

Εργασία 1

2024 - 2025

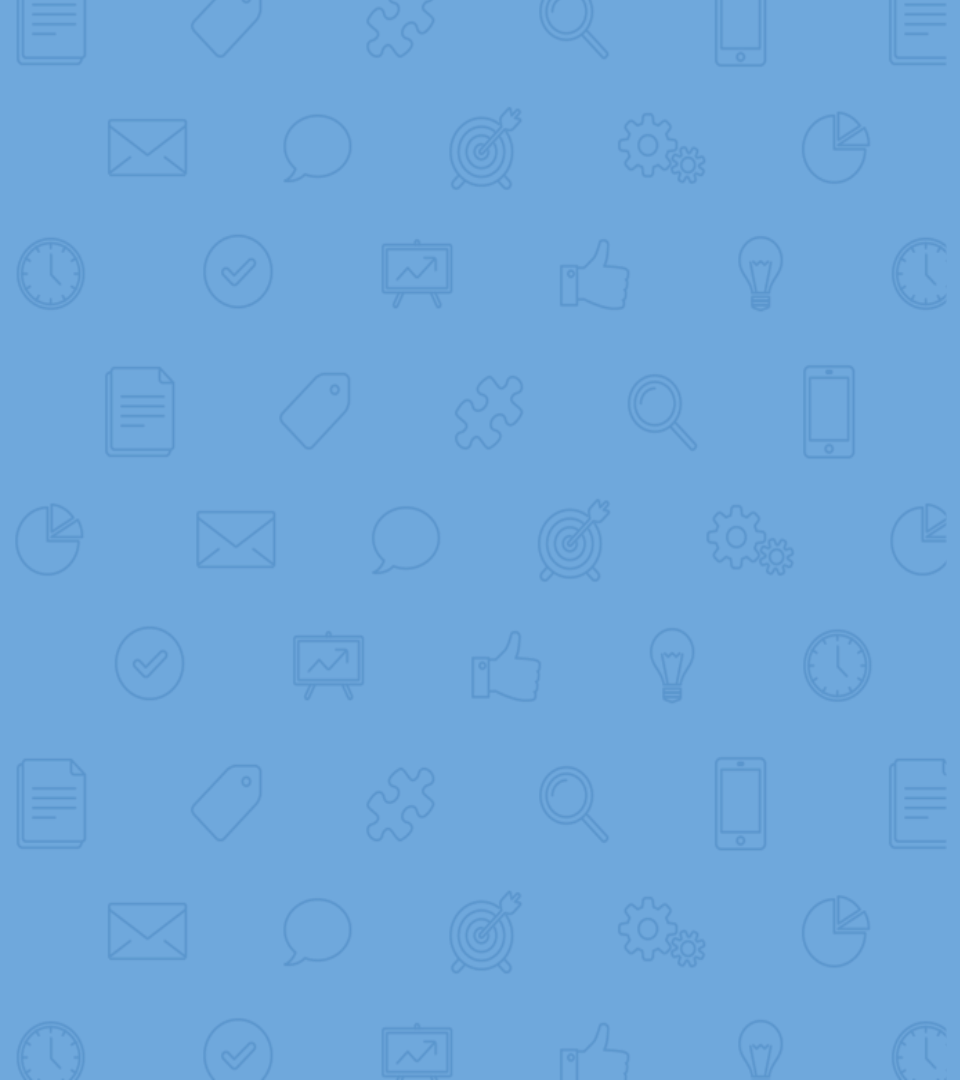
Περιγραφή εργασίας

- ▶ Σκοπός της εργασίας αυτής είναι η κατανόηση της εσωτερικής λειτουργίας των Συστημάτων Βάσεων Δεδομένων όσον αφορά τη **διαχείριση σε επίπεδο μπλοκ (block)** αλλά και ως προς τη διαχείριση σε **επίπεδο εγγραφών**.
- ▶ Η υλοποίησή των συναρτήσεων σας θα γίνει **πάνω από το επίπεδο διαχείρισης μπλοκ**, το οποίο σας δίνεται έτοιμο ως βιβλιοθήκη (BF). Καλείστε να γράψετε το σώμα των συναρτήσεων για τις οποίες σας δίνονται τα πρότυπα. Η δομή που θα υλοποιήσετε είναι **αρχείο σωρού**.



Επίπεδο block (BF)

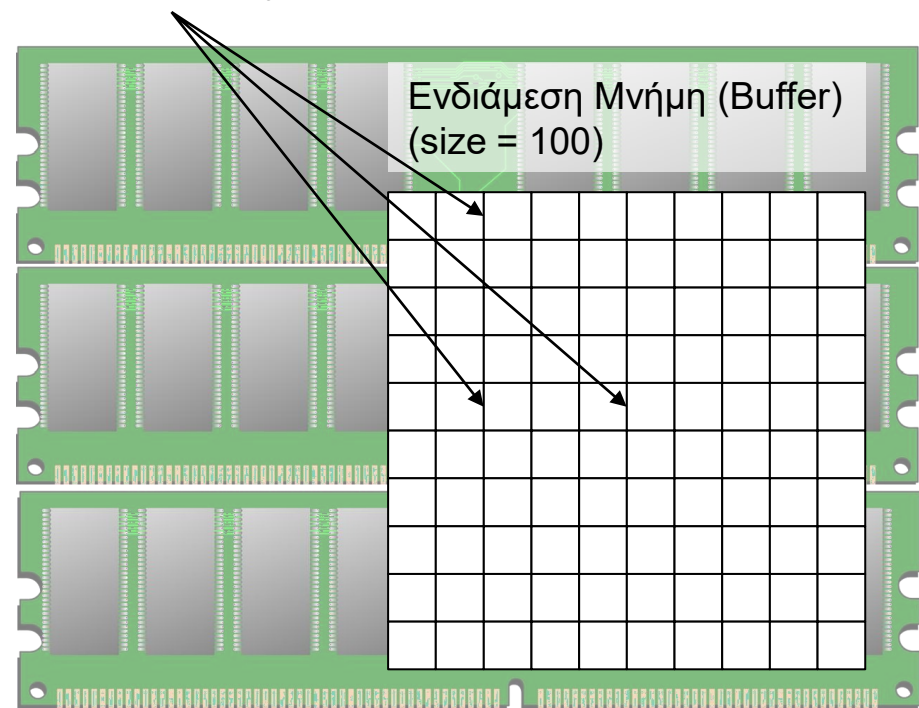
- ▶ Το επίπεδο block (διαχειριστής ενδιάμεσης μνήμης-buffer manager) κρατάει block δίσκου στην μνήμη. Κάθε φορά που ζητάμε ένα block δίσκου, το επίπεδο BF πρώτα εξετάζει την περίπτωση να το έχει φέρει ήδη στην μνήμη:
 - ▶ Αν το block υπάρχει στην μνήμη τότε δεν το διαβάζει από τον δίσκο
 - ▶ Σε αντίθετη περίπτωση το διαβάζει από τον δίσκο και το τοποθετεί στην μνήμη

The background of the slide features a repeating pattern of various business and technology icons in a light blue color. These icons include a document, a tag, a puzzle piece, a magnifying glass, a smartphone, a pie chart, an envelope, a speech bubble, a target with an arrow, interlocking gears, a clock, a checkmark, a presentation board with a line graph, a thumbs up, a lightbulb, and a smartphone. The icons are arranged in a grid-like fashion across the entire background.

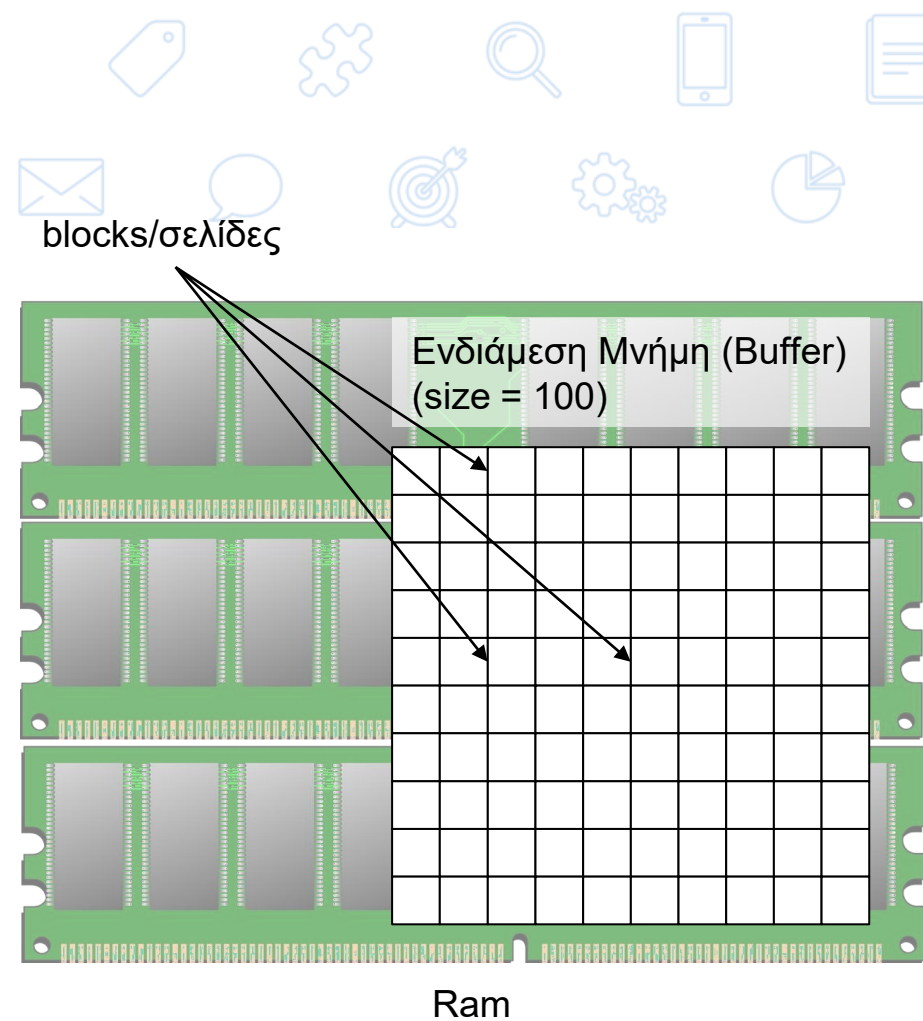
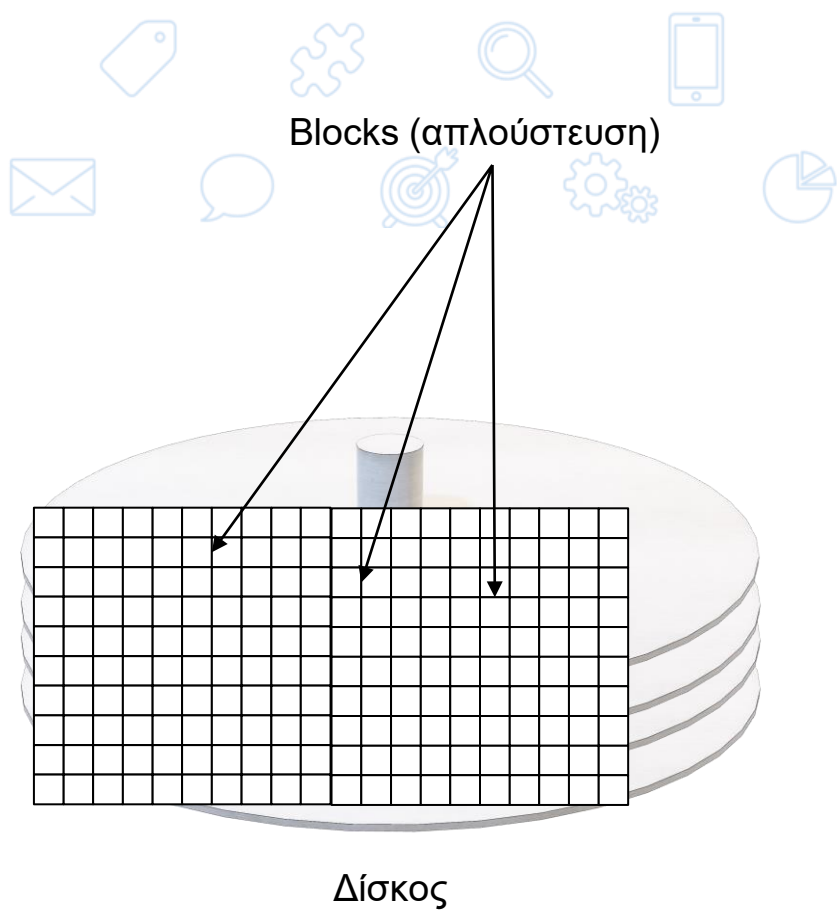
Δίσκος, Ενδιάμεση Μνήμη & Διαχειριστής Ενδιάμεσης Μνήμης



