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

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

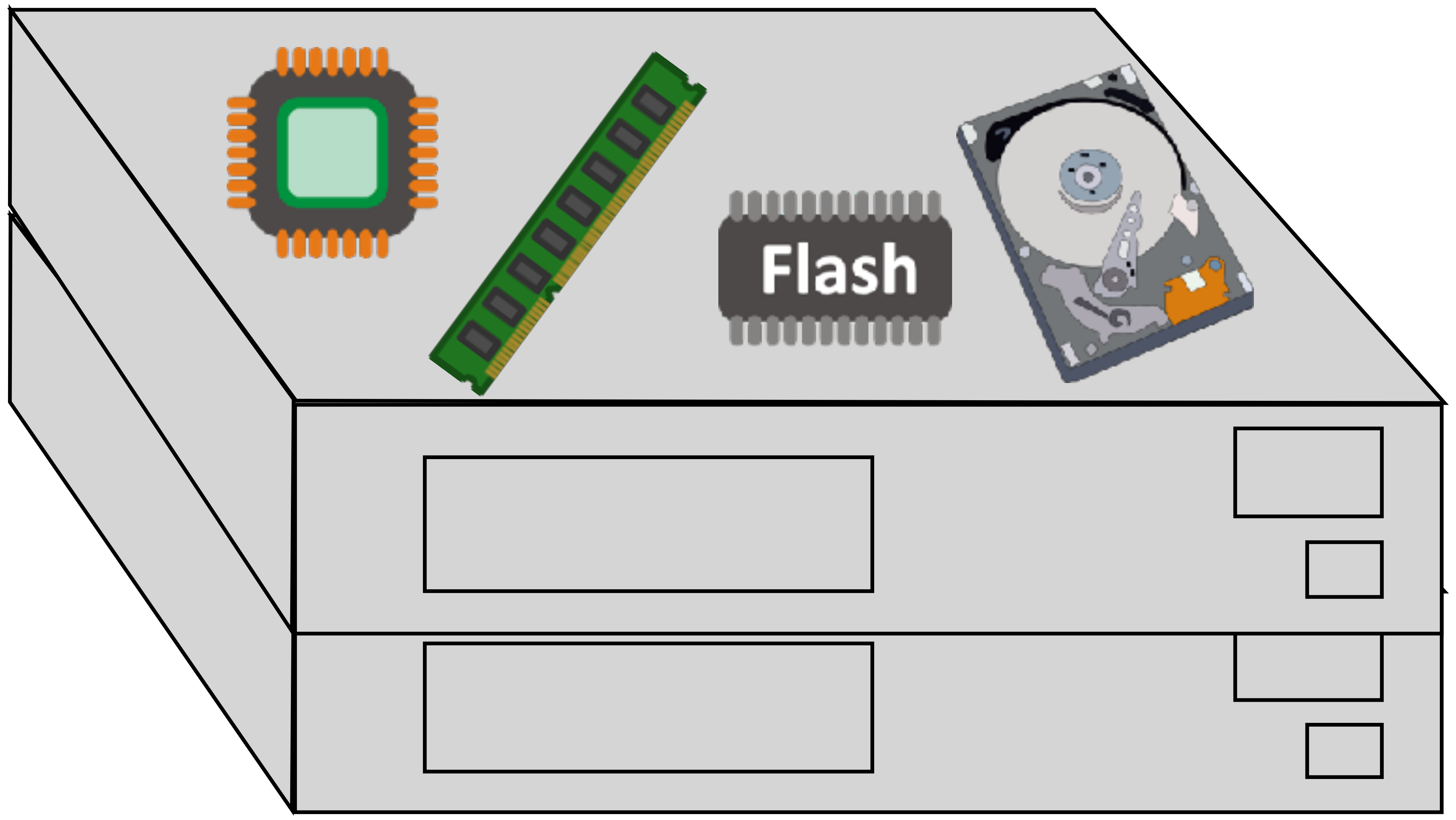
PURDUE UNIVERSITY

WukLab

UC San Diego

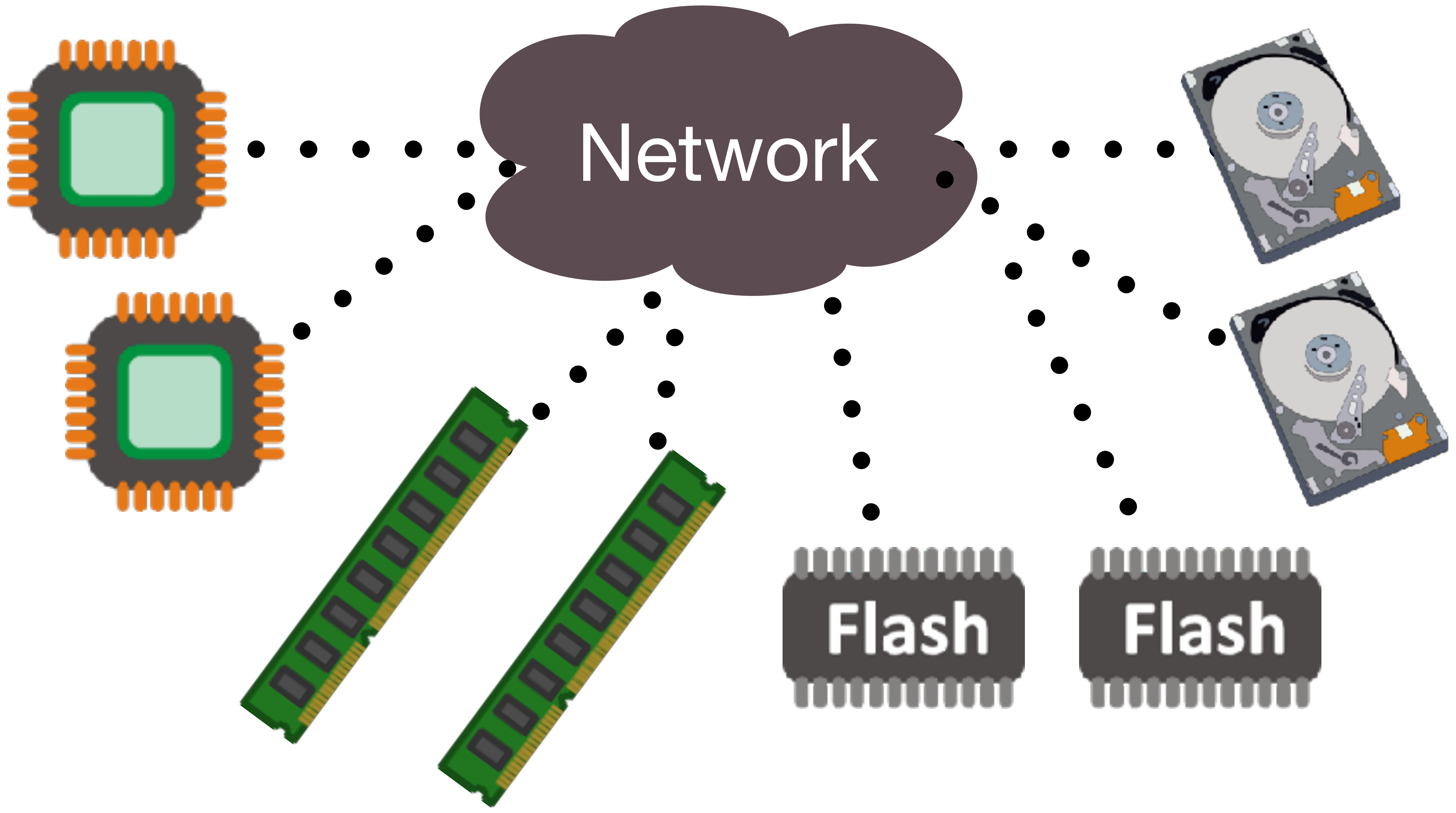
Resource Disaggregation

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



Resource Disaggregation

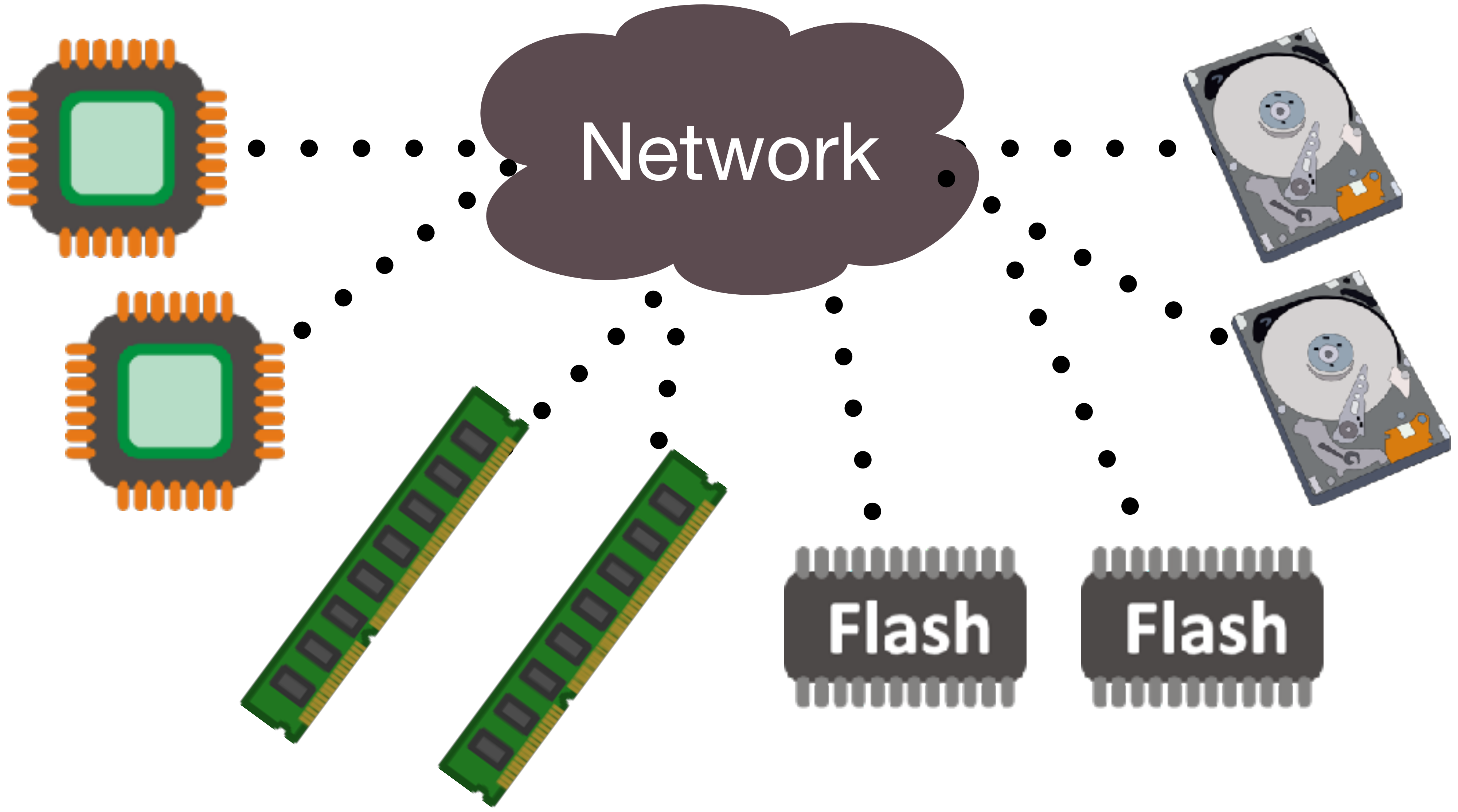
Break monolithic servers into ***network-attached*** resource pools



Resource Disaggregation

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

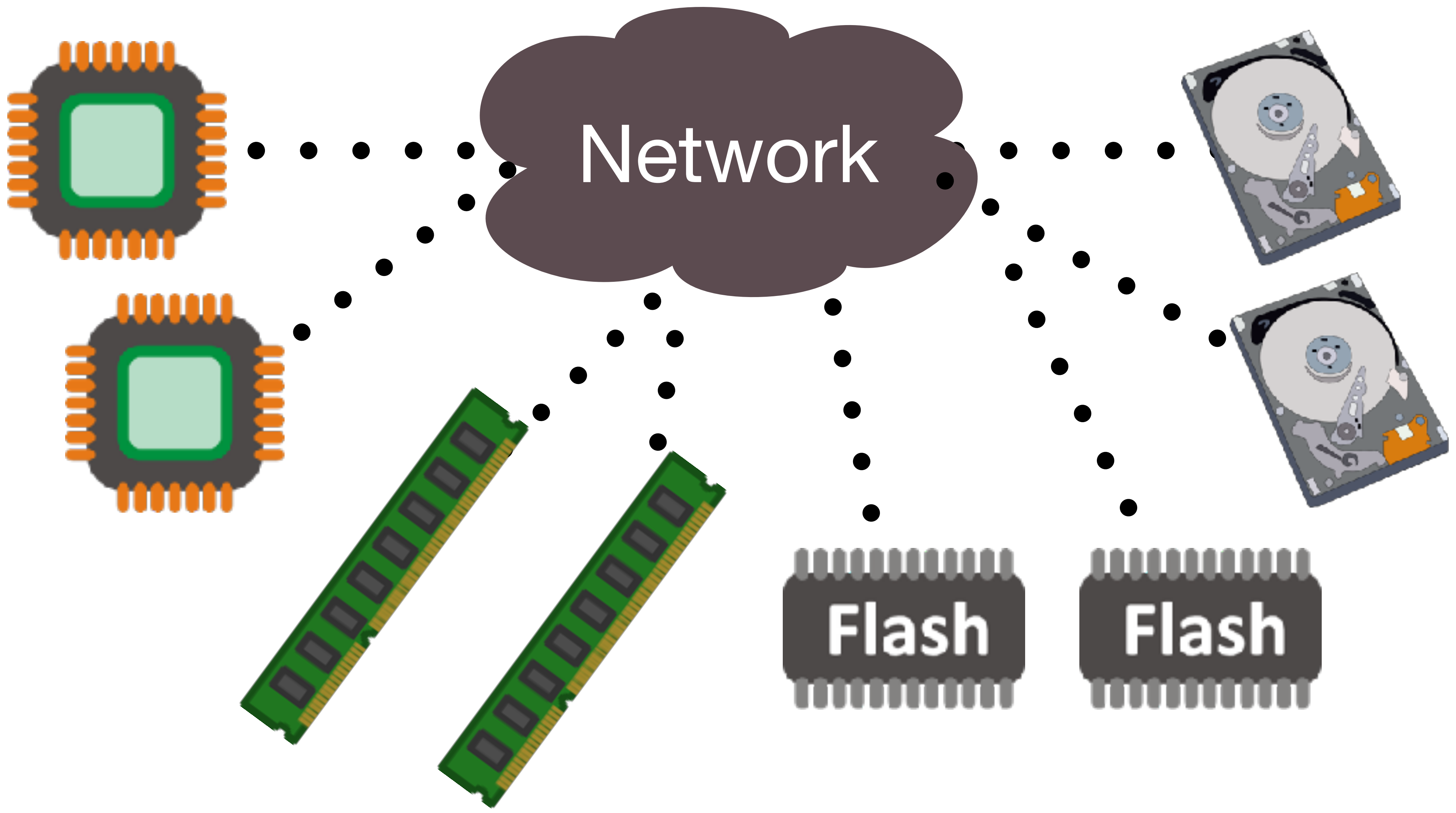
Better manageability, independent scaling, tight resource packing



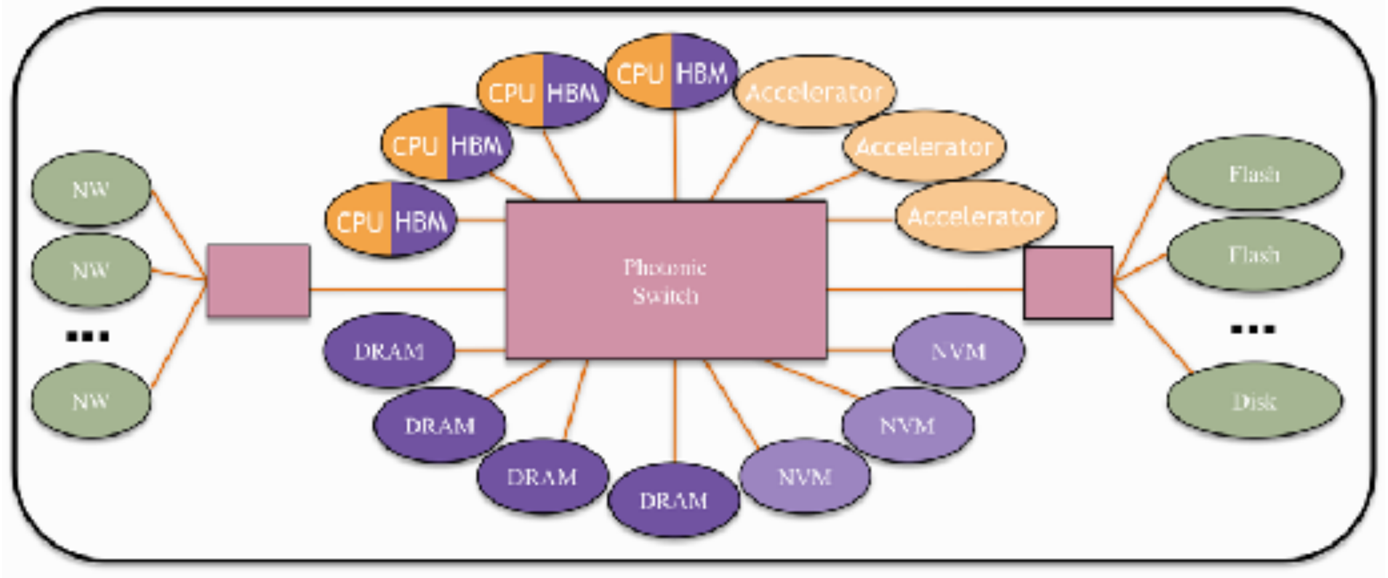
Resource Disaggregation

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

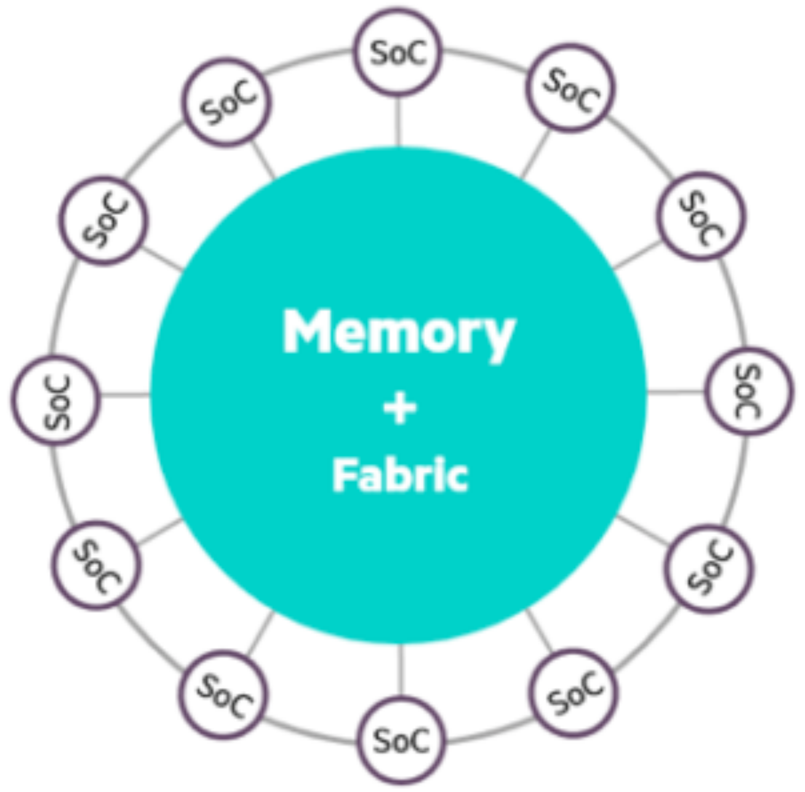
Better manageability, independent scaling, tight resource packing



LegoOS



Berkeley Firebox



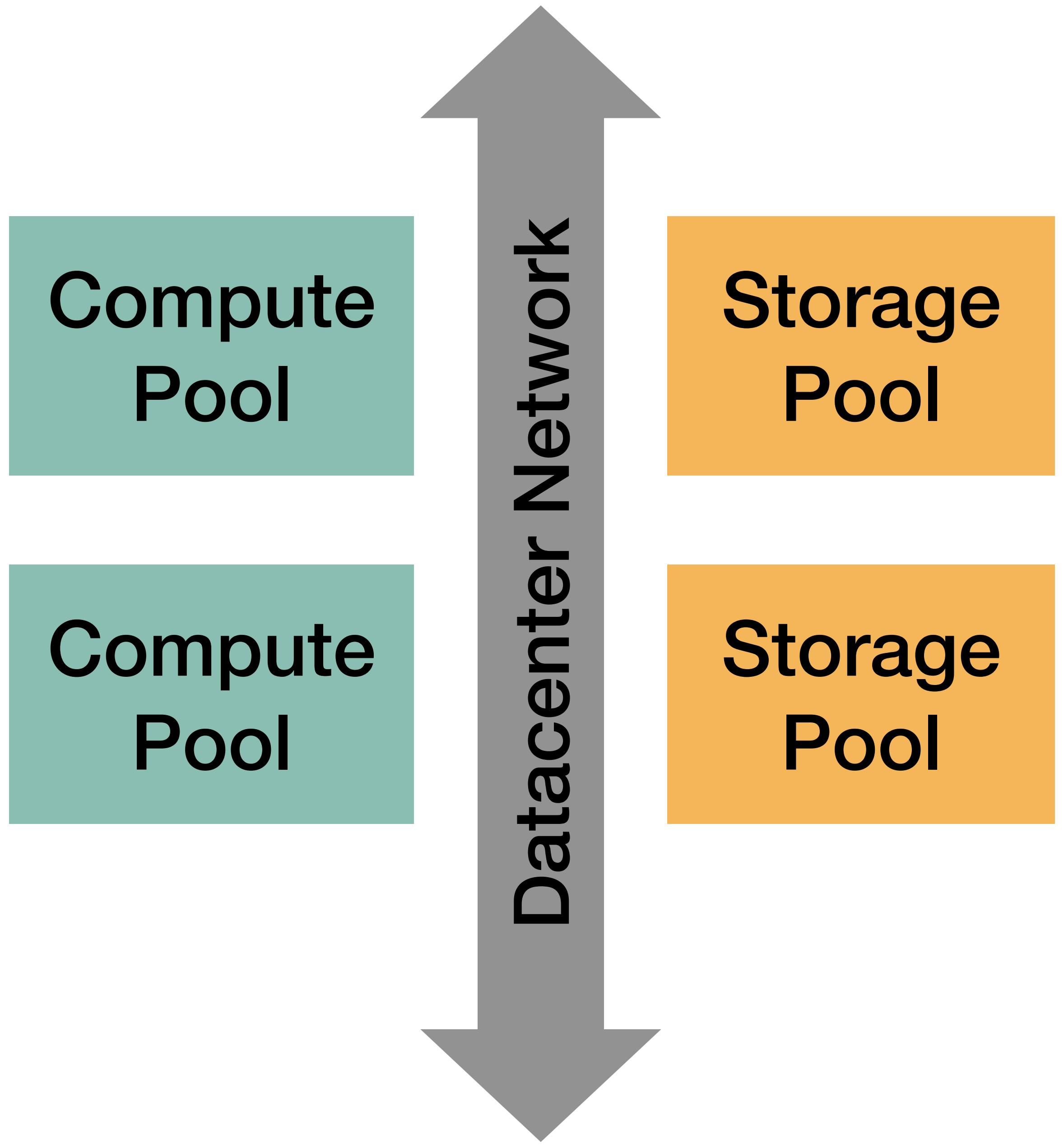
Hewlett Packard
Enterprise

Disaggregated Storage

Separate compute and storage pools

- Manage and scale independently

A common practice in datacenters and clouds

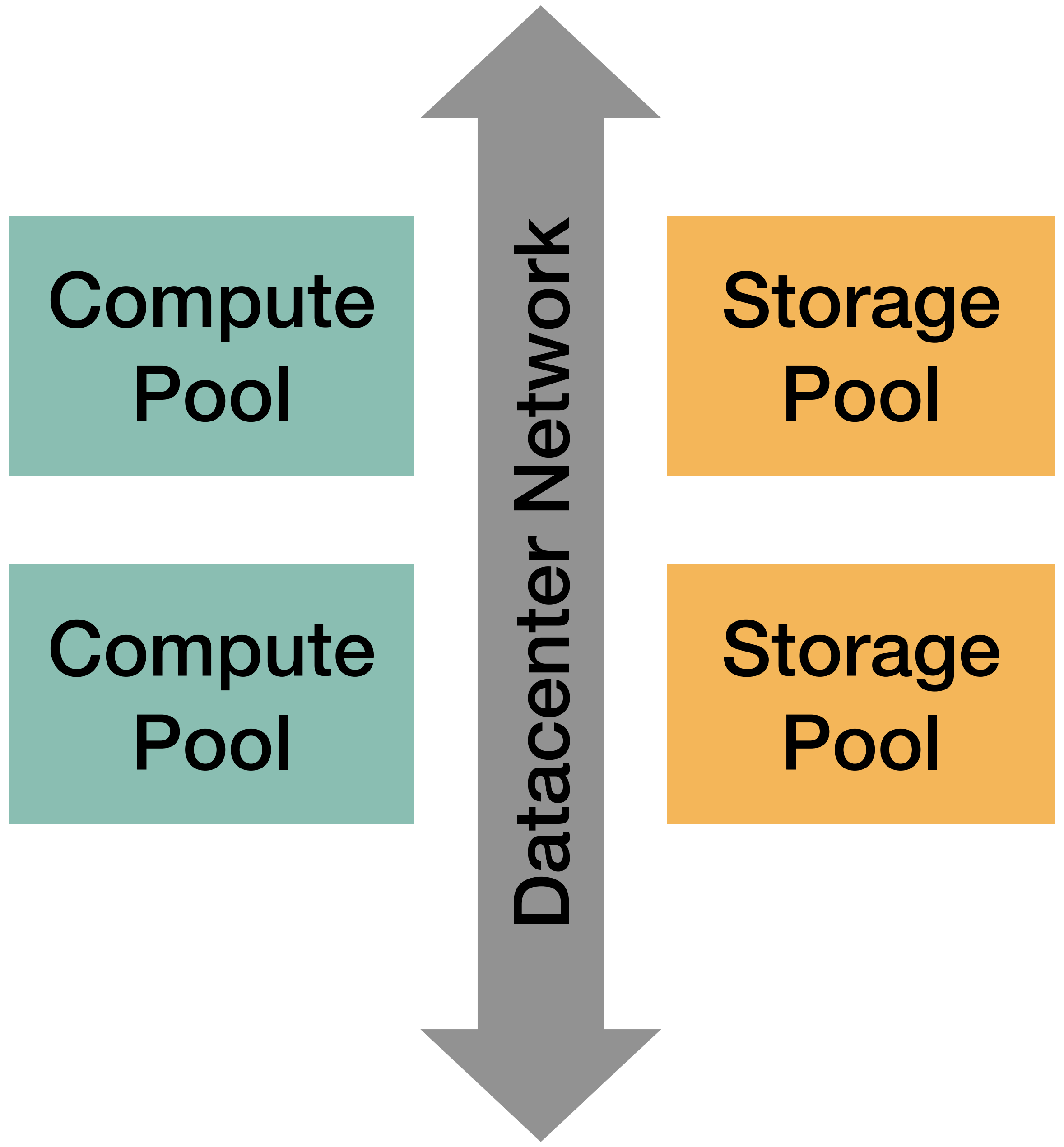


Disaggregated Storage

Separate compute and storage pools

- Manage and scale independently

A common practice in datacenters and clouds



amazon web services™ S3

Powered By vmware vSAN™

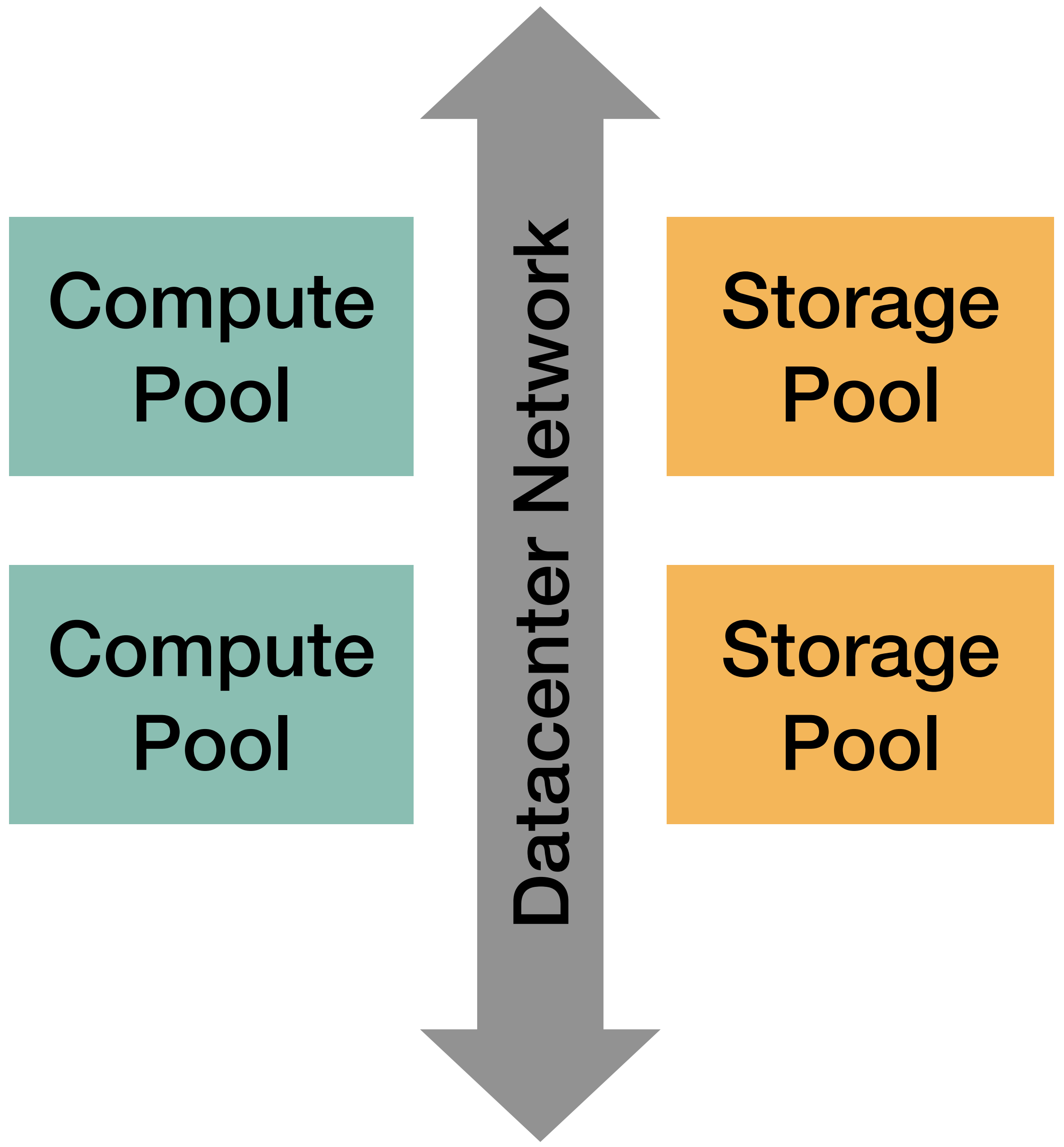
snowflake

Disaggregated Storage

Separate compute and storage pools

- Manage and scale independently

A common practice in datacenters and clouds



amazon web services™ | S3

facebook Engineering

Powered By vmware vSAN™

snowflake

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

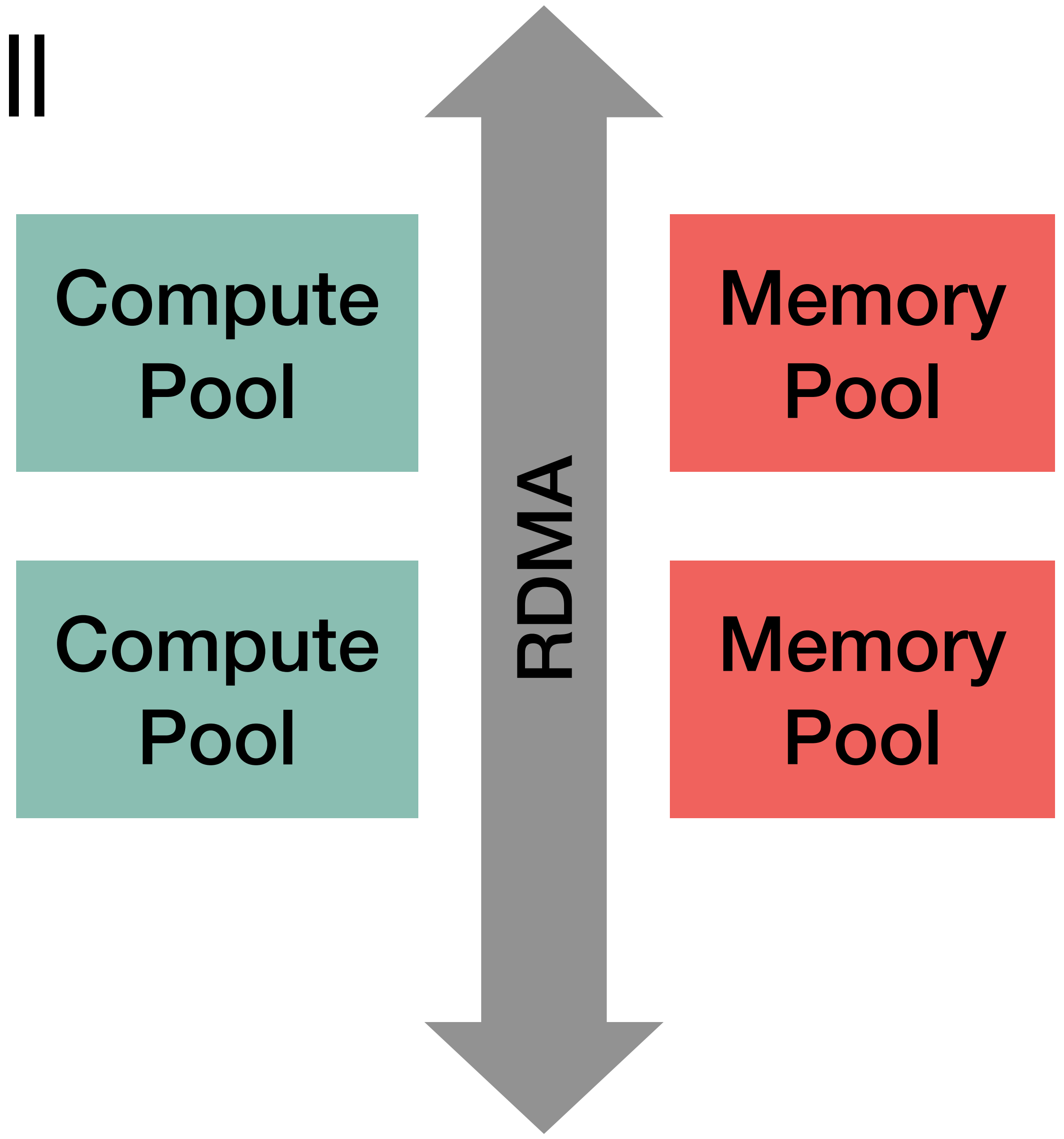
Yash Wate

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

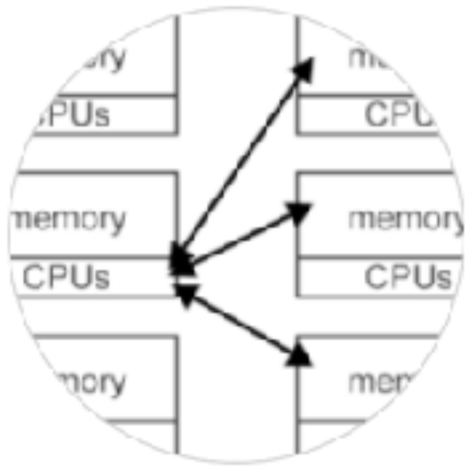


Microsoft

FaRM

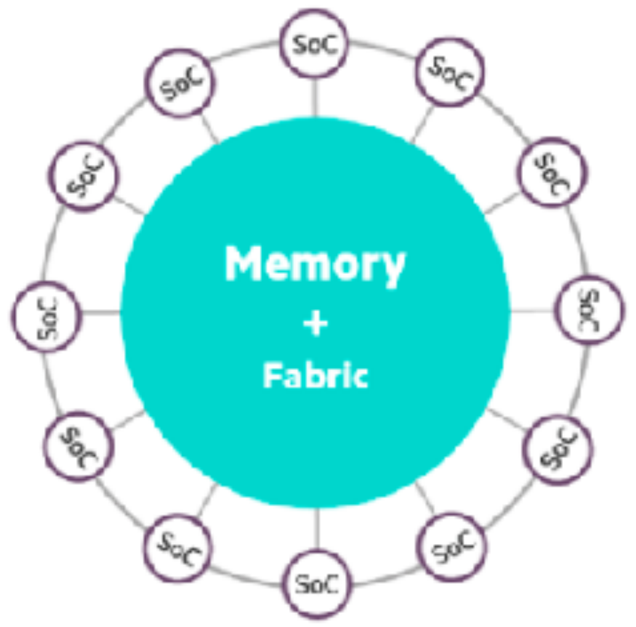
vmware

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?

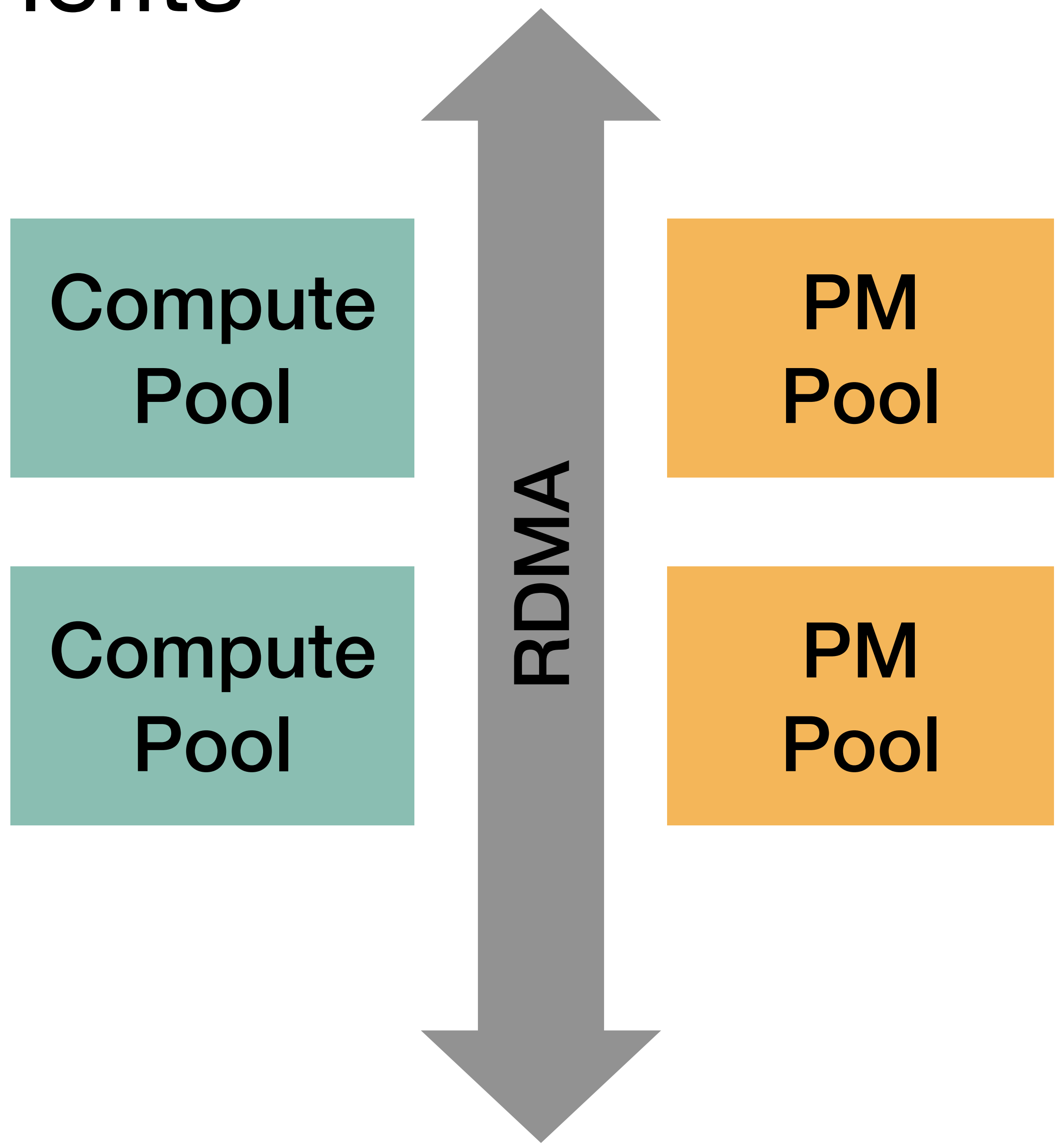
PM: byte-addressable, persistent, memory-like perf

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?

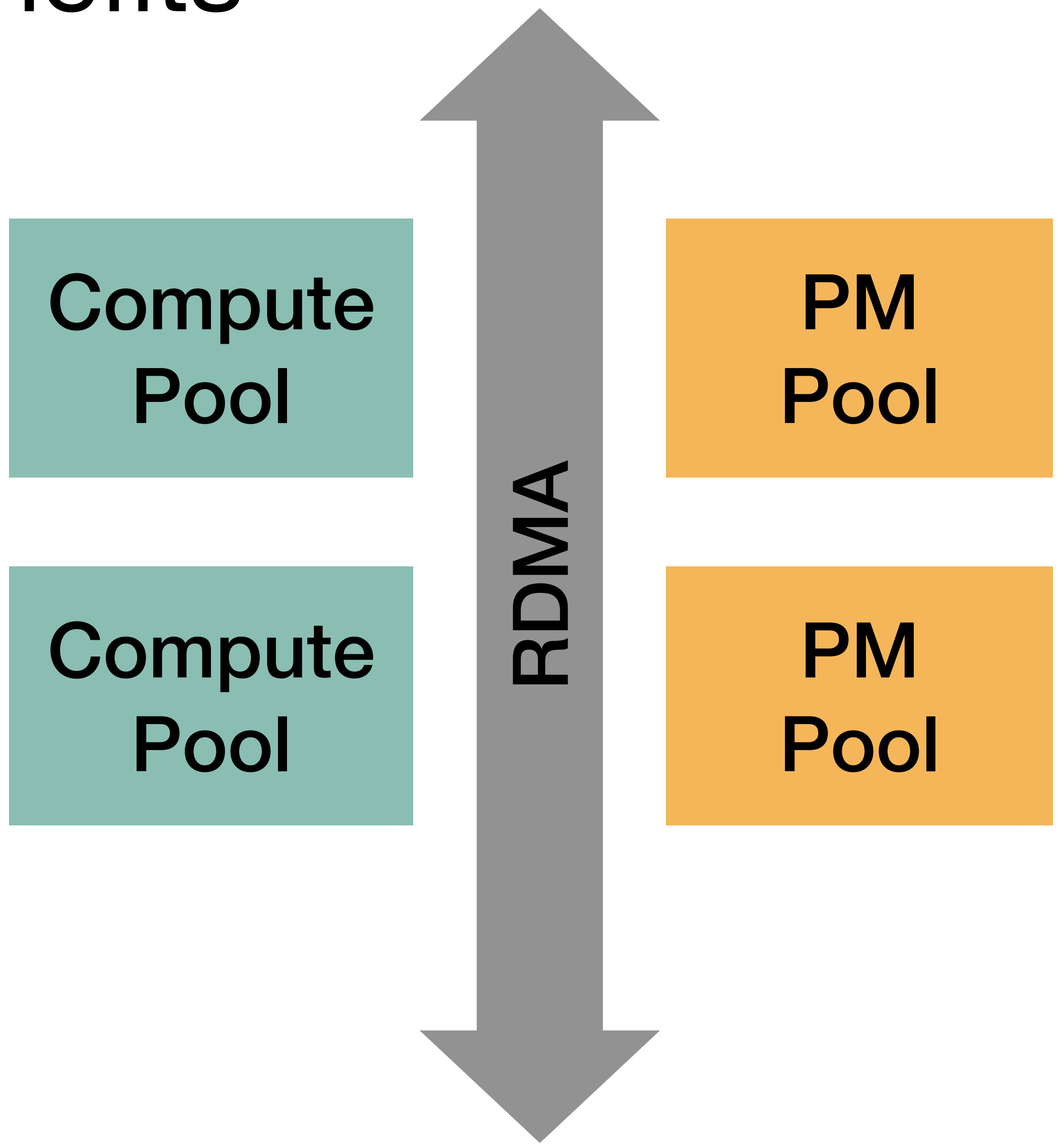
PM: byte-addressable, persistent, memory-like perf

Disaggregating PM (DPM)

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

Use existing disaggregated systems for DPM?

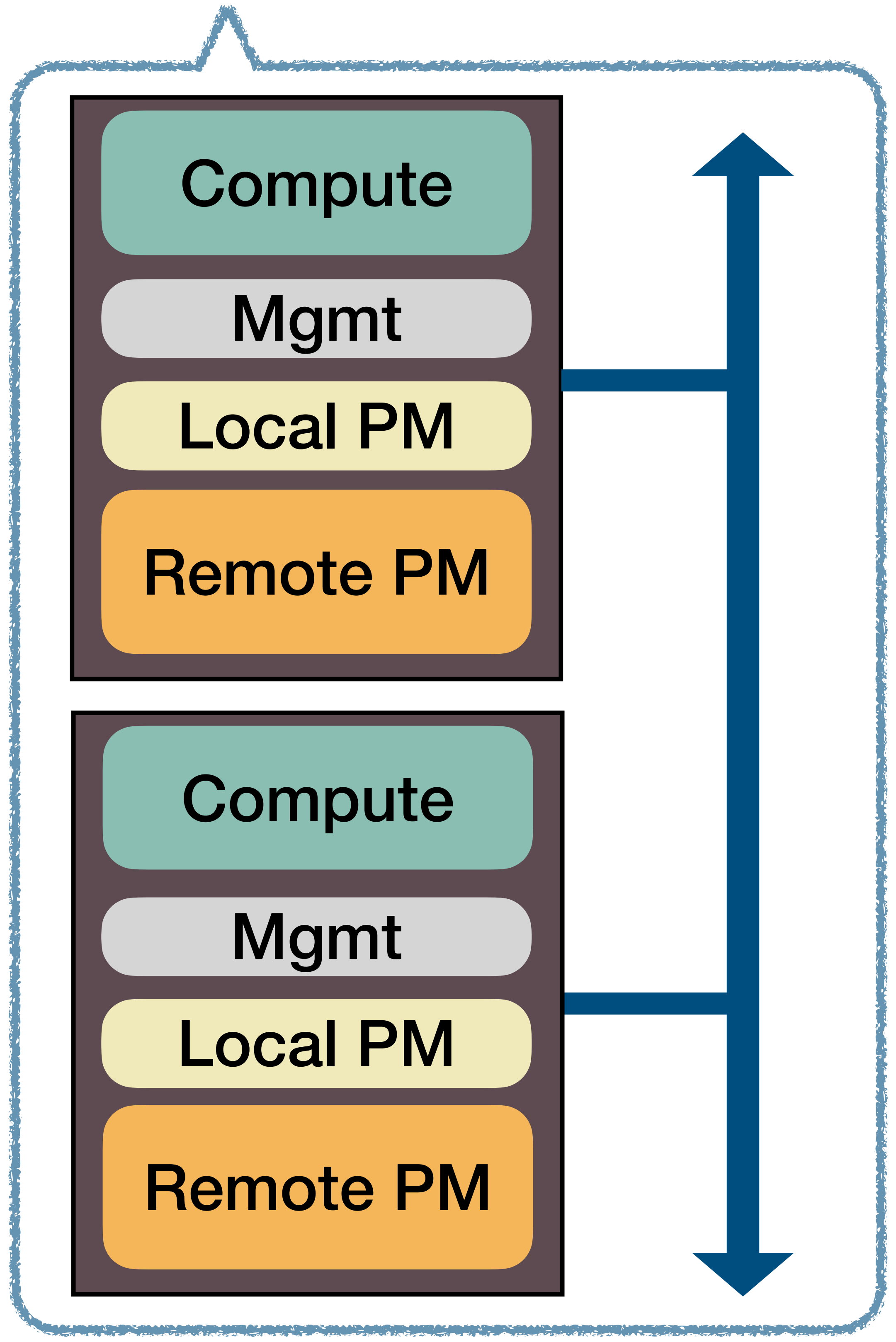
- Disaggregated storage: software stack too slow for fast PM
- Disaggregated memory: do not provide data reliability



Spectrum of Datacenter PM Deploy Models

Spectrum of Datacenter PM Deploy Models

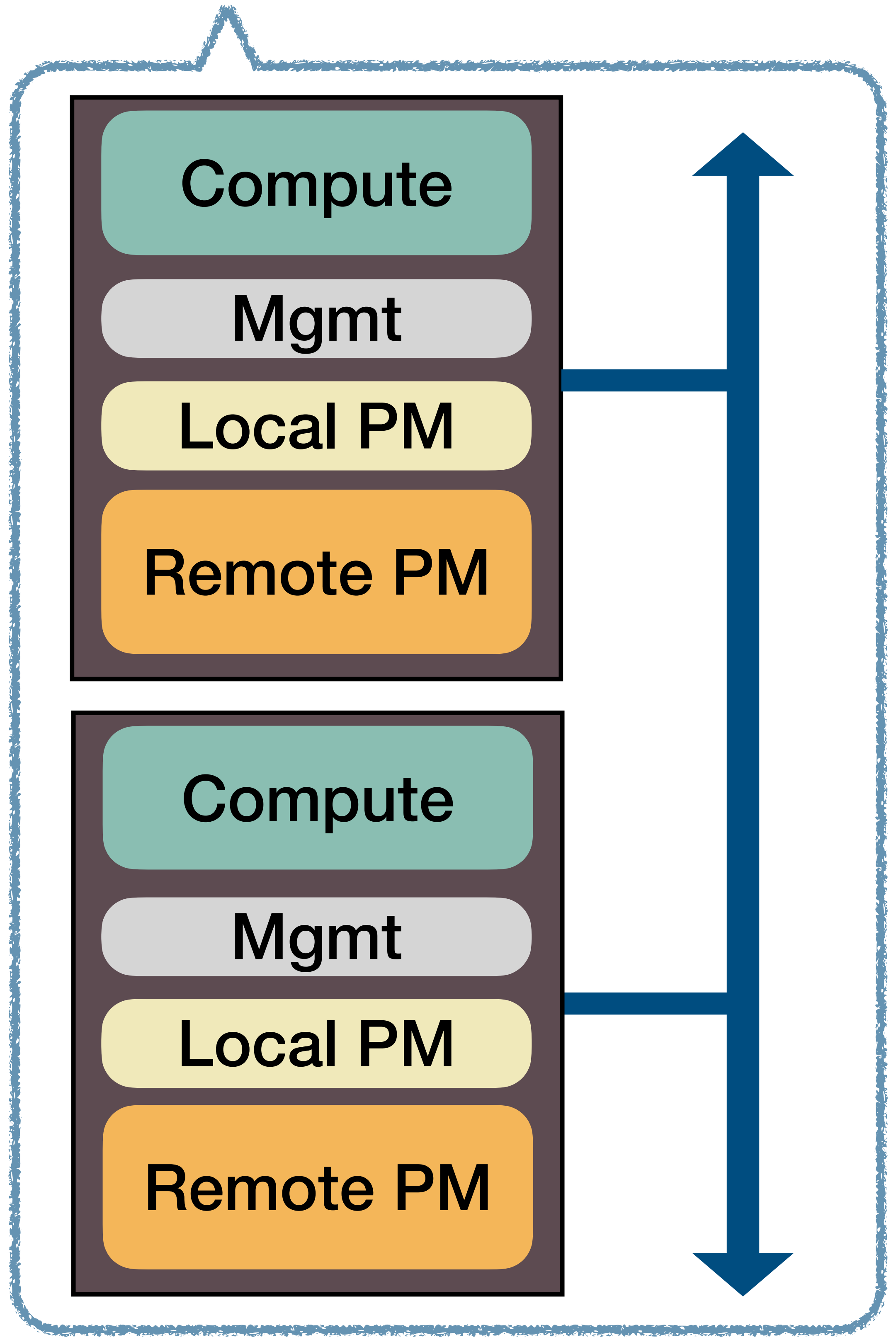
Non-Disaggregation



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

Spectrum of Datacenter PM Deploy Models

Non-Disaggregation

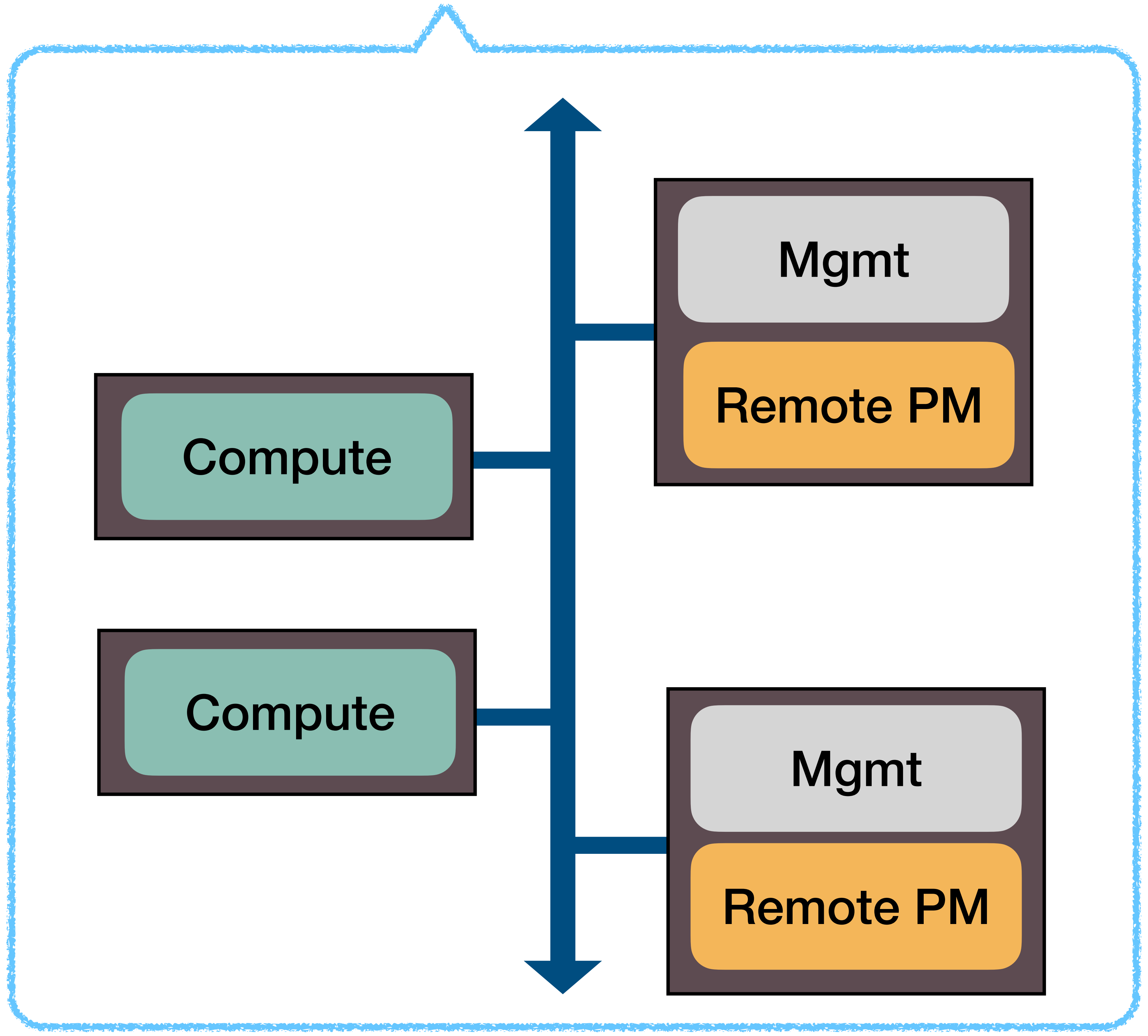


Compute
Mgmt
Local PM
Remote PM

Compute
Mgmt
Local PM
Remote PM

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

Active Disaggregation



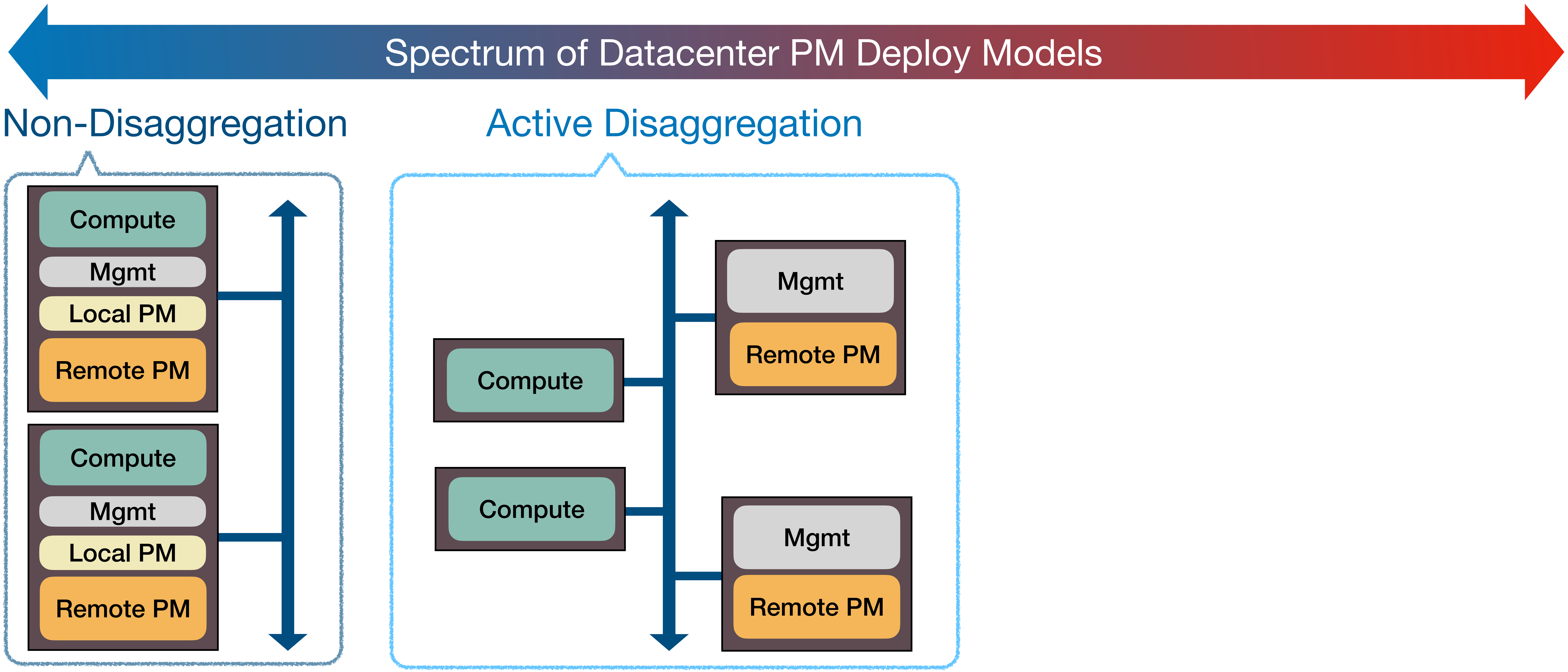
Compute
Compute

Mgmt
Remote PM

Mgmt
Remote PM

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

Traditional Storage Systems



Spectrum of Datacenter PM Deploy Models

Non-Disaggregation

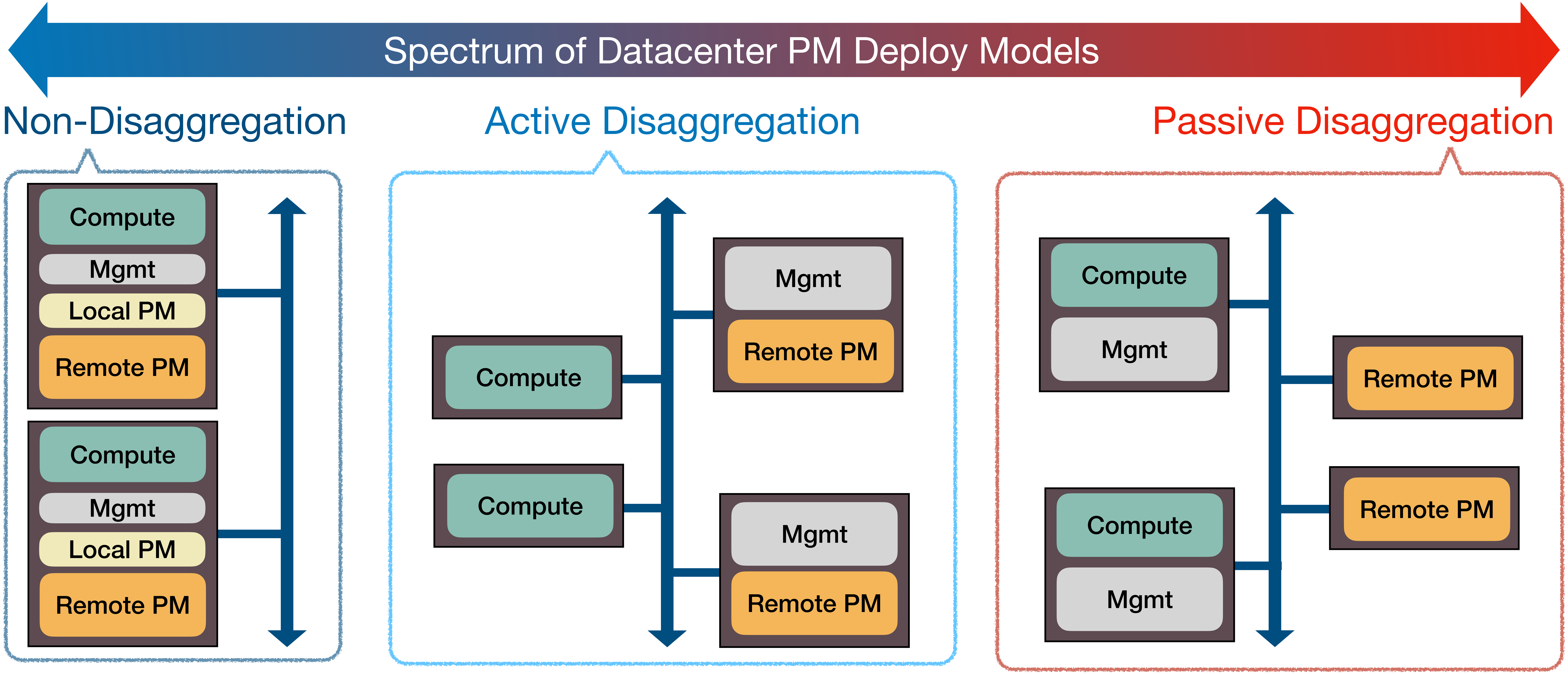
Active Disaggregation

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

Compute
Mgmt
Local PM
Remote PM

Compute
Mgmt
Local PM
Remote PM

Active Disaggregation

Compute

Compute

Mgmt
Remote PM

Mgmt
Remote PM

Passive Disaggregation

Compute
Mgmt

Compute
Mgmt

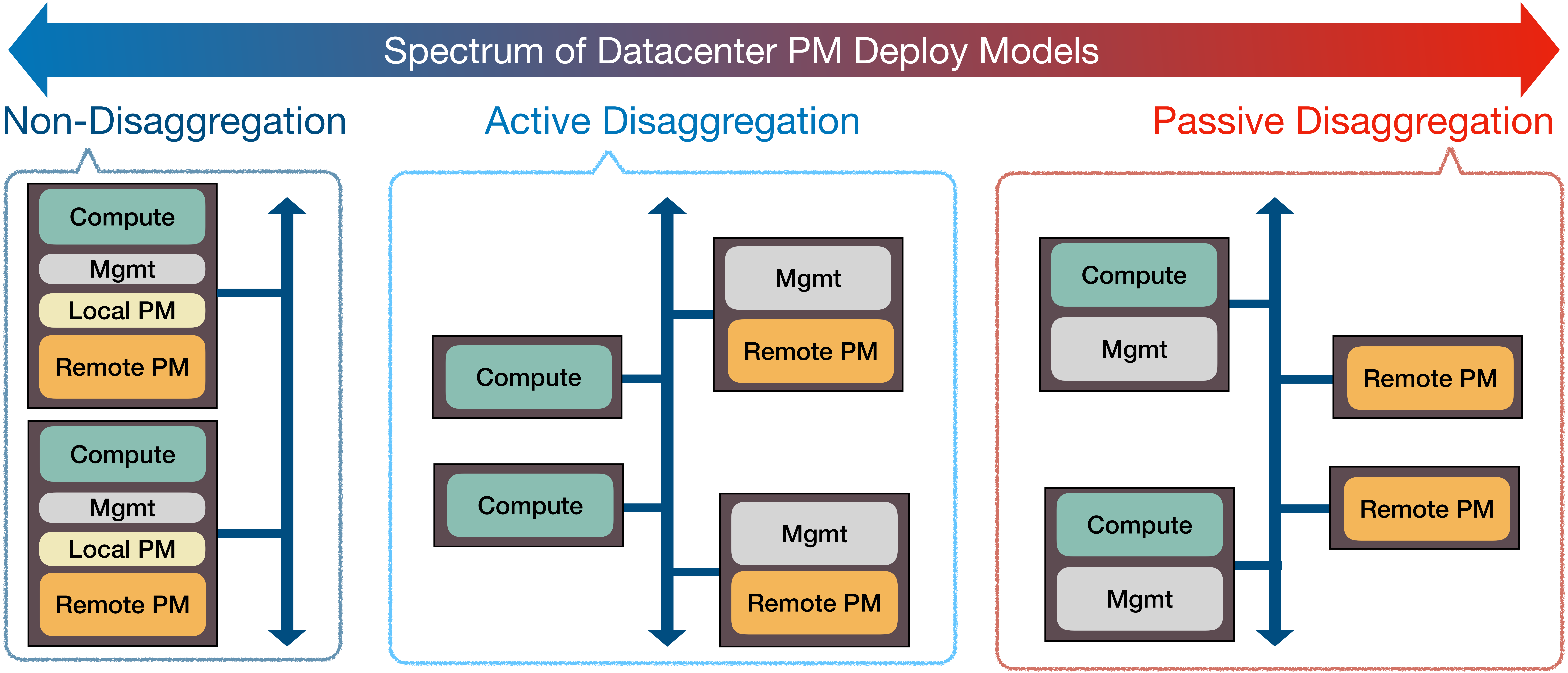
Remote PM

Remote PM

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

Compute
Mgmt
Local PM
Remote PM

Compute
Mgmt
Local PM
Remote PM

Active Disaggregation

Compute
Compute

Mgmt
Remote PM

Mgmt
Remote PM

Passive Disaggregation

Compute
Mgmt

Compute
Mgmt

Remote PM
Remote PM

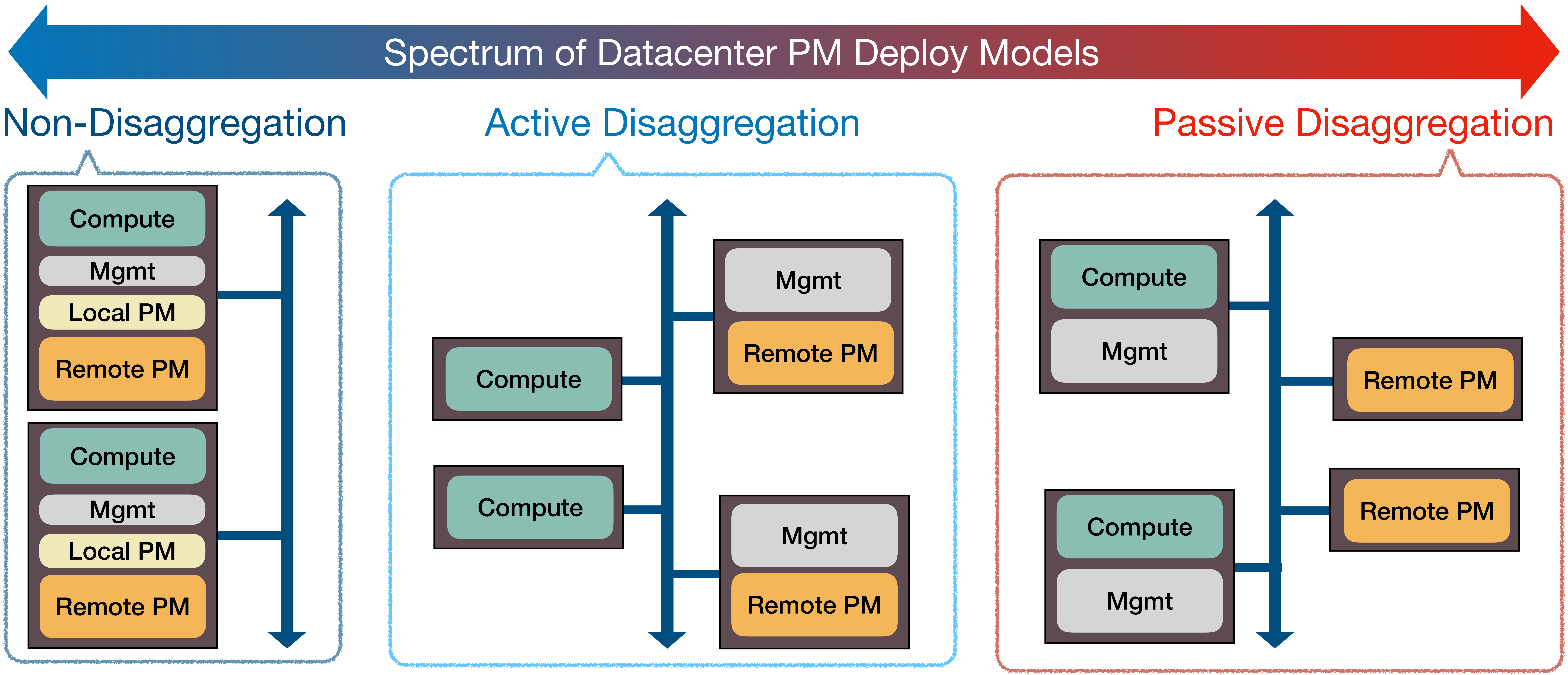
- low resource util
- inflexible

+ flexible
+ good performance

+ flexible
+ smaller failure domain

Traditional Storage Systems

Unexplored Area!



