ECE566
Enterprise Storage Architecture

Spring 2024

Storage Efficiency

Tyler Bletsch

Duke University

Two views of file system usage

- User data view:
  - "How large are my files?" (bytes-used metric)
    or
    "How much capacity am I given?" (bytes-available metric)
  - **Bytes-used**: Total size = sum of all file sizes
  - **Bytes-available**: Total size = volume size or "quota"
  - Ignore file system overhead, metadata, etc.
  - In pay-per-byte storage (e.g. cloud), you charge based bytes-used
  - In pay-for-container storage (e.g. a classic webhost), you charge based on bytes-available

- Stored data view:
  - How much actual disk space is used to hold the data?
  - Total usage is a separate measurement from file size or available space!
    - "ls –l" vs. "du"
  - Includes file system overhead and metadata
  - Can be reduced with *trickery*
  - If you're the service provider, you buy enough disks for this value

Storage efficiency

- $\text{StorageEfficiency} = \frac{UserData}{StoredData}$

- Without storage efficiency features, this value is < 1.0. Why?
  - File system metadata (inodes, superblocks, indirect blocks, etc.)
  - Internal fragmentation (on a file system with 4kB blocks, a 8193 byte file uses three data blocks; the last block is almost entirely unused)
  - RAID overhead (e.g. a 4-disk RAID5 has 25% overhead)

- Can we add features to storage system to go above 1.0?
  - Yes (otherwise I wouldn't have a slide deck called "storage efficiency")

Why improve storage efficiency?

- Why do we want to improve storage efficiency?
  - Buy fewer disks! Reduce costs!
  - If we're a service provider, you charge based on *user data*, but your costs are based on *stored data*.
    Result: More efficiency = more profit
    (and the customer never has to know)
- Note: all these techniques **depend on workload**

Techniques to improve storage efficiency

More efficient RAID

Snapshot/clone

Zero-block elimination

Thin provisioning

Deduplication

Compression

"Compaction" (partial zero block elimination)

RAID efficiency

- What's the overhead of a 4-disk RAID5?
  - 1/4 = 25%
- How to improve?
  - More disks in the RAID
- What's the overhead of a 20-disk RAID5?
  - 1/20 = 5%
- Problem with this?
  - Double disk failure very likely for such a large RAID
- How to fix?
  - More redundancy, e.g. RAID-6
    (Odds of triple disk failure are << odds of double disk failure, because we're ANDing unlikely events over a small timespan)
- What's the overhead of a 20-disk RAID6?
  - 2/20 = 10%
- **Result: Large arrays can achieve higher efficiency than small arrays**

Techniques to improve storage efficiency

- More efficient RAID
- **Snapshot/clone**
- Zero-block elimination
- Thin provisioning
- Deduplication
- Compression
- "Compaction" (partial zero block elimination)

Snapshots and clones

- This one is simple.

- If you want a copy of some data, and you don't need to write to the copy: **snapshot**.
  - Example: in-place backups to restore after accidental deletion, corruption, etc.

- If you want a copy of some data, and you do need to write to the copy: **clone**.
  - Example: copy of source code tree to do a test build against

We covered snapshots and clones in the last lecture – review if you missed it!

Techniques to improve storage efficiency

- More efficient RAID
- Snapshot/clone
- **Zero-block elimination**
- Thin provisioning
- Deduplication
- Compression
- "Compaction" (partial zero block elimination)

Zero block elimination

- This one is also simple.
- If the user writes a block of all zeroes, just note this in metadata; don't allocate any data blocks
- Why would the user do that?
  - Initializing storage for random writes (e.g. databases, BitTorrent)
  - Sparse on-disk data structures (e.g. large matrices, big data)
  - A "secure erase": overwrite data blocks to prevent recovery*

* Note that this form of secure erase only works if you're actually overwriting blocks in-place. We've learned that this isn't the case in log-structured and data-journaled file systems as well as inside SSDs. Secure data destruction is something we'll discuss when we get to security...

