

# **TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization**

Navendu Jain

Mike Dahlin

University of Texas at Austin

Renu Tewari

IBM Almaden Research Center

# Motivating Example

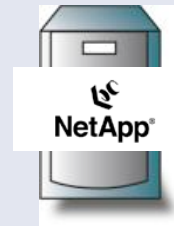
- A Content distribution network
  - Goal: Efficient synchronization of current data at master with older versions of the data at replicas

Master  
Server



[sourceforge.net]

Synchronize



Replicas

- Key Challenges:

- Minimize network bandwidth
- Minimize master overheads
- Inter-operable with heterogeneous replicas

## Broader Goal

- A Scalable Data Replication Protocol
  - **Aim:** Synchronize a large collection of data across multiple geographically distributed replica locations (possibly low-bandwidth links)
  - **Examples:**
    - Software distribution mirroring
    - Content distribution networks
    - Backup and recovery
    - Versioning systems
    - Federated file systems

# Our Contributions: TAPER

- A **universal** data synchronization protocol
  - Provide four key properties
    - **Bandwidth Efficient**
      - Tiered approach --- four redundancy elimination phases that **trade** computation for bandwidth
      - 15-71% bandwidth savings over Rsync
    - **Scalable**
      - Low, re-usable computation at the master
    - **Speed**
      - Fast matching at the replica
    - **Content-based**
      - Does not require prior knowledge of replica state

# Talk Outline



Motivation



**TAPER**

— [ Overview

— [ Algorithm

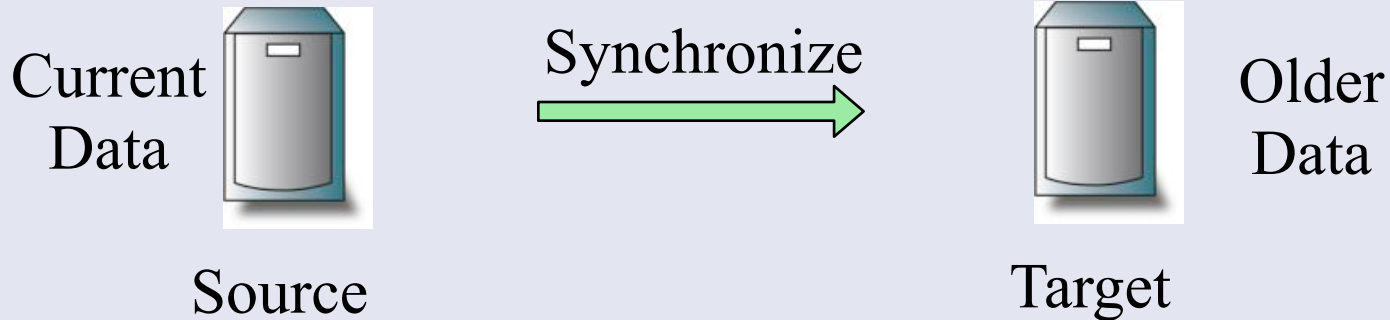


Experimental Evaluation



Conclusions

# Background: Data Synchronization



## ▪ Existing Solutions:

### ➤ Perfect Knowledge of remote replica

- Delta-compression, Snapshot Differencing, Versioning
- Not applicable in our context

### ➤ Negotiation-based: Hash-based differential compression

- Sliding Block (Rsync)
- Content-defined Chunking (LBFS)

# Our solution: TAPER

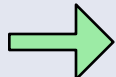
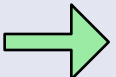
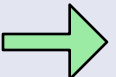
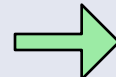
- Hash-based technique for data synchronization

- Matching granularity

- Coarse-grained matching
      - Lower metadata, computation overhead but fewer matches
    - Fine-grained matching
      - Lower # bytes transferred but higher metadata, computation

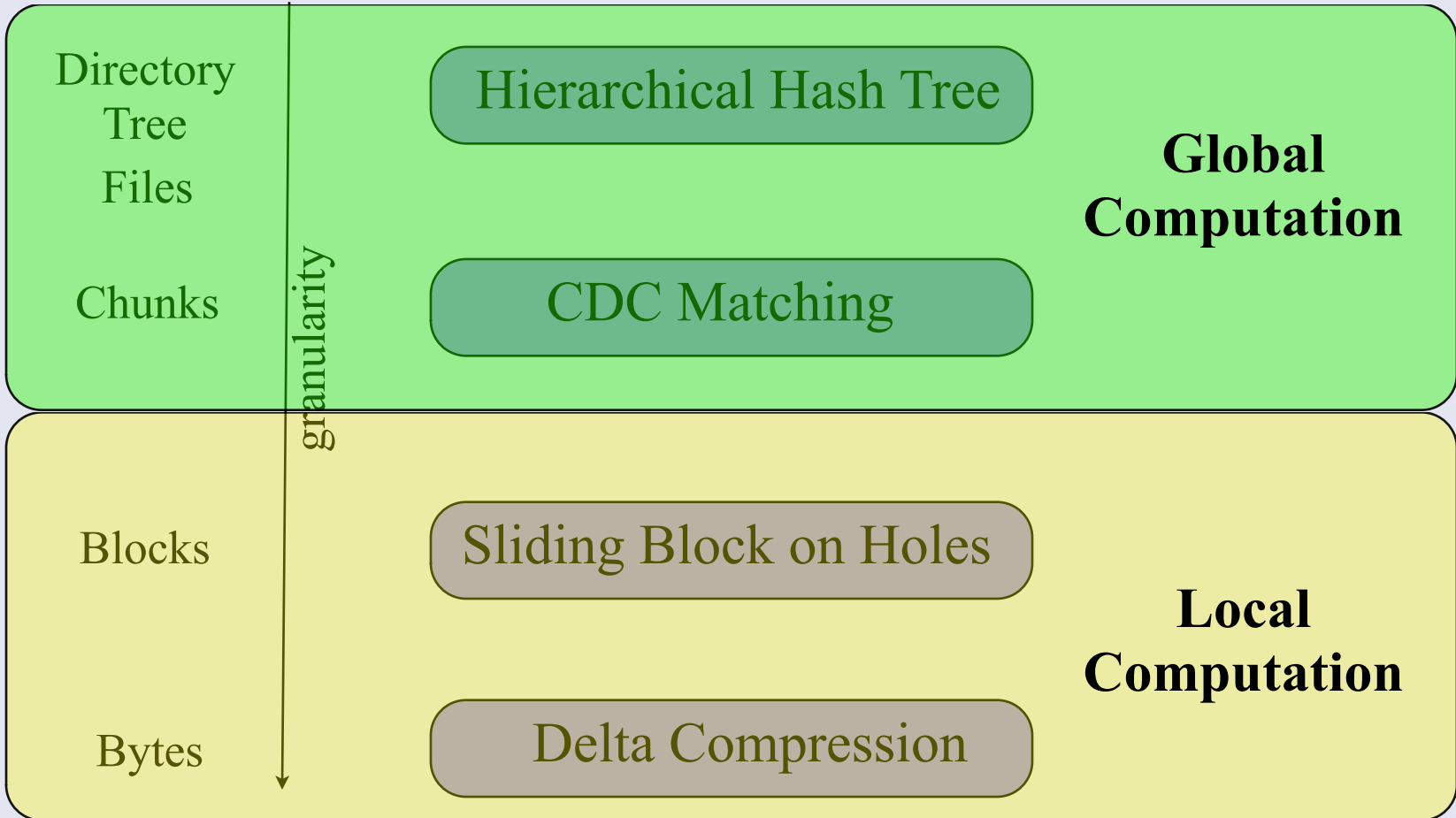
- Key Insight

- Tiered Approach (Different Granularity)

- Progressively match from larger to finer granularity
    - Multi-phase hierarchical protocol
    - Directory  Files  Chunks  Blocks  Bytes

