MIGRATORY COMPRESSION

Coarse-grained Data Reordering to Improve Compressibility

Xing Lin*, Guanlin Lu, Fred Douglis, Philip Shilane, Grant Wallace

**University of Utah*

EMC Corporation
Data Protection and Availability Division

THE UNIVERSITY OF UTAH®

EMC²

Overview

- Compression: finds redundancy among strings within a certain distance (window size)
  - Problem: window sizes are small, and similarity across a larger distance will not be identified
- Migratory compression (MC): coarse-grained reorganization to group similar blocks to improve compressibility
  - A generic pre-processing stage for standard compressors
  - In many cases, improve **both** compressibility and throughput
  - Effective for improving compression for archival storage

Background

- Compression Factor (CF) = ratio of original / compressed
- Throughput = original size / (de)compression time

Compressor	MAX Window size	Techniques
gzip	64 KB	LZ; Huffman coding
bzip2	900 KB	Run-length encoding; Burrows-Wheeler Transform; Huffman coding
7z	1 GB	LZ; Markov chain-based range coder

Background

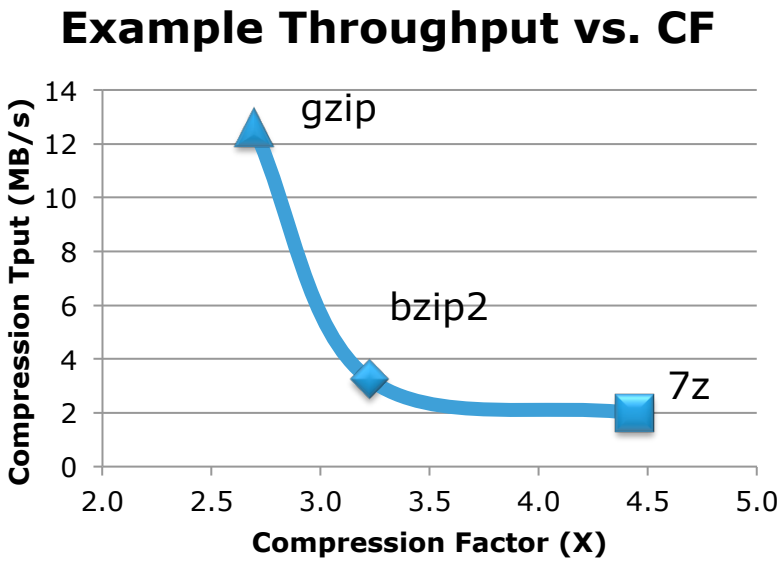
- Compression Factor (CF) = ratio of original / compressed
- Throughput = original size / (de)compression time

Compressor	MAX Window size	Techniques
gzip	64 KB	LZ; Huffman coding
bzip2	900 KB	Run-length encoding; Burrows-Wheeler Transform; Huffman coding
7z	1 GB	LZ; Markov chain-based range coder

Background

- Compression Factor (CF) = ratio of original / compressed
- Throughput = original size / (de)compression time

Compressor	MAX Window size	Techniques
gzip	64 KB	LZ; Huffman coding
bzip2	900 KB	Run-length encoding; Burrows-Wheeler Transform; Huffman coding
7z	1 GB	LZ; Markov chain-based range coder



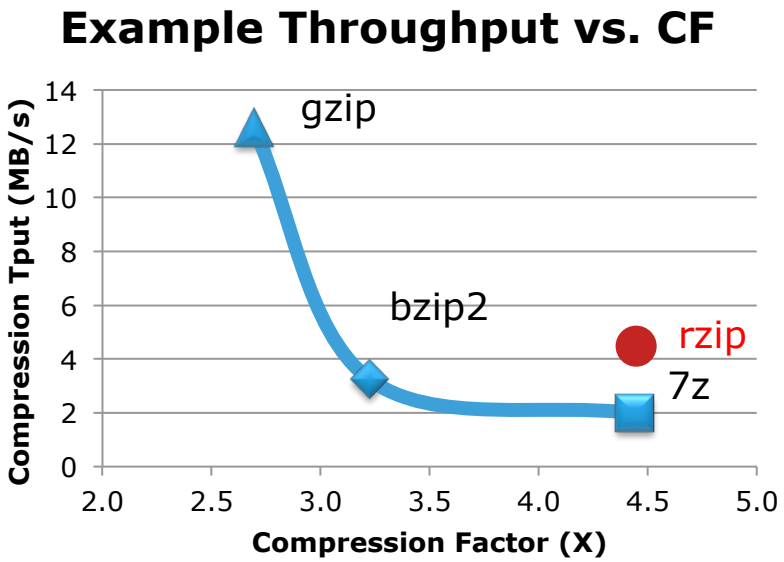
The larger the window,
the **better** the compression
but **slower**

Background

- Compression Factor (CF) = ratio of original / compressed
- Throughput = original size / (de)compression time

Compressor	MAX Window size	Techniques
gzip	64 KB	LZ; Huffman coding
bzip2	900 KB	Run-length encoding; Burrows-Wheeler Transform; Huffman coding
7z	1 GB	LZ; Markov chain-based range coder

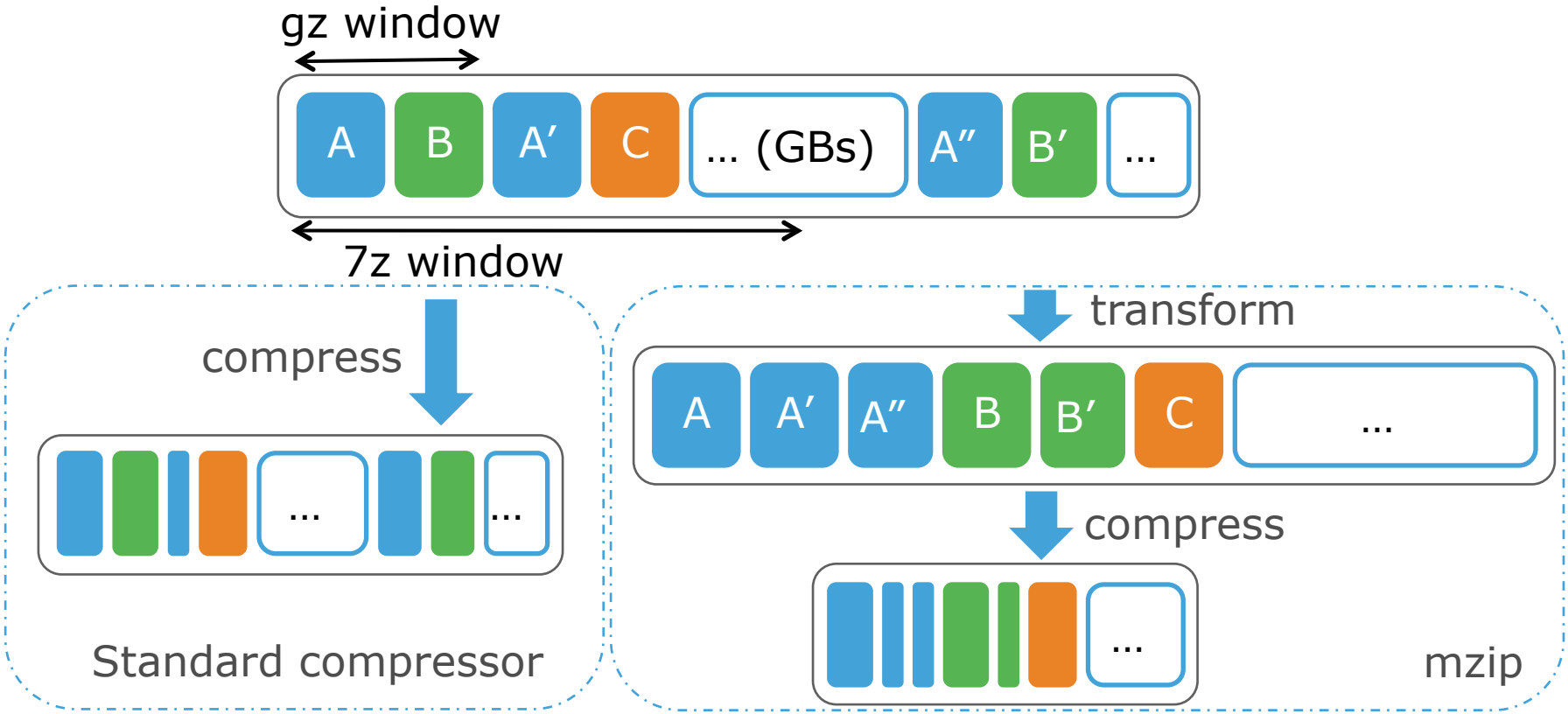
rzip (from rsync):
intra-file deduplication;
bzip2 the remainder



The larger the window,
the **better** the compression
but **slower**

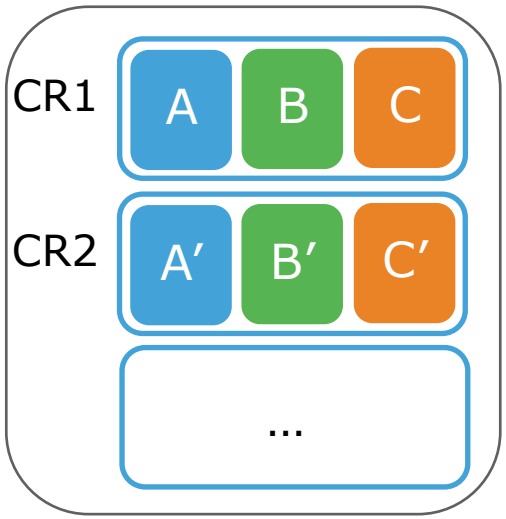
Motivation

- Compress a single, large file
  - Traditional compressors are unable to exploit redundancy across a large range of data (e.g., many GB)



