Data Storage Formats

Outline

Storage devices wrap-up

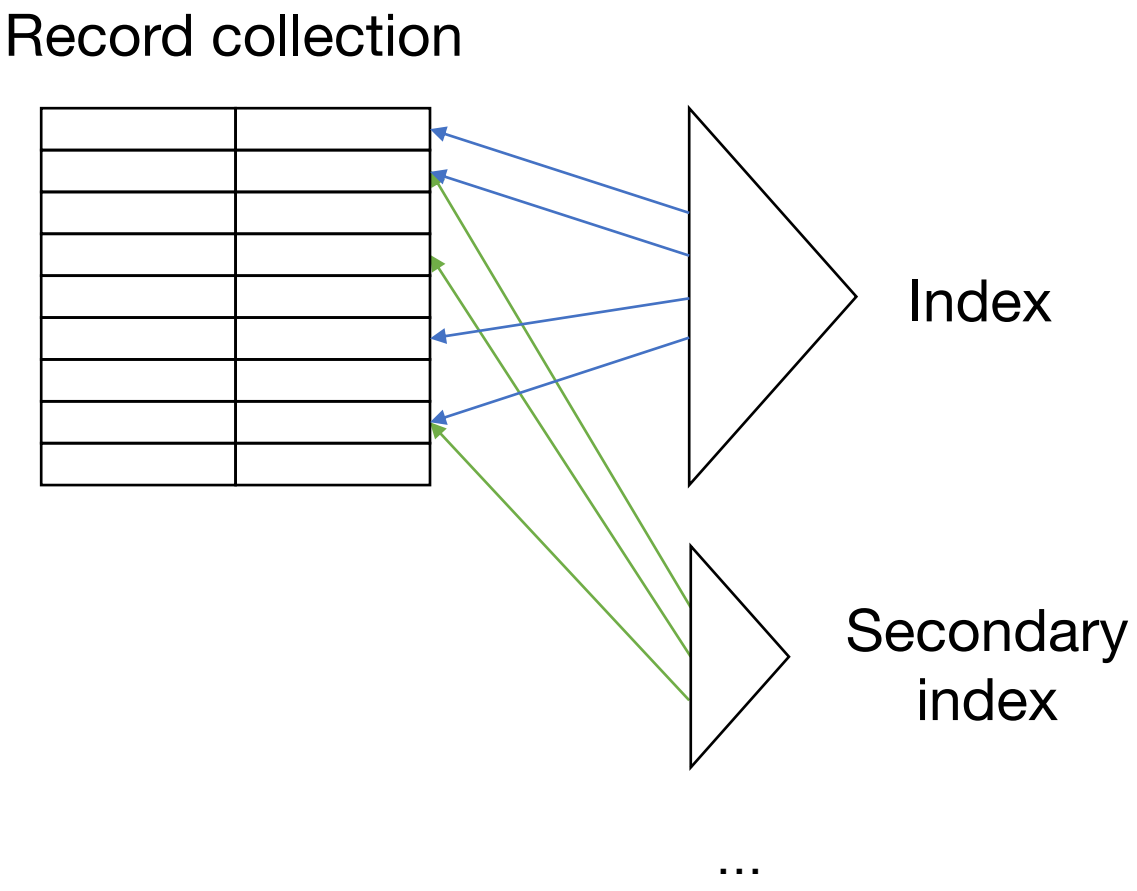
Record encoding

Collection storage

C-Store paper

Indexes

General Setup



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?

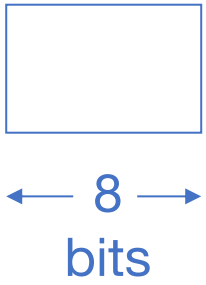
a salary

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 |

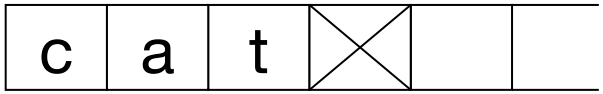
Floating-point: n-bit mantissa, m-bit exponent

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

Variable-Length Items

String of characters:

» Null-terminated



» Length + data

» Fixed-length

Bag of bits:

Representing 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

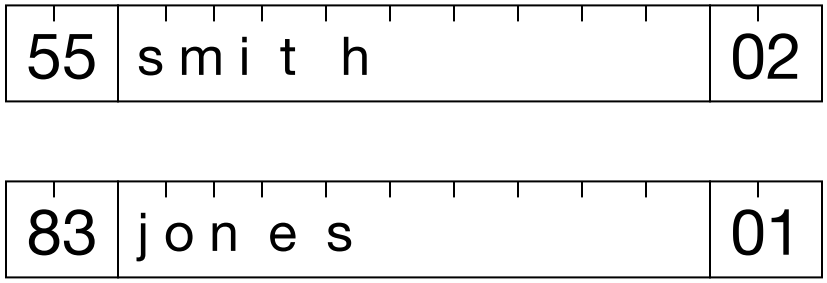
Employee record

(1) EID, 2 byte integer

(2) Name, 10 chars

(3) Dept, 2 byte code

Schema



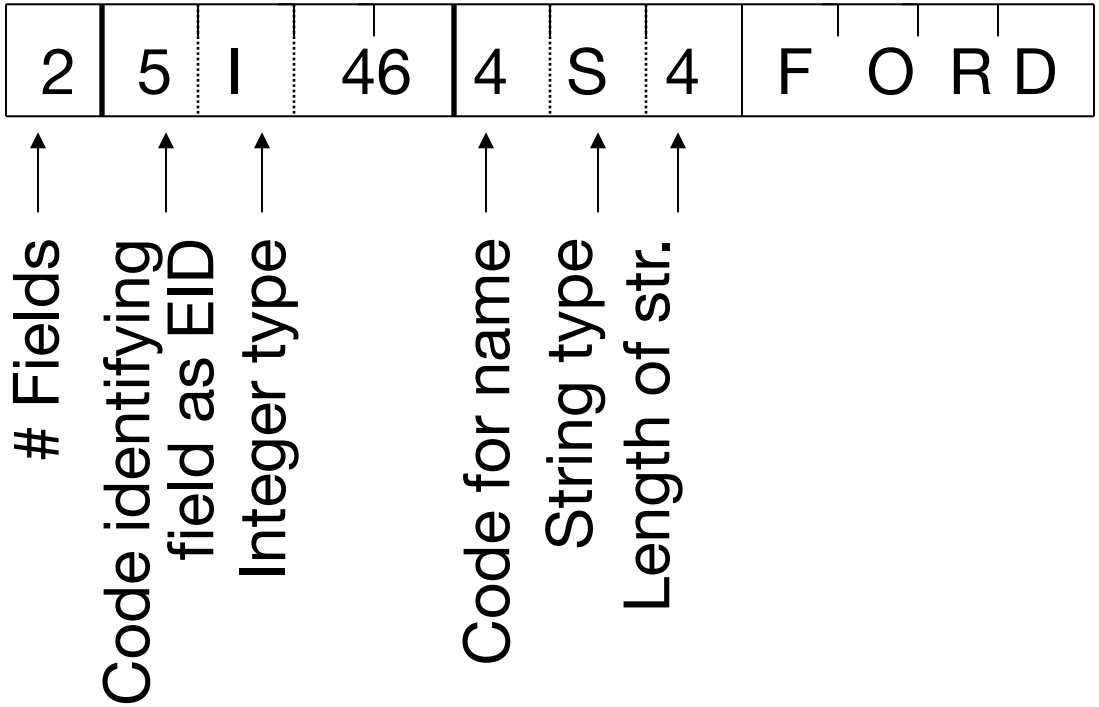
Records

Variable Format

Record itself contains format

"Self-describing"

Example: Variable Format & Length



Variable Format Useful For

“Sparse” records

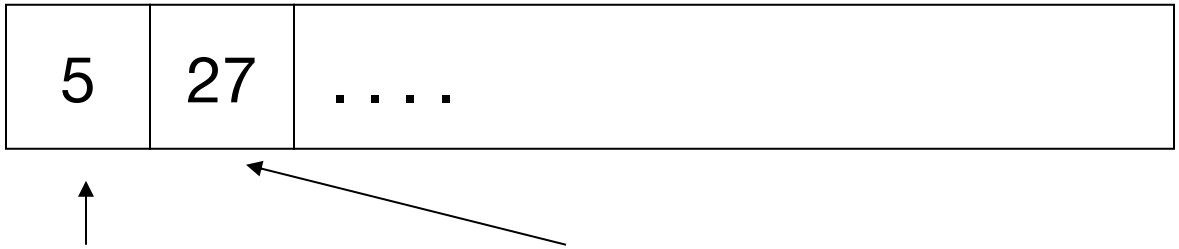
Repeating fields

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

Z-Ordering

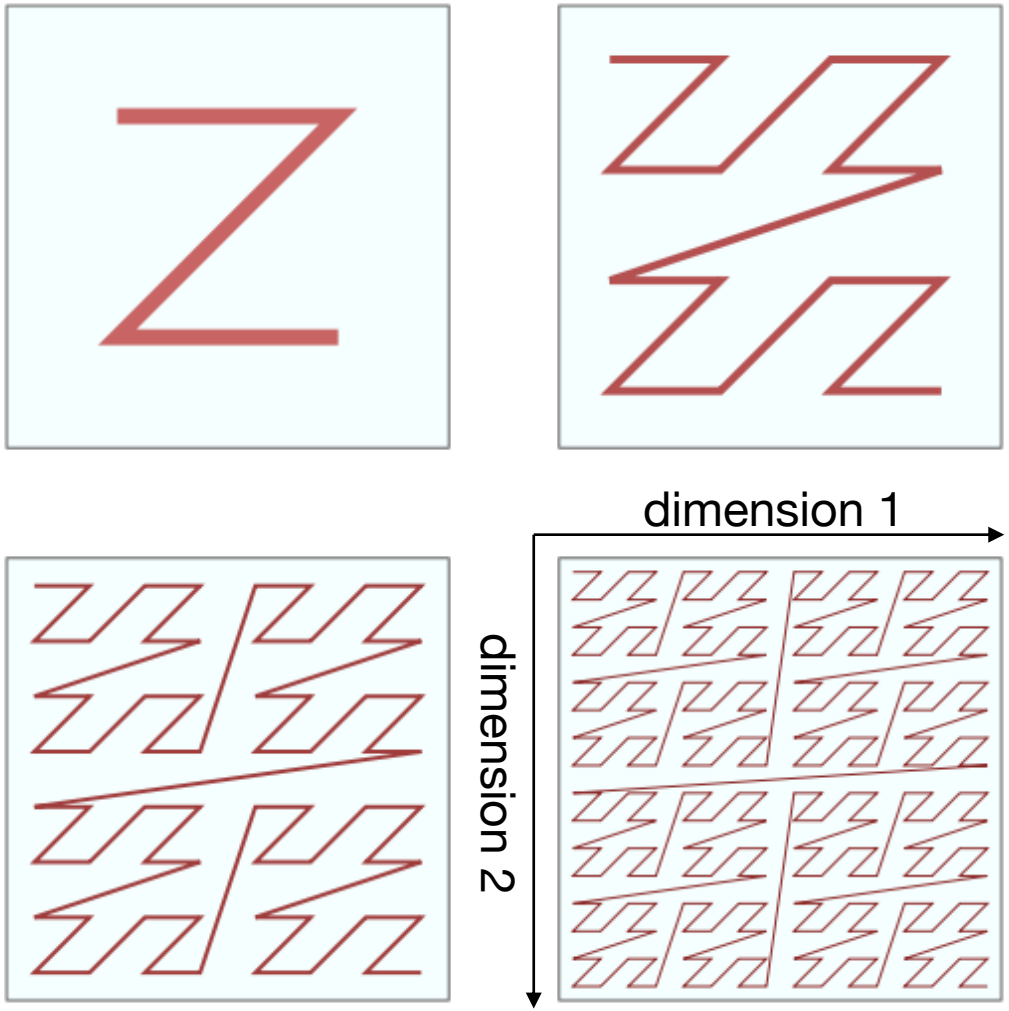
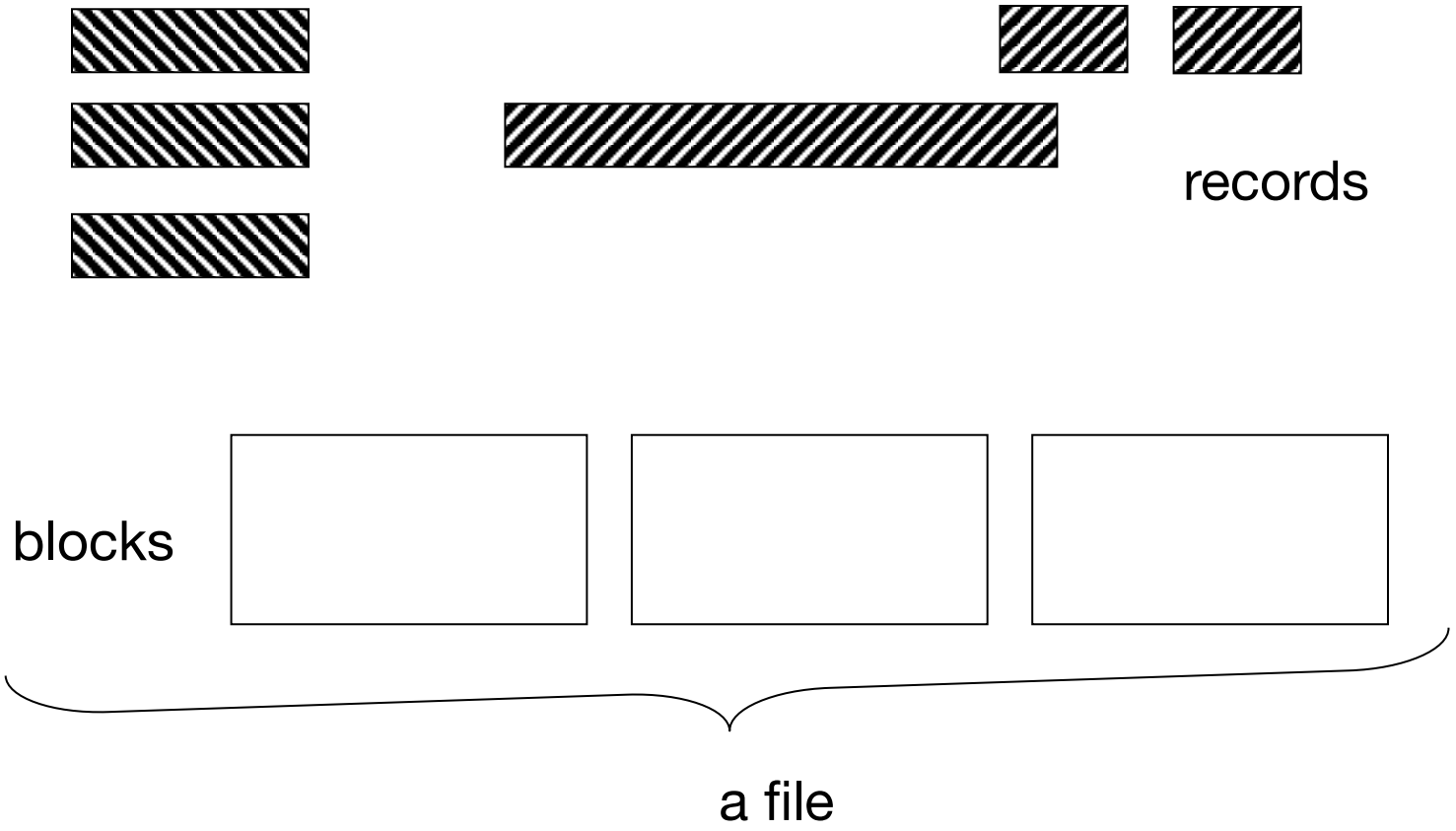
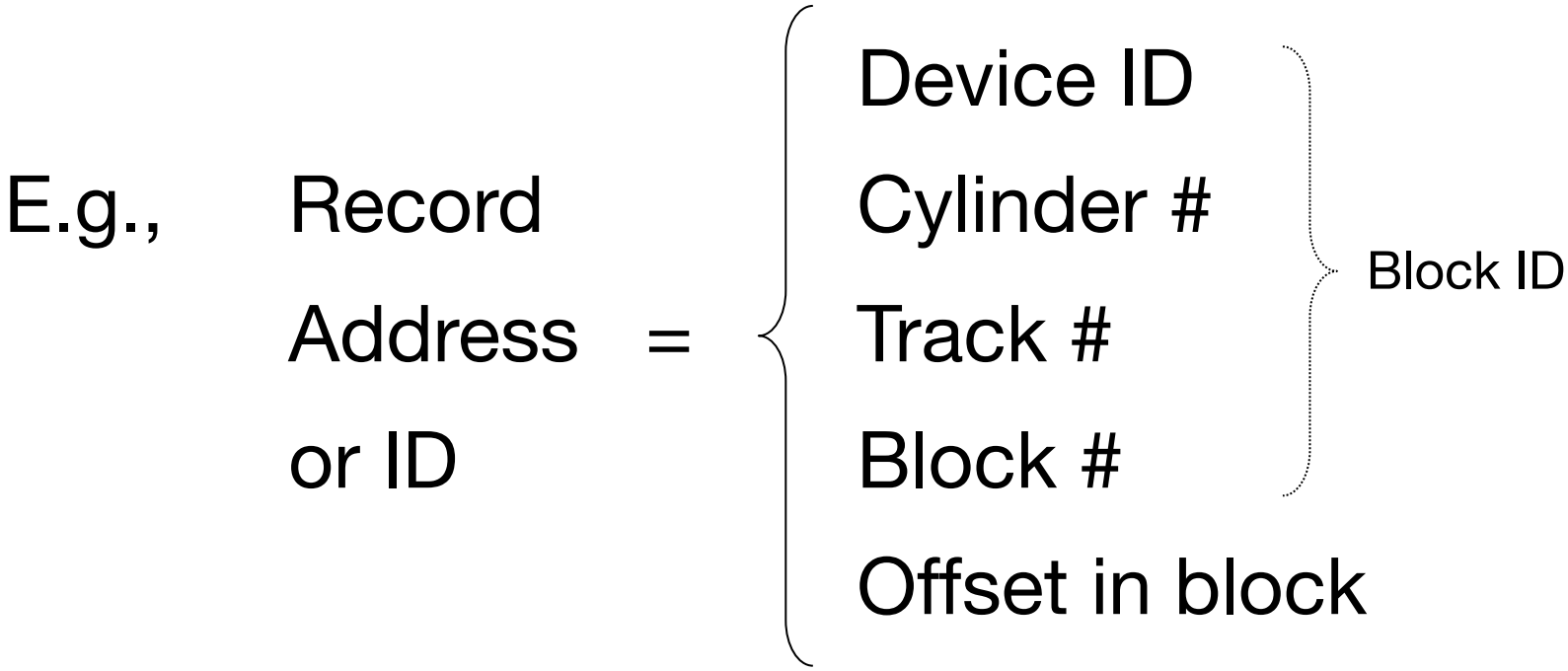


