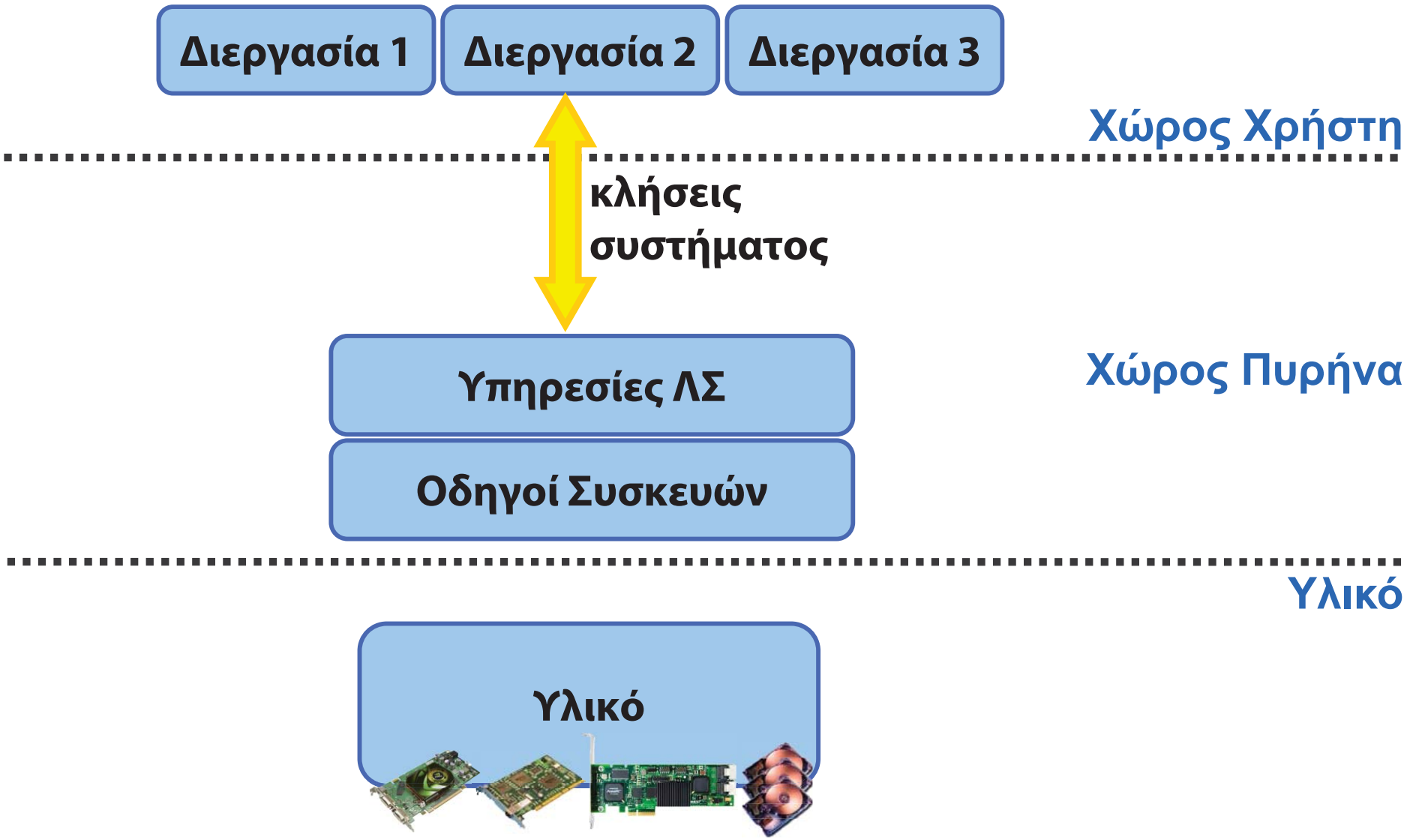
Διεργασίες - Σύνοψη

- Οργάνωση ενός σύγχρονου ΛΣ
  - Διακοπές, προνομιούχος κατάσταση
  - Κλήσεις συστήματος
- Διεργασία – Process
  - Ορισμός, μεταβάσεις κατάστασης – κύκλος ζωής
- Πίνακας Ελέγχου Διεργασίας
- Χρονοδρομολόγηση σε συστήματα καταμερισμού χρόνου
- Λειτουργίες διεργασιών
  - Δημιουργία διεργασιών - το μοντέλο του UNIX
- Διαδιεργασιακή Επικοινωνία

Διεργασίες - Σύνοψη

- Οργάνωση ενός σύγχρονου ΛΣ
  - Διακοπές, προνομιούχος κατάσταση
  - Κλήσεις συστήματος
- Διεργασία – Process
  - Ορισμός, μεταβάσεις κατάστασης – κύκλος ζωής
- Πίνακας Ελέγχου Διεργασίας
- Χρονοδρομολόγηση σε συστήματα καταμερισμού χρόνου
- Λειτουργίες διεργασιών
  - Δημιουργία διεργασιών - το μοντέλο του UNIX
- Διαδιεργασιακή Επικοινωνία

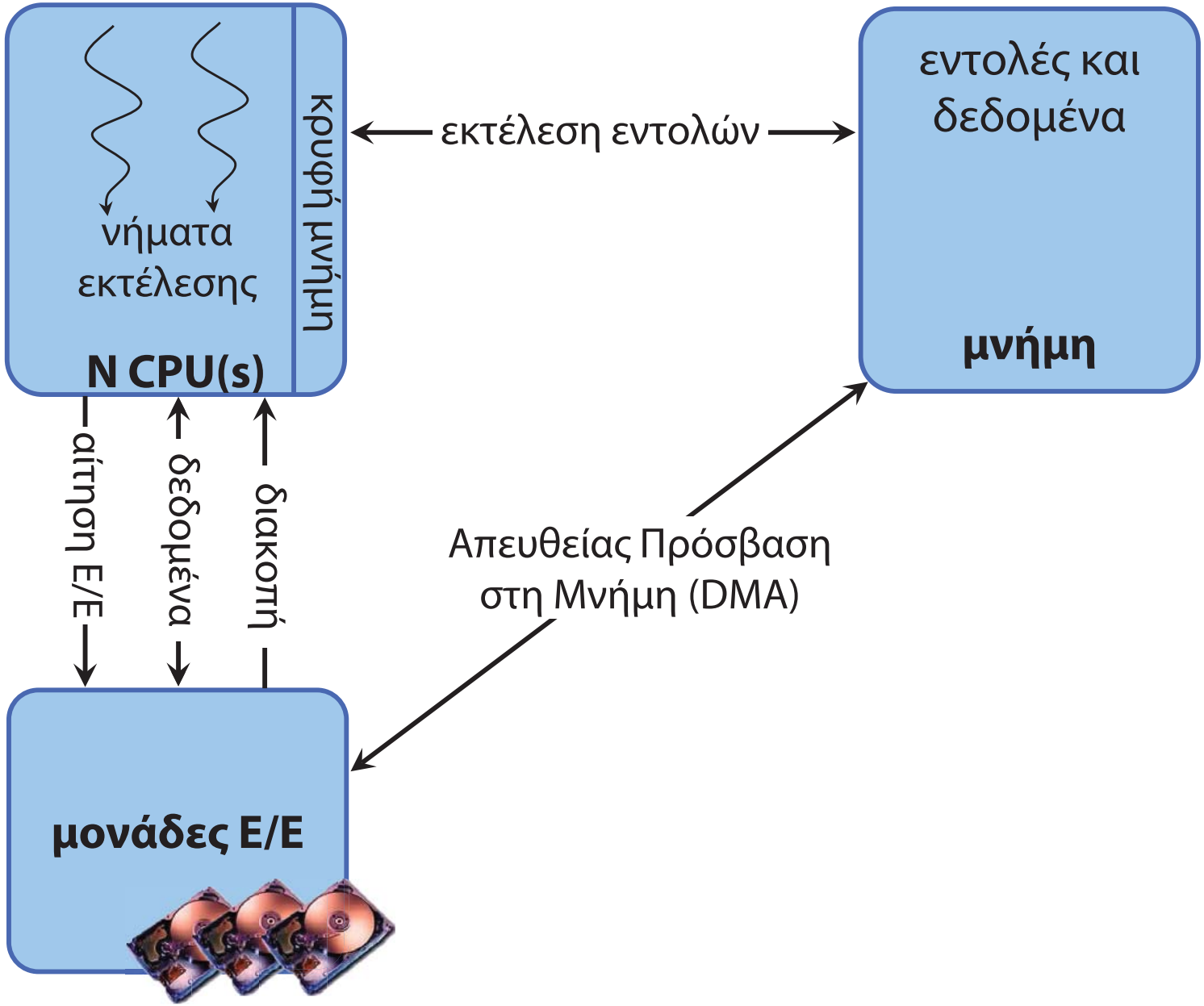
Οργάνωση ενός ΛΣ



Λειτουργία ενός Σύγχρονου Συστήματος (1)