blocks/σελίδες

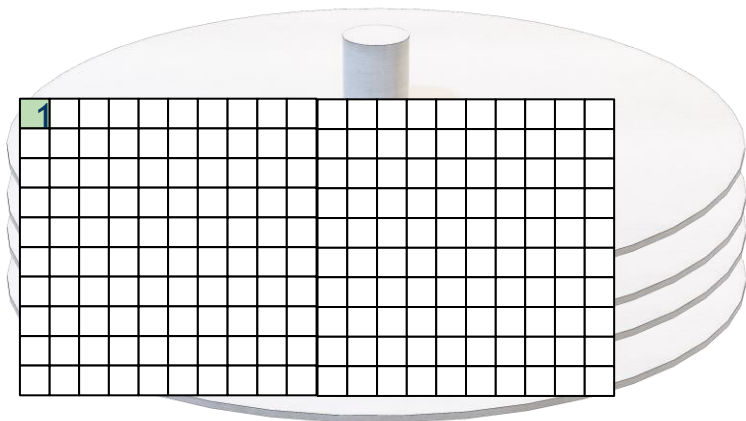


Ram

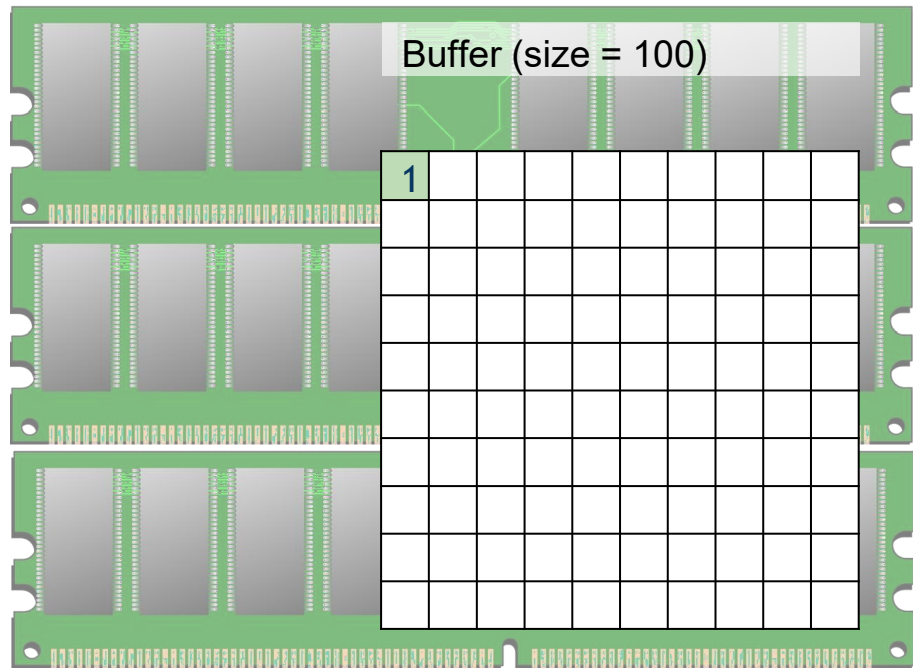


Όταν ζητάμε ένα block και δεν υπάρχει στον buffer αυτό

- **Διαβάζεται** από τον δίσκο και **γράφεται** στον buffer
- **Καρφιτσώνεται** (pin) στον buffer δηλαδή δεν μπορεί να απομακρυνθεί



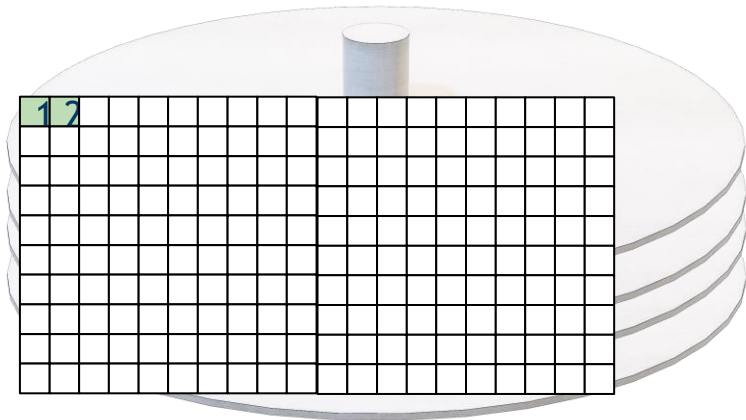
Δίσκος



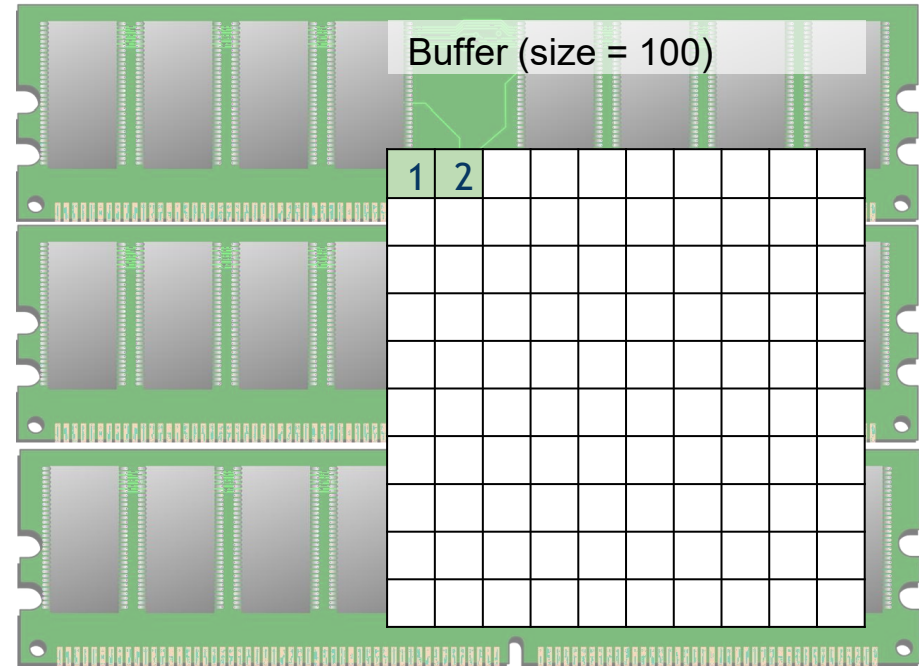
Ram

Έστω ότι ζητάω ένα ακόμα block

- Εφόσον και αυτό δεν υπάρχει στον buffer θα γραφτεί στον buffer
- Επίσης θα γίνει pinned

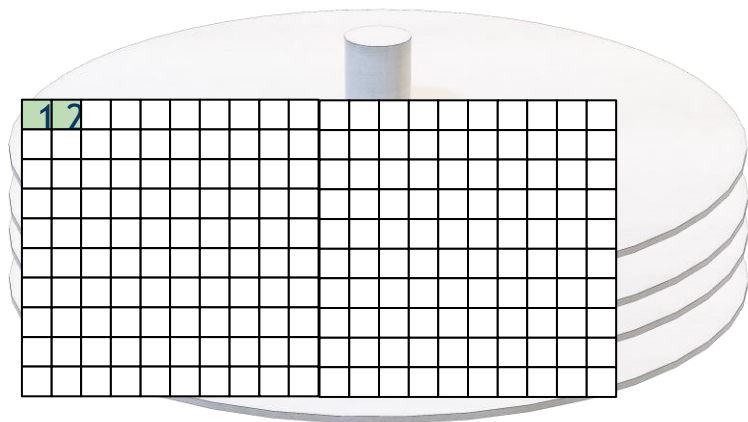


Δίσκος

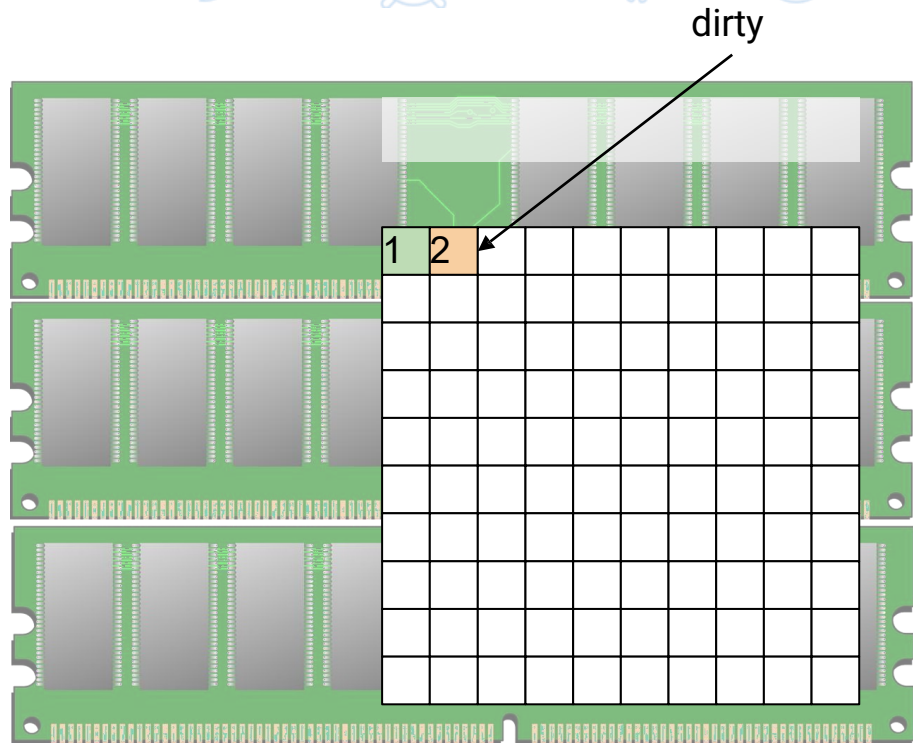


Ram

- Έστω ότι αλλάζω τα περιεχόμενα του δεύτερου block
- Τότε θα πρέπει να **ειδοποιήσω τον διαχειριστή** για την αλλαγή και ότι πρέπει να **αναλάβει να το γράψει στον δίσκο**

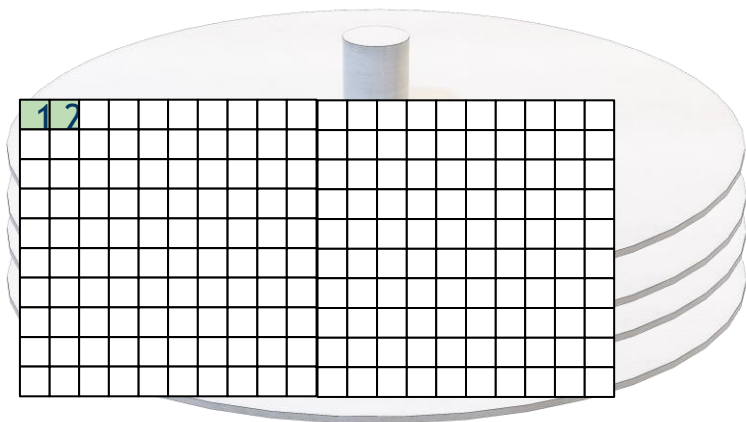


Δίσκος

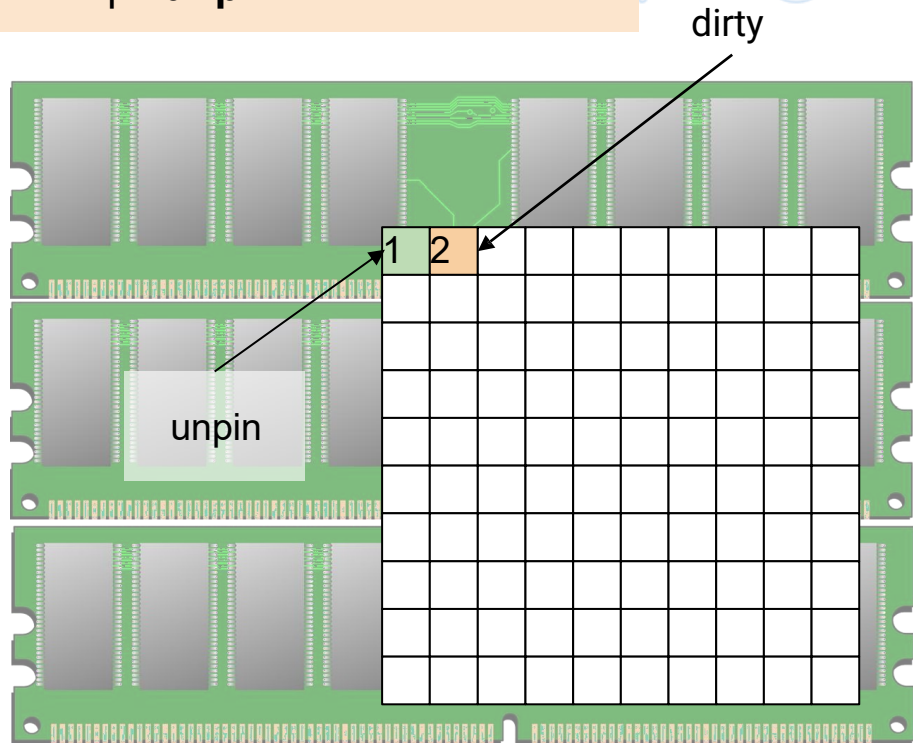


Ram

- Ο buffer έχει πεπερασμένο μήκος πολύ μικρότερο από τον δίσκο.
- Θα **γεμίσει** λοιπόν πολύ γρήγορα αν δεν επιτρέψουμε την απομάκρυνση στοιχείων
- Όταν ένα στοιχείο δεν το χρειαζόμαστε το κάνουμε **unpin**



