



Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. και Μηχανικών Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Παρουσίαση 2^{ης} Άσκησης:

Παραλληλοποίηση και βελτιστοποίηση αλγορίθμων σε αρχιτεκτονικές κοινής μνήμης

Ακ. Έτος 2023-2024

Συστήματα Παράλληλης Επεξεργασίας

9^ο Εξάμηνο

- Διαχωρισμός N αντικειμένων σε k μη επικαλυπτόμενες ομάδες (συστάδες - clusters)

until convergence (or fixed loops)

for each object

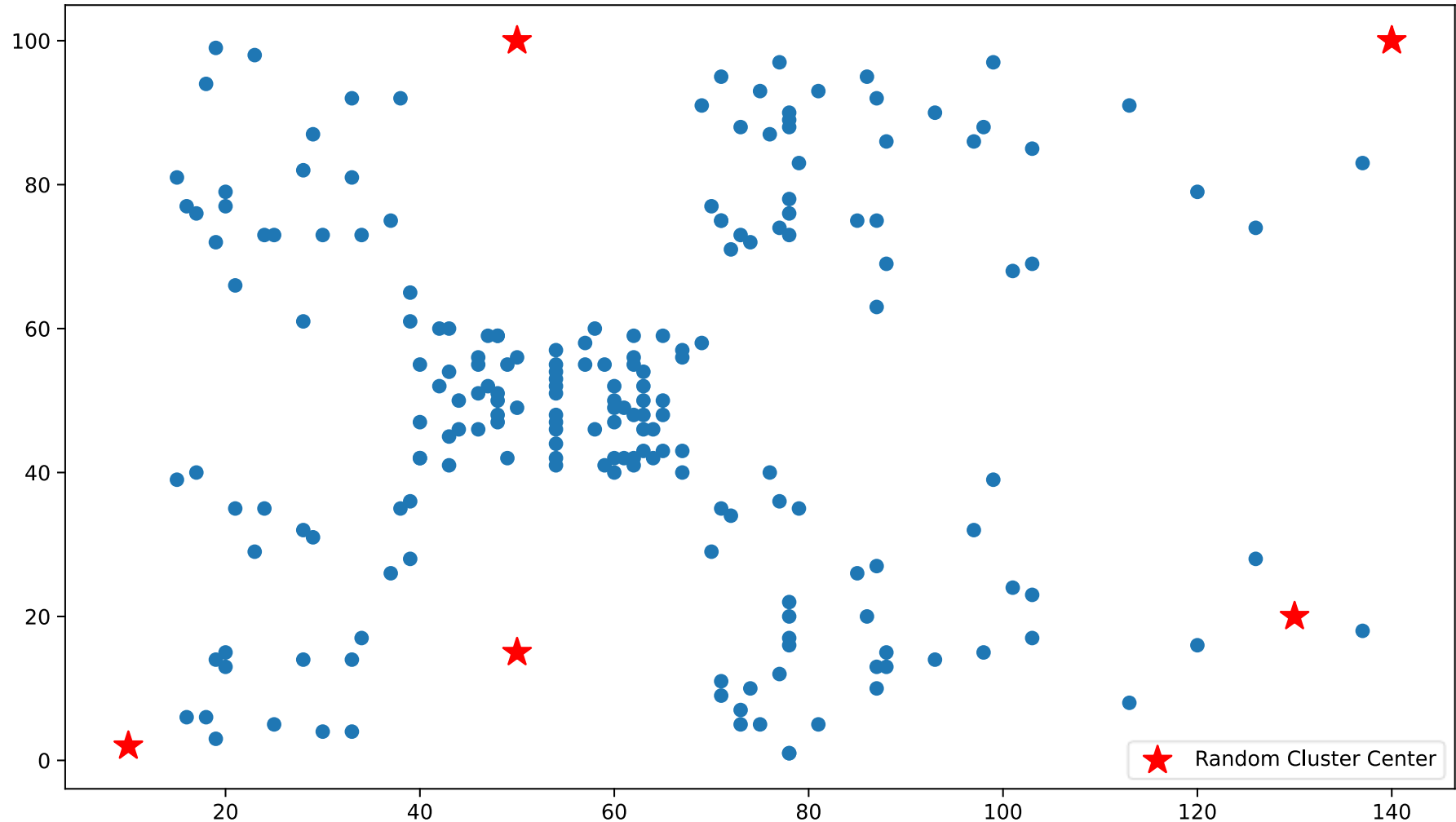
find nearest cluster

for each cluster

calculate new cluster center coordinates

- Σε κάθε επανάληψη υπολογίζει για κάθε αντικείμενο το κοντινότερο κέντρο των k διαθέσιμων clusters (είτε αλλάζει, είτε παραμένει το ίδιο με την προηγούμενη επανάληψη).
- Αφού όλα τα αντικείμενα εξεταστούν, ενημερώνονται οι συντεταγμένες των clusters με βάση τις αλλαγές που έχουν προκύψει από την εξέταση των αντικειμένων.
- Τερματισμός σύμφωνα με κριτήριο σύγκλισης
 - Αλλαγές στα κοντινότερα clusters μόνο για μικρό ποσοστό των αντικειμένων (πχ 0.1%)
 - Μέγιστο πλήθος επαναλήψεων

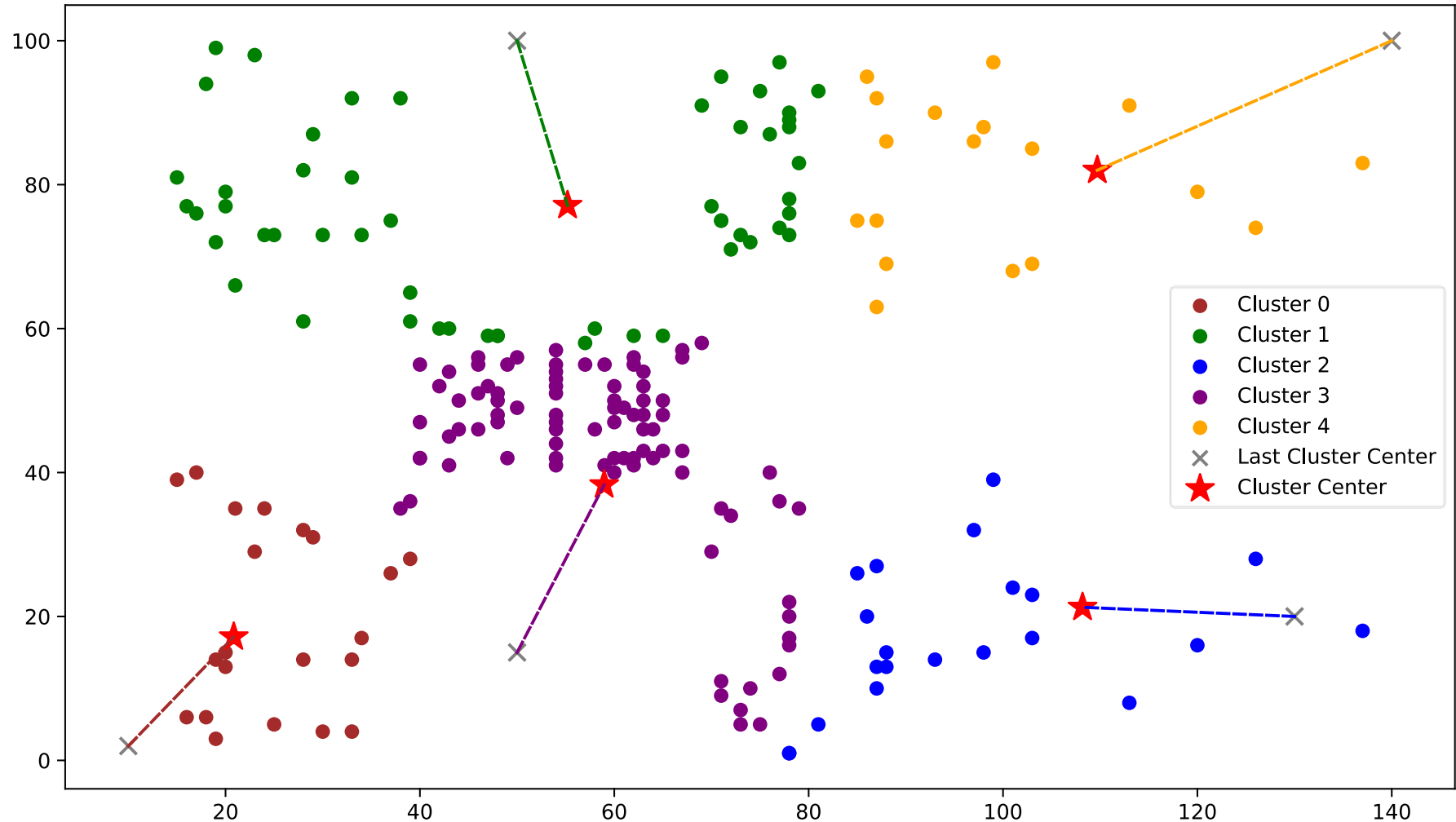
Random Initialization of Cluster Centers



