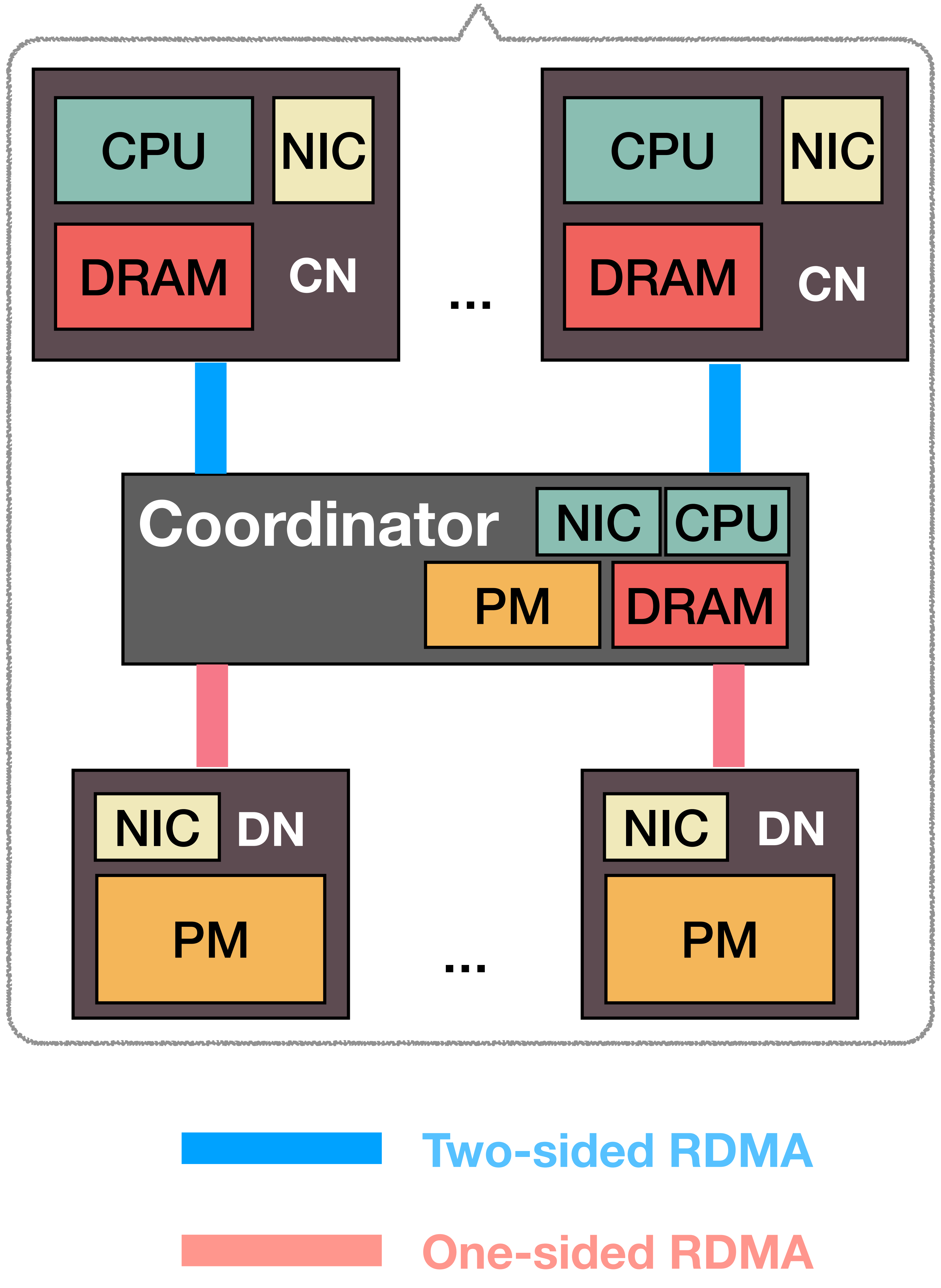
pDPM-Central: A Central Coordinator between CNs and DNs

The central coordinator

- Manages DN space
- Serializes CNs accesses with local locking

CNs communicate with the coordinator through two-sided RDMA

Coordinator accesses DNs through one-sided RDMA



pDPM-Central: A Central Coordinator between CNs and DNs

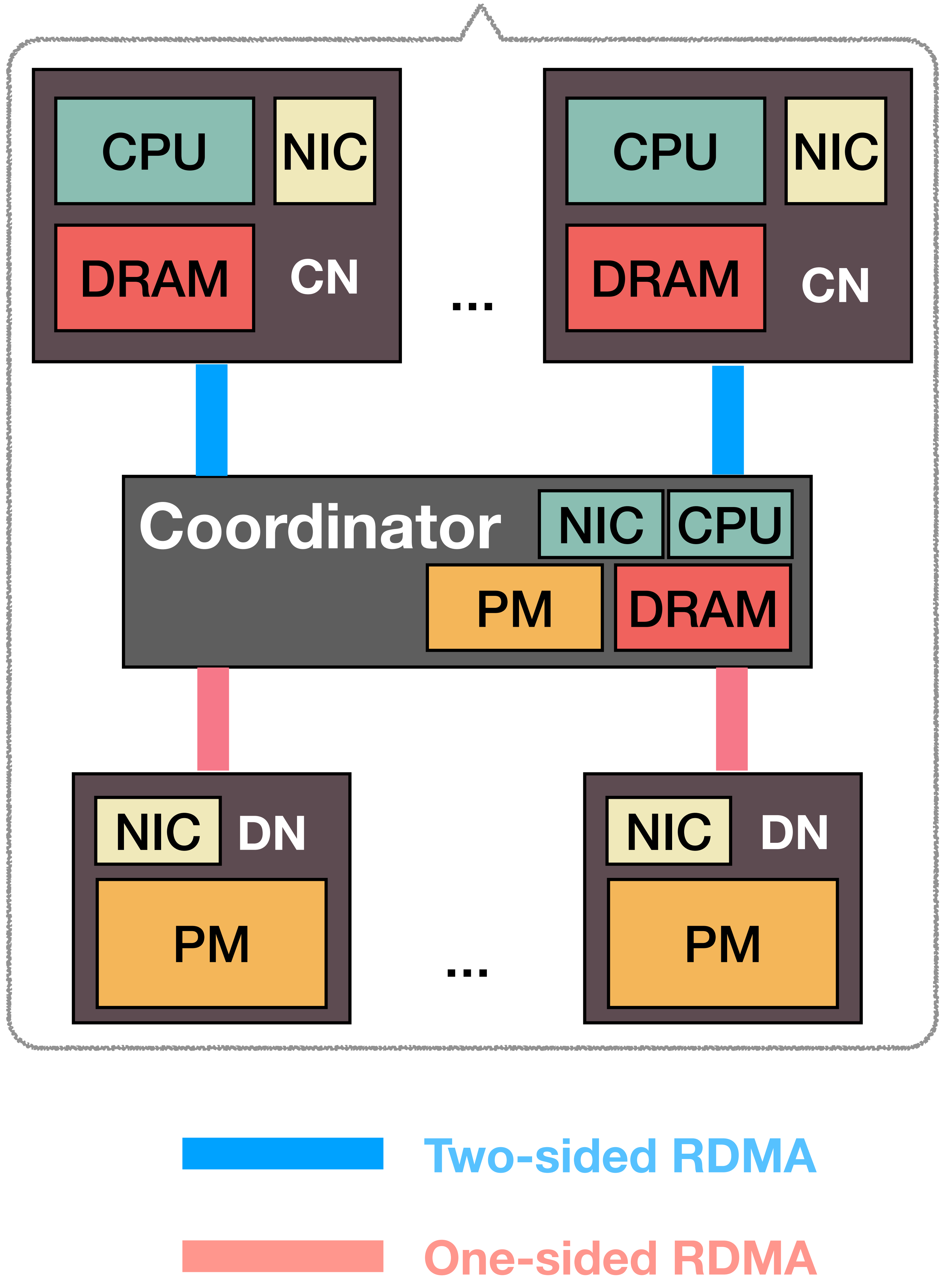
The central coordinator

- Manages DN space
- Serializes CNs accesses with local locking

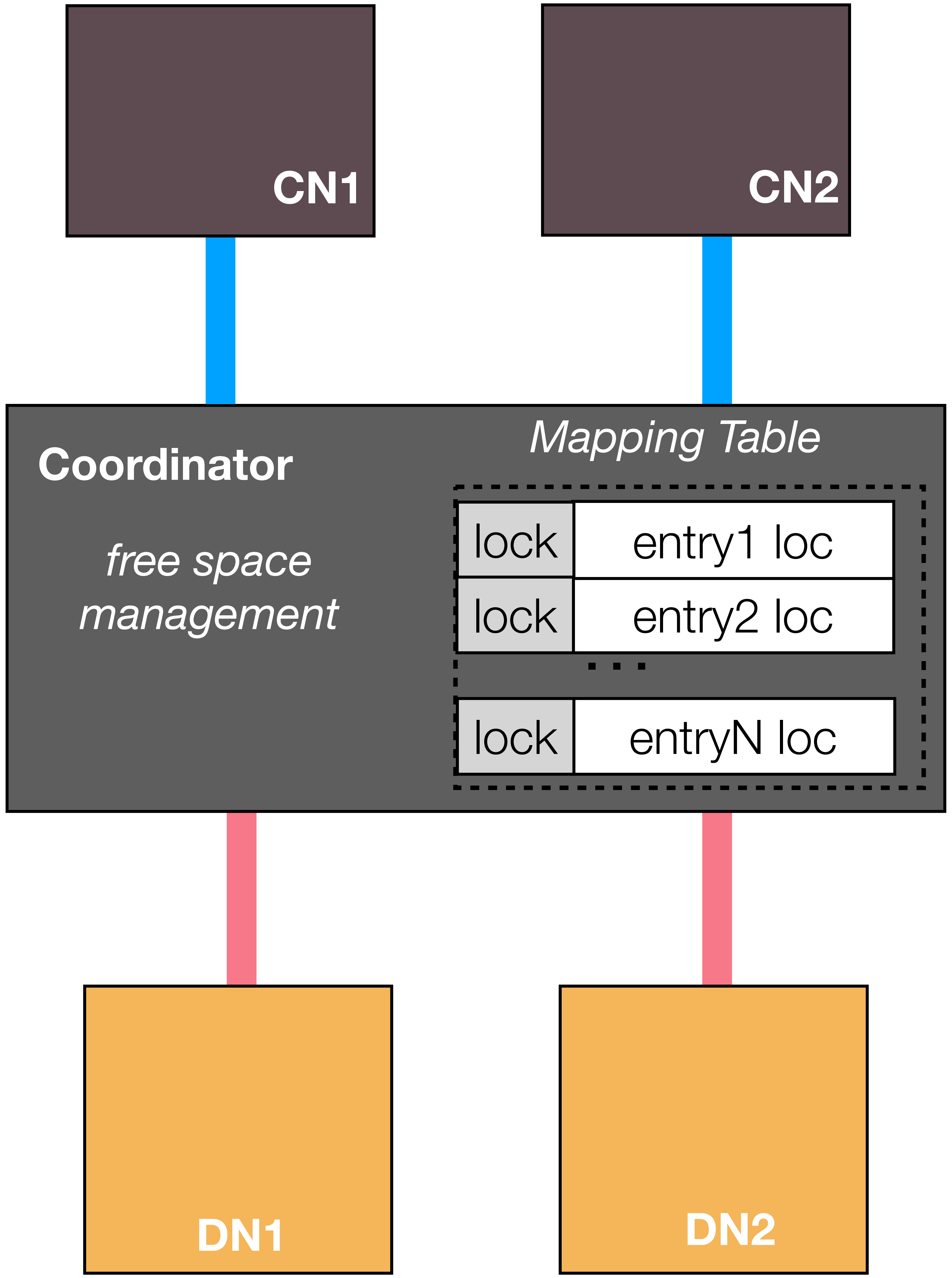
CNs communicate with the coordinator through two-sided RDMA

Coordinator accesses DNs through one-sided RDMA

Easier to manage DNs and coordinate concurrent accesses

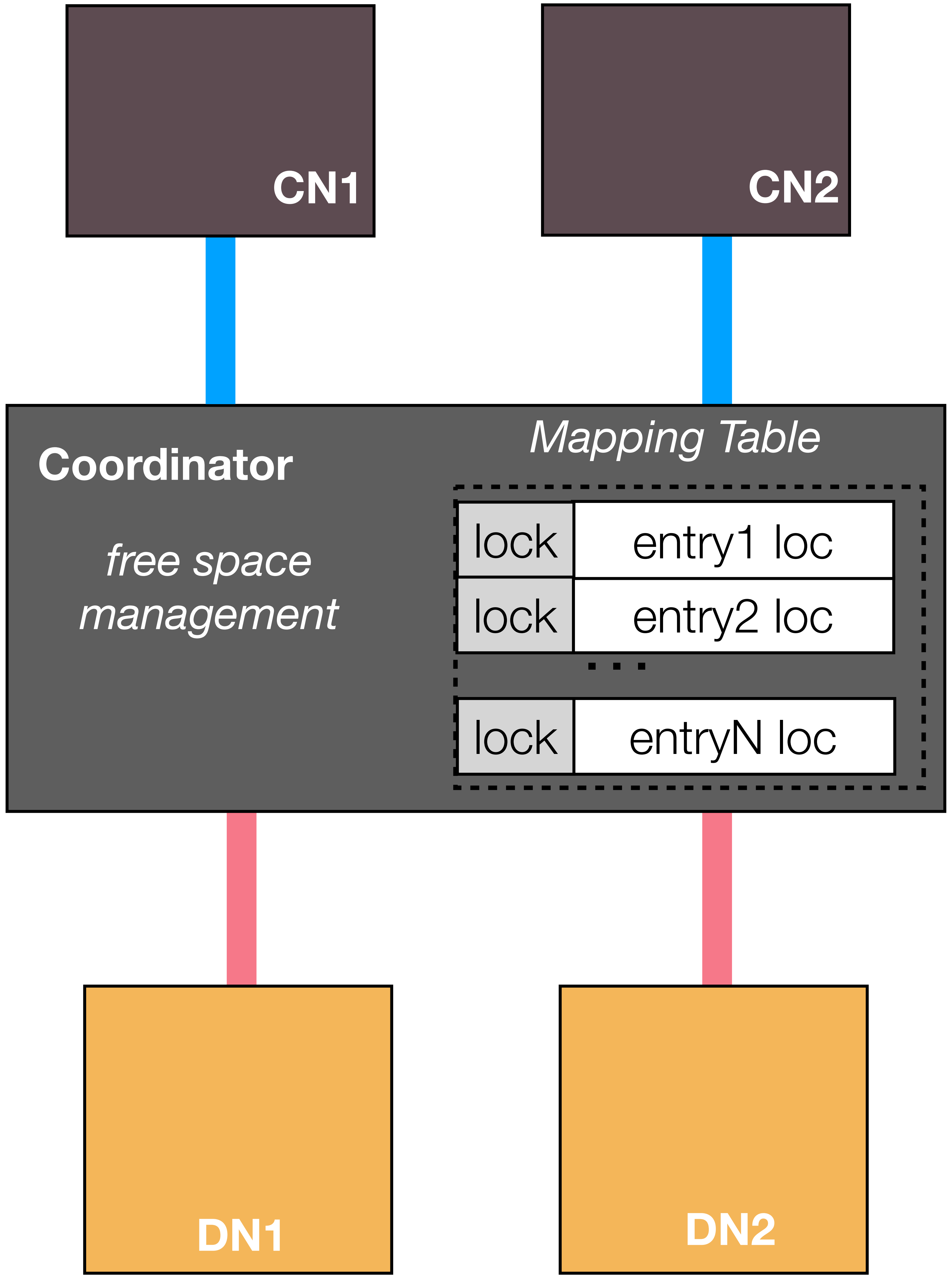


pDPM-Central



Write Flow

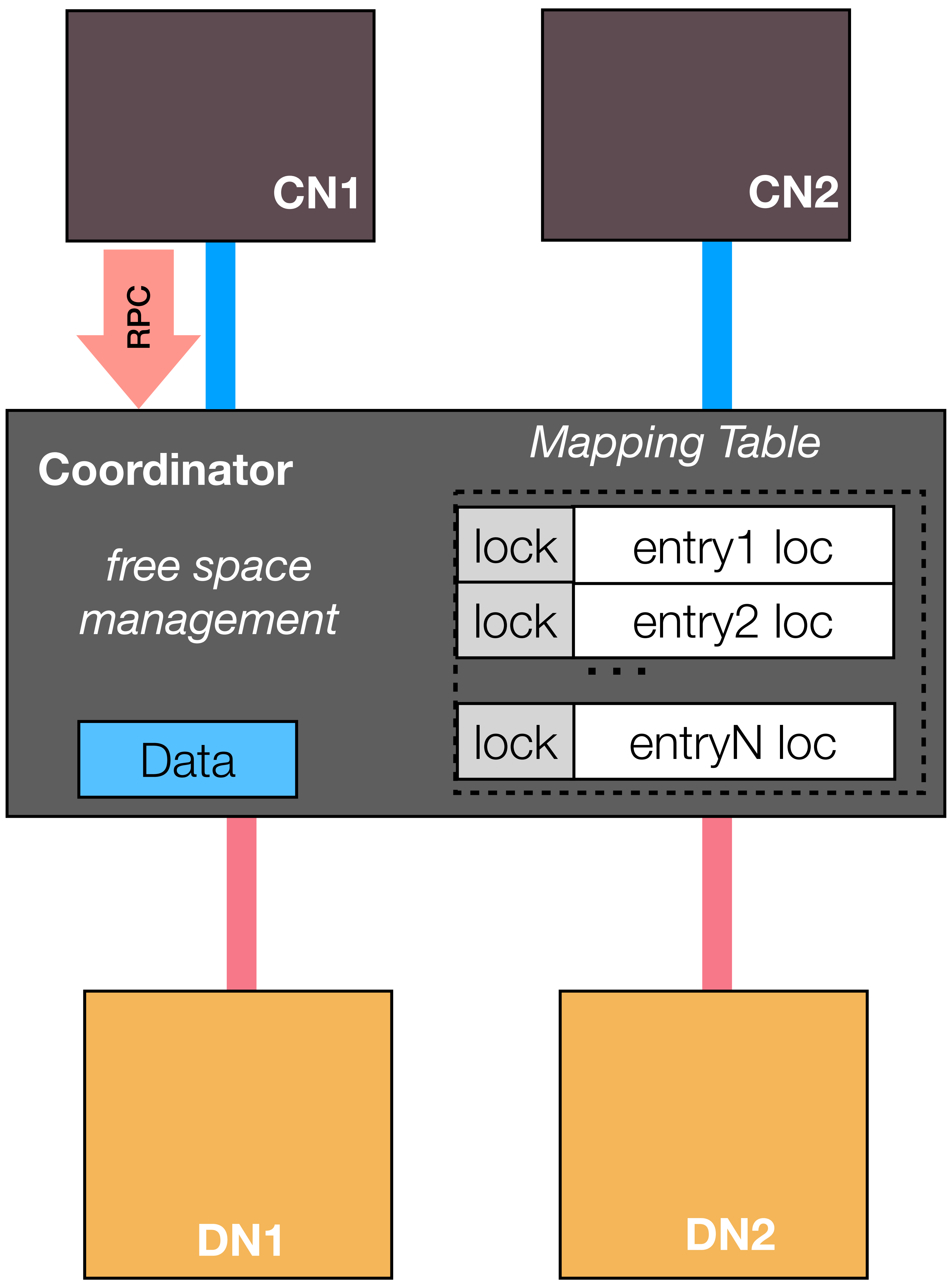
pDPM-Central



Write Flow

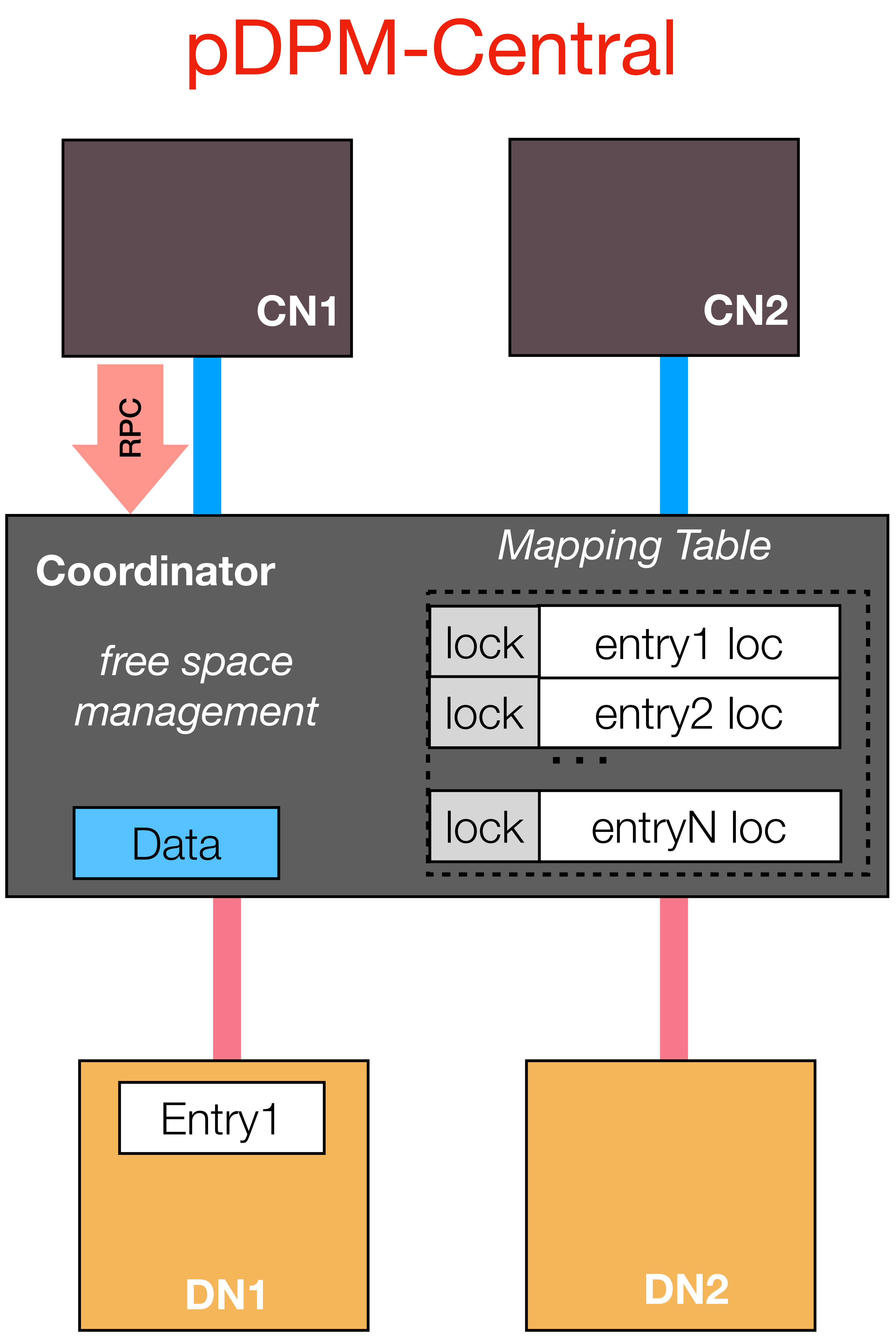
- CN sends RPC (with data) to Coordinator

pDPM-Central



Write Flow

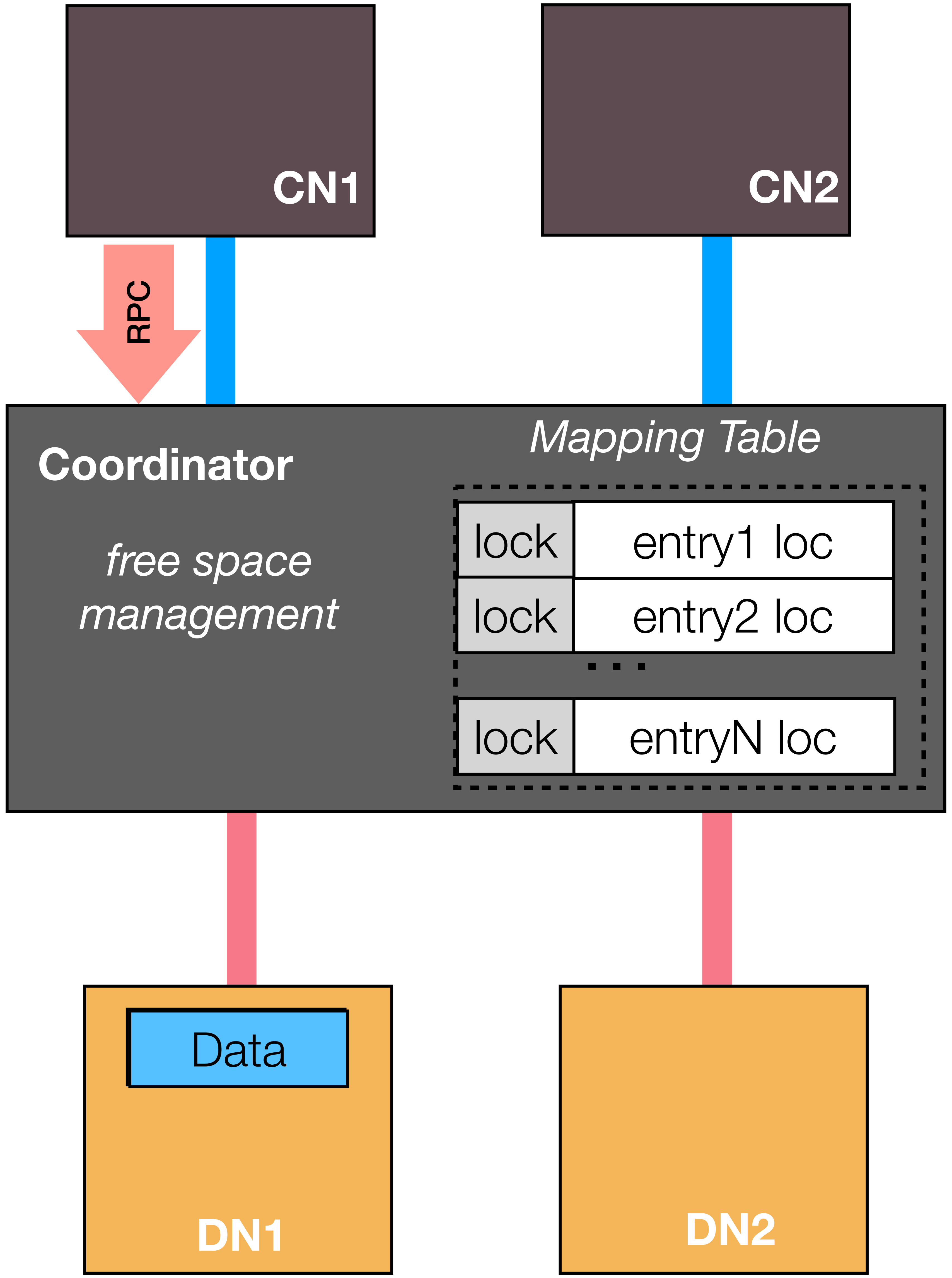
- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write



Write Flow

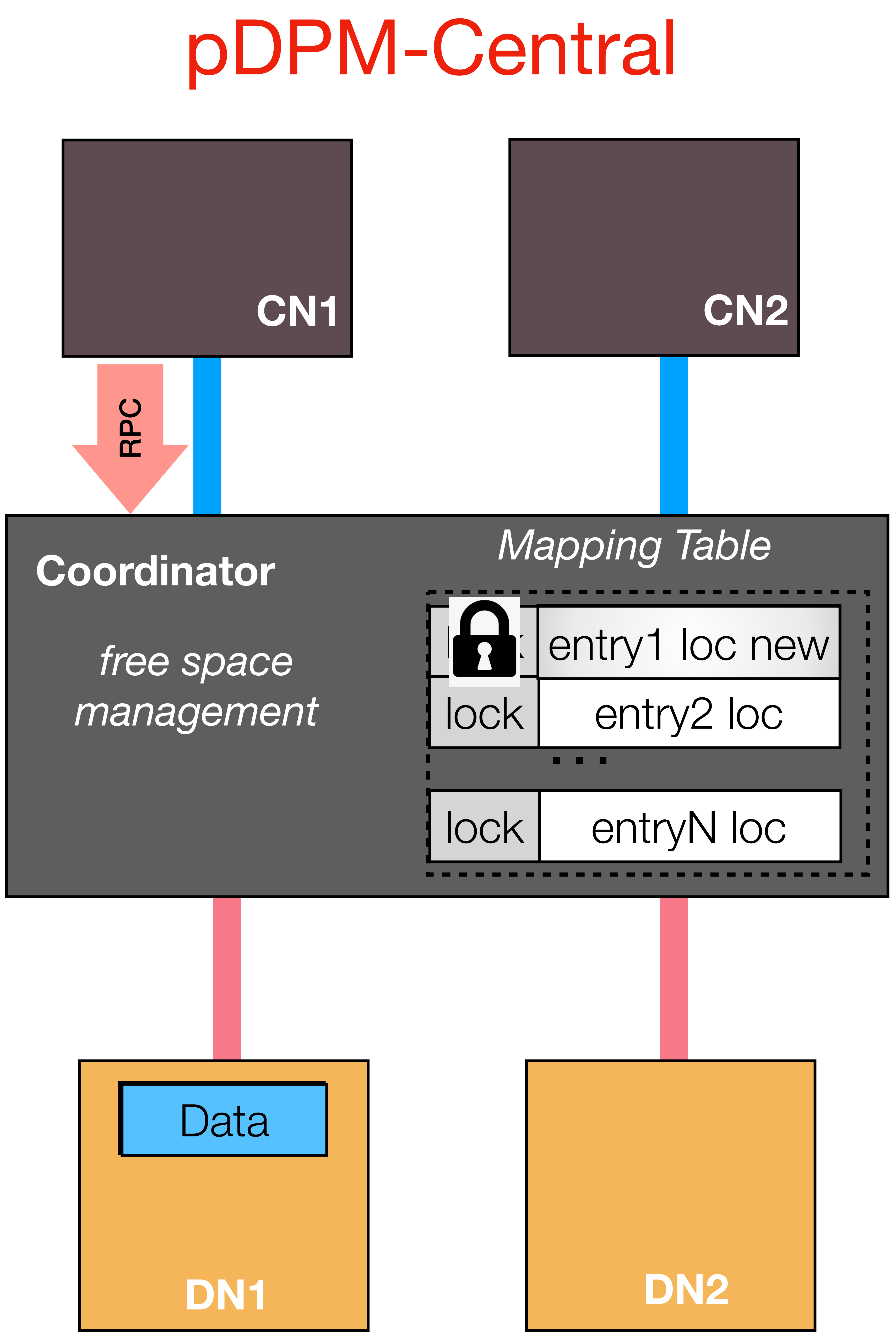
- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)

pDPM-Central



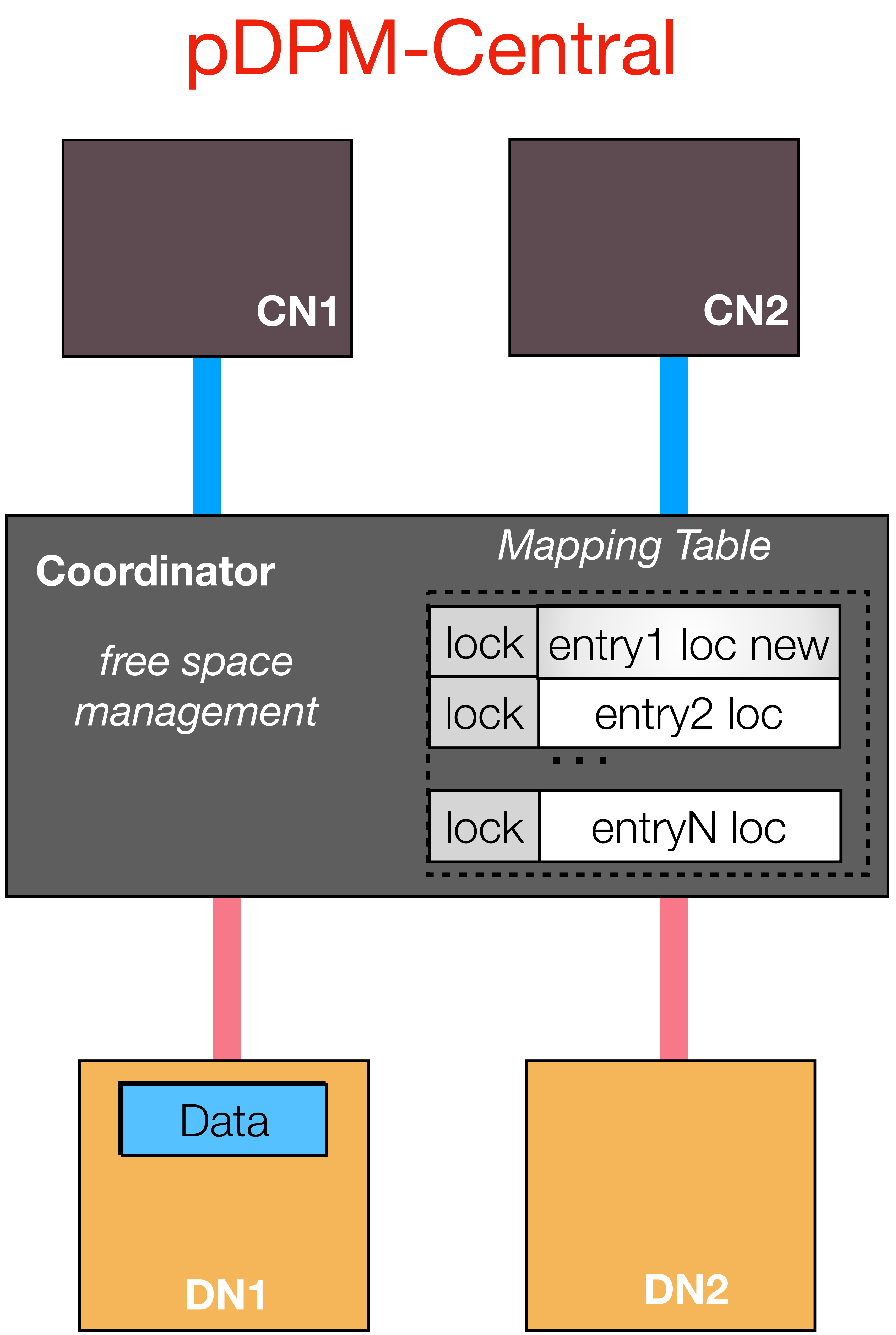
Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)



Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

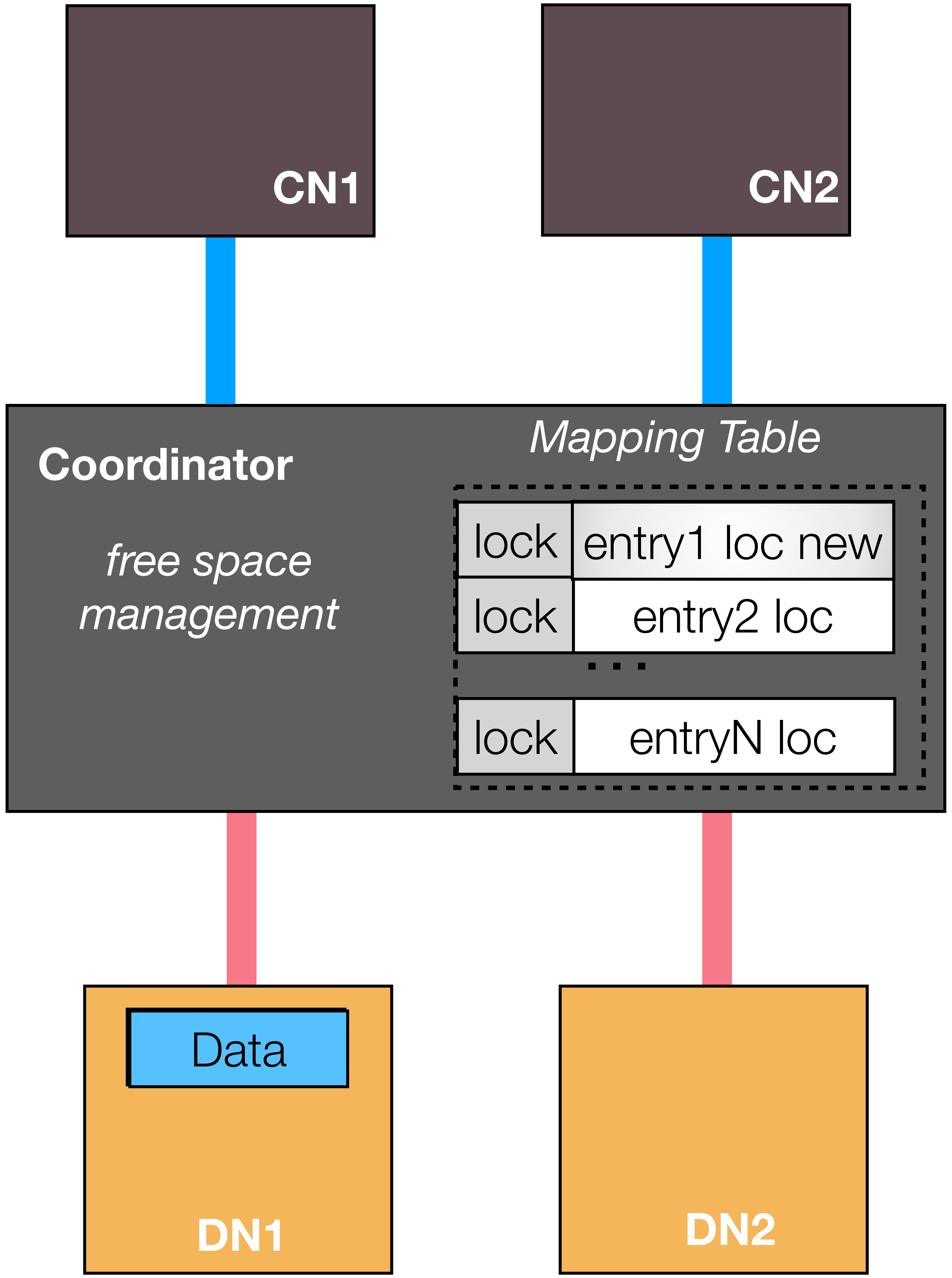


Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

Read Flow

pDPM-Central

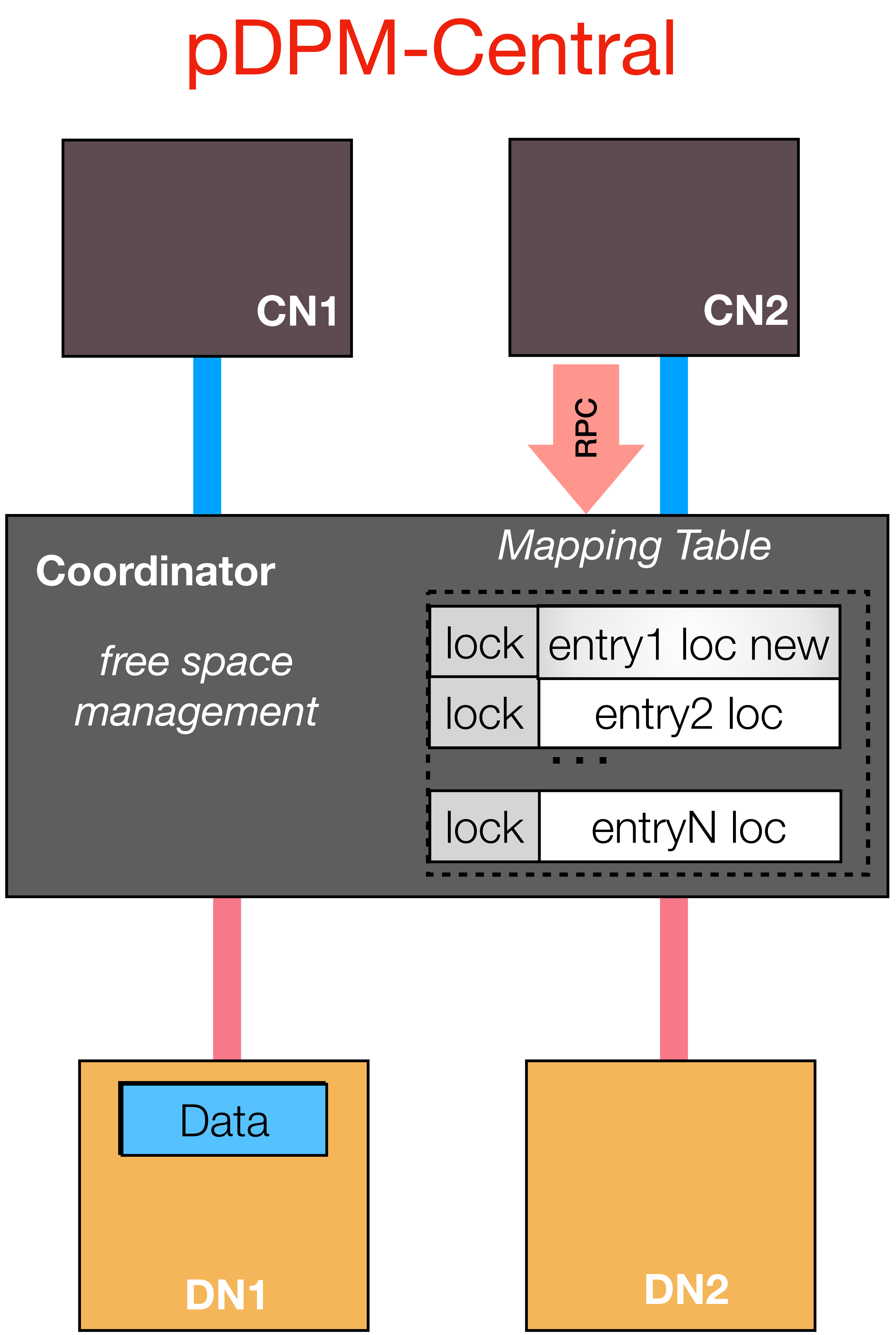


Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

Read Flow

- CN sends RPC to Coordinator



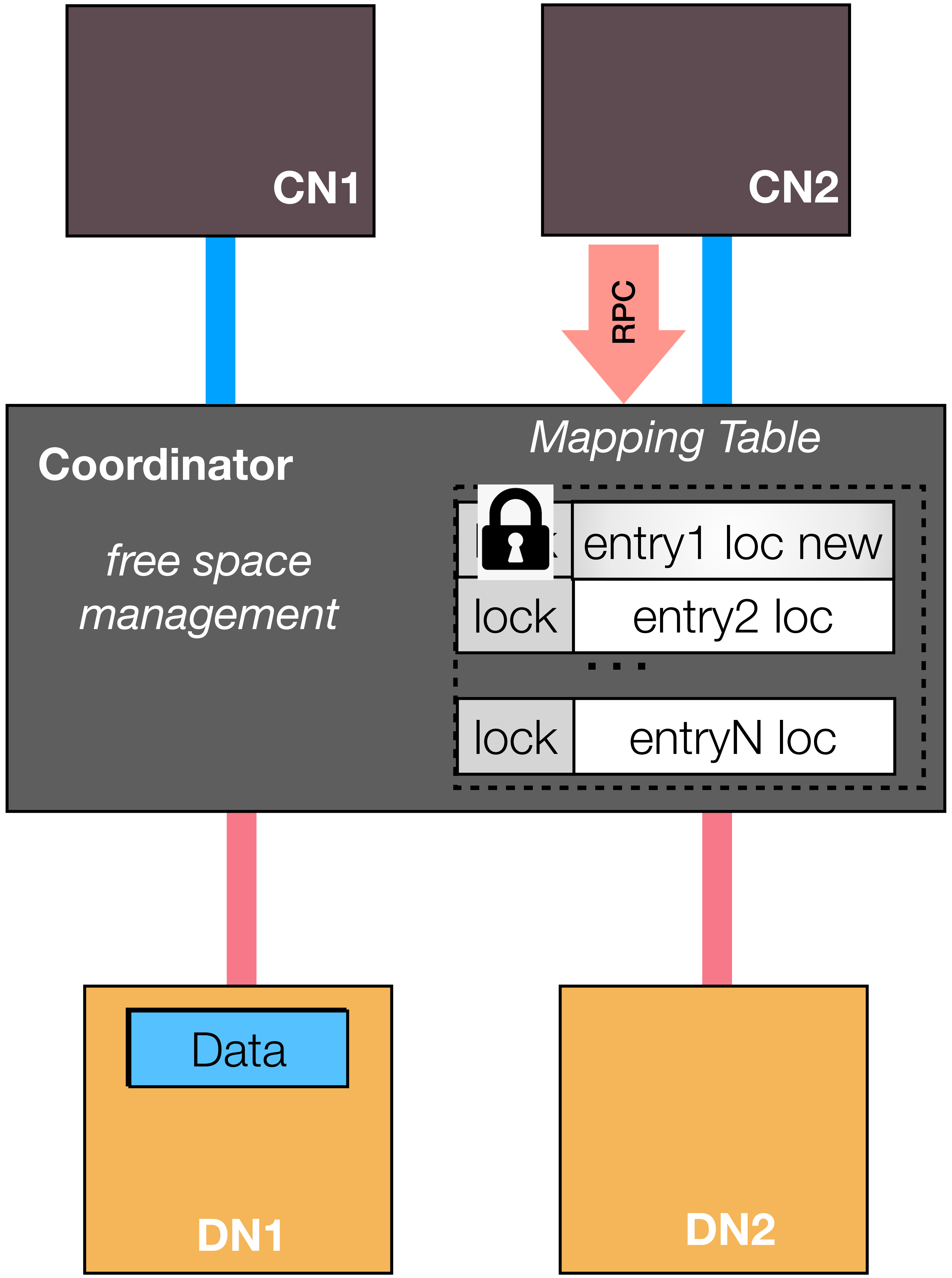
Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

Read Flow

- CN sends RPC to Coordinator
- Coordinator locks the entry in mapping table

pDPM-Central



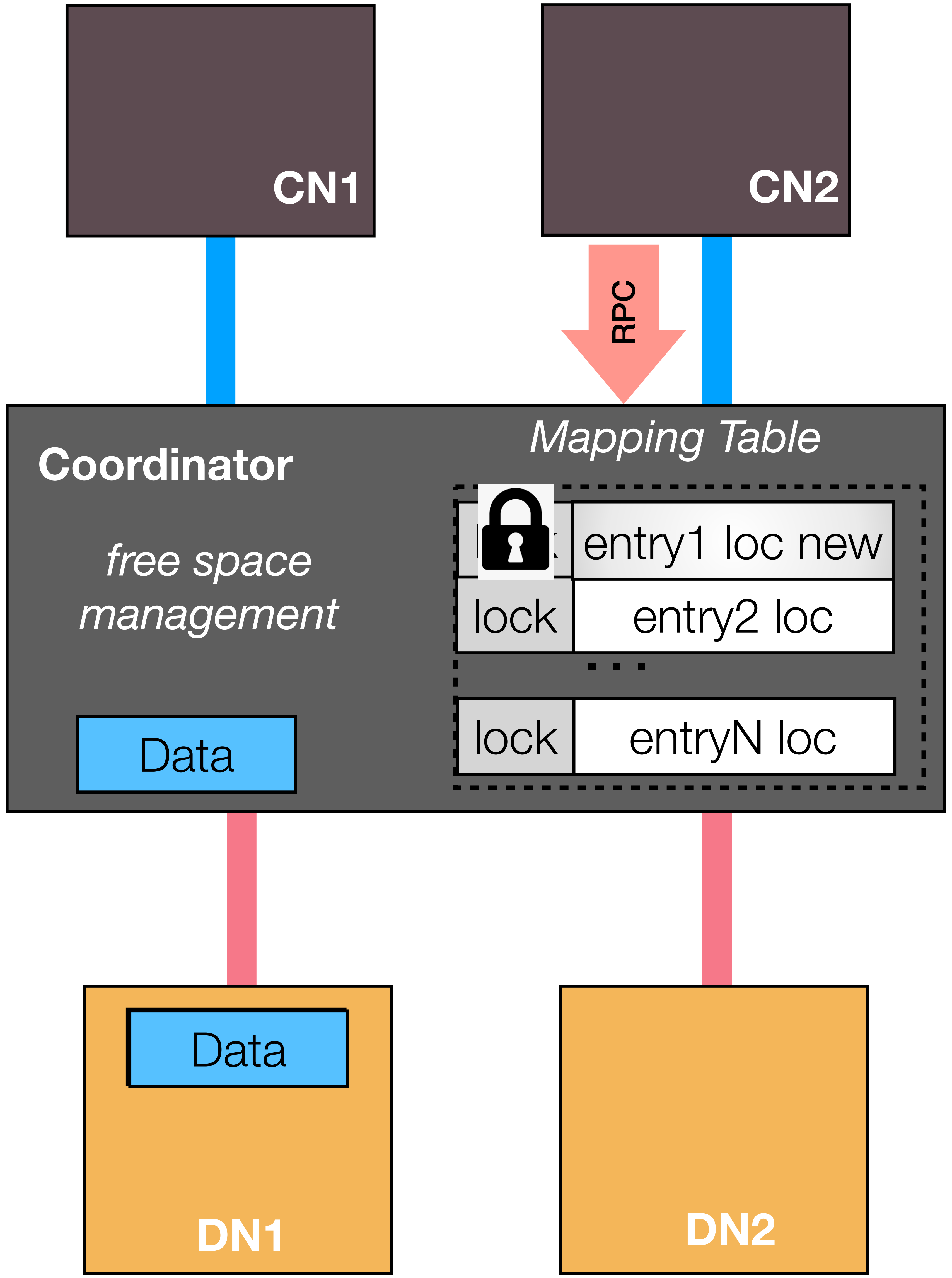
pDPM-Central

Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

Read Flow

- CN sends RPC to Coordinator
- Coordinator locks the entry in mapping table
- Coordinator reads data from DN and then replies to CN

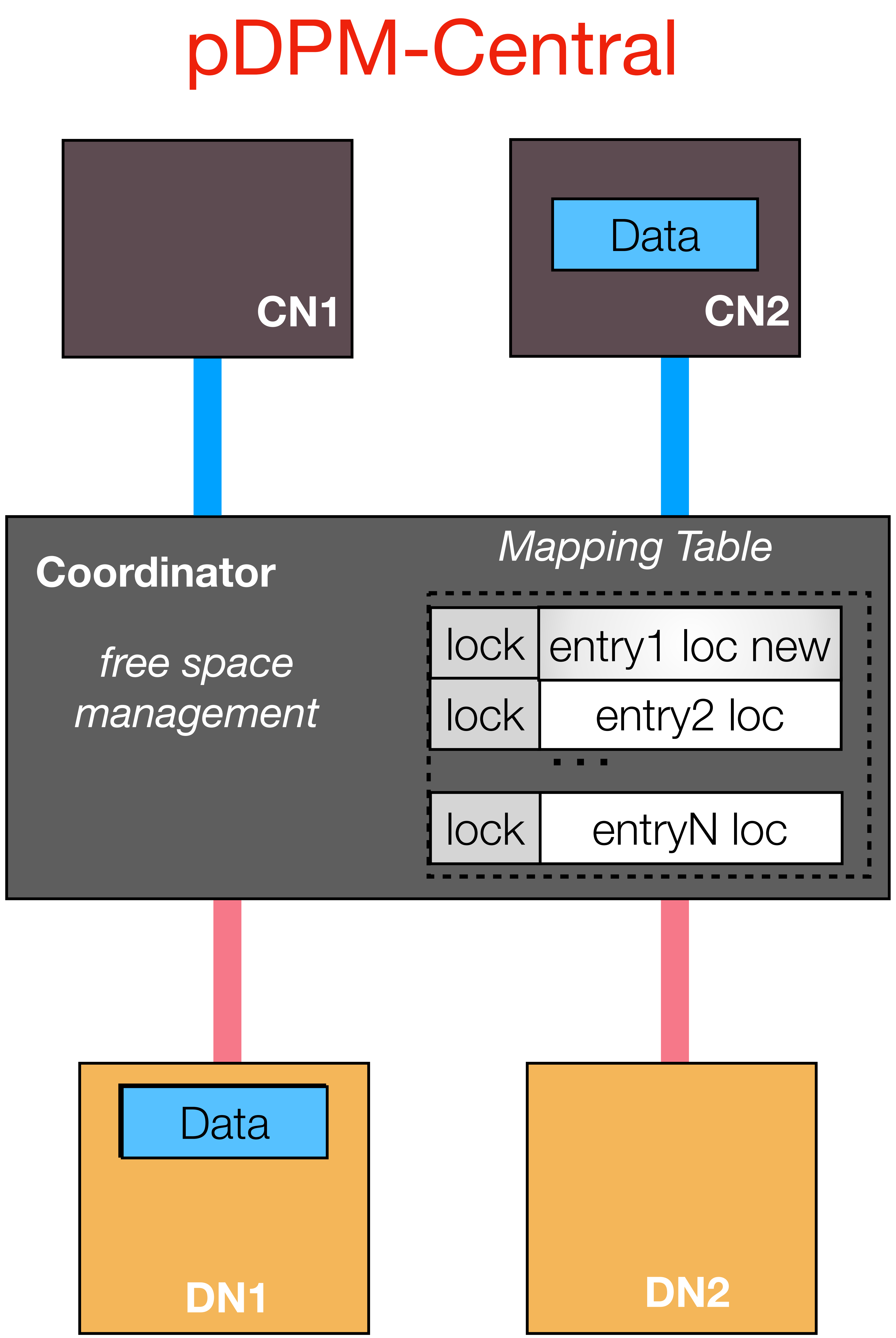


Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

Read Flow

- CN sends RPC to Coordinator
- Coordinator locks the entry in mapping table
- Coordinator reads data from DN and then replies to CN



Write Flow

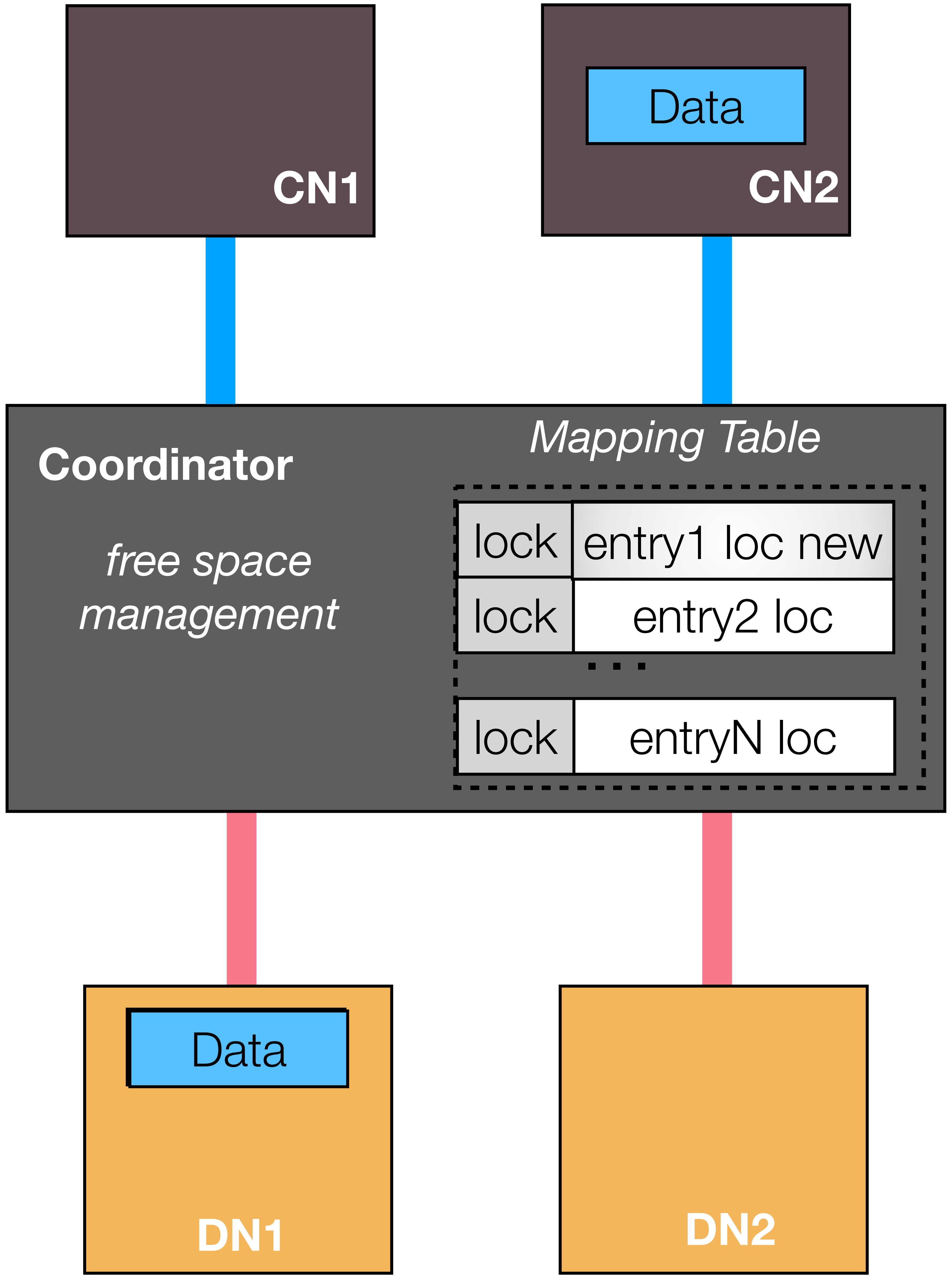
- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

Read Flow

- CN sends RPC to Coordinator
- Coordinator locks the entry in mapping table
- Coordinator reads data from DN and then replies to CN

All cases
Read: 2 RTTs
Write: 2 RTTs

pDPM-Central



Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator allocates a new space for the write
- Coordinator writes data to it (as redo-copy)
- Coordinator updates its local map table (with a local lock)

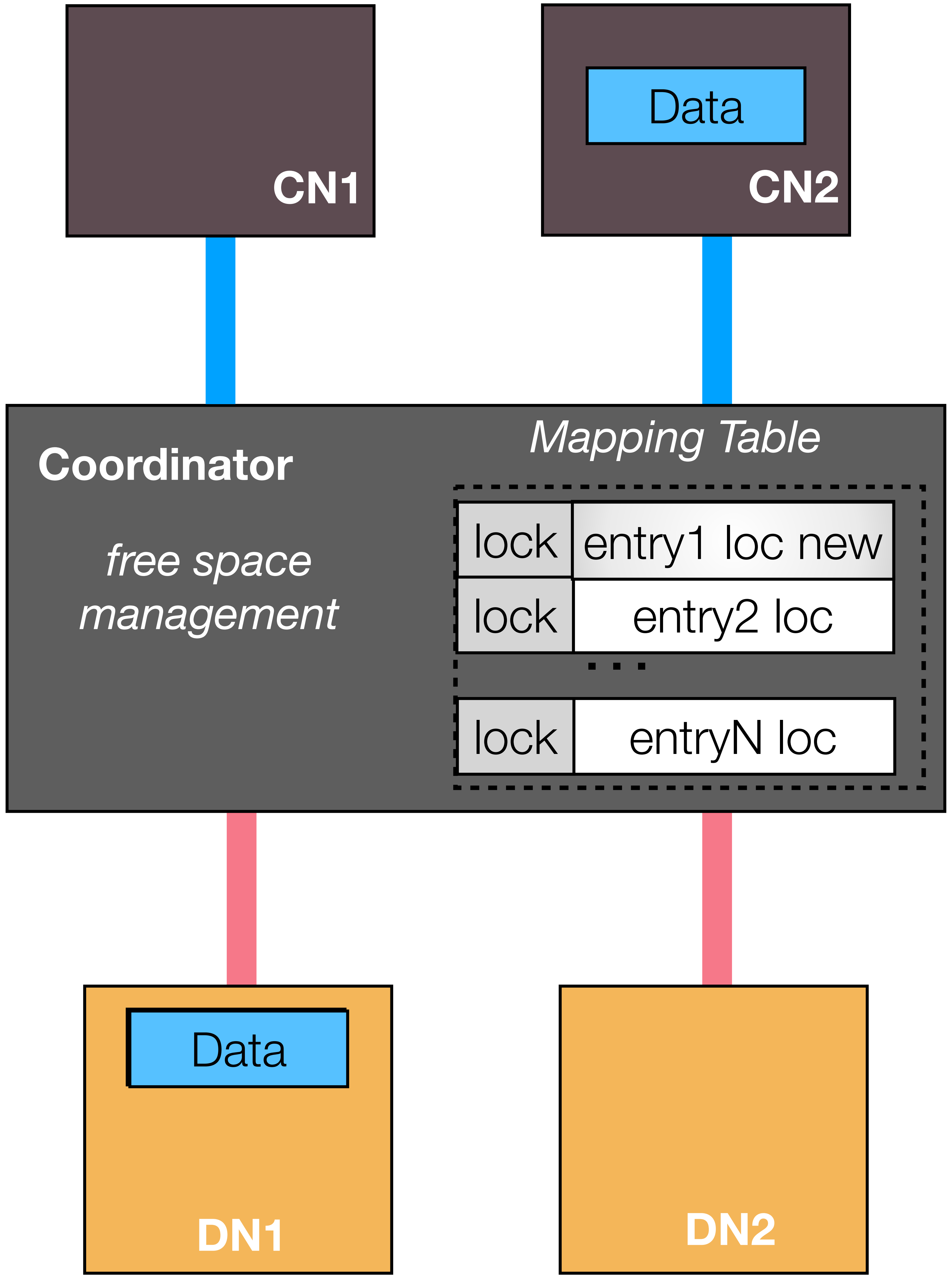
Read Flow

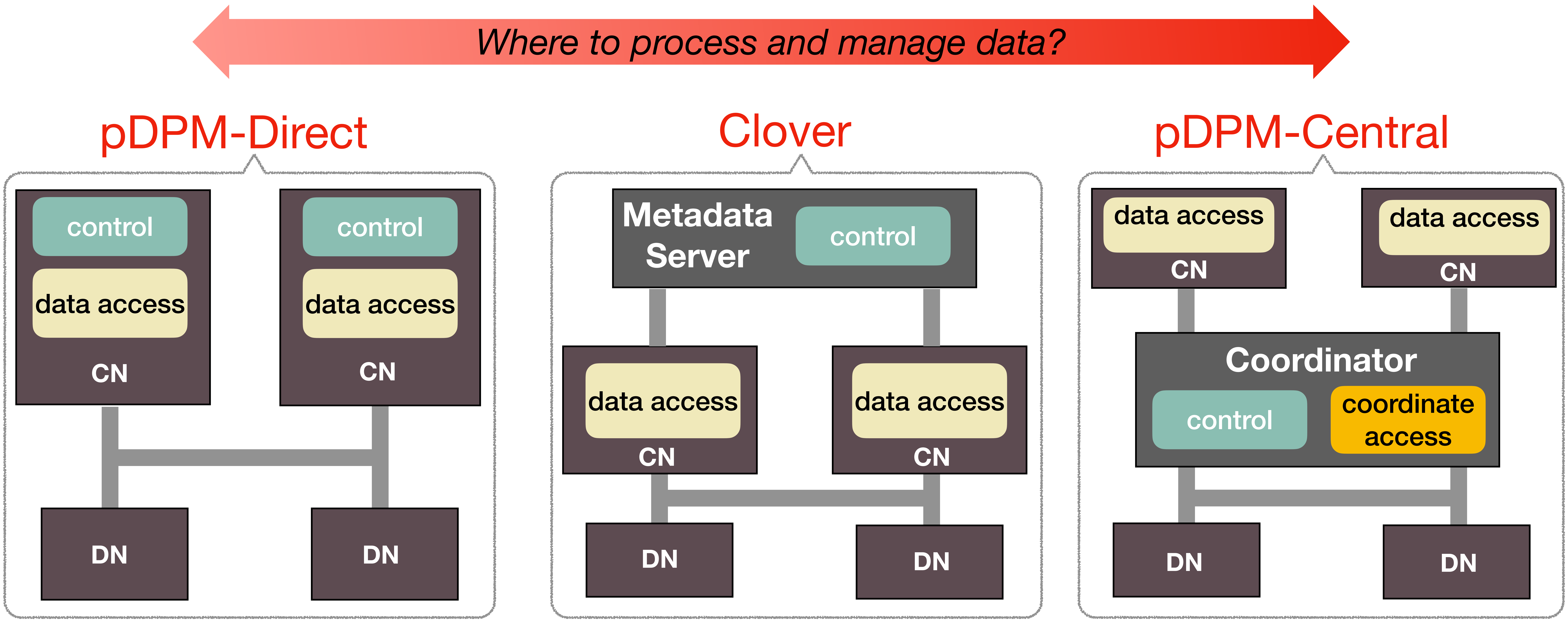
- CN sends RPC to Coordinator
- Coordinator locks the entry in mapping table
- Coordinator reads data from DN and then replies to CN

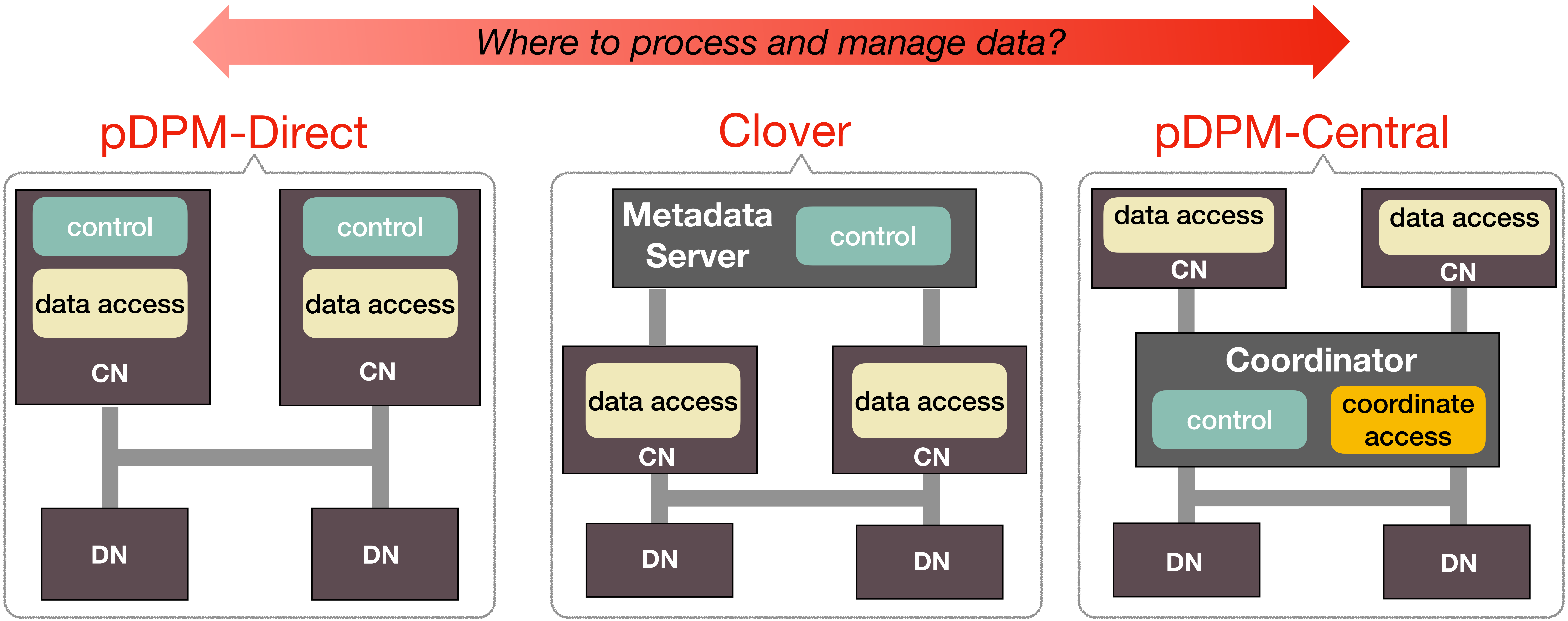
All cases
Read: 2 RTTs
Write: 2 RTTs

☹ **Slower read**
Poor scalability: coordinator is the bottleneck

pDPM-Central

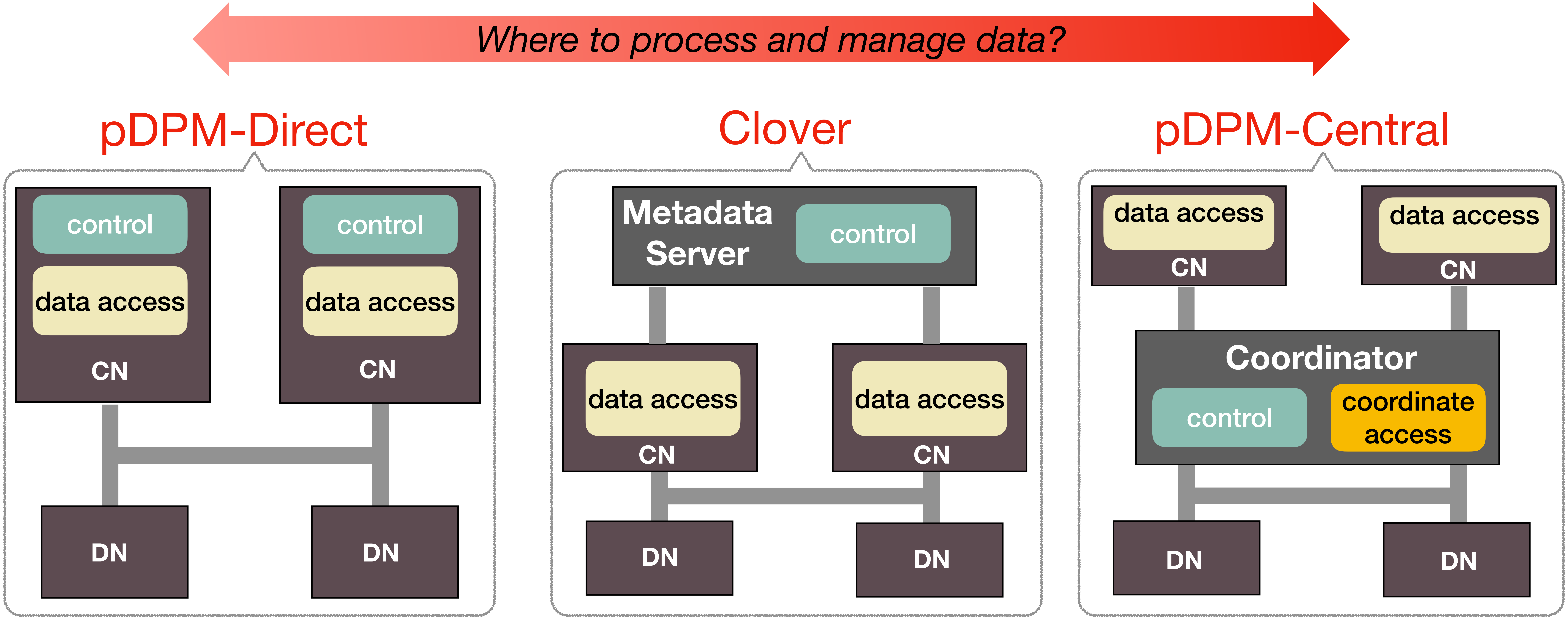



Where to process and manage data?
pDPM-Direct
control
data access
CN
control
data access
CN
DN
DN
Clover
Metadata Server
control
data access
CN
data access
CN
DN
DN
pDPM-Central
data access
CN
data access
CN
Coordinator
control
coordinate access
DN
DN