Δίσκος



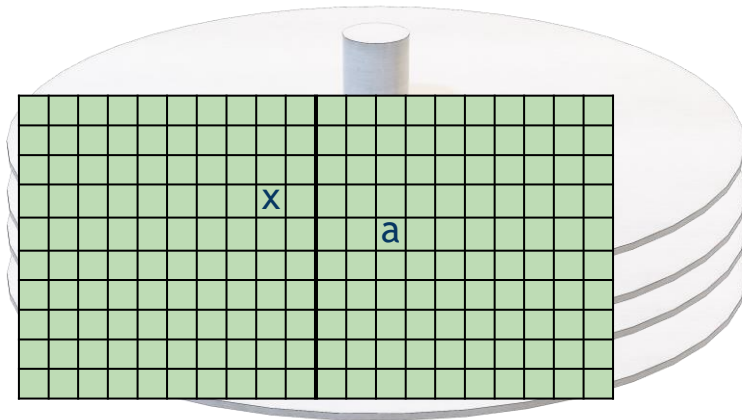
Ram

Ένα block
στον buffer
μπορεί να
είναι

- ▶ **καρφιτωμένο** (pinned)
- ▶ **καρφιτωμένο** (pinned) και **βρώμικο** (dirty)
- ▶ **μη καρφιτωμένο** (unpinned)
- ▶ **μη καρφιτωμένο** (unpinned) και **βρώμικο** (dirty)

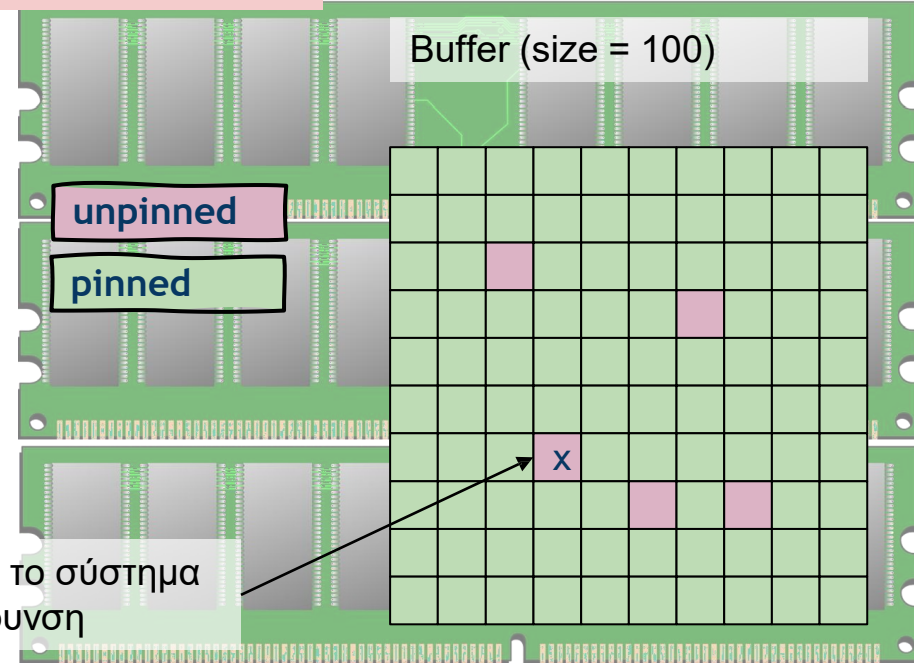
Κάποια στιγμή ο buffer θα γεμίσει.

- Τότε για να αποθηκεύσει ένα καινούργιο block (το block a) θα πρέπει να απομακρύνει κάποιο άλλο
- Αυτό που θα απομακρυνθεί να επιλεγεί μεταξύ των **unpinned**



Δίσκος

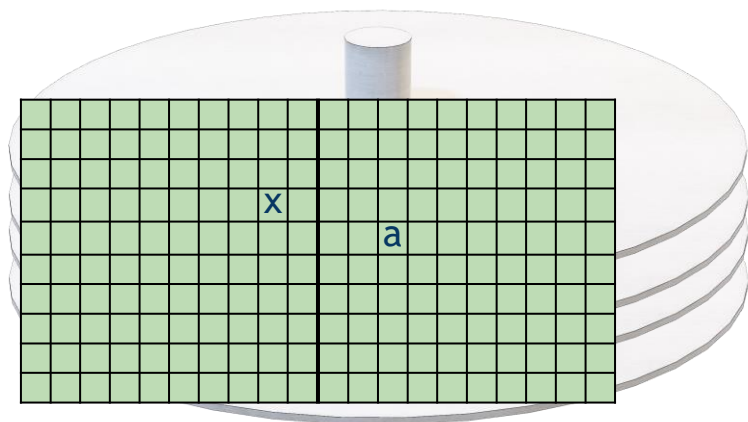
επιλογή από το σύστημα
για απομάκρυνση



Ram

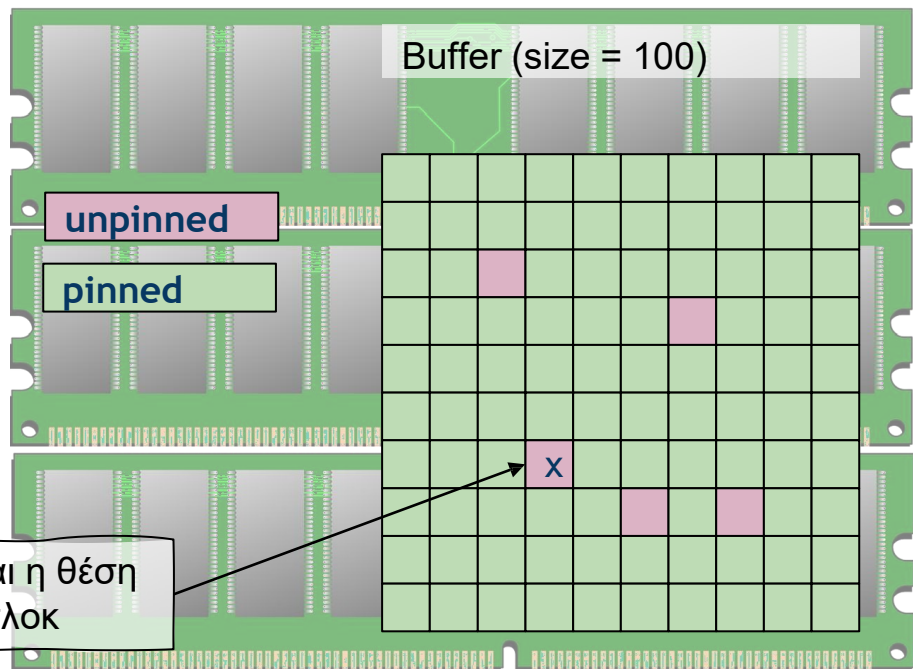
Κάποια στιγμή ο buffer θα γεμίσει.

- Τότε για να αποθηκεύσει ένα καινούργιο block (το block a) θα πρέπει να απομακρύνει κάποιο άλλο
- Αυτό που θα απομακρυνθεί να επιλεγεί μεταξύ των **unpinned**



Δίσκος

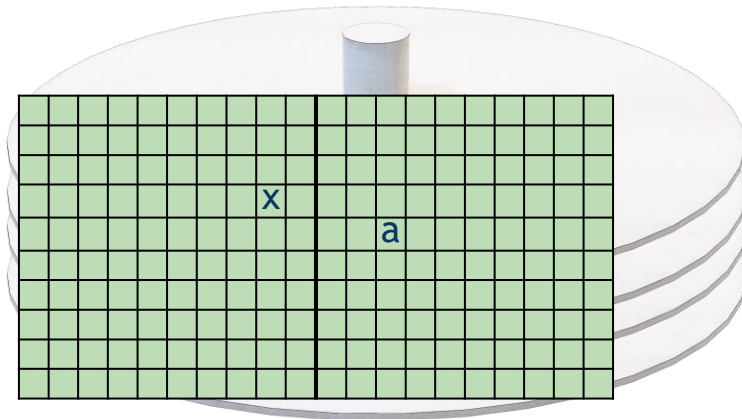
Απελευθερώνεται η θέση
από το παλιό μπλοκ



Ram

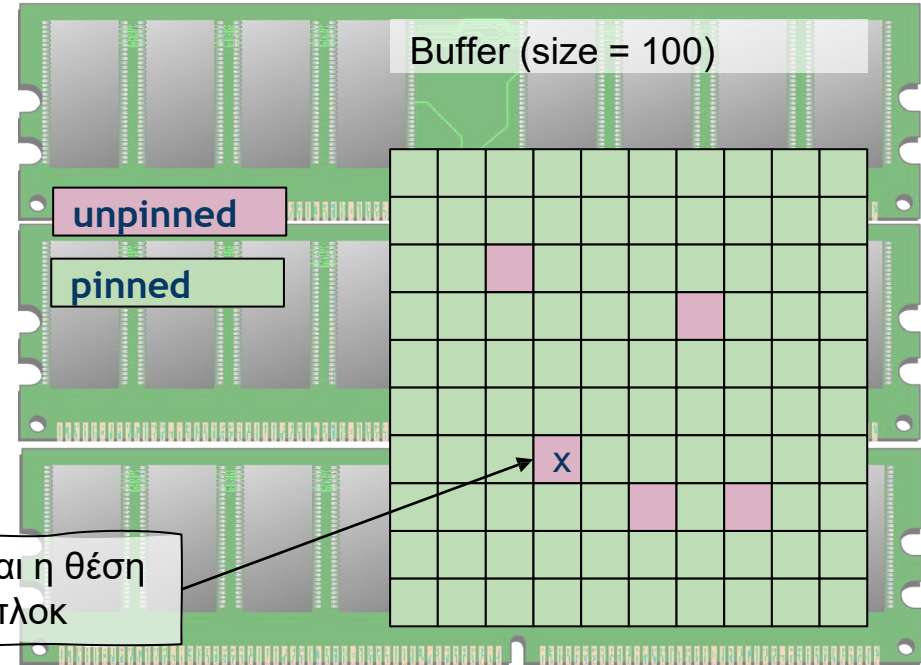
Κάποια στιγμή ο buffer θα γεμίσει.

