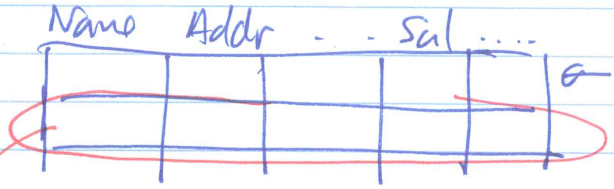
Storing Data on Disks (Storage Structure)

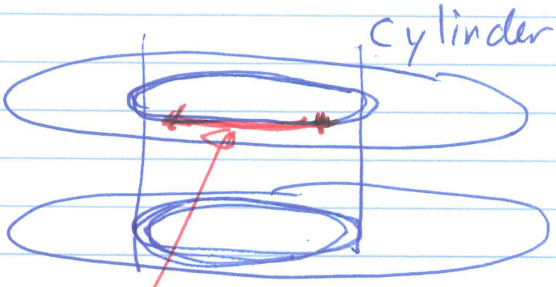
Fixed sized Records

Overview: Relation:



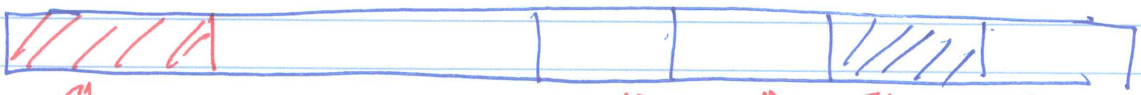
One tuple stored as one record

Disk:



block = a number of sectors

Block Structure:



Contains information about the Data records stored in the block

Normally: size(one record) < size(one Block)

Discussion:

(1) Storing fixed length record.
(record header).

(2) Packing fixed length record
into 1 block.
(block header)

(3) Addresses to identify
a block / record

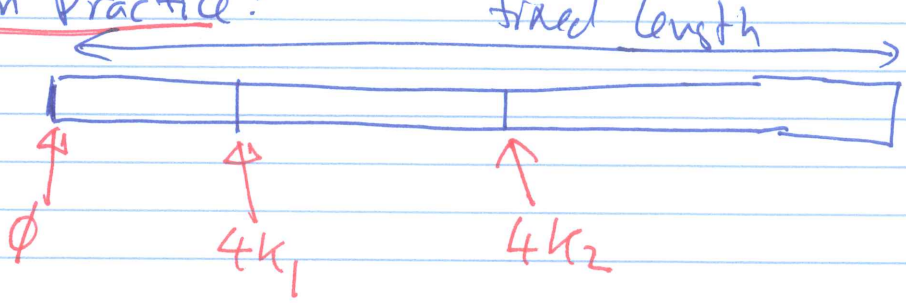
- Physical Addr (Host ID,
Disk ID
Cyl # Trk #, Blk #).

- Virtual Address (when block in memory)

(4) Logical Address + Map Table
↑
stored on Disk!
(Motivation: more record!!!)

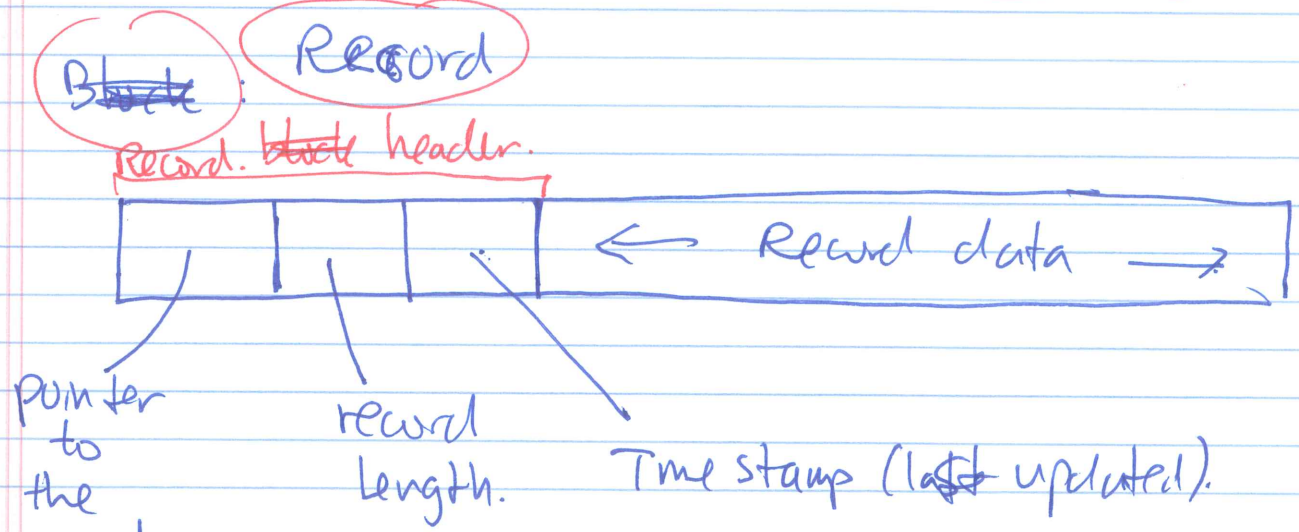
Storing FIXED-LENGTH data records.

- Common Practice:



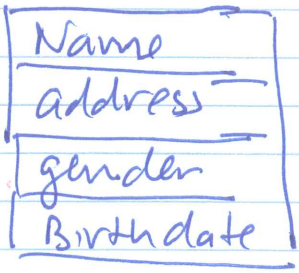
Each field begins at an address that is a multiple of (4) or (8).
(That's because of the "alignment requirement" of int or float (4) or double (8) variables.)