Techniques to improve storage efficiency

- More efficient RAID
- Snapshot/clone
- Zero-block elimination
- **Thin provisioning**
- Deduplication
- Compression
- "Compaction" (partial zero block elimination)

Thin provisioning

- Technique to improve efficiency for the bytes-available metric
- Based on insight in how people size storage requirements

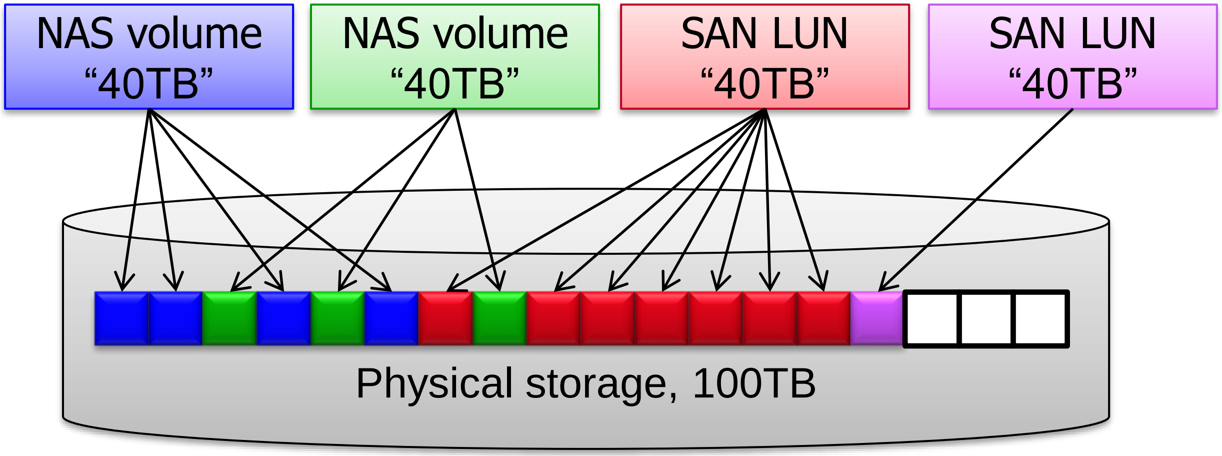
- System administrator:
  - “I need storage for this app. I don’t know exactly how much it needs.”
  - “If I guess too low, it runs out of storage and fails, and I get yelled at.”
  - “If I guess too high, it works and has room for the future.”
  - Conclusion: Always guess high.

Thin provisioning

- Storage provider:
  - “Four sysadmins need storage, each says they need 40 TB.”
  - “I know they’re all over-estimating their needs.”
  - “Therefore, the odds that *all* of them need *all* their storage is very low.”
  - “I can’t tell them I think they’re lying and give them less, or they’ll yell at me.”
  - “Therefore, each admin must *think* they have 40TB to use”
  - “I don’t want to pay for 4*40=160TB of storage because I know most of it will remain unused.”
  - **“I will <u>pool</u> a lesser amount of storage together, and everyone can pull from the same pool (<u>thin provisioning</u>)”**

Thin provisioning

- Result:
  - Buy 100TB of raw storage
  - For each sysadmin, make a 40TB file system (NAS) or LUN (SAN)
  - When used, all four containers use blocks from the 100TB pool

