Data Storage Formats

Instructor: Matei Zaharia

cs245.stanford.edu

Outline

Storage devices wrap-up

Record encoding

Collection storage

C-Store paper

Indexes

Outline

Storage devices wrap-up

Record encoding

Collection storage

C-Store paper

Indexes

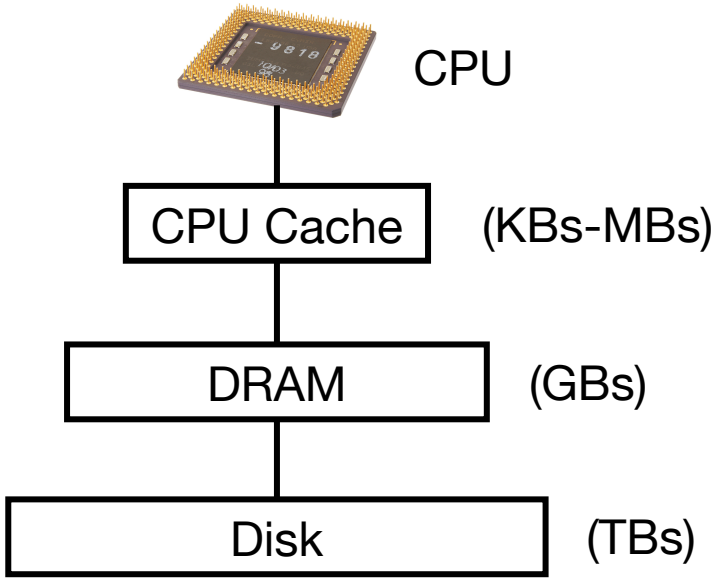
From Last Time

Storage devices offer various tradeoffs in terms of latency, throughput and cost

In all devices, random $\ll$ sequential access

Storage Hierarchies

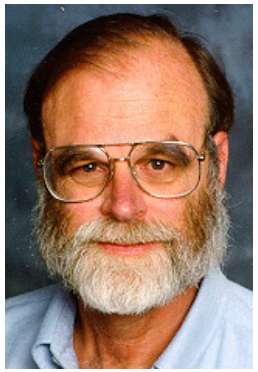
Typically **cache** frequently accessed data on faster storage to improve performance



Sizing Storage Tiers

How much high-tier storage should we have?

Can determine based on workload & cost



"The 5 Minute Rule for Trading Memory Accesses for Disc Accesses"
Jim Gray & Franco Putzolu
May 1985

The Five Minute Rule

Say a page is accessed every X seconds

Assume a disk costs D dollars and can do I operations/sec; cost of keeping this page on disk is

$$C_{disk} = C_{iop} / X = D / (I X)$$

Assume 1 MB of RAM costs M dollars and holds P pages; then the cost of keeping it in DRAM is:

$$C_{mem} = M / P$$

Five Minute Rule

This tells us that the page is worth caching when $C_{mem} < C_{disk}$, i.e.

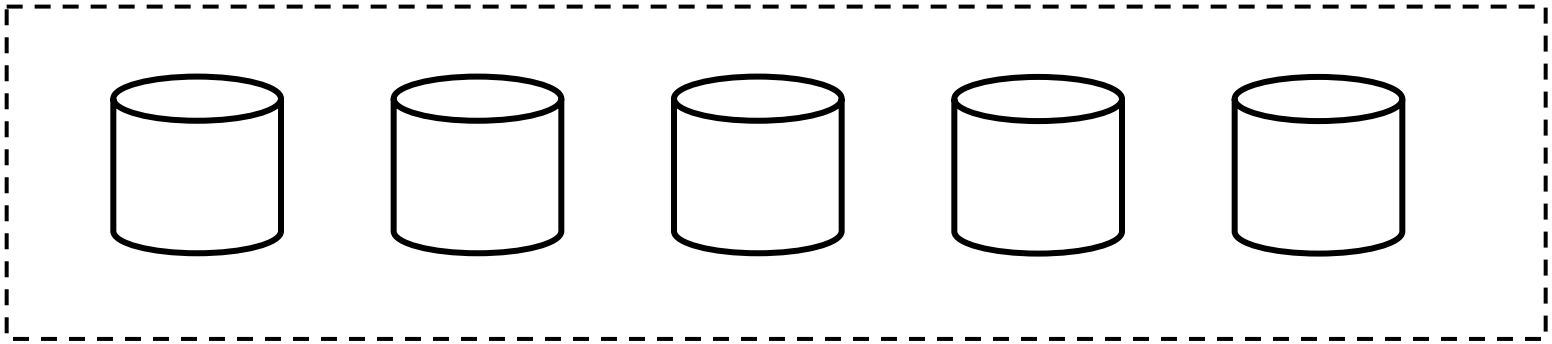
$$X < \frac{PagesPerMBofDRAM}{AccessesPerSecondPerDisk} \times \frac{PricePerDiskDrive}{PricePerMBofDRAM}$$

Tier	1987	1997	2007	2017
DRAM–HDD	5m	5m	1.5h	4h
DRAM–SSD	-	-	15m	7m (r) / 24m (w)
SSD–HDD	-	-	2.25h	1d

Source: The Five-minute Rule Thirty Years Later and its Impact on the Storage Hierarchy

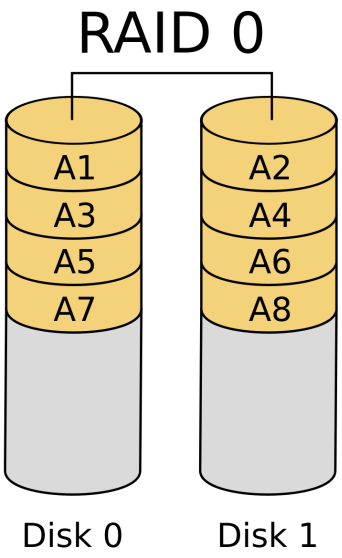
Combining Storage Devices

Many flavors of “RAID”: striping, mirroring, etc to increase **performance** and **reliability**

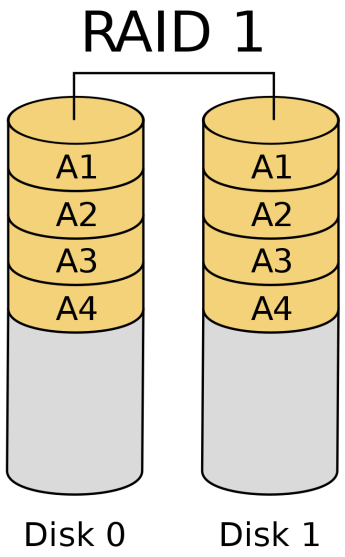


logically one disk

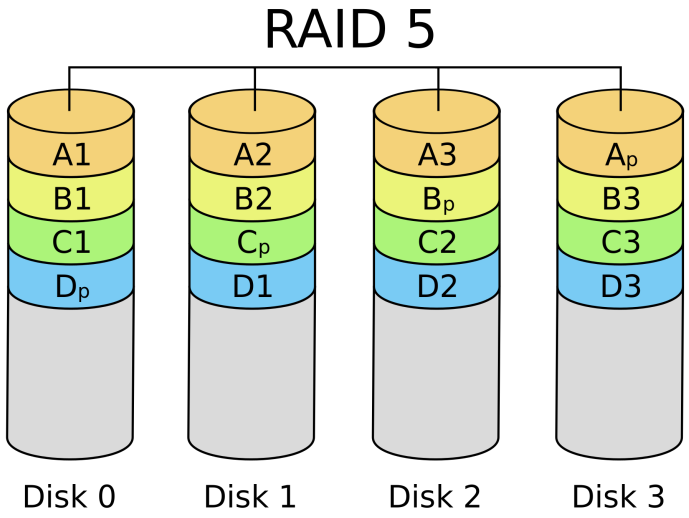
Common RAID Levels



Striping across 2 disks: adds performance but not reliability



Mirroring across 2 disks: adds reliability but not performance (except for reads)



