ALACC: Accelerating Restore Performance of Data Deduplication Systems Using Adaptive Look-Ahead Window Assisted Chunk Caching

Zhichao Cao, Hao Wen, Fenggang Wu and David H.C. Du

University of Minnesota, Twin Cities

02/15/2018

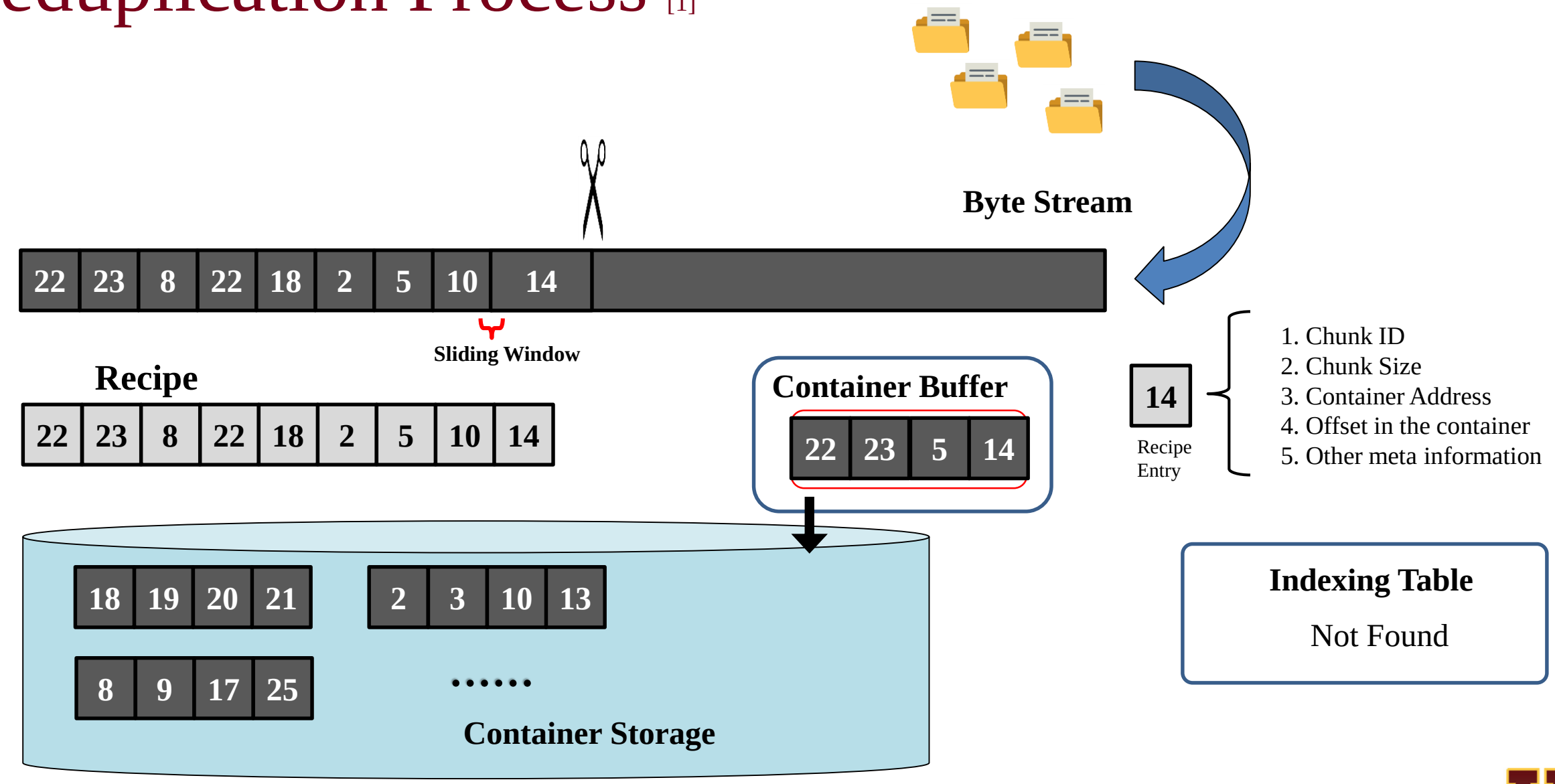
Agenda

- Deduplication Process
- Restore Process with Different Caching Schemes
  - Container/chunk based caching
  - Forward Assembly
- Objective and Challenges
- Proposed Approach
  - Look-ahead window assisted chunk based caching (all fixed)
  - Adaptive Look-ahead Chunk-based Caching (ALACC)
- Evaluations
- Conclusions and Future Work

Agenda

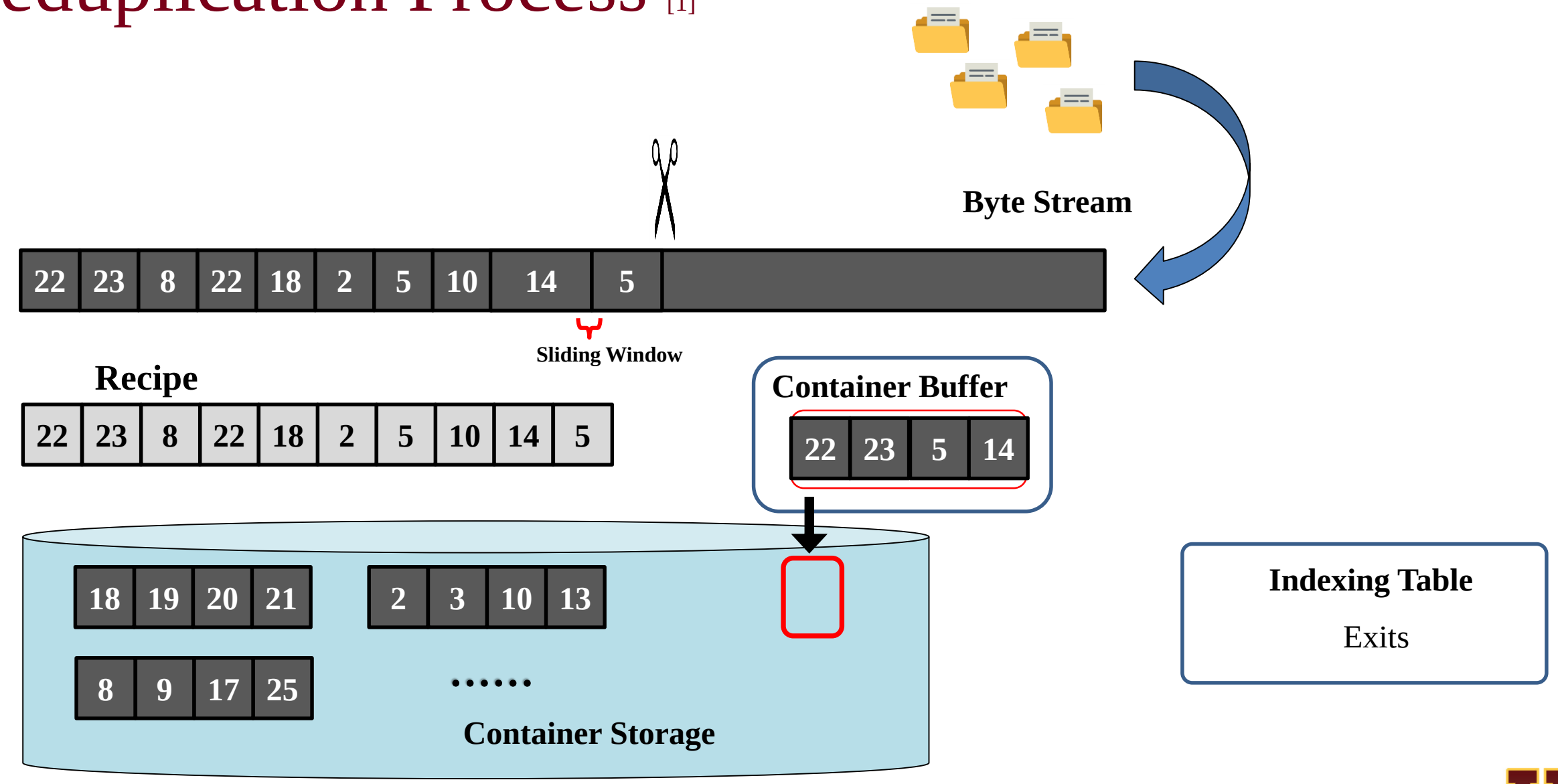
- Deduplication Process
- Restore Process with Different Caching Schemes
  - Container/chunk based caching
  - Forward Assembly
- Objective and Challenges
- Proposed Approach
  - Look-ahead window assisted chunk based caching (all fixed)
  - Adaptive Look-ahead Chunk-based Caching (ALACC)
- Evaluations
- Conclusions and Future Work

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]



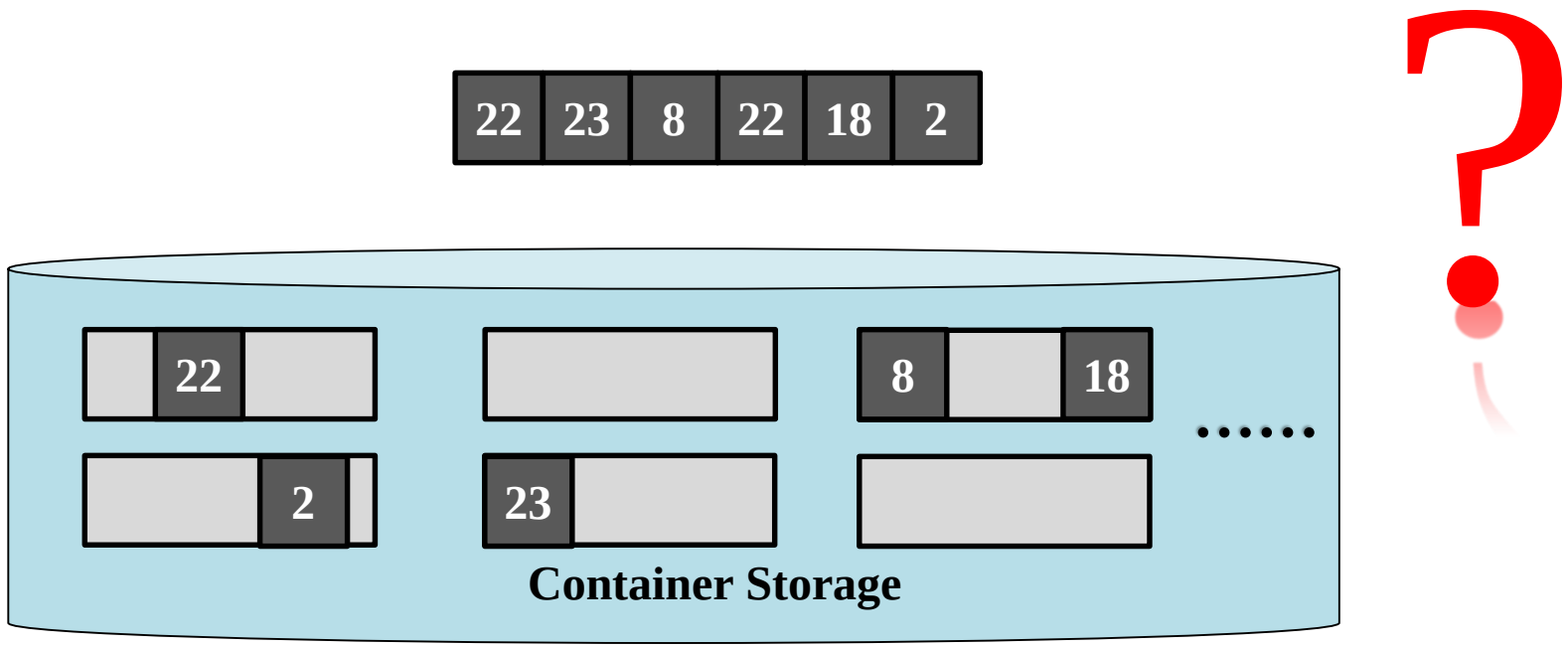
[1] Zhu B, Li K, Patterson R H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System[C]//Fast. 2008, 8: 1-14.

Agenda

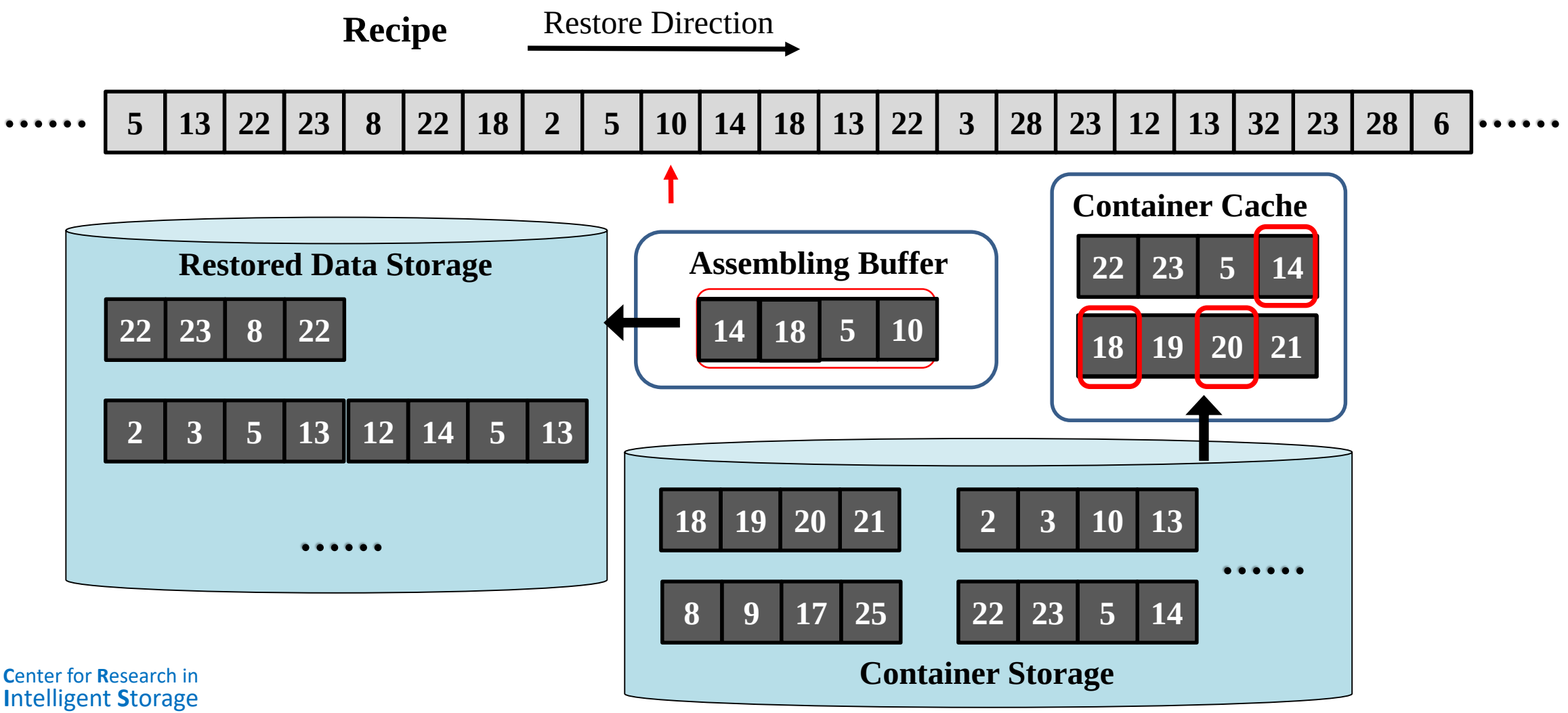
- Deduplication Process
- Restore Process with Different Caching Schemes
  - Container/chunk based caching
  - Forward Assembly
- Objective and Challenges
- Proposed Approach
  - Look-ahead window assisted chunk based caching (all fixed)
  - Adaptive Look-ahead Chunk-based Caching (ALACC)
- Evaluations
- Conclusions and Future Work

Why Improving Restore Performance is Important?