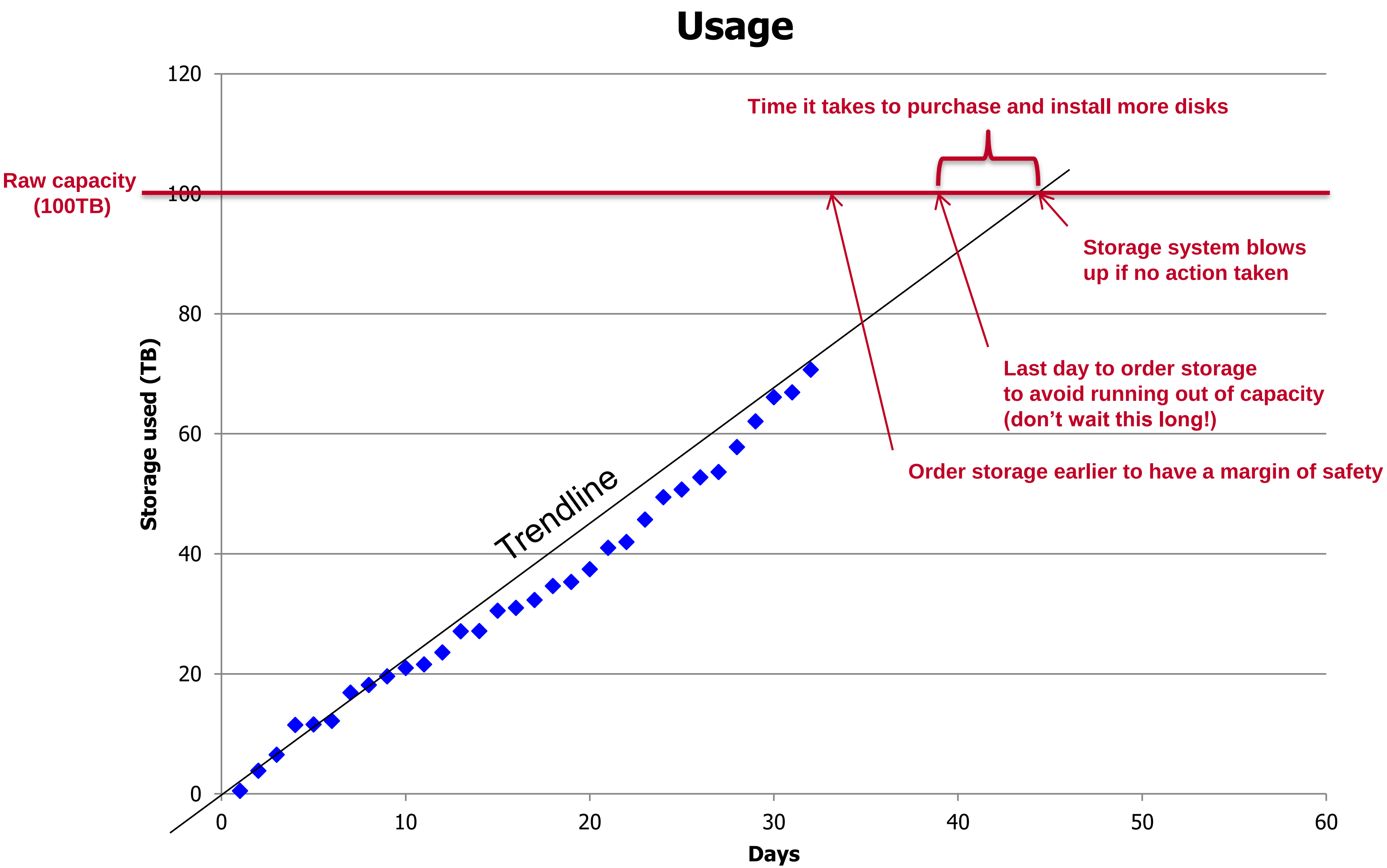


Managing thin provisioning

- Storage is "**over-subscribed**" (more allocated than available)
  - Need to monitor usage and add capacity ahead of running out

- Administrator can set their *risk level*:
  - More over-subscribed = cheaper, but more risk of running out if a sudden burst in usage happens
  - Less over-subscribed = more expensive, less risk

Managing thin provisioning



Reservations

- Per-user guarantees: “reservations”
  - Can set controller to guarantee a certain capacity per user
  - Reservations must add up to less than total capacity

- Example: Every user guaranteed 100/4=25TB
  - Limits damage if capacity runs out

