Building a High-performance Fine-grained Deduplication Framework for Backup Storage with High Deduplication Ratio

Xiangyu Zou and Wen Xia, Harbin Institute of Technology, Shenzhen; Philip Shilane, Dell Technologies; Haijun Zhang and Xuan Wang, Harbin Institute of Technology, Shenzhen

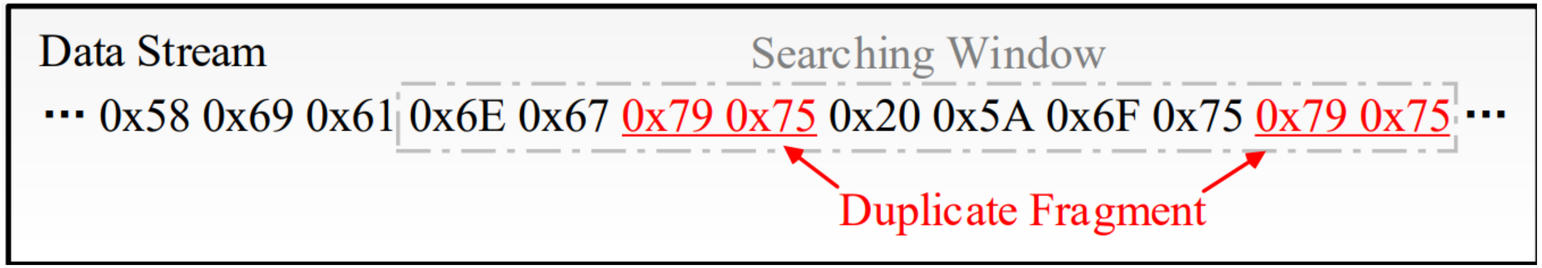
Speaker wrl

ATC 2022

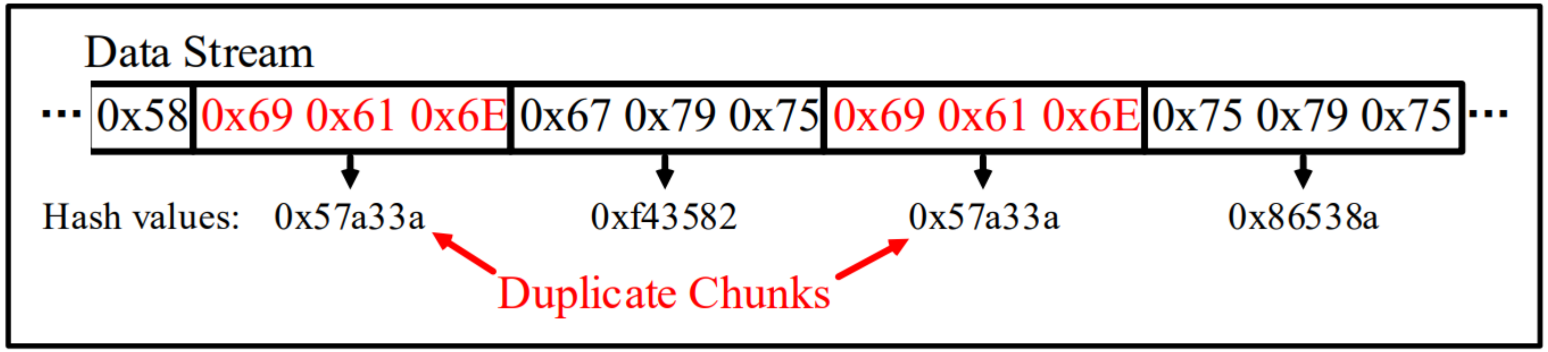
Background

- **Data Reduction:** the purpose is to reduce the physical capacity required to store the growing amounts of logical backup data.

- **General Compression :**
  - For usual-size files
  - String-level
  - Limited window



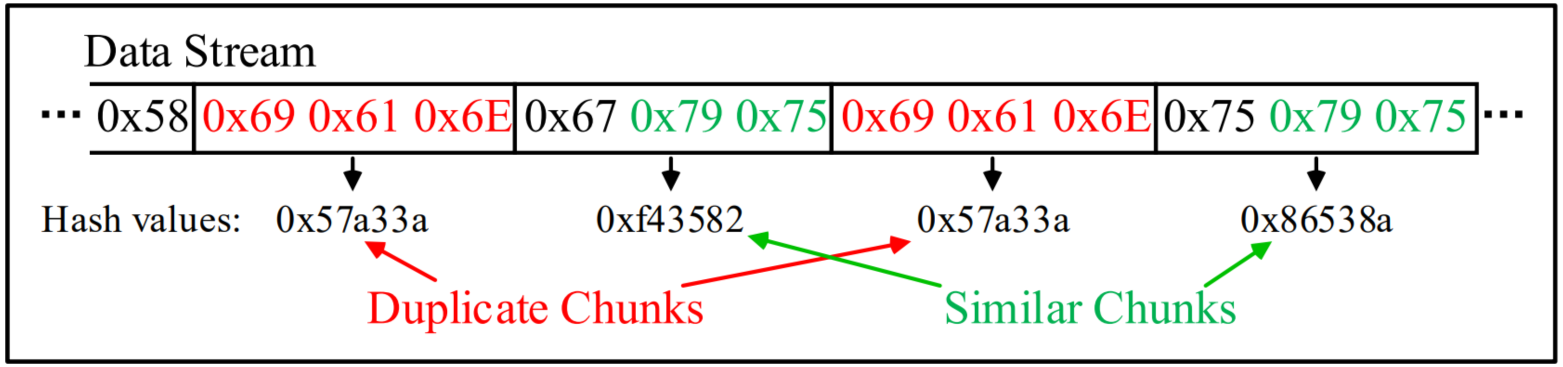
- **Deduplication:**
  - For very large files
  - Chunk-level
  - Global

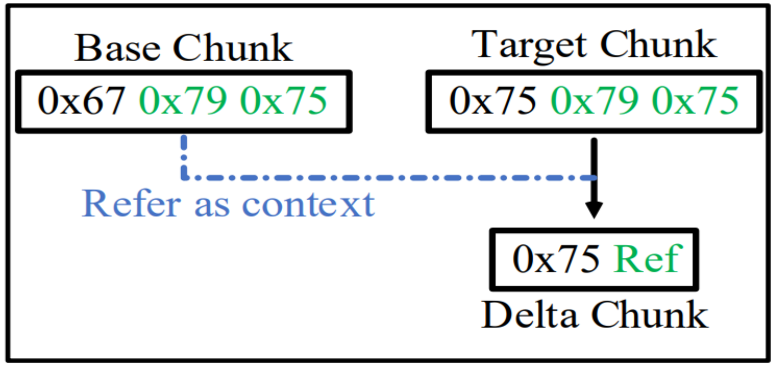


- Both have been widely used in storage products
- Can not fully utilize data's compressibility

Background

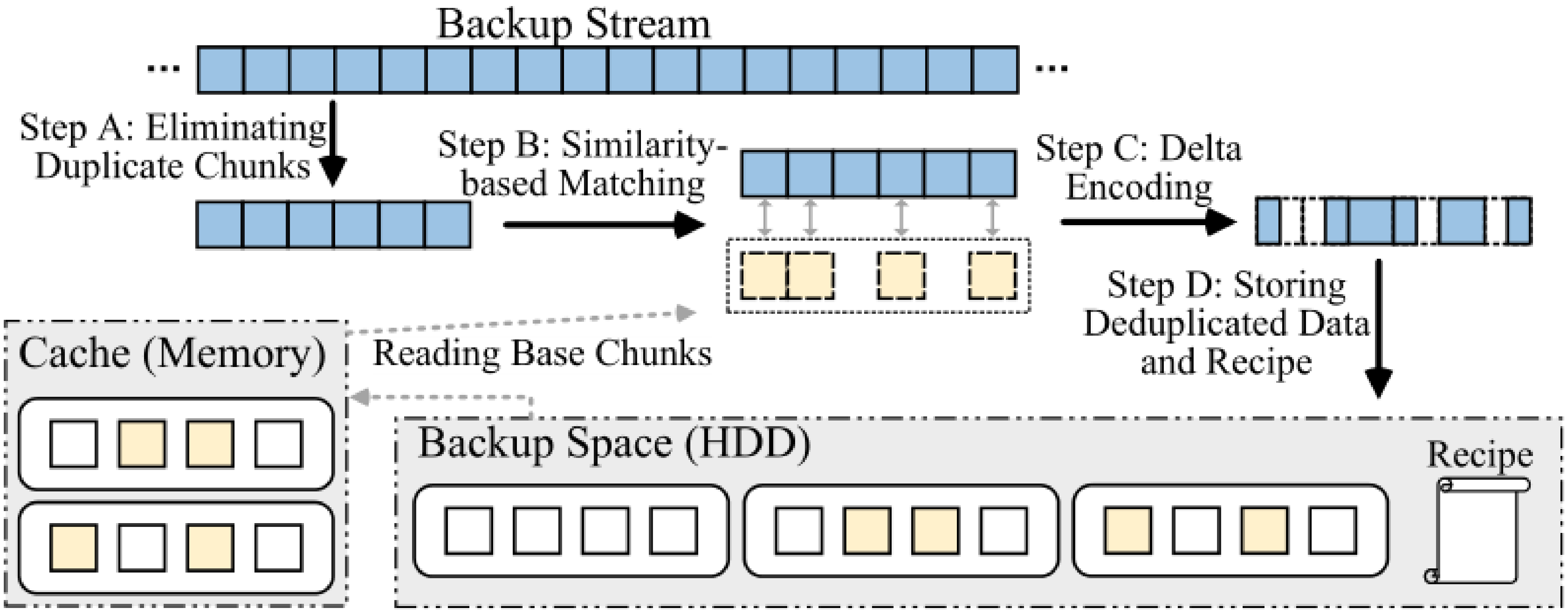
- **Fine-grained deduplication：**
  - Leverages not only identical chunks, but also similar chunks
  - Introduces additional steps on post-deduplication chunks
    - Detects similar chunks for an unduplicated chunk (i.e., target chunk for delta encoding)
    - Reads back a similar one (an already stored chunk) as a base chunk
    - Calculates delta difference between the target chunk and the base chunk
  - String-level, Global