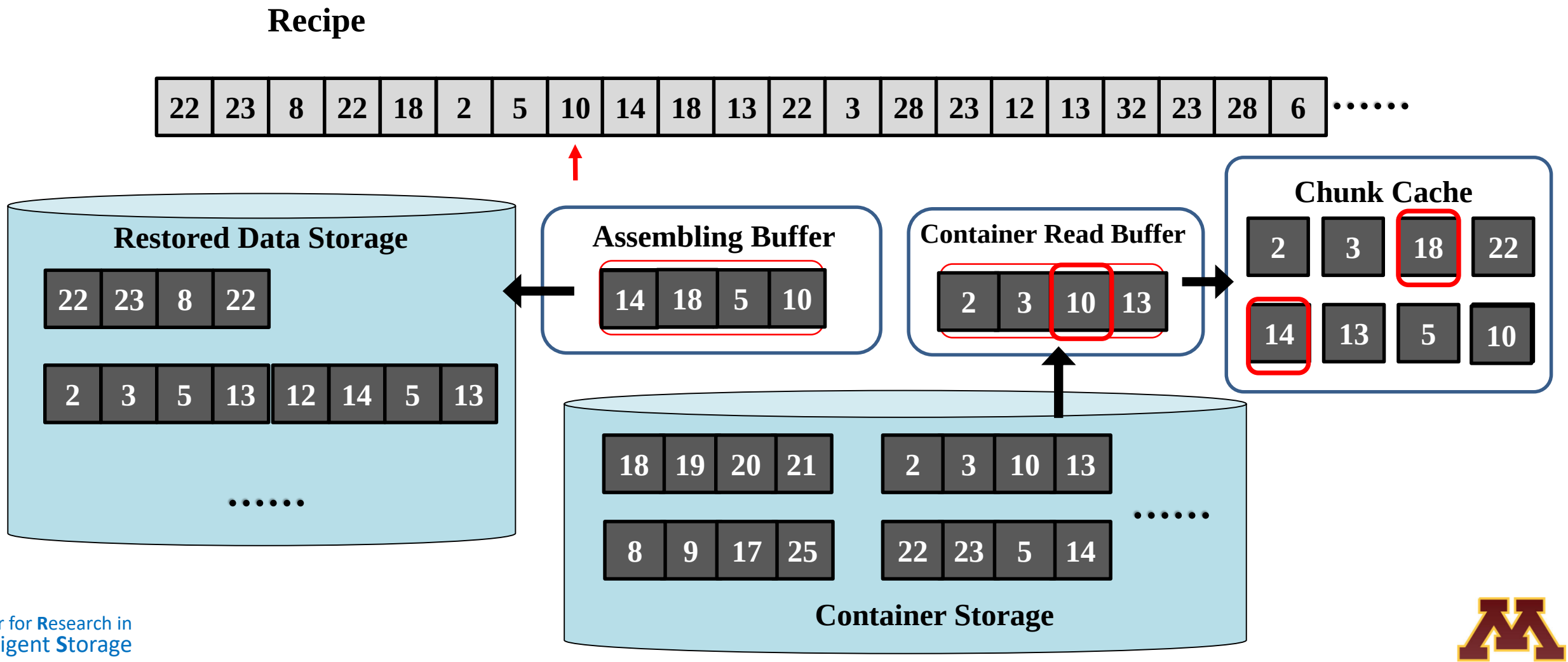
- Due to the **serious data fragmentation** and **size mismatching of requested data and I/O unite**, the restore performance is **much lower** than that of directly reading out the data which is not deduplicated.
- CPU and memory resources are limited.



Restore Process with Container-based Caching



Restore Process with Chunk-based Caching



Container-based Caching vs. Chunk-based Caching

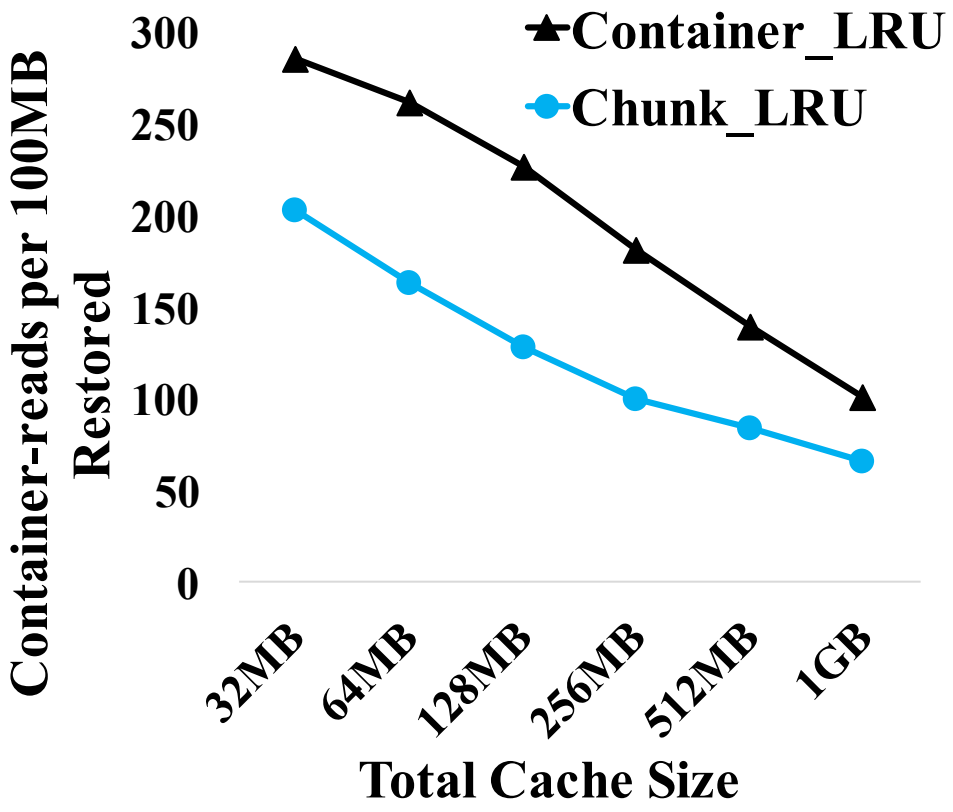
Container-based Caching

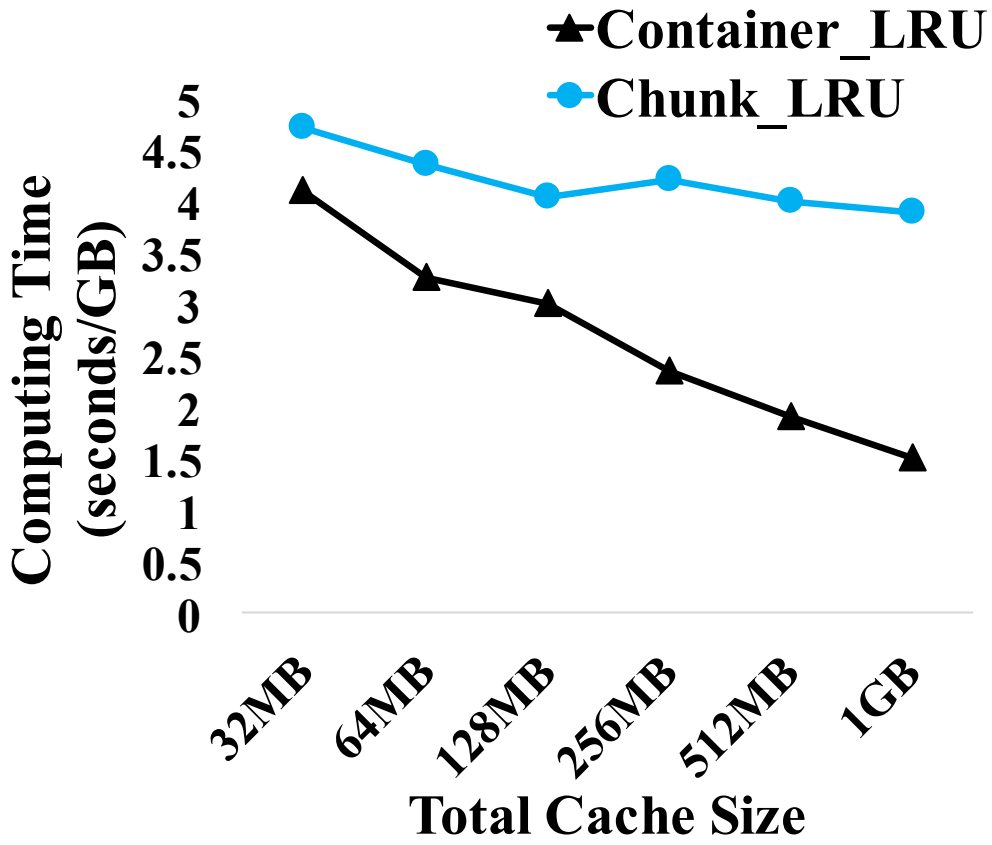
- 😀 Less operating and management overhead
- ☹️ Relatively higher cache miss ratio, especially when the caching space is limited.

Chunk-based Caching

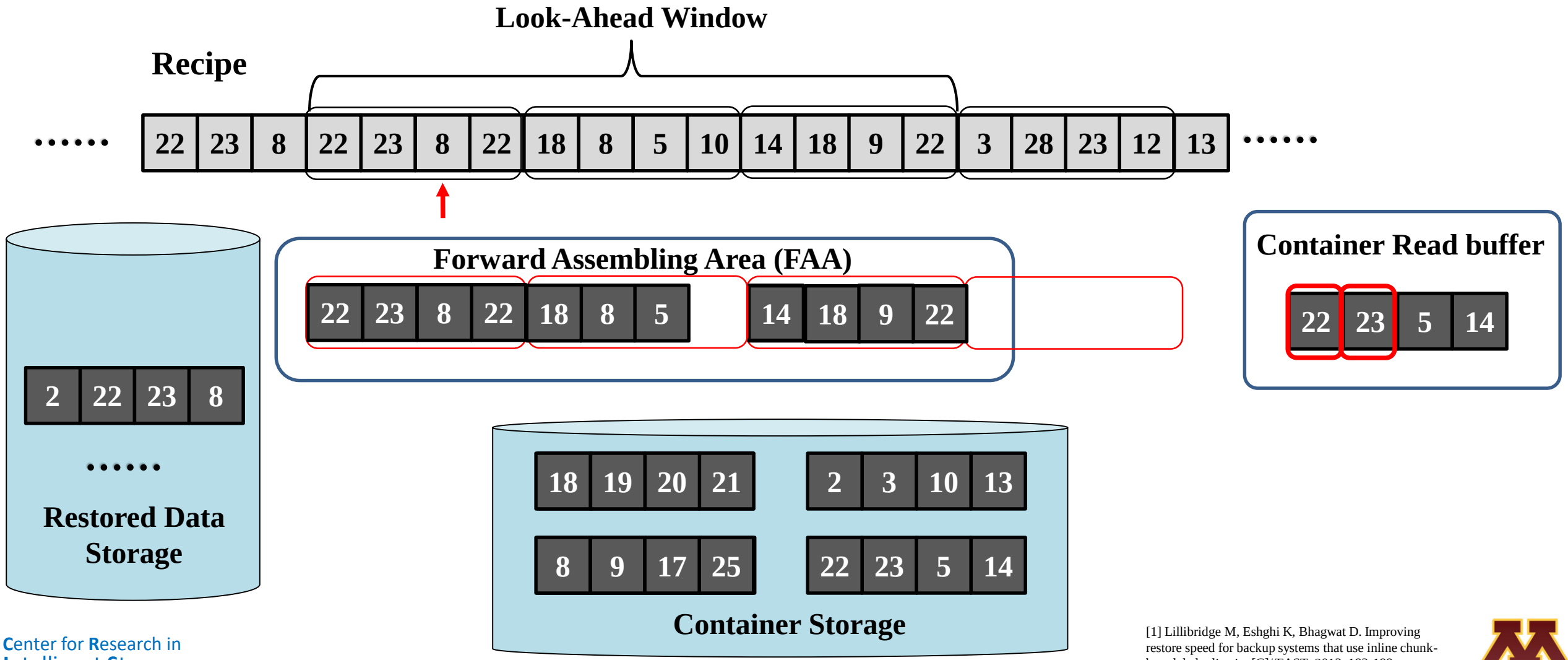
- 😀
  1. Higher cache hit ratio
  2. Even much higher if look-ahead window is applied
- ☹️ Higher operating and management overhead

Container-based Caching vs. Chunk-based Caching





Forward Assembly Scheme [1]



[1] Lillibridge M, Eshghi K, Bhagwat D. Improving restore speed for backup systems that use inline chunk-based deduplication[C]//FAST. 2013: 183-198.