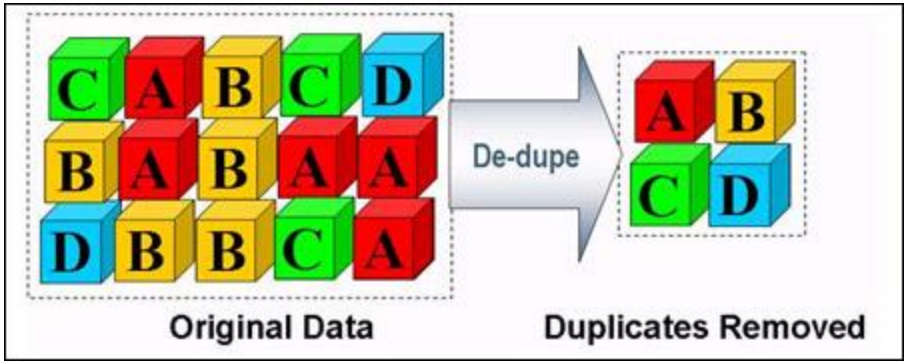
- Example: Priority app guaranteed 40TB,
  rest have no reservation
  - Priority app will ALWAYS get its full capacity, even if system otherwise fills up

Techniques to improve storage efficiency

- More efficient RAID
- Snapshot/clone
- Zero-block elimination
- Thin provisioning
- **Deduplication**
- Compression
- "Compaction" (partial zero block elimination)

Deduplication simplified

- Basic concept:



- Split the file in to chunks
- Hash each chunk with a big hash
- If hashes match, data matches:
  - Replace this with a reference to the matching data
- Else:
  - It's new data, store it.

Figure from http://www.eweek.com/c/a/Data-Storage/How-to-Leverage-Data-Deduplication-to-Green-Your-Data-Center/

Common deduplication data structures

- A simplistic model of deduplication:
  - Metadata:
    - Directory structure, permissions, size, date, etc.
    - Each file's contents are stored as a **list of hashes**
  - Data pool:
    - A flat table of hashes and the data they belong to
    - Must keep a reference count to know when to free an entry

^ A perfectly fine way to make a simple dedupe system in FUSE

- But now we know about filesystems and can be more clever:
  - Rather than files being a list of hashes, a deduplicating *file system* can use the inode's usual block pointers!
    - Difference: multiple block pointers can point to the same block
    - Blocks have reference counts
    - A table mapping *Block hash* to *block number* stored on disk (and cached in memory as hash table)

Inline vs. post-process

- Big design question: **inline** vs **post-process**
- Inline:
  - When a write occurs, determine the resulting block hash and deduplicate at that time.

  \+ File system is always fully deduplicated

  \+ Simple implementation

  – Writes are slowed by additional computation
- Post-process
  - Write committed normally, background daemon periodically hashes unhashed blocks to deduplicate them.

  \+ Low overhead to the write itself

  – More overall writes to disk (write + read + possible change)

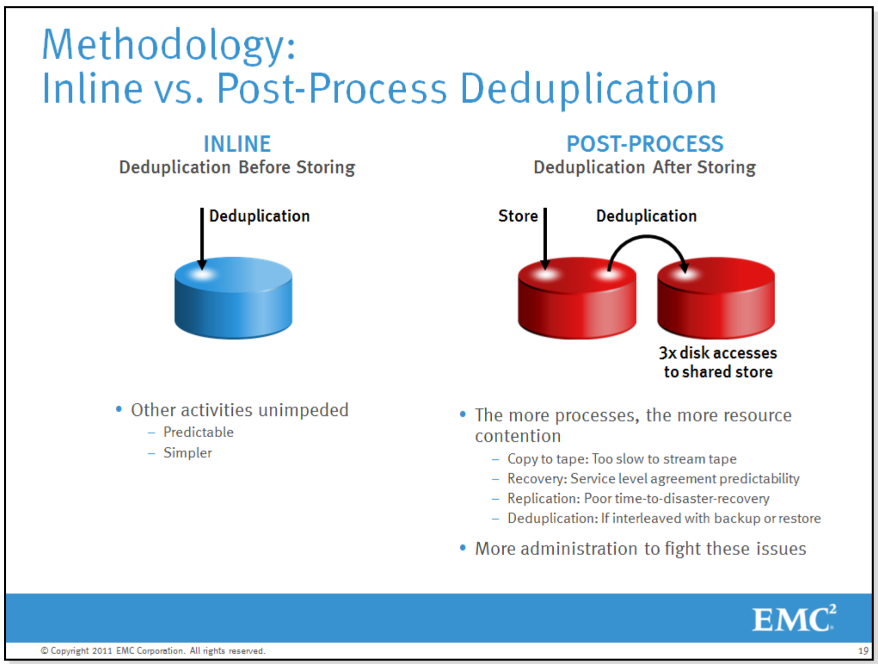
  – Disk not fully deduplicated until later (increased average space usage)

