# Scalable Distributed Inverted List Indexes in Disaggregated Memory

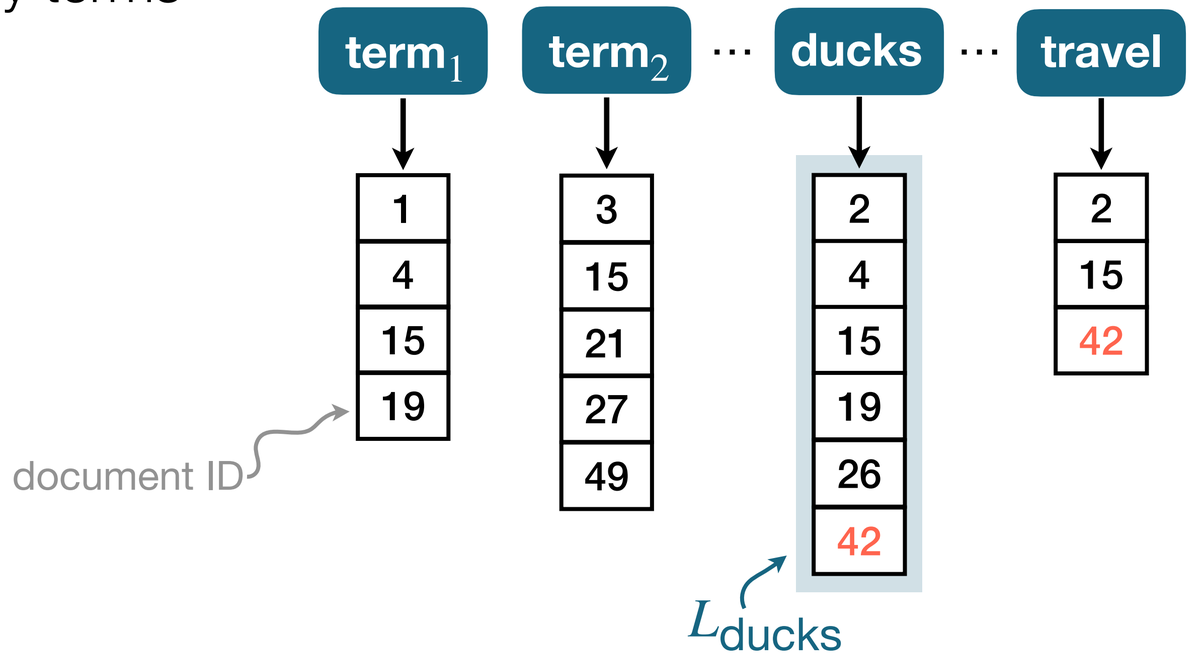
**MANUEL WIDMOSER**
DANIEL KOCHER
NIKOLAUS AUGSTEN

University of Salzburg, Austria
*June 13th, 2024*

# Inverted List Index

A term maps to an *ordered* list of documents that contain this term

**GOAL** Return all documents with common query terms




**APPLICATIONS**

- Information retrieval
- Graph analytics
- DB query processing
- Similarity search

# Inverted List Index

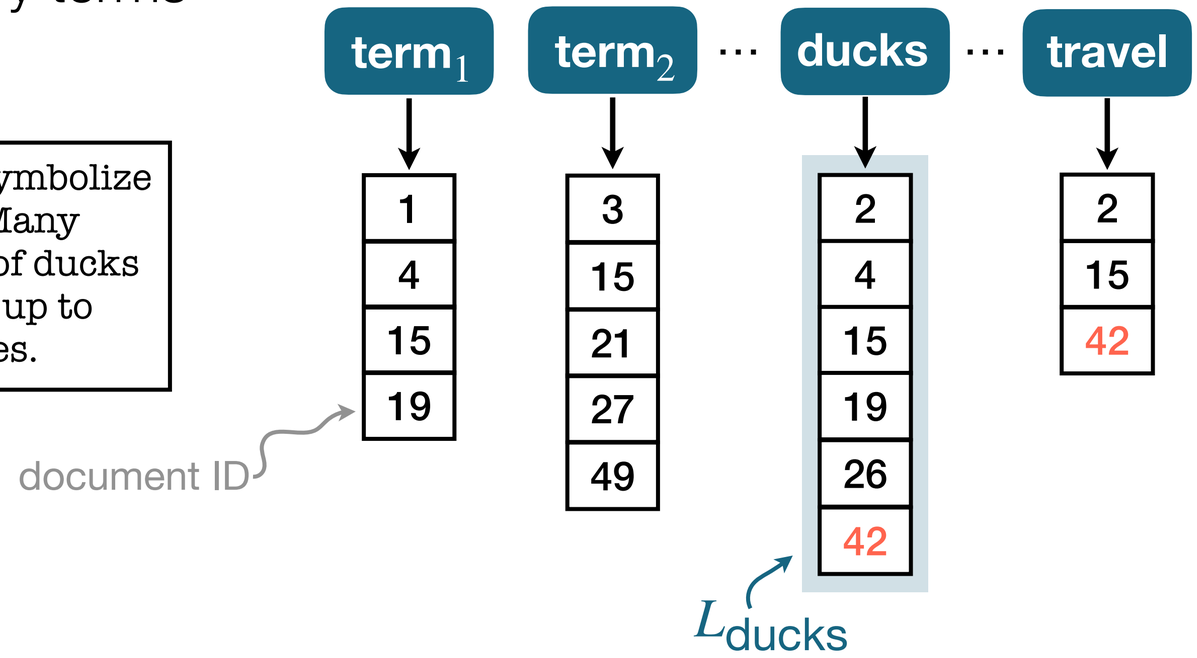
A term maps to an *ordered* list of documents that contain this term

**GOAL** Return all documents with common query terms

- **INSERT**:
  - For each term in document **D**:
    Append D.id to list $L_{\text{term}}$

**doc**$_{42}$

> **Ducks** symbolize **travel**. Many species of ducks migrate up to 800 miles.




**APPLICATIONS**

- Information retrieval
- Graph analytics
- DB query processing
- Similarity search

# Inverted List Index

A term maps to an *ordered* list of documents that contain this term

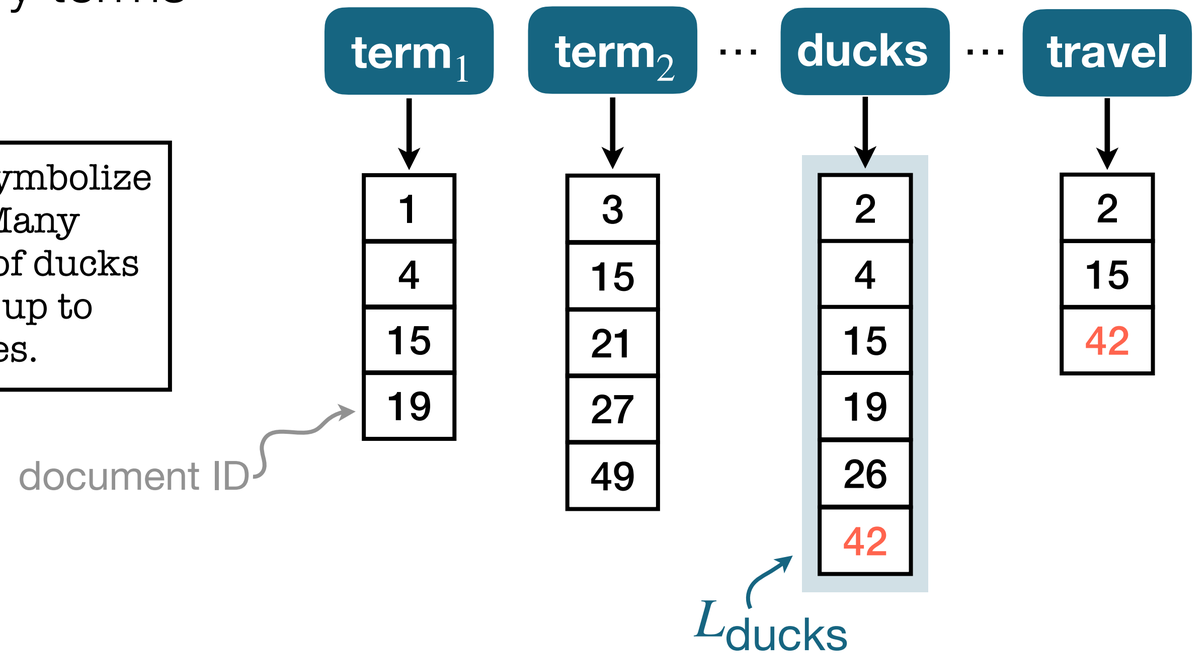
**GOAL** Return all documents with common query terms

- **INSERT**:
  - For each term in document D:
    Append D.id to list $L_{\text{term}}$

- **QUERY**:
  - Query terms: **ducks** **travel**
  - Result: $L_{\text{ducks}} \cap L_{\text{travel}} = \{2,15,42\}$

**doc**$_{42}$

> **Ducks** symbolize **travel**. Many species of ducks migrate up to 800 miles.




**APPLICATIONS**

- Information retrieval
- Graph analytics
- DB query processing
- Similarity search

# Why to Distribute Index Structures?

Two major reasons:

**MEMORY**

- Very large datasets
- Spilling data to secondary (slow) storage is undesirable
  (e.g., index structure in an in-memory DBMS)

# Why to Distribute Index Structures?

Two major reasons:

**MEMORY**

- Very large datasets
- Spilling data to secondary (slow) storage is undesirable
  (e.g., index structure in an in-memory DBMS)

**PERFORMANCE**

- Scale out to massive amount of queries

# Why to Distribute Index Structures?

Two major reasons:

**MEMORY**

- Very large datasets
- Spilling data to secondary (slow) storage is undesirable
  (e.g., index structure in an in-memory DBMS)

**PERFORMANCE**

- Scale out to massive amount of queries

---

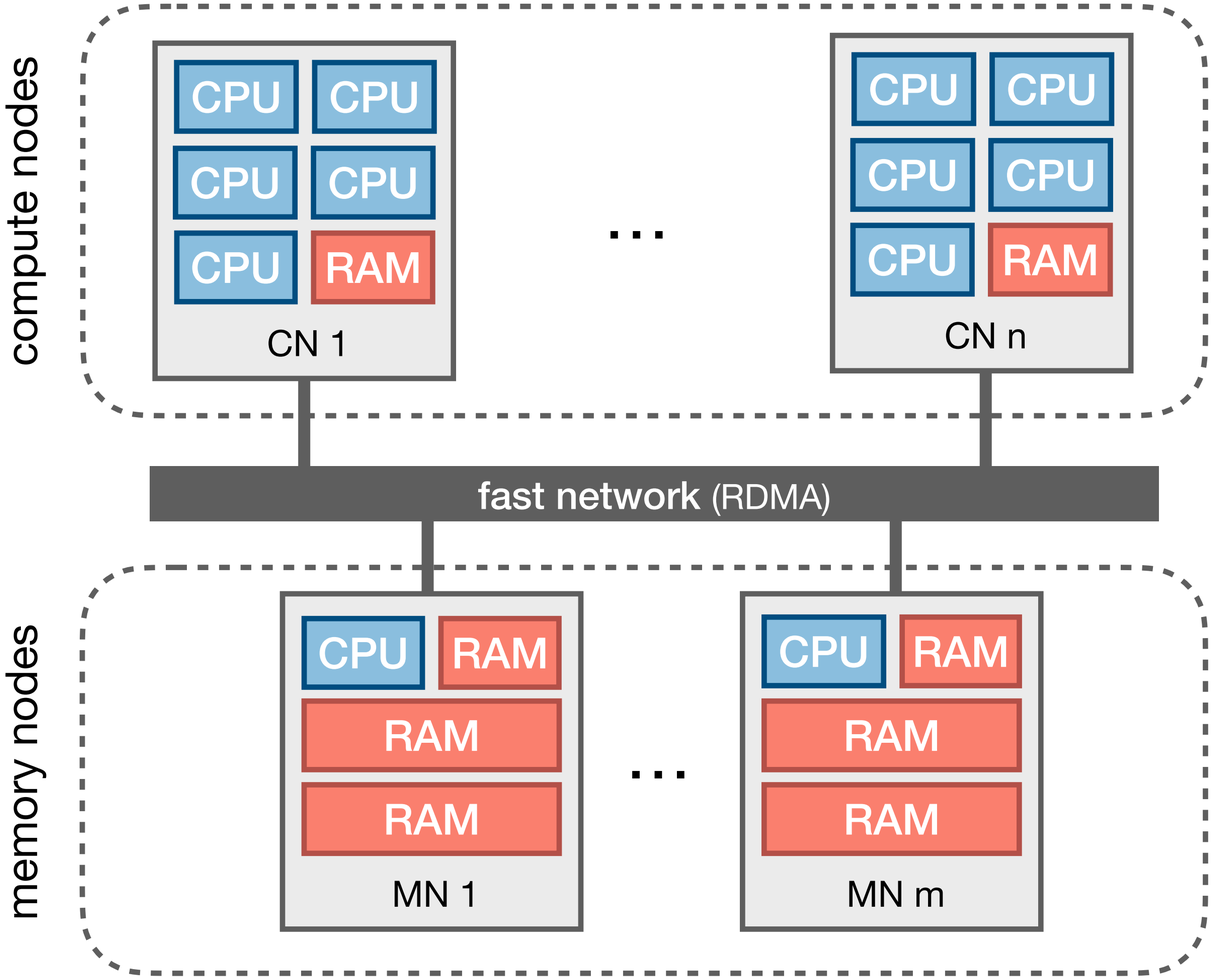
Architecture:

- ⛔ Classic monolithic server architectures underutilize CPU and memory

  CPU and RAM tightly coupled

# Disaggregated Memory (DM)

- High flexibility (elasticity)
- Cost-efficient resource utilization (sustainability)

