# Sliding Look-Back Window Assisted Data Chunk Rewriting for Improving Deduplication Restore Performance

**Zhichao Cao**[1], Shiyong Liu[2], Fenggang Wu[1], Guohua Wang[3], Bingzhe Li[1], and David H.C. Du[1]

[1]*University of Minnesota, Twin Cities*

[2]*Ocean University of China* [3]*South China University of Technology*

02/26/2019









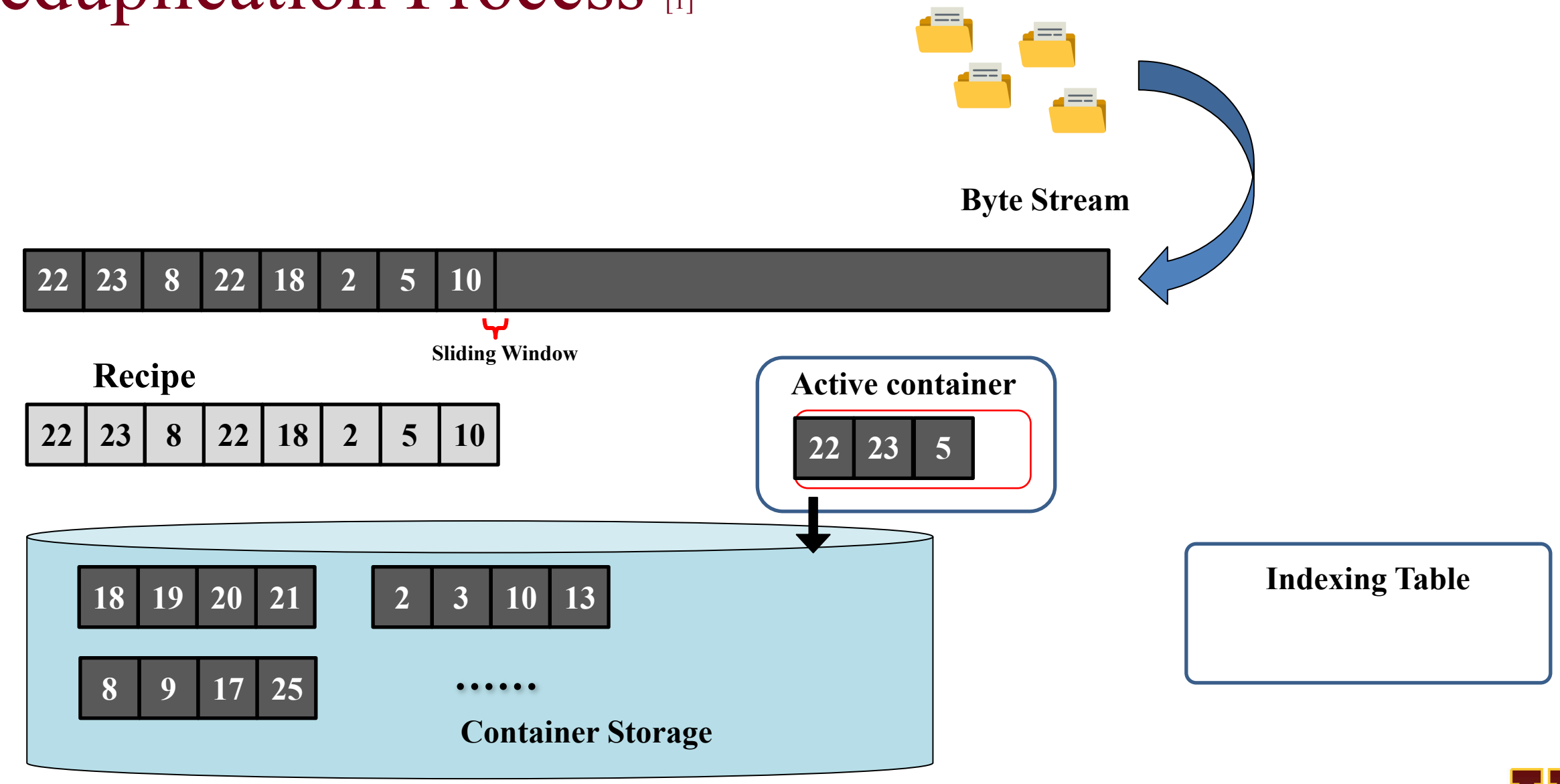
# Agenda

- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

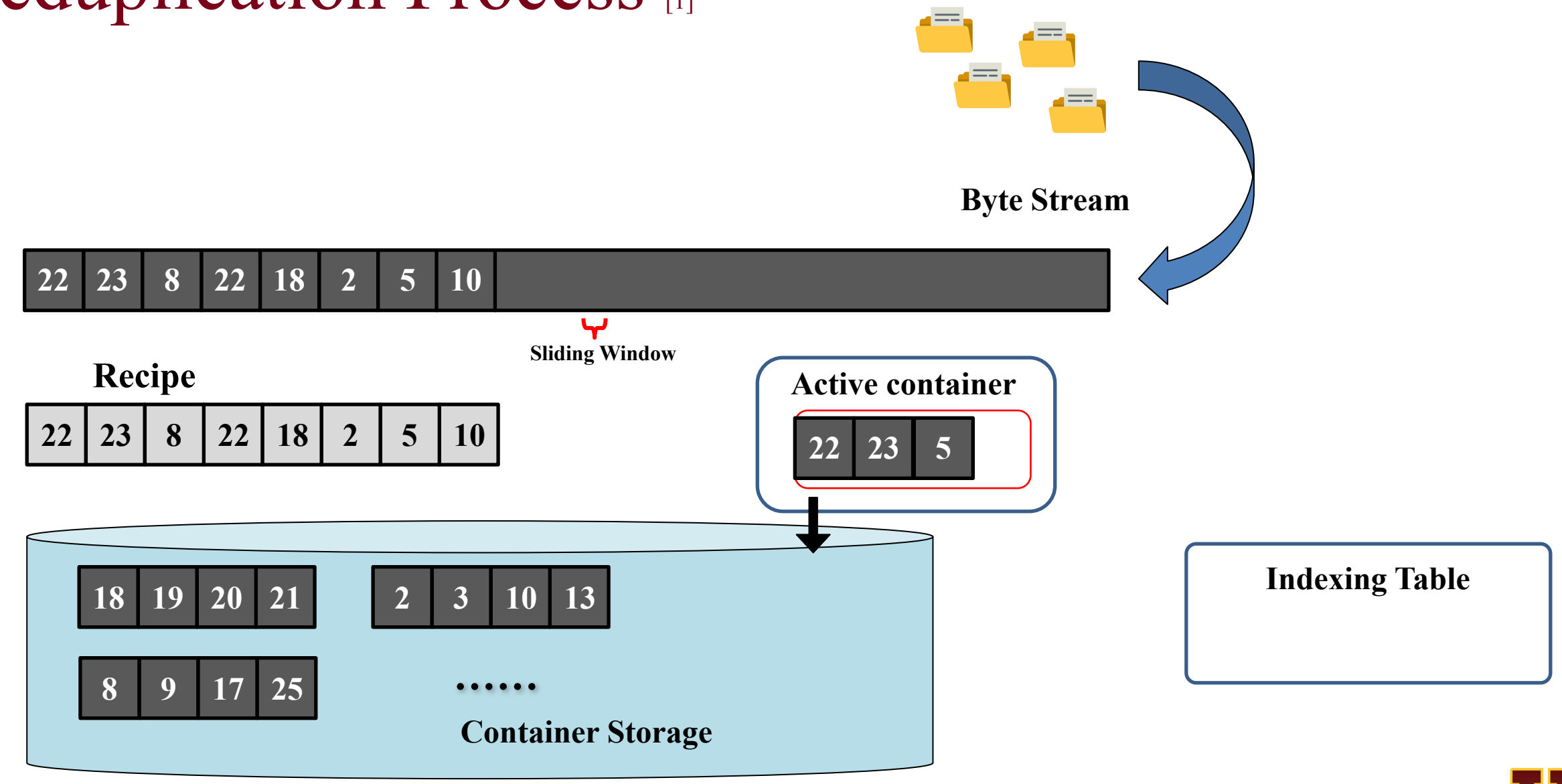
# Agenda

- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

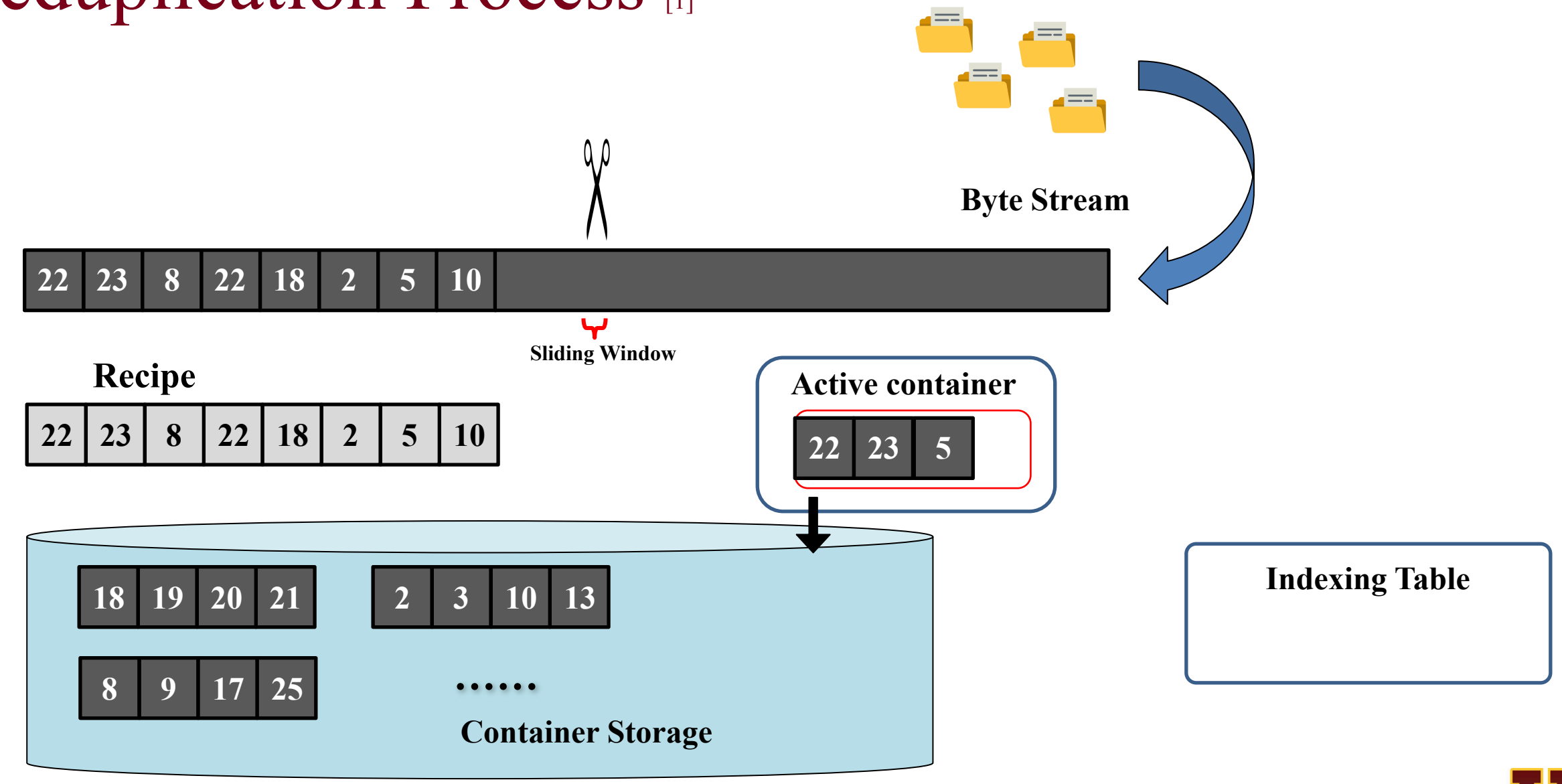
# Deduplication Process [1]



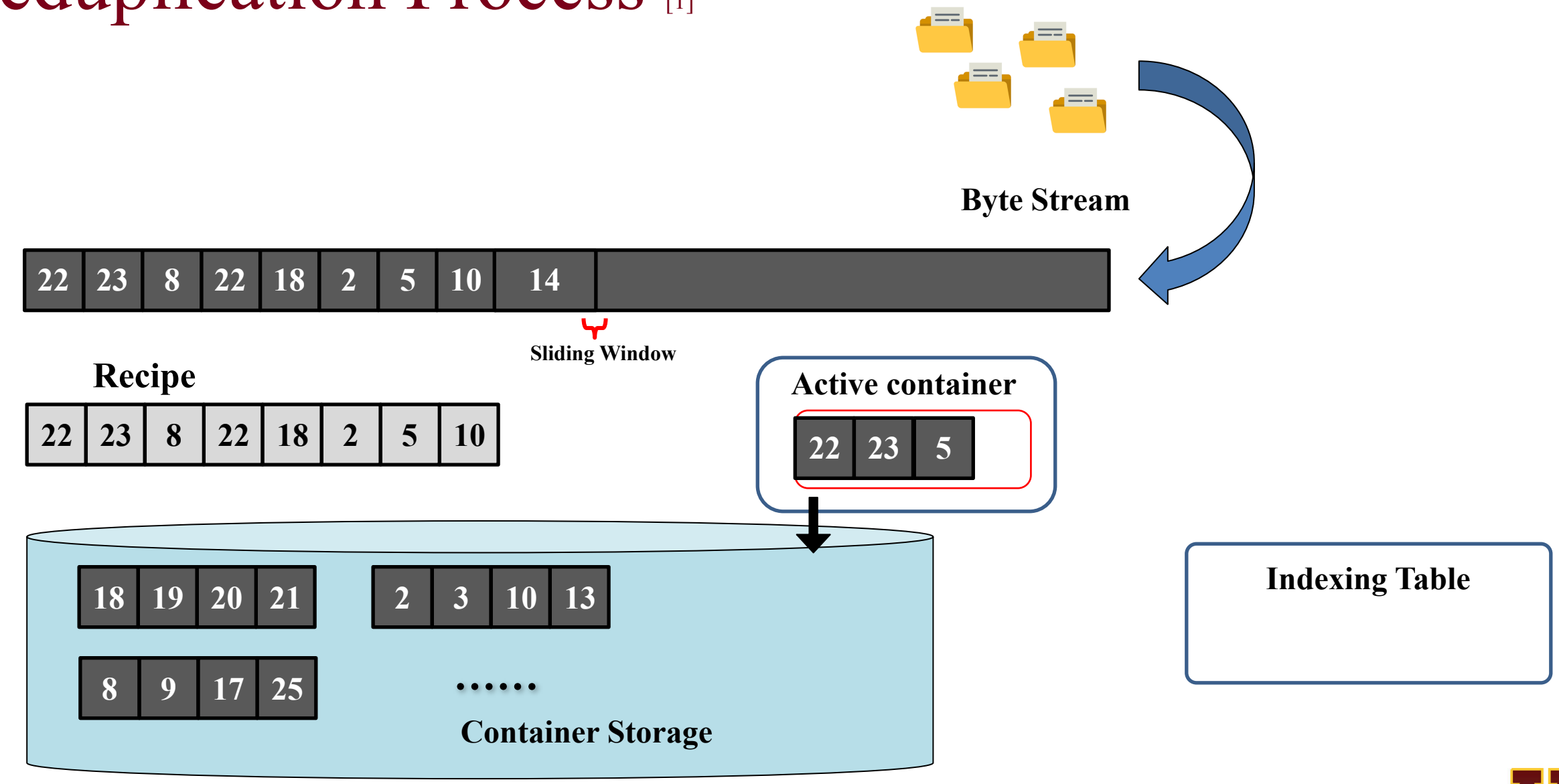
# Deduplication Process [1]



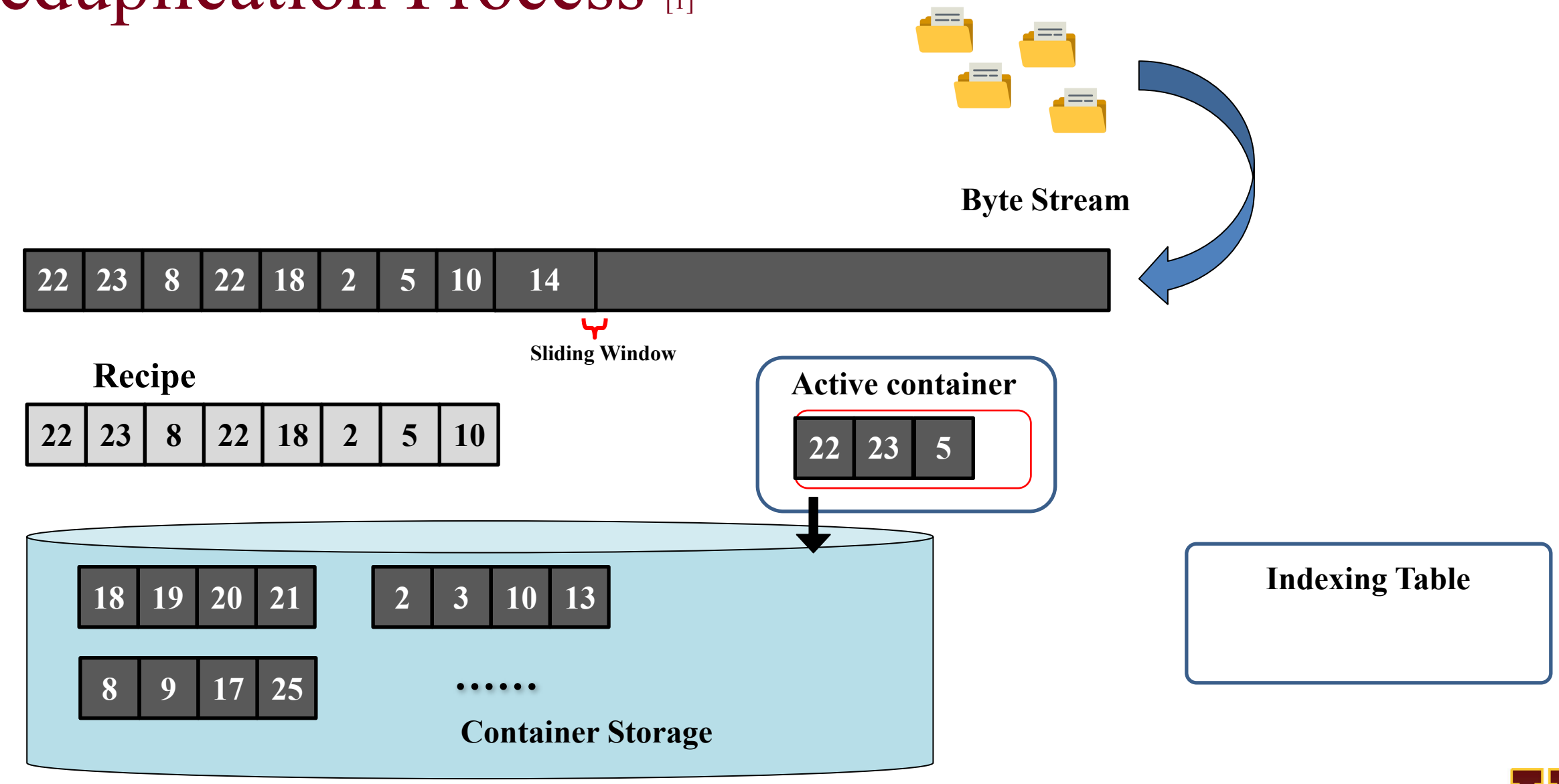
# Deduplication Process [1]



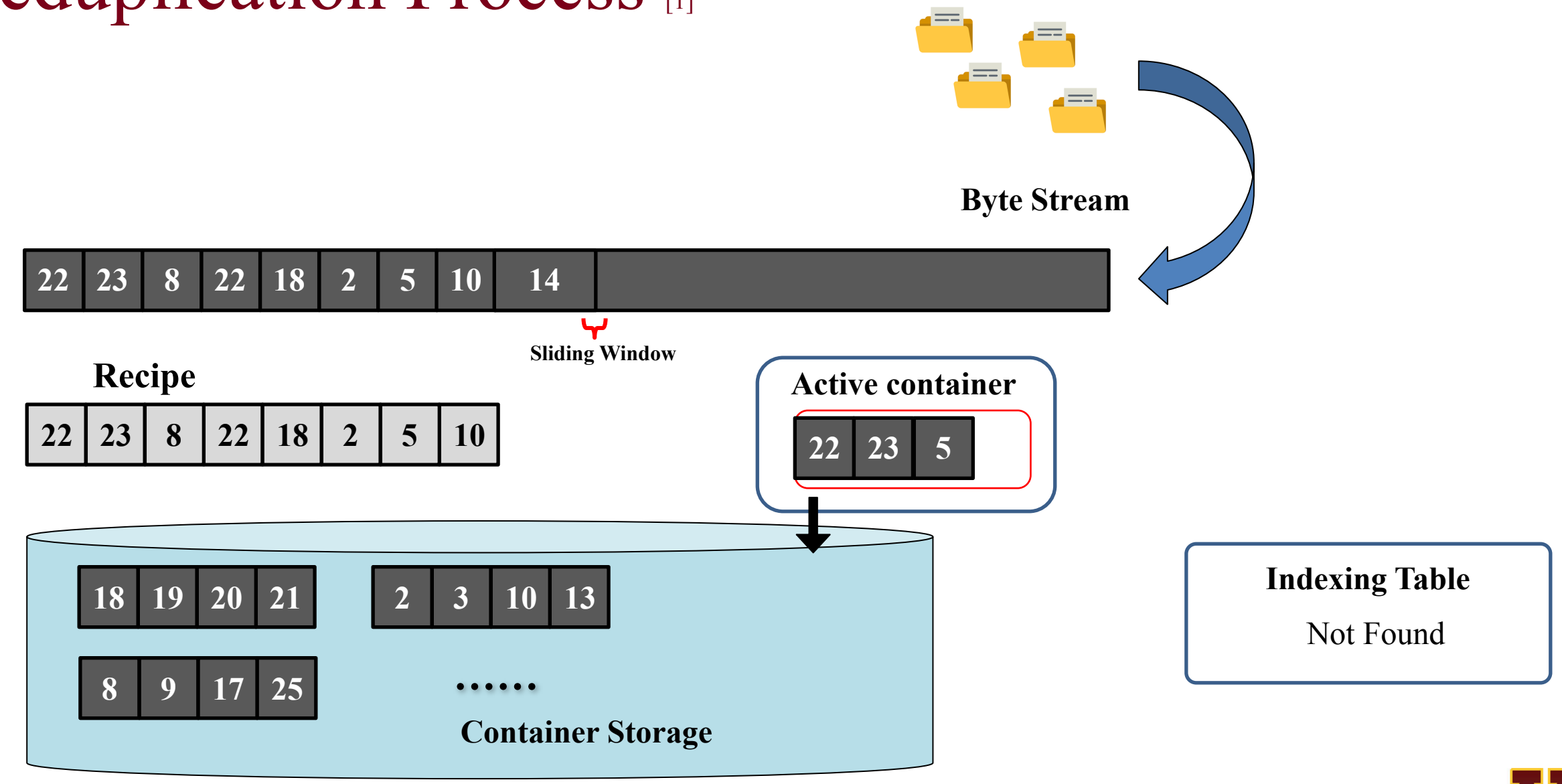
# Deduplication Process [1]



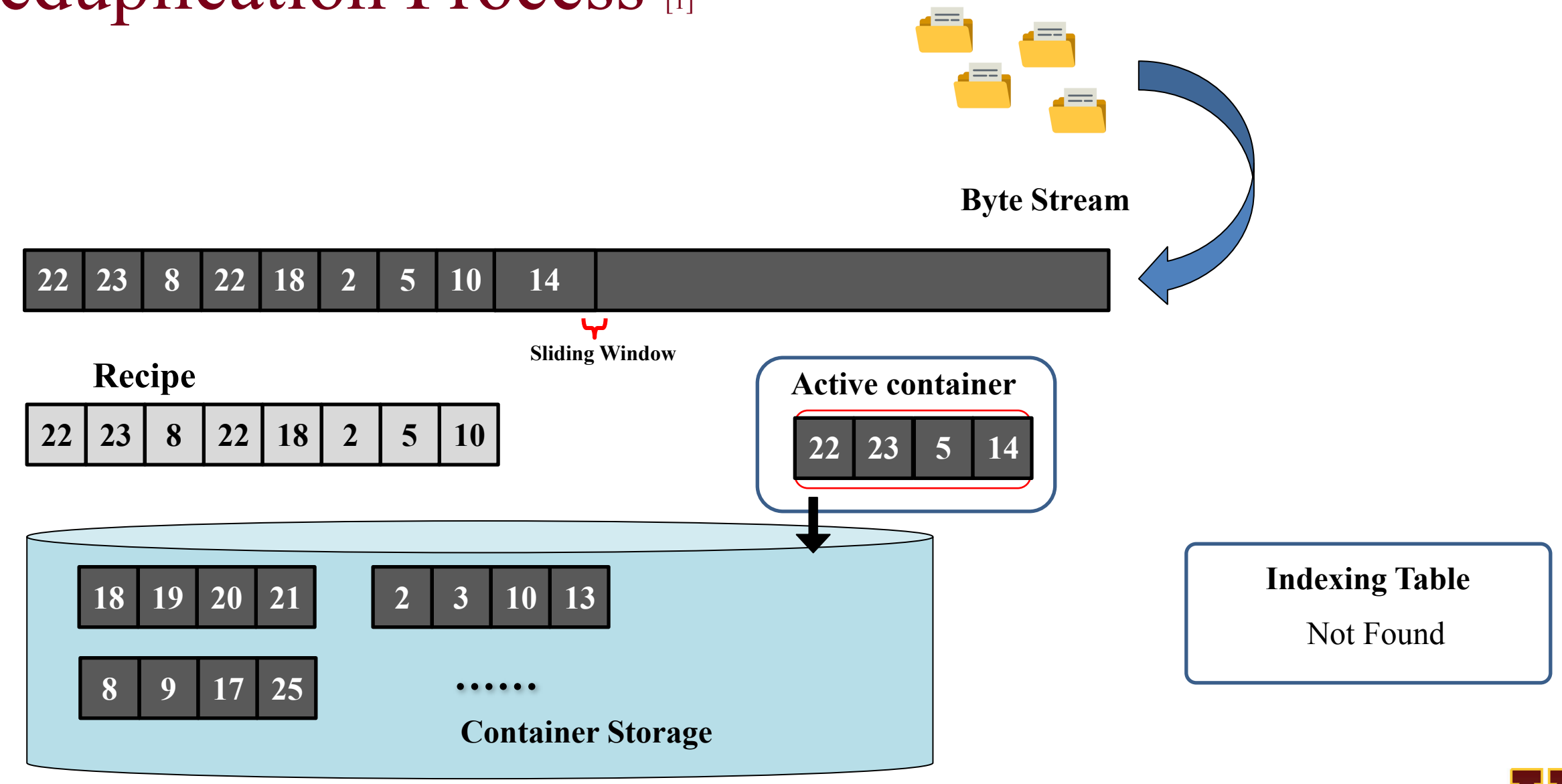
# Deduplication Process [1]



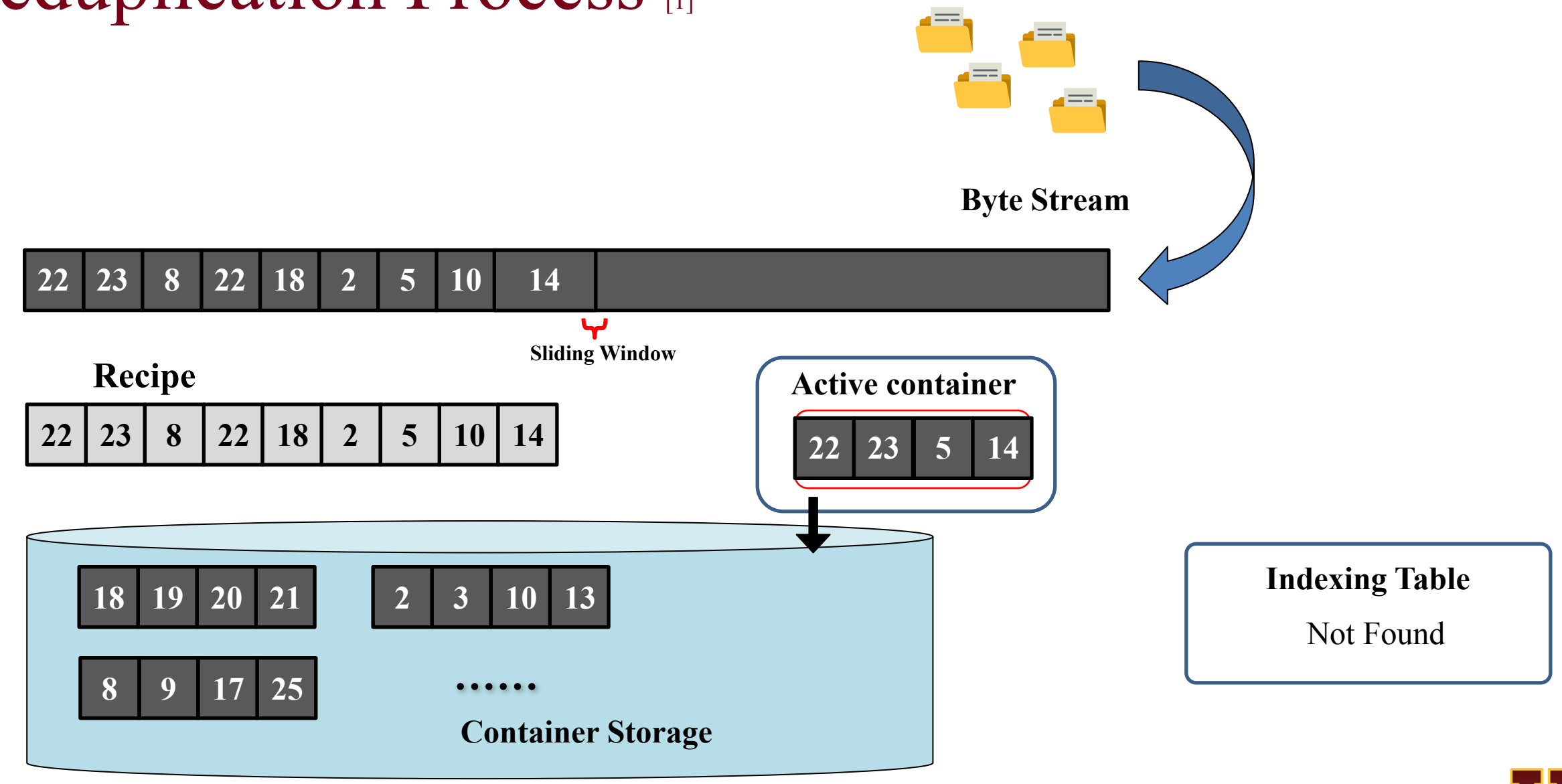
# Deduplication Process [1]



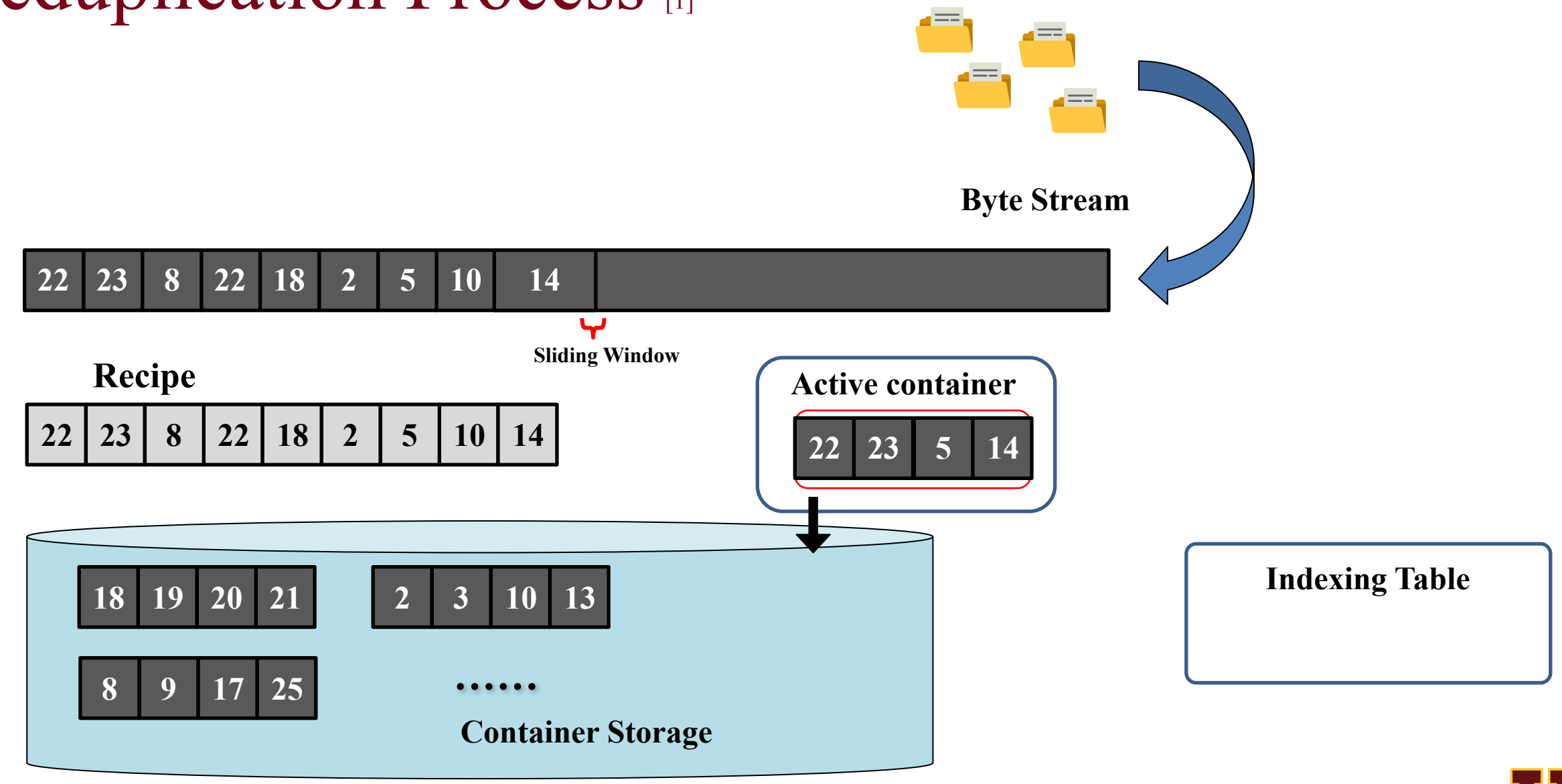
# Deduplication Process [1]



# Deduplication Process [1]



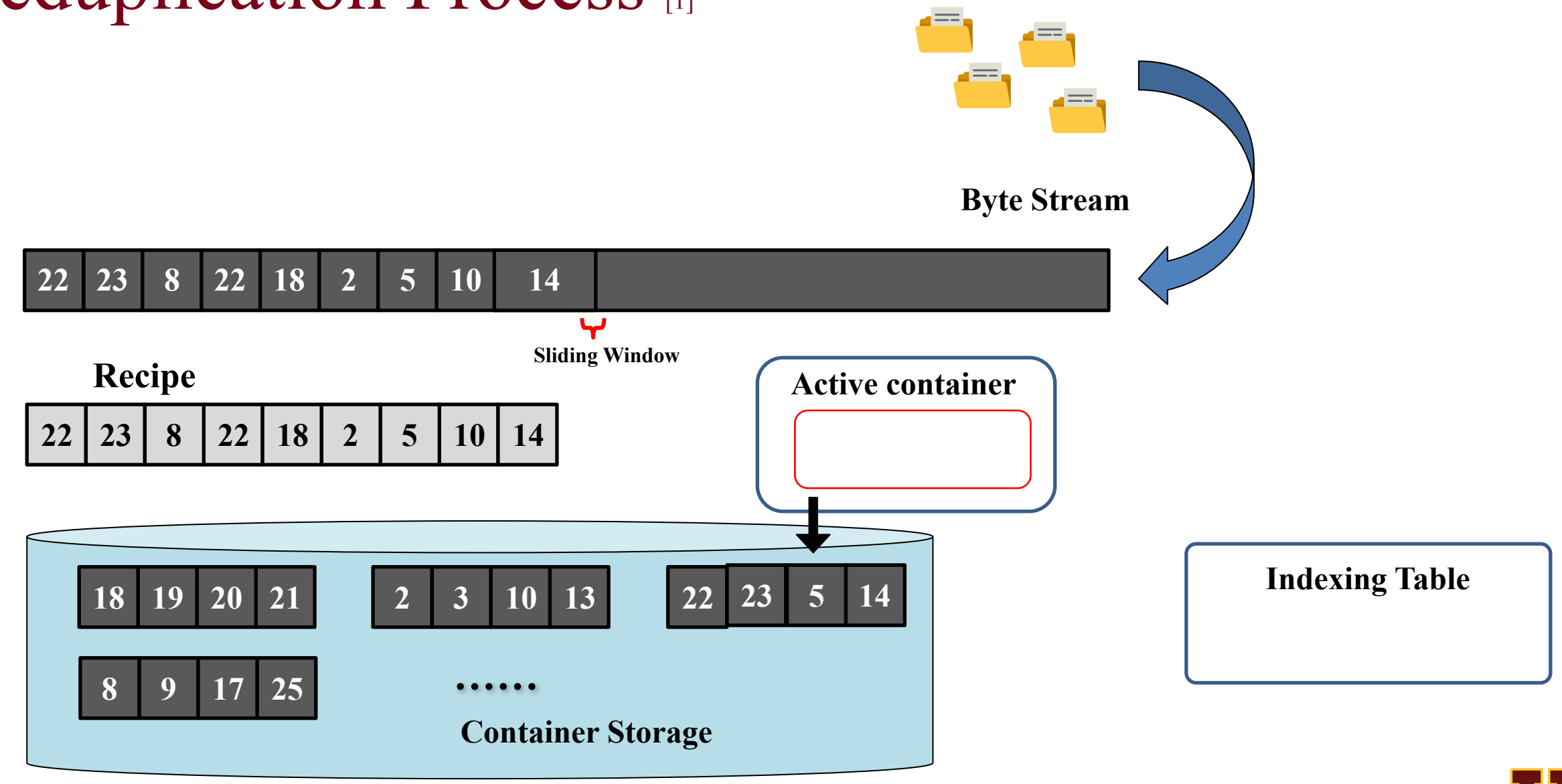
# Deduplication Process [1]