- Format for fixed-length records:



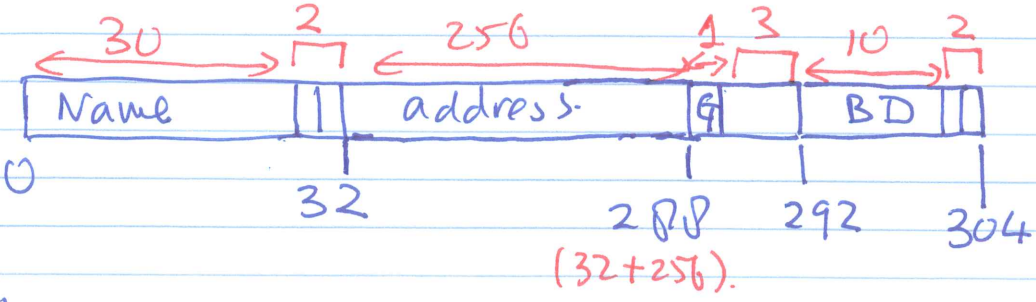
Example:

Movie Star Record:

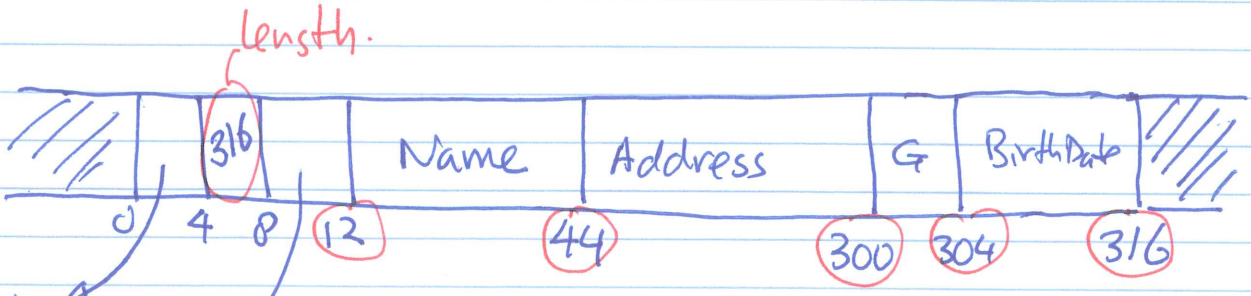


30 char.
256 char.
1 char.
Date (10 bytes)

(1) Records stored as follows:



(2) Record + Record Header.



pointer to Record Schema

time stamp.

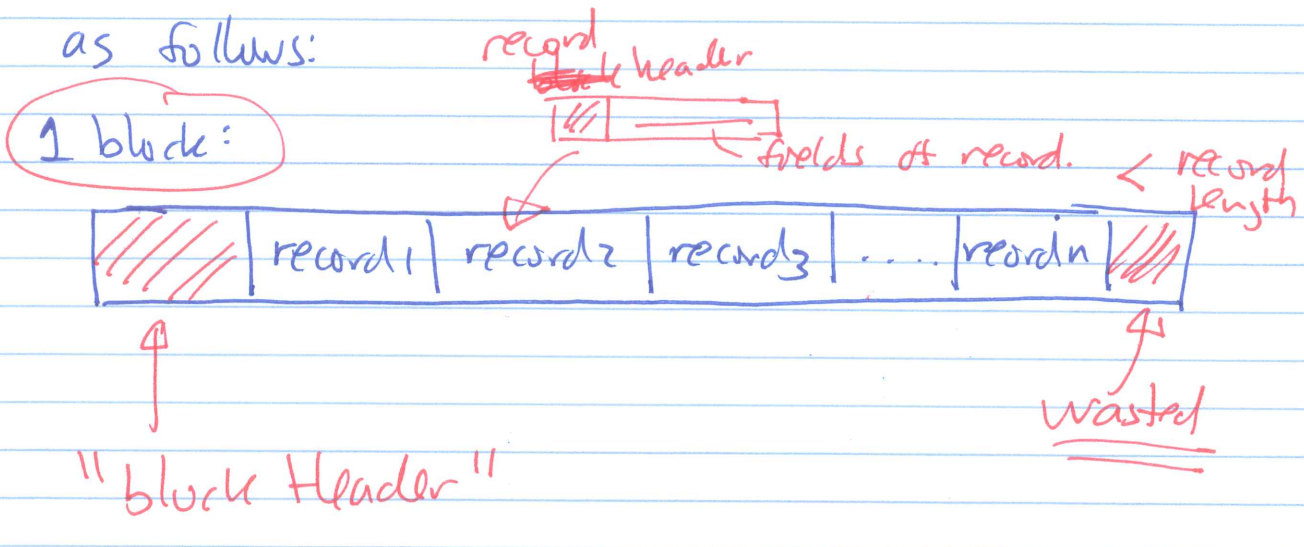
stored in start of file

Assume each field is 4 bytes long.

Packing fixed-length Records into Blocks:

- Records (of the format given previously) as stored (packed) into 1 block.

as follows:



- In general:

1 block can store

records from DIFFERENT relations

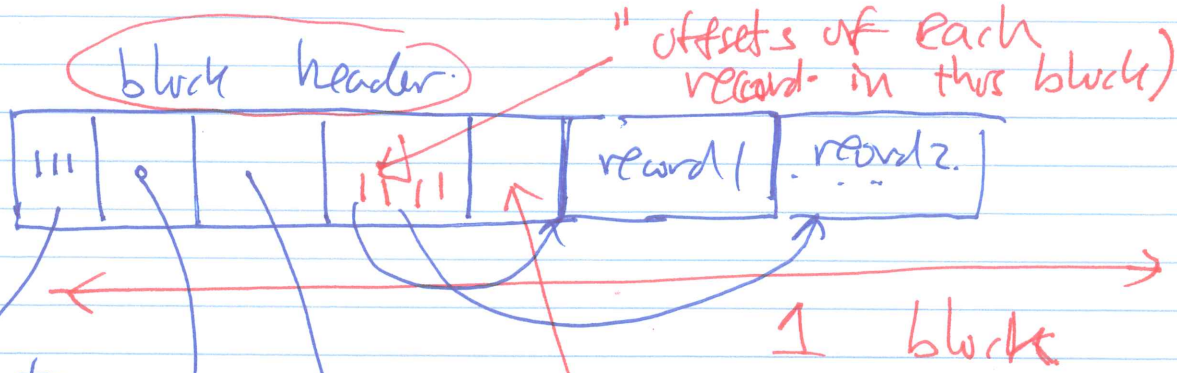
More common:

1 block store

records from SAME relation.

Assume this is the rule

What info. stored in Block Header:



Links to other blocks.

info. about the role of THIS block.

time stamp of block's last modification / access.

relation that records belong to

all records belongs to this relation.

Move in "indexing" (next chapter)

Block and Record Addresses

- Fact:

  A block and a record is identified by:

  block → block address

  record → record address

- A block/record can be located in 2 places:

  (1) On the disk

  (that's the permanent storage location.

  (2) In main memory.

  (while the block/record is being accessed by the DBMS).

- A block (record) located on a disk is identified by: a physical address.