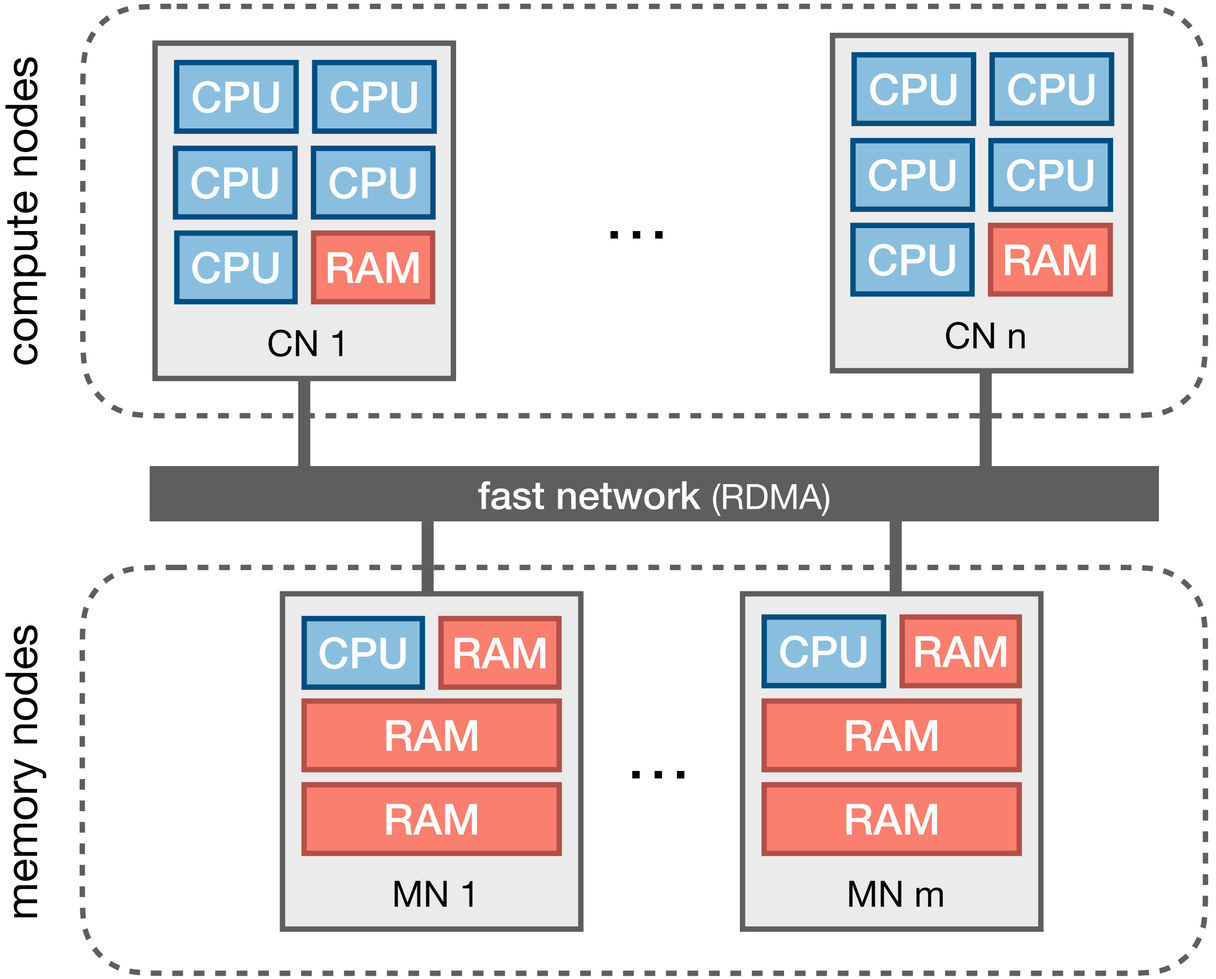


# Disaggregated Memory (DM)

- High flexibility (elasticity)
- Cost-efficient resource utilization (sustainability)

**DESIGN CHALLENGES**

- Data must be accessed over the network
- MNs have near-zero computation power



# Disaggregated Memory (DM)

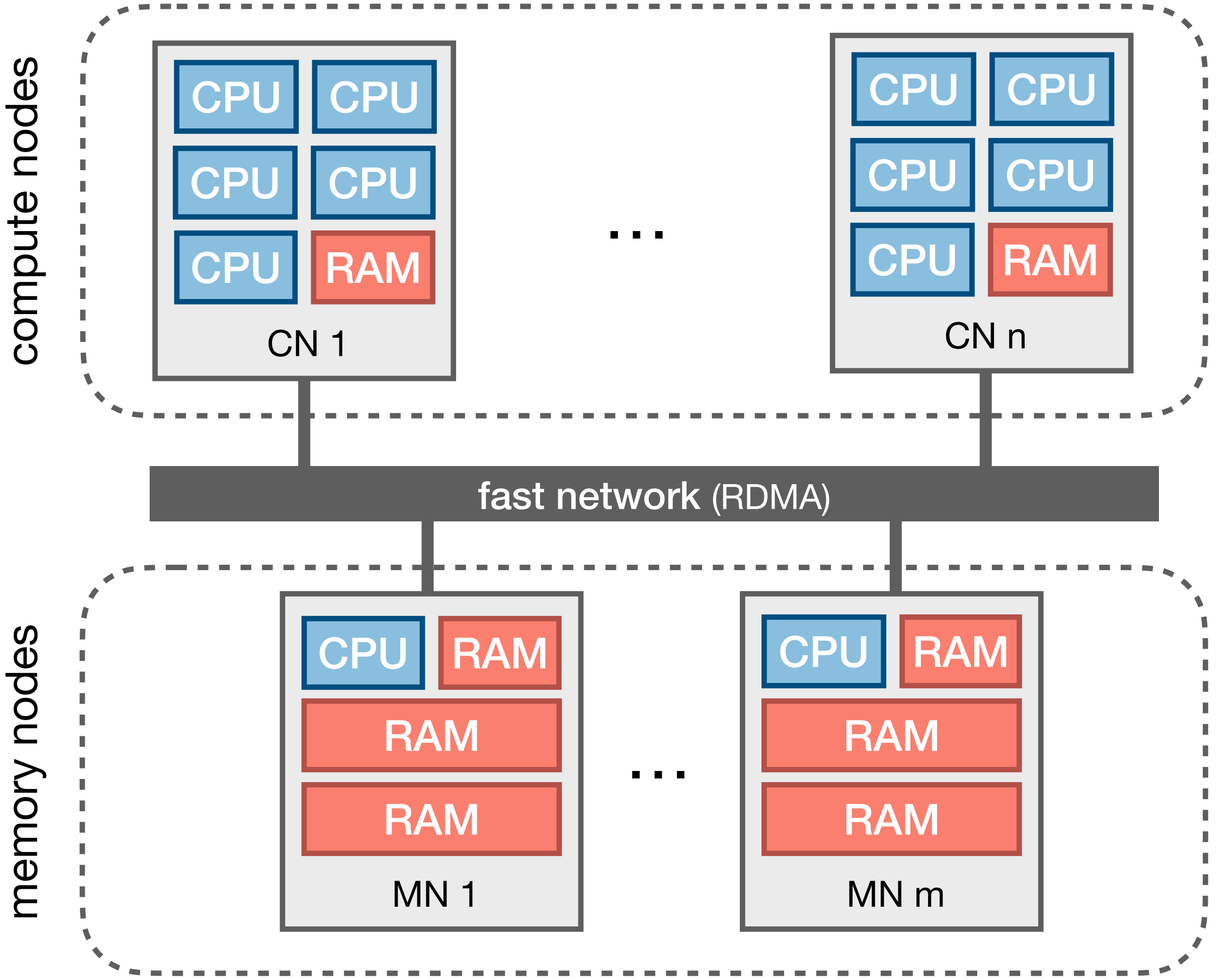
- High flexibility (elasticity)
- Cost-efficient resource utilization (sustainability)

**DESIGN CHALLENGES**

- Data must be accessed over the network
- MNs have near-zero computation power

**RDMA**

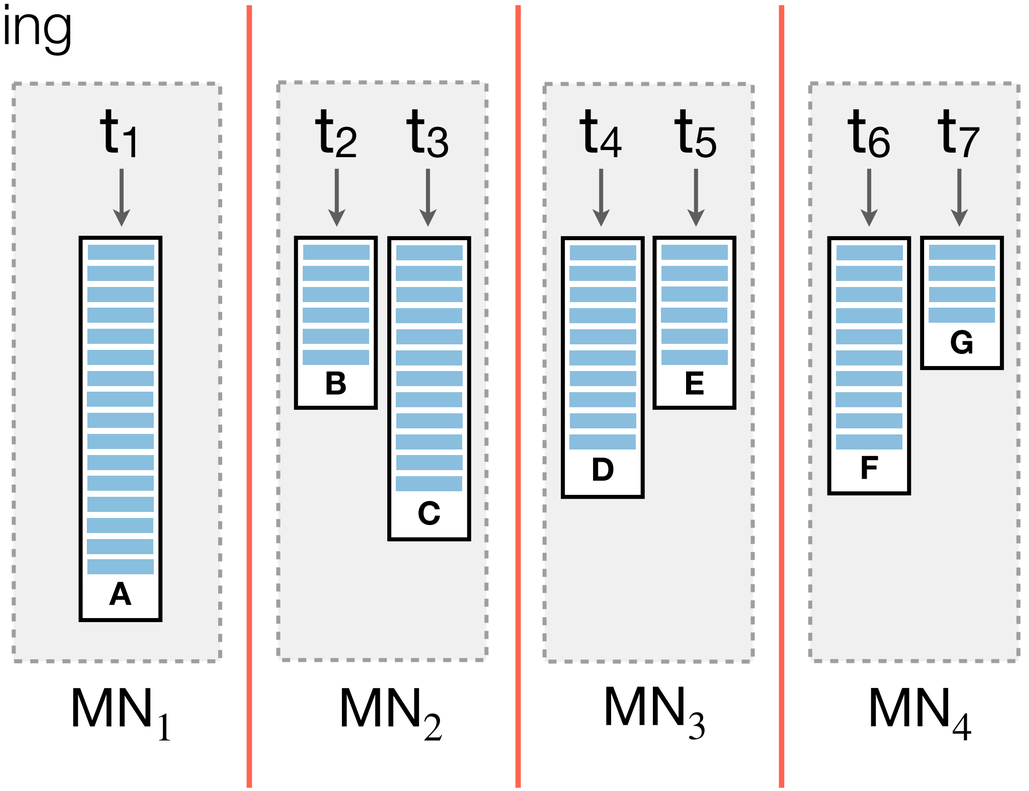
- Fully bypasses the remote CPU
- Low latency (single digit $\mu s$)



# Traditional Distribution Schemes

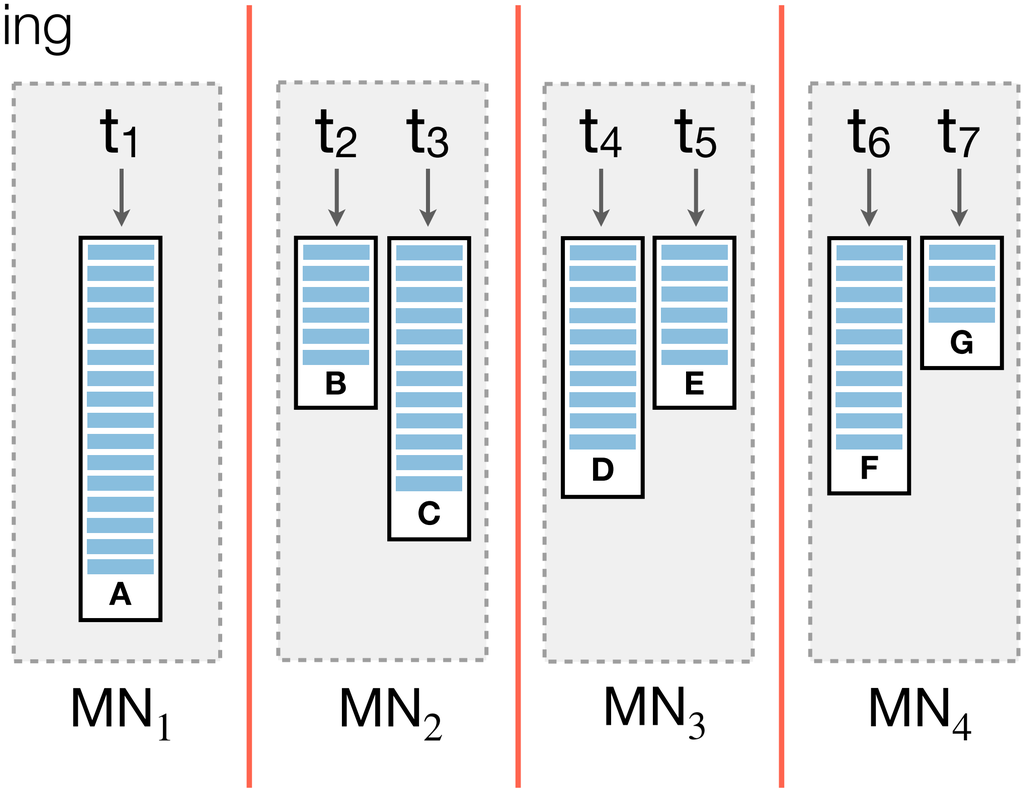
# Method 1: VERTICAL Partitioning

- Distribute complete lists across MNs using load balancing



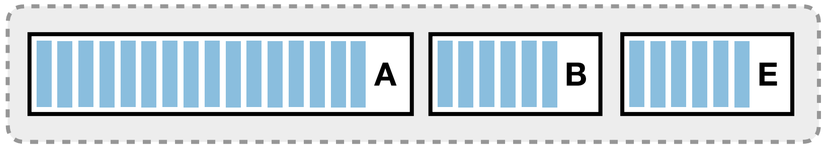
# Method 1: VERTICAL Partitioning

- Distribute complete lists across MNs using load balancing
- Query (on worker):
  - Read lists from remote memory to local buffer
  - Perform list operation