[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

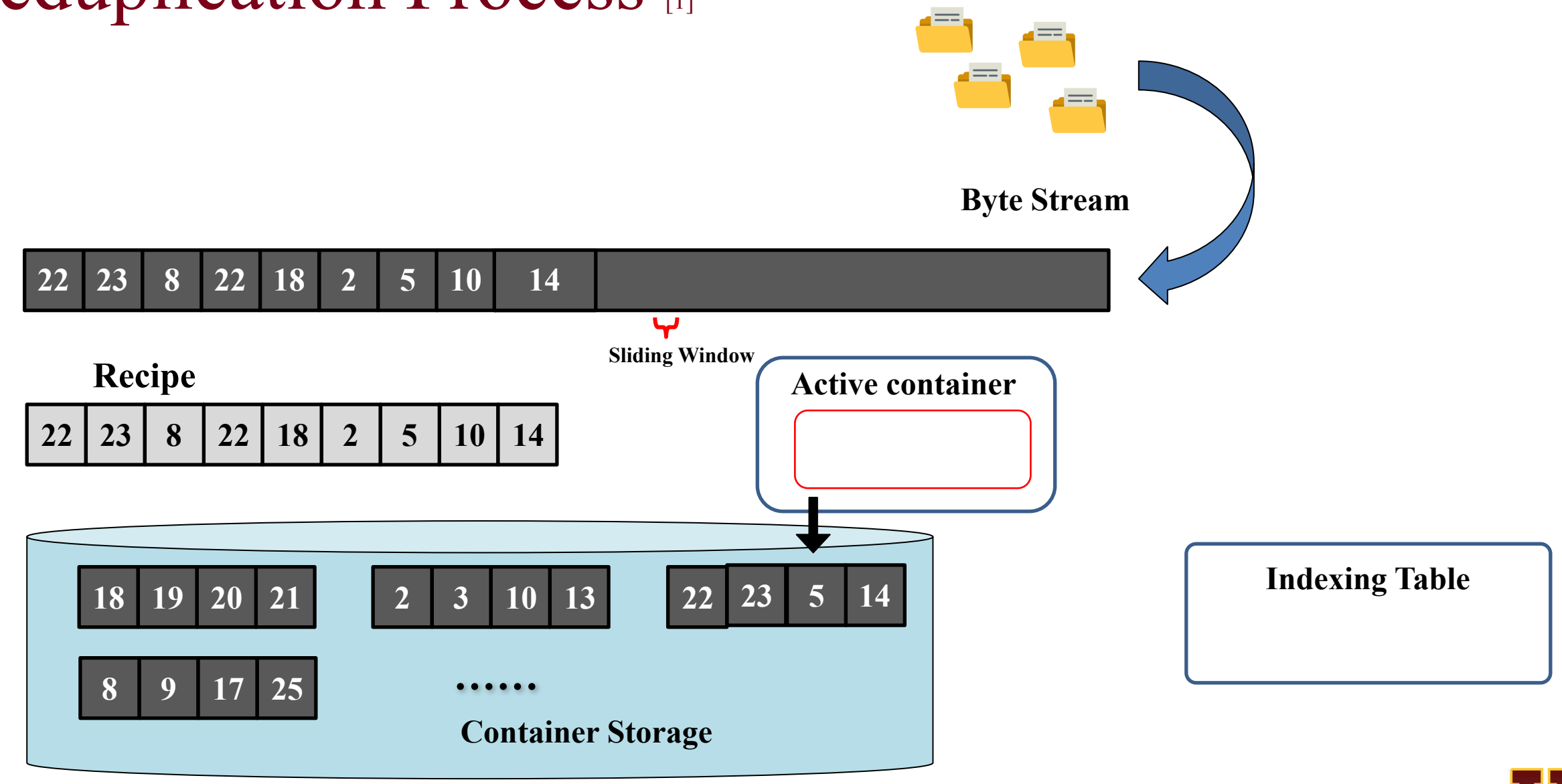
# Deduplication Process [1]






[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

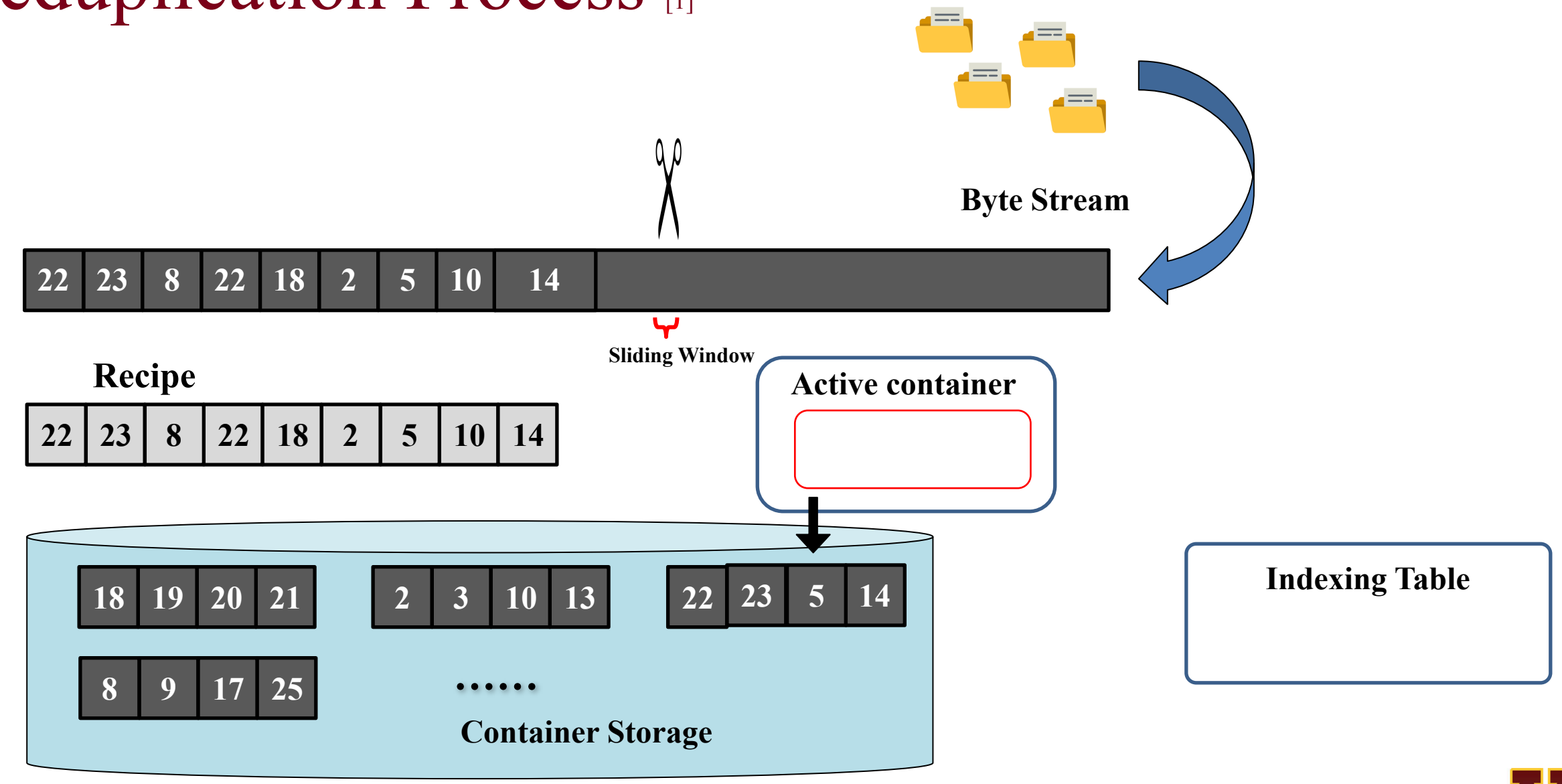
# Deduplication Process [1]






[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

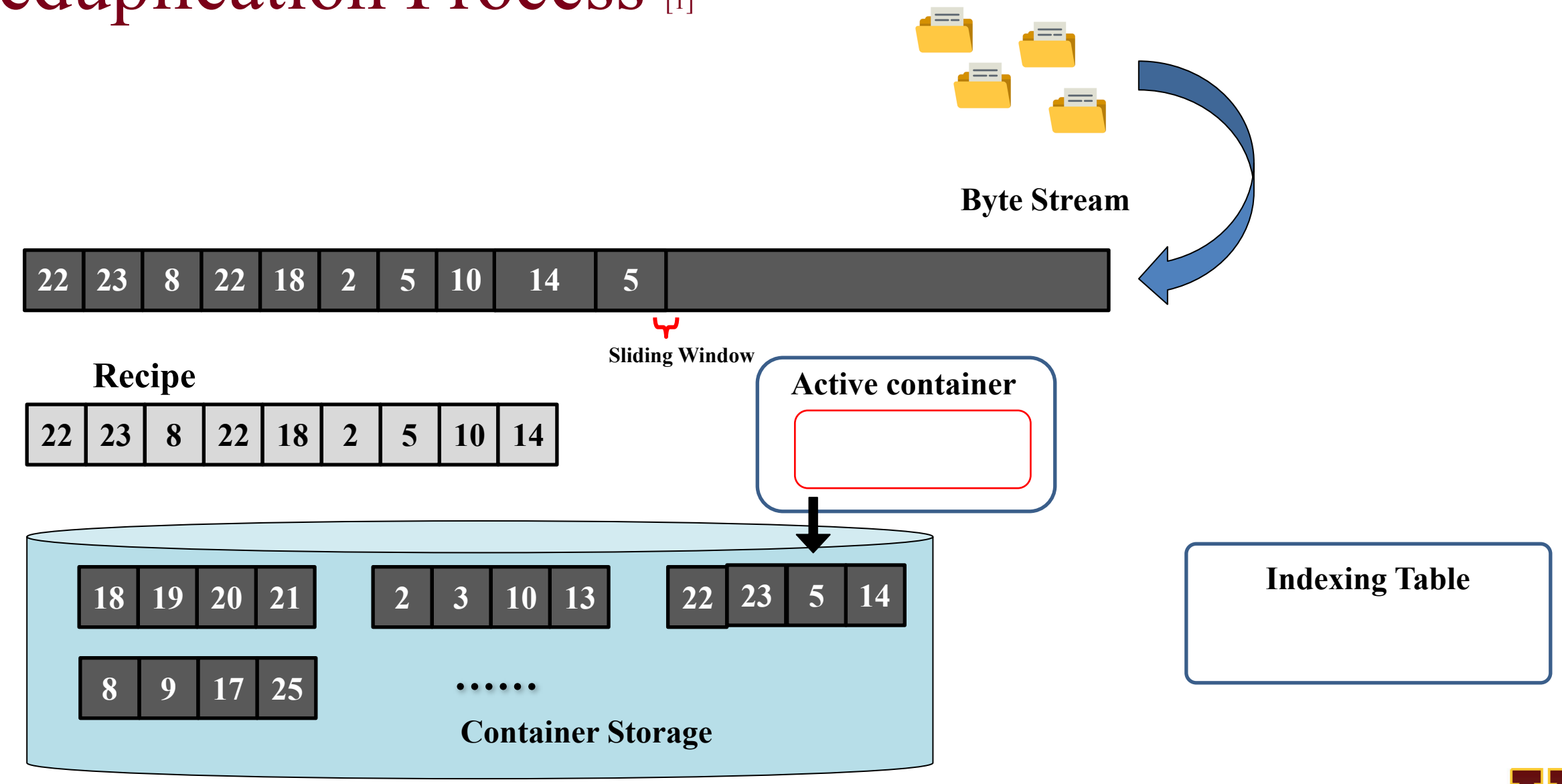
# Deduplication Process [1]






[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

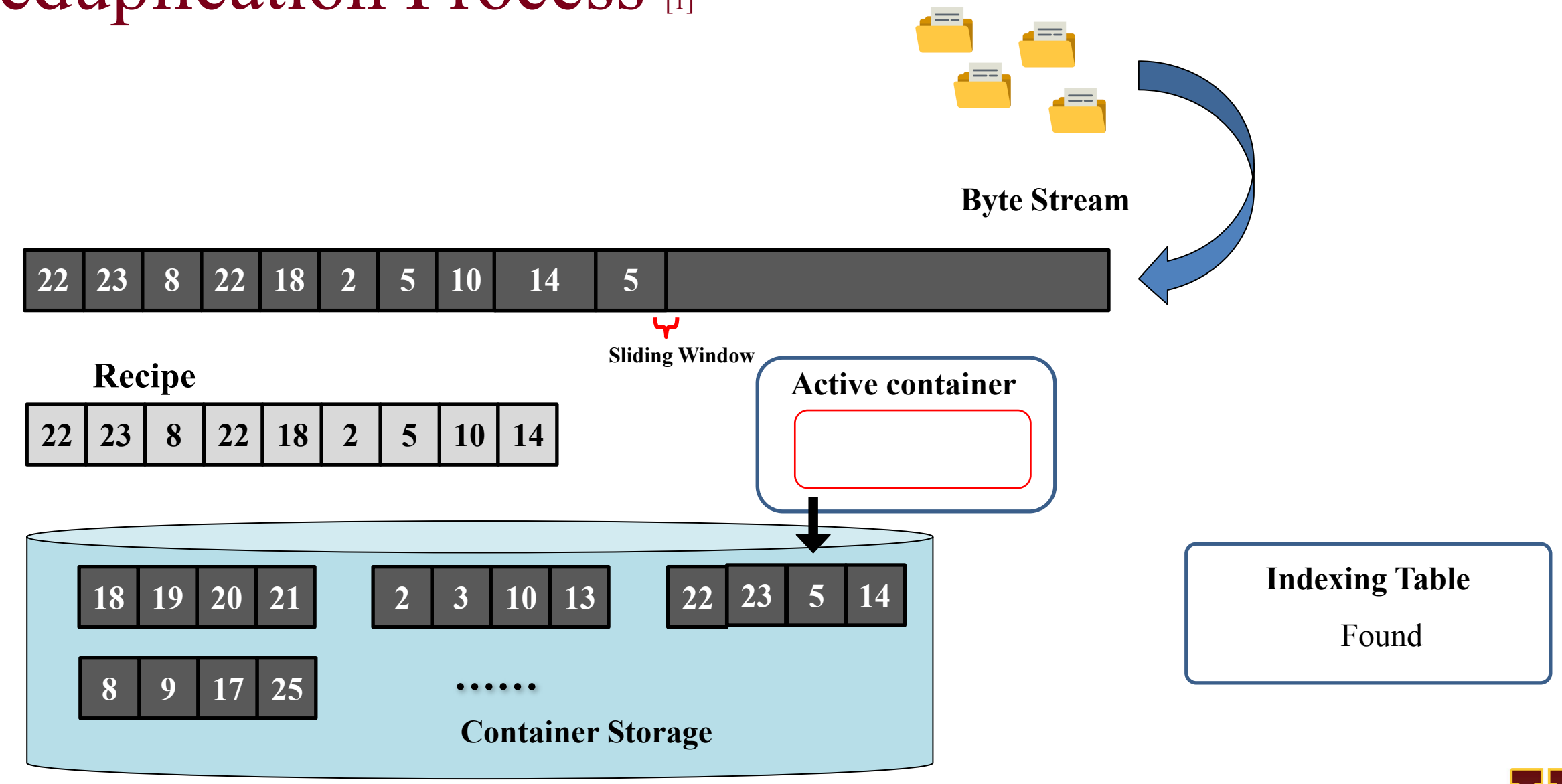
# Deduplication Process [1]






[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

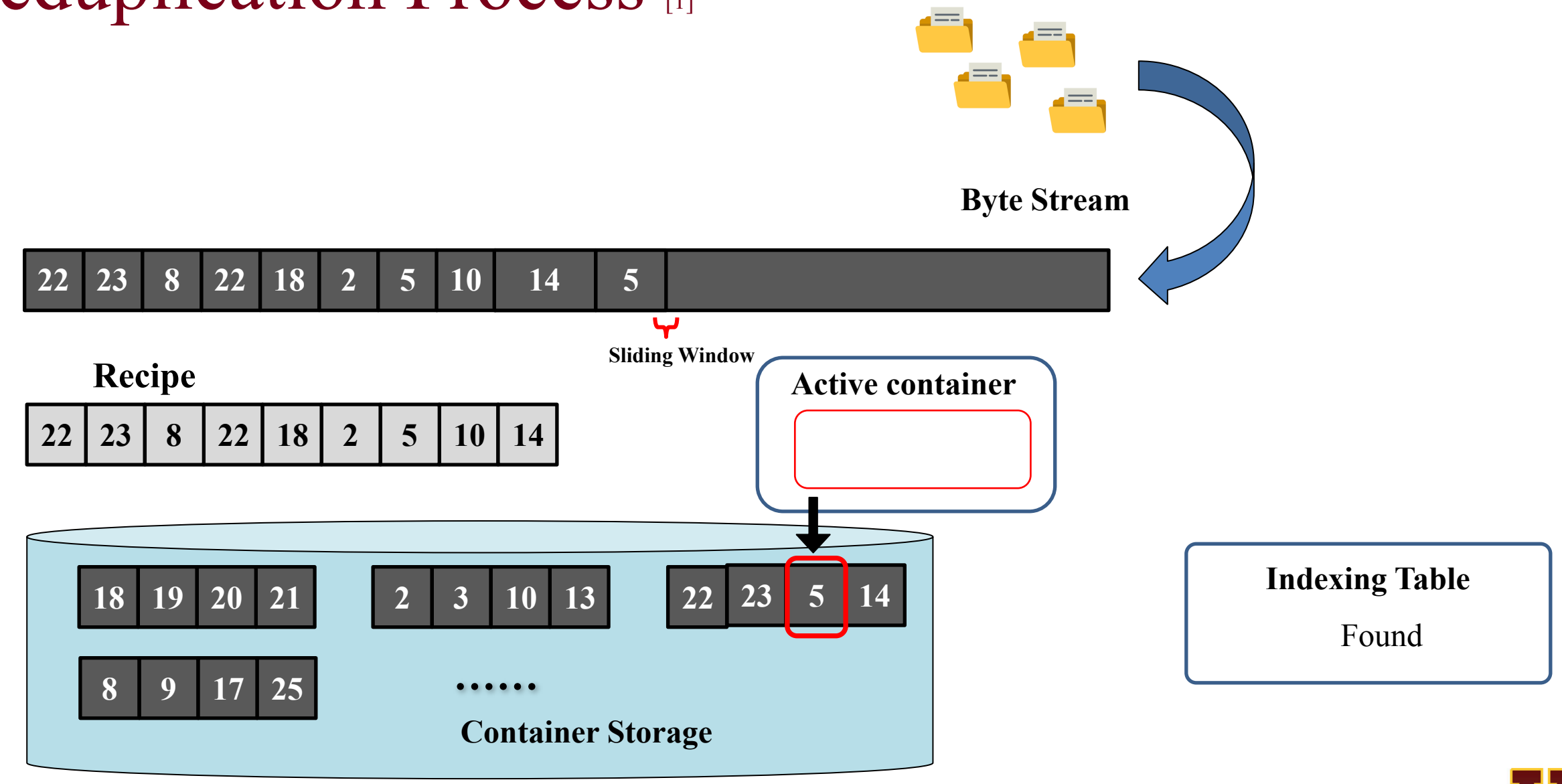
# Deduplication Process [1]






[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

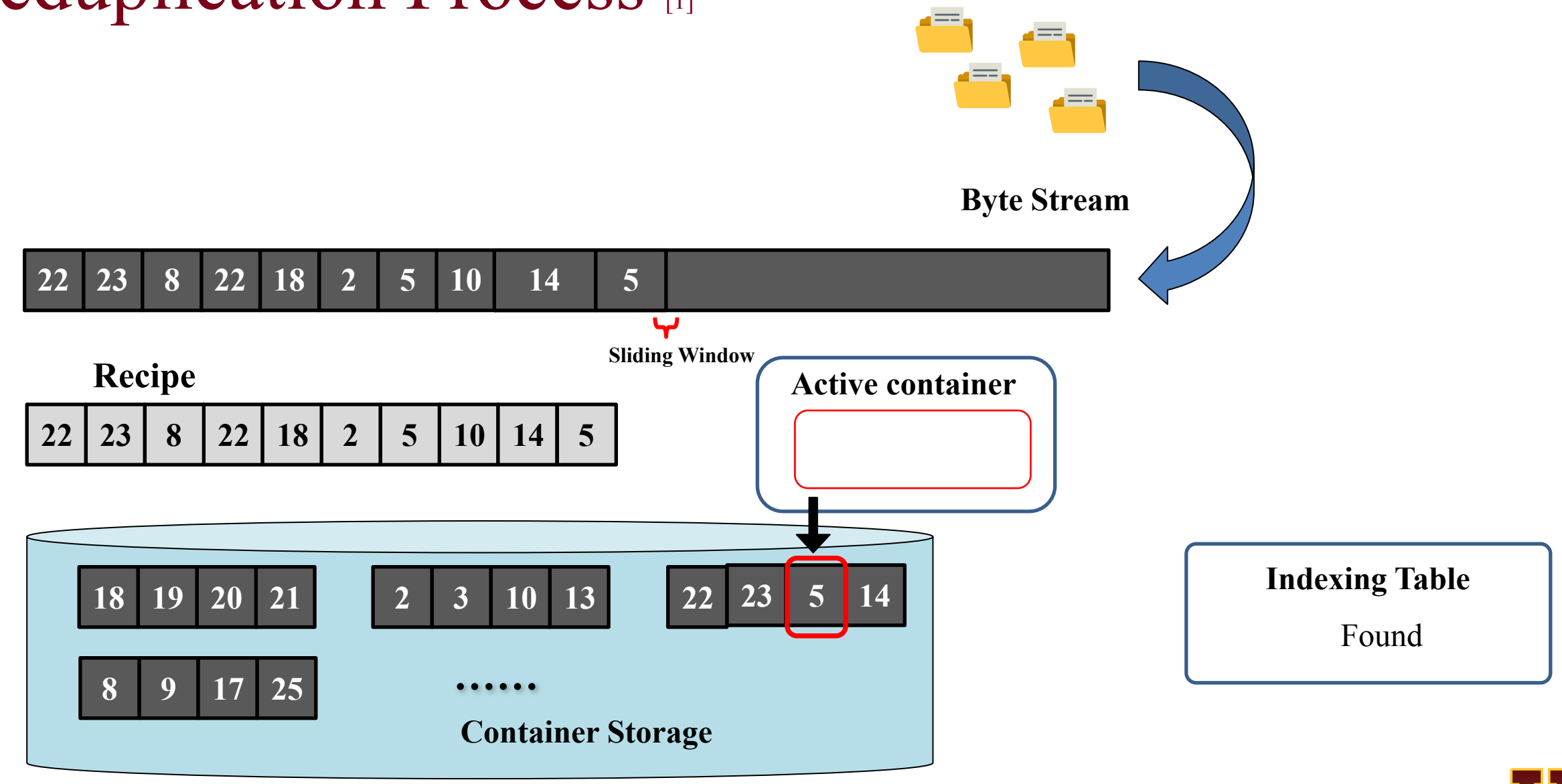
# Deduplication Process [1]






[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

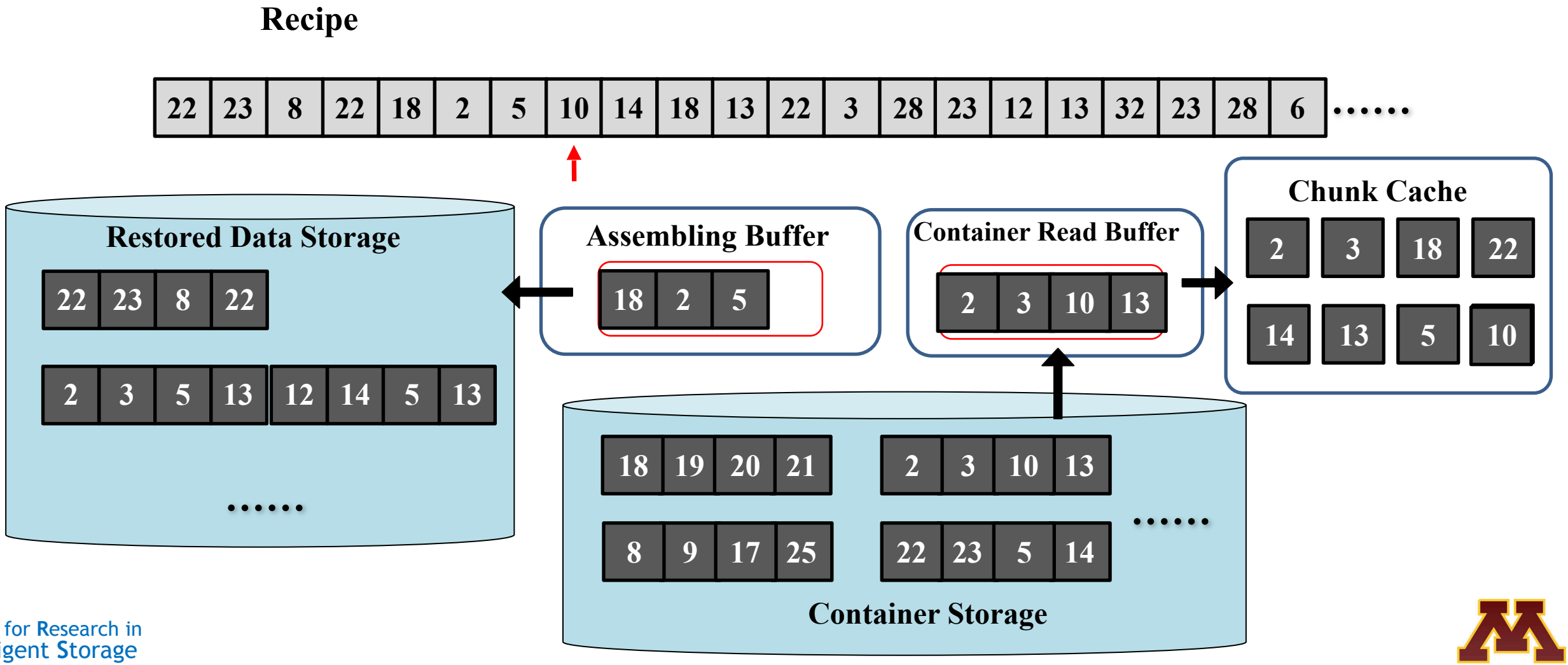
# Deduplication Process [1]



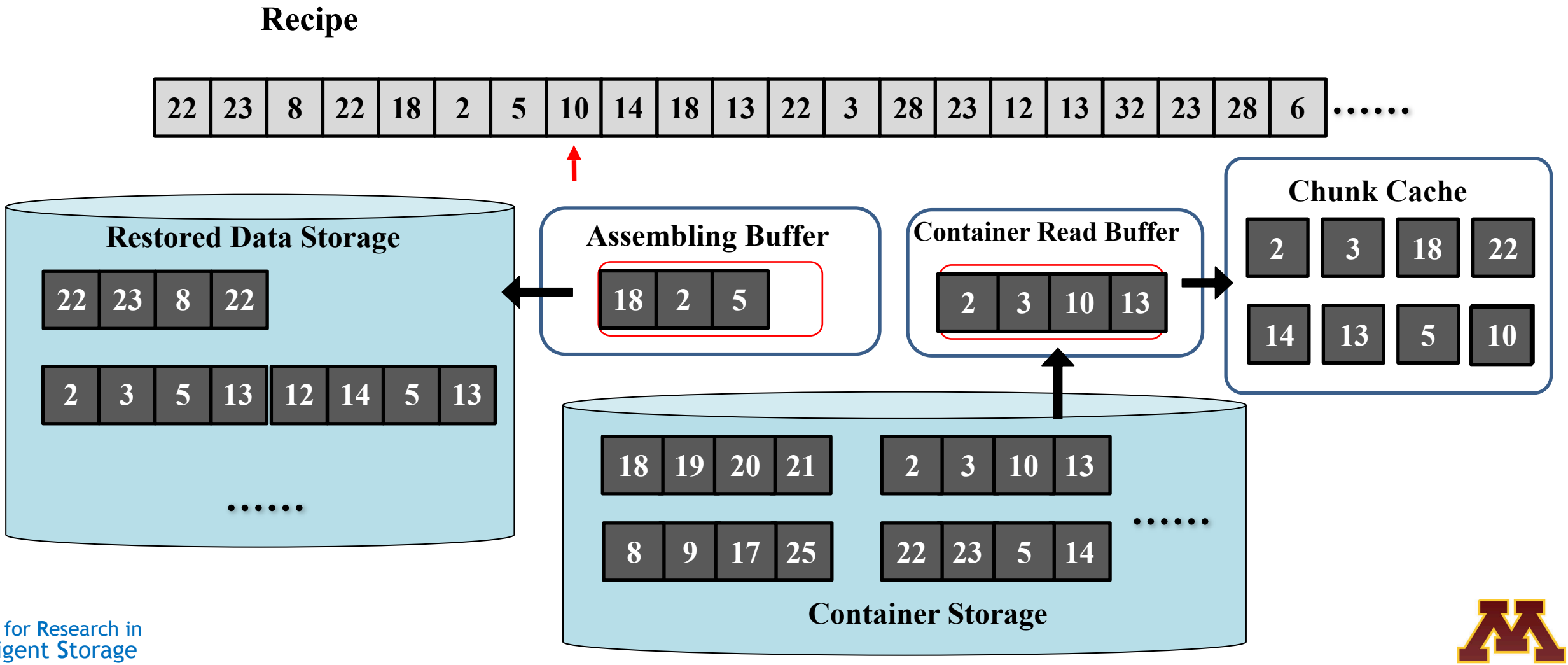



[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

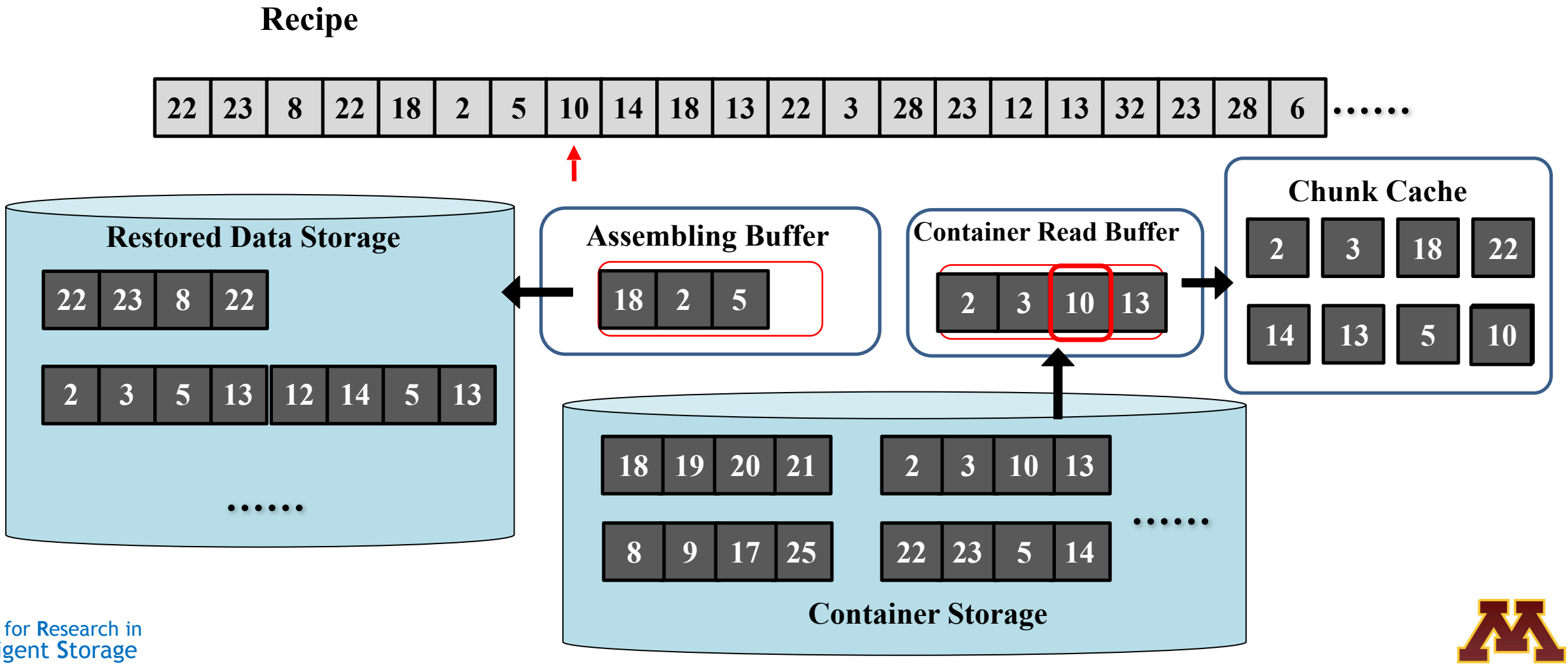
# Restore Process with Chunk-based Caching[2]



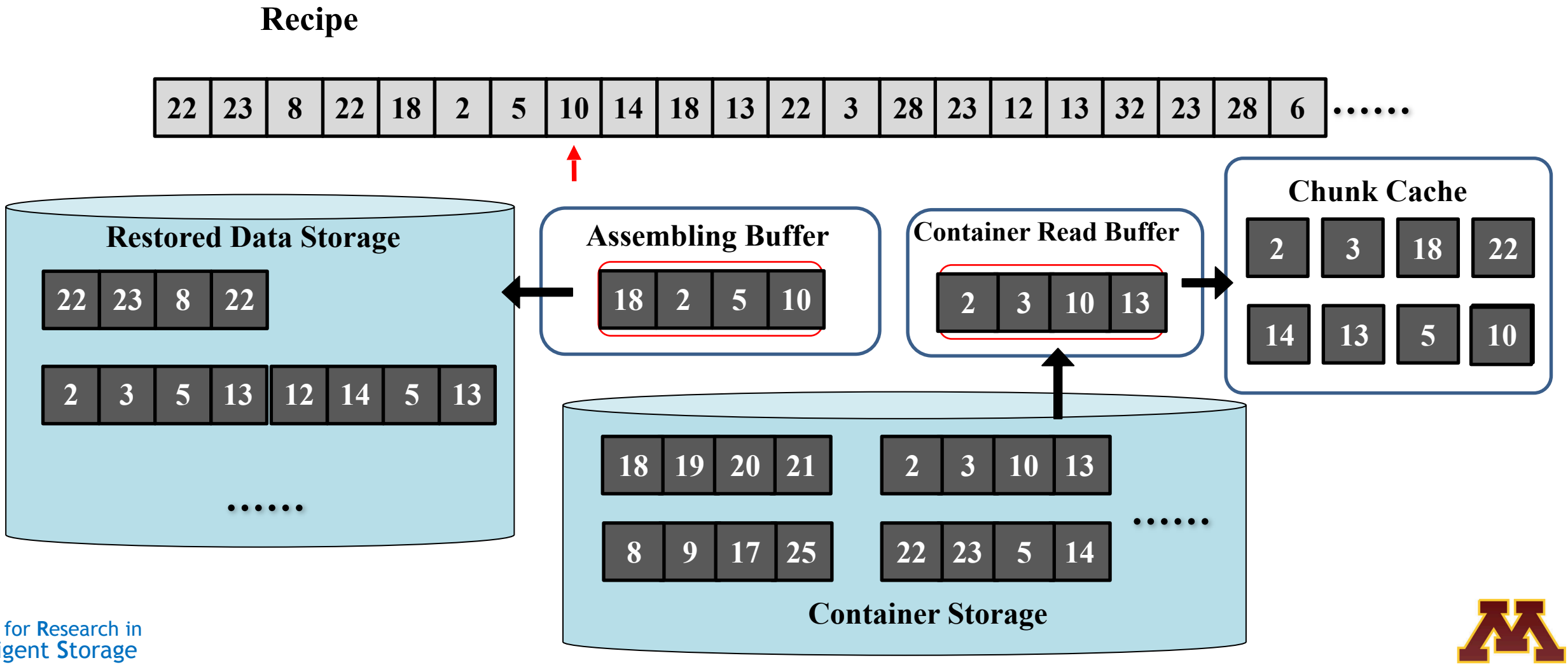
# Restore Process with Chunk-based Caching[2]



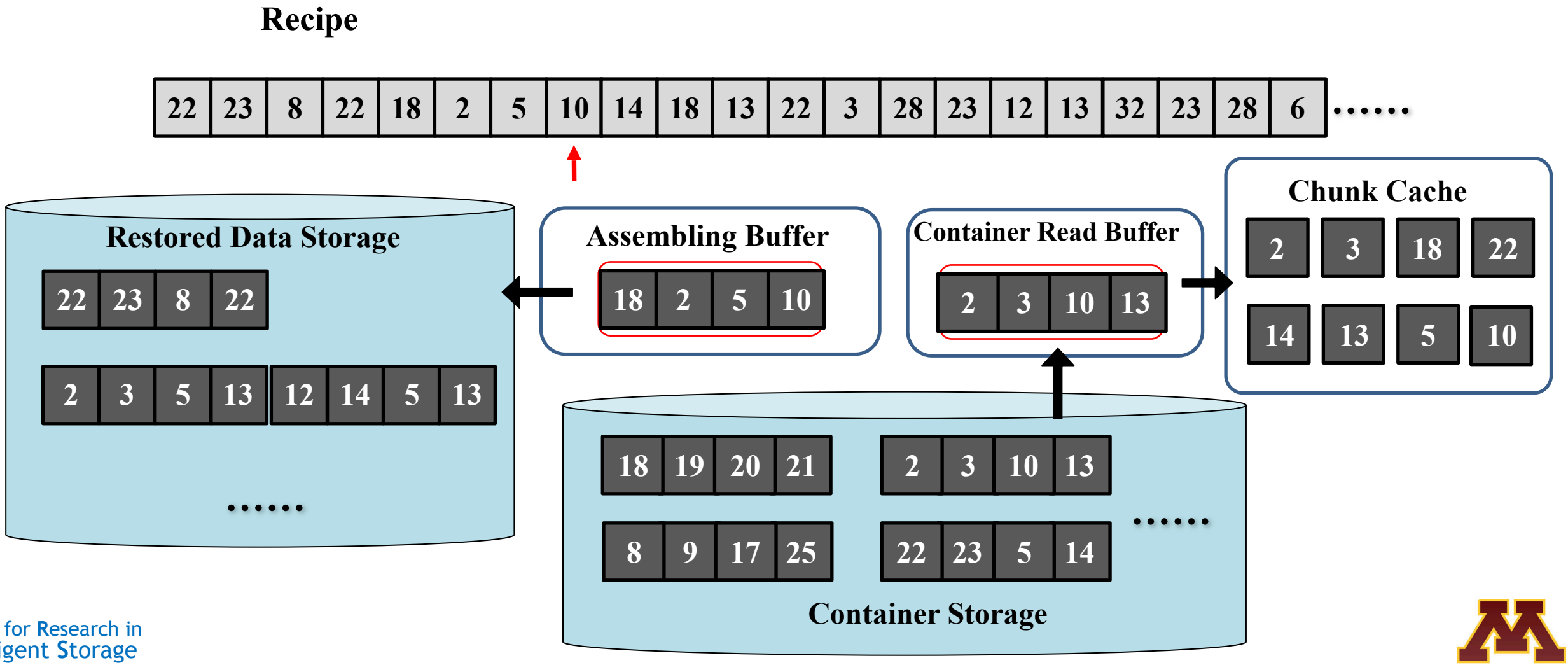
# Restore Process with Chunk-based Caching[2]



