



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

3^η Εργαστηριακή Άσκηση:

Συγχρονισμός

Λειτουργικά Συστήματα Υπολογιστών
7ο Εξάμηνο, 2015-2016

Σύνοψη



- ◆ Τρία προβλήματα συγχρονισμού
- ◆ Χρήση νημάτων: Υλοποιήσεις με POSIX Threads
- ◆ Μηχανισμοί συγχρονισμού:
 - POSIX Mutexes
 - POSIX Semaphores
 - POSIX Condition Variables
 - GCC atomic operations

Σύνοψη



- ◆ Z1: Συγχρονισμός σε υπάρχοντα κώδικα (κρίσιμο τμήμα)
 - ➔ `simplesync.c`
 - ➔ Με POSIX mutexes και GCC atomic ops
- ◆ Z2: Παραλληλοποίηση υπάρχοντα κώδικα (ανάγκη σειριοποίησης)
 - ➔ Συγχρονισμός νημάτων για παράλληλο υπολογισμό
- ◆ Z3: Επίλυση προβλήματος συγχρονισμού
 - ➔ Με δεδομένους περιορισμούς για τα νήματα

Σύνοψη



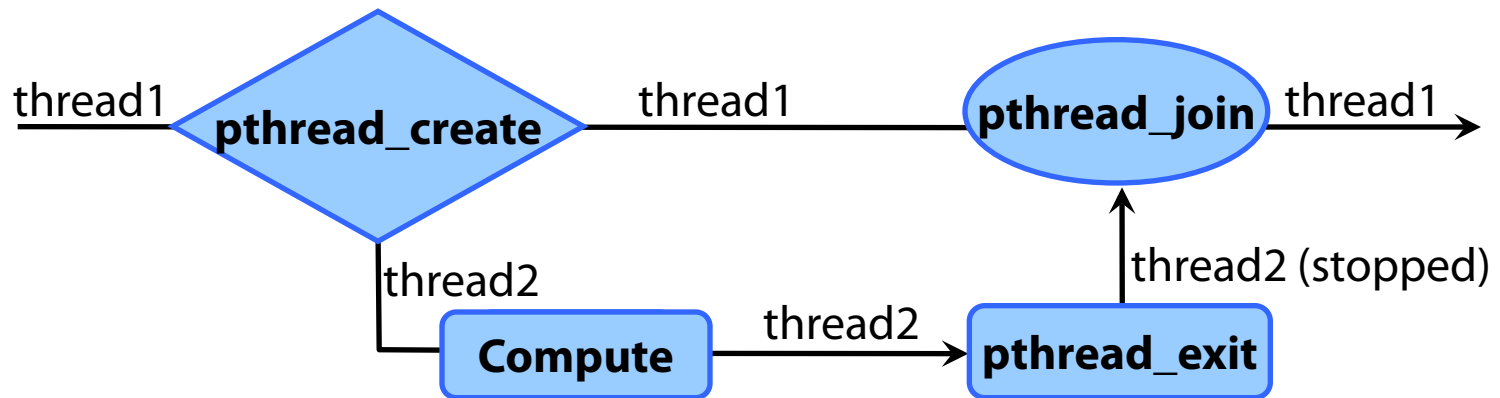
- ◆ Τρία προβλήματα συγχρονισμού
- ◆ Χρήση νημάτων: Υλοποιήσεις με POSIX Threads
- ◆ Μηχανισμοί συγχρονισμού:
 - POSIX Mutexes
 - POSIX Semaphores
 - POSIX Condition Variables
 - GCC atomic operations

Δημιουργία νημάτων στα POSIX Threads

◆ Δημιουργία με **pthread_create()**

- ➔ `int pthread_create(pthread_t * thread, pthread_attr_t * attr, void * (*start_routine)(void *), void * arg);`
- ➔ π.χ. `pthread_create(&tid, &attr, thread_fn, arg)`

◆ Αναμονή για τερματισμό (**pthread_exit()**) με **pthread_join()**



Σύνοψη



- ◆ Τρία προβλήματα συγχρονισμού
- ◆ Χρήση νημάτων: Υλοποιήσεις με POSIX Threads
- ◆ Μηχανισμοί συγχρονισμού:
 - POSIX Mutexes
 - POSIX Semaphores
 - POSIX Condition Variables
 - GCC atomic operations

Μηχανισμοί (POSIX)

- ◆ POSIX Threads <pthread.h>
 - ➔ pthread_create(), pthread_join()
- ◆ POSIX Mutexes <pthread.h>
 - ➔ pthread_mutex_init()
- ◆ POSIX (unnamed) Semaphores <semaphore.h>
 - ➔ Manpages: sem_overview(7), sem_init(3), sem_post(3), sem_wait(3).
- ◆ POSIX condition variables:
 - ➔ Pthread_cond_init(), pthread_cond_wait(), pthread_cond_signal(), pthread_cond_broadcast()
- ◆ **Εγκαταστήστε** τα πακέτα manpages-posix, manpages-posix-dev: **man – a sem_post**

Μηχανισμοί (GCC atomic operations)

- ◆ GCC atomic operations
 - ➔ <http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Atomic-Builtins.html>
- ◆ Ειδικές εντολές (builtins) / συναρτήσεις για **ατομική πρόσβαση**
- ◆ **`__sync_add_and_fetch()`, `__sync_sub_and_fetch()`, ...**

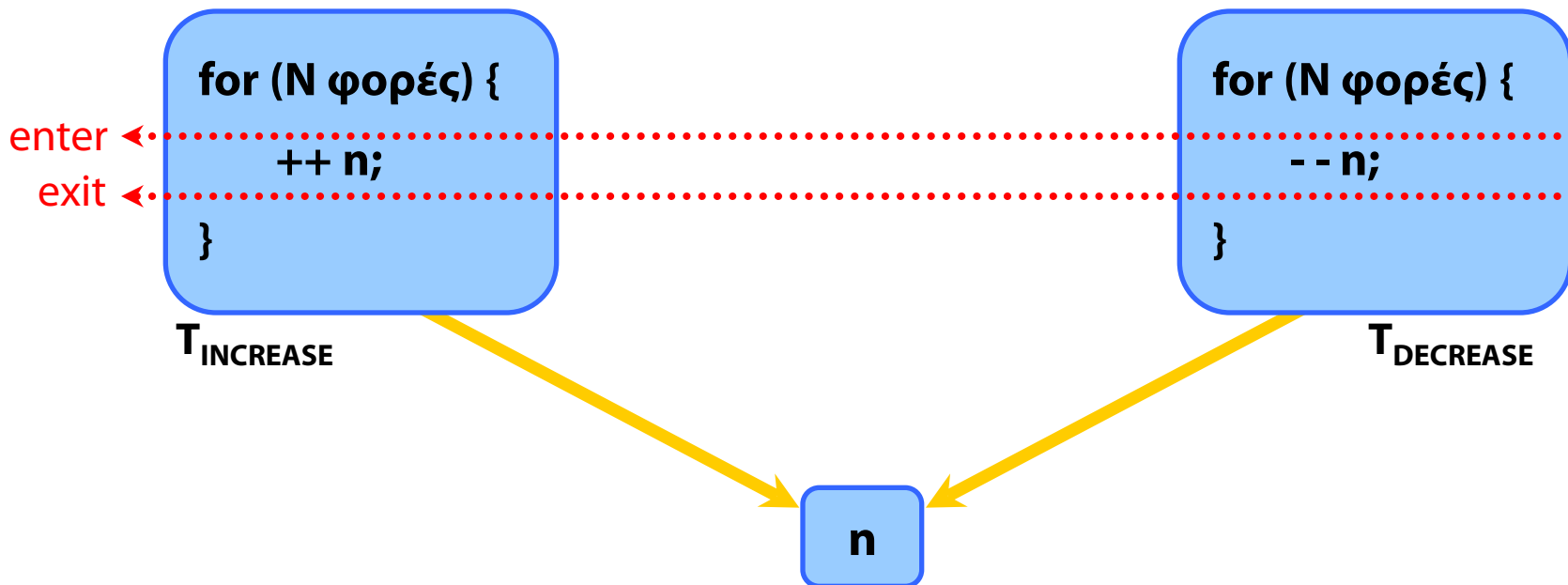
Σύνοψη



- ◆ Z1: Συγχρονισμός σε υπάρχοντα κώδικα (κρίσιμο τμήμα)
 - ➔ `simplesync.c`
 - ➔ Με POSIX mutexes και GCC atomic ops
- ◆ Z2: Παραλληλοποίηση υπάρχοντα κώδικα (ανάγκη σειριοποίησης)
 - ➔ Συγχρονισμός νημάτων για παράλληλο υπολογισμό
- ◆ Z3: Επίλυση προβλήματος συγχρονισμού
 - ➔ Με δεδομένους περιορισμούς για τα νήματα

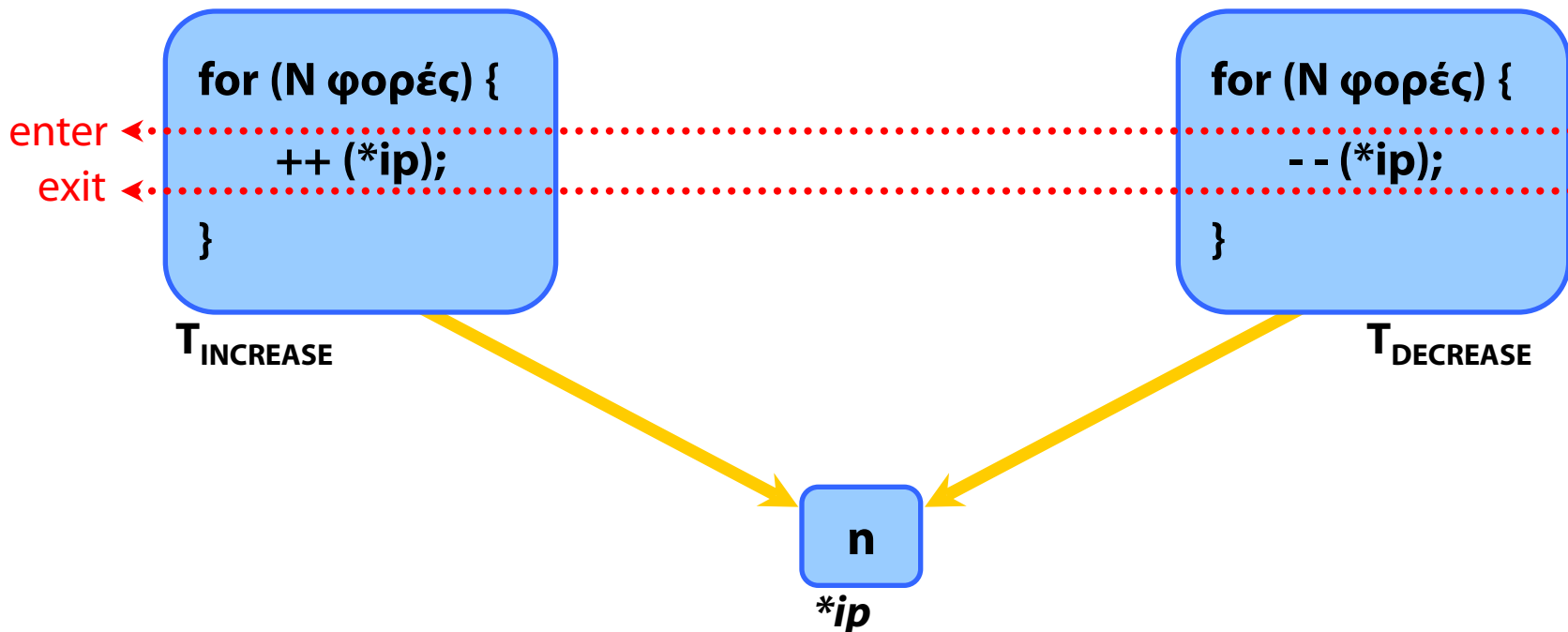
Z1: Συγχρονισμός σε υπάρχοντα κώδικα

- ◆ Δύο νήματα: T_{INCREASE} , T_{DECREASE}
- ◆ Αυξάνουν/μειώνουν το **κοινό** n , N φορές, αντίστοιχα
- ◆ Αρχική τιμή $n = 0$. Σχήμα συγχρονισμού ώστε
Το n να παραμείνει **0** μετά το τέλος της εκτέλεσής τους.



Z1: Συγχρονισμός στο **simplesync.c**

- ◆ **Δύο** υλοποιήσεις
- ◆ **Z1α.** POSIX mutexes
- ◆ **Z1β.** GCC atomic operations: `__sync_*`()



Z1: Συγχρονισμός σε υπάρχοντα κώδικα

◆ Z1α. POSIX mutexes/semaphores

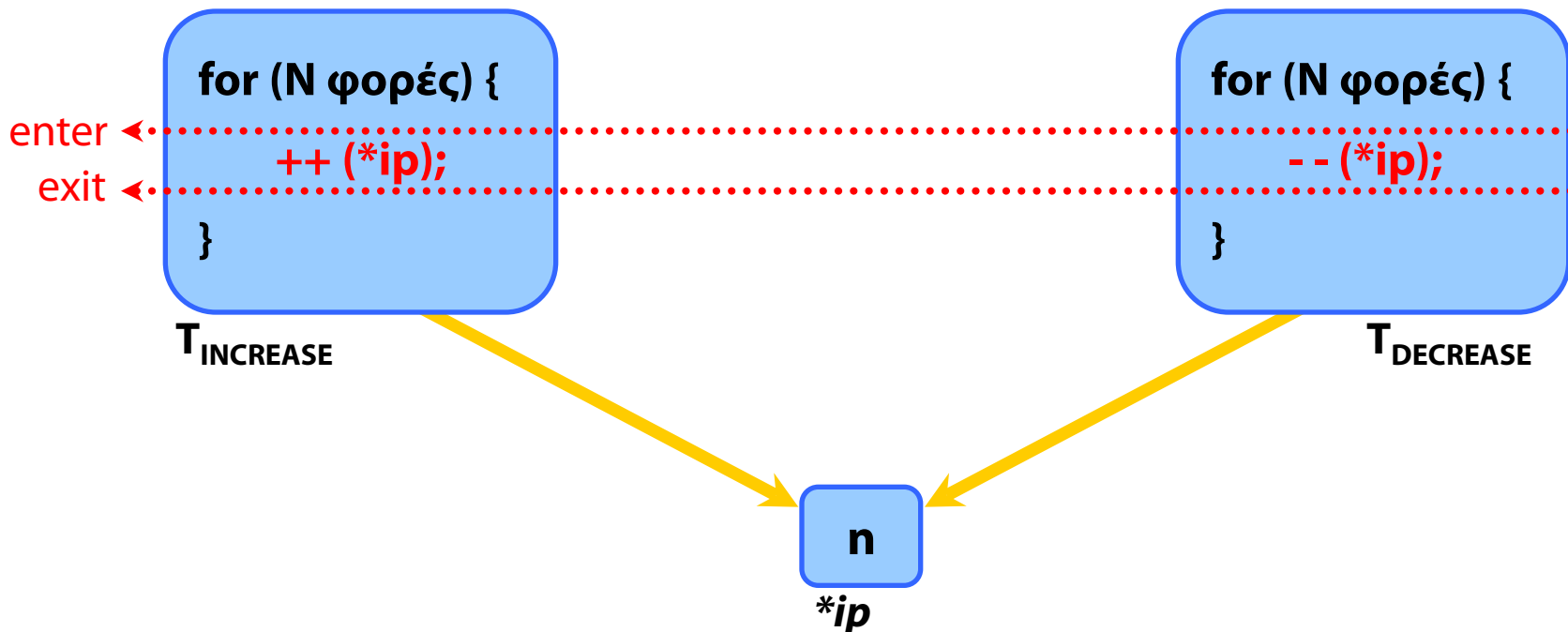
- ➔ Κώδικας **μόνο** στα σημεία “enter”, “exit”
- ➔ Κατάλληλα αρχικοποιημένα mutexes ή σηματοφόροι
- ➔ wait(), signal() σε αυτούς
- ➔ Χωρίς αλλαγή του κώδικα που πειράζει τη μεταβλητή

◆ Z1β. GCC atomic operations

- ➔ **Αλλαγή** του τρόπου πρόσβασης στη μεταβλητή
- ➔ Απαιτείται πλέον κώδικας στα “enter”, “exit”;

Z1: Συγχρονισμός στο **simplesync.c**

- ◆ **Δύο** υλοποιήσεις
- ◆ **Z1α.** POSIX mutexes
- ◆ **Z1β.** GCC atomic operations: `__sync_*`()

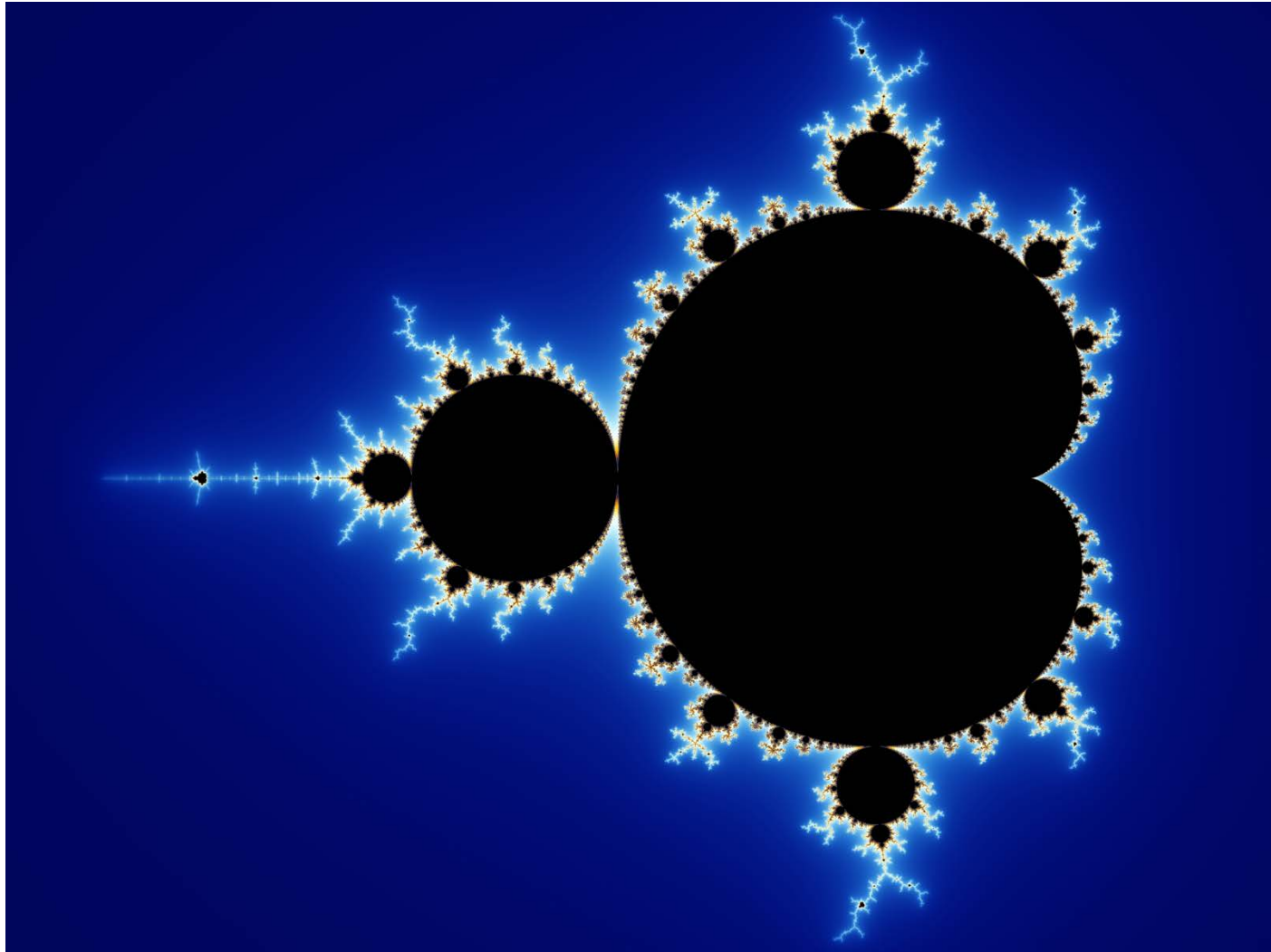


Σύνοψη



- ◆ Z1: Συγχρονισμός σε υπάρχοντα κώδικα (κρίσιμο τμήμα)
 - ➔ `simplesync.c`
 - ➔ Με POSIX mutexes και GCC atomic ops
- ◆ Z2: Παραλληλοποίηση υπάρχοντα κώδικα (ανάγκη σειριοποίησης)
 - ➔ Συγχρονισμός νημάτων για παράλληλο υπολογισμό
- ◆ Z3: Επίλυση προβλήματος συγχρονισμού
 - ➔ Με δεδομένους περιορισμούς για τα νήματα

Z2: Παραλληλοποίηση: the Mandelbrot Set

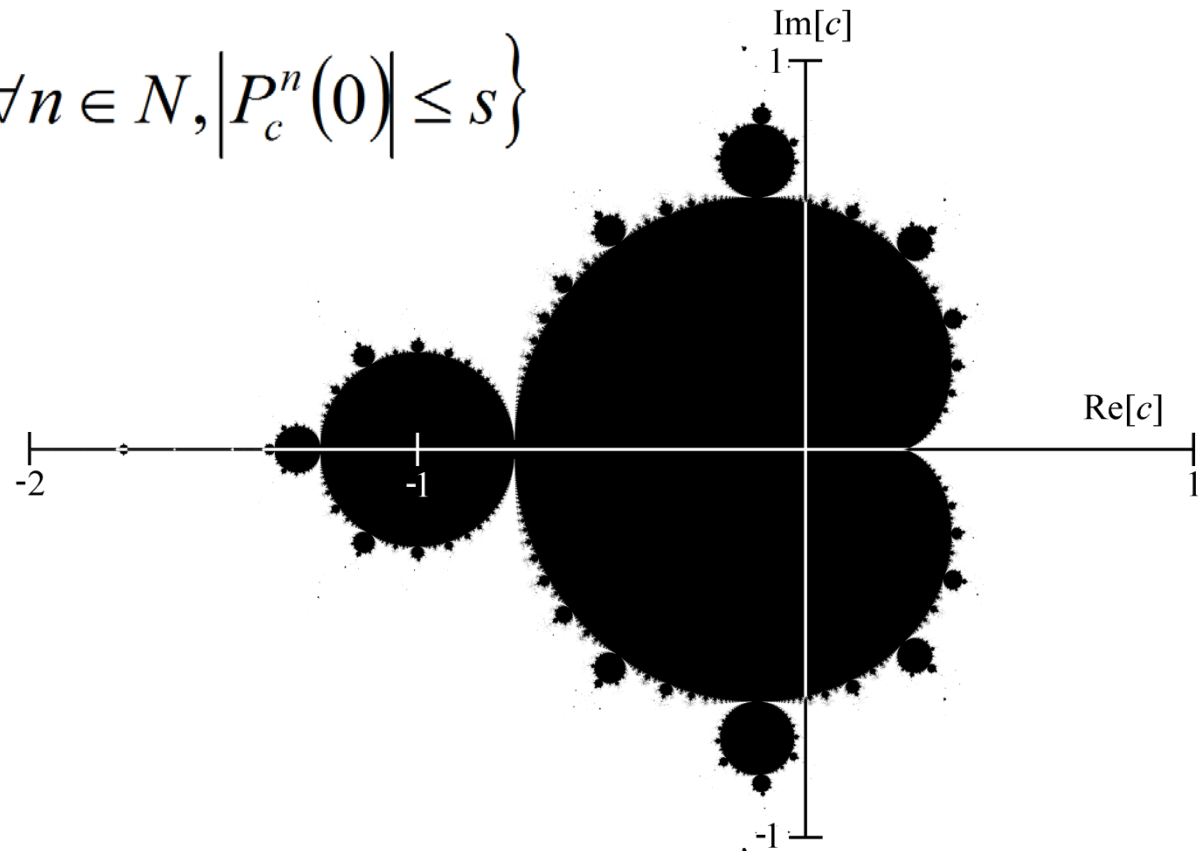


The Mandelbrot Set: Ορισμός

$$P_c : C \rightarrow C$$

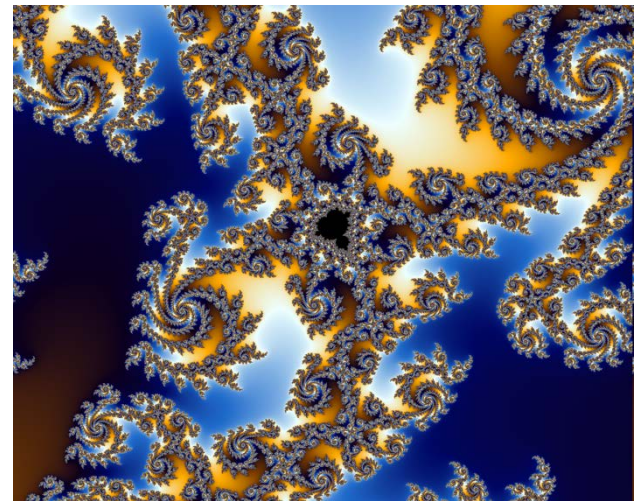
$$P_c : z \rightarrow z^2 + c$$

$$M = \left\{ c \in C : \exists s \in R, \forall n \in N, |P_c^n(0)| \leq s \right\}$$



The Mandelbrot Set: σχεδίαση

- ◆ Για κάθε σημείο c μιας περιοχής του μιγαδικού επιπέδου
 - ➔ Επαναληπτικός υπολογισμός του $\mathbf{z}_{n+1} = \mathbf{z}_n^2 + \mathbf{c}$, $\mathbf{z}_0 = \mathbf{0}$, μέχρι να ξεφύγει το $|\mathbf{z}_n|$
 - ➔ Κάθε pixel χρωματίζεται ανάλογα με τον αριθμό των επαναλήψεων που χρειάστηκαν, ή \mathbf{n}_{\max}
- ◆ Υπάρχουν κι άλλοι αλγόριθμοι



The Mandelbrot Set: κώδικας



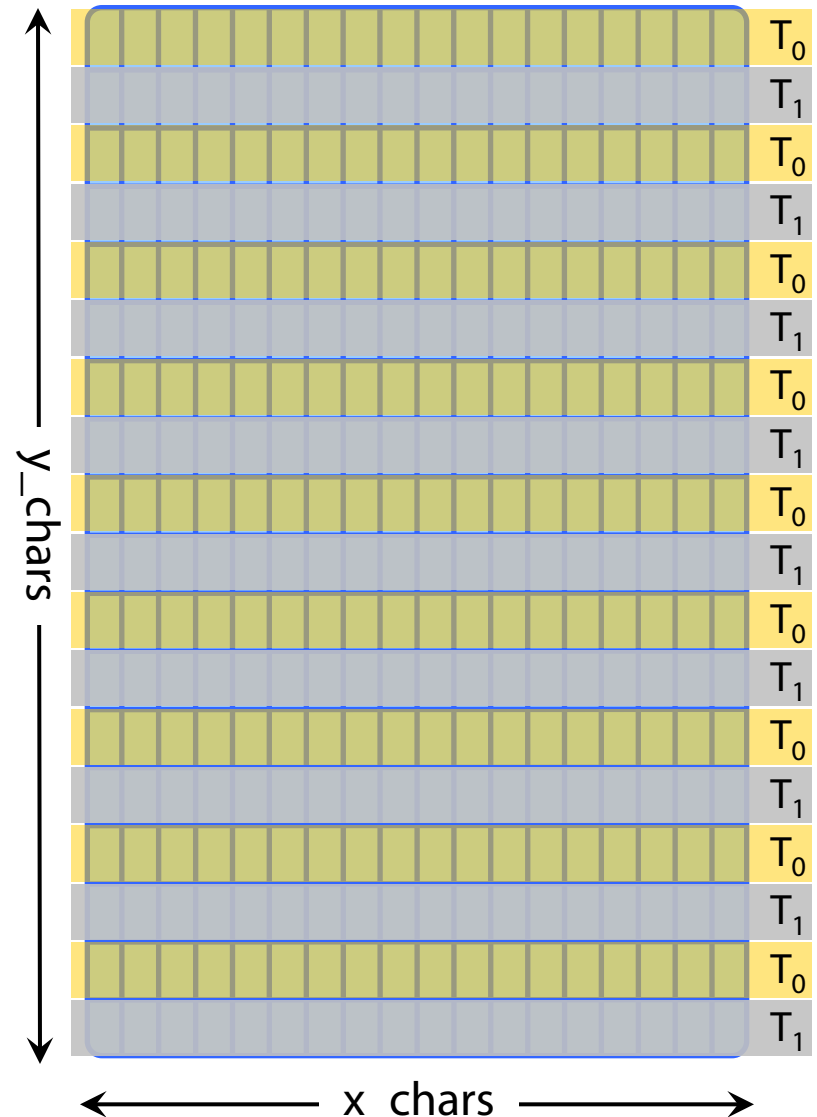
- ◆ Σας δίνεται κώδικας (mandel.c) που ζωγραφίζει εικόνες από το σύνολο Mandelbrot
 - ➔ Στο τερματικό, με χρωματιστούς χαρακτήρες
 - ➔ Κάθε εικόνα είναι πλάτους **x_chars**, ύψους **y_chars**
- ◆ Η σχεδίαση γίνεται επαναληπτικά, για κάθε γραμμή
- ◆ Συναρτήσεις
 - ➔ `compute_and_output_mandel_line(fd, line)`
 - ➔ `mandel_iterations_at_point(x, y, MAX)`
 - ➔ `set_xterm_color(fd, color)`

The Mandelbrot Set: Παραλληλοποίηση

- ◆ Κατανομή του φορτίου ανά γραμμές
- ◆ Ξεκινώντας από το πρώτο νήμα, ανάθεση γραμμών με κυκλική επαναφορά
- ◆ Νήμα i από N :

$i, i + N, i + 2 * N, i + 3 * N$ κλπ

Συγχρονισμός;



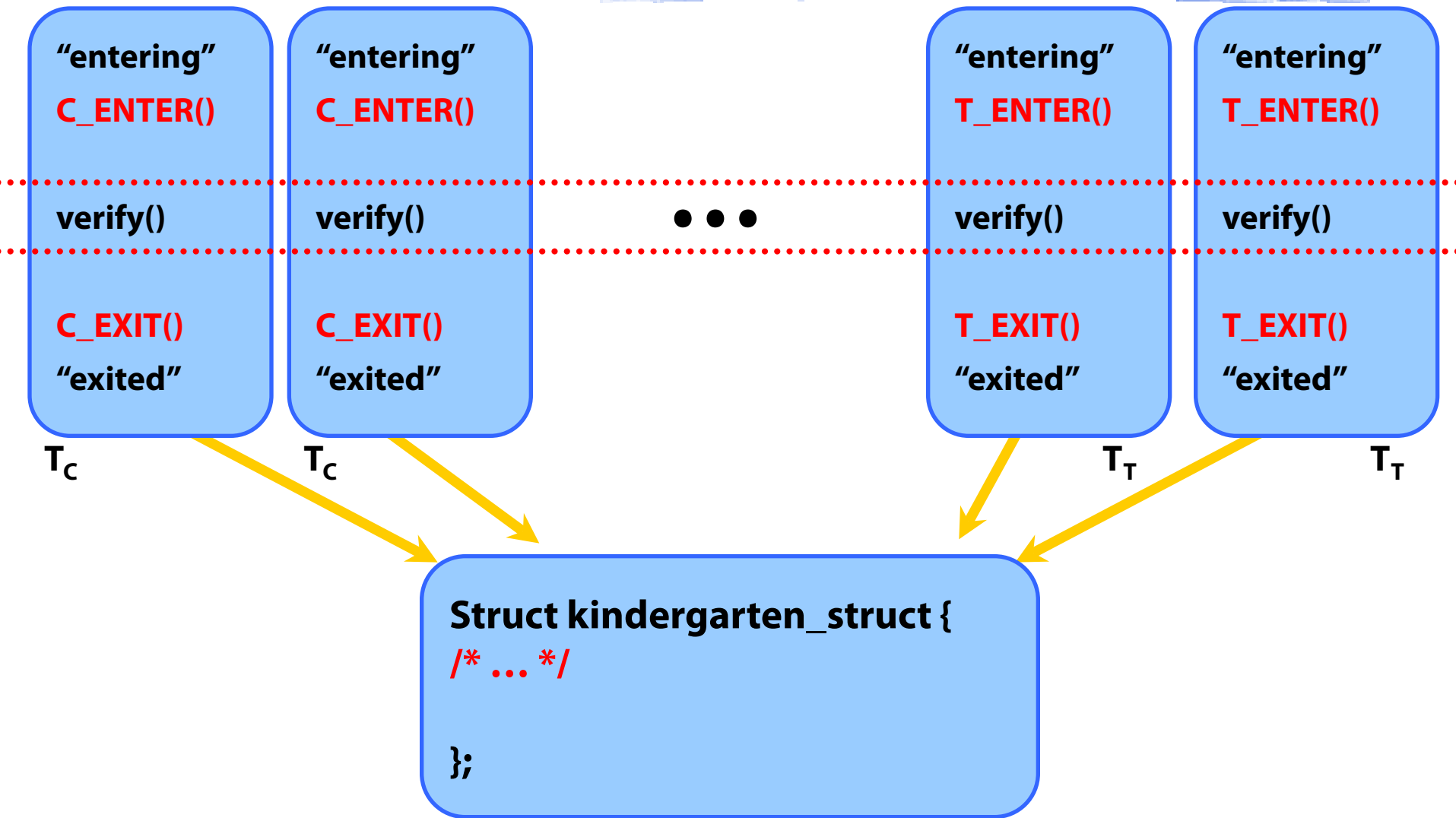
Σύνοψη

- ◆ Z1: Συγχρονισμός σε υπάρχοντα κώδικα (κρίσιμο τμήμα)
 - ➔ `simplesync.c`
 - ➔ Με POSIX mutexes και GCC atomic ops
- ◆ Z2: Παραλληλοποίηση υπάρχοντα κώδικα (ανάγκη σειριοποίησης)
 - ➔ Συγχρονισμός νημάτων για παράλληλο υπολογισμό
- ◆ Z3: Επίλυση προβλήματος συγχρονισμού
 - ➔ Με δεδομένους περιορισμούς για τα νήματα

Z3: Επίλυση προβλήματος συγχρονισμού

- ◆ Ένα **νηπιαγωγείο (Kindergarten)**
- ◆ **Δάσκαλοι** και **παιδιά**.
- ◆ Καθορισμένη μέγιστη αναλογία παιδιών ανά δάσκαλο:
R παιδιά ανά δάσκαλο, π.χ. 3:1.
- ◆ Δεδομένη υλοποίηση
- ◆ **N** νήματα: **C** νήματα προσομοιώνουν παιδιά, τα υπόλοιπα **N - C** δασκάλους.
- ◆ Σας δίνεται κώδικας, που αποτυγχάνει.

Z3: Επίλυση προβλήματος συγχρονισμού



Z3: Επίλυση προβλήματος συγχρονισμού

◆ Συνθήκες αλλαγής κατάστασης:

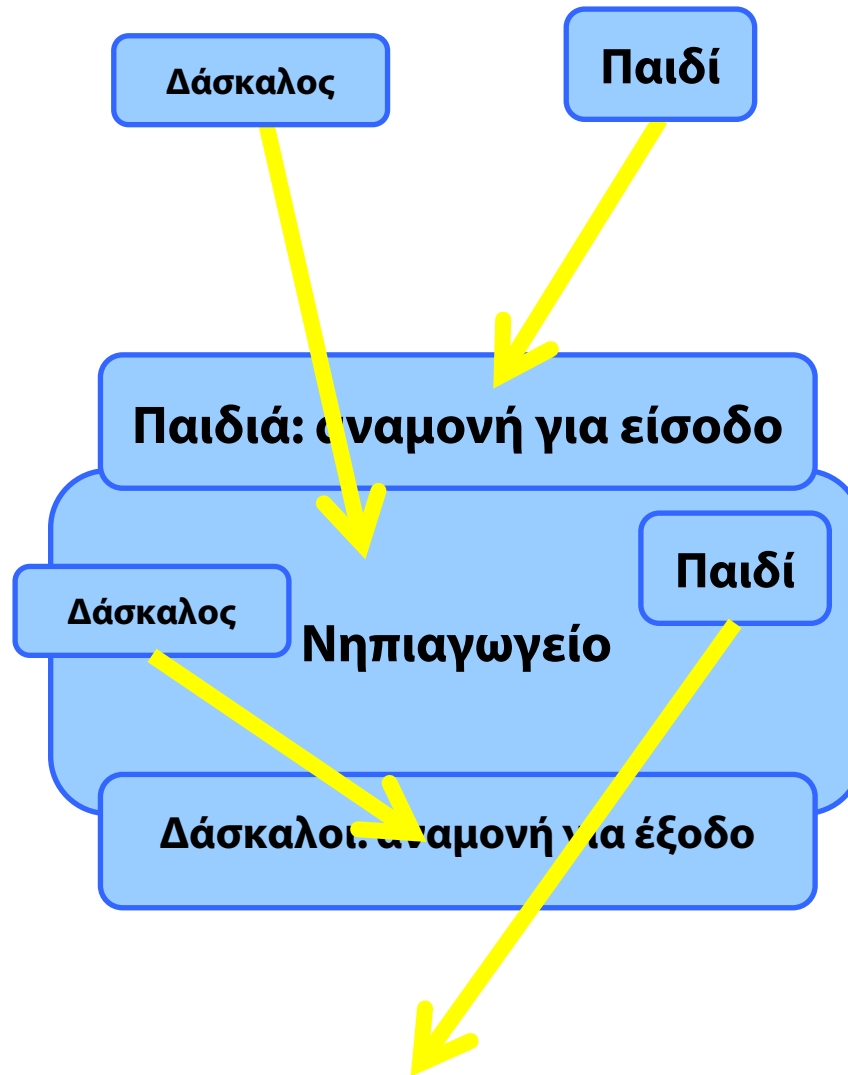
➔ Παιδί:

- Μπαίνει -> υπάρχουν τουλάχιστον $(C+1)/R$ δάσκαλοι για να με υποστηρίξουν;
- Βγαίνει -> άνευ όρων (ενημερώνει αν θέλει κάποιος δάσκαλος να βγει αν $(N - C - 1) * R \geq C$)

➔ Δάσκαλος:

- Μπαίνει -> αν περιμένουν παιδιά, μπορούν να μπουν μέχρι R
- Βγαίνει -> υπάρχουν αρκετοί δάσκαλοι για να υποστηρίξουν τα παιδιά; $(N - C - 1) * R \geq C$.

Z3: Επίλυση προβλήματος συγχρονισμού



Z3: Επίλυση προβλήματος συγχρονισμού (condition variables – 2 νήματα)

```
pthread_mutex_t Lock;  
pthread_cond_t cond;  
int counter = 0;
```

```
/* Thread A */  
pthread_mutex_lock(&Lock);  
while (counter < 10)  
    pthread_cond_wait(&cond, &Lock);  
...  
pthread_mutex_unlock(&Lock);
```

```
/* Thread B */  
pthread_mutex_lock(&Lock);  
counter++;  
If (counter >= 10)  
    pthread_cond_signal(&cond);  
pthread_mutex_unlock(&Lock);
```

Z3: Επίλυση προβλήματος συγχρονισμού (condition variables – N νήματα)

```
pthread_mutex_t Lock;  
pthread_cond_t cond;  
int counter = 0;
```

```
/* Thread X */  
pthread_mutex_lock(&Lock);  
while (counter < 10)  
    pthread_cond_wait(&cond, &Lock);  
...  
pthread_mutex_unlock(&Lock);
```

```
/* Thread Y */  
pthread_mutex_lock(&Lock);  
counter++;  
If (counter >= 10)  
    pthread_cond_broadcast(&cond);  
pthread_mutex_unlock(&Lock);
```

Ερωτήσεις;



και στη λίστα:

OS@lists.cslab.ece.ntua.gr