- Βασισμένη σε *αιτήματα / συμβάντα / διακοπές*
- Το ΛΣ ενεργοποιείται όταν γίνει κάτι από τα παρακάτω:
  - Αίτηση διακοπής από το λογισμικό **(trap, software interrupt)**
  - Ολοκλήρωση λειτουργίας Ε/Ε **(διακοπή υλικού, hardware interrupt)**
  - Εσφαλμένη ή μη επιτρεπόμενη εντολή, π.χ. διαίρεση με το μηδέν, **(exception, εξαίρεση)**
- Ο έλεγχος περνάει σε προκαθορισμένο κώδικα του ΛΣ, ανάλογα με τη διακοπή (άνυσμα διακοπών), την αίτηση ή την εξαίρεση
  - Ρουτίνα εξυπηρέτησης διακοπής
  - Μετά το τέλος της, επιστροφή στο προηγούμενο σημείο, υπό προϋποθέσεις

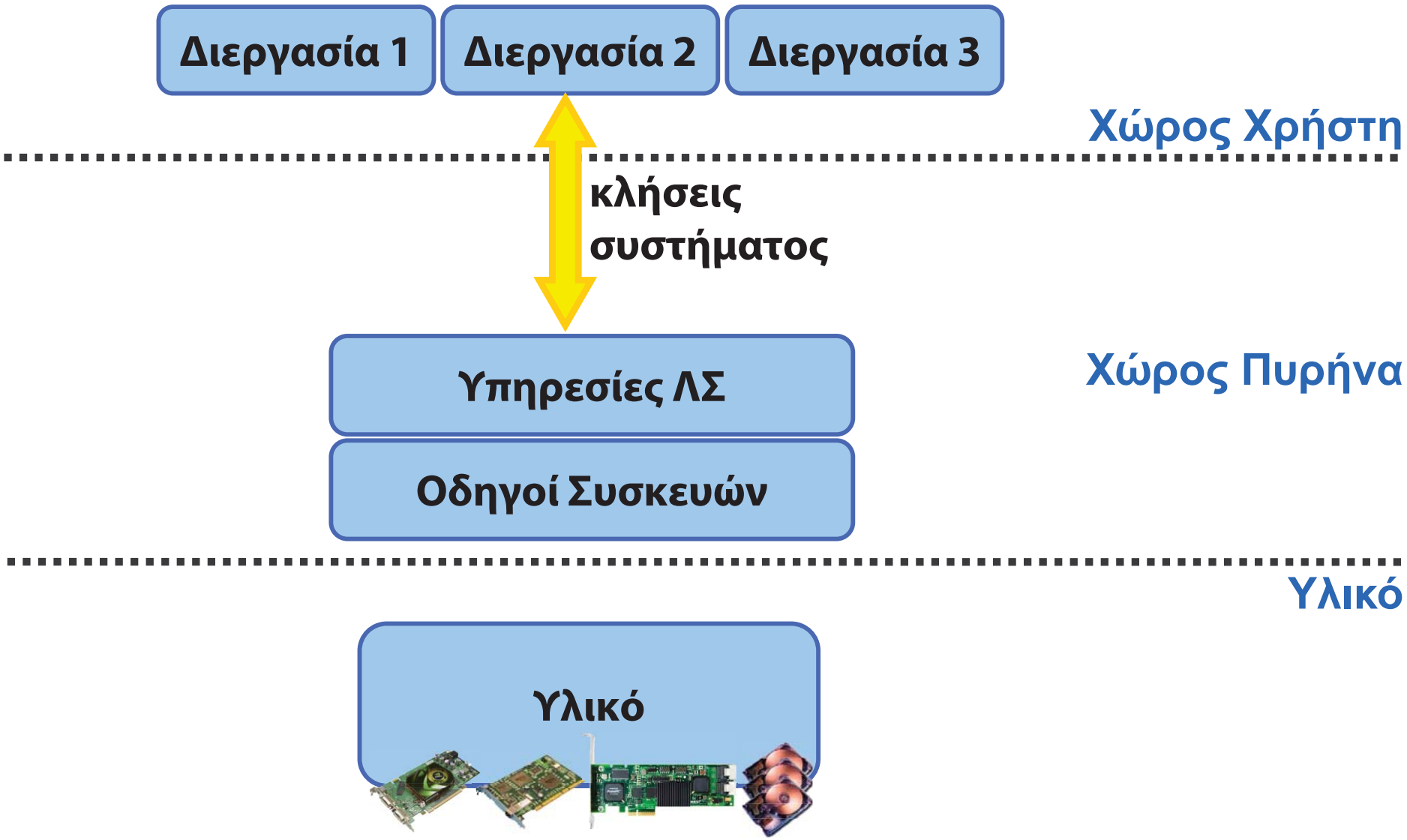
Λειτουργία ενός Σύγχρονου Συστήματος (2)



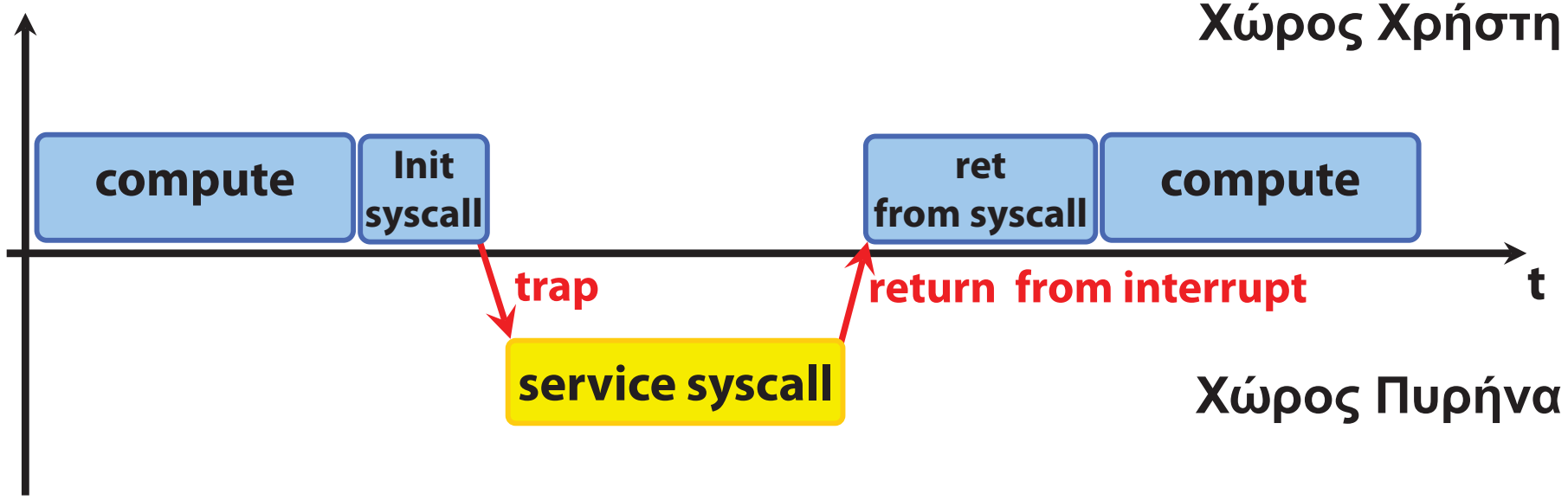
Λειτουργία ενός Σύγχρονου Συστήματος (3)

- Ανάγκη για Προστασία / Απομόνωση
  - Των υπόλοιπων προγραμμάτων
  - Των δομών του ΛΣ
  - Από κακογραμμένο / κακόβουλο κώδικα
- Λύση: Δύο καταστάσεις λειτουργίας
  - Κατάσταση Χρήστη – User Mode
  - Κατάσταση Επόπτη – Supervisor / Kernel Mode
- Πρέπει να παρέχεται από το **Υλικό** (x86;)
  - Ορισμένες εντολές είναι *προνομιούχες* (privileged)
  - Οι ρουτίνες εξυπηρέτησης διακοπών τρέχουν σε κατάσταση επόπτη