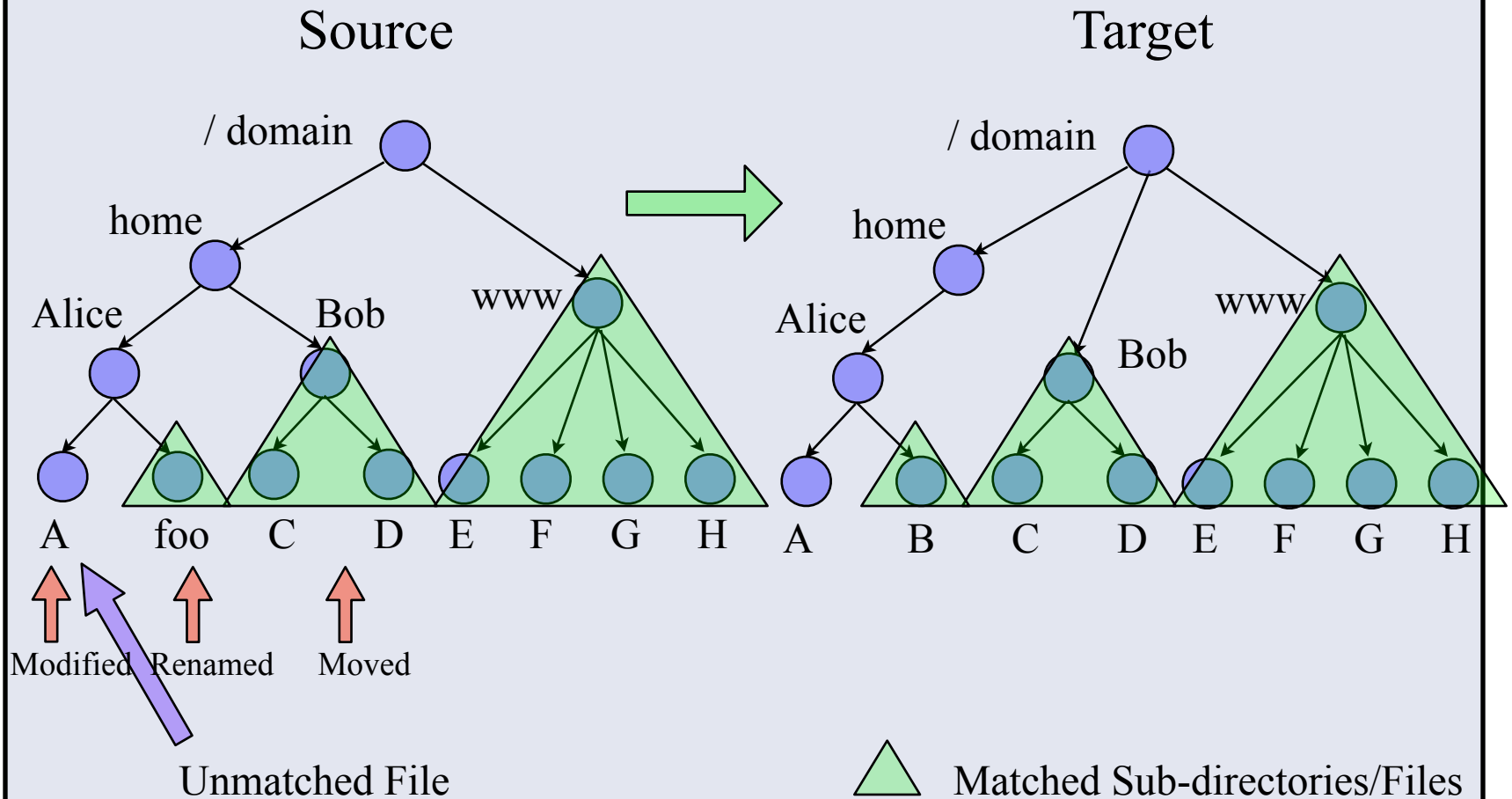
# TAPER Protocol Phases

- four redundancy elimination phases

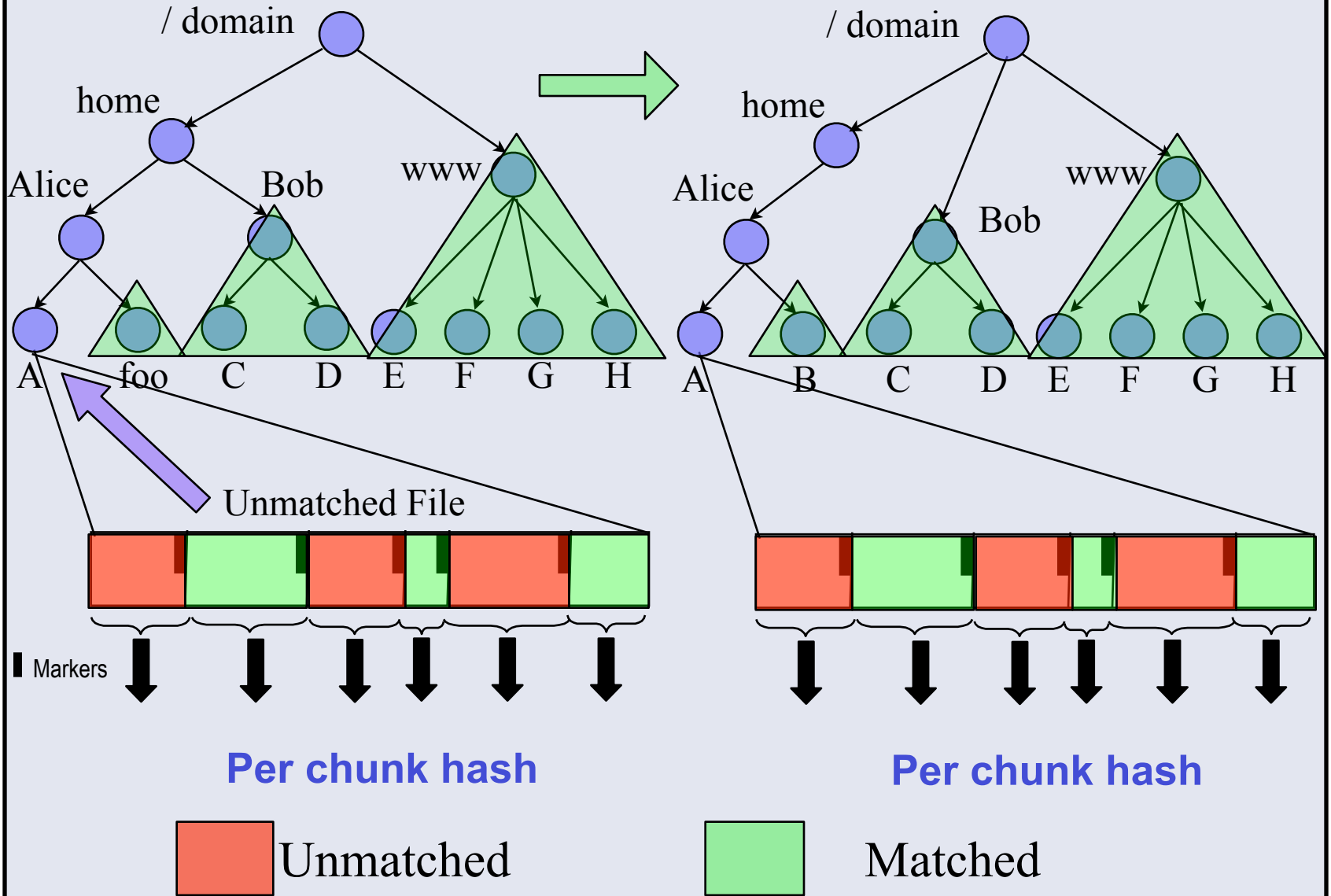


# Phase I: Directory Tree and