(Source : <https://satishgunjal.com/kmeans/>)

Αλγόριθμος k-means

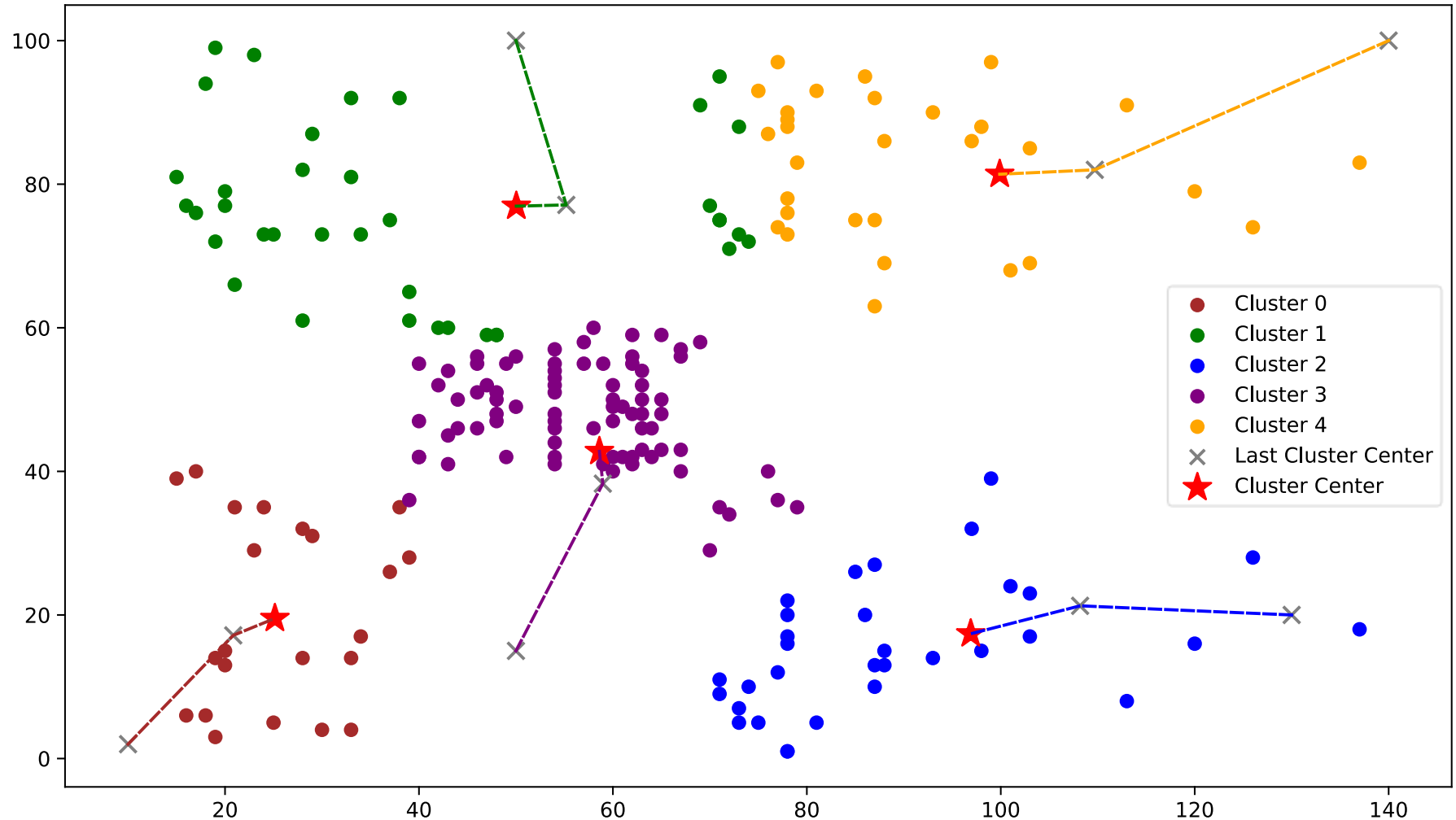
Iteration:1 Assignment and Move Cluster Centers Step



(Source : <https://satishgunjal.com/kmeans/>)

