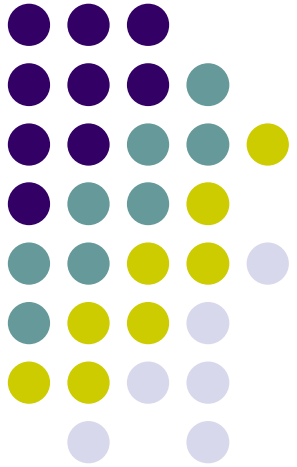
Κατανεμημένα Συστήματα

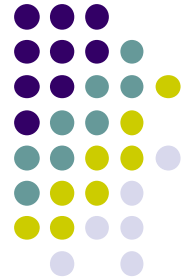
Μάθημα #7



Περιεχόμενα

- **Καθολική κατάσταση κατανεμημένου υπολογισμού**
- **Αλγόριθμοι Εκλογής Αρχηγού**
- **Αλγόριθμος Εκλογής Αρχηγού σε Δακτύλιο**
- **Αλγόριθμος Εκλογής Αρχηγού σε Δένδρο**

Καθολική ιστορία κατανεμημένου υπολογισμού



- Η **τοπική ιστορία** μιας διεργασίας P_i συμβολίζεται με

$$h_i = e_i^1 e_i^2 \cdots e_i^{ci}$$

και αποτελεί την ακολουθία γεγονότων που έχουν εκτελεστεί στην P_i

- Η **καθολική ιστορία** Η ενός κατανεμημένου υπολογισμού ορίζεται ως η ένωση των τοπικών ιστοριών όλων των διεργασιών που συμμετέχουν σε αυτόν, δηλ.,

$$H = h_1 \cup h_2 \cup \ldots \cup h_n$$

Καθολική κατάσταση (global state) κατανεμημένου υπολογισμού

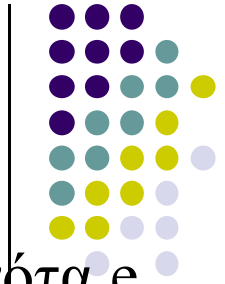
Η **τοπική κατάσταση** μιας διεργασίας P_i αμέσως μετά την εκτέλεση του γεγονότος e_i^k συμβολίζεται με σ_i^k και περιέχει τις τιμές όλων των τοπικών μεταβλητών της.

Η **καθολική κατάσταση** Σ ενός κατανεμημένου υπολογισμού είναι η ένωση όλων των τοπικών καταστάσεων των επιμέρους διεργασιών $\Sigma=(\sigma_1^{k1}, \sigma_2^{k2}, \dots, \sigma_n^{kn})$

Η **καθολική κατάσταση μπορεί να αναπαρασταθεί γραφικά με την τομή.** Η **τομή** είναι ένα υποσύνολο της καθολικής ιστορίας, δηλαδή $C = (h_1^{c1}, h_2^{c2}, \dots, h_n^{cn})$ και προσδιορίζεται μέσω του διανύσματος (c1,c2,...,cn)

Το σύνολο των τελευταίων γεγονότων $(e_1^{c1}, e_2^{c2}, \dots, e_n^{cn})$ καλείται **σύνορο της τομής**

Καθολική κατάσταση (global state)



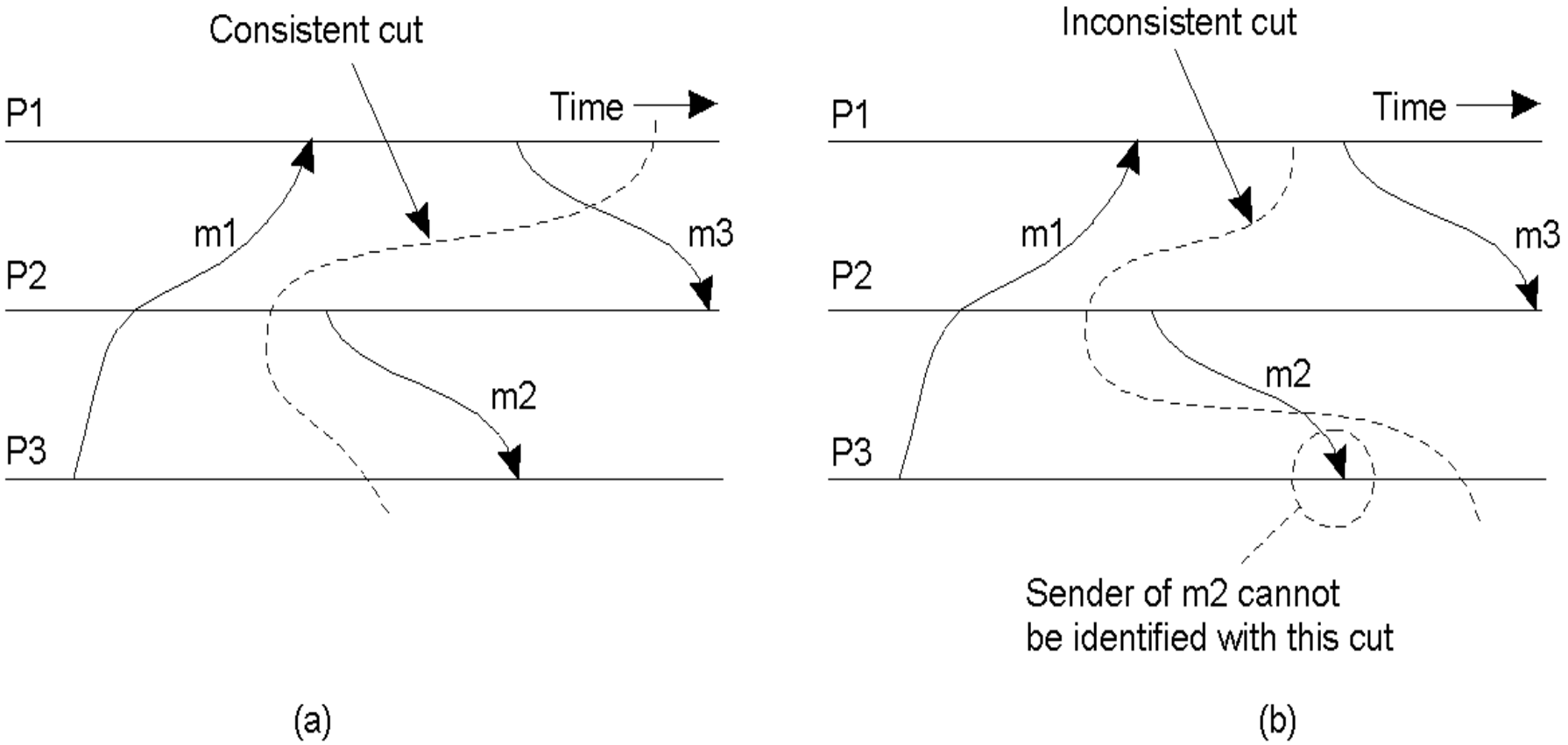
Μία **τομή C** είναι **συνεπής (consistent)** αν για όλα τα γεγονότα e και e' ισχύει ότι

$$\forall e', e: \ (e \in C) \wedge (e' \rightarrow e) \Rightarrow e' \in C$$

Διαφορετικά, η τομή καλείται **ασυνεπής (inconsistent).**

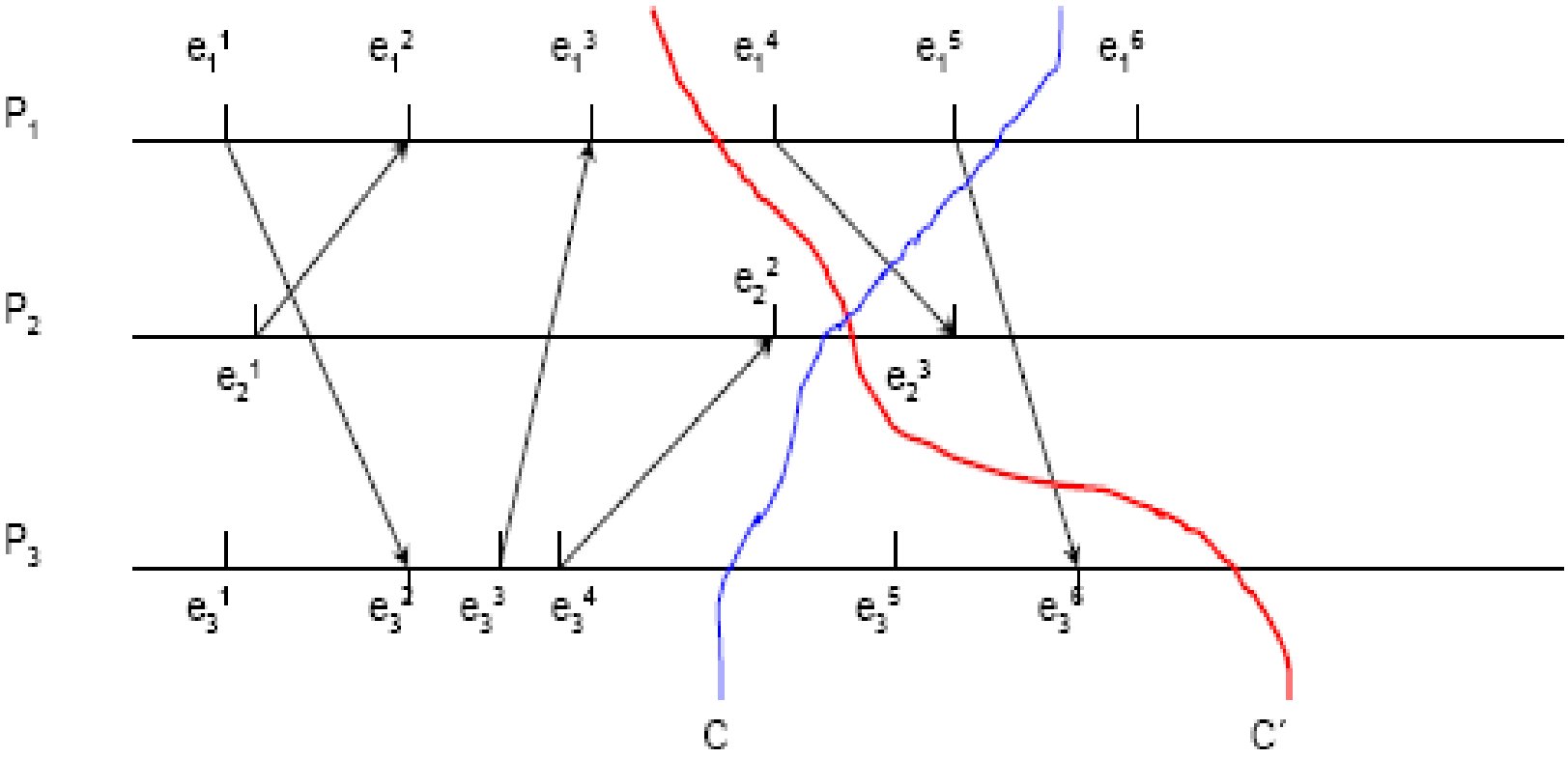
Αντίστοιχα, μια **καθολική κατάσταση είναι συνεπής** αν αντιστοιχεί σε μία συνεπή τομή.

Καθολική κατάσταση (global state)



(a) Συνεπής τομή **(b) Ασυνεπής τομή**

Καθολική κατάσταση (global state)



Consistent

$\Sigma=(\sigma_1^5, \sigma_2^2,\ldots,\sigma_3^4)$

Inconsistent

$\Sigma=(\sigma_1^3, \sigma_2^2,\ldots,\sigma_3^6)$

$(e_3^6 \in C') \wedge (e_1^5 \rightarrow e_3^6)$, but $e_1^5 \notin C'$

Κατασκευή καθολικών καταστάσεων

Αν μία διεργασία θέλει να γνωρίζει την καθολική κατάσταση ενός κατανεμημένου υπολογισμού πρέπει να την **«συλλέξει»** με τρόπο ώστε αυτή να **μην είναι**

- **ξεπερασμένη (obsolete)** – να μην αντικατοπτρίζει παρελθούσα κατάσταση του συστήματος
- **ατελής (incomplete)** –να περιέχει τις τοπικές καταστάσεις όλων των διεργασιών
- **ασυνεπής**

Η κατασκευή καθολικών καταστάσεων γίνεται από μια διεργασία – ενεργοποιητή. Αυτή ακολουθεί μια από τις εξής δύο στρατηγικές:

1. **Παθητική: Όλες οι διεργασίες μόλις εκτελέσουν ένα γεγονός στέλνουν μήνυμα στην διεργασία – ενεργοποιητή, η οποία συλλέγει τις απαντήσεις και συνθέτει την καθολική κατάσταση**
2. **Ενεργητική: Η διεργασία – ενεργοποιητής ζητάει από τις υπόλοιπες διεργασίες τις τοπικές τους καταστάσεις για να συνθέσει την καθολική κατάσταση**

Αλγόριθμοι Εκλογής Αρχηγού

Πρόβλημα: Επιλογή μίας μόνο διεργασίας αρχηγού/συντονιστή προκειμένου να εκτελέσει ένα συγκεκριμένο καθήκον (εκτέλεση συγκεντρωτικών αλγορίθμων, επαναφορά από αδιέξοδο, να αποτελέσει τη διεργασία-ρίζα στη δημιουργία ενός δένδρου επικάλυψης)

Δεν μπορεί αυθαίρετα μια διεργασία να αυτοανακηρυχθεί αρχηγός. Θα πρέπει να προηγηθεί η εκτέλεση ενός αλγορίθμου εκλογής.

- **Κάθε διεργασία πρέπει να αποφασίσει αν είναι ο αρχηγός ή όχι.**
- **Μία μόνο από τις διεργασίες θα πρέπει να αποφασίσει πως είναι ο αρχηγός.**

Στόχος ενός αλγορίθμου εκλογής *είναι να διασφαλίσει ότι όταν ξεκινήσει η διαδικασία εκλογής θα καταλήξει με τη συμφωνία όλων των διεργασιών ως προς το ποιος είναι ο νέος αρχηγός.*

Αλγόριθμοι Εκλογής Αρχηγού

Αρχικά

– ένα αυθαίρετο μη κενό σύνολο διεργασιών

– όλες οι διεργασίες ξεκινάνε από την ίδια κατάσταση (candidate)

Κάθε διεργασία εκτελεί τον ίδιο αλγόριθμο

Υπάρχουν δύο είδη τερματικών καταστάσεων για μια διεργασία:

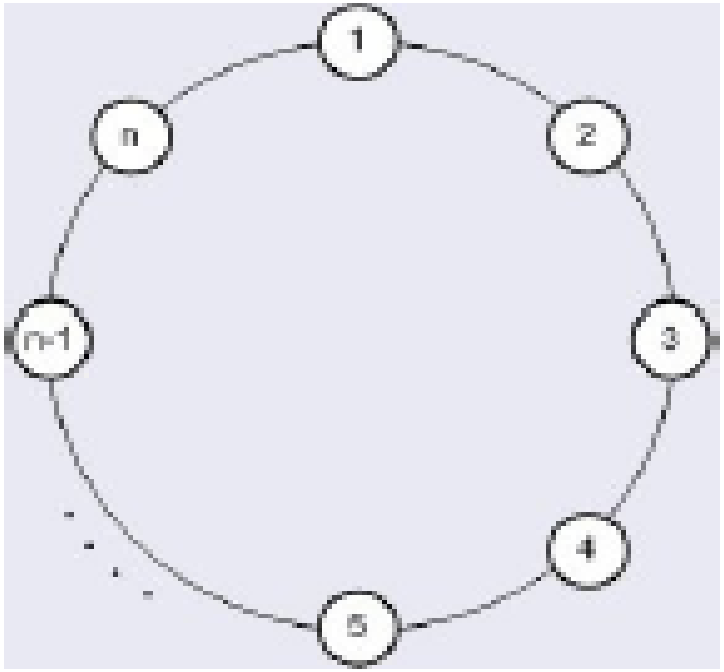
- **η τερματική κατάσταση στην οποία η διεργασία έχει εκλεγεί αρχηγός (*κατάσταση ισχύος ή αρχηγού*) και**
- **η τερματική κατάσταση στην οποία η διεργασία δεν είναι ο αρχηγός (*κατάσταση μη-ισχύος ή χαμένου*).**

Επιτρεπτές εκτελέσεις:

- **Κάθε διεργασία τελικά εισέρχεται σε μια τερματική κατάσταση, ισχύος ή μη.**
- **Μόνο μια διεργασία, ο αρχηγός, μπαίνει σε τερματική κατάσταση ισχύος.**

Εκλογή Αρχηγού σε Δακτύλιο

Έστω ένα κατανεμημένο
σύστημα από *n* διεργασίες τοποθετημένες
σε ένα δίκτυο δακτυλίου.



- **Οι διεργασίες έχουν μοναδικές ταυτότητες (IDs).**
- **Οι διεργασίες δεν γνωρίζουν τις ταυτότητες των υπόλοιπων διεργασιών**

Κάθε διεργασία γνωρίζει μόνο ποιος είναι ο αριστερός *(σύμφωνα με τους δείκτες του ρολογιού)* και ποιος ο δεξιός *(αντίθετα με τους δείκτες του ρολογιού)* της γείτονας.

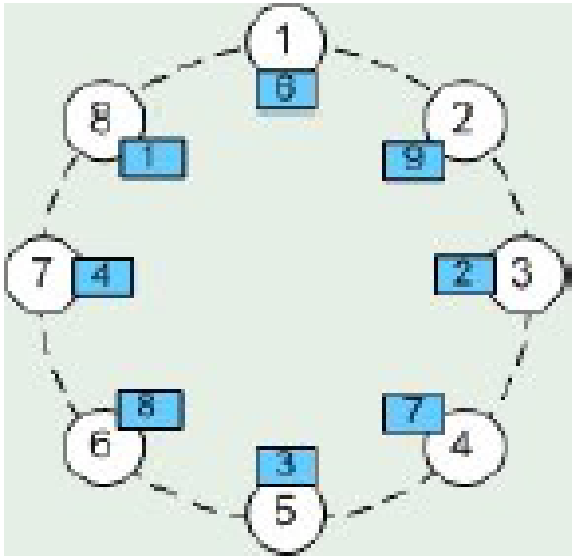
Θεωρούμε ότι οι διεργασίες είναι αριθμημένες από 1 έως n χωρίς οι ίδιες να γνωρίζουν αυτή την αρίθμηση

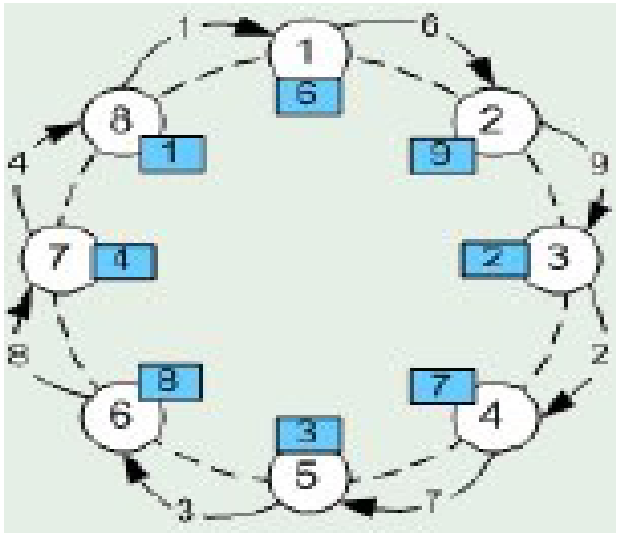
Αλγόριθμος Εκλογής Αρχηγού σε Δακτύλιο

Ενέργειες κάθε διεργασίας p:

- **Αποστολή του ID της στα αριστερά.**
- **Όταν η p λάβει ένα ID (από δεξιά) κάνει τα εξής:**
  - **αν είναι μεγαλύτερο από το δικό της, το προωθεί προς τα αριστερά**
  - **αν είναι μικρότερο από το δικό της, το αγνοεί (και δεν το προωθεί)**
  - **αν είναι ίσο με το δικό της, αποφασίζει πως αυτή είναι ο αρχηγός στο σύστημα και στέλνει ένα μήνυμα τερματισμού προς τα αριστερά**
- **Όταν η p λάβει μήνυμα τερματισμού, το προωθεί προς τα αριστερά και εισέρχεται σε τερματική κατάσταση μη-ισχύος.**

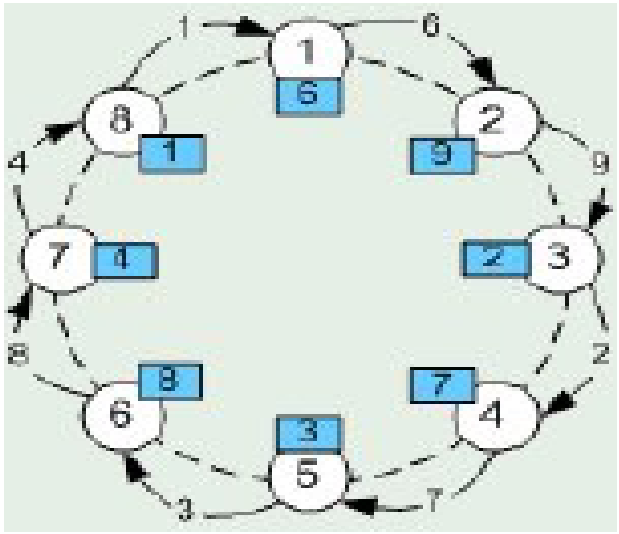
Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο



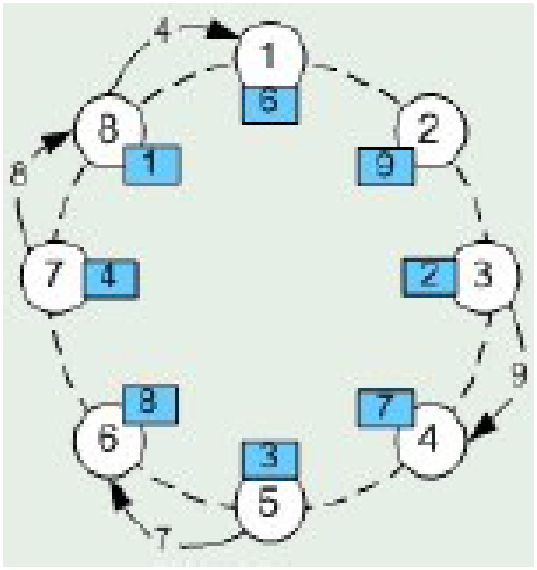


Βήμα 1

Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο

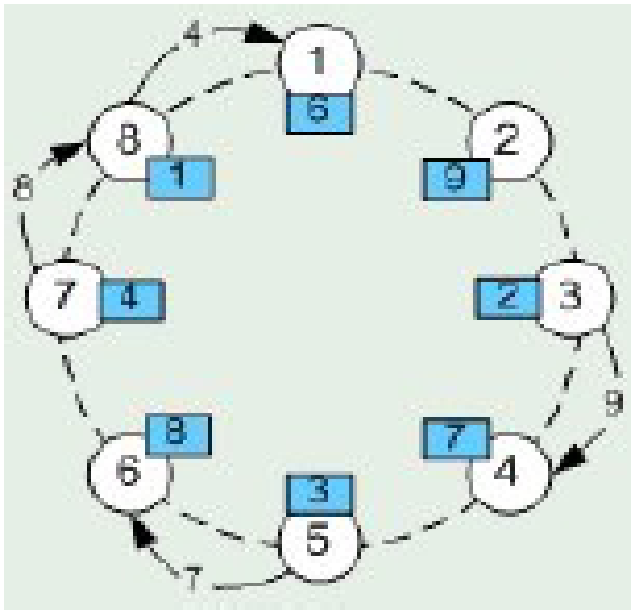


Βήμα 1

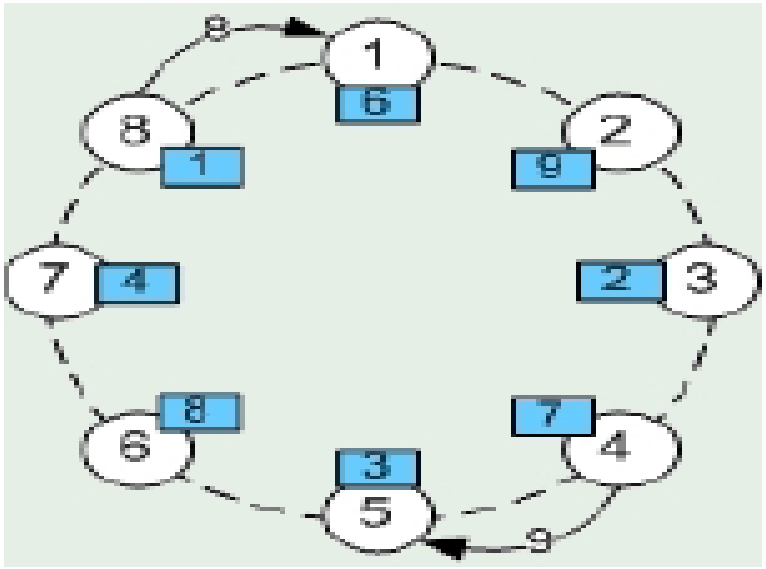


Βήμα 2

Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο

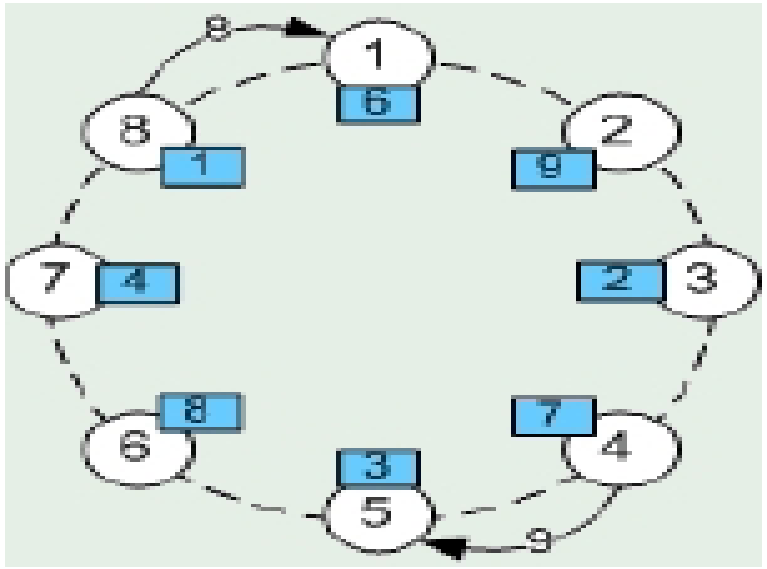


Βήμα 2

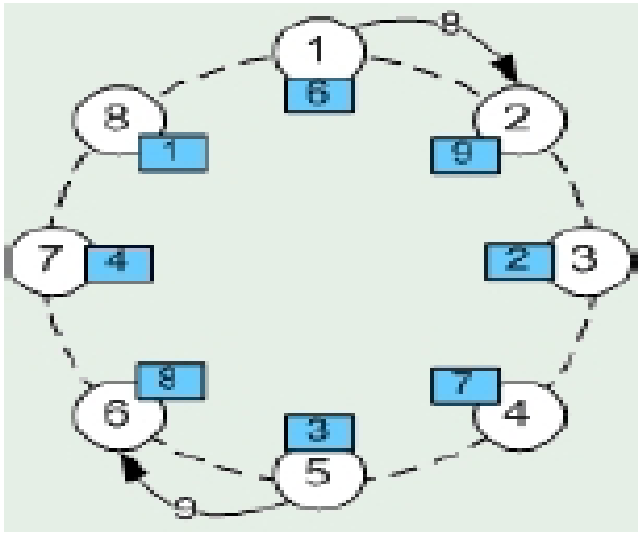


Βήμα 3

Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο

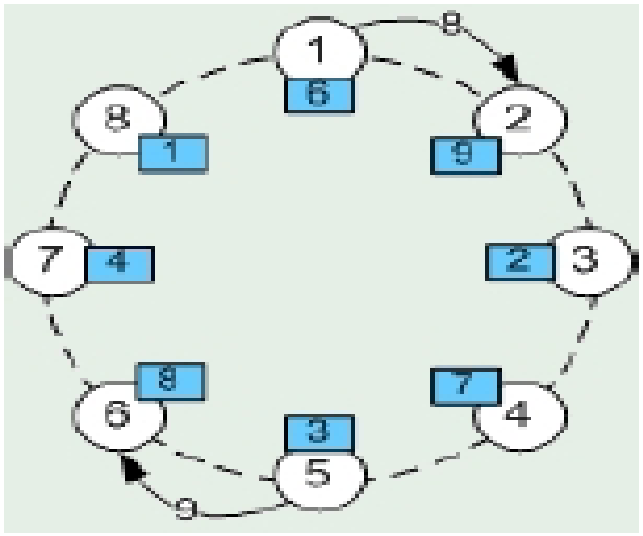


Βήμα 3

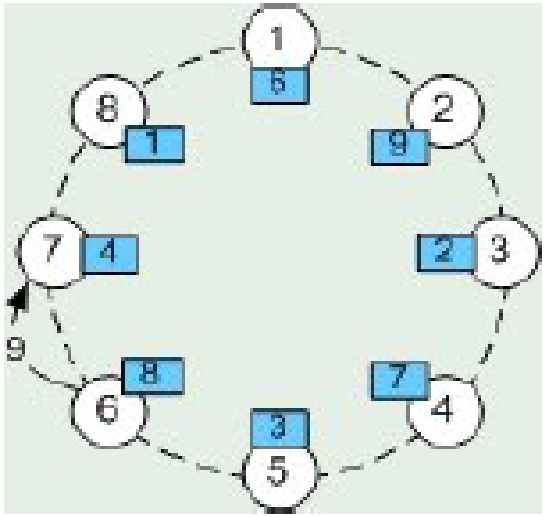


Βήμα 4

Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο

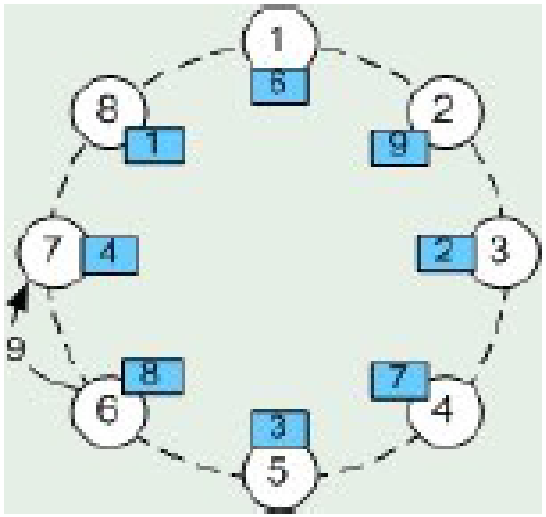


Βήμα 4

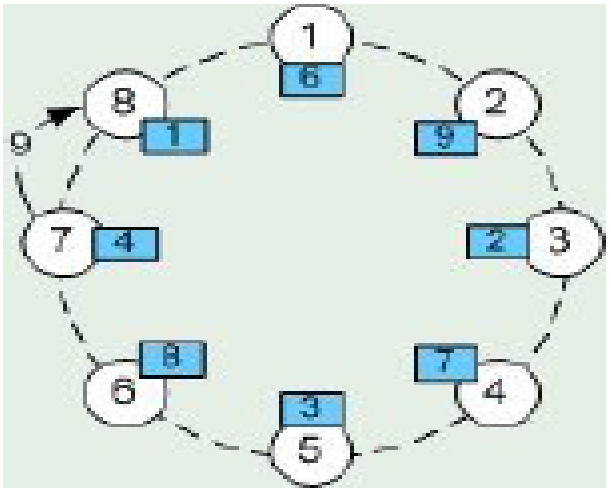


Βήμα 5

Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο

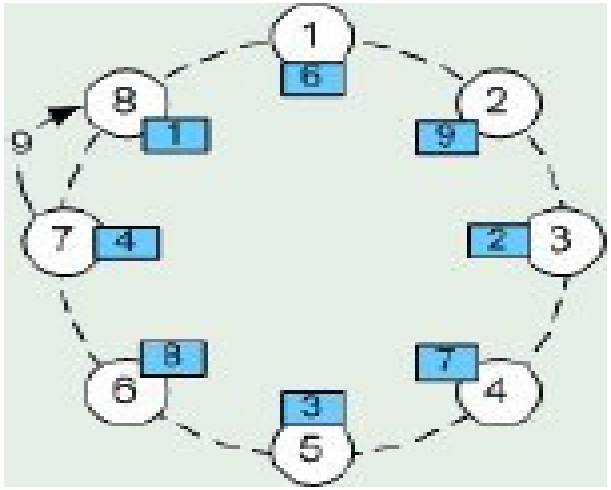


Βήμα 5

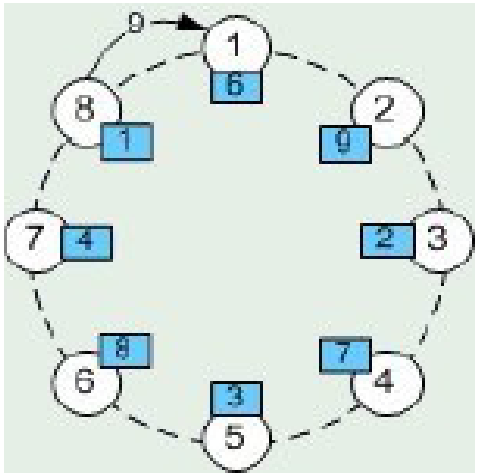


Βήμα 6

Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο

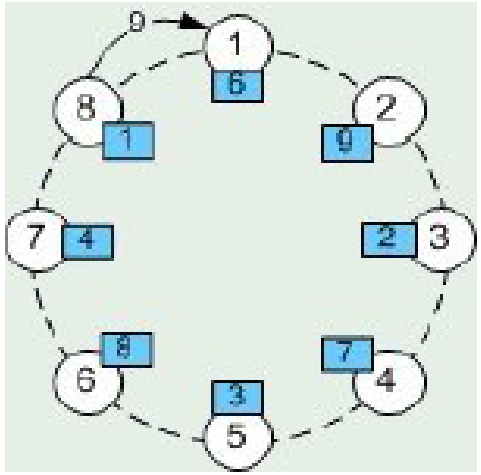


Βήμα 6

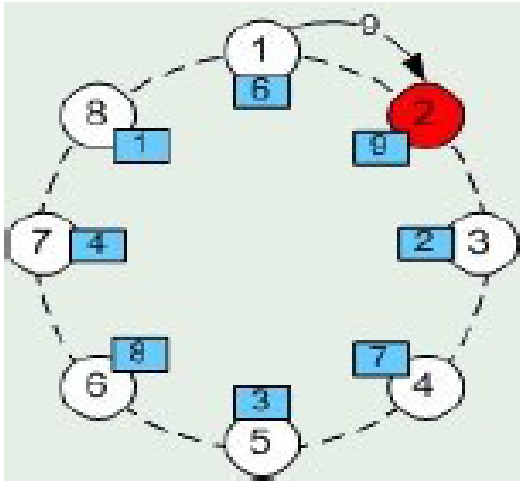


Βήμα 7

Εκτέλεση αλγορίθμου εκλογής αρχηγού σε σύγχρονο δακτύλιο

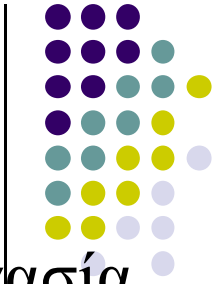


Βήμα 7



Βήμα 8

Ορθότητα - Πολυπλοκότητα



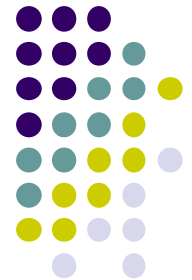
Ορθότητα αλγορίθμου: Θα **εκλεγεί ως αρχηγός** η διεργασία με **το μεγαλύτερο ID**. Το μήνυμα με αυτό το ID θα περάσει από όλες τις διεργασίες.

Χρονική πολυπλοκότητα: $2n = O(n)$

Η Πολυπλοκότητα Επικοινωνίας **εξαρτάται από τη διάταξη των διεργασιών**

- Το μέγιστο ID θα περάσει από όλες τις διεργασίες (*n* μηνύματα)
- Το δεύτερο μέγιστο ID θα ταξιδέψει έως ότου συναντήσει το μέγιστο ID
- Το τρίτο μέγιστο ID θα ταξιδέψει έως ότου συναντήσει το μέγιστο ή το δεύτερο μέγιστο ID.
- κ.ο.κ

Πολυπλοκότητα Επικοινωνίας: απαιτούνται $O(n^2)$ μηνύματα



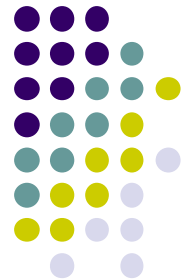
Η **χειρότερη διάταξη των IDs** είναι σε φθίνουσα σειρά κατά την ωρολογιακή φορά

Τότε:

- το δεύτερο μέγιστο ID συνεισφέρει *n-1* μηνύματα
- το τρίτο μέγιστο ID συνεισφέρει *n-2* μηνύματα
- το τέταρτο μέγιστο ID συνεισφέρει *n-3* μηνύματα
- κ.οκ.

Άρα συνολικά απαιτούνται $\sum_{i=1}^{n}(n-i+1) = \sum_{i=1}^{n} i = \Theta(n^2)$ μηνύματα

Εκλογή Αρχηγού σε Δακτύλιο που απαιτεί *O(nlogn)* μηνύματα (διατηρώντας τη χρονική πολυπλοκότητα σε O(n)) (Αλγόριθμος Hirschberg-Sinclair)



m*-γειτονιά** μιας διεργασίας ***p: **το σύνολο των διεργασιών που βρίσκονται σε απόσταση το πολύ *m* από την *p* στο δακτύλιο** (είτε προς τα αριστερά ή προς τα δεξιά).

Περιγραφή Αλγορίθμου

Λειτουργεί σε $k=0,\ldots,\log n - 1$ φάσεις.

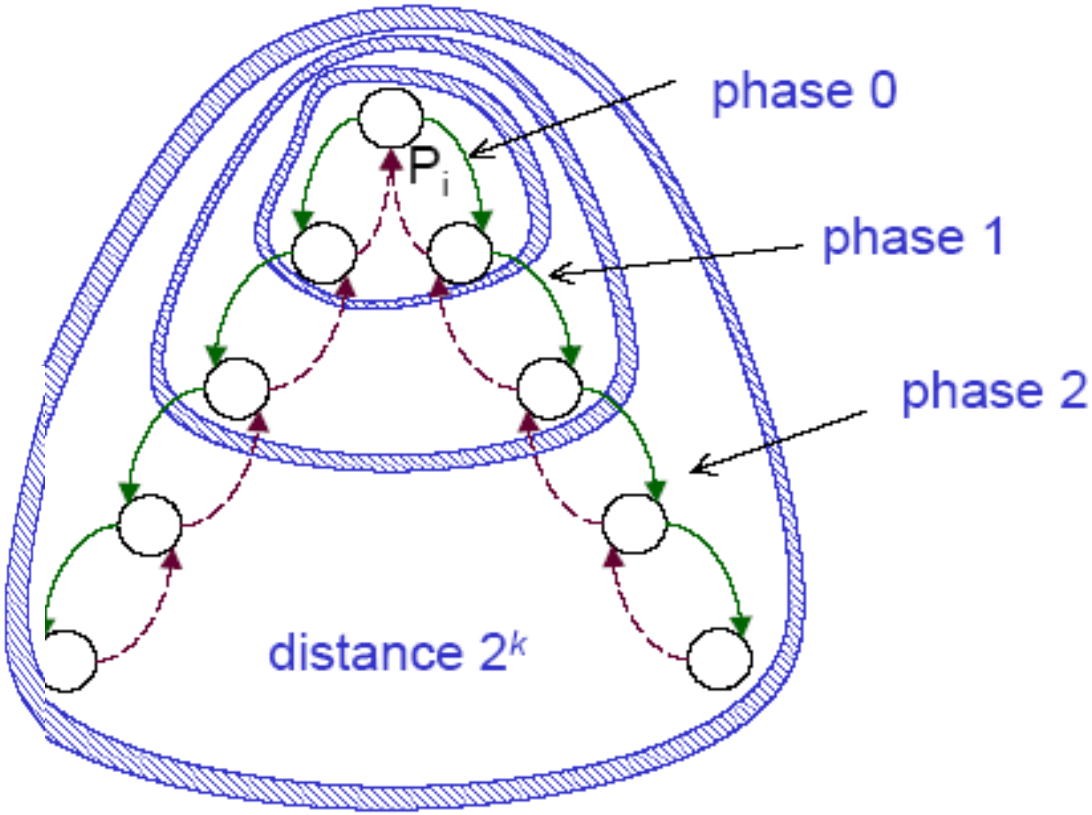
- Στην ***k*-οστή** φάση, ένας επεξεργαστής προσπαθεί να γίνει ο προσωρινός αρχηγός της ***2^k*-γειτονιάς** του.
- Μόνο οι επεξεργαστές που εκλέγονται αρχηγοί στην ***k*-οστή** φάση θα συνεχίσουν στην ***(k+1)*-οστή** φάση.

Αλγόριθμος Hirschberg-Sinclair

Περιγραφή *k-οστής Φάσης*

- Κάθε διεργασία *p*, που εκλέχθηκε προσωρινός αρχηγός στην *(k-1)-οστή φάση*, στέλνει μηνύματα <*probe*> με το ID της σε όλους τους κόμβους στην ***2^k-γειτονιά*** της. Κάθε μήνυμα <*probe*> περιέχει τον αριθμό της τρέχουσας φάσης ***k***, και έναν μετρητή ***d*** (του μήκους του μονοπατιού που έχει διανυθεί).
- Μια διεργασία αγνοεί ένα μήνυμα τύπου <*probe*>, αν αυτό περιέχει ID που είναι μικρότερο από το δικό της.
- Όταν ένα μήνυμα τύπου <*probe*> φθάσει στην τελευταία διεργασία στη τρέχουσα γειτονιά, τότε αυτή η διεργασία στέλνει στην *p* ένα μήνυμα τύπου <*reply*>.
- Αν η *p* λάβει το <*reply*> και από τις δύο κατευθύνσεις, αποφασίζει πως είναι ο αρχηγός της ***2^k-γειτονιάς*** της στη φάση *k*.
- Η διεργασία που θα λάβει το δικό της μήνυμα τύπου <*probe*>, τερματίζει σε κατάσταση ισχύος (στέλνοντας μήνυμα τερματισμού στις υπόλοιπες).

Αλγόριθμος Hirschberg-Sinclair



Algorithm 5 Asynchronous leader election: code for processor p_i, $0 \leq i < n$.

Initially, *asleep* = true

```
1:  upon receiving no message:
2:      if asleep then
3:          asleep := false
4:          send ⟨probe,id,0,1⟩ to left and right

5:  upon receiving ⟨probe,j,k,d⟩ from left (resp., right):
6:      if j = id then terminate as the leader
7:      if j > id and d < 2^k then                          // forward the message
8:          send ⟨probe,j,k,d + 1⟩ to right (resp., left)   // increment hop counter
9:      if j > id and d ≥ 2^k then                          // reply to the message
10:         send ⟨reply,j,k⟩ to left (resp., right)
                                          // if j < id, message is swallowed

11: upon receiving ⟨reply,j,k⟩ from left (resp., right):
12:     if j ≠ id then send ⟨reply,j,k⟩ to right (resp., left)   // forward the reply
13:     else                                                     // reply is for own probe
14:         if already received ⟨reply,j,k⟩ from right (resp., left) then
15:             send ⟨probe,id,k + 1,1⟩ to left and right !      // phase k winner
```

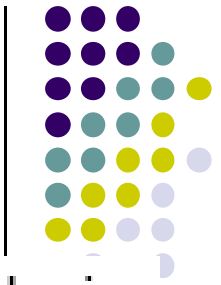
Αλγόριθμος Hirschberg-Sinclair
Πολυπλοκότητα μηνυμάτων

- **Κάθε μήνυμα ανήκει σε μία φάση και στέλνεται από μία συγκεκριμένη διεργασία**
- **Η απόσταση εξερεύνησης στη φάση *k* είναι 2^k , $k>=0$**
- **Ο αριθμός των μηνυμάτων που αποστέλλονται προερχόμενα από μία συγκεκριμένη διεργασία στη φάση *k* είναι το πολύ $4 \cdot 2^k$ (σήματα <*probe*> και <*reply*>)**
- **Ο αριθμός των διεργασιών που εκλέγονται αρχηγοί στη φάση *k* είναι το πολύ $n/(2^k+1)$.**

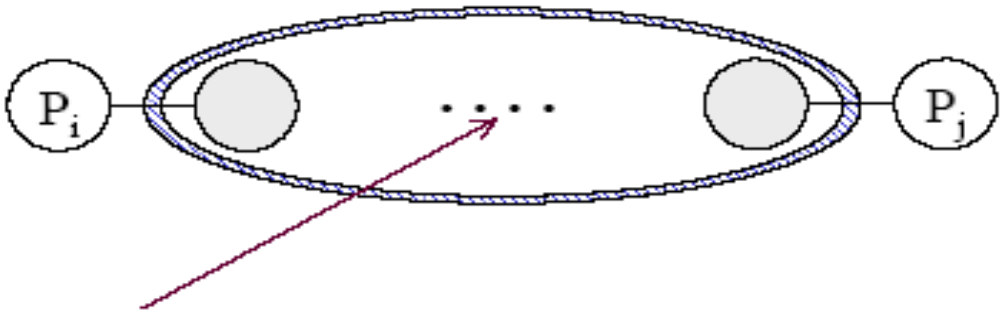
  Δύο αρχηγοί της *k-οστής* φάσης θα πρέπει να έχουν ανάμεσά τους τουλάχιστον 2^k διεργασίες αφού για να εκλεγούν αρχηγοί θα πρέπει να έχουν το μεγαλύτερο ID σε μια ακτίνα 2^k γύρω τους.

- **Υπάρχουν το πολύ log n φάσεις αφού σε κάθε φάση διπλασιάζεται η απόσταση εξερεύνησης**
- **Συνολικά μηνύματα:** $4 \cdot 2^k \cdot n/(2^k+1) \cdot \log n = O(n \log n)$
- **Χρονική πολυπλοκότητα:** $2 \cdot (2^0+2^1+2^2+\ldots+2^{\log n-1}) = O(n)$

Αλγόριθμος Hirschberg-Sinclair
Πολυπλοκότητα μηνυμάτων



The closest together than two *k* temporal leader, P_i and P_j, can be is if the left side of P_i's *k*-neighbourhood is exactly the right side of P_j's *k*-neighbourhood.

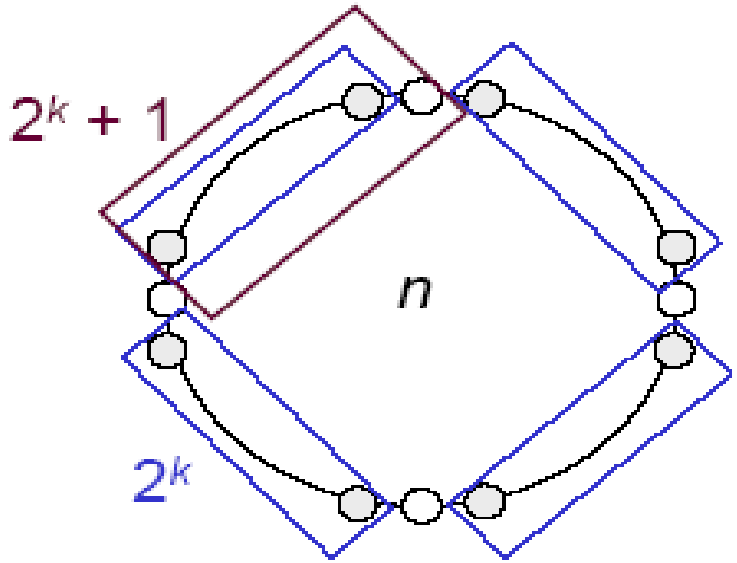


That is there are 2^k processes in between P_i and P_j.

The maximum number of phase *k* temporary leaders is achieved when this pattern continues around the ring.

Thus, in a group of $2^k + 1$, at most one can initiate messages along paths of length 2^{k+1}.

Αλγόριθμος Hirschberg-Sinclair
Πολυπλοκότητα μηνυμάτων



The number of leader in this case is:

$$\frac{n}{2^k + 1}$$

Αλγόριθμος Hirschberg-Sinclair
Συνολικός αριθμός μηνυμάτων

Ο συνολικός αριθμός μηνυμάτων είναι

$$
\begin{aligned}
&\leq 4 \cdot n + \sum_{i=1}^{\lg n} 4 \cdot 2^i \cdot \frac{n}{2^{i-1}+1} + n \\
&\leq 5n + 4n \cdot \sum_{i=1}^{\lg n} \frac{2^i}{2^{i-1}} \\
&= 5n + 4n \cdot \sum_{i=1}^{\lg n} 2 \\
&= 5n + 8n \lg n \\
&\in O(n \lg n) .
\end{aligned}
$$

Εκλογή Αρχηγού σε Δένδρο

- Δίκτυα με τοπολογία δένδρου
- Δίκτυα στα οποία κατασκευάζεται ένα spanning tree
- Κάθε διεργασία γνωρίζει τα IDs των γειτόνων της
- Αρχηγός εκλέγεται η διεργασία με το μικρότερο ID
- Initiators (εναρκτές)
  - τουλάχιστον το σύνολο των φύλλων του δένδρου
  - όλες ξεκινούν τον αλγόριθμο στέλνοντας ένα μήνυμα <tok, ID>
- Non- Initiators
  - Οι υπόλοιπες διεργασίες στο δένδρο

Αλγόριθμος Εκλογής Αρχηγού σε Δένδρο

Κάθε διεργασία

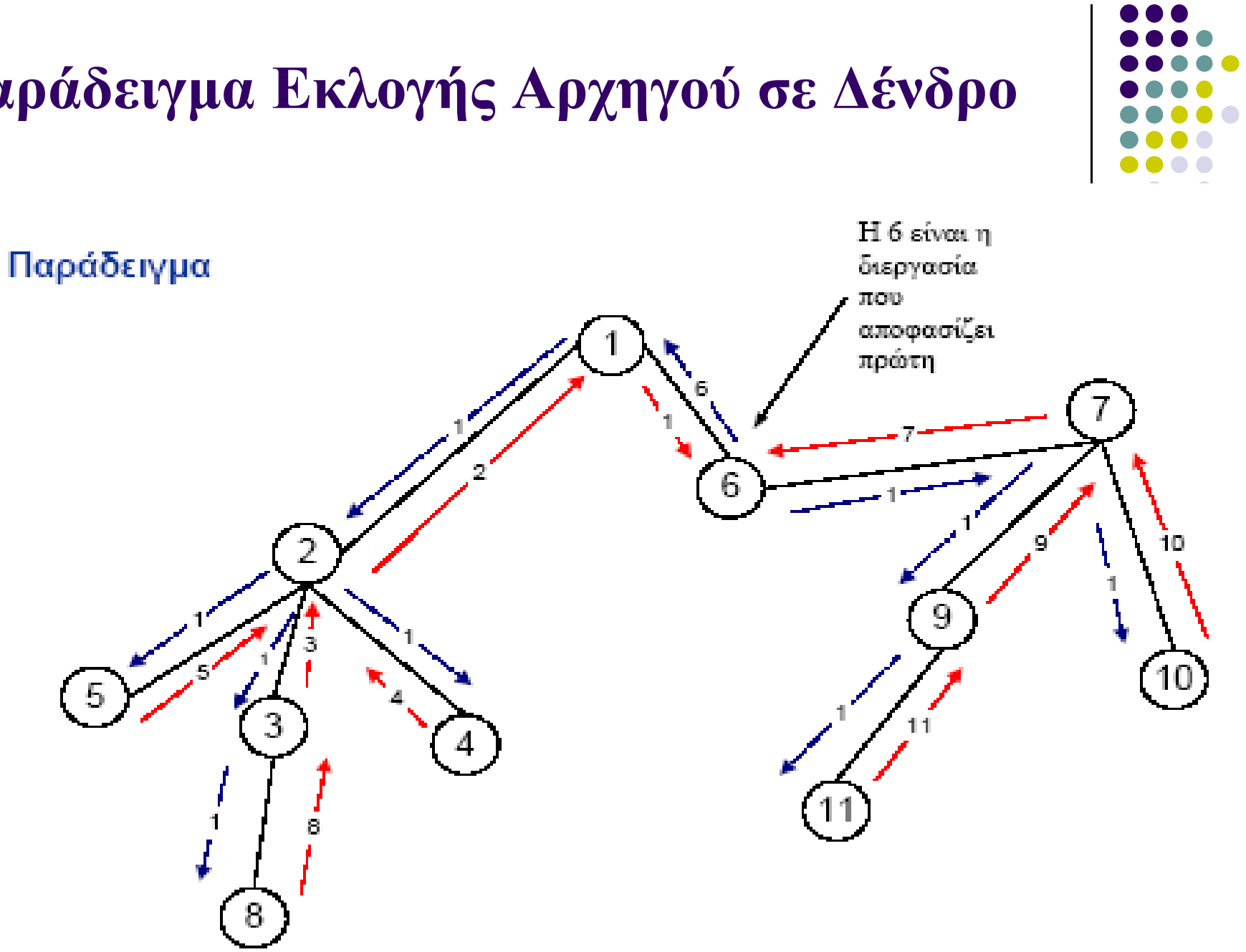
Περιμένει να λάβει μηνύματα <tok, ID> από όλους τους γείτονές της εκτός από το πολύ έναν, έστω τον P_0

Όταν ικανοποιηθεί η συνθήκη αυτή

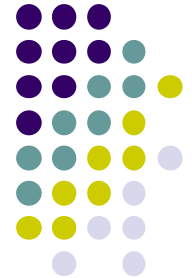
{αρχικά ισχύει μόνο για τα φύλλα του δέντρου}

- Υπολογίζει το min ID από τα ID's που έχει λάβει και το δικό της
- Στέλνει μήνυμα <tok, min ID > στον P_0
- Περιμένει μήνυμα <tok, ID> από τον P_0
- Υπολογίζει το νέο min ID από τα ID's που έχει λάβει και το δικό της
  - Αν τώρα min ID είναι το δικό της ανακηρύσσεται σε κατάσταση αρχηγού (ισχύος) – διαφορετικά ανακηρύσσεται σε κατάσταση χαμένου (μη ισχύος)
- Στέλνει σε όλους τους γείτονές της (εκτός του P_0) μήνυμα <tok,min ID >

Παράδειγμα Εκλογής Αρχηγού σε Δένδρο



Παράδειγμα Εκλογής Αρχηγού σε Δένδρο



- **5, 8, 4, 11, 10: στέλνουν <tok, min ID>**
- **3, 9: στέλνουν <tok, min ID>**
- **2,7: στέλνουν <tok, min ID>**
- **1: στέλνει <tok, min ID> (=1)**
- **6: στέλνει <tok, min ID> (=6) στην 1**

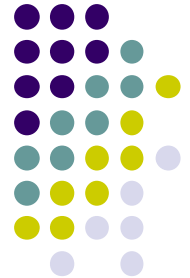
{ 11 μηνύματα <tok, min ID> }

- **6: αποφασίζει lost**
- **1: αποφασίζει leader**
- **6, 7, 9, 1, 2, 3 : στέλνουν <tok, 1> στους απογόνους τους (εκτός P_0)**

(1) (2) (1) (1) (3) (1) ***{ 9 μηνύματα <tok, 1> }***

{ συνολικά 20 μηνύματα = 2N-2 =2x11 -2 }

Εκλογή Αρχηγού σε Δένδρο



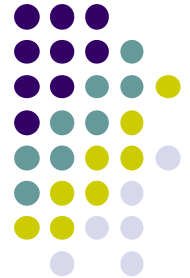
Πρόβλημα

- **Αν δεν είναι όλα τα φύλλα του δένδρου initiators τότε ο αλγόριθμος δεν δουλεύει !**

Αντιμετώπιση προβλήματος

- **Προστίθεται επιπλέον φάση *(Wake Up)* που «ξυπνάει» όλες τις υπόλοιπες διεργασίες και τις κάνει initiators**

Αλγόριθμος Wake Up



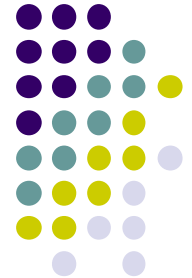
Κάθε διεργασία initiator

- **στέλνει ένα μήνυμα <wake up> σε κάθε γείτονά της**
- **περιμένει να λάβει μηνύματα <wake up> από όλους τους γείτονές της**
- **αρχίζει την εκτέλεση του αλγορίθμου εκλογής**

Κάθε διεργασία Non- Initiator όταν λάβει ένα μήνυμα <wake up>

- **γίνεται initiator**
- **στέλνει μηνύματα <wake up> σε κάθε γείτονά της**
- **περιμένει να λάβει μηνύματα <wake up> από όλους τους γείτονές της**
- **αρχίζει την εκτέλεση του αλγορίθμου εκλογής**

Αλγόριθμος Wake Up

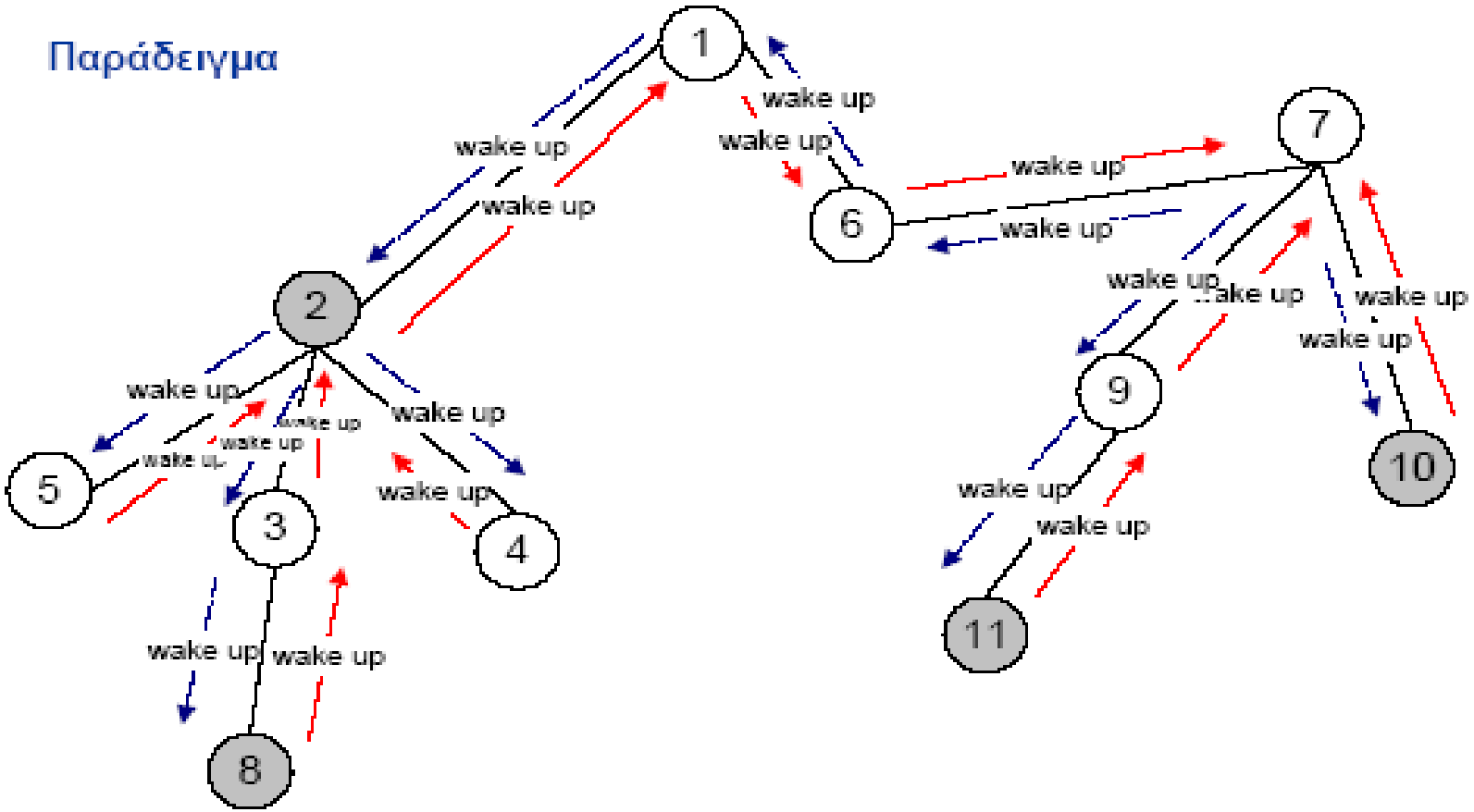


ws_p : *true* αν η διεργασία *p* έχει στείλει <*wake up*>
wr_p : # <*wake up*> μηνυμάτων που έχει λάβει η διεργασία
$Neigh_p$: τα αναγνωριστικά των γειτόνων της *p*

```
{
 Initialization: for all  p do { ws_p := false;   wr_p := 0 }

  if p is initiator then
            { ws_p := true;
              for all q ∈ Neigh_p do send <wake up> to q }
   while wr_p < #Neigh_p do
                { receive <wake up>;
                  wr_p := wr_p + 1;
                   if not ws_p then
                            { ws_p := true;
                              for all q ∈ Neigh_p do send <wake up> to q }
                  }
}
```

Παράδειγμα εκτέλεσης Wake Up



Πολυπλοκότητα Εκλογής Αρχηγού σε Δένδρο

Wake up

– **κάθε διεργασία ξεκινάει την εκτέλεση του αλγορίθμου αφού έχει λάβει το μήνυμα <wake up> από κάθε γείτονά της**

– **Από κάθε κανάλι επικοινωνίας στέλνονται 2 μηνύματα <wake up>**

– **Συνολικά στέλνονται 2N - 2 μηνύματα**

Εκλογή αρχηγού

– **Απαιτούνται 2N - 2 μηνύματα για την Wake up φάση**

– **Τα μηνύματα <tok, ID> που στέλνονται είναι 2 για κάθε κανάλι, ήτοι 2N - 2 μηνύματα**

Άρα, ο συνολικός αριθμός μηνυμάτων είναι 4N - 4