Striping + 1 parity disk: adds performance and reliability at lower storage cost

Image source: Wikipedia

Handling Storage Failures

Detection: e.g., checksums

Correction: requires replicating data

Can be done at various levels:

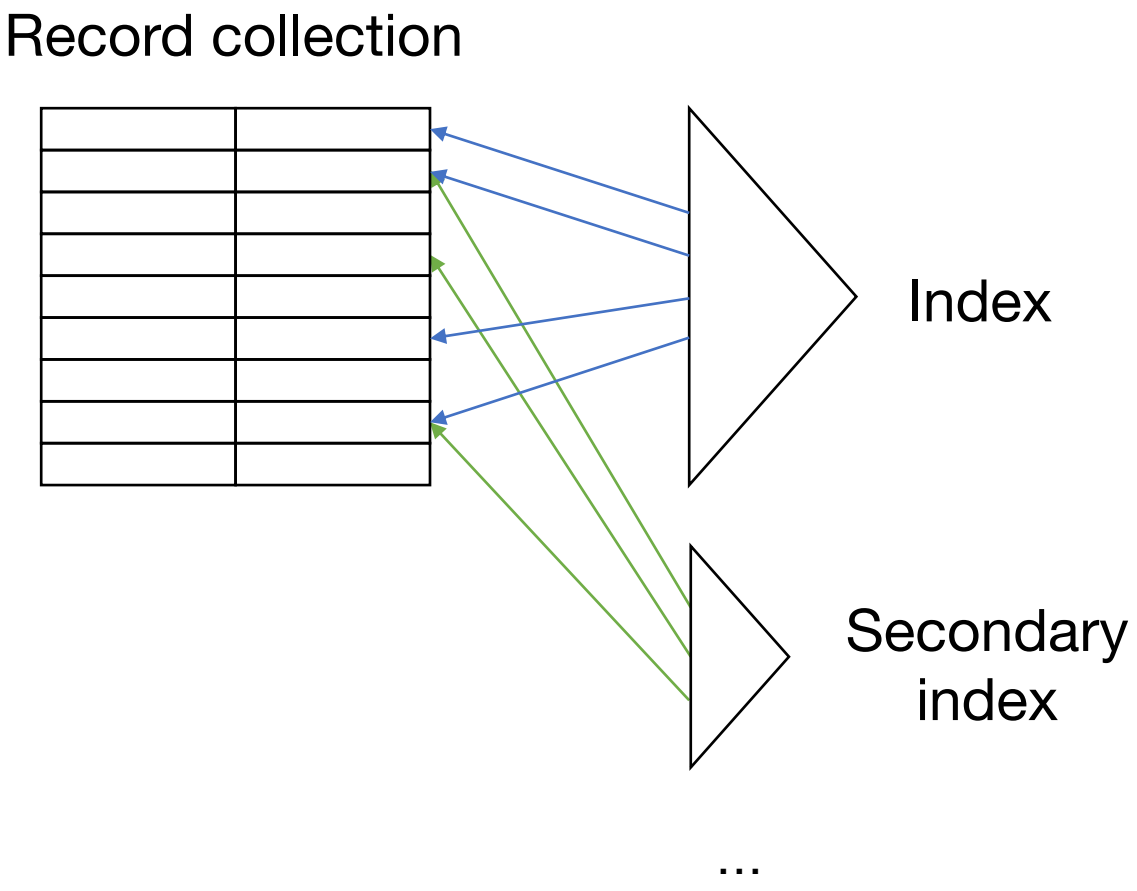
- » Single device (e.g., ECC RAM)
- » Disk array
- » OS
- » Database system (e.g., logging)

Designing Storage Formats

Key concerns:

- » **Access time:** minimize # of random accesses, bytes transferred, etc
  - Main way: place co-accessed data together!
- » **Space:** storage costs $
- » **Ease of updates**

General Setup



Outline

Storage devices wrap-up

Record encoding

Collection storage

C-Store paper

Indexes

What Are the Data Items We Want to Store?

a salary

a name

a date

a picture

What Are the Data Items We Want to Store?

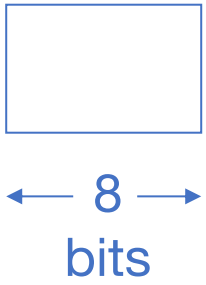
a salary

a name

a date

a picture

What we have available: bytes



Fixed-Length Items

Integer: fixed # of bytes (e.g., 2 bytes)

e.g., 35 is | 00000000 | 00100011 |

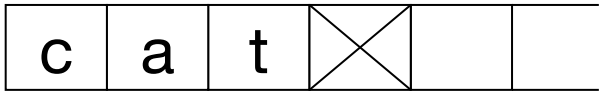
Floating-point: n-bit mantissa, m-bit exponent

Character: encode as integer (e.g. ASCII)

Variable-Length Items

String of characters:

» Null-terminated

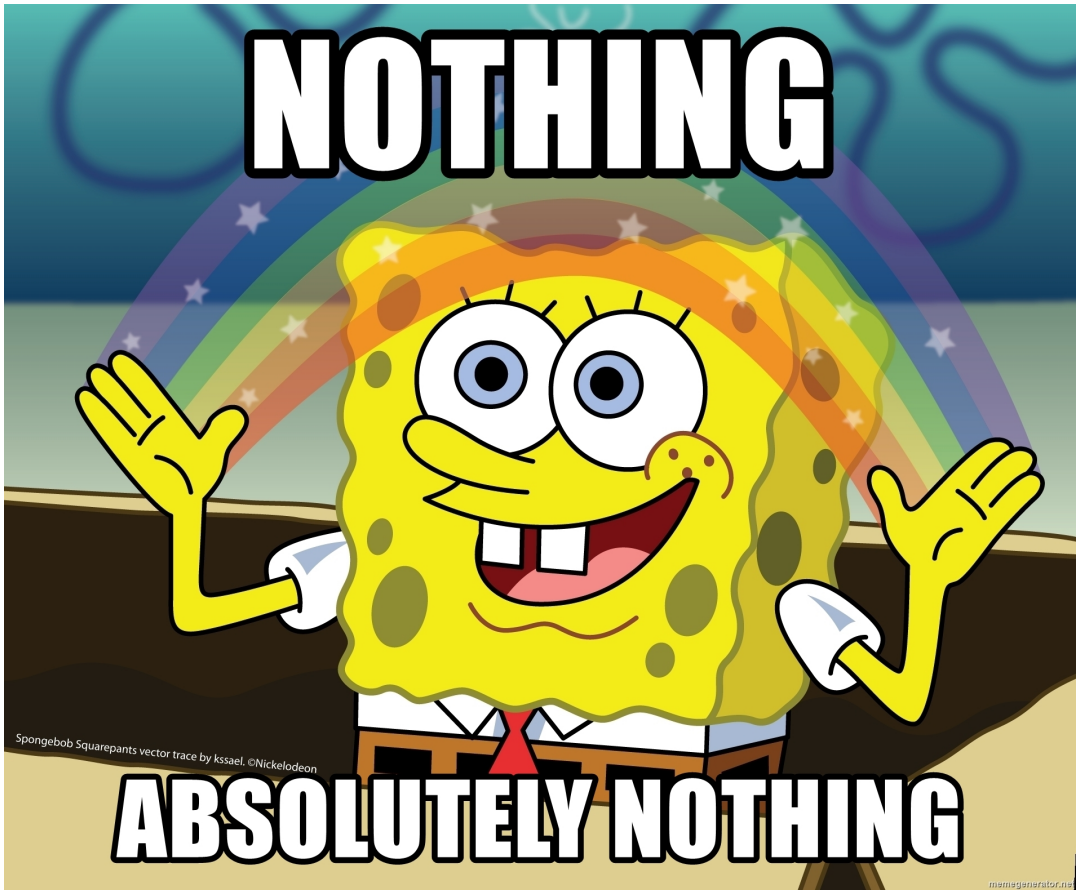


» Length + data

» Fixed-length

Bag of bits:

Representing



Representing Nothing

NULL concept in SQL (not same as 0 or “”)

Physical representation options:

- » Special “sentinel” value in fixed-length field
- » Boolean “is null” flag
- » Just skip the field in a sparse record format

Pretty common in practice!

