Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής
Παράλληλος Προγραμματισμός
2023-24

Συγχρονισμός threads

(Δομές συγχρονισμου των POSIX threads)

https://mixstef.github.io/courses/parprog/

Μ.Στεφανιδάκης

Συγχρονισμός threads

- Όταν εκτελούνται σε διαφορετικές μονάδες εκτέλεσης και προσπελαύνουν κοινούς πόρους
  - Πρέπει να εξασφαλίζεται η ελεγχόμενη προσπέλαση των πόρων αυτών
- Αναμονή αποτελεσμάτων – τερματισμού προηγούμενων φάσεων υπολογισμού
  - Οι παράλληλες εφαρμογές αποτελούνται από εναλλασσόμενους κύκλους υπολογισμού (computation) και ανταλλαγής δεδομένων (communication)
  - Θα πρέπει να υπάρχει συγχρονισμός των threads μεταξύ των κύκλων αυτών

Pthread Mutexes

- Δομή συγχρονισμού για την αποκλειστική πρόσβαση (αμοιβαίο αποκλεισμό) σε κρίσιμες περιοχές κώδικα
  - Κρίσιμη περιοχή: κομμάτι κώδικα που πρέπει να εκτελείται από ένα μοναδικό thread κάθε χρονική στιγμή
  - Όλα τα threads μπορούν να εκτελέσουν τον κώδικα στην κρίσιμη περιοχή, ποτέ όμως ταυτόχρονα
- Η δομή mutex επιτρέπει το «κλείδωμα» κατά την είσοδο στην κρίσιμη περιοχή
  - Το thread που εκτελεί κώδικα στην κρίσιμη περιοχή πρέπει να «ξεκλειδώσει» το mutex κατά την έξοδό του από την περιοχή

Δήλωση ενός mutex

- Δήλωση μεταβλητής mutex
  - Στις global μεταβλητές (για πρόσβαση από όλους)
  - Ως global ορίζονται και οι κοινές μεταβλητές που προστατεύει το mutex

```
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
```

  - `PTHREAD_MUTEX_INITIALIZER` → macro για default αρχικοποίηση
    - Υπάρχει επίσης η δυνατότητα για αρχικοποίηση με ειδικά attributes

Χρήση ενός mutex

- Κρίσιμη περιοχή – lock και unlock
  - Πριν την είσοδο και αντίστοιχα κατά την έξοδο από την κρίσιμη περιοχή
  - Προστασία κοινών πόρων (μοιραζόμενων μεταβλητών)

```
// lock count mutex
pthread_mutex_lock(&count_mutex);

  // ...κώδικας κρίσιμης περιοχής...

// unlock count mutex
pthread_mutex_unlock(&count_mutex);
```

Αποδέσμευση ενός mutex

- Τυπικά στο τέλος του προγράμματος
  - Το mutex δεν θα πρέπει να είναι «κλειδωμένο» κατά την αποδέσμευση

```
// destroy mutex - should be unlocked
pthread_mutex_destroy(&count_mutex);
```

Pthread Condition Variables

- Δίνουν τη δυνατότητα για αναμονή μέχρι να συμβεί κάτι
  - Κάποιο άλλο thread θα πρέπει να δημιουργήσει μια «ειδοποίηση» επανεκκίνησης
  - Μέχρι να έρθει η «ειδοποίηση» το λειτουργικό σύστημα αφαιρεί το thread που περιμένει από την ουρά εκτέλεσης
- Δεν εξασφαλίζουν αμοιβαίο αποκλεισμό
  - Ο χειρισμός των condition variables θα πρέπει να γίνεται σε κρίσιμη περιοχή με προστασία από κάποιο mutex
  - Το λειτουργικό σύστημα φροντίζει την κατάσταση του mutex όταν βάζει το thread σε αναμονή και όταν το ξεκινά πάλι

Δήλωση ενός condition variable

- **Δήλωση μεταβλητής condition variable**
  - Στις global μεταβλητές (για πρόσβαση από όλους)
  - Προσοχή: το condition variable είναι μοιραζόμενος πόρος
    - Πρέπει να προστατεύεται μέσω mutex

```
pthread_cond_t msg_out = PTHREAD_COND_INITIALIZER;
```

