ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

Μάθημα #10

CUDA (Atomic Instructions)

Σκοπός

- Να καταλάβετε την έννοια των ατομικών εντολών
  - Ανάγνωση-Αλλαγή-Εγγραφή σε παράλληλους υπολογισμούς
  - Χρήση ατομικών εντολών στην CUDA
  - Γιατί οι ατομικές εντολές μειώνουν την επίδοση του συστήματος μνήμης
  - Περιπτώσεις όπου μπορείτε να αποφύγετε την χρήση ατομικών εντολών σε παράλληλους αλγόριθμους
- Υπολογισμός ιστογράμματος ως παράδειγμα χρήσης ατομικών εντολών
  - Ο βασικός αλγόριθμος υπολογισμού ιστογράμματος
  - Ιδιωτικοποίηση μεταβλητών (Privatization)

Ένα συχνά εμφανιζόμενο μοτίβο συνεργασίας

- Πολλοί ταμίες μιας τράπεζας πρέπει να καταμετρήσουν το σύνολο των μετρητών σε ένα χρηματοκιβώτιο
  - Καθένας παίρνει ένα σύνολο δεσμίδων και κάνει καταμέτρηση
  - Πρέπει να υπάρχει μια κεντρική οθόνη όπου απεικονίζεται το τρέχων σύνολο
  - Όταν κάποιος τελειώσει με την καταμέτρηση των δεσμίδων του προσθέτει το ποσό στο τρέχων σύνολο
- Πρόβλημα που μπορεί να προκύψει
  - Μερικές δεσμίδες τελικά δεν καταμετρήθηκαν

Ένα συχνά εμφανιζόμενο μοτίβο συντονισμού

- Πολλοί υπάλληλοι εξυπηρετούν πελάτες
  - Κάθε πελάτης παίρνει έναν αριθμό εξυπηρέτησης
  - Μια κεντρική οθόνη δείχνει τον αριθμό του επόμενου πελάτη που θα εξυπηρετηθεί
- Όταν ένας υπάλληλος ελευθερωθεί, καλεί τον επόμενο αριθμό προσθέτοντας 1 στον αριθμό της οθόνης
- Προβλήματα που μπορεί να προκύψουν
  - Πολλοί πελάτες μπορεί να πάρουν τον ίδιο αριθμό εξυπηρέτησης
  - Πολλοί υπάλληλοι μπορεί να καλέσουν τον ίδιο αριθμό προς εξυπηρέτηση

Ένα συχνά εμφανιζόμενο μοτίβα διαιτησίας

- Πολλοί πελάτες κλείνουν αεροπορικά εισιτήρια
- Ο καθένας τους:
  - Εμφανίζει μια απεικόνιση των θέσεων του αεροπλάνου
  - Αποφασίζει ποια θέση θέλει
  - Ενημερώνει την απεικόνιση των θέσεων με την κατειλημμένη θέση
- Πρόβλημα που μπορεί να προκύψει
  - Πολλοί επιβάτες καταλήγουν να έχουν την ίδια θέση στο αεροπλάνο

Ατομικές εντολές

Thread 1:	Old ← Mem[x]		Thread 2:	Old ← Mem[x]
	New ← Old + 1			New ← Old + 1
	Mem[x] ← New			Mem[x] ← New

- Αν η θέση μνήμης Mem[x] έχει αρχική τιμή 0, ποια θα είναι η τιμή της Mem[x] όταν ολοκληρώσουν την εκτέλεση τους τα νήματα 1 και 2;
  - Τι θα περιέχει κάθε νήμα στην δική του μεταβλητή Old;
- Η απάντηση μπορεί να διαφέρει λόγω data race. Για την αποφυγή τους θα πρέπει να χρησιμοποιηθούν ατομικές εντολές
  - Όταν αυτό είναι εφικτό
  - Διαφορετικά πρέπει να χρησιμοποιηθεί άλλος μηχανισμός (lock, mutex, κλπ).

Πρώτο σενάριο εκτέλεσης

Χρόνος	Νήμα 1	Νήμα 2
1	(0) Old ← Mem[x]	
2	(1) New ← Old + 1	
3	(1) Mem[x] ← New	
4		(1) Old ← Mem[x]
5		(2) New ← Old + 1
6		(2) Mem[x] ← New

- Νήμα 1: Old = 0
- Νήμα 2: Old = 1
- Mem[x] = 2

Δεύτερο σενάριο εκτέλεσης

Χρόνος	Νήμα 1	Νήμα 2
1		(0) Old ← Mem[x]
2		(1) New ← Old + 1
3		(1) Mem[x] ← New
4	(1) Old ← Mem[x]	
5	(2) New ← Old + 1	
6	(2) Mem[x] ← New	

- Νήμα 1: Old = 1
- Νήμα 2: Old = 0
- Mem[x] = 2

Τρίτο σενάριο εκτέλεσης

Χρόνος	Νήμα 1	Νήμα 2
1	(0) Old ← Mem[x]	
2	(1) New ← Old + 1	
3		(0) Old ← Mem[x]
4	(1) Mem[x] ← New	
5		(1) New ← Old + 1
6		(1) Mem[x] ← New

- Νήμα 1: Old = 0
- Νήμα 2: Old = 0
- Mem[x] = 1

Τέταρτο σενάριο εκτέλεσης

Χρόνος	Νήμα 1	Νήμα 2
1		(0) Old ← Mem[x]
2		(1) New ← Old + 1
3	(0) Old ← Mem[x]	
4		(1) Mem[x] ← New
5	(1) New ← Old + 1	
6	(1) Mem[x] ← New	

- Νήμα 1: Old = 0
- Νήμα 2: Old = 0
- Mem[x] = 1

Ατομικές εντολές για την διασφάλιση σωστών αποτελεσμάτων

Old ← Mem[x]	
New ← Old + 1	
Mem[x] ← New	
	Old ← Mem[x]
	New ← Old + 1
	Mem[x] ← New

Ή

	Old ← Mem[x]
	New ← Old + 1
	Mem[x] ← New
Old ← Mem[x]	
New ← Old + 1	
Mem[x] ← New	

Χωρίς ατομικές εντολές

- Και τα δύο νήματα παίρνουν ως αποτέλεσμα στο Old την τιμή 0
- Η θέση μνήμης Mem[x] παίρνει την τιμή 1

	Αρχικοποίηση Mem[x] σε 0	
Χρόνος	Νήμα 1	Νήμα 2
1	Old ← Mem[x]	
2		Old ← Mem[x]
3	New ← Old + 1	
4	Mem[x] ← New	
5		New ← Old + 1
6		Mem[x] ← New

Ατομικές εντολές (Γενικά)

- Πραγματοποιείται από μια εντολή μηχανής στα περιεχόμενα μιας διεύθυνση μνήμης
  - Διάβασε την παλιά τιμή, υπολόγισε την νέα τιμή και αποθήκευσε την νέα τιμή στην διεύθυνση μνήμης
- Το υλικό εξασφαλίζει πως κανένα άλλο νήμα δεν μπορεί να προσπελάσει την διεύθυνση μνήμης μέχρι να ολοκληρωθεί η ατομική εντολή
  - Κάθε άλλο νήμα που θα προσπαθήσει να προσπελάσει την διεύθυνση μνήμης θα πρέπει να αναστείλει την εκτέλεση του
  - Όλα τα νήματα τελικά εκτελούν την ατομική εντολή σειριακά

