Database System Principles

chapter 3: Record storage

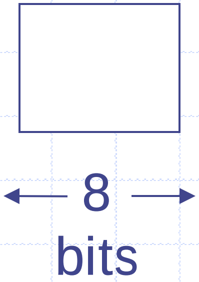
Topics for today

- 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



8

bits

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

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?

To represent:

- Boolean

  e.g., TRUE

  FALSE

1111 1111

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, 1970

  - 8 characters, YYYYMMDD
  - 7 characters,

    YYYYDDD

    (not YYMMDD! Why?)

- Time

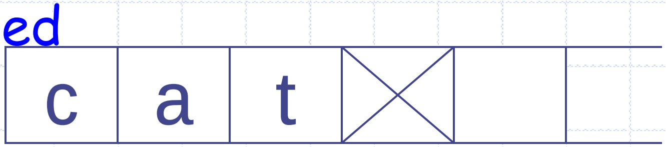
  e.g. - Integer, seconds since midnight

  - characters, HHMMSS

To represent:

- String of characters
  - Null terminated

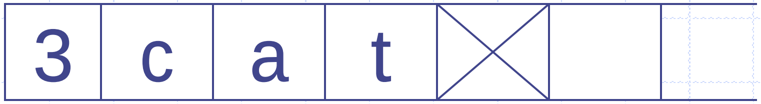
    e.g.,



c a t

  - Length given

    e.g.,



3 c a t

- Fixed length

To represent:

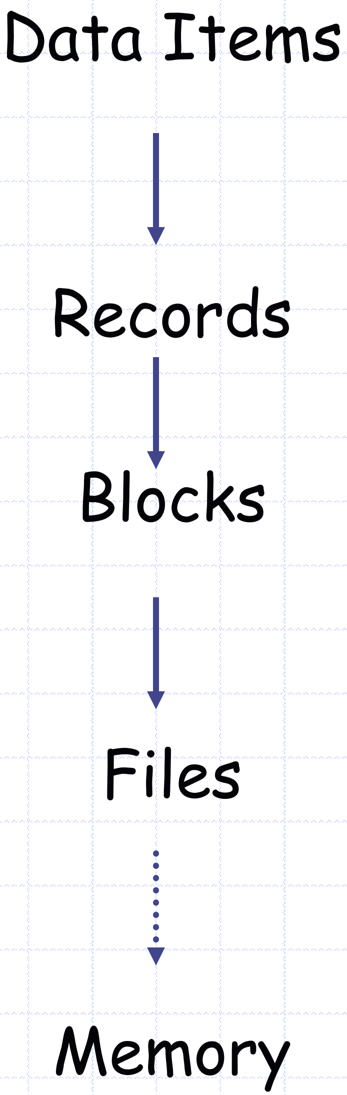
- Bag of bits

Length	Bits

Key Point

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

Overview



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

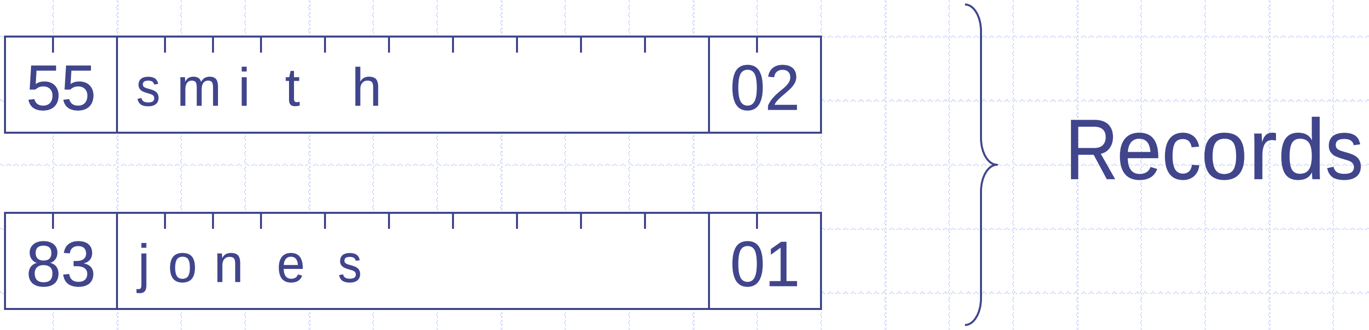
Employee record

(1) E#, 2 byte integer

(2) E.name, 10 char.

(3) Dept, 2 byte code

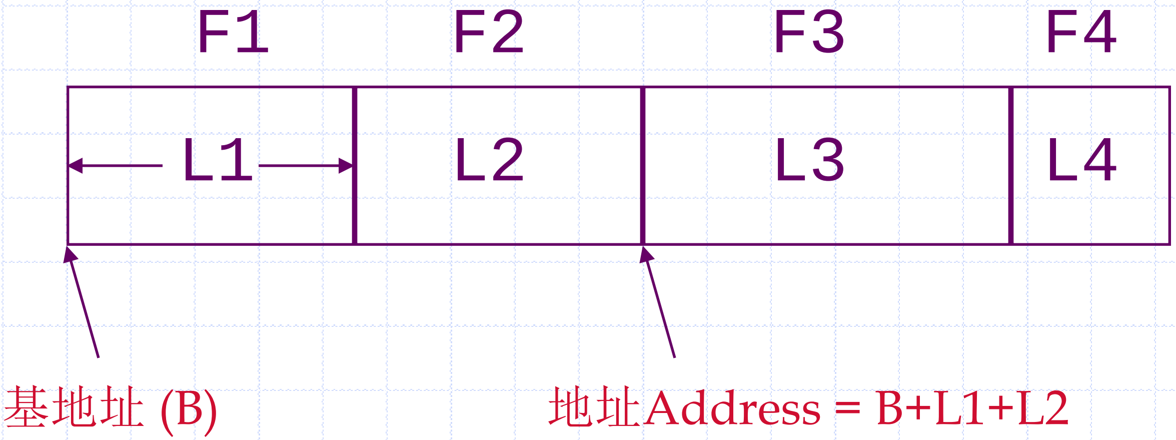
Schema



55	s m i t h	02
83	j o n e s	01

Records

记录格式：定长记录

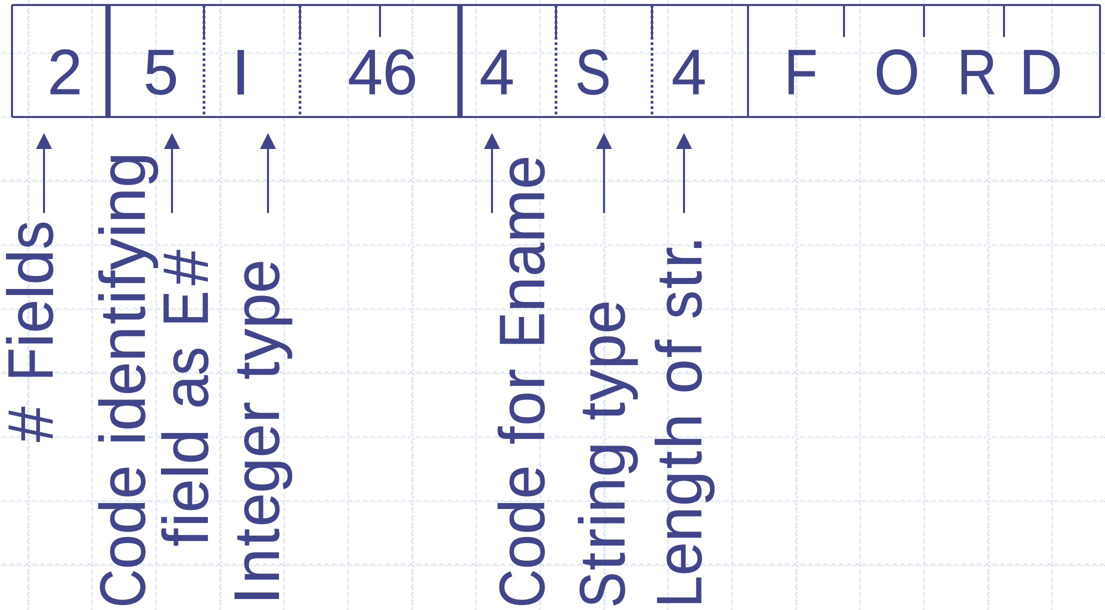


- 通过扫描记录，可以查找第i个字段

Variable format

- Record itself contains format
  “Self Describing”

Example: variable format and length



2	5	I	46	4	S	4	F	O	R	D

Fields

Code identifying field as E#

Integer type

Code for Ename

String type

Length of str.

