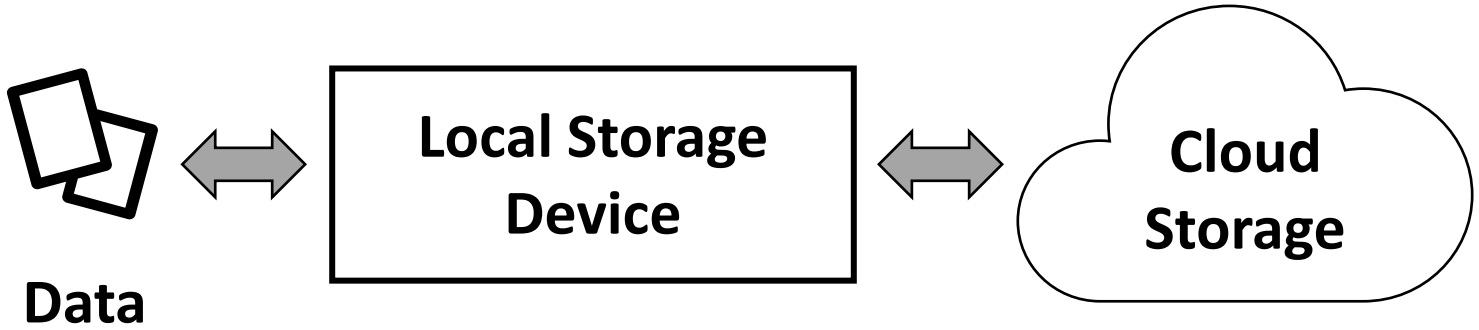
# Data Domain Cloud Tier: Backup here, backup there, deduplicated everywhere!

**USENIX ATC'19**

# Background

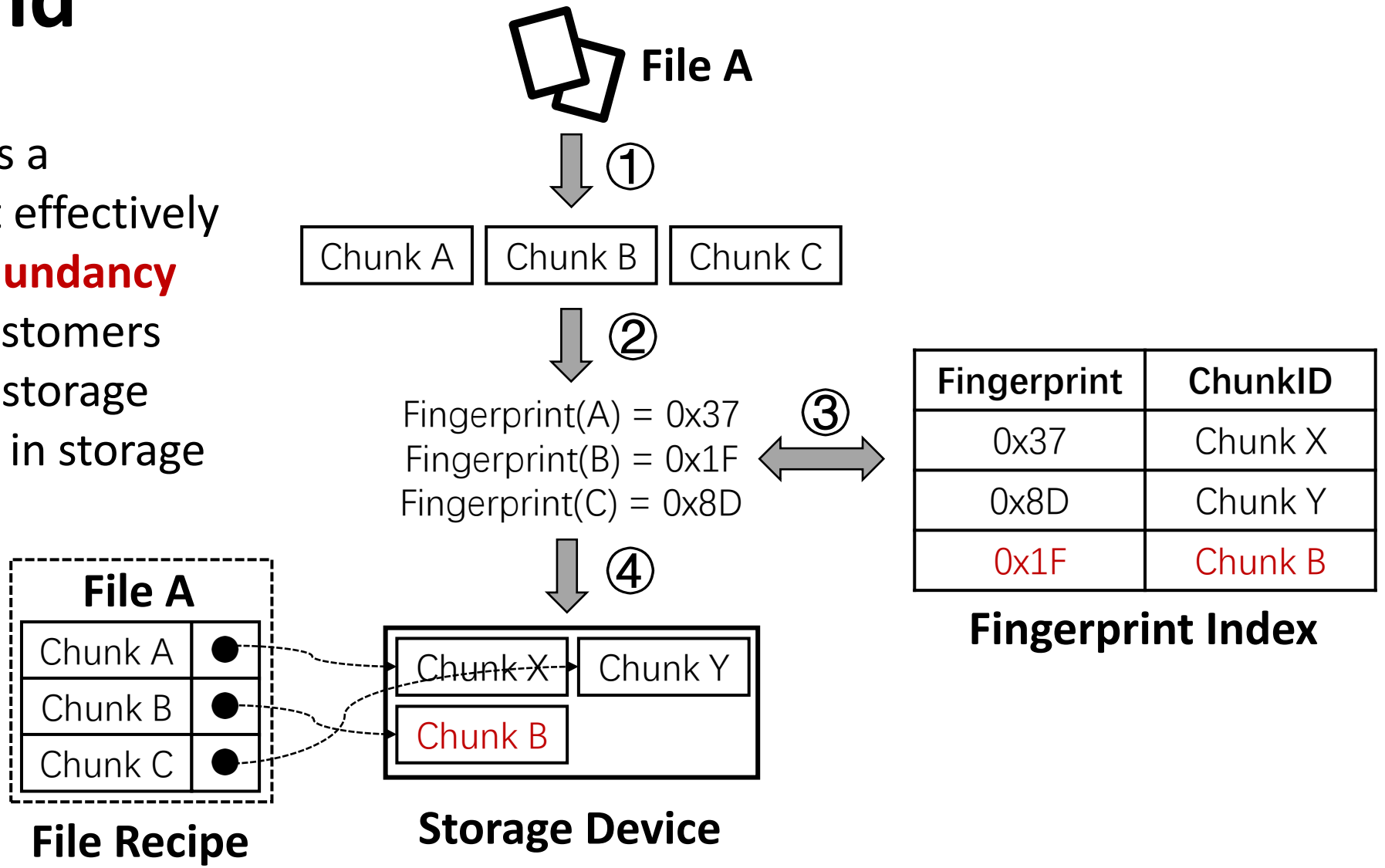
- The **need** for many customers and companies **to upload their data to the cloud** has led to the existence of many mature cloud storage products



