

Ullman et al. :
Database System Principles

Disk Organization

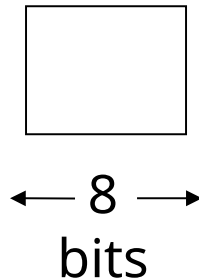
Topics for today

- How to lay out data on disk
- How to move it to memory

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

| | |
|----------|----------|
| 00000000 | 00100011 |
|----------|----------|

- Real, floating point
 n bits for mantissa, m for exponent....

To represent:

- Characters
→ various coding schemes suggested,
most popular is ascii

Example:

A: 1000001

a: 1100001

5: 0110101

LF: 0001010

To represent:

- Boolean

e.g., TRUE

| |
|-----------|
| 1111 1111 |
|-----------|

FALSE

| |
|-----------|
| 0000 0000 |
|-----------|

- Application specific

e.g., RED → 1 GREEN → 3

BLUE → 2 YELLOW → 4 ...

⇒ 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

(not YYMMDD! Why?)

- Time

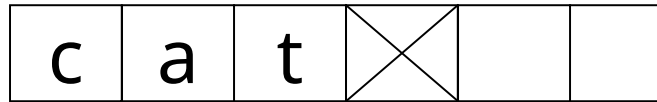
e.g. - Integer, seconds since midnight

- characters, HHMMSSFF

To represent:

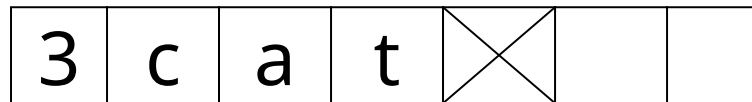
- String of characters
 - Null terminated

e.g.,



- Length given

e.g.,



- Fixed length

To represent:

- Bag of bits

| | |
|--------|------|
| Length | Bits |
|--------|------|

Key Point

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

Also

- Type of an item: Tells us how to interpret
(plus size if fixed)

Overview

w

Data Items



Records



Blocks



Files



Memory

Record - Collection of related data items (called 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 information

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

Example: fixed format and length

Employee record

- (1) E#, 2 byte integer
- (2) E.name, 10 char.
- (3) Dept, 2 byte code

Schema

| | | |
|----|-----------|----|
| 55 | s m i t h | 02 |
|----|-----------|----|

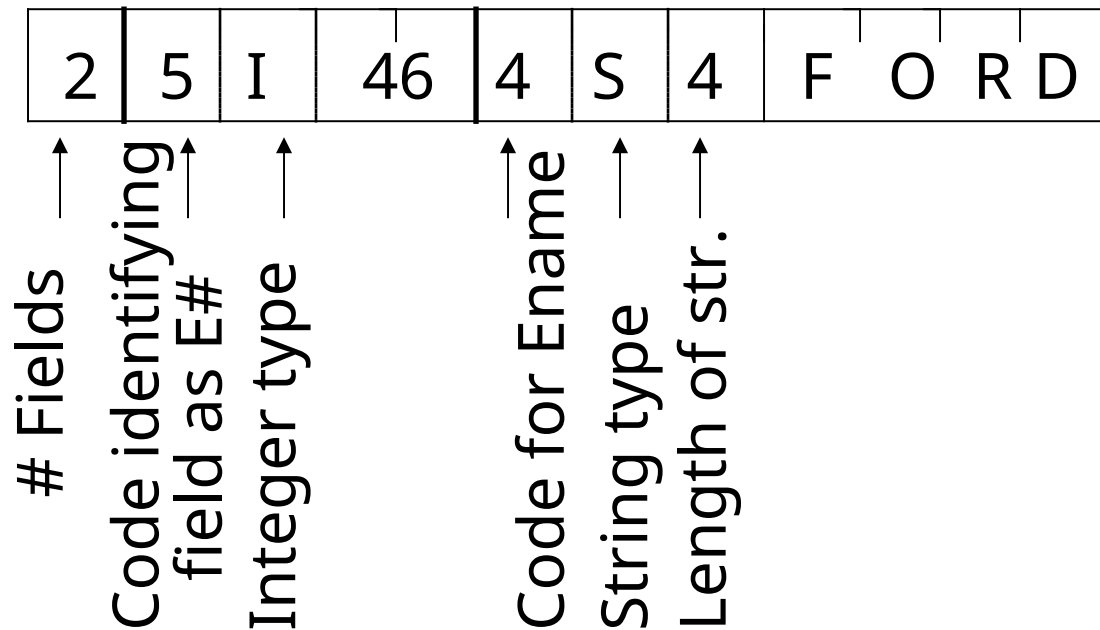
| | | |
|----|-----------|----|
| 83 | j o n e s | 01 |
|----|-----------|----|