Ατομικές εντολές (στην CUDA)

- Είναι υλοποιημένες ως κλήσεις συναρτήσεων, που τελικά μεταφράζονται σε απλές εντολές μηχανής (intrinsics)
  - Atomic add, sub, inc, dec, min, max, exch (exchange), CAS (compare and swap)
  - Διαβάστε το CUDA C Programming Guide 7.5 για λεπτομέρειες
    - https://docs.nvidia.com/cuda/cuda-c-programming-guide
- Ατομική πρόσθεση προσημασμένου ακεραίου 32-bit
  - `int atomicAdd(int *address, int val);`
  - Διαβάζει την τρέχουσα τιμή των 32-bit στην οποία δείχνει ο δείκτης **address** (καθολική ή κοινή μνήμη)
  - Υπολογίζει την τιμή **(old + val)**
  - Αποθηκεύει την τιμή αυτή στην ίδια διεύθυνση μνήμης
  - Επιστρέφει ως αποτέλεσμα την προηγούμενη τιμή που υπήρχε στην διεύθυνση μνήμης

Περισσότερες ατομικές εντολές πρόσθεσης στην CUDA

- Ατομική πρόσθεση μη προσημασμένου ακεραίου 32-bit
  - `unsigned int atomicAdd(unsigned int *address, unsigned int val);`
- Ατομική πρόσθεση προσημασμένου ακεραίου 64-bit
  - `unsigned long long int atomicAdd(unsigned long long int *address, unsigned long long int val);`
- Ατομική πρόσθεση αριθμού κινητής υποδιαστολής μονής ακρίβειας (**Compute capability > 2.0**)
  - `float atomicAdd(float *address, float val);`

Υλοποίηση μη υπάρχουσας ατομικής εντολής με χρήση υπάρχουσας

```
__device__ void atomicAdd(float *address, float val)
{
  int *address_as_i = (int *)address;
  int old = *address_as_i, assumed;
  do {
     assumed = old;
     old = atomicCAS(address_as_i, assumed,
     __float_as_int(val + __int_as_float(assumed)));
  } while (assumed != old);
}
```

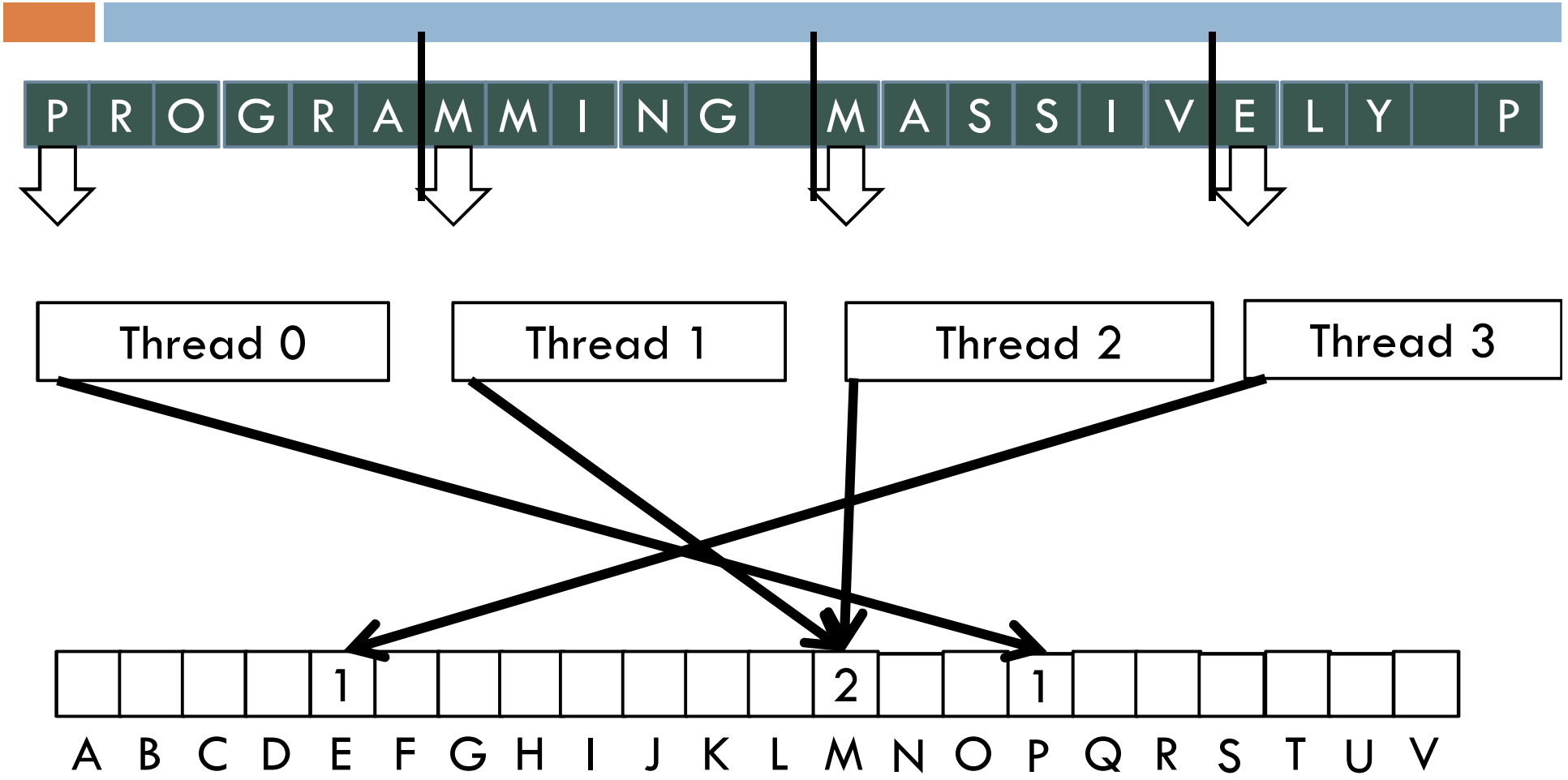
Υπολογισμός ιστογράμματος

- Μέθοδος για την εξαγωγή χρήσιμων χαρακτηριστικών και μοτίβων από μεγάλα σύνολα δεδομένων
  - Εξαγωγή χαρακτηριστικών για την αναγνώριση αντικειμένων σε εικόνες
  - Ανίχνευση απάτης σε συναλλαγές με πιστωτικές κάρτες
  - Συσχέτιση κινήσεων ουράνιων σωμάτων στην αστροφυσική
  - …
- Βασικός αλγόριθμος
  - Χρησιμοποίησε την τιμή κάθε στοιχείου του συνόλου δεδομένων ως αναγνωριστικό ενός «δοχείου», του οποίου την τιμή θα αυξήσεις κατά ένα

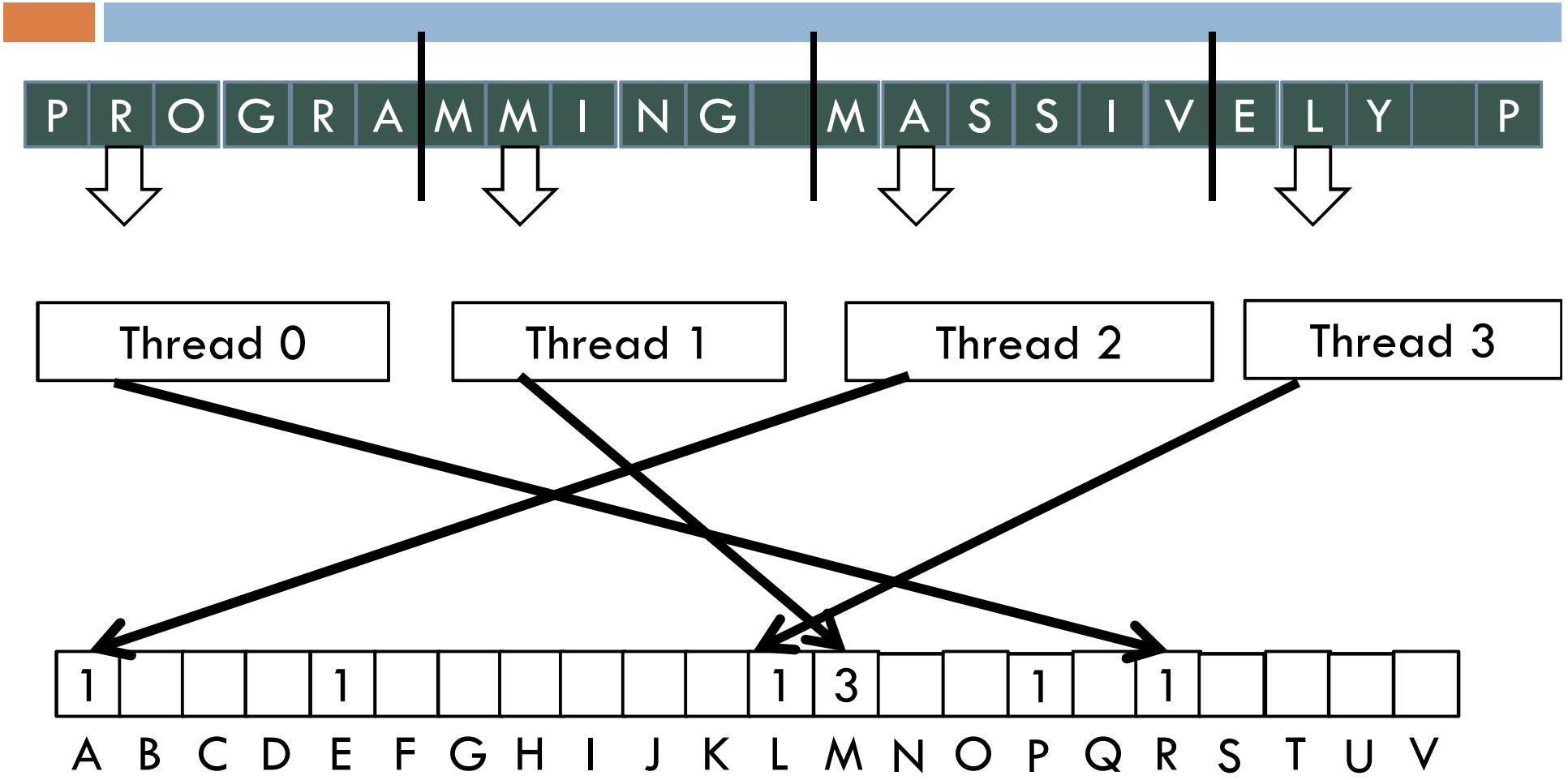
Παράδειγμα ιστογράμματος

- Για την πρόταση “Programming Massively Parallel Processors” φτιάξτε ένα ιστόγραμμα για την συχνότητα εμφάνισης κάθε γράμματος
- A(4), C(1), E(3), G(1), …
  - Πως το κάνουμε αυτό παράλληλα;
    - Ανέθεσε σε κάθε νήμα τον υπολογισμό για ένα τμήμα των δεδομένων εισόδου
    - Για κάθε χαρακτήρα, χρησιμοποίησε ατομικές εντολές για την δημιουργία του ιστογράμματος

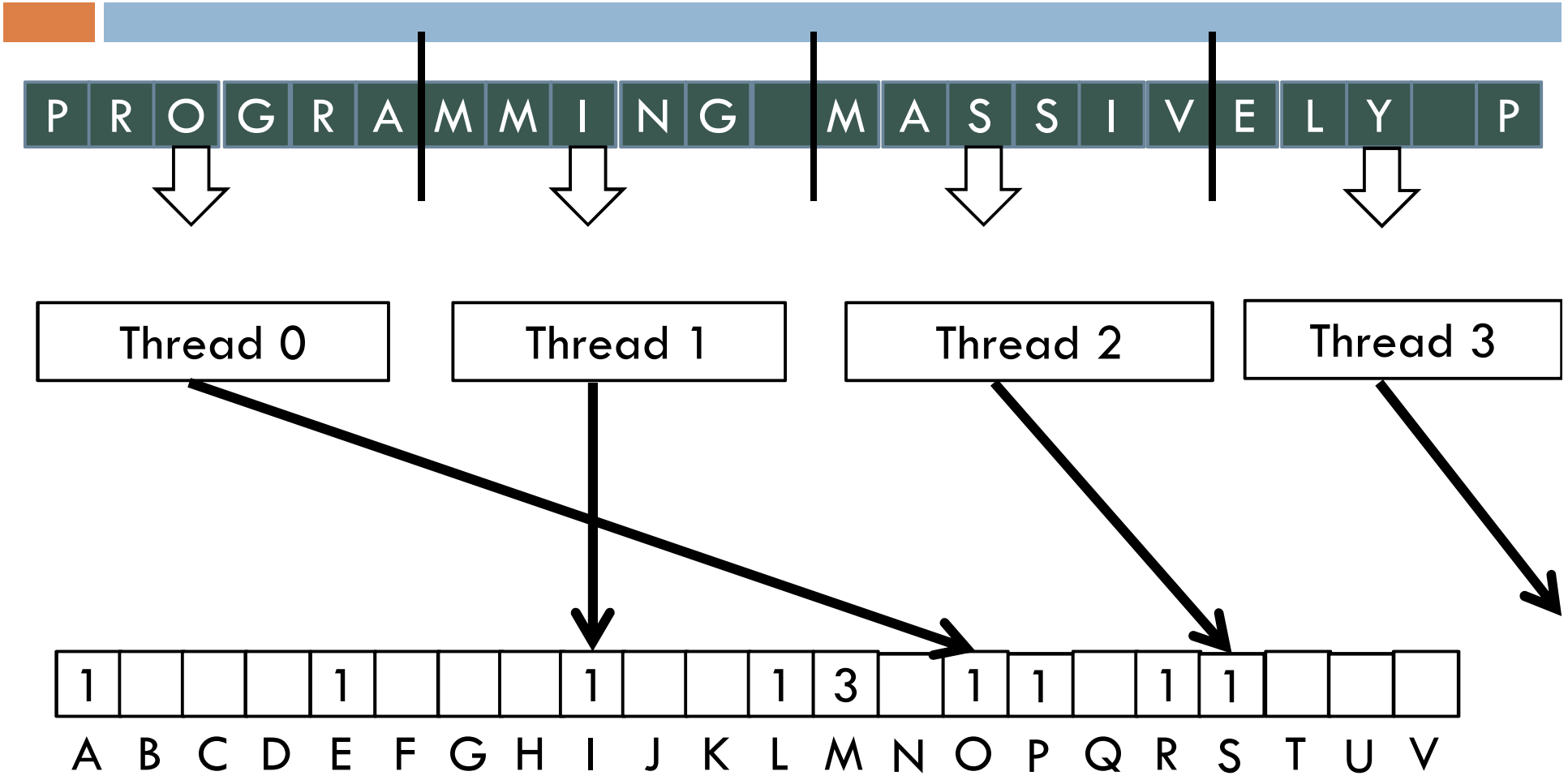
1η επανάληψη – 1ο γράμμα σε κάθε τμήμα



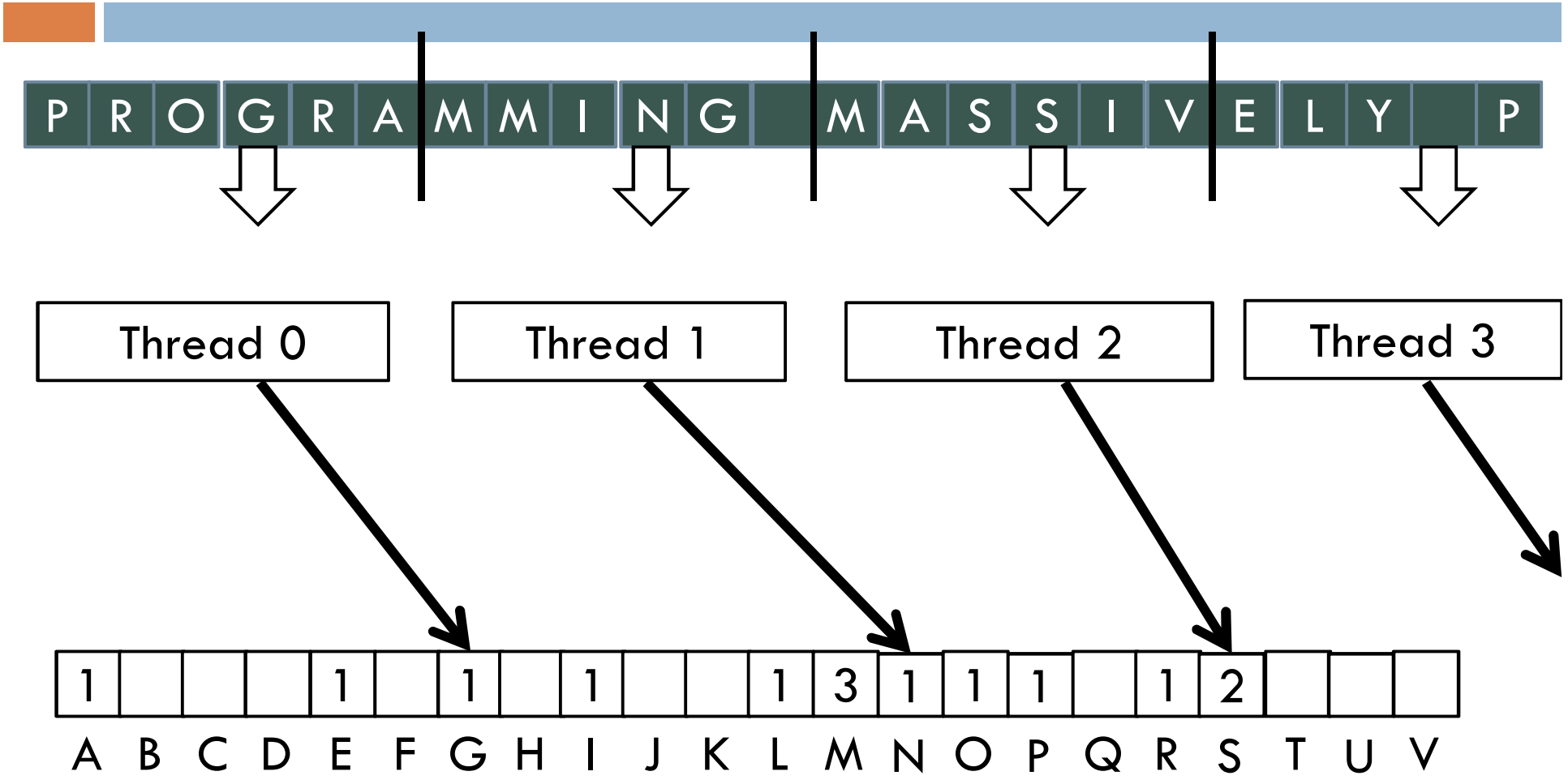
2η επανάληψη – 2ο γράμμα σε κάθε τμήμα



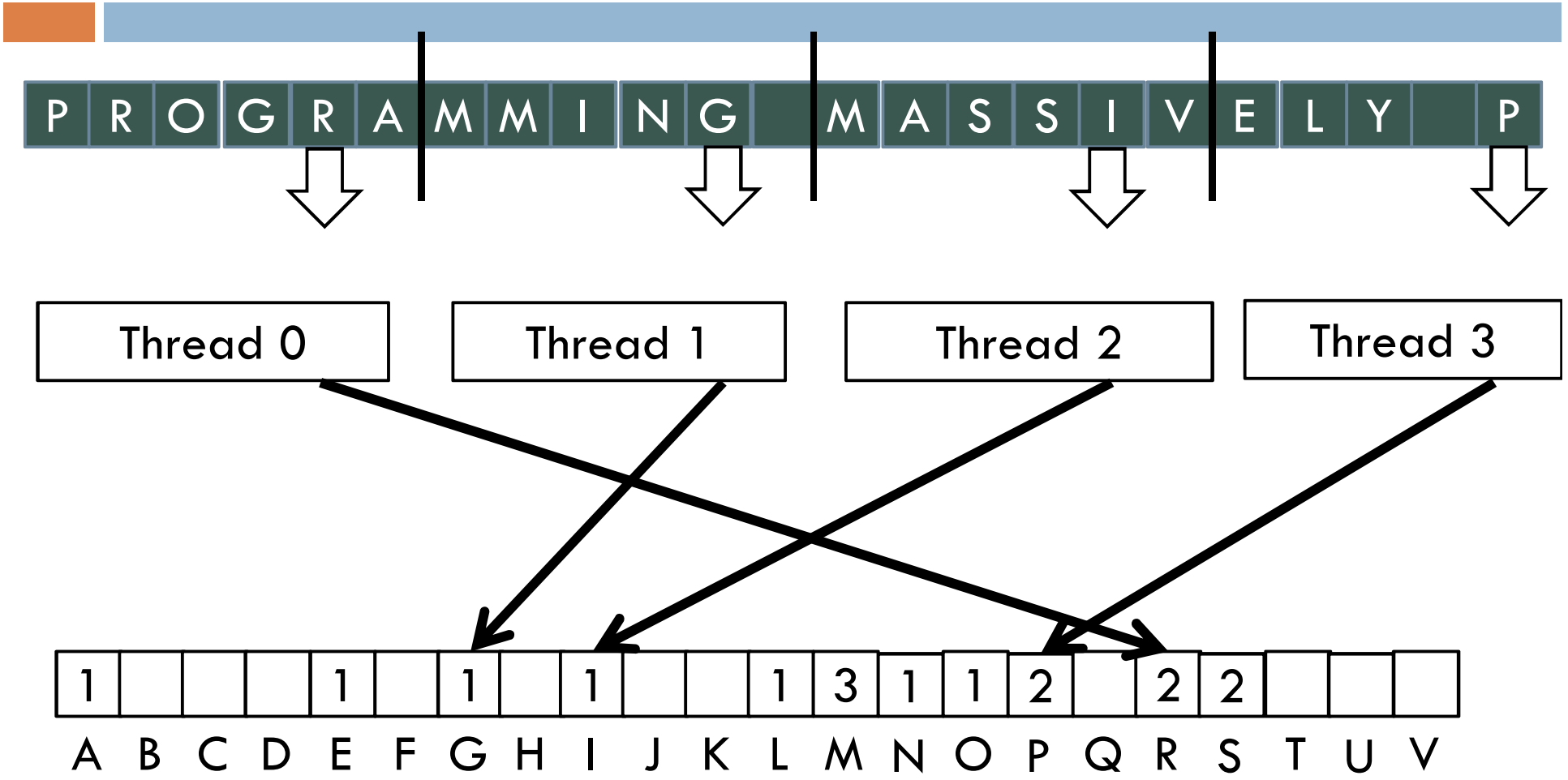
3η επανάληψη – 3ο γράμμα σε κάθε τμήμα



4η επανάληψη – 4ο γράμμα σε κάθε τμήμα

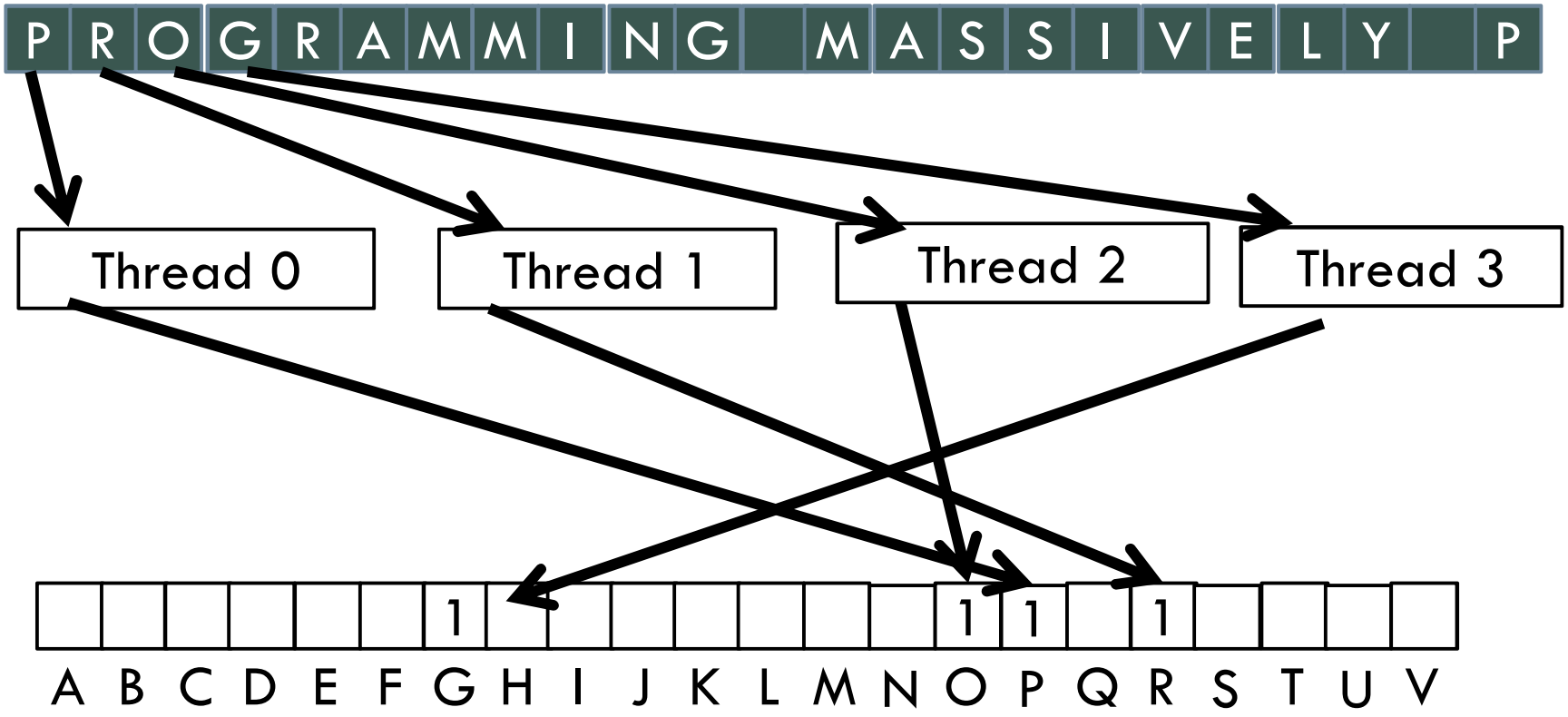


5η επανάληψη – 5ο γράμμα σε κάθε τμήμα



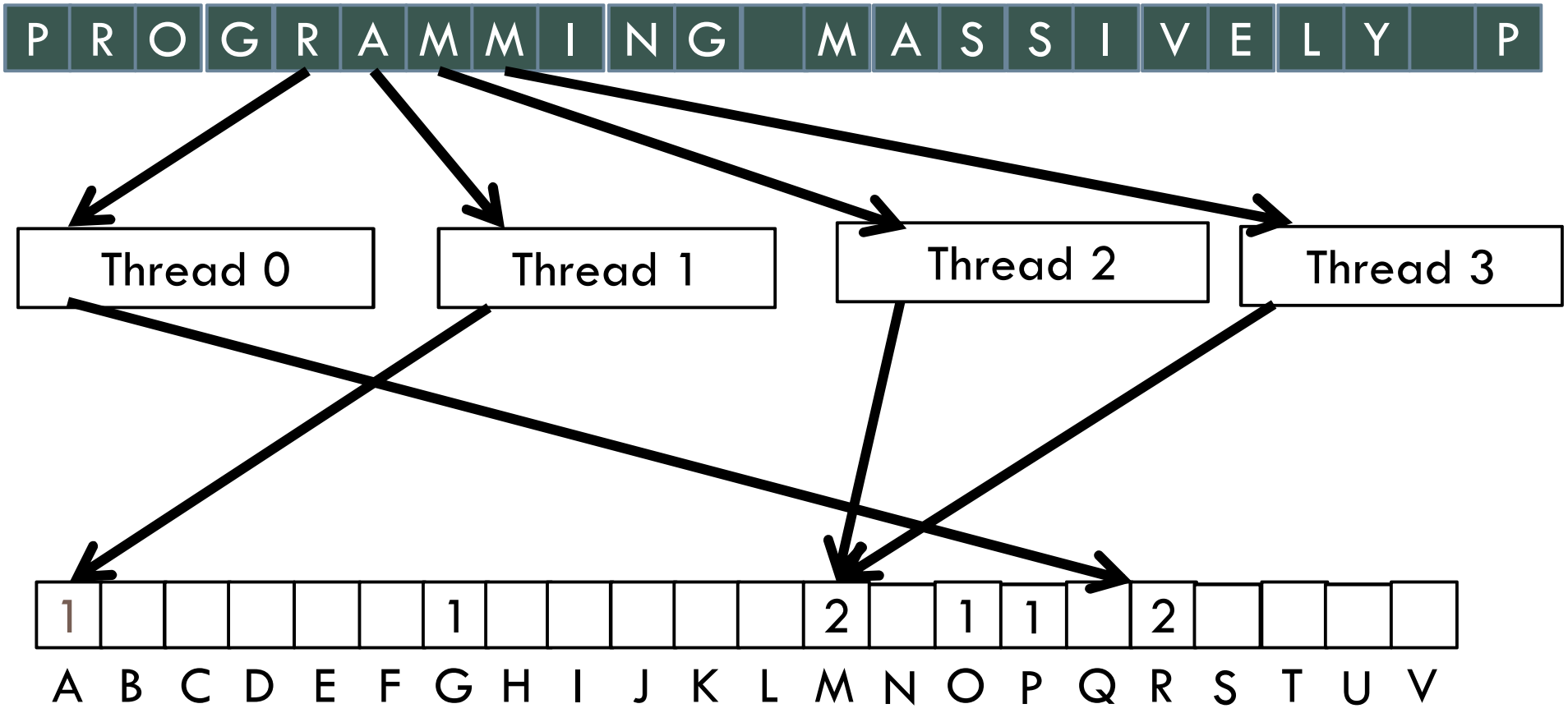
Που υπάρχει πρόβλημα στον αλγόριθμο;

- Διαβάζει είσοδο από μη συνεχόμενες θέσεις μνήμης
  - Ανάθεση στοιχείων στα νήματα σε βήματα (strides)
  - Διαδοχικά νήματα επεξεργάζονται διαδοχικά γράμματα



2η επανάληψη

- Όλα τα νήματα προχωράνε στο επόμενο τμήμα των δεδομένων εισόδου



Συνάρτηση πυρήνα για ιστόγραμμα

- Η συνάρτηση πυρήνα παίρνει ως παράμετρο έναν δείκτη προς τα δεδομένα εισόδου
- Κάθε νήμα επεξεργάζεται τα δεδομένα αυτά σε βήματα (strided)

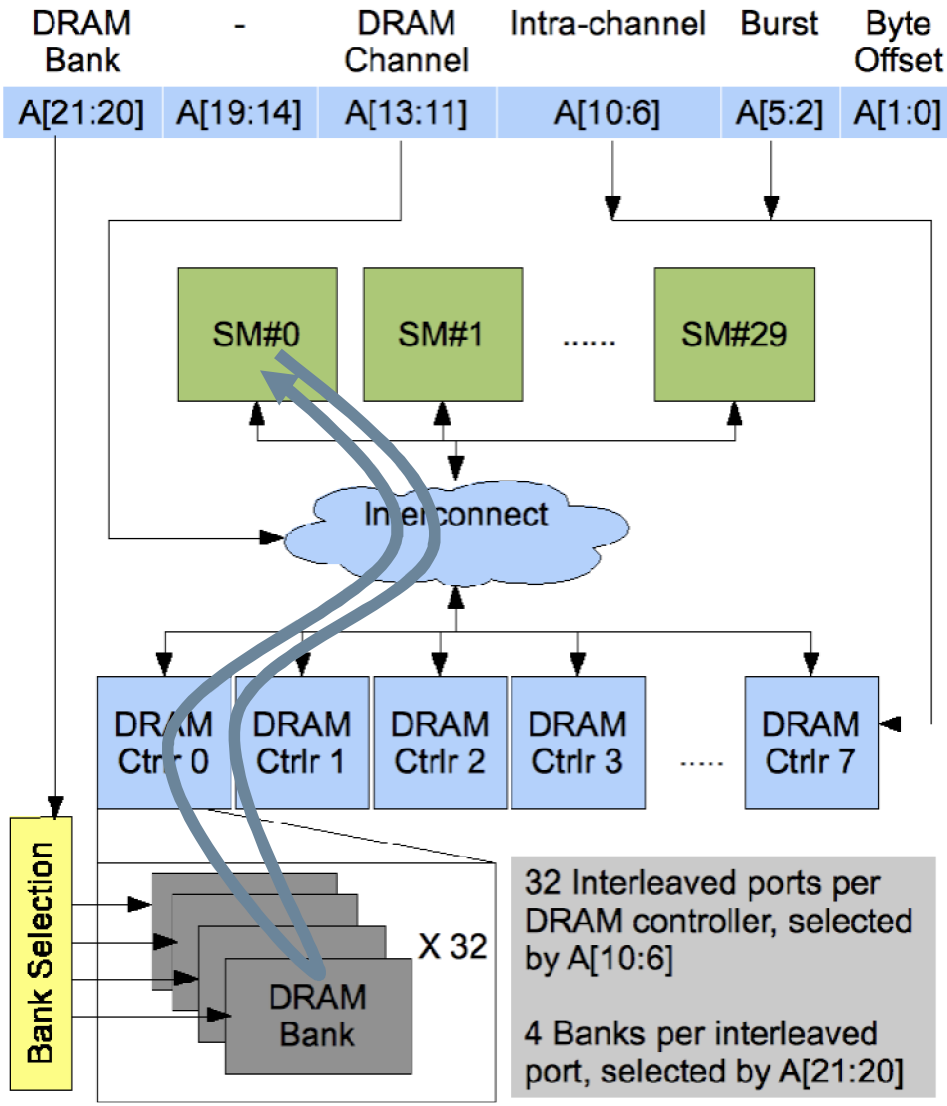
```
__global__ void histo_kernel(unsigned char *buffer,
                             long size, unsigned int *histo)
{
    int i = threadIdx.x + blockIdx.x * blockDim.x;

    // stride is total number of threads
    int stride = blockDim.x * gridDim.x;
```

Συνέχεια συνάρτησης πυρήνα

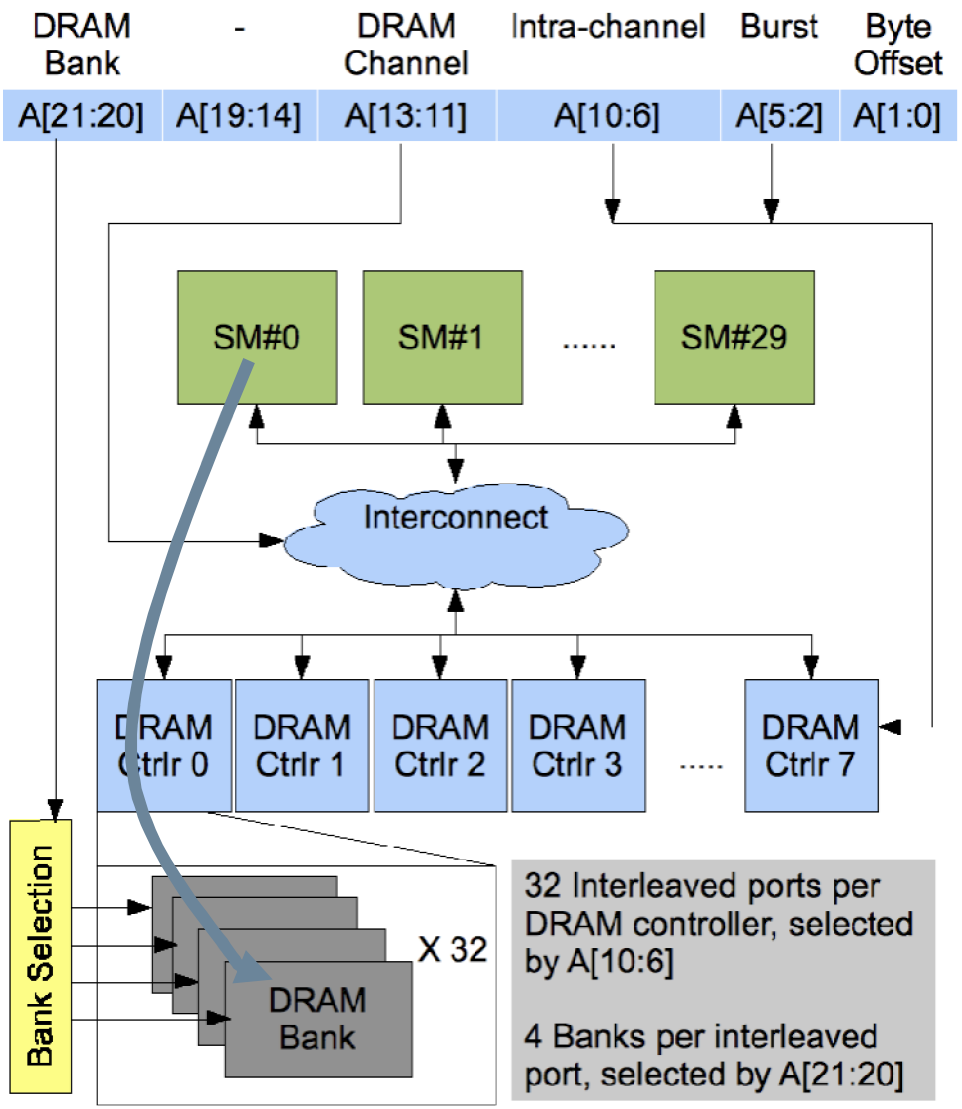
```
    // All threads handle blockDim.x * gridDim.x
    // consecutive elements
    while (i < size) {
        atomicAdd( &(histo[buffer[i]]), 1);
        i += stride;
    }
}
```

Ατομικές εντολές στην καθολική μνήμη



- Μια ατομική εντολή ξεκινάει με μια ανάγνωση δεδομένων
  - Καθυστέρηση αρκετών εκατοντάδων κύκλων ρολογιού

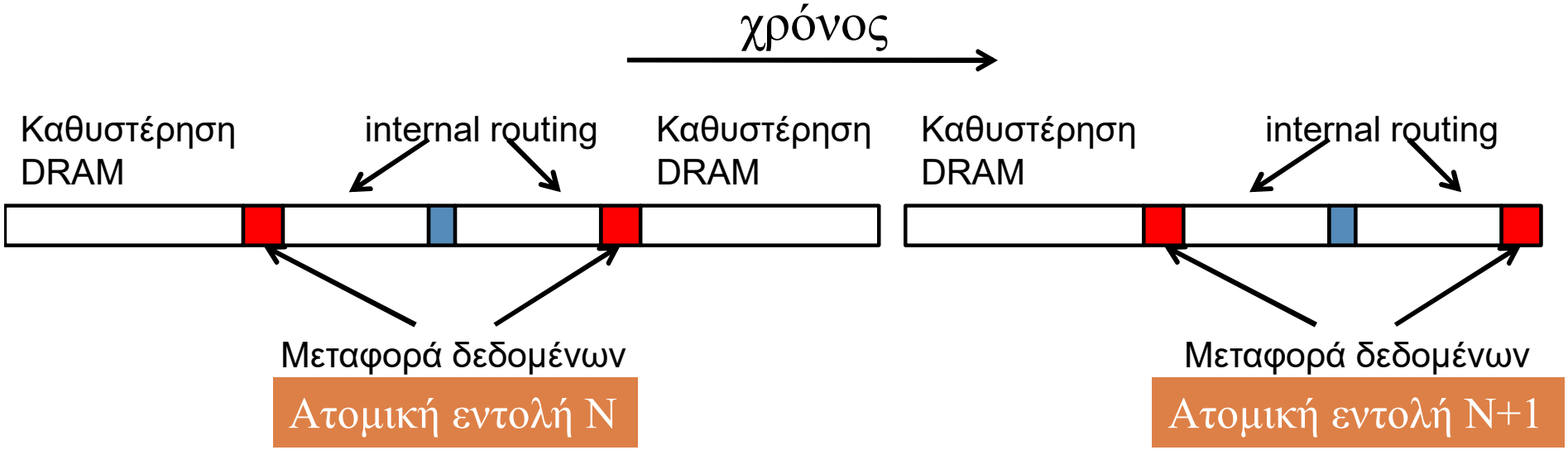
Ατομικές εντολές στην καθολική μνήμη



- Μια ατομική εντολή ξεκινάει με μια ανάγνωση δεδομένων
  - Καθυστέρηση αρκετών εκατοντάδων κύκλων ρολογιού
- Μια ατομική εντολή τελειώνει με μια εγγραφή δεδομένων
  - Καθυστέρηση αρκετών εκατοντάδων κύκλων ρολογιού
- Κατά την διάρκεια των παραπάνω λειτουργιών, κανένα άλλο νήμα δεν μπορεί να προσπελάσει την θέση μνήμης

Ατομικές εντολές στην καθολική μνήμη

- Κάθε κύκλος Ανάγνωσης-Τροποποίησης-Εγγραφής έχει καθυστέρηση δύο προσπελάσεων στην μνήμη
  - Όλες οι ατομικές εντολές που εκτελούνται στην ίδια μεταβλητή (ίδια θέση μνήμης) σειριοποιούνται



Ταχύτητα εκτέλεσης ατομικών εντολών

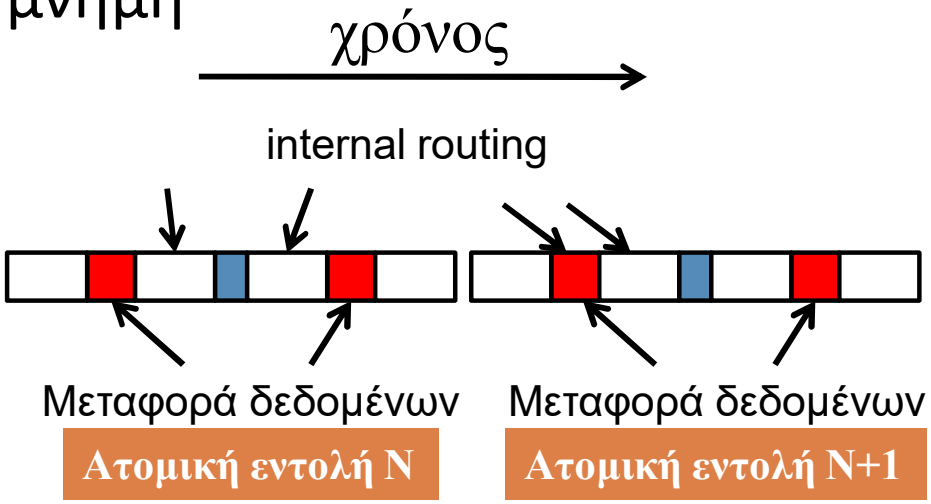
- Ο χρόνος απόκρισης (latency) καθορίζει τον ρυθμό εκτέλεσης (throughput) των ατομικών εντολών
  - Ο ρυθμός εκτέλεσης μιας ατομικής εντολής είναι ο ρυθμός με τον οποίο μπορεί μια εφαρμογή να εκτελεί την ατομική εντολή για μια συγκεκριμένη θέση μνήμης
- Ο ρυθμός αυτός οριοθετείται από τον συνολικό χρόνο που απαιτεί ένας κύκλος Ανάγνωσης-Τροποποίησης-Εγγραφής
  - Τυπικά πάνω από 1000 κύκλοι ρολογιού για θέσεις μνήμης στην καθολική μνήμη (DRAM)
- Αυτό σημαίνει πως αν πολλά νήματα προσπαθούν να εκτελέσουν μια ατομική εντολή στην ίδια θέση μνήμης, το εύρος ζώνης μειώνεται σε < 1/1000 του διαθέσιμου!

Ανάλογο με την ουρά αναμονής σε Super Market

- Κάποιοι πελάτες συνειδητοποιούν ότι ξέχασαν κάποιο είδος αφότου έχουν ξεκινήσει να αφήνουν πράγματα στον ιμάντα
- Τρέχουν στον διάδρομο και παίρνουν το είδος όσο όλοι οι υπόλοιποι στην ουρά περιμένουν
  - Ο ρυθμός εξυπηρέτησης στο ταμείο μειώνεται λόγω του μεγάλου χρόνου που απαιτείται για να πάμε στον διάδρομο και να επιστρέψουμε
- Φανταστείτε ένα κατάστημα στο οποίο κάθε πελάτης μπαίνει στην ουρά του ταμείου πριν καν πάρει οποιοδήποτε είδος!
  - Ο ρυθμός εξυπηρέτησης θα είναι 1 / (Συνολικός χρόνος αγορών για κάθε πελάτη)

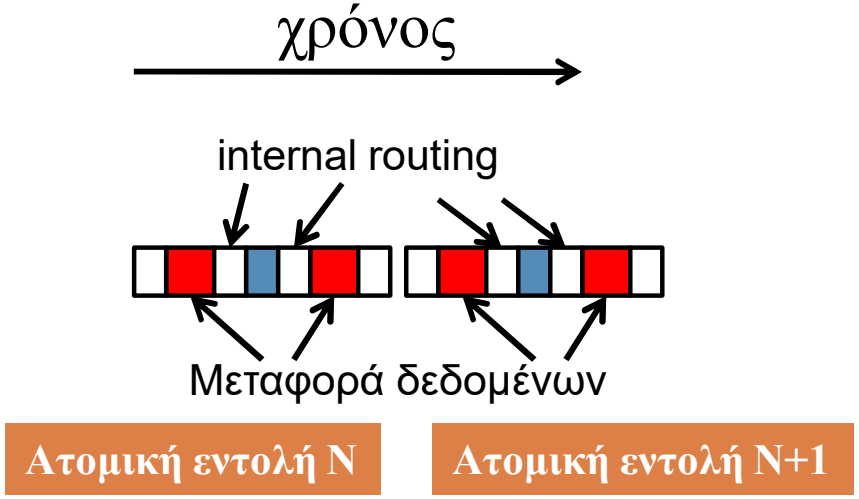
Βελτιώσεις στο υλικό

- Οι ατομικές εντολές στην αρχιτεκτονική Fermi εκτελούνται στην κρυφή μνήμη L2
  - Μικρότερη καθυστέρηση, αλλά πάλι σειριοποιημένες
  - Διαμοιραζόμενη μεταξύ όλων των block
  - "Δωρεάν" βελτίωση σε σχέση με ατομικές εντολές στην καθολική μνήμη



Βελτιώσεις στο υλικό

- Ατομικές εντολές στην κοινή μνήμη
  - Ακόμα μικρότερη καθυστέρηση, αλλά πάλι σειριοποιημένες
  - Τοπική για κάθε block
  - Χρειάζεται προσαρμογή του αλγορίθμου από τον προγραμματιστή



Ατομικές εντολές στην κοινή μνήμη απαιτούν ιδιωτικοποίηση μεταβλητών

- Δημιουργία τοπικών αντιγράφων του πίνακα `histo[]` για κάθε block νημάτων

```
__global__ void histo_kernel(unsigned char *buffer,
                             long size, unsigned int *histo)
{
    // Each character is 1 byte => 256 different characters
    __shared__ unsigned int histo_private[256];

    if (threadIdx.x < 256)
        histo_private[threadidx.x] = 0;
   __syncthreads();
```

Δημιουργία τοπικού ιστογράμματος

```
int i = threadIdx.x + blockIdx.x * blockDim.x;

// stride is total number of threads
int stride = blockDim.x * gridDim.x;

while (i < size) {
    atomicAdd( &(histo_private[buffer[i]), 1);
    i += stride;
}
```

Δημιουργία συνολικού ιστογράμματος

```
    // wait for all other threads in the block to finish
    __syncthreads();

    if (threadIdx.x < 256)
        atomicAdd( &(histo[threadIdx.x]),
                   histo_private[threadIdx.x] );

}
```

Ιδιωτικοποίηση μεταβλητών

- Ισχυρή και συχνά χρησιμοποιούμενη τεχνική για την παραλληλοποίηση εφαρμογών
- Η πράξη πρέπει να είναι αντιμεταθετική και προσεταιριστική
  - Η πρόσθεση που χρησιμοποιούμε στον υπολογισμό του ιστογράμαμτος έχει αυτές τις ιδιότητες
- Το μέγεθος του ιστογράμματος πρέπει να είναι μικρό
  - Να χωράει στην κοινή μνήμη
- Τι μπορούμε να κάνουμε αν το ιστόγραμμα είναι πολύ μεγάλο και δεν μπορούμε να το ιδιωτικοποιήσουμε;