Data Storage Formats

Instructor: Matei Zaharia

cs245.stanford.edu

Outline

Overview

Record encoding

Collection storage

Indexes

Outline

Overview

Record encoding

Collection storage

Indexes

Overview

Recall from last time: I/O slow compared to compute, random I/O ≪ sequential

Key concerns in storage:

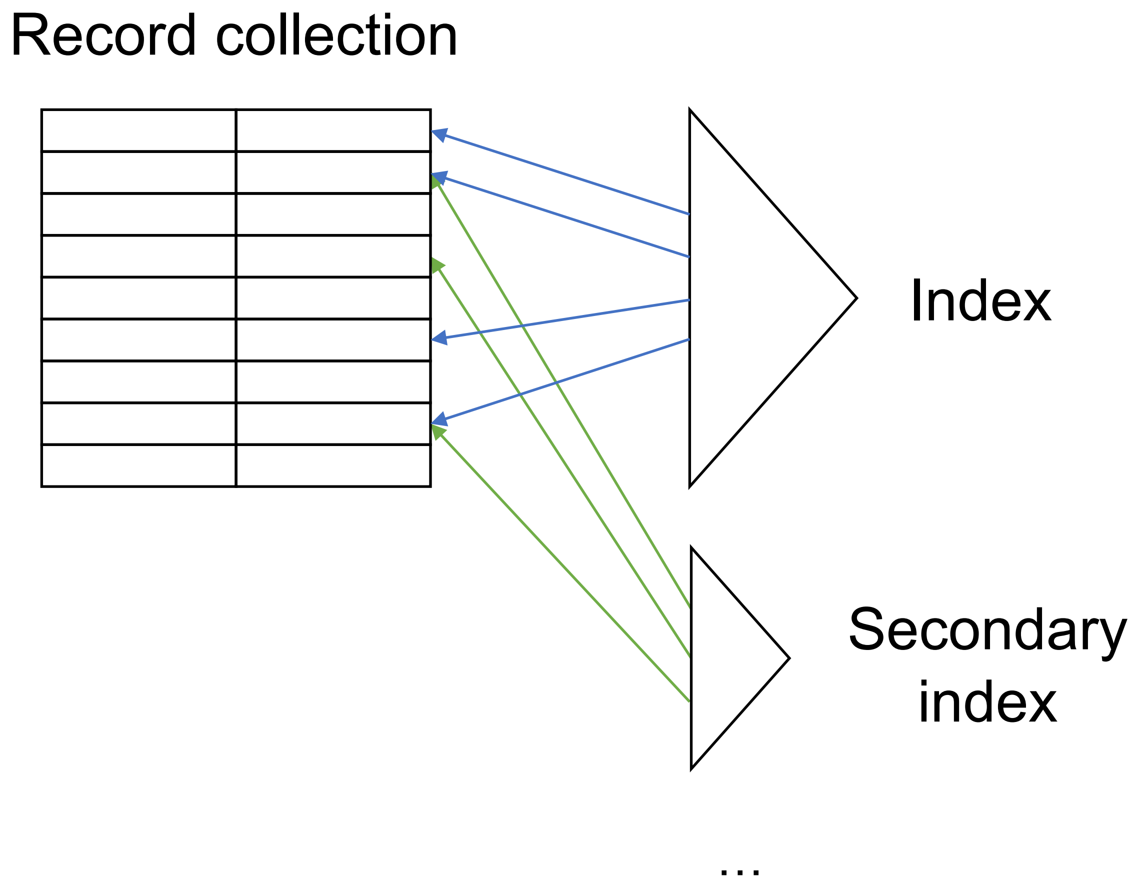
» **Access time:** minimize # of random accesses, bytes transferred, etc

- Main way: place co-accessed data together!

» **Size:** storage costs $

» **Ease of updates**

General Setup



Outline

Overview

Record encoding

Collection storage

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?

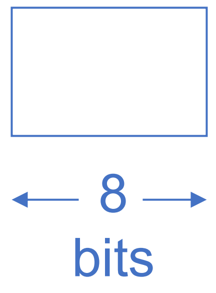
a salary

a name

a date

a picture

What we have available: **bytes**



To Represent:

Integer (short): 2 bytes

e.g., 35 is $\boxed{00000000}$ $\boxed{00100011}$

Real, floating point

n bits for mantissa, m for exponent….

To Represent:

Characters

→ Various coding schemes available

<u>Example: ASCII</u>

A:	1000001
a:	1100001
5:	0110101
LF:	0001010

To Represent:

Boolean

e.g., TRUE $\boxed{1111\ 1111}$

FALSE $\boxed{0000\ 0000}$

Application specific

e.g., RED $\rightarrow$ 1 GREEN $\rightarrow$ 3

BLUE $\rightarrow$ 2 YELLOW $\rightarrow$ 4 …

$\Rightarrow$ Can we use less than 1 byte/code?

Yes, but only if desperate...

