Locating records by the DBMS

- Fact: pointers (addresses) are part (stored) ~~in~~ of records.
- (This is not typical in ~~R~~ tuples of a relation)
- But: this is common for tuples of an Object Oriented DBMS. (Object Relational DBMS).

- Pointers are also used in index files.

needs

- ~~The DBMS has a management system for pointers:~~

  ~~pointer to a block on disk = a physical address~~

  ~~pointer to a block in memory = a virtual address~~

quick translation

- Recall that:

  A database object (block/record) is identified by:

  (1) Database address of object (= logical/physical address) when object is on Disk.

  (2) A virtual addr. in memory when object is read/stored in memory.

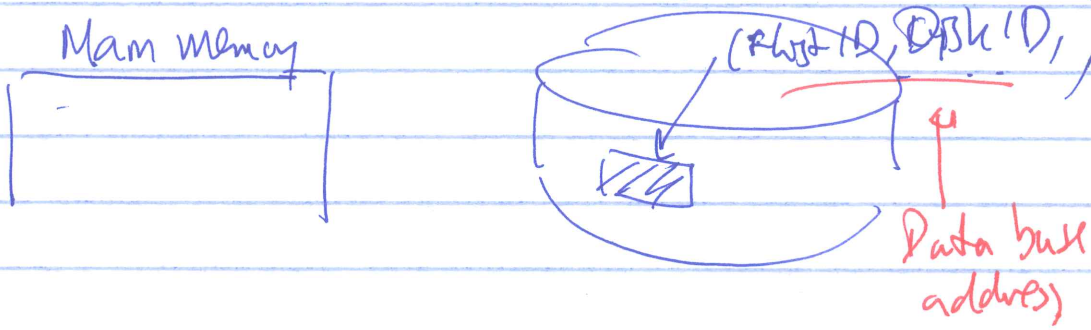
  (The object is still on disk, but the one on disk is NOT used until the in-memory copy is written back to disk).

- The DBMS has/needs a management system to convert:

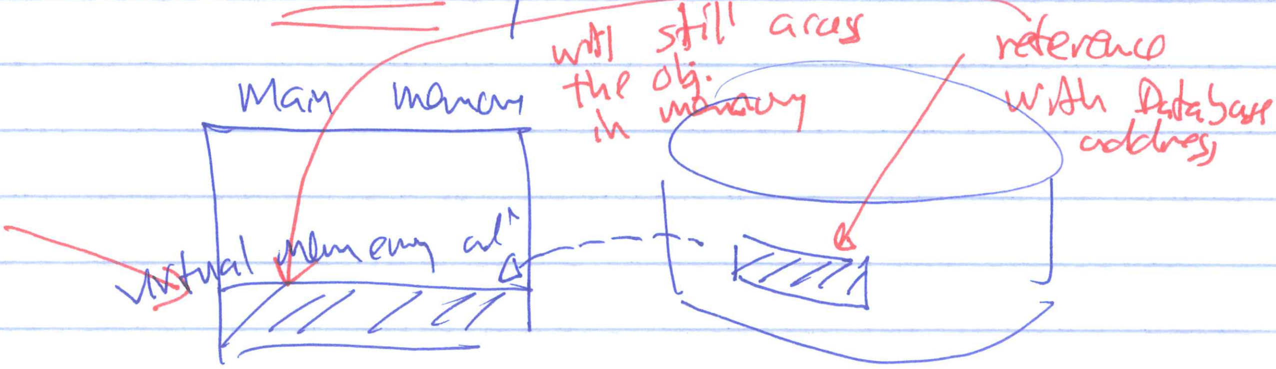
  Database address ⟷ virtual address

Observation:

(1) When a database object (block/record) is on disk, we can ONLY refer to the object using a database address.



(2) When a database object (block/record) is in memory:


Main memory
will still access the obj. in memory
reference with Database address
virtual memory ad

(A) we must use the object in memory.

(B) but: we can refer ,identify this object by:
- virtual memory addr
- Database address.

- So: when the DBMS uses

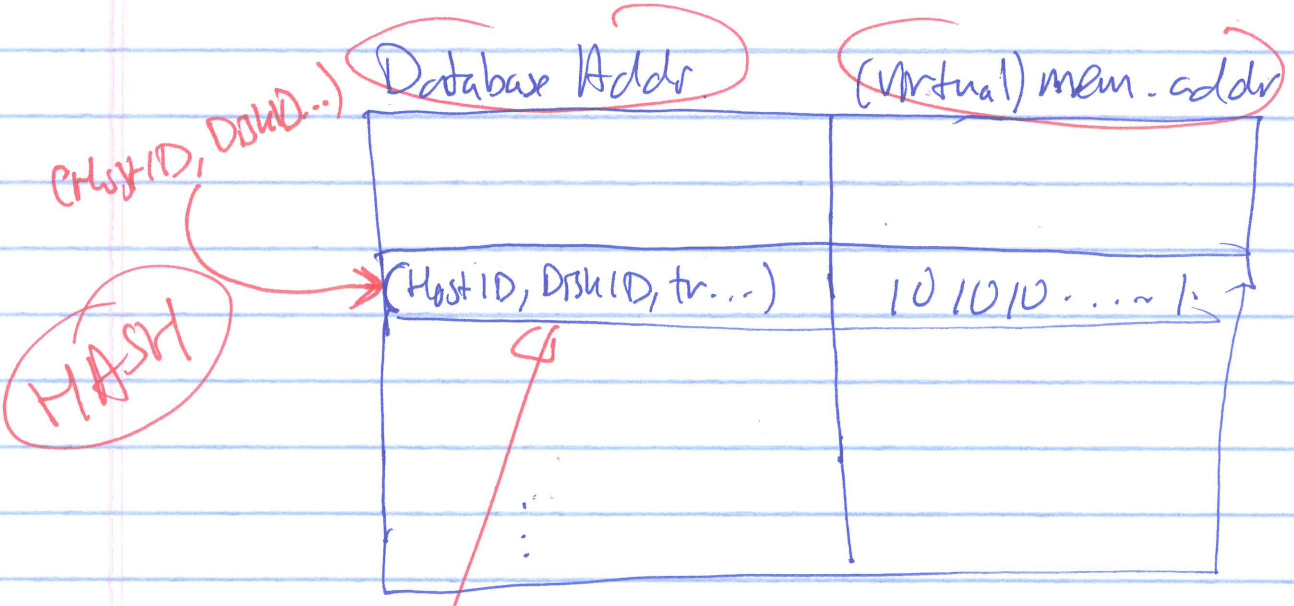
  a Database Address

  to reference an object that is in memory,
  we must:

  translate the Database Address

  → to a (virtual) memory addr!!!

