Using Similarity in Content and Access Patterns to Improve Space Efficiency and Performance in Storage Systems

Xing Lin

PhD Defense
July 22, 2015

Photos Stored in Facebook

Reference

"Finding a needle in Haystack: Facebook's photo storage", in USENIX OSDI '10

Photos Stored in Facebook

Reference
"Finding a needle in Haystack: Facebook's photo storage", in USENIX OSDI '10

Photos Stored in Facebook

- Number of photos uploaded every week

Reference
"Finding a needle in Haystack: Facebook's photo storage", in USENIX OSDI '10

Photos Stored in Facebook

- Number of photos uploaded every week

1 billion
(60 terabytes)

Reference
"Finding a needle in Haystack: Facebook's photo storage", in USENIX OSDI '10

Photos Stored in Facebook

- Number of photos uploaded every week

  1 billion
  (60 terabytes)

- Total number of photos stored by 2010

  260 billion
  (20 petabytes)

Reference
"Finding a needle in Haystack: Facebook's photo storage", in USENIX OSDI '10

Amazon S3

Amazon S3 – Two Trillion Objects

by Jeff Barr | on 18 APR 2013 | in Amazon S3 | Permalink | Comments

Amazon S3

Amazon S3 – Two Trillion Objects

by Jeff Barr | on 18 APR 2013 | in Amazon S3 | Permalink | Comments

5 objects for each star in the galaxy

Amazon S3

Amazon S3 – Two Trillion Objects

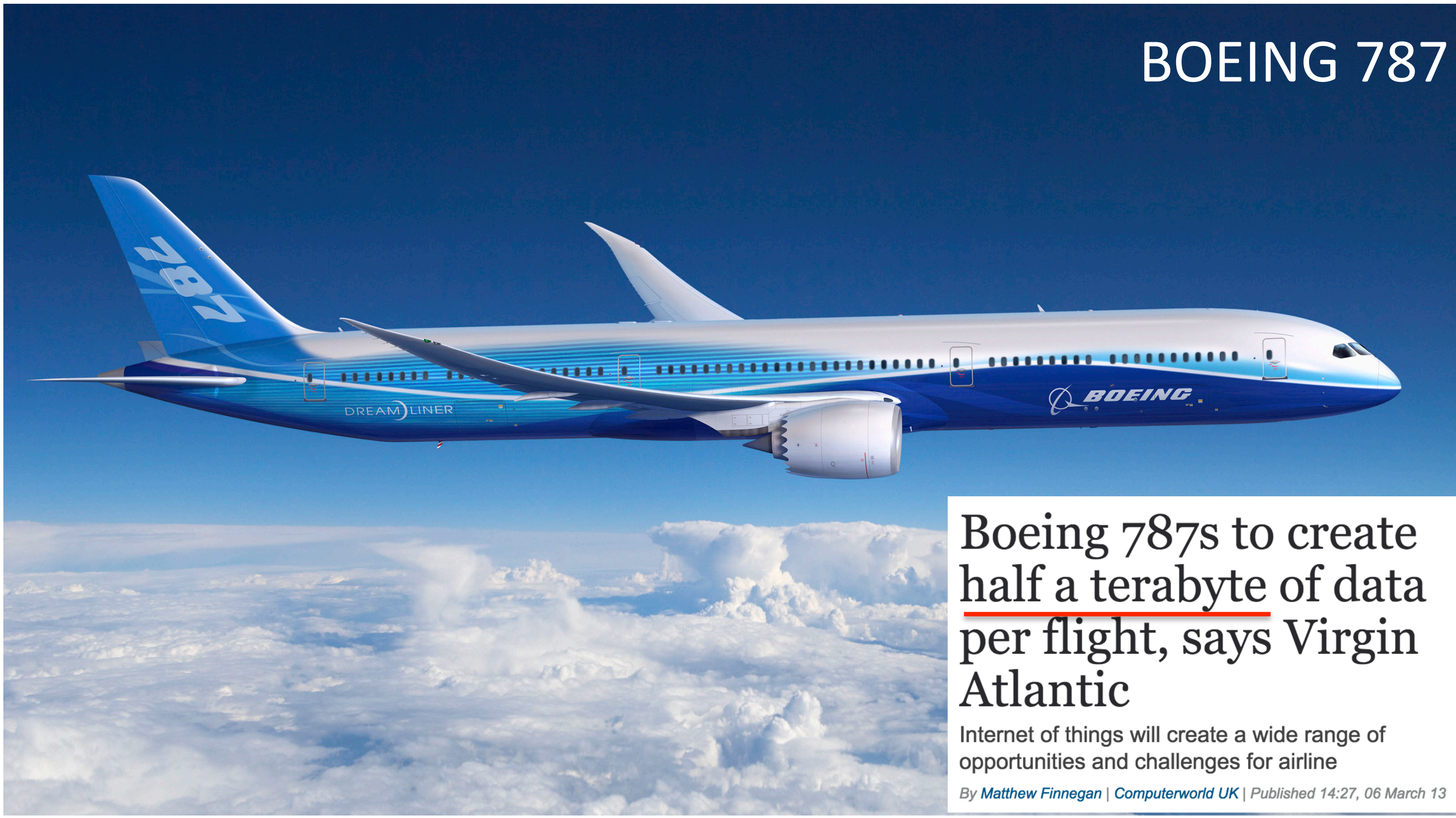
by Jeff Barr | on 18 APR 2013 | in Amazon S3 | Permalink | Comments

5 objects for each star in the galaxy

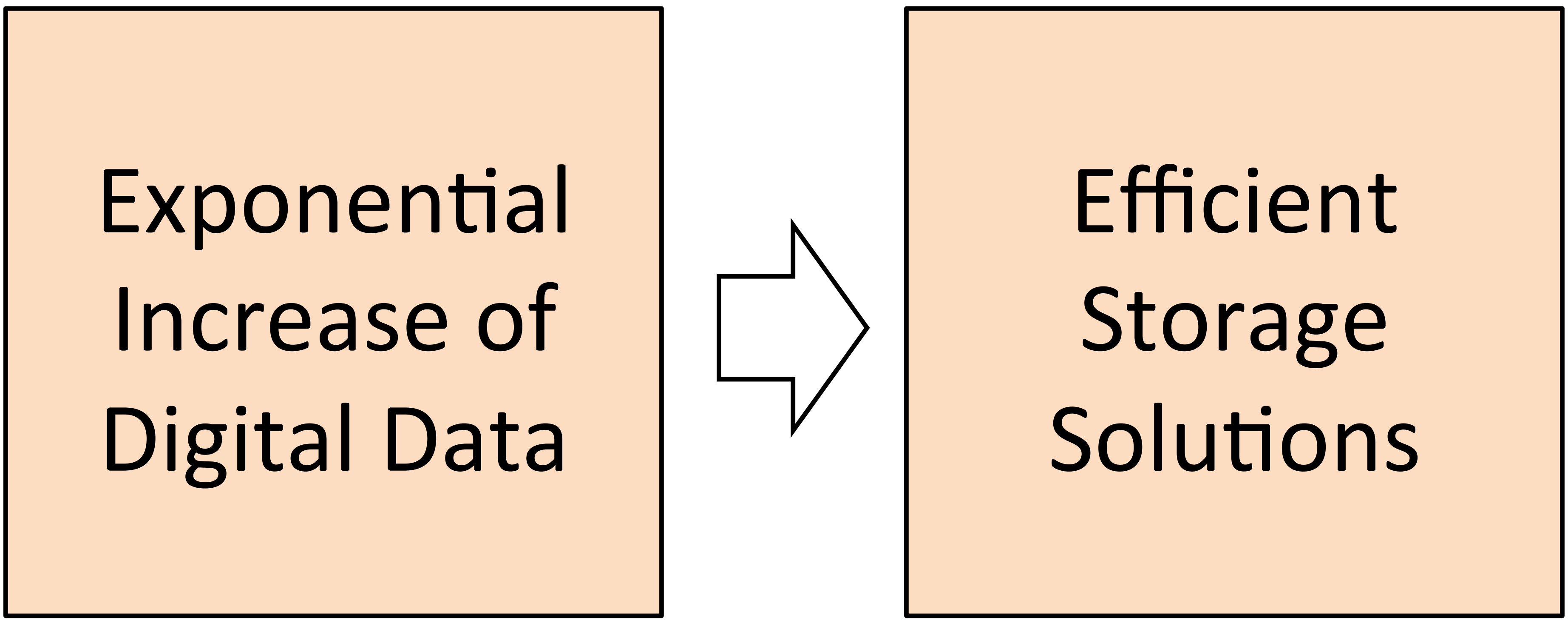
(Assume) average object size is 100 KB, total data size is 200 PB

How Much Data Does This Plane Generate per Flight?

How Much Data Does This Plane Generate per Flight?



Exponential Increase of Digital Data


Exponential Increase of Digital Data
Efficient Storage Solutions

Data Reduction Techniques

- **Compression**: find redundant strings and replace with compact encodings
  - 2x reduction
  - LZ ([Ziv and Lempel 1997]), lz4, gzip, bzip2, 7z, xz, …

Data Reduction Techniques

- **Compression**: find redundant strings and replace with compact encodings
  - 2x reduction
  - LZ ([Ziv and Lempel 1997]), lz4, gzip, bzip2, 7z, xz, …
- **Deduplication**: find duplicate chunks and store unique ones
  - 10x reduction for backups
  - Venti ([Quinlan FAST02]), DataDomain FileSystem ([Zhu FAST08]), iDedup ([Srinivasan FAST12]), …

Limitations

- **Compression**: search redundancy in ***string*** level
  - Does not scale for detecting redundant strings across a large range

Limitations

- **Compression**: search redundancy in ***string*** level
  - Does not scale for detecting redundant strings across a large range
- **Deduplication**: interleave metadata with data
  - Frequent metadata changes introduce many unnecessary unique chunks

Proposed Solutions

- **Migratory Compression**: detect ***similarity*** in block level to group similar blocks (Chap2, FAST14)

Proposed Solutions

- **Migratory Compression**: detect ***similarity*** in block level to group similar blocks (Chap2, FAST14)
- **Deduplication**:
  - Separate metadata from data, to store ***same*** type of data together (Chap3, HotStorage15)

Proposed Solutions

- **Migratory Compression**: detect ***similarity*** in block level to group similar blocks (Chap2, FAST14)
- **Deduplication**:
  - Separate metadata from data, to store ***same*** type of data together (Chap3, HotStorage15)
  - Use deduplication for efficient disk image storage (Chap4, TridentCom15)

Proposed Solutions

- **Migratory Compression**: detect ***similarity*** in block level to group similar blocks (Chap2, FAST14)
- **Deduplication**:
  - Separate metadata from data, to store ***same*** type of data together (Chap3, HotStorage15)
  - Use deduplication for efficient disk image storage (Chap4, TridentCom15)
- **Differential IO Scheduling**: schedule ***same*** type of IO requests for predictable and efficient performance (Chap5, HotCloud12)

Thesis Statement

Similarity in content and access patterns can be utilized to improve **space efficiency**,
by storing **similar** data together
and **performance predictability and efficiency**,
by scheduling **similar** IO requests to the same hard drive.

Outline

- ✓ Introduction
    - ✓ ***Migratory Compression***
    - ✓ Improve deduplication by separating metadata from data
    - ✓ Using deduplication for efficient disk image deployment
    - ✓ Performance predictability and efficiency for Cloud Storage Systems
- ✓ Conclusion

Background on Compression

- Compression: finds redundant strings within a window size and encodes with more compact structures

Background on Compression

- Compression: finds redundant strings within a window size and encodes with more compact structures
- Metrics
  - Compression Factor (CF) = original size / compressed size
  - Throughput = original size / (de)compression time

Background on Compression

- Compression: finds redundant strings within a window size and encodes with more compact structures
- Metrics
  - Compression Factor (CF) = original size / compressed size
  - 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	Variant of LZ77; Markov chain-based range coder

Background on Compression

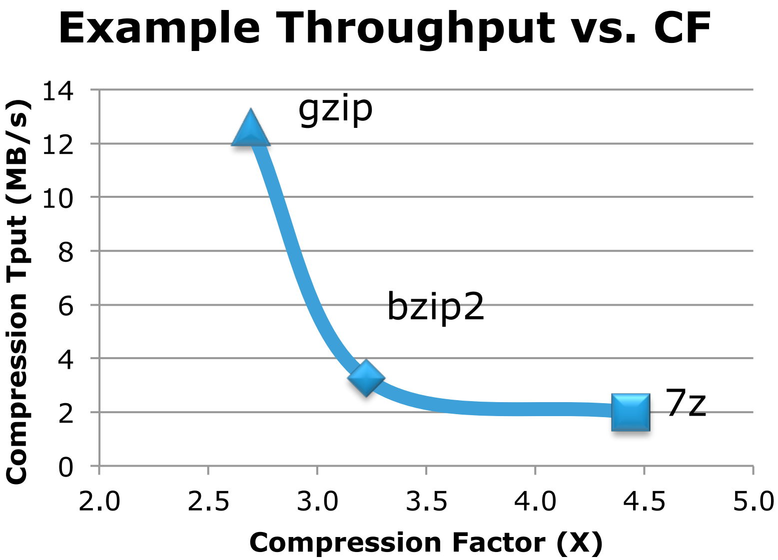
- Compression: finds redundant strings within a window size and encodes with more compact structures
- Metrics
  - Compression Factor (CF) = original size / compressed size
  - Throughput = original size / (de)compression time

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	Variant of LZ77; Markov chain-based range coder

Background on Compression

- Compression: finds redundant strings within a window size and encodes with more compact structures
- Metrics
  - Compression Factor (CF) = original size / compressed size
  - Throughput = original size / (de)compression time

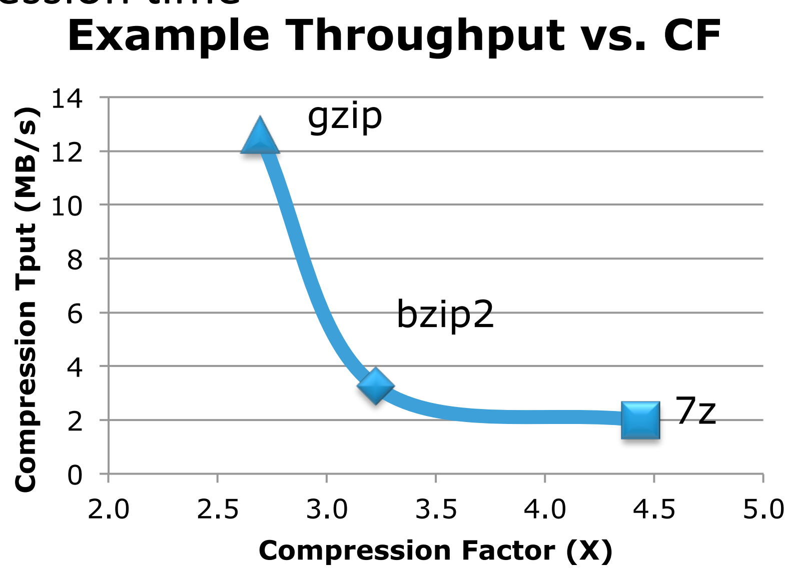
Compressor	MAX Window size	Techniques
gzip	64 KB	LZ; Huffman coding
bzip2	900 KB	Run-length encoding; …
7z	1 GB	Markov chain-based range coder



Background on Compression

- Compression: finds redundant strings within a window size and encodes with more compact structures
- Metrics
  - Compression Factor (CF) = original size / compressed size
  - Throughput = original size / (de)compression time

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



The larger the window, the **better** the compression but **slower**.
Fundamental reason: finding redundancy in ***string level*** does **not** scale to ***large windows***

Migratory Compression

- **Problem**: finding redundancy in ***string level*** does not ***scale*** to large windows
- **Key idea**: group by similarity in ***block level***, enabling standard compressors to find repeated strings with ***small windows***

Migratory Compression

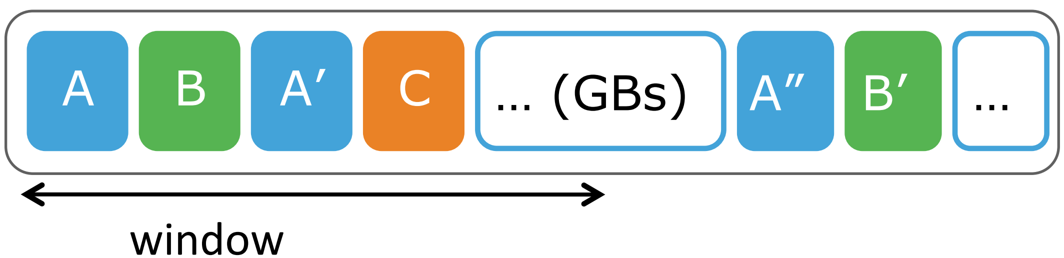
- **Problem**: finding redundancy in ***string level*** does not ***scale*** to large windows
- **Key idea**: group by similarity in ***block level***, enabling standard compressors to find repeated strings with ***small windows***
- ***Migratory compression (MC)***: coarse-grained reorganization to 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

Migratory Compression

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

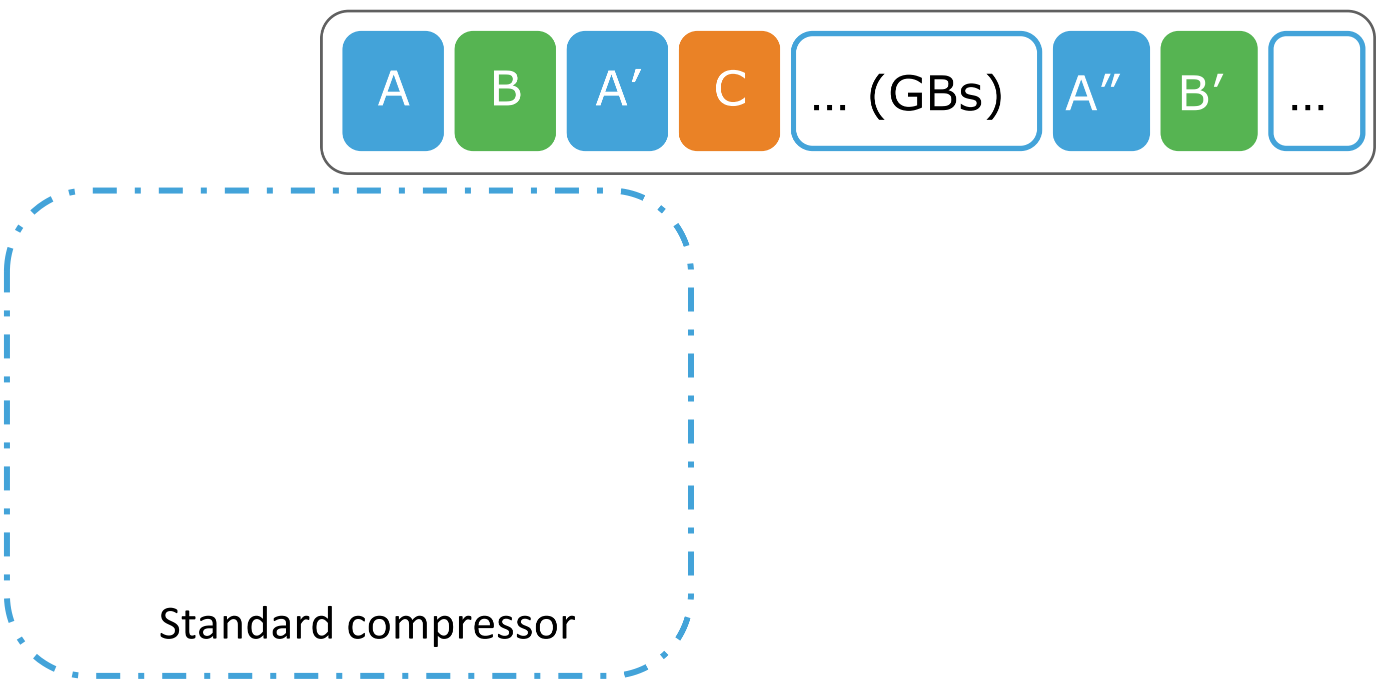
Migratory Compression

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



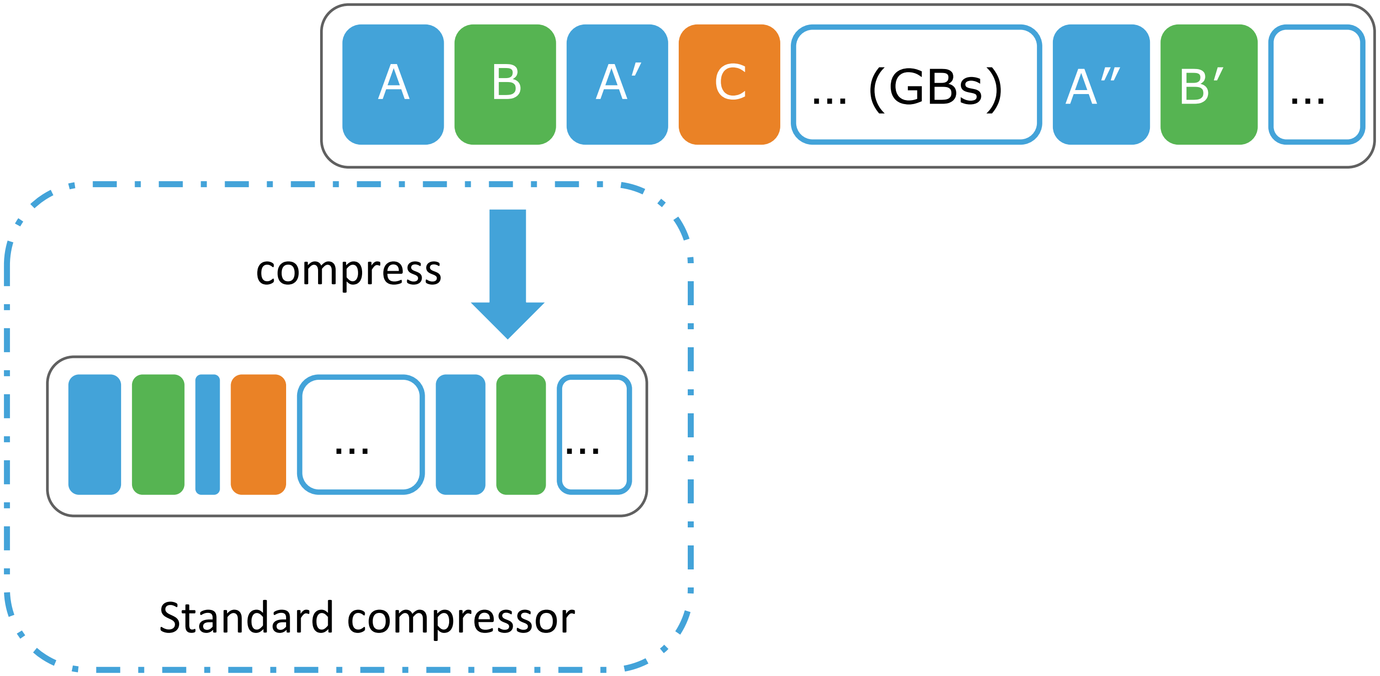
Migratory Compression

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



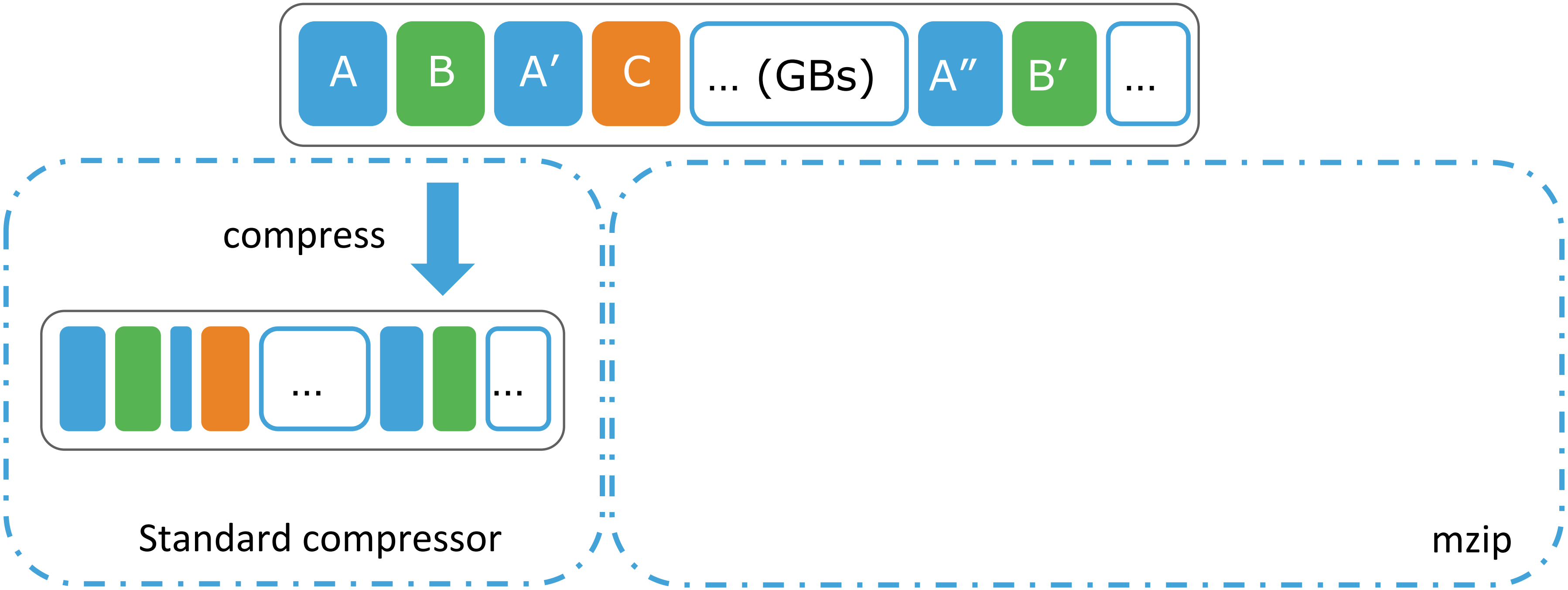
Migratory Compression

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



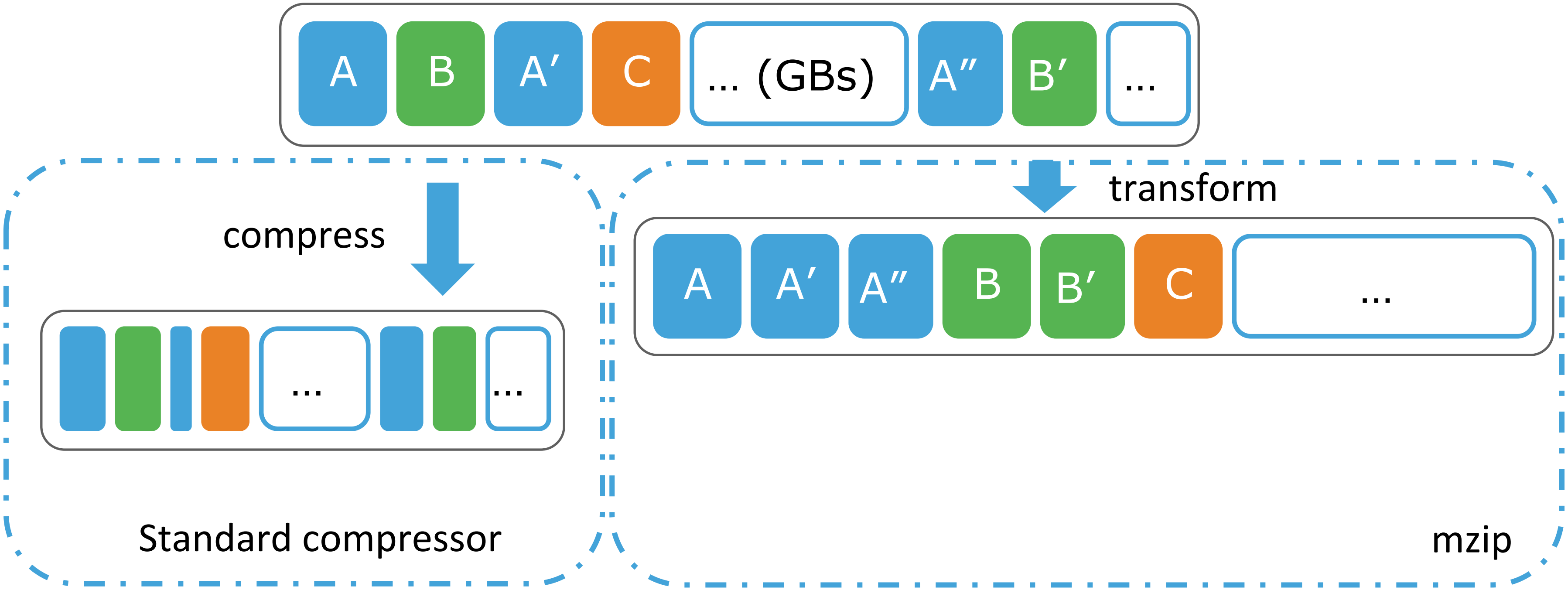
Migratory Compression

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



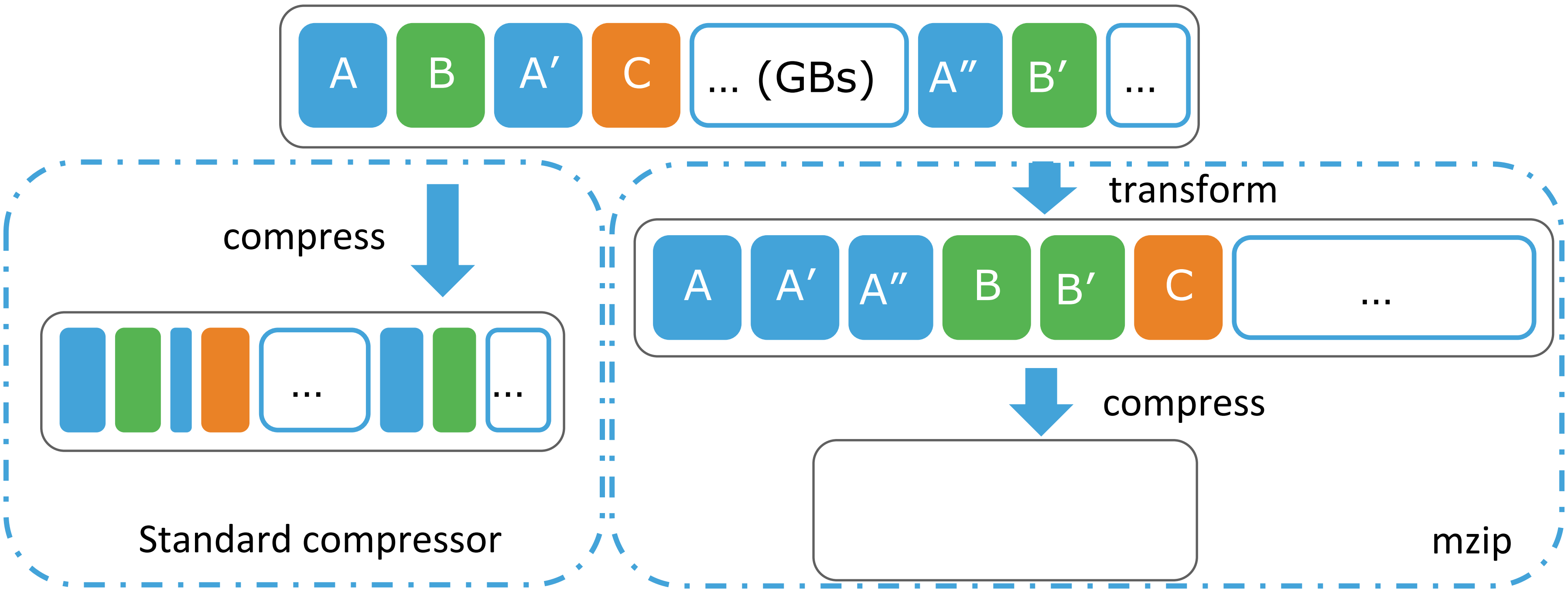
Migratory Compression

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



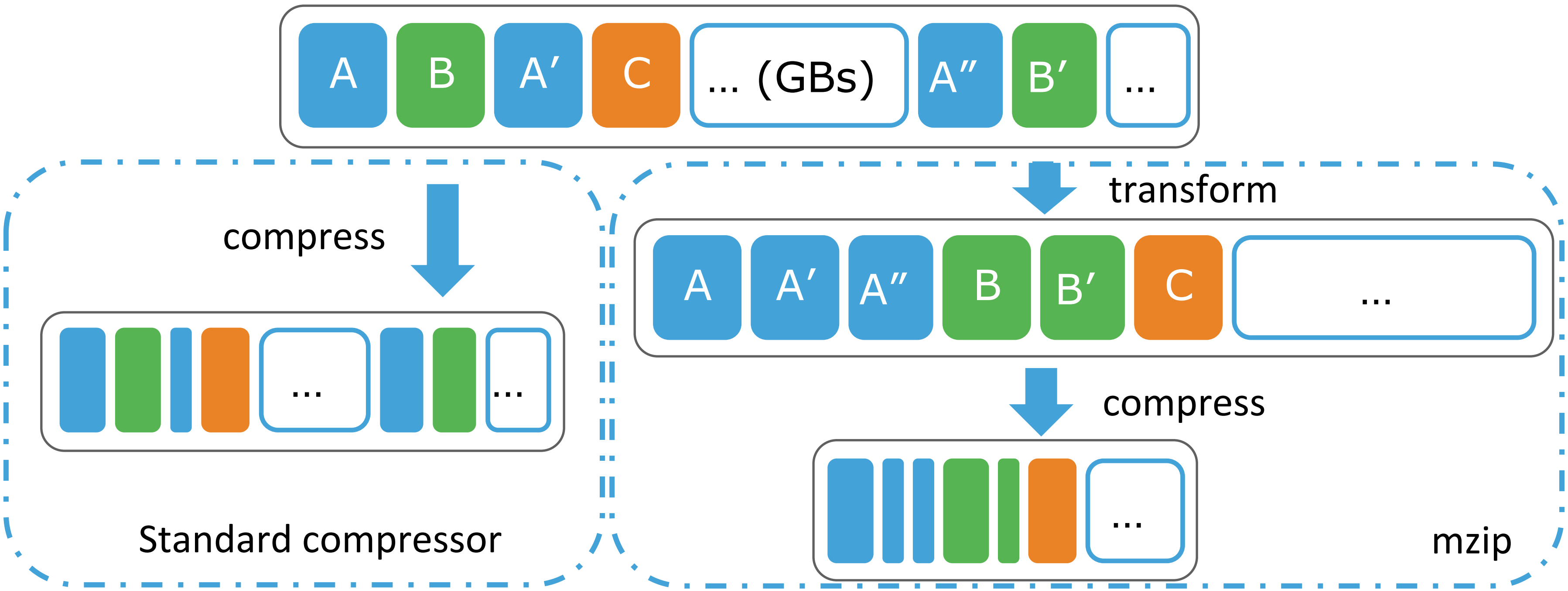
Migratory Compression

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

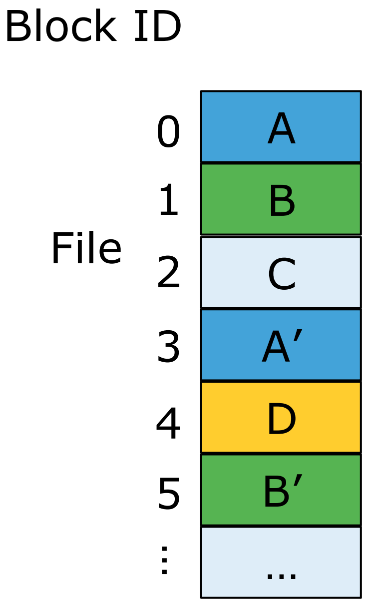


Migratory Compression

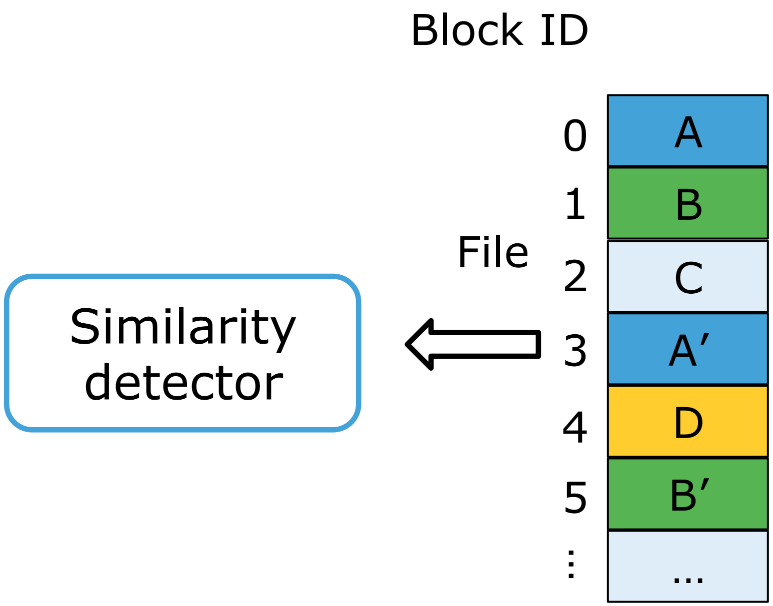
- Compress a single, large file (mzip)
  - Traditional compressors are unable to exploit redundancy across a large range of data (e.g., many GB)