Motivation

- Better compression for long-term retention
  - Data migration from backup to archive tier
  - *Observation:* similar blocks could be scattered across "compression regions" (CRs) in the backup tier

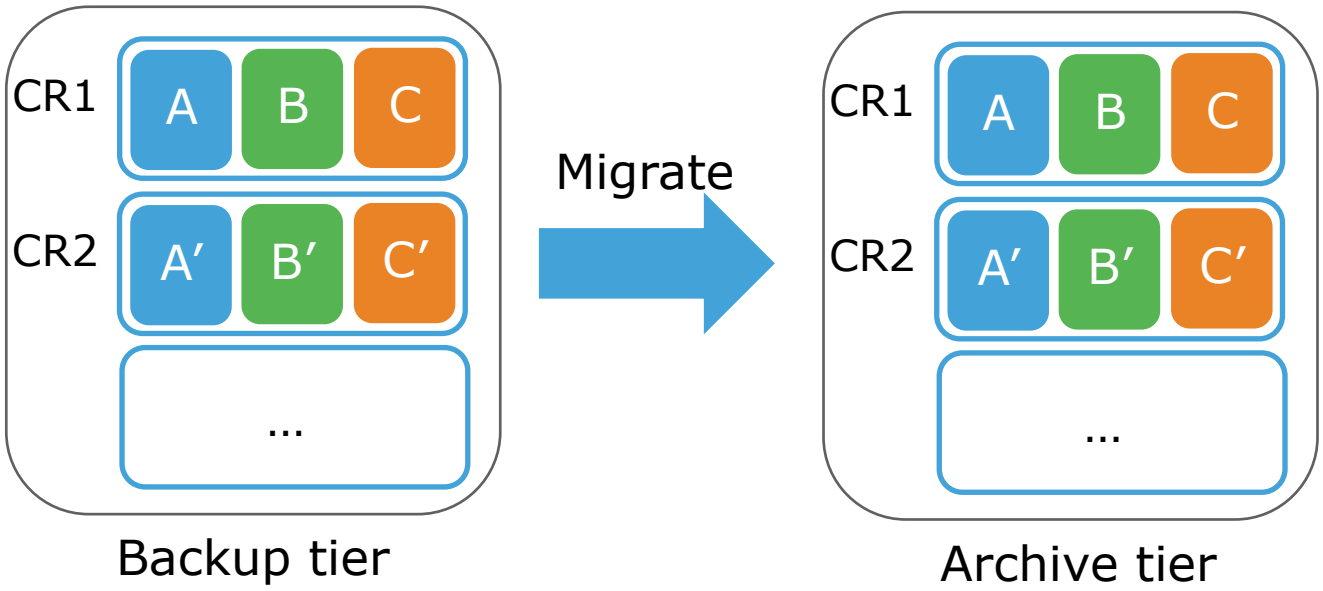


Backup tier

CR: unit of data that is compressed together (e.g. 128 KB)

Motivation

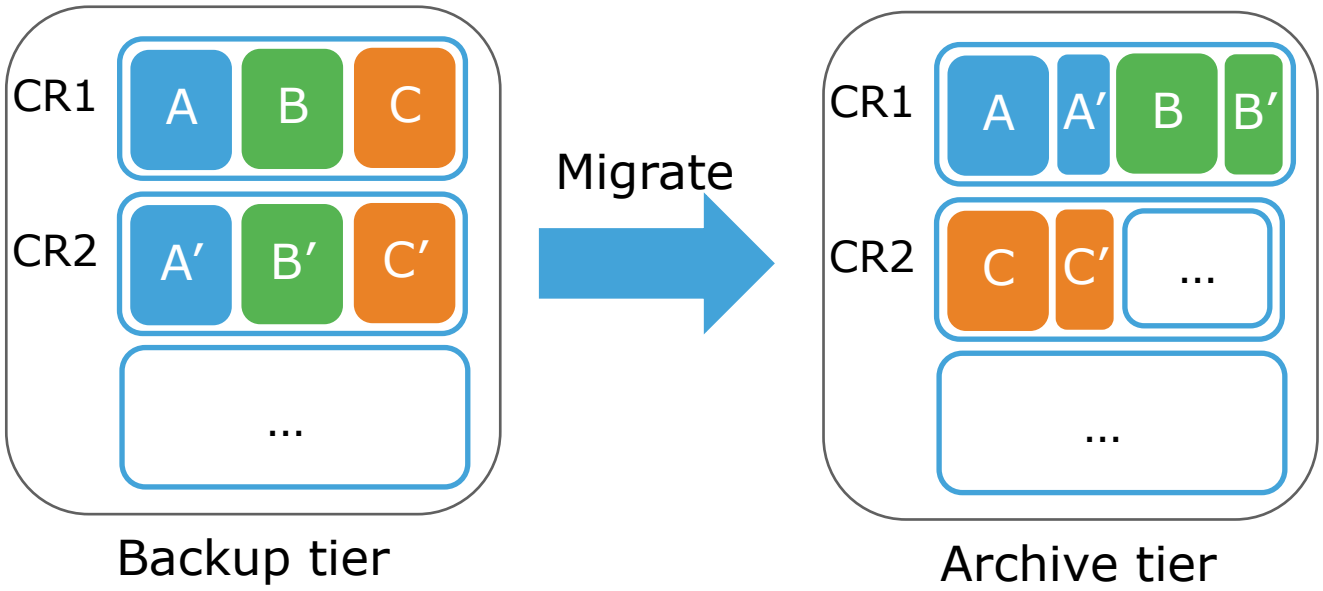
- Better compression for long-term retention
  - Data migration from backup to archive tier
  - *Observation:* similar blocks could be scattered across "compression regions" (CRs) in the backup tier



Current practice

Motivation

- Better compression for long-term retention
  - Data migration from backup to archive tier
  - *Observation:* similar blocks could be scattered across "compression regions" (CRs) in the backup tier
  - *Proposal:* fill each CR with similar data

