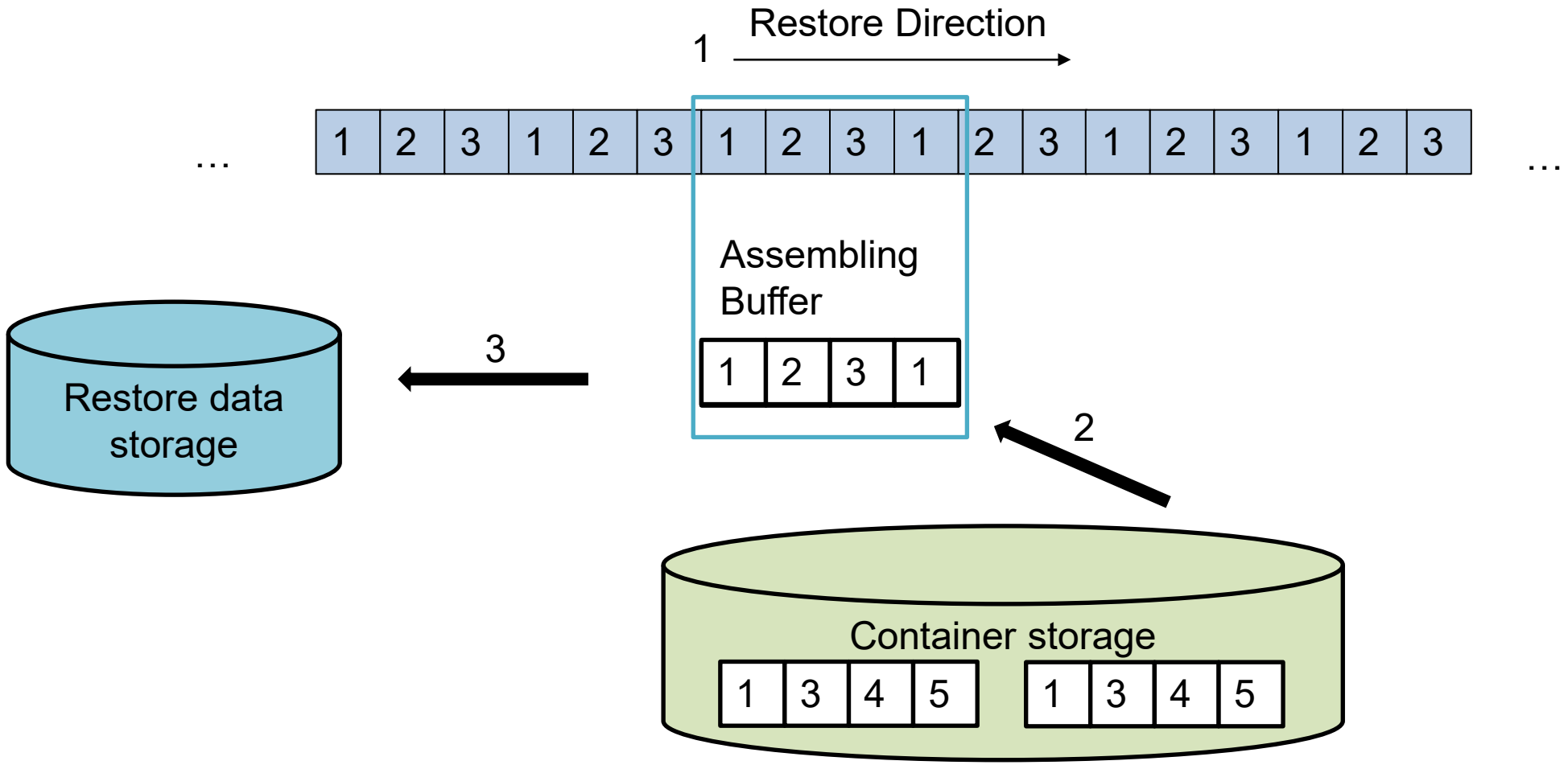
ALACC: Accelerating Restore Performance of Data Deduplication Systems Using Adaptive Look-Ahead Window Assisted Chunk Caching