Files

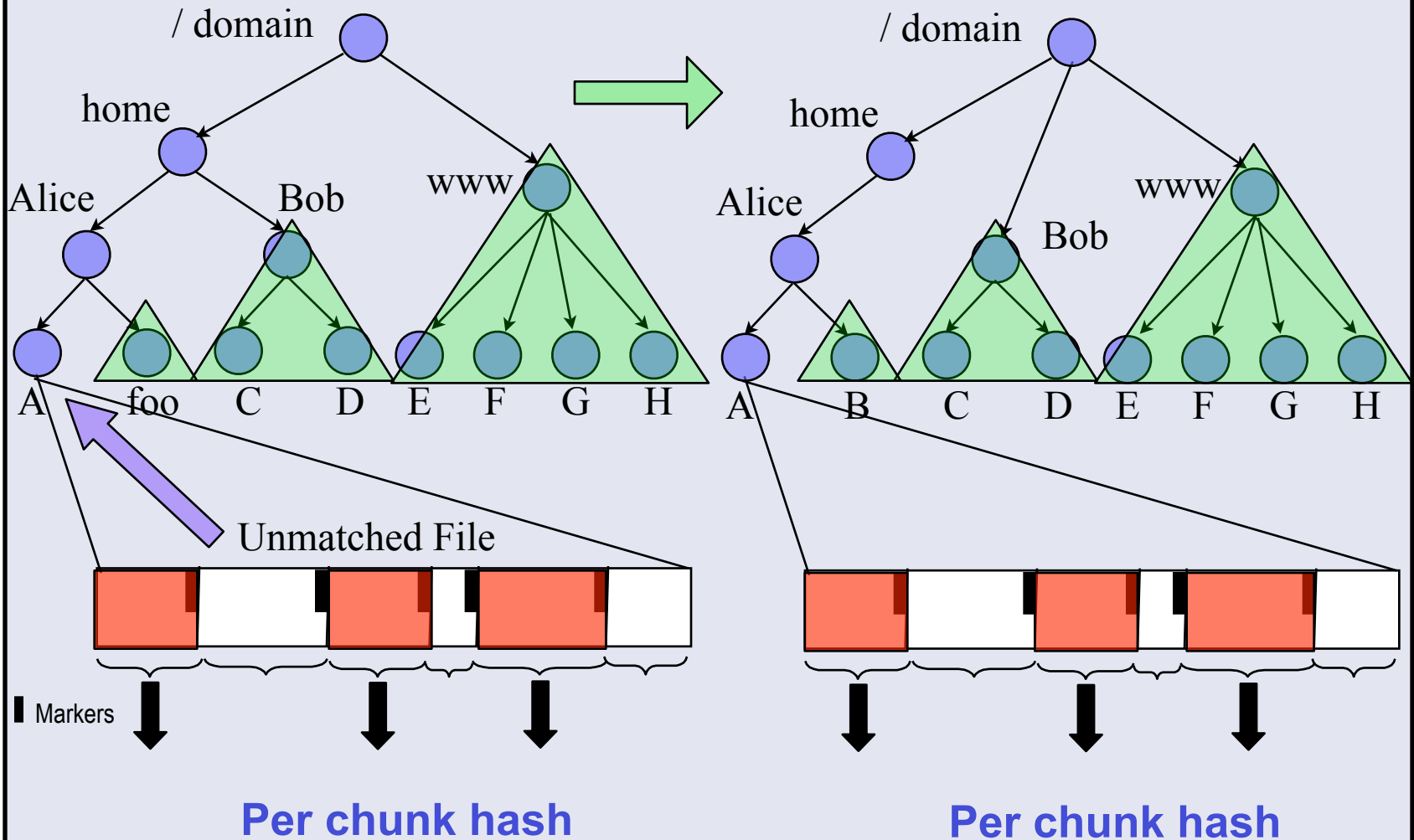
- Hash of Directory Tree + File Hashes



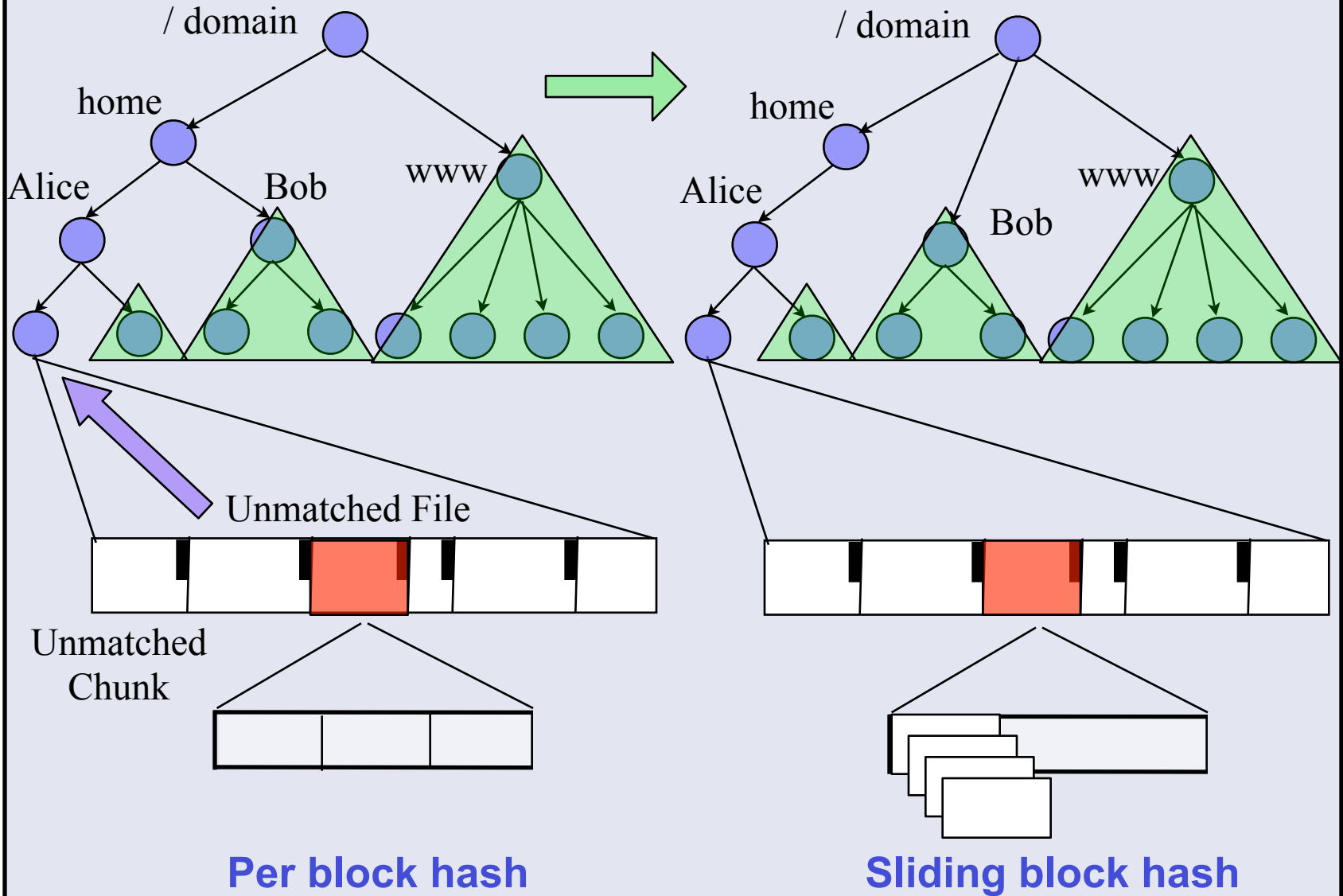
# Phase II: Large File Chunks (Unmatched Files)