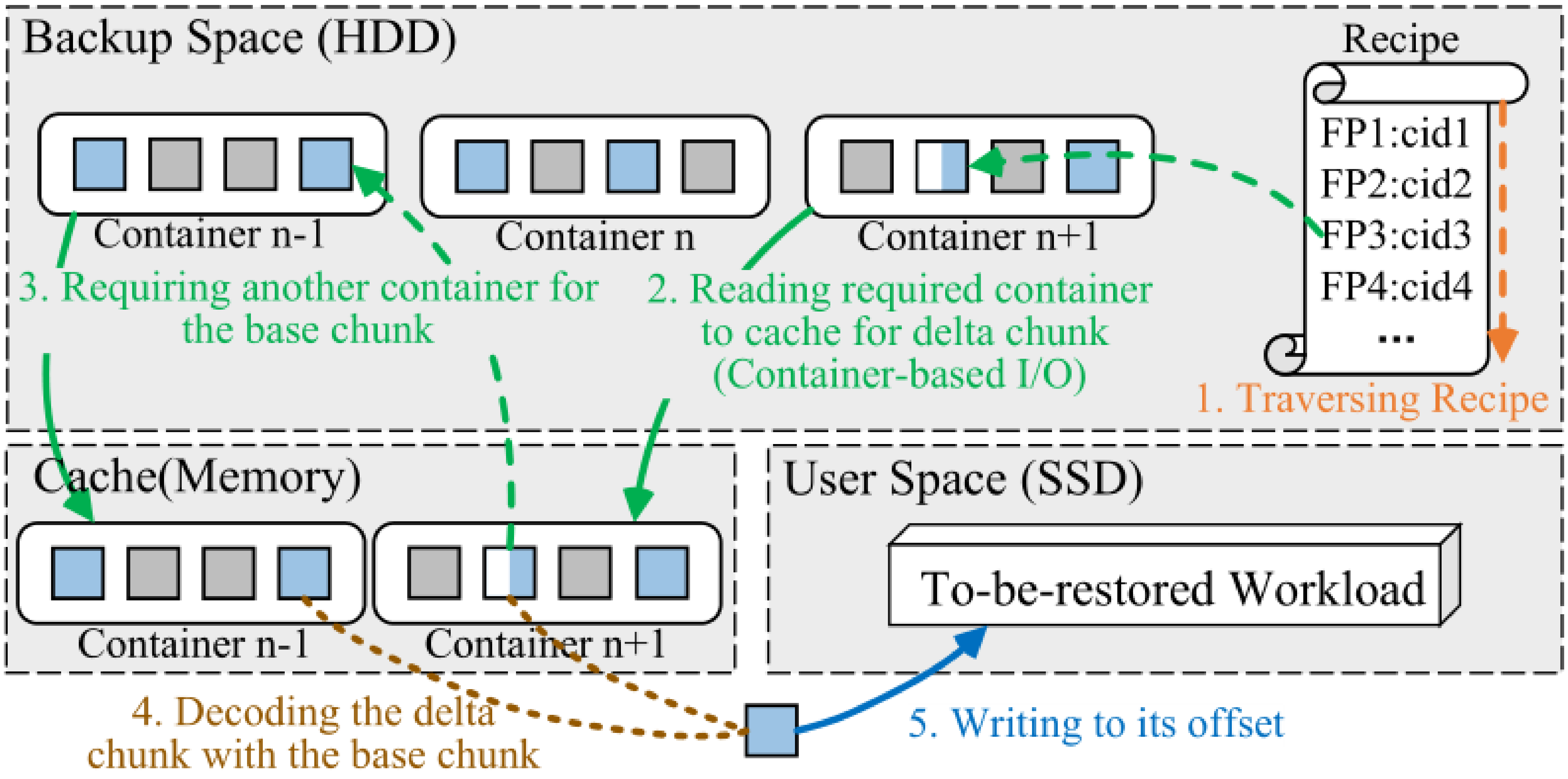
Background

➢**The backup workflow of fine-grained deduplication.**



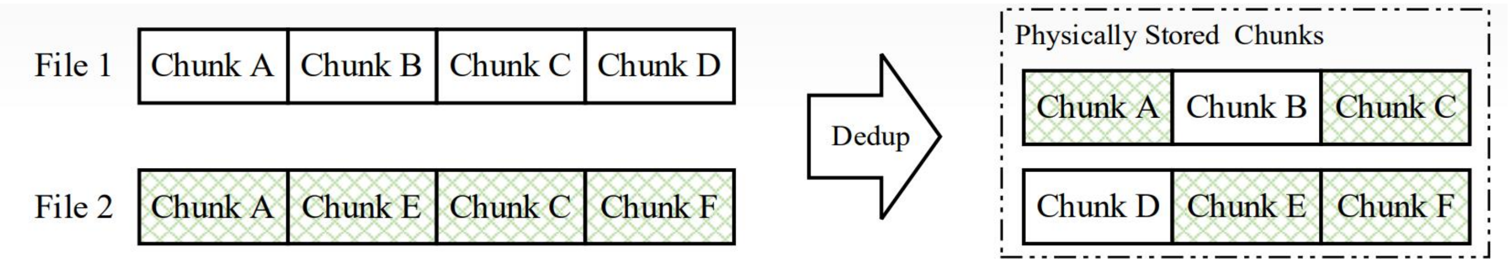
Background

- Restoring a delta chunk in the restore workflow of fine-grained deduplication.



Problem

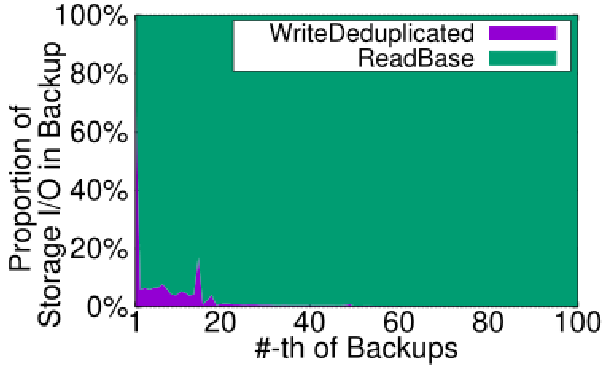
- **Dedupcation: Reusing data hurts locality ,declines systems' performance.**



- **The problem: Fine-grained deduplication introduces a new form of data reuse, make some additional locality issues**

  - **Challenge 1:** Poor locality in the backup workflow causes inefficient I/O when reading base chunks. **—Poor locality in base chunks (the write path)**
  - **Challenge 2:**Delta-base relationships lead to more complex fragmentation problems than deduplication alone.**—Poor locality in restore-required chunks (the read path)**
  - **Challenge 3:** Delta-base dependencies cause poor temporal locality during delta decoding and causes repeated container reads. **—Poor locality in delta-base pairs (the read path)**