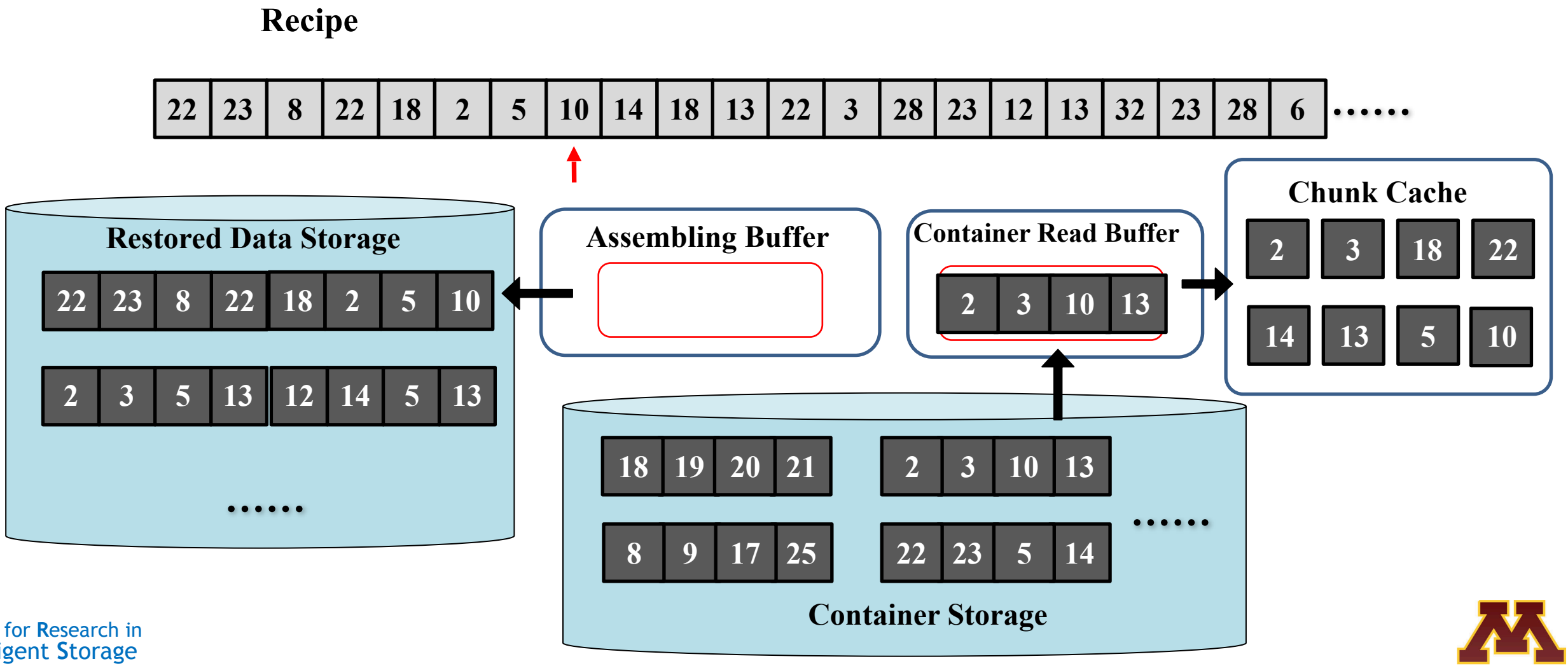
# Restore Process with Chunk-based Caching[2]



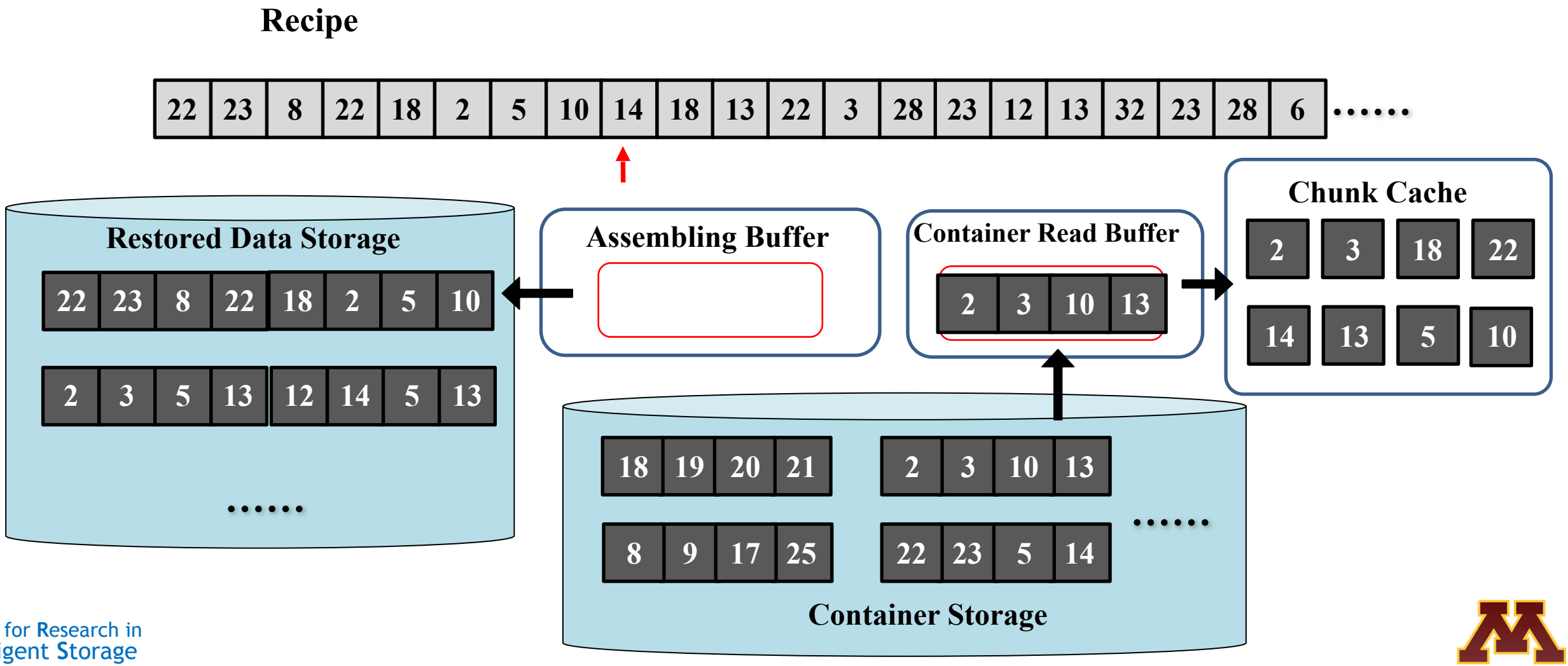
# Restore Process with Chunk-based Caching[2]



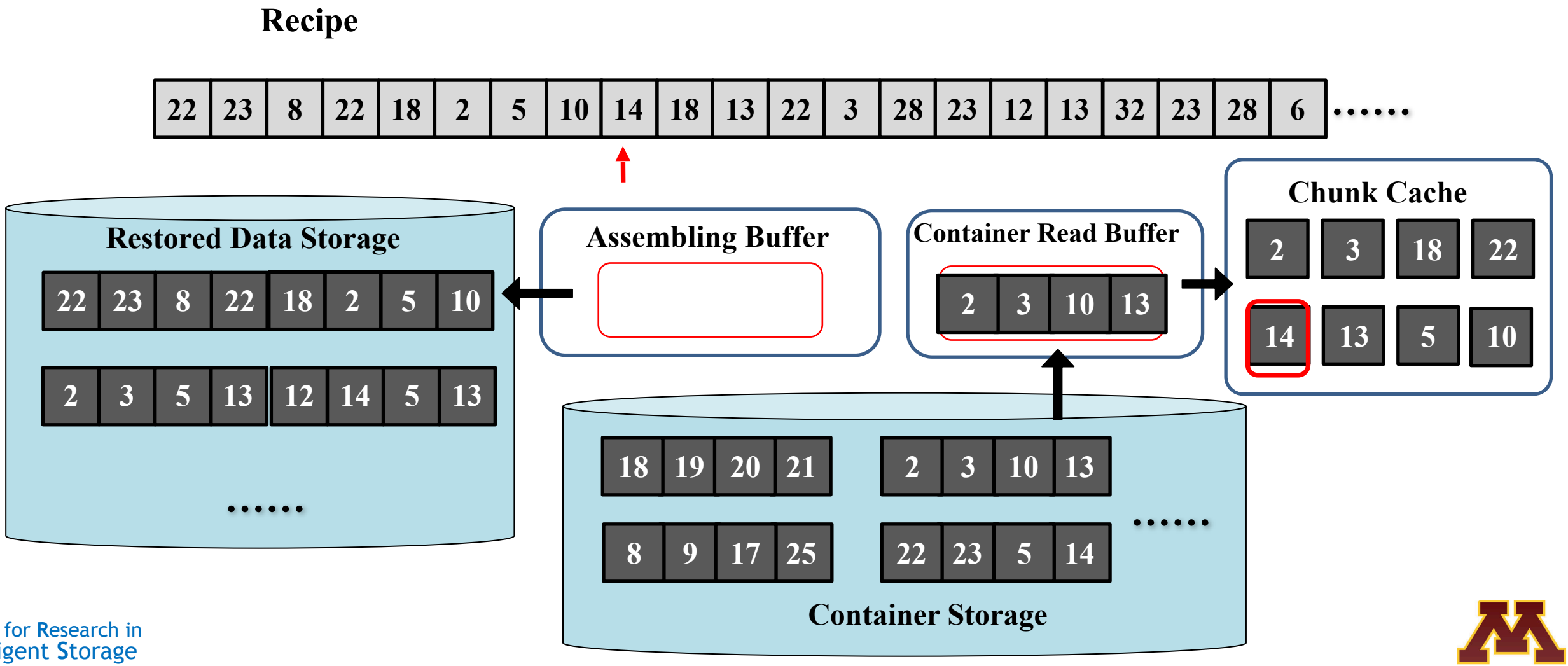
# Restore Process with Chunk-based Caching[2]



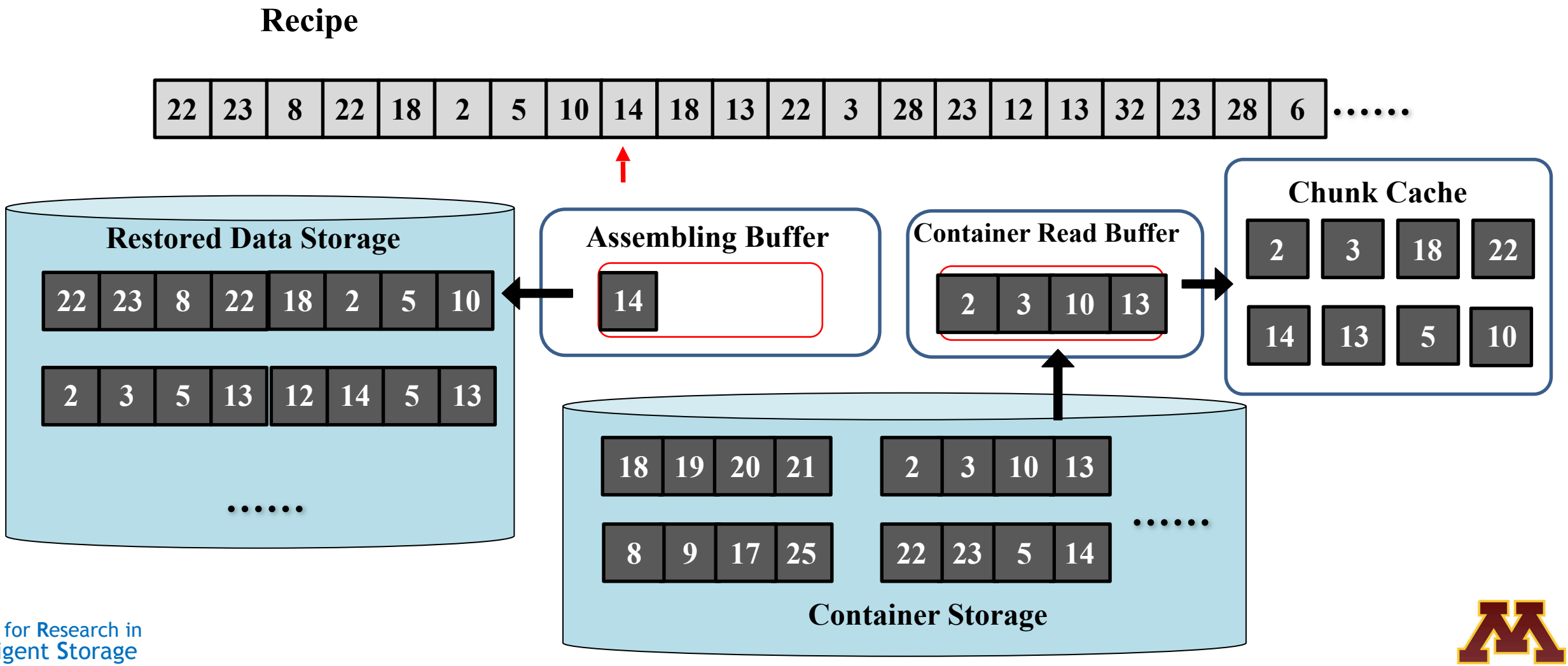
# Restore Process with Chunk-based Caching[2]



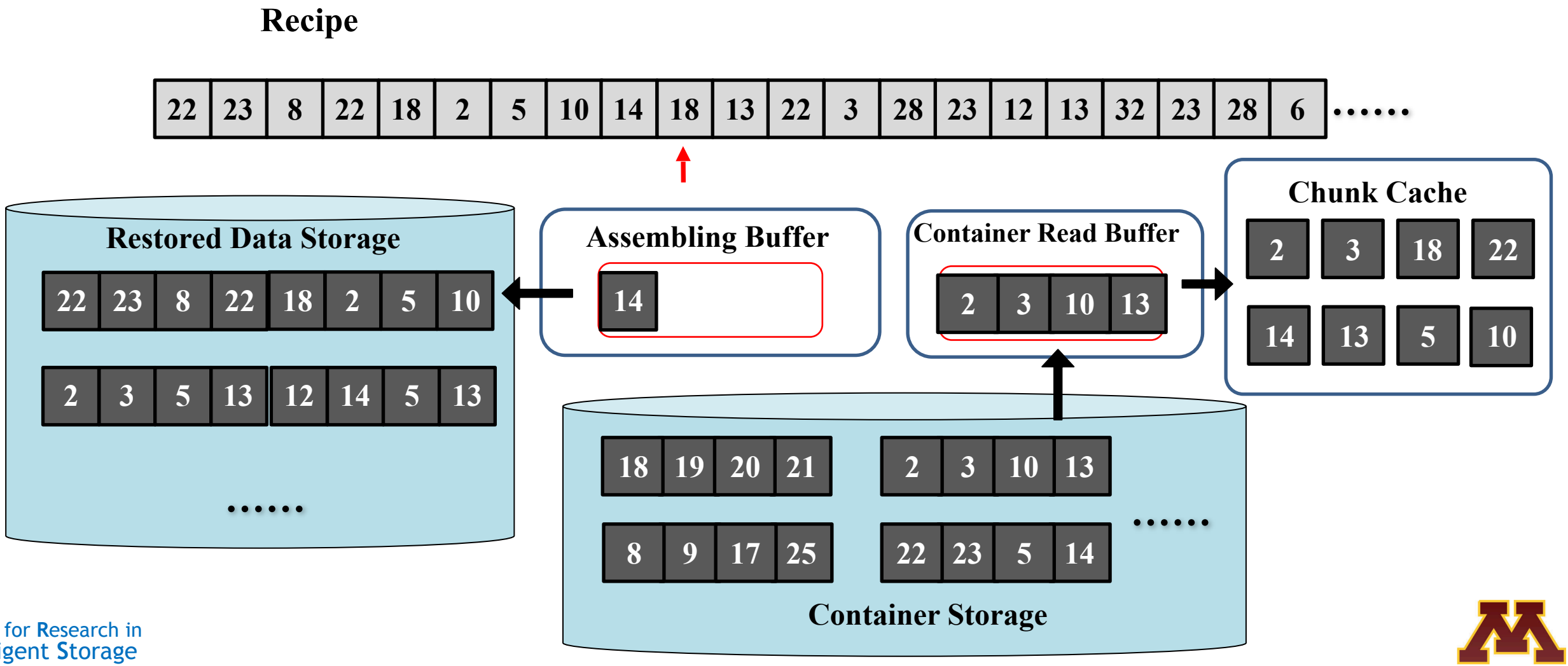
# Restore Process with Chunk-based Caching[2]



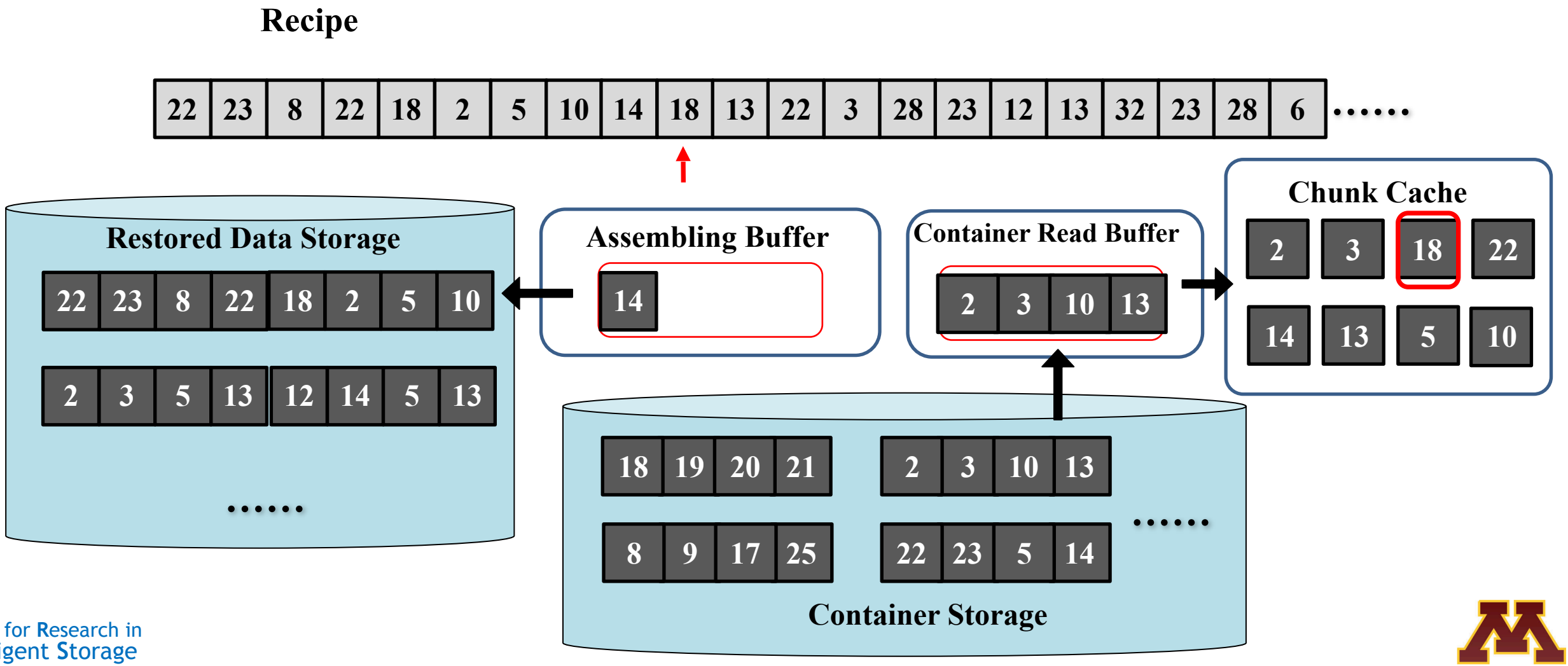
# Restore Process with Chunk-based Caching[2]



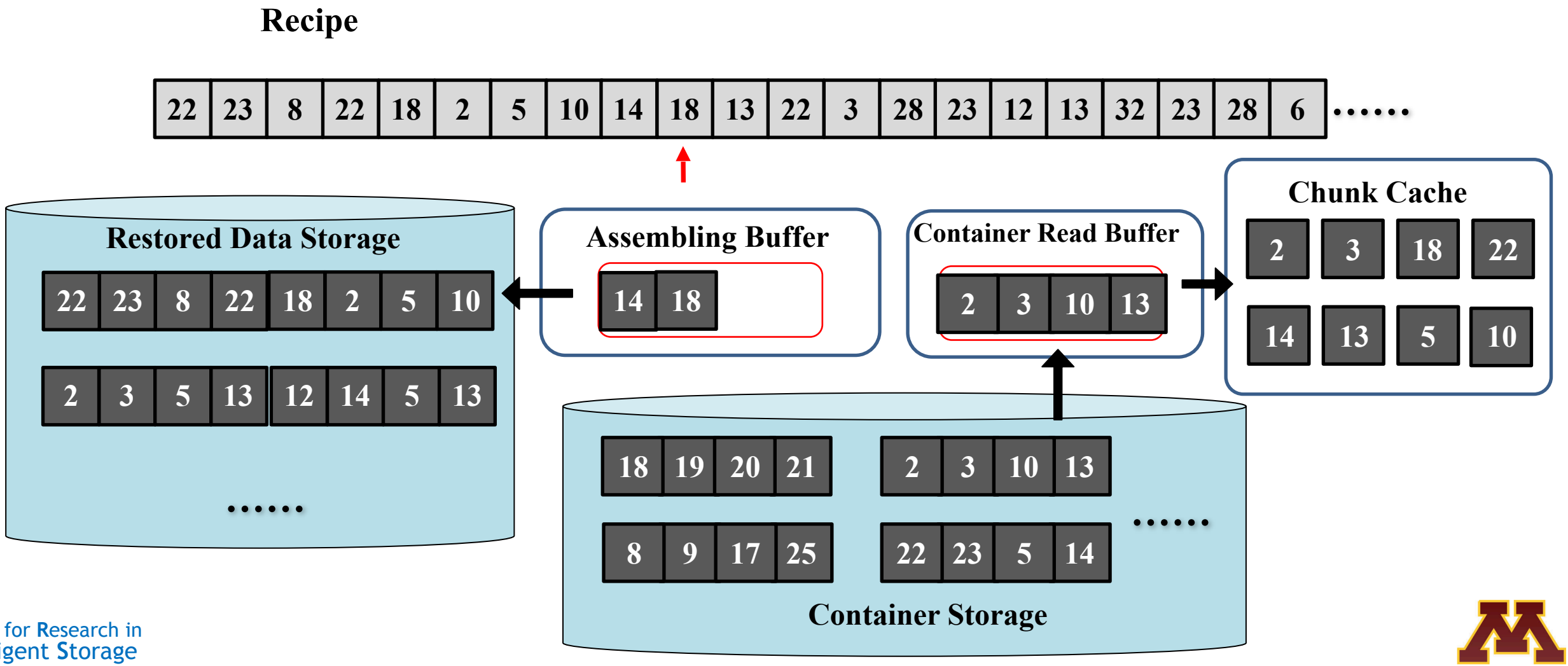
# Restore Process with Chunk-based Caching[2]



# Restore Process with Chunk-based Caching[2]



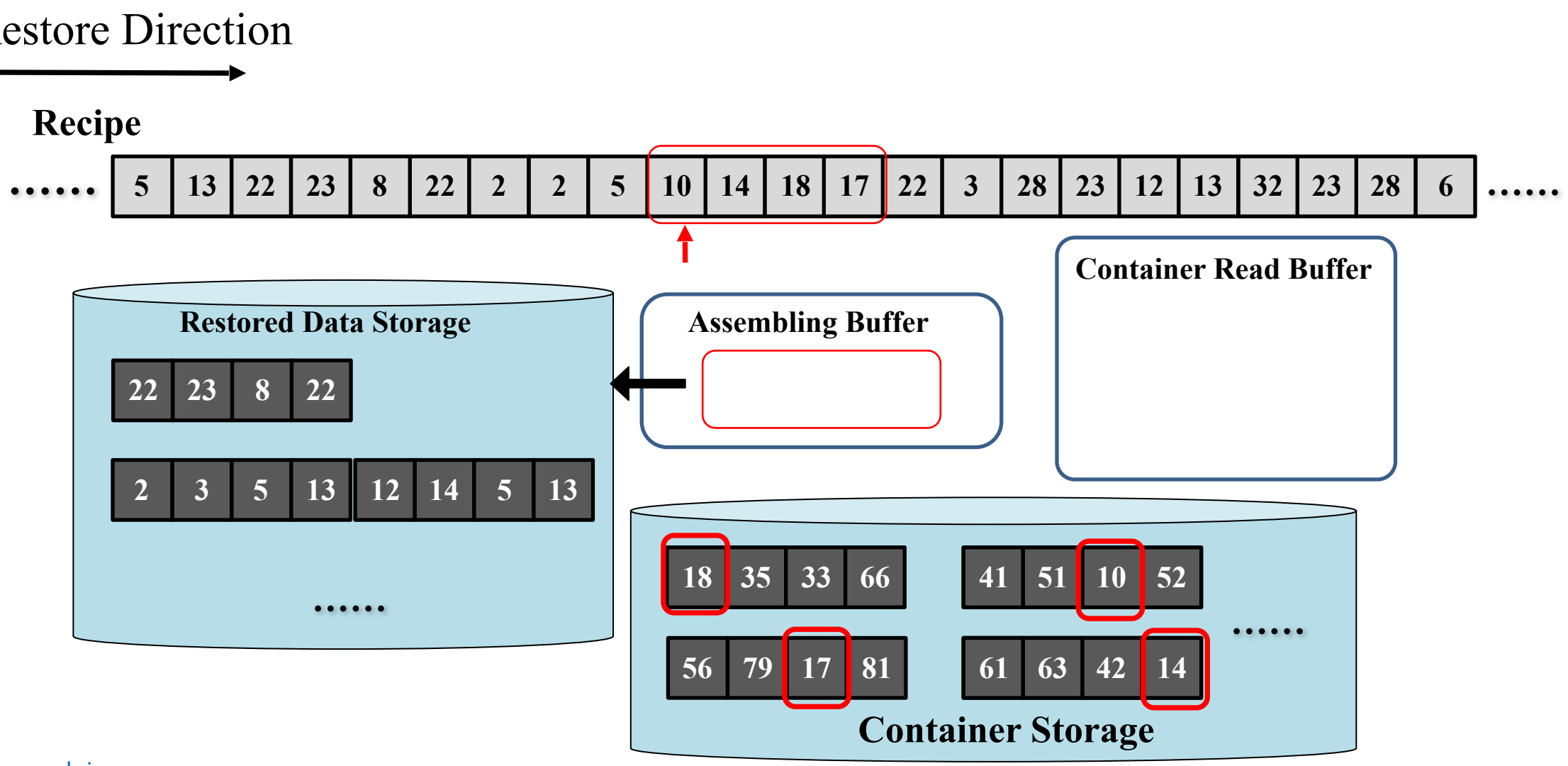
# Restore Process with Chunk-based Caching[2]



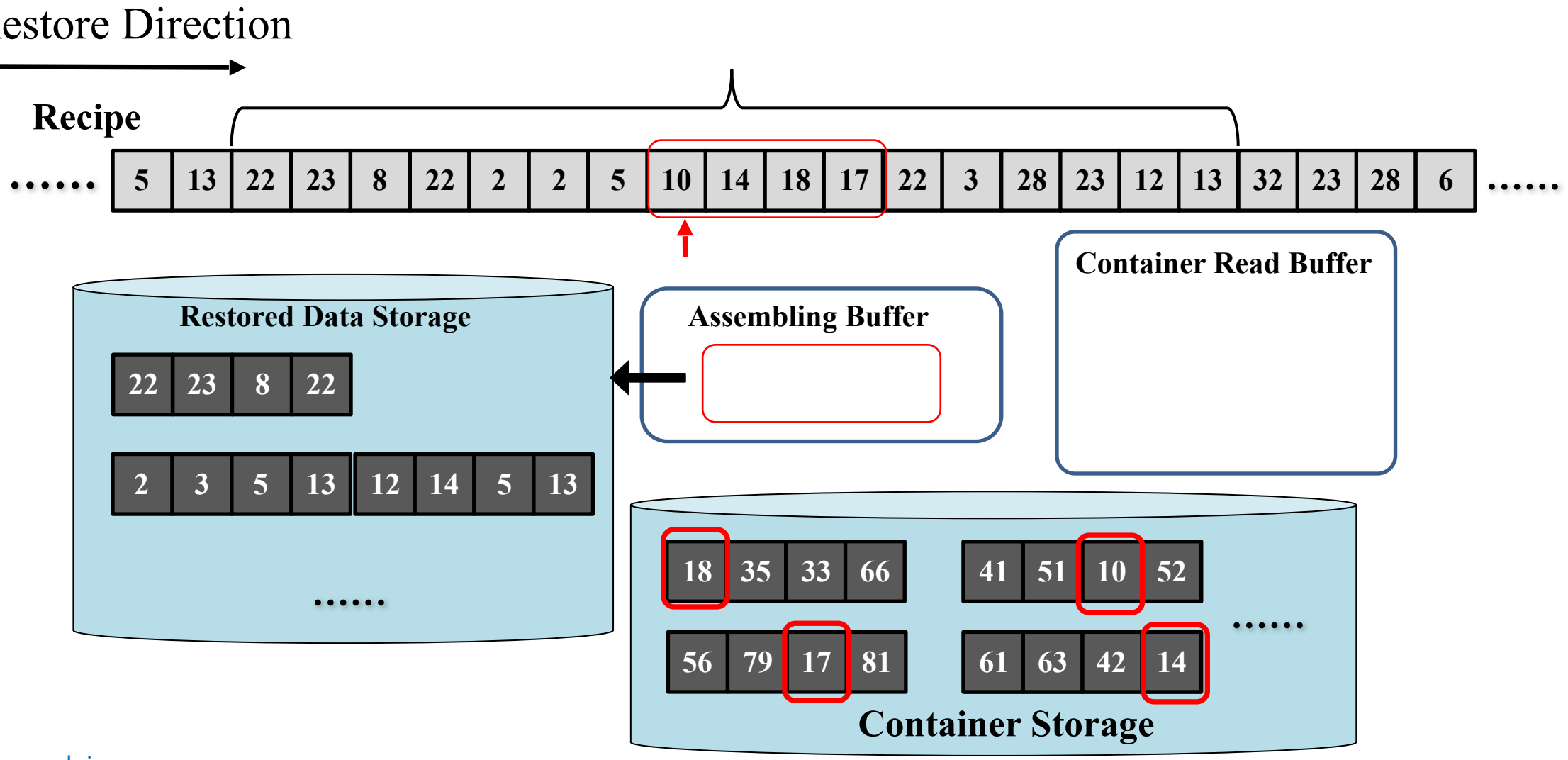
# Agenda

- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

# Limitations of Caching Scheme

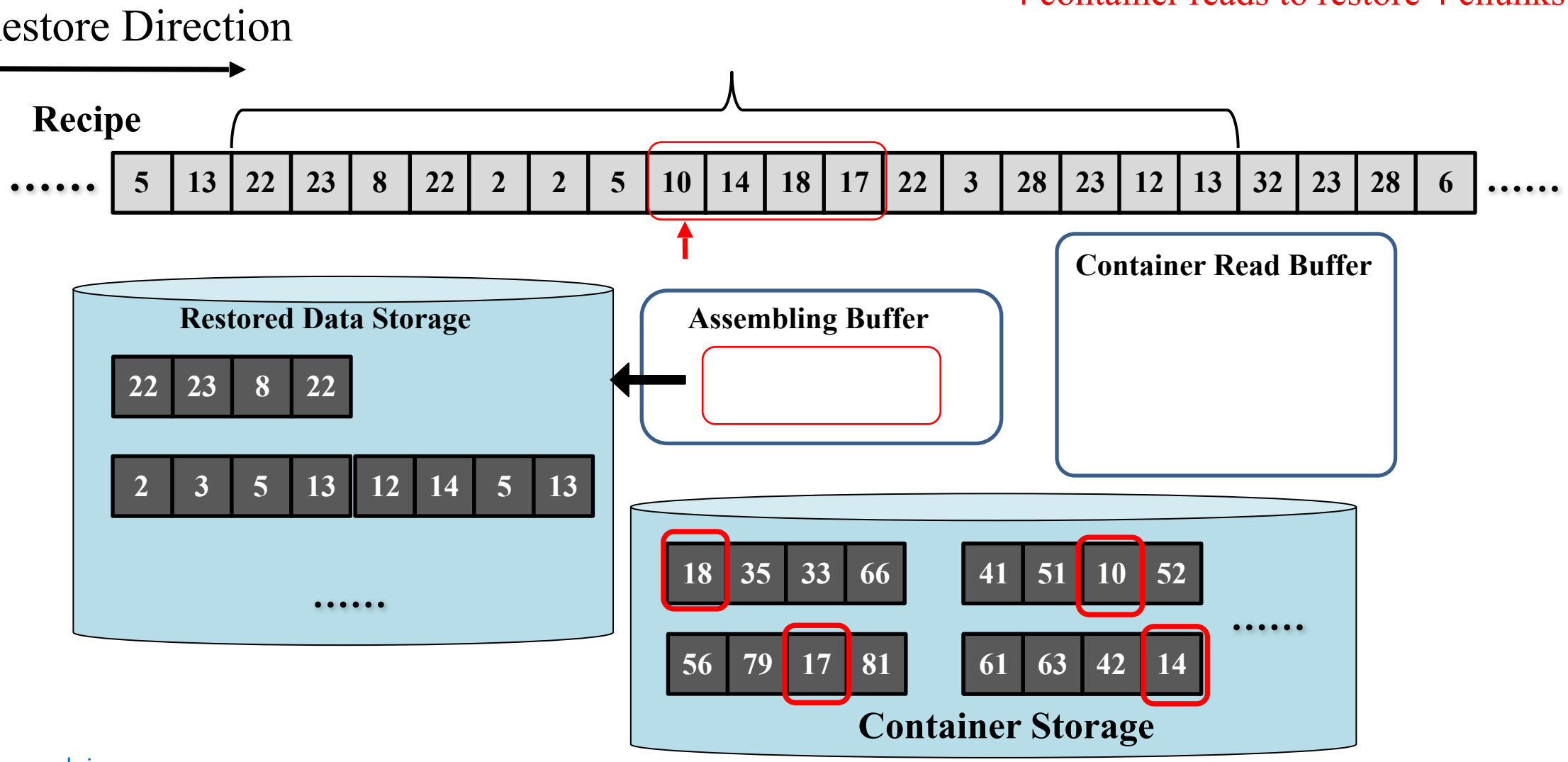


# Limitations of Caching Scheme



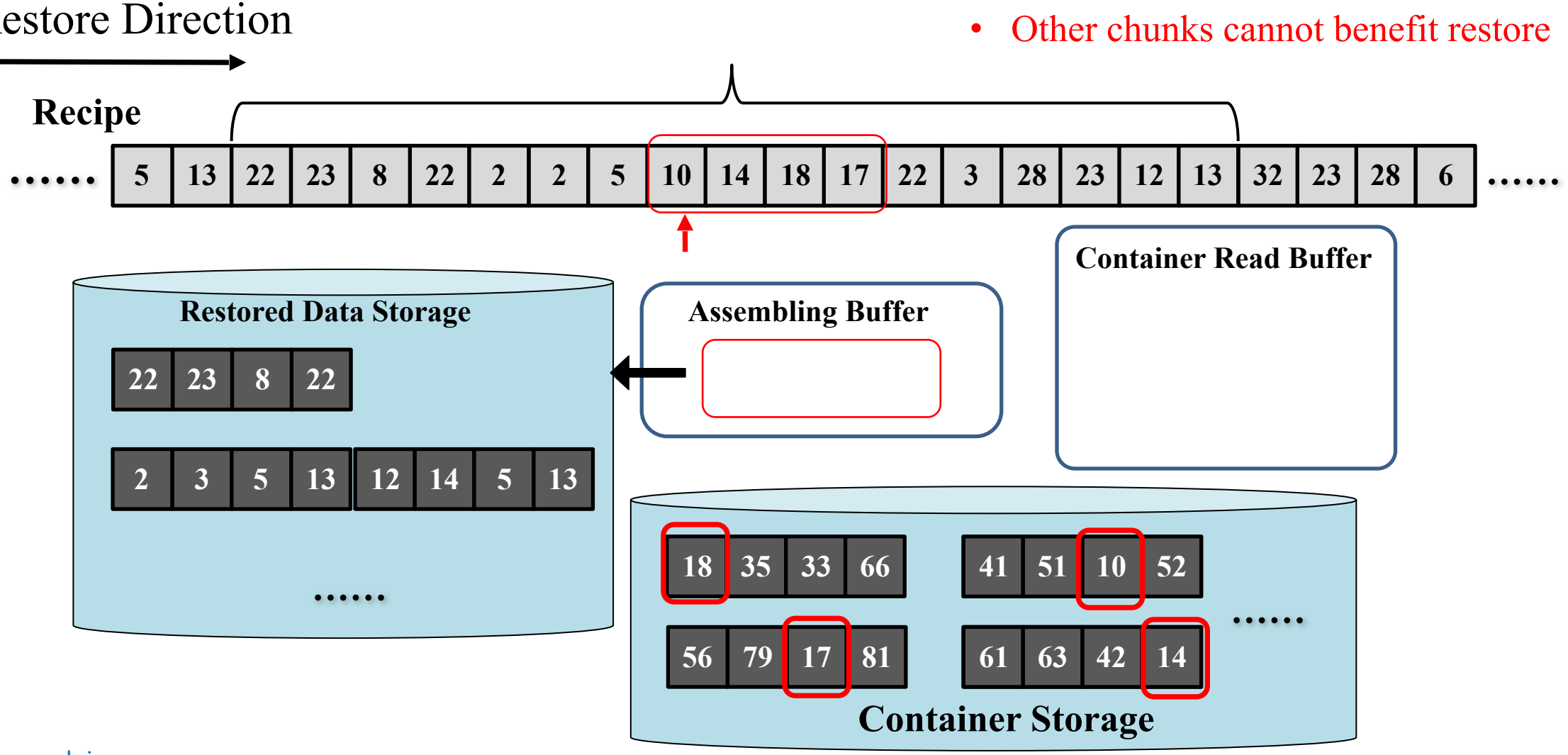
# Limitations of Caching Scheme

- 4 container reads to restore 4 chunks

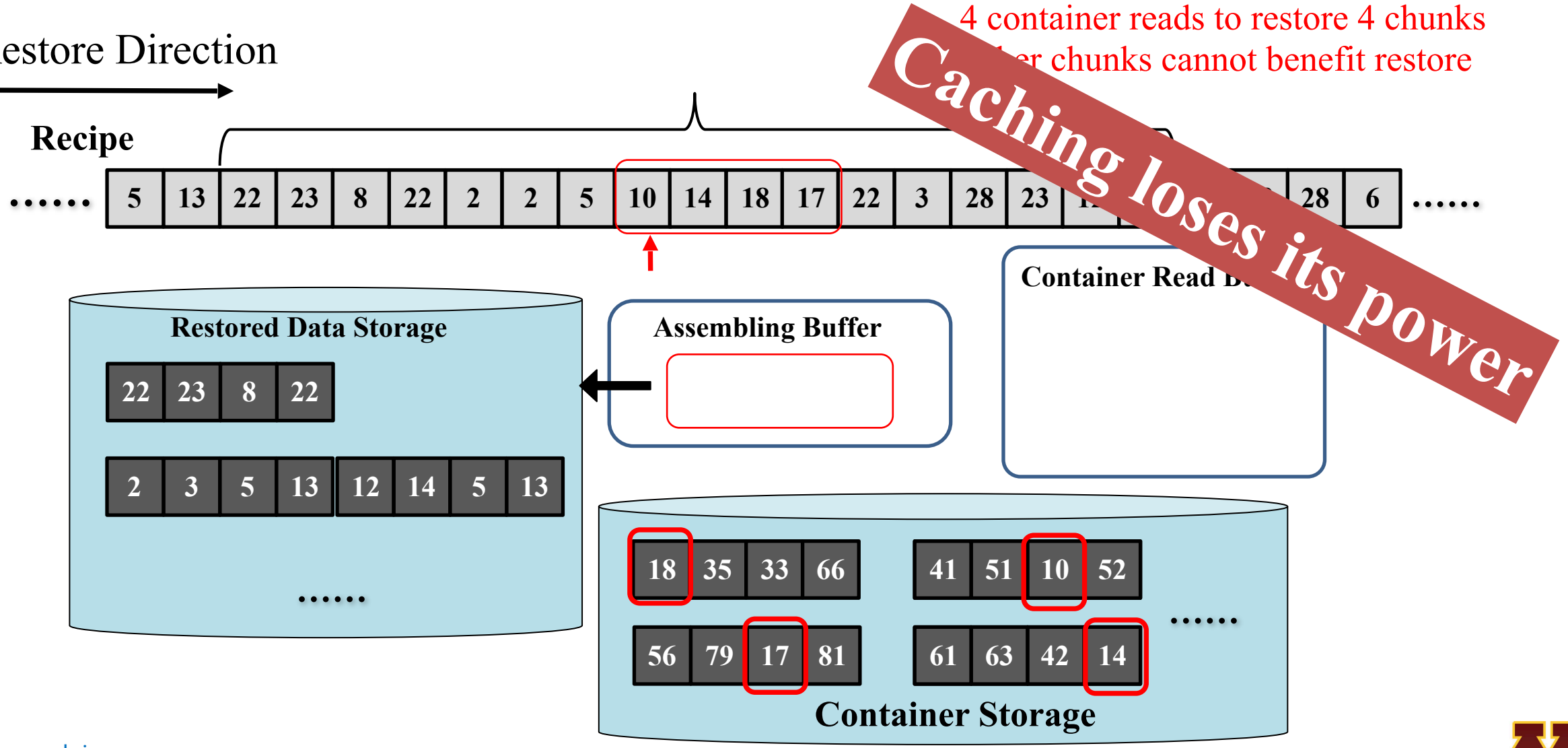


# Limitations of Caching Scheme

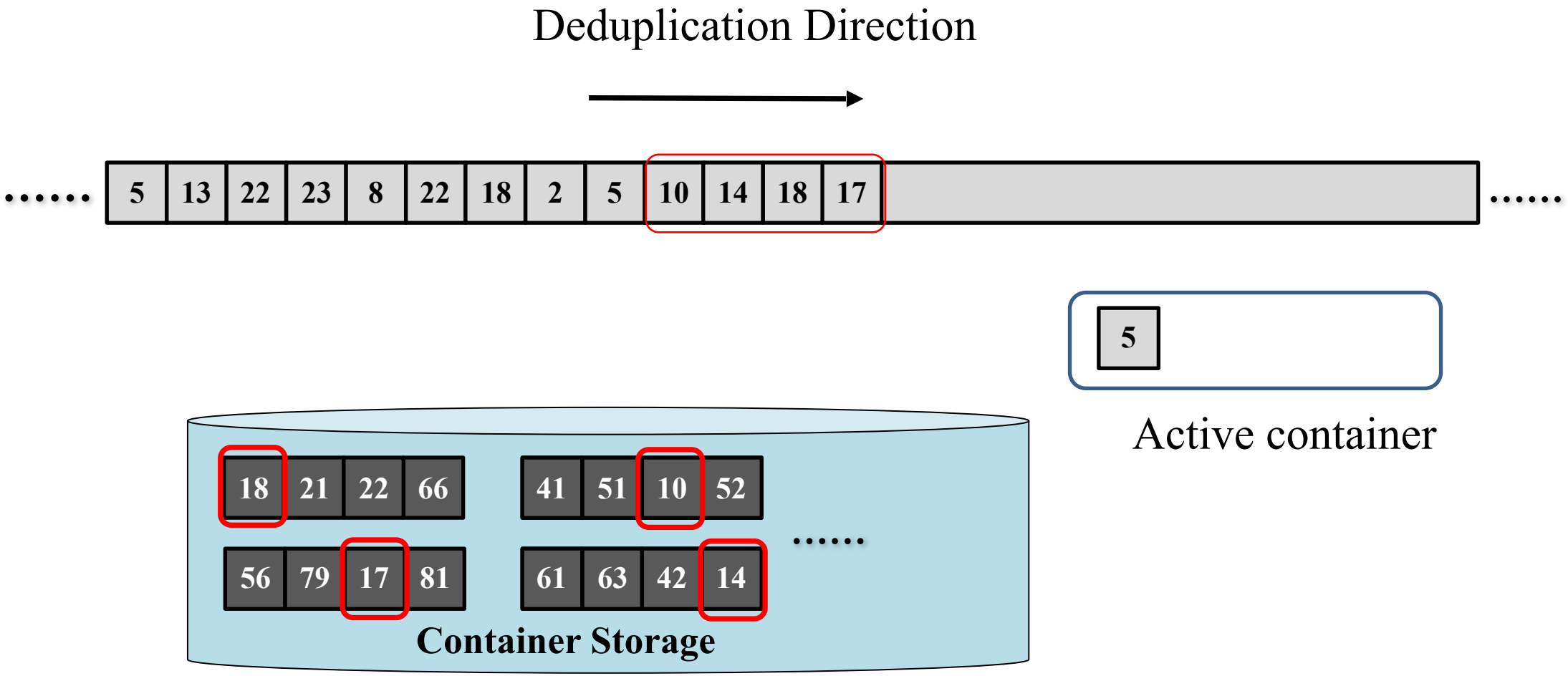
- 4 container reads to restore 4 chunks
- Other chunks cannot benefit restore



