Ενσωματωμένα Συστήματα

Ενότητα 5: Μεθοδολογία Σχεδίασης Ιεραρχίας Μνήμης και Επαναχρησιμοποίησης Δεδομένων.

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.icte.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

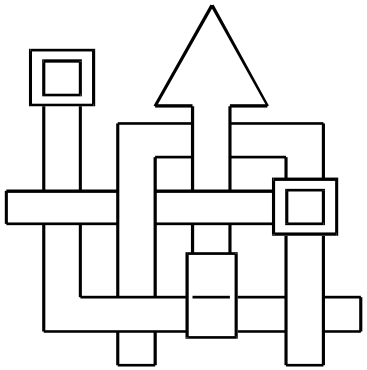
- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοπός Ενότητας

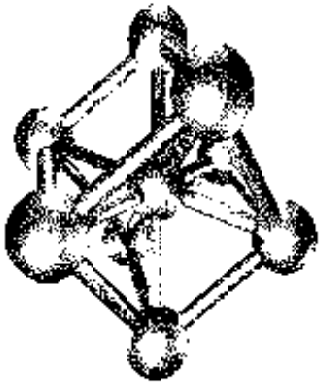
- Η κατανόηση του προβλήματος της αργής και ενεργοβόρας εξωτερικής μνήμης.
- Η παρουσίαση της διαδικασίας σχεδιασμού προσαρμοσμένης ιεραρχίας μνήμης, ελεγχόμενης από το λογισμικό, με σκοπό την ελαχιστοποίηση το ενεργειακού κόστους.

Ιεραρχία Μνήμης & Εξερεύνηση Αποφάσεων Επαναχρησιμοποίησης Δεδομένων

Υλικό από παρουσίαση του: Pol Marchal
IMEC, Leuven, Belgium



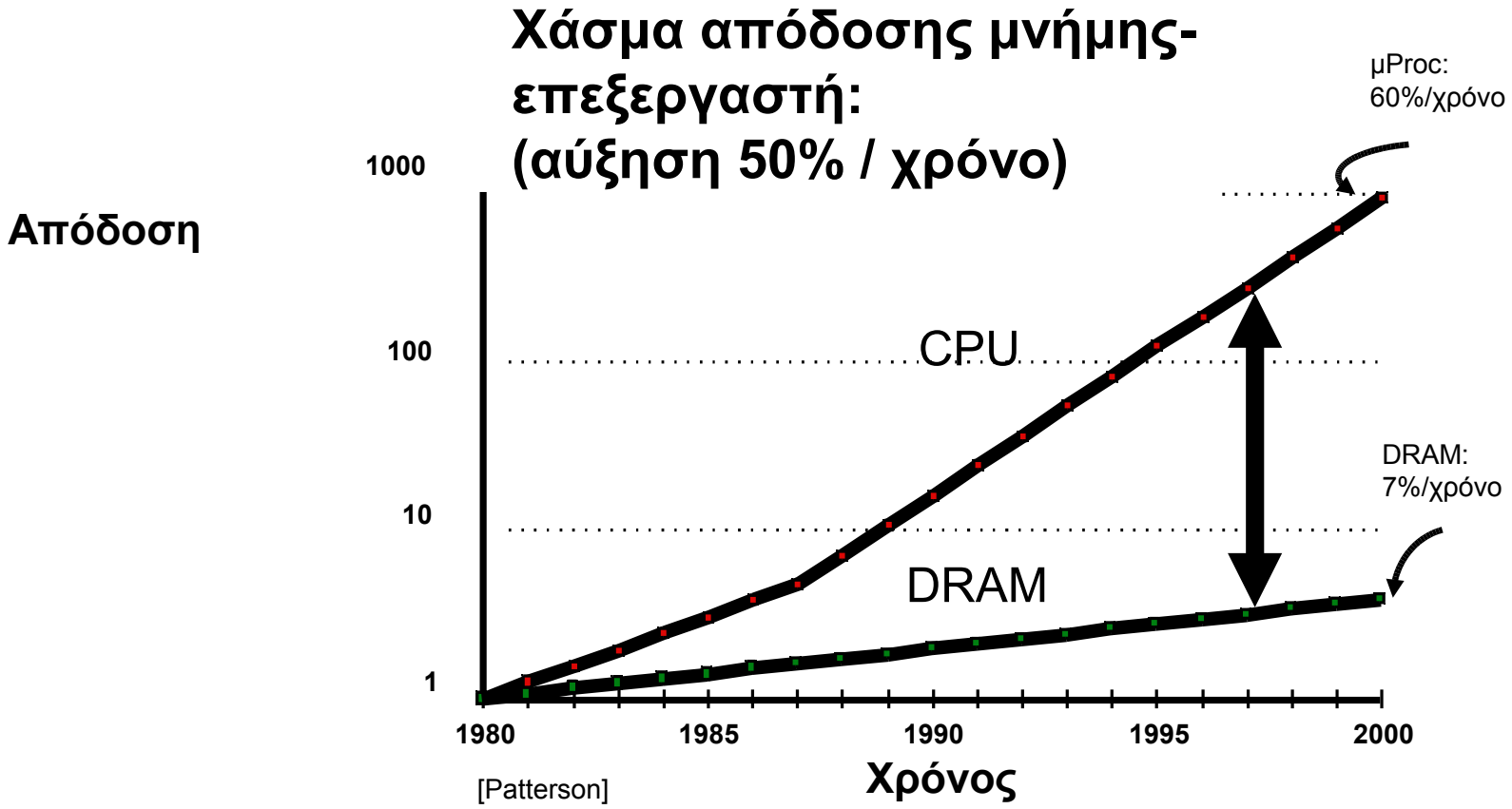
IMEC



ATOMIUM

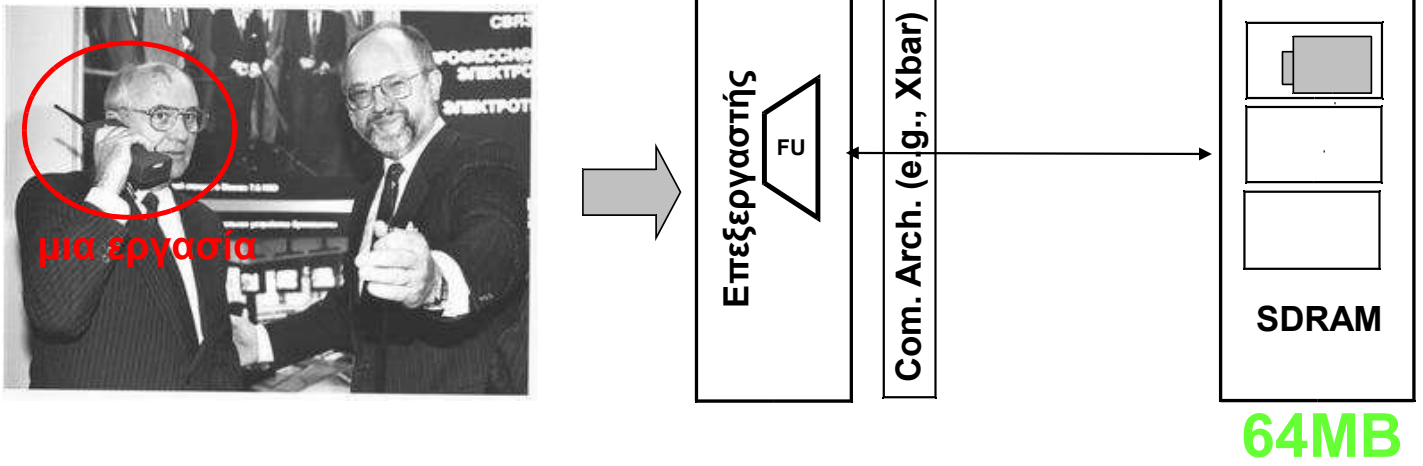
ACROPOLIS

Το χάσμα απόδοσης μνήμης/επεξεργαστή



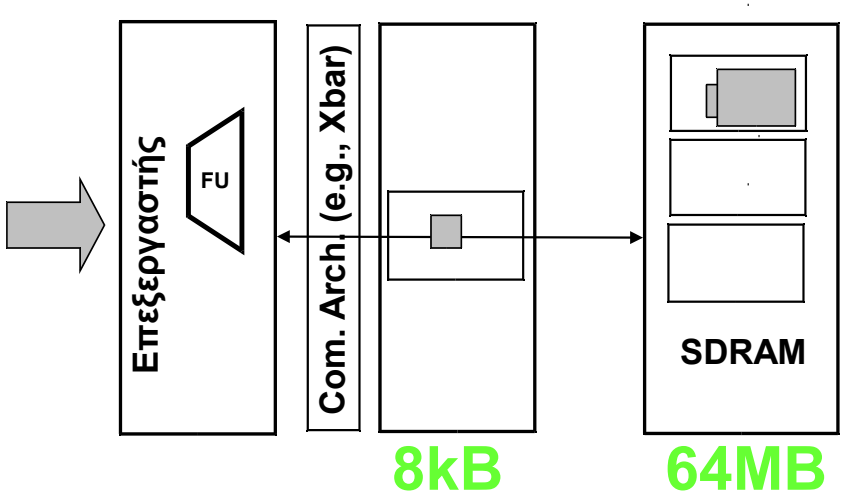
Ιεραρχίας μνήμης (1/2)

- Παρουσίαση μιας ιεραρχίας μνήμης για ελαχιστοποίηση του χάσματος απόδοσης μεταξύ μνήμης και στοιχείων προς επεξεργασία

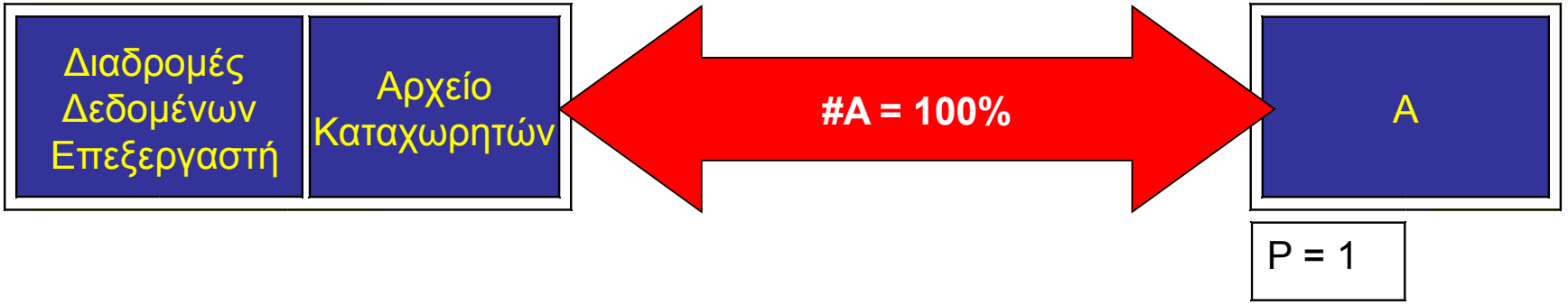


Ιεραρχίας μνήμης (2/2)

Εισαγωγή μικρότερης μνήμης, ώστε να ρυθμίζονται συχνότερα τα δεδομένα που προσπελάστηκαν [Hen96][Benini01][Wolf03][Panda01]

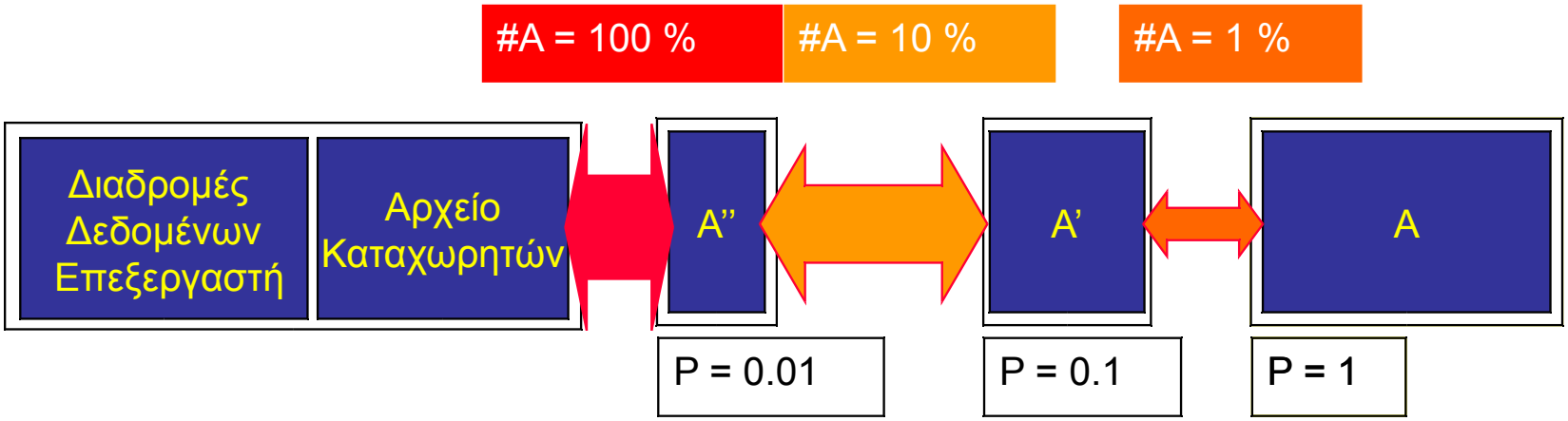


Μια ιεραρχία μνήμης δύναται να εξοικονομεί ενέργεια (1/2)



P total (πριν) = 100%

Μια ιεραρχία μνήμης δύναται να εξοικονομεί ενέργεια (2/2)



P total (πριν) = 100%

P total (μετά) = 100%x0.01+10%x0.1+1%x1
= 3%

Διάφορες προσεγγίσεις σχεδιασμού ιεραρχίας μνήμης

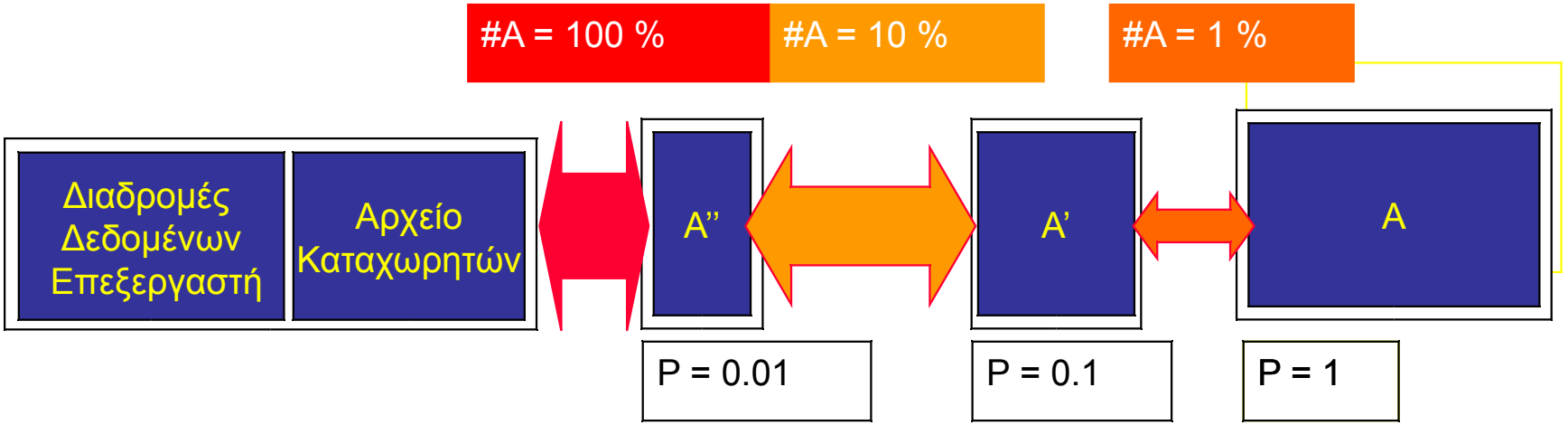
- Καθορισμένη ή προσαρμοσμένη ιεραρχία μνήμης.
  - Καθορισμένη: η ιεραρχία μνήμης καθορίζεται εξ' αρχής. Για παράδειγμα, στον P4 της Intel, η ιεραρχία μνήμης είναι καθορισμένη.
  - Προσαρμοσμένη: ο σχεδιαστής μπορεί ελεύθερα να σχεδιάσει την ιεραρχία μνήμης, κατά το “χτίσιμο” ενός ASIC.
- Πως γίνεται η **προμετάκληση** δεδομένων;
  - Κρυφή μνήμη ελεγχόμενη από το υλικό.
    - ✓ Μια απλή FSM μεταφέρει δεδομένα, όποτε αυτά δεν είναι διαθέσιμα στην τοπική μνήμη.
    - ✓ Η FSM προκαλεί ενεργειακή επιβάρυνση.
    - ✓ Εύκολη στη χρήση.
  - Μνήμες scratchpad ελεγχόμενες από το λογισμικό.
    - ✓ Το prefetching(προμετάκληση) πρέπει να προγραμματιστεί στο λογισμικό.
    - ✓ Ελαχιστοποίηση του ενεργού υλικού, έτσι ώστε εξ' αρχής να υπάρχει περισσότερη αποτελεσματική ισχύς.

Η προσαρμογή της ιεραρχίας μνήμης συνεπάγεται δύο προβλήματα: 1ο πρόβλημα

Κώδικας εφαρμογής:

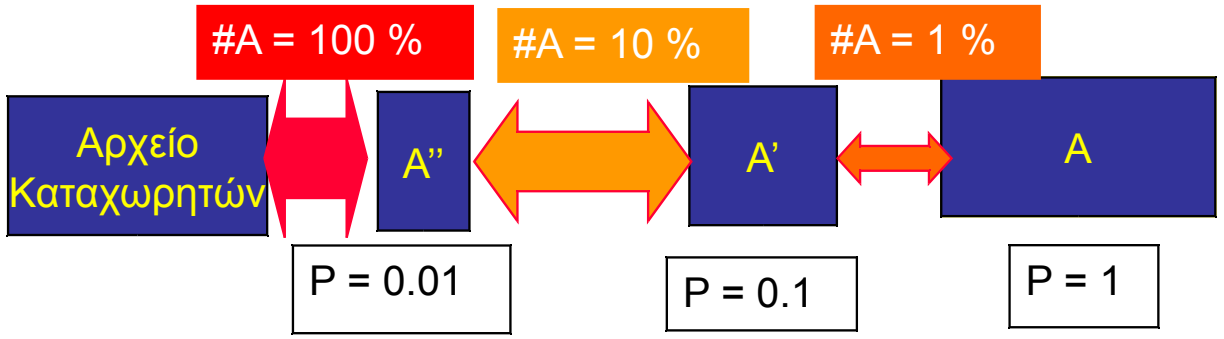
```
for (i=0; i<4; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = A[i*4+k];
```

1. Πως θα διακρίνουμε το Α'' και το Α' από τον κώδικα της εφαρμογής? (Πως θα διακρίνουμε τα συχνότερα μέρη/δομές δεδομένων εντός του κώδικα)

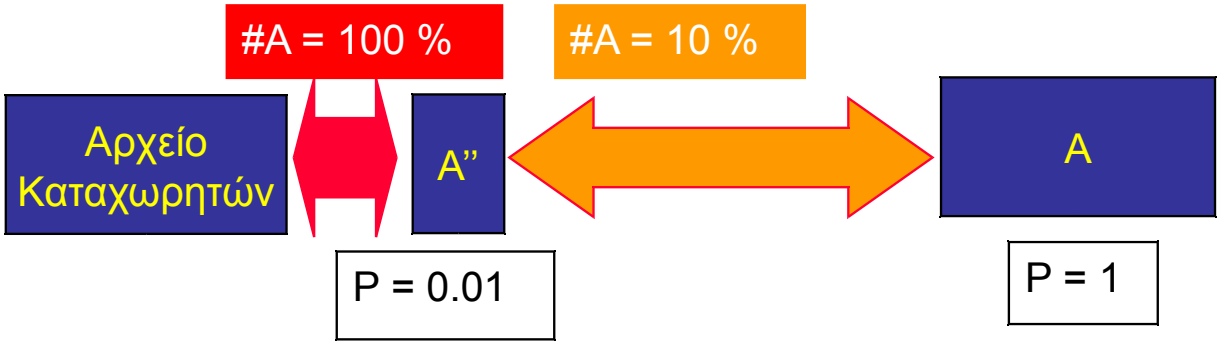


Η προσαρμογή της ιεραρχίας μνήμης συνεπάγεται δύο προβλήματα: 2ο πρόβλημα

2. Πόσα επίπεδα να χρησιμοποιηθούν, ανταλλάσσοντας επιφάνεια έναντι ισχύς/απόδοση;



P=0.01*100+0.1*10+1*1= 3% Επιφάνεια=A''+A'+A

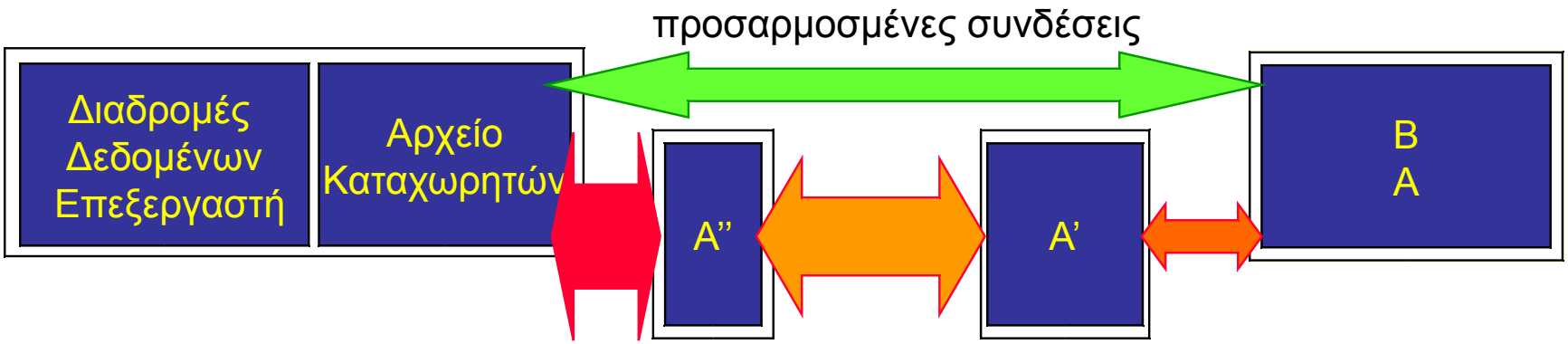


P=0.01*100+1*10+= 20% Επιφάνεια=A+A''

Ερμηνεία μιας προσαρμοσμένης ιεραρχίας μνήμης μέσω ενός απλοποιημένου “script”

1. **Αναγνώρισε τα σήματα με επαρκές ενδεχόμενο επαναχρησιμοποίησης.**
2. Προσδιόρισε τις αλυσίδες επαναχρησιμοποίησης και απομάκρυνε αυτές.
   (για κάθε ανάγνωση “σειράς”)
3. Προσδιόρισε τα δέντρα επαναχρησιμοποίησης και απομάκρυνε αυτά.
   (για κάθε σειρά)
4. Προσδιόρισε το γράφημα επαναχρησιμοποίησης, συμπεριλαμβάνοντας τις παρακάμψεις και απομάκρυνε αυτό.
   (για όλη την εφαρμογή)
5. Προσδιόρισε την εκχώρηση διάταξης για την ιεραρχία μνήμης, ενσωματώνοντας τους δεδομένους περιορισμούς μνήμης παρασκηνίου *(επιπέδων),* καθώς επίσης και περιορισμούς πραγματικού χρόνου.

Πρότυπο αρχιτεκτονικής προσαρμοσμένης μνήμης

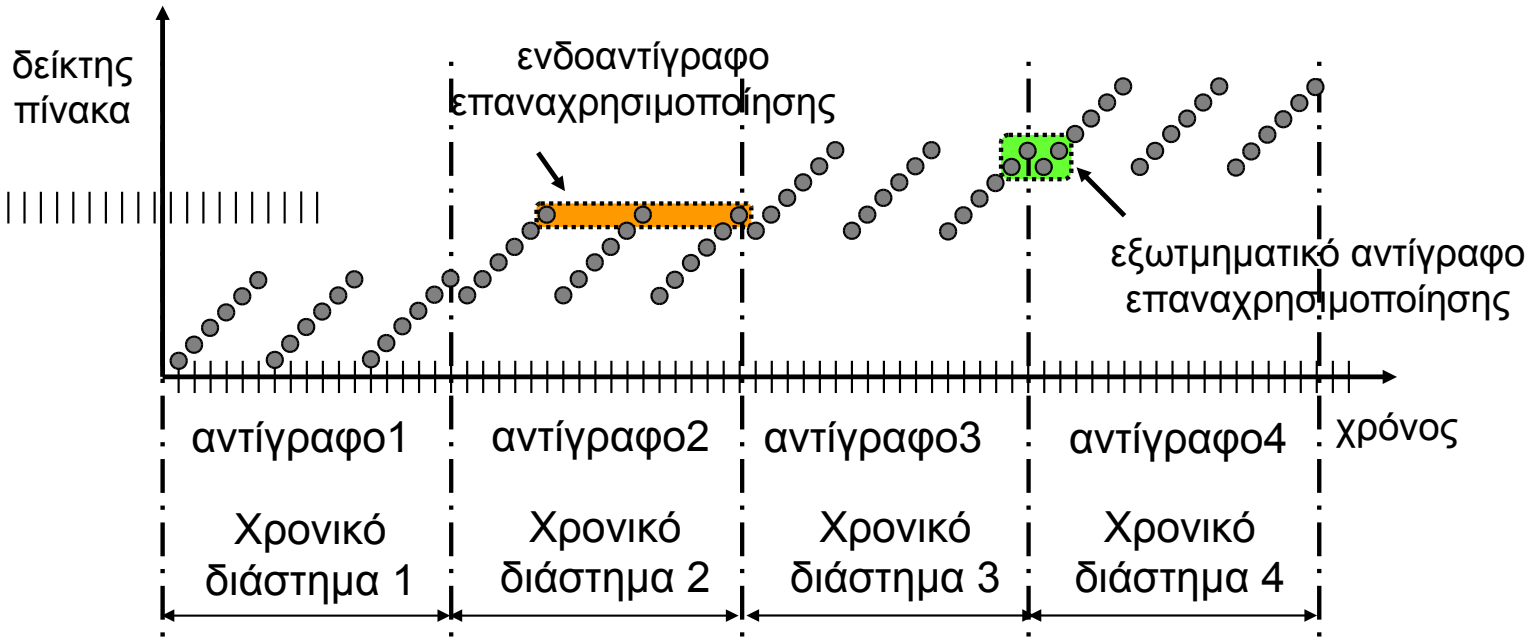


Προσαρμοσμένες συνδέσεις στο υποσύστημα μνήμης, ώστε να παρακαμφθεί η ιεραρχία μνήμης και να αποτραπεί η επιβάρυνση.

Βήμα 1

Προσδιόρισε τις σειρές με ενδεχόμενο επαναχρησιμοποίησης δεδομένων. Ανέλυσε κάθε πρόσβαση ανάγνωσης για επαναχρησιμοποίηση.

```
for (i=0; i<4; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = A[i*4+k];
```



Ορισμός χρονικού διαστήματος

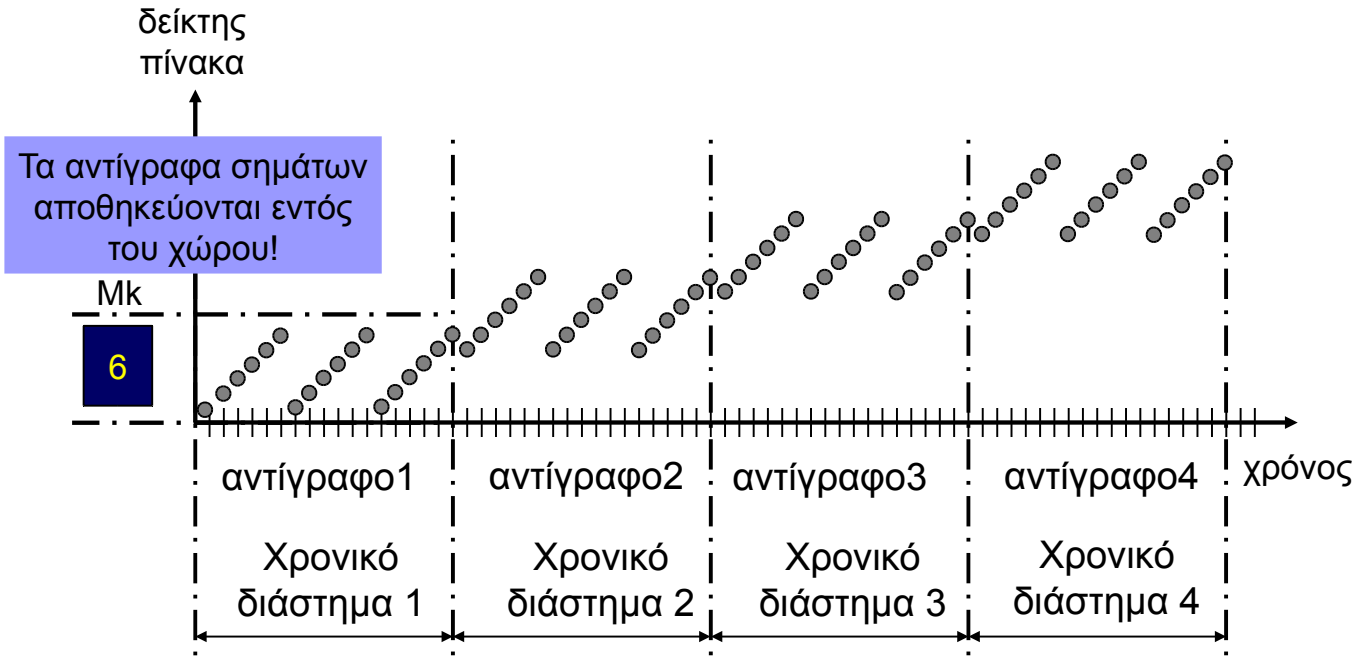
- **Χρονικό διάστημα =** περίοδος χρόνου, κατά την οποία τα επαναχρησιμοποιήσιμα δεδομένα μιας δεδομένης σειράς, αντιγράφονται σε ένα προσωρινό αντίγραφο και διαβάζονται από αυτό, αντί να διαβαστούν από την πρωτότυπη σειρά.