  – Need to synchronize user I/Os versus background daemon I/Os for consistency

LOL industry

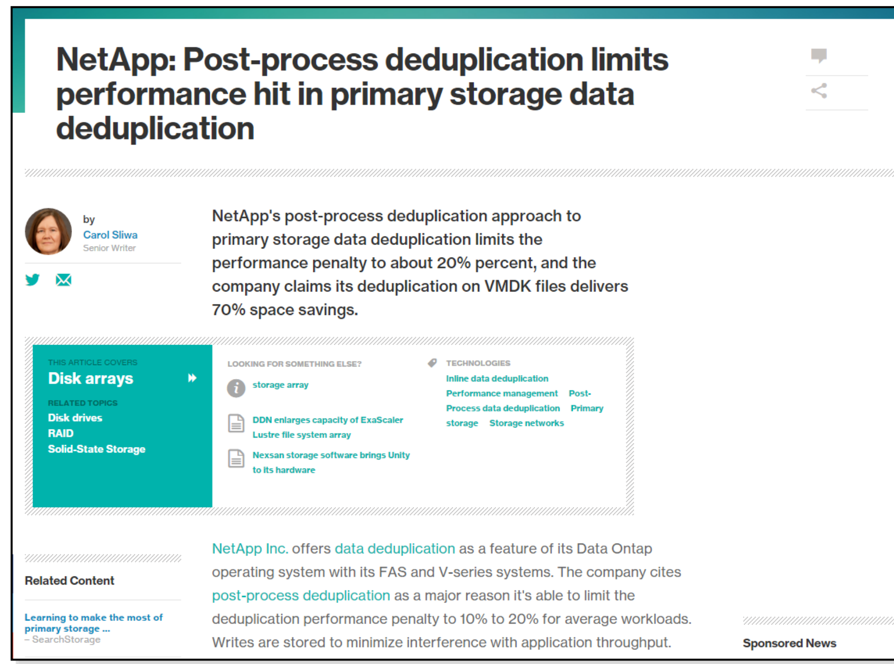
- Choice between inline and post-process is tradeoff, no one right answer.
- That doesn't stop industry vendors from using it to spread FUD (Fear, Uncertainty, and Doubt).

EMC product slide



“Post-process dedupe does more accesses, so it must be slow!”

NetApp-friendly article



NetApp: Post-process deduplication limits performance hit in primary storage data deduplication

by Carol Sliwa
Senior Writer

NetApp's post-process deduplication approach to primary storage data deduplication limits the performance penalty to about 20% percent, and the company claims its deduplication on VMDK files delivers 70% space savings.

THIS ARTICLE COVERS
Disk arrays
RELATED TOPICS
Disk drives
RAID
Solid-State Storage

LOOKING FOR SOMETHING ELSE?
storage array
DDN enlarges capacity of ExaScaler Lustre file system array
Nexsan storage software brings Unity to its hardware

TECHNOLOGIES
Inline data deduplication
Performance management
Post-Process data deduplication
Primary storage
Storage networks

NetApp Inc. offers data deduplication as a feature of its Data Ontap operating system with its FAS and V-series systems. The company cites post-process deduplication as a major reason it's able to limit the deduplication performance penalty to 10% to 20% for average workloads. Writes are stored to minimize interference with application throughput.

Related Content
Learning to make the most of primary storage ...
– SearchStorage

Sponsored News

“Post-process dedupe makes writes faster, anything that lacks it must be slow!”

Fixed vs. variable-sized blocks

- Insertion/deletion: A common modification.

MY TEXT FILE
This is my text file.
It contains bytes.
I like my text file.
It is a very good text file.
01234567890123456789...