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

Variable format useful for:

- “sparse” records
- repeating fields
- evolving formats

- EXAMPLE: var format record with repeating fields

Employee → one or more → children

3	E_name: Fred	Child: Sally	Child: Tom

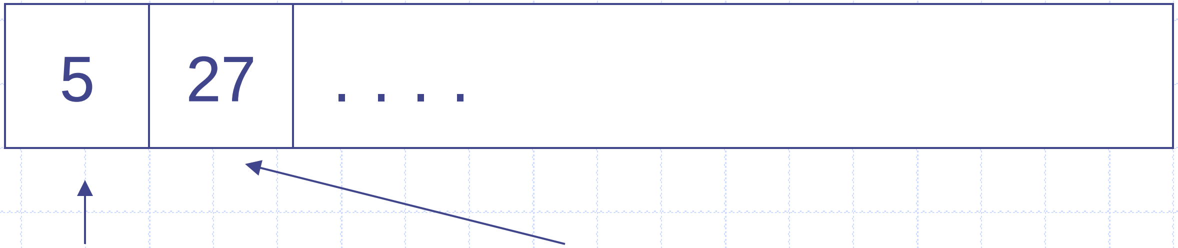
Note: Repeating fields does not imply

- variable format, nor
- variable size

John	Sailing	Chess	--

☆ Many variants between fixed - variable format:

<u>Ex. #1:</u> Include <u>record type</u> in record



5

27

. . . . .

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

record length

Record header - data at beginning that describes record

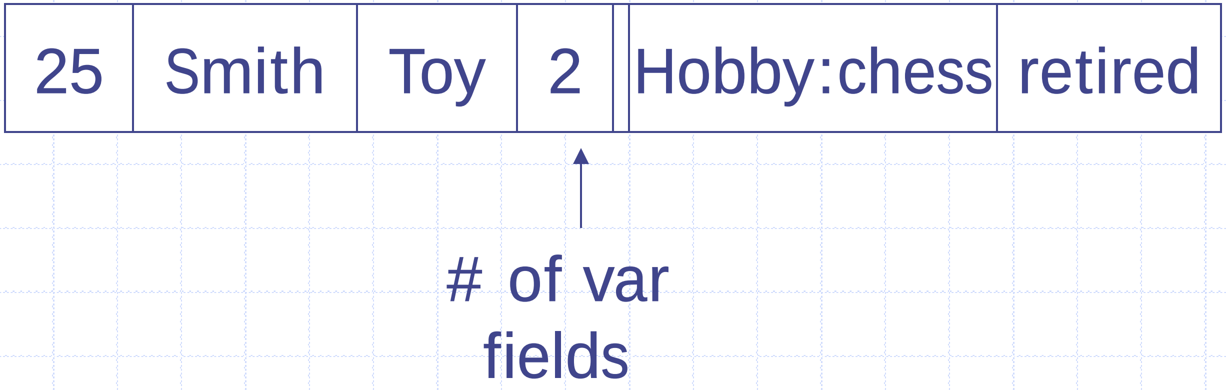
May contain:

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

Ex #2 of variant between FIXED/VAR format

- Hybrid format
  - one part is fixed, other variable

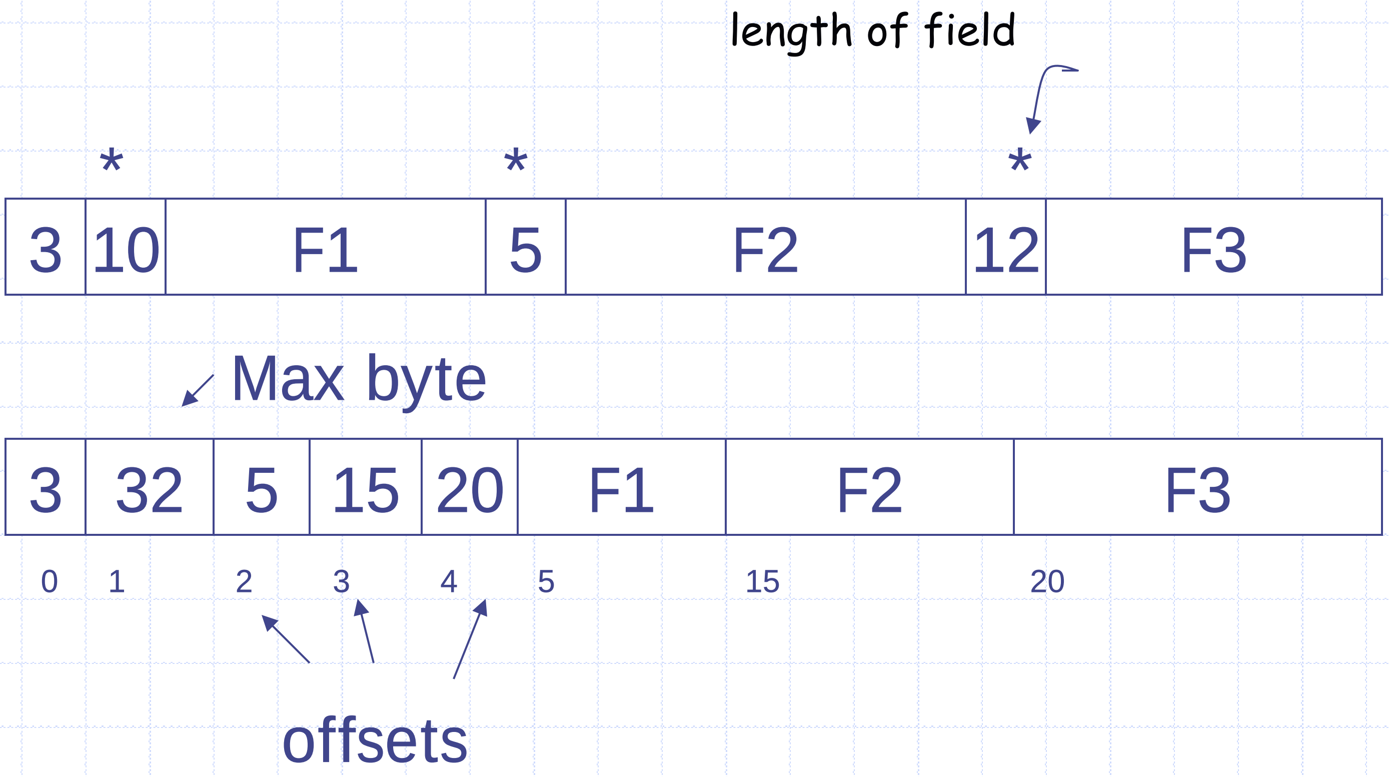
E.g.: All employees have E#, name, dept
other fields vary.