- Slow write
- Slow for large data

- Extra read RTTs
- Coordinator cannot scale

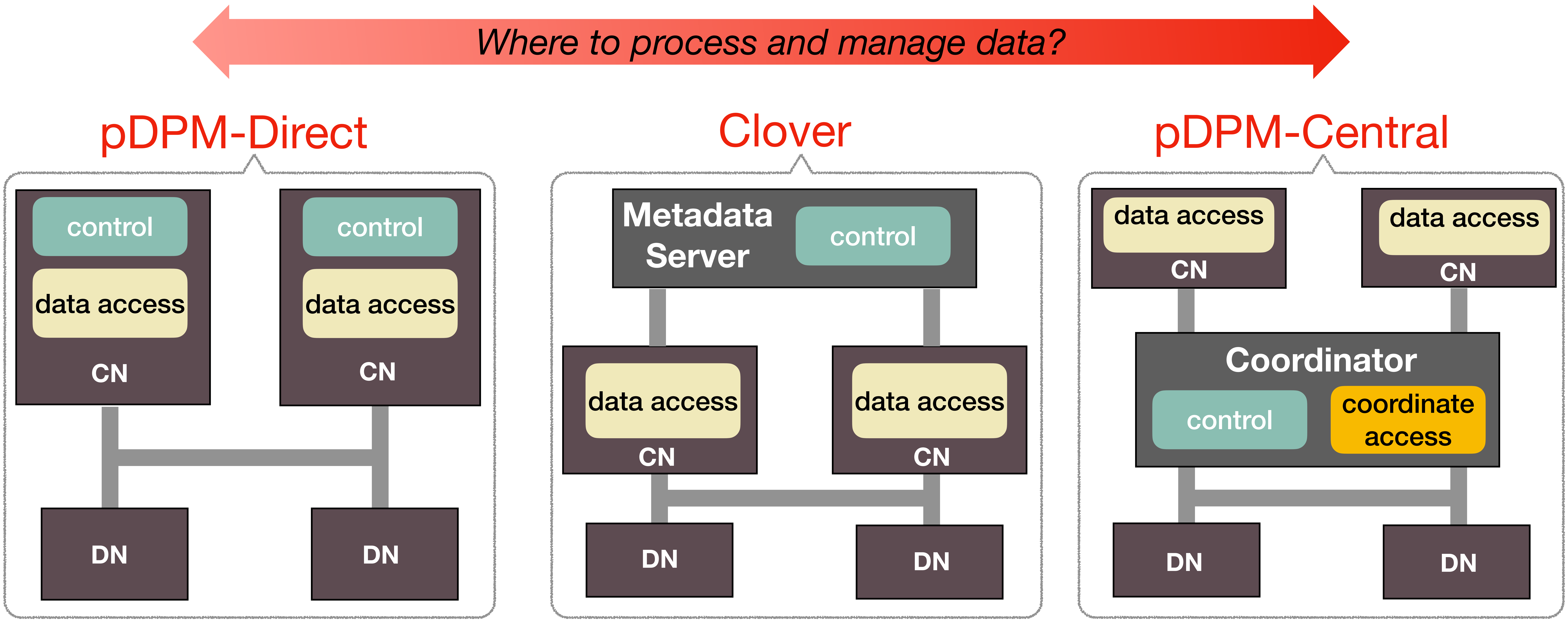


- Slow write
- Slow for large data

- Extra read RTTs
- Coordinator cannot scale

Distributed data & metadata planes

Centralized data & metadata planes



- Slow write
- Slow for large data

Distributed data & metadata planes

Separate data & metadata planes

- Extra read RTTs
- Coordinator cannot scale

Centralized data & metadata planes

- Slow write
- Slow for large data

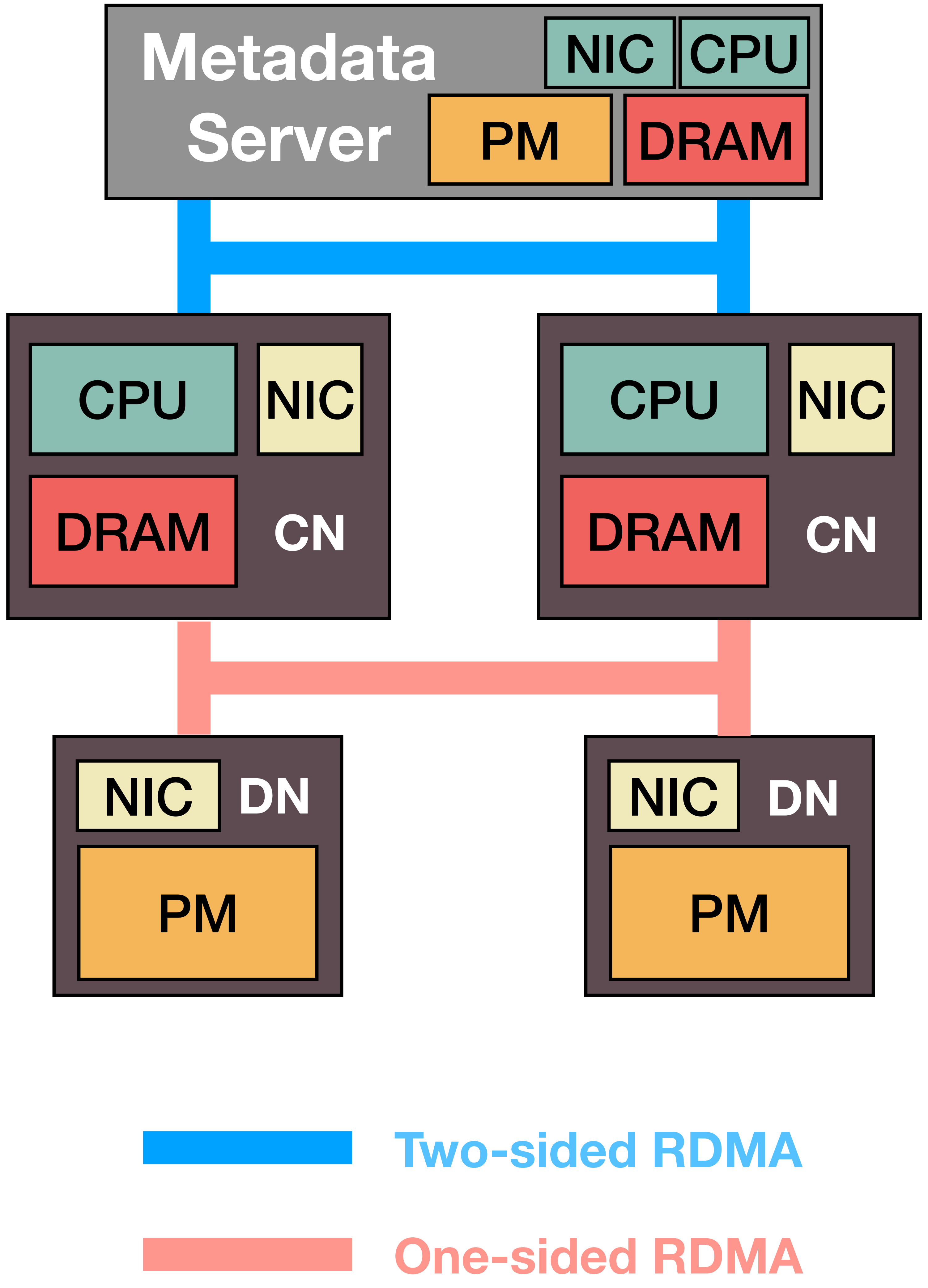
Distributed data & metadata planes

Separate data & metadata planes

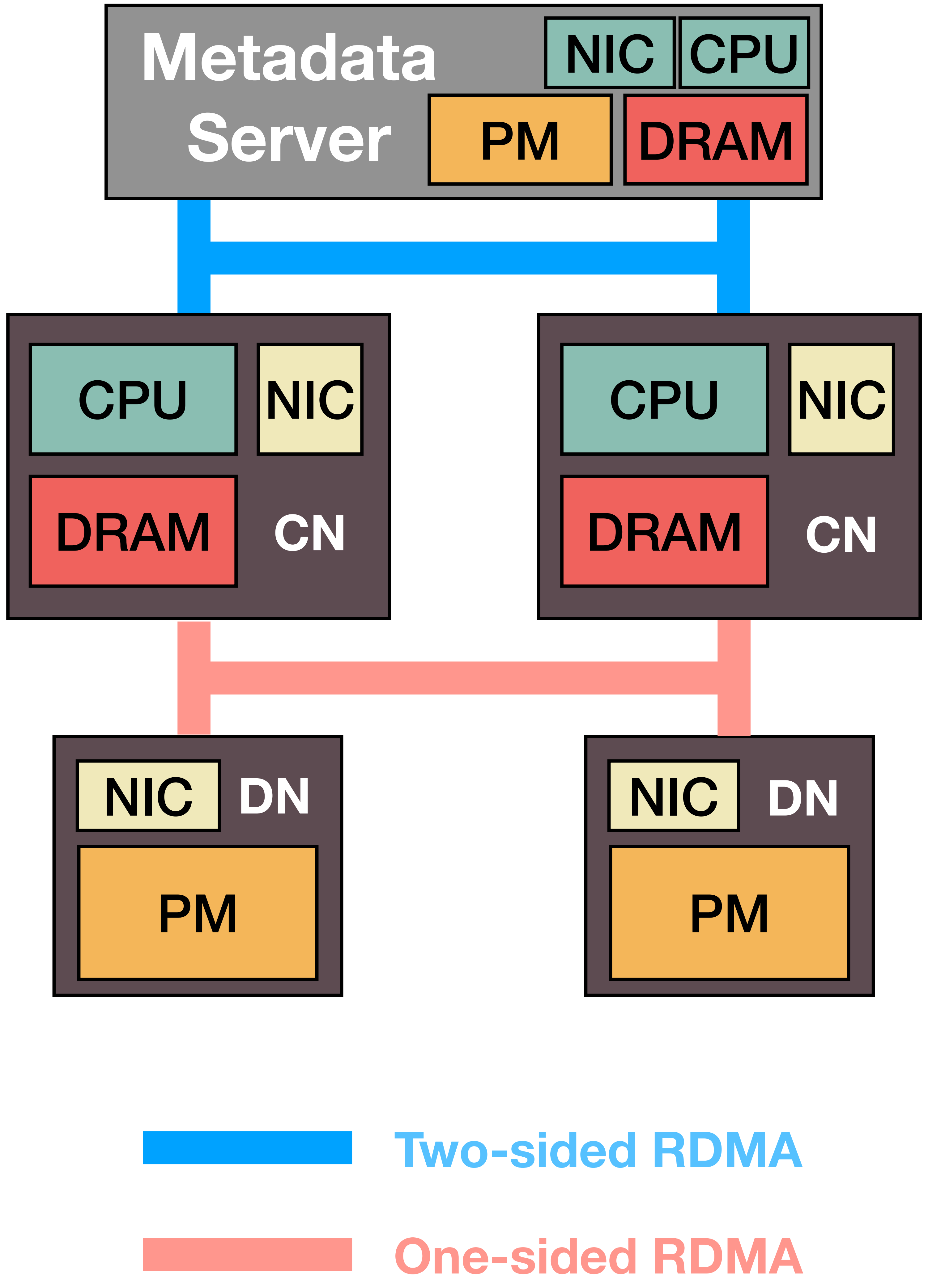
- Extra read RTTs
- Coordinator cannot scale

Centralized data & metadata planes

Clover: Combining Distributed and Centralized Approaches



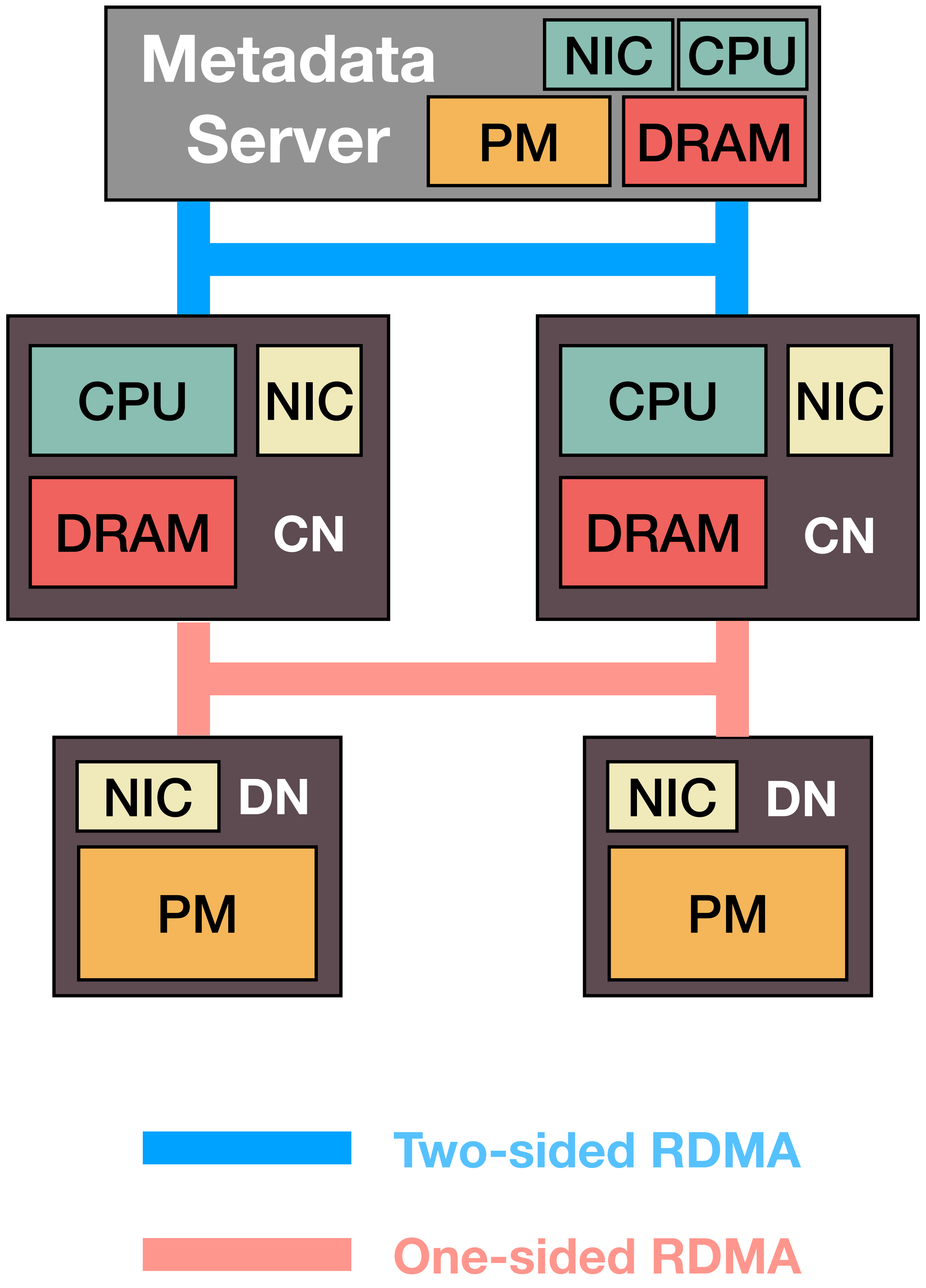
Clover: Combining Distributed and Centralized Approaches



High-level idea: separate data and metadata plane

- Separate locations
- Different communication methods
- Different management strategy

Clover: Combining Distributed and Centralized Approaches



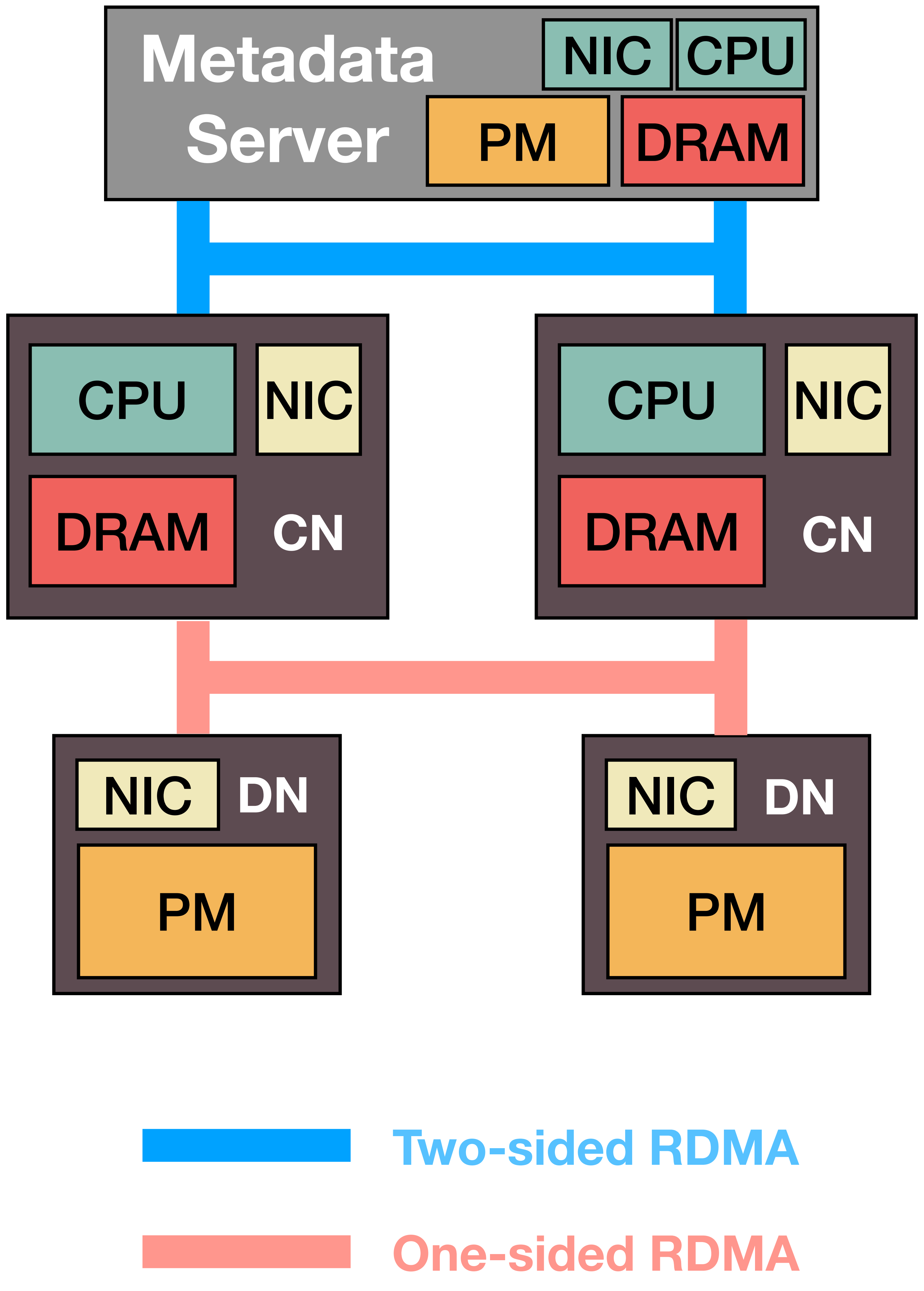
High-level idea: separate data and metadata plane

- Separate locations
- Different communication methods
- Different management strategy

Data Plane

- `CNs` directly access `DNs` with one-sided RDMA

Clover: Combining Distributed and Centralized Approaches



High-level idea: separate data and metadata plane

- Separate locations
- Different communication methods
- Different management strategy

Data Plane

- **`CNs`** directly access **`DNs`** with one-sided RDMA

Metadata Plane

- **`CNs`** talk to metadata server (**`MS`**) with two-sided RDMA

Main Challenge in Data Plane:

How to efficiently support concurrent data accesses from CNs to DNs?

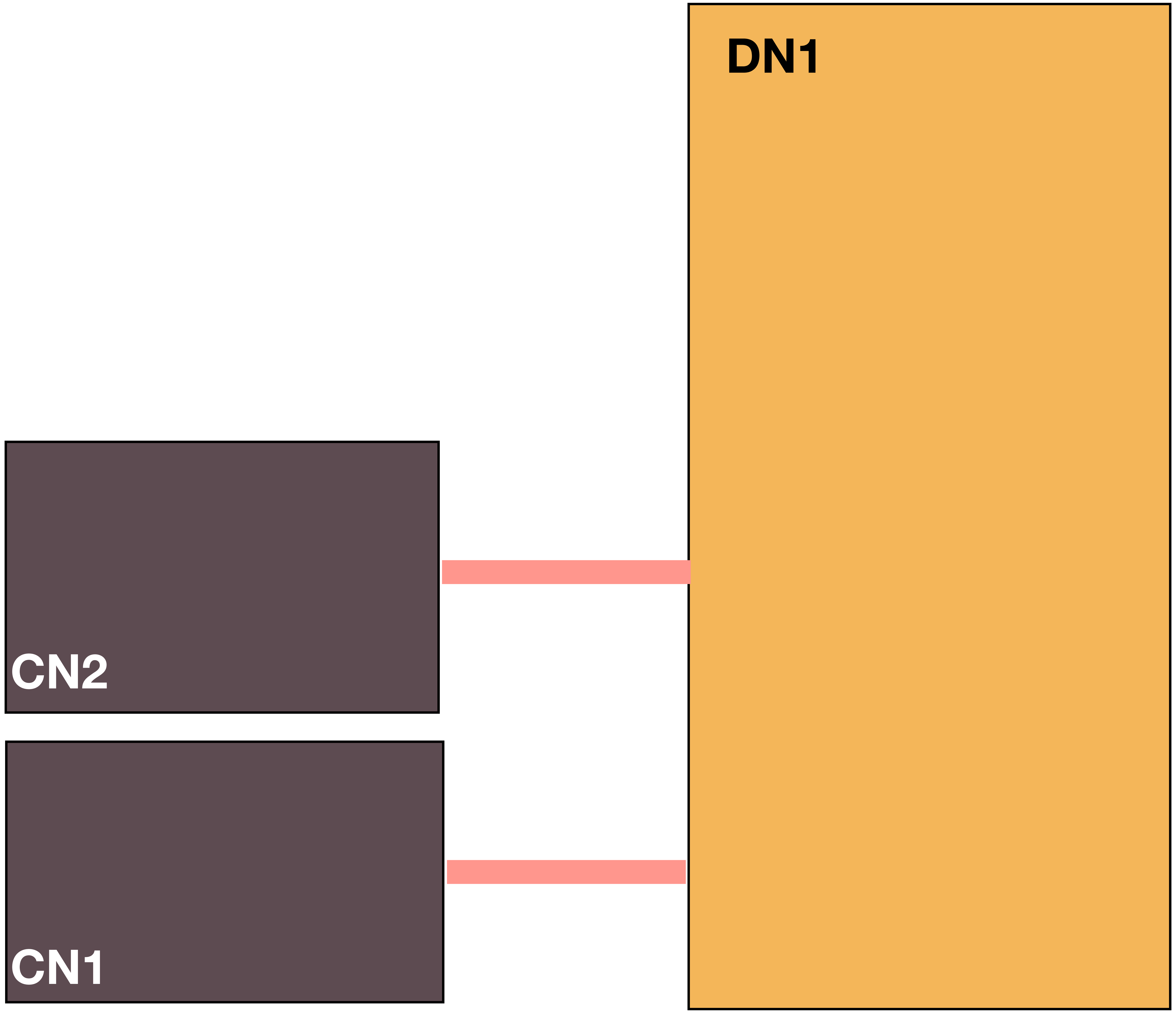
Main Challenge in Data Plane:

How to efficiently support concurrent data accesses from CNs to DNs?

Our Approaches:

- Lock-free data structures for un-orchestrated concurrent accesses
- Optimizations to further reduce read/write RTTs

➡ Guarantees ***read committed*** and ***atomic write***



Lock-free data structures

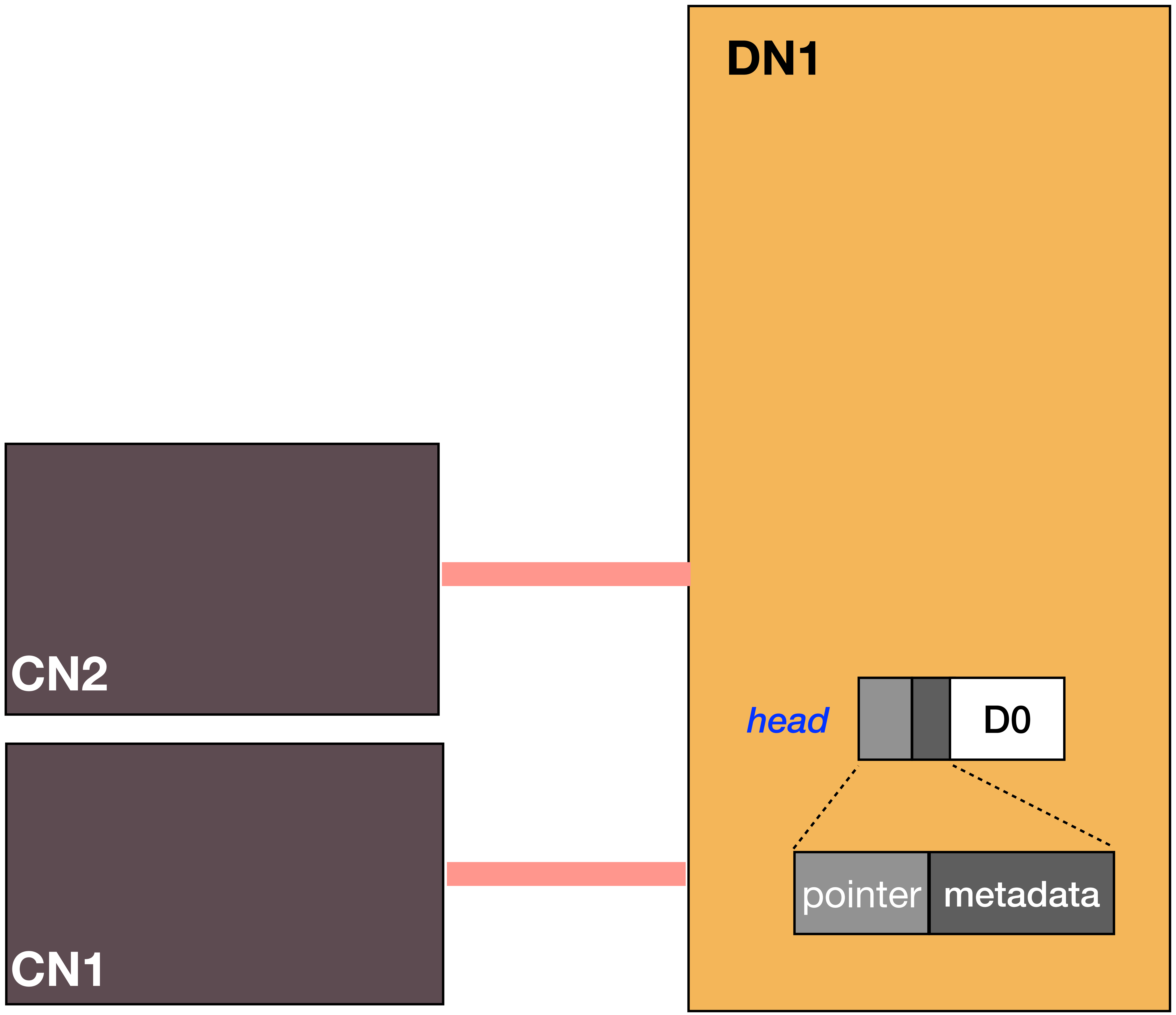
Chained redo copies (versions) at `DNs`

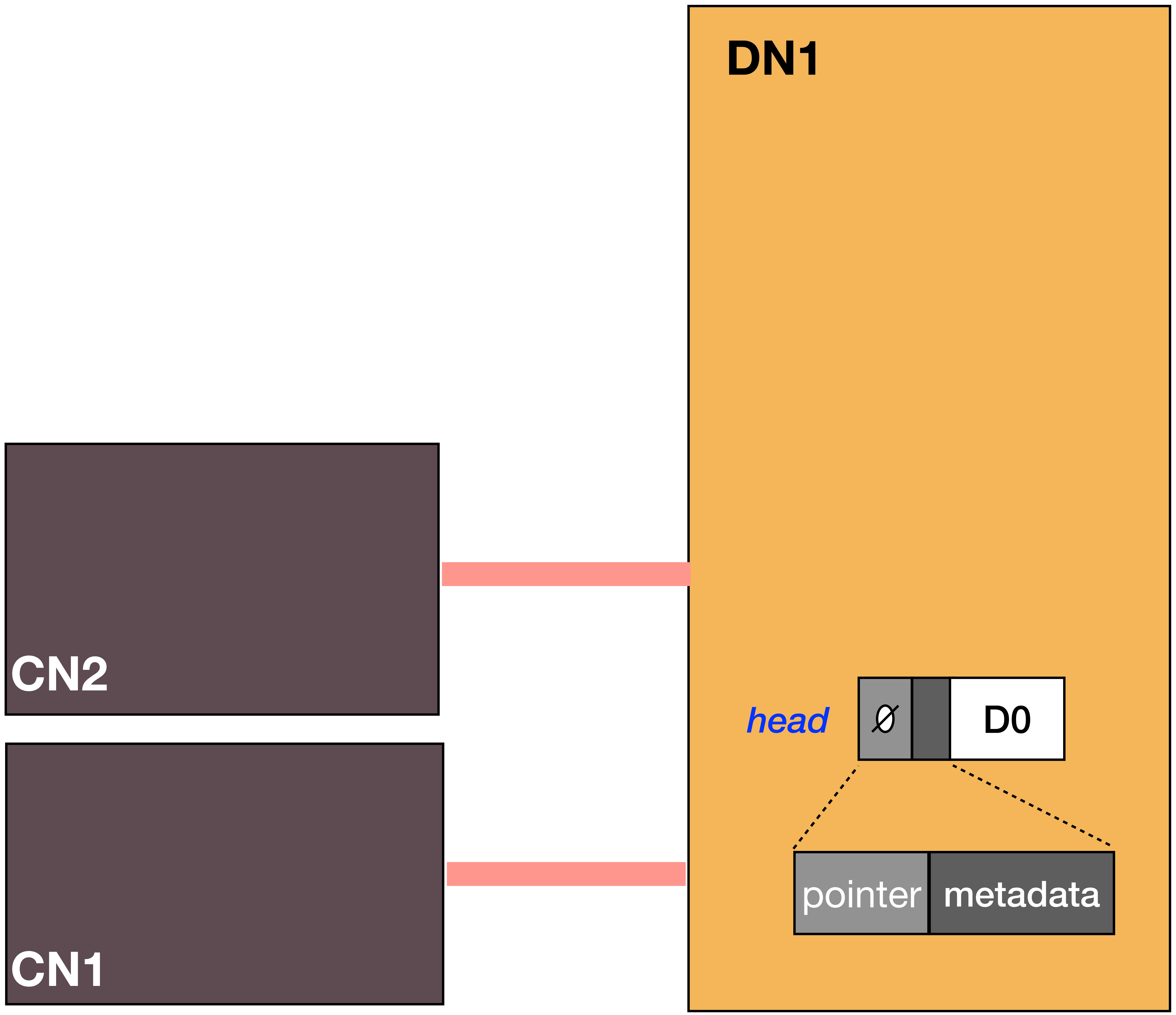
`CNs` cache a *cursor* that points to a version

Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version





Lock-free data structures

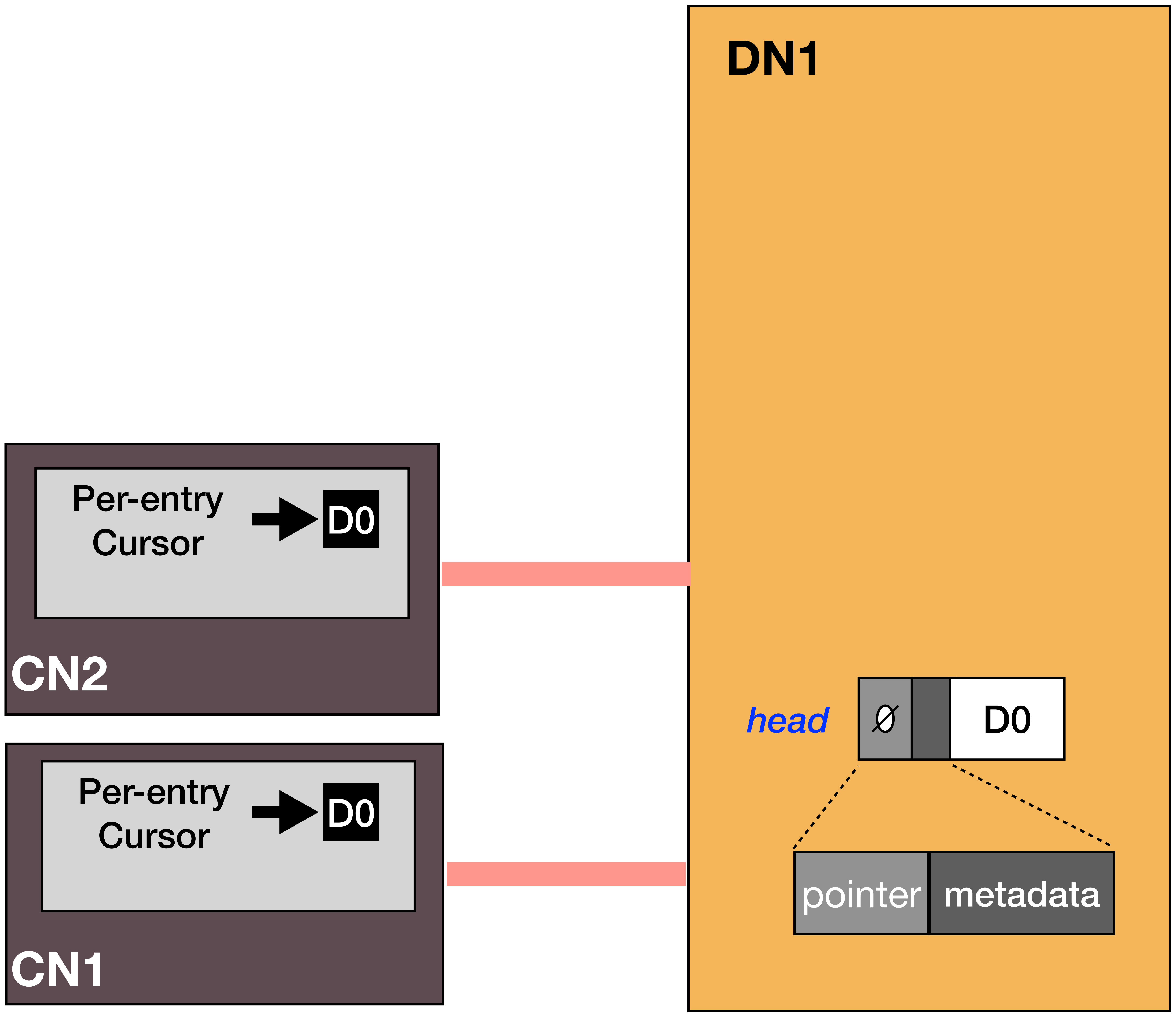
Chained redo copies (versions) at `DNs`

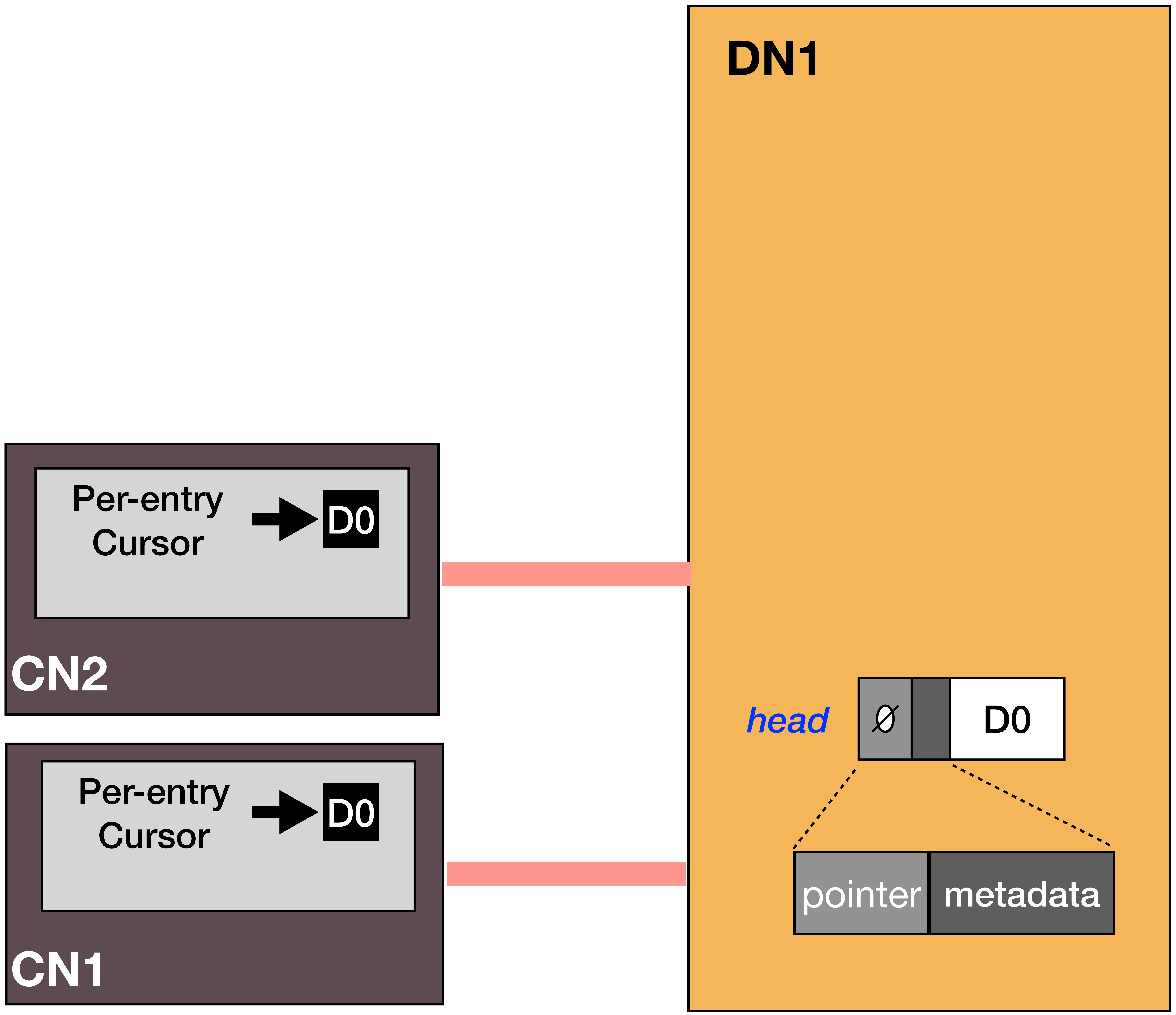
`CNs` cache a ***cursor*** that points to a version

Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a *cursor* that points to a version