Copy+modify →

MY TEXT FILE
By Tyler Bletsch
This is my text file.
~~It contains bytes.~~
I like my text file.
It is a very good text file.
01234567890123456789...

(Side note: you can't literally "insert" or "delete" stuff to a file and have it shift like this – your text editor reads the whole file, you change it in RAM, then you save the whole file. The actual file system only supports in-place changes; no shifts.)

Fixed vs. variable-sized blocks

- Insertion/deletion: A common modification.

MY TEXT FILE
This is my text file.
It contains bytes.
I like my text file.
It is a very good text file.
01234567890123456789...

Copy+modify →

MY TEXT FILE
By Tyler Bletsch
This is my text file.
~~It contains bytes.~~
I like my text file.
It is a very good text file.
01234567890123456789...

- With 8-byte fixed-sized blocks:

MY TEXT	FILE|Thi	s is my	text fil	e.|It co	ntains b	ytes.|I	like my	text fil	e.|It is	a very	good tex	t file.|	01234567	89012345	6789...
MY TEXT	FILE|By	Tyler Bl	etsch|Th	is is my	text fi	le.|I li	ke my te	xt file.	|It is a	very go	od text	file.|01	23456789	01234567	89...

- All blocks past the change differ!
- Bad, because this is a common case

Variable-sized blocks

- What if, instead of fixed-sized blocks, we made blocks divided based on the *content* of the file?
  - Resulting blocks may be of variable size
- Naive rule: divide a block whenever there's a space

<table><tr><td>MY TEXT </td><td>FILE|This </td><td>is </td><td>my </td><td>text </td><td>file.|It </td><td>contains </td><td>bytes.|I </td><td>like </td><td>my </td><td>text </td><td>file.|It </td><td>is </td><td>a </td><td>very </td><td>good </td><td>text </td><td>file.|0123456789012345678 9...</td></tr></table>

<table><tr><td>MY TEXT </td><td>FILE|By </td><td>Tyler </td><td>Bletsch|This </td><td>is </td><td>my </td><td>text </td><td>file.|I </td><td>like </td><td>my </td><td>text </td><td>file.|It </td><td>is </td><td>a </td><td>very </td><td>good </td><td>text </td><td>file.|01234567890123456789...</td></tr></table>

- Way more blocks match! Mismatches only near the insertion/deletion, which is what we want!

- Could there be any issue with the "divide on space" rule?
  - Yes, obviously. Blocks too small (text file), or blocks too large (binary file).
  - Need a content-based dividing rule that won't go crazy on specific data

Rabin-Karp Fingerprinting

- Hash every offset with a "sliding window":

- Declare a block boundary every time the hash value equals a "special constant" (e.g. zero)
- Boundaries will depend on data, but in a "deterministically random" way (i.e. the byte sequences that cause division won't be "special" in any way)
- Parameters:
  - **Hash size:** On average, block size will be 2^{hash_bits}; can select hash size to give desired average block size
  - **Window size:** How much data to consider to make boundaries. The number of byte sequences that result in a boundary is, on average, $2^{window\_bits - hash\_bits}$

Rabin-Karp Fingerprinting

- Efficiency: all those hashes must be expensive, right?
  - Given windows size ***m*** and file size ***n***, don't you need ***n*m*** hash updates?
  - Not if we use *trickery*: **rolling hash**

```
for i from 1 to n-m+1
  h = hash(s[i+1 .. i+m])
```

```
h = hash(s[1 to m])
for i from 2 to n-m+1
  h = h – s[i-1]
  h = h + s[i+m]
```

"–" means "computationally remove from the hash"
"+" means "computationally add to the hash"

- Now just ***n-m*** "hash updates"

Techniques to improve storage efficiency

- More efficient RAID
- Snapshot/clone
- Zero-block elimination
- Thin provisioning
- Deduplication
- **Compression**
- "Compaction" (partial zero block elimination)

Compression

- Represent the data with fewer bits.
- Fundamental concept: Identify patterns which can be abbreviated
  - Many, many, many algorithms out there – beyond scope of course
    - Lempel-Ziv and descendants (deflate, PKZIP, GZIP, etc.)
    - Probabilistic models
    - Grammar-based codes