Image source: Wikipedia

How Do We Encode Records into Blocks & Files?

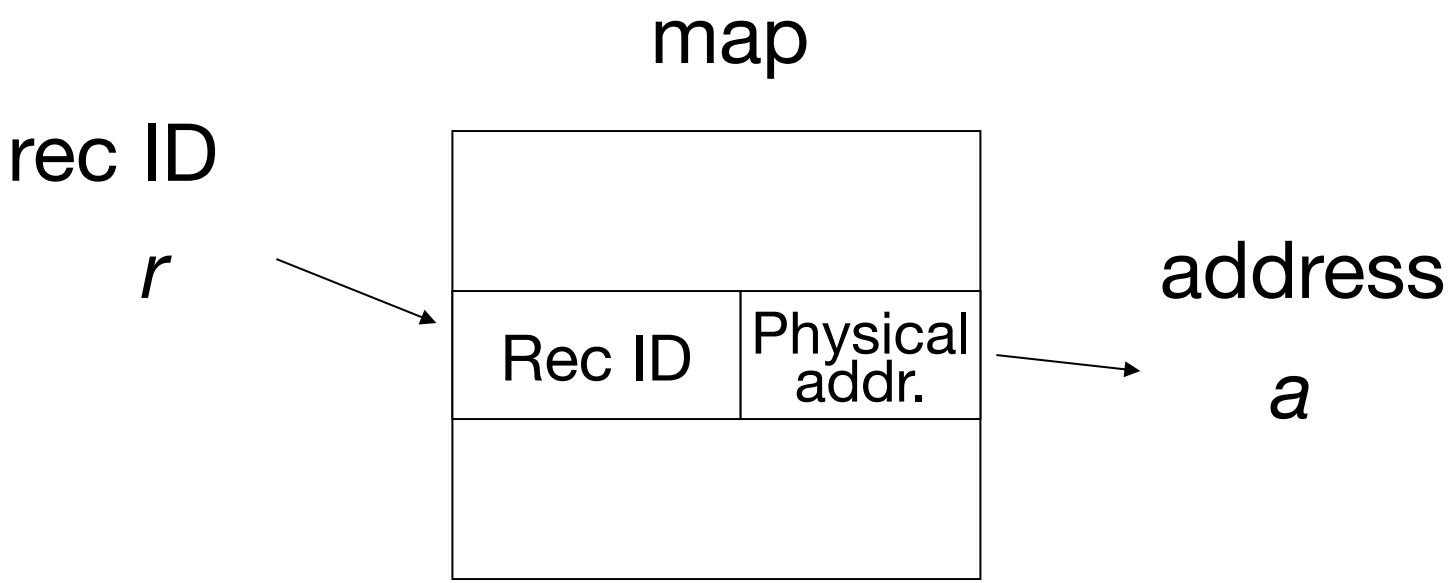


Purely Physical



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

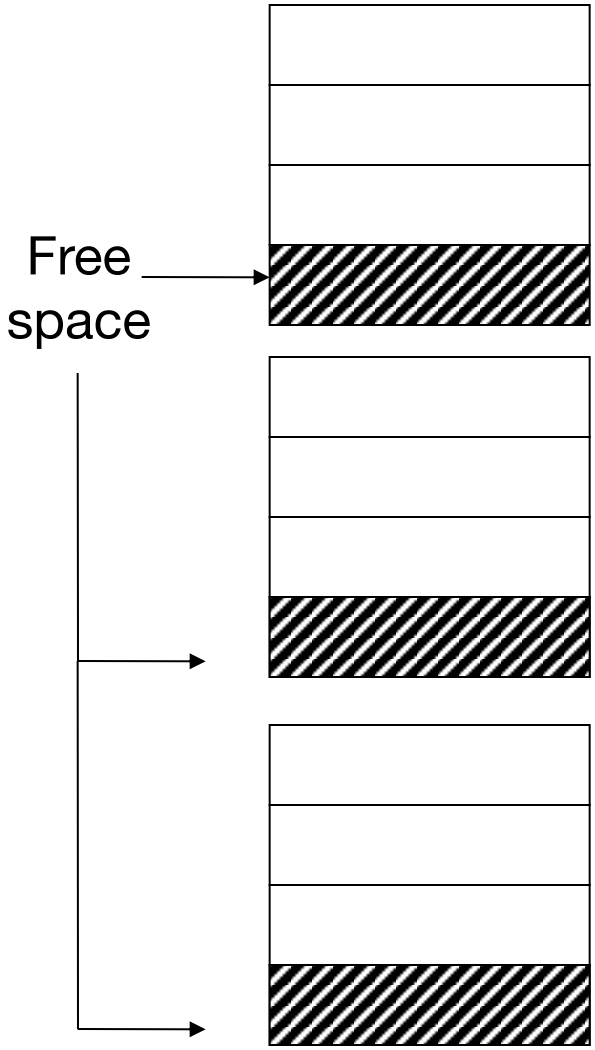
Mark deleted

– 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

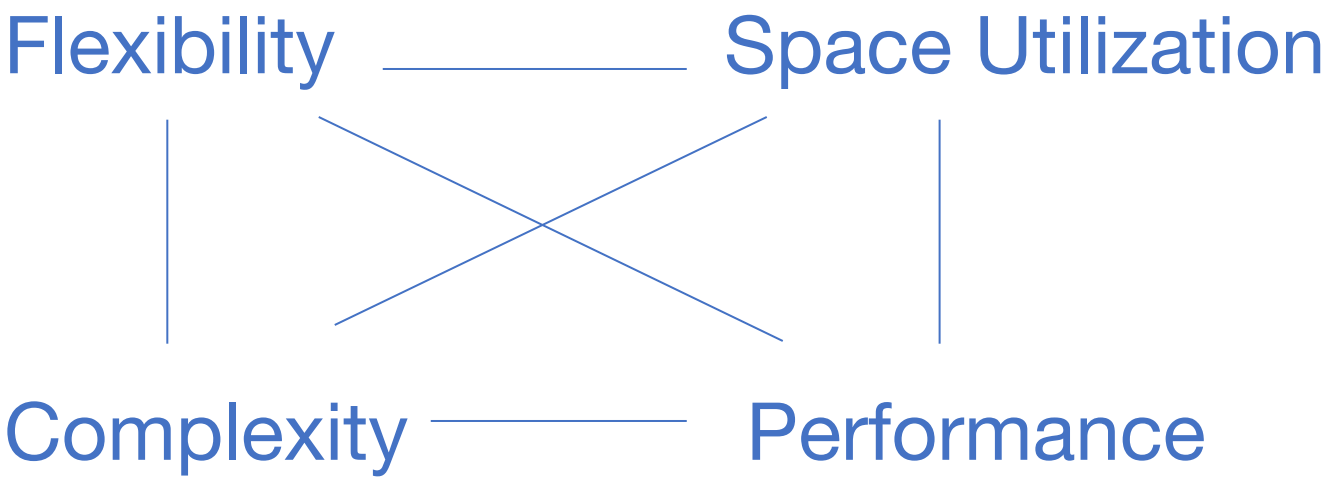
» Benefits from placing similar items together

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
- …