Distributed Storage Systems

Data Deduplication

Agenda

Today's topics

- Data deduplication
- Practical ideas
- A generalized view of data deduplication

Class Structure

	Lecture	Lab
Week 1	Course introduction, networking basics, socket programming	Python sockets
Week 2	RPC, NFS, Practical RPC	Flask, JsonRPC, REST API
Week 3	AFS, reliable storage introduction	ZeroMQ, ProtoBuf
Week 4	Hard drives, RAID levels	RPi stack intro, RPi RAID with ZMQ
Week 5	Finite fields, Reed-Solomon Codes	Kodo intro, RS and RLNC with Kodo
Week 6	Repair problem, RS vs Regenerating codes	RPi simple distributed storage with Kodo RS
Week 7	Regenerating codes, XORBAS	RPi Regenerate lost fragments with RS
Week 8	Hadoop	RPi RLNC, recovery with recode
Week 9	Storage Virtualization, Network Attached Storage, Storage Area Networks	RPi basic HDFS (namenode+datanode, read & write pipeline)
Week 10	Object Storage	RPi basic S3 API
Week 11	Compression, Delta Encoding	Mini project consultation
Week 12	**Data Deduplication**	**RPi Dedup**
Week 13	Fog storage	Mini project consultation
Week 14	Security for Storage Systems and Recap	Mini project consultation

Cost of Storing Data

- …

Cost of Storing Data

- Individual file/data sizes
- Redundancy introduced for protection

Cost of Storing Data

- Individual file/data sizes
- Redundancy introduced for protection

Can we reduce the storage costs?

Cost of Storing Data

- Individual file/data sizes
- Redundancy introduced for protection

Can we reduce the storage costs?

- RAID / Erasure Codes / Network Codes
- ...

Cost of Storing Data

- Individual file/data sizes
- Redundancy introduced for protection

Can we reduce the storage costs?

- RAID / Erasure Codes / Network Codes
- Compress individual files
- ...

Cost of Storing Data

- Individual file/data sizes
- Redundancy introduced for protection

Can we reduce the storage costs?

- RAID / Erasure Codes / Network Codes
- Compress individual files
- Data deduplication (say what?)

Basic example

Imagine:

- You have a mail server
- An email is sent to 100 of your employees
- Email has a 10MB attachment (same for everyone)
- What is the cost of that email to you?

Basic example

Imagine:

- You have a mail server
- An email is sent to 100 of your employees
- Email has a 10MB attachment (same for everyone)
- What is the cost of that email to you?
  - ~1000MB + 100x (Rest of email) (without reliability)
  - ~3000MB + 300x (Rest of email) (3-way replication)
- Can we do better?
  - Store the 10MB

Basic example

Imagine:

- You have a mail server
- An email is sent to 100 of your employees
- Email has a 10MB attachment (same for everyone)
- What is the cost of that email to you?
    - ~1000MB + 100x (Rest of email) (without reliability)
    - ~3000MB + 300x (Rest of email) (3-way replication)
- Can we do better?
    - Store the 10MB once per receiver
    - 10MB + overhead + 100x(Rest of email)

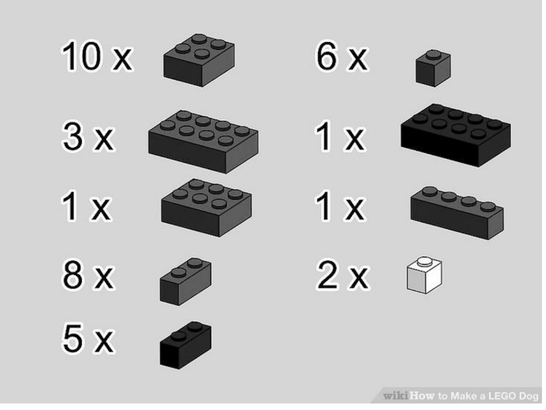
Basic example

Imagine:

- You have a mail server
- An email is sent to 100 of your employees
- Email has a 10MB attachment (same for everyone)
- What is the cost of that email to you?
  - ~1000MB + 100x (Rest of email) (without reliability)
  - ~3000MB + 300x (Rest of email) (3-way replication)
- Can we do better?
  - Store the 10MB once per receiver
  - 10MB + overhead + 100x(Rest of email)
- Do we need to do it in a full file?

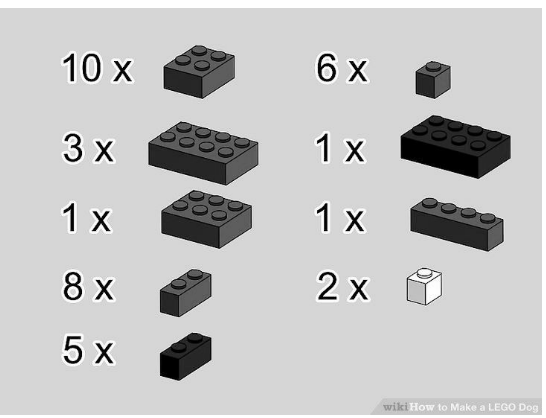
What is Deduplication?

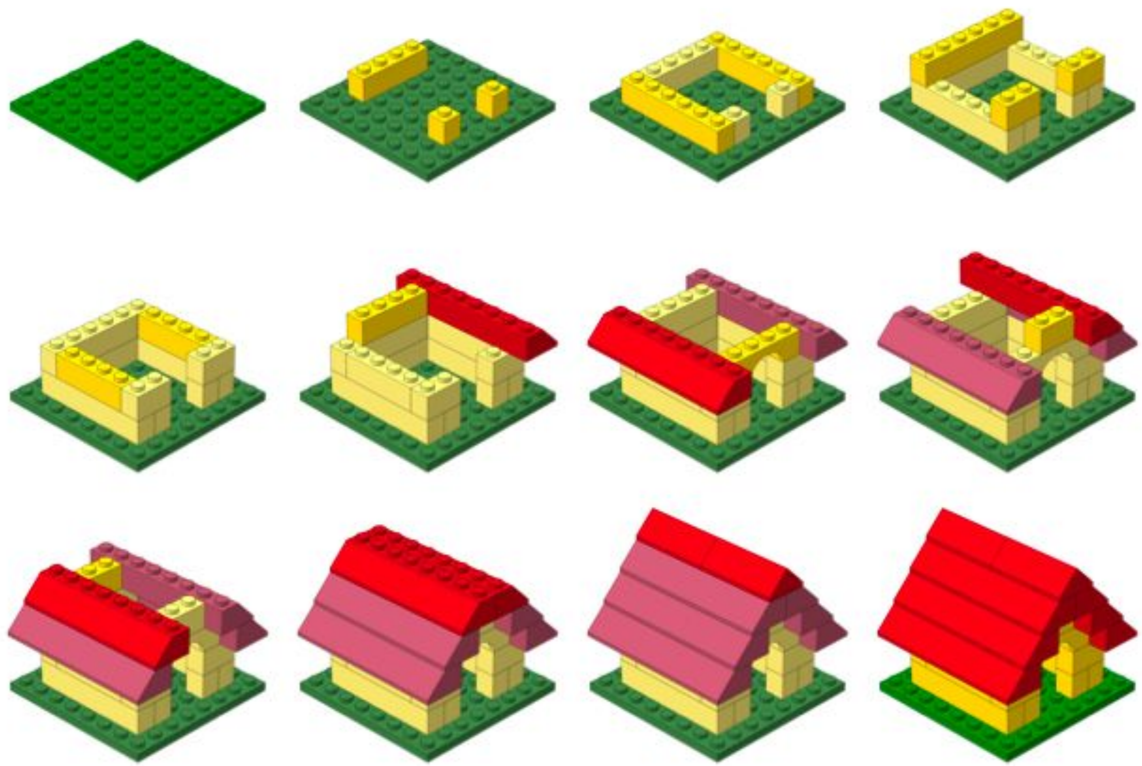
Think of it as a LEGO problem



What is Deduplication?

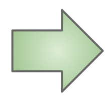
Think of it as a LEGO problem

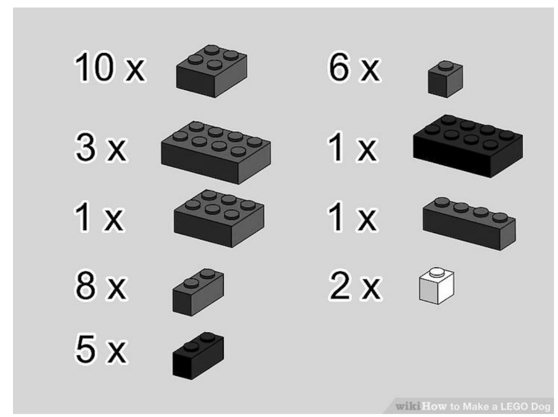




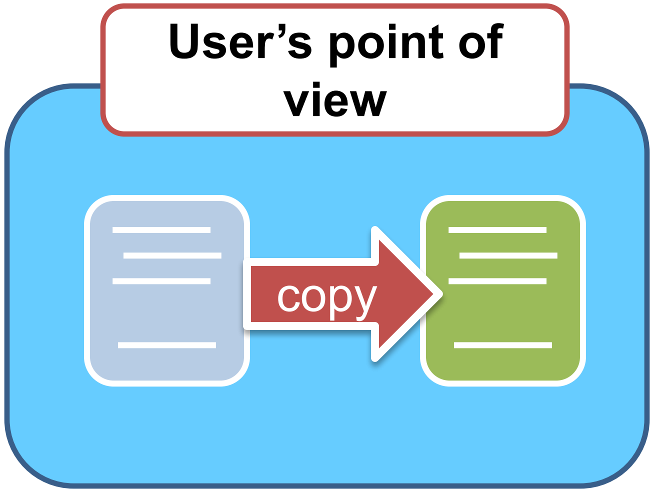
What is Deduplication?

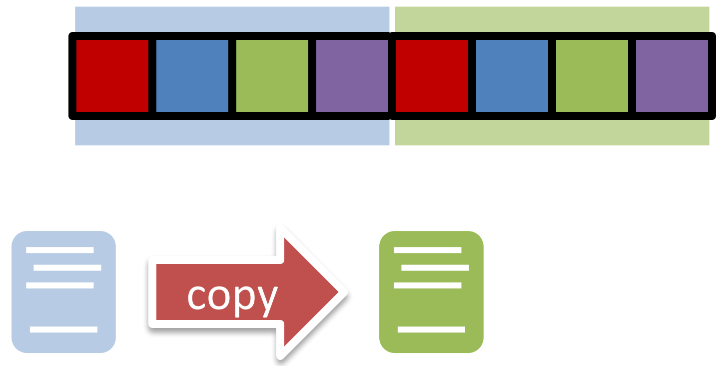
But you can also take the data objects and convert them into pieces:





What is Deduplication?