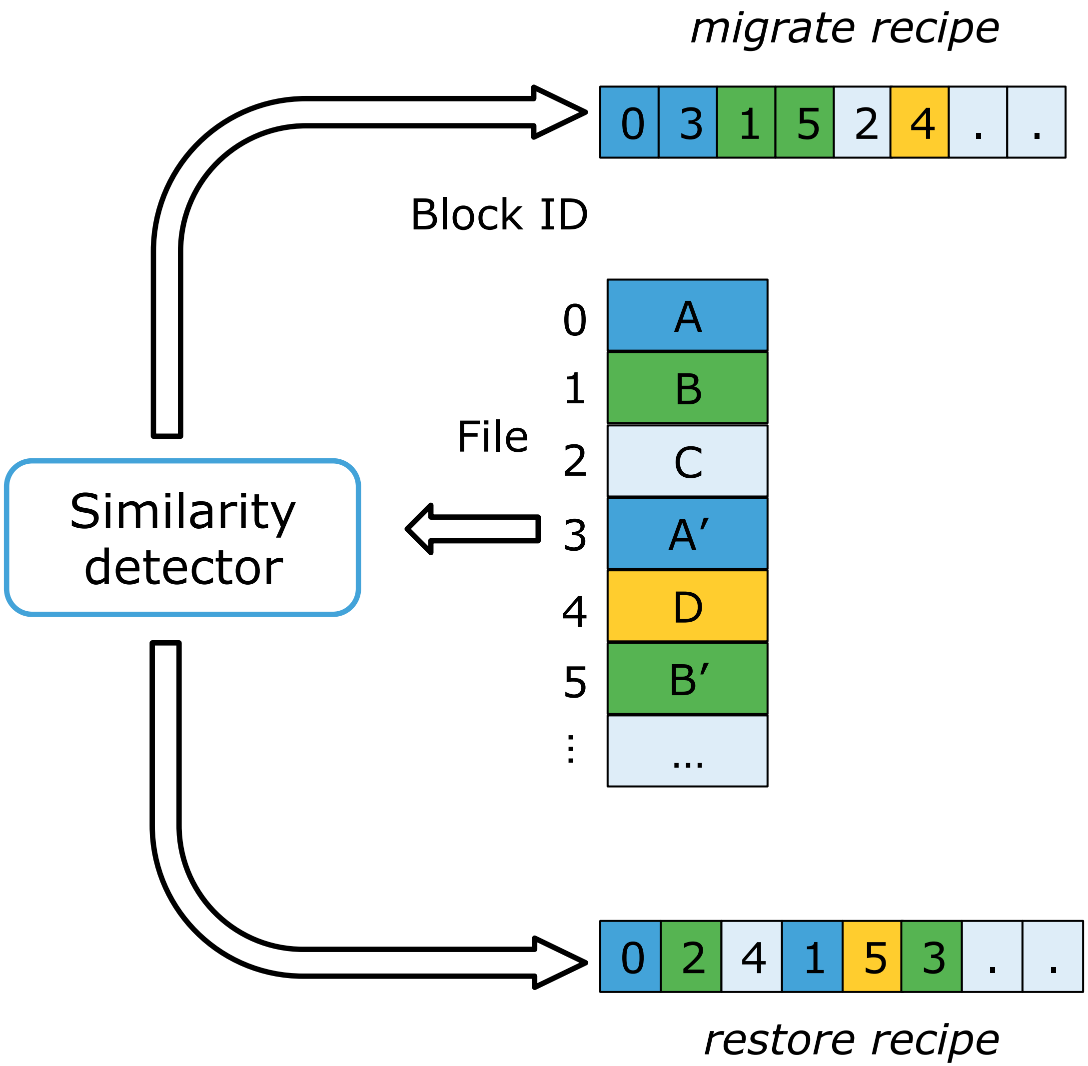
mzip Example



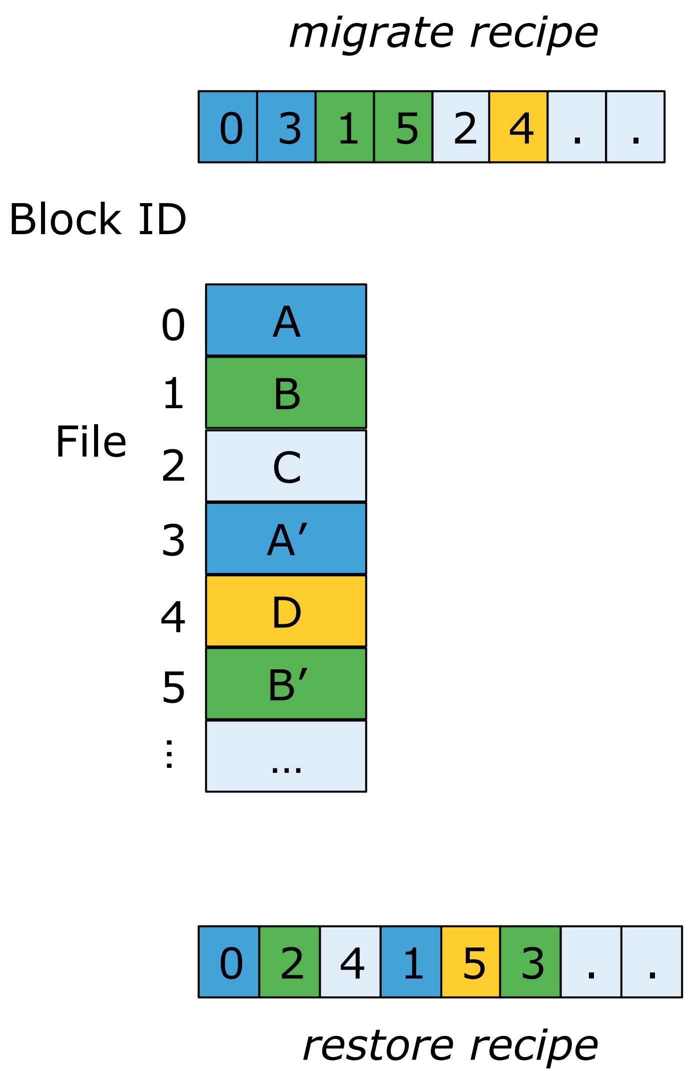
mzip Example



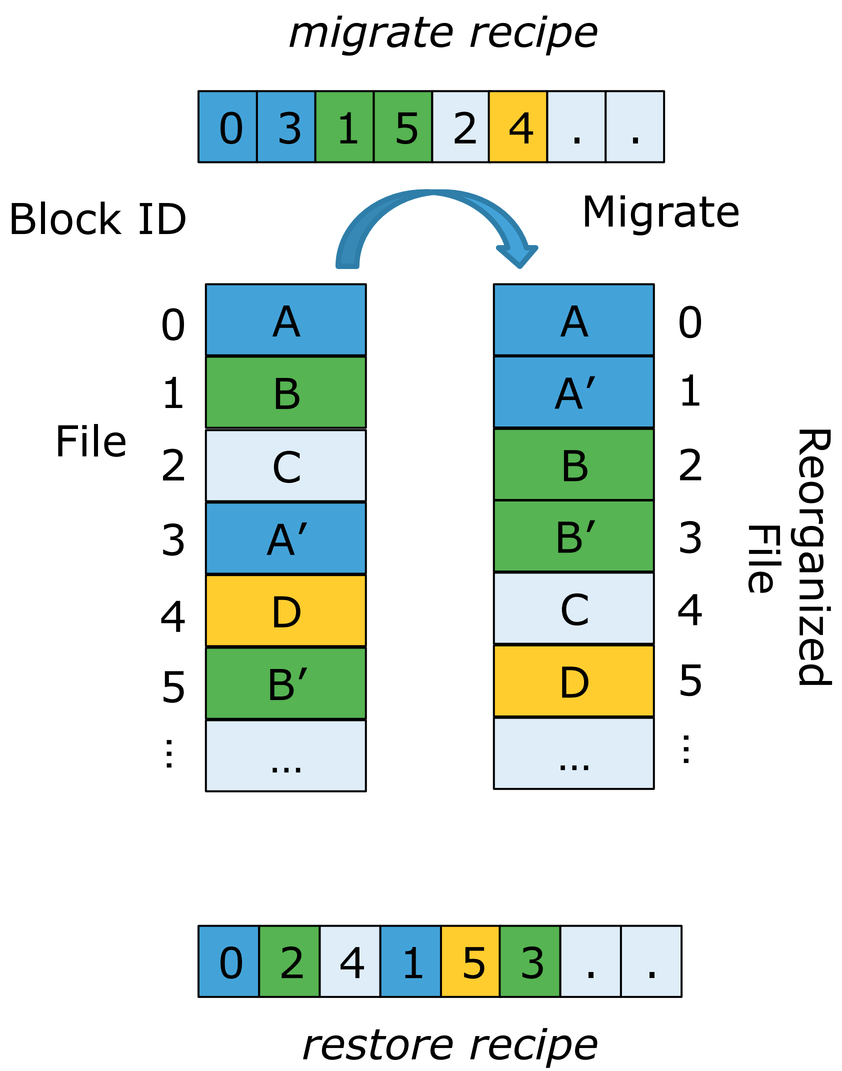
mzip Example



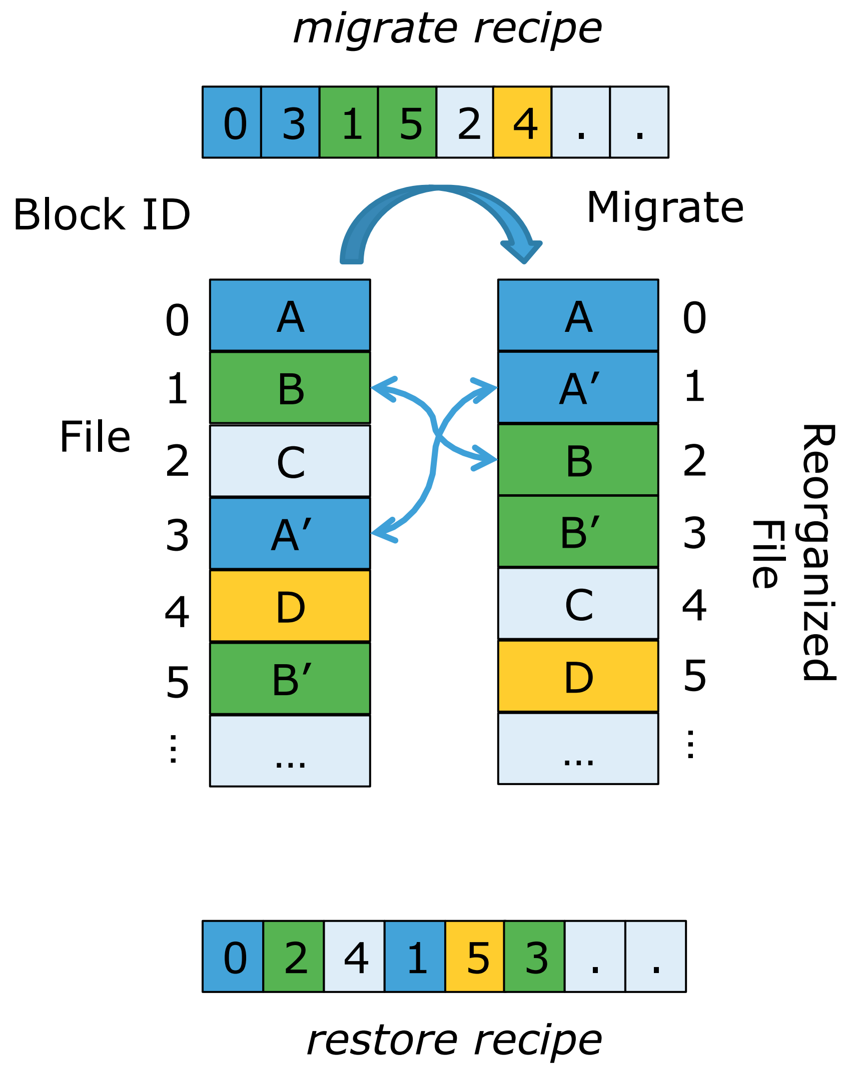
mzip Example



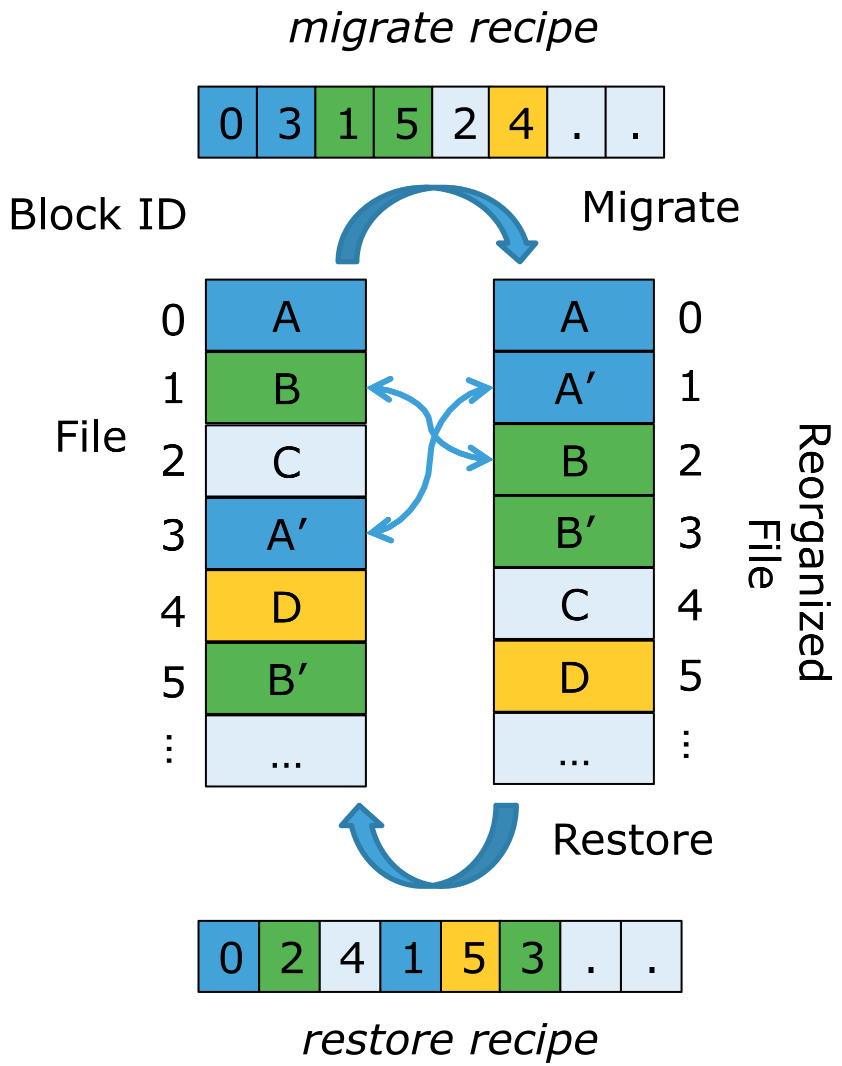
mzip Example



mzip Example



mzip Example



Detect Similar Blocks

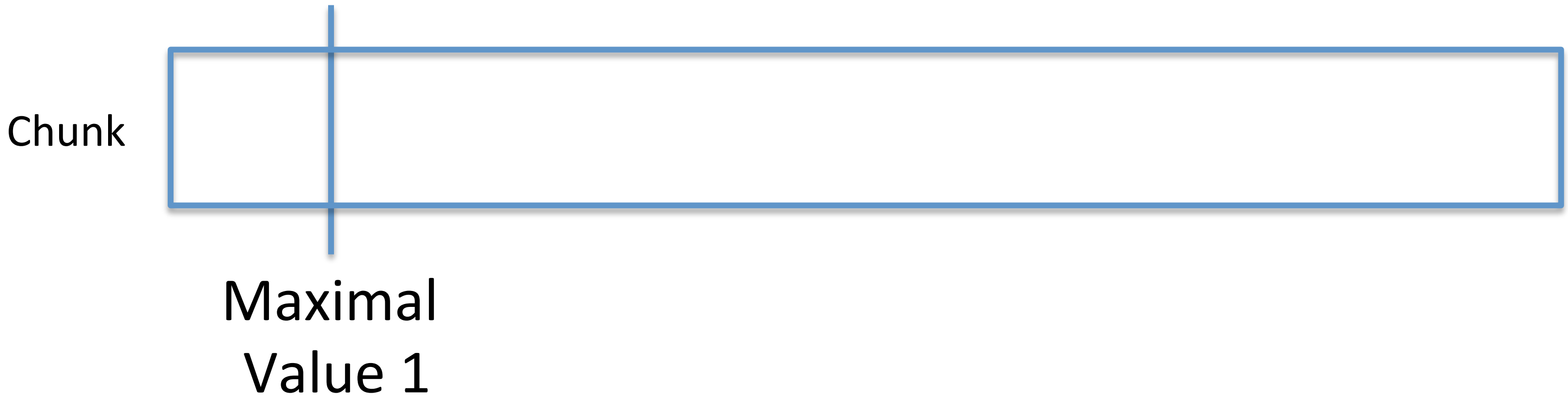
Similarity feature: hash ([Broder 1997])

Detect Similar Blocks

Similarity feature: hash ([Broder 1997])

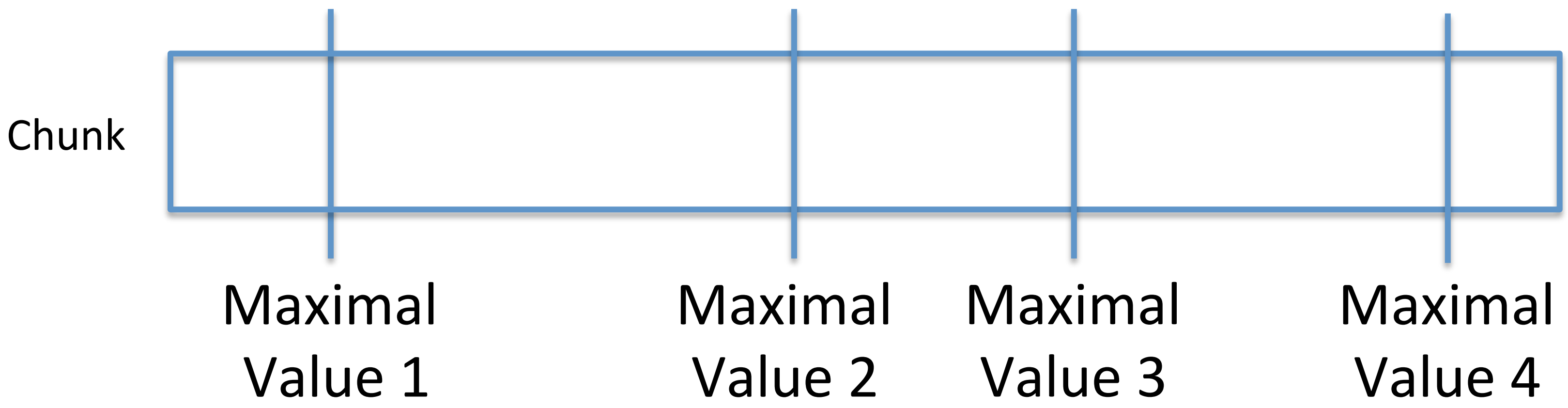
Detect Similar Blocks

Similarity feature: hash ([Broder 1997])



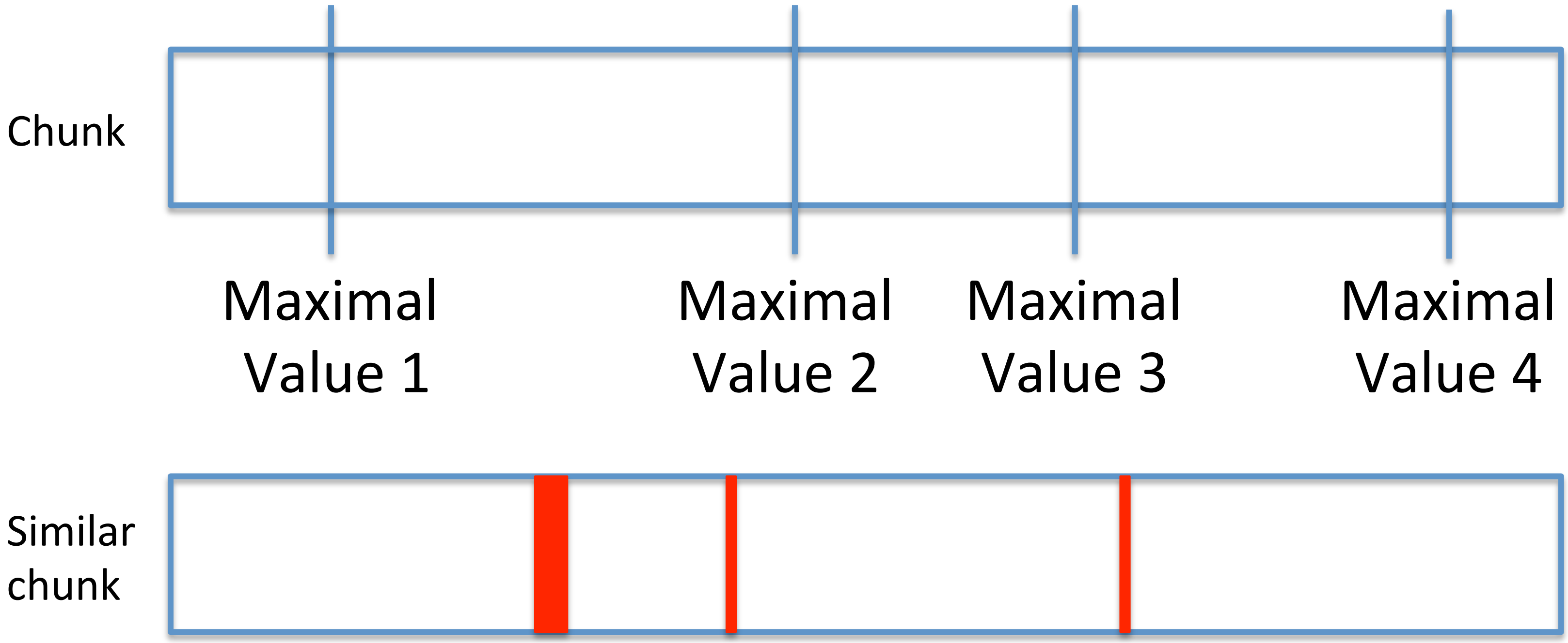
Detect Similar Blocks

Similarity feature: hash ([Broder 1997])



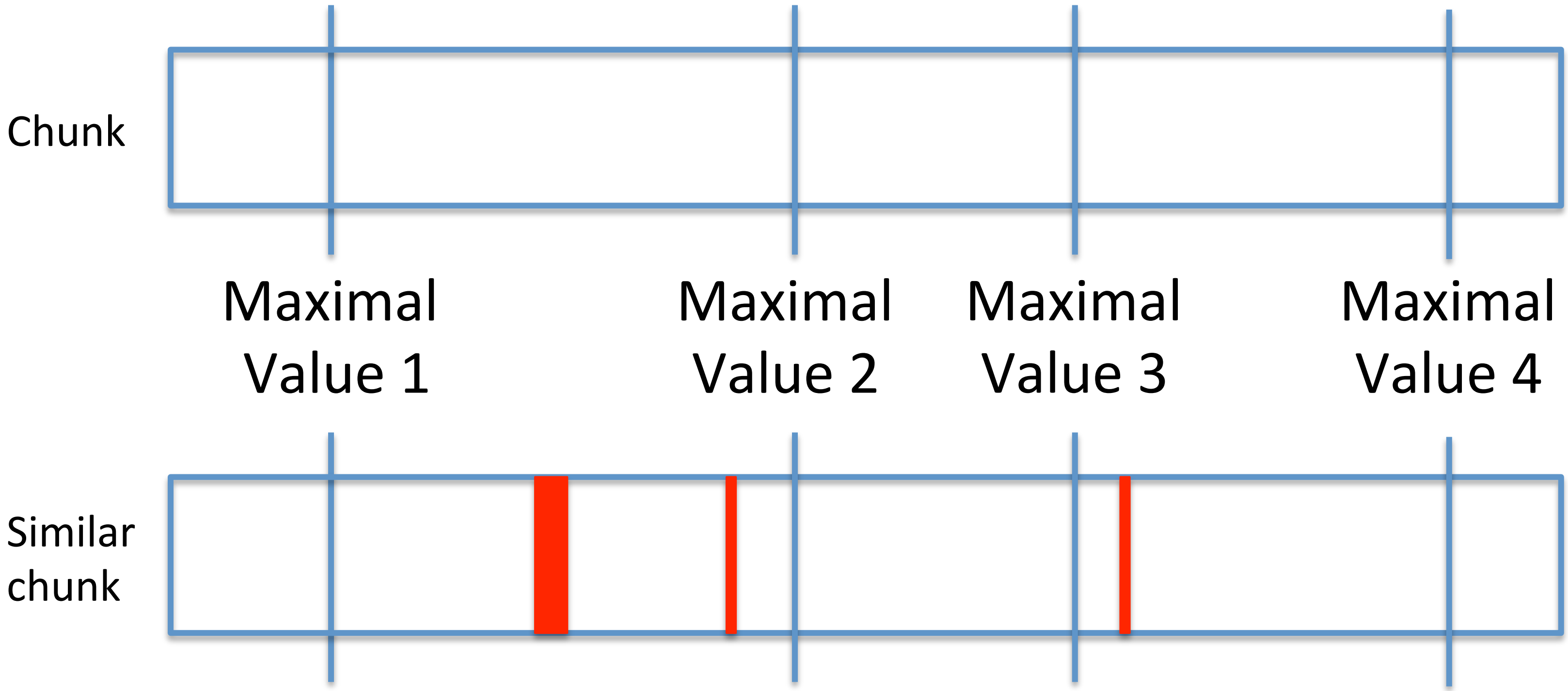
Detect Similar Blocks

Similarity feature: hash ([Broder 1997])



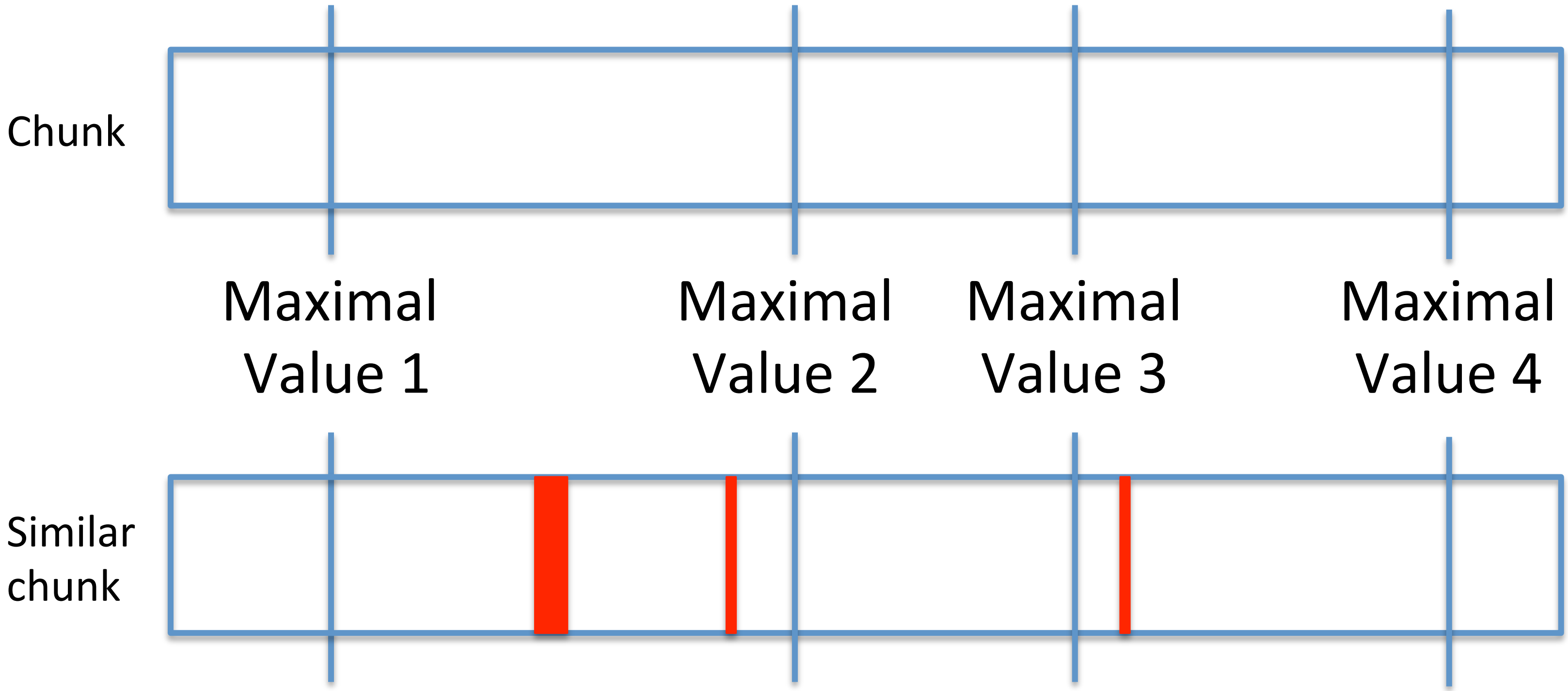
Detect Similar Blocks

Similarity feature: hash ([Broder 1997])



Detect Similar Blocks

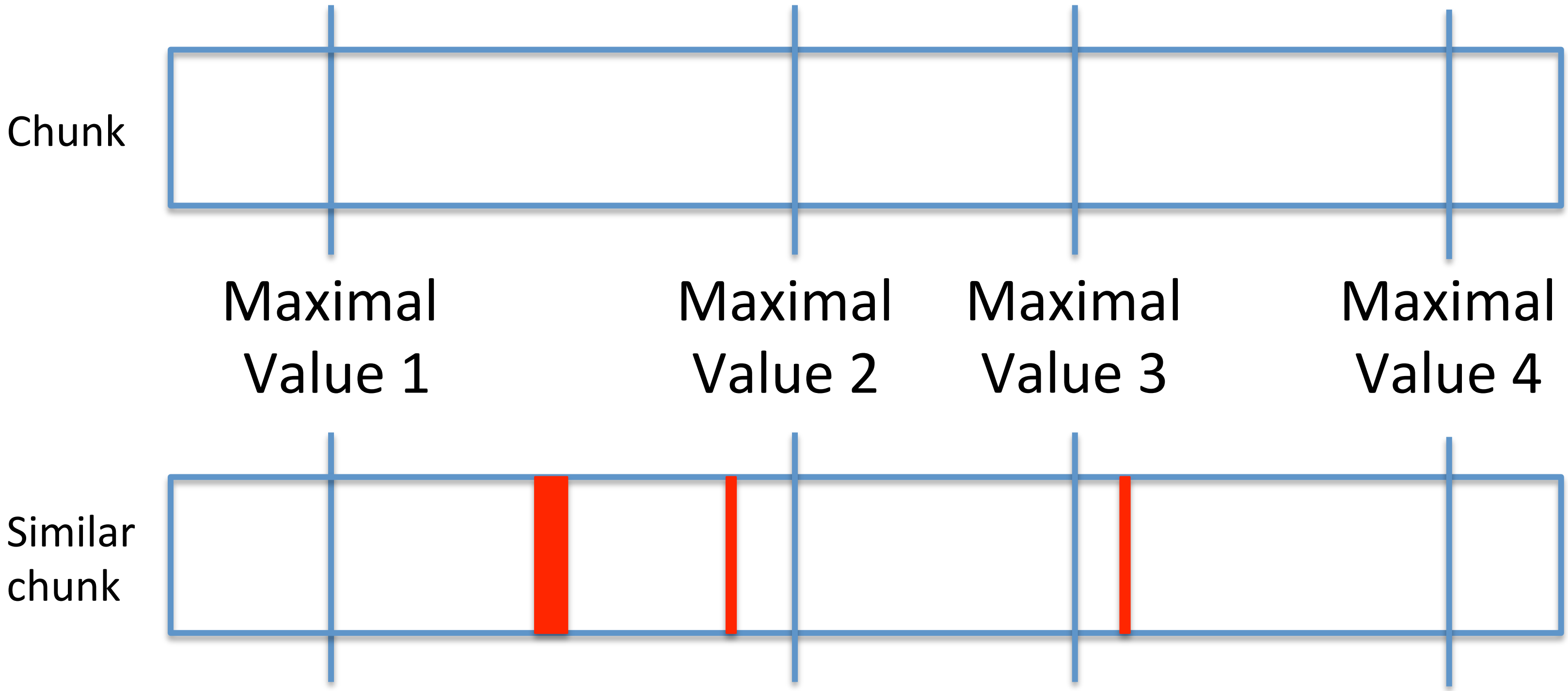
Similarity feature: hash ([Broder 1997])



Super-feature ([Shilane FAST12])

Detect Similar Blocks

Similarity feature: hash ([Broder 1997])



Super-feature ([Shilane FAST12])

A match on any super-feature is good enough

Efficient Data Reorganization

- **Problem**: requires lots of random IOs

Efficient Data Reorganization

- **Problem**: requires lots of random IOs
- Hard drives: does not perform well
- SSD: provides good random IO performance

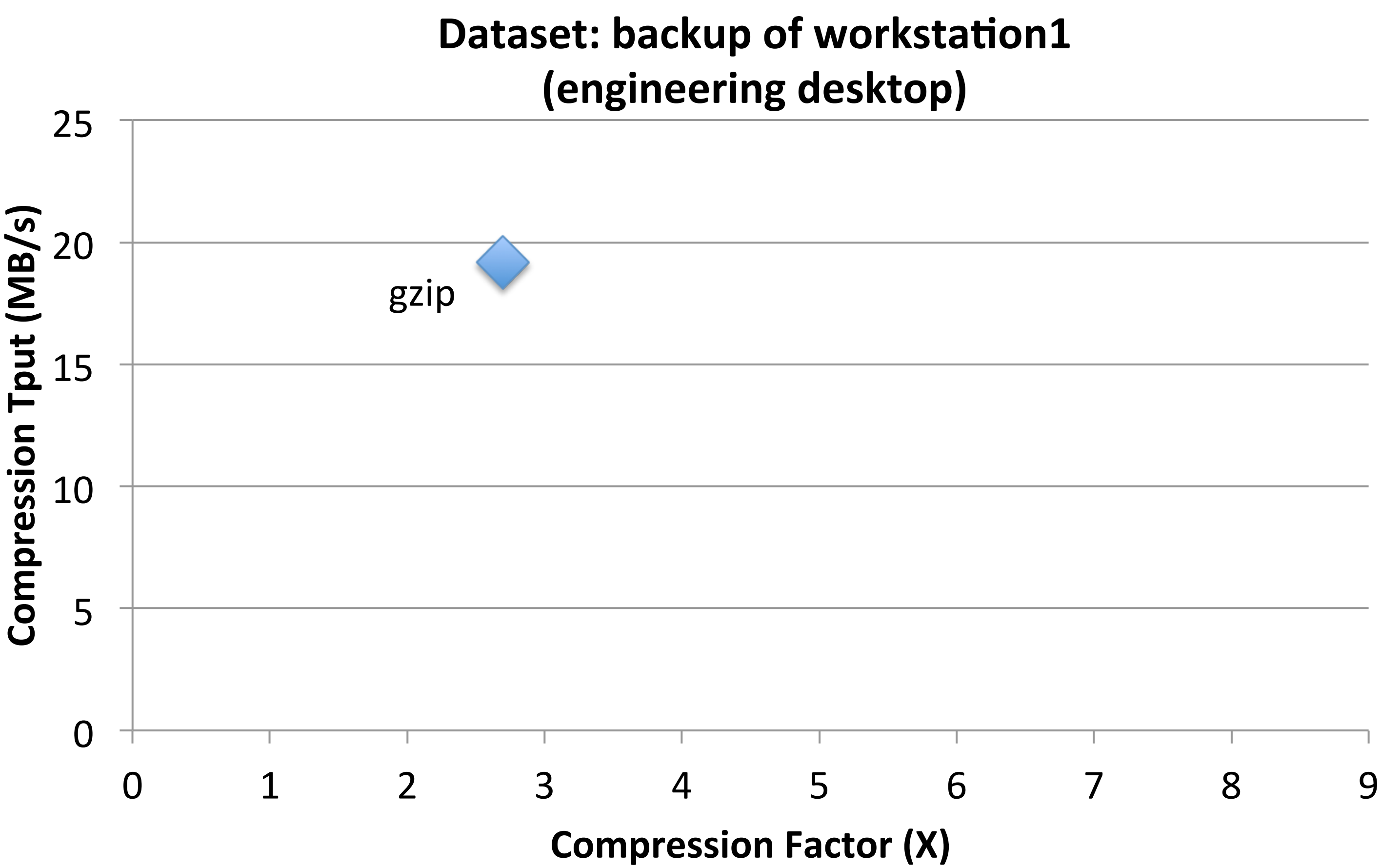
Efficient Data Reorganization

- **Problem**: requires lots of random IOs
- Hard drives: does not perform well
- SSD: provides good random IO performance
- Multi-pass: helps considerably for hard drives
  - convert random IOs into multiple scans of input files

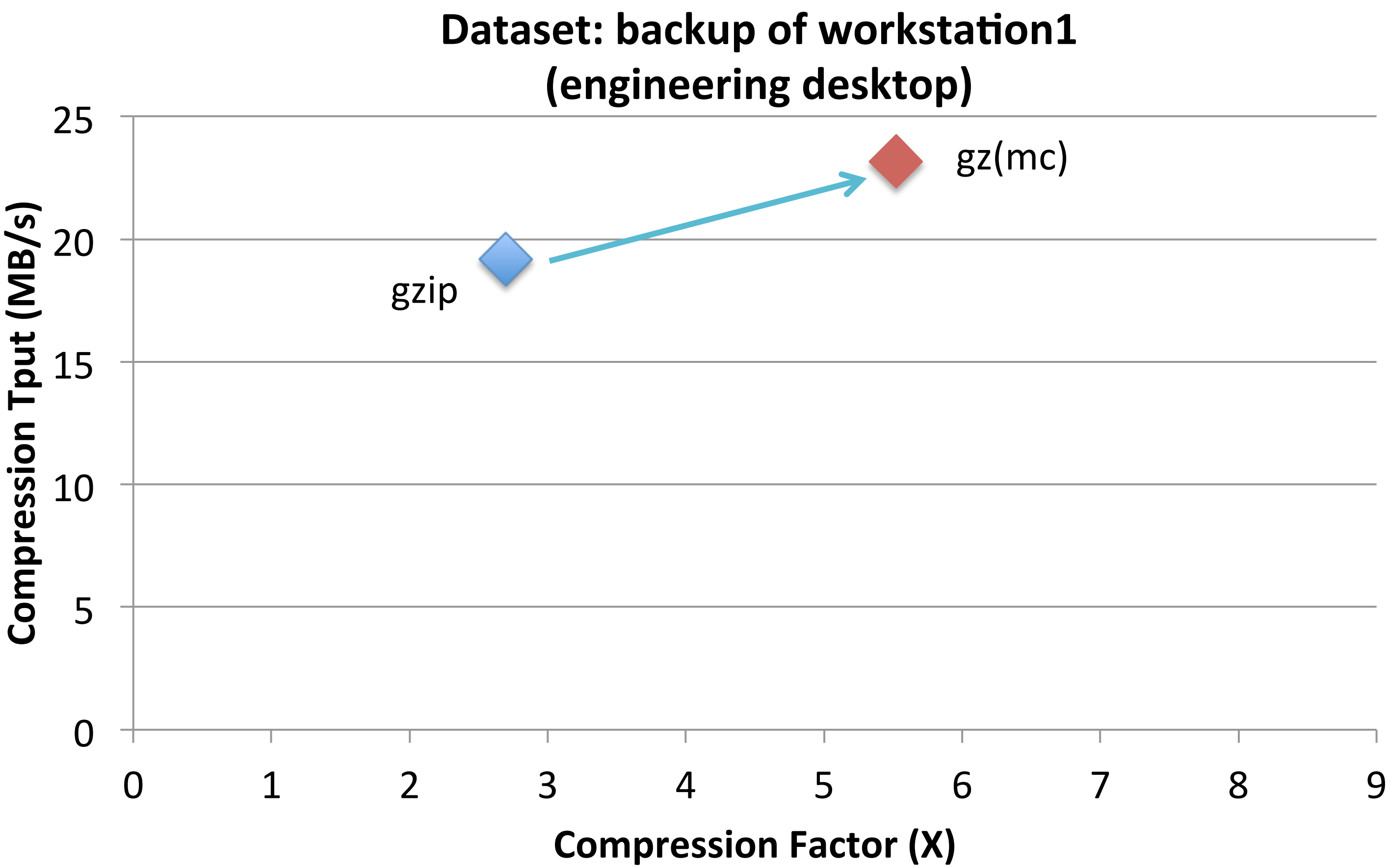
Evaluation

- How much can compression be improved?
  - Compression Factor (CF): original size / compressed size
- What is the complexity (runtime overhead)?
  - Compression throughput: original size / runtime
- More in the paper
  - How does SSD or HDD affect data reorganization performance? For HDD, does multi-pass help?
  - How does MC perform, compared with delta compression?
  - What are the configuration options for MC?