Non-Disaggregation:

- low resource util
- inflexible

Active Disaggregation:

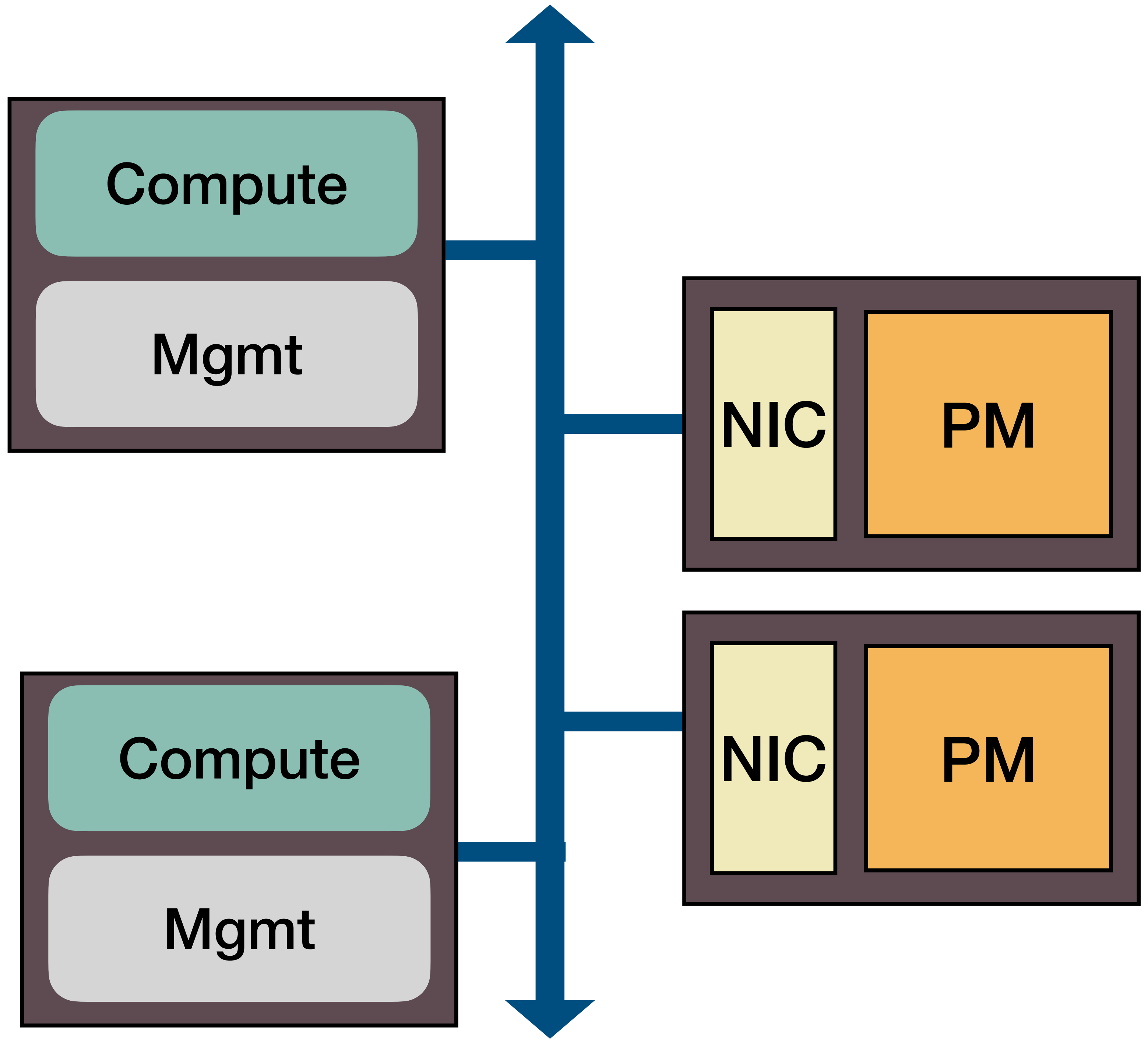
+ flexible
+ good performance

- high CAPEX and OPEX
- storage node: scalability bottleneck

Passive Disaggregation:

+ flexible
+ smaller failure domain
+ low cost
* performance?

Passive Disaggregated PM (pDPM)



pDPM

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

Why pDPM?

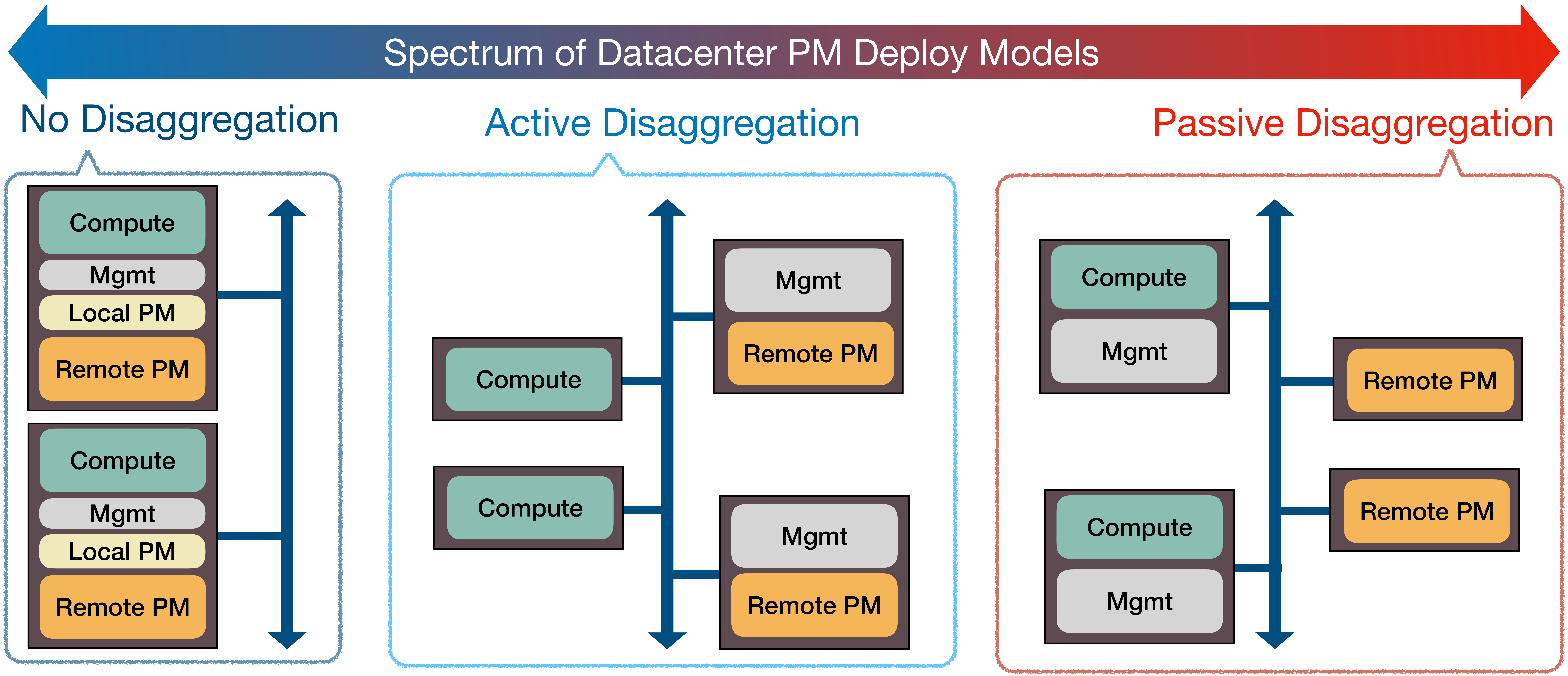
- Low CapEx and OpEx
- Easy to add, remove, and change
- No scalability bottleneck at storage nodes
- Research value in exploring new design area

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

Compute

Mgmt

Local PM

Remote PM

Compute

Mgmt

Local PM

Remote PM

Active Disaggregation

Compute

Compute

Mgmt

Remote PM

Mgmt

Remote PM

Passive Disaggregation

Compute

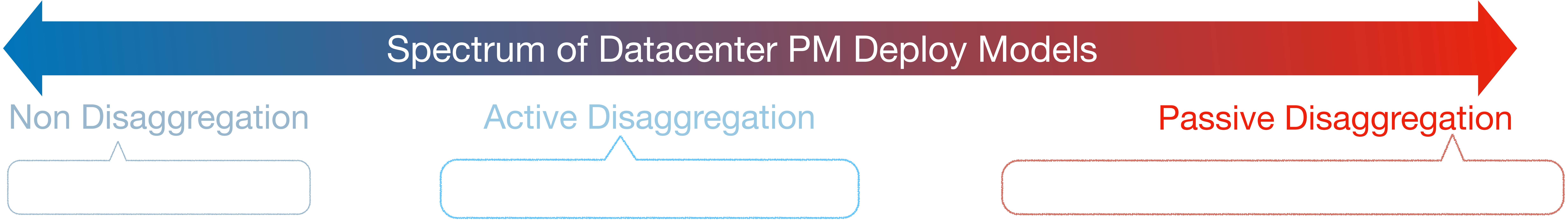
Mgmt

Compute

Mgmt

Remote PM

Remote PM

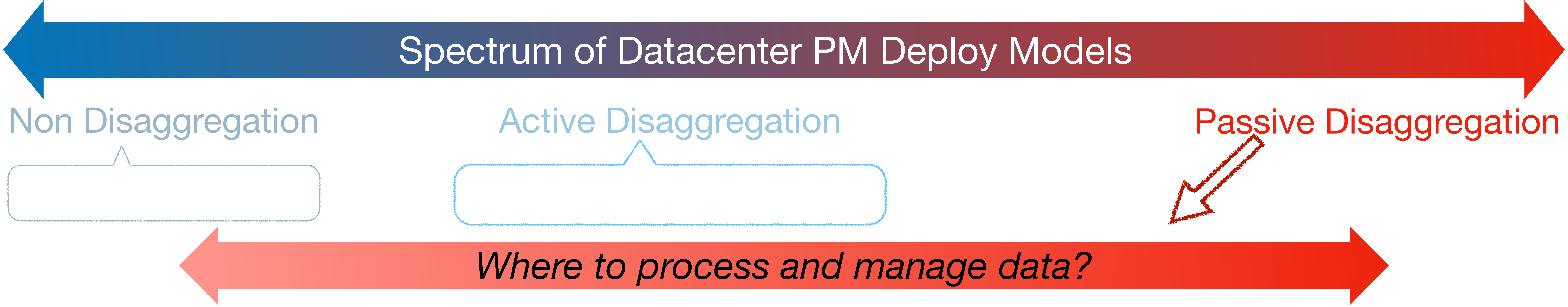


Spectrum of Datacenter PM Deploy Models

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

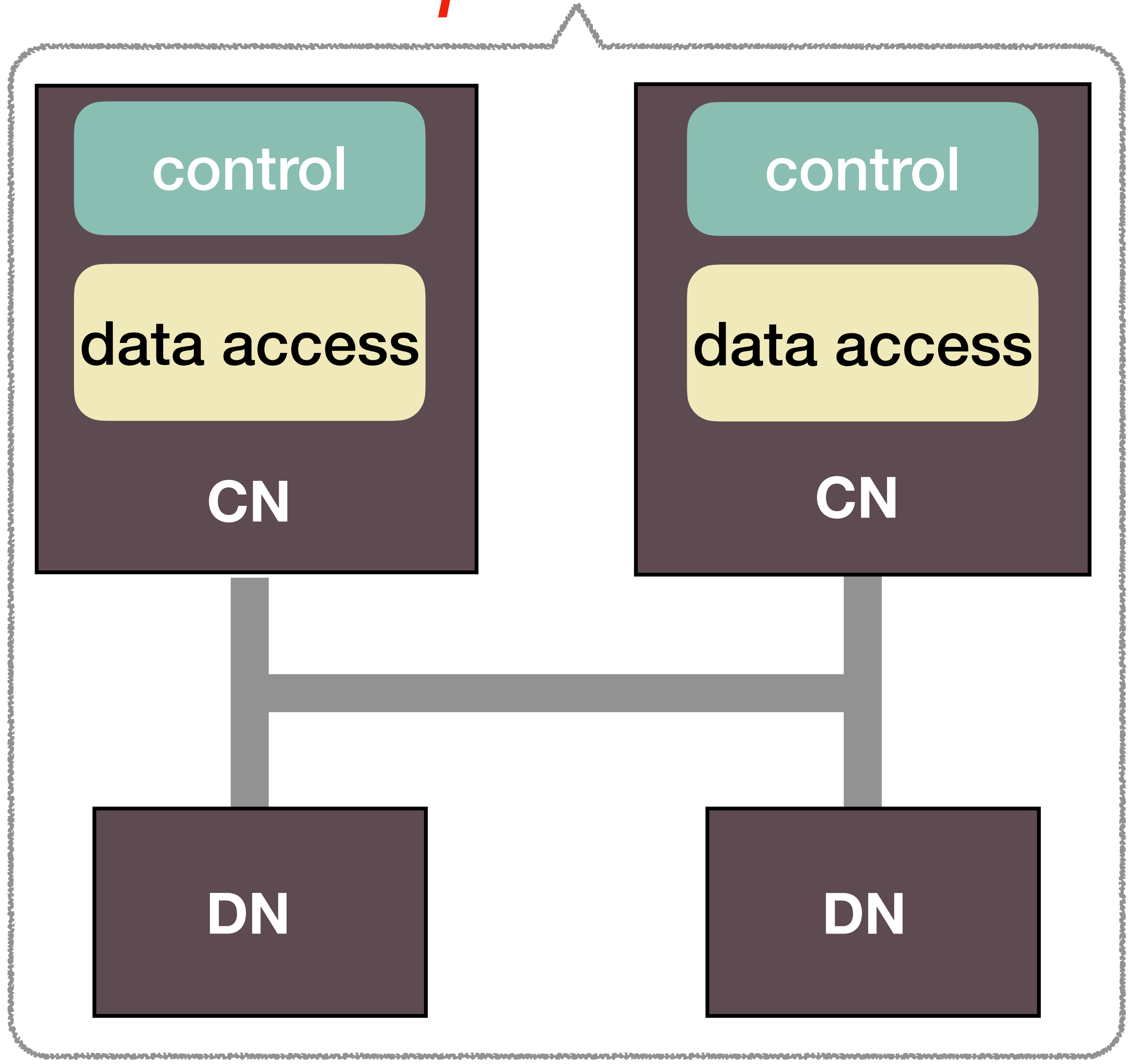
Non Disaggregation

Active Disaggregation

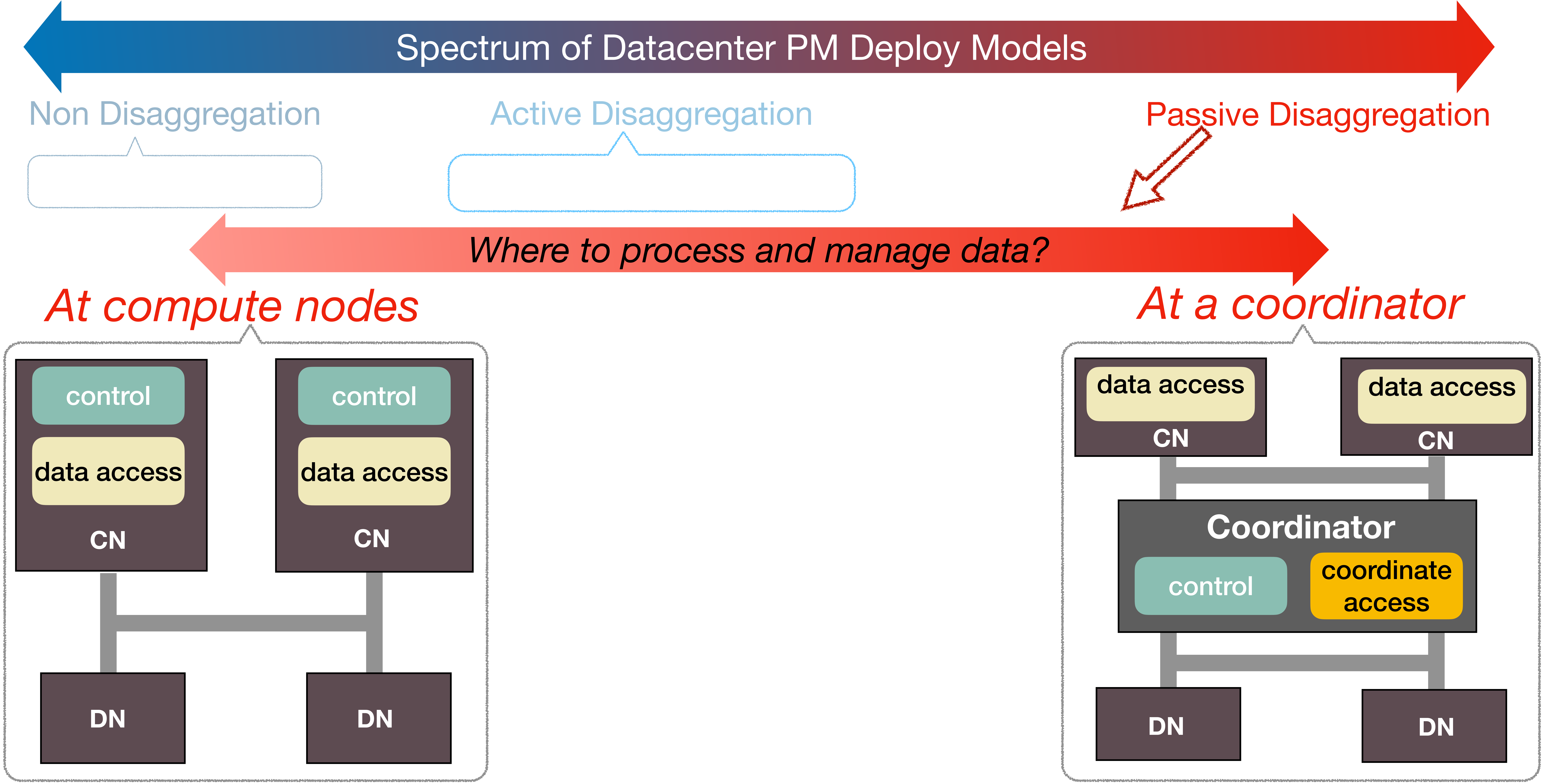
Passive Disaggregation

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

control

data access

CN

control

data access

CN

DN

DN

At a coordinator

data access

CN

data access

CN

Coordinator

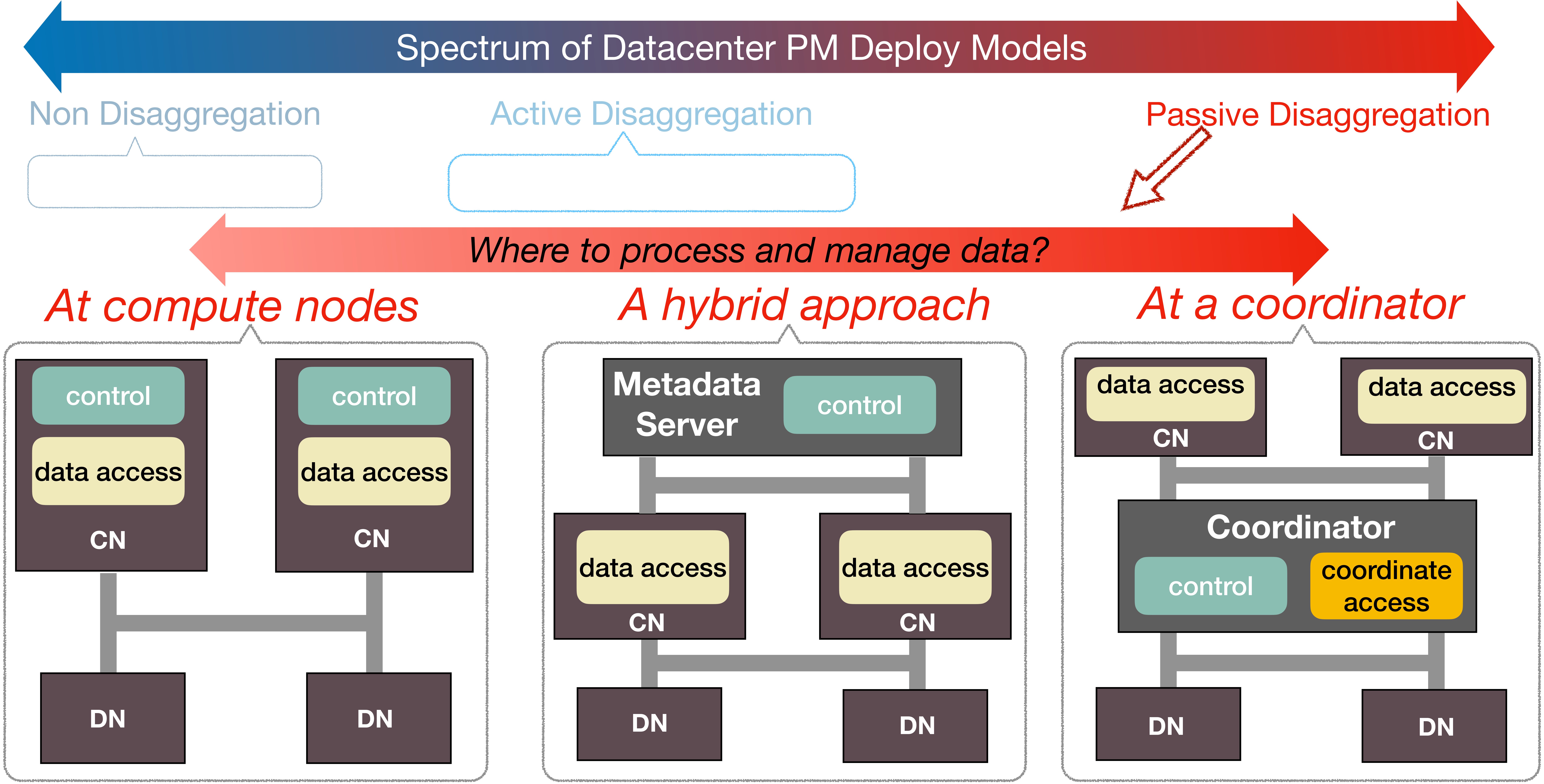
control

coordinate access

DN

DN

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

control

data access

CN

control

data access

CN

DN

DN

A hybrid approach

Metadata Server

control

data access

CN

data access

CN

DN

DN

At a coordinator

data access

CN

data access

CN

Coordinator

control

coordinate access

DN

DN

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

Clover

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

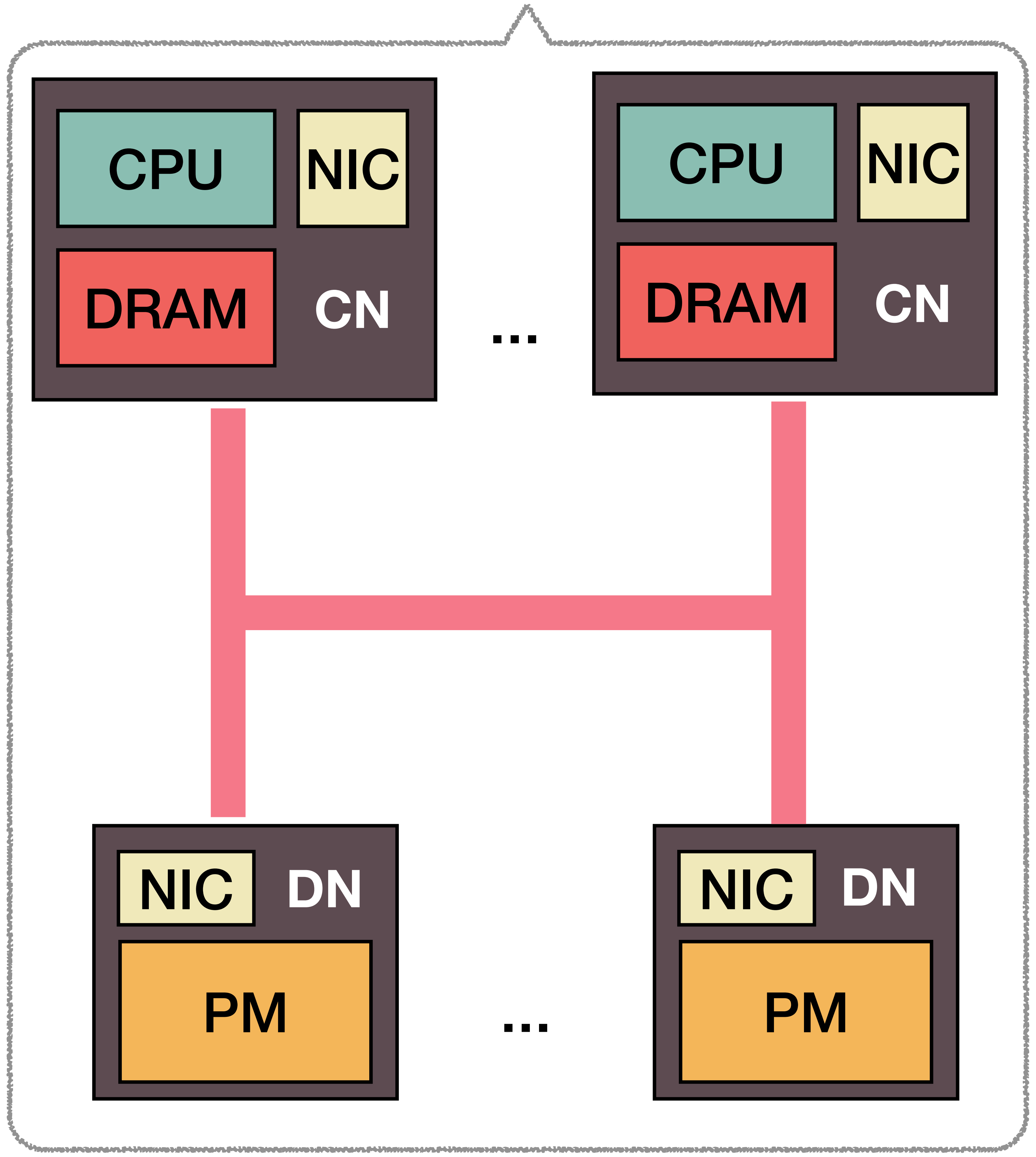
control

coordinate access

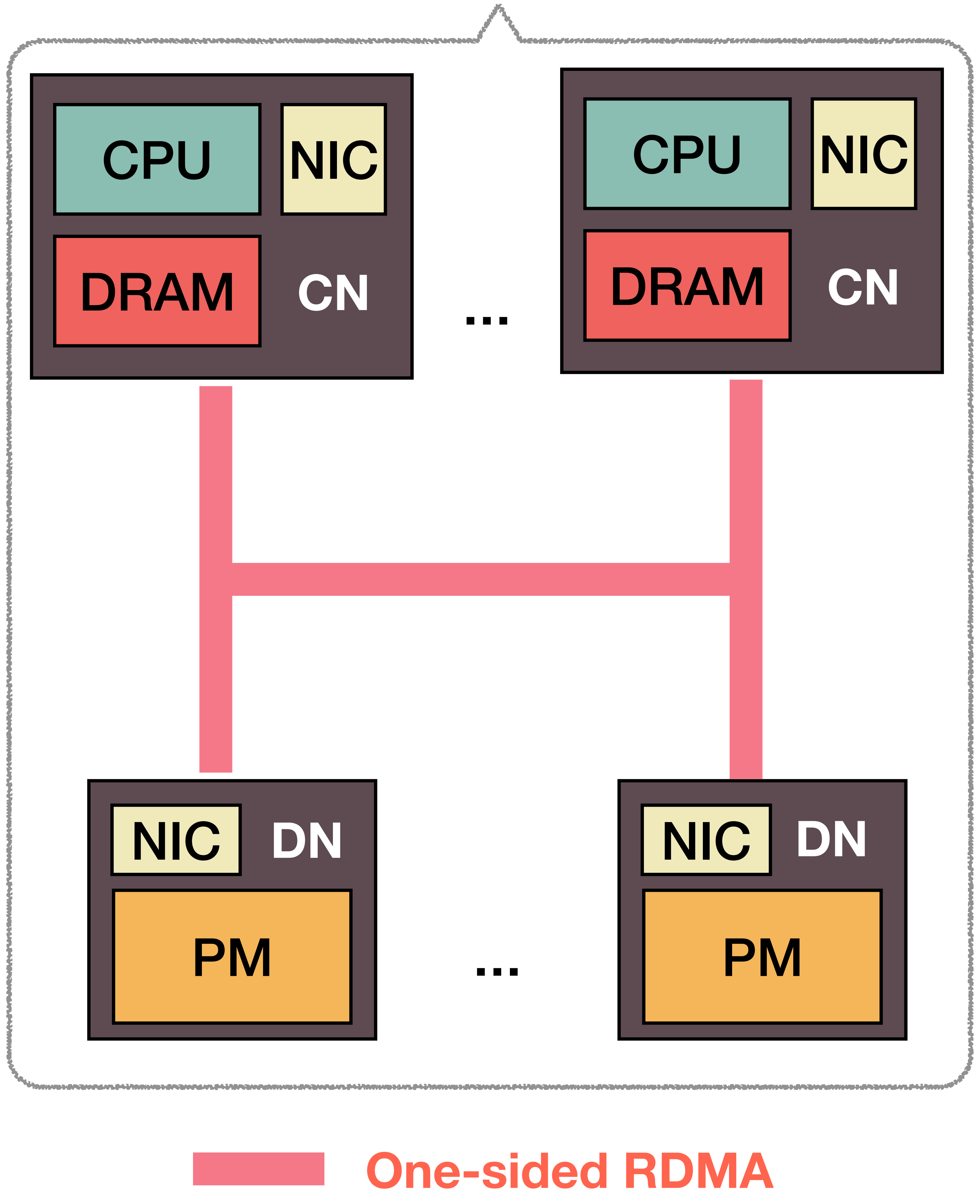
DN

DN

pDPM-Direct: Directly Access and Manage DNs from CNs



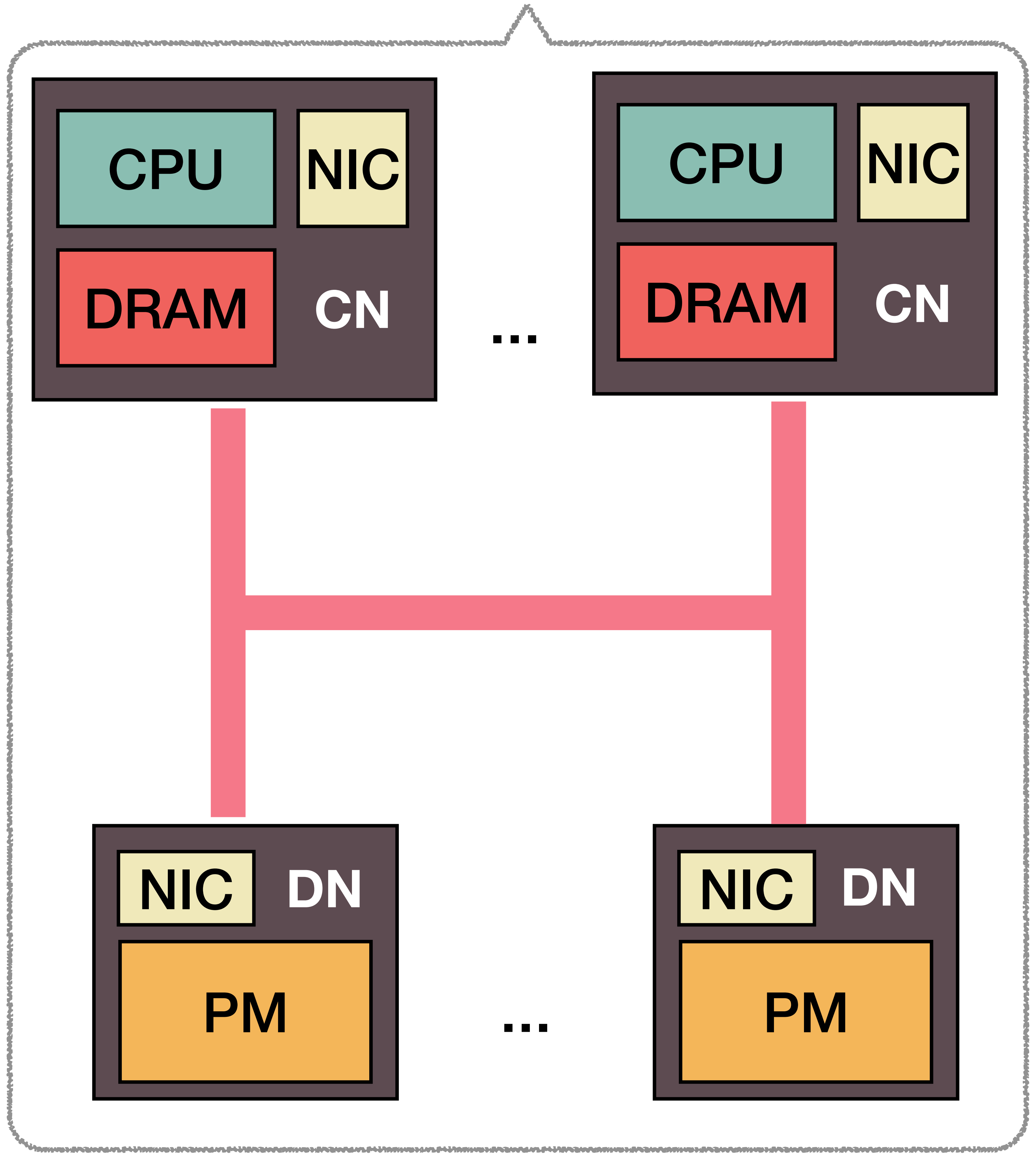
pDPM-Direct: Directly Access and Manage DNs from CNs



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

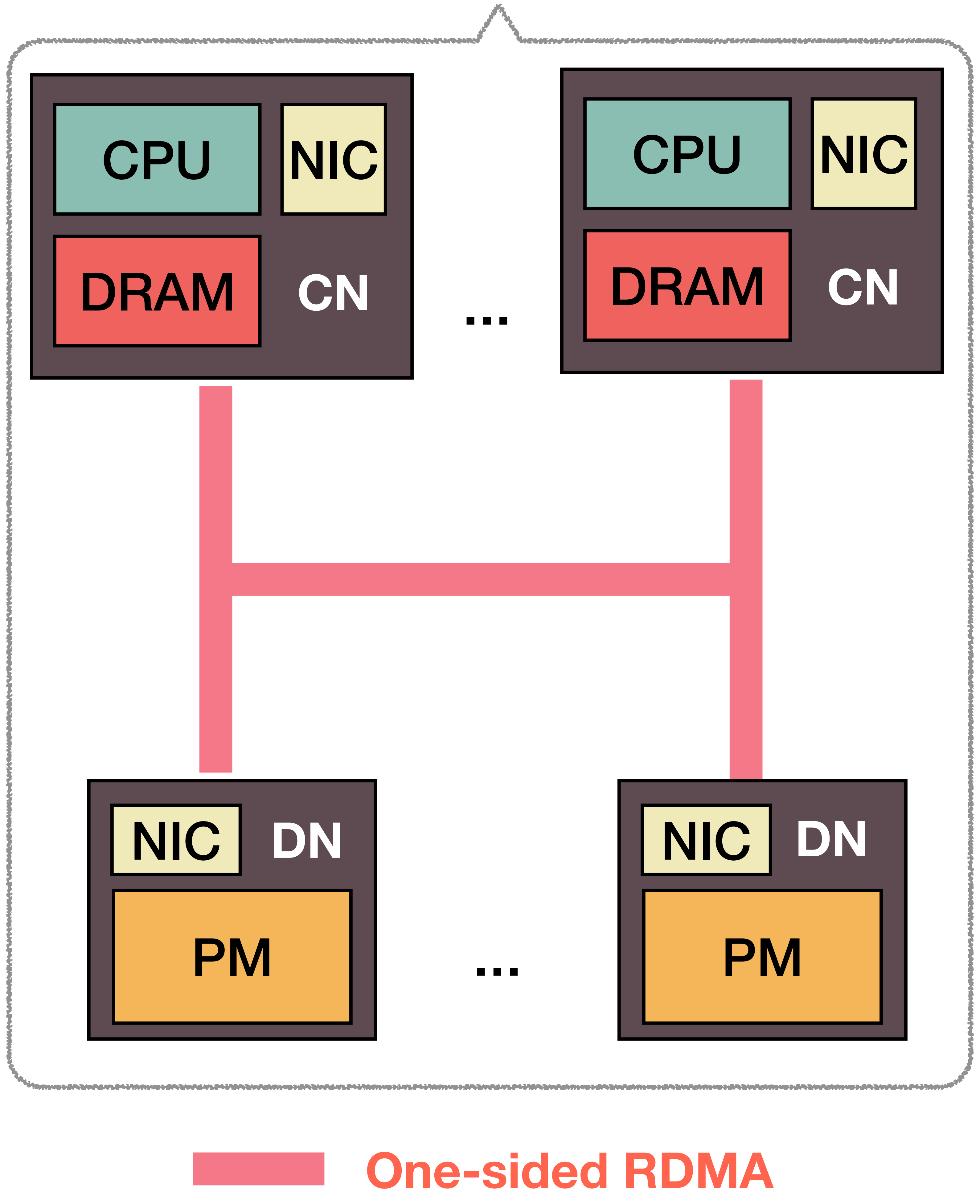


▬ One-sided RDMA

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



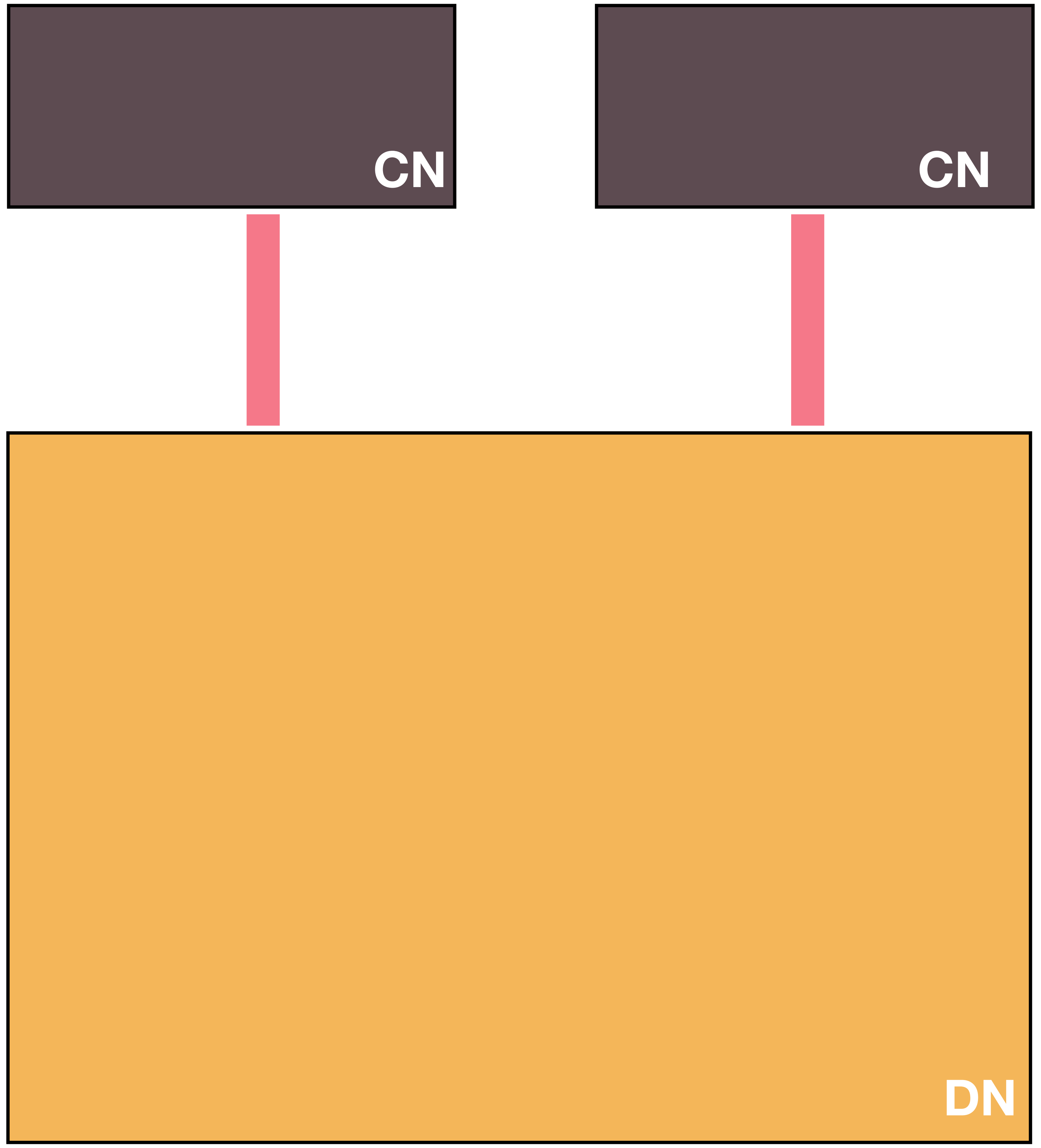
Overall Architecture

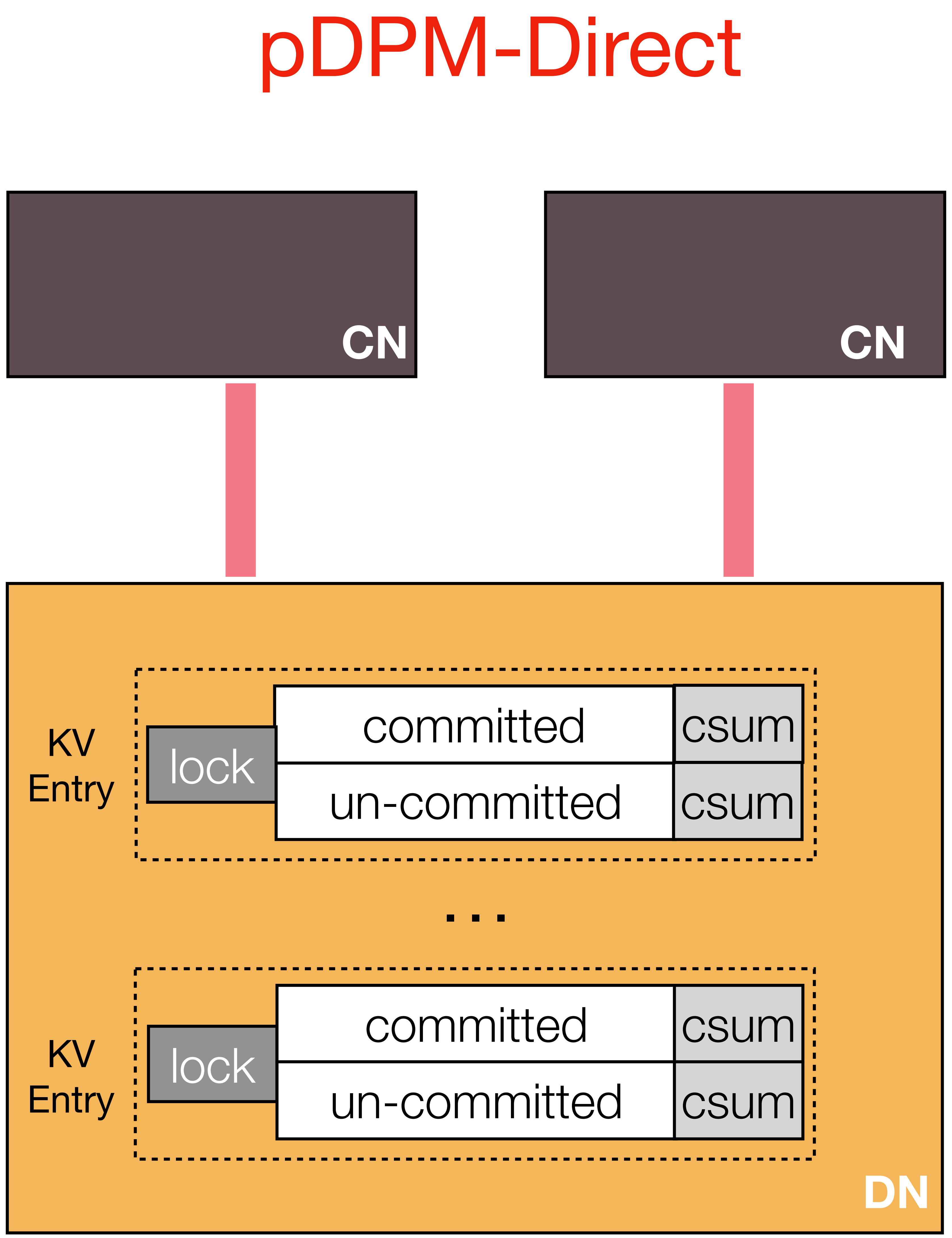
- CNs access and manage DNs directly via one-sided RDMA
- Both data and control planes run within CNs

Challenges

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

pDPM-Direct

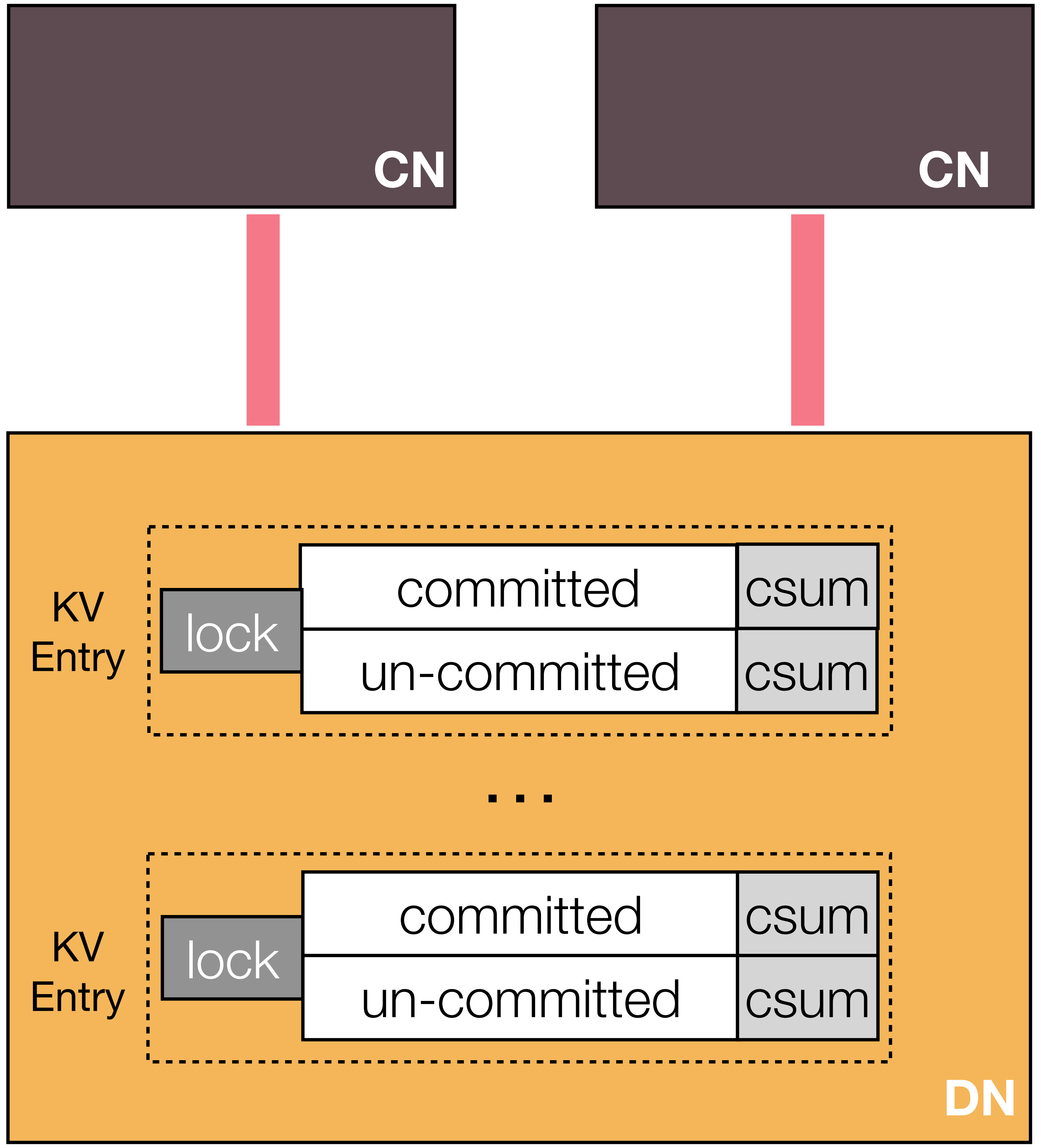




Our solution

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

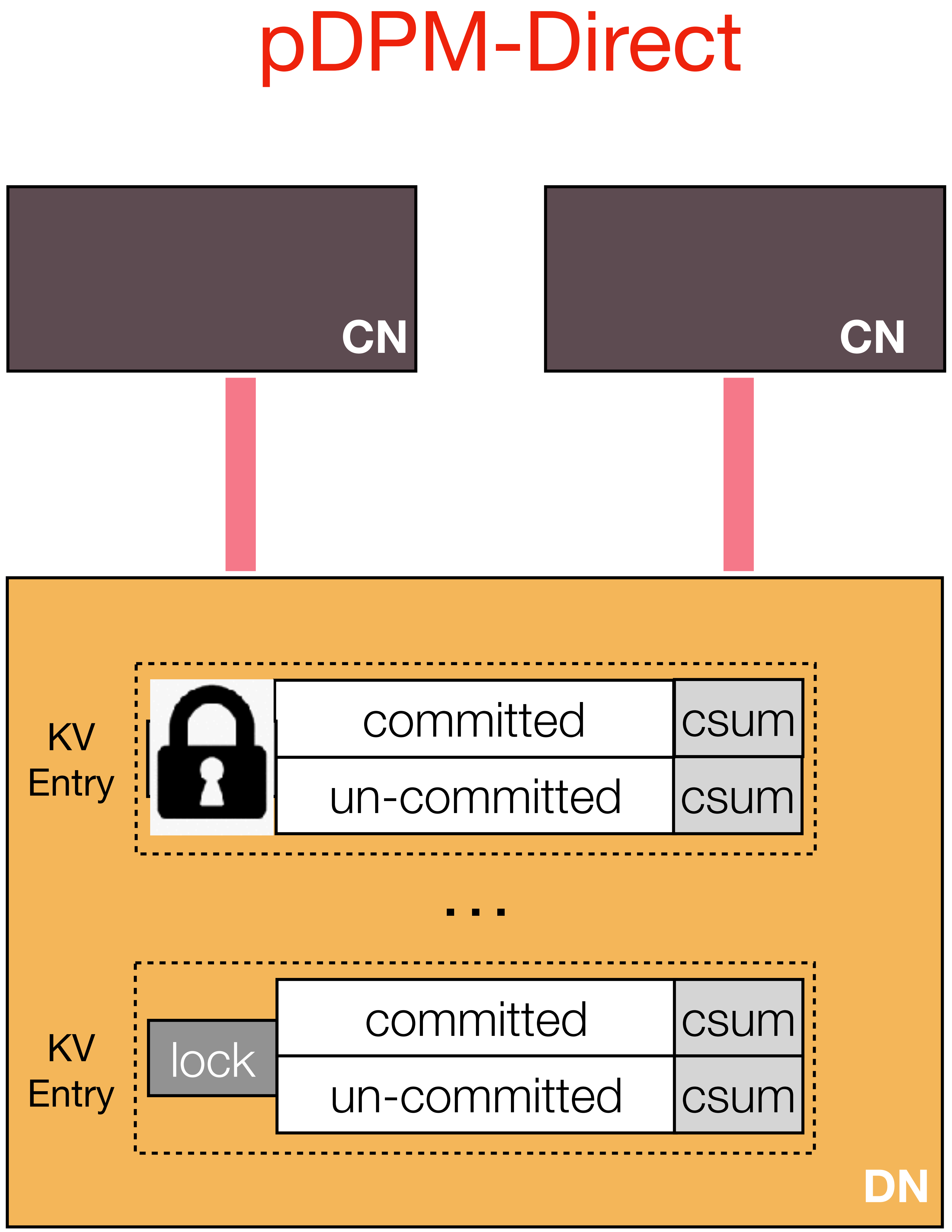
pDPM-Direct



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



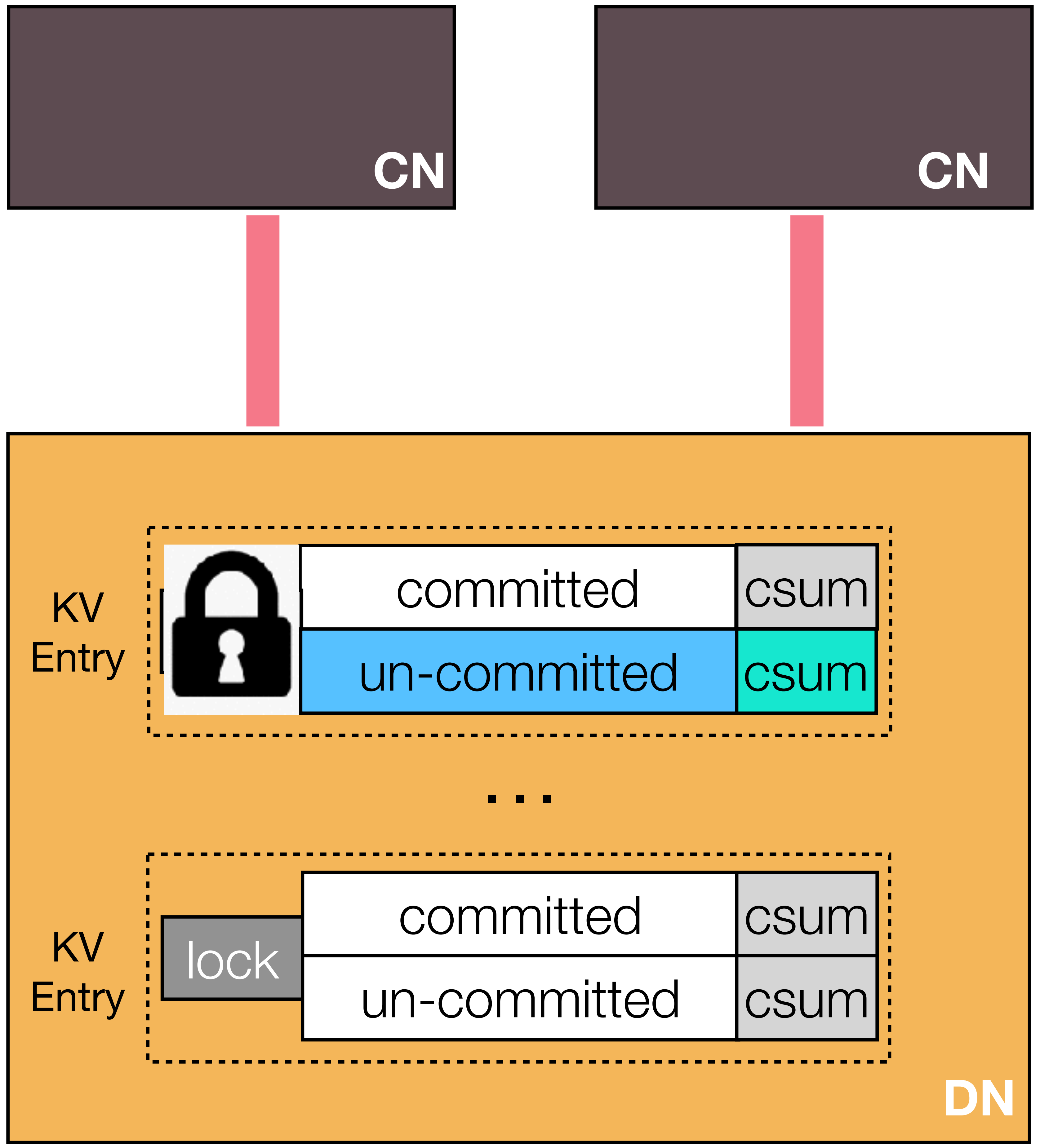
Our solution

- 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

pDPM-Direct



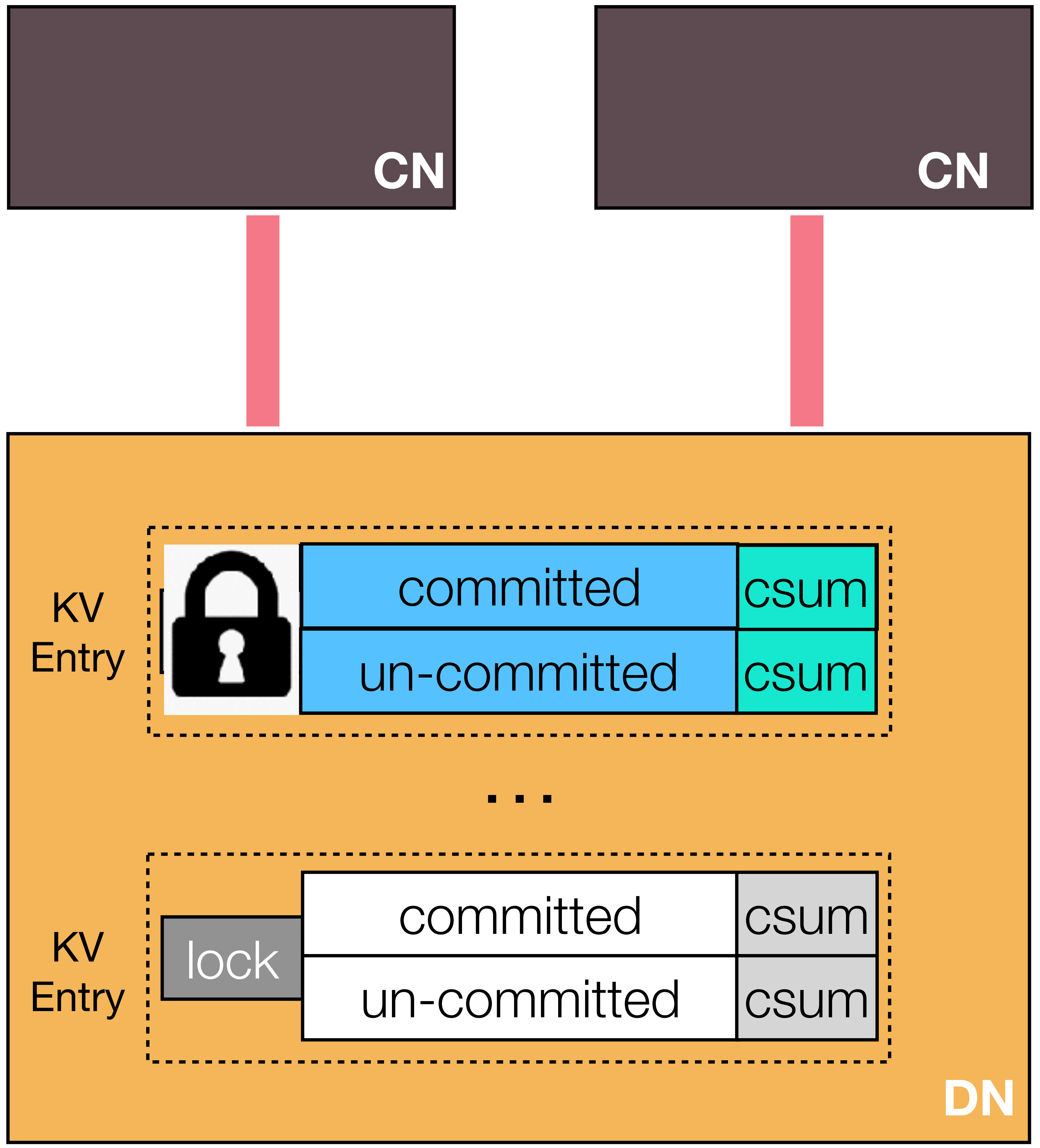
Our solution

- 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)