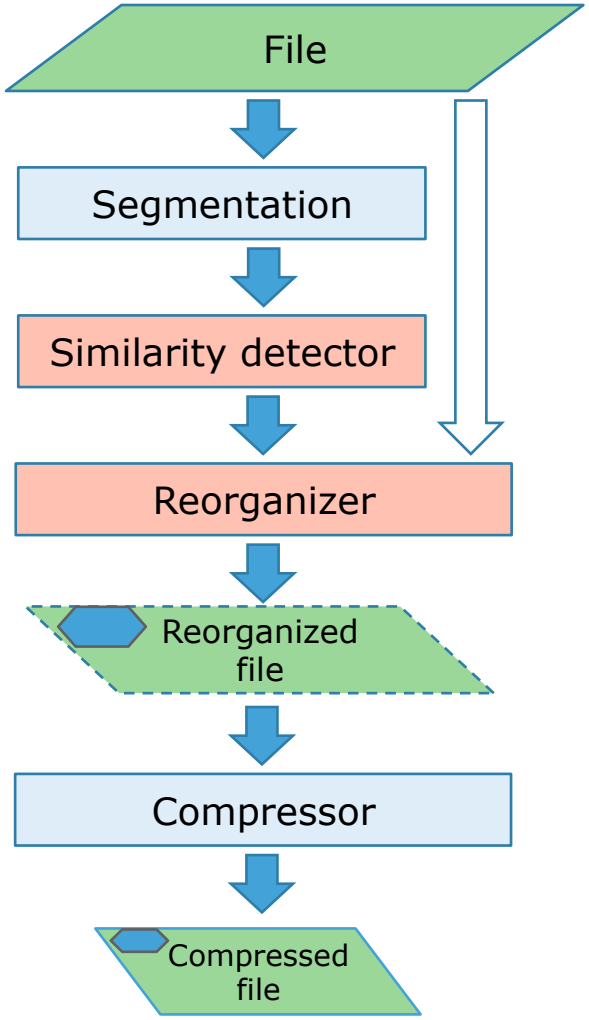


Migratory compression

Recap

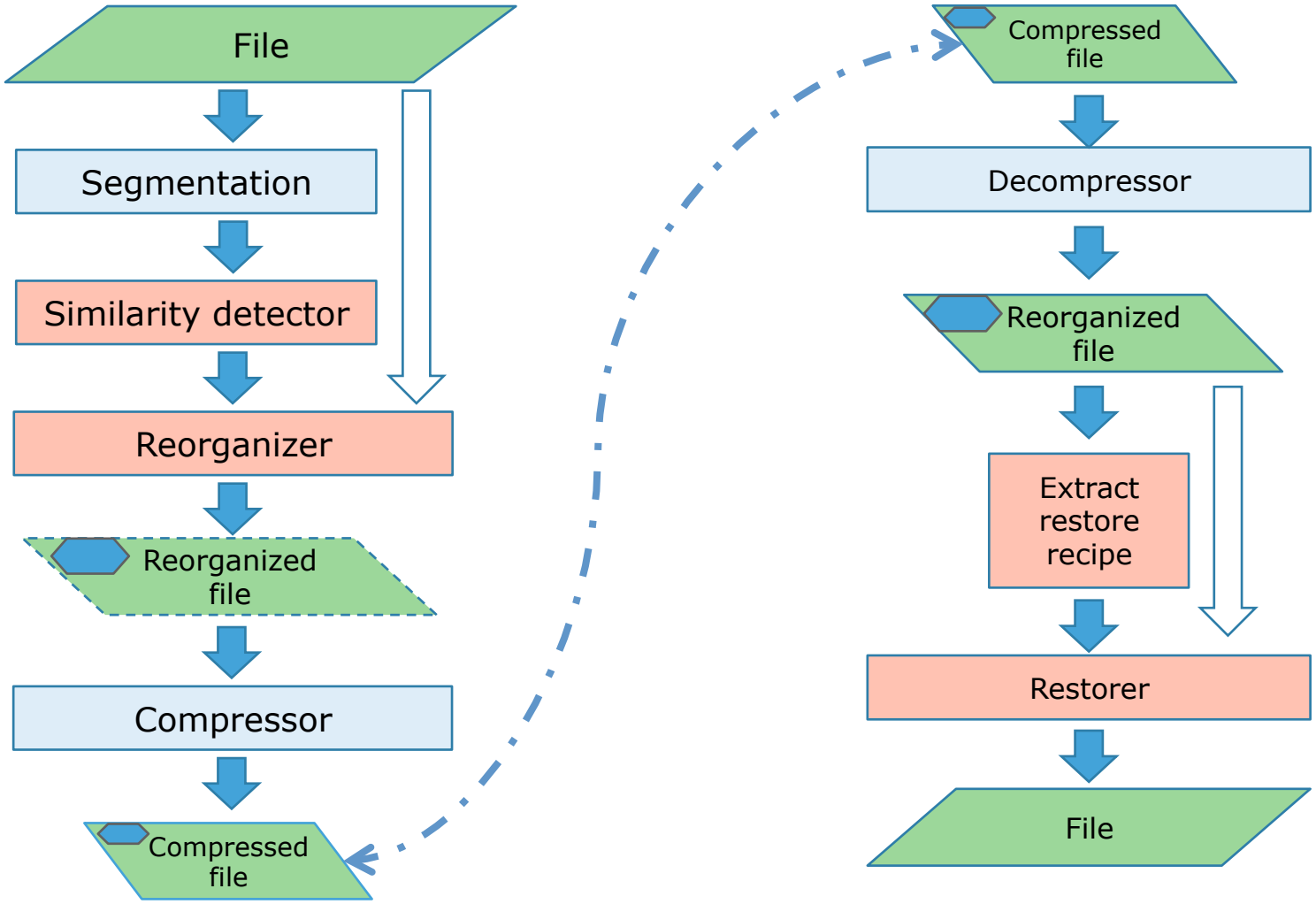
- Migratory compression
  - Idea: improve compression by grouping similar blocks
  - Use cases
    - mzip: compress a single, large file
    - Data migration for archival storage
- Considerations
    - How to identify similar blocks?
    - How to reorganize blocks efficiently?
    - Other tradeoffs (refer to the paper)

mzip Workflow

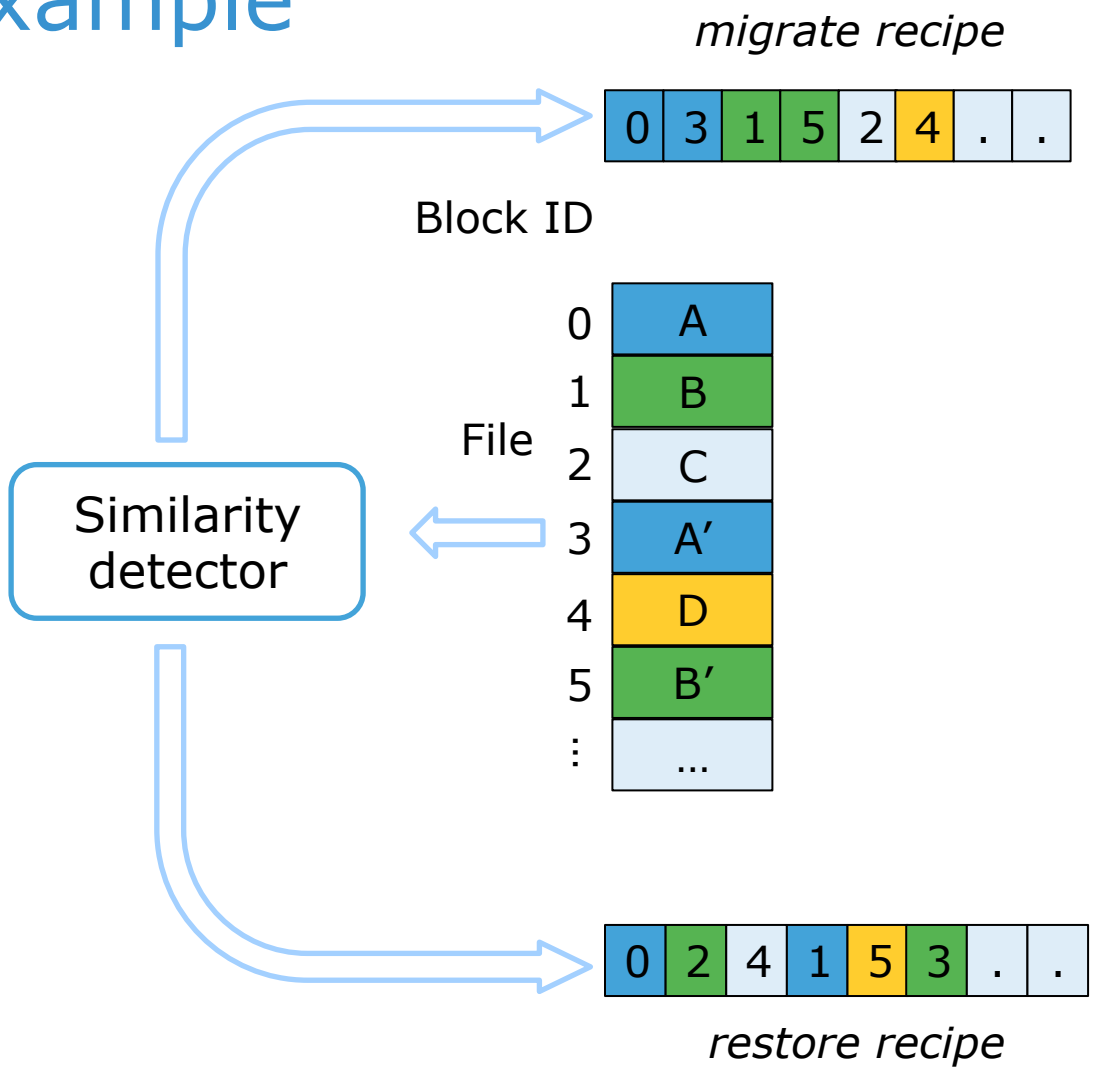


- ***Segmentation:*** partition into blocks, calculate similarity "features"
- ***Similarity detector:*** group by content and identify duplicate and similar blocks; output *migrate and restore recipe*
- ***Reorganizer:*** rearrange the input file
  - Reorganized file may exist only as a pipe into compressor

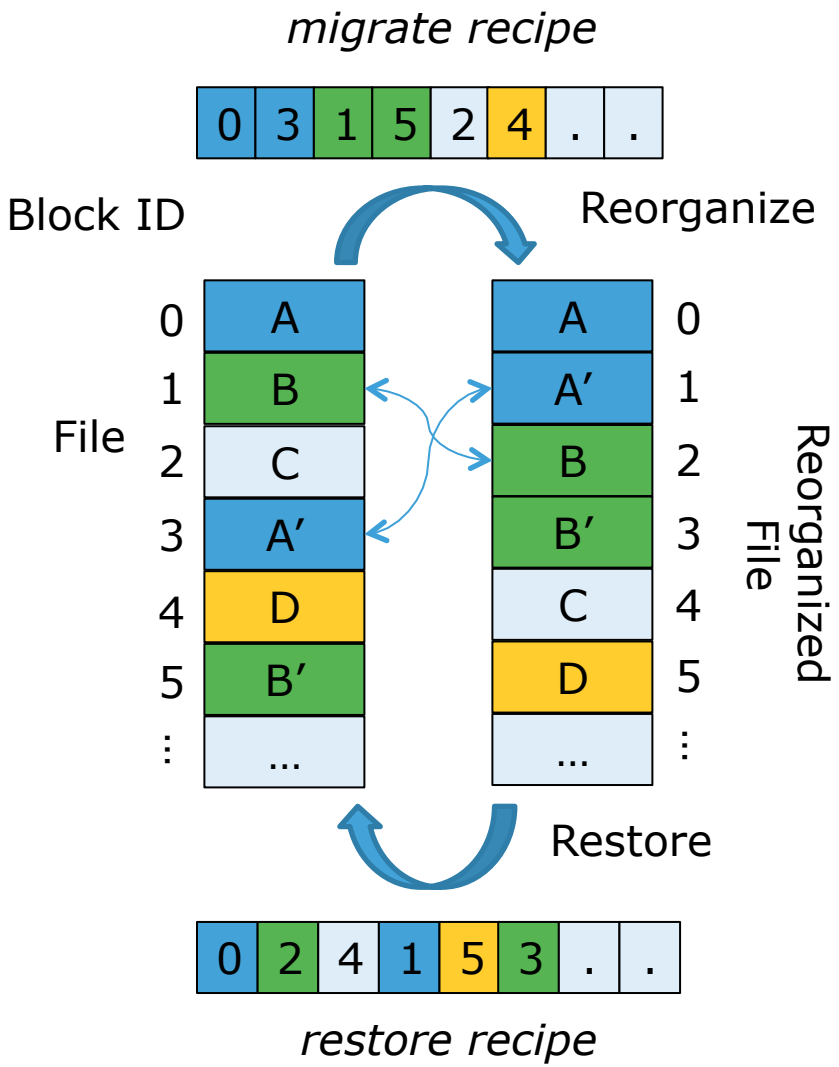
mzip Workflow



mzip Example



mzip Example



Similarity Detection

Similarity feature (based on [Broder 1997])