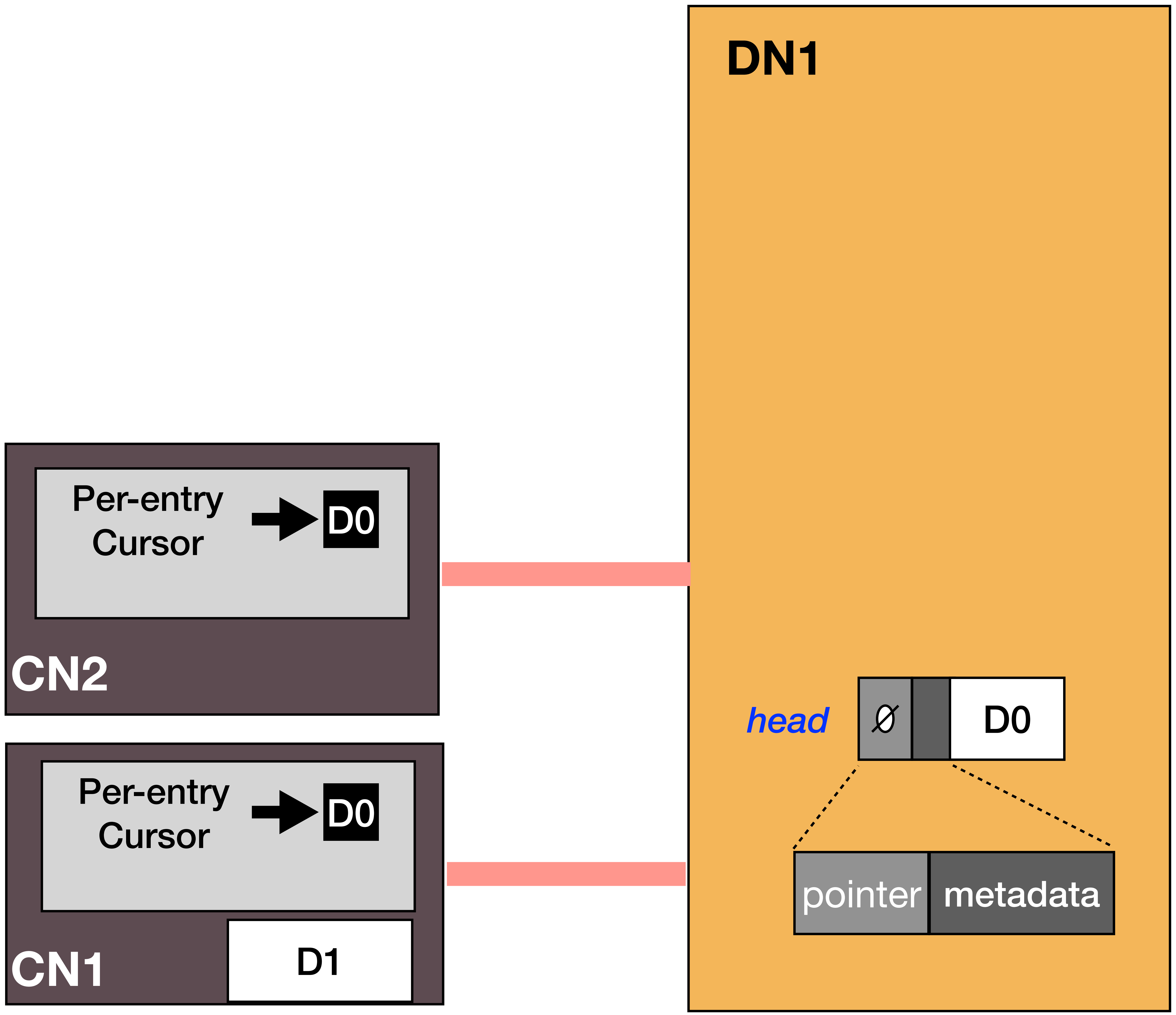
Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry



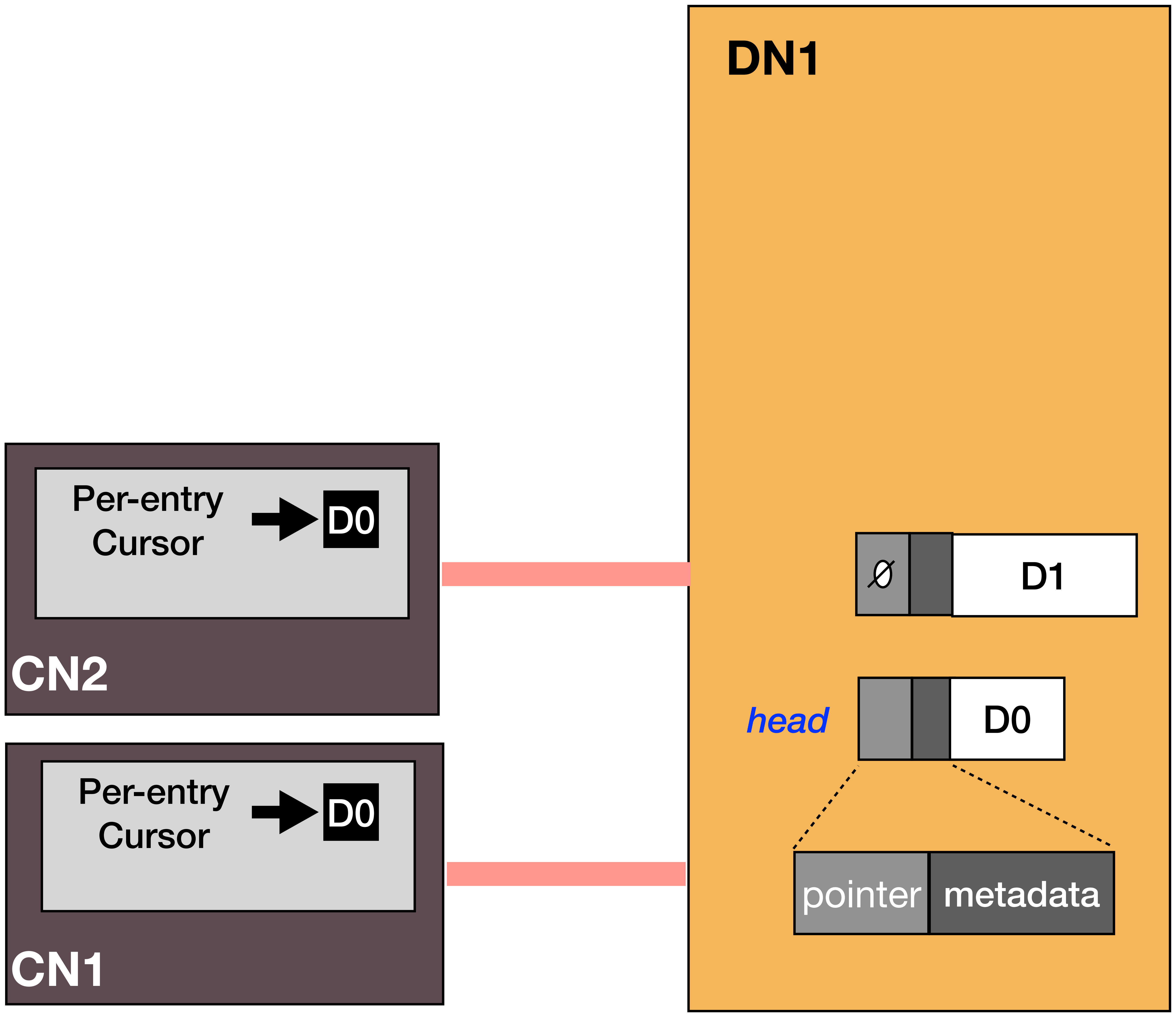
Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry



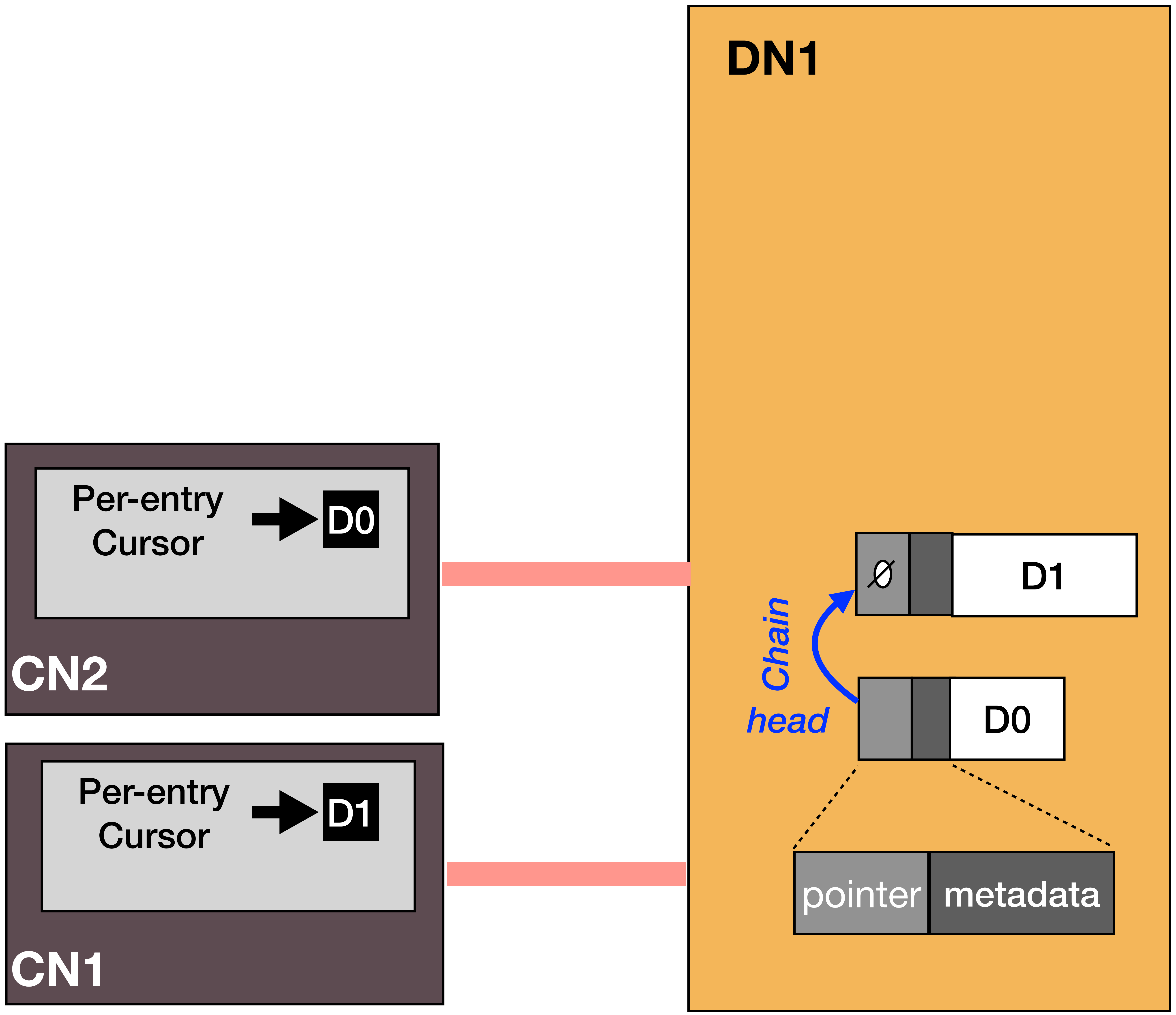
Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry



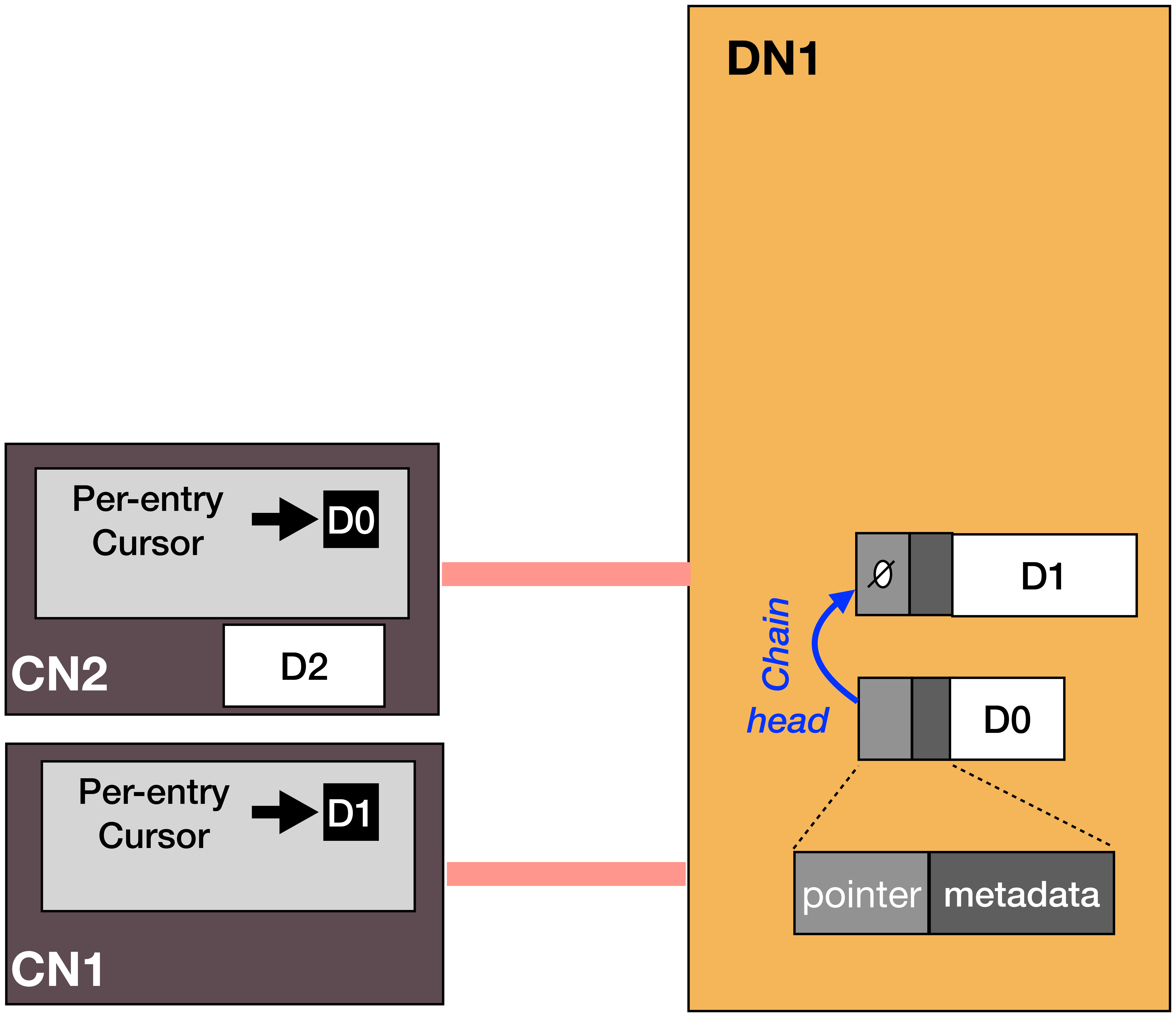
Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry



Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

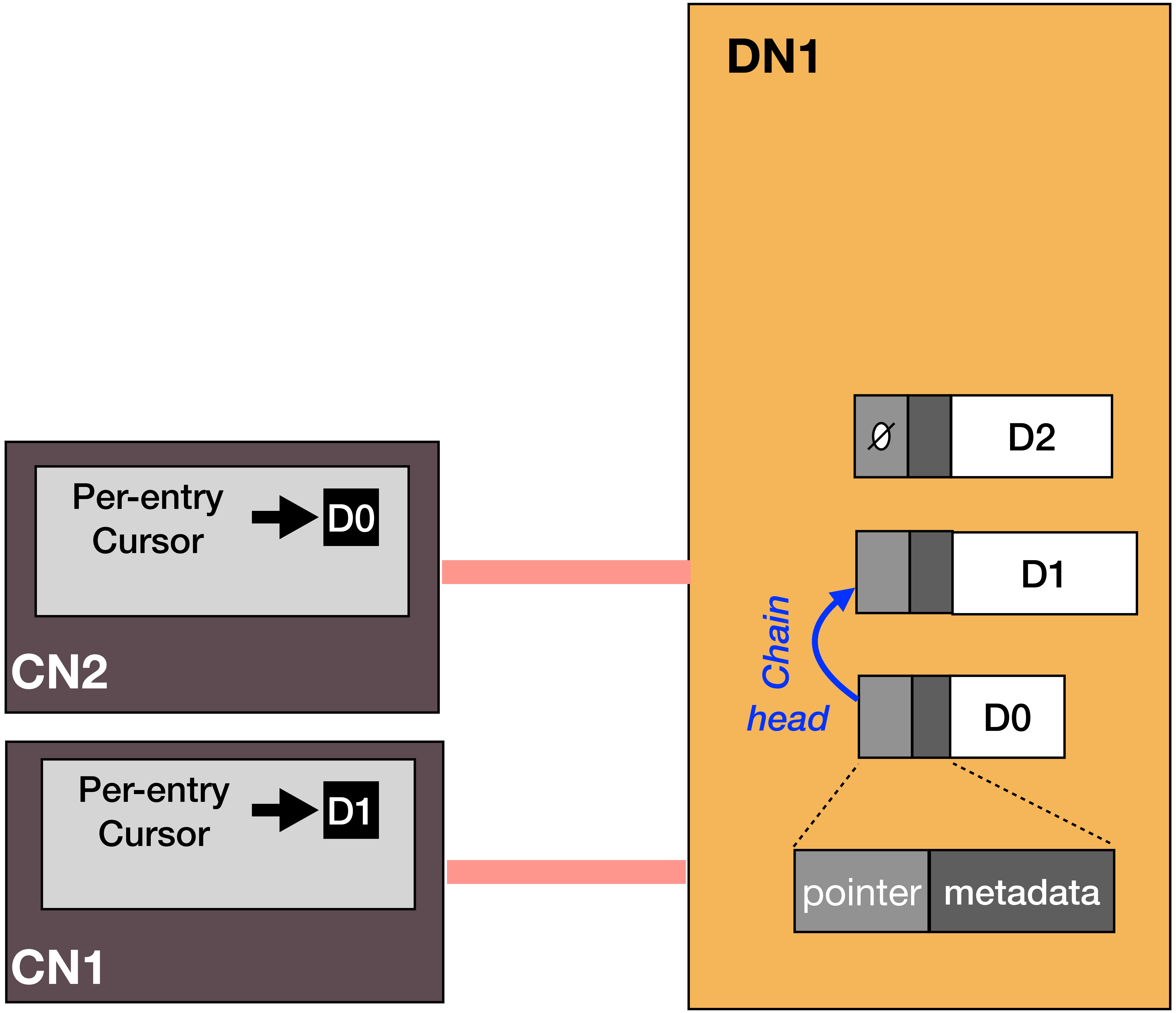
Lock-free data structures

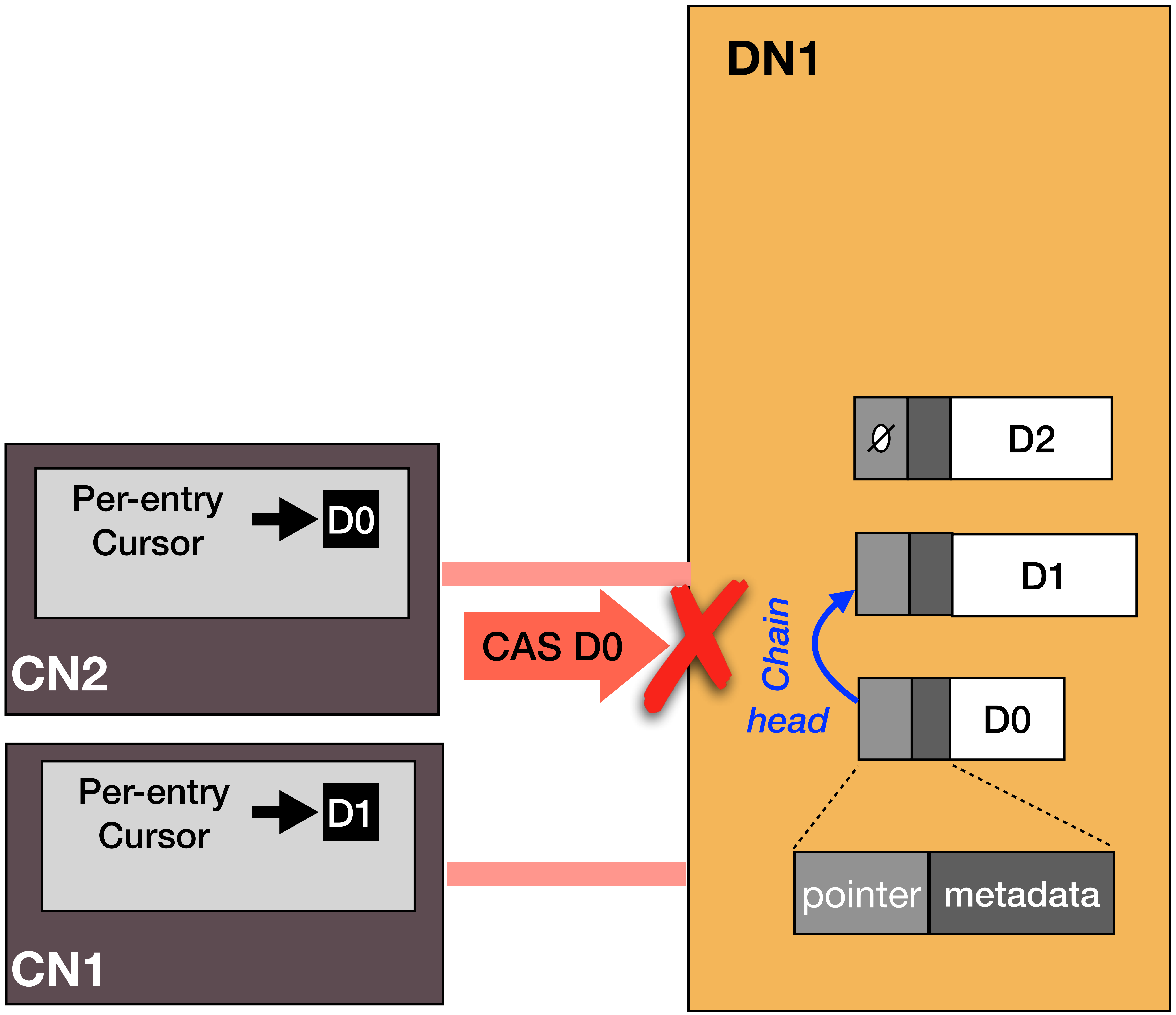
Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry





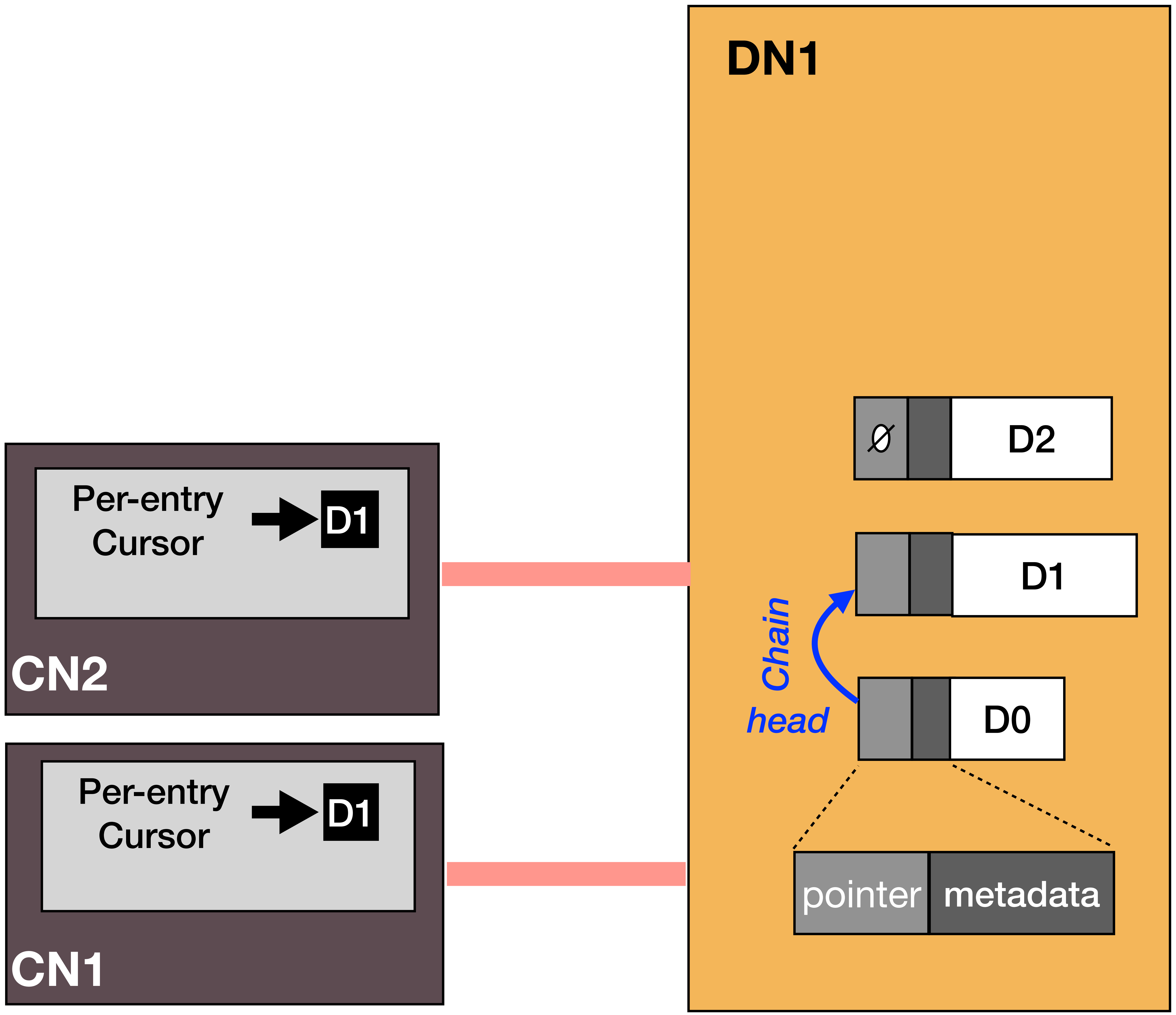
Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a *cursor* that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***c&s***
3. If 2. fails, update cursor and retry



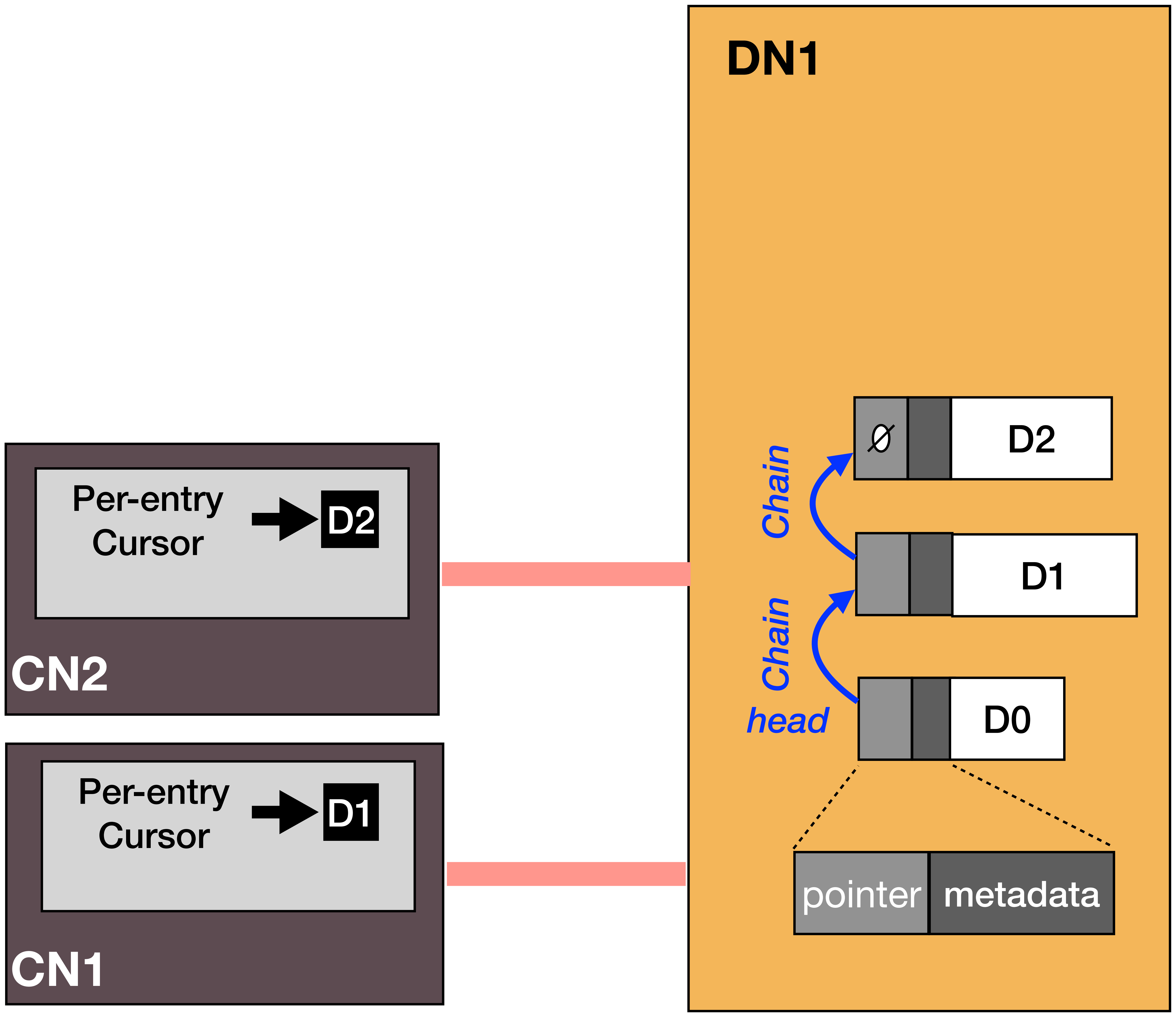
Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry



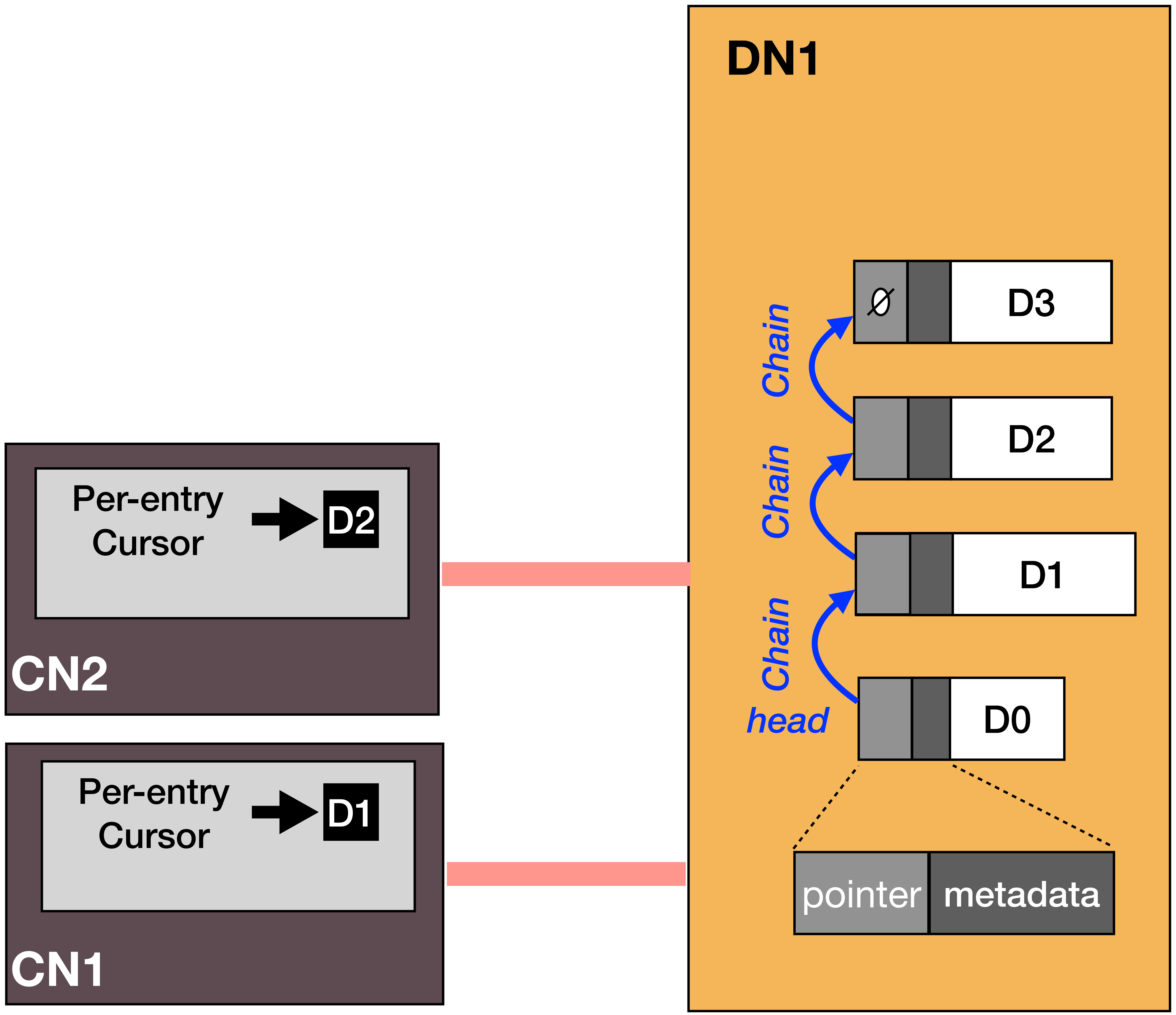
Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry



Lock-free data structures

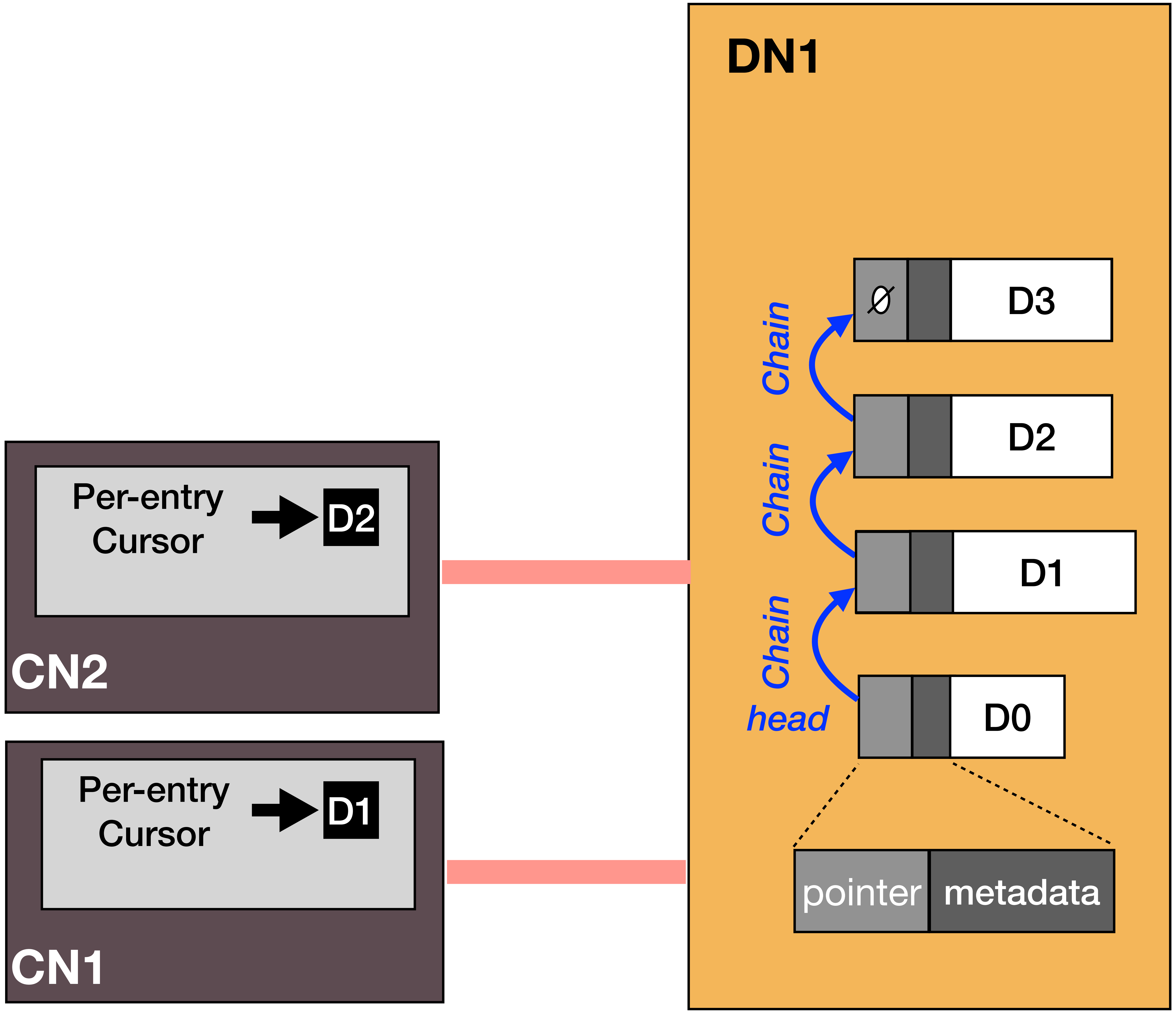
Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