Chunk-based Caching vs. Forward Assembly

Chunk-based Caching

😀

1. When chunks are re-used in a relatively long distance (larger than FAA), caching is more effective

☹️ Higher operating and management overhead

Forward Assembly

😀

1. Highly efficient, when chunks from the same container are used most in the FAA range
2. Low operating and management overhead

☹️ Workload sensitive, requires good workload locality

Agenda

- Deduplication Process
- Restore Process with Different Caching Schemes
  - Container/chunk based caching
  - Forward Assembly
- Objective and Challenges
- Proposed Approach
  - Look-ahead window assisted chunk based caching (all fixed)
  - Adaptive Look-ahead Chunk-based Caching (ALACC)
- Evaluations
- Conclusions and Future Work

Objective and Challenges

- **Objective:**
  - Forward assembly + chunk-based caching + LAW (limited memory space)

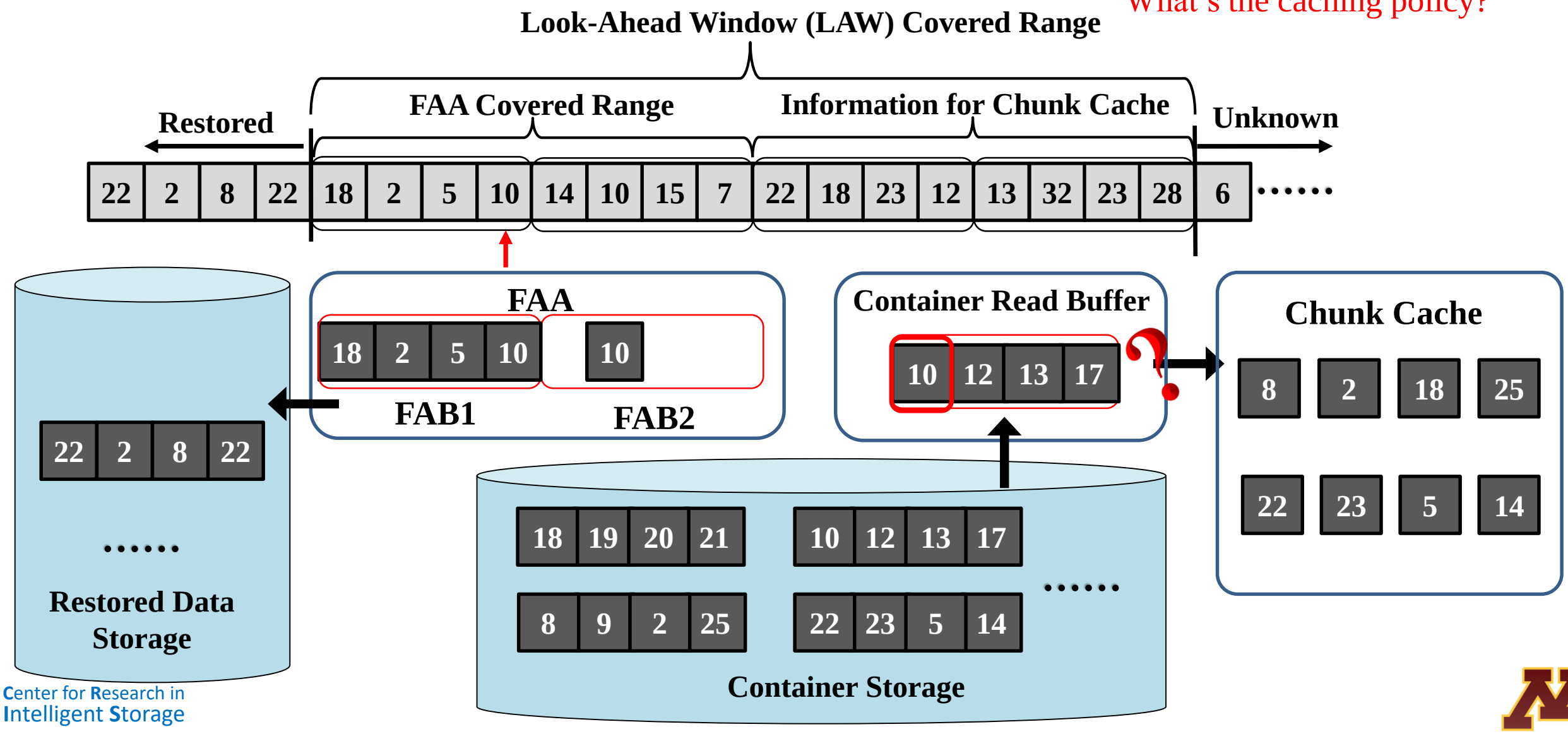
- **Challenges**
  - When the total size of available memory for restore is limited and fixed, how to use these schemes in an efficient way, is unclear.
  - How to make better trade-offs to achieve fewer container-reads, but limit the computing overhead including the LAW, caching and forward assembly overhead.
  - How to make the design adapt to the changing workload is very challenging.

Agenda

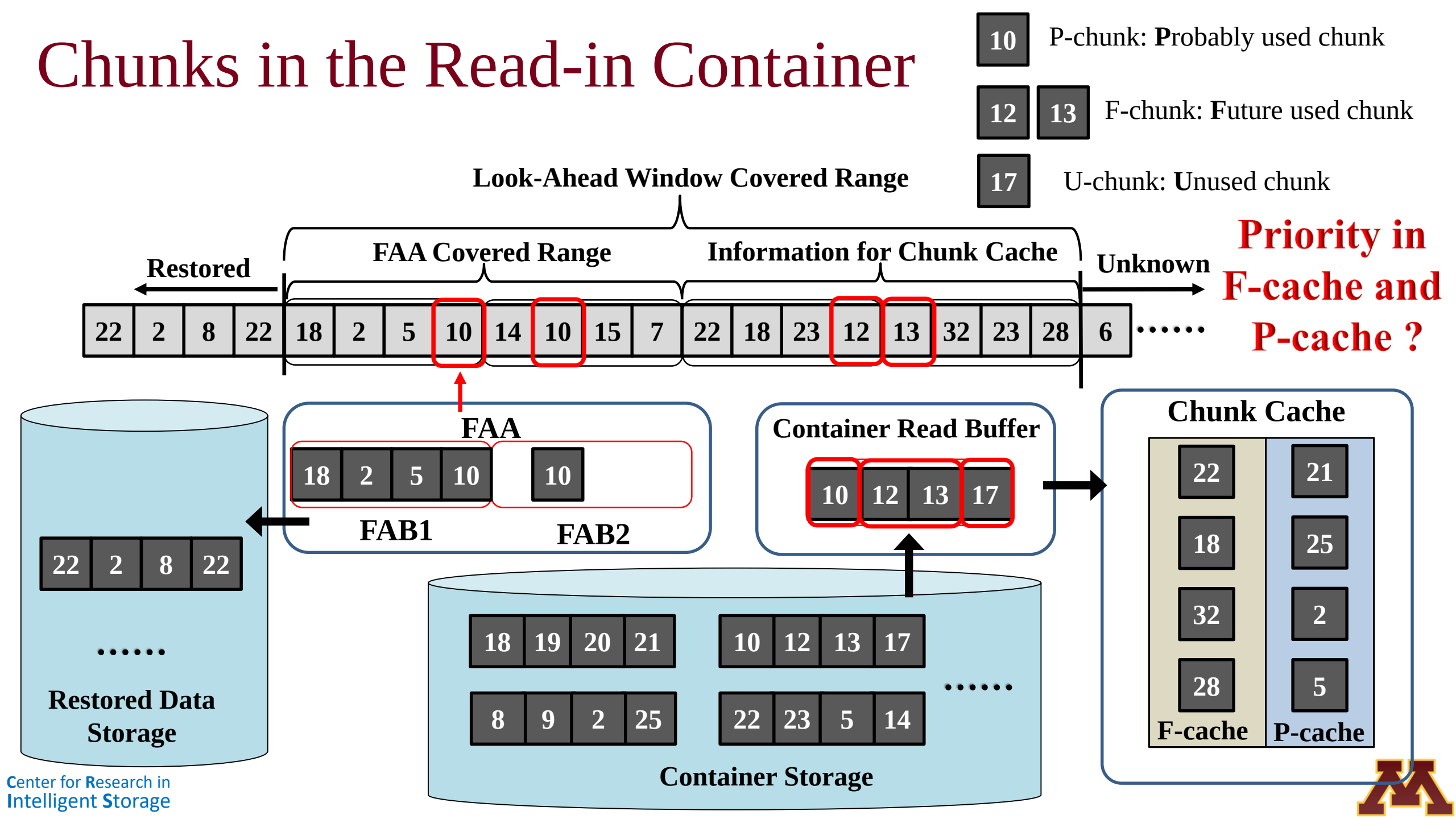
- Deduplication Process
- Restore Process with Different Caching Schemes
  - Container/chunk based caching
  - Forward Assembly
- Objective and Challenges
- Proposed Approach
  - Look-ahead window assisted chunk based caching (all fixed)
  - **A**daptive **L**ook-**A**head **C**hunk-based **C**aching (ALACC)
- Evaluations
- Conclusions and Future Work

Look-ahead Window Assisted Chunk Cache

What's the caching policy?

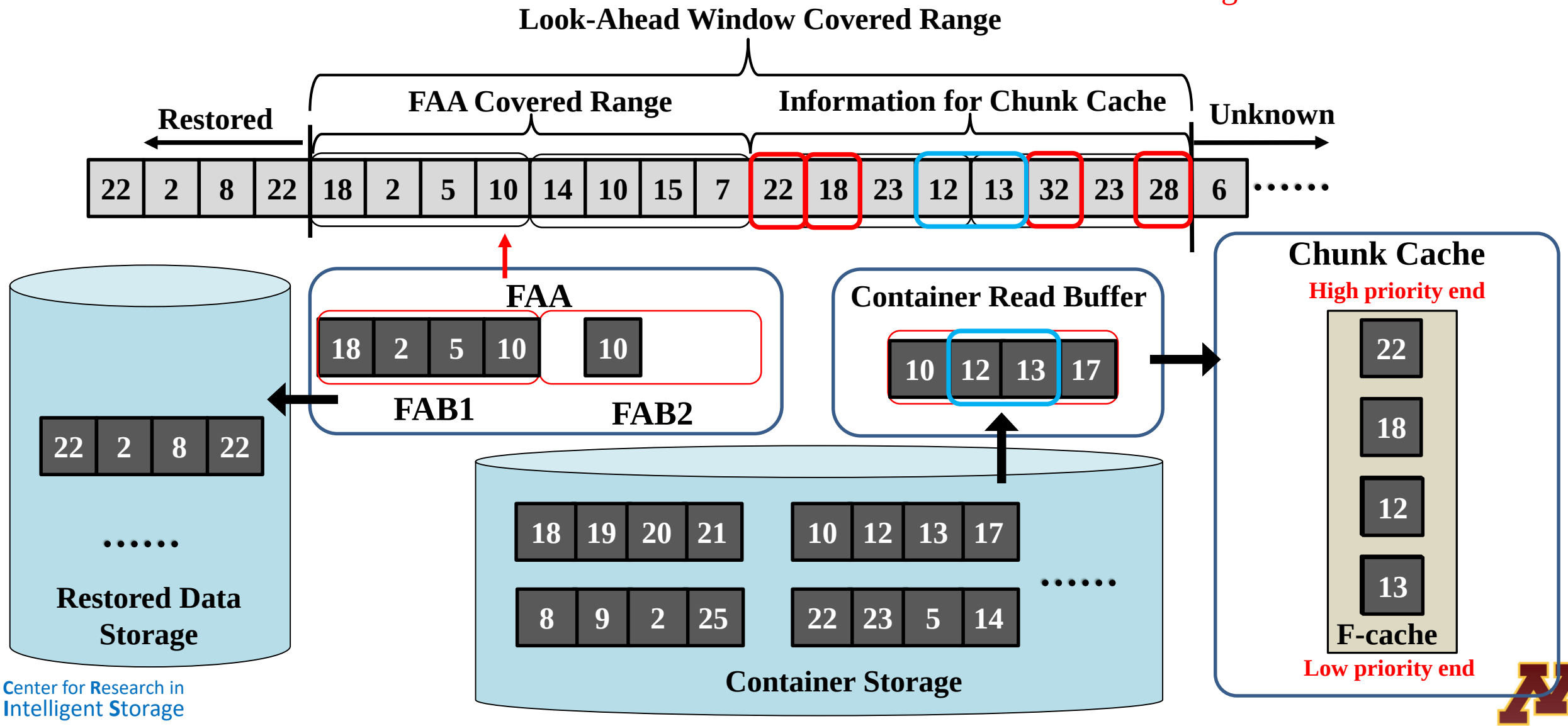


Chunks in the Read-in Container



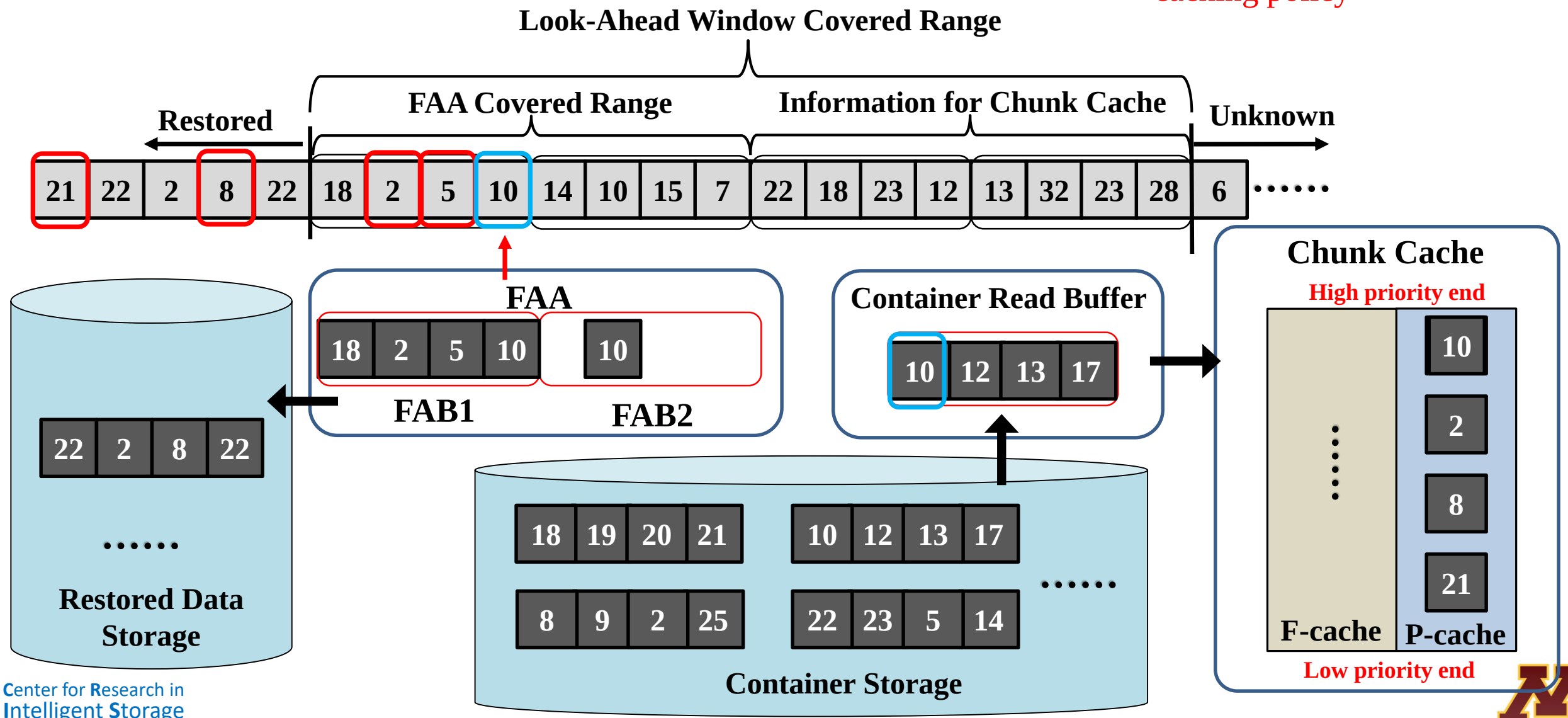
Caching Priority of F-cache

F-chunks being used in the near future have higher priority than F-chunks being used in the far future



