Προγραμματισμός Ι (ECE115)

#14 αναδρομή

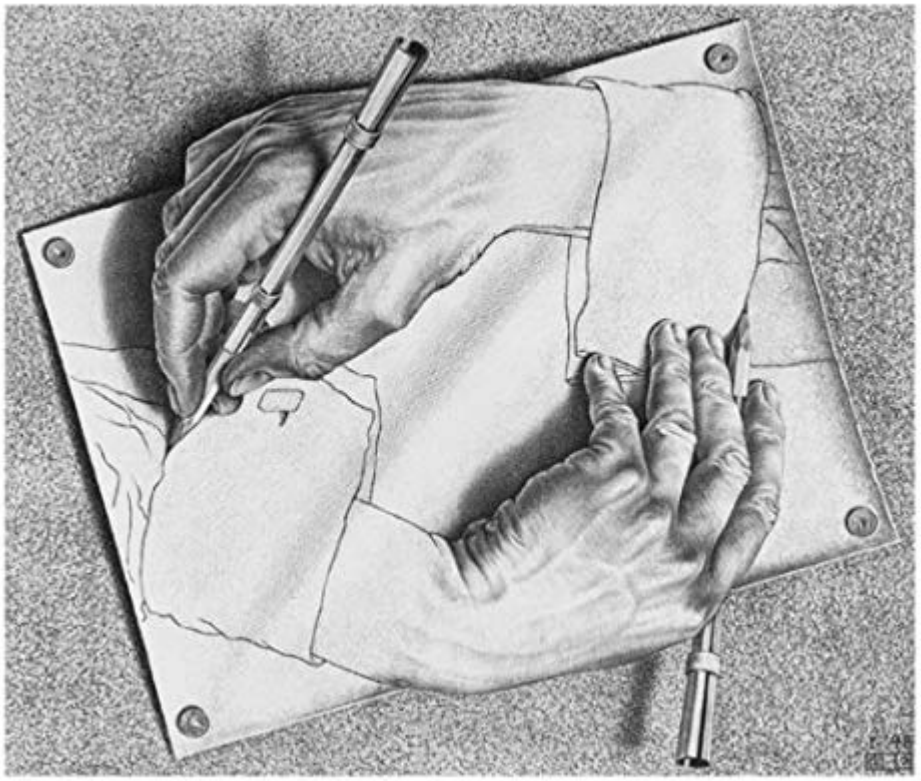
Σπύρος Λάλης
lalis@uth.gr

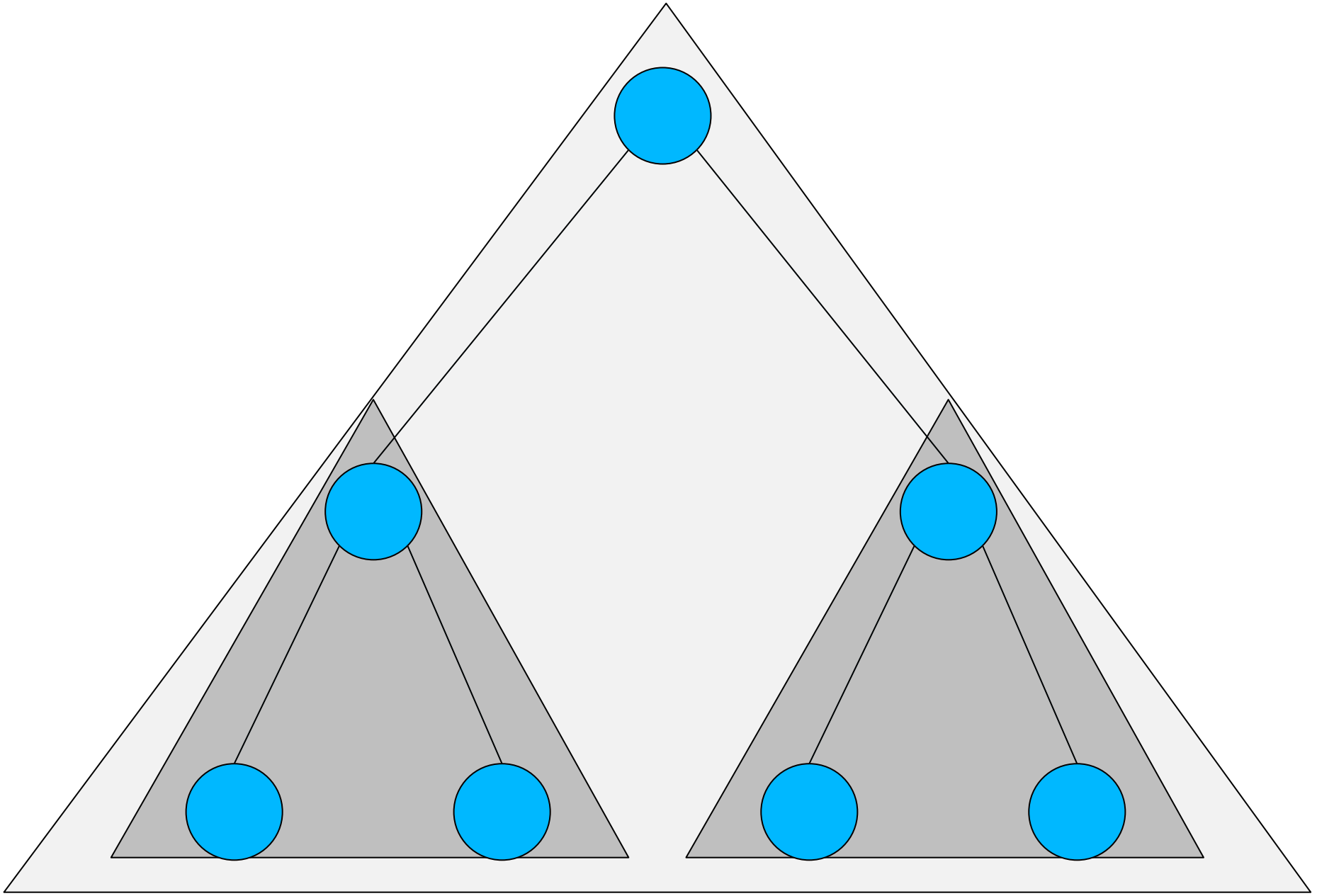
Αναδρομή

- Συγγενής έννοια με την μαθηματική επαγωγή που συνήθως γίνεται «αυξητικά»
  - αρχίζουμε από την απλή περίπτωση (π.χ. 0), υποθέτουμε ότι ισχύει για k, δείχνουμε ότι ισχύει για $k+1$
- Η αναδρομή μπορεί να είναι και «φθίνουσα»
  - ανάγουμε την κατασκευή του πολύπλοκου/γενικού προβλήματος σε μια πιο απλή έκδοση του ίδιου προβλήματος, και σταματάμε στην απλή περίπτωση

- Συγγενής έννοια με την επανάληψη

- Μια αναδρομική διαδικασία «εμπεριέχει», ως βασικό δομικό στοιχείο, τον ίδιο τον **εαυτό** της





Αναδρομικές συναρτήσεις

- Μια συνάρτηση ονομάζεται **αναδρομική** όταν καλεί ... τον **εαυτό** της
  - άμεσα (μέσα από το ίδιο της της σώμα)
  - έμμεσα (μέσα από άλλες συναρτήσεις)
- Μια αναδρομική συνάρτηση πρέπει να **τερματίζει**
  - η ατέρμονη αναδρομή είναι προγραμματιστικό λάθος (αντίστοιχο με αυτό της ατέρμονης επανάληψης)
  - οδηγεί σε τερματισμό του προγράμματος, λόγω υπερχείλισης της στοίβας (stack overflow)
- Η αναδρομή μπορεί να θεωρηθεί σαν μια ειδική τεχνική προγραμματισμού
- Διάφορα (πολύπλοκα) προβλήματα μπορεί να λυθούν με **απλό** τρόπο χρησιμοποιώντας αναδρομή

```c
/* εκτύπωση τιμών από 0 μέχρι n με συμβατικό τρόπο */

#include <stdio.h>

void printInts(int from, int to) {
  int i;

  for(i = from; i <= to; i++) {
    printf("%d ",i);
  }
}

int main(int argc, char *argv[]) {
  int n;

  printf("enter int: ");
  scanf("%d",&n);
  printInts(0,n);

  return(0);
}
```

Η επανάληψη ως αναδρομή

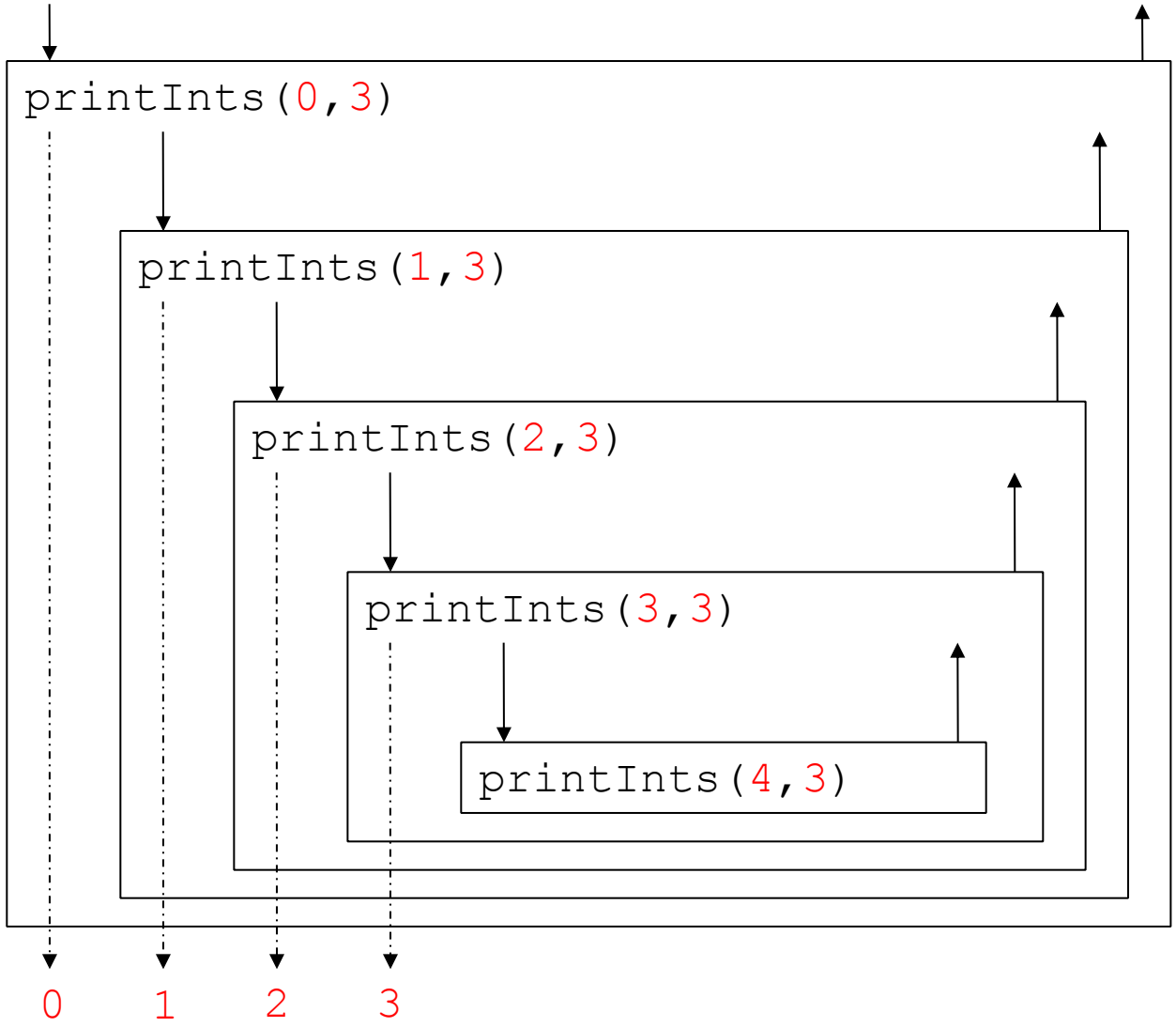
Συμβατικός τρόπος σκέψης

- Τύπωσε τις τιμές `from` ως (με το) `to`
- μέσω ενός μετρητή
- που παίρνει τιμές από `from` ως `to`