pDPM-Direct



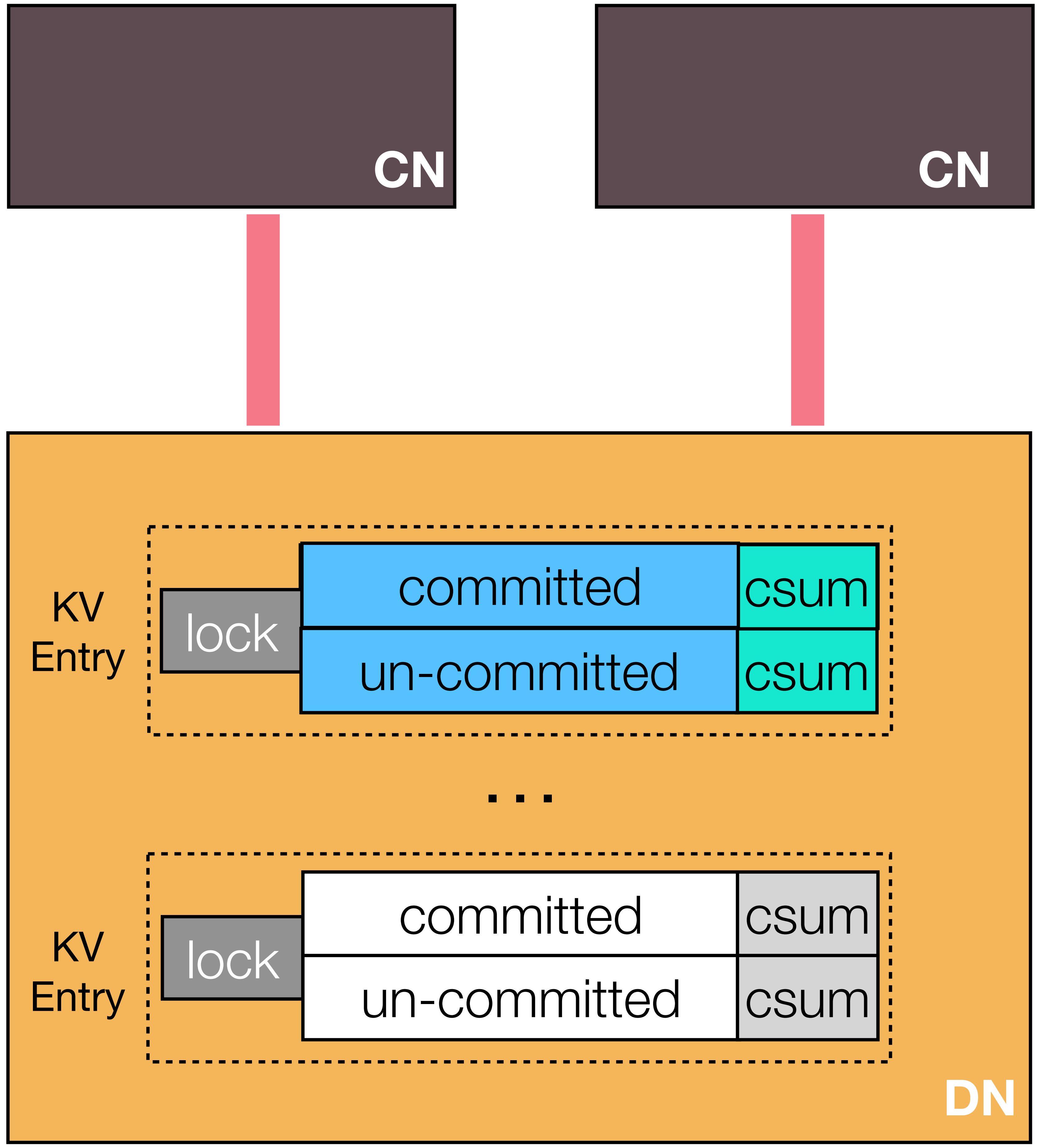
Our solution

- 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

pDPM-Direct



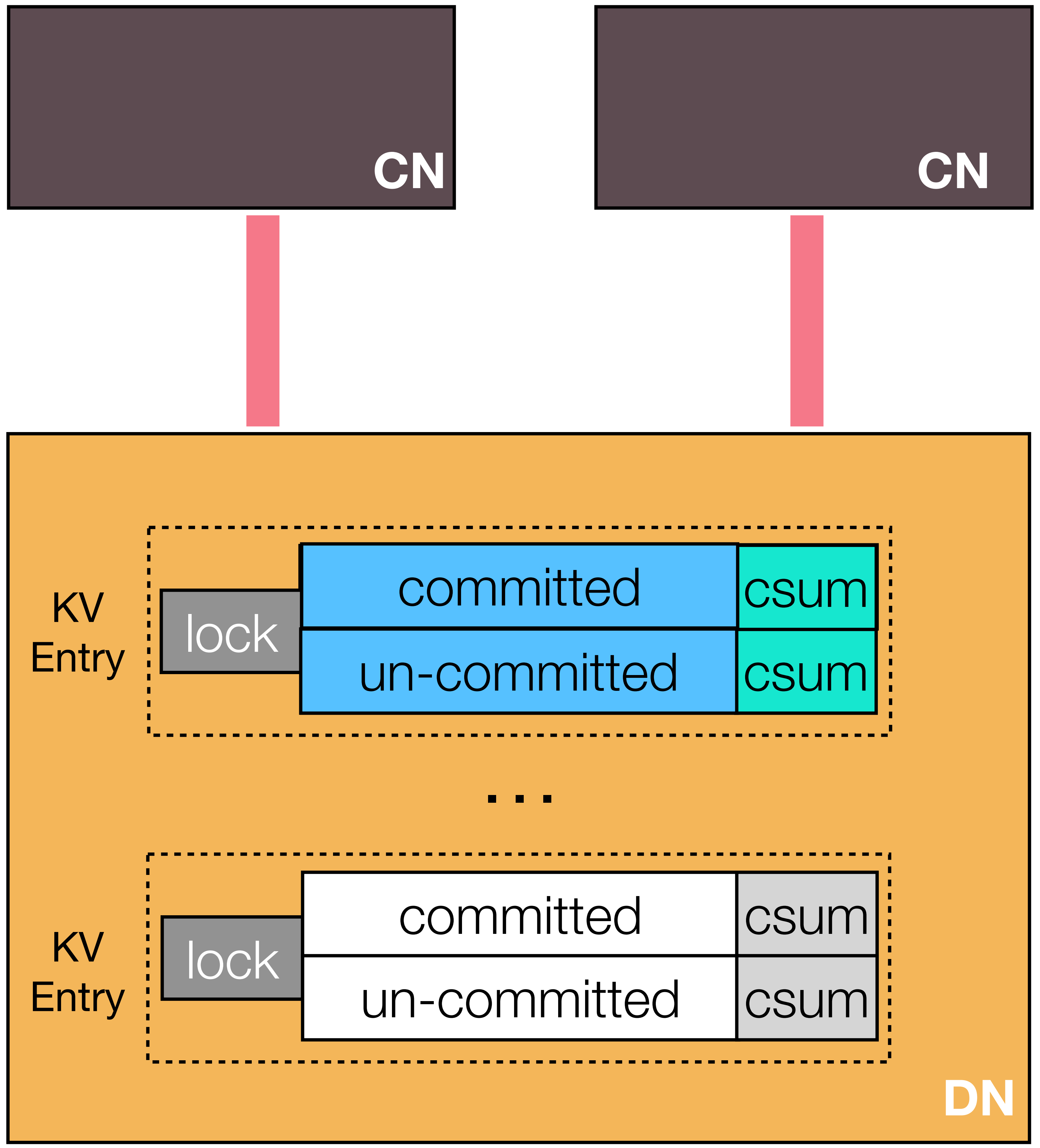
Our solution

- 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`)

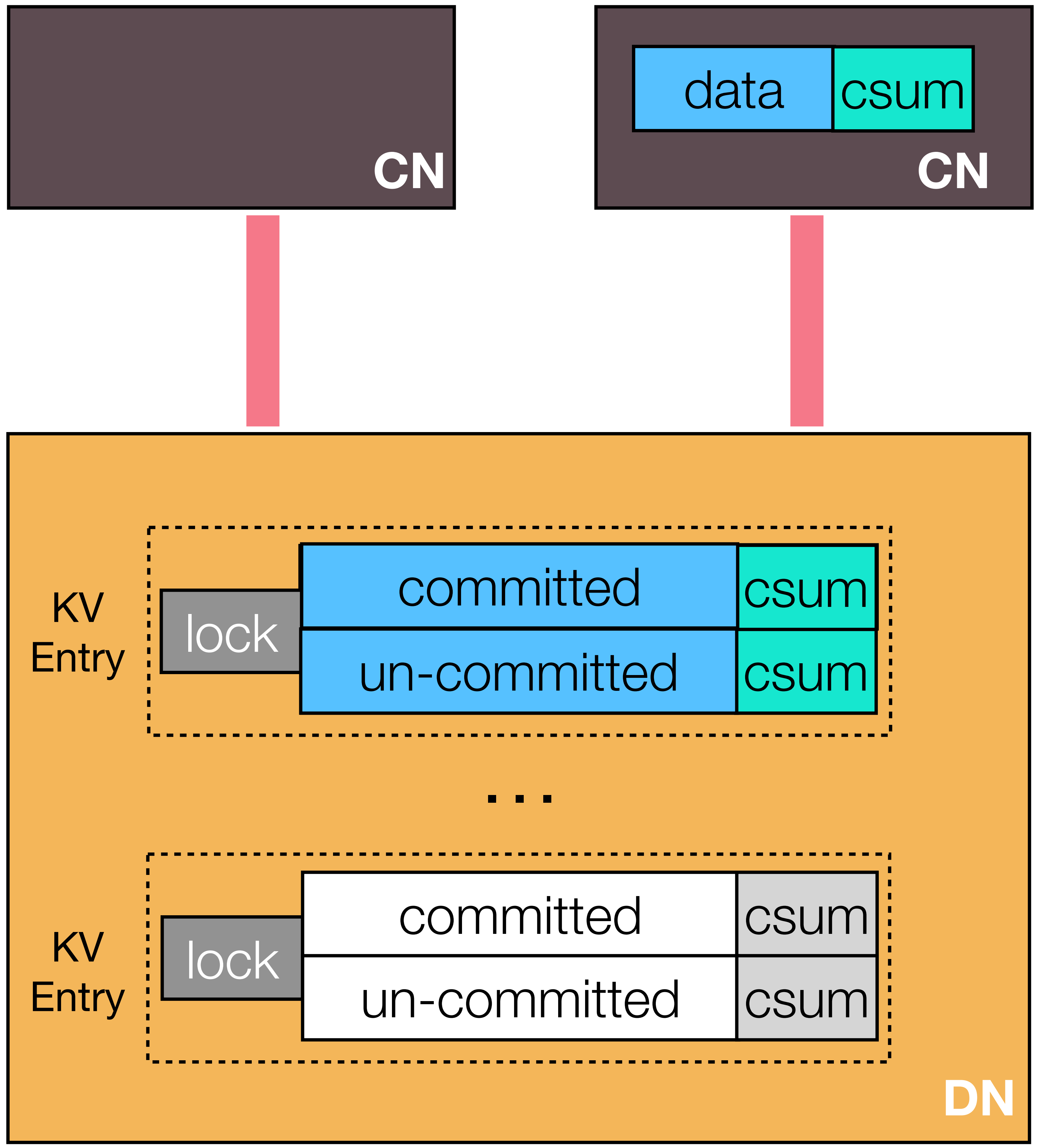
Write Flow

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

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`)

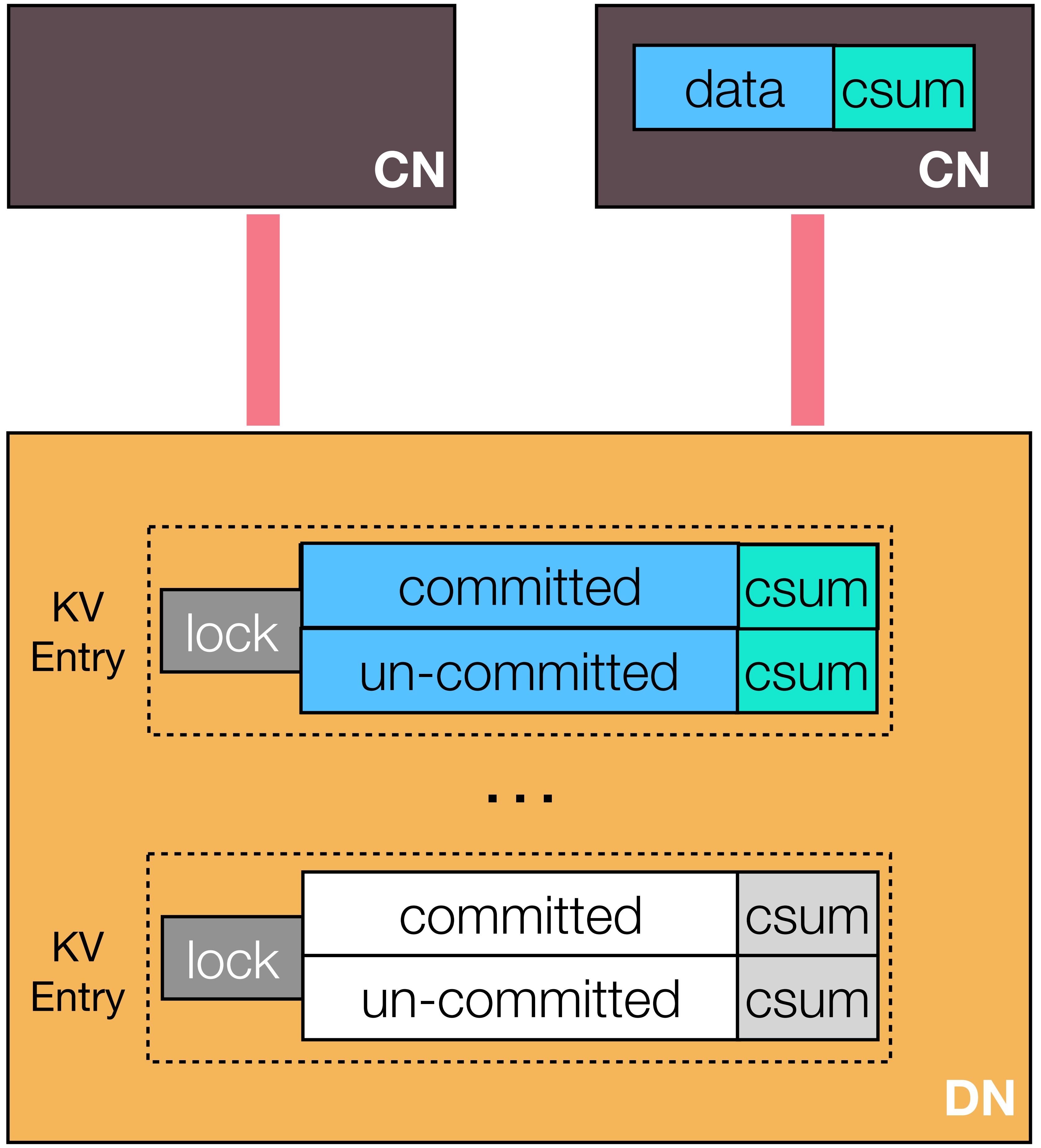
Write Flow

- 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

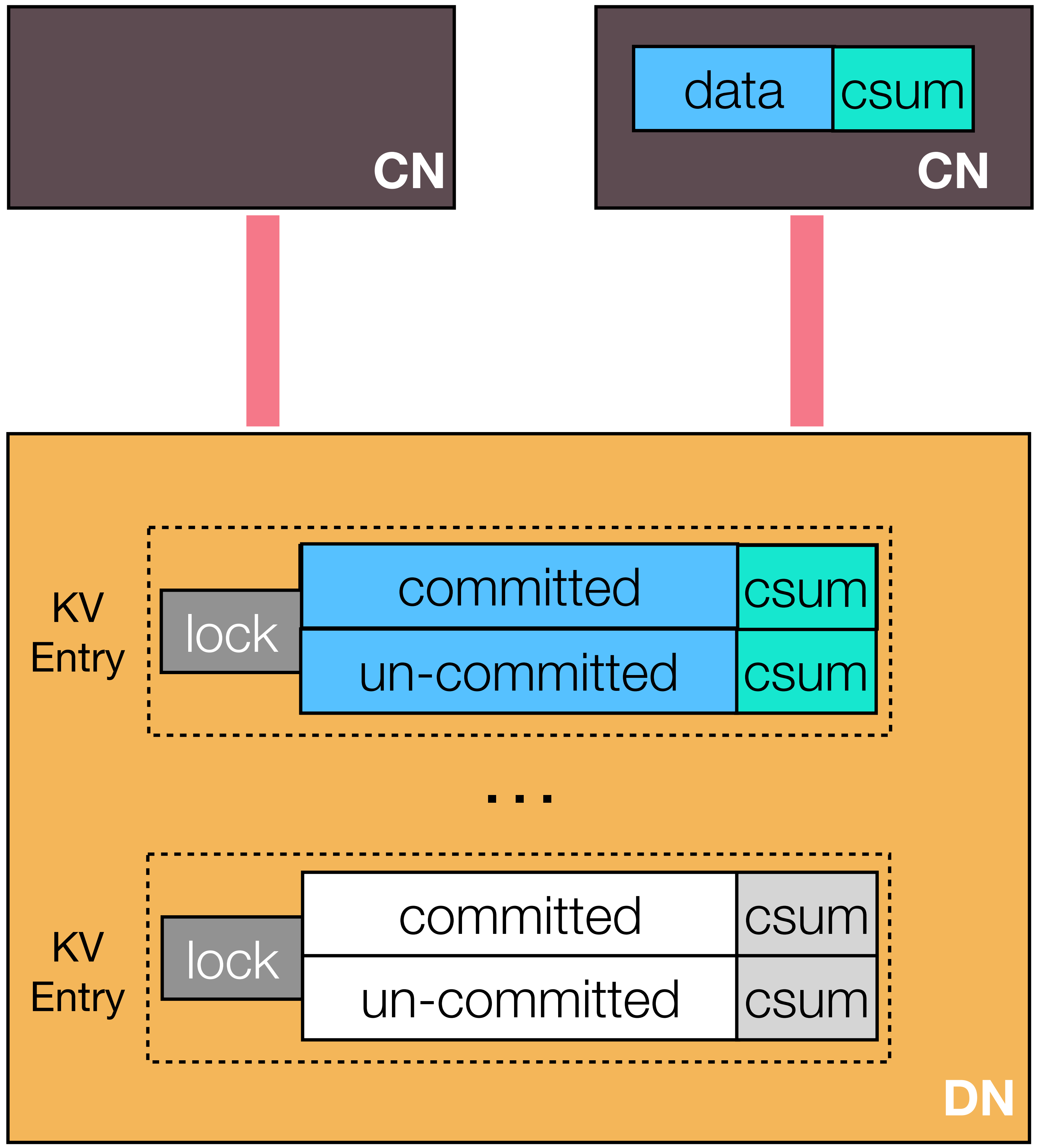
- Acquire lock
- Write new data+CRC into `uncommitted` space (redo-copy)
- Write new data+CRC into `committed` space
- Release lock

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 read with large data ☹
Poor scalability under concurrent accesses

Where to process and manage data?

pDPM-Direct

Clover

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

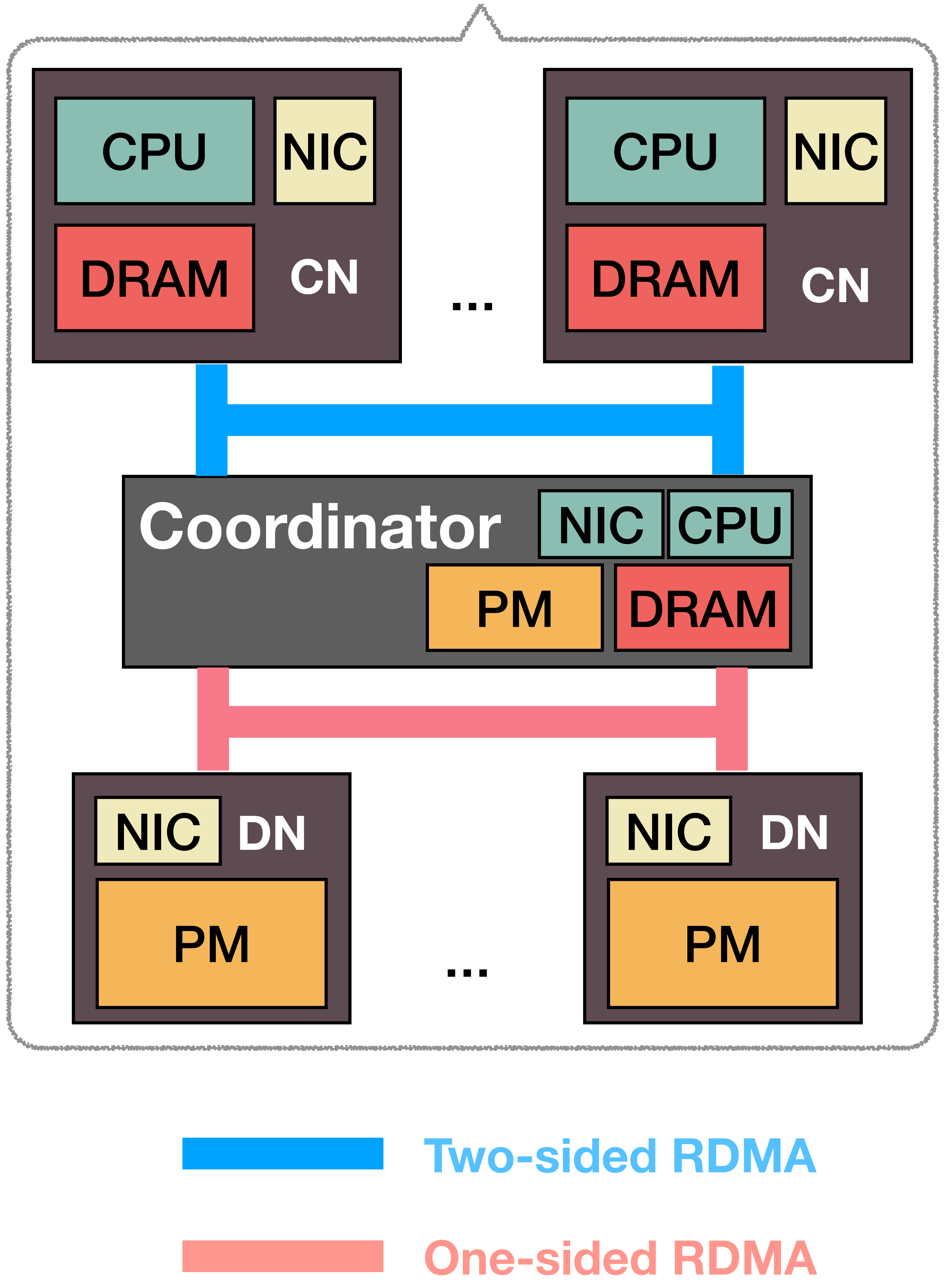
pDPM-Central: A Central Coordinator between CNs and DNs

The central coordinator

- Manages DN space
- Serializes CNs accesses with local locking

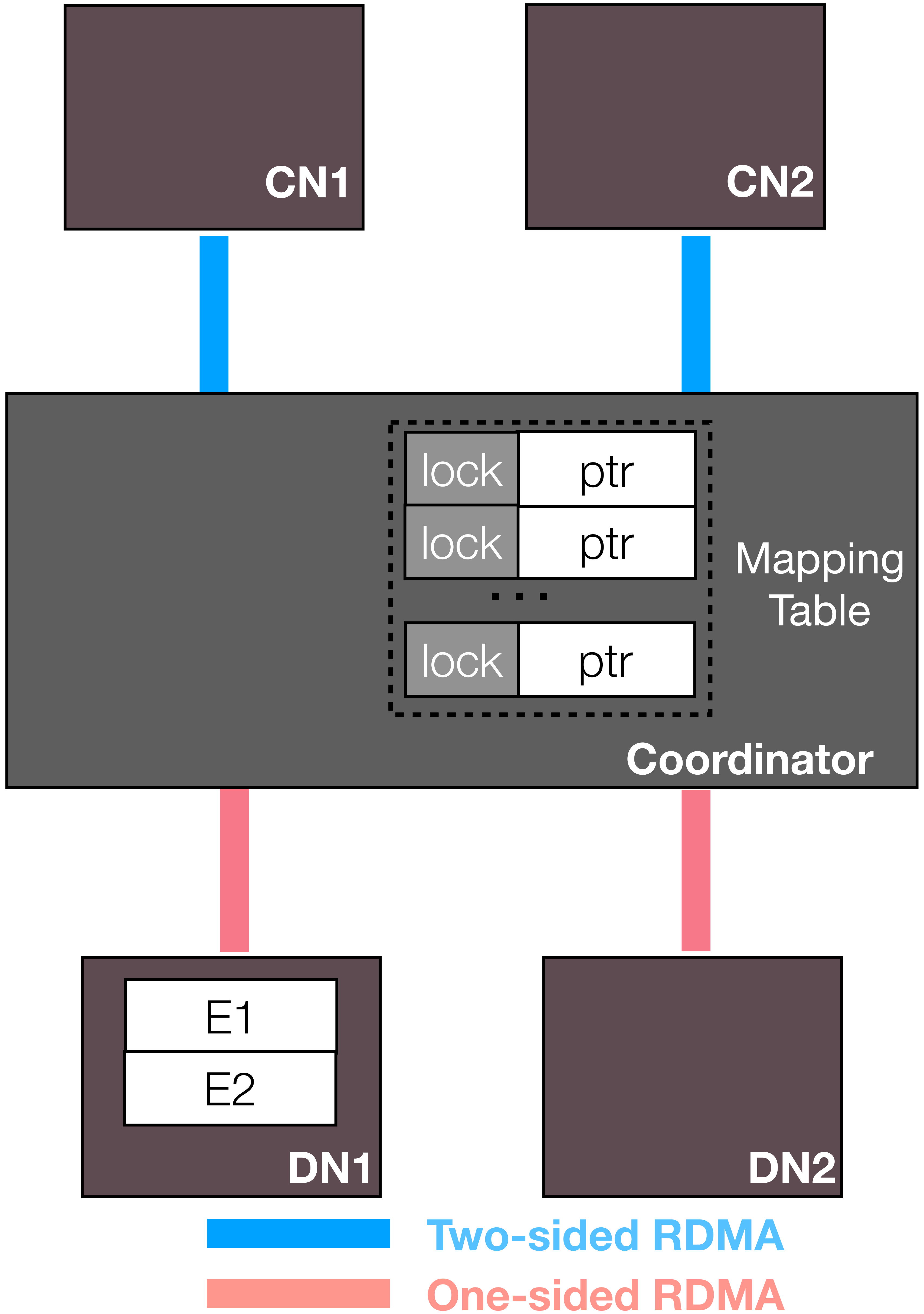
CNs communicate with the coordinator through two-sided RDMA

Coordinator accesses DNs through one-sided RDMA



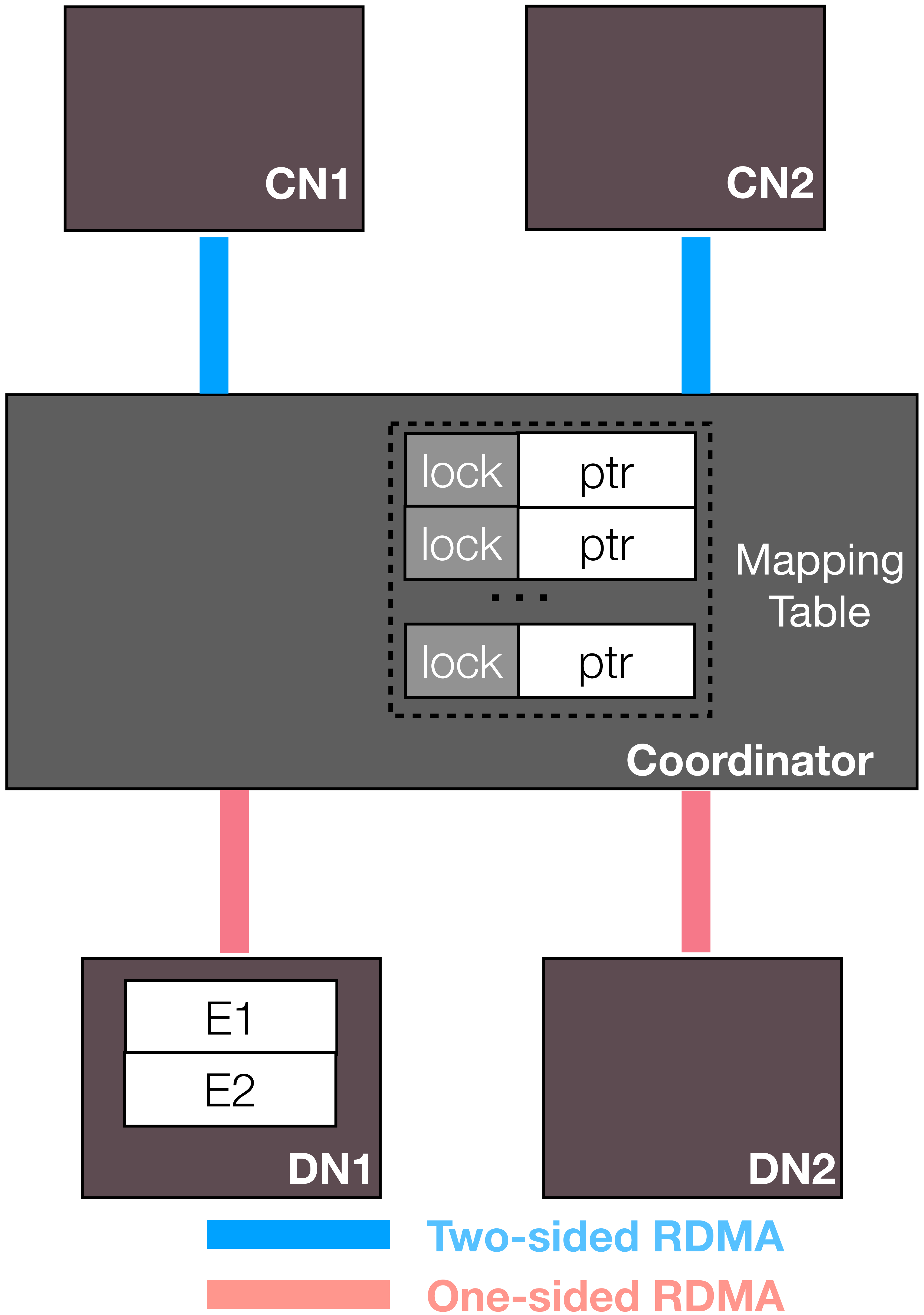
☺ **Easier to manage DNs and coordinate concurrent accesses**

pDPM-Central



pDPM-Central

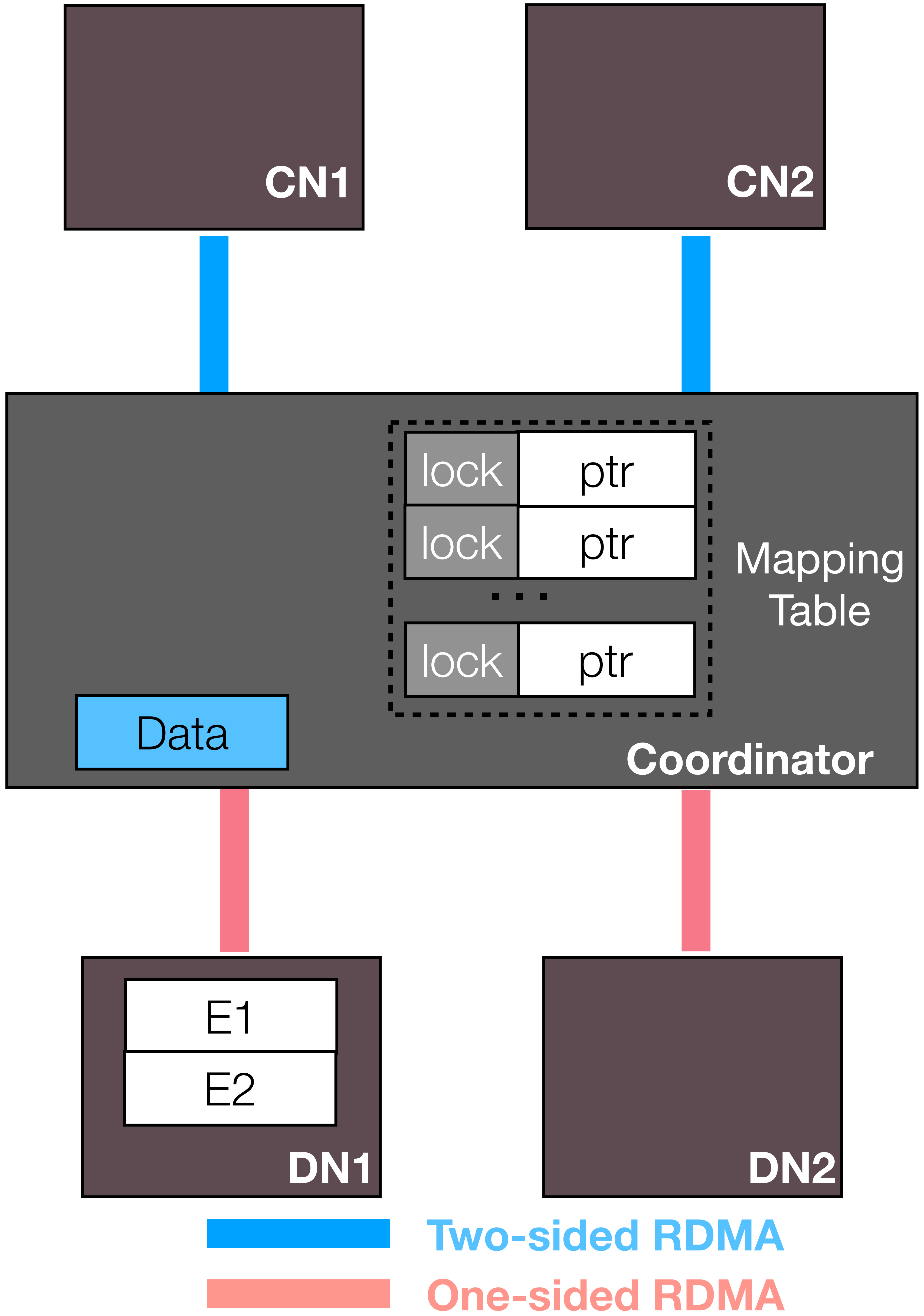
Write Flow



Write Flow

- CN sends RPC (with data) to Coordinator

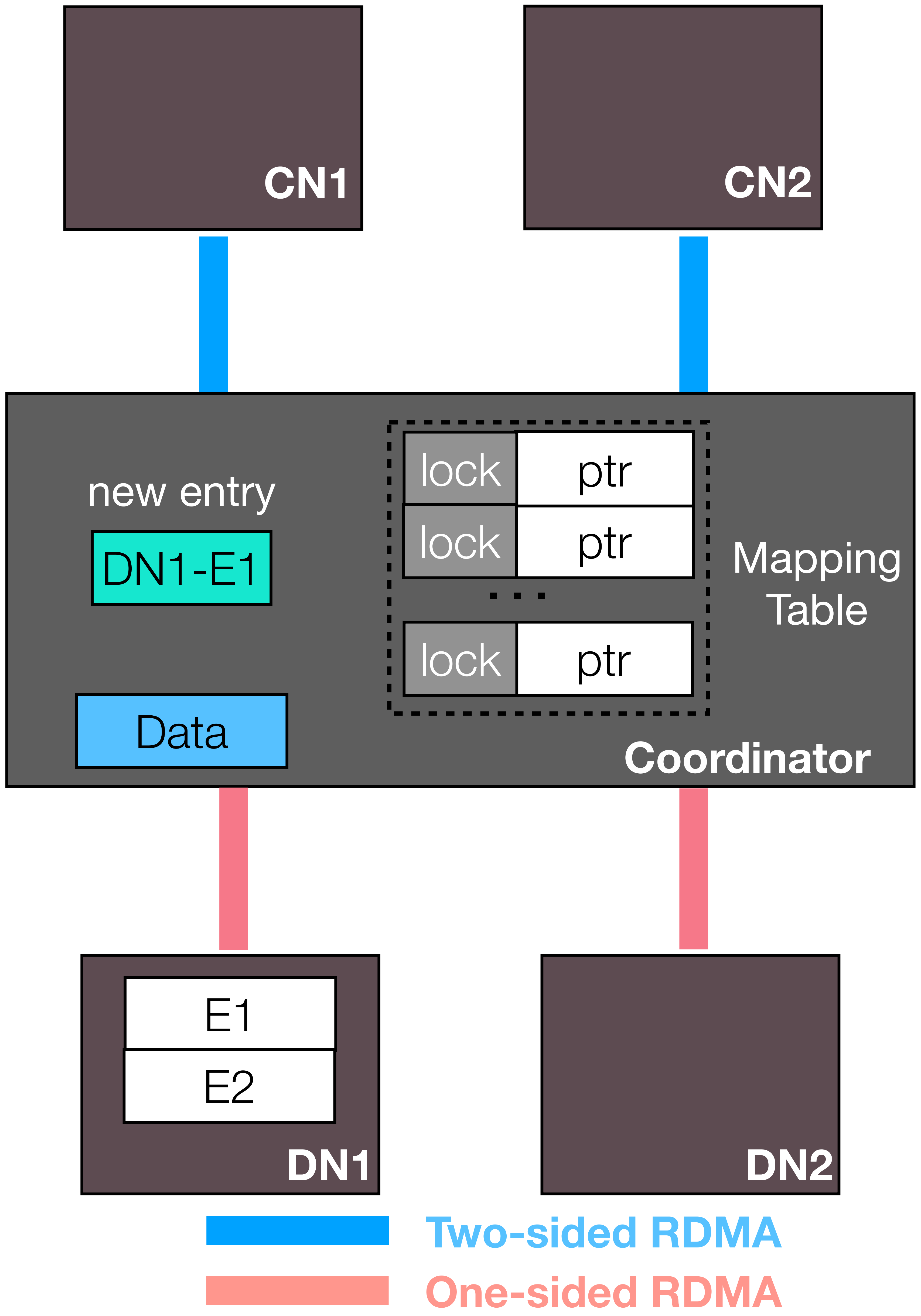
pDPM-Central



pDPM-Central

Write Flow

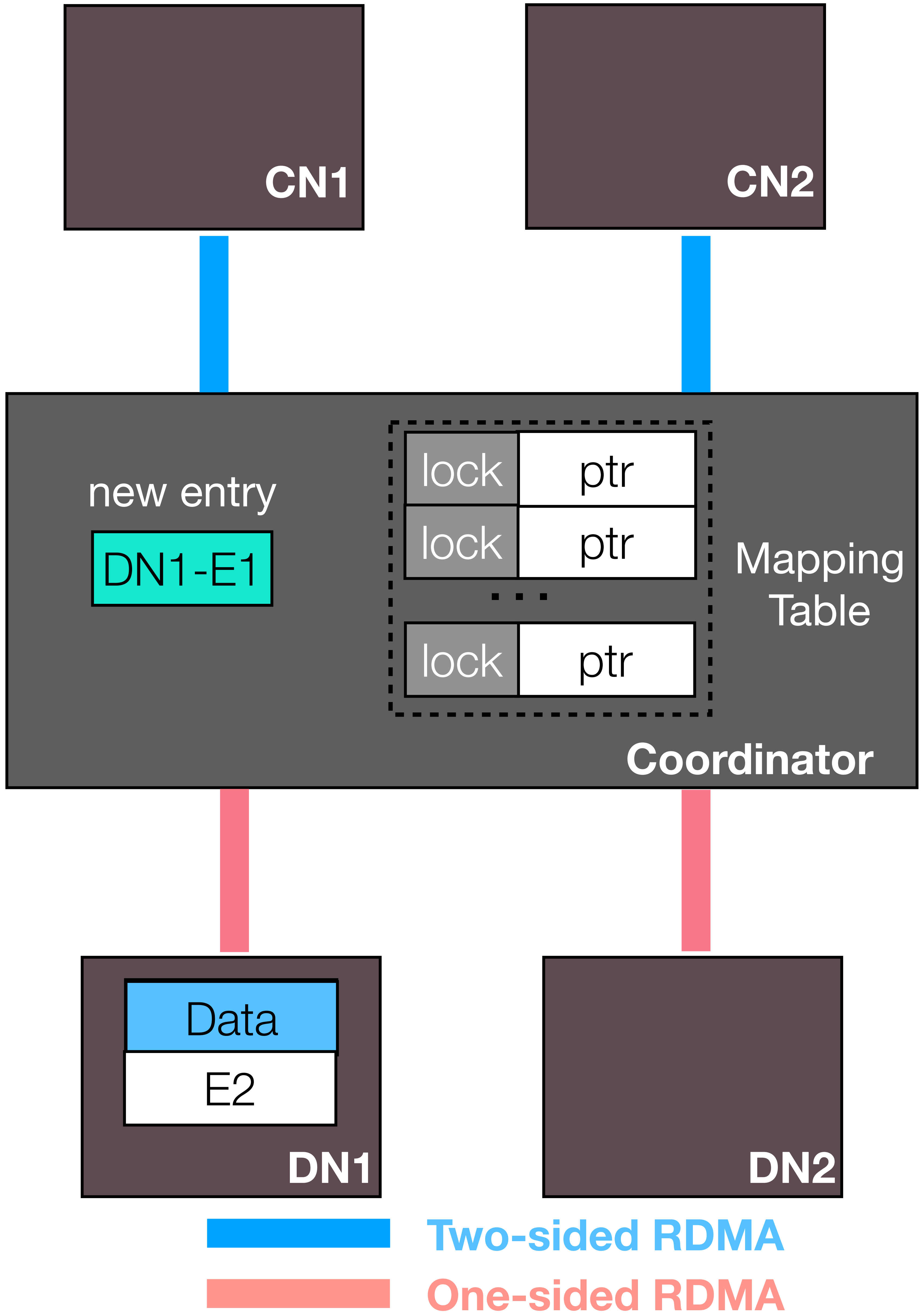
- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)



Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)

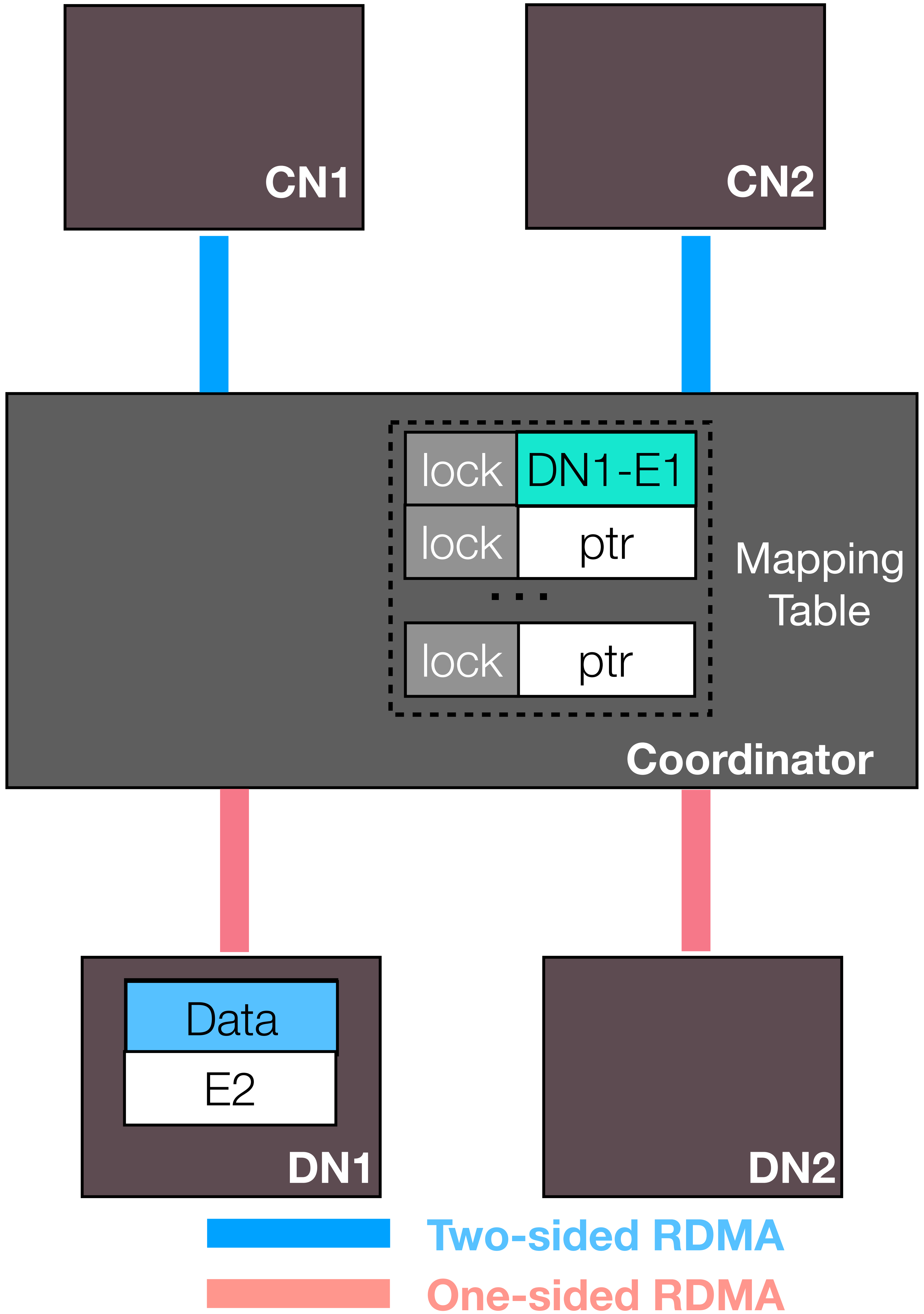
pDPM-Central



Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)
- Coordinator locks the entry in mapping table and update ptr

pDPM-Central

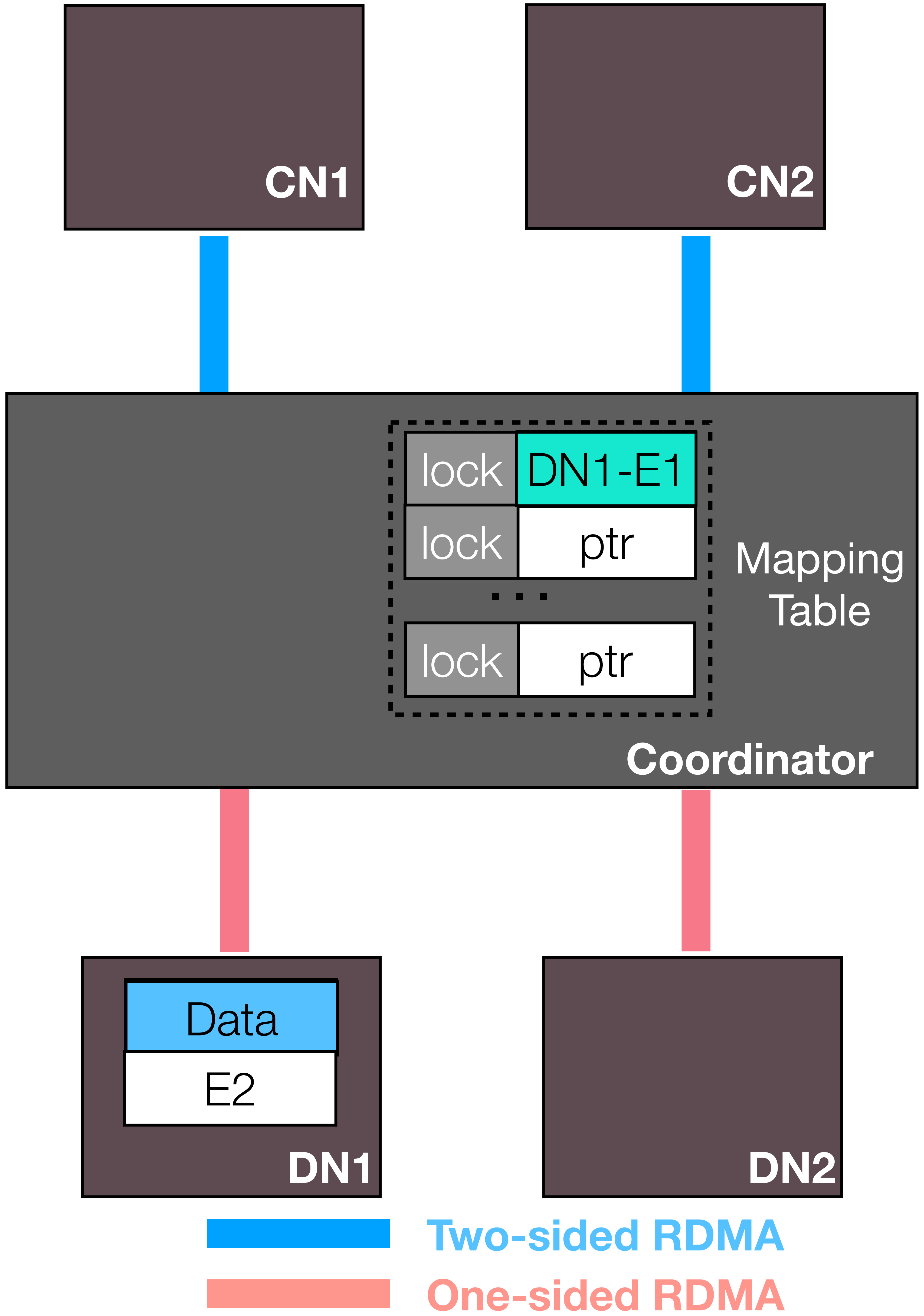


pDPM-Central

Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)
- Coordinator locks the entry in mapping table and update ptr

Read Flow



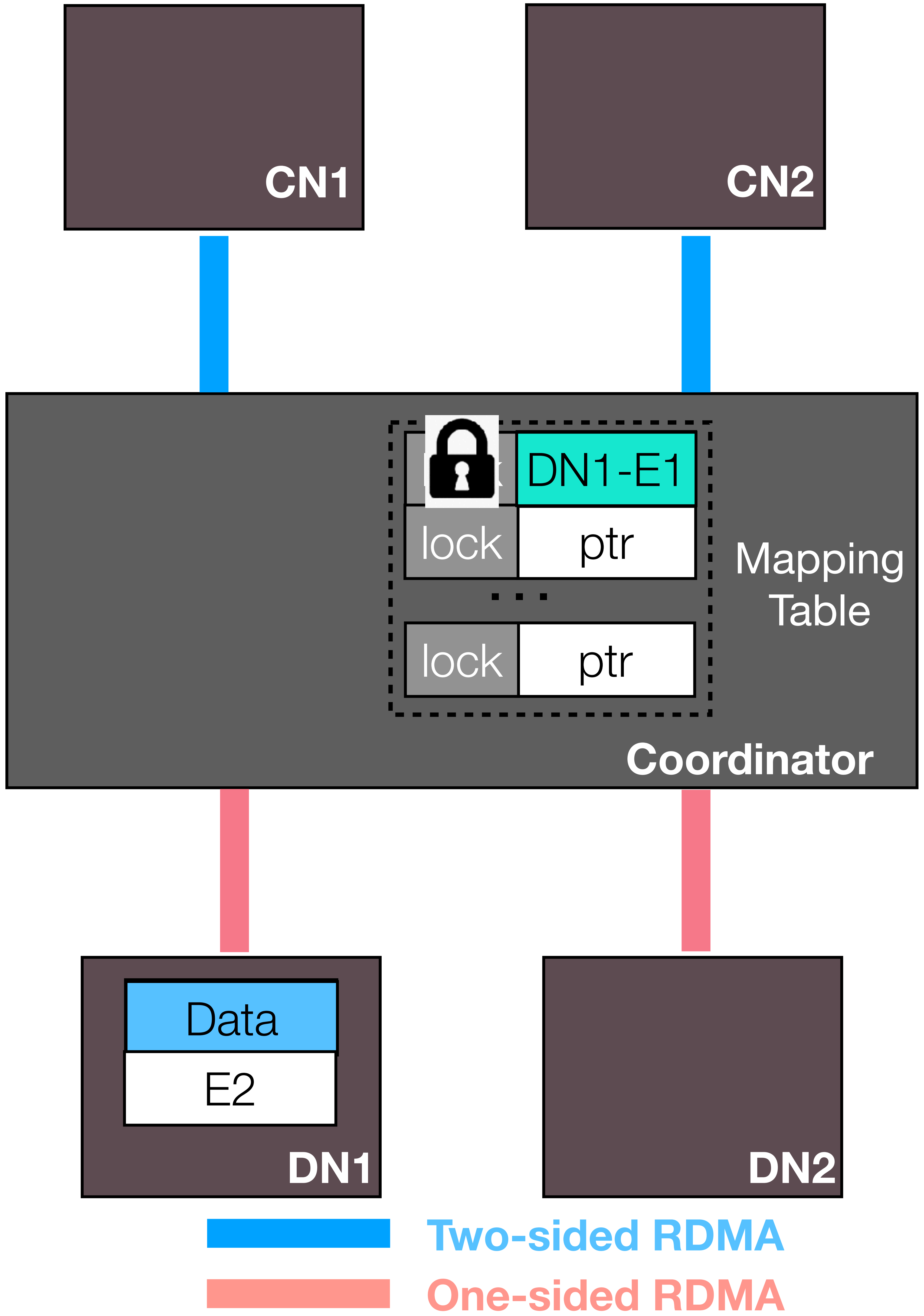
pDPM-Central

Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)
- Coordinator locks the entry in mapping table and update ptr

Read Flow

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



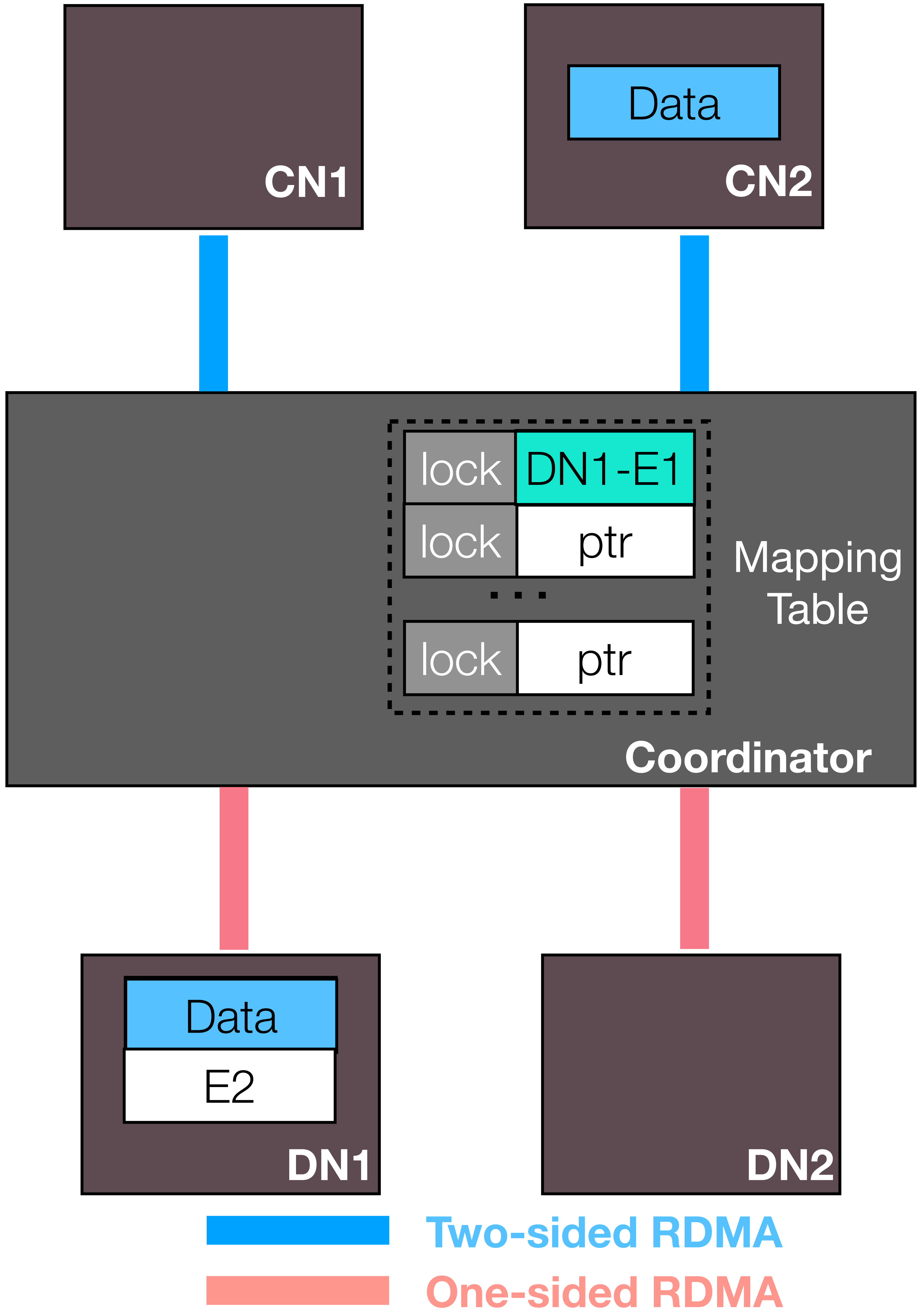
pDPM-Central

Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)
- Coordinator locks the entry in mapping table and update ptr