Caching Priority of P-cache

P-cache is LRU based caching policy

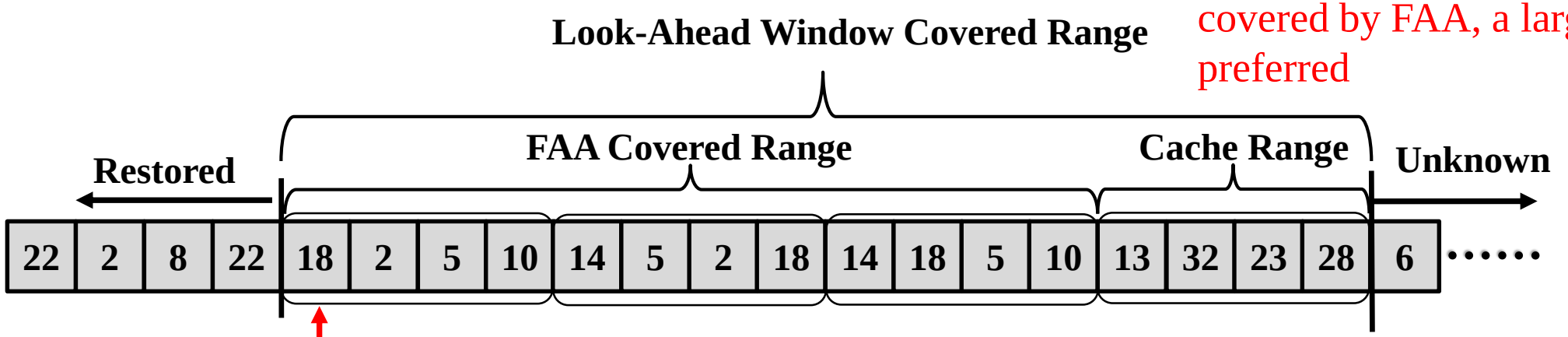


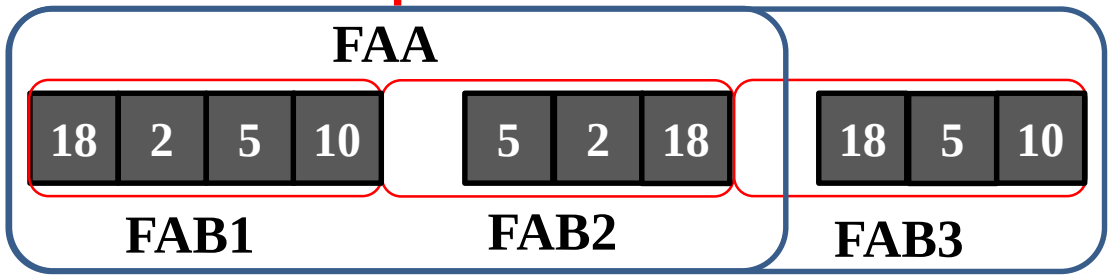
Good Enough?

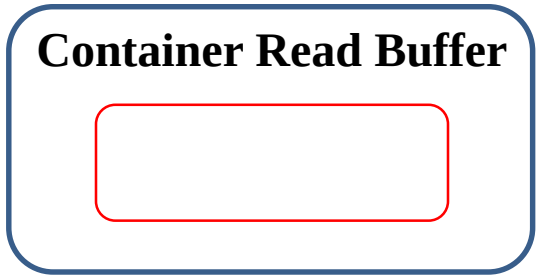
- What's the **memory space ratio** between forward assembly and chunk cache?
- What's the **size of LAW**?
  - Too large: the computing overhead large but the extra information in the LAW is wasting
  - Too small: it becomes forward assembly+LRU cache
- What if the workload locality changes?

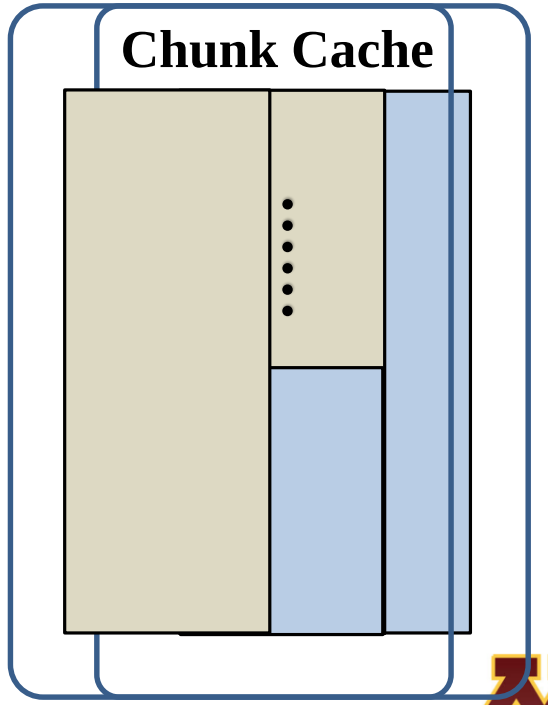
Large FAA Size (Small Cache)

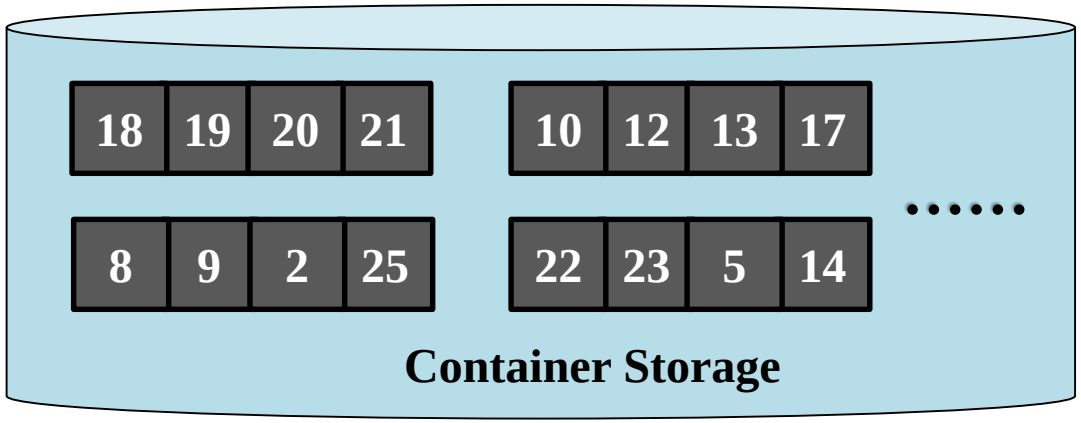
If most of the chunks (e.g., duplicated chunks) from the same container can be covered by FAA, a large FAA size is preferred





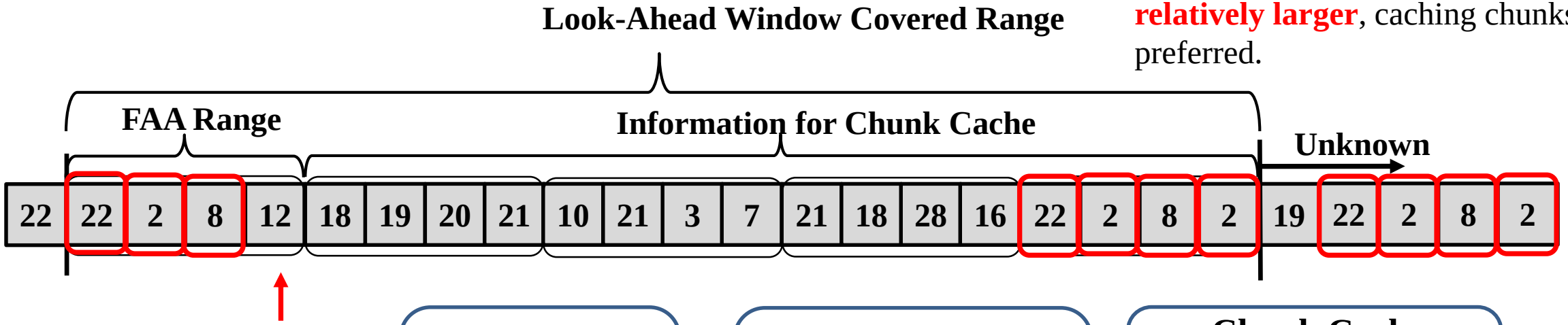


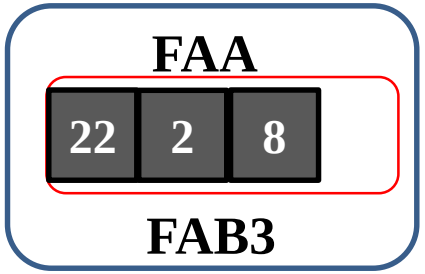


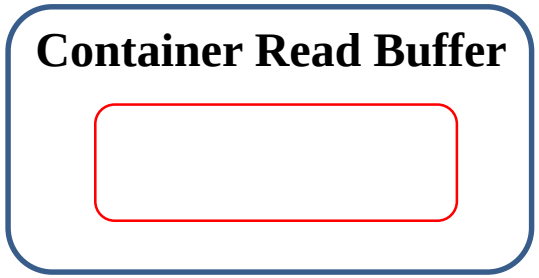


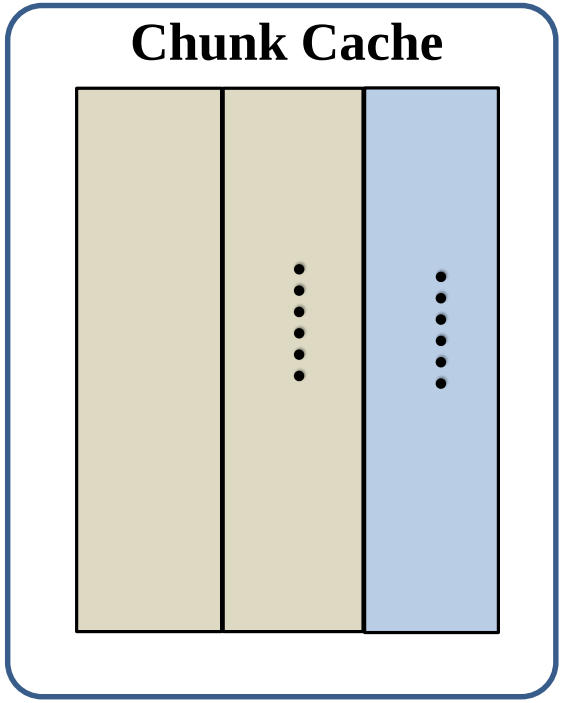
Large Chunk Cache Size (Small FAA)

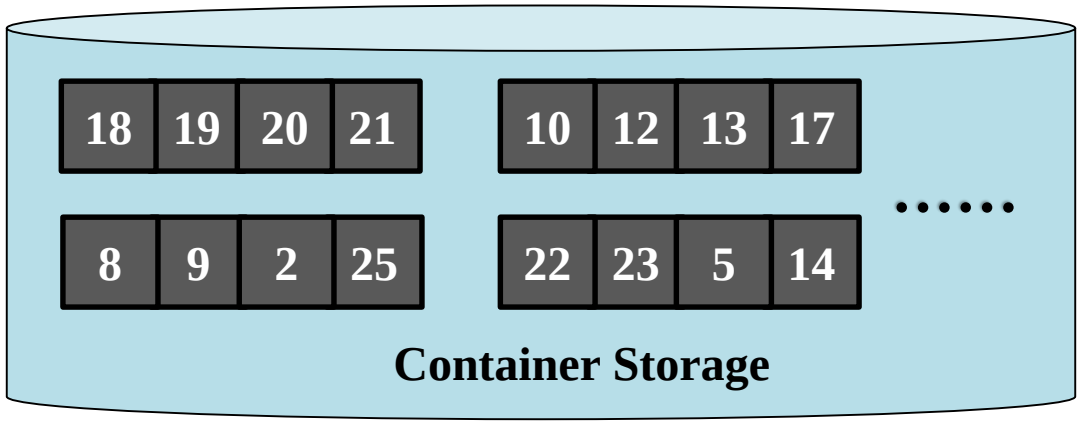
If most the chunks **reuse distance is relatively larger**, caching chunks is preferred.