Οργάνωση ενός ΛΣ



Κλήση Συστήματος



- Αίτηση προς το ΛΣ (π.χ. Ε/Ε από συσκευή ή αρχείο, αναμονή συμβάντος, διαχείριση διεργασιών)
- Μετάβαση από κατάσταση χρήστη σε κατάσταση πυρήνα (privileged mode)
- Κι αν χρειάζεται να περιμένει έως ότου ολοκληρωθεί μια λειτουργία Ε/Ε;
- Πόσο είναι το κόστος της κλήσης; Υπάρχουν κι άλλοι τρόποι.
  - ➡ **SYSENTER / SYSEXIT**

Διεργασίες - Σύνοψη

- Οργάνωση ενός σύγχρονου ΛΣ
  - Διακοπές, προνομιούχος κατάσταση
  - Κλήσεις συστήματος
- Διεργασία – Process
  - Ορισμός, μεταβάσεις κατάστασης – κύκλος ζωής
- Πίνακας Ελέγχου Διεργασίας
- Χρονοδρομολόγηση σε συστήματα καταμερισμού χρόνου
- Λειτουργίες διεργασιών
  - Δημιουργία διεργασιών - το μοντέλο του UNIX
- Διαδιεργασιακή Επικοινωνία

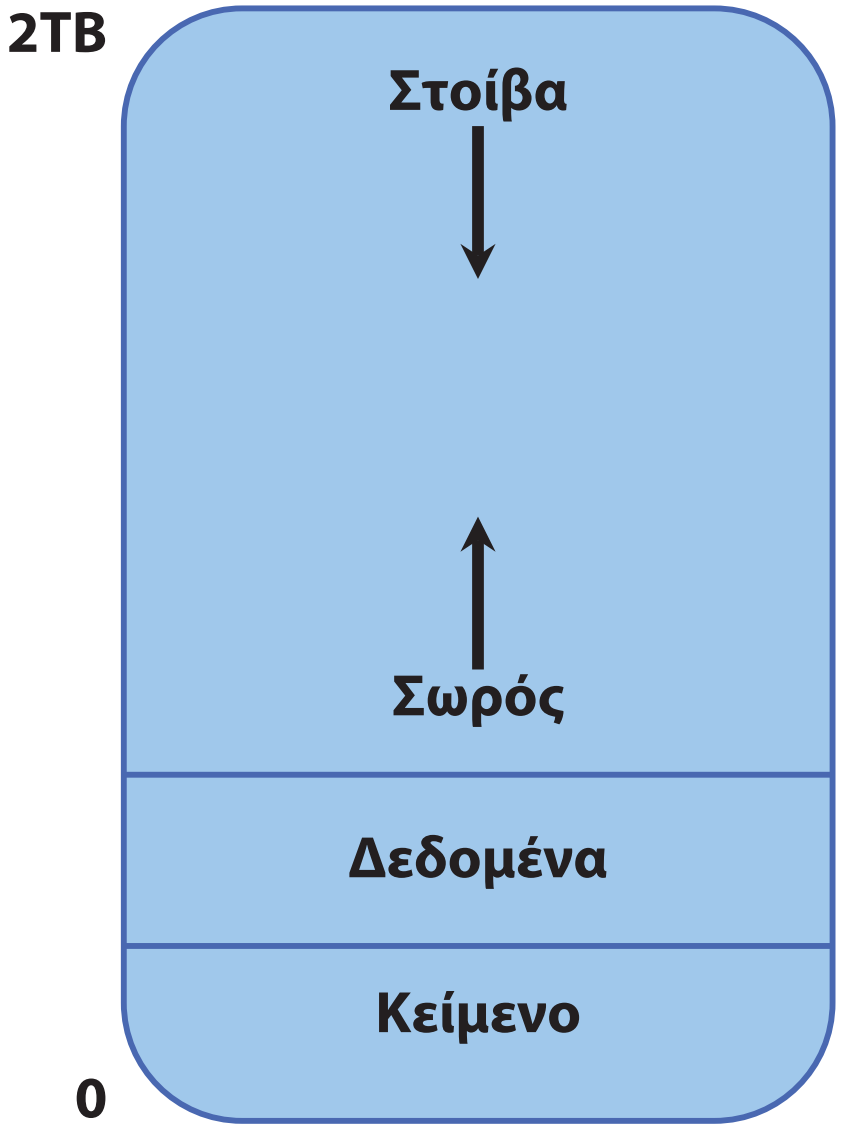
Διεργασία – Process

- Διεργασία: ένα πρόγραμμα υπό *εκτέλεση*
  - Ένα *ζωντανό* πρόγραμμα, φορτωμένο στη μνήμη
  - Διεργασία = κώδικας + *κατάσταση*
- Πολλές διεργασίες μπορεί να έχουν κοινό πρόγραμμα, κοινό κώδικα
  - Κάθε μία έχει τη δική της, χωριστή κατάσταση
- Πολλές διεργασίες εκτελούνται ταυτόχρονα κάτω από ΛΣ καταμερισμού χρόνου
  - Multiprogramming με time sharing
- Αναγνωρίζεται από μοναδικό Process ID (**PID**)

Τι είναι μια Διεργασία;

- κώδικας - program *text*
- Αρχιτεκτονική κατάσταση
  - Μετρητής προγράμματος (PC)
  - Καταχωρητές CPU (%eax, %ebx, %ecx)
- Ιδιωτικός, απομονωμένος χώρος μνήμης
  - Τμήμα δεδομένων (data segment) – μεταβλητές
  - Σωρός (heap)
  - Στοίβα (stack)
- Χρησιμοποιούμενοι πόροι του συστήματος
  - Ανοιχτά αρχεία, δικτυακές συνδέσεις

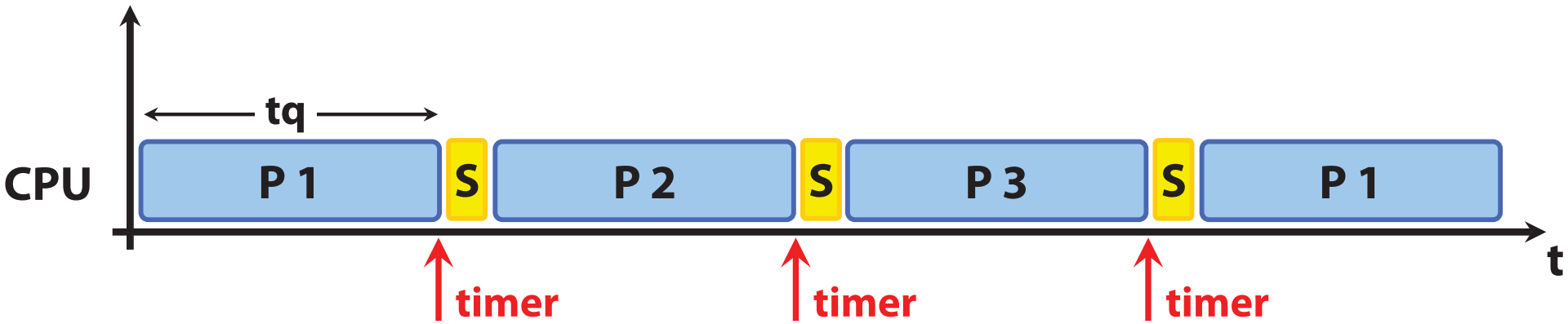
Στιγμιότυπο Διεργασίας στη Μνήμη



Σύγχρονα ΛΣ καταμερισμού χρόνου

- Διεργασία: η βασική μονάδα εκτέλεσης σε σύγχρονα ΛΣ καταμερισμού χρόνου
  - Ταυτόχρονη, *απομονωμένη* εκτέλεση
    - Με τη δική της αρχιτεκτονική κατάσταση
    - Σε δικό της, απομονωμένο χώρο μνήμης
    - Αλληλεπίδραση με τον έξω κόσμο → **κλήσεις συστήματος**
- Το ΛΣ δημιουργεί και συντηρεί την ψευδαίσθηση της αποκλειστικής χρήσης της μηχανής
  - καταμερισμός χρόνου ανάμεσα στις υπό εκτέλεση διεργασίες
    - Ανάγκη για *χρονοδρομολόγηση* (**CPU Scheduling**)
    - Δυνητικά σε πολλούς επεξεργαστές (SMP/multicore)
  - Με δίκαιη κατανομή πόρων ανάμεσα στις διεργασίες