Bigger Collections

Data Items

Records

Blocks

Files

Record: Set Data Items (Fields)

E.g. employee record:

- » name field
- » salary field
- » date-of-hire field
- » ...

Record Encodings

Fixed vs variable **format**

Fixed vs variable **length**

Fixed Format

A **schema** for all records in table specifies:

- # of fields
- type of each field
- order in record
- meaning of each field

Example: Fixed Format & Length

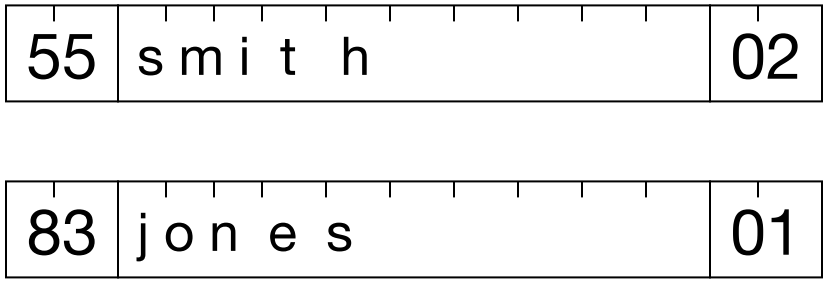
Employee record

(1) EID, 2 byte integer

(2) Name, 10 chars

(3) Dept, 2 byte code

Schema



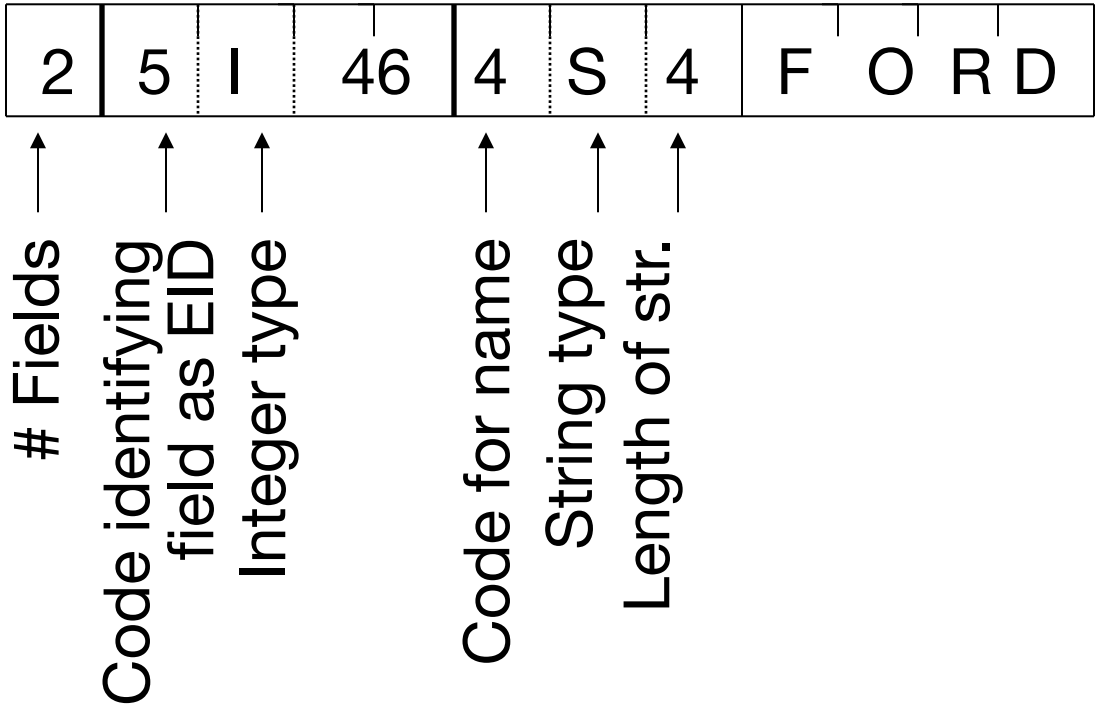
Records

Variable Format

Record itself contains format

"Self-describing"

Example: Variable Format & Length



Variable Format Useful For

“Sparse” records

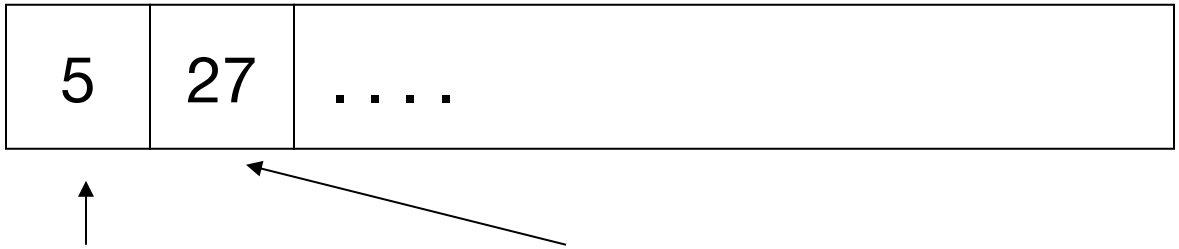
Repeating fields

Evolving formats

But may waste space...

Many Variants Between Fixed and Variable Format

Example: Include a **record type** in record



record type record length

Type is a pointer to one of several schemas

Outline

Overview

Record encoding

Collection storage

Indexes

Collection Storage Questions

How do we place data items and records for efficient access?

» **Locality** and **searchability**

How do we physically encode records in blocks and files?

Placing Data for Efficient Access

Locality: which items are accessed together

» When you read one field of a record, you're likely to read other fields of the same record

» When you read one field of record 1, you're likely to read the same field of record 2

Searchability: quickly find relevant records

» E.g. sorting the file lets you do binary search

Locality Example: Row Stores vs Column Stores

Row Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Fields stored contiguously in one file

Column Store

name
Alex
Bob
Carol
David
Eve
Frances
Gia
Harold
Ivan

age
20
30
42
21
26
56
19
28
41

state
CA
CA
NY
MA
CA
NY
MA
AK
CA

Each column in a different file

Locality Example: Row Stores vs Column Stores

Row Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Fields stored contiguously in one file

Column Store

name
Alex
Bob
Carol
David
Eve
Frances
Gia
Harold
Ivan

age
20
30
42
21
26
56
19
28
41

state
CA
CA
NY
MA
CA
NY
MA
AK
CA

Each column in a different file

Accessing all fields of one record: 1 random I/O for row, 3 for column

Locality Example: Row Stores vs Column Stores

Row Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Fields stored contiguously in one file

Column Store

name
Alex
Bob
Carol
David
Eve
Frances
Gia
Harold
Ivan

age
20
30
42
21
26
56
19
28
41

state
CA
CA
NY
MA
CA
NY
MA
AK
CA

Each column in a different file

Accessing one field of all records: 3x less I/O for column store

Can We Have Hybrids Between Row & Column?