To Represent:

Dates

e.g.: - Integer, # days since Jan 1, 1900
- 8 characters, YYYYMMDD
- 7 characters, YYYYDDD

Time

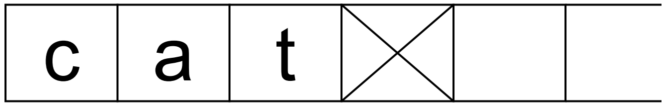
e.g. - Integer, seconds since midnight
- characters, HHMMSSFF

To Represent:

String of characters

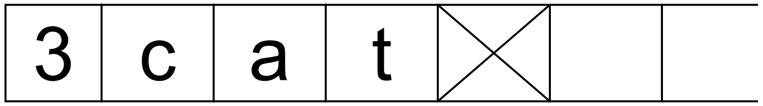
- » Null terminated

  e.g.,



- » Length given

  e.g.,



- Fixed length

To Represent:

Bag of bits

Length	Bits

To Represent:

To Represent: Nothing

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

Physical representation options:

» Special “sentinel” value in fixed0length field

» Boolean “is null” flag

» Just skip the field in a sparse record format

Pretty common in practice!

Key Point

- Fixed length items
- Variable length items
  - usually length given at beginning

Also

Type of an item: tells us how to interpret the bytes, plus size if fixed

Bigger Collections

$$\begin{tikzcd}
\text{Data Items} \arrow[d] \\
\text{Records} \arrow[d] \\
\text{Blocks} \arrow[d] \\
\text{Files}
\end{tikzcd}$$

Record: Set of Related Data Items (“Fields”)

E.g.: Employee record:

name field,

salary field,

date-of-hire field, ...

Types of Records

Main choices:

- » Fixed vs variable **format**
- » Fixed vs variable **length**

Fixed Format

A **schema** (not record) contains following info:

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

Example: Fixed Format & Length

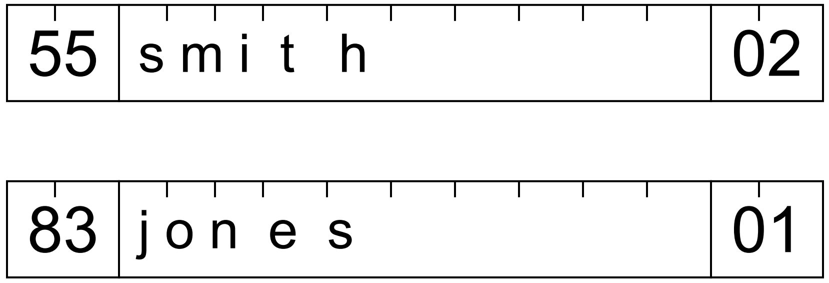
Employee record

(1) E#, 2 byte integer

(2) E.name, 10 char.

(3) Dept, 2 byte code

Schema



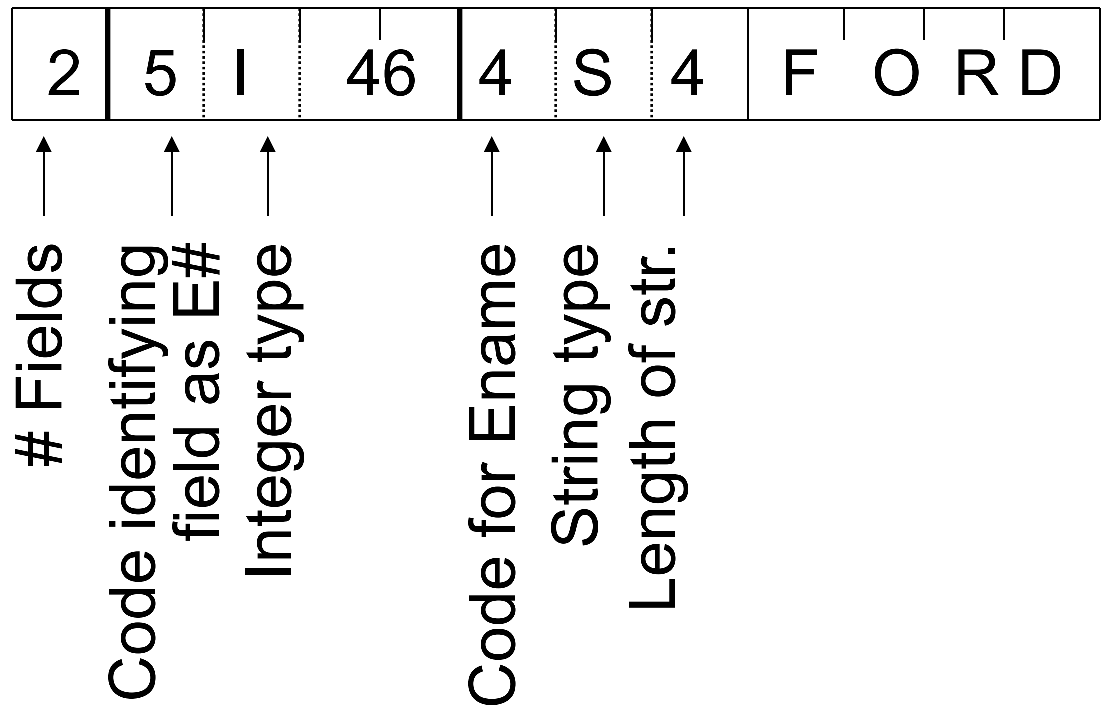
Records

Variable Format

Record itself contains format

“Self Describing”

Example: Variable Format & Length



Field name codes could also be strings, i.e. TAGS

Variable Format Useful For

"Sparse" records

Repeating fields

Evolving formats

But may waste space...