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

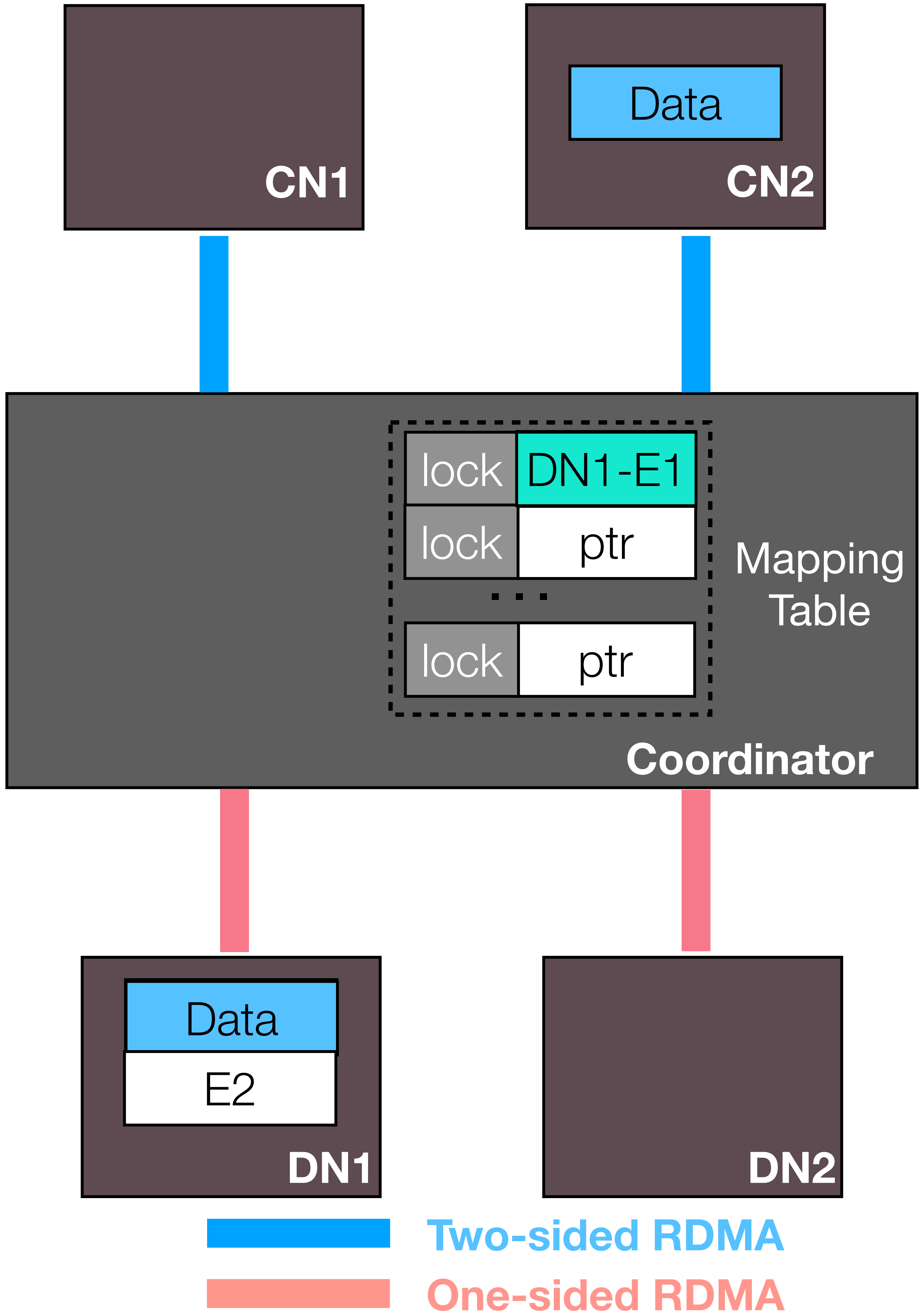
- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)
- Coordinator locks the entry in mapping table and update ptr

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



pDPM-Central

Write Flow

- CN sends RPC (with data) to Coordinator
- Coordinator alloc a new DN entry, and write data to it (as redo-copy)
- Coordinator locks the entry in mapping table and update ptr

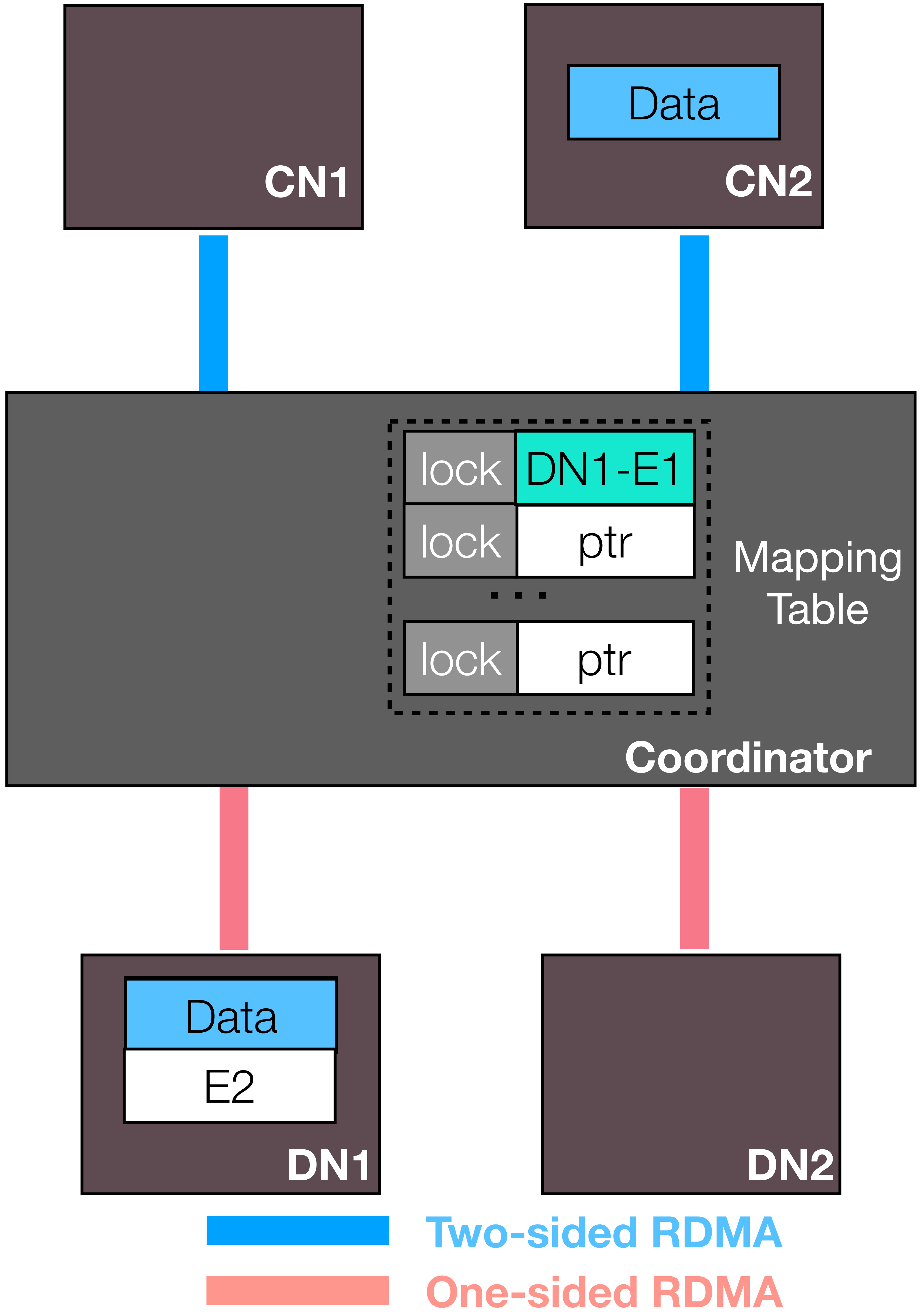
Read Flow

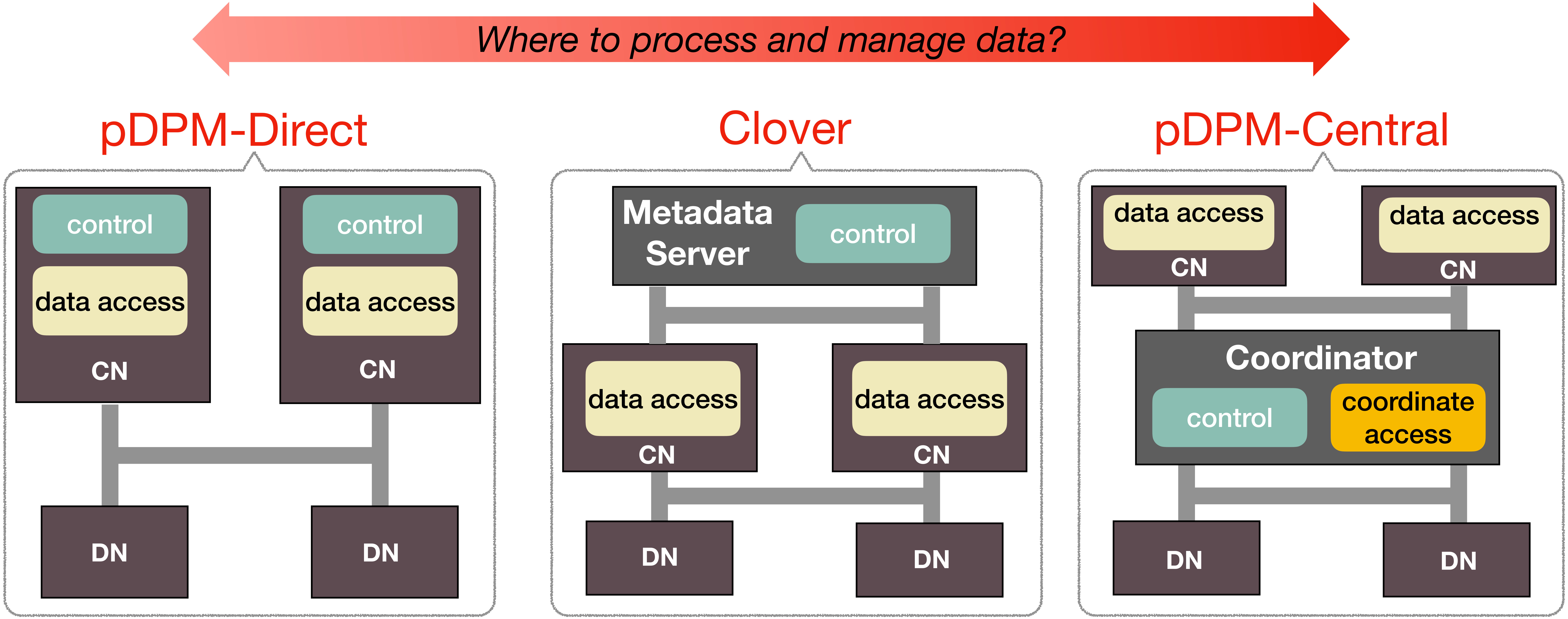
- CN sends RPC to Coordinator
- Coordinator locks the entry in mapping table
- Coordinator reads data from DN and then replies to CN

All cases
Read: 2 RTTs
Write: 2 RTTs

☹

Slower read
Poor scalability: coordinator is the bottleneck





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

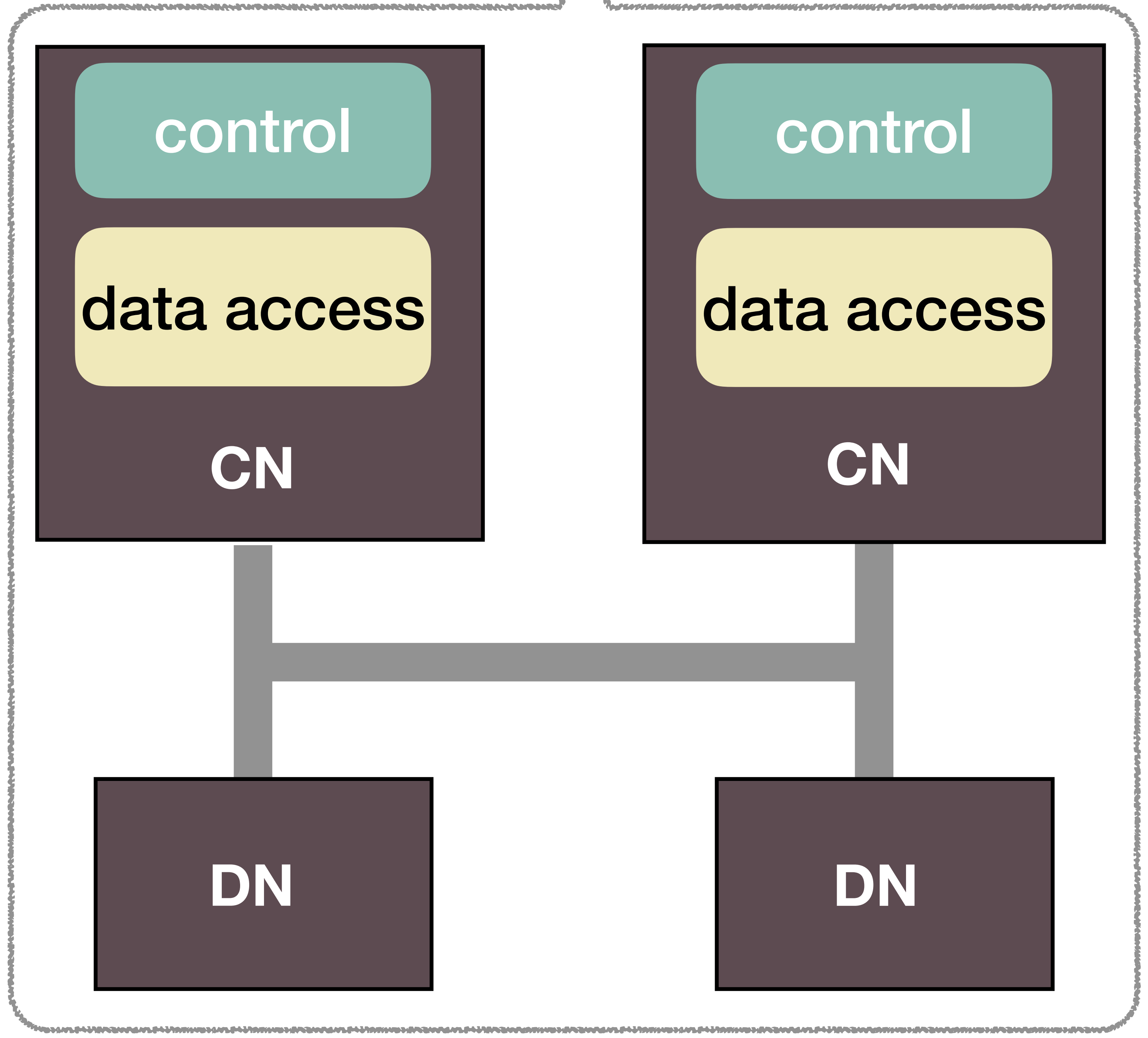
coordinate access

DN

DN

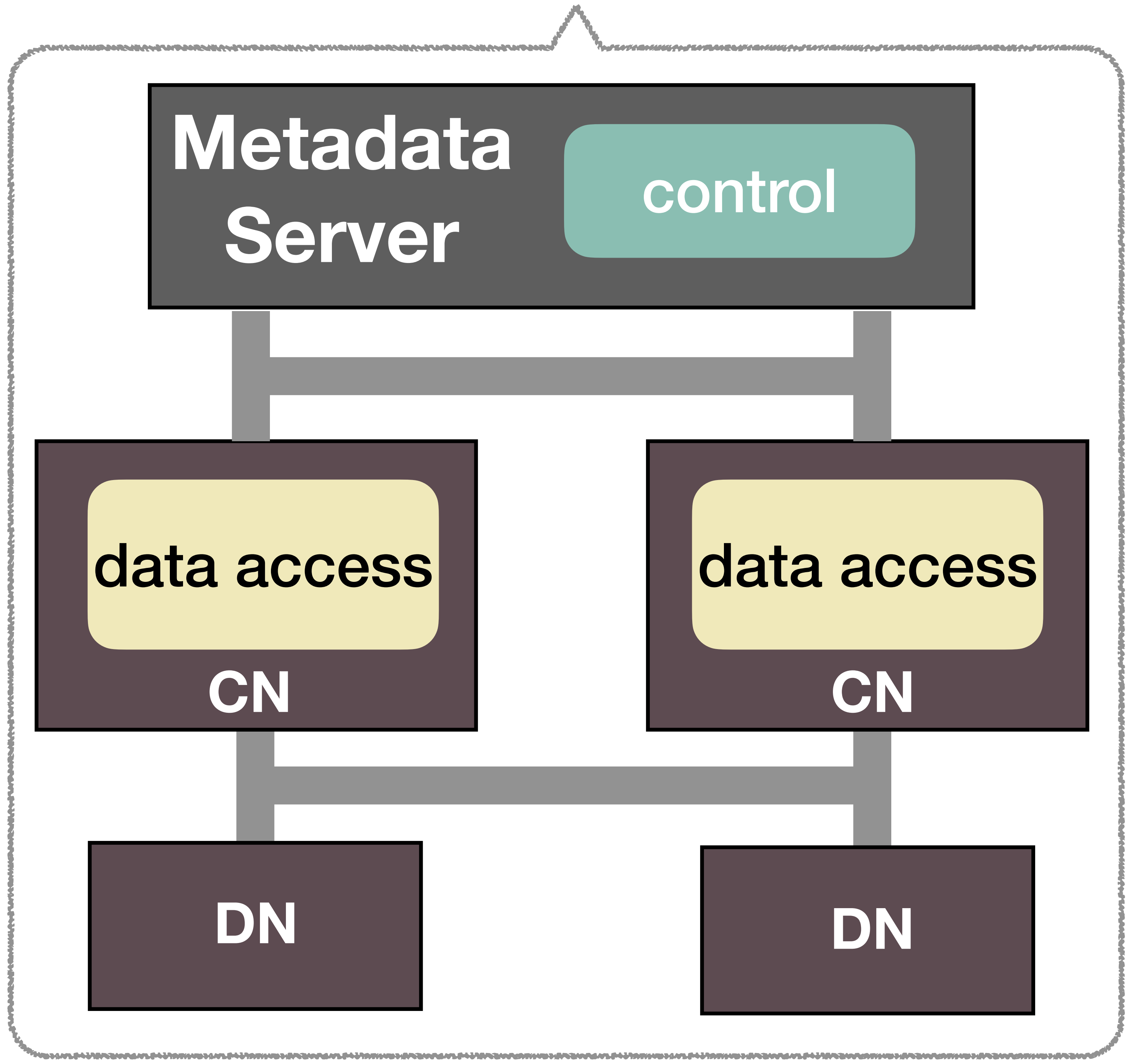
Where to process and manage data?

pDPM-Direct

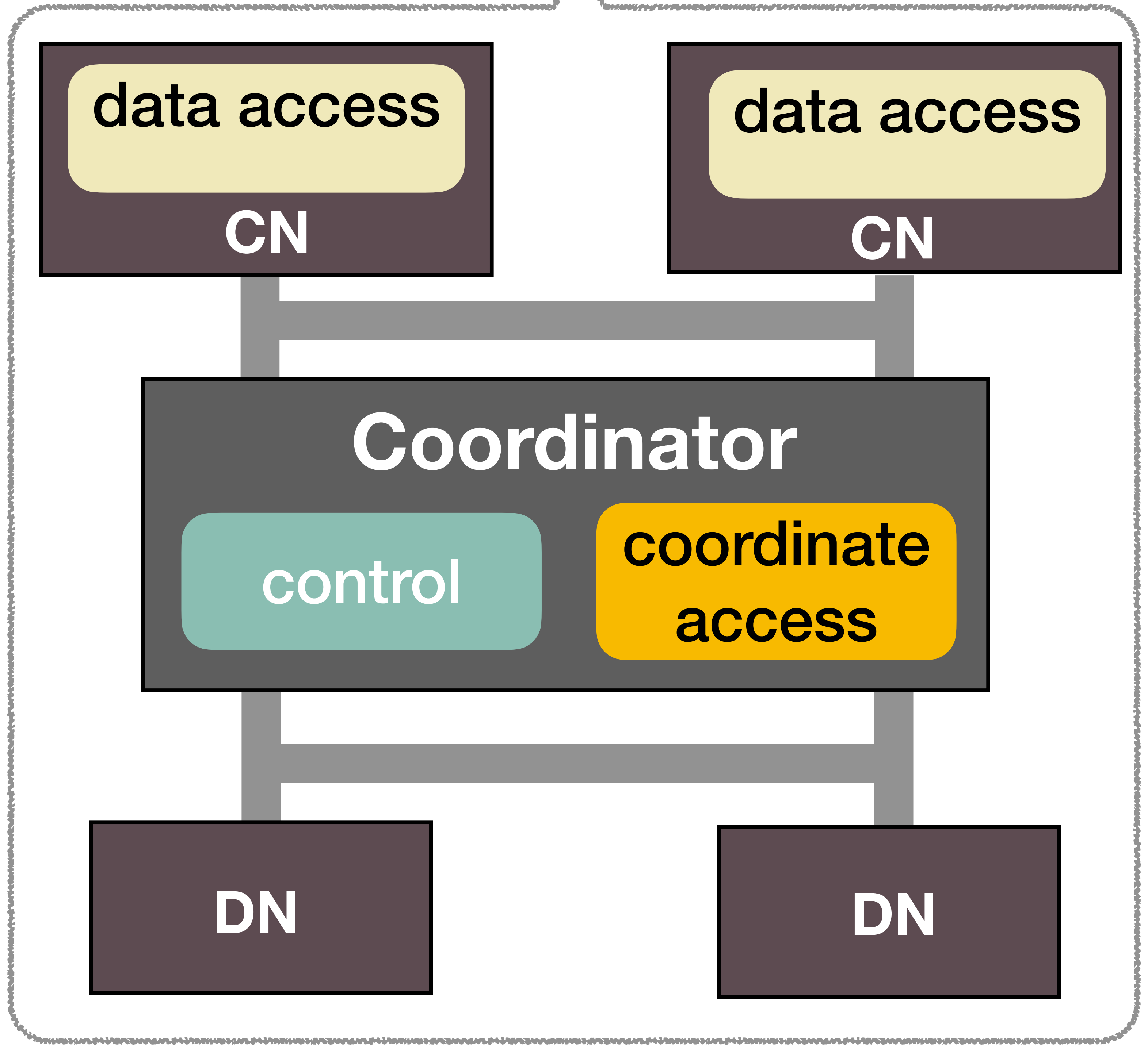


- Write cannot scale
- Large metadata consumption

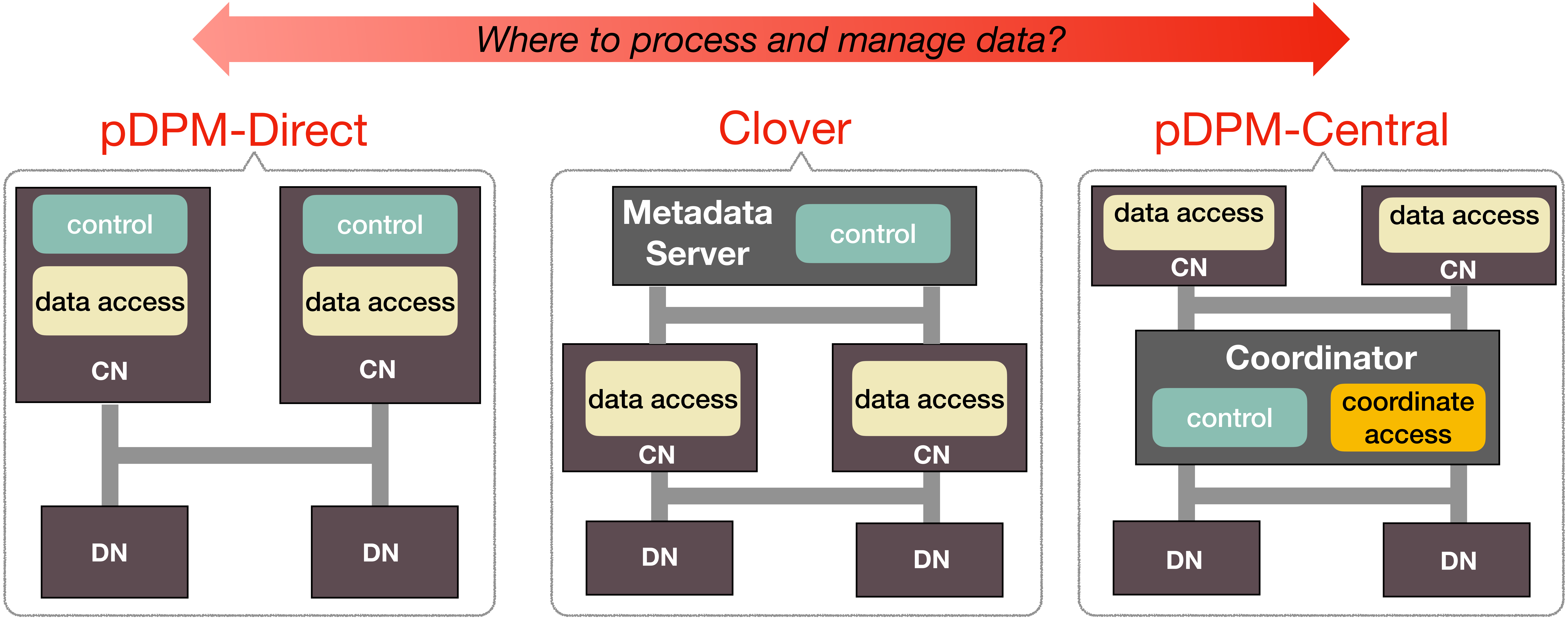
Clover



pDPM-Central



- Extra read RTTs
- Coordinator cannot scale

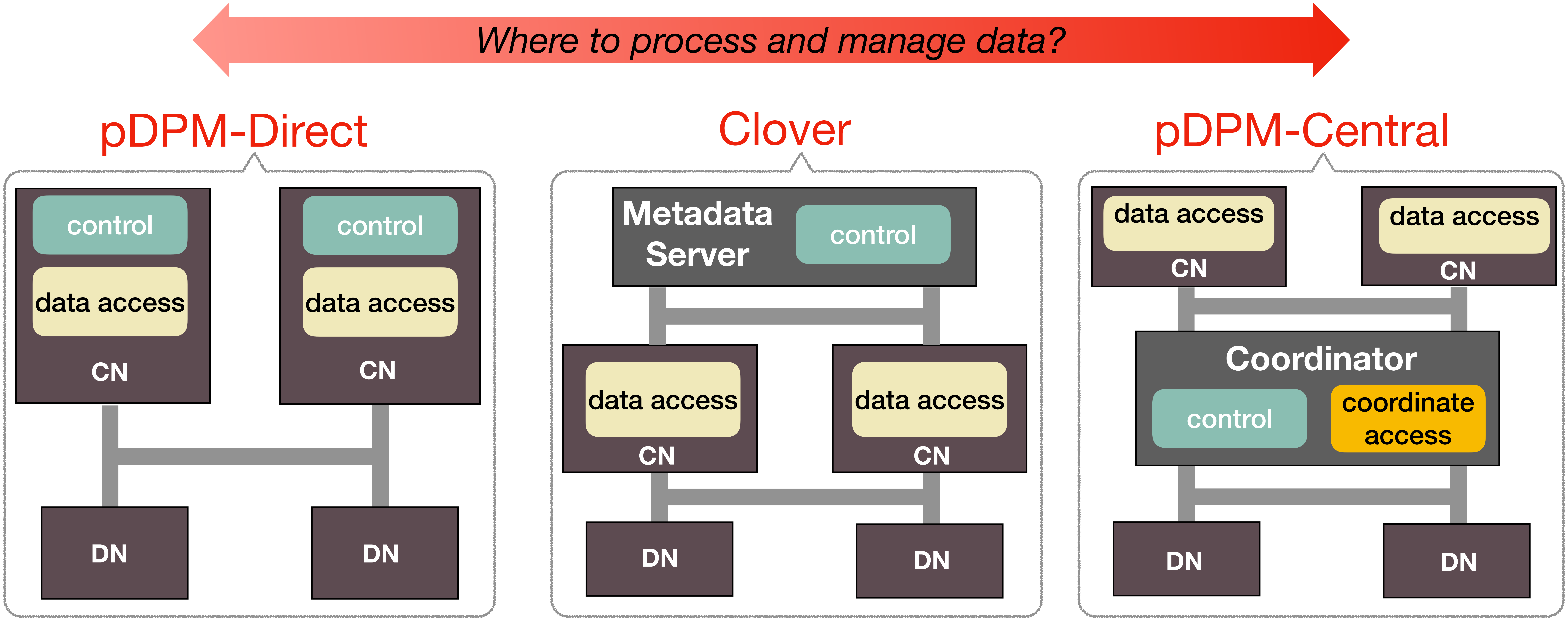


- Write cannot scale
- Large metadata consumption

- Extra read RTTs
- Coordinator cannot scale

Distributed data & metadata planes

Centralized data & metadata planes



- Write cannot scale
- Large metadata consumption

Distributed data & metadata planes

Separate data & metadata planes

- Extra read RTTs
- Coordinator cannot scale

Centralized data & metadata planes

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

- Write cannot scale
- Large metadata consumption

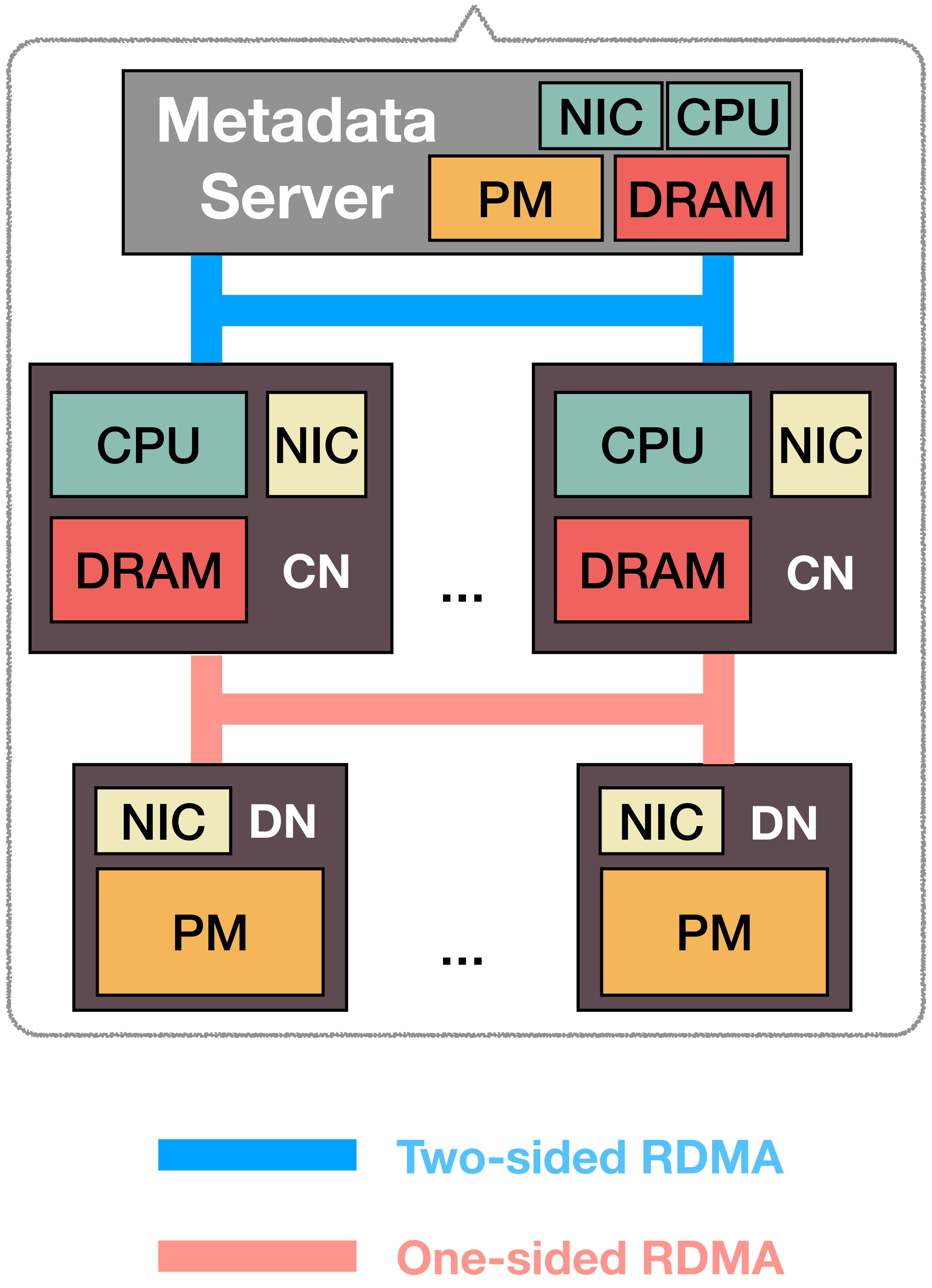
Distributed data & metadata planes

Separate data & metadata planes

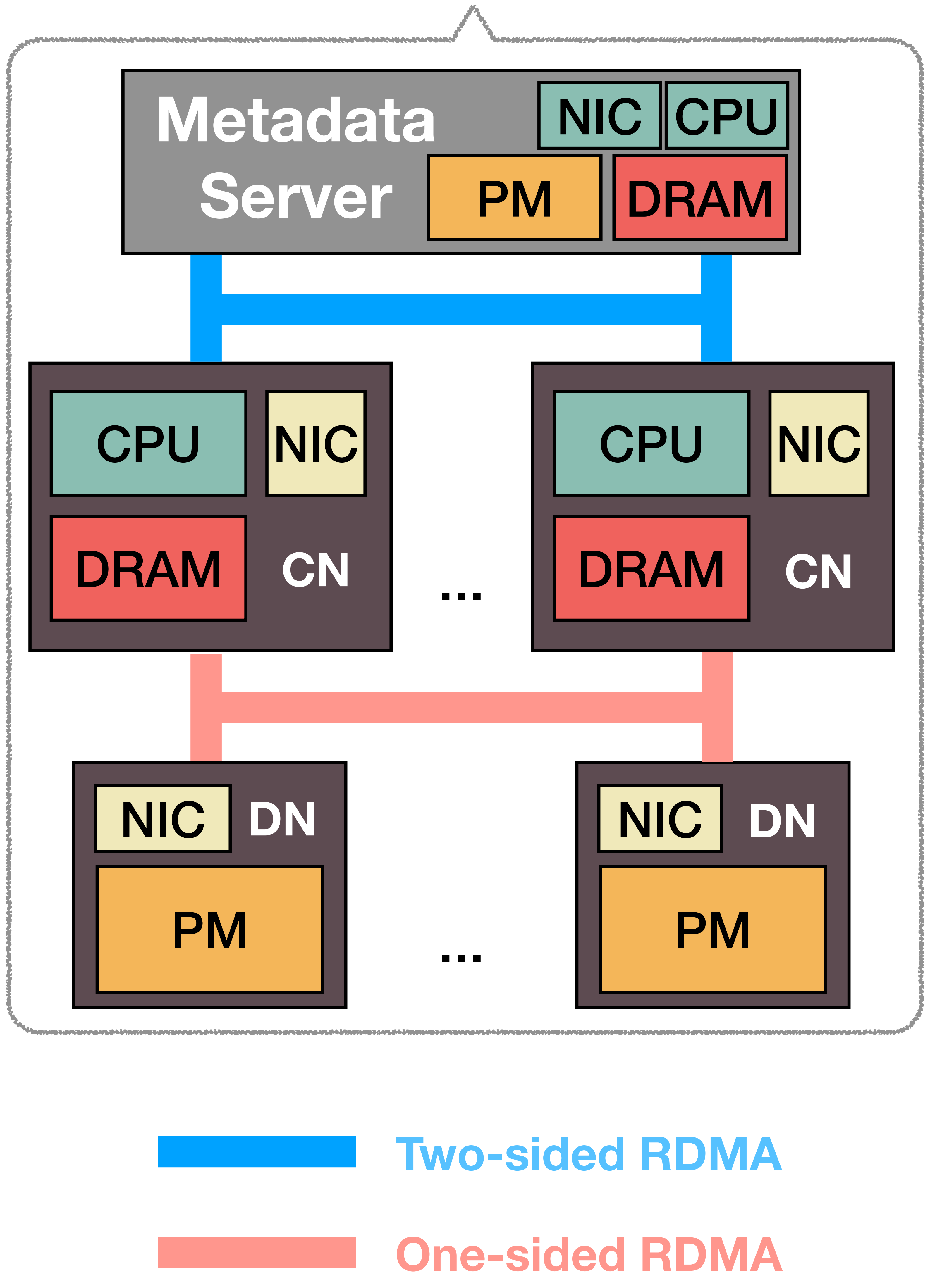
- Extra read RTTs
- Coordinator cannot scale

Centralized data & metadata planes

Clover: Combining Distributed and Centralized Approaches



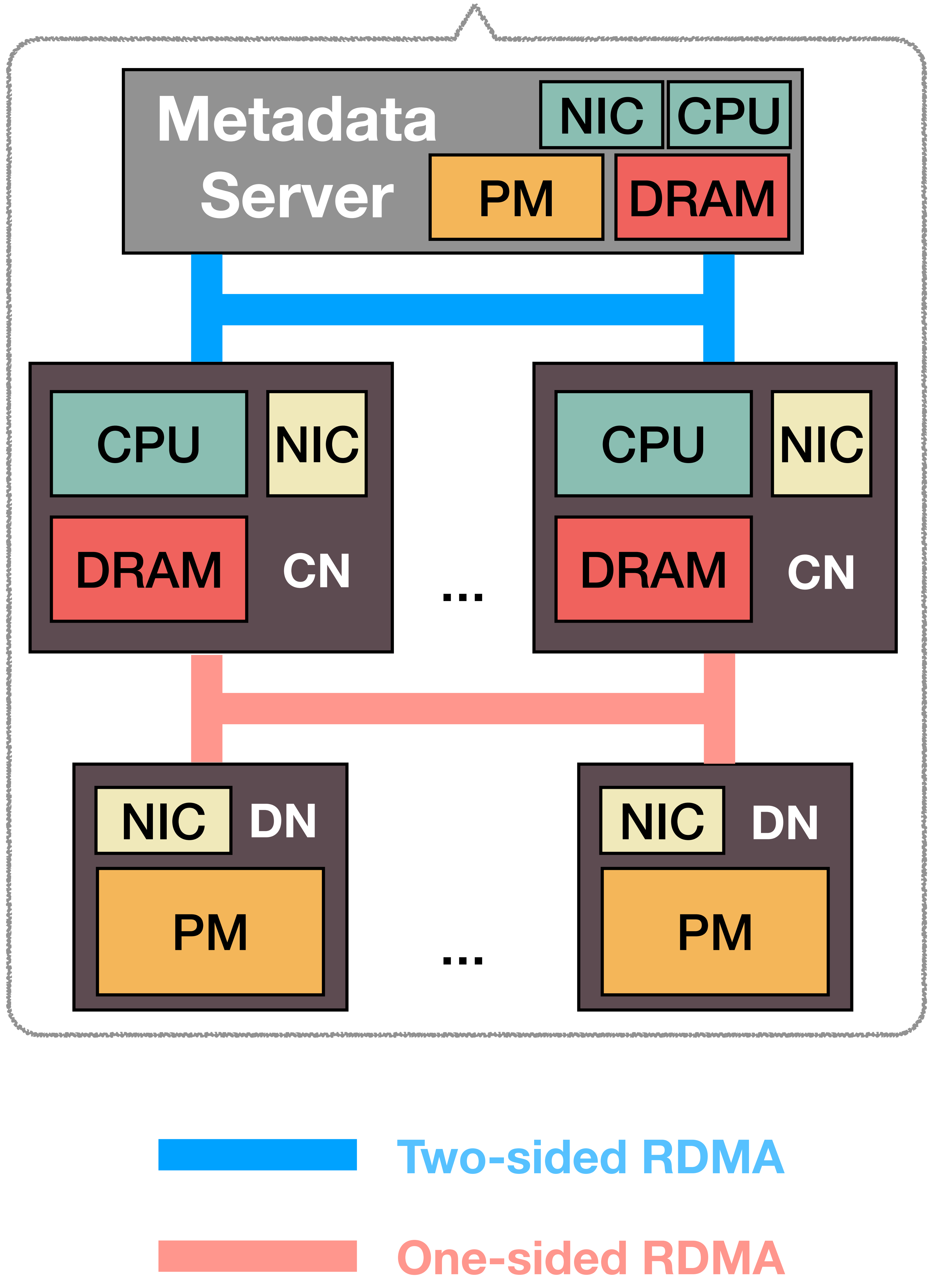
Clover: Combining Distributed and Centralized Approaches



High-level idea: separate data and metadata plane

- Separate locations
- Different communication methods
- Different management strategy

Clover: Combining Distributed and Centralized Approaches



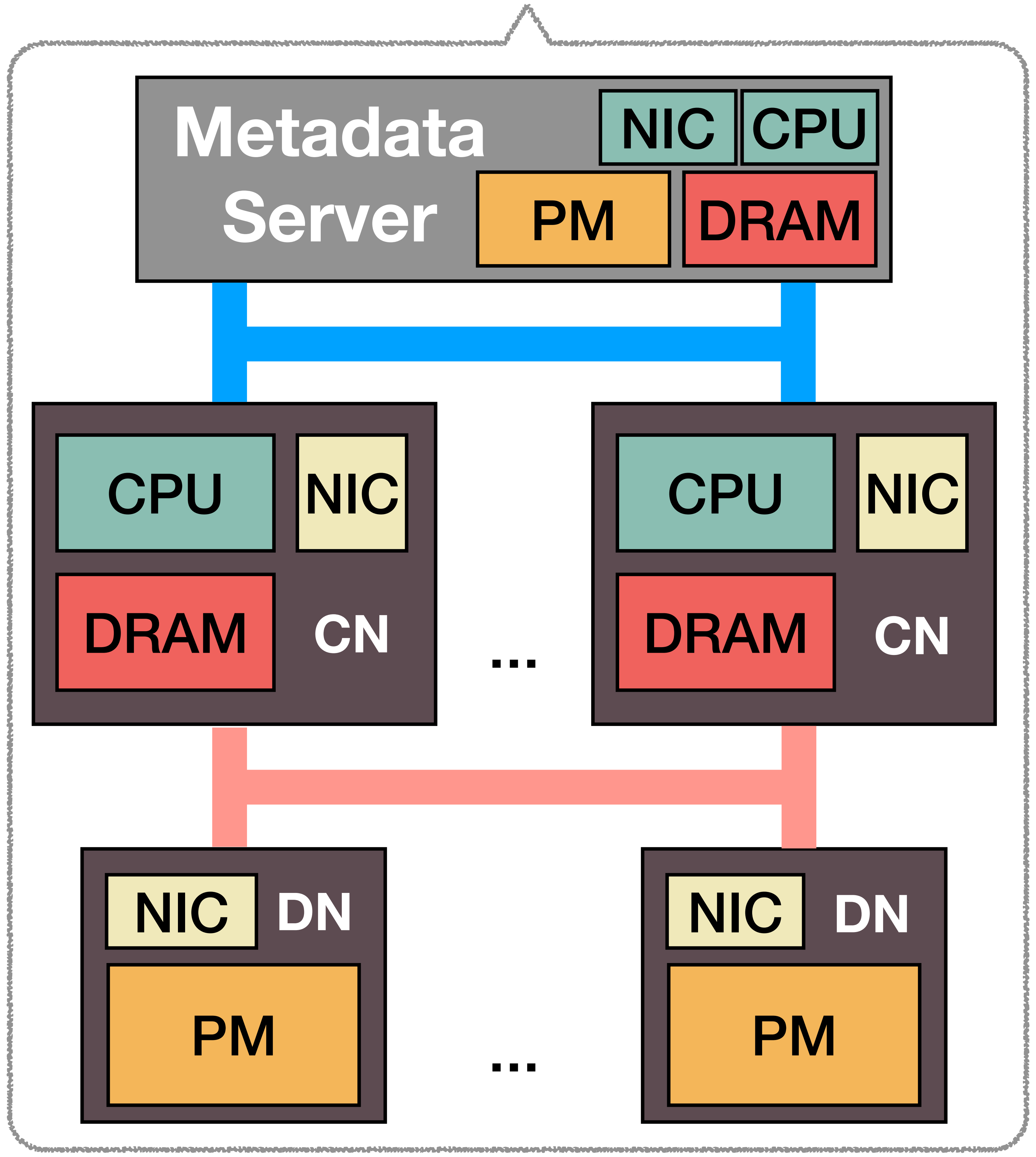
High-level idea: separate data and metadata plane

- 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 approach

- Lock-free data structures to increase scalability
- Optimizations to reduce read/write RTTs

Main Challenge in Data Plane:

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

Our approach

- Lock-free data structures to increase scalability
- Optimizations to reduce read/write RTTs

Our goal is to support concurrent RW w/ read committed and atomic write

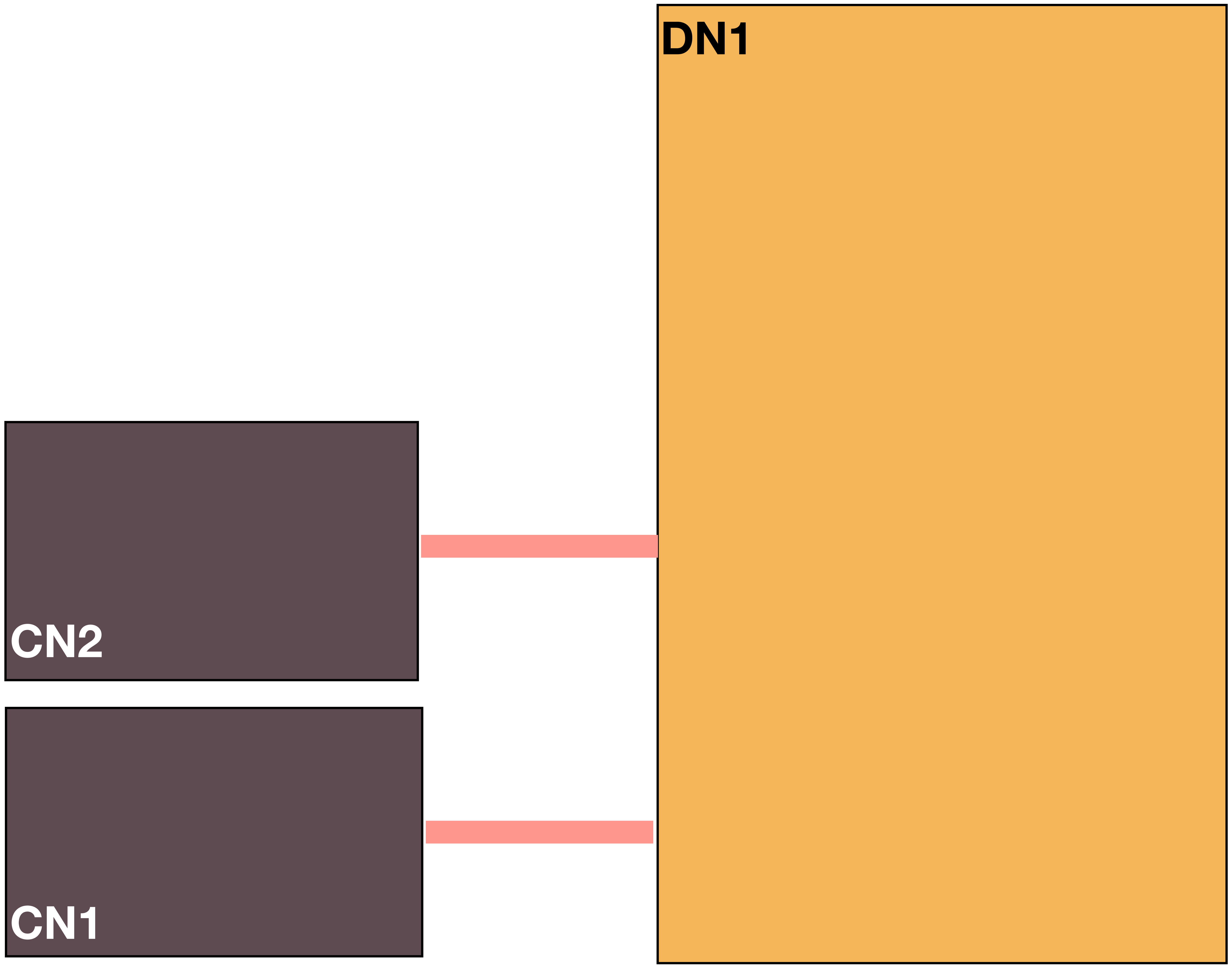
And the challenge is that these RW are un-orchestrated

Main Challenge in Data Plane:

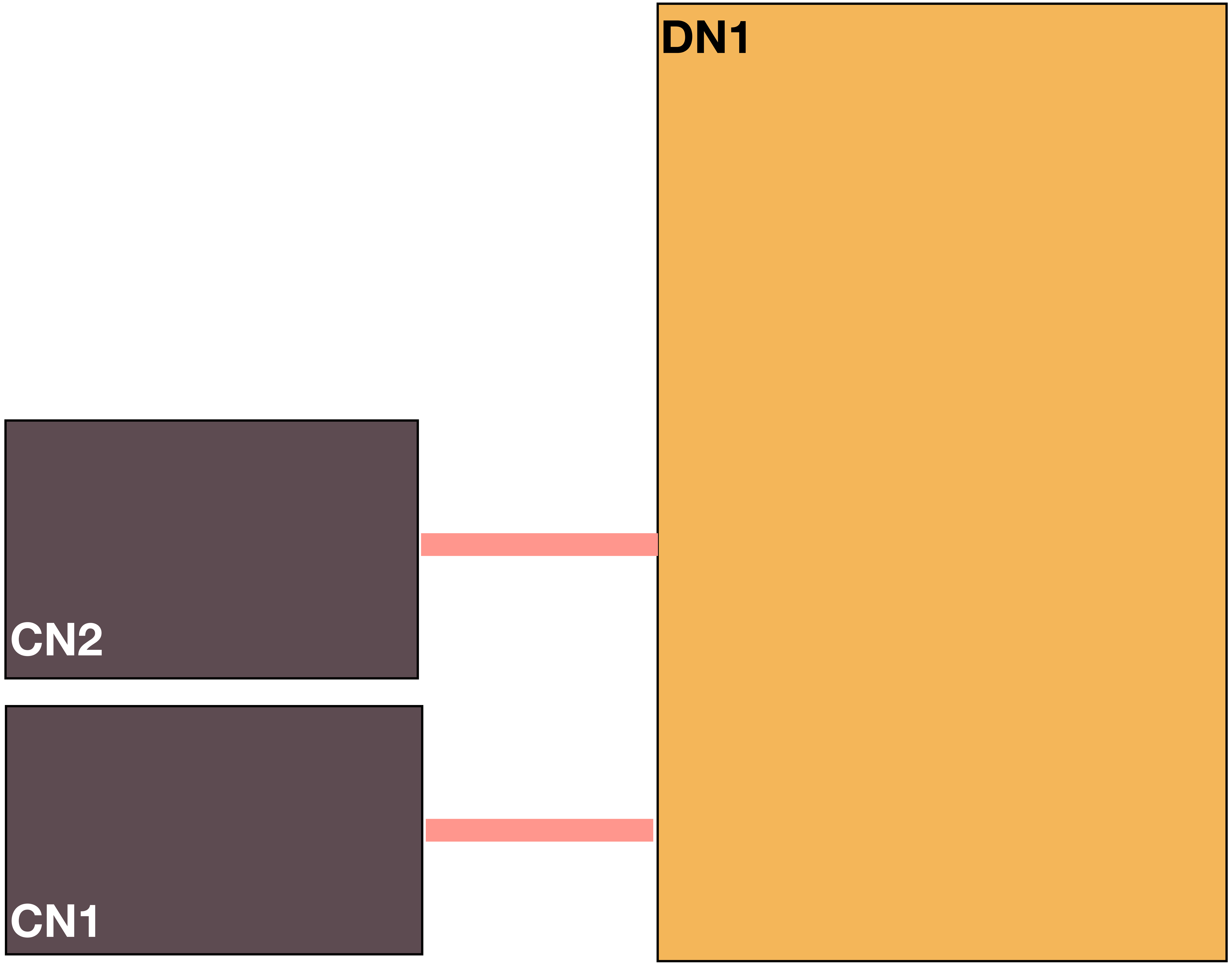
How to efficiently support concurrent data accesses from CNs to DNs?

Our approach

- Lock-free data structures to increase scalability
- Optimizations to reduce read/write RTTs

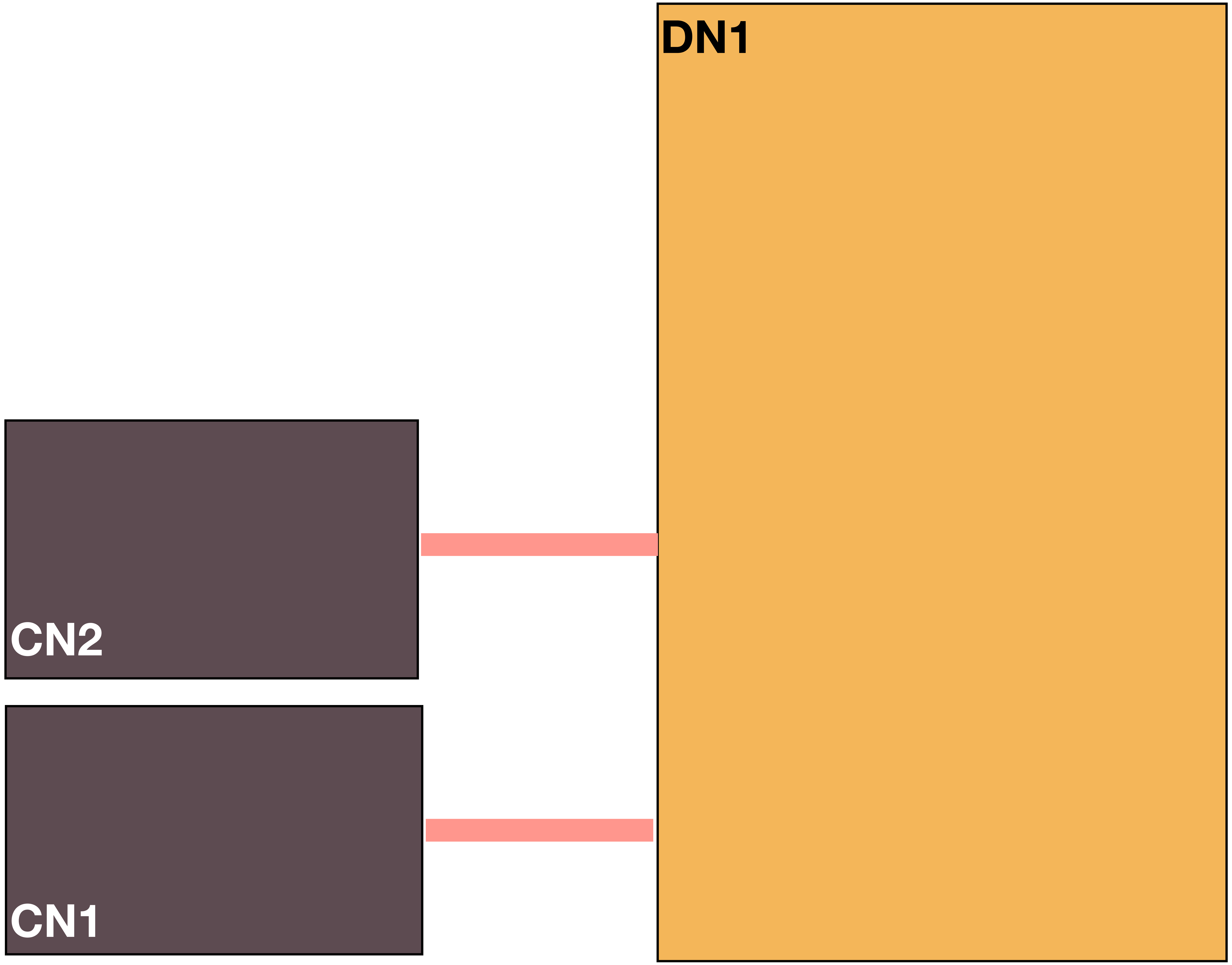

DN1
CN2
CN1

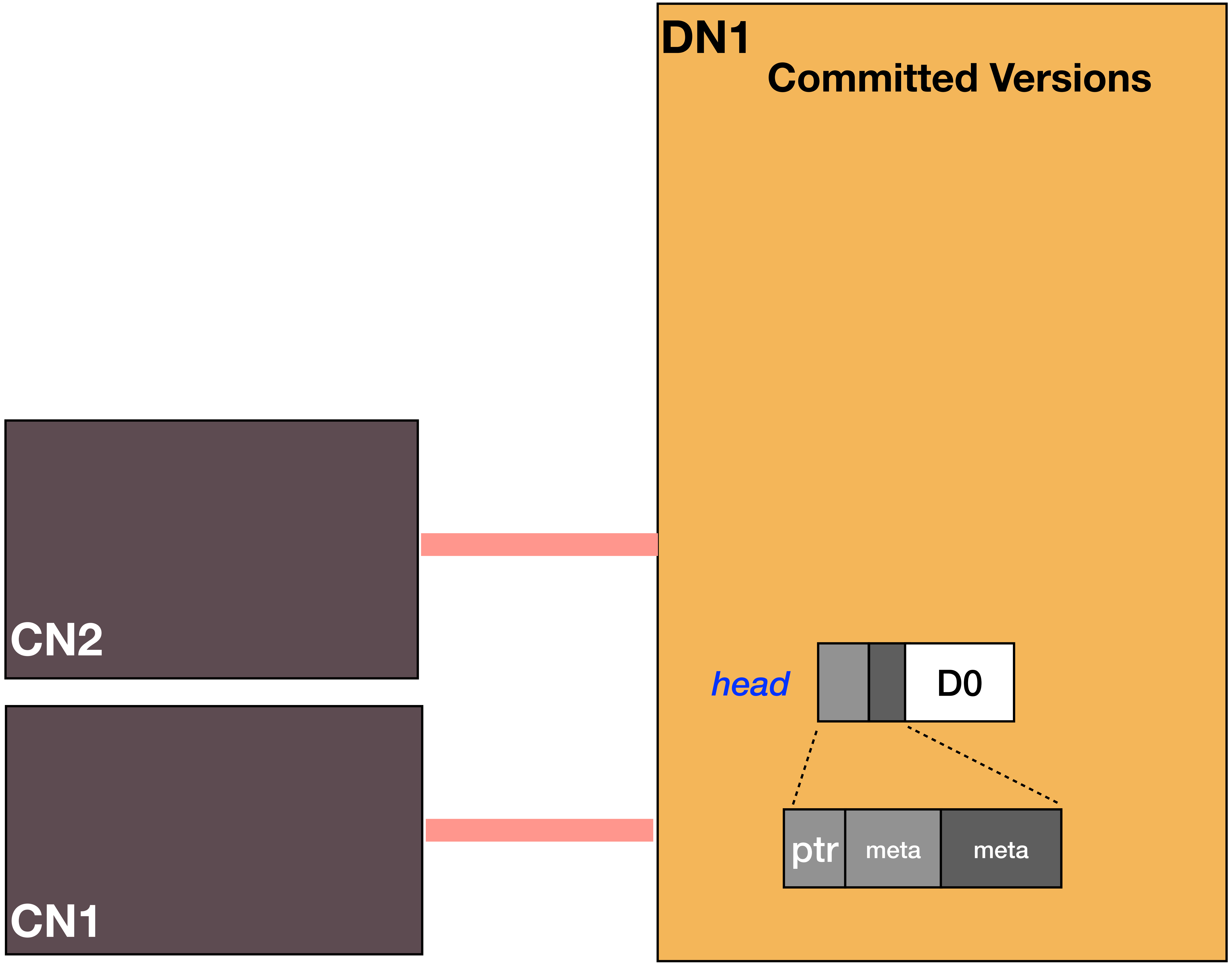
Design: lock-free data structures



Design: lock-free data structures

Our-of-place write (redo copy)





Design: lock-free data structures

Our-of-place write (redo copy)