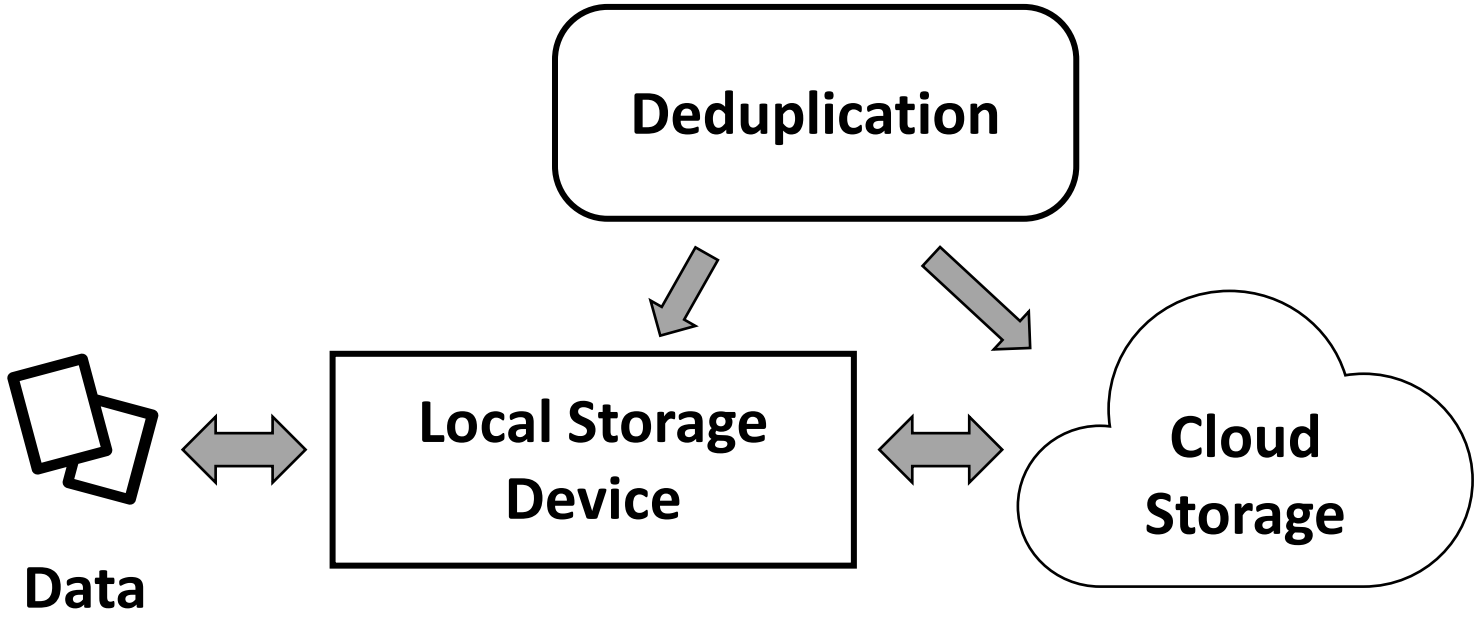

**Normal Cloud Storage Product**

# Background

- **Deduplication** is a technology that effectively **reduces file redundancy** and can help customers save significant storage cost when used in storage products



# Background




- With the introduction of deduplication in cloud storage products, there are **more details to consider** for user needs

# Main Idea

- Focus on innovations needed to support key functionality for customers
- Improvements to existing cloud storage systems that support deduplication

1) Customer determines which files to migrate to the cloud by estimating **how much space** will be freed on local storage
2) Customer transfers selected files to the cloud and later **restores** files back
3) Customer **deletes** a file in the cloud

# Main Work

- A deduplication-enabled cloud storage **architecture**
- A fast **algorithm** for scanning chunks
- **Optimization** of key functions using algorithms under the architecture:

    1) Estimate freeable space

    2) Seeding: the process of **migrating data** to the cloud for the **first time** by the customer, which involves a lot of file transfers

    3) File migration: data migration process after the first time

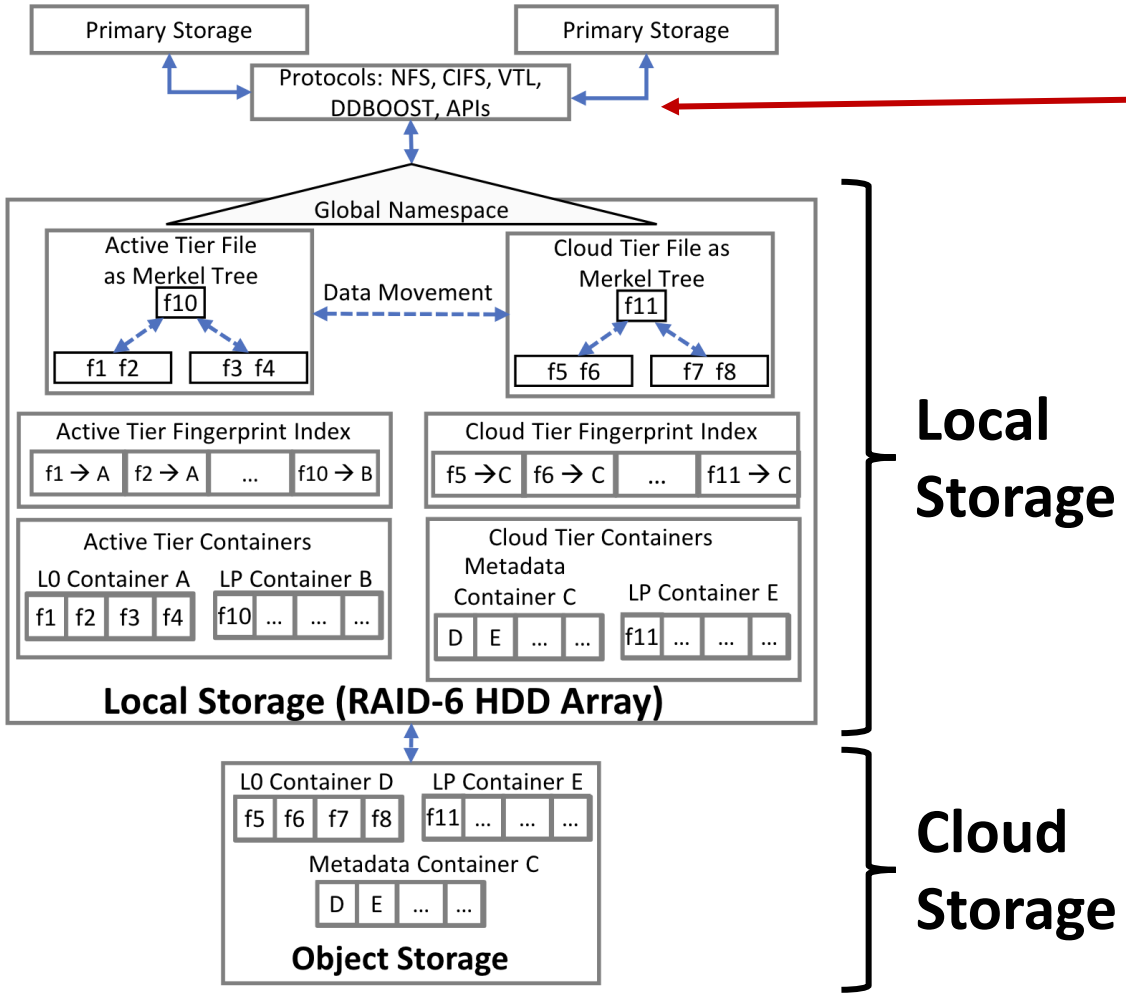
    4) File restore

    5) Garbage collection

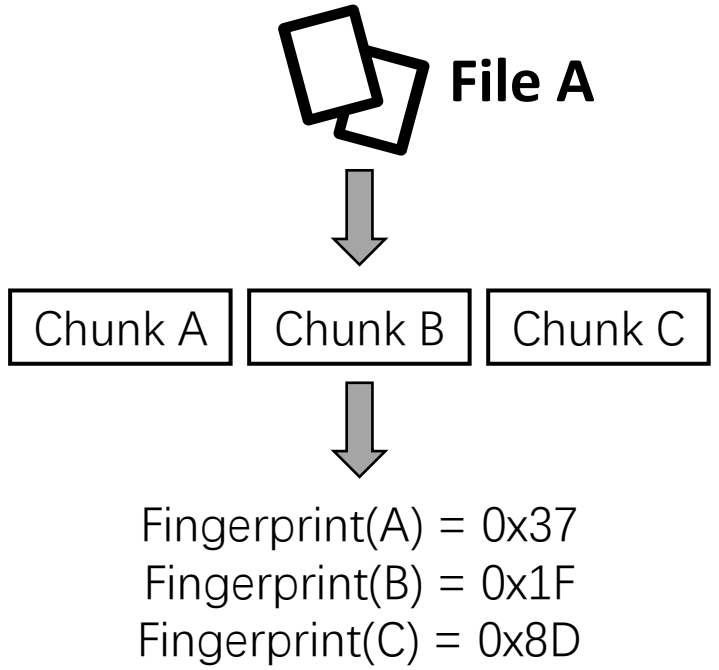
# Brief Summary

- The paper presents an **architecture** and an **algorithm** that optimizes for some specific **processes in cloud storage products** that support deduplication