- **Ευρετικός αλγόριθμος:**
  χρονικό διάστημα = όρια του βρόχου επαναλήψεων.

- Όλα τα αντίγραφα που ανήκουν σε δεδομένο επίπεδο χρονικού διαστήματος αποθηκεύονται στο χώρο.

Σπουδαιότητα των HL εντός του χώρου (in place)

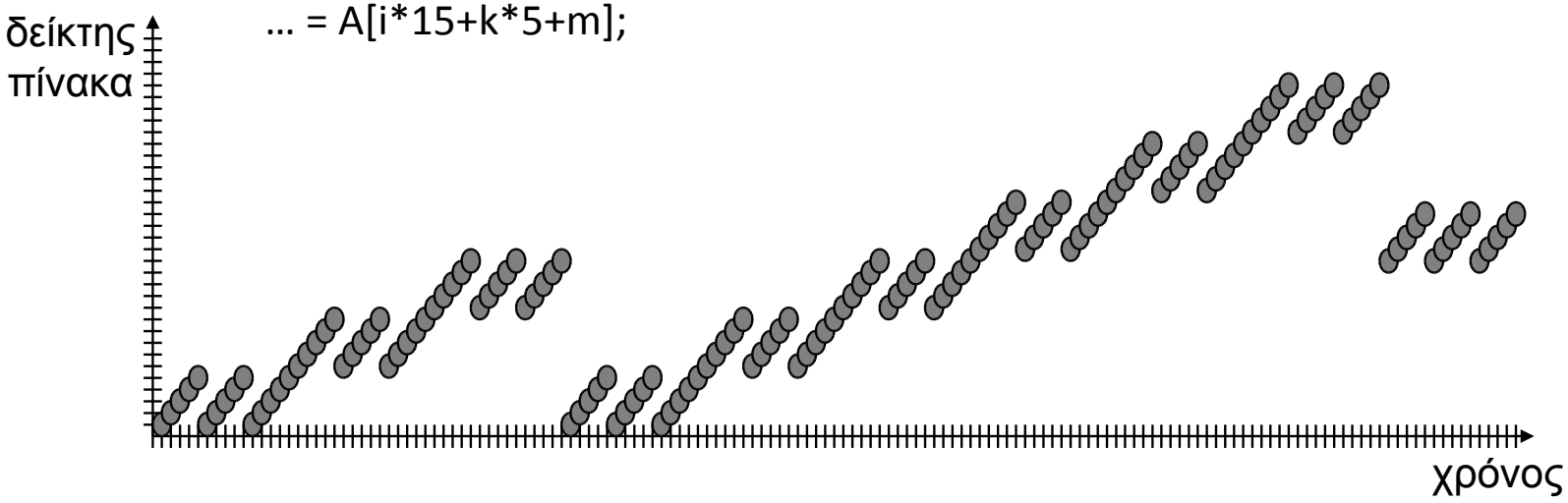
```
for (i=0; i<4; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = A[i*4+k];
```



Επαναχρησιμοποιήσιμα αντίγραφα (1/3)

Διαφορετικά χρονικά διαστήματα οδηγούν σε διαφορετικά επαναχρησιμοποιήσιμα αντίγραφα, τα οποία μπορούν να χρησιμοποιούνται ιεραρχικά.

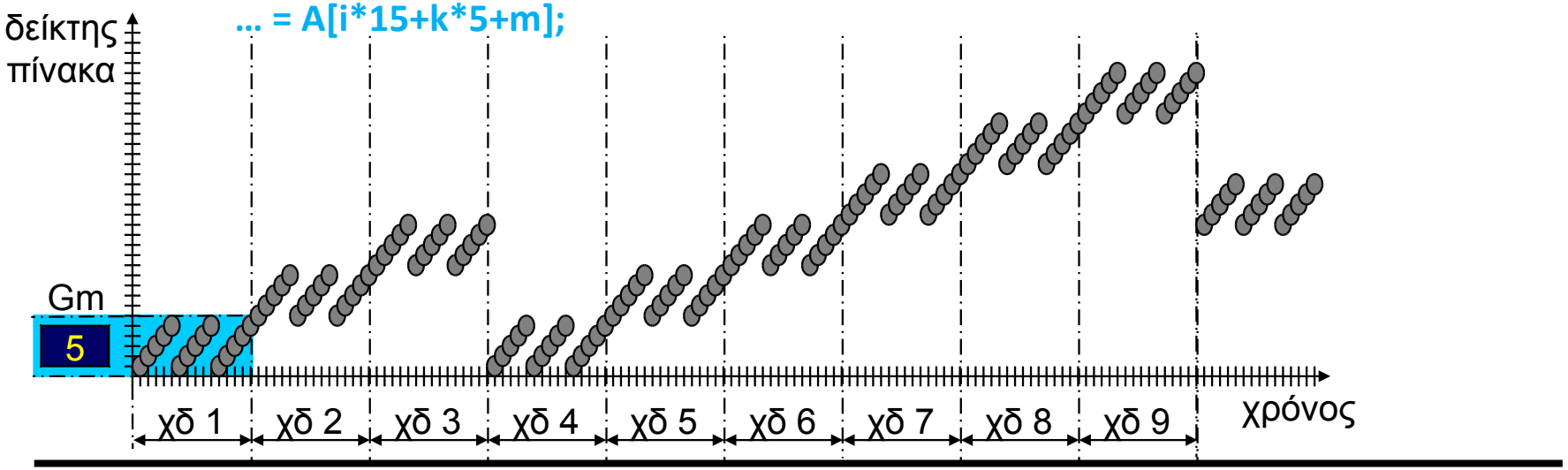
```
for (i=0; i<10; i++)
  for (j=0; j<2; j++)
    for (k=0; k<3; k++)
      for (l=0; l<3; l++)
        for (m=0; m<5; m++)
          … = A[i*15+k*5+m];
```



Επαναχρησιμοποιήσιμα αντίγραφα (2/3)

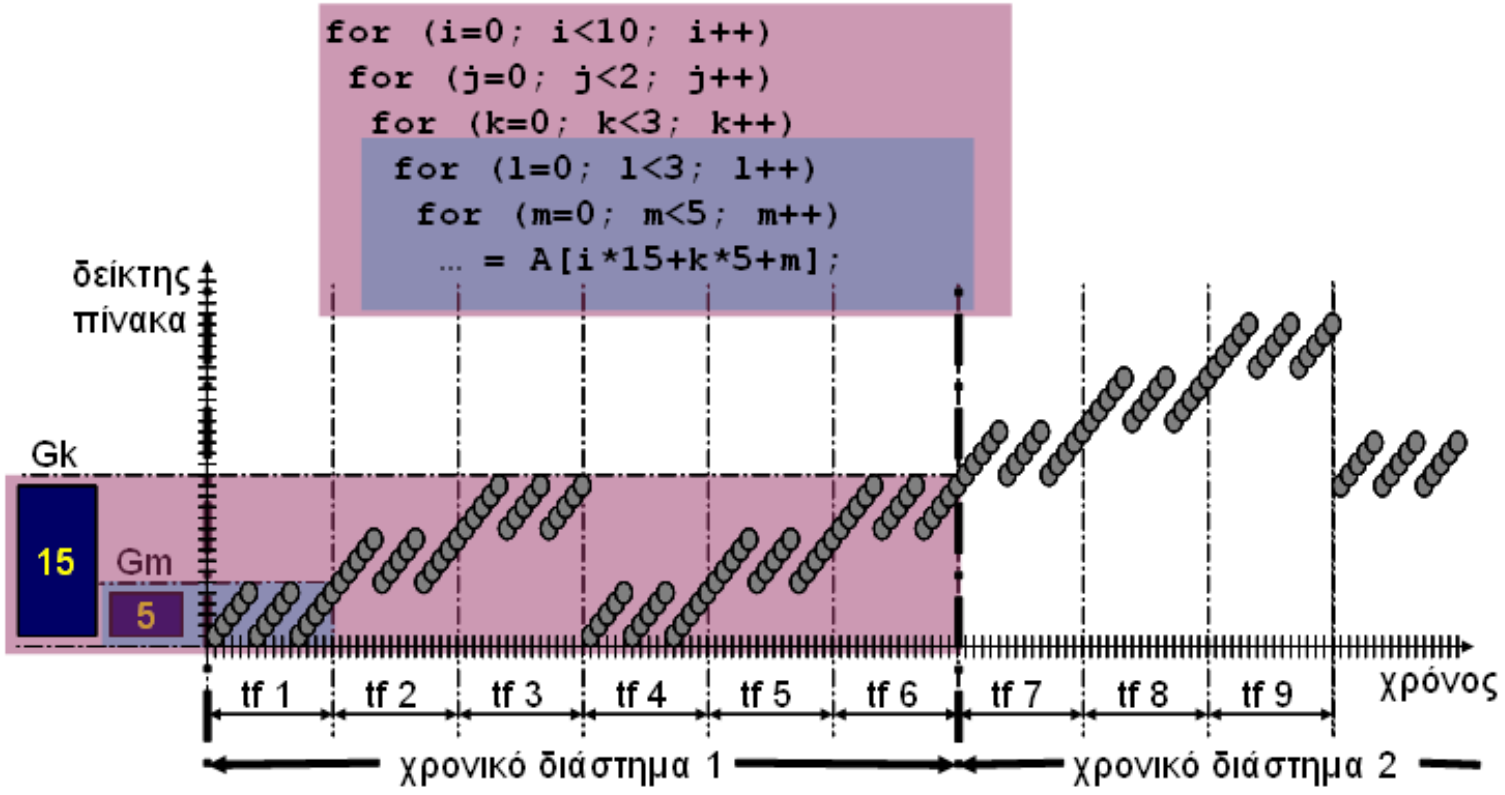
Διαφορετικά χρονικά διαστήματα οδηγούν σε διαφορετικά επαναχρησιμοποιήσιμα αντίγραφα, τα οποία μπορούν να χρησιμοποιούνται ιεραρχικά.

```
for (i=0; i<10; i++)
  for (j=0; j<2; j++)
    for (k=0; k<3; k++)
      for (l=0; l<3; l++)
        for (m=0; m<5; m++)
          … = A[i*15+k*5+m];
```



Επαναχρησιμοποιήσιμα αντίγραφα (3/3)

Διαφορετικά χρονικά διαστήματα οδηγούν σε διαφορετικά επαναχρησιμοποιήσιμα αντίγραφα, τα οποία μπορούν να χρησιμοποιούνται ιεραρχικά.



Ερμηνεία μιας προσαρμοσμένης ιεραρχίας μνήμης μέσω απλοποιημένου “script”

1. Αναγνώρισε τα σήματα με επαρκές ενδεχόμενο επαναχρησιμοποίησης.
2. **Προσδιόρισε τις αλυσίδες επαναχρησιμοποίησης και απομάκρυνε αυτές.**
   (για κάθε ανάγνωση “σειράς”)
3. Προσδιόρισε τα δέντρα επαναχρησιμοποίησης και απομάκρυνε αυτά.
   (για κάθε σειρά)
4. Προσδιόρισε το γράφημα επαναχρησιμοποίησης, συμπεριλαμβάνοντας τις παρακάμψεις και απομάκρυνε αυτό.
   (για όλη την εφαρμογή)

5. Προσδιόρισε την εκχώρηση διάταξης για την ιεραρχία μνήμης, ενσωματώνοντας τους δεδομένους περιορισμούς μνήμης παρασκηνίου *(επιπέδων),* καθώς επίσης και περιορισμούς πραγματικού χρόνου.

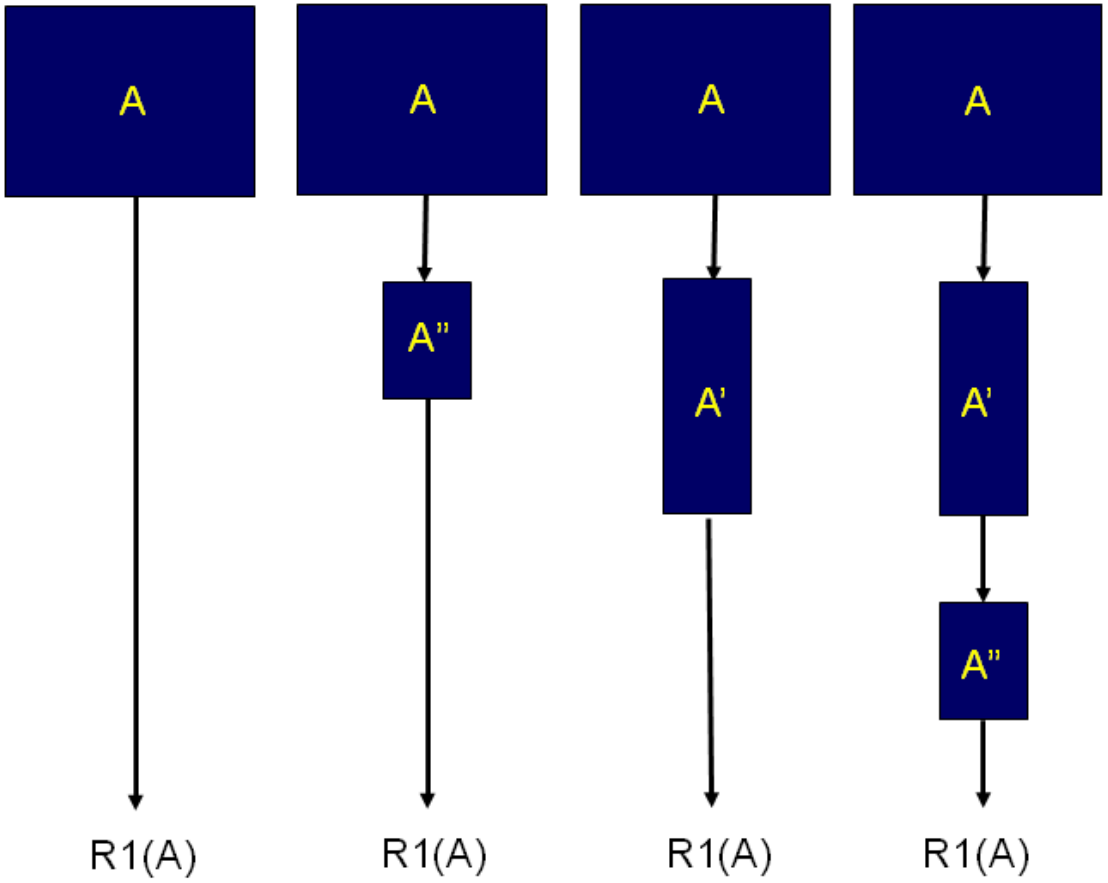
Βήμα 2

Προσδιόρισε τις αλυσίδες επαναχρησιμοποίησης δεδομένων

Πολλές πιθανότητες επαναχρησι-μοποίησης

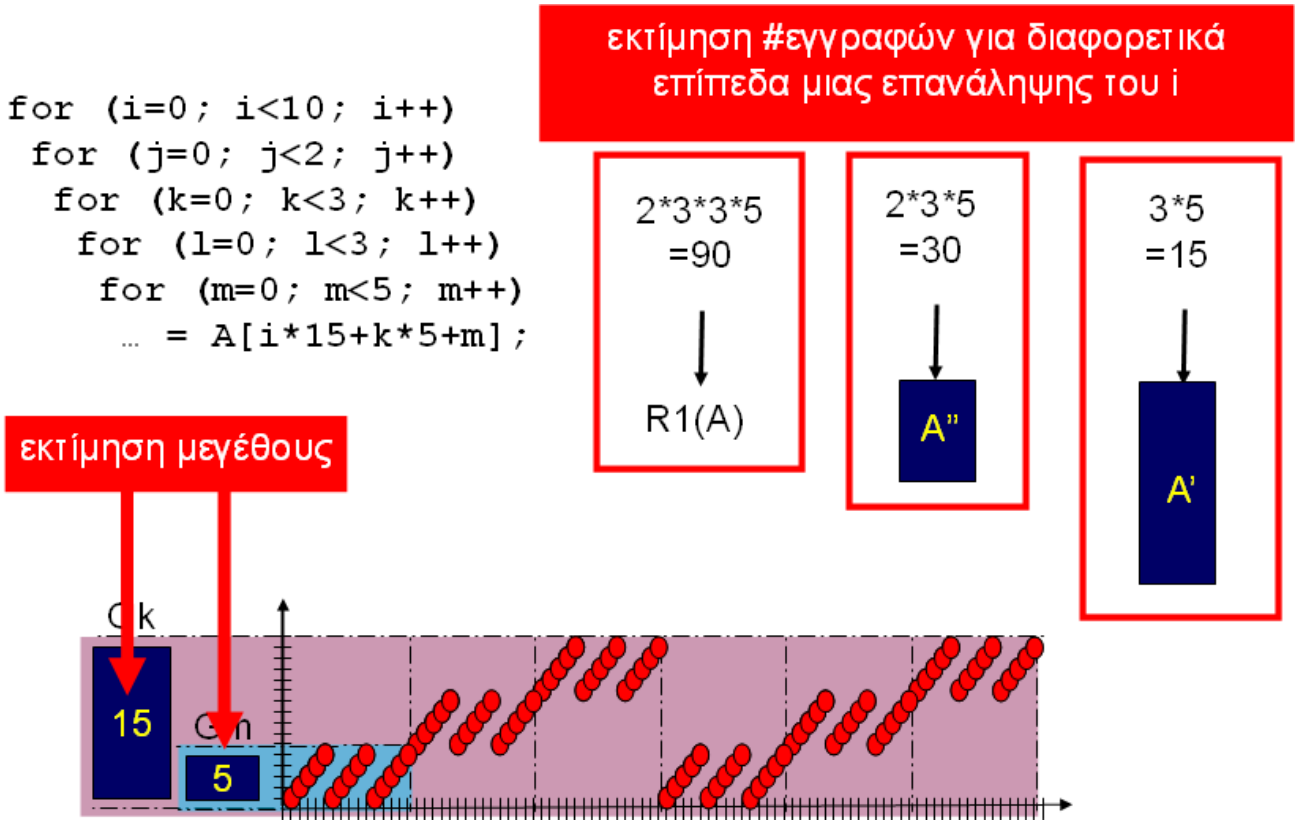
Διατήρηση αυτών με τα μεγαλύτερα πιθανα οφέλη

Απαιτείται εκτίμηση κόστους



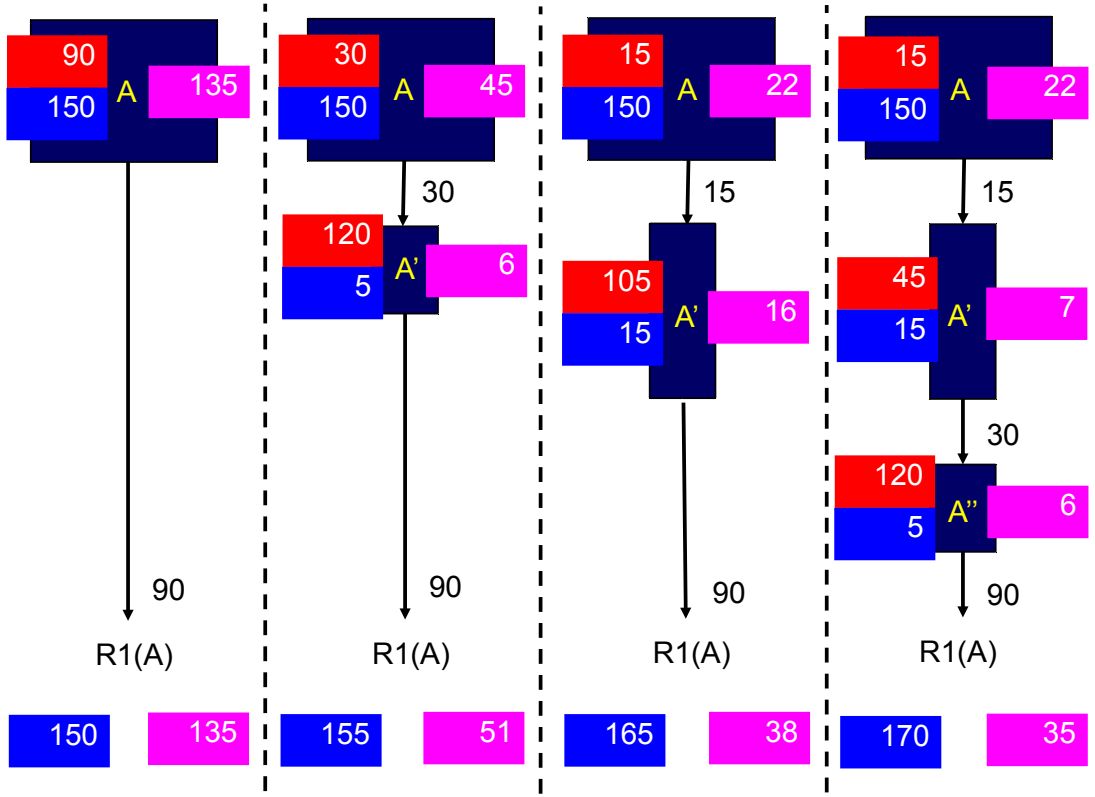
Συνάρτηση κόστους

- Η συνάρτηση κόστους χρειάζεται τόσο το μέγεθος όσο και τον αριθμό προσβάσεων για τους ενδιάμεσους πίνακες



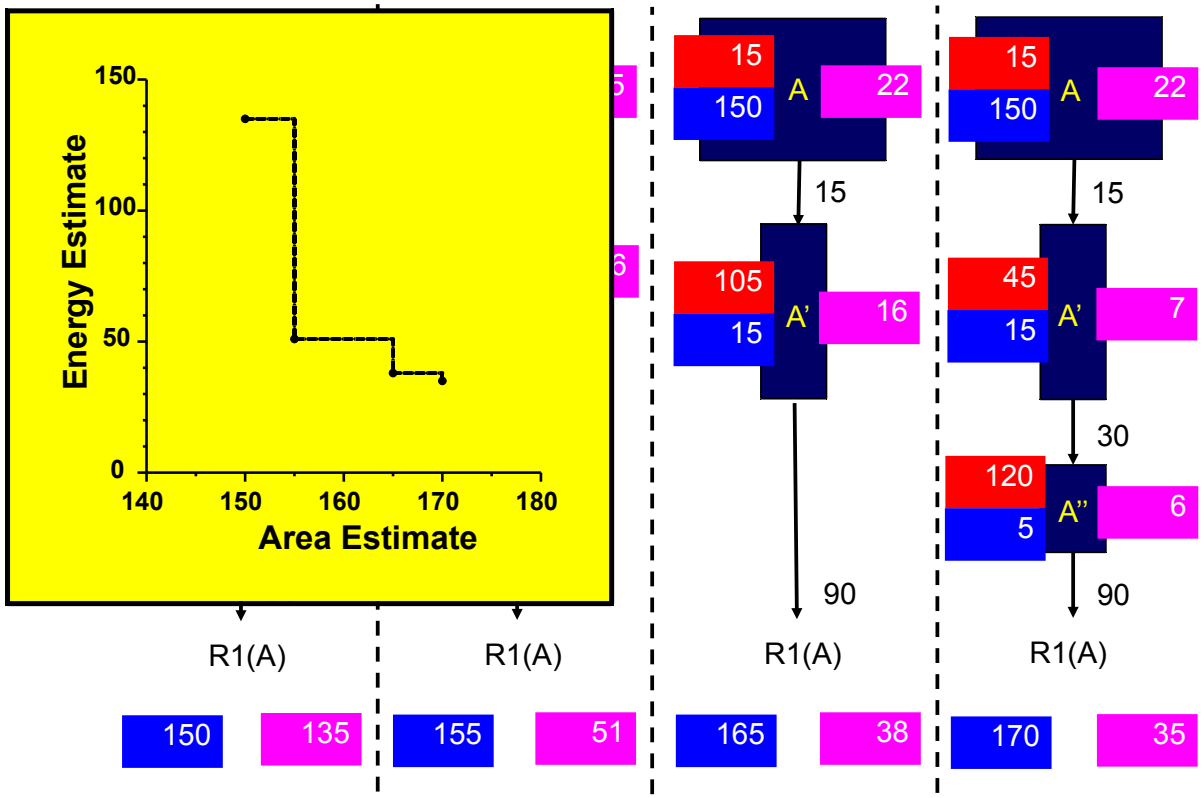
Διαφορετικές αλυσίδες οδηγούν σε διαφορετική κατανάλωση ισχύος/αξιοποίηση επιφάνειας (1/2)

Κόκκινο για τον αριθμό προσβάσεων, Μπλε για την επιφάνεια, Ροζ για ενέργεια.



Διαφορετικές αλυσίδες οδηγούν σε διαφορετική κατανάλωση ισχύος/αξιοποίηση επιφάνειας (2/2)

Κόκκινο για τον αριθμό προσβάσεων, Μπλε για την επιφάνεια, Ροζ για ενέργεια.

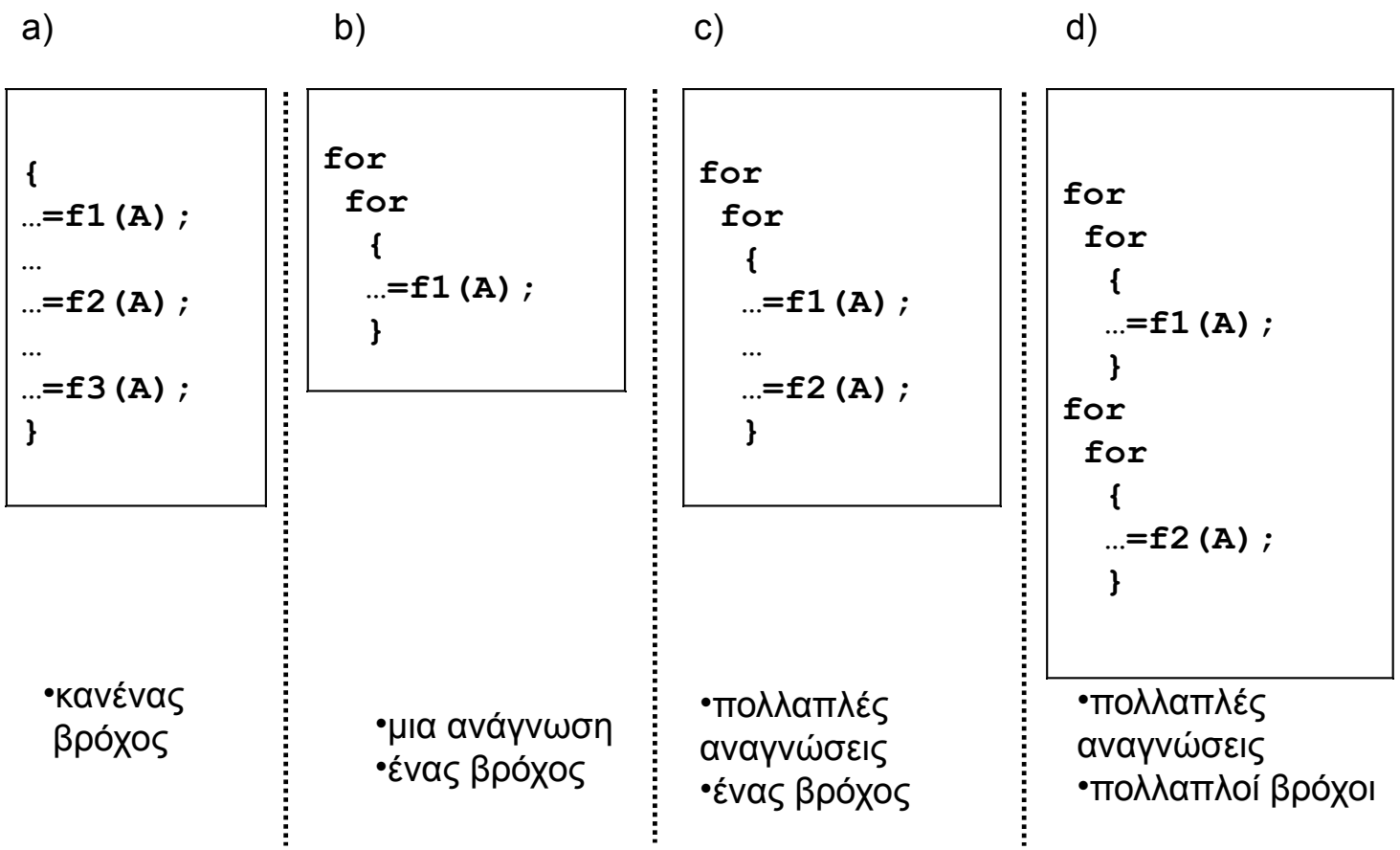


Ερμηνεία προσαρμοσμένης ιεραρχίας μνήμης μέσω ενός απλοποιημένου “script”

1. Αναγνώρισε τα σήματα με επαρκές ενδεχόμενο επαναχρησιμοποίησης.
2. Προσδιόρισε τις αλυσίδες επαναχρησιμοποίησης και απομάκρυνε αυτές.
   (για κάθε ανάγνωση “σειράς”)
3. **Προσδιόρισε τα δέντρα επαναχρησιμοποίησης και απομάκρυνε αυτά.**
   (για κάθε σειρά)
4. Προσδιόρισε το γράφημα επαναχρησιμοποίησης, συμπεριλαμβάνοντας τις παρακάμψεις και απομάκρυνε αυτό.
   (για όλη την εφαρμογή)

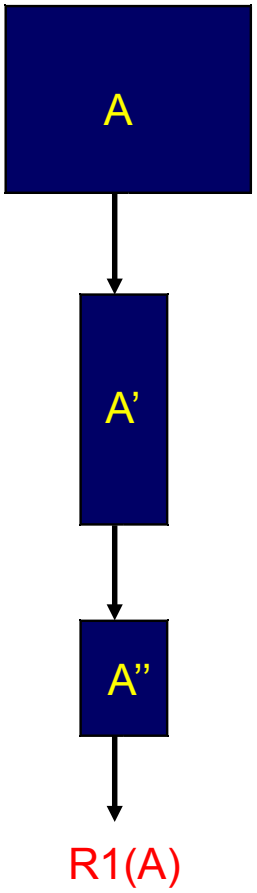
5. Προσδιόρισε την εκχώρηση διάταξης για την ιεραρχία μνήμης, ενσωματώνοντας τους δεδομένους περιορισμούς μνήμης παρασκηνίου *(επιπέδων),* καθώς επίσης και περιορισμούς πραγματικού χρόνου.