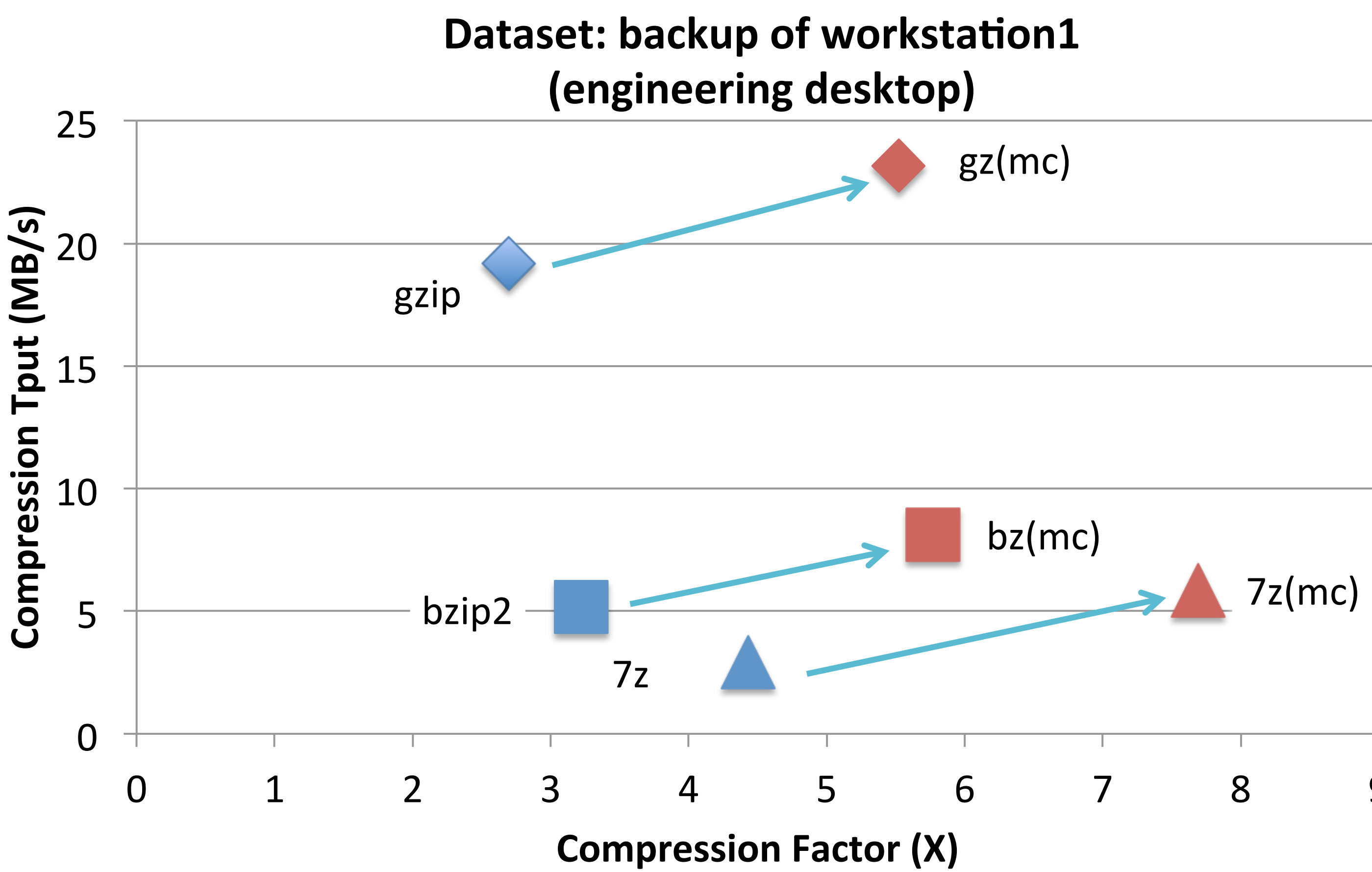
Migratory Compression - Evaluation



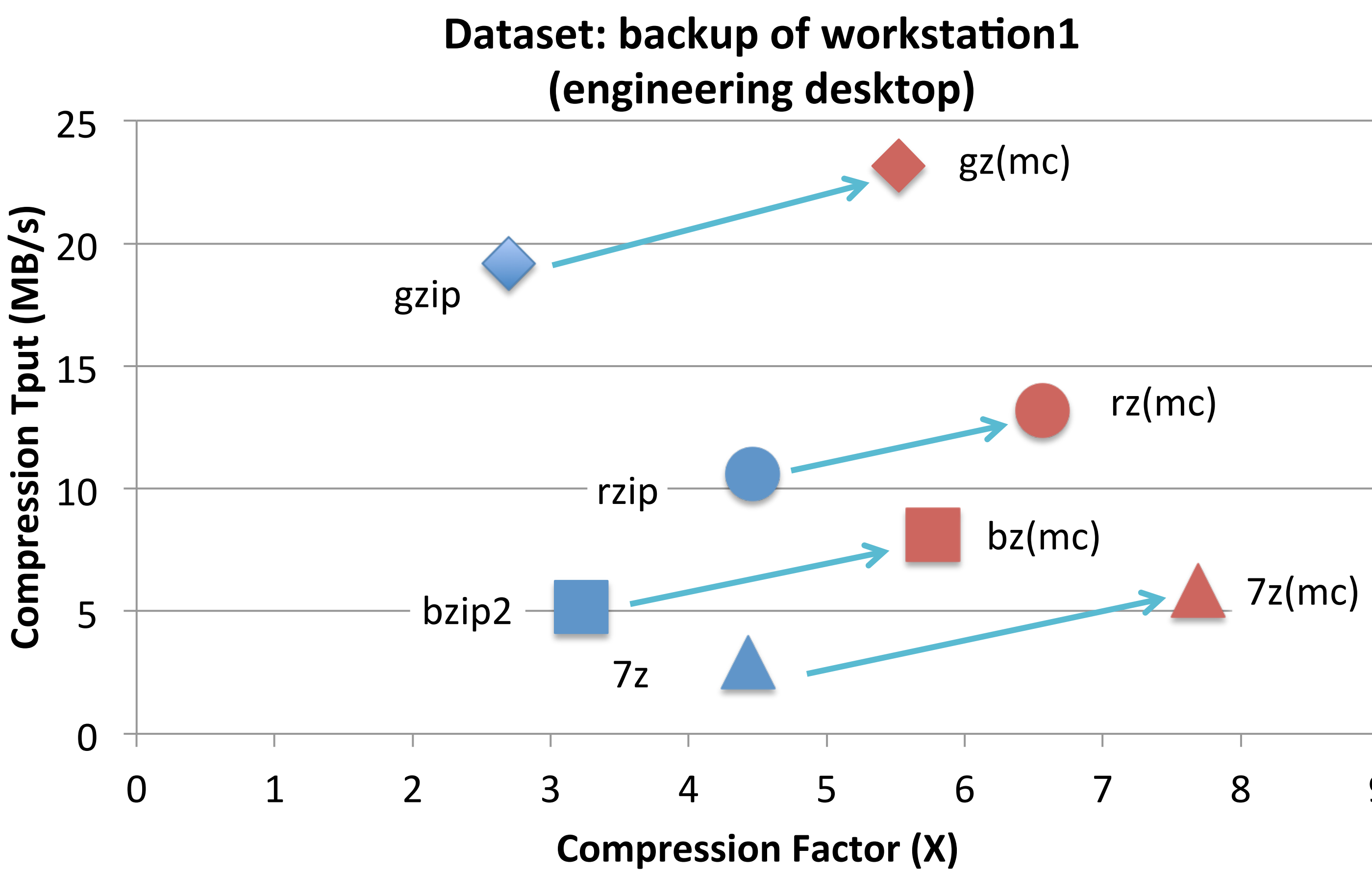
Migratory Compression - Evaluation



Migratory Compression - Evaluation



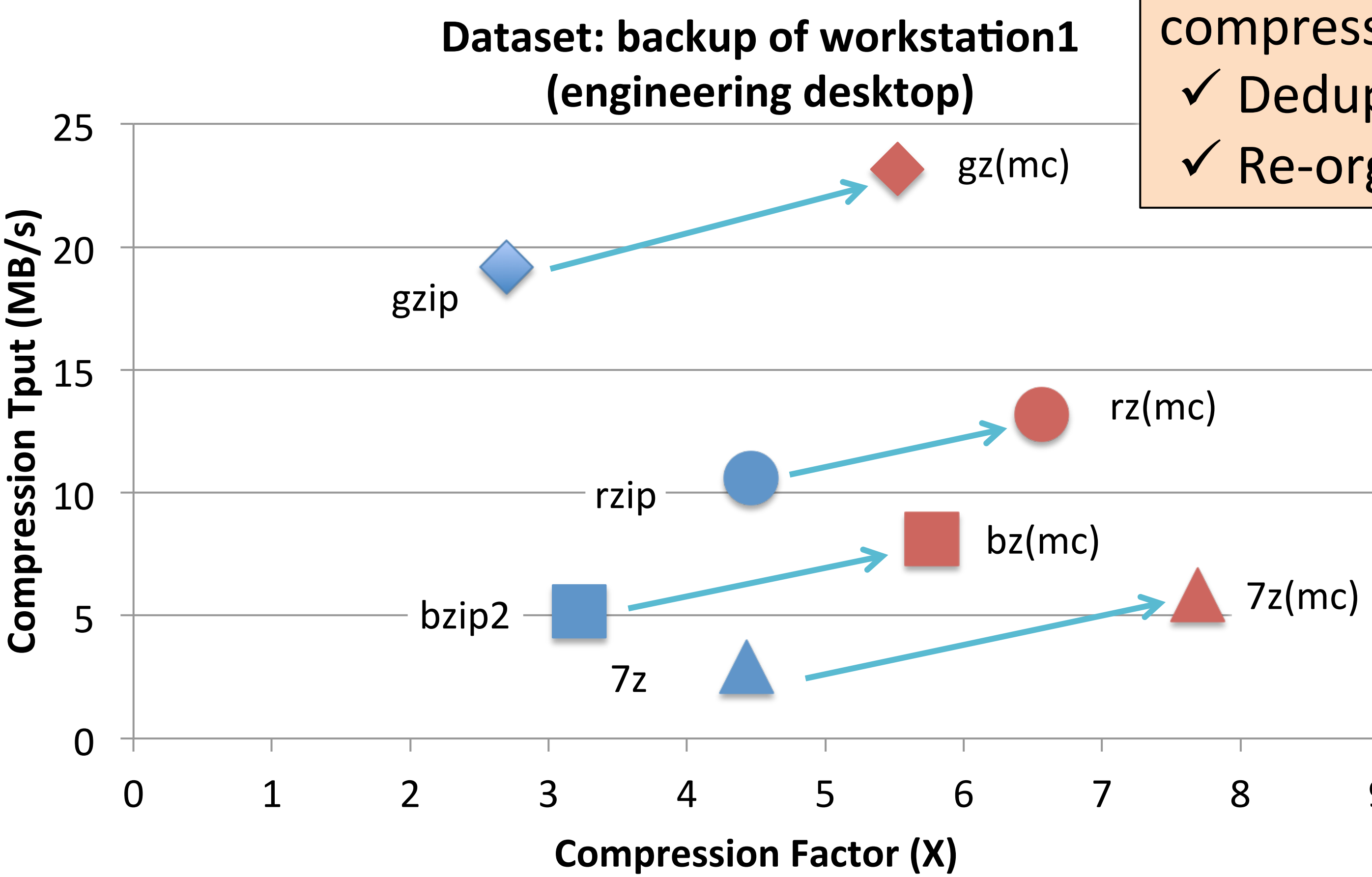
Migratory Compression - Evaluation



rzip
- intra-file dedupe
- bzip2 remainder

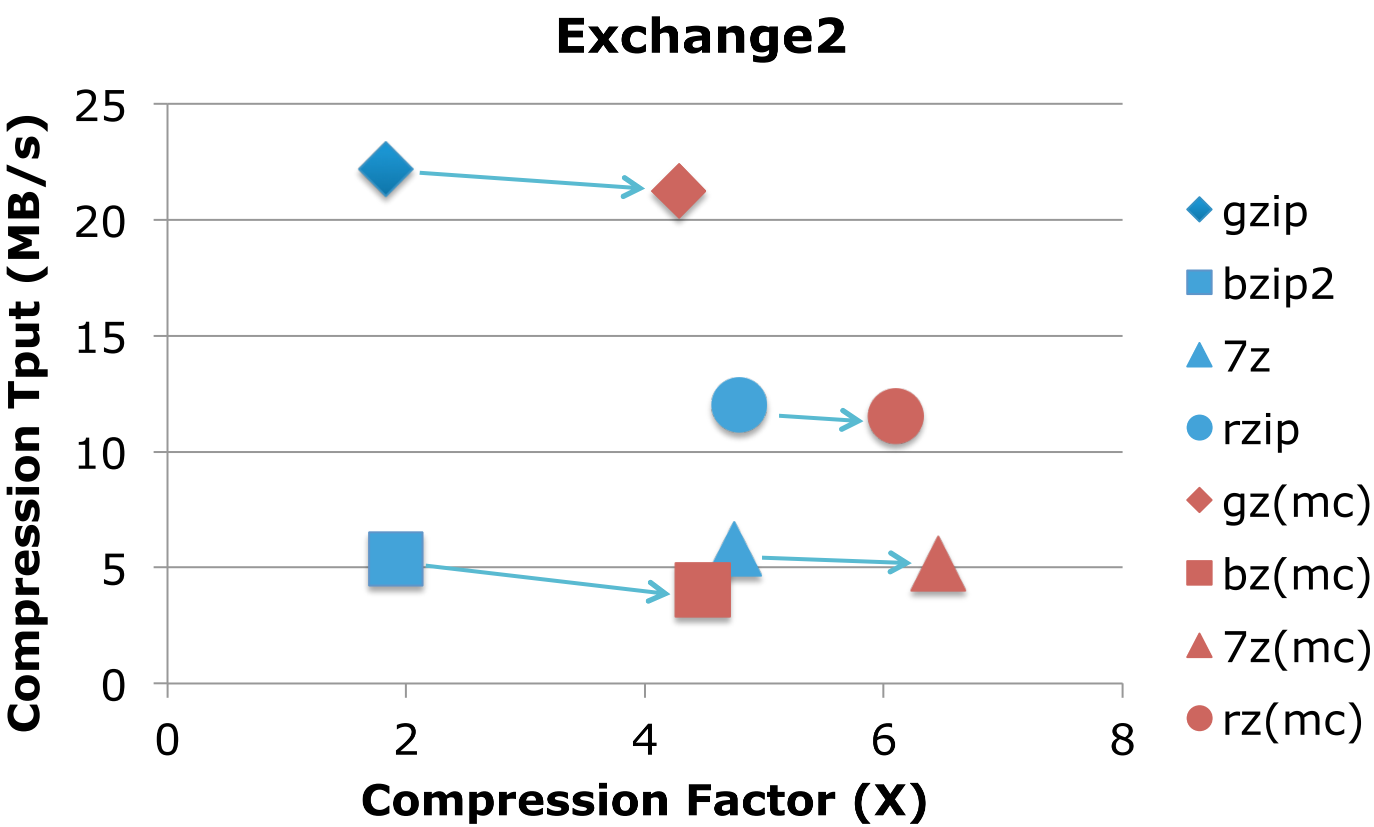
Migratory Compression - Evaluation

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



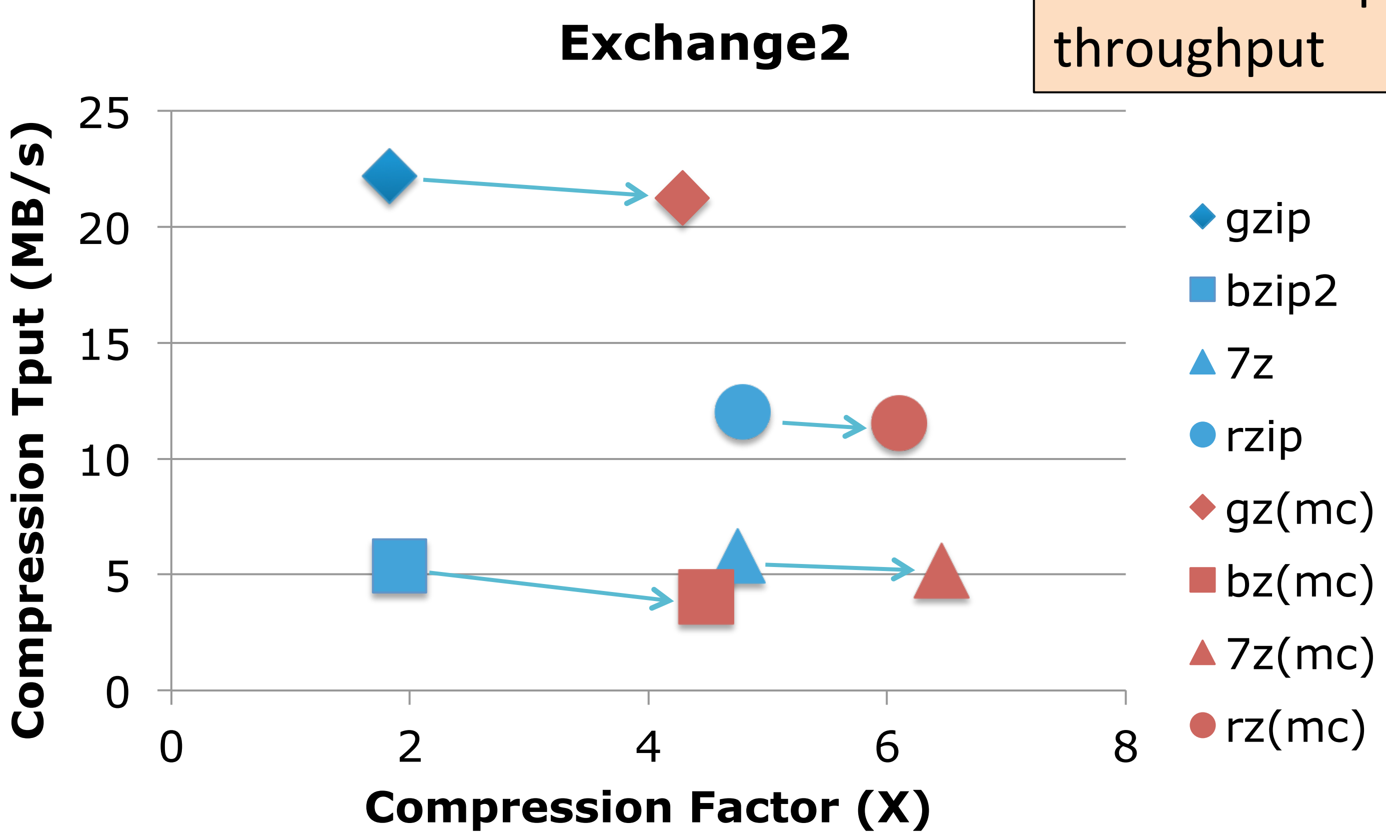
rzip
- intra-file dedupe
- bzip2 remainder

Compression Factor vs. Compression Throughput



Compression Factor vs. Compression Throughput

MC improves CF but slightly reduces compression throughput



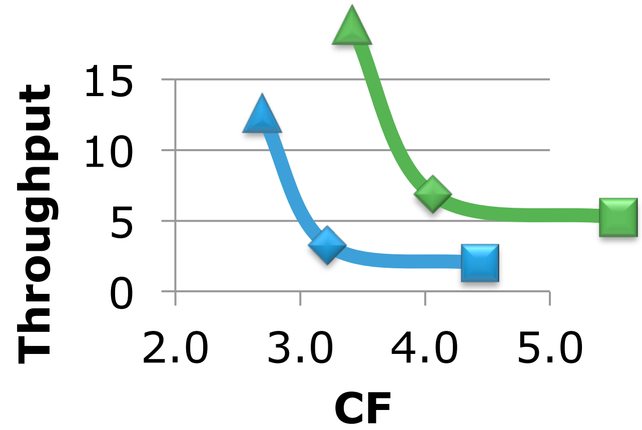
Migratory Compression - Summary

- More results in the paper
  - Decompression performance
  - Default vs. maximal compression
  - Memory vs. SSD vs. hard drives
  - Application of MC for archival storage

Migratory Compression: Coarse-grained Data Reordering to Improve Compressibility
Xing Lin, Guanlin Lu, Fred Douglis, Philip Shilane, and Grant Wallace
FAST '14: Proceedings of the 12th USENIX Conference on File and Storage Technologies, Feb. 2014

Migratory Compression - Summary

- More results in the paper
  - Decompression performance
  - Default vs. maximal compression
  - Memory vs. SSD vs. hard drives
  - Application of MC for archival storage



- Migratory Compression
  - Improves existing compressors, in both **compressibility** and frequently **runtime**
  - ***Redraw the performance-compression curve!***

Migratory Compression: Coarse-grained Data Reordering to Improve Compressibility
Xing Lin, Guanlin Lu, Fred Douglis, Philip Shilane, and Grant Wallace
FAST '14: Proceedings of the 12th USENIX Conference on File and Storage Technologies, Feb. 2014

Outline

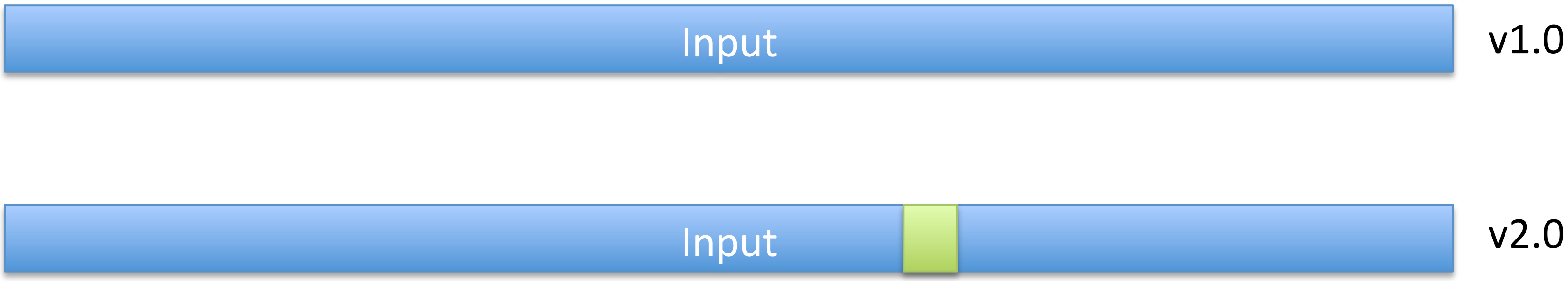
- ✓ Introduction
  - ✓ Migratory Compression
  - ✓ ***Improve deduplication by separating metadata from data***
  - ✓ Using deduplication for efficient disk image deployment
  - ✓ Performance predictability and efficiency for Cloud Storage Systems
- ✓ Conclusion

Deduplication

Idea: identify duplicate data blocks and store a single copy

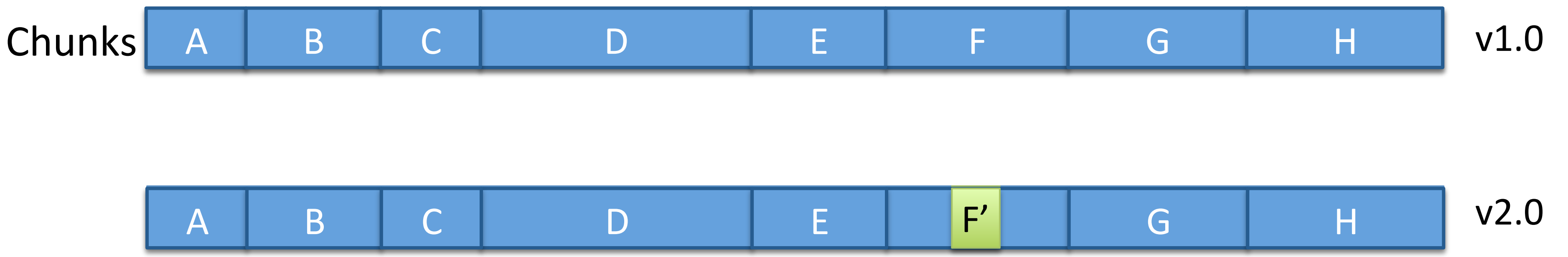
Deduplication

Idea: identify duplicate data blocks and store a single copy



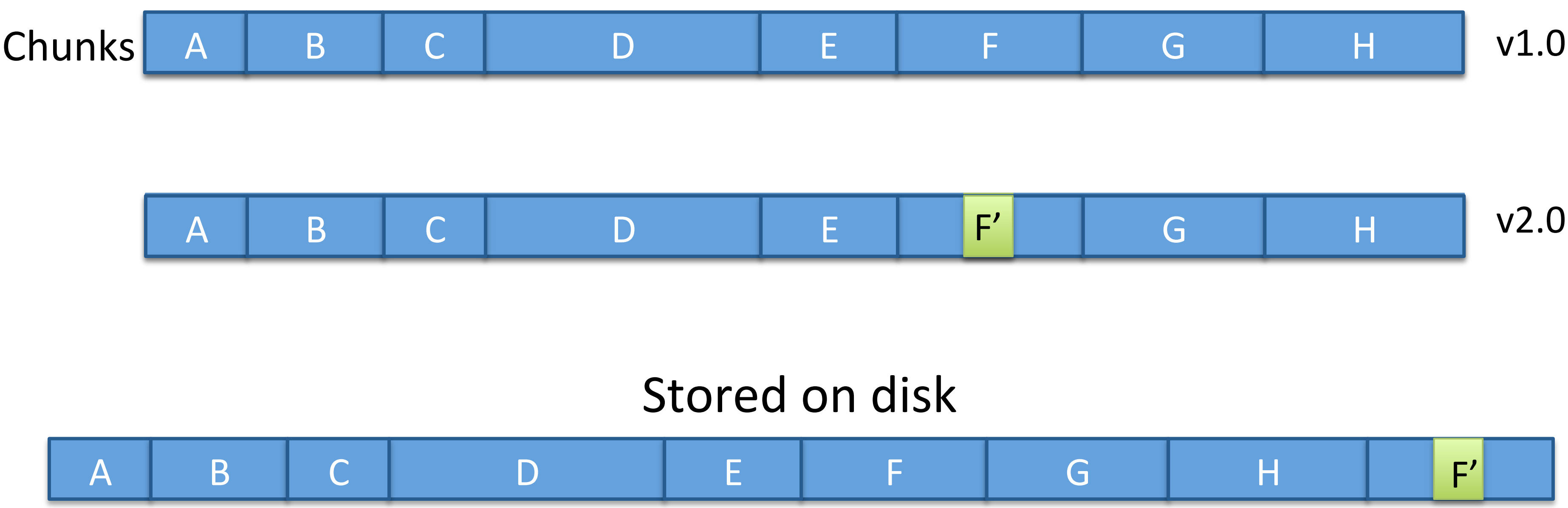
Deduplication

Idea: identify duplicate data blocks and store a single copy



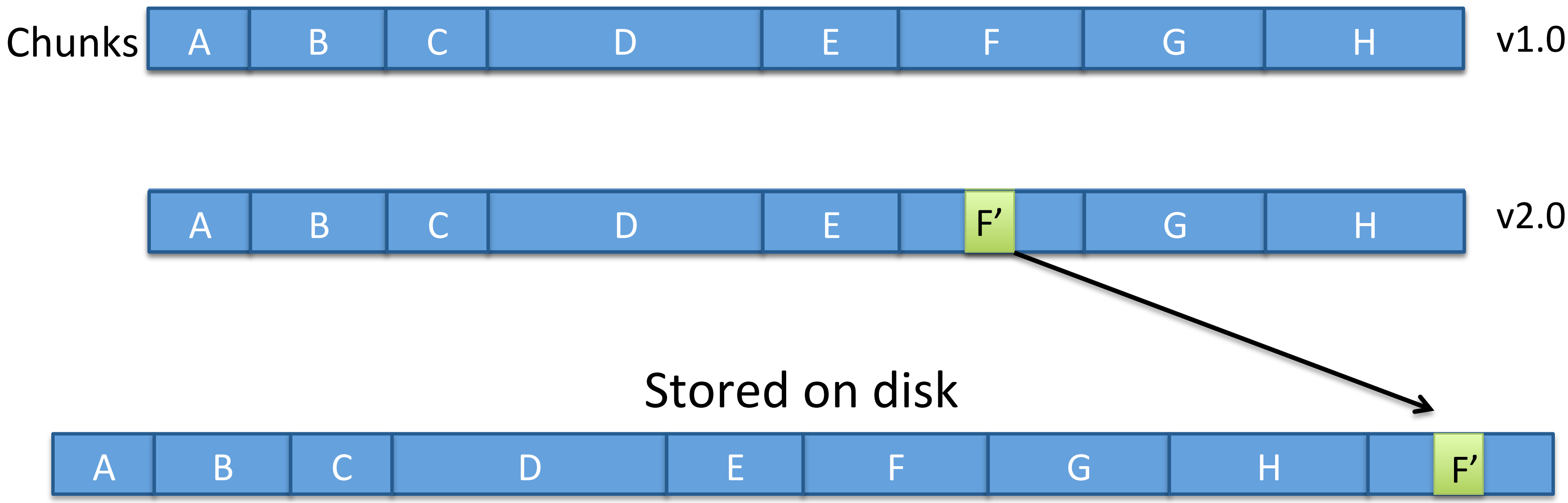
Deduplication

Idea: identify duplicate data blocks and store a single copy



Deduplication

Idea: identify duplicate data blocks and store a single copy



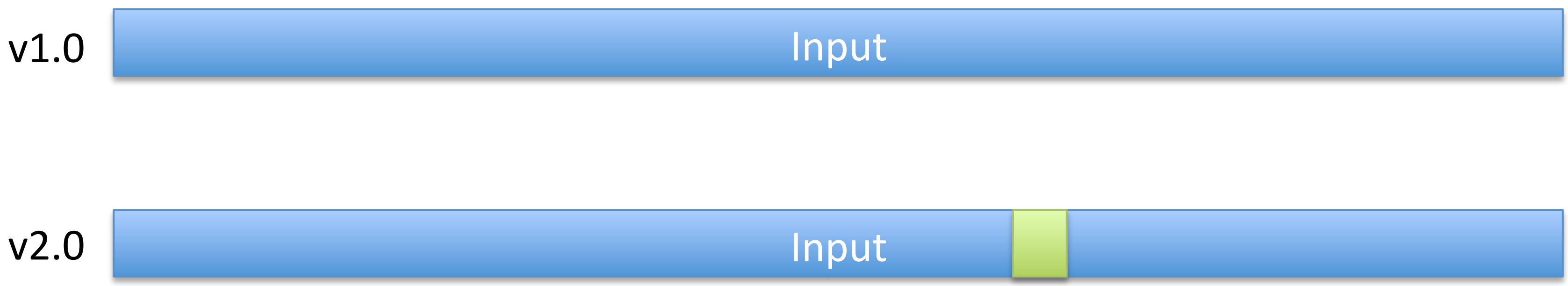
Eliminate nearly all of v2.0 on disk

Expected Deduplication

What if we have many versions?

Expected Deduplication

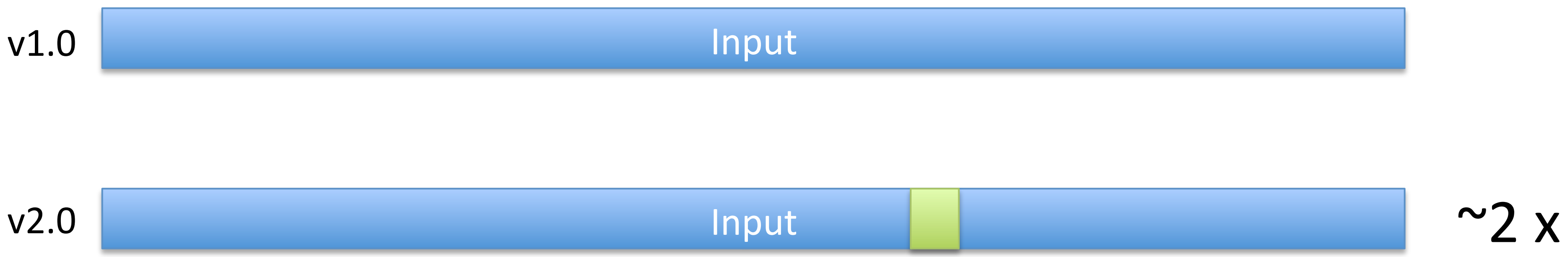
What if we have many versions?



Expected Deduplication

What if we have many versions?

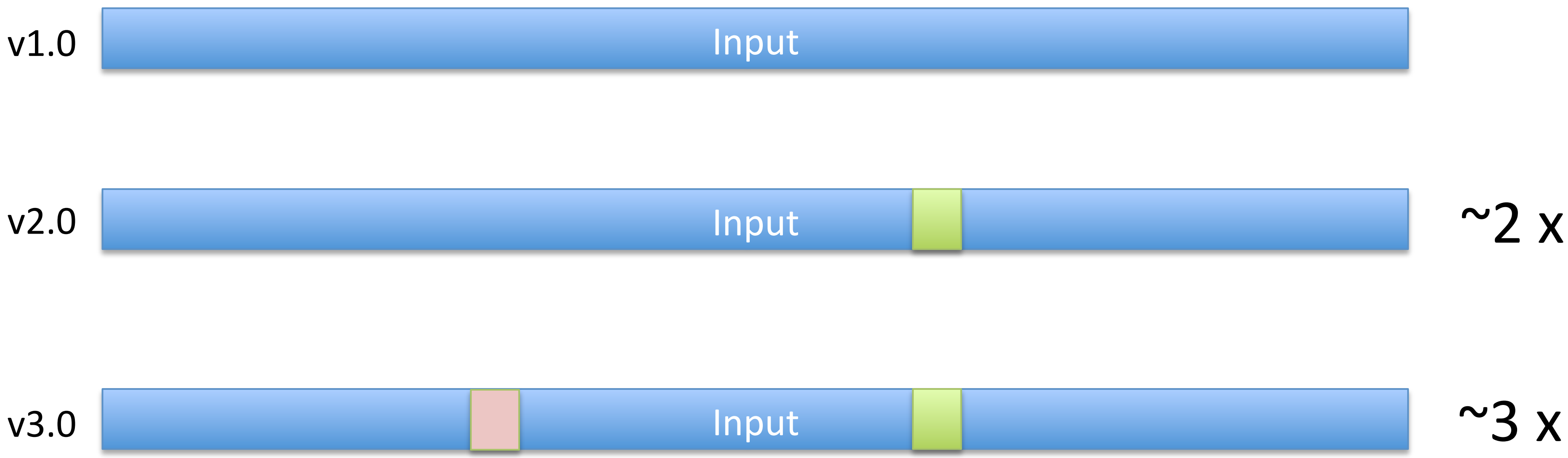
Dedup ratio = original_size/post_dedup_size



Expected Deduplication

What if we have many versions?

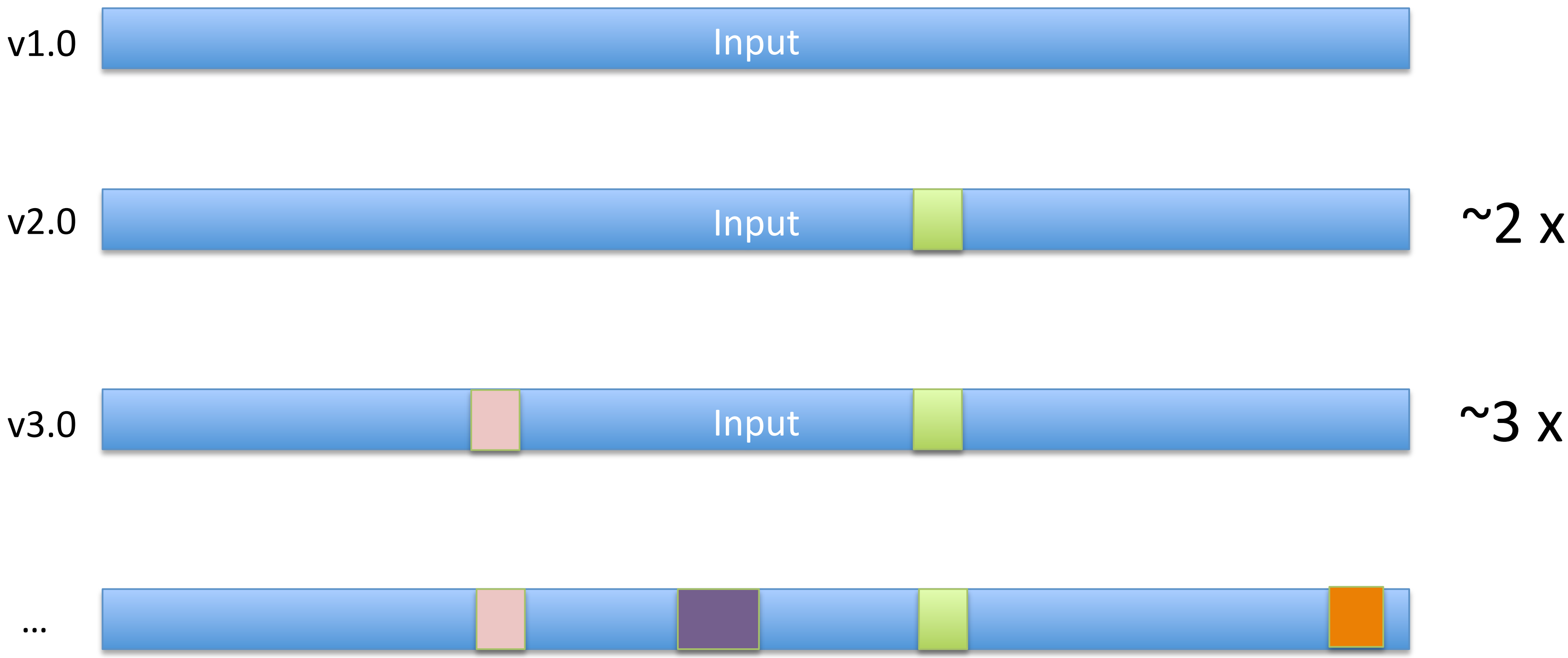
Dedup ratio = original_size/post_dedup_size



Expected Deduplication

What if we have many versions?

Dedup ratio = original_size/post_dedup_size



Great (Deduplication) Expectations

- ***In Reality***
  - ***308*** versions of Linux source code: ***2 x***
  - Other examples of awful deduplication

Great (Deduplication) Expectations

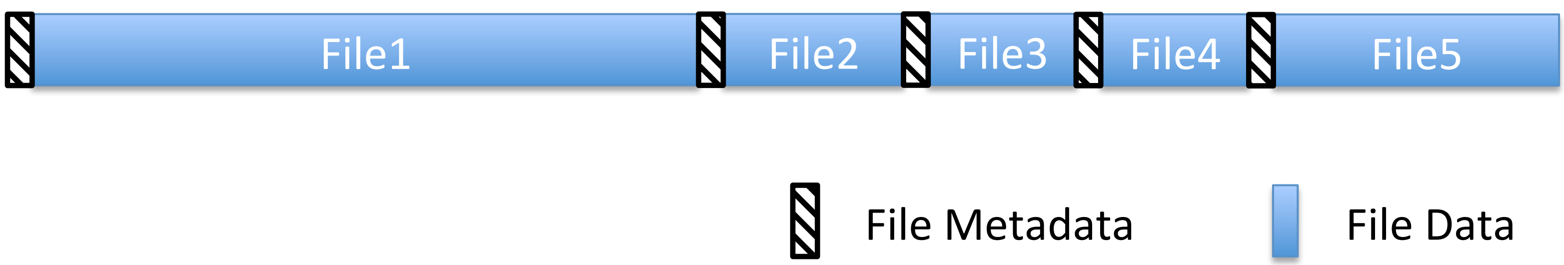
- ***In Reality***
  - ***308*** versions of Linux source code: ***2 x***
  - Other examples of awful deduplication
- ***Contributions***
  - Identify and categorize barriers to deduplication
  - Solutions
    - EMC Data Domain (industrial experience)
    - GNU tar (academic research)

Metadata Impacts Deduplication

- Case 1: metadata changes
  - The input is an aggregate of many small user files, interleaved with file metadata

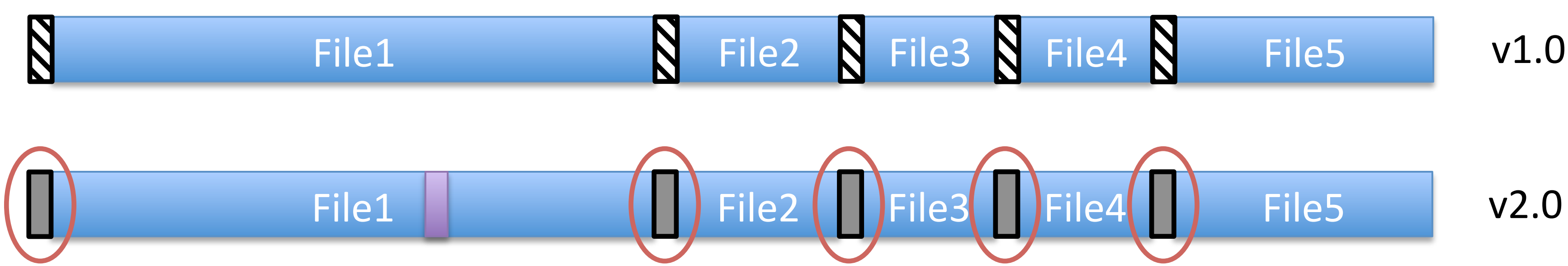
Metadata Impacts Deduplication

- Case 1: metadata changes
  - The input is an aggregate of many small user files, interleaved with file metadata



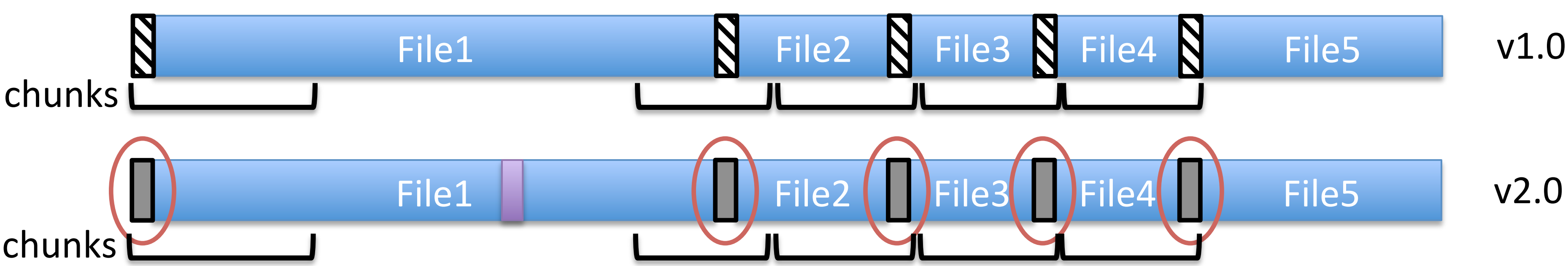
Metadata Impacts Deduplication

- Case 1: metadata changes
  - The input is an aggregate of many small user files, interleaved with file metadata



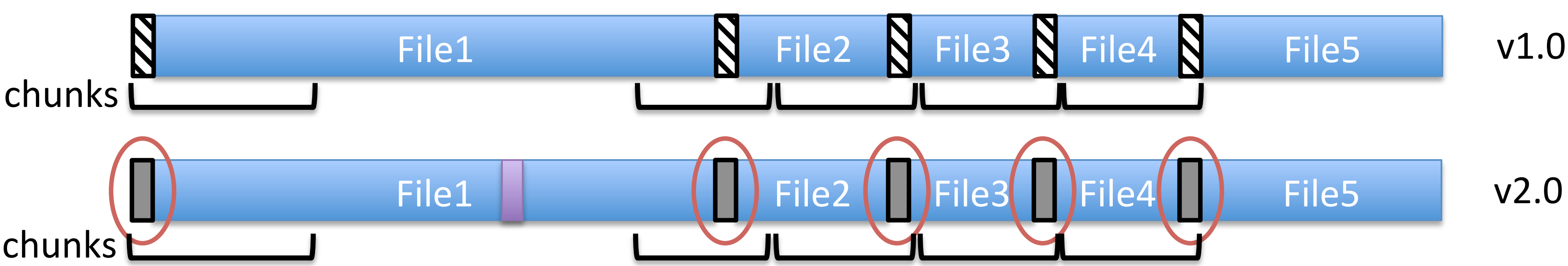
Metadata Impacts Deduplication

- Case 1: metadata changes
  - The input is an aggregate of many small user files, interleaved with file metadata



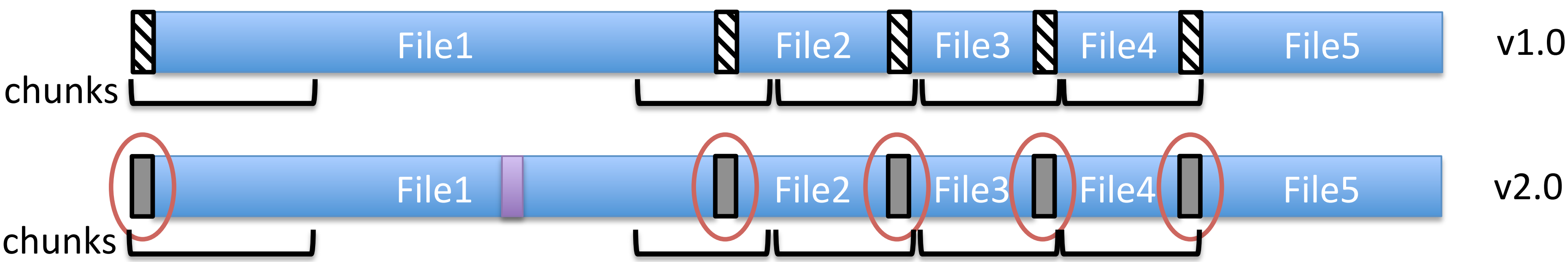
Metadata Impacts Deduplication

- Case 1: metadata changes
  - The input is an aggregate of many small user files, interleaved with file metadata
  - GNU tar, EMC NetWorker, Oracle RMAN backup



Metadata Impacts Deduplication

- Case 1: metadata changes
  - The input is an aggregate of many small user files, interleaved with file metadata
  - GNU tar, EMC NetWorker, Oracle RMAN backup
  - Videos suffer from a similar problem ([Dewakar HotStorage15])



Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block

Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts

Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts
  - Tape format

Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts
  - Tape format

v1.0

Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts
  - Tape format

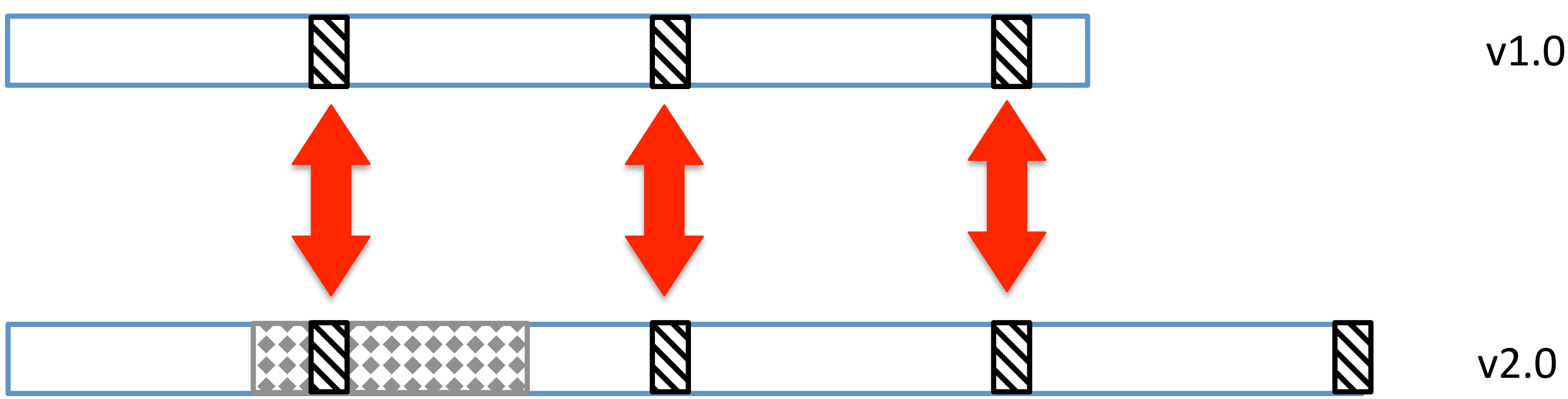
v1.0

Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts
  - Tape format

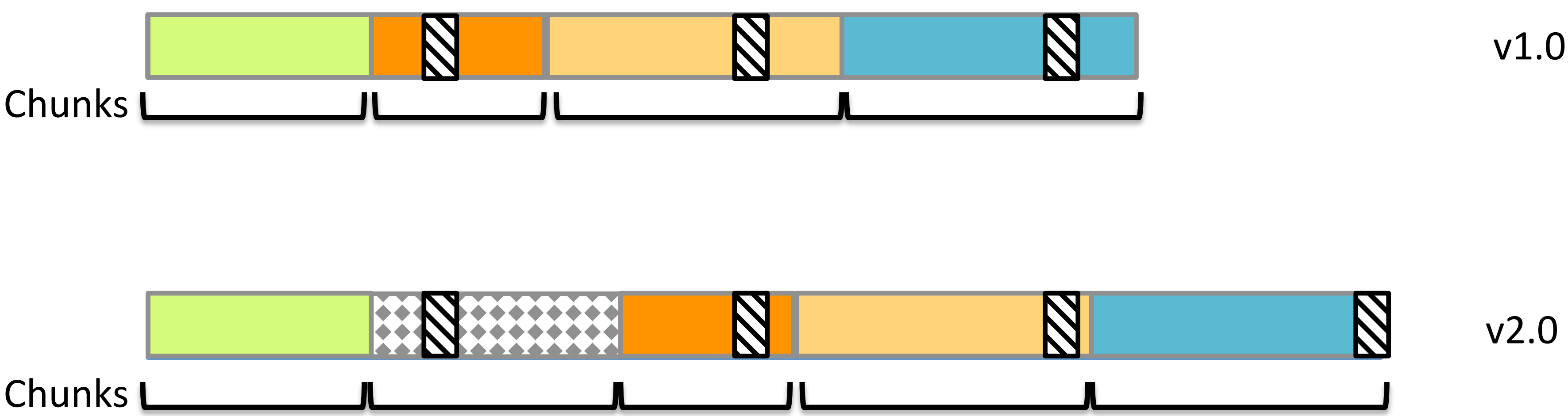
Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts
  - Tape format



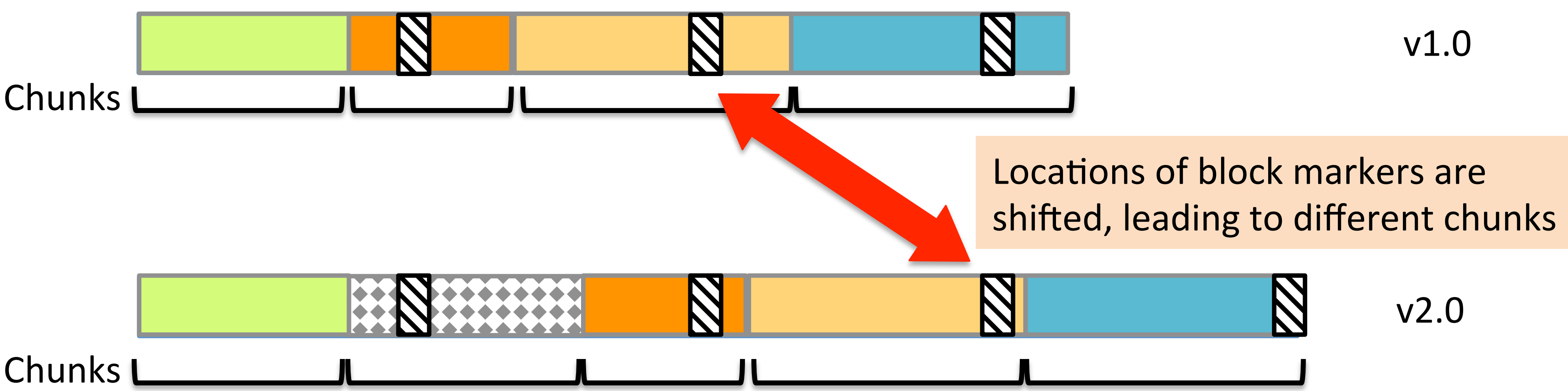
Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts
  - Tape format



Metadata Impacts Deduplication

- Case 2: metadata location changes
  - The input is encoded in (fixed-size) blocks and metadata is inserted for each block
  - Data insertion/deletion leads to metadata shifts
  - Tape format



Solution: Separate Metadata from Data

Solution: Separate Metadata from Data

- Three approaches:
  - ***Recommended***: design deduplication-friendly formats
    - Case study: EMC NetWorker

Solution: Separate Metadata from Data

- Three approaches:
  - ***Recommended***: design deduplication-friendly formats
    - Case study: EMC NetWorker
  - ***Transparent***: application-level post-processing
    - Case study: GNU tar

Solution: Separate Metadata from Data

- Three approaches:
  - ***Recommended***: design deduplication-friendly formats
    - Case study: EMC NetWorker
  - ***Transparent***: application-level post-processing
    - Case study: GNU tar
  - ***Last resort***: format-aware deduplication
    - Case studies: 1) virtual tape library (VTL)
      2) Oracle RMAN backup

Solution: Separate Metadata from Data

- Three approaches:
  - ***Recommended***: design deduplication-friendly formats
    - Case study: EMC NetWorker
  - ***Transparent***: application-level post-processing
    - ***Case study: GNU tar***
  - ***Last resort***: format-aware deduplication
    - Case studies: 1) virtual tape library (VTL)