Query: $\{t_1, t_2, t_5\}$

local buffer (on CN; per worker):

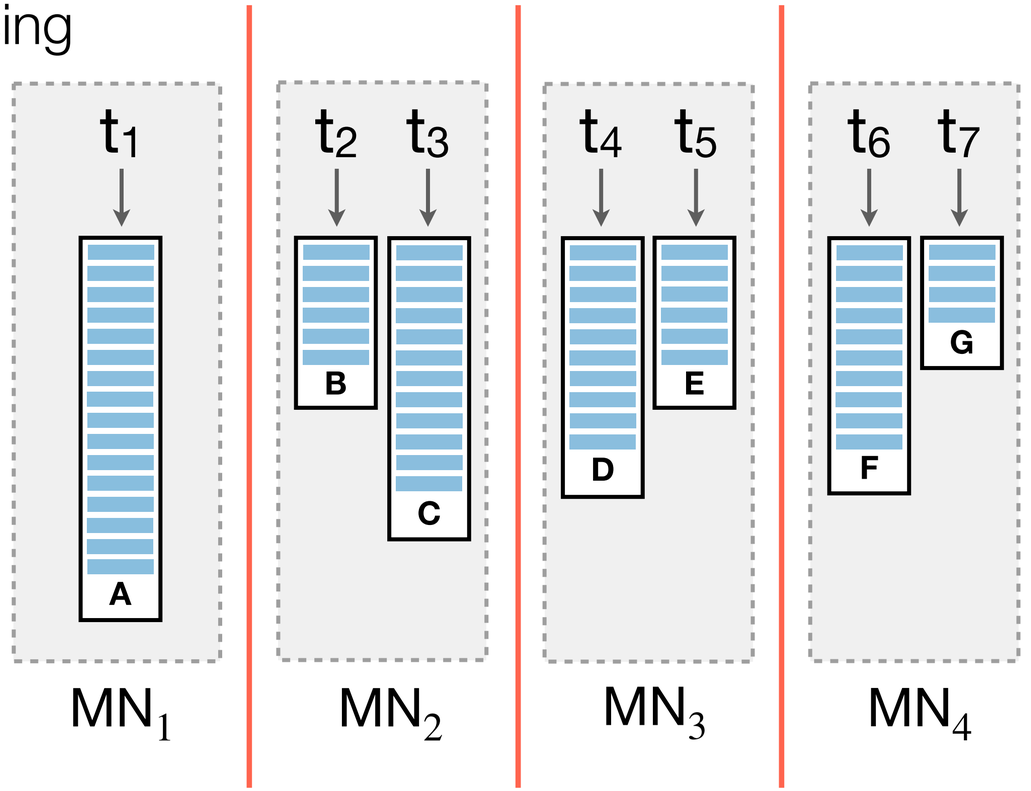


# Method 1: VERTICAL Partitioning

- Distribute complete lists across MNs using load balancing
- Query (on worker):
  - Read lists from remote memory to local buffer
  - Perform list operation

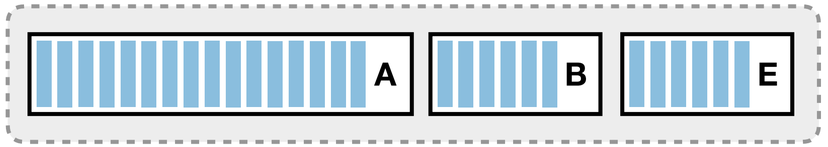
**SCALABILITY CHALLENGES**

**C1** Network latency (waiting for long list)




Query: $\{t_1, t_2, t_5\}$

local buffer (on CN; per worker):

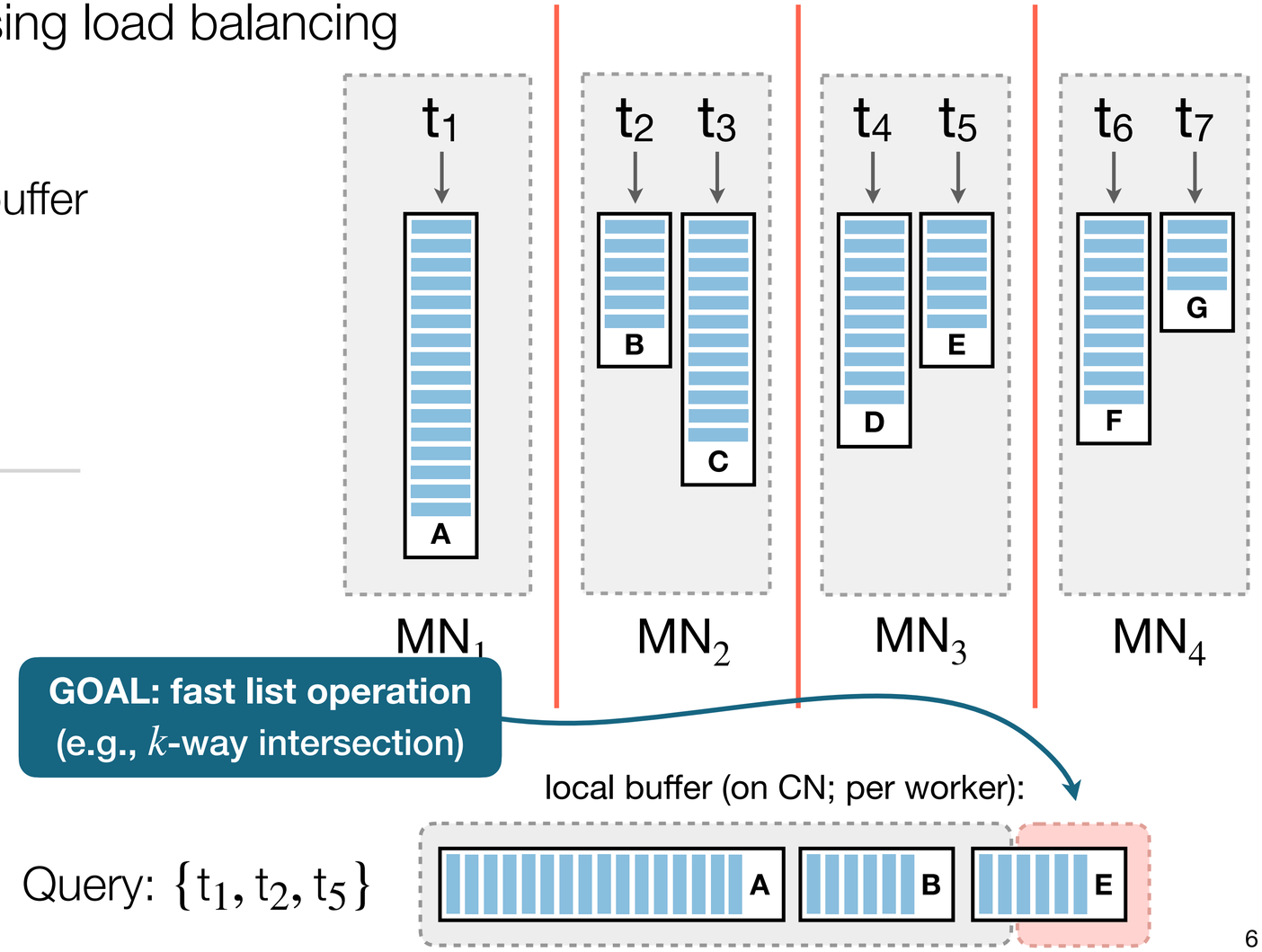


# Method 1: VERTICAL Partitioning

- Distribute complete lists across MNs using load balancing
- Query (on worker):
  - Read lists from remote memory to local buffer
  - Perform list operation

**SCALABILITY CHALLENGES**

- C1 Network latency (waiting for long list)
- C2 Limited memory on CNs



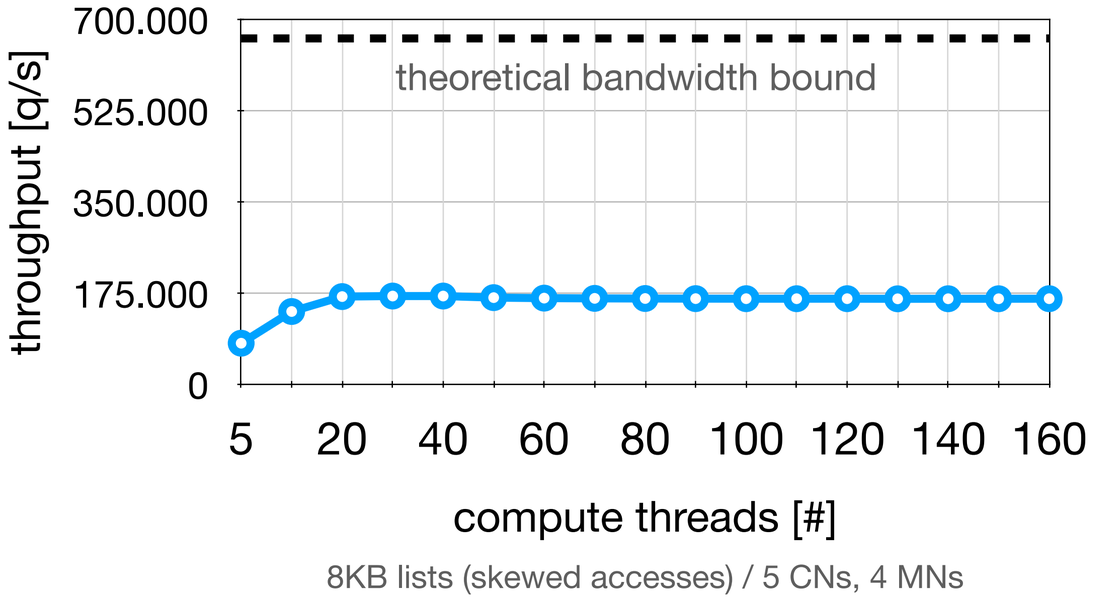
# Method 1: VERTICAL Partitioning

- Distribute complete lists across MNs using load balancing
- Query (on worker):
  - Read lists from remote memory to local buffer
  - Perform list operation

**SCALABILITY CHALLENGES**

- C1 Network latency (waiting for long list)
- C2 Limited memory on CNs
- C3 Access skew

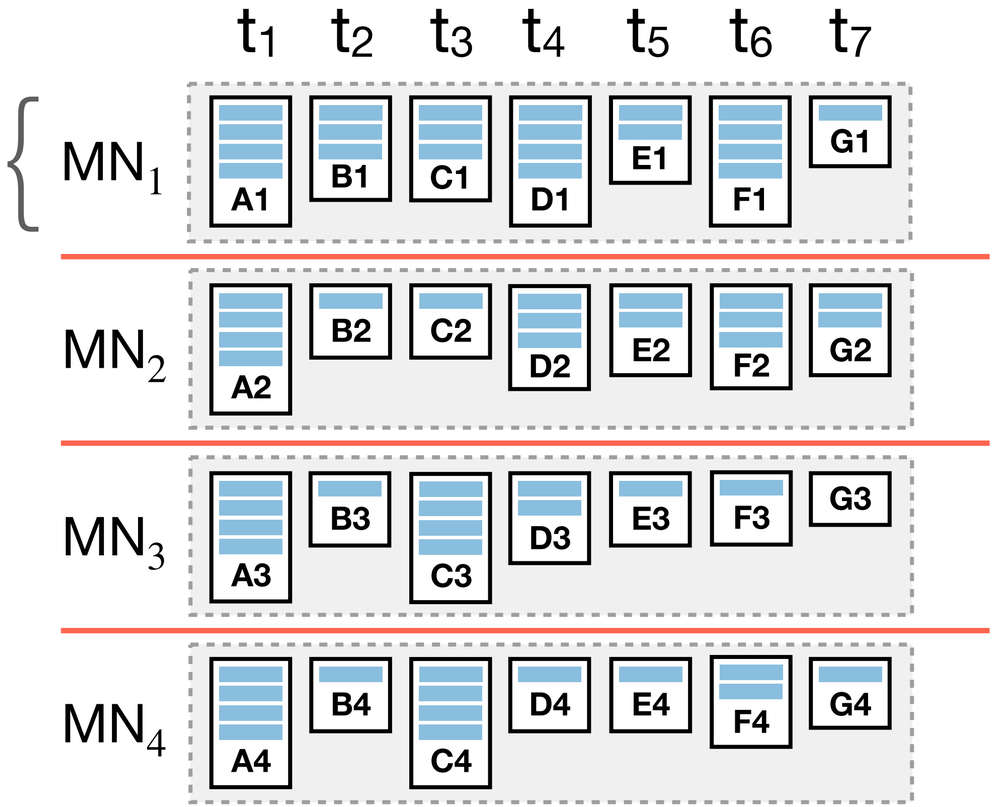
(higher is better)




