Εργασίες 1 & 2

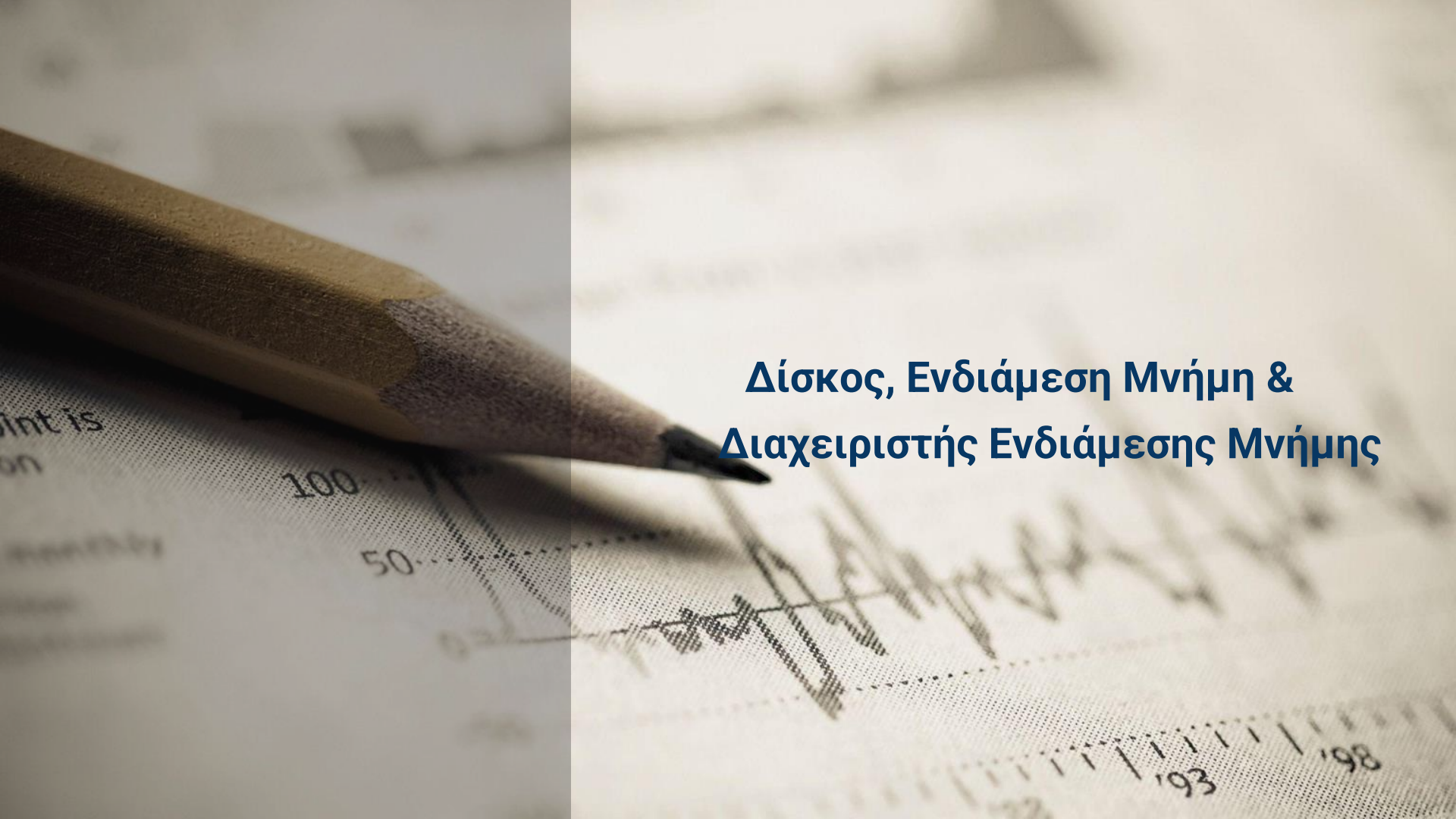
2022 - 2023

Περιγραφή εργασίας

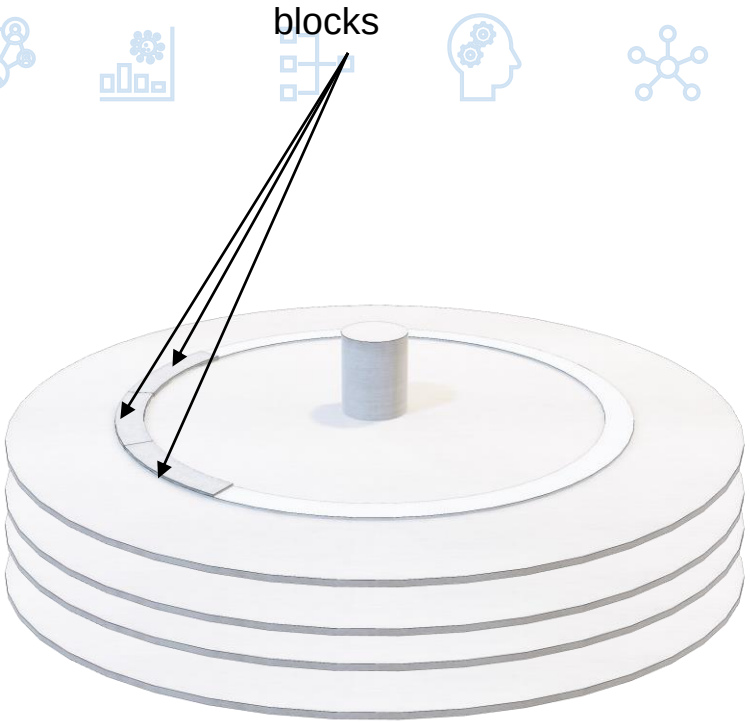
- Σκοπός της εργασίας αυτής είναι η κατανόηση της εσωτερικής λειτουργίας των Συστημάτων Βάσεων Δεδομένων όσον αφορά τη **διαχείριση σε επίπεδο μπλοκ** (block) αλλά και ως προς τη **διαχείριση σε επίπεδο εγγραφών**.
- Η υλοποίησή των συναρτήσεων σας θα γίνει **πάνω από το επίπεδο διαχείρισης μπλοκ**, 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 **στατικός κατακερματισμός.**

Επίπεδο block (BF)

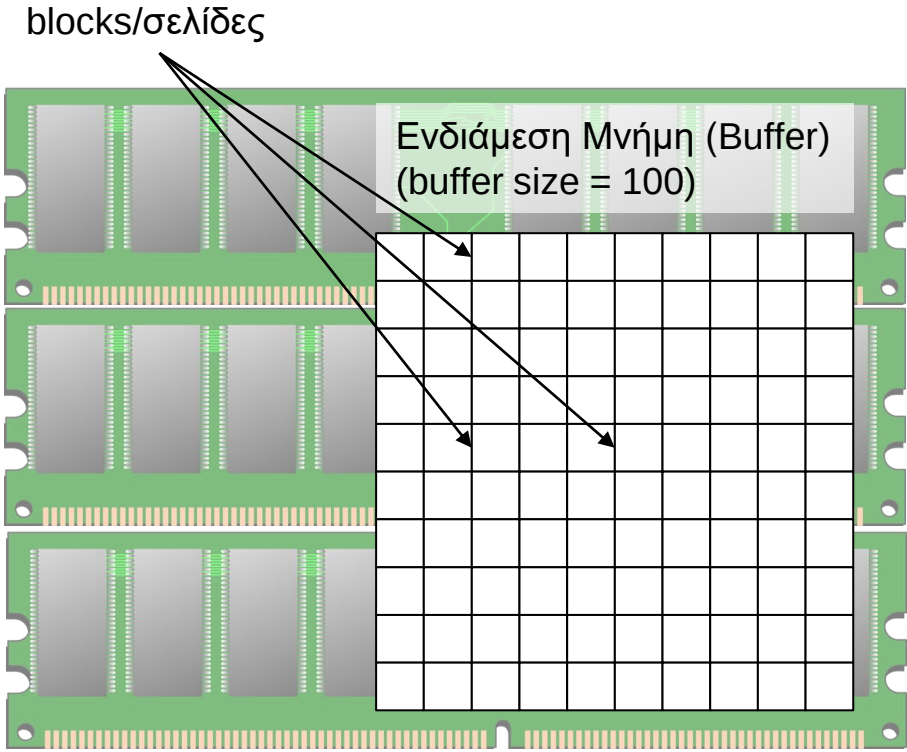
- Το **επίπεδο block** (**διαχειριστής ενδιάμεσης μνήμη** – **buffer manager**) κρατάει block δίσκου στην μνήμη. Κάθε φορά που ζητάμε ένα block δίσκου, το επίπεδο BF πρώτα εξετάζει την περίπτωση να το έχει φέρει ήδη στην μνήμη.
  - Αν το **block υπάρχει** στην μνήμη τότε δεν το διαβάζει από τον δίσκο,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



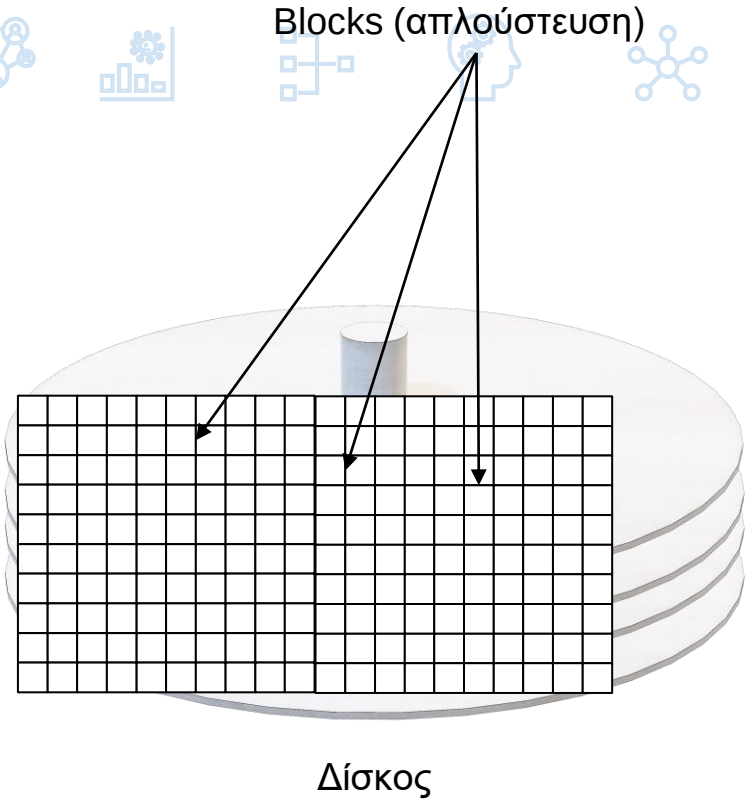
Δίσκος, Ενδιάμεση Μνήμη & Διαχειριστής Ενδιάμεσης Μνήμης

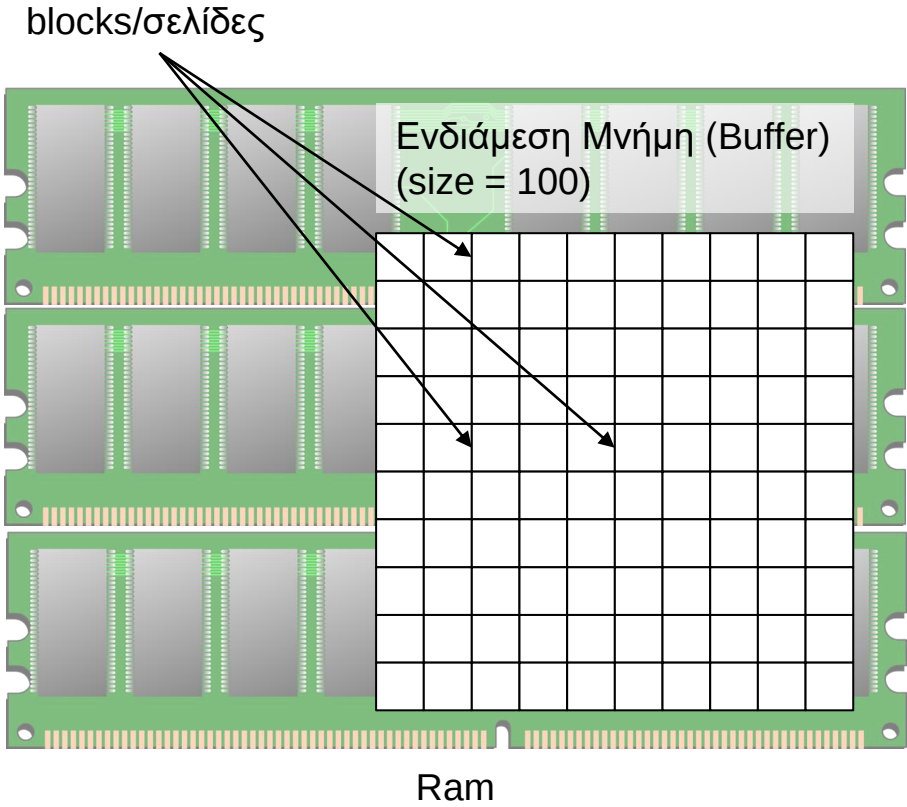

blocks

Δίσκος

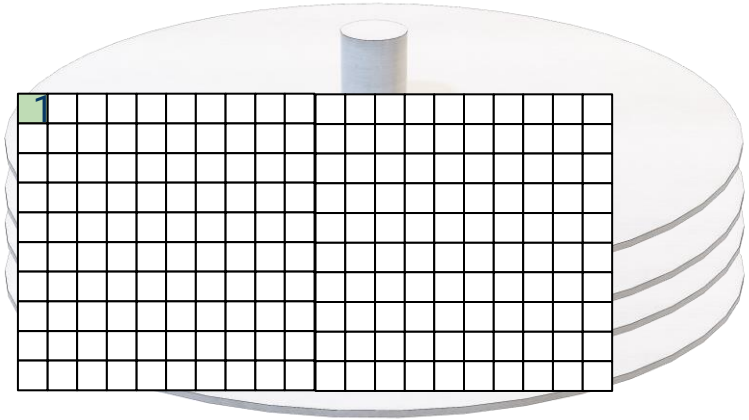

blocks/σελίδες
Ενδιάμεση Μνήμη (Buffer)
(buffer size = 100)

Ram

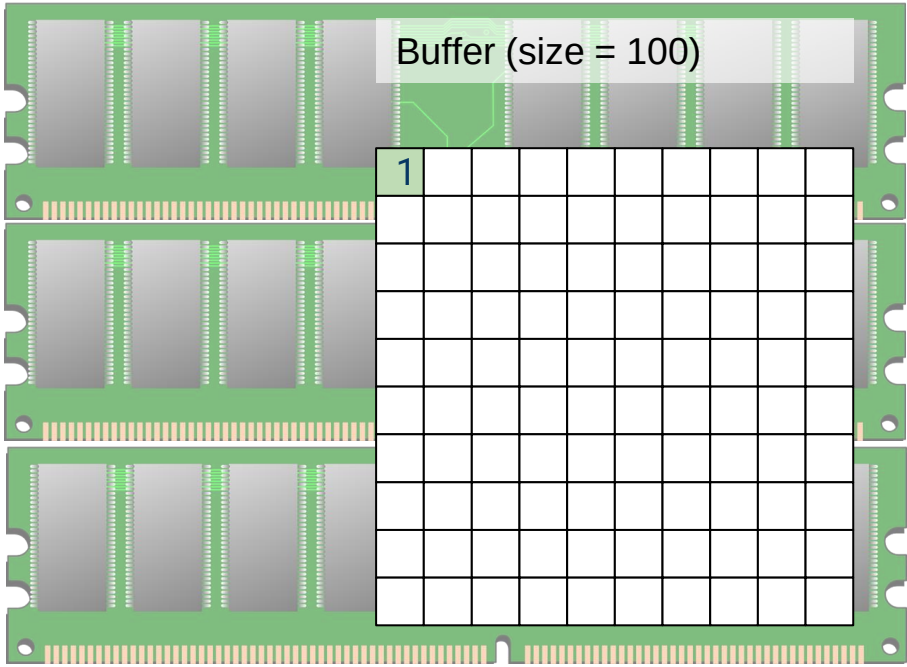

Blocks (απλούστευση)
Δίσκος


blocks/σελίδες
Ενδιάμεση Μνήμη (Buffer)
(size = 100)
Ram

Όταν ζητάμε ένα block και δεν υπάρχει στον buffer αυτό
- Δ**ιαβάζεται** από τον δίσκο και **γράφεται** στον buffer.
- **Καρφιτσώνεται** (pin) στον buffer δηλαδή δεν μπορεί να απομακρυνθεί.