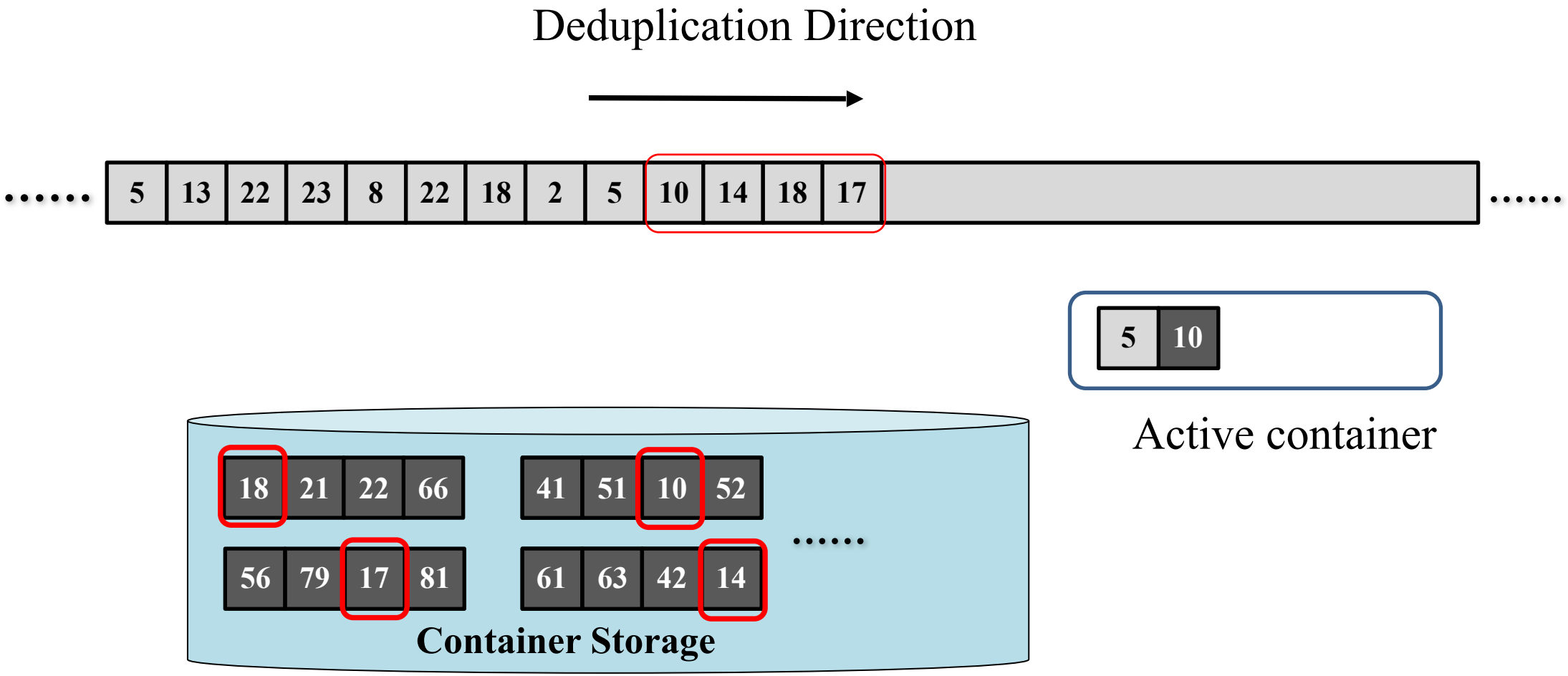
# Limitations of Caching Scheme



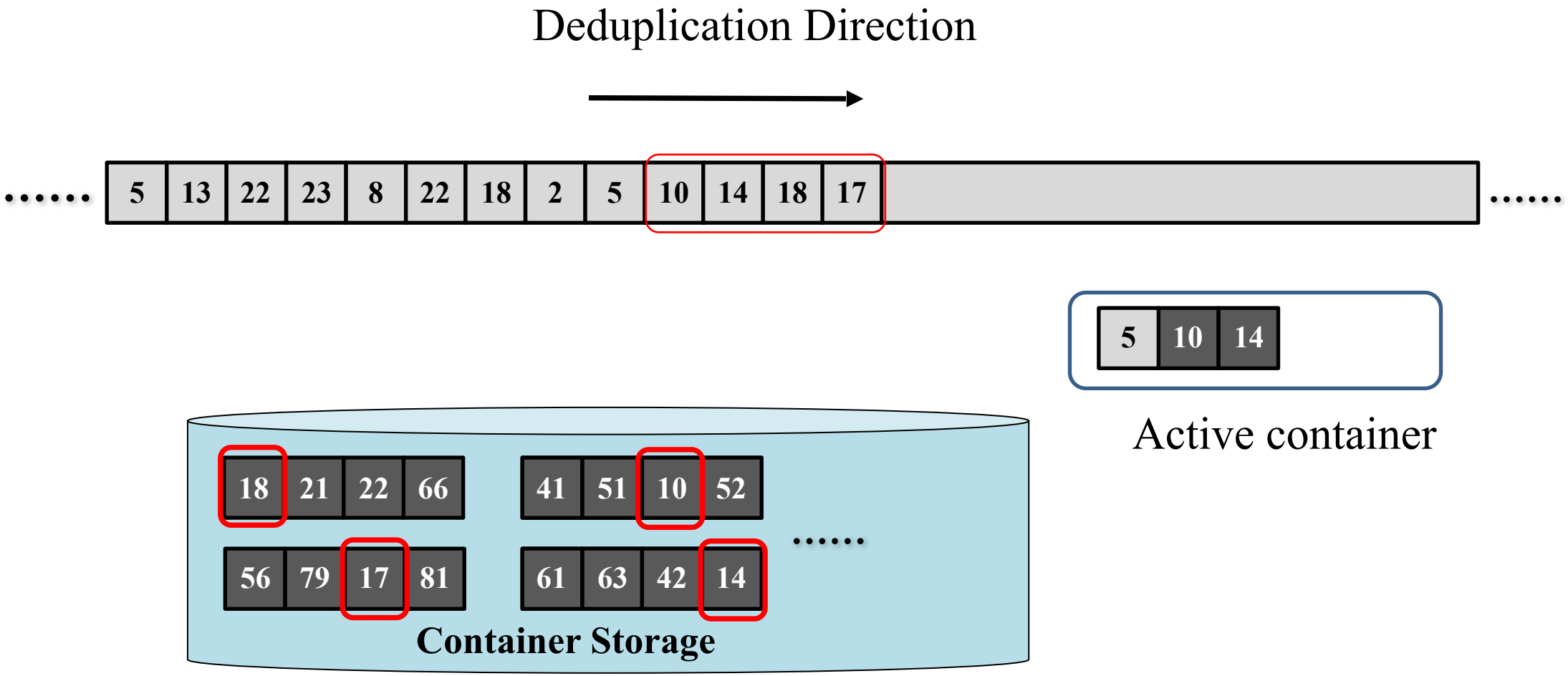
# Data Chunk Rewrite Schemes



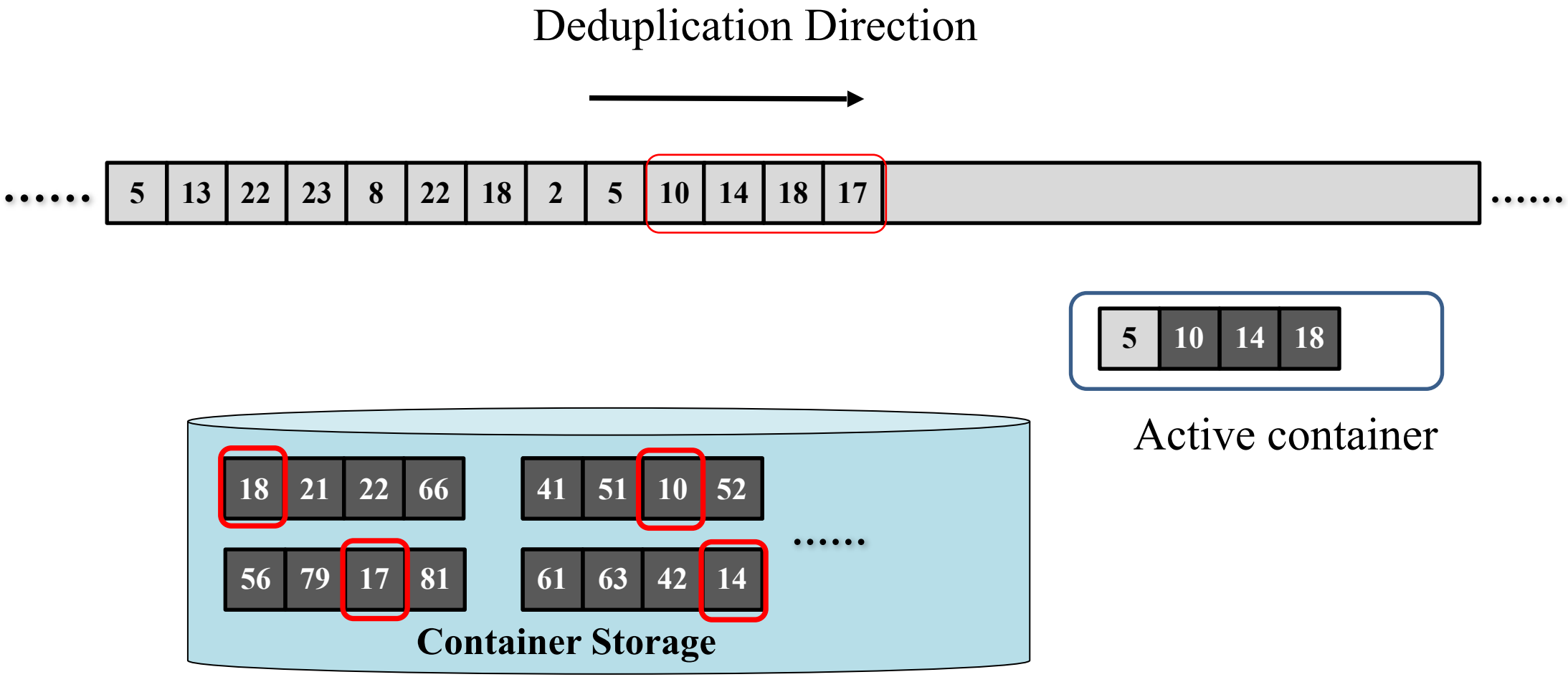
# Data Chunk Rewrite Schemes



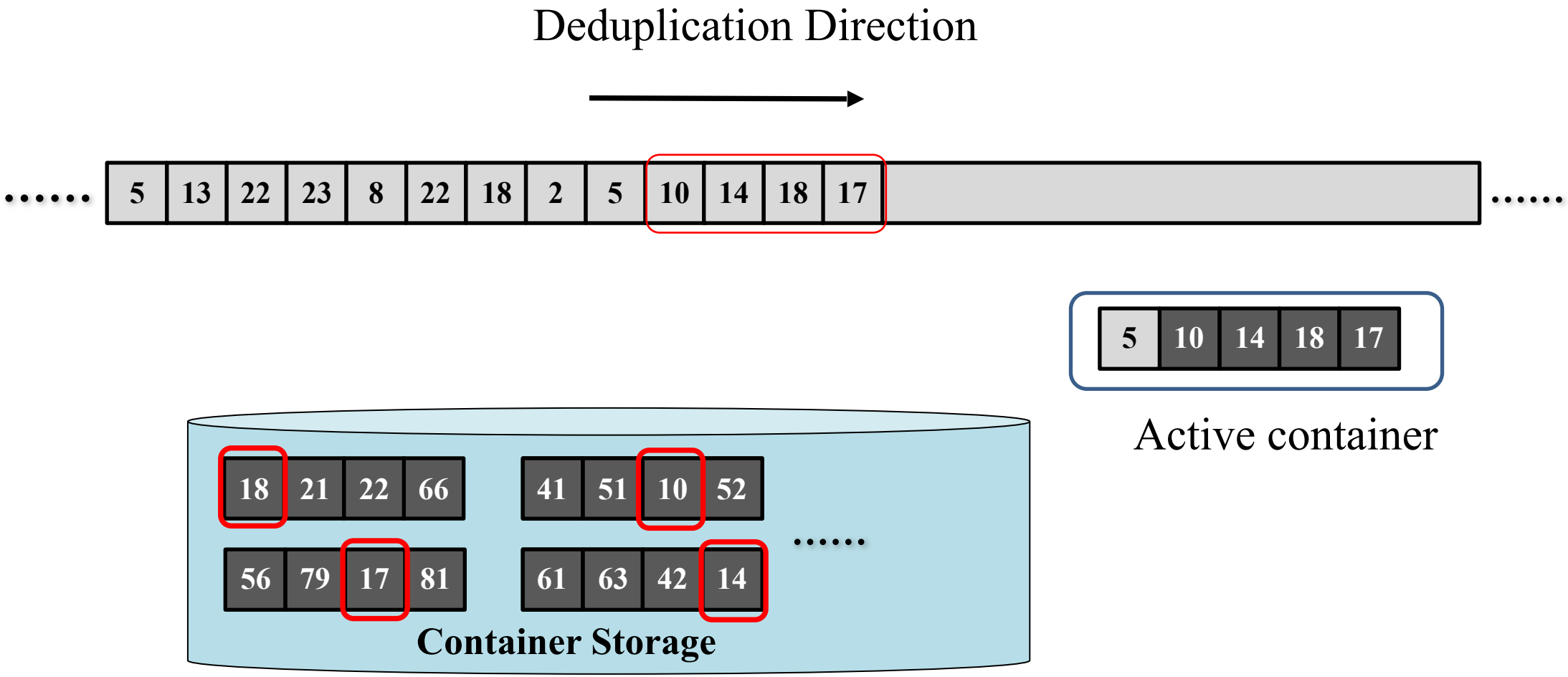
# Data Chunk Rewrite Schemes



# Data Chunk Rewrite Schemes

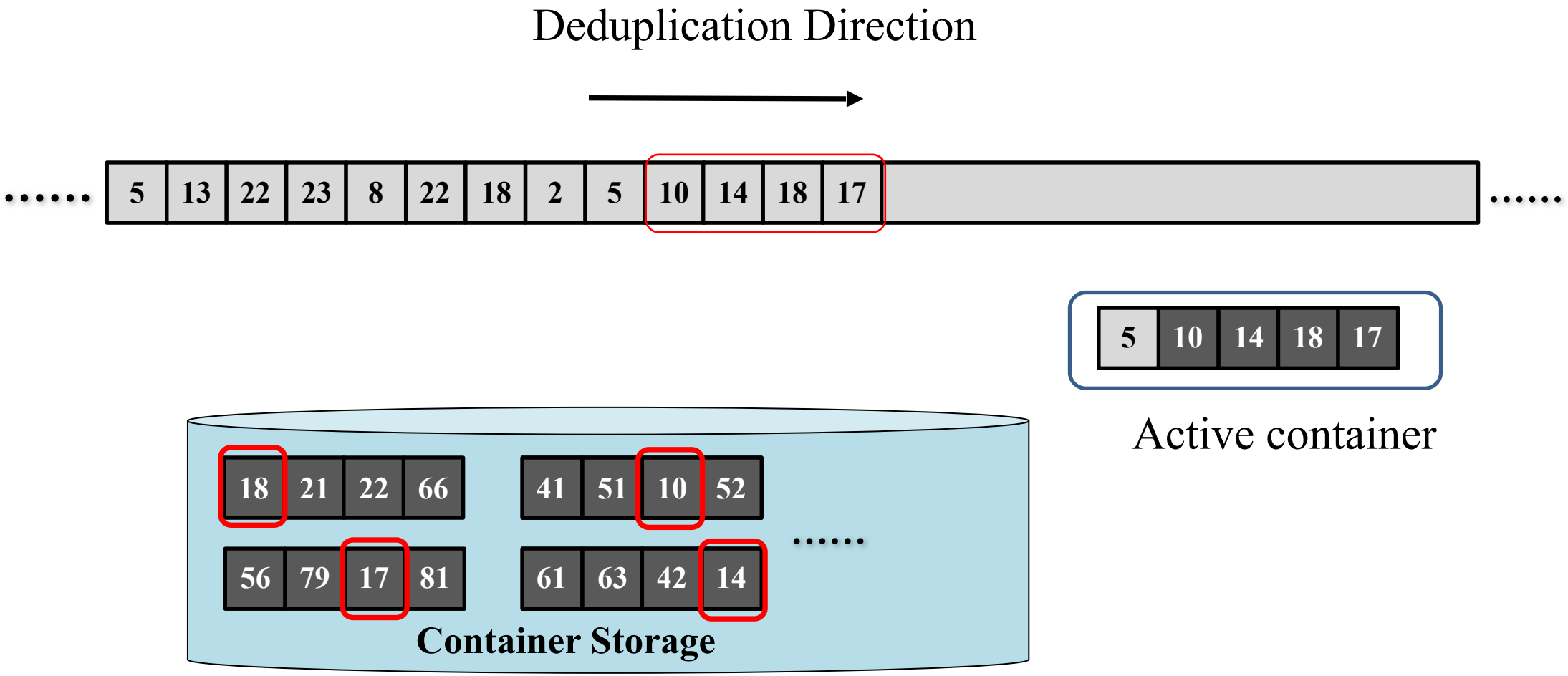


# Data Chunk Rewrite Schemes



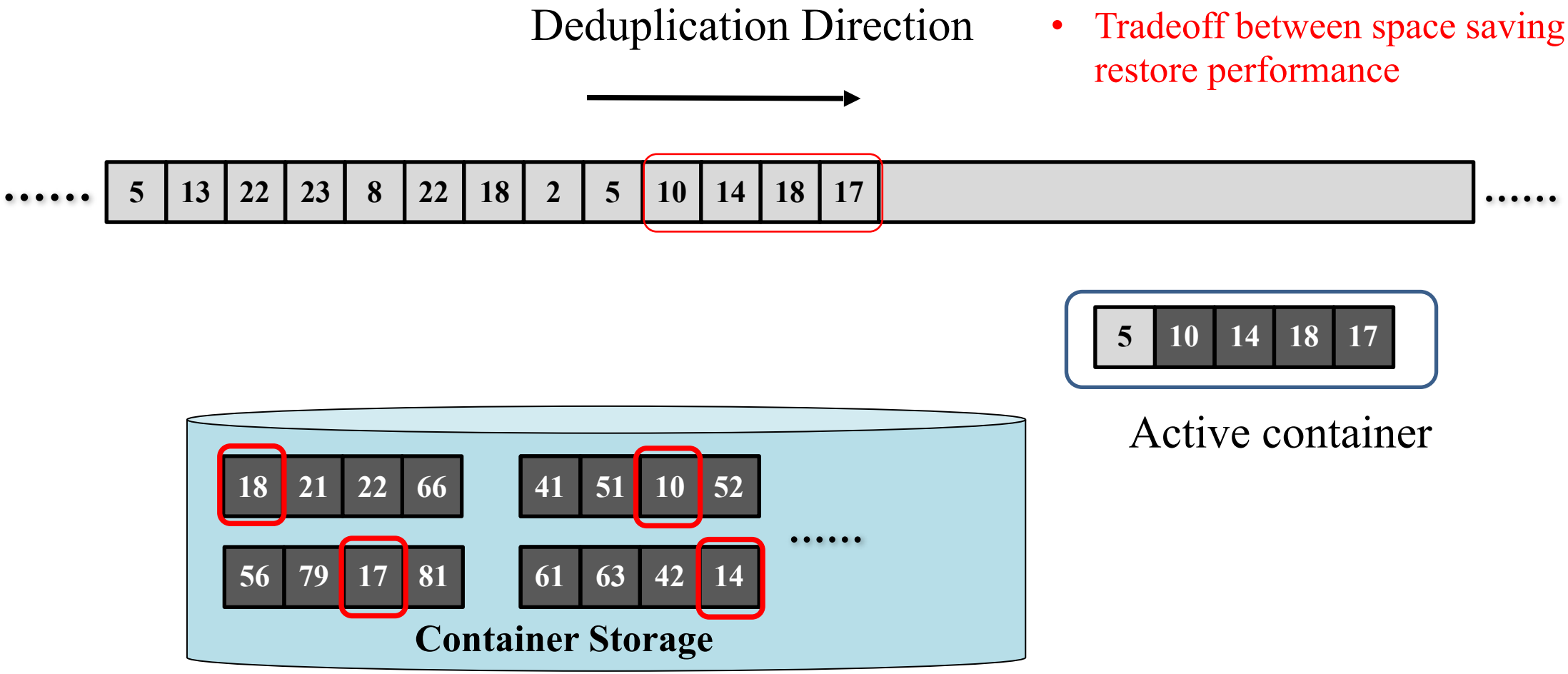
# Data Chunk Rewrite Schemes

- **1** container reads can restore **4** chunks



# Data Chunk Rewrite Schemes

- **1** container reads can restore **4** chunks
- Tradeoff between space saving and restore performance



# Related Work

- Nam *et al.* introduced the **Chunk Fragmentation Level** based data chunk rewrite [4-5]
- The **mismatch level** between byte stream context and data chunk disk context is used to decide the data chunks to be rewrite, which is presented by Kaczmarczyk *et al.* [6]
- Fu *et al.* proposed a **History-Aware Rewriting algorithm (HAR)** which identifies and rewrites sparse containers [7-8]
- Tan *et al.* proposed a **Fine-Grained defragmentation** approach (FGDefrag) to identify and rewrite the fragmental chunks [9]
- **Container capping** was proposed by Lillibridge *et al.* [3]
- ……

# Challenges of Rewrite

- Rewrite sacrifices deduplication ratio (space saving), how to make **better tradeoffs** between **deduplication ratio** and **restore performance** is challenging;

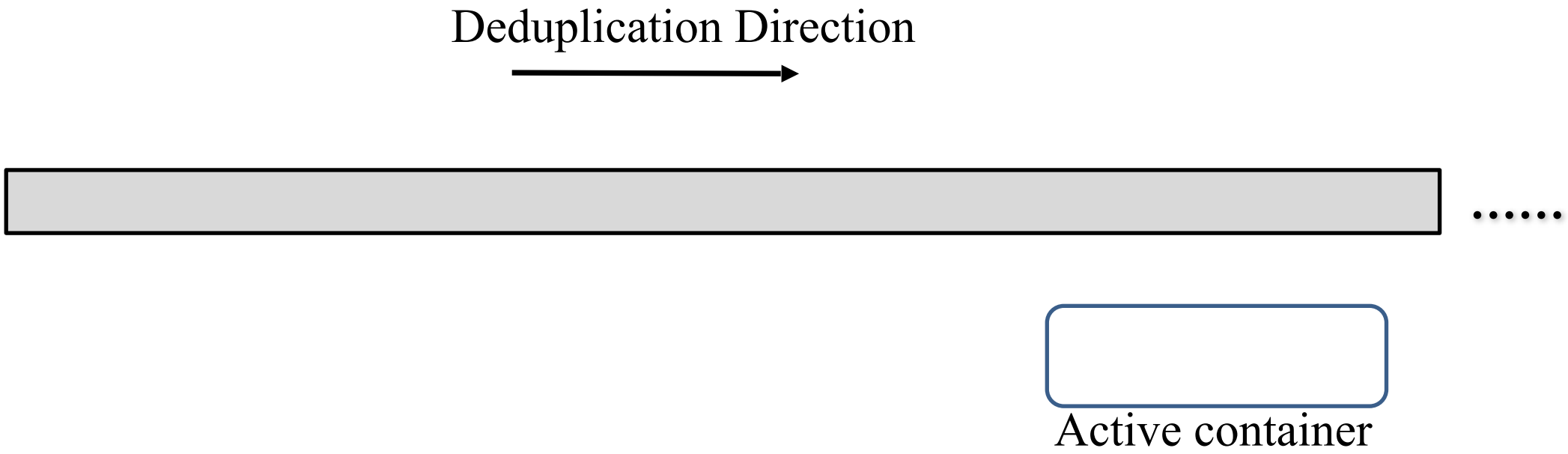
- Rewrite is done during the deduplication process, information is limited. Most related studies are based on the **past statistic information** to make the decision. How to decide the data chunks to be rewritten with limited information is challenging;

- The **restore caching effectiveness** should be considered during the dedup to reduce unnecessary rewrites. However, how to integrate caching with rewrite is not clearly investigated

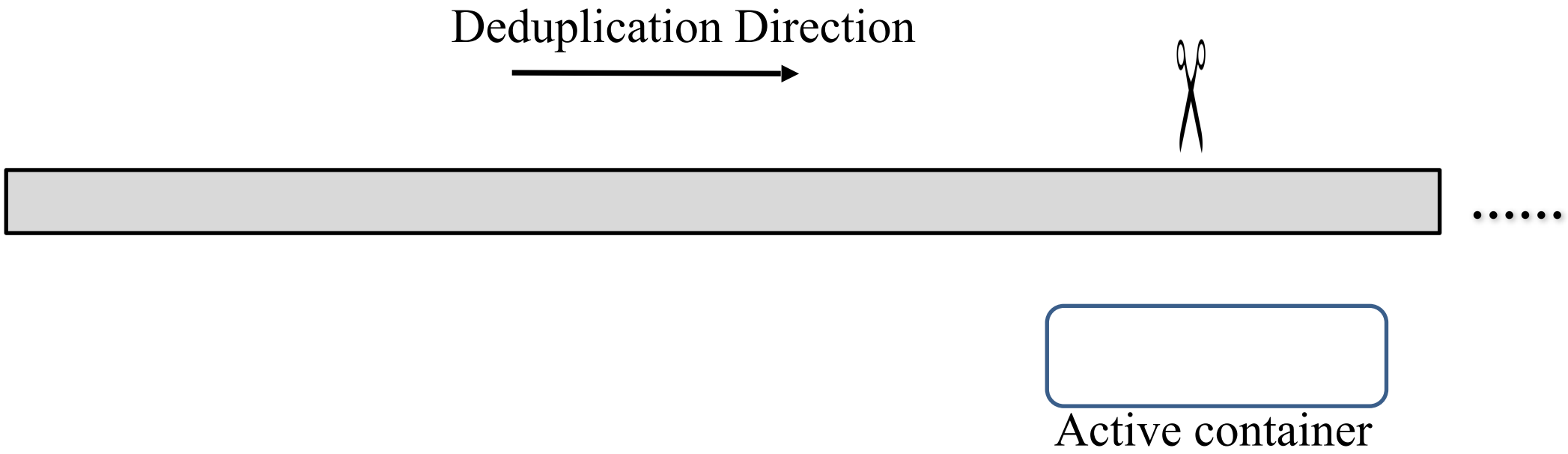
# Agenda

- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

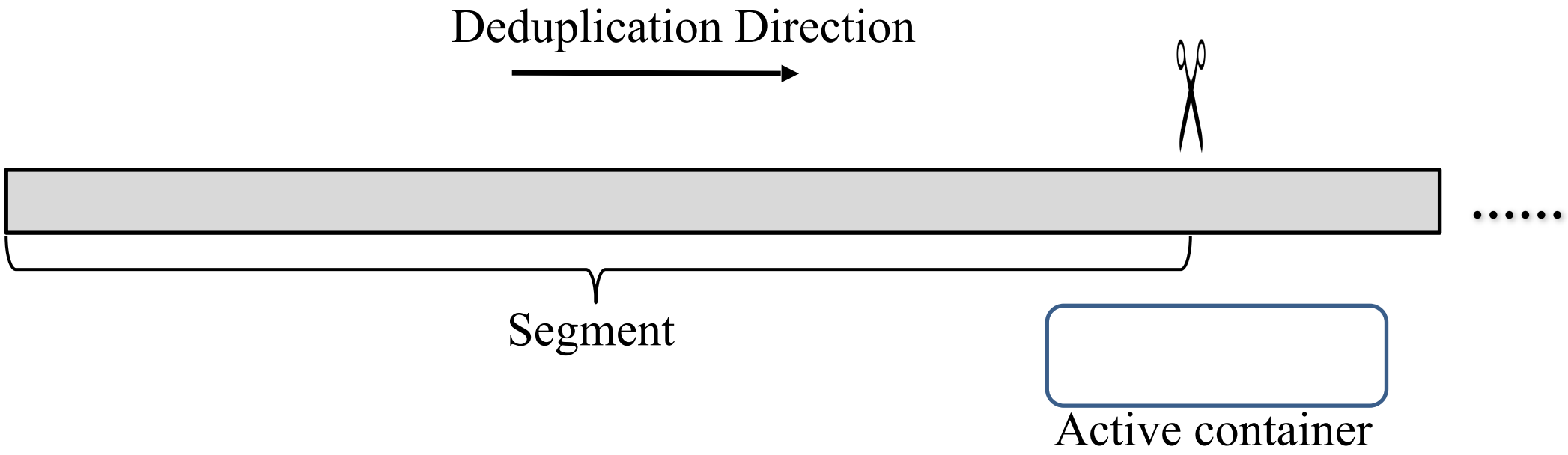
# Introduction of Container Capping[3]



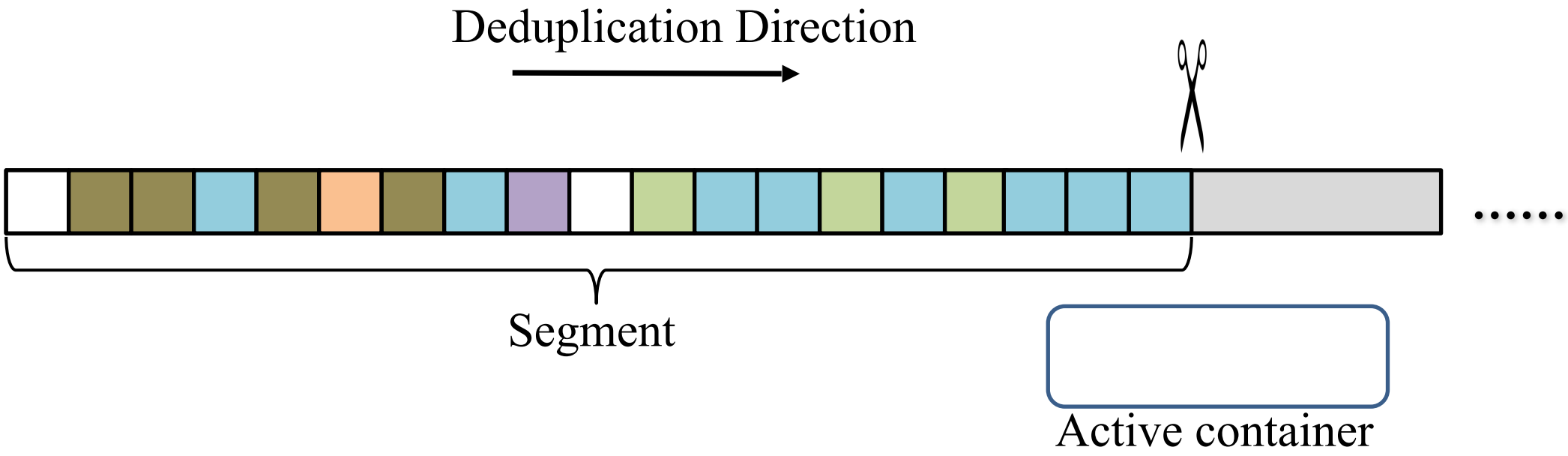
# Introduction of Container Capping[3]



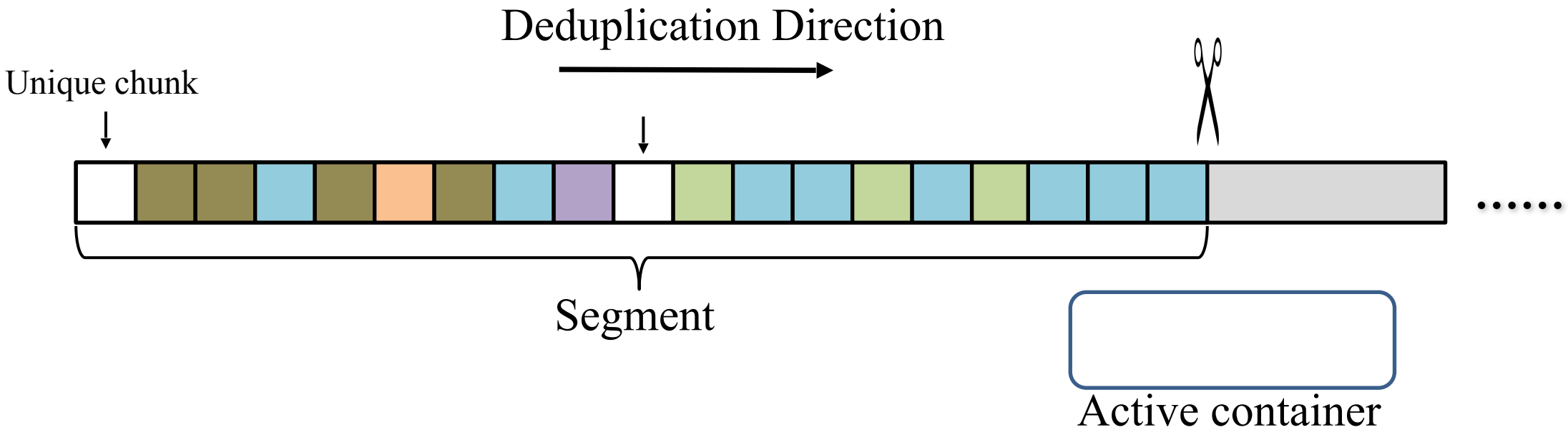
# Introduction of Container Capping[3]