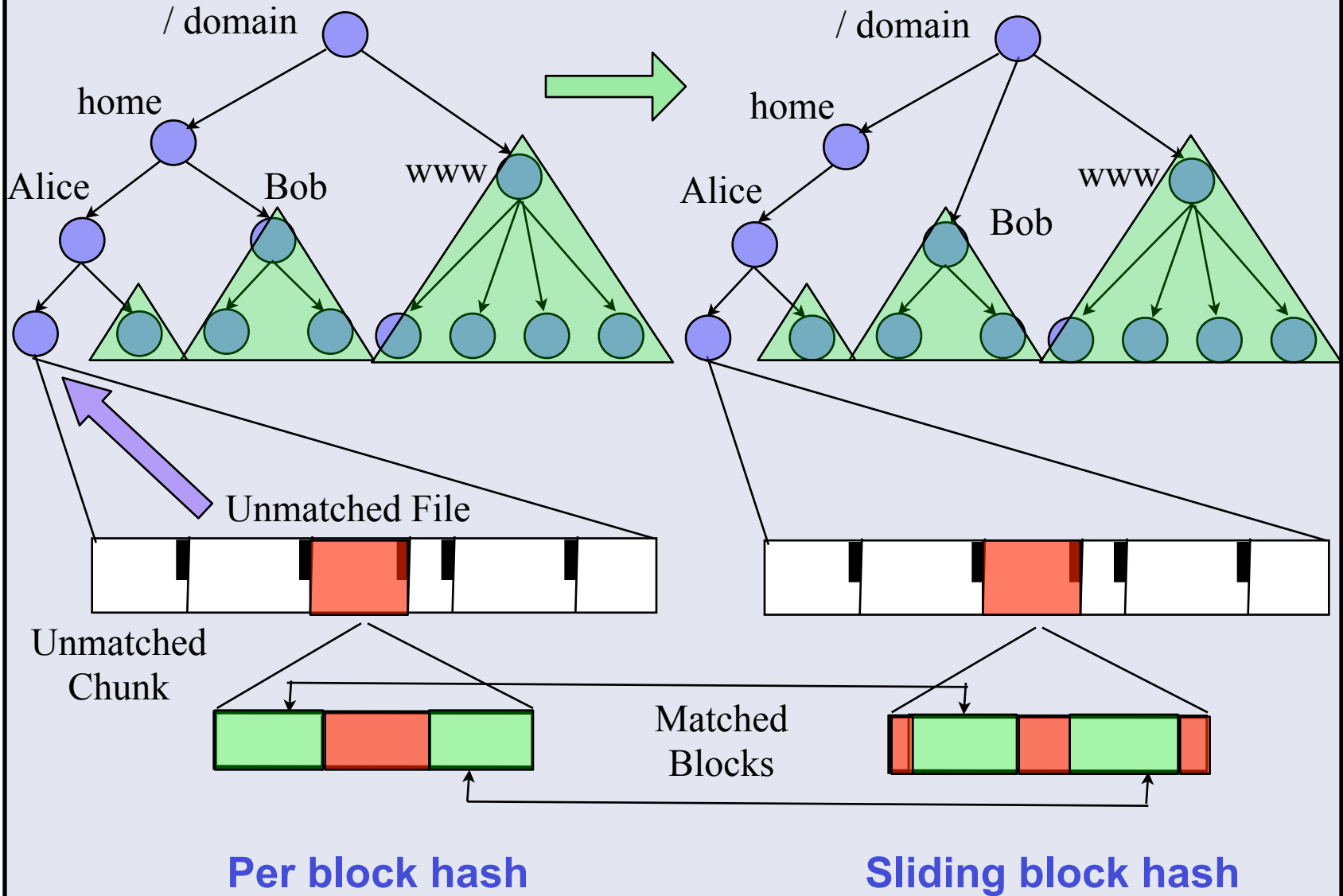
# Phase II: Large File Chunks (Unmatched Files)



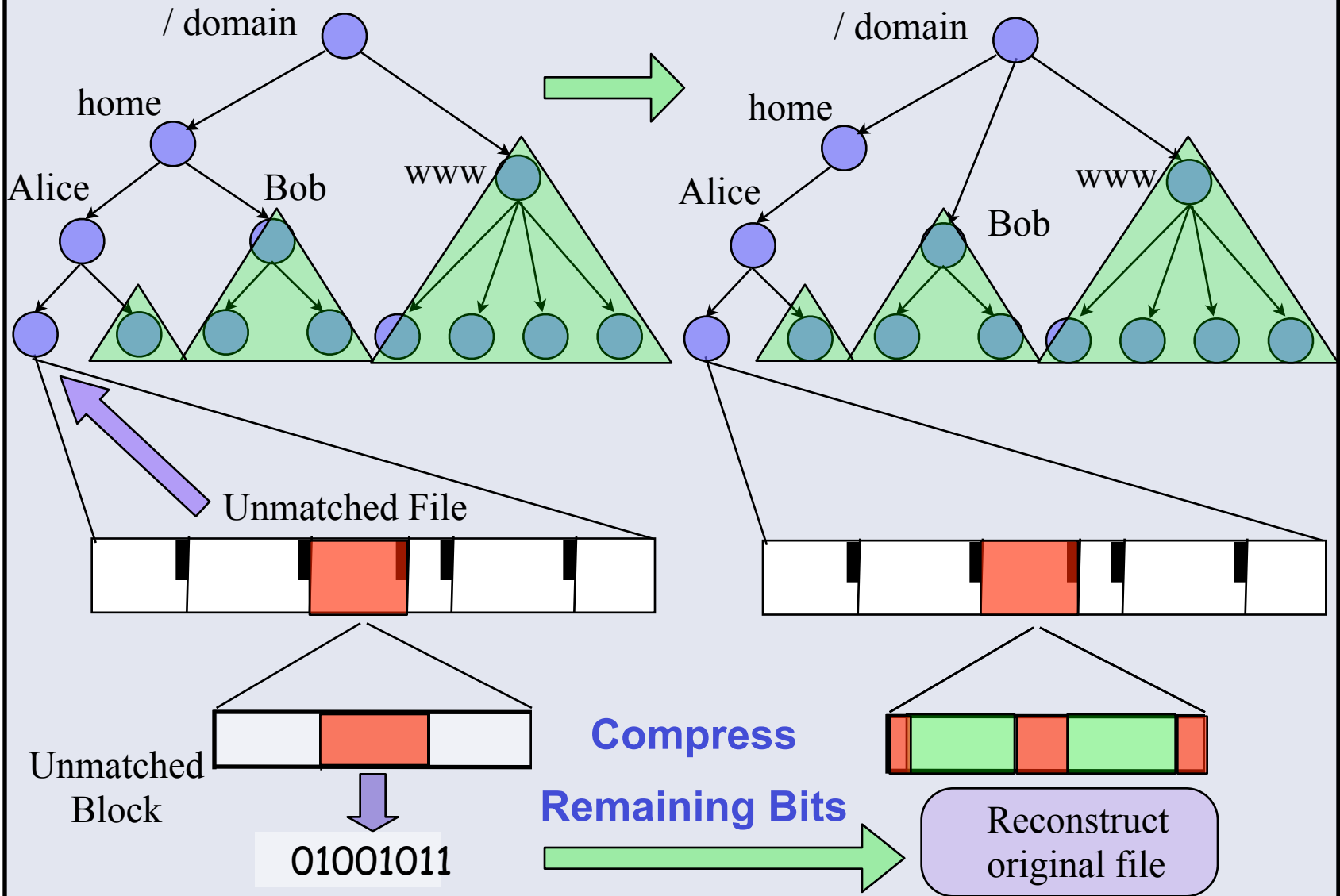
# Phase III: Small File Blocks (Unmatched chunks)