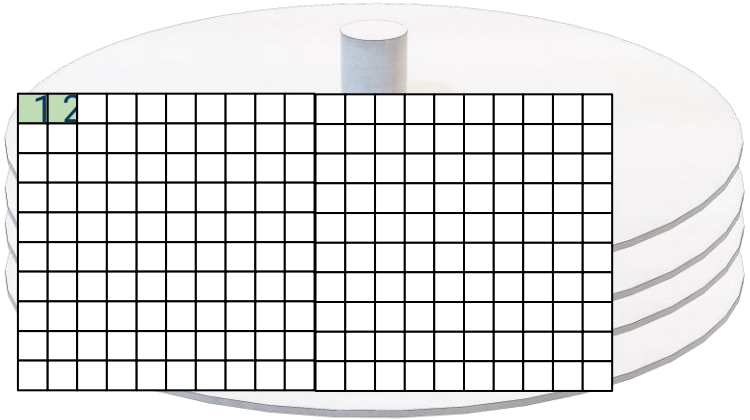
Δίσκος



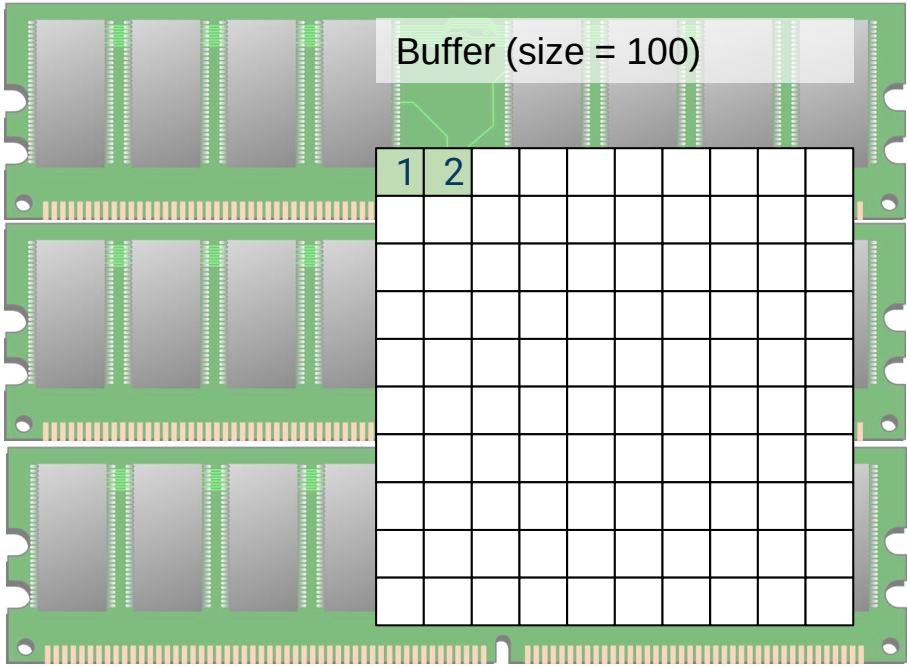
Ram

Έστω ότι ζητάω ένα ακόμα block.

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

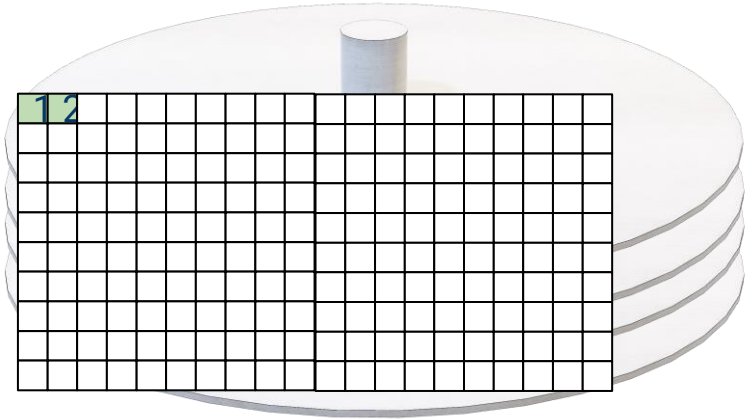


Δίσκος

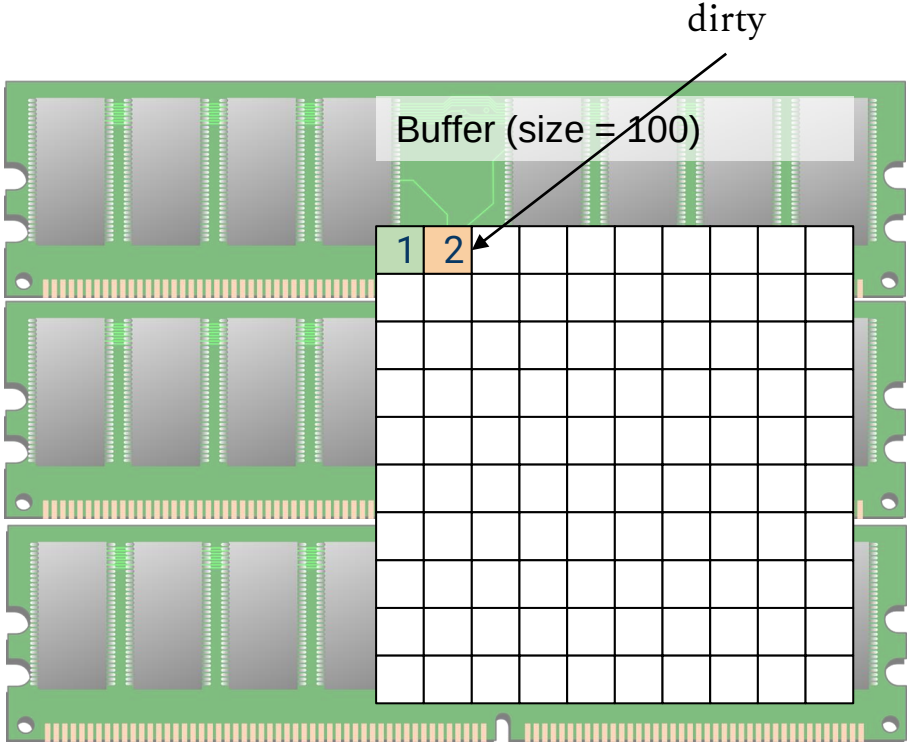


Ram

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**.

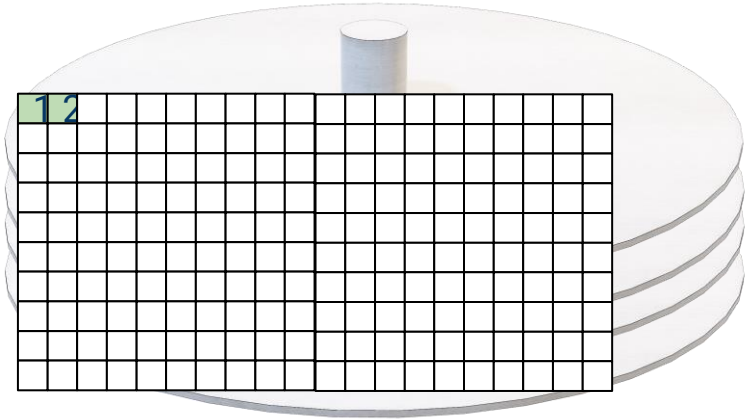


Δίσκος

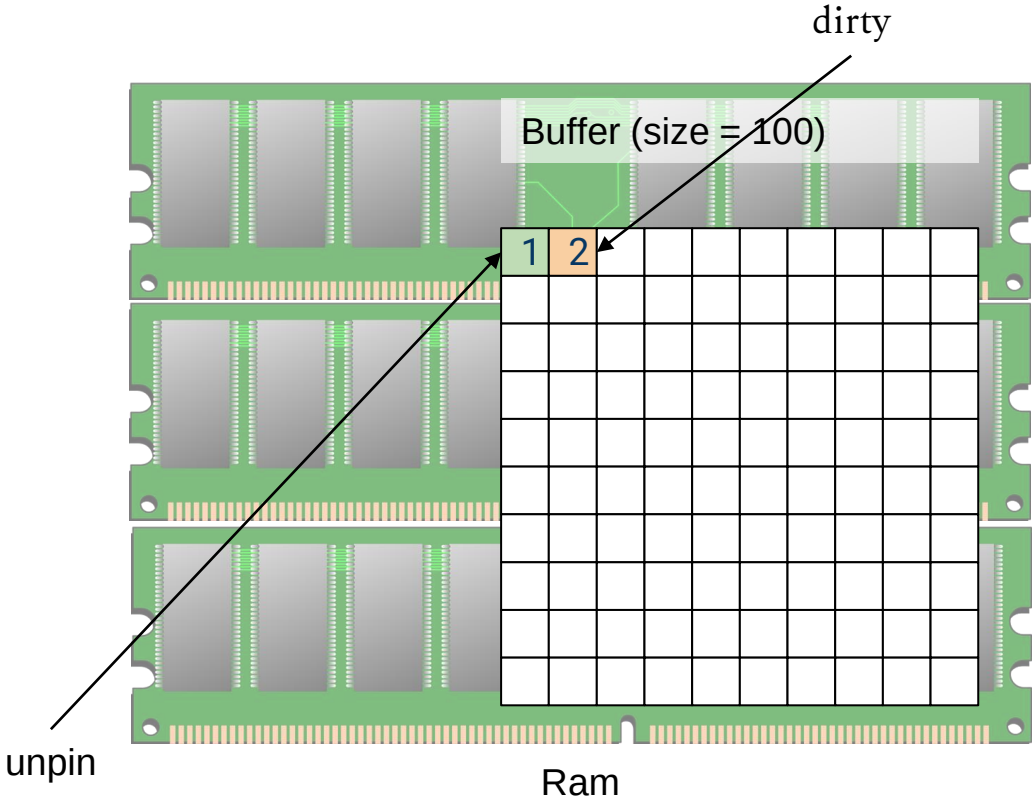


Ram

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



Δίσκος

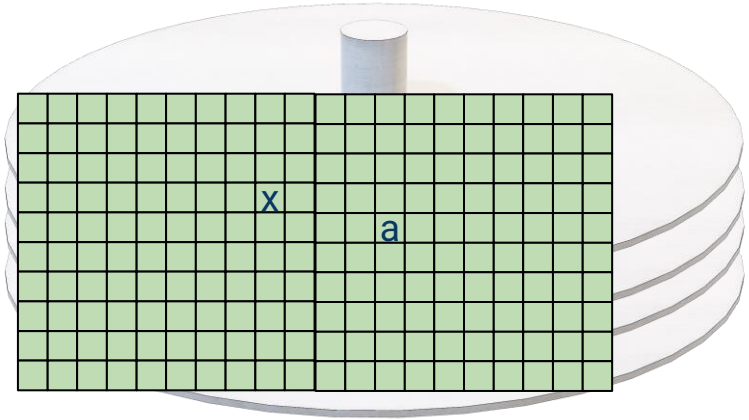


Ram

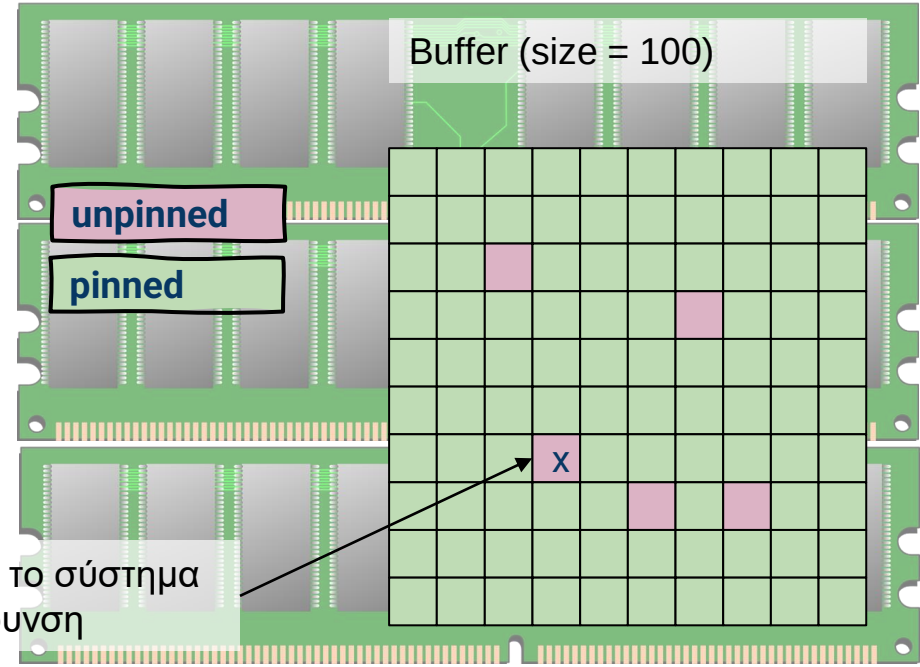
Ένα block στον buffer μπορεί να είναι

- **καρφιτσωμένο** (pinned)
- **καρφιτσωμένο** (pinned) και **βρώμικο** (dirty)
- **μη καρφιτσωμένο** (unpinned)
- **μη καρφιτσωμένο** (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 επιλεχθεί μεταξύ των unpinned



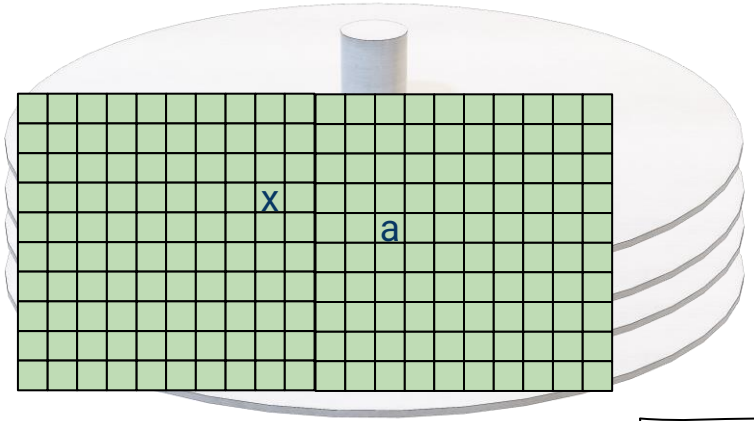
Δίσκος



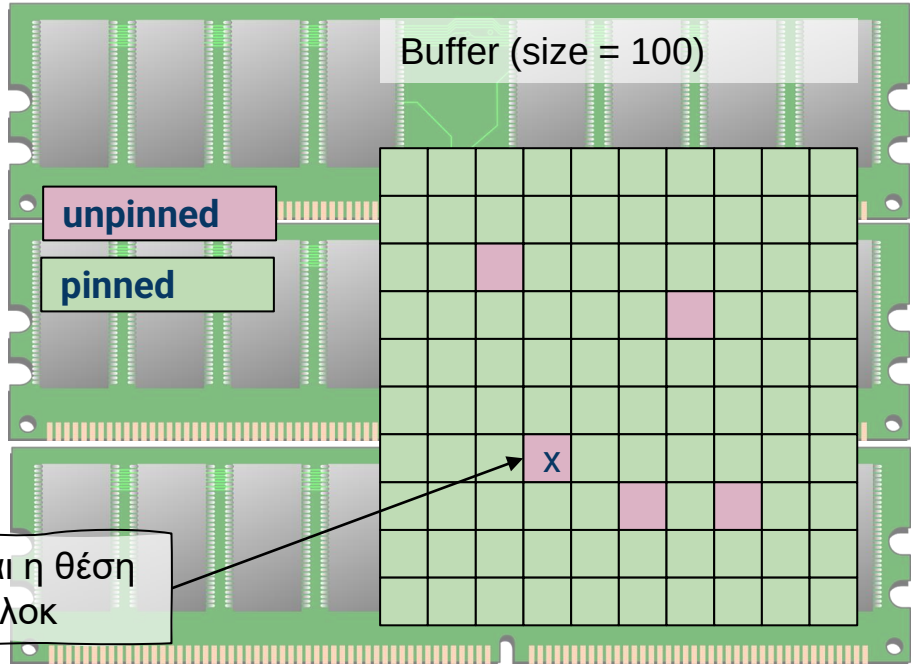
επιλογή από το σύστημα
για απομάκρυνση

Ram

Κάποια στιγμή ο buffer θα **γεμίσει**.
- Τότε για να αποθηκεύσει ένα καινούργιο block θα πρέπει να απομακρύνει κάποιο άλλο.
- Αυτό που θα απομακρυνθεί να επιλεχθεί μεταξύ των unpinned