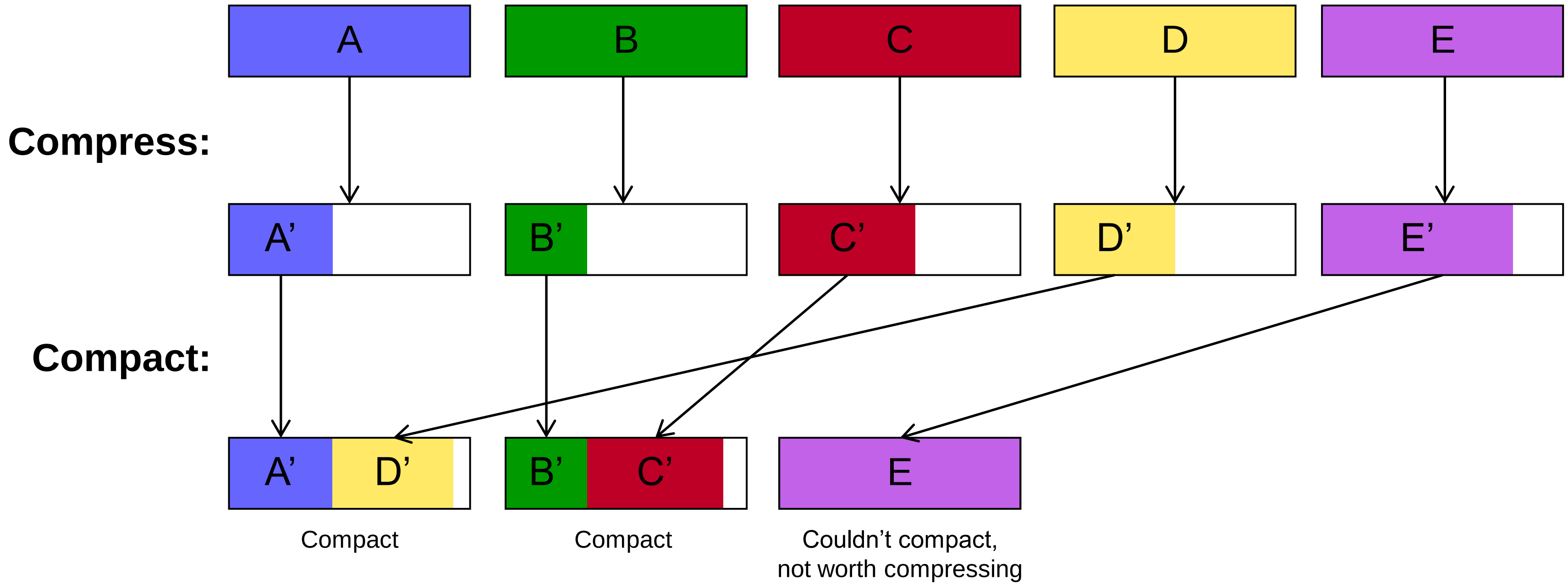
- A truth we've seen a hundred times: this is a tradeoff
  - Time vs. storage

Challenge when applied to disk storage

- Still need to **seek**: if we compress a file end-to-end, we don't know where to go to find a given offset
  - Solutions:
    - Compress blocks rather than files ← Upcoming example
    - Store some kind of index to allow seeking in compressed data (e.g., an uncompressed offset -> compressed offset table)
    - Probably other ideas...
- **Block storage:** If we compress a data block, but we still store it in a disk block, we didn't save anything...
  - Solutions:
    - Pack multiple compressed blocks into one real block ← Upcoming example
    - Consider larger "chunks" and compress them down to fewer blocks
    - Probably other ideas...