Καταμερισμός Χρόνου: η γενική ιδέα

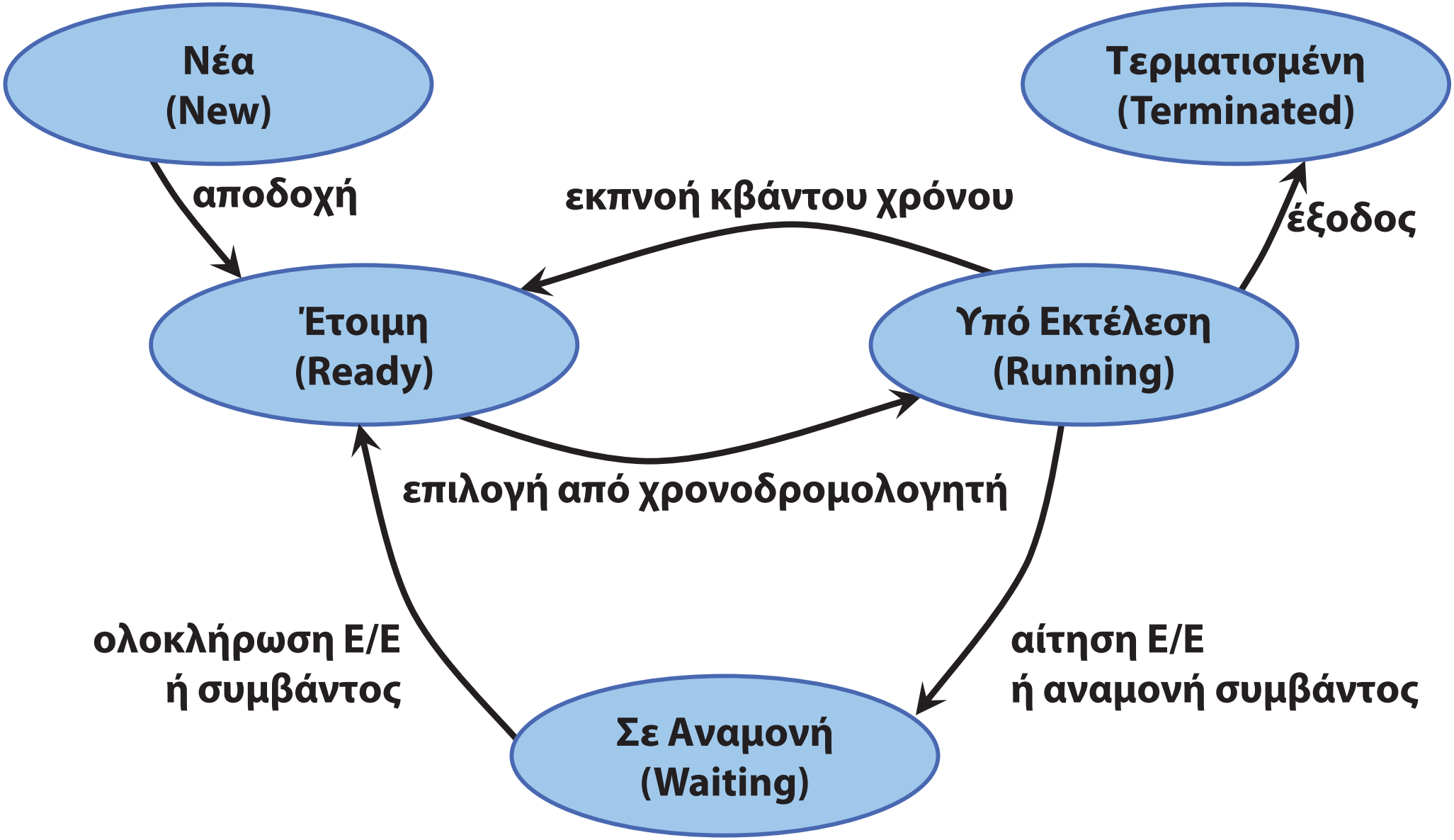


- Ο υπολογιστικός χρόνος κατανέμεται ανάμεσα στις διεργασίες που είναι έτοιμες να τρέξουν (P1, P2, P3)
  - Κάθε διεργασία τρέχει για χρόνο <= του *κβάντου χρόνου* (time quantum)
- Το χρονοδρομολογητή ενεργοποιούν *διακοπές χρονιστή* (timer interrupts)

Κατάσταση Διεργασίας – Μεταβάσεις

- Κάθε διεργασία μεταβαίνει από κατάσταση σε κατάσταση (*process state)*
  - **Νέα (New)**
    - δημιουργείται
  - **Υπό Εκτέλεση (Running)**
    - εκτελείται τώρα στη CPU
  - **Έτοιμη (Ready)**
    - έτοιμη προς εκτέλεση σε CPU, χρειάζεται χρόνο CPU
  - **Σε Αναμονή (Waiting)**
    - περιμένει να ολοκληρωθεί Ε/Ε, αναμονή συμβάντος, δεν είναι υποψήφια για εκτέλεση σε CPU
  - **Τερματισμένη (Terminated)**
    - η εκτέλεσή της έχει ολοκληρωθεί

Κύκλος Ζωής μιας Διεργασίας (1)



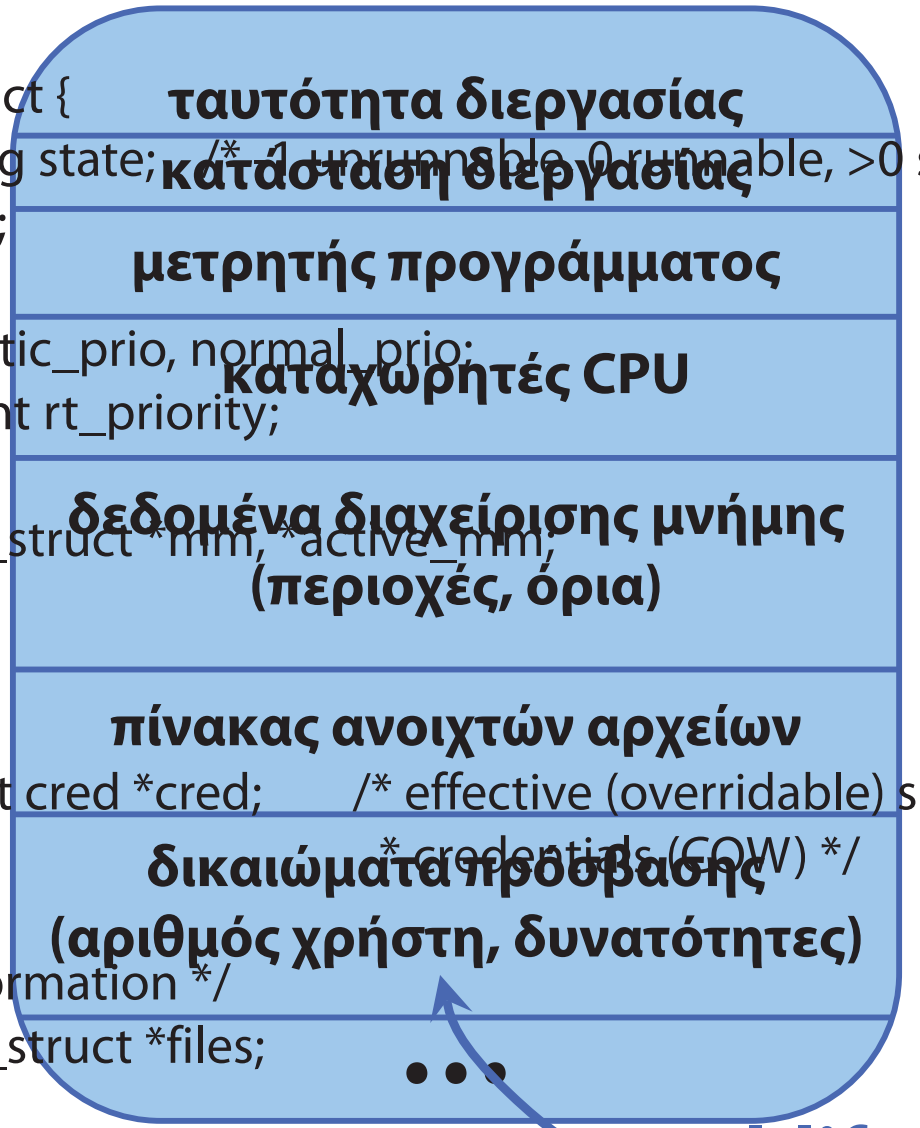
Πίνακας Ελέγχου Διεργασίας (1)