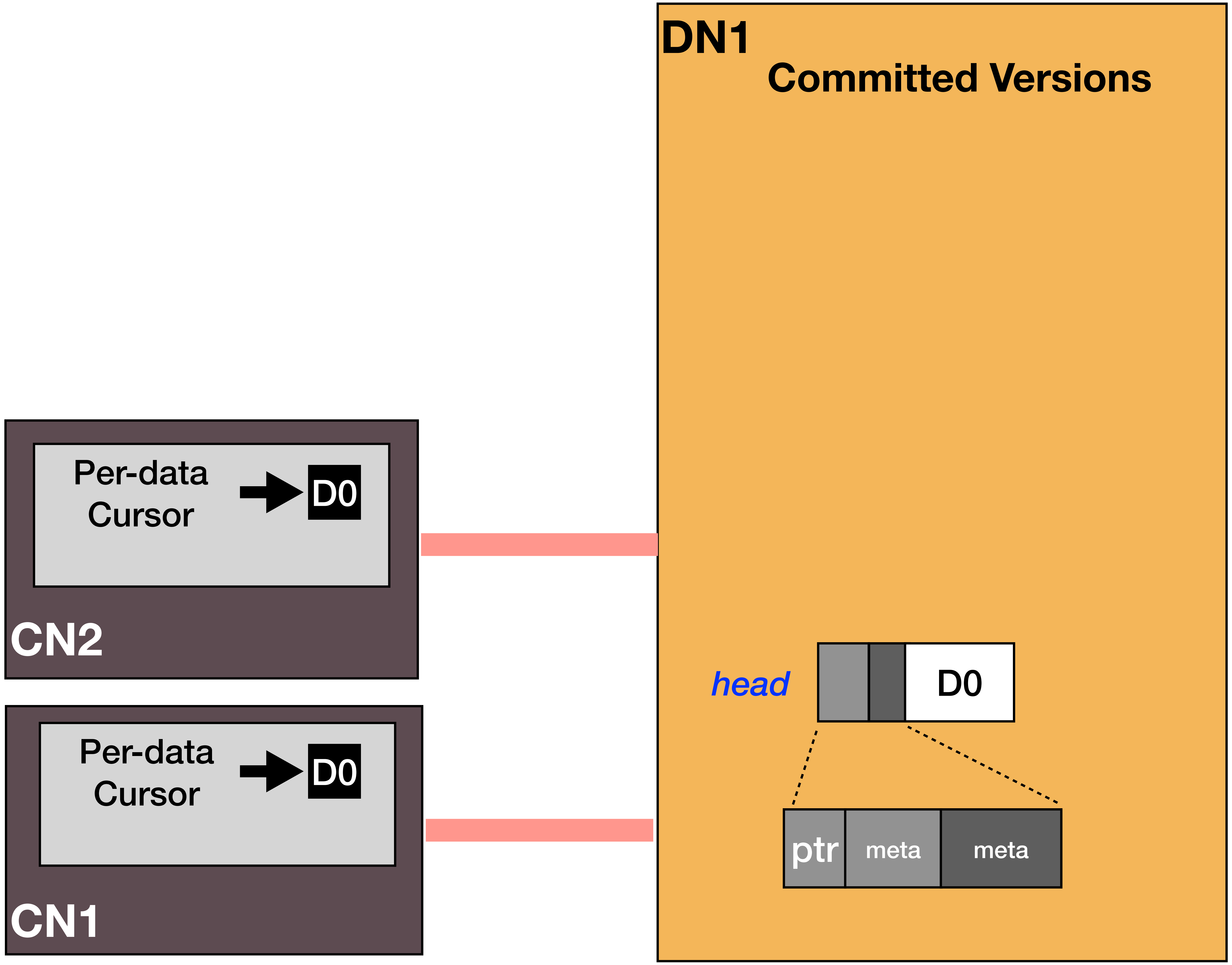
Chained redo copies at `DN`

Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at `DN`

`CN` caches a ***cursor*** points to a version

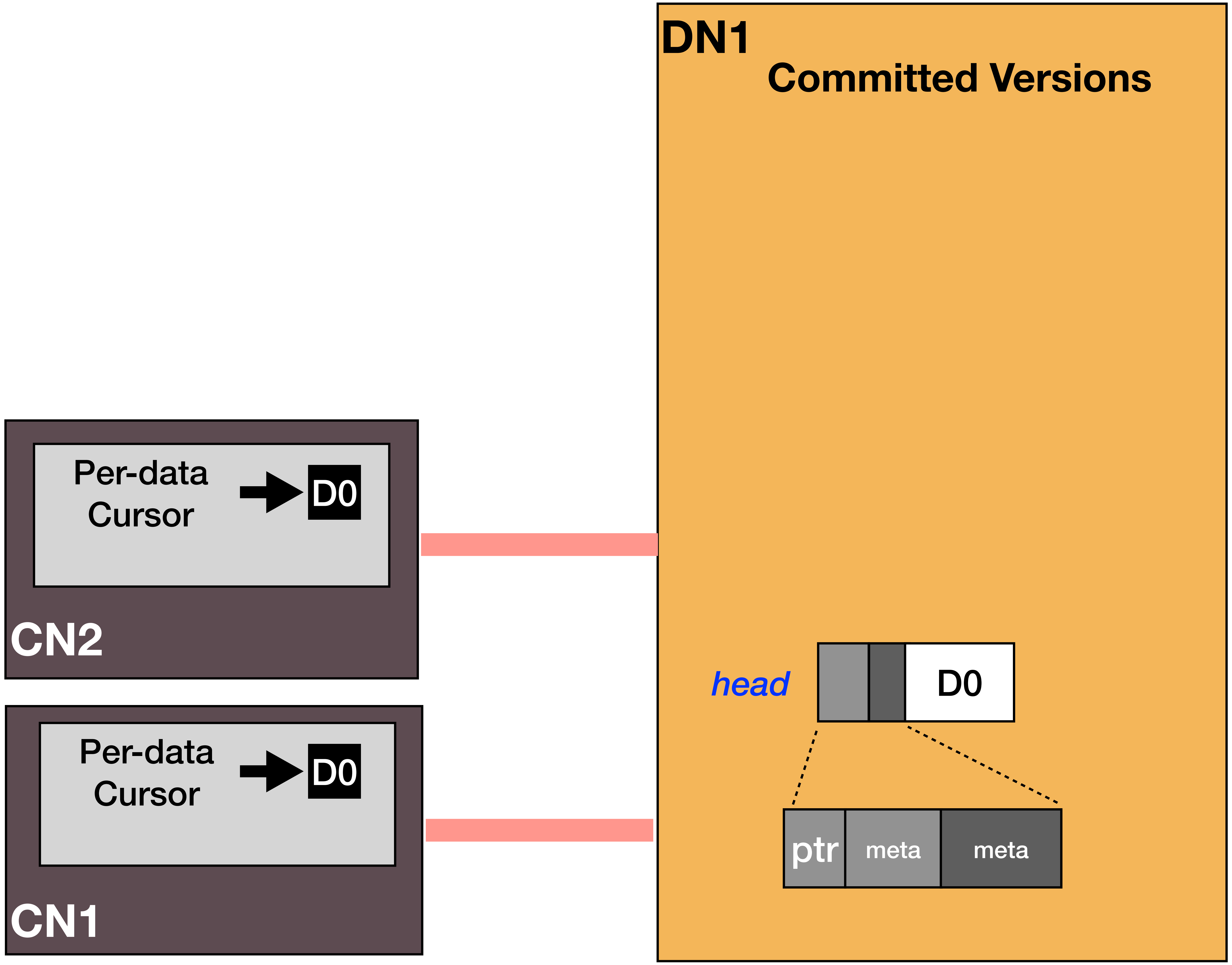


Design: lock-free data structures

Our-of-place write (redo copy)

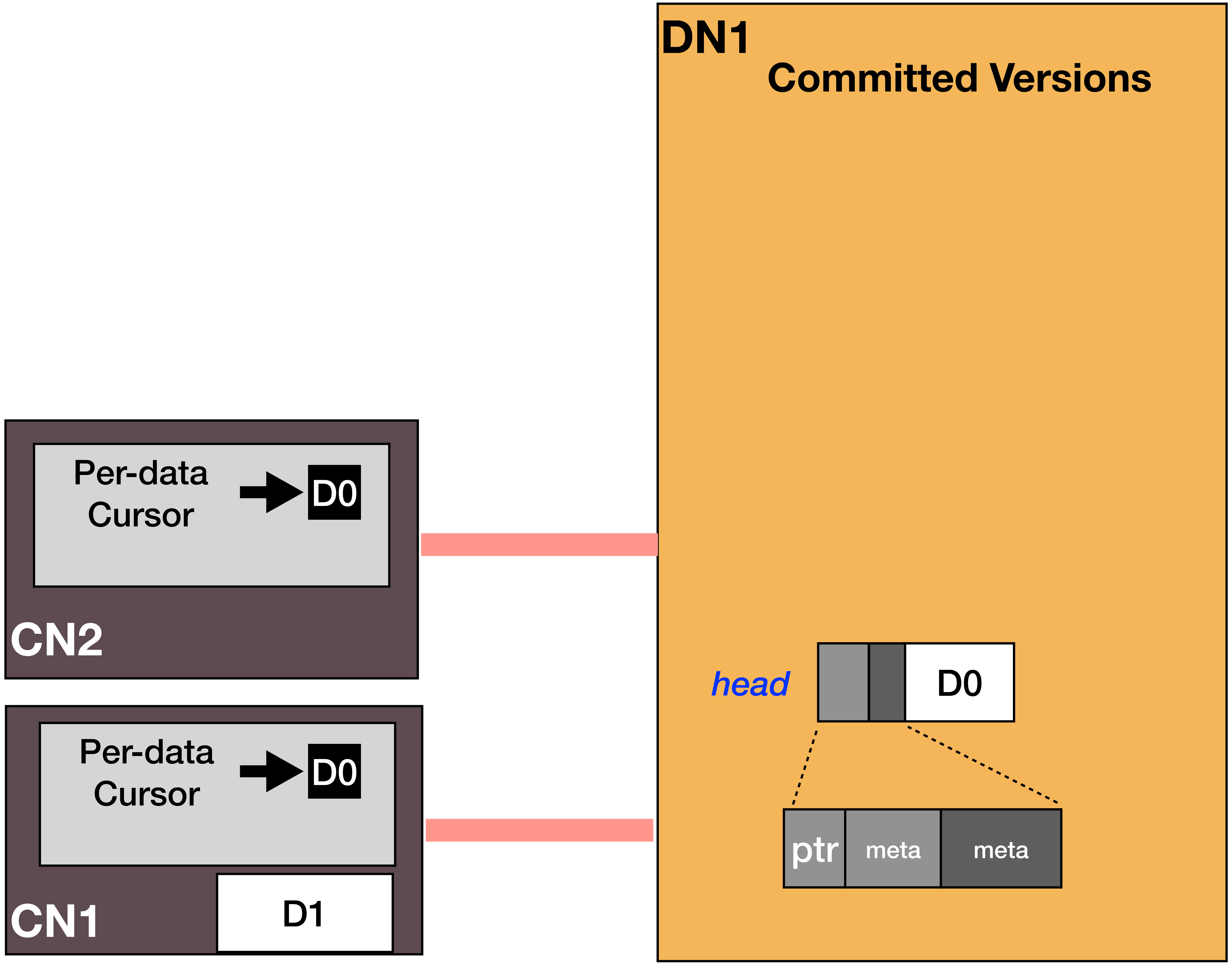
Chained redo copies at `DN`

`CN` caches a ***cursor*** 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



Design: lock-free data structures

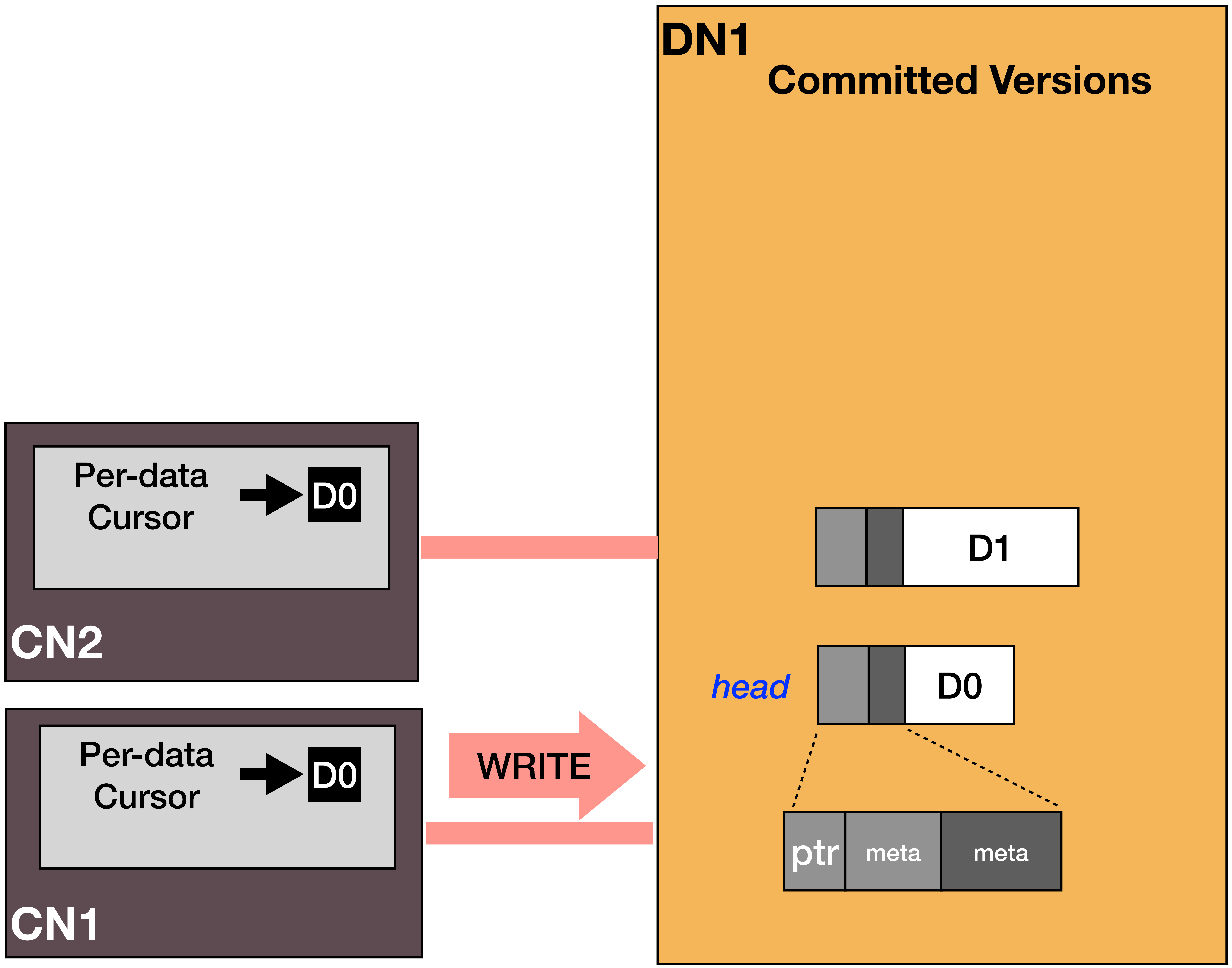
Our-of-place write (redo copy)

Chained redo copies at `DN`

`CN` caches a ***cursor*** 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



Design: lock-free data structures

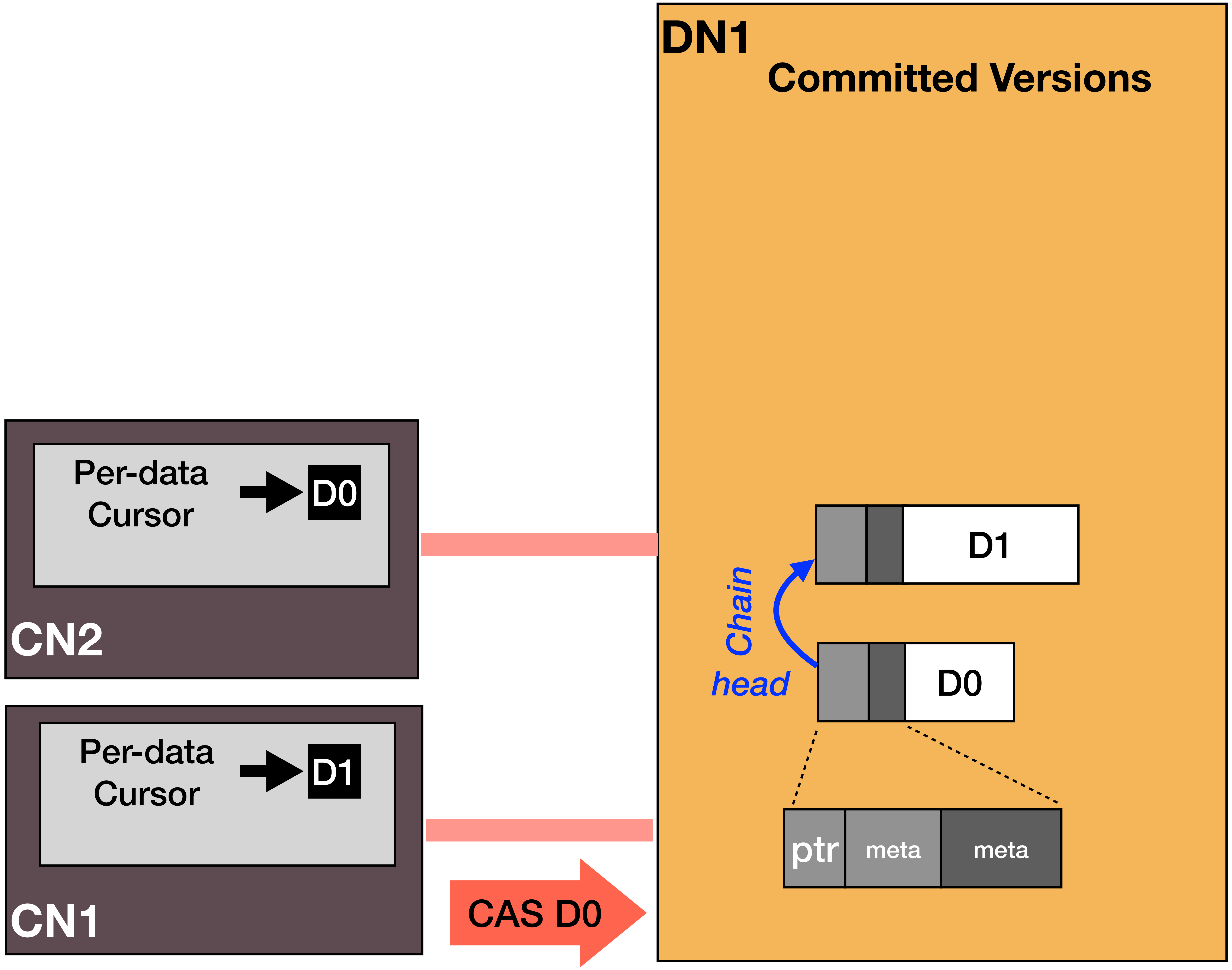
Our-of-place write (redo copy)

Chained redo copies at `DN`

`CN` caches a ***cursor*** 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



Design: lock-free data structures

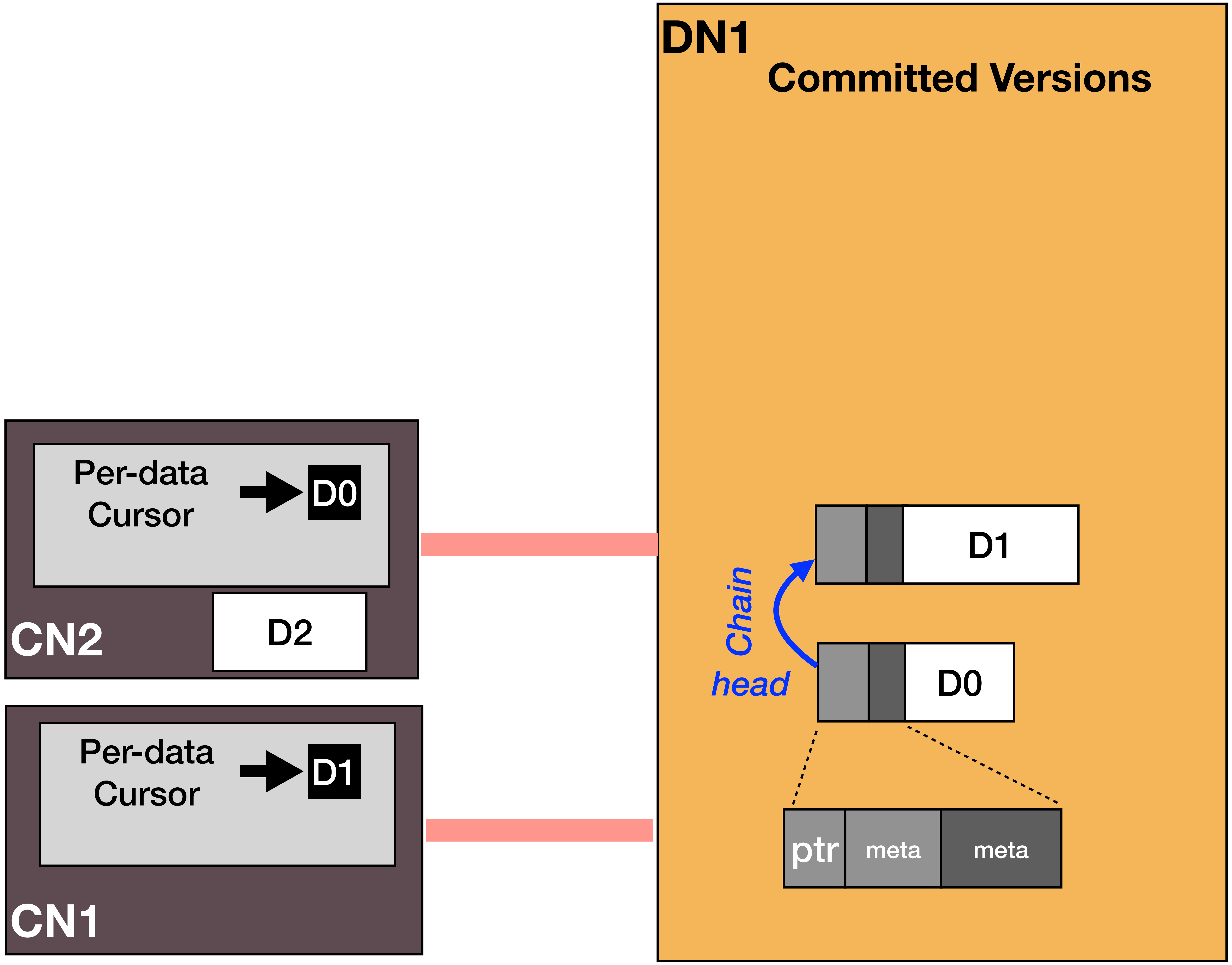
Our-of-place write (redo copy)

Chained redo copies at DN

CN caches a ***cursor*** 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



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

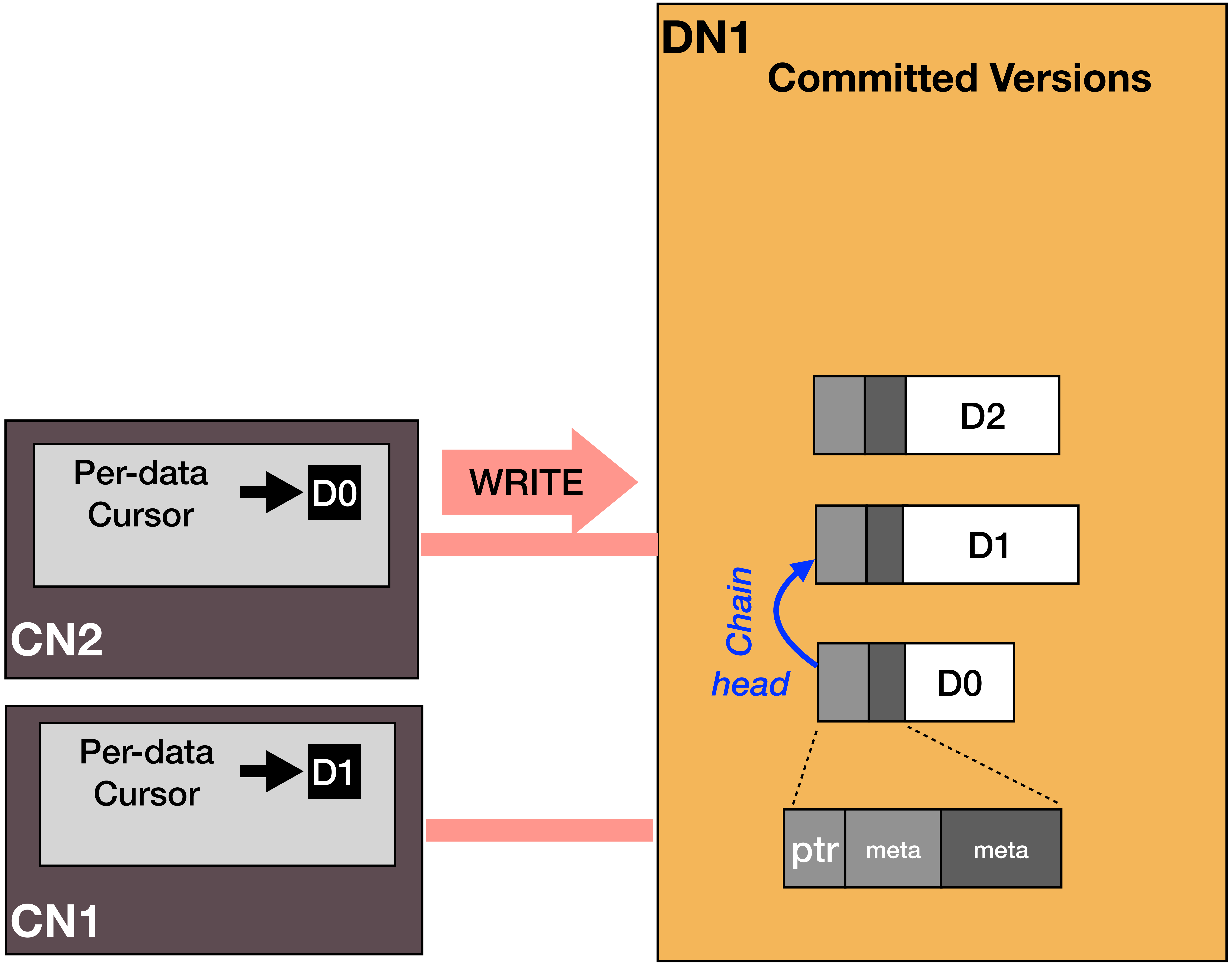
CN caches a ***cursor*** points to a version

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

E1: CN1 writes D1. Update cursor.

E2: CN2 writes D2. Two CAS.



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at `DN`

`CN` caches a ***cursor*** 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

E1: CN1 writes D1. Update cursor.

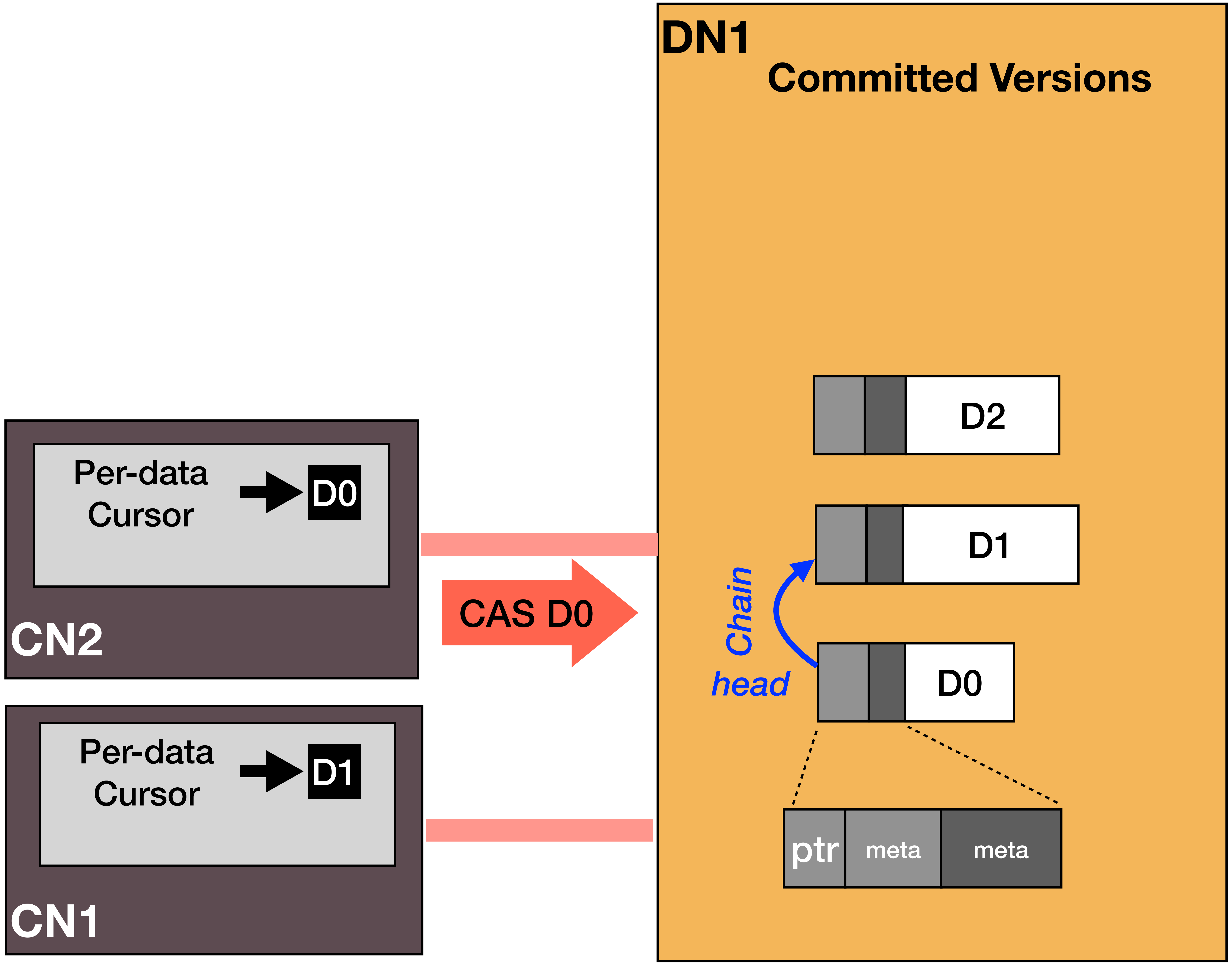
E2: CN2 writes D2. Two CAS.

Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

CN caches a ***cursor*** points to a version

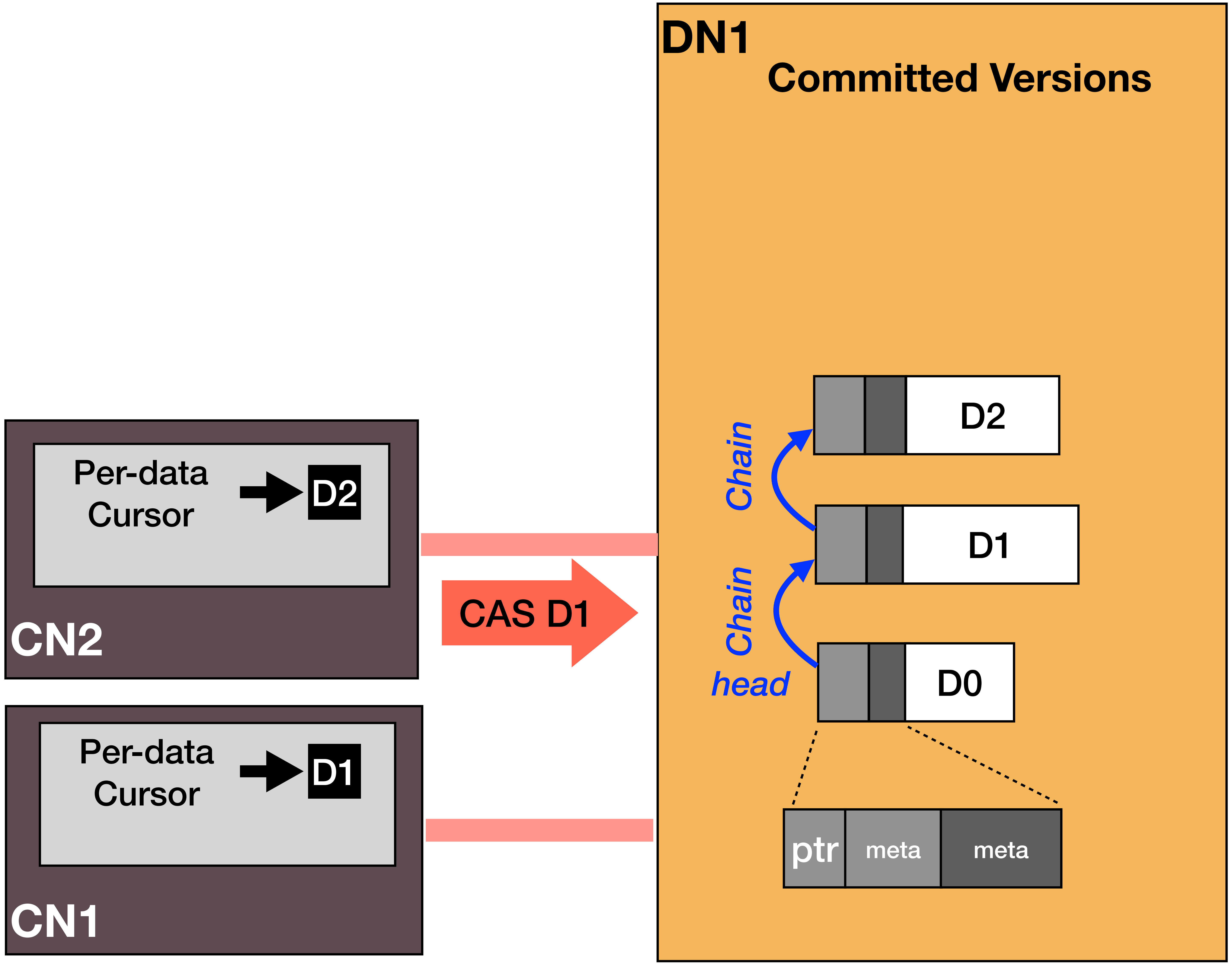


Write Flow

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

E1: CN1 writes D1. Update cursor.

E2: CN2 writes D2. Two CAS.



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at `DN`

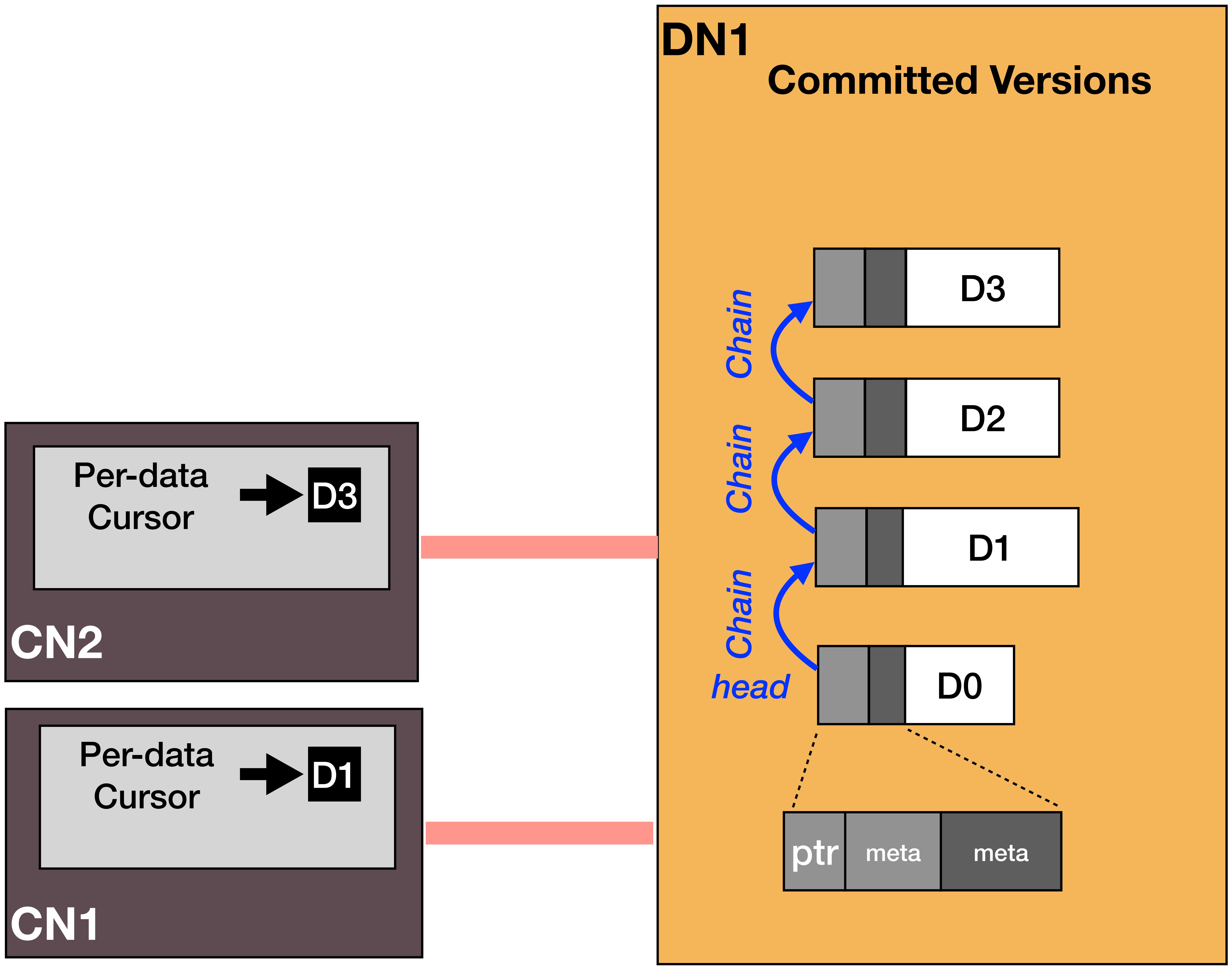
`CN` caches a ***cursor*** points to a version

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

E1: CN1 writes D1. Update cursor.

E2: CN2 writes D2. Two CAS.



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

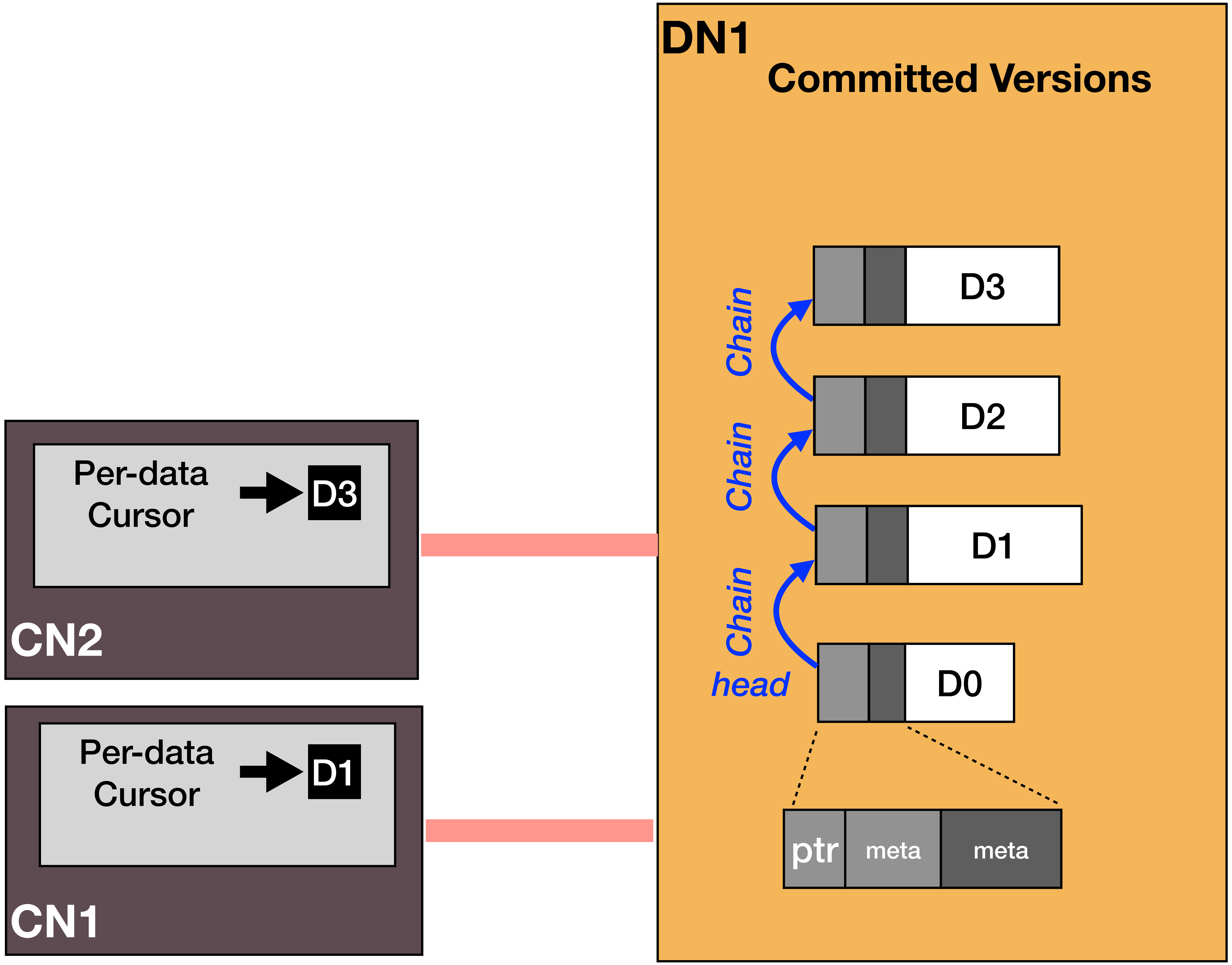
CN caches a ***cursor*** points to a version

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

E1: CN1 writes D1. Update cursor.

E2: CN2 writes D2. Two CAS.



Design: lock-free data structures

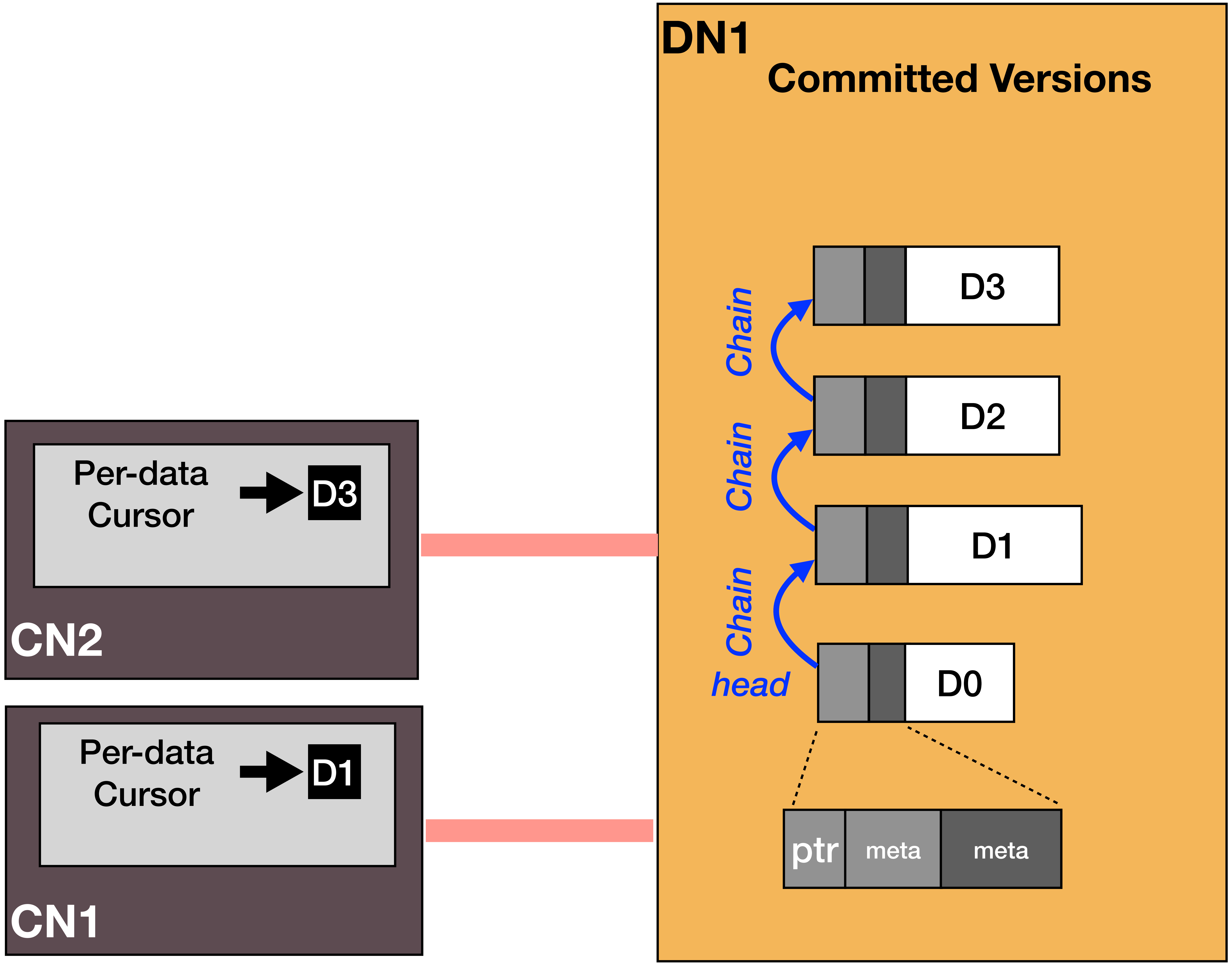
Our-of-place write (redo copy)

Chained redo copies at `DN`

`CN` caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***`c&s`***
3. If 2. fails, update cursor and retry



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

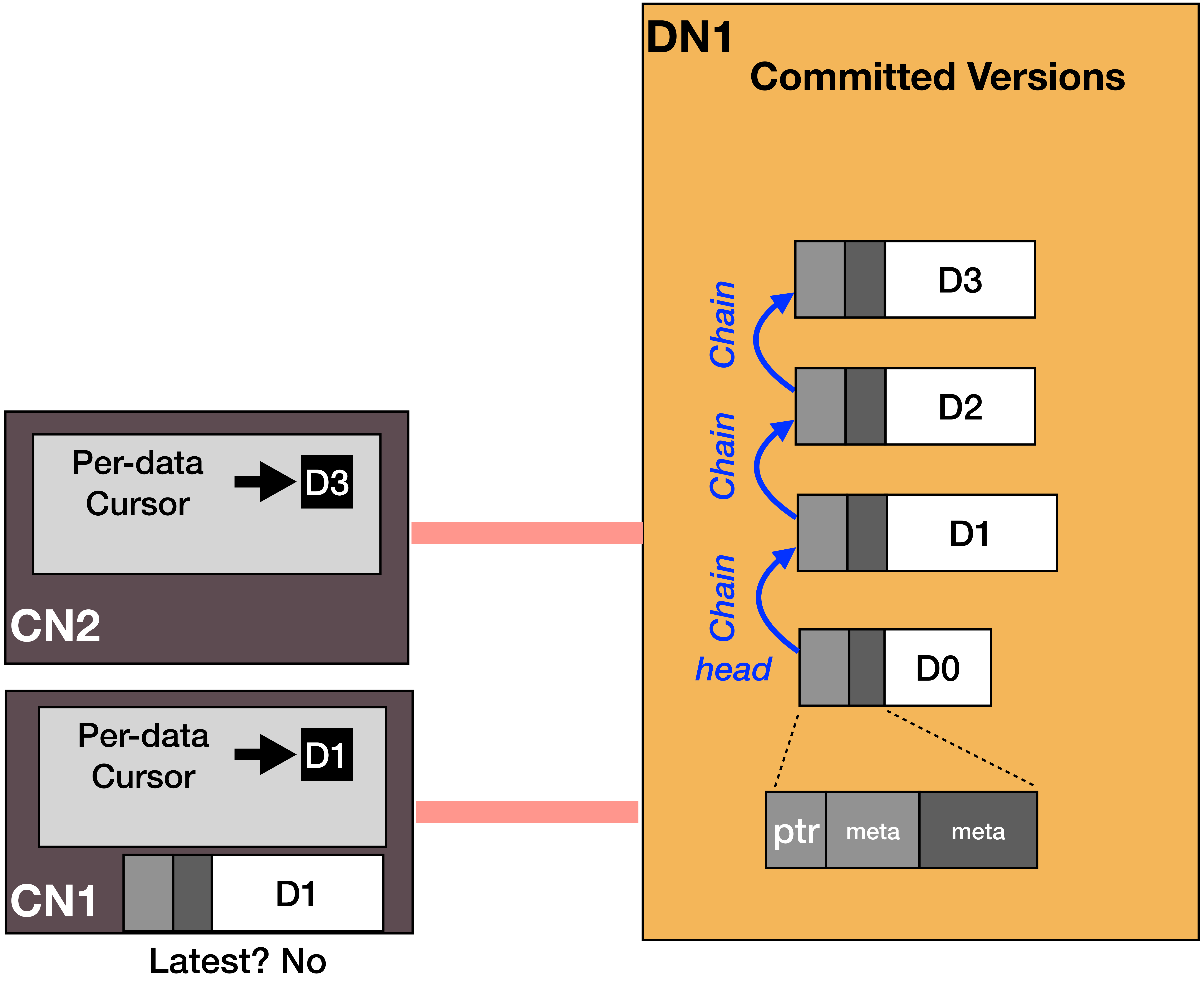
CN caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching cursor-pointed data
2. Walks the chain until found the latest



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at `DN`

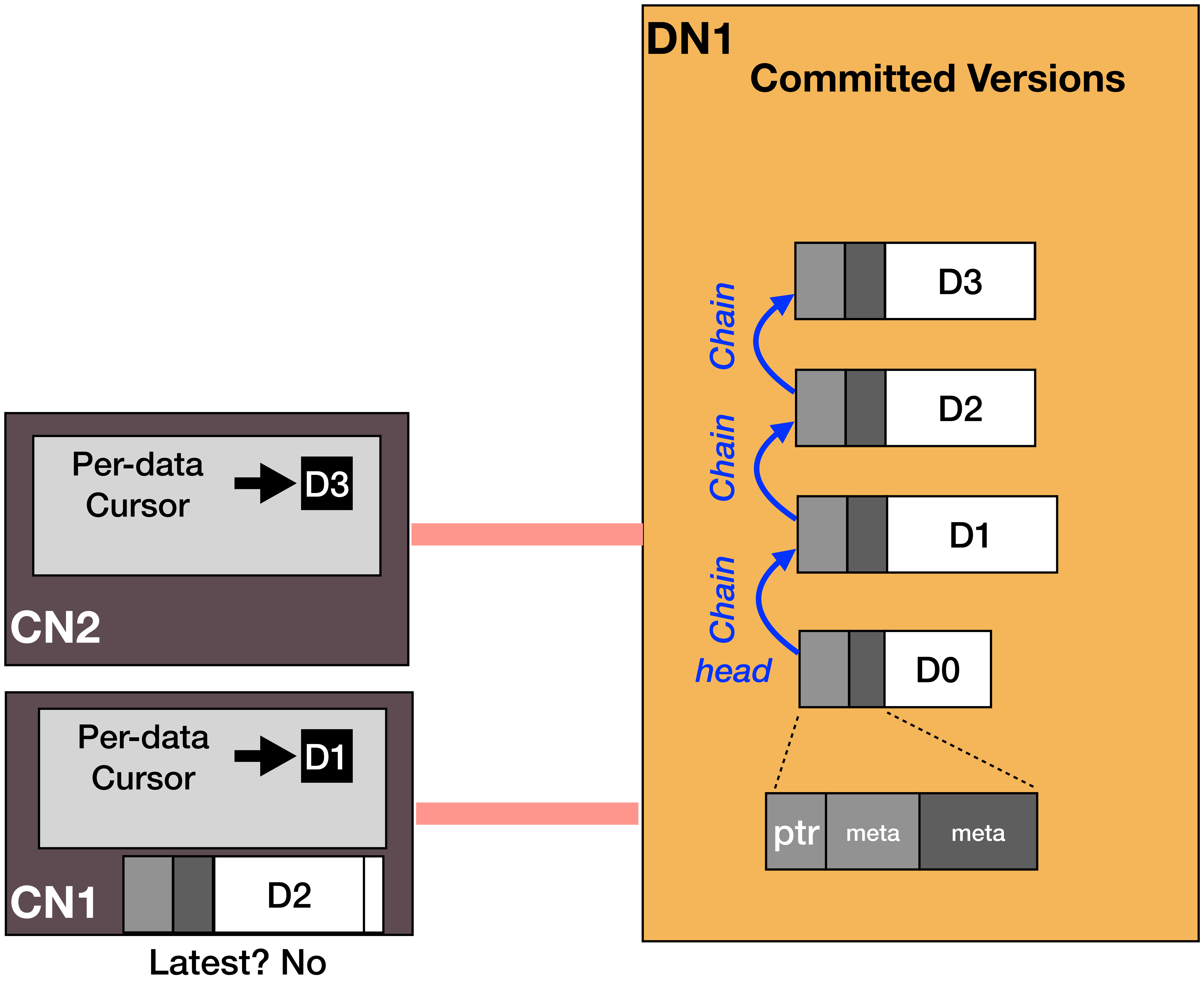
`CN` caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching cursor-pointed data
2. Walks the chain until found the latest



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

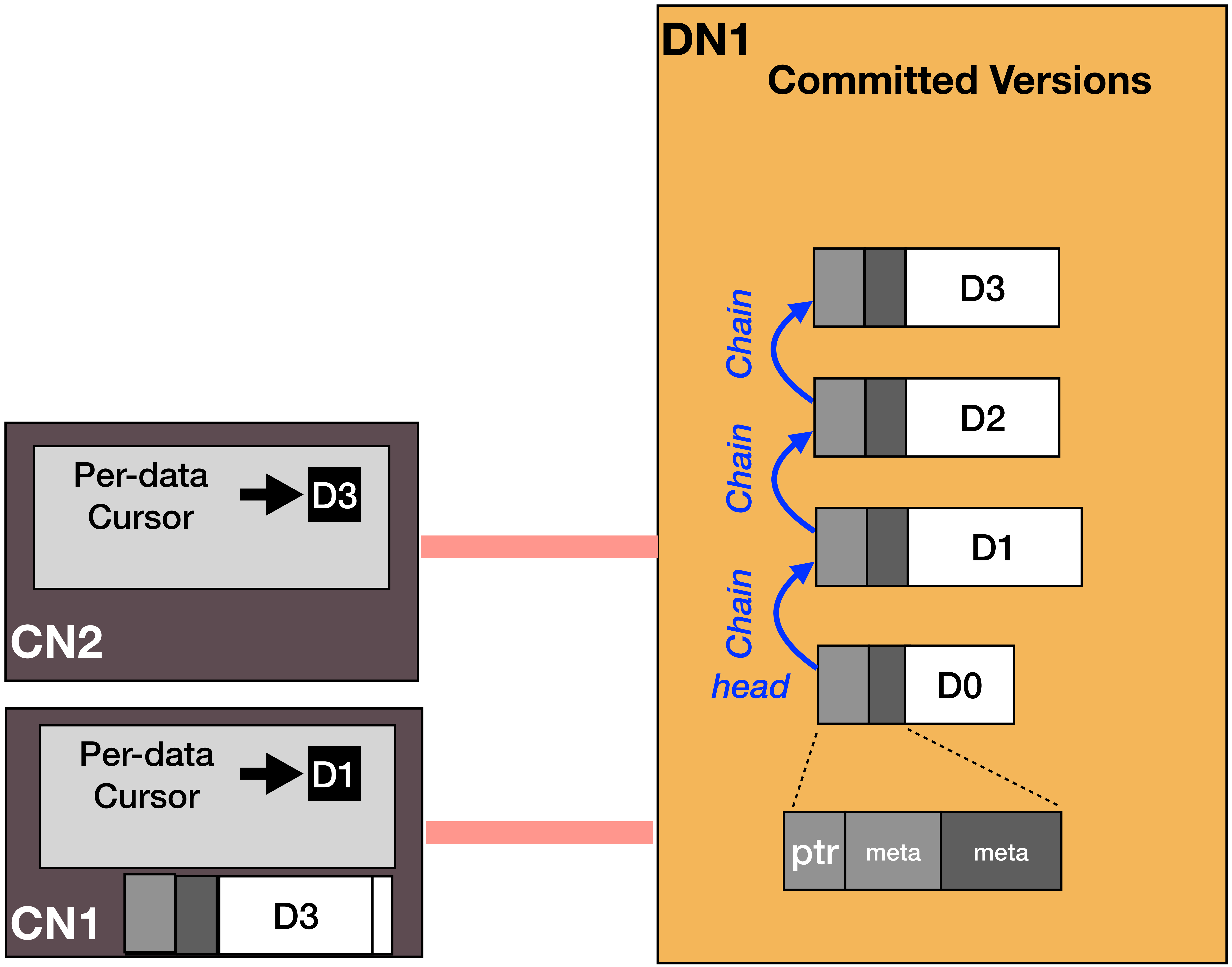
CN caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching cursor-pointed data
2. Walks the chain until found the latest



Latest? Yes

CN1's cursor points to D1, needs 3 chain read until each D3

Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

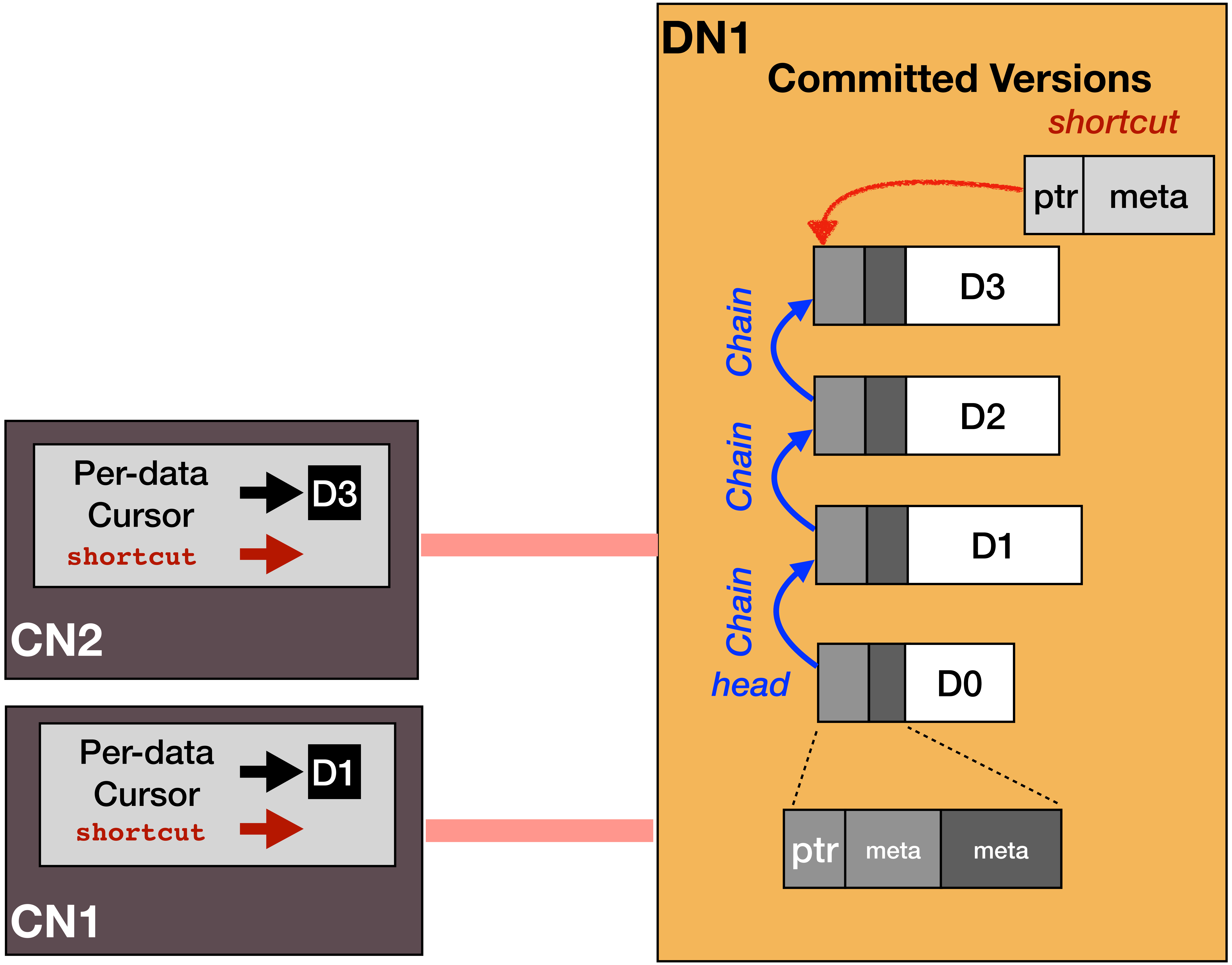
CN caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching cursor-pointed data
2. Walks the chain until found the latest



Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

CN caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching 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

Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

CN caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching cursor-pointed data
2. Walks the chain until found the latest

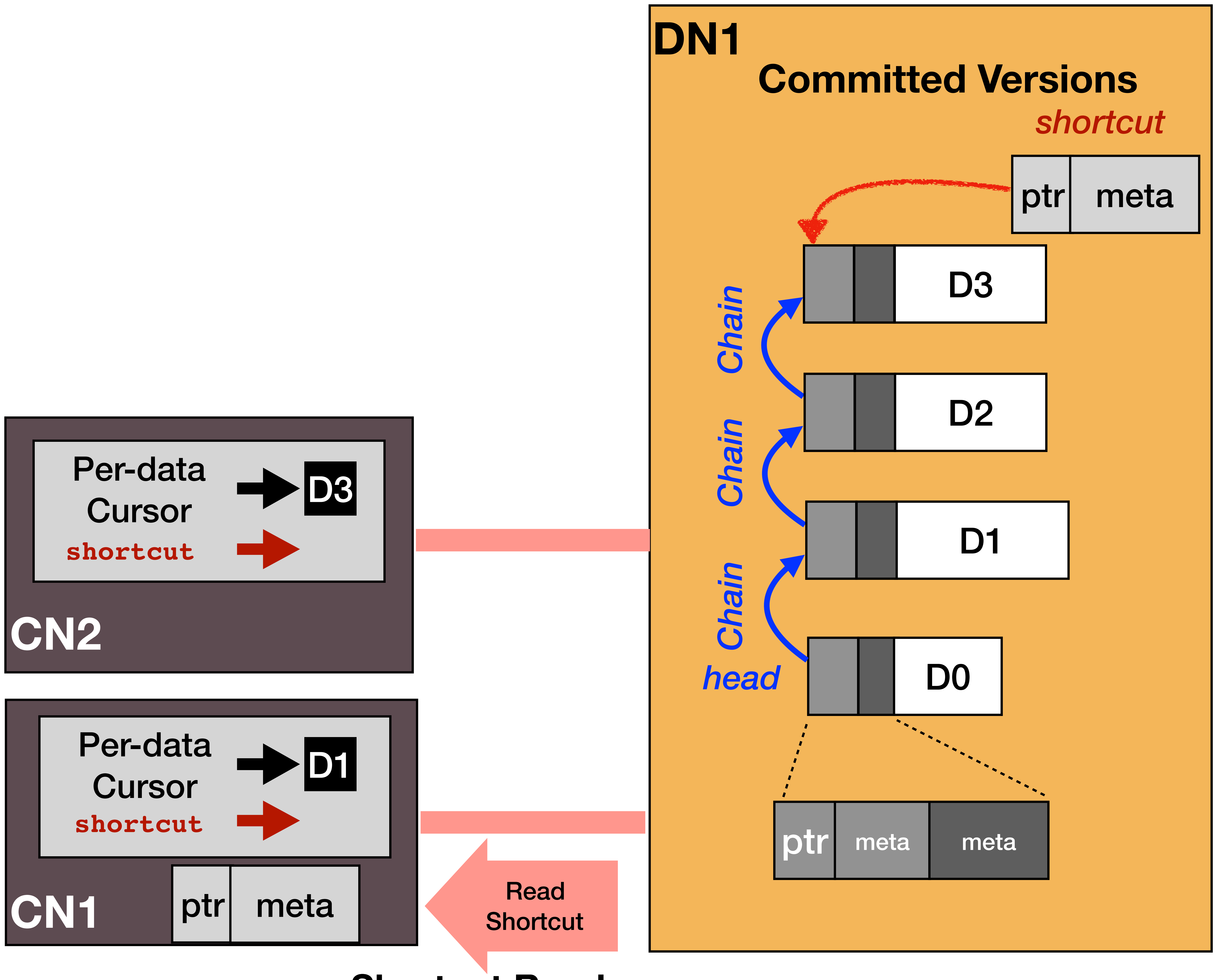
Optimization: Shortcut

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



Shortcut Read

1. Read shortcut
2. Read latest version

Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

CN caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching cursor-pointed data
2. Walks the chain until found the latest