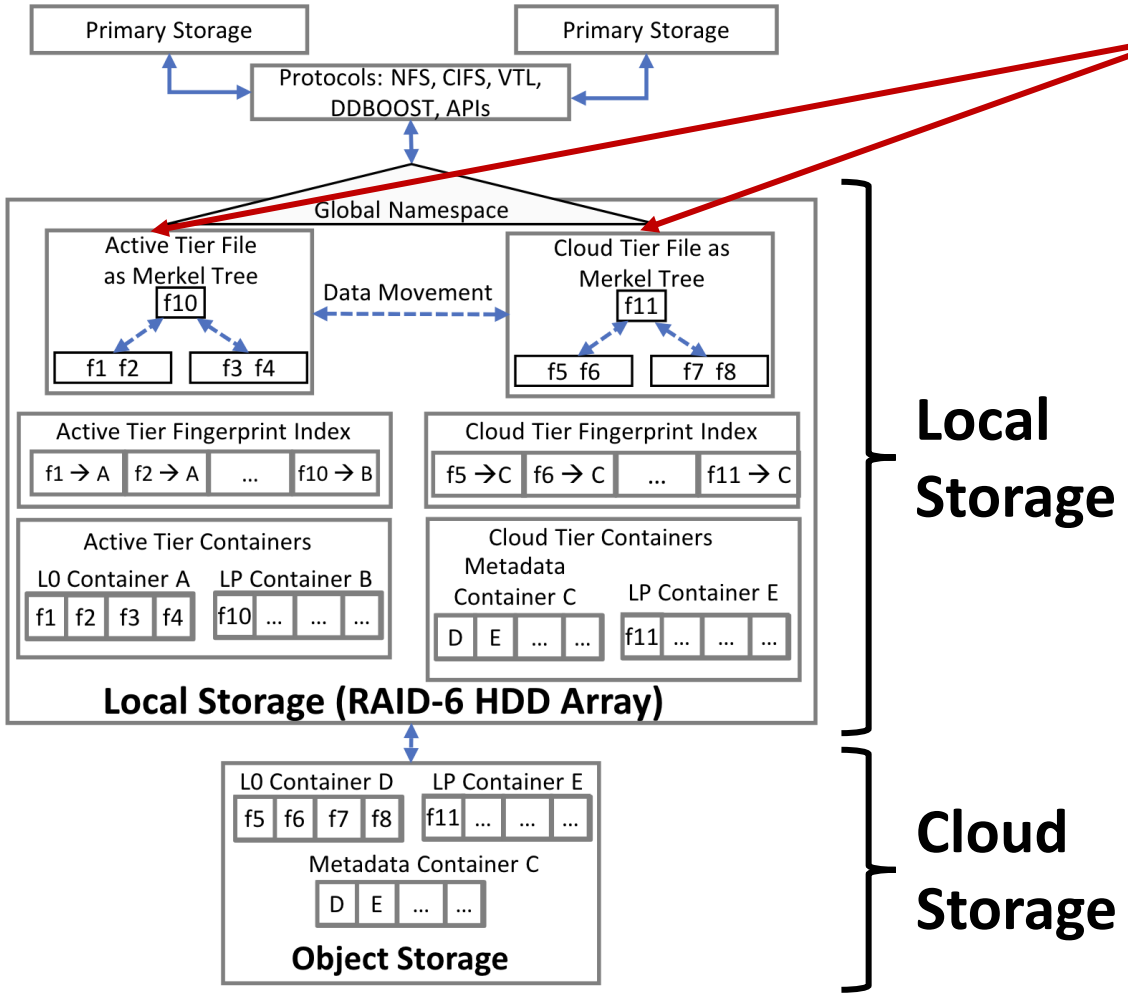
# Architecture



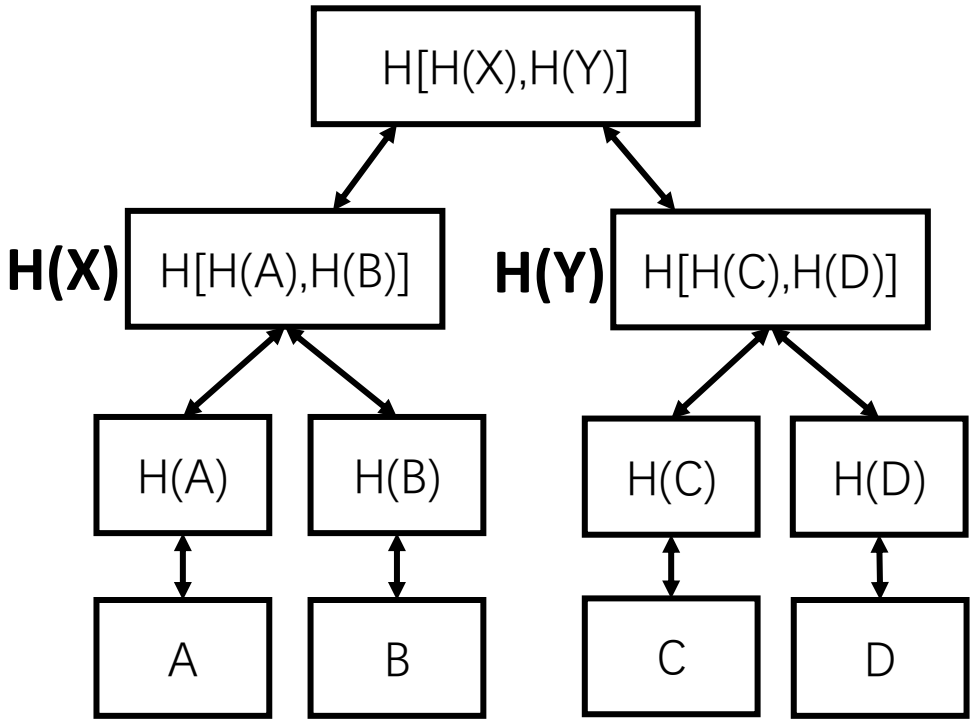

- When a file enters the system, it is **divided into variable sized chunks**
- Get the **fingerprint** of each chunk