Example: Variable Format Record with Repeated Fields

Employee $\rightarrow$ one or more $\rightarrow$ children

3	E_name: Fred	Child: Sally	Child: Tom

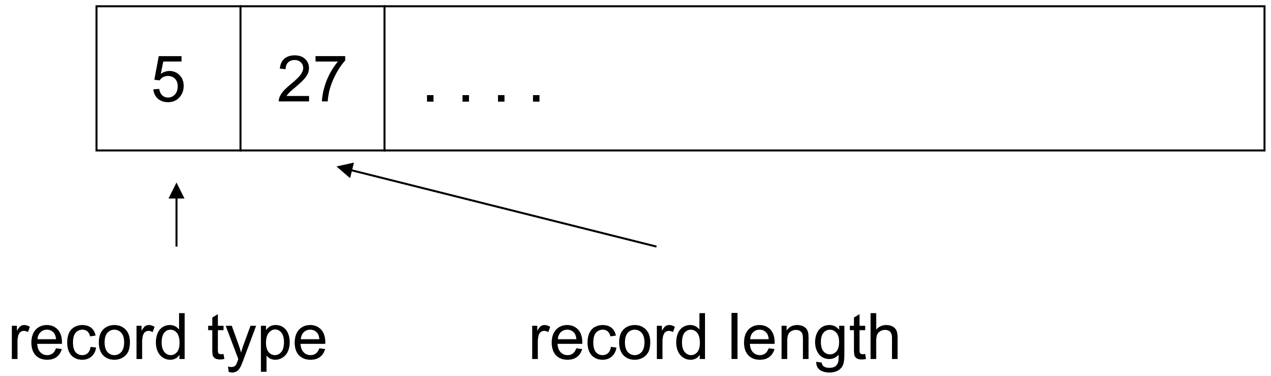
Note: Repeated Fields Does Not Imply Variable Format/Length

Could have fixed space for a max # of items and their sizes

John	Sailing	Chess	(null)

Many Variants Between Fixed and Variable Format

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



Type is a pointer to one of several schemas

Record Header: Data at Start that Describes a Record

May contain:

- record type
- record length
- timestamp
- concurrency stuff ...

Exercise: How to store XML data?

```
<table>
<description> people on the fourth floor <\description>
<people>
     <person>
          <name> Alan <\name>
          <age> 42 <\age>
          <email> agb@abc.com <\email>
     <\person>
     <person>
          <name> Sally <\name>
          <age> 30 <\age>
          <email> sally@abc.com <\email>
     <\person>
<\people>
<\table>
```

Source: Data on the Web, Abiteboul et al

Other Interesting Issues

Compression

- » Within record: e.g. encoding selection
- » Collection of records: use common patterns