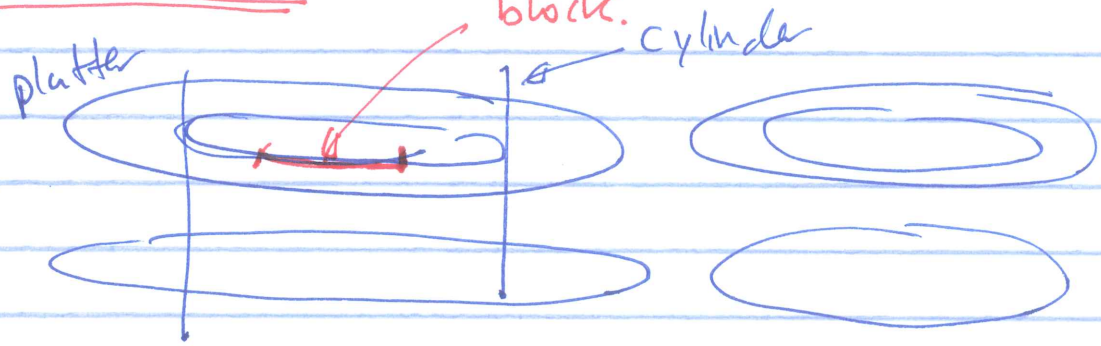
A block/record located in main memory is identified by: a virtual address

(64 bit binary number)

Physical Address

how to locate a block/record on DISK.

Block address on Disk:



(1) The "physical" address of a block consists of:

≥ 8 bytes

(1) host ID (that contains the disk)

(2) Disk ID (that identifies the disk)

(3) Cylinder number (of the disk)

(4) Track number (which platter)

(5) Block number (inside the track)

(2) The "physical" address of a record:

= physical addr. of the block (that contains the record)

+ offset of the record in the block.

Alternative way to identify block:

Logical Addresses of blocks/records.

- Each block/record is assigned a logical address.

Alternative way to addr. a block

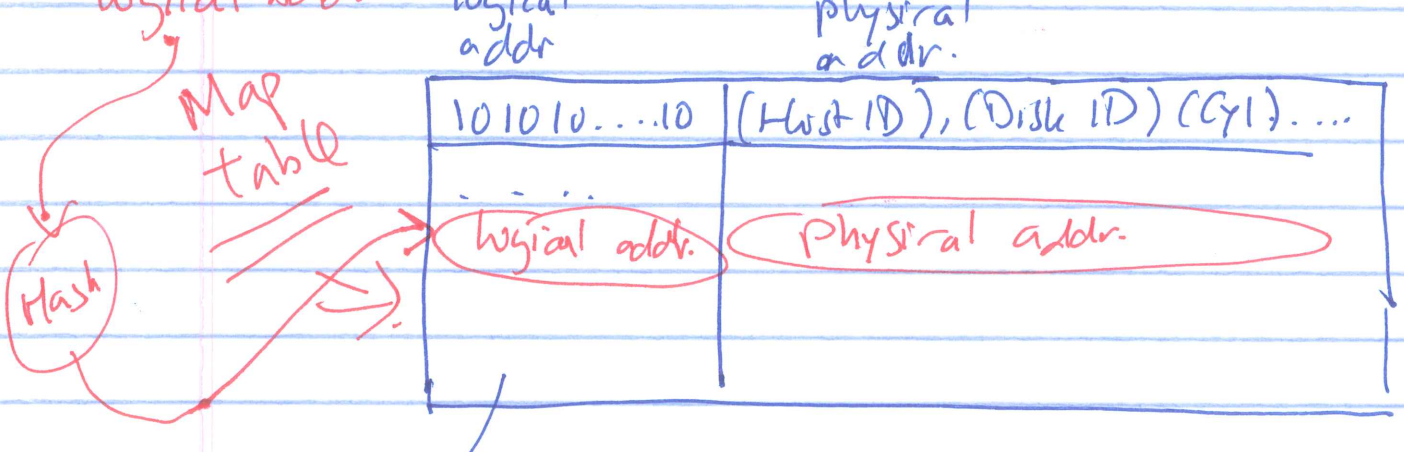
logical addr = an arbitrary string of bits of some fixed length

(large enough to identify every block/record in the whole data base)

- Map table = table that maps mappings.

logical address ⟷ physical Address

Access: logical Addr



n bits ⇒ 2^n "database objects"

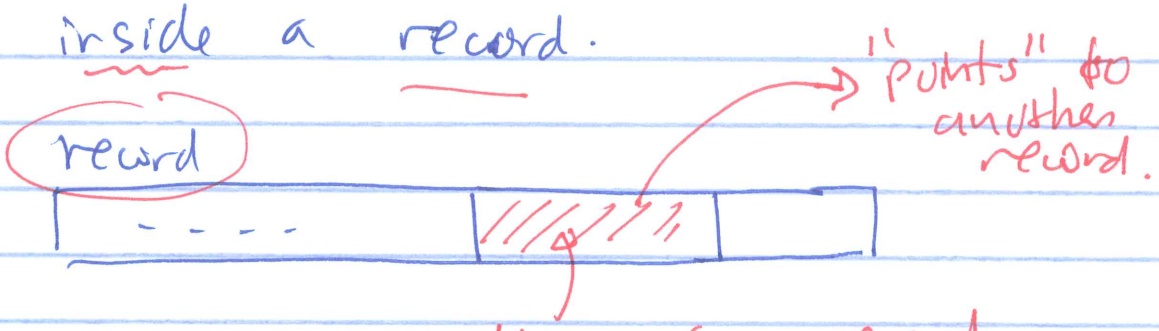
Fact: The Map Table is stored on disk (in a known location)

Motivation: Why use "logical" block/record address?

Reason: flexibility.