Ταξινόμηση ευκαιριών επαναχρησιμοποίησης δεδομένων



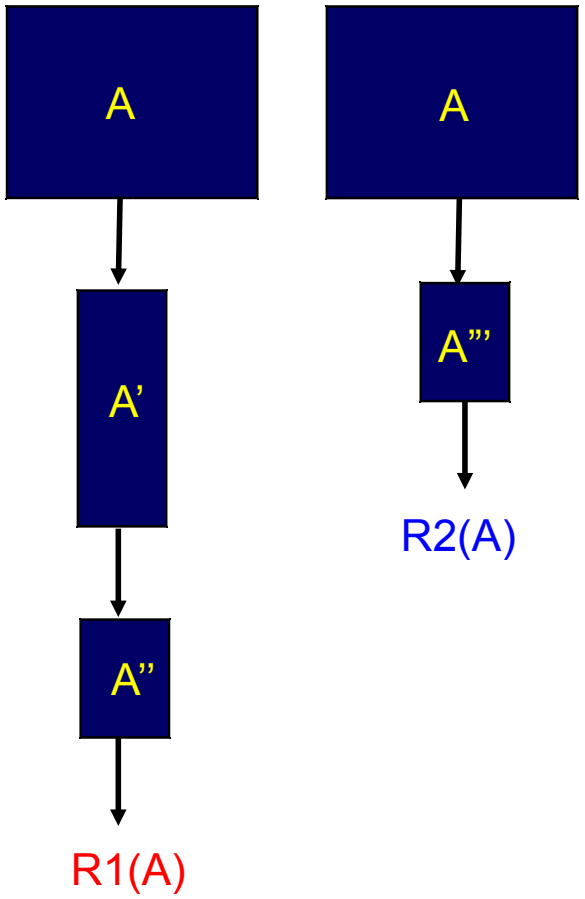
Βήμα 3 (1/3)

Προσδιόρισε τις αλυσίδες και τα δέντρα επαναχρησιμοποίησης δεδομένων.




```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++)
     … = A[i*15+k*5+m];
```

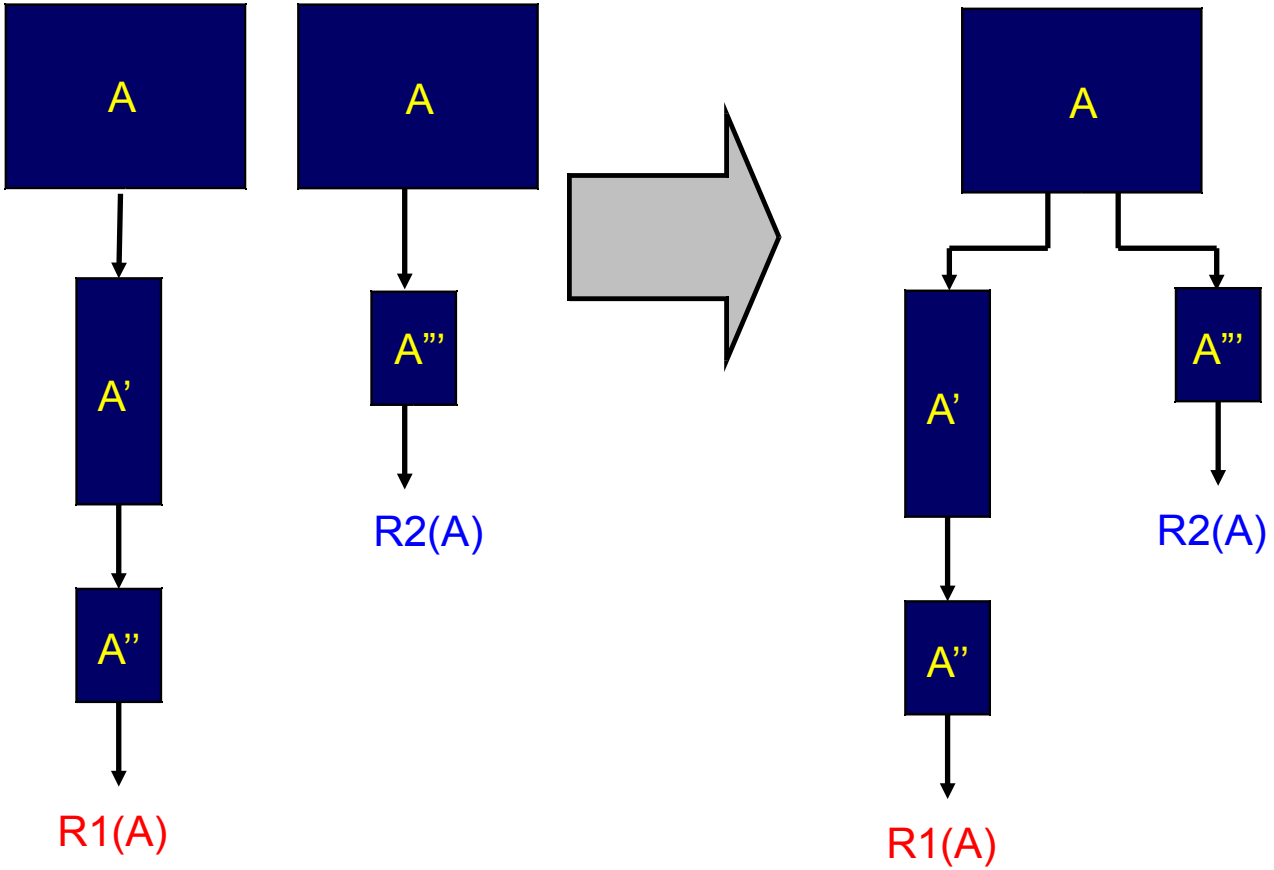
Βήμα 3 (2/3)




```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++)
     … = A[i*15+k*5+m];

 for (x=0; x<8; x++)
  for (y=0; y<5; y++)
     … = A[i*5+y];
```

Βήμα 3 (3/3)

Δέντρο επαναχρησιμοποίησης



Αλυσίδες → Δέντρα (1/9)

```
for … {
 for …
  for … {
   …=A[][];
   …=A[][];
   }
 for … {
  …=A[][];
  }
 }
for …
 for …
  for …{
   …=A[][];
   }
```

1

2

3

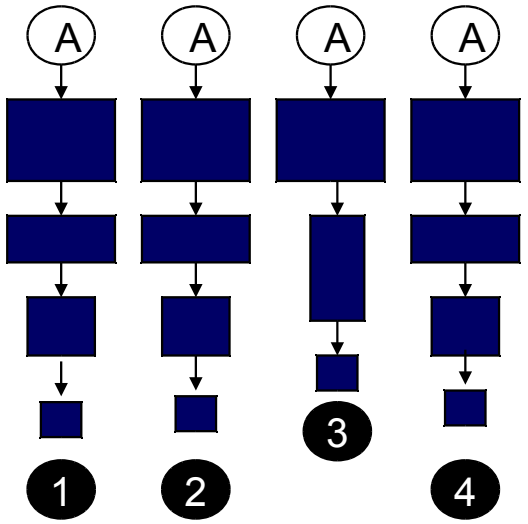
4

Από την επαναχρησιμοποίηση αλυσίδων δεδομένων στην επαναχρησιμοποίηση δέντρων δεδομένων.

Αλυσίδες → Δέντρα (2/9)

```
for … {
 for …
  for … {                1
   …=A[][];              2
   …=A[][];
   }
 for … {
  …=A[][];               3
  }
 }
for …
 for …
  for …{
   …=A[][];
   }                     4
```

Επαναχρησιμοποίηση αλυσίδων: υποψήφιοι προς αντιγραφή

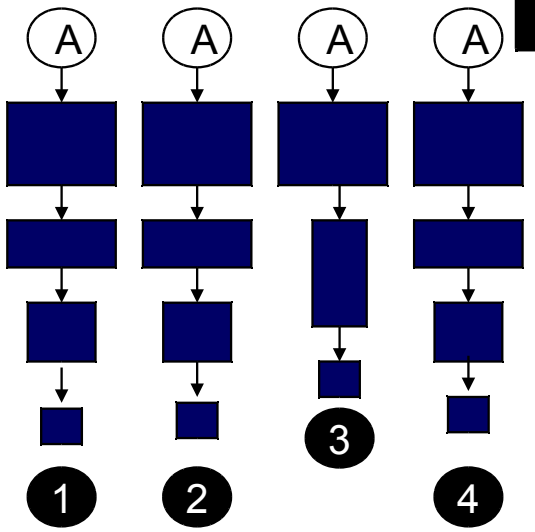


Αλυσίδες → Δέντρα (3/9)

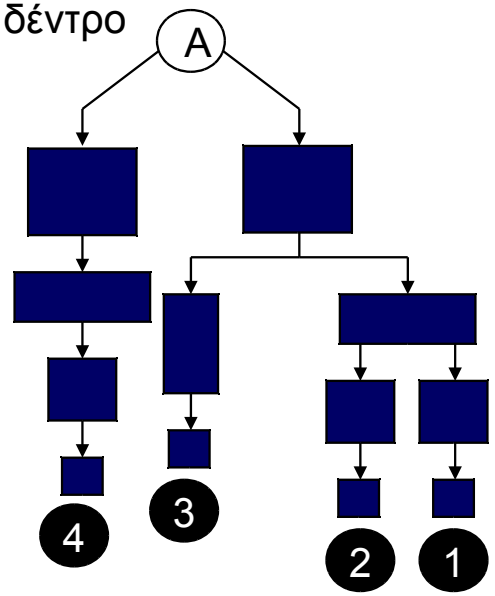
```
for … {
 for …
  for … {
   …=A[][];
   …=A[][];
    }
 for … {
  …=A[][];
  }
 }
for …
 for …
  for …{
   …=A[][];
    }
```

1
2
3
4

Επαναχρησιμοποίηση αλυσίδων: υποψήφιοι προς αντιγραφή



Συνδυασμός αλυσίδων σε εμφωλευμένους βρόχους: δέντρο



Αλυσίδες → Δέντρα (4/9)

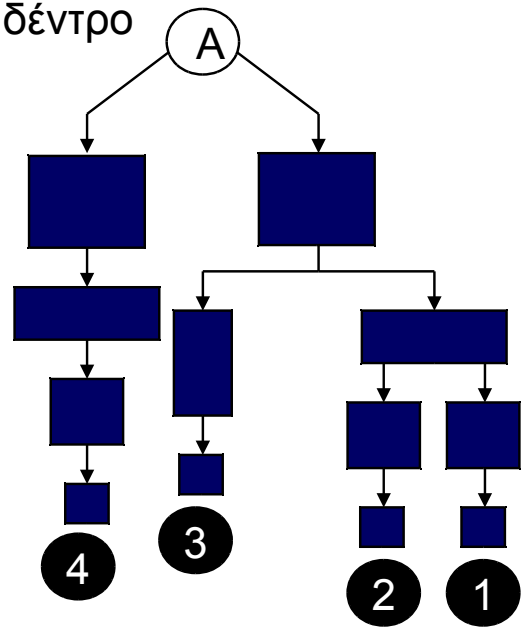
```
for … {
 for …
  for … {
   …=A[][];
   …=A[][];
   }
 for … {
  …=A[][];
  }
 }
for …
 for …
  for …{
   …=A[][];
   }
```

1
2
3
4

Αλυσίδες → Δέντρα (5/9)

```
for … {
 for …
  for … {                1
   …=A[][];              2
   …=A[][];
   }
 for … {
  …=A[][];               3
  }
 }
for …
 for …
  for …{
   …=A[][];
   }                     4
```

Συνδυασμός αλυσίδων σε εμφωλευμένους βρόχους: δέντρο

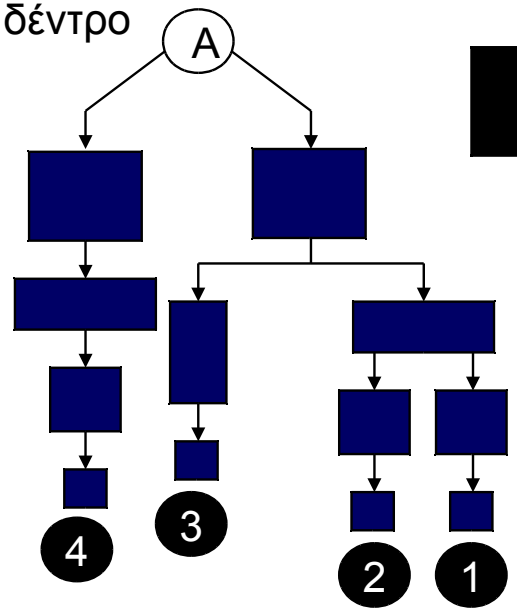


Αλυσίδες → Δέντρα (6/9)

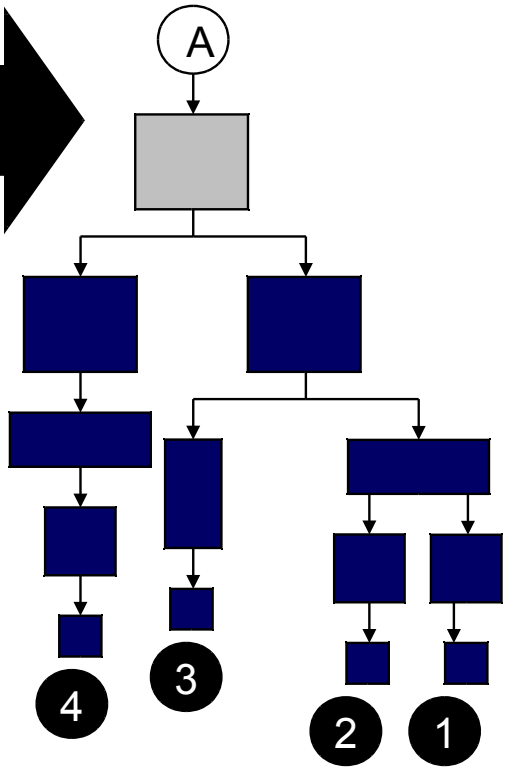
```
for … {
 for …
  for … {
   …=A[][];
   …=A[][];
   }
 for … {
  …=A[][];
  }
 }
for …
 for …
  for …{
   …=A[][];
   }
```

1
2
3
4

Συνδυασμός αλυσίδων σε εμφωλευμένους βρόχους: δέντρο



Συνδυασμός μεταξύ βρόχων: καθολικό δέντρο



Αλυσίδες → Δέντρα (7/9)

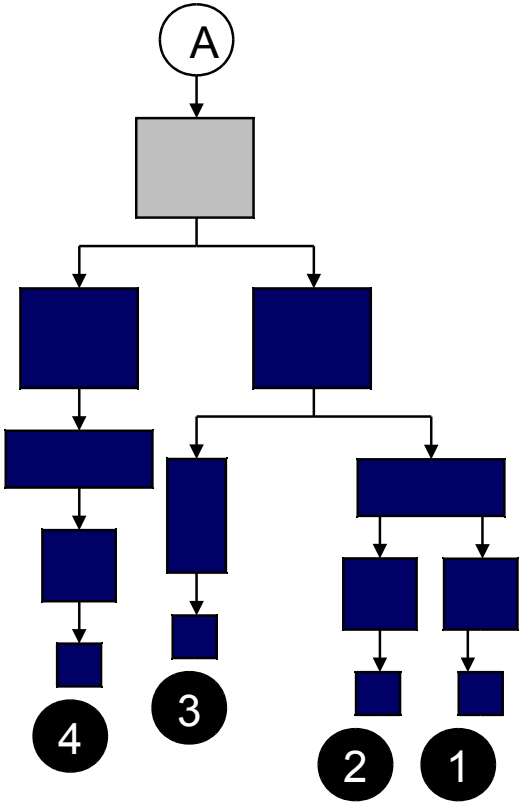
```
for … {
 for …
  for … {          1
   …=A[][];
   …=A[][];        2
   }
 for … {
  …=A[][];         3
  }
 }
for …
 for …
  for …{
   …=A[][];
   }               4
```

Αλυσίδες → Δέντρα (8/9)

```
for … {
 for …
  for … {
   …=A[][];
   …=A[][];
   }
 for … {
  …=A[][];
  }
 }
for …
 for …
  for …{
   …=A[][];
   }
```

1
2
3
4

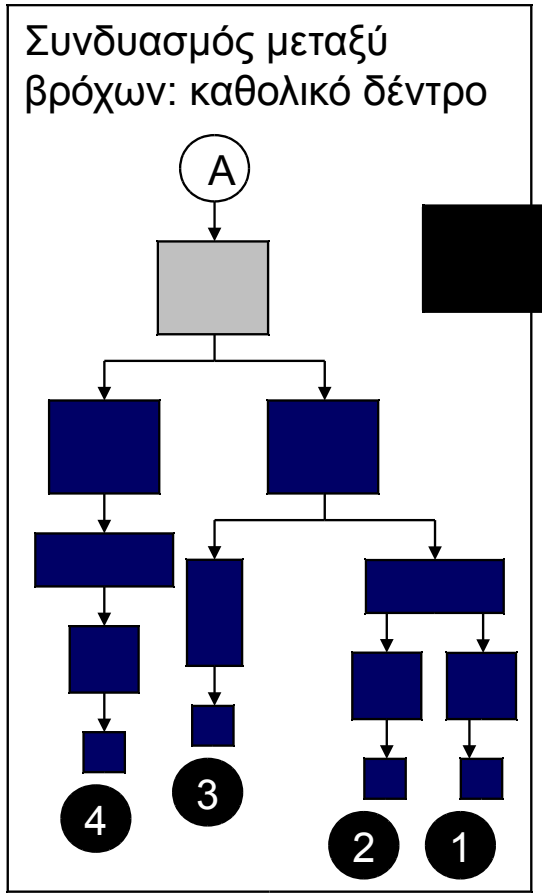
Συνδυασμός μεταξύ βρόχων: καθολικό δέντρο

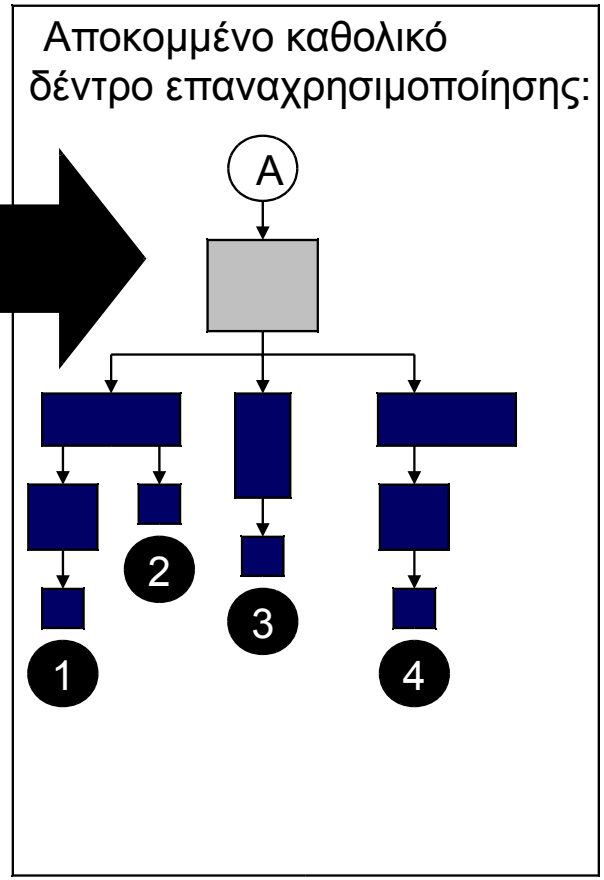


Αλυσίδες → Δέντρα (9/9)

```
for … {
 for …
  for … {
   …=A[][];
   …=A[][];
   }
 for … {
  …=A[][];
  }
 }
for …
 for …
  for …{
   …=A[][];
   }
```

1
2
3
4





Ερμηνεία προσαρμοσμένης ιεραρχίας μνήμης μέσω απλοποιημένου “script”

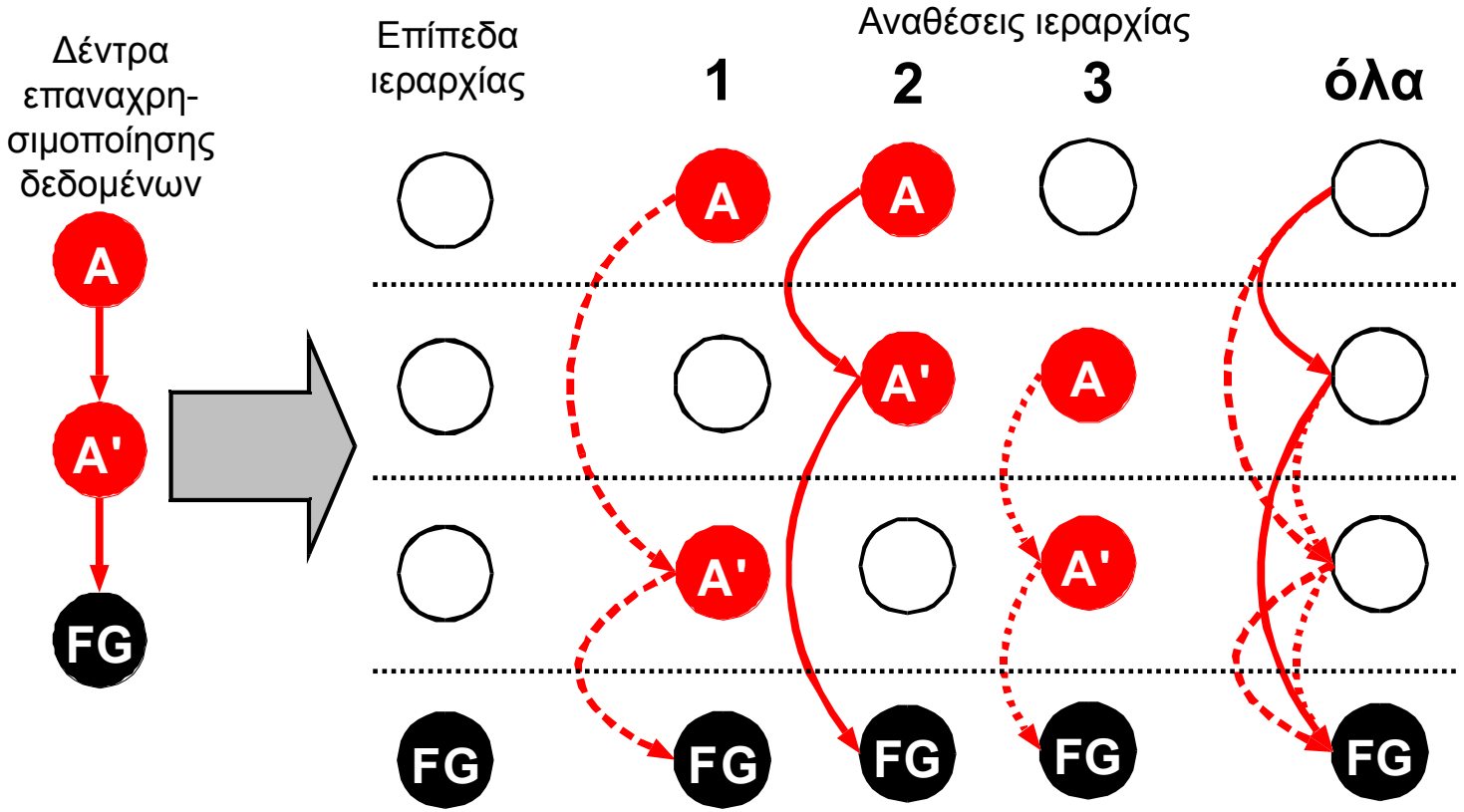
1. Αναγνώρισε τα σήματα με επαρκές ενδεχόμενο επαναχρησιμοποίησης.
2. Προσδιόρισε τις αλυσίδες επαναχρησιμοποίησης και απομάκρυνε αυτές.
   (για κάθε ανάγνωση “σειράς”)
3. Προσδιόρισε τα δέντρα επαναχρησιμοποίησης και απομάκρυνε αυτά.
   (για κάθε σειρά)
4. **Προσδιόρισε το γράφημα επαναχρησιμοποίησης, συμπεριλαμβάνοντας τις παρακάμψεις και απομάκρυνε αυτό.**
   (για όλη την εφαρμογή)

5. Προσδιόρισε την εκχώρηση διάταξης για την ιεραρχία μνήμης, ενσωματώνοντας τους δεδομένους περιορισμούς μνήμης παρασκηνίου *(επιπέδων),* καθώς επίσης και περιορισμούς πραγματικού χρόνου.

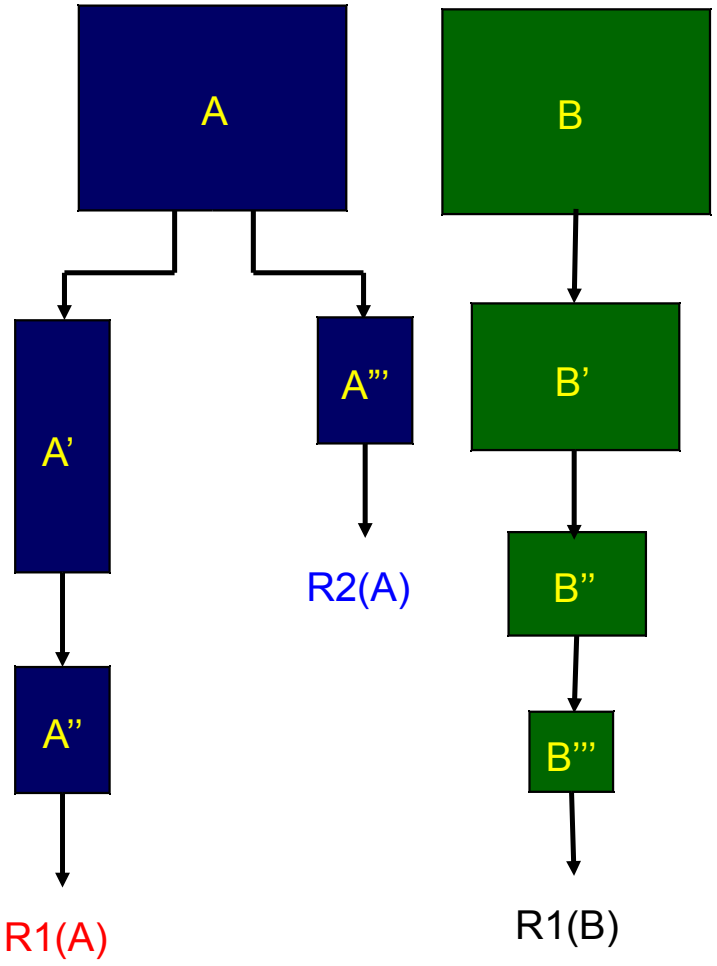
Βήμα 4 (1/5)

Προσδιόρισε το γράφημα επαναχρησιμοποίησης δεδομένων.

Ελευθερία εκχώρησης του Α στη μνήμη ιεραρχίας.