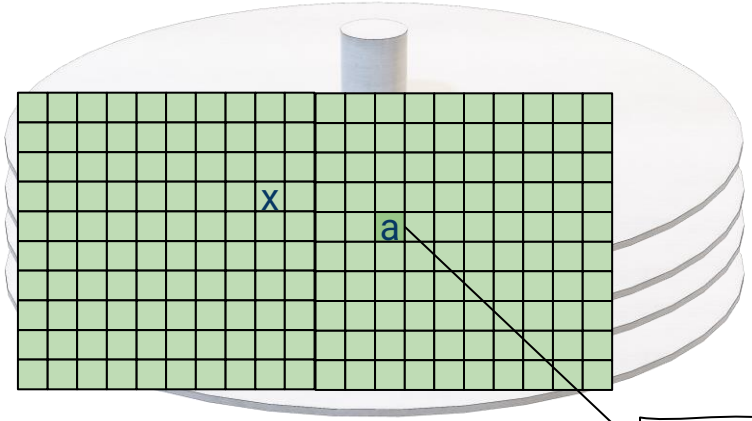
Δίσκος



Απελευθερώνεται η θέση από το παλιό μπλοκ

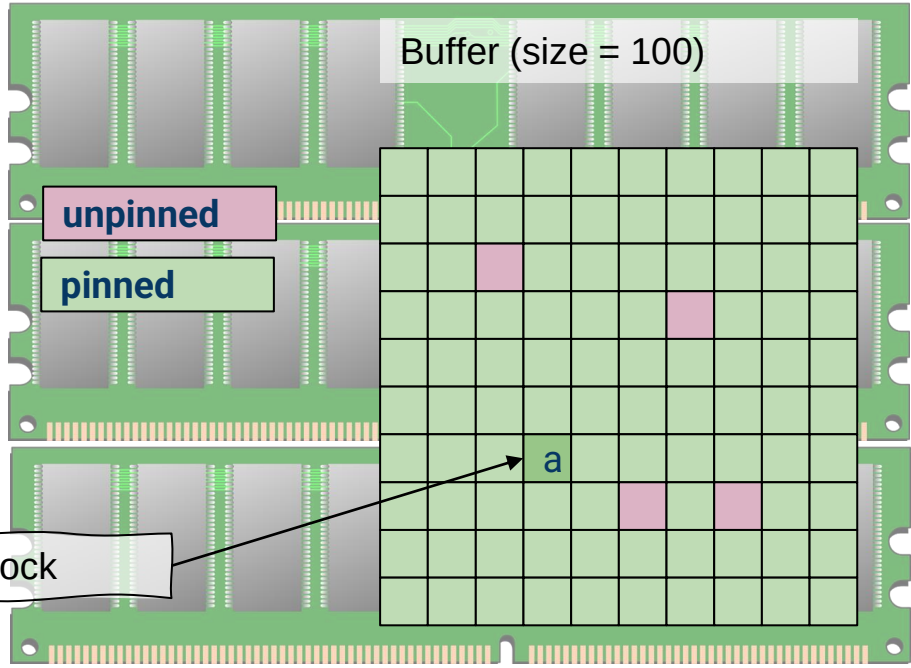
Ram

Κάποια στιγμή ο buffer θα **γεμίσει**.
- Τότε για να αποθηκεύσει ένα καινούργιο block θα πρέπει να απομακρύνει κάποιο άλλο.
- Αυτό που θα απομακρυνθεί να επιλεχθεί μεταξύ των unpinned



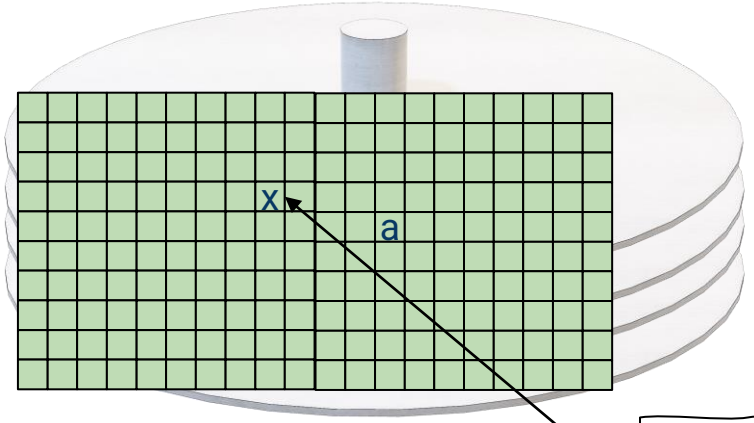
Δίσκος

Εγγραφή νέου block

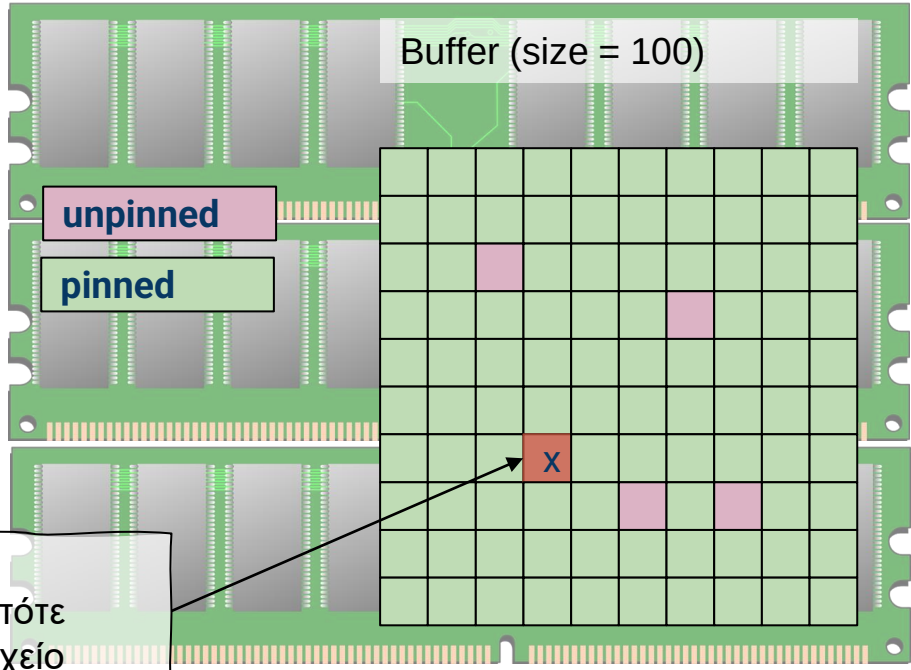


Ram

Αν το unpined block που επιλέγονται για απομάκρυνση από τον buffer manager **είναι και βρώμικα (dirty)**
- τότε θα **επανεγγραφούν στον δίσκο από τον buffer manager.**



Δίσκος



Ram

επιλογή για απομάκρυνση οπότε **εγγραφή** στο αρχείο

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 αλληλοεπιδρά 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 αλλαχθεί 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

- **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.

Συναρτήσεις
BF-Block

- **BF_ErrorCode BF_AllocateBlock(**

  const int file_desc, BF_Block *block)

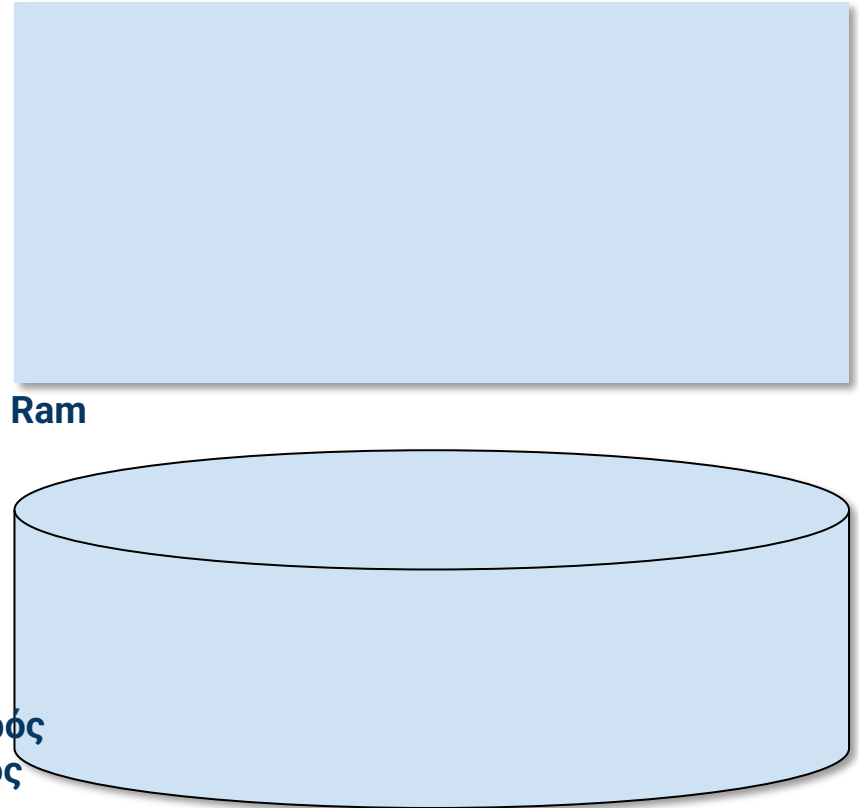
  Με τη συνάρτηση BF_AllocateBlock δεσμεύεται ένα καινούριο block για το αρχείο με αναγνωριστικό αριθμό blockFile. Το νέο block δεσμεύεται πάντα στο τέλος του αρχείου, οπότε ο αριθμός του block είναι BF_getBlockCounter(file_desc) - 1. Το block που δεσμεύεται καρφιτσώνεται στην μνήμη (pin) και επιστρέφεται 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());
```



Δημιουργία - Αρχείου

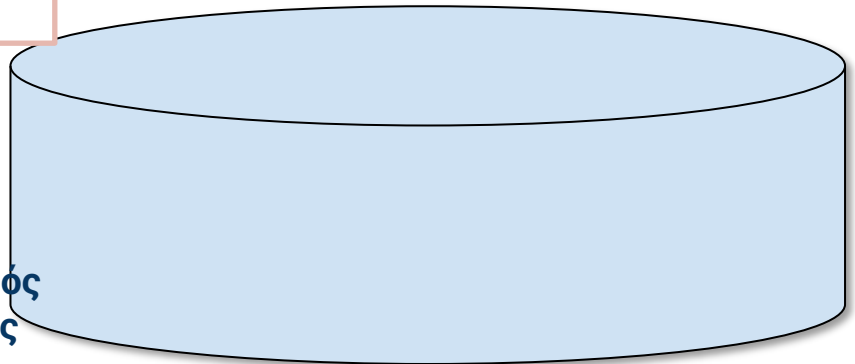
```
int fd1;
BF_Block* block;
void* data;

CALL_OR_DIE(BF_Init(LR
CALL_OR_DIE(BF_CreateF
CALL_OR_DIE(BF_OpenFil
BF_Block_Init(&block);

for (int i = 0; i < 10
  CALL_OR_DIE(BF_Alloc
  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());
```

- 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 που περιέχει δεδομένα

m



Σκληρός Δίσκος

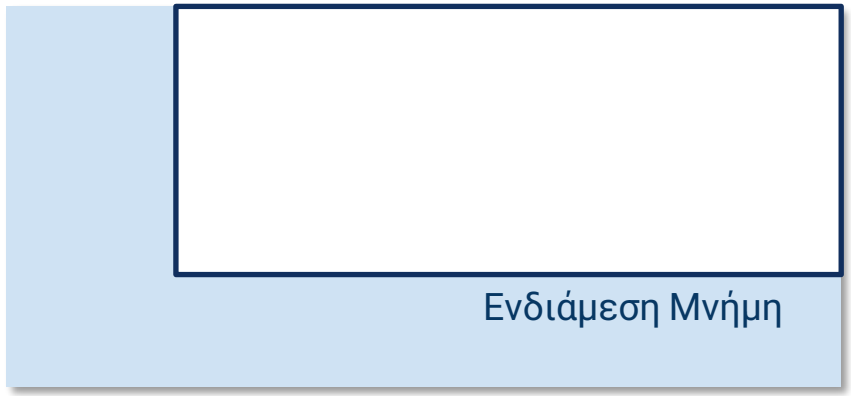
Δημιουργία - Αρχείου

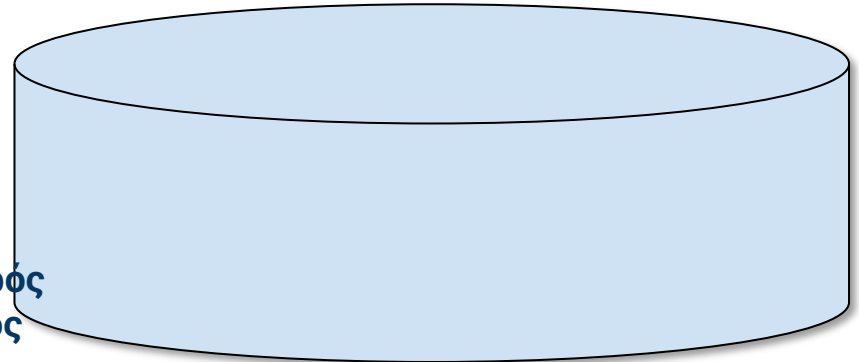
```
int fd1;
BF_Block* block;
void* data;

CALL_OR_DIE(BF_Init(LRU));
CALL_OR_DIE(BF_C
CALL_OR_DIE(BF_O
BF_Block_Init(&b

for (int i = 0;
  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



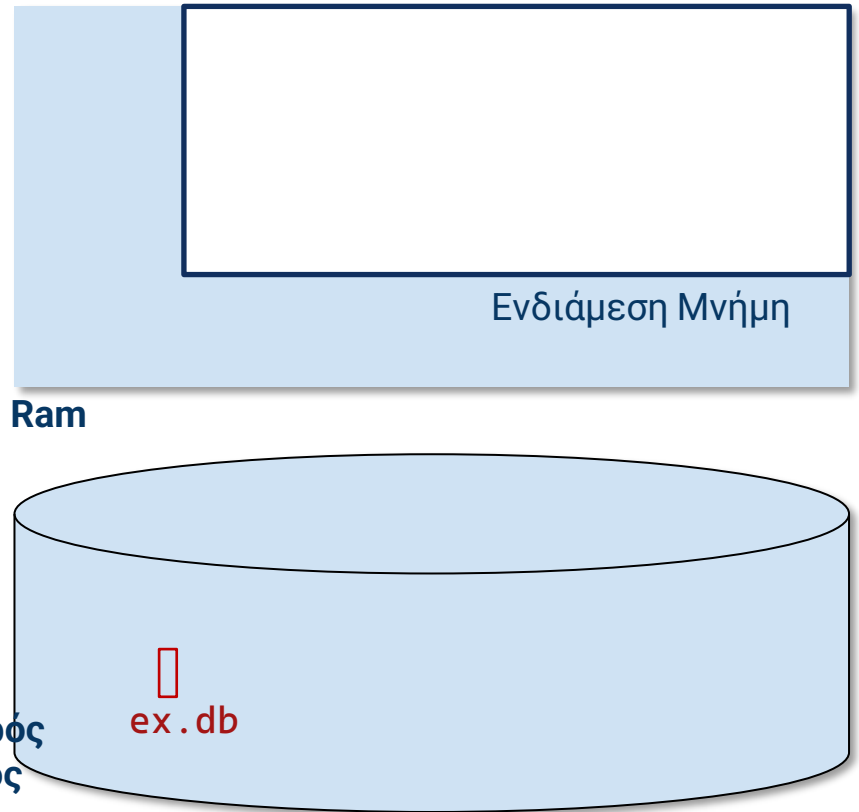


Δημιουργία - Αρχείου

```
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(&
for (int i = 0;
  CALL_OR_DIE(B
  data = BF_Blo
  Record* rec =
  rec[0] = rand
  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



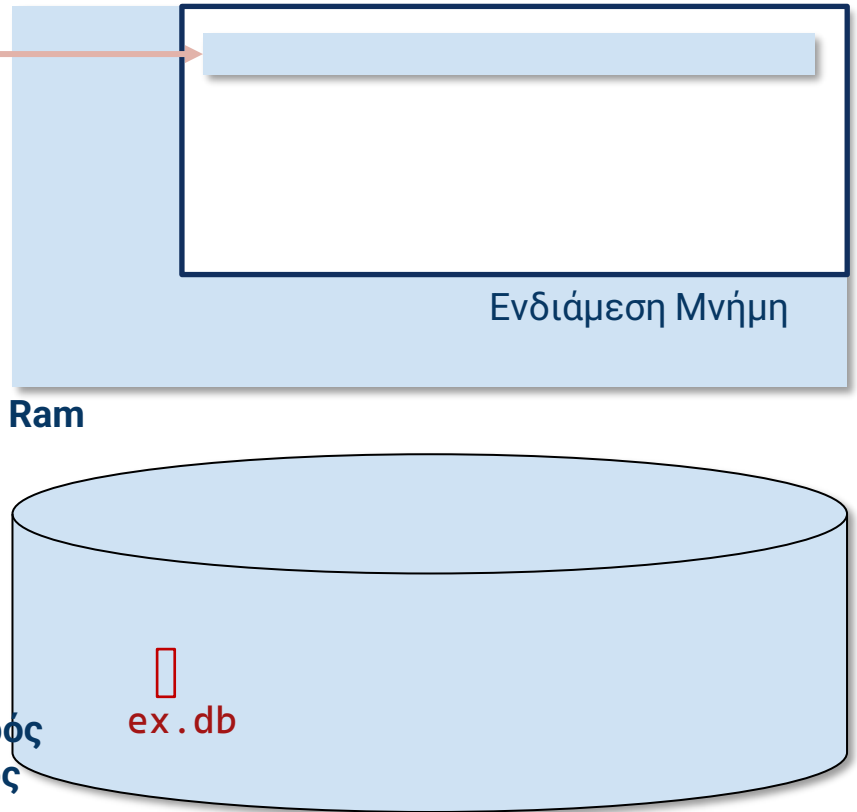
Δημιουργία - Αρχείου

```
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;
  CALL_OR_DIE(B
  data = BF_Blo
  Record* rec =
  rec[0] = rand
  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 μία **σελίδα/μπλοκ** από την ενδιάμεση μνήμη μου