Encryption

- » Usually operates on large blocks

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 filesystem or S3:

```
/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 data by date, then group by customer ID)

2) Interleaved orderings such as Z-ordering

Z-Ordering

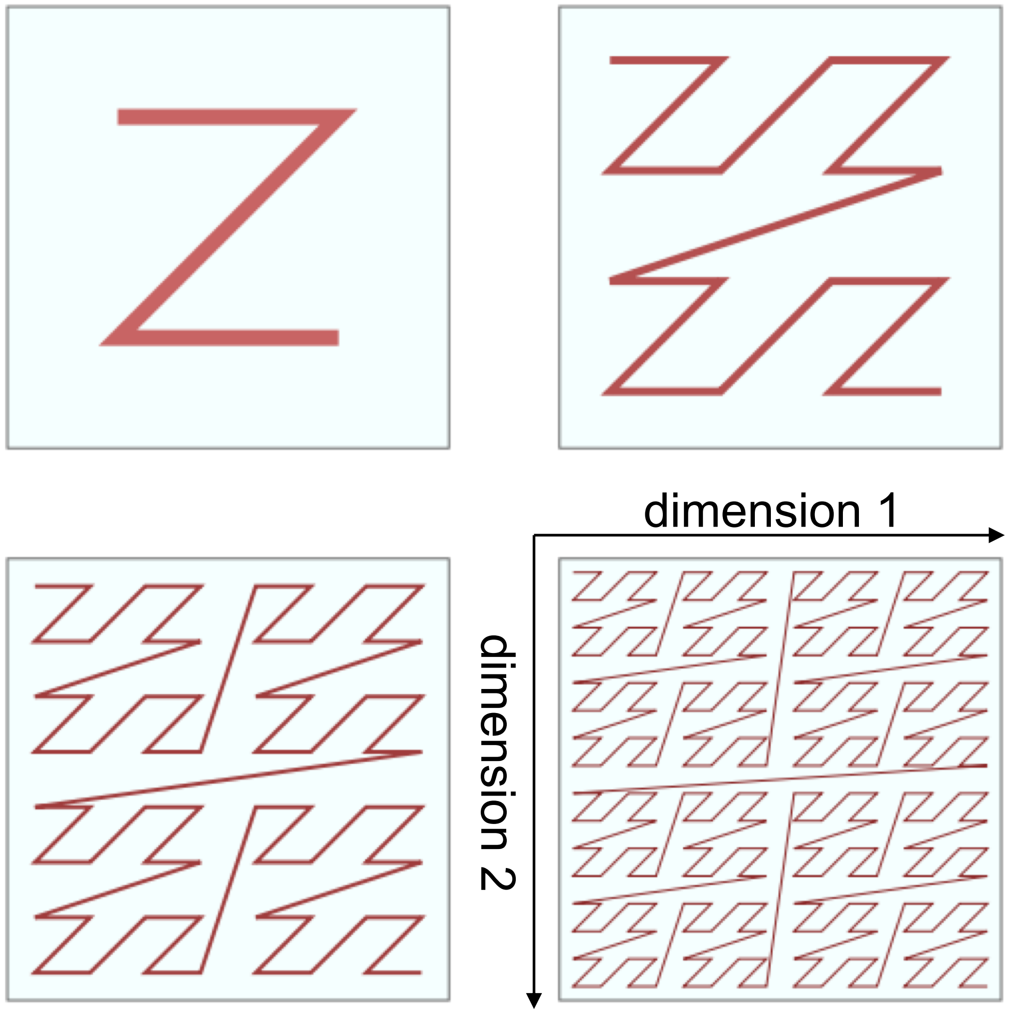
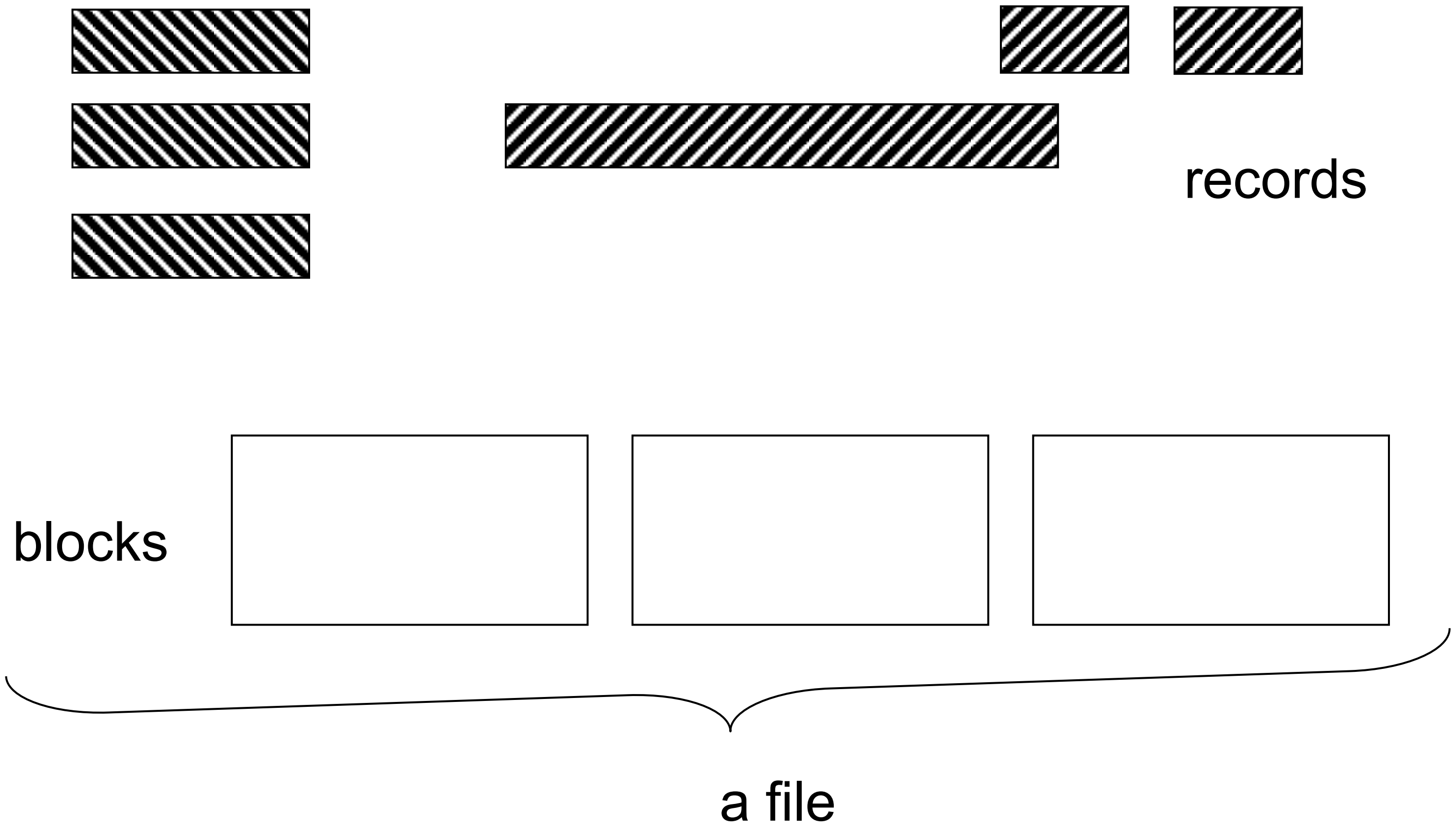


Image source: Wikipedia

How Do We Encode Records into Blocks & Files?

How Do We Encode Records into Blocks & Files?



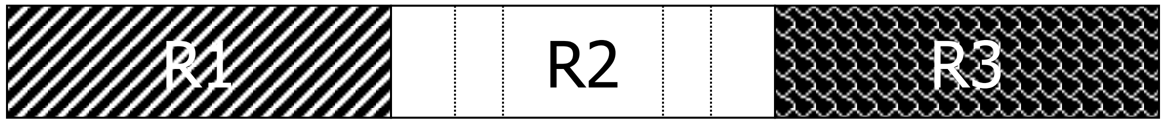
Questions in Storing Records

(1) separating records

(2) spanned vs. unspanned

(3) indirection

(1) Separating Records

Block 

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

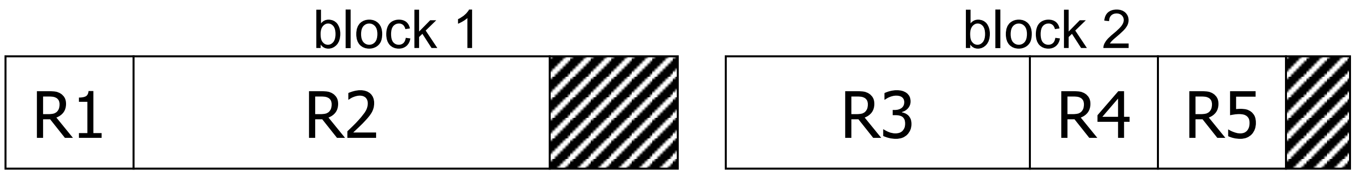
(b) special marker

(c) give record lengths (or offsets)

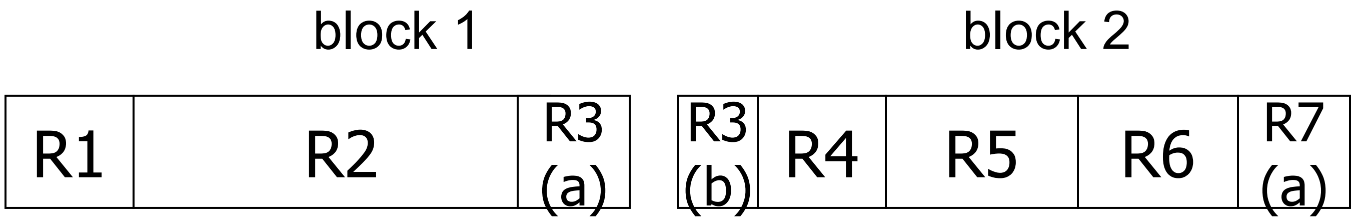