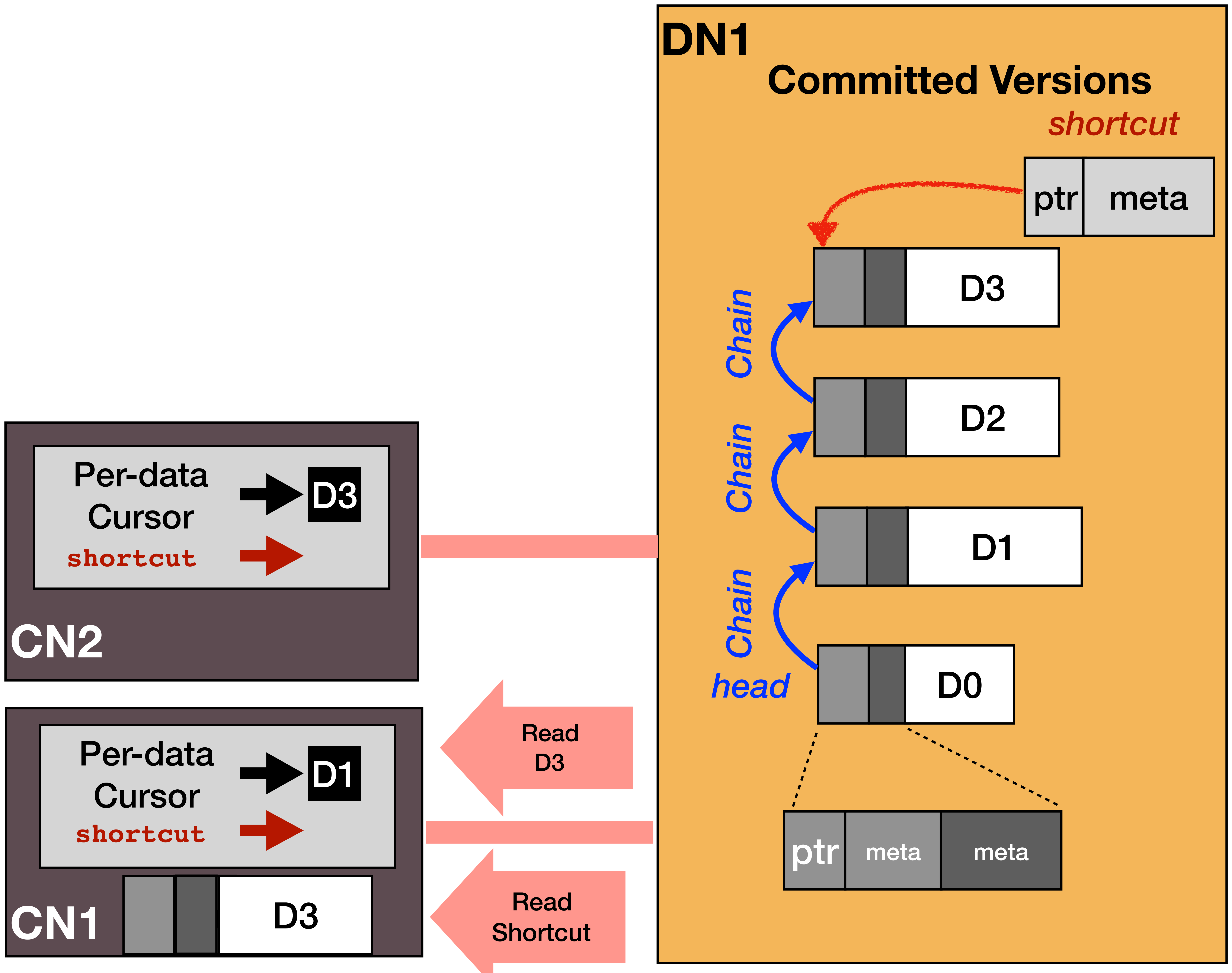
Optimization: Shortcut

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

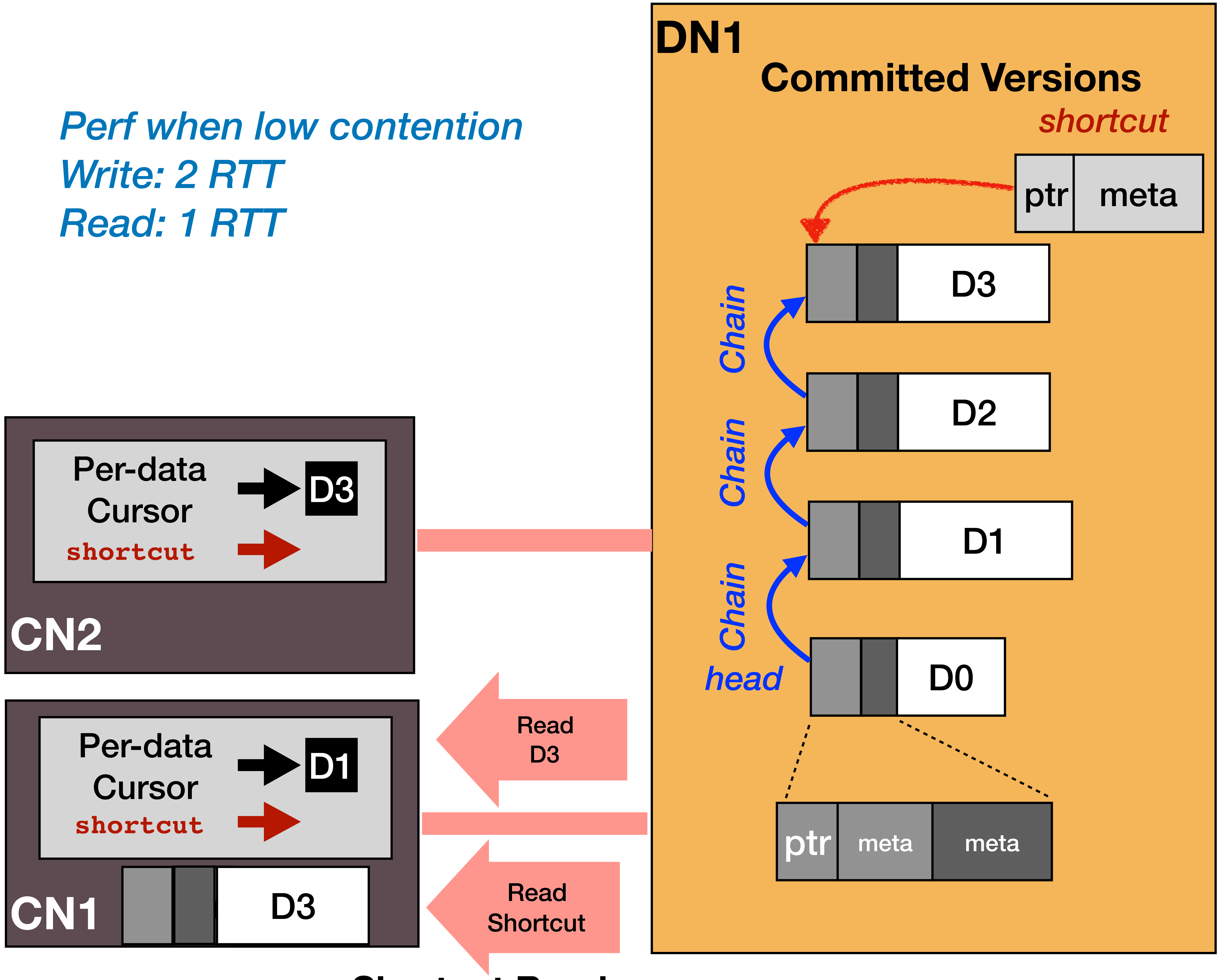
- Returns when the faster of 1 and 2 finish



Shortcut Read

1. Read shortcut
2. Read latest version

Perf when low contention
Write: 2 RTT
Read: 1 RTT



Shortcut Read

1. Read shortcut
2. Read latest version

Design: lock-free data structures

Our-of-place write (redo copy)

Chained redo copies at DN

CN caches a ***cursor*** points to a version

Write Flow

1. Write to a new location at DN
2. Chain the redo copy using ***c&s***
3. If 2. fails, update cursor and retry

Read Flow

1. Starts by fetching cursor-pointed data
2. Walks the chain until found the latest

Optimization: Shortcut

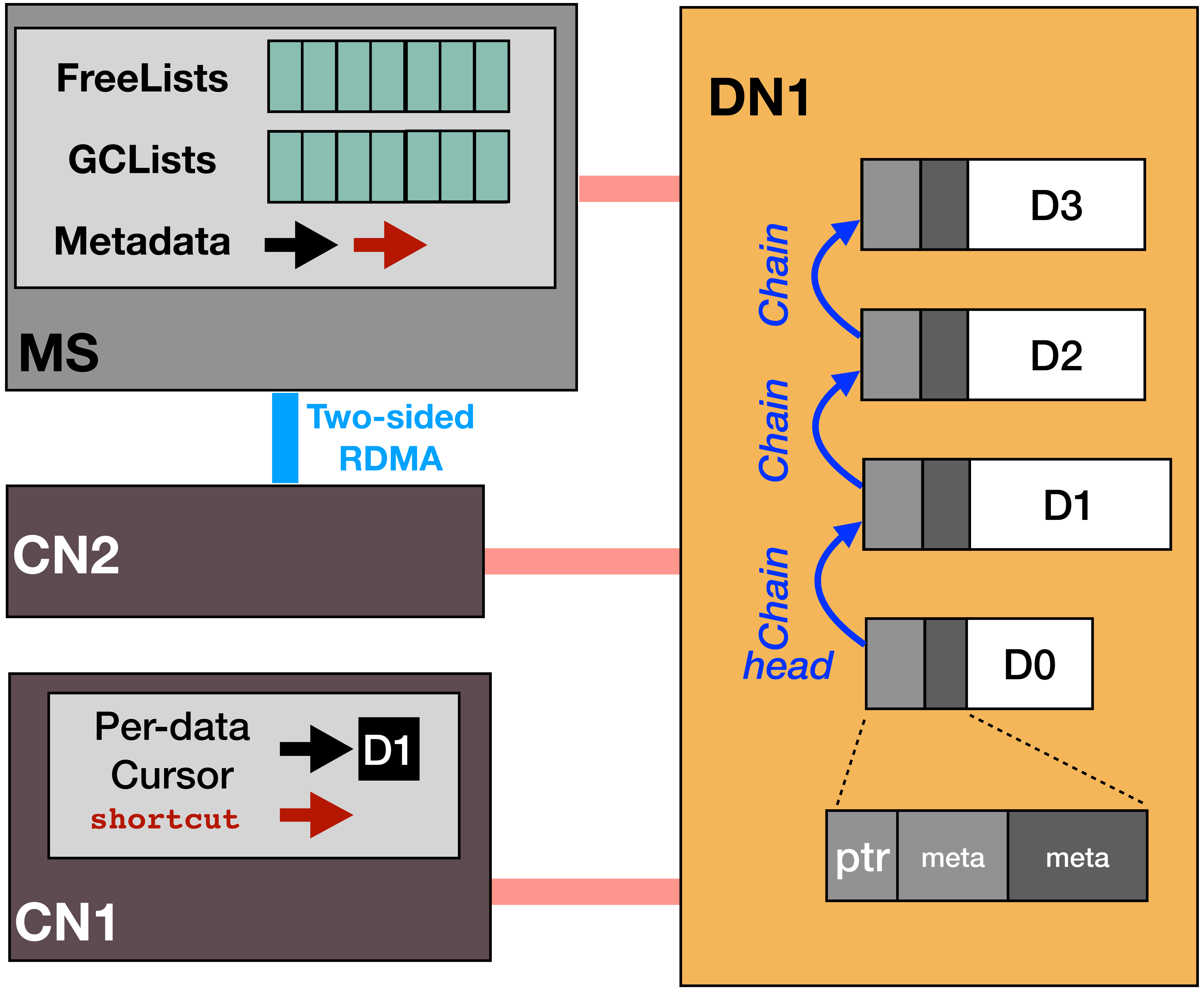
Uses a shortcut to avoid long chain walk

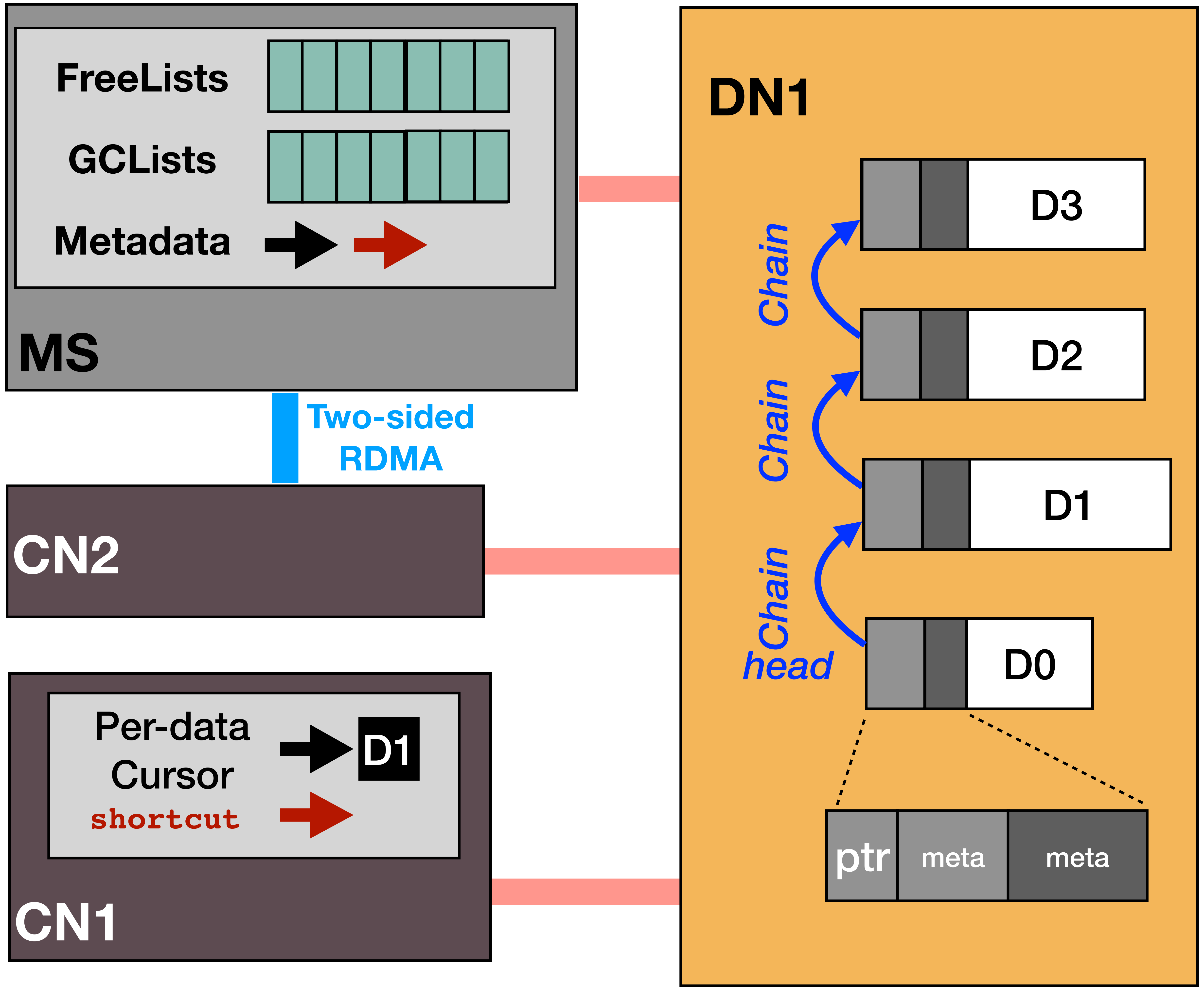
A shortcut at DN (mostly) points to the latest data

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

Metadata Server (MS)

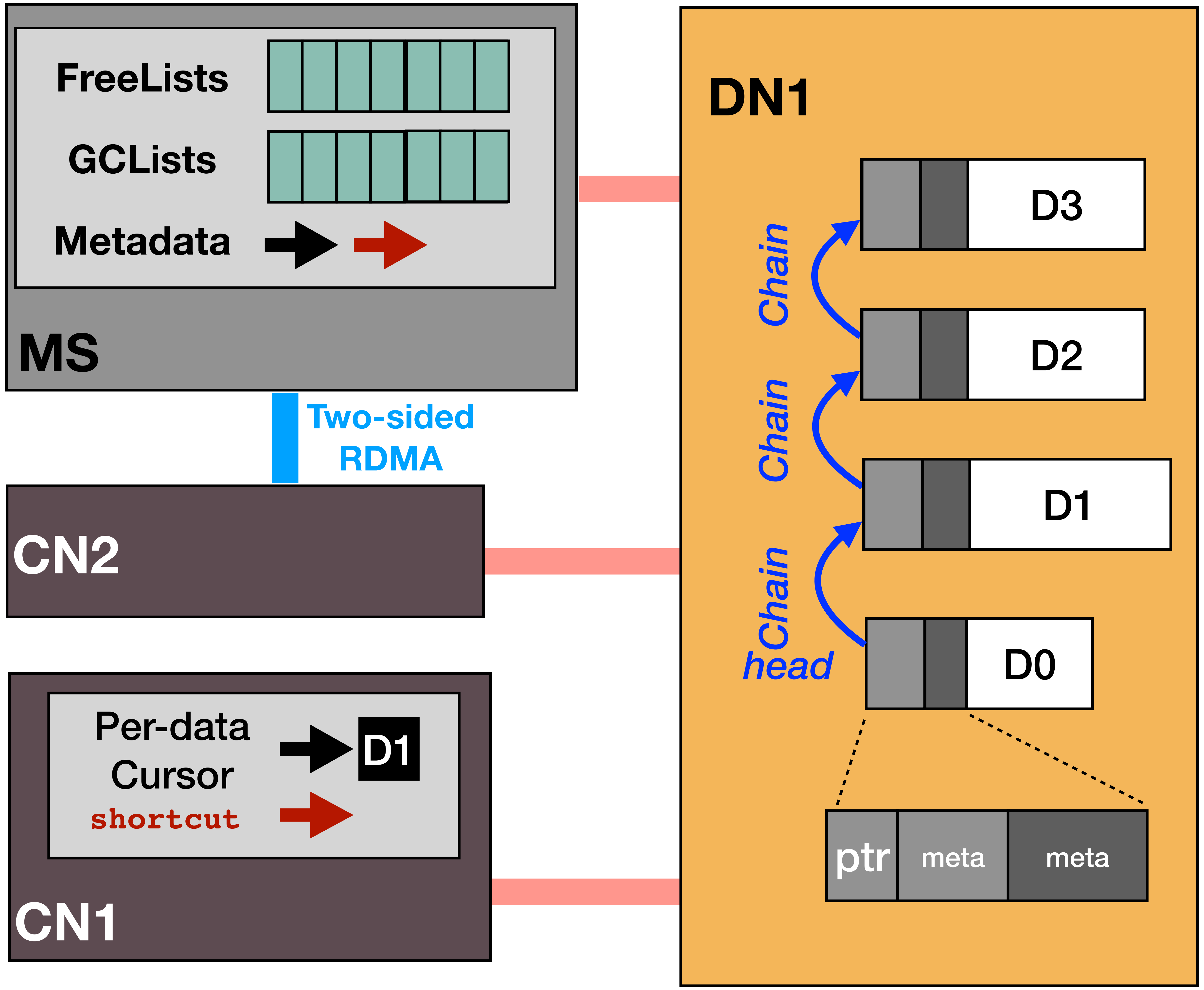




Metadata Server (MS)

Tasks

- Space management
- Garbage collection
- Global load balancing

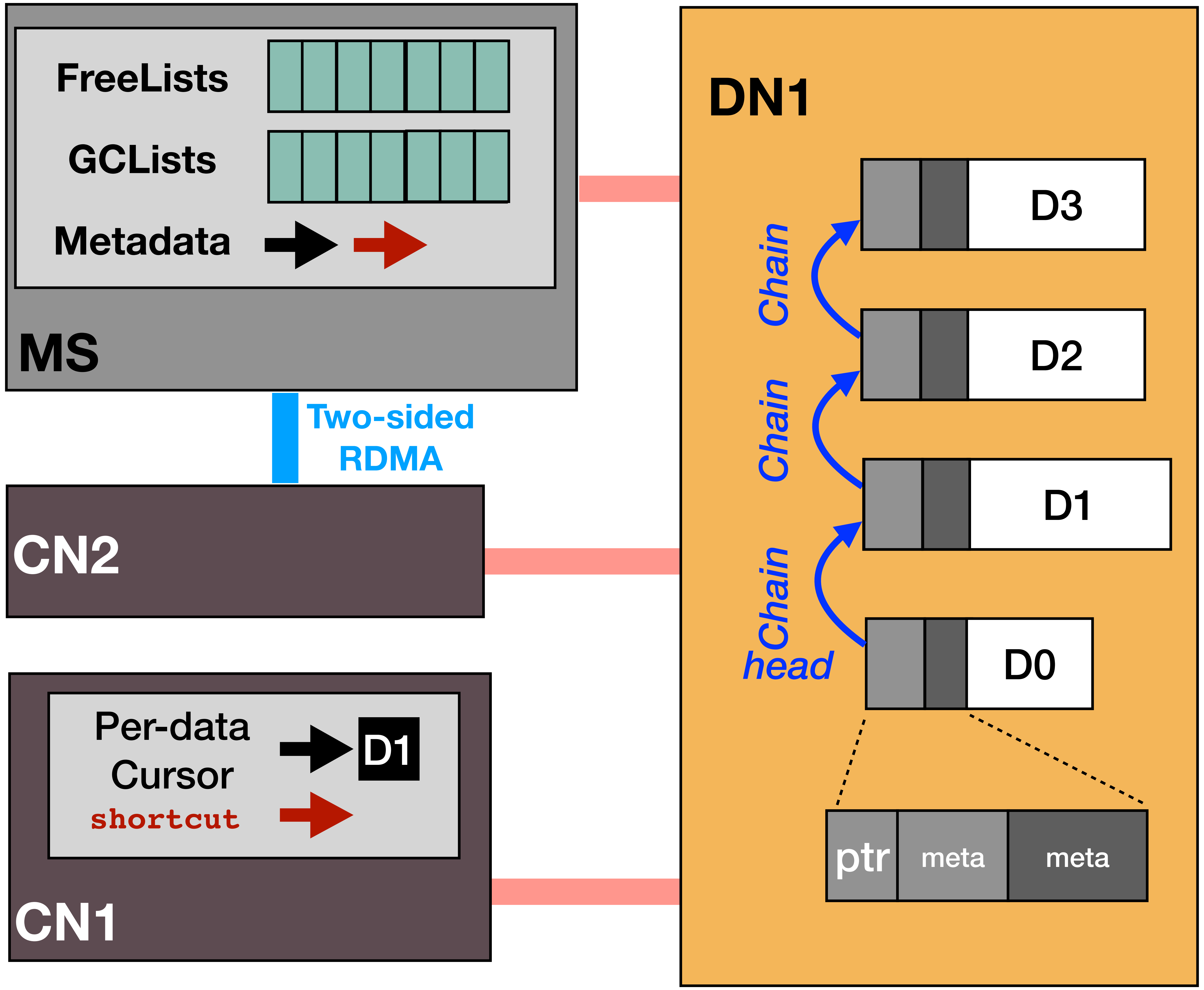


Metadata Server (MS)

Tasks

- Space management
- Garbage collection
- Global load balancing

Design principles



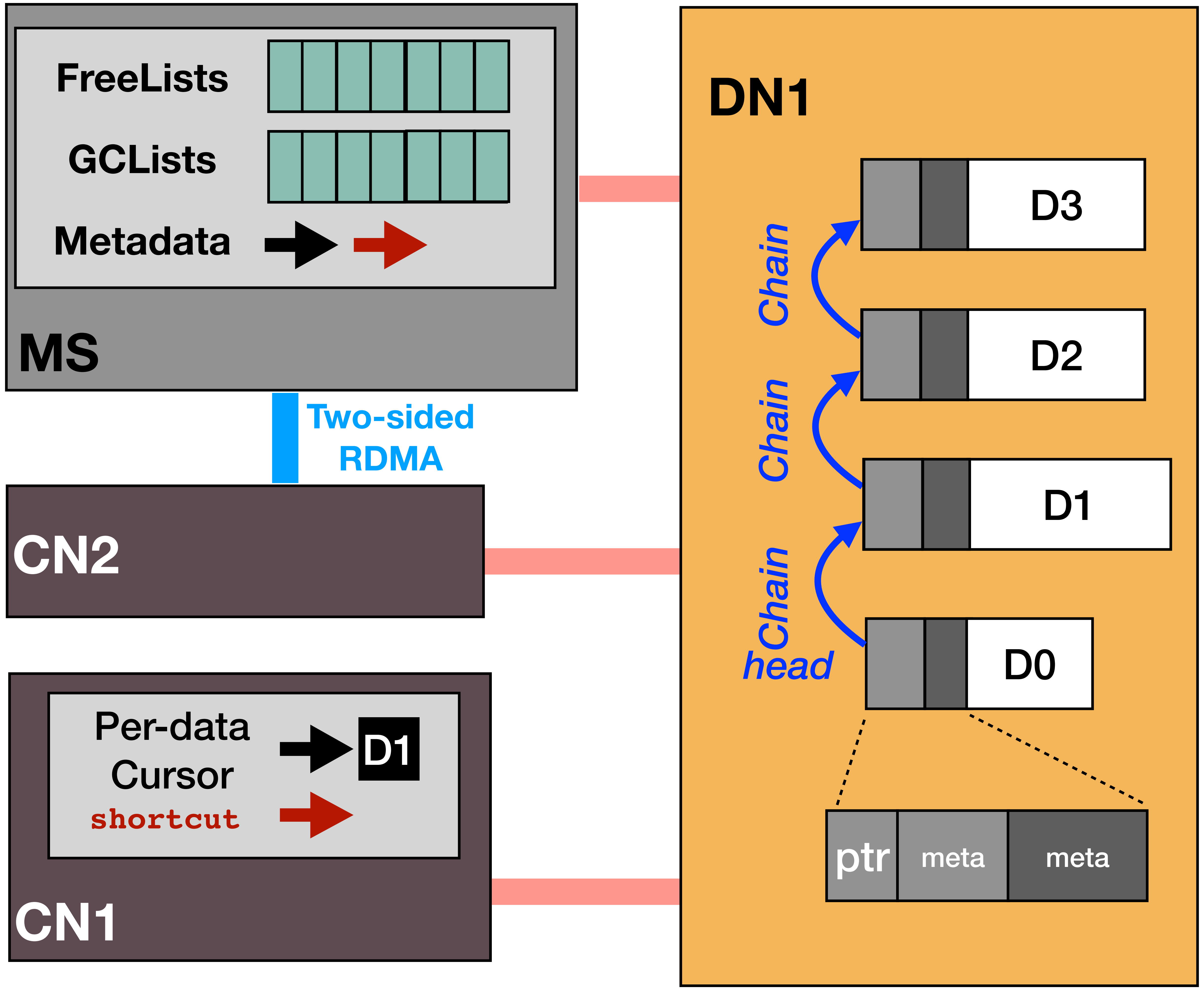
Metadata Server (MS)

Tasks

- Space management
- Garbage collection
- Global load balancing

Design principles

- All metadata ops are off critical path



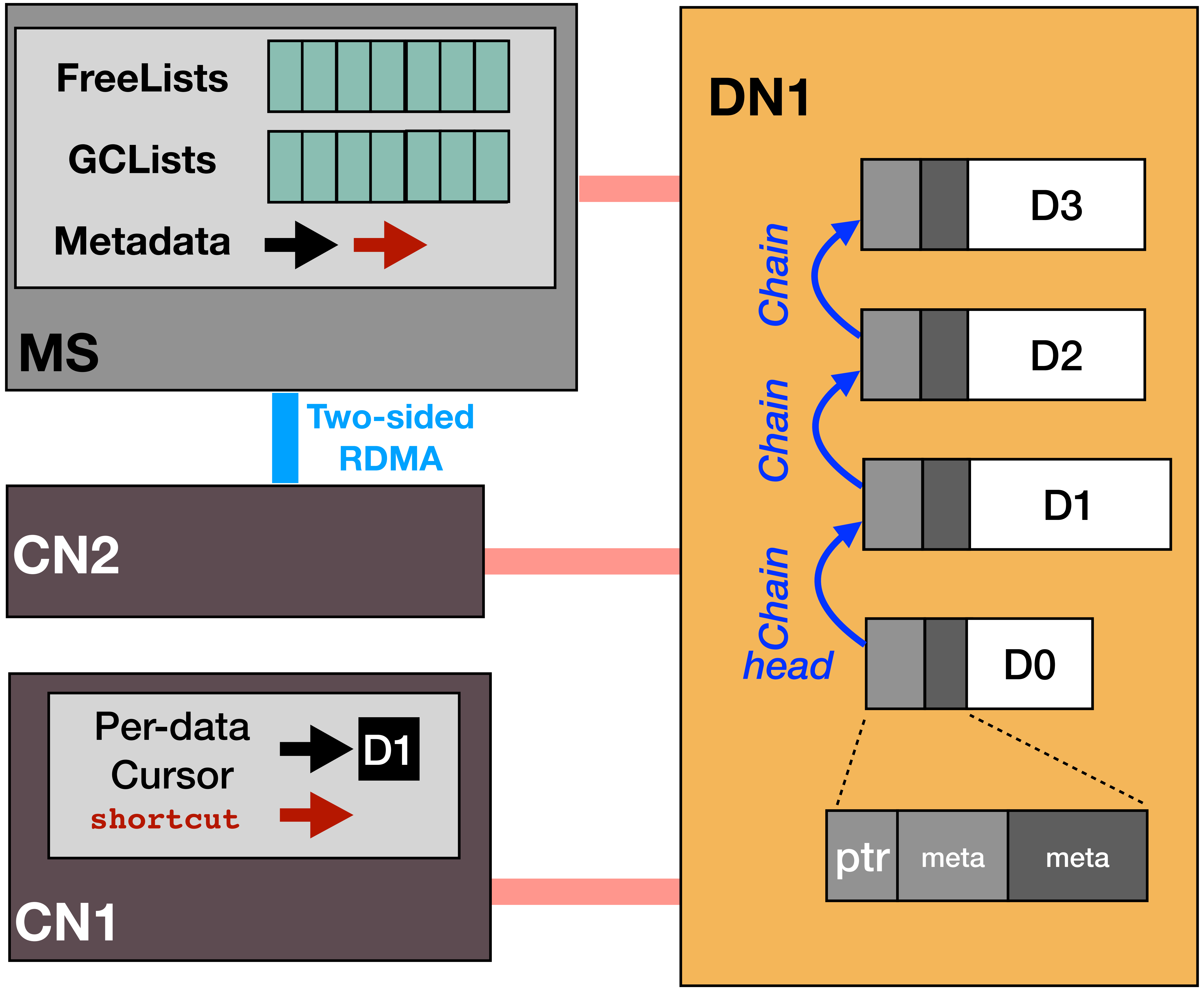
Metadata Server (MS)

Tasks

- Space management
- Garbage collection
- Global load balancing

Design principles

- All metadata ops are off critical path
- MS manages DN space without accessing DNs



Metadata Server (MS)

Tasks

- Space management
- Garbage collection
- Global load balancing

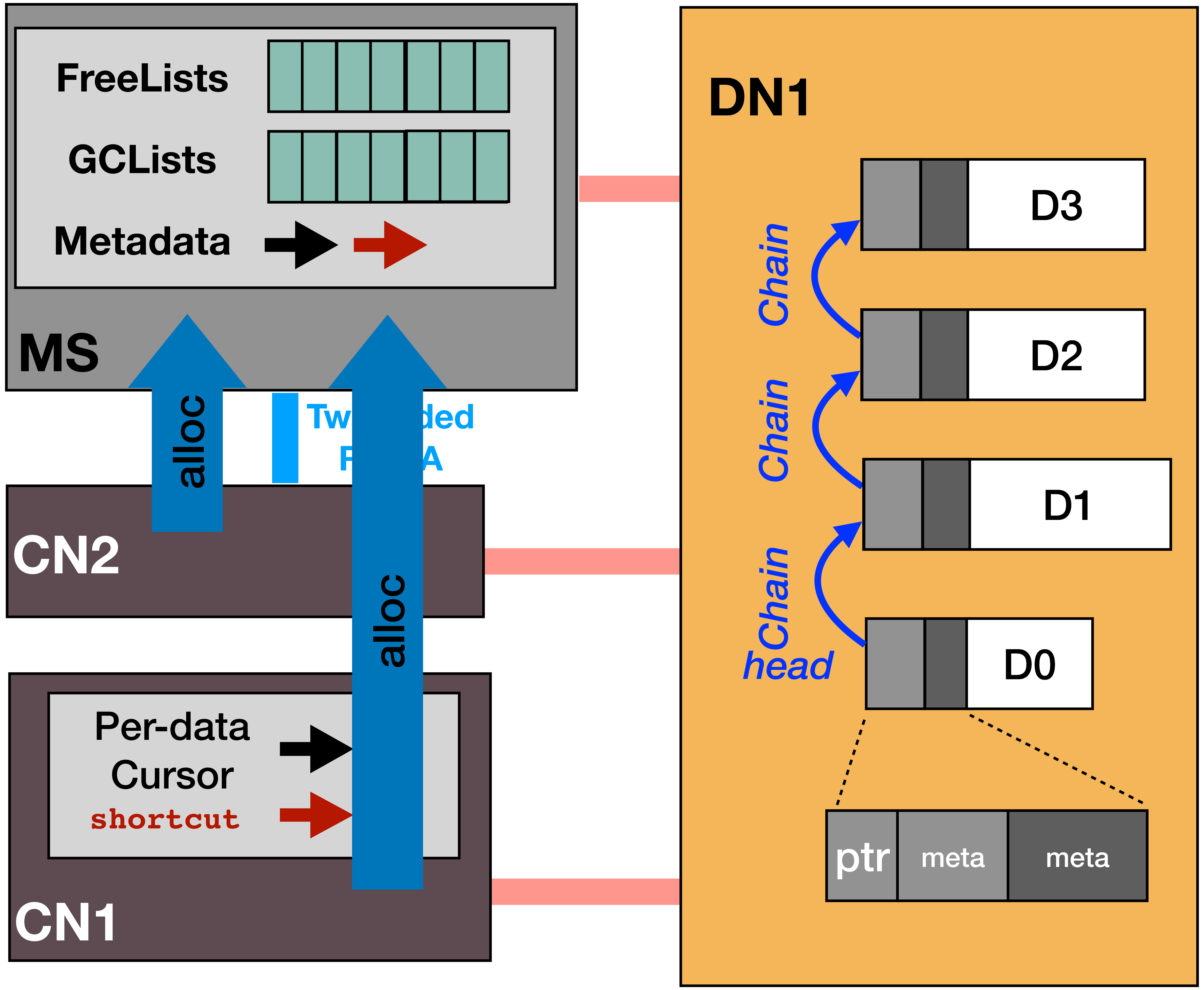
Design principles

- All metadata ops are off critical path
- MS manages DN space without accessing DNs

Techniques

- Batched allocation
- Async garbage collection
- No cache invalidation

Metadata Server (MS)



Alloc Flow

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

Tasks

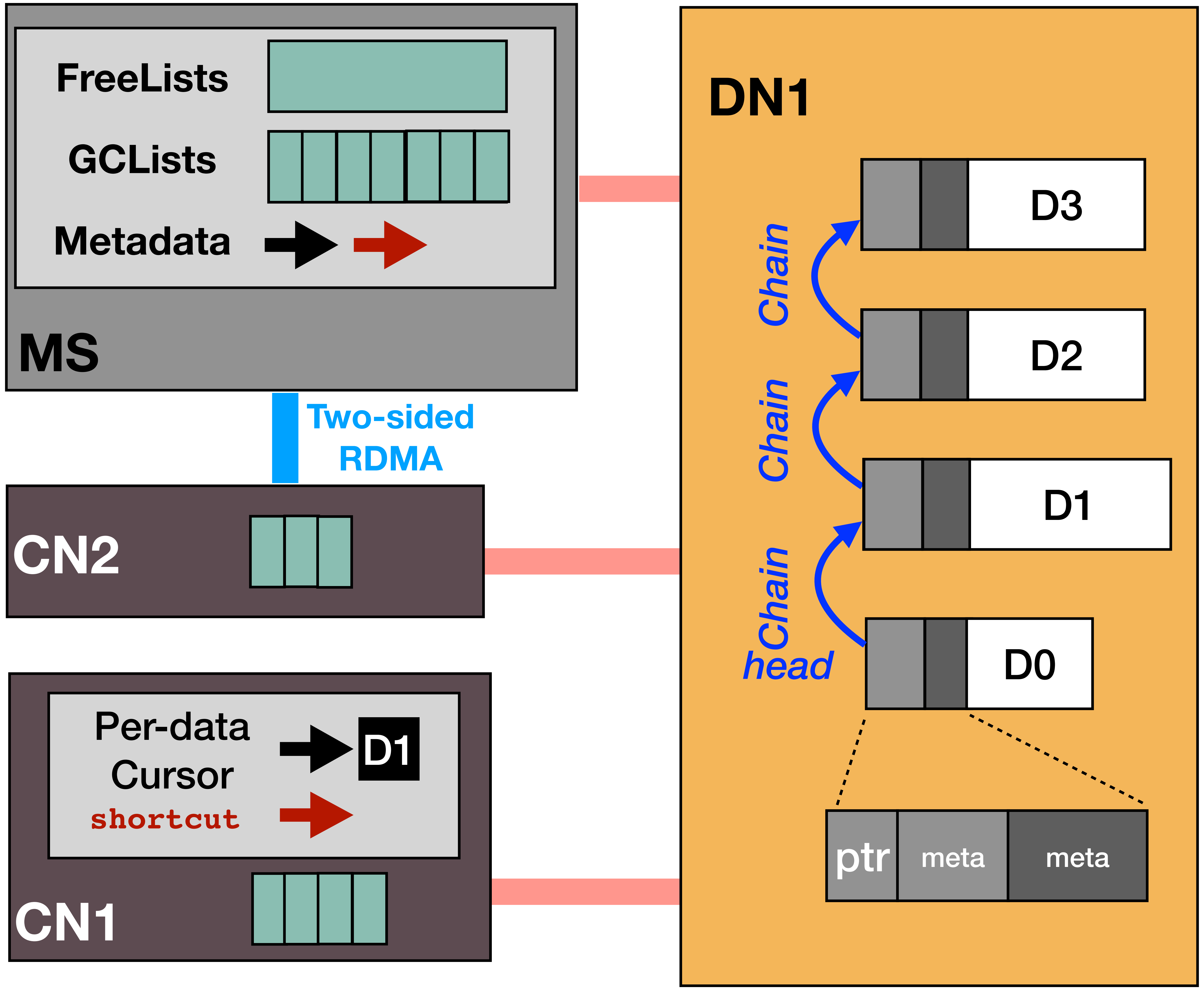
- Space management
- Garbage collection
- Global load balancing

Design principles

- All metadata ops are off critical path
- MS manages DN space without accessing DNs

Techniques

- Batched allocation
- Async garbage collection
- No cache invalidation



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)

Tasks

- Space management
- Garbage collection
- Global load balancing

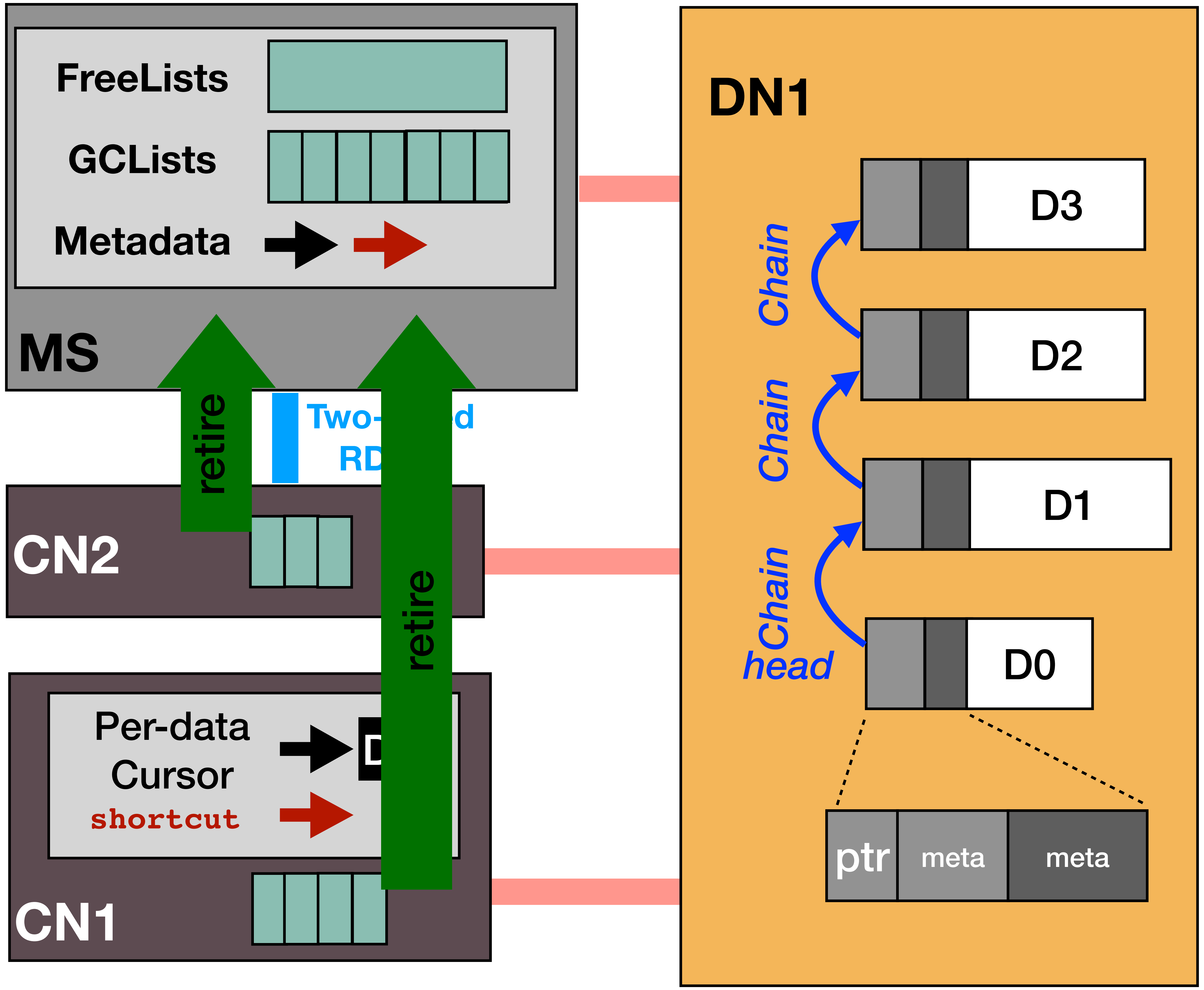
Design principles

- All metadata ops are off critical path
- MS manages DN space without accessing DNs

Techniques

- Batched allocation
- Async garbage collection
- No cache invalidation

Metadata Server (MS)



Tasks

- Space management
- Garbage collection
- Global load balancing

Design principles

- All metadata ops are off critical path
- MS manages DN space without accessing DNs

Techniques

- Batched allocation
- Async garbage collection
- No cache invalidation

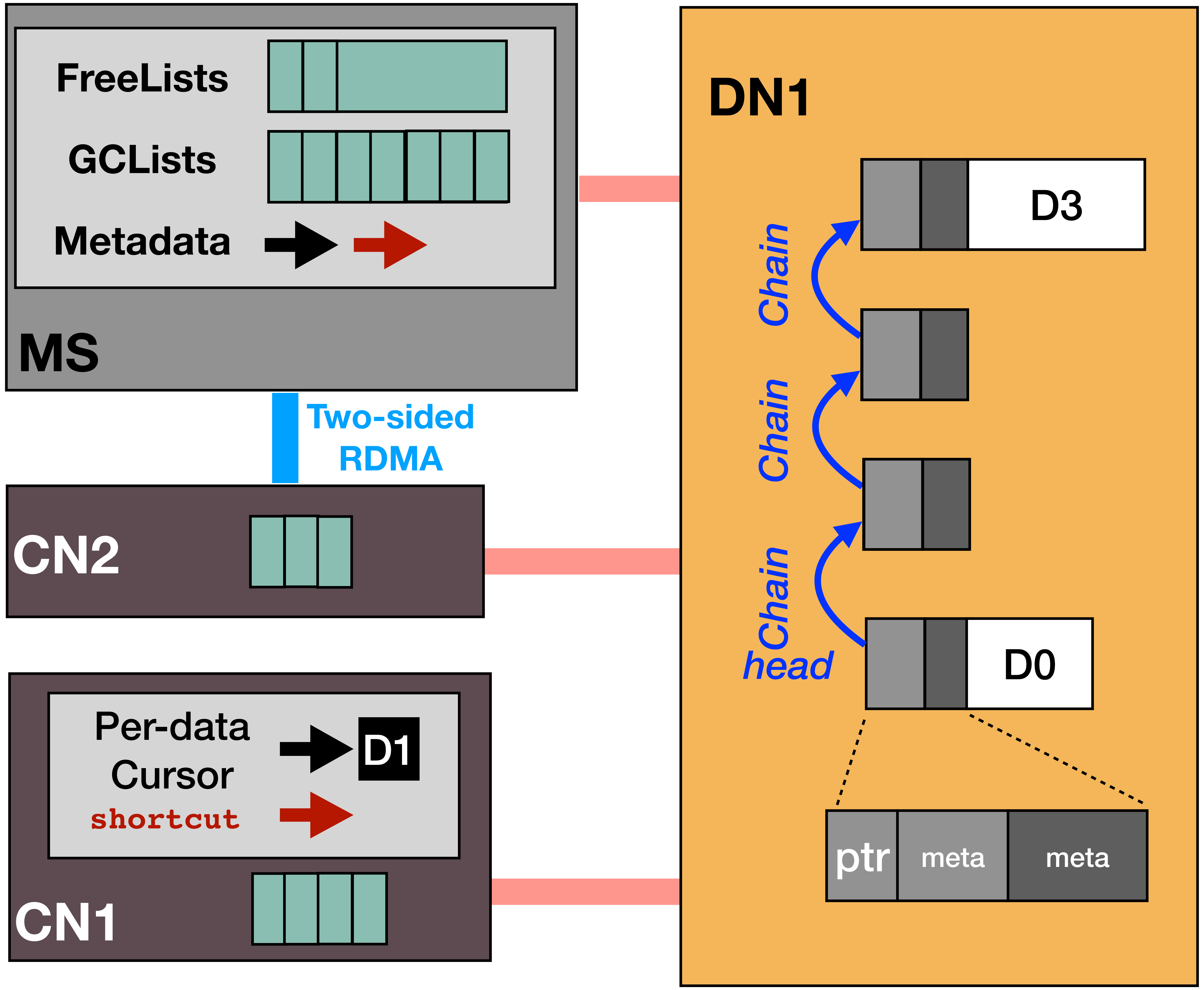
Alloc Flow

- 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)



Tasks

- Space management
- Garbage collection
- Global load balancing

Design principles

- All metadata ops are off critical path
- MS manages DN space without accessing DNs

Techniques

- Batched allocation
- Async garbage collection
- No cache invalidation

Alloc Flow

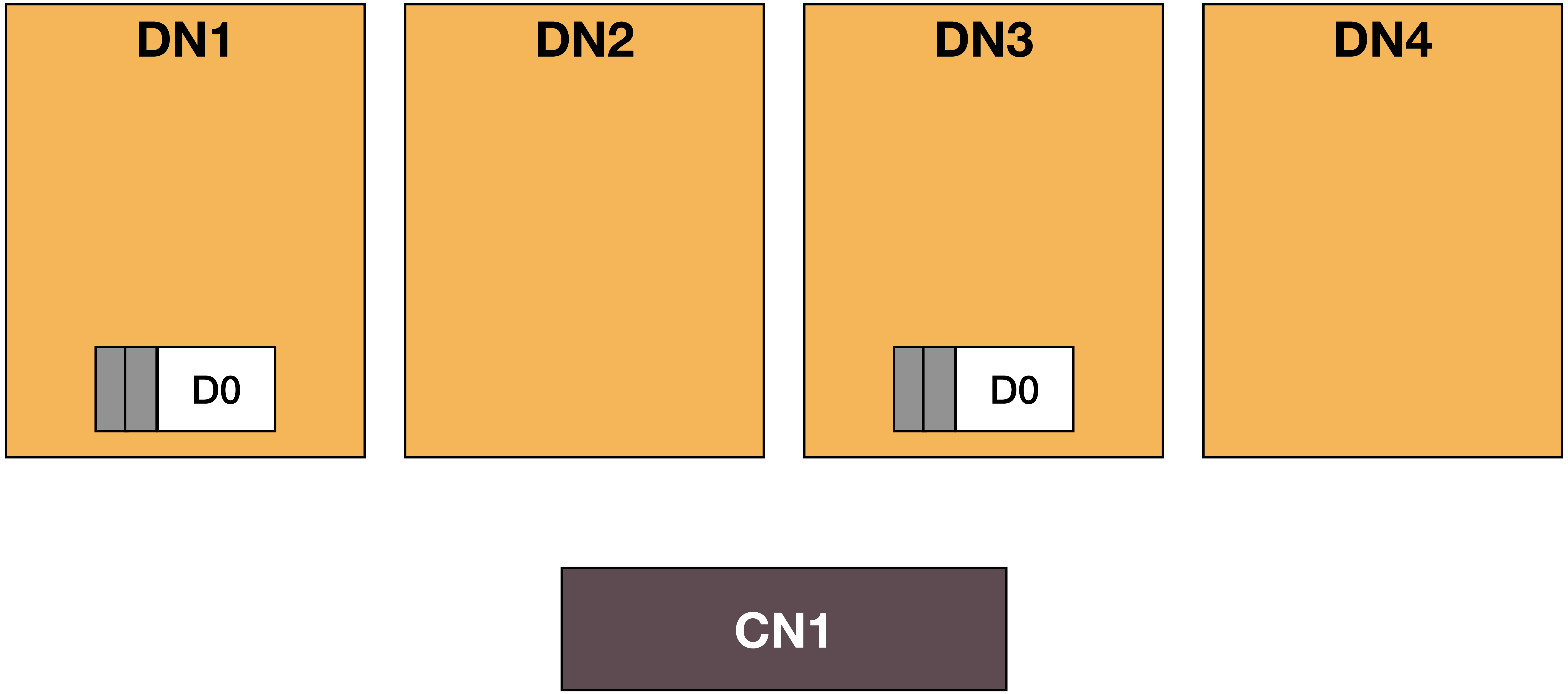
- CN asks MS for a bunch of free buffers at a time
- MS assigns spaces from FreeLists (with load balancing consideration)

GC Flow

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

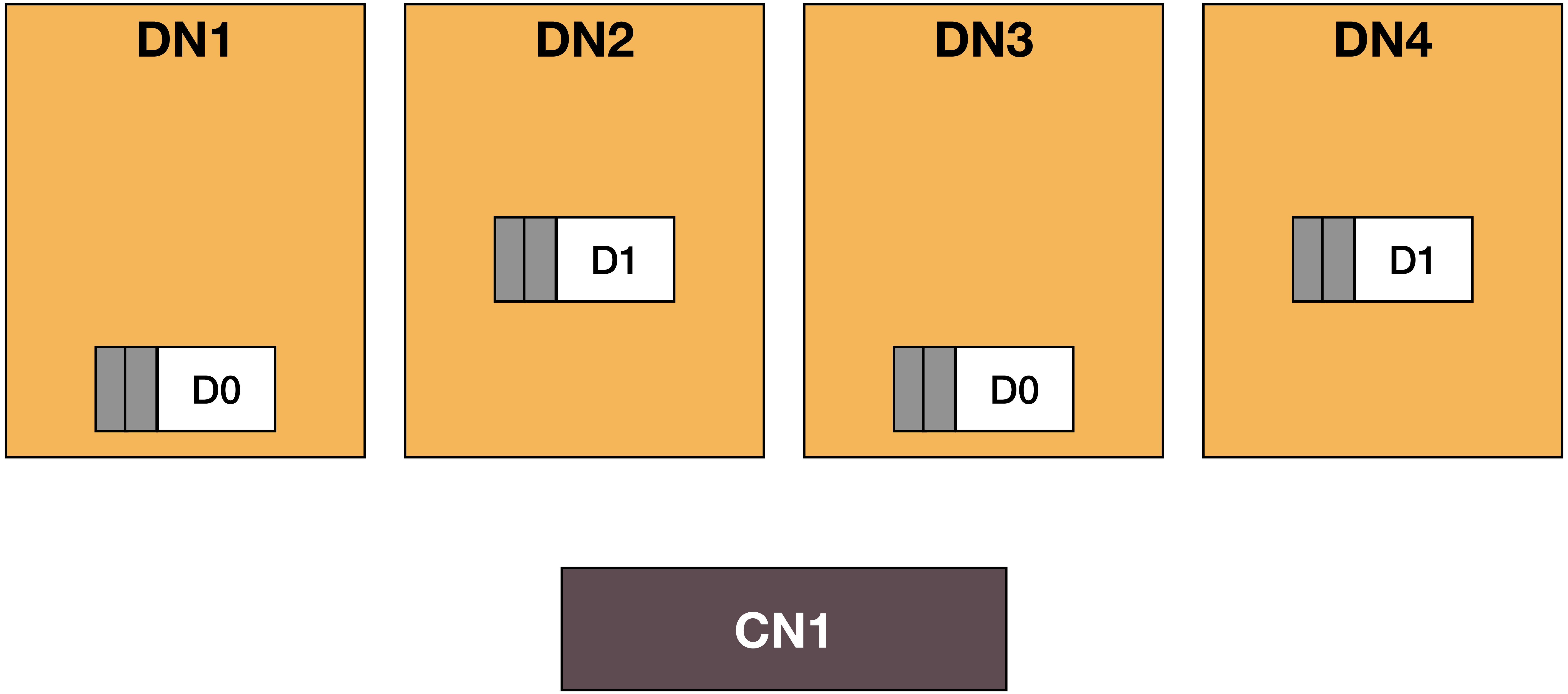
Clover Replication

- Data Redundancy
  - User-defined replication degree
  - a novel atomic chaining replication
  - link a version to all the replicas of next version
- Clover can handle both DN and MS failures



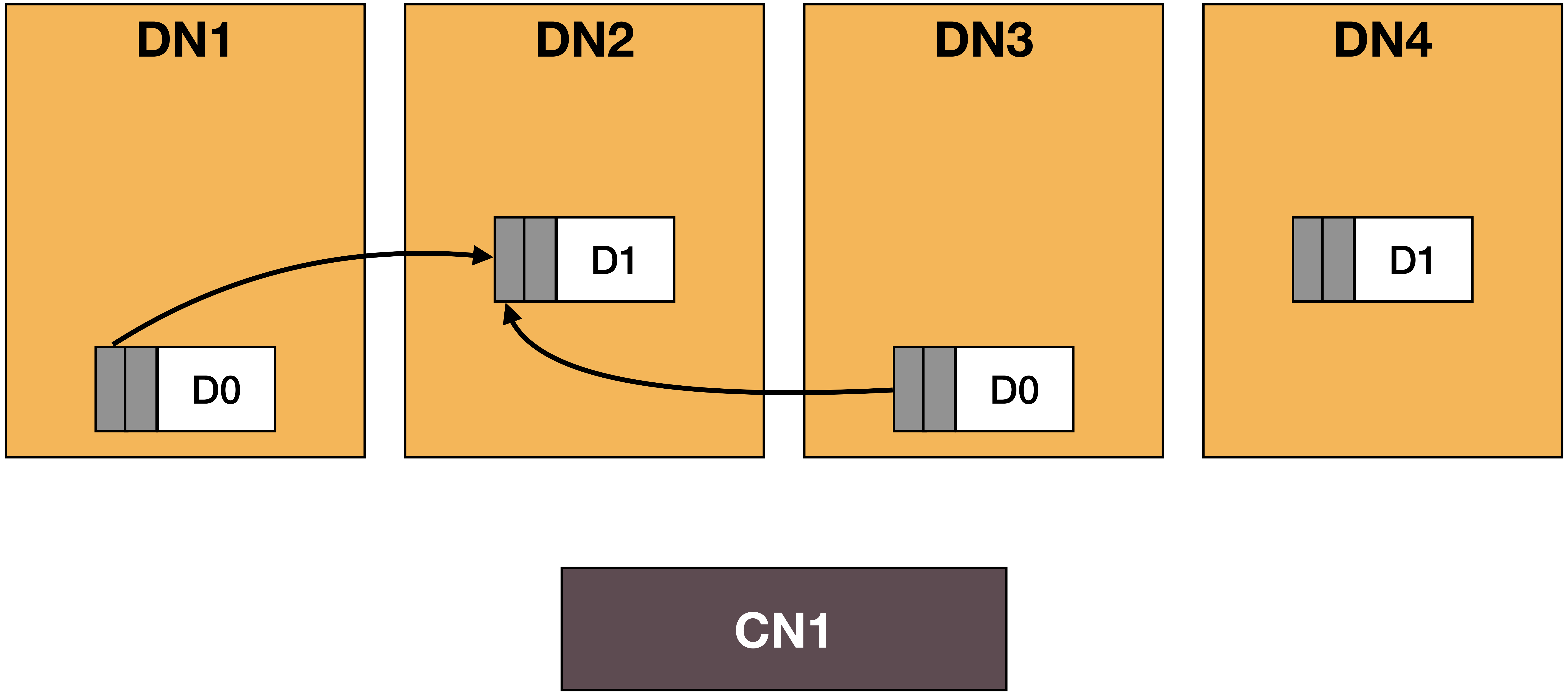
Clover Replication

- Data Redundancy
  - User-defined replication degree
  - a novel atomic chaining replication
  - link a version to all the replicas of next version
- Clover can handle both DN and MS failures



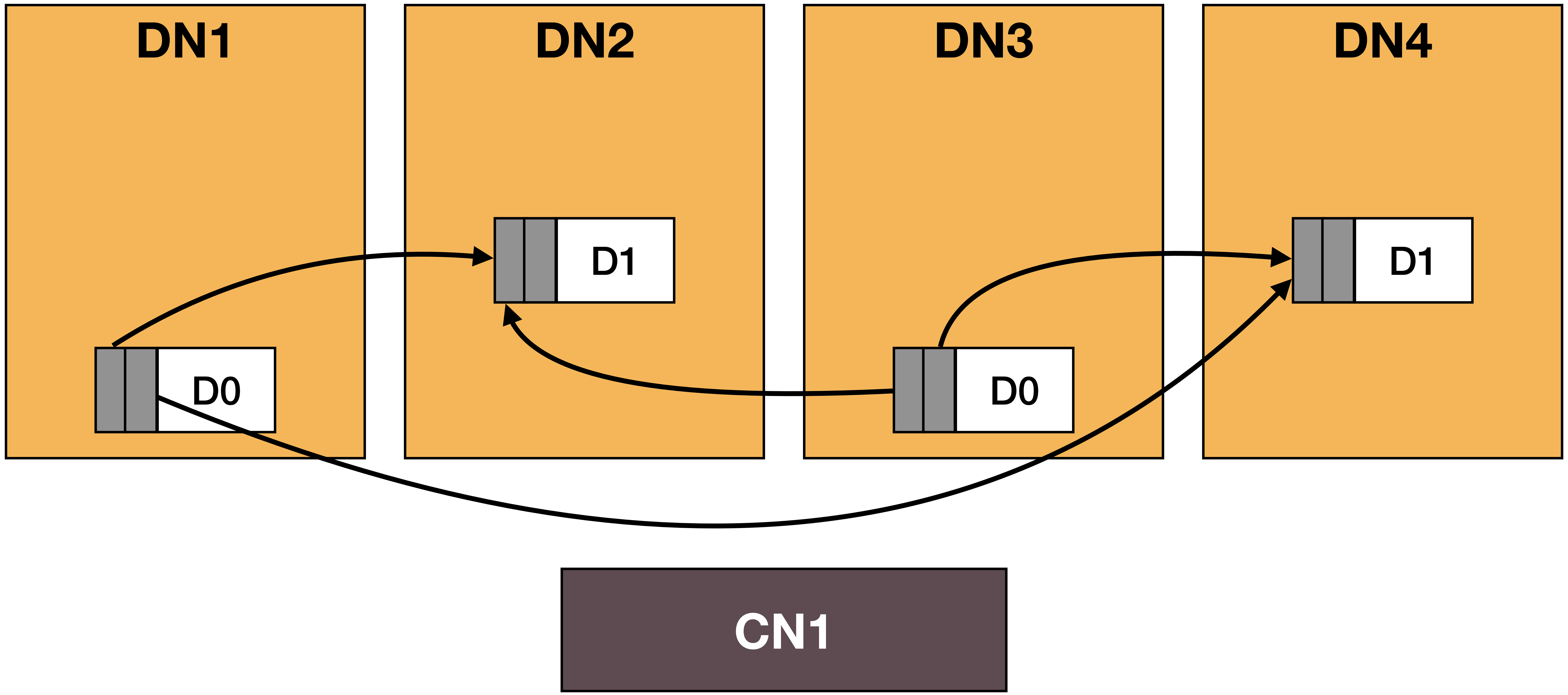
Clover Replication

- Data Redundancy
  - User-defined replication degree
  - a novel atomic chaining replication
  - link a version to all the replicas of next version
- Clover can handle both DN and MS failures



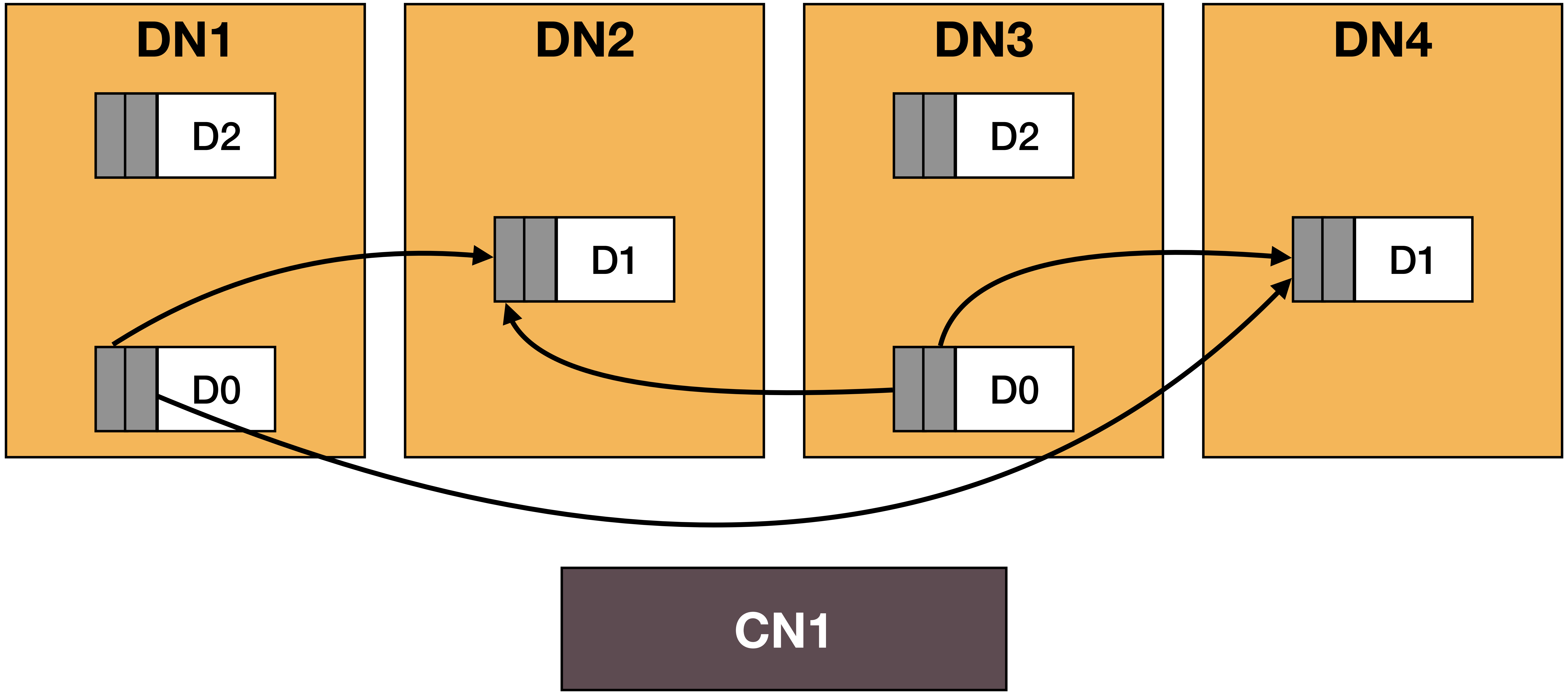
Clover Replication

- Data Redundancy
  - User-defined replication degree
  - a novel atomic chaining replication
  - link a version to all the replicas of next version
- Clover can handle both DN and MS failures