# Phase III: Small File Blocks (Unmatched chunks)

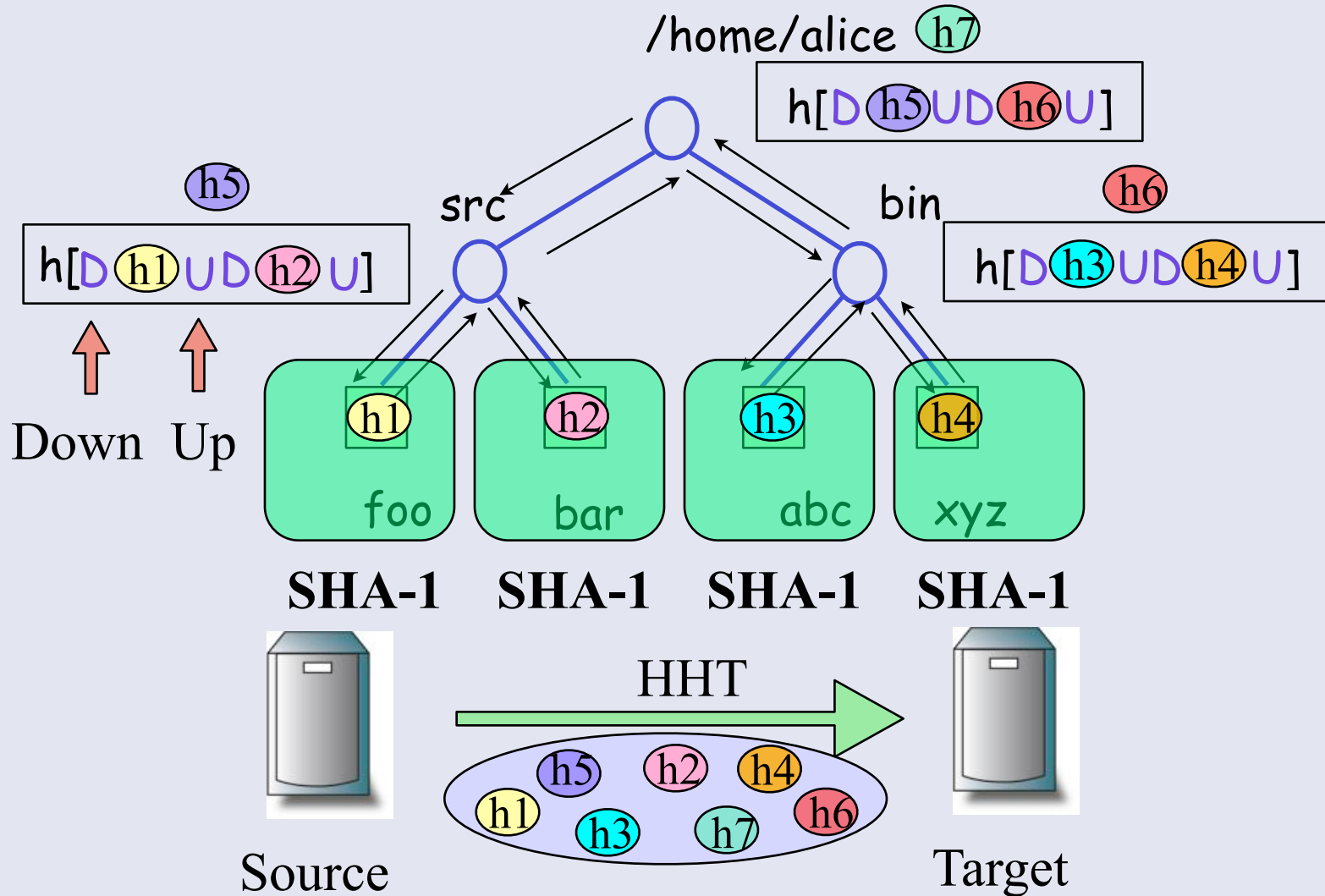


# Phase IV: Individual bits (Unmatched blocks)

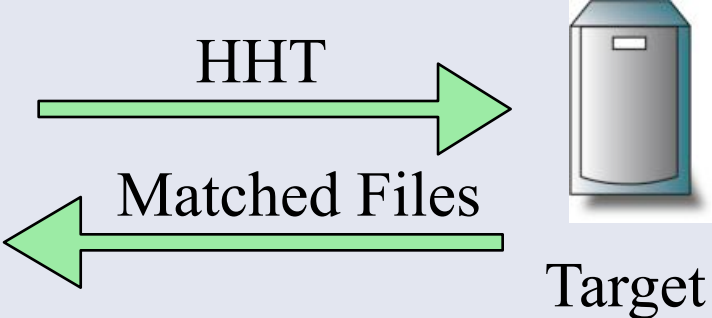
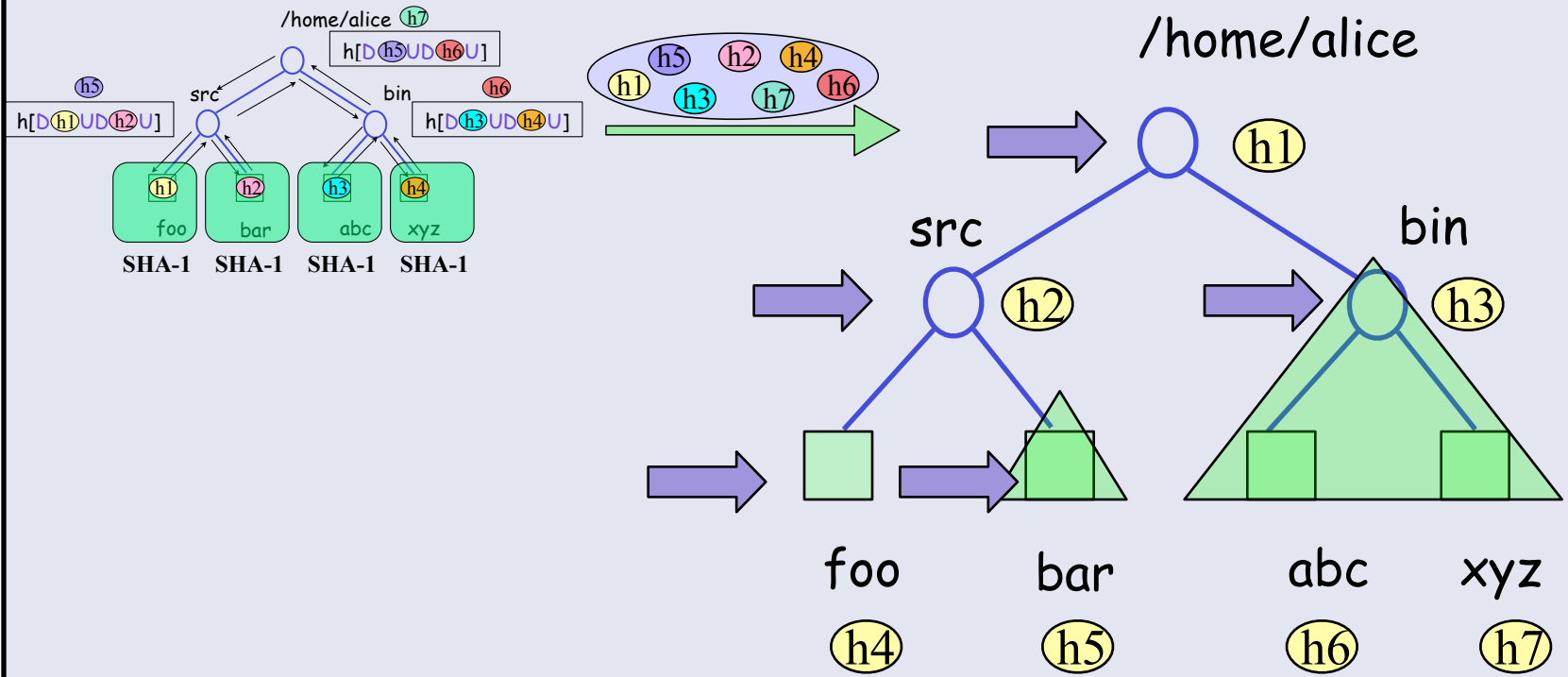