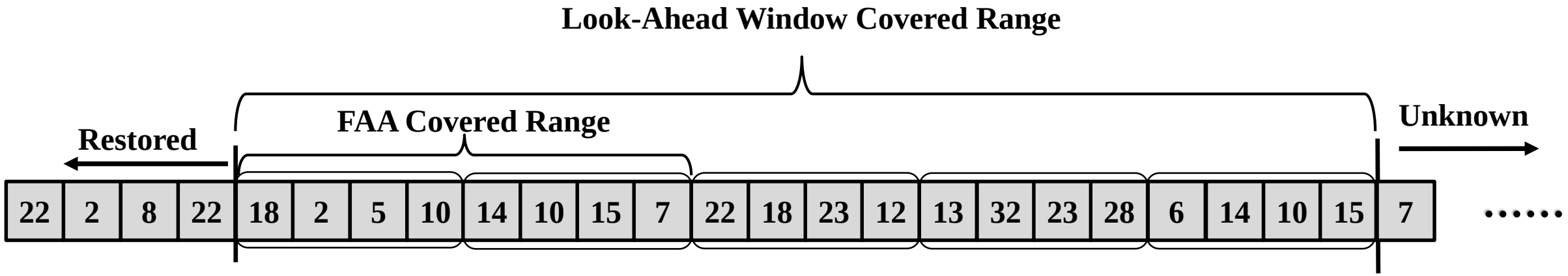




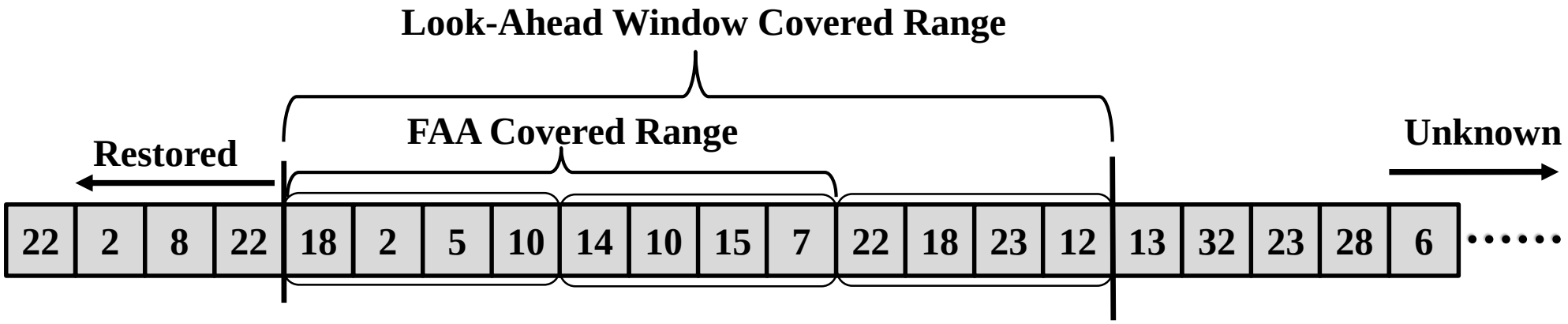


LAW Size

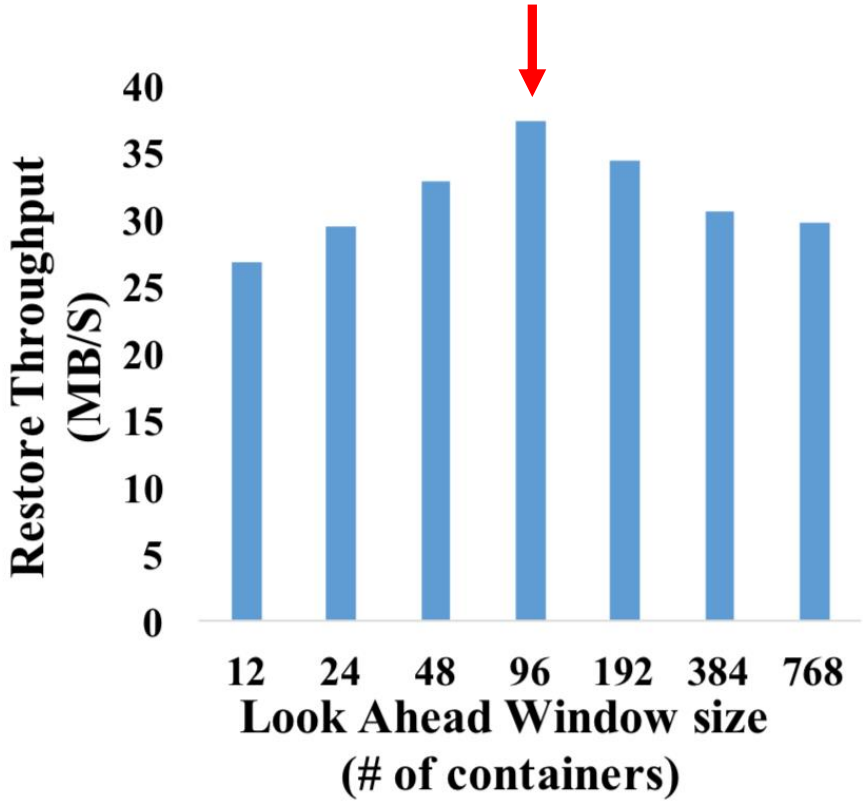
With larger LAW, more F-chunks can be potentially identified and cached. However, once the cache is full of F-chunks, further increasing the LAW size cannot improve the cache efficiency. In contrast, it will bring in more unnecessary overhead.



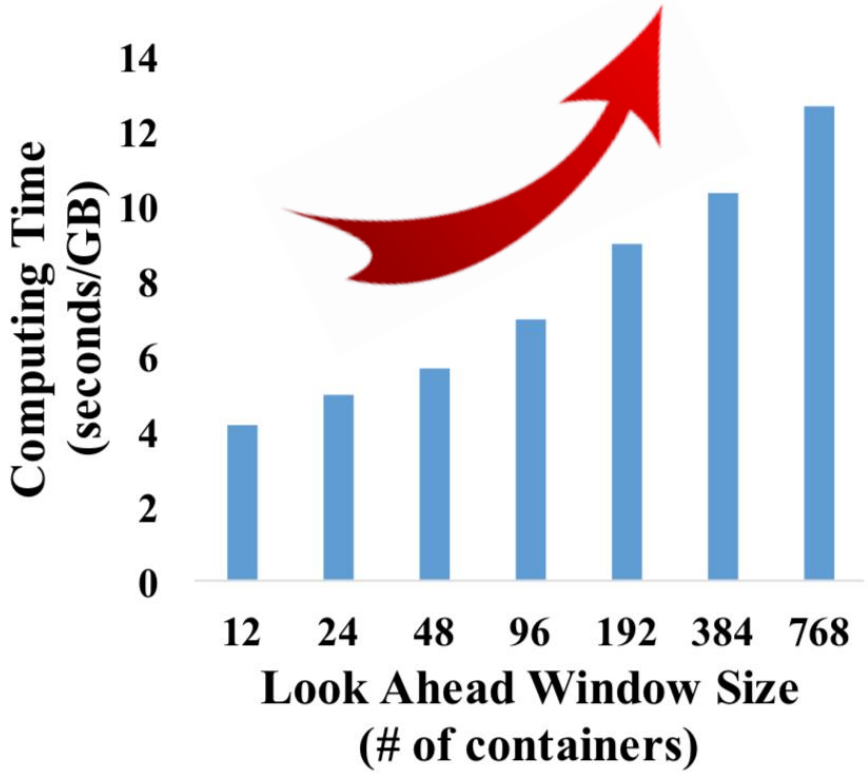
With smaller LAW, lower computing overhead can be achieved, but cache will store more P-chunks, which potentially reduce the caching efficiency.



The LAW Size Influence



(a) Restore throughput



(b) Computing time per 1GB restored

ALACC

The re-use distances of most duplicated data chunks are **within the FAA range**

OR

The data chunks in the first FAB are identified mostly as **unique data chunks** and these chunks are **stored in the same or close containers**

Increase FAA → Shrink Cache → Shrink LAW

Not satisfied

P-chunk number is very **small**

OR

F-chunk added during the restore cycle is **very large**

Enlarge Cache → Shrink FAA → Adjust LAW

Not satisfied

Adjust LAW

Agenda

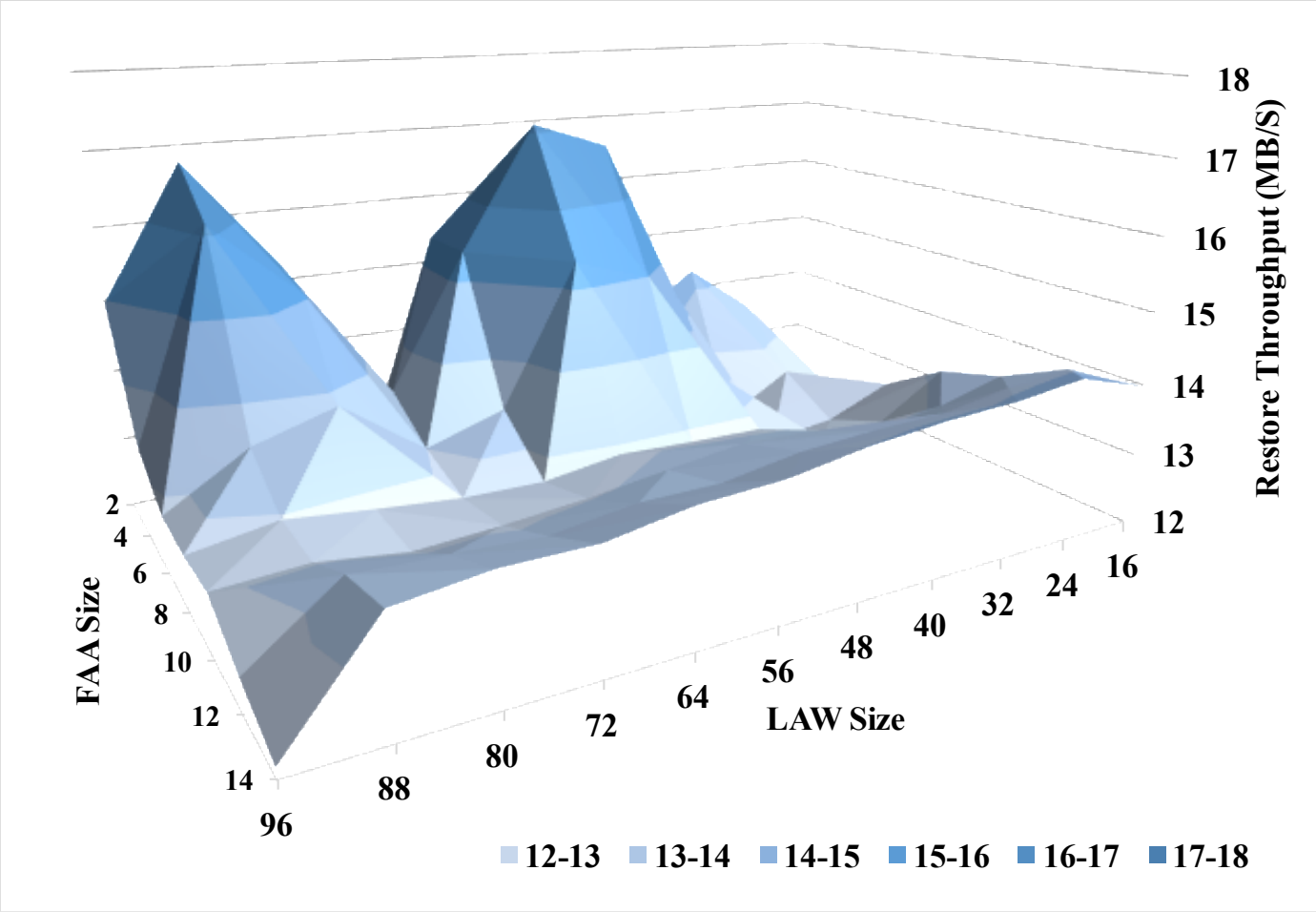
- Deduplication Process
- Restore Process with Different Caching Schemes
  - Container/chunk based caching
  - Forward Assembly
- Objective and Challenges
- Proposed Approach
  - Look-ahead window assisted chunk based caching (all fixed)
  - Adaptive Look-ahead Chunk-based Caching (ALACC)
- Evaluations
- Conclusions and Future Work

Experiment Setup

- Five Caching Designs:
  - LRU-based container caching **(Container_LRU)**
  - LRU-based chunk caching **(Chunk_LRU)**
  - Forward assembly **(FAA)**
  - Optimal configuration with fixed forward assembly and chunk-based caching **(Fix_Opt)**
  - **ALACC**
- Four Traces:
  - 2 FSL traces from FSL /home directory snapshots of the year 2014 [1]
  - 2 EMC weekly full-backup traces [2]

[1] http://tracer.filesystems.org/.
[2] Nohhyun Park and David J Lilja. Characteriz- ing datasets for data deduplication in backup ap- plications. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1– 10. IEEE, 2010.

How We Get the Fix_Opt



Chunk cache size = total memory size – FAA size

The Fix_Opt configuration of each trace (FAA/chunk cache/LAW size in container unite)

	FSL_1	FSL_2	EMC_1	EMC_2
Size	4/12/56	6/10/72	2/14/92	4/12/64

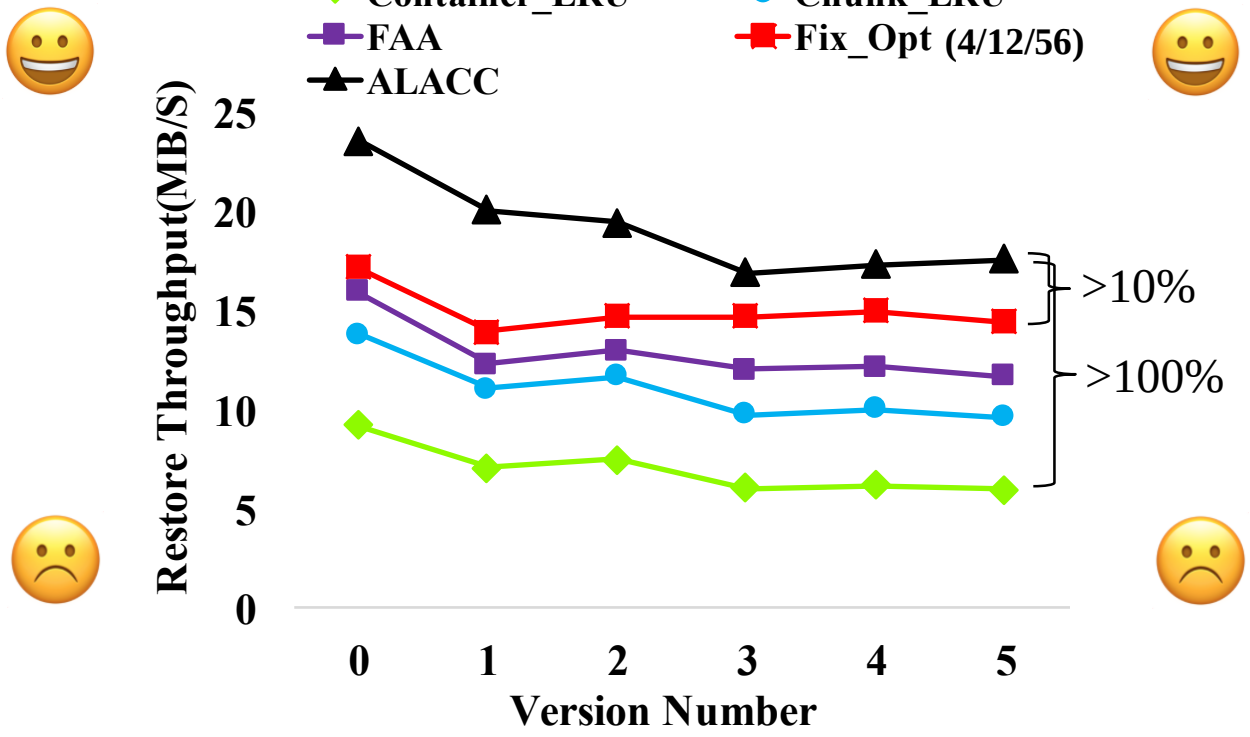
We run **all possible configurations** for each trace to discover the optimal throughput. Notice that we need tens of experiments to find out the optimal configurations of Fix_Opt which is almost **impossible to carry out in a real-world** production scenario.