Records

Variable format

- Record itself contains format
“Self Describing”

Example: variable format and 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: var format record with
repeating fields
Employee → one or more → children

| | | | |
|---|--------------|--------------|------------|
| 3 | E_name: Fred | Child: Sally | Child: Tom |
|---|--------------|--------------|------------|

Note: Repeating fields does not imply

- variable format, nor
- variable size

| | | | |
|------|---------|-------|----|
| John | Sailing | Chess | -- |
|------|---------|-------|----|

- Key is to allocate maximum number of repeating fields (if not used → null)

☆ Many variants between
fixed - variable format:

Example: Include record type in record



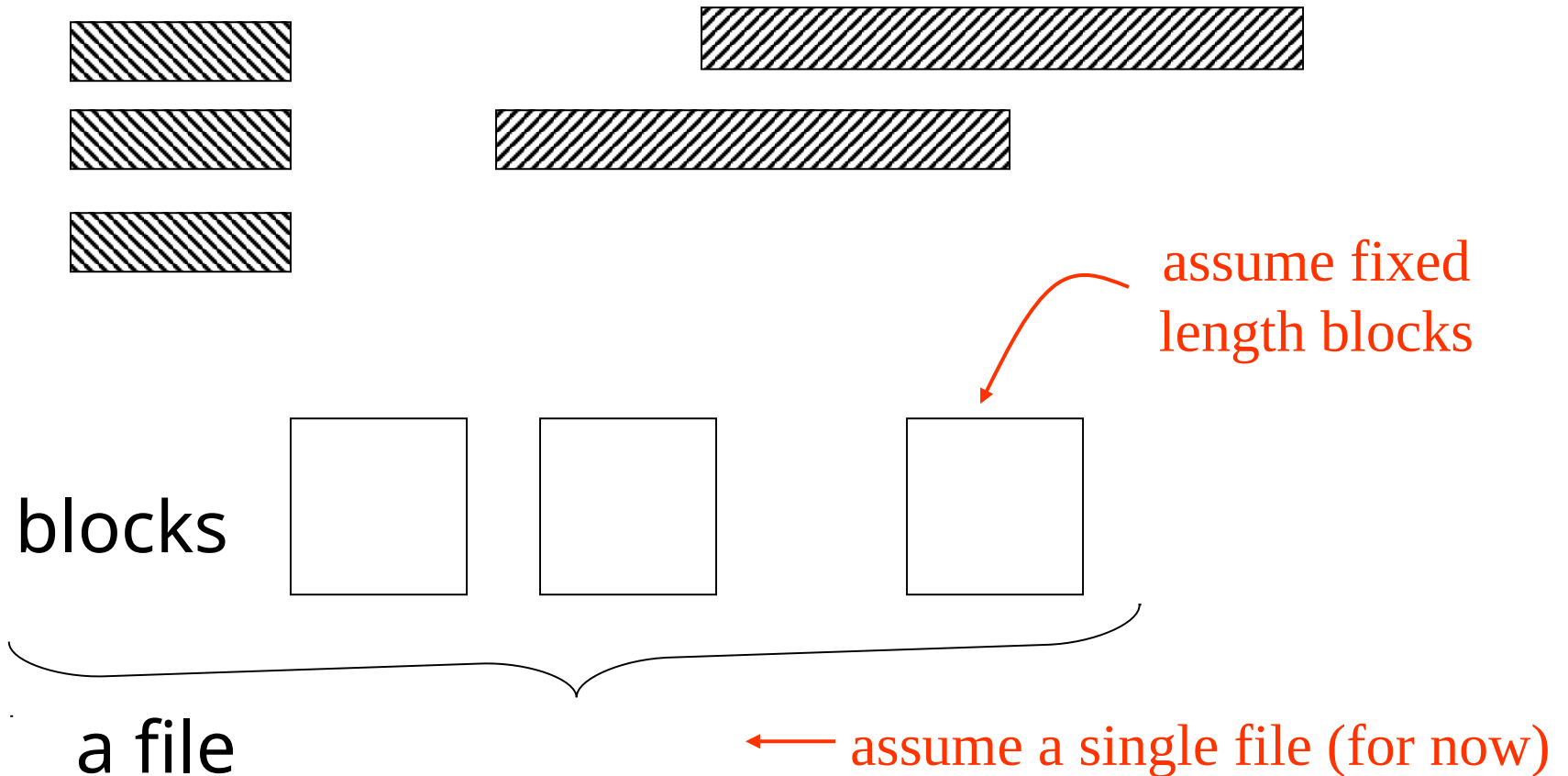
↑
record type record length
tells me what
to expect
(i.e. points to schema)