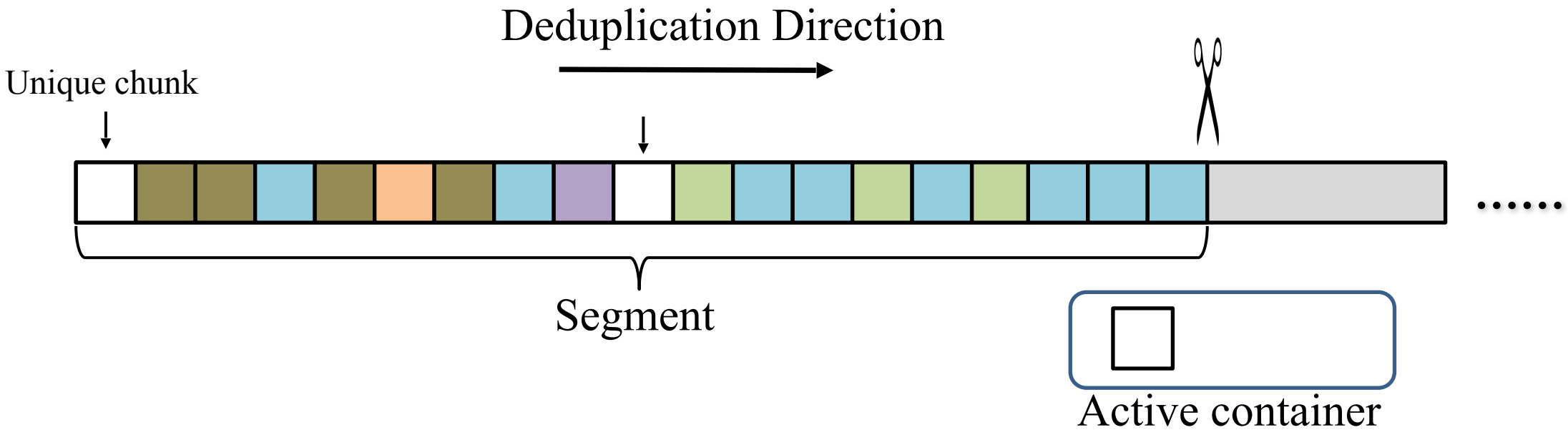
# Introduction of Container Capping[3]



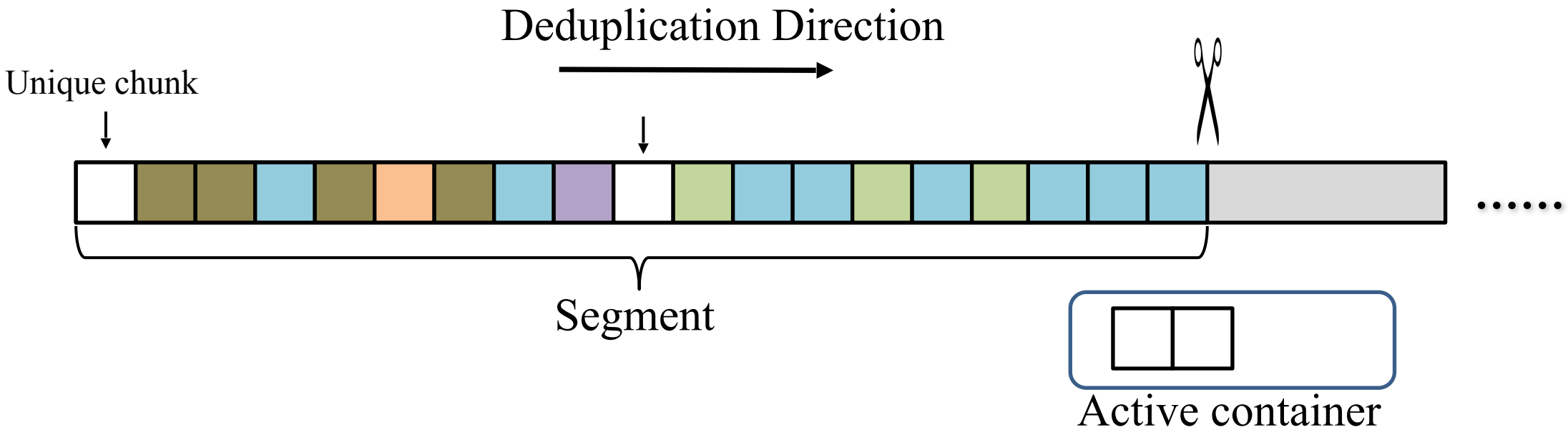
# Introduction of Container Capping[3]



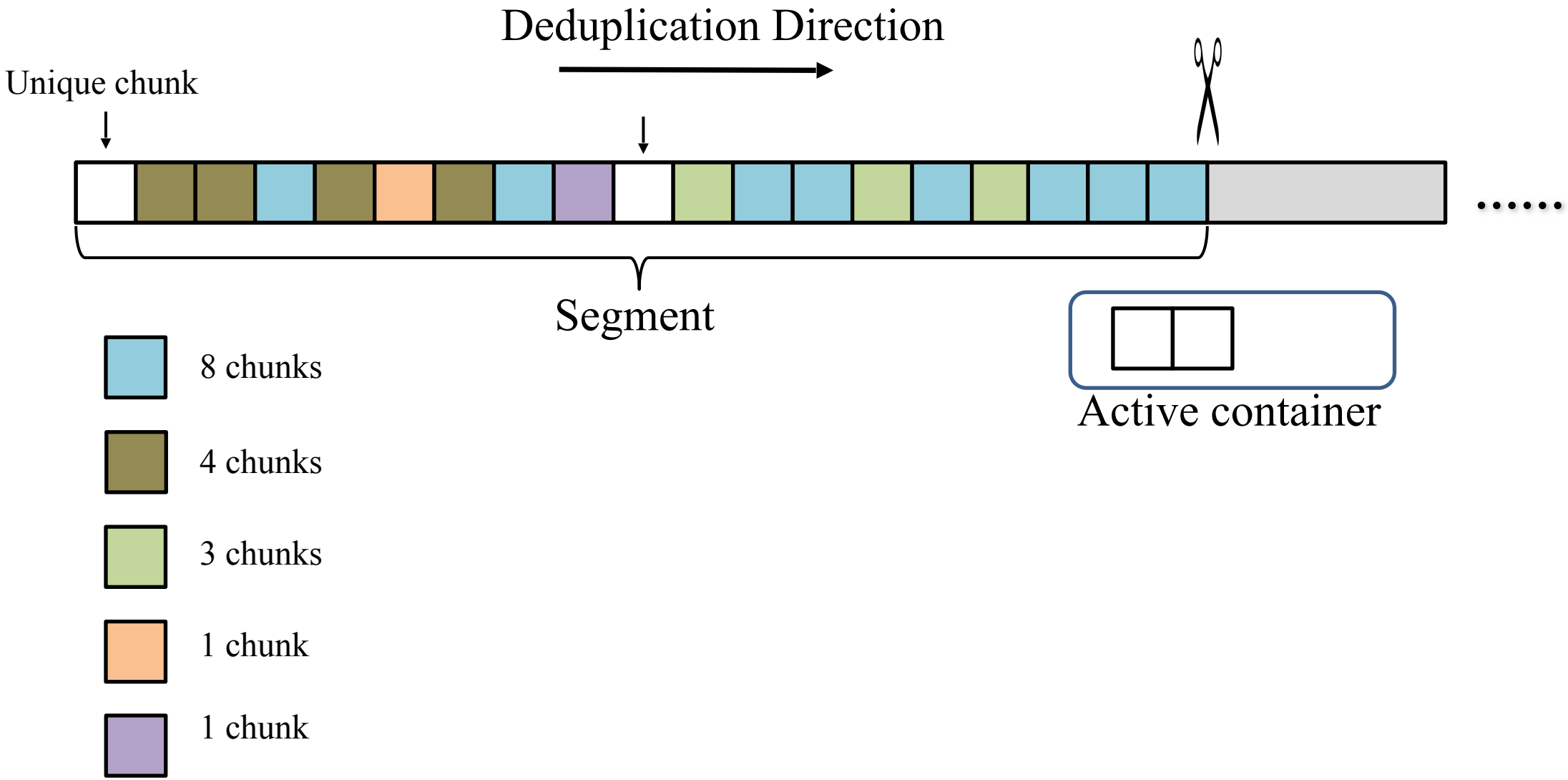
# Introduction of Container Capping[3]



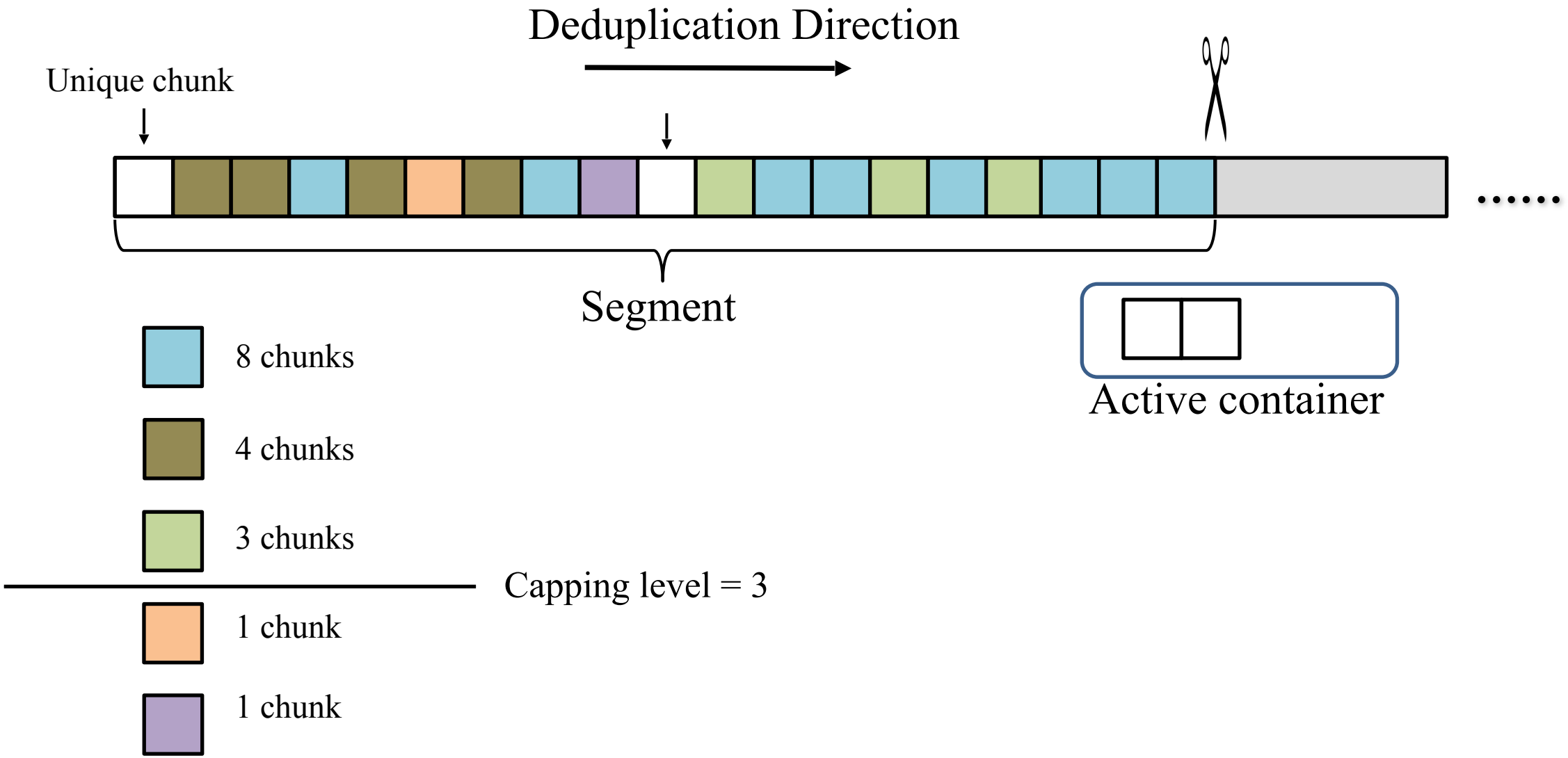
# Introduction of Container Capping[3]



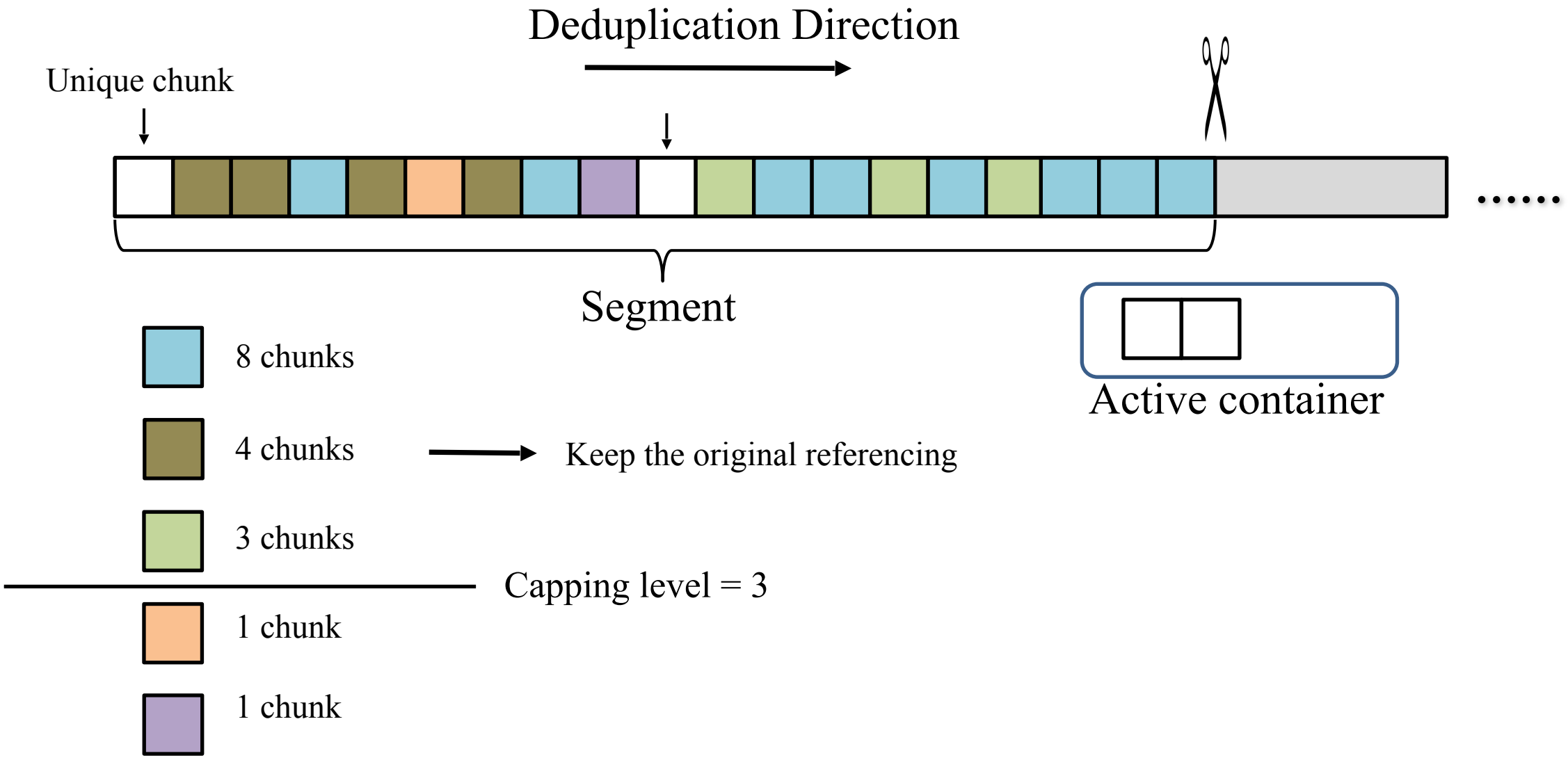
# Introduction of Container Capping[3]



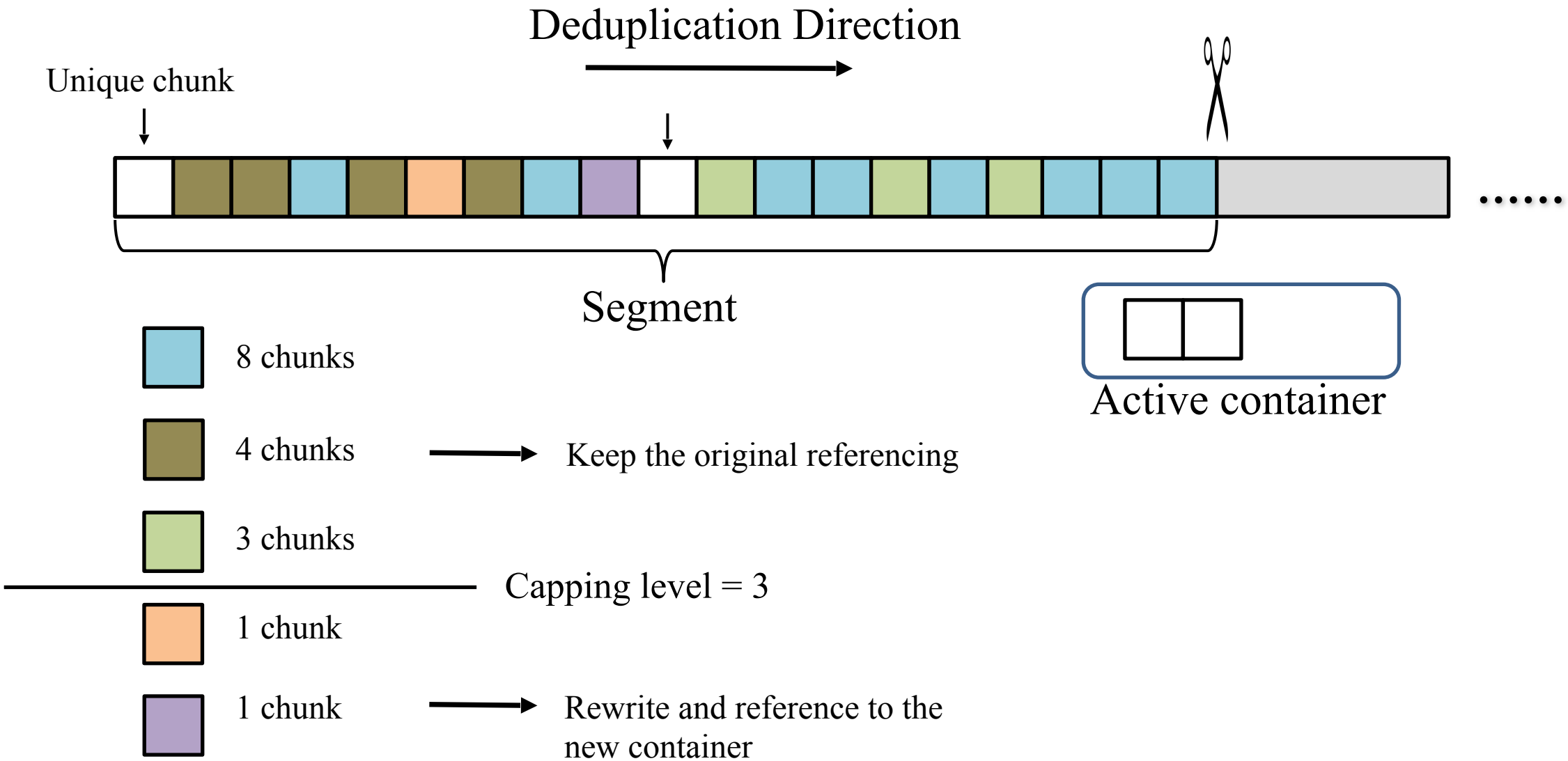
# Introduction of Container Capping[3]



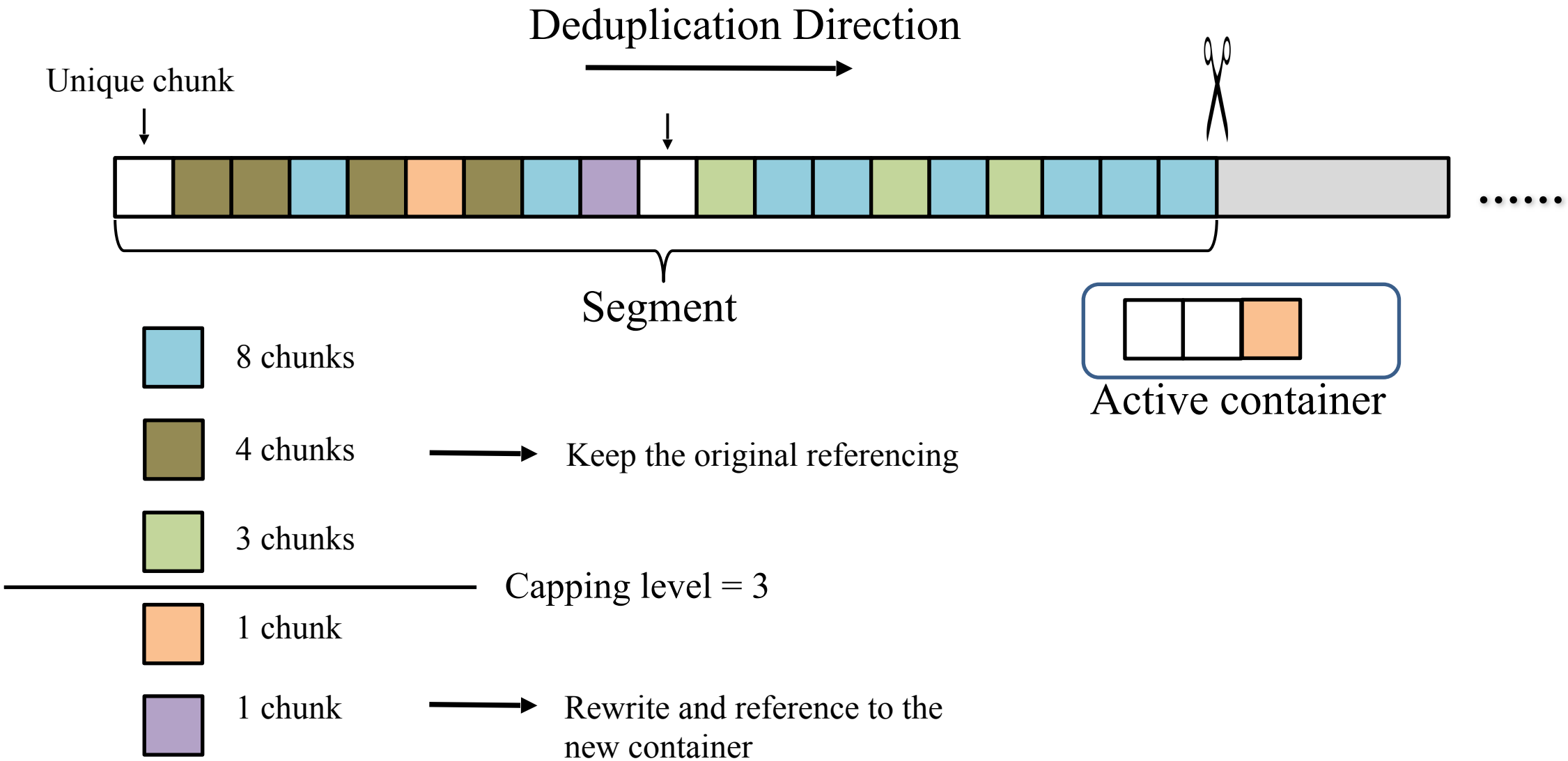
# Introduction of Container Capping[3]



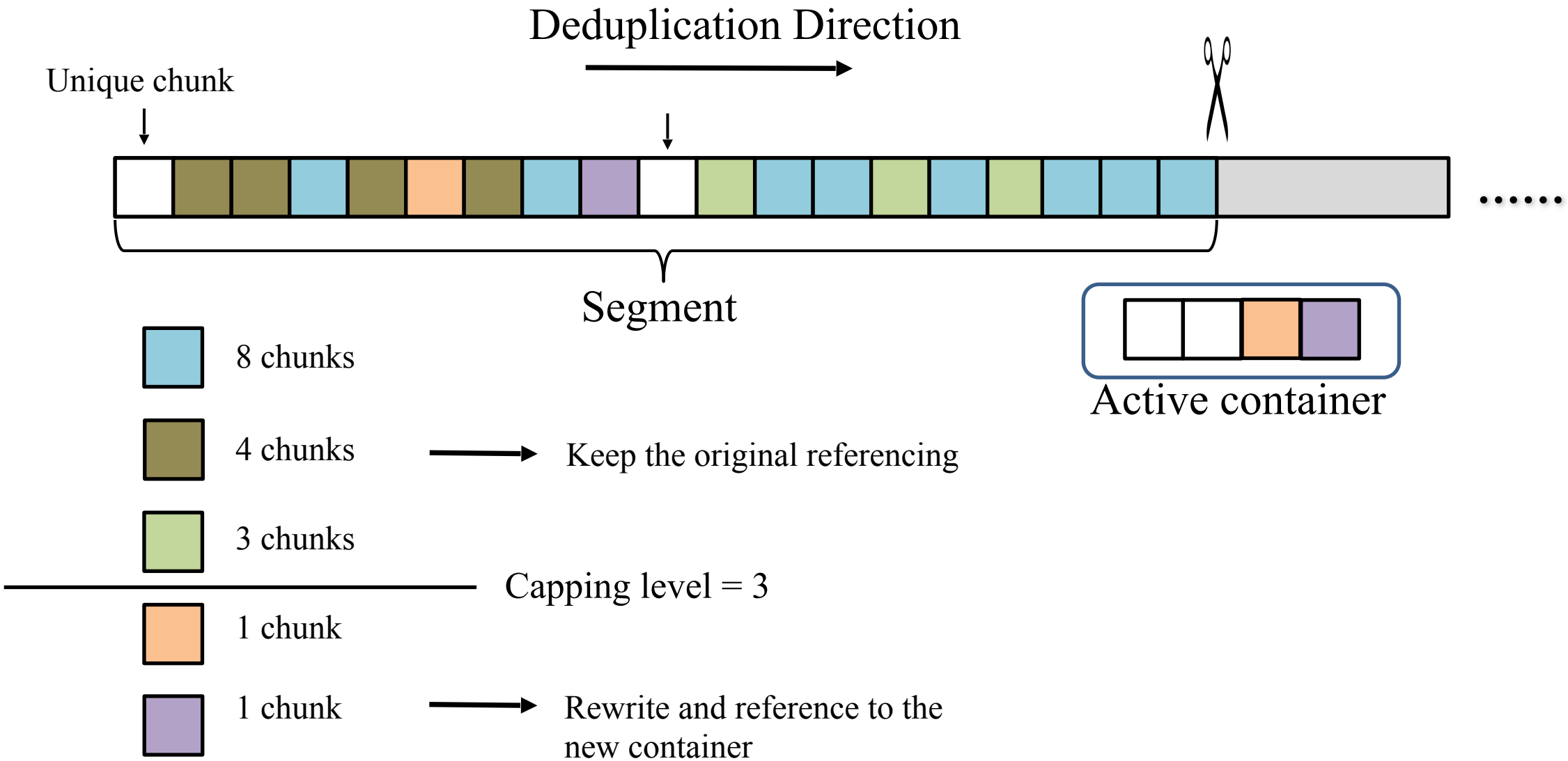
# Introduction of Container Capping[3]



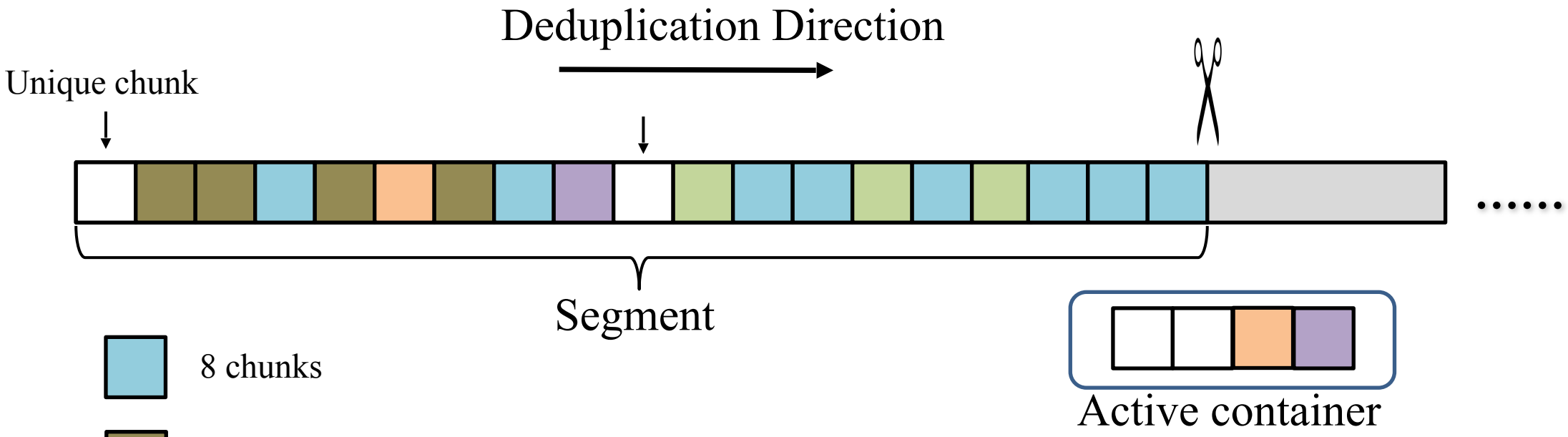
# Introduction of Container Capping[3]



# Introduction of Container Capping[3]



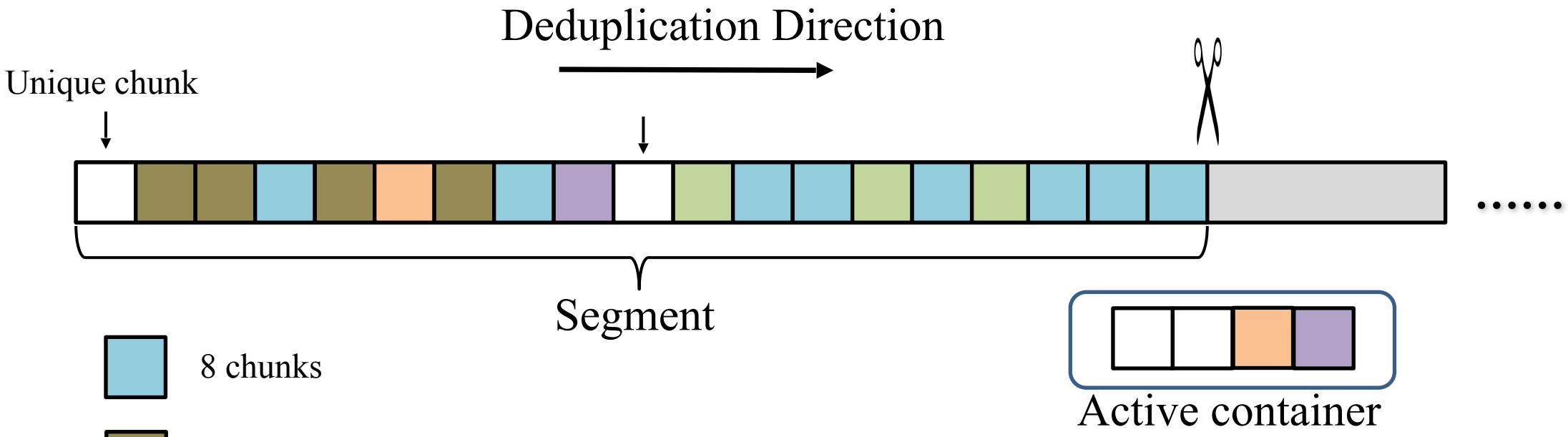
# Introduction of Container Capping[3]




**Pros:**

1) simple and efficient
2) guarantee the higher bound of container reads

# Introduction of Container Capping[3]



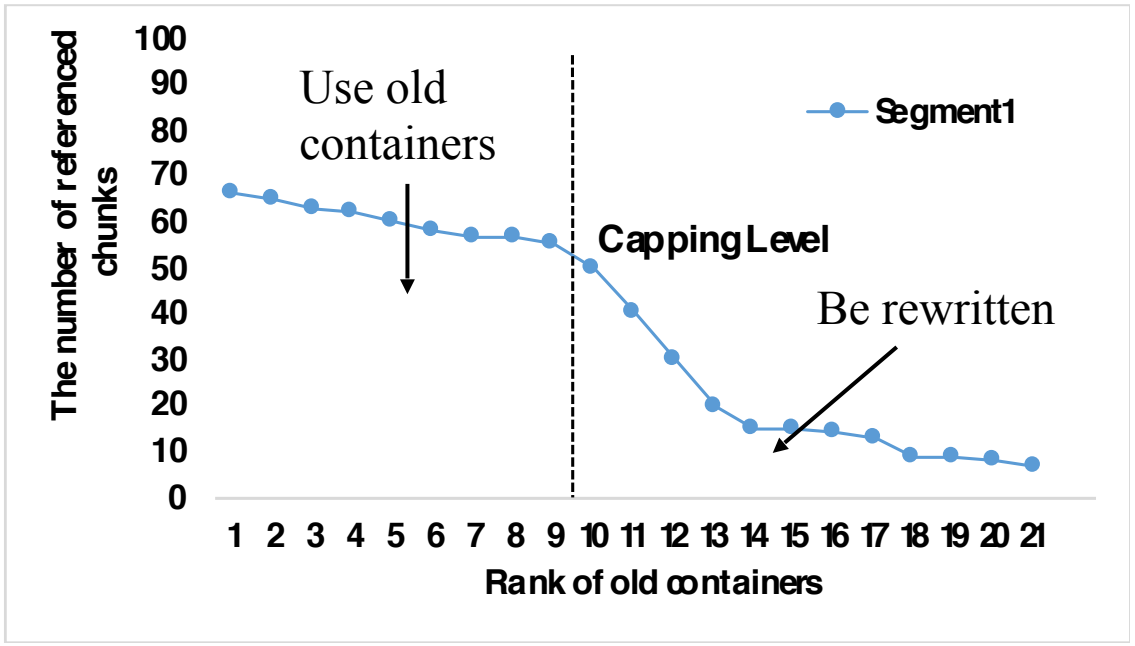

**Pros:**

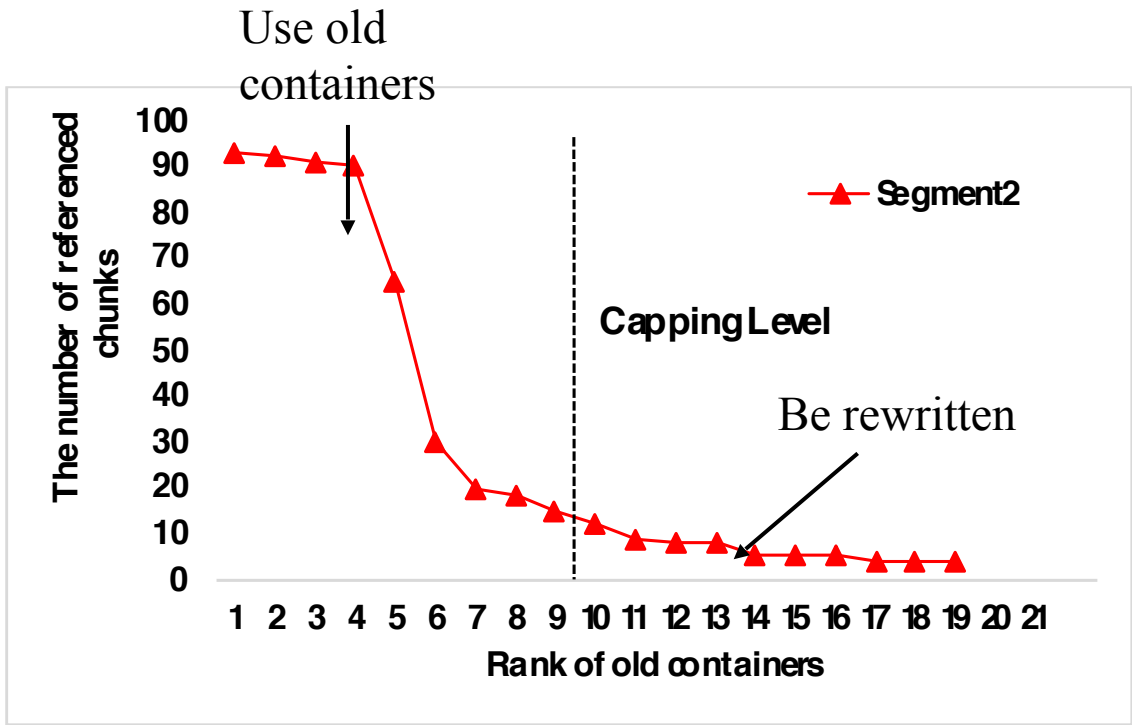
1) simple and efficient
2) guarantee the higher bound of container reads

**Cons:**

1) a fixed capping level cannot adapt to the workload
2) might make wrong rewrite decision due to the segment cut
3) deduplication ratio is not guaranteed

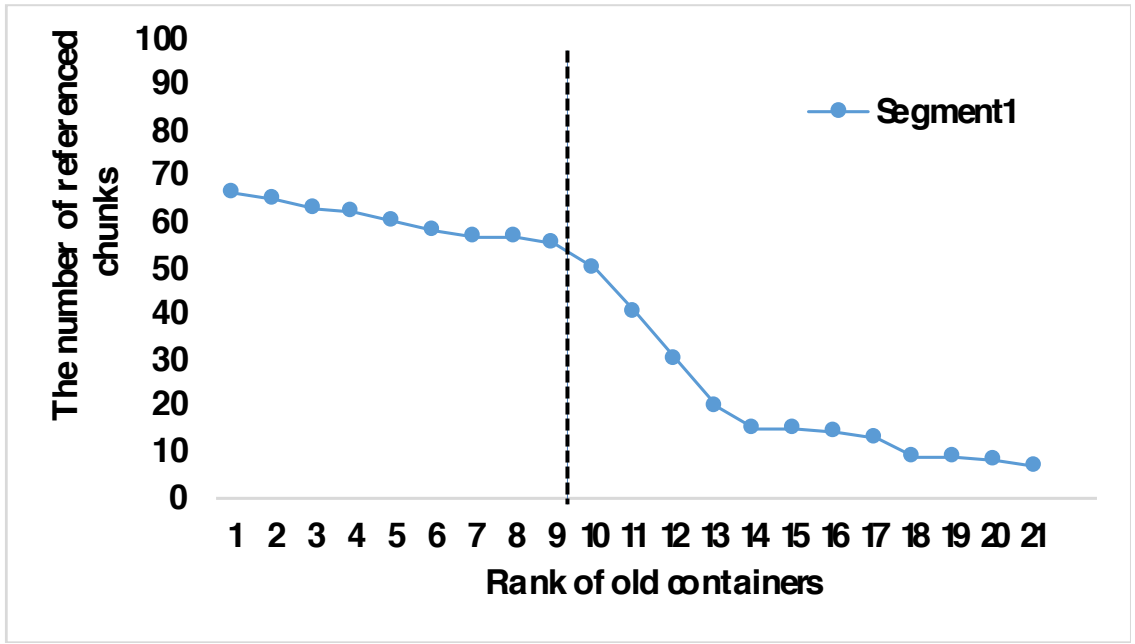
# Capping Limitation 1: Fixed Capping Level