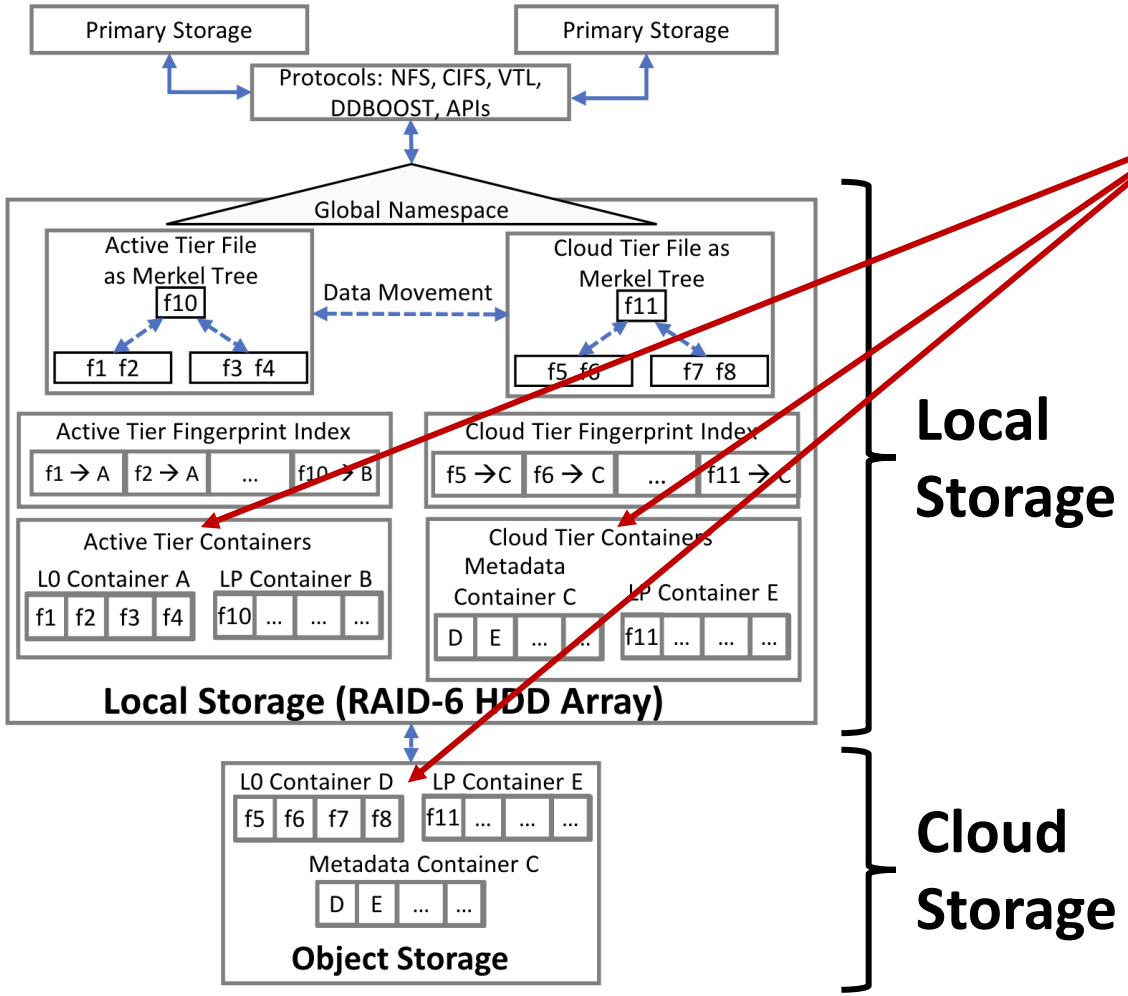
# Architecture



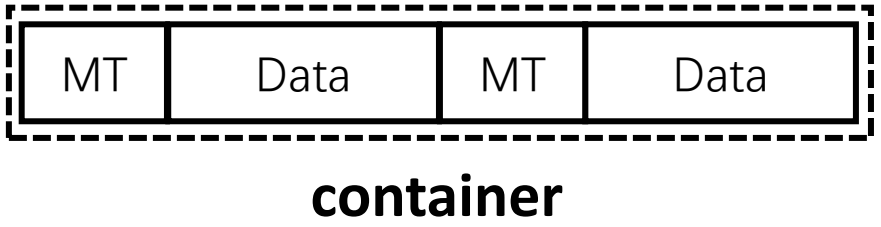

- Both locally stored and cloud stored files are presented by **Merkle Tree**



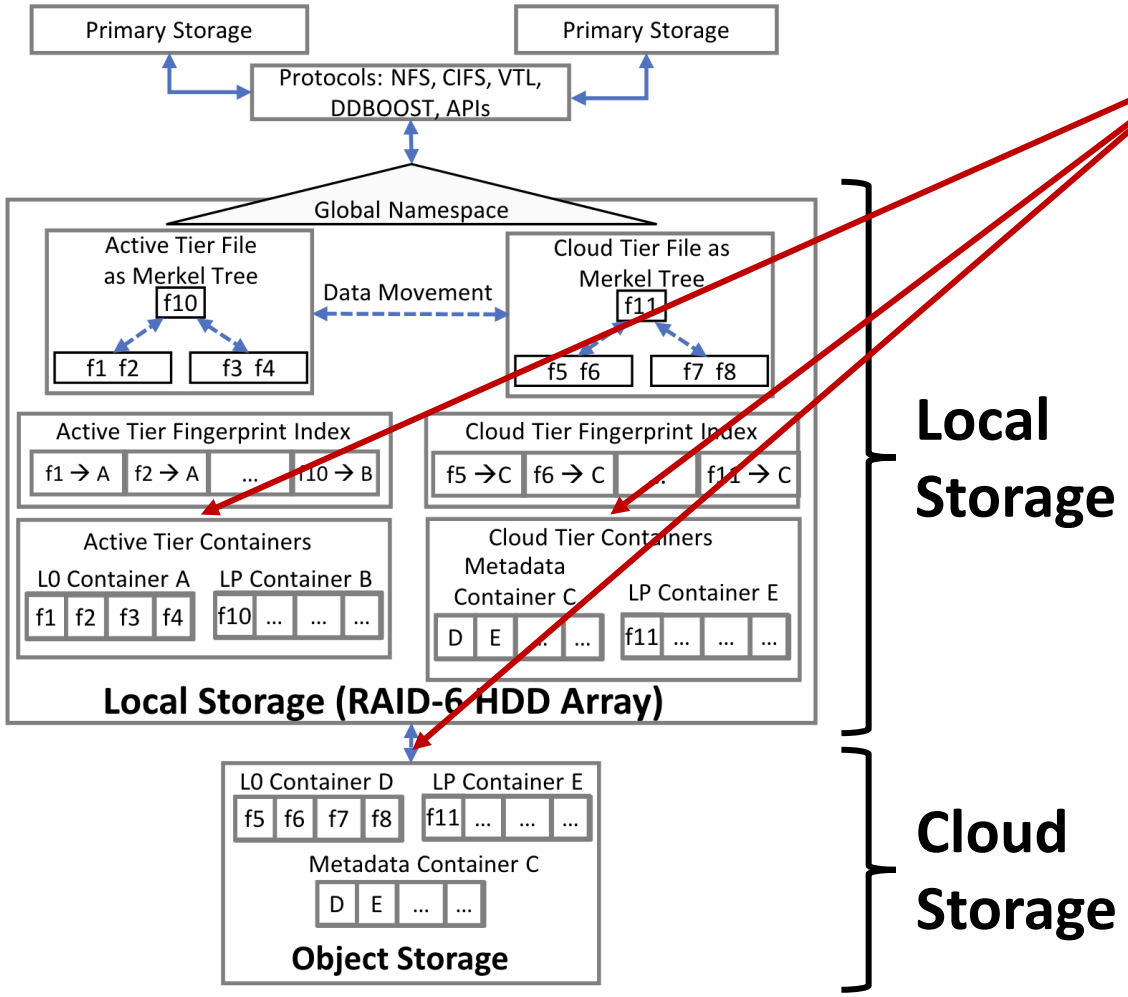
# Architecture



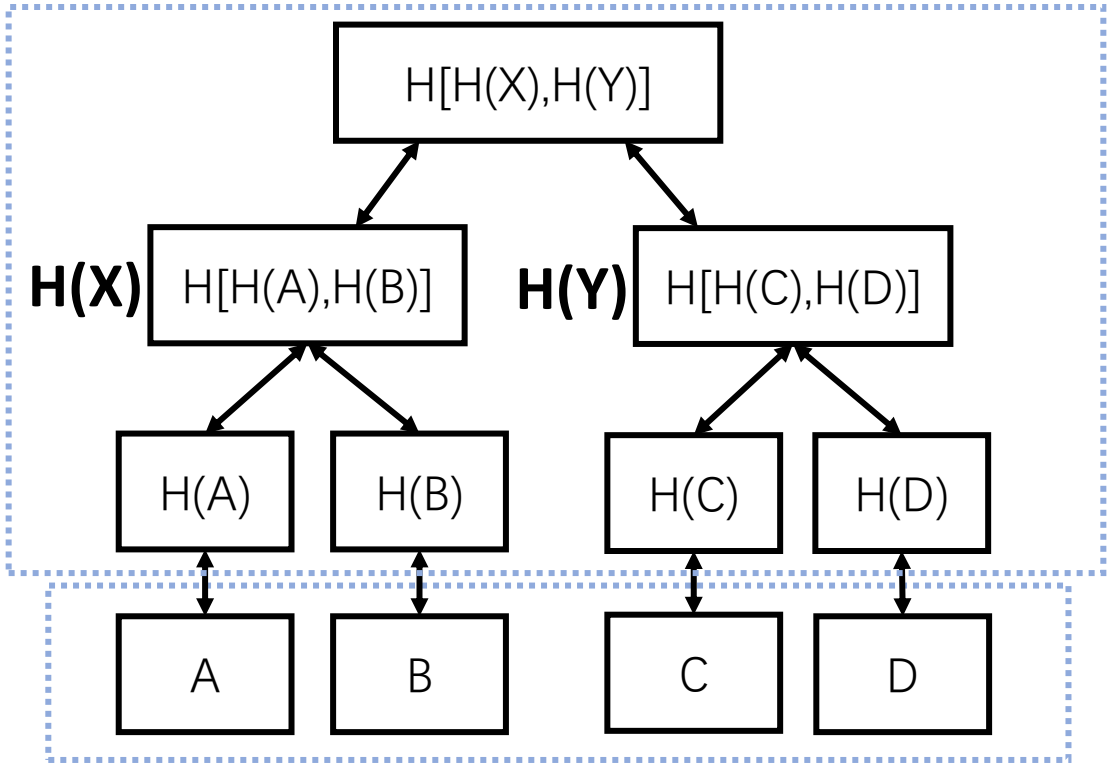

- Using **containers** to store chunks
- Each container will store chunks and their metadata including fingerprints