# Hierarchical Hash Tree: Source [Phase I]

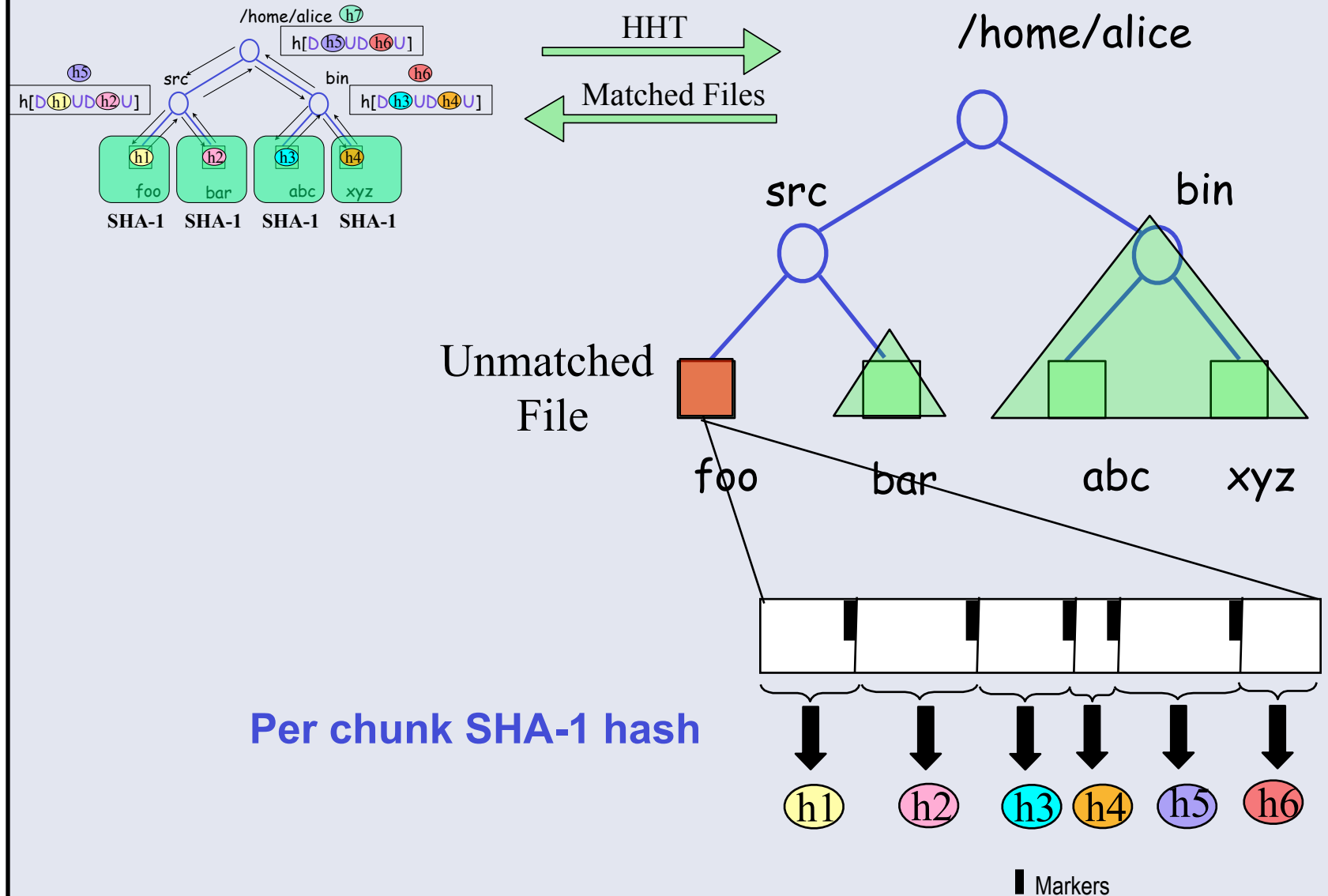
Aim: tolerate rename/move/copy operations



# Hierarchical Hash Tree: Target [Phase I]

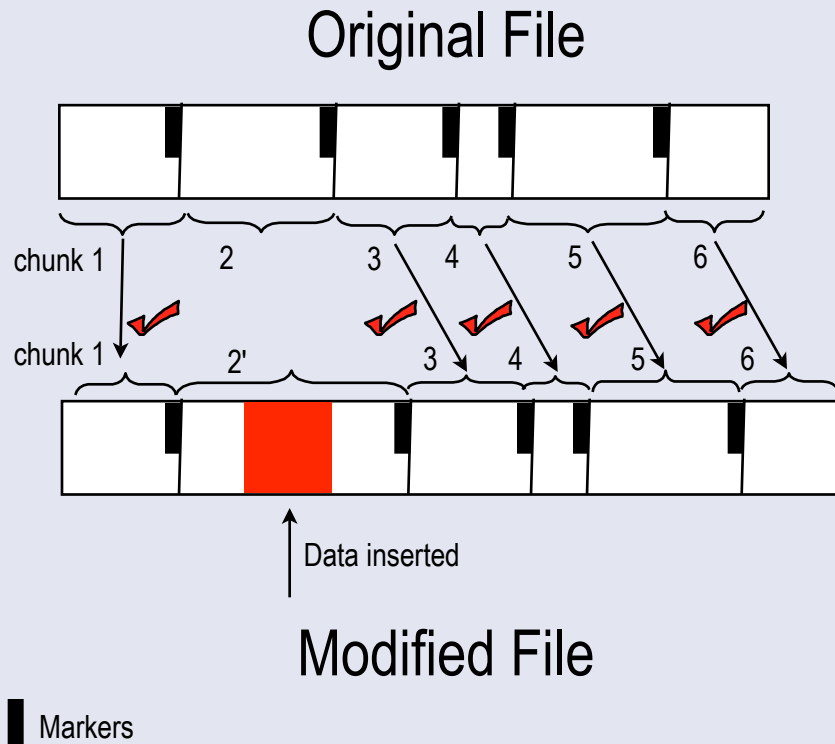


# Target [Phase II]



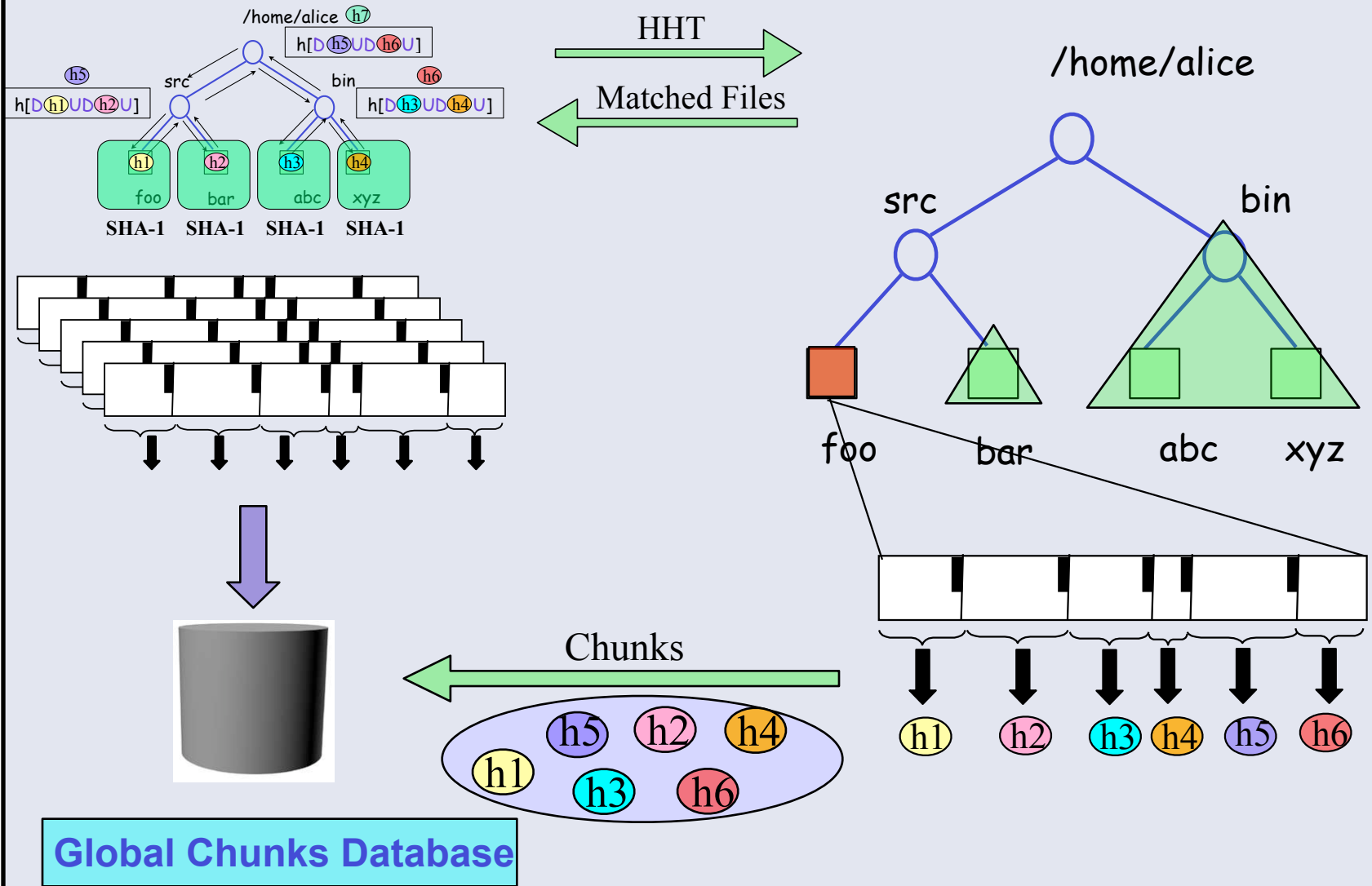
# Content-defined Chunking (CDC): Overview