Challenge <u>Poor locality in base chunks (the write path)</u>

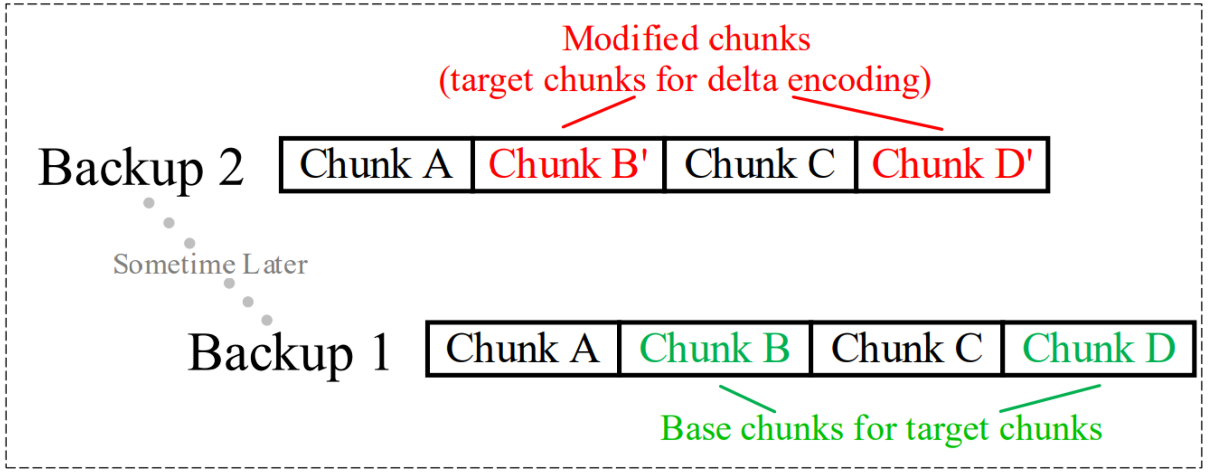


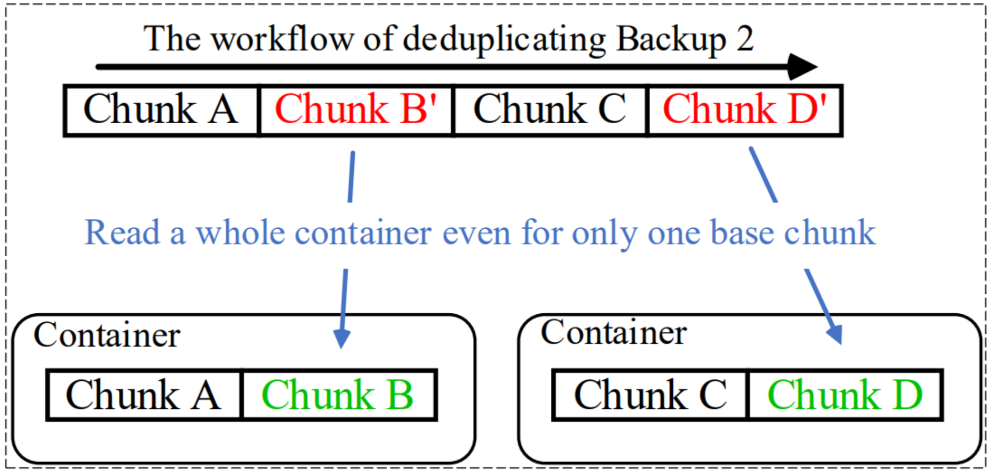
➢ Causes:

- The distribution of base chunks' physical positions is random
- Consecutive chunks are usually compressed together (local compression)
  - Accessing the whole compression unit (e.g., container) even for only one chunk

➢ Results:

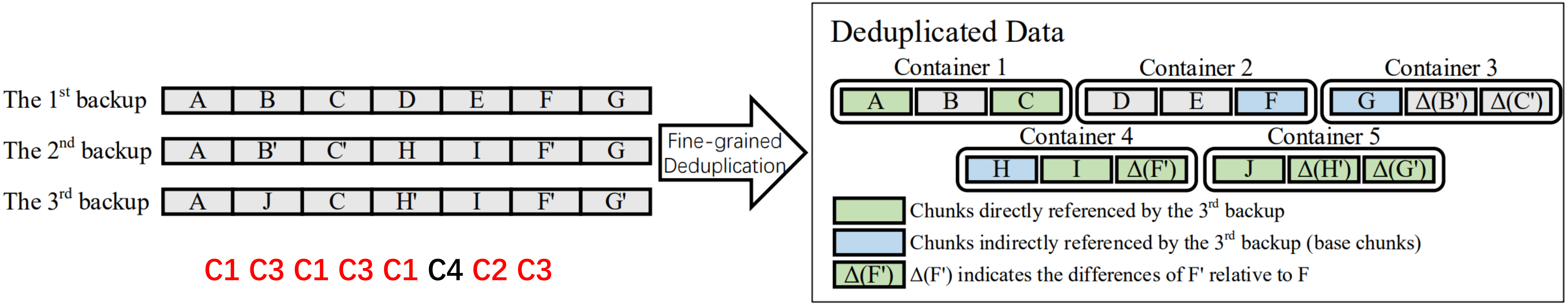
- Need to read a whole container even for only one base chunk
- Inefficient I/O when reading base chunks in the write path



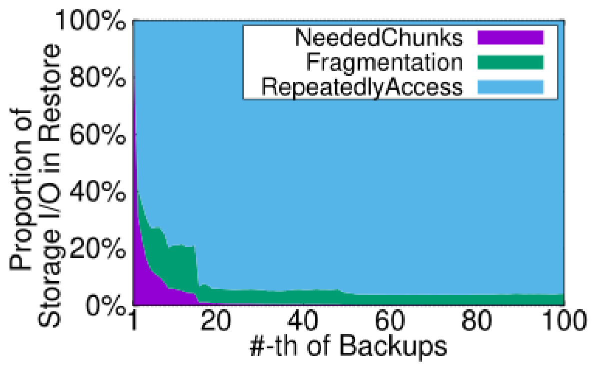


Challenge <u>Poor locality in restore-required chunks (the read path)</u>

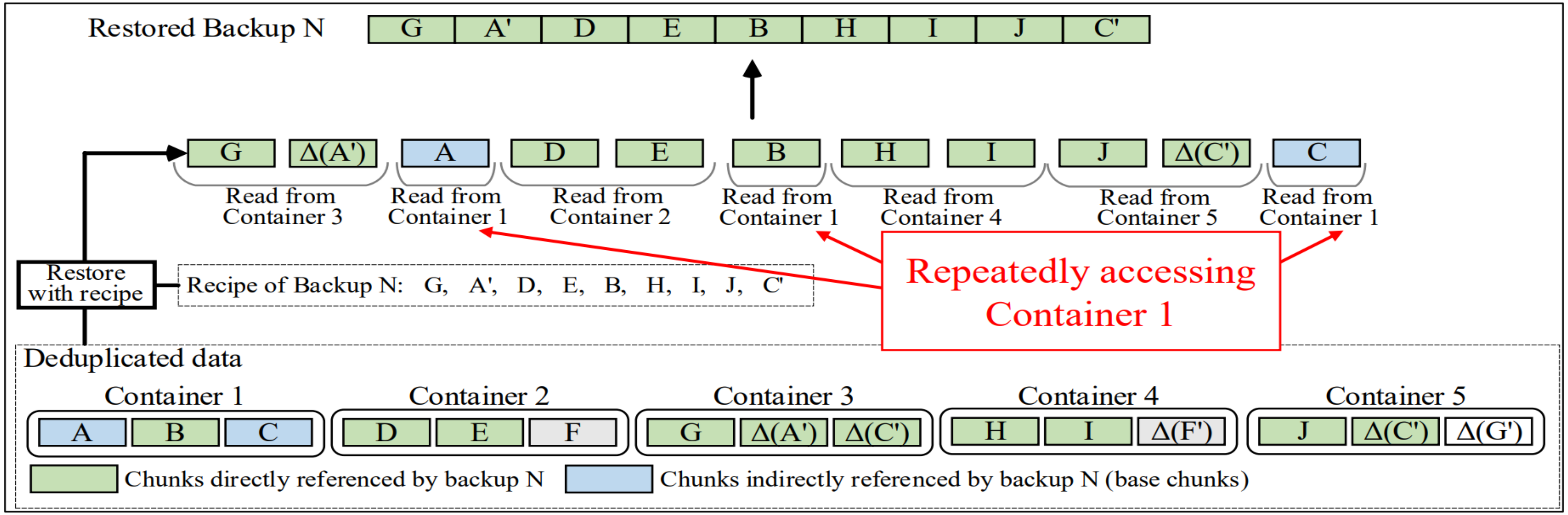
- **Causes:**
  - Two kinds of reference relationships
    - Backup workloads – Chunks (introduced by chunk-level deduplication)
    - Delta chunks – Base chunks (additionally introduced by delta encoding)
    - Aggravate the fragmentation problem
  - Local compression leads to a large I/O unit
- **Results:**
  - Inefficient I/O when reading restore-required chunks in the read path



Challenge Poor locality in delta-base pairs (the read path)

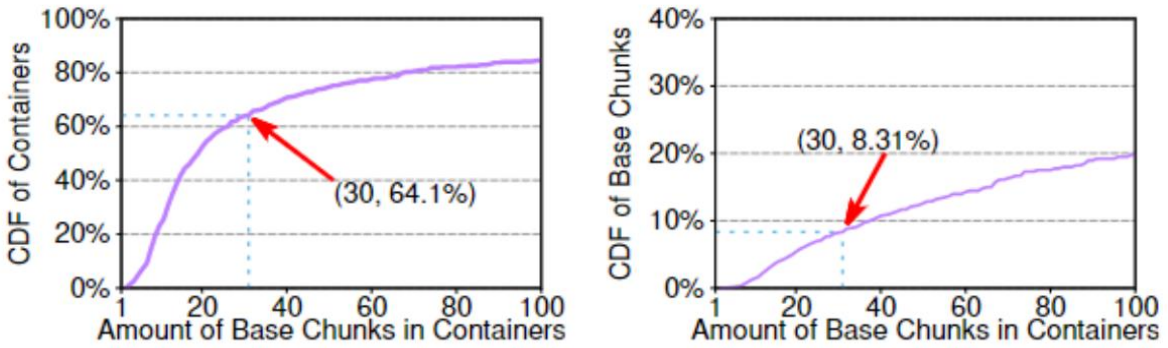


- **Causes:**
  - Traversing restore-required chunks when restoring a deduplicated backup
  - Delta chunks have dependencies, but usually are far away from their bases
- **Results:**
  - Repeatedly accessing containers in the read path



Design <u>Selective Delta Encoding</u>

- **Key Idea: Skip delta encoding if base chunks are in base-sparse containers**
  - An observation: Base chunks are not distributed evenly
  - For example, in an evaluated dataset:
    - 64.1% containers hold ~30 base chunks (“base-sparse containers”)
    - These 64.1% containers only includes 8.31% of the total base chunks.
  - Avoids reading these “inefficient” containers in the deduplication workflow