- Τότε για να αποθηκεύσει ένα καινούργιο block (το block a) θα πρέπει να απομακρύνει κάποιο άλλο
- Αυτό που θα απομακρυνθεί να επιλεγεί μεταξύ των **unpinned**



Δίσκος

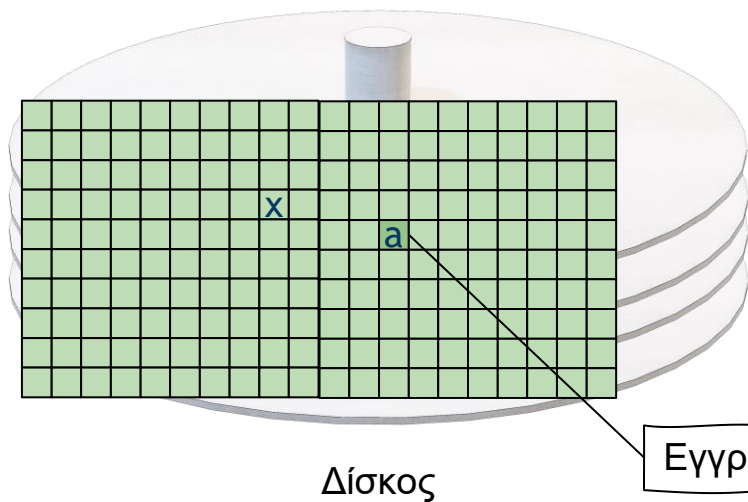
Απελευθερώνεται η θέση
από το παλιό μπλοκ



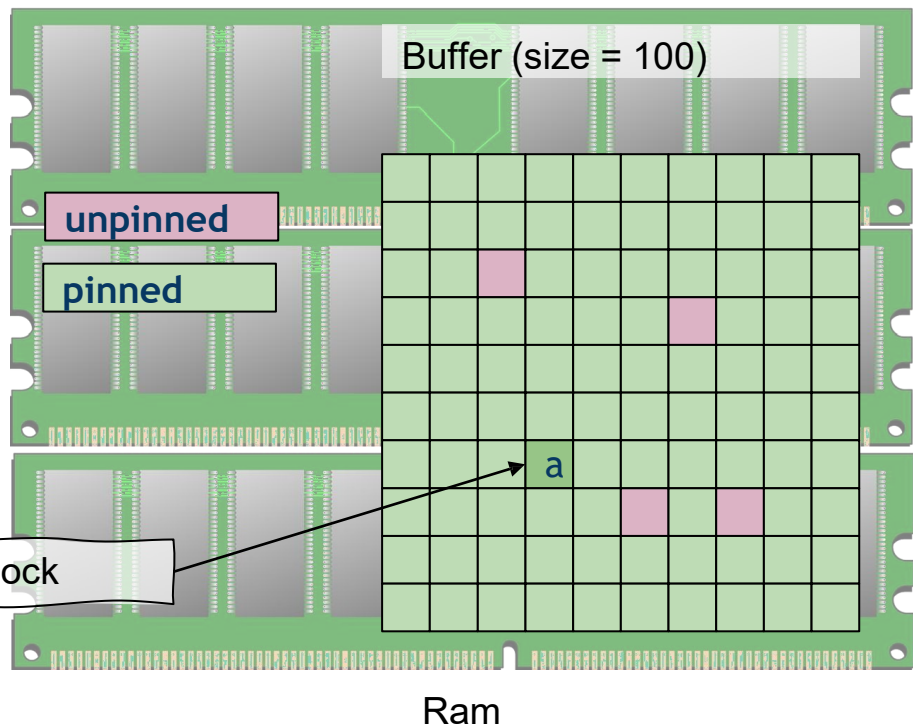
Ram

Κάποια στιγμή ο buffer θα γεμίσει.

- Τότε για να αποθηκεύσει ένα **καινούργιο** block (το block a) θα πρέπει να απομακρύνει κάποιο άλλο
- Αυτό που θα απομακρυνθεί να επιλεγθεί μεταξύ των **unpinned**

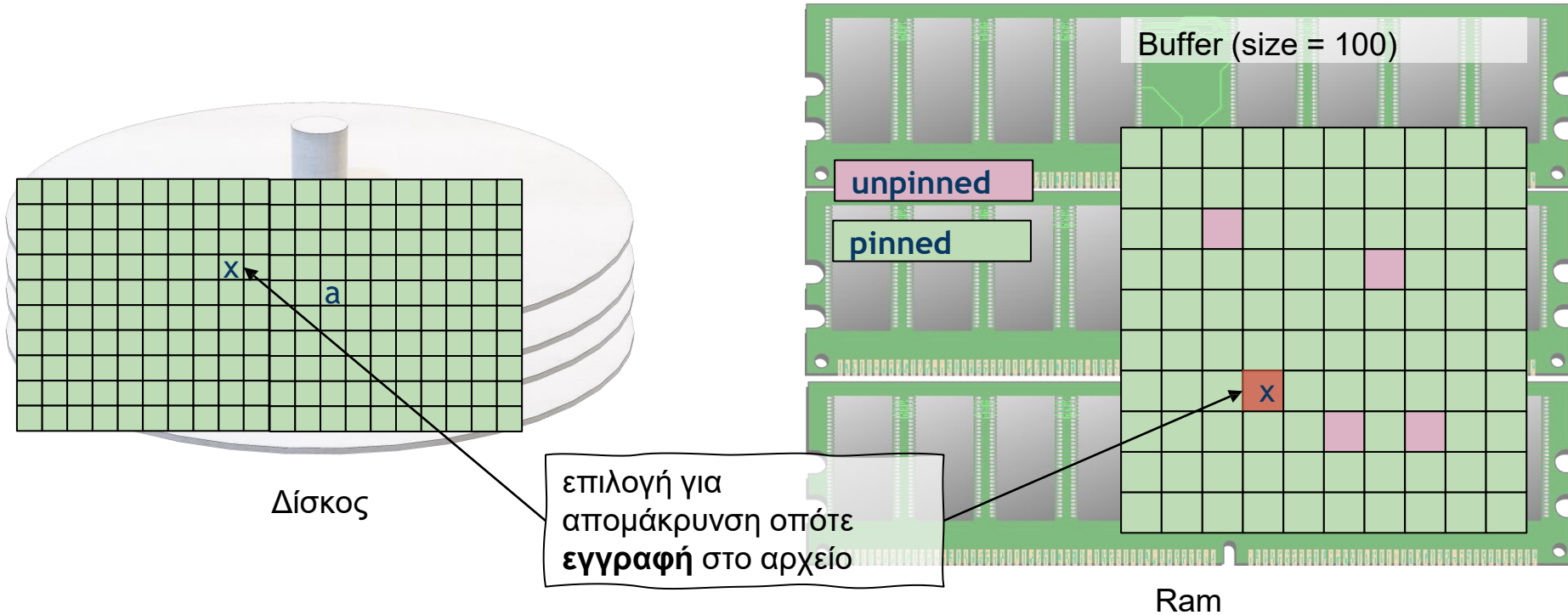


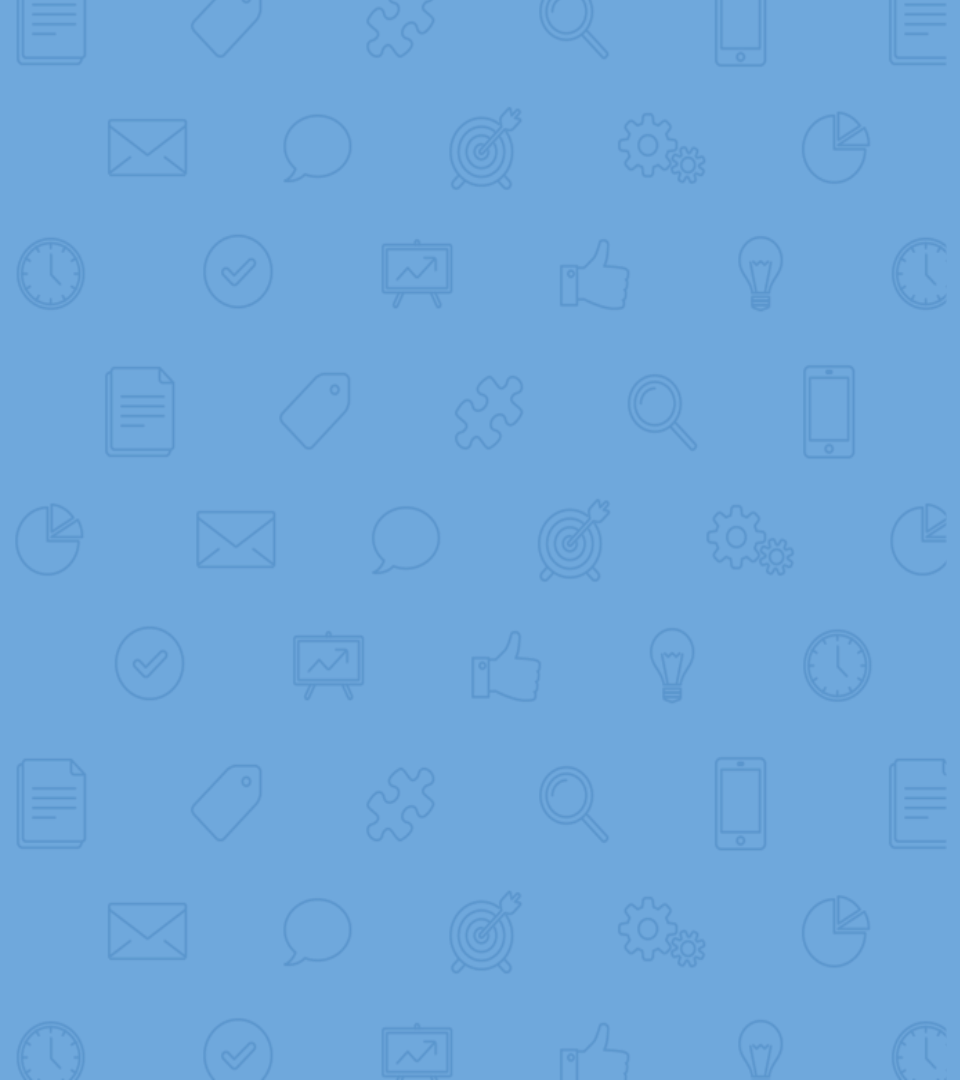
Εγγραφή νέου block



Αν το unpinned block που επιλέγονται για απομάκρυνση από τον buffer manager είναι και **βρώμικα (dirty)**

- τότε θα **επανεγγραφούν στον δίσκο** από τον buffer manager.





Συναρτήσεις BF block (bf.h)

Συναρτήσεις Διαχειριστή Ενδιάμεσης Μνήμης

- ▶ Οι προηγούμενες λειτουργίες του **διαχειριστή ενδιάμεσης μνήμης** (buffer manager) υλοποιούνται από συναρτήσεις οι οποίες σας δίνονται σε μορφή βιβλιοθήκης:
 - ▶ BF block (bf.h)

Μεταβλητές BF-Block

- ▶ Ο προγραμματιστής αλληλεπιδρά με το επίπεδο μπλοκ και όχι απευθείας με τον δίσκο.
- ▶ Η βασική δομή που χρησιμοποιείται είναι η BF-Block
 - ▶ Οι βασικές μεταβλητές είναι:

BF_BLOCK_SIZE 512 /* Το μέγεθος ενός block σε bytes */

BF_BUFFER_SIZE 100 /* Ο μέγιστος αριθμός block που κρατάμε στην μνήμη */

BF_MAX_OPEN_FILES 100 /* Ο μέγιστος αριθμός ανοικτών αρχείων */

Συναρτήσεις BF-Block

- ▶ **void BF_Block_Init(BF_Block **block)**
αρχικοποιεί και δεσμεύει την κατάλληλη μνήμη για την δομή BF_BLOCK.
- ▶ **void BF_Block_Destroy(BF_Block **block)**
αποδεσμεύει την μνήμη που καταλαμβάνει η δομή BF_BLOCK.
- ▶ **void BF_Block_SetDirty(BF_Block *block)**
η συνάρτηση BF_Block_SetDirty αλλάζει την κατάσταση του block σε dirty. Αυτό πρακτικά σημαίνει ότι τα δεδομένα του block έχουν αλλαχτεί και το επίπεδο BF όταν χρειαστεί θα γράψει το block ξανά στον δίσκο. Σε περίπτωση που απλώς διαβάζουμε τα δεδομένα χωρίς να τα αλλάζουμε τότε δεν χρειάζεται να καλέσουμε την συνάρτηση.
- ▶ **char* BF_Block_GetData(const BF_Block *block)**
επιστρέφει ένα δείκτη στα δεδομένα του Block. Άμα αλλάξουμε τα δεδομένα θα πρέπει να κάνουμε το block dirty με την κλήση της συνάρτησης BF_Block_GetData.