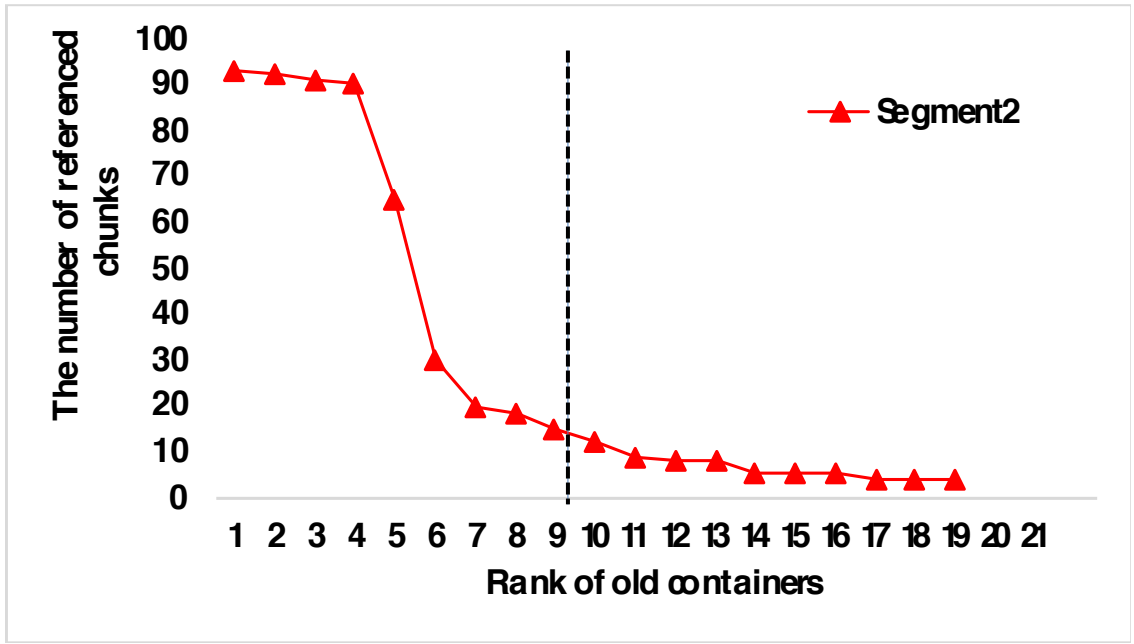


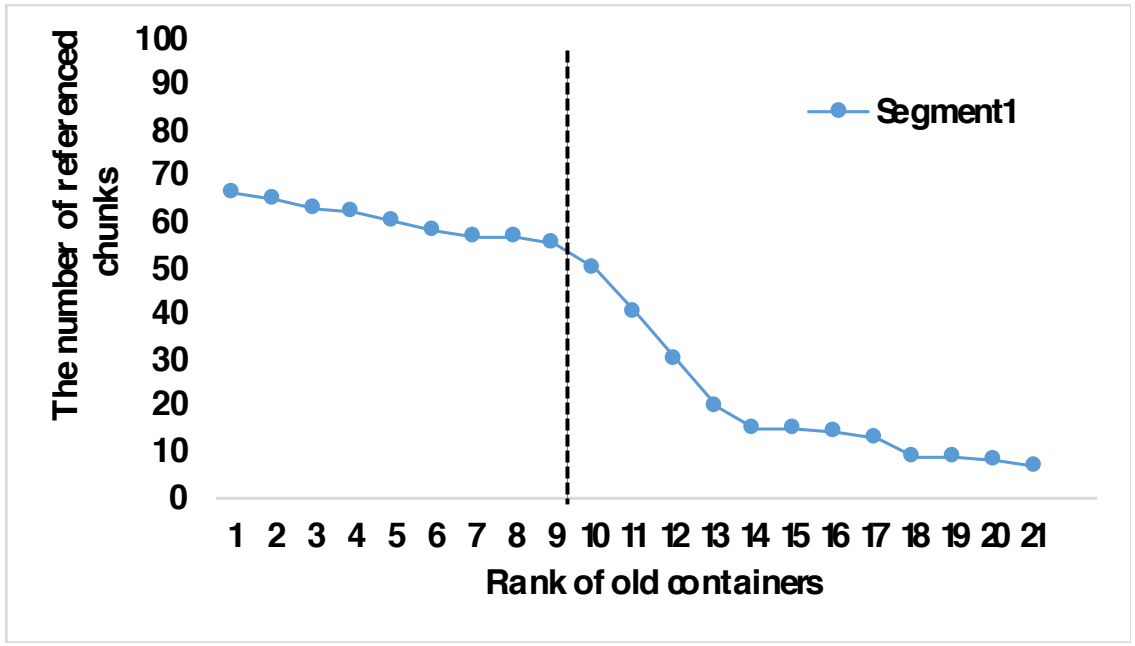

- The number of data chunks referenced by one container in a segment is called ***container referenced count (CNRC)***
- If we **sort the CNRC of containers in a segment**, we can get the distributions above, the distributions of different segment can be very different
- *Capping level* is a fixed threshold, **containers with CNRC ranked lower than the capping level are rewritten**

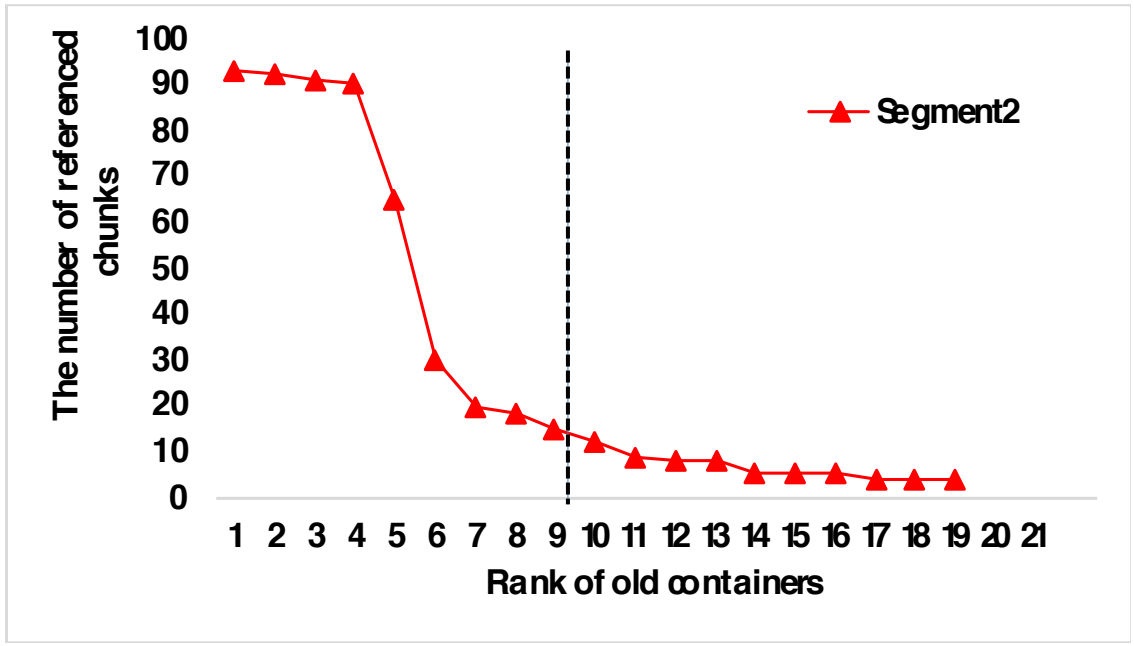
# Capping Limitation 1: Fixed Capping Level



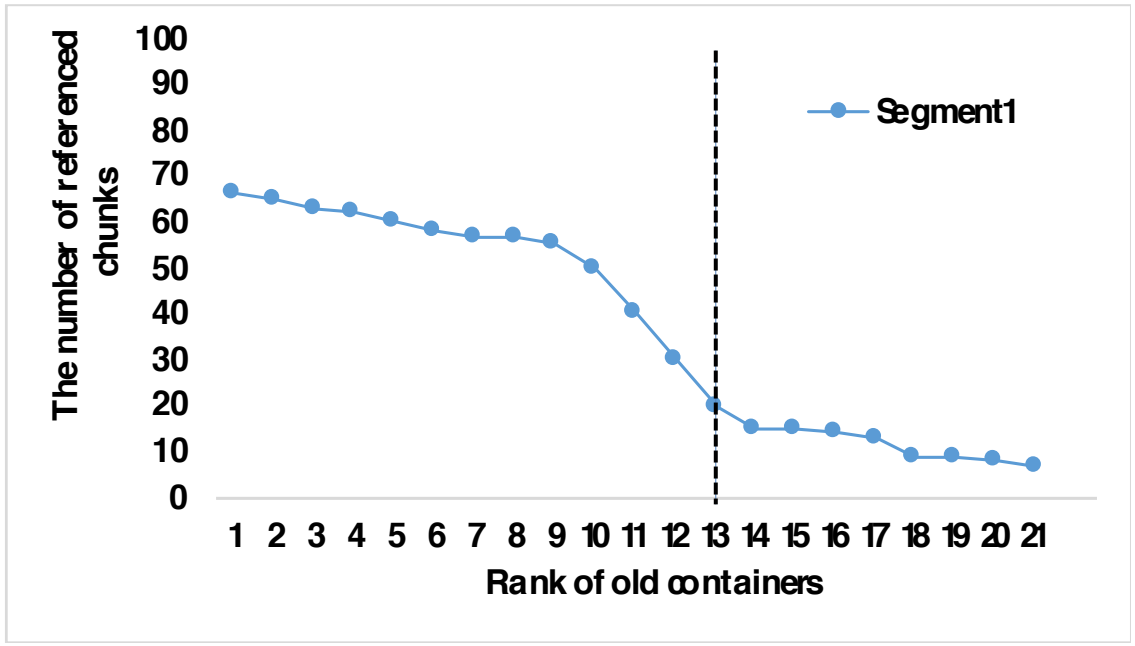
# Capping Limitation 1: Fixed Capping Level

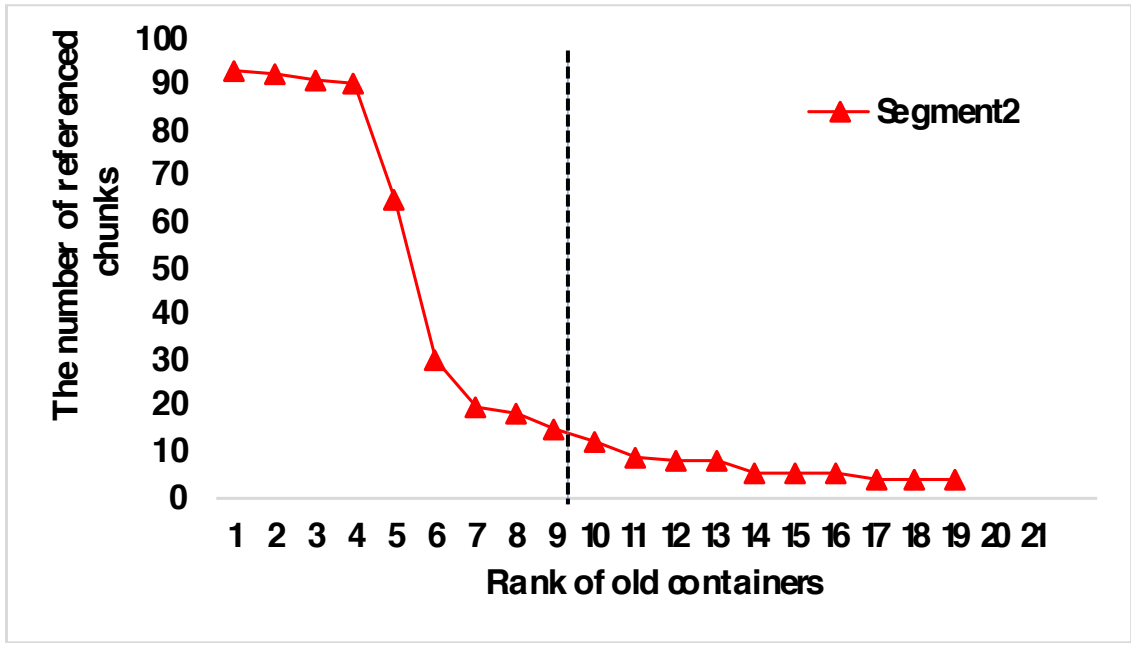



- Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks

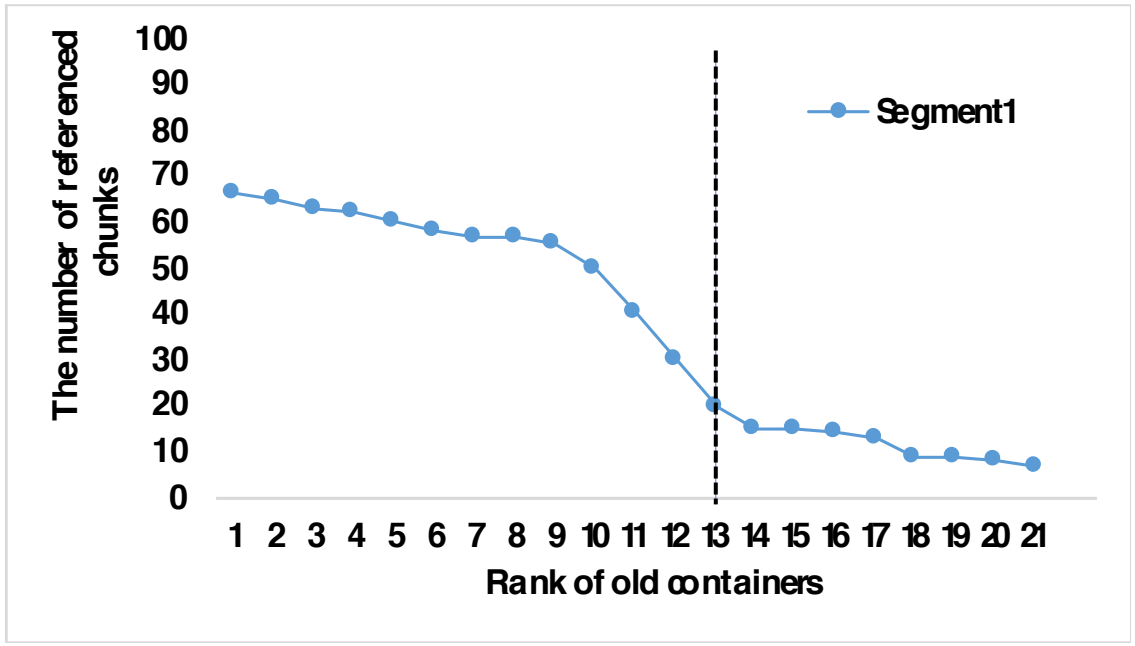
# Capping Limitation 1: Fixed Capping Level

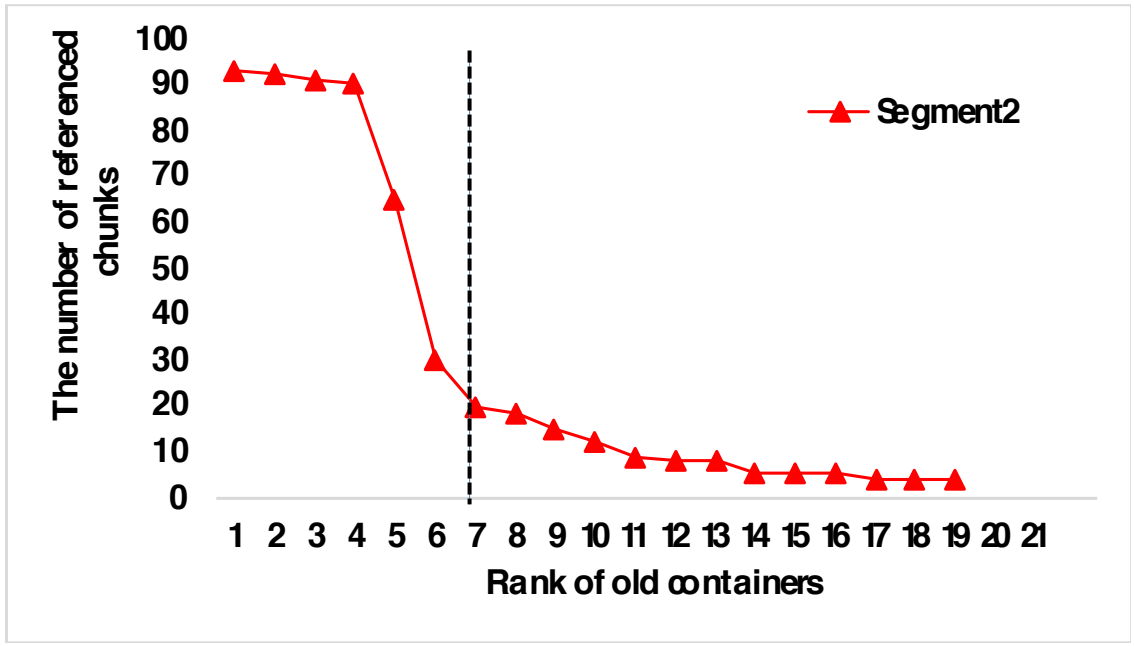



- Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks

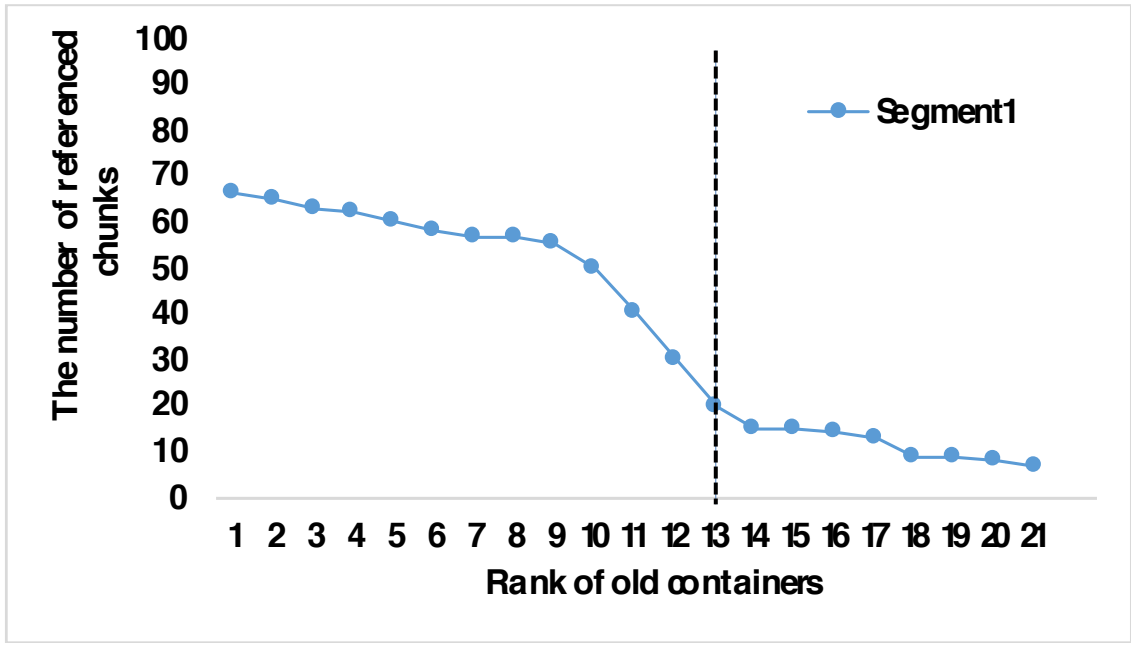
# Capping Limitation 1: Fixed Capping Level

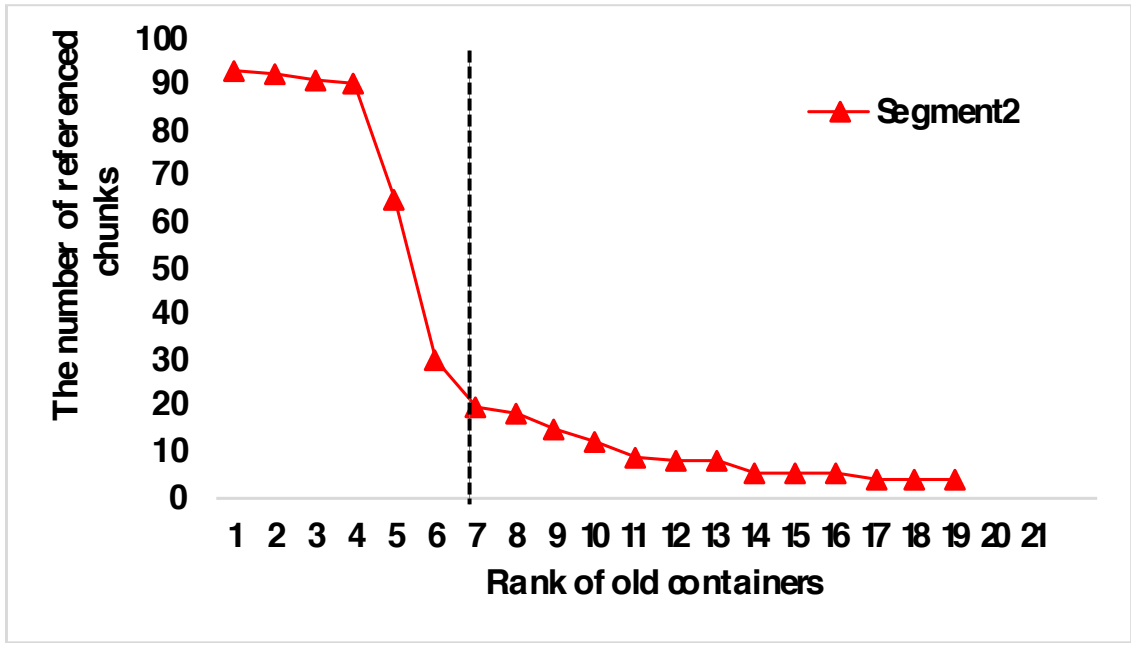



- Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks

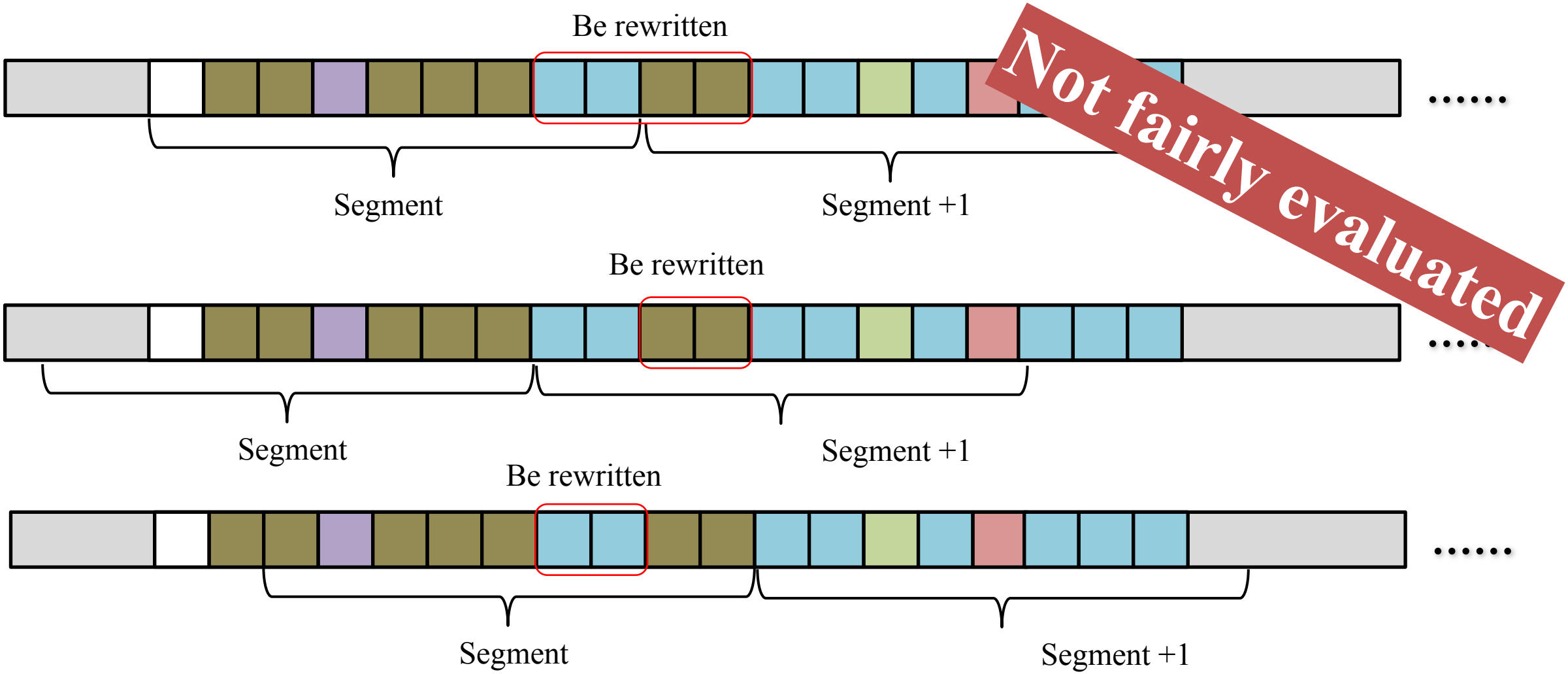
# Capping Limitation 1: Fixed Capping Level



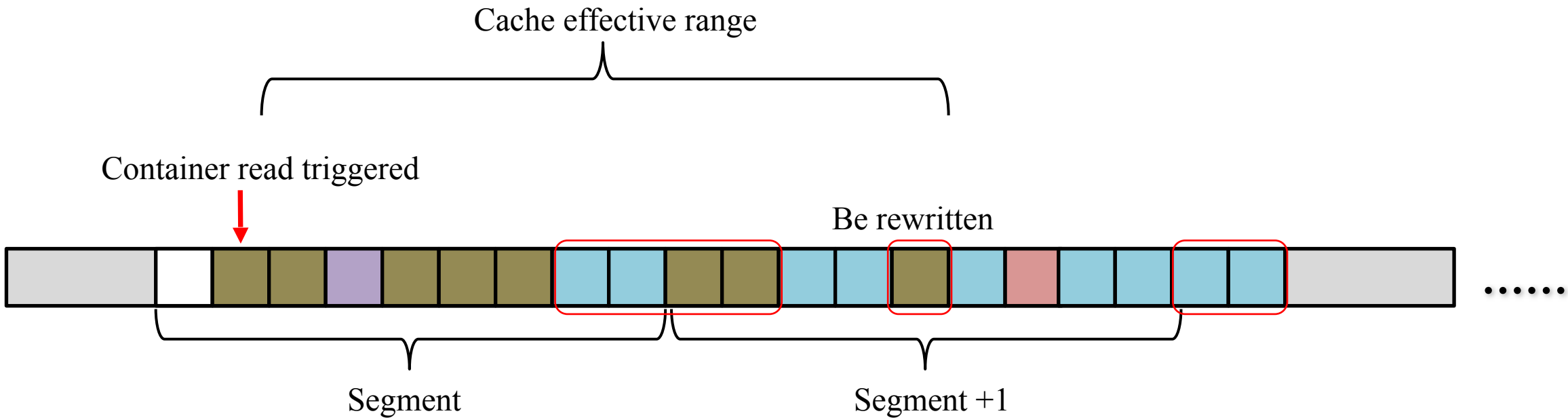

- Use the same capping level, we will have 20 old container reads, and need rewrite 232 chunks
- Use different actual capping level for different segments, the total old container reads are still 20, but we **rewrite fewer data chunks** (187 chunks)

# Capping Limitation 2: Fixed Segment Cutting Issue




Data chunks close to the cutting boundary have higher probability to be rewritten

# Capping Limitation 3: Restore Caching

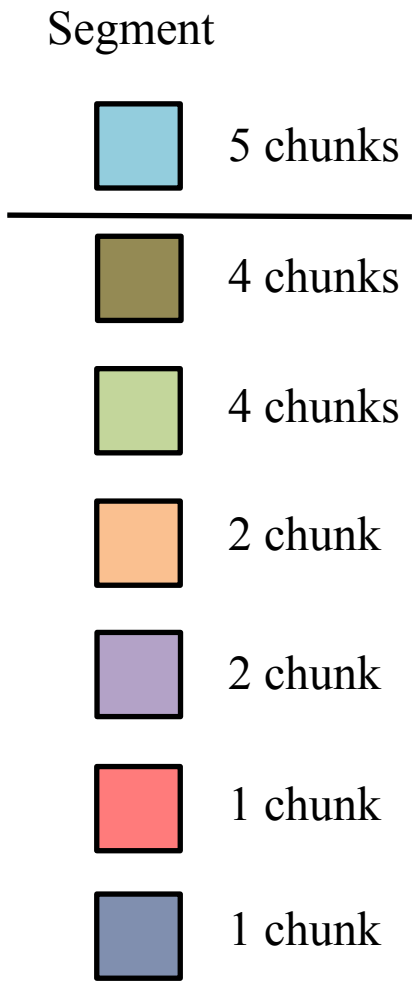



When caching is applied, data chunks from one container must be cached at least a small range of restore. The data chunks that are **covered by the cache effective range should not be rewritten**

# Agenda

- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

# Flexible Container Referenced Count based Design (FCRC)

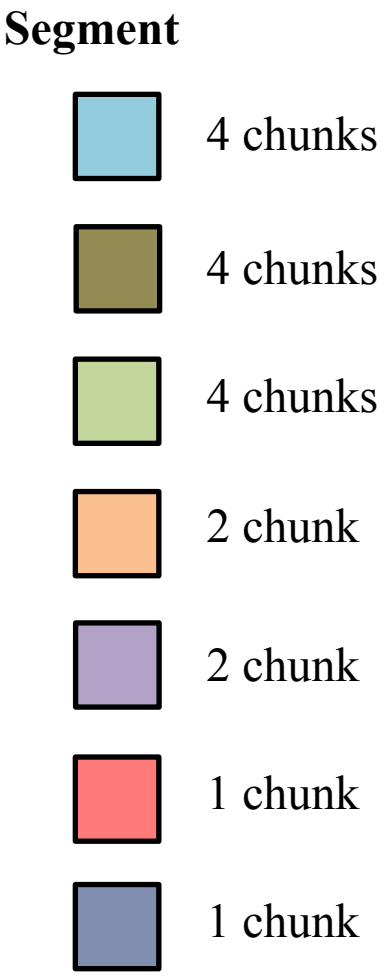



- To be adaptive to the **C**ontai**N**er **R**eferenced **C**ount (CNRC) distributions, we can use a CNRC value as the threshold
- **Higher threshold** → rewrite **more chunks** (lower deduplication ratio), but **fewer container reads** (e.g., threshold = 4)
- **Lower threshold** → rewrite **fewer chunks** (higher deduplication ratio), but **more container reads** (e.g., threshold = 1)

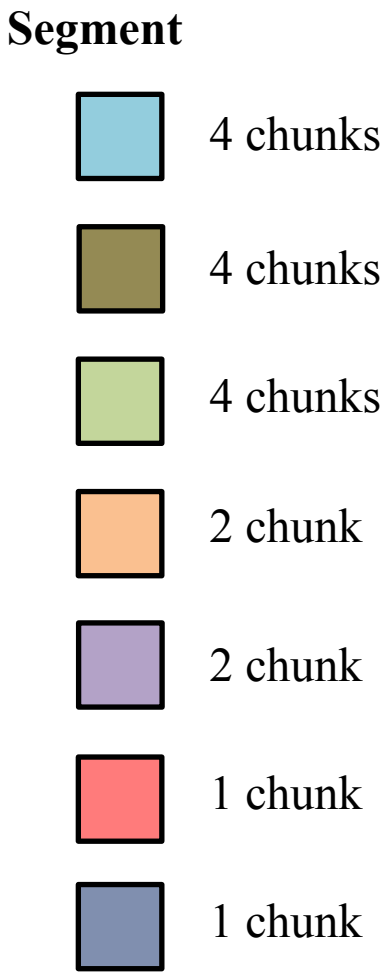
How can we decide the threshold for each segment?

According to the **target container reads** and **target deduplication ratio** to calculate (estimate) the **lower bound** and **higher bound** of the threshold in each segment.