  - `PTHREAD_COND_INITIALIZER` → macro για default αρχικοποίηση
    - Υπάρχει επίσης η δυνατότητα για αρχικοποίηση με ειδικά attributes

Χρήση ενός CV – αναμονή

```
// lock mutex
pthread_mutex_lock(&mutex);

while (global_availmsg>0) {

  // mutex must be locked here
  pthread_cond_wait(&msg_out,&mutex);

}

// useful work here, protected by mutex
work();

// unlock mutex
pthread_mutex_unlock(&mutex);
```

προστατεύει τη χρήση των κοινών πόρων

συνθήκη, κοινός πόρος

χρήση while και όχι if: όταν το thread «ξυπνήσει» δεν ξέρουμε την τιμή της συνθήκης

αν το mutex είναι locked το λειτουργικό σύστημα βάζει το thread σε αναμονή και ξεκλειδώνει το mutex

Όταν υπάρξει ειδοποίηση στο CV msg_out, κάποιο thread ξεκινά με το mutex κλειδωμένο

Χρήση ενός CV – ειδοποίηση

- Υπενθύμιση: το condition variable είναι κοινός πόρος
  - Πρέπει να προστατεύεται από κάποιο mutex

```
pthread_cond_signal(&msg_out);
```

Αποδέσμευση CV

- Τυπικά στο τέλος του προγράμματος
  - Δεν πρέπει να υπάρχει thread σε αναμονή για ειδοποίηση από το CV αυτό

```
// destroy CV - no process should be waiting on this
pthread_cond_destroy(&msg_out);
```

Pthread Barrier

- Δομή συγχρονισμού που εξασφαλίζει τη συνέχιση εκτέλεσης μιας ομάδας threads μόλις φτάσουν όλα σε κάποιο σημείο κώδικα
  - Χρήσιμο όταν ένας αλγόριθμος εκτελείται από έναν αριθμό threads σε βήματα και πρέπει να εξασφαλιστεί η ολοκλήρωση του προηγούμενου βήματος από όλους
  - Θα μπορούσε να γίνει με πολλαπλά fork – join αλλά η επιβάρυνση της επαναληπτικής δημιουργίας και διαγραφής threads είναι πολύ μεγάλη

Δήλωση και αρχικοποίηση ενός barrier

- Δήλωση μεταβλητής barrier
  - Ως global μεταβλητή (για πρόσβαση από όλους)

```
pthread_barrier_t barrier;
```

  - Επειδή το barrier πρέπει να αρχικοποιηθεί με τον αριθμό των threads που θα περιμένουν σε αυτό, δεν υπάρχει default αρχικοποίηση!
    - Πρέπει να γίνει μέσα σε συνάρτηση (π.χ. στο main)
- Αρχικοποίηση barrier
  - Με τον αριθμό των threads για αναμονή στο 3ο όρισμα

```
pthread_barrier_init(&barrier,NULL,THREADS);
```

  - Το 2ο όρισμα επιτρέπει πρόσθετα χαρακτηριστικα

Αναμονή σε ένα barrier

- Εκτελείται από τον κώδικα κάθε thread που πρέπει να περιμένει
  - Προσοχή: πρέπει να φτάσουν στο σημείο αυτό όλα τα εμπλεκόμενα threads
    - Αποφυγή αναμονής σε κομμάτια κώδικα υπό συνθήκη

```
pthread_barrier_wait(&barrier);
```