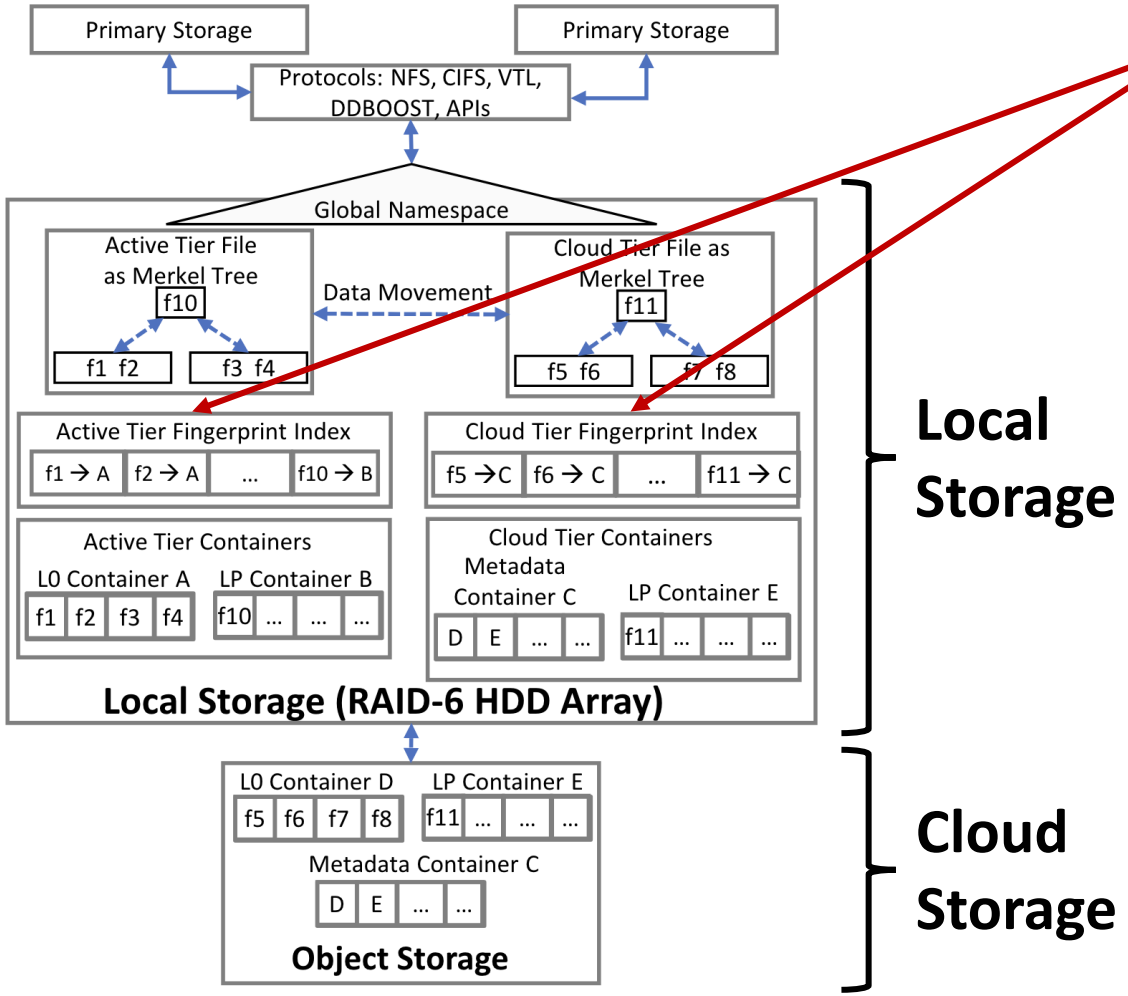
# Architecture




- Determine which container the fingerprint(data chunks) should be stored in according to its **position in the Merkle tree**



# Architecture



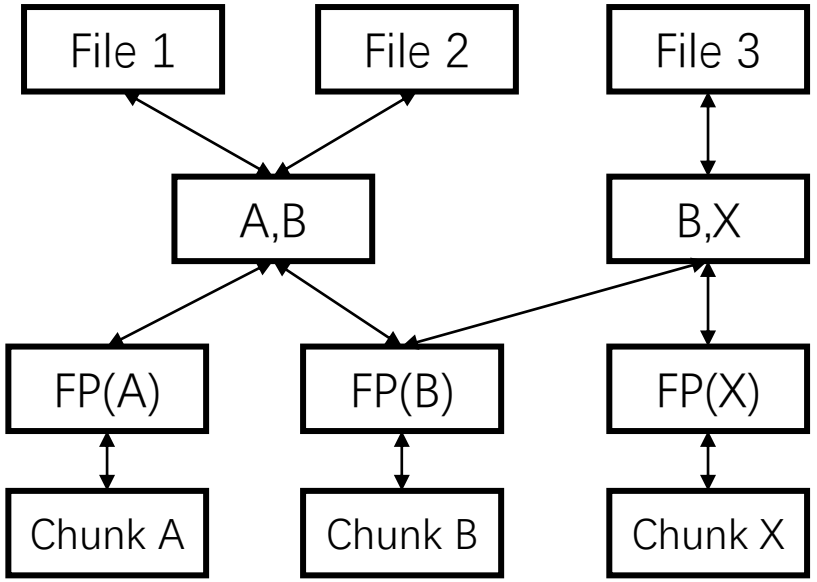

- Fingerprint indexes are used to find chunks while performing deduplication
- Consists of a mapping of **fingerprint to container number**