- Translation table:

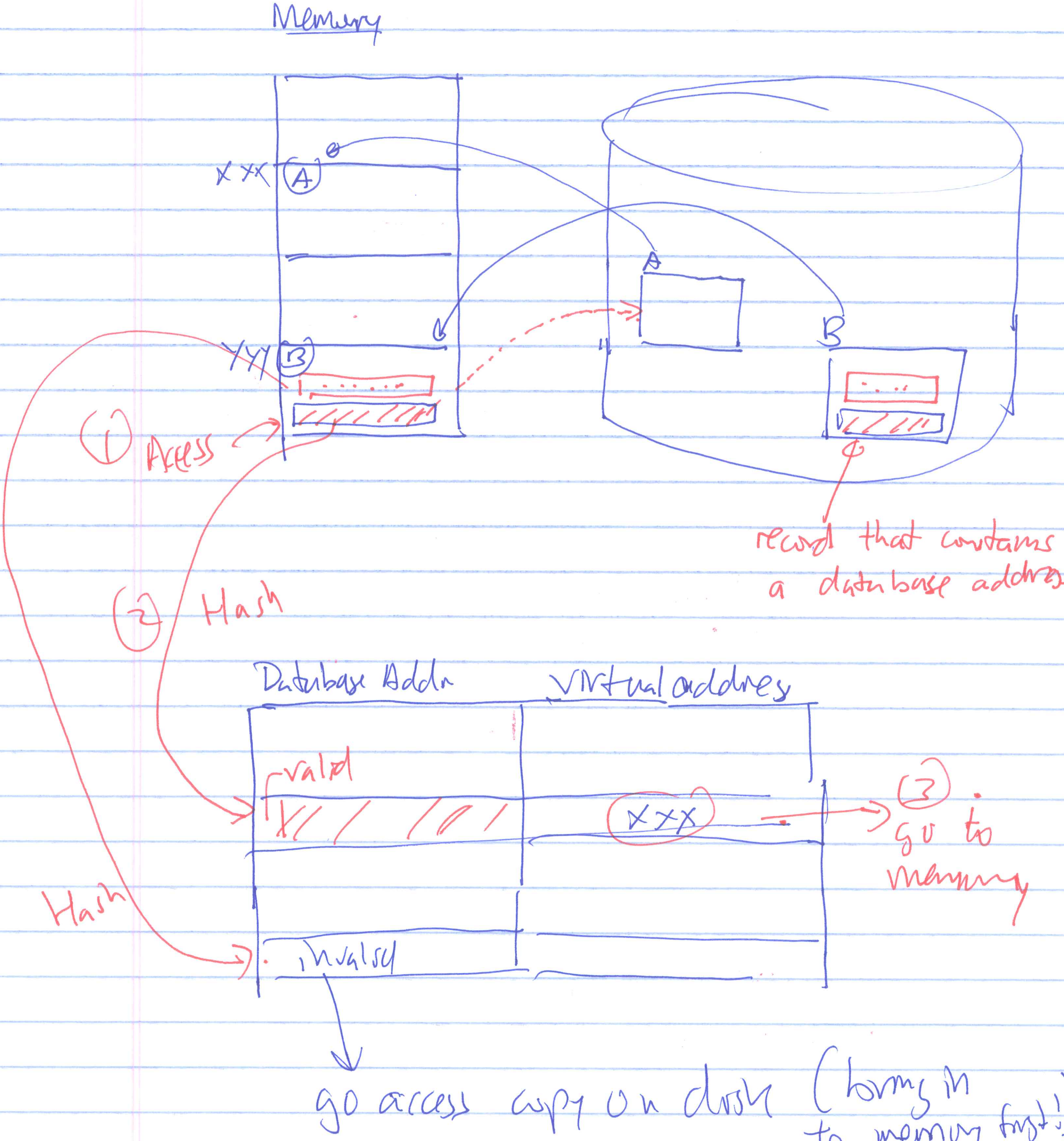


Database Addr	(virtual) mem. addr
(HostID, DBMID, tr...)	101010....1
⋮	

Table is SPARCE (only objects in memory will have a VALID entry!

⟹ use HASHING!!! speed + space savings

Naive Access to Database Objects.



go access copy on disk (bring in to memory first!)

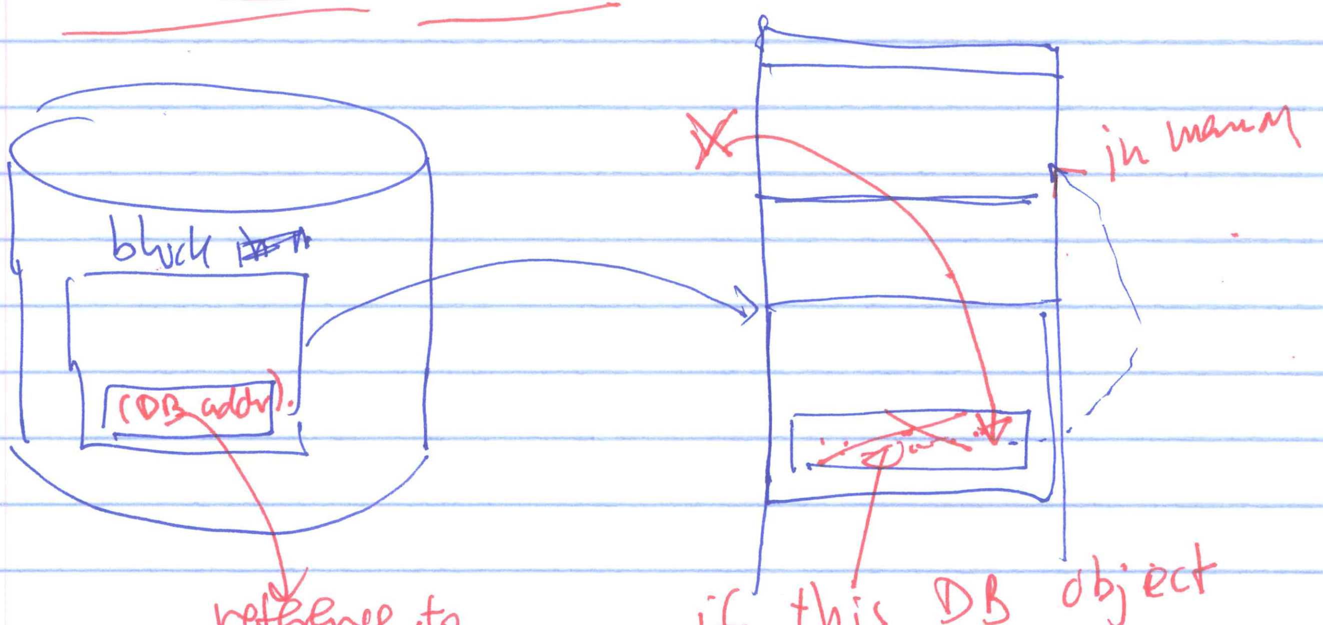
- **Advantage:** ① easy to implement

(only need a hashing table

② records/blocks are not "pinned" in memory (later)

Disadvantage: slow...

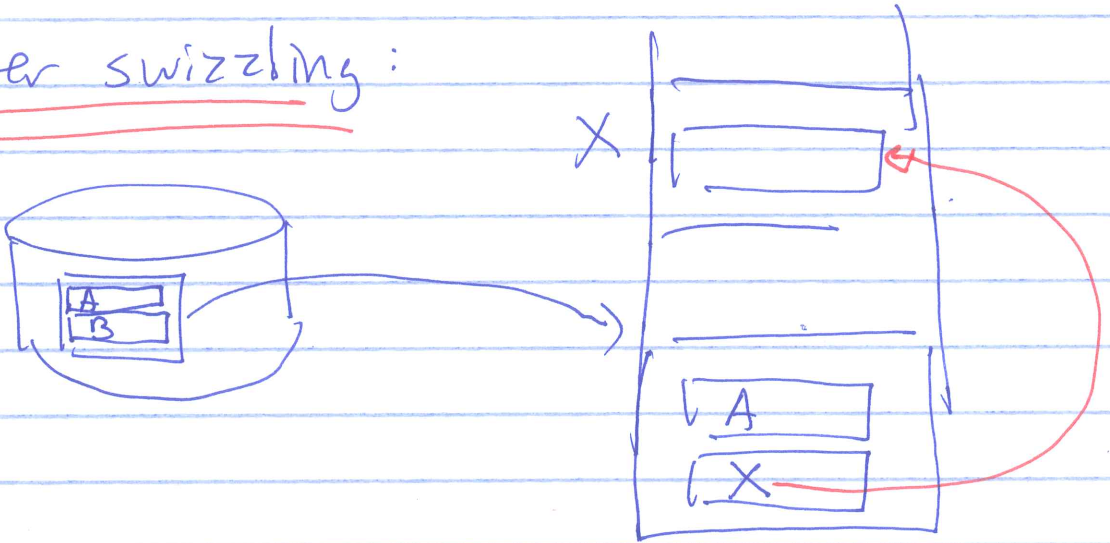
Faster Access: Trick (pointer swizzling)



reference to another block.