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

Department of Computer Science, University of Minnesota, Twin Cities

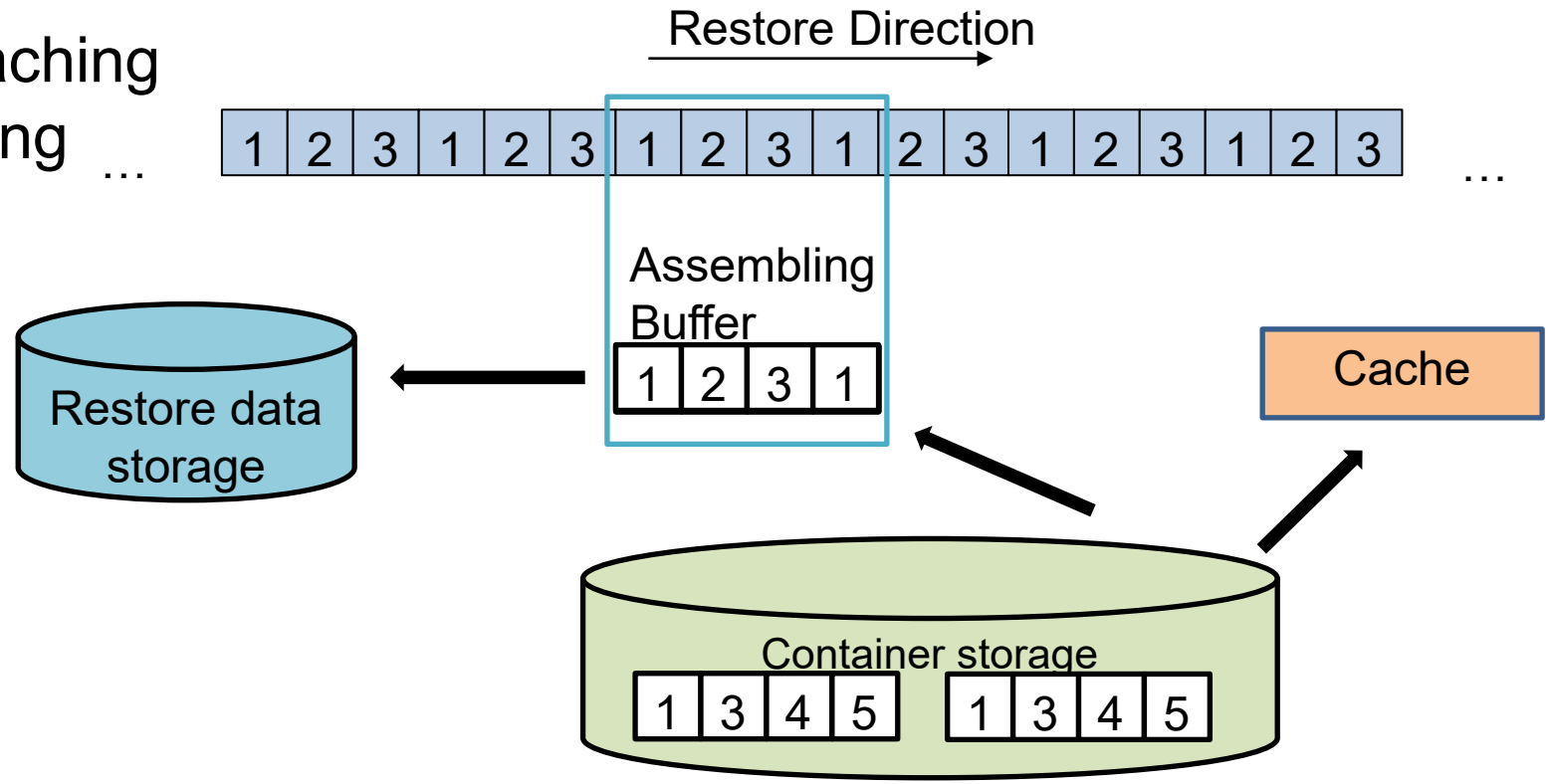
Background

- **Restore workflow**



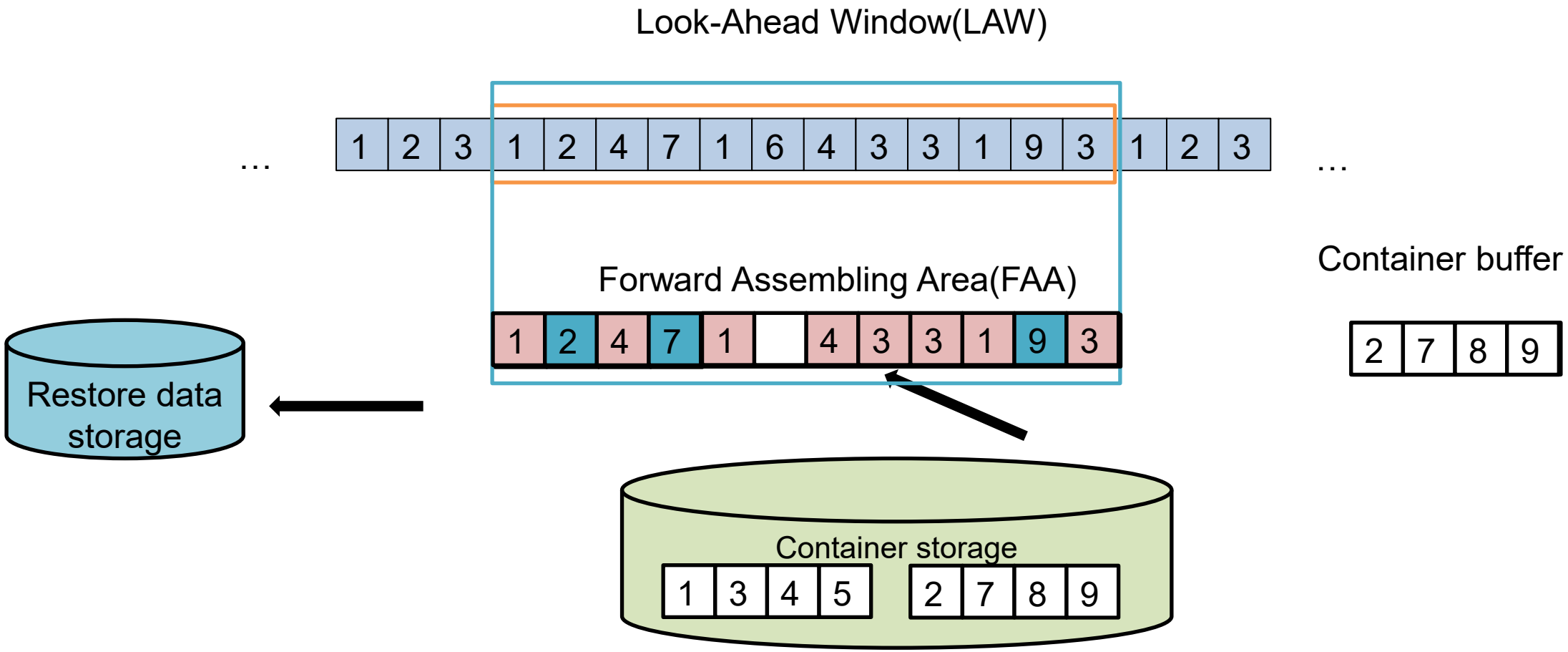
Background

- **Existing optimization methods**
  - **Add cache**
    - Container-based Caching
    - Chunk-based Caching
  - **Forward Assembly**