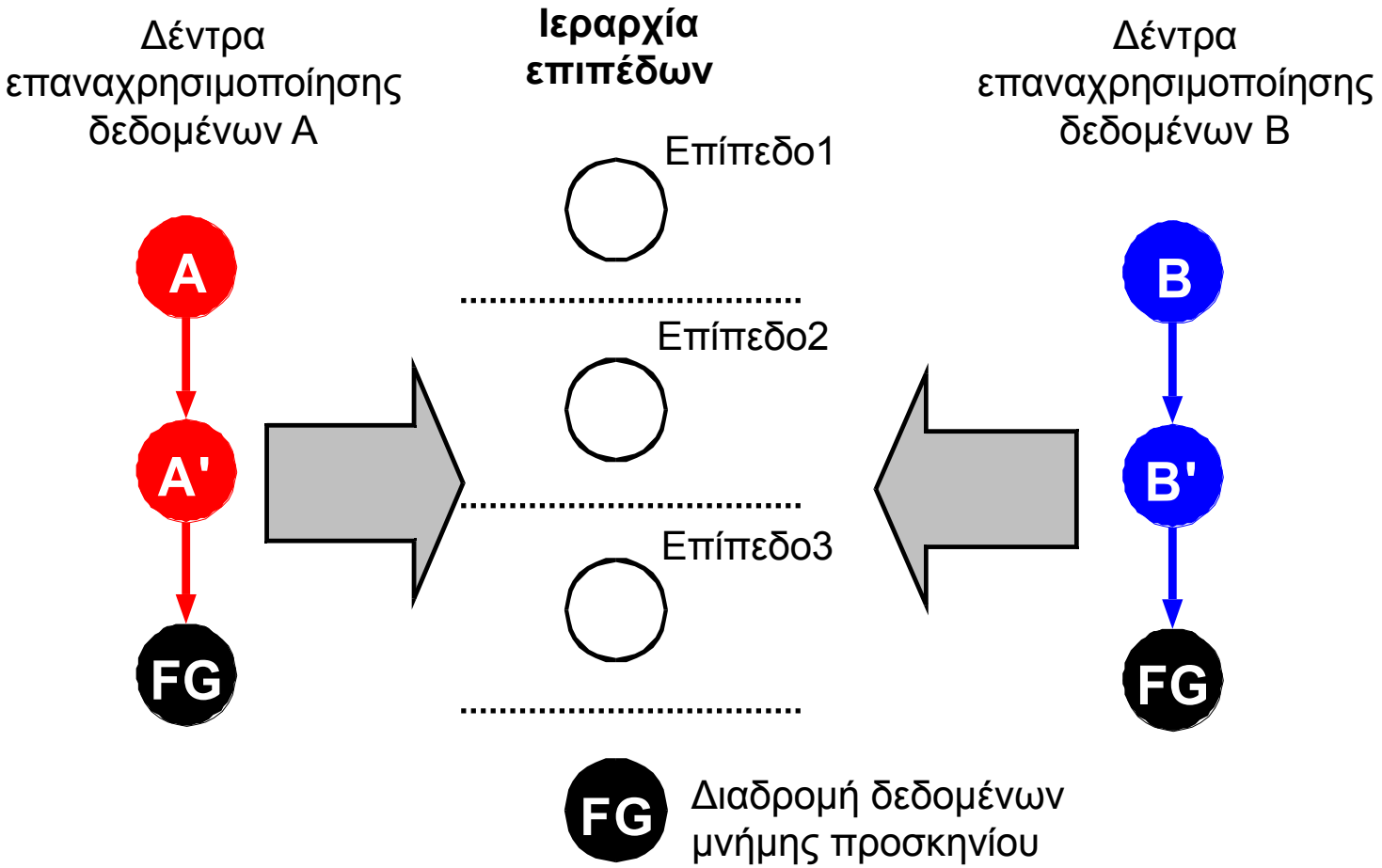


Βήμα 4 (2/5)



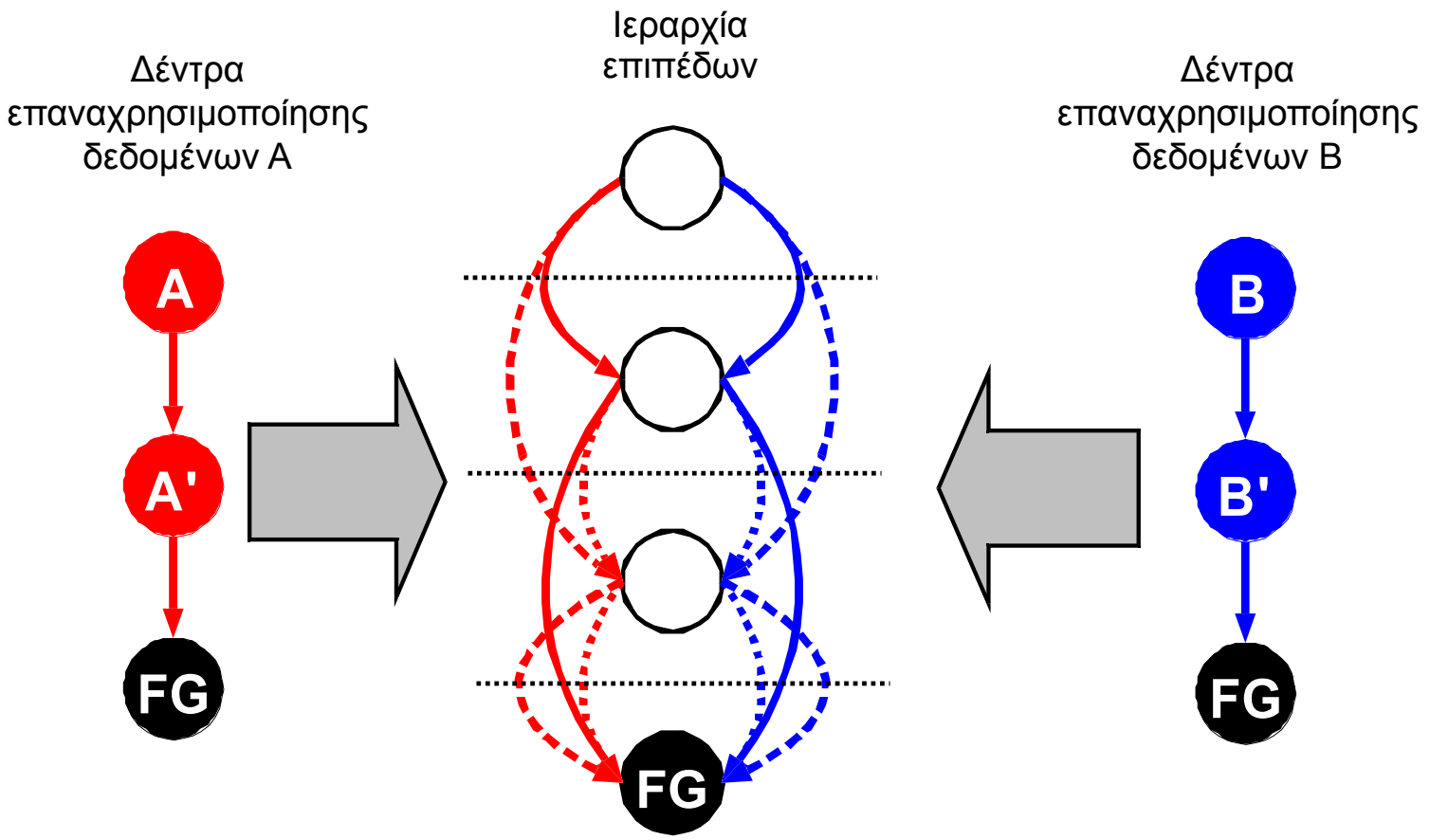
Προσδιόρισε το γράφημα επαναχρησιμοποίησης δεδομένων και τον αριθμό επιπέδων.

Βήμα 4 (3/5)


Δέντρα επαναχρησιμοποίησης δεδομένων Α
Ιεραρχία επιπέδων
Δέντρα επαναχρησιμοποίησης δεδομένων Β
Επίπεδο1
Επίπεδο2
Επίπεδο3
A
A'
FG
B
B'
FG
FG
Διαδρομή δεδομένων μνήμης προσκηνίου

Βήμα 4 (4/5)

Όλη η ελευθερία του σήματος, στην ιεραρχία μνήμης

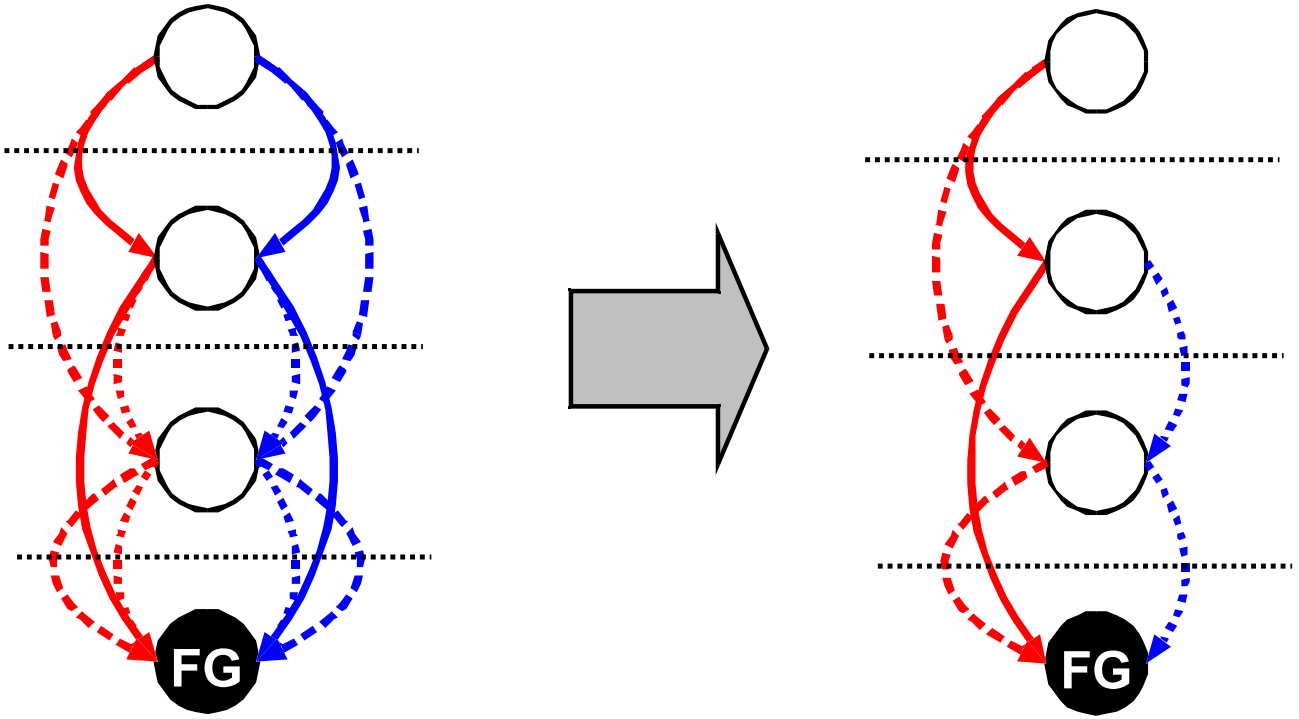


Βήμα 4 (5/5)

Απομάκρυνε το γράφημα επαναχρησιμοποίησης *(ανεξαρτήτως πλατφόρμας)*

Επίπεδα ιεραρχίας
Απόλυτη ελευθερία

Αποκομμένα επίπεδα ιεραρχίας



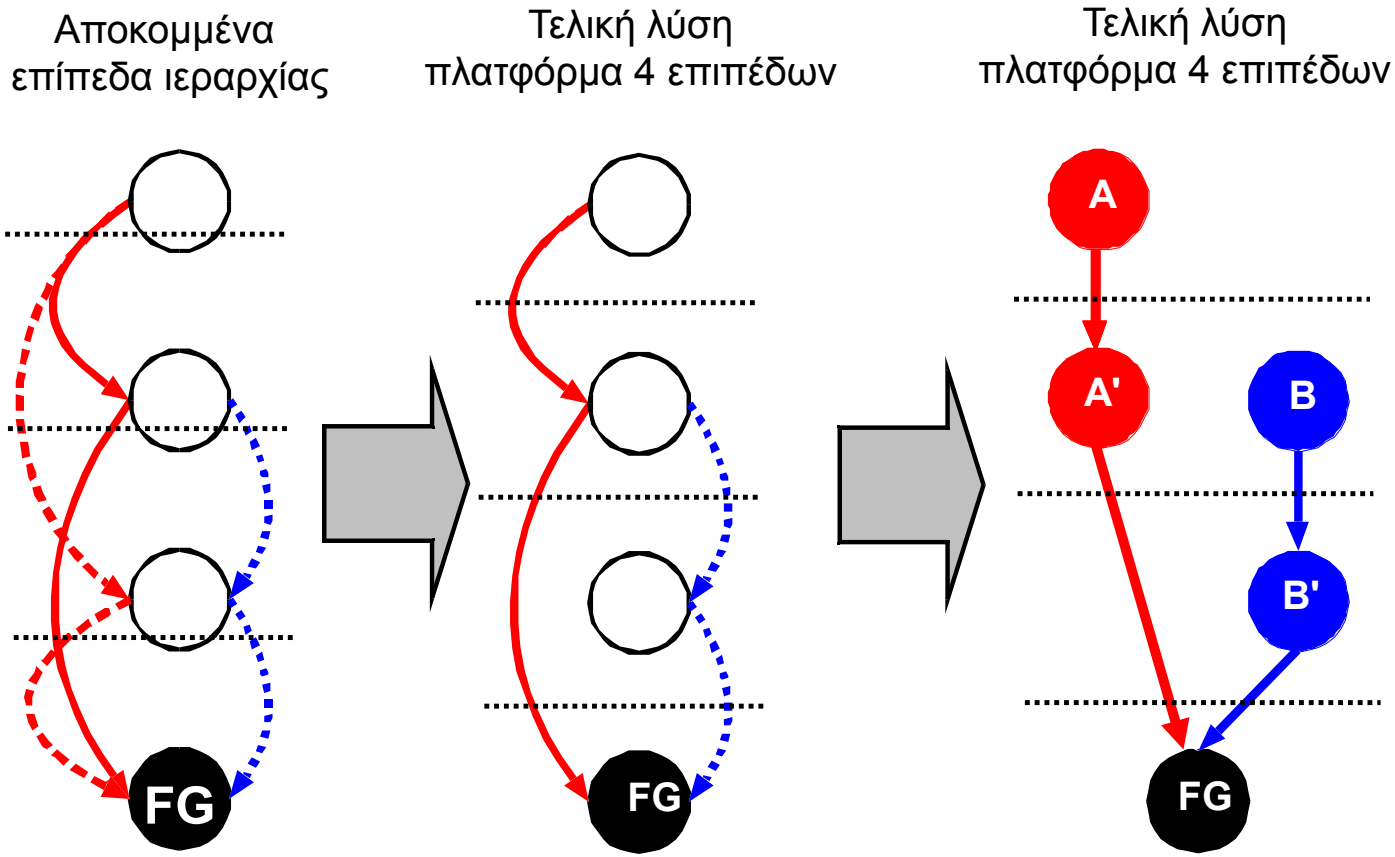
Ερμηνεία της προσαρμοσμένης ιεραρχίας μνήμης μέσω απλοποιημένου “script”

1. Αναγνώρισε τα σήματα με επαρκές ενδεχόμενο επαναχρησιμοποίησης.
2. Προσδιόρισε τις αλυσίδες επαναχρησιμοποίησης και απομάκρυνε αυτές.
   (για κάθε ανάγνωση “σειράς”)
3. Προσδιόρισε τα δέντρα επαναχρησιμοποίησης και απομάκρυνε αυτά.
   (για κάθε σειρά)
4. Προσδιόρισε το γράφημα επαναχρησιμοποίησης, συμπεριλαμβάνοντας τις παρακάμψεις και απομάκρυνε αυτό.
   (για όλη την εφαρμογή)

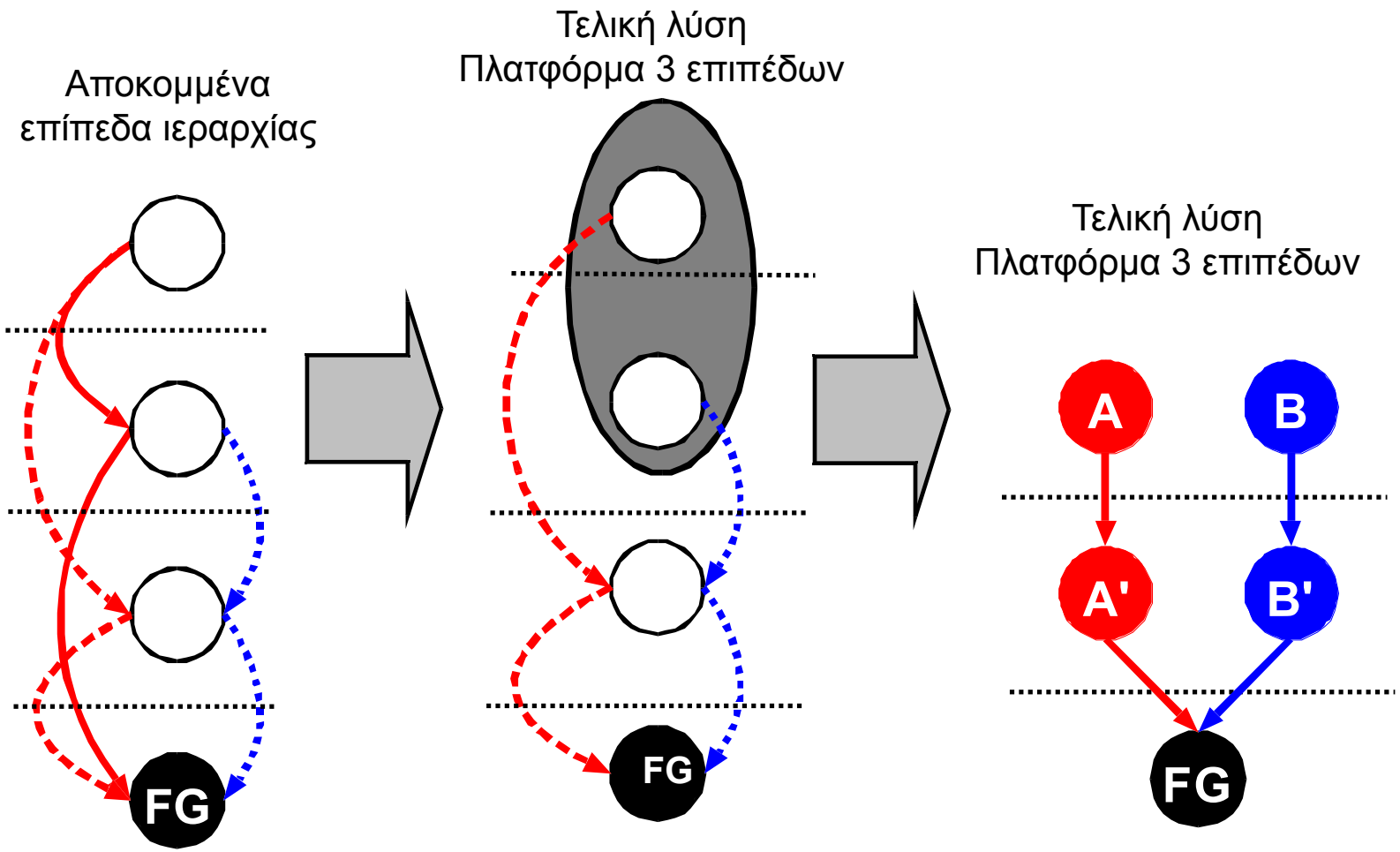
5. **Προσδιόρισε την εκχώρηση διάταξης για την ιεραρχία μνήμης, ενσωματώνοντας τους δεδομένους περιορισμούς μνήμης παρασκηνίου *(επιπέδων),* καθώς επίσης και περιορισμούς πραγματικού χρόνου.**

Βήμα 5 (1/2)

Προσδιόρισε το γράφημα επαναχρησιμοποίησης δεδομένων.
Απομάκρυνε περαιτέρω αυτό το γράφημα *(ανεξαρτήτως πλατφόρμας).*



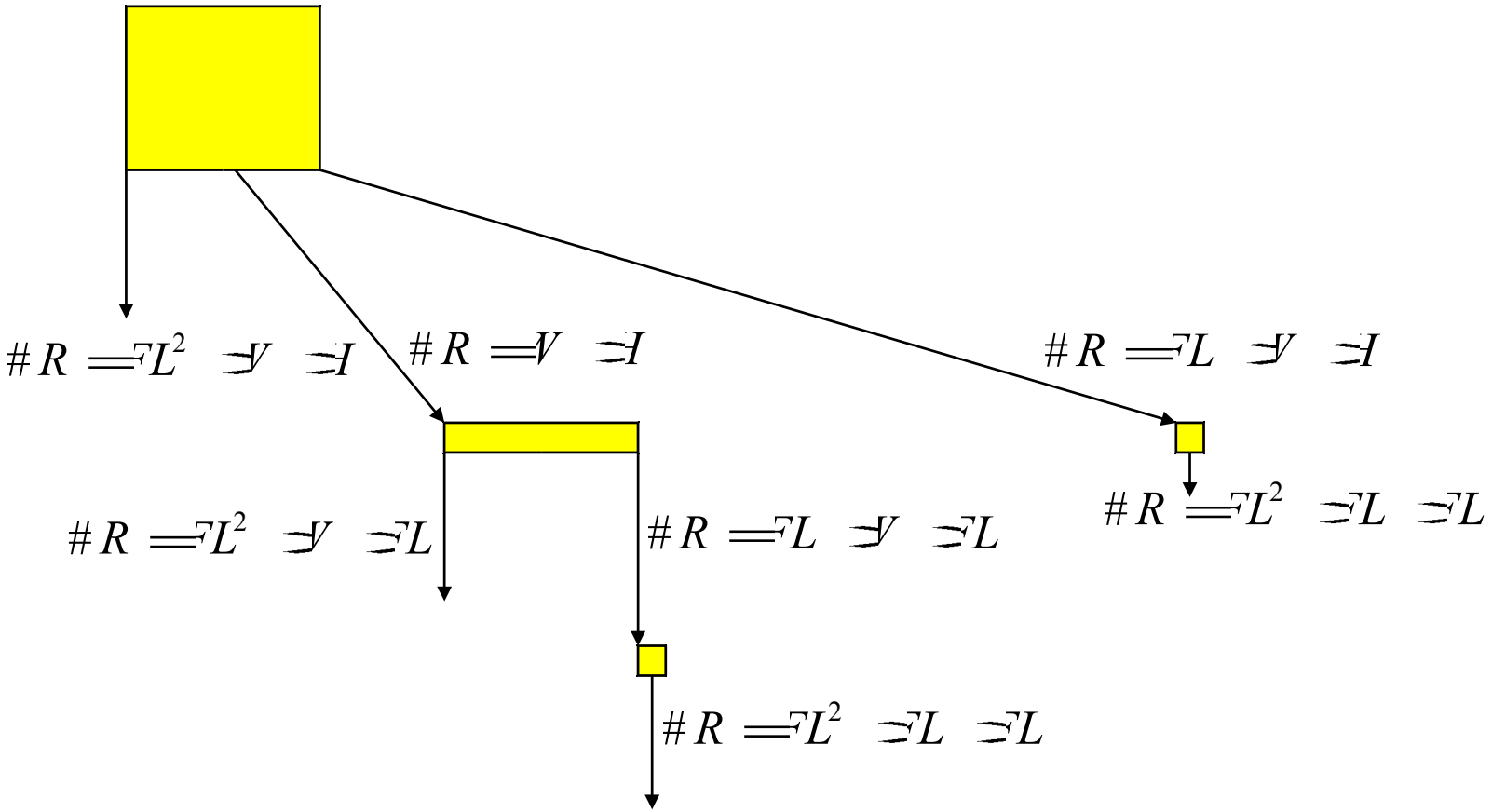
Βήμα 5 (2/2)


Αποκομμένα
επίπεδα ιεραρχίας
Τελική λύση
Πλατφόρμα 3 επιπέδων
Τελική λύση
Πλατφόρμα 3 επιπέδων
A
B
A'
B'
FG
FG
FG

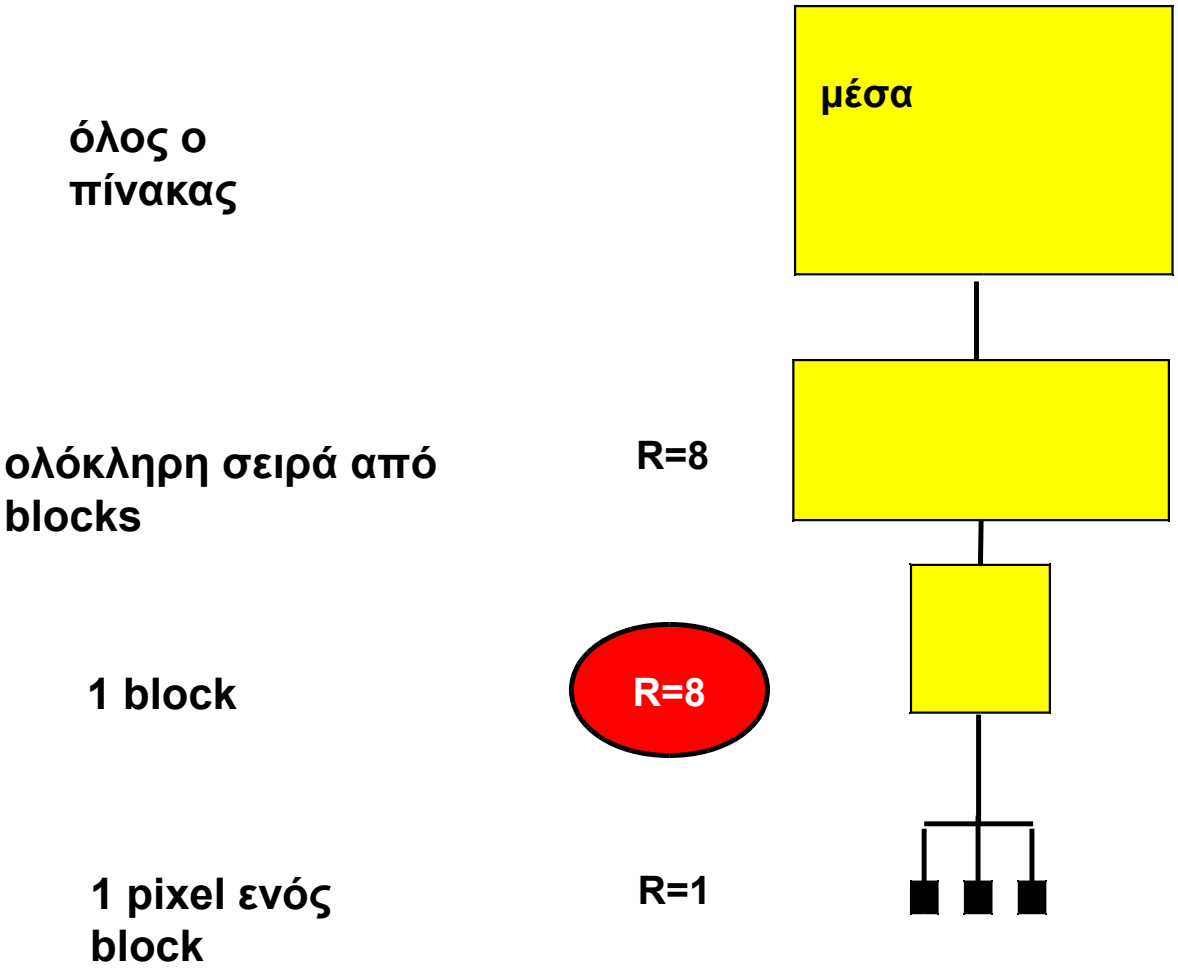
Παράδειγμα απλής συνέλιξης

```
int in[H][W+8], out[H][W];
const int c[] = {1,0,1,2,2,1,0,1};
for (r=0; r < H; r++)
  for (c=0; c < W; c++)
     for (dc=0; dc < 8; dc++)
        out[r][c] += in[r][c+dc]*cf[dc];
```

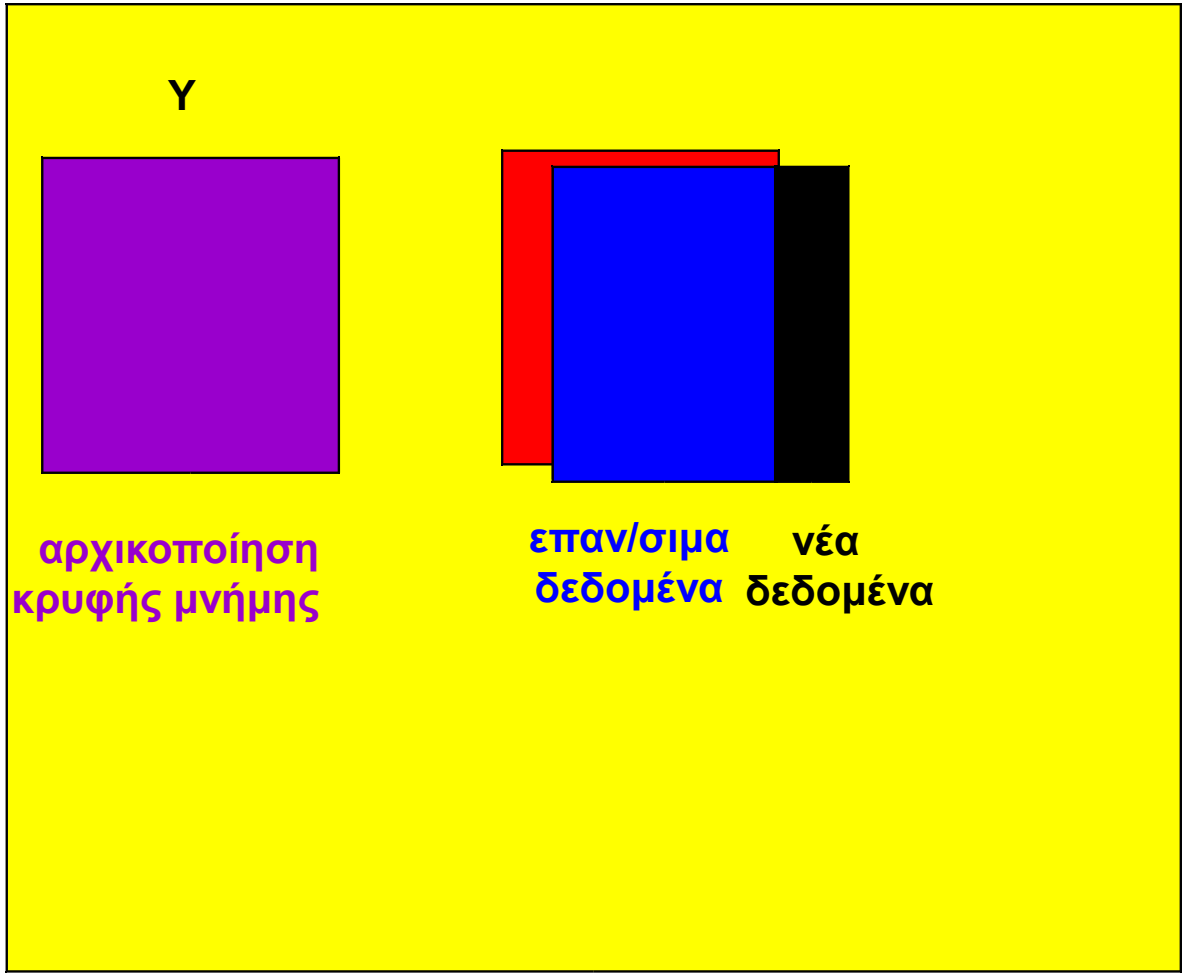
Ιεραρχία μνήμης για το παράδειγμα της συνέλιξης



Επαναχρησιμοποίηση Δεδομένων σε 8x8 συνέλιξη



Πως παράγουμε σαφή αντίγραφα



Παρουσίαση επαναχρησιμοποιήσιμης κρυφής μνήμης 1Δ (1/3)