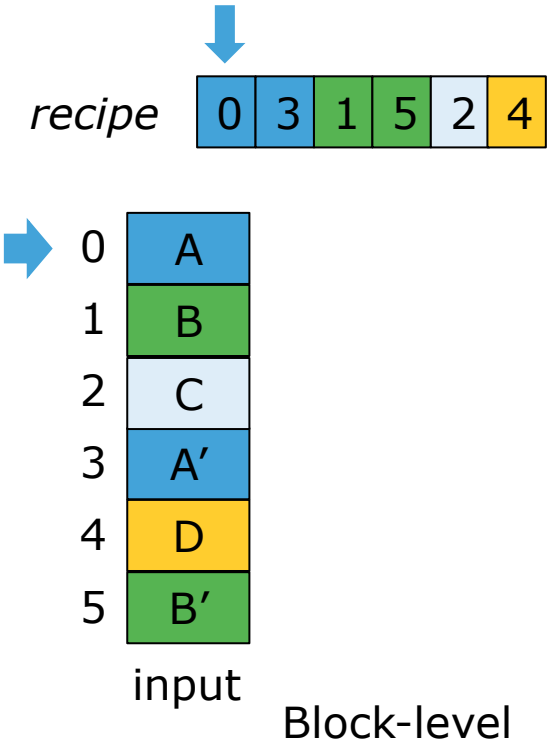
- A strong hash for duplication detection
- Features: weak hashes provide hints about similarity among blocks
  - Two blocks are similar when they share values for some weak hashes

Mature technique

- REBL: Redundancy Elimination at Block Level [Kulkarni et al. 2004]
- WAN Optimized Replication [Shilane et al. 2012]

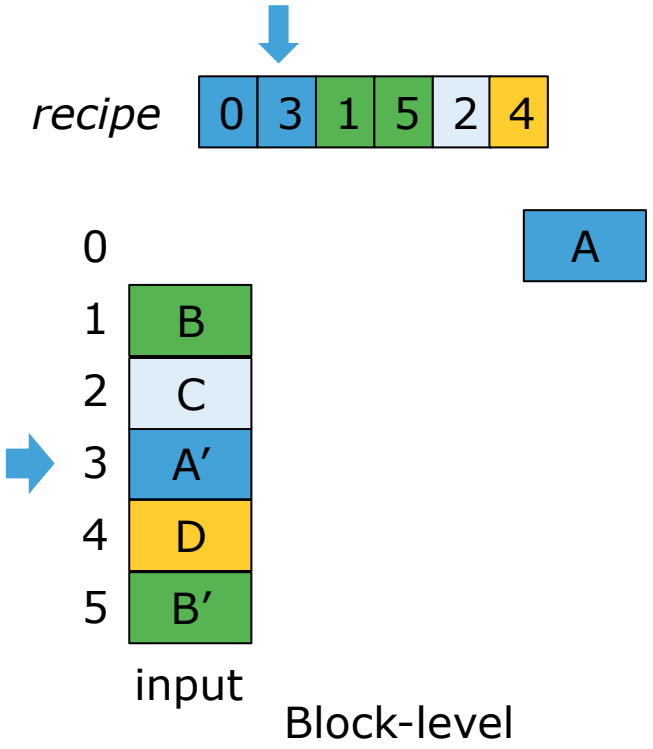
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level



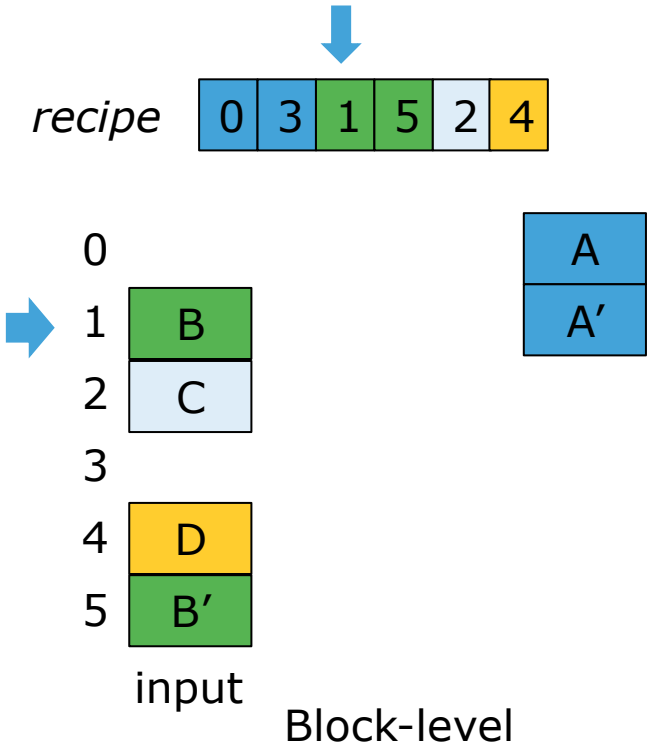
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level



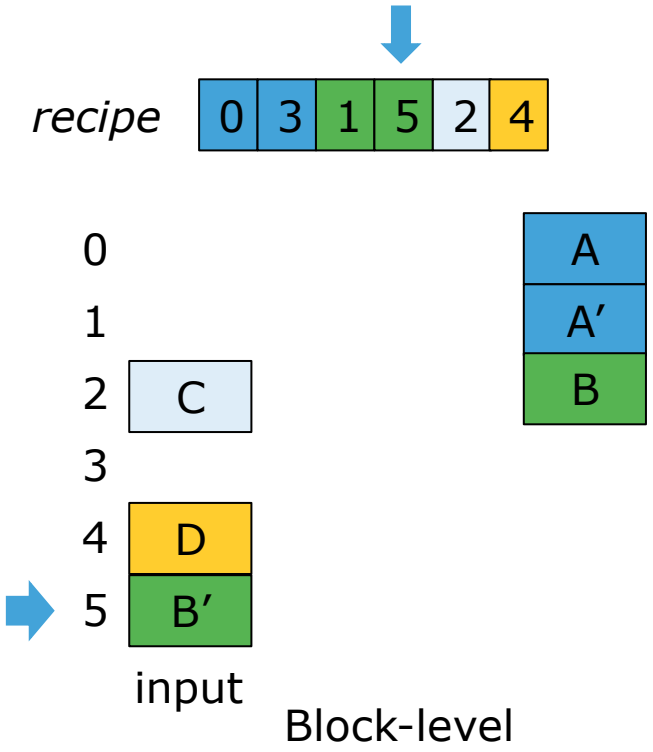
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level



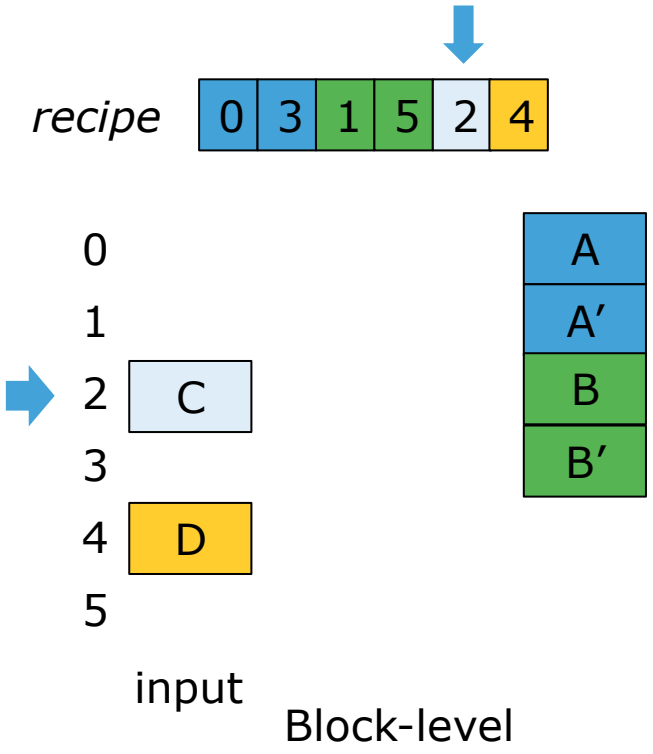
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level



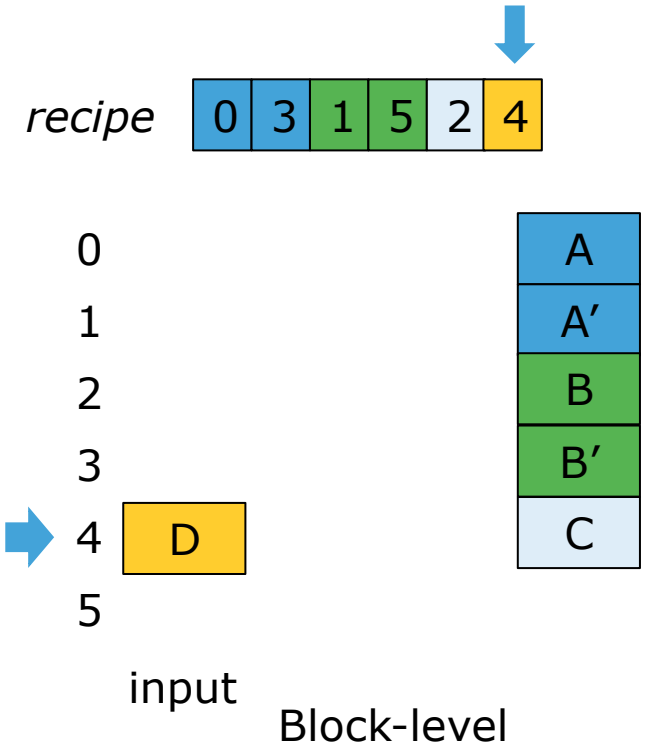
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level