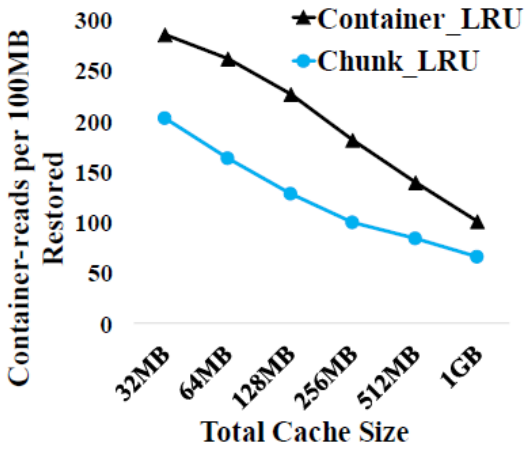


Background

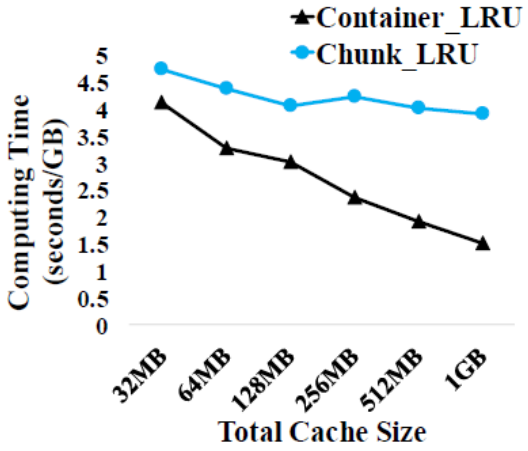
- Forward Assembly



Analysis of Cache Efficiency

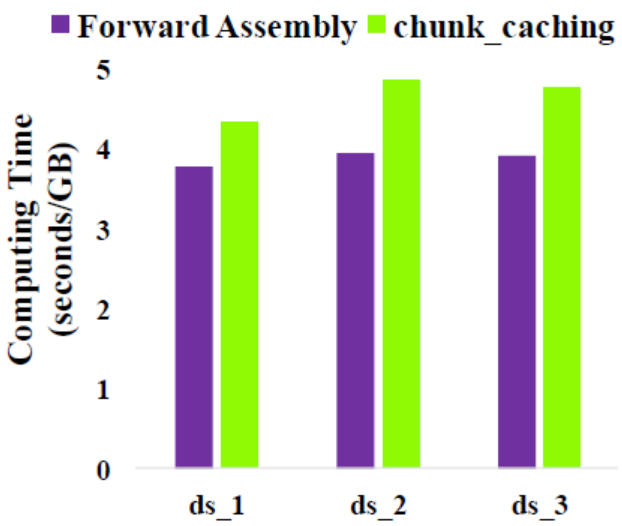


(a) # Containers-reads per 100MB restored as cache size varies from 32MB to 1GB

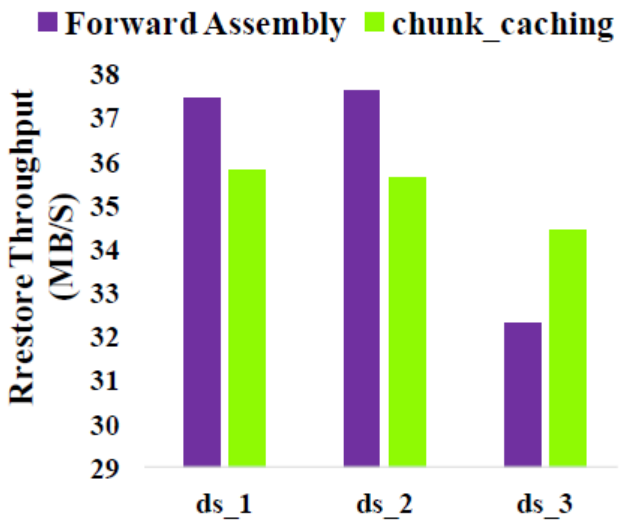


(b) Computing time per 1GB restored as cache size varies from 32MB to 1GB

Container-based caching vs. Chunk-based caching



(a) Computing time per 1GB restored



(b) Restore throughput

Forwarding Assembly vs. Chunk-based caching

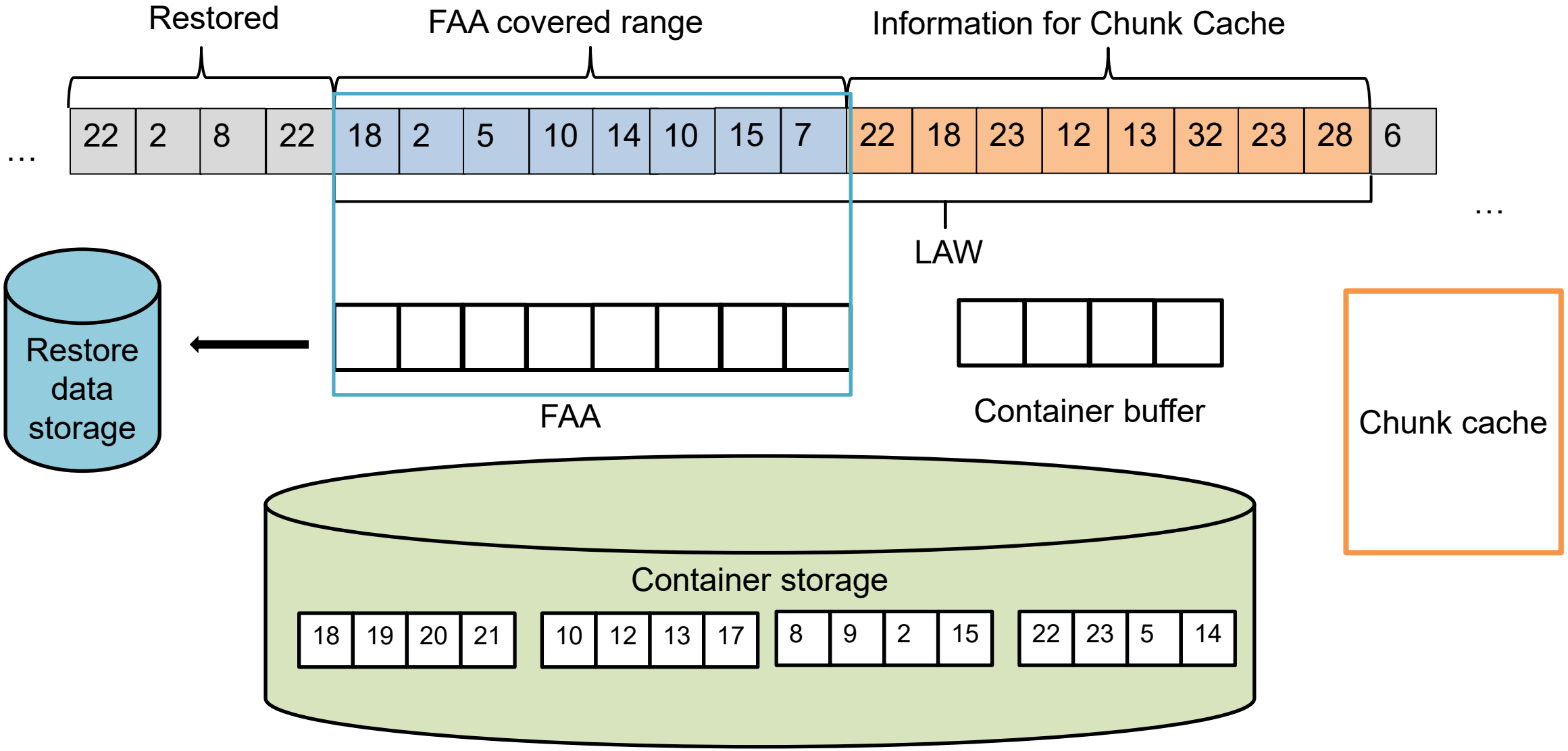
Problems

Scheme	**Container-based Caching**	**Chunk-based Caching**	**Forward Assembly**
Advantage	Lower operating and management overhead	**Higher cache hit ratio (higher if look-ahead window is applied)**	**Low operating and management overhead**