Αναδρομικός τρόπος σκέψης

- Αν `from > to` μην τυπώσεις τίποτα
- Αν `from <= to`
  - τύπωσε την τιμή `from`
  - ζήτα να τυπωθούν οι τιμές `from+1` ως `to`

```c
/* εκτύπωση τιμών από 0 μέχρι n με αναδρομικό τρόπο */

#include <stdio.h>

void printInts(int from, int to) {
  if (from <= to) {
    printf("%d ",from);
    printInts(from+1,to);
  }
}

int main(int argc, char *argv[]) {
  int n;

  printf("enter int: ");
  scanf("%d",&n);
  printInts(0,n);

  return(0);
}
```


printInts(0,3)
printInts(1,3)
printInts(2,3)
printInts(3,3)
printInts(4,3)
0
1
2
3

Και τώρα με ανάποδη σειρά

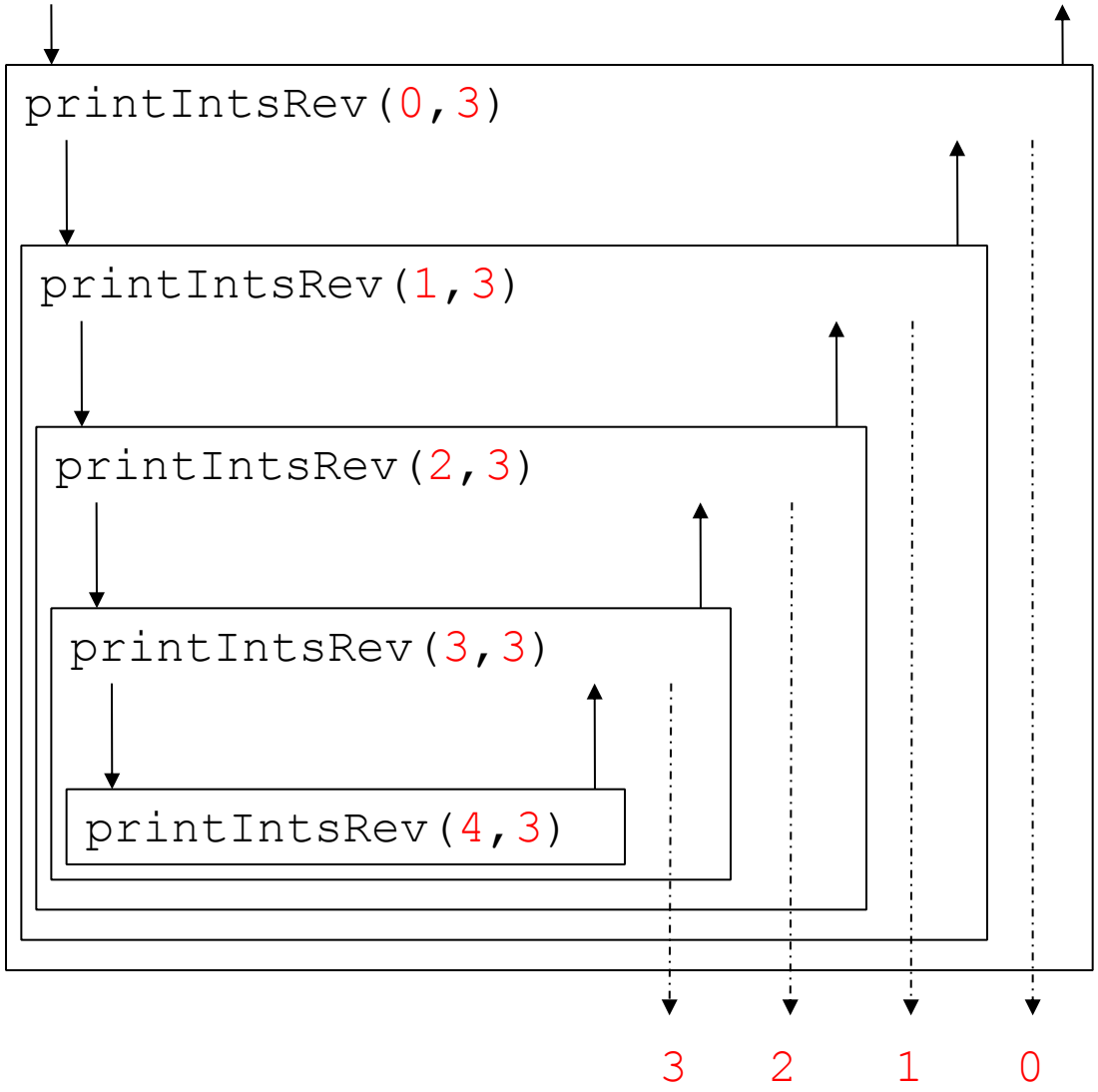
Συμβατικός τρόπος σκέψης

- Τύπωσε τις τιμές `from` ως (με το) `to`
- μέσω ενός μετρητή
- που παίρνει τιμές από `to` ως `from`

Αναδρομικός τρόπος σκέψης

- Αν `from > to` μην τυπώσεις τίποτα
- Αν `from <= to`
  - ζήτα να τυπωθούν οι τιμές `from+1` ως `to`
  - τύπωσε την τιμή `from`

```c
/* εκτύπωση τιμών από n μέχρι 0 */

#include <stdio.h>

void printIntsRev(int from, int to) {
  if (from <= to) {
    printIntsRev(from+1,to);
     printf("%d ",from);
  }
}

int main(int argc, char *argv[]) {
  int n;

  printf("enter int: ");
  scanf("%d",&n);
  printIntsRev(0,n);

  return(0);
}
```


printIntsRev(0,3)
printIntsRev(1,3)
printIntsRev(2,3)
printIntsRev(3,3)
printIntsRev(4,3)
3
2
1
0

Αναδρομική σκέψη / λύση προβλημάτων

- Για να λυθεί ένα πρόβλημα με αναδρομή, πρέπει να **εκφραστεί** με αναδρομικό τρόπο

- Τυπική προσέγγιση:

1. Προσπαθούμε να βρούμε την λύση του προβλήματος για την **πιο απλή** περίπτωση (όπου τερματίζεται η αναδρομή)
2. Στην συνέχεια, ανάγουμε/κατασκευάζουμε την λύση της (αμέσως) πιο πολύπλοκης περίπτωσης, **με βάση** την λύση της πιο απλής περίπτωσης (αναδρομή)

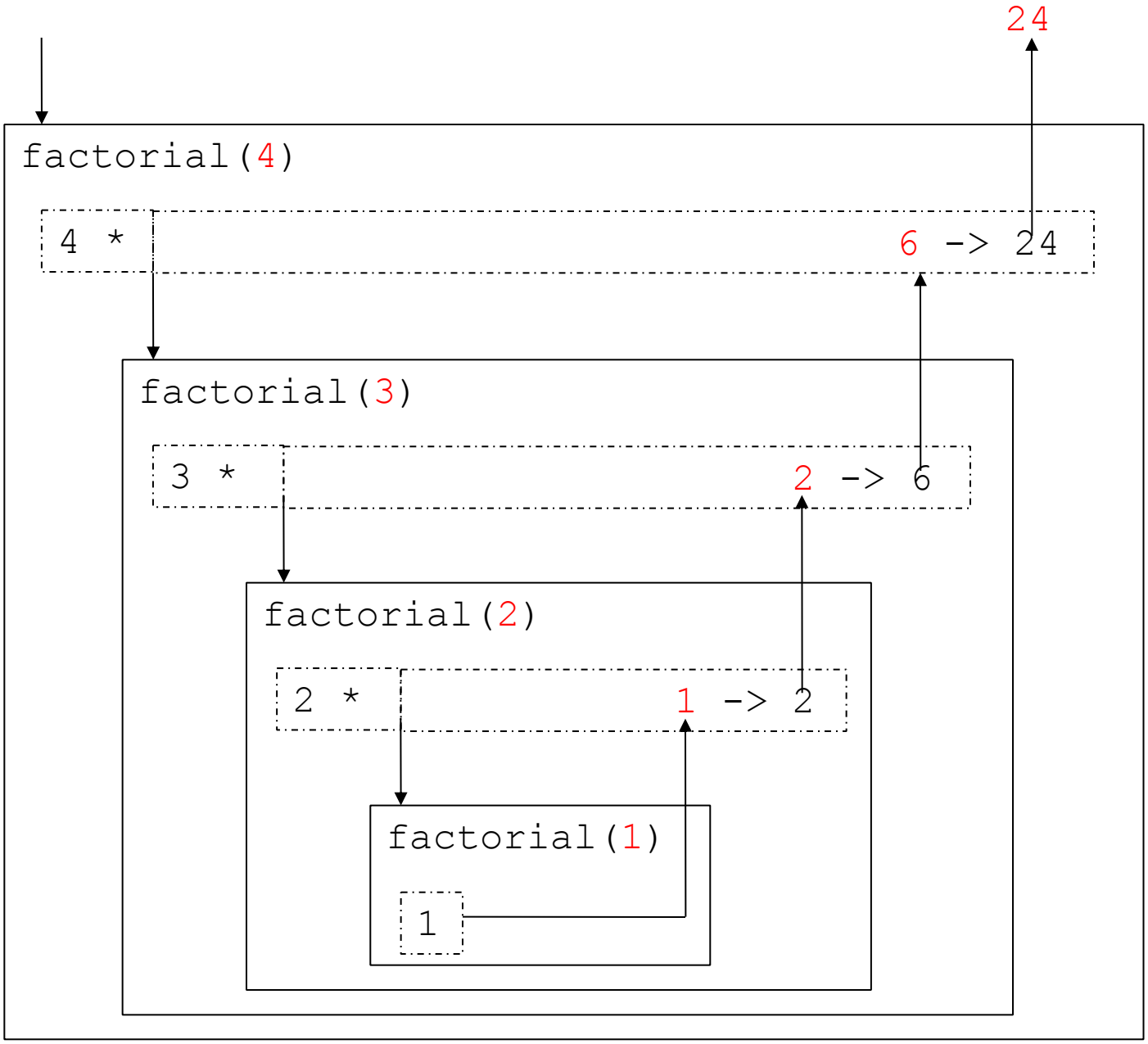
- Αν τα (1) και (2) γίνουν «σωστά», έχουμε **ήδη κατασκευάσει** την λύση στο πρόβλημα μας!

```
/* συμβατικός υπολογισμός n! */

int factorial(int n) {
  int i, res;

  res = 1;
  for (i = 2; i <=n ; i++) {
    res = res * i;
  }

  return(res);
}
```

Αναδρομικός υπολογισμός

- Επιθυμούμε να υπολογίζουμε την έκφραση
  `x!` είναι `1*2*3*…*(x-1)*x`
- Περίπτωση τερματισμού
  `x <= 1` : επιστρέφεται `1`
- Γενική περίπτωση
  `x > 1`: επιστρέφεται `x*(x-1)!`
- Η λύση της γενικής περίπτωσης μπορεί να κατασκευαστεί με βάση την λύση του **ίδιου προβλήματος**, σε μικρότερη «κλίμακα»

```
/* αναδρομικός υπολογισμός n! */

int factorial(int n) {

  if (n <= 1) {
    return(1);
  }
  else {
    return( n * factorial(n-1) );
  }

}
```


24
factorial(4)
4 *
6 -> 24
factorial(3)
3 *
2 -> 6
factorial(2)
2 *
1 -> 2
factorial(1)
1

Η σειρά Fibonacci

- Οι αριθμοί Fibonacci είναι μια άπειρη σειρά ακεραίων 0, 1, 1, 2, 3, 5, 8, 13, 21, ...
  - προτάθηκε από τον Leonardo Fibonacci (1170-1250)
- Η λογική παραγωγής: για να πάρεις την επόμενη τιμή, πρόσθεσε τις δύο τελευταίες τιμές, χρησιμοποιώντας για τις πρώτες δύο τιμές το 0 και το 1
  - 0+1->1, 1+1->2, 1+2->3, 2+3->5, …