Restore Throughput

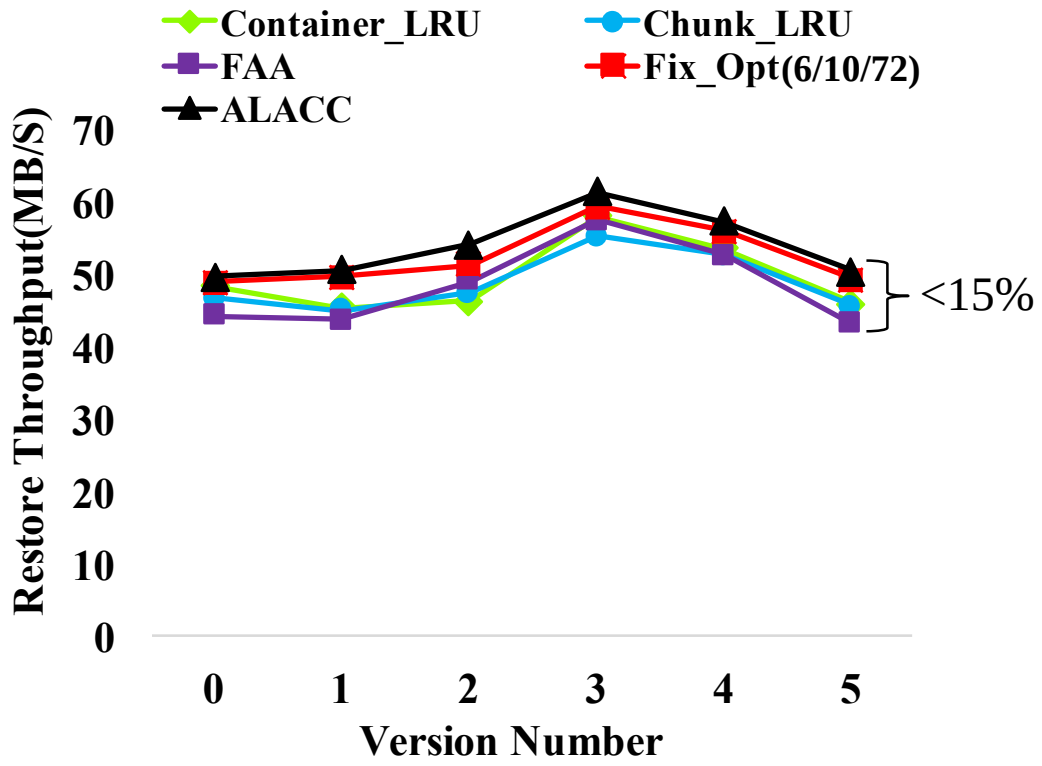
Restore Throughput (MB/S): original data stream size divided by the total restore time.

Dataset	FSL_1	FSL_2
Size	103.5GB	317.4GB
ACS[1]	4KB	4KB
DR[2]	3.82	4.88
CFL[3]	13.3	3.3

1. ACS stands for **Average Chunk Size**
2.DR stands for the **Deduplication Ratio**.
3. CFL stands for the **Chunk Fragmentation Level**



FSL_1



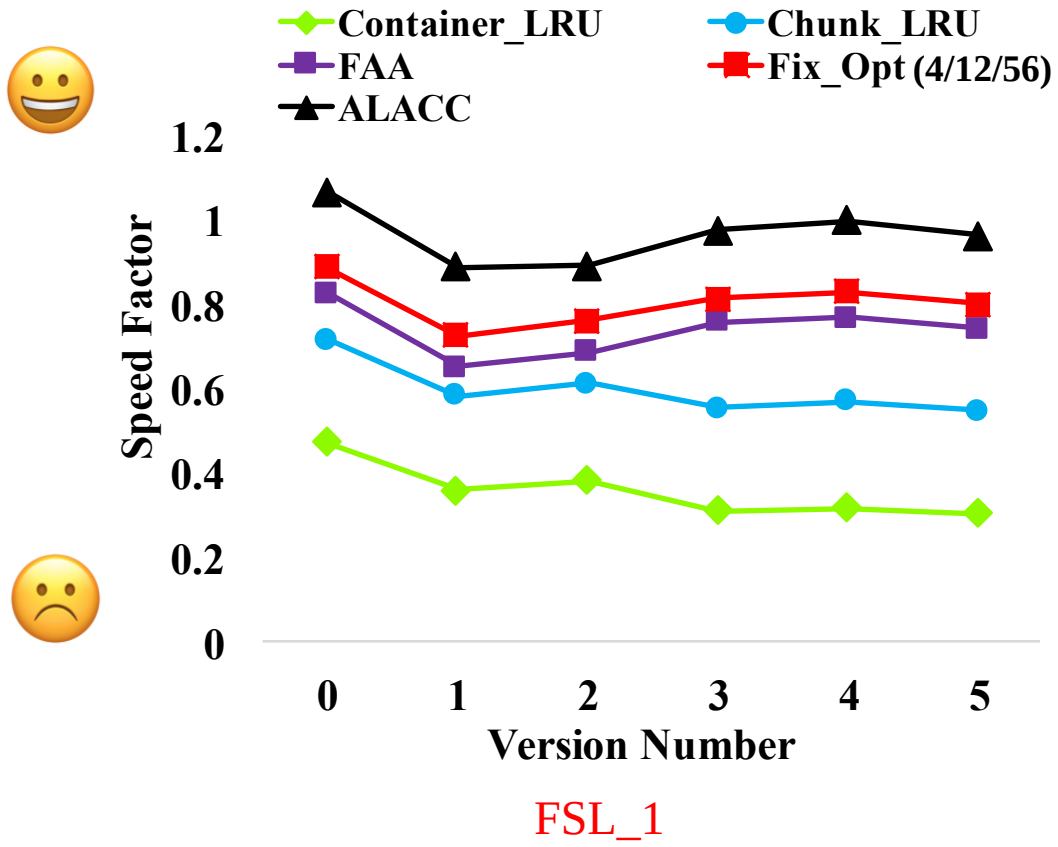
FSL_2

Speed Factor

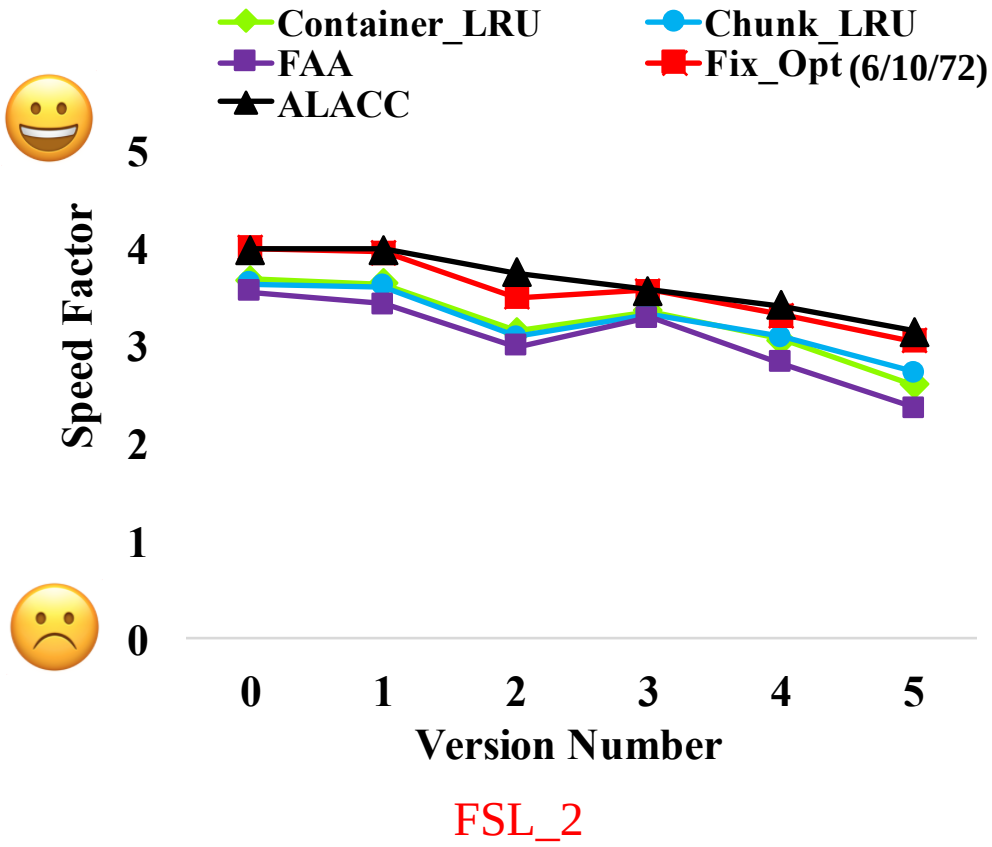
Speed Factor (MB/container-read): the mean data size restored per container read

Dataset	FSL_1	FSL_2
Size	103.5GB	317.4GB
ACS[1]	4KB	4KB
DR[2]	3.82	4.88
CFL[3]	13.3	3.3

1. ACS stands for **Average Chunk Size**
2.DR stands for the **Deduplication Ratio**.
3. CFL stands for the **Chunk Fragmentation Level**



FSL_1



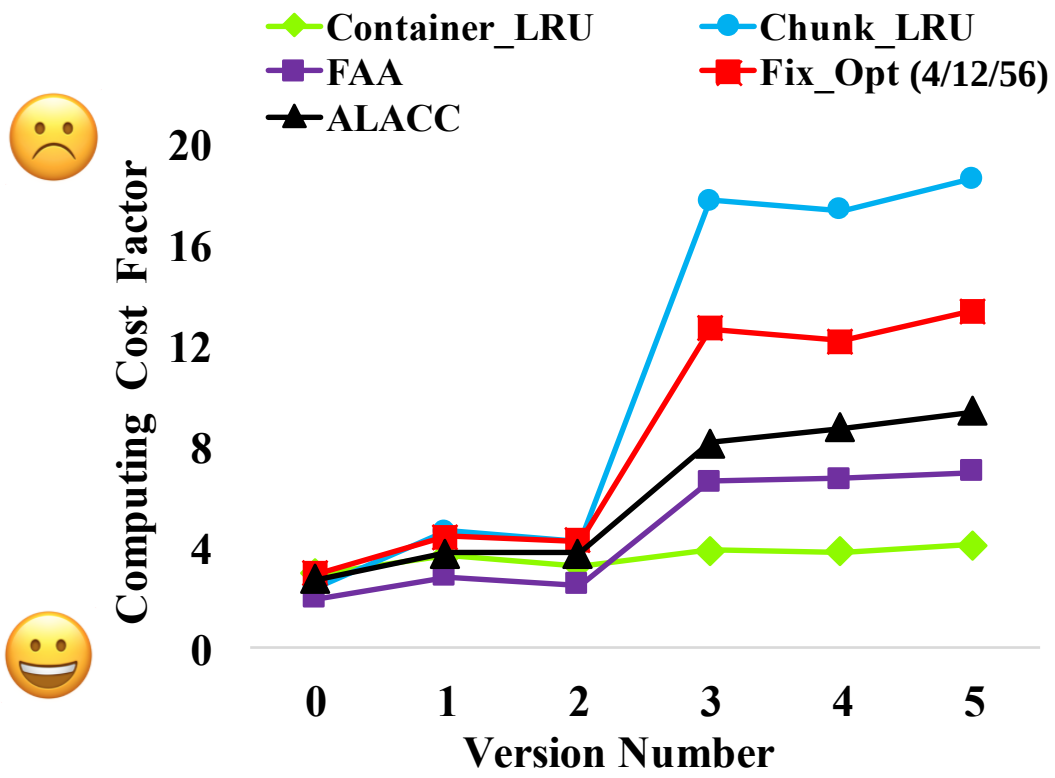
FSL_2

Computing Cost Factor

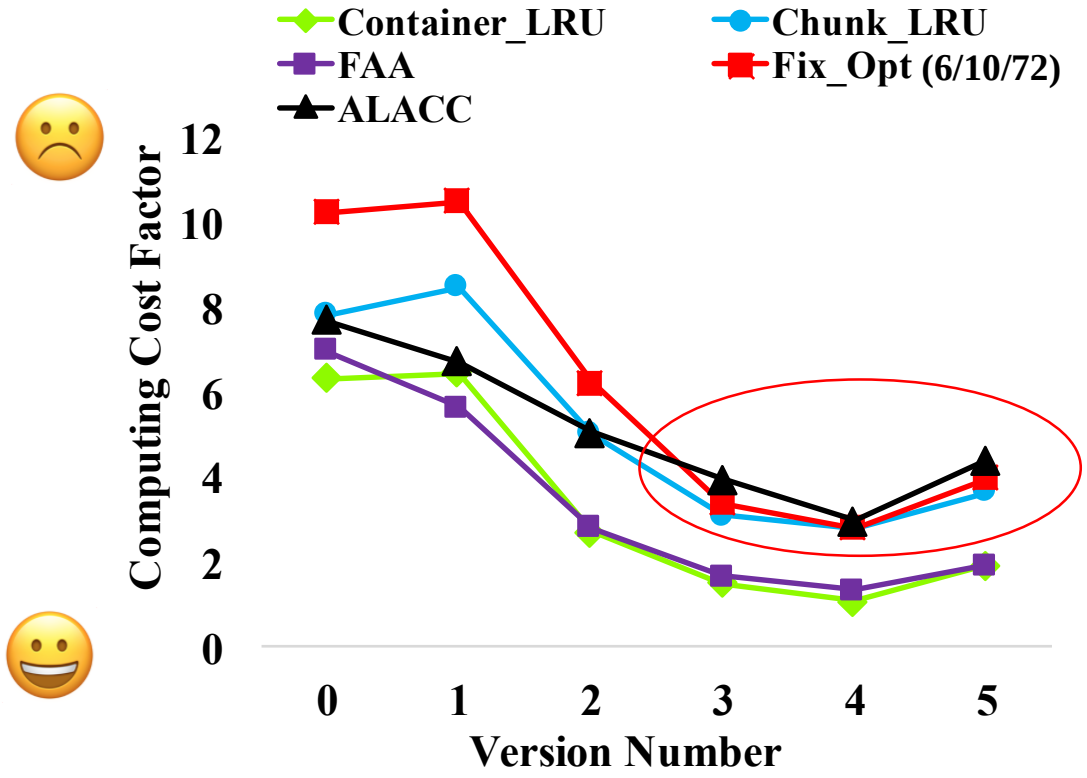
Computing Cost Factor (second/GB): the time spent on computing operations (subtracting the storage I/O time from the restore time) per GB data restored