# Set Two Bounds

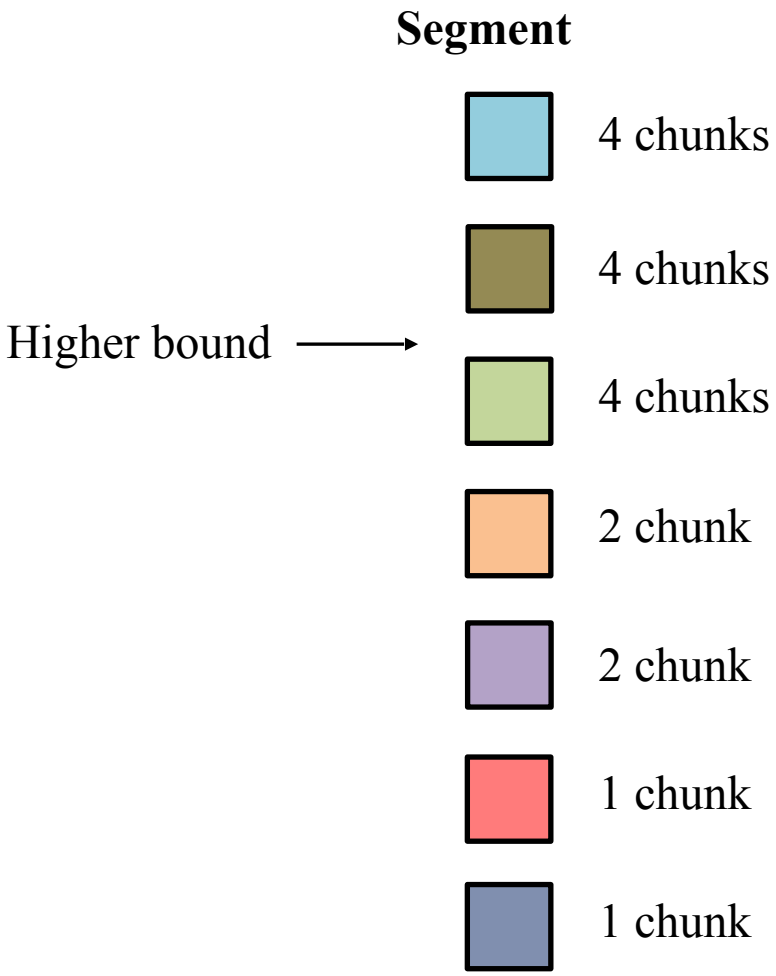


# Set Two Bounds



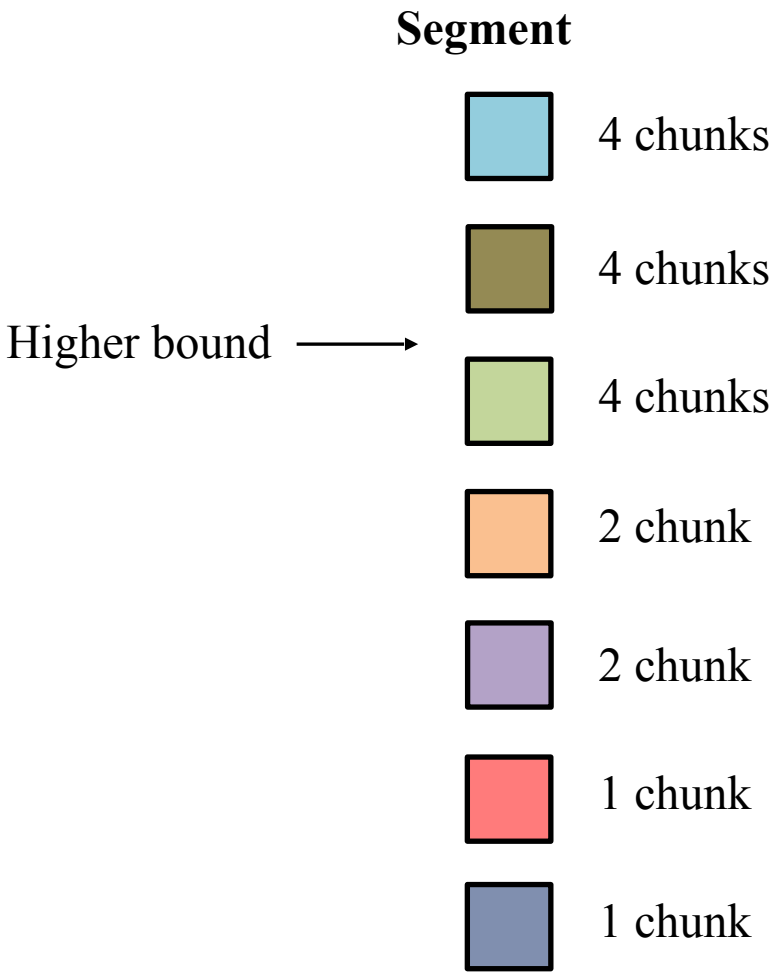

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**

# Set Two Bounds



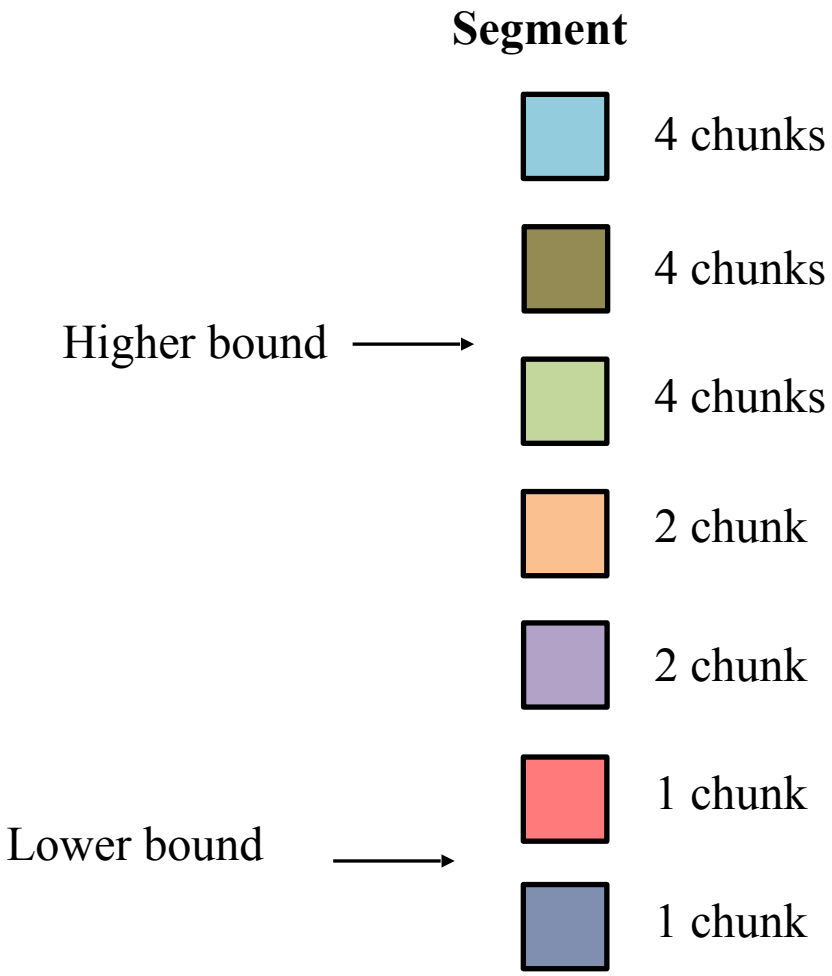

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**

# Set Two Bounds



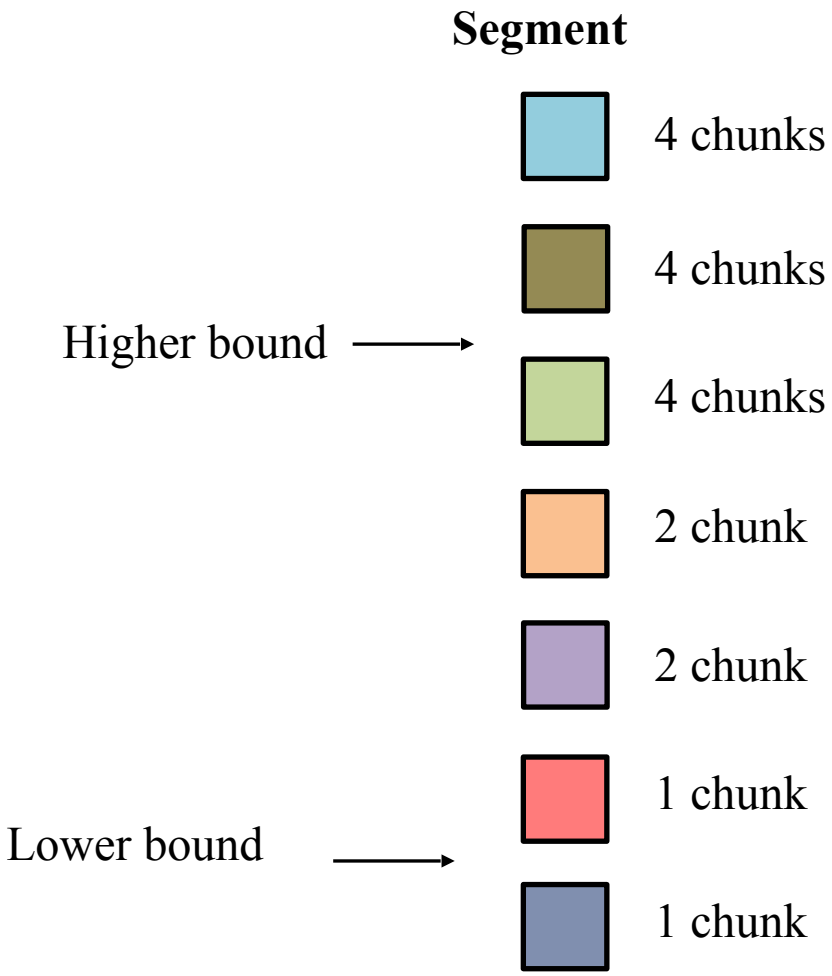

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment →**lower bound**

# Set Two Bounds



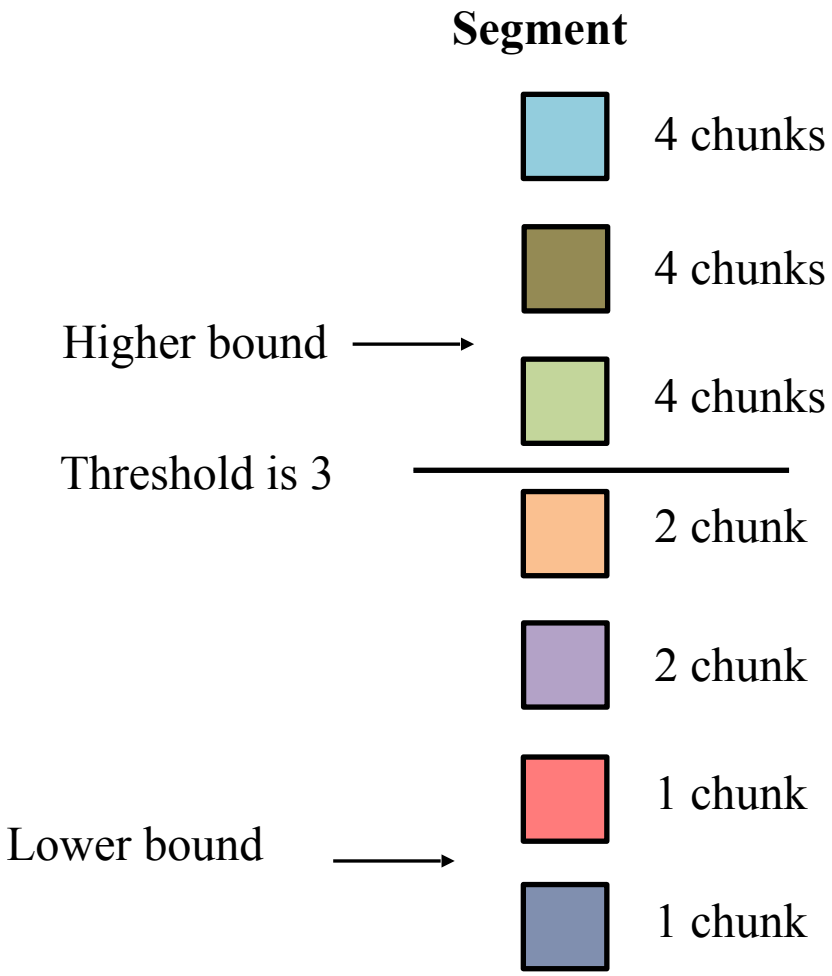

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment →**lower bound**

# Set Two Bounds



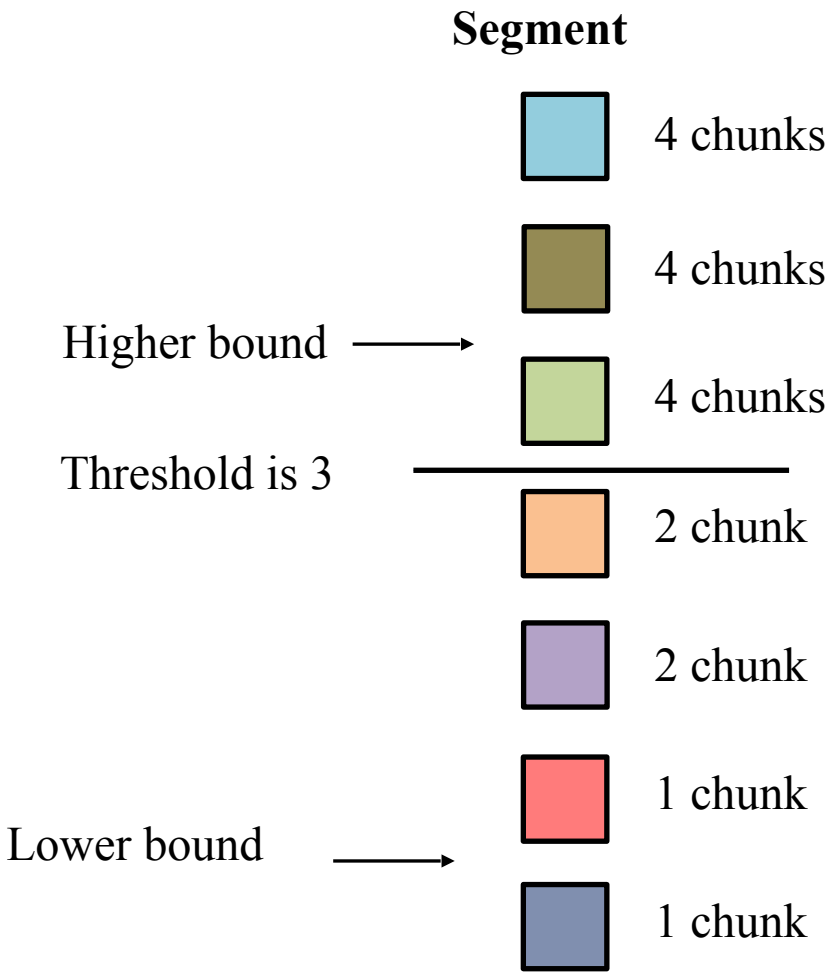

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment →**lower bound**
- The **actual CNRC threshold is in between** (if cannot be set, satisfy deduplication ratio first)

# Set Two Bounds



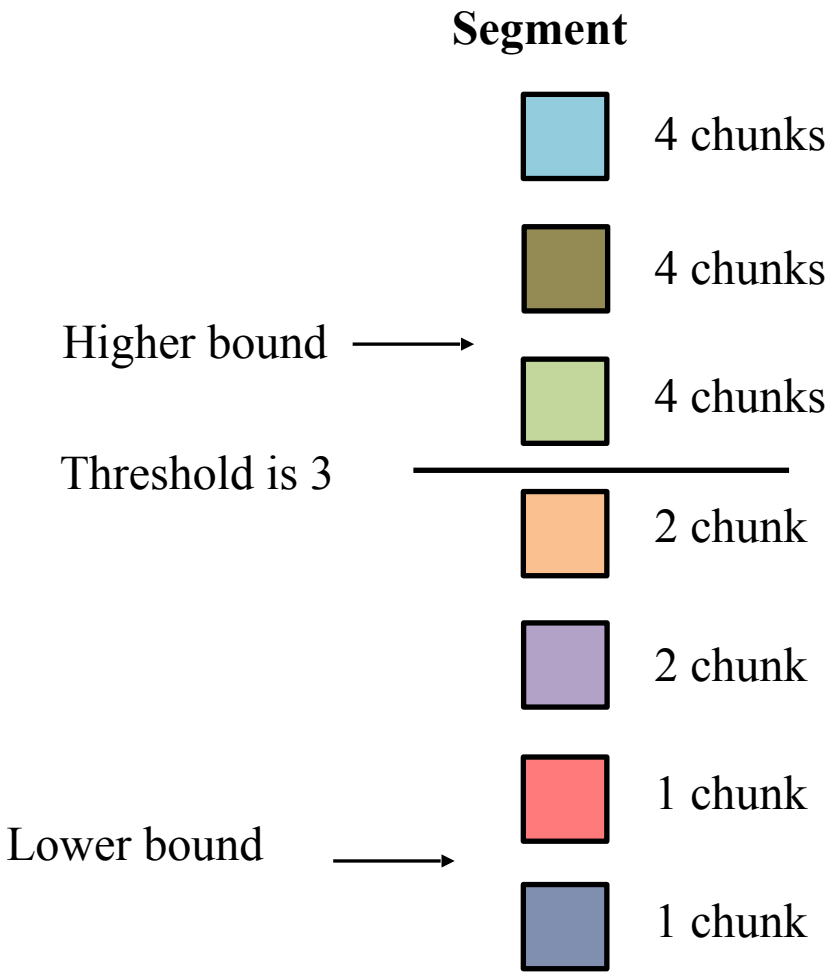

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment →**lower bound**
- The **actual CNRC threshold is in between** (if cannot be set, satisfy deduplication ratio first)

# Set Two Bounds




- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment →**lower bound**
- The **actual CNRC threshold is in between** (if cannot be set, satisfy deduplication ratio first)
- If we rewrite fewer chunks or referenced fewer containers in this segment, we can accumulate the **"credits" of container reads or rewrites savings** for the future segments to extend the two bounds

# Set Two Bounds



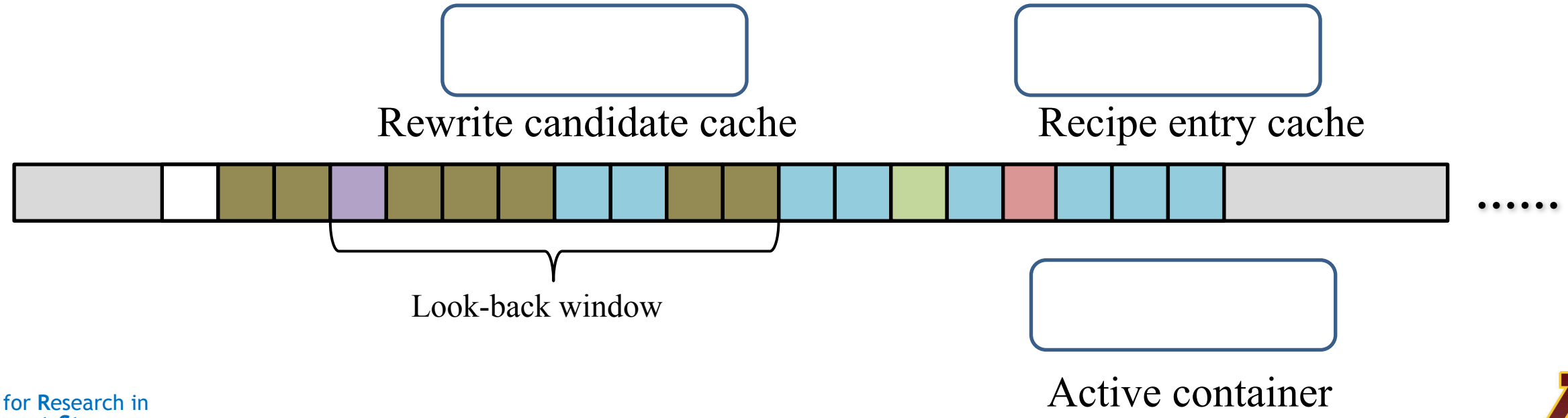

- According to the **deduplication ratio reduction limit**, we can estimate, at most, how many data chunks can be rewritten in this segment → **higher bound**
- According to the **target container reads**, we can estimate, at most, how many old containers can be referenced in this segment →**lower bound**
- The **actual CNRC threshold is in between** (if cannot be set, satisfy deduplication ratio first)
- If we rewrite fewer chunks or referenced fewer containers in this segment, we can accumulate the **"credits" of container reads or rewrites savings** for the future segments to extend the two bounds
- e.g., for next segment, we can move the lower bound **3 containers "down"** and higher bound **4 chunks "up"**.

# Agenda

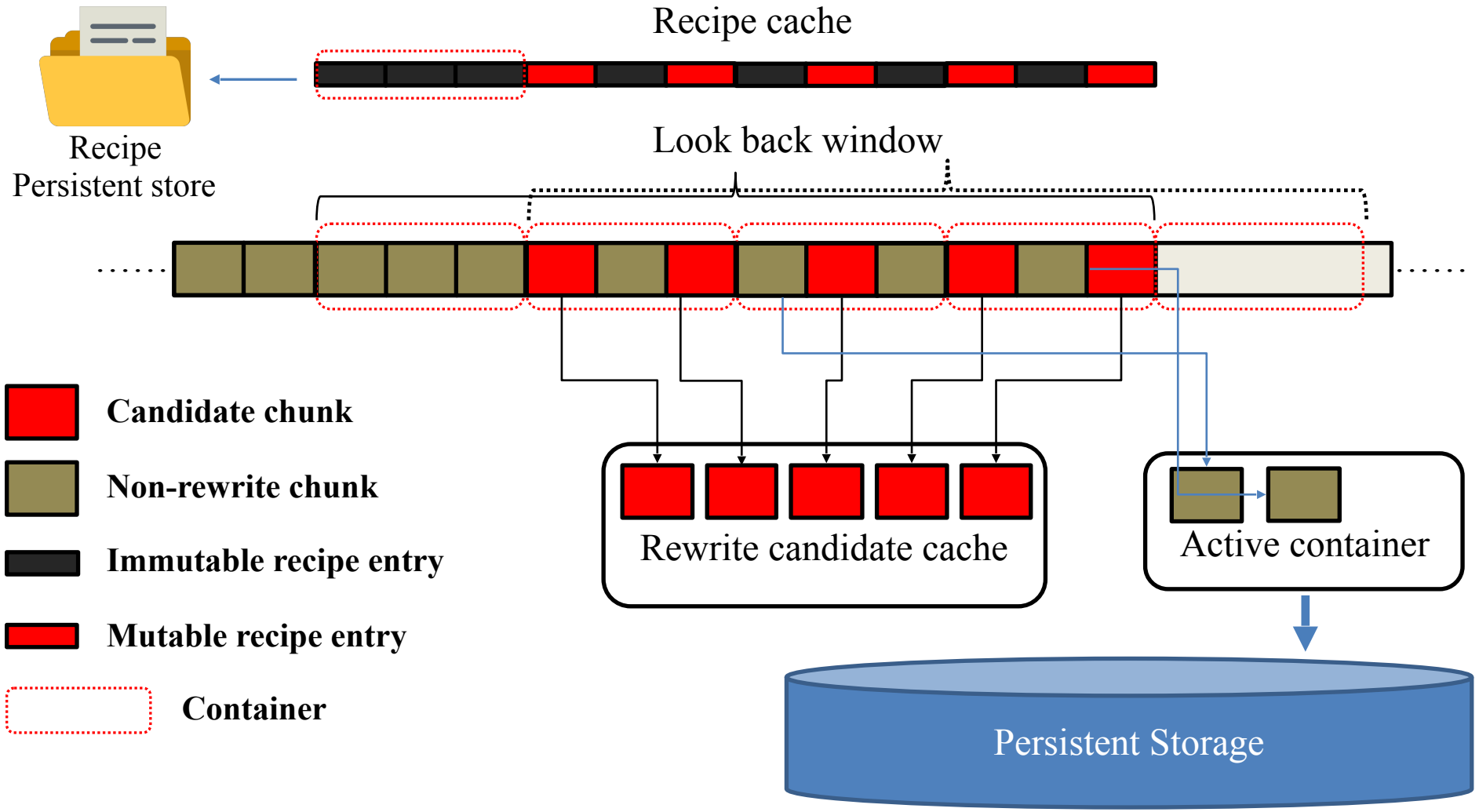
- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

# Sliding Look-back Window Assisted Rewrite

- Basic idea: using a sliding window to cover a range of data chunks, ensure that each chunk is evaluated with the **same amount of past and future information**
- To be efficient, the window is moved in container size (4 chunks in this example)
- Data chunks, whose rewrite decision cannot be made, will be temporally cached until the window moves to cover its subsequence. In this way, we can finally make the rewrite decision



# Architecture



# Considering Restore Caching During Rewrite

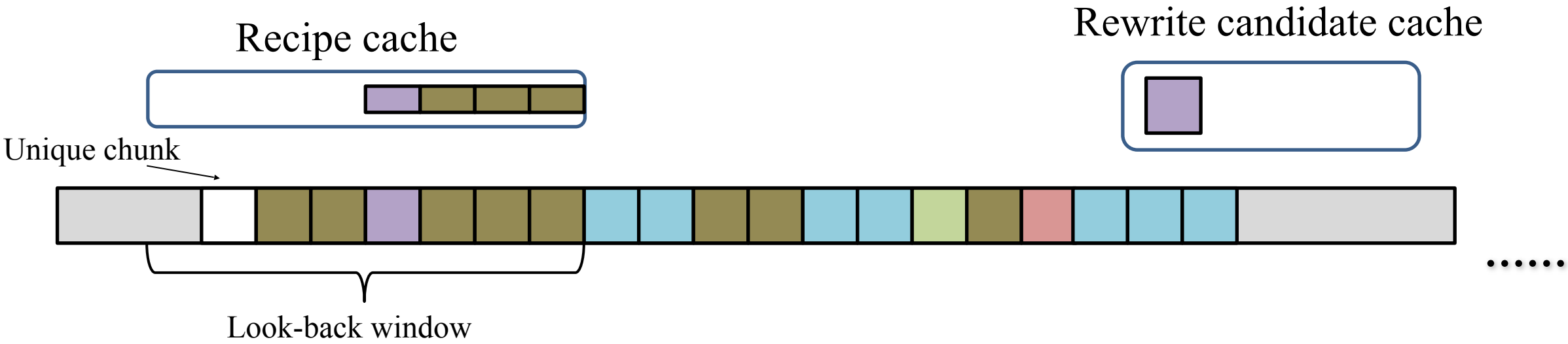
- According to the caching algorithm and cache space size, we can estimate the cache effective range (all data chunks from read-in container are cached at least # data chunks restore)

**For example:**

- **FAA**: the cache effective range is the FAA size (guaranteed)
- **Chunk-based cache**: if the cache space is *S* chunks, *x%* data chunks in a container are used in average, container size is *C* chunks, the cache effective range is: (not guaranteed, just the estimate in average)

# Example

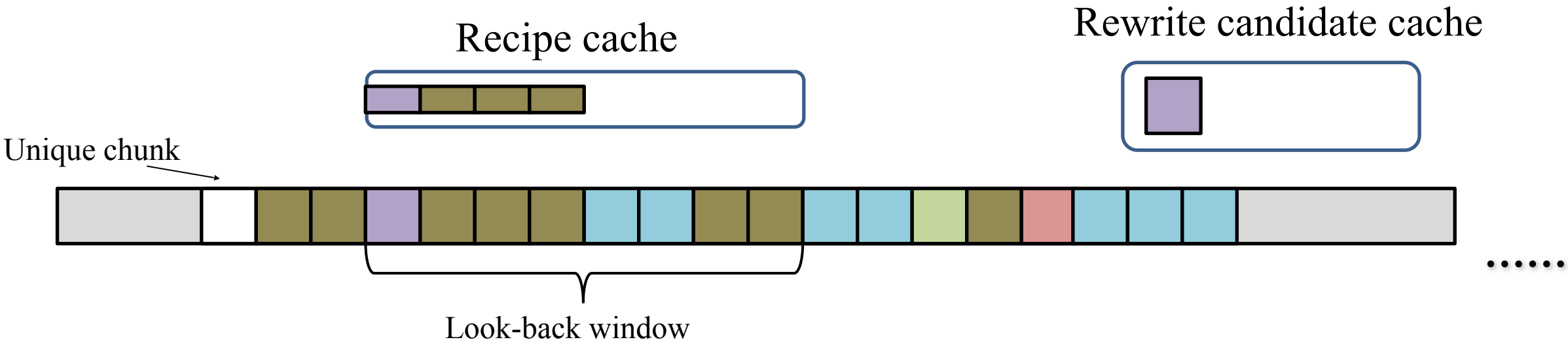
- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2




Active container

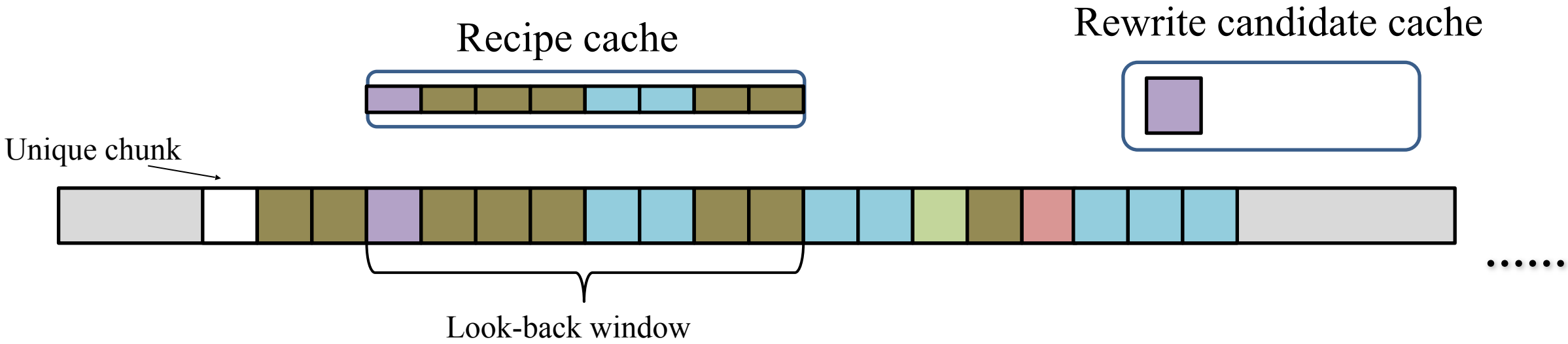
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



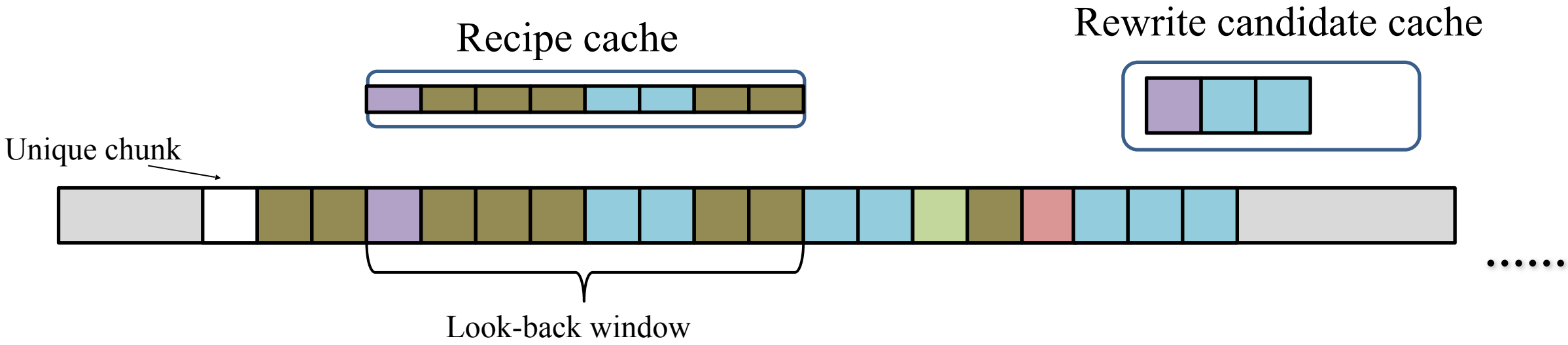
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



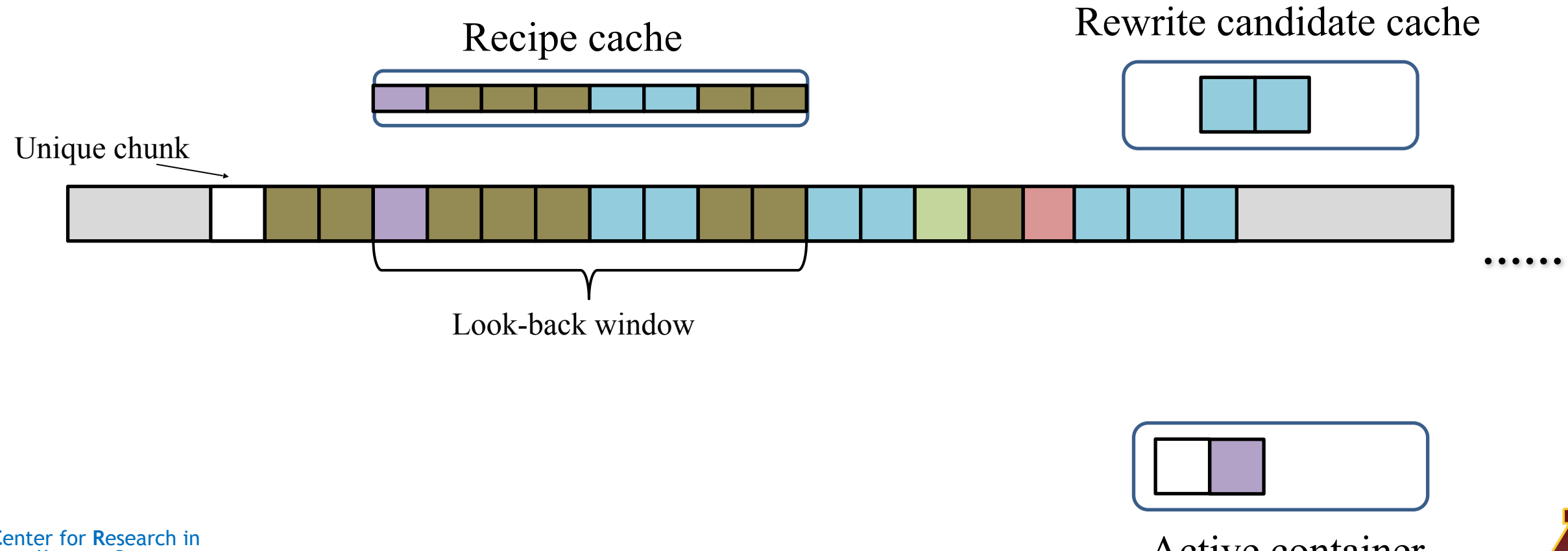
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



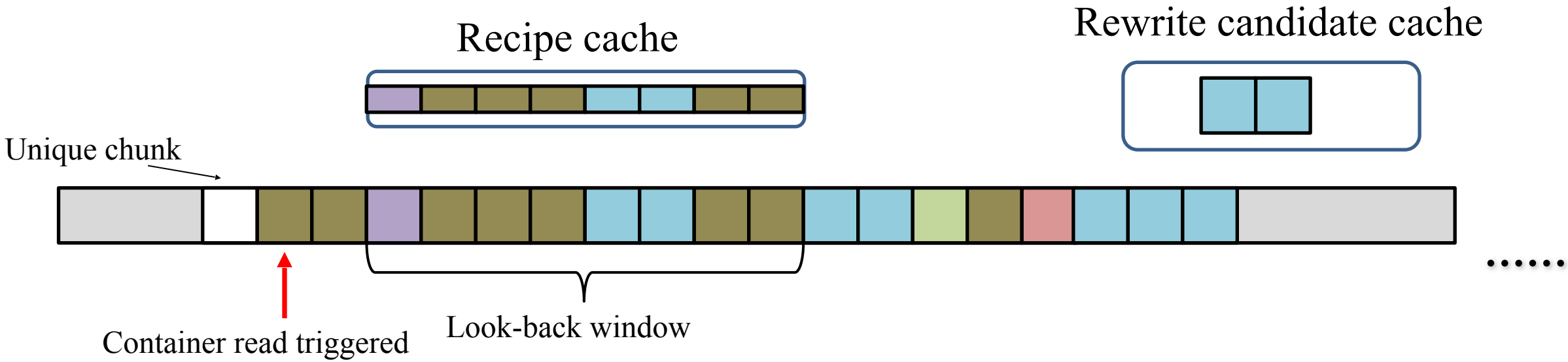
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



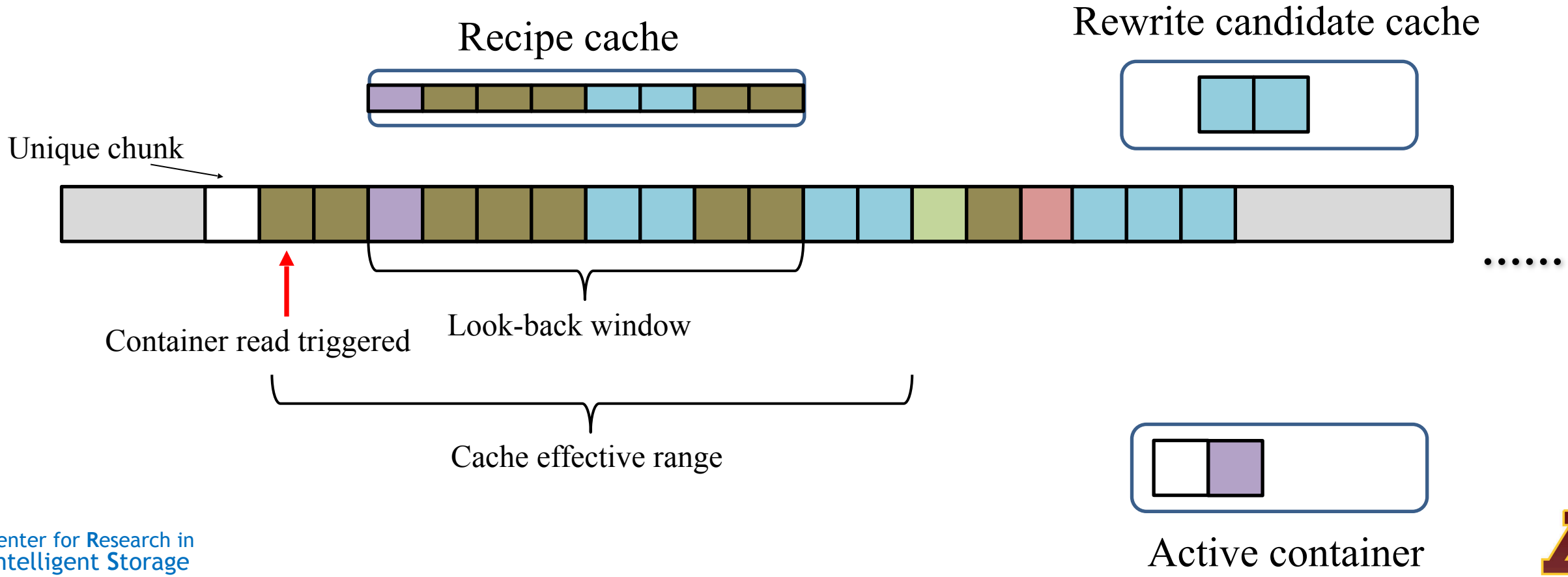
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



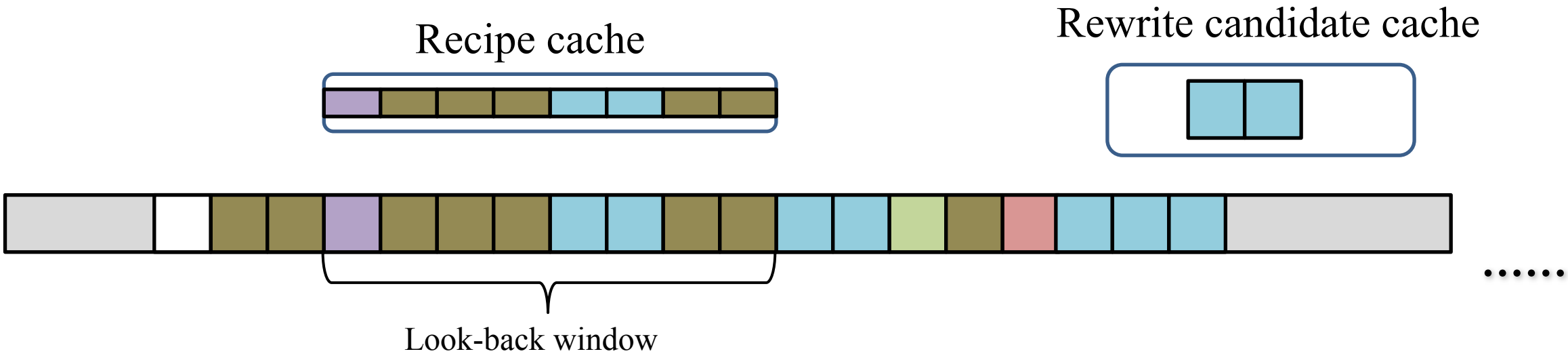
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



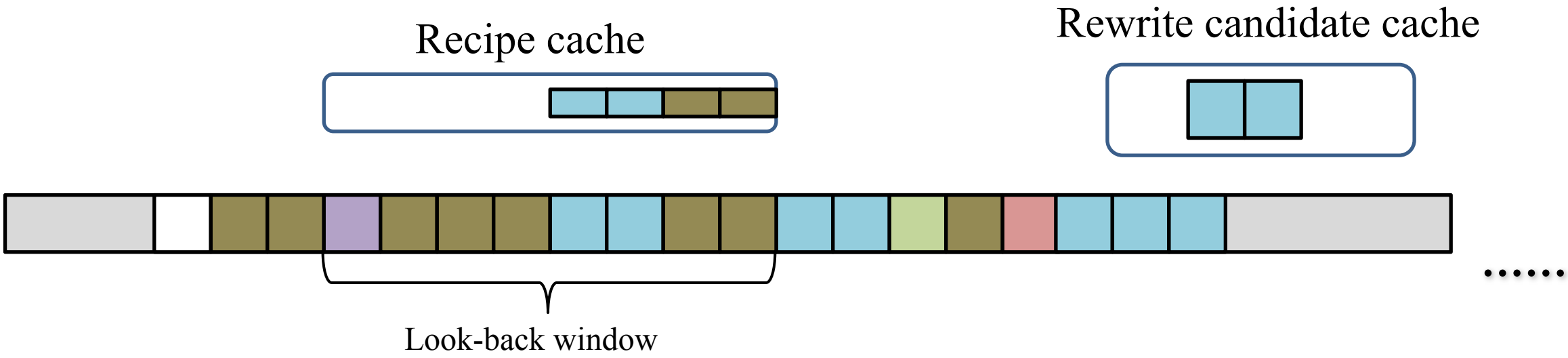
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



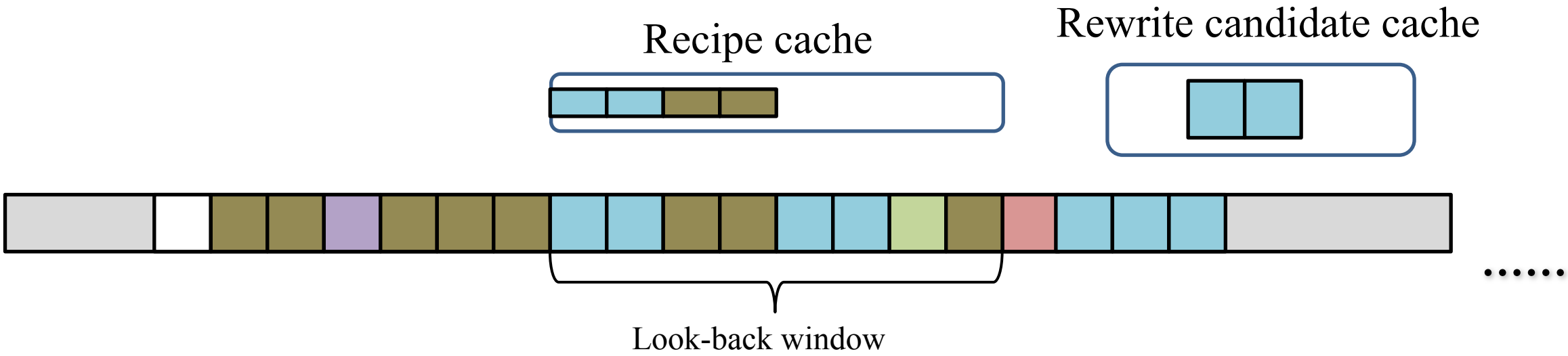
# Example

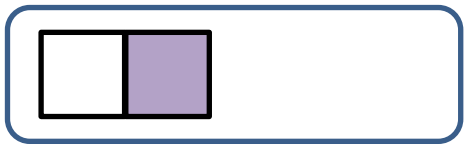
- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



# Example

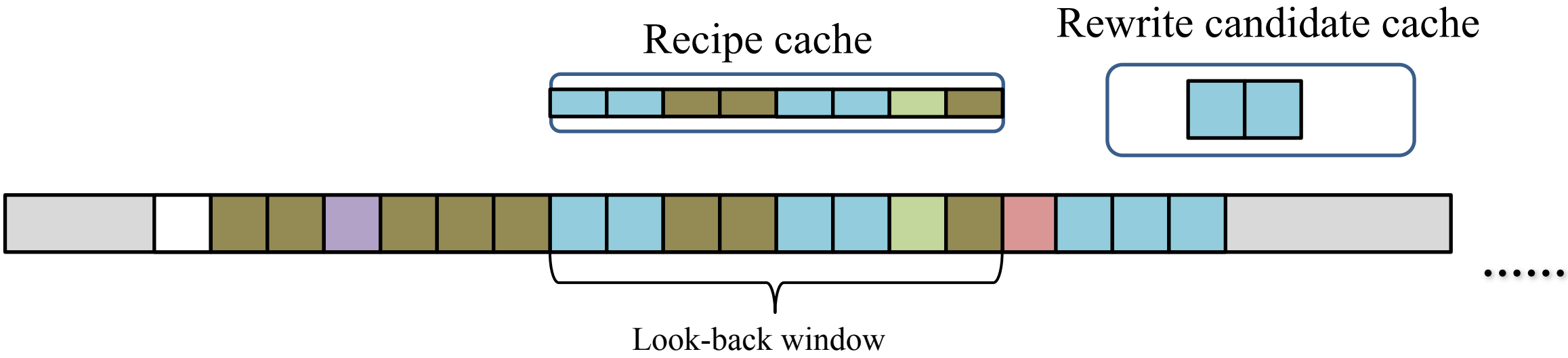
- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