Record header - data at beginning
that describes record

May contain:

- record type
- record length
- time stamp
- other stuff ...

Next: placing records into blocks



Options for storing records in blocks:

- (1) separating records
- (2) spanned vs. unspanned
- (3) sequencing
- (4) indirection

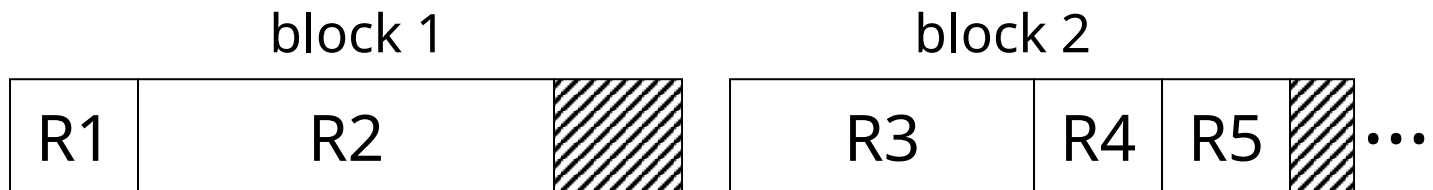
(1) Separating records



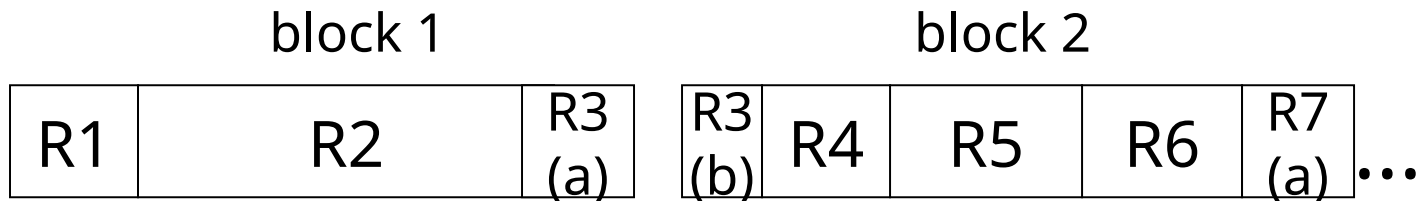
- (a) no need to separate - fixed size recs.
- (b) special marker
- (c) give record lengths (or offsets)
 - within each record
 - in block header

(2) Spanned vs. Unspanned

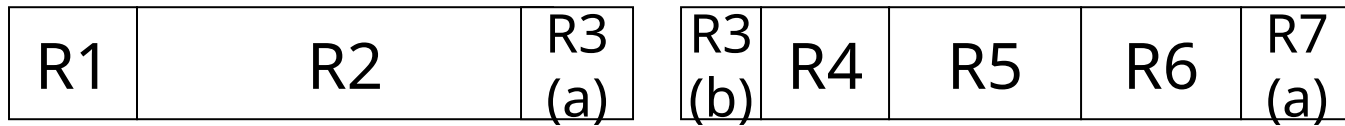
- Unspanned: records must be within one block



- Spanned



With spanned records:



need indication
of partial record
“pointer” to rest

need indication
of continuation
(+ from where?)

Spanned vs. unspanned:

- Unspanned is much simpler, but may waste space...
- Spanned essential if
record size > block size

(3) Sequencing

- Ordering records in file (and block) by some key value

Sequential file (\Rightarrow sequenced)

Why sequencing?

Typically to make it possible to efficiently
read records in order

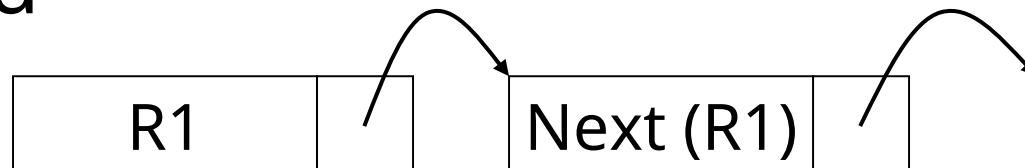
(e.g., to do a merge-join — discussed later)

Sequencing Options

(a) Next record physically contiguous



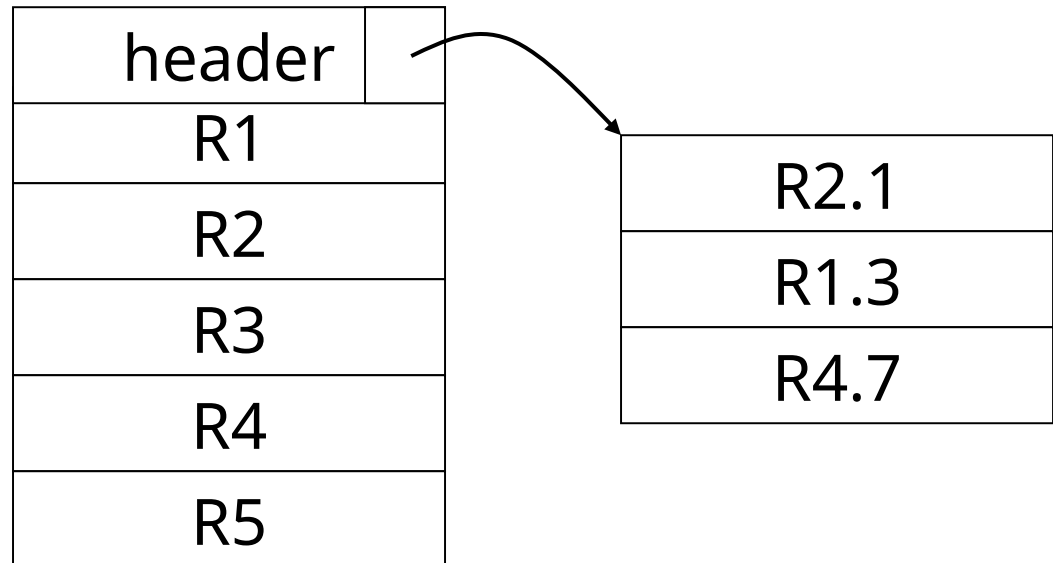
(b) Linked



Sequencing Options

(c) Overflow area

Records
in sequence



(4) Indirection

- How does one refer to records?



(4) Indirection

- How does one refer to records?



Many options:

Physical



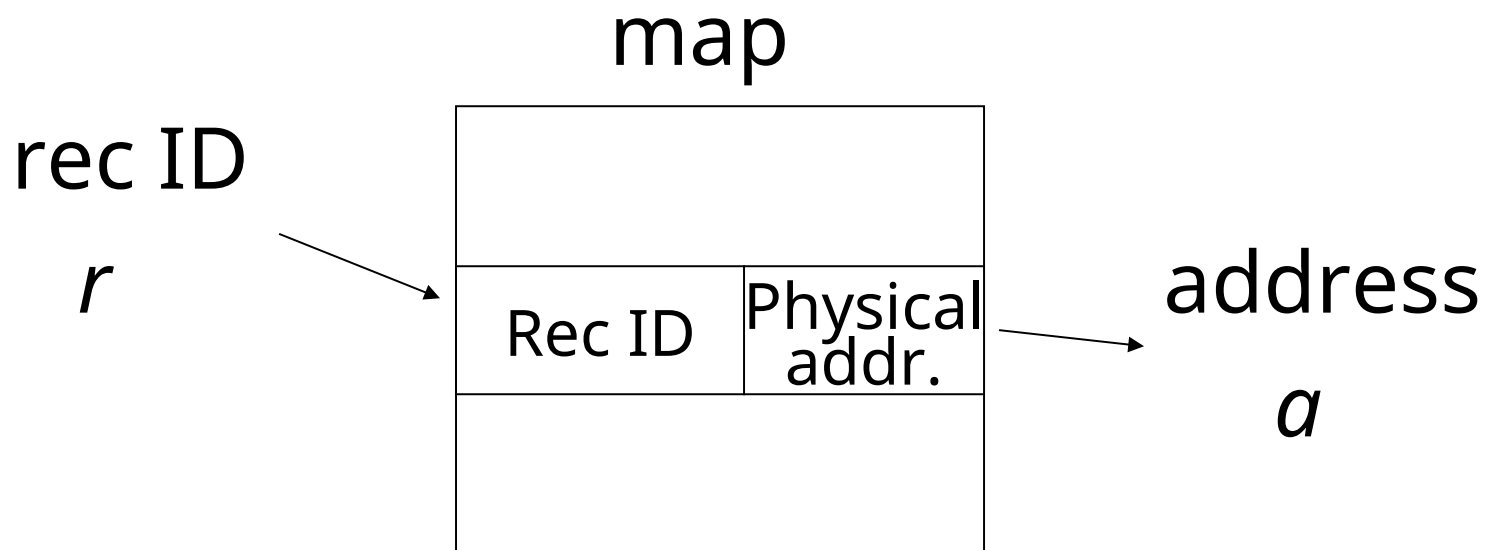
Indirect

☆ Purely Physical

E.g., Record
Address = {
 Device ID
 Cylinder #
 Track #
 Block #
 Offset in block
 } Block ID
or ID

☆ Fully Indirect

E.g., Record ID is arbitrary bit string



Tradeoff

Flexibility \longleftrightarrow Cost

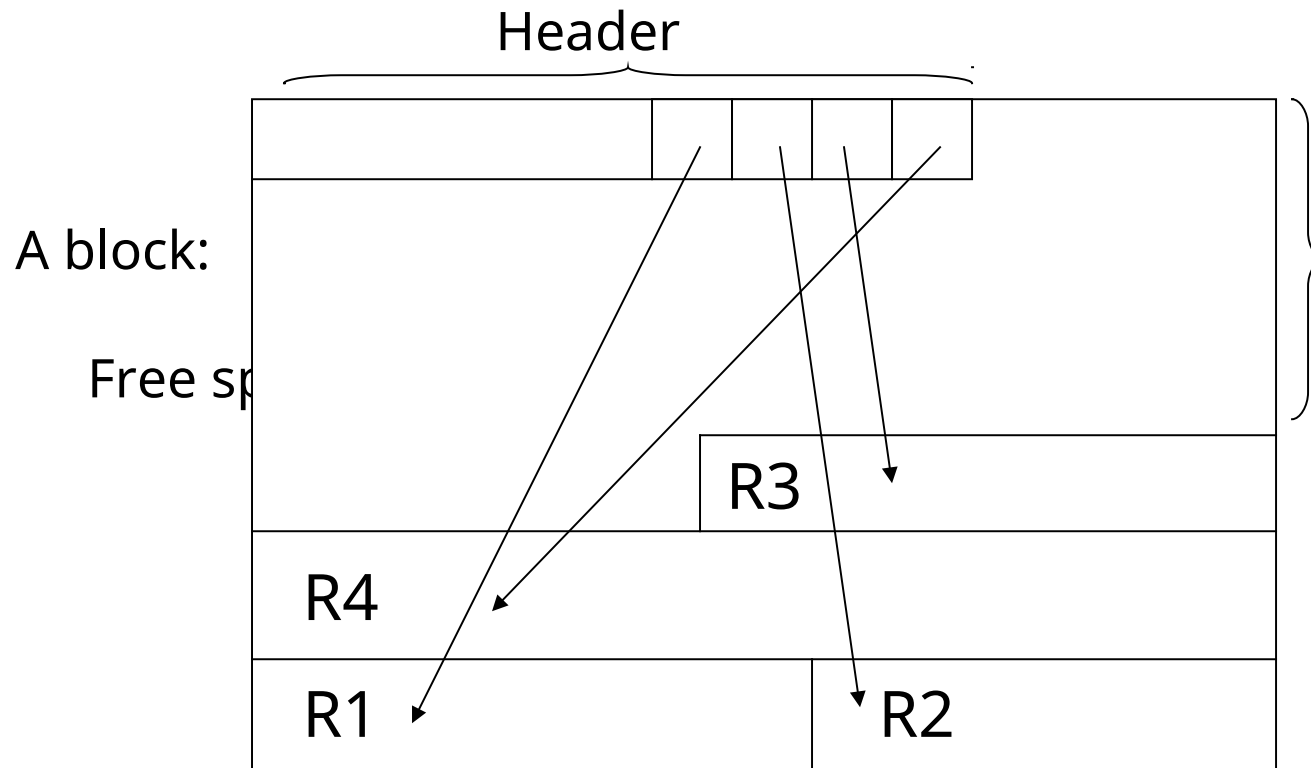
to move records of indirection
(for deletions, insertions)

Physical \longleftrightarrow Indirect



Many options
in between ...

Example: Indirection in block



Block header - data at beginning
that
describes block

May contain:

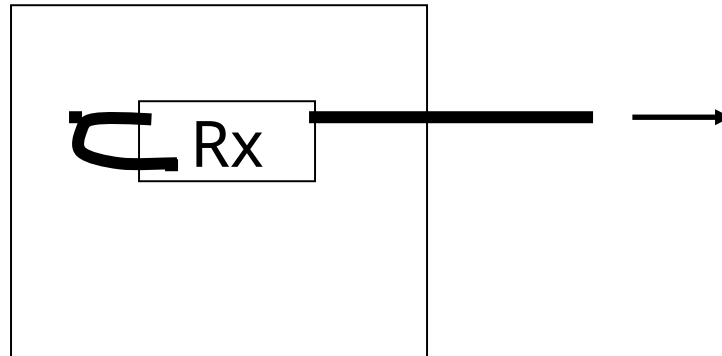
- File ID (or RELATION or DB ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4;
is overflow, ...)
- Pointer to other blocks "like it"
- Timestamp ...

Other Topics

- (1) Insertion/Deletion
- (2) Buffer Management
- (3) Comparison of Schemes

Deletion

Block



Options:

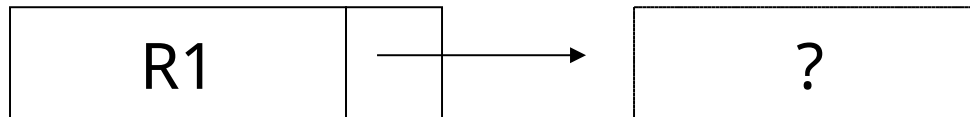
- (a) Immediately reclaim space
- (b) Mark deleted
 - May need chain of deleted records
(for re-use)
 - Need a way to mark:
 - special characters
 - delete field
 - in map

☆ 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

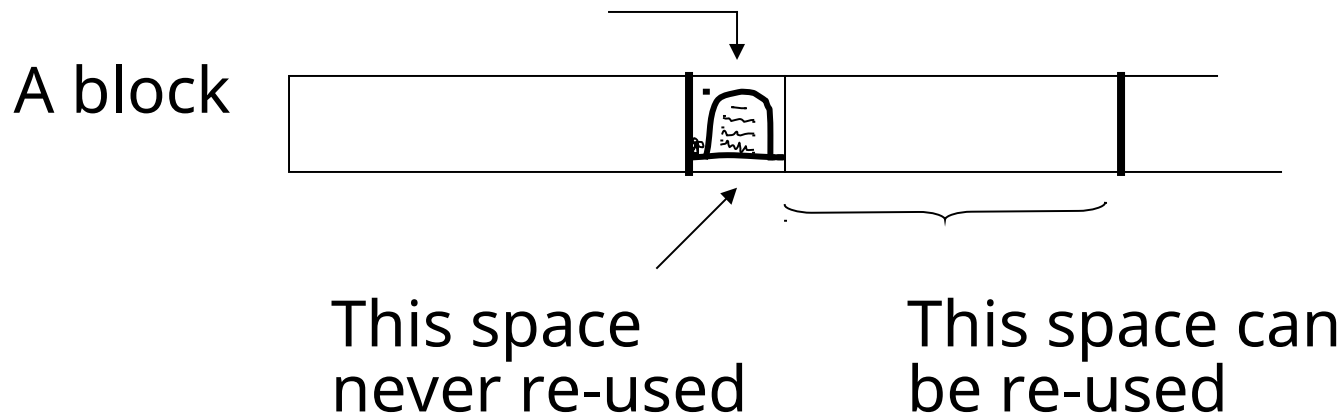


Solution #1: Do not worry

Solution #2: Tombstones

E.g., Leave “MARK” in map or old location

- Physical IDs




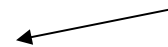
Solution #2: Tombstones

E.g., Leave “MARK” in map or old location

- Logical IDs

map

| ID | LOC |
|------|--|
| | |
| 7788 |  |
| | |



Never reuse
ID 7788 nor
space in map...

Insert

Easy case: records not in sequence

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

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

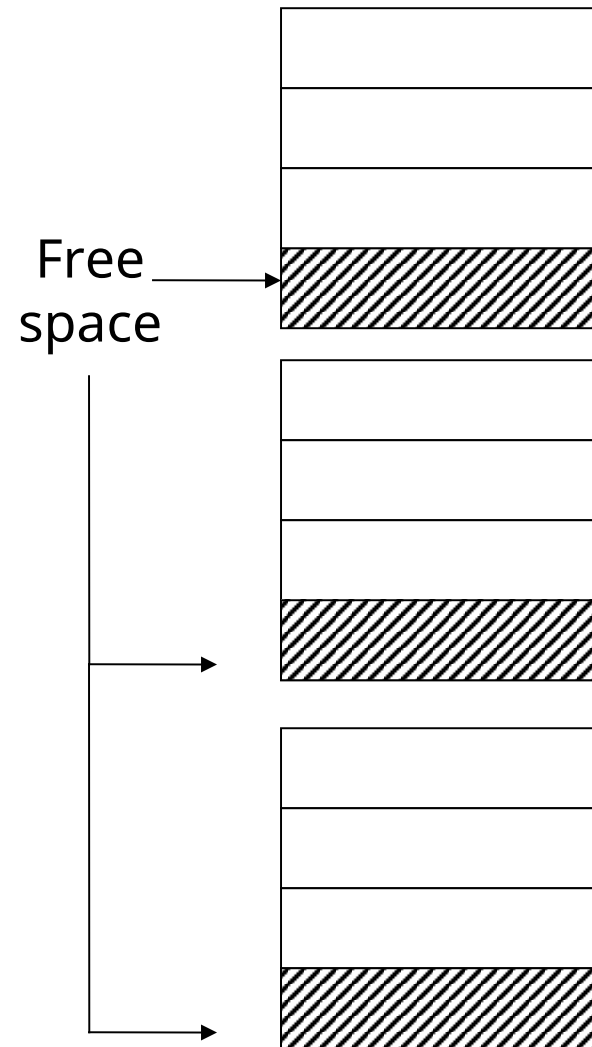
Insert

Hard case: records in sequence

- If free space “close by”, not too bad...
- Or use overflow idea...

Interesting problems:

- How much free space to leave in each block, track, cylinder?
- How often do I reorganize file + overflow?



Buffer Management

- DB features needed
- Why LRU may be bad
- Pinned blocks
- Forced output
- Double buffering

Row vs Column Store

- So far we assumed that fields of a record are stored contiguously (row store)...
- Another option is to store like fields together (column store)

Row Store

- Example: Order consists of
 - id, cust, prod, store, price, date, qty

| | | | | | | |
|-----|-------|-------|--------|--------|-------|------|
| id1 | cust1 | prod1 | store1 | price1 | date1 | qty1 |
|-----|-------|-------|--------|--------|-------|------|

| | | | | | | |
|-----|-------|-------|--------|--------|-------|------|
| id2 | cust2 | prod2 | store2 | price2 | date2 | qty2 |
|-----|-------|-------|--------|--------|-------|------|

| | | | | | | |
|-----|-------|-------|--------|--------|-------|------|
| id3 | cust3 | prod3 | store3 | price3 | date3 | qty3 |
|-----|-------|-------|--------|--------|-------|------|

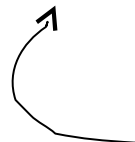
Column Store

- Example: Order consists of
 - id, cust, prod, store, price, date, qty

| | |
|-----|-------|
| id1 | cust1 |
| id2 | cust2 |
| id3 | cust3 |
| id4 | cust4 |
| ... | ... |
| | |

| | |
|-----|-------|
| id1 | prod1 |
| id2 | prod2 |
| id3 | prod3 |
| id4 | prod4 |
| ... | ... |
| | |

| | | |
|-----|--------|------|
| id1 | price1 | qty1 |
| id2 | price2 | qty2 |
| id3 | price3 | qty3 |
| id4 | price4 | qty4 |
| ... | ... | ... |
| | | |

ids may or may not be stored explicitly

Row vs Column Store

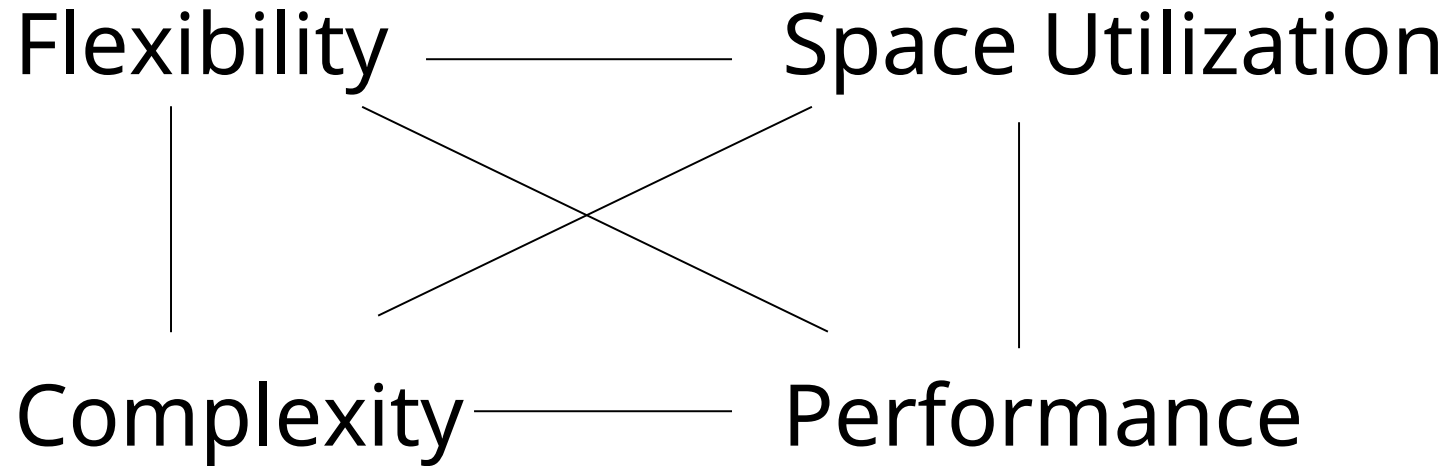
- Advantages of Column Store
 - more compact storage (fields need not start at byte boundaries)
 - efficient reads on data mining operations
- Advantages of Row Store
 - writes (multiple fields of one record) more efficient
 - efficient reads for record access (OLTP)

Comparison

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

Which is right for me?

Issues:



☆ To evaluate a given strategy, compute following parameters:

-> 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

Summary

- How to lay out data on disk

Data Items

Records



Blocks



Files



Memory



DBMS





Next

How to find a record quickly,
given a key