Aim: tolerate insert/delete/copy operations

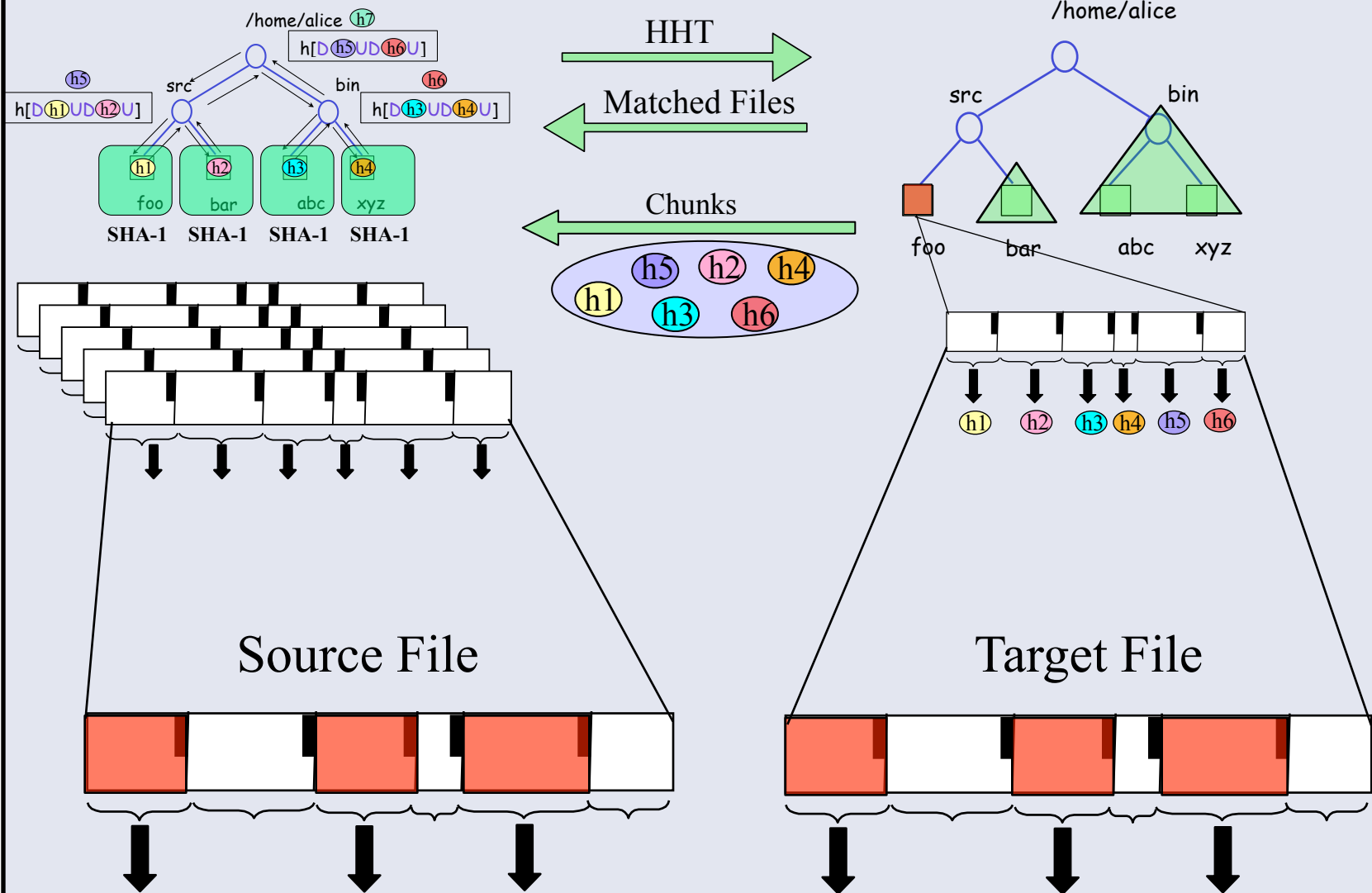


- Divide a file into variable-sized blocks (called chunks)
- Use Rabin fingerprint to compute block boundaries
- Compute SHA-1 hash of each chunk

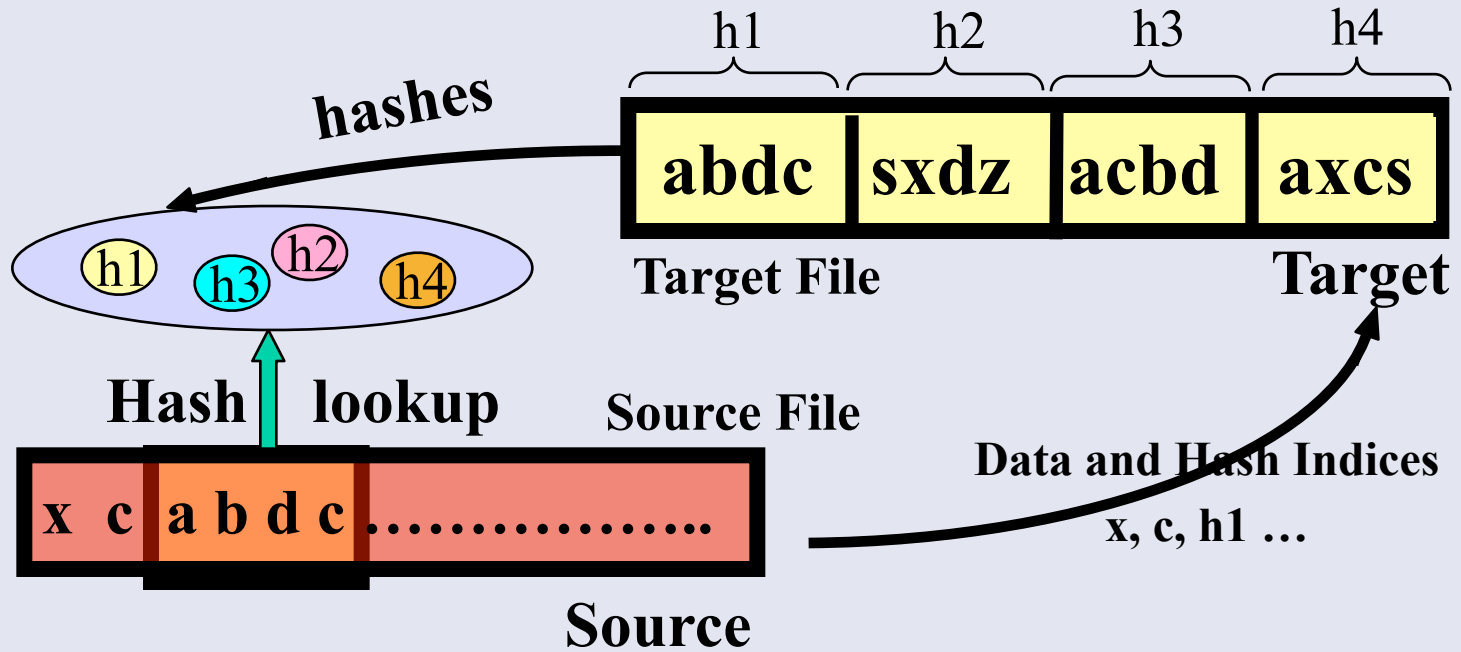
# CDC: Source [Phase II]