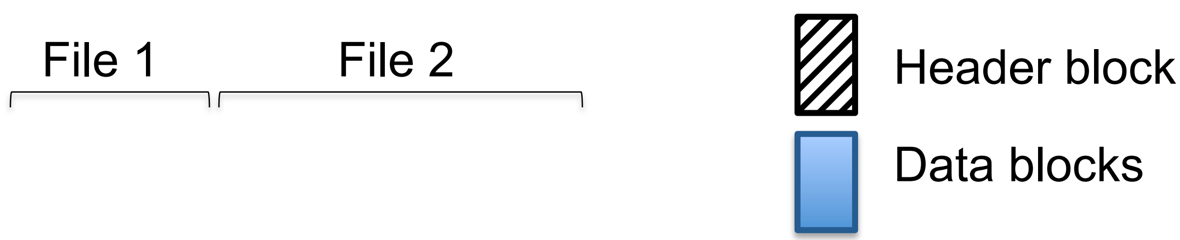
      2) Oracle RMAN backup

Application-level Post-processing

- tar (*t*ape *ar*chive)
  - Collects files into one archive file
  - File system archiving, source code distribution, …
- GNU tar format
  - A sequence of entries, one per file
  - For each file: a file header and data blocks
  - Header block: file path, size, modification time

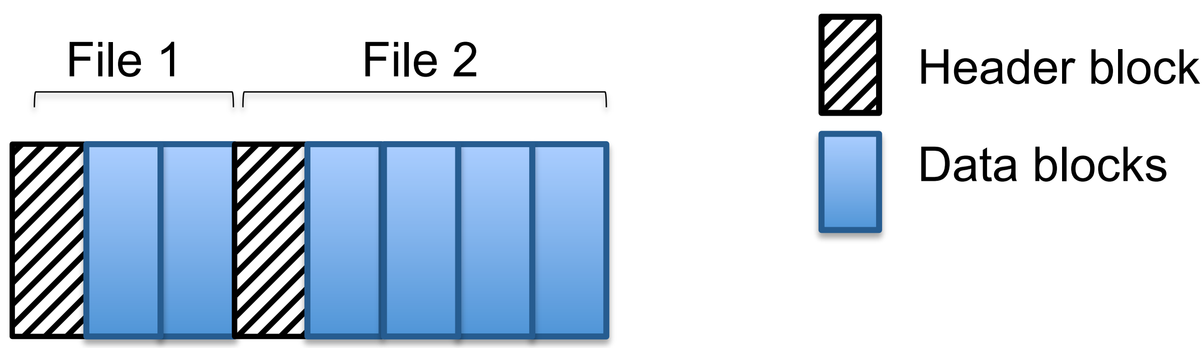
Application-level Post-processing

- tar (*t*ape *ar*chive)
  - Collects files into one archive file
  - File system archiving, source code distribution, …
- GNU tar format
  - A sequence of entries, one per file
  - For each file: a file header and data blocks
  - Header block: file path, size, modification time



Application-level Post-processing

- tar (*t*ape *ar*chive)
  - Collects files into one archive file
  - File system archiving, source code distribution, …
- GNU tar format
  - A sequence of entries, one per file
  - For each file: a file header and data blocks
  - Header block: file path, size, modification time



Metadata Changes with GNU tar

Metadata Changes with GNU tar

SHA1s	linux-2.6.39.3/README	a735c31cef6d19d56de6824131527fdce04ead47
	linux-2.6.39.4/README	a735c31cef6d19d56de6824131527fdce04ead47

Metadata Changes with GNU tar

SHA1s	linux-2.6.39.3/README	a735c31cef6d19d56de6824131527fdce04ead47
	linux-2.6.39.4/README	a735c31cef6d19d56de6824131527fdce04ead47

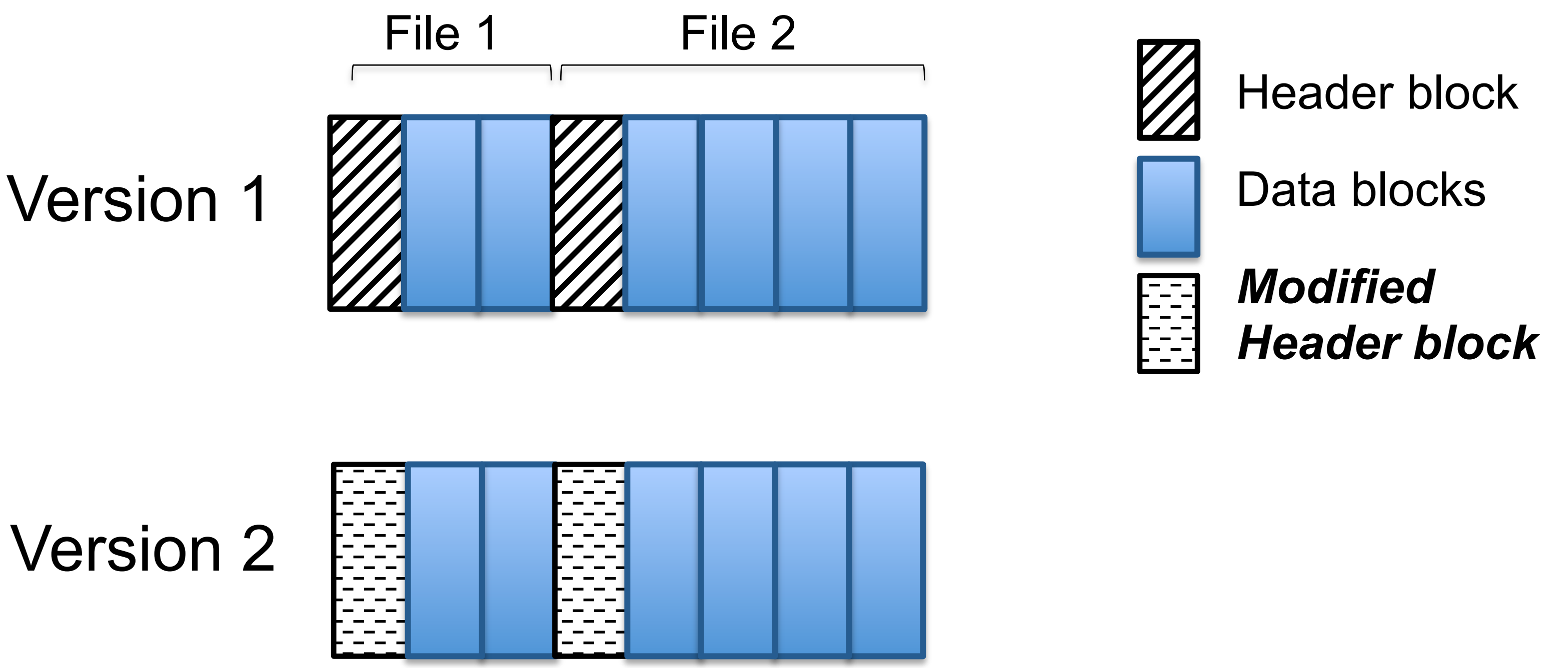
```
linux-2.6.39.3/README  -rw-rw-r-- 1 root root 17525 Jul   9  2011
linux-2.6.39.4/README  -rw-rw-r-- 1 root root 17525 Aug  3  2011
```

Metadata Changes with GNU tar

SHA1s

linux-2.6.39.3/README a735c31cef6d19d56de6824131527fdce04ead47
linux-2.6.39.4/README a735c31cef6d19d56de6824131527fdce04ead47

linux-2.6.39.3/README -rw-rw-r-- 1 root root 17525 Jul 9 2011
linux-2.6.39.4/README -rw-rw-r-- 1 root root 17525 Aug 3 2011

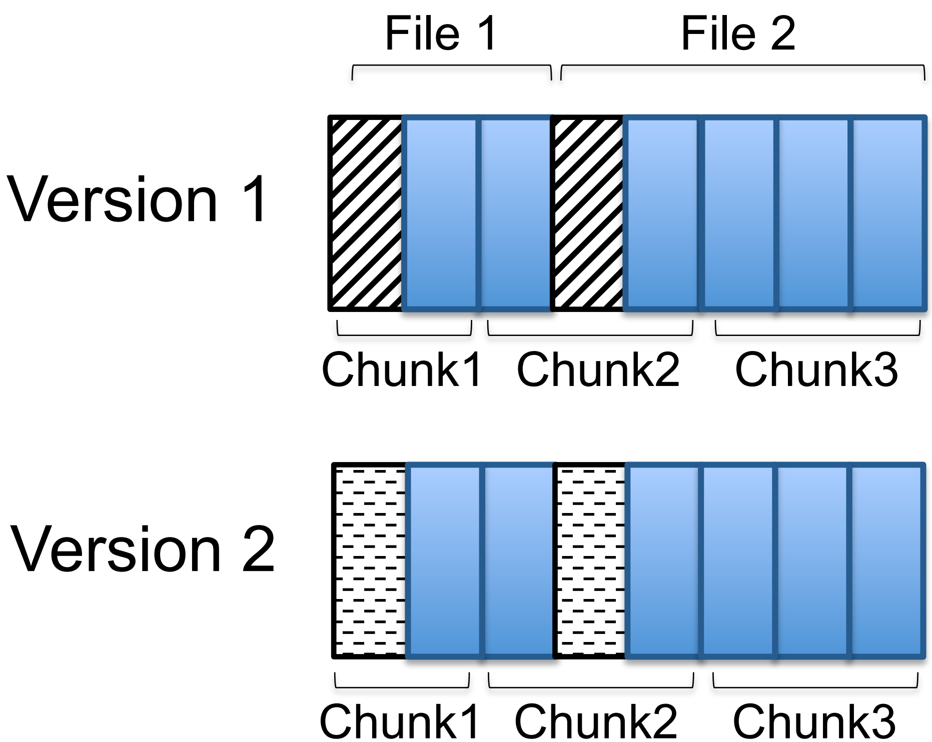


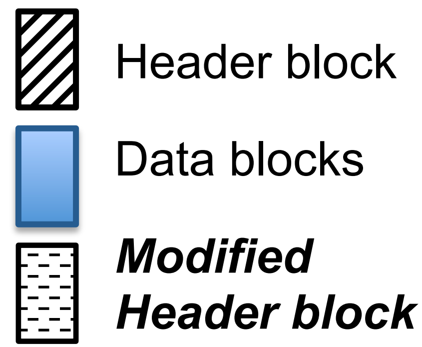
Metadata Changes with GNU tar

SHA1s

linux-2.6.39.3/README	a735c31cef6d19d56de6824131527fdce04ead47
linux-2.6.39.4/README	a735c31cef6d19d56de6824131527fdce04ead47

linux-2.6.39.3/README	-rw-rw-r--	1	root	root	17525	Jul 9 2011
linux-2.6.39.4/README	-rw-rw-r--	1	root	root	17525	Aug 3 2011





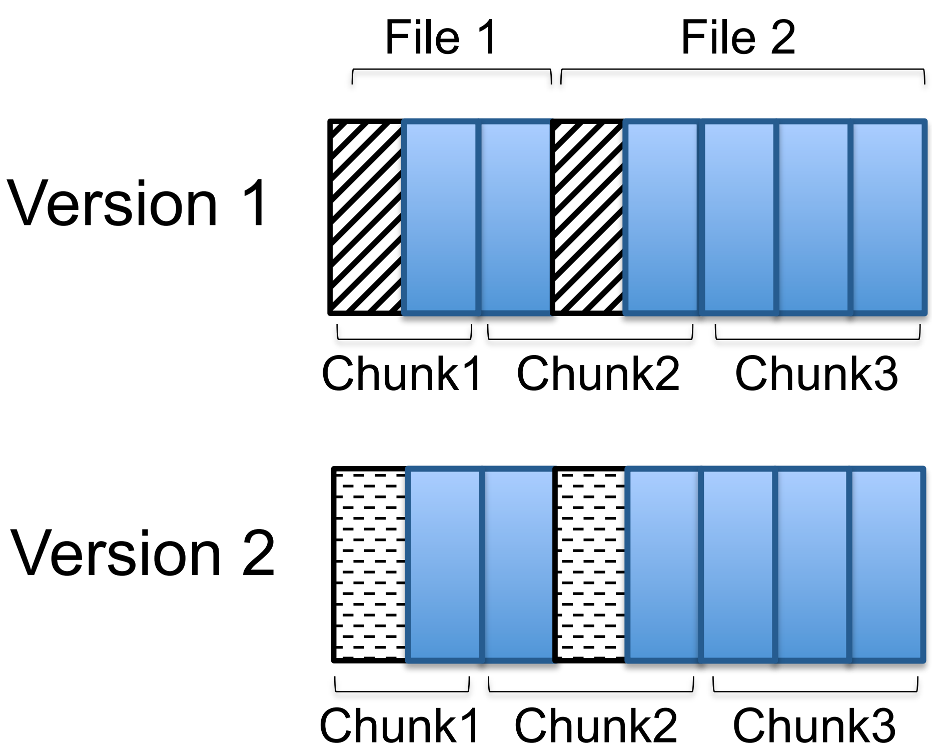
Chunk1 and Chunk2 in version 2 are ***different.***

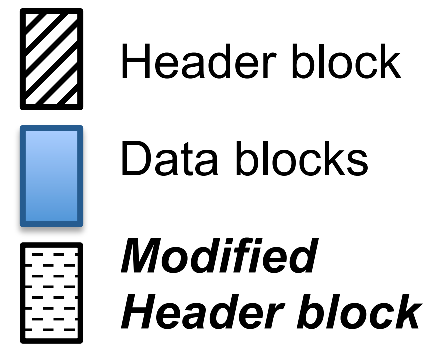
Metadata Changes with GNU tar

SHA1s

linux-2.6.39.3/README a735c31cef6d19d56de6824131527fdce04ead47
linux-2.6.39.4/README a735c31cef6d19d56de6824131527fdce04ead47

linux-2.6.39.3/README -rw-rw-r-- 1 root root 17525 Jul 9 2011
linux-2.6.39.4/README -rw-rw-r-- 1 root root 17525 Aug 3 2011

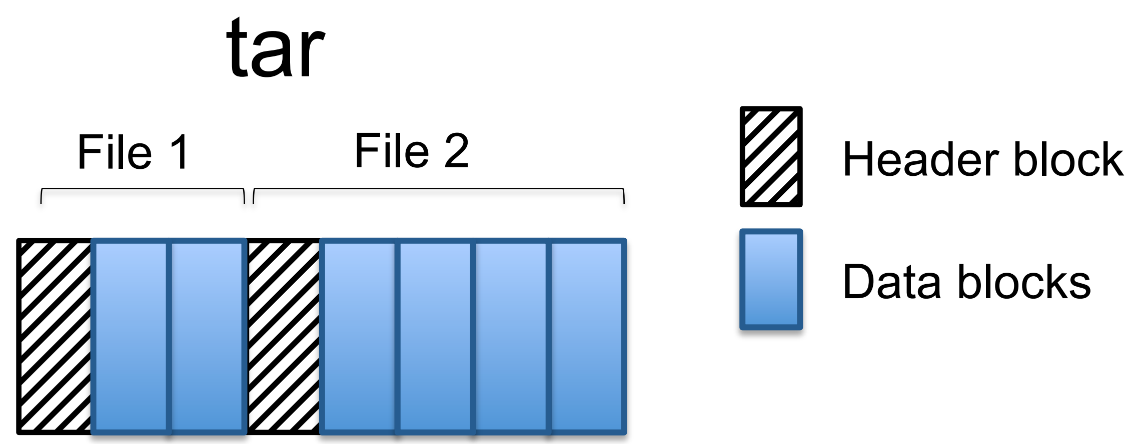




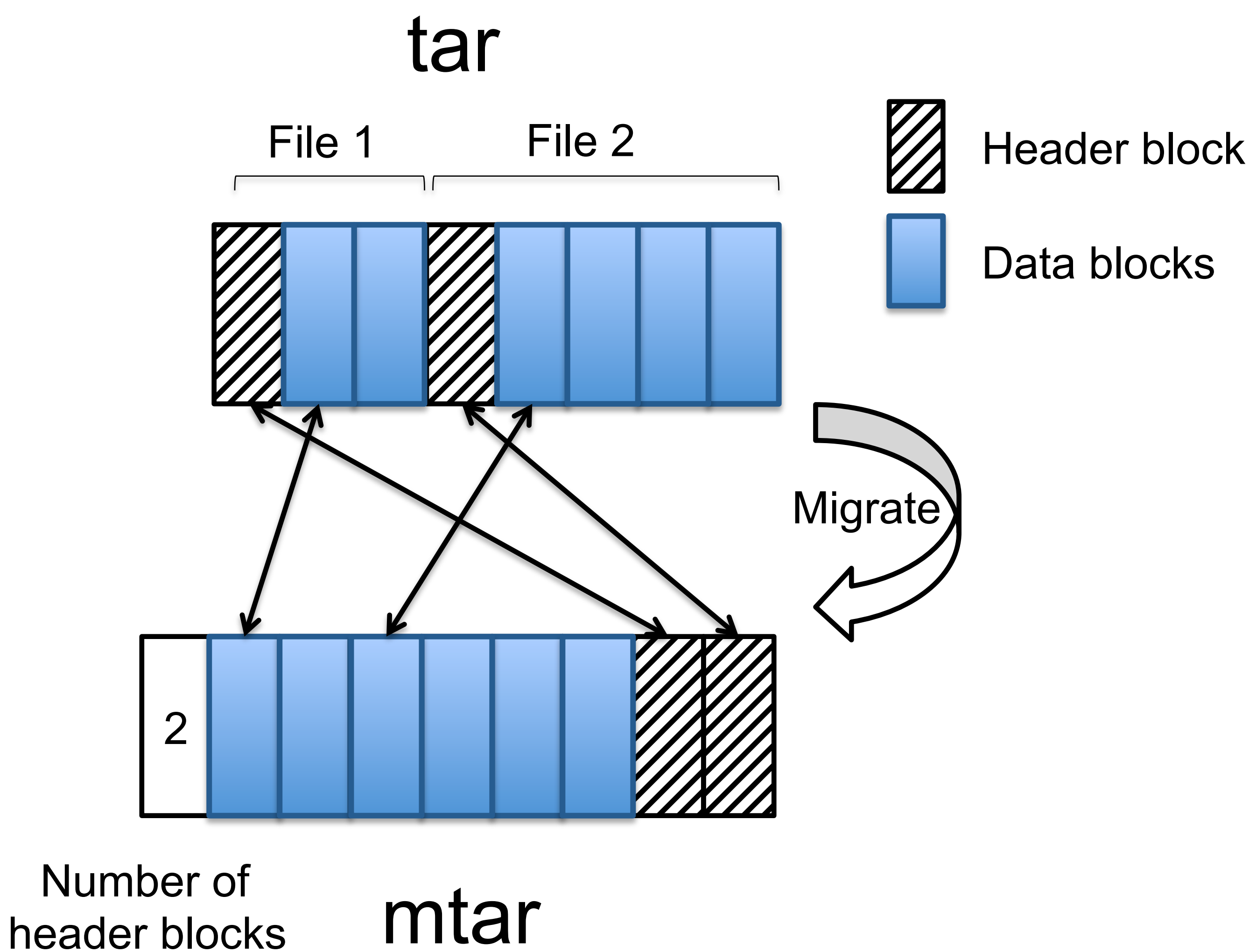
Chunk1 and Chunk2 in version 2 are ***different.***

2 x for 308 versions of Linux

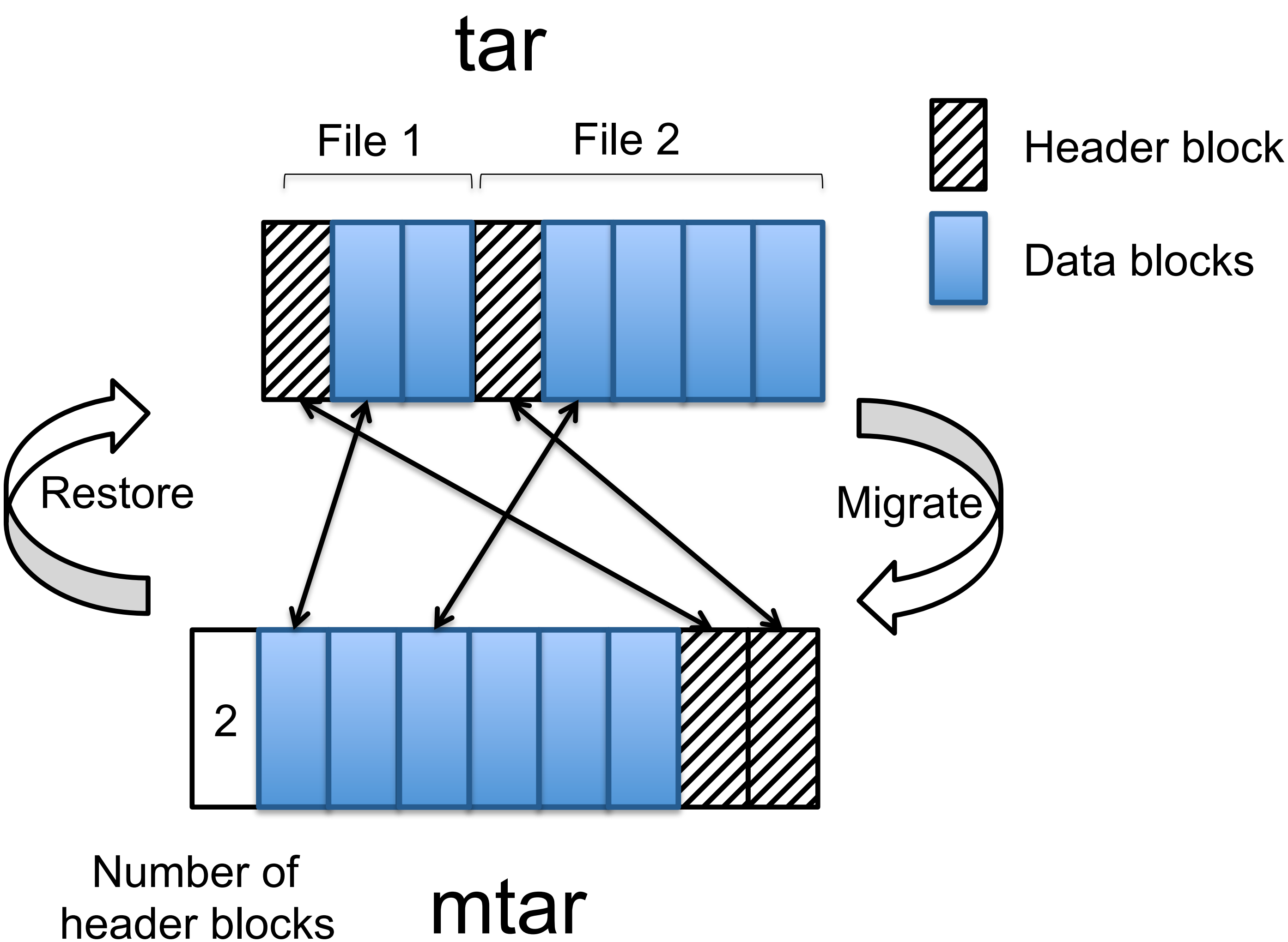
Migratory tar (mtar)



Migratory tar (mtar)



Migratory tar (mtar)

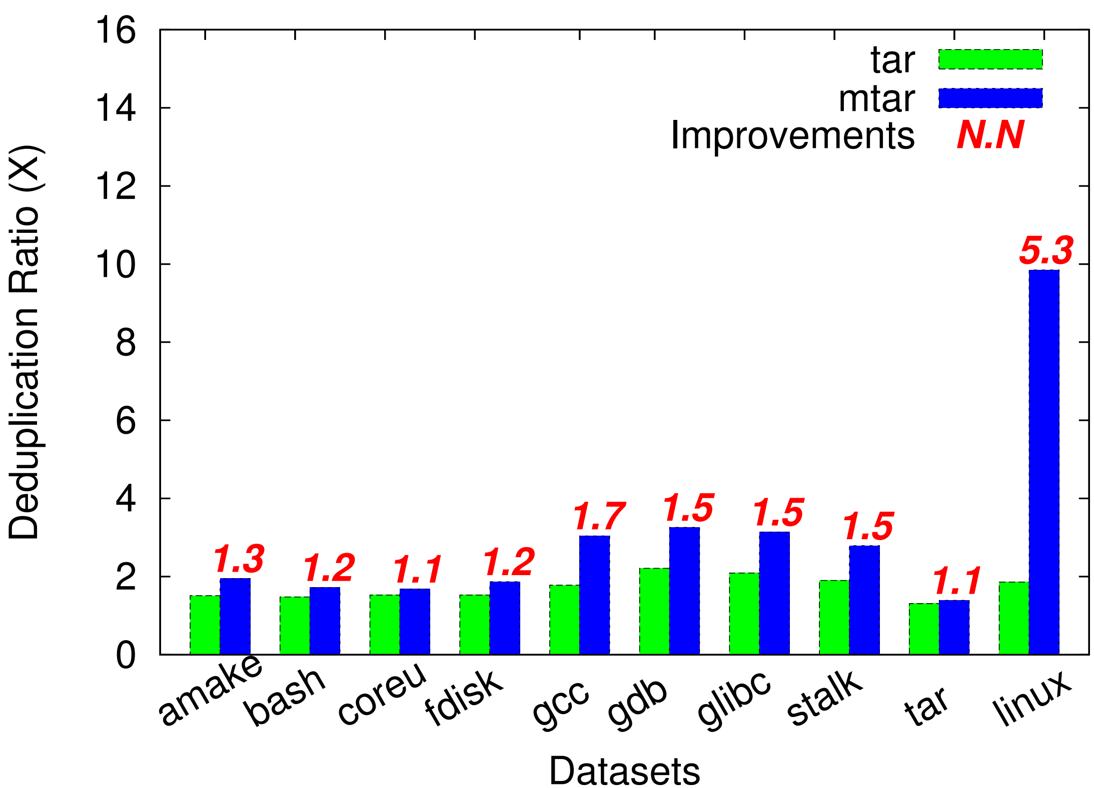


mtar - Evaluation

- 9 GNU software and Linux kernel source code
- Many versions: 13 ~ 308

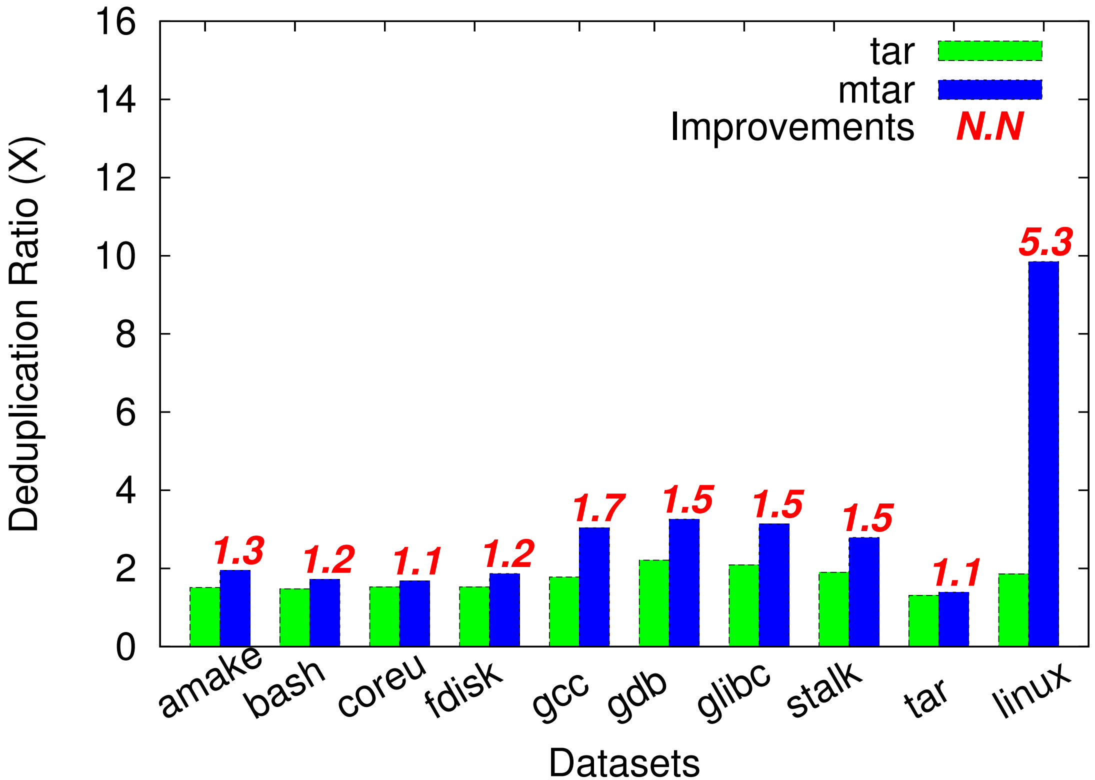
mtar - Evaluation

- 9 GNU software and Linux kernel source code
- Many versions: 13 ~ 308



mtar - Evaluation

- 9 GNU software and Linux kernel source code
- Many versions: 13 ~ 308



Improvements:
1. Across all datasets
2. Huge: 1.5-5.3×

More in the Paper

- ***Design deduplication-friendly formats***
  - ***Case study: EMC NetWorker***
- Application-level post-processing
  - Case study: GNU tar
- ***Format-aware Deduplication***
  - ***Case studies: 1) virtual tape library***
    2) Oracle RMAN backup

Summary

- Metadata impacts deduplication
  - Metadata changes more frequently, introducing many *unnecessary* unique chunks
- Solution: separate metadata from data
  - Up to 5× improvements in deduplication

Metadata Considered Harmful … to Deduplication
Xing Lin, Fred Douglis, Jim Li, Xudong Li, Robert Ricci, Stephen Smaldone and Grant Wallance
HotStorage '15: 7th USENIX Workshop on Hot Topics in Storage and File Systems

Outline

- ✓ Introduction
  - ✓ Migratory Compression
  - ✓ Improve deduplication by separating metadata from data
  - ✓ ***Using deduplication for efficient disk image deployment***
  - ✓ Performance predictability and efficiency for Cloud Storage Systems
- ✓ Conclusion

Introduction

- Cloud providers or network testbeds maintain a large number of operating system images (disk images)
  - Amazon Web Service (AWS): 37,000+ images
  - Emulab: 1020+

Introduction

- Cloud providers or network testbeds maintain a large number of operating system images (disk images)
  - Amazon Web Service (AWS): 37,000+ images
  - Emulab: 1020+
- Disk image: a snapshot of a disk's content
  - Several to hundreds of GBs.
  - ***Similarity*** across disk images ([Jin, SYSTOR12])

Introduction

- Cloud providers or network testbeds maintain a large number of operating system images (disk images)
  - Amazon Web Service (AWS): 37,000+ images
  - Emulab: 1020+
- Disk image: a snapshot of a disk's content
  - Several to hundreds of GBs.
  - ***Similarity*** across disk images ([Jin, SYSTOR12])

Requirements: efficient image storage

What is Image Deployment?

The process of distributing and installing a disk image at multiple compute nodes

What is Image Deployment?

The process of distributing and installing a disk image at multiple compute nodes

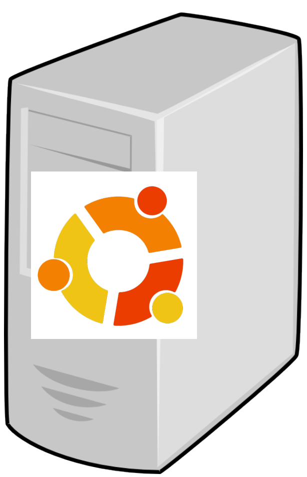
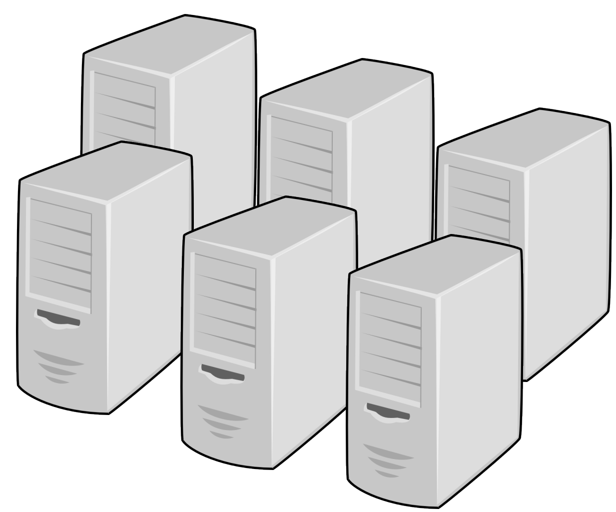


Image server



Compute Nodes

What is Image Deployment?

The process of distributing and installing a disk image at multiple compute nodes

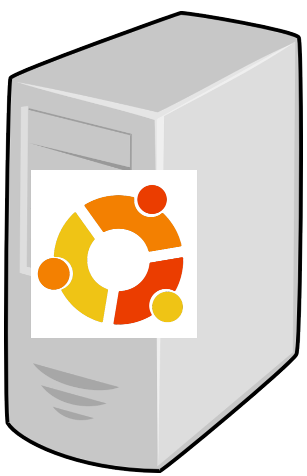


Image server

Compute Nodes

What is Image Deployment?

The process of distributing and installing a disk image at multiple compute nodes

Requirements: efficient image transmission and installation

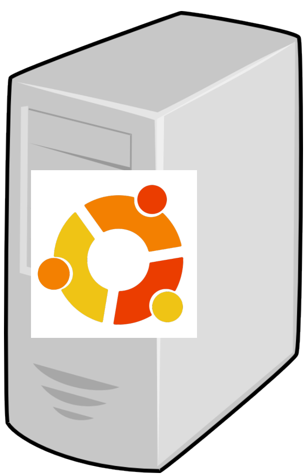


Image server

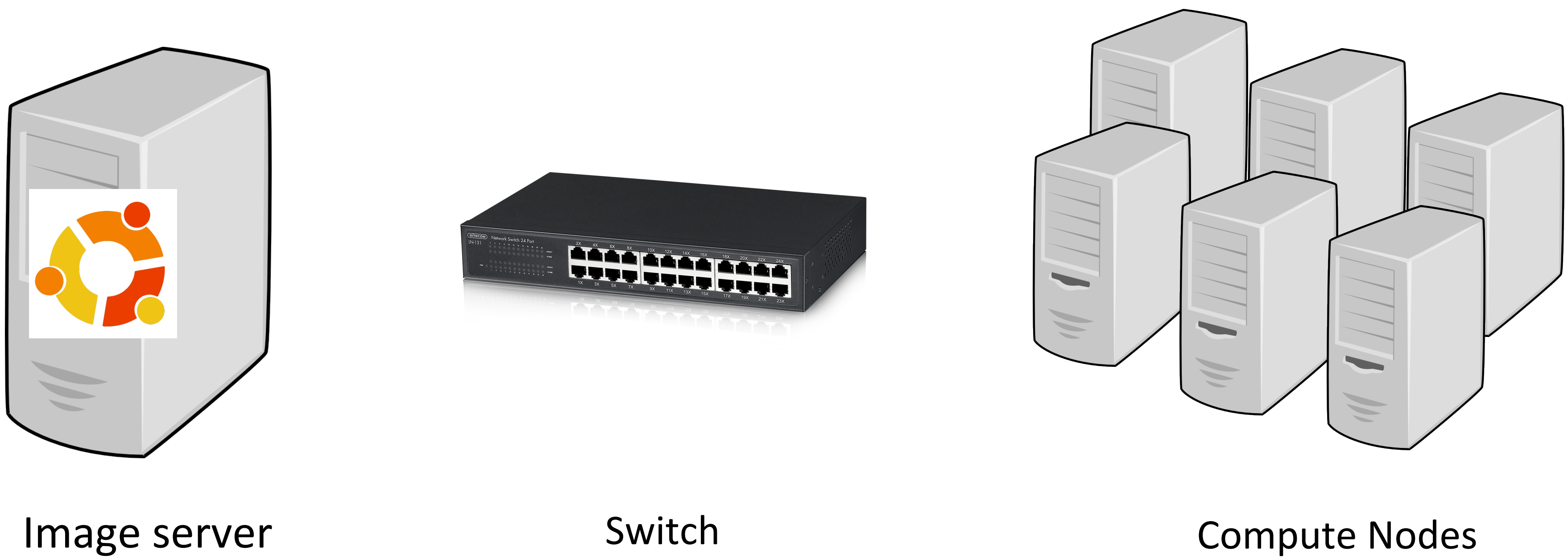
Compute Nodes

Existing Systems: Frisbee and Venti

- Frisbee: an efficient image deployment system ([Hibler ATC03])
- Venti: a deduplicating storage system ([Quinlan FAST02])

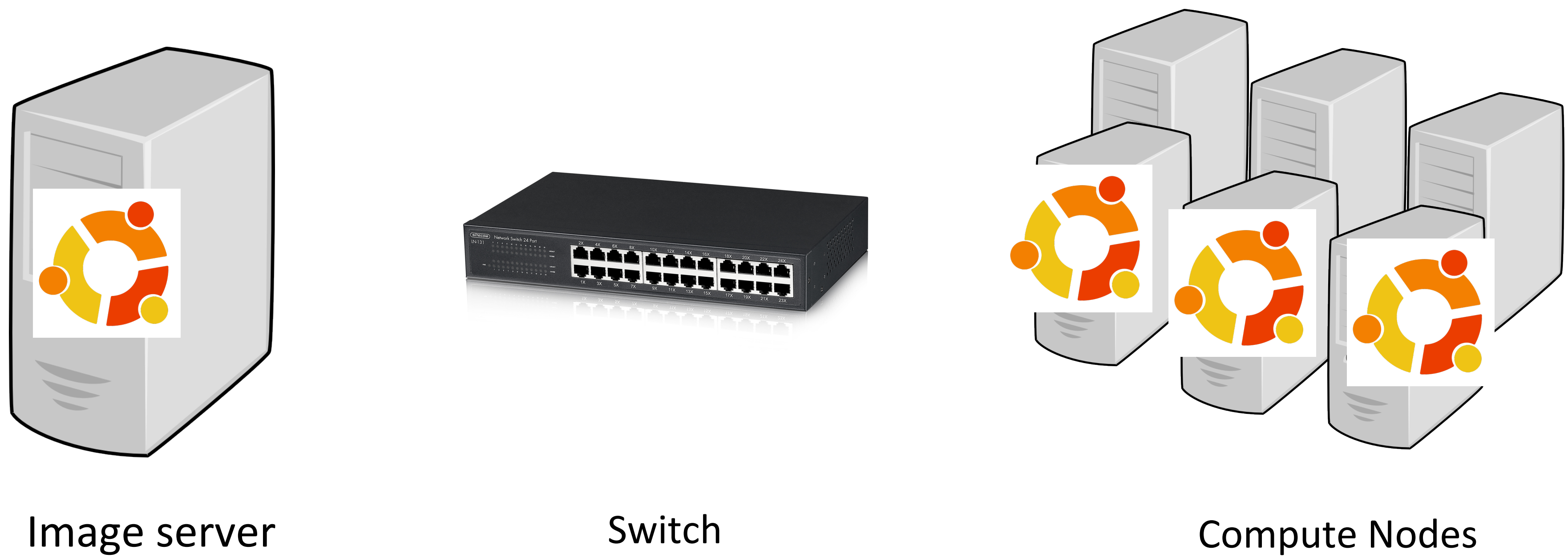
Existing Systems: Frisbee and Venti

- Frisbee: an efficient image deployment system ([Hibler ATC03])
- Venti: a deduplicating storage system ([Quinlan FAST02])



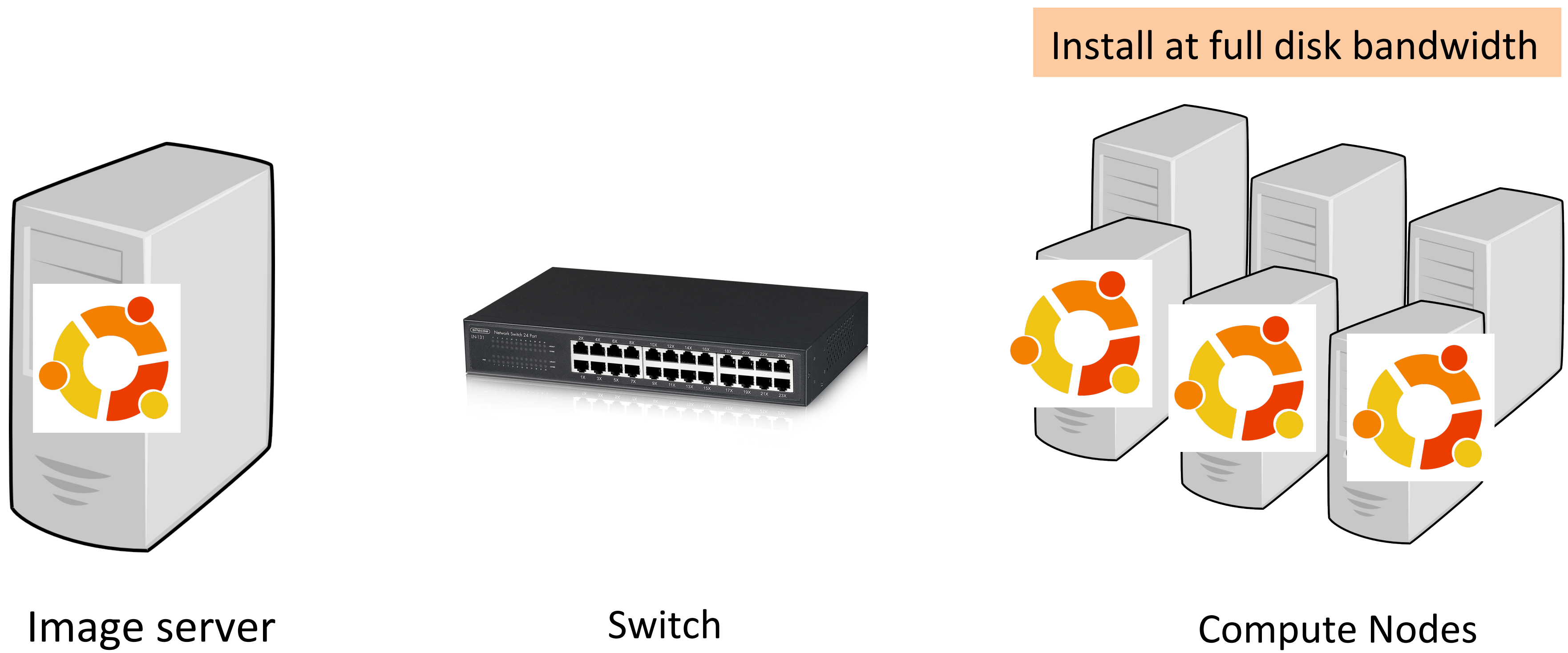
Existing Systems: Frisbee and Venti

- Frisbee: an efficient image deployment system ([Hibler ATC03])
- Venti: a deduplicating storage system ([Quinlan FAST02])



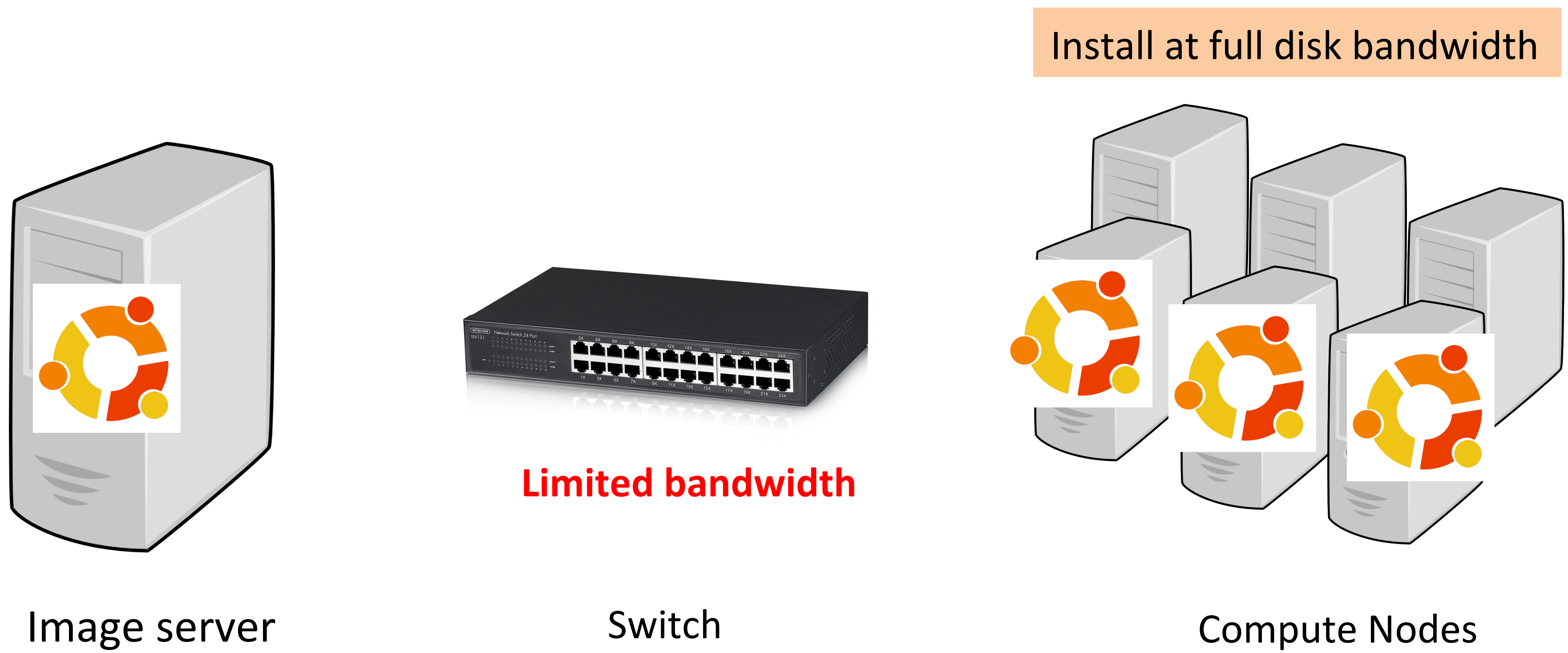
Existing Systems: Frisbee and Venti

- Frisbee: an efficient image deployment system ([Hibler ATC03])
- Venti: a deduplicating storage system ([Quinlan FAST02])



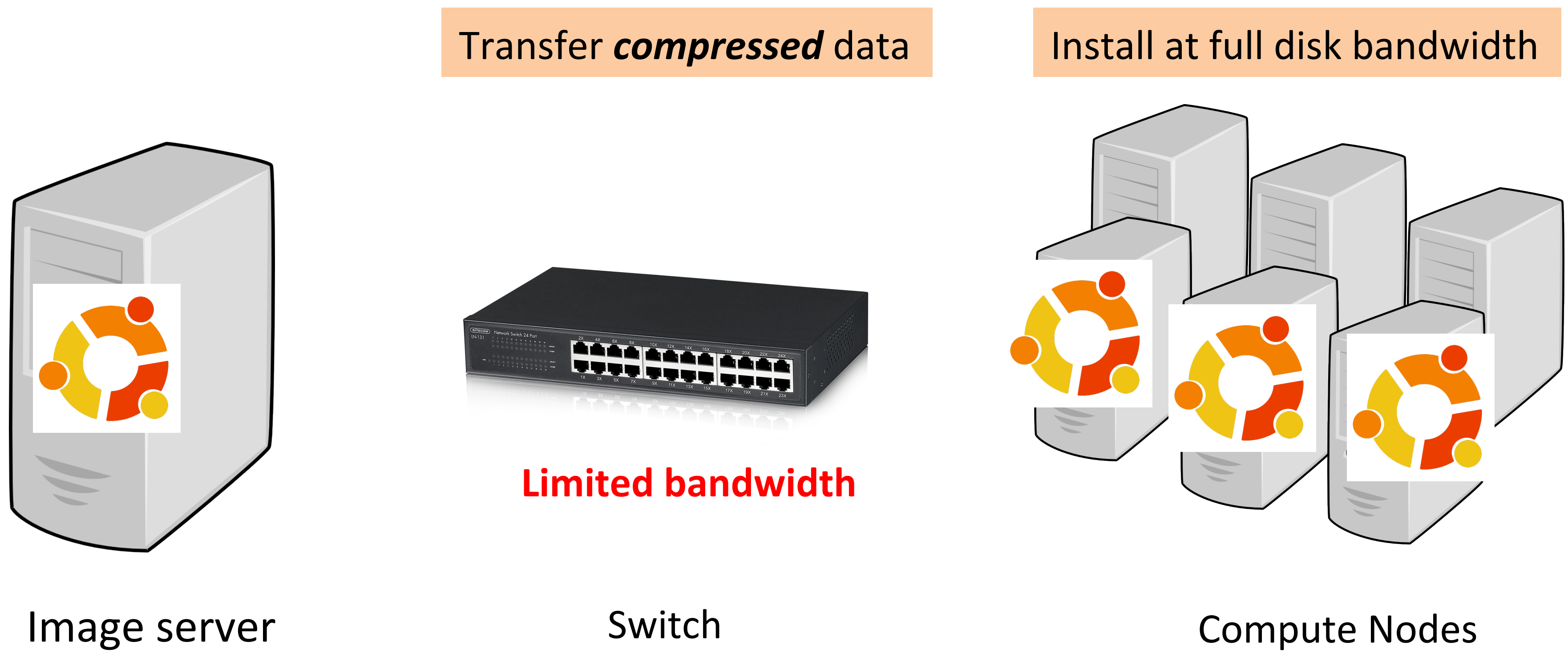
Existing Systems: Frisbee and Venti

- Frisbee: an efficient image deployment system ([Hibler ATC03])
- Venti: a deduplicating storage system ([Quinlan FAST02])



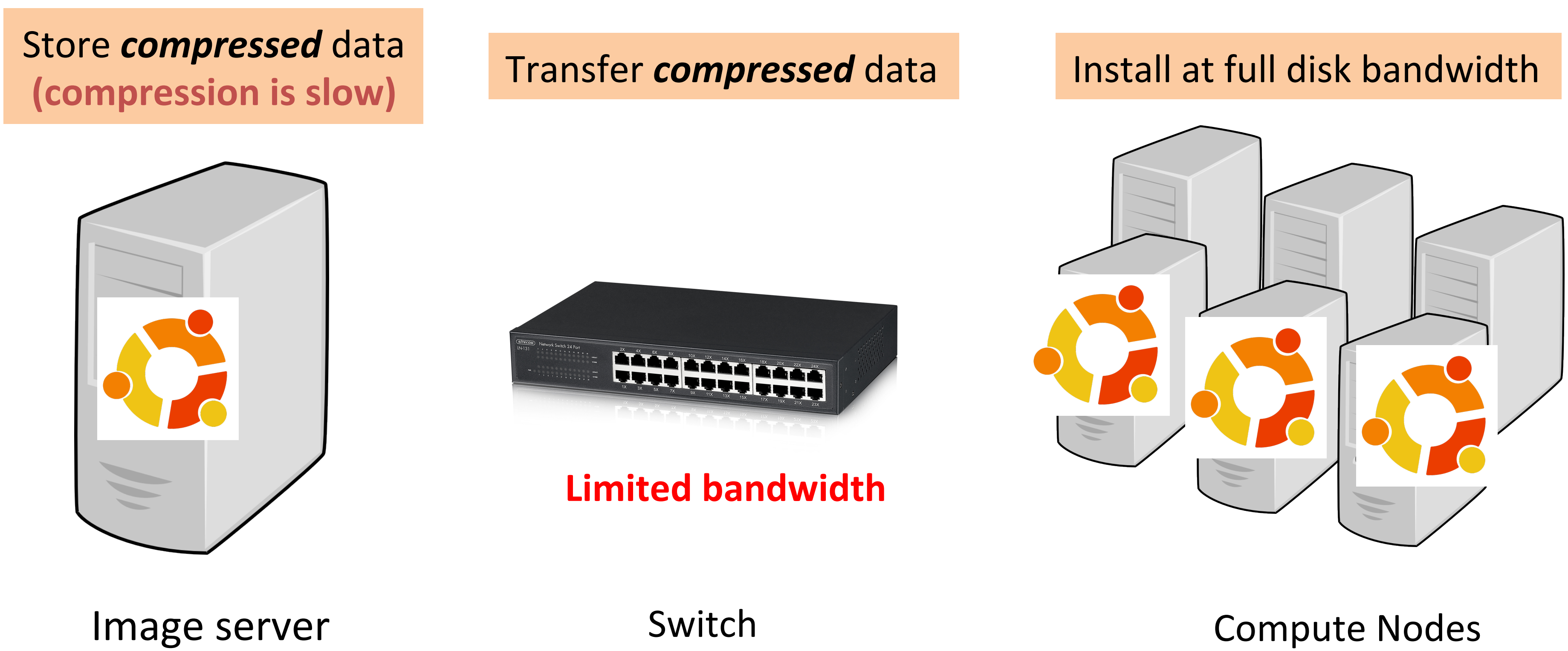
Existing Systems: Frisbee and Venti

- Frisbee: an efficient image deployment system ([Hibler ATC03])
- Venti: a deduplicating storage system ([Quinlan FAST02])

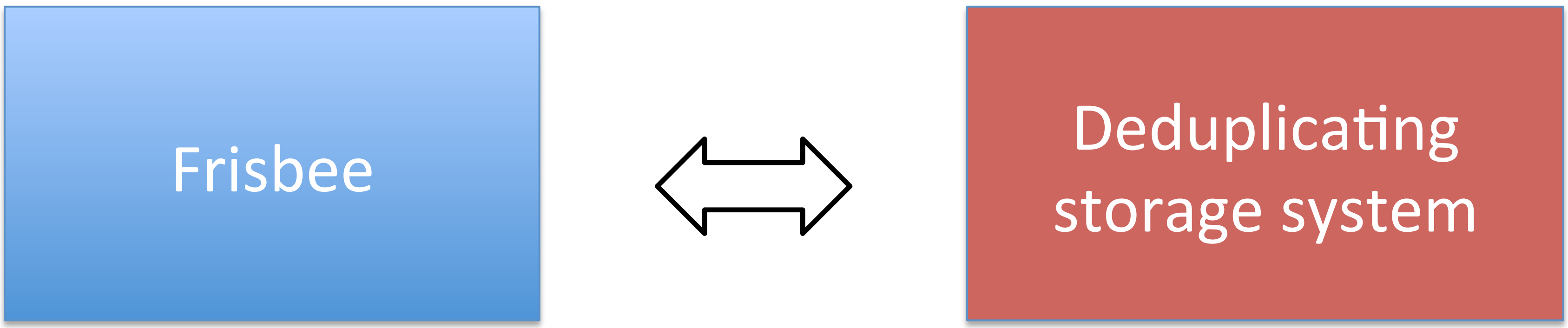


Existing Systems: Frisbee and Venti

- Frisbee: an efficient image deployment system ([Hibler ATC03])
- Venti: a deduplicating storage system ([Quinlan FAST02])



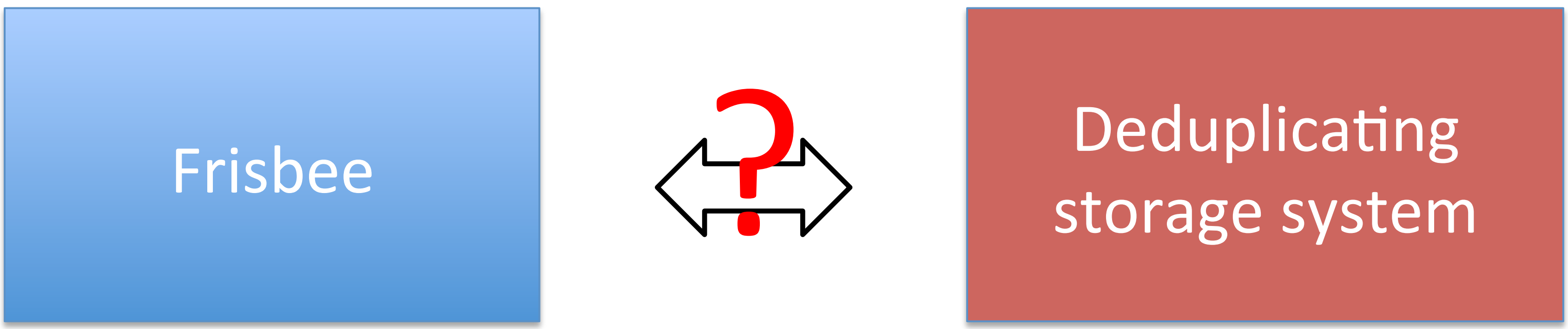
Integrate Frisbee with Deduplicating Storage



Efficient Image Deployment

Efficient Image Storage

Integrate Frisbee with Deduplicating Storage



Efficient Image Deployment

Efficient Image Storage

How to achieve efficient image deployment and storage ***simultaneously***, by integrating these two systems?

Requirements for the Integration

Use compression

Use filesystem to skip unallocated data

Image installation at full disk bandwidth

Independent installable Frisbee chunks

Requirements

Solutions

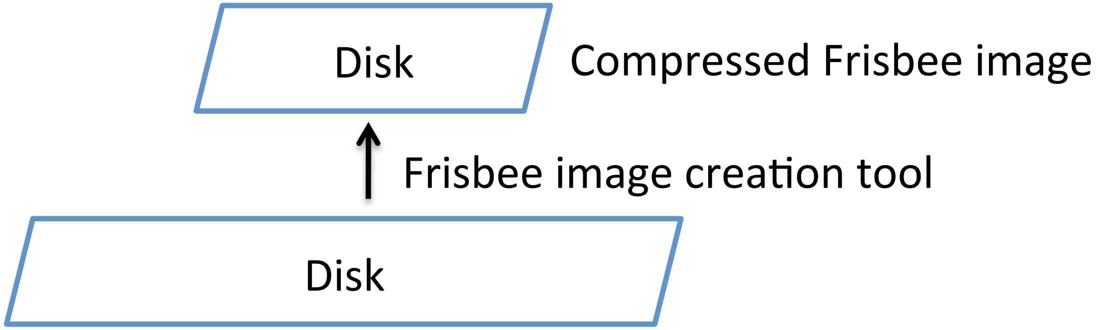
Requirements for the Integration

Requirements		Solutions
Use compression	→	Compress each chunk
Use filesystem to skip unallocated data	→	Aligned Fixed-size Chunking
Image installation at full disk bandwidth	→	Careful selection of chunk size
Independent installable Frisbee chunks	→	Pre-computation of Frisbee chunk headers

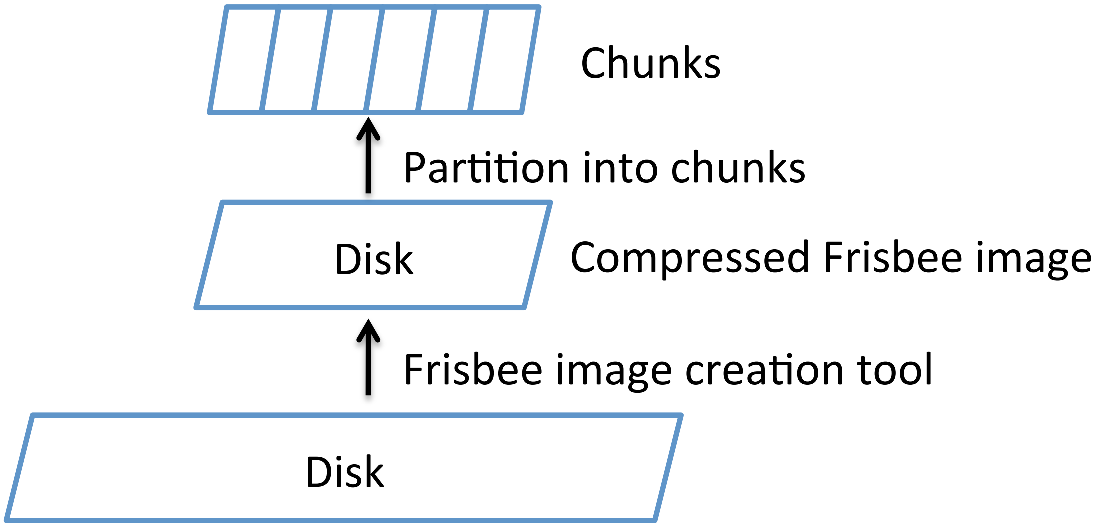
Requirements for the Integration

Requirements		Solutions
Use compression	→	Compress each chunk
Use filesystem to skip unallocated data	→	Aligned Fixed-size Chunking
Image installation at full disk bandwidth	→	Careful selection of chunk size
Independent installable Frisbee chunks	→	Pre-computation of Frisbee chunk headers

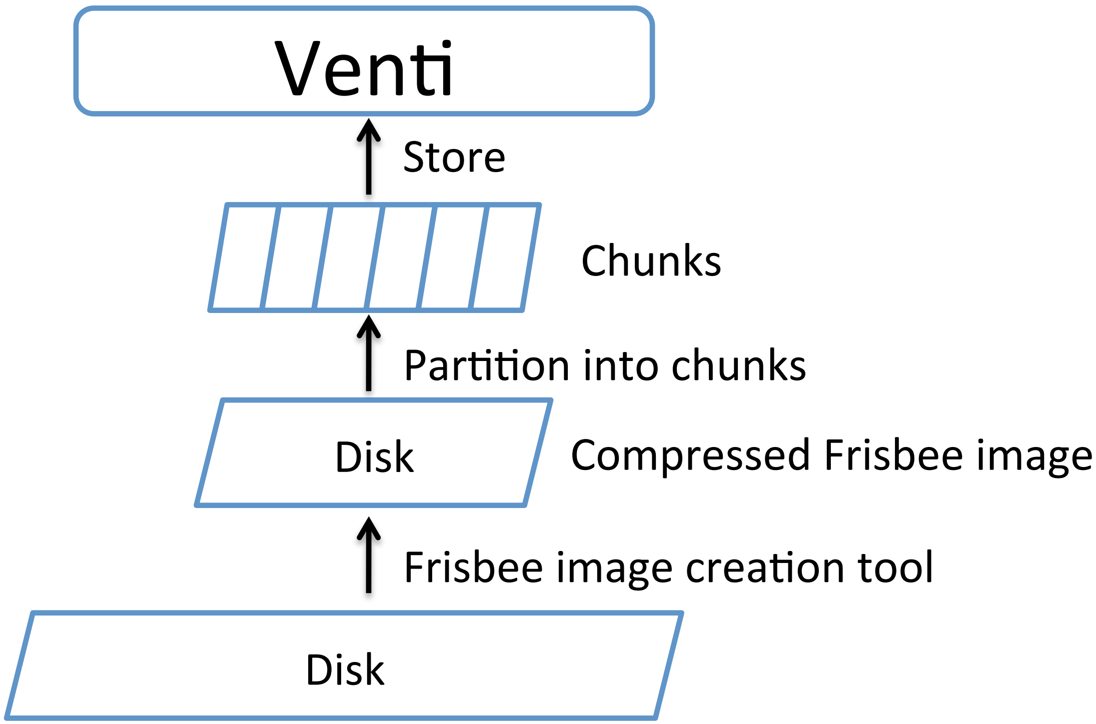
Use Compression – Store Frisbee Images



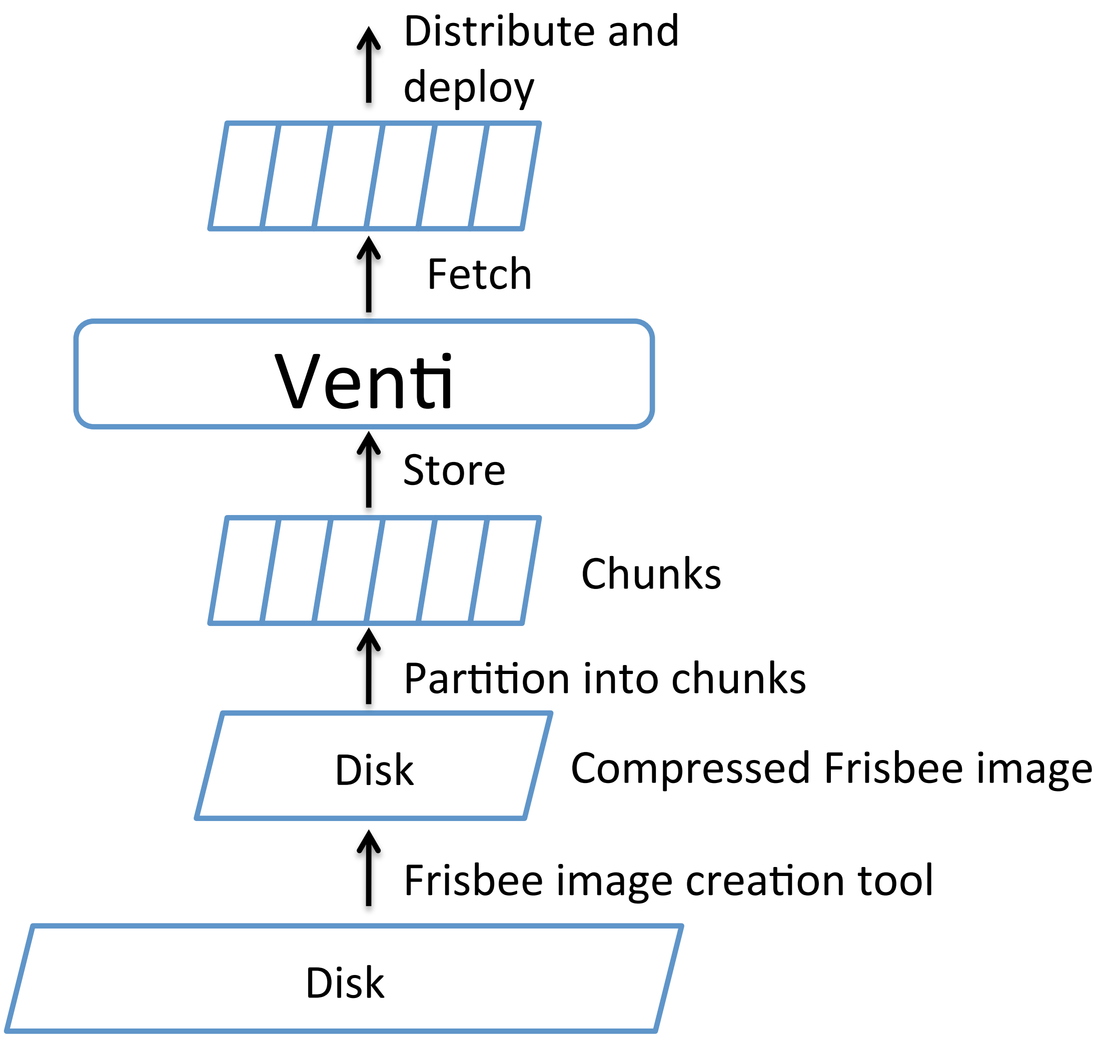
Use Compression – Store Frisbee Images



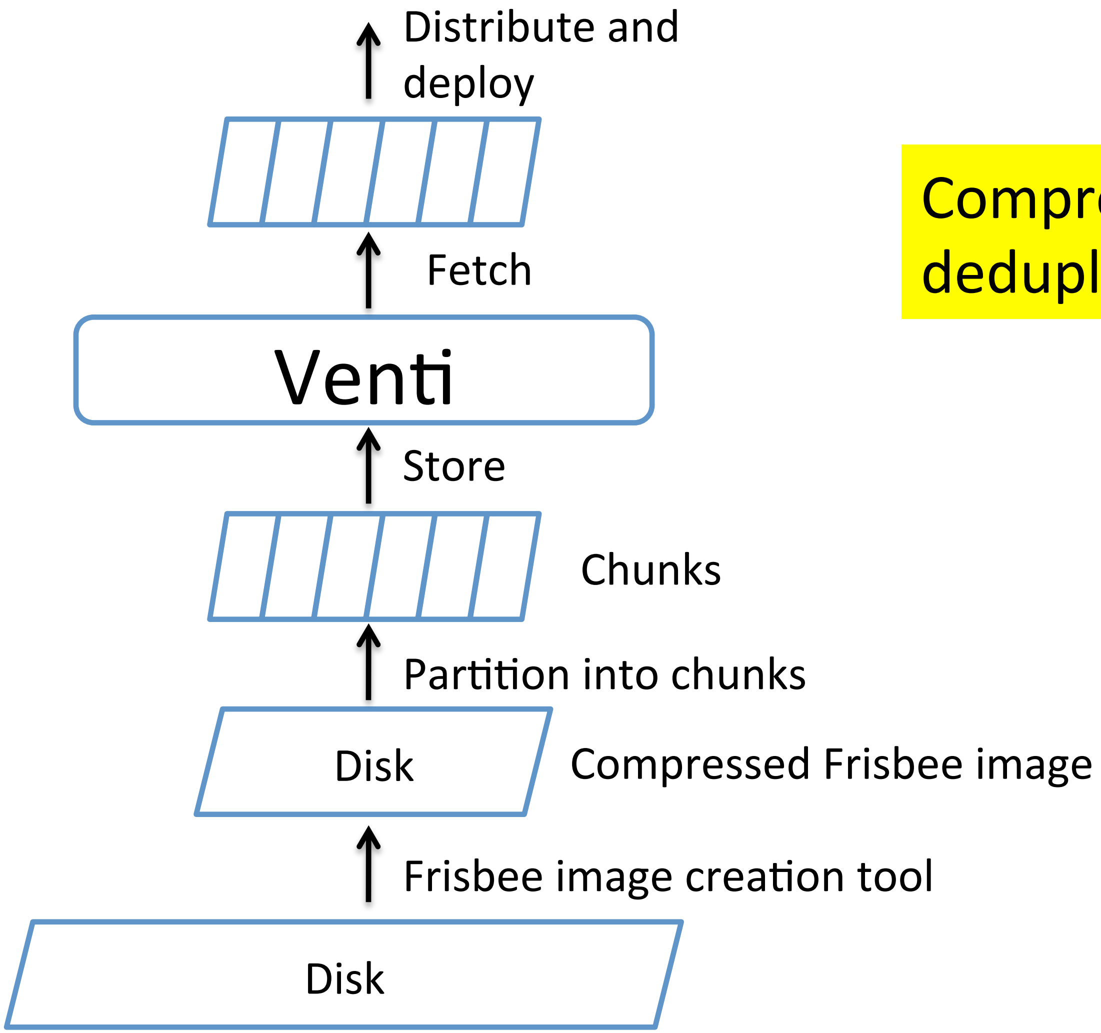
Use Compression – Store Frisbee Images



Use Compression – Store Frisbee Images

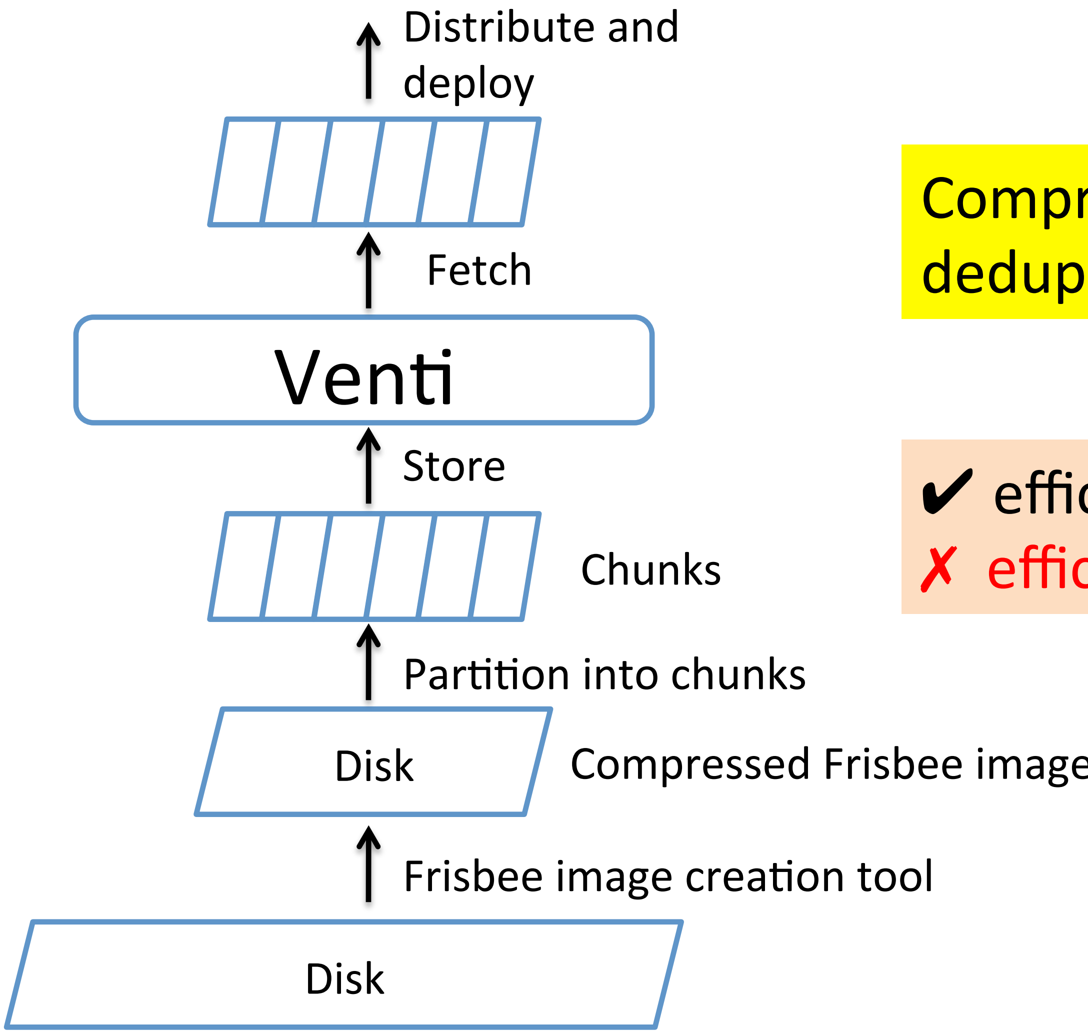


Use Compression – Store Frisbee Images



Compressed data does not deduplicate well

Use Compression – Store Frisbee Images

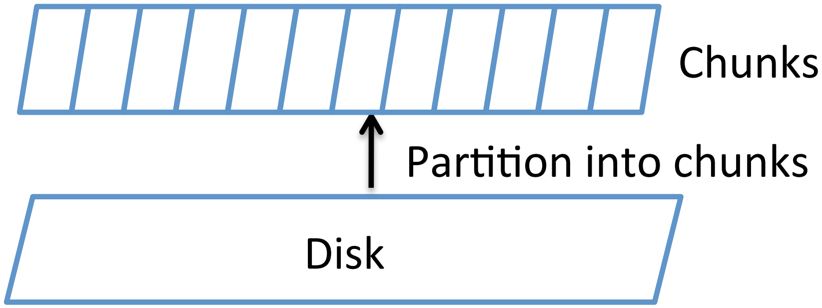


Compressed data does not deduplicate well

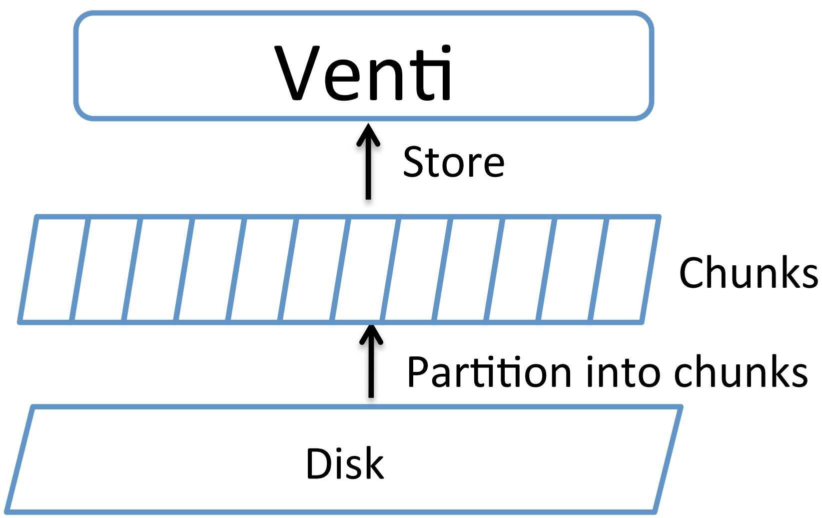
✔ efficient image deployment
✗ efficient image storage

Use Compression – Store Raw Chunks

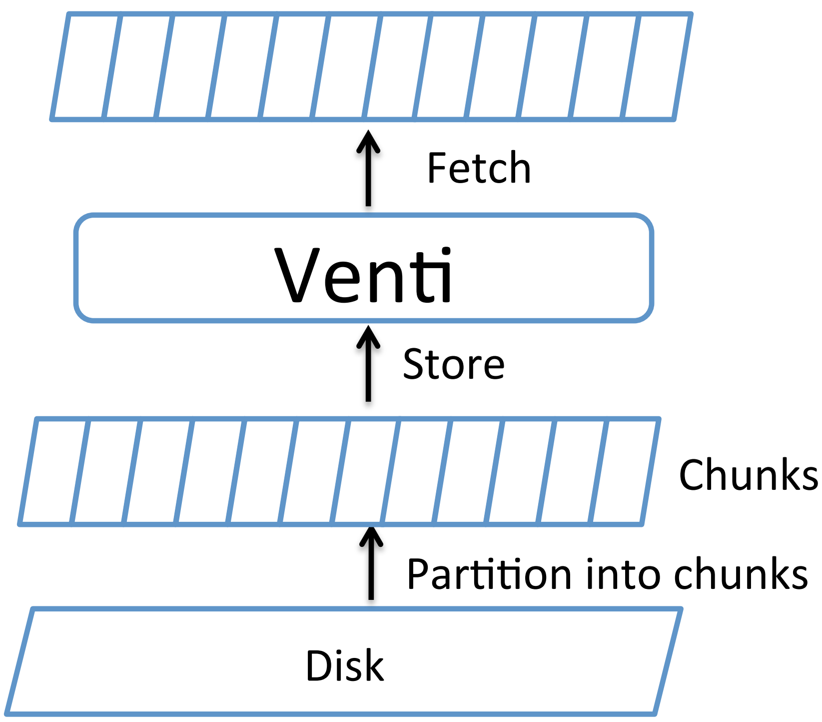
Use Compression – Store Raw Chunks



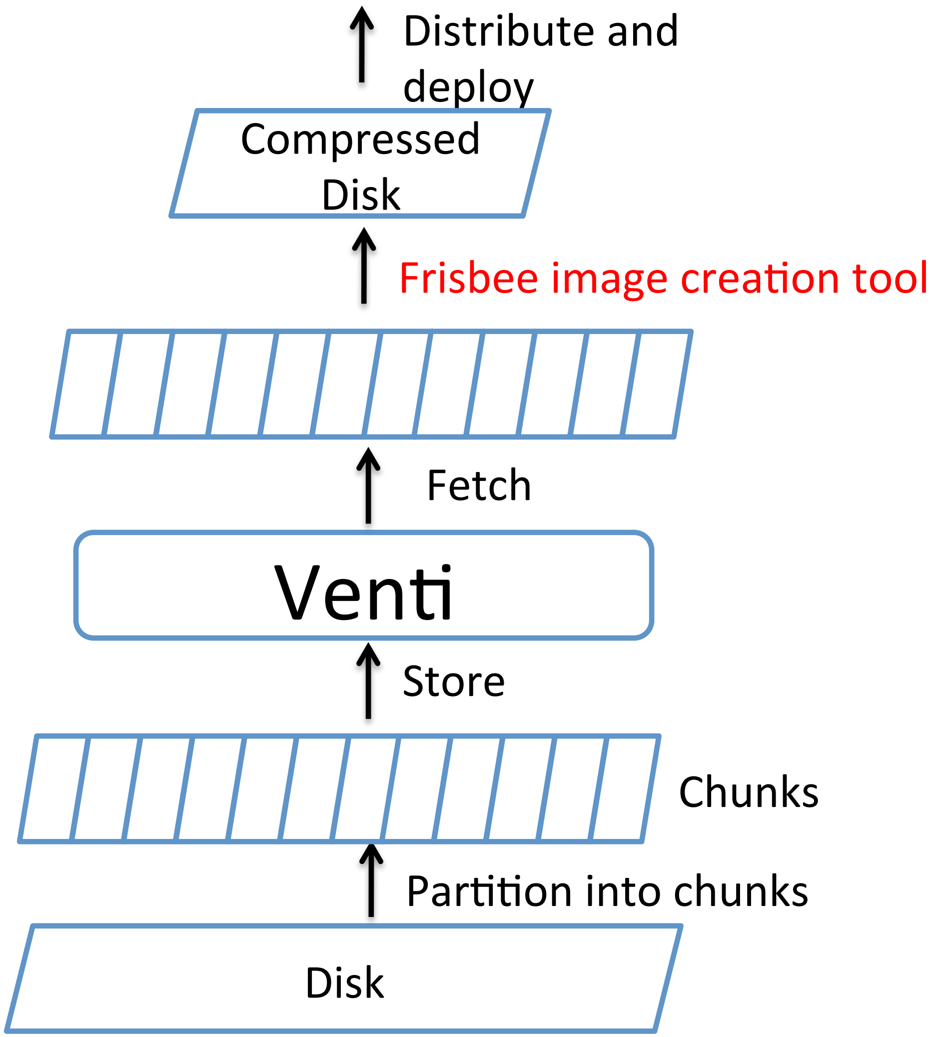
Use Compression – Store Raw Chunks



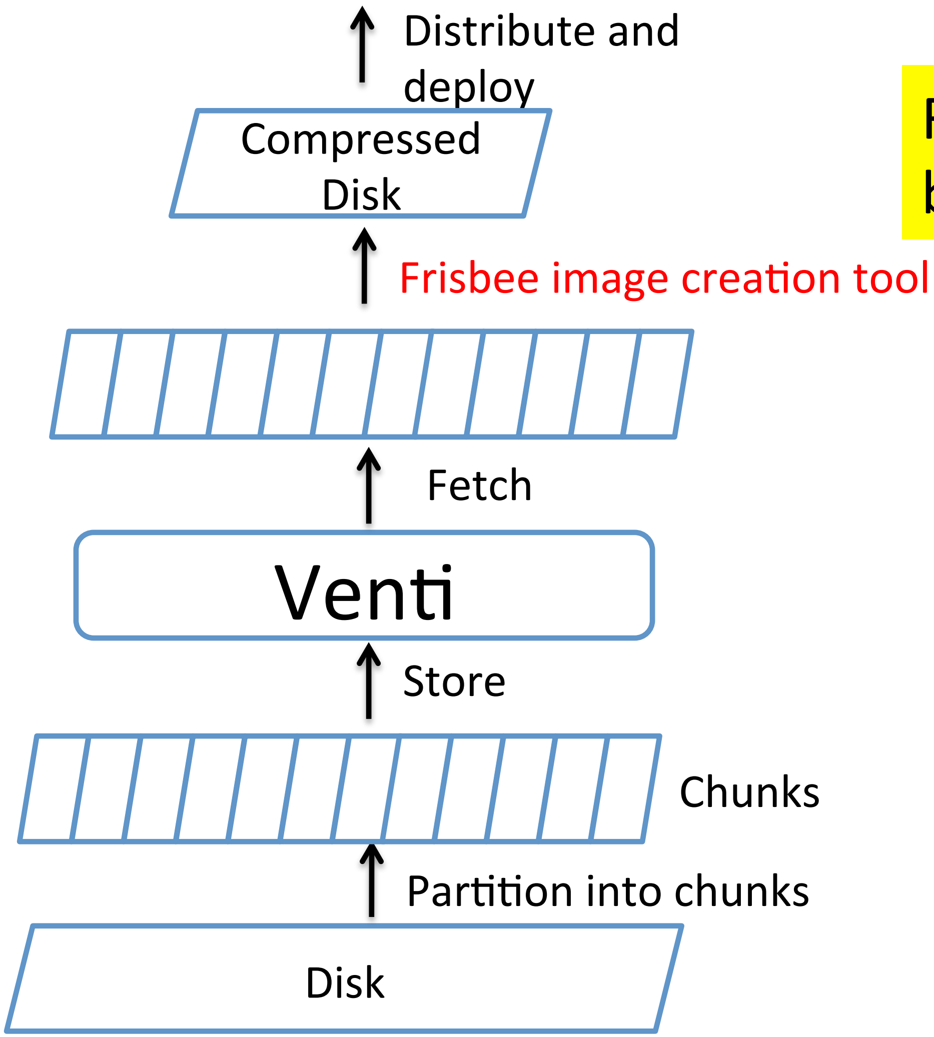
Use Compression – Store Raw Chunks



Use Compression – Store Raw Chunks

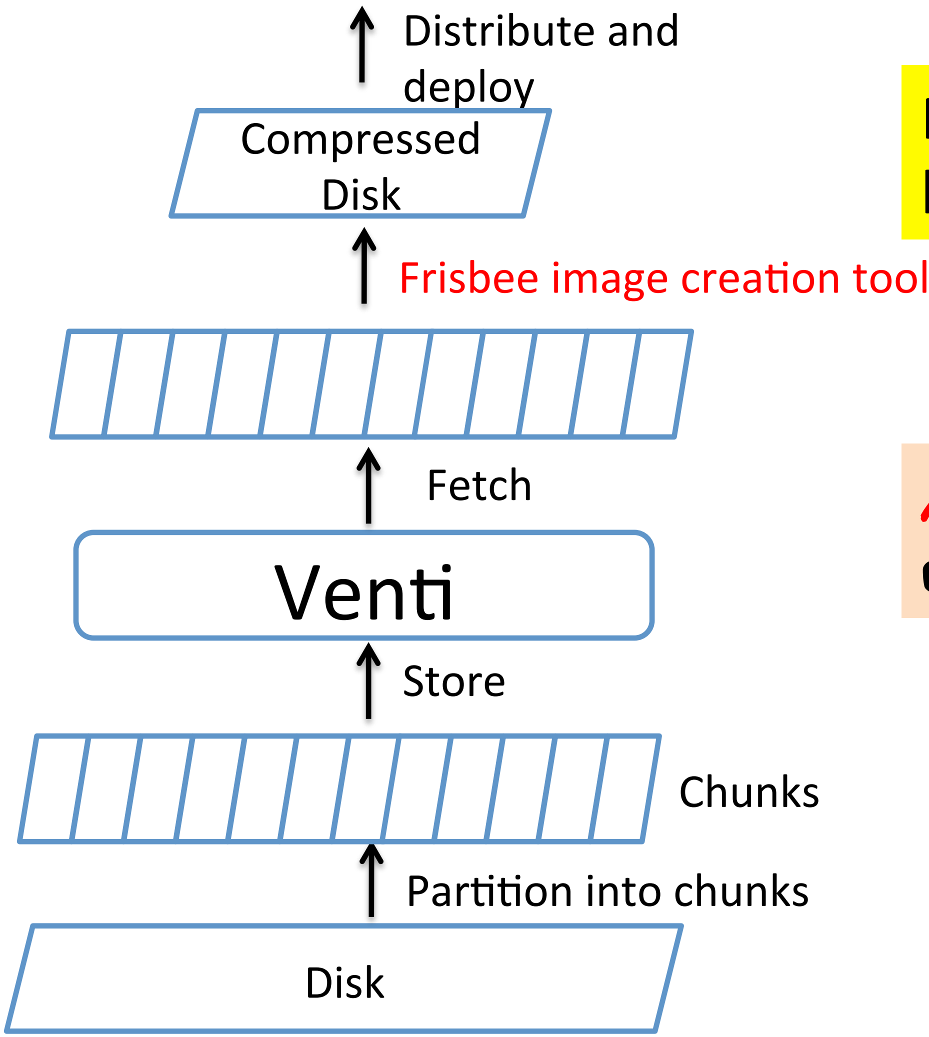


Use Compression – Store Raw Chunks



Frisbee compression will become the bottleneck

Use Compression – Store Raw Chunks

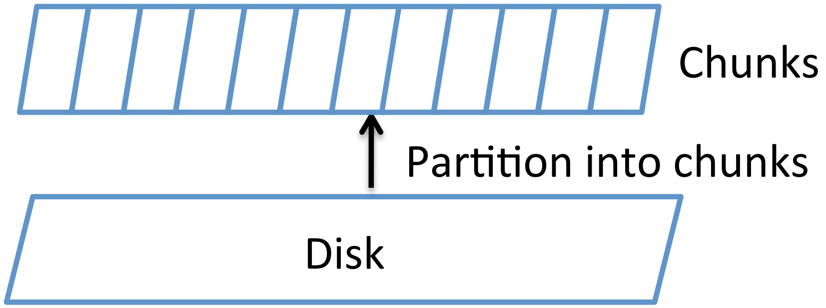


Frisbee compression will become the bottleneck

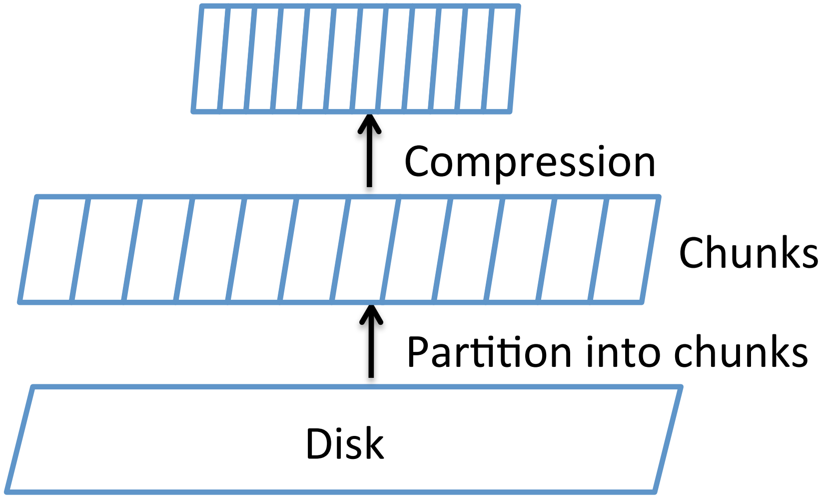
✗ efficient image deployment
✔ efficient image storage

Use Compression – Store Compressed Chunks

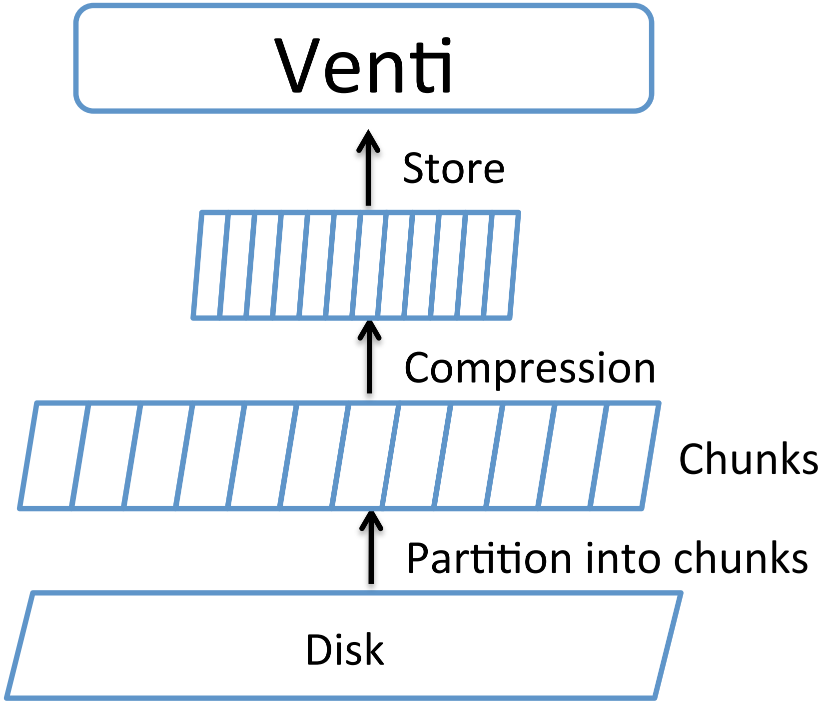
Use Compression – Store Compressed Chunks



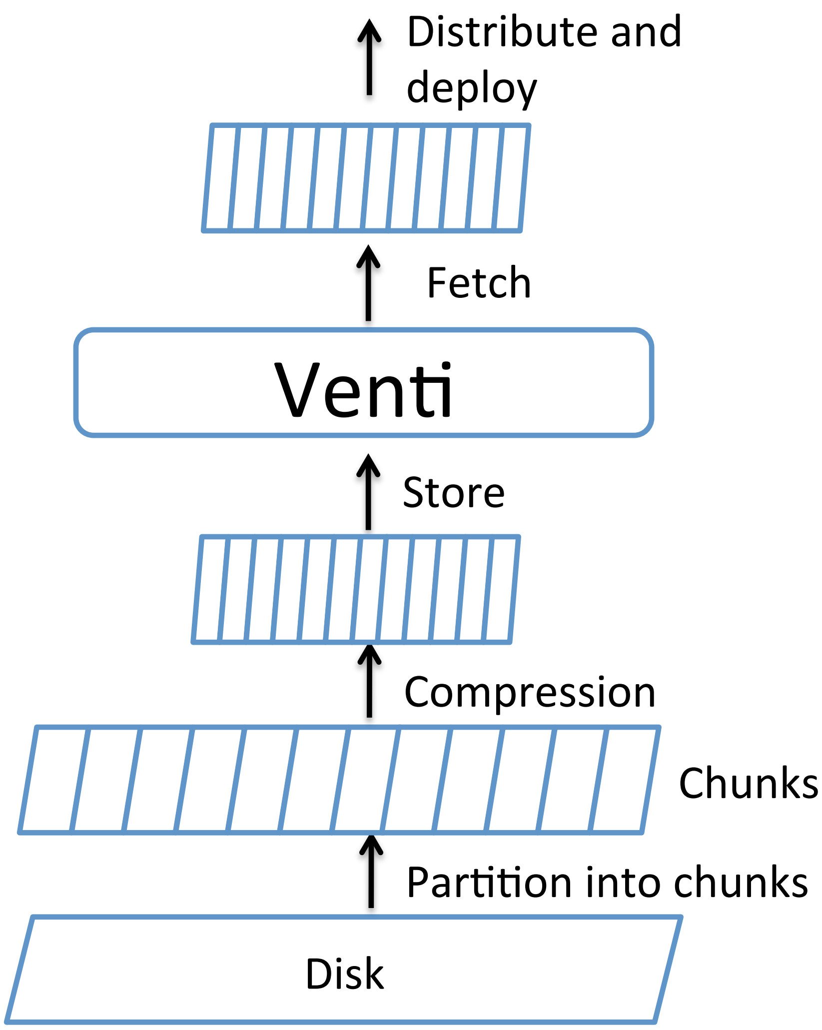
Use Compression – Store Compressed Chunks



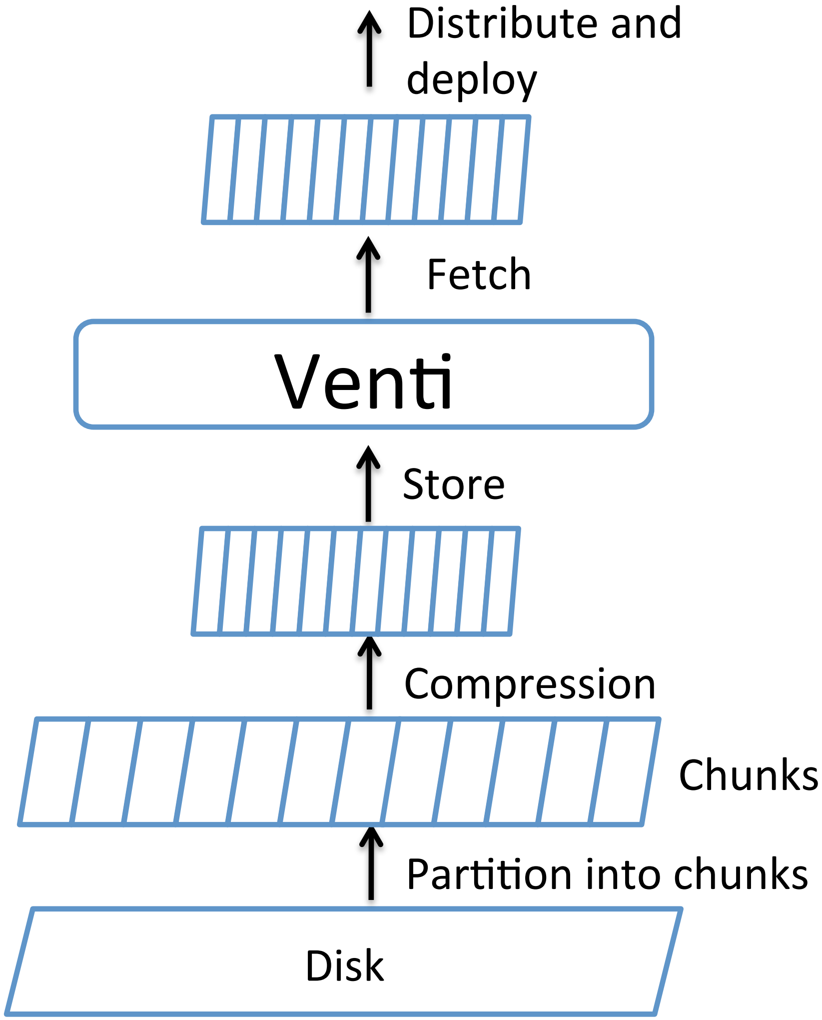
Use Compression – Store Compressed Chunks



Use Compression – Store Compressed Chunks

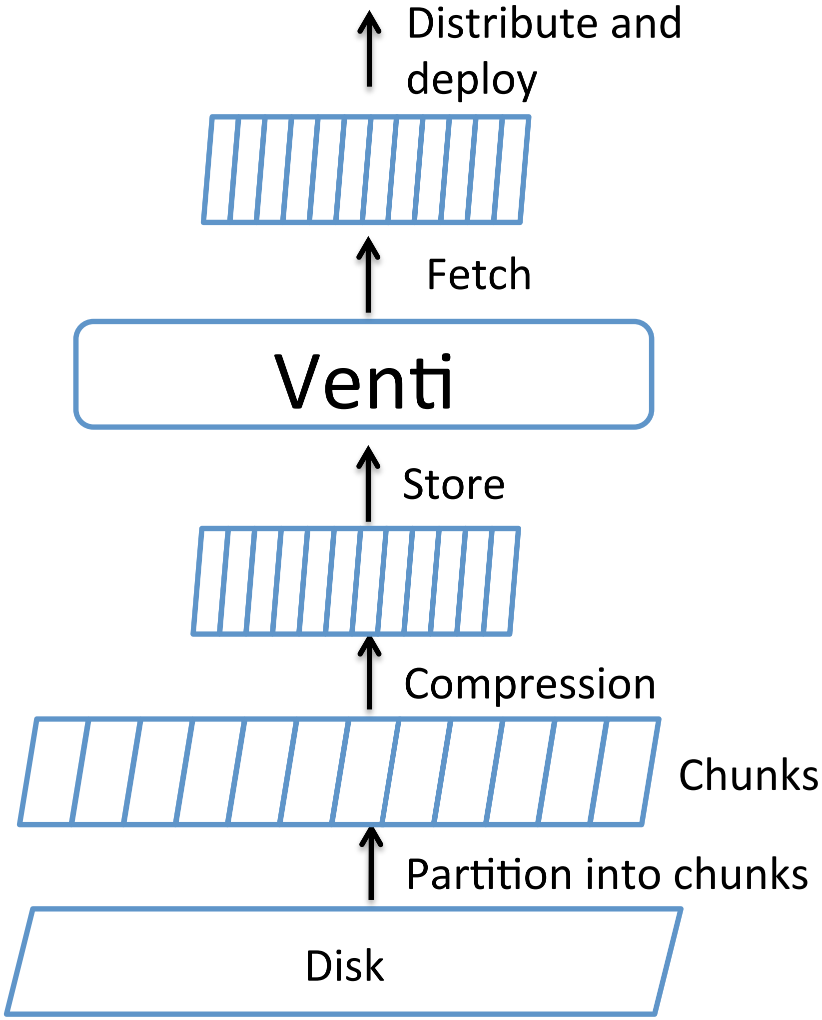


Use Compression – Store Compressed Chunks



- No compression in image deployment
- Compression of two identical chunks => same compressed chunk

Use Compression – Store Compressed Chunks



- No compression in image deployment
- Compression of two identical chunks => same compressed chunk

✔ efficient image deployment
✔ efficient image storage

Efficient Image Storage

- Dataset: 430 Linux images
  - Filesystem size: 21 TB (allocated 2 TB)

Format	Size (GB)
Frisbee Images	233
Venti 32KB	75

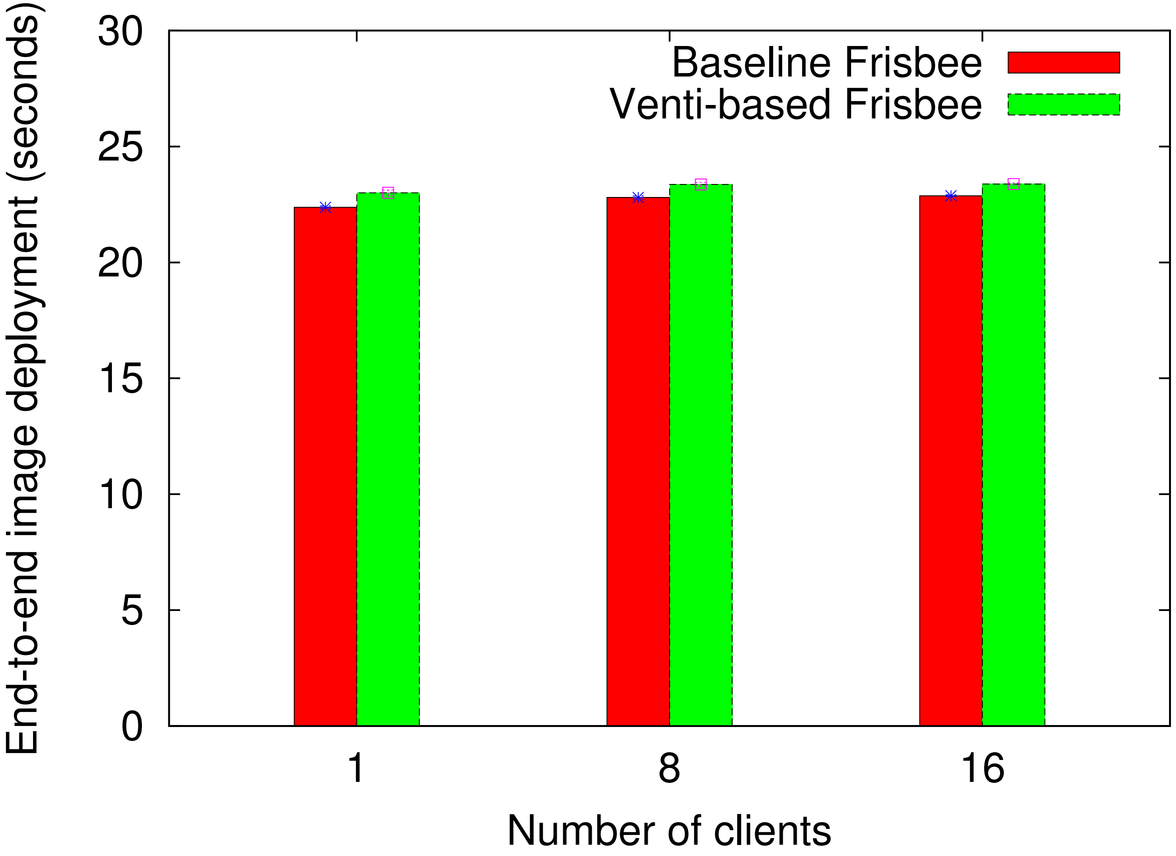
Efficient Image Storage

- Dataset: 430 Linux images
  - Filesystem size: 21 TB (allocated 2 TB)

Format	Size (GB)
Frisbee Images	233
Venti 32KB	75

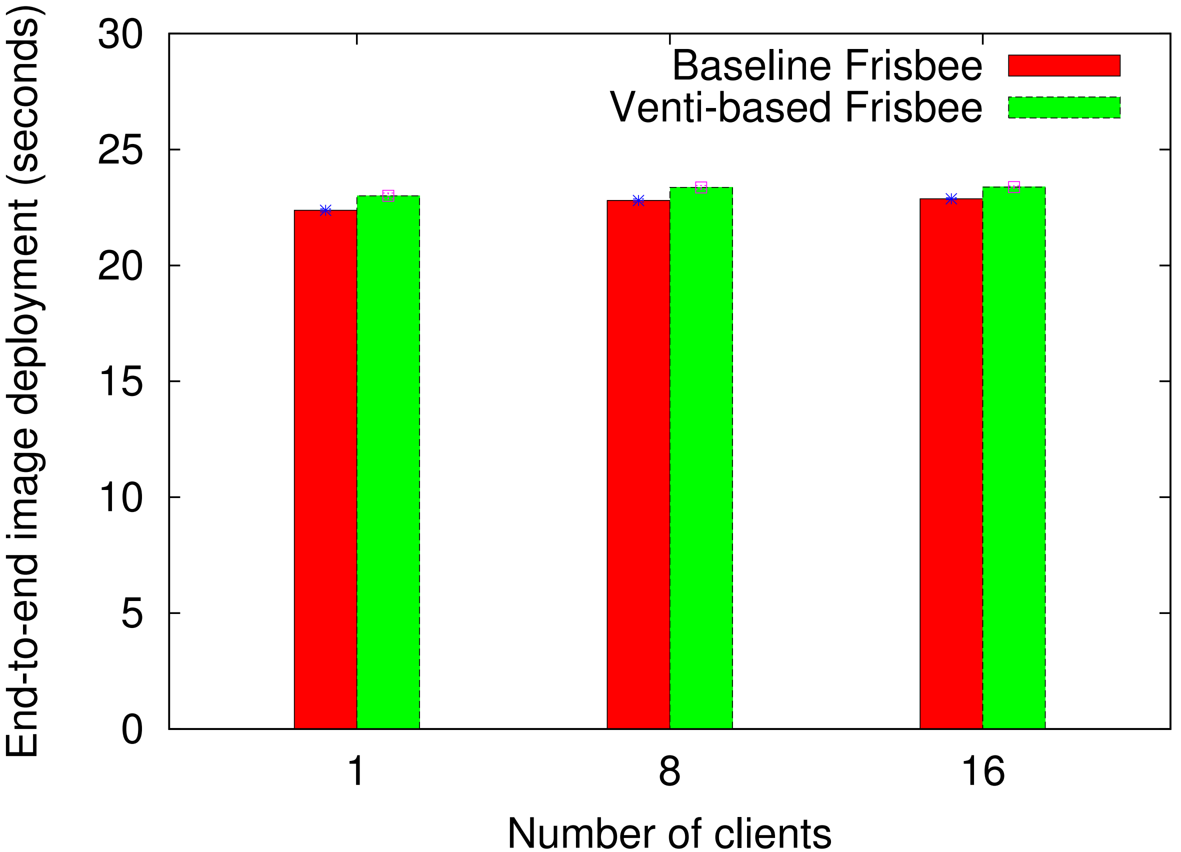
3× reduction

Efficient Image Deployment



End-to-End Image Deployment Performance

Efficient Image Deployment



Similar performance
In image deployment

End-to-End Image Deployment Performance

More in the Paper

- Aligned Fixed-size Chunking (AFC)
- Image retrieve time based on chunk sizes
  - Smaller chunk size: better deduplication
  - Larger chunk size: better image deployment performance
- Image deployment performance with staging nodes

Summary

- Compress each chunk
- Aligned Fixed-size Chunking
- Careful selection of chunk size
- Pre-computation of Frisbee chunk headers

→

✔ efficient image deployment
✔ efficient image storage

Using Deduplicating Storage for Efficient Disk Image Deployment
Xing Lin, Mike Hibler, Eric Eide, and Robert Ricci
TridentCom '15: 10th international conference on testbeds and research infrastructures for the development of networks & communities

Outline

- ✓ Introduction
  - ✓ Migratory Compression
  - ✓ Improve deduplication by separating metadata from data
  - ✓ Using deduplication for efficient disk image deployment
  - ✓ ***Performance predictability and efficiency for Cloud Storage Systems***
- ✓ Conclusion

Introduction

- Cloud Computing is becoming popular
  - Amazon Web Service (AWS), Microsoft Azure ...
- Virtualization enables efficient resource sharing
  - Multiplexing and consolidation; economics of scale

Introduction

- Cloud Computing is becoming popular
  - Amazon Web Service (AWS), Microsoft Azure ...
- Virtualization enables efficient resource sharing
  - Multiplexing and consolidation; economics of scale
- Sharing introduces interference
  - Random and sequential workloads [Gulati VPACT '09]
  - Read and write workloads [Ahmad WWC '03]

Introduction

- Cloud Computing is becoming popular
  - Amazon Web Service (AWS), Microsoft Azure ...
- Virtualization enables efficient resource sharing
  - Multiplexing and consolidation; economics of scale
- Sharing introduces interference
  - Random and sequential workloads [Gulati VPACT '09]
  - Read and write workloads [Ahmad WWC '03]
- Targeting environments: a ***large*** number of ***mixed*** workloads

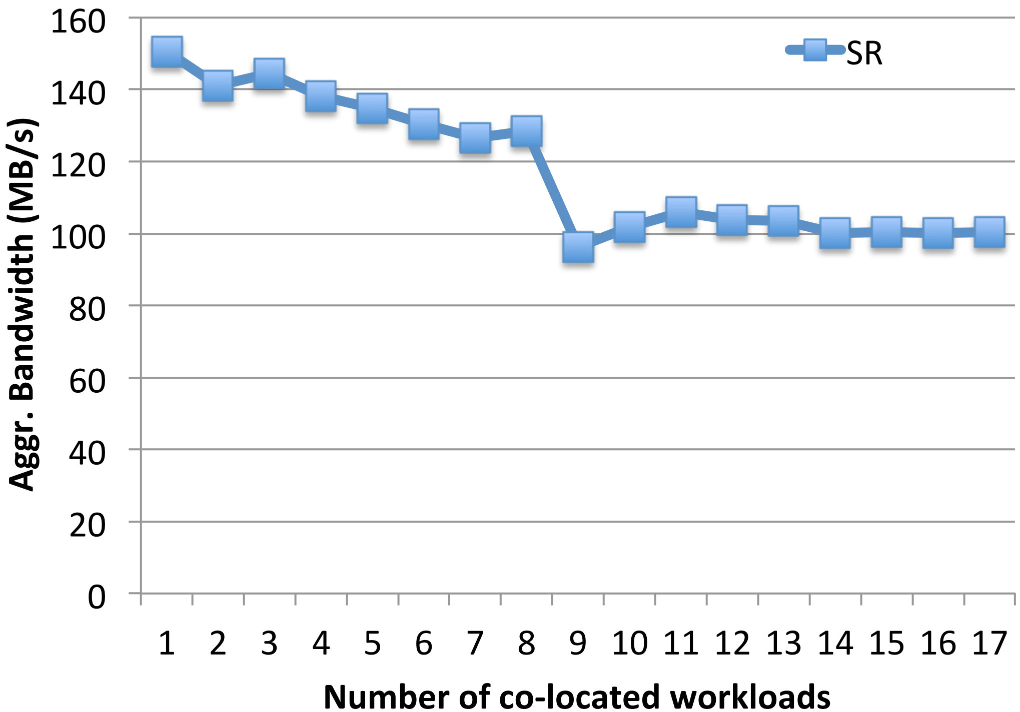
Random Workloads' Impact on Disk Bandwidth

Workloads:

- All sequential
- 1 sequential, adding more random ones

High disk bandwidth = high disk utilization

Random Workloads' Impact on Disk Bandwidth

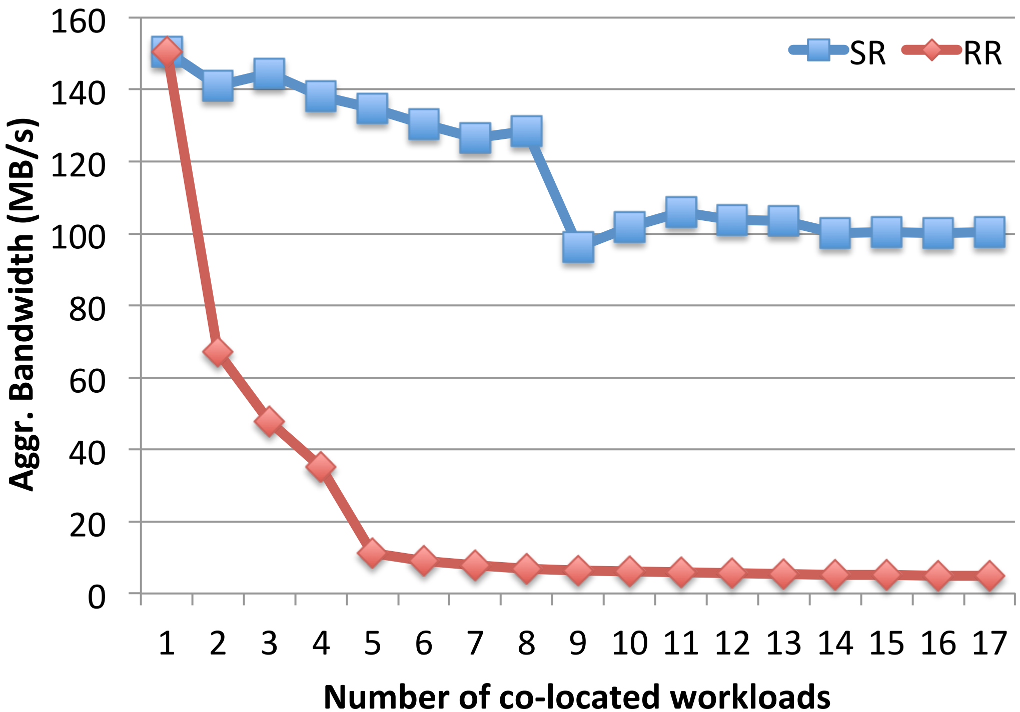


Workloads:

- All sequential
- 1 sequential, adding more random ones

High disk bandwidth = high disk utilization

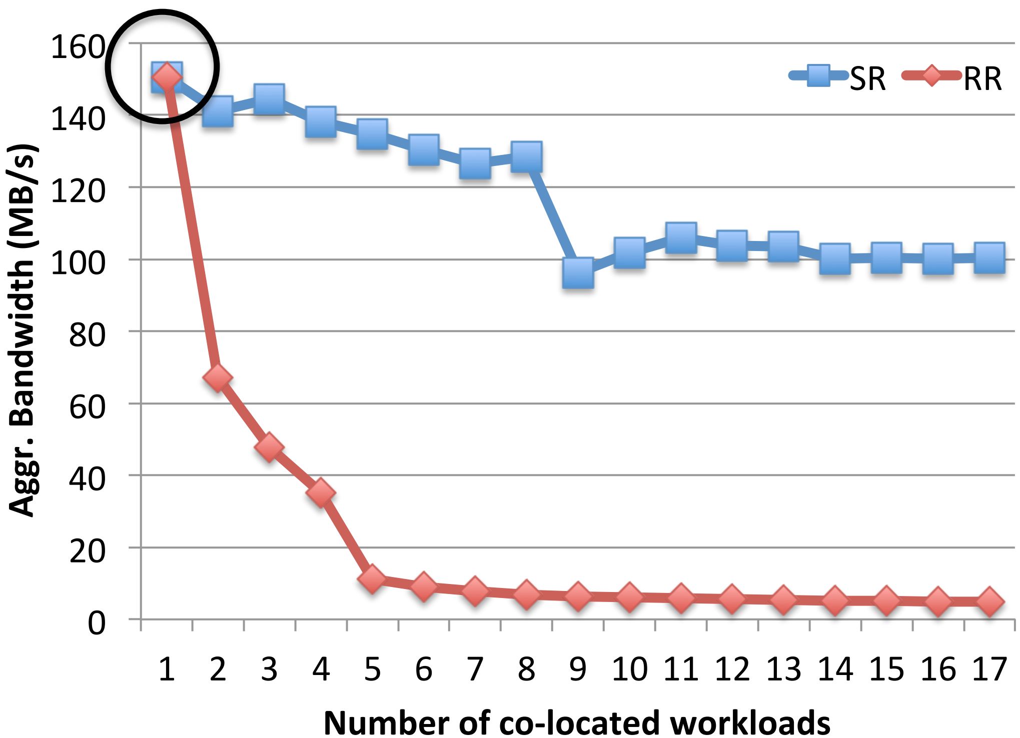
Random Workloads' Impact on Disk Bandwidth



Workloads:
- All sequential
- 1 sequential, adding more random ones

High disk bandwidth = high disk utilization

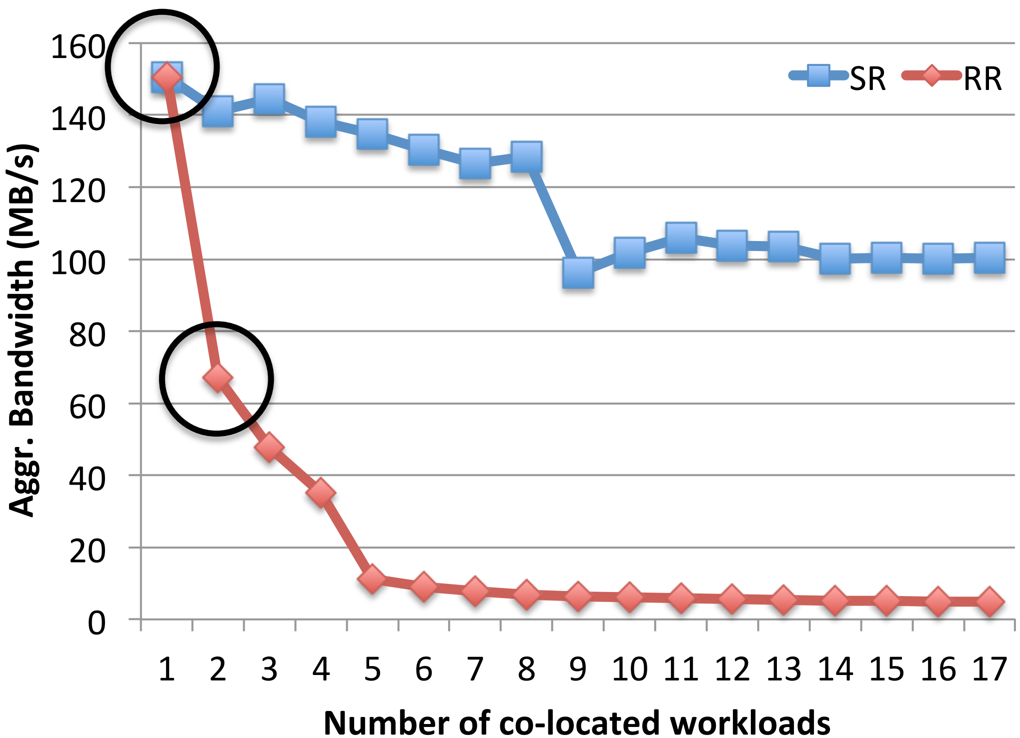
Random Workloads' Impact on Disk Bandwidth



Workloads:
- All sequential
- 1 sequential, adding more random ones

High disk bandwidth = high disk utilization

Random Workloads' Impact on Disk Bandwidth

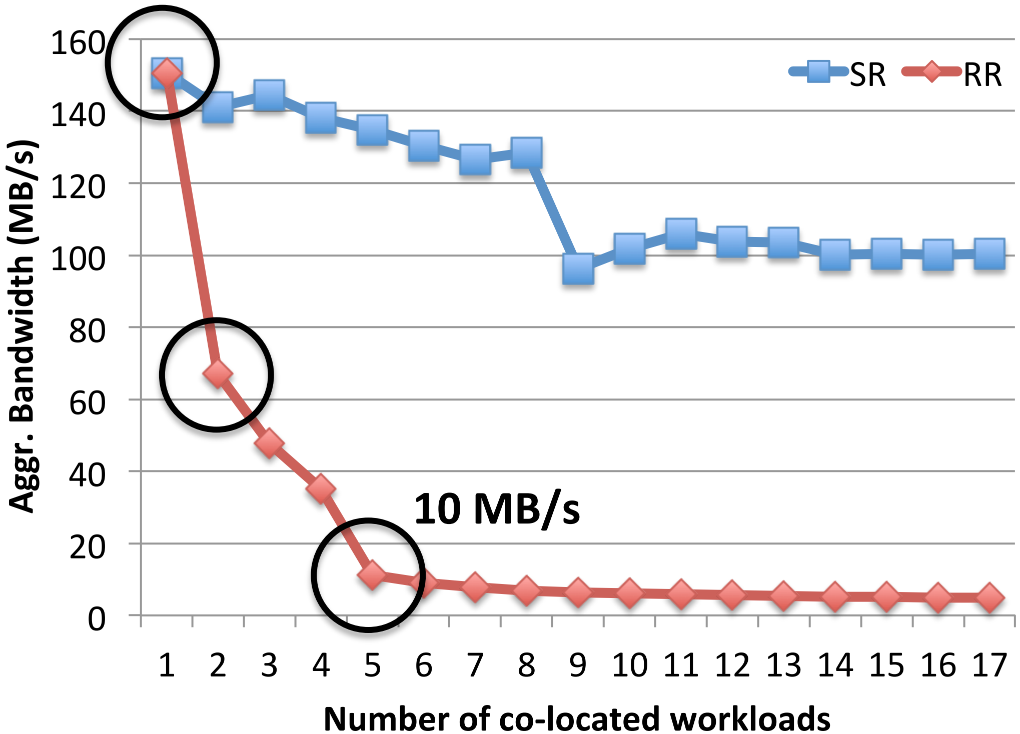


Workloads:

- All sequential
- 1 sequential, adding more random ones

High disk bandwidth = high disk utilization

Random Workloads' Impact on Disk Bandwidth

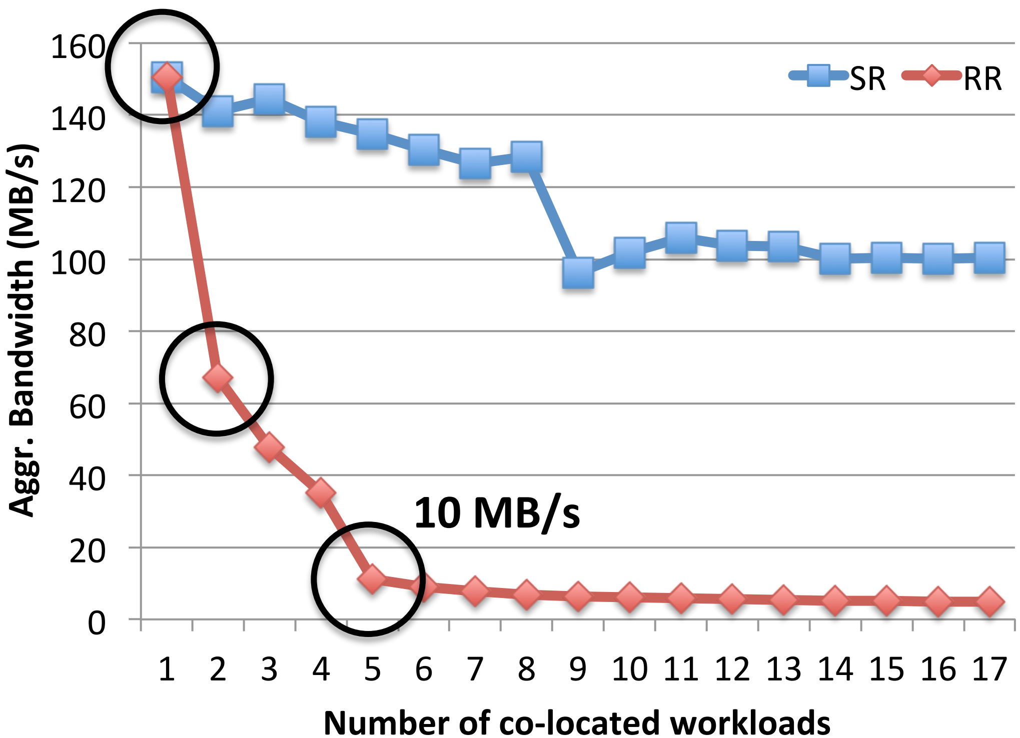


Workloads:

- All sequential
- 1 sequential, adding more random ones

High disk bandwidth = high disk utilization

Random Workloads' Impact on Disk Bandwidth



Workloads:
- All sequential
- 1 sequential, adding more random ones

Random workloads are harmful for disk bandwidth

High disk bandwidth = high disk utilization

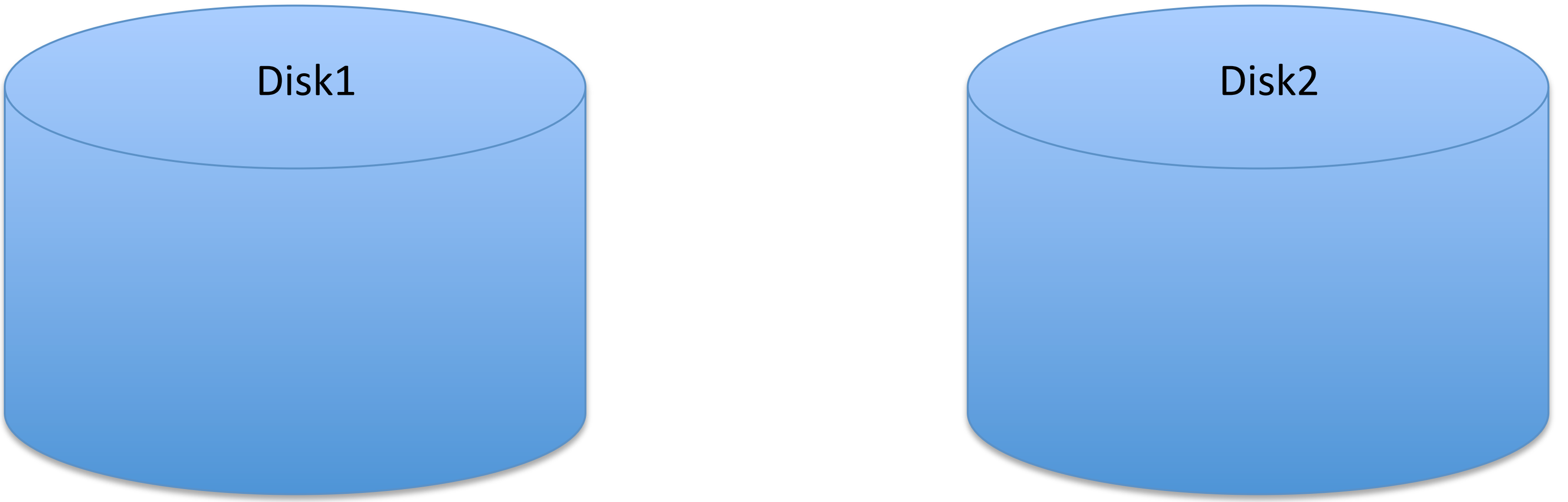
Differential I/O Scheduling (DIOS)

- **Opportunity**: replication is commonly used in cloud storage systems
  - Google Filesystem, Ceph, sheepdog, etc.
- **Idea**: dedicate each disk to serve one type of requests

Differential I/O Scheduling (DIOS)

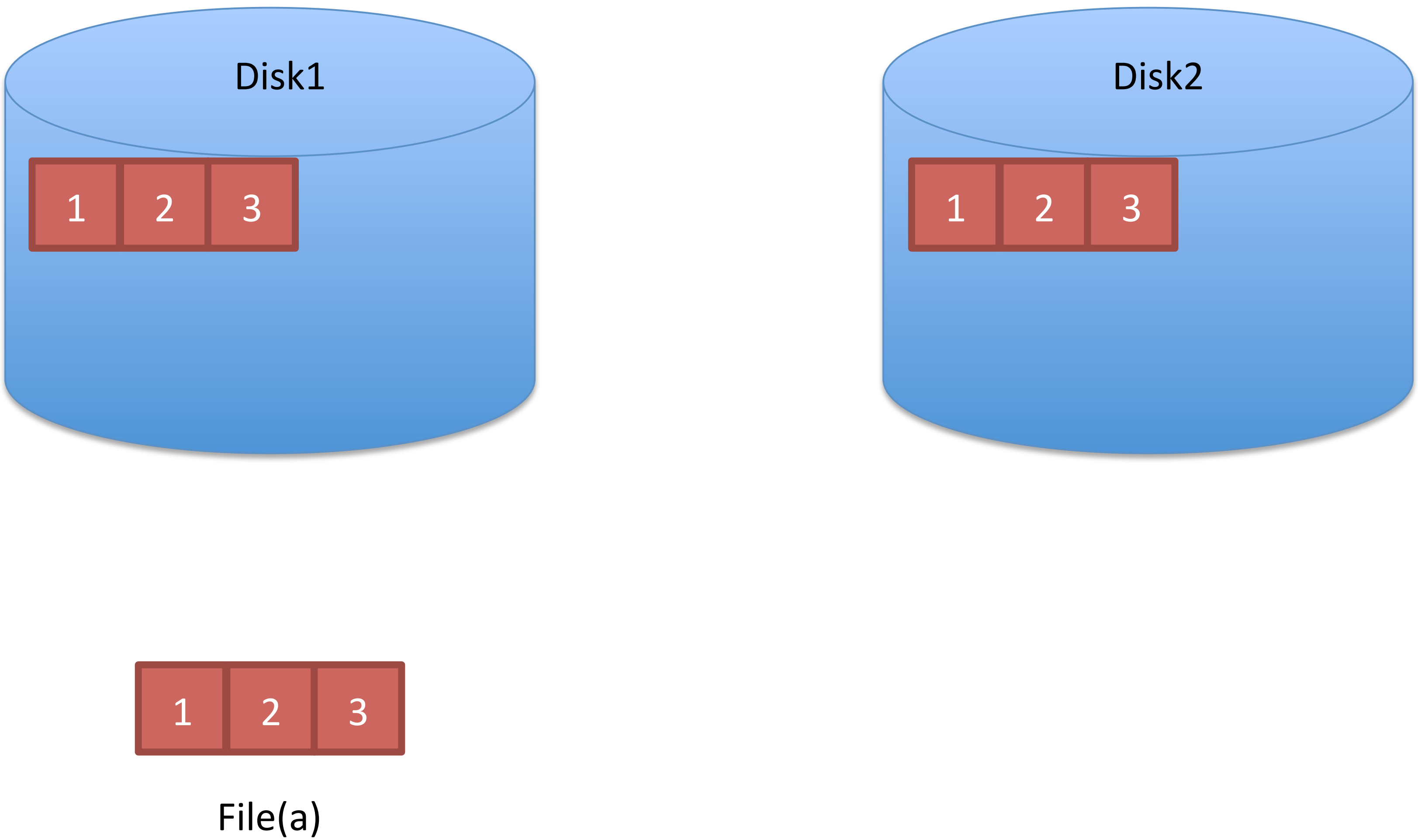
- **Opportunity**: replication is commonly used in cloud storage systems
  - Google Filesystem, Ceph, sheepdog, etc.
- **Idea**: dedicate each disk to serve one type of requests
- **System implications (tradeoff)**
  - High bandwidth for disks serving sequential requests
  - Lower performance for random workloads

Architecture



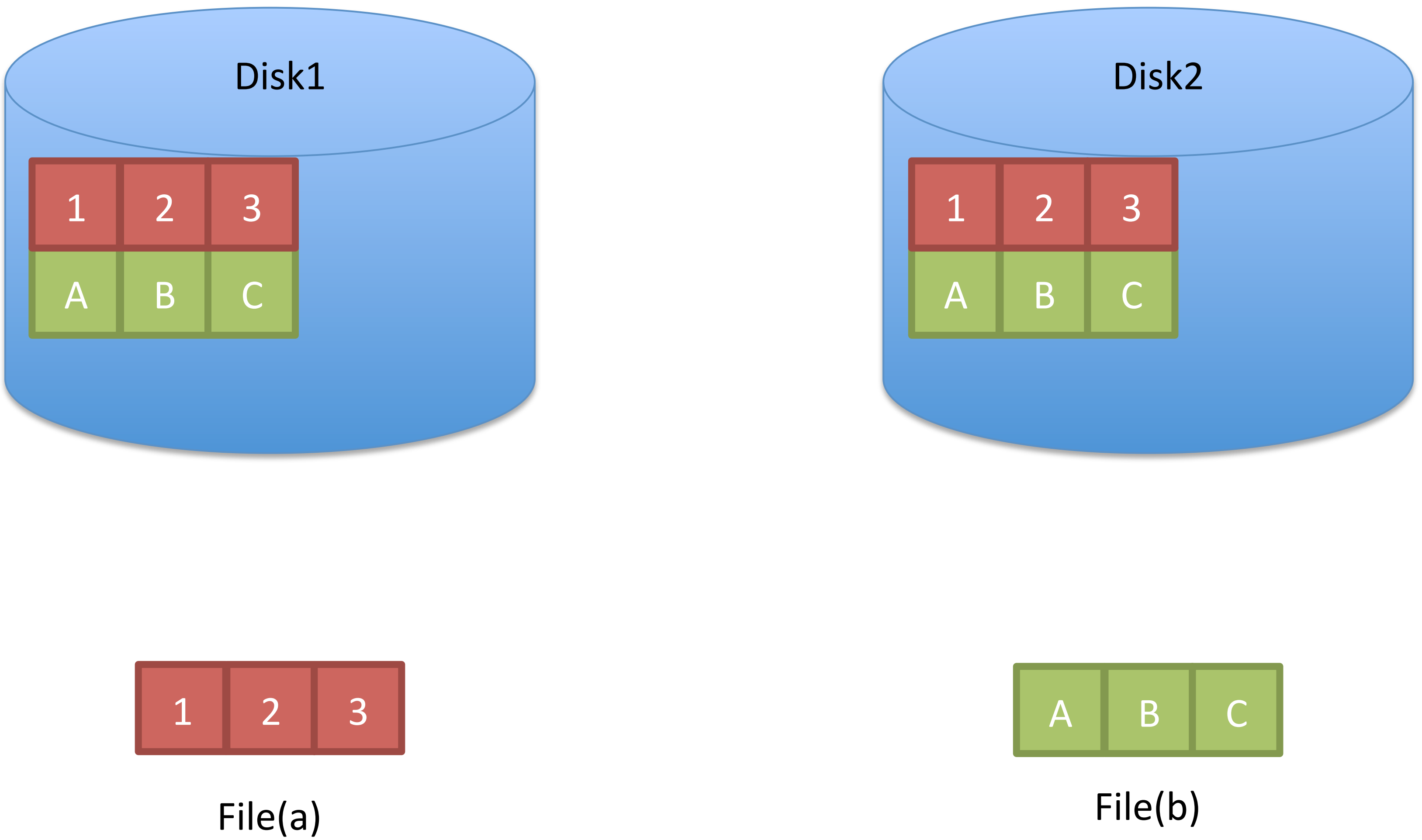
Architecture

Architecture

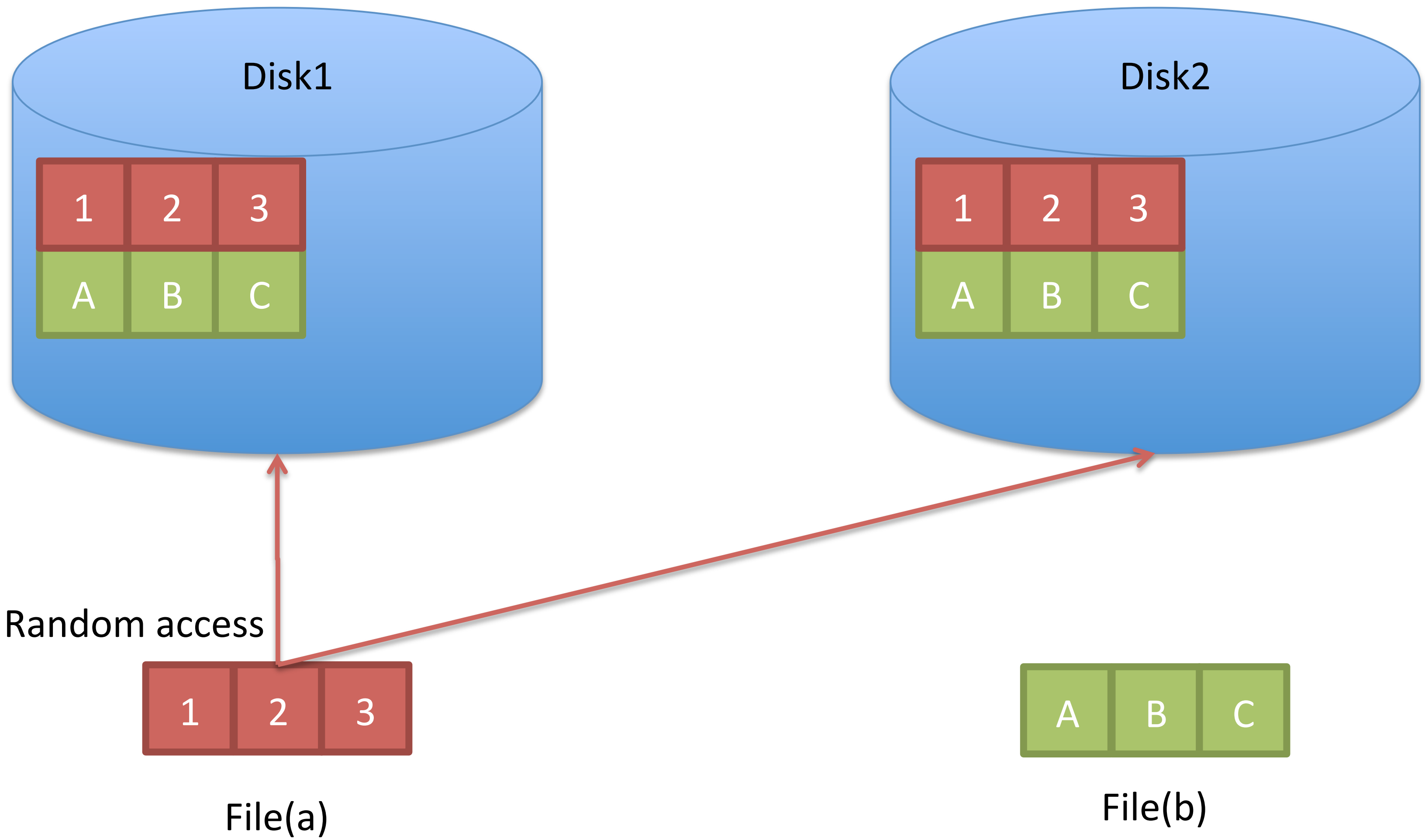


Architecture

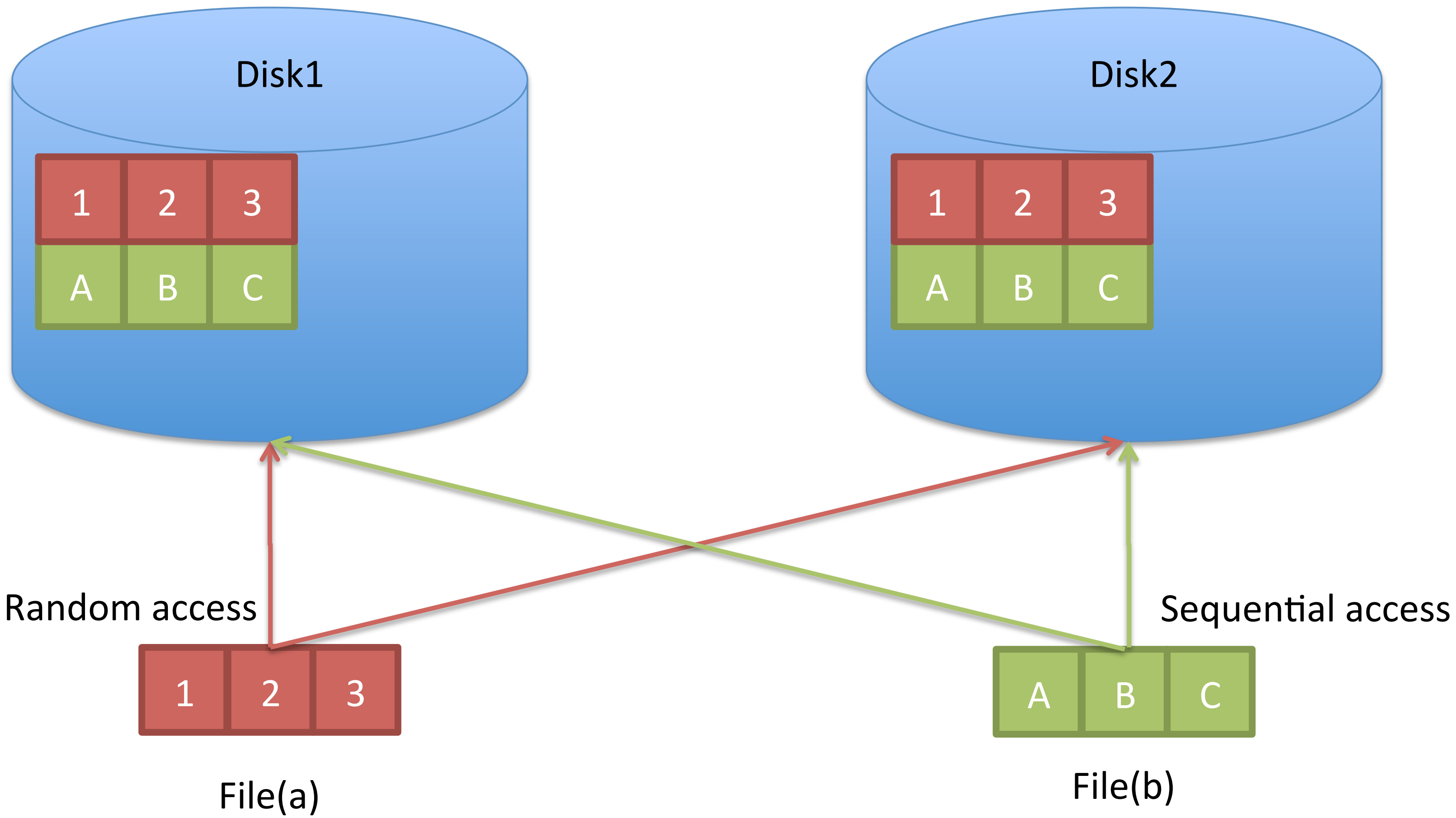
Architecture



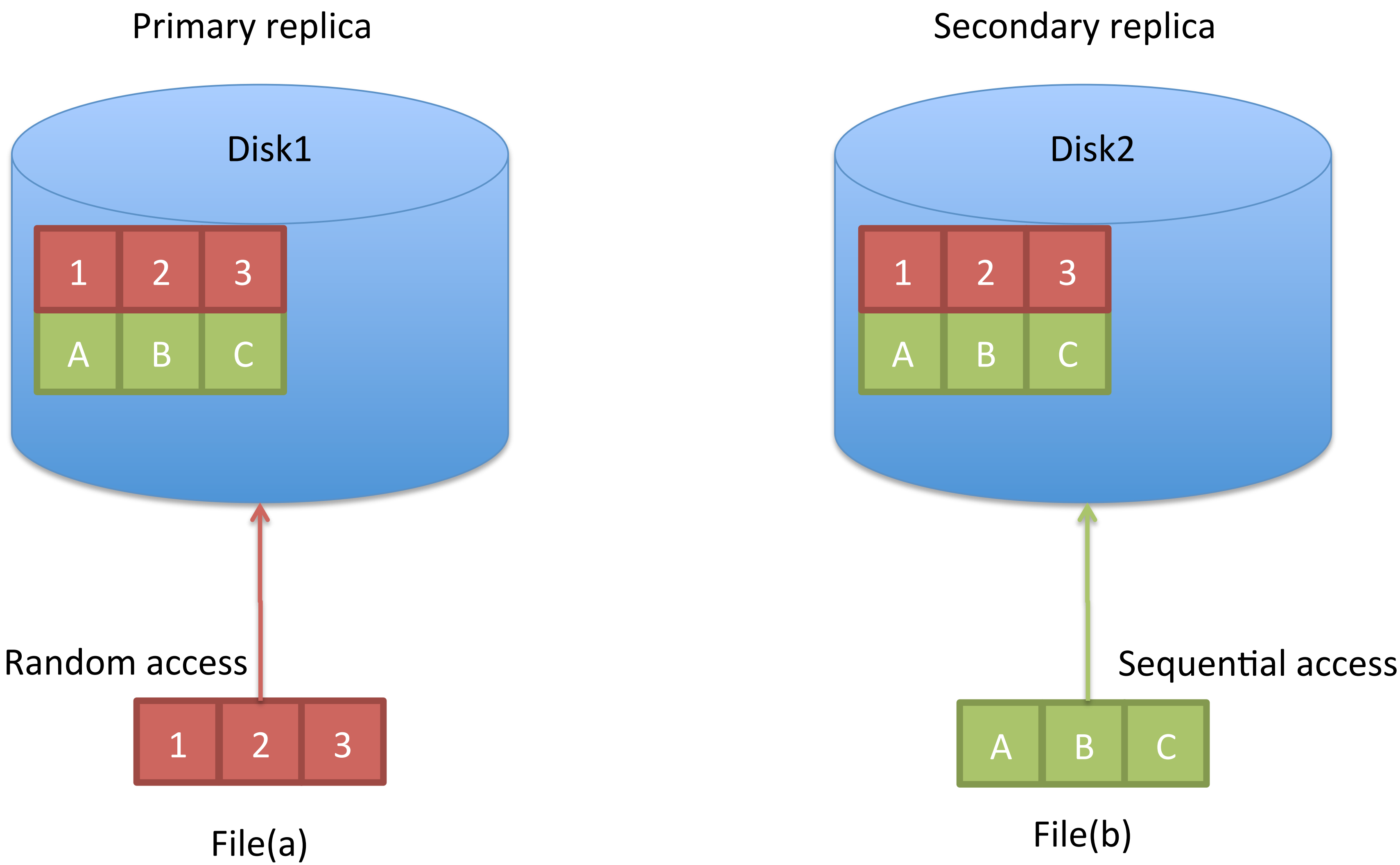
Architecture



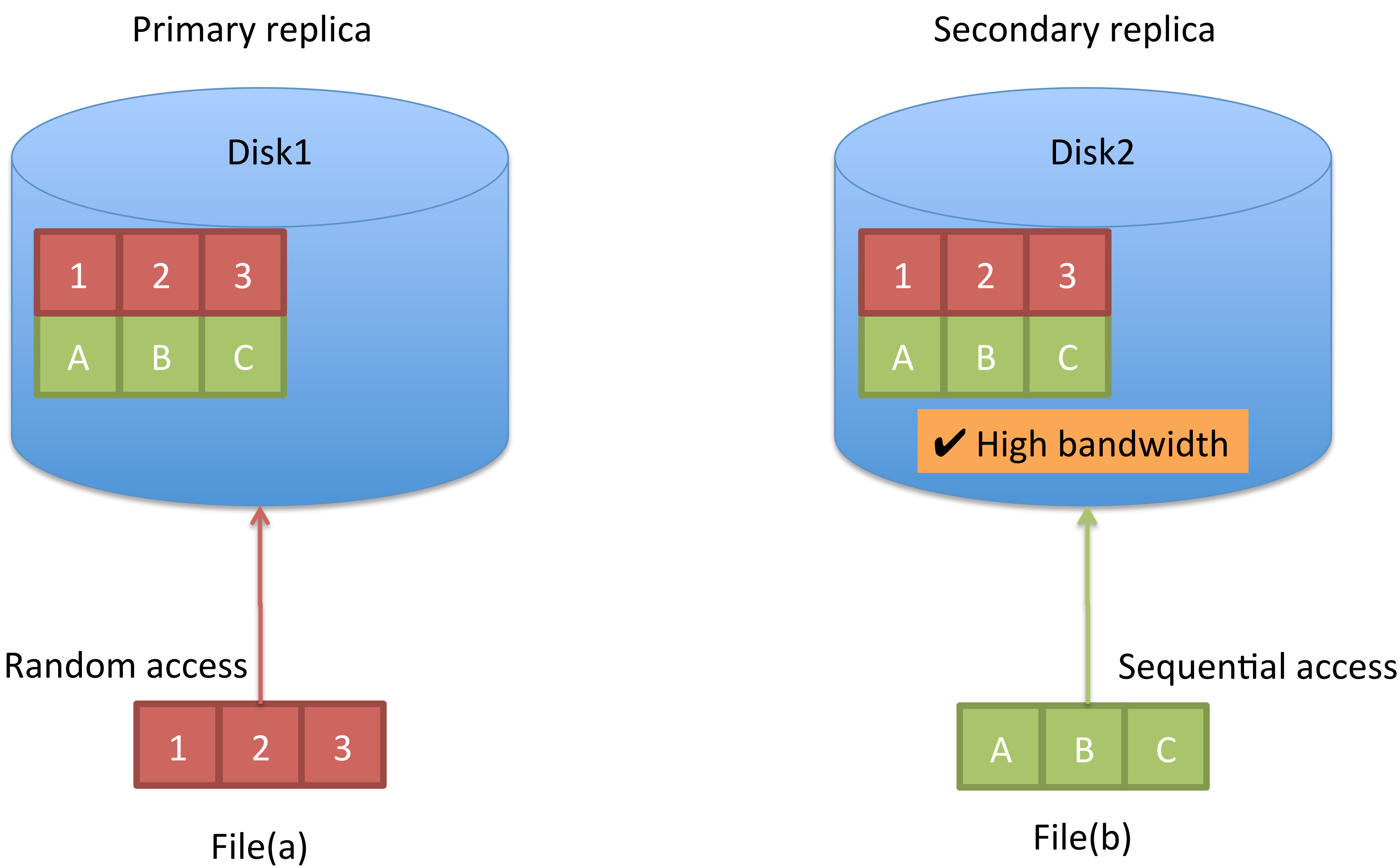
Architecture



Architecture



Architecture



Implementation

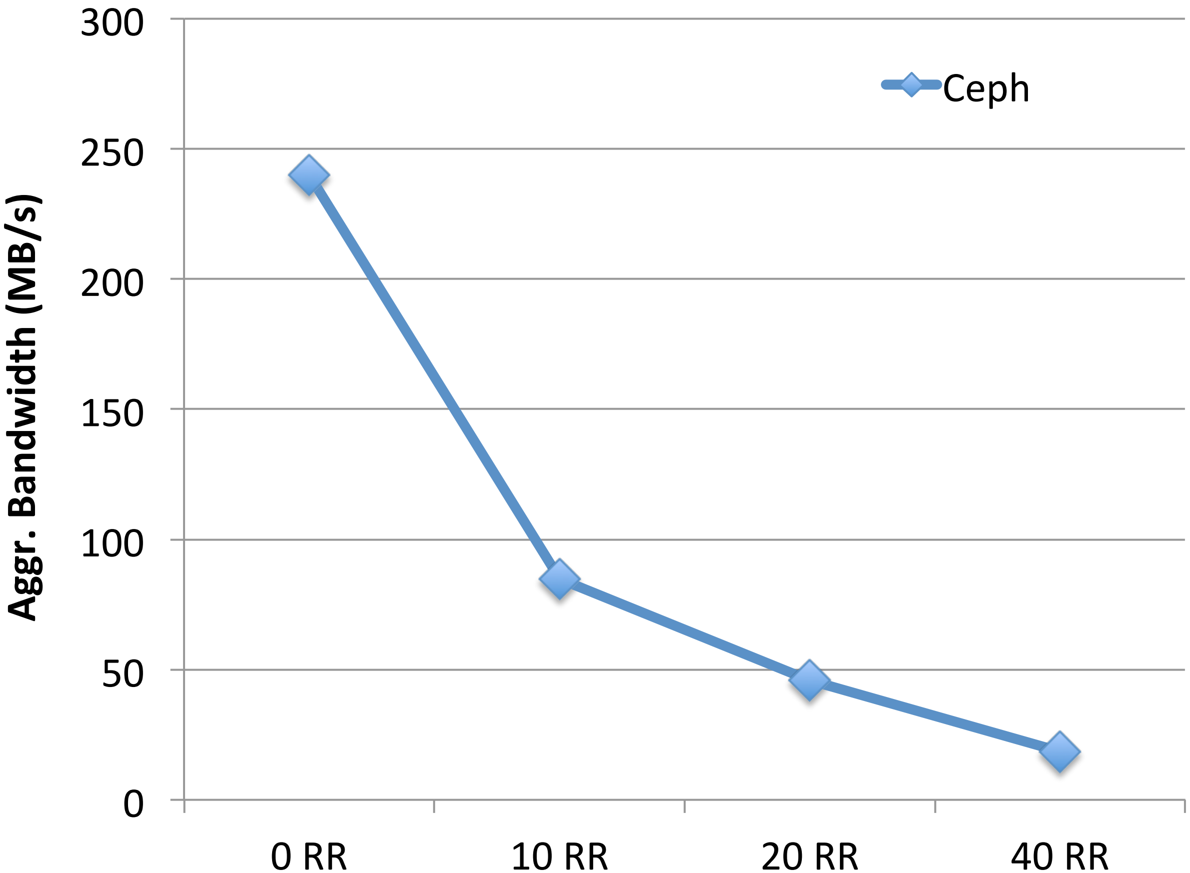
- Ceph: distributed storage system that implements replication
  - Use randomness in selecting replicas
- DIOS: modified Ceph with deterministic replica selections and request type-aware scheduling

Evaluation – Sequential Workloads

Workload:
20 sequential, mixing with different numbers of random workloads.

Total data disks: 6

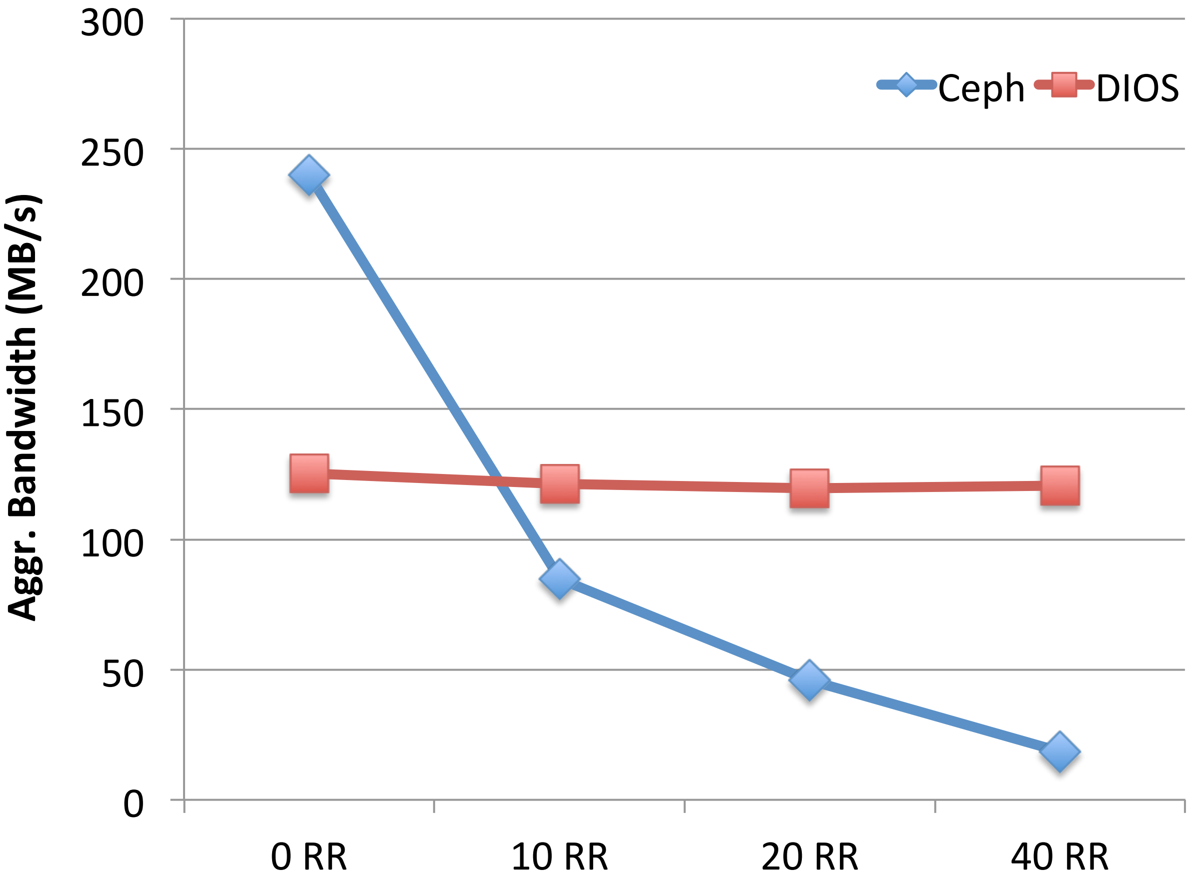
Evaluation – Sequential Workloads



Workload:
20 sequential, mixing with different numbers of random workloads.

Total data disks: 6

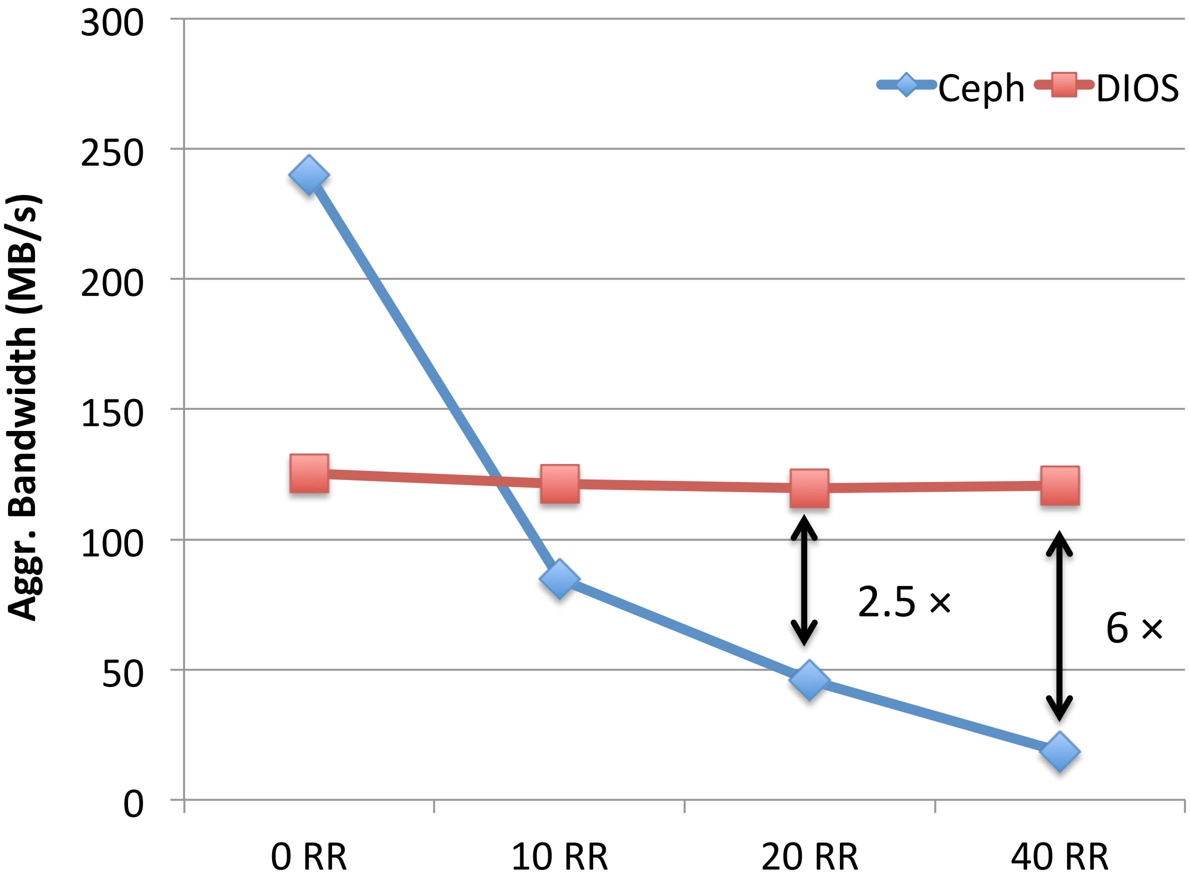
Evaluation – Sequential Workloads



Workload:
20 sequential, mixing with different numbers of random workloads.

Total data disks: 6

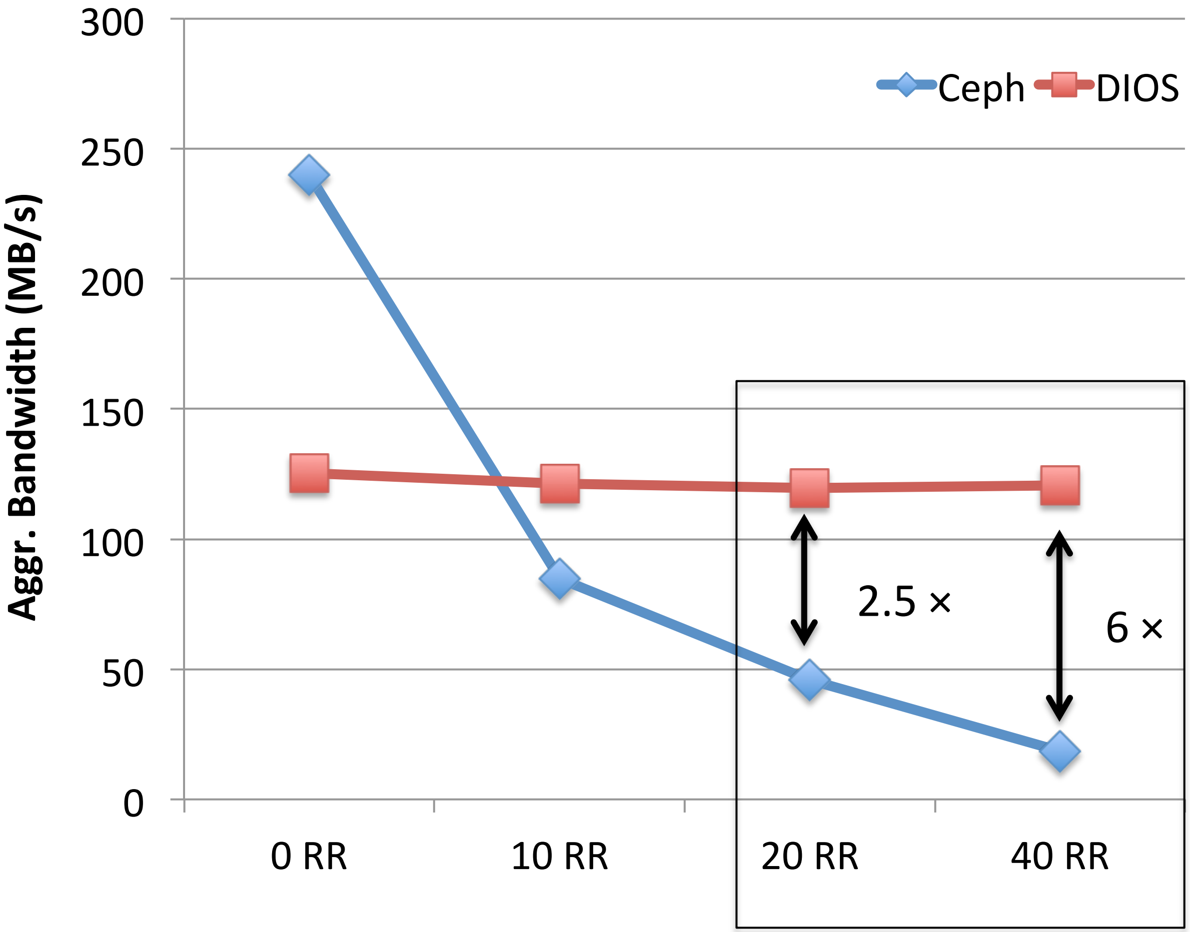
Evaluation – Sequential Workloads



Workload:
20 sequential, mixing with different numbers of random workloads.

Total data disks: 6

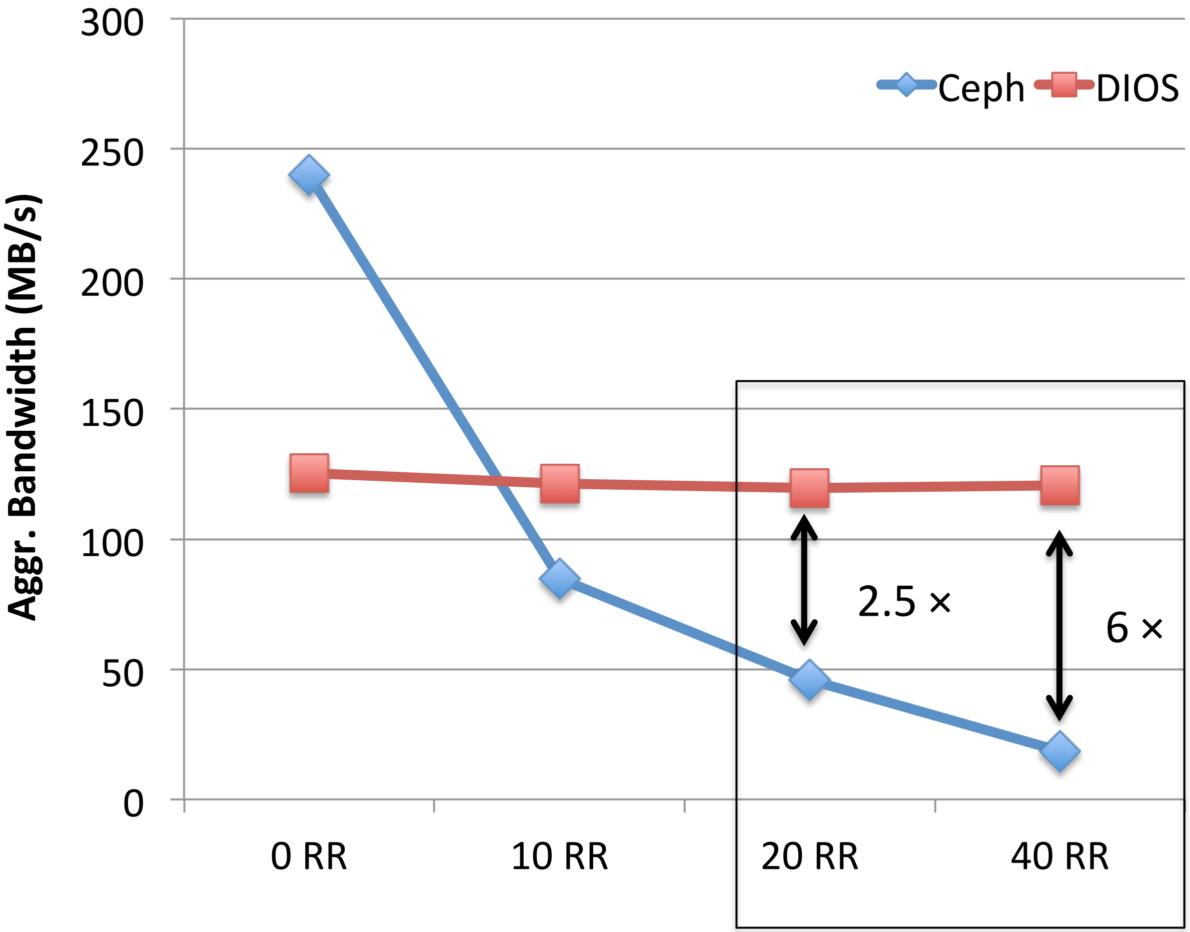
Evaluation – Sequential Workloads



Workload:
20 sequential, mixing with different numbers of random workloads.

Total data disks: 6

Evaluation – Sequential Workloads

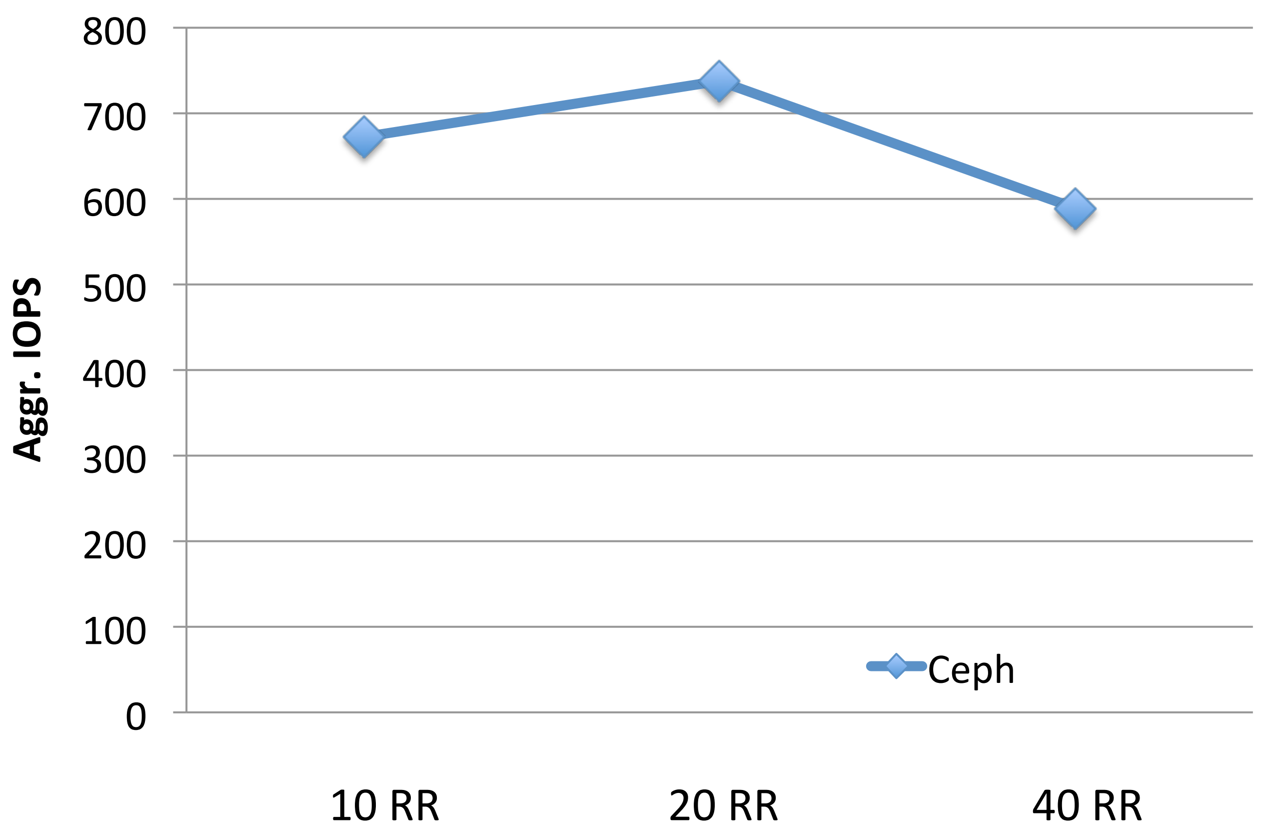


Workload:
20 sequential, mixing with different numbers of random workloads.

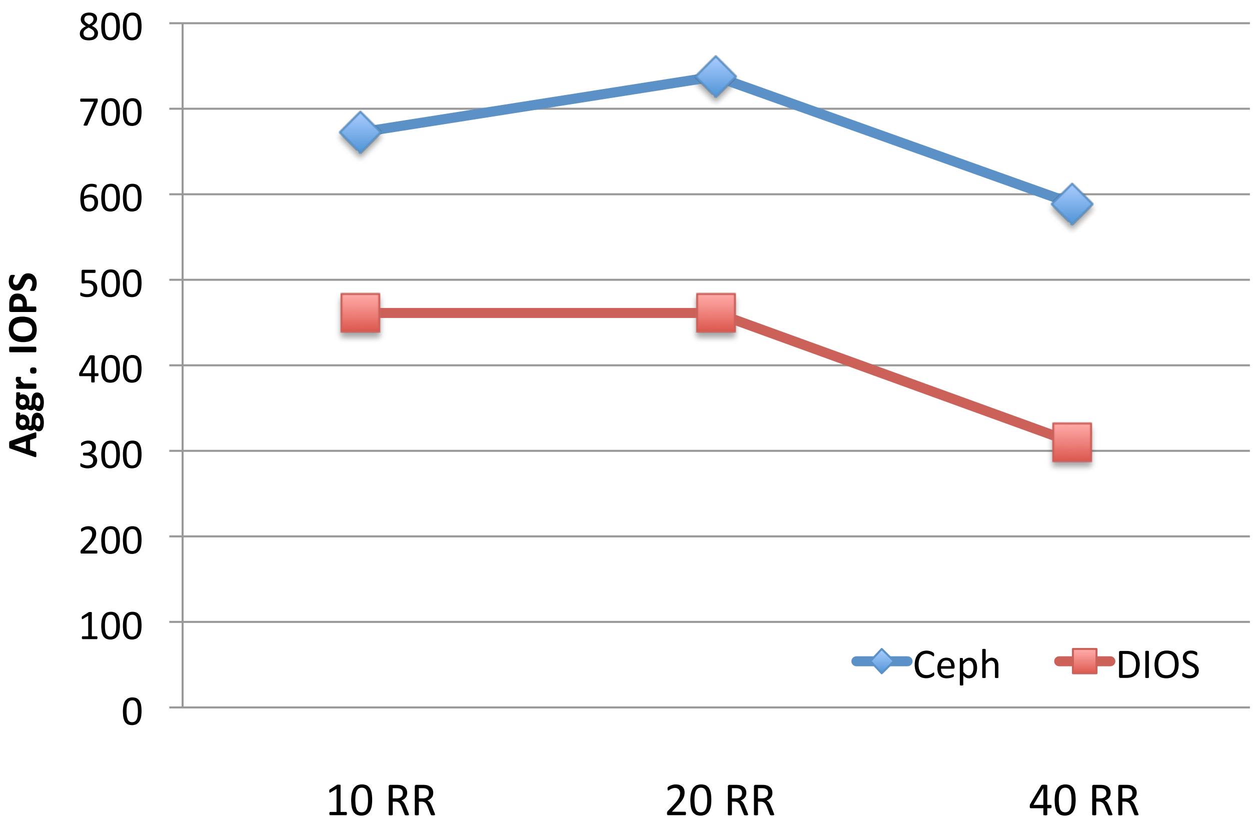
Total data disks: 6

DIOS:
- Consistent performance
- Higher bandwidth (disk util)

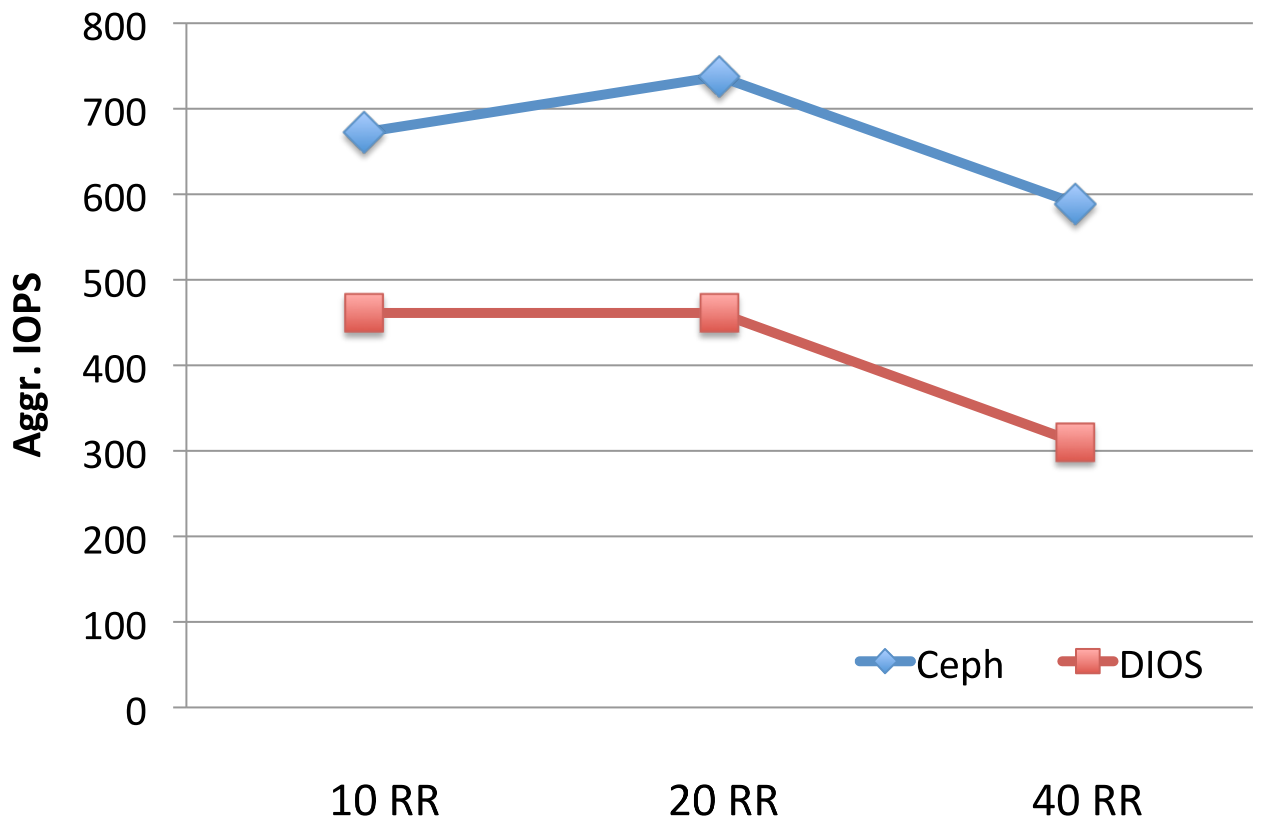
Evaluation – Random Workloads



Evaluation – Random Workloads

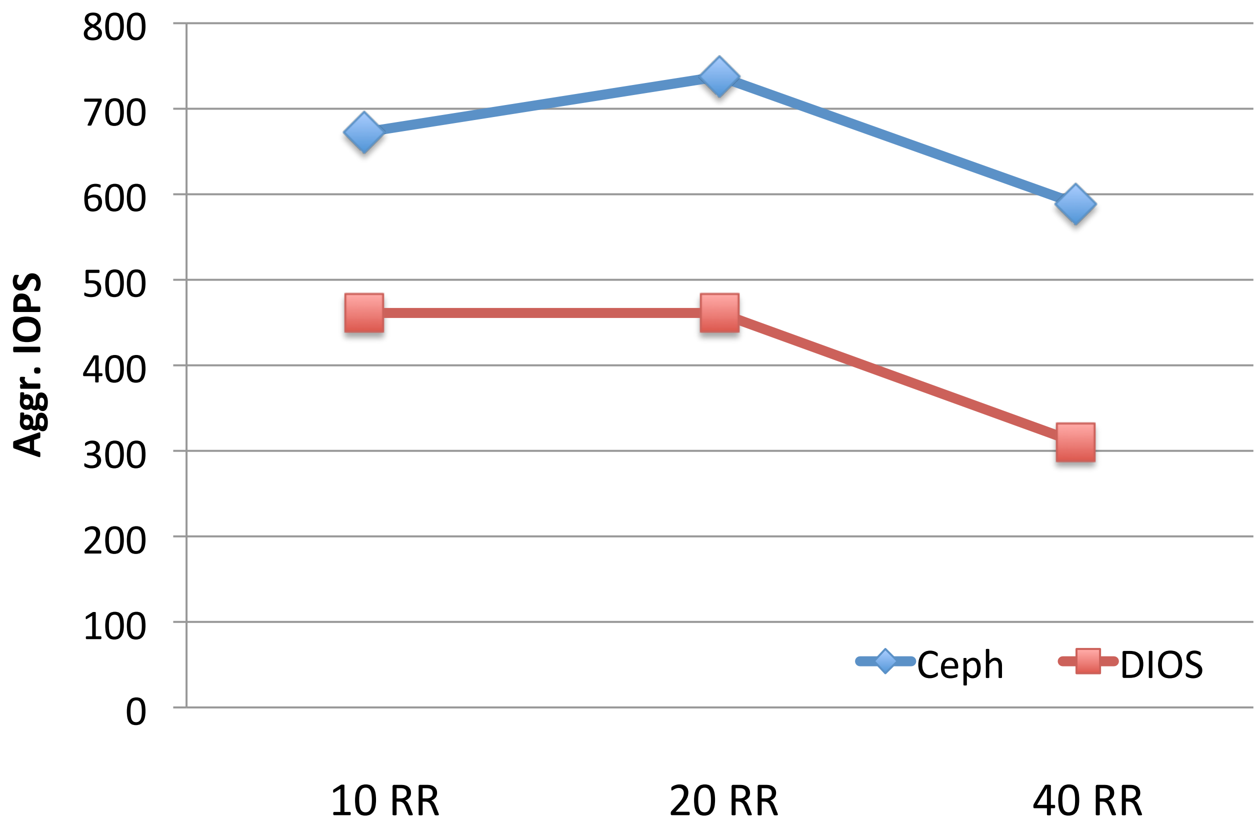


Evaluation – Random Workloads



Half of disks are serving random workloads

Evaluation – Random Workloads



Half of disks are serving random workloads

High disk utilization for the other half of disks

Summary

- It is more ***efficient*** and leads to ***more predictable*** performance, by dedicating a disk to serve a single type of read requests.
- Future directions
  - Write workloads
  - Extension to 3-way replication (load balance)

Towards Fair Sharing of Block Storage in a Multi-tenant Cloud
Xing Lin, Yun Mao, Feifei Li, Robert Ricci
HotCloud '12: 4th USENIX Workshop on Hot Topics in Cloud Computing, June 2012

Outline

- ✓ Introduction
  - ✓ Migratory Compression
  - ✓ Improve deduplication by separating metadata from data
  - ✓ Using deduplication for efficient disk image deployment
  - ✓ Performance predictability and efficiency for Cloud Storage Systems
- ✓ ***Conclusion***

Conclusion

- Problems:
  - Limitations in traditional compression and deduplication
  - Performance interference for cloud storage
- Use **similarity** in content and access patterns
  - Migratory compression: find **similarity** in block level and do re-organization
  - Deduplication: store **same** type of data blocks together
  - Differential IO scheduling: schedule **same** type of requests to same disk

Thesis Statement

Similarity in content and access patterns can be utilized to improve **space efficiency**,
by storing **similar** data together
and **performance predictability and efficiency**,
by scheduling **similar** IO requests to the same hard drive.

Acknowledgements

- **Advisor**, for ***always*** supporting my work
  - Robert Ricci
- **Committee members**, for providing feedback
  - Rajeev Balasubramonian
  - Fred Douglis
  - Feifei Li
  - Jacobus Van der Merwe
- **Flux members**
  - Eric Eide, Mike Hibler, David Johnson, Gary Wong, …
- **EMC/Data Domain** colleagues
  - Philip Shilane, Stephen Smaldone, Grant Wallace, …
- **Families and friends**, for kindly support

Acknowledgements

- **Advisor**, for ***always*** supporting my work
  - Robert Ricci (HotCloud12, HotStorage15, TridentCom15)
- **Committee members**, for providing feedback
  - Rajeev Balasubramonian (WDDD11)
  - Fred Douglis (FAST14, HotStorage15)
  - Feifei Li (HotCloud12)
  - Jacobus Van der Merwe (SOCC15)
- **Flux members**
  - Eric Eide, Mike Hibler, David Johnson, Gary Wong, …
- **EMC/Data Domain** colleagues
  - Philip Shilane, Stephen Smaldone, Grant Wallace, …
- **Families and friends**, for kindly support

Thank You !

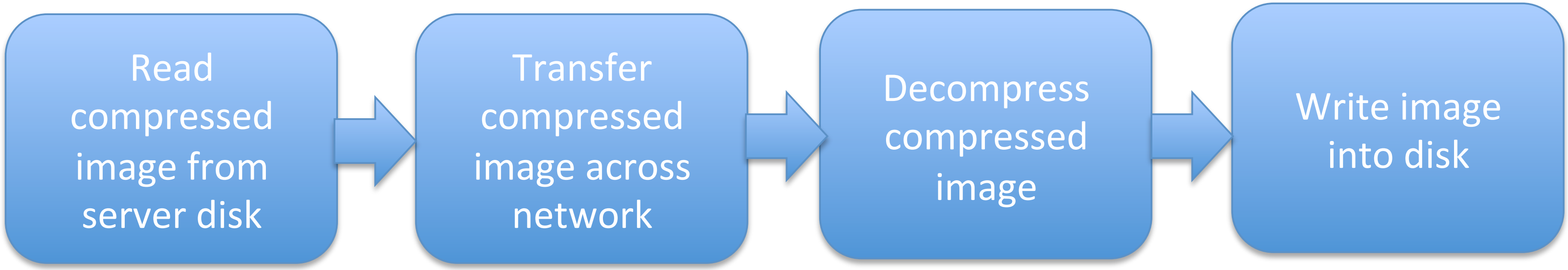
Backup Slides

VF - Related Work

- Deduplication analysis for virtual machine images [Jin SYSTOR '09]
  - **Benefit**: 70~80% of blocks are duplicates
  - Only analysis
- LiveDFS: deduplicating filesystem for storing disk images for OpenStack [Ng middleware '11]
  - Focus on spatial locality and metadata prefetching
  - Scales linearly and only 4 VMs; no compression
- Emustore: an early attempt for this project [Pullakandam Master Thesis 2011]
  - Compression during image deployment
  - Used regular fixed-size chunking

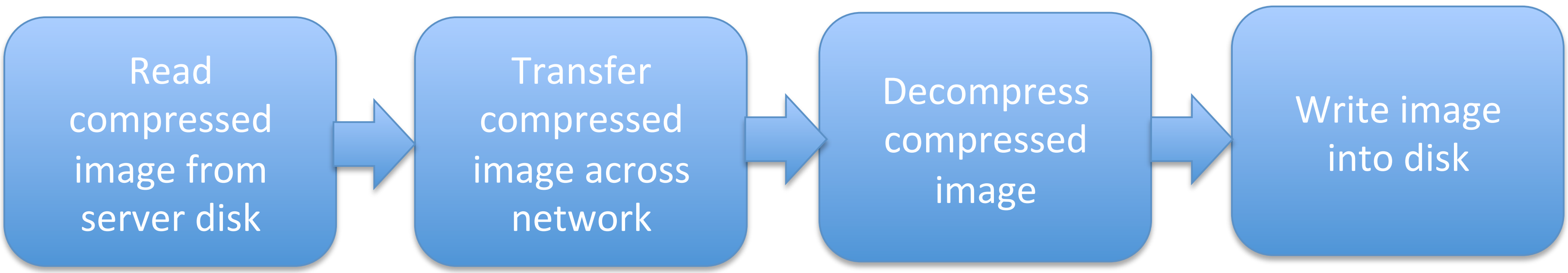
Disk Image Deployment Pipeline

- Compression factor is 3.18X for this image
- Available network bandwidth: 500 Mb/s (60 MB/s)



Disk Image Deployment Pipeline

- Compression factor is 3.18X for this image
- Available network bandwidth: 500 Mb/s (60 MB/s)

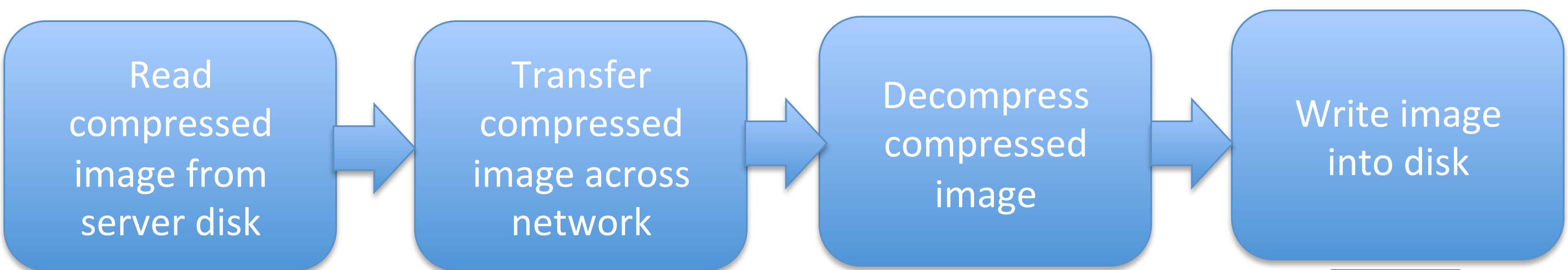


480 (150) | 172.84 (54.26) | 149.10 | 71.07

Effective throughput (uncompressed data) (MB/s)

Disk Image Deployment Pipeline

- Compression factor is 3.18X for this image
- Available network bandwidth: 500 Mb/s (60 MB/s)

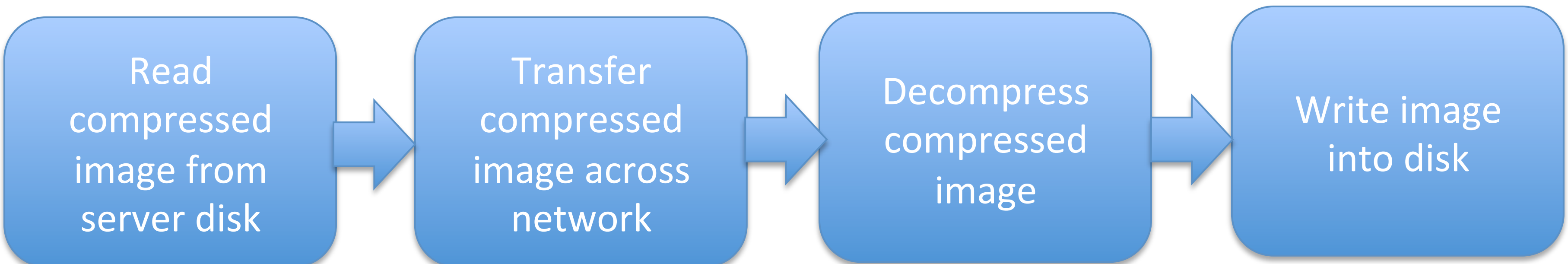


480 (150) | 172.84 (54.26) | 149.10 | 71.07

Effective throughput (uncompressed data) (MB/s)

Disk Image Deployment Pipeline

- Compression factor is 3.18X for this image
- Available network bandwidth: 500 Mb/s (60 MB/s)

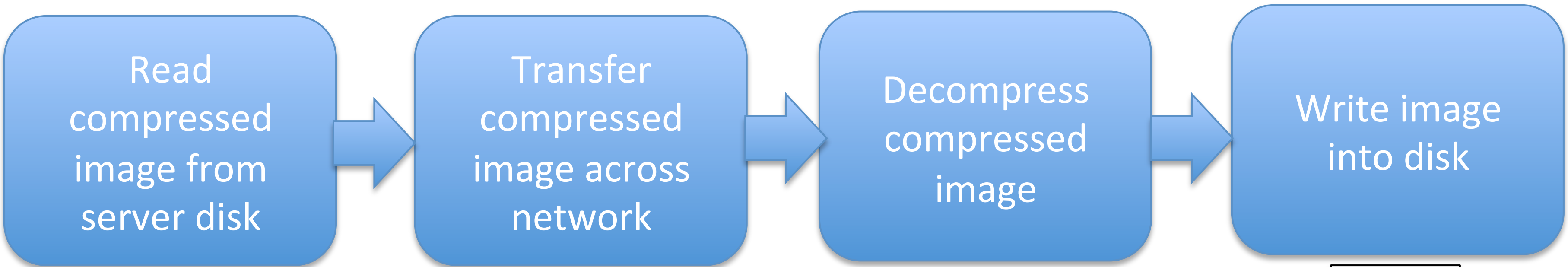


480 (150) 172.84 (54.26) 149.10 71.07

Effective throughput (uncompressed data) (MB/s)

Disk Image Deployment Pipeline

- Compression factor is 3.18X for this image
- Available network bandwidth: 500 Mb/s (60 MB/s)

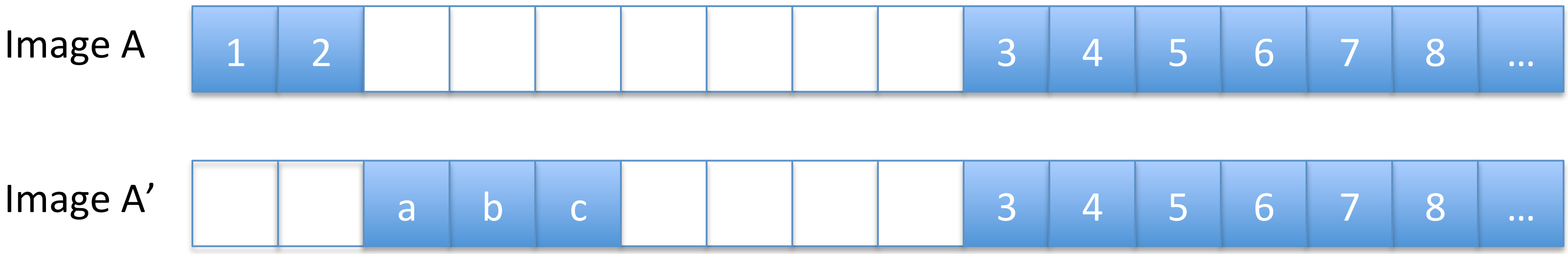


480 (150) | 172.84 (54.26) | 149.10 | 71.07

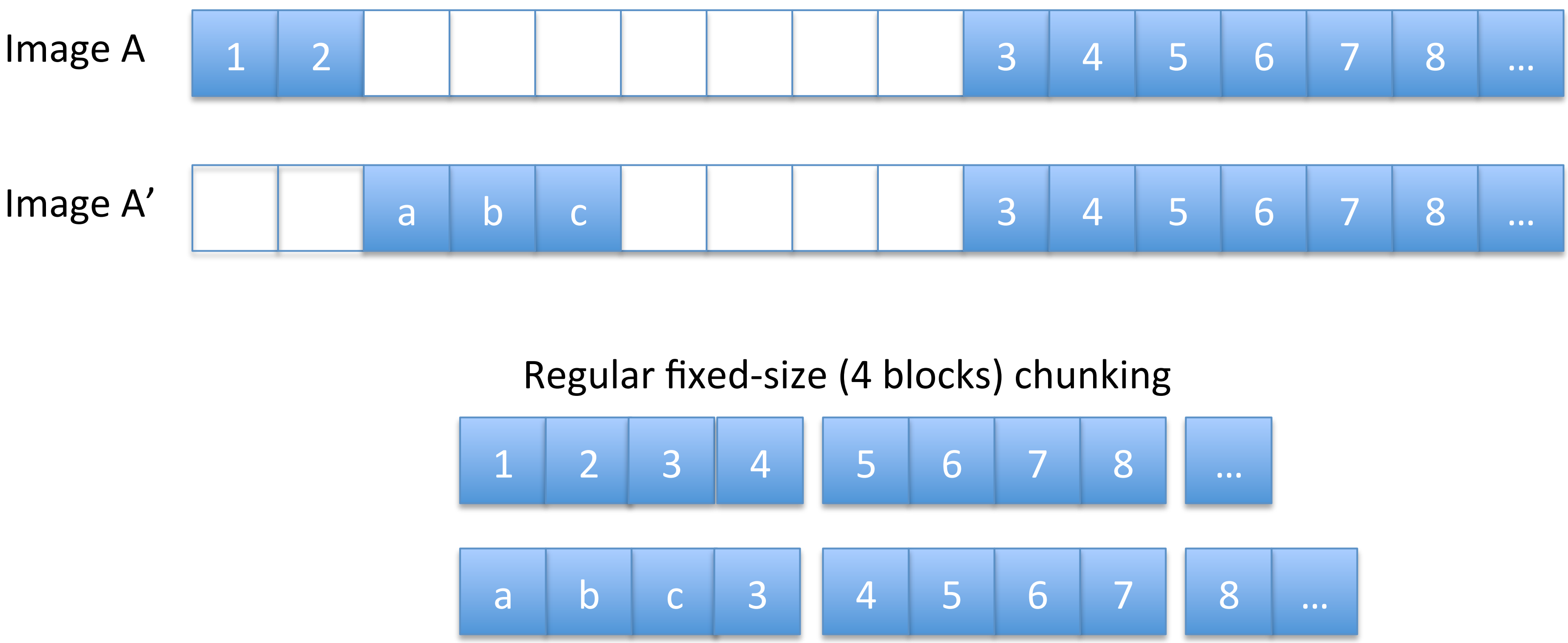
Effective throughput (uncompressed data) (MB/s)

Compression	**29.68**

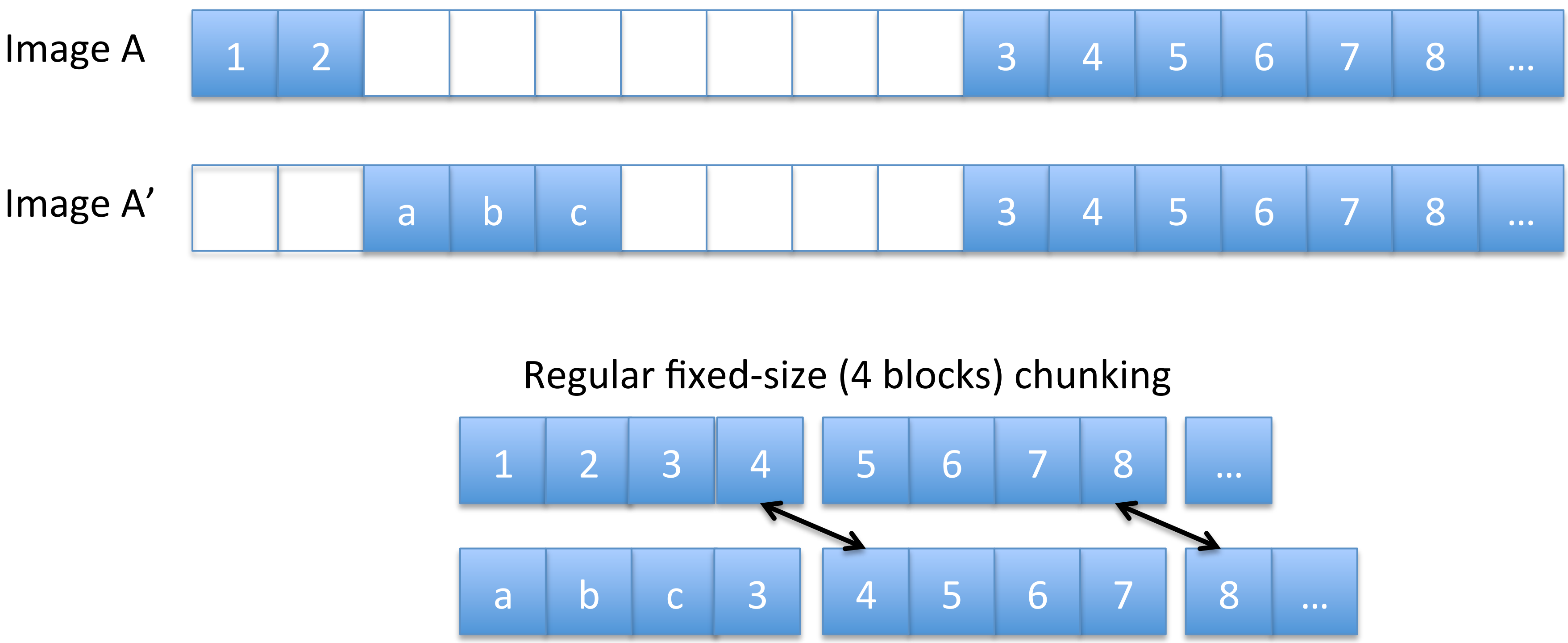
Chunk Boundary Shift Problem



Chunk Boundary Shift Problem

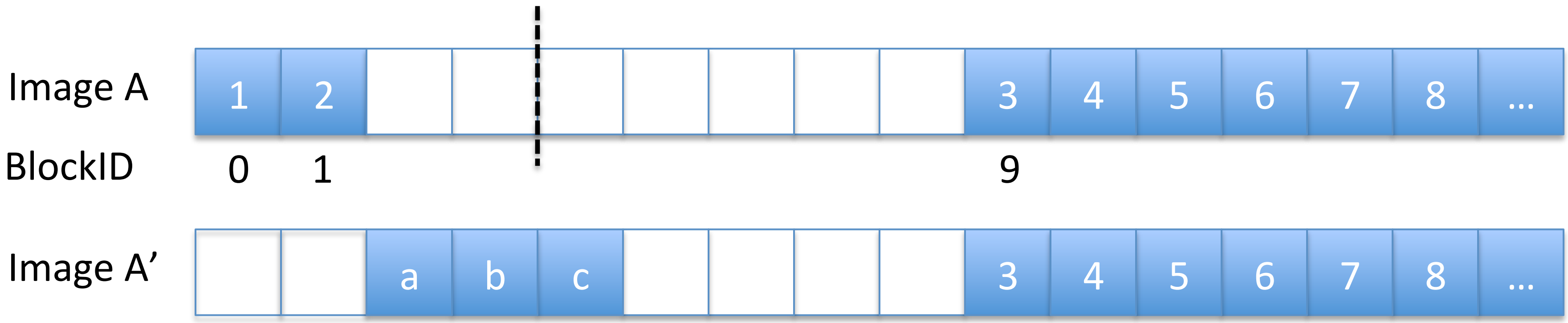


Chunk Boundary Shift Problem

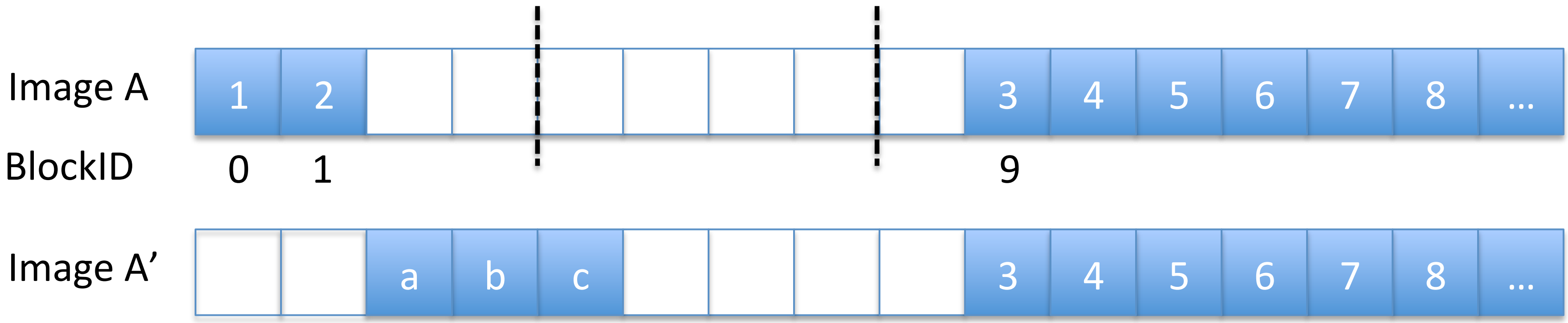


Aligned Fixed-size Chunking (AFC)

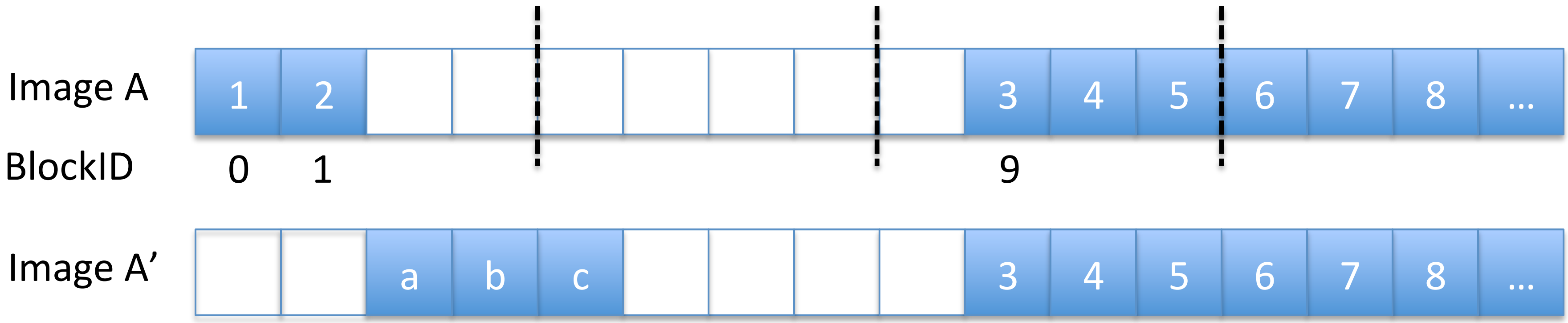
Aligned Fixed-size Chunking (AFC)



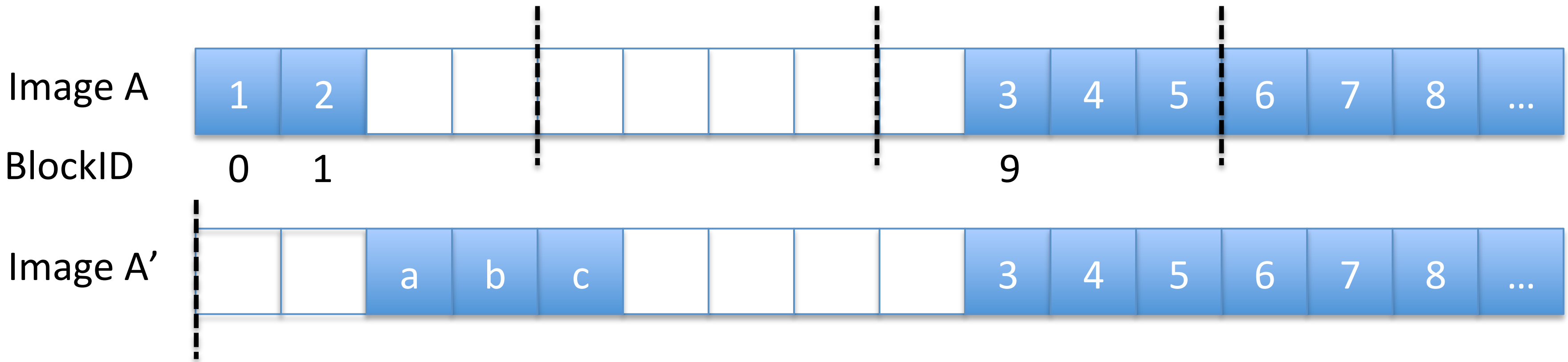
Aligned Fixed-size Chunking (AFC)



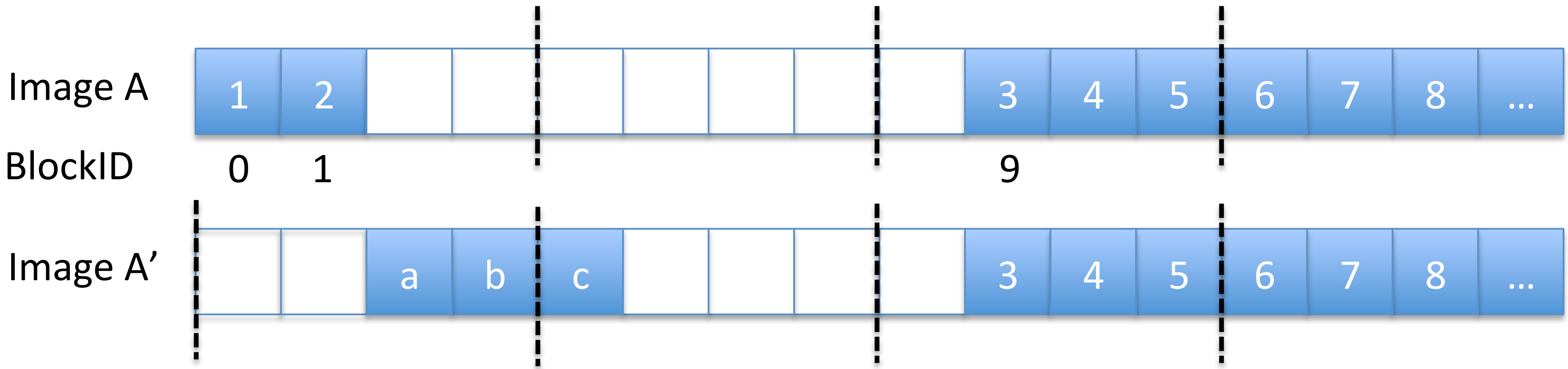
Aligned Fixed-size Chunking (AFC)



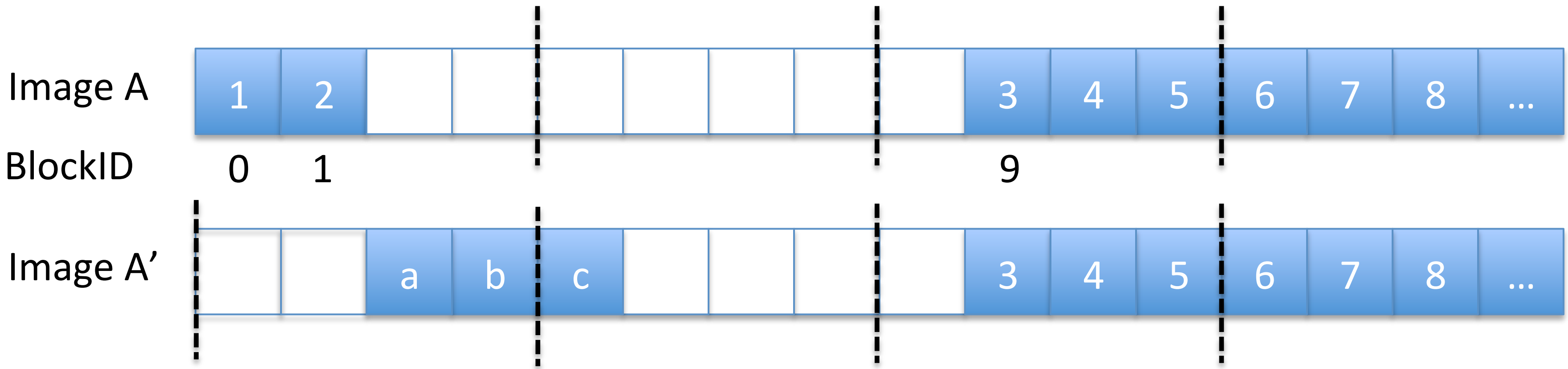
Aligned Fixed-size Chunking (AFC)



Aligned Fixed-size Chunking (AFC)

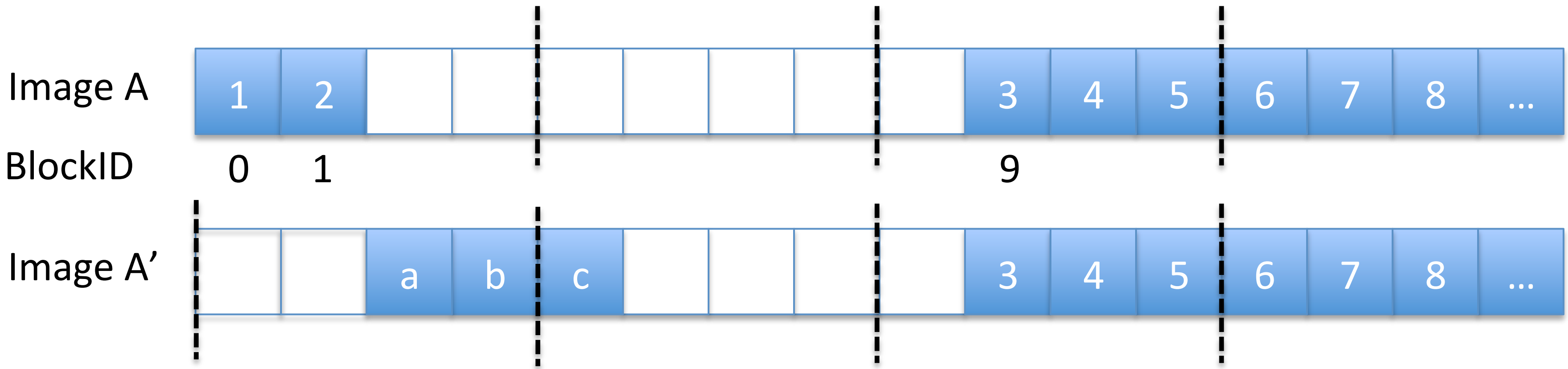


Aligned Fixed-size Chunking (AFC)

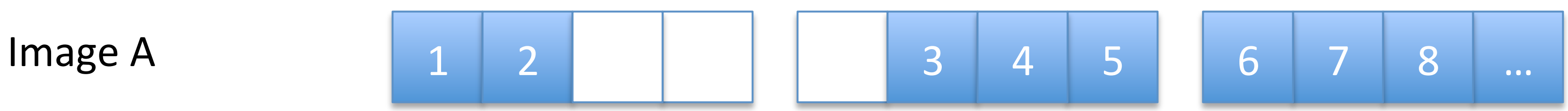


Aligned Fixed-size (4 blocks) chunks

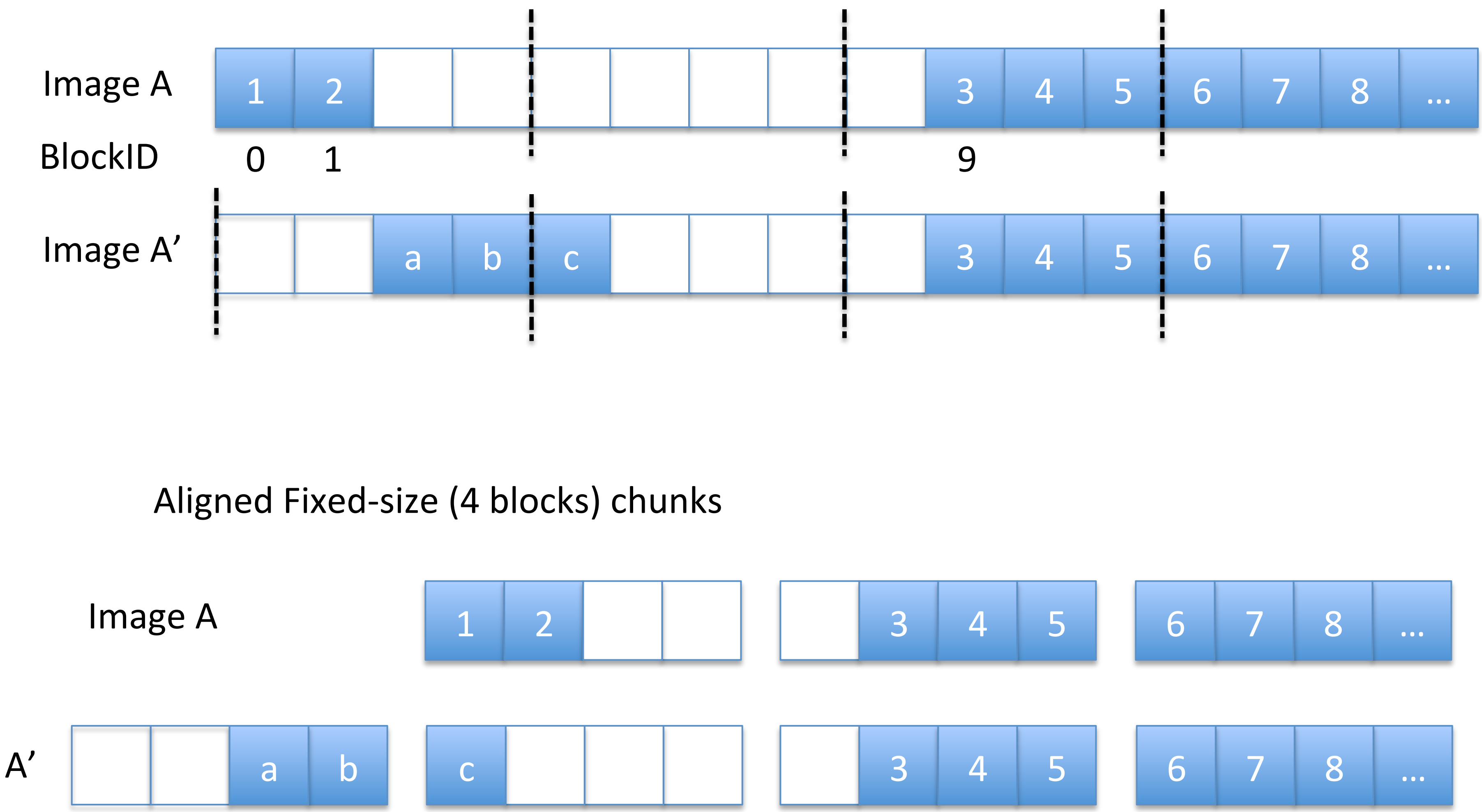
Aligned Fixed-size Chunking (AFC)



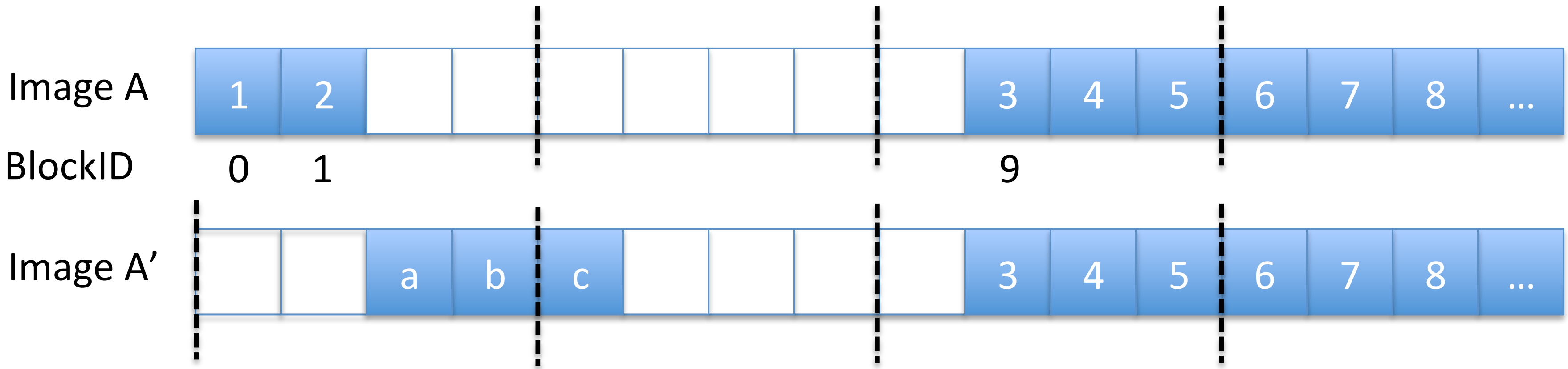
Aligned Fixed-size (4 blocks) chunks



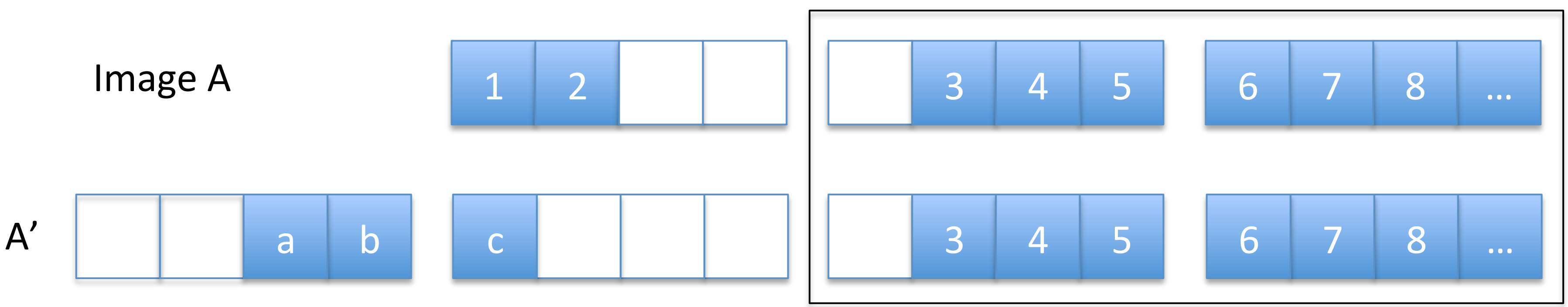
Aligned Fixed-size Chunking (AFC)



Aligned Fixed-size Chunking (AFC)



Aligned Fixed-size (4 blocks) chunks



AFC - Evaluation