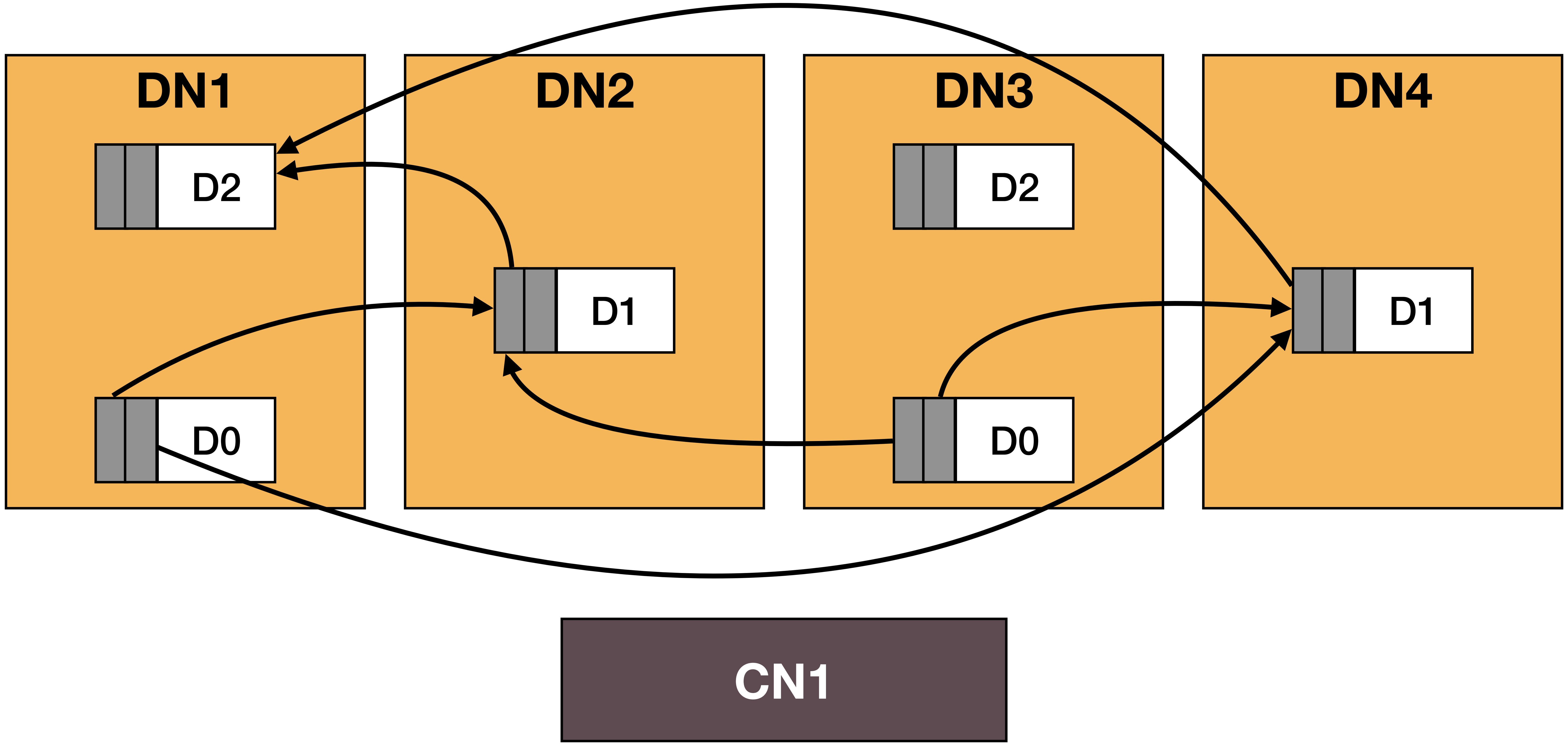
Clover Replication

- Data Redundancy
  - User-defined replication degree
  - a novel atomic chaining replication
  - link a version to all the replicas of next version
- Clover can handle both DN and MS failures



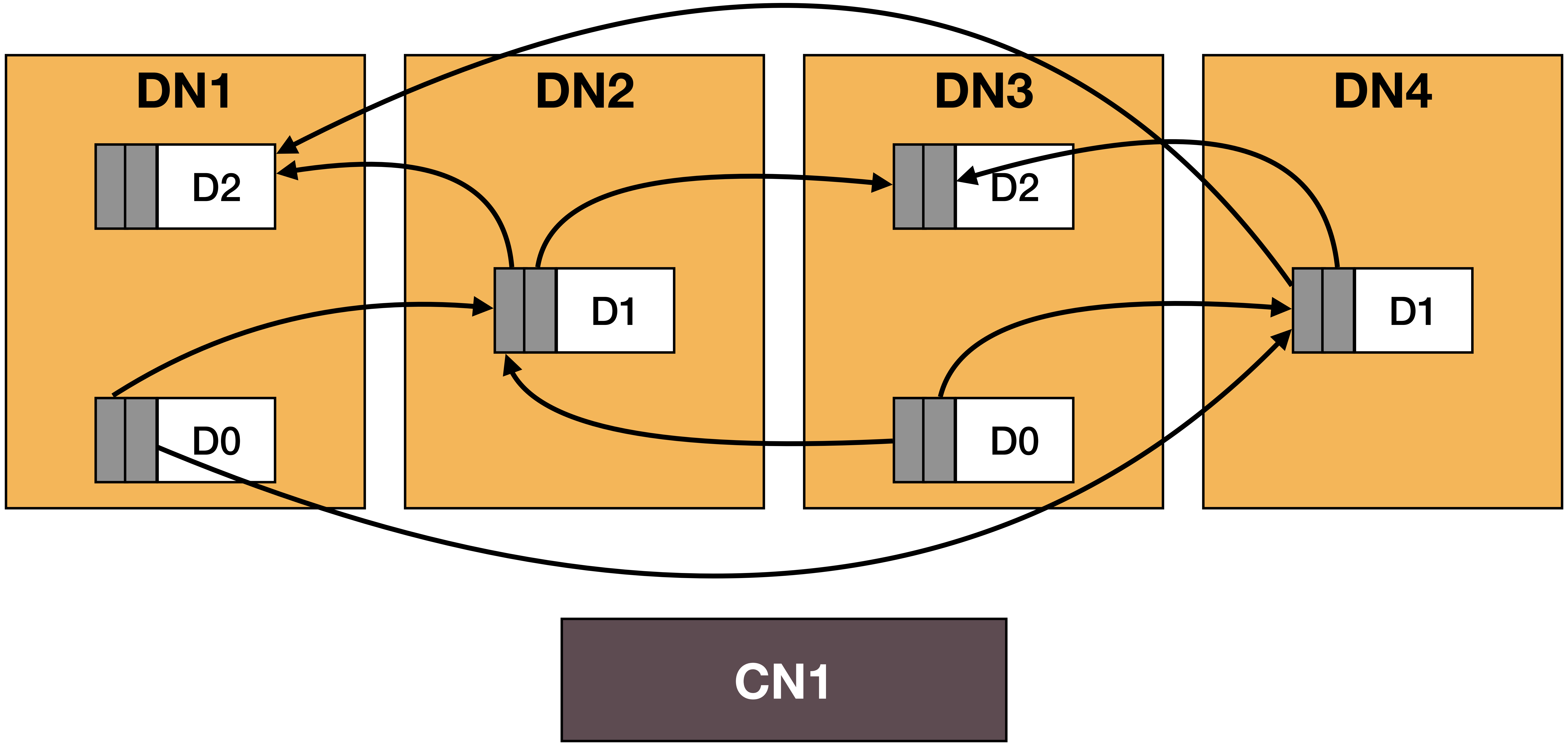
Clover Replication

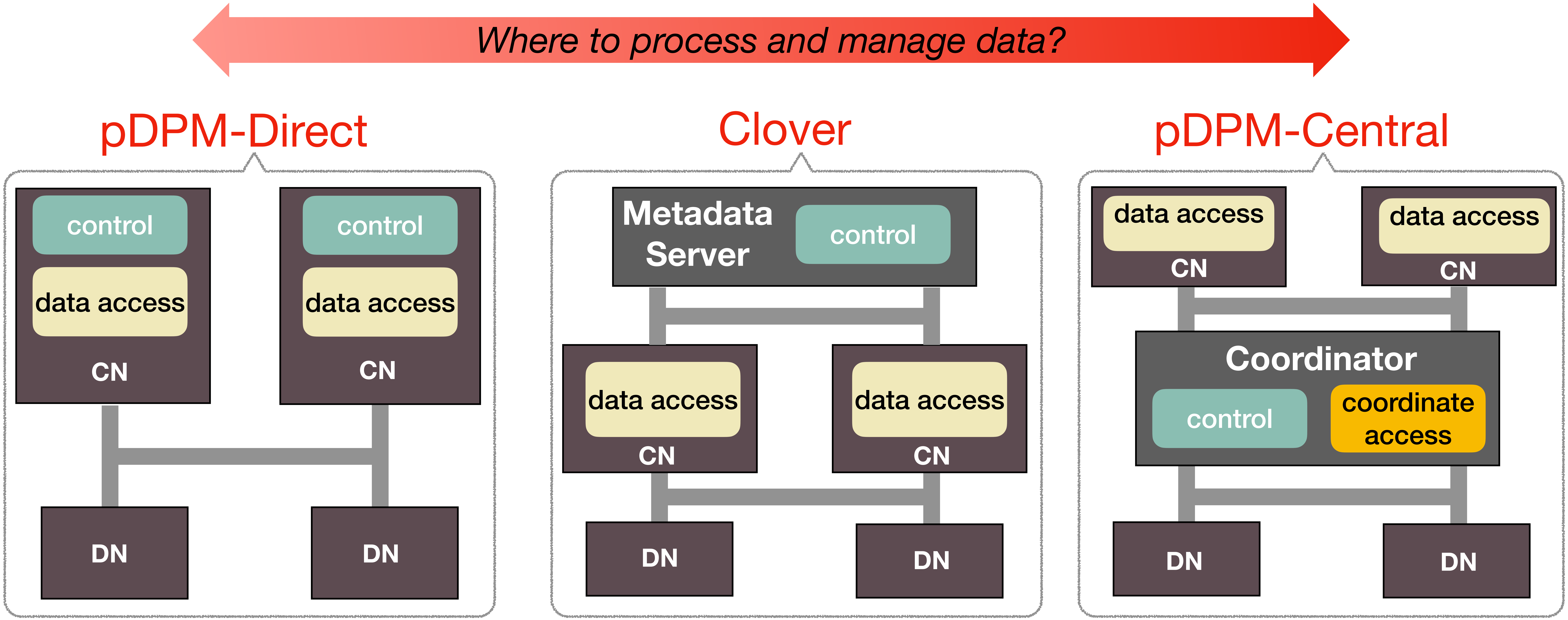
- Data Redundancy
  - User-defined replication degree
  - a novel atomic chaining replication
  - link a version to all the replicas of next version
- Clover can handle both DN and MS failures



Clover Replication

- Data Redundancy
  - User-defined replication degree
  - a novel atomic chaining replication
  - link a version to all the replicas of next version
- Clover can handle both DN and MS failures





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

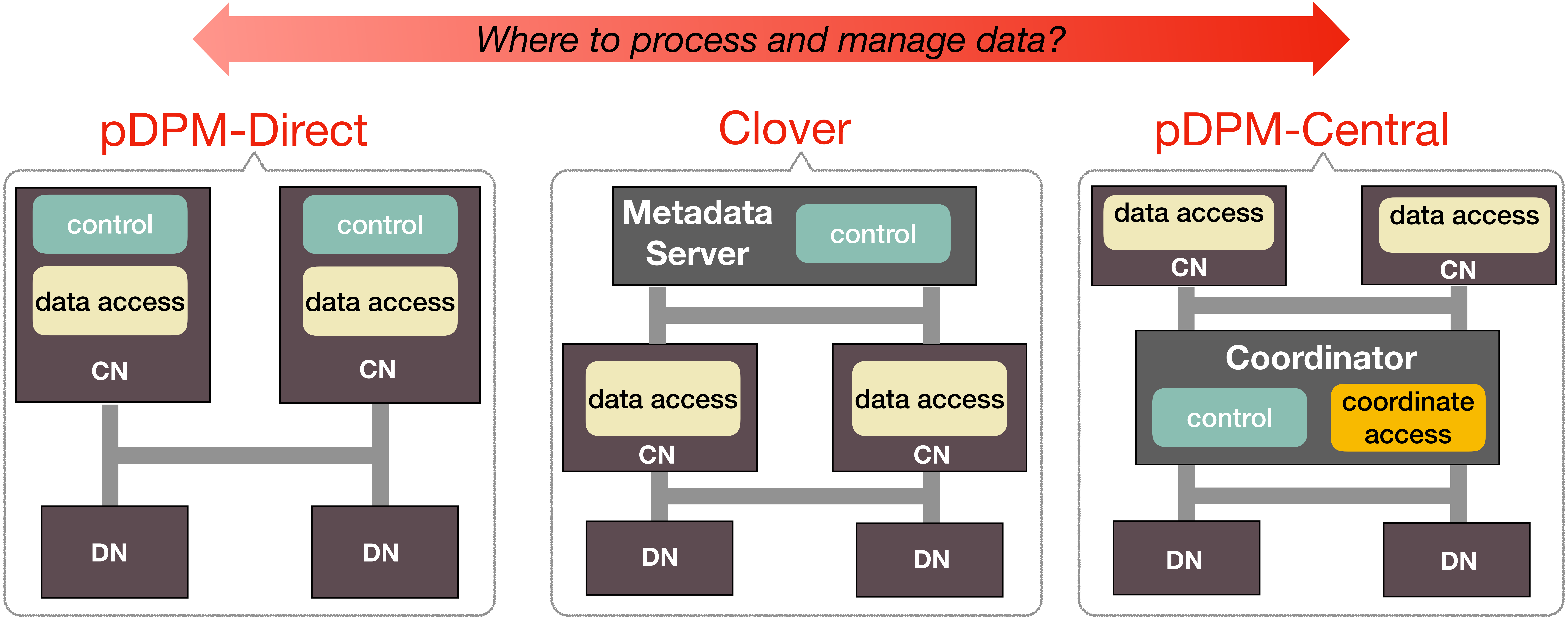
Coordinator

control

coordinate access

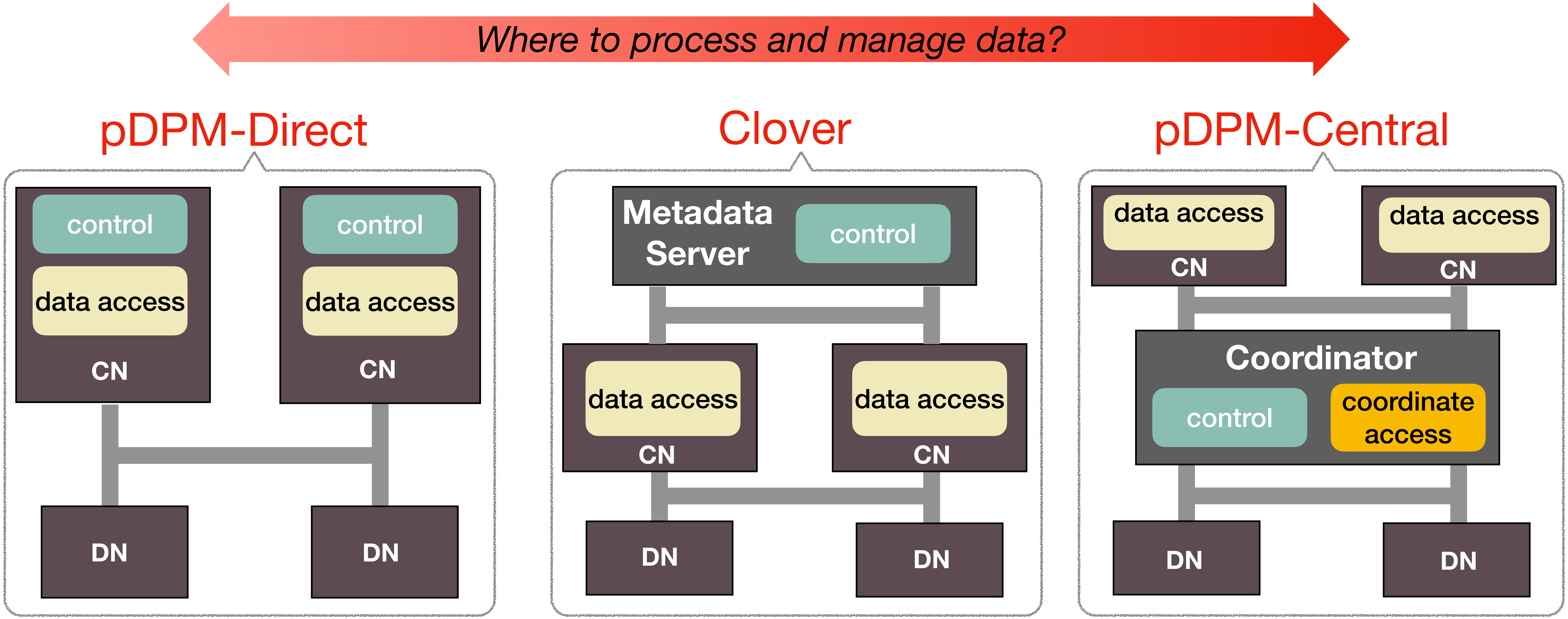
DN

DN



- Write cannot scale
- Large metadata consumption

- Extra read RTTs
- Coordinator cannot scale

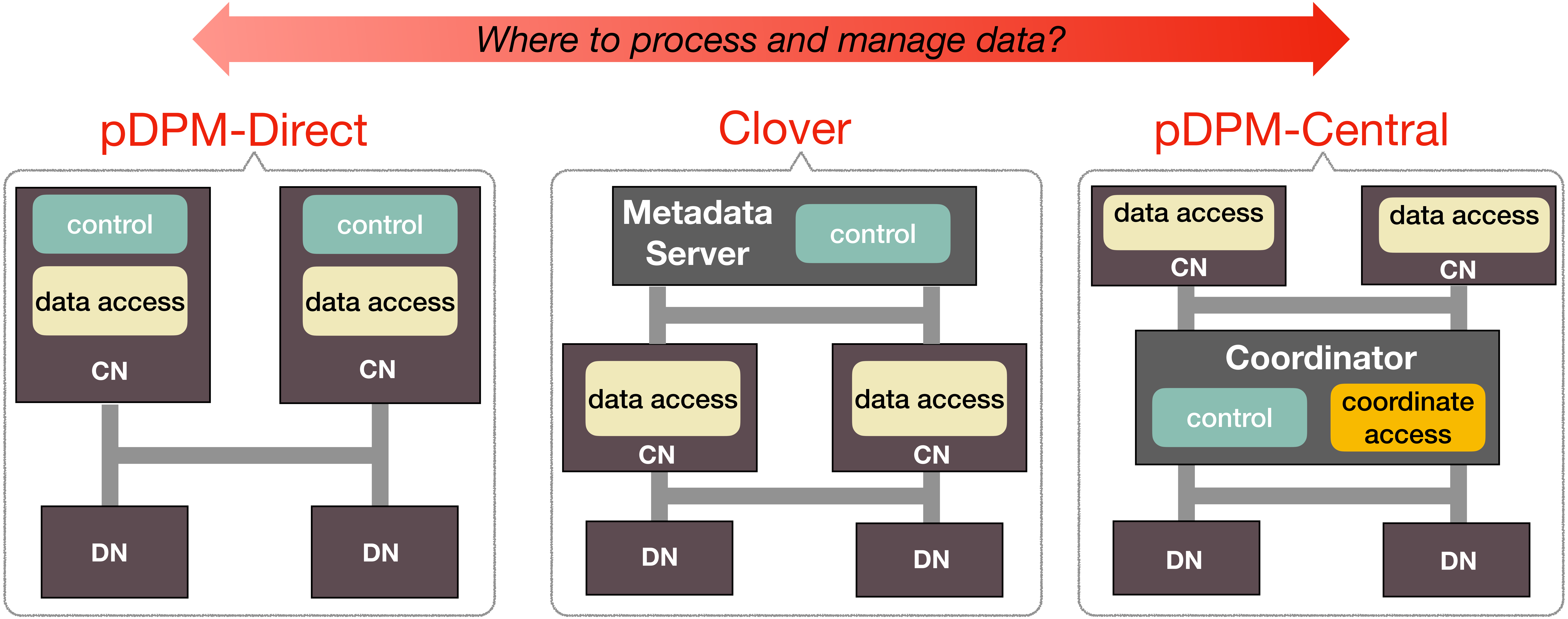


- Write cannot scale
- Large metadata consumption

- Extra read RTTs
- Coordinator cannot scale

Distributed data & metadata

Centralized data & metadata



- Write cannot scale
- Large metadata consumption

- Extra read RTTs
- Coordinator cannot scale

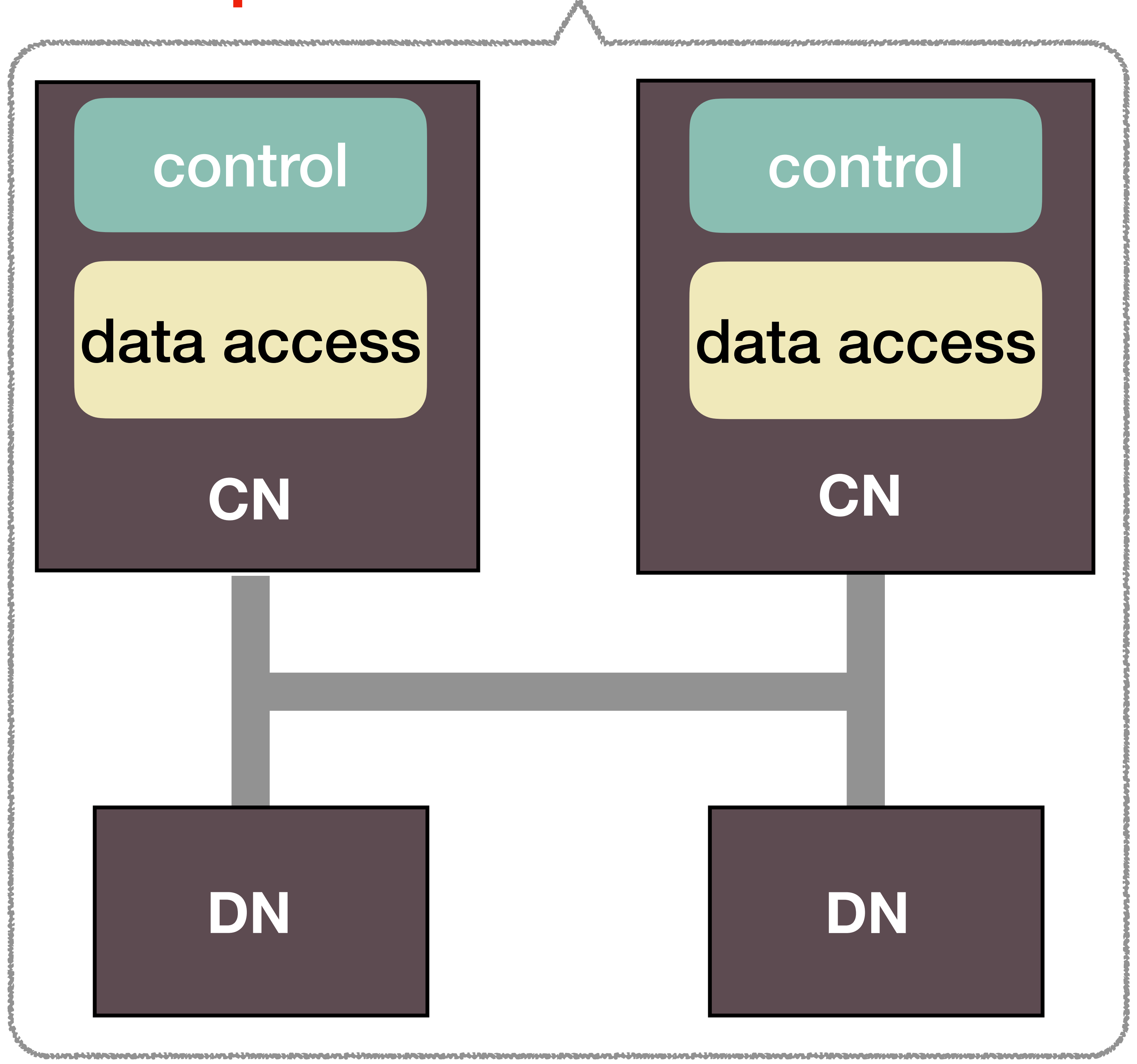
Distributed data & metadata

Separate data & metadata

Centralized data & metadata

Where to process and manage data?

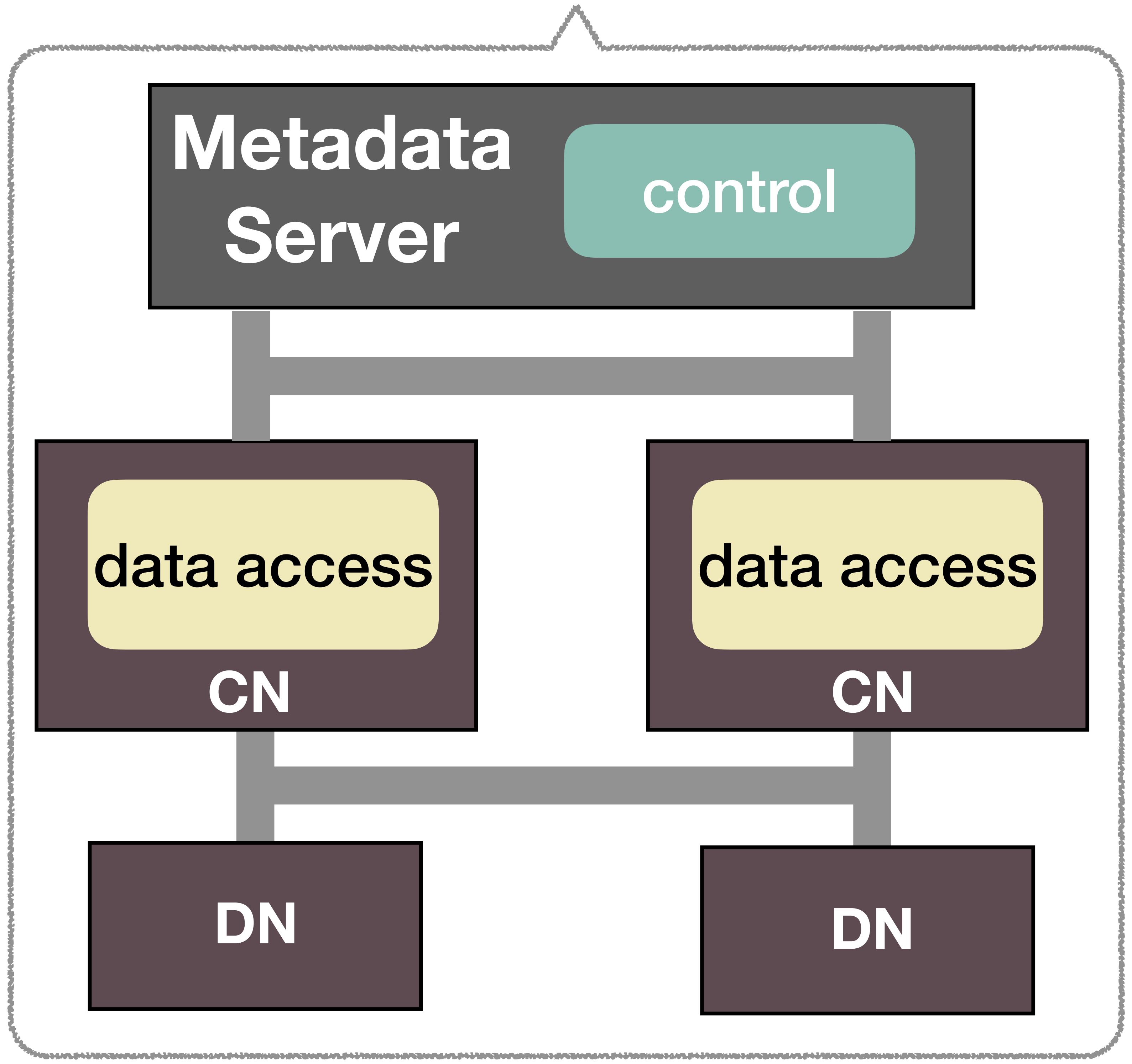
pDPM-Direct



- Write cannot scale
- Large metadata consumption

Distributed data & metadata

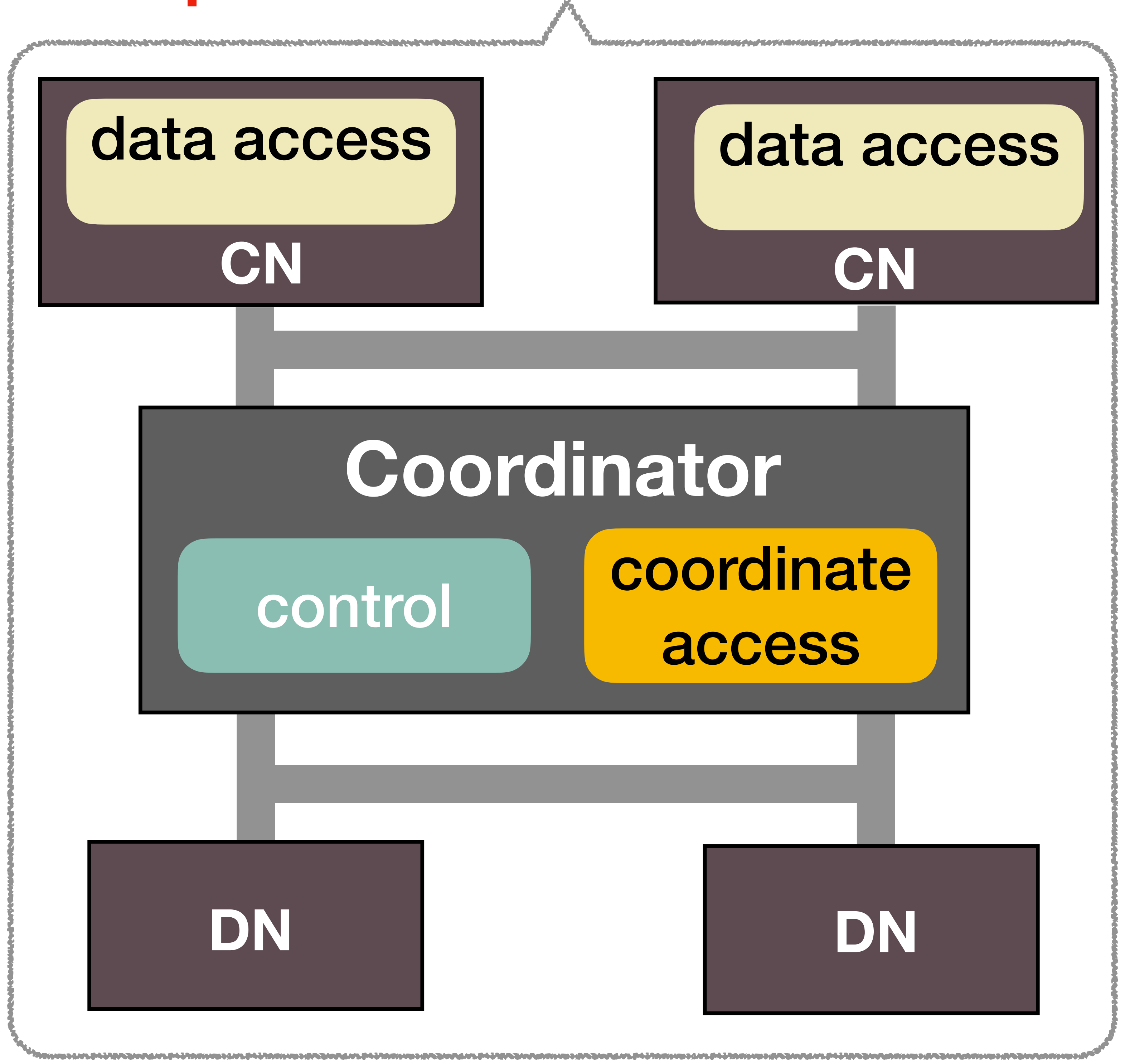
Clover



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

Separate data & metadata

pDPM-Central



- Extra read RTTs
- Coordinator cannot scale

Centralized data & metadata

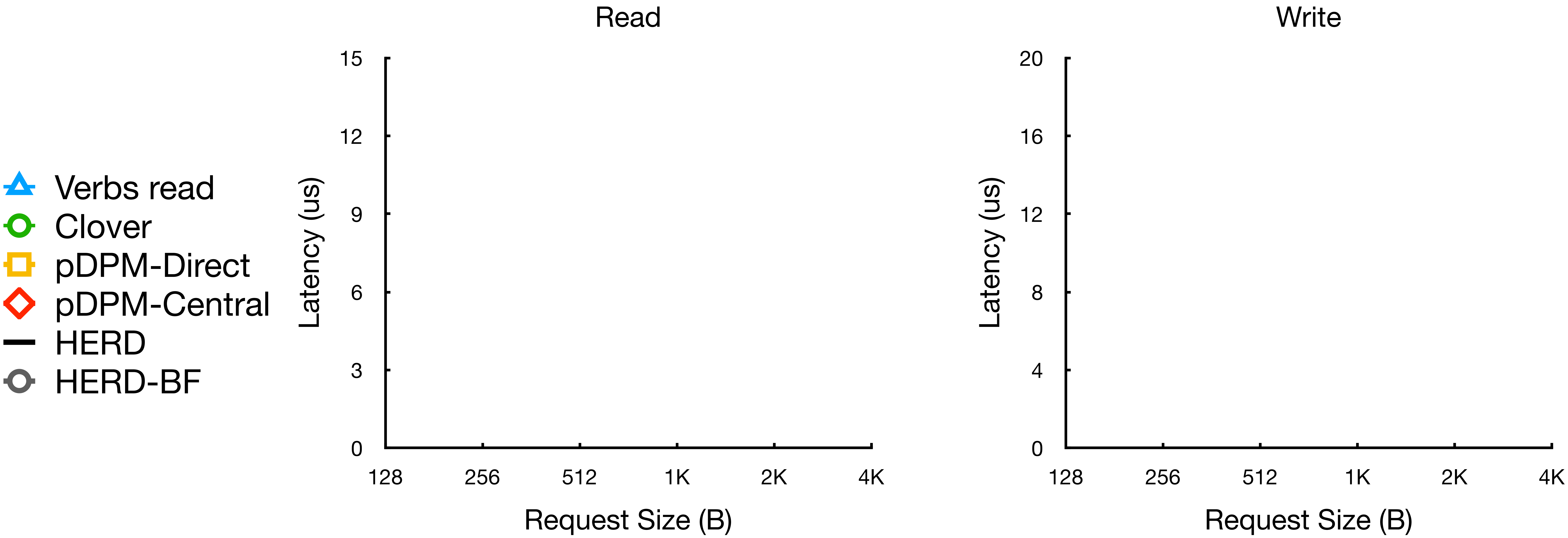
Evaluation Setup

- Systems evaluated
  - **pDPM Systems**: pDPM-Direct, pDPM-Central, Clover
  - **Non-disaggregated Systems**: Octopus, ATC’17 and Hotpot, SoCC’17
  - **Two-sided RDMA KVS**: HERD, SIGCOMM’14, ported HERD-BF (Bluefield)
- Testbed
  - 14 servers, each has an Intel Xeon E5-2620, 128 GB DRAM, and 100 Gpbs Mellanox ConnectX-4 NIC, all connected via a 100 Gpbs IB switch
  - Mellanox BlueField SmartNIC for HERD

Microbenchmark - Latency

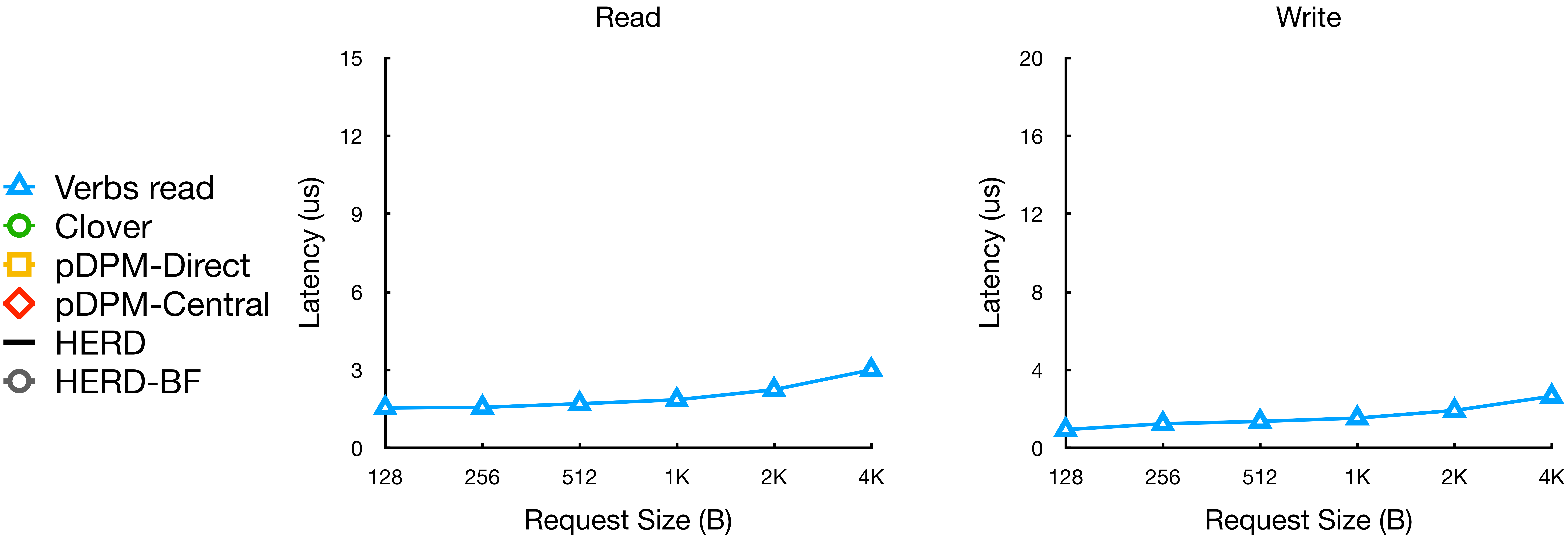
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-Bluefield use 12 polling threads

Microbenchmark - Latency



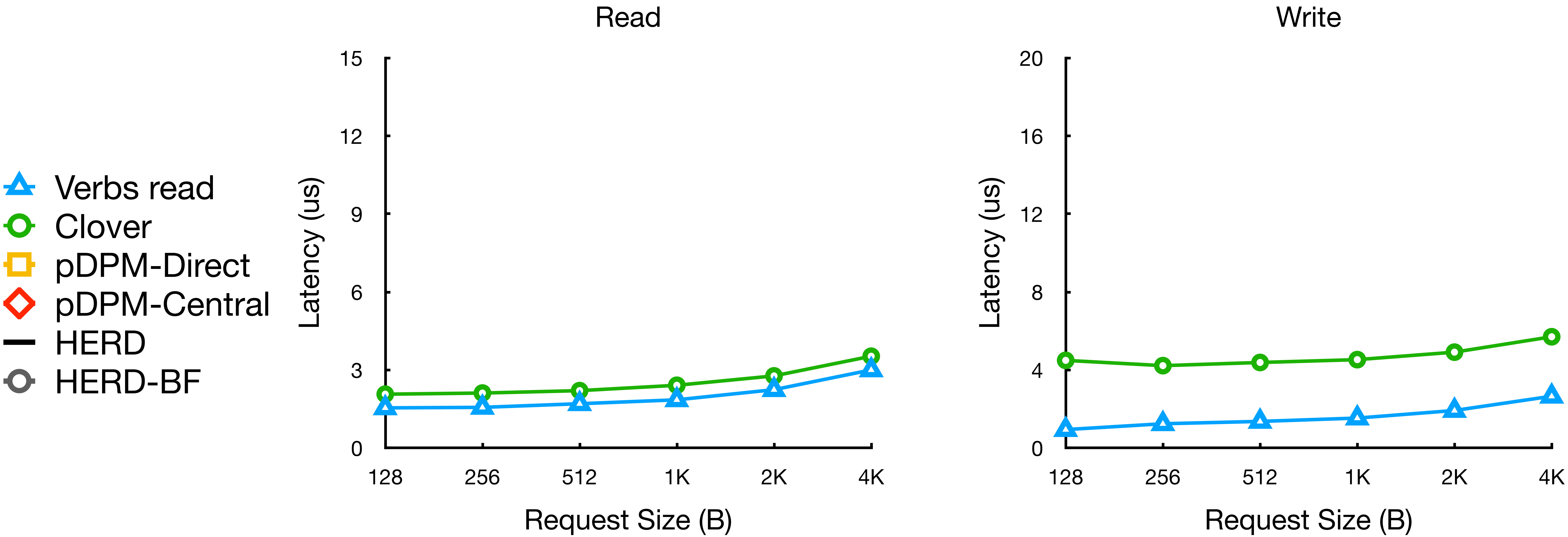
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-Bluefield use 12 polling threads

Microbenchmark - Latency



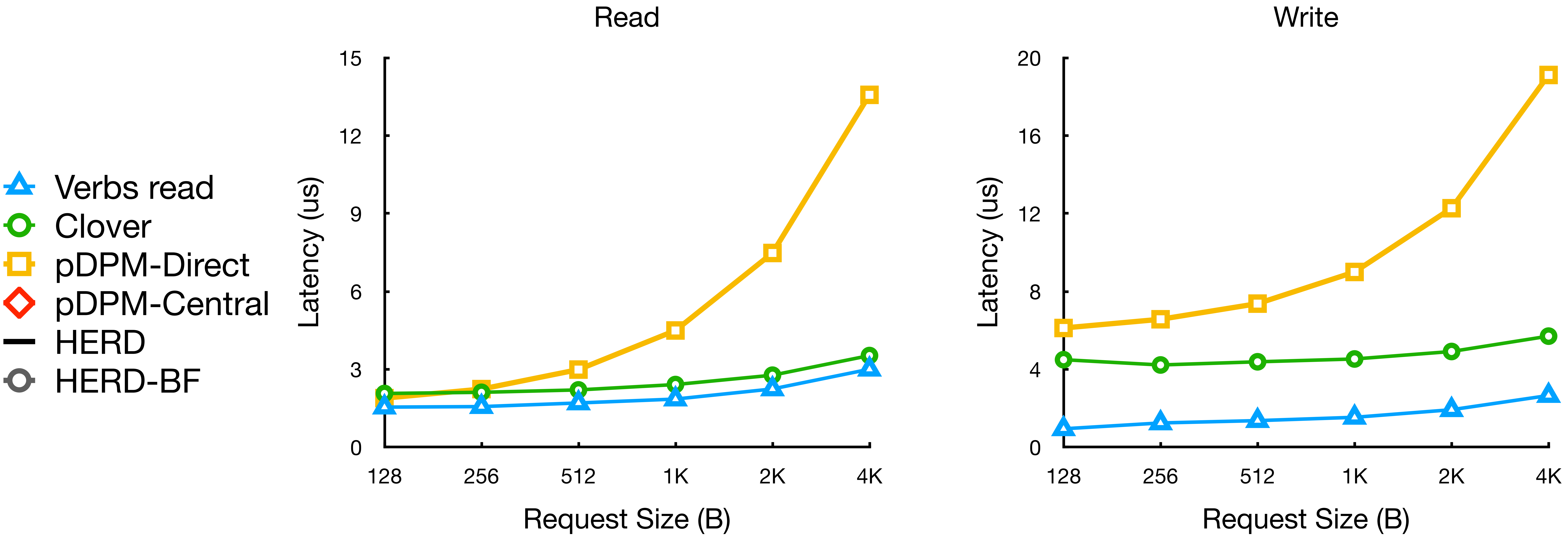
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-Bluefield use 12 polling threads

Microbenchmark - Latency



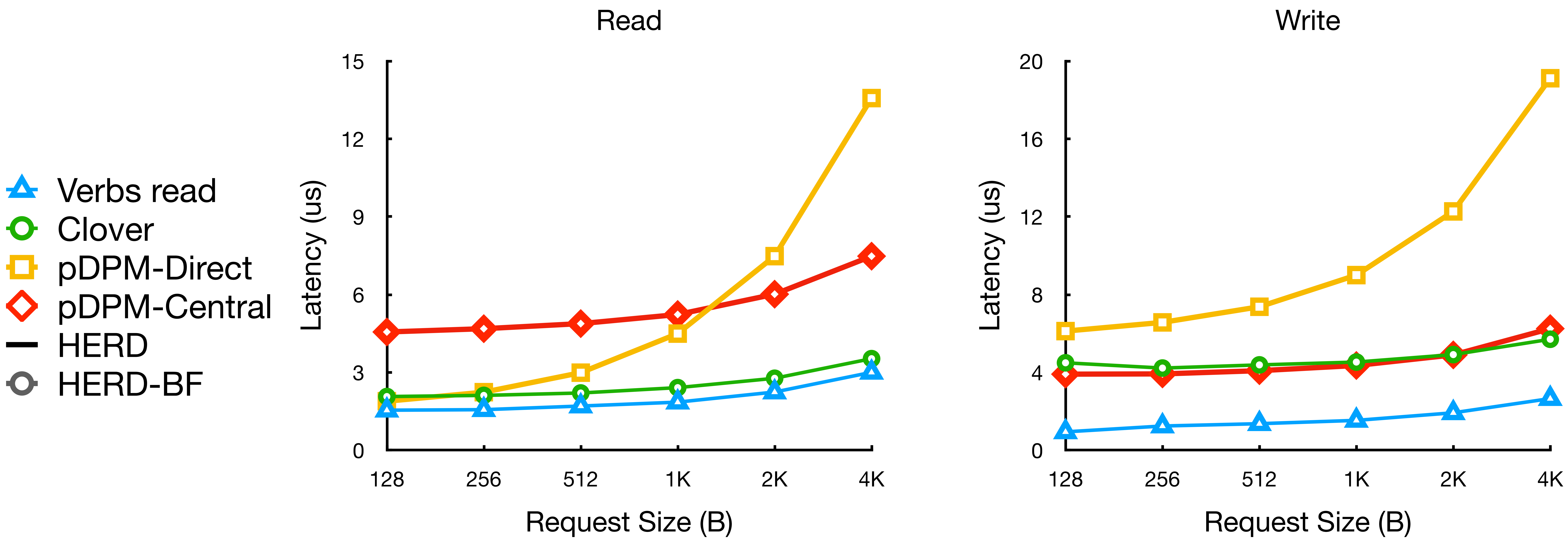
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-Bluefield use 12 polling threads

Microbenchmark - Latency



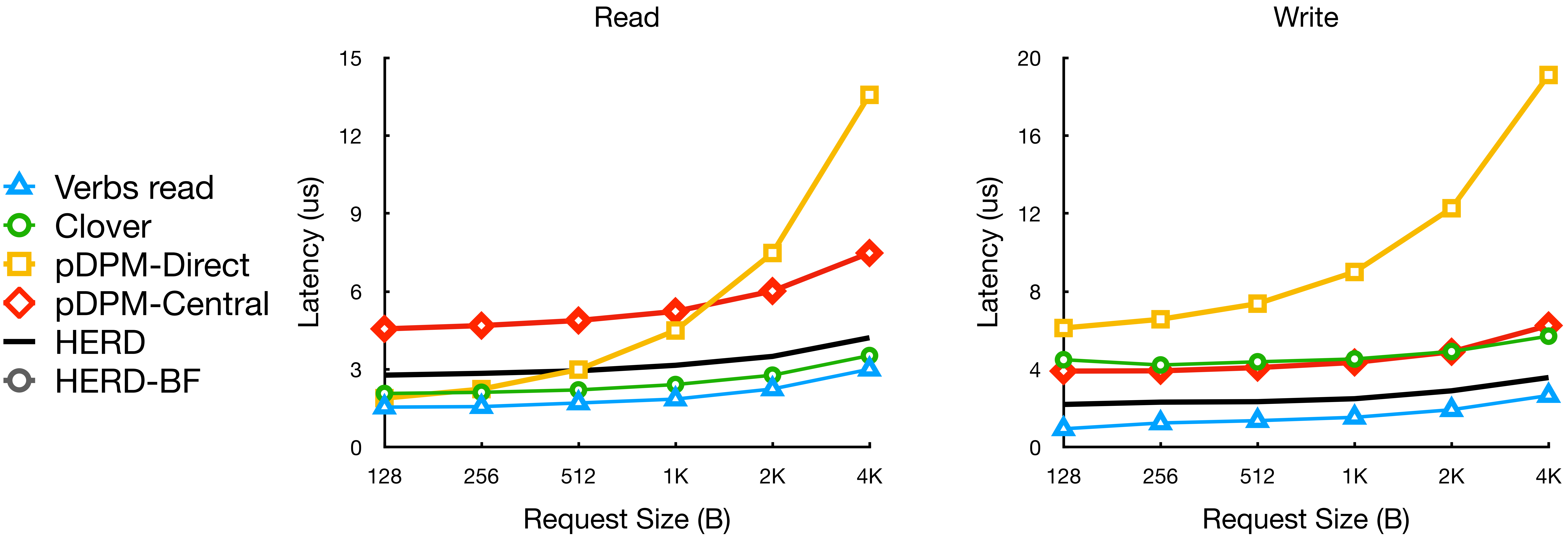
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-Bluefield use 12 polling threads

Microbenchmark - Latency



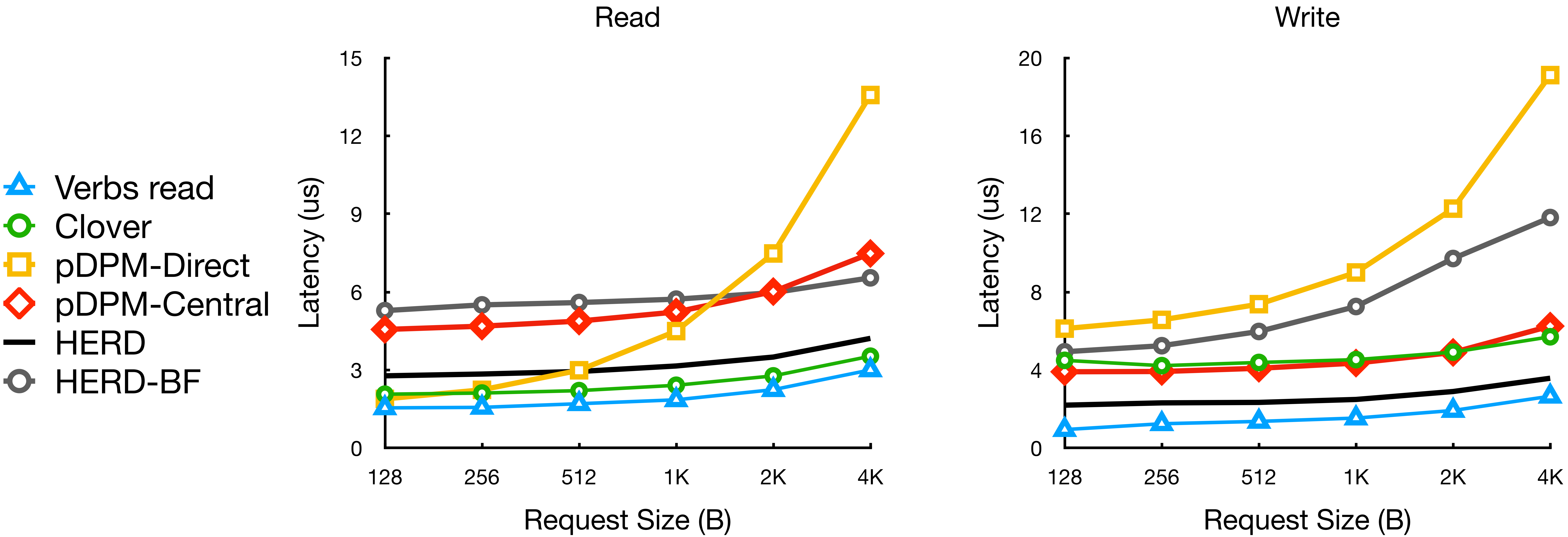
- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-Bluefield use 12 polling threads

Microbenchmark - Latency



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

Microbenchmark - Latency

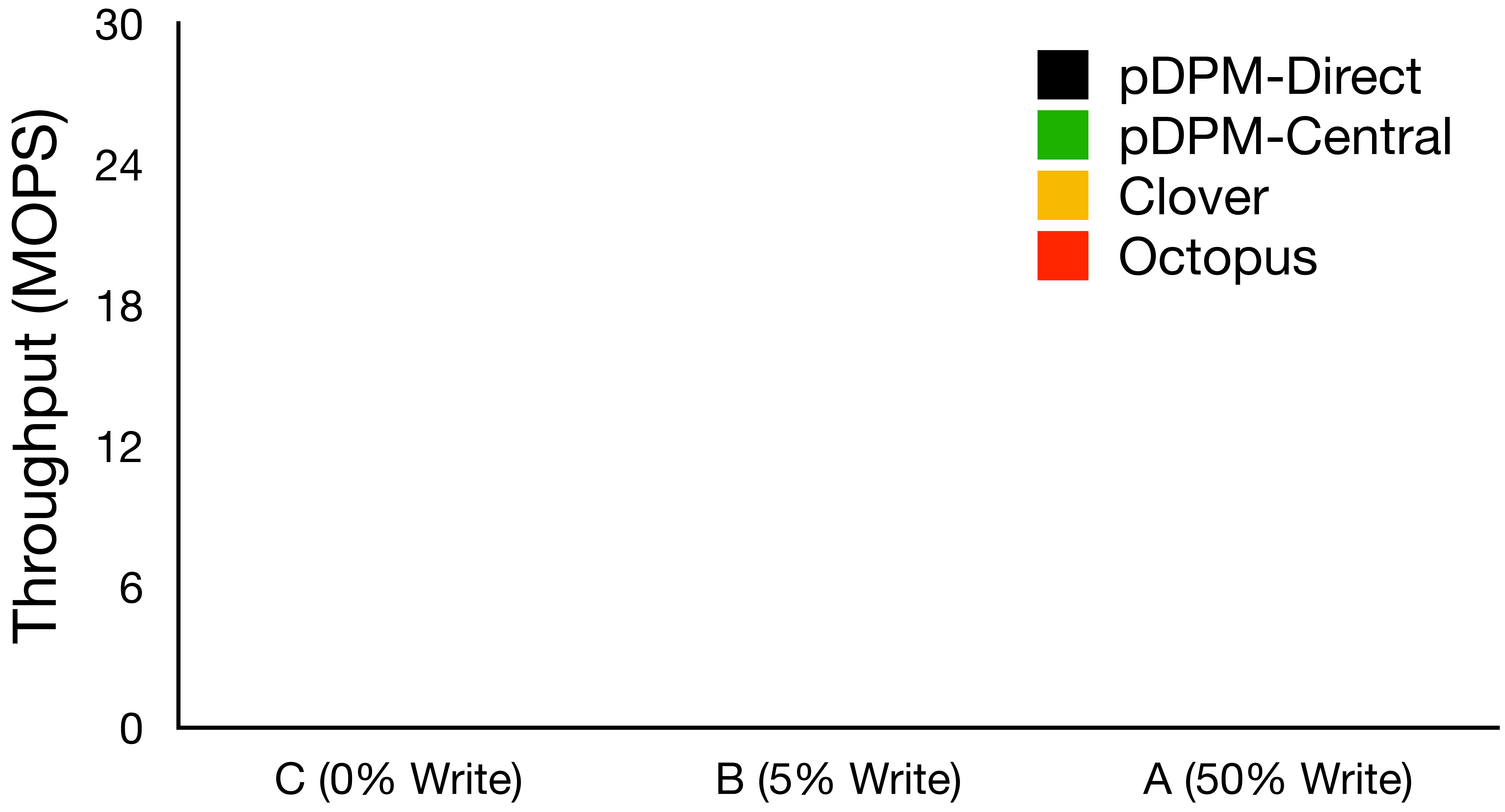


- One CN synchronously reads/writes a KV entry on a DN
- HERD and HERD-Bluefield use 12 polling threads

YCSB Results

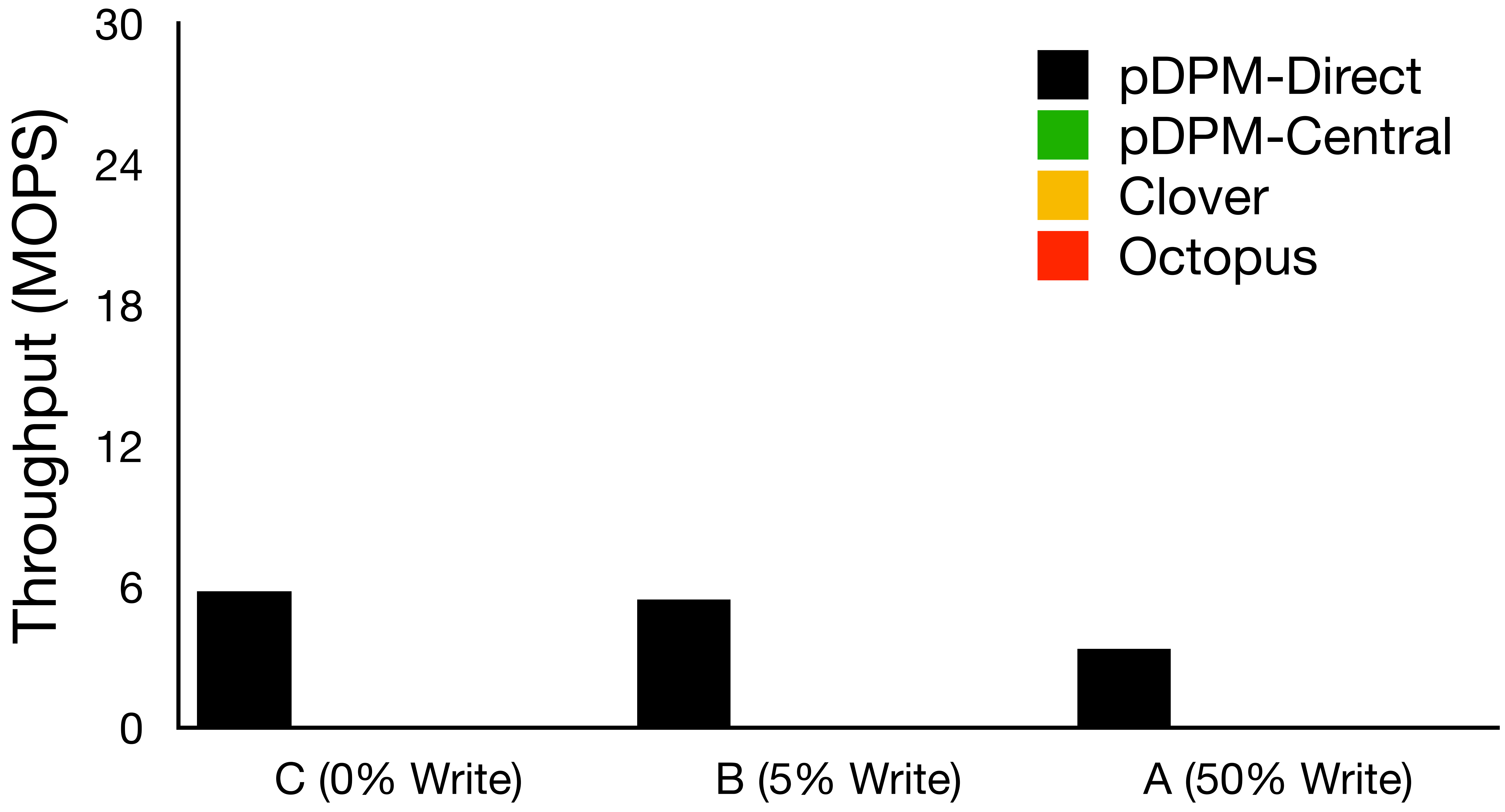
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



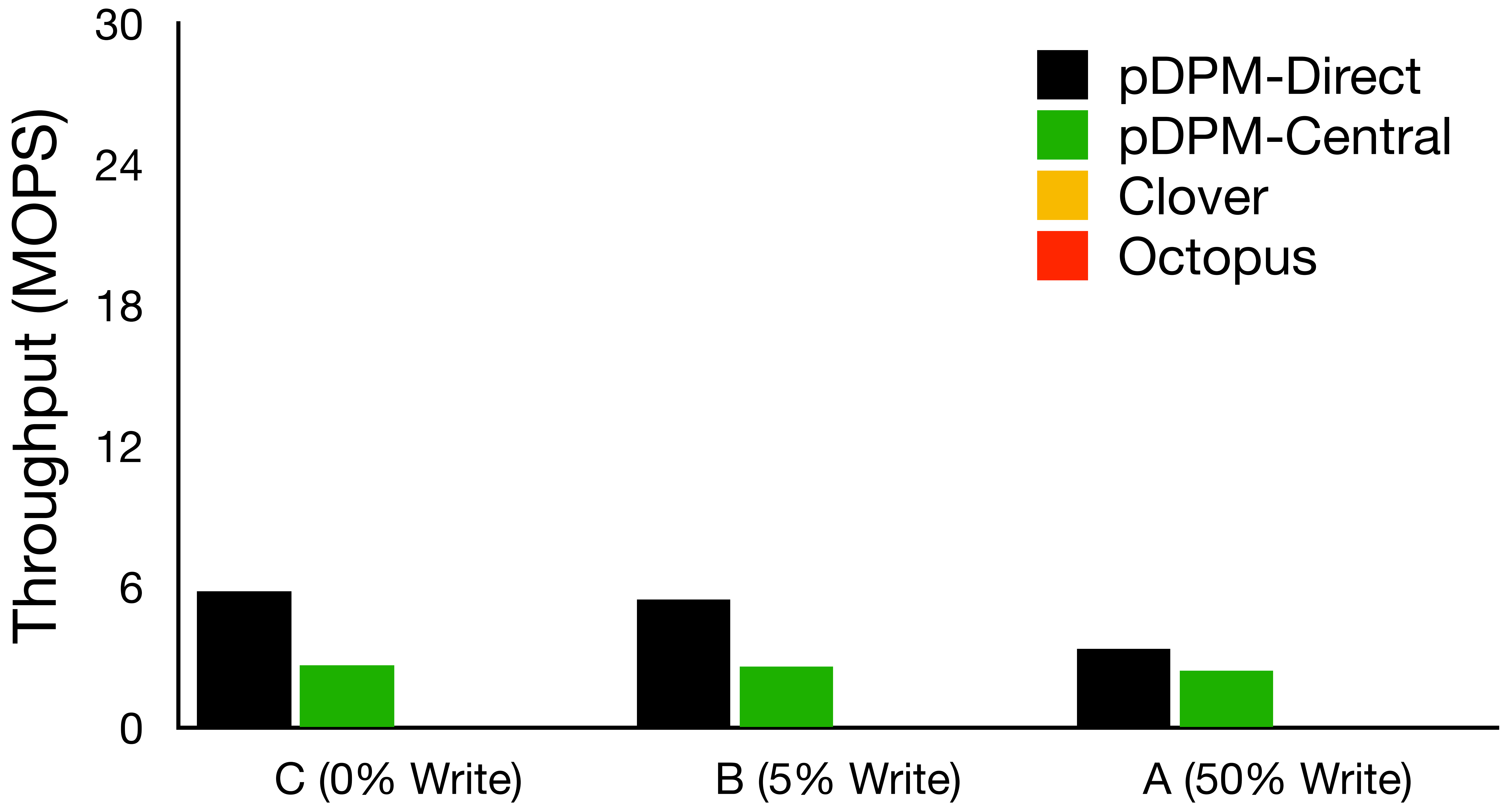
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