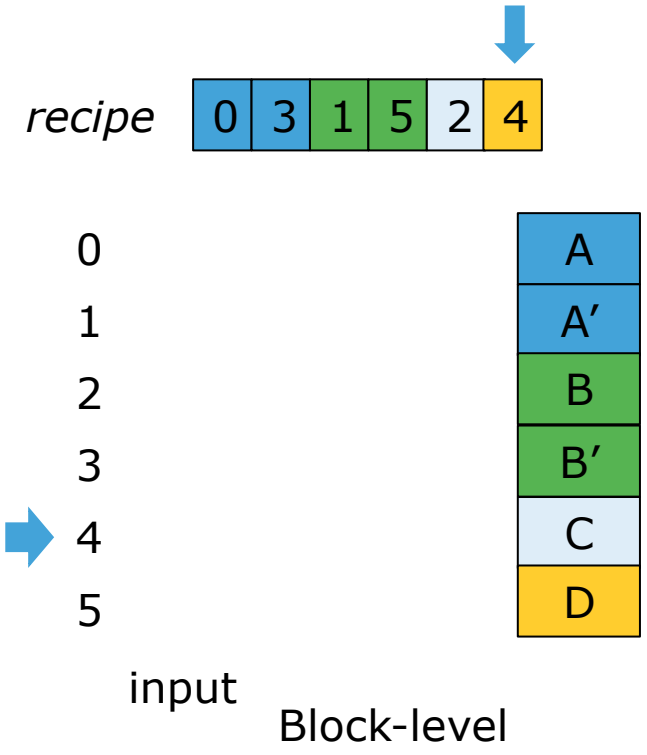
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level



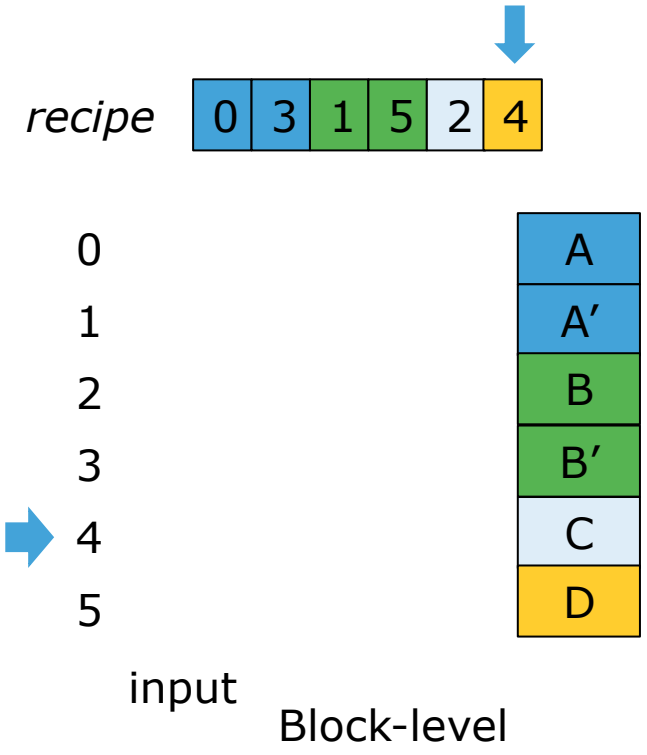
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level



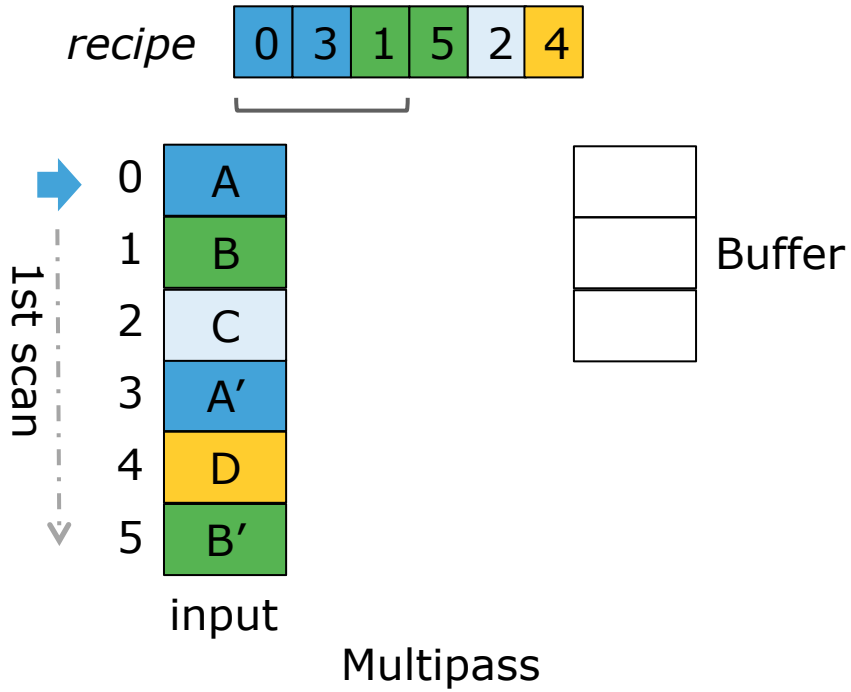
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level
    - Random IOs
    - Fine for memory and SSD



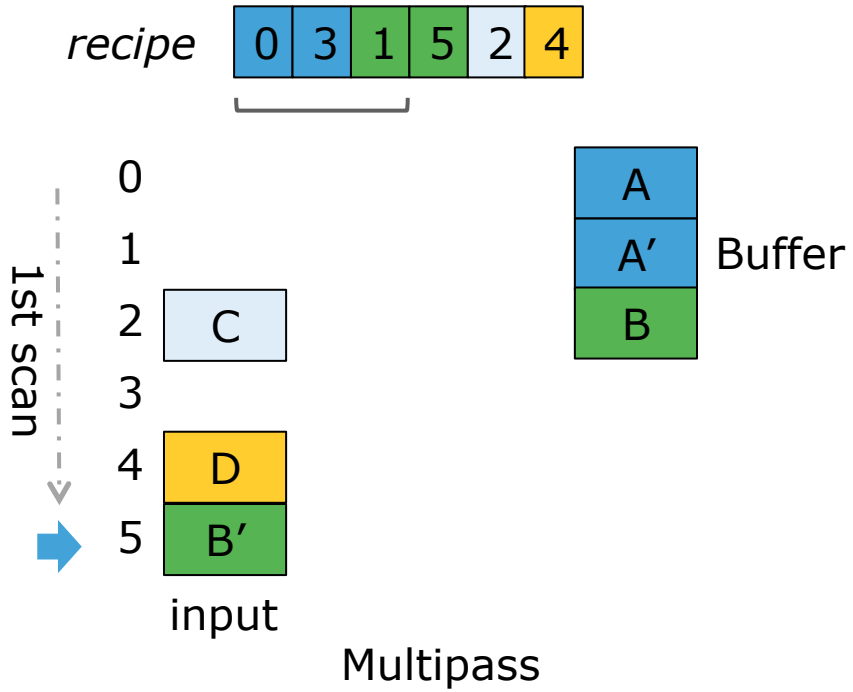
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level
    - Random IOs
    - Fine for memory and SSD
  - Multipass (HDD)



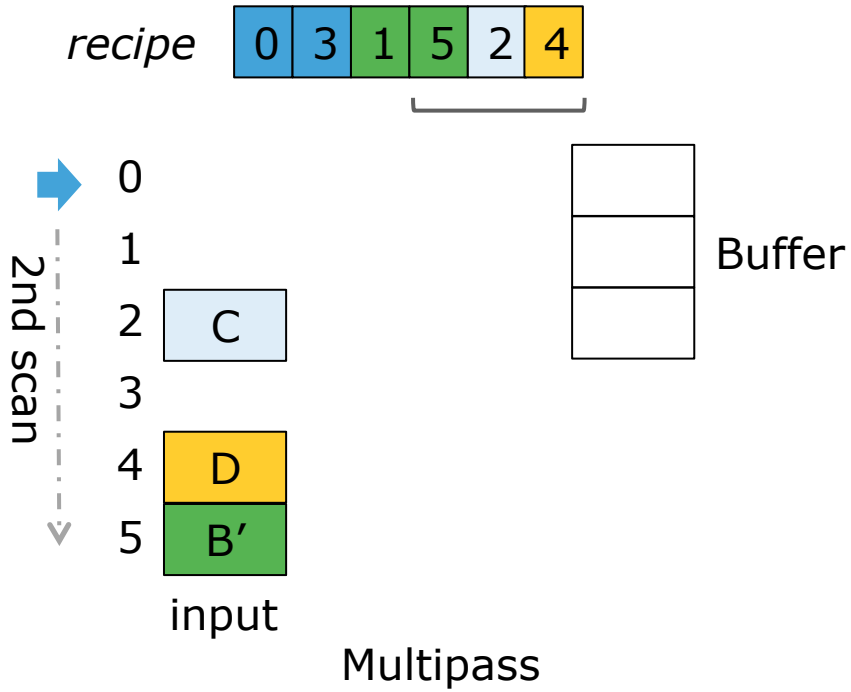
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level
    - Random IOs
    - Fine for memory and SSD
  - Multipass (HDD)