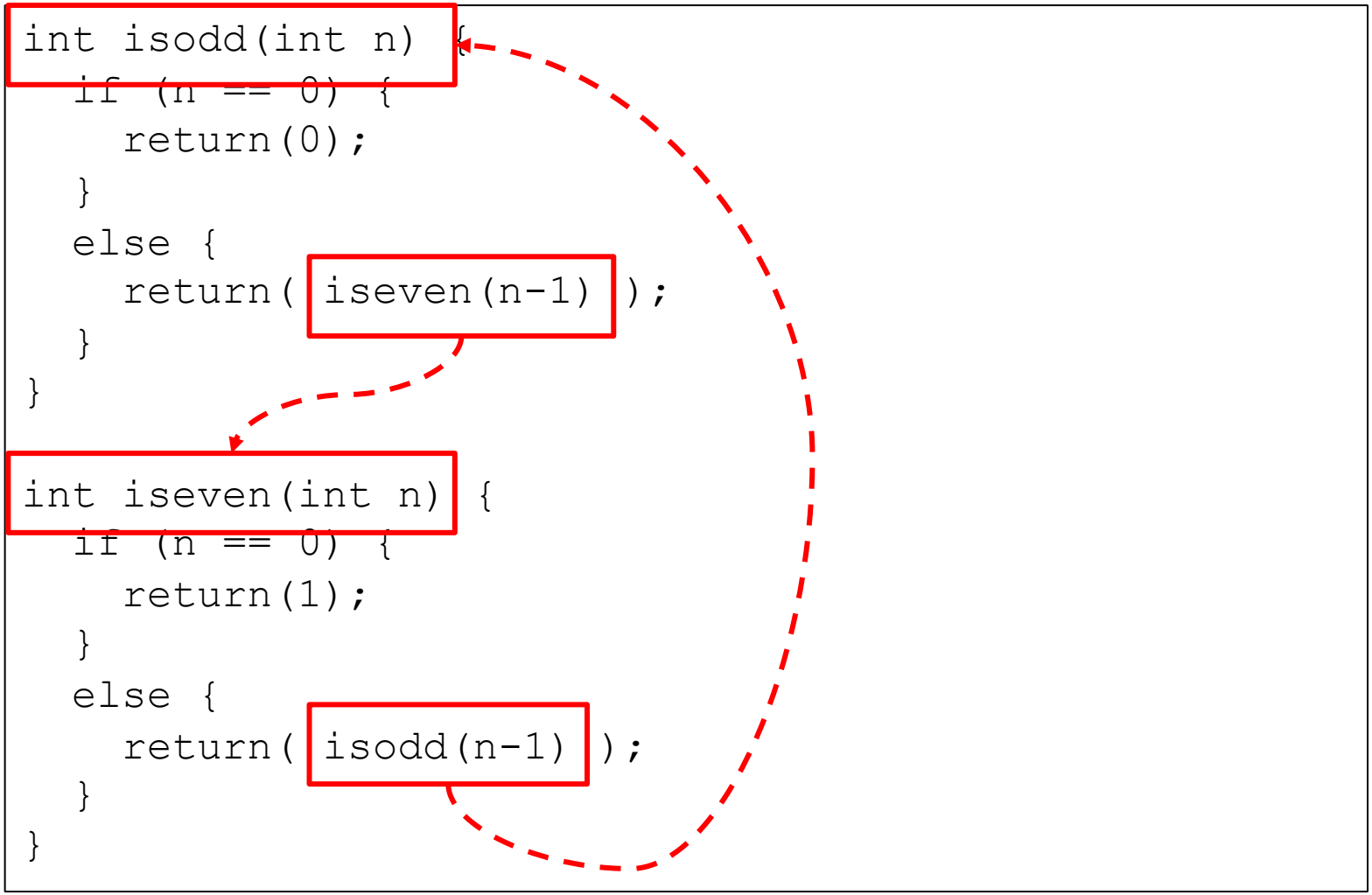
- Μια πιο μαθηματική περιγραφή

  fib(0): 0, fib(1): 1

  fib(n): fib(n-1) + fib(n-2)

```
/* υπολογισμός fibonacci */

int fib(int n) {

  if (n <= 1) {
    return(n);
  }
  else {
    return( fib(n-1) + fib(n-2) );
  }

}
```

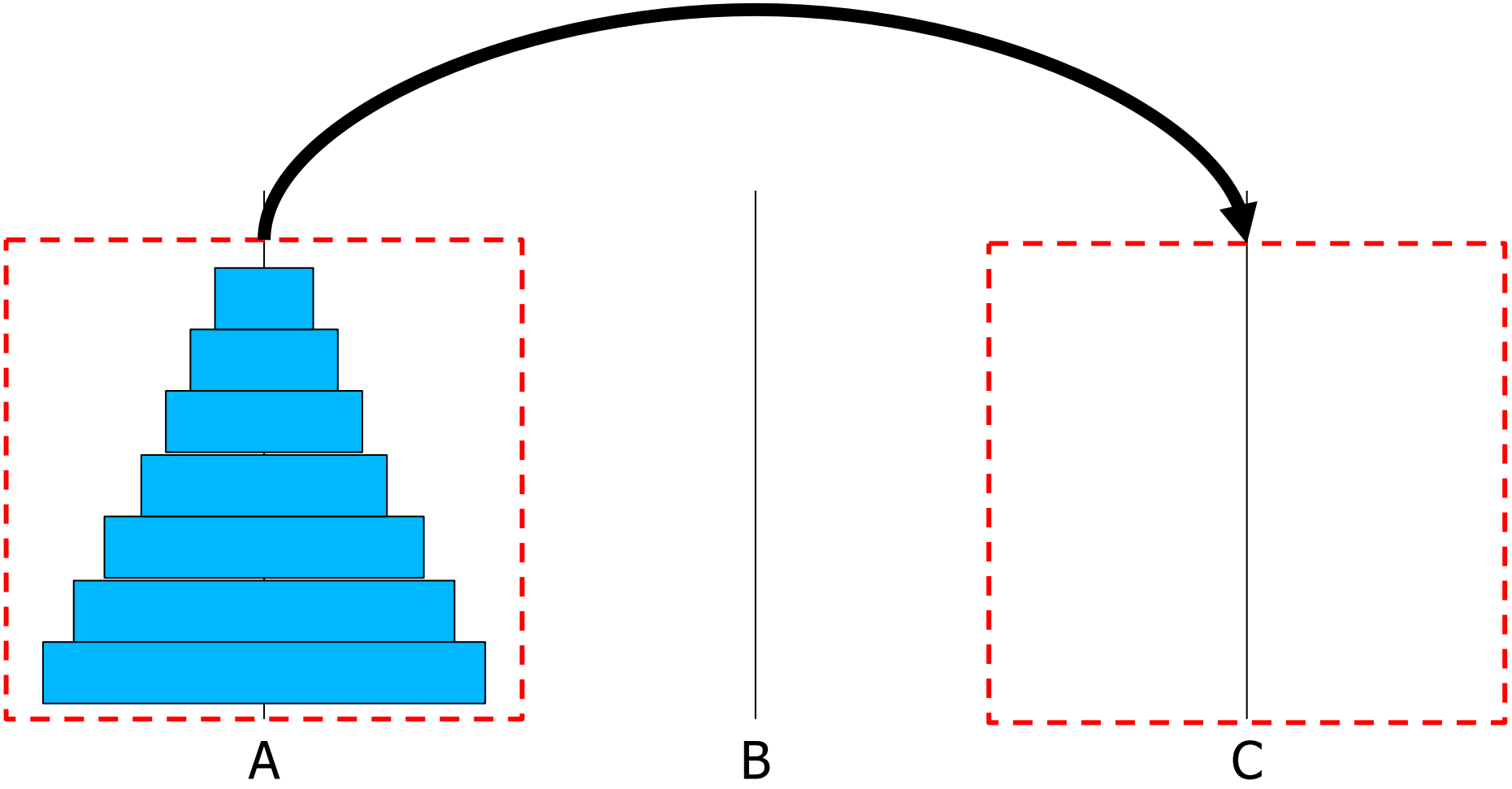
Αναδρομή με πολλές συναρτήσεις



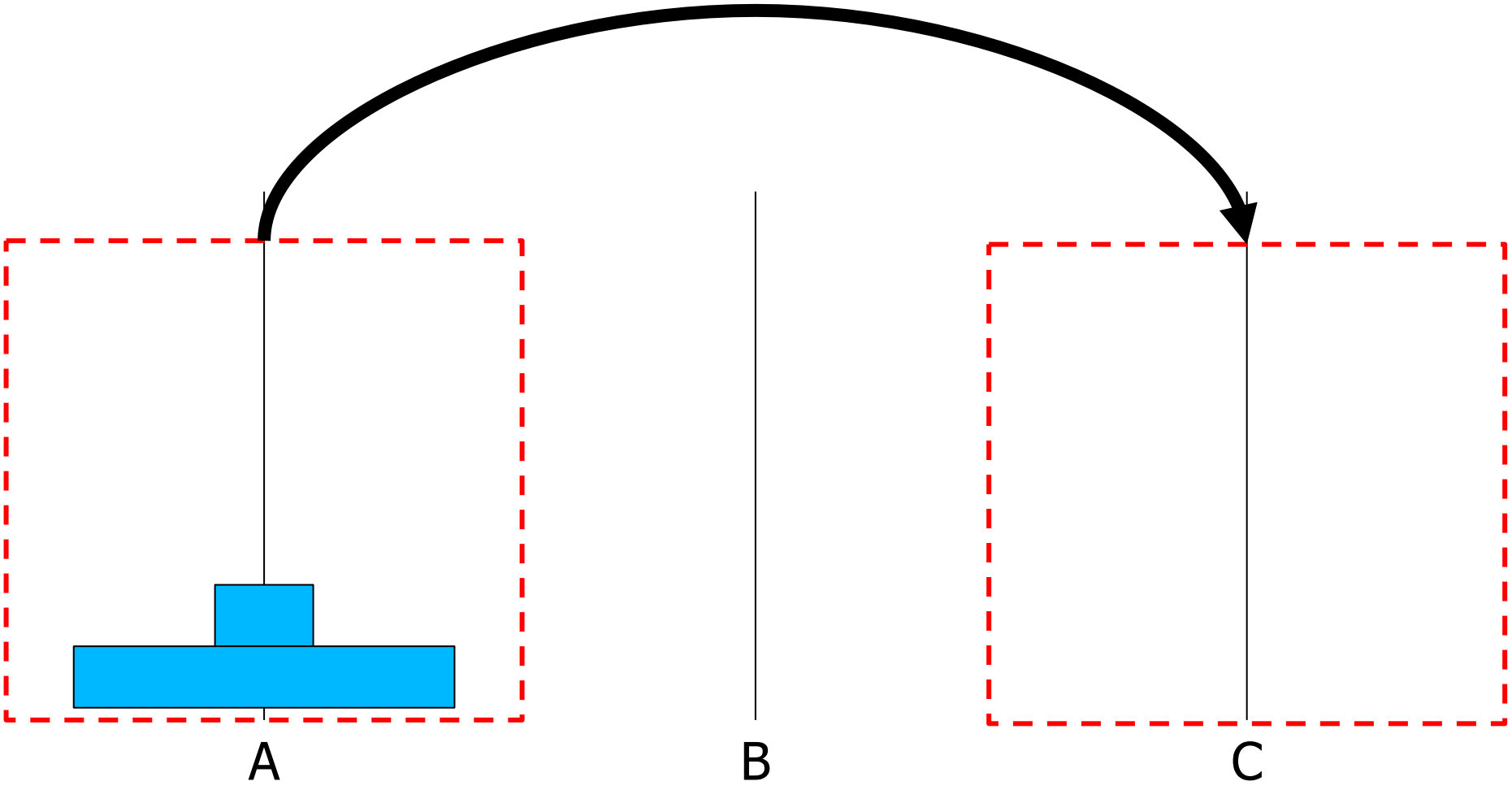
Οι πύργοι του Hanoi

- Στην θέση Α υπάρχει μια στοίβα από δίσκους, τοποθετημένους με φθίνουσα σειρά μεγέθους
  - οι μικρότεροι δίσκοι πάνω από τους μεγαλύτερους
- Πρέπει να μεταφέρουμε την στοίβα σε την θέση C
- Μέσω μιας επιπλέον βοηθητικής στοίβας B