Αλγόριθμος k-means

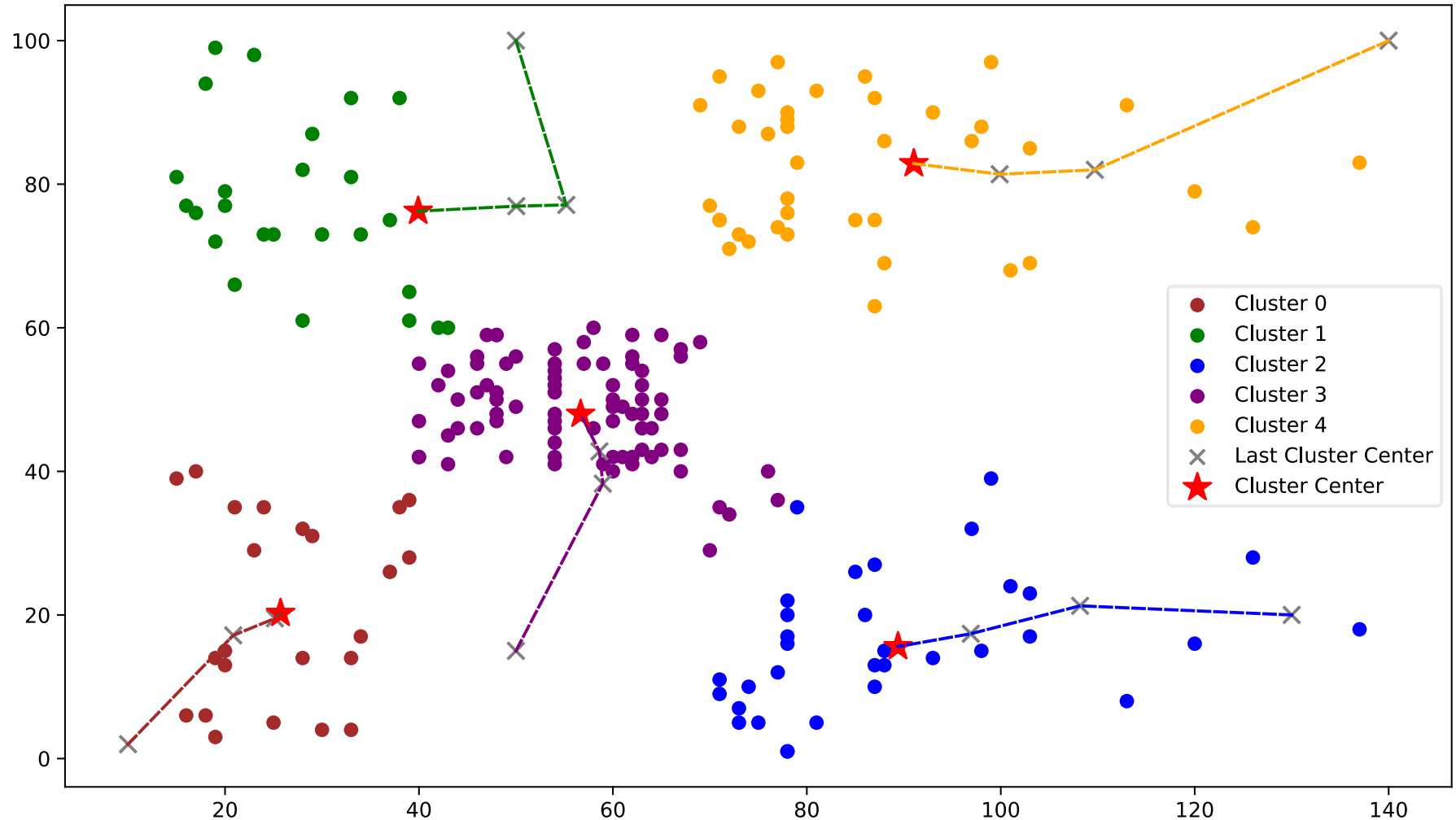
Iteration:2 Assignment and Move Cluster Centers Step



(Source : <https://satishgunjal.com/kmeans/>)

Αλγόριθμος k-means

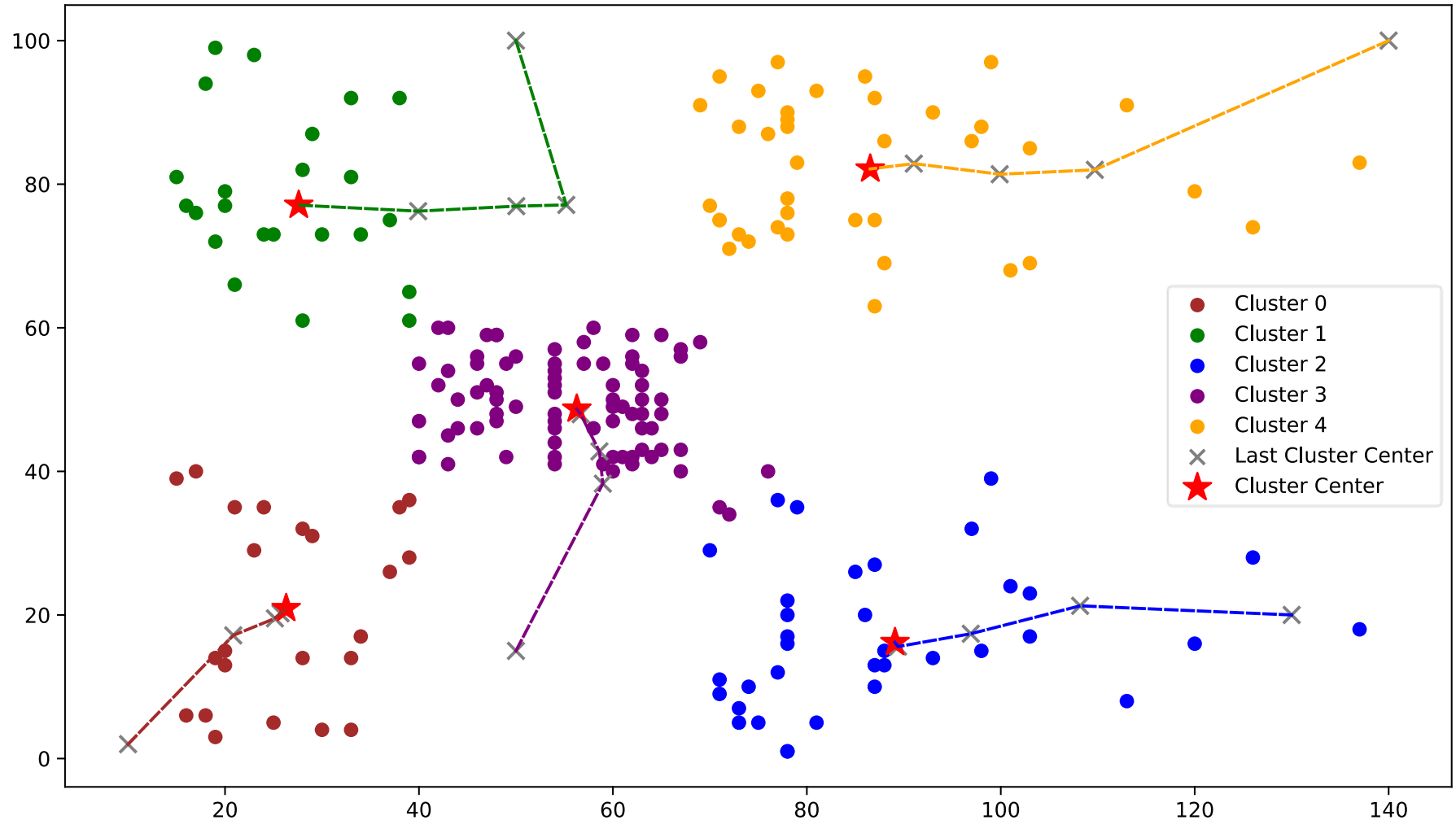
Iteration:3 Assignment and Move Cluster Centers Step



(Source : <https://satishgunjal.com/kmeans/>)