Yes! For example, colocated **column groups**:

name
Alex
Bob
Carol
David
Eve
Frances
Gia
Harold
Ivan

File 1

age	state
20	CA
30	CA
42	NY
21	MA
26	CA
56	NY
19	MA
28	AK
41	CA

File 2: age & state

Helpful if age & state are frequently co-accessed

Improving Searchability: Ordering

Ordering the data by a field will give:

- » Closer I/Os if queries tend to read data with nearby values of the field (e.g. time ranges)
- » Option to accelerate search via an ordered index (e.g. B-tree), binary search, etc

What's the downside of having an ordering?

Improving Searchability: Partitions

Just place data into buckets based on a field (but not necessarily fine-grained order)

E.g. Hive table storage over a filesystem:

```
/my_table/date=20190101/file1.parquet
                       /file2.parquet
         /date=20190102/file1.parquet
                       /file2.parquet
         /date=20190103/file1.parquet
                       ...
```

Easy to add, remove & list files in any directory

Can We Have Searchability on Multiple Fields at Once?

Yes! Many possible ways:

1) Multiple partition or sort keys (e.g., partition by date, then sort by userID)

2) Interleaved orderings such as Z-ordering

Z-Ordering

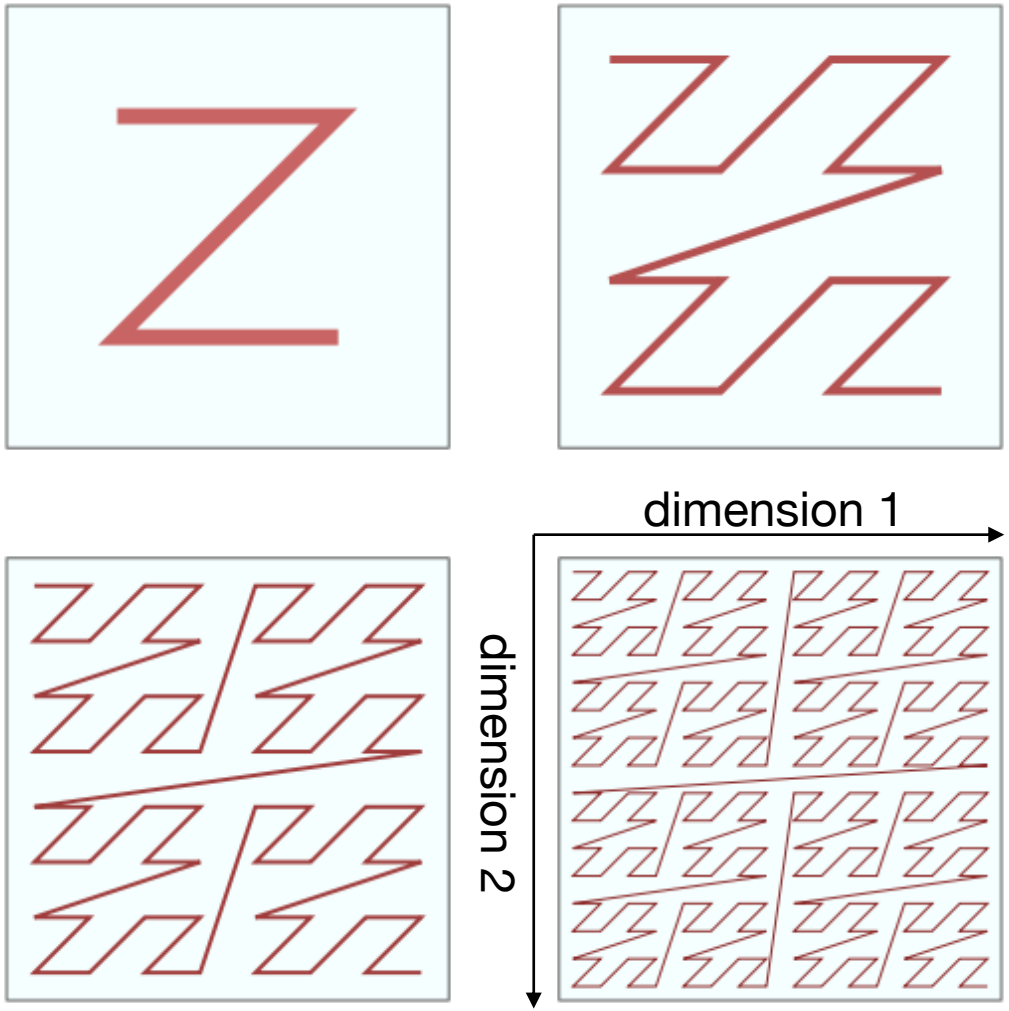
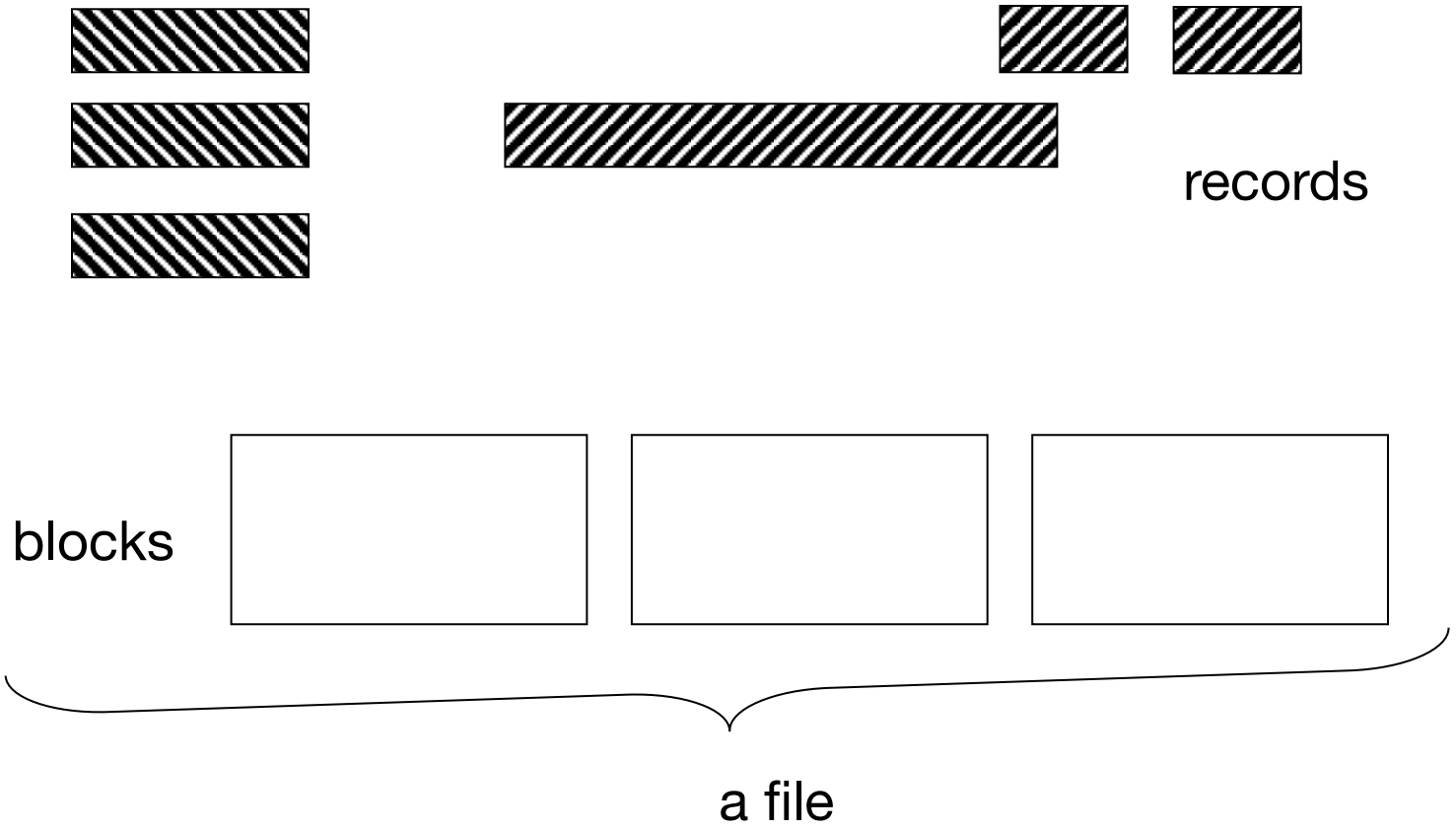


Image source: Wikipedia

How Do We Encode Records into Blocks & Files?