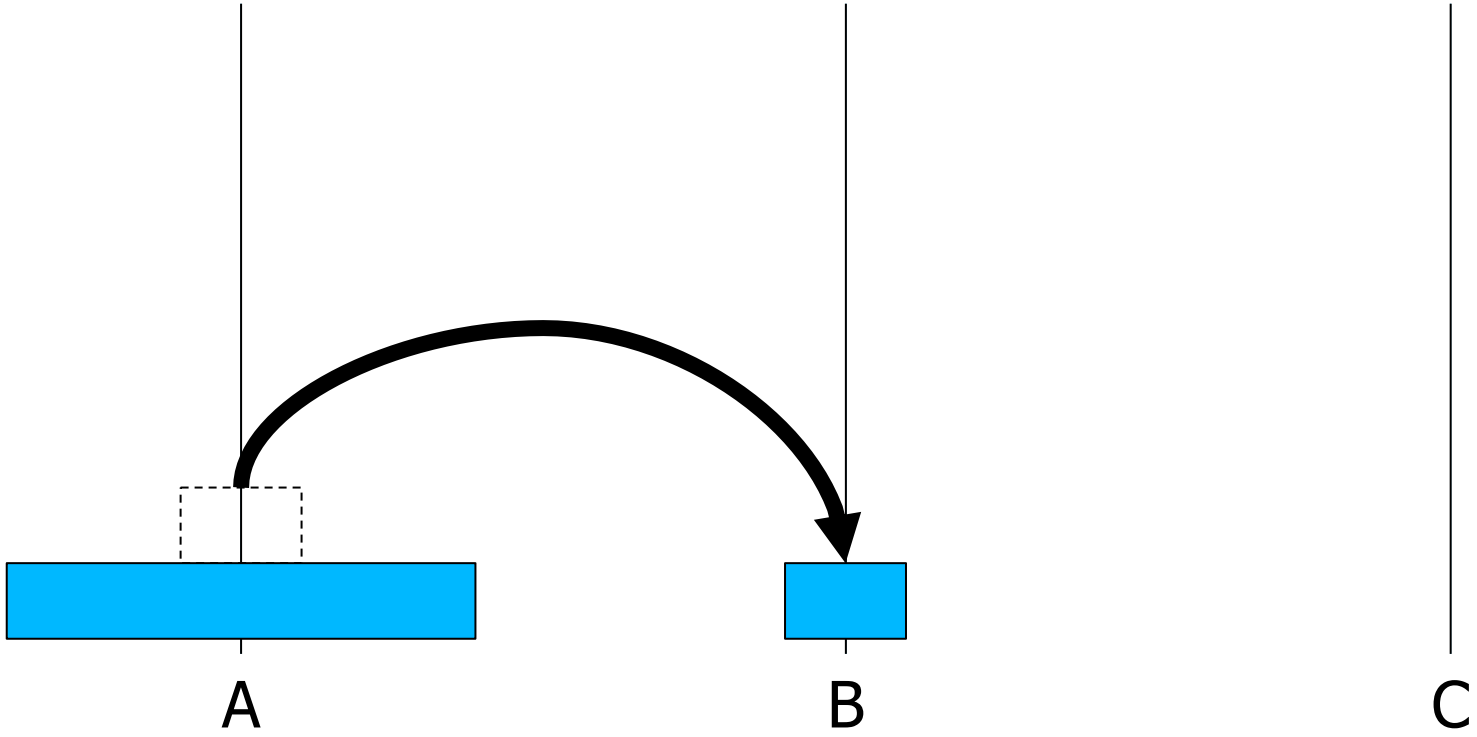
- **Κανόνας 1:** σε κάθε κίνηση που κάνουμε, επιτρέπεται να μετακινούμε **μόνο 1 δίσκο**
- **Κανόνας 2:** οι δίσκοι πρέπει να στοιβάζονται ο ένας πάνω στον άλλο με **φθίνουσα σειρά μεγέθους**


A
B
C

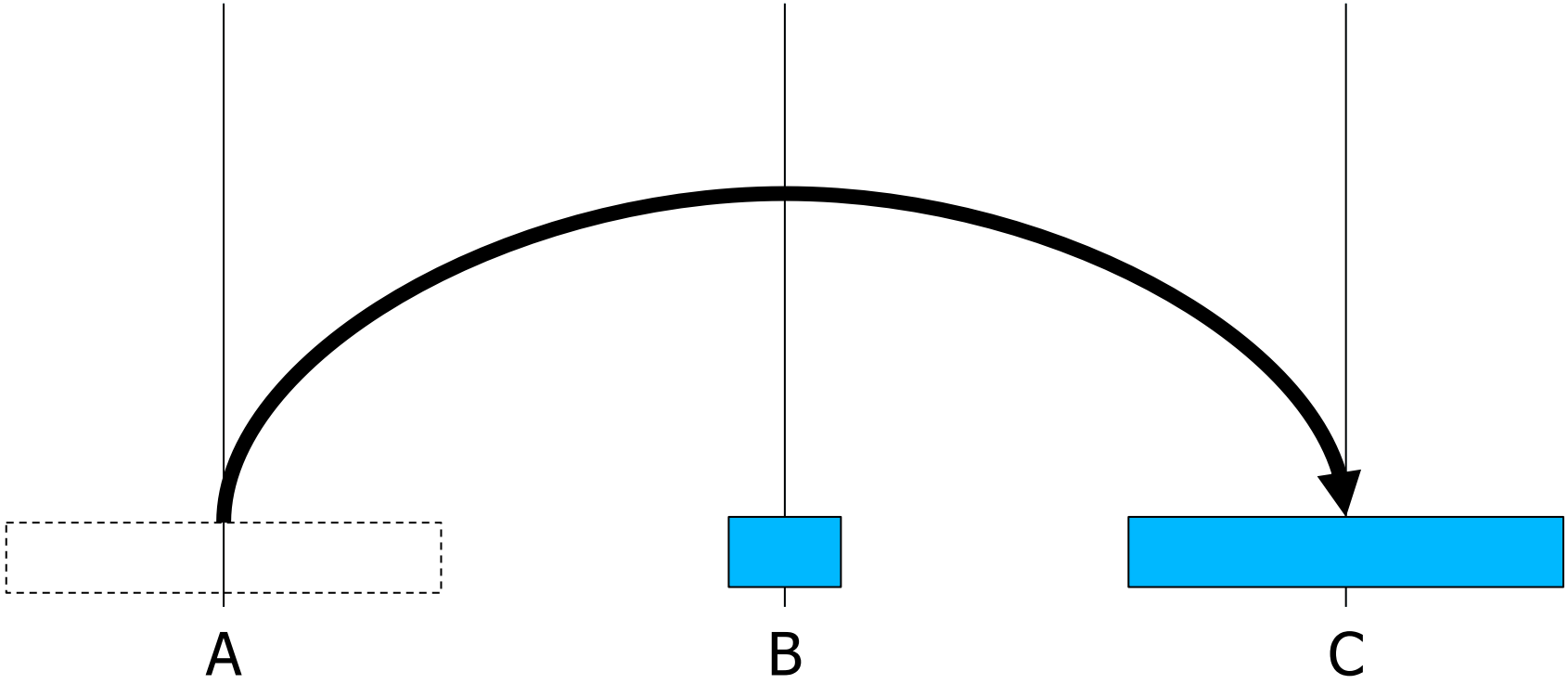
- Το πρόβλημα φαίνεται πολύπλοκο ...
- Ποια είναι η **απλούστερη** (μη τετριμμένη) έκδοση του προβλήματος;
- Το ίδιο πρόβλημα, **μόνο** για 2 δίσκους


A
B
C

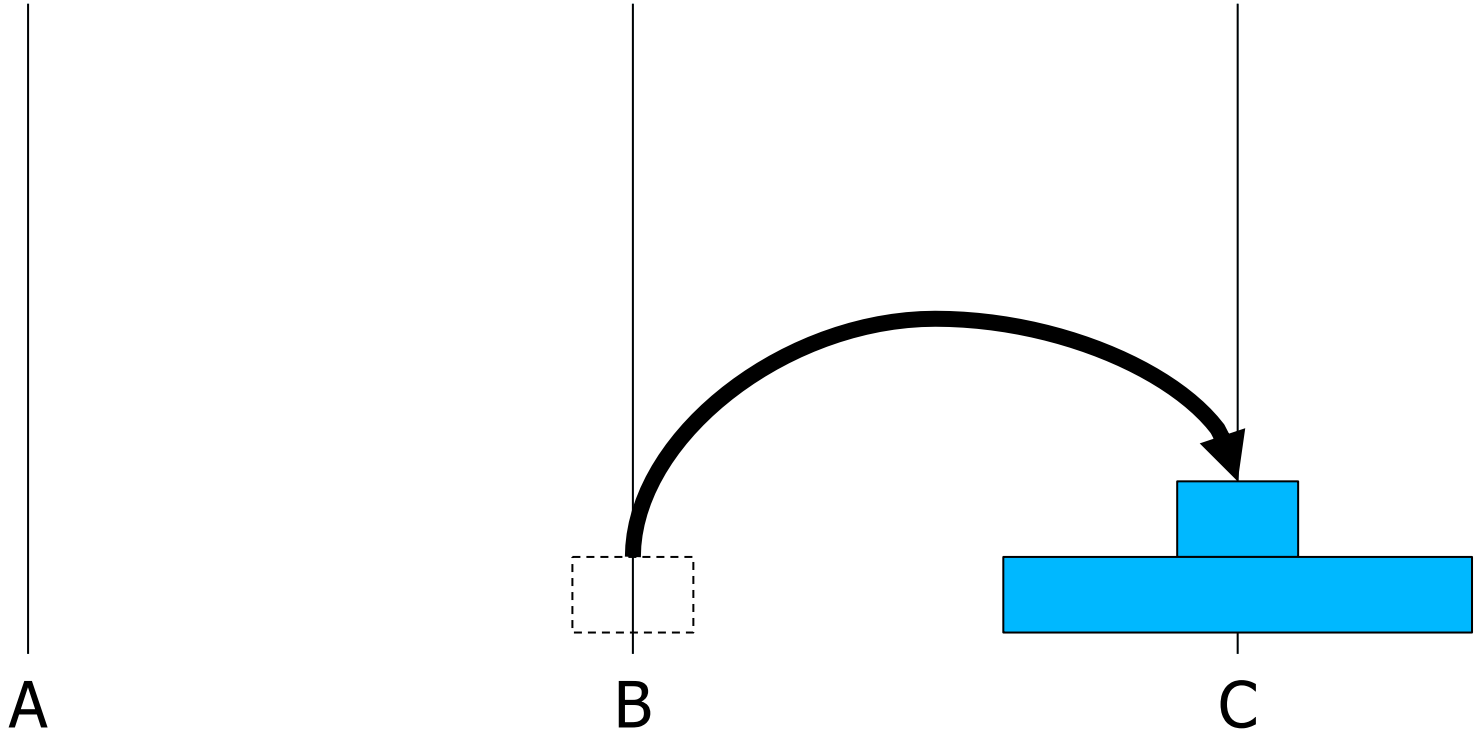
1ο βήμα



2ο βήμα



3ο βήμα



Τα βήματα της λύσης

1. Μετακίνησε τον **μικρό** δίσκο από **Α σε Β**
2. Μετακίνησε τον **μεγάλο** δίσκο από **Α σε C**
3. Μετακίνησε τον **μικρό** δίσκο από **Β σε C**