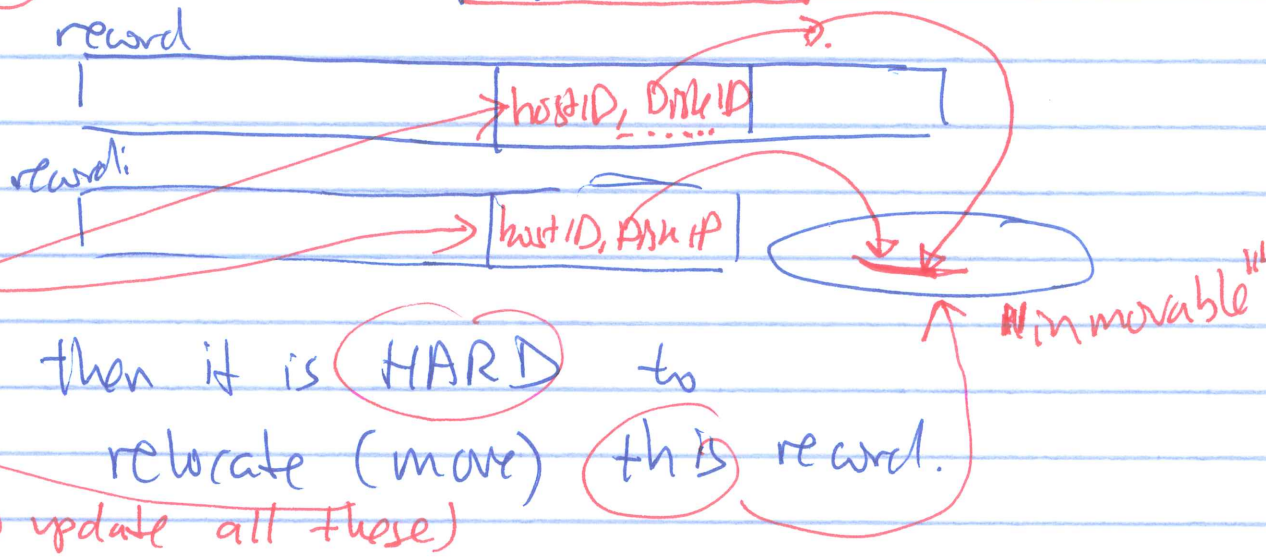
Example:

Some techniques store "pointers" to a record (block) inside a record.



(eg: foreign key).

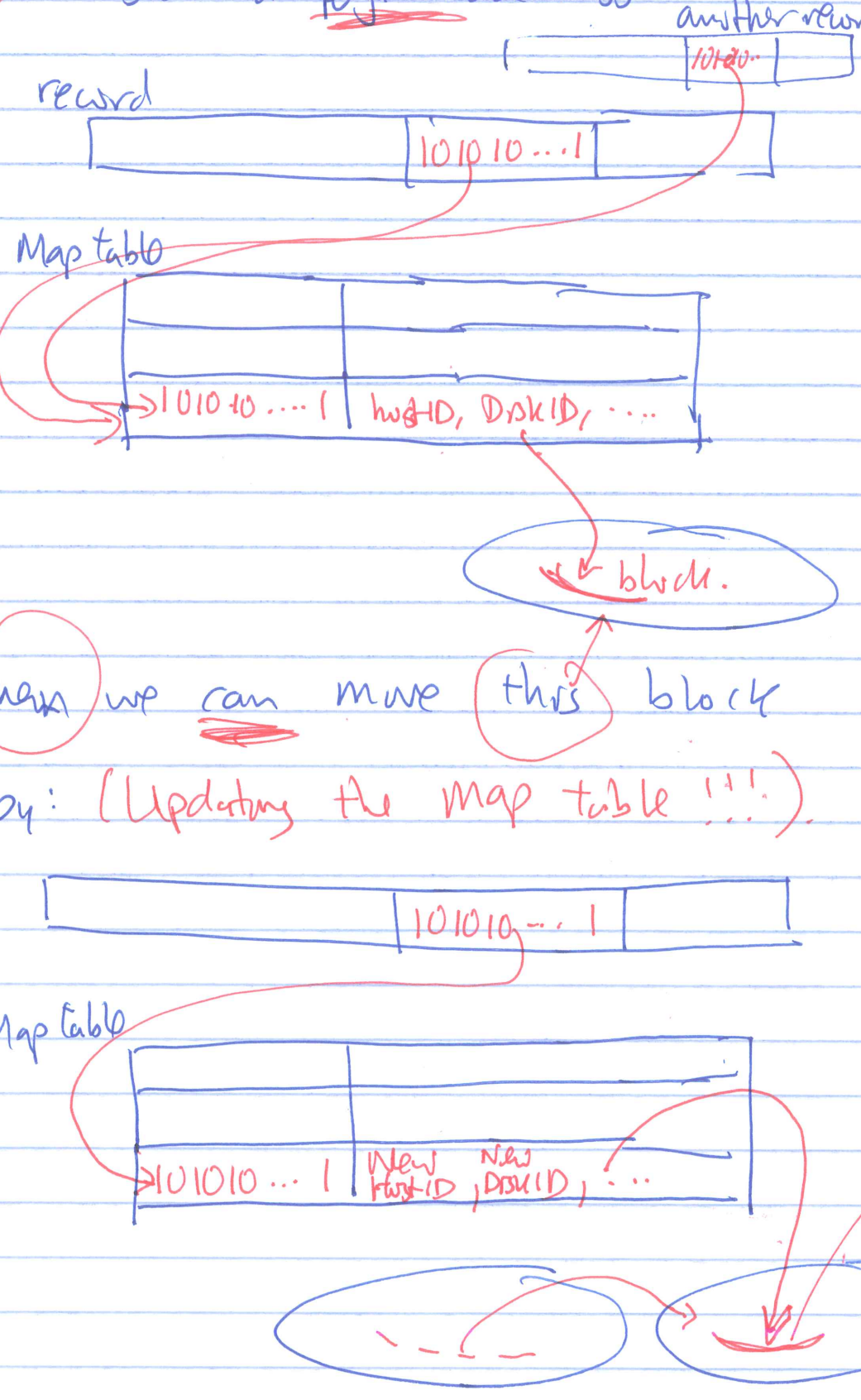
- If we store a physical address:



then it is HARD to relocate (move) this record.

(Need to update all these)

- If we store a logical address:



Then we can move this block by: (Updating the map table !!!)