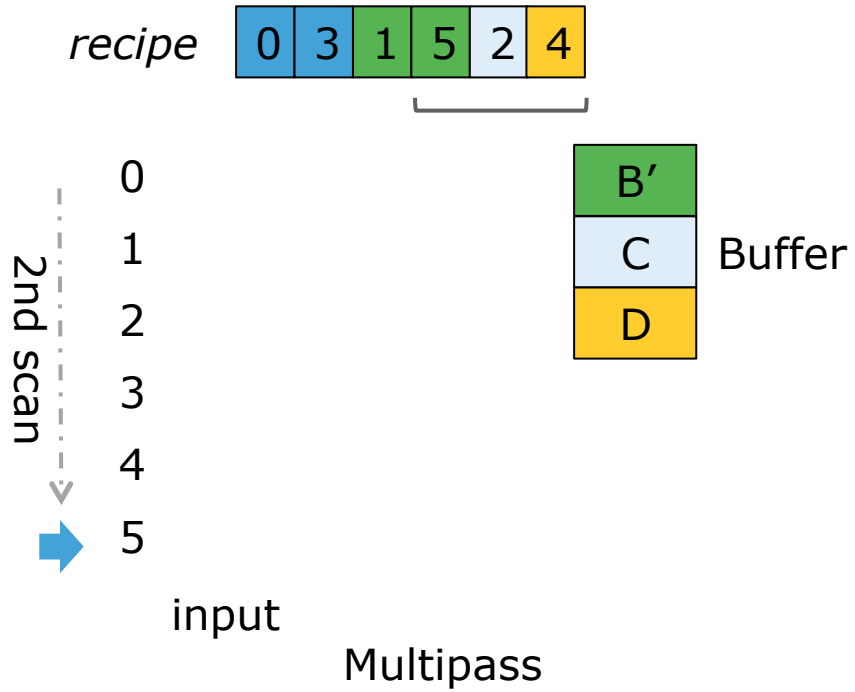
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level
    - Random IOs
    - Fine for memory and SSD
  - Multipass (HDD)



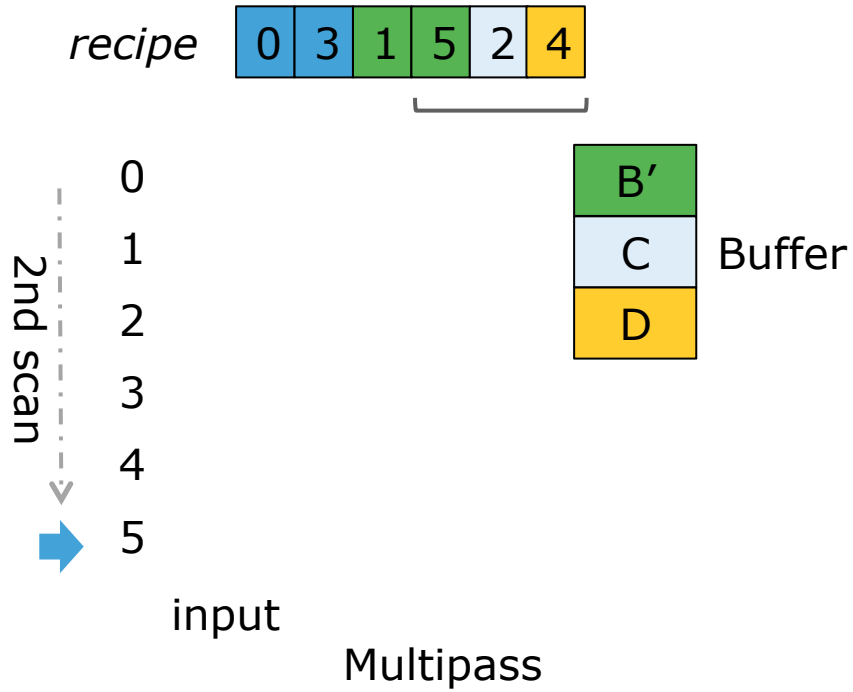
Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level
    - Random IOs
    - Fine for memory and SSD
  - Multipass (HDD)



Data Reorganization

- Re-arrange the input file, according to a recipe
  - Reorganizer & Restorer
- Options
  - Block-level
    - Random IOs
    - Fine for memory and SSD
  - Multipass (HDD)
    - Convert random IOs into sequential scans



Evaluation

- mzip:
  - How much can compression be improved?
  - What is the complexity (runtime overhead)?
  - How does SSD or HDD affect data reorganization performance? For HDD, does multi-pass help?
  - How does MC perform, compared with delta compression? (refer to our paper)
  - What are the configuration options for MC? (refer to our paper)
- Data migration for archival storage
  - Compression and performance

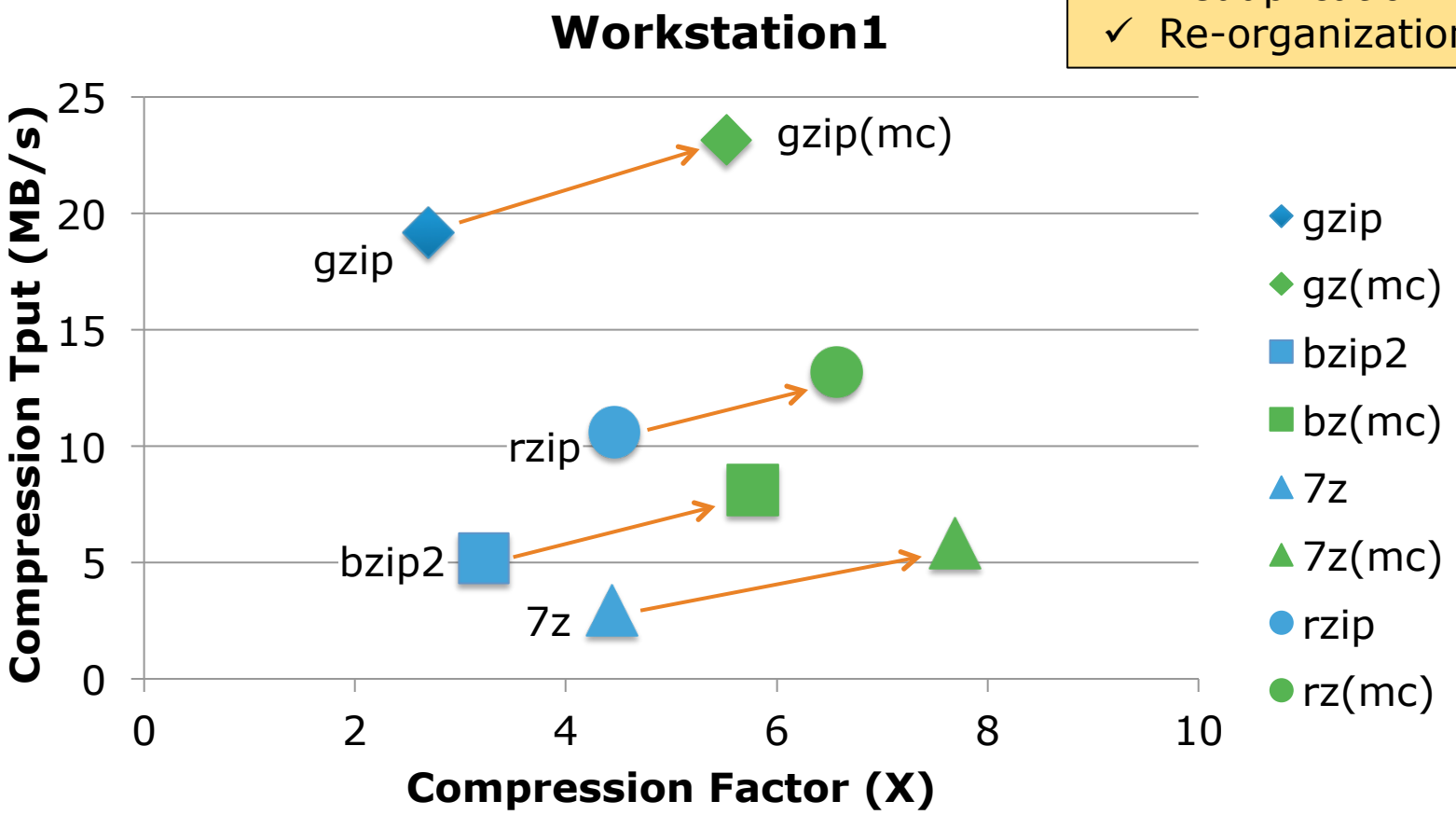
Datasets

Type	Name	Size (GB)	Dedupe Factor (X)	Compression Factor (standalone - baseline)			
				gzip	bzip2	7z	rzip
Workstation backup	WS1	17.36	1.69	2.70	3.22	4.44	4.46
Email server backup	EXCHANGE2	17.32	1.02	2.78	3.13	4.75	4.79
VM image	Ubuntu-VM	6.98	1.51	3.90	4.26	6.71	6.69

* Only one dataset for each type is shown here. Refer to our paper for others.