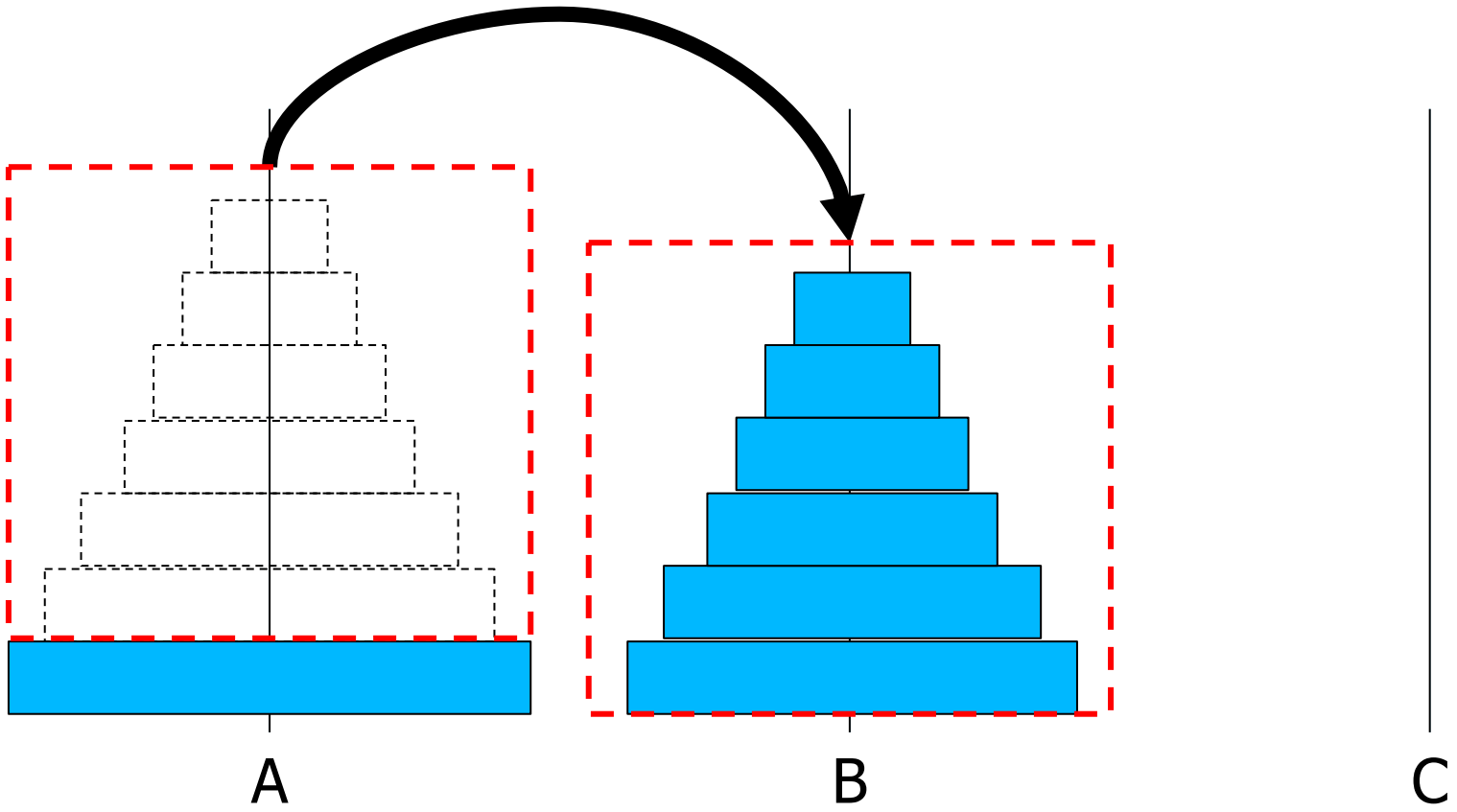
Αναδρομική προσέγγιση

- **Μικρός δίσκος** = όσοι δίσκοι είναι στοιβαγμένοι πάνω στον μεγαλύτερο δίσκο

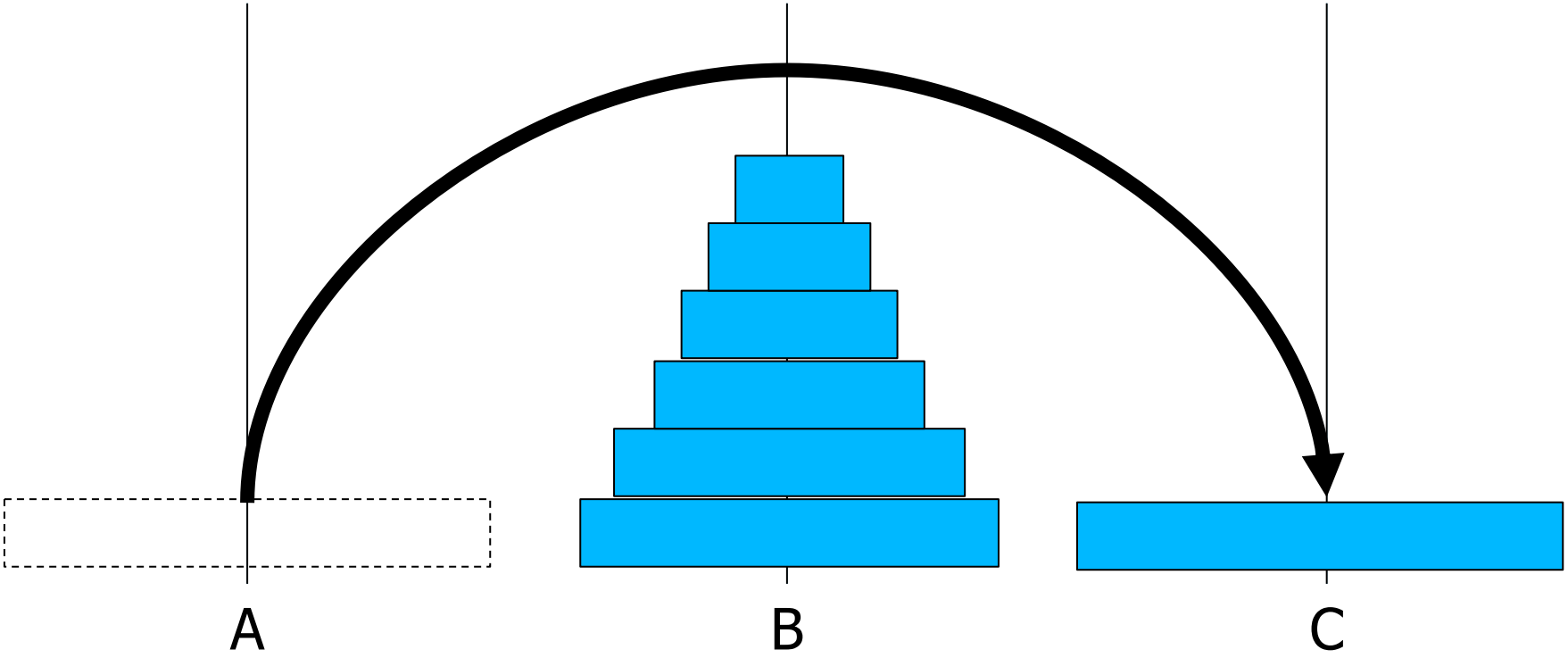
- Εφαρμόζομε την λύση που **ήδη** έχουμε για 2 δίσκους

1. Μετακίνησε **«τον μικρό δίσκο»** από Α σε Β
   - χρησιμοποιώντας ως βοηθητική θέση/στοίβα την C
2. Μετακίνησε τον μεγάλο δίσκο από Α σε C
   - δεν χρειαζόμαστε κάποια βοηθητική θέση/στοίβα
3. Μετακίνησε **«τον μικρό δίσκο»** από Β σε C
   - Χρησιμοποιώντας ως βοηθητική θέση/στοίβα την Α

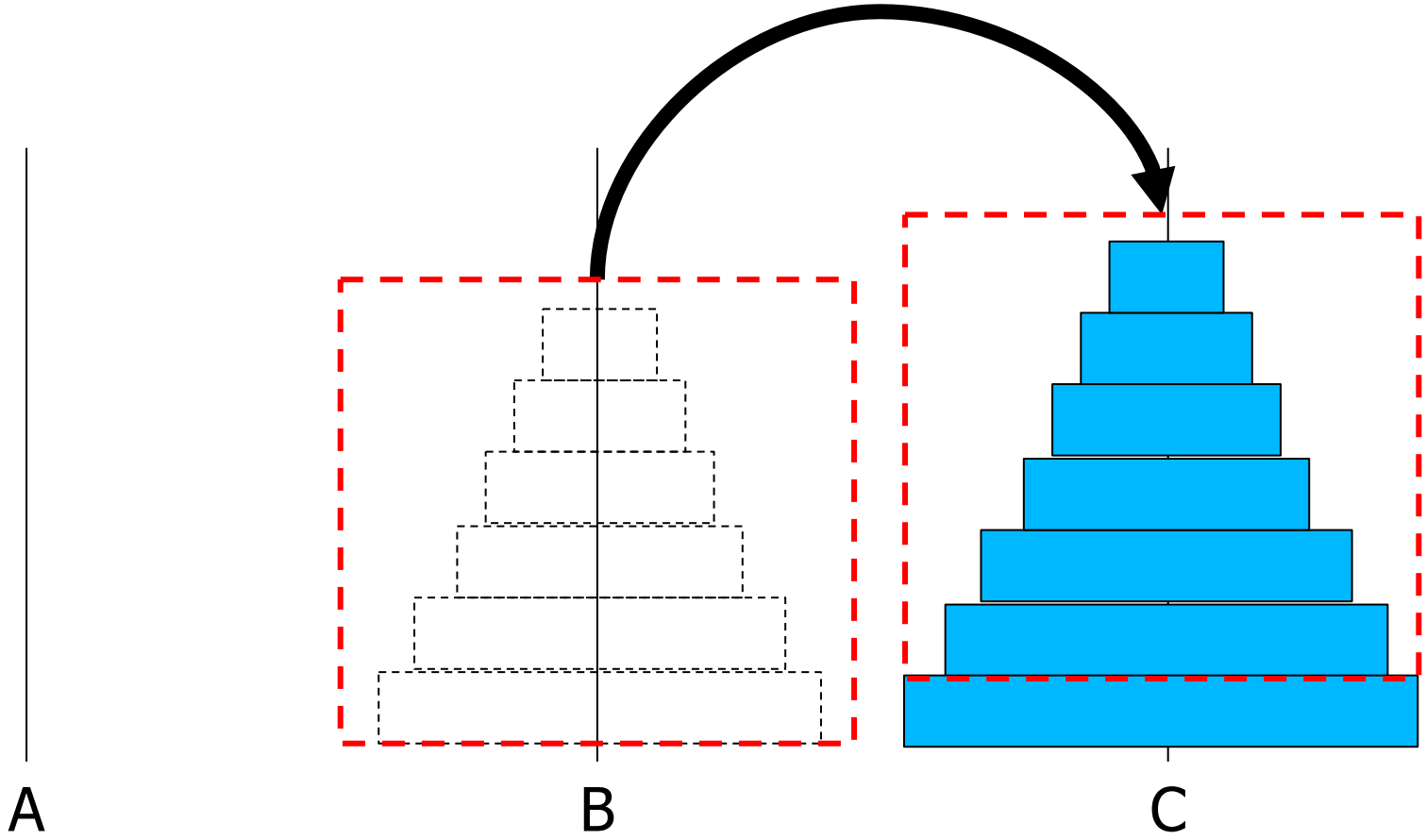
1ο βήμα



2ο βήμα



3ο βήμα



```
/* οι πύργοι του Hanoi */

void moveDisks(int disks, stack a, stack b, stack c) {
  if (disks == 1) {
    move1Disk(a, c);
  }
  else {
    moveDisks(disks-1, a, c, b);   /* step 1 */
    move1Disk(a, c);               /* step 2 */
    moveDisks(disks-1, b, a, c);   /* step 3 */
  }
}
```

Αναδρομή για πολύπλοκα προβλήματα

- Η αναδρομή μπορεί να χρησιμοποιηθεί για την επίλυση αρκετά πολύπλοκων προβλημάτων
  - με (σχετικά) εύκολο/φυσιολογικό τρόπο

- Κλασική περίπτωση: έλεγχος καταστάσεων που μπορεί να κατασκευαστούν με αναδρομικό τρόπο