Dataset	FSL_1	FSL_2
Size	103.5GB	317.4GB
ACS[1]	4KB	4KB
DR[2]	3.82	4.88
CFL[3]	13.3	3.3

1. ACS stands for **Average Chunk Size**
2.DR stands for the **Deduplication Ratio**.
3. CFL stands for the **Chunk Fragmentation Level**



FSL_1



FSL_2

Agenda

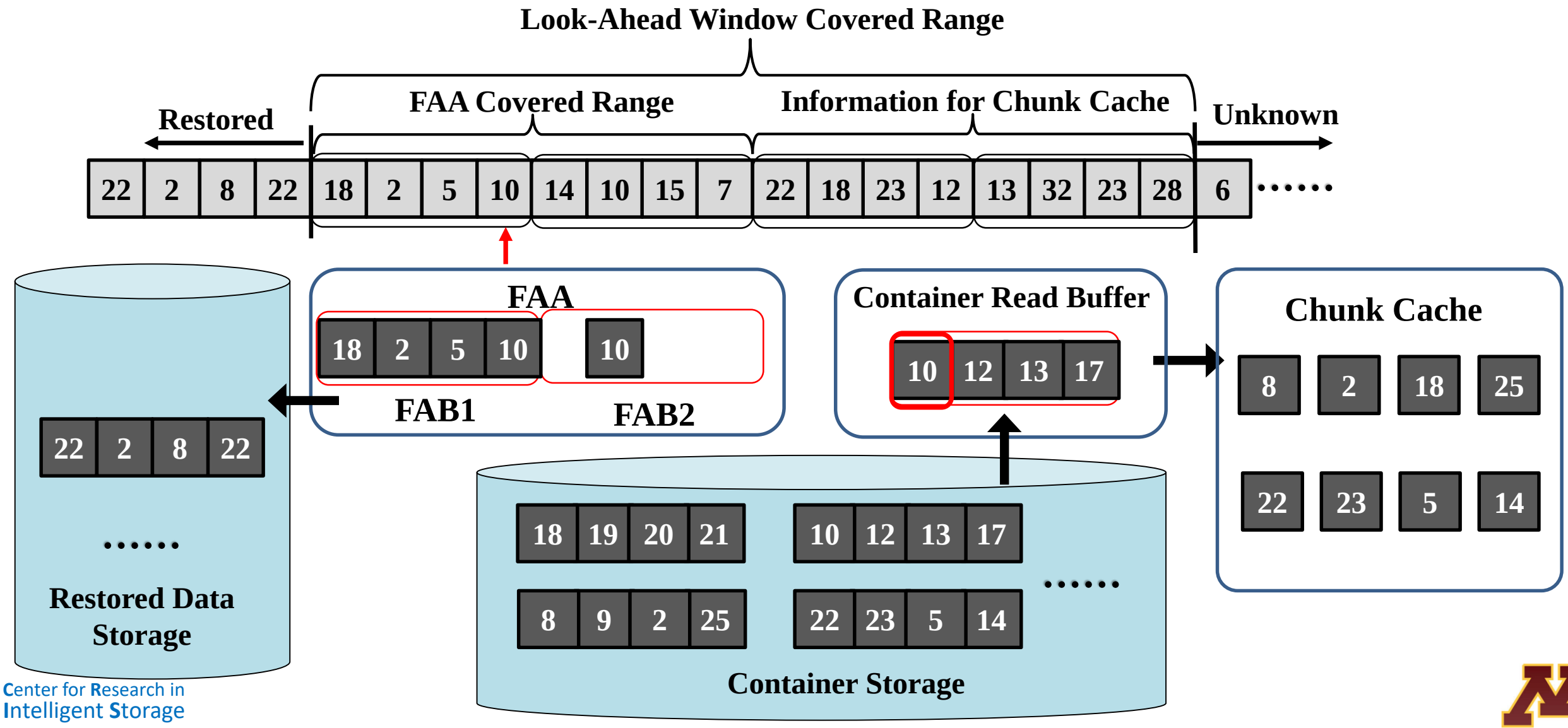
- Deduplication Process
- Restore Process with Different Caching Schemes
  - Container/chunk based caching
  - Forward Assembly
- Objective and Challenges
- Proposed Approach
  - Look-ahead window assisted chunk based caching (all fixed)
  - Adaptive Look-ahead Chunk-based Caching (ALACC)
- Evaluations
- Conclusions and Future Work

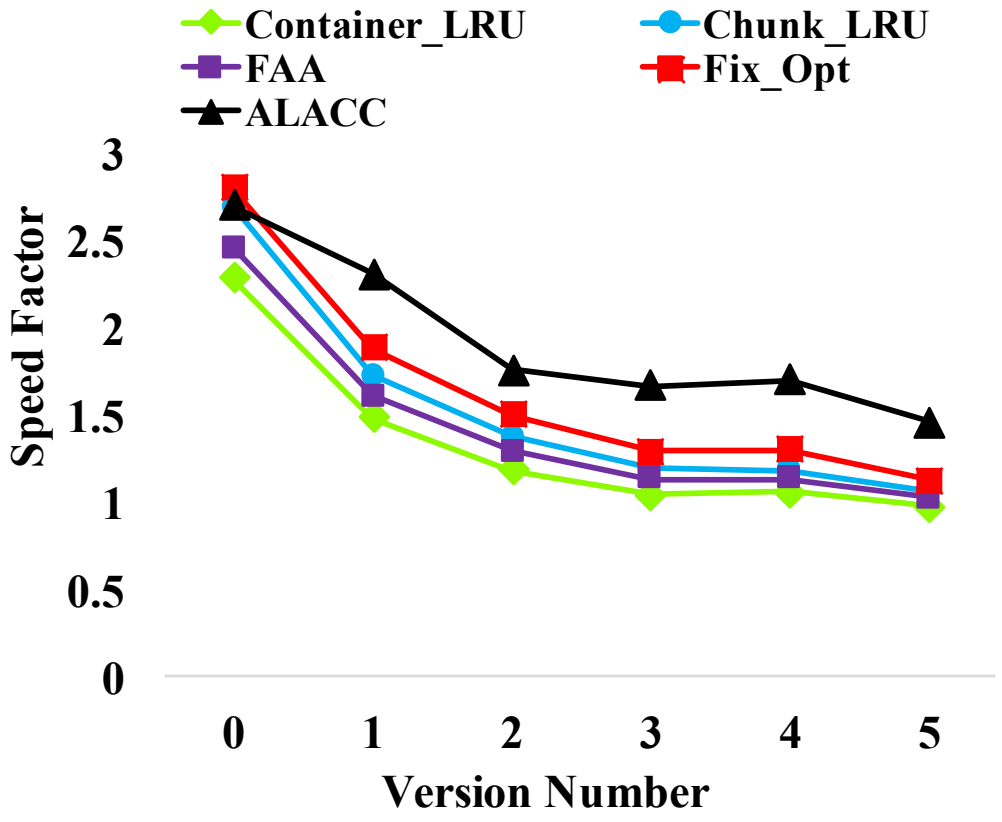
Conclusions and Future Work

- Studied the effectiveness and the efficiency of different caching mechanisms.
- Designed an adaptive algorithm called ALACC which is able to adaptively adjust the sizes of the FAA, chunk cache and LAW according to the workload changing.
- In our future work, duplicated data chunk rewriting, multi-threading implementation will be investigated and integrated with ALACC to further improve the restore performance.

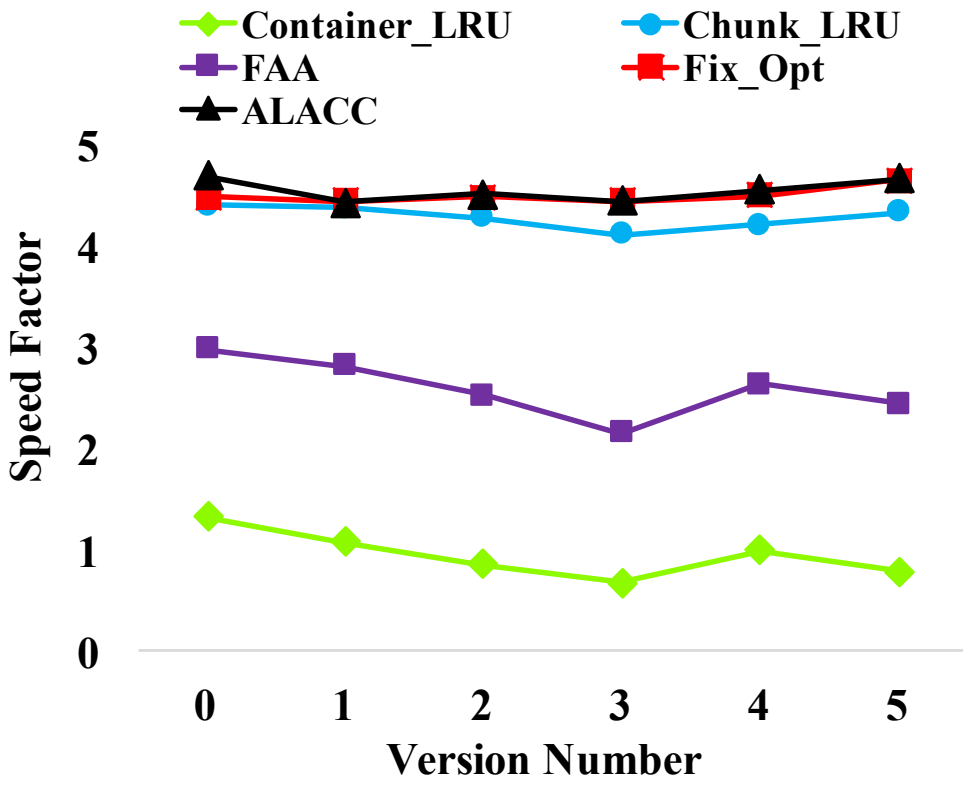
Thanks!
Q&A

Look-ahead Window Assisted Chunk Cache

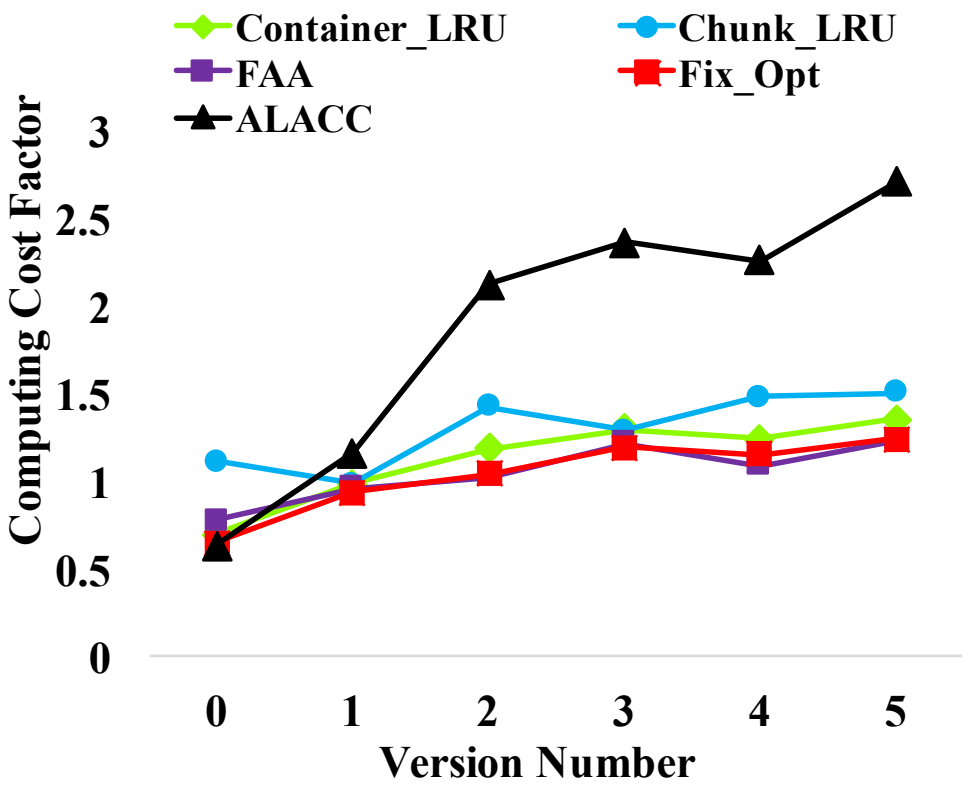



Container_LRU
Chunk_LRU
FAA
Fix_Opt
ALACC
Speed Factor
3
2.5
2
1.5
1
0.5
0
0
1
2
3
4
5
Version Number

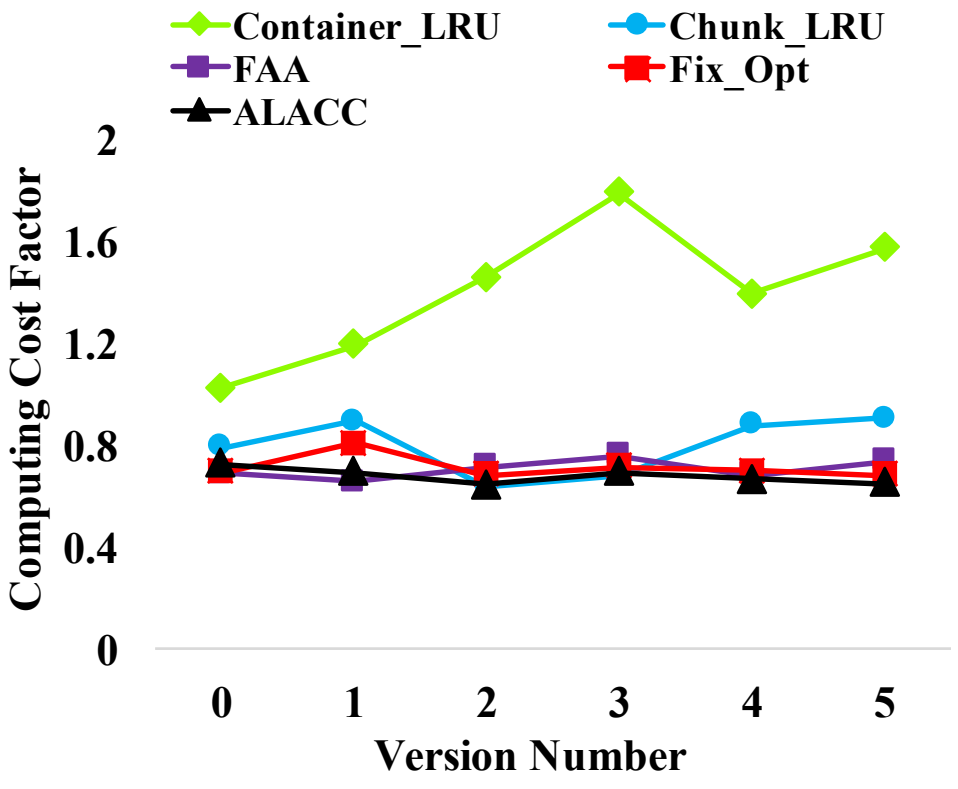
EMC_1


Container_LRU
Chunk_LRU
FAA
Fix_Opt
ALACC
Speed Factor
5
4
3
2
1
0
0
1
2
3
4
5
Version Number