Deduplication happens when different files refer to the same L0 and LP chunks. As an example, if two files are exactly the same, they would have the same L6 fingerprint. But if two files only partially overlap in content, then some branches of the tree will be identical (LP and L0 chunks), while other branches will have different fingerprints. Multi-

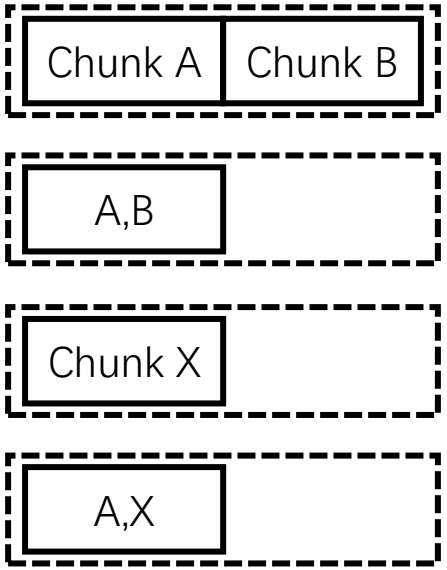
# Architecture

- Consider the following 3 files:

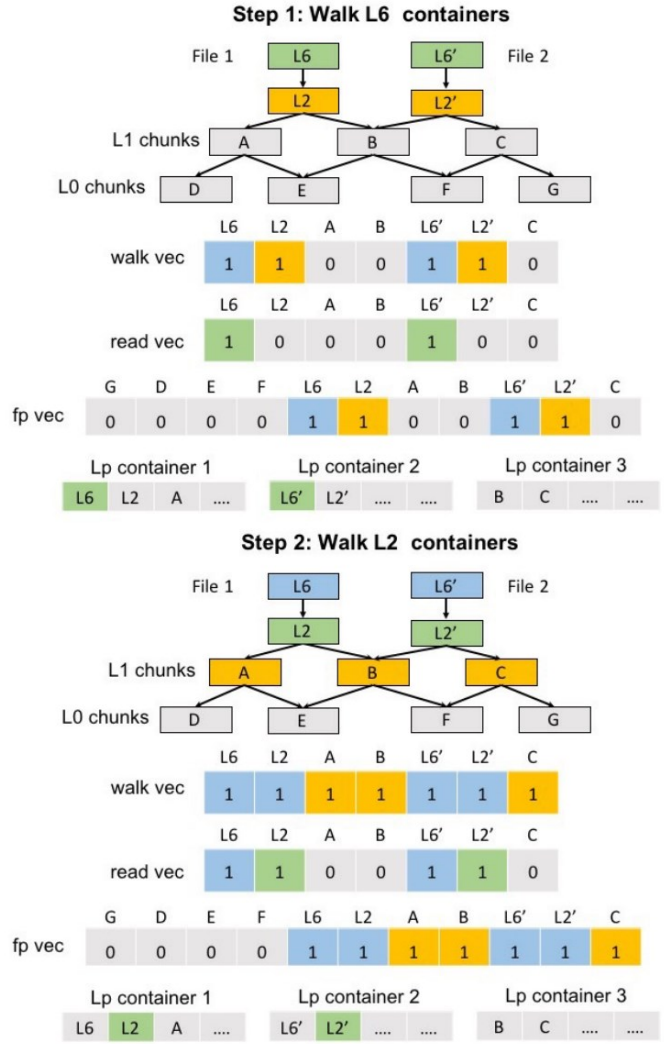
file1: chunk A; B
file2: chunk A; B
file3: chunk B; X




| Fingerprint | ChunkID |
|---|---|
| FP(A) | Chunk A |
| FP(B) | Chunk B |
| FP(A,B) | Chunk A,B |
| FP(X) | Chunk X |
| FP(B,X) | Chunk A,X |



# Fast Scanning Algorithm

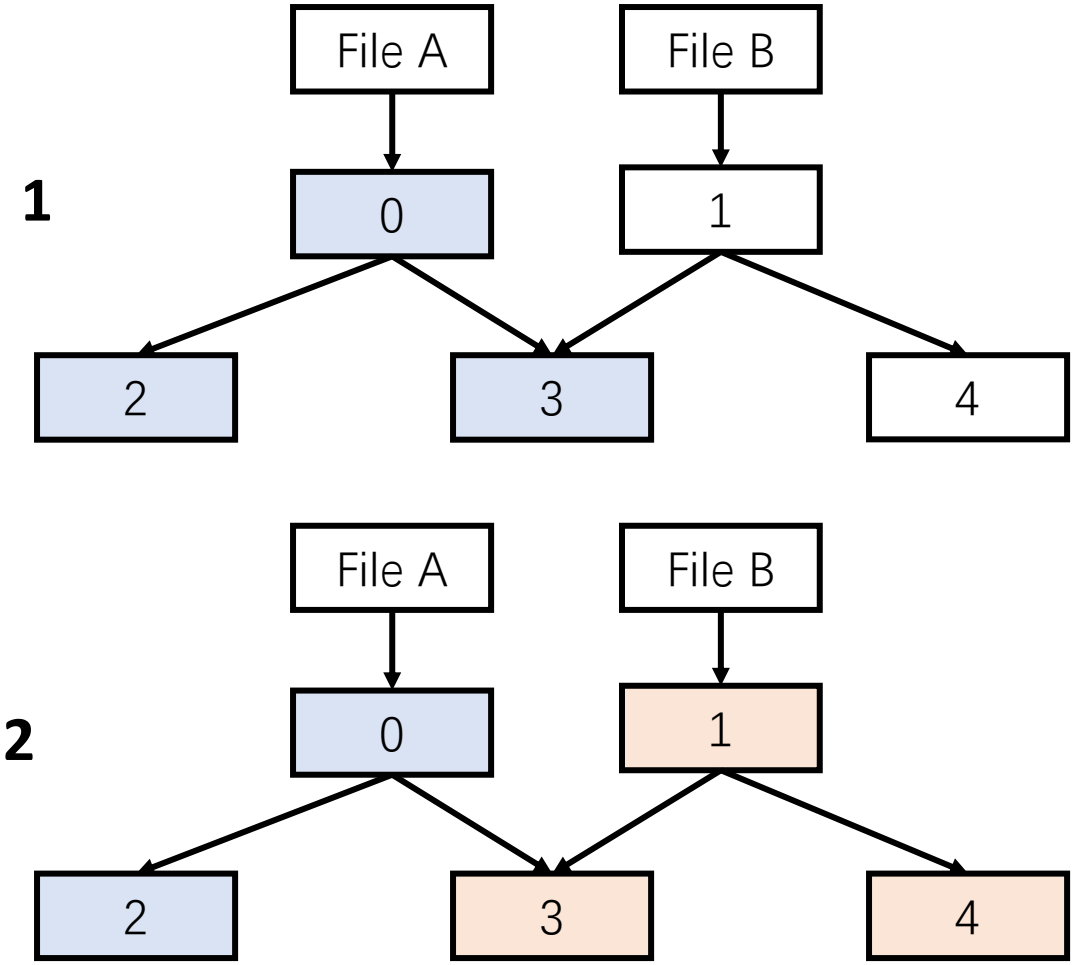



- Constructing three hash spaces
- The size of two of them is **equal to the number of fingerprints in the LP container**
- The size of the other one is equal to the number of **all fingerprints**

