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

Παράλληλος προγραμματισμός:
Σχεδίαση και υλοποίηση παράλληλων προγραμμάτων

Συστήματα Παράλληλης Επεξεργασίας

9ο Εξάμηνο

National Technical University of Athens
CSLab

Σύνοψη παρουσίασης

- Παράλληλες υπολογιστικές πλατφόρμες
  - PRAM: Η ιδανική παράλληλη πλατφόρμα
  - Η ταξινόμηση του Flynn
  - Συστήματα κοινής μνήμης
  - Συστήματα κατανεμημένης μνήμης
- Ανάλυση παράλληλων προγραμμάτων
  - Μετρικές αξιολόγησης επίδοσης
  - Ο νόμος του Amdahl
  - Μοντελοποίηση παράλληλων προγραμμάτων
- **Σχεδίαση παράλληλων προγραμμάτων**
  - **Κατανομή υπολογισμών σε υπολογιστικές εργασίες (tasks)**
  - **Ορισμός ορθής σειράς εκτέλεσης (χρονοδρομολόγηση)**
  - **Οργάνωση πρόσβασης στα δεδομένα (συγχρονισμός / επικοινωνία)**
  - **Ανάθεση εργασιών (απεικόνιση) σε οντότητες εκτέλεσης (processes, threads)**

Σύνοψη παρουσίασης

- Παράλληλα προγραμματιστικά μοντέλα
  - Κοινού χώρου διευθύνσεων
  - Ανταλλαγής μηνυμάτων
- Παράλληλες προγραμματιστικές δομές
  - SPMD
  - fork / join
  - task graphs
  - parallel for
- Γλώσσες και εργαλεία
  - POSIX threads, MPI, OpenMP, Cilk, Cuda, Γλώσσες PGAS
- Αλληλεπίδραση με το υλικό
  - Συστήματα κοινής μνήμης
  - Συστήματα κατανεμημένης μνήμης και υβριδικά

Σχεδιασμός παράλληλων προγραμμάτων

- **Στόχος:** Η μετατροπή ενός σειριακού αλγορίθμου σε παράλληλο
  - Κατανομή υπολογισμών σε υπολογιστικές εργασίες (tasks)
  - Ορισμός ορθής σειράς εκτέλεσης (χρονοδρομολόγηση)
  - Διαμοιρασμός των δεδομένων - Οργάνωση πρόσβασης στα δεδομένα (συγχρονισμός / επικοινωνία)
  - Ανάθεση εργασιών (απεικόνιση) σε οντότητες εκτέλεσης (processes, threads)
- Δεν υπάρχει αυστηρά ορισμένη μεθοδολογία για το σχεδιασμό και την υλοποίηση παράλληλων προγραμμάτων
- “It’s more art than science”
- Ιδανικά θα θέλαμε ο σχεδιασμός και η υλοποίηση να λαμβάνουν χώρα ανεξάρτητα. Στην πράξη υπάρχει αλληλεπίδραση, π.χ. τον σχεδιασμό επηρεάζουν:
  - Η αρχιτεκτονική της πλατφόρμας εκτέλεσης
  - Τα υποστηριζόμενα προγραμματιστικά μοντέλα
  - Οι δυνατότητες του περιβάλλοντος υλοποίησης

Σχεδιασμός παράλληλων προγραμμάτων

- Μετρικές αξιολόγησης:
  - **Επίδοση** του παραγόμενου κώδικα
    - Επιτάχυνση
    - Αποδοτικότητα
    - Κλιμακωσιμότητα
  - **Παραγωγικότητα** (code productivity)
    - Χαμηλός χρόνος υλοποίησης
    - Μεταφερσιμότητα (portability)
    - Ευκολία στη συντήρηση (maintainability)
  - **Κατανάλωση ενέργειας / ισχύος**

Στάδια σχεδιασμού

- **Στάδιο 1:** Κατανομή υπολογισμών
- **Στάδιο 2:** Ορισμός ορθής σειράς εκτέλεσης
- **Στάδιο 3:** Οργάνωση πρόσβασης στα δεδομένα
- **Στάδιο 4:** Ανάθεση εργασιών σε οντότητες εκτέλεσης

Στάδιο 1: Κατανομή υπολογισμών

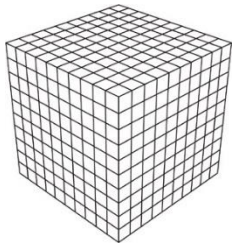
- **Στόχος:** κατανομή σε υπολογιστικές εργασίες (tasks)
- Σε τι να εστιάσω **πρώτα**; Στους υπολογισμούς ή τα δεδομένα;
- 2 βασικές προσεγγίσεις:
  - **task centric:** πρώτα μοιράζονται οι υπολογισμοί και στη συνέχεια τα δεδομένα
  - **data centric:** πρώτα μοιράζονται τα δεδομένα εισόδου και στη συνέχεια οι υπολογισμοί
- Και στις 2 περιπτώσεις θα προκύψουν εργασίες (tasks)
- **Παράδειγμα:** Πολλαπλασιασμός πίνακα με διάνυσμα ($y = A * x$)
  - task centric: *π.χ. 1 task = υπολογισμός της τιμής του* y_i *:*
    - *Ποια δεδομένα χρειάζεται αυτό το task;*
  - data centric: π.χ. μοιράζω τον πίνακα Α κατά γραμμές, οπότε *1 task = υπολογισμοί που εμπλέκουν 1 γραμμή του πίνακα Α*
    - *Ποιοι υπολογισμοί εμπλέκουν τη συγκεκριμένη γραμμή;*
    - *Ποια άλλα δεδομένα χρειάζεται το συγκεκριμένο task;*
- Ειδική περίπτωση της task centric προσέγγισης είναι η **function centric**
  - Η κατανομή γίνεται με βάση τις συναρτήσεις (διαφορετικές λειτουργίες/στάδια)
  - Είναι κατάλληλη για εφαρμογές που αποτελούνται από διαδοχικά στάδια υπολογισμών που εφαρμόζονται σε ροές δεδομένων

Στάδιο 1: Κατανομή υπολογισμών

- Επιλογή στρατηγικής ανάλογα με τον αλγόριθμο
- Η task centric προσέγγιση είναι η πιο γενική
- Οι task centric και data centric μπορεί να οδηγήσουν σε ακριβώς την ίδια κατανομή
- Απαιτήσεις:
  - Επίδοση
    - Μεγιστοποίηση παραλληλίας
    - Ελαχιστοποίηση επιβαρύνσεων λόγω διαχείρισης tasks, συγχρονισμού, επικοινωνίας
    - Ισοκατανομή φορτίου
  - Απλότητα
    - Μικρός χρόνος υλοποίησης
    - Ευκολία στην επέκταση και τη συντήρηση
  - Ευελιξία (ανεξαρτησία από απαιτήσεις υλοποίησης)

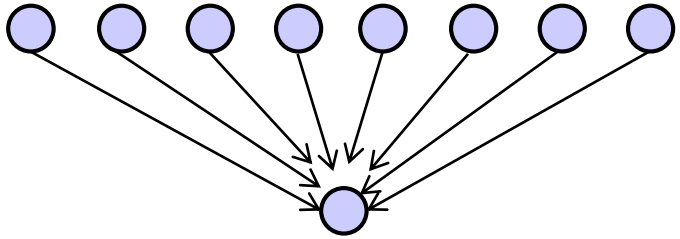
Rule of thumb

- Όταν το πρόβλημα είναι “embarrassingly parallel” (π.χ. συμπίεση Ν αρχείων, επίλυση Ν ανεξάρτητων συστημάτων) η data centric προσέγγιση είναι προφανής.
  - Οδηγεί σε **data parallel** υλοποιήσεις
- Ταιριάζει στη λογική **map-reduce**
- Αλγόριθμοι σε κανονικές δομές δεδομένων (π.χ. regular computational grids, αλγεβρικοί πίνακες) συχνά ευνοούν τη **data centric** κατανομή:
  - Πολλαπλασιασμός πινάκων (γενικά βασικές αλγεβρικές ρουτίνες)
  - Επίλυση γραμμικών συστημάτων
  - “Stencil computations”



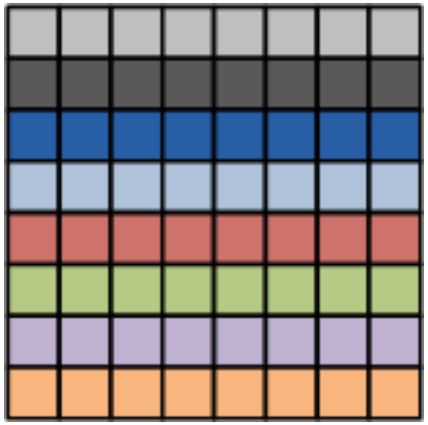
- Όταν μπορεί να εφαρμοστεί, η data centric προσέγγιση οδηγεί σε κομψές λύσεις
  - *Δοκιμάζουμε πρώτα την data centric και αν δεν πετύχει, καταφεύγουμε στην task centric*

$$P_{\text{Jac}} = \begin{pmatrix} -\frac{1749}{2} & \frac{2485}{2} & 73 & \frac{3842}{5} & -\frac{40721}{4} & -\frac{1054231}{250} & \frac{735629}{100} & -\frac{128827971}{6250} \\ \frac{1450}{3} & -\frac{691}{6} & \frac{2021}{12} & -\frac{26039}{150} & \frac{196313}{30} & -\frac{2540087}{1875} & -\frac{14477359}{1500} & \frac{487255997}{31250} \\ \frac{747}{2} & -\frac{2115}{2} & -\frac{199}{2} & -291 & \frac{24885}{4} & \frac{175083}{50} & -\frac{109947}{20} & \frac{14131653}{1250} \\ \frac{1391}{3} & -\frac{3761}{6} & -\frac{8363}{12} & -\frac{9011}{30} & \frac{35339}{6} & \frac{1302562}{375} & \frac{307709}{300} & \frac{52789903}{6250} \\ \frac{620}{3} & -\frac{815}{6} & -\frac{9395}{12} & -\frac{209503}{150} & \frac{63548}{15} & \frac{2706476}{1875} & \frac{4732207}{1500} & \frac{386472719}{31250} \\ \frac{1031}{6} & -\frac{674}{3} & -\frac{5911}{12} & -\frac{13333}{30} & \frac{10199}{12} & \frac{1417027}{750} & \frac{232033}{75} & \frac{13882747}{3125} \\ \frac{343}{6} & \frac{80}{3} & -\frac{4919}{12} & -\frac{28789}{150} & \frac{19211}{60} & -\frac{2996549}{3750} & \frac{1232629}{375} & \frac{54383186}{15625} \\ \frac{200}{3} & -\frac{109}{3} & -\frac{145}{6} & -\frac{15569}{75} & \frac{3473}{15} & \frac{425546}{1875} & -\frac{56839}{750} & \frac{44325837}{15625} \end{pmatrix}$$



Διαμοιρασμός κανονικών δομών

1-dimensional
(ανά γραμμή/στήλη)



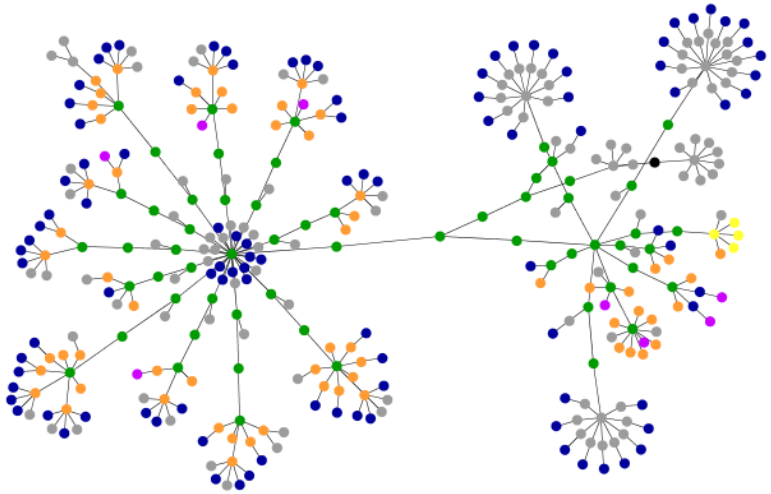
+ Απλή στην υλοποίηση
- Περιοριστική
(το task που προκύπτει εξαρτάται από το N του πίνακα)

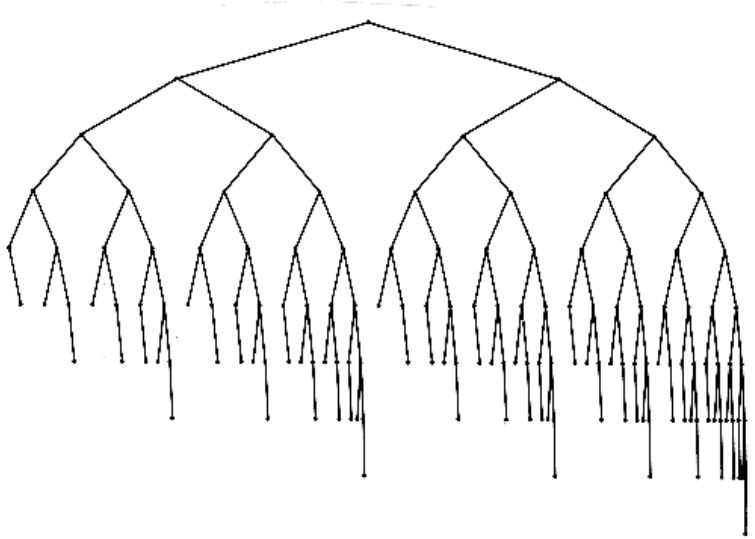
2-dimensional
(ανά μπλοκ)

- Δυσκολότερη στην υλοποίηση
+ Πιο ευέλικτη
(ο προγραμματιστής ελέγχει το μέγεθος του task)

Rule of thumb

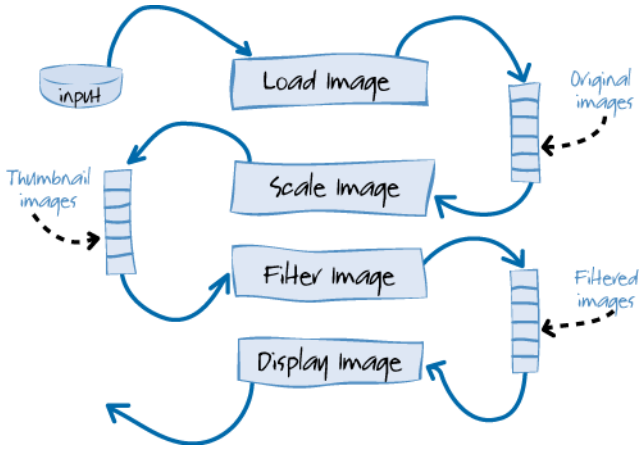
- Η **task centric** προσέγγιση αποτελεί λύση για πιο σύνθετους ή δυναμικούς αλγορίθμους σε ακανόνιστες δομές δεδομένων (irregular data structures)
  - Αλγόριθμοι σε λίστες, δέντρα, γράφους, κλπ
  - Αναδρομικοί αλγόριθμοι
  - Event-based εφαρμογές
- Οδηγεί σε πιο λεπτομερή και ακριβέστερη αναπαράσταση του παραλληλισμού





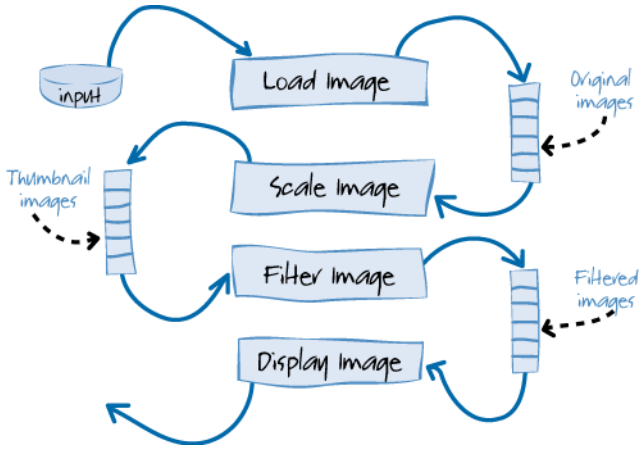
Rule of thumb

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση
  - Η παραλληλοποίηση σε κάθε φάση με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης)

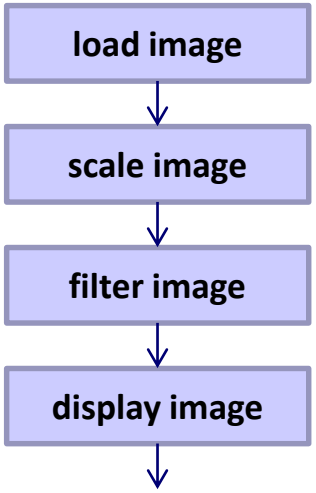


Rule of thumb

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση
  - Η παραλληλοποίηση σε κάθε φάση με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης)

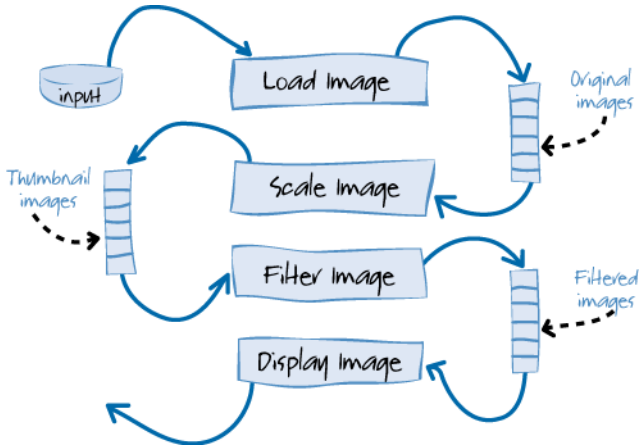


data centric:
π.χ. 32 threads ανά φάση

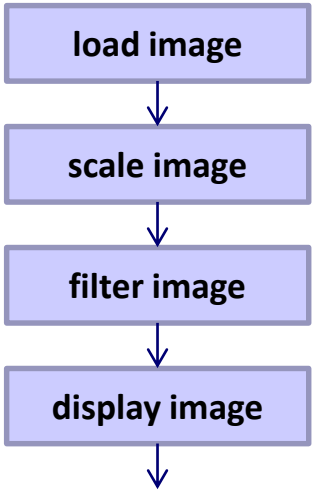


Rule of thumb

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση
  - Η παραλληλοποίηση σε κάθε φάση με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης)



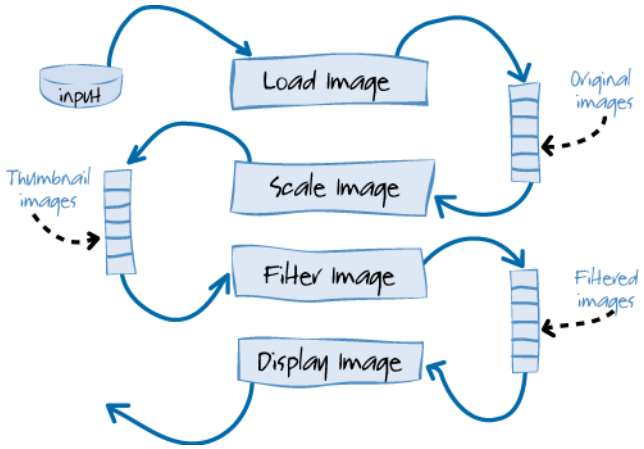
data centric:
π.χ. 32 threads ανά φάση



Αν κάθε φάση κλιμακώνει μέχρι τα 8 threads, έχουμε προφανή σπατάλη πόρων

Rule of thumb

- Η **function centric** προσέγγιση μπορεί να υιοθετηθεί όταν υπάρχουν διακριτές φάσεις και «ροή δεδομένων»
  - Υπάρχει κάποιου είδους υποστήριξη στο υλικό για κάθε φάση
  - Η παραλληλοποίηση σε κάθε φάση με την data centric προσέγγιση δεν οδηγεί σε ικανοποιητική αξιοποίηση των πόρων (π.χ. δεν κλιμακώνει στο διαθέσιμο αριθμό πυρήνων λόγω συμφόρησης στο διάδρομο μνήμης)



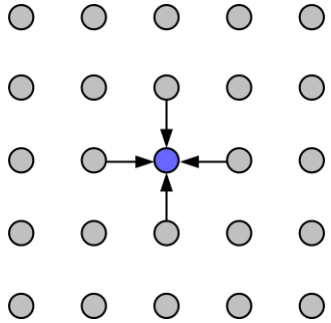
load image (#4)	scale image (#3)	filter image (#2)	display image (#1)
8 threads	8 threads	8 threads	8 threads

load image (#5)	scale image (#4)	filter image (#3)	display image (#2)
8 threads	8 threads	8 threads	8 threads

Παράδειγμα: Εξίσωση θερμότητας

- Αλγόριθμος του Jacobi (2-διάστατο πλέγμα X * Y)

```
A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] +
                       A[step][i+1][j] + A[step][i][j-1] +
                       A[step][i][j+1])
```



Data centric

- Μοιράζουμε τα δεδομένα ανά σημείο / γραμμή / στήλη / μπλοκ
- Κάθε task αναλαμβάνει να υπολογίσει την τιμή σε κάθε διαμοιρασμένο δεδομένο

Task centric

- **1 task =** υπολογισμός θερμότητας για κάθε σημείο του χωρίου
- **1 task =** υπολογισμός θερμότητας για κάθε χρονικό βήμα
- **1 task =** υπολογισμός θερμότητας για μία γραμμή / στήλη / block του δισδιάστατου πλέγματος για όλες τις χρονικές στιγμές
- **1 task =** υπολογισμός θερμότητας για μία γραμμή / στήλη / block του δισδιάστατου πλέγματος για μία χρονική στιγμή

Στάδιο 2: Ορισμός ορθής σειράς εκτέλεσης

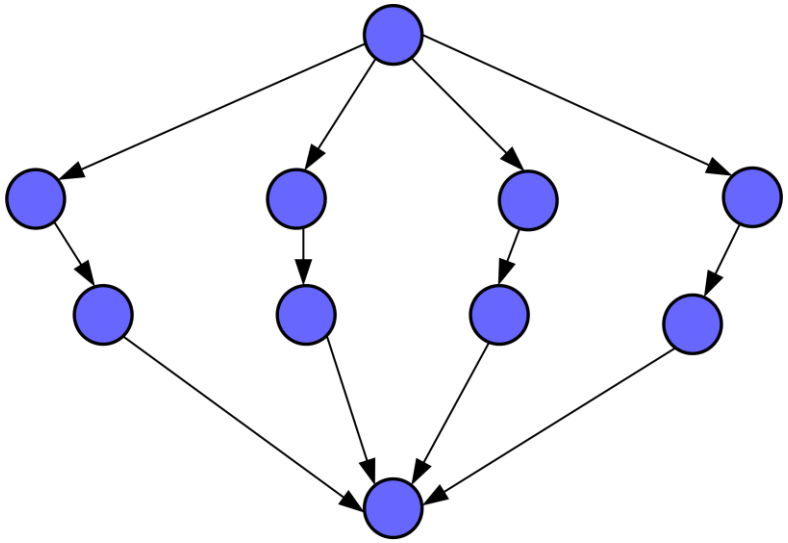
- Οι εργασίες που ορίστηκαν στο προηγούμενο στάδιο πρέπει να μπουν στη σωστή σειρά ώστε να εξασφαλίζεται η ίδια σημασιολογία με το σειριακό πρόγραμμα
- Το στάδιο αυτό συχνά αναφέρεται και ως χρονοδρομολόγηση = ανάθεση εργασιών σε χρονικές στιγμές
- Απαιτείται εντοπισμός των **εξαρτήσεων** ανάμεσα στις εργασίες και κατάστρωση του γράφου των εξαρτήσεων (task dependence graph)

Εξαρτήσεις

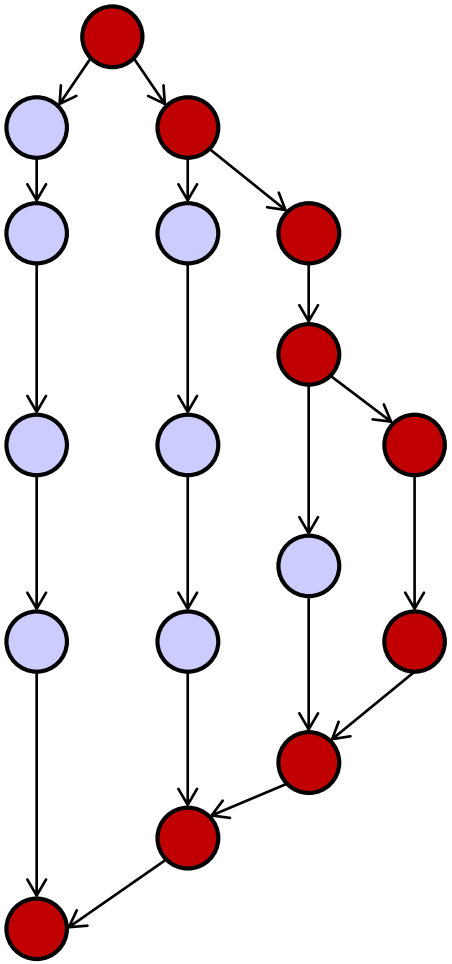
- Εξαρτήσεις δεδομένων υπάρχουν ήδη από το σειριακό πρόγραμμα
- Εξάρτηση υπάρχει όταν 2 εντολές αναφέρονται στα ίδια δεδομένα (θέση μνήμης)
- 4 είδη εξαρτήσεων
  - Read-After-Write (RAW) ή true dependence
  - Write-After-Read (WAR) ή anti dependence
  - Write-After-Write (WAW) ή output dependence
  - Read-After-Read (not really a dependence)
- Η παράλληλη εκτέλεση πρέπει να σεβαστεί τις εξαρτήσεις του προβλήματος
- Η κατανομή των tasks δημιουργεί κατά κανόνα εξαρτήσεις ανάμεσα στα tasks (π.χ. το task1 πρέπει να διαβάσει δεδομένα που παράγει το task2)
- Διατήρηση των εξαρτήσεων: **σειριοποίηση** μεταξύ tasks

Γράφος εξαρτήσεων (task dependence graph)

- Ή απλά **task graph**
- Κορυφές: tasks
  - **Label κορυφής**: κόστος υπολογισμού του task
- Ακμές: εξαρτήσεις ανάμεσα στα tasks
  - **Βάρος ακμής**: όγκος δεδομένων που πρέπει να μεταφερθούν (στην περίπτωση της επικοινωνίας)



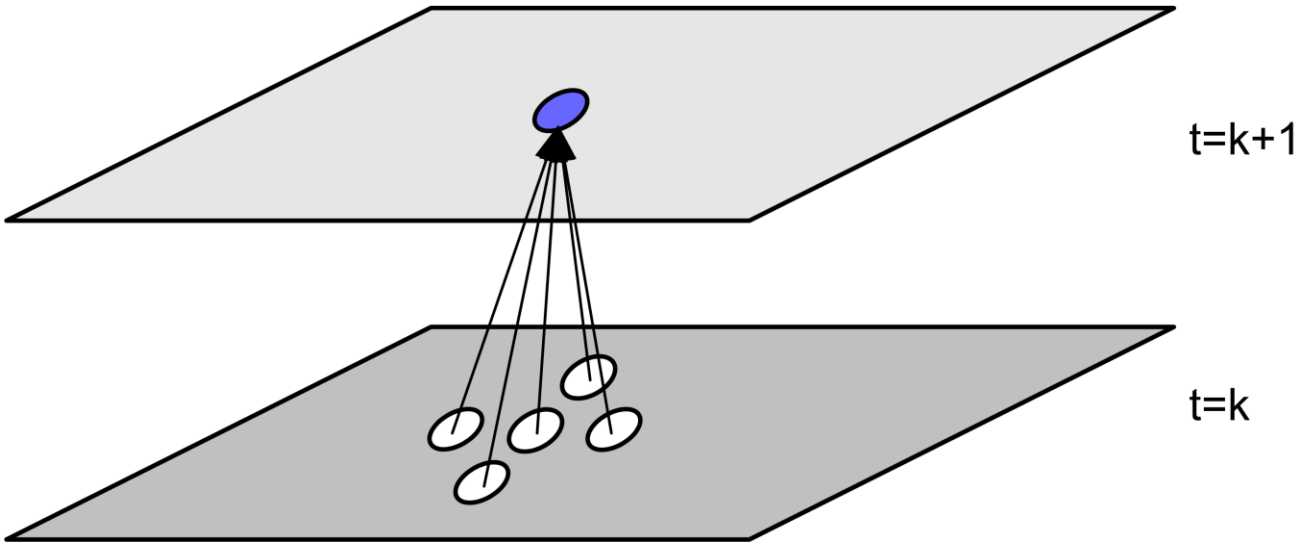
Task graphs: βασικές ιδιότητες



- **T_1:** Συνολική εργασία **(work)**, ο χρόνος που απαιτείται για την εκτέλεση σε 1 επεξεργαστή
- **T_p:** Χρόνος εκτέλεσης σε ***p*** επεξεργαστές
- Κρίσιμο μονοπάτι (critical path): Το μέγιστο μονοπάτι ανάμεσα στην πηγή και τον προορισμό του γράφου
- **T_∞ :** Χρόνος εκτέλεσης σε **∞** επεξεργαστές **(span)** και χρόνος εκτέλεσης του κρίσιμου μονοπατιού
- Ισχύει:
  - **$T_1 / p \leq T_p$**
  - **$T_\infty \leq T_p$**
  - **$T_p \leq T_1 / p + T_\infty$ (Brent's law)**
  - **Μέγιστο speedup T_1 / T_∞**

Εξαρτήσεις στην εξίσωση θερμότητας

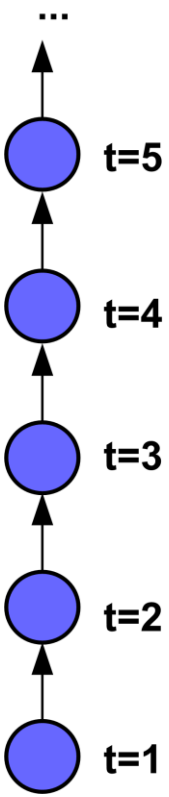
- 1 task = υπολογισμός θερμότητας για κάθε σημείο του χωρίου




```
A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] + A[step][i+1][j]
   + A[step][i][j-1] + A[step][i][j+1])
```

Εξαρτήσεις στην εξίσωση θερμότητας

- 1 task = υπολογισμός θερμότητας για κάθε χρονικό βήμα

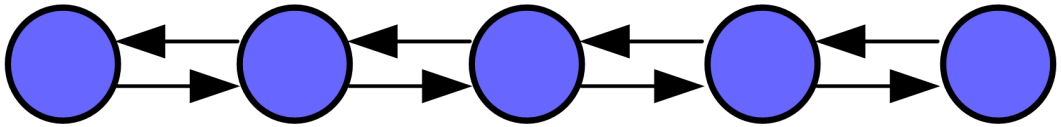


Δεν υπάρχουν ταυτόχρονα tasks!

```
A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] + A[step][i+1][j]
   + A[step][i][j-1] + A[step][i][j+1])
```

Εξαρτήσεις στην εξίσωση θερμότητας

- 1 task = υπολογισμός θερμότητας για μία γραμμή του δισδιάστατου πλέγματος για όλες τις χρονικές στιγμές

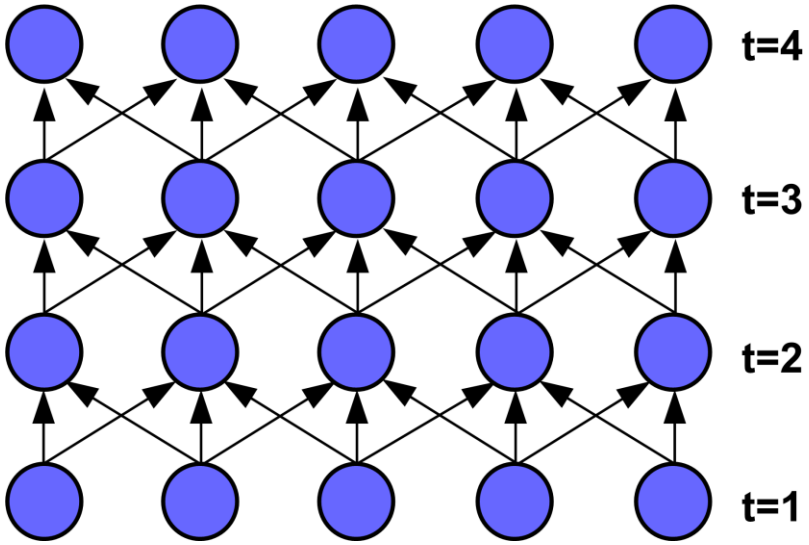


Δεν υπάρχει έγκυρη παράλληλη εκτέλεση!

```
A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] + A[step][i+1][j]
   + A[step][i][j-1] + A[step][i][j+1])
```

Εξαρτήσεις στην εξίσωση θερμότητας

- 1 task = υπολογισμός θερμότητας για μία γραμμή του δισδιάστατου πλέγματος για μία χρονική στιγμή




```
A[step+1][i][j] = 1/5 (A[step][i][j] + A[step][i-1][j] + A[step][i+1][j]
   + A[step][i][j-1] + A[step][i][j+1])
```

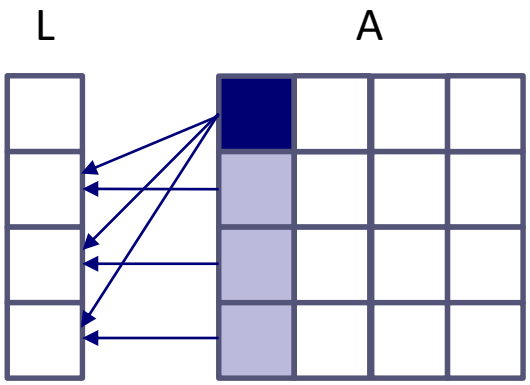
Παράδειγμα: Task graph για LU decomposition

```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      L[i] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - L[i] * A[k][j];
   }
```

- **Task centric προσέγγιση:** 1 task = μία στοιχειώδης αλγεβρική πράξη
- Data centric προσέγγιση: μοιράζω όλα στοιχεία των πινάκων L και A. 1 task = ό,τι χρειάζεται για να υπολογιστεί ένα στοιχείο.
- Σημείωση: Οι δύο προσεγγίσεις στο συγκεκριμένο τμήμα κώδικα είναι ισοδύναμες.
  - Υπάρχει έκδοση του υπολογιστικού πυρήνα όπου αντί για `L[i] = A[i][k] / A[k][k]` υπολογίζεται `A[i][k] = A[i][k] / A[k][k]` (βλέπε συνέχεια).

Παράδειγμα: Task graph για LU decomposition

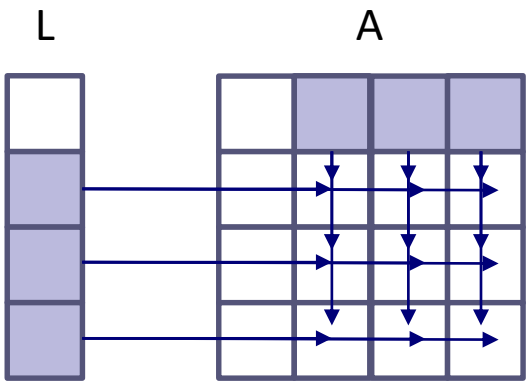
```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      L[i] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - L[i] * A[k][j];
   }
```



k = 0

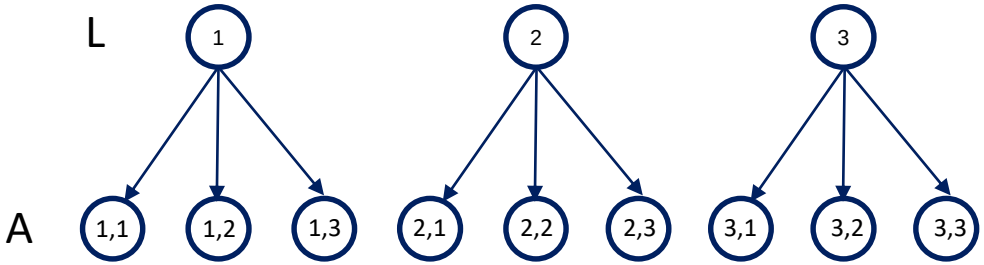
Παράδειγμα: Task graph για LU decomposition

```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      L[i] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - L[i] * A[k][j];
   }
```



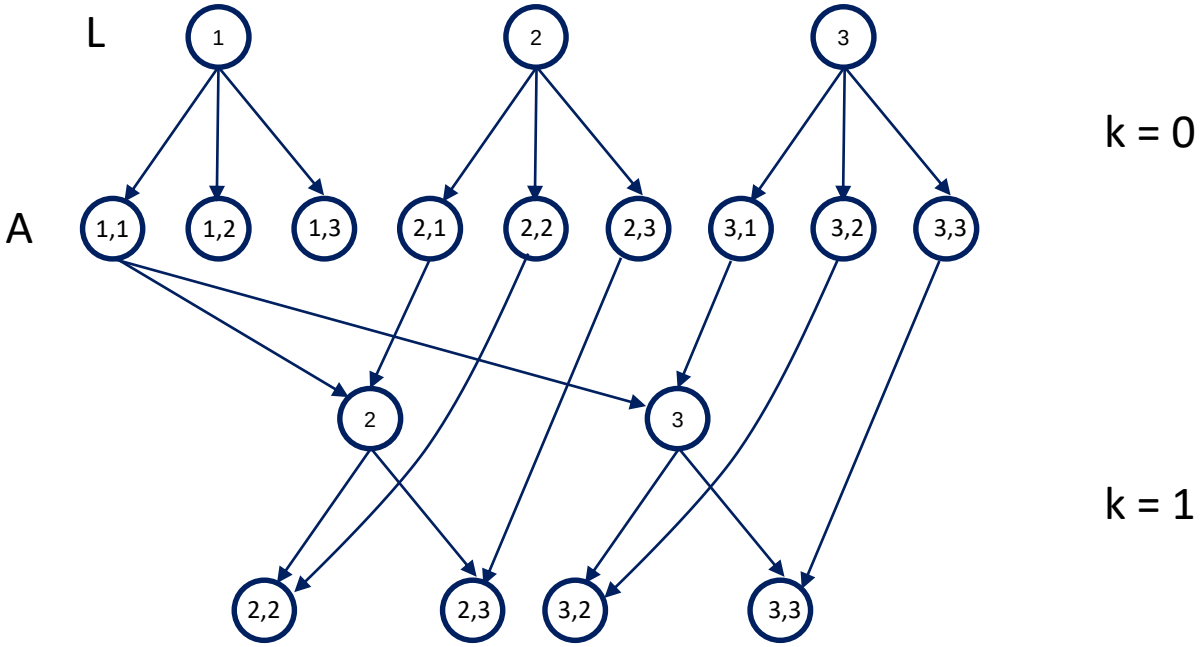
k = 0

Παράδειγμα: Task graph για LU decomposition

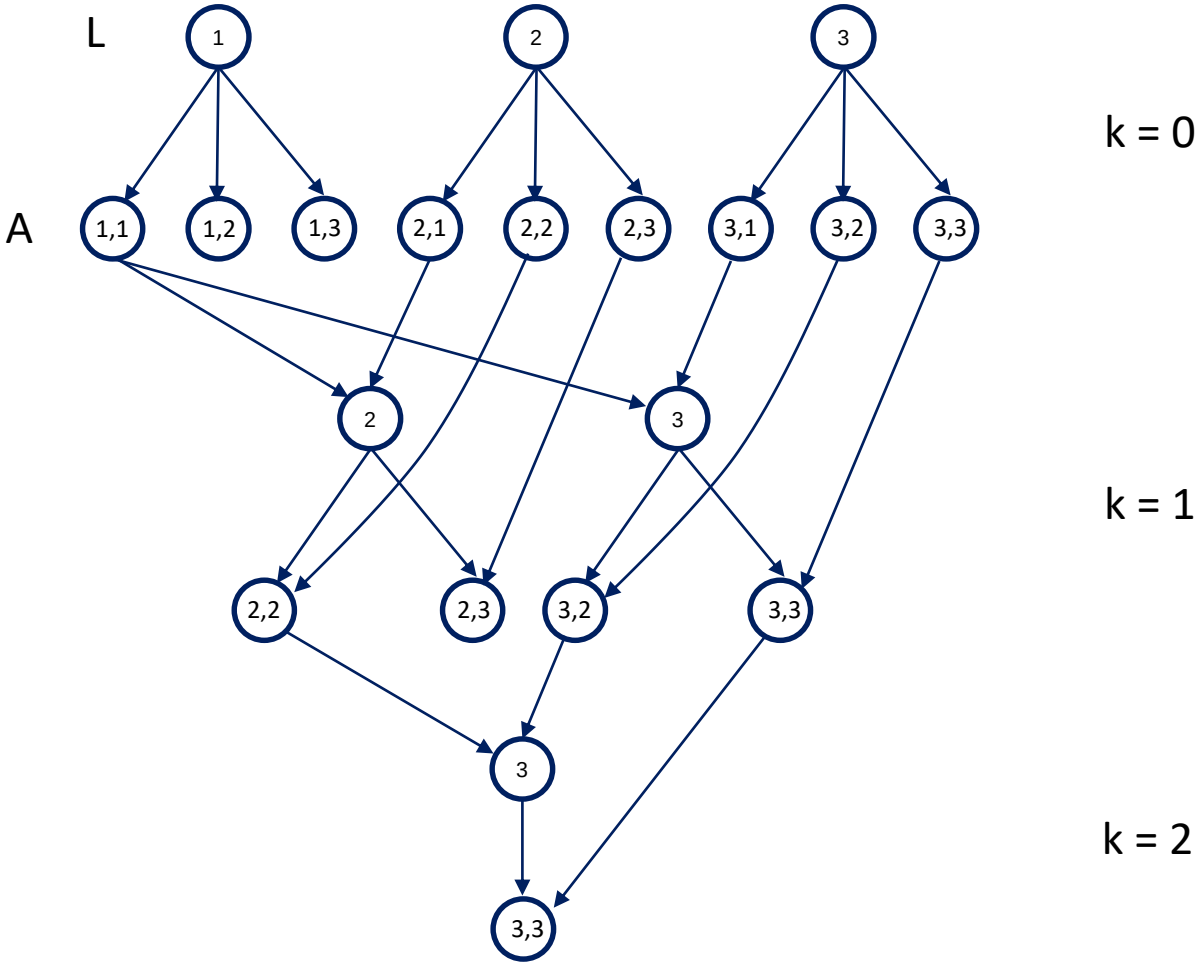


k = 0

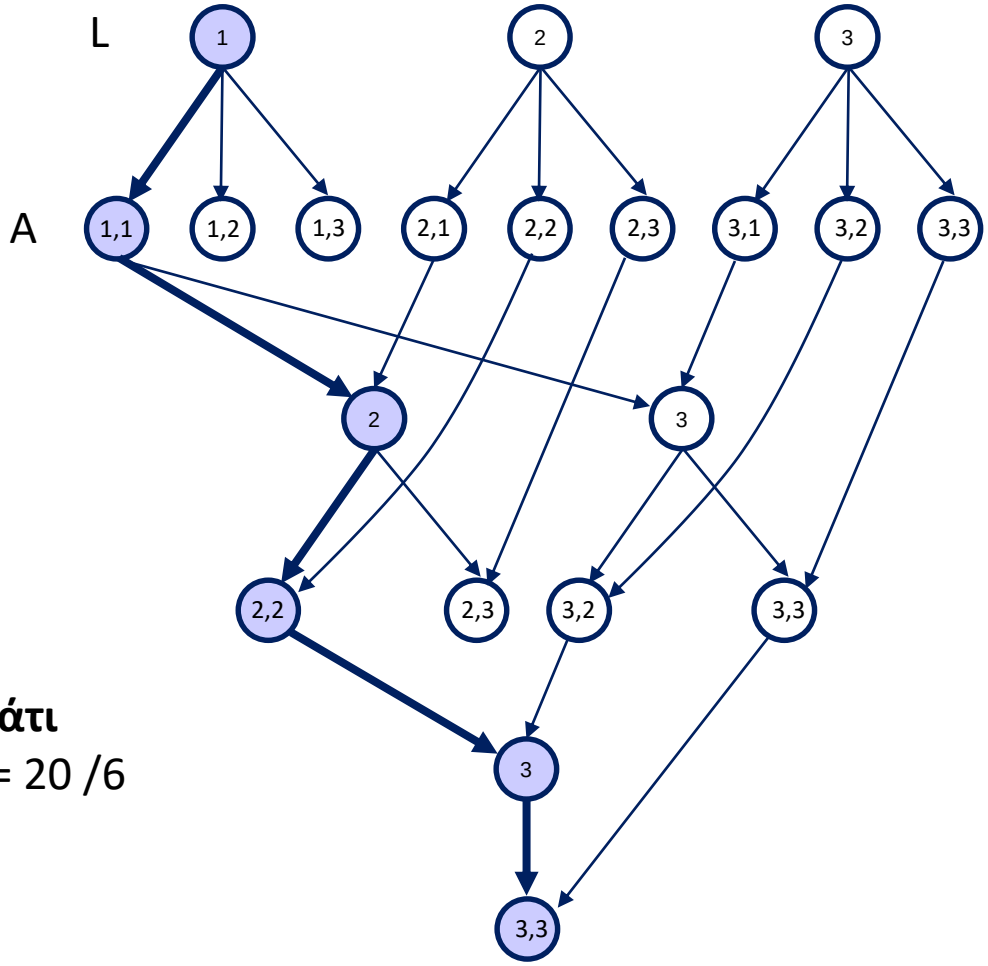
Παράδειγμα: Task graph για LU decomposition



Παράδειγμα: Task graph για LU decomposition



Παράδειγμα: Task graph για LU decomposition



Κρίσιμο μονοπάτι

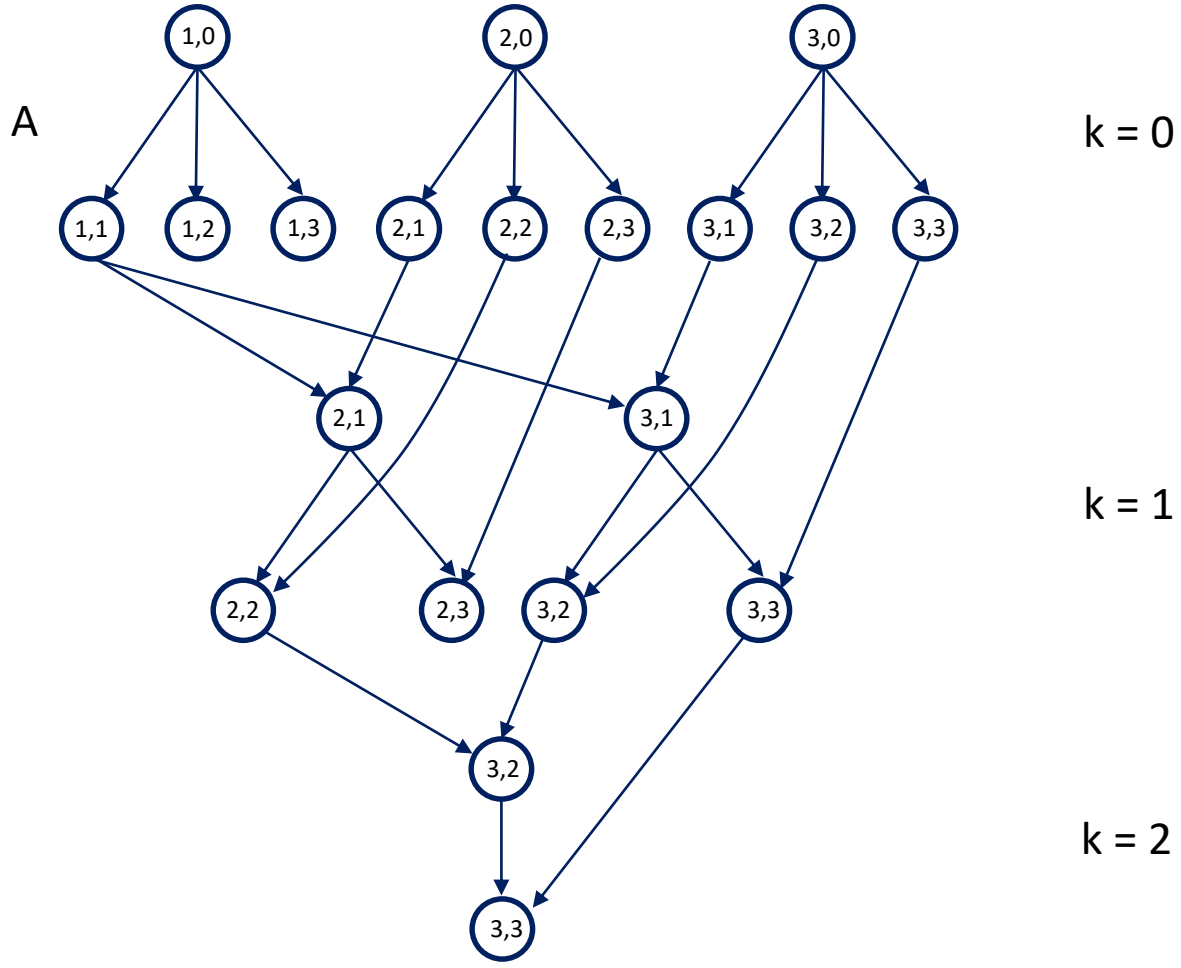
Μέγιστο speedup = 20 /6

Παράδειγμα: Task graph για LU decomposition

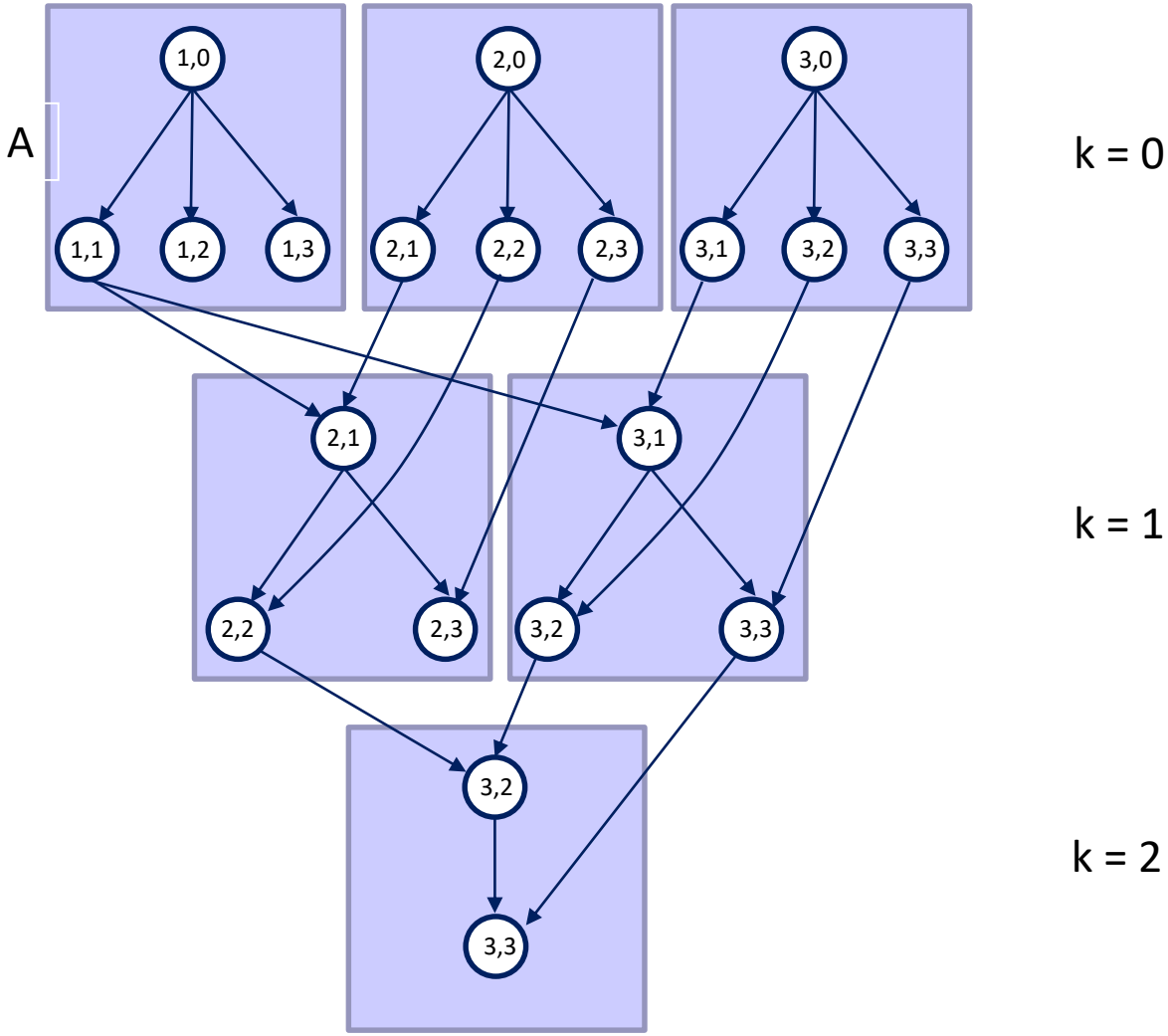
```
//LU decomposition kernel
for (k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      A[i][k] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j] = A[i][j] - A[i][k] * A[k][j];
   }
```

- Data centric προσέγγιση: μοιράζω κατά γραμμές τον πίνακα A

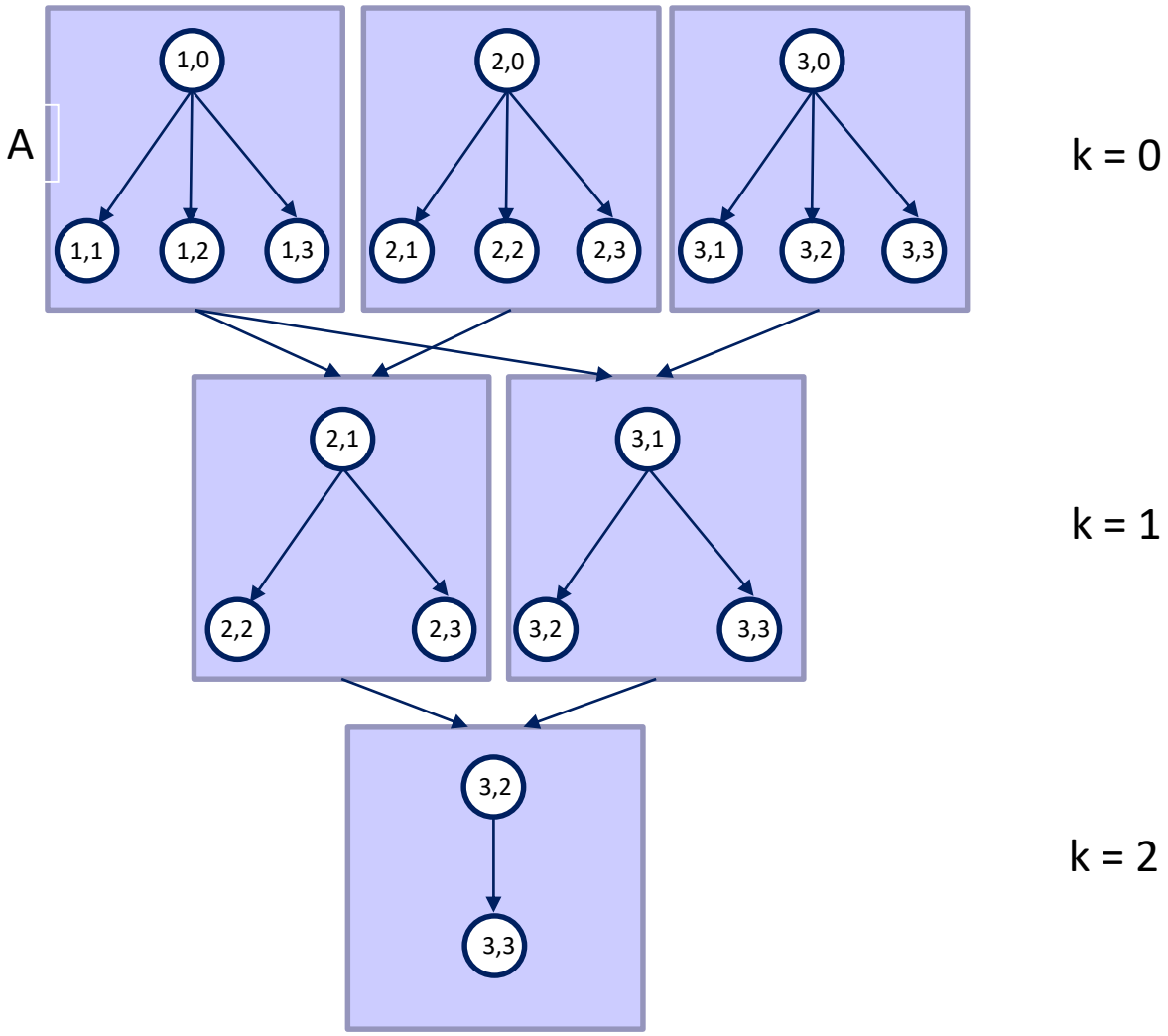
Παράδειγμα: Task graph για LU decomposition



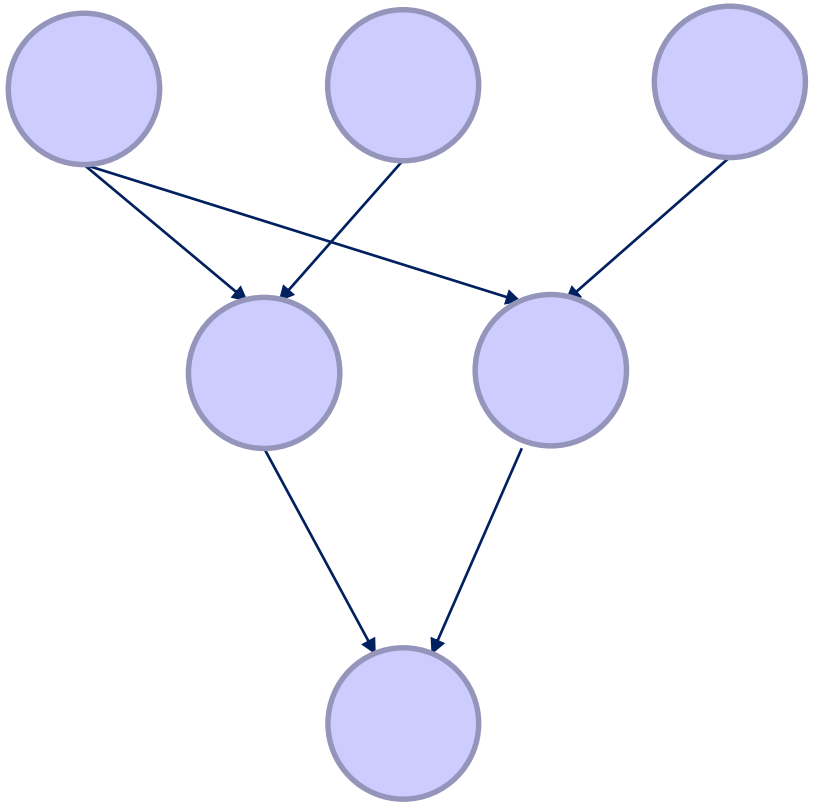
Παράδειγμα: Task graph για LU decomposition



Παράδειγμα: Task graph για LU decomposition



Παράδειγμα: Task graph για LU decomposition



k = 0

k = 1

k = 2

Στάδιο 3: Οργάνωση πρόσβασης στα δεδομένα

- **Χαρακτηρισμός δεδομένων**
- Τα δεδομένα του προβλήματος μπορούν να είναι:
  - *Μοιραζόμενα (shared data)*:
    - Μπορεί να υποστηριχθεί μόνο από προγραμματιστικά μοντέλα κοινού χώρου διευθύνσεων (βλ. παρακάτω: υλοποίηση παράλληλων προγραμμάτων)
    - Αποτελεί ένα πολύ βολικό τρόπο «κατανομής»
    - Χρειάζεται ιδιαίτερη προσοχή για τον εντοπισμό race conditions
  - *Κατανεμημένα* (*distributed data*):
    - Τα δεδομένα κατανέμονται ανάμεσα στα tasks
    - Αποτελεί τη βασική προσέγγιση στα προγραμματιστικά μοντέλα ανταλλαγής μηνυμάτων (βλ. παρακάτω: υλοποίηση παράλληλων προγραμμάτων)
    - Επιβάλλει επιπλέον προγραμματιστικό κόστος
  - *Αντιγραμμένα* (*replicated data*):
    - Για αντιγραφή (συνήθως μικρών) read-only δομών δεδομένων
    - Για αντιγραφή υπολογισμών
    - Βοηθάει στην αποφυγή επικοινωνίας

Στάδιο 3: Οργάνωση πρόσβασης στα δεδομένα

- Η κατανομή των υπολογισμών και των δεδομένων καθώς και ο χαρακτηρισμός τους δημιουργεί ανάγκες για **επικοινωνία** και **συγχρονισμό**
- **Επικοινωνία:**
  - Κατανεμημένα δεδομένα
  - 1 task χρειάζεται να διαβάσει δεδομένα που κατέχει ένα άλλο task
    - Είτε εξαιτίας του γράφου εξαρτήσεων (παράχθηκαν κατά την εκτέλεση σε άλλο task)
    - Είτε επειδή βρίσκονται αποθηκευμένα σε άλλο task (π.χ. από την αρχική κατανομή)
  - Σημείο-προς-σημείο και συλλογική
  - Καθορισμός ποια δεδομένα πρέπει να αποσταλούν, σε ποιες εργασίες και πότε

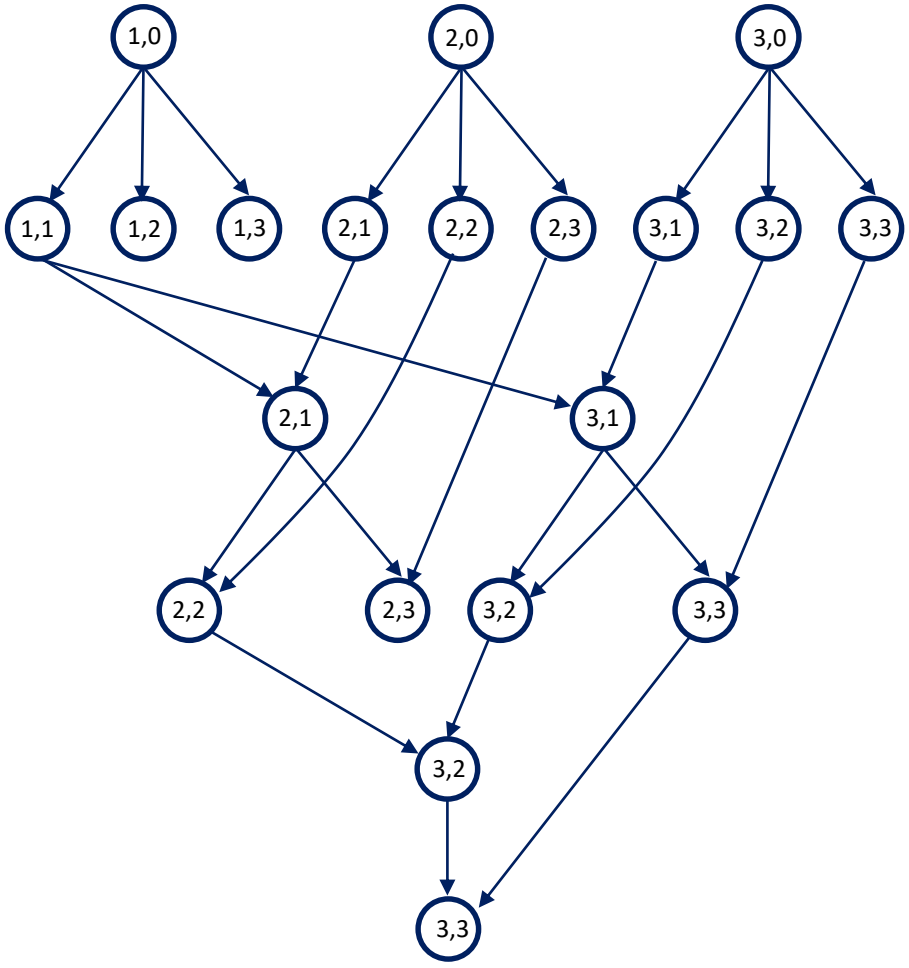
Στάδιο 3: Οργάνωση πρόσβασης στα δεδομένα

- Η κατανομή των υπολογισμών και των δεδομένων καθώς και ο χαρακτηρισμός τους δημιουργεί ανάγκες για **επικοινωνία** και **συγχρονισμό**
- **Συγχρονισμός:**
  - Απαιτείται είτε λόγω του γράφου εξαρτήσεων (σειριοποίηση) είτε λόγω καταστάσεων συναγωνισμού (race conditions)
  - **Σειριοποίηση**:
    - **Μηχανισμοί:** Barriers, condition variables, semaphores
    - Τους εισάγει ο προγραμματιστής απευθείας ή το αναλαμβάνει το σύστημα χρόνου εκτέλεσης (runtime system). Αφορά κυρίως το προηγούμενο στάδιο του ορισμού της ορθής σειράς εκτέλεσης
  - Εντοπισμός καταστάσεων συναγωνισμού και εισαγωγή κατάλληλου σχήματος ελέγχου ταυτόχρονης πρόσβασης (concurrency control)
  - **Αμοιβαίος αποκλεισμός:**
    - **Μηχανισμοί:** Critical section, Locks, readers-writers locks

Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης

- Μέχρι στιγμής έχουμε υποθέσει άπειρο αριθμό επεξεργαστών (όπως στο μοντέλο PRAM)
- Το στάδιο αυτό αναλαμβάνει να αναθέσει εργασίες (tasks) σε οντότητες εκτέλεσης (process, tasks, κλπ) με πεπερασμένο αριθμό.
- Ο τρόπος με τον οποίο ανατίθενται τα tasks σε οντότητες εκτέλεσης μπορεί να επηρεάσει σημαντικά την εκτέλεση:
  - Παραλληλισμός και ισοκατανομή φορτίου
  - Τοπικότητα δεδομένων
  - Κόστος συγχρονισμού και επικοινωνίας
  - Κόστος διαχείρισης

Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης

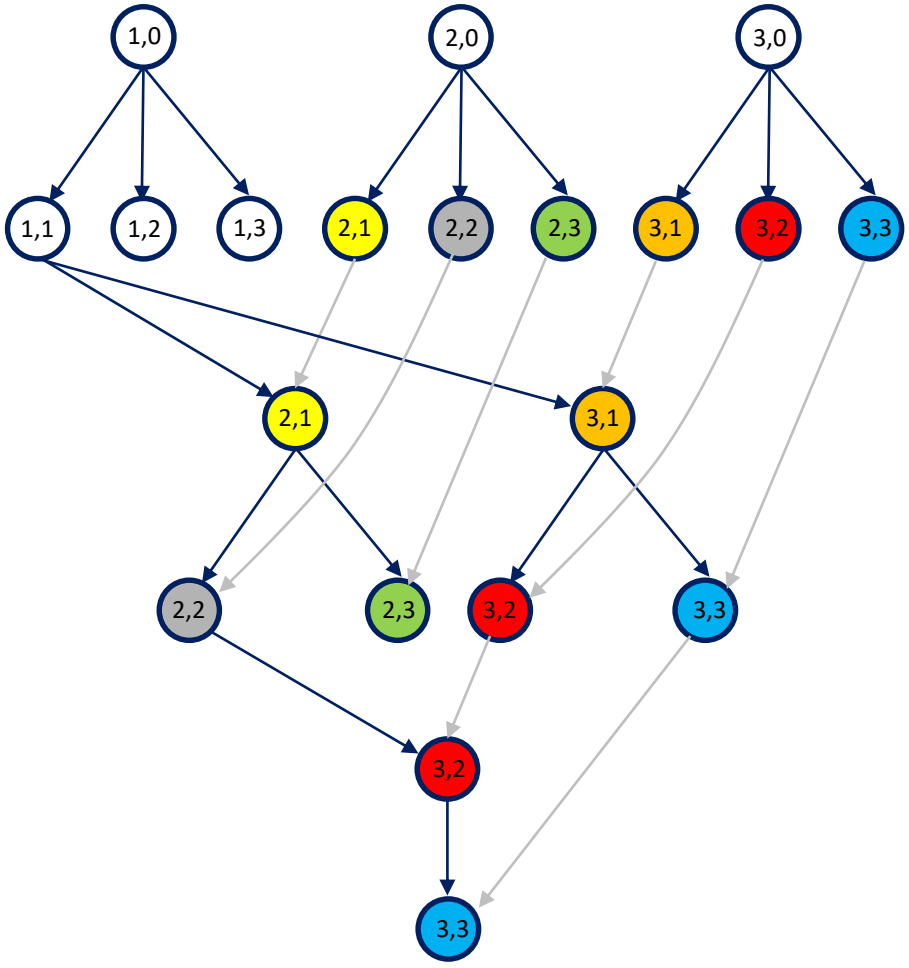


k = 0

k = 1

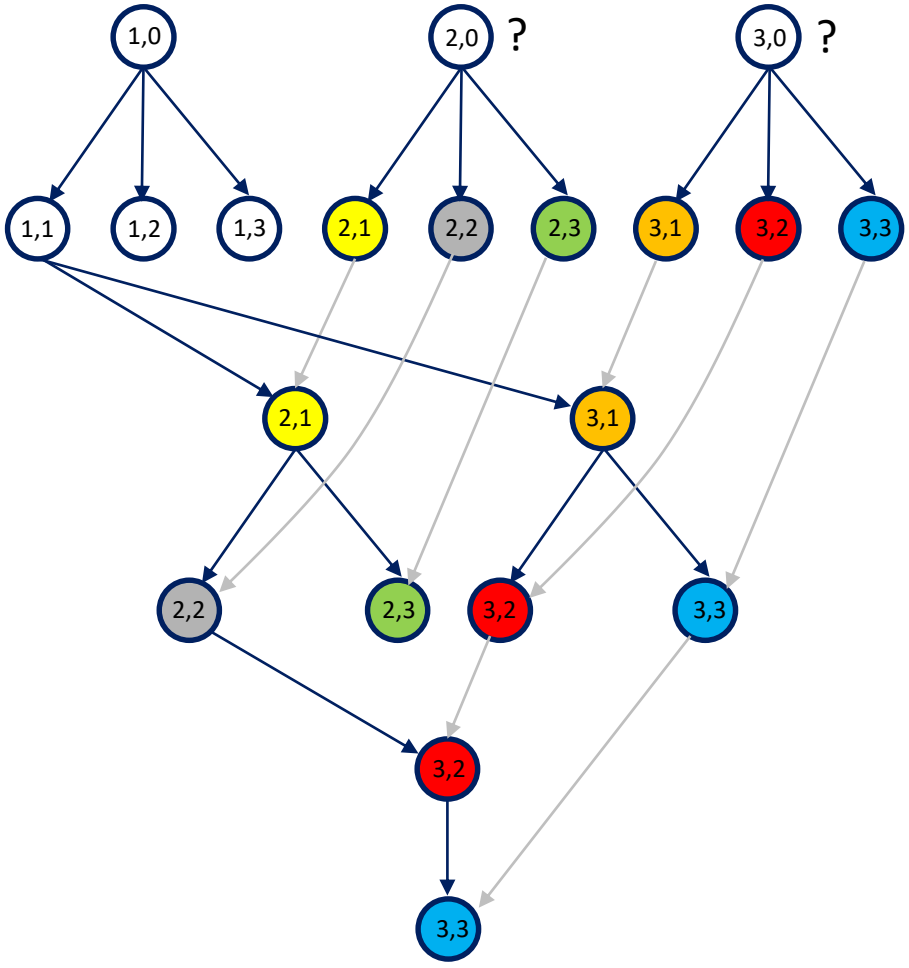
k = 2

Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης



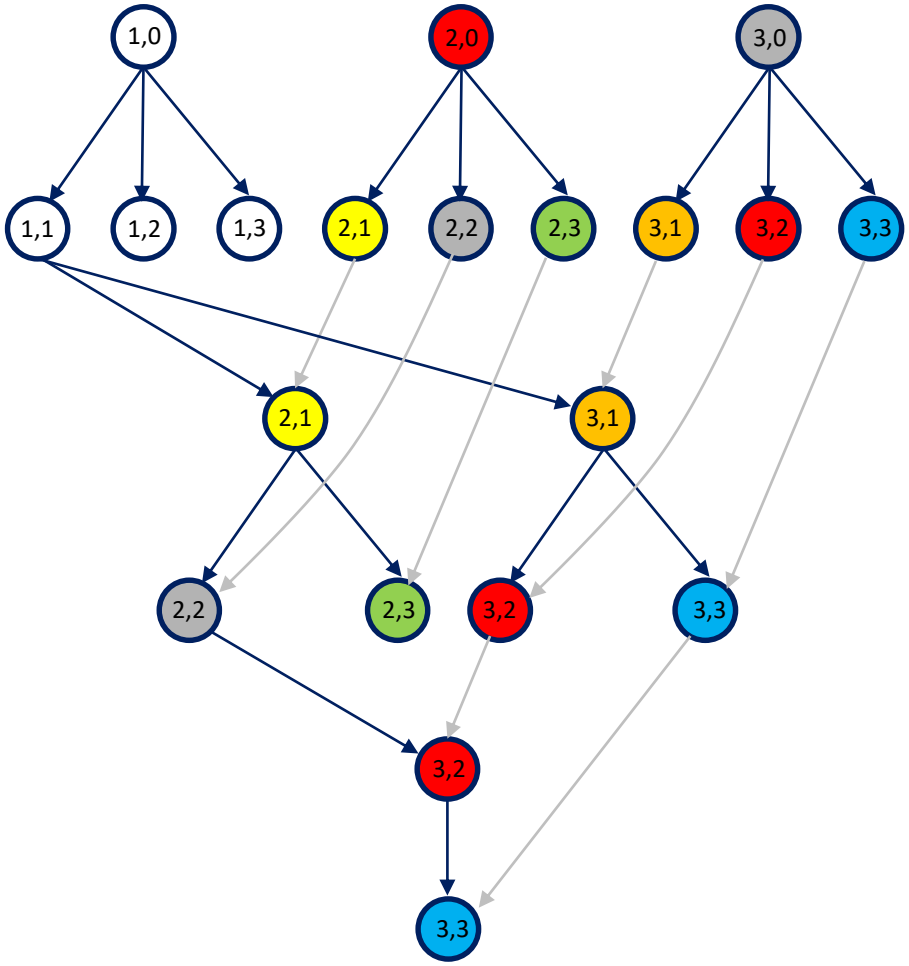
Τοπικότητα / μείωση επικοινωνίας

Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης



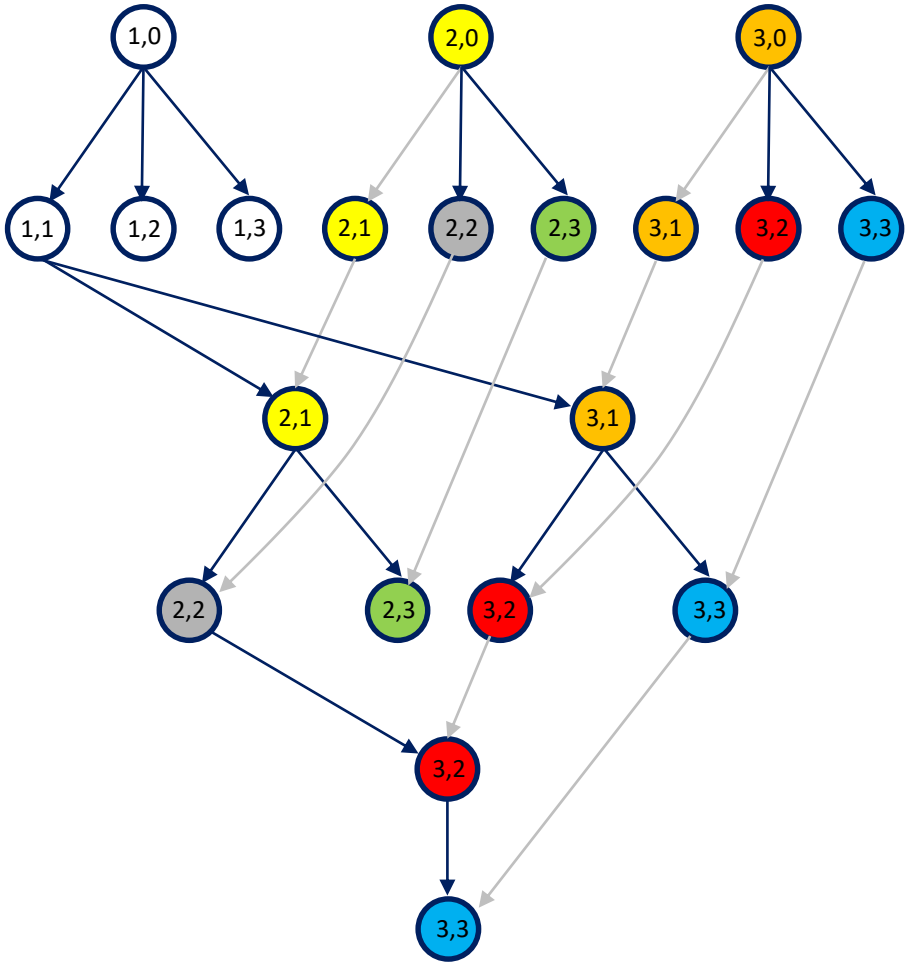
Τοπικότητα / μείωση επικοινωνίας

Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης



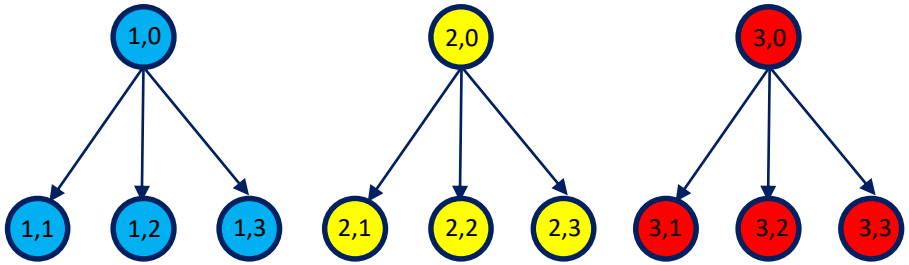
Τοπικότητα / μείωση επικοινωνίας

Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης



Τοπικότητα / μείωση επικοινωνίας

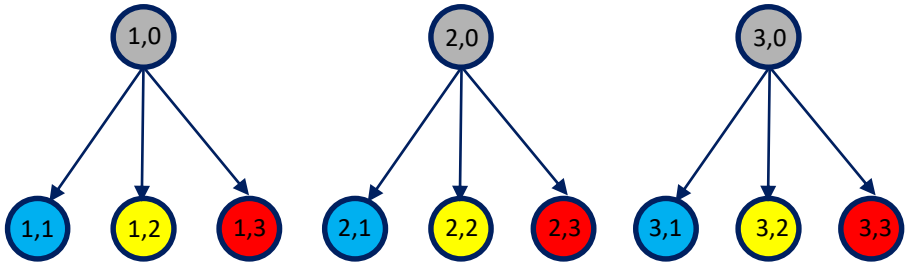
Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης



Πρόσβαση στη μνήμη:

Συνεχόμενες θέσεις μνήμης θα προσπελαστούν από τον ίδιο επεξεργαστή: χωρική τοπικότητα αναφορών για multicore αρχιτεκτονικές, βλ. cache lines, prefetching

Στάδιο 4: Ανάθεση εργασιών σε οντότητες εκτέλεσης

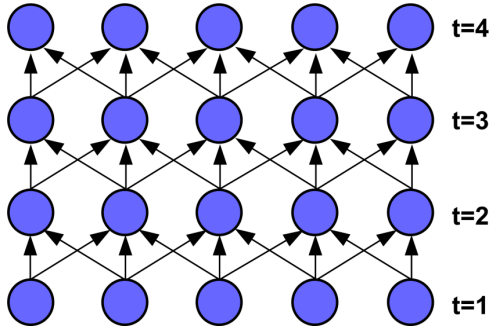


Πρόσβαση στη μνήμη:

Συνεχόμενες θέσεις μνήμης θα προσπελαστούν από διαφορετικό επεξεργαστή:
ευνοεί την ταυτόχρονη πρόσβαση στη μνήμη σε κάρτες γραφικών

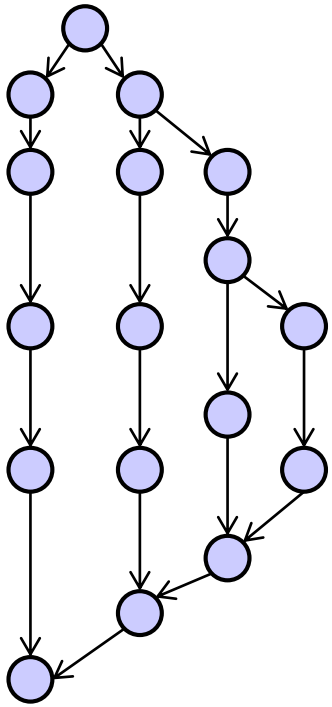
Στατική vs. Δυναμική ανάθεση εργασιών

- **Στατική απεικόνιση:**
  - Διαμοιρασμός των tasks σε threads πριν την έναρξη της εκτέλεσης
  - Καλή πρακτική: Αν η εφαρμογή το επιτρέπει, συνηθίζουμε να έχουμε 1 task / thread (προσαρμόζοντας κατάλληλα το σχεδιασμό μας και στο Στάδιο 1)
  - Κατάλληλη στρατηγική για εφαρμογές με:
    - ομοιόμορφη και γνωστή εξαρχής κατανομή φορτίου (π.χ. regular task graph)
    - ανομοιόμορφη αλλά προβλέψιμη κατανομή φορτίου
  - Πλεονεκτήματα :
    - Απλή στην υλοποίηση
    - Καλή επίδοση για «κανονικές εφαρμογές»
    - Μηδενικό overhead
  - Μειονεκτήματα:
    - Κακή επίδοση για δυναμικές και ακανόνιστες εφαρμογές λόγω ανισοκατανομής φορτίου

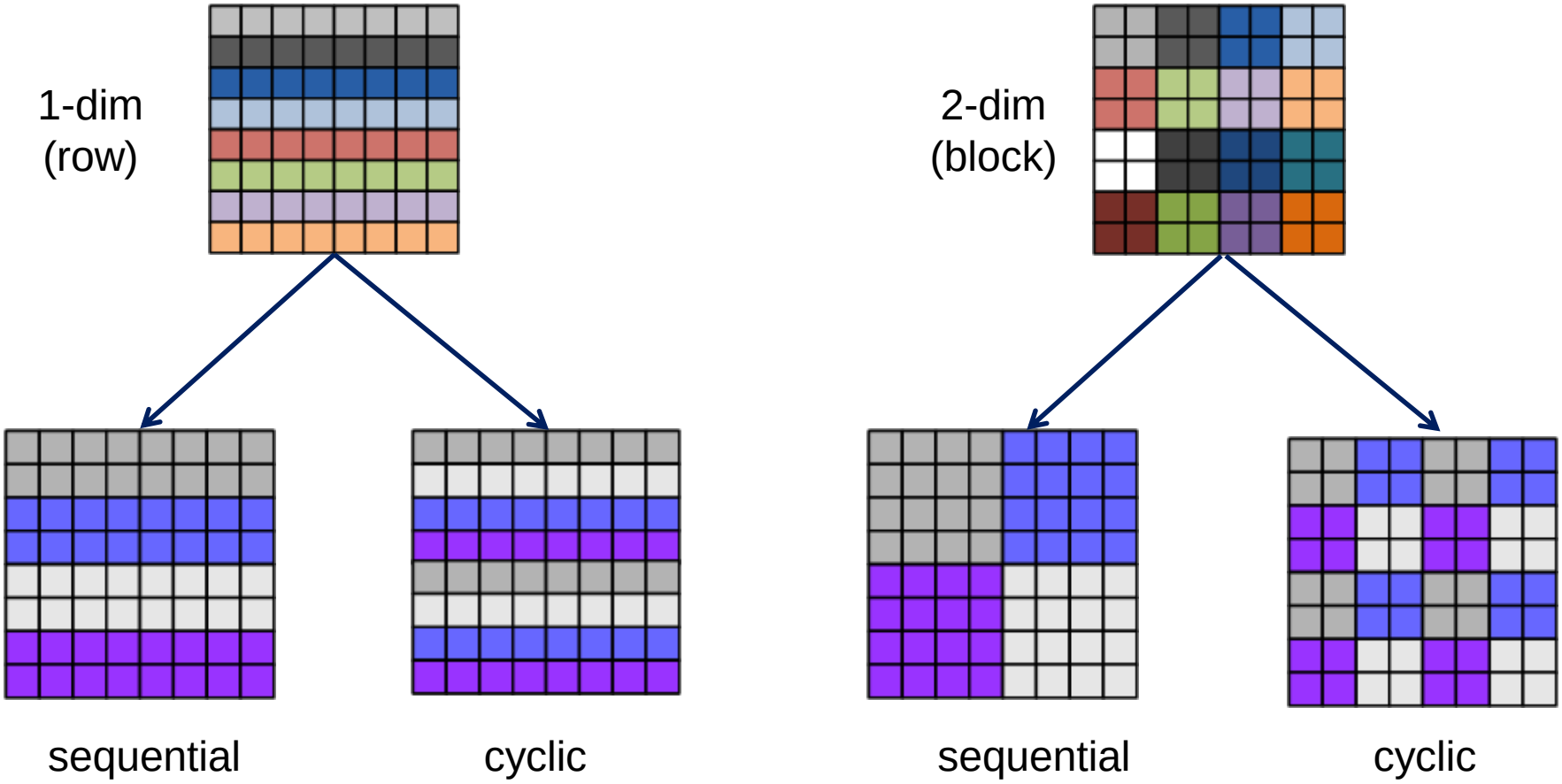


Στατική vs. Δυναμική απεικόνιση tasks

- **Δυναμική απεικόνιση:**
  - Διαμοιρασμός των tasks σε threads κατά την εκτέλεση
  - Κατάλληλη στρατηγική για εφαρμογές με:
    - ακανόνιστο γράφο εργασιών
    - μη προβλέψιμο φορτίο
    - δυναμικά δημιουργούμενα tasks
  - Πλεονεκτήματα:
    - Εξισορρόπηση φορτίου σε δυναμικές και ακανόνιστες εφαρμογές
  - Μειονεκτήματα:
    - Δύσκολη στην υλοποίηση (συνήθως το αναλαμβάνει το run-time σύστημα)
    - Μπορεί να δημιουργήσει bottleneck σε περίπτωση υλοποίησης με ένα κεντρικό scheduling thread (απαιτούνται κατανεμημένοι αλγόριθμοι)



data centric, στατική απεικόνιση



Παράδειγμα: Επιλογή απεικόνισης για LU και stencil

```
//LU decomposition kernel
for(k = 0; k < N-1; k++)
   for(i = k+1; i < N; i++){
      L[i] = A[i][k] / A[k][k];
      for(j = k+1; j < N; j++)
         A[i][j]=A[i][j]-L[i]*A[k][j];
   }
```

```
//stencil
for (t = 1; t < T; t++){
   for (i = 1; i < X; i++)
      for (j = 1; j < Y; j++)
         A[t][i][j] = 0.2*(A[t-1][i][j] + A[t-1][i-1][j] +
                           A[t-1][i+1][j] + A[t-1][i][j-1] +
                           A[t-1][i][j]);
}
```

Σχετικά ζητήματα απεικόνισης (δρομολόγησης)

- Δρομολόγηση εργασιών σε ένα υπερυπολογιστικό σύστημα
  - *N* κόμβοι, c πυρήνες ανά κόμβο
  - *K* jobs, κάθε job *i* ζητάει N_i κόμβους και c_i πύρήνες
  - Στατικές προσεγγίσεις
- Δρομολόγηση παράλληλων εφαρμογών σε επίπεδο λειτουργικού
  - Πολυπύρηνο σύστημα
  - *K* εφαρμογές με διαφορετικός αριθμό από threads η κάθε μία
  - Οι τρέχουσες προσεγγίσεις δεν είναι ικανοποιητικές
- Δρομολόγηση των tasks μίας παράλληλης εφαρμογής σε ένα πολυπύρηνο σύστημα
  - Επαναλήψεις ενός παράλληλου loop
  - Tasks μίας παράλληλης εφαρμογής

Σύνοψη σχεδιασμού παράλληλων προγραμμάτων

- Δεν υπάρχει ξεκάθαρη μεθοδολογία παραλληλοποίησης προγραμμάτων
- Δύο σημαντικά βήματα (συχνά όχι σειριακά):
  - **Σχεδιασμός (σκέψη)**
    - **Στάδιο 1:** Κατανομή υπολογισμών και διαμοιρασμός δεδομένων
    - **Στάδιο 2:** Ορισμός ορθής σειράς εκτέλεσης
    - **Στάδιο 3:** Οργάνωση πρόσβασης στα δεδομένα
    - **Στάδιο 4:** Ανάθεση εργασιών σε οντότητες εκτέλεσης
- **Στη συνέχεια:**
  - **Υλοποίηση (ομιλία)**
    - Έκφραση παραλληλίας
    - Επιλογή προγραμματιστικού μοντέλου και εργαλείων

Ερωτήσεις;