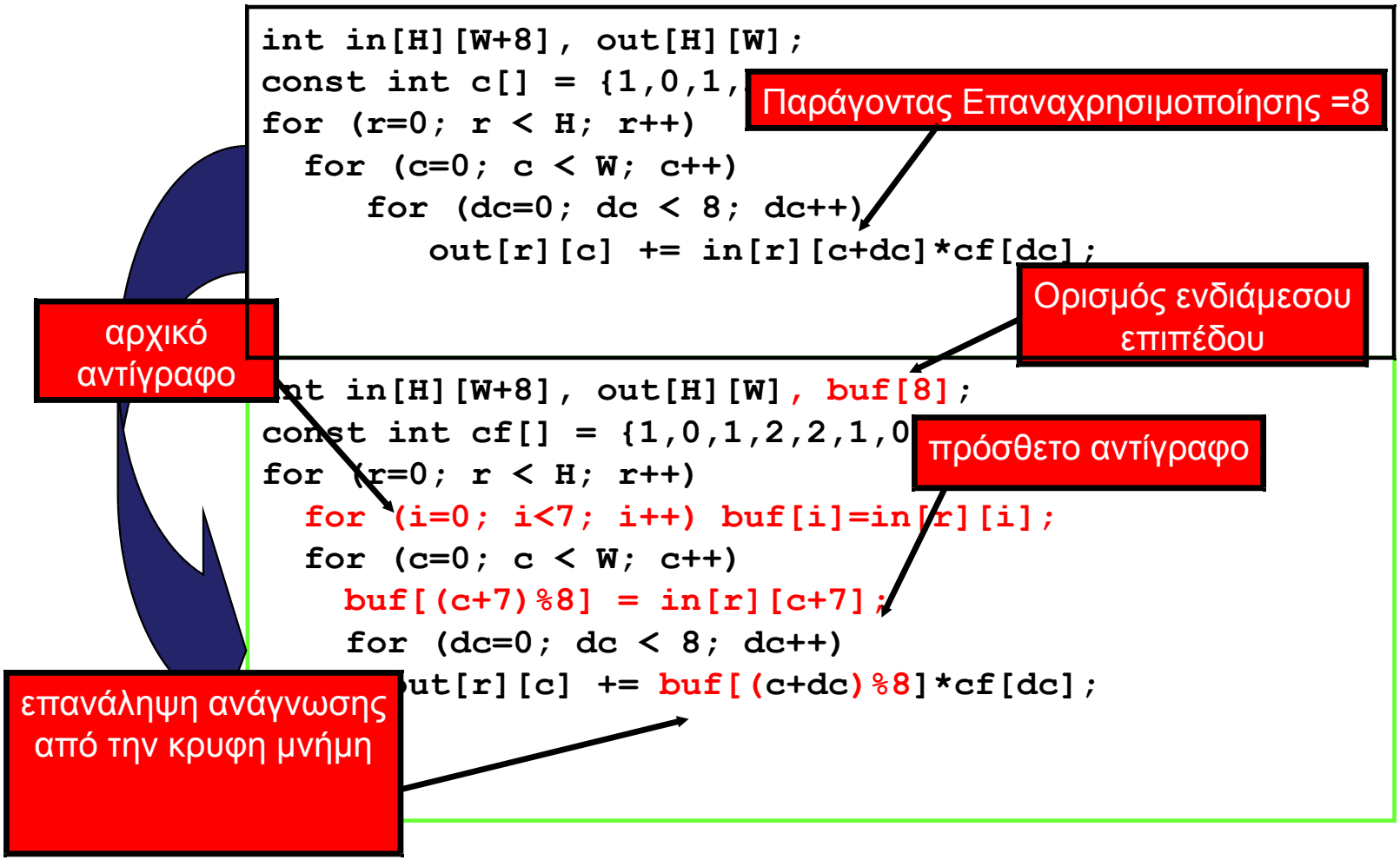
```
int in[H][W+8], out[H][W];
const int c[] = {1,0,1,2,2,1,0,1};
for (r=0; r < H; r++)
  for (c=0; c < W; c++)
     for (dc=0; dc < 8; dc++)
        out[r][c] += in[r][c+dc]*cf[dc];
```

Παρουσίαση επαναχρησιμοποιήσιμης κρυφής μνήμης 1Δ (2/3)

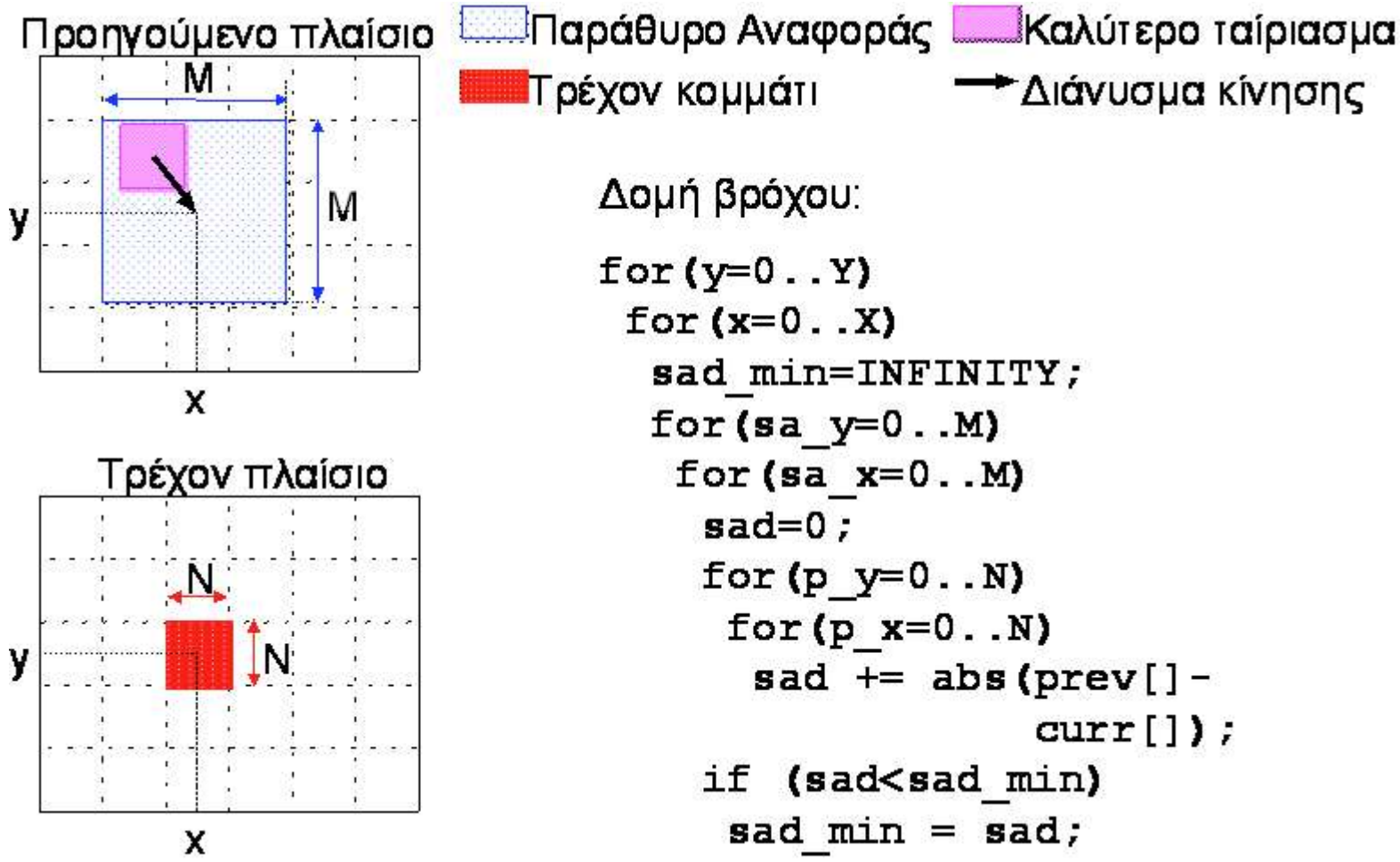
```
int in[H][W+8], out[H][W];
const int c[] = {1,0,1,2,2,1,0,1};
for (r=0; r < H; r++)
  for (c=0; c < W; c++)
     for (dc=0; dc < 8; dc++)
        out[r][c] += in[r][c+dc]*cf[dc];
```

Παράγοντας Επαναχρησιμοποίησης =8

Παρουσίαση επαναχρησιμοποιήσιμης κρυφής μνήμης 1Δ (3/3)



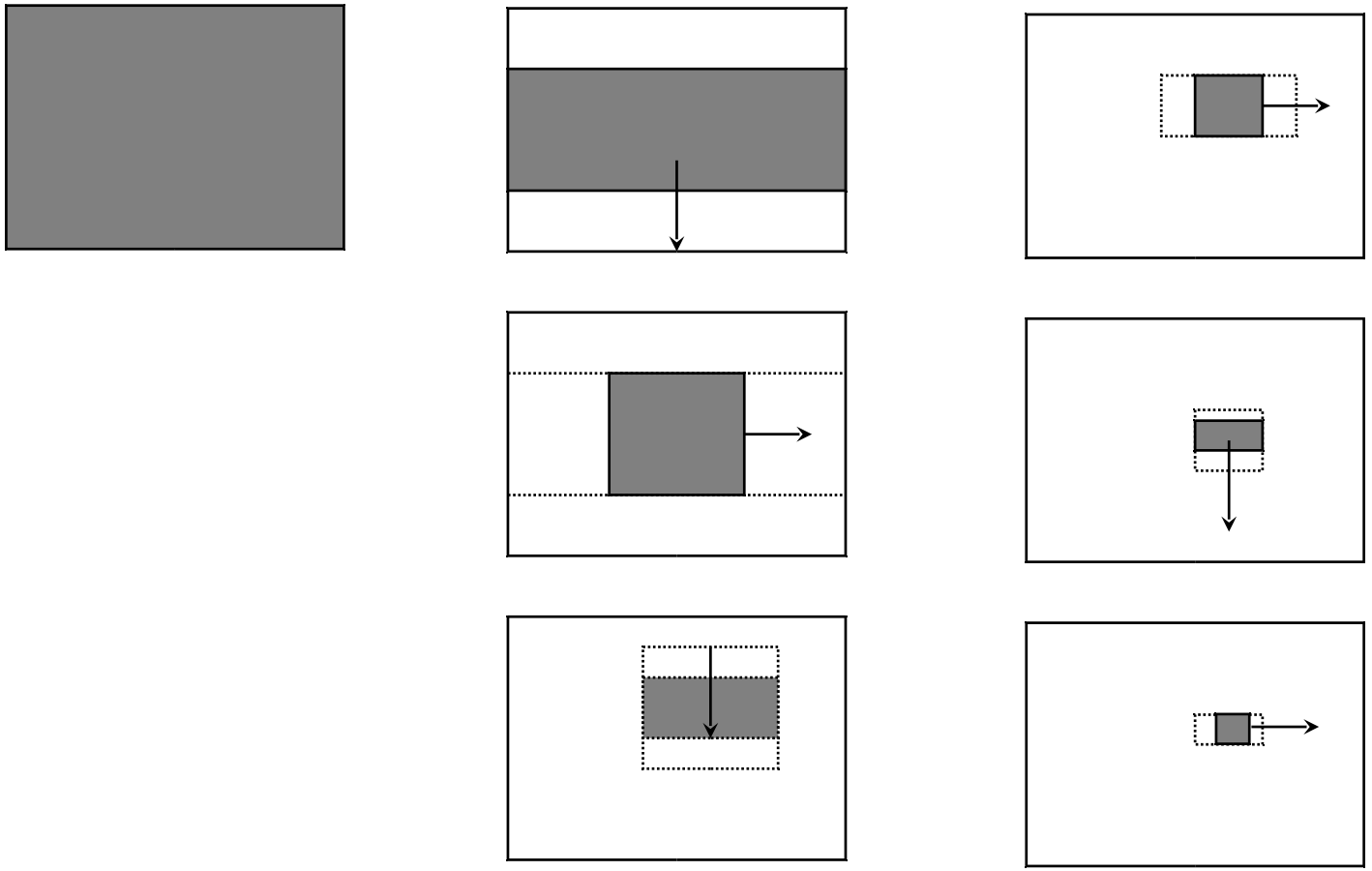
Εφαρμογή βίντεο: βασική αρχή εκτίμησης κίνησης



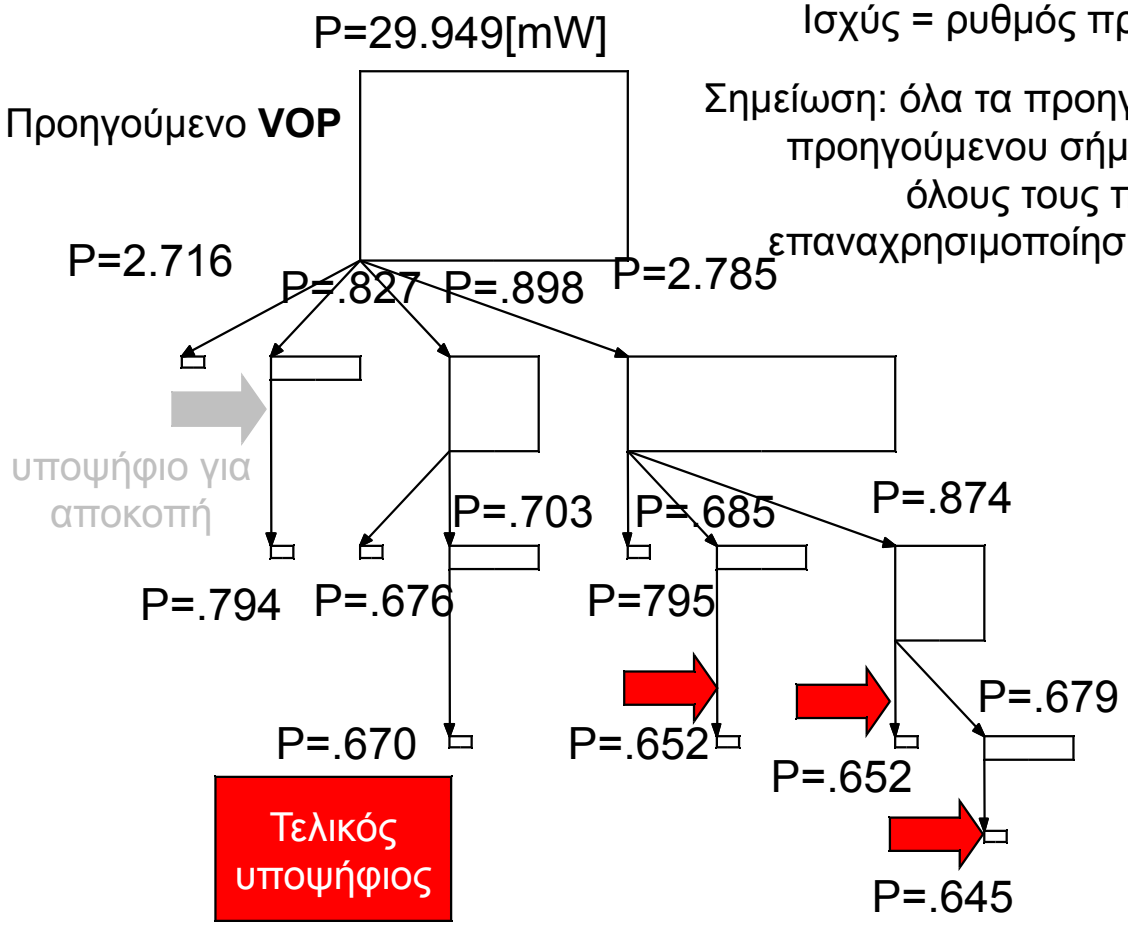
Δομή βρόχου:

```
for(y=0..Y)
 for(x=0..X)
  sad_min=INFINITY;
  for(sa_y=0..M)
   for(sa_x=0..M)
    sad=0;
    for(p_y=0..N)
     for(p_x=0..N)
      sad += abs(prev[]-
                 curr[]);
    if (sad<sad_min)
     sad_min = sad;
```

Πιθανότητες επαναχρησιμοποίησης για την εκτίμηση κίνησης

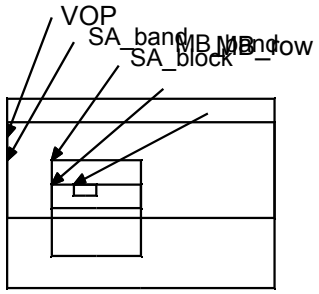


Αποκοπή δέντρου επαναχρησιμοποίησης για το MPEG4 ME



Ισχύς = ρυθμός πρόσβασης * f(μέγεθος)

Σημείωση: όλα τα προηγούμενα αντίγραφα σημάτων του προηγούμενου σήματος(VOP), προσδιορίζουν όλους τους πιθανούς παράγοντες επαναχρησιμοποίησης (υπολογίζονται μια φορά)



ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗΣ ΔΕΔΟΜΕΝΩΝ

Μεταφορά Block κατά την Αποθήκευση Δεδομένων

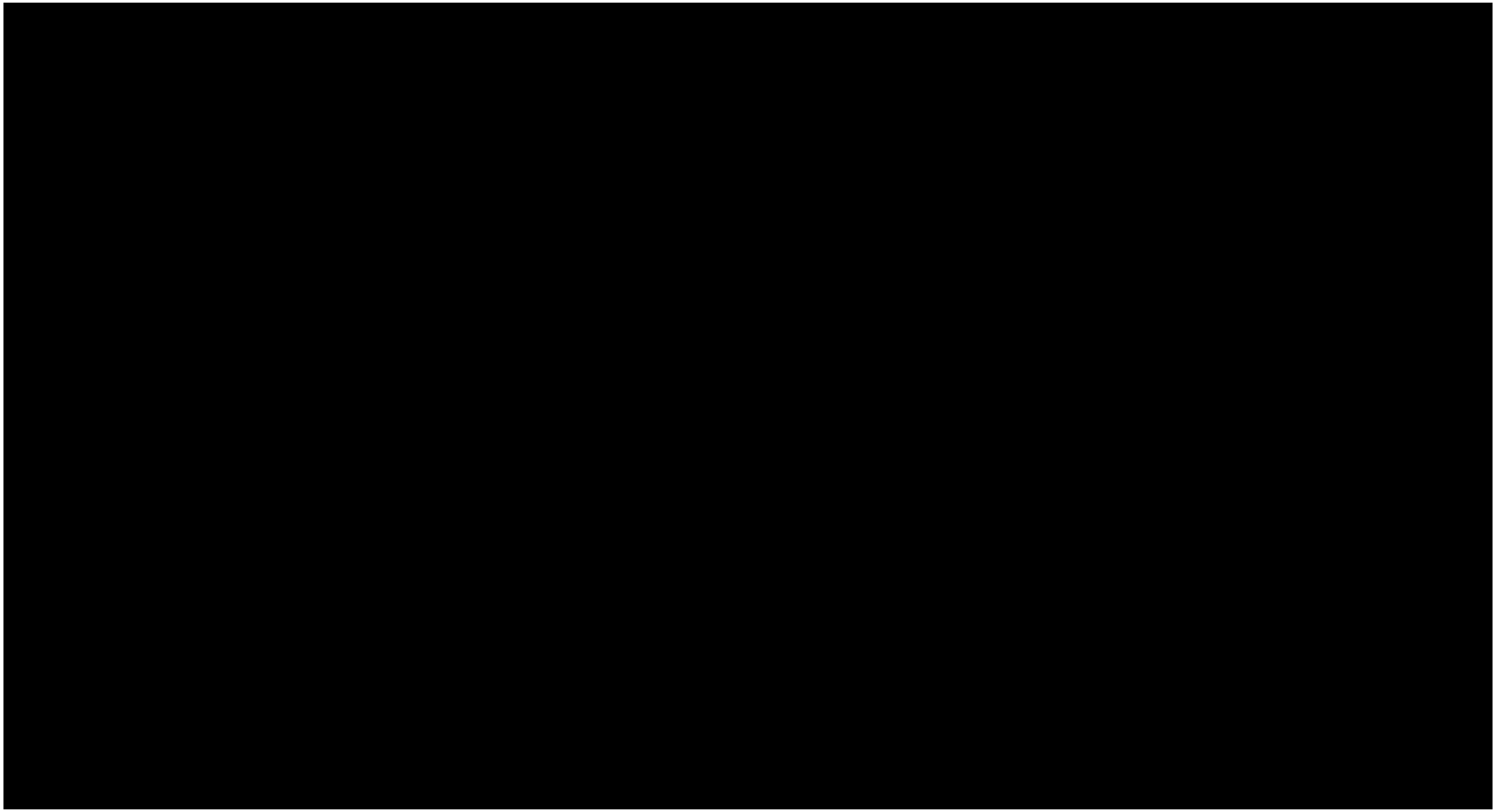
Επαναχρησιμοποίηση Δεδομένων

Παρουσίαση από IMEC

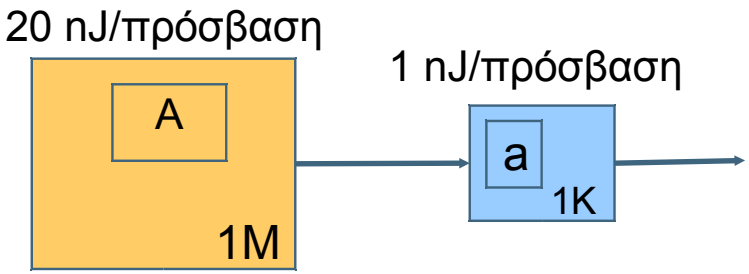
Παράδειγμα

```
for (i=0; i<1000; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = f(A[i],j,k);
```

Παράδειγμα: Αποθήκευση σε καταχωρητή



Κατανάλωση ενέργειας στην ιεραρχία μνήμης (1/2)



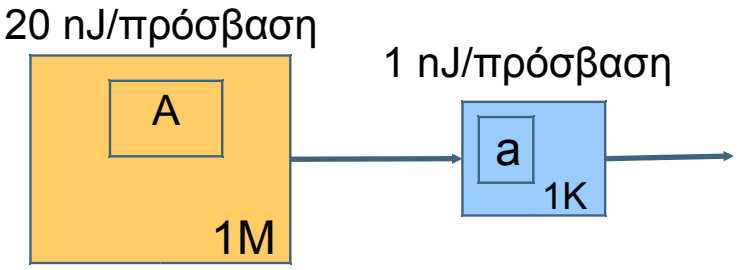

```
for (i=0; i<1000; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = f(A[i],j,k);
```

E=?

```
for (i=0; i<1000; i++)
  a = A[i];
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = f(a,j,k);
```

E=?

Κατανάλωση ενέργειας στην ιεραρχία μνήμης (2/2)



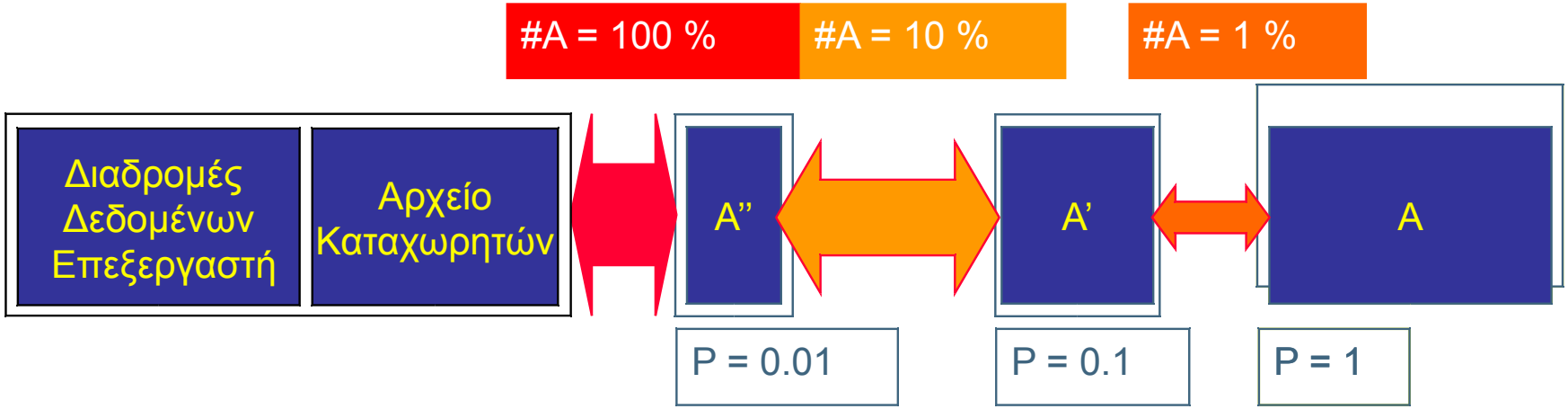

```
for (i=0; i<1000; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = f(A[i],j,k);
```

E = 1000 x 3 x 6 x 20 nJ
= 360 μJ

```
for (i=0; i<1000; i++)
  a = A[i];
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = f(a,j,k);
```

E = 1000 x 20 nJ
+ 1000 x 1 nJ
+ 1000 x 3 x 6 x 1 nJ
= 39 μJ

Απόφαση επαναχρησιμοποίησης δεδομένων και ιεραρχίας μνήμης: αρχή



Εισαγωγή ιεραρχίας μνήμης για ελαχιστοποίηση του αριθμού αναγνώσεων στην κύρια μνήμη

Ανταλλαγή 1: Περισσότερες αναγνώσεις από μικρότερες μνήμες έναντι αυξημένης επιβάρυνσης μεταφερθέντων δεδομένων *(αντίγραφα)*.

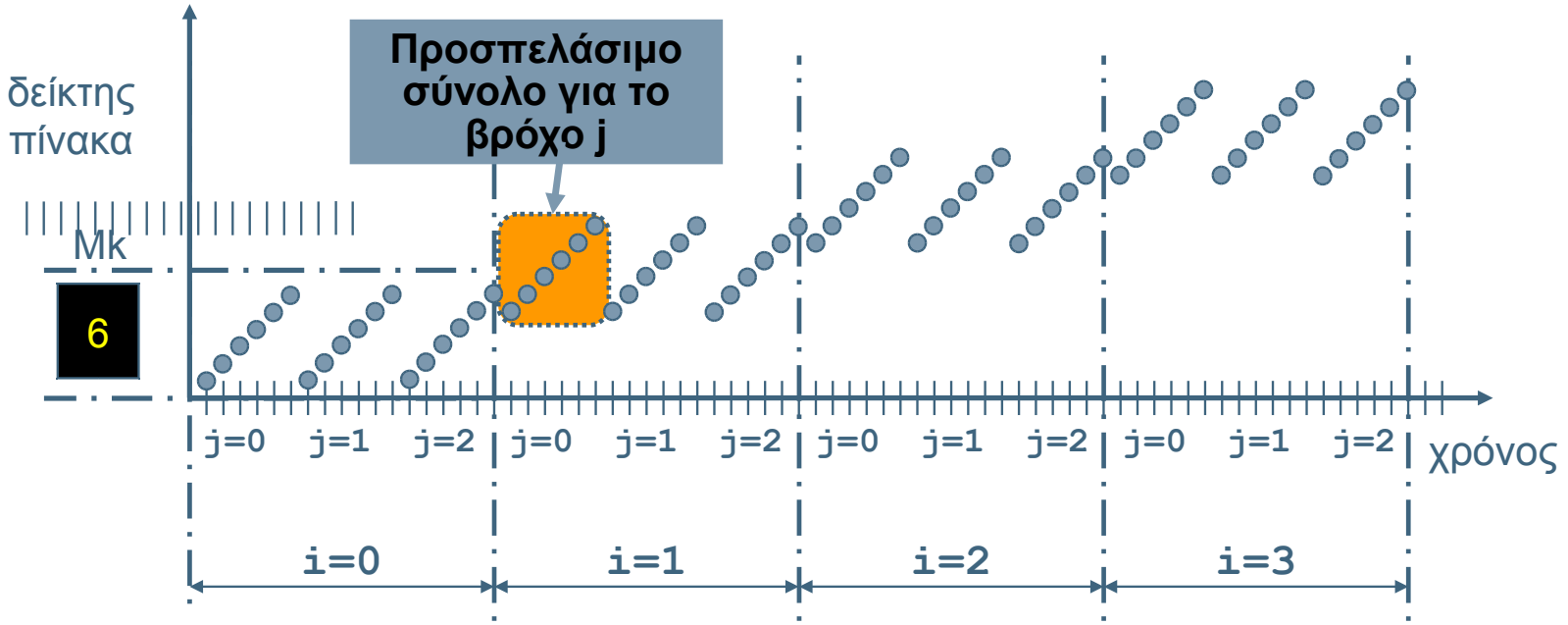
Ανταλλαγή 2: Ισχύς **×** Επιφάνεια (**×** Πολυπλοκότητα Κώδικα)

Πως θα βρεθούν τα δεδομένα που πρέπει να αντιγραφούν; (1/2)