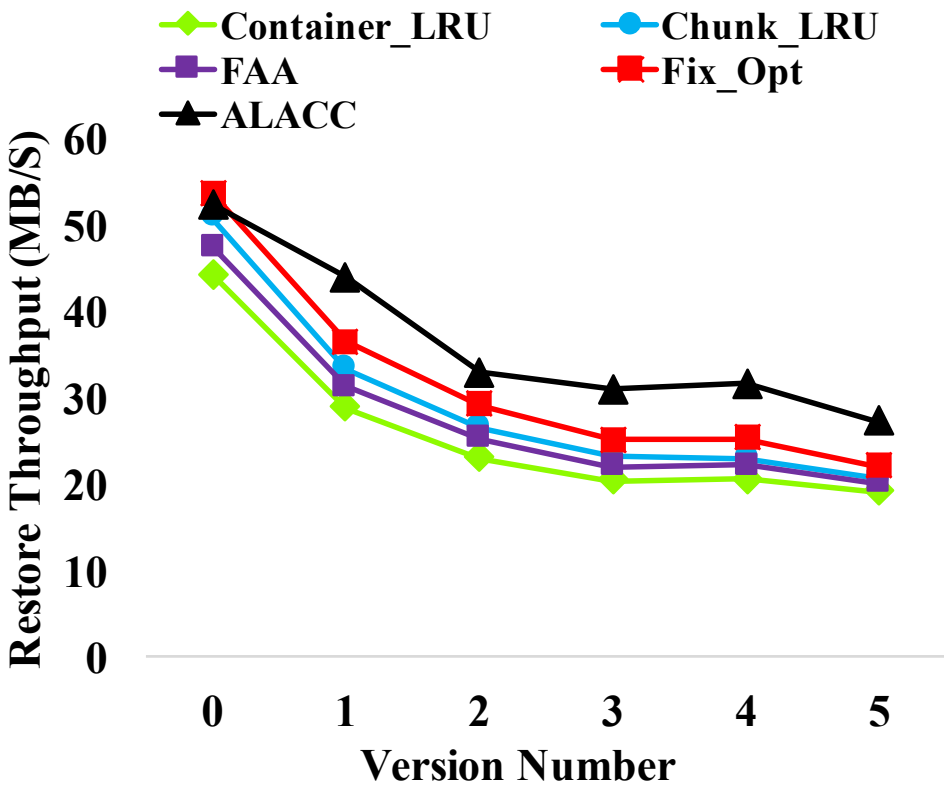
EMC_2


Container_LRU
Chunk_LRU
FAA
Fix_Opt
ALACC
Computing Cost Factor
3
2.5
2
1.5
1
0.5
0
0
1
2
3
4
5
Version Number

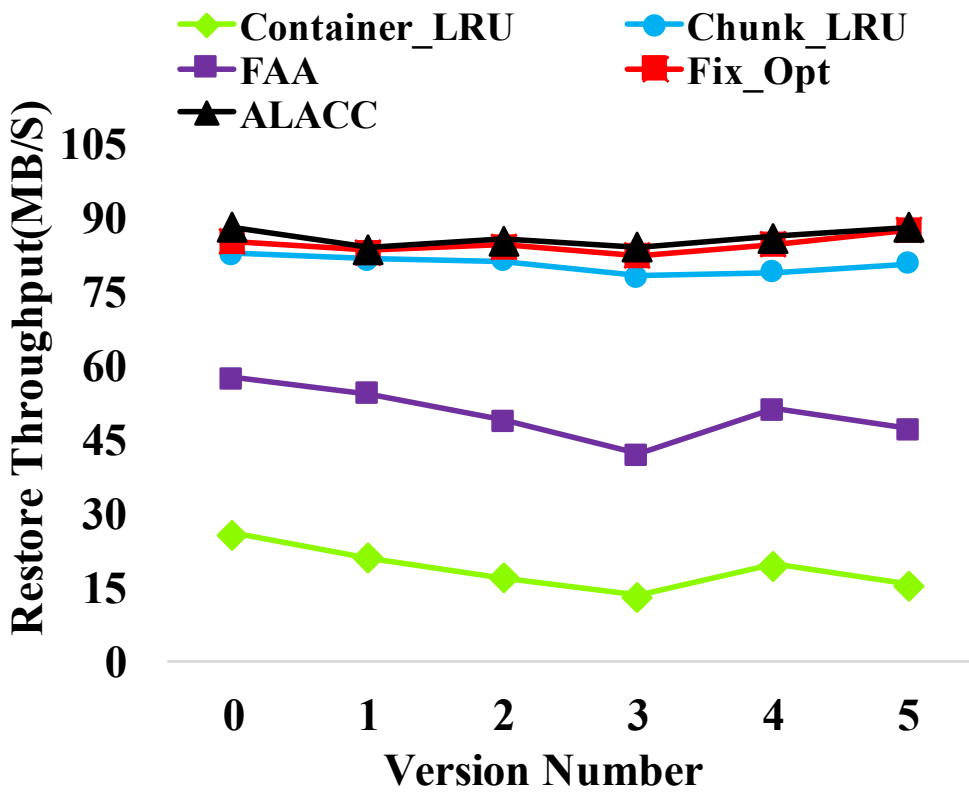
EMC_1


Container_LRU
Chunk_LRU
FAA
Fix_Opt
ALACC
Computing Cost Factor
2
1.6
1.2
0.8
0.4
0
0
1
2
3
4
5
Version Number

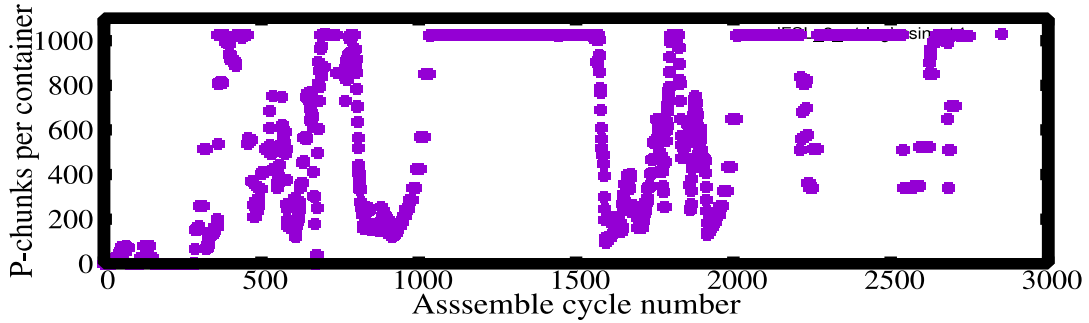
EMC_2

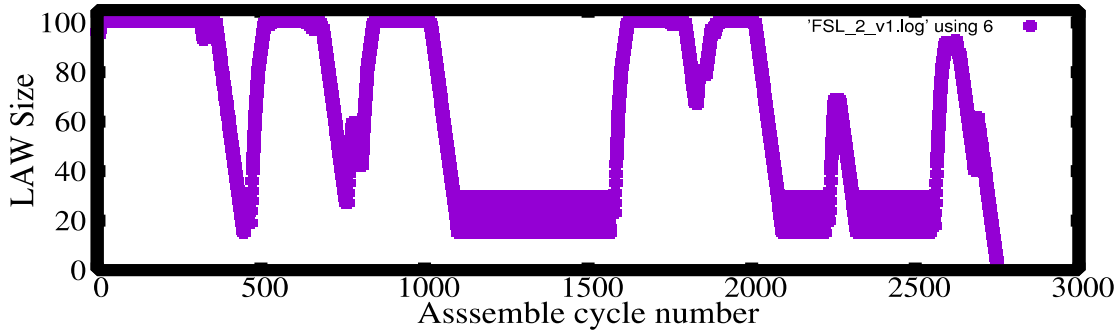

Container_LRU
Chunk_LRU
FAA
Fix_Opt
ALACC
Restore Throughput (MB/S)
60
50
40
30
20
10
0
0
1
2
3
4
5
Version Number

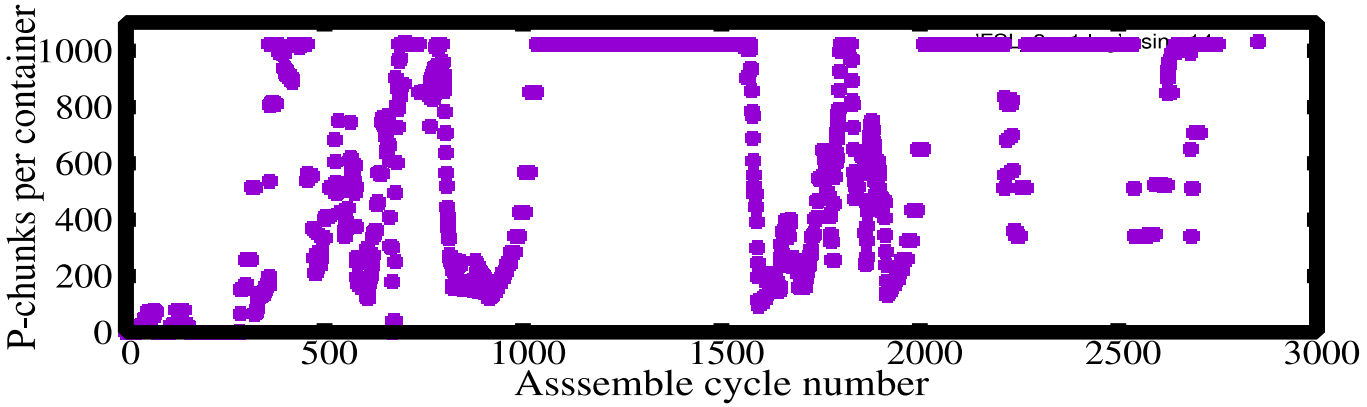
EMC_1

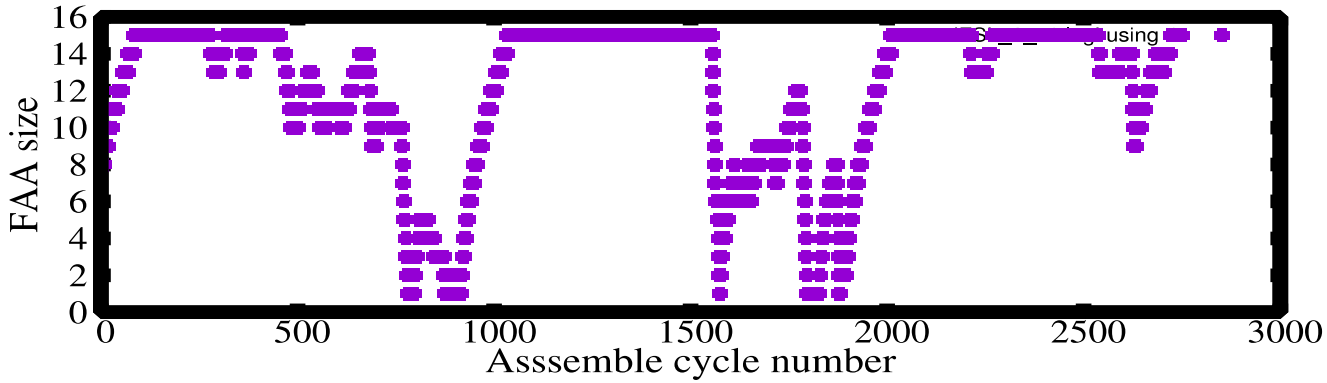

Container_LRU
Chunk_LRU
FAA
Fix_Opt
ALACC
Restore Throughput(MB/S)
105
90
75
60
45
30
15
0
0
1
2
3
4
5
Version Number

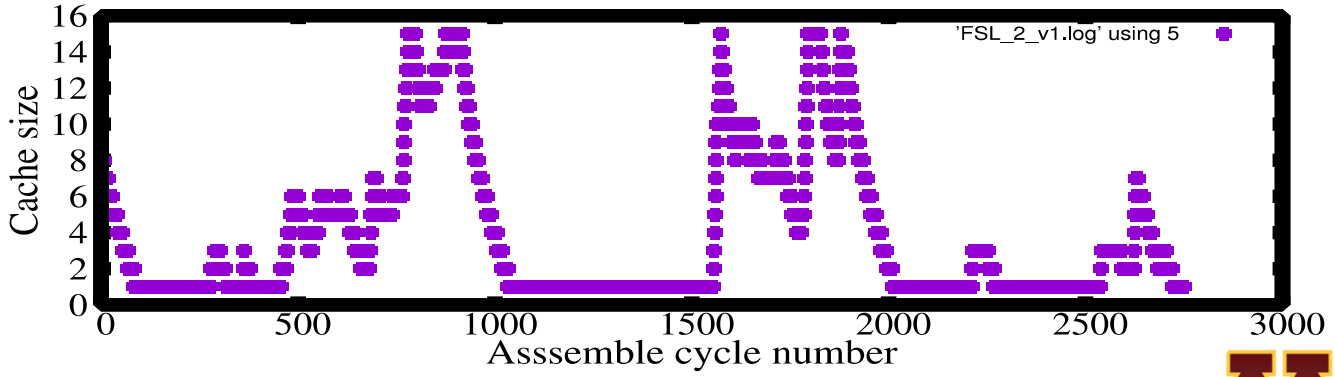
EMC_2


P-chunks per container
0
200
400
600
800
1000
0
500
1000
1500
2000
2500
3000
Asssemble cycle number


LAW Size
'FSL_2_v1.log' using 6
0
20
40
60
80
100
0
500
1000
1500
2000
2500
3000
Asssemble cycle number


P-chunks per container
0
200
400
600
800
1000
0
500
1000
1500
2000
2500
3000
Asssemble cycle number


FAA size
0
2
4
6
8
10
12
14
16
0
500
1000
1500
2000
2500
3000
Asssemble cycle number


Cache size
'FSL_2_v1.log' using 5
0
2
4
6
8
10
12
14
16
0
500
1000
1500
2000
2500
3000
Asssemble cycle number