25	Smith	Toy	2	Hobby:chess	retired

# of var fields

Also, many variations in internal organization of record



length of field

	*		*		*	
3	10	F1	5	F2	12	F3

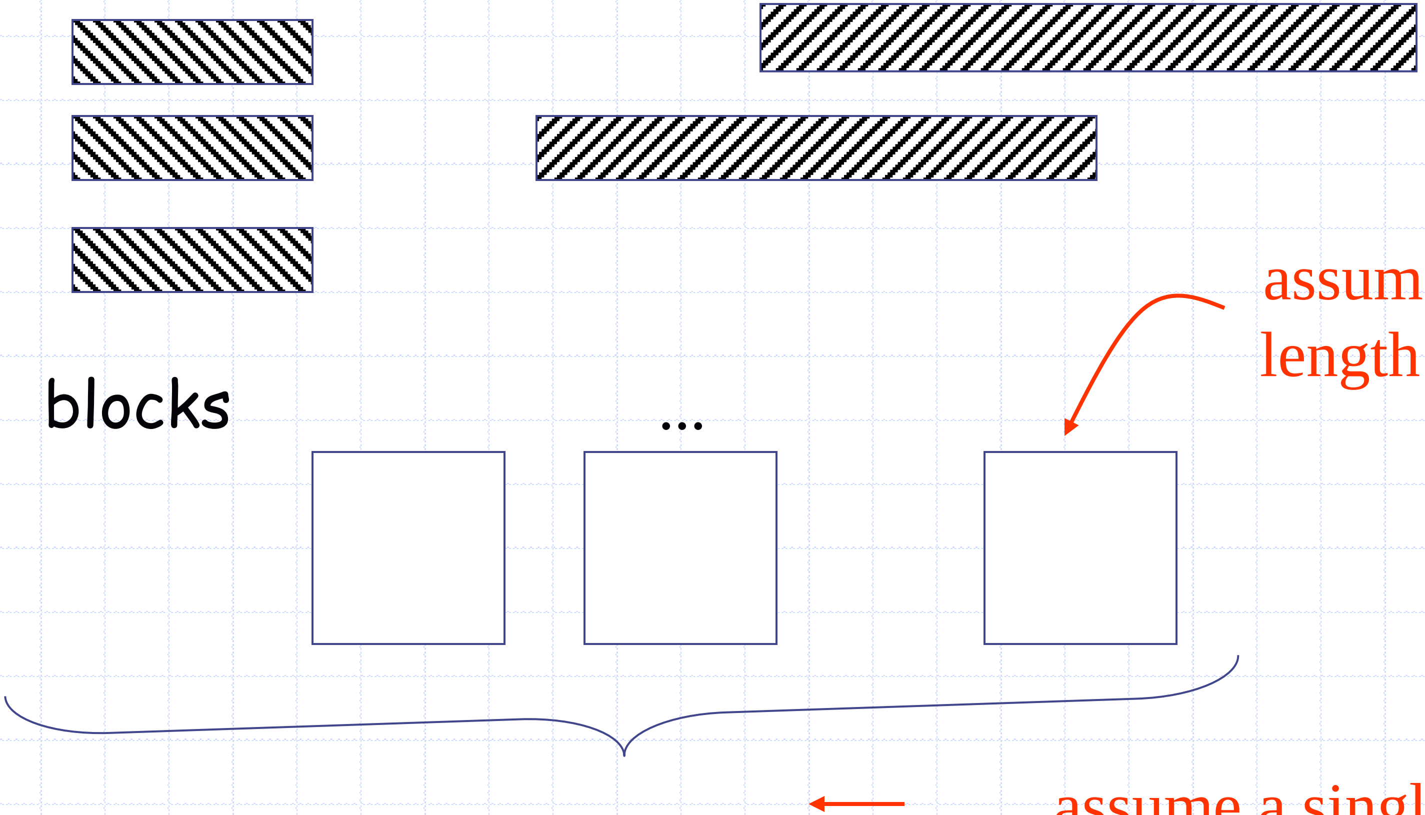
Max byte

3	32	5	15	20	F1	F2	F3
0	1	2	3	4	5	15	20

offsets

Until now, we have introduced fields storage

Next: placing records into blocks



blocks

...

assume fixed length blocks

assume a single file

Options for storing records in blocks:

(1) separating records

(2) spanned （跨区记录） vs. unspanned

(3) mixed record types – clustering

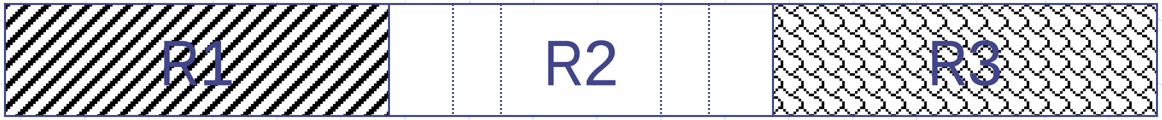
(4) split records

(5) sequencing

(6) indirection

(1) Separating records

Block



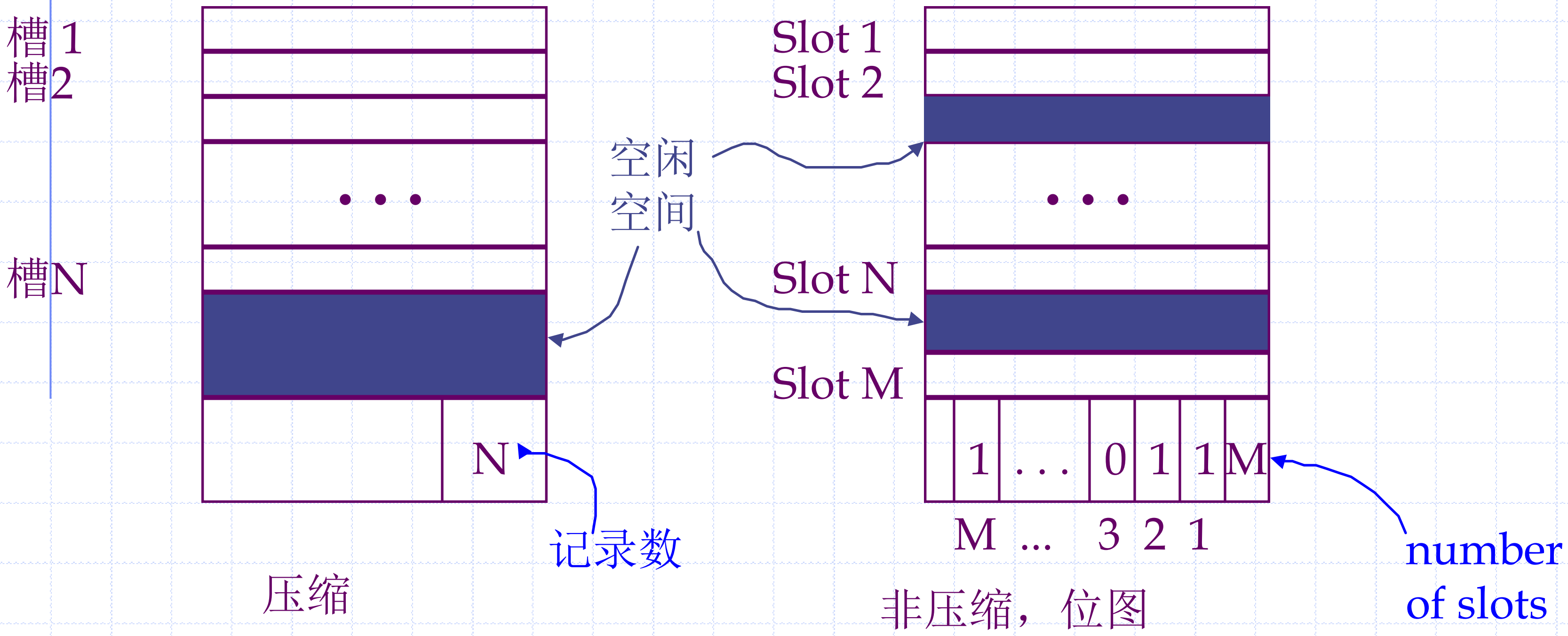
(a) no need to separate - fixed size recs.