Αλγόριθμος k-means

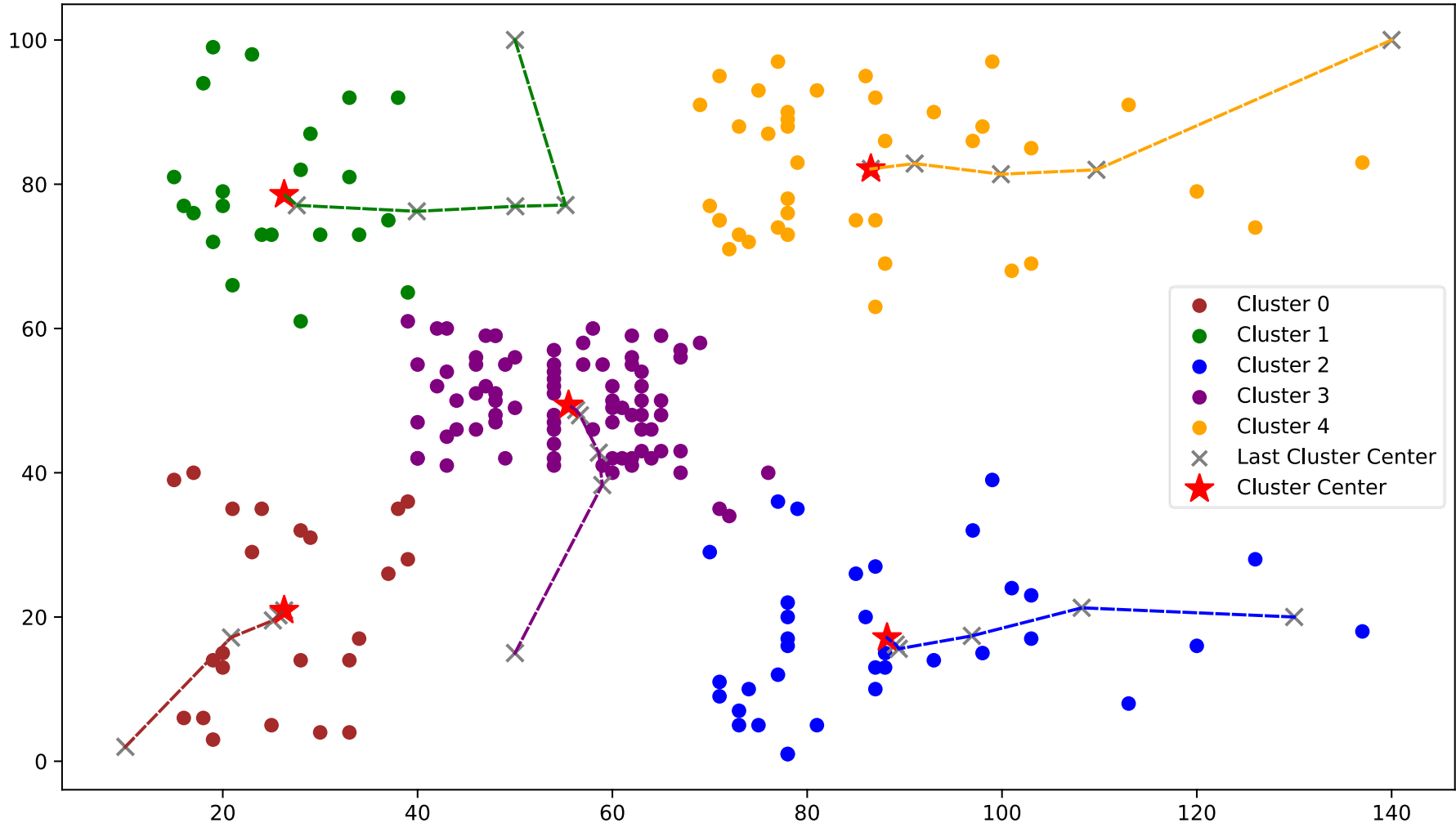
Iteration:4 Assignment and Move Cluster Centers Step



(Source : <https://satishgunjal.com/kmeans/>)

Αλγόριθμος k-means

Iteration:5 Assignment and Move Cluster Centers Step



(Source : <https://satishgunjal.com/kmeans/>)

Αλγόριθμος k-means -1- shared clusters

```
while(!convergence) {  
    for (i=0; i<N; i++) {  
        index = find_nearest_cluster(object[i])  
        ...  
        // update new cluster centers. protect update on  
        // shared "newClusterSize" & "newClusters" arrays  
        newClusterSize[index]++;  
        for (j=0; j<numCoords; j++)  
            newClusters[index][j] += objects[i][j];  
    }  
}
```

προσθέτουμε την συντεταγμένη j του object i στην συντεταγμένη j του cluster index. αυτό συμβαίνει ώστε στο τέλος να διαιρέσουμε με το μέγεθος του cluster για να υπολογίσουμε το μέσο όρο των points στο cluster ώστε να μετακινηθεί το κέντρο πιο κοντά στο cluster

```
// average sum and calculate new cluster centers  
for (k=0; k<numClusters; k++)  
    for (j=0; j<numCoords; j++)  
        Clusters[i][j] = newClusters[i][j] / newClusterSize[i];
```

Παρατηρήσεις:

1. Το 1ο for πρέπει να εκτελεστεί πριν το 3ο (σειριακότητα)
2. Τα ξεχωριστά iterations του 1ου είναι ανεξάρτητα μεταξύ τους (υπολογισμός των αποστάσεων των σημείων από τα κέντρα)
3. τα ξεχωριστά iterations του 3ου είναι ανεξάρτητα μεταξύ τους (επιλογή κέντρου για ενημέρωση των συντεταγμένων του)

- Τι μπορεί να παραλληλοποιηθεί?
- Μοιραζόμενες μεταβλητές newClusters και newClusterSize
 - Πώς θα εξασφαλίσουμε ορθότητα δεδομένων?

Αλγόριθμος k-means -2- copied clusters

```
while(!convergence) {
    for (i=0; i<N; i++) {
        index = find_nearest_cluster(object[i])
        ...
        // update new cluster centers.
        // *** replace global arrays with local (per-thread) ***
        newClusterSize[index]++;
        for (j=0; j<numCoords; j++)
            newClusters[index][j] += objects[i][j];
    }
    reduction for delta?
    // average sum and calculate new cluster centers
    for (k=0; k<numClusters; k++)
        for (j=0; j<numCoords; j++)
            Clusters[i][j] = newClusters[i][j] / newClusterSize[i];
}
```

Initialization: At the start of the parallel region, OpenMP initializes a private copy of the delta variable for each thread. The type of initialization depends on the reduction operation. For addition (+), each private copy is typically

- Ελαχιστοποίηση προσβάσεων σε μοιραζόμενα δεδομένα
- Δημιουργία τοπικών δομών για κάθε νήμα
- Μετά την εξέταση όλων των αντικειμένων, reduction από τοπικές στην κεντρική δομή
 - Από ένα νήμα

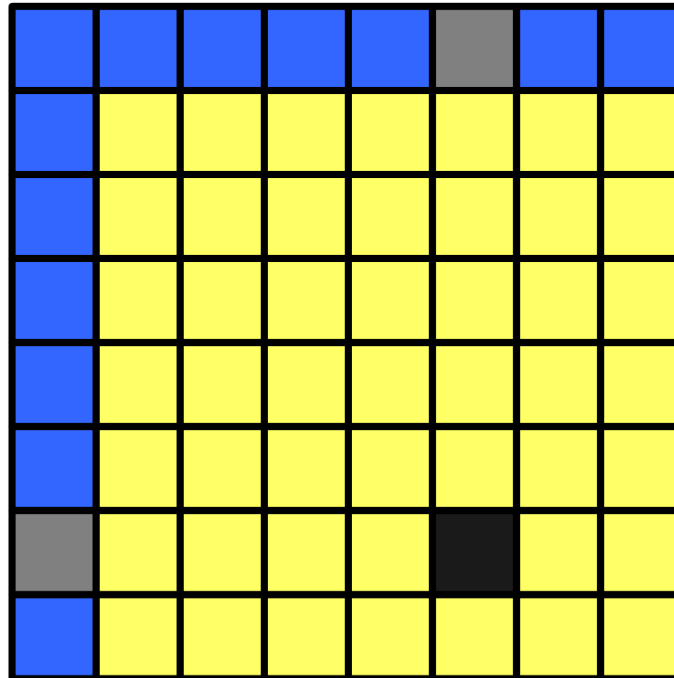
Αλγόριθμος Floyd-Warshall (FW)