# Estimate Freeable Space

- For a cloud storage system that supports deduplication, the space freed by migrating a file **is not equal** to the size of the file
- The system needs to estimate for the customer the size of space that can be freed up by migrating the files
- Achieved by two times of fast scanning algorithms

# Estimate Freeable Space



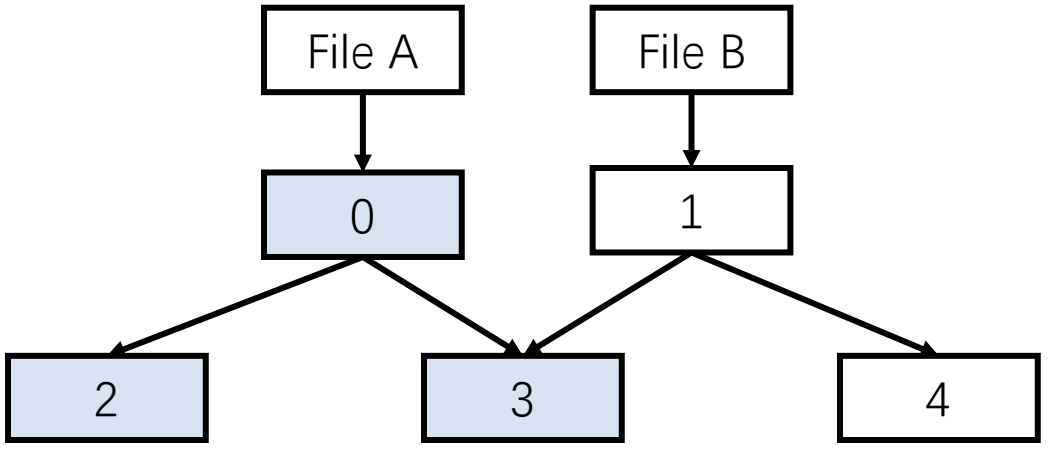

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **FP Vec** | 1 | 0 | 1 | 1 | 0 |

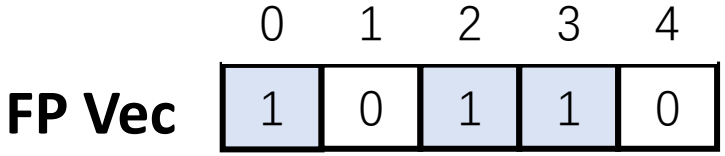
| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **FP Vec** | 1 | 0 | 1 | 0 | 0 |

# Seeding

- The first time a customer uploads a large number of files to the cloud tier
- The process can involve terabytes of files, and file uploads can take weeks

1. Use the fast scan algorithm to get all the chunks that need to be uploaded
2. Remove the chunks from the original container, then store them in a new container and upload them
3. For each chunk uploaded, set the corresponding bit position 0 of that chunk in the fingerprint vector



# File Migration

- The process of uploading a small number of files after seeding

1. Scan the file and get the chunks that need to be uploaded
2. Query cloud tier fingerprint index with chunk's fingerprint
3. If there is no corresponding record in the cloud fingerprint index, the chunk is stored in the new container and then uploaded.

# File Restore

- The process of recovering data from the cloud tier to active tier

1. Scan the file and get the chunks of the file
2. Use chunk‘s fingerprint to query the active tier fingerprint index
3. If the corresponding record is found, means the chunk is available locally and does not need to be downloaded from the cloud

# Garbage Collection

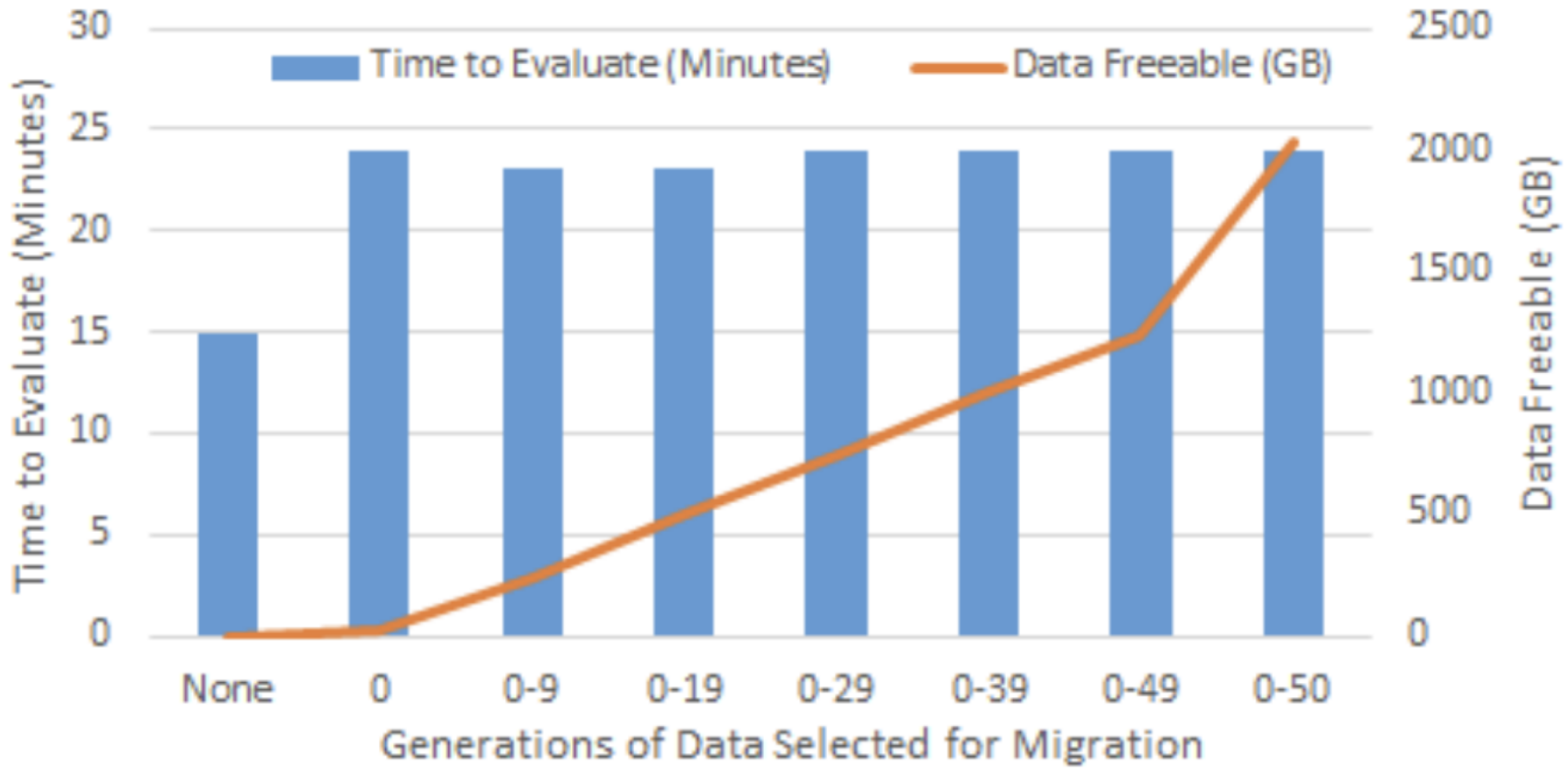
- Delete the useless chunks and store the retained chunks in a new container
- Use the fast scan algorithm to get the chunks that need to be retained
- For data in the cloud tier, API are provided for the cloud provider to perform the above process directly in the cloud tier, without the need to transfer the data back to the active tier