- within each record
- in block header

(2) Spanned vs Unspanned

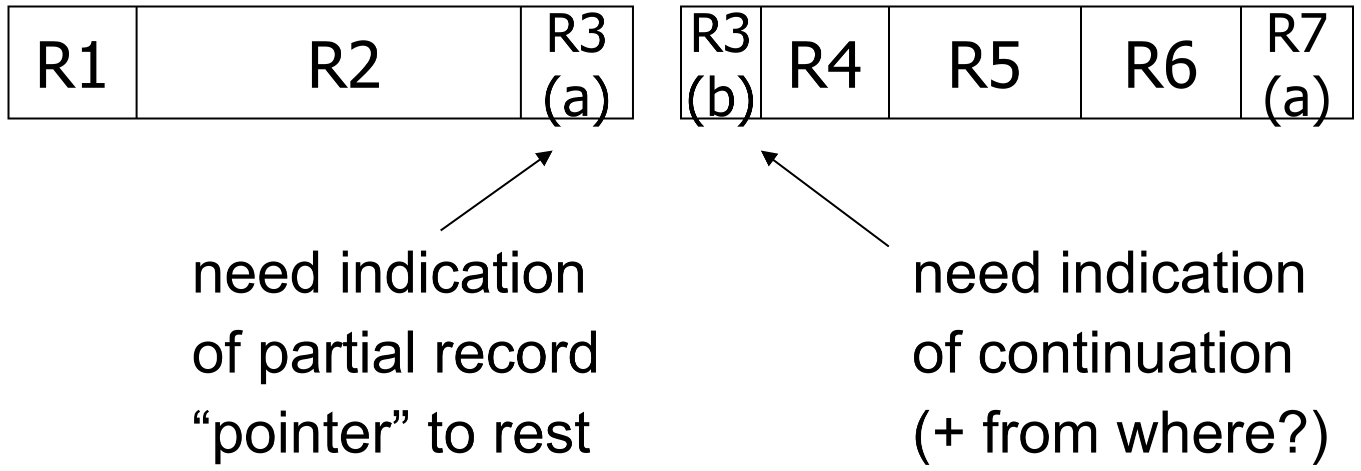
Unspanned: records must be within one block



Spanned:



With Spanned Records


R1
R2
R3 (a)
R3 (b)
R4
R5
R6
R7 (a)
need indication of partial record "pointer" to rest
need indication of continuation (+ from where?)

Spanned vs Unspanned

Unspanned is **much** simpler, but may waste storage space…

Spanned essential if record size > block size

(4) Indirection

How does one refer to specific records?
(e.g. in metadata or in other records)

(4) Indirection

How does one refer to records?

Many options:

Physical $\Longleftrightarrow$ Indirect

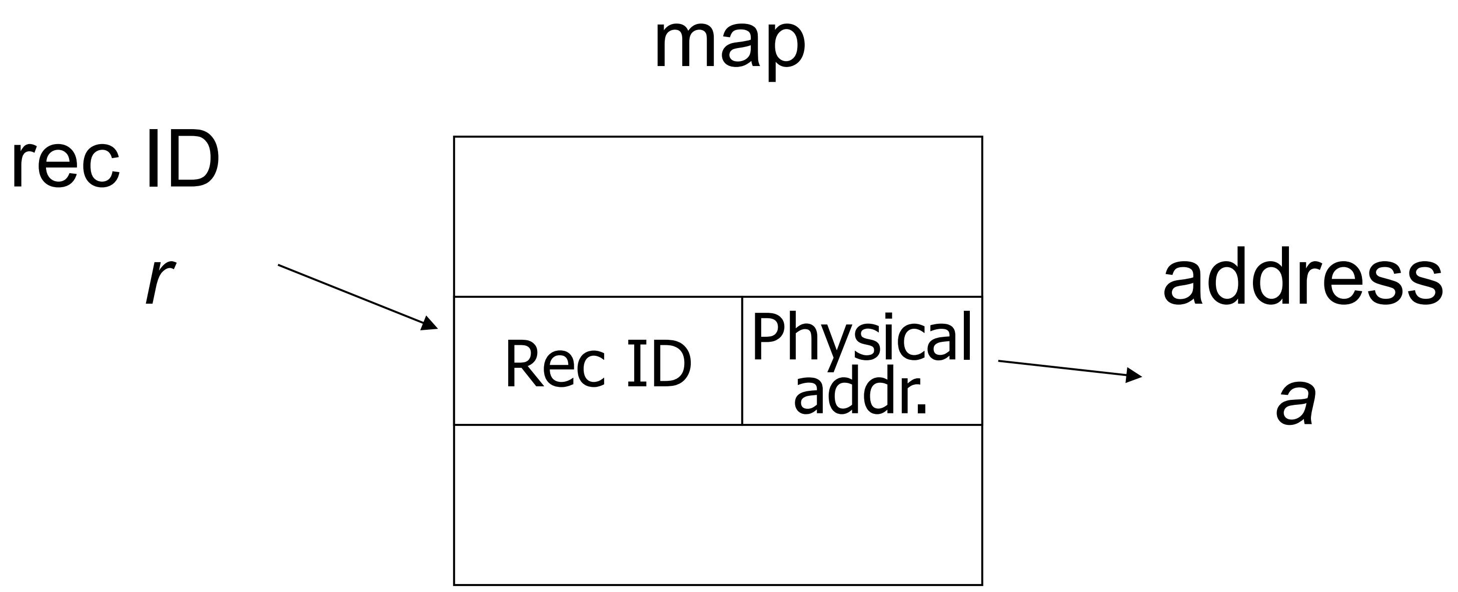
Purely Physical

E.g.,

$$\begin{array}{l}\text{Record}\\ \text{Address}\\ \text{or ID}\end{array} = \left\{ \begin{array}{l} \left.\begin{array}{l}\text{Device ID}\\ \text{Cylinder \#}\\ \text{Track \#}\\ \text{Block \#}\end{array}\right\} \text{Block ID} \\ \text{Offset in block}\end{array}\right.$$

Fully Indirect

E.g., Record ID is arbitrary bit string



Tradeoff

Flexibility $\longleftrightarrow$ Cost

to move records of indirection

(for deletions, insertions)

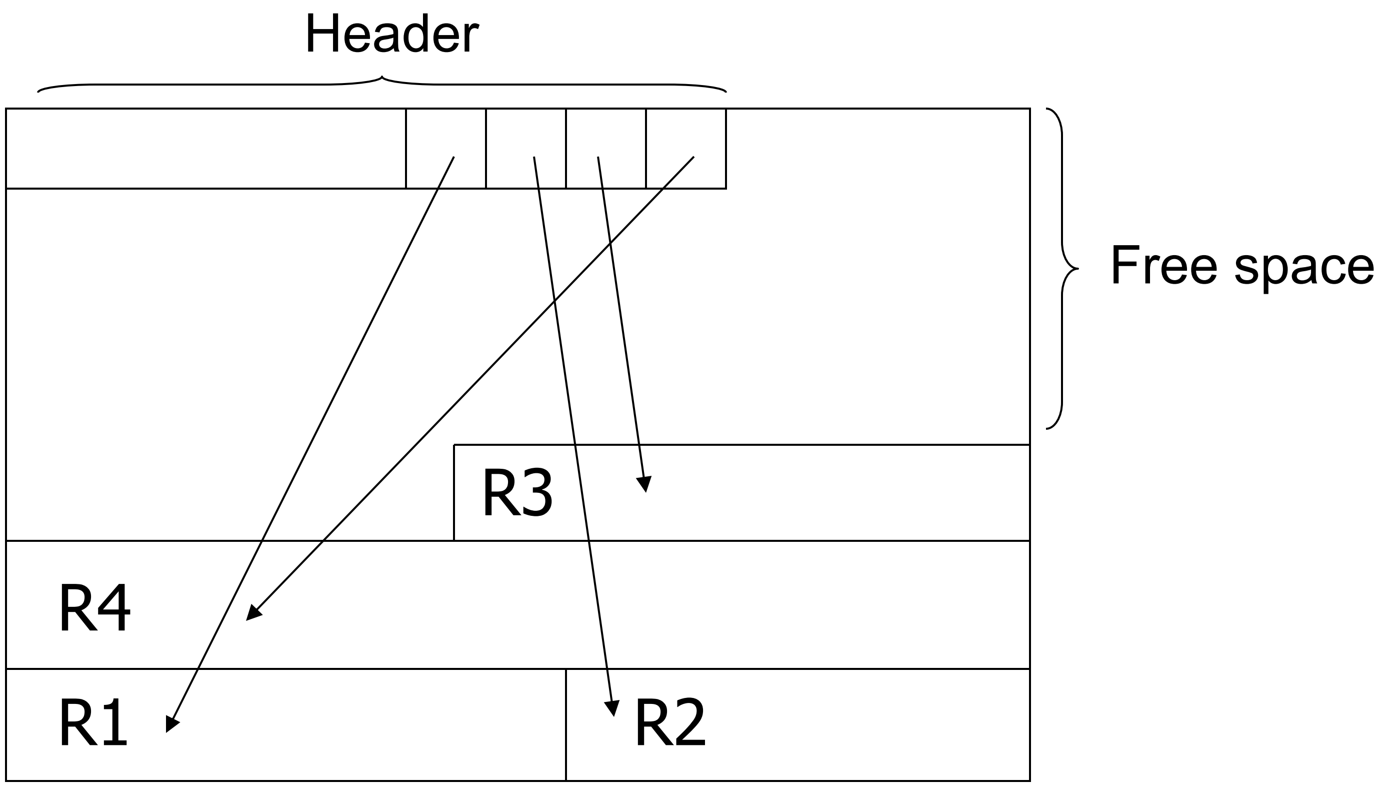
Physical $\Longleftrightarrow$ Indirect

$\uparrow$

Many options
in between …

Example: Indirection in Block

A block:

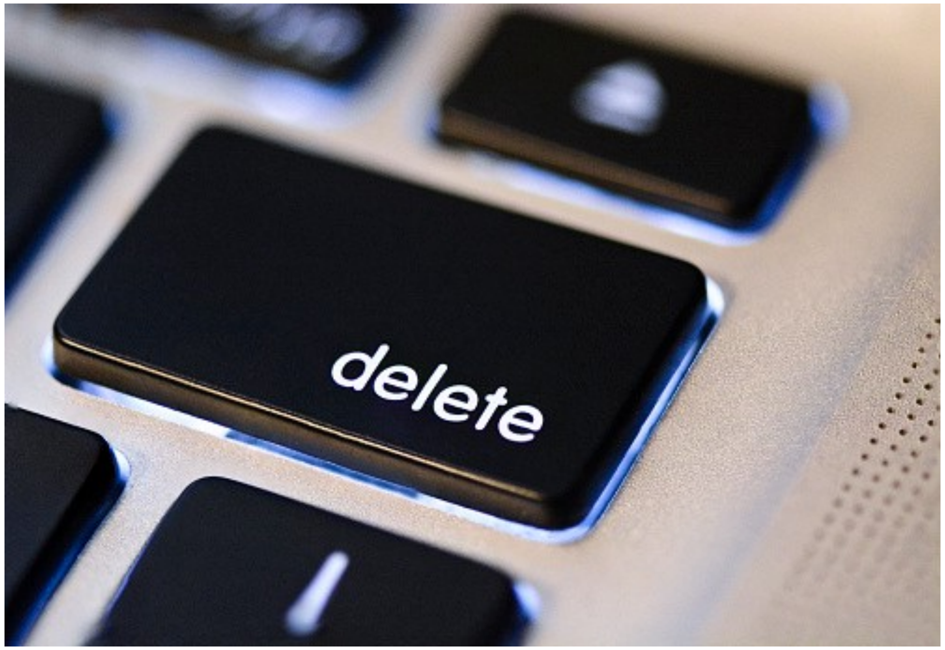


Block Header: Data at Start that Describes Block

May contain:

- File ID (or table or database ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4)
- Pointer to other blocks “like it”
- Timestamp ...

Other Concern: Deletion!



Options

(a) Immediately reclaim space

(b) Mark deleted

Options

(a) Immediately reclaim space

(b) Mark deleted

- May need chain of deleted records (for space re-use)
- Need a way to mark:
  - special characters
  - delete field
  - entries in maps

As Usual, Many Tradeoffs

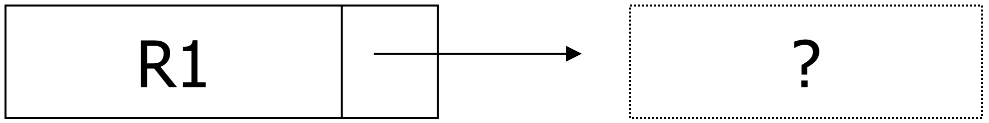
How expensive is to move valid record to free space for immediate reclaim?

How much space is wasted?

» e.g., deleted records, delete fields, free space chains,...

Concern with Deletions

Dangling pointers



Solution 1: Do Not Worry

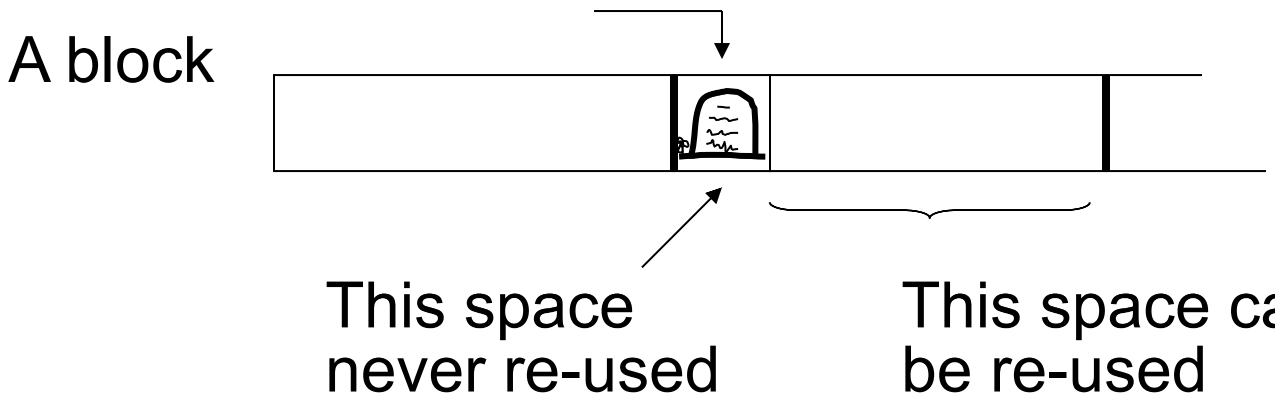
Solution 2: Tombstones

Special mark in old location or mappings

Solution 2: Tombstones

Special mark in old location or mappings

Physical IDs:



Solution 2: Tombstones

Special mark in old location or mappings

Logical IDs:

map

ID	LOC
7788	🪦

Never reuse
ID 7788 nor
space in map...