- Εύρεση **ελάχιστου μονοπατιού** ανάμεσα σε οποιοδήποτε ζεύγος κόμβων ενός κατευθυνόμενου γράφου (τα βάρη των ακμών μπορούν να είναι και αρνητικά).

```
for (k=0; k<N; k++)  
  for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
      A[i][j] = min(A[i][j], A[i][k]+A[k][j]);
```

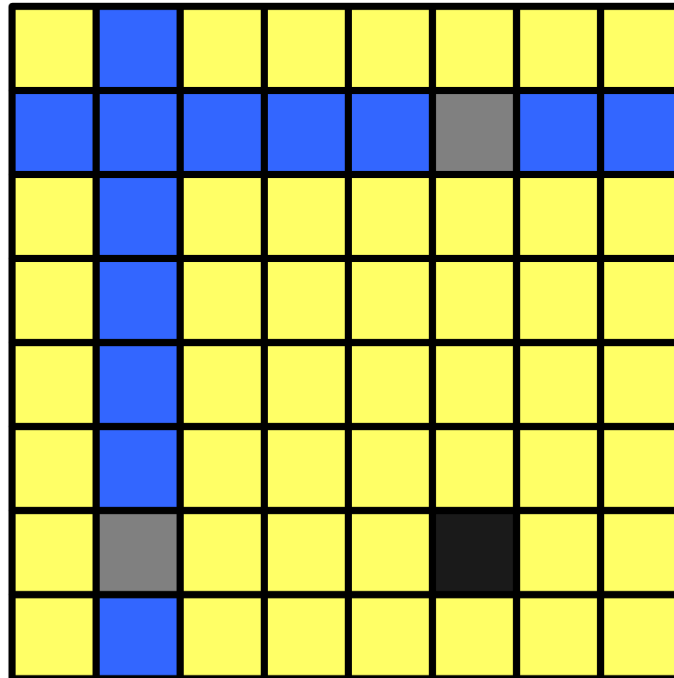
- Για κάθε χρονικό βήμα ***k*** υπολογίζει για κάθε ζεύγος κόμβων ***i-j*** αν υπάρχει συντομότερο μονοπάτι από τον *i* προς τον *j* περνώντας από το κόμβο *k*
- N: αριθμός κόμβων του γράφου
- A: πίνακας διπλανών κορυφών (αν ***i, j*** δεν συνδέονται τότε $A[i][j] = \infty$ αρχικά)
- Πολυπλοκότητα: $\Theta(n^3)$

k=0



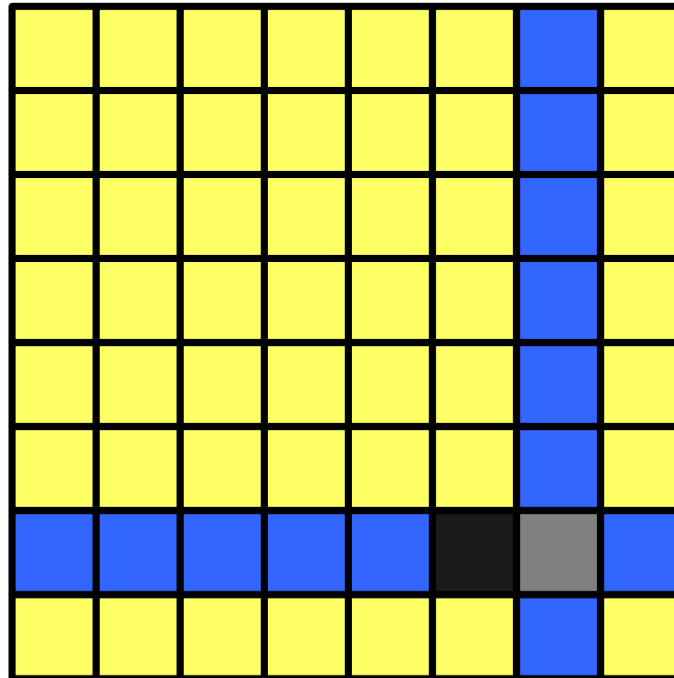
$$A[i][j] = \min(A[i][j], A[i][0] + A[0][j])$$

$k=1$



$$A[i][j] = \min(A[i][j], A[i][1] + A[1][j])$$

k=6

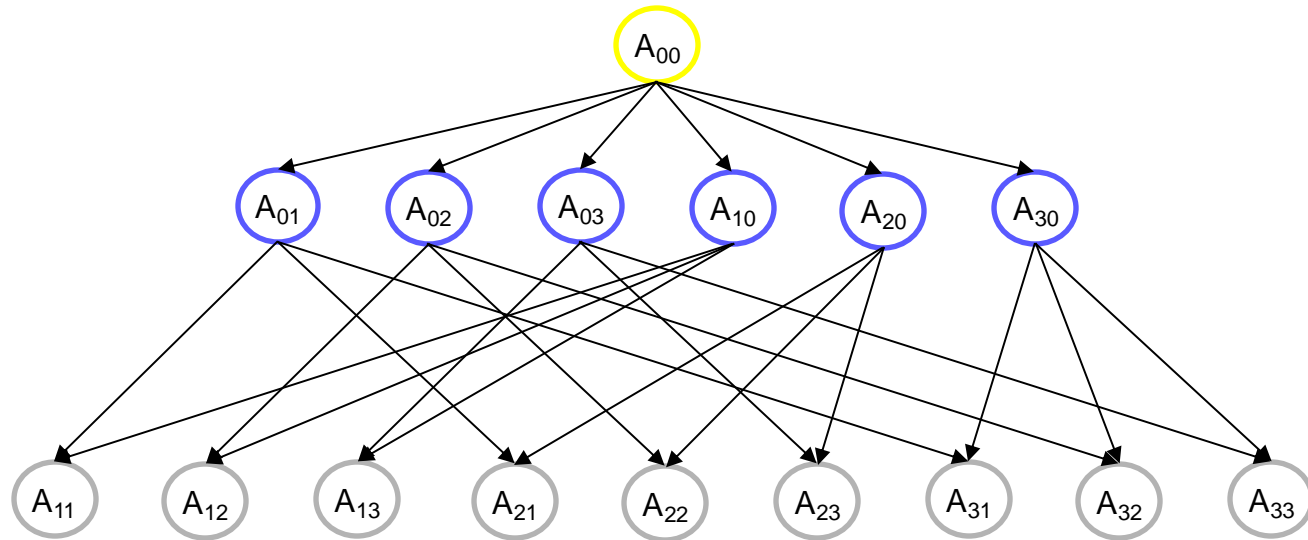


$$A[i][j] = \min(A[i][j], A[i][6] + A[6][j])$$

FW Task graph

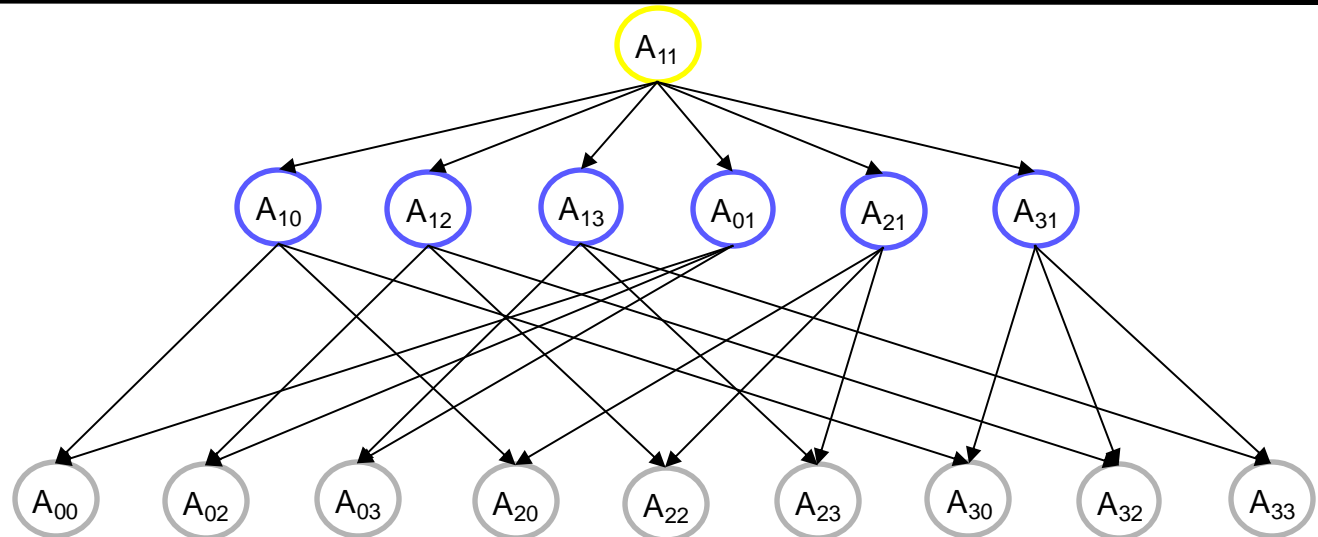
k=0

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}



k=1

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}



- Για μεγάλα N (ο A δεν χωράει στην cache), ο FW είναι memory bound:
 - Ο πίνακας A πρέπει να μεταφέρεται από την κύρια μνήμη σε κάθε επανάληψη k
 - Οι πράξεις που γίνονται είναι πολύ απλές (σύγκριση / πρόσθεση) σε ακέραιους ή πραγματικούς απλής ακρίβειας
- Παράλληλη εκτέλεση:
 - Τα loops i, j είναι παράλληλα
 - Ο αλγόριθμος δεν κλιμακώνει καλά σε αρχιτεκτονικές κοινής μνήμης

Εναλλακτικές υλοποιήσεις: recursive

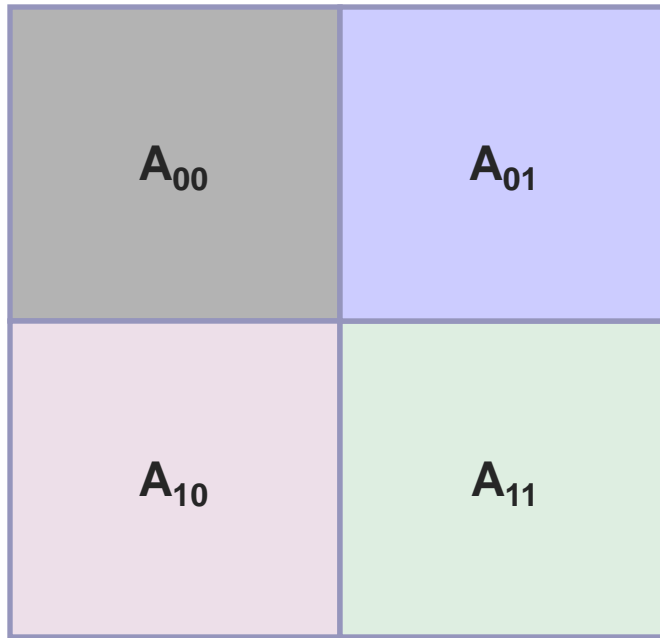
- J.-S. Park, M. Penner, and V. K. Prasanna, “**Optimizing Graph Algorithms for Improved Cache Performance,**” *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, VOL. 15, NO. 9, SEPTEMBER 2004.

```
FWR (A, B, C)
    if (base case)
        FWI (A, B, C)
    else
        FWR (A00, B00, C00);
        FWR (A01, B00, C01);
        FWR (A10, B10, C00);
        FWR (A11, B10, C01);
        FWR (A11, B10, C01);
        FWR (A10, B10, C00);
        FWR (A01, B00, C01);
        FWR (A00, B00, C00);

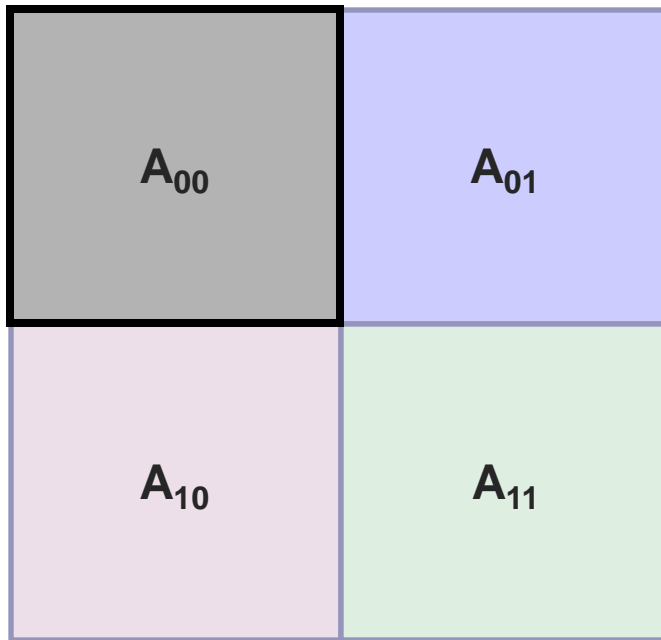
FWI (A, B, C)
    for (k=0; k<N; k++)
        for (i=0; i<N; i++)
            for (j=0; j<N; j++)
                A[i][j] = min(A[i][j], B[i][k]+C[k][j]);
```

- Καλείται ως: FWR(A, A, A);

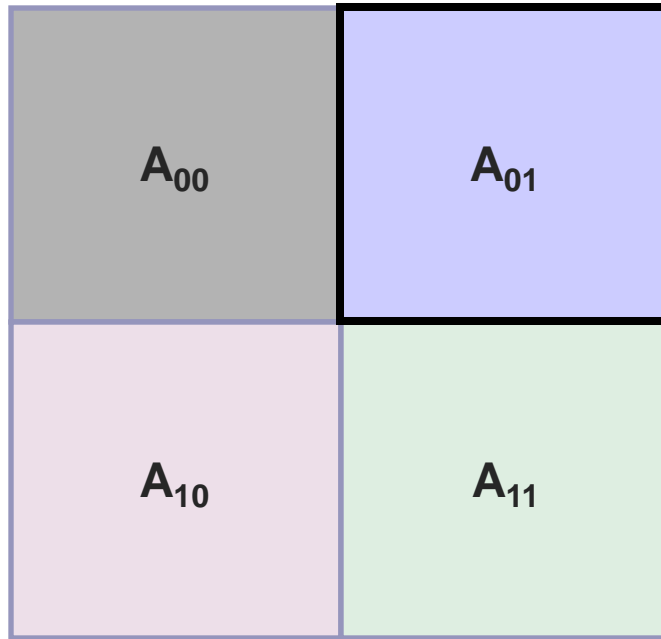
Εναλλακτικές υλοποιήσεις: recursive



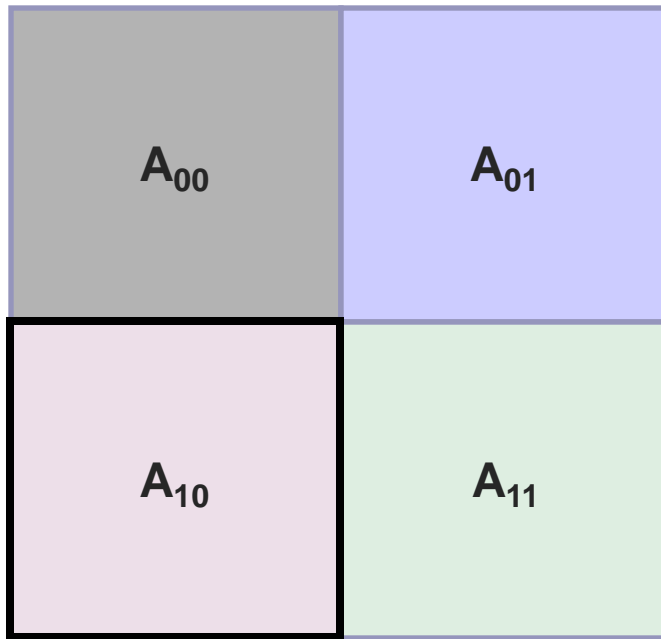
Εναλλακτικές υλοποιήσεις: recursive



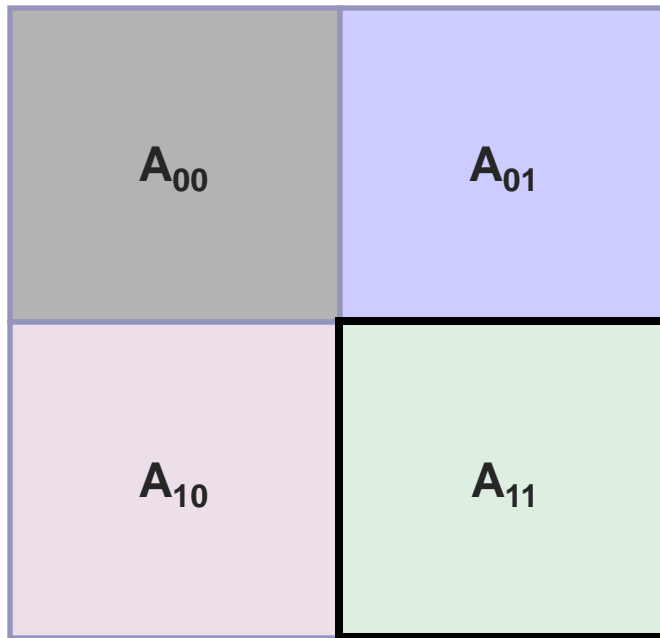
Εναλλακτικές υλοποιήσεις: recursive



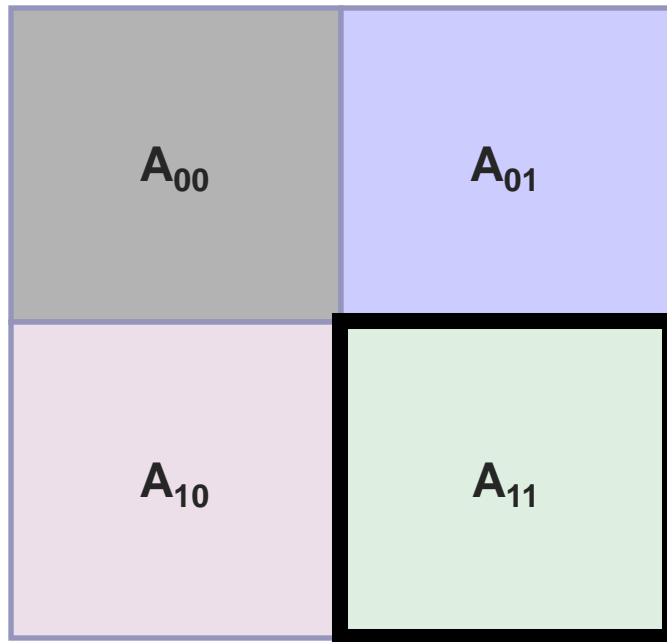
Εναλλακτικές υλοποιήσεις: recursive



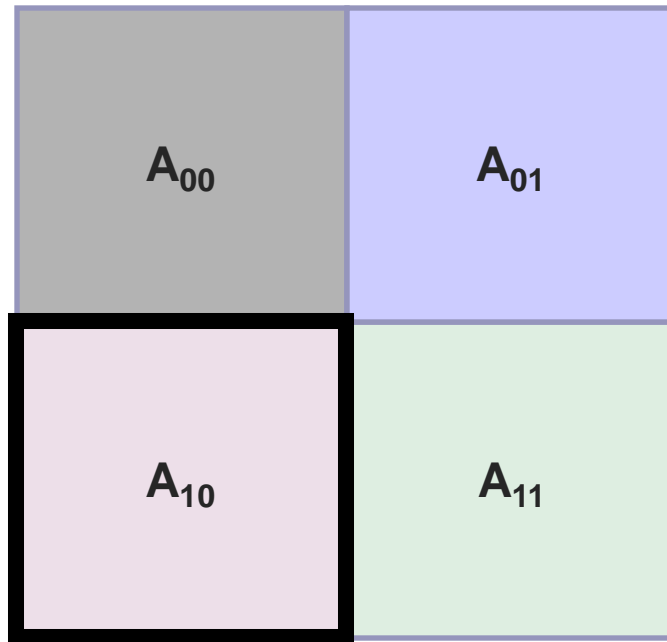
Εναλλακτικές υλοποιήσεις: recursive



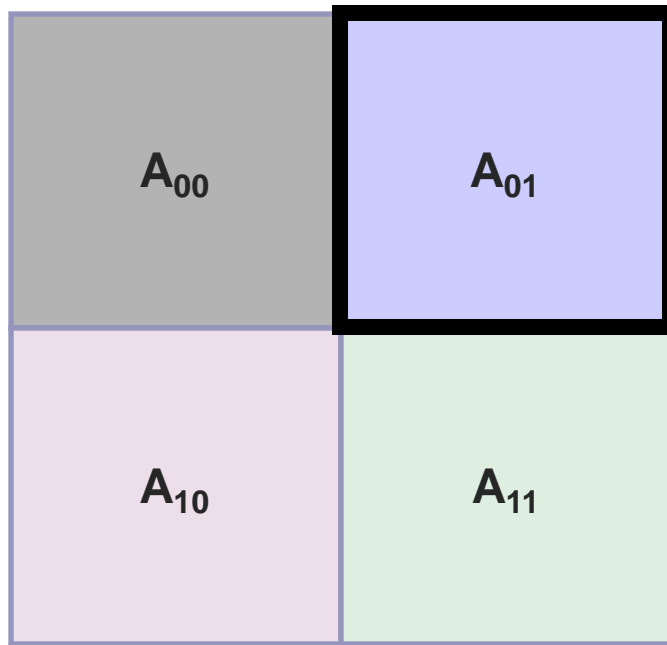
Εναλλακτικές υλοποιήσεις: recursive



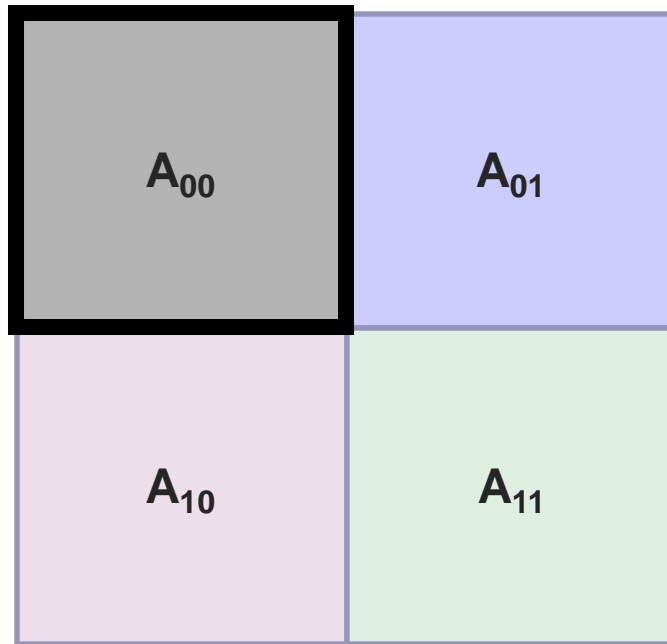
Εναλλακτικές υλοποιήσεις: recursive



Εναλλακτικές υλοποιήσεις: recursive



Εναλλακτικές υλοποιήσεις: recursive



Εναλλακτικές υλοποιήσεις: recursive

```
FWR (A, B, C)
  if (base case)
    FWI (A, B, C)
```

```
else
```

```
FWR (A00, B00, C00) ;
```

1

```
FWR (A01, B00, C01) ;
```

2

```
FWR (A10, B10, C00) ;
```

```
FWR (A11, B10, C01) ;
```

3

```
FWR (A11, B10, C01) ;
```

4

```
FWR (A10, B10, C00) ;
```

5

```
FWR (A01, B00, C01) ;
```

```
FWR (A00, B00, C00) ;
```

6

Παραλληλία

Εναλλακτικές υλοποιήσεις: tiled

1	2	2	2
2	3	3	3
2	3	3	3
2	3	3	3

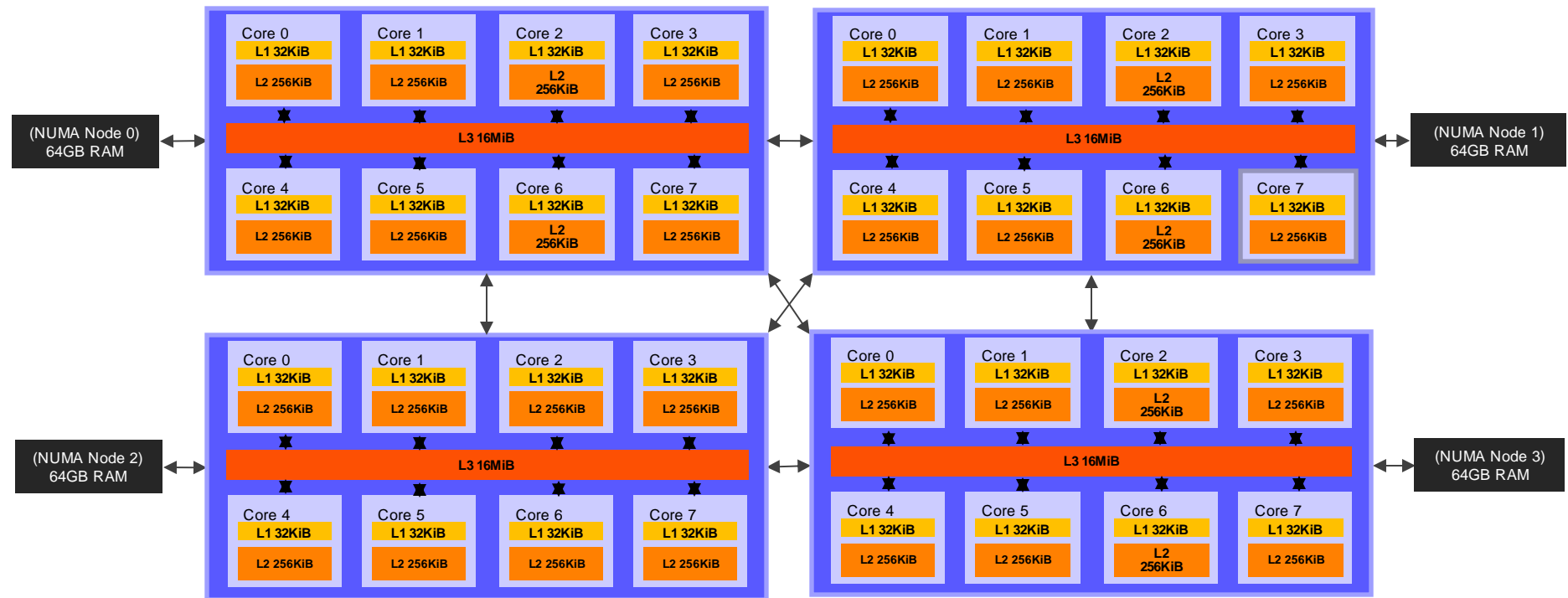
6	5	6	6
5	4	5	5
6	5	6	6
6	5	6	6

9	9	8	9
9	9	8	9
8	8	7	8
9	9	8	9

12	12	12	11
12	12	12	11
12	12	12	11
11	11	11	10

Περιβάλλον εκτέλεσης

- *sandman*: 4 x Intel Xeon E5-4620 (Sandy Bridge)
 - Συνολικά 32 πυρήνες (και 64 threads)



- Για χρήση του sandman:
 - `$ qsub -q serial -l nodes=sandman:ppn=64 <script>`
- **ΠΡΟΣΟΧΗ:** Χρησιμοποιείτε τα μηχανήματα της ουράς **parlab** για ανάπτυξη προγραμμάτων/έλεγχο ορθής λειτουργίας/εκσφαλμάτωση
- Μπορείτε να χρησιμοποιείτε τα μηχανήματα της ουράς parlab για την ανάπτυξη του παράλληλου κώδικα
- Θα βρείτε τον κώδικα της άσκησης στον scirouter στο path:
`/home/parallel/pps/2023-2024/a2/`