# Sliding Block on Holes [Phase III]



# Sliding Block (Rsync): [Phase III]



# Bloom Filters for Similarity Detection

- A novel similarity detection technique using **content-defined chunking** and **Bloom filters** to determine similar files
- **Key Features**
  - **Low Space overhead (Compact Representation)**
    - Incurs only about **0.02-0.4% extra bytes** per file
  - **Low Time overhead (Quick Matching)**
    - Document similarity using **bit-wise AND** of their feature representations

# Talk Outline

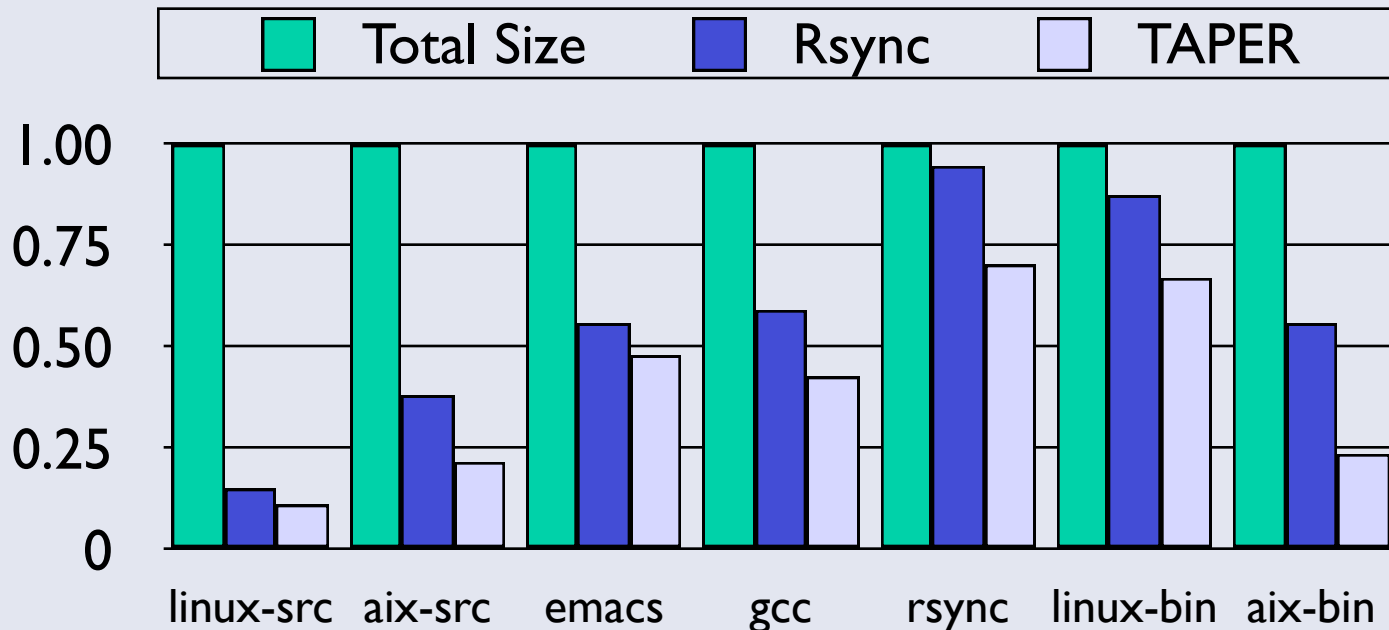
- Motivation
- TAPER Algorithm
- Experimental Evaluation
- Conclusions

# Experimental Evaluation

- Implementation
  - Prototype written in C and Perl
  - Berkeley DB database for global HHT/chunk database
- Datasets
  - Software Distribution sources
    - Linux-src, AIX-src, Emacs, GCC, Rsync
  - Object Binaries
    - Linux-bin, AIX-bin (/usr, /etc, /var, /sbin)
  - Web Data
    - CNN, Yahoo, IBM, Google Groups

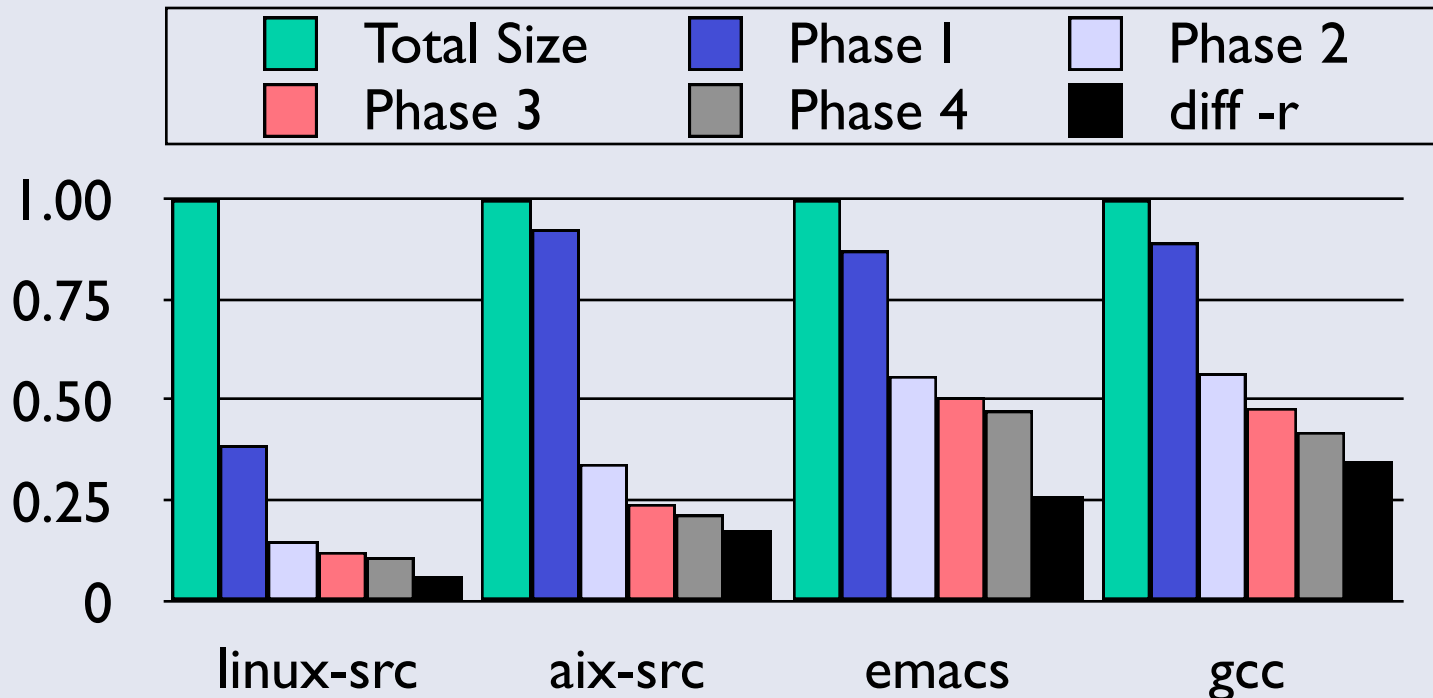
# Comparing Bandwidth Efficiency

- TAPER's improvement over Rsync
  - 15%-43% (software sources)
  - 24%-58% (object binaries)
  - 10%-71% (web datasets)



# Evaluating TAPER Phases

- 52%-88% overall data reduction
- 0.5%-1.4% metadata overhead
- Final data transmitted comparable to 'diff -r'



# Conclusions

- Problem: Efficient Replica Synchronization
  - Minimize network bandwidth, master overheads
  - No prior assumption of replica state
- Our Solution: TAPER
  - Bandwidth-efficient
  - Scalable
  - Fast Matching
  - Content-based
  - Overall 15-71% bandwidth savings over Rsync

**For more information:**

<http://www.cs.utexas.edu/users/nav/TAPER>

# Comparing Computational Overhead

- Emacs dataset: [User + System CPU time]
  - Tar+gzip: 11 secs
  - Rsync: 15.8 secs
  - TAPER: 28 secs
    - Global (HHT+CDC): 14.5 secs
    - Target-specific computation: 13.5s comparable to rsync
- For synchronizing 10 replicas:
  - Rsync:  $15.8 * 10 = 158$  seconds
  - TAPER:  $14.5 + 13.5 * 10 = 150$  seconds