Definition:

Database address of an object (block/record)

= logical/physical address of object

If you use a logical address,
you can use the MAP table
to obtain the physical addr

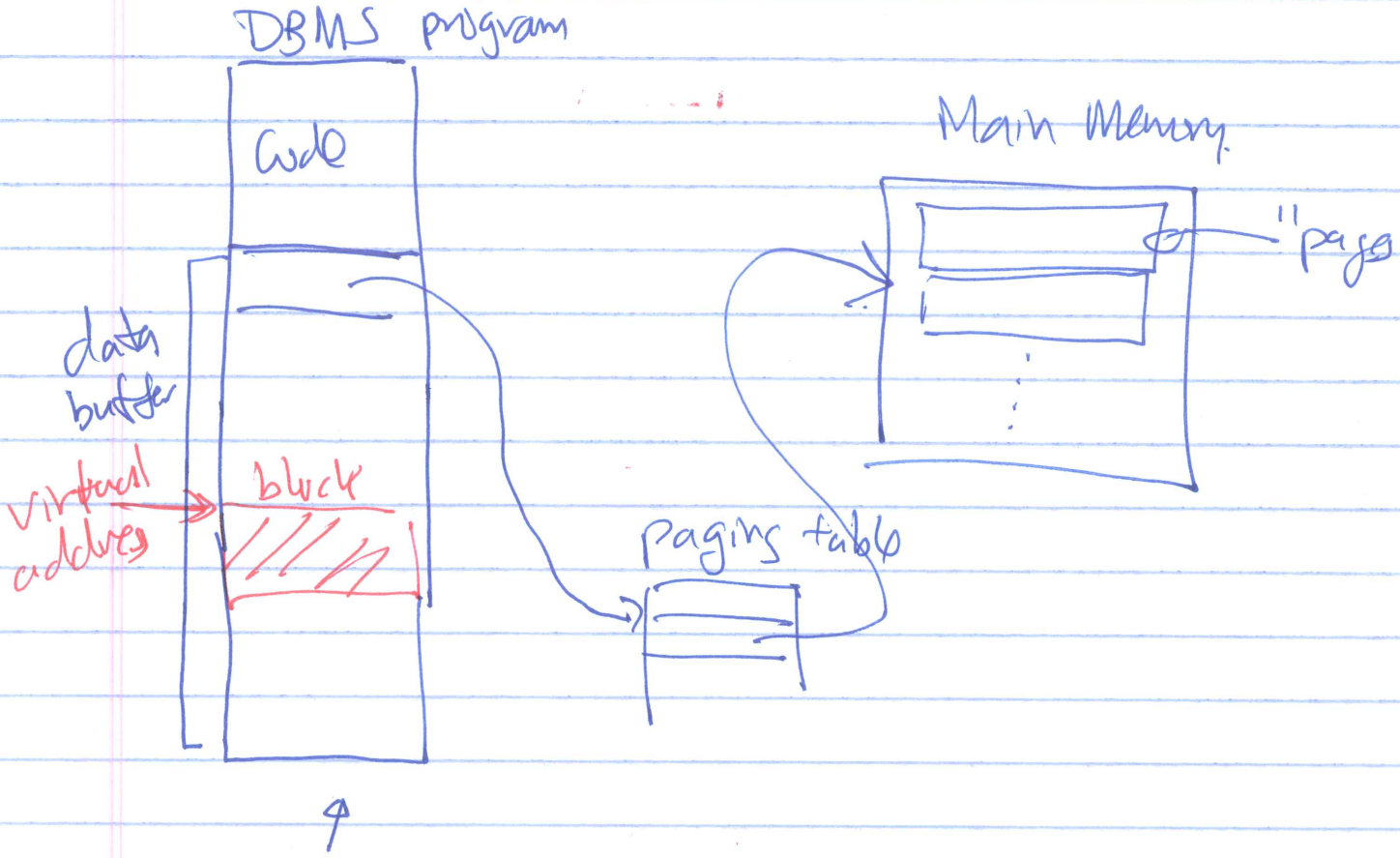
So: without loss of generality,
by a "Database addr",
we will always assume:

HostID, DiskID, Cyl #, trk #, Blk #

= physical address.

Virtual addr. is used to IDENT. a block while it's IN MEMORY !!!

- Virtual Address: paging.



Very large memory space

"Virtual memory"

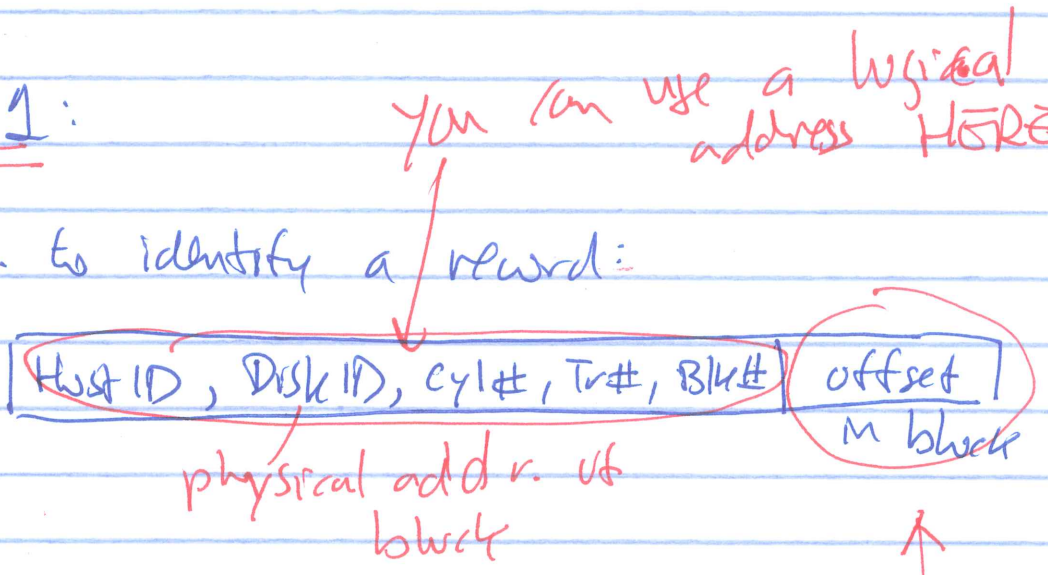
- If a data block is read (from disk) into a buffer inside the DBMS program, the block in memory will be used instead of the one stored on disk !!!