Συναρτήσεις BF-Block

- ▶ **BF_ErrorCode BF_Init (const ReplacementAlgorithm repl_alg)**
ορίζει τον αλγόριθμο αντικατάστασης.
- ▶ **BF_ErrorCode BF_CreateFile(const char* filename)**
δημιουργεί ένα αρχείο με όνομα filename το οποίο αποτελείται από blocks. Αν το αρχείο υπάρχει ήδη τότε επιστρέφεται κωδικός λάθους.
- ▶ **BF_ErrorCode BF_OpenFile(const char* filename, int *file_desc)**
ανοίγει ένα υπάρχον αρχείο από blocks με όνομα filename και επιστρέφει το αναγνωριστικό του αρχείου στην μεταβλητή file_desc.
- ▶ **BF_ErrorCode BF_CloseFile(int file_desc)**
κλείνει το ανοιχτό αρχείο με αναγνωριστικό αριθμό file_desc

Συναρτήσεις BF-Block

- ▶ **BF_ErrorCode BF_GetBlockCounter(const int file_desc, int *blocks_num)**

δέχεται ως όρισμα τον αναγνωριστικό αριθμό file_desc ενός ανοιχτού αρχείου από block και βρίσκει τον αριθμό των διαθέσιμων blocks του, τον οποίο και επιστρέφει στην μεταβλητή blocks_num.

- ▶ **BF_ErrorCode BF_AllocateBlock(const int file_desc, BF_Block *block)**

Με τη συνάρτηση BF_AllocateBlock δεσμεύεται ένα καινούριο block για το αρχείο με αναγνωριστικό αριθμό blockFile. Το νέο block δεσμεύεται πάντα στο τέλος του αρχείου, οπότε ο αριθμός του block είναι $\text{BF_getBlockCounter}(\text{file_desc}) - 1$. Το block που δεσμεύεται καρφιτσώνεται στην μνήμη (pin) και επιστρέφεται στην μεταβλητή block. Όταν δεν το χρειαζόμαστε άλλο αυτό το block τότε πρέπει να ενημερώσουμε τον επίπεδο block καλώντας την συνάρτηση BF_UnpinBlock.

Δημιουργία Αρχείου

```
int fd1;
BF_Block* block;
void* data;

CALL_OR_DIE(BF_Init(LRU));
CALL_OR_DIE(BF_CreateFile("ex.db"));
CALL_OR_DIE(BF_OpenFile("ex.db", &fd1));
BF_Block_Init(&block);

for (int i = 0; i < 10; ++i) {
    CALL_OR_DIE(BF_AllocateBlock(fd1, block));
    data = BF_Block_GetData(block);
    Record* rec = data;
    rec[0] = randomRecord();
    rec[1] = randomRecord();
    BF_Block_SetDirty(block);
    CALL_OR_DIE(BF_UnpinBlock(block));
}
CALL_OR_DIE(BF_CloseFile(fd1));
CALL_OR_DIE(BF_Close());
```

Ram

Σκληρός Δίσκος

Δημιουργία Αρχείου

- Η μεταβλητή **fd1** θα χρησιμοποιηθεί για την αναγνώριση ενός αρχείου το οποίο δημιουργείται από την βιβλιοθήκη BF
- Η μεταβλητή **block** είναι δείκτης σε ένα block το οποίο έχει μεταφερθεί στην ενδιάμεση μνήμη
- Η μεταβλητή **data** είναι δείκτης σε μία περιοχή της μνήμης RAM που περιέχει δεδομένα

```
int fd1;
BF_Block* block;
void* data;

CALL_OR_DIE(BF_Init(LRU));
CALL_OR_DIE(BF_CreateFile("ex.db"));
CALL_OR_DIE(BF_OpenFile("ex.db", &fd1));
BF_Block_Init(&block);

for (int i = 0; i < 10; ++i) {
    CALL_OR_DIE(BF_AllocateBlock(fd1, block));
    data = BF_Block_GetData(block);
    Record* rec = data;
    rec[0] = randomRecord();
    rec[1] = randomRecord();
    BF_Block_SetDirty(block);
    CALL_OR_DIE(BF_UnpinBlock(block));
}

CALL_OR_DIE(BF_CloseFile(fd1));
CALL_OR_DIE(BF_Close());
```

Ram

Σκληρός Δίσκος

Δημιουργία Αρχείου

```
int fd1;  
BF_Block* block;  
void* data;
```

```
CALL_OR_DIE(BF_Init(LRU));  
CALL_OR_DIE(BF_CreateFile("ex.db"));  
CALL_OR_DIE(BF_OpenFile("ex.db", &fd1));  
BF_Block_Init(&block);
```

```
for (int i = 0; i < 10; ++i) {  
    CALL_OR_DIE(BF_AllocateBlock(fd1, block));  
    data = BF_Block_GetData(block);  
    Record* rec = data;  
    rec[0] = randomRecord();  
    rec[1] = randomRecord();  
    BF_Block_SetDirty(block);  
    CALL_OR_DIE(BF_UnpinBlock(block));  
}  
CALL_OR_DIE(BF_CloseFile(fd1));  
CALL_OR_DIE(BF_Close());
```

Αρχικοποιώ τον διαχειριστή
ενδιάμεση μνήμης με πολιτική
αντικατάστασης **Least Recently
Used**

Ενδιάμεση Μνήμη

Ram

Σκληρός Δίσκος

Δημιουργία Αρχείου

```
int fd1;  
BF_Block* block;  
void* data;
```

```
CALL_OR_DIE(BF_Init(LRU));  
CALL_OR_DIE(BF_CreateFile("e  
CALL_OR_DIE(BF_OpenFile("ex.db", &fd1));  
BF_Block_Init(&block);
```

```
for (int i = 0; i < 10; ++i) {  
    CALL_OR_DIE(BF_AllocateBlock(fd1, block));  
    data = BF_Block_GetData(block);  
    Record* rec = data;  
    rec[0] = randomRecord();  
    rec[1] = randomRecord();  
    BF_Block_SetDirty(block);  
    CALL_OR_DIE(BF_UnpinBlock(block));  
}  
CALL_OR_DIE(BF_CloseFile(fd1));  
CALL_OR_DIE(BF_Close());
```

- Δημιουργώ το αρχείο ex.db στον σκληρό μου
- Ανοίγω το συγκεκριμένο αρχείο και δίνω τιμή στον αντίστοιχο file descriptor

Ενδιάμεση Μνήμη

Ram

ex.db

Σκληρός Δίσκος

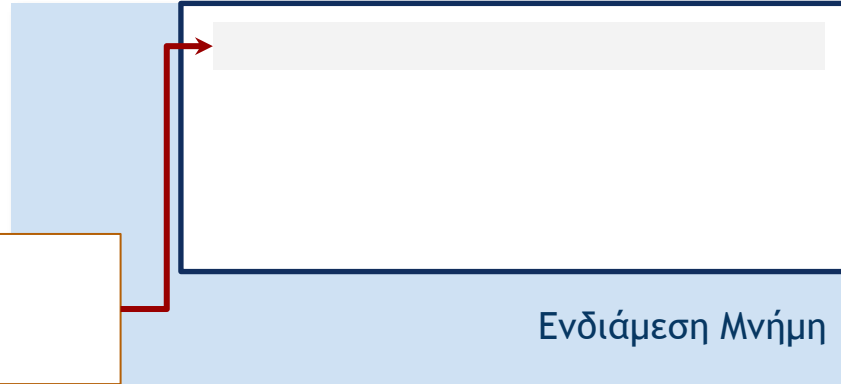
Δημιουργία Αρχείου

```
int fd1;  
BF_Block* block;  
void* data;
```

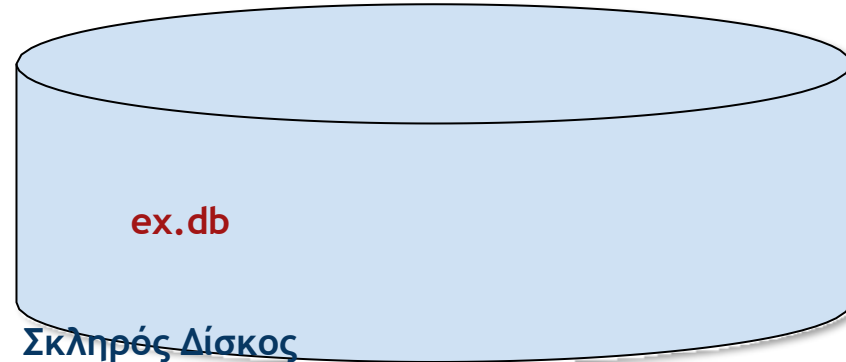
```
CALL_OR_DIE(BF_Init(LRU));  
CALL_OR_DIE(BF_CreateFile("ex.db",  
CALL_OR_DIE(BF_OpenFile("ex.db",  
BF_Block_Init(&block));
```

```
for (int i = 0; i < 10; ++i) {  
    CALL_OR_DIE(BF_AllocateBlock(fd1, block));  
    data = BF_Block_GetData(block);  
    Record* rec = data;  
    rec[0] = randomRecord();  
    rec[1] = randomRecord();  
    BF_Block_SetDirty(block);  
    CALL_OR_DIE(BF_UnpinBlock(block));  
}  
CALL_OR_DIE(BF_CloseFile(fd1));  
CALL_OR_DIE(BF_Close());
```

Αρχικοποιώ και δεσμεύω την κατάλληλη μνήμη για την δομή BF_Block



Ram



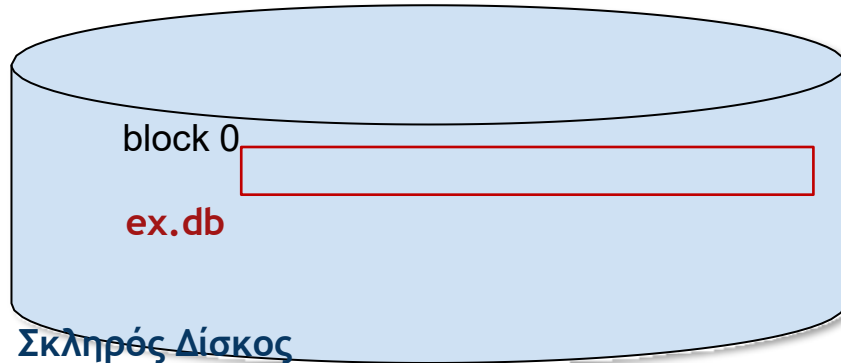
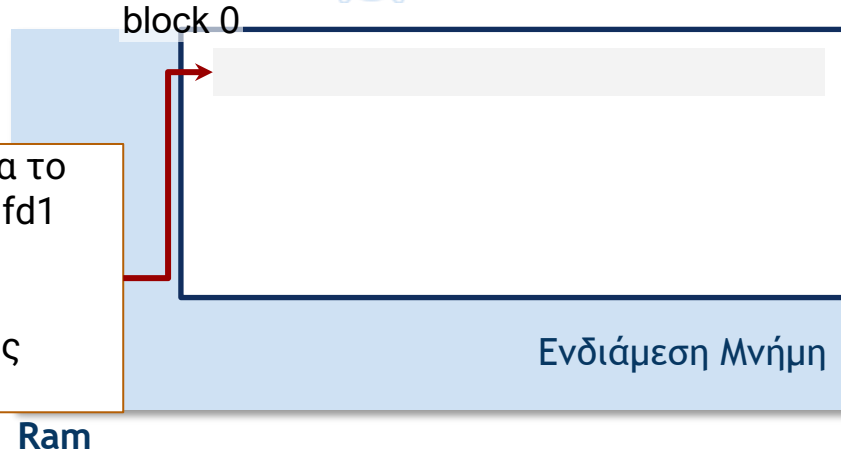
Δημιουργία Αρχείου

```
int fd1;  
BF_Block* block;  
void* data;
```

```
CALL_OR_DIE(BF_Init(  
CALL_OR_DIE(BF_Create  
CALL_OR_DIE(BF_OpenF  
BF_Block_Init(&block
```

```
for (int i = 0; i < 10; ++i) {  
    CALL_OR_DIE(BF_AllocateBlock(fd1, block));  
    data = BF_Block_GetData(block);  
    Record* rec = data;  
    rec[0] = randomRecord();  
    rec[1] = randomRecord();  
    BF_Block_SetDirty(block);  
    CALL_OR_DIE(BF_UnpinBlock(block));  
}  
CALL_OR_DIE(BF_CloseFile(fd1));  
CALL_OR_DIE(BF_Close());
```