disadvantage	Cache space wasted, Relatively higher cache miss ratio	**Higher operating and management overhead**	**Workload sensitive**

Contribution

- **ALACC(Adaptive Look-Ahead Window Assisted Chunk Caching)**
  - Combine **chunk-based caching** and **forward assembly** scheme
  - Dynamically adjusted look-ahead window and cache size according to the workload

Architecture

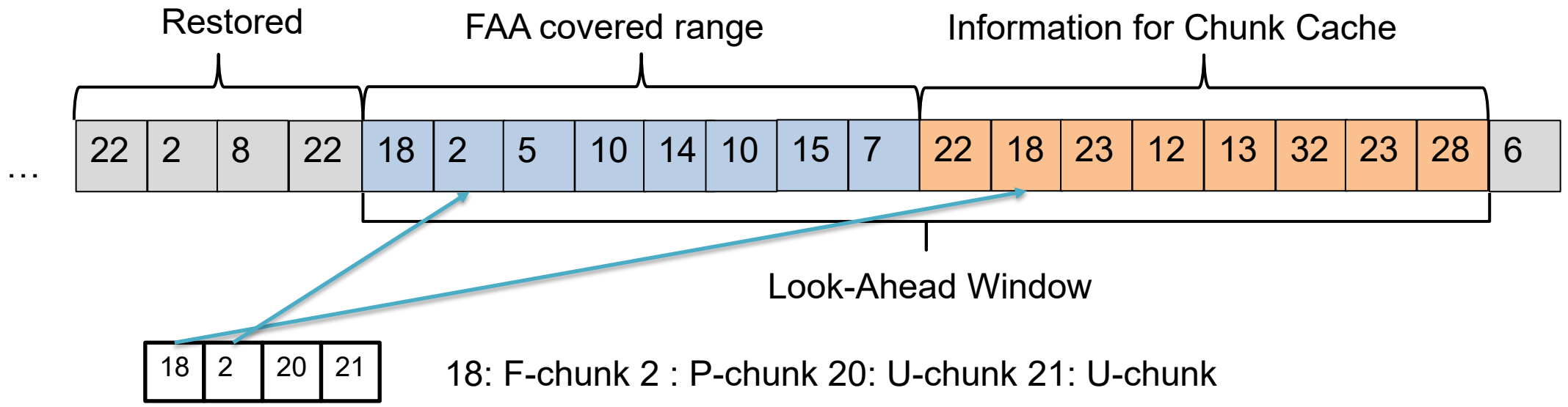


Workflow

- If current chunk has been stored by previous assembling operation, move pointer to next chunk;
- Else, check the **chunk cache**, if this chunk exists in cache, use this chunk to conduct assembling operation;
- Else, load container from storage to **container buffer**;
- Conduct assembling operation with all chunks in container buffer;
- Update chunk cache.

Caching policy

- **Chunk classification**
  - **U-chunk** (unused chunk): does not appear
  - **F-chunk** (Future chunk): Will appear in the future
  - **P-chunk** (Probably used chunk): appears in the current FAA but won't appear in the future



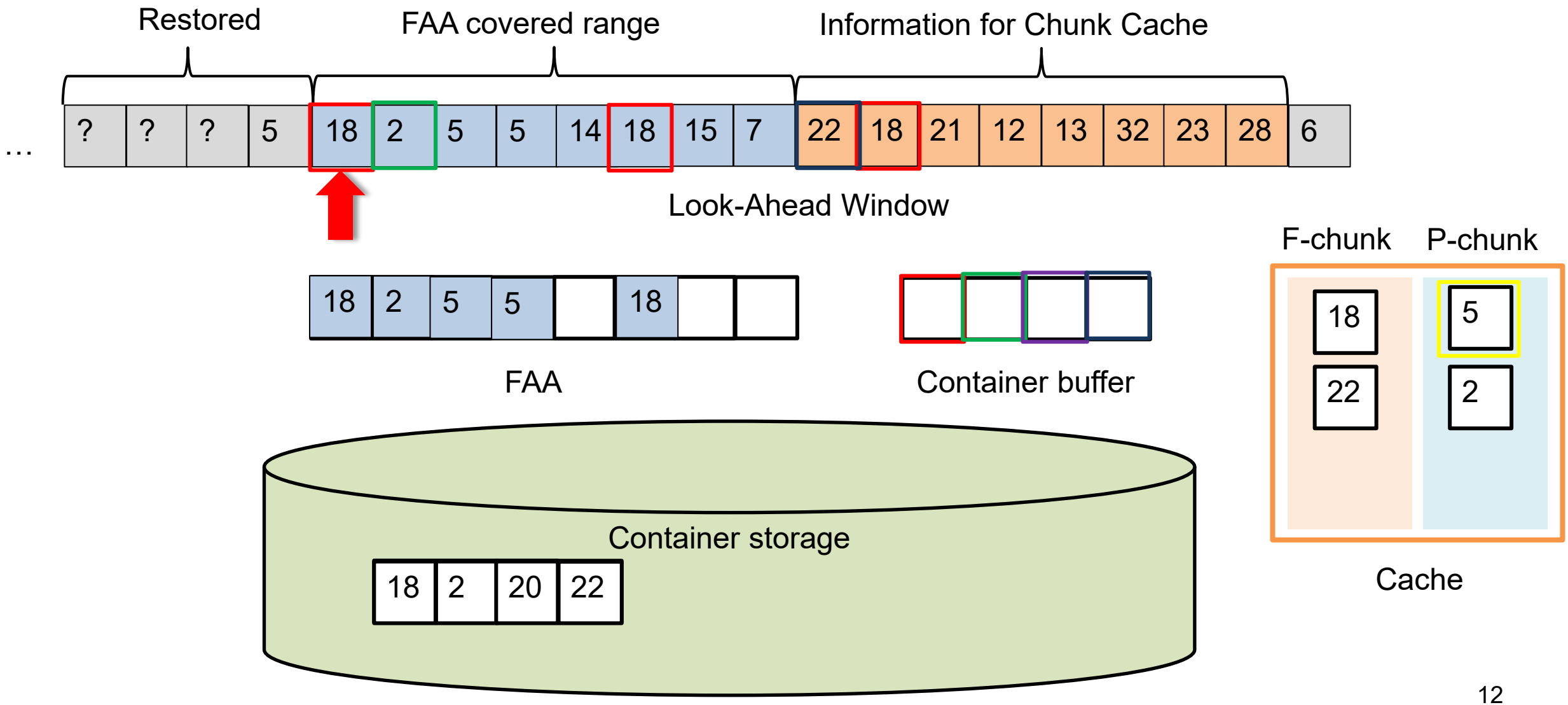
Caching policy

- **Priority**
  - F-chunk has the highest cache priority
  - If the cache space is sufficient, the P-chunk should also be cached
  - U chunk does not need to be cached
- **Extra thinking**

OPT: the theoretically best algorithm → F-chunk

LRU: the most common caching algorithm → P-chunk

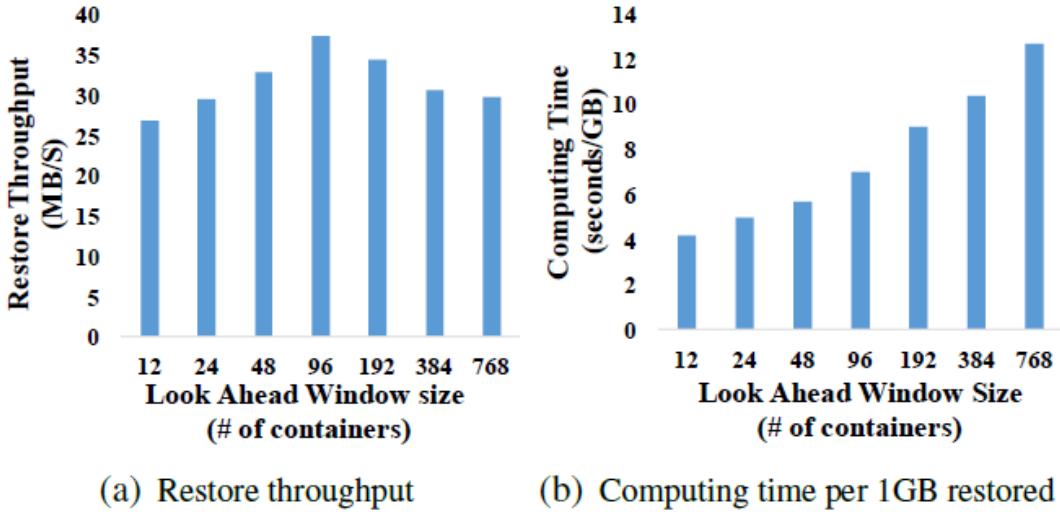
Example


Restored
FAA covered range
Information for Chunk Cache
…
? ? ? 5 18 2 5 5 14 18 15 7 22 18 21 12 13 32 23 28 6
Look-Ahead Window
18 2 5 5 18
FAA
Container buffer
F-chunk
P-chunk
18
22
5
2
Cache
Container storage
18 2 20 22

Adaptive algorithm

- **LAW size is too large**
  - The number of chunks looking forward is too small
  - In the extreme case(length(LAW) = length(FAA)), it degenerates into a complete LRU
- **LAW size is too small**
  - Part of the F-chunks cannot be put into the cache when the LAW is large enough
  - More CPU and memory overhead

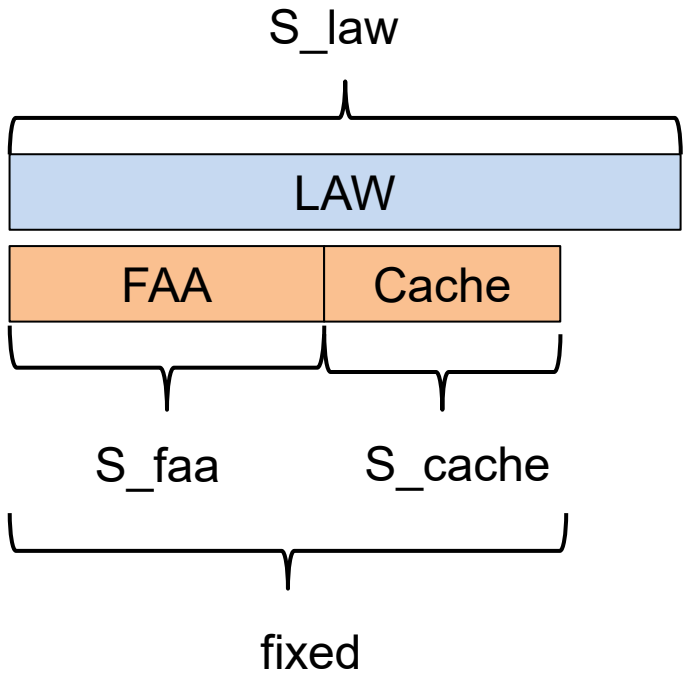
Fixed the size of FAA and chunk cache
Change LAW size



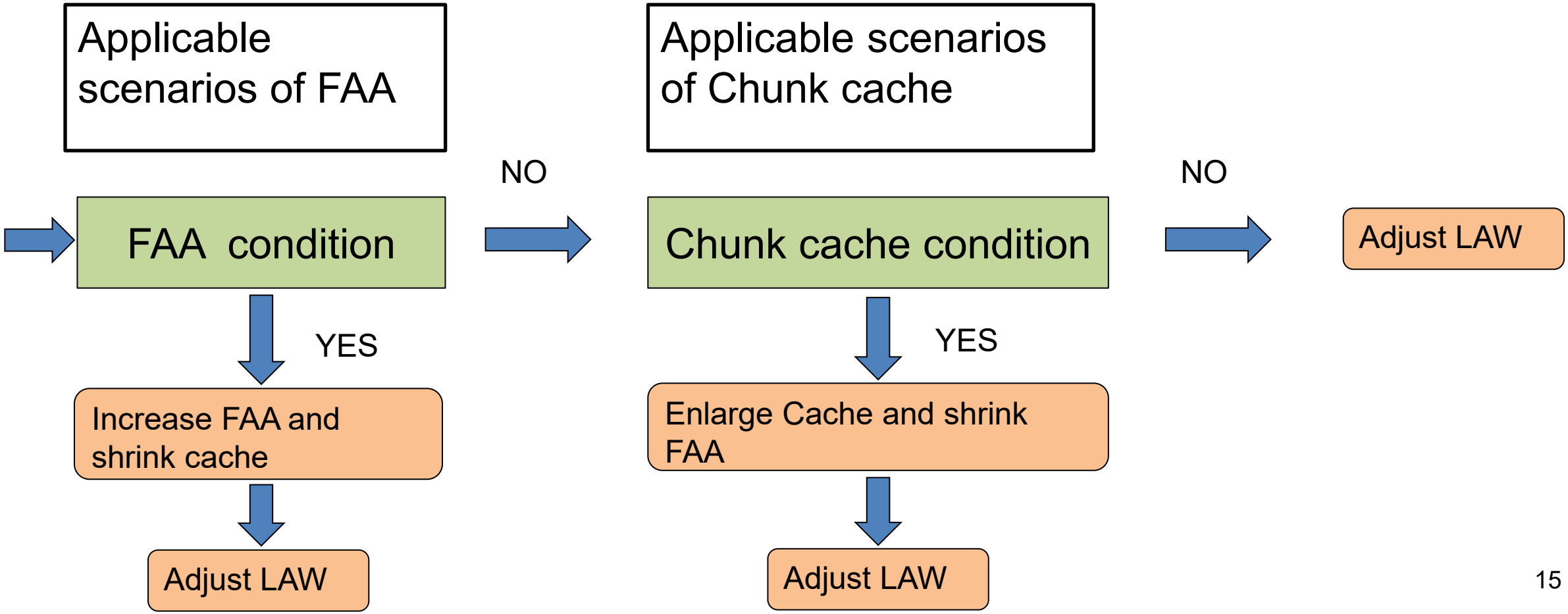
(a) Restore throughput (b) Computing time per 1GB restored

ALACC

- **Size information**
  - Size of LAW is variable but has a max value
  - Size of FAA and cache are variable but their sum is fixed
- **Basic strategy**
  - Evaluate the effects of FAA and Cache according to the chunk sequence, and then adjust the proportion of these two parts
  - Finally adjust the size of LAW



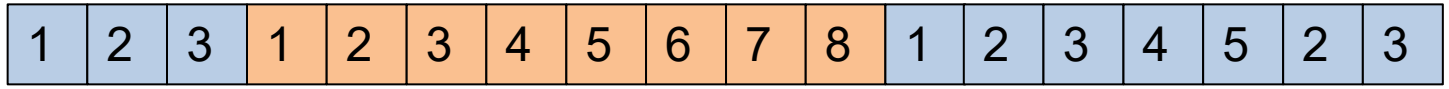
Adjustment strategy



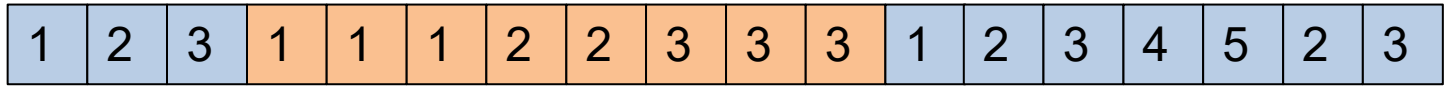
FAA size adjustment

➢ Applicable Conditions

- Re-use distances of most duplicated data chunks are within the FAA range



- Chunks in the first are identified mostly as unique data chunks



→ **Increase FAA size and decrease cache and LAW size**

Chunk cache and LAW size adjustment

➢ Chunk Conditions

- P-chunk number is very small (Space is occupied by F-chunk with higher priority)
- F-chunk added during the restore cycle is very large

→ **Increase cache size, decrease LAW size**

- F-chunk number is very small
  - LAW is too small(Insufficient ability to find F-chunk) -> **increase LAW size**

➢ LAW Independent adjustment

- Check threshold for F-chunk

Evaluation

- **Five Cache Designs**
  - LRU-based container caching(Container_LRU)
  - LRU-based chunking caching(Chunk_LRU)
  - Forward assembly(FAA)
  - Optimal fix configuration with fixed forward assembly and chunk-based caching (Fix_Opt)
  - ALACC
- **Dataset**
  - FSL
  - EMC

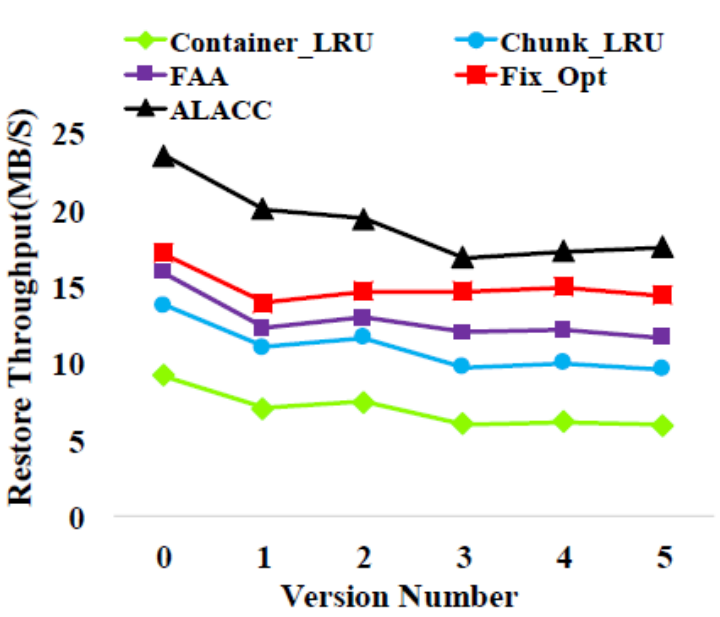
Dataset	FSL_1	FSL_2	EMC_1	EMC_2
Size	103.5GB	317.4GB	29.2GB	28.6GB
ACS[1]	4KB	4KB	8KB	8KB
DR[2]	3.82	4.88	1.04	4.8
CFL[3]	13.3	3.3	14.7	19.3

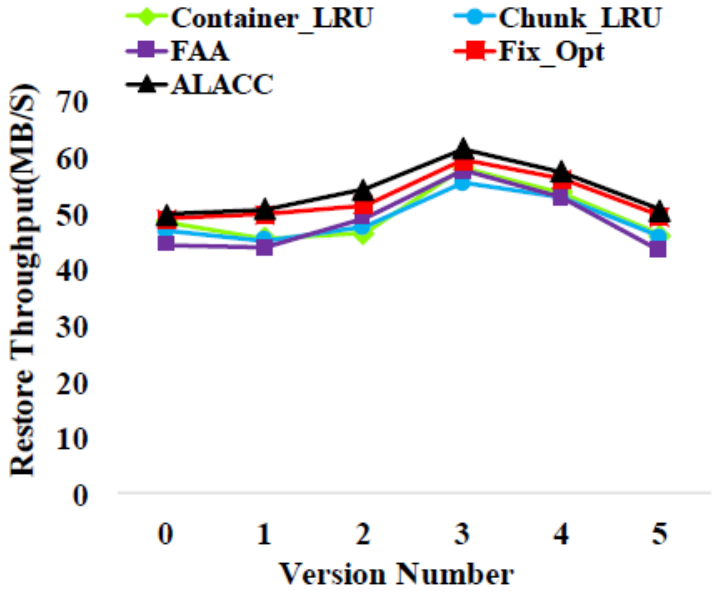
ACS: average chunk size
DR: Deduplication Ratio
CFL: Chunk Fragmentation Level(average container number which store one container size's data chunks of original data stream)

Evaluation

➢ Restore Throughput

Original data stream size divided by the total restore time.



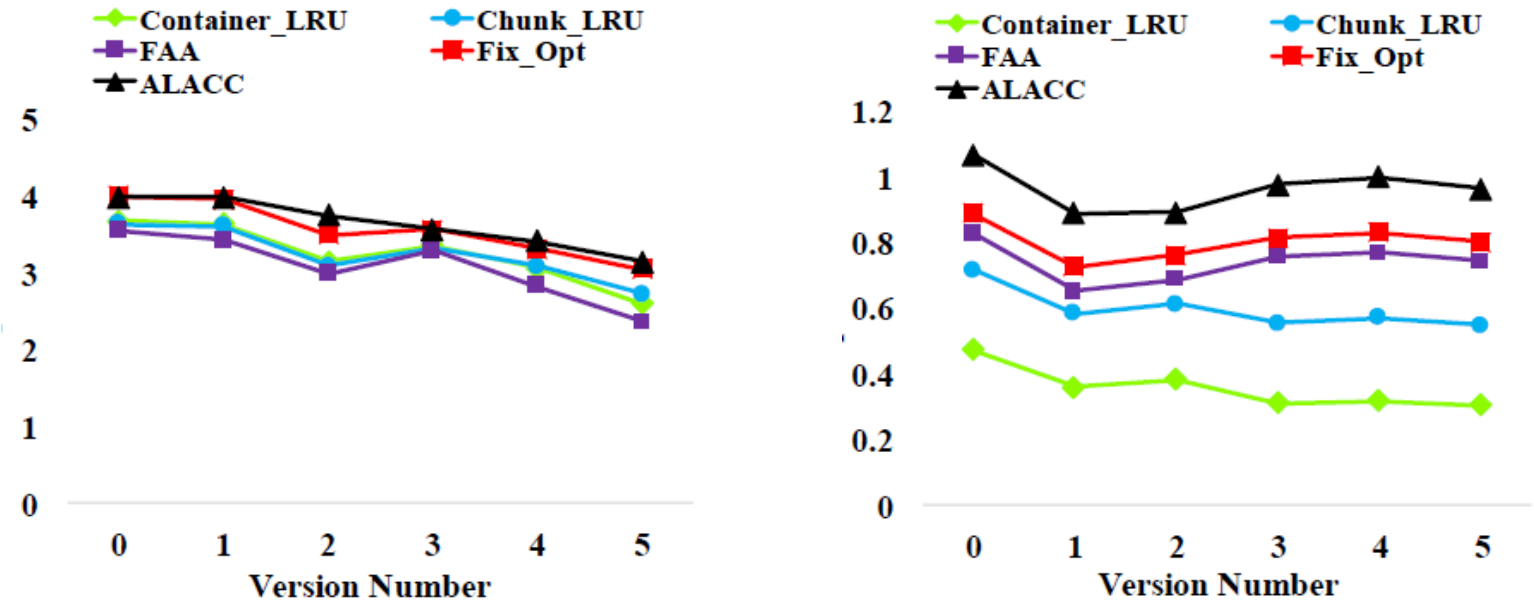


FSL_1(13.3 CFL) and FSL_2(3.3 CFL)

Evaluation

➢ Speed Factor

The mean data size restored per container read.

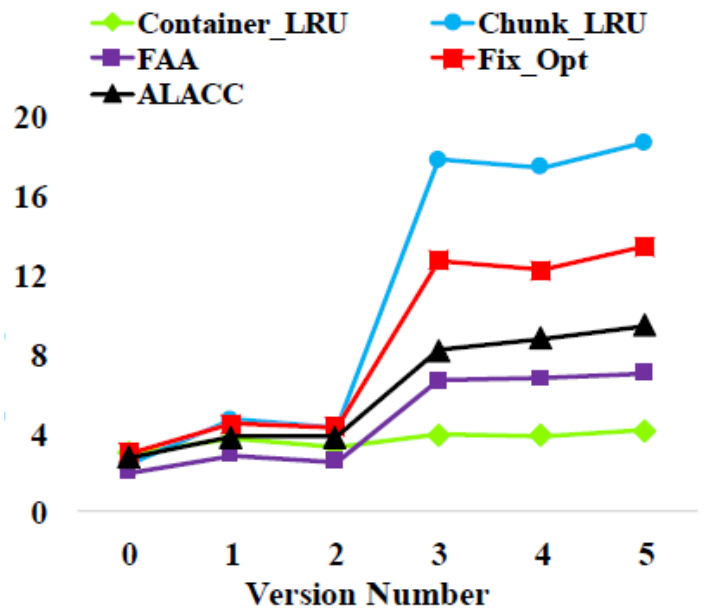


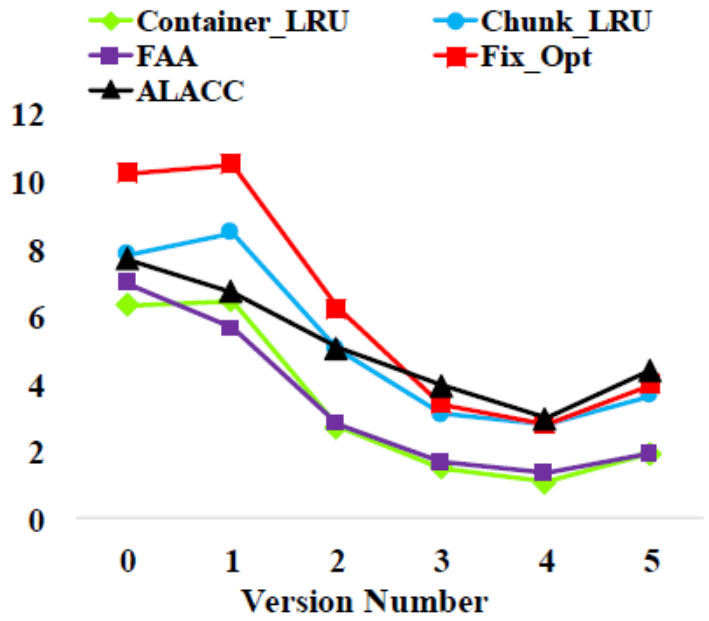
FSL_1(13.3 CFL) and FSL_2(3.3 CFL)

Evaluation

➢ Computing Cost Factor

Time spent on computing operations.





FSL_1(13.3 CFL) and FSL_2(3.3 CFL)

Conclusion

- Combine chunk-based caching and forward assembly scheme
- Dynamically adjusted look-ahead window and cache size according to the workload

Conclusion

- Combine chunk-based caching and forward assembly scheme
- Dynamically adjusted look-ahead window and cache size according to the workload
- Problem
  1. 讲的太紧张
  2. 鼠标晃慢一点
  3. 慢+清楚
  4. 列关键点
  5. 逻辑链不够清楚