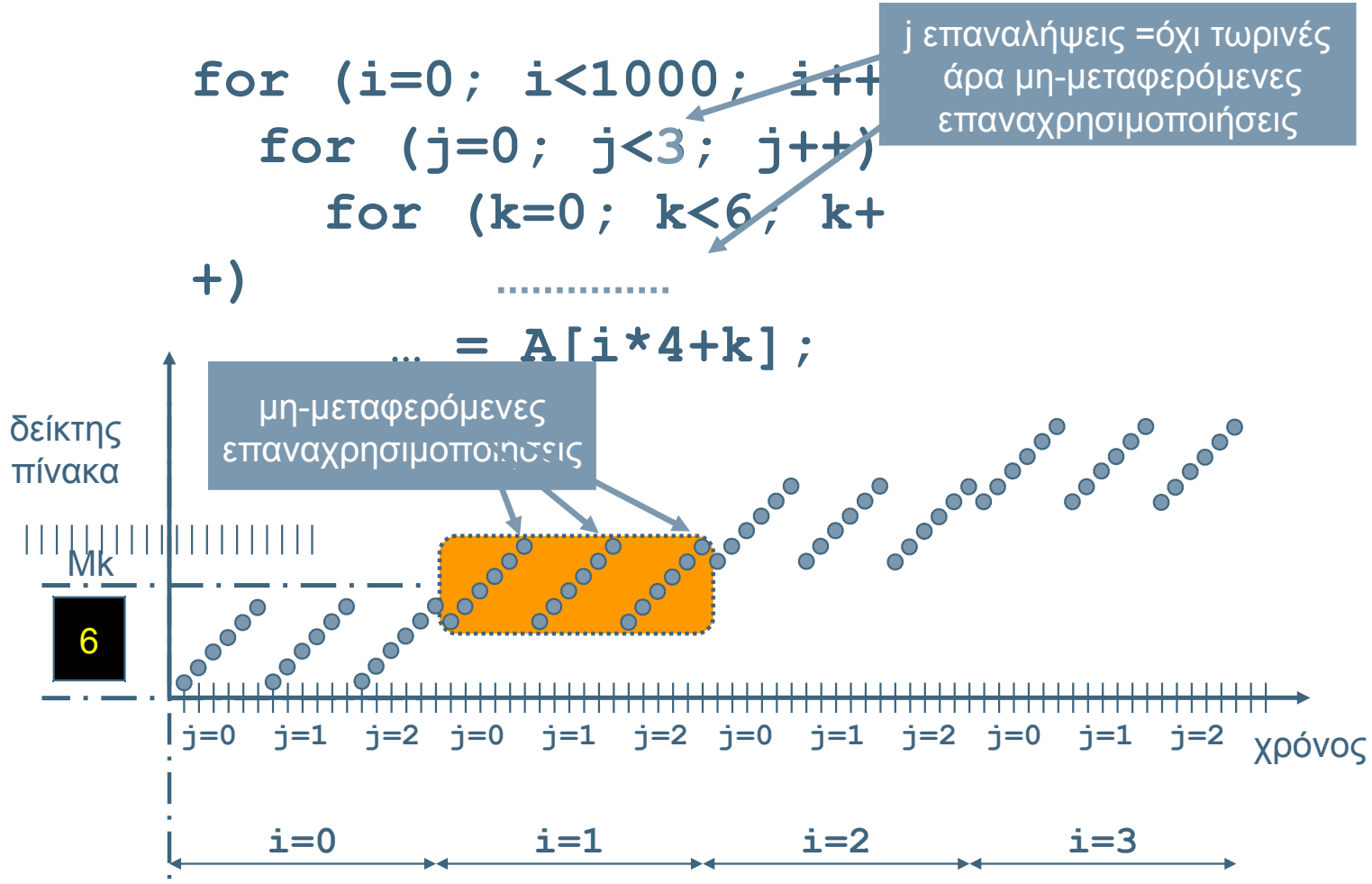
```
for (i=0; i<1000; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = A[i*4+k];
```

Πως θα βρεθούν τα δεδομένα που πρέπει να αντιγραφούν; (2/2)

```
for (i=0; i<1000; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = A[i*4+k];
```

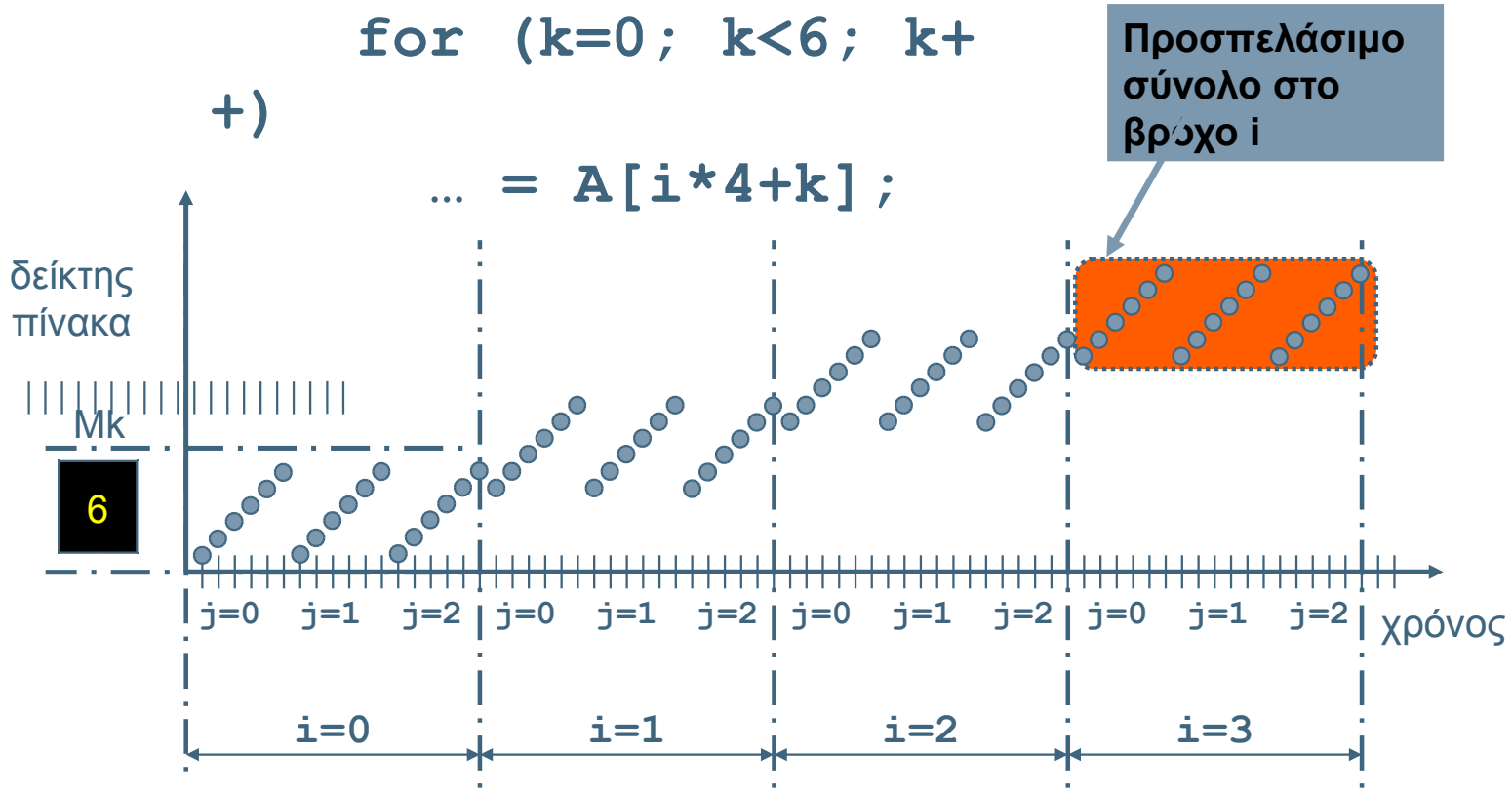


Μη-μεταφερόμενες επαναχρησιμοποιήσεις εντός μιας επανάληψης



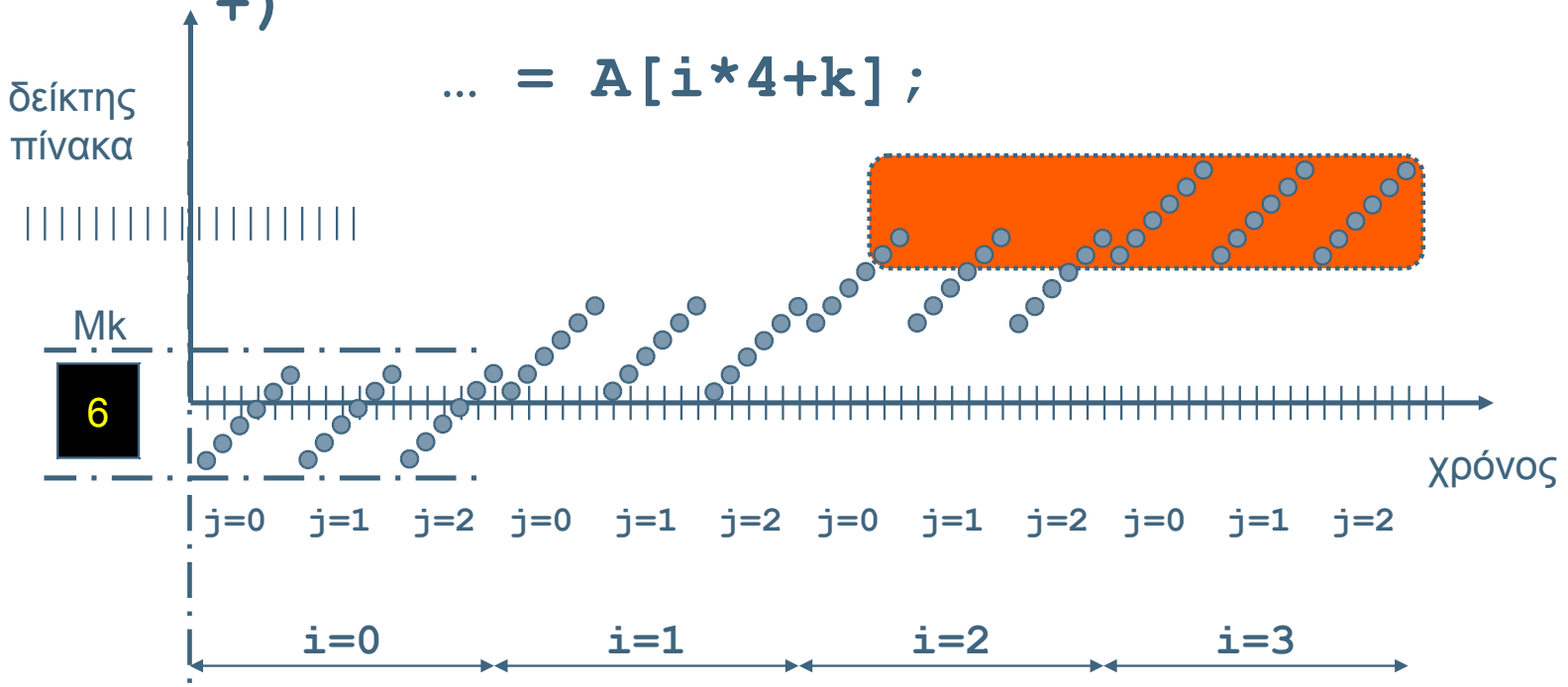
Προσπελάσιμο σύνολο στο βρόχο i

```
for (i=0; i<n; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k+
+)
        … = A[i*4+k];
```



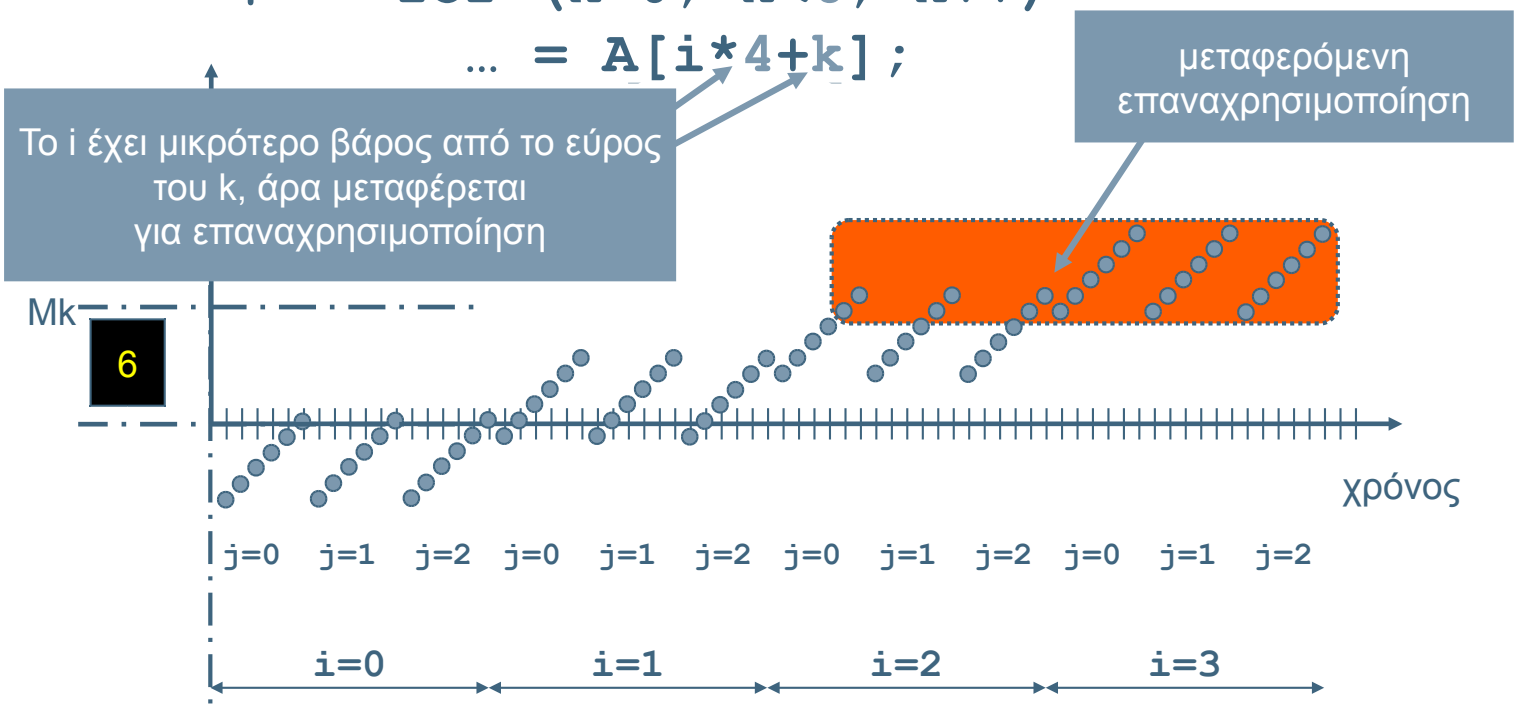
Μεταφερόμενες επαναχρησιμοποιήσεις μεταξύ διαφορετικών επαναλήψεων (1/2)

```
for (i=0; i<n; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++
+)
          … = A[i*4+k];
```



Μεταφερόμενες επαναχρησιμοποιήσεις μεταξύ διαφορετικών επαναλήψεων (2/2)

```
for (i=0; i<n; i++)
 for (i=0; i<1000; i++)
   for (j=0; j<3; j++)
+    for (k=0; k<6; k++)
       … = A[i*4+k];
```



Πως υλοποιείται ένα αντίγραφο; (1/3)

```
for (i=0; i<1000; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = A[i*4+k];
```

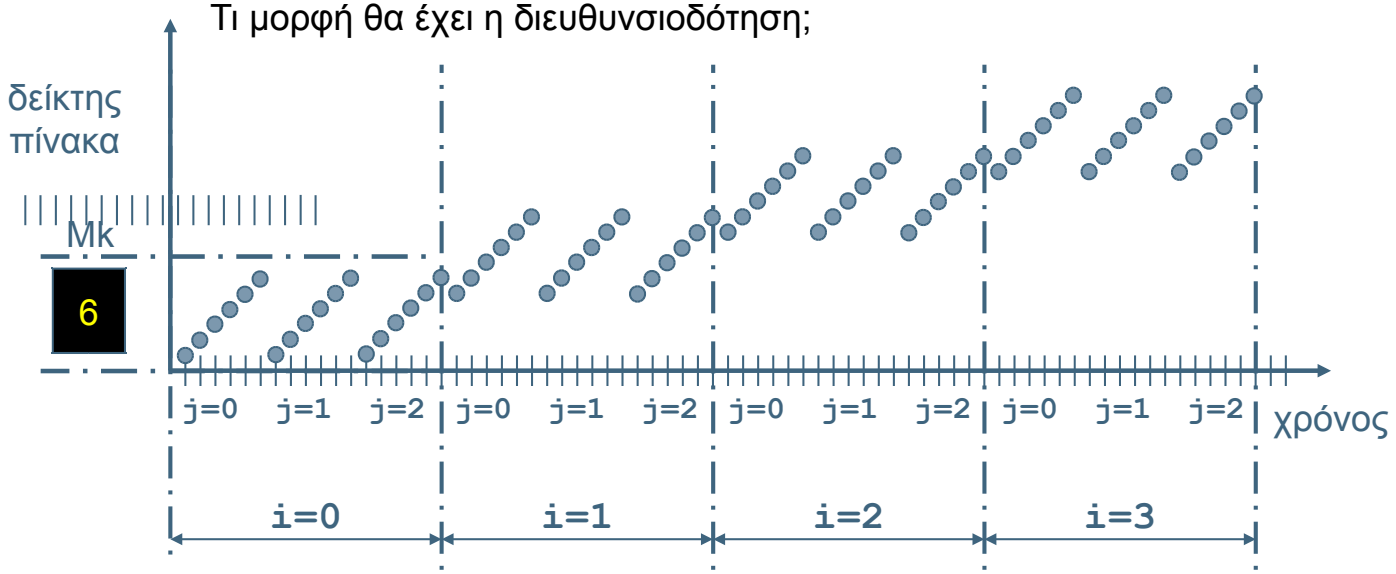
- Πώς θα γεμίσουμε το αντίγραφο με τα σωστά δεδομένα;
- Τι μορφή θα έχει η διευθυνσιοδότηση;

Πως υλοποιείται ένα αντίγραφο; (2/3)

```
for (i=0; i<n; i++)
  for (j=0; j<3; j++)
    for (k=0; k<6; k+
+)
        … = A[i*4+k];
```

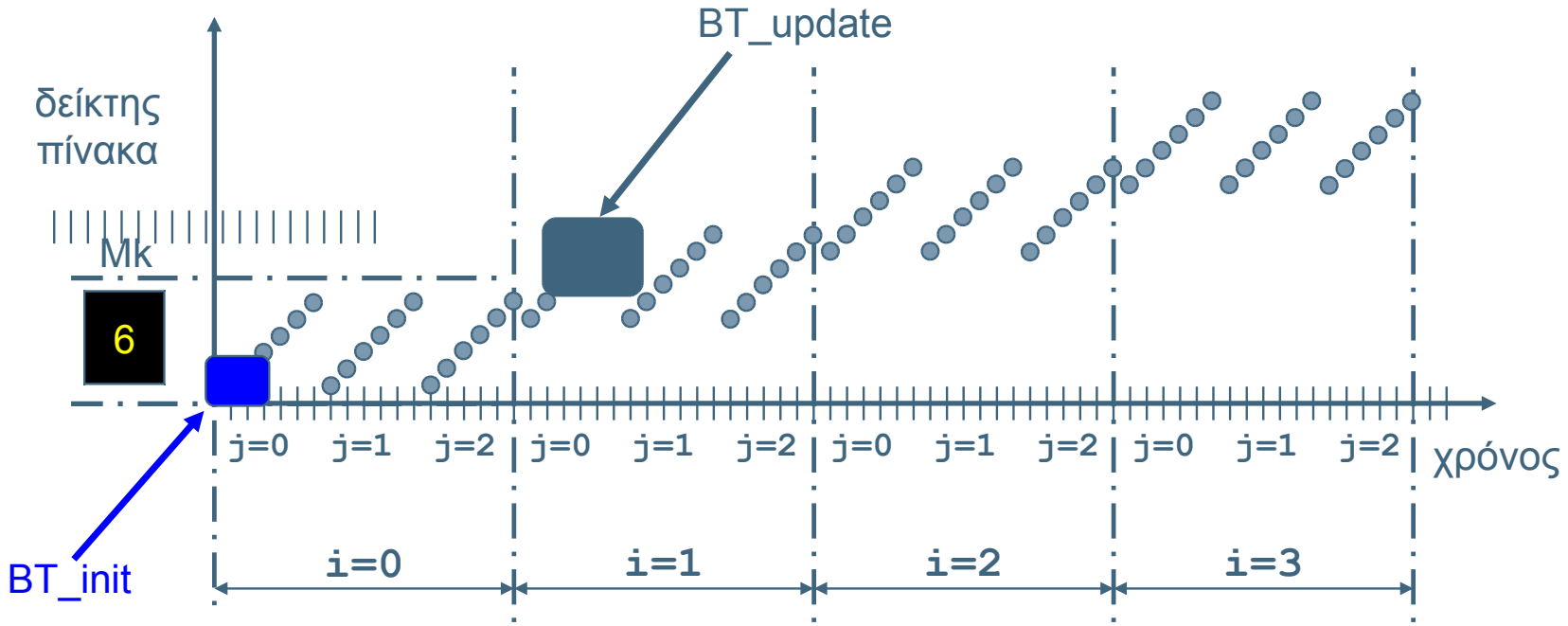
Πώς θα γεμίσουμε το αντίγραφο με τα σωστά δεδομένα;

Τι μορφή θα έχει η διευθυνσιοδότηση;



Πως υλοποιείται ένα αντίγραφο; (3/3)

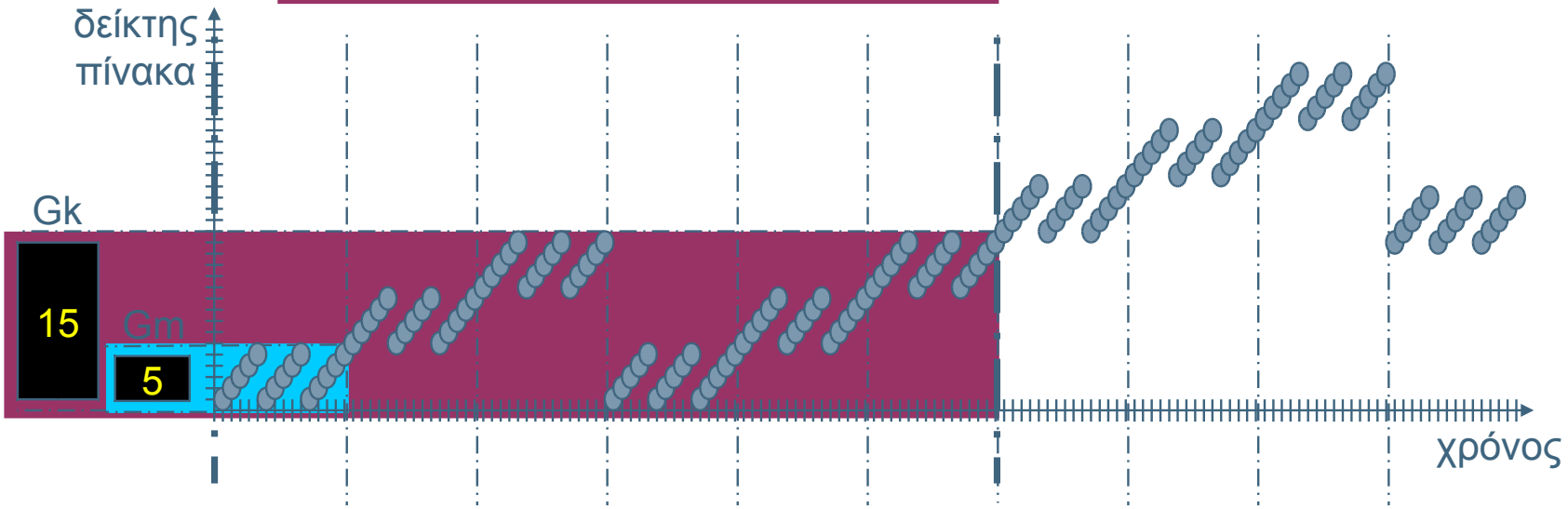
```
BT_init(A[0..1] → a[(0..1)%6]);
for (i=0; i<n; i++)
  BT_update(A[i*4+2..5] → a[(i*4+2..5)%6]);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];
```



Δυνατότητα ιεραρχίας πολλαπλών επιπέδων

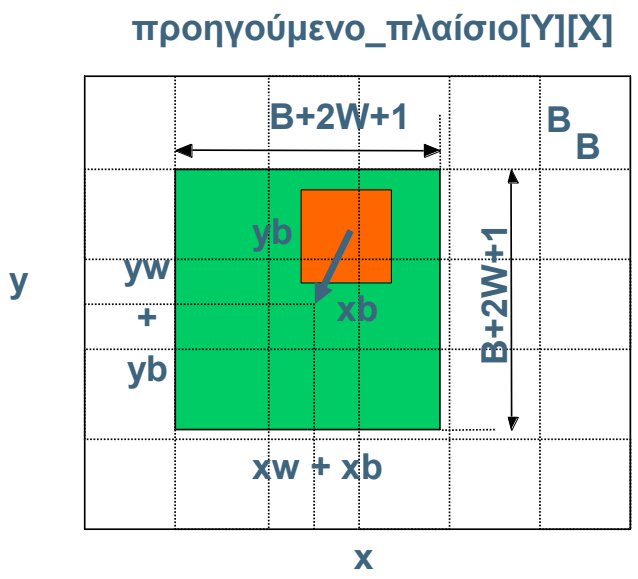
```
for (i=0; i<10; i++)
  for (j=0; j<2; j++)
    for (k=0; k<3; k++)
      for (l=0; l<3; l++)
        for (m=0; m<5; m++)
          … = A[i*15+k*5+m];
```

Επίσης τα αντίγραφα χωρίς επαναχρησιμοποίηση δεδομένων (προμετάκληση)

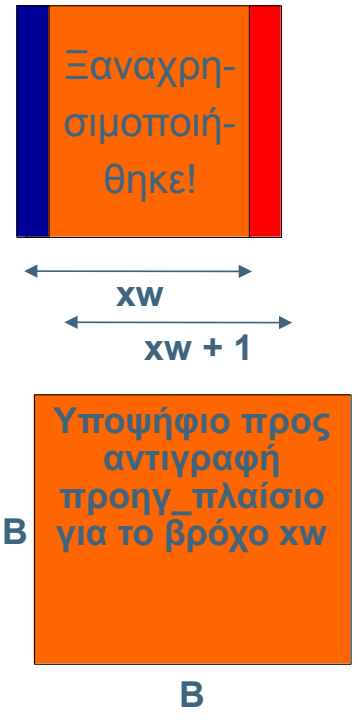


Ένα παράδειγμα …

Επαναχρησιμοποίηση δεδομένων για εκτίμηση πλήρους αναζήτησης κίνησης.



Επαναχρησιμοποίηση κατά την xw επανάληψη στο εύρος xb:



Παράθυρο Αναφοράς

Καλύτερο ταίριασμα του block στο πλαίσιο

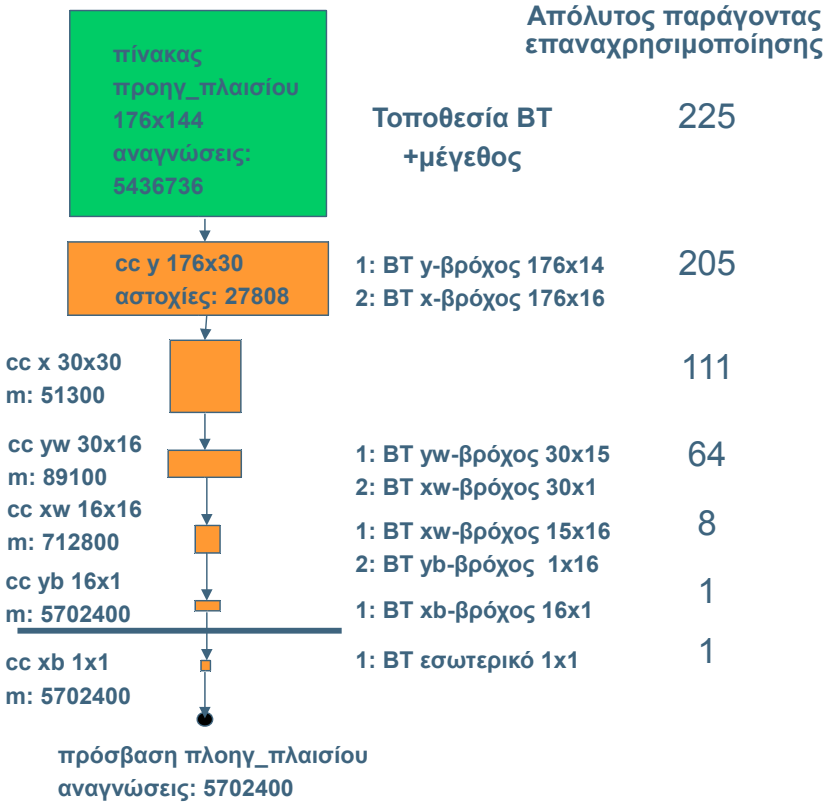
Διάνυσμα Κίνησης

Υποψήφια προς αντιγραφή και μεταφορές block για όλα τα επίπεδα

Για κάθε βρόχο και κάθε πίνακα, υπάρχουν CCs:

- με επαναχρησιμοποίηση.
  - Εκμετάλλευση τοπικότητας χρόνου.
- σε blocks.
  - Εκμετάλλευση τοπικότητας χώρου.

Κάθε CC έχει 1 ή περισσότερες μεταφορές block στο επίπεδο βρόχου CC ή βαθύτερα.