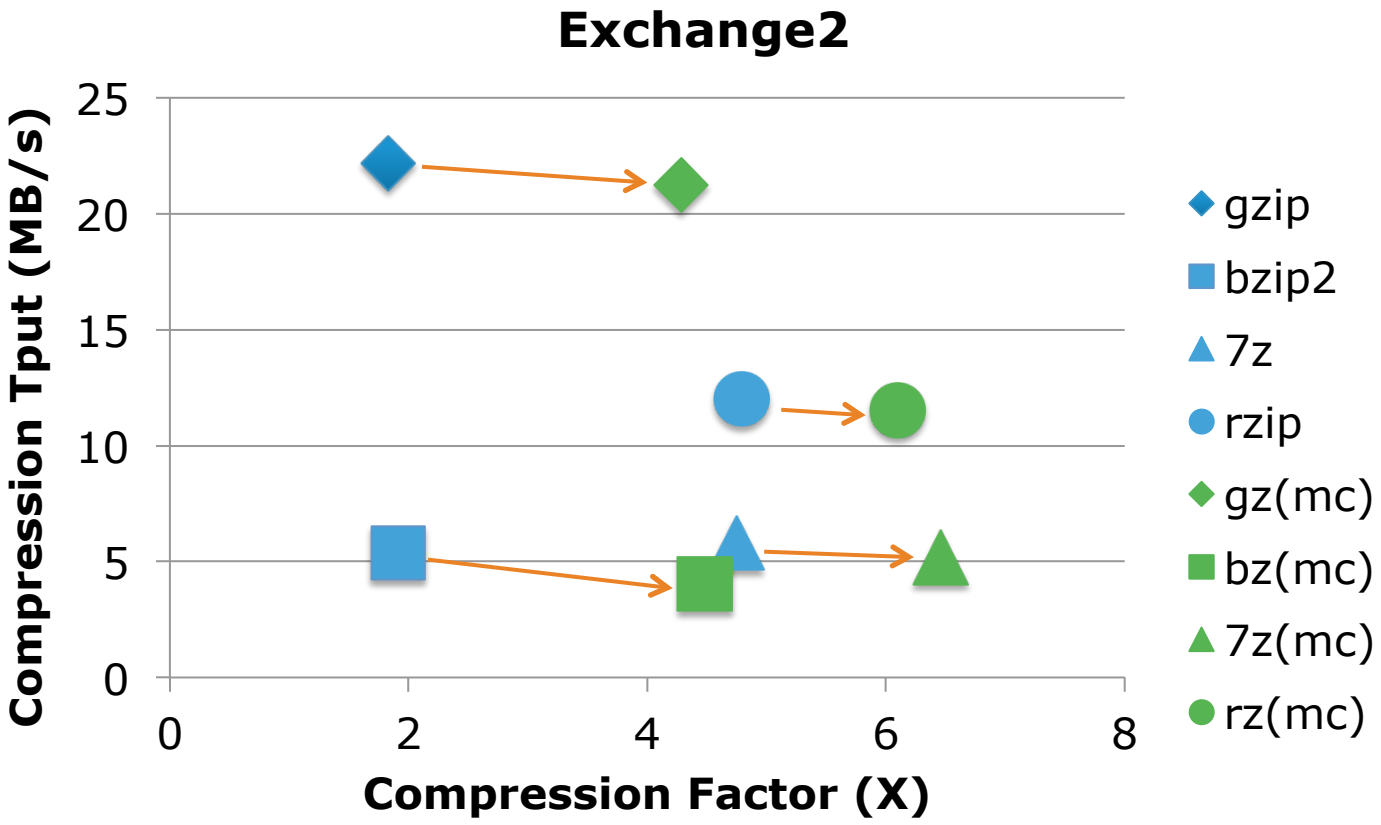
Compression Factor vs. Compression Throughput

MC improves both CF and compression throughput
- ✓ Deduplication
- ✓ Re-organization



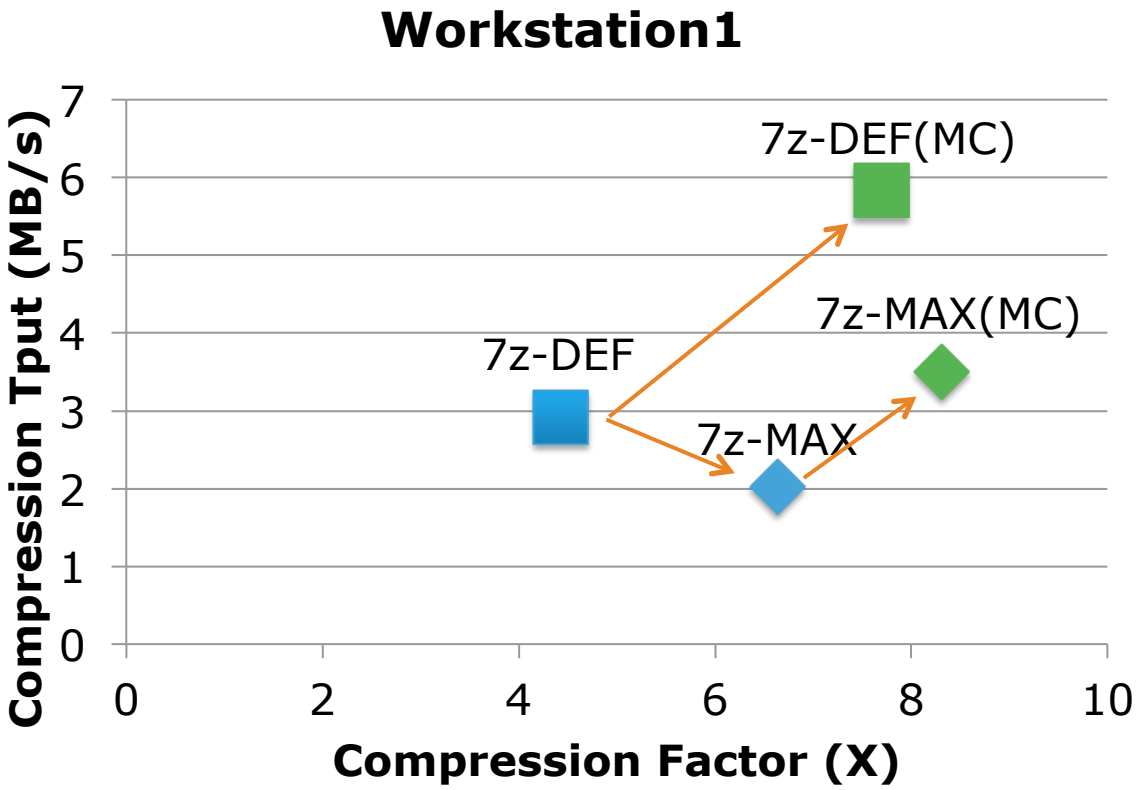
Compression Factor vs. Compression Throughput

MC improves CF but slightly reduces compression throughput



Maximal Compression

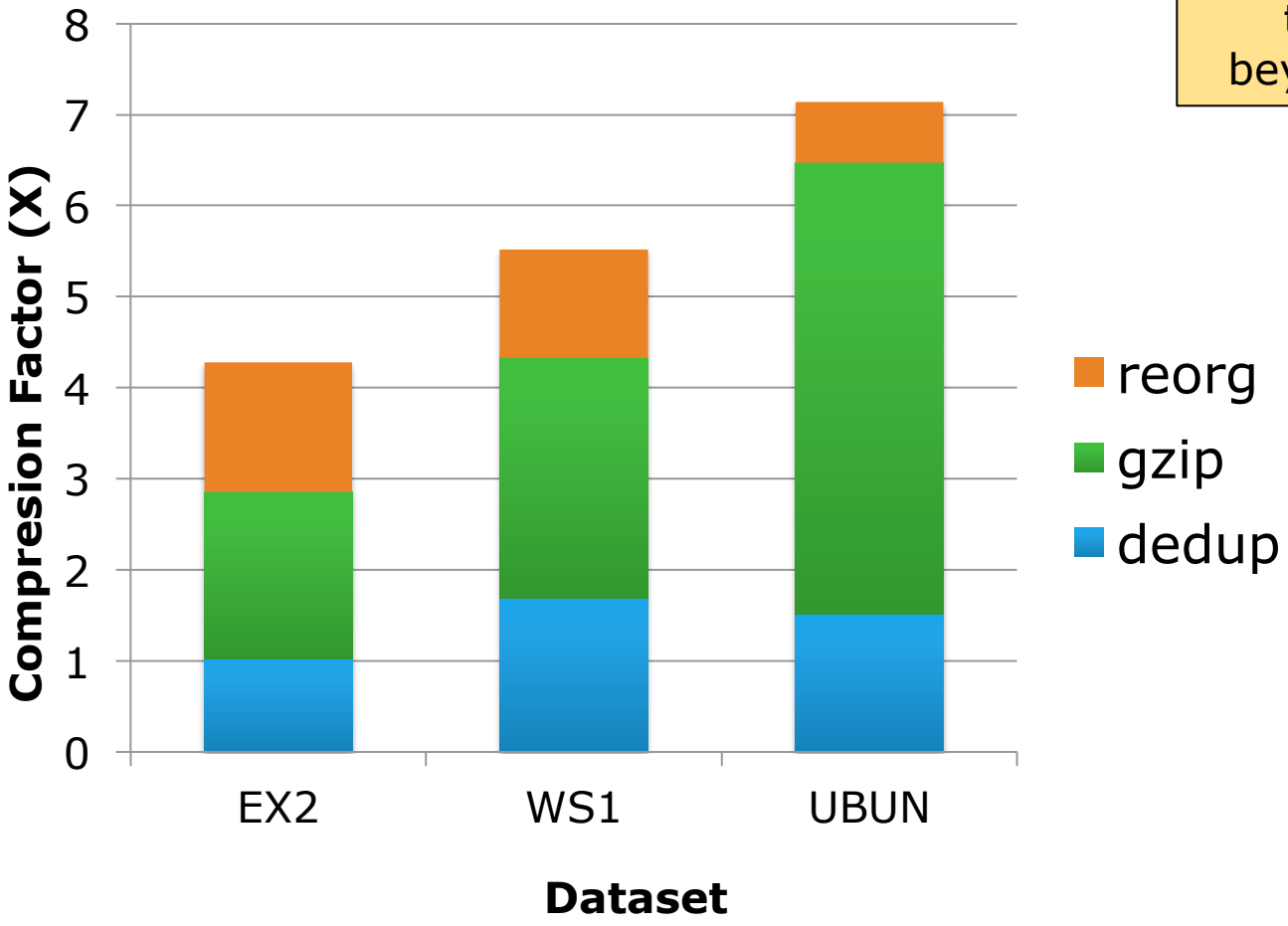
Compressors can be run at various compression levels and window sizes, making a tradeoff between compression and runtime