Παράδειγμα

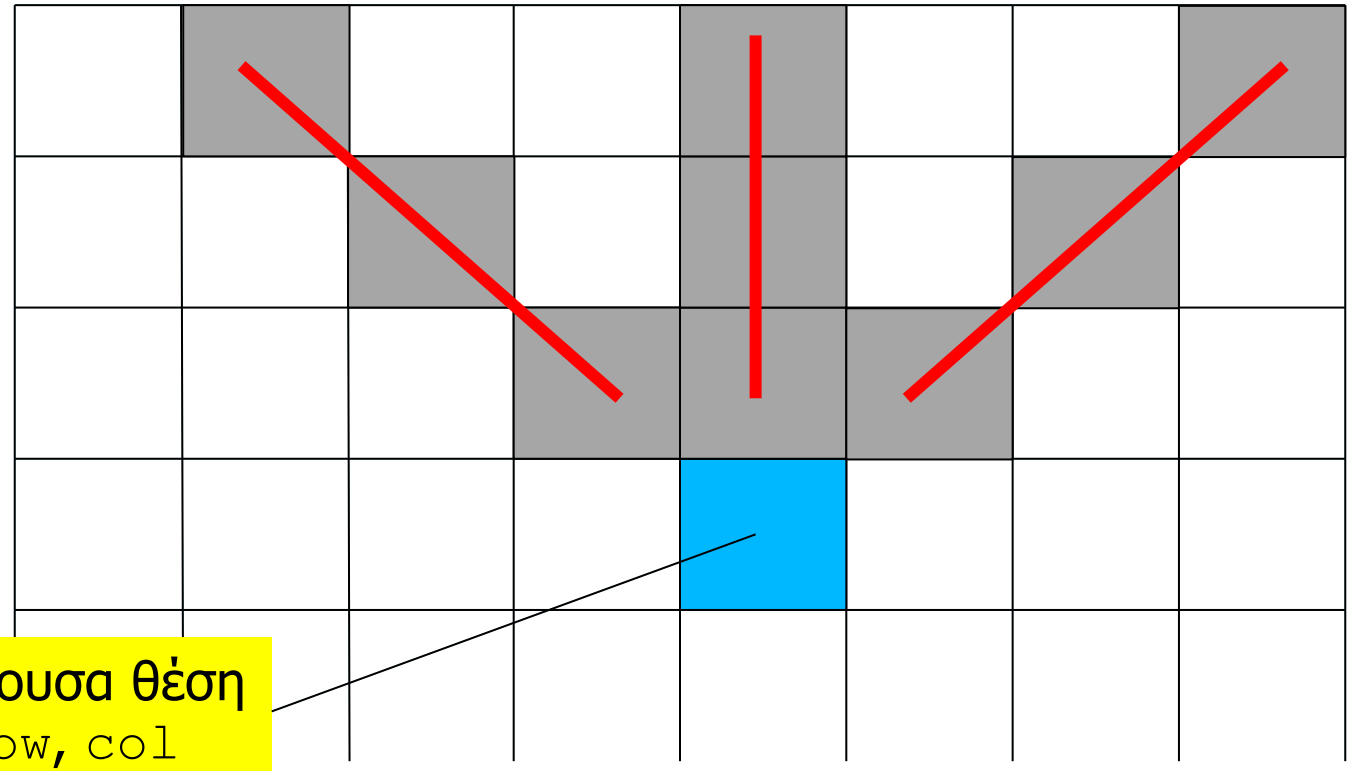
- **The N queens problem**
- Τοποθέτησε N βασίλισσες σε μια σκακιέρα NxN έτσι ώστε να μην απειλούνται μεταξύ τους

Αναδρομική προσέγγιση (κατά γραμμές)

Διαδικασία τοποθέτησης βασίλισσας στην γραμμή `row` (που μπορεί να λάβει τιμές από `0` ως `N-1`)

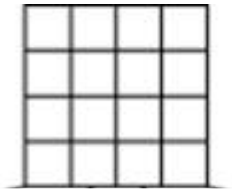
- Αν `row` είναι ίσο με `N`, επέστρεψε τιμή επιτυχίας
- Για κάθε μια στήλη `col` από `0` έως `N-1`
  - έλεγξε αν μπορεί να τοποθετηθεί μια βασίλισσα στην θέση `row`, `col`
  - αν όχι, συνέχισε με την επόμενη στήλη (της ίδιας γραμμής)
  - αν ναι, τοποθέτησε μια βασίλισσα στην θέση `row`, `col` και επανέλαβε την διαδικασία για την γραμμή `row+1`
  - αν πετύχει η διαδικασία, επέστρεψε τιμή επιτυχίας
  - αν αποτύχει η διαδικασία, ακύρωσε την τοποθέτηση στην θέση `row`, `col` και συνέχισε με την επόμενη στήλη
- Αν ο έλεγχος δεν είναι επιτυχής για καμία από τις στήλες της γραμμής `row`, επέστρεψε τιμή αποτυχίας

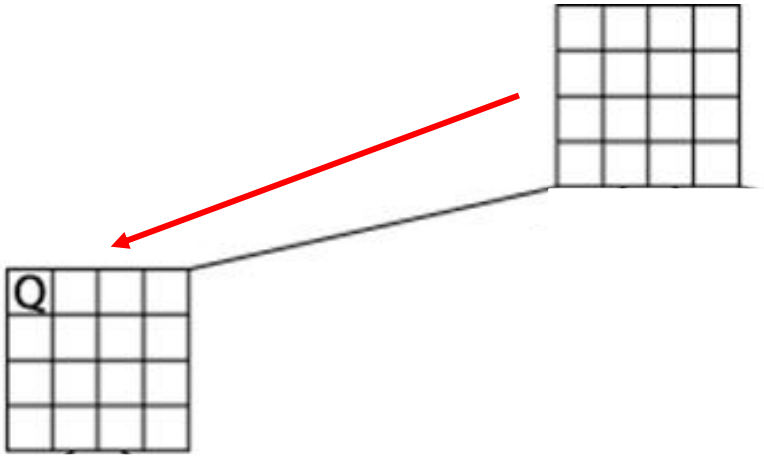
Έλεγχος θέσης `row, col`

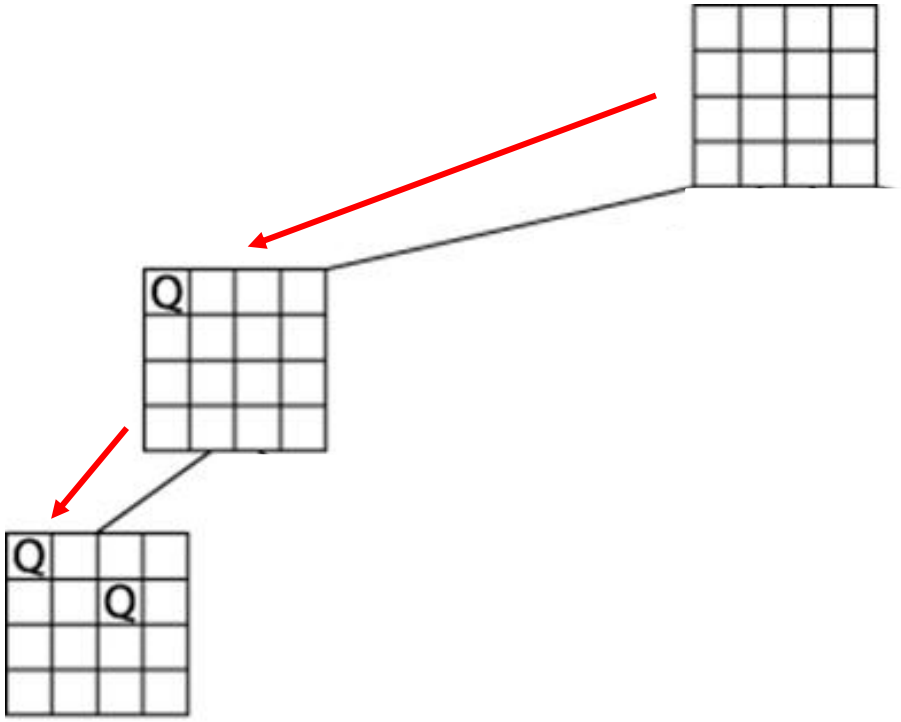


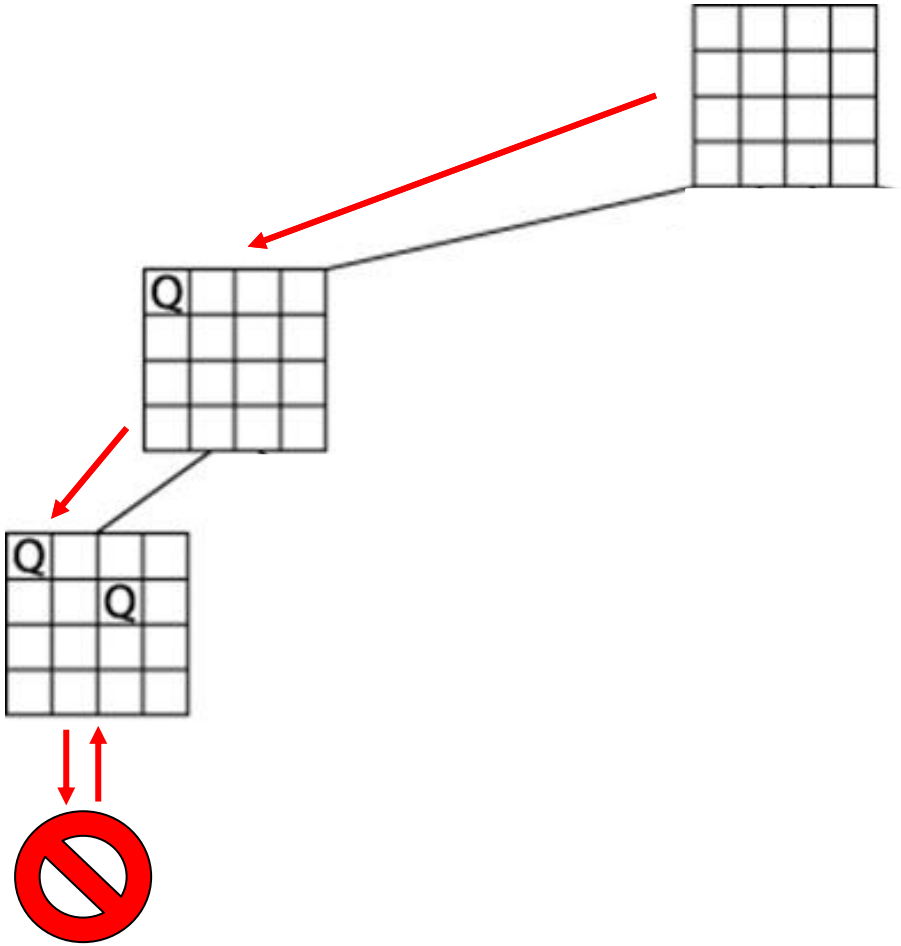
δεν χρειάζεται να ελέγχουμε θέσεις που βρίσκονται στην τρέχουσα ή στις παρακάτω γραμμές – γιατί;

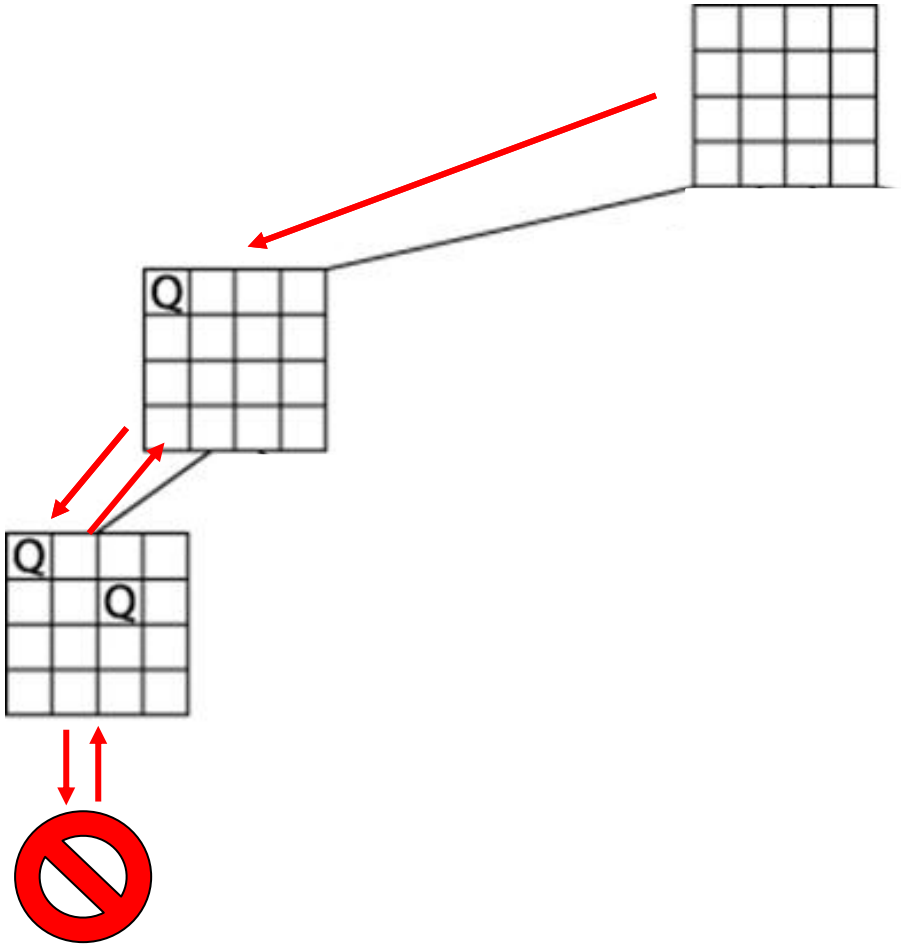
```
/* διαδικασία τοποθέτησης στην γραμμή row */
int placeInRow(int board[N][N], int row) {
  int col;

  if (row == N) {
      return(1);  /* end of recursive descent; success */
  }

  /* try placing a queen in some column in this row */
  for (col = 0; col < N; col++) {
    if (isSafe(board, row, col)) {
       board[row][col] = 1; /* mark position */
       if (placeInRow(board, row + 1)) {
          return(1);  /* recursive descent successful */
       }
       else {
         board[row][col] = 0; /* unmark position */
       }
    }
  }
  return(0);  /* failure */
}
```