Questions in Storing Records

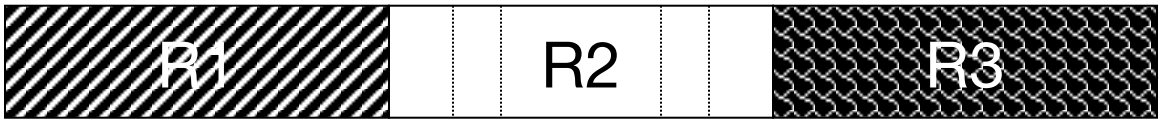
(1) separating records

(2) spanned vs. unspanned

(3) indirection

(1) Separating Records

Block



(a) no need to separate - fixed size recs.

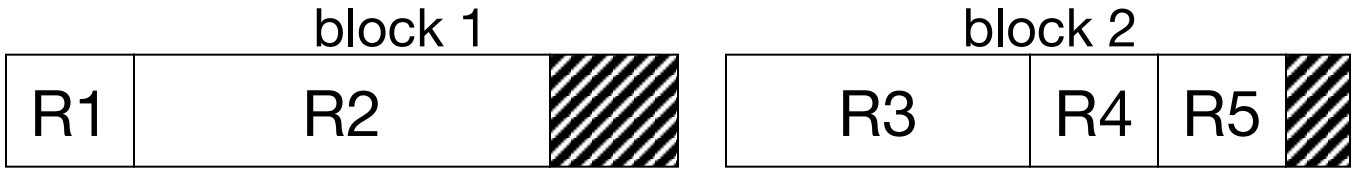
(b) special marker

(c) give record lengths (or offsets)

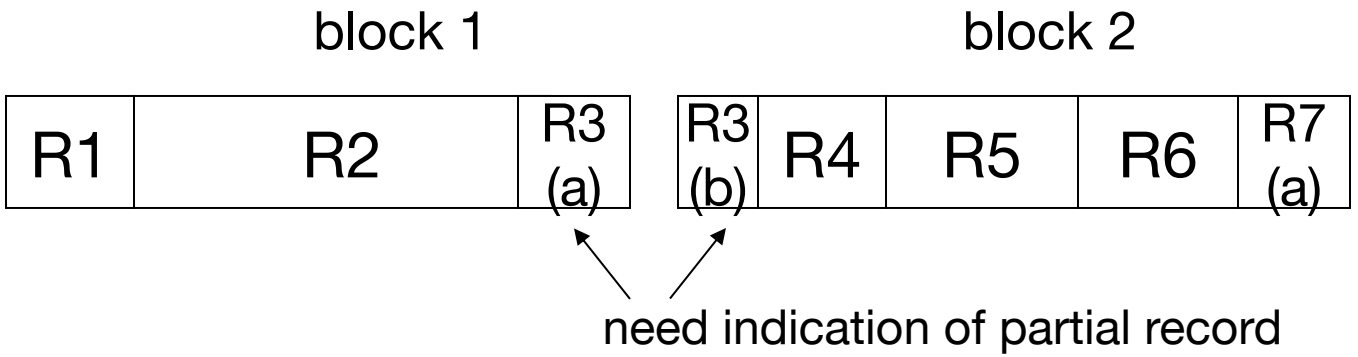
- within each record
- in block header

(2) Spanned vs Unspanned

Unspanned: records must be within one block



Spanned:



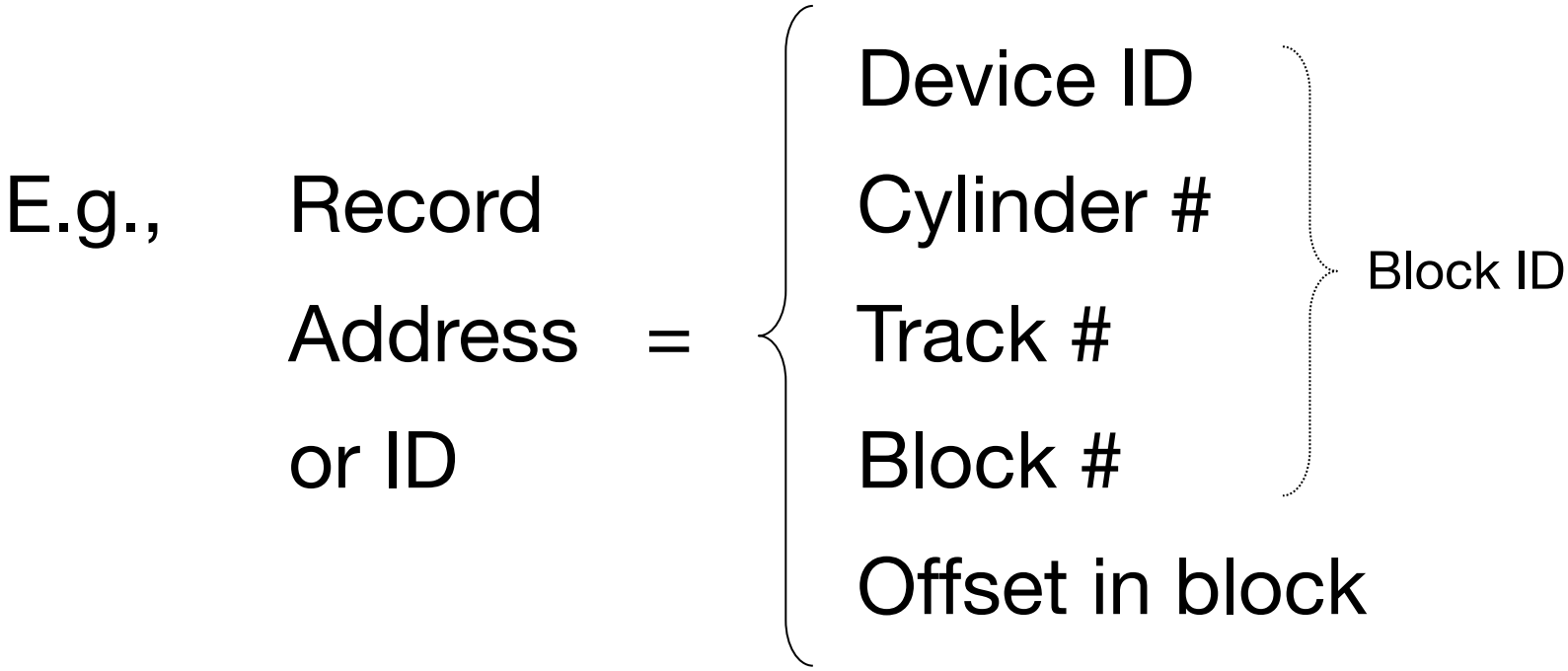
(3) Indirection

How does one refer to other records?

Many options:

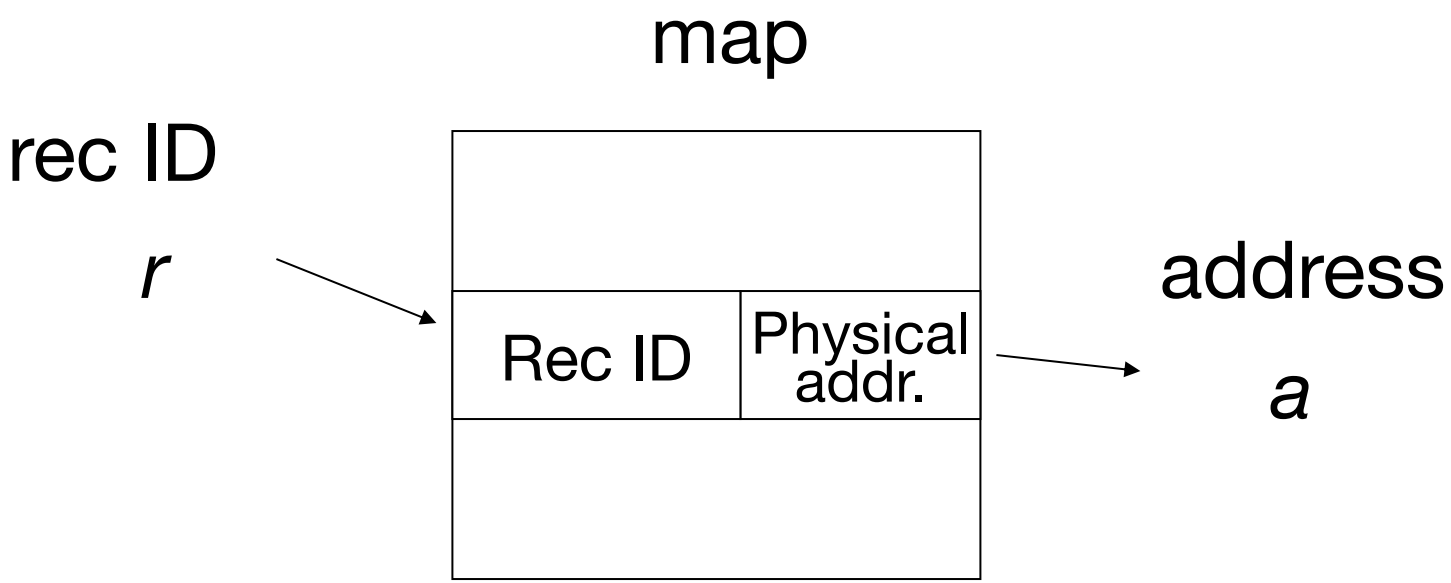
Physical ⟷ Indirect

Purely Physical



Fully Indirect

E.g., Record ID is arbitrary bit string



Tradeoff

Flexibility  Cost

to move records

of indirection

(for deletions, insertions)

Inserting Records

Easy case: records not ordered

» Insert record at end of file or in a free space

» Harder if records are variable-length

Hard case: records are ordered

» If free space close by, not too bad...

» Otherwise, use an **overflow** area and reorganize the file periodically

Deleting Records

Immediately reclaim space

OR

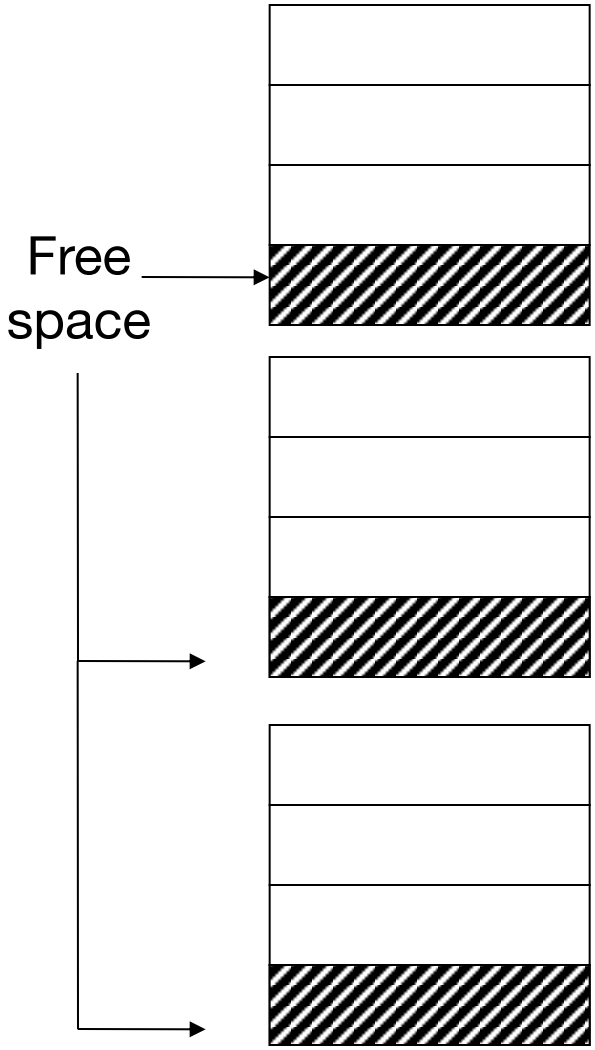
Mark deleted

– And keep track of freed spaces for later use

Interesting Problems

How much free space to leave in each block, track, cylinder, etc?

How often to reorganize file + merge overflow?



Compressing Collections

Usually for a block at a time

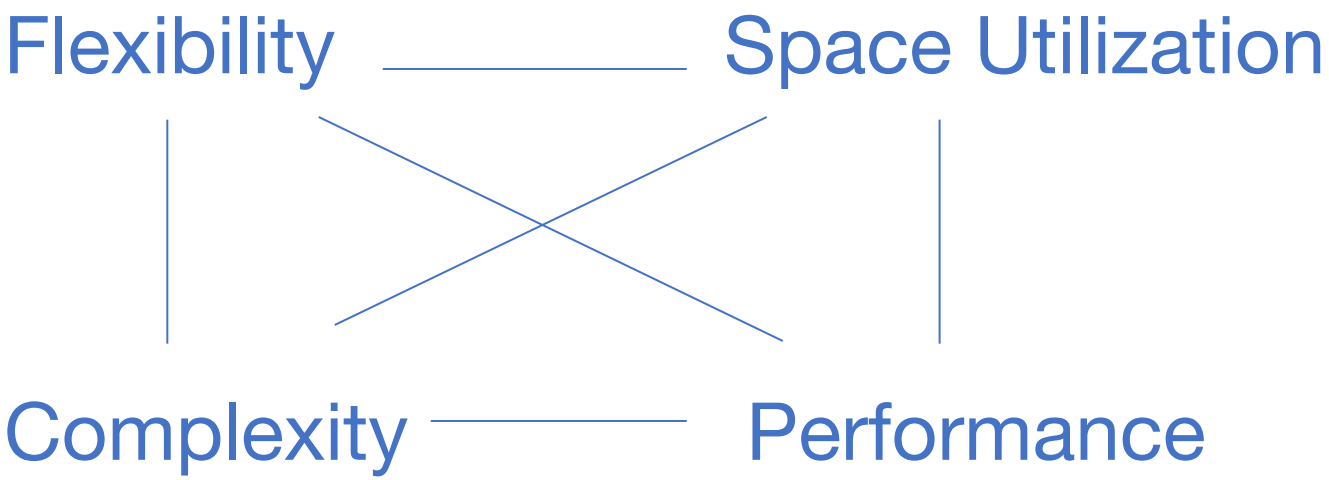
» Benefits from placing similar items together

Can be integrated with execution (C-Store)

Summary

There are many ways to organize data on disk

Key tradeoffs:



To Evaluate a Strategy, Compute:

Space used for expected data

Expected time to

- fetch record given key
- read whole file
- insert record
- delete record
- update record
- reorganize file
- …

Outline

Storage devices wrap-up

Record encoding

Collection storage

C-Store paper

Indexes

C-Store Storage

The storage construct was a “projection”; what does that mean?

C-Store Compression

Five types of compression:

- » Null suppression
- » Dictionary encoding
- » Run-length encoding
- » Bit-vector encoding
- » Lempel-Ziv

Tradeoff: size vs ease of computation

API for Compressed Blocks

Properties	Iterator Access	Block Information
isOneValue()	getNext()	getSize()
isValueSorted()	asArray()	getStartValue()
isPosContig()		getEndPosition()

Table 1: Compressed Block API

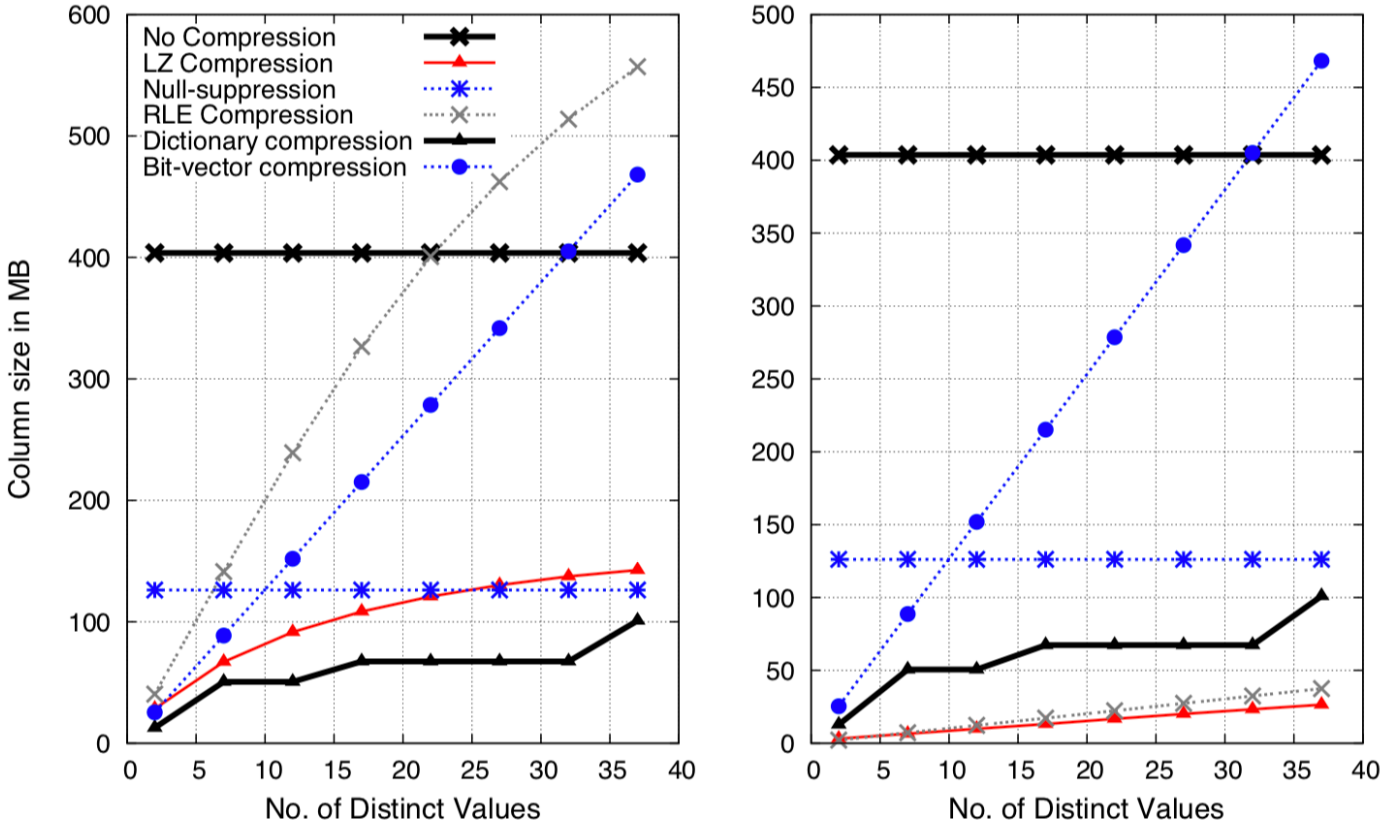
Encoding Type	Sorted?	1 value?	Pos. contig.?
RLE	yes	yes	yes
Bit-string	yes	yes	no
Null Supp.	no/yes	no	yes
Lempel-Ziv	no/yes	no	yes
Dictionary	no/yes	no	yes
Uncompressed	no/yes	no	no/yes

Using the Block API

```
COUNT(COLUMN c1)
 b = GET NEXT COMPRESSED BLOCK FROM c1
 WHILE b IS NOT NULL
   IF b.ISONEVALUE()
     x = FETCH CURRENT COUNT FOR b.GETSTARTVAL()
     x = x + b.GETSIZE()
   ELSE
     a = b.ASARRAY()
     FOR EACH ELEMENT i IN a
       x = FETCH CURRENT COUNT FOR i
       x = x + 1
   b = GET NEXT COMPRESSED BLOCK FROM c1
```

Figure 2: Pseudocode for Simple Count Aggregation

Data Size with Each Scheme

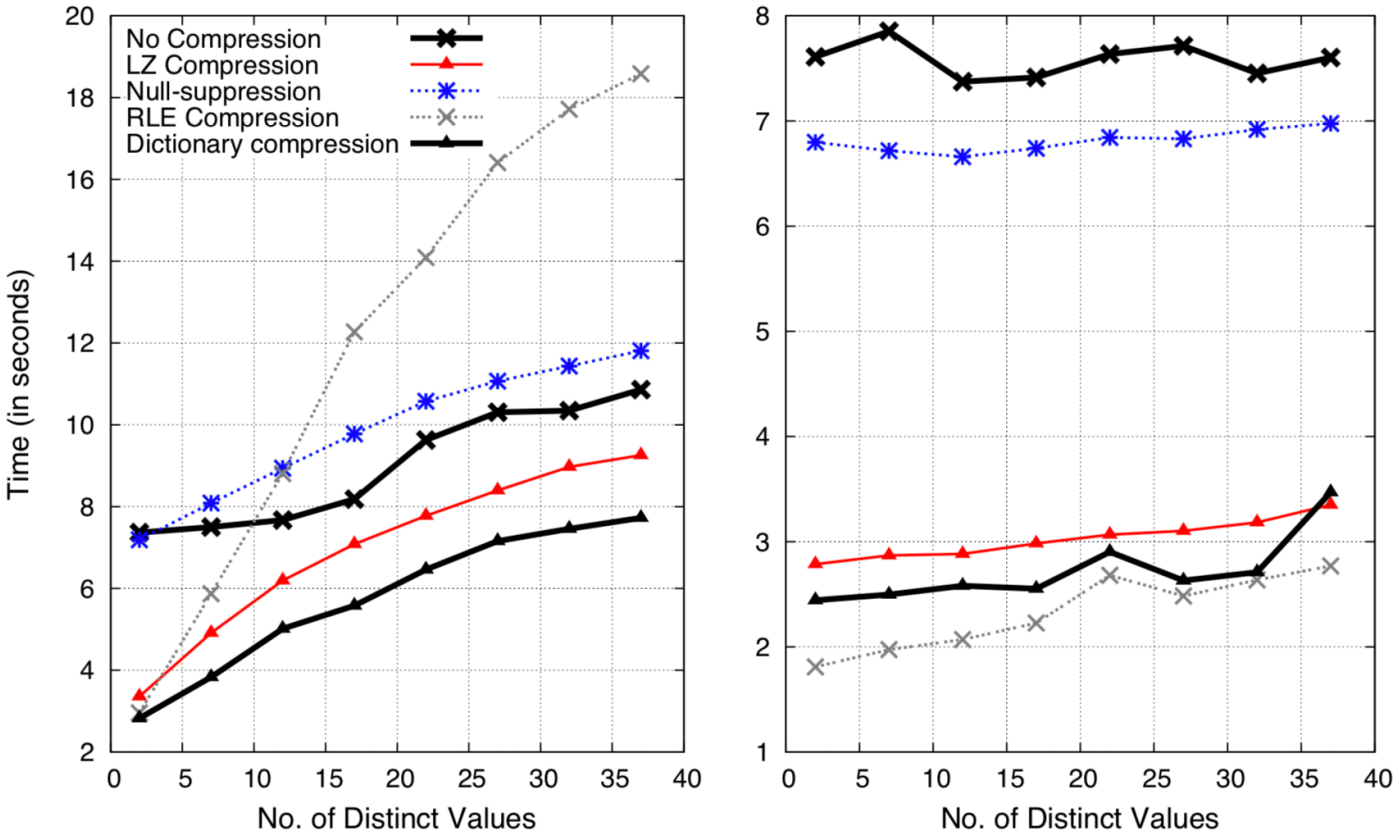


(a) Sorted runs of length 50

(a) Sorted runs of length 1000

Figure 4: Compressed column sizes for varied compression schemes on column with sorted runs of size 50 (a) and 1000 (b)

Performance: Eager Decompress

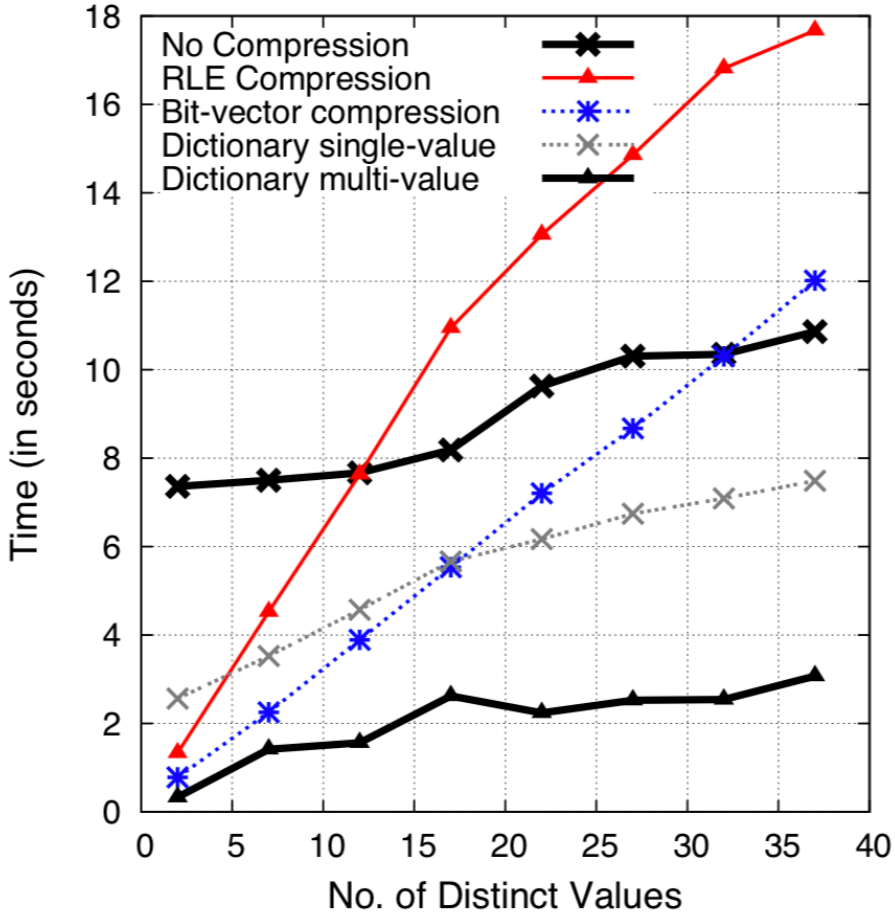


(a) Sorted runs of length 50

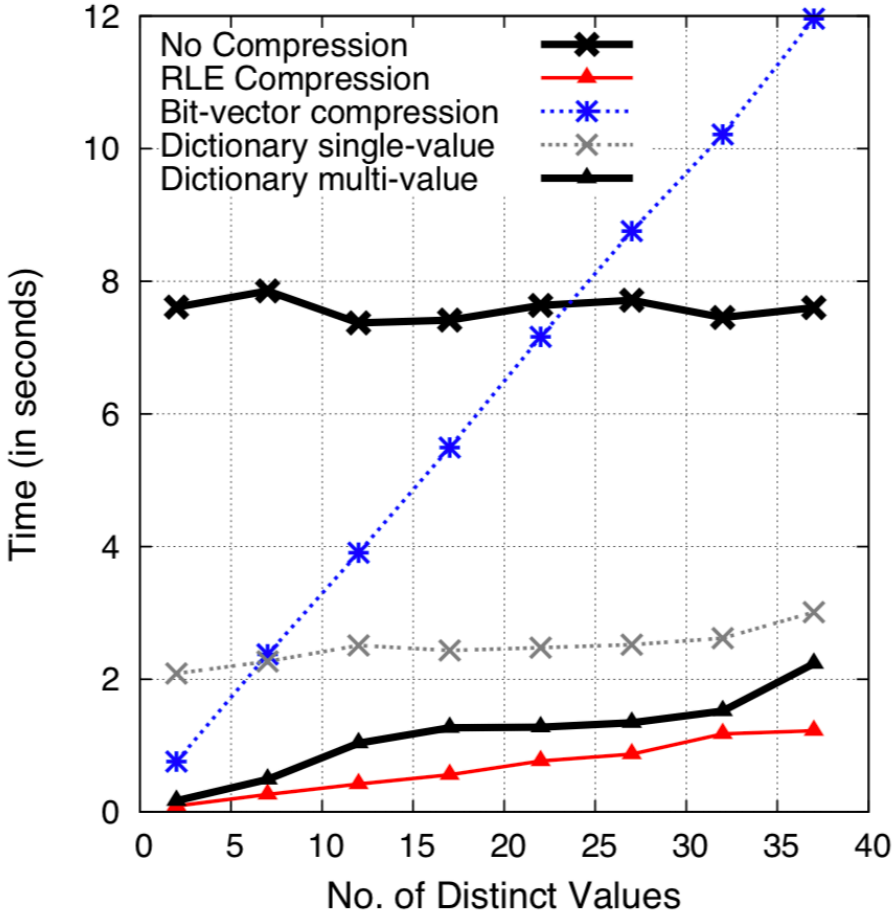
(b) Sorted runs of length 1000

Figure 5: Query Performance With Eager Decompression on column with sorted runs of size 50 (a) and 1000 (b)

Performance: Compressed Eval



(a) Sorted runs of length 50



(b) Sorted runs of length 1000

Fig ata

How would the results change on SSDs?