MC benefits maximal compression

Results for other compressors have same trends

Compression Factor Breakdown

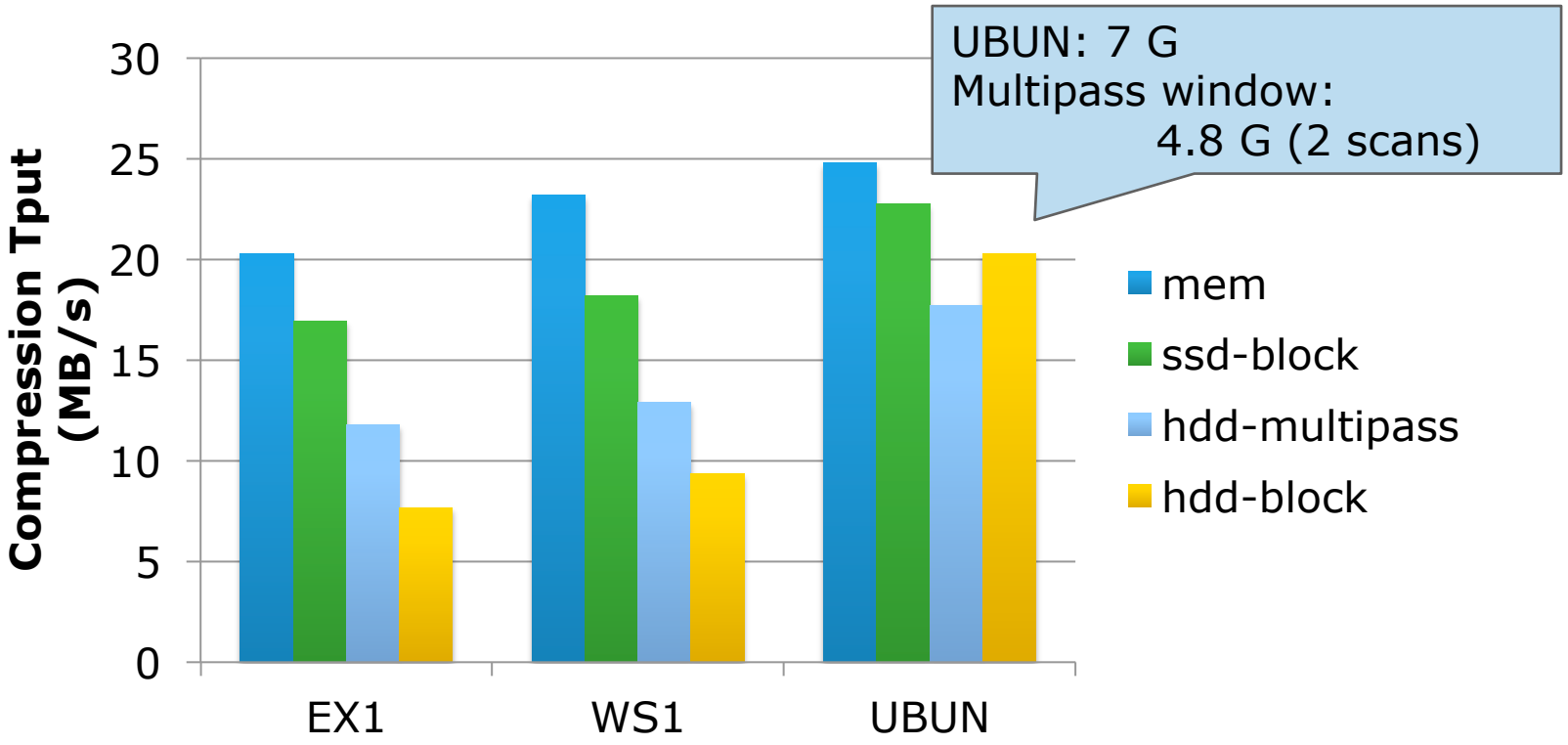


MC improves compression to varying extents beyond dedup and gzip

Data Reorganization

SSD: reasonably good
HDD: multipass helps

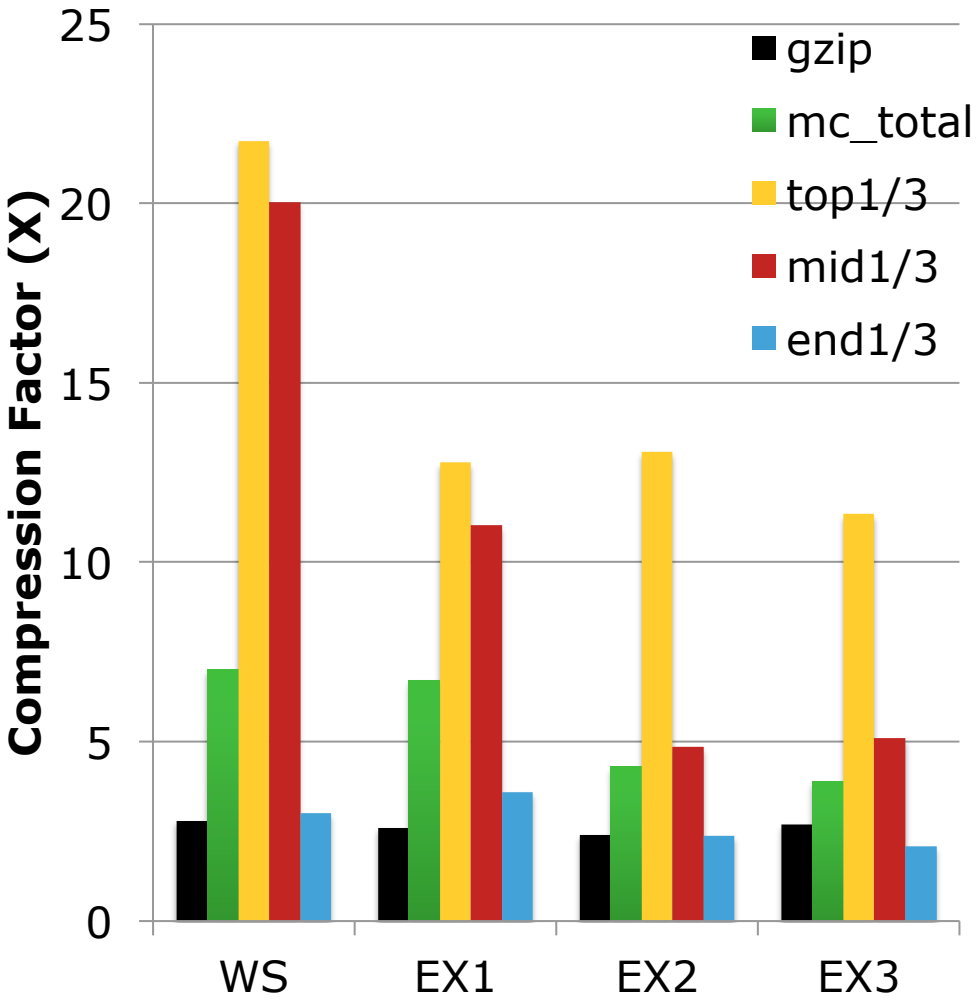
- Configurations
  - mem: input and output stored in tmpfs
  - SSD/HDD: used to store input; output to HDD; 8 GB mem



Data Migration for Archival Storage

- DataDomain Filesystem (DDFS)
  - Backup tier: LZ (fast)
  - Archive tier: recompresses with GZ (25-44% better CF)
- With MC,
  - Identify and sort by cluster sizes of similar blocks
  - Migrate in 3 stages: top third largest clusters, middle third, bottom third (including distinct blocks)
- Datasets
  - WORKSTATIONS: many backups of several workstations
  - Exchange[123]: many backups of 3 exchange email servers

Effect of MC on Archival Migration



- ✓ MC improves CF
  - [44% (EX3), 157% (WS)]
  - Top 2/3 compresses very well

Archival Migration – Costs

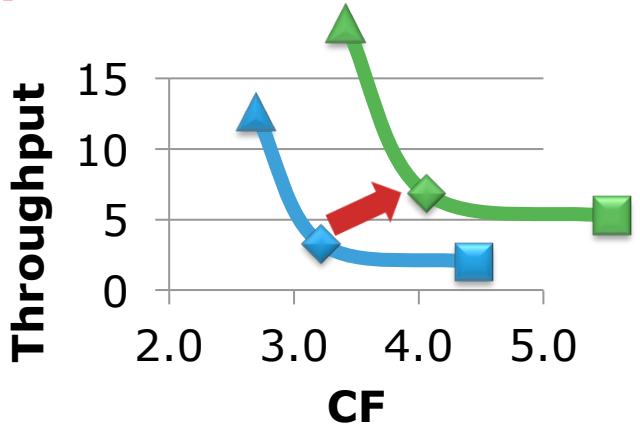
- Migration Runtime
  - Exchange1: 3X longer
- Read Performance
  - As data is scattered in file system, read performance suffers [Lillibridge 2013]
  - entire EXCHANGE1: 1.3 X longer
  - final backup: 7X longer (1.24 X longer if just the top third of similar clusters are reorganized)
- Memory overheads

Related Work

- Improving traditional compressors
  - LZMA algorithm with extra-large window (7z)
  - rzip rolling hash to deduplicate with a large window
  - Burrows-Wheeler Transform to reorder data (bzip2)
- Delta compression (MC is slightly better in CF and throughput)
  - Similar to MC when limited to a single file
  - Bookkeeping complexities in a file system
- Similarity detection
  - WAN Optimized Replication: delta-compress similar chunks for replication
  - REBL: Redundancy Elimination at Block Level (Never applied practically at scale, or in self-contained file)

Conclusions

- Migratory Compression preprocesses data to make it more compressible
  - Identify and cluster similar data
- mzip
  - Improves existing compressors, in both **compressibility** and frequently **runtime**
  - ***Redraw the performance-compression curve!***
- Archival storage
  - MC reduces $/GB further



Thank you!

Questions?