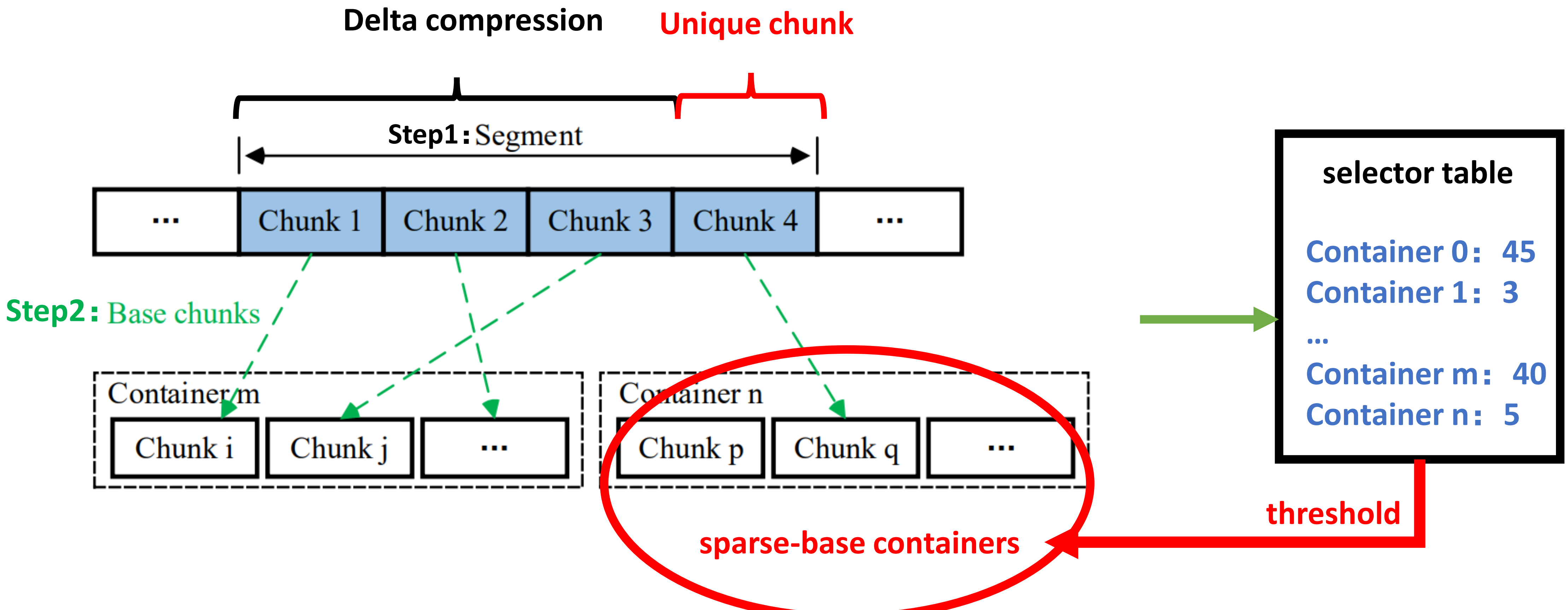


(a) 64.1% of containers contain only ~30 base chunks.

(b) These 64.1% containers only includes 8.31% of the total base chunks.

Design <u>Selective Delta Encoding</u>

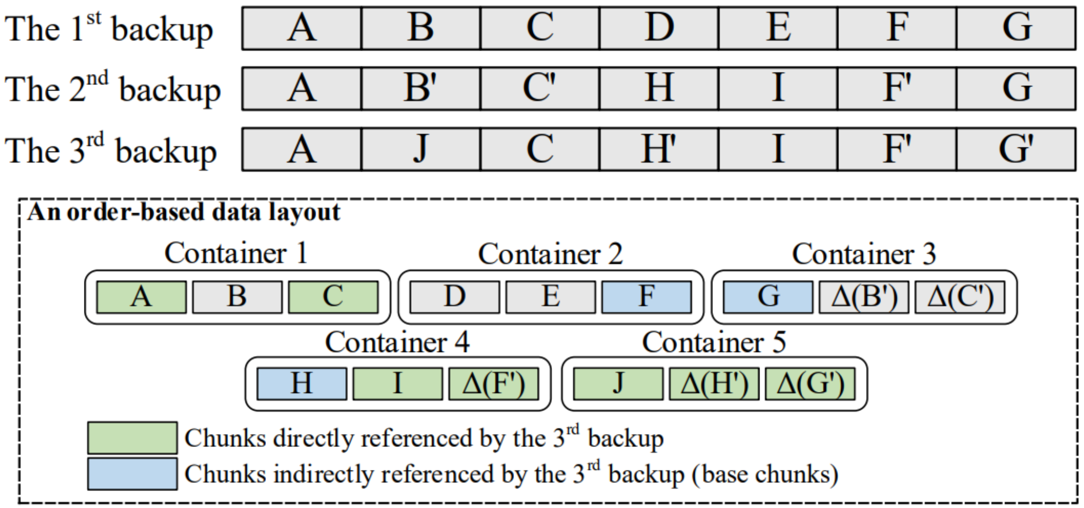
- **Key Idea: Skip delta encoding if base chunks are in base-sparse containers➞lower I/O and lower compress ratio.**



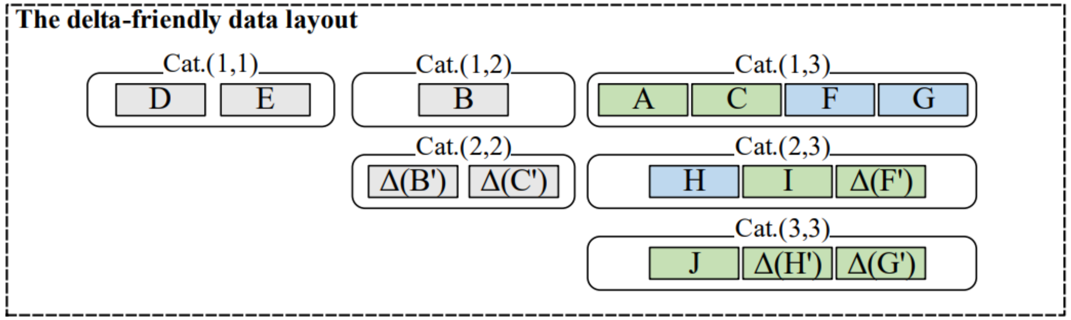
Design Delta-friendly Data Layout

➢ Key Idea: Put the chunks from the same backup together

- Consider two kinds of reference relationship
  - The “Necessary Chunks” of a backup
  - The combination of a backup’s directly and indirectly referenced chunks
- The lifecycle of a chunk
  - A set of backups whose "Necessary chunks" includes this chunks.
- Lifecycle-based classification
- Avoids reading sparse containers in the restore workflow

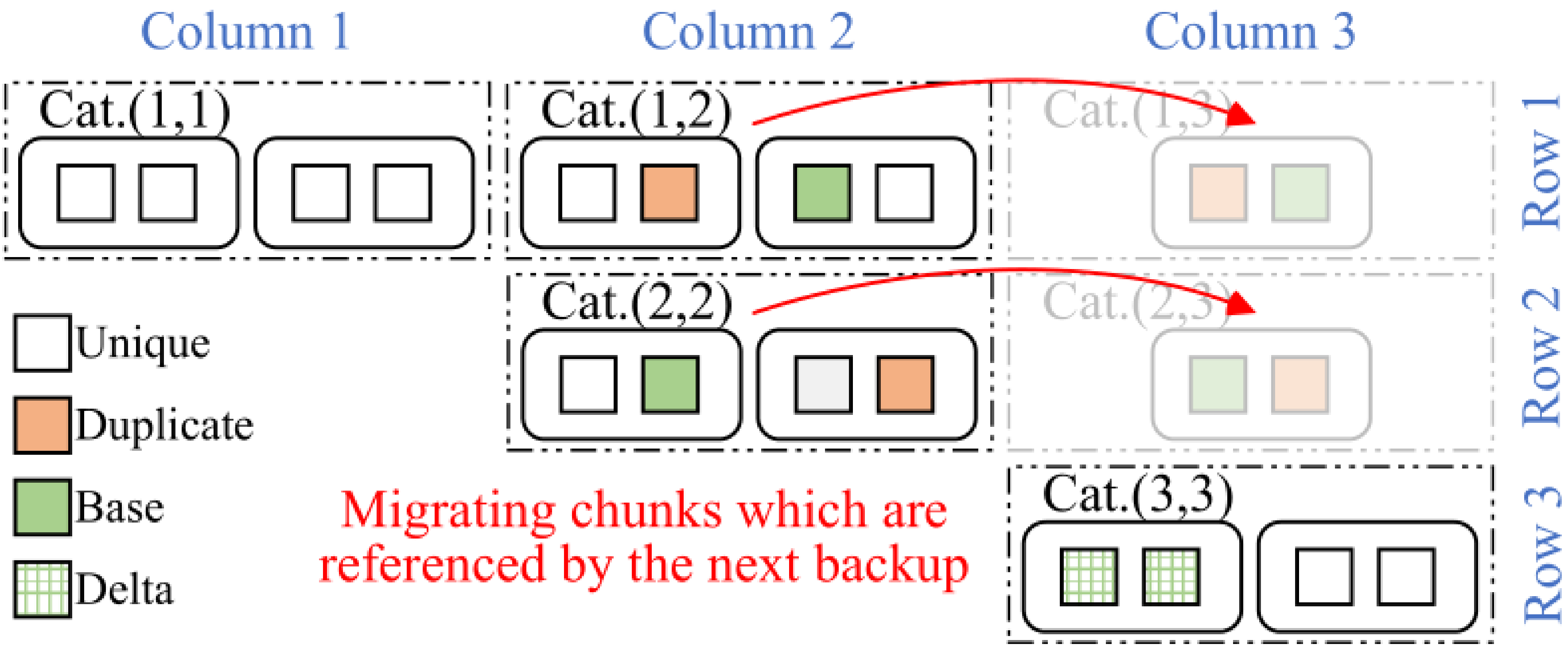


- NC_Backup1: A, B, C, D, E, F, G
- NC_Backup2: A, B, Δ(B’), C, Δ(C’), H, I, F, Δ(F’), G
- NC_Backup3: A, J, C, H, Δ(H’), I, F, Δ(F’), G, Δ(G’)



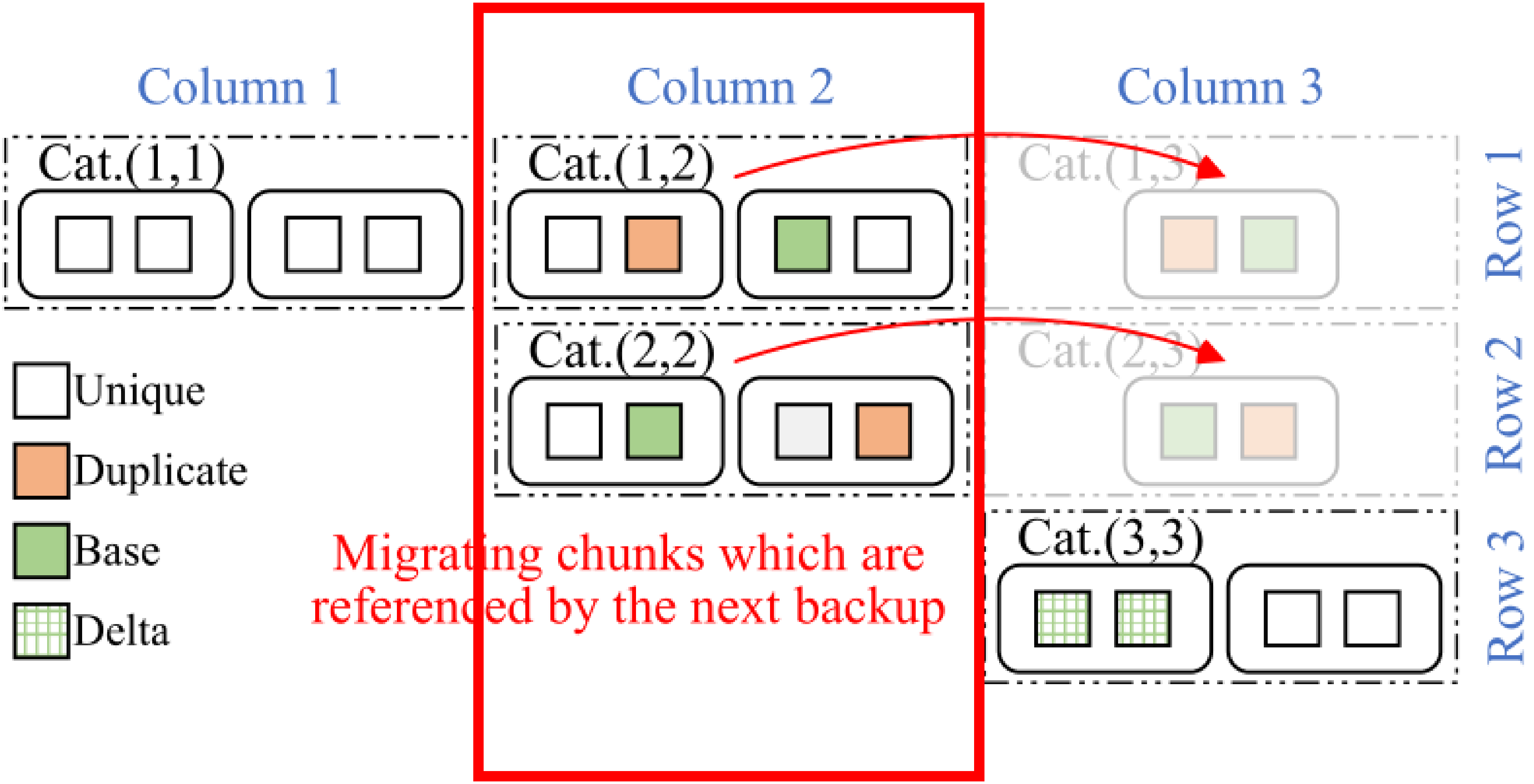
Design <u>Delta-friendly Data Layout</u>

- **Key Idea: Put the chunks from the same backup together**



Design Delta-friendly Data Layout

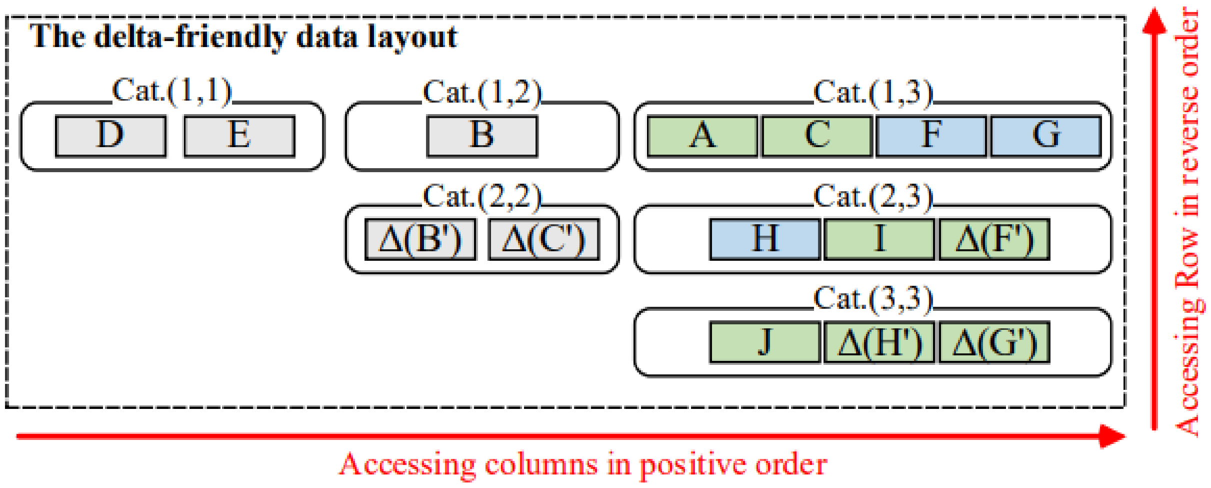
- **Key Idea: Put the chunks from the same backup together**

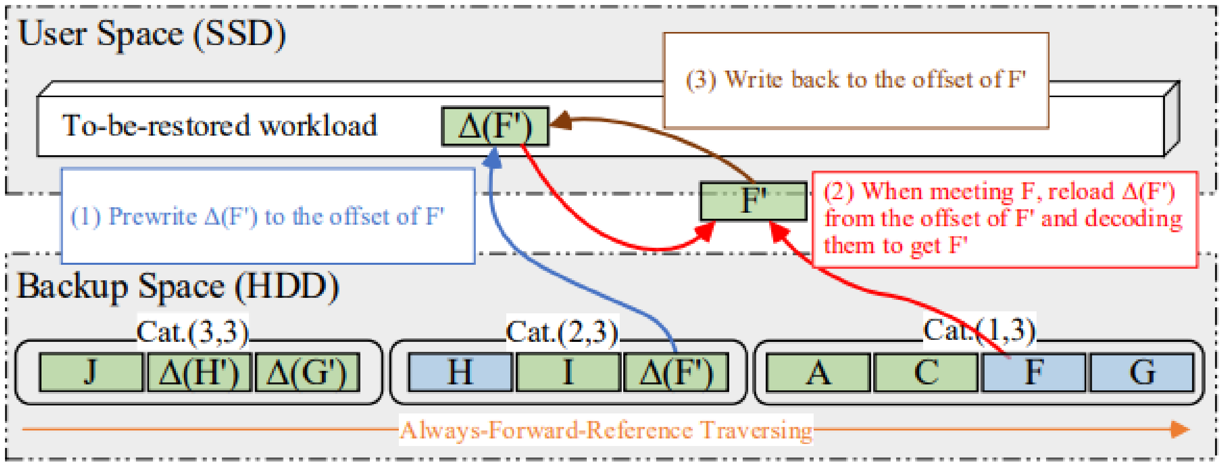


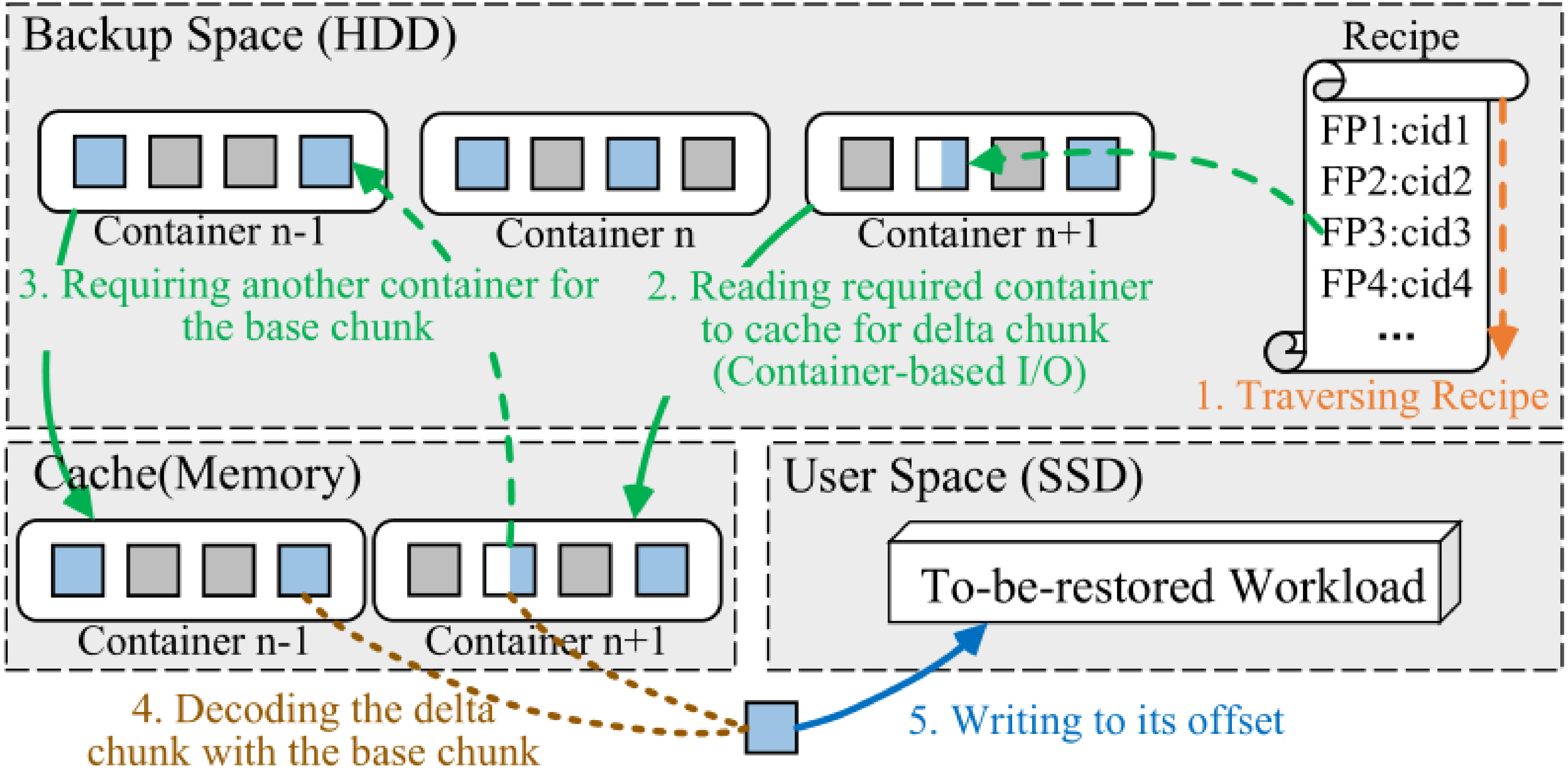
Design Always-Forward-Reference Traversing and Delta Prewriting

➢ Key Idea: Read the Container once and delta chunks before the base chunks

- A special path to traverse the restore-required chunks
  - Promises that delta chunks always appear before their base chunks
  - Rules to achieve AFR traversing
- Prewriting delta chunks
  - Asymmetric I/O characteristics of backup's/user's storage media
- Avoids repeatedly accessing restore-required chunks/containers







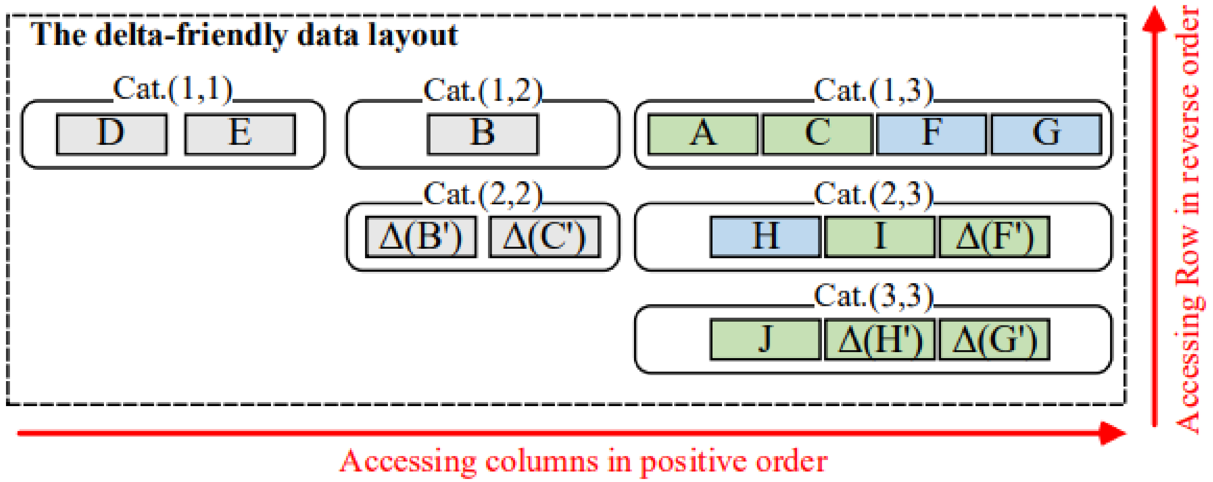
Backup Space (HDD)
Recipe
FP1:cid1
FP2:cid2
FP3:cid3
FP4:cid4
...
Container n-1
Container n
Container n+1
3. Requiring another container for the base chunk
2. Reading required container to cache for delta chunk (Container-based I/O)
1. Traversing Recipe
Cache(Memory)
User Space (SSD)
Container n-1
Container n+1
To-be-restored Workload
4. Decoding the delta chunk with the base chunk
5. Writing to its offset

Design Always-Forward-Reference Traversing and Delta Prewriting

➢ Key Idea: Read the Container once and delta chunks before the base chunks

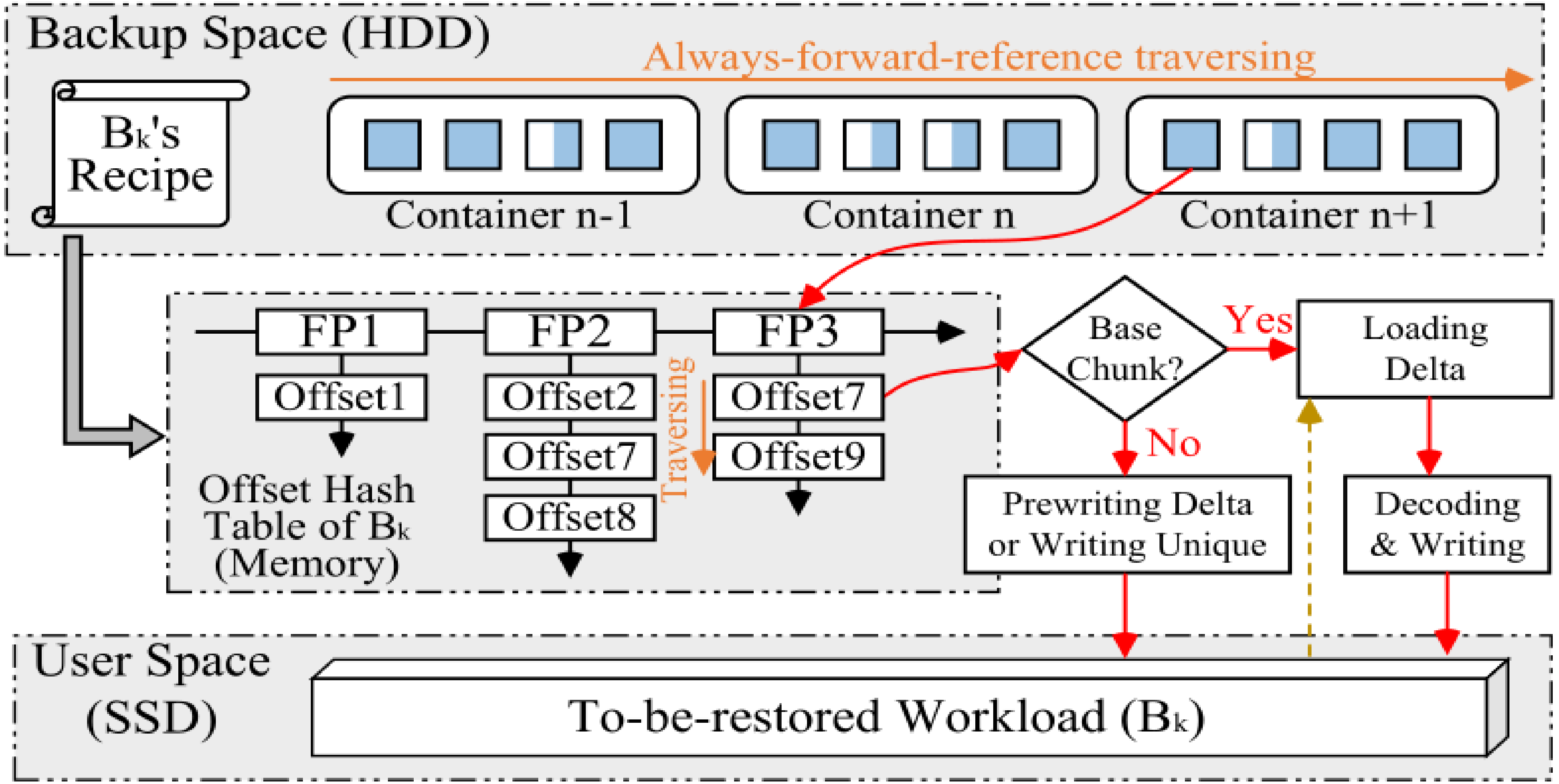
- A special path to traverse the restore-required chunks
  - Promises that delta chunks always appear before their base chunks
  - Rules to achieve AFR traversing
- Prewriting delta chunks
  - Asymmetric I/O characteristics of backup's/user's storage media
- Avoids repeatedly accessing restore-required chunks/containers

Delta Chunks' Positions		Corresponding Base Chunks' Possible Positions
Cat.(1,2)	⇒	Cat.(1,2), Cat.(1,3)
Cat.(2,2)	⇒	Cat.(1,2), Cat.(2,2), Cat.(1,3), Cat.(2,3)
Cat.(1,3)	⇒	Cat.(1,3)
Cat.(2,3)	⇒	Cat.(1,3), Cat.(2,3)



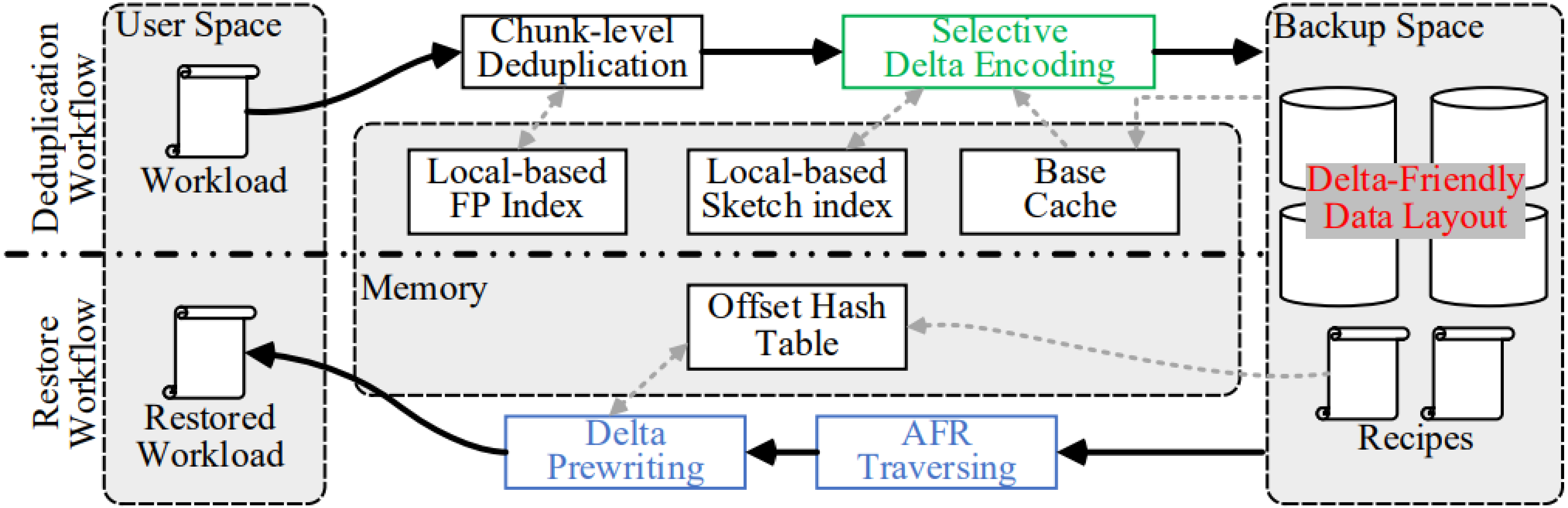
Design Always-Forward-Reference Traversing and Delta Prewriting

- **Key Idea: Read the delta chunks before the base chunks**



Architecture

- ➢ Techniques to address these three additional locality issues
  - Selective Delta Encoding
  - Delta-friendly Data Layout
  - Always-Forward-Reference Traversing and Delta Prewriting
- ➢ A fine-grained deduplication framework – MeGA



Evaluation <u>Experimental Setup</u>

➢ Evaluated approaches

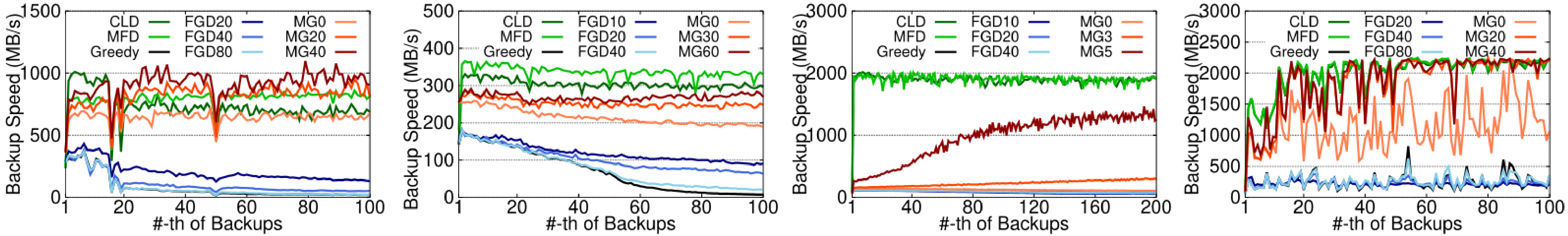
- **MeGA** Our proposed approach, using the three proposed techniques
- **Greedy** A fine-grained dedup approach with a greedy strategy
- **FGD** A fine-grained dedup approach with the Capping rewriting technique
- **CLD** A chunk-level dedup approach with Capping rewrite technique
- **MFD** A chunk-level dedup approach with an optimized data layout

Greedy

➢ Dataset

Name	Original Size	Versions	Workload Descriptions
WEB	269 GB	100	Backups of website: news.sina.com, captured from Jun. to Sep. in 2016
CHM	279 GB	100	Source codes of Chromium project from v82.0.4066 to v85.0.4165
SYN	1.38 TB	200	Synthetic backups by simulating file create/delete/modify operations
VMS	1.55 TB	100	Backups of an Ubuntu 12.04 Virtual Machine

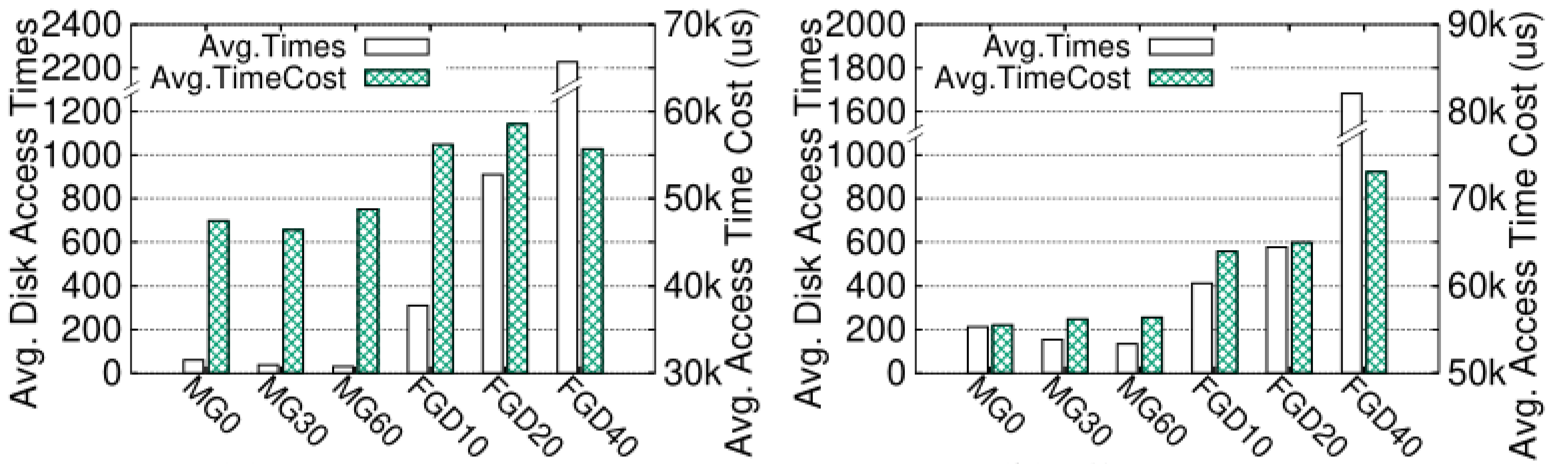
Evaluation backup speed



(a) WEB Dataset (b) CHM Dataset (c) SYN Dataset (d) VMS Dataset

- Applying several parameters for FGD and MeGA
- MeGA achieves a 4.47–34.45× higher backup speed than Greedy

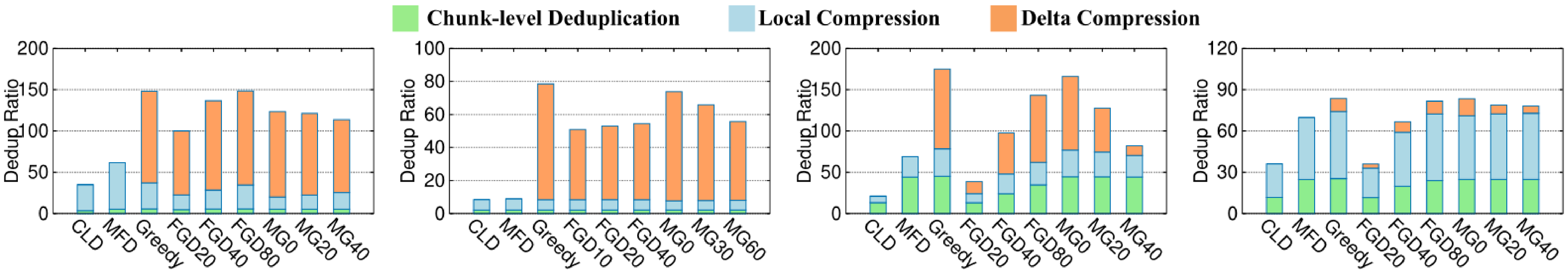
Evaluation statistics about accessing disks for reading bases



(a) WEB Dataset (b) CHM Dataset

- Selective Delta Encoding hugely reduces disk accessing times
- Skipping more delta encoding will lead to a better speed.

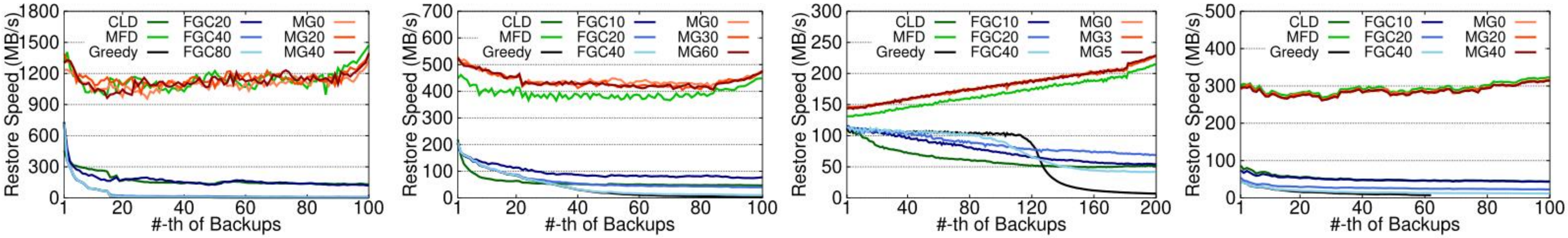
Evaluation Deduplication Ratio



(a) WEB Dataset (b) CHM Dataset (c) SYN Dataset (d) VMS Dataset

- Fine-grained dedup achieves higher dedup ratio on most datasets
- There are few similar chunks in the VMS dataset
- MeGA preserves deduplication ratio advantage
- The deduplication ratio loss caused by Selective Delta Encoding is limited

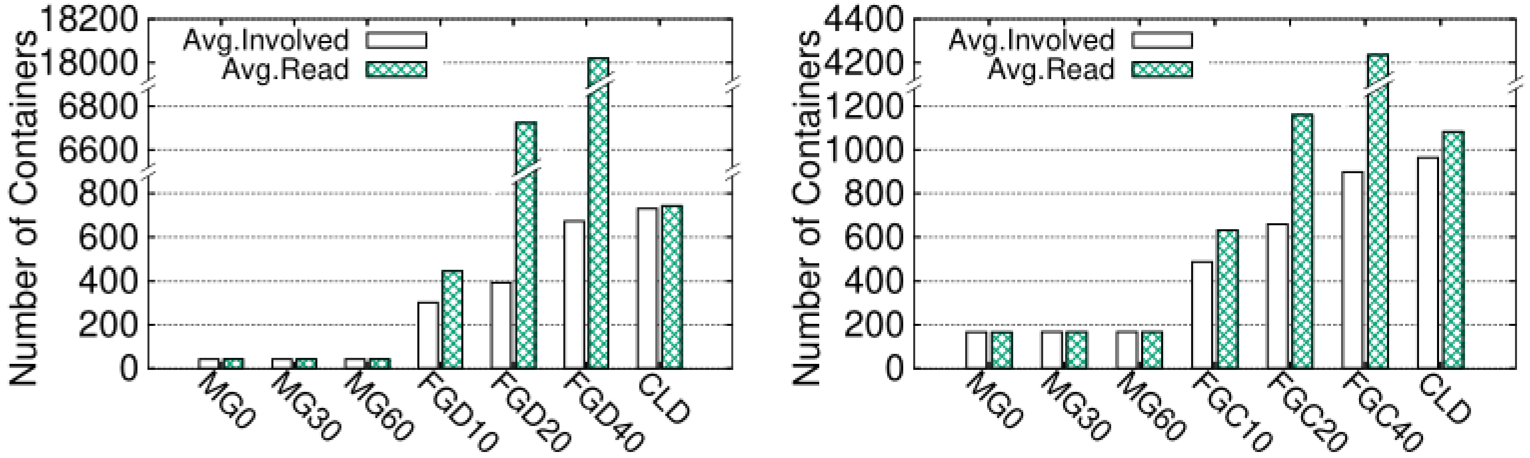
Evaluation restore speed



(a) WEB Dataset (b) CHM Dataset (c) SYN Dataset (d) VMS Dataset

- MeGA achieves a 30–105× higher restore speed than Greedy

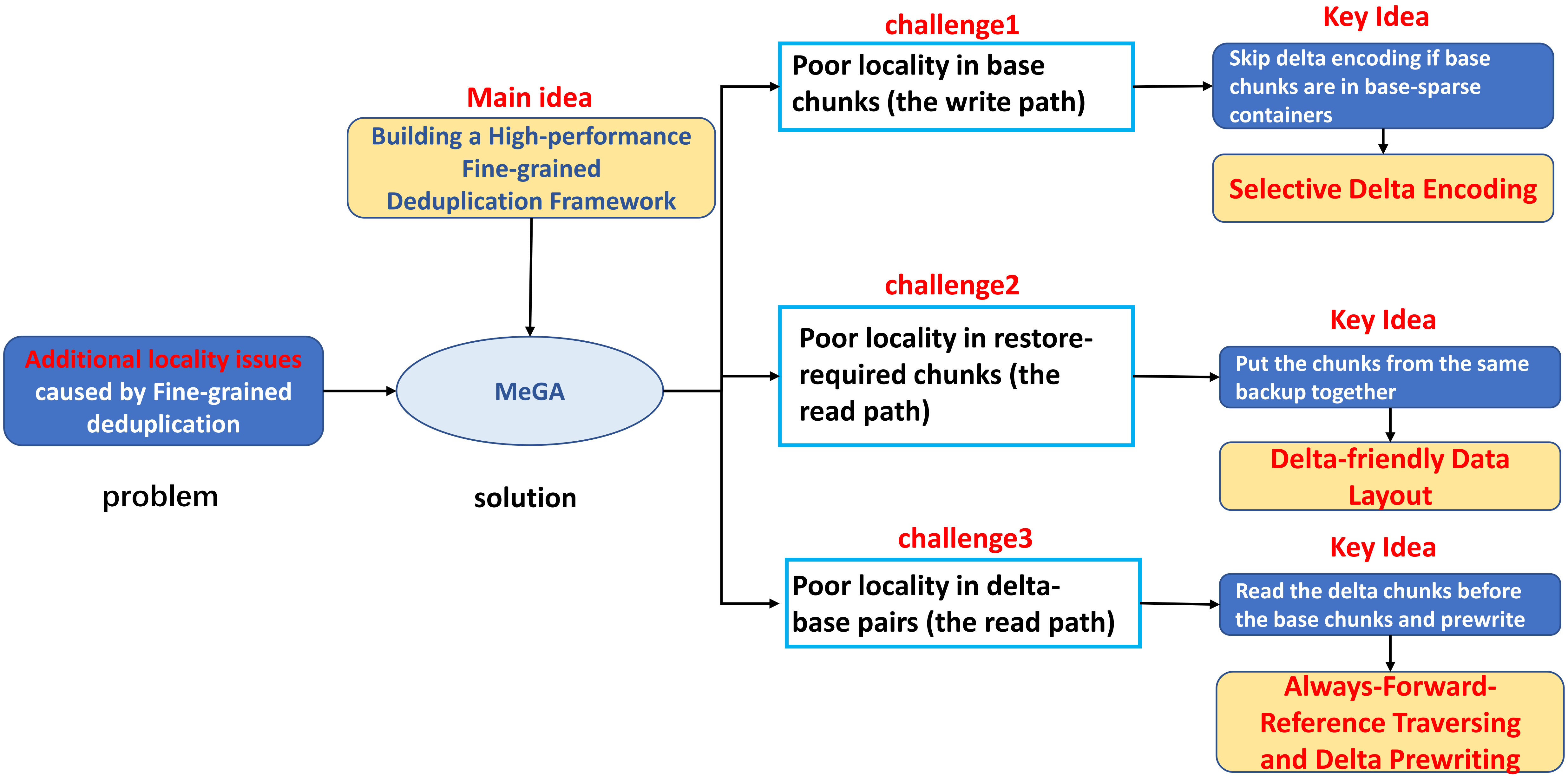
Evaluation statistics about accessing disks for required chunks



(a) WEB Dataset (b) CHM Dataset

- Our data layout hugely reduces the restore-involved containers
- Always-Forward-Reference Traversing and Delta Prewriting avoid the repeatedly accessing

Conclusion


Main idea
Building a High-performance Fine-grained Deduplication Framework
Additional locality issues caused by Fine-grained deduplication
problem
MeGA
solution
challenge1
Poor locality in base chunks (the write path)
challenge2
Poor locality in restore-required chunks (the read path)
challenge3
Poor locality in delta-base pairs (the read path)
Key Idea
Skip delta encoding if base chunks are in base-sparse containers
Selective Delta Encoding
Key Idea
Put the chunks from the same backup together
Delta-friendly Data Layout
Key Idea
Read the delta chunks before the base chunks and prewrite
Always-Forward-Reference Traversing and Delta Prewriting