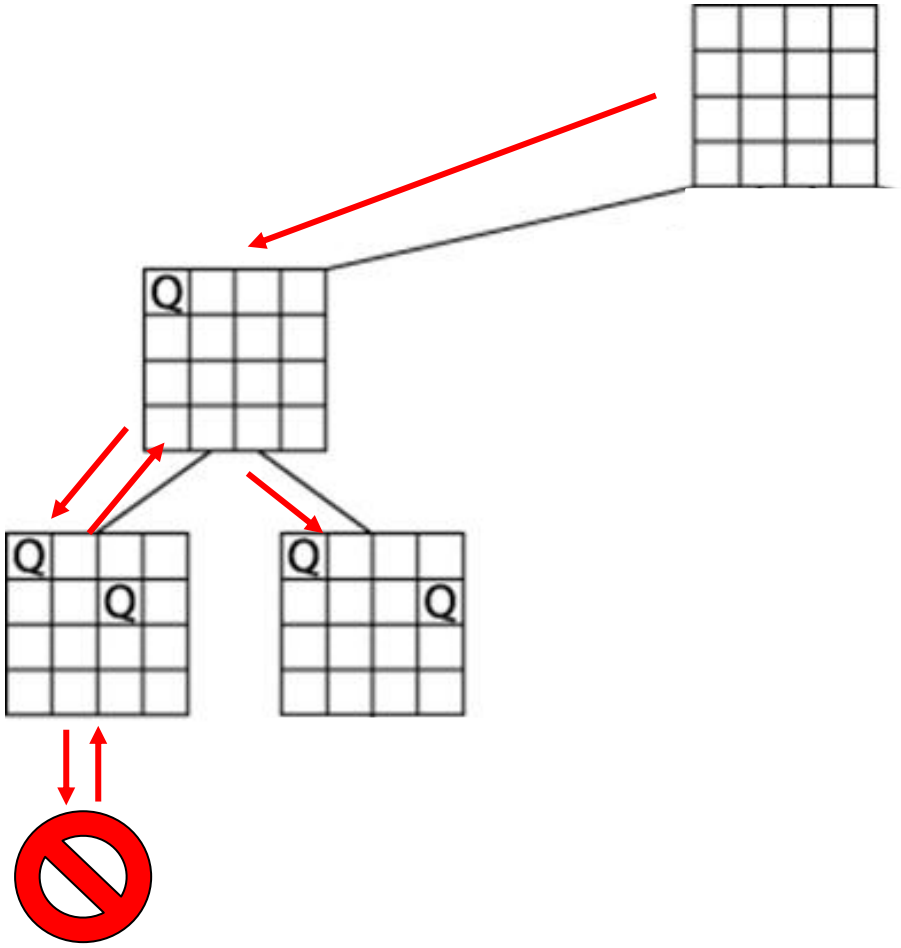


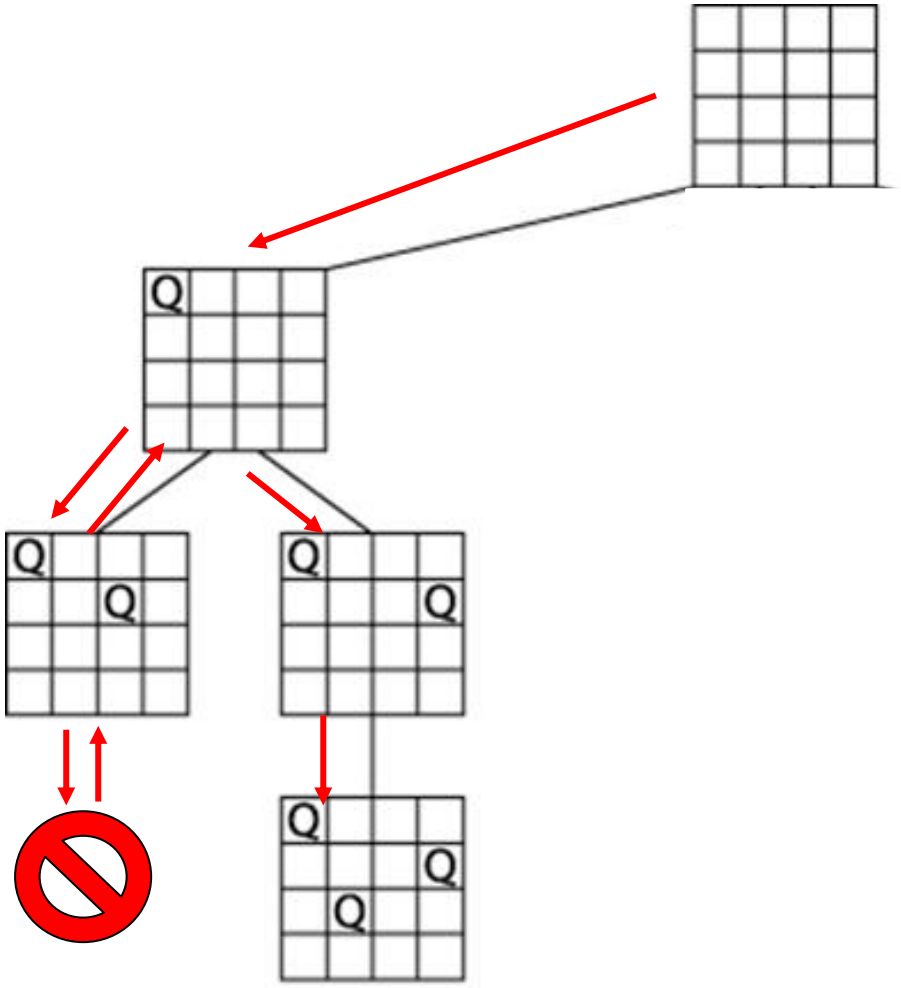

Q

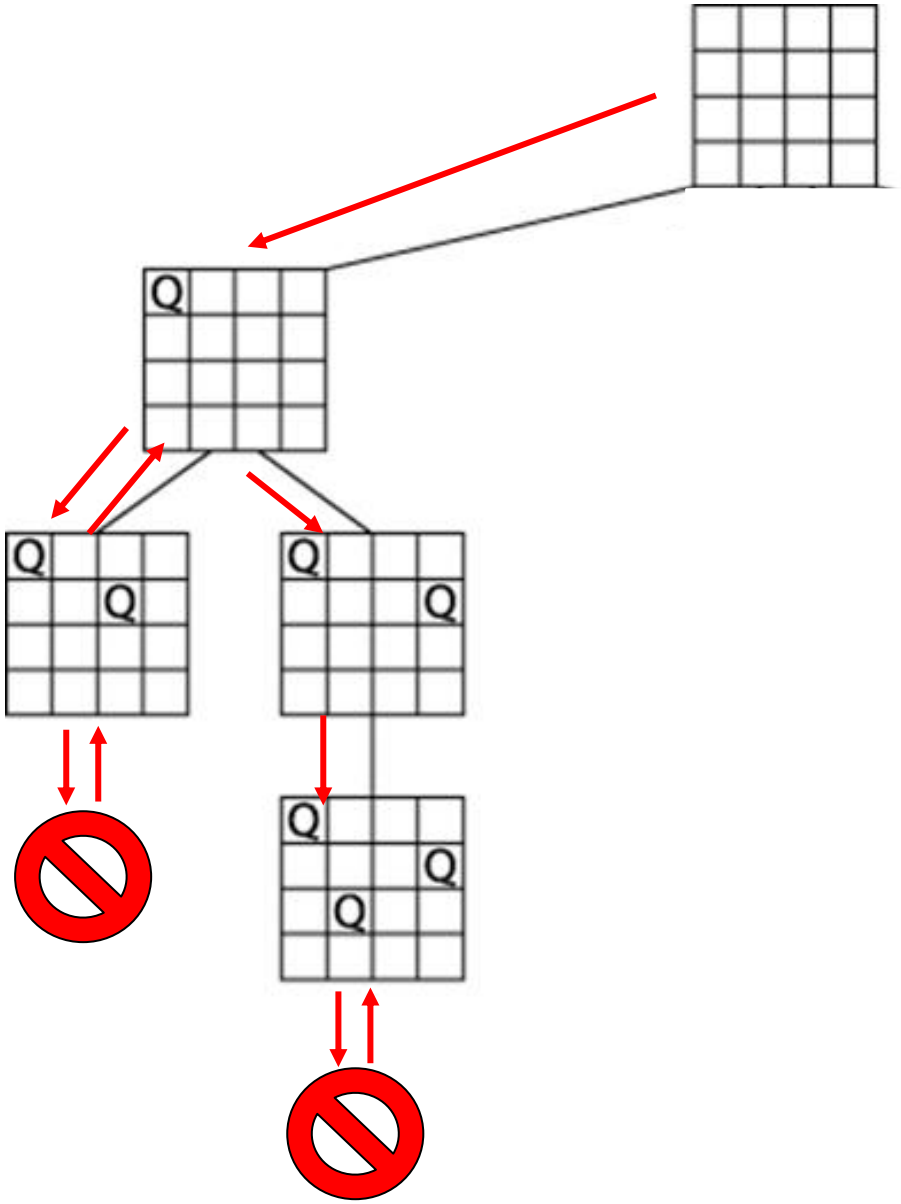

Q
Q
Q

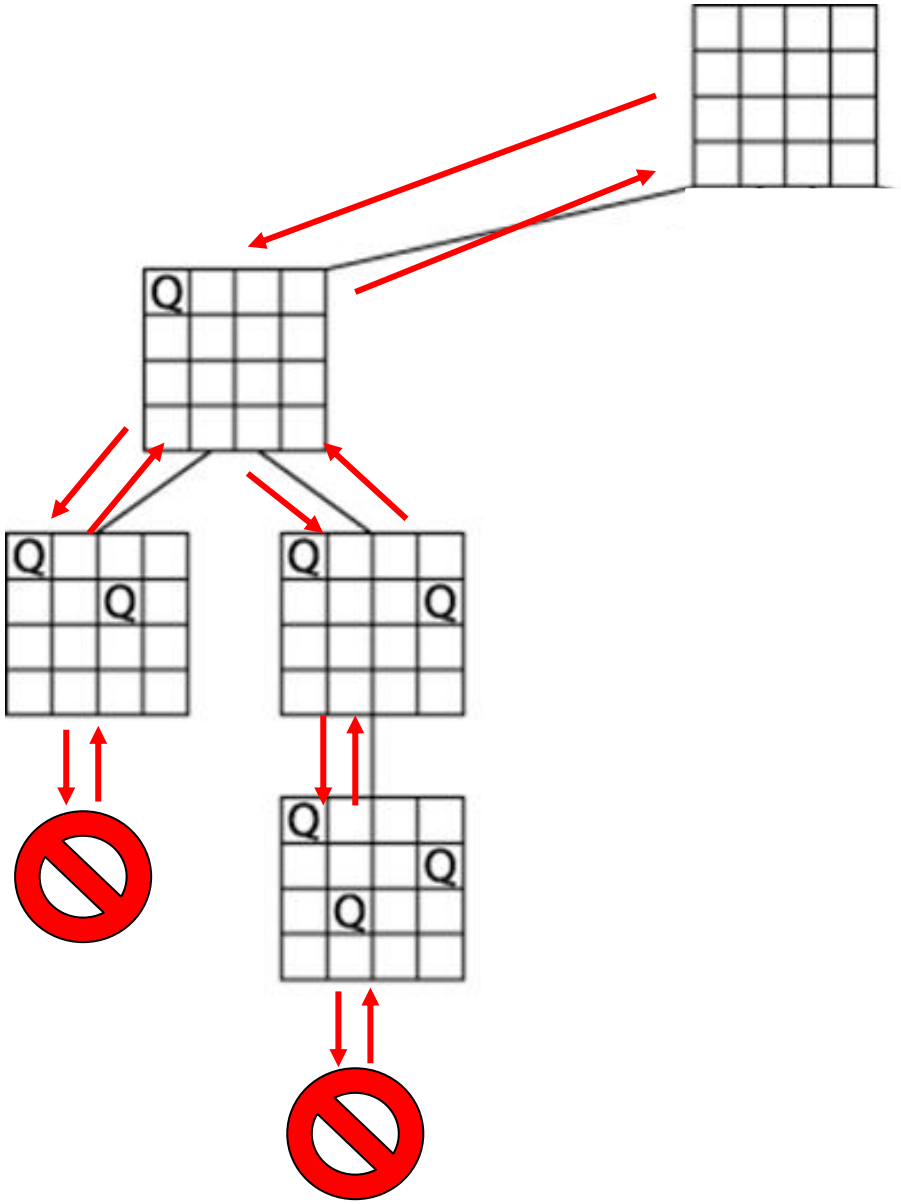

Q
Q
Q

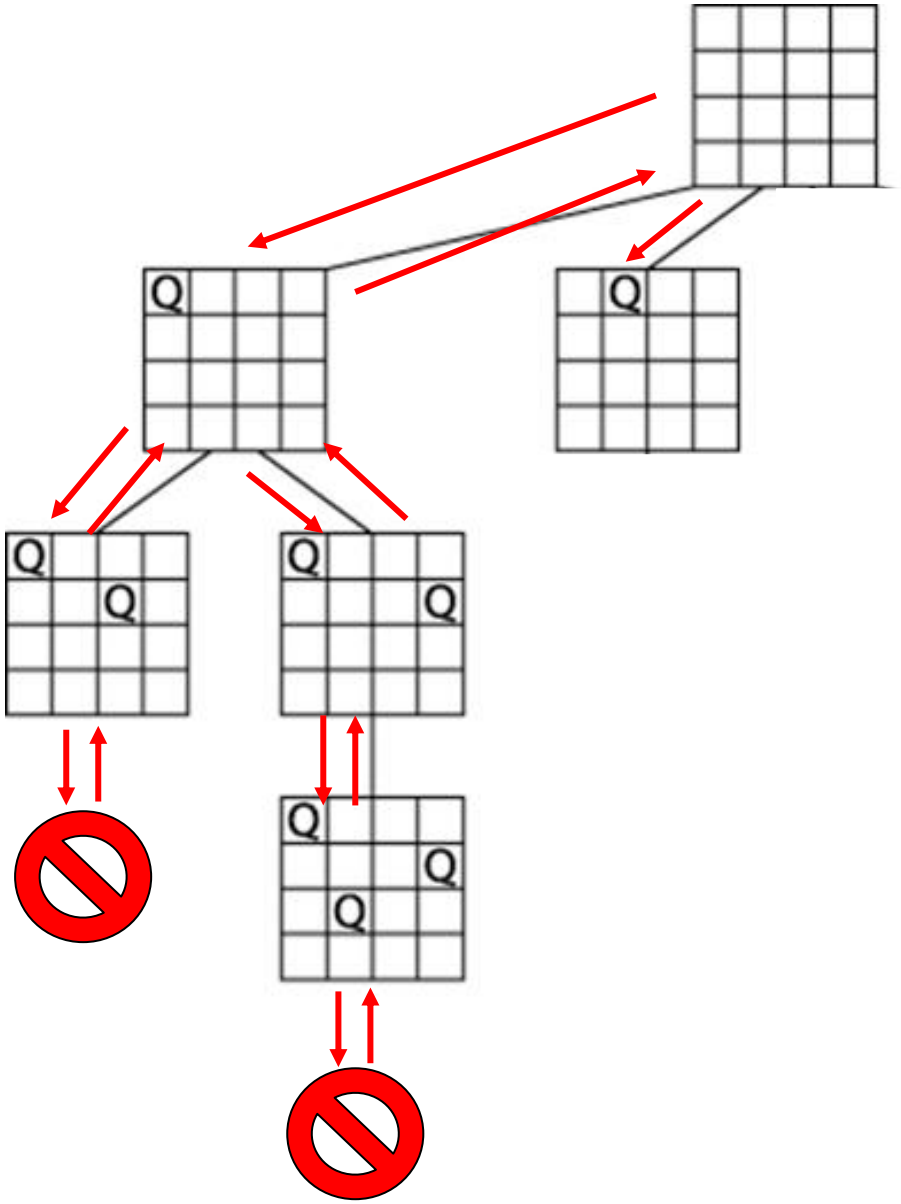