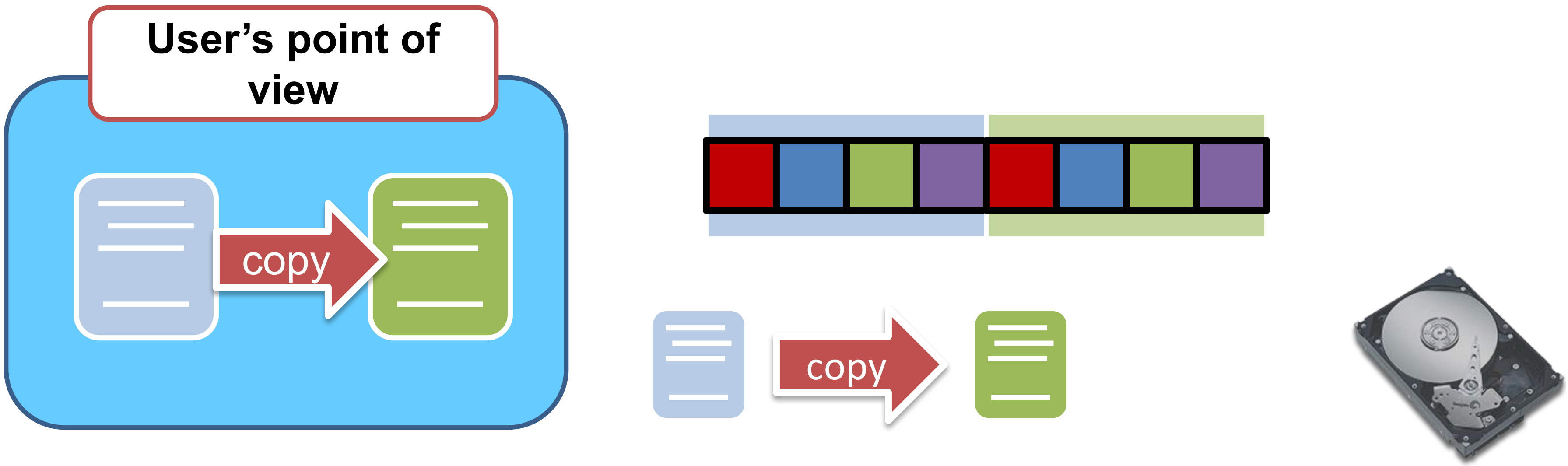




Goal: Eliminate storage of data with same content

Slide adapted from Patrick Lee

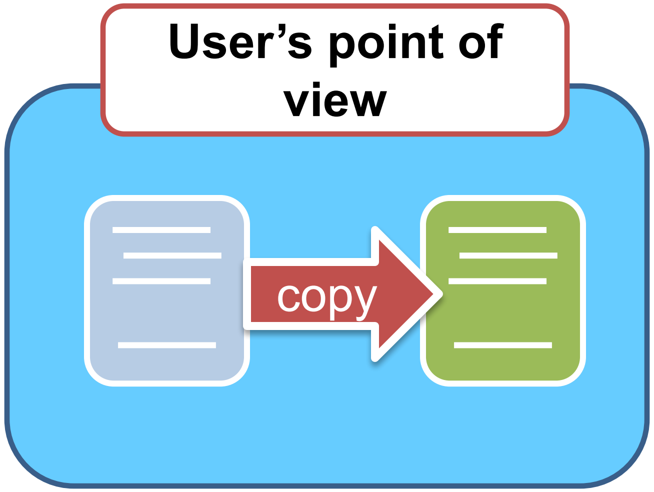
What is Deduplication?

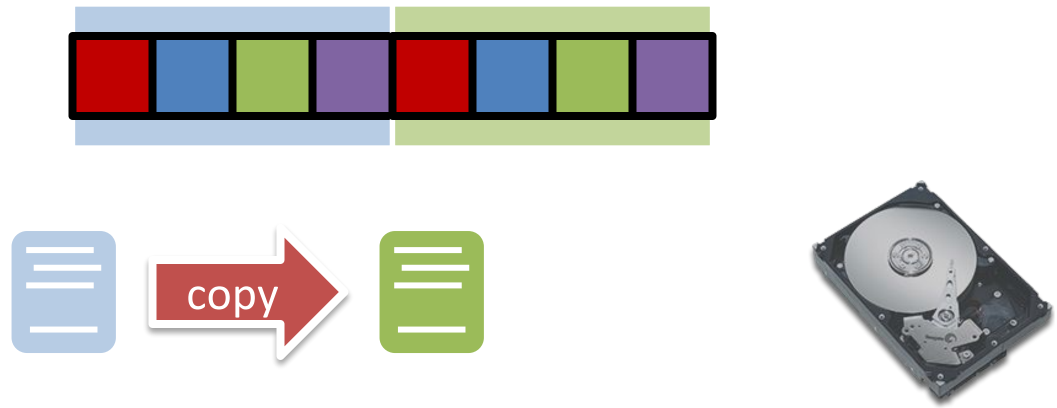


Goal: Eliminate storage of data with same content

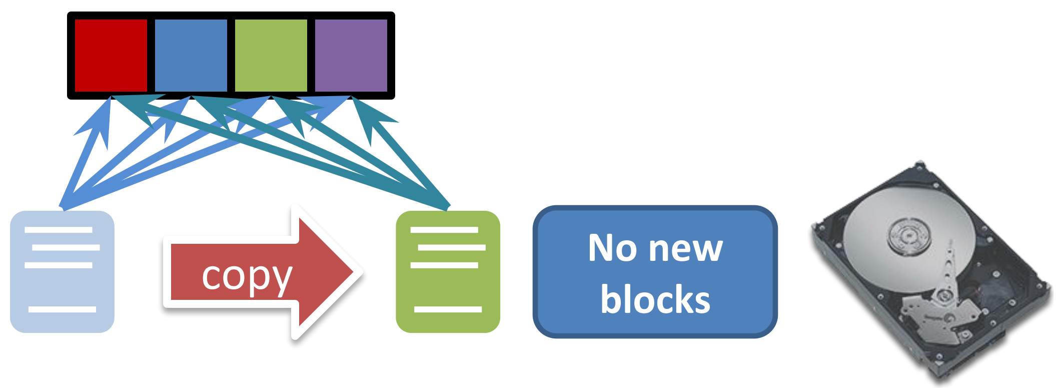
Slide adapted from Patrick Lee

What is Deduplication?



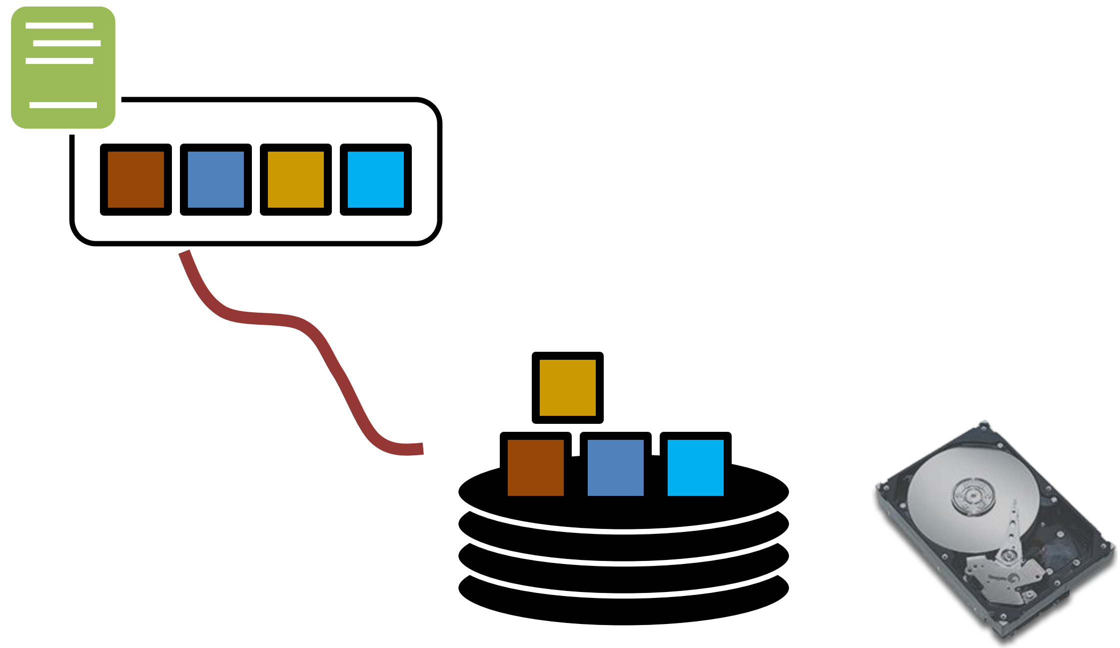


Goal: Eliminate storage of data with same content



Slide adapted from Patrick Lee

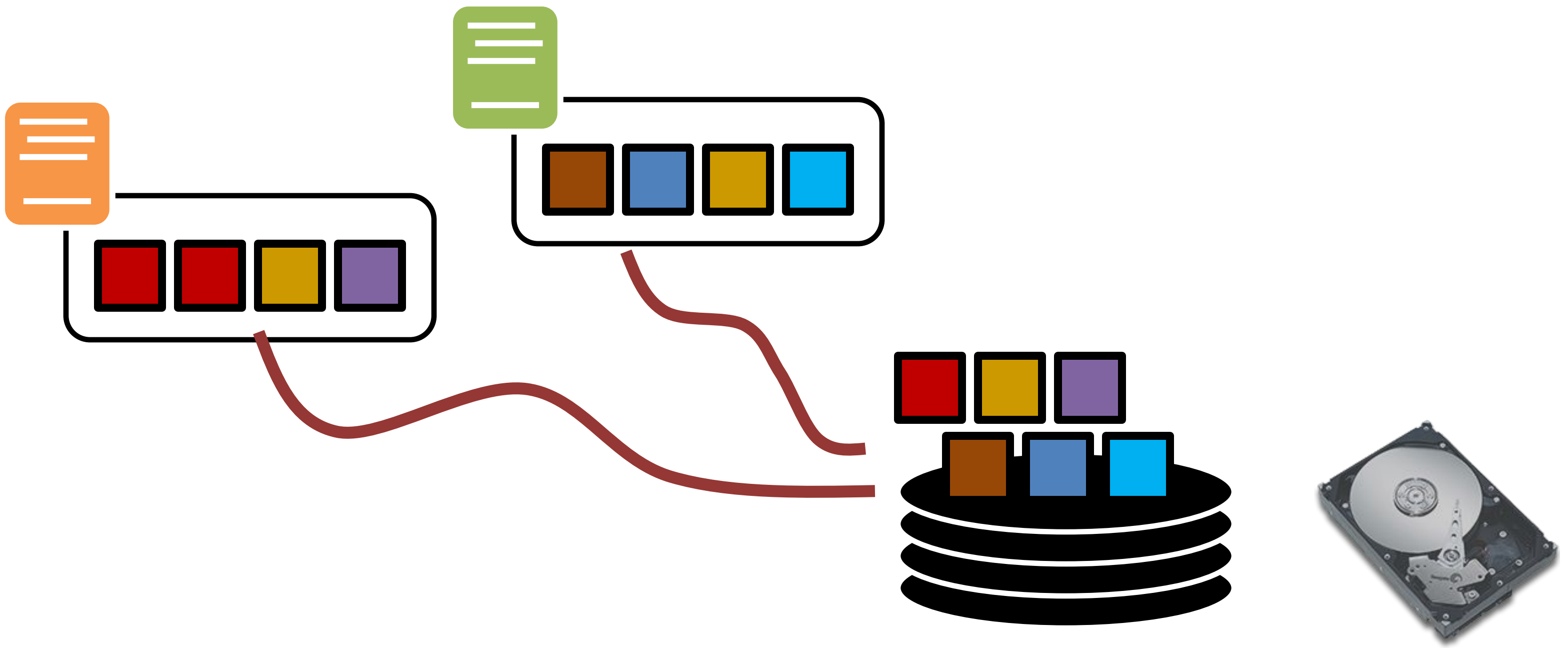
Block-level deduplication



Example: storage space reduced by ....