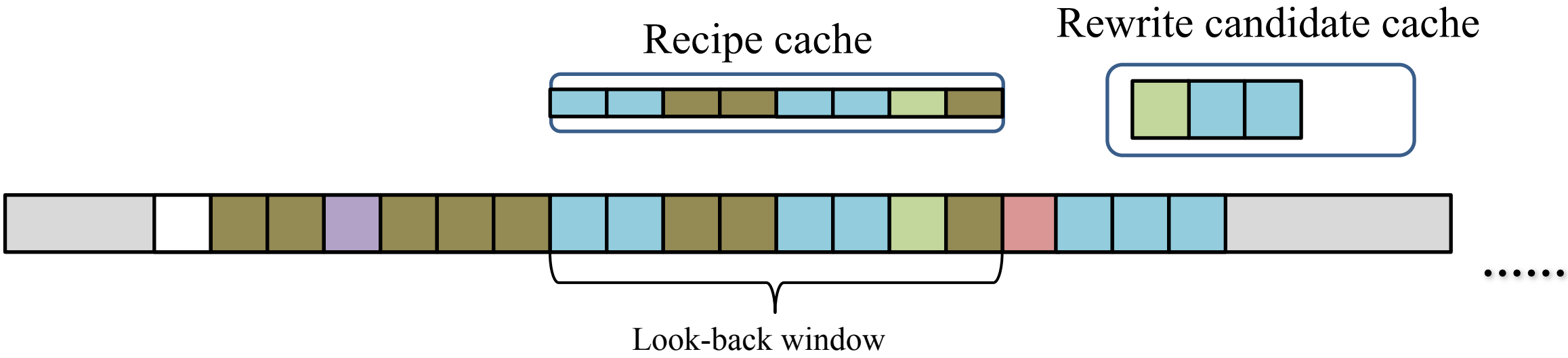
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



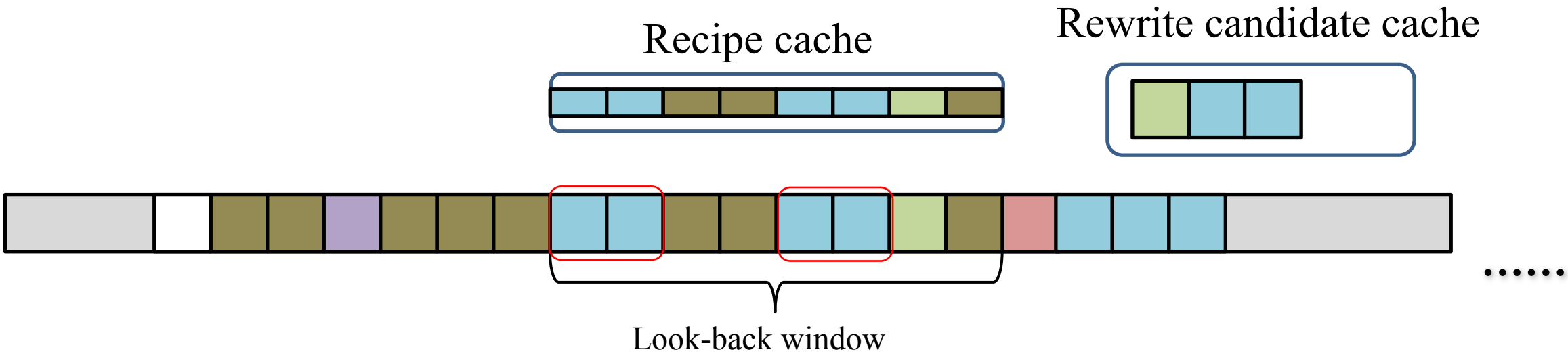
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



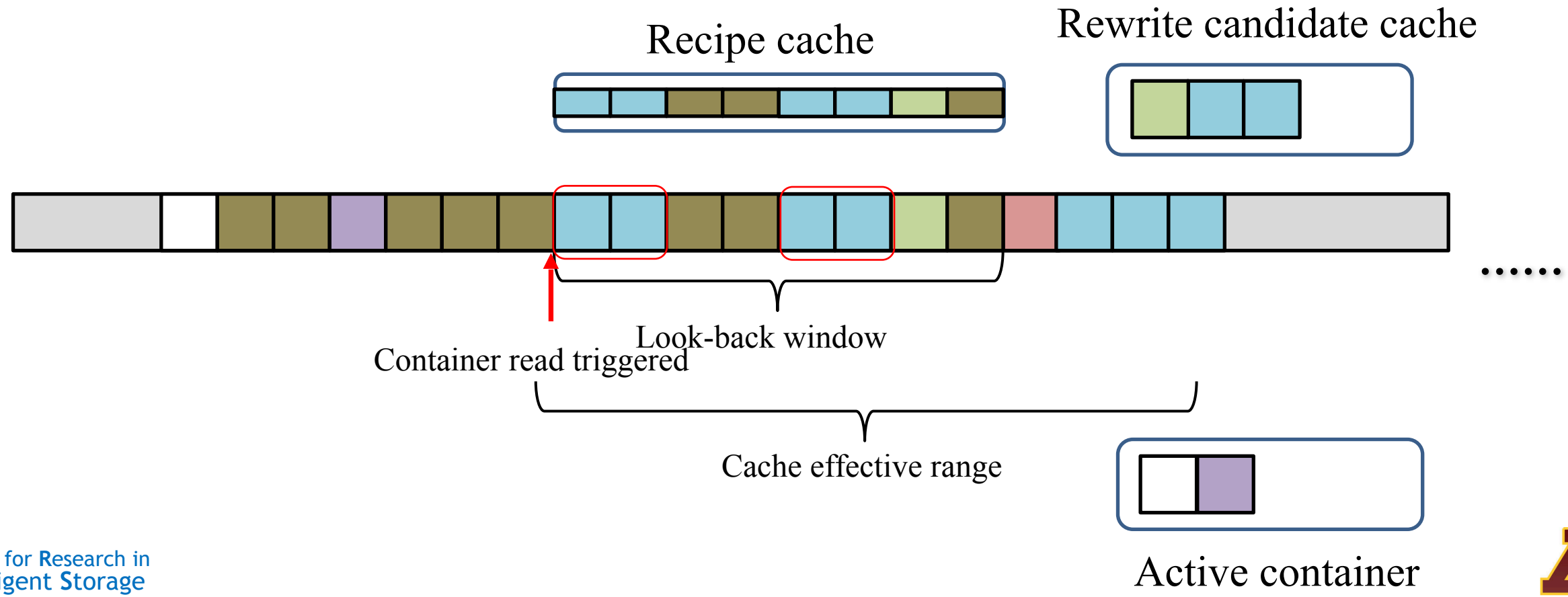
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



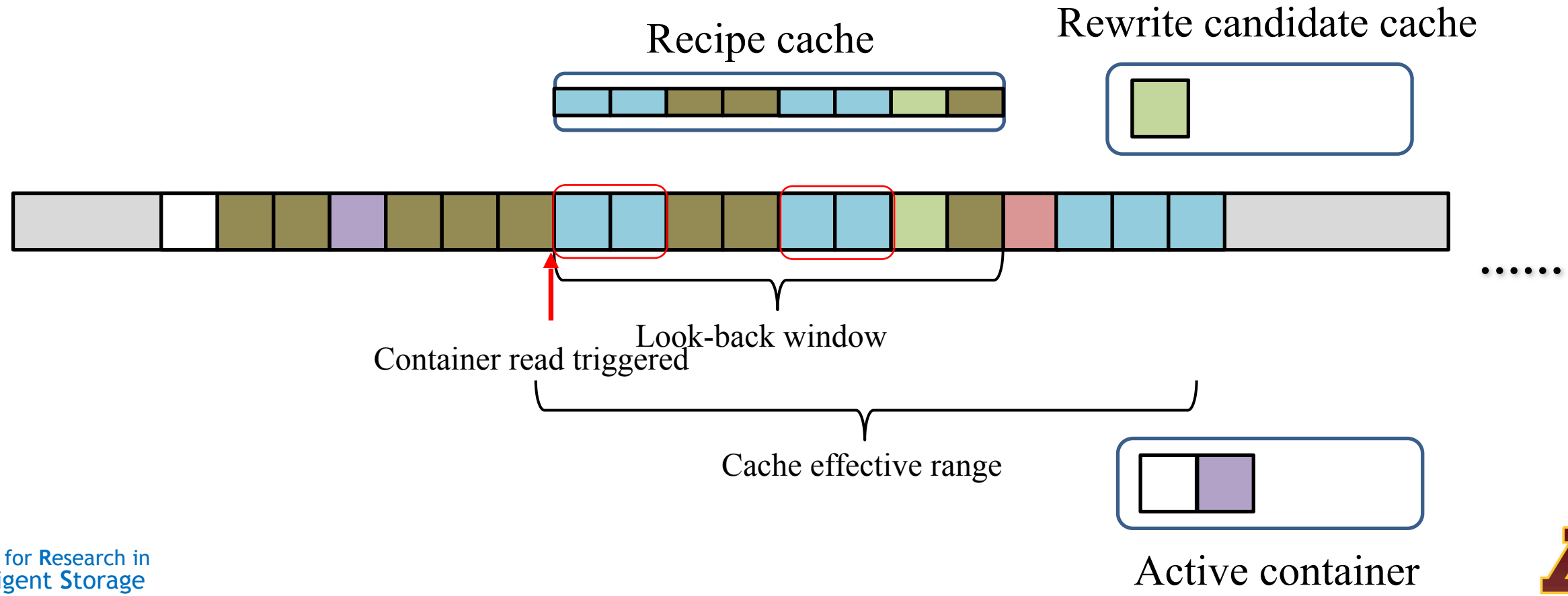
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



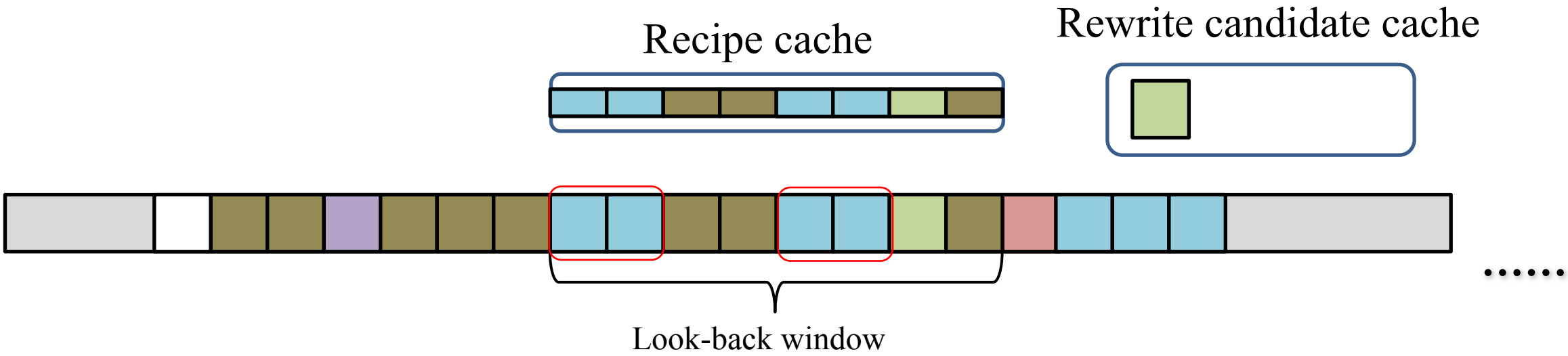
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



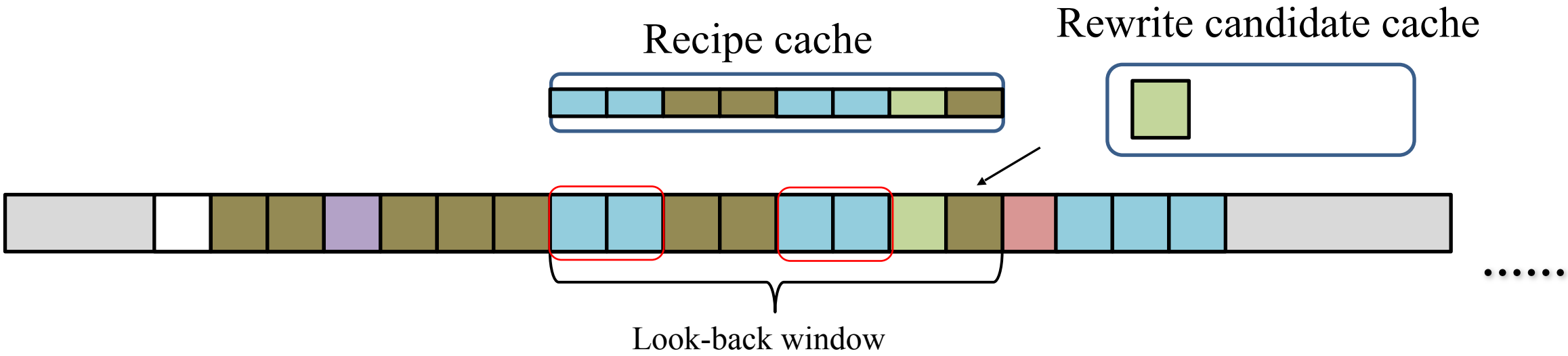
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



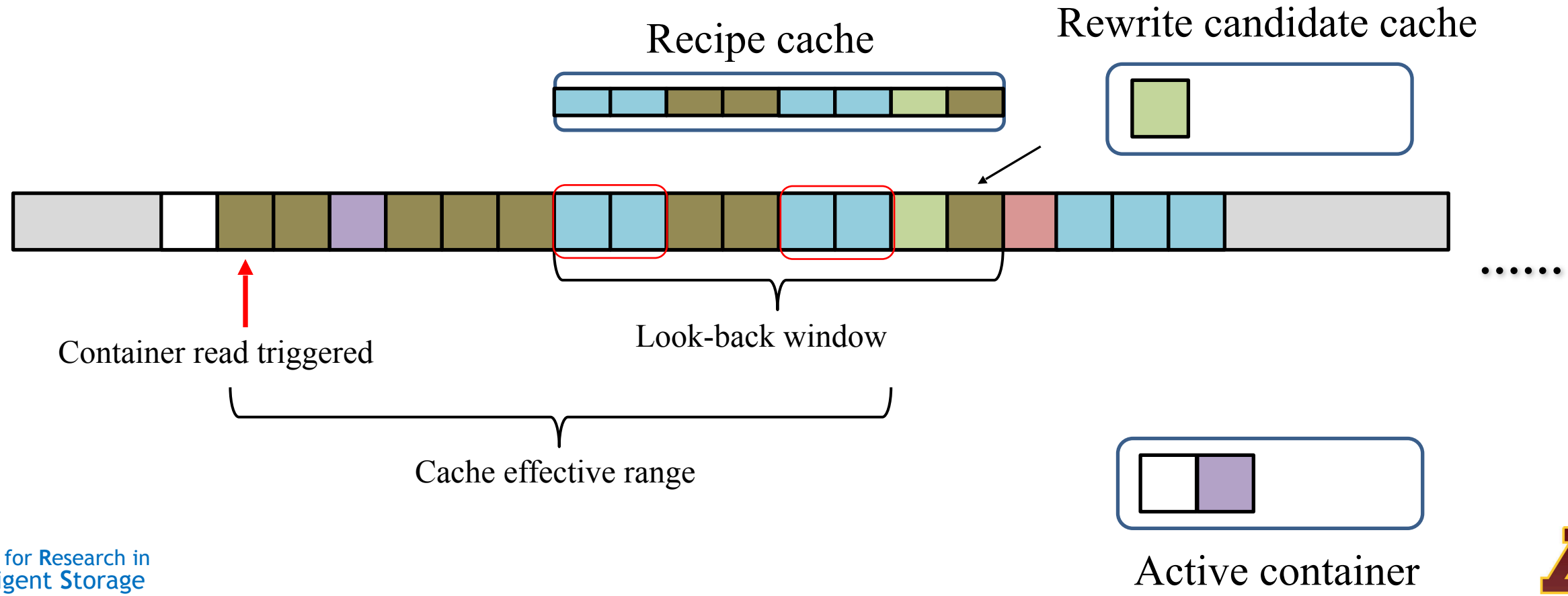
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



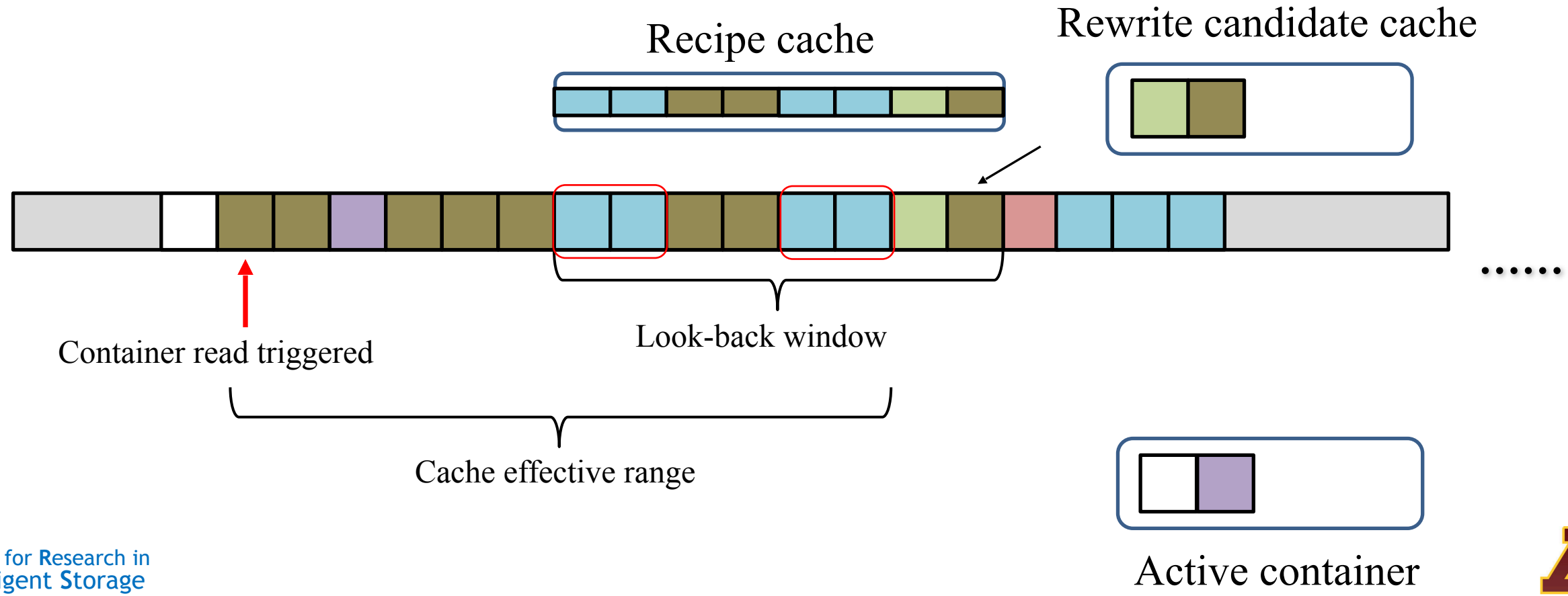
# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



# Example

- LBW size: 2 containers (8 chunks)
- Cache effective range: 3 containers (12 chunks)
- Rewrite condition:
  - The container has not been referenced for the LBW size chunks
  - The container reference count of aforementioned container is always <= 2



# Rewrite Policy

- **Combine FCRC** algorithm with the **sliding look-back window** design to decide the CNRC threshold for the containers that to be rewritten
- Make the **rewrite or non-rewrite decision as early as possible** in each LBW movement to reduce the caching overhead
- Considering the **cache effect of restore engine** to configure the LBW size and cache sizes
- Considering the **workload data locality** to slightly adjust the threshold after it is decided, such that the restore engine cache space utilization can be potentially improved.

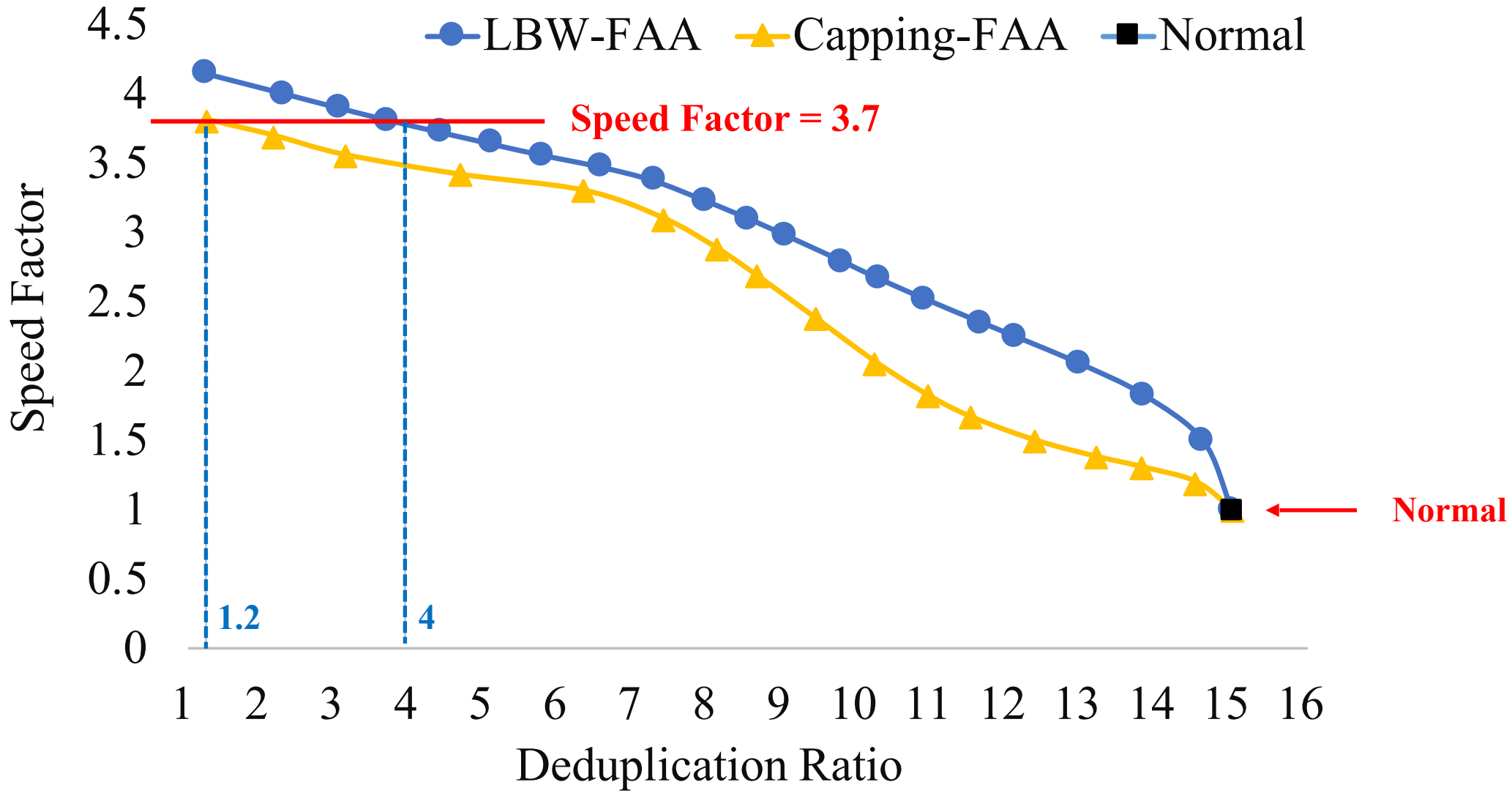
# Agenda

- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

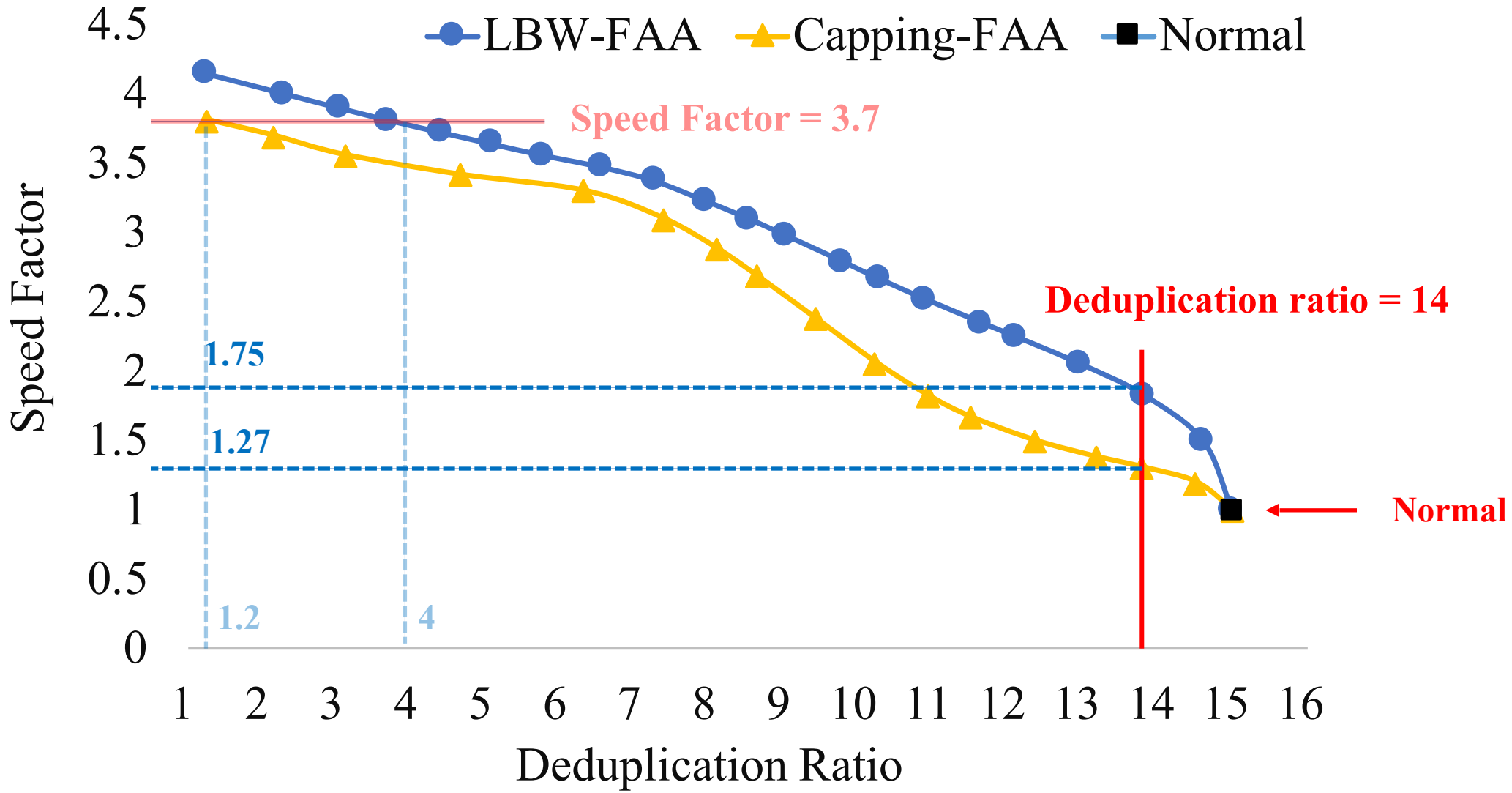
# Experiment Setup

- Evaluation metrics
  - **Speed factor:** mean size data being restored (MB) per container read
  - **Deduplication ratio:** total amount of data in the original byte stream divided by the total size of stored unique data chunks
- **Eight deduplication-restore combinations**:
  - Normal – FAA,  Normal – ALACC
  - Capping – FAA,  Capping – ALACC
  - FCRC – FAA,  FCRC – ALACC
  - LBW – FAA,  LBW – ALACC
- **Six Traces** with 10 backup versions each from FSL[]:
  - 3 MacOS server snapshots traces (5, 20, and 60 days backup intervals)
  - 3 students' home directory snapshots traces (5, 20, and 60 days backup intervals)

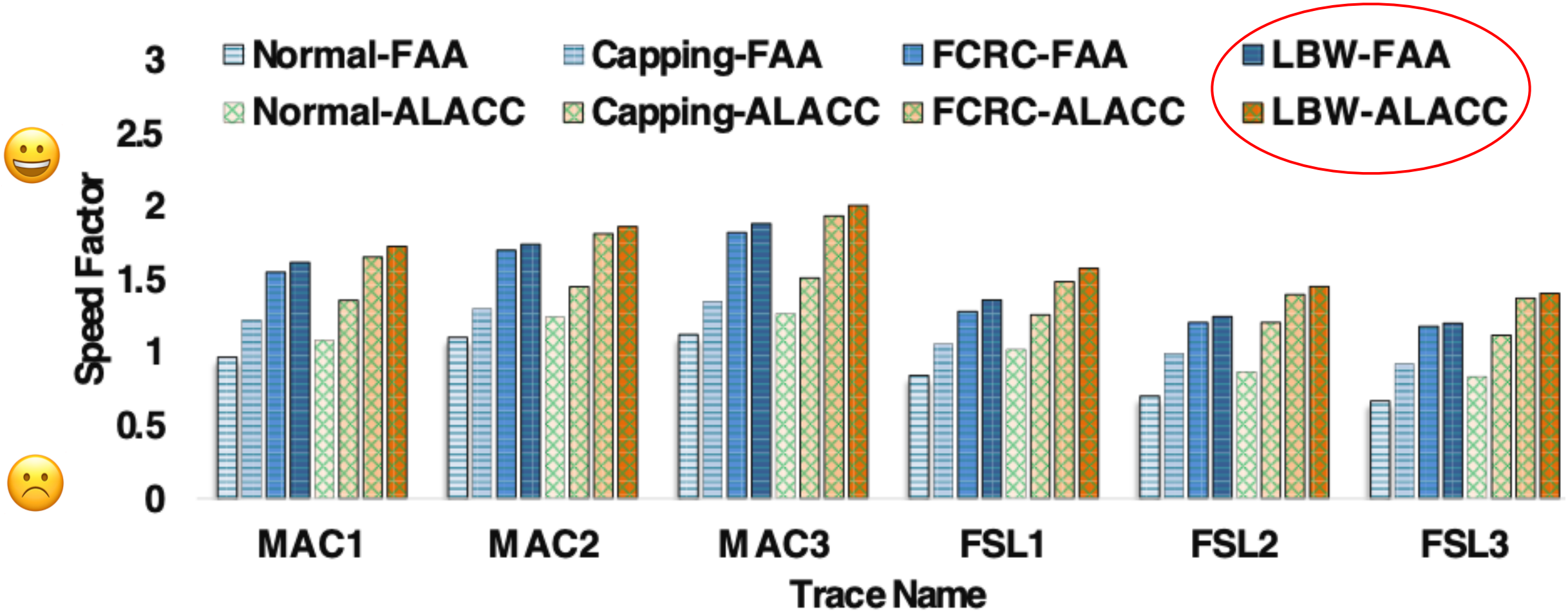
# Capping vs. LBW



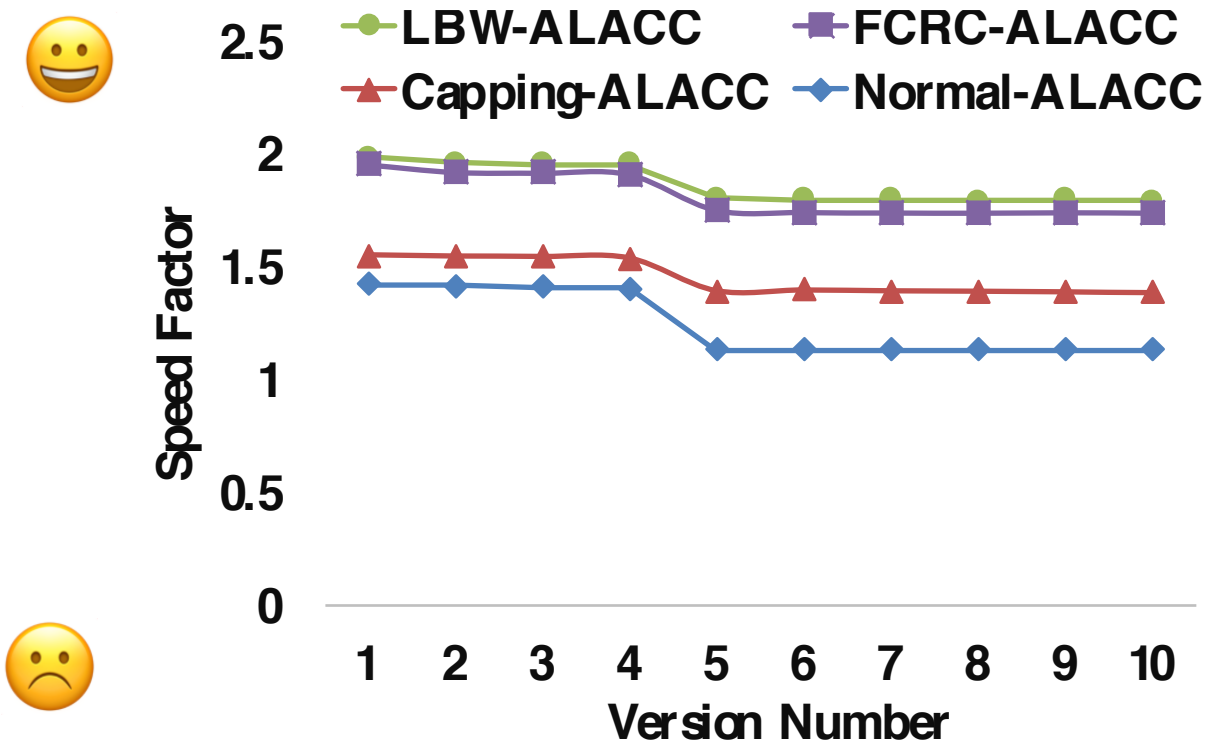
# Capping vs. LBW

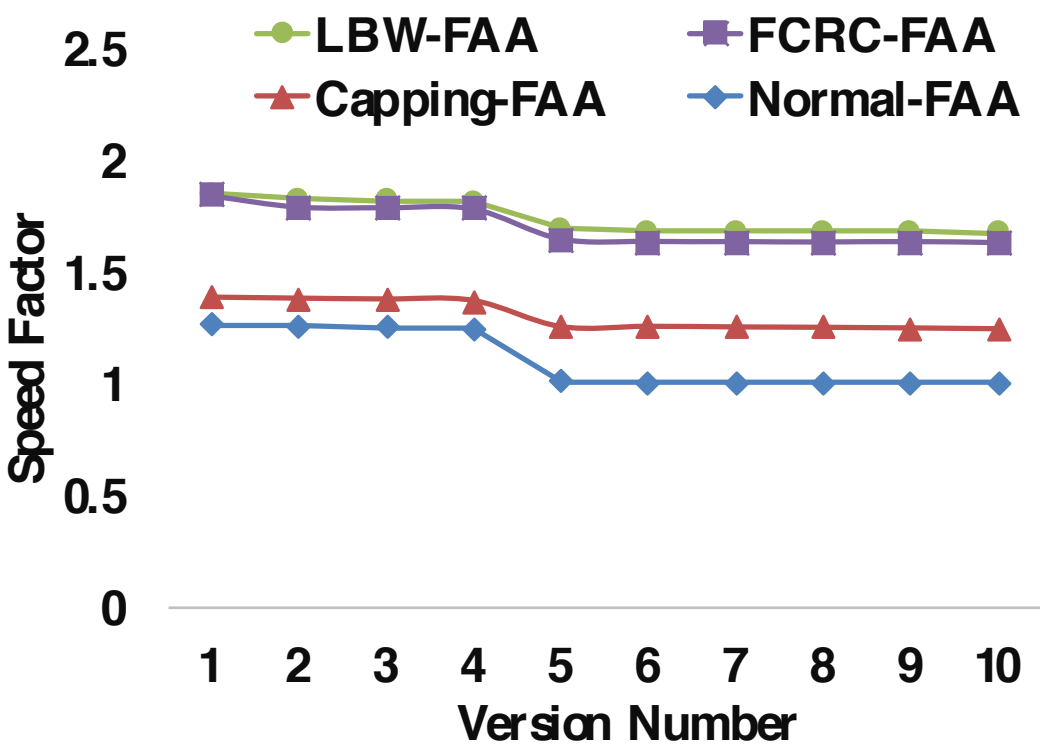


# Speed Factor Comparison



# Detailed Comparison



# Agenda

- Deduplication and Restore
- Data Chunk Rewrite Preliminary
- Container Capping Introduction and Limitations
- Proposed Solutions
  - Flexible Container Referenced Count based Design (FCRC)
  - Sliding Look-Back Window (LBW)
- Evaluations
- Conclusions and Future Work

# Conclusions and Future Work

- Investigated the advantages and limitations of capping
- Proposed an improved scheme based on capping (FCRC), which reduces container reads and rewrites fewer data chunks
- Proposed a new scheme called sliding look-back window based rewrite scheme, which solves the cutting boundary issue of capping and integrated the rewrite algorithm of FCRC
- LBW achieves the best restore performance (highest speed factor) in our evaluations
- In the future, we will investigate the restore friendly GC mechanism.

# Thanks!
# Q&A





# Reference


[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

[2] Zhichao Cao, Hao Wen, Fenggang Wu, and David H. C. Du. 2018. ALACC: Accelerating restore performance of data deduplication systems using adaptive look-ahead window assisted chunk caching. In Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18). 309–324.

[3] Mark Lillibridge, Kave Eshghi, and Deepavali Bhag- wat. Improving restore speed for backup systems that use inline chunk-based deduplication. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 183–198, 2013.

[4] Youngjin Nam, Guanlin Lu, Nohhyun Park, Weijun Xiao, and David HC Du. Chunk fragmentation level: An effective indicator for read performance degrada- tion in deduplication storage. In *2011 IEEE Inter- national Conference on High Performance Computing and Communications (HPCC)*, pages 581–586. IEEE, 2011

[5] Young Jin Nam, Dongchul Park, and David HC Du. Assuring demanded read performance of data dedupli- cation storage with backup datasets. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Sys- tems (MASCOTS 12)*, pages 201–208. IEEE, 2012.

[6] Michal Kaczmarczyk, Marcin Barczynski, Wojciech Kilian, and Cezary Dubnicki. Reducing impact of data fragmentation caused by in-line deduplication. In *Pro- ceedings of the 5th Annual International Systems and Storage Conference, Haifa, Israel (SYSTOR 12)*, pages 1–12, 2012.

[7] Min Fu, Dan Feng, Yu Hua, Xubin He, Zuoning Chen, Wen Xia, Fangting Huang, and Qing Liu. Acceler- ating restore and garbage collection in deduplication-based backup systems via exploiting historical infor- mation. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 181–192, 2014.

[8] Min Fu, Dan Feng, Yu Hua, Xubin He, Zuoning Chen, Jingning Liu, Wen Xia, Fangting Huang, and Qing Liu. Reducing fragmentation for in-line deduplication backup storage via exploiting backup history and cache knowledge. *IEEE Transactions on Parallel and Dis- tributed Systems*, 27(3):855–868, 2016.

[9] Yujuan Tan, Baiping Wang, Jian Wen, Zhichao Yan, Hong Jiang, and Witawas Srisa-an. Improving re- store performance in deduplication-based backup sys- tems via a fine-grained defragmentation approach. *IEEE Transactions on Parallel and Distributed Sys- tems*, 2018.

[10] http://tracer.filesystems.org/.