if this DB object is IN MEMORY, replace DB addr with virtual addr

Pointer swizzling:



When we move a block from disk to main memory, the pointers (= Database addresses) in the block may be "swizzled"

//

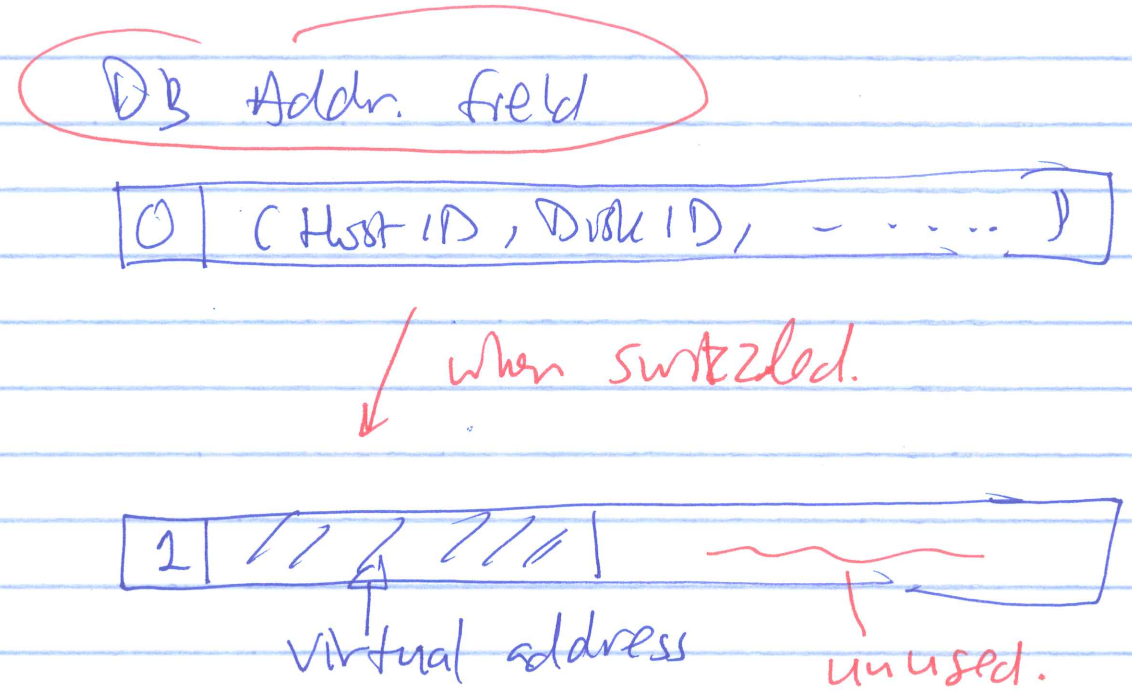
translated (using the map Hash table, from Database Address

⇓

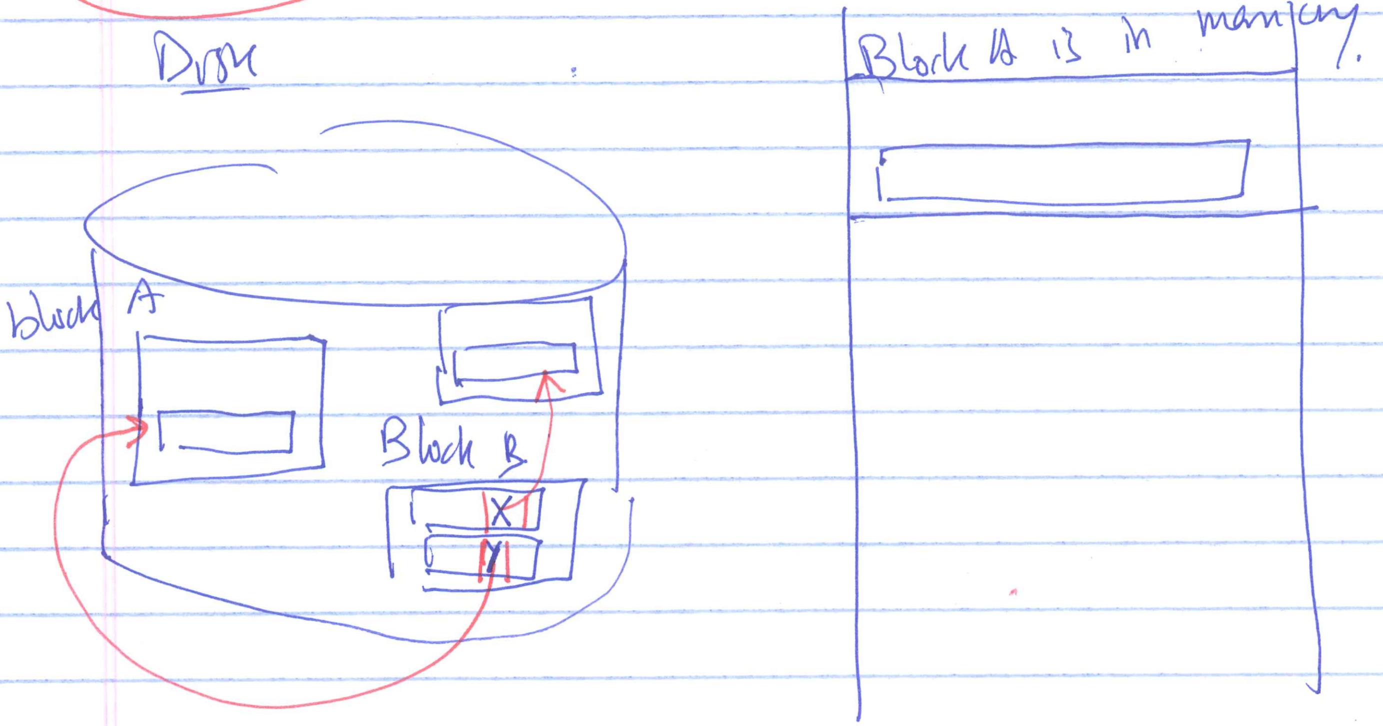
virtual address.

Implementation:

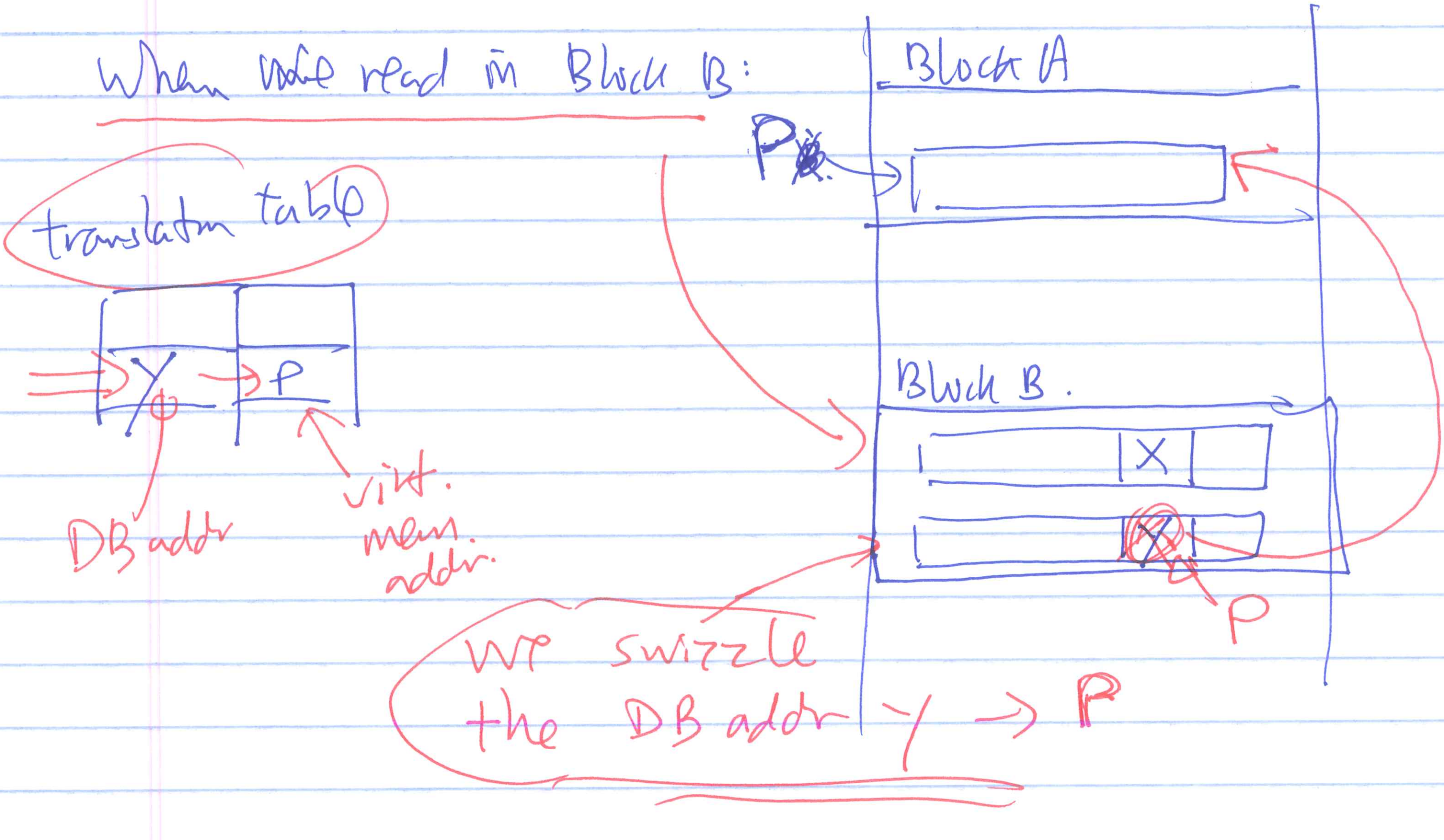
The ~~data~~ record field for a DB addr. is expanded with 1 bit:



Example:



When we read in Block B:



We swizzle the DB addr Y -> P

Swizzling Techniques:

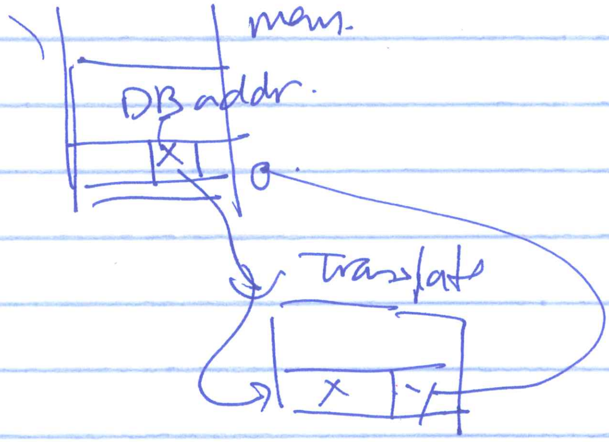
(1) **Automatic** Swizzling

- Swizzle all DB addr → mem.add
  inside a block
  when the block is FIRST
  read in to memory.

(2) **On Demand** swizzling

- Leave all DB addr. unswizzle
  when block read in.

- When DB addr. is used.
  (first time)



Swizzle (so next accesses will be fast).

Implementing automatic swizzling:

- Information required:

  Knowlegde on the location of every DB address type field in a block.

  (1) Block holds records of one schema

  ⇒ Schema will tell us the location of the DB addr. fields

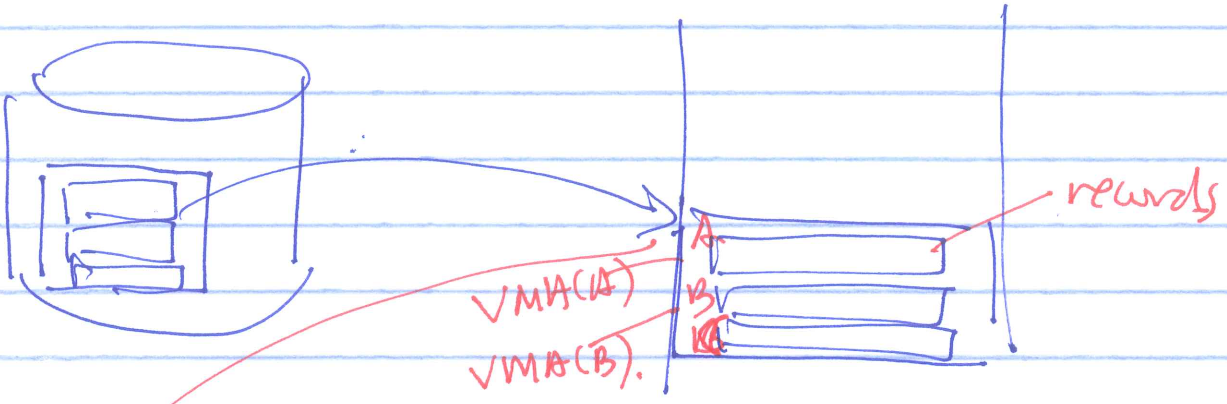
  (2) Block used for indexes (later)

  - Structure is known
  - DB addr's can be located

  (3) Structure unknown ...

  → put a list of offsets in Block header that contain locations of DB addresses.

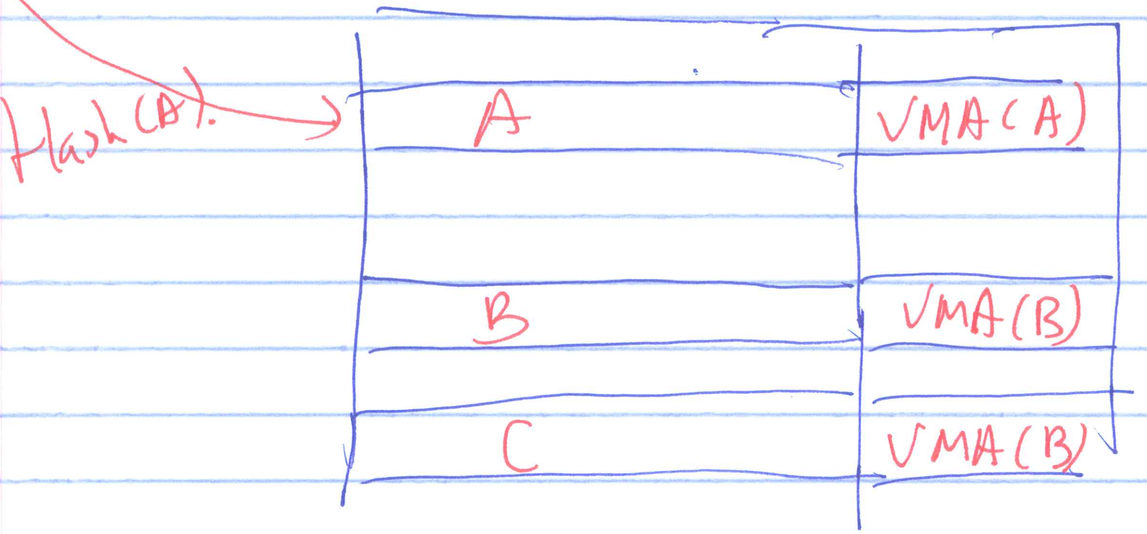
Implementing on-demand swizzling:



When we read in a block:

Enter translation entries for every record in the block:

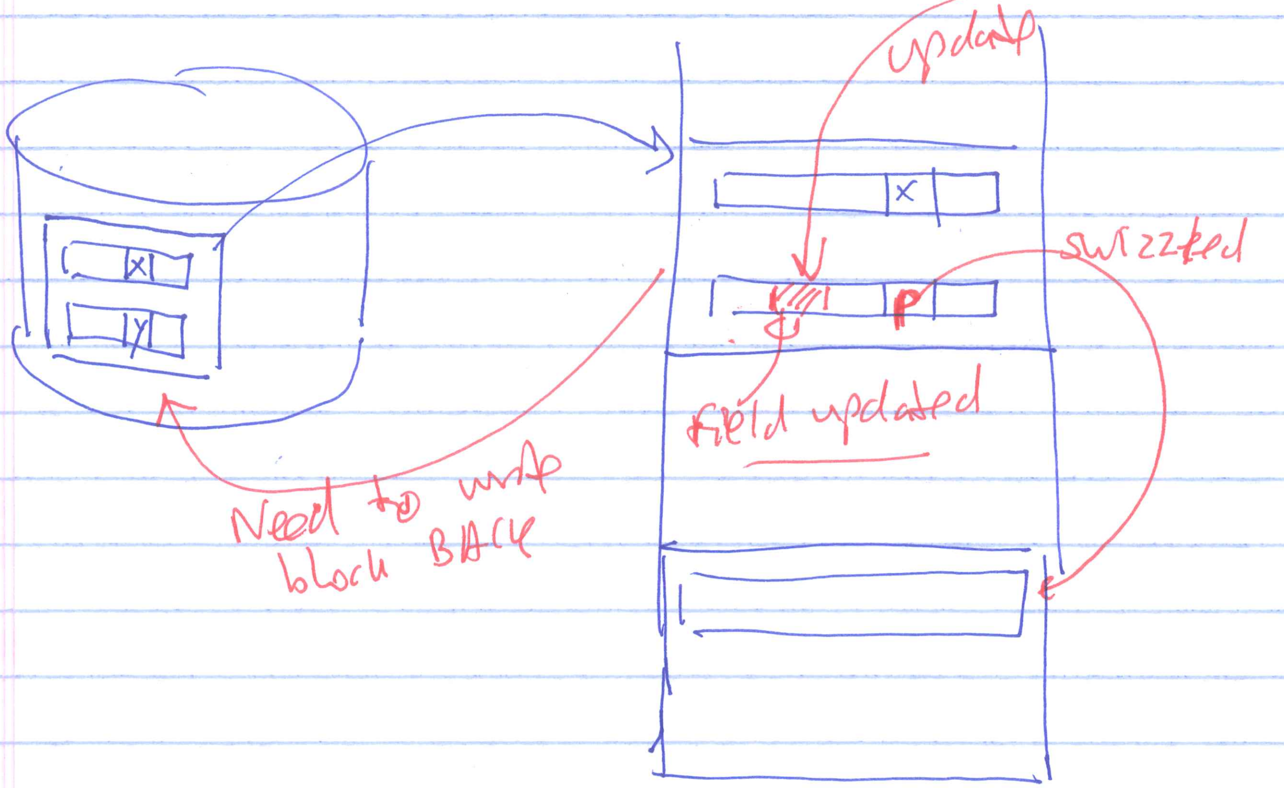
translation table



A	VMA(A)
B	VMA(B)
C	VMA(B)

Problem created by Pointer Swizzling

(1) Problem: the block in memory is changed:



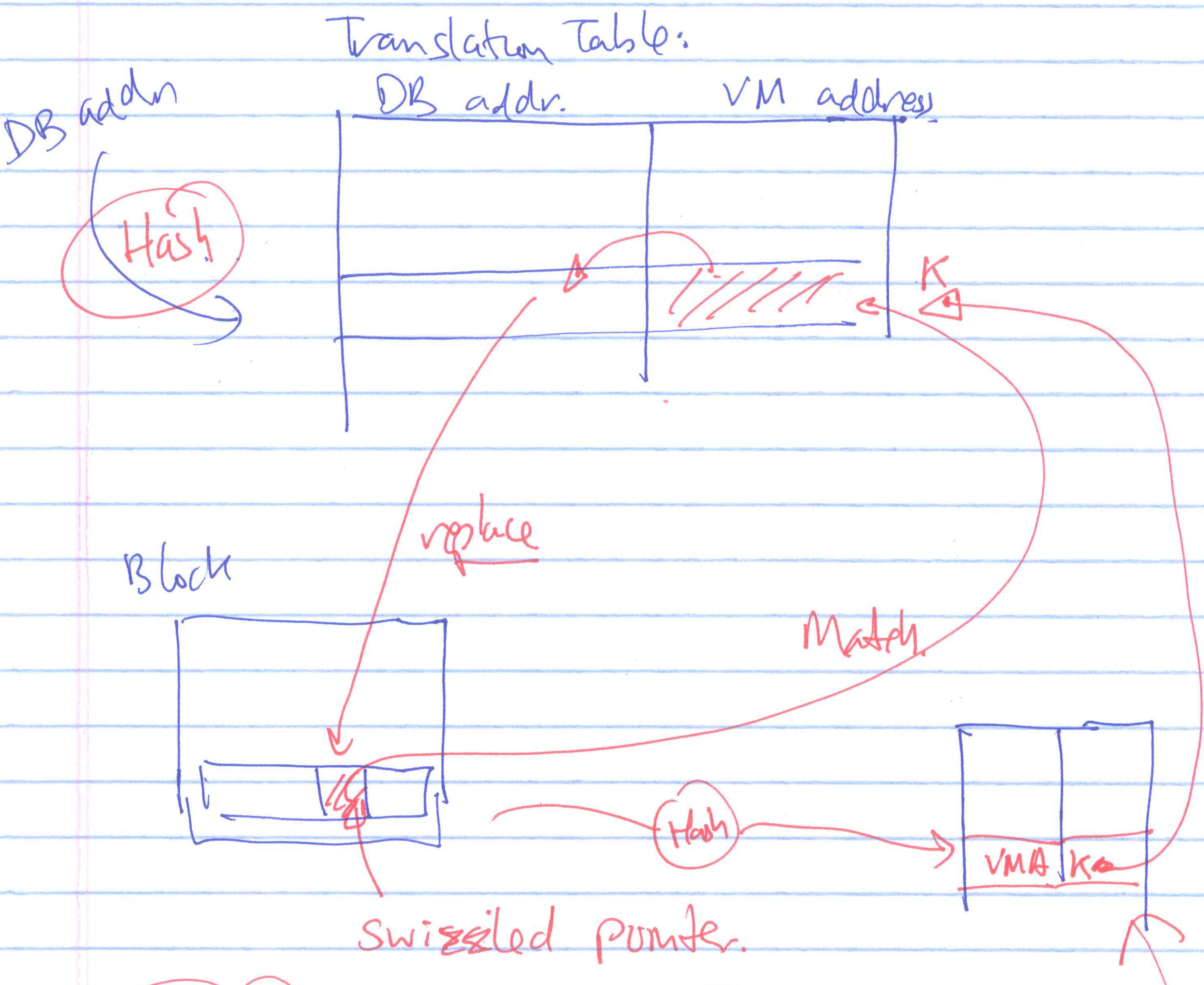
⇒ We must UNSWIZZLE the

Virtual addr. P ⇒ DB addr. y

BEFORE writing the block

Back to the DISK !!!