# Evaluation

- Data moved to the cloud versus total compression

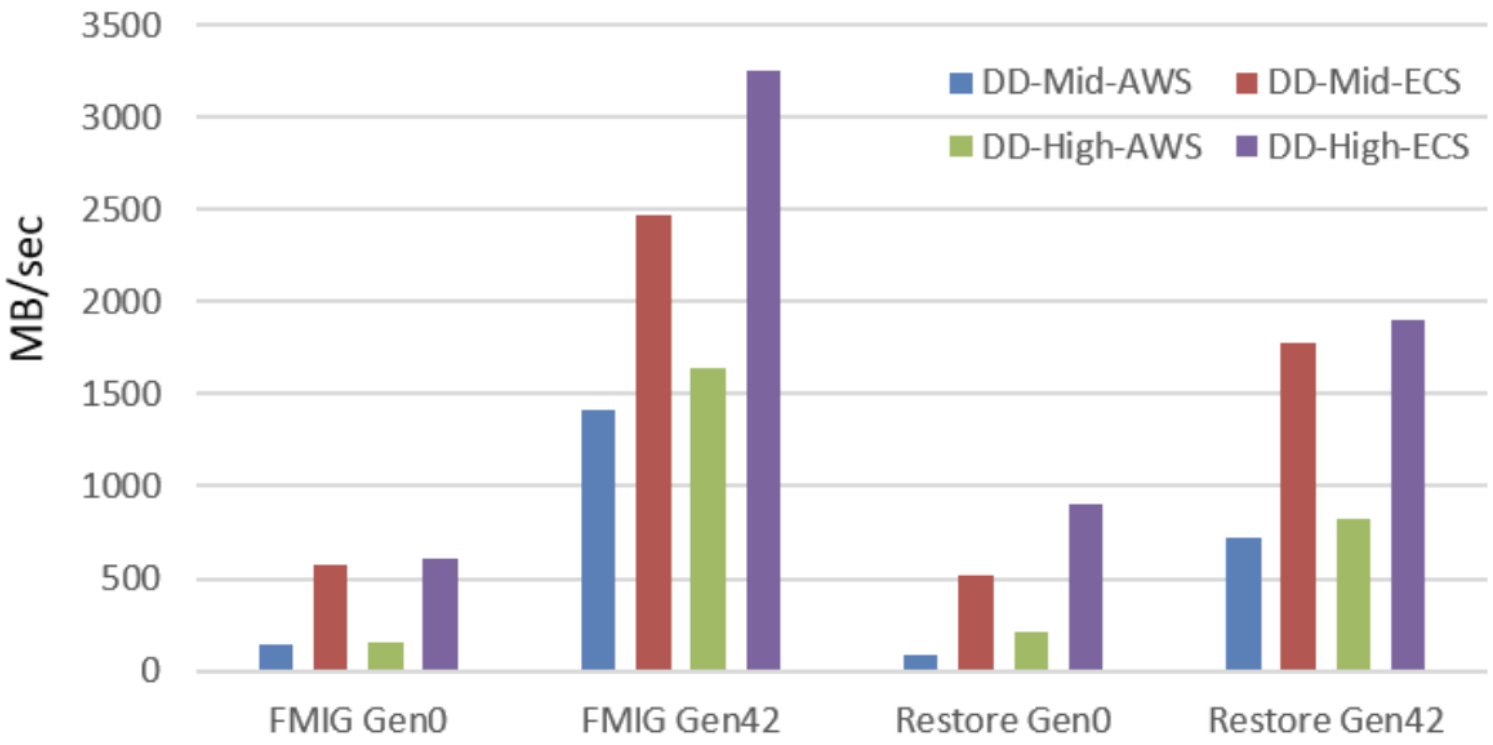
# Evaluation

- Space estimation performance



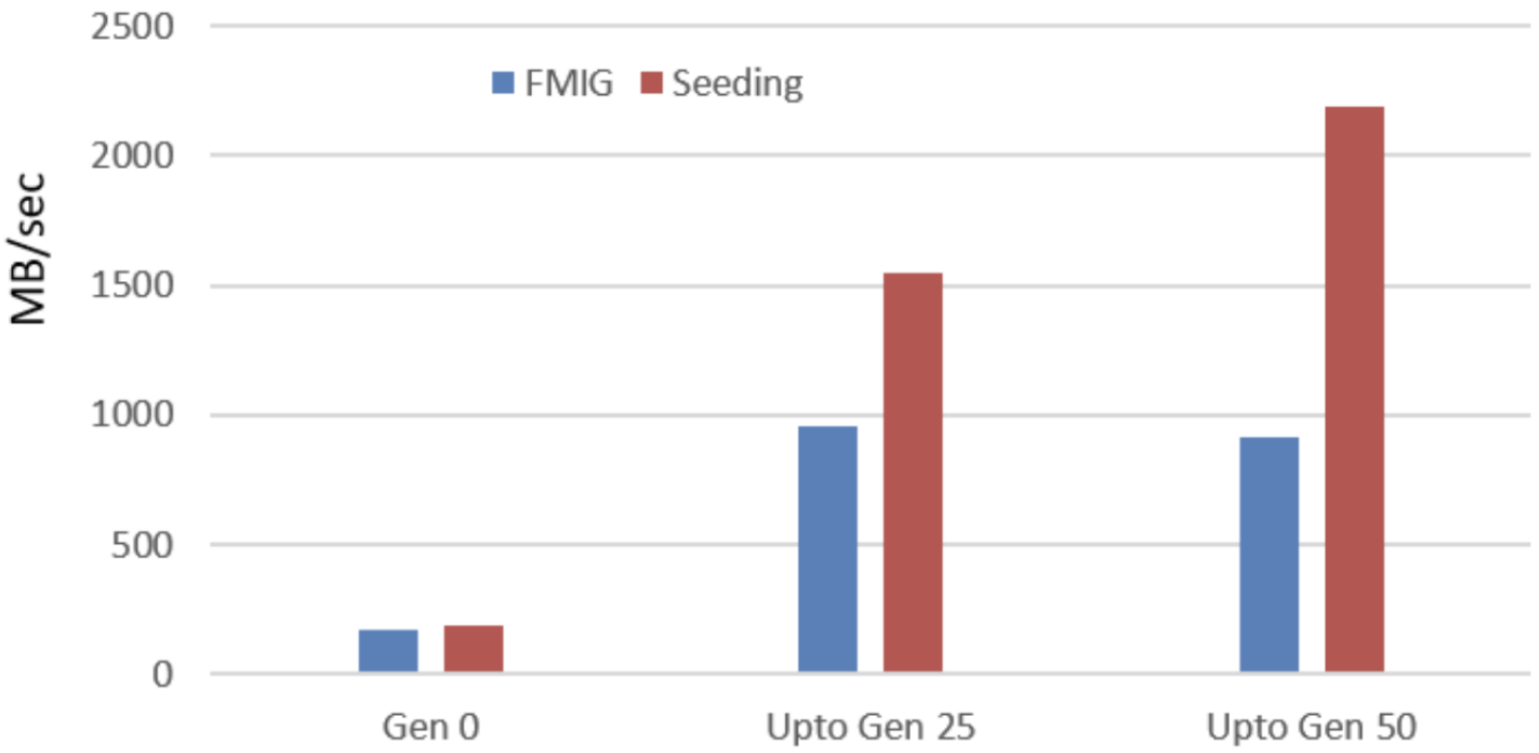
# Evaluation

- File Migration and Restore performance



# Evaluation

- File Migration vs. Seeding performance



# Evaluation

- Figure 11: GC copy forward with different algorithms