Modifying records

- Modifications:
  1. Insert a new record
  2. Delete an existing record
  3. Update (i.e., delete + insert)

- Insertion:
  unordered records: (easy).
  - Find a block with an empty space → insert record there
  - Get a new block, insert record into the new block.

No record need to be moved!!
Ordered records: + insertion

- Insertion requires "moving" of some records.

Result of movement:

Block contains inserted record:

New record goes here

Another block with record that points to X

Recall there are many ways to find record in block:
(1) Block phys. addr + offset
(2) Block phys. addr + key

Assume for sorted records: always use Key (Binary Search!!)
Ordered records.

- Insert into an existing block:

  Before:
  \[\text{block header} \rightarrow \text{record } 4 \rightarrow \text{record } 3 \rightarrow \text{record } 2 \rightarrow \text{record } 1\]
  Unused

  New record goes in here

  Slide records

After insertion:

\[\text{block header} \rightarrow \text{new} \rightarrow \text{record } 4 \rightarrow \text{record } 3 \rightarrow \text{new} \rightarrow \text{record } 2 \rightarrow \text{record } 1\]

- If no space in the block:

  1. Find space in a "nearby" block.
     - Slide
  2. Use "overflow" blocks.
Insertion causes record to move to neighbor block

**Before:**

```
B1   1 2 3  \ | /  " "  C  B  A
          ^    ^
          |    |
          |    |
          V    V
          B2   1 2  ED
```

Insert \( \times \) here

Leave forwarding pointer !!!

Another Block:

```
B1   B1C
          ^
          |
          V
B2   B1C
```

**After insertion:**

```
B1   B1X A
          ^  ^
          |  |
          V  V
          B2   E  D  E
```

When C gets pushed out, we must leave a forwarding pointer in original block.
Using Overflow Blocks:

Primary Data blocks:

Black Header - Block header contains an "overflow block pointer".

Overflow Blocks

Chain of overflow Data blocks

This organization is "cleaner".

(Note: these 2 approaches are similar to the overflow approaches used in Hashing: Open addressing!!! separate chaining!!!)
Deleting records:

- Unsorted: (+ variable length)

\[ B1 \quad D \quad I \quad Z \]
\[ A \quad B \quad C \quad E \quad I \quad X \quad B \quad A \]

(reclaim space)

(Tombstone (Ø))

leave a tombstone for B

Because:

Another block B2

\[(B2, 1)\]

Can cause reference to deleted record to refer to

Another record !!!
- You cannot compact or delete a record if you use offset (as record pointer).

Block phys. addr + offset in record = offset.

- In that case, deleting a record must set a bit in record header to mark it as a tombstone.
- Database Address is a logical record addr.

Recall:

DB addr

physical record addr

logical record addr

map table (in draw)

```
10101... phys record addr
```

tombstone

Delete a record → replace phys. record addr. in map table with a tombstone entry.

( special bid or special phys. addr. value )
Variable length record:

If you use

( block phys. addr, key )

we can delete the record with the key.

⇒ key not found in block

(or overflow block)

means: record does not exist.
Update:

Fixed length records: easy
- in place update.
(no dangling record references)

Variable length records:

Handle it as an

(delete + insert)

may result in

using forwarding references

But

no need to create the

tombstone ! ! !.