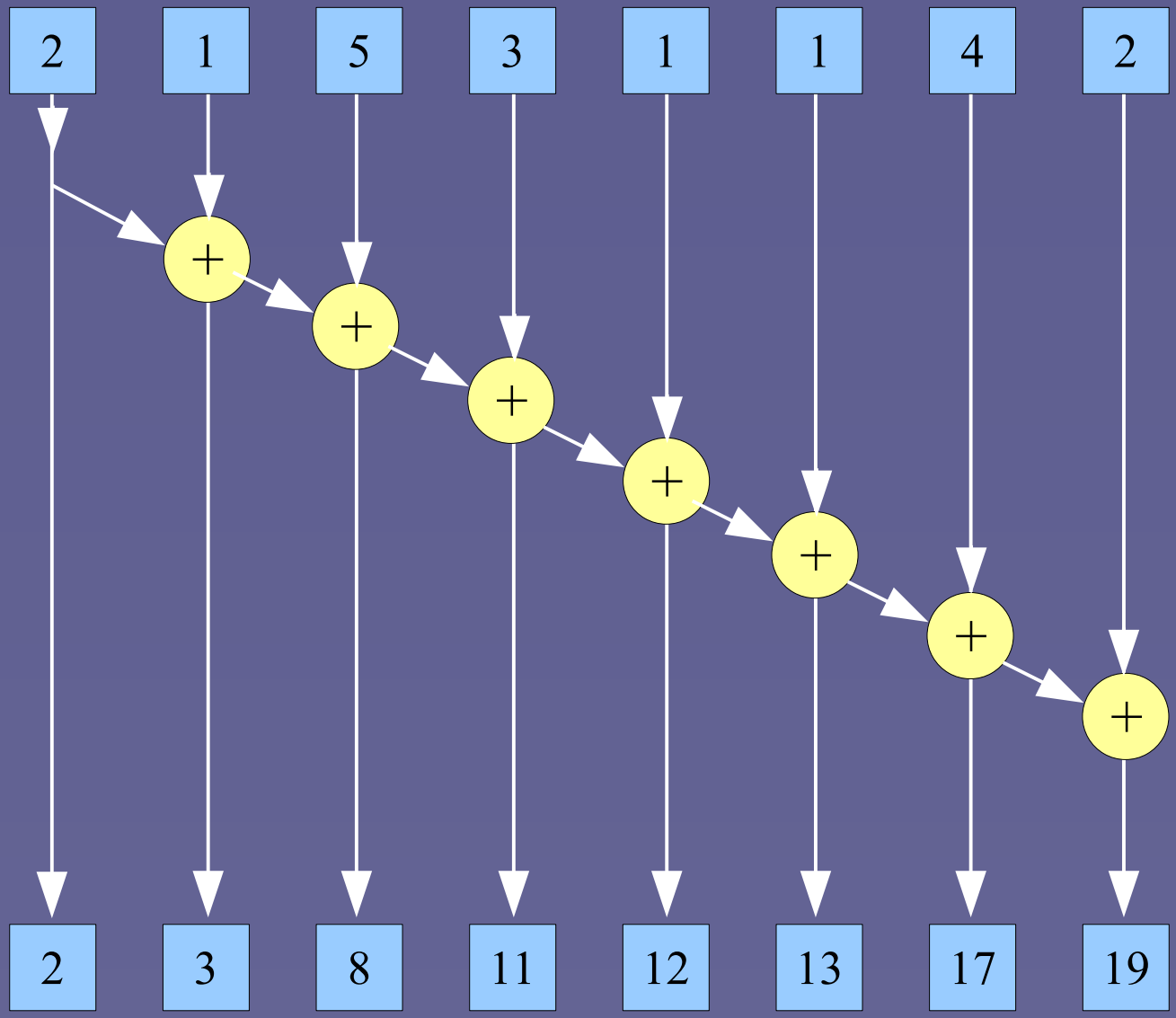
  - Η συνάρτηση δεν επιστρέφει παρά μόνο όταν ο προσδιορισμένος στην αρχικοποίηση αριθμός threads φτάσει στο σημείο αυτό και την καλέσει
  - Αμέσως μετά την επανεκκίνηση των threads το barrier επιστρέφει στην αρχική κατάσταση και μπορεί να χρησιμοποιηθεί ξανά

Αποδέσμευση barrier

- Τυπικά στο τέλος του προγράμματος
  - Δεν πρέπει να υπάρχει thread σε αναμονή στο barrier αυτό

```
// destroy barrier - no thread should be waiting on it
pthread_barrier_destroy(&barrier);
```

Παράδειγμα: Prefix sum (scan)



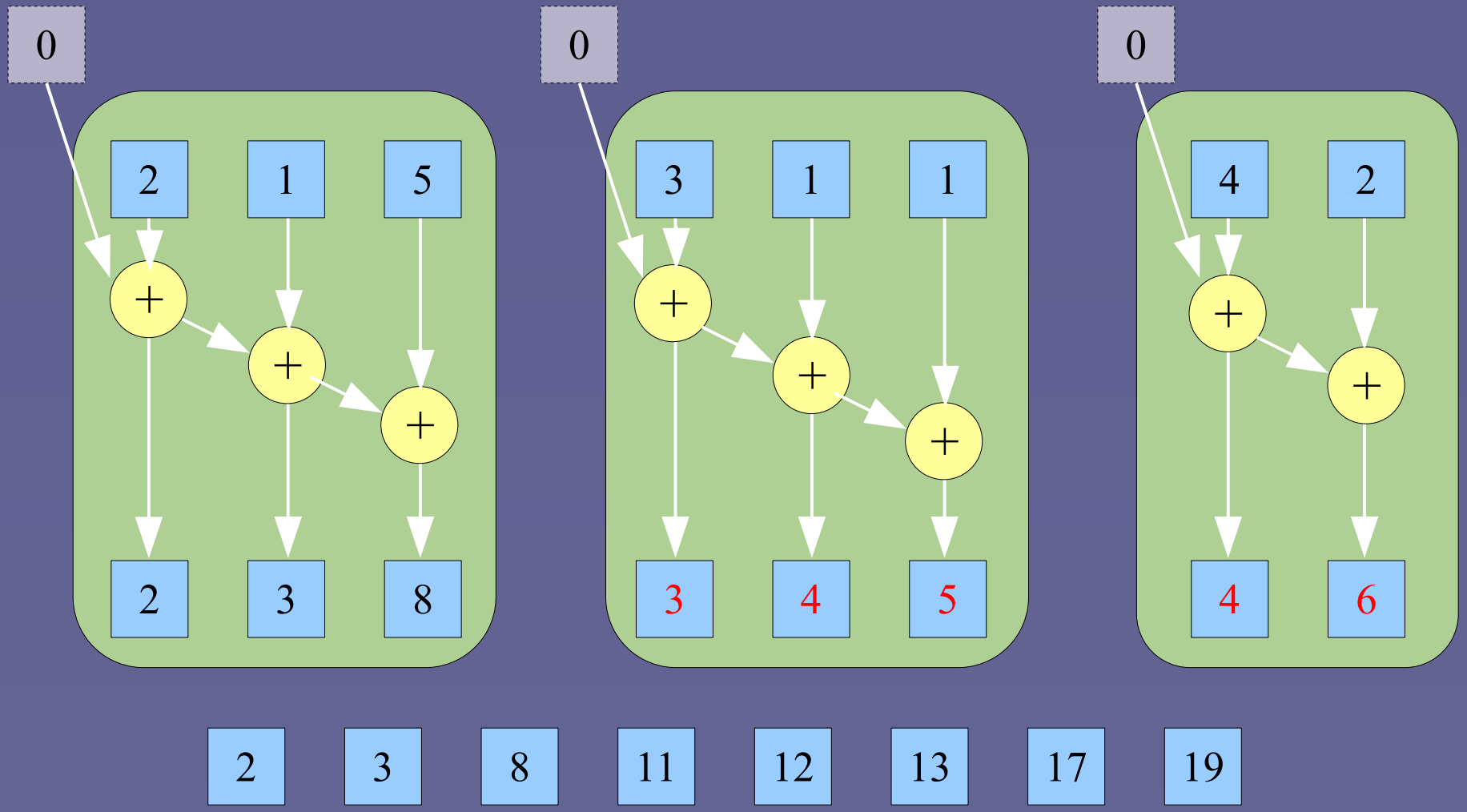
Prefix sum

- Χρησιμοποιείται ως βασικό δομικό στοιχείο στην παράλληλη επεξεργασία ευρύτερων προβλημάτων
- Πώς μπορεί να παραλληλοποιηθεί με threads;
  - Χωρίς την υπέρμετρη αύξηση του εκτελούμενου έργου (work)
  - Λαμβάνοντας υπόψη
    - Τον αριθμό των διαθέσιμων hardware threads
    - Το κόστος επικοινωνίας – συγχρονισμού για τη μεταβίβαση των μερικών αποτελεσμάτων
    - Την εκμετάλλευση της τοπικότητας των κρυφών μνημών
    - Την πιθανή επιβάρυνση των software threads (λειτουργικό σύστημα)

Prefix sum στην πράξη

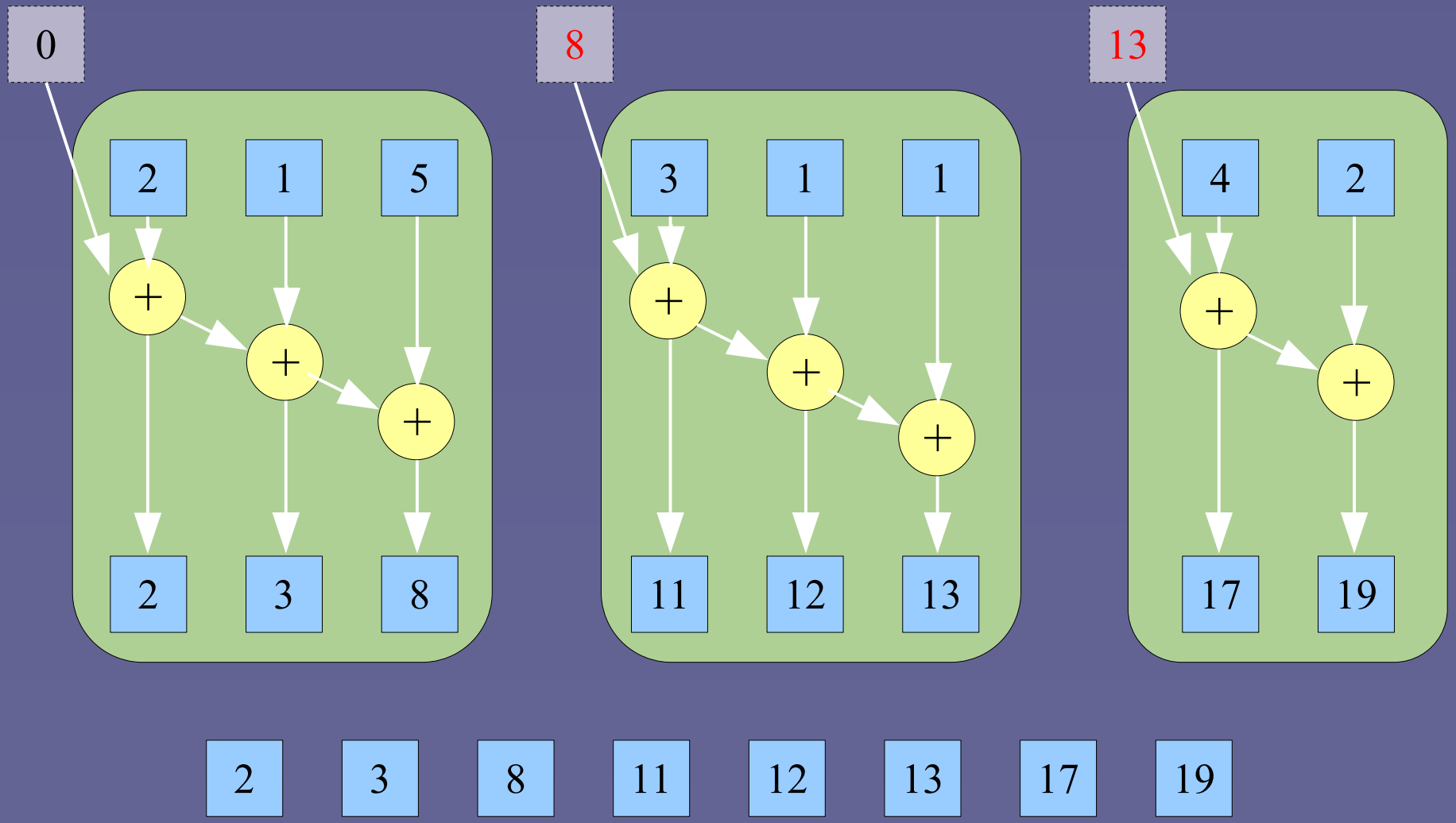
- Πρακτικές υλοποιήσεις
  - Παραλλαγές των θεωρητικών λύσεων που λαμβάνουν επίσης υπόψη
    - Τον αριθμό των διαθέσιμων hardware threads
    - Το κόστος επικοινωνίας για τη μεταβίβαση των μερικών αποτελεσμάτων
    - Την εκμετάλλευση της τοπικότητας των κρυφών μνημών
    - Την πιθανή επιβάρυνση των software threads (λειτουργικό σύστημα)
- Υλοποίηση prefix sum με blocks
  - CPU – shared memory

Μπορεί να εκτελεστεί με threads όπως το άθροισμα;

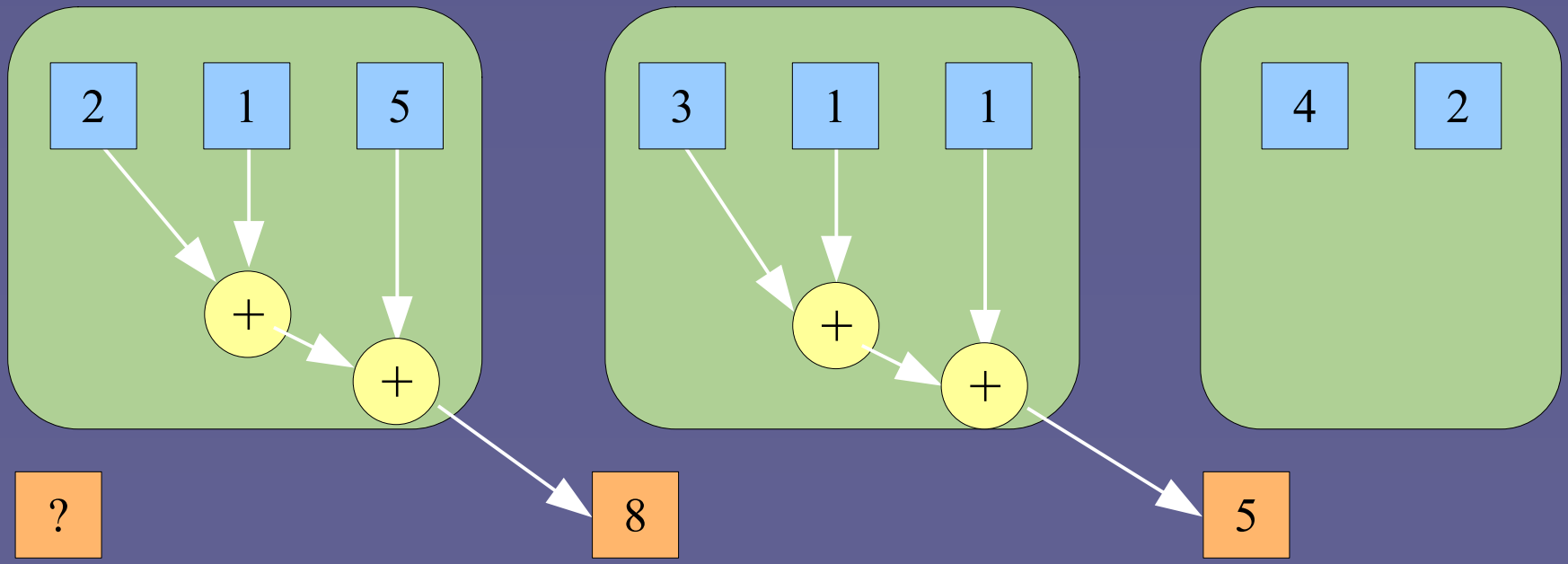


Το αποτέλεσμα είναι λάθος! (τα μερικά αθροίσματα δεν μεταδίδονται από μπλοκ σε μπλοκ)

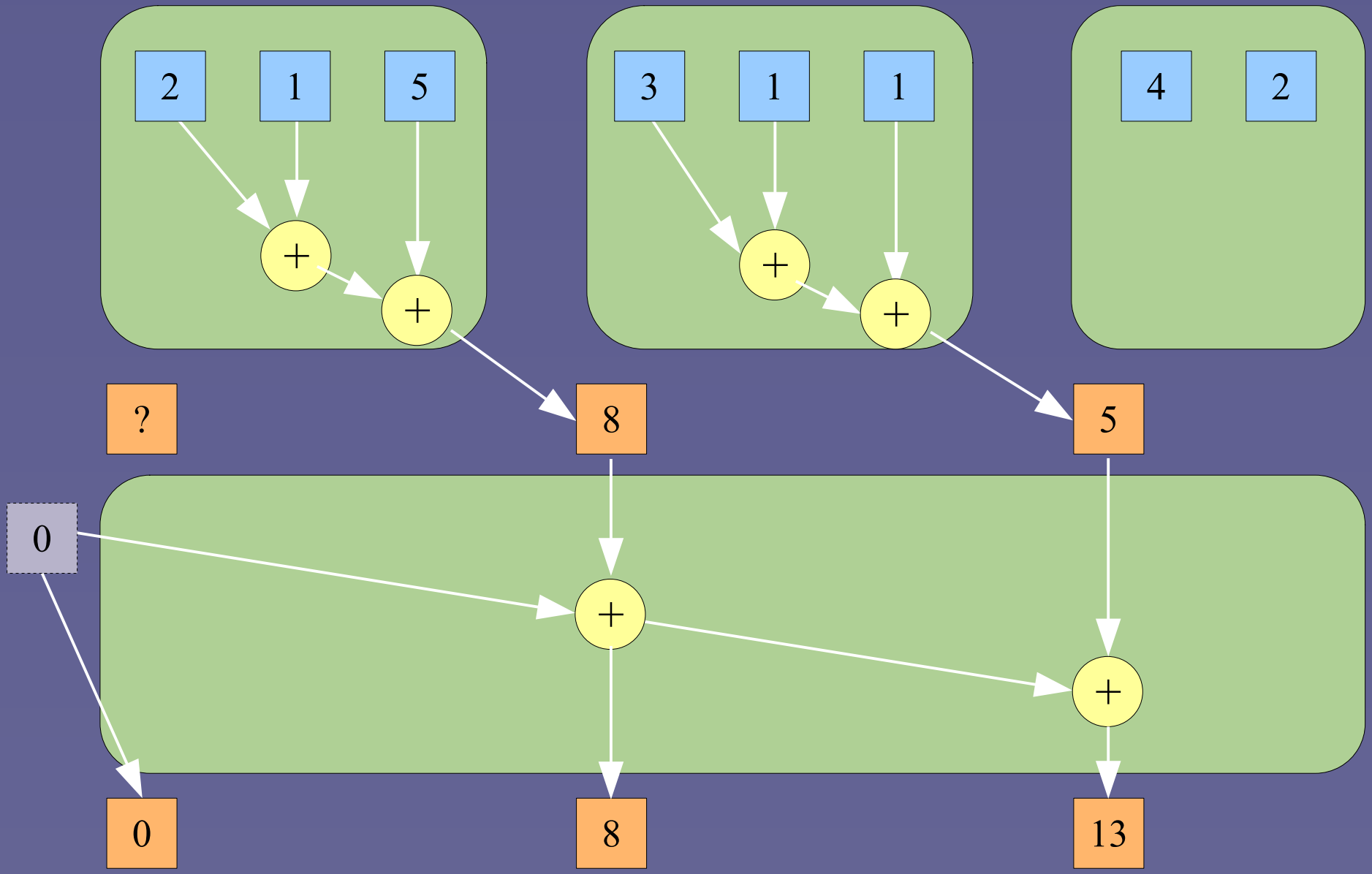
Αν υπάρχει ανατροφοδότηση μερικών αθροισμάτων;



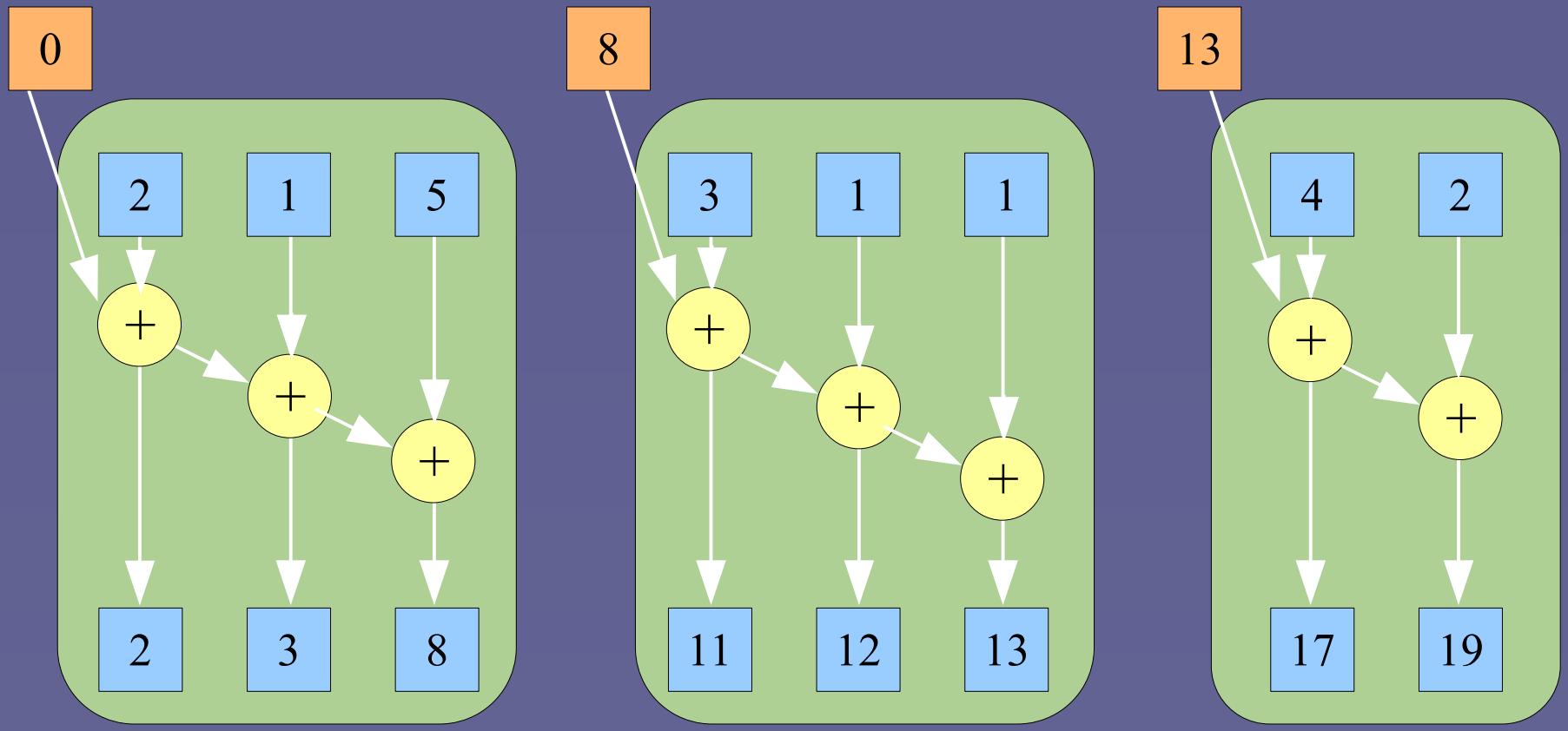
Το αποτέλεσμα θα είναι σωστό



Α΄ φάση (όλα τα threads, εκτός από το τελευταίο)



Β΄ φάση (ένα μόνο thread)



Γ΄ φάση (όλα τα threads)