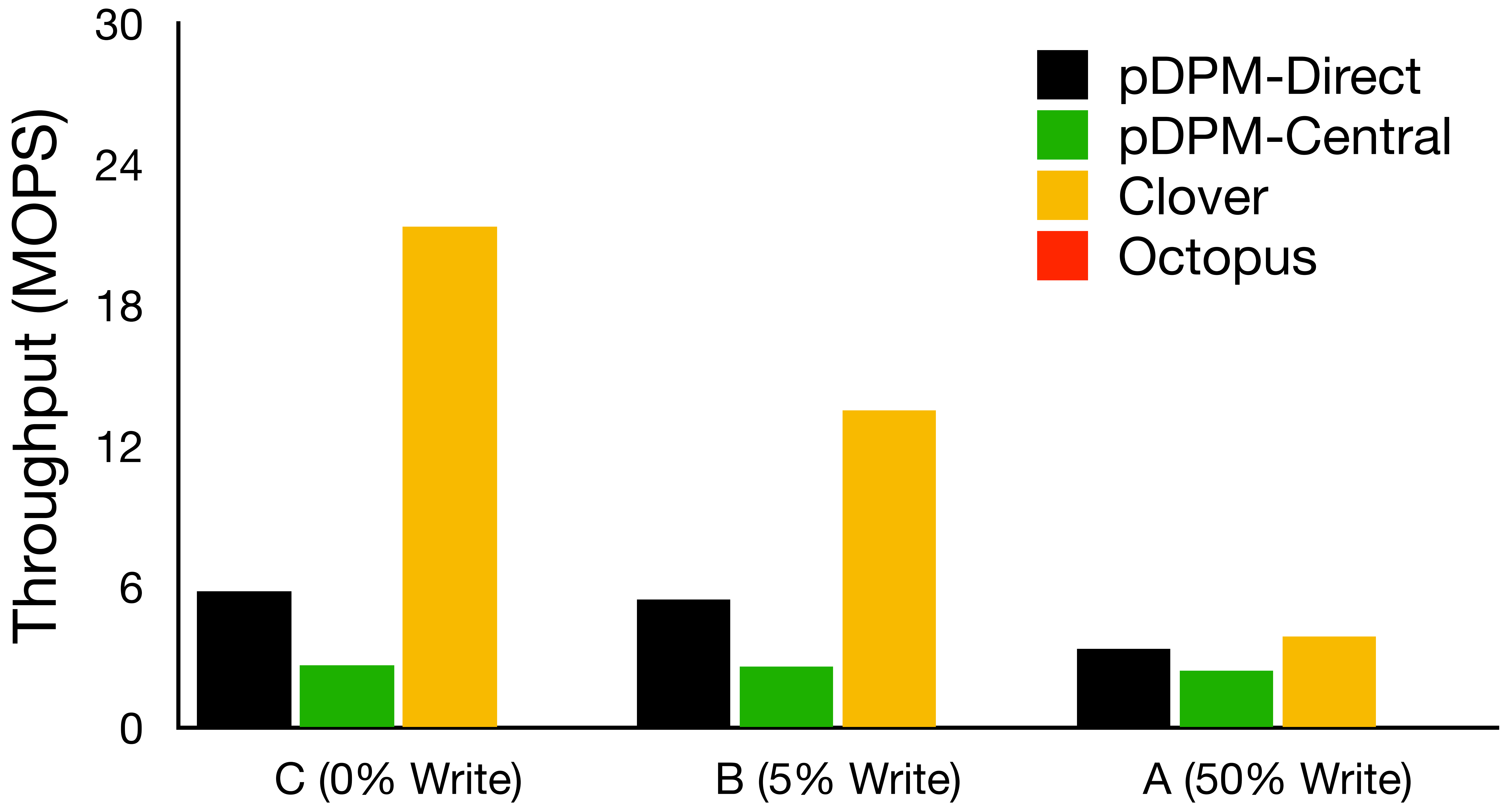
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



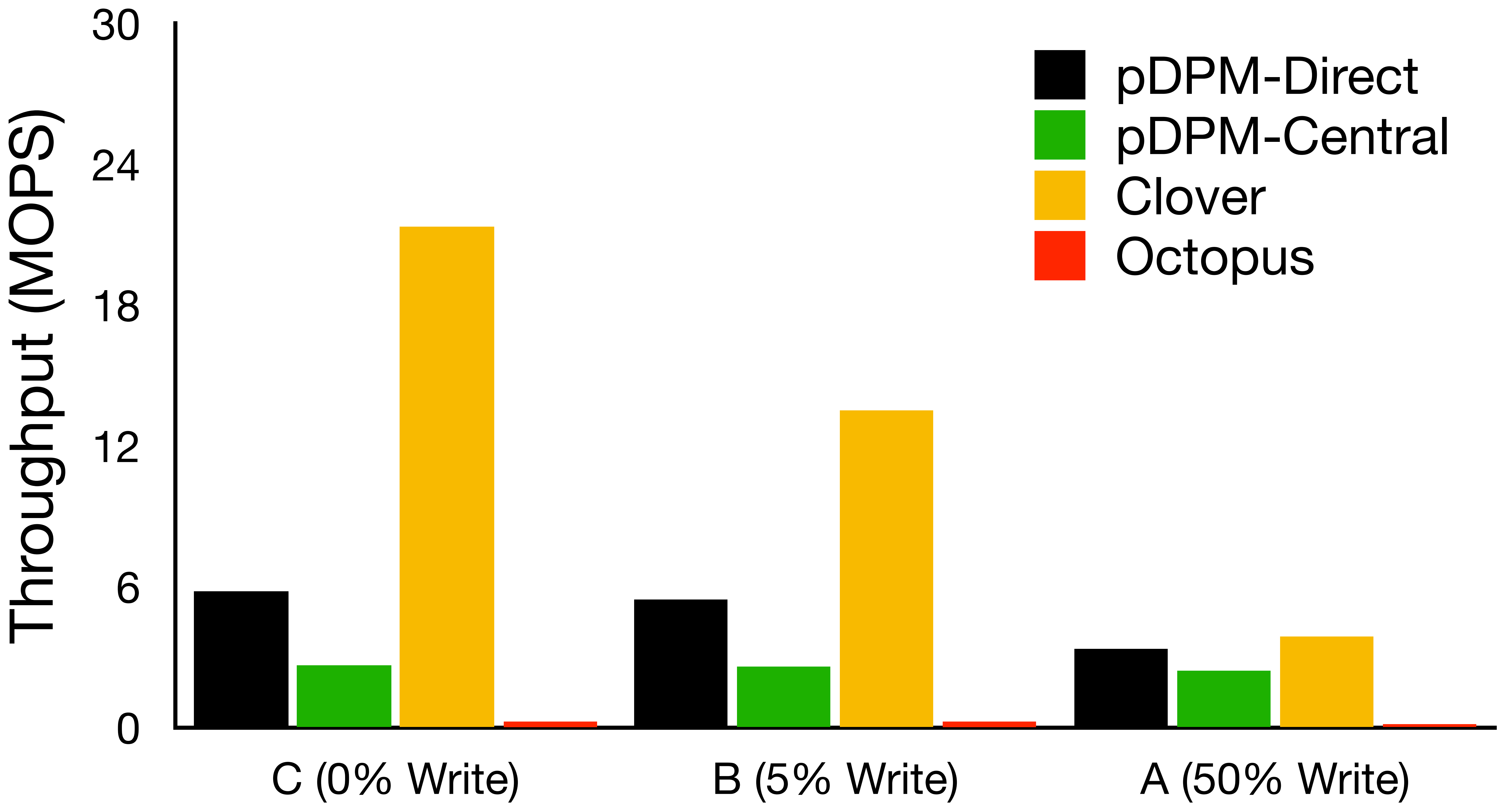
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



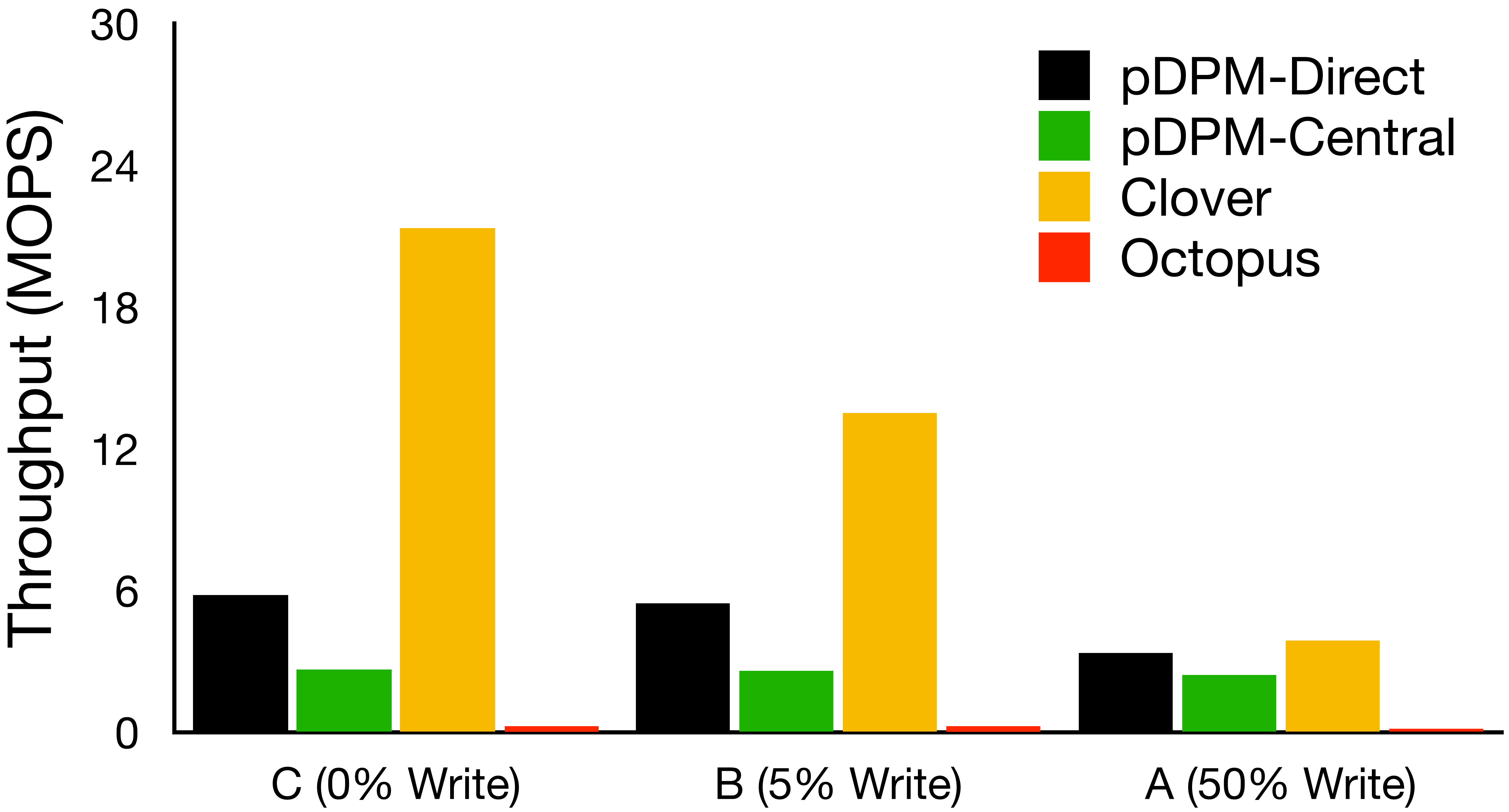
- 100K KV entries, 1 million operations, Zipf access distribution
- 4 CNs (8 threads per CN), 4 DNs

YCSB Results



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

YCSB Results

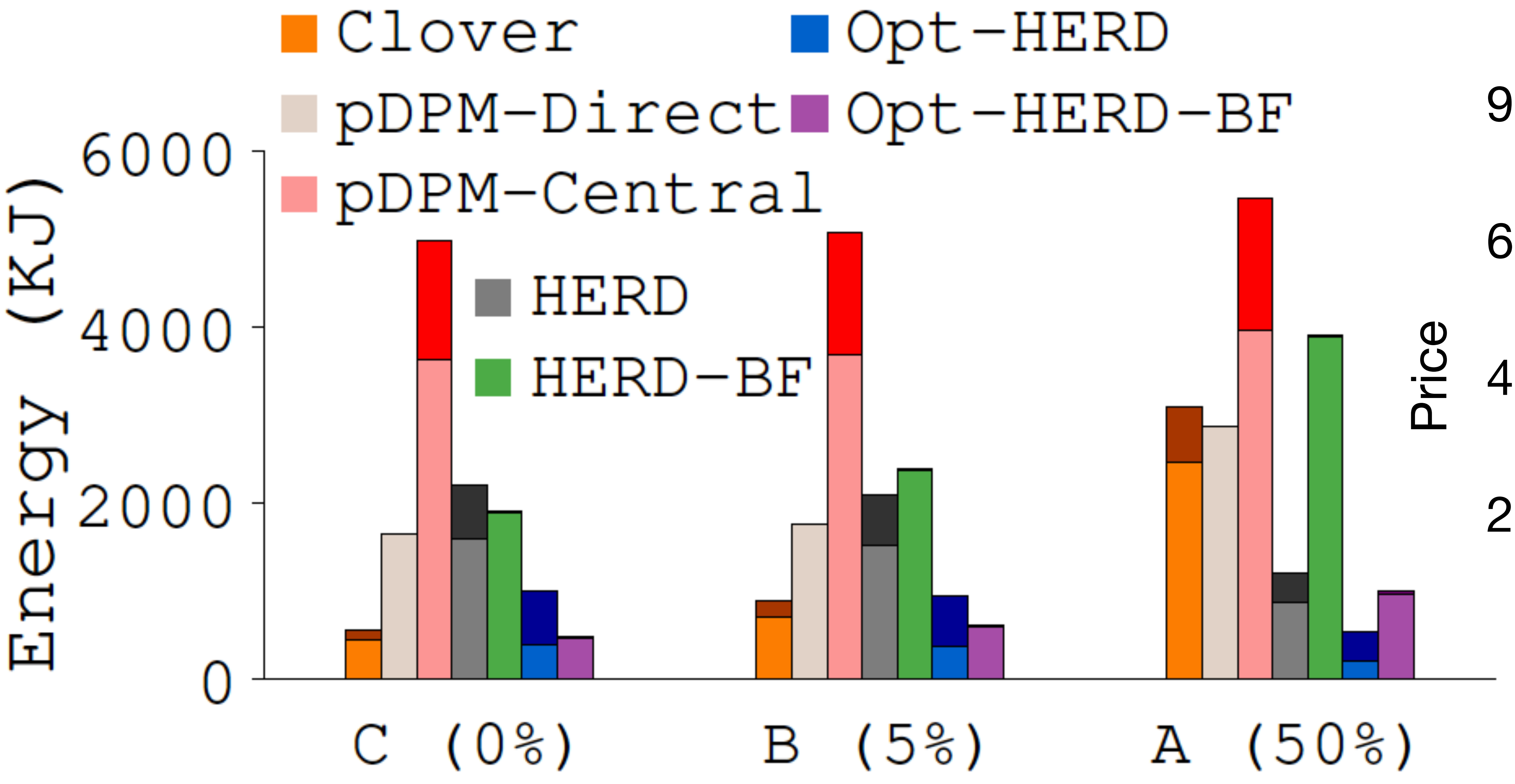


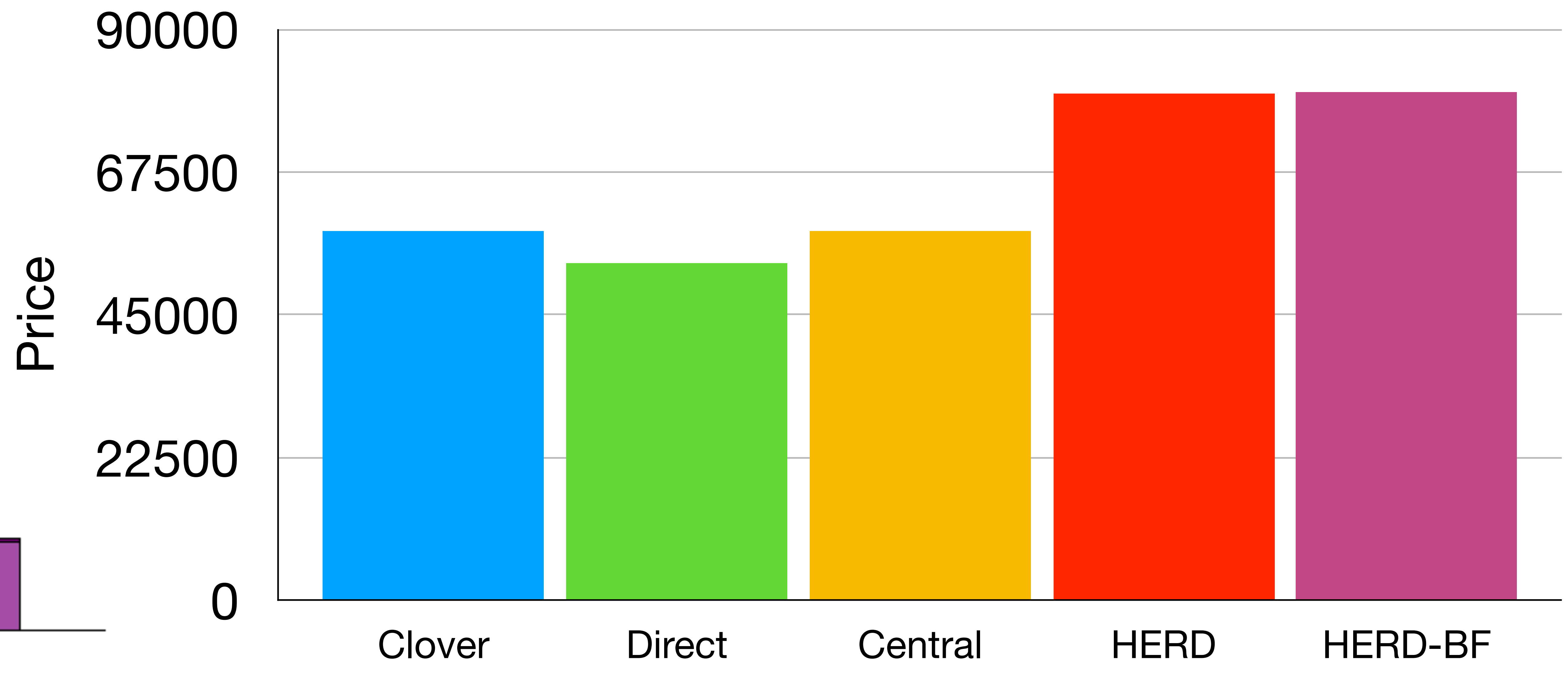
Clover RTTs

	Median	Avg	99%
C	1	1	1
B	1	1.26	5
A	1	1.33	6

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

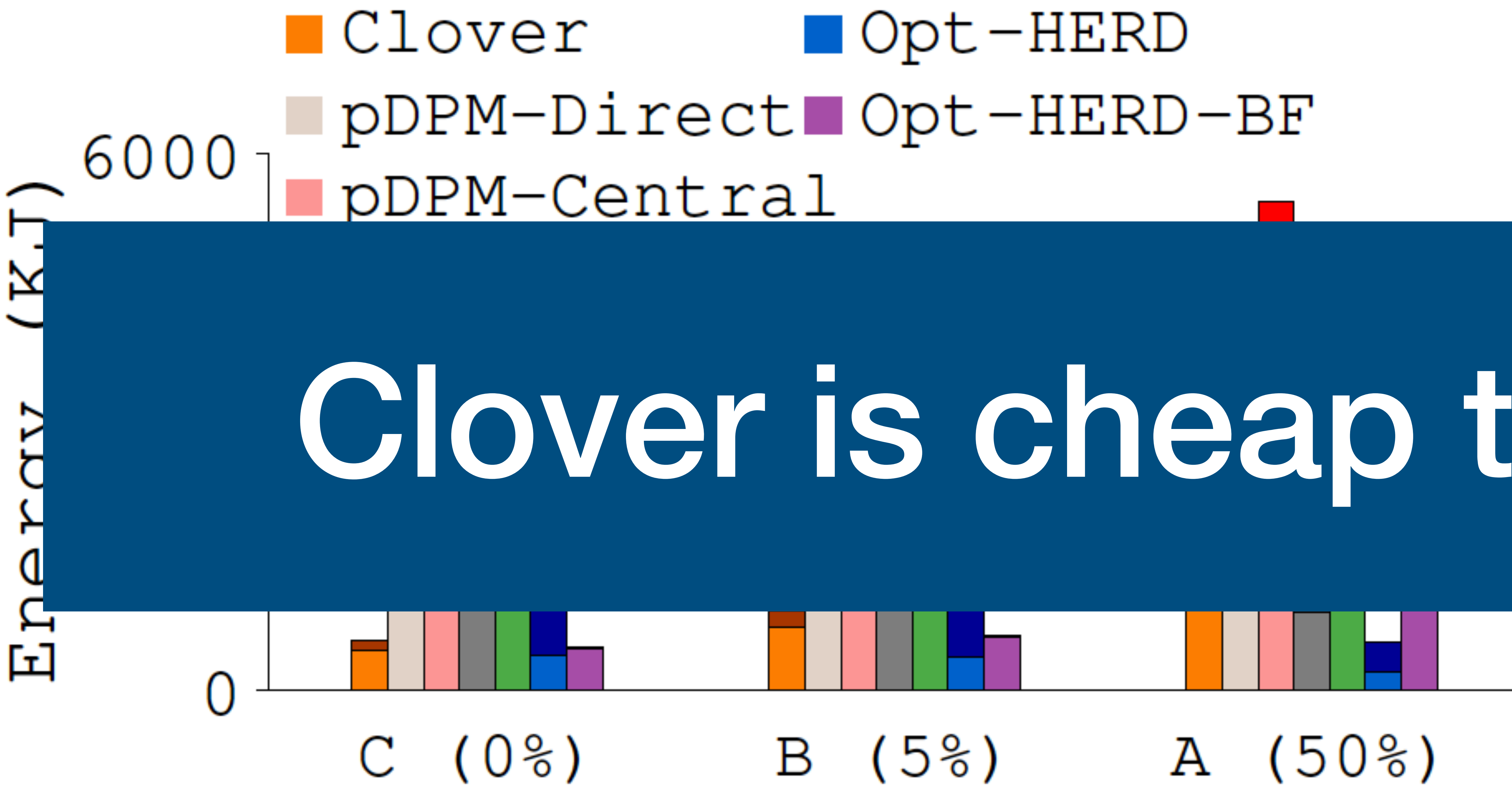
OPEX and CAPEX

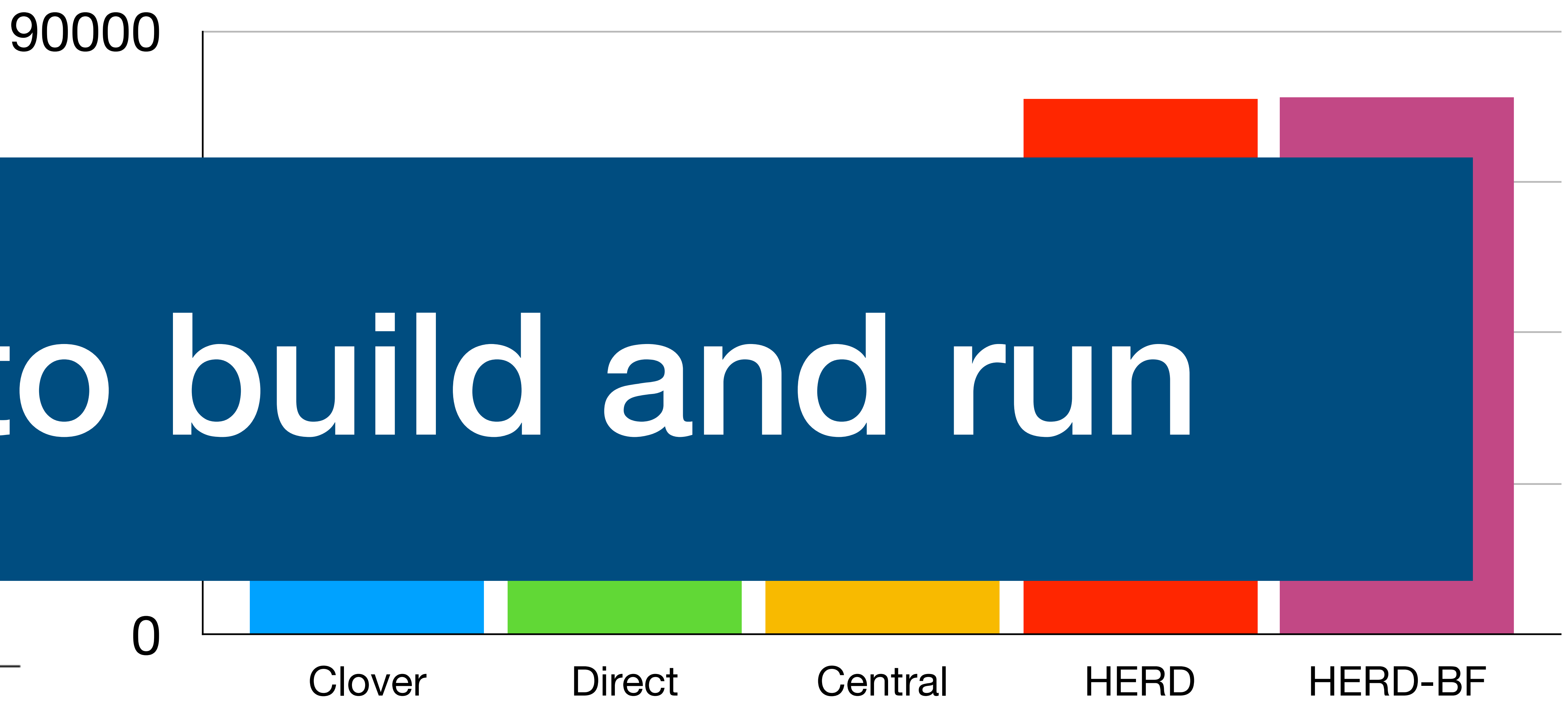




- Total energy to complete 10 million YCSB requests
- Includes all parties (CN and CN), except PM power usage

OPEX and CAPEX





Clover is cheap to build and run

- Total energy to complete 10 million YCSB requests
- Includes all parties (CN and CN), except PM power usage

Conclusion

- pDPM offers deployment, cost, and performance benefits
- Separating data and metadata is crucial
- Future system could benefit from a hybrid hardware model

Thank you!

Visit us @ wuklab.io
sysnet.ucsd.edu

Open source @ github.com/WukLab/pDPM

WukLab.io

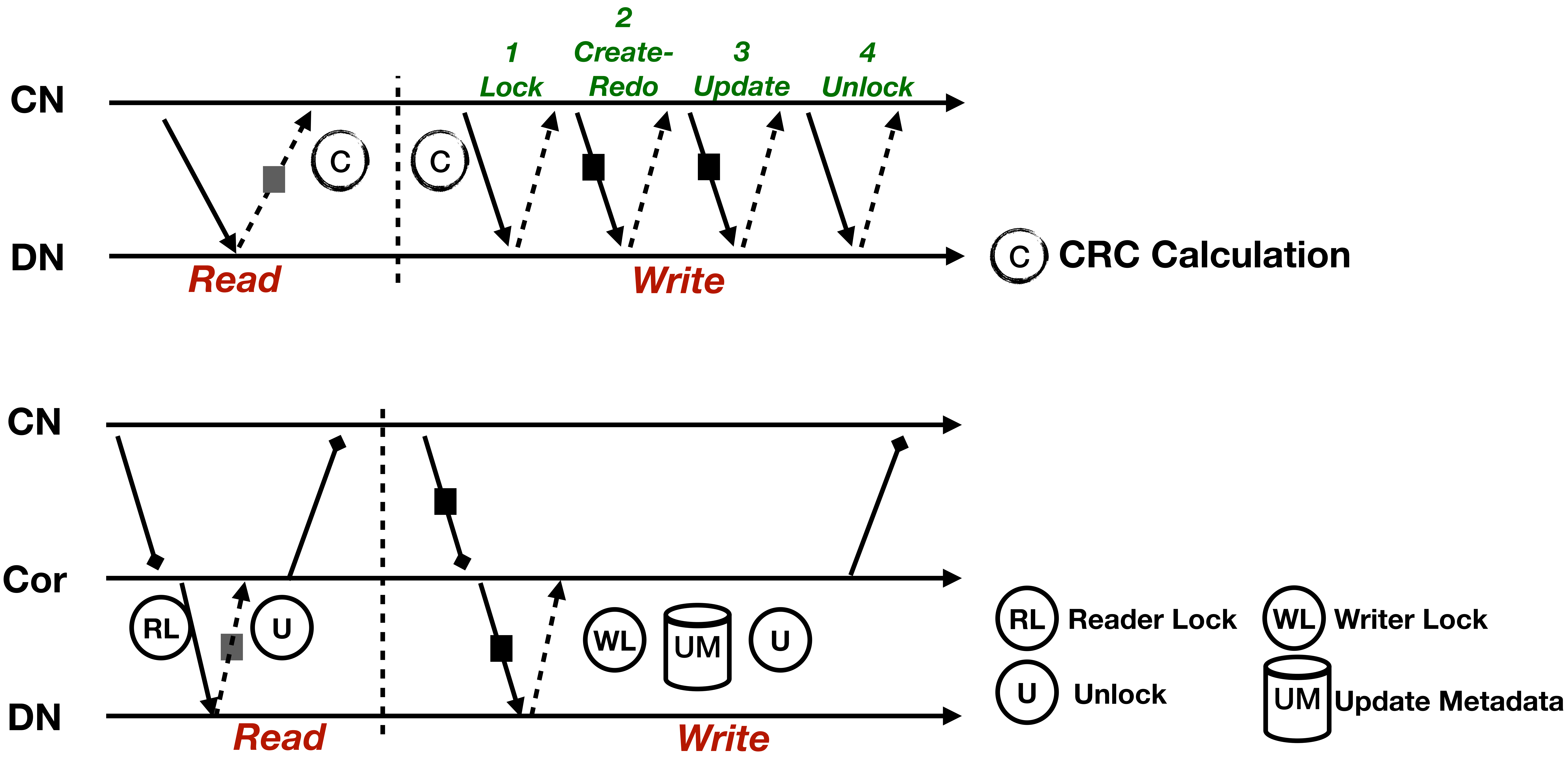
USENIX ATC '20

PURDUE UNIVERSITY

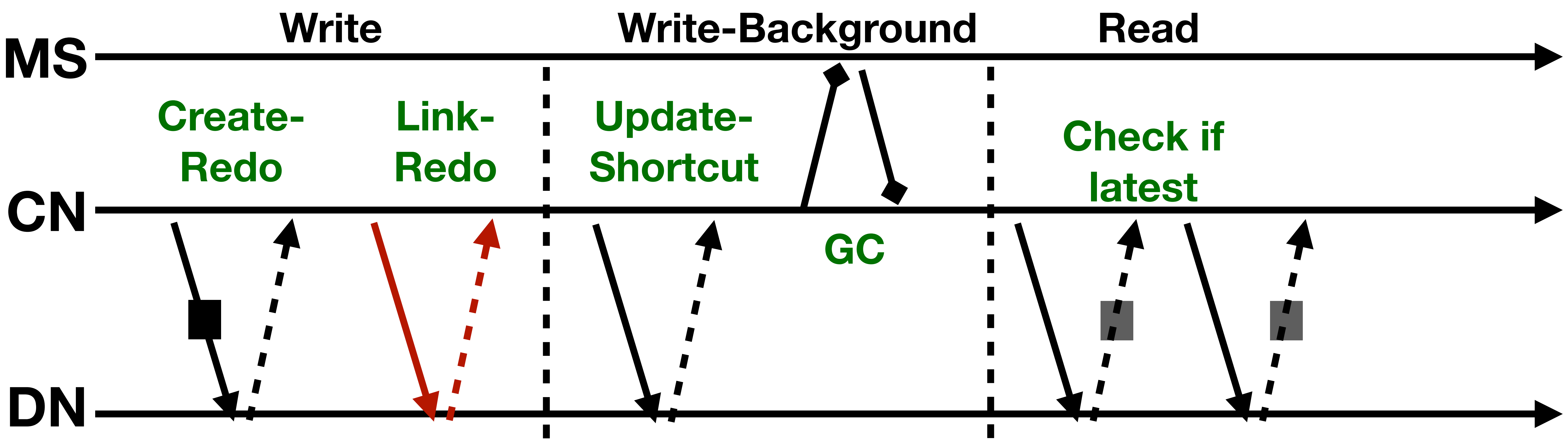
UC San Diego

Backup Slides

pDPM-Direct/Central RW Protocols

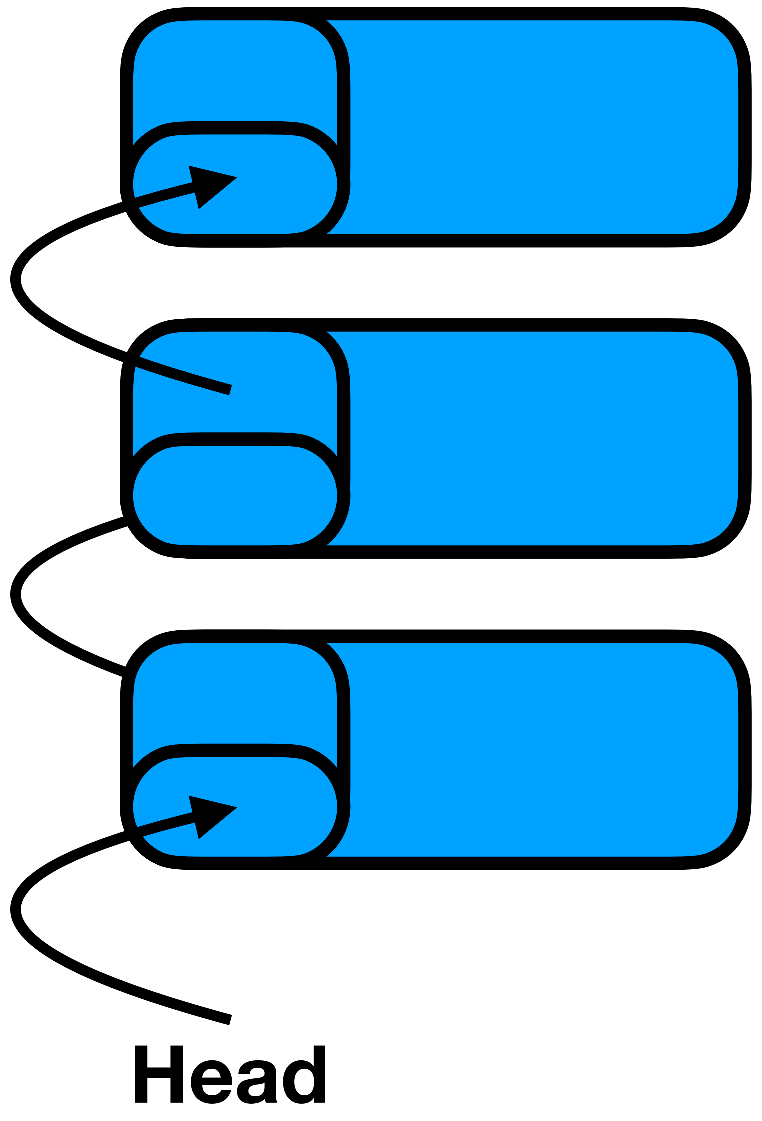


Clover RW Protocols

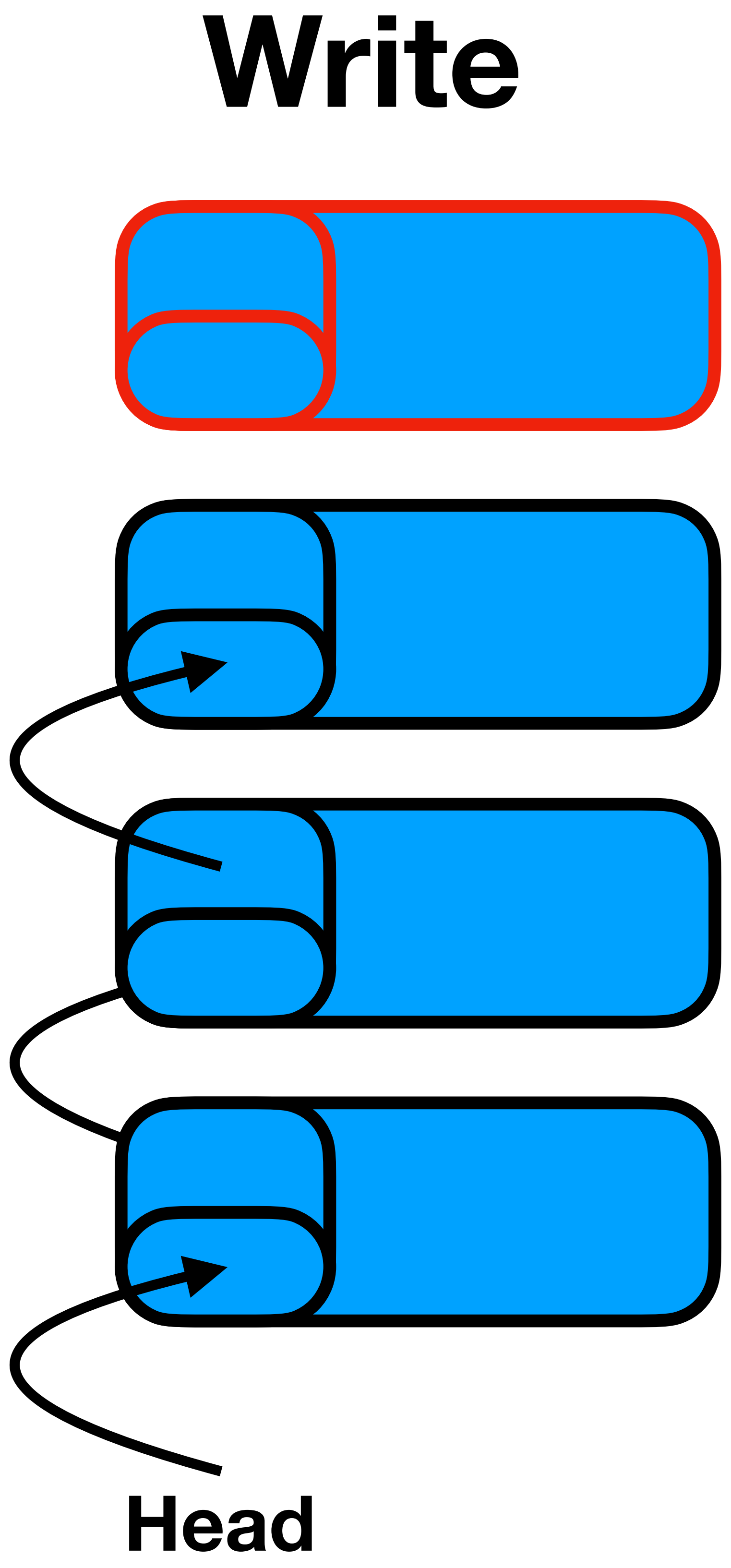


Clover Data Structure

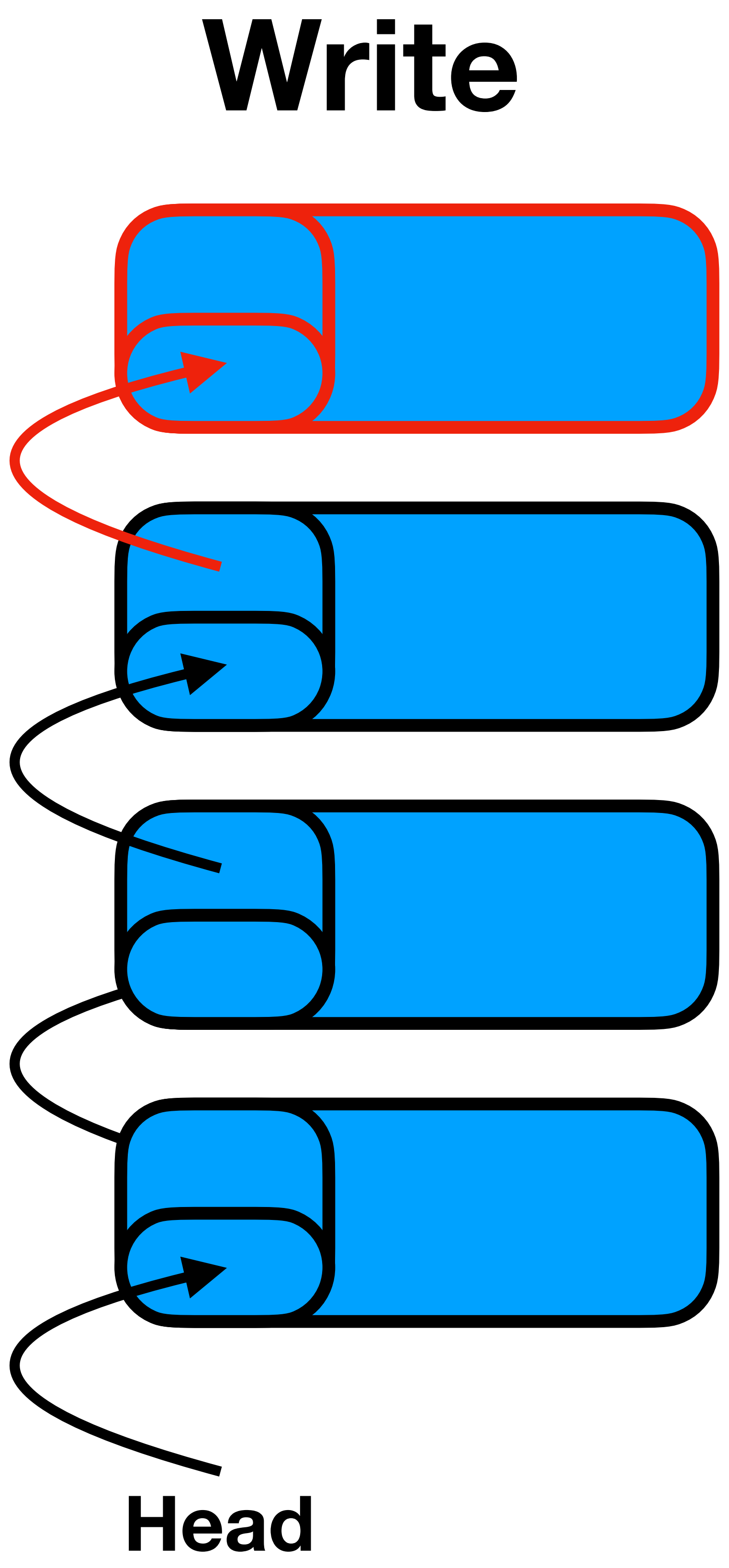
Write



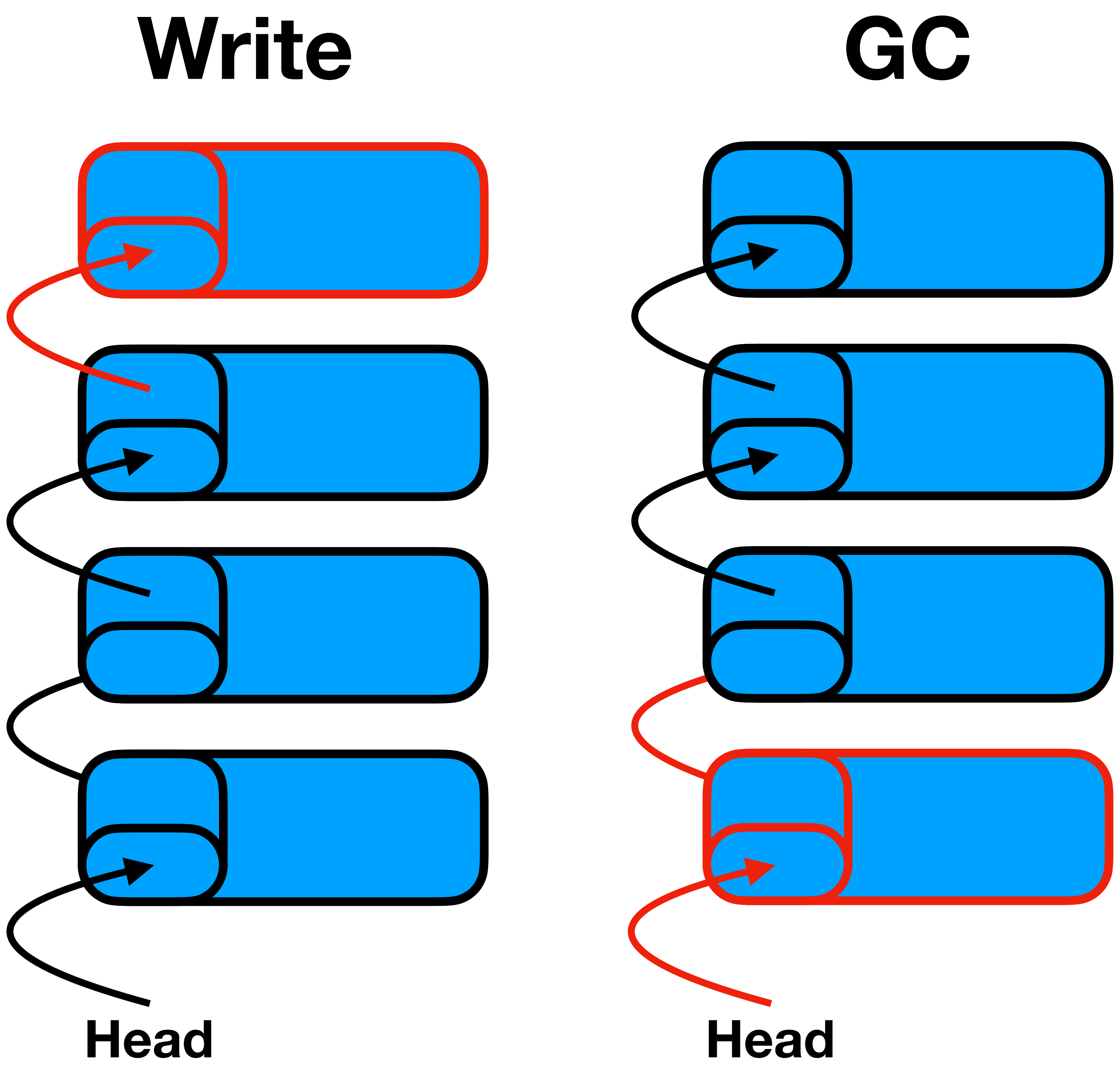
Clover Data Structure



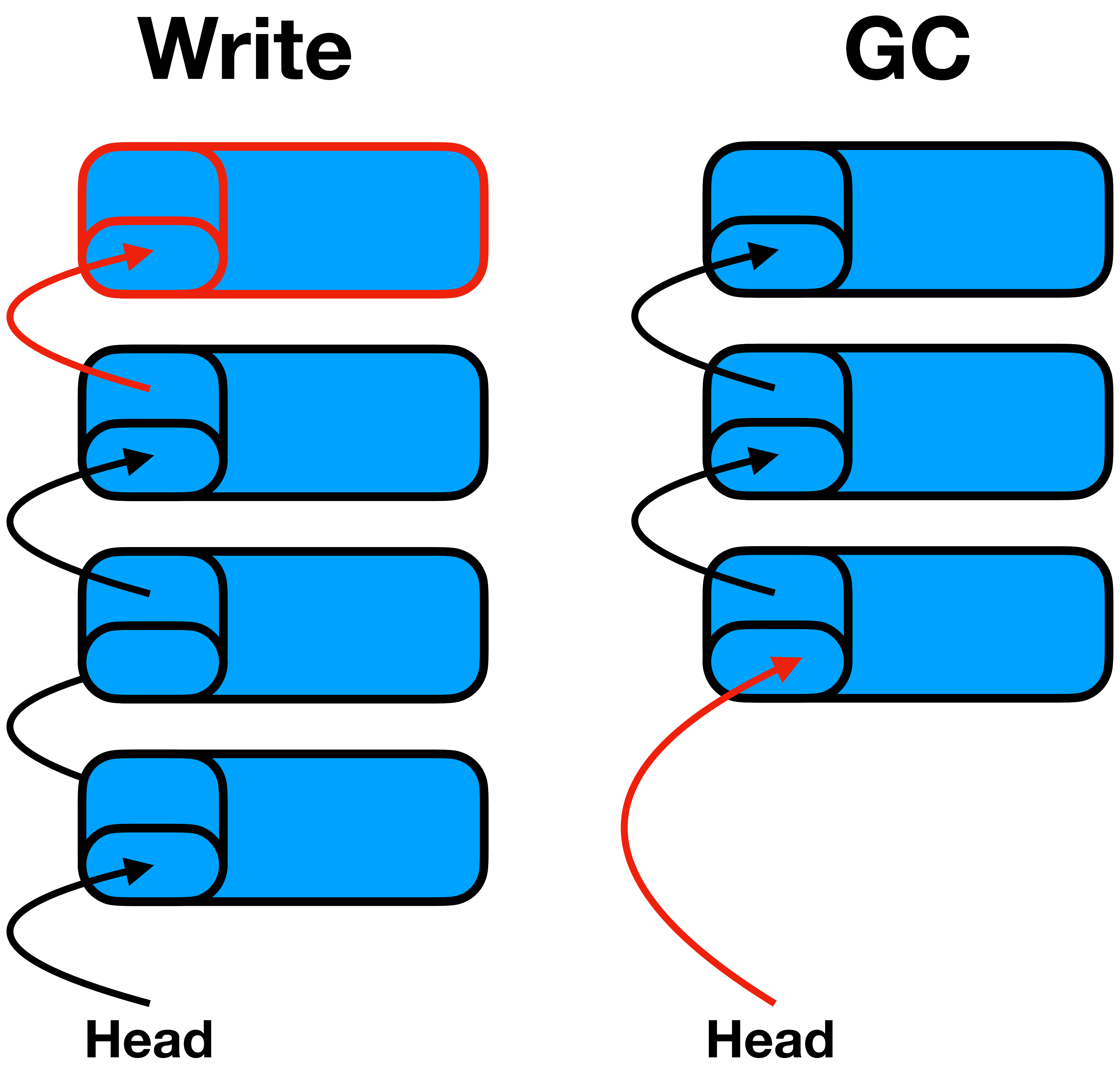
Clover Data Structure



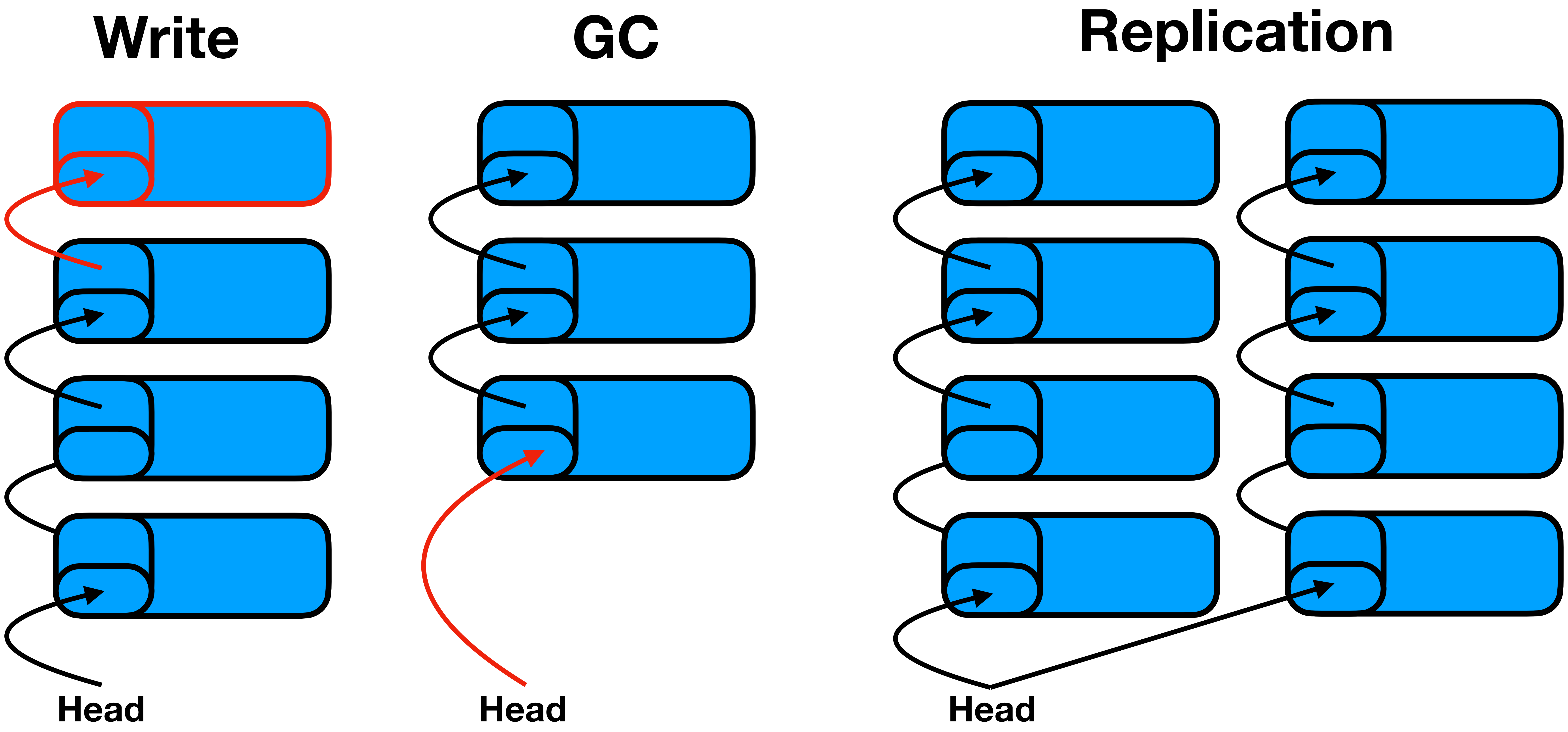
Clover Data Structure



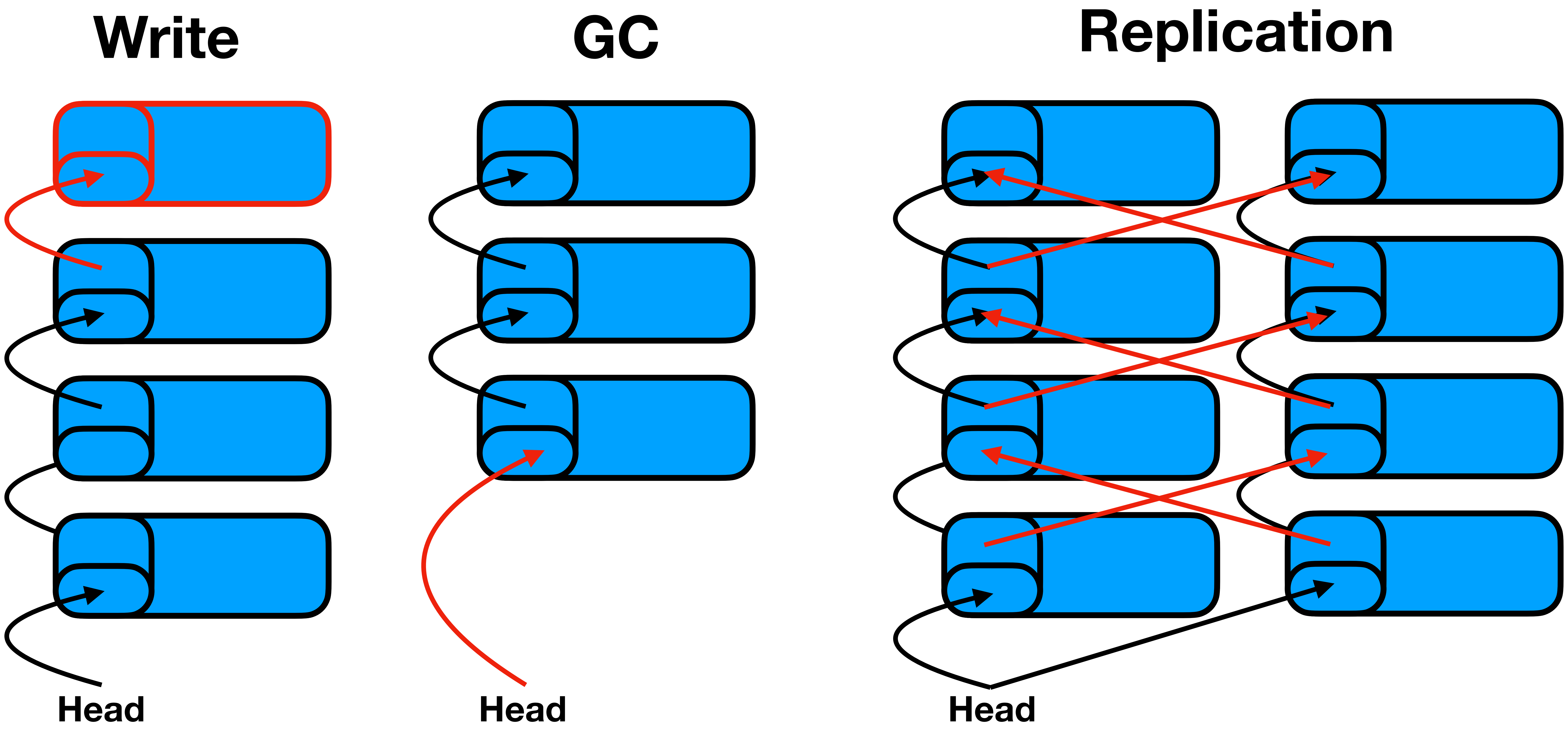
Clover Data Structure



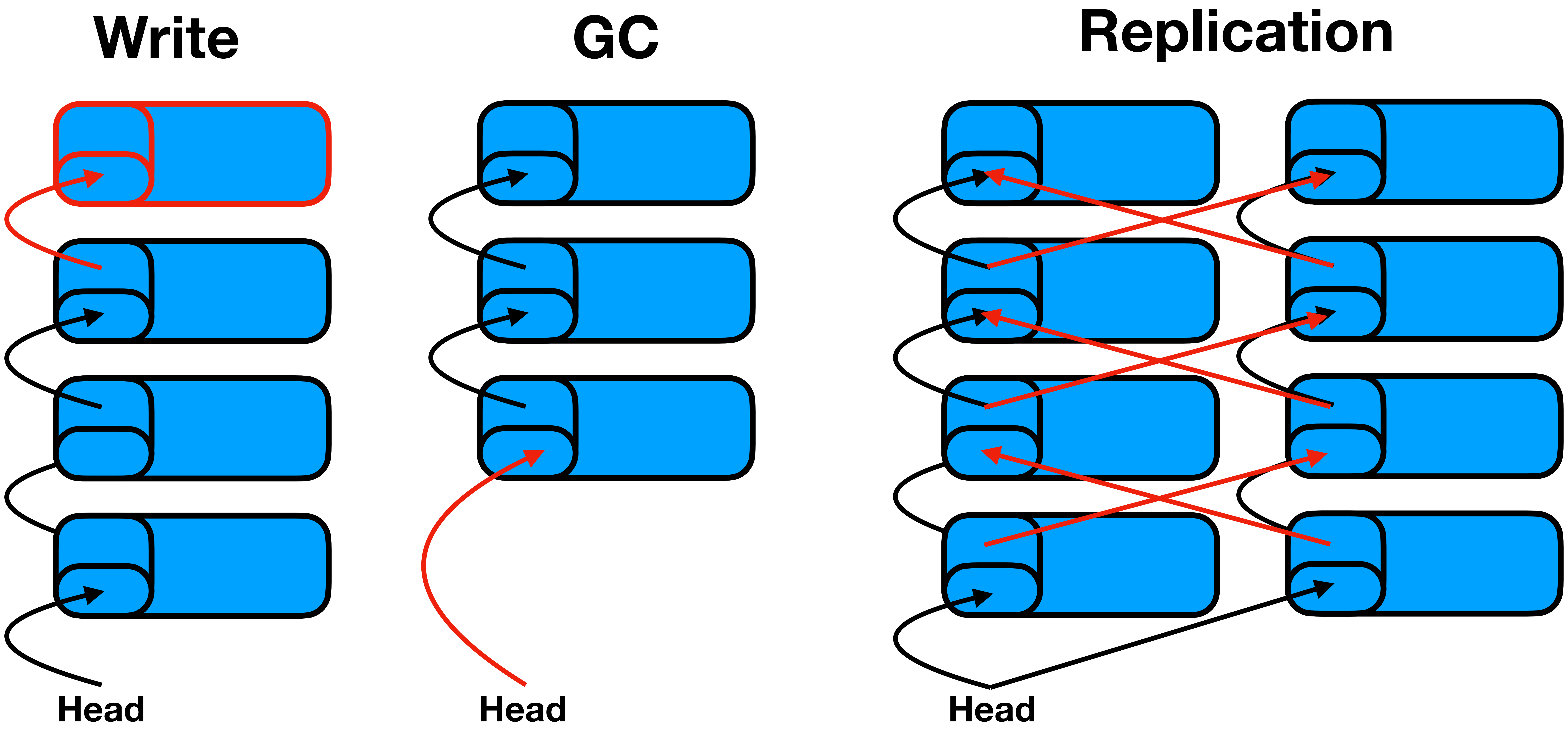
Clover Data Structure



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?