Slide adapted from Patrick Lee

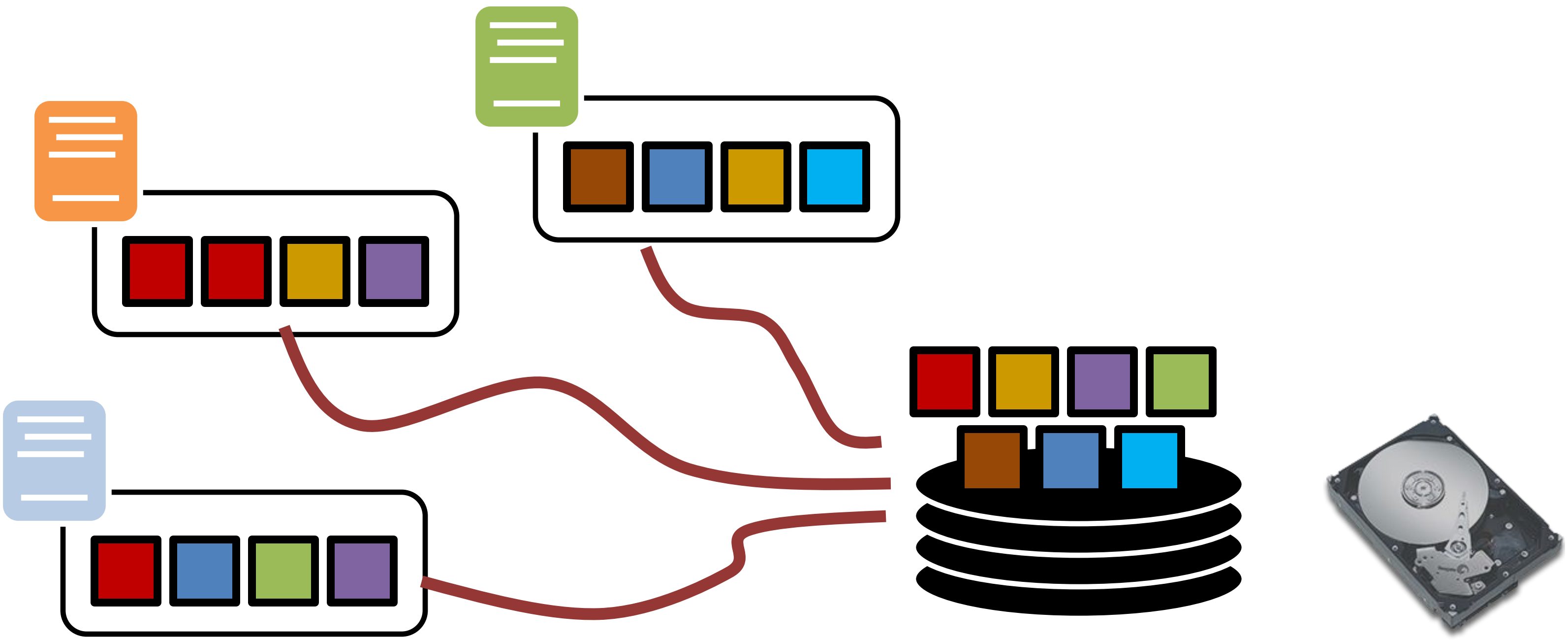
Block-level deduplication



Example: storage space reduced by ….

Slide adapted from Patrick Lee

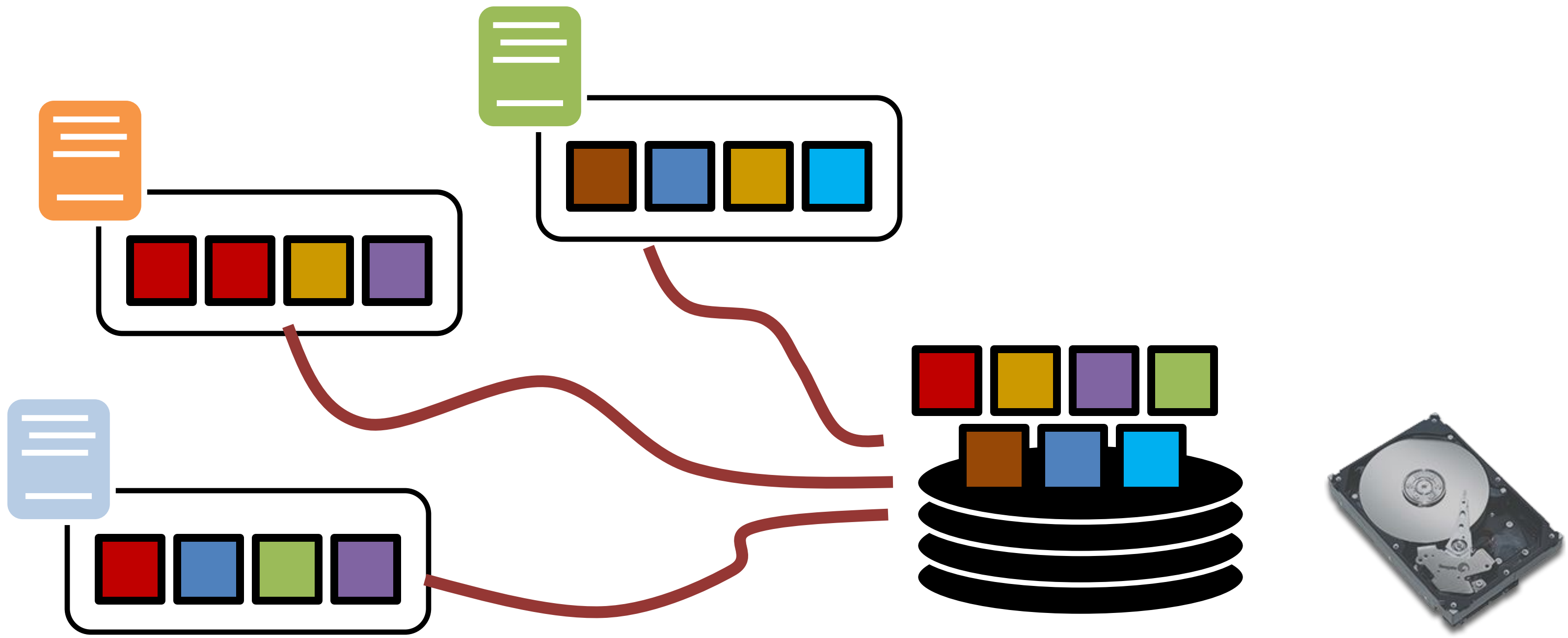
Block-level deduplication



Example: storage space reduced by ....

Slide adapted from Patrick Lee

Block-level deduplication



Example: storage space reduced by 5/12 = 42%

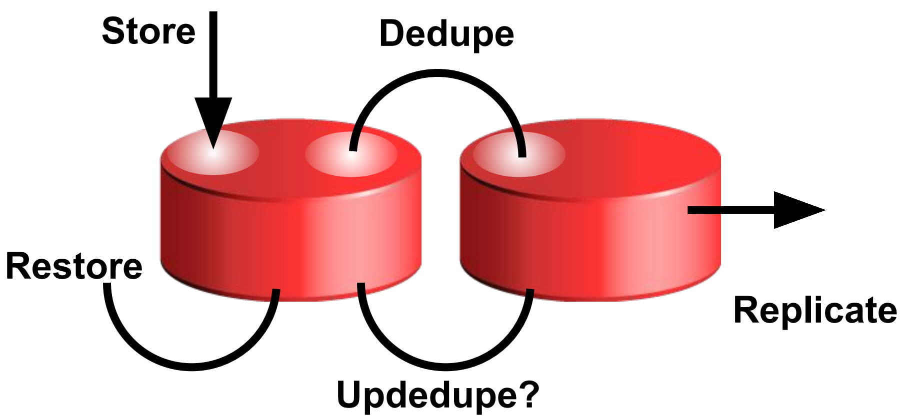
Performance tends to improve as the system stores more data

Slide adapted from Patrick Lee

Post-process vs Inline

Post-process
(Deduplication after storage)

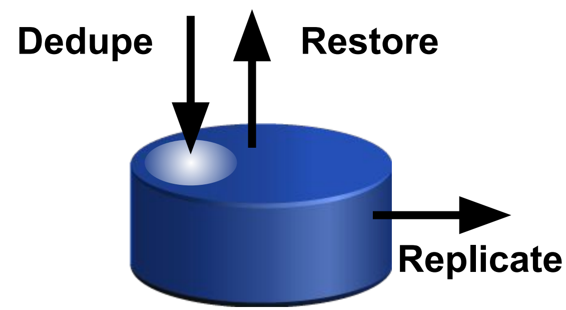
At least 3x disk accesses
to shared store



Process contention increases with #processes
- Copy to tape: Too slow to stream tape
- Recovery: SLA predictability
- Replication: Poor time-to-DR
- Deduplication itself if interleaved with backup or restore

More admin needed to fight these issues

Inline
(Deduplication during storage)



Other activities unimpeded
- Predictable
- Simpler

Challenges

- Can we preserve the performance?
- Can we support general file system operations?
  - Read, write, modify, delete
- Can we deploy deduplication on low-cost commodity systems?
  - e.g., a few GB of RAM, 32/64-bit CPU, standard OS

Slide adapted from Patrick Lee

Some Work in Deduplication

- Deduplication backup systems
  - e.g., Venti [Quinlan & Dorward ’02], Data Domain [Zhu et al. ’08], Foundation [Rhea et al. ’08]
  - Assume data is not modified or deleted
- Deduplication file systems
  - OpenSolaris ZFS, OpenDedup SDFS
  - Consume significant memory space, not for commodity systems
- VM image storage
  - e.g., Lithium [Hansen & Jul ’10], mainly on fault tolerance, but not on deduplication

Slide adapted from Patrick Lee

Basics: Layout

- Deduplication (typically) operates on fixed-size blocks
  - Saves one copy if two fixed-size blocks have the same content
- For VM image storage, deduplication efficiencies similar for fixed-size blocks and variable-size blocks [Jin & Miller, ’09]

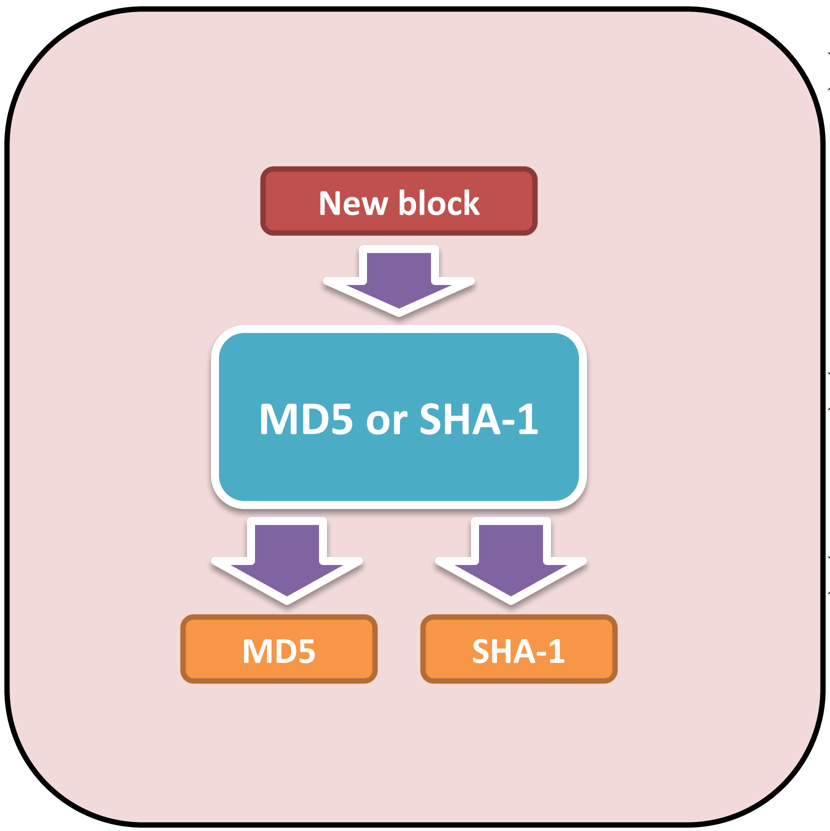
Slide adapted from Patrick Lee

Basics: Comparing blocks

- How to compare blocks?