8KB lists (skewed accesses) / 5 CNs, 4 MNs

# Method 2: HORIZONTAL Partitioning

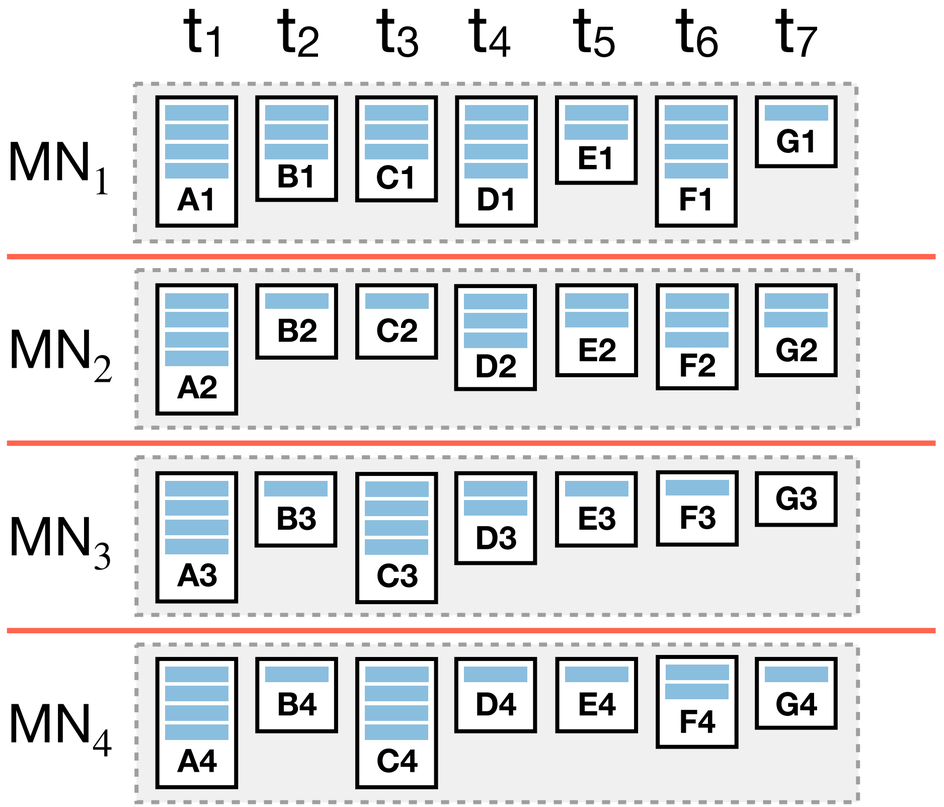
- Assign a range of documents to a dedicated MN



# Method 2: HORIZONTAL Partitioning

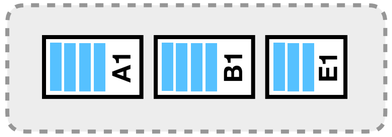
- Assign a range of documents to a dedicated MN
- Query (on worker):

  For each MN:
  - Read partial lists to local buffer
  - Perform list operation




local buffer (on CN; per worker):

Query: $\{t_1, t_2, t_5\}$



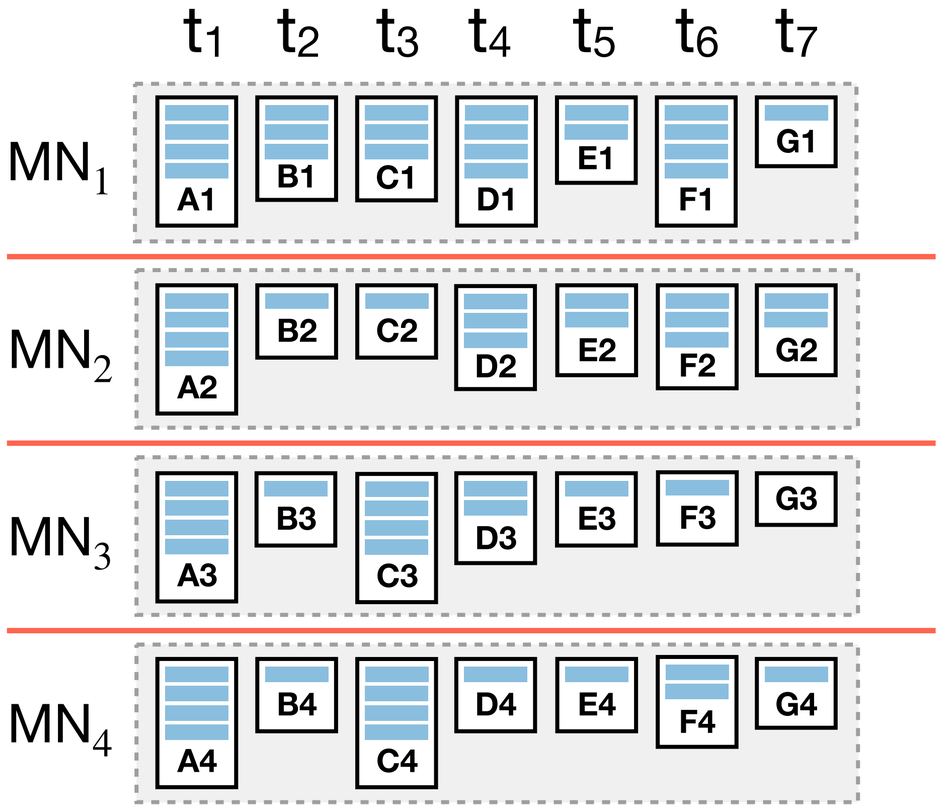
# Method 2: HORIZONTAL Partitioning

- Assign a range of documents to a dedicated MN
- **Query** (on worker):

  For each **MN**:
  - Read partial lists to local buffer
  - Perform list operation

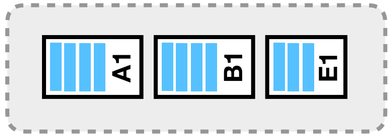
**SCALABILITY CHALLENGES**

- C1 Network latency
- C2 Limited memory on CNs
- ~~C3 Access skew~~




local buffer (on CN; per worker):

Query: $\{t_1, t_2, t_5\}$



# Method 2: HORIZONTAL Partitioning

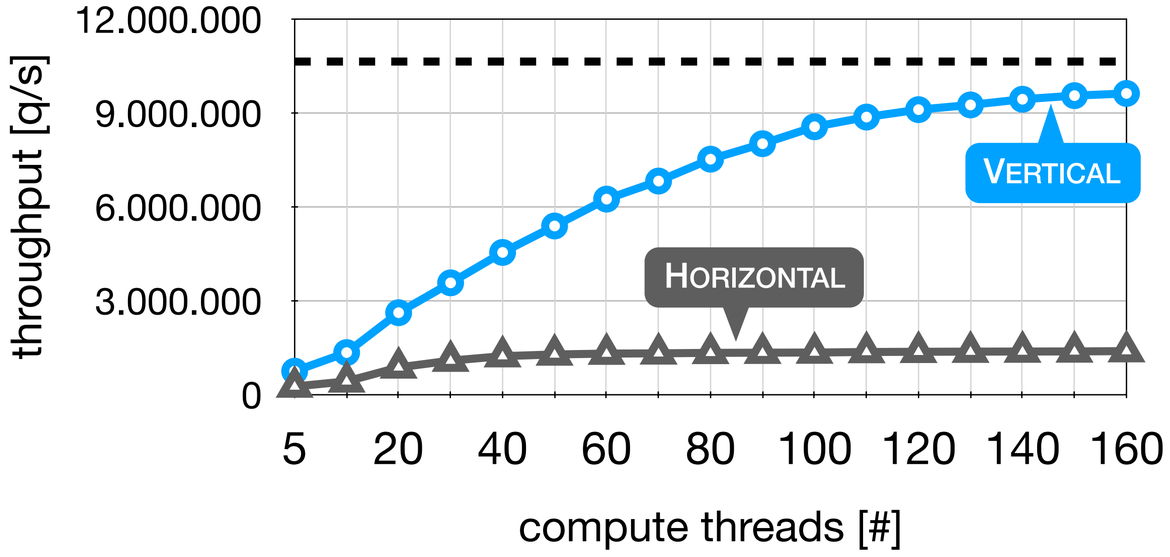
- Assign a range of documents to a dedicated **MN**
- **Query** (on worker):

  For each **MN**:
  - Read partial lists to local buffer
  - Perform list operation

**SCALABILITY CHALLENGES**

- C1 Network latency
- C2 Limited memory on CNs
- ~~C3 Access skew~~
- C4 Network roundtrips

(higher is better)

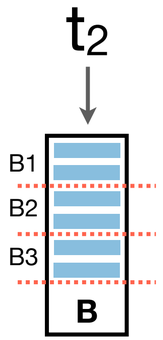



512B lists (uniform accesses) / 5 CNs, 4 MNs

# Block-based Scheme for Inverted Lists in DM

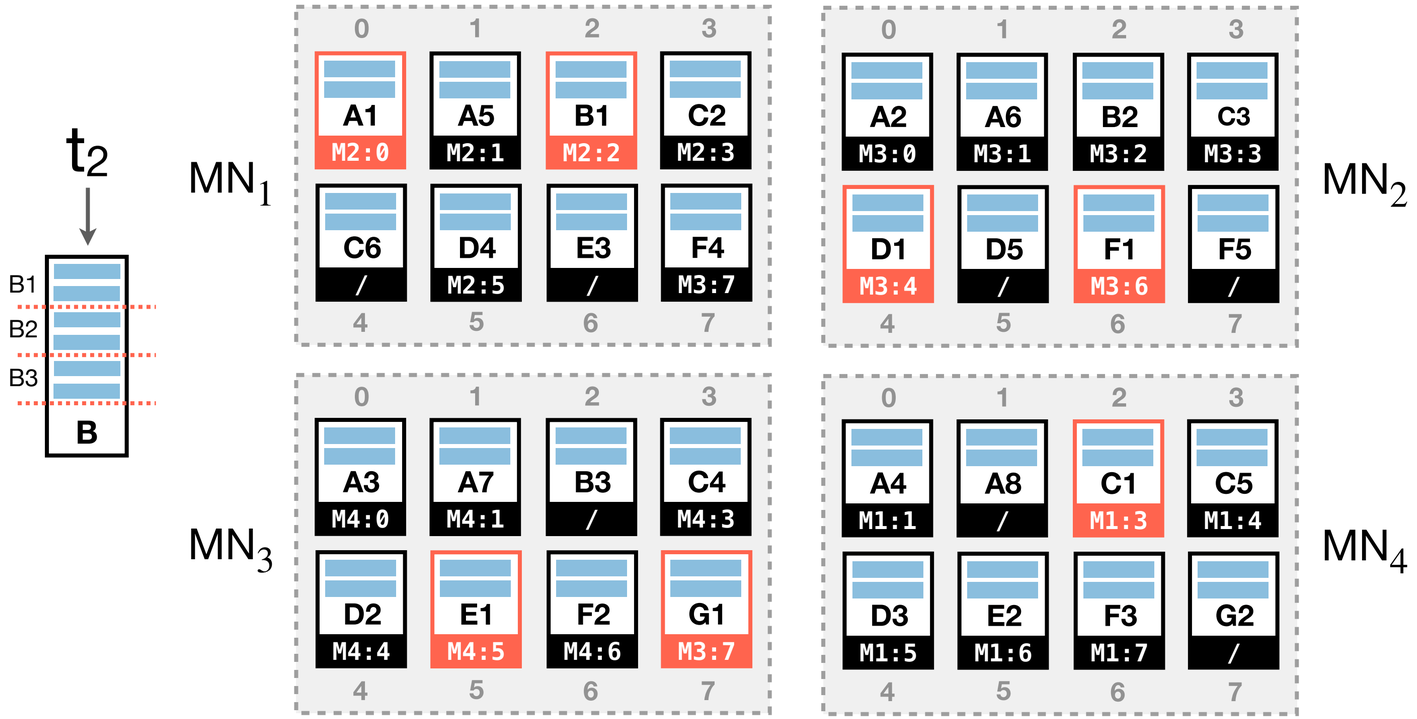
# Block-based Partitioning

- Divide lists into **fix-sized blocks**



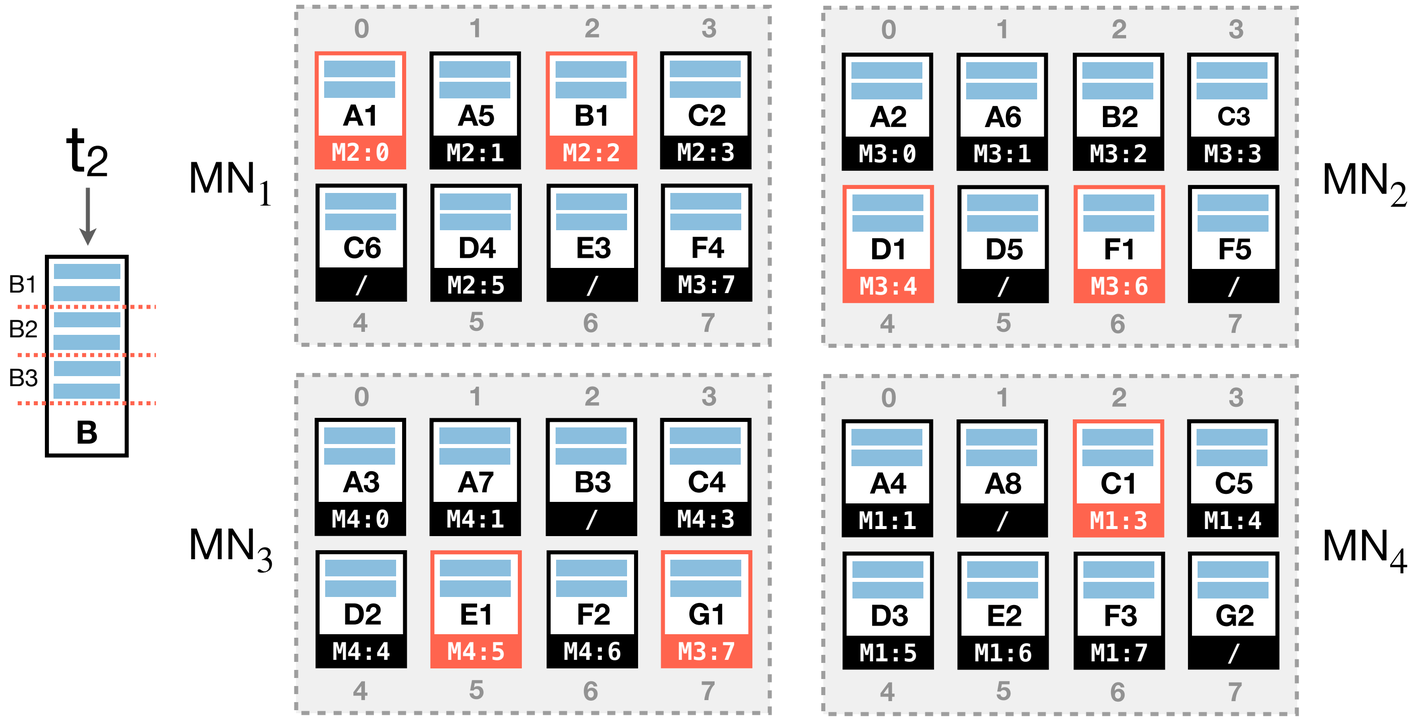
# Block-based Partitioning

- Divide lists into **fix-sized blocks**
- Distribute blocks among all **MNs**
  (e.g., round-robin, hash-based, …)



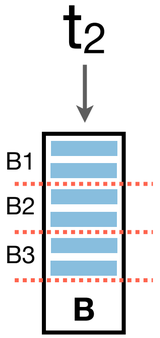
# Block-based Partitioning

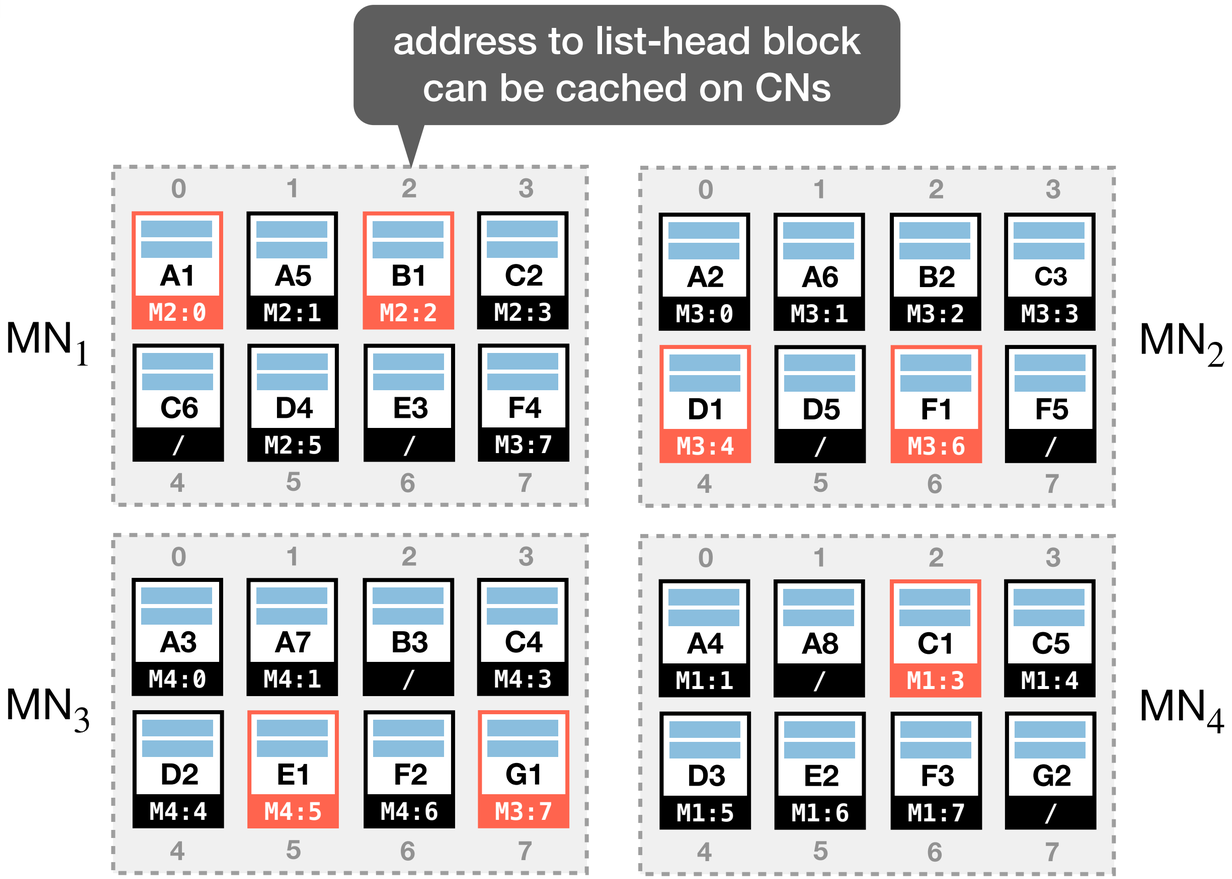
- Divide lists into **fix-sized blocks**
- Distribute blocks among all **MNs**
  (e.g., round-robin, hash-based, …)
- Connect blocks via remote pointers



# Block-based Partitioning

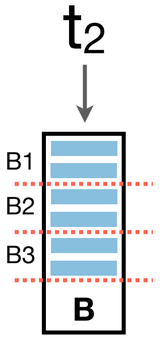
- Divide lists into **fix-sized blocks**
- Distribute blocks among all **MNs**
  (e.g., round-robin, hash-based, …)
- Connect blocks via remote pointers

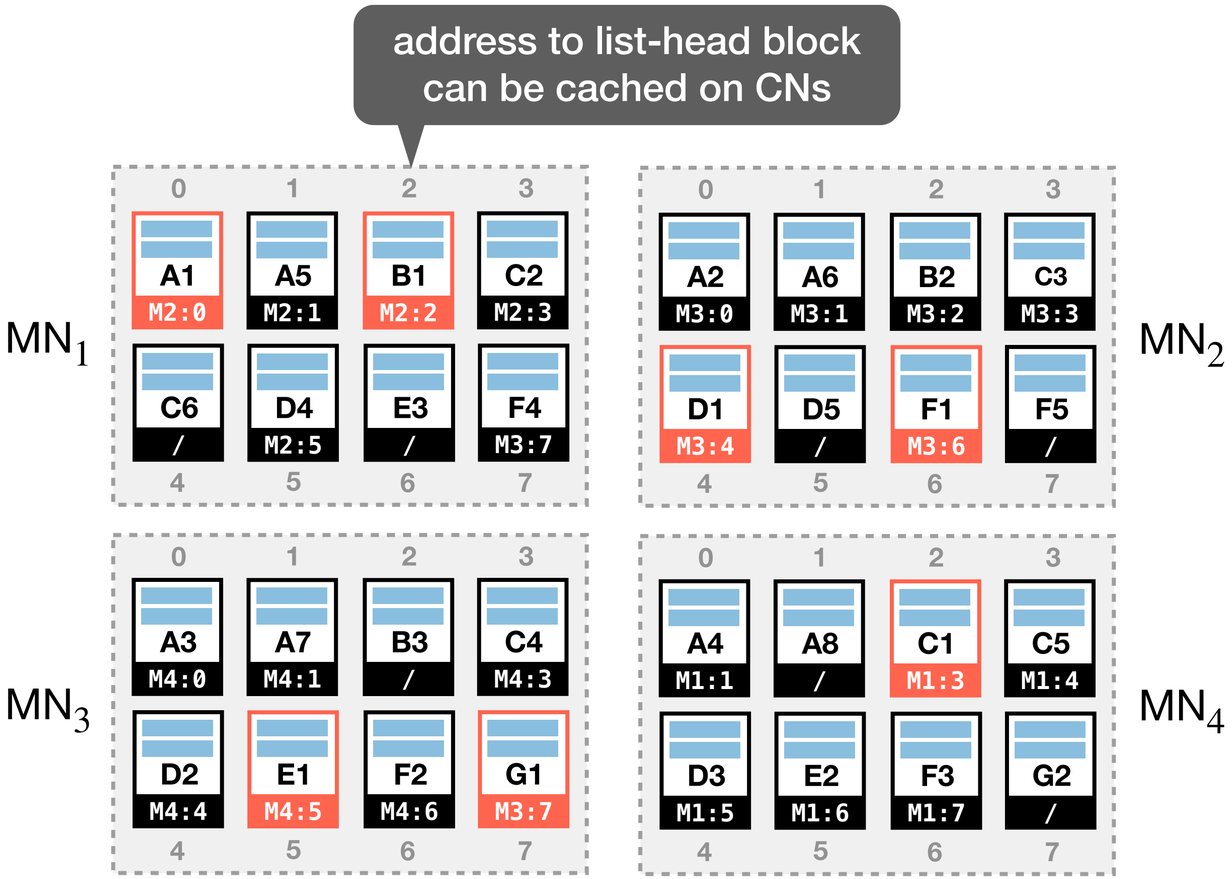


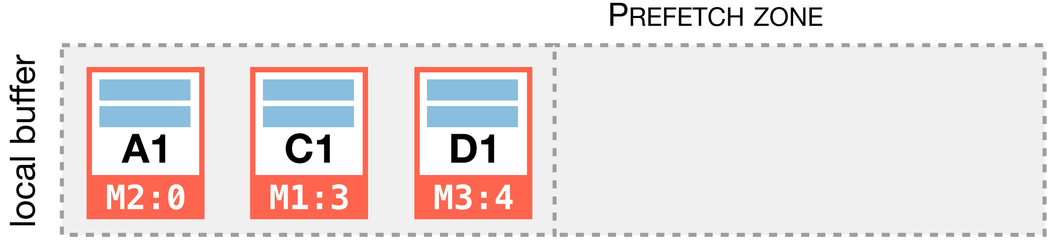



# Block-based Partitioning

- Divide lists into **fix-sized blocks**
- Distribute blocks among all **MNs**
  (e.g., round-robin, hash-based, …)
- Connect blocks via remote pointers
- **Query** (on worker):
  1. Read list-head blocks to local buffer

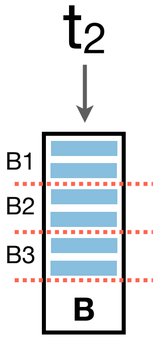


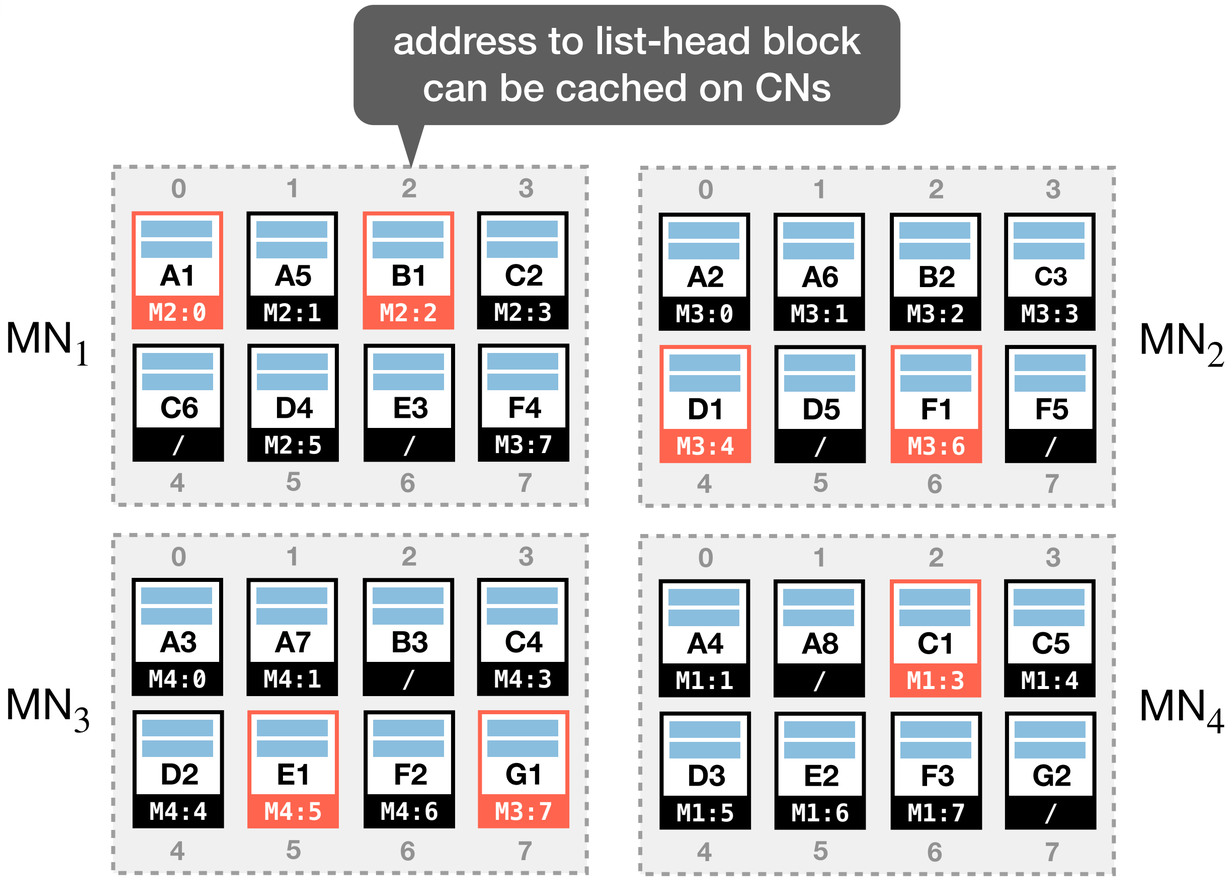



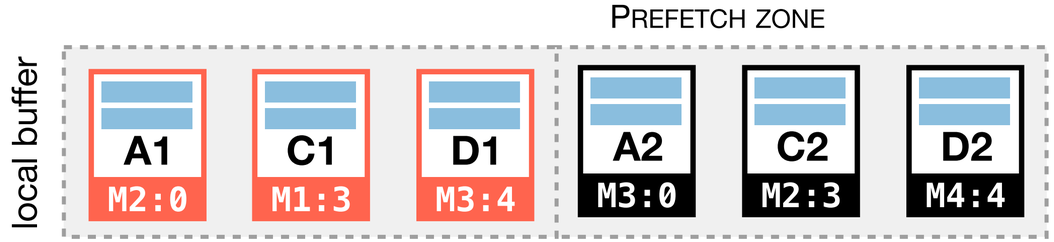



# Block-based Partitioning

- Divide lists into **fix-sized blocks**
- Distribute blocks among all **MNs**
  (e.g., round-robin, hash-based, …)
- Connect blocks via remote pointers
- **Query** (on worker):
  1. Read list-head blocks to local buffer
  2. Prefetch subsequent blocks (issue `RDMA_READ`)

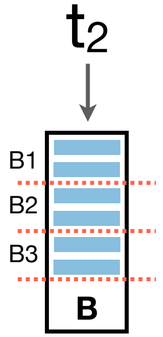


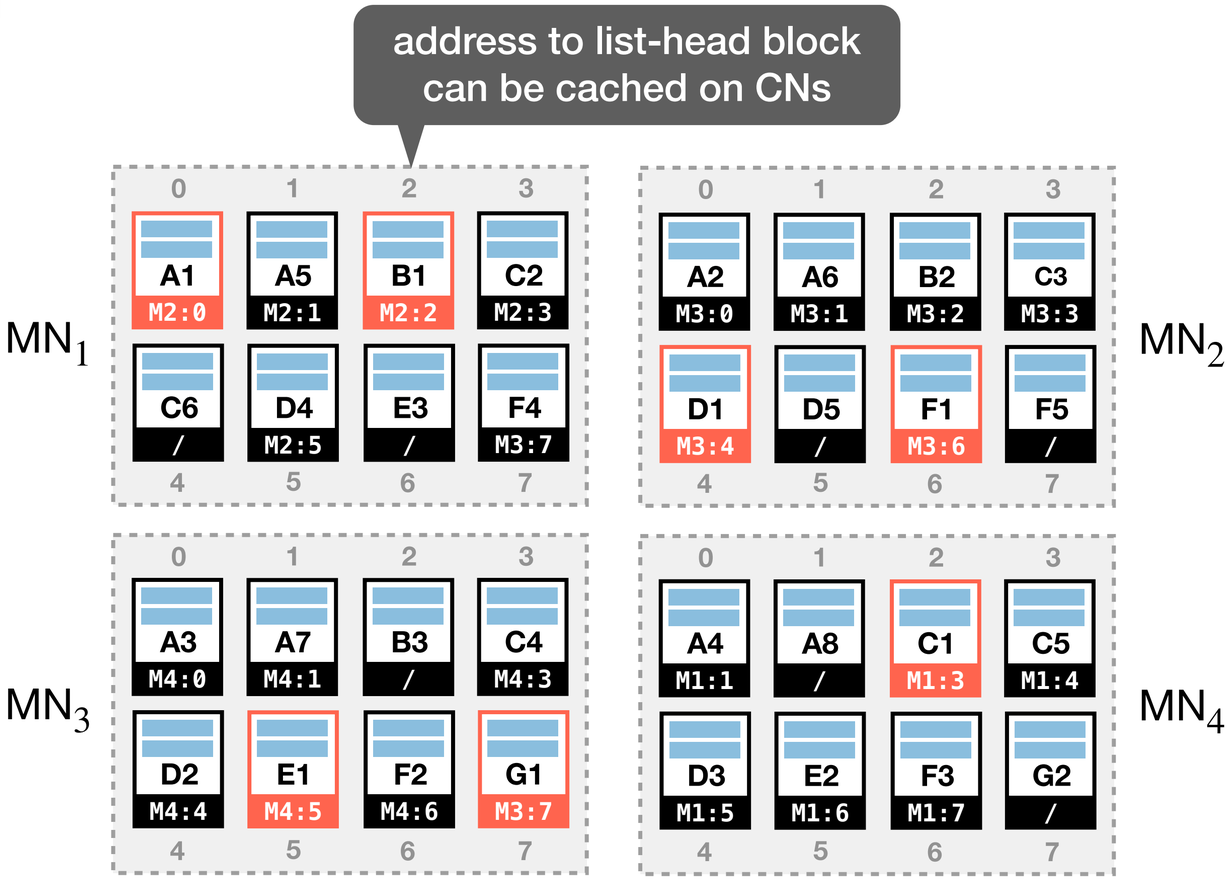



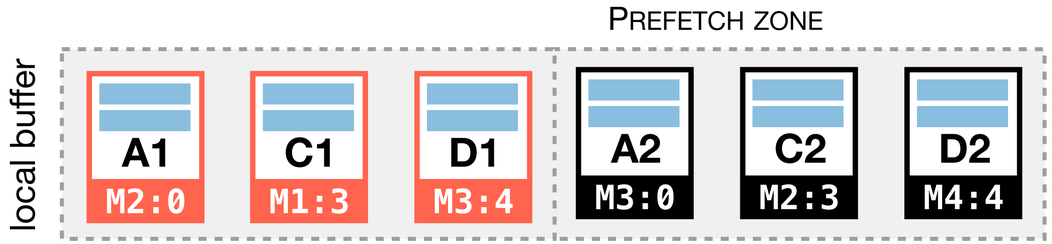



# Block-based Partitioning

- Divide lists into **fix-sized blocks**
- Distribute blocks among all **MNs**
  (e.g., round-robin, hash-based, …)
- Connect blocks via remote pointers
- **Query** (on worker):
  1. Read list-head blocks to local buffer
  2. Prefetch subsequent blocks (issue `RDMA_READ`)
  3. Perform the list operation — on block switch, go to 2



# Addressing Scalability Challenges

**C1** — **NETWORK LATENCY**

Network accesses and list operations are fully interleaved

# Addressing Scalability Challenges

**C1** — **Network Latency**

Network accesses and list operations are fully interleaved

**C2** — **Limited Memory on CN**

Local buffer size is independent of the data (list sizes)

# Addressing Scalability Challenges

**C1** — **Network Latency**

Network accesses and list operations are fully interleaved

**C2** — **Limited Memory on CN**

Local buffer size is independent of the data (list sizes)

**C3** — **Access Skew**

Skewed workloads do not overload individual MNs
(due to the fine granularity of blocks)

# Addressing Scalability Challenges

**C1** — **Network Latency**

Network accesses and list operations are fully interleaved

**C2** — **Limited Memory on CN**

Local buffer size is independent of the data (list sizes)

**C3** — **Access Skew**

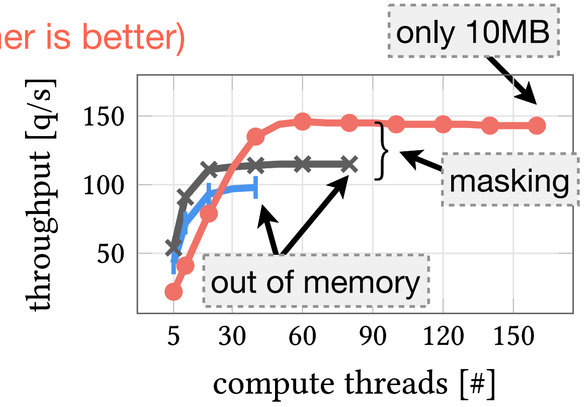
Skewed workloads do not overload individual MNs
(due to the fine granularity of blocks)

**C4** — **Network Roundtrips**

Number of roundtrips is minimal for very small lists
(otherwise, latency is masked with list operation)

# Scalability Experiments

(higher is better)




**(a) CCNEWS**
(information retrieval)

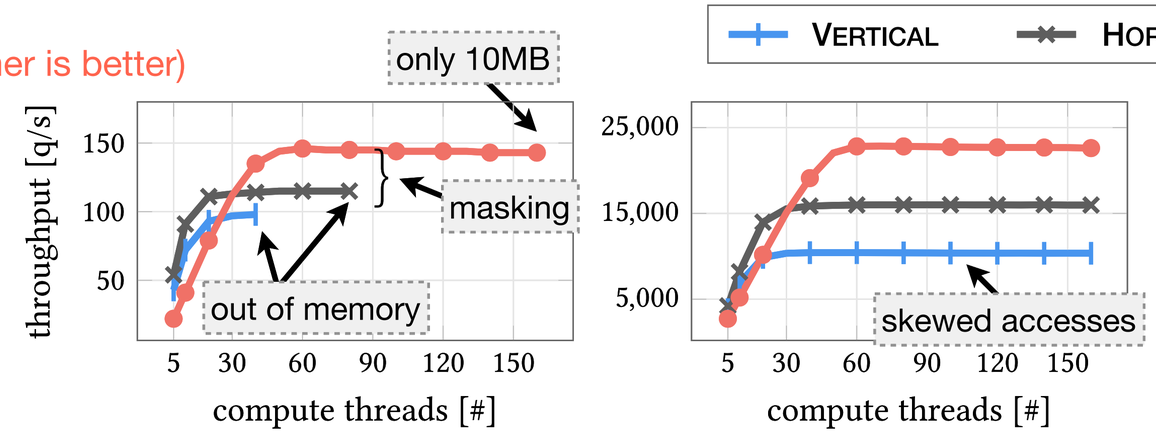
average query size: 6


**SETUP**:

- 5 CNs (32 threads / 10GB RAM)
- 4 MNs (1 thread / 96GB RAM)
- Block size: 1KB

# Scalability Experiments

(higher is better)




**(a) CCNEWS**
(information retrieval)

average query size: 6



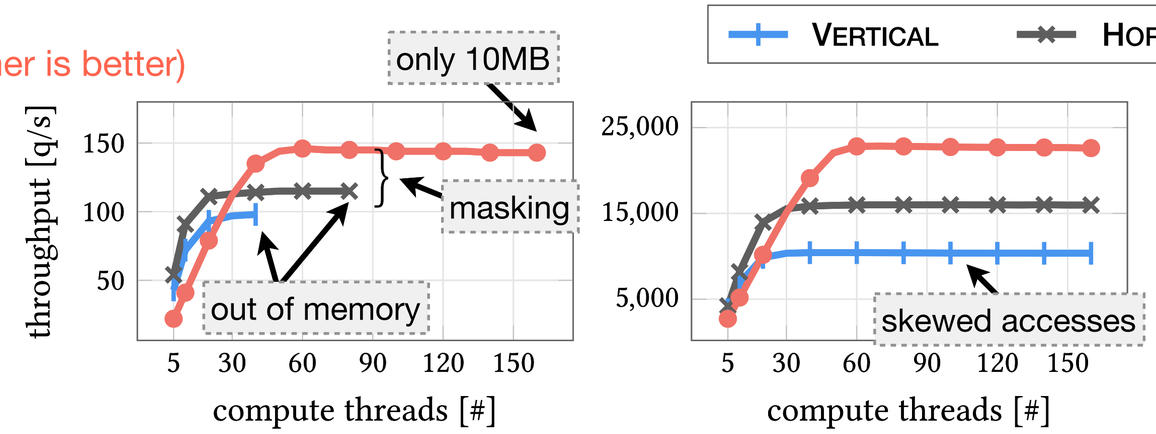

**(b) TWITTER**
(graph analytics)

**SETUP**:

- 5 CNs (32 threads / 10GB RAM)
- 4 MNs (1 thread / 96GB RAM)
- Block size: 1KB

# Scalability Experiments

(higher is better)

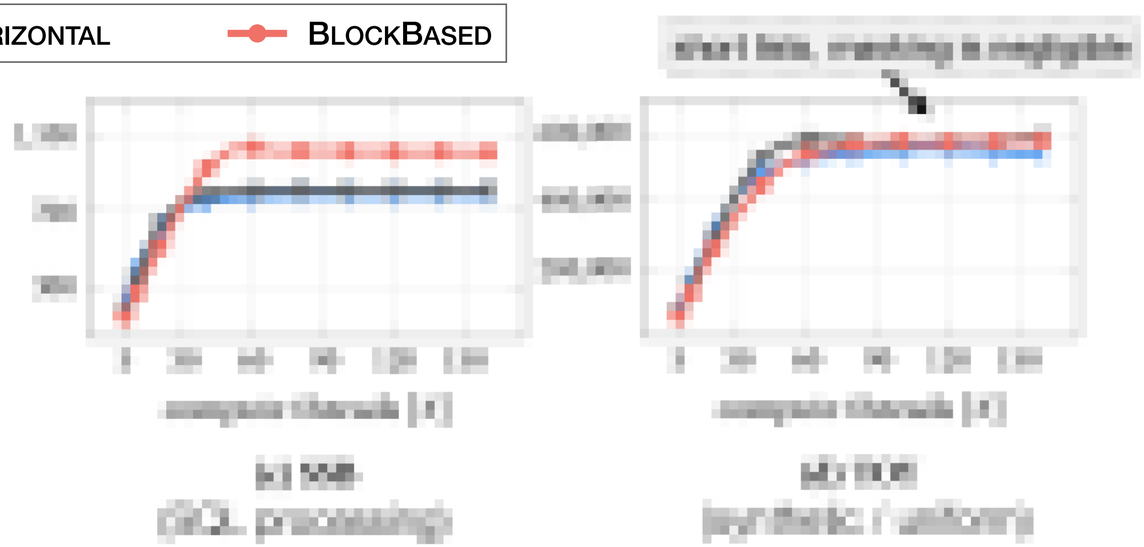



**(a) CCNEWS**
(information retrieval)




**(b) TWITTER**
(graph analytics)

average query size: 6



MORE RESULTS IN THE PAPER

**SETUP**:

- 5 CNs (32 threads / 10GB RAM)
- 4 MNs (1 thread / 96GB RAM)
- Block size: 1KB

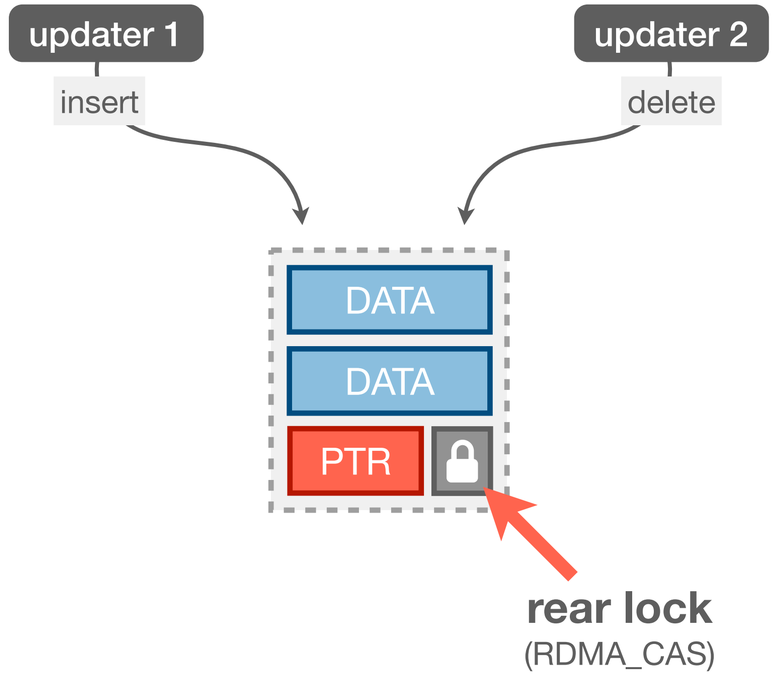
# Incremental Index Updates

# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks



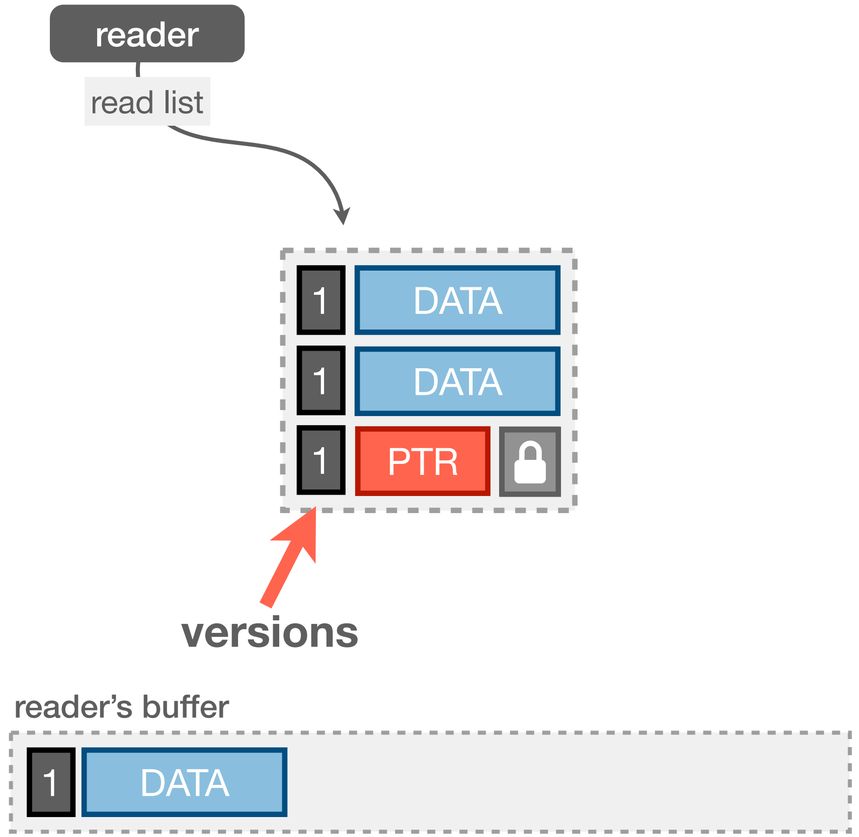
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)

OPTIMISTIC CONCURRENCY CONTROL



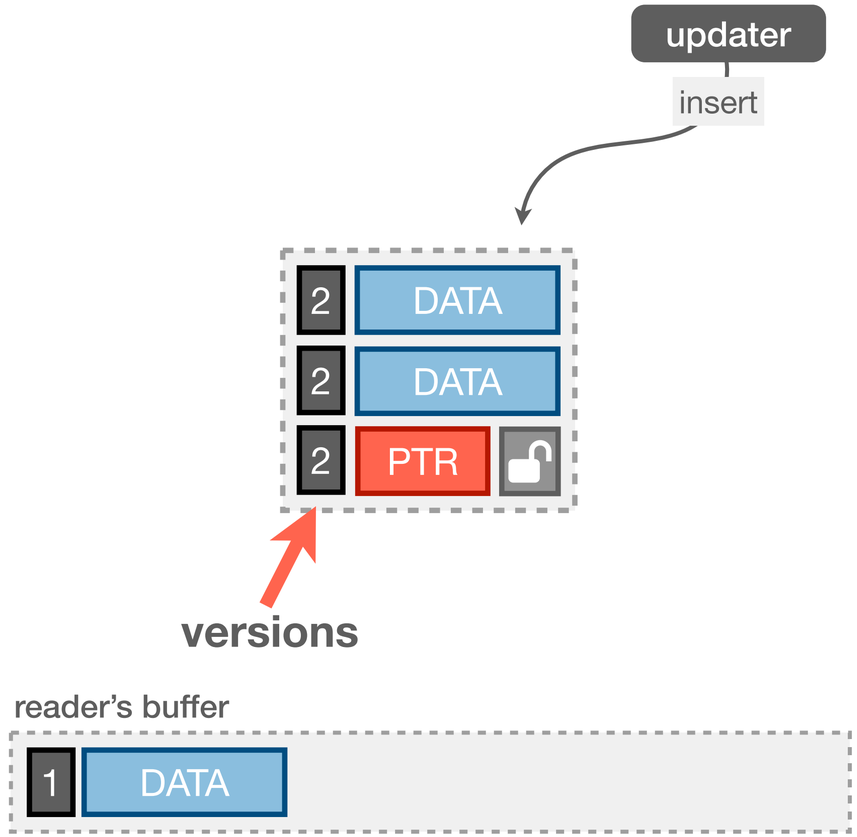
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)

OPTIMISTIC CONCURRENCY CONTROL



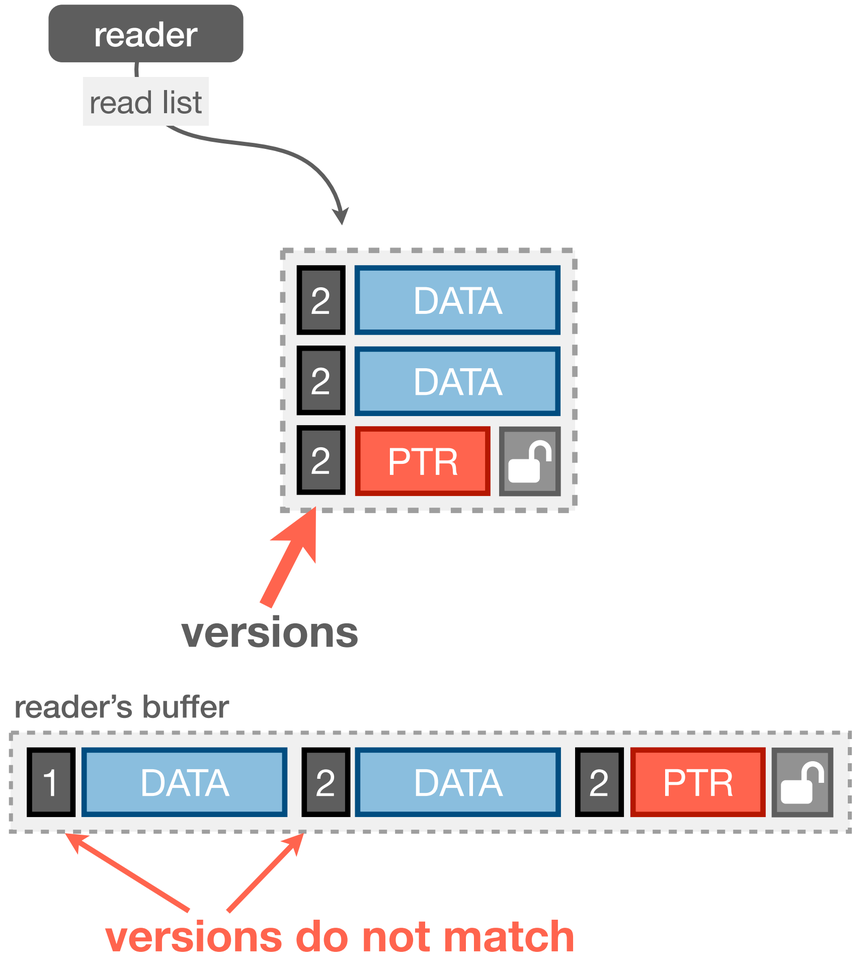
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)

OPTIMISTIC CONCURRENCY CONTROL



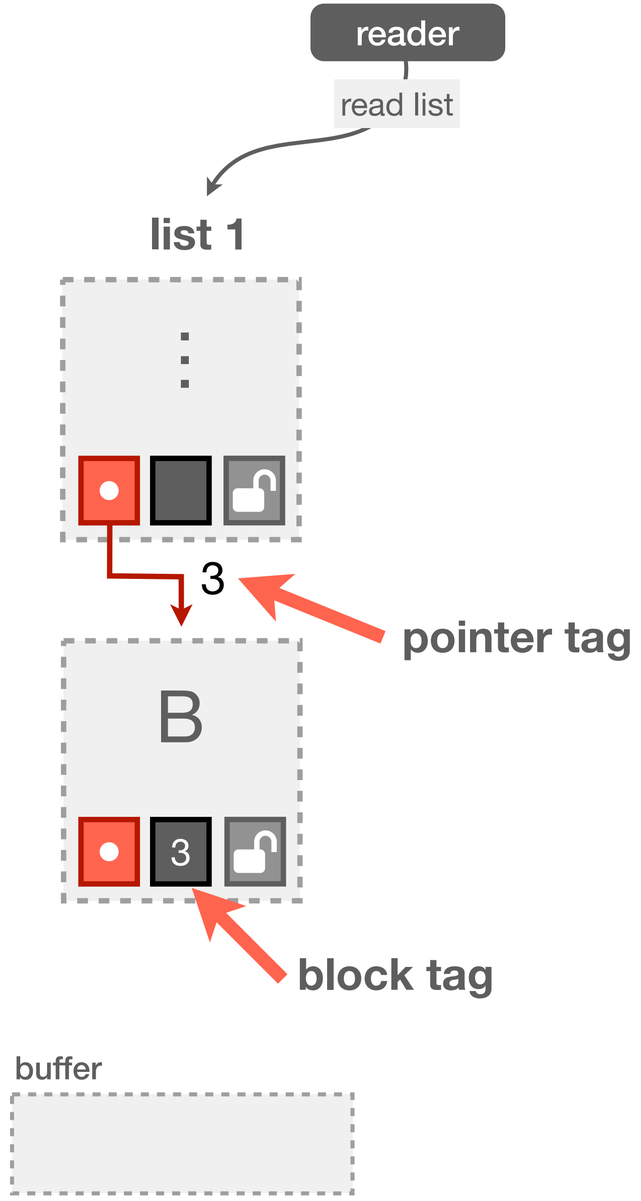
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)
- Block re-allocations
  - Pointer/block tagging

OPTIMISTIC CONCURRENCY CONTROL



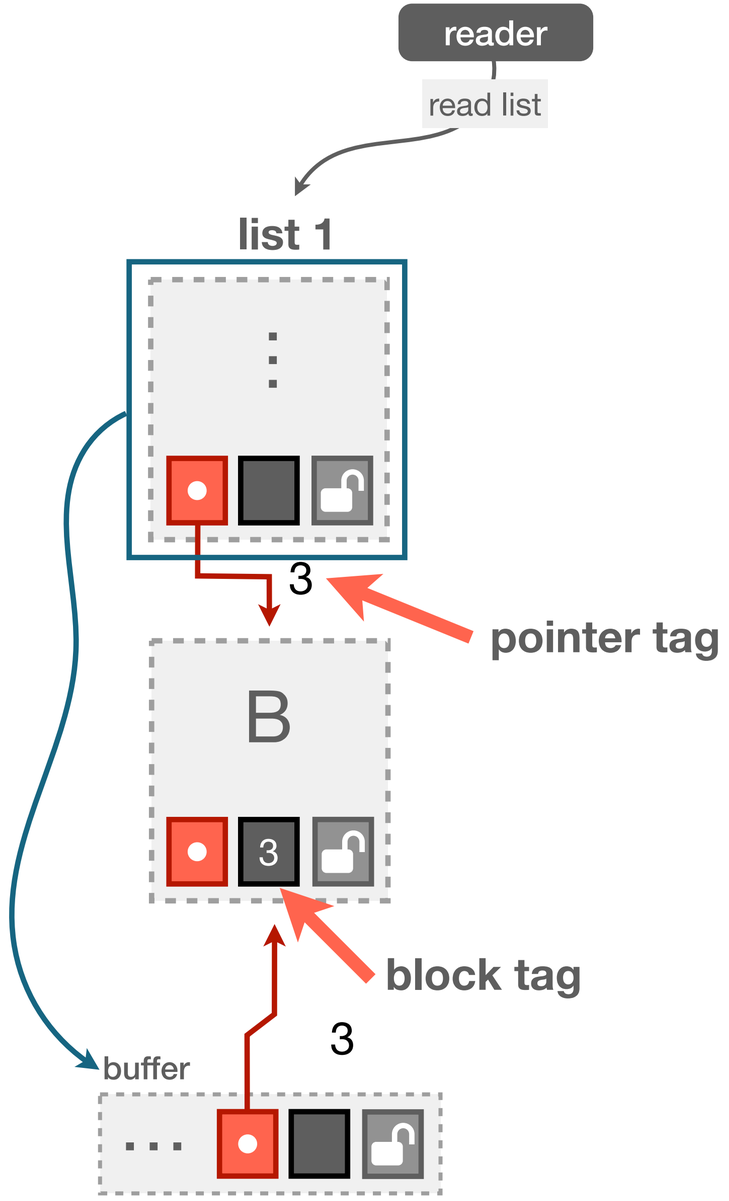
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)
- Block re-allocations
  - Pointer/block tagging