Insertion

Easy case: records not ordered

→ Insert new record at end of file or in a deleted slot

→ If records are variable size, not as easy...

Insertion

Hard case: records are ordered

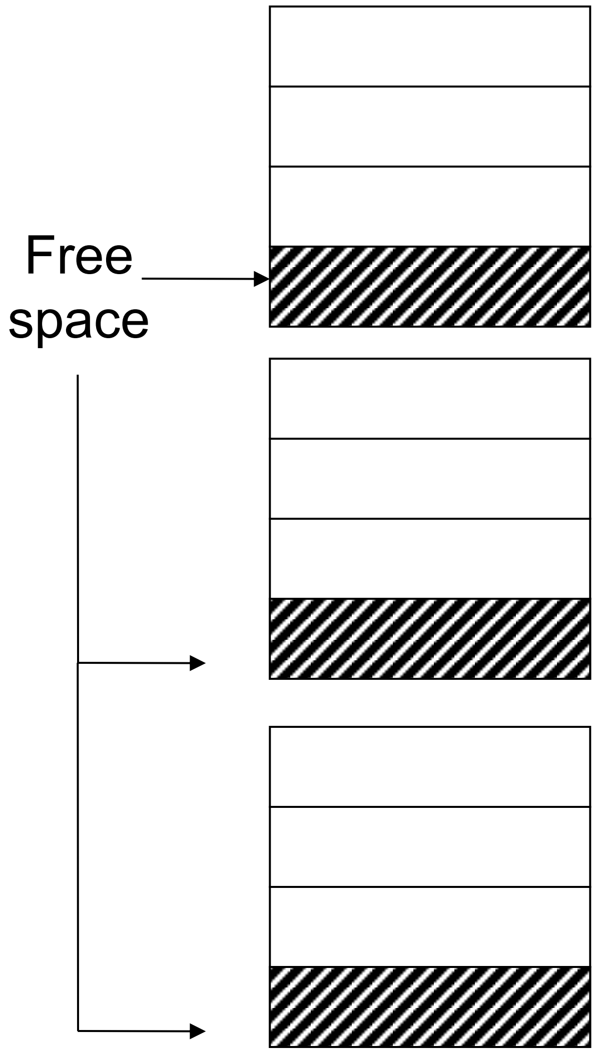
$\rightarrow$ If free space close by, not too bad...

$\rightarrow$ Otherwise, use an **overflow** area?

Interesting Problems

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

How often do I reorganize file + overflow?

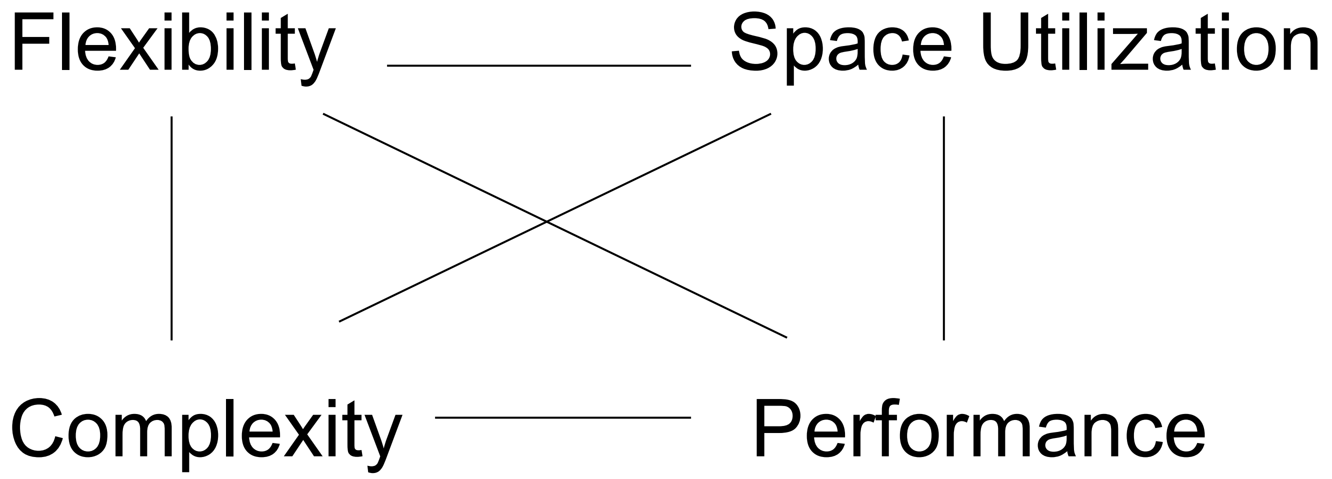


Summary

There are 10,000,000 ways to organize my data on disk…

Which is right for me?

Issues



To Evaluate a Strategy, Compute:

Space used for expected data

Expected time to

- fetch record given key
- fetch record with next key
- insert record
- append record
- delete record
- update record
- read all file
- reorganize file