Και τι γίνεται με το χρόνο;

```
for (i=0; i<n; i++) {

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = B[j+k];

  BT update(A[i*4+2..5]
     → a[(i*4+2..5)%6]);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];

}
```

επεξεργασία: 18 κύκλοι

block μεταφοράς: 24 κύκλοι

επεξεργασία: 18 κύκλοι

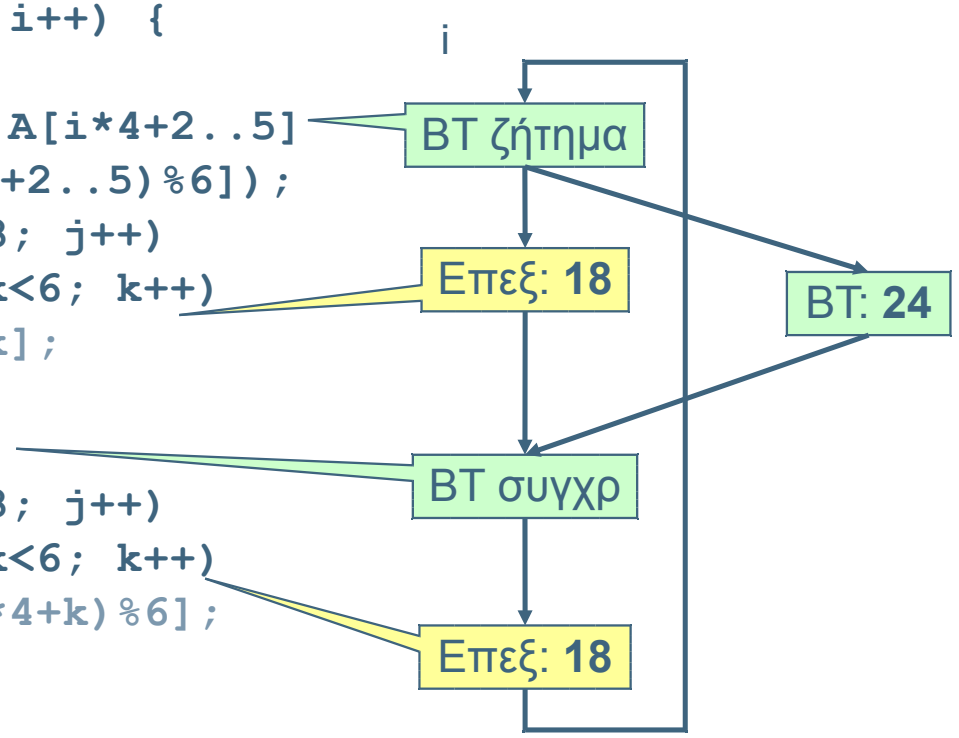
Συνολικός χρόνος: n*(18+24+18)=n*60

Πως μπορούμε να αποφύγουμε την αναμονή μεταφοράς block έως ότου τελειώσει;

Περίπτωση 1: απλή επέκταση χρόνου (1/2)

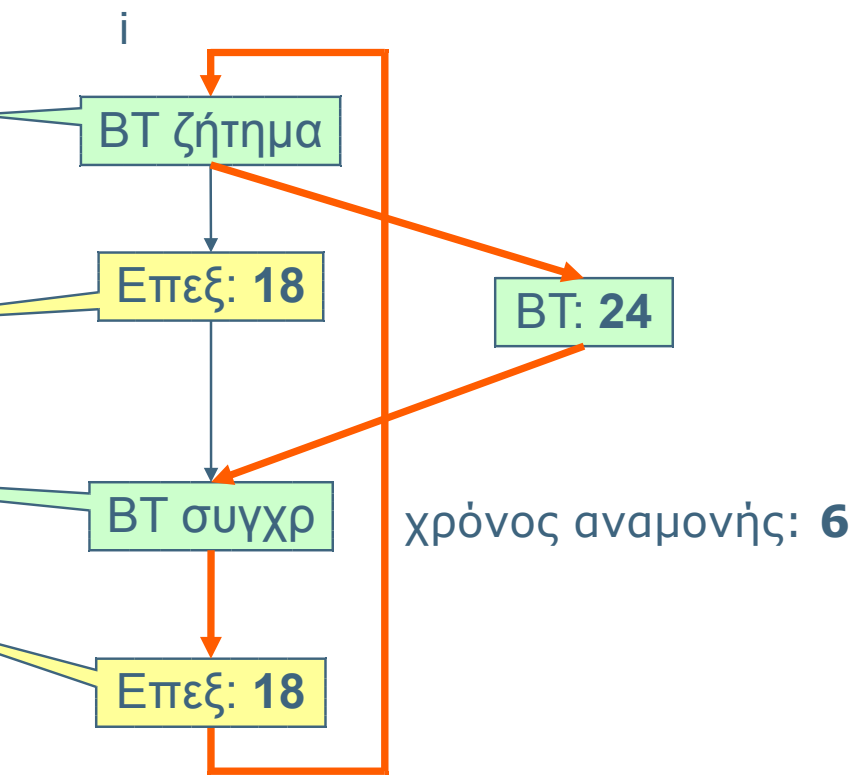
Οι μεταφορές των block μπορούν να εκτελούνται παράλληλα με την επεξεργασία.

```
for (i=0; i<n; i++) {

  BT_issue(id, A[i*4+2..5]
      → a[(i*4+2..5)%6]);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = B[j+k];

  BT_sync(id);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];

  }
```



Περίπτωση 1: απλή επέκταση χρόνου (2/2)

```
for (i=0; i<n; i++) {

  BT_issue(id, A[i*4+2..5]
      → a[(i*4+2..5)%6]);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = B[j+k];

  BT_sync(id);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];

  }
```



Συνολικός χρόνος για την κρίσιμη διαδρομή:
n*(24+18)=n*42

Πώς θα βελτιστοποιηθεί ο χρόνος σε αυτή την περίπτωση;

```
for (i=0; i<n; i++) {

  BT update(A[i*4+2..5]
     → a[(i*4+2..5)%6]);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = B[j+k];

}
```

BT ζήτημα: (24 κύκλοι)

Επεξεργασία: (18 κύκλοι)

Επεξεργασία: (18 κύκλοι)

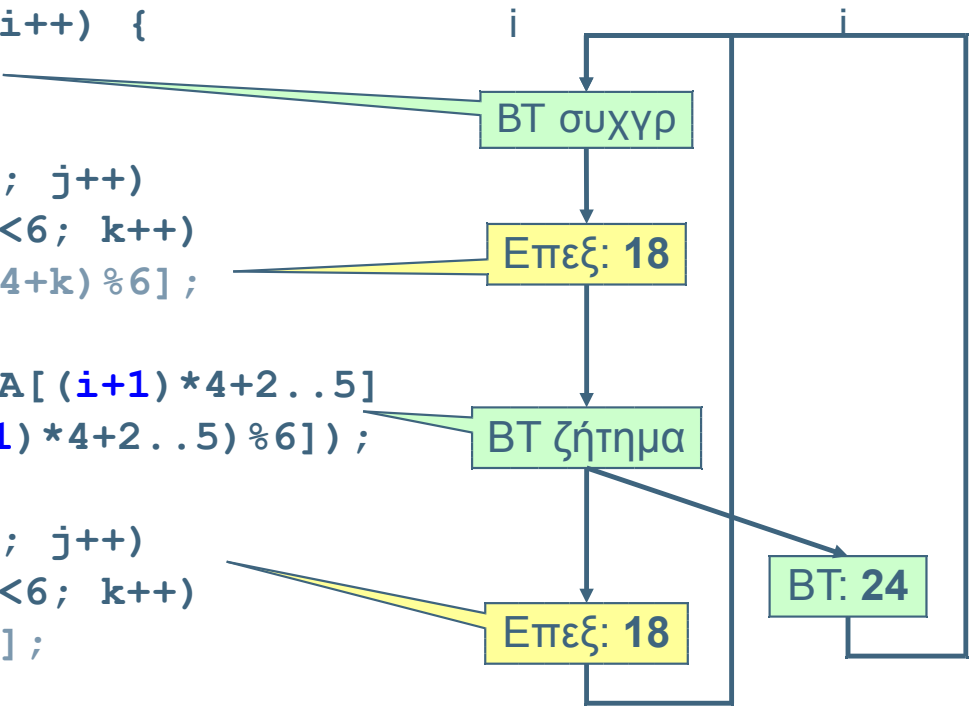
Συνολικός χρόνος: n*(18+24+18)=n*60

Πως μπορούμε να αποφύγουμε την αναμονή μεταφοράς block έως ότου τελειώσει;

Περίπτωση 2: διασωλήνωση (1/2)

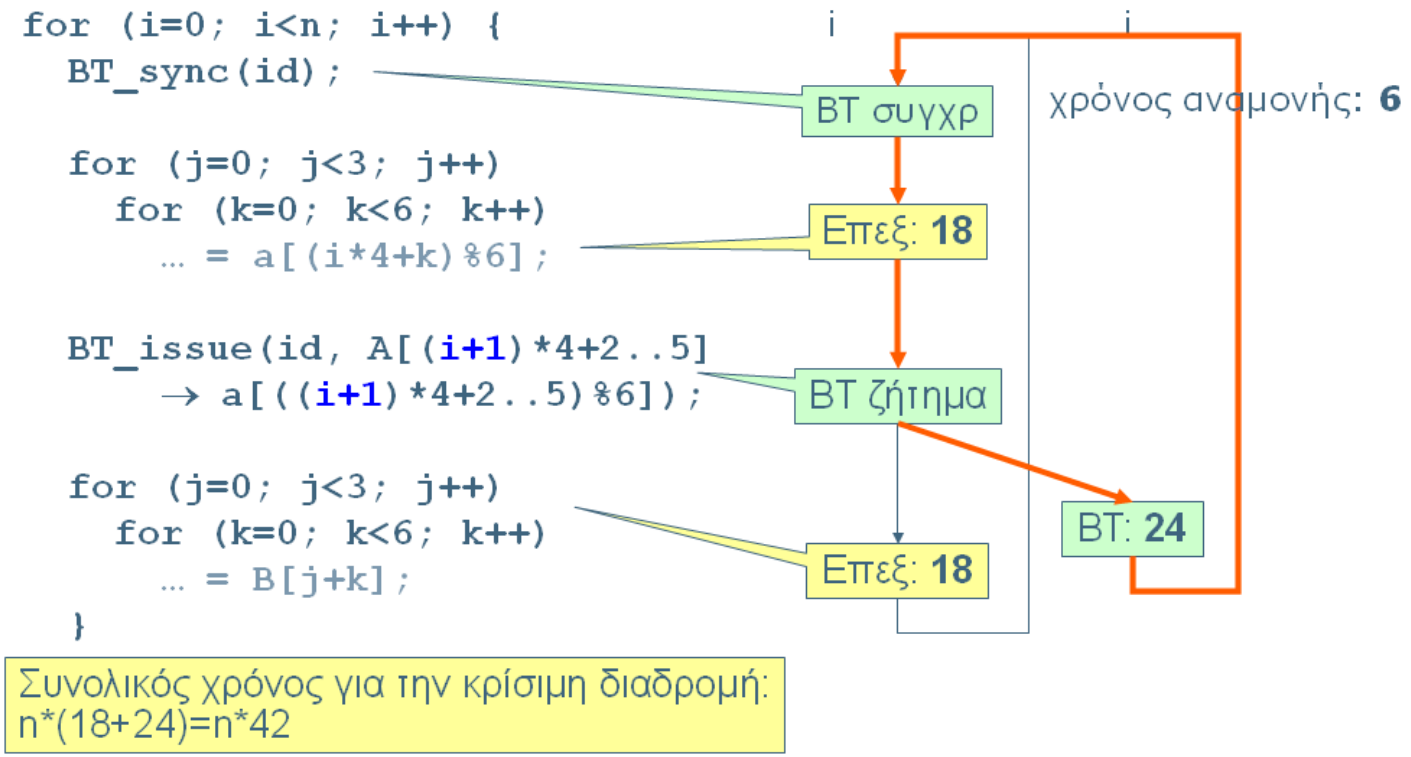
Οι μεταφορές των block μπορούν να εκτελούνται παράλληλα με την επεξεργασία

```
for (i=0; i<n; i++) {
  BT_sync(id);

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];

  BT_issue(id, A[(i+1)*4+2..5]
      → a[((i+1)*4+2..5)%6]);

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = B[j+k];
  }
```



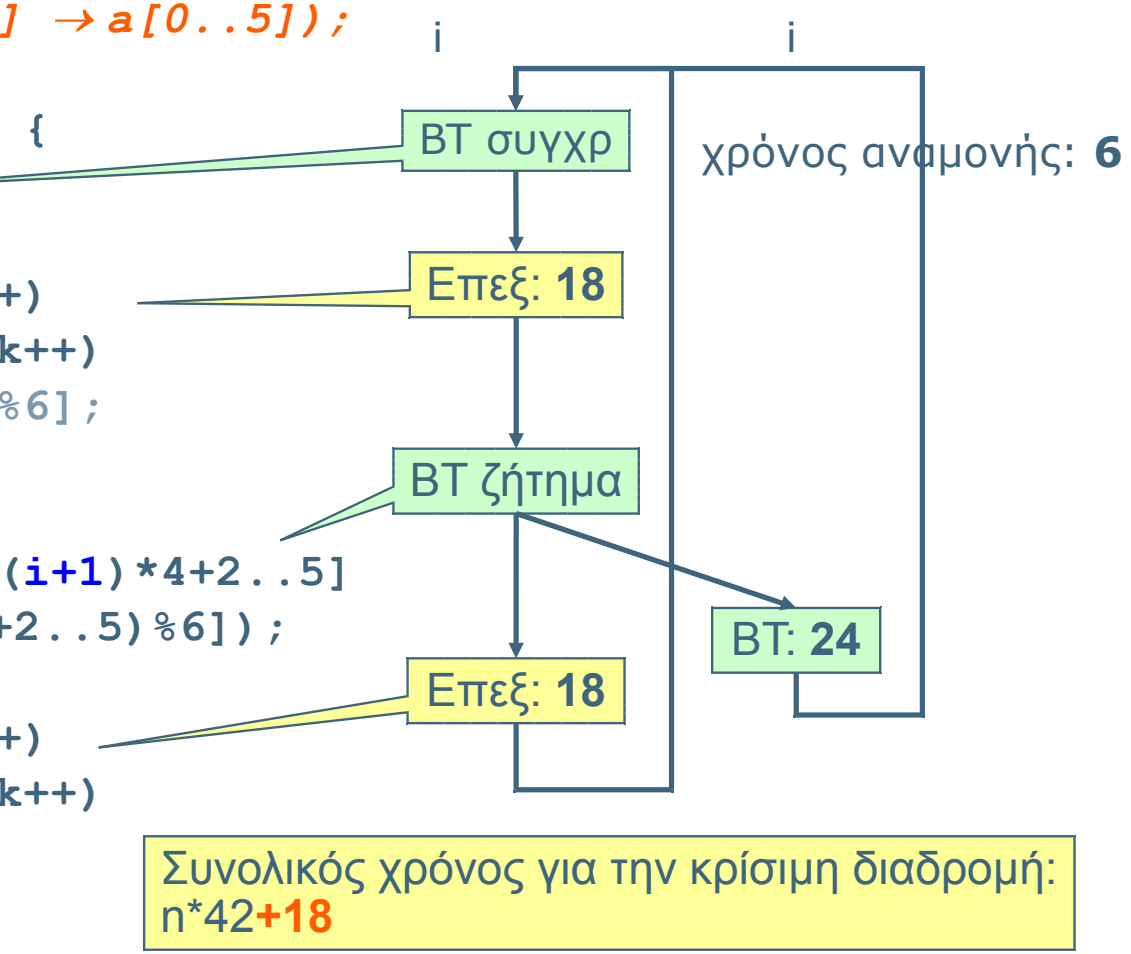
Περίπτωση 2: διασωλήνωση (2/2)

Οι μεταφορές των block μπορούν να εκτελούνται παράλληλα με την επεξεργασία



Η διασωλήνωση απαιτεί προοίμιο & παρελκόμενο τμήμα κώδικα

```
BT_issue(id, A[0..5] → a[0..5]);

for (i=0; i<n; i++) {
  BT_sync(id);

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];

  if (i+1<n)
    BT_issue(id, A[(i+1)*4+2..5]
      → a[((i+1)*4+2..5)%6]);

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = B[j+k];
}
```



Τι γίνεται σε αυτή την περίπτωση;

```
for (i=0; i<n; i++) {

  BT update(A[i*4+2..5]
     → a[(i*4+2..5)%6]);
  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%6];

  }
```

BT ζήτημα: (24 κύκλοι)

Επεξεργασία: (18 κύκλοι)

Συνολικός χρόνος: n*(24+18)=n*42

Τι θα συμβεί αν δεν υπάρχουν άλλοι πυρήνες,
με τους οποίους θα μπορούσε να παραλληλιστεί;

Περίπτωση 3: παράλληλη διασωλήνωση (1/3)

Οι μεταφορές των block μπορούν να εκτελούνται παράλληλα με την επεξεργασία.

```
for (i=0; i<n; i++) {

  BT_sync(id);

  BT issue(id, A[(i+1)*4+2..5]
    → a[((i+1)*4+2..5)%10]);

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%10];

  }
```

BT συχρ: χρ.αναμονής 24-18=6 κύκλοι

BT ζήτημα: (24 κύκλοι) BT // επεξεργασία

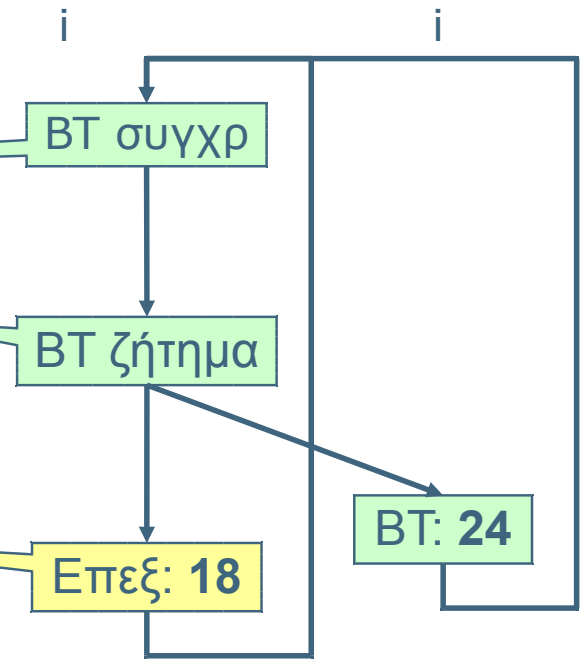
Επεξεργασία: (18 κύκλοι)

Συνολικός χρόνος: n*(6+18)=n*24

ΤΟ ΜΕΓΕΘΟΣ ΑΥΞΑΝΕΤΑΙ !

Περίπτωση 3: παράλληλη διασωλήνωση (2/3)

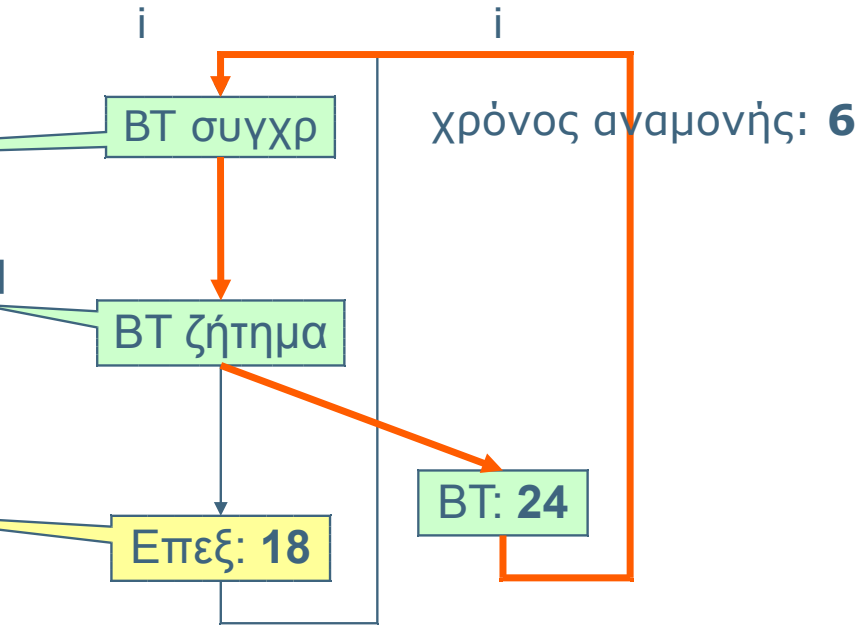
```
for (i=0; i<n; i++) {

  BT_sync(id);

  BT issue(id, A[(i+1)*4+2..5]
    → a[((i+1)*4+2..5)%10]);

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%10];

  }
```



ΤΟ ΜΕΓΕΘΟΣ ΑΥΞΑΝΕΤΑΙ!

Περίπτωση 3: παράλληλη διασωλήνωση (3/3)

```
for (i=0; i<n; i++) {

  BT_sync(id);

  BT issue(id, A[(i+1)*4+2..5]
    → a[((i+1)*4+2..5)%10]);

  for (j=0; j<3; j++)
    for (k=0; k<6; k++)
      … = a[(i*4+k)%10];

  }
```



Συνολικός χρόνος για την κρίσιμη διαδρομή: n*24

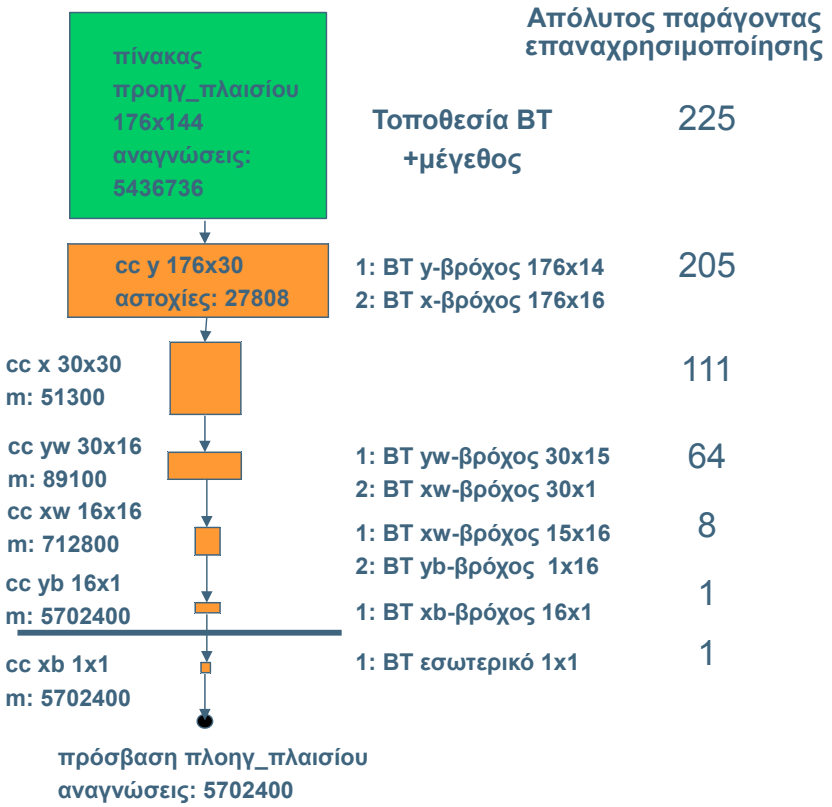
ΤΟ ΜΕΓΕΘΟΣ ΑΥΞΑΝΕΤΑΙ!

Υποψήφια προς αντιγραφή και μεταφορά block για όλα τα επίπεδα

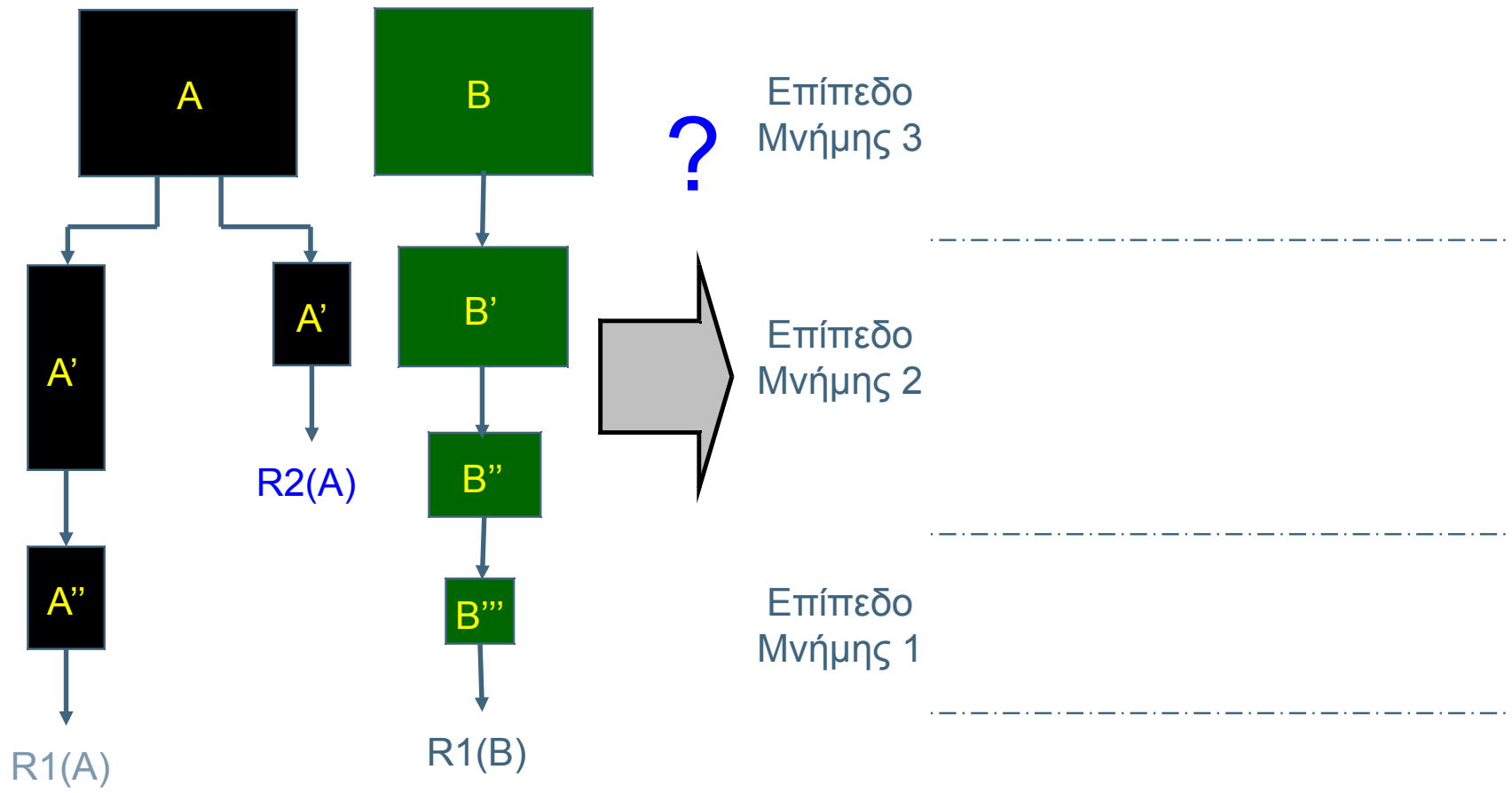
Για κάθε βρόχο και κάθε πίνακα, υπάρχουν CCs:

- με επαναχρησιμοποίηση.
  - Εκμετάλλευση τοπικότητας χρόνου.
- σε blocks.
  - Εκμετάλλευση τοπικότητας χώρου.

Κάθε CC έχει 1 ή περισσότερες μεταφορές block στο επίπεδο βρόχου CC ή βαθύτερα.