OPTIMISTIC CONCURRENCY CONTROL



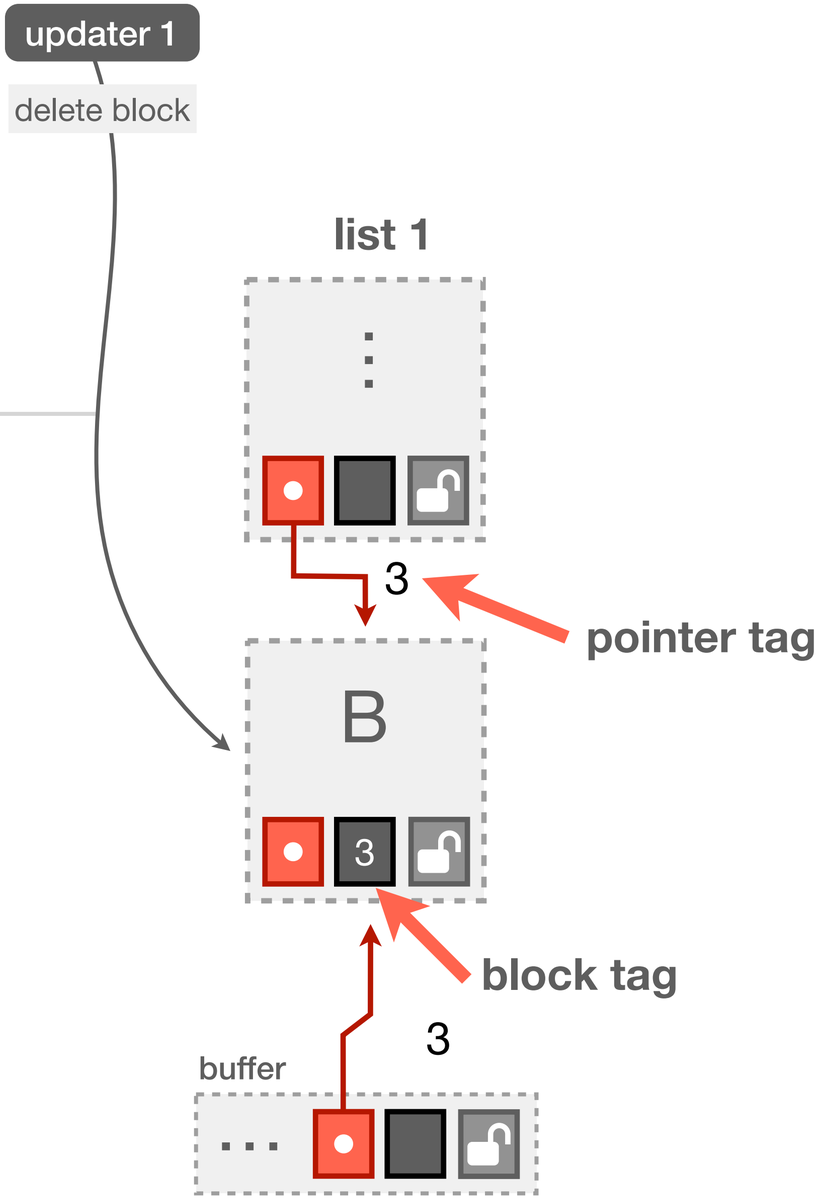
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)
- Block re-allocations
  - Pointer/block tagging

OPTIMISTIC CONCURRENCY CONTROL



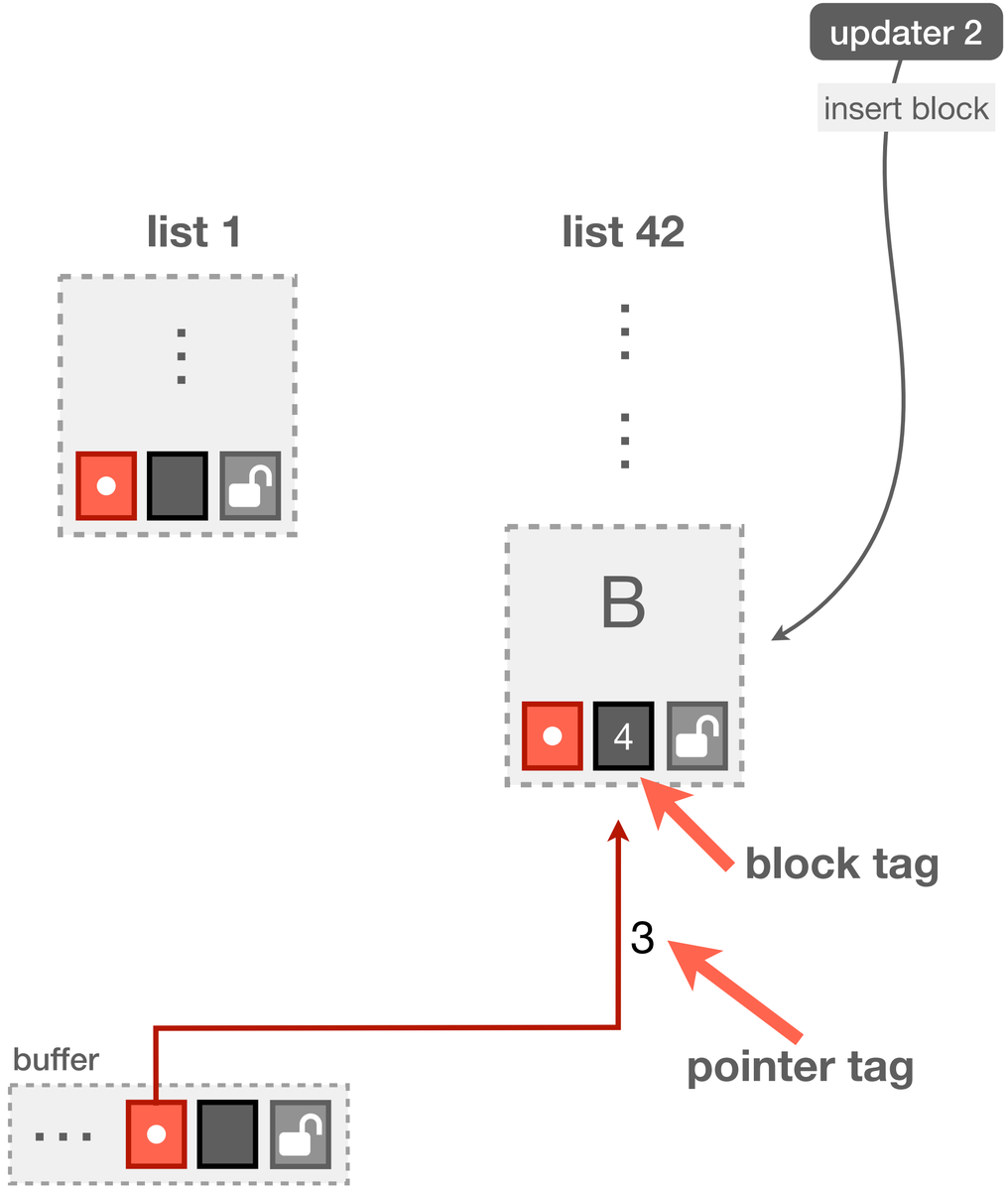
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)
- Block re-allocations
  - Pointer/block tagging

OPTIMISTIC CONCURRENCY CONTROL



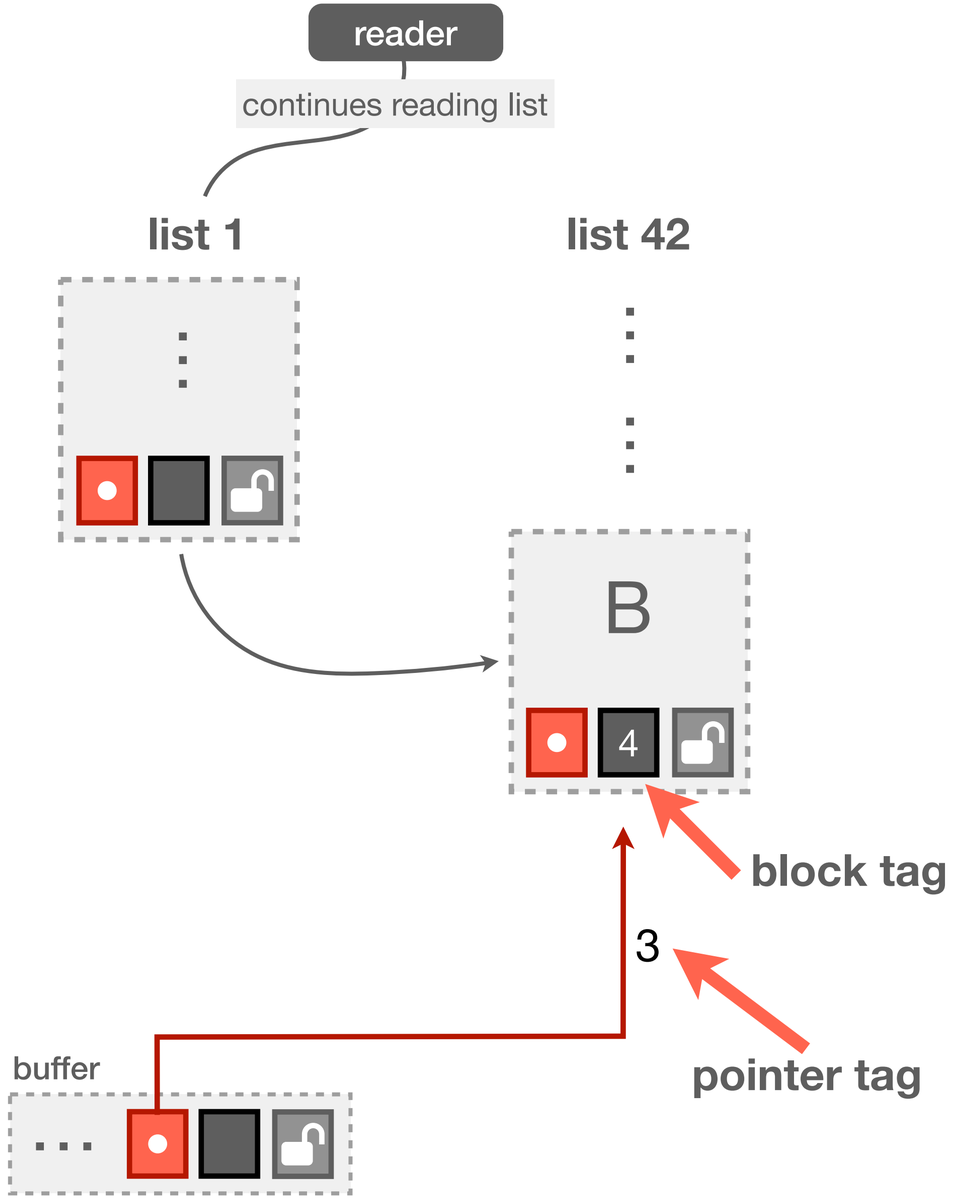
# Index Updates

Protocol for incremental concurrent index updates

**MAIN CHALLENGES**

- Write-write conflicts
  - Rear write locks
- Read-write conflicts
  - Verification through versioning (lock-free)
- Block re-allocations
  - Pointer/block tagging

OPTIMISTIC CONCURRENCY CONTROL



# Conclusion

- Identified key performance bottlenecks of traditional partitioning schemes

not efficient under memory disaggregation

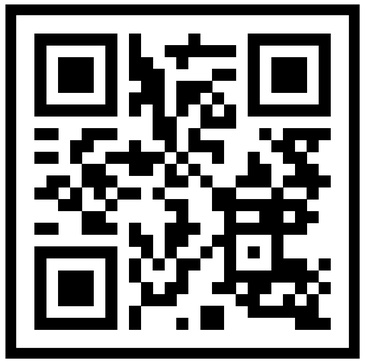
- Proposed a scalable inverted list index design for disaggregated memory
- Developed a protocol to support fast concurrent index updates:
  - Lock-free read queries
  - Fine-grained write locks for updates
- Scalability evaluation on real-world datasets

# Scalable Distributed Inverted List Indexes in Disaggregated Memory

**MANUEL WIDMOSER**
DANIEL KOCHER
NIKOLAUS AUGSTEN


manuel.widmoser@plus.ac.at


github.com/DatabaseGroup/rdma-inverted-index


link to paper