(b) special marker

(c) give record lengths (or offsets)

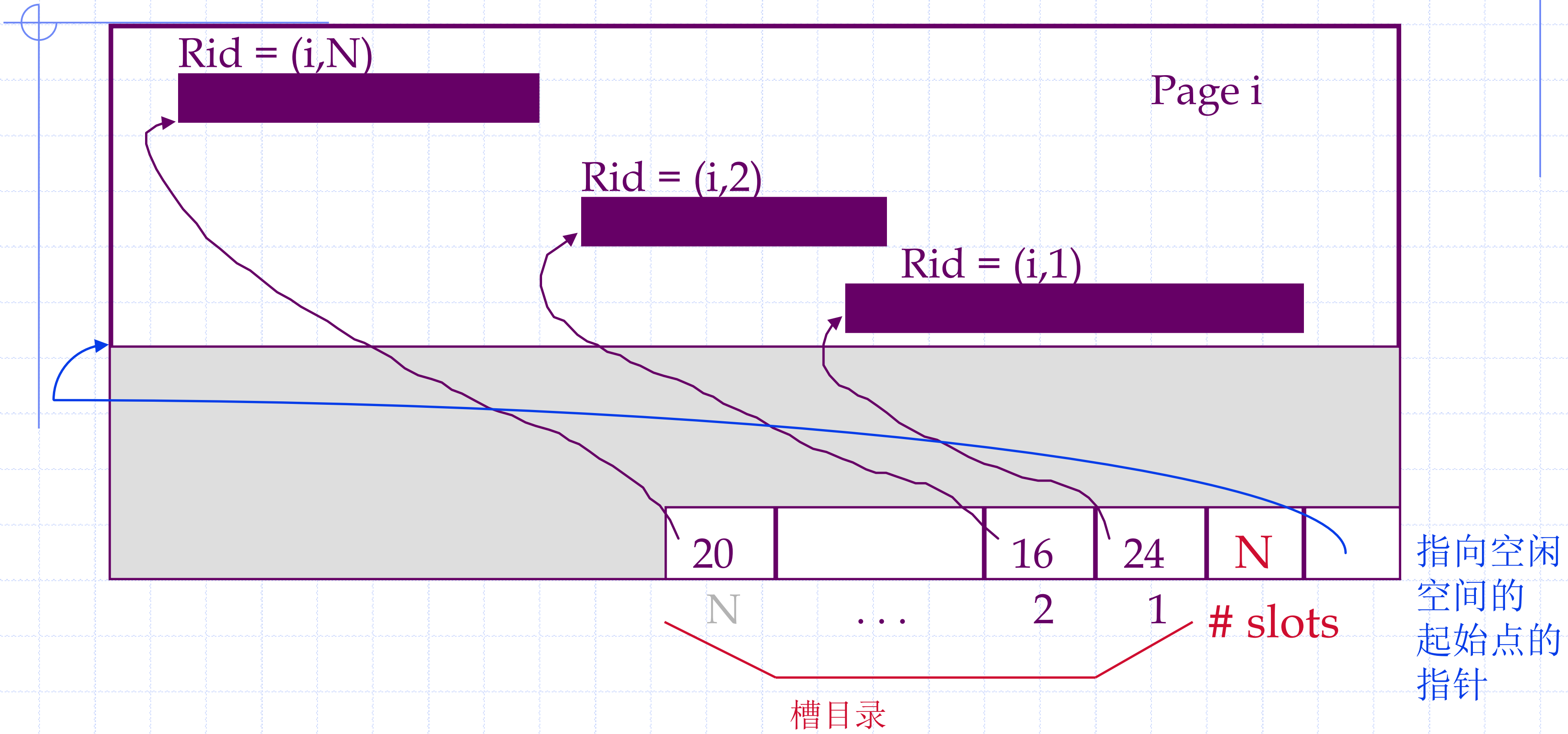
- within each record
- in block header

页面格式：固定长度



- 记录 id = <页 id, 槽号 #>

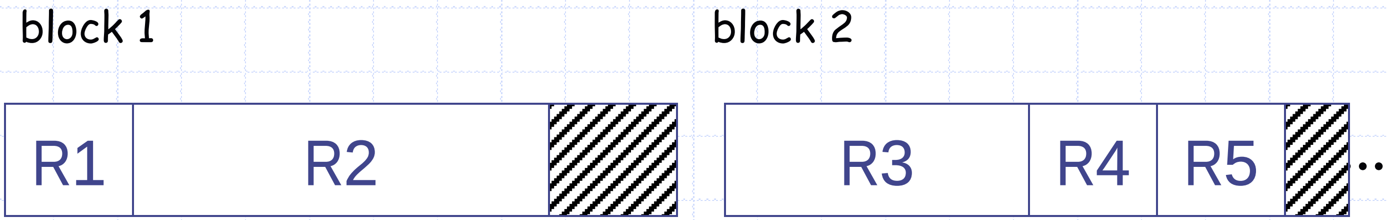
页面格式：变长记录



- 可以在页面中移动记录，而不改变记录id

(2) Spanned vs. Unspanned

- Unspanned: records must be within one block



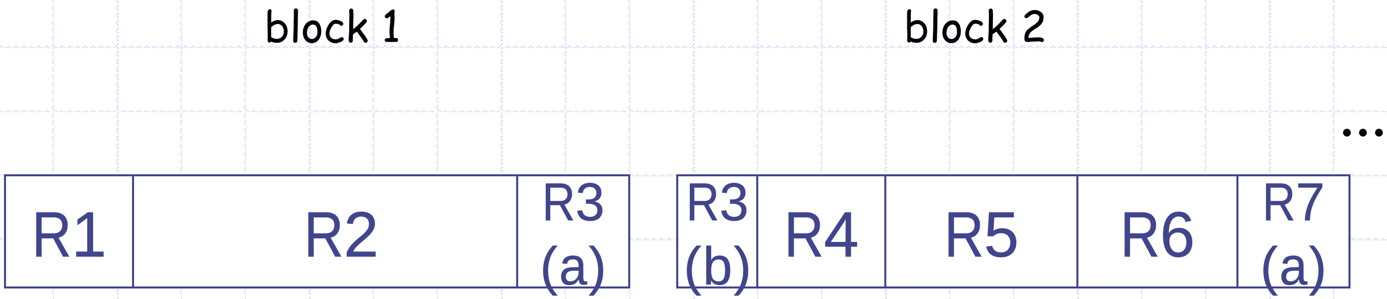
block 1

block 2

R1 R2 ▨

R3 R4 R5 ▨ ..

- Spanned



block 1

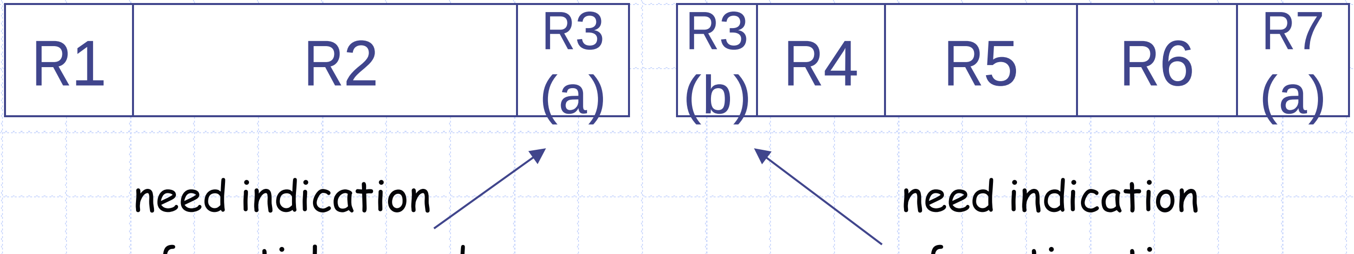
block 2

...

R1 R2 R3 (a)

R3 (b) R4 R5 R6 R7 (a)

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 space…
- Spanned essential if

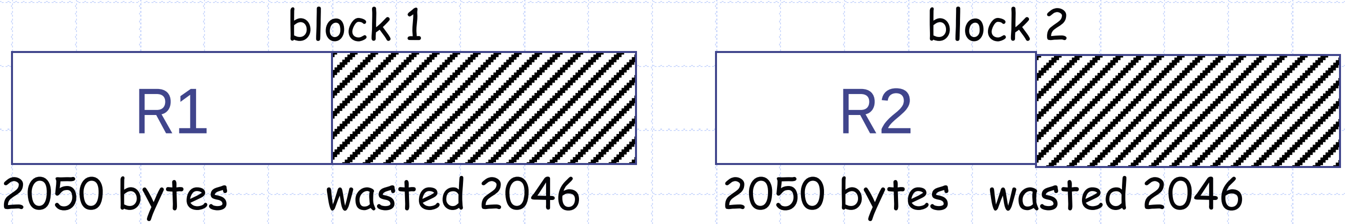
  record size > block size

Example

10^6 records

each of size 2,050 bytes (fixed)

block size = 4096 bytes

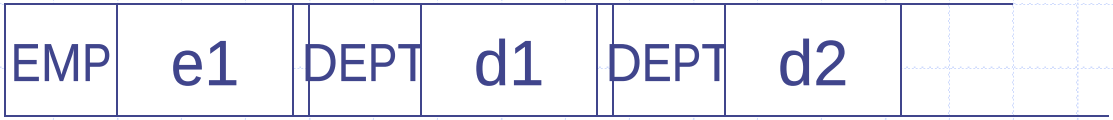


- Total wasted = 2×10^9 Utiliz = 50%
- Total space = 4×10^9

(3) Mixed record types

- Mixed - records of different types (e.g. EMPLOYEE, DEPT) allowed in same block

e.g., a block



EMP	e1	DEPT	d1	DEPT	d2	

Why do we want to mix?