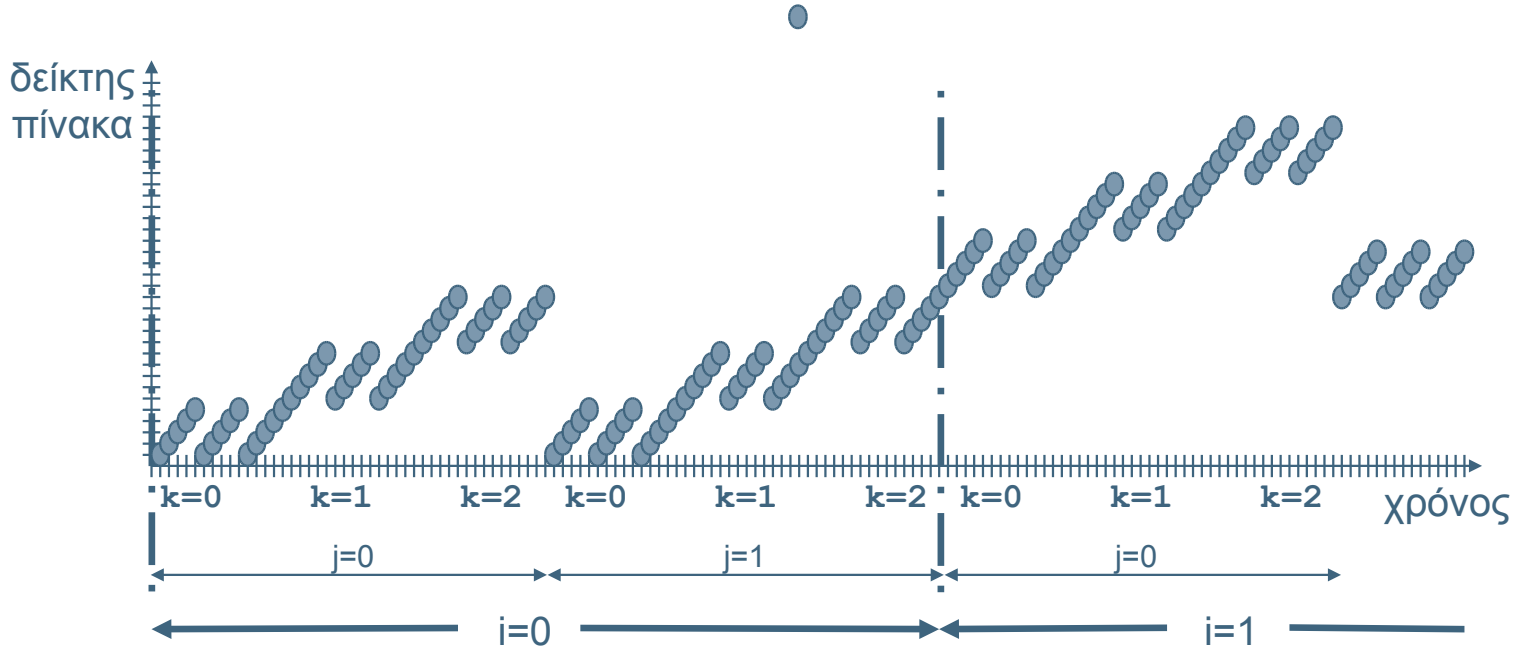


Πώς επιλέγουμε τα υποψήφια αντίγραφα; (1/2)



Πώς επιλέγουμε τα υποψήφια αντίγραφα; (2/2)

```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++)
     … = A[i*15+k*5+m];
```

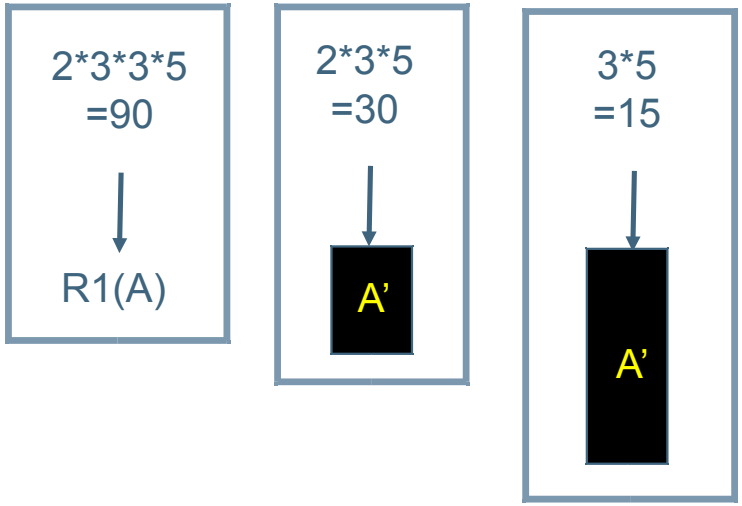


Η συνάρτηση κόστους

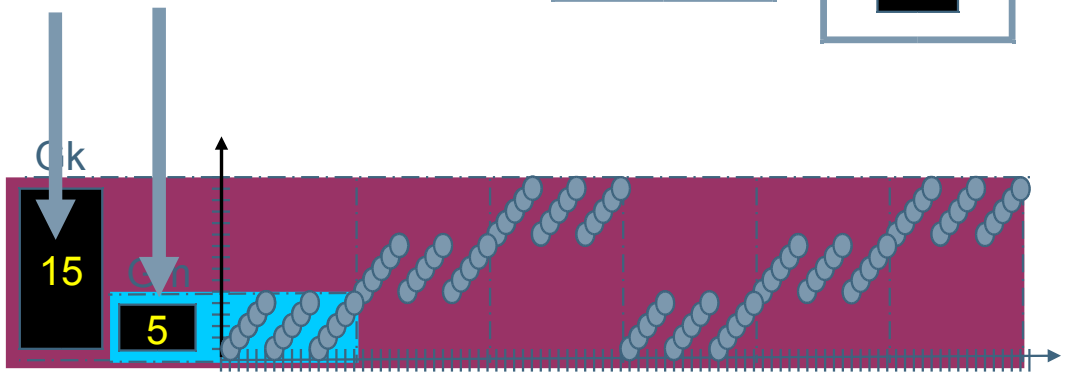
Η συνάρτηση κόστους χρειάζεται τόσο το μέγεθος όσο και τον αριθμό προσβάσεων για τους ενδιάμεσους πίνακες.

```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++)
     … = A[i*15+k*5+m];
```

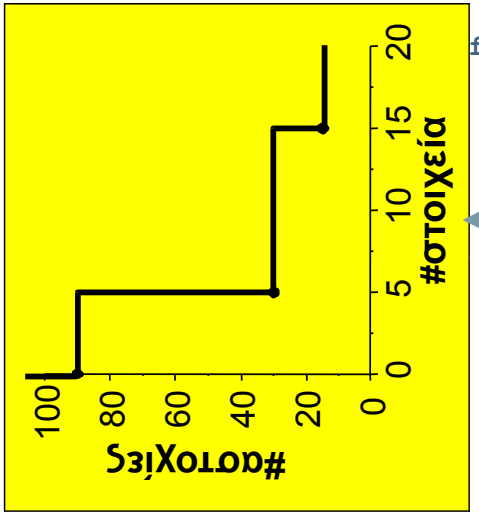
εκτίμηση #αστοχιών από διαφορετικά επίπεδα για μια επανάληψη του i



εκτίμηση μεγέθους

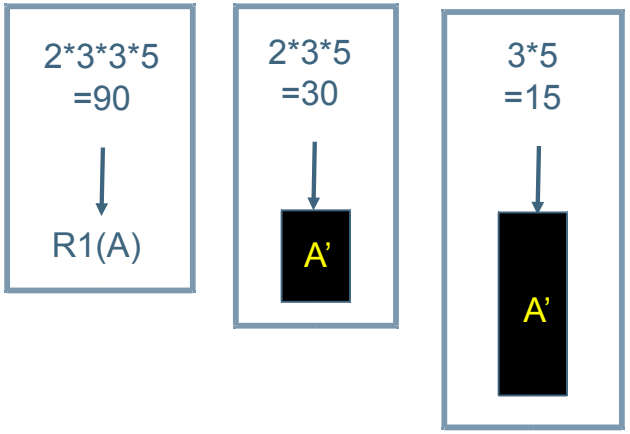


Η συνάρτηση κόστους απαιτεί τόσο το μέγεθος όσο και τον αριθμό προσβάσεων για τους ενδιάμεσους πίνακες

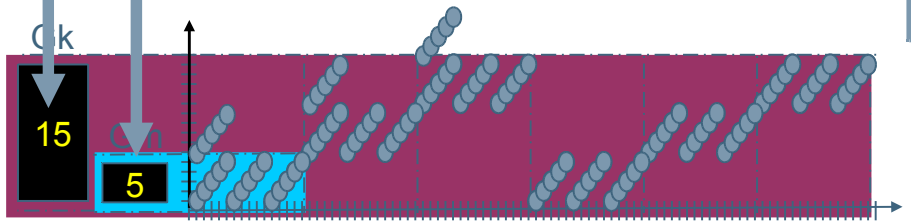



```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++)
     … = A[i*15+k*5+m];
```

εκτίμηση #αστοχιών από διαφορετικά επίπεδα για μια επανάληψη του i



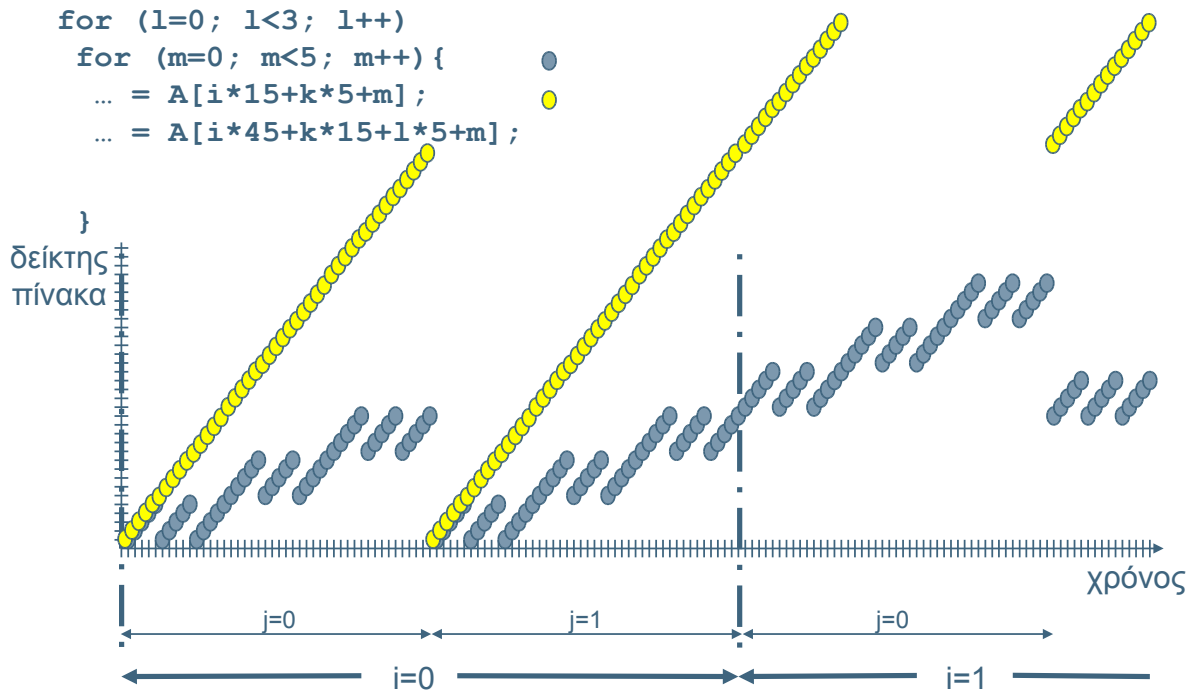
εκτίμηση μεγέθους



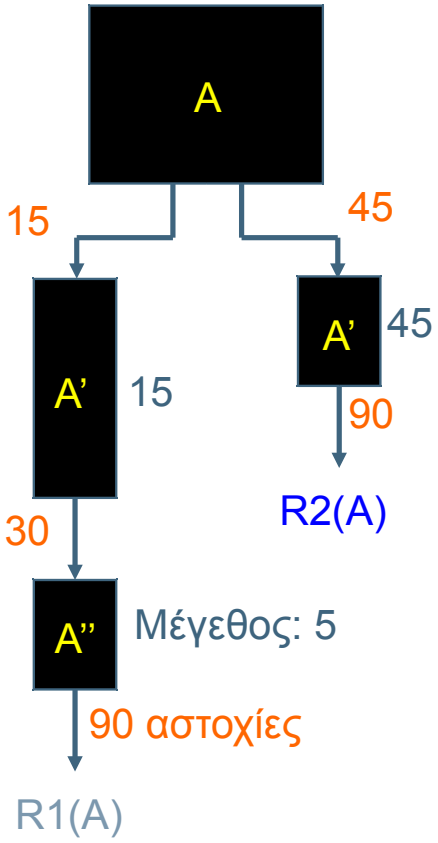
Αυτή η πληροφορία δίνεται από το εργαλείο επαναχρησιμοποίησης δεδομένων

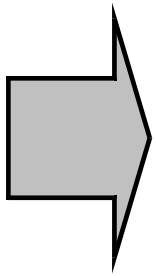
Επιλογή υποψήφιων αντιγράφων (1/4)

```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++){
     … = A[i*15+k*5+m];
     … = A[i*45+k*15+l*5+m];
    }
```



Επιλογή υποψήφιων αντιγράφων (2/4)





?

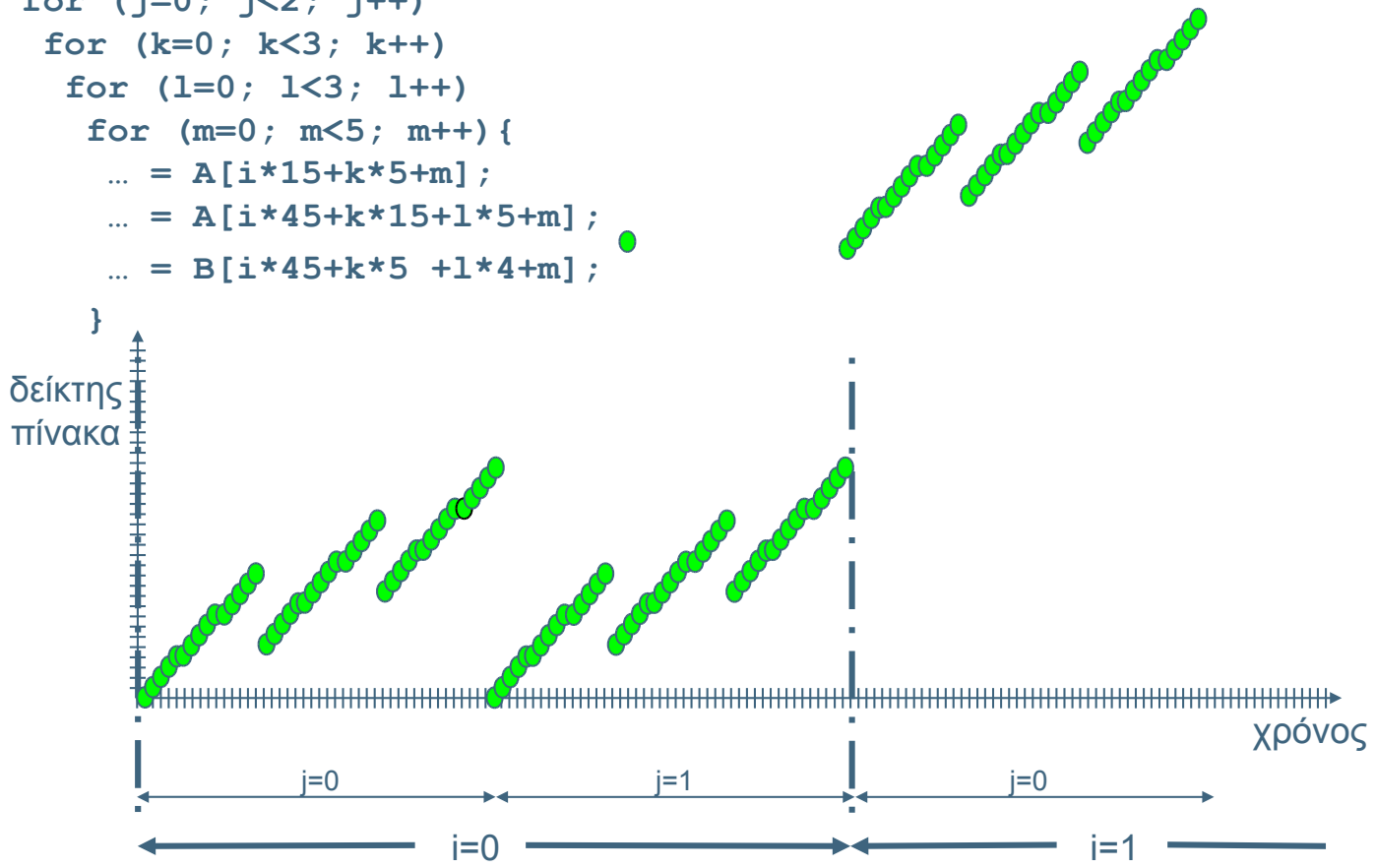
Επίπεδο Μνήμης 3

Επίπεδο Μνήμης 2

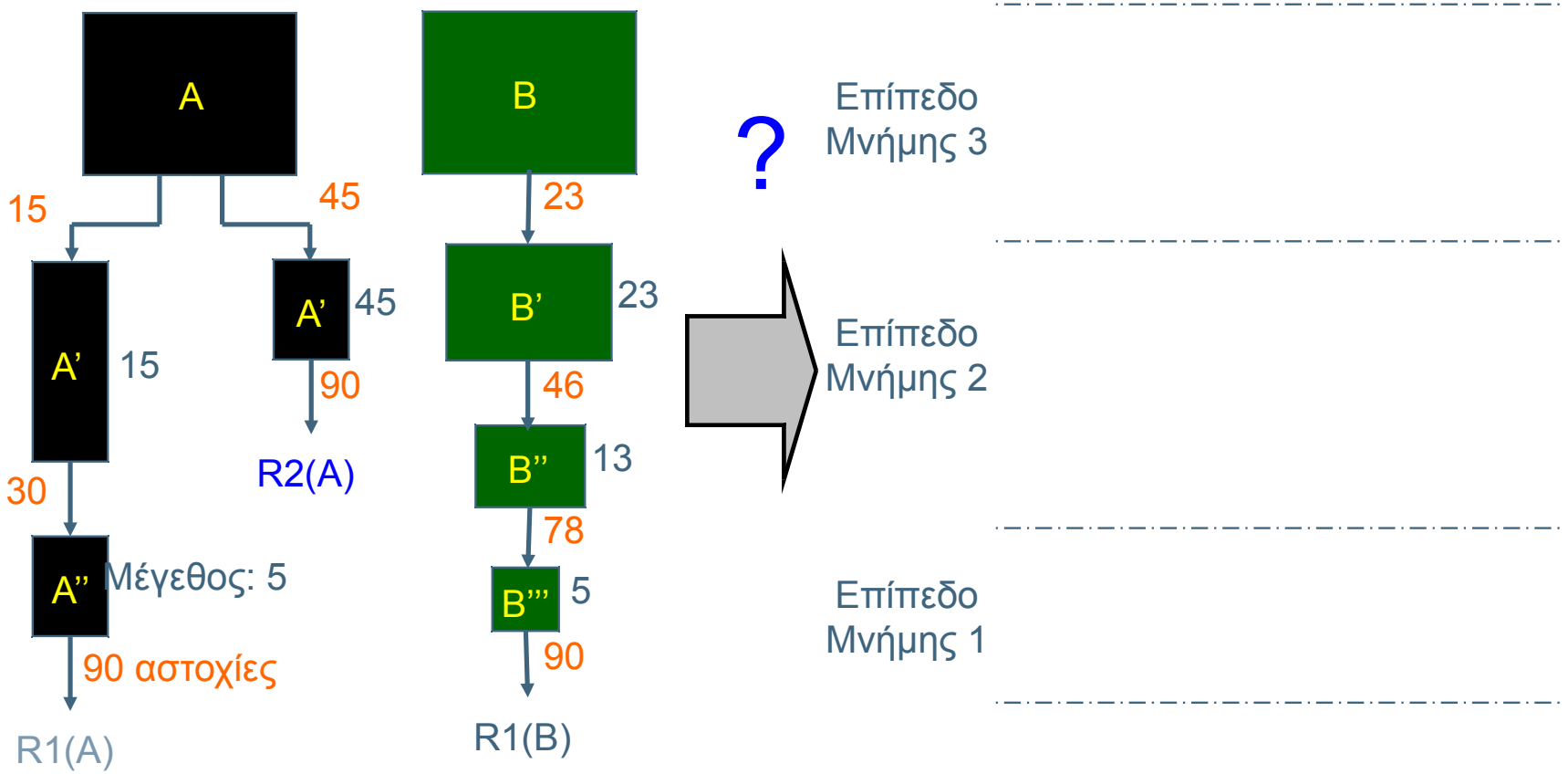
Επίπεδο Μνήμης 1

Επιλογή υποψήφιων αντιγράφων (3/4)

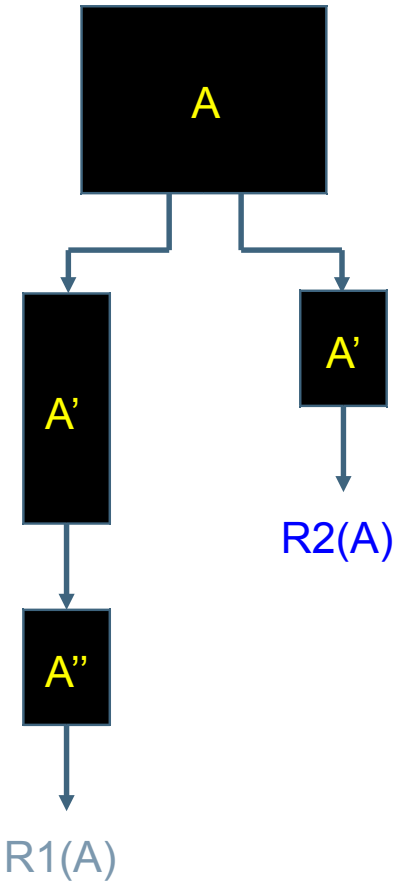
```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++){
     … = A[i*15+k*5+m];
     … = A[i*45+k*15+l*5+m];
     … = B[i*45+k*5 +l*4+m];
    }
```

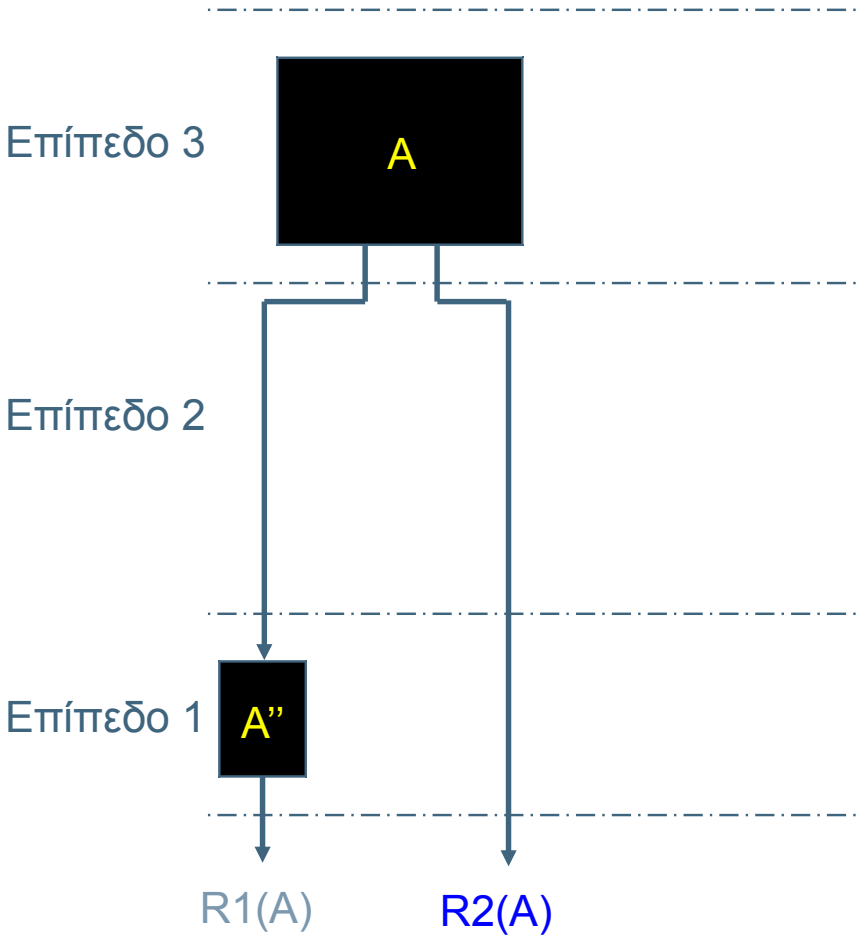


Επιλογή υποψήφιων αντιγράφων (4/4)

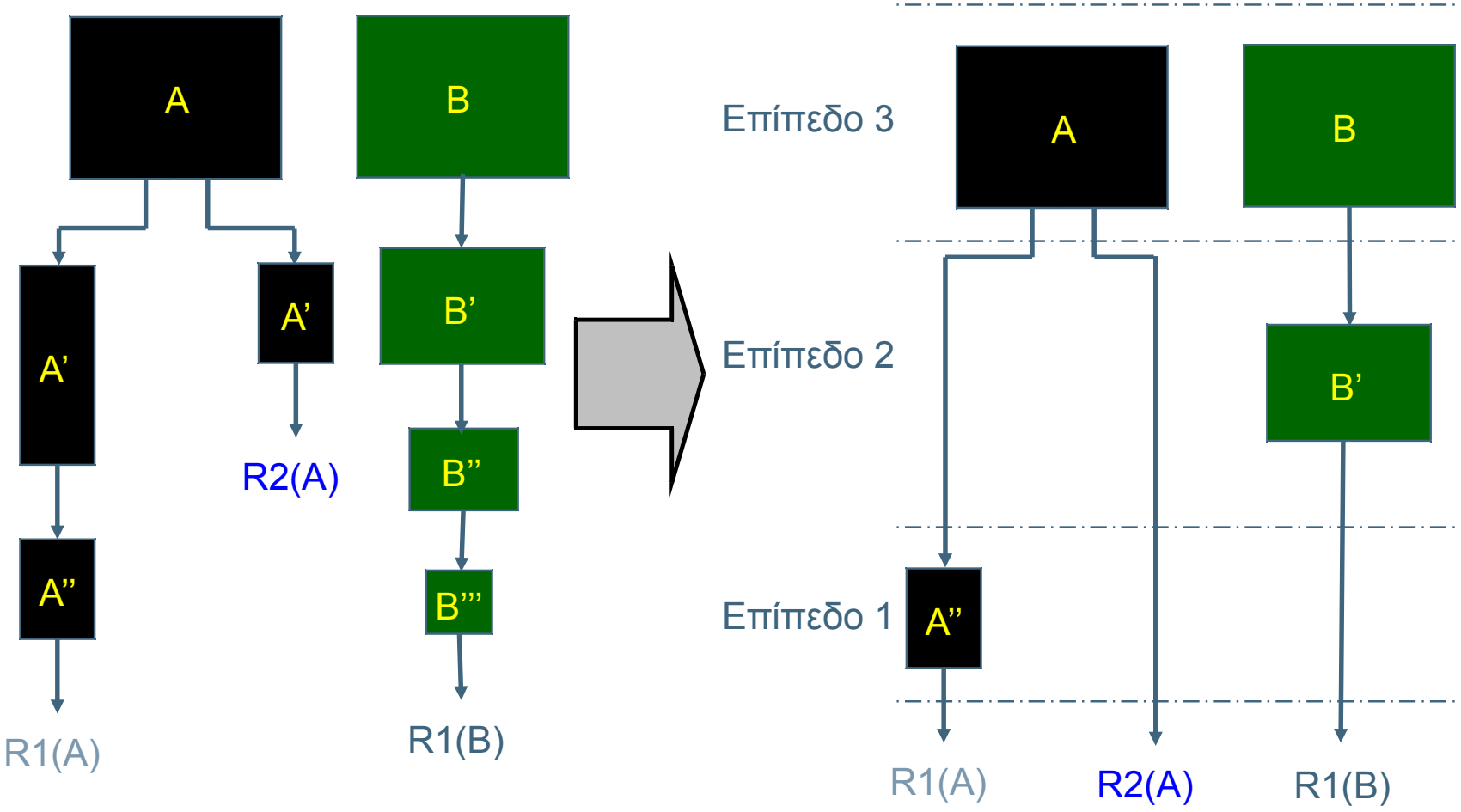


Ανάθεση Επιπέδων Ιεραρχίας Μνήμης (MHLA) Ολική βελτιστοποίηση ενέργειας και χρόνου (1/3)

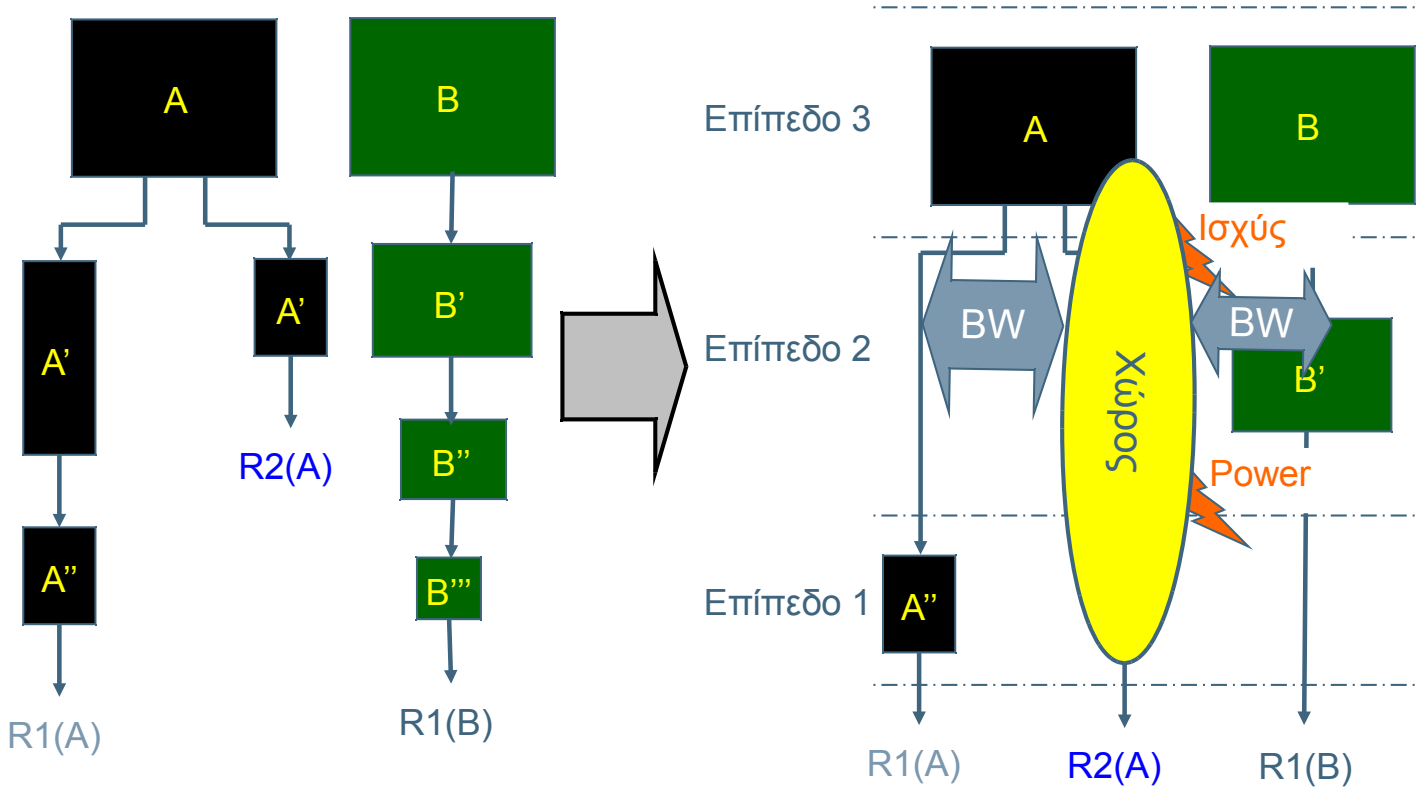




Ανάθεση Επιπέδων Ιεραρχίας Μνήμης (MHLA) Ολική βελτιστοποίηση ενέργειας και χρόνου (2/3)



Ανάθεση Επιπέδων Ιεραρχίας Μνήμης (MHLA) Ολική βελτιστοποίηση ενέργειας και χρόνου (3/3)



Περίληψη

- Προσδιορισμός των υποψήφιων αντιγράφων (CCs) για επαναχρησιμοποίηση και προμετάκληση.
- **Συναφή κόστη:** μέγεθος και # αστοχίες.
- Δεδομένης μιας ιεραρχίας μνήμης, η MHLA επιλέγει και αναθέτει τα CC, βελτιστοποιώντας ολικά την κατανάλωση ενέργειας στην ιεραρχία μνήμης και το χρόνο εκτέλεσης.

Τέλος Ενότητας

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης