Προηγμένες αρχιτεκτονικές υπολογιστών και προγραμματισμός παραλλήλων συστημάτων

10. Παραλληλοποίηση κώδικα

Παράλληλες εφαρμογές

- Επιστημονικές εφαρμογές, μεγάλες βάσεις δεδομένων, visualization.
- Έμφαση στην επεξεργαστική ισχύ.
- Προβλήματα grand-challenge.

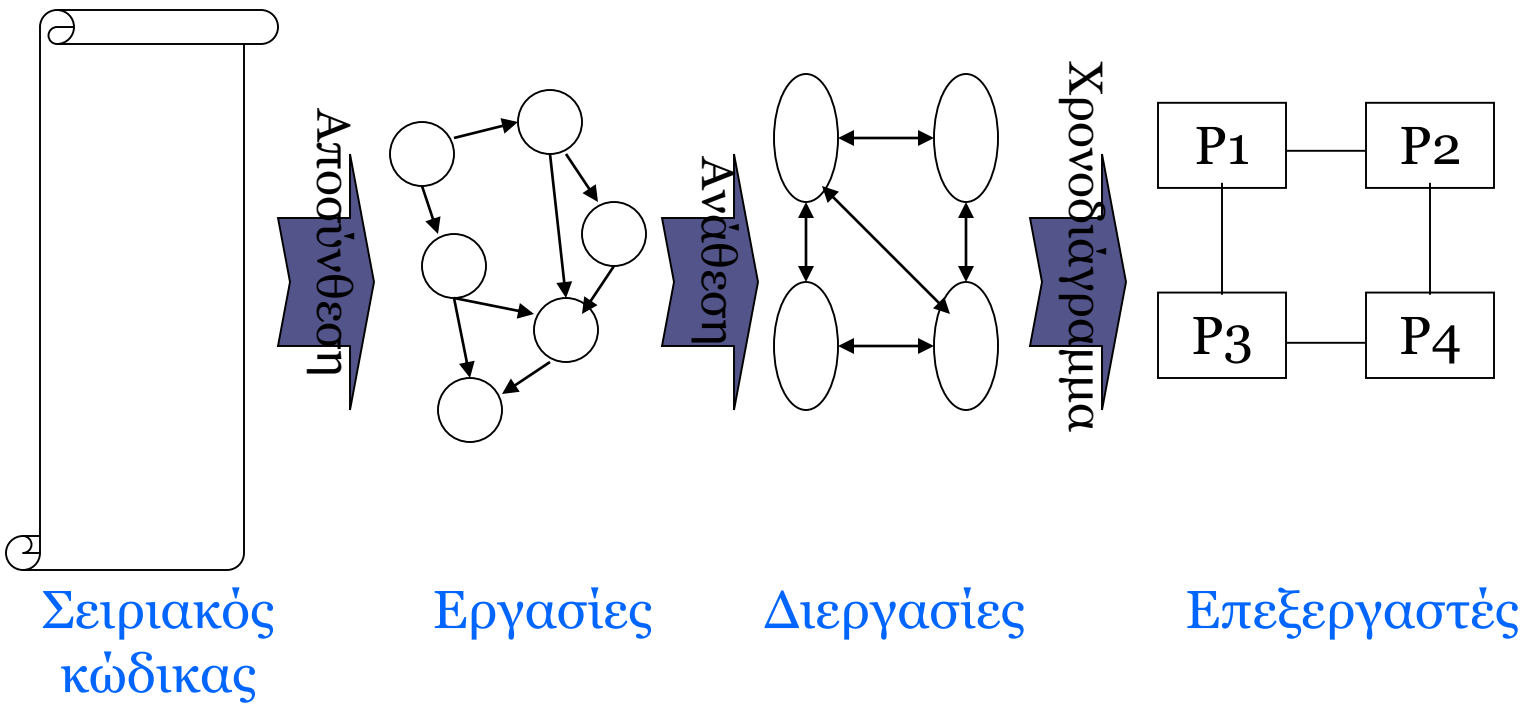
Προβλήματα grand-challenge

- Μεγάλες προκλήσεις της πληροφορικής.
- 3-διάστατη οπτικοποίηση σε πραγματικό χρόνο.
  - Εφαρμογές: 3D rendering, εικονική πραγματικότητα, 3D graphics, τηλεόραση, κινηματογράφος.
- Αναζήτηση σε μεγάλες βάσεις δεδομένων.
  - Google: 6000 CPUs, 12000 δίσκοι, 1,5 δισεκ. σελίδες.
- Σχεδιασμός μεγάλων οργανικών μορίων.
  - Εφαρμογές: σχεδιασμός φαρμάκων, βιοχημεία
- Εξομοίωση μεγάλων συστημάτων.
  - Εφαρμογές: κίνηση ουράνιων σωμάτων, μελέτη ωκεάνιων ρευμάτων, πρόγνωση καιρού, αεροδυναμική.

Πρόβλημα προγραμματισμού

- Είμαστε συνηθισμένοι/εκπαιδευμένοι σε συγγραφή σειριακού κώδικα.
- Μεγάλος όγκος σειριακού κώδικα προϋπάρχει.
- Στόχος: η μετατροπή σειριακού κώδικα σε παράλληλο.

Μετατροπή σειριακού κώδικα σε παράλληλο



Παραλληλοποίηση κώδικα

- **Αποσύνθεση**: Ο κώδικας σε σειριακή μορφή μετατρέπεται σε γράφο εξάρτησης (dependence graph).
- **Ανάθεση σε διεργασίες**: οι κόμβοι του γράφου εξάρτησης ομαδοποιούνται σε διεργασίες και αποφασίζουμε σε ποιον επεξεργαστή θα εκτελεστεί κάθε διεργασία.
- **Χρονοδρομολόγηση**: αποφασίζουμε πότε θα εκτελεστεί κάθε διεργασία. Συχνά συνδυάζεται με την ανάθεση και γίνεται μαζί.

Παράλληλοι μεταφραστές

- Παράλληλη Fortran, παράλληλη C, C++.
- Η πλήρης αυτοματοποίηση είναι δύσκολη ιδίως στο επίπεδο αποσύνθεσης σειριακού κώδικα σε εργασίες.
- Η βέλτιστη ομαδοποίηση σε διεργασίες είναι επίσης δύσκολη και με πολλές ελεύθερες ή άγνωστες παραμέτρους.

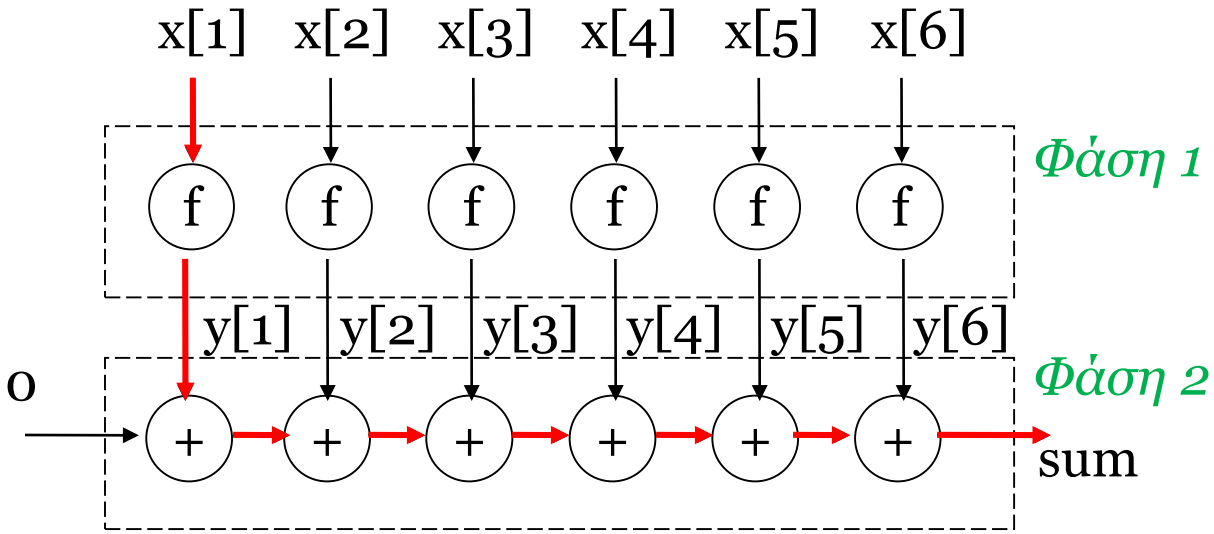
Αποσύνθεση κώδικα

```
for (i=1 to 6) { /*  Φάση  1 */
 y[i] = f(x[i]);
}
sum = 0;
for (i=1 to 6) { /*Φάση 2*/
      sum = sum + y[i];
}
```

```
y[1] = f(x[1]);
y[2] = f(x[2]);
...
y[6] = f(x[6]);
sum = 0;
sum = sum + y[1];
sum = sum + y[2];
...
sum = sum + y[6];
```

Παράδειγμα αποσύνθεσης κώδικα (1)

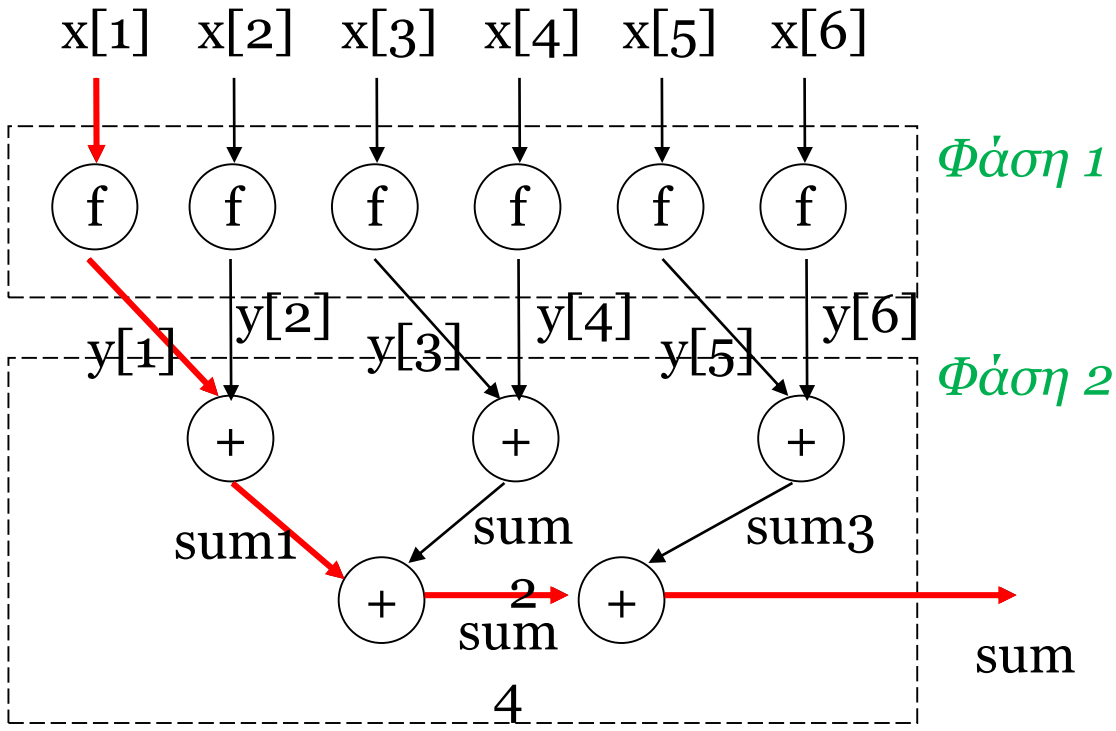
Αυτόματη αποσύνθεση



Παράδειγμα αποσύνθεσης κώδικα (2)

Βέλτιστη αποσύνθεση

```
sum1=y[1]+y[2];
sum2=y[3]+y[4];
sum3=y[5]+y[6];
sum4=sum1+sum2;
sum=sum3+sum4;
```



Συμπεράσματα Αποσύνθεσης

- Η απλή μηχανική αποσύνθεση σε Γράφο Εξάρτησης (ΓΕ) είναι σχετικά απλή.
- Η βελτιστοποίηση του κώδικα ίσως απαιτεί την τροποποίηση του σειριακού κώδικα.
- Η βελτιστοποίηση είναι δύσκολη.

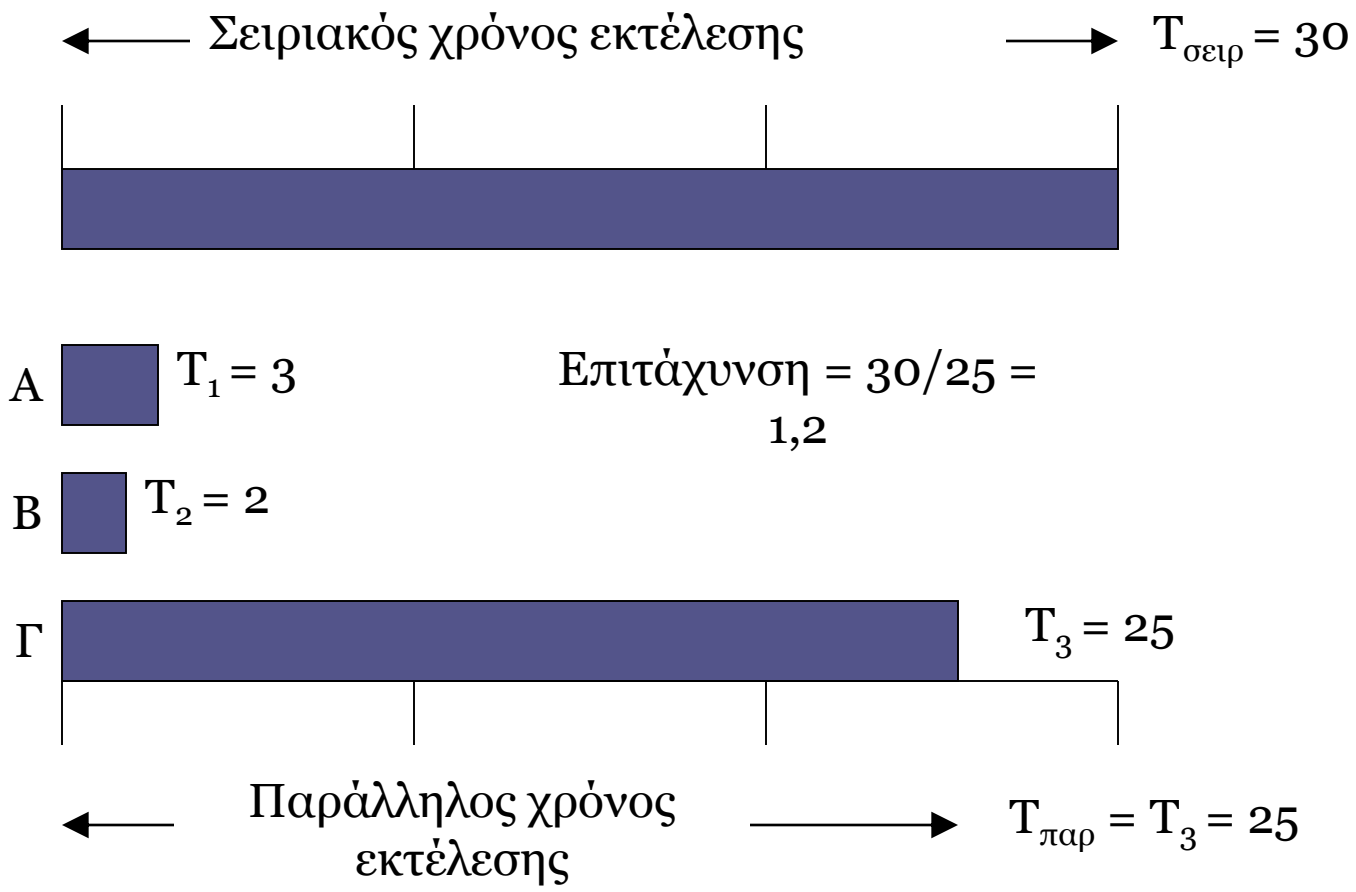
Ανάθεση εργασιών σε διεργασίες

- Ανάθεση = ομαδοποίηση εργασιών σε ομάδες (διεργασίες). Κάθε διεργασία εκτελείται σε ένα επεξεργαστή.
- Στόχοι ανάθεσης:
  - Ίσο μοίρασμα φόρτου εργασίας.
  - Ελαχιστοποίηση επικοινωνίας.
  - Μείωση του χρόνου ανάθεσης.

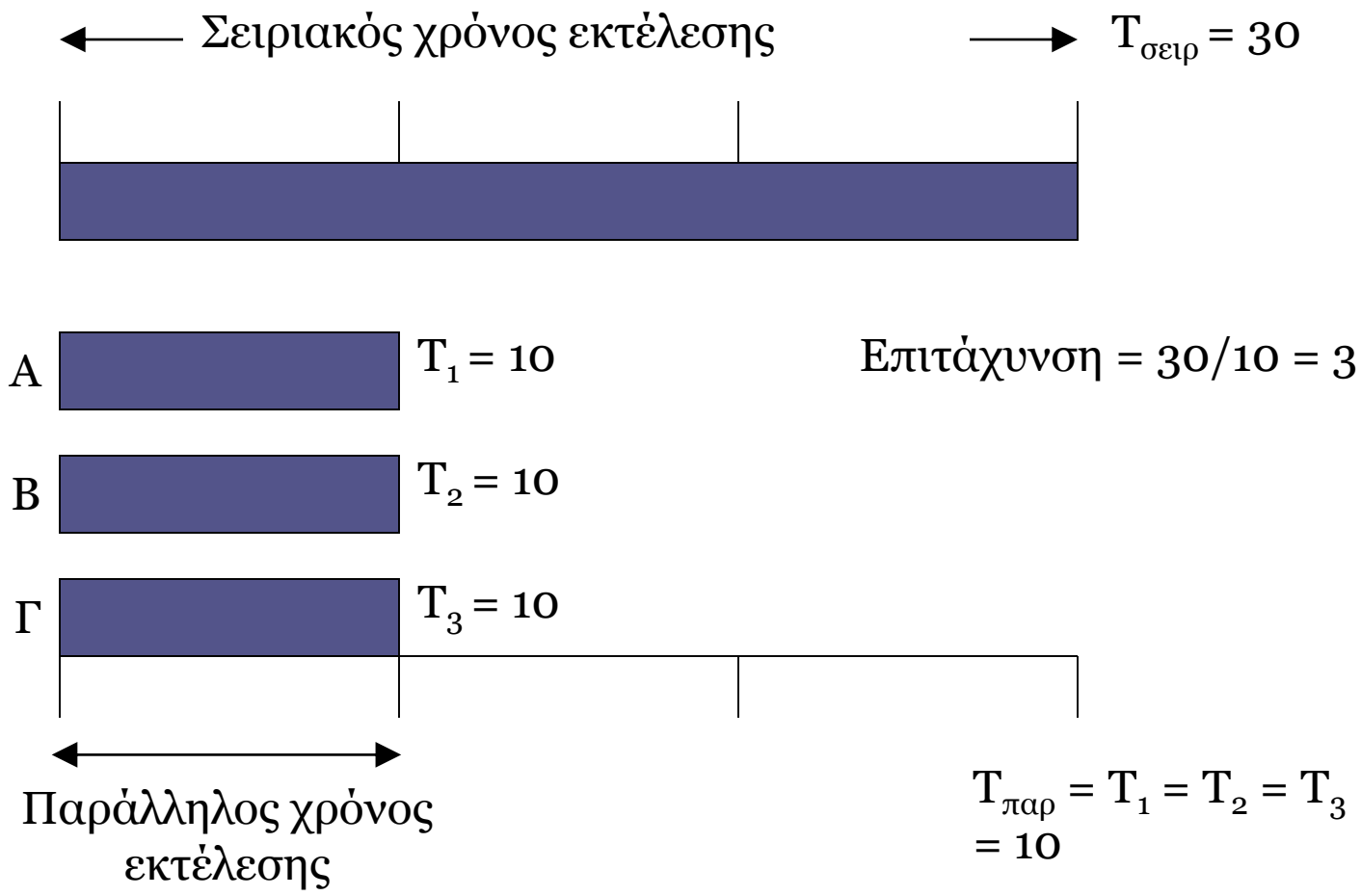
Εξισορρόπηση φόρτου εργασίας

- Εξισορρόπηση χρήσης πόρων. Δηλαδή:
- Εξισορρόπηση χρήσης CPU.
- Εξισορρόπηση χρήσης μνήμης.
- Εξισορρόπηση χρήσης I/O.
- Εξισορρόπηση επικοινωνίας.

Κακή ανάθεση φόρτου εργασίας



Βέλτιστη εξισορρόπηση φόρτου εργασίας



Προβλήματα εξισορρόπησης φόρτου εργασίας

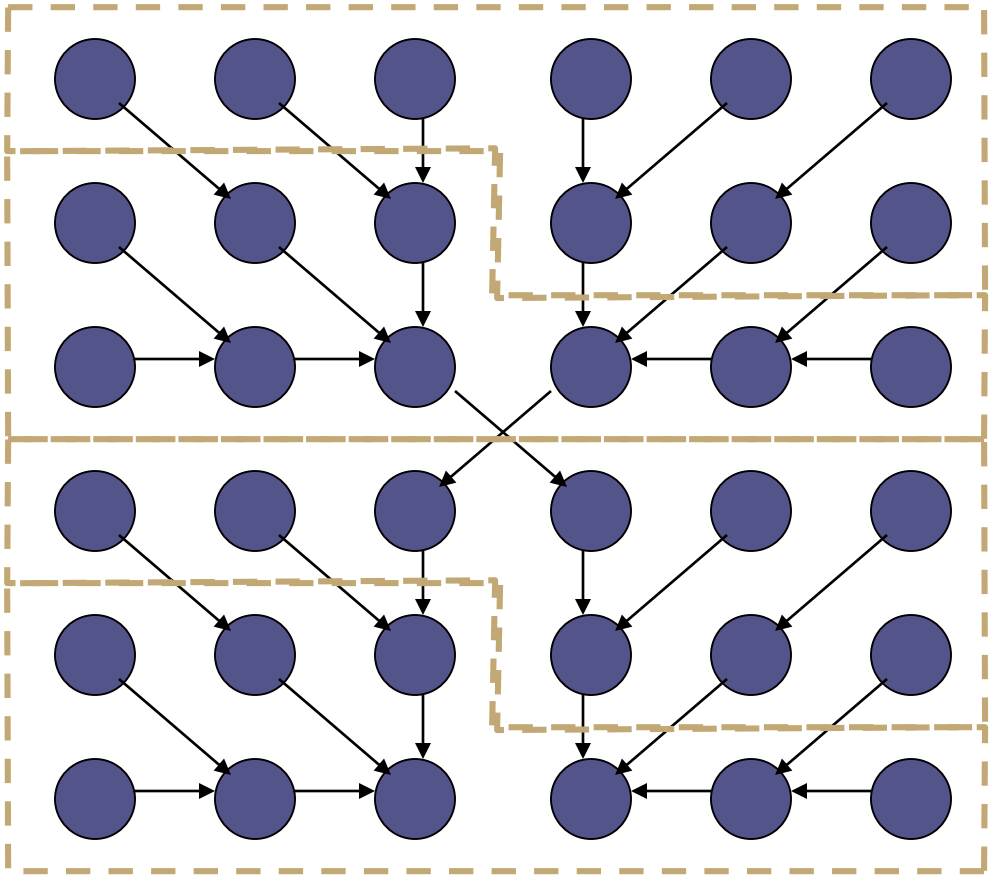
- Πώς εκτιμάμε το φόρτο εργασίας?
  - Εκ των προτέρων εκτίμηση (compile-time).
    - Πλεονέκτημα: στατικός αλγόριθμος ανάθεσης.
    - Μειονέκτημα: αγνοεί τις πραγματικές συνθήκες εκτέλεσης, πχ. cache-miss, network-traffic, κλπ.
  - Εν θερμώ εκτίμηση (run-time).
    - Πλεονέκτημα: δυνατότητα βέλτιστης διαχείρισης.
    - Μειονέκτημα: δυναμική αναπροσαρμογή της ανάθεσης.

Ελαχιστοποίηση επικοινωνίας (1)

- Προσπάθεια διατήρησης στην ίδια ομάδα εκείνων των εργασιών που ανταλλάσσουν δεδομένα.
- Ίδια ομάδα = ίδια διεργασία.
- Όλες οι εργασίες μιας διεργασίας εκτελούνται στον ίδιο επεξεργαστή = ελαχιστοποίηση επικοινωνίας.

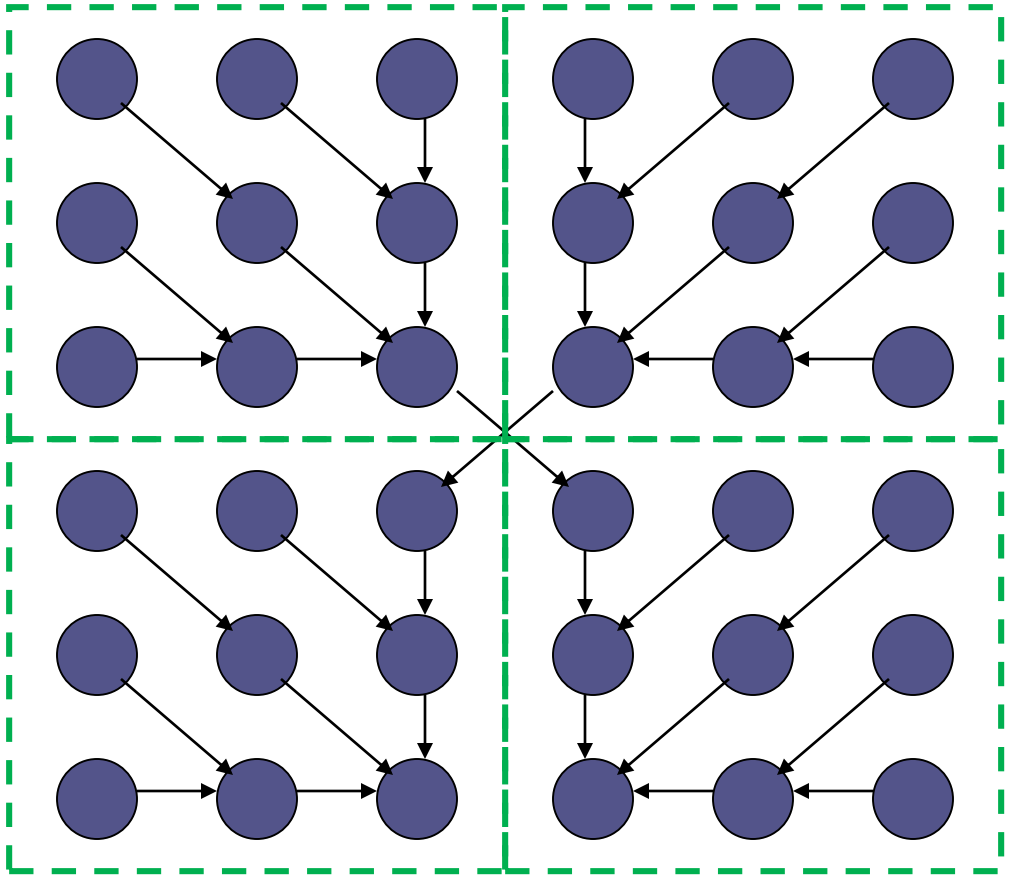
Ελαχιστοποίηση επικοινωνίας (2)

Κακή ανάθεση



Ελαχιστοποίηση επικοινωνίας (3)

Καλή ανάθεση



Μείωση χρόνου ανάθεσης

- Επιλογές διαδικασίας ανάθεσης:
- **Στατική ανάθεση**: Γίνεται την ώρα της μετάφρασης (compile-time).
- **Δυναμική ανάθεση**: Ομαδοποίηση εργασιών σε διεργασίες κατά τη διάρκεια εκτέλεσης του προγράμματος (run-time).
- **Ημι-στατική ανάθεση**: Αρχικά στατική, τροποποίηση κατά το run-time.

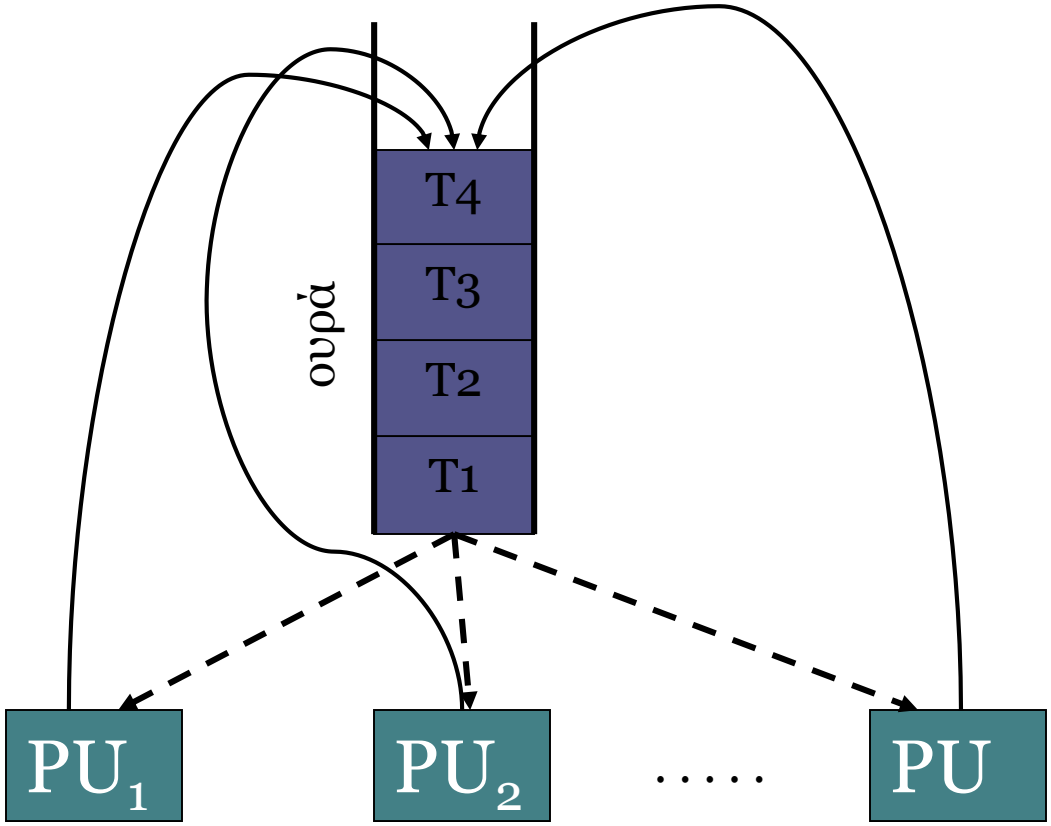
Στατική ανάθεση

- Γίνεται την ώρα της μετάφρασης (compile-time).
- Γρήγοροι ευρηστικοί αλγόριθμοι.
- Δεν απασχολείται κάποιος διαχειριστής-επεξεργαστής την ώρα της εκτέλεσης.
- Πρόβλημα: αγνοεί τις πραγματικές συνθήκες εκτέλεσης του παράλληλου προγράμματος. Με αυτή την έννοια, όχι βέλτιστη ανάθεση.

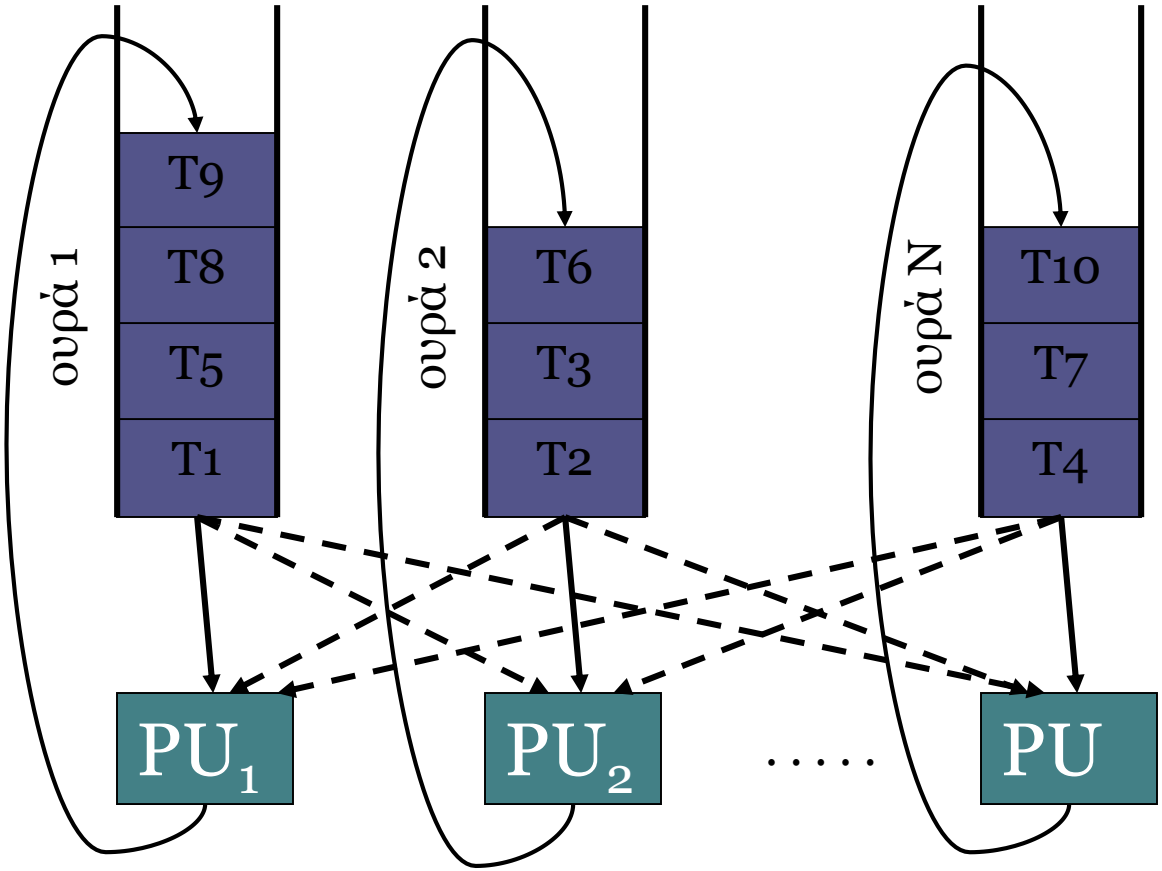
Δυναμική ανάθεση

- Οι εργασίες ομαδοποιούνται σε διεργασίες καθώς το πρόγραμμα εκτελείται. Δεν έχει προαποφασιστεί σε ποιον επεξεργαστή θα εκτελεστεί κάθε διεργασία.
- Ουρές εργασιών:
  - Μια κοινή (κεντρική) ουρά.
  - Πολλές κατανεμημένες ουρές.

Δυναμική ανάθεση: Μια κεντρική ουρά



Δυναμική ανάθεση: πολλές κατανεμημένες ουρές



Ημιστατική ανάθεση

- Αρχικά, στατική ανάθεση.
- Κατόπιν, κατά τακτά χρονικά διαστήματα η ανάθεση τροποποιείται λαμβάνοντας υπ' όψη τις συνθήκες εκτέλεσης σε κάθε επεξεργαστή μέχρι τώρα.

Χρονοδρομολόγηση (scheduling, 1)

- Χρονοδιάγραμμα (schedule) : Πότε θα εκτελεστεί η εργασία Τ? Σε ποιον επεξεργαστή?
- Χρονοδρομολογητής (scheduler): πρόγραμμα διαχειριστής του χρονοδιαγράμματος.
- Συχνά η χρονοδρομολόγηση γίνεται σε συνδυασμό με την ανάθεση.

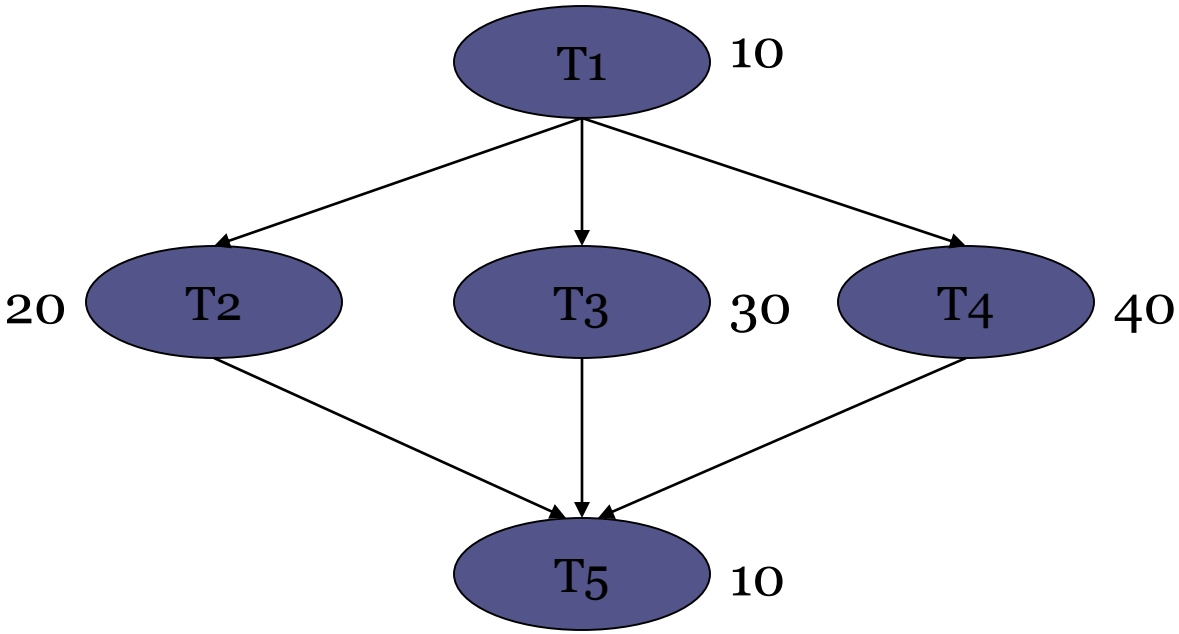
Χρονοδρομολόγηση (scheduling, 2)

- Κατηγοριοποίηση μεθόδων scheduling:
- Ως προς τον αλγόριθμο:
  - Ντετερμινιστικό: υποθέτουμε ότι ο χρόνος εκτέλεσης των εργασιών είναι σταθερός και δεδομένος.
  - Μη ντετερμινιστικό: υποθέτουμε ότι ο χρόνος εκτέλεσης των εργασιών είναι τυχαίος σύμφωνα με κάποια κατανομή.
- Ως προς την ανάθεση εργασιών σε επεξεργαστές:
  - στατικό: απόφαση σε compile-time.
  - δυναμικό: απόφαση σε run-time.

Ντετερμινιστικό scheduling

- Ντετερμινιστική χρονοδρομολόγηση = η πιο απλή περίπτωση. Μακριά από την πραγματικότητα.
- Δύο είδη ανάλογα με το μοντέλο επικοινωνίας.
  - μηδενικός χρόνος επικοινωνίας.
  - μη-μηδενικός αλλά σταθερός χρόνος επικοινωνίας.

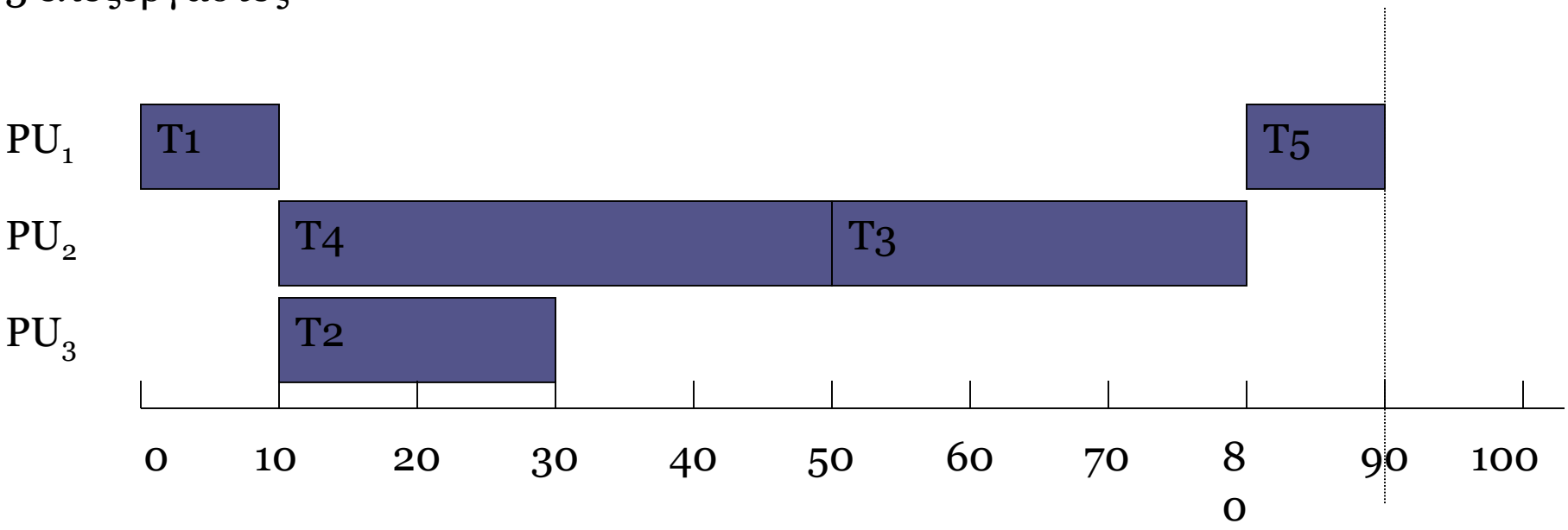
Ντετερμινιστικό scheduling με μηδενικό κόστος επικοινωνίας (1)



Ντετερμινιστικό scheduling με μηδενικό κόστος επικοινωνίας (2)

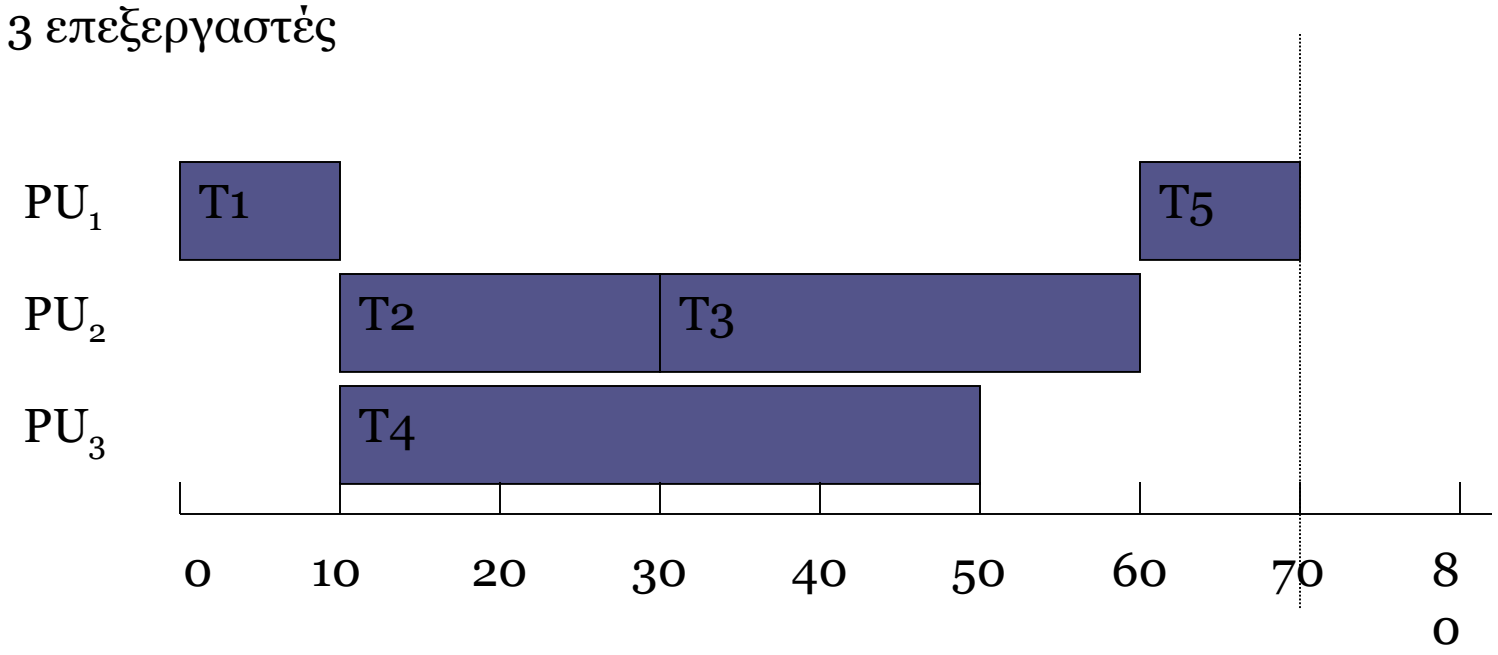
Χρονοδιάγραμμα 1, t = 90

3 επεξεργαστές

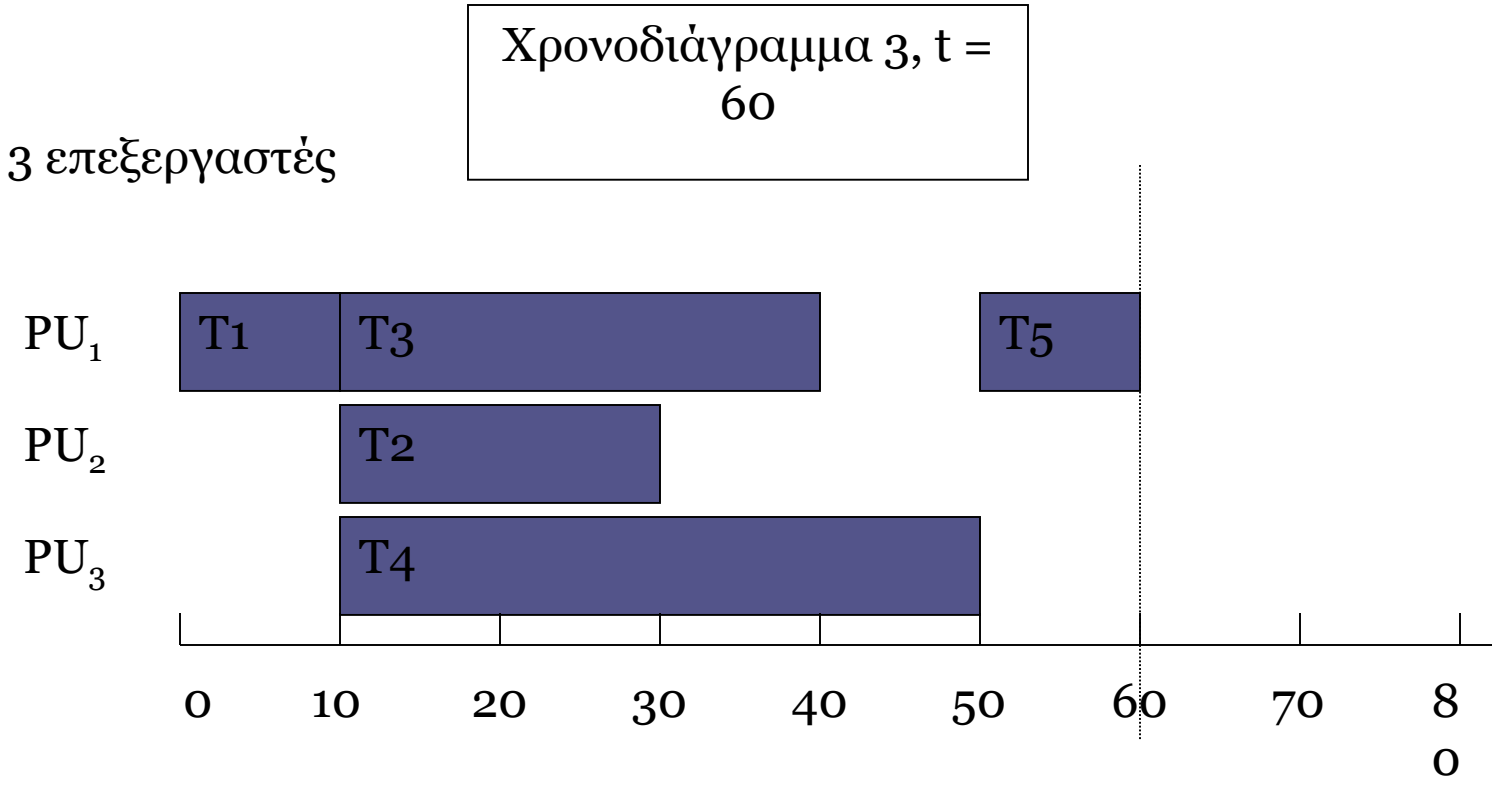


Ντετερμινιστικό scheduling με μηδενικό κόστος επικοινωνίας (3)

Χρονοδιάγραμμα 2, t = 70



Ντετερμινιστικό scheduling με μηδενικό κόστος επικοινωνίας (4)



Βέλτιστο χρονοδιάγραμμα

- Η βέλτιστη λύση δύσκολη σε γενικές γραμμές.
- Εκθετικά αυξανόμενος αριθμός πιθανών σεναρίων.
  - Πρόβλημα NP-complete.

- Χρήση ευρηστικών αλγορίθμων.
- Βέλτιστες λύσεις σε ειδικές περιπτώσεις.

NP-completeness

- Στην υπολογιστική θεωρία πολυπλοκότητας, ένα πρόβλημα είναι **NP-complete** όταν:
- Επιλύεται από μια μη ντετερμινιστική μηχανή Τούρινγκ σε πολυωνυμικό χρόνο.
- Επιλύεται από μια ντετερμινιστική μηχανή Τούρινγκ σε μεγάλο (εκθετικό) χρόνο. Έλεγχος λύσεων σε πολυωνυμικό χρόνο.
- Χρησιμοποιείται για την εξομοίωση άλλων προβλημάτων με παρόμοιο τρόπο λύσης
- Κάθε είσοδος του προβλήματος αντιστοιχίζεται με ένα σύνολο λύσεων πολυωνυμικού μήκους. Η ορθότητα των λύσεων μπορεί να ελεγχθεί σε πολυωνυμικό χρόνο.
- Η έξοδος για κάθε είσοδο είναι «ναι» αν το σύνολο λύσεων δεν είναι κενό. Αλλιώς η έξοδος είναι «όχι».

Ειδ. περίπτωση 1: Δάσος εισόδου (εξόδου)

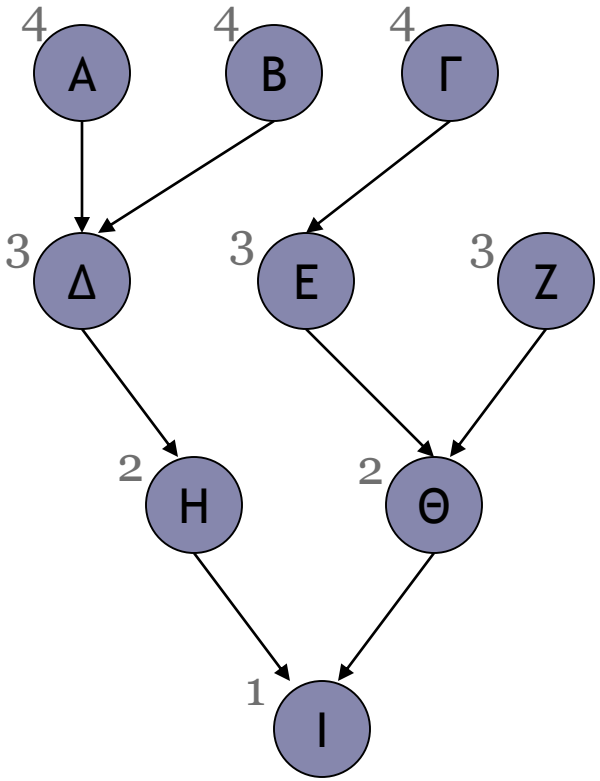
- Ορισμός:
- Δάσος εισόδου (εξόδου) καλείται ένας γράφος όπου κάθε κόμβος έχει μόνο μια ακμή εξόδου (εισόδου).

Scheduling με δάσος εισόδου-εξόδου

- Υπάρχει η έννοια των επιπέδων.
- Προτεραιότητα με βάση το επίπεδο.
- Αναθέτουμε πρώτα τις εργασίες με τη μεγαλύτερη προτεραιότητα στον πρώτο διαθέσιμο επεξεργαστή.
- Σεβόμαστε τις ακμές εξάρτησης.

Scheduling με δάσος εισόδου

Παράδειγμα



επίπεδο 4

επίπεδο 3

επίπεδο 2

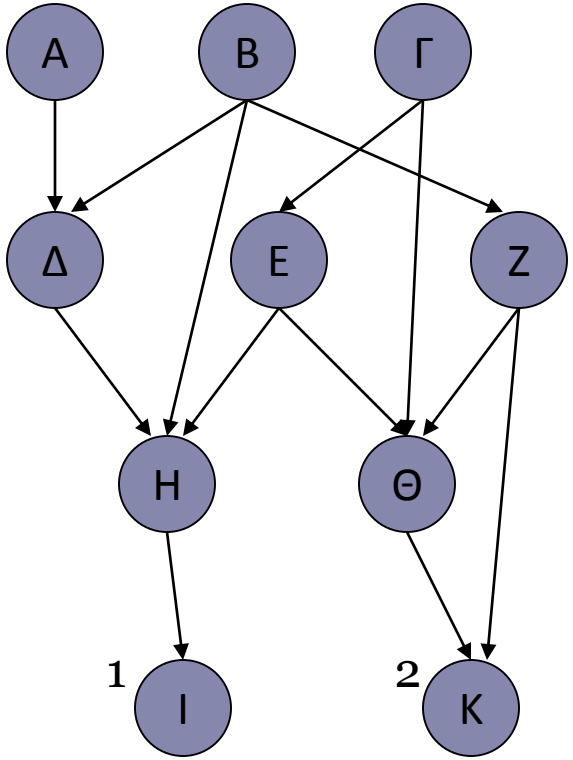
επίπεδο 1

PU1	A^4	B^4	Δ^3	Θ^2	I^1
PU2	Γ^4	E^3	Z^3	H^2	

PU1	A^4	Δ^3	Θ^2	I^1
PU2	B^4	E^3	H^2	
PU3	Γ^4	Z^3		

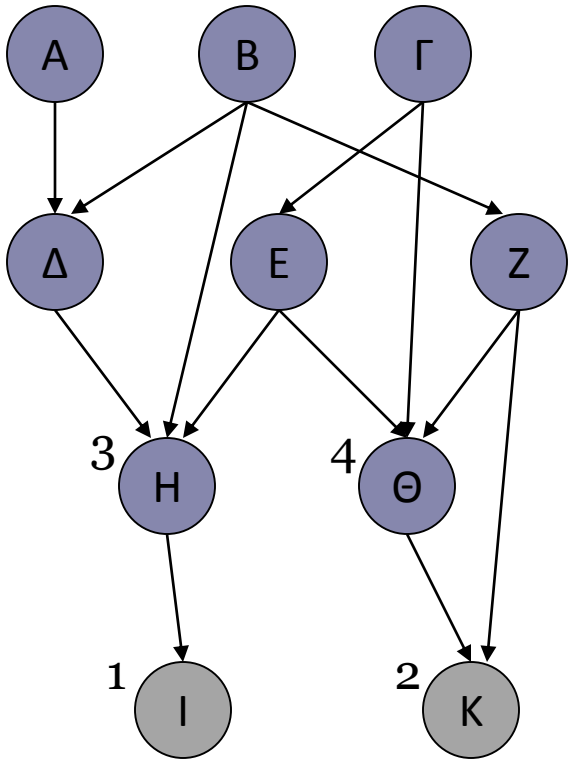
Scheduling: Δύο επεξεργαστές (1)

Παράδειγμα



Βήμα 1: δίνουμε αυθαίρετα προτεραιότητα 1, 2 στα φύλλα Ι, Κ (κόμβους χωρίς παιδιά)

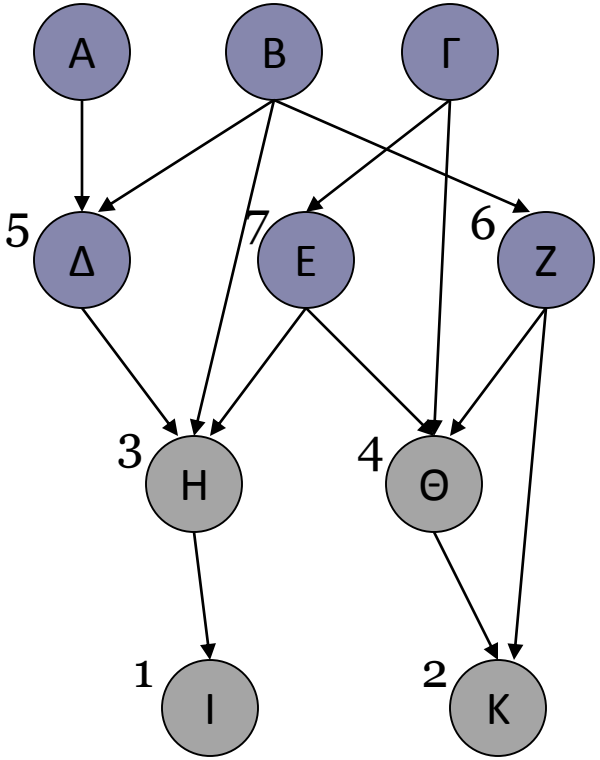
Scheduling: Δύο επεξεργαστές (2)



Βάφουμε γκρι τα φύλλα που μόλις πήραν προτεραιότητα

Βήμα 2: δίνουμε προτεραιότητα σε όλους τους κόμβους (Η,Θ) που έχουν όλα τα παιδιά τους γκρι

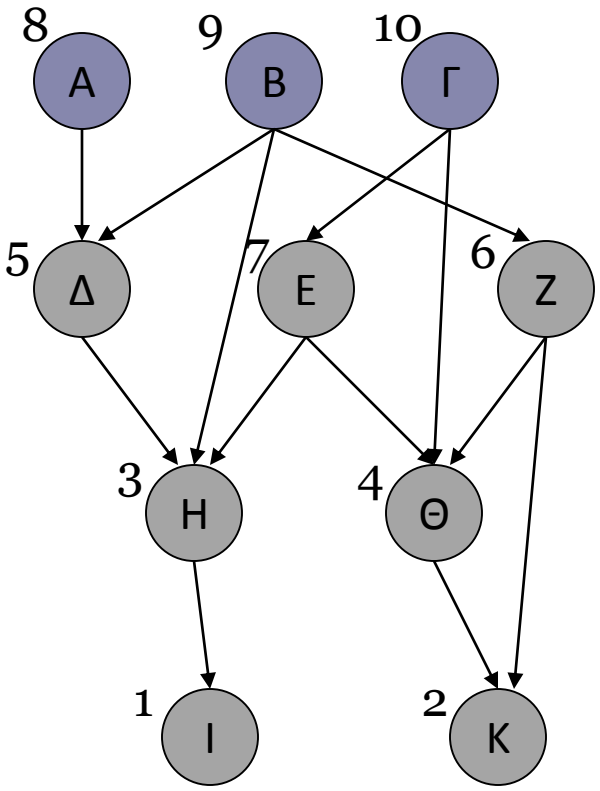
Scheduling: Δύο επεξεργαστές (3)



Βάφουμε γκρι τα φύλλα που μόλις πήραν προτεραιότητα

Βήμα 2: δίνουμε προτεραιότητα σε όλους τους κόμβους (Δ,Ε,Ζ) που έχουν όλα τα παιδιά τους γκρι

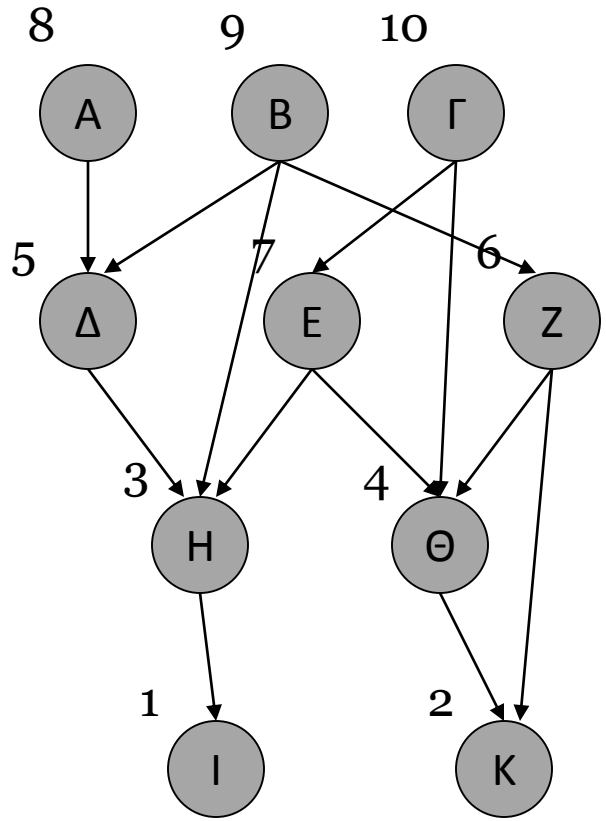
Scheduling: Δύο επεξεργαστές (4)



Βάφουμε γκρι τα φύλλα που μόλις πήραν προτεραιότητα

Βήμα 2: δίνουμε προτεραιότητα σε όλους τους κόμβους (Α,Β,Γ) που έχουν όλα τα παιδιά τους γκρι

Scheduling: Δύο επεξεργαστές (5)



Βήμα 3: Αναθέτουμε εργασίες στους δύο επεξεργαστές με σειρά προτεραιότητας

PU1	Γ^{10}	A^{8}	Z^{6}	Θ^{4}	K^{2}
PU2	B^{9}	E^{7}	Δ^{5}	H^{3}	I^{1}