Basics: Fingerprints

- How to compare blocks?
- Solution: Use cryptographic hashes (or **fingerprints**)



- ➢ Hash-based comparisons
  - Same content ➔ same hash
  - Different content ➔ different hashes with high probability
- ➢ Pros: block comparison reduced to hash comparison
- ➢ Cons: collision may occur, but with negligible probability

[Quinlan & Dorward, ’02]

Slide adapted from Patrick Lee

Basics: How often do I get matches?

Basics: How often do I get matches?

- Let us assume a bad scenario
  - Independent data blocks
  - Data blocks have equal probability to occur

Basics: How often do I get matches?

- Let us assume a bad scenario
  - Independent data blocks
  - Data blocks have equal probability to occur
- Familiar with the birthday paradox?
  - Imagine you have N people and 365 possible birthdays, how many people do you need to have a probability of at least 2 having the same birthdate
  -

Basics: How often do I get matches?

- Let us assume a bad scenario
  - Independent data blocks
  - Data blocks have equal probability to occur
- Familiar with the birthday paradox?
  - Imagine you have N people and 365 possible birthdays, how many people do you need to have a probability of at least 2 having the same birthdate.
  - Number of values to test to have a probability of at least one collision greater than ½ is $N^{1/2}$ for N possible values

    → Block of size $N = 2^m$

    → Need $2^{m/2}$

Basics: How often do I get matches?

- Let us assume a bad scenario
  - Independent data blocks
  - Data blocks have equal probability to occur
- Familiar with the birthday paradox?
  - Imagine you have N people and 365 possible birthdays, how many people do you need to have a probability of at least 2 having the same birthdate.
  - Number of values to test to have a probability of at least one collision greater than ½ is $N^{1/2}$ for N possible values

    → Block of size $N = 2^m$
    → Need $2^{m/2}$

Example: m = 10 bits → N = 1024, need $2^5 = 32$

Basics: How often do I get matches?

- Let us assume a bad scenario
  - Independent data blocks
  - Data blocks have equal probability to occur
- Familiar with the birthday paradox?
  - Imagine you have N people and 365 possible birthdays, how many people do you need to have a probability of at least 2 having the same birthdate.
  - Number of values to test to have a probability of at least one collision greater than ½ is $N^{1/2}$ for N possible values

    → Block of size $N = 2^m$

    → Need $2^{m/2}$

Example: m = 1000 bits → $N = 2^{1000}$ → 2^{500} ~$3x10^{150}$

Basics: How often do I get matches?

- Let us assume a bad scenario
  - Independent data blocks
  - Data blocks have equal probability to occur
- Familiar with the birthday paradox?
  - Imagine you have N people and 365 possible birthdays, how many people do you need to have a probability of at least 2 having the same birthdate
  - Number of values to test to have a probability of at least one collision greater than ½ is $N^{1/2}$ for N possible values

    → Block of size $N = 2^m$

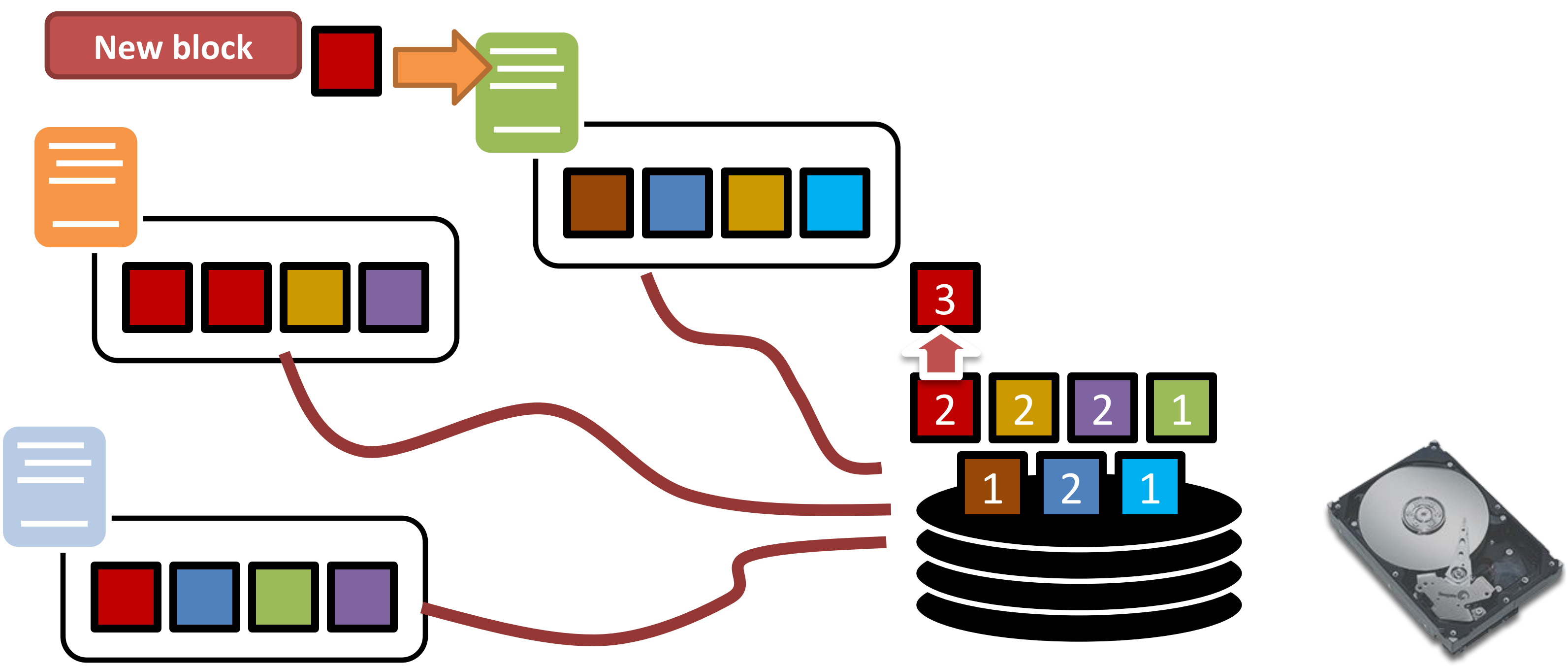
    → Need $2^{m/2}$

Typically, there is a lot more correlation in the data :-)

Basics: How to know if a block should be deleted?

Basics: Reference Counts

- How to know if a block should be deleted?
- **Solution:** Keep a **reference count** for each block. Zero means the block is no longer referenced



Slide adapted from Patrick Lee

Inline Deduplication

- How to check if a block being written can be deduplicated with existing blocks?
- Solution: maintain an **index structure**
  - Keep track of fingerprints of existing blocks
- Goal: design of index structure must be efficient in **space** and **speed**
- Two options of keeping an index structure:
  - Putting whole index structure in RAM
  - Putting whole index structure on disk

Option 1: Index Structure in RAM

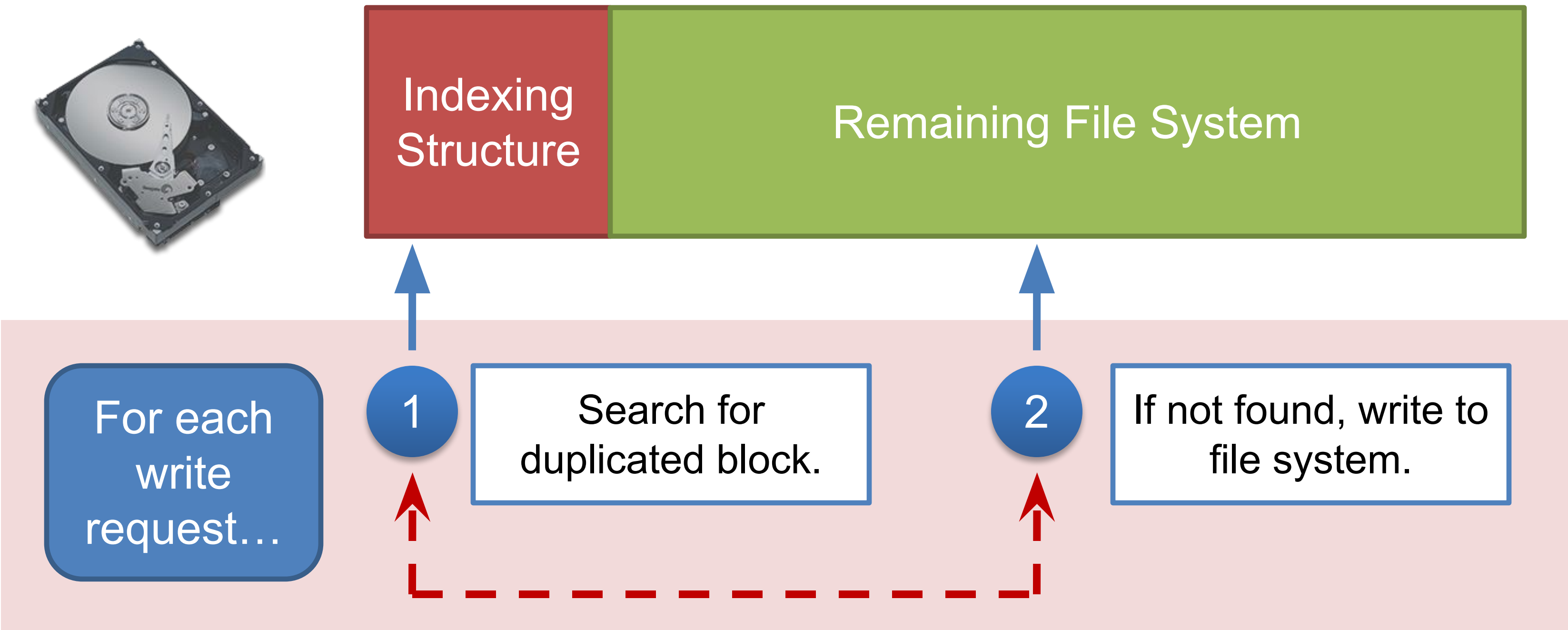
- How about putting whole index structure **in RAM**?
  - Used in existing dedup file systems (e.g., ZFS, OpenDedup)
- Challenge: need large amount of RAM
- Example: per 1TB of disk content

Block Size	4KB
Using MD5 checksum	16 bytes per block
Size of Index	1TB / 4KB x 16 bytes = **4GB**

Slide adapted from Patrick Lee

Option 2: Index Structure on Disk

- How about putting whole index structure **on disk**?

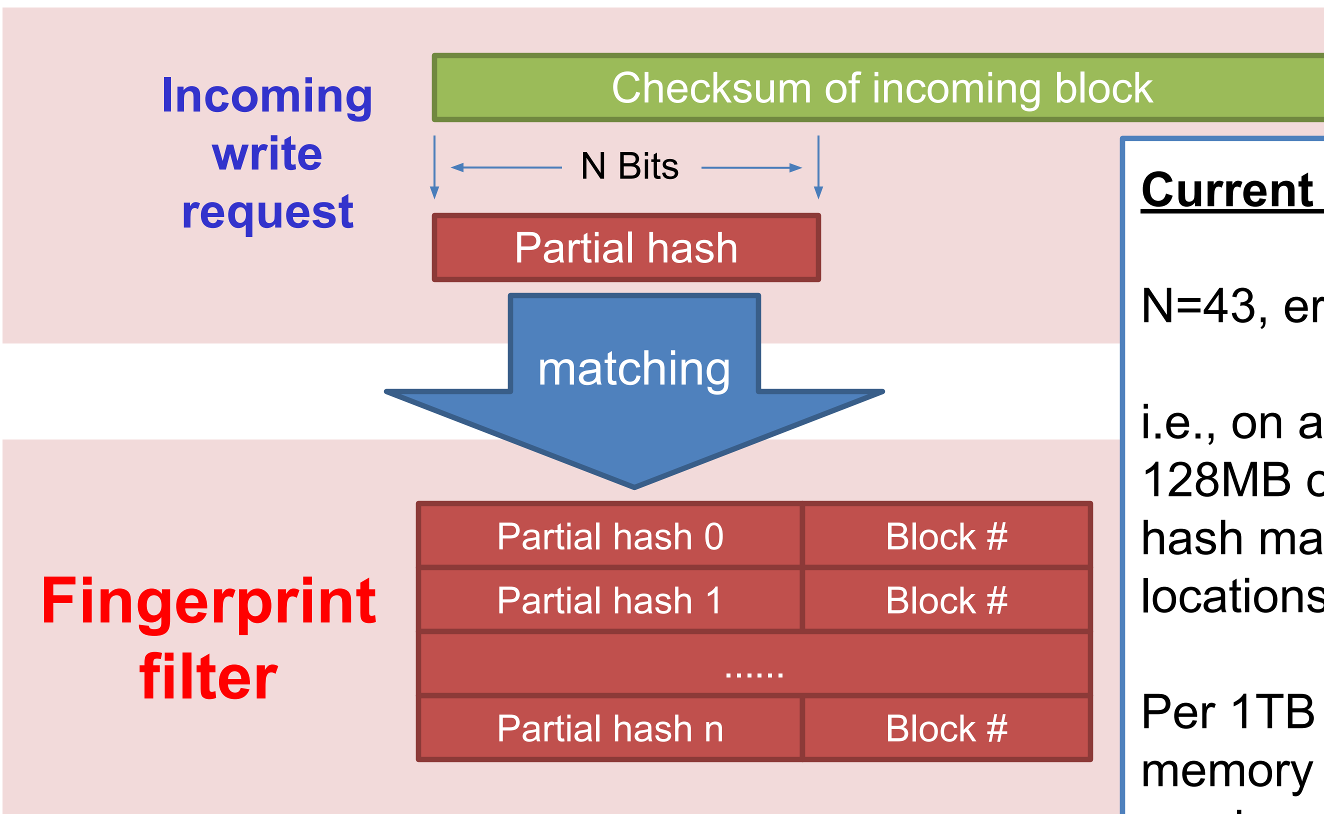


- Challenge: updating each data block and its index keeps the disk head moving, which hurts performance

Slide adapted from Patrick Lee

Example: LiveDFS Design

- Store partial fingerprints in memory
  - Infer if same block exists, and where it is “potentially” located



<u>Current Implementation</u>

N=43, error rate = 2^{-15}

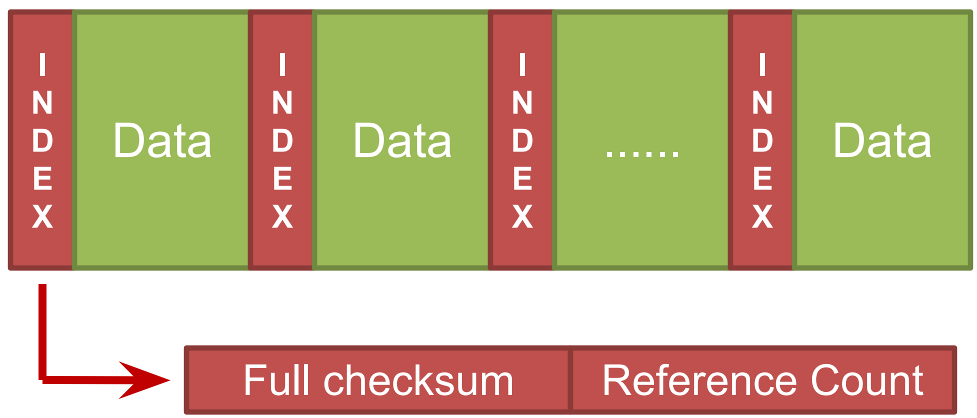
i.e., on average every 128MB of data, one partial hash mapped to two block locations.

Per 1TB of disk, reduce memory usage from previous 4GB to **1.6GB**

Slide adapted from Patrick Lee

Example: LiveDFS Design

- Store full fingerprints on disk, with **spatial locality**
  - Verify whether the same block is actually located



➢ Partition index structure according to block groups
  - Each block group has a **fingerprint store**
  - Each fingerprint store keeps fingerprints and reference counts for the respective data blocks in the same block group

➢ Writing with close proximity incurs minimal seeks

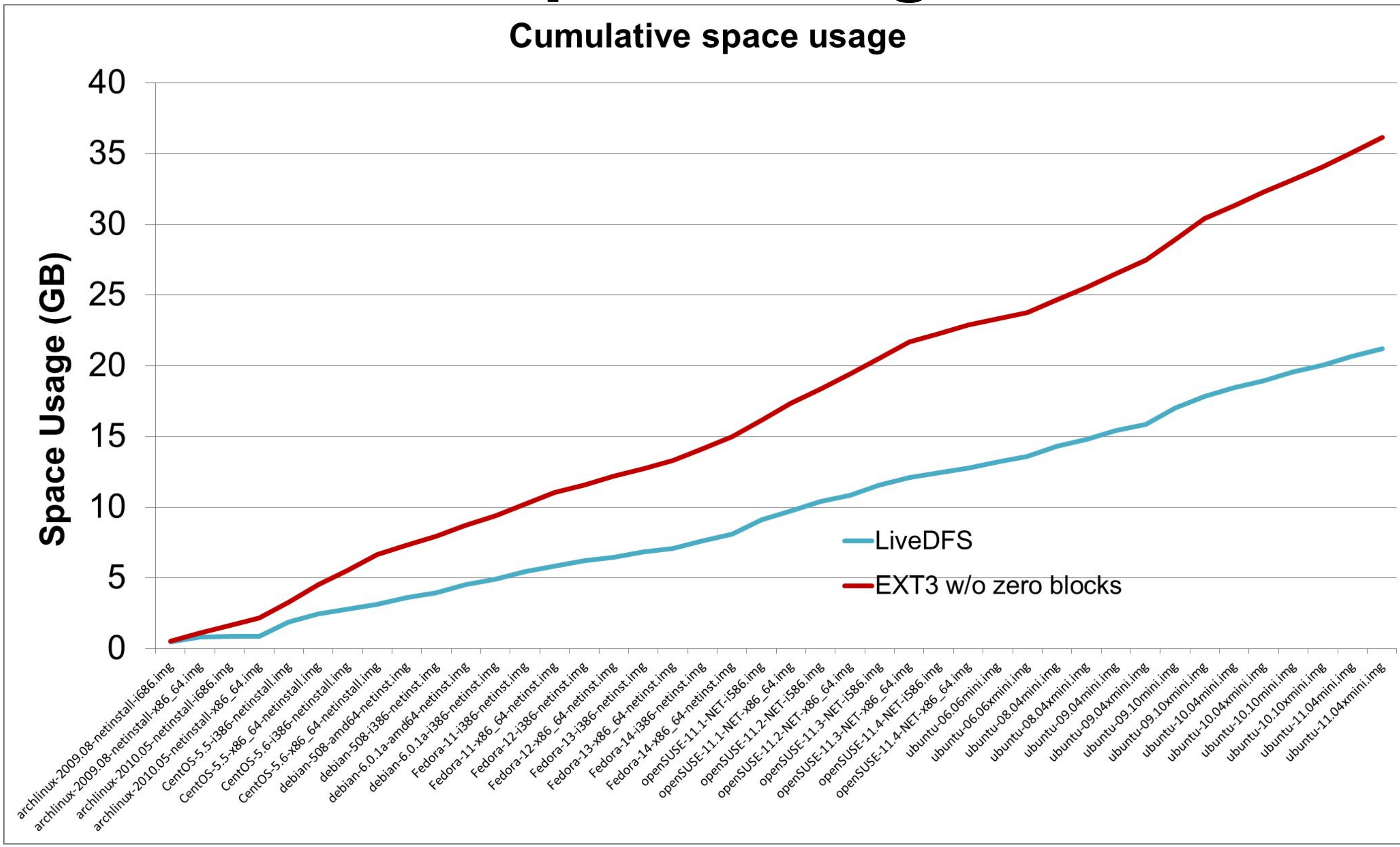
Slide adapted from Patrick Lee

Example: LiveDFS Design

- Take-away: LiveDFS arranges fingerprints in memory and on disk according to **underlying file system layout on disk**
- Other features:
  - Prefetching of fingerprint store:
    - load entire fingerprint store of same block group into page cache
    - subsequent writes updates fingerprint store directly in page cache.
  - Journaling:
    - follow Linux file system journaling design
    - enable crash recovery and enhance write performance by combining block writes in batch

Slide adapted from Patrick Lee

Space Usage

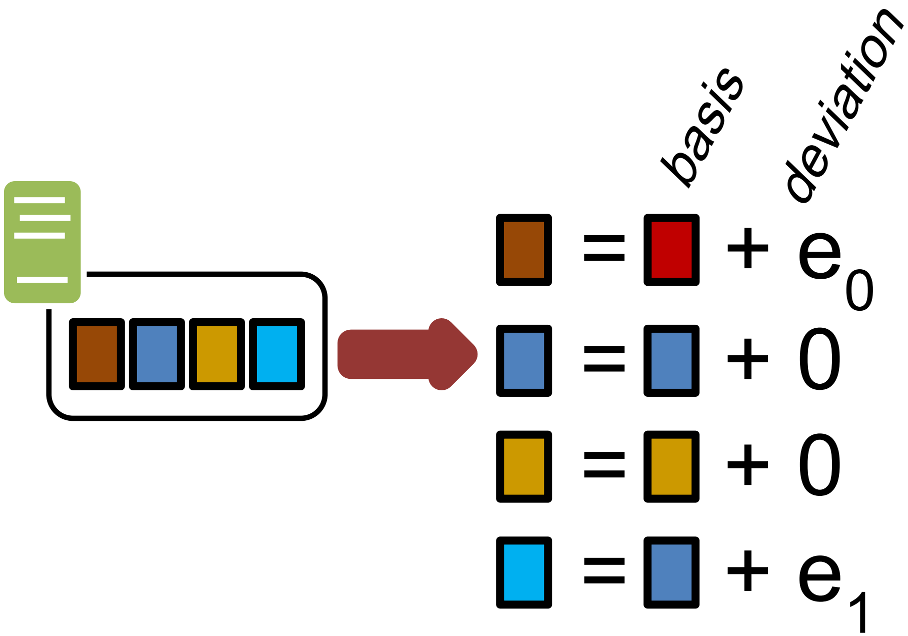


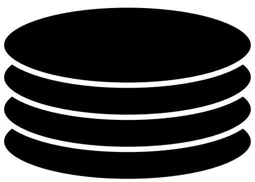
LiveDFS saves 40% storage over Ext3

Slide adapted from Patrick Lee

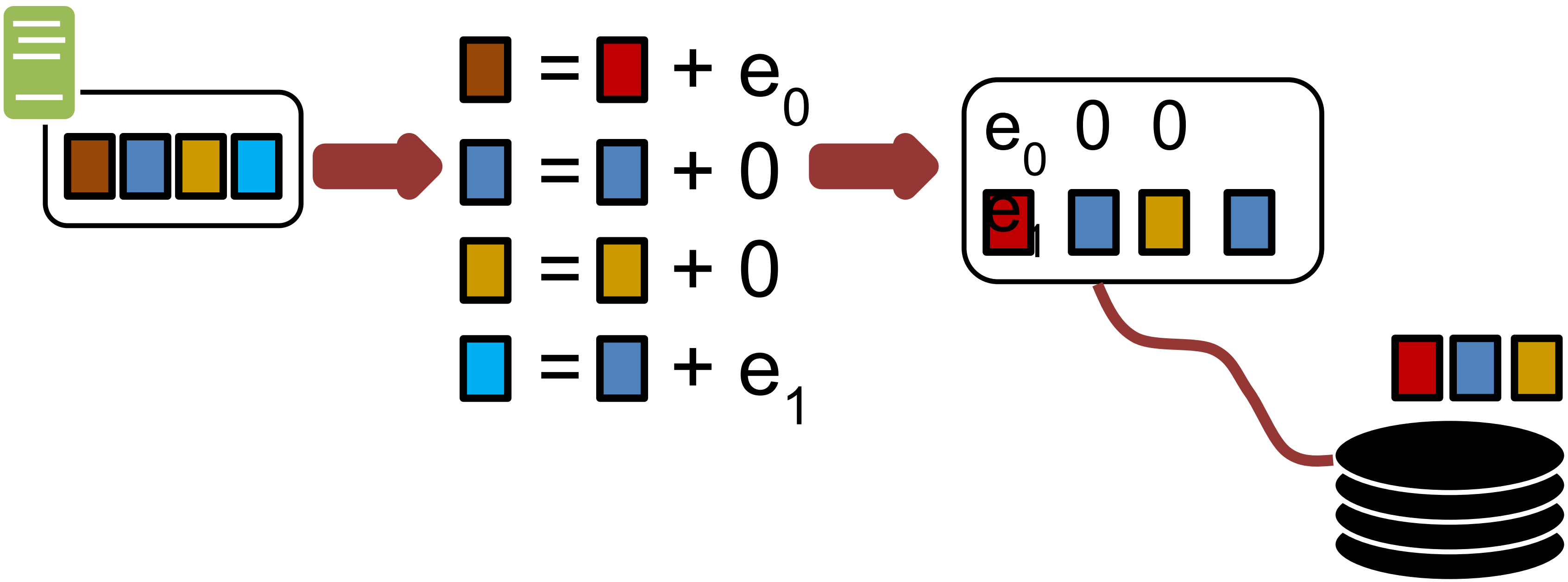
A Generalized View of Deduplication

Generalized Deduplication

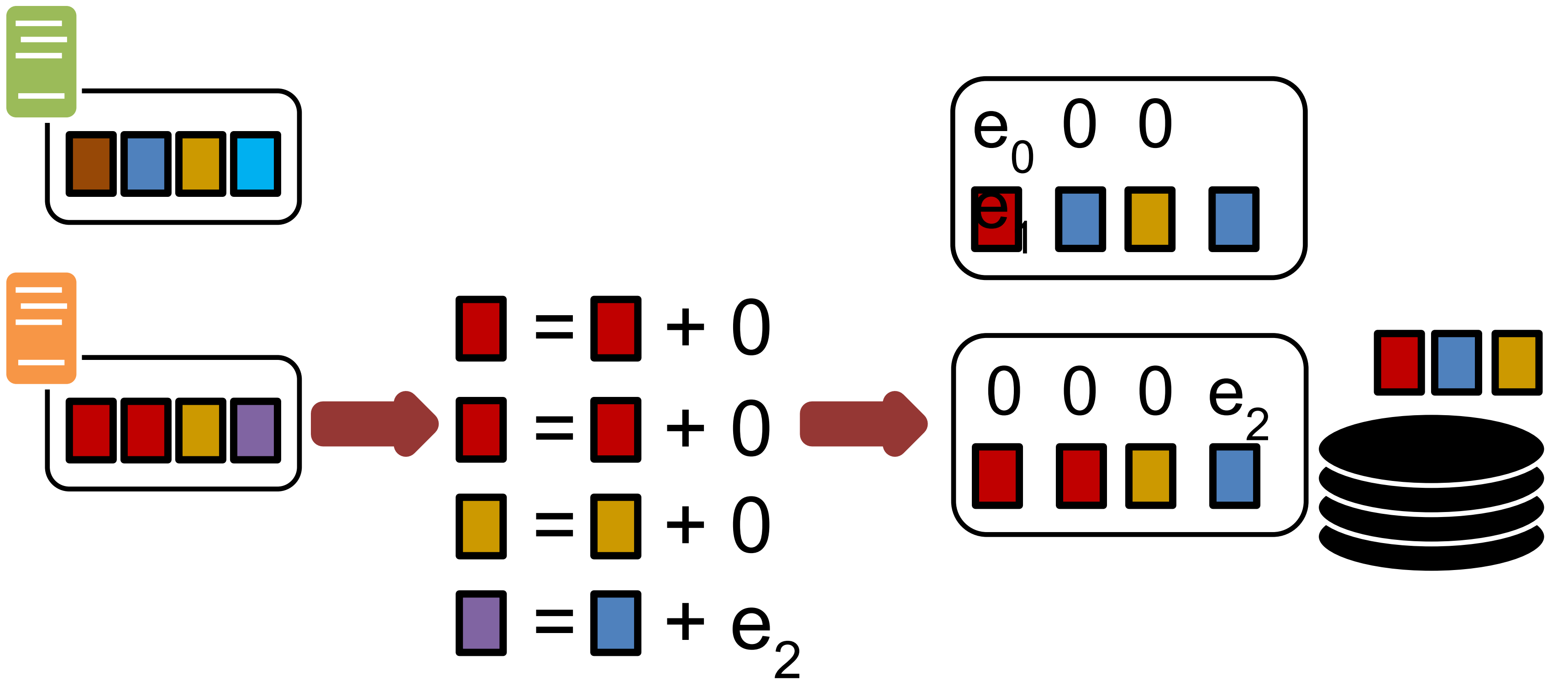




Generalized Deduplication

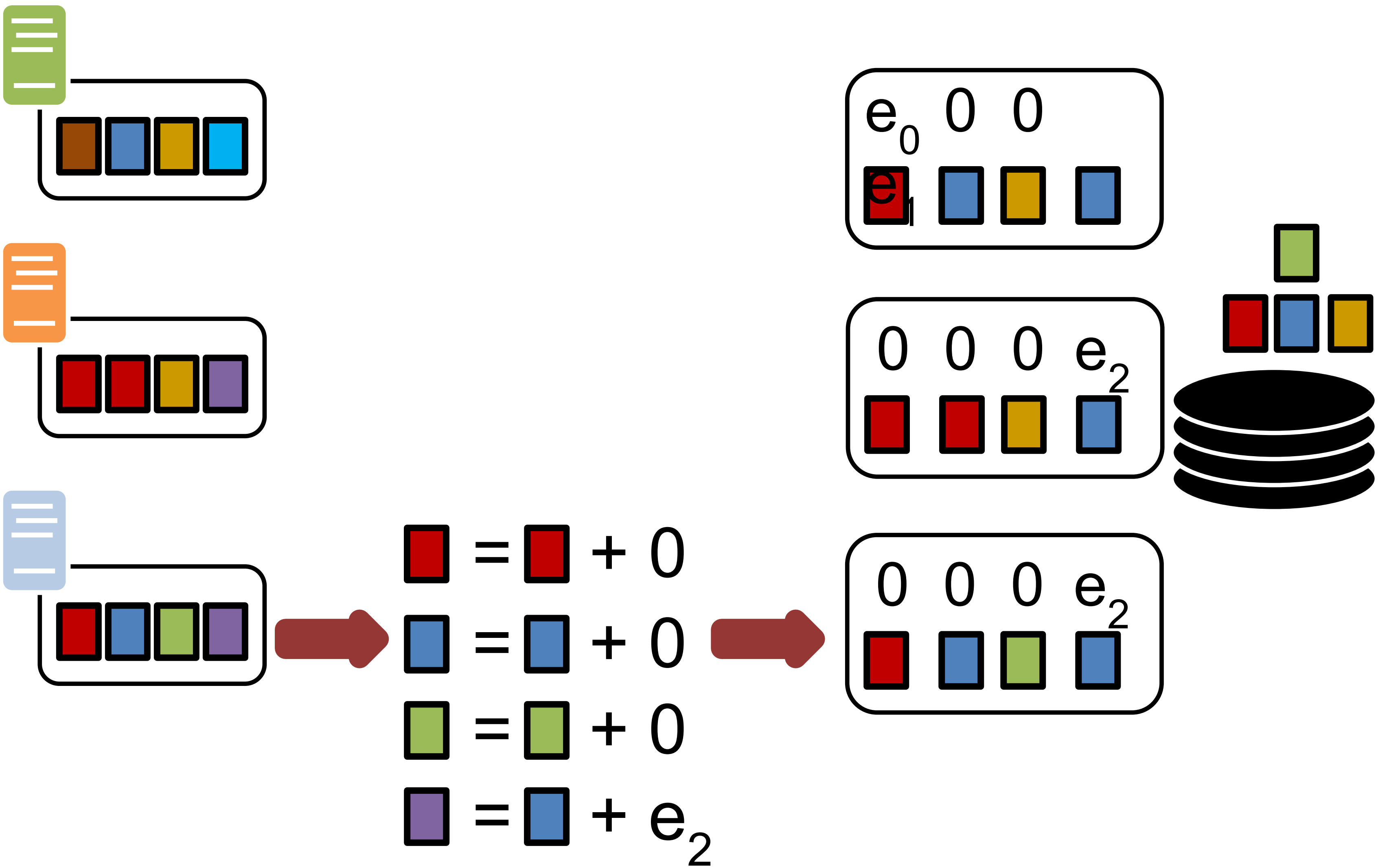


Generalized Deduplication



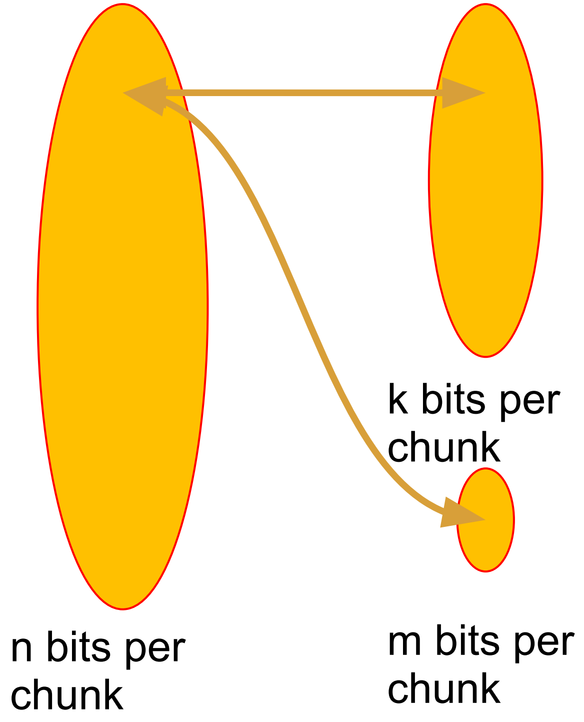
Generalized Deduplication

Compression gain is 12/4 = 3



HOW TO DO THE TRANSFORMATION?

- Promotes large amount of matches
- Small deviation (few bits)
- There are a number of options



Example of Transformation

Error Correcting Codes: case where code can correct errors

Data: 1 0 1 0 → Add redundancy: 1 0 1 0 1 1 0 → Channel with errors → 1 0 1 1 1 1 0 → Recovered data: 1 0 1 0

Example of Transformation

Error Correcting Codes: case where code can correct errors

Data: 1 0 1 0 → Add redundancy: 1 0 1 0 1 1 0 → Channel with errors → 1 0 1 1 1 1 0 → Recovered data: 1 0 1 0

In our case...

Data		Map (basis + deviation)
1 0 1 0 0 1 0		S = bit 4
1 0 1 1 1 1 0		S = bit 3
1 0 1 0 1 1 0	→ 1 0 1 0 +	S = no error
1 0 1 0 1 1 0		S = bit 2
0 0 1 0 1 1 0		S = bit 0

Example of Transformation

Error Correcting Codes: case where code can correct errors

Data: 1 0 1 0 → Add redundancy: 1 0 1 0 1 1 0 → Channel with errors → 1 0 1 1 1 1 0 → Recovered data: 1 0 1 0

In our case...

Data			Map (basis + deviation)
1 0 1 0 0 1 0			S = bit 4
1 0 1 1 1 1 0			S = bit 3
1 0 1 0 1 1 0	→	1 0 1 0 +	S = no error
1 0 1 0 1 1 0			S = bit 2
0 0 1 0 1 1 0			S = bit 0

Only stored once

Example of Transformation

Error Correcting Codes: case where code can correct errors

Data: 1 0 1 0 → Add redundancy: 1 0 1 0 1 1 0 → Channel with errors → 1 0 1 1 1 1 0 → Recovered data: 1 0 1 0

In our case...

Data			Map (basis + deviation)
1 0 1 0 0 1 0	→	1 0 1 0 +	S = bit 4
1 0 1 1 1 1 0	→	1 0 1 0 +	S = bit 3
1 0 1 0 1 1 0	→	1 0 1 0 +	S = no error
1 0 1 0 1 1 0	→	1 0 1 0 +	S = bit 2
0 0 1 0 1 1 0	→	1 0 1 0 +	S = bit 0

m bit for deviation

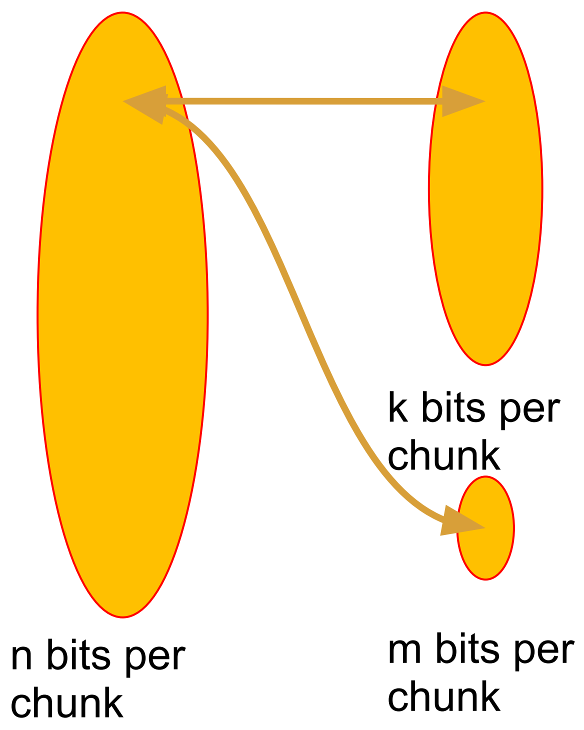
$n = 2^m - 1$ bits

$k = 2^m - m - 1$ bits

n + 1 sequences matched to 1 basis

HOW TO DO THE TRANSFORMATION?

- Promotes large amount of matches: n +1 matches
- Small deviation (few bits): m bits
- There are a number of options with better characteristics



If m = 15 bits

n ~ 4 KB

32768 matches

Toy Example

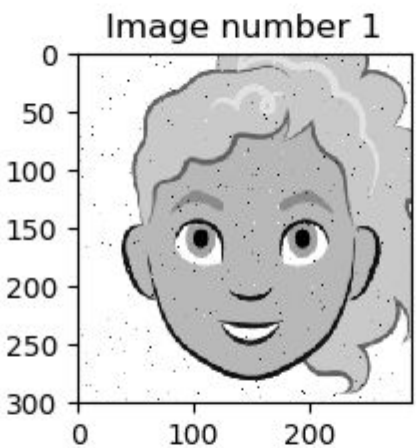
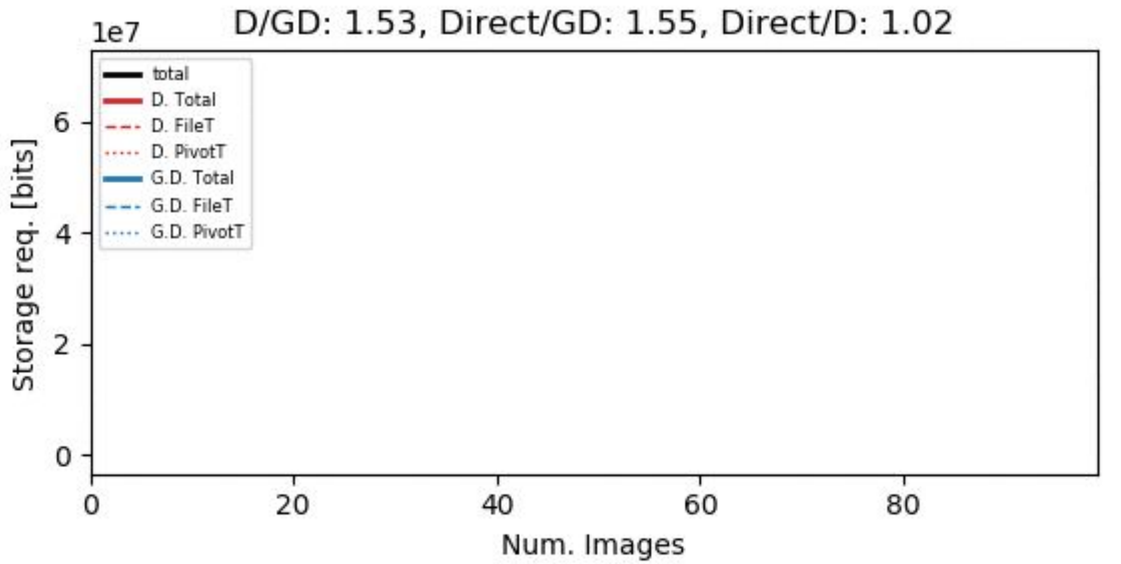
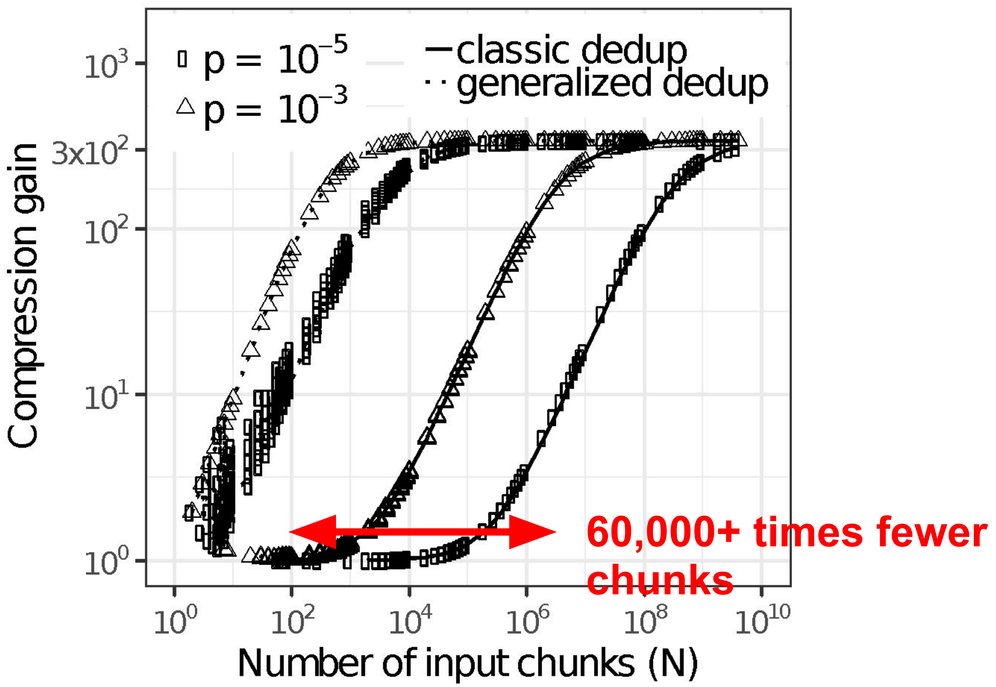


Image number 1
0
50
100
150
200
250
300
0
100
200

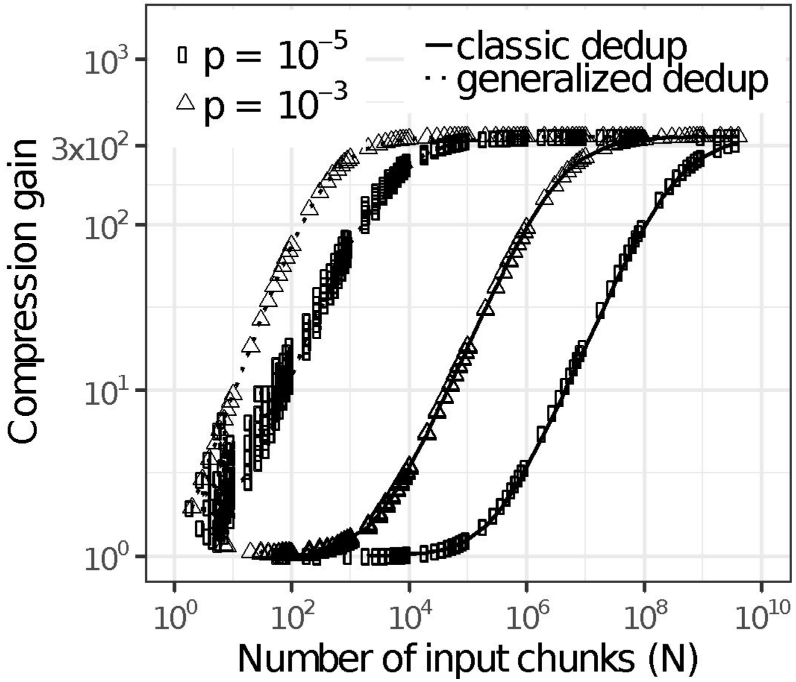


D/GD: 1.53, Direct/GD: 1.55, Direct/D: 1.02
1e7
total
D. Total
D. FileT
D. PivotT
G.D. Total
G.D. FileT
G.D. PivotT
Storage req. [bits]
0
2
4
6
0
20
40
60
80
Num. Images

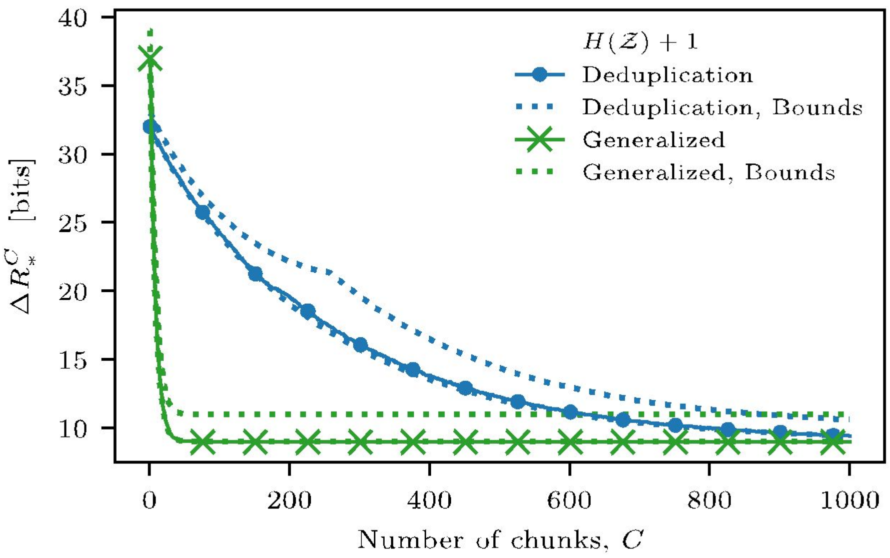
Compression Starts Earlier



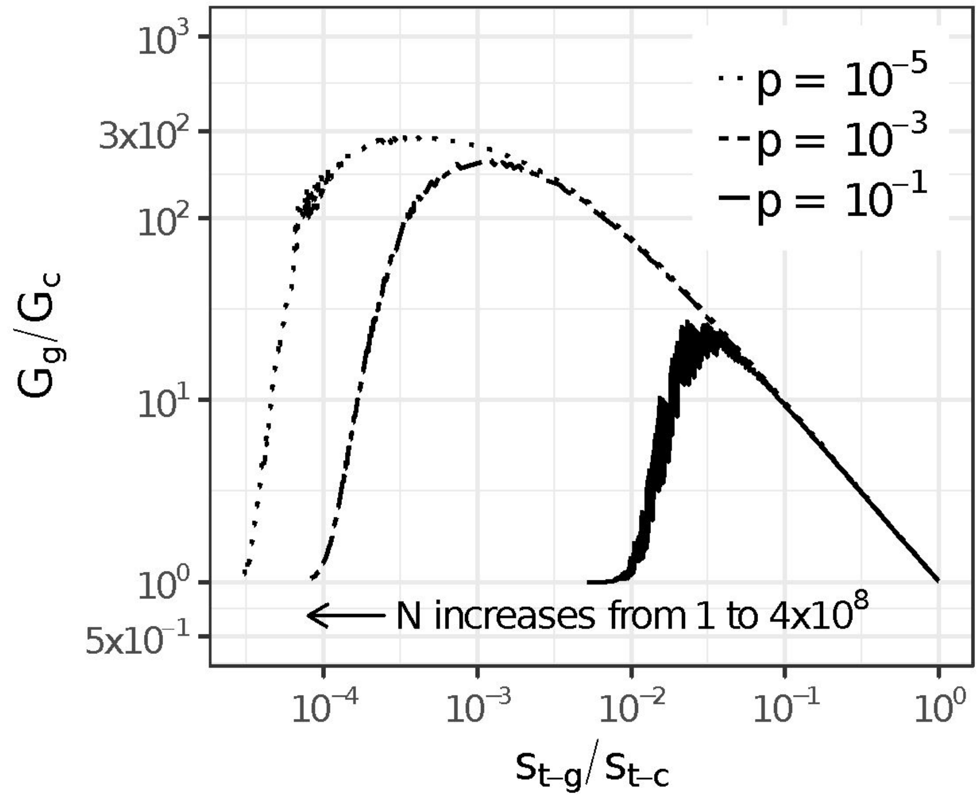
Compression Starts Earlier



Information Theory – proven this speed up



Need Smaller Registry

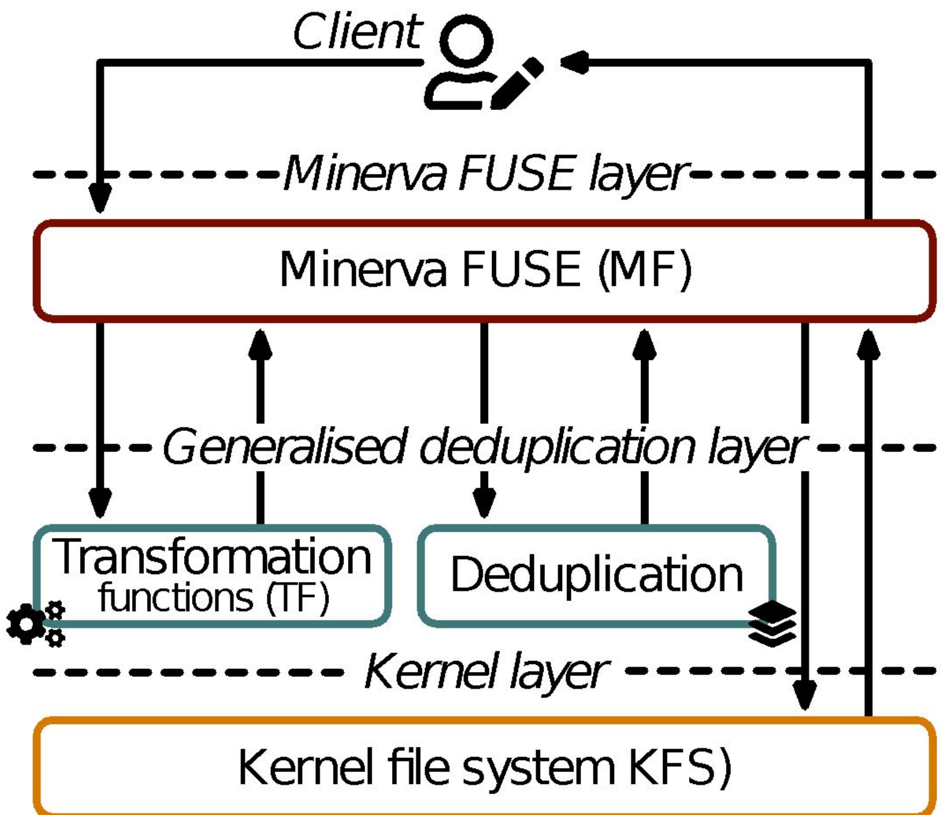


Size of the registry gen dedup vs dedup

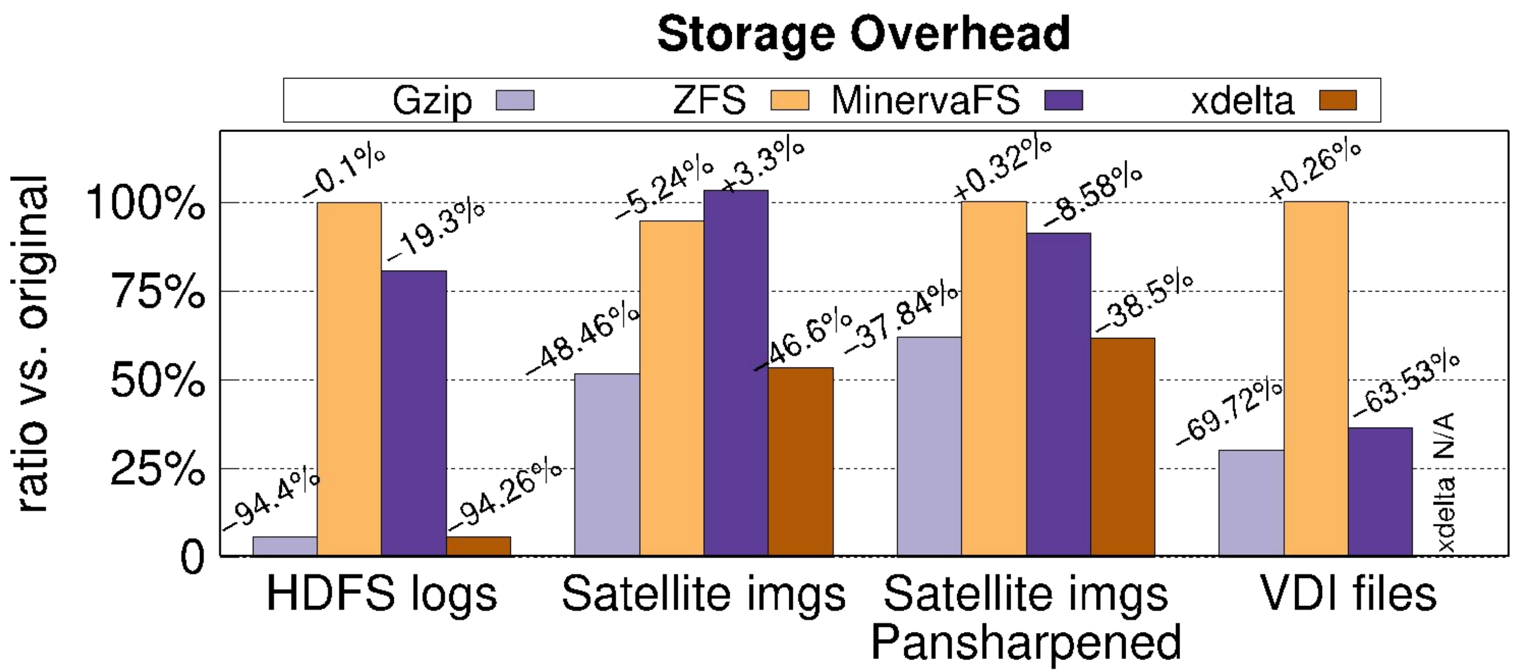
ZFS: rule of thumb 5GB of Memory for every 1TB of data

Can lower this requirement (e.g. 10 – 10000 times)

MINERVA-FS


Client
Minerva FUSE layer
Minerva FUSE (MF)
Generalised deduplication layer
Transformation
functions (TF)
Deduplication
Kernel layer
Kernel file system KFS)

MINERVA-FS



Also… much more compression than LiveDFS!

FUSE over EXT4

- ZFS: ~5GB of Memory for every 1TB of data

- MinervaFS: due to GD & implementation is an order of magnitude lower for the same chunk size