Compression with compaction

- Compression with simple compaction



- Data block pointers are now {block_num, offset, length}

Techniques to improve storage efficiency

- More efficient RAID
- Snapshot/clone
- Zero-block elimination
- Thin provisioning
- Deduplication
- Compression
- **“Compaction” (partial zero block elimination)**

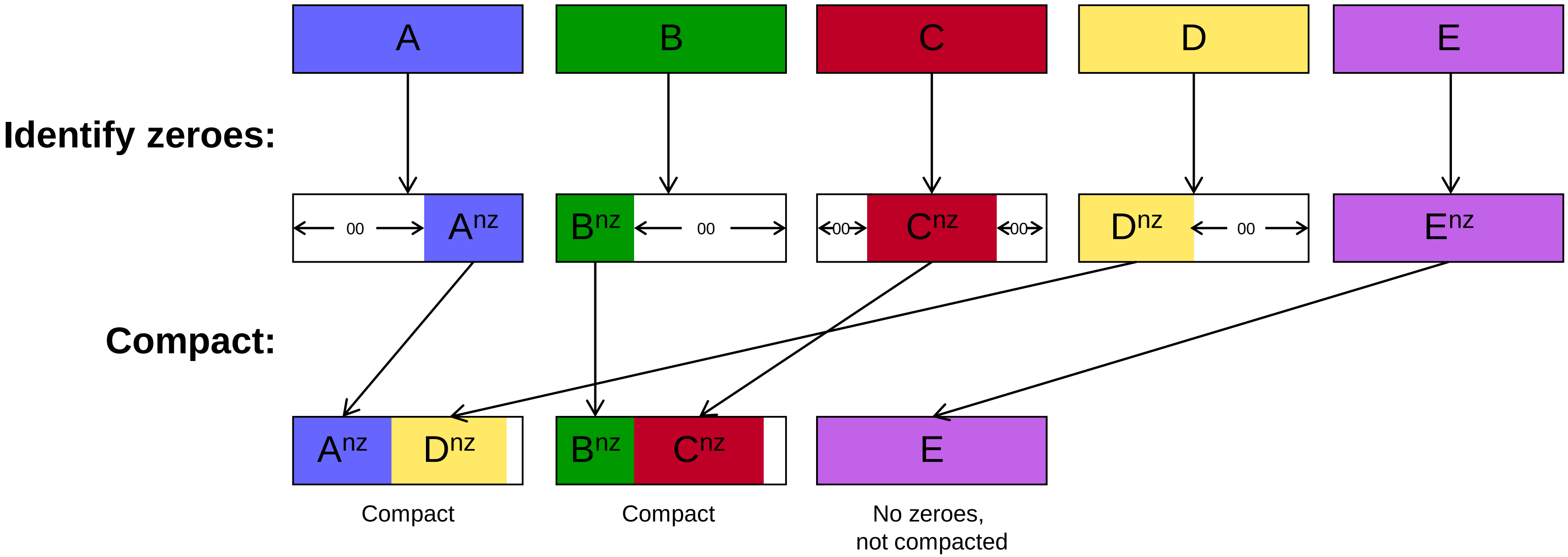
Compaction

- Remember how we were able to ignore zero-blocks?
- What if a block is partially zeroed...can we take advantage of that?

- Basically same as the compaction step we saw in compression, except just for zero data
  - Simple idea, only worth doing if the workload has sparse files
  - Sparse files: files that are mostly “empty” (zeroes), e.g. scientific sparse matrix data

Compression with compaction

- Compression with simple compaction



- Data block pointers are now {block_num, offset, length} (again)

Conclusion

- There are many ways to reduce physical storage needs
- By doing many at once, can often cut storage needs dramatically (50%+)
- **Depends strongly on workload:**

More efficient RAID	• Need large array
Snapshot/clone	• Only if you need copies
Zero-block elimination	• Only for sparse data
Thin provisioning	• Only if average utilization << peak utilization
Deduplication	• Only if data has duplication
Compression	• Only if data is compressible
"Compaction" (partial zero block elimination)	• Only for sparse data

- Example: For a long time, NetApp ran a promotion called the "NetApp 50% Virtualization Guarantee": if you're storing VMs on NetApp, they guaranteed you'd need 50% less disk capacity vs. competitors. They pay you otherwise.
  - Note: NetApp arrays are large, VMs are often cloned, virtual disks are sparse, have low average utilization, lots of duplication, and are often compressible.
  - Result: They very rarely had to pay out.