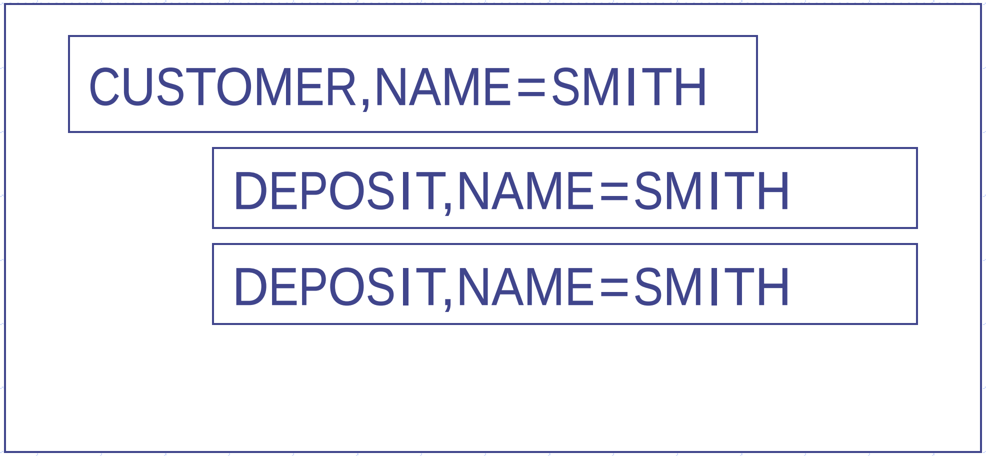
Answer: CLUSTERING

Records that are frequently accessed together should be in the same block

Example

```
Q1:  select A#, C_NAME, C_CITY, ...
         from DEPOSIT, CUSTOMER
         where DEPOSIT.C_NAME =
                          CUSTOMER.C.NAME
```

a block



CUSTOMER,NAME=SMITH

DEPOSIT,NAME=SMITH

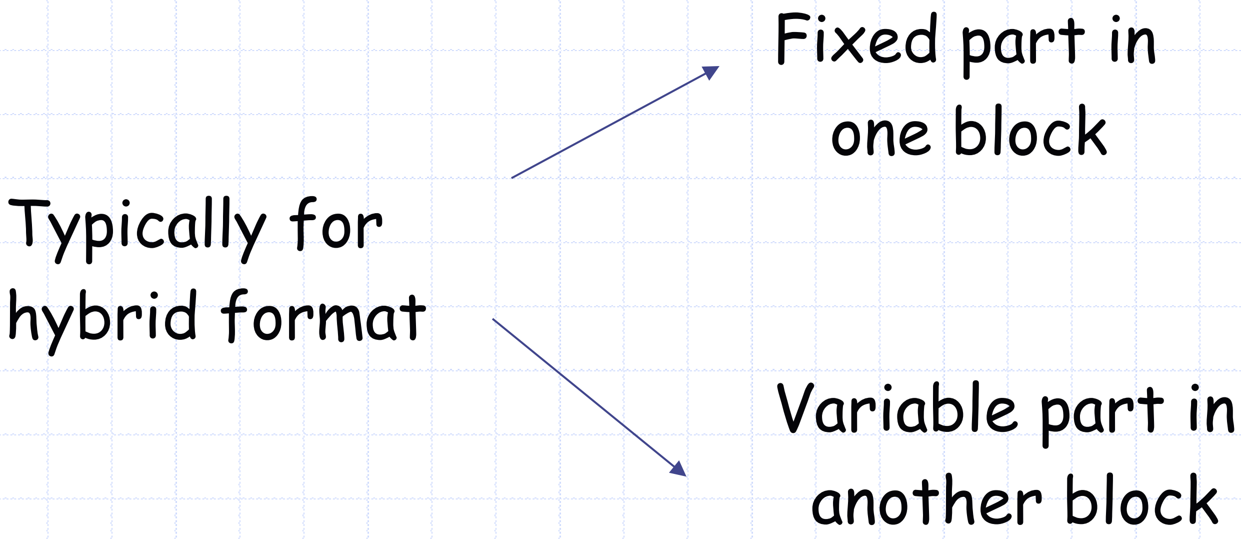
DEPOSIT,NAME=SMITH

- If Q1 frequent, clustering good
- But if Q2 frequent

```
Q2:    SELECT *
         FROM CUSTOMER
```

CLUSTERING IS COUNTER PRODUCTIVE

(4) Split records

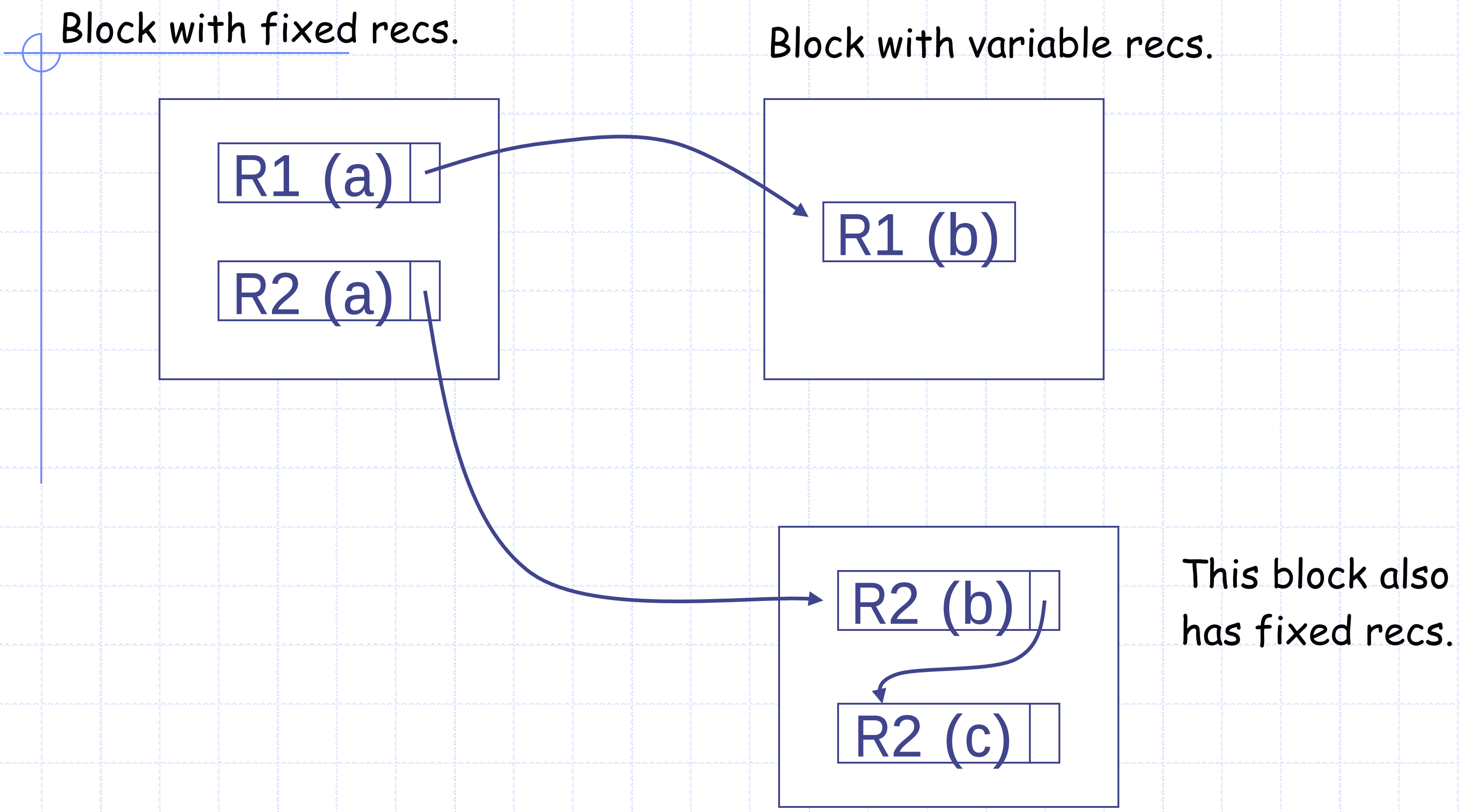


Typically for hybrid format

Fixed part in one block

Variable part in another block

We will give an example, then



Block with fixed recs.