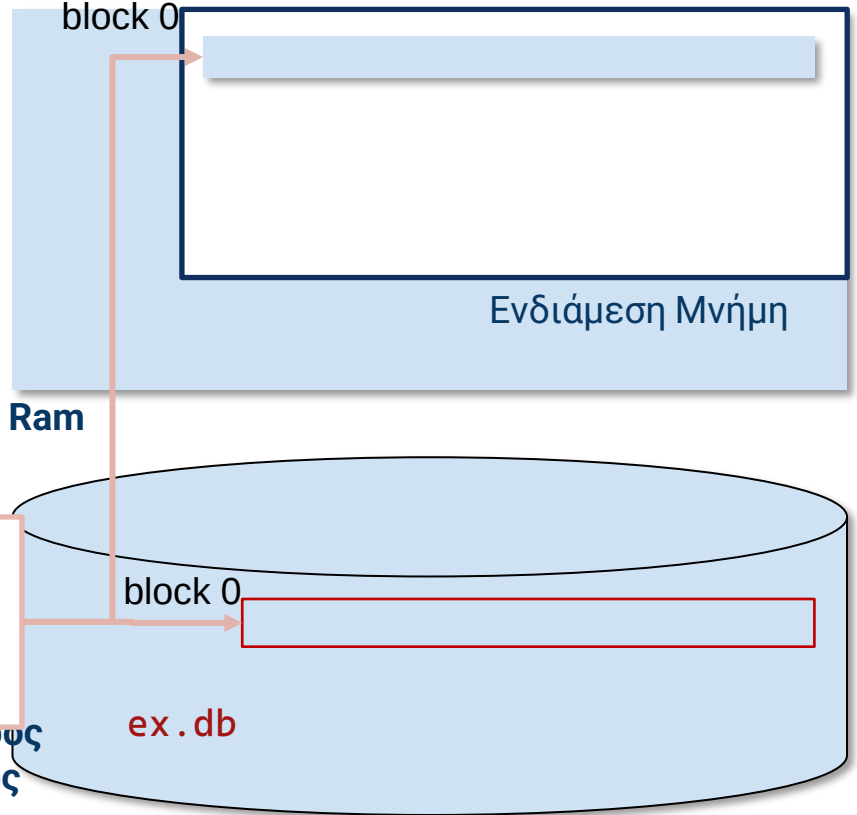
Δημιουργία - Αρχείου

```
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_Get
  Record* r
  rec[0] =
  rec[1] =
  BF_Block_
  CALL_OR_D
 }
 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



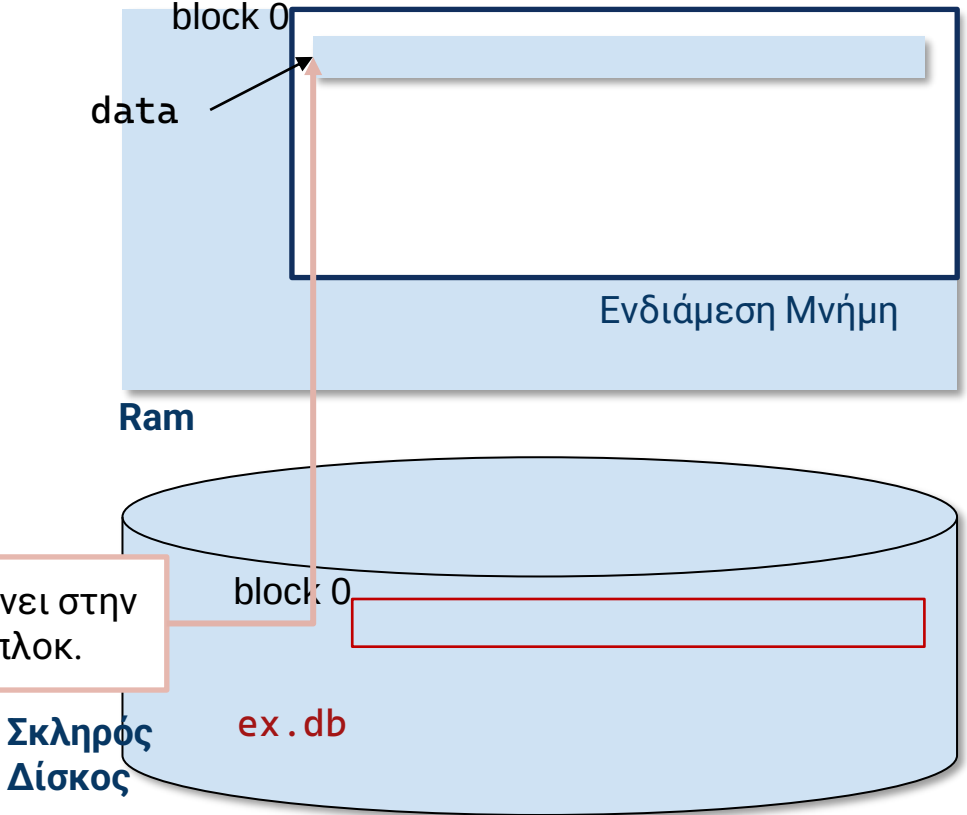
Δημιουργία - Αρχείου

```
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("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] = ra
  rec[1] = ra
  BF_Block_Se
  CALL_OR_DIE
 }
 CALL_OR_DIE(BF_CloseFile(fd1));
 CALL_OR_DIE(BF_Close());
```

- Δημιουργώ τον δείκτη **data** που δείχνει στην αρχή των δεδομένων της σελίδας/μπλοκ.



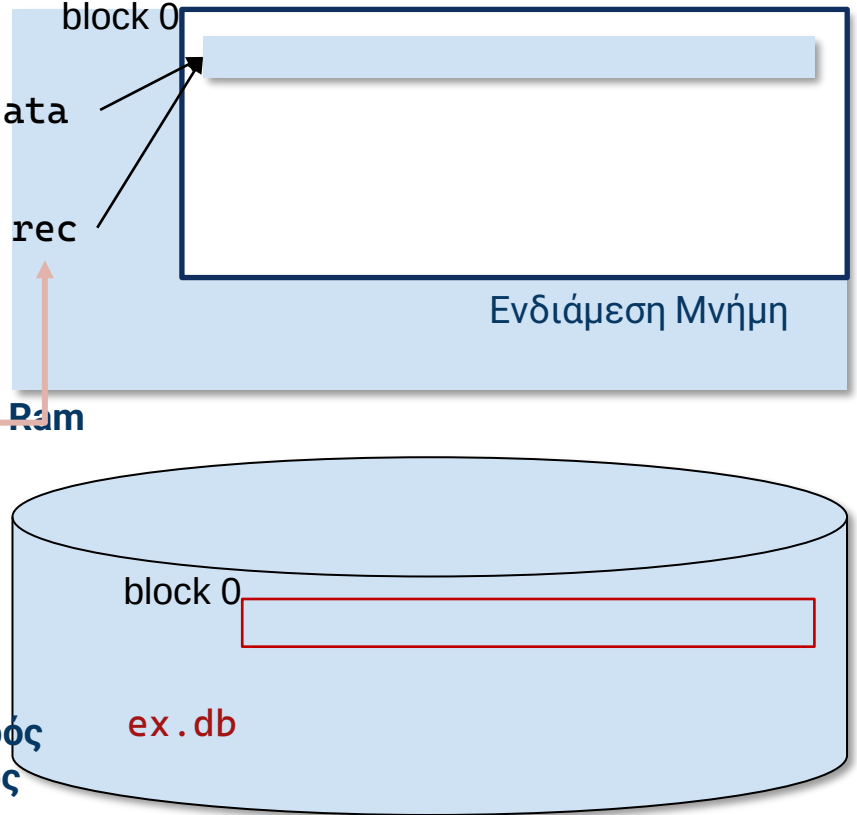
Δημιουργία - Αρχείου

```
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_AllocateBlock(fd1, block));
  data = BF_Block_GetData(block);
  Record* rec = data;
  rec[0] = randomR
  rec[1]
  BF_Bloc
  CALL_OR
 }
 CALL_OR_
 CALL_OR_DIE(BF_Close());
```

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



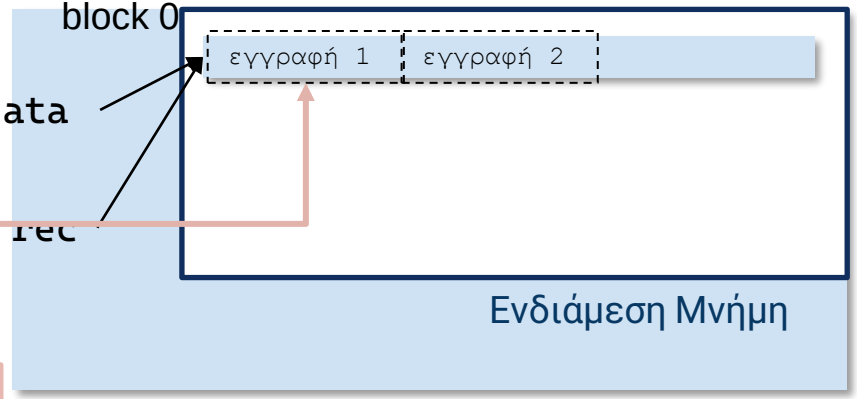
Δημιουργία - Αρχείου

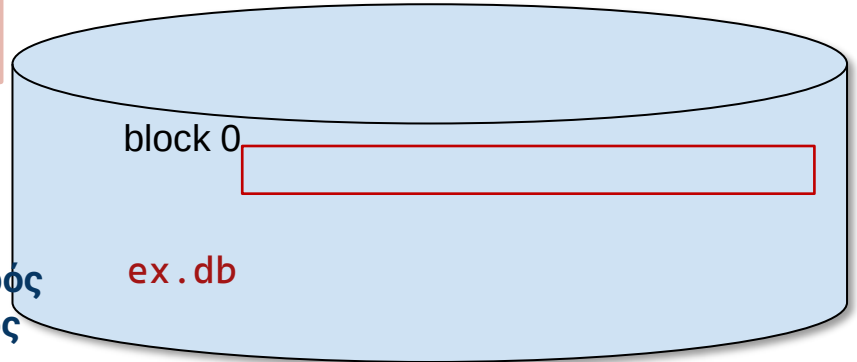
```
int fd1;
BF_Block* block;
void* data;

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

for (int i
  CALL_OR_
  data = B
  Record* rec = dat
  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());
```

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



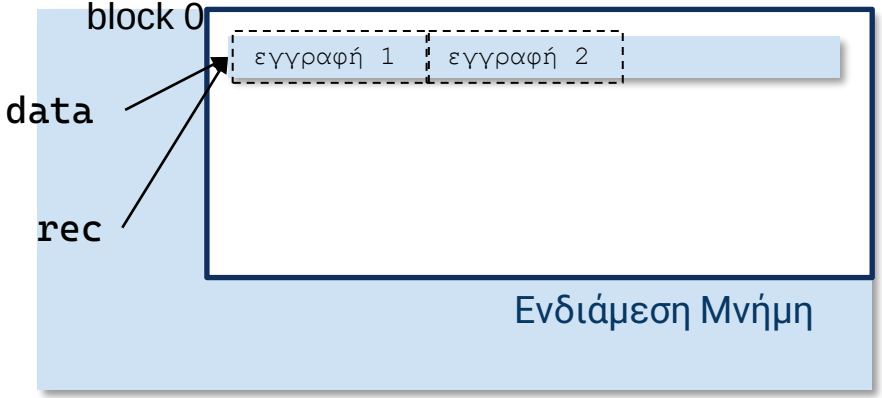


Δημιουργία - Αρχείου

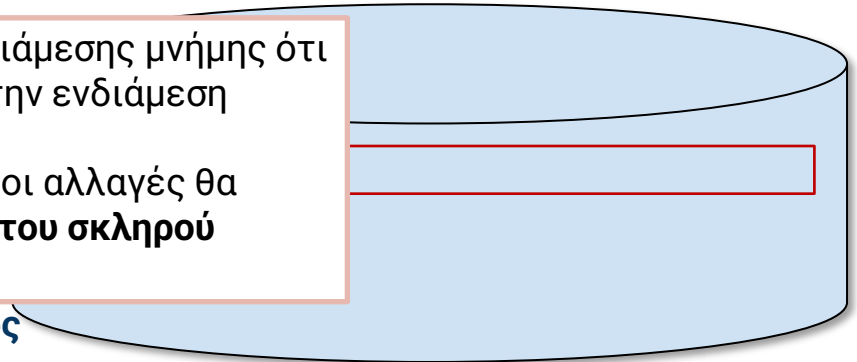
```
int fd1;
BF_Block* block;
void* data;

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(
  data = BF_Block_GetData(block
  Record* rec = data;
  rec[0] = randomRecord();
  rec[1] = randomRecord();
  BF_Block_SetDirty(block);
  CALL_OR_DIE(BF_UnpinBlock(b
 }
 CALL_OR_DIE(BF_CloseFile(fd1));
 CALL_OR_DIE(BF_Close());
```



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

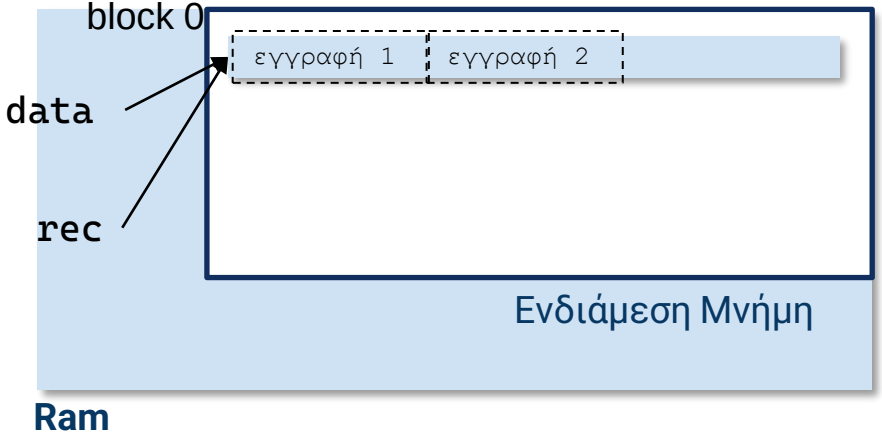


Δημιουργία - Αρχείου

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

CALL_OR_DIE(BF_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(
  data = BF_Block_GetData(block
  Record* rec = data;
  rec[0] = randomRecord();
  rec[1] = randomRecord();
  BF_Block_SetDirty(block);
  CALL_OR_DIE(BF_UnpinBlock
 }
 CALL_OR_DIE(BF_CloseFile(fd1))
 CALL_OR_DIE(BF_Close());
```



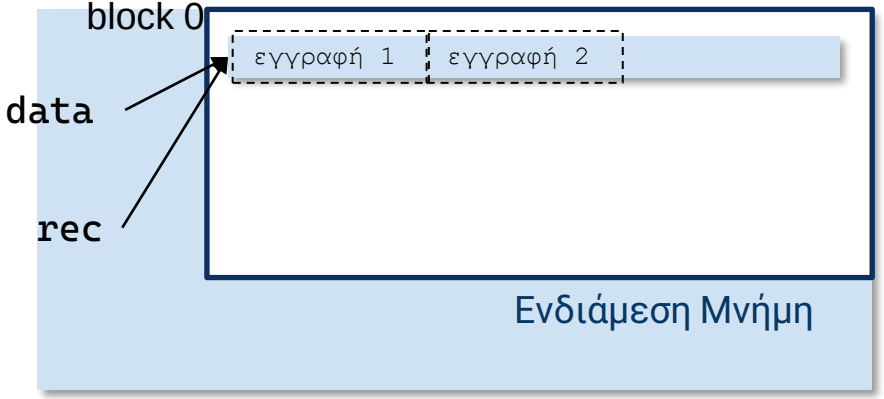
- Λέω στο σύστημά μου, ότι δεν χρειάζομαι πλέον την συγκεκριμένη σελίδα ενδιάμεσης μνήμης οπότε να γίνει **unpin**
- Αυτό σημαίνει ότι βάσει της πολιτικής αντικατάστασης κάποια άλλη σελίδα μπορεί να δεσμεύσει την συγκεκριμένη περιοχή μνήμης

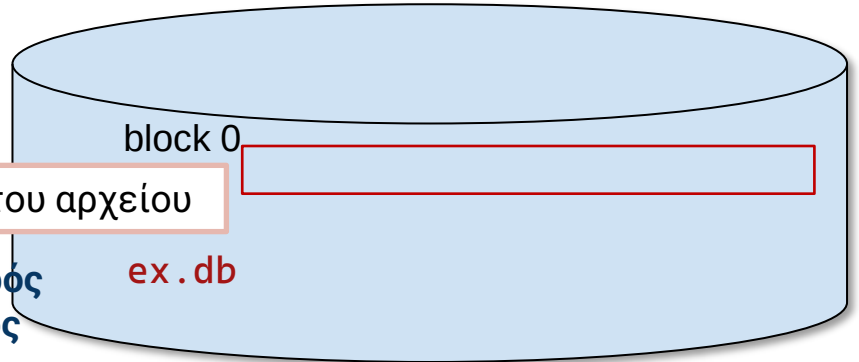
Δημιουργία - Αρχείου

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





Τερματίζω την επεξεργασία του αρχείου

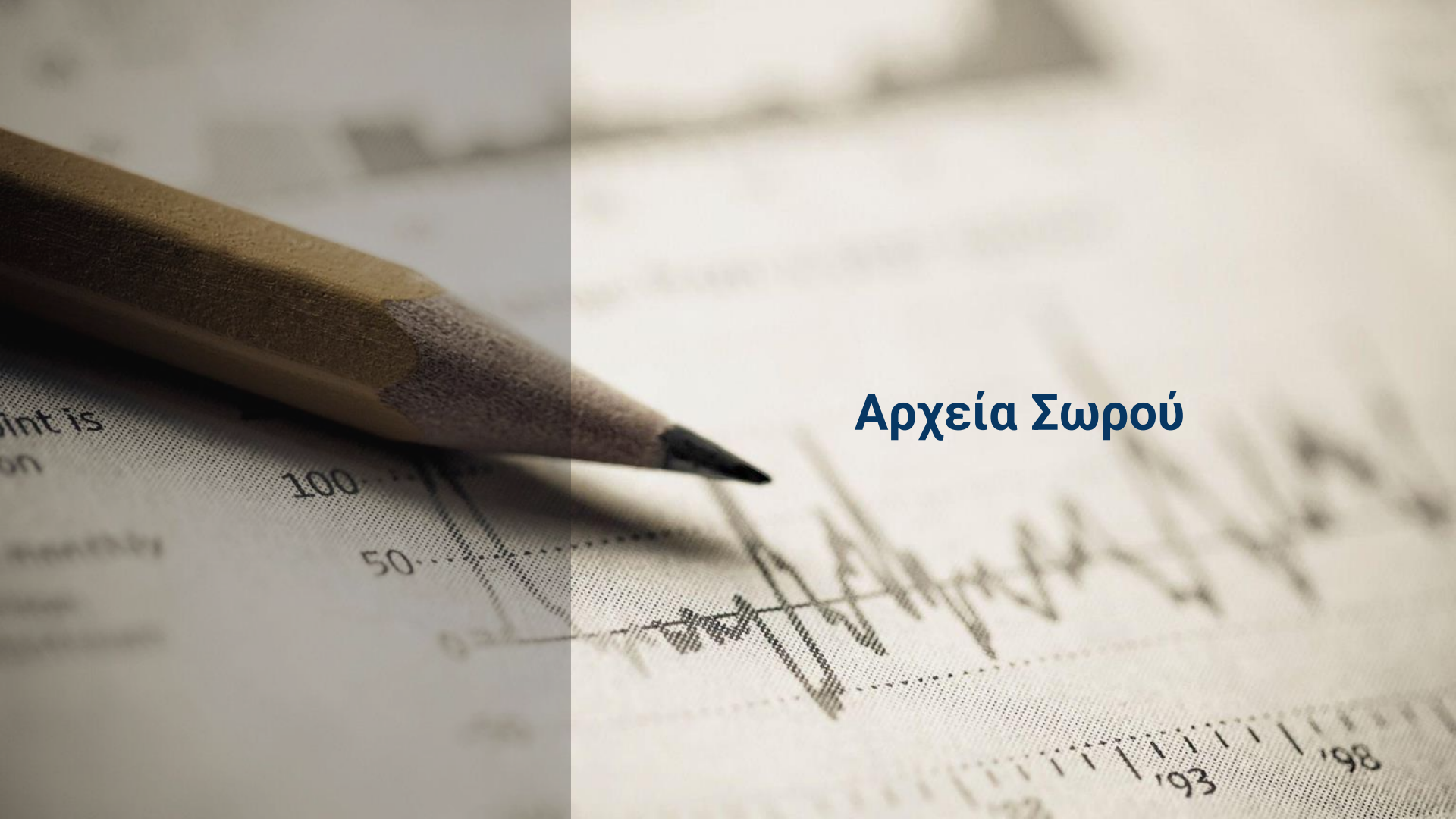
Αναφορές

- 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

1η Εργασία

Στόχος

- Καλείστε να υλοποιήσετε μια σειρά συναρτήσεων που διαχειρίζονται
  - **αρχεία Σωρού** και
  - **αρχεία στατικού κατακερματισμού**
  - χρησιμοποιώντας την **βιβλιοθήκη BF.h**



Αρχεία Σωρού

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 Franscisco”

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;

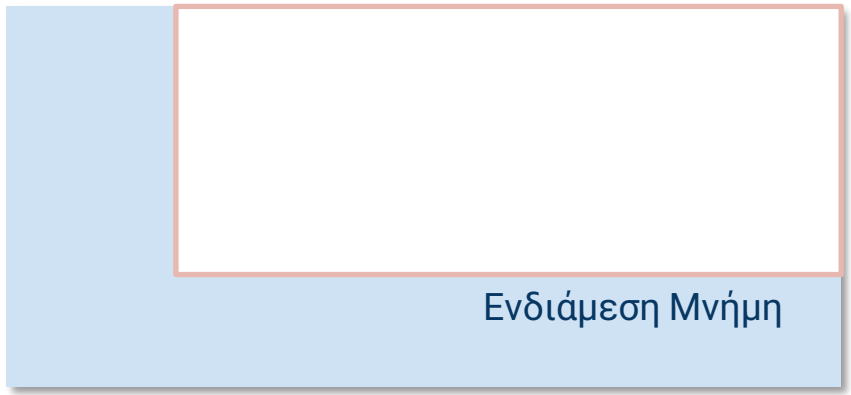
Αρχεία Σωρού (Μη ταξινομημένα)

- 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**

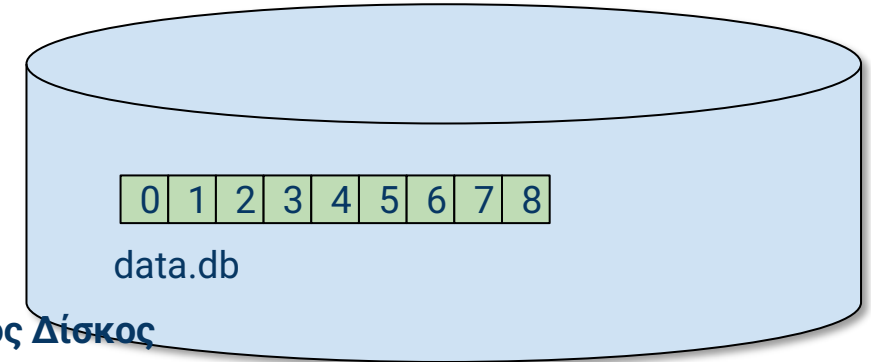
Εισαγωγή Εγγραφών

- 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



Ram



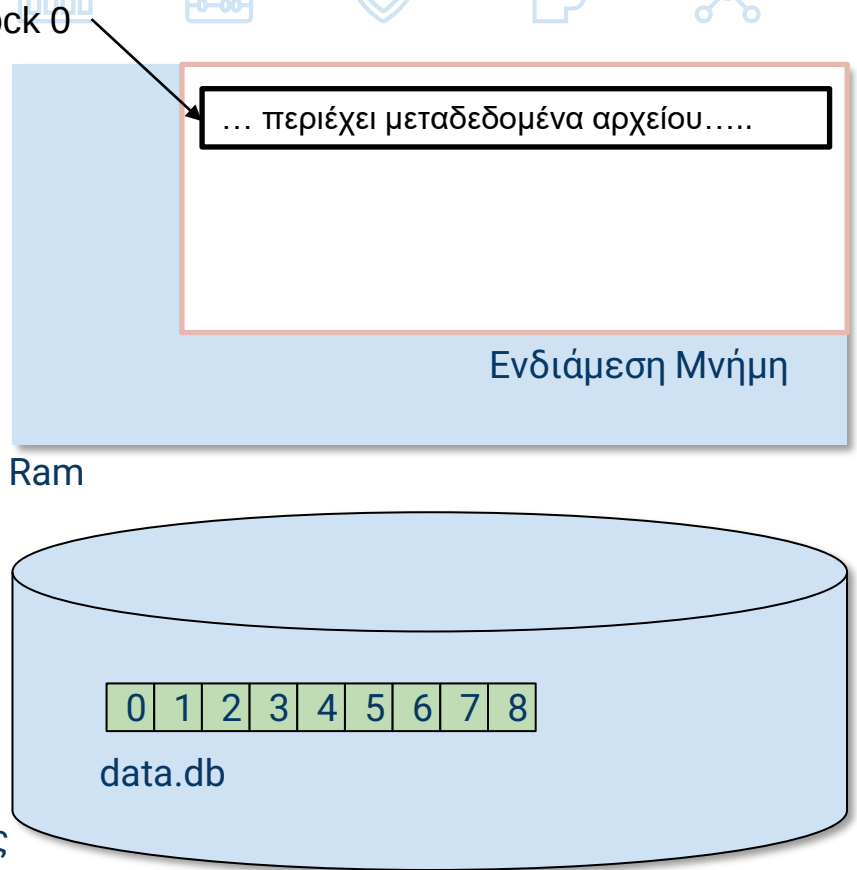
Σκληρός Δίσκος

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

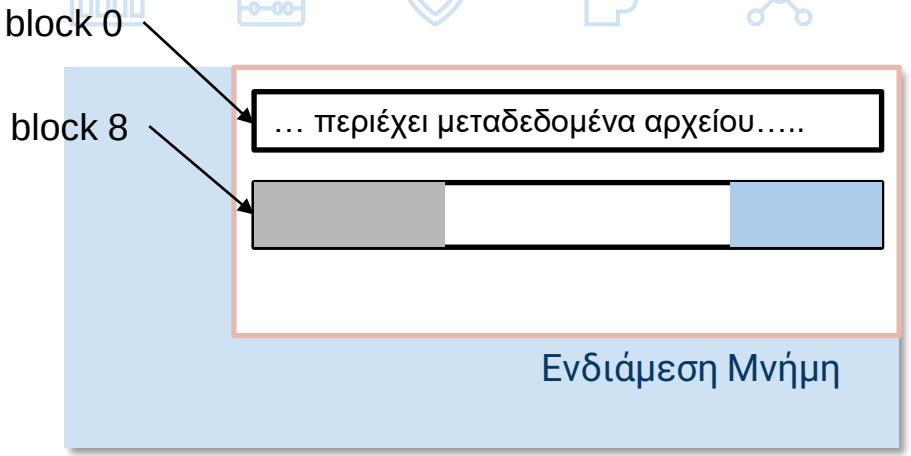
Βήμα 1: Φορτώνω το μπλοκ 0 του αρχείου στην ενδιάμεση μνήμη



Εισαγωγή Εγγραφών

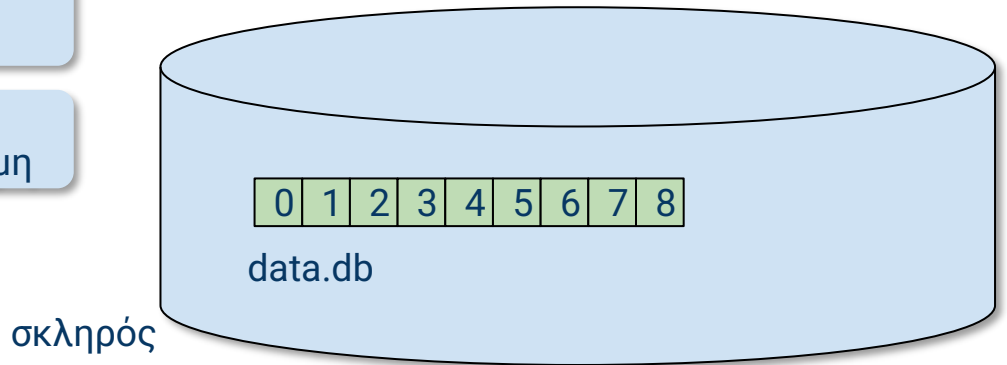
- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής
  (15, “Marianna”, “Rezkalla”, “Hong Kong”)

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



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")

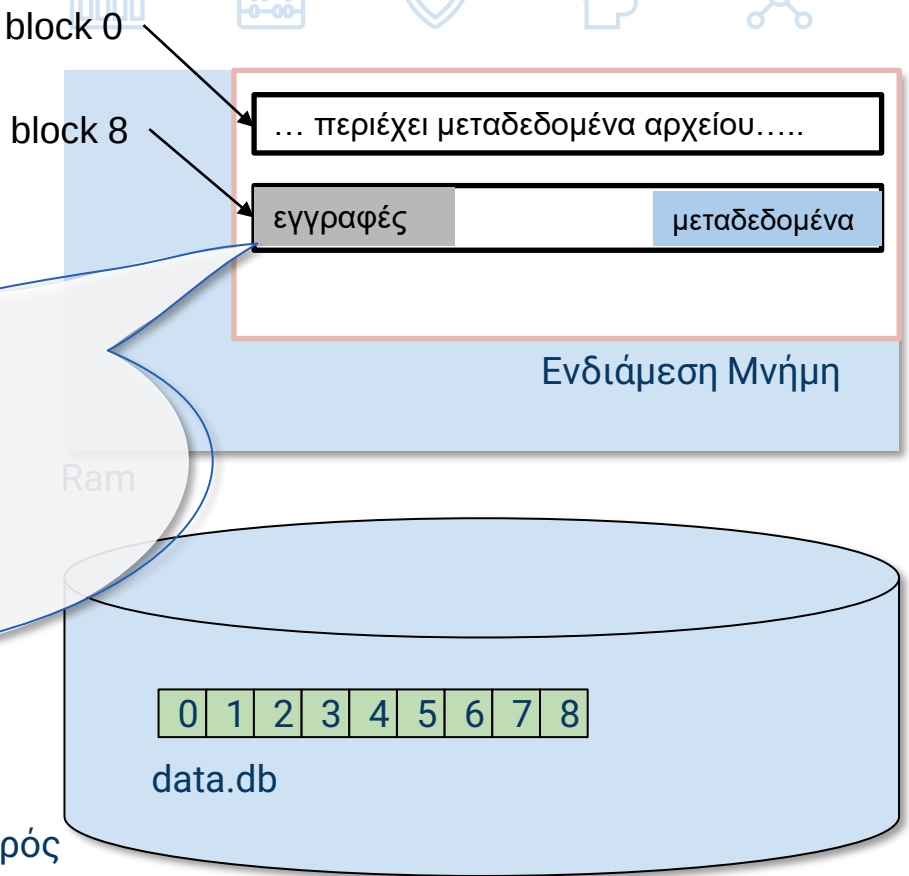
Μέγεθος εγγραφής: 76 bytes

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: Φορτώνω το μπλοκ του αρχείου στην ενδιάμεση μνήμη

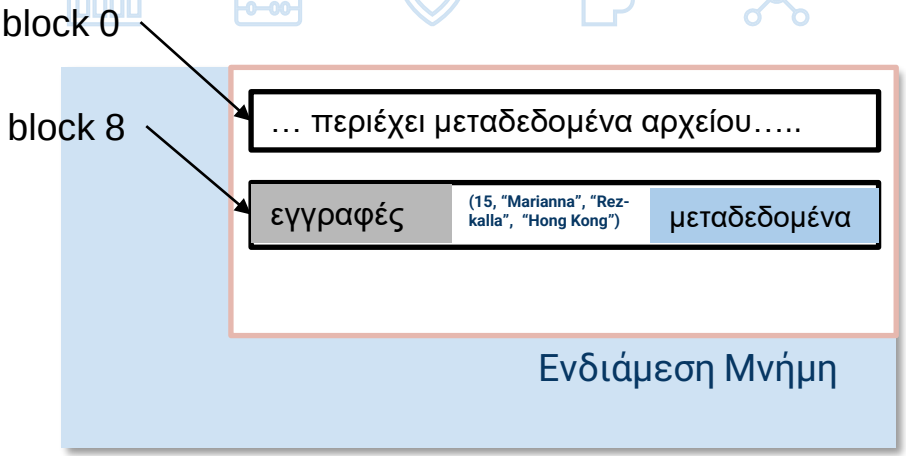
Βήμα 2: Βρίσκω ποιο είναι το τελευταίο block του αρχείου μου και το φορτώνω στην ενδιάμεση μνήμη



Εισαγωγή Εγγραφών

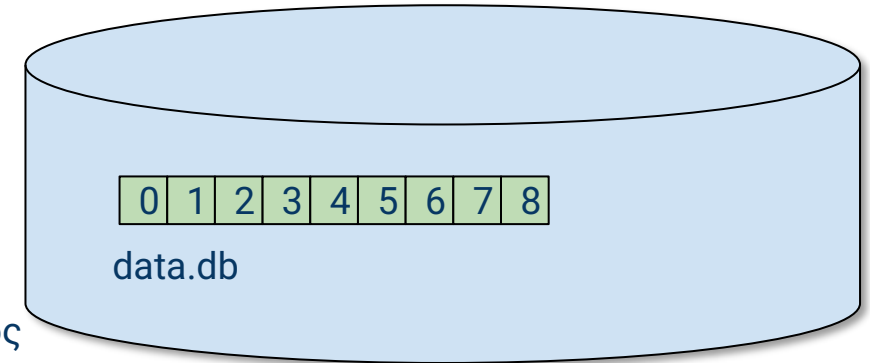
- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής
  (15, “Marianna”, “Rezkalla”, “Hong Kong”)

Μέγεθος εγγραφής: 76 bytes



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

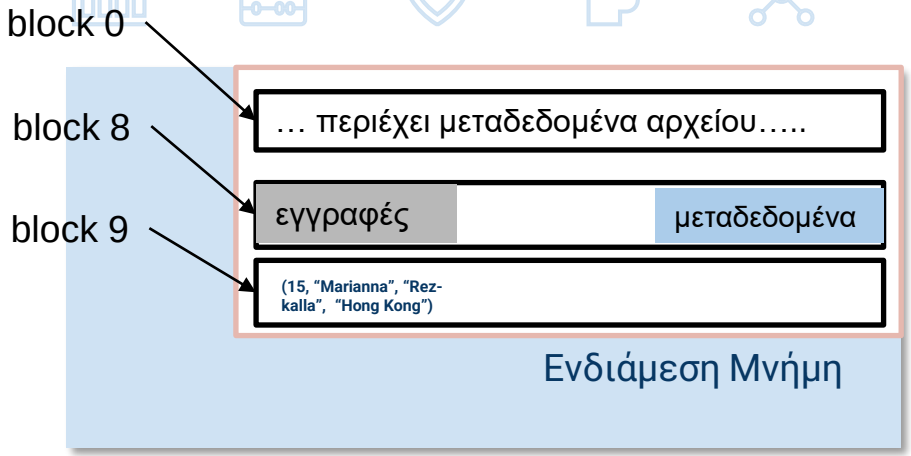
- Επίσης κάνω το block 8 dirty και unpin εφόσον έχω ολοκληρώσει τις πράξεις μου σε αυτό



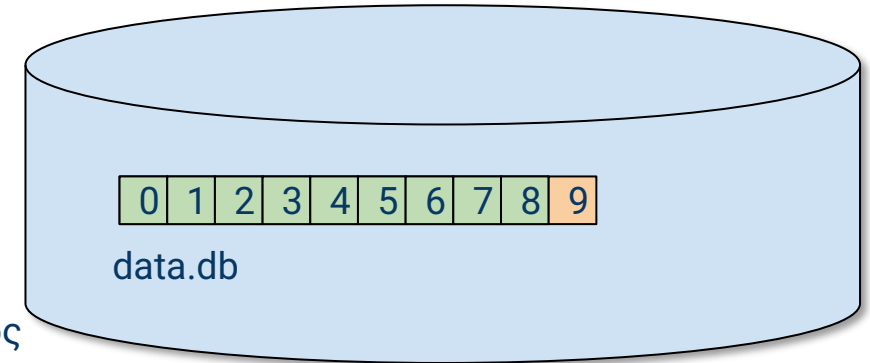
Εισαγωγή Εγγραφών

- 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



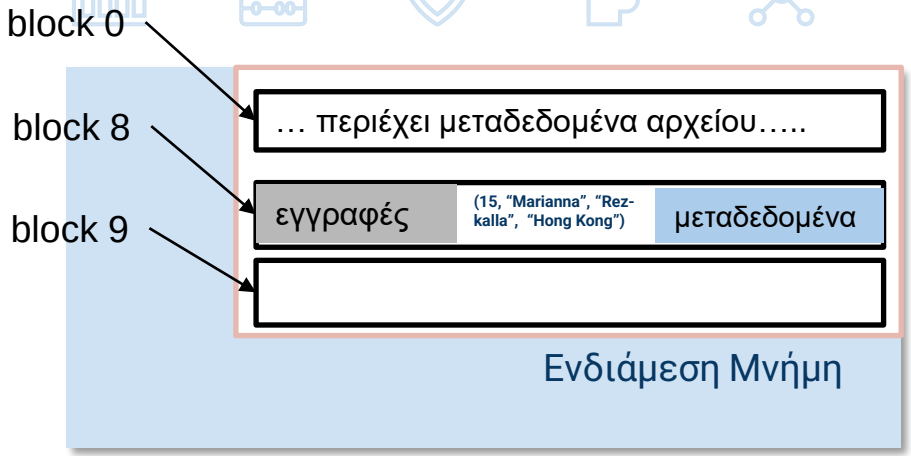
Βήμα 3β: Αν δεν χωράει κάνω allocate καινούριο block (block 9) το φορτώνω στην ενδιάμεση μνήμη και προχωράω αναλόγως την εισαγωγή μου



Εισαγωγή Εγγραφών

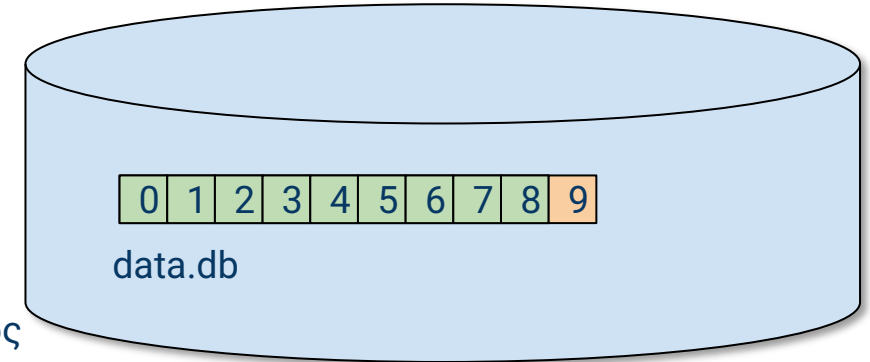
- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής
  (15, “Marianna”, “Rezkalla”, “Hong Kong”)

Μέγεθος εγγραφής: 76 bytes



Βήμα 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

HP_Info
Δομή Μεταδεδομένων Αρχείου Σωρού

- Μια **ενδεικτική δομή** με τις πληροφορίες που πρέπει να κρατάτε δίνεται στη συνέχεια:

```
typedef struct {
     int fileDesc; /* αναγνωριστικός αριθμός ανοίγματος
     αρχείου από το επίπεδο block */
} HP_info;
```

- Όπου **fileDesc** είναι ο αναγνωριστικός αριθμός του ανοίγματος του αρχείου, όπως επιστράφηκε από το επίπεδο διαχείρισης μπλοκ.
- Η συγκεκριμένη δομή μπορεί να κρατάει επιπλέον πληροφορίες όπως: το block στο οποίο είναι αποθηκευμένη, το id του τελευταίου block του αρχείου σωρού, τον αριθμό των εγγραφών που χωράνε σε κάθε block του αρχείου σωρού.

HP_block_info Δομή Μεταδεδομένων του block ενός Αρχείου Σωρού

```
typedef struct {
        ...
} HP_block_info;
```

- Στο τέλος κάθε block θα πρέπει να υπάρχει μία δομή στην οποία θα αποθηκεύετε πληροφορίες σε σχέση με το block όπως: τον αριθμό των εγγραφών στο συγκεκριμένο block, έναν δείκτη στο επόμενο block δεδομένων.

HP_CreateFile

```
int HP_CreateFile(
    char *fileName,    /*όνομα αρχείου*/);
```

- Η συνάρτηση HP_CreateFile χρησιμοποιείται για τη δημιουργία και κατάλληλη αρχικοποίηση ενός άδειου αρχείου σωρού με όνομα fileName.
- Στα πλαίσια της δημιουργίας, αποθηκεύεται στο πρώτο block του αρχείου σωρού ένα στιγμιότυπο του HP_block_info που κρατάει τα μεταδεδομένα της δομής.
- Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1.

HP_OpenFile

```
HP_info* HP_OpenFile(
    char *fileName /* όνομα αρχείου */ )
```

- Η συνάρτηση HP_OpenFile ανοίγει το αρχείο με όνομα filename και διαβάζει από το πρώτο μπλοκ την πληροφορία που αφορά το αρχείο σωρού.
- Επιστρέφετε η δομή πληροφοριών του αρχείου. Σε περίπτωση που συμβεί οποιοδήποτε σφάλμα, επιστρέφεται τιμή NULL.
- Αν το αρχείο που δόθηκε για άνοιγμα δεν αφορά αρχείο κατακερματισμού, τότε αυτό επίσης θεωρείται σφάλμα.

HP_CloseFile

int HP_CloseFile(HP_info* header_info)

- Η συνάρτηση HP_CloseFile κλείνει το αρχείο που προσδιορίζεται μέσα στη δομή header_info.
- Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφεται 0, ενώ σε διαφορετική περίπτωση -1.
- Η συνάρτηση είναι υπεύθυνη και για την αποδέσμευση της μνήμης που καταλαμβάνει η δομή που περάστηκε ως παράμετρος, στην περίπτωση που το κλείσιμο πραγματοποιήθηκε επιτυχώς.

HP_InsertEntry

```
int HP_InsertEntry(
    HP_info*  header_info,  /*  επικεφαλίδα  του
αρχείου*/
    Record  record  /*  δομή  που  προσδιορίζει  την
εγγραφή */ )
```

- Η συνάρτηση HP_InsertEntry χρησιμοποιείται για την εισαγωγή μιας εγγραφής στο αρχείο σωρού.
- Οι πληροφορίες που αφορούν το αρχείο βρίσκονται στη δομή header_info, ενώ η εγγραφή προς εισαγωγή προσδιορίζεται από τη δομή record.
- Σε περίπτωση που εκτελεστεί επιτυχώς, επιστρέφετε τον αριθμό του block στο οποίο έγινε η εισαγωγή (blockId) , ενώ σε διαφορετική περίπτωση -1.

HP_GetAllEntries

```
int HP_GetAllEntries(
    HP_info* header_info, /* επικεφαλίδα του αρχείου*/
    int id /* η τιμή id της εγγραφής στην οποία πραγματοποιείται η αναζήτηση*/)
```

- Η συνάρτηση αυτή χρησιμοποιείται για την εκτύπωση όλων των εγγραφών που υπάρχουν στο αρχείο κατακερματισμού οι οποίες έχουν τιμή στο πεδίο-κλειδί ίση με value.
- Η πρώτη δομή δίνει πληροφορία για το αρχείο κατακερματισμού, όπως αυτή είχε επιστραφεί από την HP_OpenFile.
- Για κάθε εγγραφή που υπάρχει στο αρχείο και έχει τιμή στο πεδίο id ίση με value, **εκτυπώνονται τα περιεχόμενά της** (συμπεριλαμβανομένου και του πεδίου-κλειδιού).
- Να επιστρέφεται επίσης το πλήθος των blocks που διαβάστηκαν μέχρι να βρεθούν όλες οι εγγραφές.
- Σε περίπτωση επιτυχίας επιστρέφει το **πλήθος των blocks που διαβάστηκαν**, ενώ σε περίπτωση λάθους επιστρέφει -1.

Αρχεία Κατακερματισμού

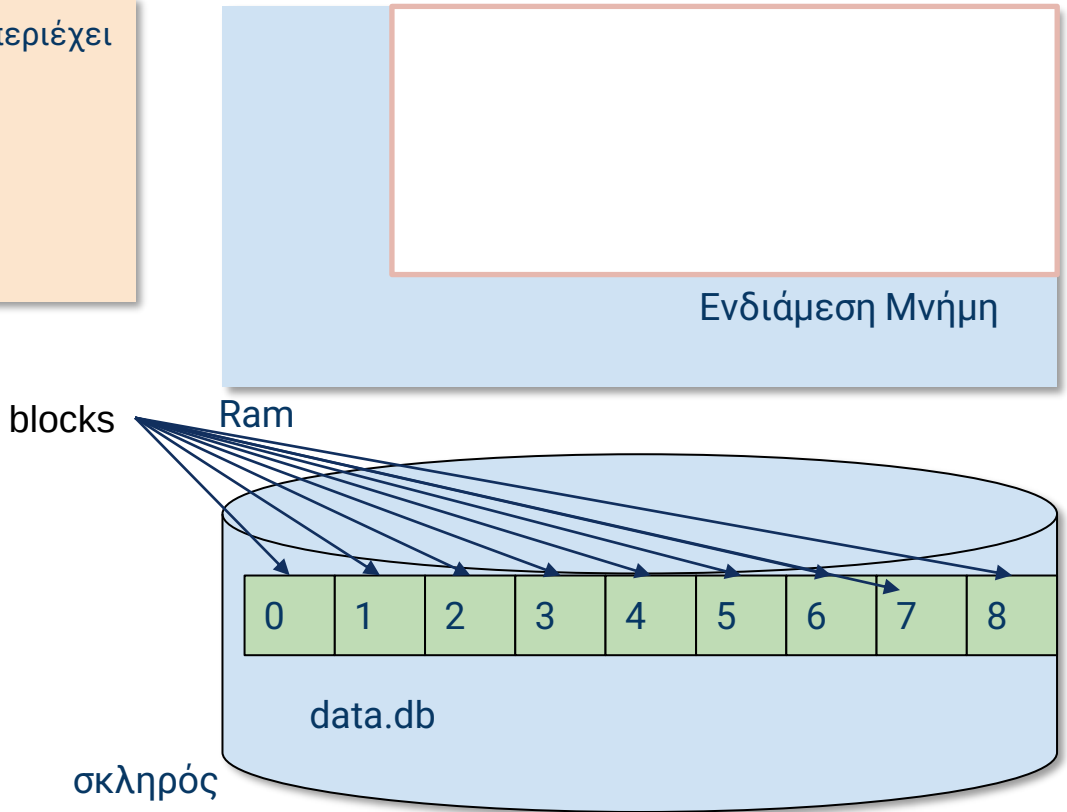
Αρχεία Κατακερματισμού

- Ο κατακερματισμός για αρχεία στο δίσκο ονομάζεται **Εξωτερικός Κατακερματισμός**
- Τα μπλοκ του αρχείου χωρίζονται σε M ίσου μεγέθους κάδους, αριθμημένους κάδος$_0$, κάδος$_1$, ..., κάδος$_{M-1}$.
  - Τυπικά, ένας κάδος αντιστοιχεί σε ένα μπλοκ δίσκου.
- Ένα από τα πεδία του αρχείου καθορίζεται να είναι το **κλειδί κατακερματισμού** του αρχείου.
- Η εγγραφή με κλειδί κατακερματισμού K αποθηκεύεται στον κάδο i, όπου i=h(K), και h είναι η **συνάρτηση κατακερματισμού**.

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

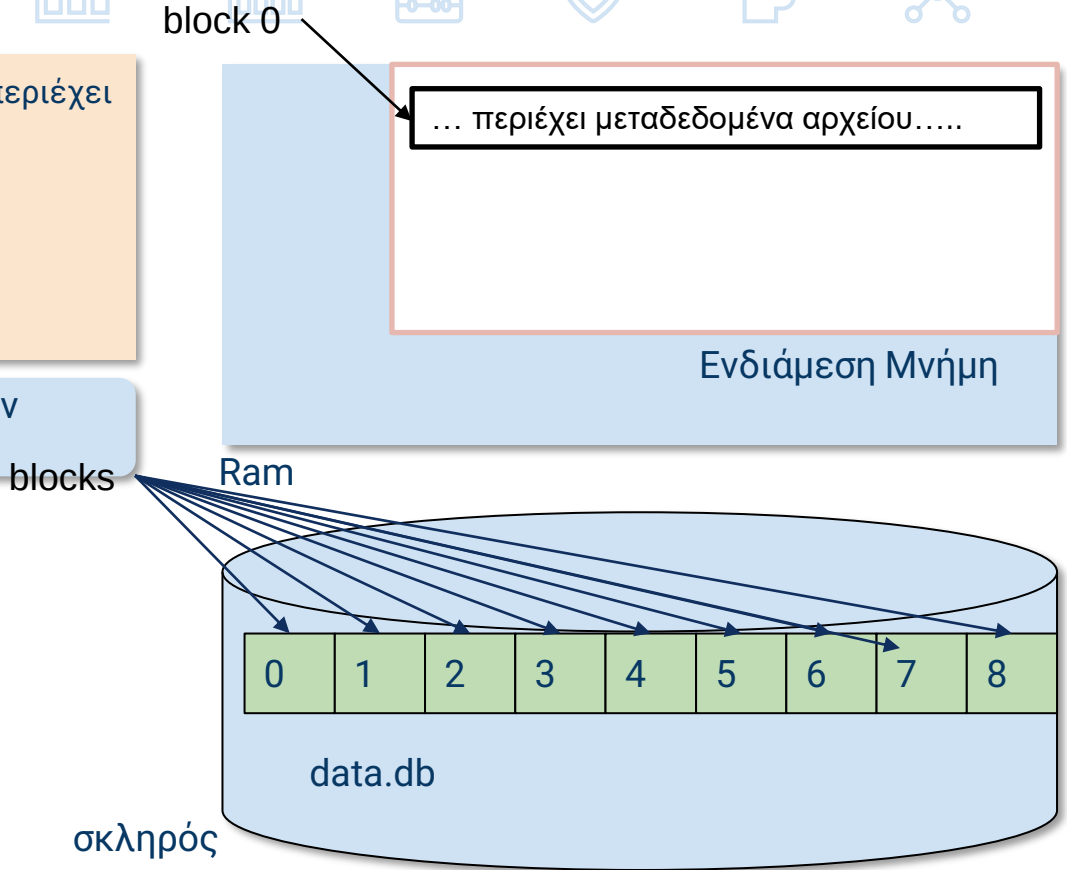


Εισαγωγή Εγγραφών

- 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



Εισαγωγή Εγγραφών

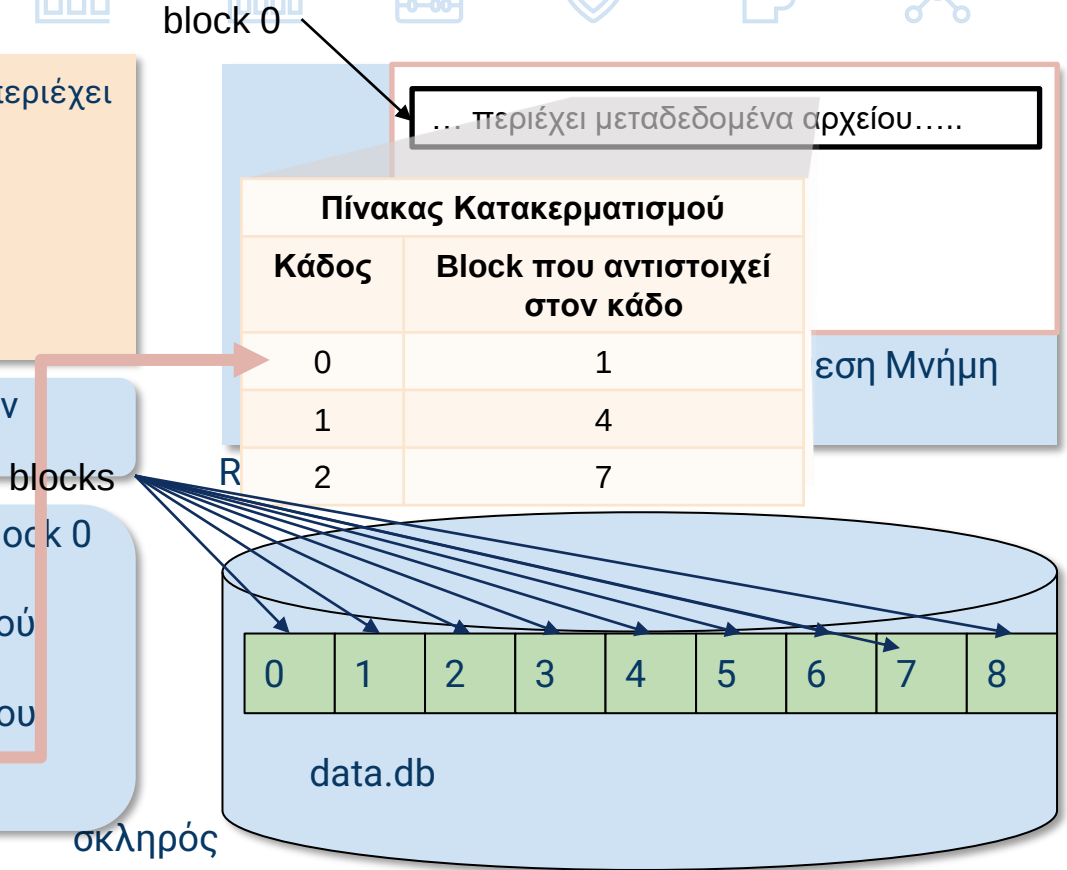
- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής
  (15, "Marianna", "Rezkalla", "Hong Kong")

Μέγεθος εγγραφής: 76 bytes

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

Βήμα 2: Διαβάζω από τα μεταδεδομένα στο block 0 (HT_Info) τον πίνακα κατακερματισμού και

- Εφαρμόζω την συνάρτηση κατακερματισμού στο πεδίο id. Π.χ. hash(15) = 15 % 3= 0
- Έχω εισαγωγή στον **κάδο 0** τα δεδομένα του οποίου βρίσκονται στο **block 1**

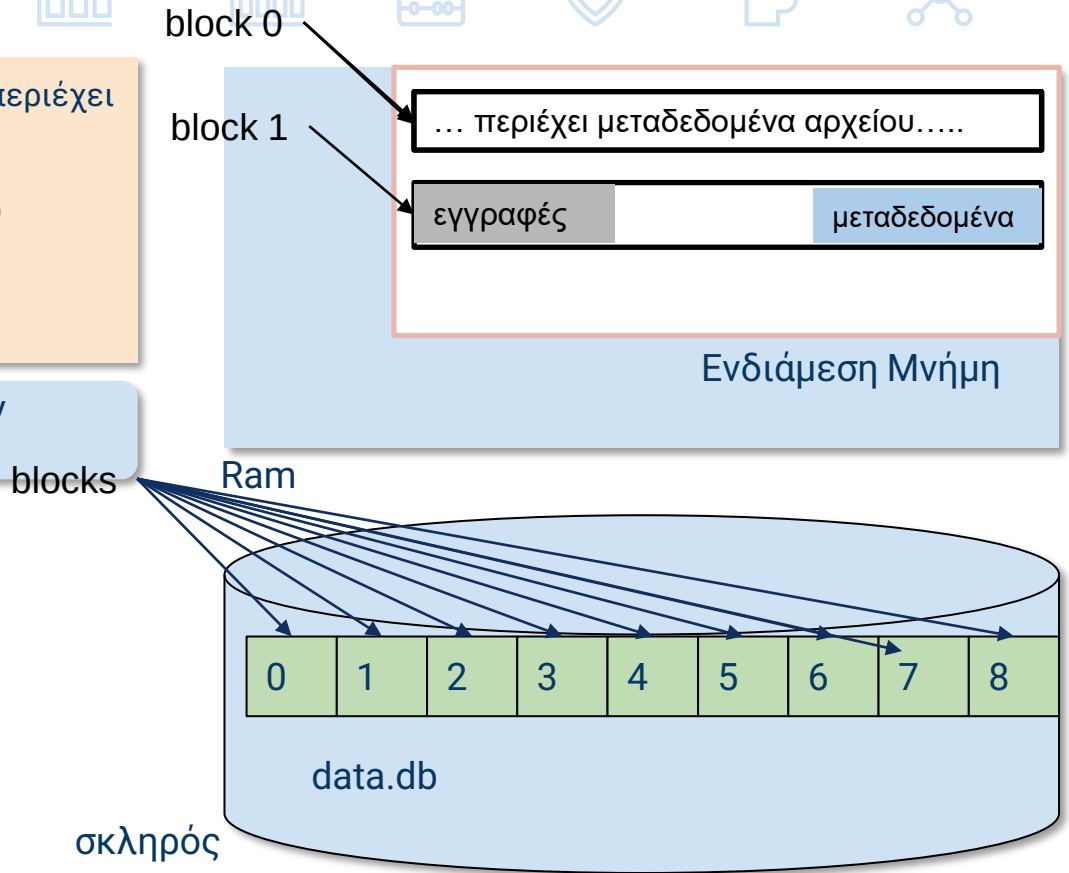


Εισαγωγή Εγγραφών

- 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 3: Φορτώνω το **block 1** του αρχείου στην ενδιάμεση μνήμη



Εισαγωγή Εγγραφών

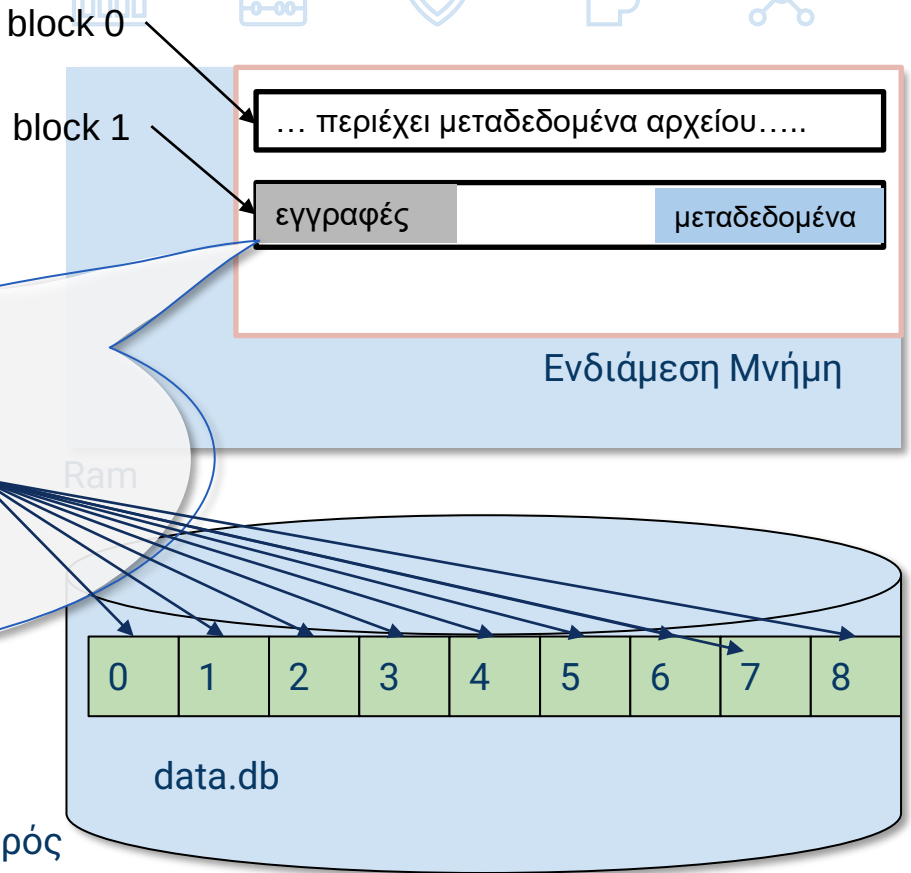
- Έστω ότι έχω το αρχείο σωρού data.db το οποίο περιέχει ήδη κάποιες εγγραφές.
- Θέλω να κάνω την εισαγωγή της εγγραφής
  (15, “Marianna”, “Rezkalla”, “Hong Kong”)

Μέγεθος εγγραφής: 76 bytes

Βήμα 3: Φορτώνω το block 1 του αρχείου στην ενδιάμεση μνήμη

Το συγκεκριμένο block χωρίζεται σε 3 περιοχές:

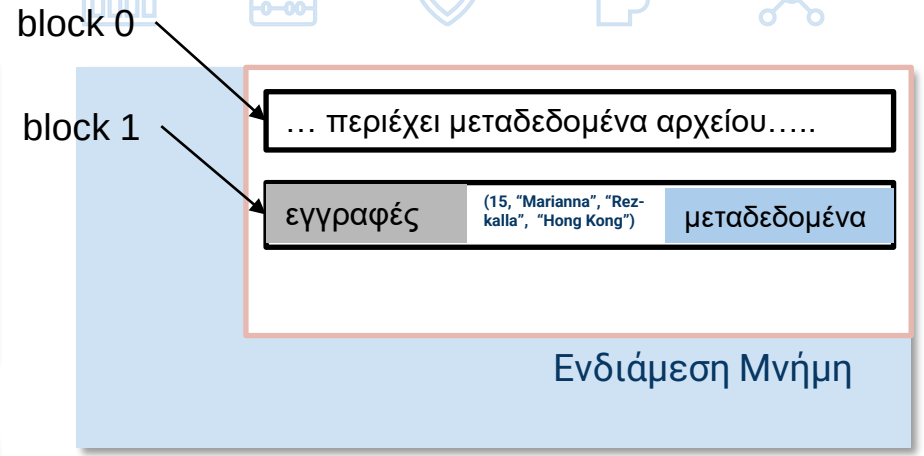
- μία περιοχή που περιέχει **εγγραφές**,
- μία περιοχή **κενή**, και
- μία περιοχή που περιέχει **μεταδεδομένα** για το block



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”)

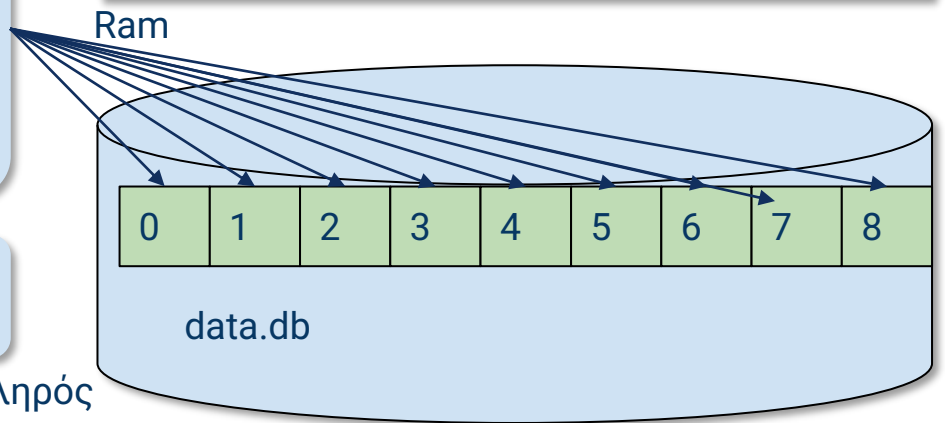
Μέγεθος εγγραφής: 76 bytes



Βήμα 4α: Εφόσον χωράει κάνω **εισαγωγή της εγγραφής στην σελίδα της ενδιάμεσης μνήμης** που περιέχει το block

- Επίσης κάνω το block 1 dirty και unpin εφόσον έχω ολοκληρώσει τις πράξεις μου σε αυτό

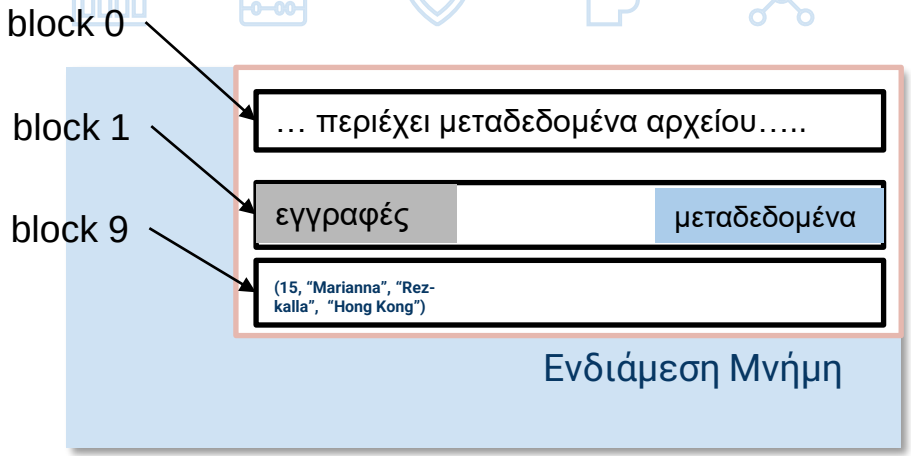
Βήμα 5: Ανανεώνω αντίστοιχα τα μεταδεδομένα μου



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”)

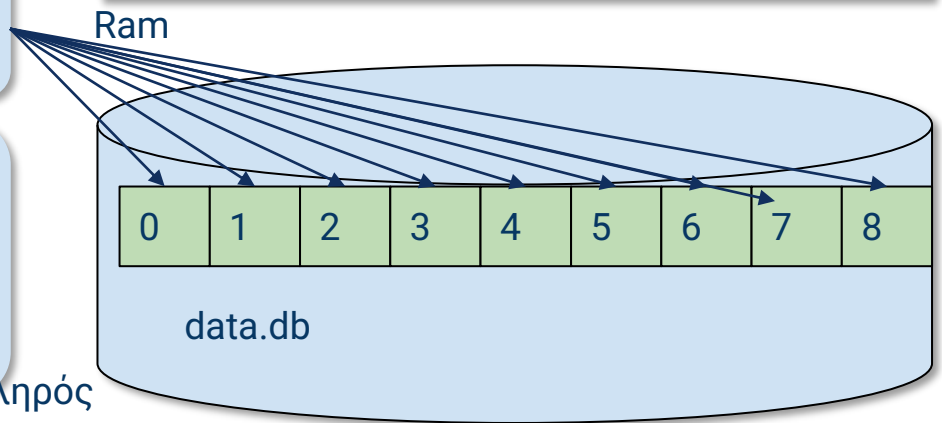
Μέγεθος εγγραφής: 76 bytes



Βήμα 4β: Αν δεν χωράει κάνω allocate καινούριο block (block 9) το φορτώνω στην ενδιάμεση μνήμη και προχωράω αναλόγως την εισαγωγή μου

Βήμα 5: Ανανεώνω τα μεταδεδομένα μου.

- Στα μεταδεδομένα αρχείου (HT_info) ο κάδος 0 πρέπει να απεικονίζεται στο block 9
- Στα μεταδεδομένα block (HT_block_info) ο κάδος 9 θα πρέπει να δείχνει τον κάδο 1 που περιέχει επίσης δεδομένα του ίδιου κάδου



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 στατικού κατακερματισμού για την καταχώρηση εγγραφών:
  - Χρησιμοποιώντας μία κατάλληλη δομή μεταδεδομένων

HT_info
Δομή Μεταδεδομένων Αρχείου Κατακερματισμού

- Μια **ενδεικτική δομή** με τις πληροφορίες που πρέπει να κρατάτε δίνεται στη συνέχεια:

```
typedef struct {
    int fileDesc; /* αναγνωριστικός αριθμός ανοίγματος αρχείου από το επίπεδο block */
    long int numBuckets; /* το πλήθος των "κάδων" του αρχείου κατακερματισμού */ } HT_info;
```

- Όπου fileDesc είναι ο αναγνωριστικός αριθμός του ανοίγματος του αρχείου, όπως επιστράφηκε από το επίπεδο διαχείρισης μπλοκ, και numBuckets είναι το πλήθος των κάδων που υπάρχουν στο αρχείο.
- Επίσης θα χρειαστεί να κρατάτε επιπλέον πληροφορίες όπως: το block στο οποίο είναι αποθηκευμένη, ο πίνακας κατακερματισμού, η χωρητικότητα ενός block σε εγγραφές, και ό,τι άλλο κρίνετε.

Δευτερεύον Ευρετήριο Κατακερματισμού

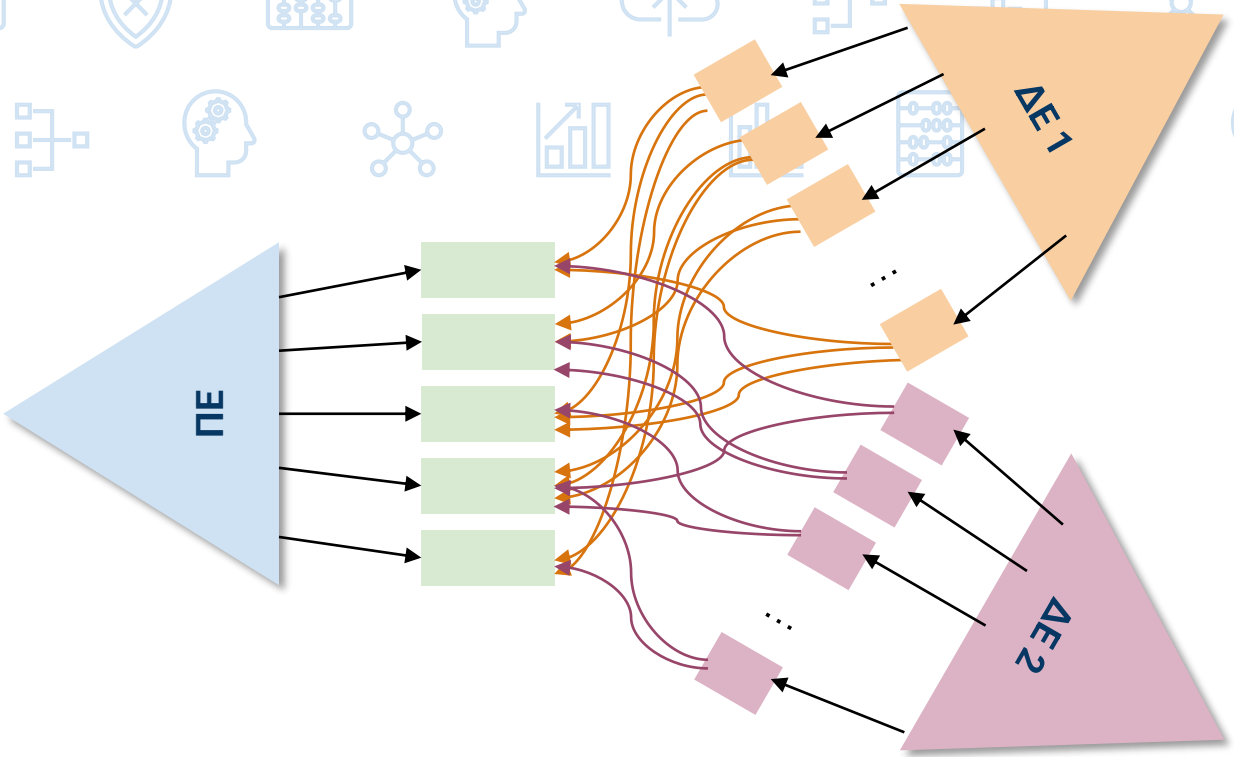
Δευτερεύοντα Ευρετήρια

- **Πρωτεύον ευρετήριο:** καθορίζει το μπλοκ του αρχείου στο οποίο **εισάγεται** μία πλειάδα με βάση το πεδίο κλειδί του ευρετηρίου

  ➔ **Το πολύ ένα**

- **Δευτερεύοντα ευρετήρια:** τα δευτερεύοντα ευρετήρια **υποδεικνύουν** το μπλοκ του αρχείου στο οποίο (πιθανώς να) **υπάρχει** μία πλειάδα

  ➔ **απεριόριστος** αριθμός από δευτερεύοντα ευρετήρια



Πρωτεύον Ευρετήριο

Μπλοκ Δεδομένων

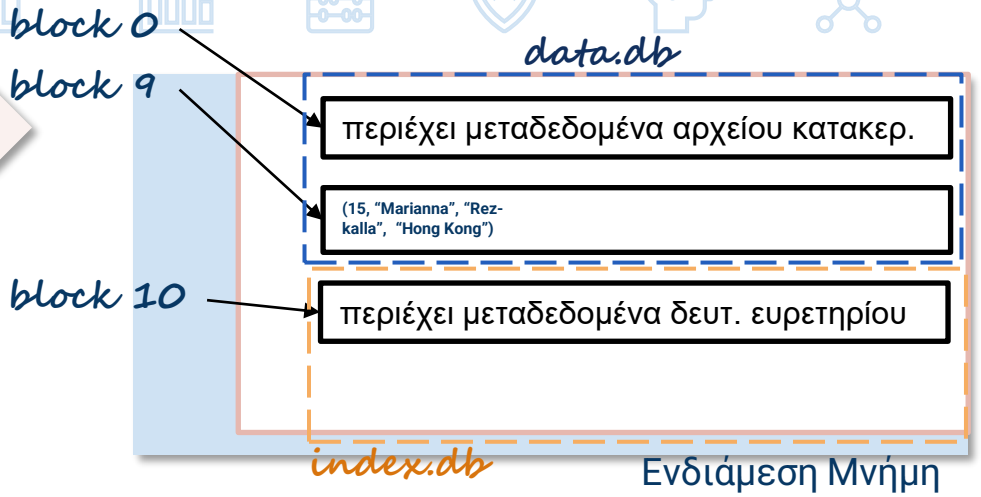
Δευτερεύοντα Ευρετήρια

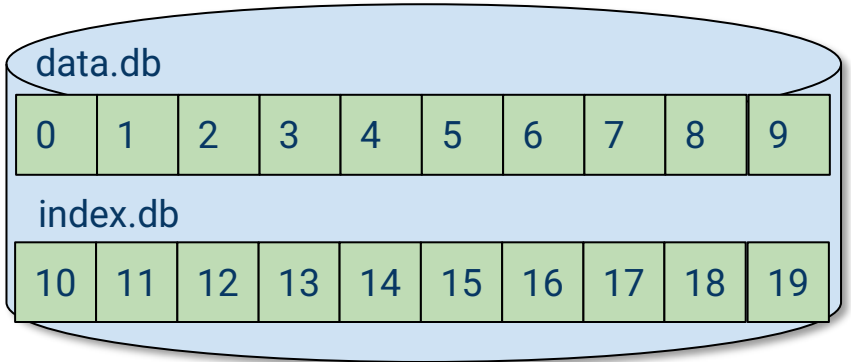
Δομή Δευτερεύοντος Ευρετηρίου

- Η δομή ενός δευτερεύοντος ευρετηρίου καταλήγει να δείχνει σε ένα **ενδιάμεσο επίπεδο μπλοκ**
  - Το **ενδιάμεσο επίπεδο** περιέχει ζεύγη **(τιμή πεδίου-κλειδιού, δείκτης εγγραφής),** ένα για κάθε πλειάδα του αρχείου
  - Ο δείκτης δείχνει σε πλειάδα όπου η τιμή στο πεδίο κλειδί είναι η τιμή του πεδίου κλειδί στο ζευγάρι
  - Κάθε πλειάδα σε μπλοκ του αρχείου δεδομένων, έχει ένα ζευγάρι **(τιμή πεδίου-κλειδιού, δείκτης εγγραφής)** στο τελικό επίπεδο που την δείχνει **για κάθε Δευτερεύον Ευρετήριο**
    - **Όσες πλειάδες υπάρχουν στα δεδομένα, τόσα ζευγάρια υπάρχουν στο ενδιάμεσο επίπεδο κάθε ευρετηρίου**

Εισαγωγή Σε Δευτερεύον Ευρετήριο

Δευτερεύον Ευρετήριο: Θα πρέπει να κάνω την αντίστοιχη εισαγωγή (Marianna, $block_9$) στο δευτερεύον ευρετήριο (με βάση το όνομα)

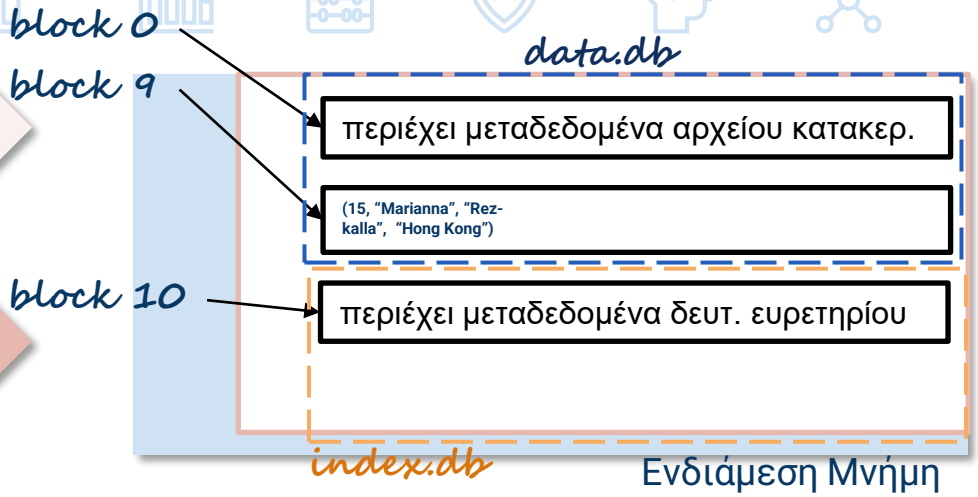


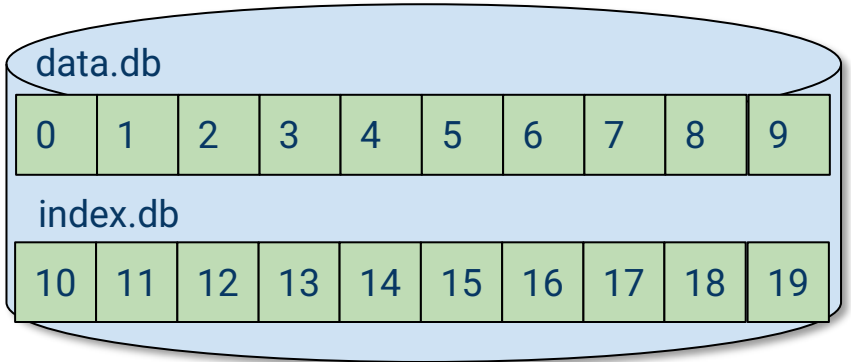


Εισαγωγή Σε Δευτερεύον Ευρετήριο

Δευτερεύον Ευρετήριο: Θα πρέπει να κάνω την αντίστοιχη εισαγωγή (Marianna, $block_9$) στο δευτερεύον ευρετήριο (με βάση το όνομα)

Η εισαγωγή γίνεται ως προς το όνομα. Αν hash(Marianna) = 2, τότε η εγγραφή μου πρέπει να μπει στο **block για το bucket 2**



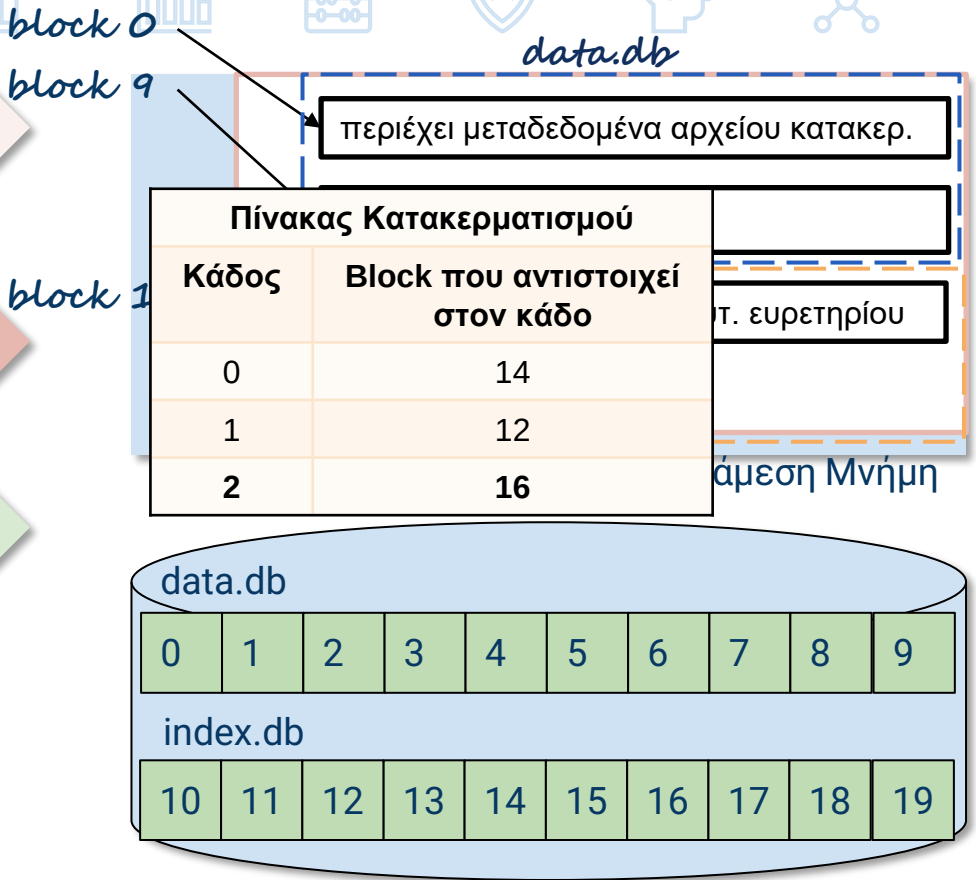


Εισαγωγή Σε Δευτερεύον Ευρετήριο

Δευτερεύον Ευρετήριο: Θα πρέπει να κάνω την αντίστοιχη εισαγωγή (Marianna, $block_9$) στο δευτερεύον ευρετήριο (με βάση το όνομα)

Η εισαγωγή γίνεται ως προς το όνομα. Αν hash(Marianna) = 2, τότε η εγγραφή μου πρέπει να μπει στο **block για το bucket 2**

Από τα μεταδεδομένα του ευρετηρίου βρίσκω ότι αυτό είναι το **$block_{16}$**



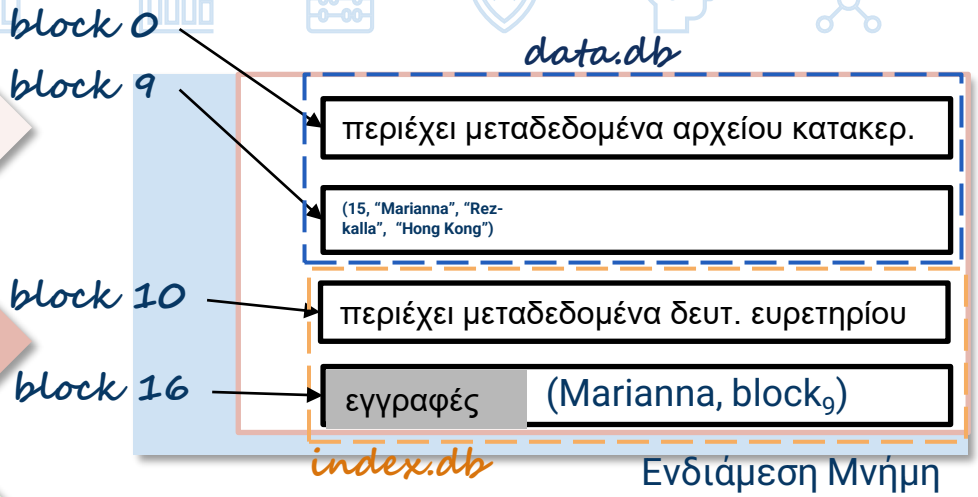
Εισαγωγή Σε Δευτερεύον Ευρετήριο

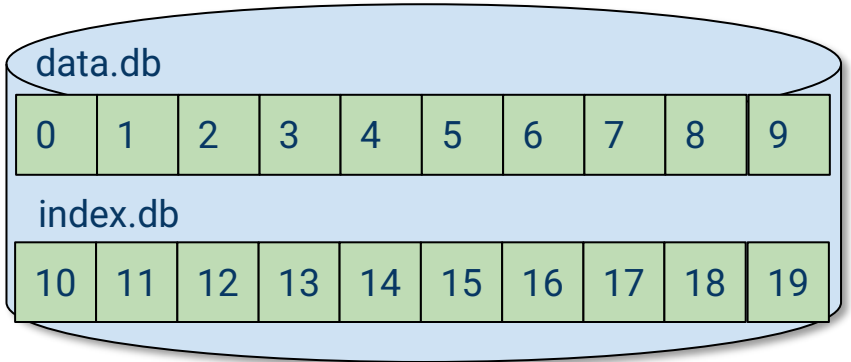
Δευτερεύον Ευρετήριο: Θα πρέπει να κάνω την αντίστοιχη εισαγωγή (Marianna, $block_9$) στο δευτερεύον ευρετήριο (με βάση το όνομα)

Η εισαγωγή γίνεται ως προς το όνομα. Αν hash(Marianna) = 2, τότε η εγγραφή μου πρέπει να μπει στο **block για το bucket 2**

Από τα μεταδεδομένα του ευρετηρίου βρίσκω ότι αυτό είναι το $block_{16}$

Φορτώνω το $block_{16}$ στην ενδιάμεση μνήμη και κάνω την εισαγωγή μου





Δομή Μεταδεδομένων

Μια ενδεικτική δομή με τις πληροφορίες που πρέπει να κρατάτε δίνεται στη συνέχεια:

```
typedef struct {
    int fileDesc; /* αναγνωριστικός αριθμός ανοίγματος αρχείου από το επίπεδο block */
    int numBuckets; /* το πλήθος των "κάδων" του αρχείου κατακερματισμού */} SHT_info;
```

Το πεδίο fileDesc είναι ο αναγνωριστικός αριθμός του ανοίγματος του αρχείου, όπως επιστράφηκε από το επίπεδο διαχείρισης μπλοκ, numBuckets είναι το πλήθος των κάδων που υπάρχουν στο αρχείο. Αφού ενημερωθεί κατάλληλα η δομή πληροφοριών του αρχείου, την επιστρέφετε.