Lock-free data structures



Chained redo copies (versions) at `DNs`

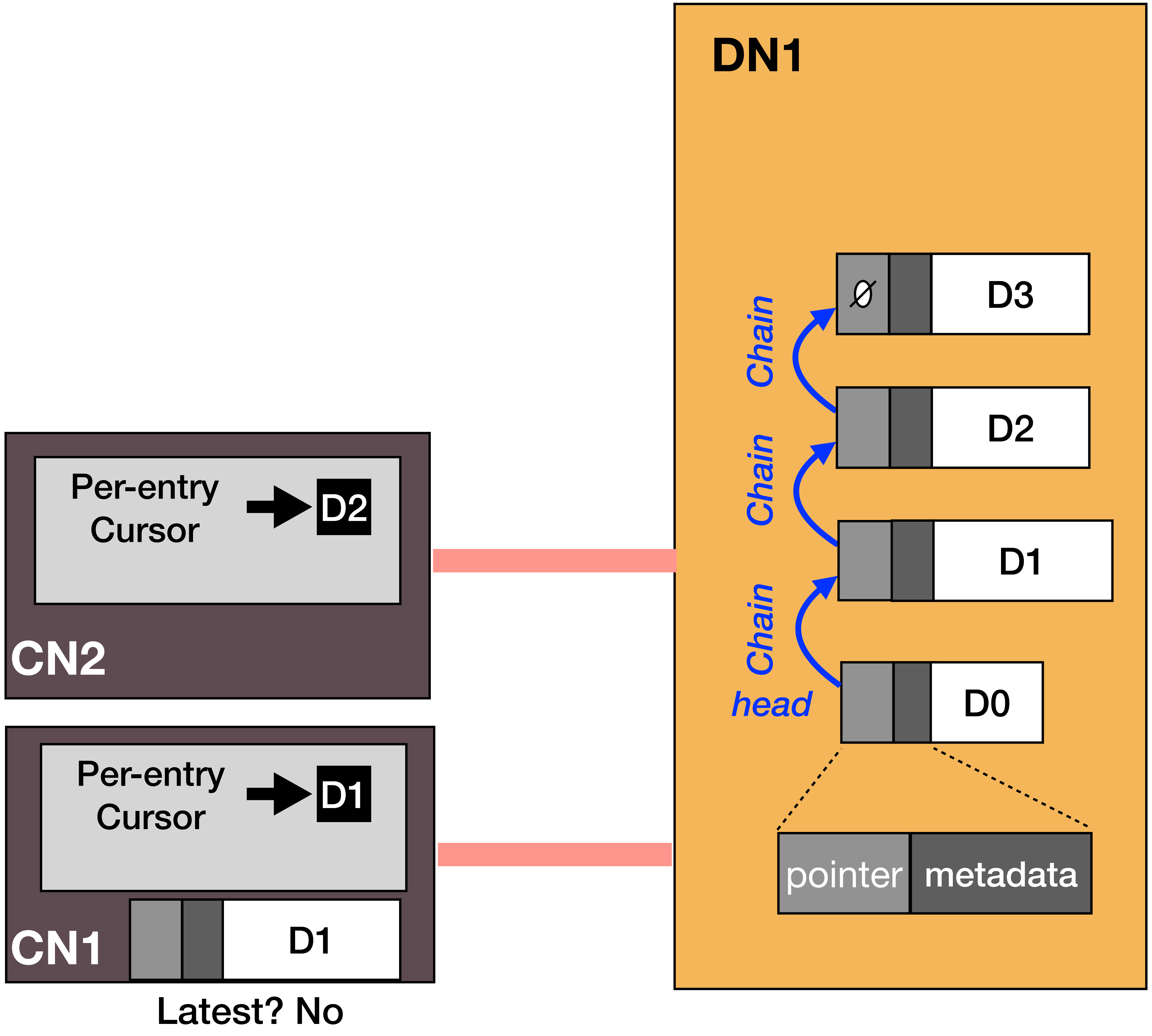
`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest



Lock-free data structures

Chained redo copies (versions) at `DNs`

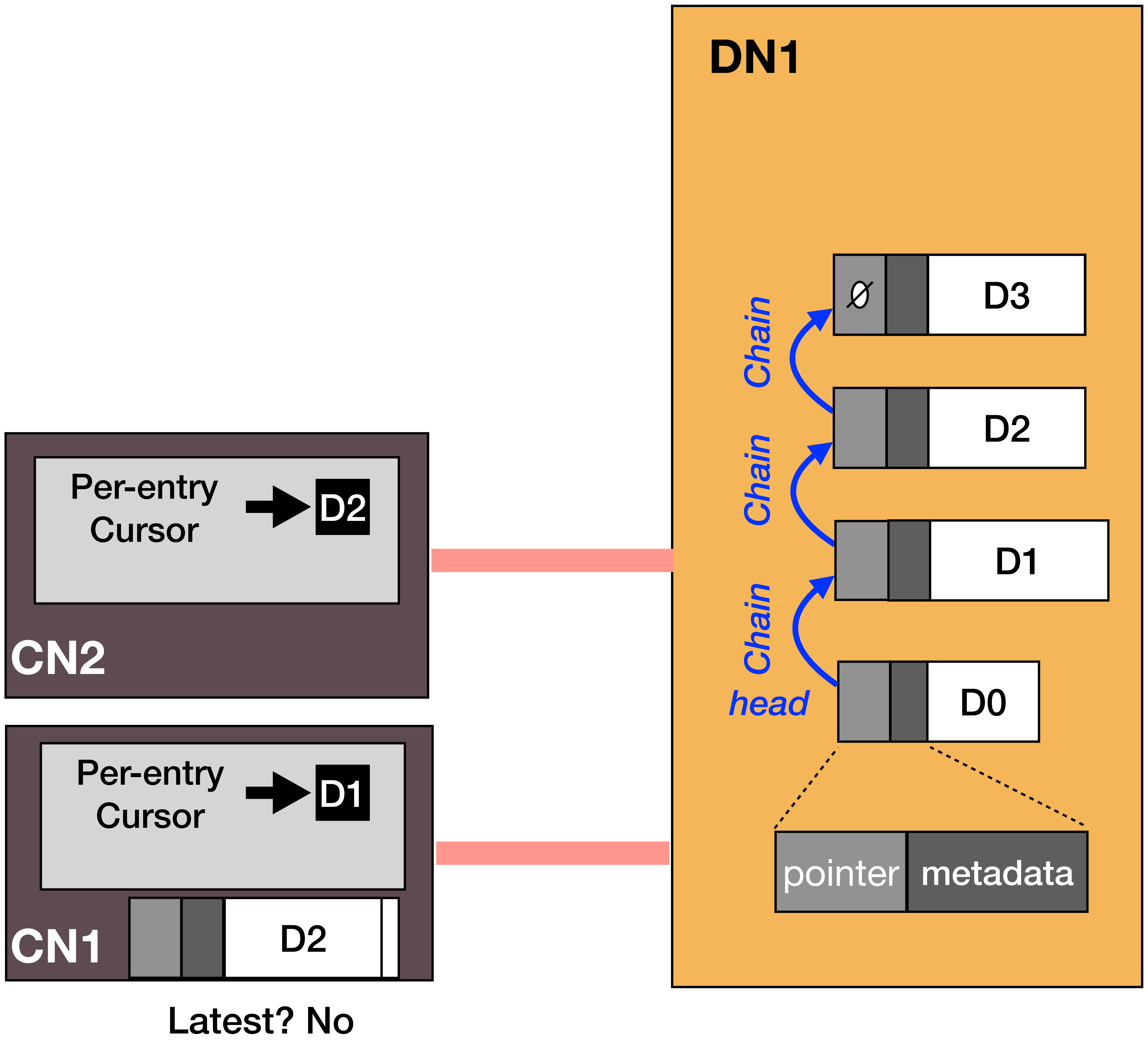
`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest



Lock-free data structures

Chained redo copies (versions) at `DNs`

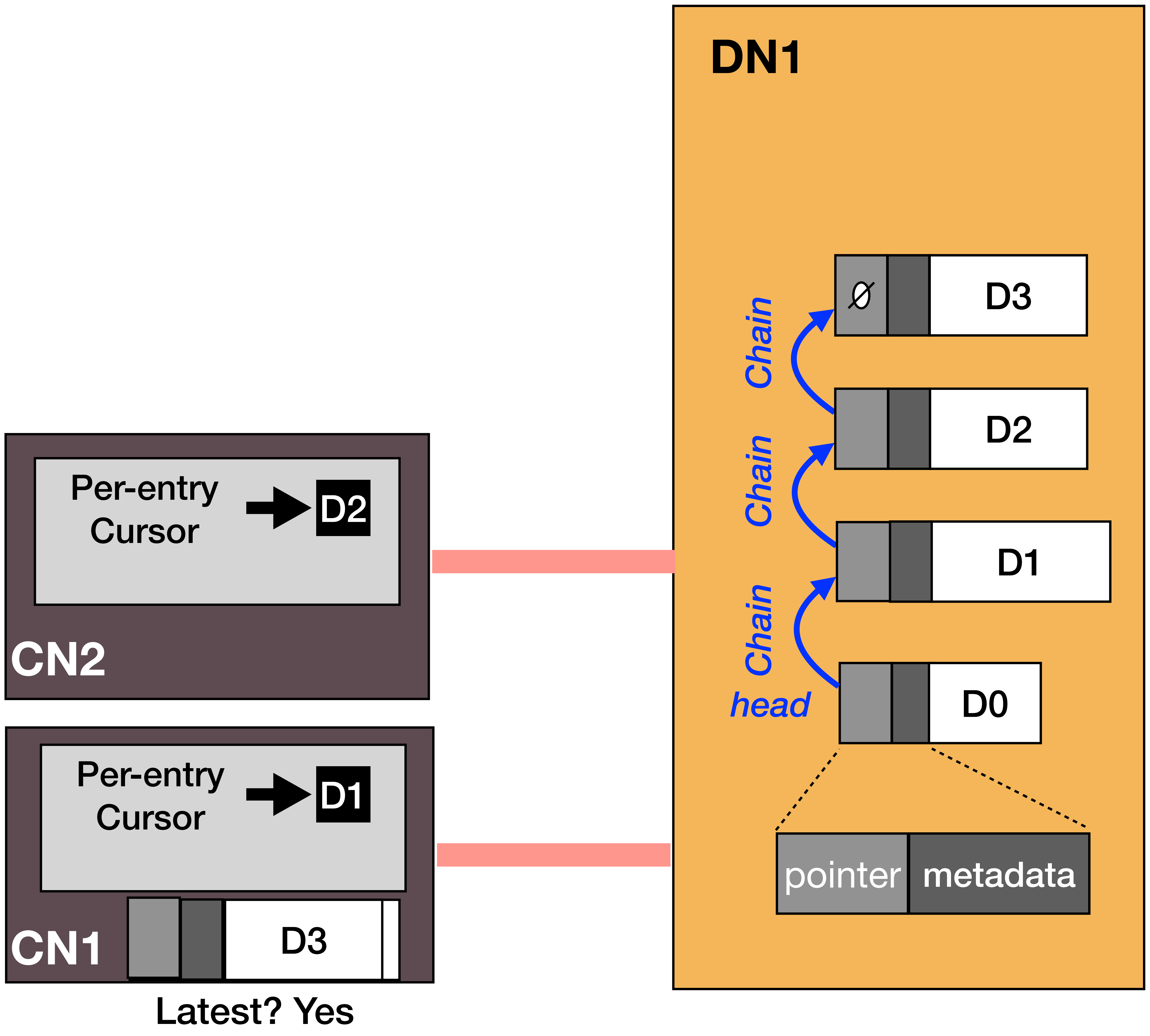
`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest



Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

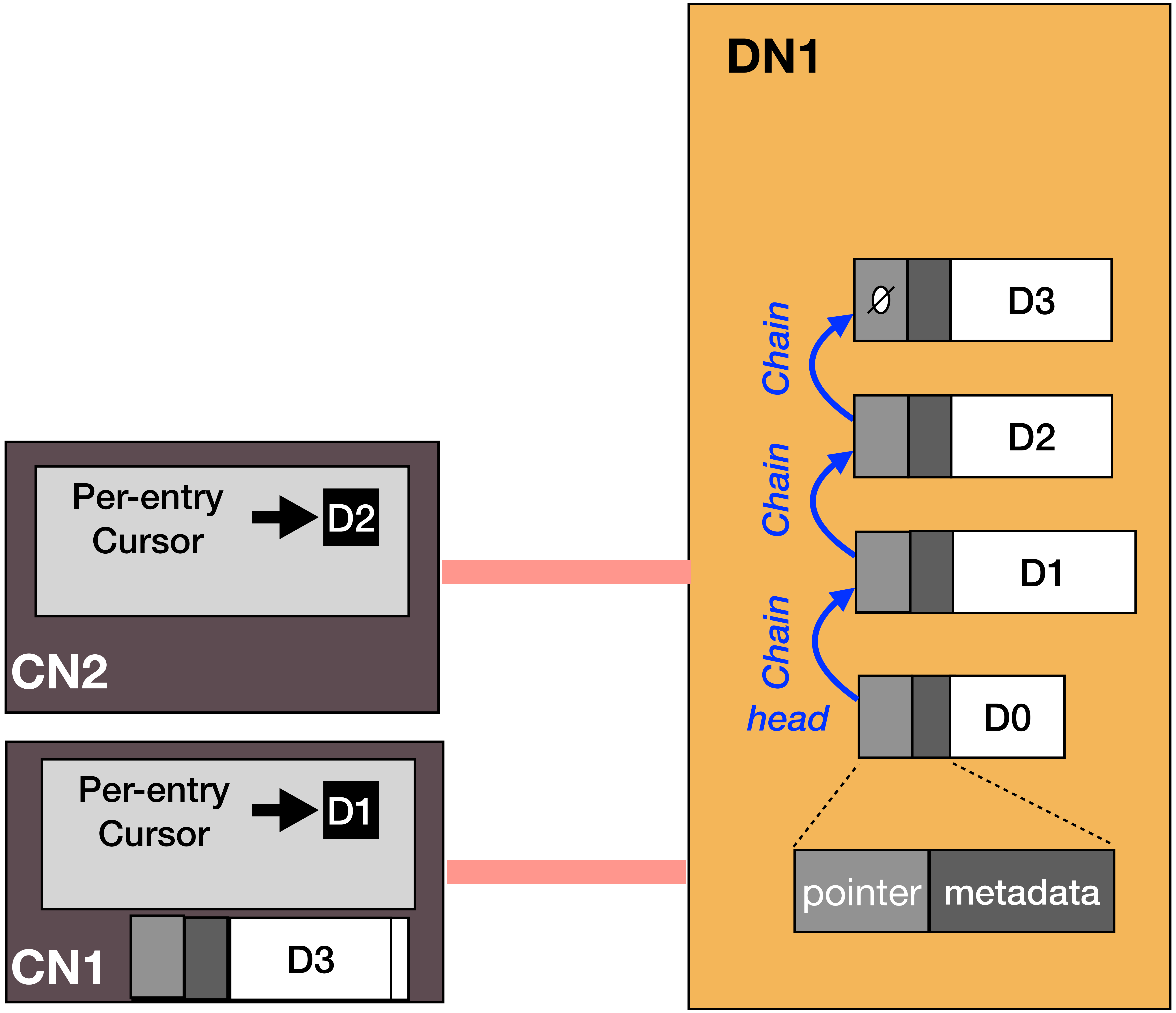
Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest

Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version



Latest? Yes

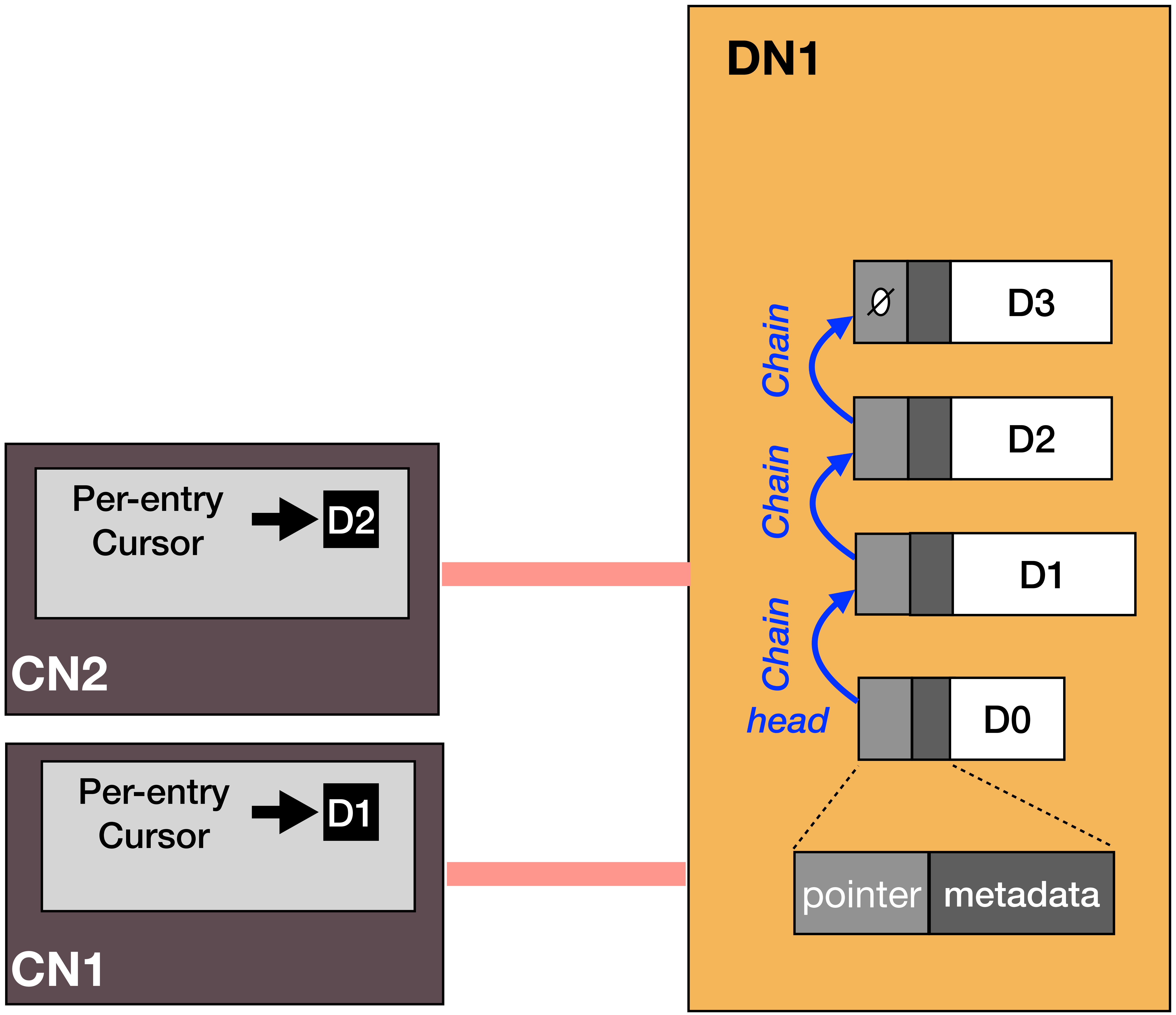
3 RTTs to finish a KV READ!

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest



3 RTTs to finish a KV READ!

Lock-free data structures

Chained redo copies (versions) at `DNs`
`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest

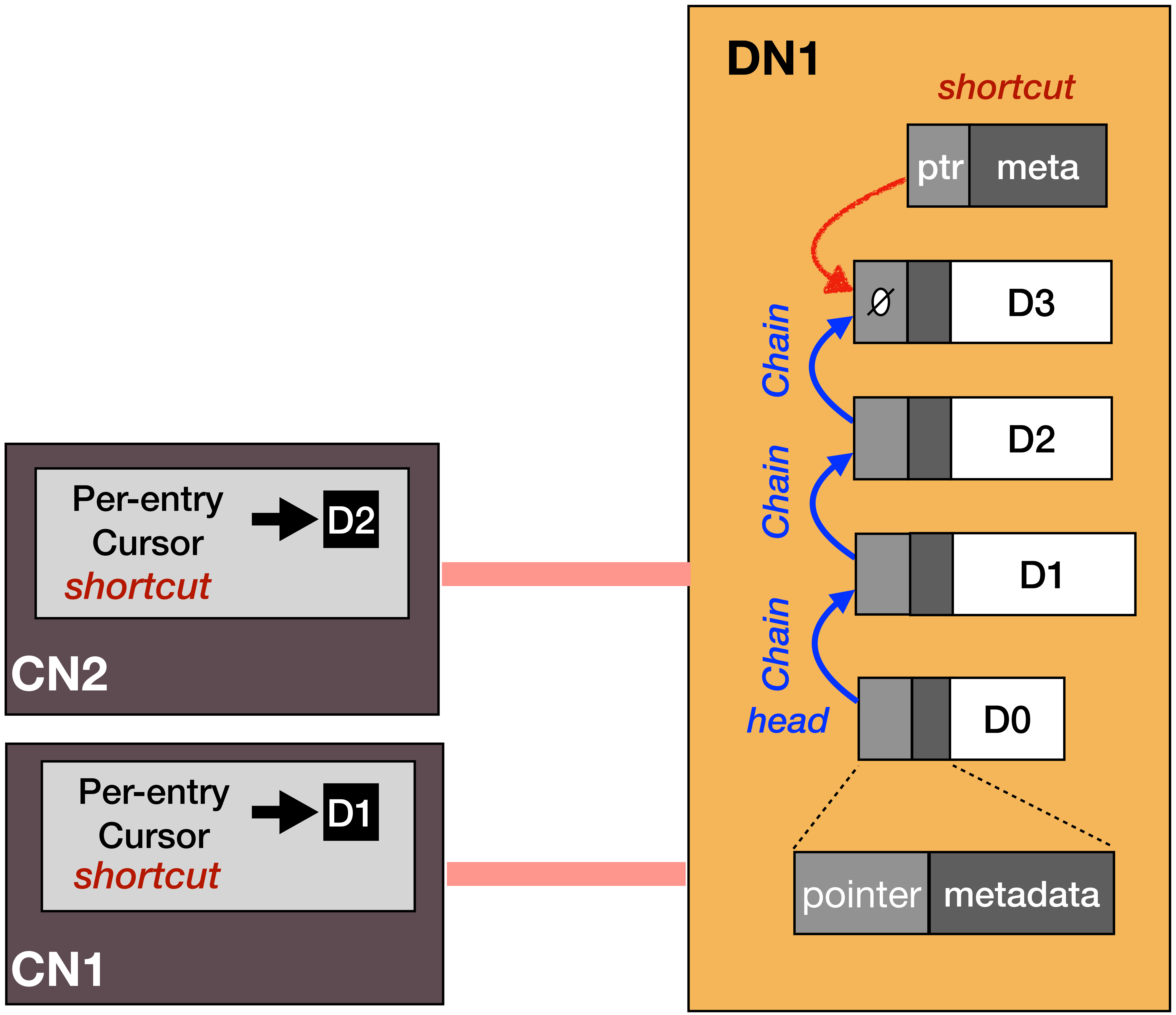
Optimization: Shortcut

Uses a shortcut to avoid long chain walk
A shortcut at DN (mostly) points to the latest data

1. CN reads shortcut, then uses it to read data
2. CN still does cursor read in parallel

- Returns when the faster of 1 and 2 finish

Lock-free data structures



3 RTTs to finish a KV READ!

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest

Optimization: Shortcut

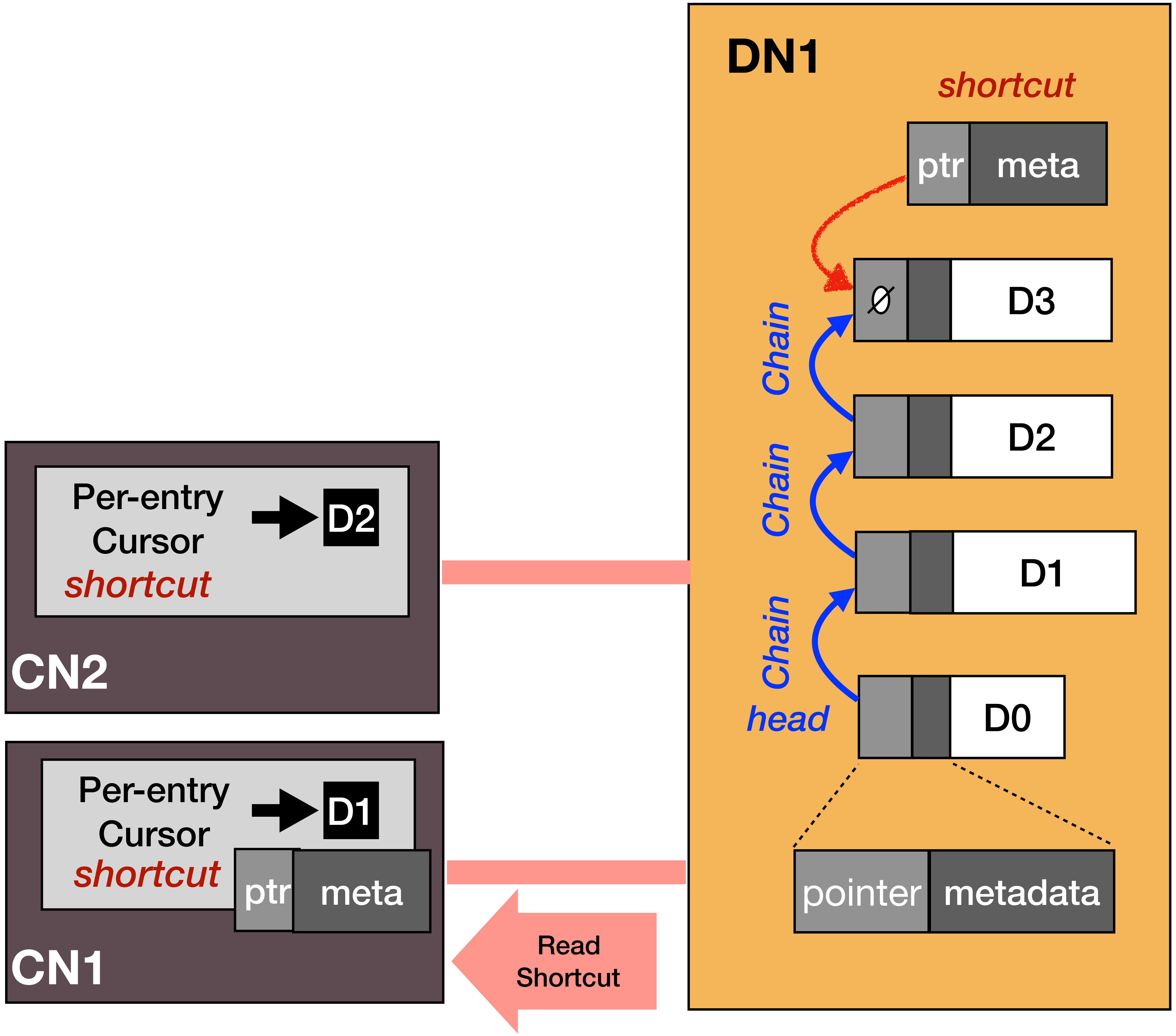
Uses a shortcut to avoid long chain walk

A shortcut at DN (mostly) points to the latest data

1. CN reads shortcut, then uses it to read data
2. CN still does cursor read in parallel

- Returns when the faster of 1 and 2 finish

Lock-free data structures



3 RTTs to finish a KV READ!

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***`c&s`***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest

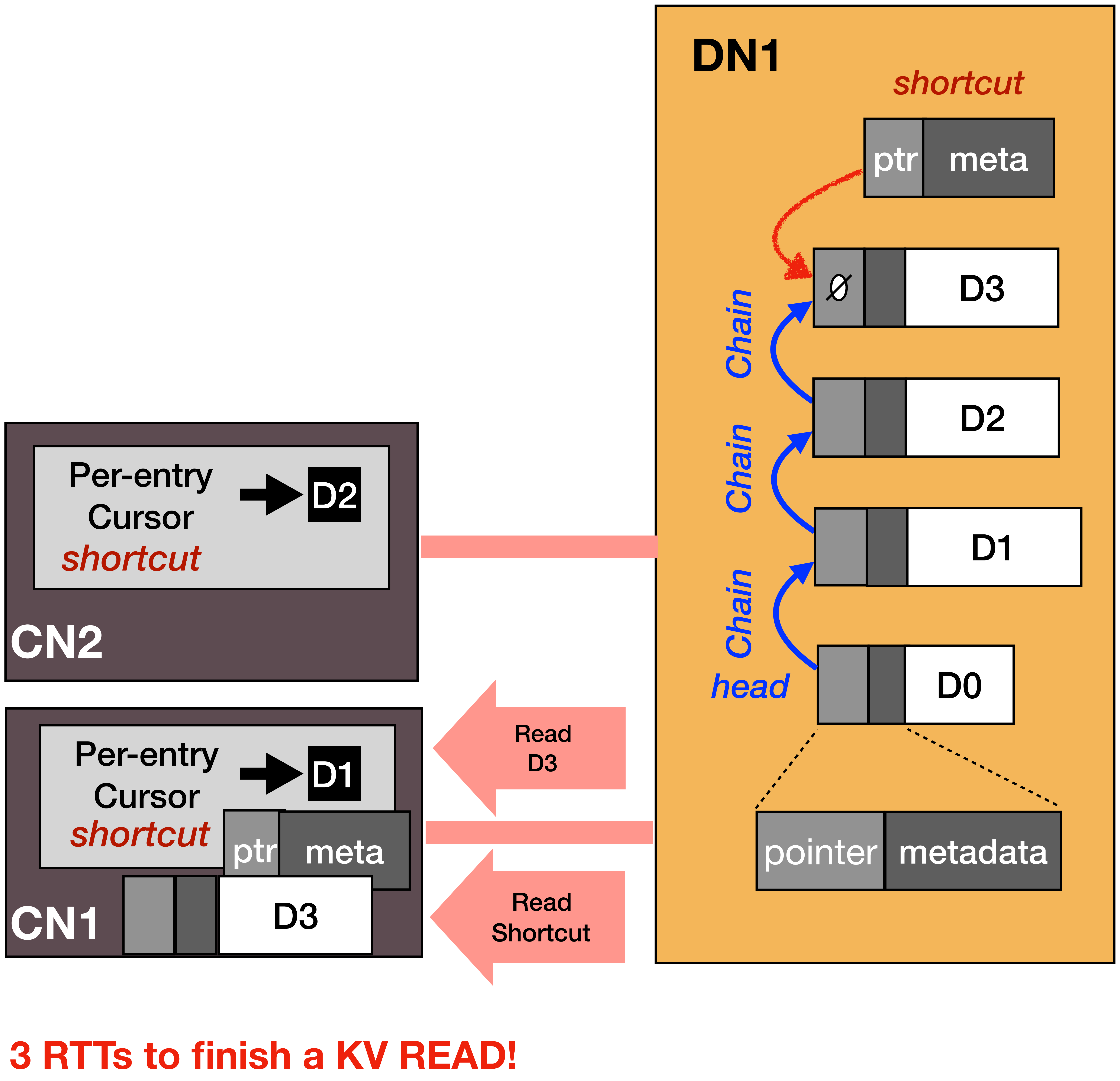
Optimization: Shortcut

Uses a shortcut to avoid long chain walk

A shortcut at DN (mostly) points to the latest data

1. CN reads shortcut, then uses it to read data
2. CN still does cursor read in parallel

- Returns when the faster of 1 and 2 finish



Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version

Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest

Optimization: Shortcut

Uses a shortcut to avoid long chain walk

A shortcut at DN (mostly) points to the latest data

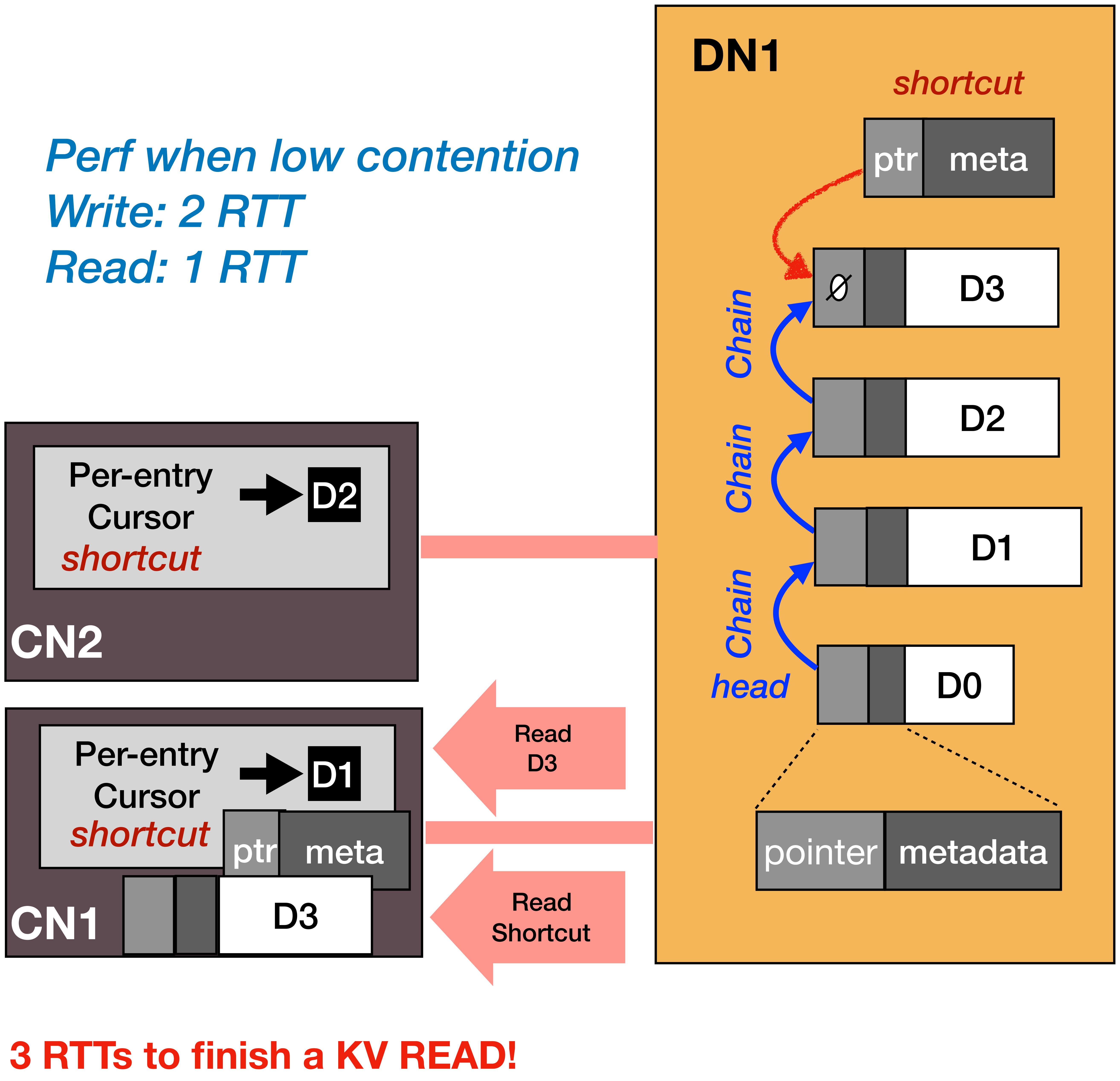
1. CN reads shortcut, then uses it to read data
2. CN still does cursor read in parallel

- Returns when the faster of 1 and 2 finish

Lock-free data structures

Chained redo copies (versions) at `DNs`

`CNs` cache a ***cursor*** that points to a version



Write Flow

1. Out-of-place write (create redo copy)
2. Chain the redo-copy, using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Fetch cursor-pointed data
2. Walks the chain until found the latest

Optimization: Shortcut

Uses a shortcut to avoid long chain walk

A shortcut at DN (mostly) points to the latest data

1. CN reads shortcut, then uses it to read data
2. CN still does cursor read in parallel

- Returns when the faster of 1 and 2 finish

Main Challenges in Metadata Plane:

How to provide low-overhead, scalable metadata service?

Main Challenges in Metadata Plane:

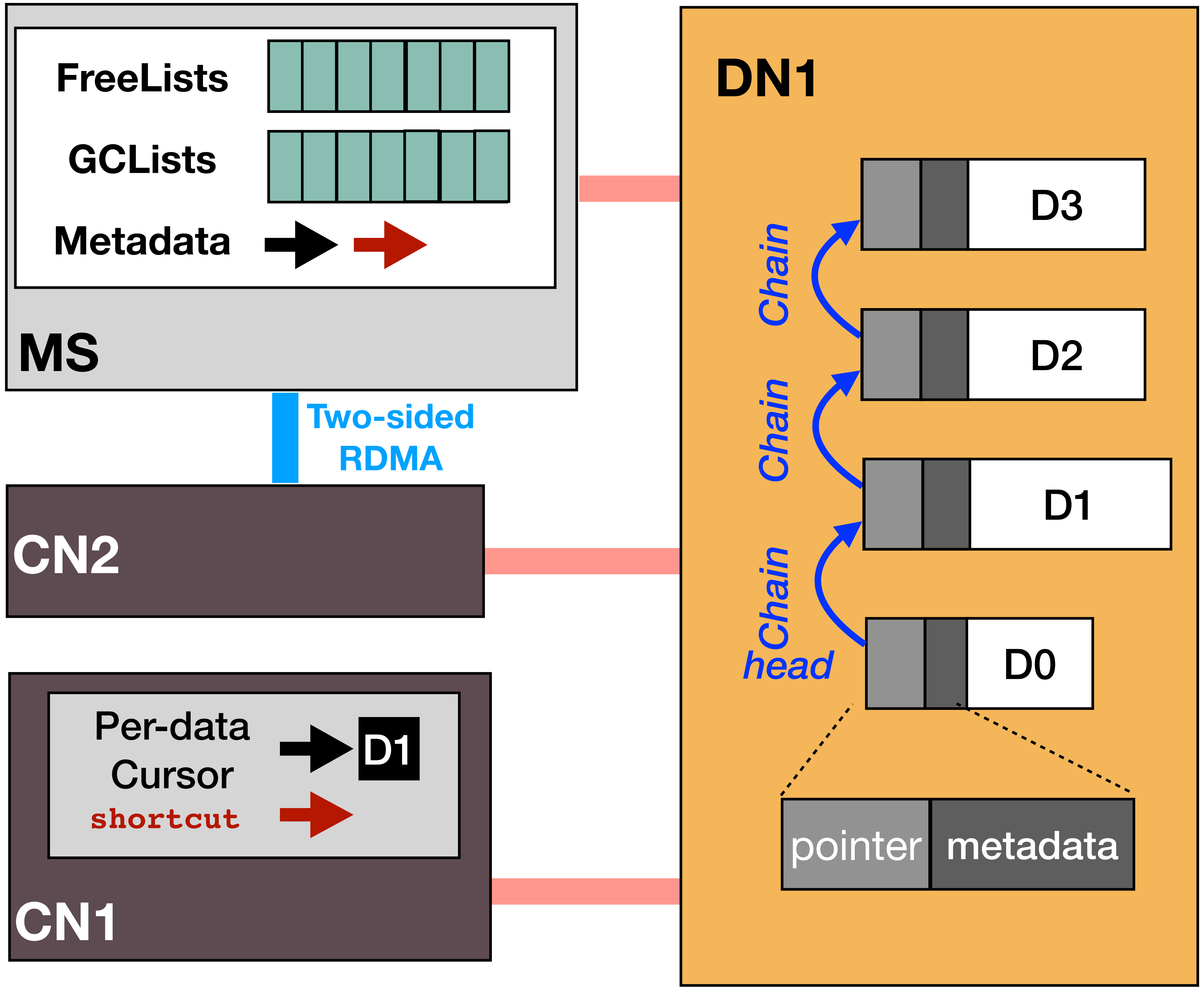
How to provide low-overhead, scalable metadata service?

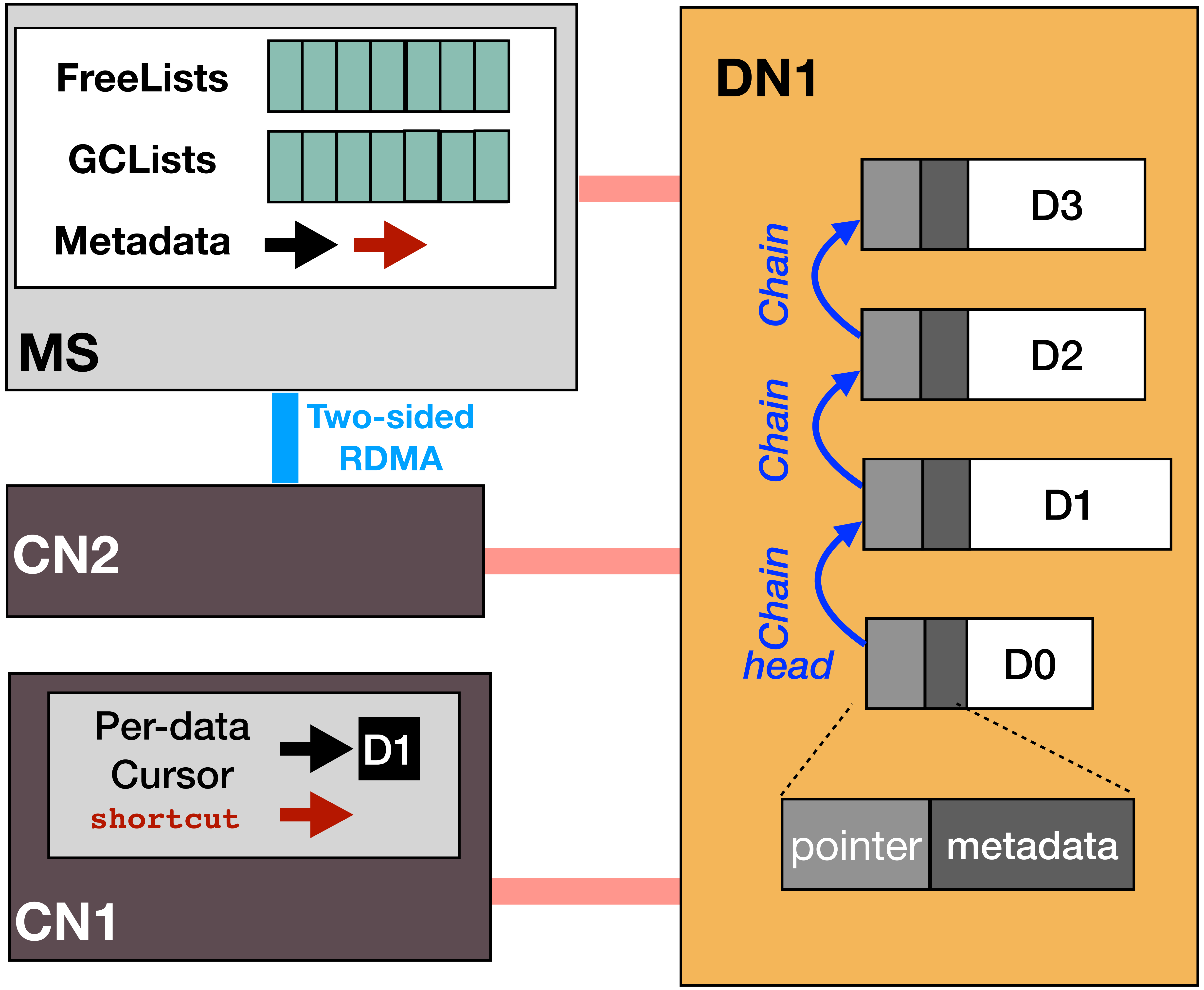
Our Approaches

- Move ***all*** metadata operations off performance critical paths
- Batch metadata operations
- ***No*** cache invalidation

➡ No performance overhead caused by metadata ops (common case)

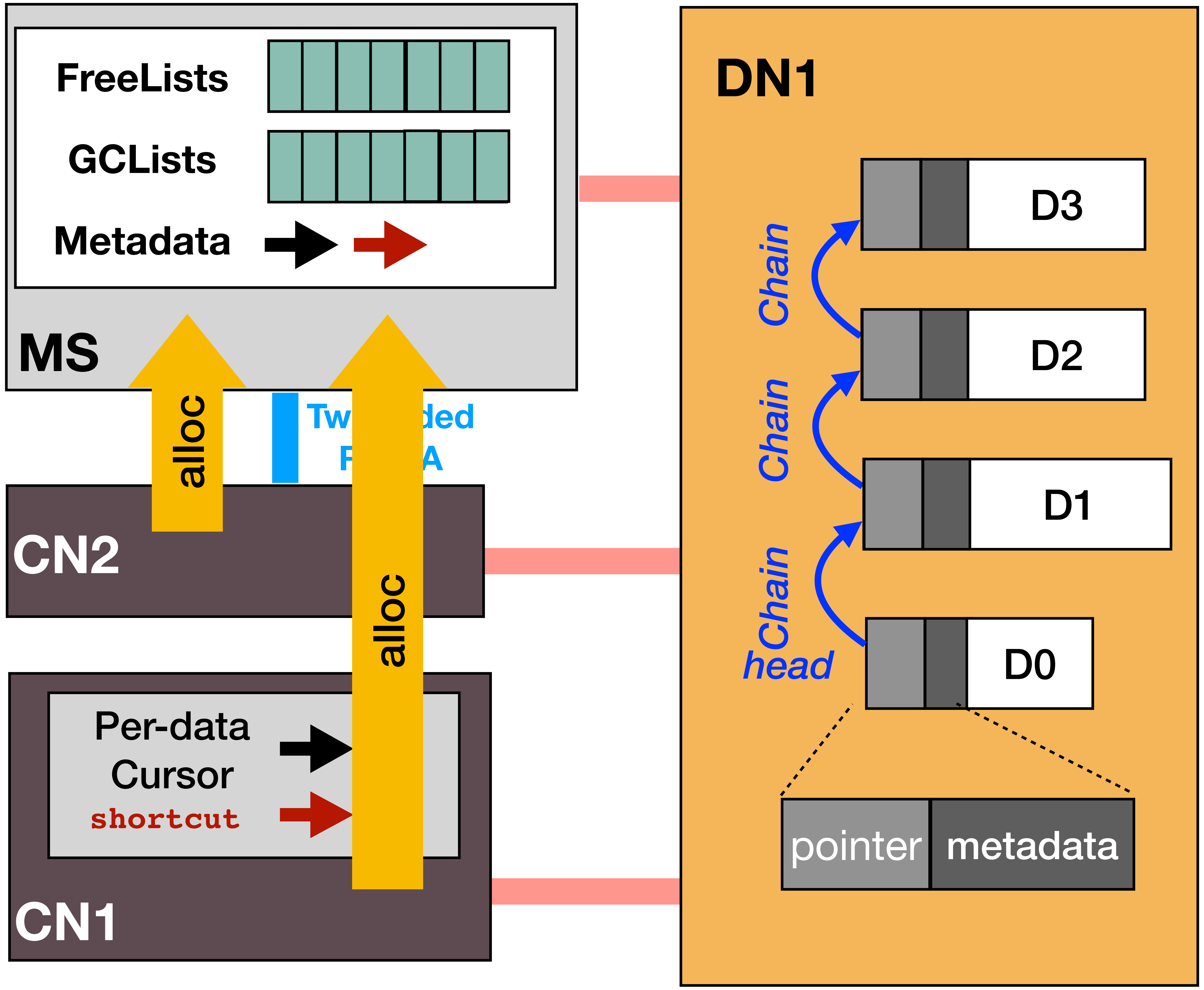
Metadata Server (MS)





Metadata Server (MS)

- Space management
- Garbage collection
- Global load balancing

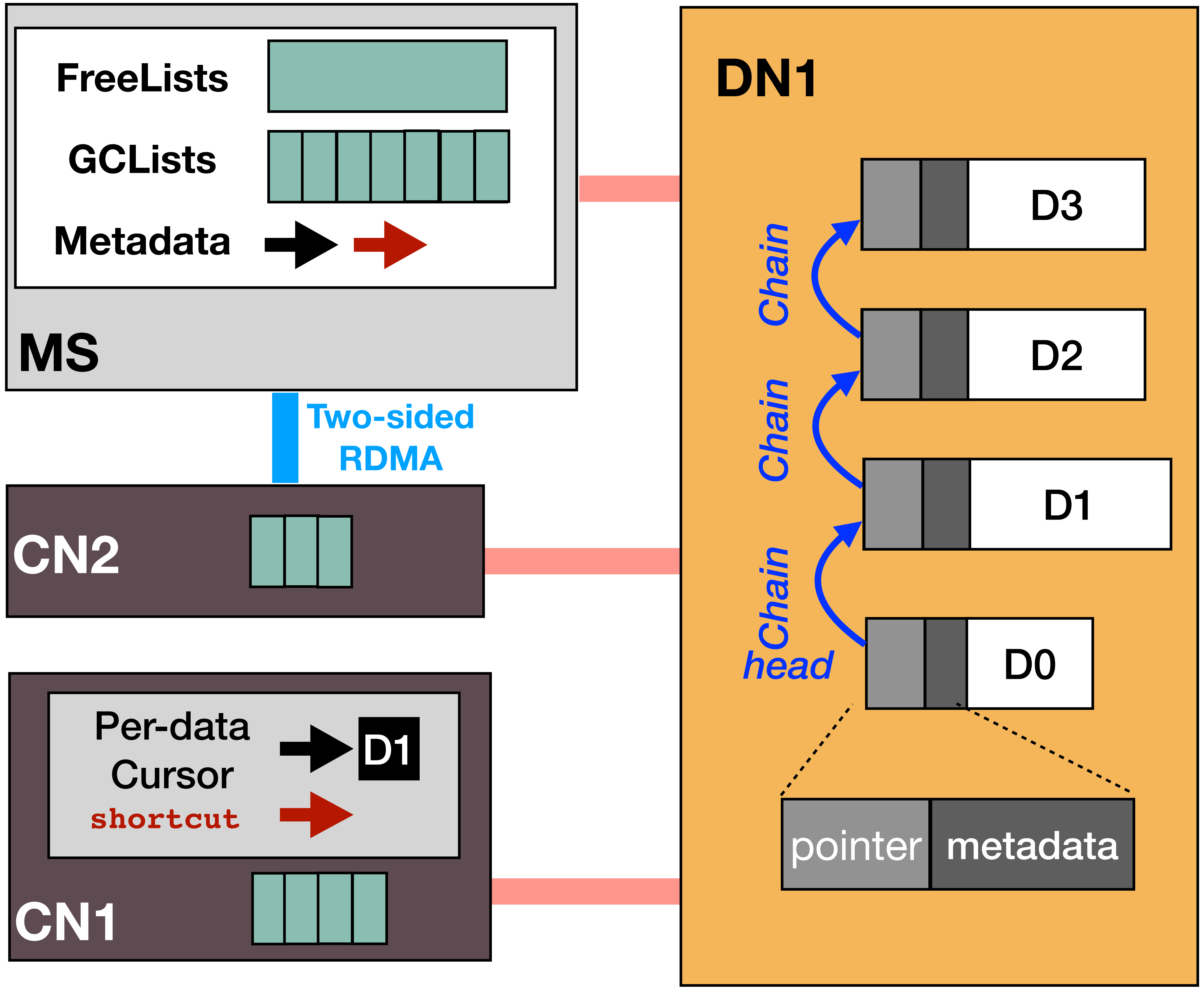


Metadata Server (MS)

- Space management
- Garbage collection
- Global load balancing

Alloc Flow

- CN asks MS for a bunch of free buffers at a time
- MS assigns spaces from FreeLists (with load balancing consideration)

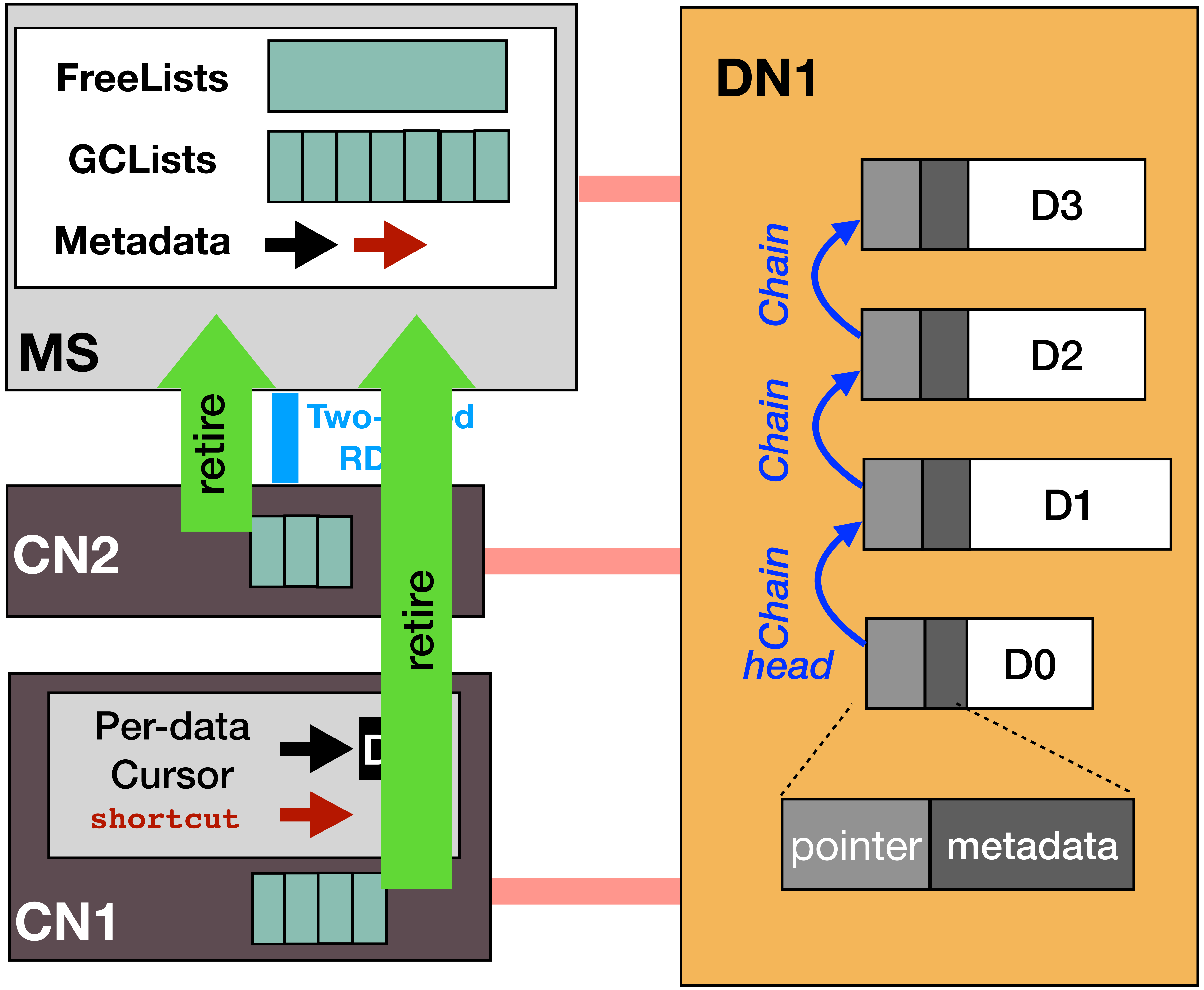


Metadata Server (MS)

- Space management
- Garbage collection
- Global load balancing

Alloc Flow

- CN asks MS for a bunch of free buffers at a time
- MS assigns spaces from FreeLists (with load balancing consideration)



Metadata Server (MS)

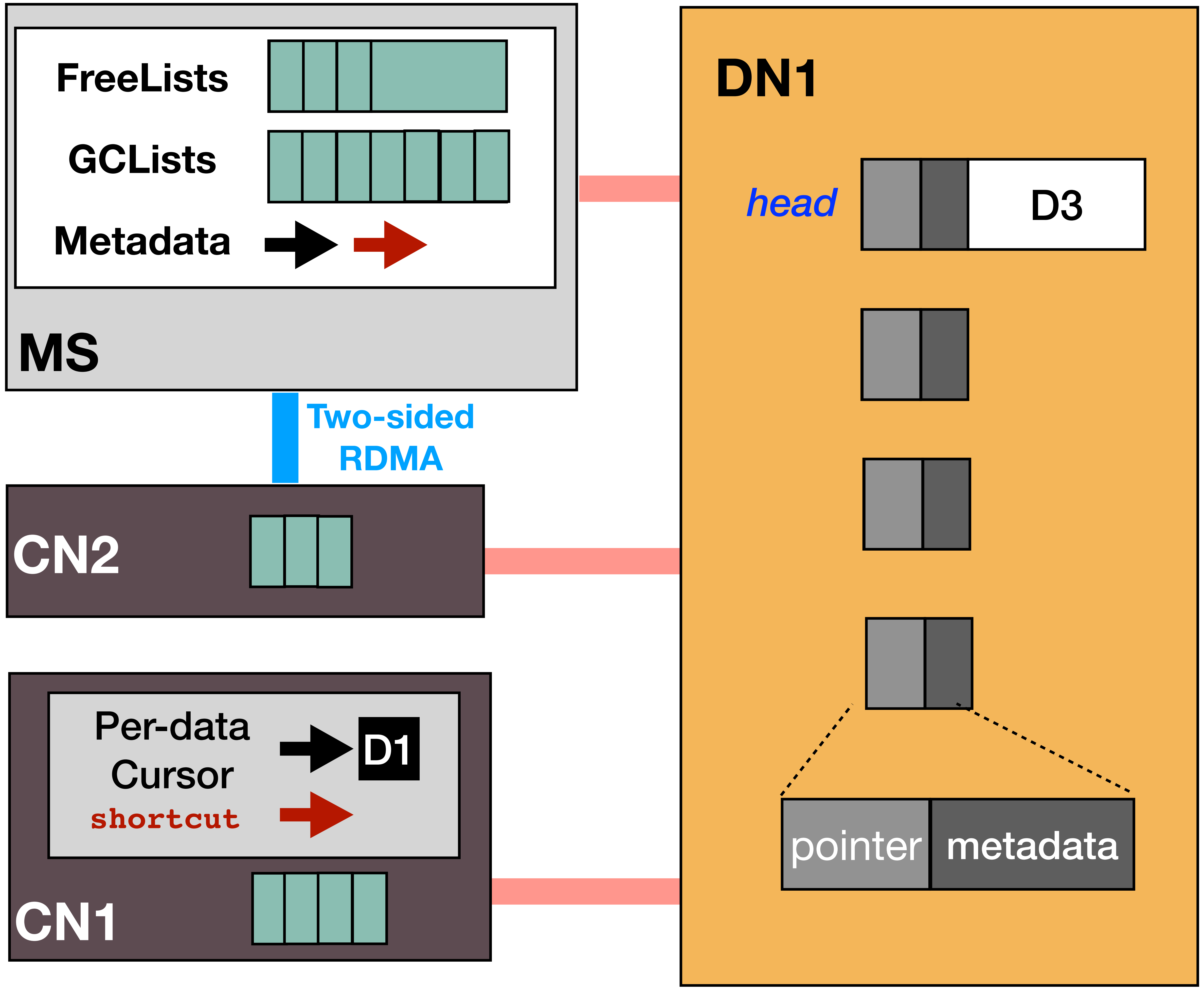
- Space management
- Garbage collection
- Global load balancing

Alloc Flow

- CN asks MS for a bunch of free buffers at a time
- MS assigns spaces from FreeLists (with load balancing consideration)

GC Flow

- After write, CN asynchronously ***retires*** a batch of old versions
- MS enqueues them into FreeLists



Metadata Server (MS)

- Space management
- Garbage collection
- Global load balancing

Alloc Flow

- CN asks MS for a bunch of free buffers at a time
- MS assigns spaces from FreeLists (with load balancing consideration)

GC Flow

- After write, CN asynchronously ***retires*** a batch of old versions
- MS enqueues them into FreeLists

Reliability and Load Balancing

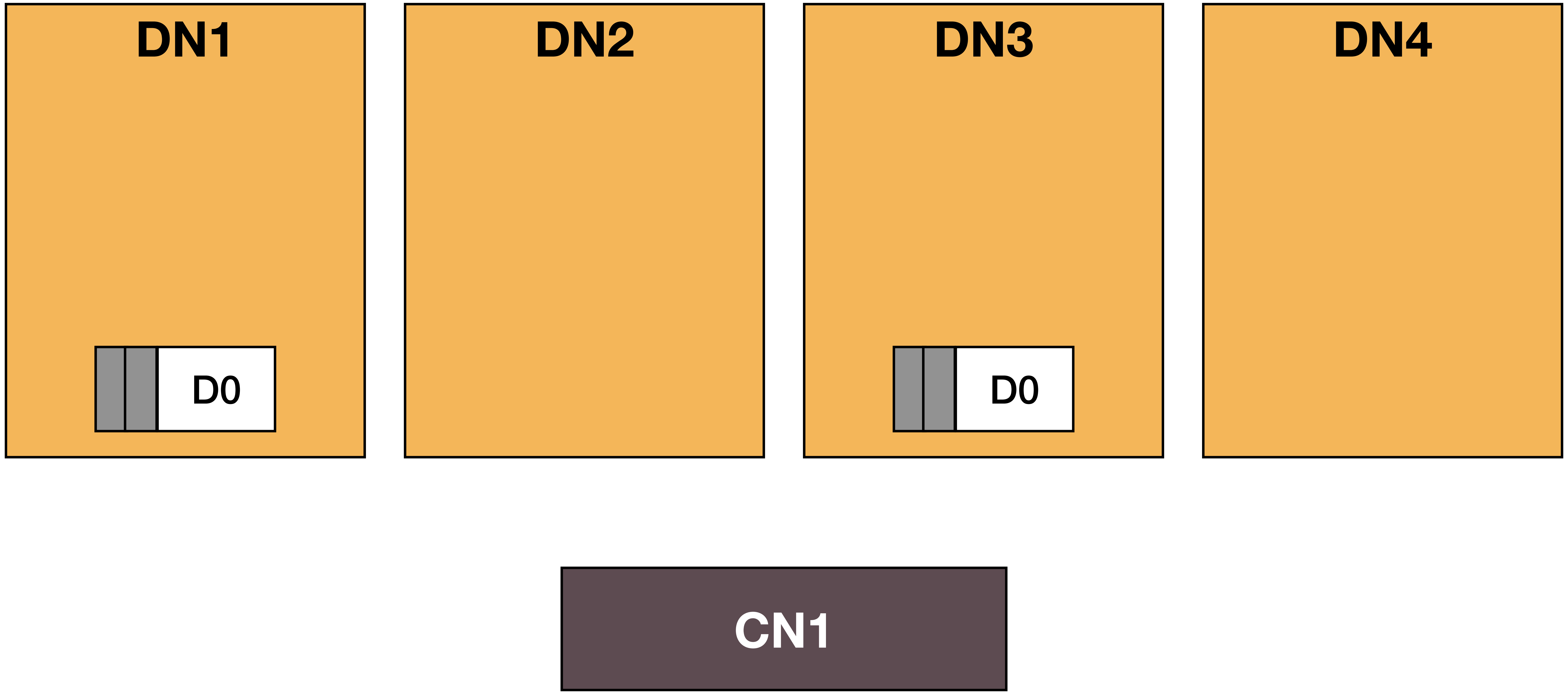
Data reliability through a novel chaining replication

- Link a version to all the replicas of next version

Metadata reliability through shadow MS servers

Load balancing via a two-level approach

- MS and CNs both control location
- Versions in a chain can be on different DNs



Reliability and Load Balancing

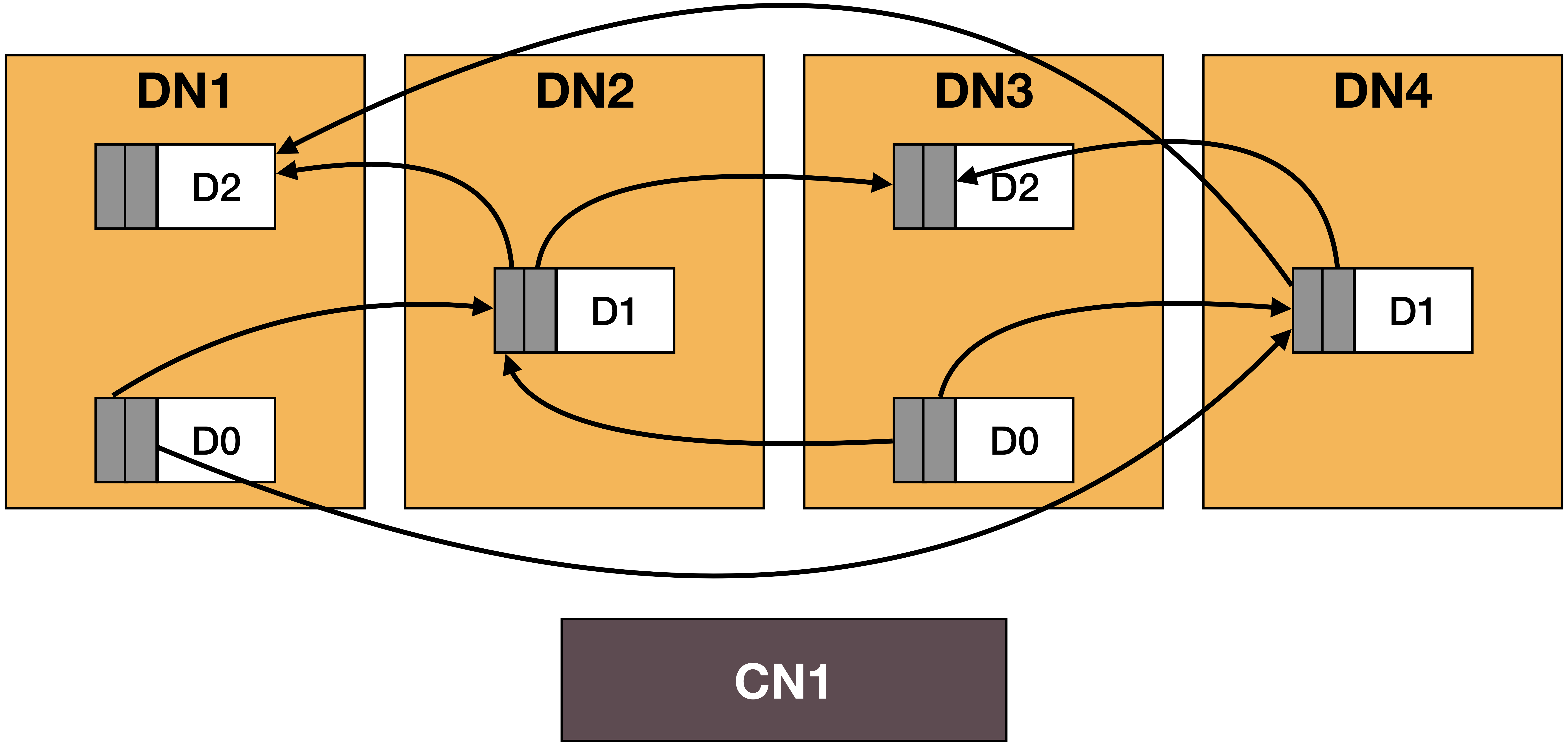
Data reliability through a novel chaining replication

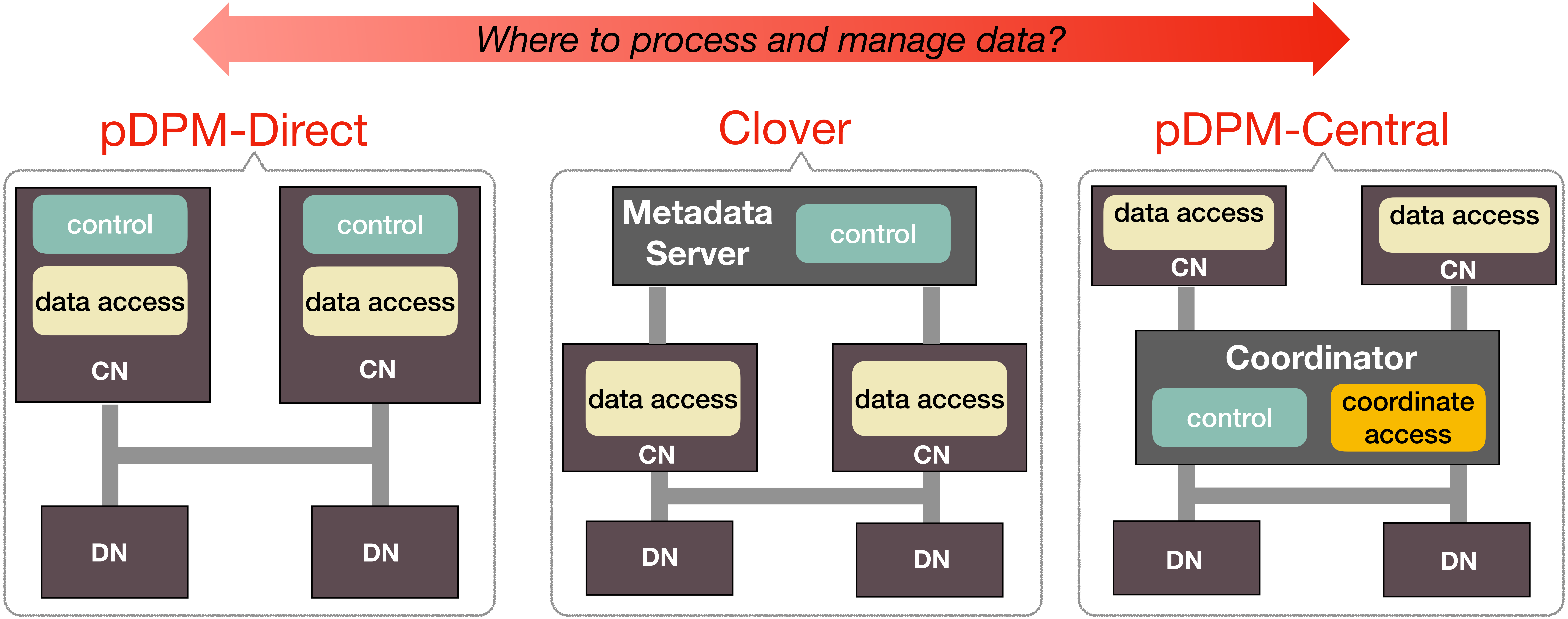
- Link a version to all the replicas of next version

Metadata reliability through shadow MS servers

Load balancing via a two-level approach

- MS and CNs both control location
- Versions in a chain can be on different DNs





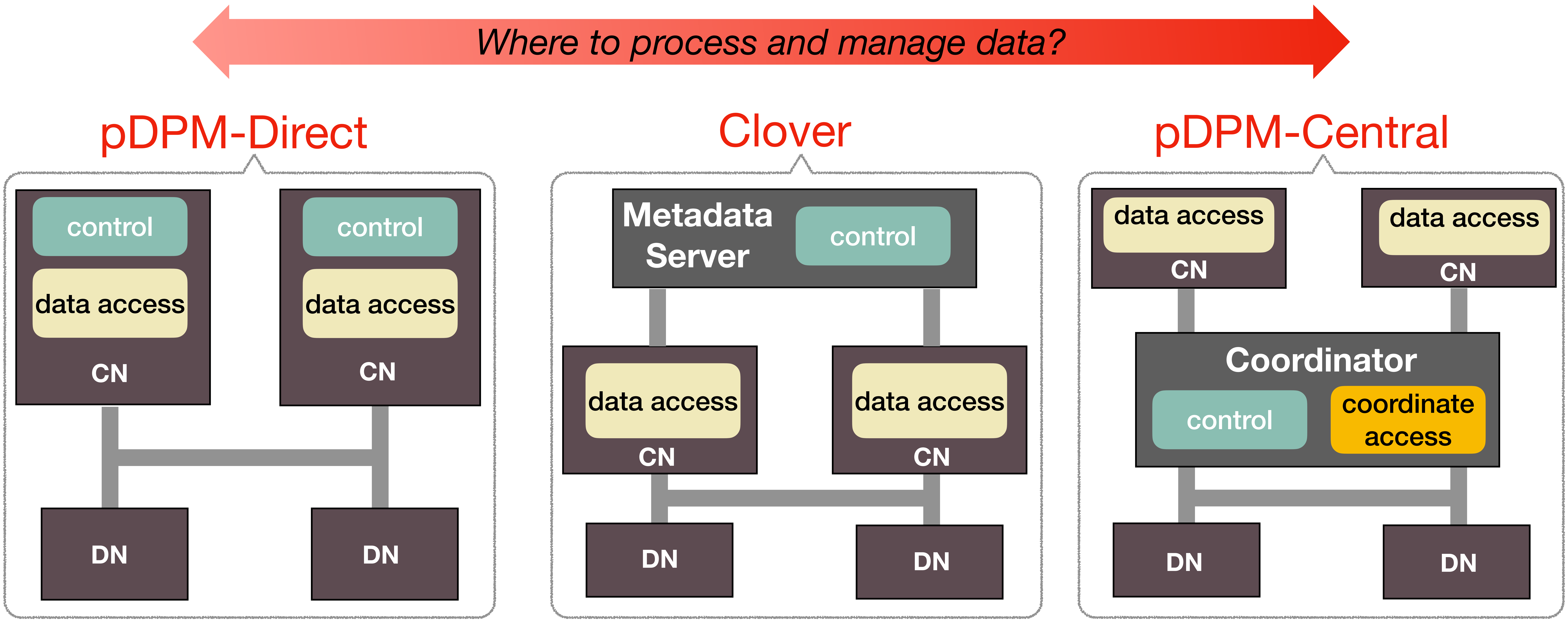
- Write cannot scale
- Large metadata consumption

- Extra read RTTs
- Coordinator cannot scale

Distributed data & metadata

Separate data & metadata

Centralized data & metadata



- Write cannot scale
- Large metadata consumption

Distributed data & metadata

\+ Good read/write performance
\+ Scale with both CNs and DNs

Separate data & metadata

- Extra read RTTs
- Coordinator cannot scale

Centralized data & metadata

Evaluation Setup

Systems evaluated

- ***pDPM systems***: pDPM-Direct, pDPM-Central, Clover
- ***Non-disaggregated PM systems***: Octopus [ATC'17] and Hotpot [SoCC'17]
- ***Two-sided KVS***: HERD [SIGCOMM'14] (also ported to BlueField SmartNIC, HERD-BF)

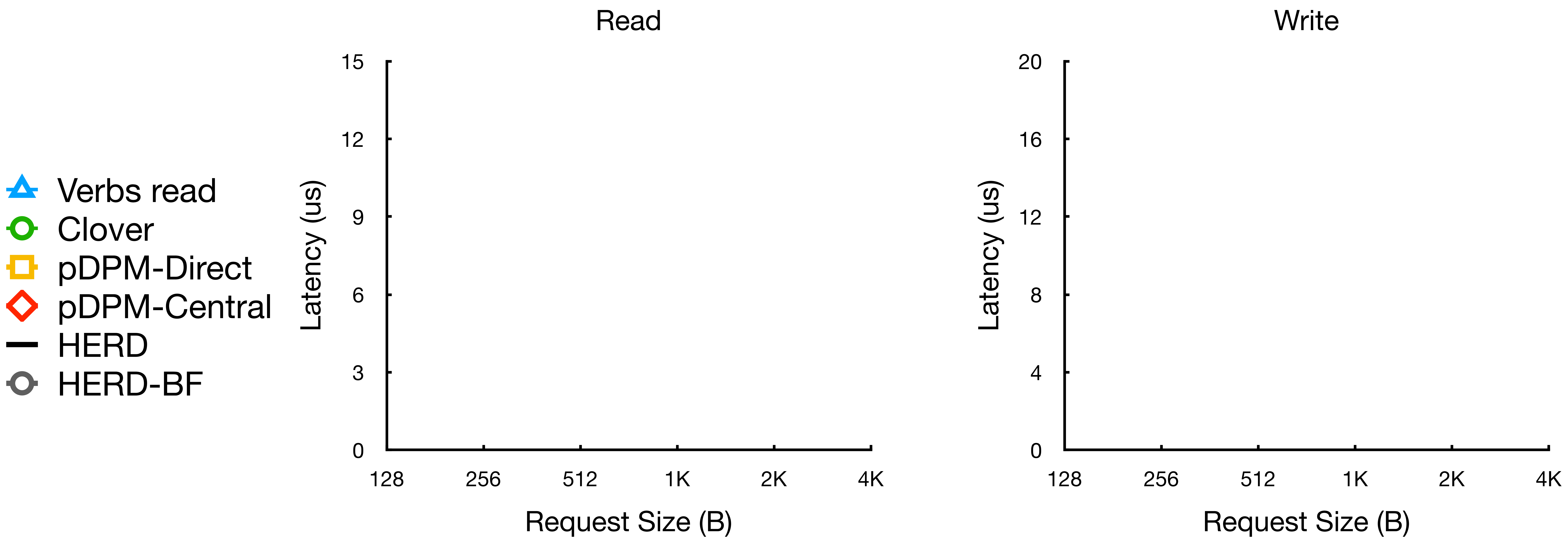
Testbed

- 14 servers, each with a 100Gbps RDMA NIC, connected via a 100Gbps IB switch
- DRAM as emulated PM

Microbenchmark - Latency

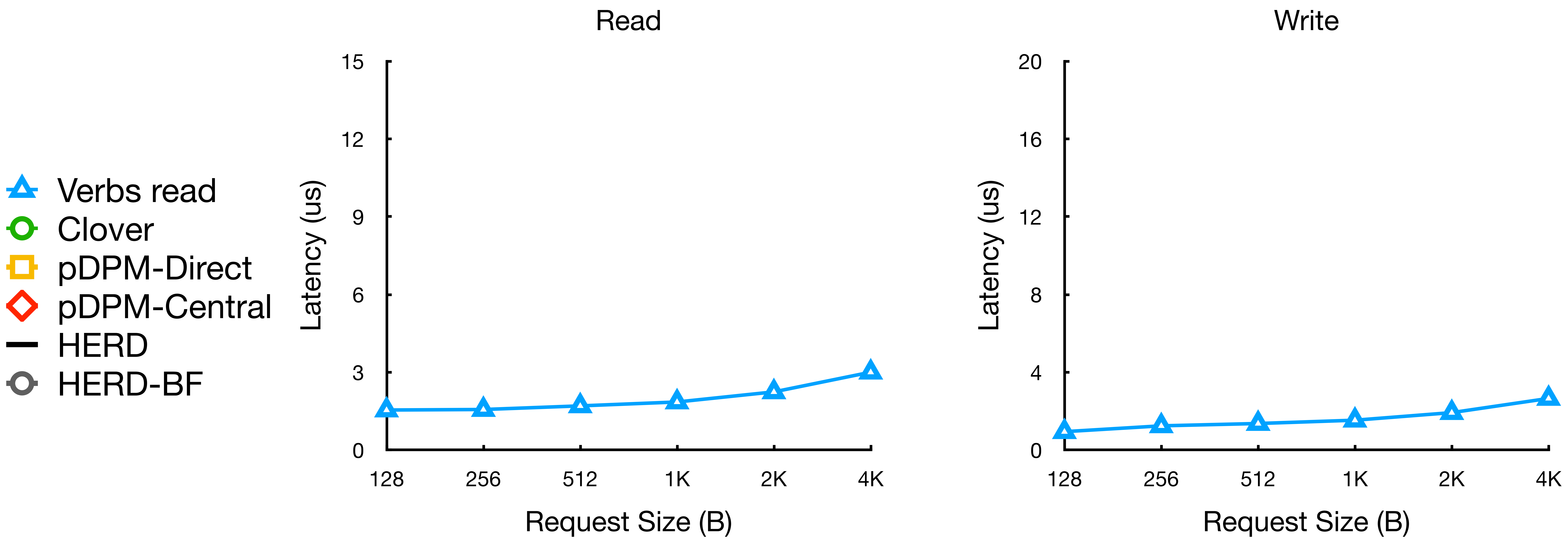
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



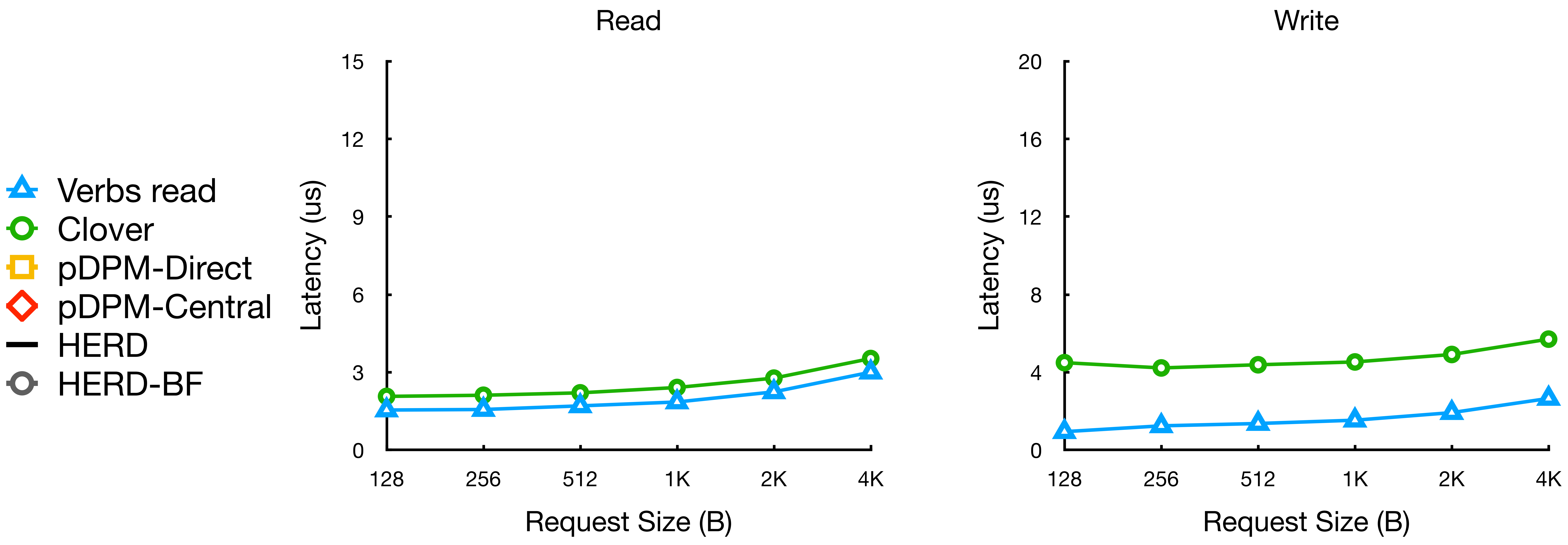
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



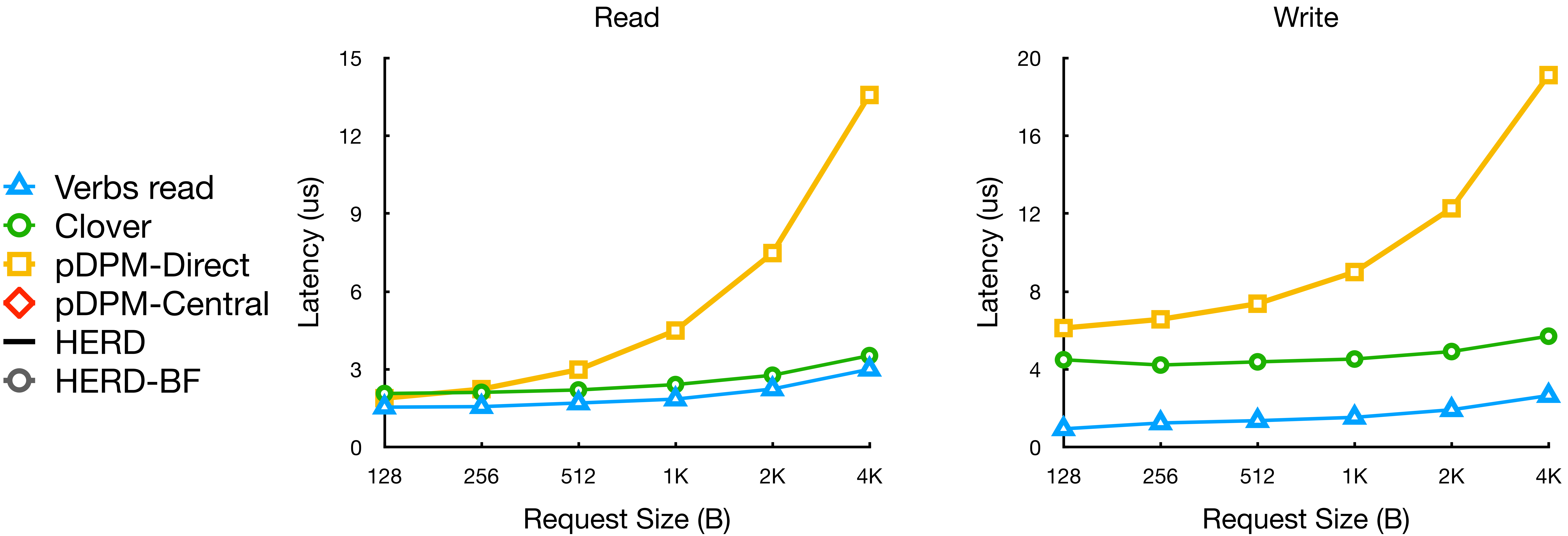
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



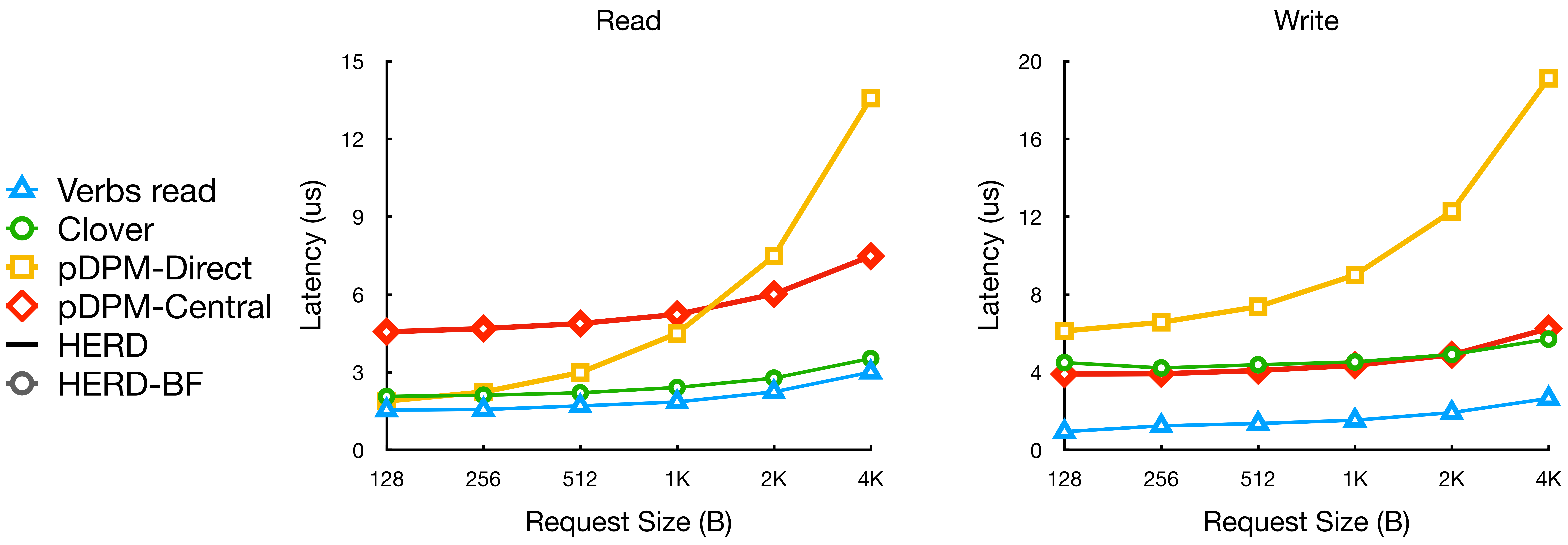
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



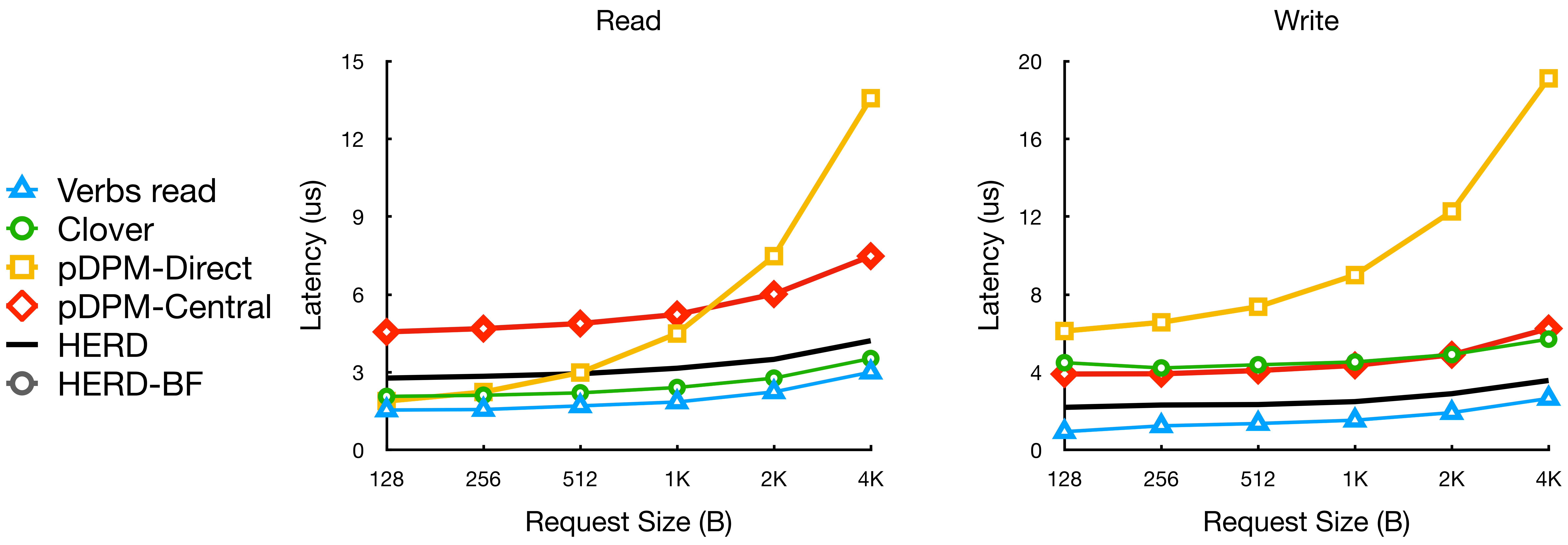
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



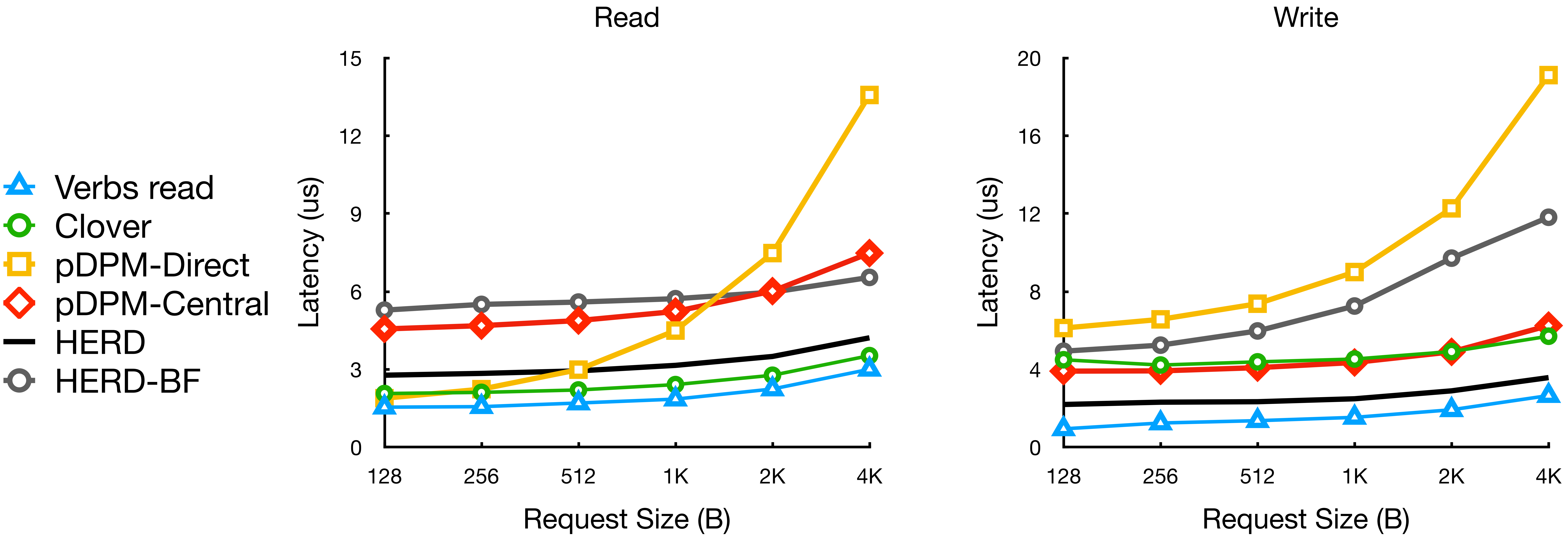
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



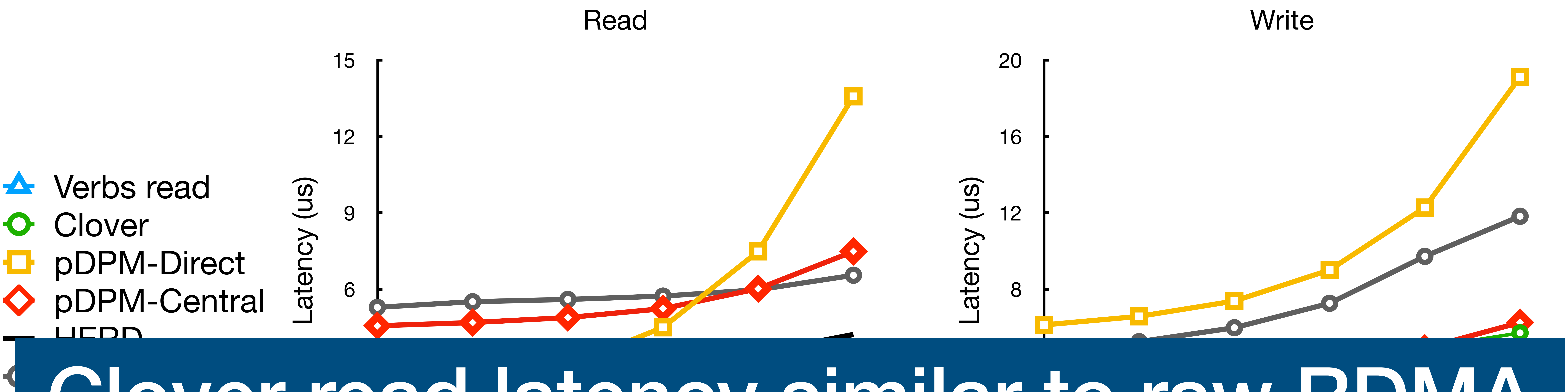
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

Microbenchmark - Latency



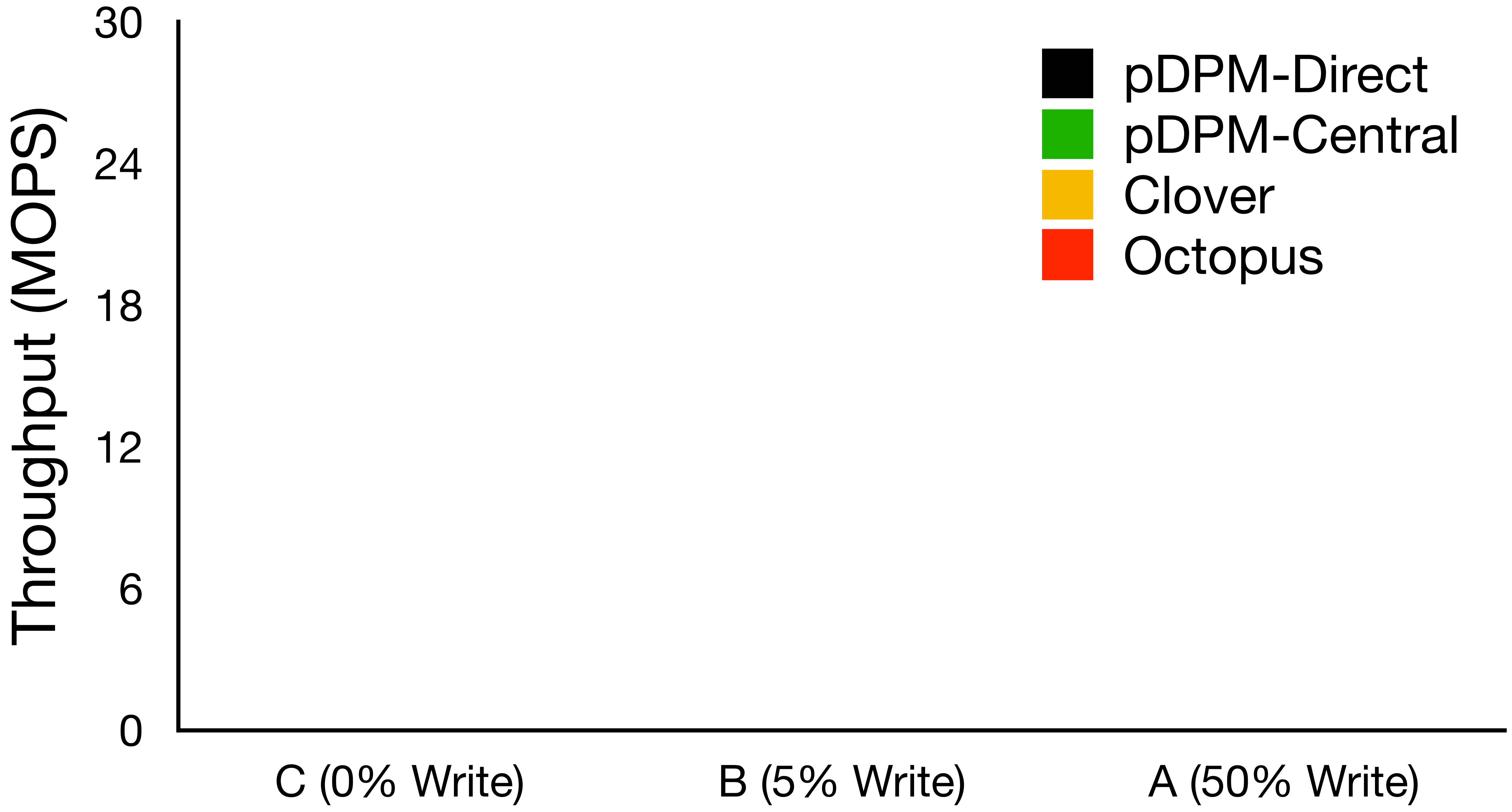
Clover read latency similar to raw RDMA
write latency around 2x of raw RDMA

- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-BF use 12 polling threads

YCSB Results

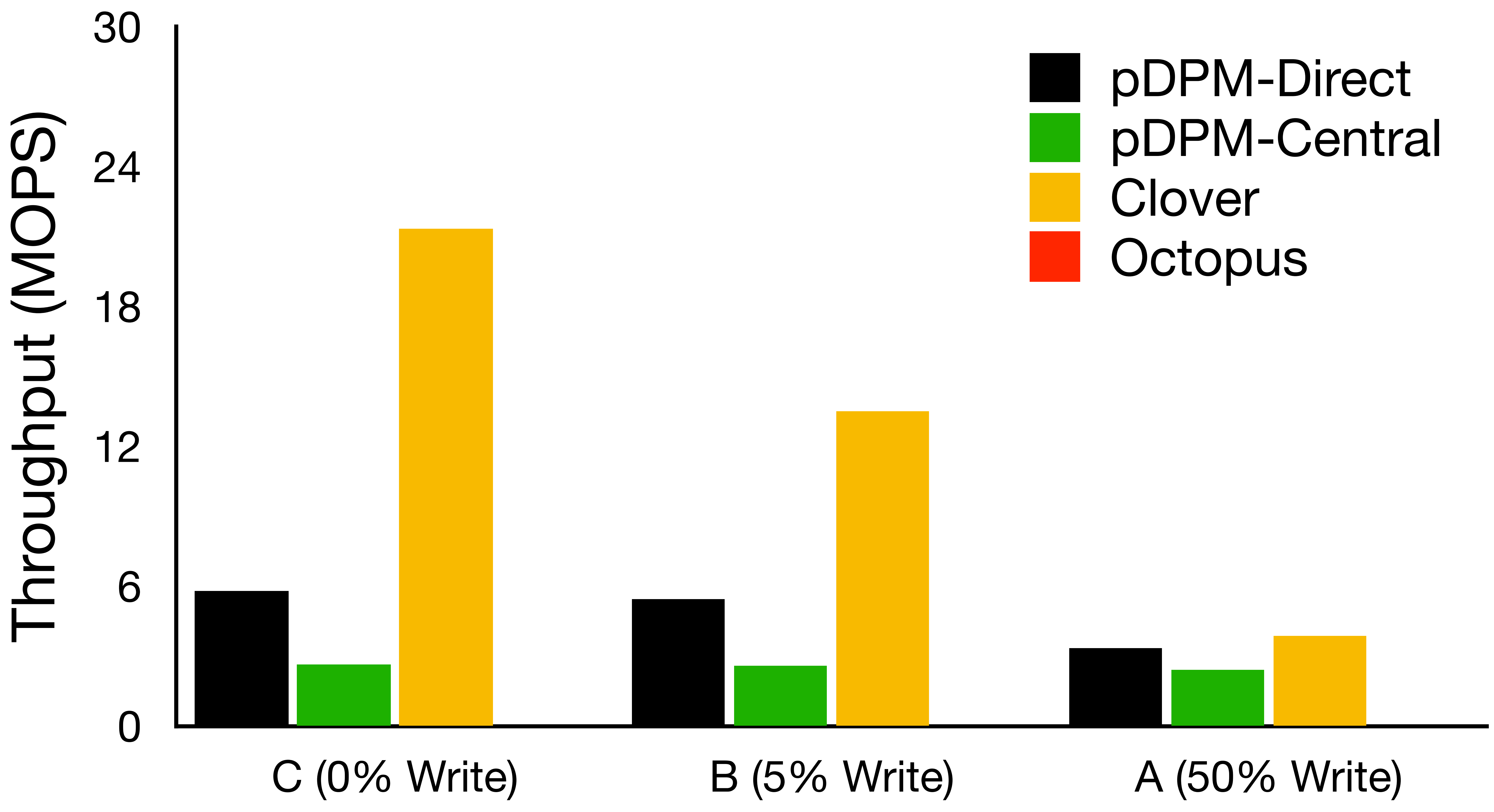
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



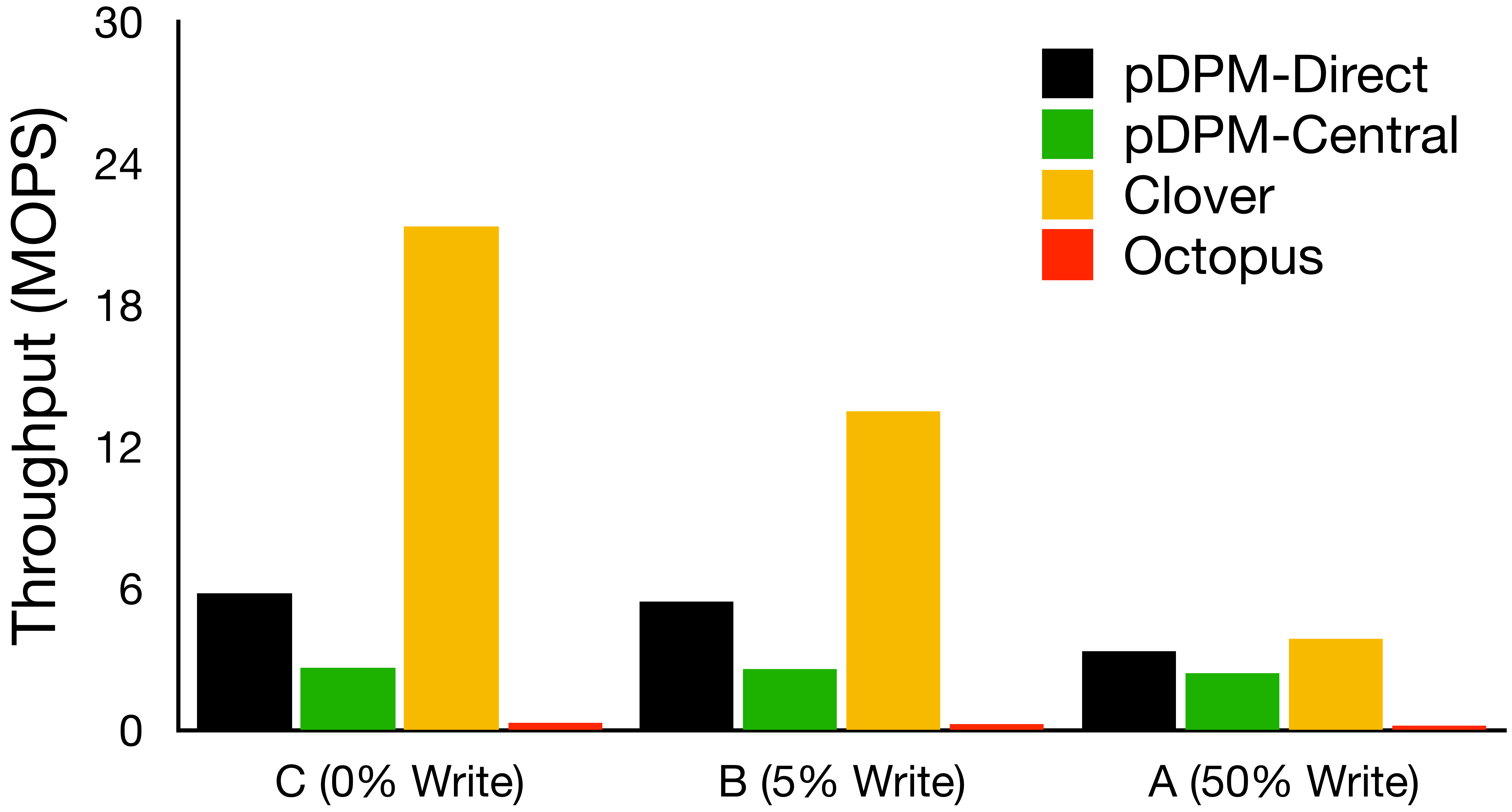
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



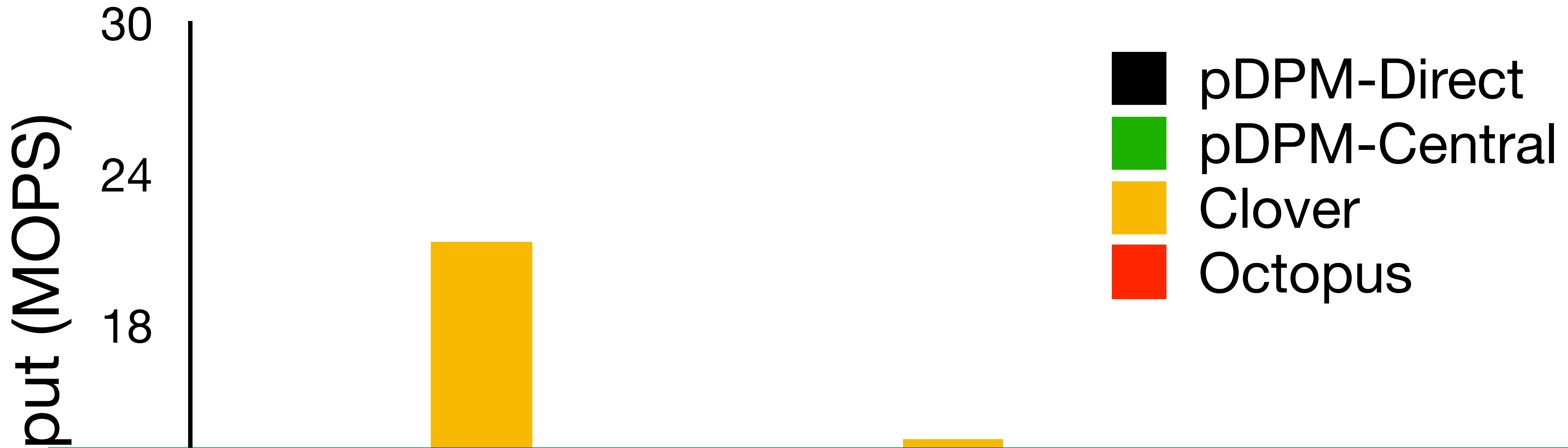
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

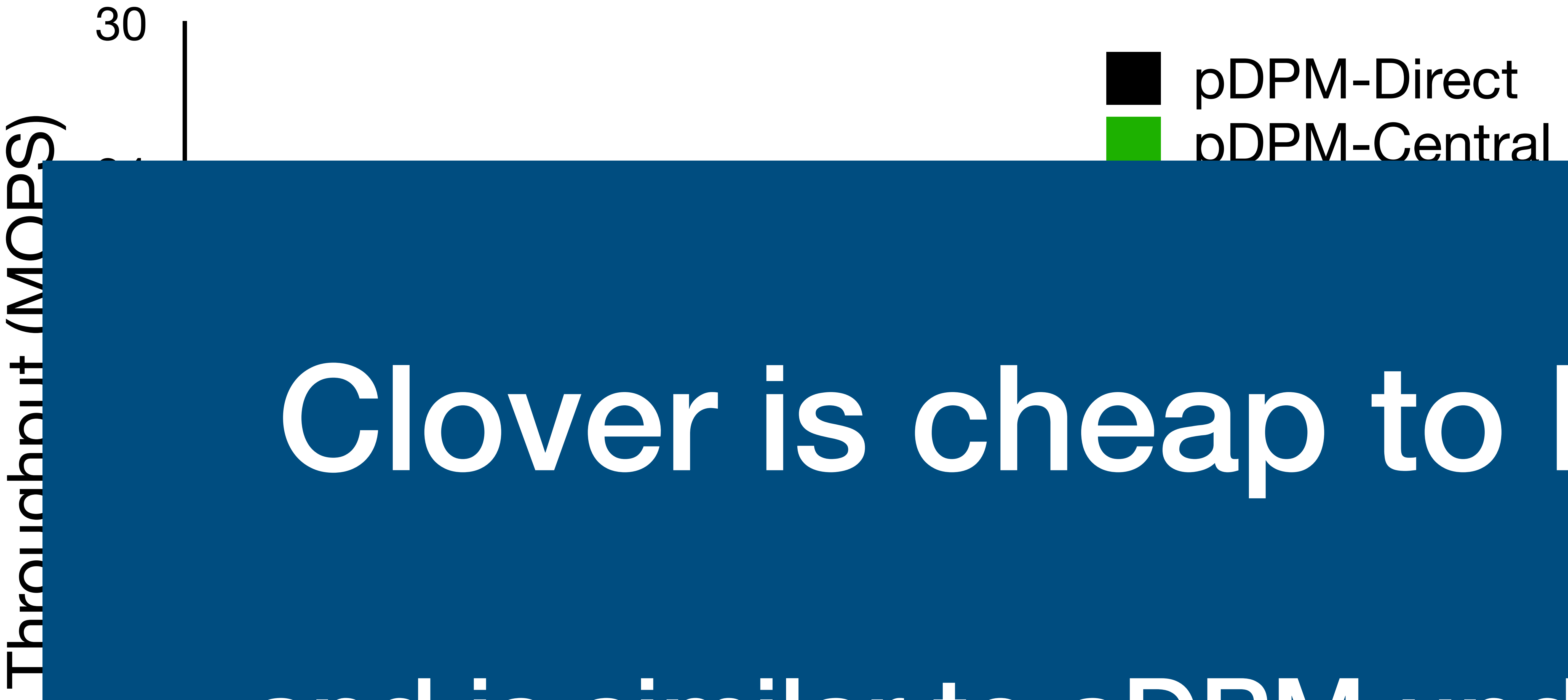
YCSB Results



Clover outperforms non-disaggregated PM systems and is similar to aDPM under common cases (worse under heavy concurrent writes)

- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



Clover is cheap to build and run

and is similar to aDPM under common cases
(worse under heavy concurrent writes)

- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

Conclusion

- pDPM offers deployment, cost, and performance benefits
- Cleanly separating data and metadata is crucial but not easy
- Our pDPM findings could also apply to disaggregated DRAM
- pDPM performs worse under high write contention or complex ops
- Future system could benefit from a hybrid disaggregation model

Thank you!

open source @ github.com/WukLab/pDPM

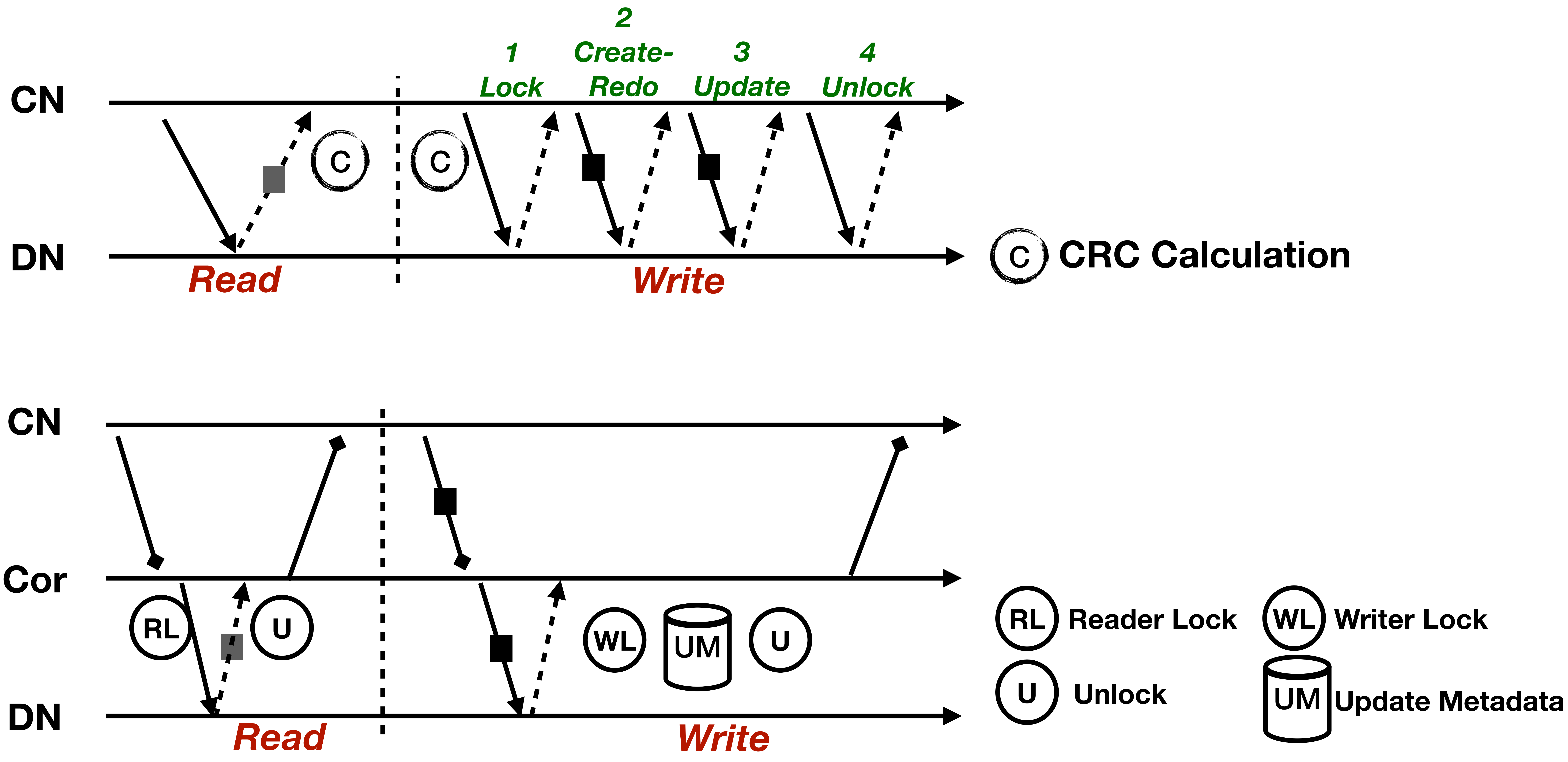
WukLab.io

PURDUE UNIVERSITY

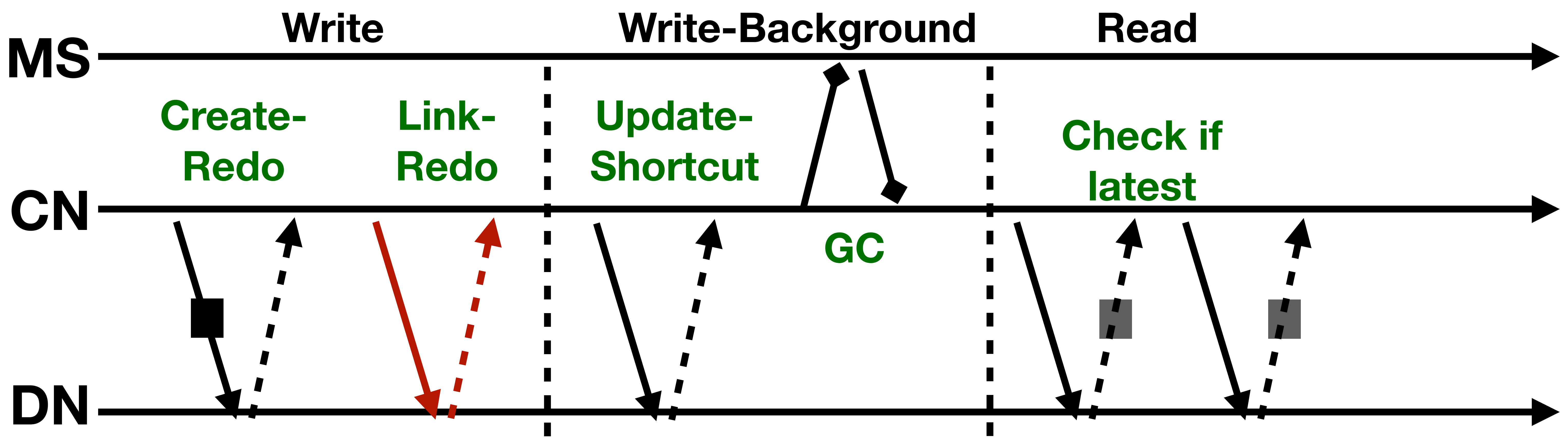
UC San Diego

Backup Slides

pDPM-Direct/Central RW Protocols

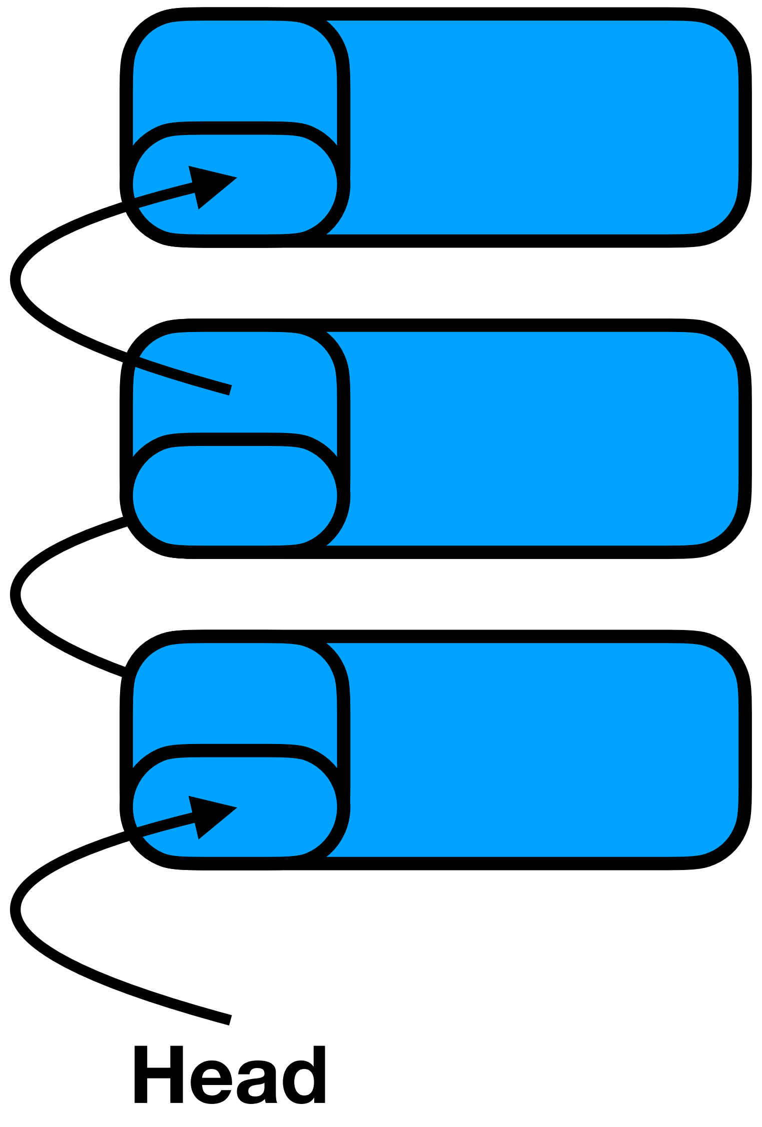


Clover RW Protocols

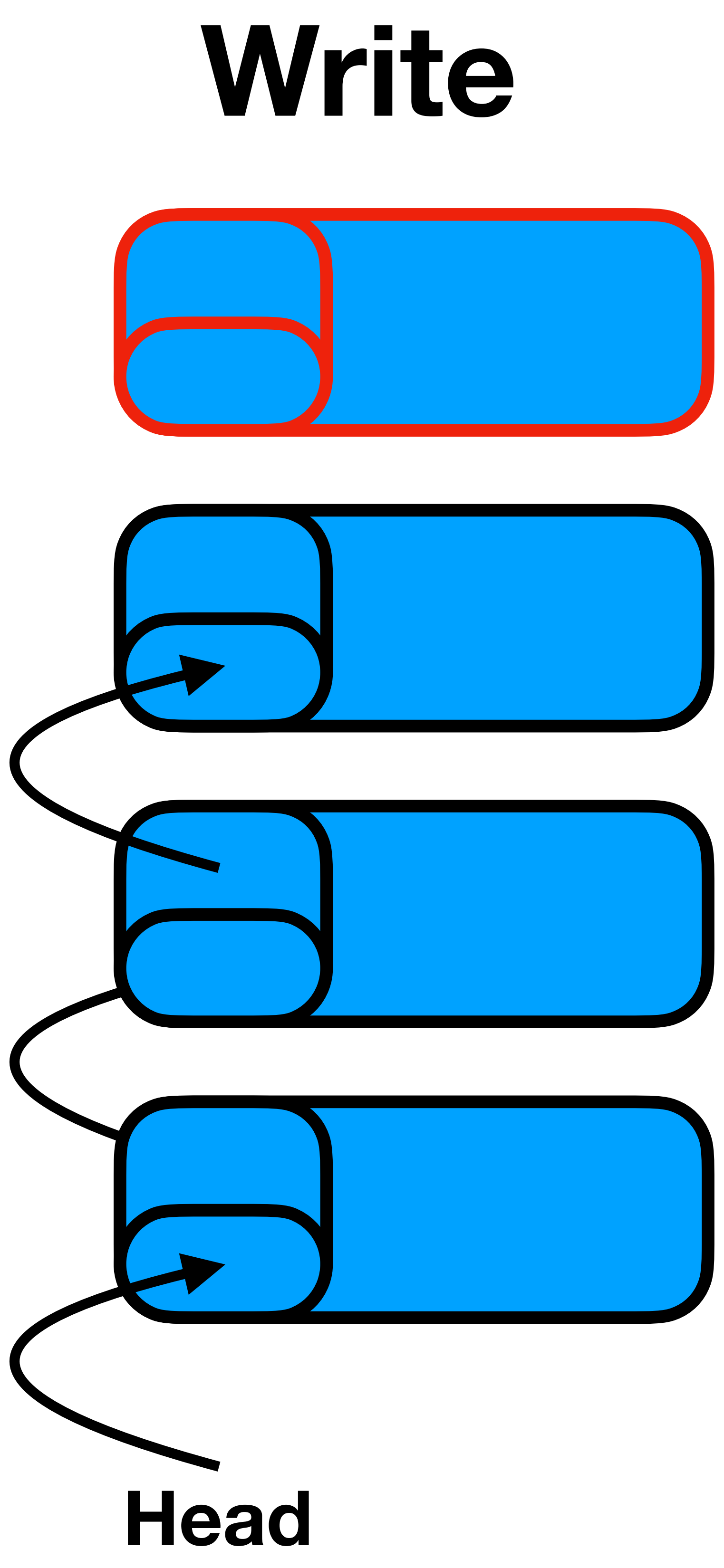


Clover Data Structure

Write

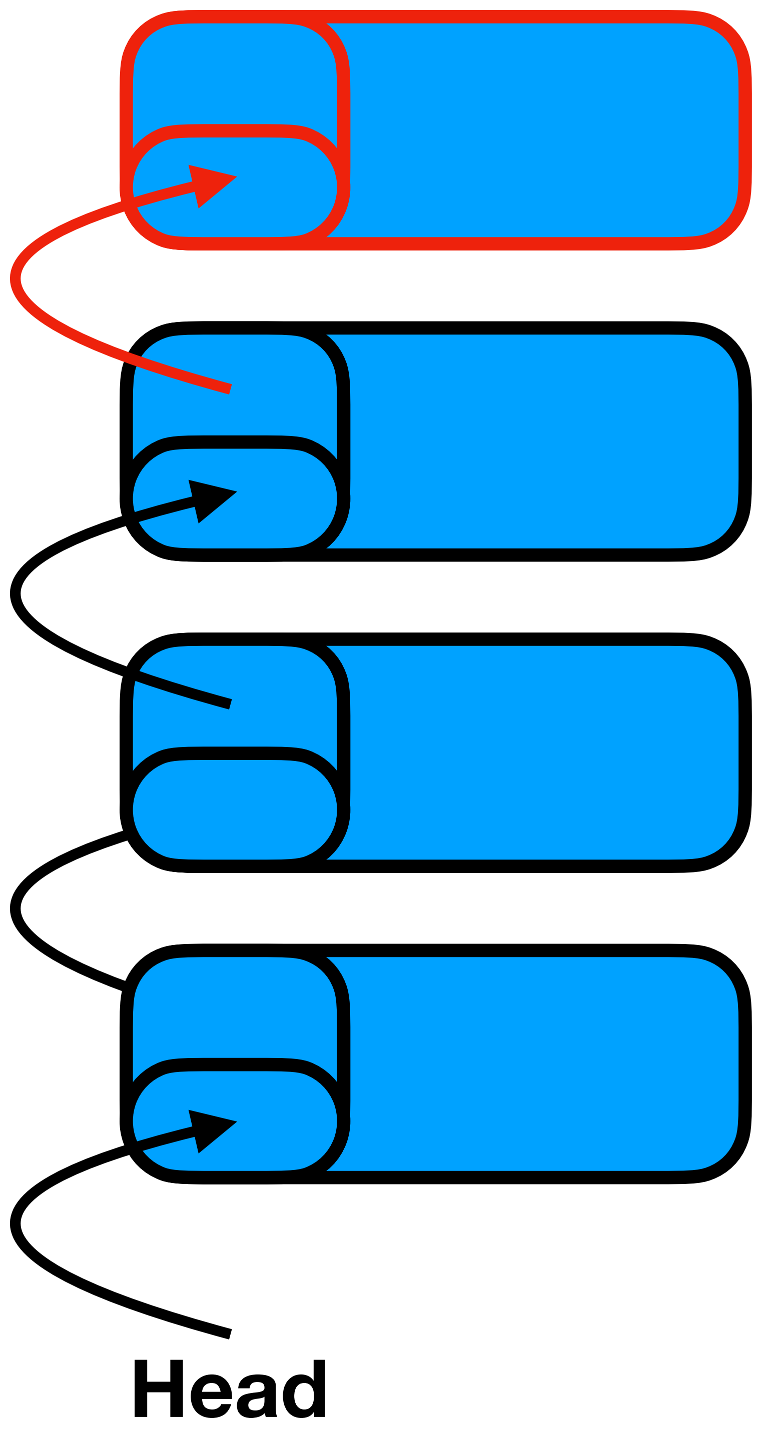


Clover Data Structure

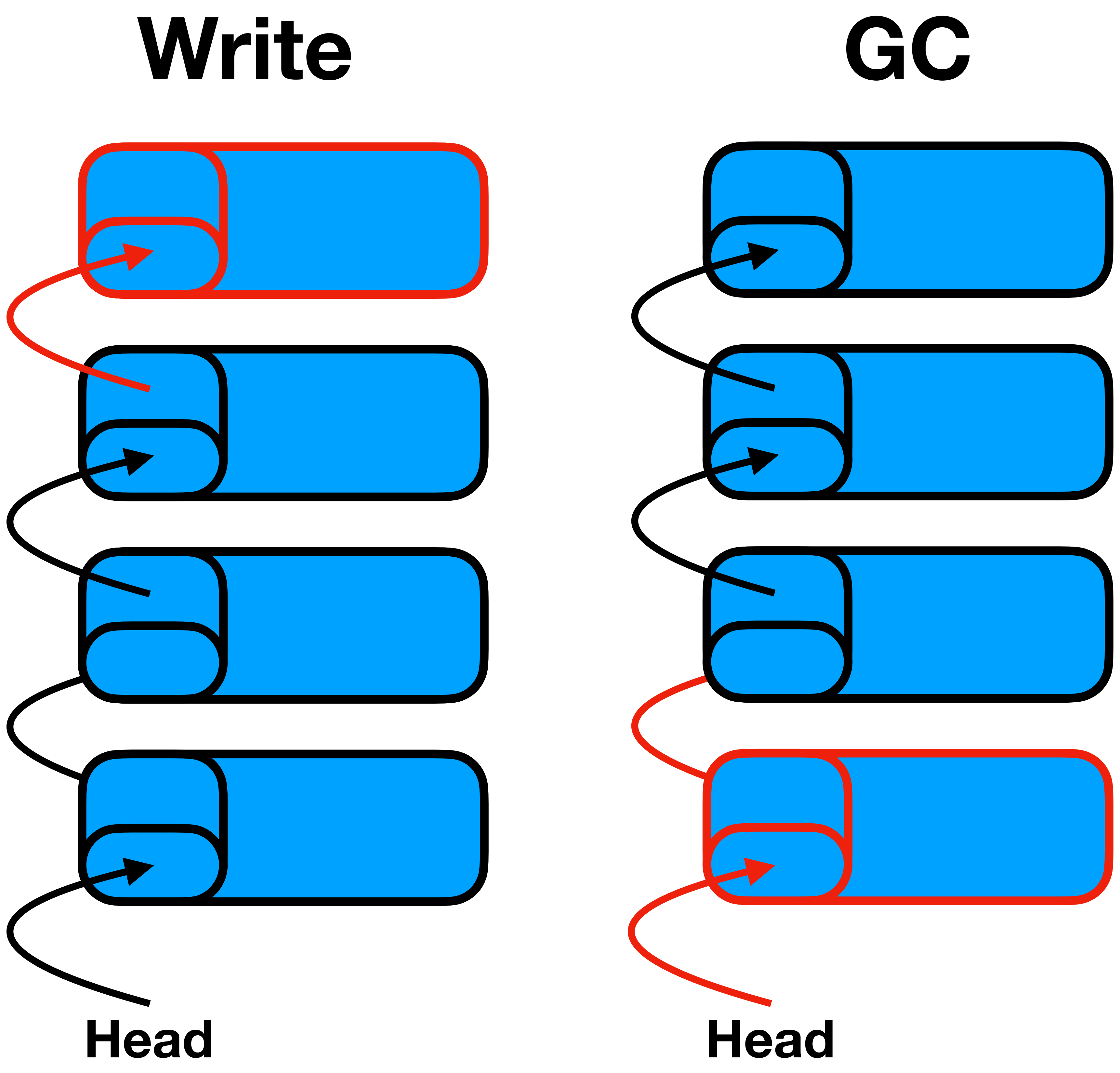


Clover Data Structure

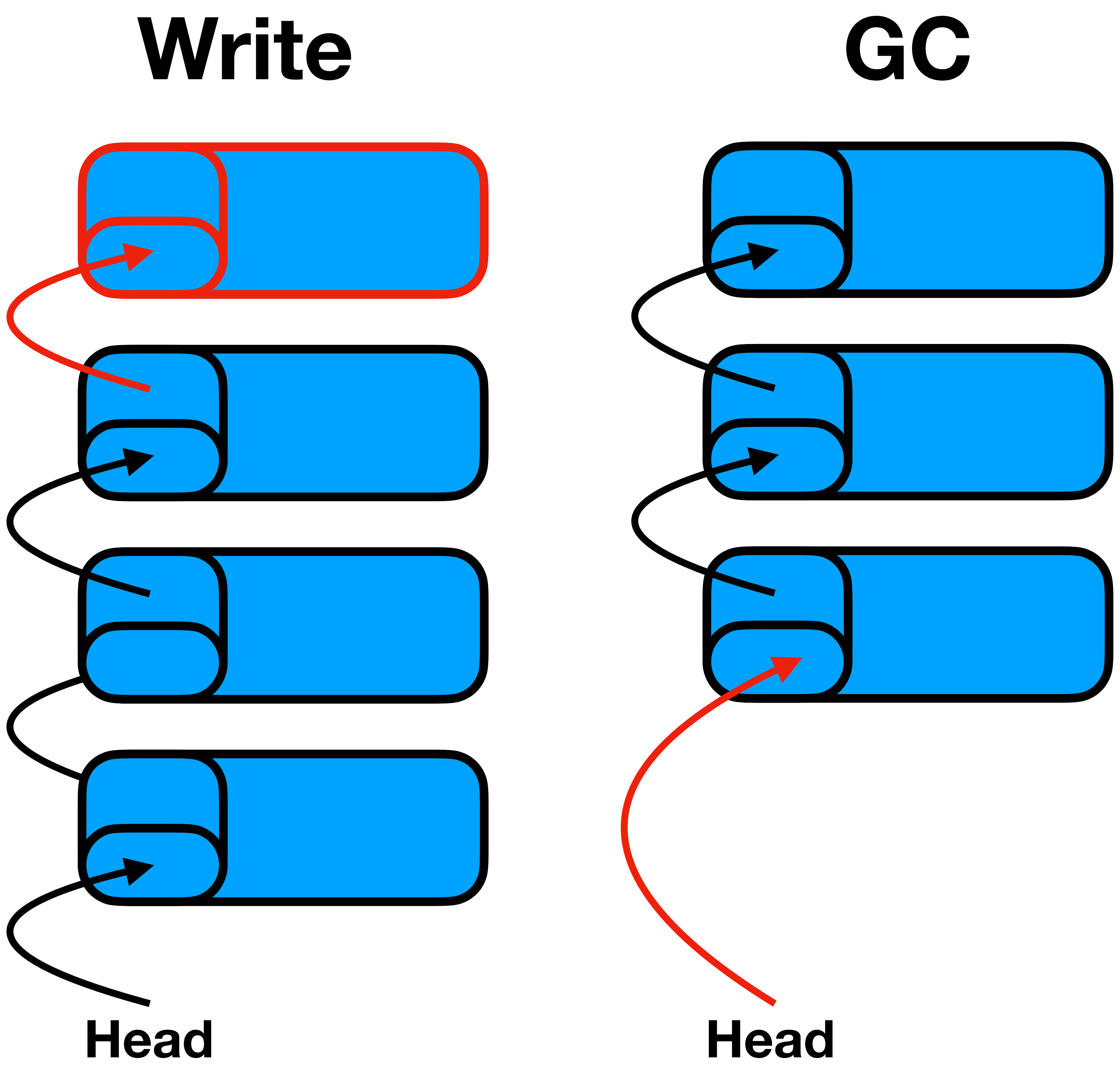
Write



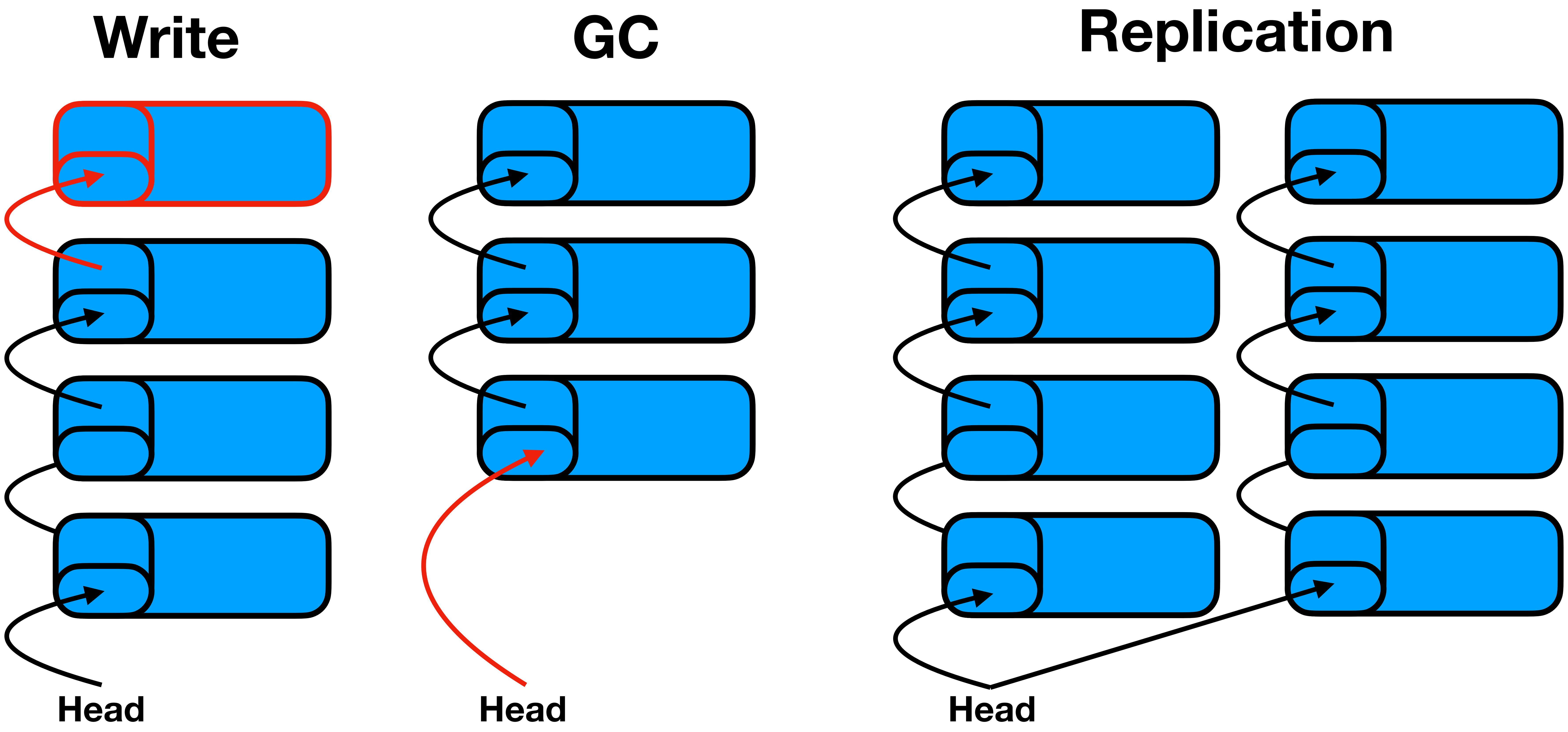
Clover Data Structure



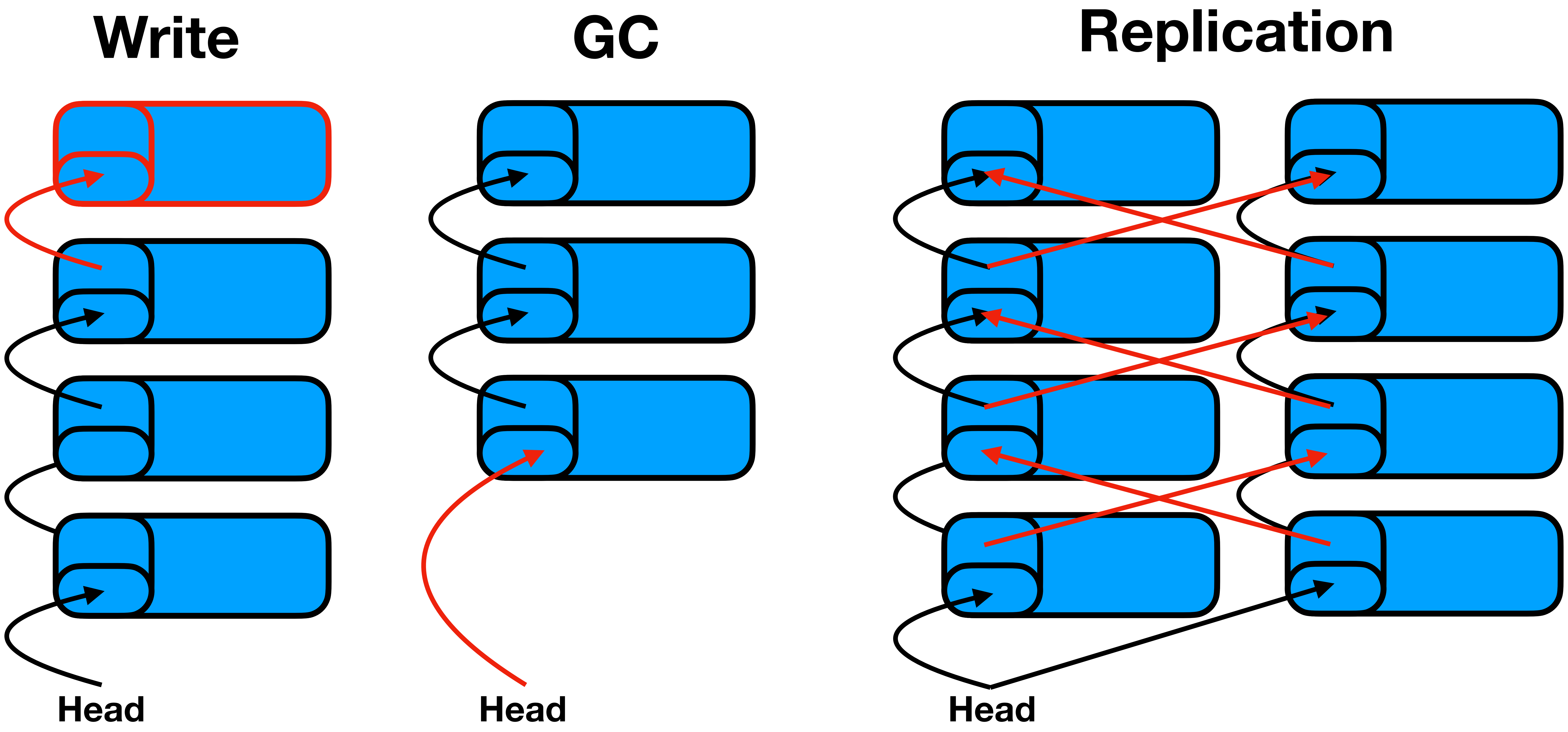
Clover Data Structure



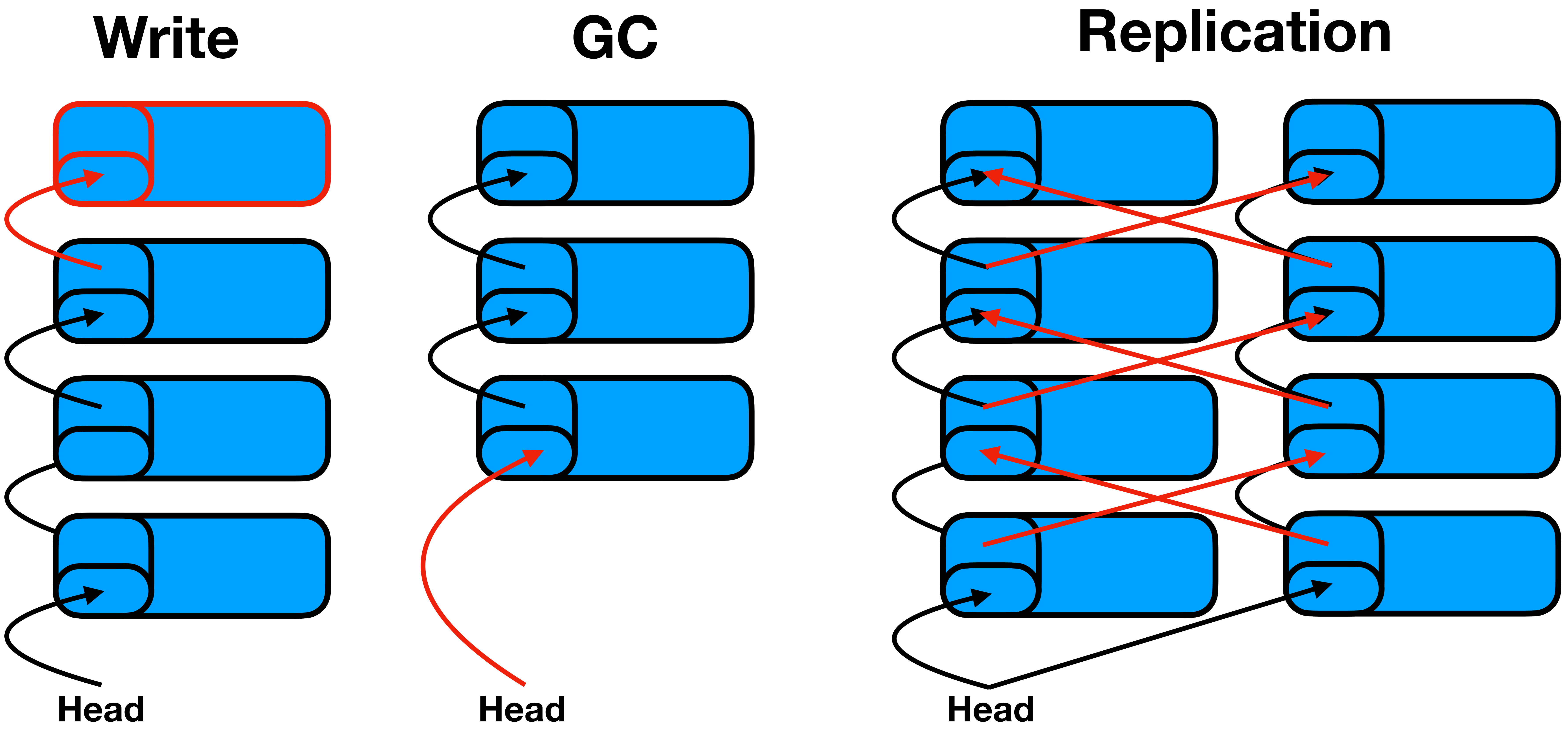
Clover Data Structure



Clover Data Structure



Clover Data Structure



Load Balancing

Where is the key-value hashtable?

- pDPM-Direct: each CN has an identical mapping table
- pDPM-Central: each CN performs CN->coordinator mapping. Each coordinator has a full identical mapping table
- Clover: MSs have full mapping table, each CN caches a portion of it

Possible Questions

- If DPM-Central has multiple coordinates, cannot it scale?
- Why not use read-after-write to ensure remote persistency?
- Where is the key-> entry hashtable?
  - The whole table is at MS, each CN caches a portion of it?