Block with variable recs.

R1 (a)

R2 (a)

R1 (b)

R2 (b)

R2 (c)

This block also has fixed recs.

(5) Sequencing

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

Sequential file (⇒ sequenced)

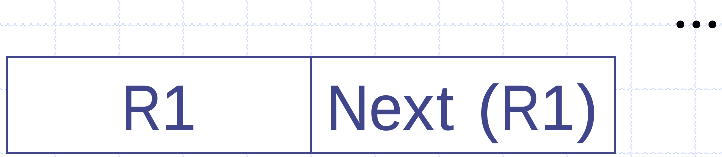
Why sequencing?

- Typically to make it possible to efficiently read records in order

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

Sequencing Options

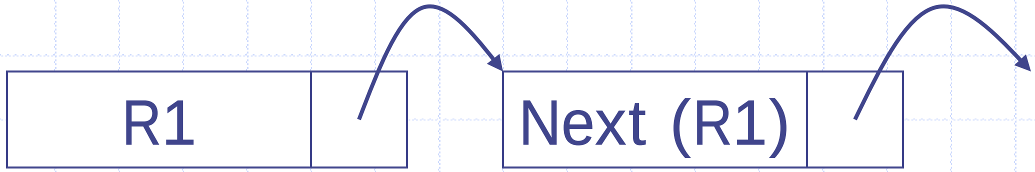
(a) Next record physically contiguous



...

R1 | Next (R1)

(b) Linked



R1

Next (R1)

(6) Indirection

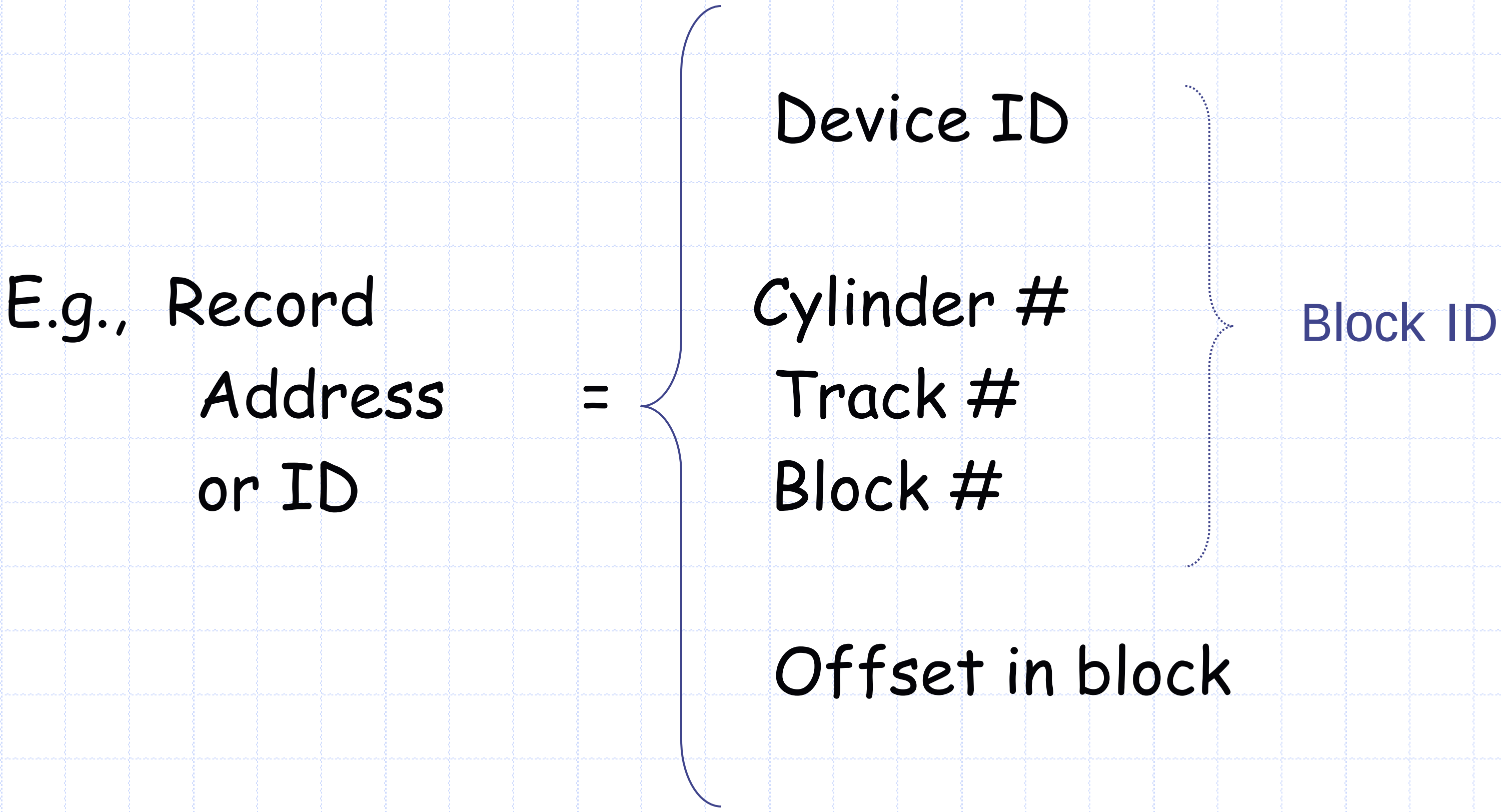
- How does one refer to records?

Rx

Many options:

Physical ↔ Indirect

Purely Physical



E.g., Record Address or ID =

Device ID

Cylinder #

Track #

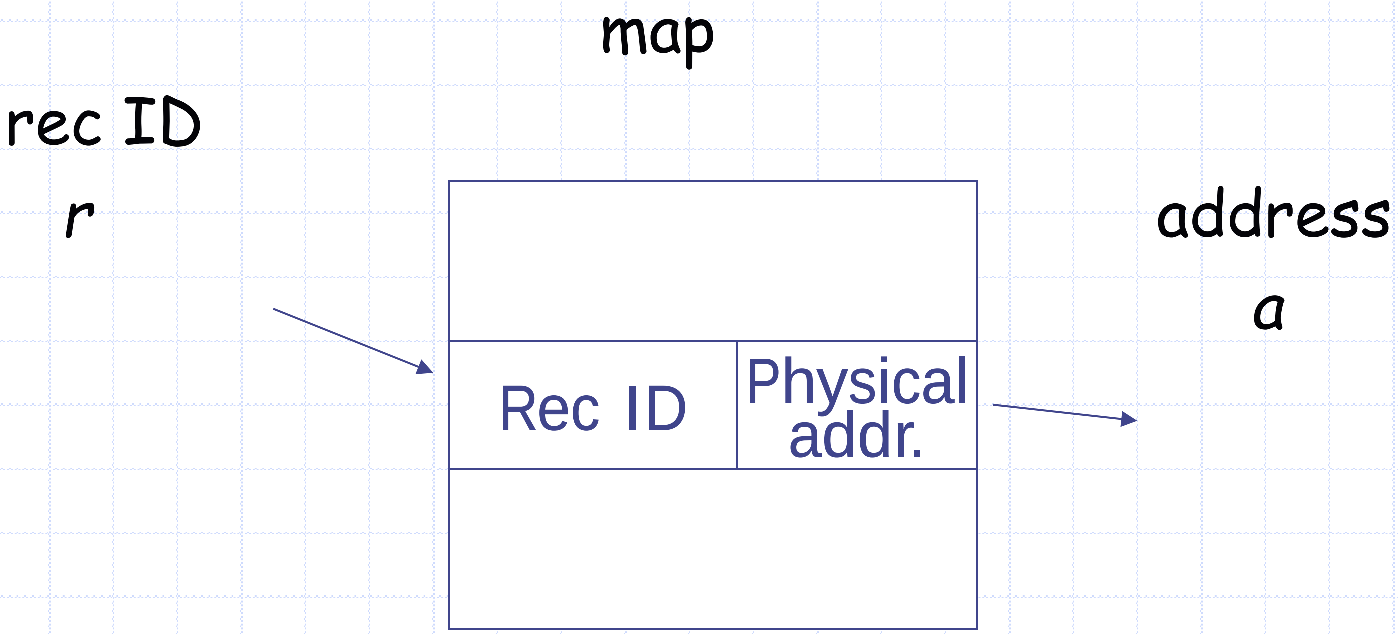
Block #

Block ID

Offset in block

☆ Fully Indirect

E.g., Record ID is arbitrary bit string



map

rec ID

r

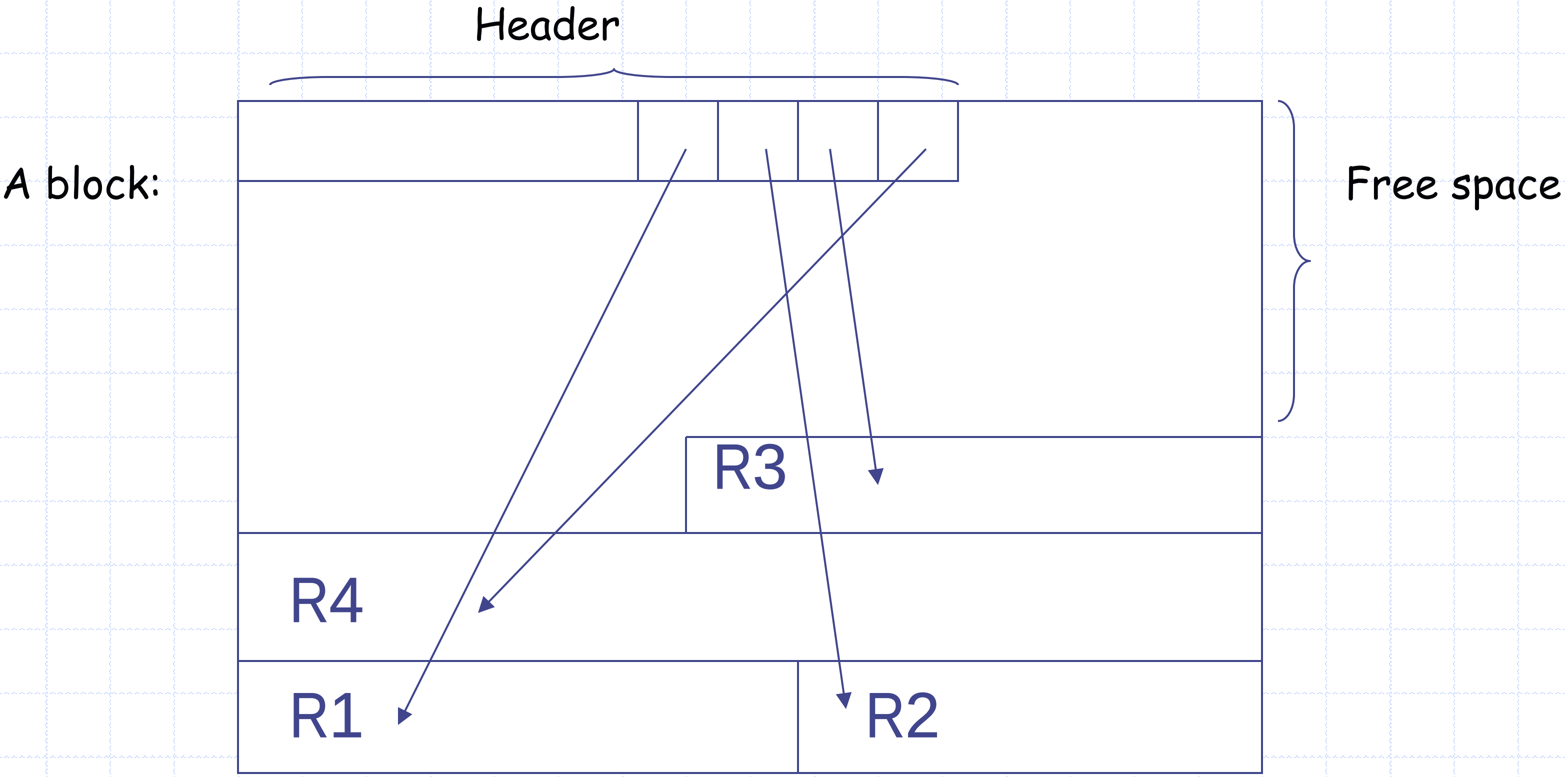
Rec ID

Physical addr.

address

a

Ex #1 Indirection in block



Block header - data at beginning that describes block

May contain:

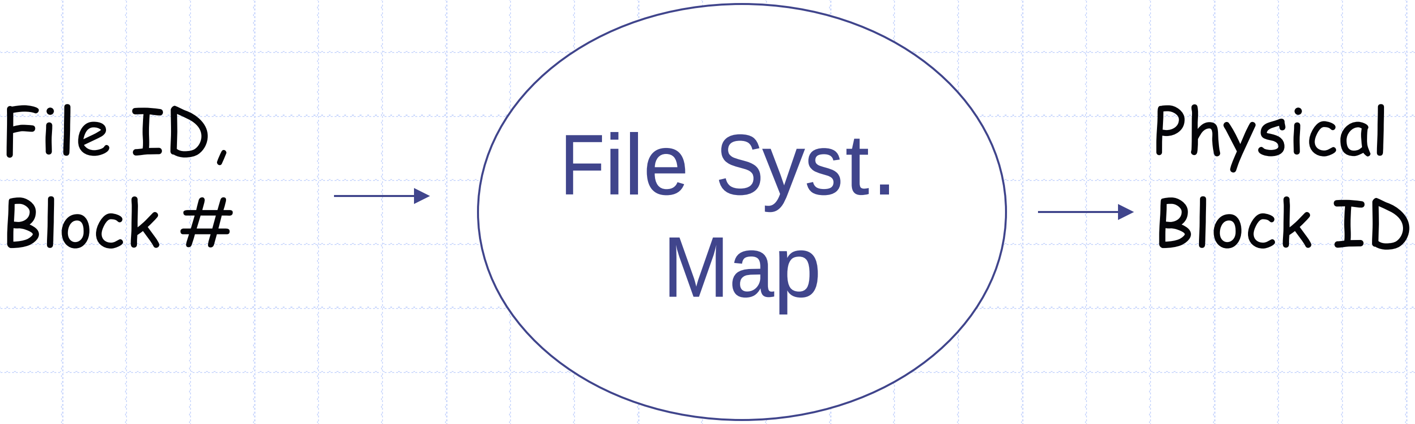
- File ID (or RELATION or DB ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4; is overflow, ...)
- Pointer to other blocks "like it"
- Timestamp ...