Naive Solution:

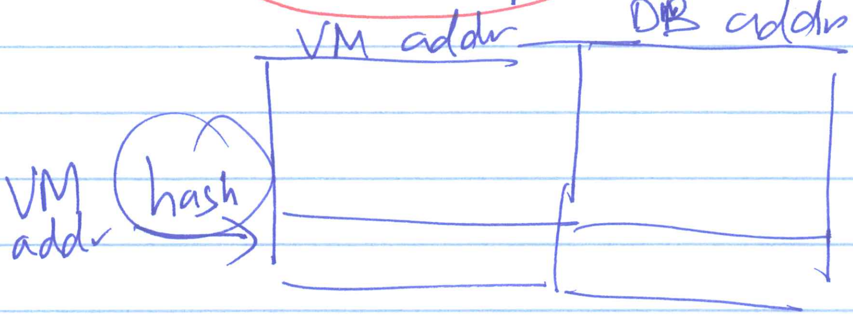


Problem: requires a search to find entry!

Faster ~~Possible~~ solution: Add an index (Hashing). based on virtual memory addresses.

Summary: faster technique to unswizzle a VM address => DB address

① Use a hash map:



In Chapter 14, the book will discuss an indexing data structure to ~~handle~~ implement this search faster.

pp: ???

Problem created by pointer swizzling

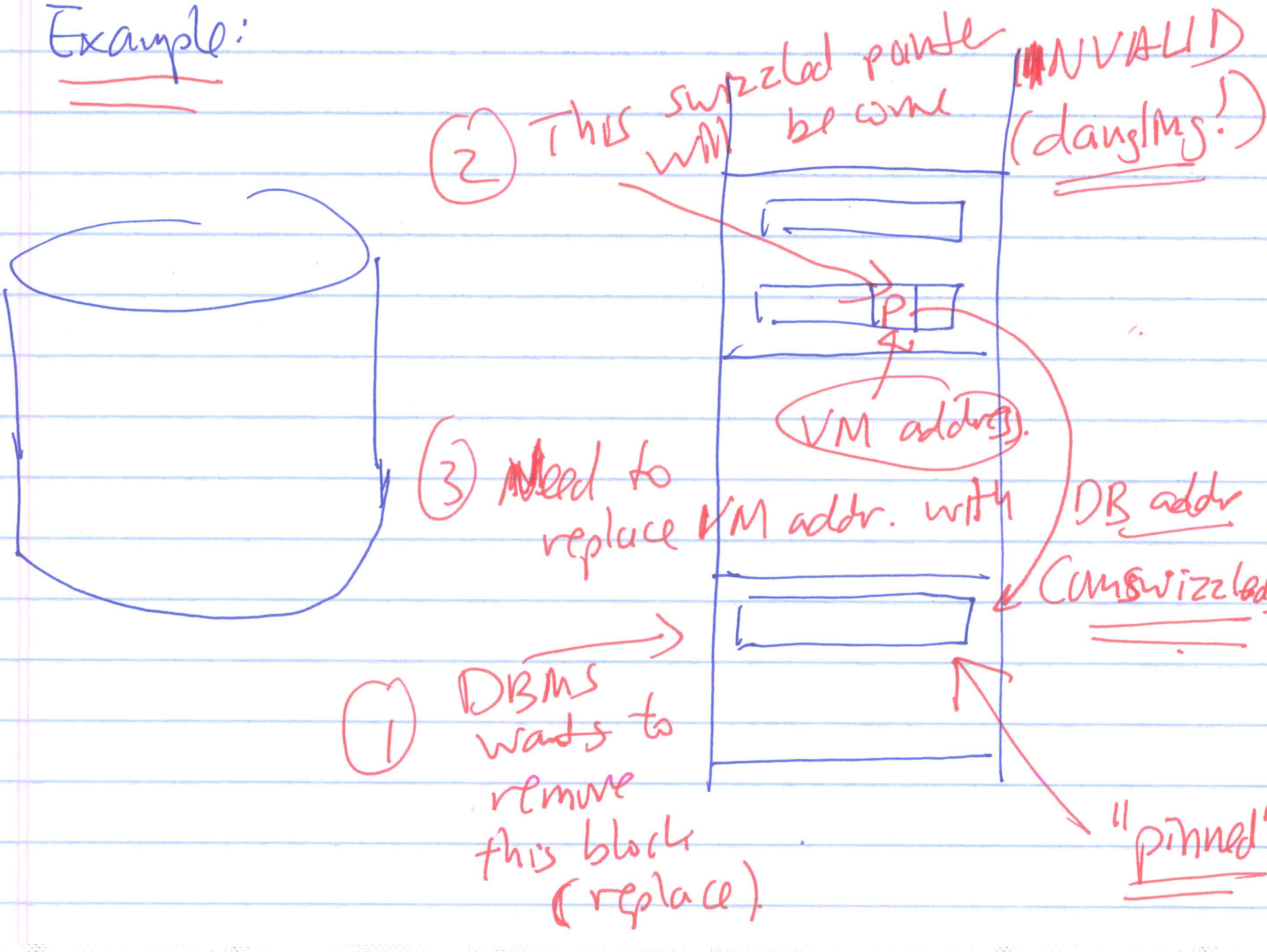
(2) Problem: a block (that the DBMS want to remove from memory)

cannot be removed from memory IF:

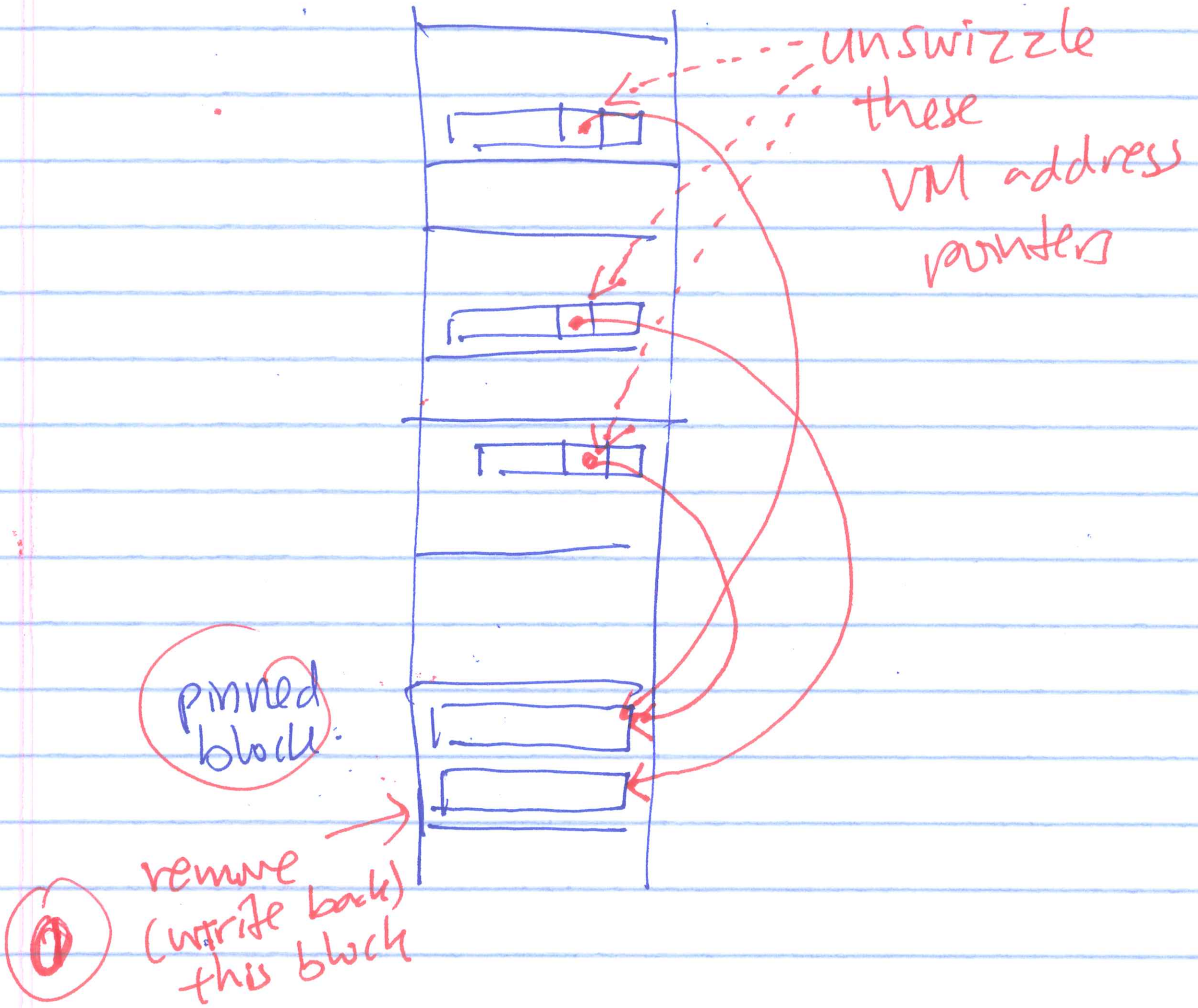
some records in block if referenced by swizzled pointers in other (in memory) blocks.

"pinned"

Example:


(2) This swizzled pointer will become INVALID (dangling!)
P
VM address
(3) Need to replace VM addr. with DB addr (unswizzled)
(1) DBMS wants to remove this block (replace)
"pinned"

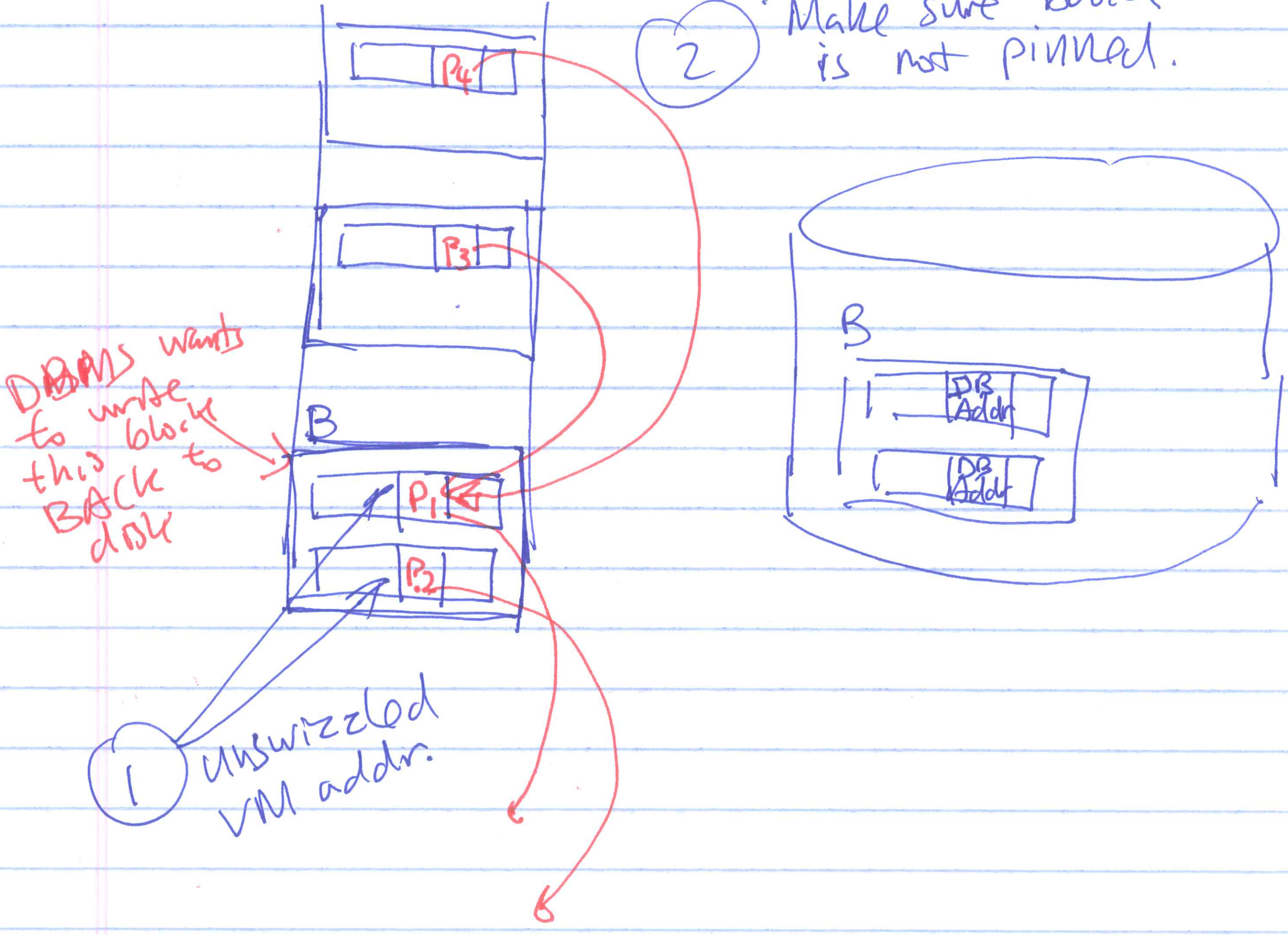
How to unpin a block:


unswizzle these VM address pointers
pinned block.
① remove (write back) this block

Conclussion:

write back

When the DBMS wants to release/remove a block from memory:



2
Make sure block is not pinned.
P4
P3
DBMS wants to write this block BACK to disk
B
P1
P2
1
Unswizzled VM addr.
B
DB Addr
DB Addr

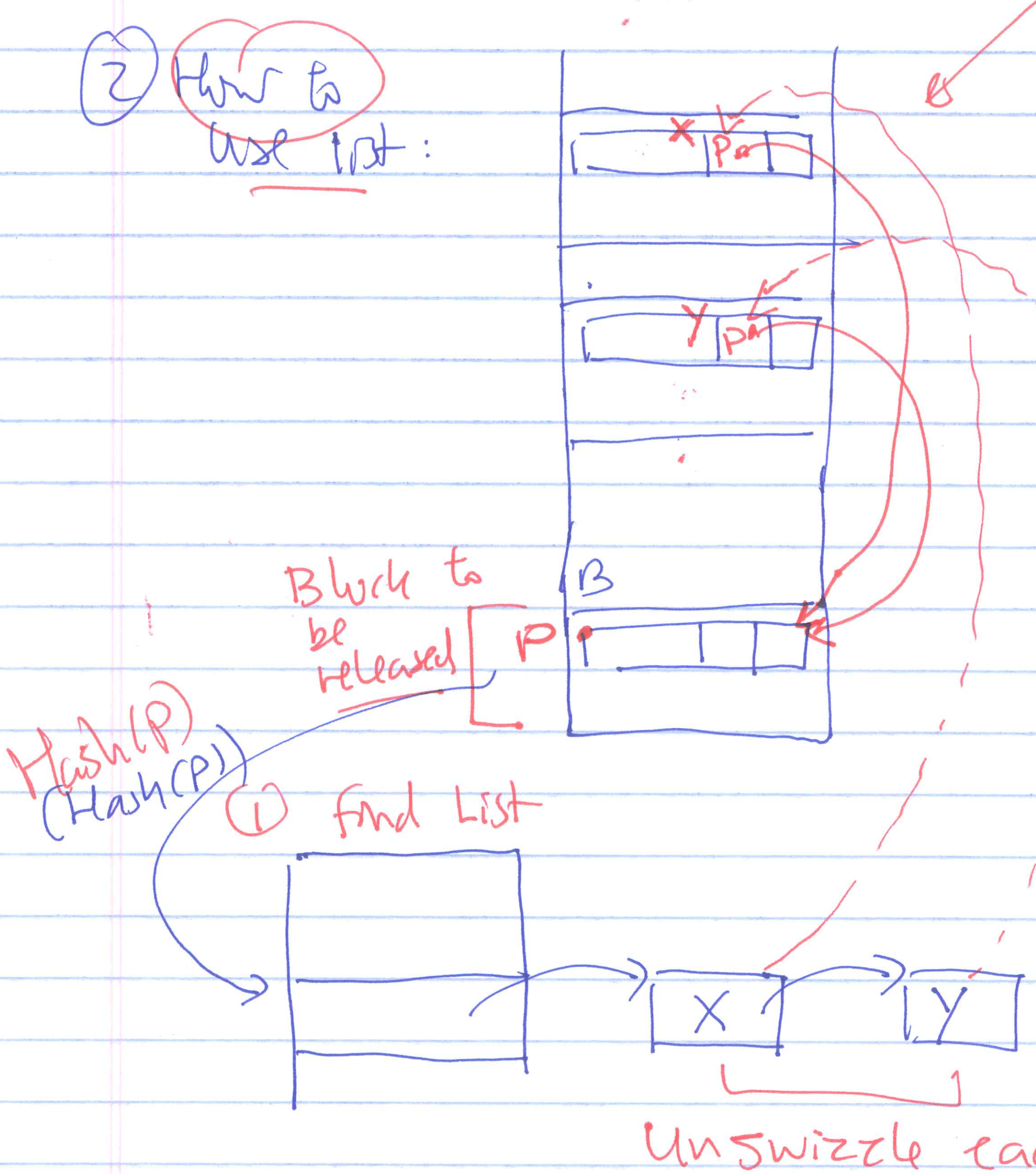
Implementing unpin efficiently

① Keep a (linked) list of

memory addresses

of swizzled addresses.

② How to use list:

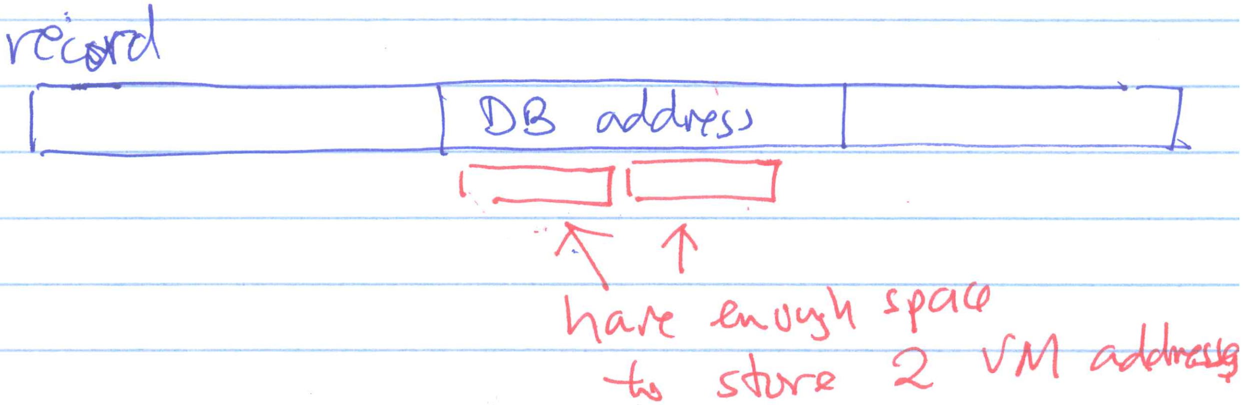


Each time you swizzle an addr, add the VMAddr of DM.addr to linked list (at the correct list!!)

Unswizzle each VM addr. at these VM addrs

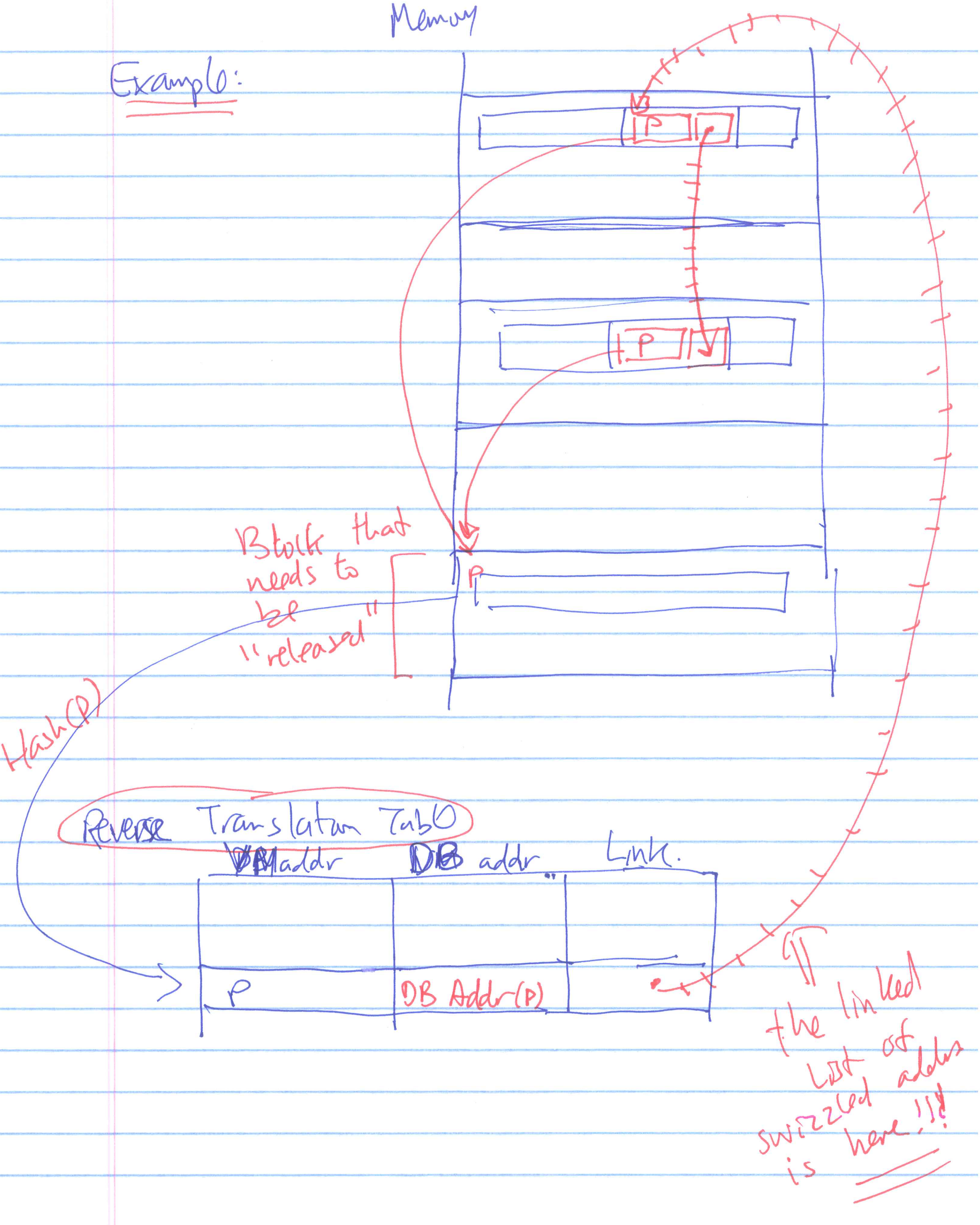
More efficient implementation of the Linked List:

- often: the DB address is much larger than a VM address:



- We can construct the Linked list of swizzled address using the space that store the DB address.

Example:


Memory
P
P
P
P
Block that needs to be "released"
Hash(P)
Reverse Translation Tabl
the linked List of swizzled addrs is here!!!

VM addr	DB addr	Link.
p	DB Addr(p)	