Combinations of logical / physical addresses

- Different combinations of logical / physical addresses to identify a record are possible

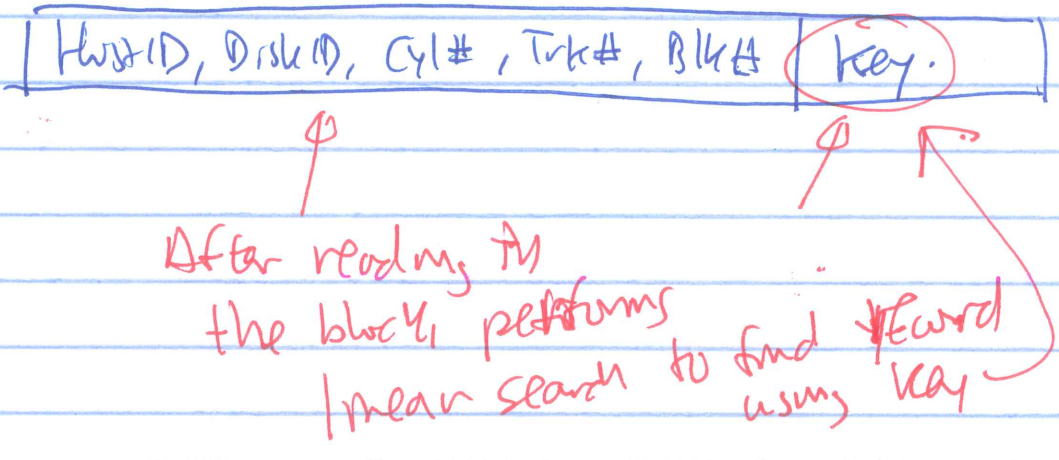
- Example 1:

Addr. to identify a record:



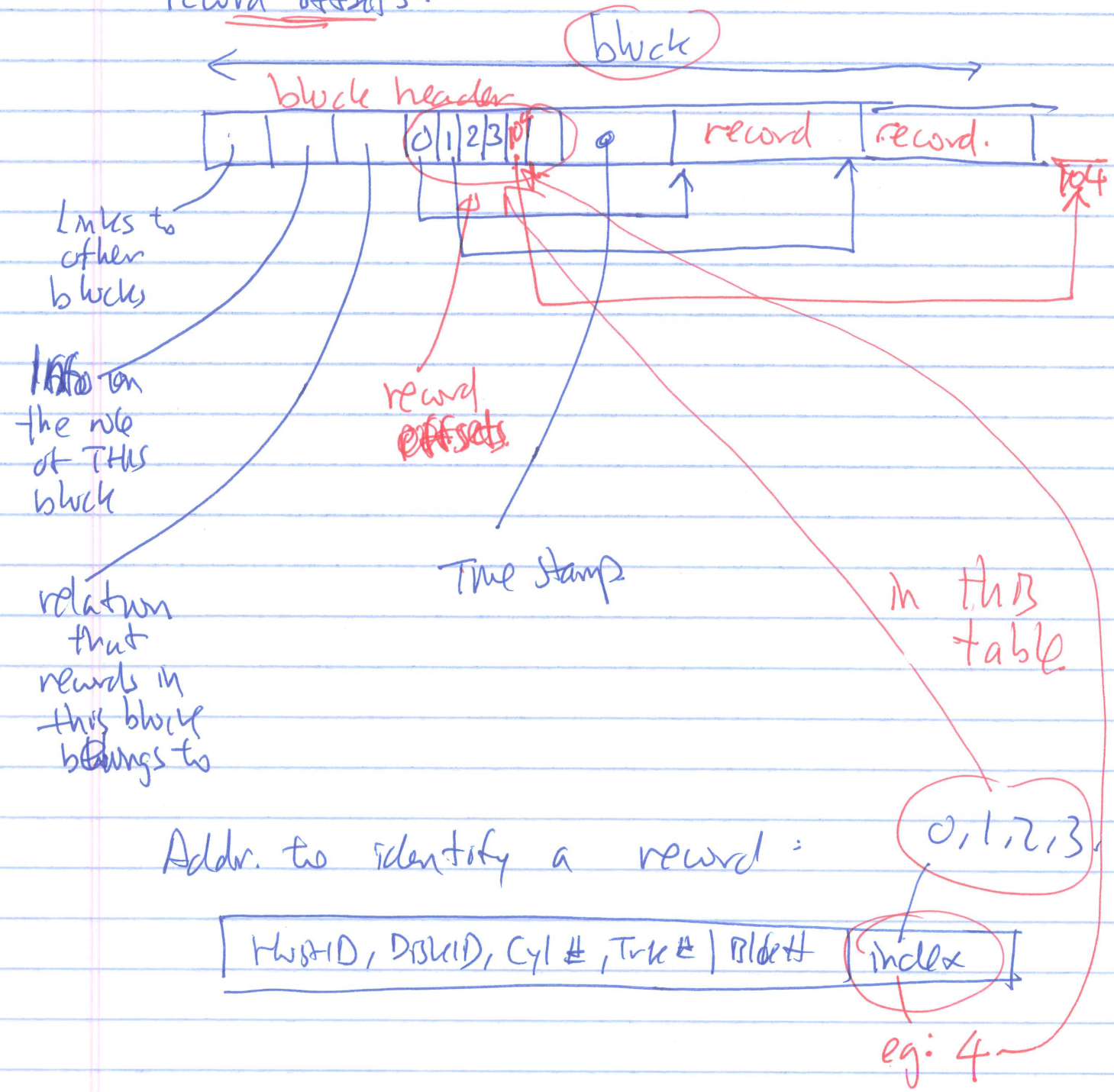
- Example 2:

Addr. to identify a record:



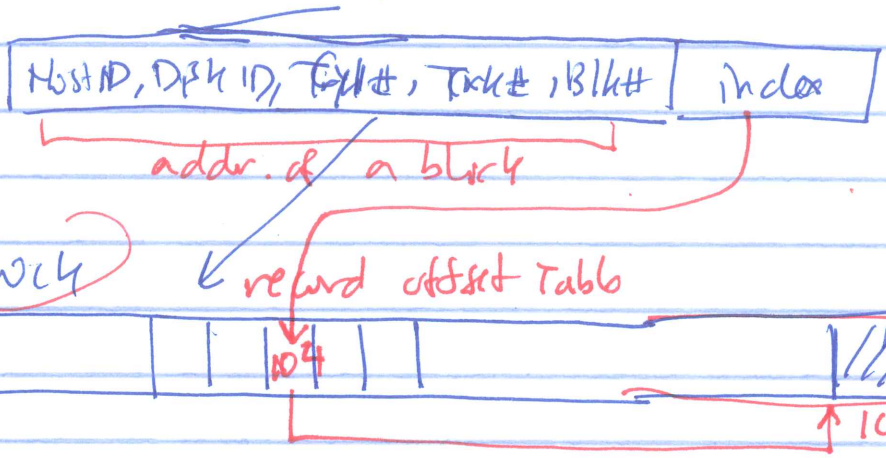
Example 3:

Recall that Block Header stores record offsets:

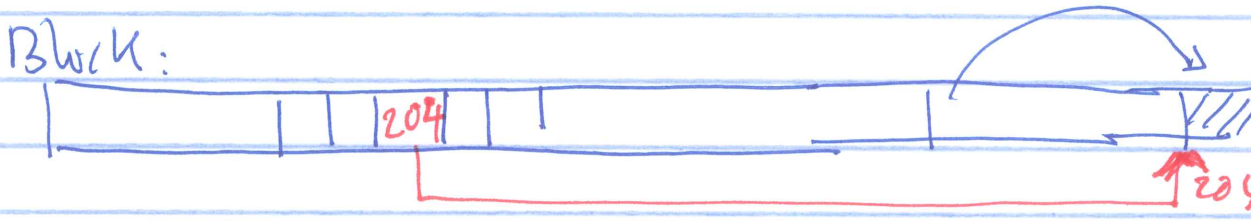


Advantages of this record addr. structure:

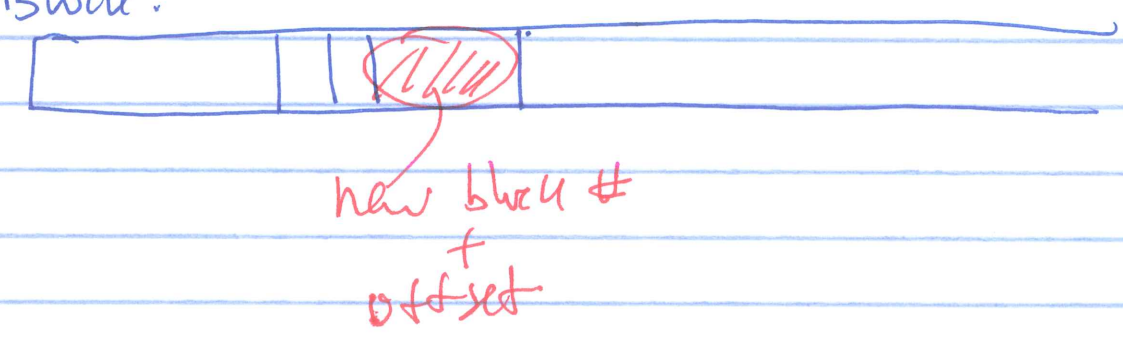
Addr. to identify a record:



(1) We can move the record around within the block:



(1') (2) Record can be moved between block using larger "forwarding addresses"!



(2) We can handle

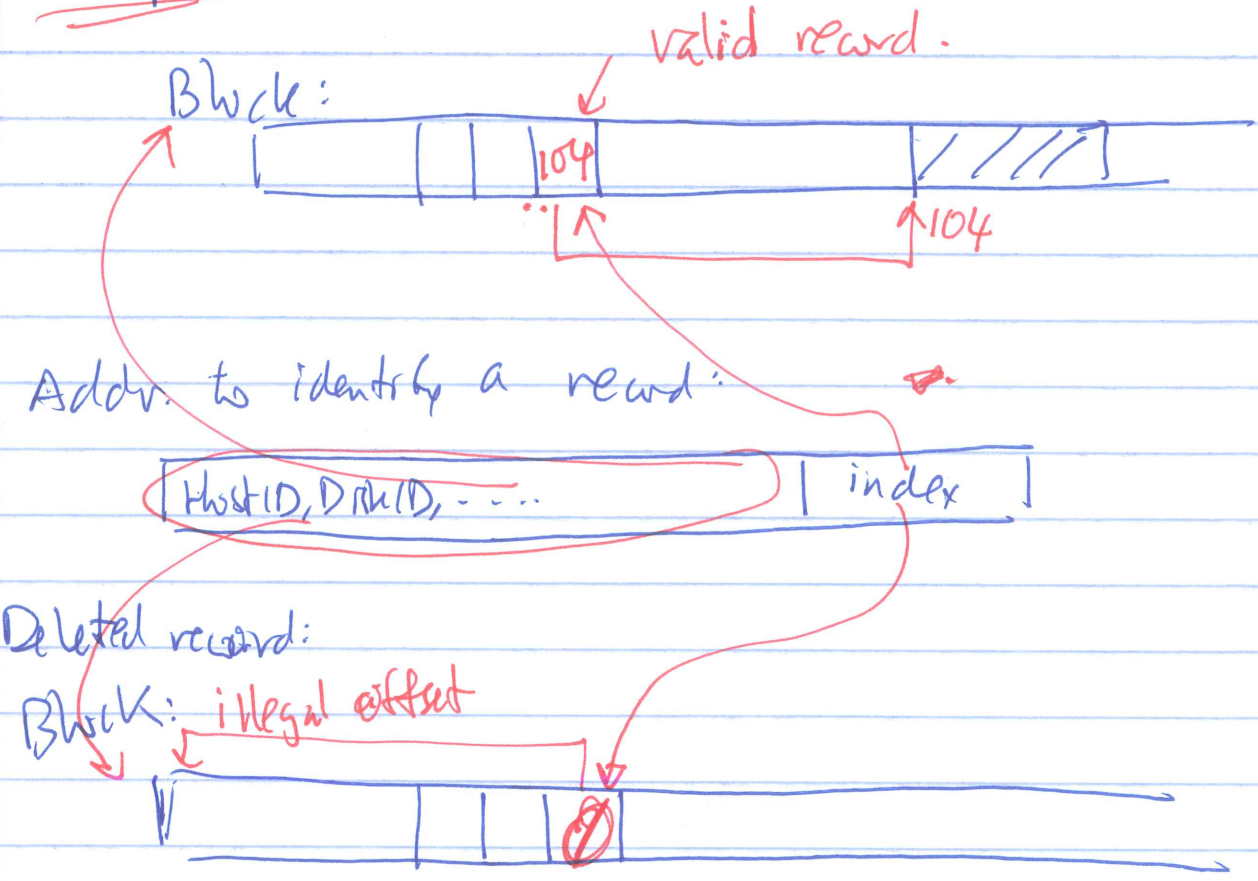
Record deletion

by leaving a

tombstone (= special value like "null")