Ex. #2 Use logical block #'s understood by file system

REC ID → File ID

Block #

Record # or Offset



File ID,

Block # → File Syst. Map → Physical Block ID

Options for storing records in blocks

(1) Separating records
(2) Spanned vs. Unspanned
(3) Mixed record types - Clustering
(4) Split records
(5) Sequencing
(6) Indirection

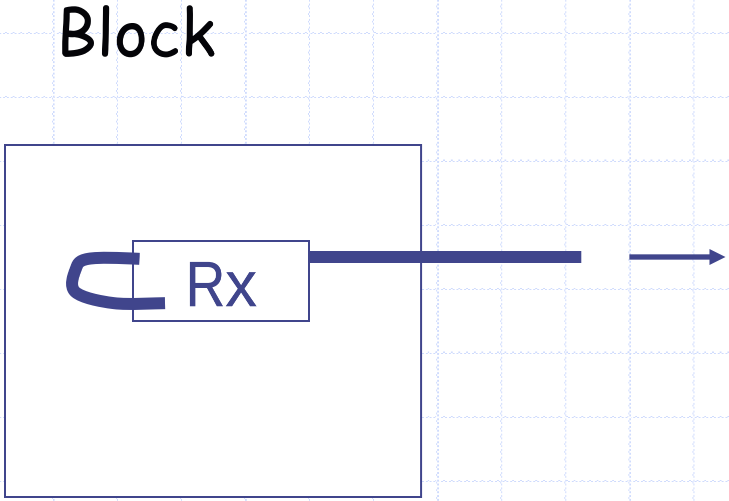
Other Topics

(1) Insertion/Deletion

(2) Buffer Management

(3) Comparison of Schemes

Deletion



Options:

(a) Immediately reclaim space

(b) Mark deleted

- May need chain of deleted records

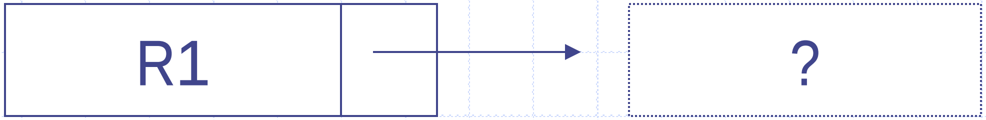
  (for re-use)
- Need a way to mark:
  - delete field

☆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

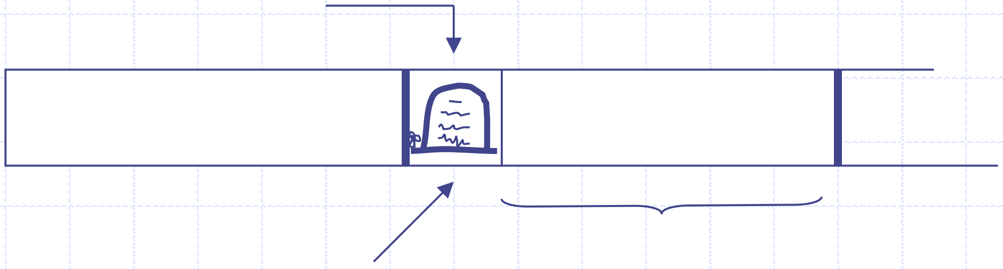


Solution : Tombstones

E.g., Leave "MARK" in map or old location

- Physical IDs

A block



This space never re-used

This space can be re-used

Solution : 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

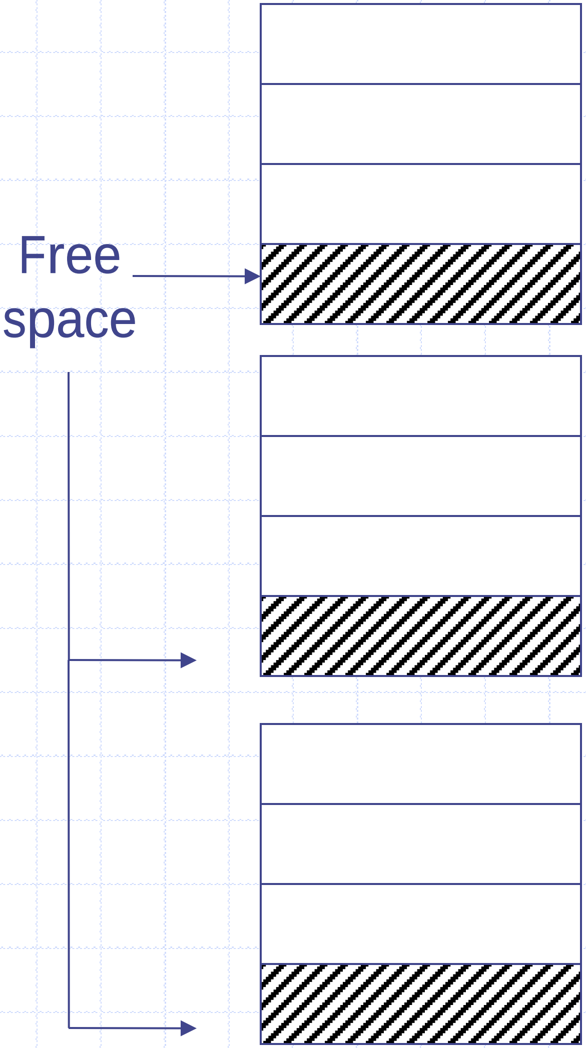
<u>Hard case:</u> records in sequence

→ 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?


Free space

Buffer Management

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

Read Textbook!

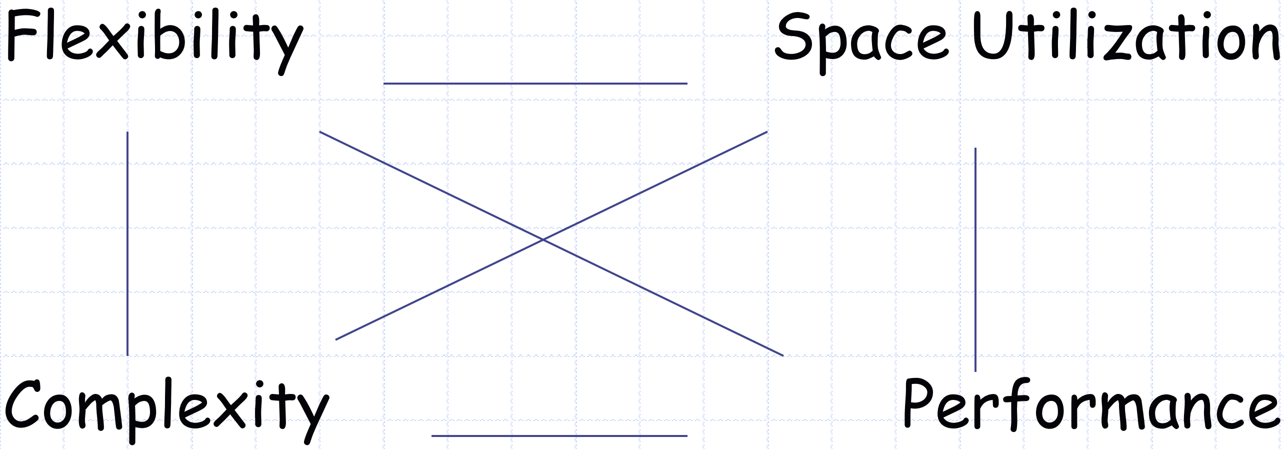
in chapter 02

Comparison

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

Which is right for me?

Issues:



Flexibility

Space Utilization

Complexity

Performance

- 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

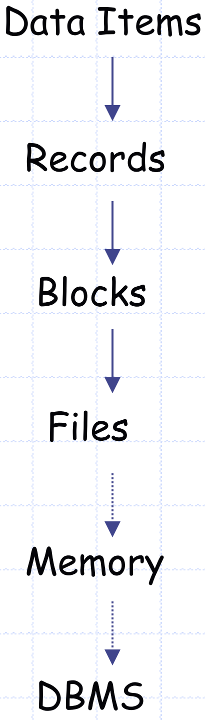
Example

How would you design storage system? (for a relational DB)

- Variable length records?
- Spanned?
- What data types?
- Fixed format?
- Record IDs ?
- Sequencing?
- How to handle deletions?

Summary

- How to lay out data on disk



Data Items

Records

Blocks

Files

Memory

DBMS

Next

How to find a record quickly – Index ()

Query parser & Execution

Any question?