- Δεσμεύω ένα καινούριο μπλοκ για το αρχείο με αναγνωριστικό αριθμό fd1 (ex.db)
- Το νέο μπλοκ φορτώνεται στην αντίστοιχη σελίδα της ενδιάμεσης μνήμης



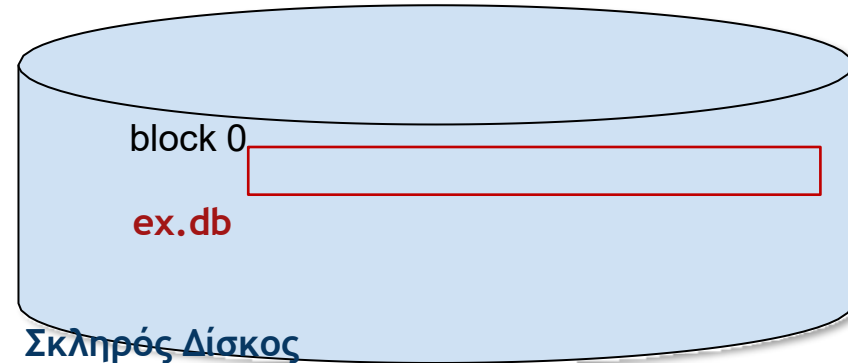
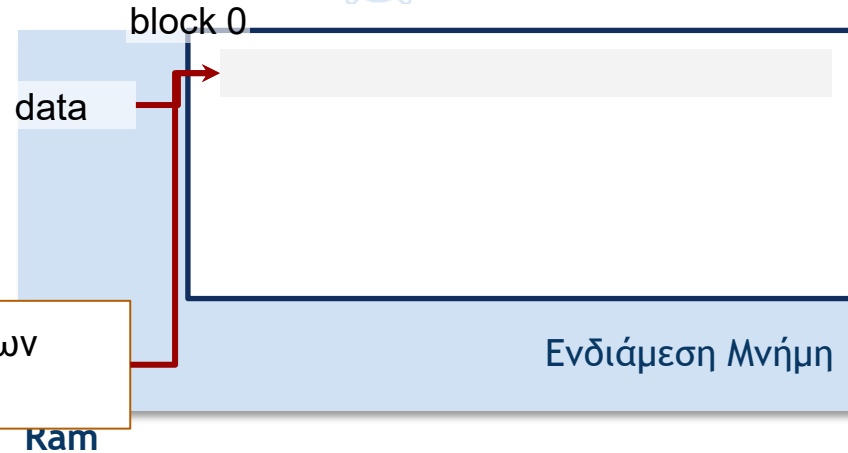
Δημιουργία Αρχείου

```
int fd1;  
BF_Block* block;  
void* data;
```

```
CALL_OR_DIE(BF_Init(LRU));  
CALL_OR_DIE(BF_CreateFile("ex.db"));  
CALL_OR_DIE(BF_OpenFile(&block));  
BF_Block_Init(&block);
```

```
for (int i = 0; i < 10; i++)  
{  
    CALL_OR_DIE(BF_AllocateBlock(&block));  
    data = BF_Block_GetData(block);  
    Record* rec = data;  
    rec[0] = randomRecord();  
    rec[1] = randomRecord();  
    BF_Block_SetDirty(block);  
    CALL_OR_DIE(BF_UnpinBlock(block));  
}  
CALL_OR_DIE(BF_CloseFile(fd1));  
CALL_OR_DIE(BF_Close());
```

Ο δείκτης data να δείχνει στην αρχή των δεδομένων της σελίδας/μπλοκ.



Δημιουργία Αρχείου

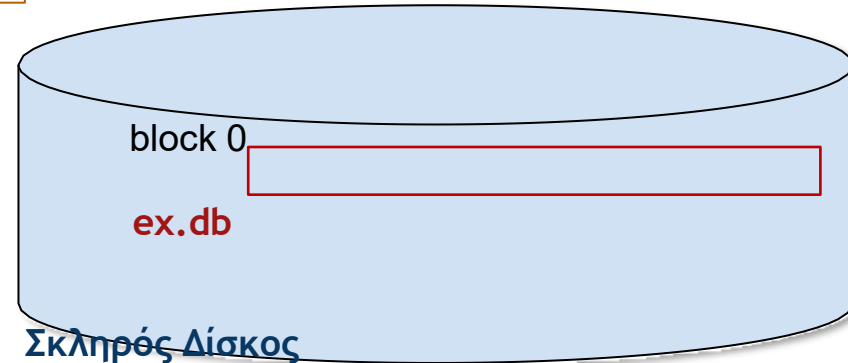
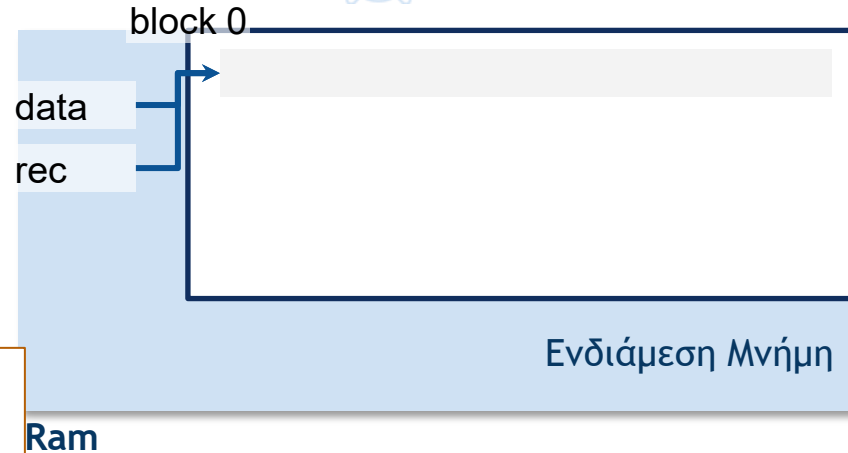
```
int fd1;
BF_Block* block;
void* data;

CALL_OR_DIE(BF_Init(LRU));
CALL_OR_DIE(BF_CreateFile("ex.db"));
CALL_OR_DIE(BF_OpenFile("ex.db", &fd1));
BF_Block_Init(block);

for (int i = 0; i < 10; i++) {
    CALL_OR_DIE(BF_GetBlock(block));
    data = BF_Block_GetData(block);
    Record* rec = data;
    rec[0] = randomRecord();
    rec[1] = randomRecord();
    BF_Block_SetDirty(block);
    CALL_OR_DIE(BF_UnpinBlock(block));
}

CALL_OR_DIE(BF_CloseFile(fd1));
CALL_OR_DIE(BF_Close());
```

Δημιουργώ έναν δείκτη σε Record ο οποίος δείχνει στην ίδια περιοχή της ενδιάμεσης με τον δείκτη data



Δημιουργία Αρχείου

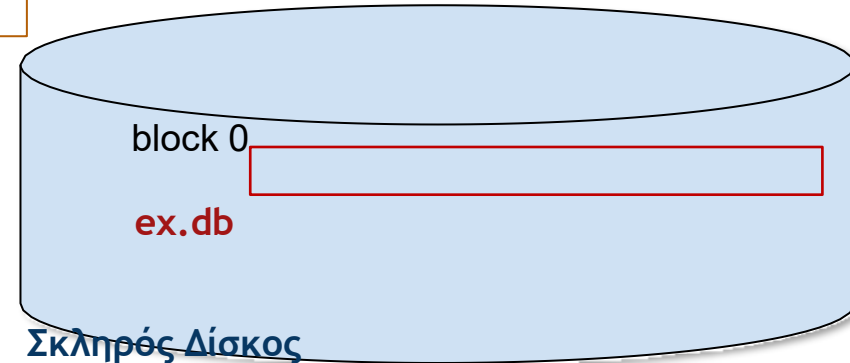
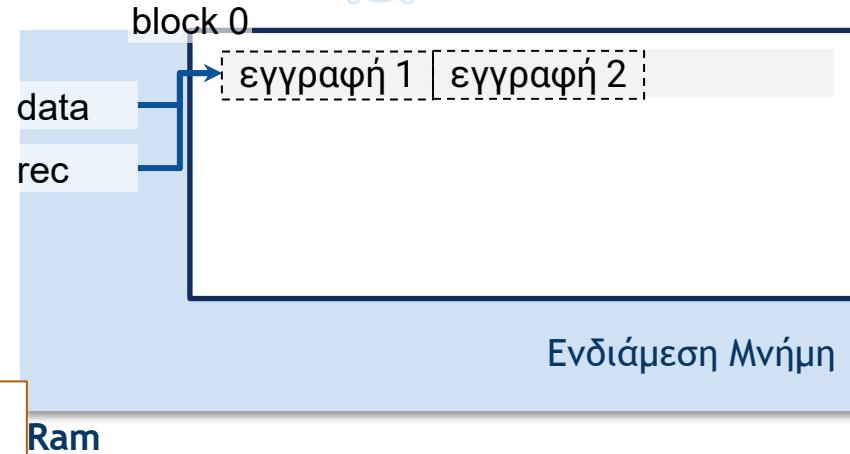
```
int fd1;
BF_Block* block;
void* data;

CALL_OR_DIE(BF_Init(LRU));
CALL_OR_DIE(BF_CreateFile("ex.db"));
CALL_OR_DIE(BF_OpenFile("ex.db", &fd1));
BF_Block_Init(&block);
```

```
for (int i = 0; i < 2; i++) {
    CALL_OR_DIE(BF_ReadRec(block, rec, i));
    Record* rec = data;
    rec[0] = randomRecord();
    rec[1] = randomRecord();
    BF_Block_SetDirty(block);
    CALL_OR_DIE(BF_UnpinBlock(block));
}
```

Κάνω εισαγωγή δύο τυχαίων εγγραφών στις
δύο πρώτες θέσεις του δείκτη rec

```
CALL_OR_DIE(BF_CloseFile(fd1));
CALL_OR_DIE(BF_Close());
```



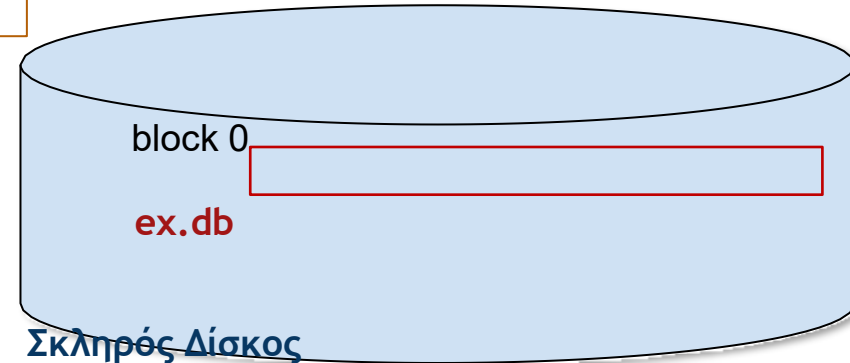
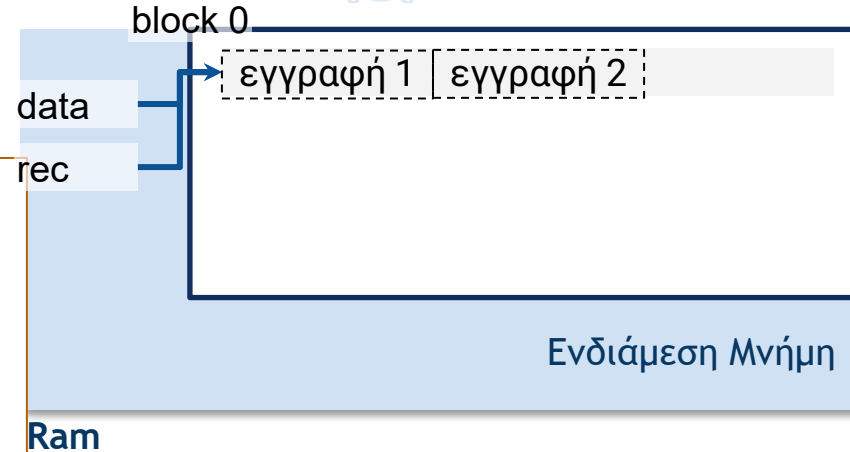
Δημιουργία Αρχείου

```
int fd1;  
BF_Block* block;  
void* data;
```

```
CALL_OR_DIE(BF  
CALL_OR_DIE(BF  
CALL_OR_DIE(BF  
BF_Block_Init(
```

```
for (int i = 0  
CALL_OR_DI  
data = BF_  
Record* rec = data;  
rec[0] = randomReco  
rec[1] = randomReco  
BF_Block_SetDirty(block);  
CALL_OR_DIE(BF_UnpinBlock(block));  
}  
CALL_OR_DIE(BF_CloseFile(fd1));  
CALL_OR_DIE(BF_Close());
```

- Λέω στον διαχειριστή ενδιάμεσης μνήμης ότι η σελίδα **block 0** που βρίσκεται στην ενδιάμεση μνήμη έχει αλλαχθεί
- Αυτό μου διασφαλίζει ότι οι αλλαγές θα μεταφερθούν στο **block 0** του σκληρού δίσκου



Δημιουργία Αρχείου

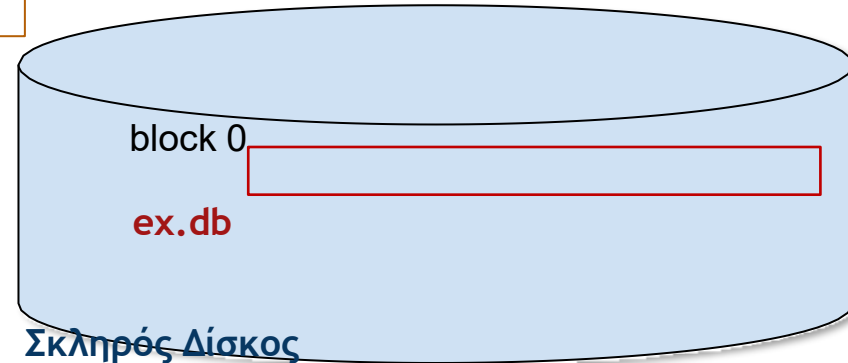
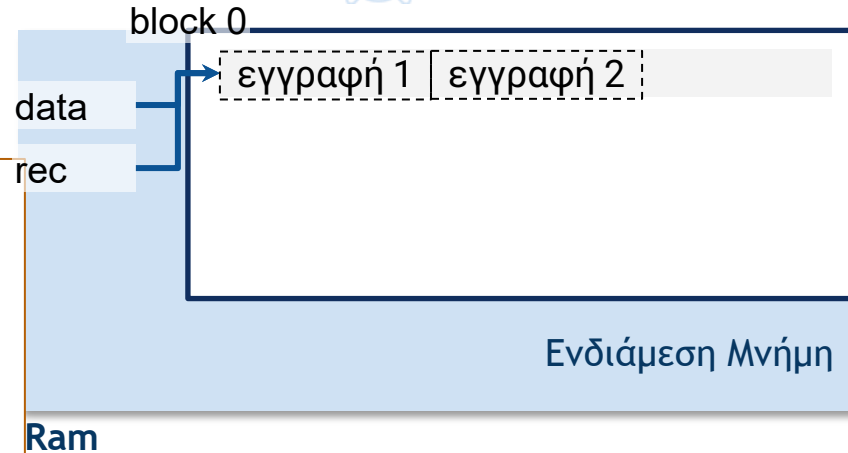
```
int fd1;  
BF_Block* block;  
void* data;
```

```
CALL_OR_DIE(BF  
CALL_OR_DIE(BF  
CALL_OR_DIE(BF  
BF_Block_Init(
```

```
for (int i = 0  
CALL_OR_DI  
data = BF_  
Record* rec = data;  
rec[0] = randomRecord  
rec[1] = randomRecord(  
BF_Block_SetDirty(block);  
CALL_OR_DIE(BF_UnpinBlock(block));
```

```
}  
CALL_OR_DIE(BF_CloseFile(fd1));  
CALL_OR_DIE(BF_Close());
```

- Λέω στο σύστημά μου, ότι δεν χρειάζεται πλέον την συγκεκριμένη σελίδα ενδιαμέσης μνήμης οπότε να γίνει **unpin**
- Αυτό σημαίνει ότι βάσει της πολιτικής αντικατάστασης κάποια άλλη σελίδα μπορεί να δεσμεύσει την συγκεκριμένη περιοχή μνήμης



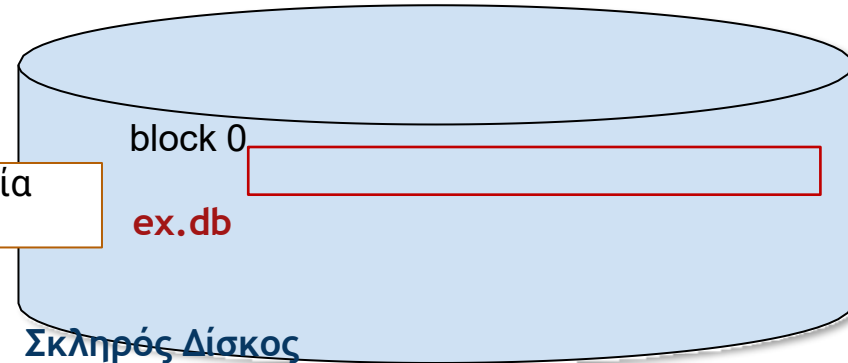
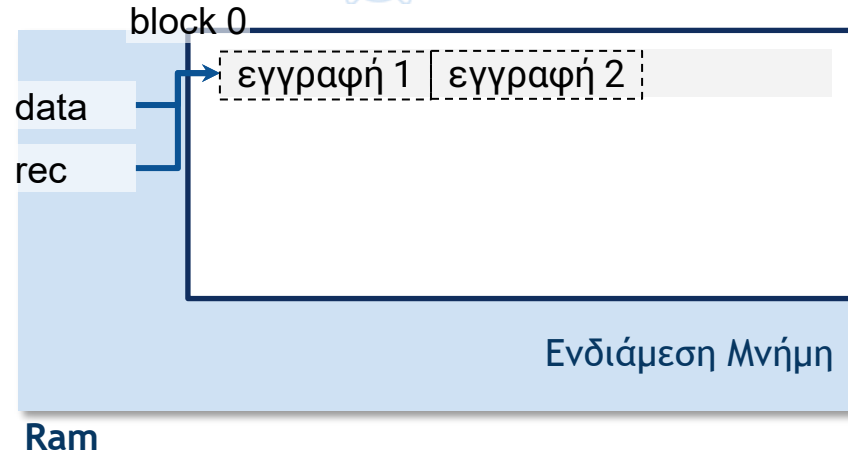
Δημιουργία Αρχείου

```
int fd1;
BF_Block* block;
void* data;

CALL_OR_DIE(BF_Init(LRU));
CALL_OR_DIE(BF_CreateFile("ex.db"));
CALL_OR_DIE(BF_OpenFile("ex.db", &fd1));
BF_Block_Init(&block);

for (int i = 0; i < 10; ++i) {
    CALL_OR_DIE(BF_AllocateBlock(fd1, block));
    data = BF_Block_GetData(block);
    Record* rec = data;
    rec[0] = randomRecord();
    rec[1] = randomRecord();
    BF_Block_SetDirty(block);
    CALL_OR_DIE(BF_UnpinBlock(blo
}
CALL_OR_DIE(BF_CloseFile(fd1));
CALL_OR_DIE(BF_Close());
```

Τερματίζω την επεξεργασία
του αρχείου



Αναφορές

- ▶ <https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html>
- ▶ https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_Recently_Used_.28LRU.29
- ▶ https://en.wikipedia.org/wiki/Cache_replacement_policies#Most_Recently_Used_.28MRU.29
- ▶ <https://linux.die.net/man/1/strace>

The background of the slide features a repeating pattern of various business and technology icons in a light blue color. These icons include a document, a tag, a puzzle piece, a magnifying glass, a smartphone, a pie chart, an envelope, a speech bubble, a target with an arrow, interlocking gears, a clock, a checkmark, a presentation board with a line graph, a thumbs up, and a lightbulb. The icons are arranged in a grid-like fashion across the entire left half of the slide.

1η Εργασία Αρχείο Σωρού



Στόχος

- ▶ Καλείστε να υλοποιήσετε μια σειρά συναρτήσεων:
 - ▶ που διαχειρίζονται **αρχεία Σωρού**
 - ▶ χρησιμοποιώντας την βιβλιοθήκη **BF.h**

Εγγραφές

```
typedef struct{  
    int id,  
    char name[15],  
    char surname[20],  
    char city[10];  
} Record;
```

Θεωρείστε οι εγγραφές σας περιέχουν τα γνωρίσματα : id, name, surname, city.

(15, "Marianna", "Rezkalla", "Hong Kong") (4, "Christoforos", "Svingos", "Athens") (300, "Sofia", "Karvounari", "San Francisco")

Π.χ.: Η "Sofia" "Karvouni" με id 300, επισκέφτηκε το "San Francisco"



Αναπαράσταση Δεδομένων

- ▶ Ερωτήματα που πρέπει να σκεφτείτε:
 - ▶ Ποιο είναι το μέγεθος του record;
 - ▶ Πόσα τέτοια records χωράνε σε ένα block;
 - ▶ Πως ξέρω/αποθηκεύω πόσες εγγραφές υπάρχουν σε ένα block;



Αρχεία Σωρού (Μη ταξινομημένα)

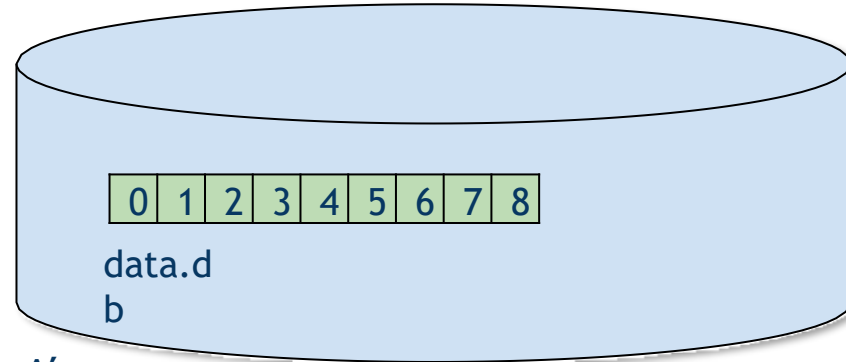
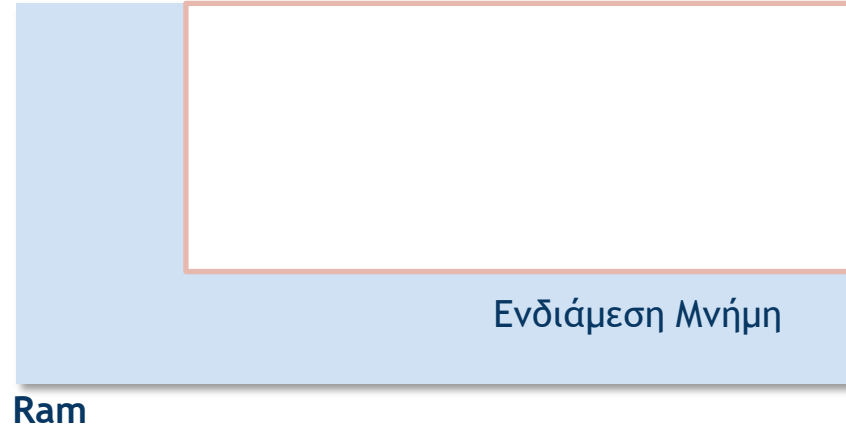
- ▶ Οι νέες εγγραφές προστίθενται στο **τέλος του αρχείου**.
- ▶ Για την αναζήτηση μιας εγγραφής είναι απαραίτητη μια **γραμμική αναζήτηση των εγγραφών του αρχείου**.
- ▶ Η **εισαγωγή** εγγραφών είναι πολύ **αποτελεσματική**.
- ▶ Η **ανάγνωση** των εγγραφών είναι μία **Ακριβή Πράξη**.

Εισαγωγή Εγγραφών

- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής

(15, "Marianna", "Rezkalla", "Hong Kong")

Μέγεθος εγγραφής: 76 bytes



Εισαγωγή Εγγραφών

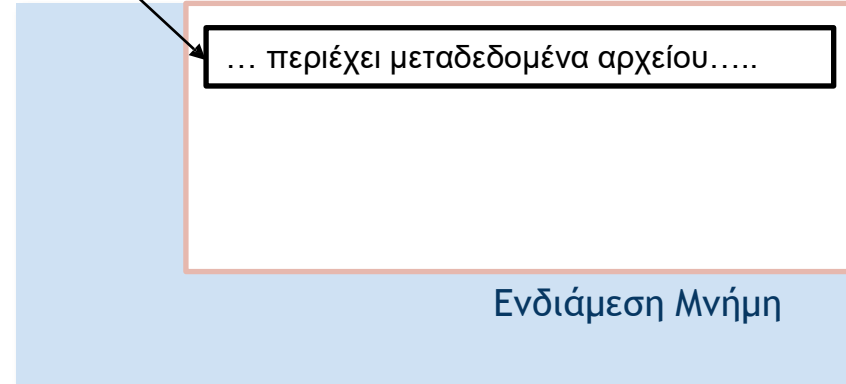
- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής

(15, "Marianna", "Rezkalla", "Hong Kong")

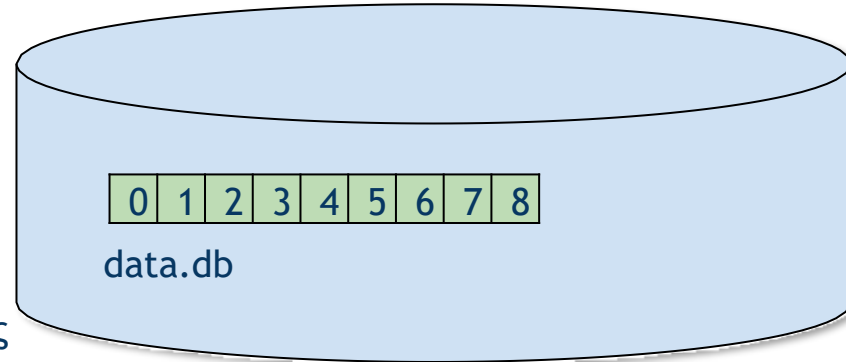
Μέγεθος εγγραφής: 76 bytes

Βήμα 1: Φορτώνω το μπλοκ 0 του αρχείου στην ενδιάμεση μνήμη

block 0



Ram



σκληρός

Εισαγωγή Εγγραφών

- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής

(15, "Marianna", "Rezkalla", "Hong Kong")

Μέγεθος εγγραφής: 76 bytes

Βήμα 1: Φορτώνω το μπλοκ 0 του αρχείου στην ενδιάμεση μνήμη

Βήμα 2: Βρίσκω ποιο είναι το τελευταίο block του αρχείου μου και το φορτώνω στην ενδιάμεση μνήμη

block 0

block 8

... περιέχει μεταδεδομένα αρχείου.....

Ενδιάμεση Μνήμη

Ram

0 1 2 3 4 5 6 7 8

data.db

σκληρός

Εισαγωγή Εγγραφών

- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.

- Θέλω να κ

(15, "Mar

Το συγκεκριμένο block χωρίζεται σε 3 περιοχές:

- μία περιοχή που περιέχει εγγραφές ,
- μία περιοχή κενή , και
- μία περιοχή που περιέχει μεταδεδομένα για το block

Βήμα 1: Φορ

ενδιάμεση μ

Βήμα 2: Βρίσκω ποιο είναι το τελευταίο block του αρχείου μου και το φορτώνω στην ενδιάμεση μνήμη

block 0

block 8

... περιέχει μεταδεδομένα αρχείου.....

εγγραφές

μεταδεδομένα

Ενδιάμεση Μνήμη

0 1 2 3 4 5 6 7 8

data.db

σκληρός

Εισαγωγή Εγγραφών

- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.

- Θέλω να κάνω την εισαγωγή της εγγραφής

(15, "Marianna", "Rez-kalla", "Hong Kong")

Μέγεθος εγγραφής: 76 bytes

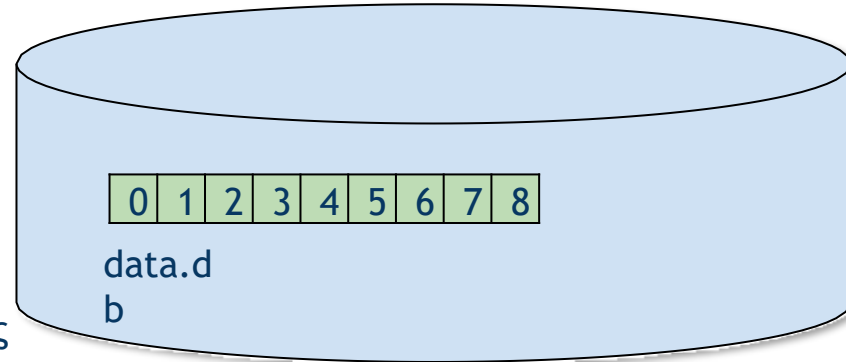
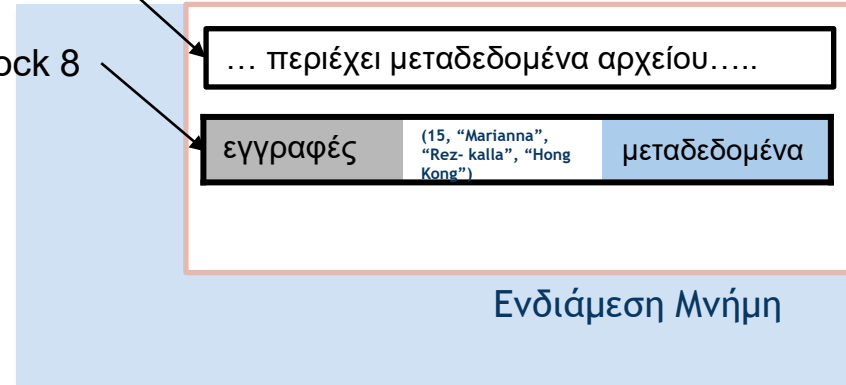
ram

Βήμα 3α: Εφόσον χωράει κάνω **εισαγωγή της εγγραφής στην σελίδα της ενδιάμεσης μνήμης** που περιέχει το block

- Επίσης κάνω το block 8 dirty και unpin εφόσον έχω ολοκληρώσει τις πράξεις μου σε αυτό

block 0

block 8



σκληρός

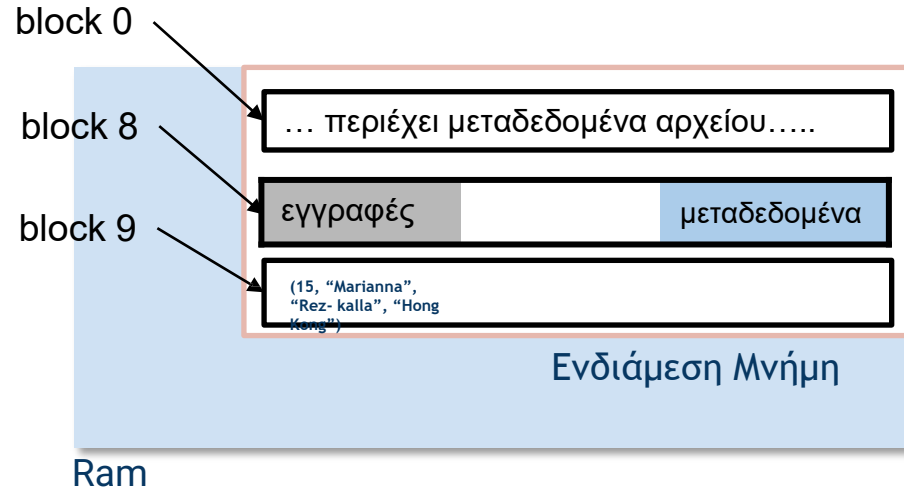
Εισαγωγή Εγγραφών

- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής

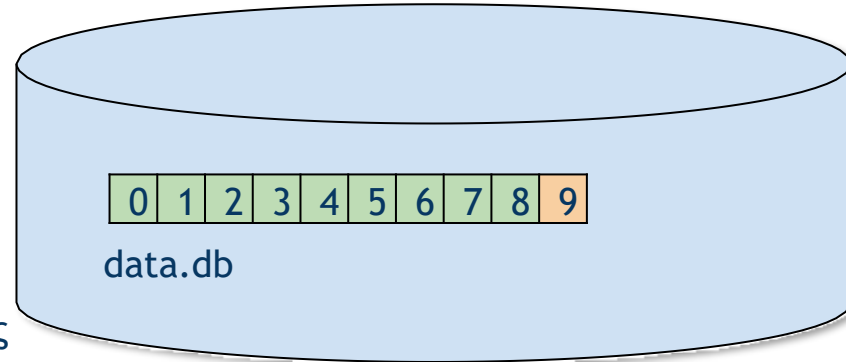
(15, "Marianna", "Rez-kalla", "Hong Kong")

Μέγεθος εγγραφής: 76 bytes

Βήμα 3β: Αν δεν χωράει κάνω allocate καινούριο block (block 9) το φορτώνω στην ενδιάμεση μνήμη και προχωρώ αναλόγως την εισαγωγή μου



σκληρός



Εισαγωγή Εγγραφών

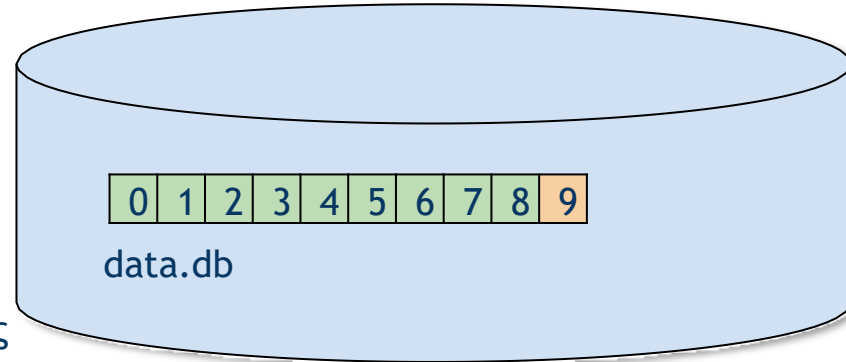
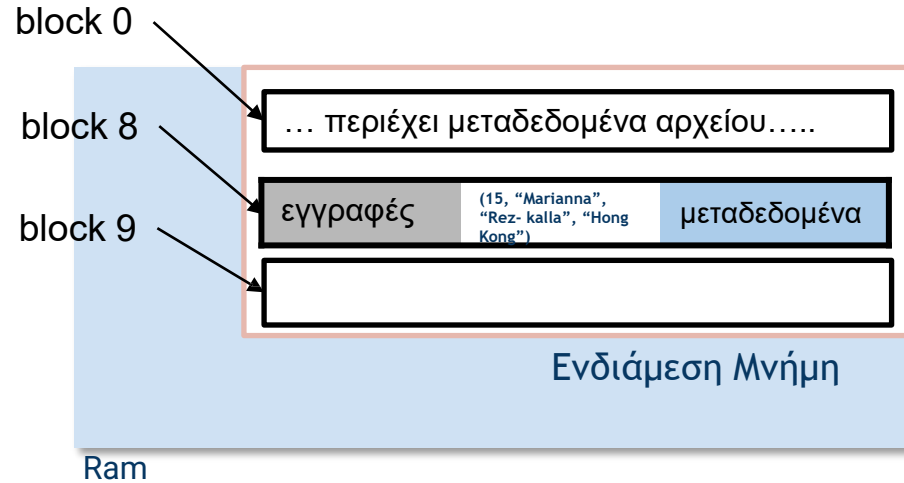
- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής

(15, "Marianna", "Rez-kalla", "Hong Kong")

Μέγεθος εγγραφής: 76 bytes

Βήμα 4: Είτε εκτελέσω το **Βήμα 3α** είτε το **Βήμα 3β**, χρειάζεται να κάνω τις αντίστοιχες αλλαγές στα μεταδεδομένα μου (είτε μεταδεδομένα του αρχείου, είτε μεταδεδομένα του block).

- Οι αλλαγές στα μεταδεδομένα πρέπει επίσης να μεταφερθούν στον δίσκο, οπότε το αντίστοιχο μπλοκ που τα περιέχει γίνεται dirty.



Τι πρέπει να
υλοποιηθεί
για τον
σωρό;

- ▶ Με χρήση των συναρτήσεων που προσφέρονται έτοιμες σε επίπεδο μπλοκ και σε επίπεδο διαχείρισης μπλοκ, καλείστε να υλοποιήσετε μια σειρά συναρτήσεων ανωτέρου επιπέδου που διαχειρίζονται αρχεία σωρού για την καταχώρηση εγγραφών:
- ▶ Χρησιμοποιώντας μία κατάλληλη δομή μεταδεδομένων