Q
Q
Q

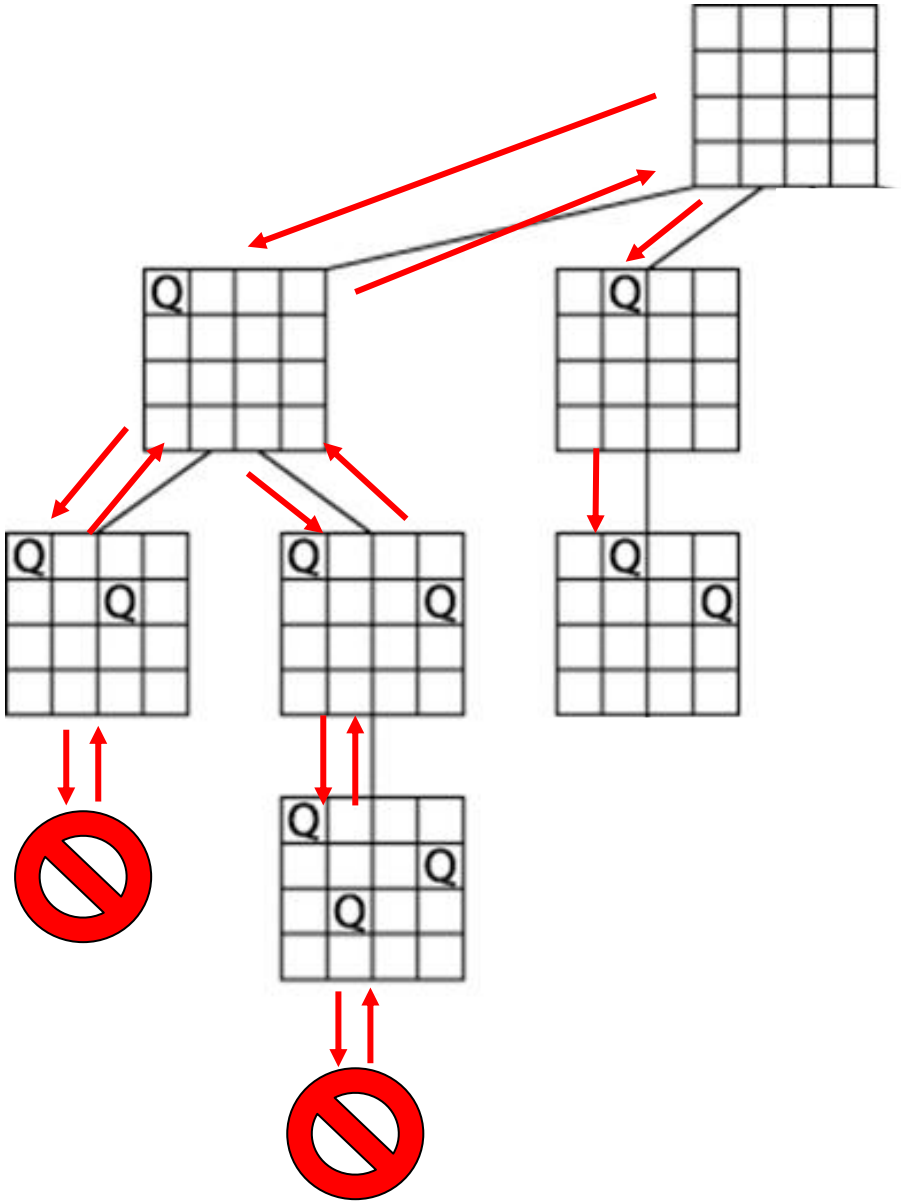

Q
Q
Q
Q
Q

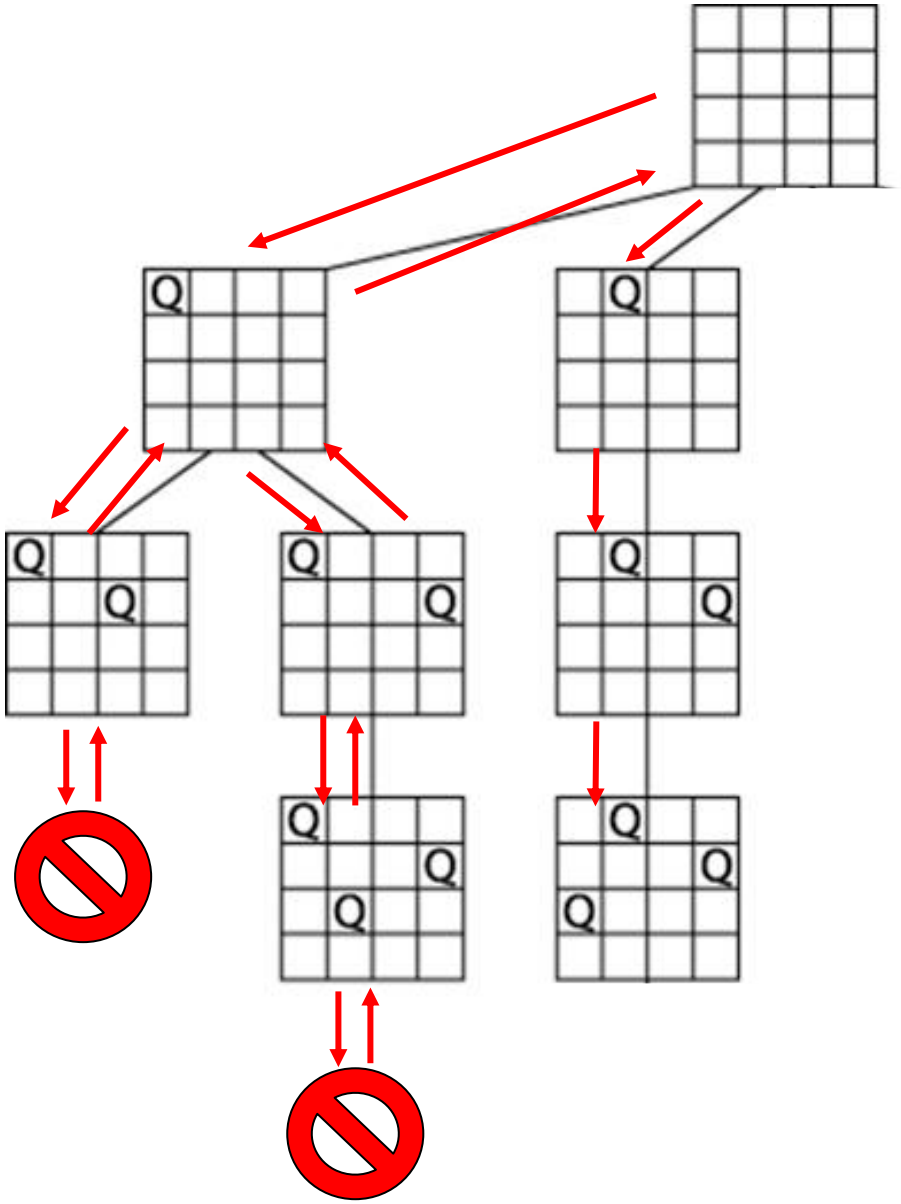

Q
Q
Q
Q
Q
Q
Q


Q
Q
Q
Q
Q
Q
Q


Q
Q
Q
Q
Q
Q
Q


Q
Q
Q
Q
Q
Q
Q
Q


Q
Q
Q
Q
Q
Q
Q
Q
Q
Q


Q
Q
Q
Q
Q
Q
Q
Q
Q
Q
Q
Q
Q
Q
Q