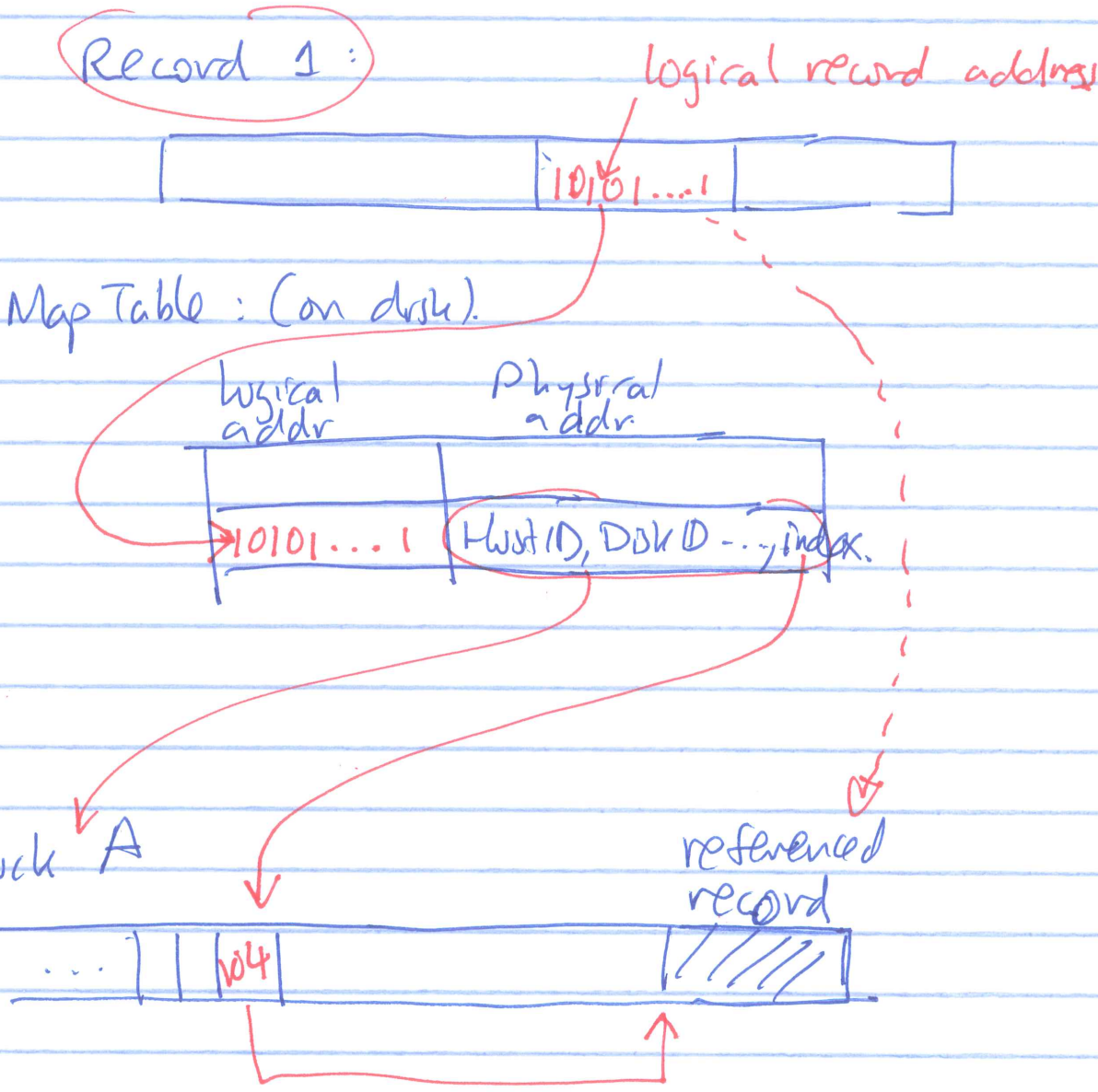
to indicate the record has been deleted.

Example:

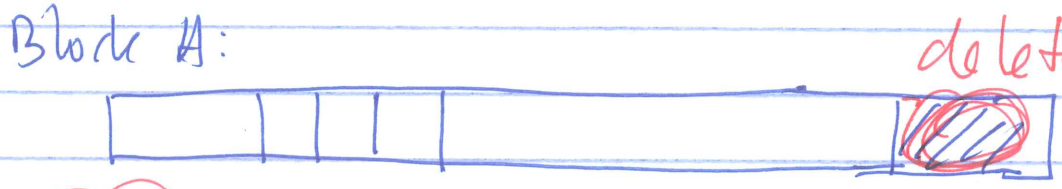


A Note on deleting records.

Recall: Some storage techniques use record addresses.

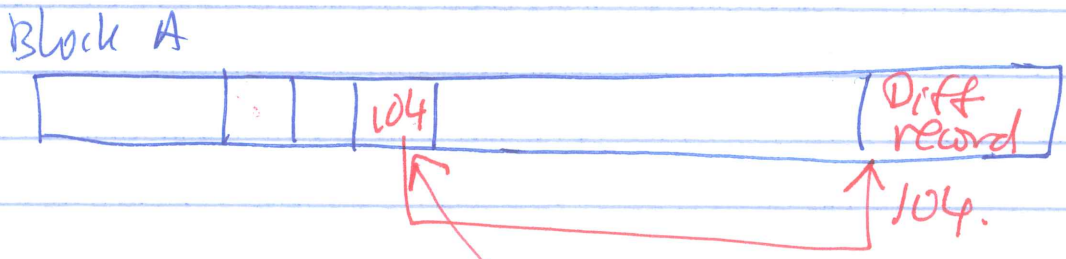


- When we DELETE a record:



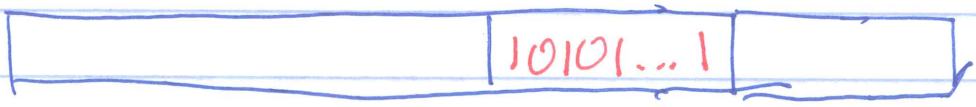
The space for
This record can be REUSED later.

Resulting in:

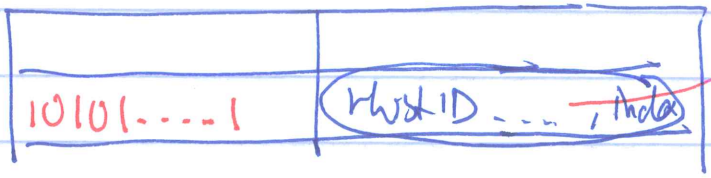


- Now, the "old reference" to the record:

Record 1:



Map Table: (on Disk).



Will now reference to

a different record !!!

- One way to handle "record deletion" is: use a "tomb stone" marker

Record I: