Organization of Records in Blocks

Read Sec. 4.2 Riguzzi et al. Sistemi Informativi

Slides derived from those by Hector Garcia-Molina

Topic

- How to lay out records on blocks

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

  $\rightarrow$ 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 $\rightarrow$ 1 GREEN $\rightarrow$ 3

  BLUE $\rightarrow$ 2 YELLOW $\rightarrow$ 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:

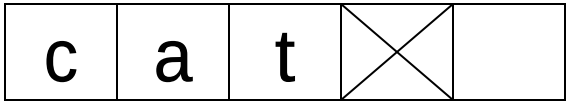
- Fixed length characters strings (CHAR(n)):
  - n bytes
  - If the value is shorter, fill the array with a *pad* charater, whose 8-bit code is not one of the legal characters for SQL strings

c	a	t	X	X	X

To represent:

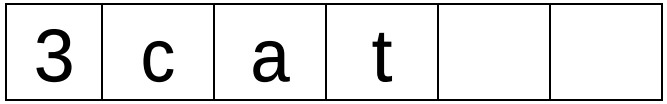
- Variable-length characters strings (CHAR VARYING(n)): n+1 bytes max
  - Null terminated

    e.g.,



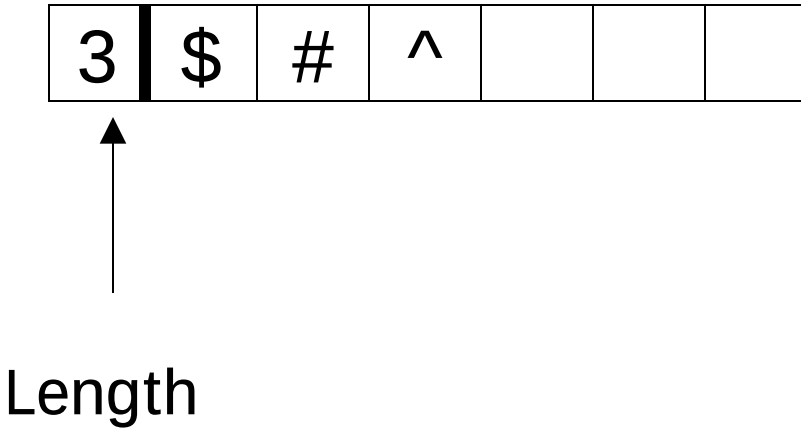
  - Length given

    e.g.,



To represent:

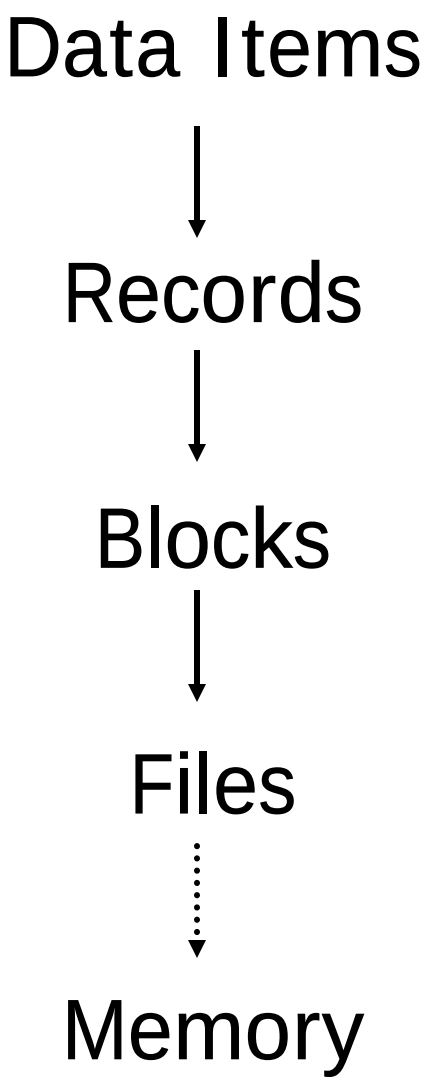
- BINARY VARYING(n)



Key Point

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

Overview



Types of records:

- Main choices:
  - FIXED vs VARIABLE LENGTH

A SCHEMA (not record) contains following information

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

Example: fixed length

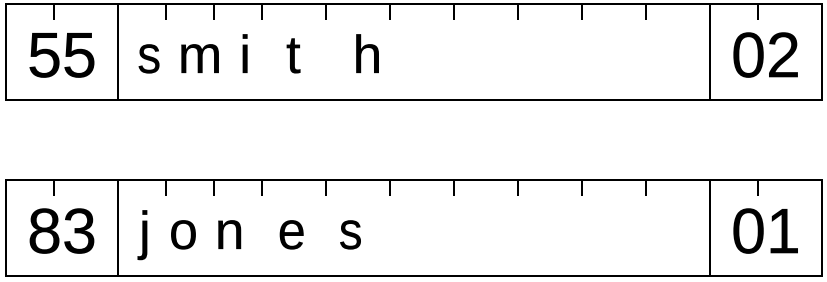
Employee record

(1) E#, 2 byte integer

(2) E.name, 10 char.

(3) Dept, 2 byte code

Schema



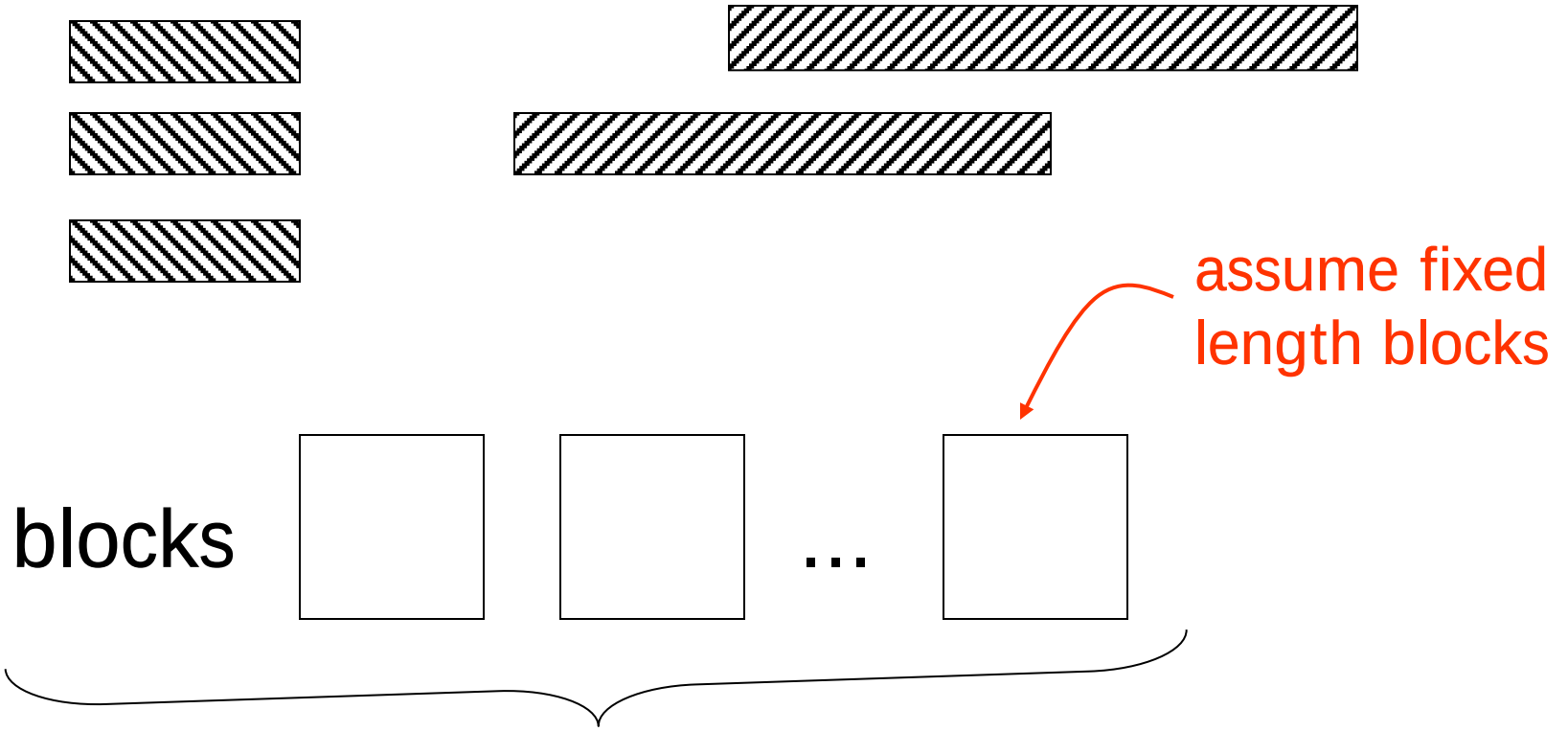
Records

Record header - data at beginning that describes record

May contain:

- record type
- record length
- time stamp
- ...

Next: placing records into blocks



Options for storing records in blocks:

(1) separating records
(2) spanned vs. unspanned
(3) mixed record types – clustering
(4) split records
(5) indirection

(1) Separating records

Block

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

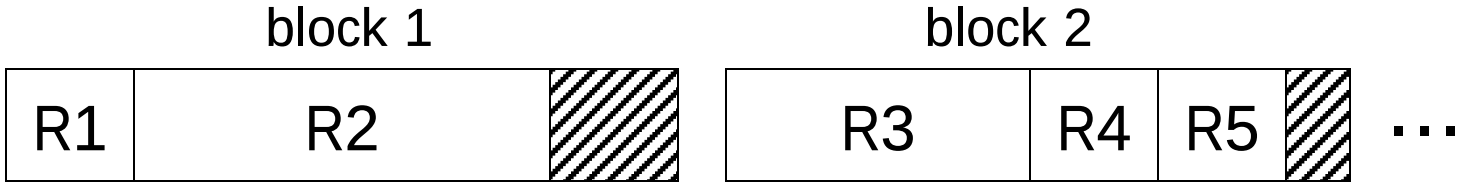
(b) special marker

(c) give record lengths (or offsets)

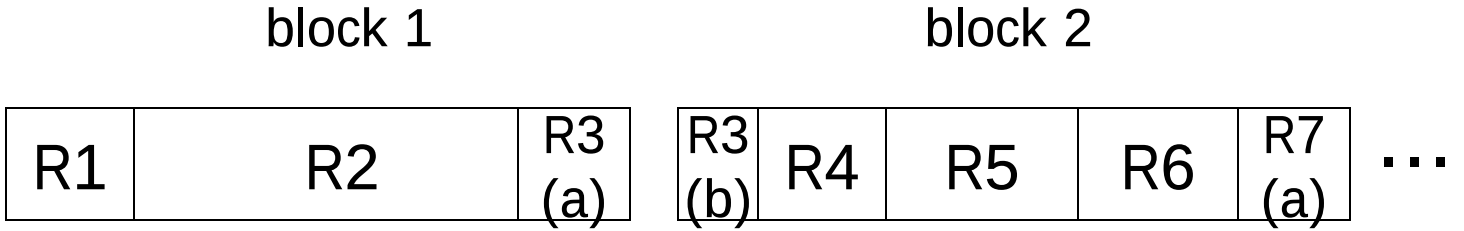
- within each record
- in block header

(2) Spanned vs. Unspanned

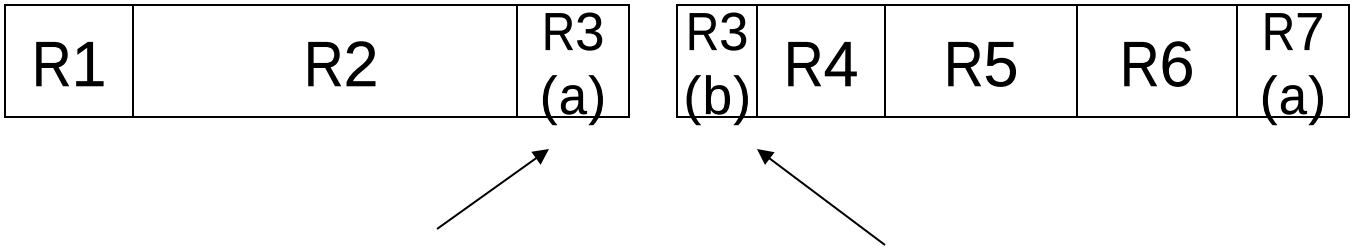
- Unspanned: records must be within one block



- Spanned



With spanned records:



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

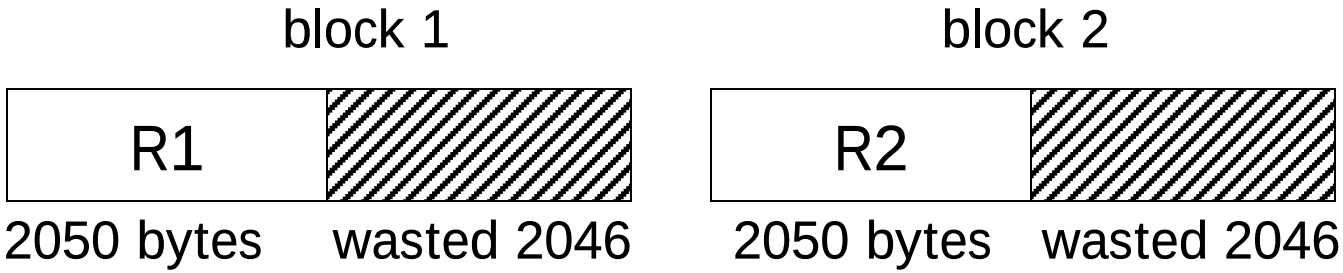
  record size > block size

Example

10^6 records

each of size 2,050 bytes (fixed)

block size = 4096 bytes

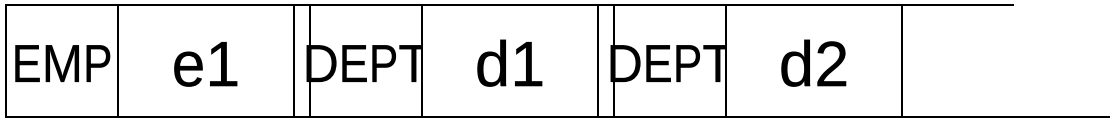


- Total wasted = 2×10^9 Utiliz = 50%
- Total space = 4×10^9

(3) Mixed record types

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

e.g., a block



Why do we want to mix?

Answer: CLUSTERING

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

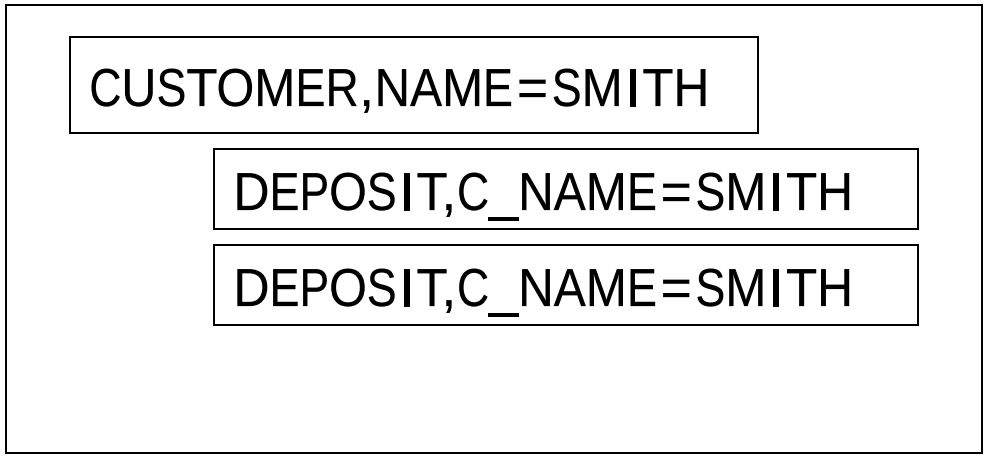
Compromise:

No mixing, but keep related records in same cylinder …

Example

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

a block



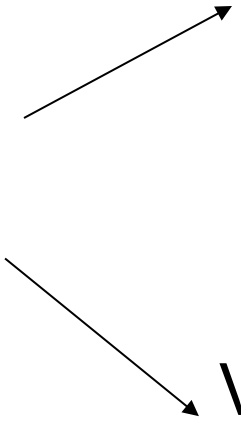
- If Q1 frequent, clustering good
- But if Q2 frequent

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

CLUSTERING IS COUNTER PRODUCTIVE

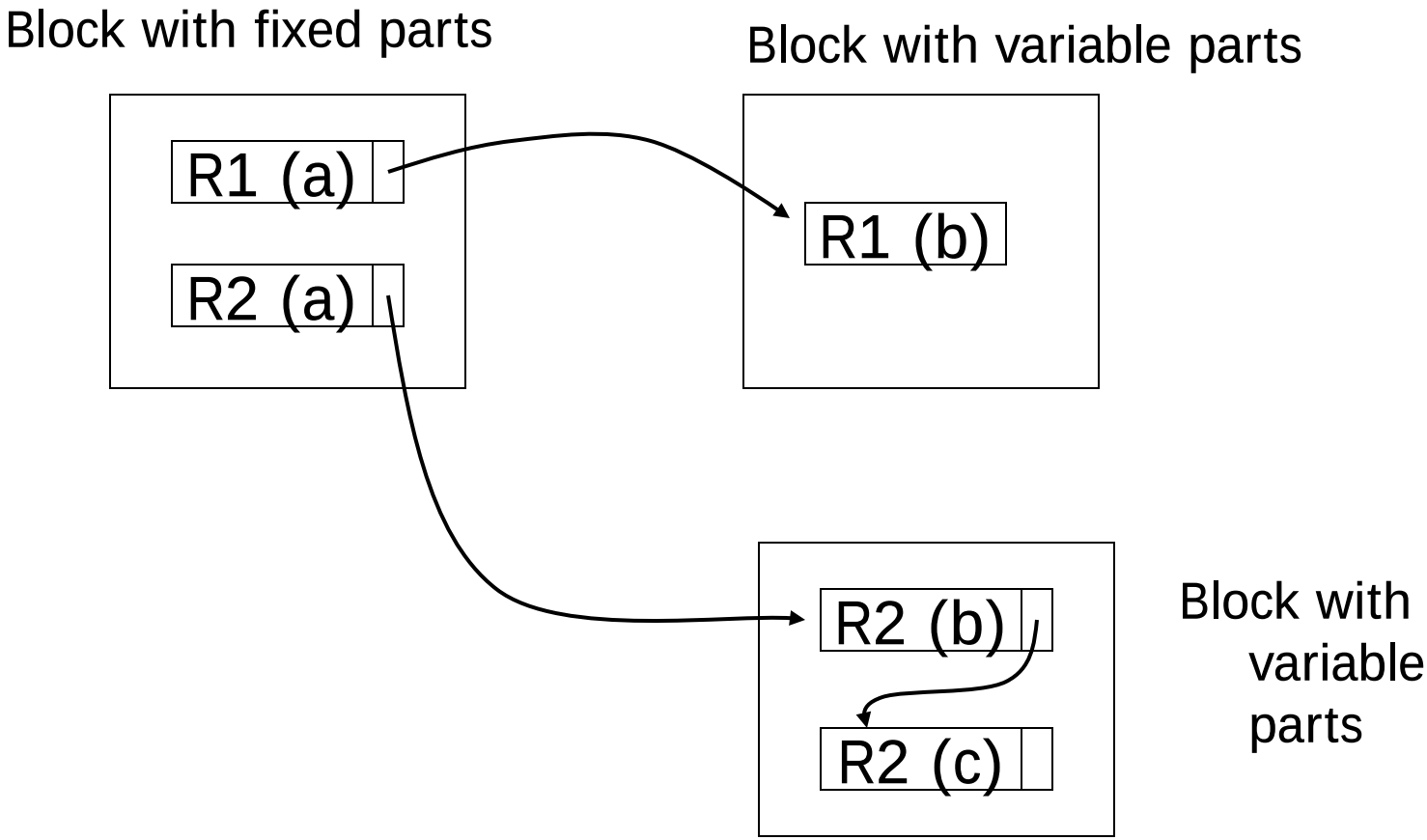
(4) Split records

Typically for Variable length records



Fixed part in one block

Variable part in another block


Block with fixed parts
Block with variable parts
R1 (a)
R2 (a)
R1 (b)
R2 (b)
R2 (c)
Block with variable parts

(5) Indirection

- How does one refer to records?

Many options:

Physical ⟷ Indirect

☆ Purely Physical

E.g., Record Address or ID =

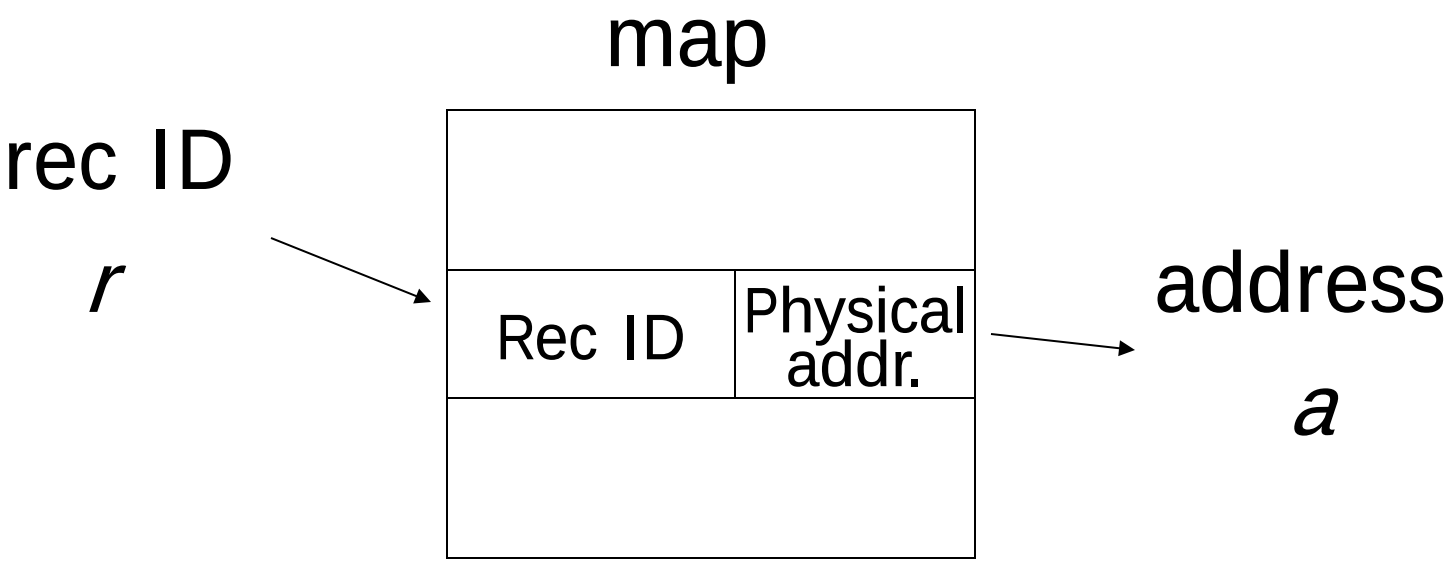
- Device ID
- Cylinder #
- Track #
- Block #

Block ID

- Offset in block

☆ Fully Indirect

E.g., Record ID is arbitrary bit string

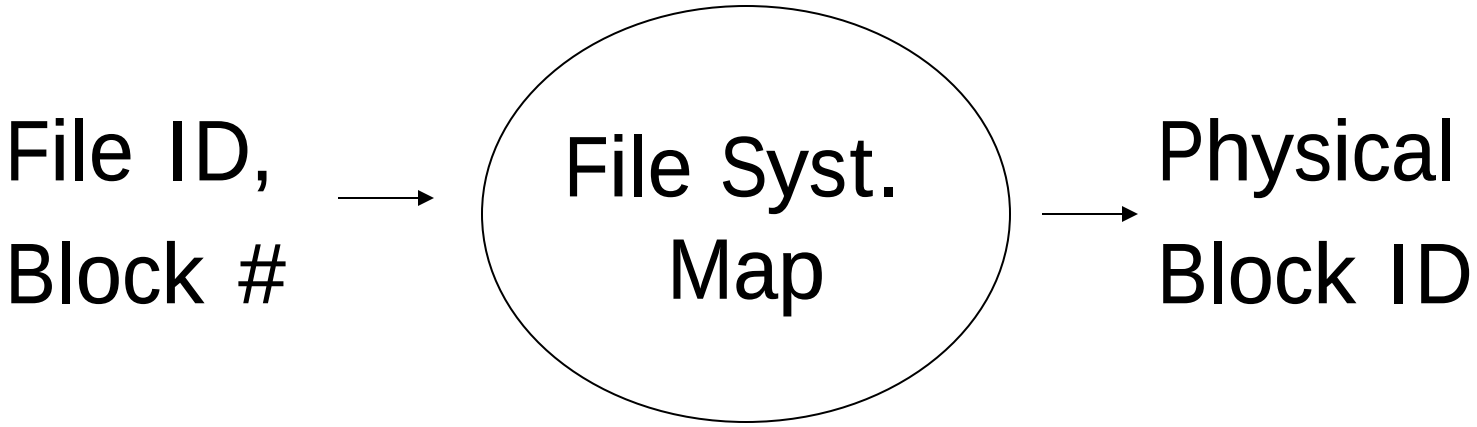


Tipical Use logical block #'s understood by file system

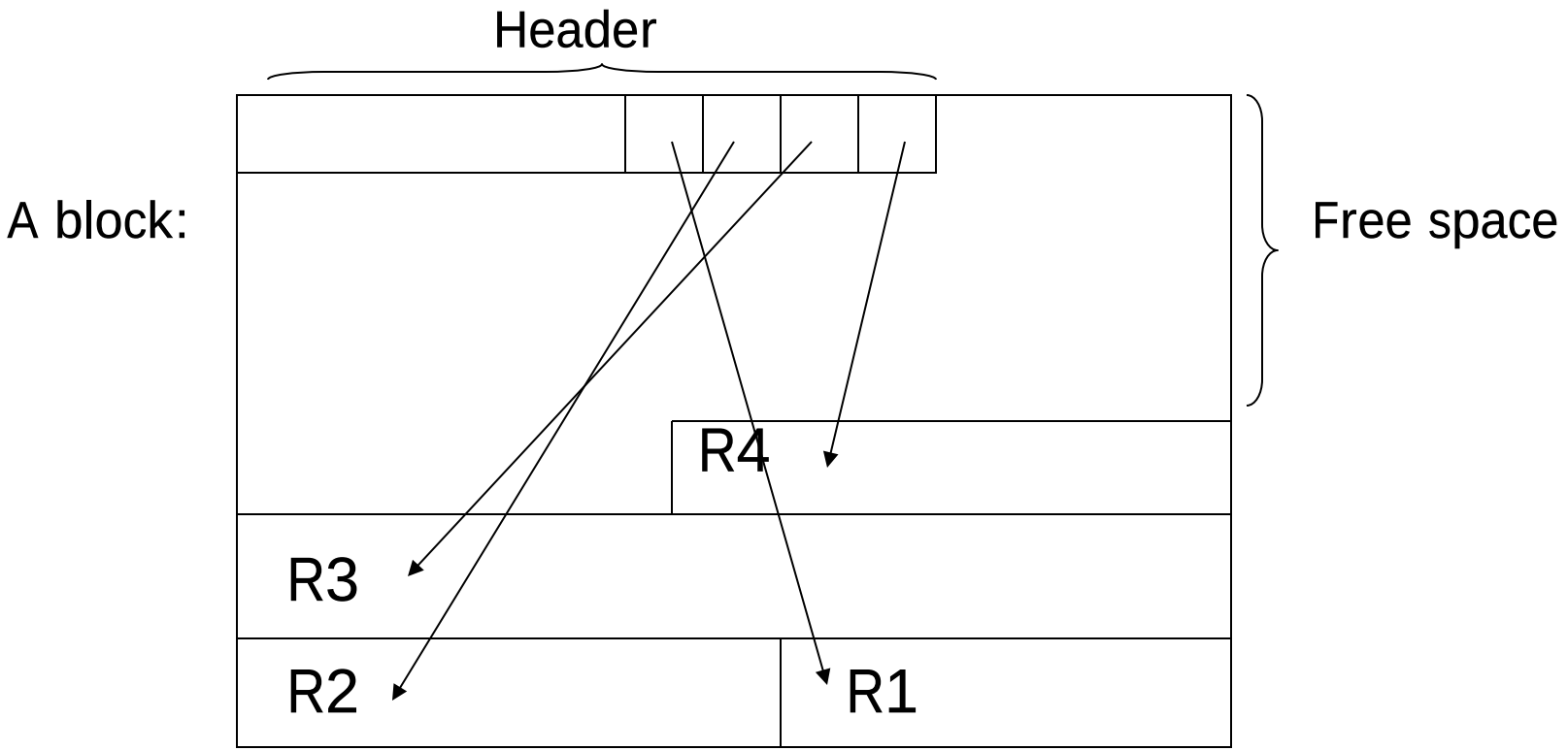
→ File ID

Block #

Offset in block



Indirection in block



Tradeoff

Flexibility ⟷ Cost

to move records of indirection

(for deletions, insertions)

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 Topic

Insertion/Deletion

Options for deletion:

(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 records to free space for immediate reclaim?
- How much space is wasted?
  - delete fields, free space chains,...

SQL Server

- The page size is 8 KB (8192 bytes), i.e. 128 pages per MB
- Each page begins with a 96-byte header that is used to store system information about the page:
  - page number, page type, the amount of free space on the page, and the allocation unit ID of the object that owns the page
- Eight physically contiguous pages form an **extent.** Extents are used to efficiently manage the pages. All pages are stored in extents.

Page Types in SQL Server

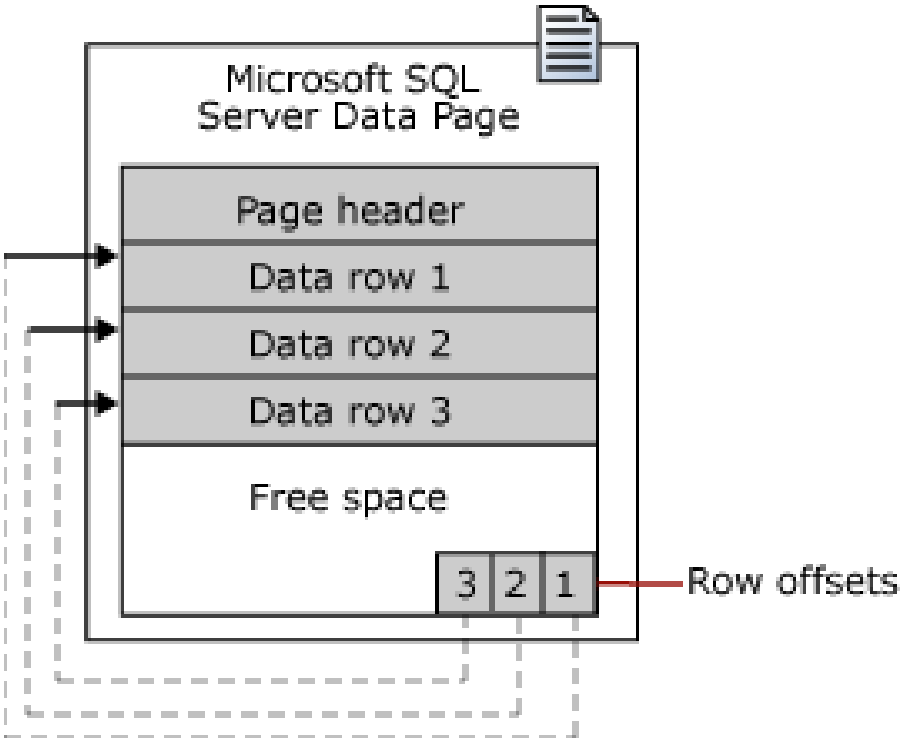
Page type	**Contents**
Data	Data rows with all data, except **text**, **ntext**, **image**
Index	Index entries.
Text/Image	•**text**, **ntext**, **image**, •**nvarchar(max)**, **varchar(max)**, **varbinary(max)**, and **xml** data when they don't fit in a block •Variable length columns when the data row exceeds 8 KB: **varchar**, **nvarchar**, **varbinary**, and **sql_variant**

Page Types in SQL Server

Page type	Contents
Global Allocation Map, Shared Global Allocation Map	Information about whether extents are allocated.
Page Free Space	Information about page allocation and free space available on pages.
Index Allocation Map	Information about extents used by a table or index per allocation unit.
Bulk Changed Map	Information about extents modified by bulk operations since the last BACKUP LOG statement per allocation unit.
Differential Changed Map	Information about extents that have changed since the last BACKUP DATABASE statement per allocation unit.

Data Pages in SQL Server

- Data rows are put on the page serially, starting immediately after the header.
- Row offset table:
  - Each entry records how far the first byte of the row is from the start of the page.



Large row support

- Rows cannot span pages in SQL Server, however portions of the row may be moved off the row's page so that the row can actually be very large.
- The maximum amount of data and overhead that is contained in a single row on a page is 8,060 bytes
- When the total row size of all fixed and variable columns in a table exceeds the 8,060 byte limitation, SQL Server dynamically moves one or more variable length columns to pages to the ROW_OVERFLOW_DATA allocation unit, starting with the column with the largest width.

Large row support

- When a column is moved to a page in the ROW_OVERFLOW_DATA allocation unit, a 24-byte pointer on the original page is maintained.
- If a subsequent operation reduces the row size, SQL Server dynamically moves the columns back to the original data page.