- Process Control Block (PCB)
  - Ταυτότητα διεργασίας (PID)
  - Κατάσταση διεργασίας (process state)
    - Νέα, Έτοιμη, Υπό Εκτέλεση, Σε Αναμονή, Τερματισμένη
  - Αρχιτεκτονική κατάσταση (architectural state)
    - Μετρητής προγράμματος, καταχωρητές CPU
  - Πληροφορίες χρονοδρομολόγησης
    - Προτεραιότητα, δείκτες σε ουρές χρονοδρομολογητή
  - Πληροφορίες διαχείρισης μνήμης
    - Δεσμευμένες περιοχές μνήμης, είδος και όρια πρόσβασης
  - Πληροφορίες για έλεγχο πρόσβασης
    - Ταυτότητα χρήστη, δικαιώματα πρόσβασης, δυνατότητες
  - Πληροφορίες για τρέχουσα κατάσταση λειτουργιών Ε/Ε
    - Πίνακας ανοιχτών αρχείων

Πίνακας Ελέγχου Διεργασίας (2)

$ **cat /usr/src/linux/include/linux/sched.h**

```
...
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
       ...
        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
       ...
        struct mm_struct *mm, *active_mm;
       ...
        pid_t pid;
       ...
        const struct cred *cred;        /* effective (overridable) subjective task
                                         * credentials (COW) */
       ...
/* open file information */
        struct files_struct *files;
       ...
};
```

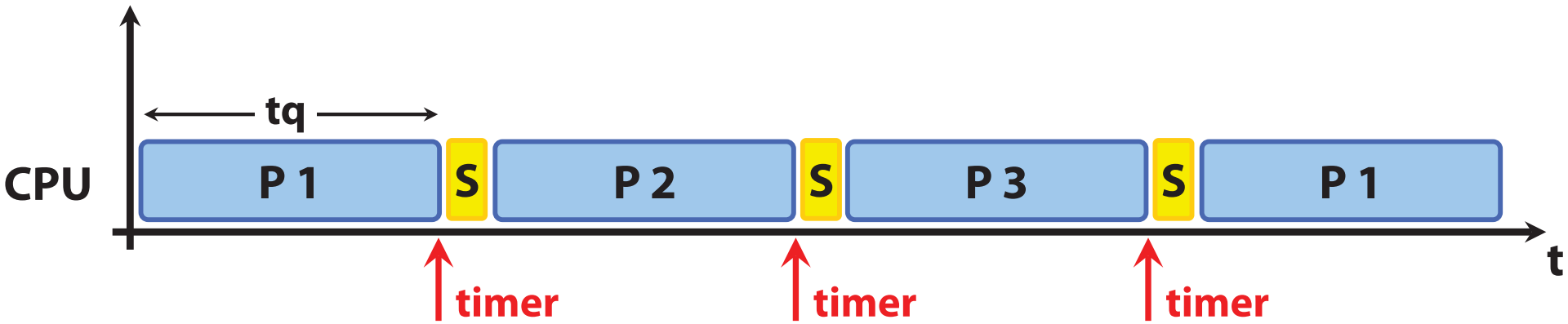


real-life Linux PCB in C

Διεργασίες - Σύνοψη

- Οργάνωση ενός σύγχρονου ΛΣ
  - Διακοπές, προνομιούχος κατάσταση
  - Κλήσεις συστήματος
- Διεργασία – Process
  - Ορισμός, μεταβάσεις κατάστασης – κύκλος ζωής
- Πίνακας Ελέγχου Διεργασίας
- Χρονοδρομολόγηση σε συστήματα καταμερισμού χρόνου
- Λειτουργίες διεργασιών
  - Δημιουργία διεργασιών - το μοντέλο του UNIX
- Διαδιεργασιακή Επικοινωνία

Καταμερισμός Χρόνου: η γενική ιδέα

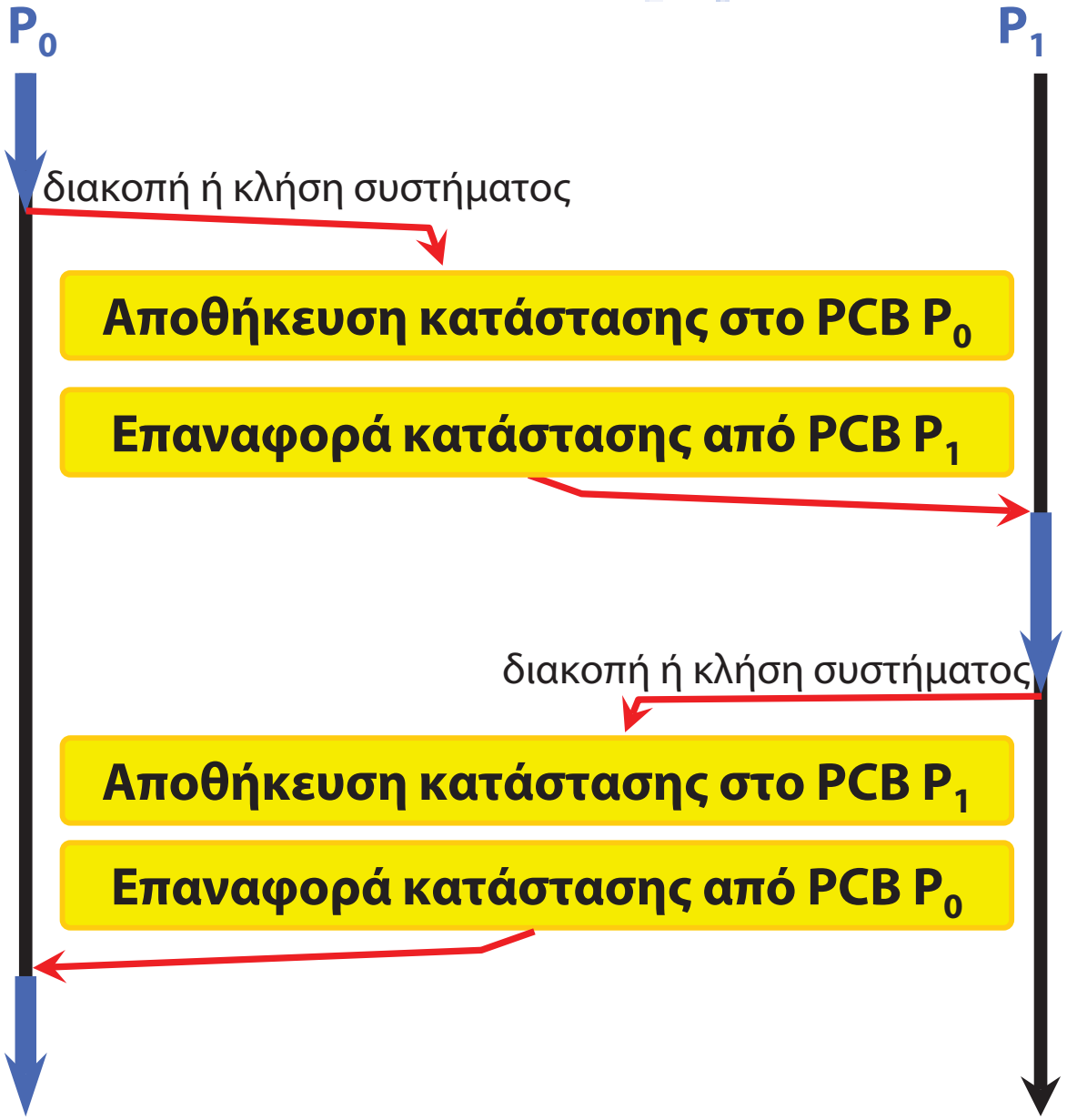


- Ο υπολογιστικός χρόνος κατανέμεται ανάμεσα στις διεργασίες που είναι έτοιμες να τρέξουν (P1, P2, P3)
  - Κάθε διεργασία τρέχει για χρόνο <= του *κβάντου χρόνου* (time quantum)
- Το χρονοδρομολογητή ενεργοποιούν *διακοπές χρονιστή* (timer interrupts)

Εναλλαγή Περιβάλλοντος Λειτουργίας (1)

- Context Switch: Η CPU αλλάζει από διεργασία σε διεργασία
  - Αποθήκευση κατάστασης παλιάς (στο PCB της)
  - Επαναφορά νέας διεργασίας (από το PCB της)
  - Πόσο διαρκεί;
    - Είναι χαμένος χρόνος για τη CPU
  - Πόσο συχνά;
    - το σύστημα πρέπει να είναι αποκρίσιμο

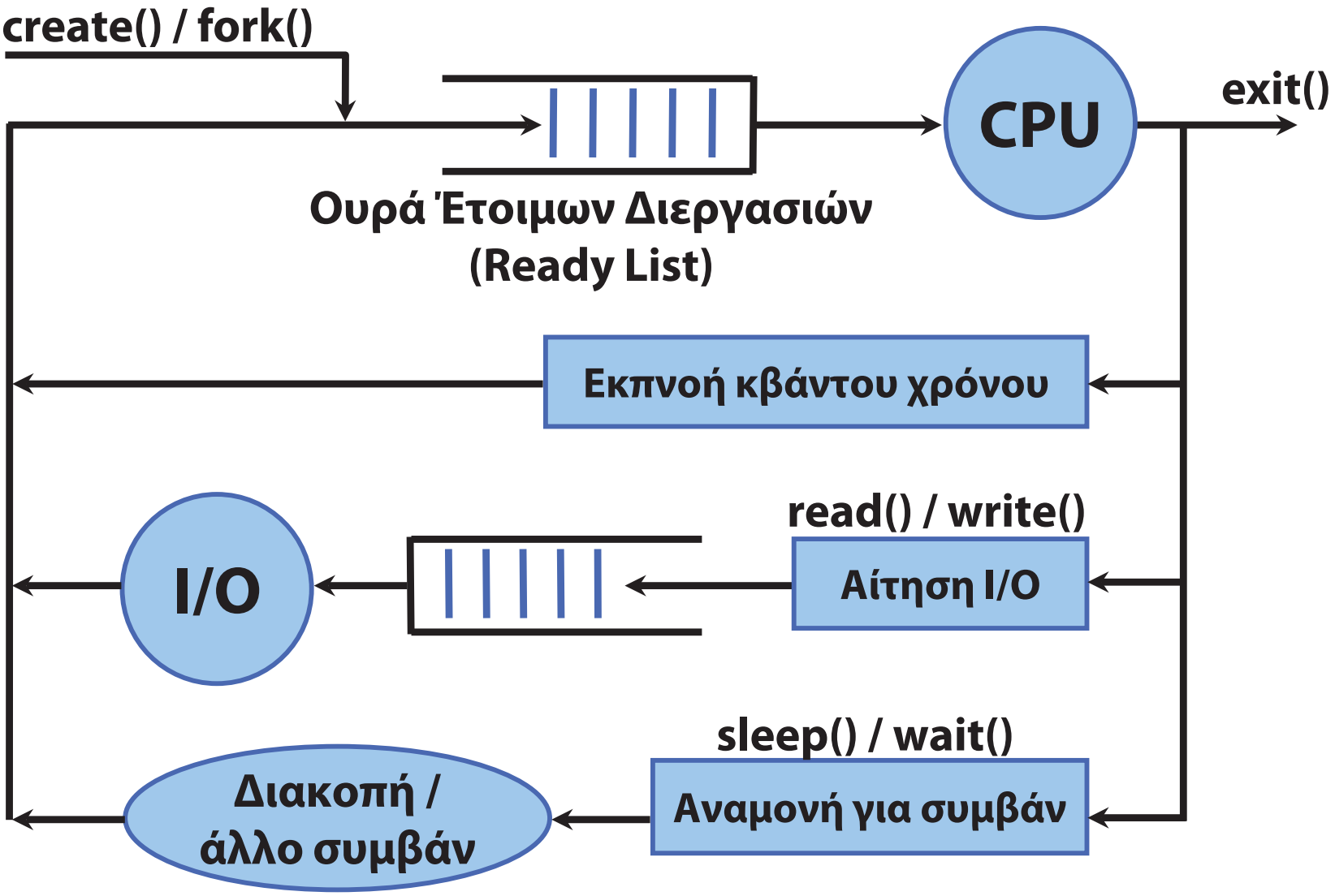
Εναλλαγή Περιβάλλοντος Λειτουργίας (2)



- Τι ποσοστό του χρόνου χάνεται στο context switch;

interrupt User Kernel

Κύκλος Ζωής μιας Διεργασίας (2)



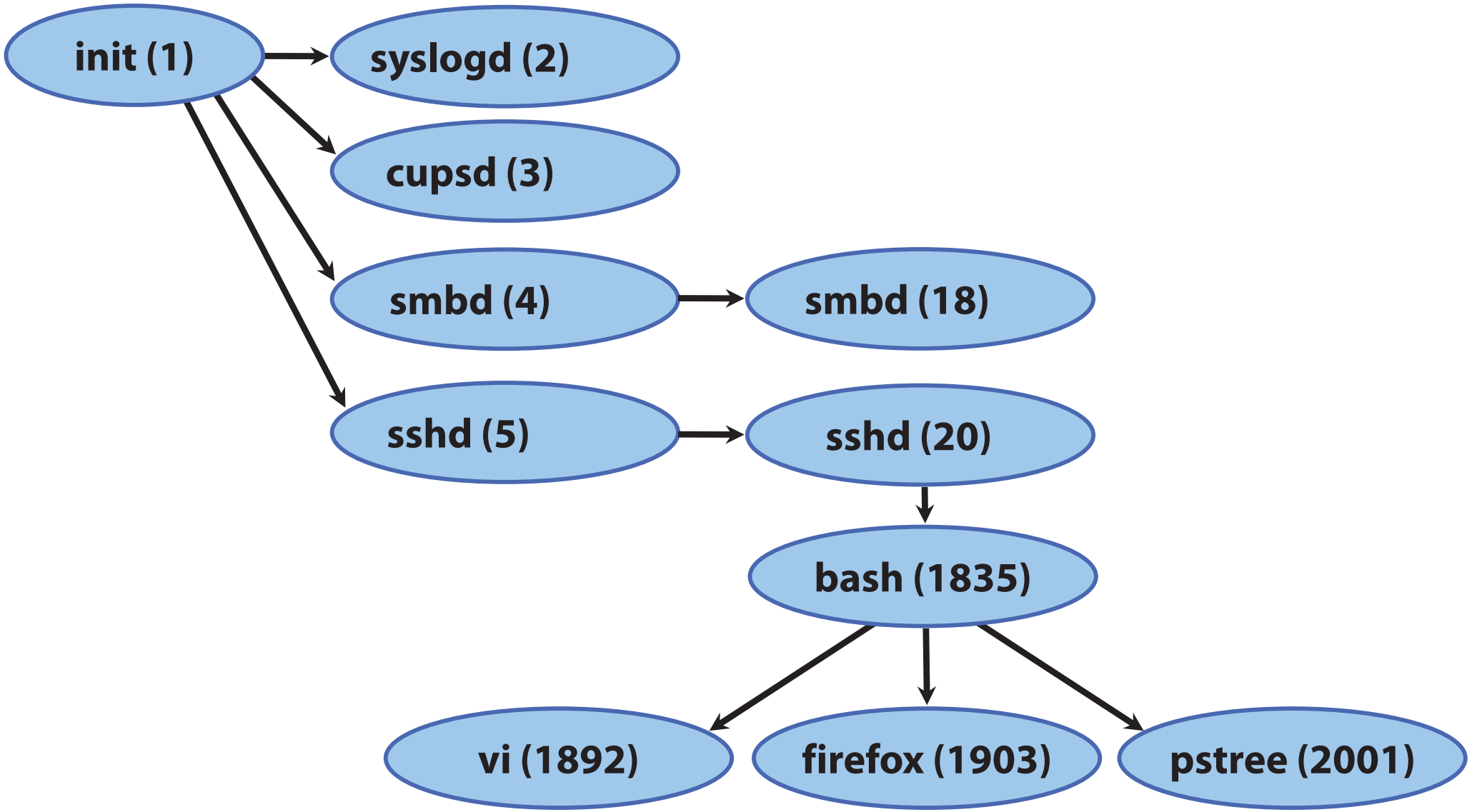
Διεργασίες - Σύνοψη

- Οργάνωση ενός σύγχρονου ΛΣ
  - Διακοπές, προνομιούχος κατάσταση
  - Κλήσεις συστήματος
- Διεργασία – Process
  - Ορισμός, μεταβάσεις κατάστασης – κύκλος ζωής
- Πίνακας Ελέγχου Διεργασίας
- Χρονοδρομολόγηση σε συστήματα καταμερισμού χρόνου
- Λειτουργίες διεργασιών
  - Δημιουργία διεργασιών - το μοντέλο του UNIX
- Διαδιεργασιακή Επικοινωνία

Λειτουργίες Διεργασιών

- Δημιουργία και καταστροφή διεργασιών
  - Δυναμική, κατά τη λειτουργία του συστήματος (**fork()**)
  - Μέσω κλήσεων συστήματος
- Ιεραρχία διεργασιών
  - Μια *γονική* διεργασία δημιουργεί διεργασίες-*παιδιά* (parent και children processes)
  - Προκύπτει ιεραρχική οργάνωση, σε *δέντρο* διεργασιών
- Τερματισμός διεργασίας
  - Οικειοθελής, όταν ολοκληρώσει το έργο της (**exit()**)
  - Βίαιος, λόγω εξωτερικού παράγοντα
    - Αποστολή σήματος (**kill()**) από άλλη διεργασία
    - τερματισμός από το λειτουργικό λόγω εσφαλμένης λειτουργίας ("This program has performed an illegal operation" στα Windows)
  - Ο γονέας ενημερώνεται (**wait()**)
    - Κι αν πεθάνει πριν από το παιδί του;

Δέντρο Διεργασιών στο Linux



Δημιουργία στο μοντέλο του UNIX: fork()

Δεδομένα :
p = -1 mypid = ?

Κείμενο:

```
pid_t p, mypid;
…
p = fork(); p = 987, errno =
if (p < 0) {  ENOMEM
        perror("fork");
        exit(1);
} else if (p == 0) {
        mypid = getpid();
        child();
} else {
        father();
}
```

PID=1981

Δεδομένα :
p = 0 mypid = 987

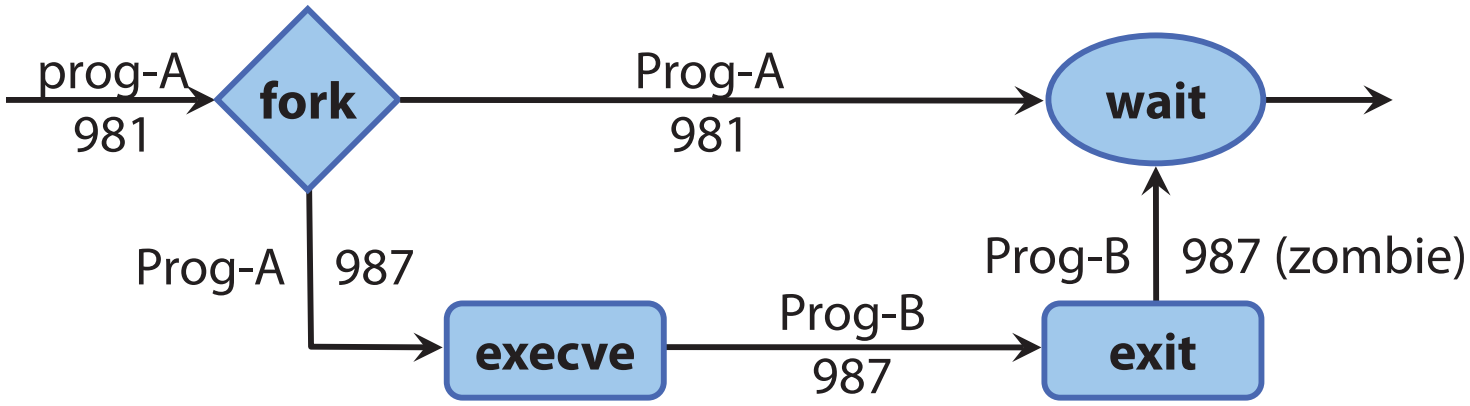
Κείμενο:

```
pid_t p, mypid;
…
p = fork(); p = 0
if (p < 0) {
        perror("fork");
        exit(1);
} else if (p == 0) {
        mypid = getpid();
        child();
} else {
        father();
}
```

PID=1987

Δημιουργία στο μοντέλο του UNIX: fork()

- Όλες οι διεργασίες προκύπτουν με fork() [*σχεδόν* όλες]
  - Ίδιο πρόγραμμα με γονική διεργασία, αντίγραφο χώρου μνήμης, κληρονομεί ανοιχτά αρχεία, συνδέσεις, δικαιώματα πρόσβασης
- Αντικατάσταση προγράμματος διεργασίας: **execve()**
- Η γονική διεργασία ενημερώνεται για το θάνατο του παιδιού με wait() → συλλογή τιμής τερματισμού (exit status)
  - Μέχρι τότε, παιδί που έχει καλέσει την exit() είναι *zombie*
  - Αν ο γονέας πεθάνει πρώτα, η διεργασία γίνεται παιδί της init (PID = 1), που κάνει συνεχώς wait()



Διεργασίες - Σύνοψη

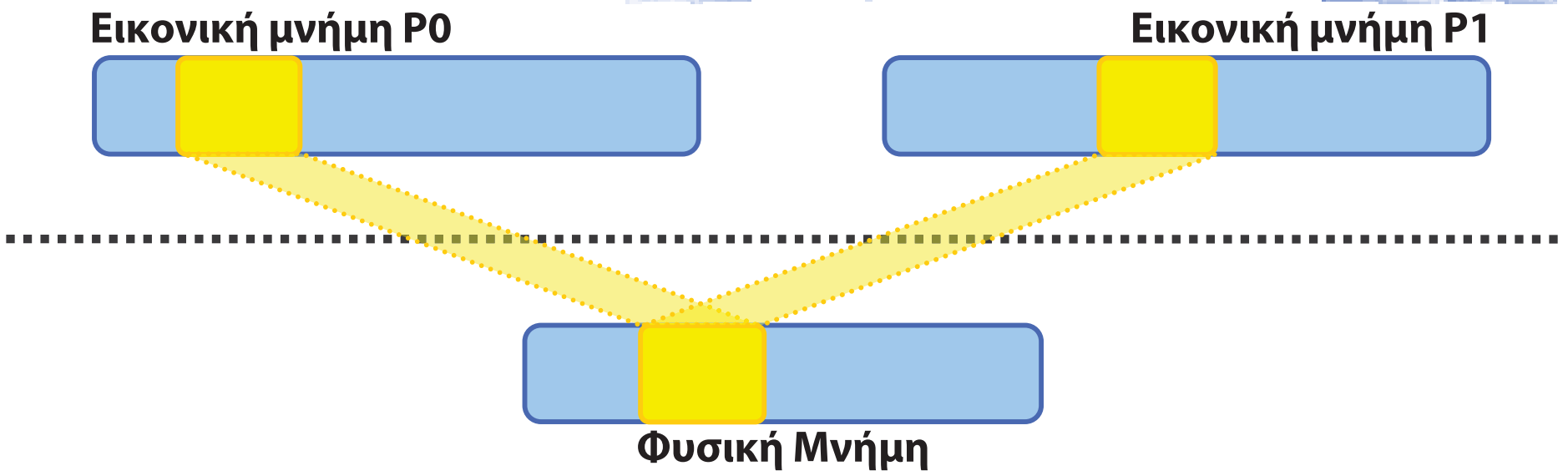
- Οργάνωση ενός σύγχρονου ΛΣ
  - Διακοπές, προνομιούχος κατάσταση
  - Κλήσεις συστήματος
- Διεργασία – Process
  - Ορισμός, μεταβάσεις κατάστασης – κύκλος ζωής
- Πίνακας Ελέγχου Διεργασίας
- Χρονοδρομολόγηση σε συστήματα καταμερισμού χρόνου
- Λειτουργίες διεργασιών
  - Δημιουργία διεργασιών - το μοντέλο του UNIX
- Διαδιεργασιακή Επικοινωνία

Διαδιεργασιακή Επικοινωνία (IPC)

- Inter-Process Communication (IPC)
- Διεργασίες που συνεργάζονται. Γιατί;
  - Παράλληλη εκτέλεση, επιτάχυνση
  - Δομημένη σχεδίαση, διαχωρισμός προνομίων
  - Ταυτόχρονη πρόσβαση σε κοινά δεδομένα
- Διαφορετικοί Μηχανισμοί από ΛΣ σε ΛΣ
  - Μοιραζόμενη Μνήμη (Shared Memory Segments)
  - Μηνύματα (Microkernels - Mach, POSIX messages)
  - Σήματα (UNIX)
  - Pipes (UNIX, Win32)

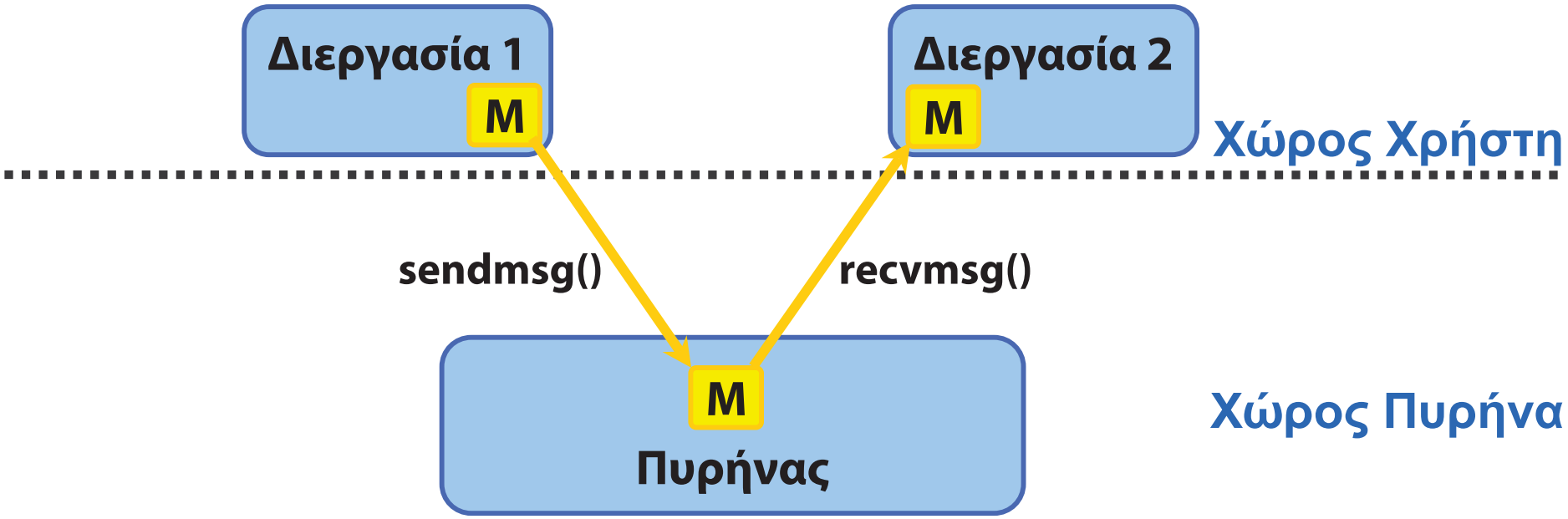
    Sockets

Μοιραζόμενη Μνήμη



- Πρόσβαση σε κοινό τμήμα μνήμης (shared memory segment)
- Για περισσότερες από μία διεργασίες
- Άμεση αλληλεπίδραση, με εντολές load/store
  - Χωρίς κλήσεις συστήματος
  - Ενημέρωση για αλλαγές; → έλεγχος (polling)
  - Πώς εξασφαλίζεται ότι δεν γράφουν στο ίδιο σημείο ταυτόχρονα;
- Το κοινό τμήμα *απεικονίζεται* στο χώρο εικονικής μνήμης των διεργασιών

Επικοινωνία με πέρασμα μηνυμάτων (1)



- Επικοινωνία χωρίς μοιραζόμενη μνήμη
  - μέσω κλήσεων συστήματος

Επικοινωνία με πέρασμα μηνυμάτων (2)

- Σχεδιαστικές αποφάσεις
  - Άμεση ή έμμεση επικοινωνία;
  - Σύγχρονη ή ασύγχρονη επικοινωνία;
  - Προσωρινή αποθήκευση μηνυμάτων σε απομονωτές (buffering)

Επικοινωνία με πέρασμα μηνυμάτων (2)

- Ονοματολογία
  - Άμεσα, συμμετρικά
    - send(P, message); receive(Q, message)
    - όπου P, Q είναι Process IDs
  - Άμεσα, ασύμμετρα
    - send(P, message); receive(id, message)
    - όπου το P δίνεται, το “id” τίθεται από την receive()
  - Έμμεσα, με χρήση *γραμματοθυρίδων* (mailboxes) / *θυρών* (ports)
    - send(A, message); receive(A, message)
    - όπου A το αναγνωριστικό της θυρίδας (π.χ. ένας ακέραιος)
    - δημιουργία και καταστροφή θυρίδων;

Επικοινωνία με πέρασμα μηνυμάτων (3)

- Συγχρονισμός
  - Αποστολές με ή χωρίς αναμονή (blocking / non-blocking send)
  - Λήψη με ή χωρίς αναμονή (blocking / non-blocking receive)
  - Σύγχρονος ή ασύγχρονος τρόπος λειτουργίας (rendez-vous)

- Προσωρινή αποθήκευση σε απομονωτές
  - Χωρίς προσωρινή αποθήκευση (σύγχρονα)
  - Πεπερασμένη χωρητικότητα
    - ο αποστολέας μπλοκάρει όταν ο απομονωτής γεμίσει
  - Απεριόριστη χωρητικότητα
    - με δυναμική δέσμευση χώρου

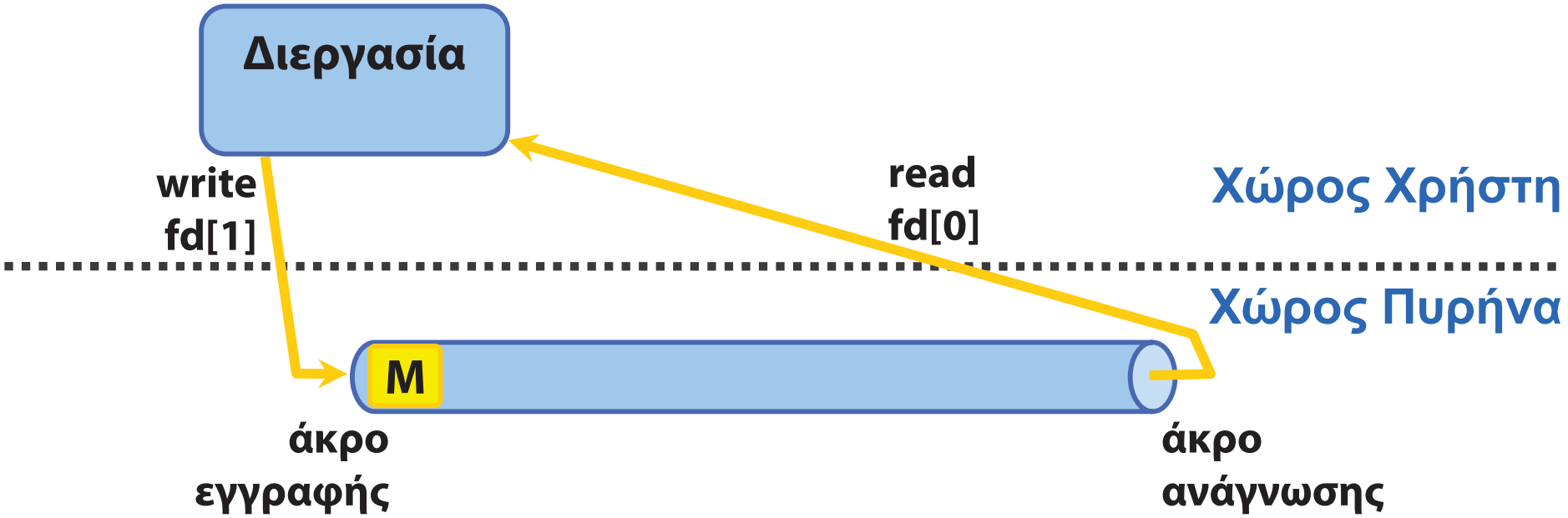
Σήματα στο UNIX (1)

- Ενσωματωμένα ήδη από τις πρώτες εκδόσεις του UNIX (Έκδοση 7)
- Ασύγχρονη Ειδοποίηση για Εξωτερικό Γεγονός
- Κάθε σήμα έχει ένα συμβολικό όνομα
  - SIGINT, SIGTSTP, SIGCONT, SIGSTOP, SIGFPE, SIGSEGV, SIGABRT, SIGQUIT, SIGWINCH, SIGTERM, SIGKILL
- Και διαφορετική σημασία, π.χ.
  - Διακοπή Πληκτρολογίου, Ctrl-C (SIGINT)
  - Αναστολή από το πληκτρολόγιο (SIGTSTP)
  - Τερματισμός (SIGTERM)
  - Βίαιος Τερματισμός (SIGKILL)
- Κάποια αποστέλλονται αυτόματα από το ΛΣ, κάποια από άλλη διεργασία (**kill()**)

Σήματα στο UNIX (2)

- Για εισερχόμενο σήμα, εκτελείται προκαθορισμένη ενέργεια (συνήθως τερματισμός)
  - Η wait() επιστρέφει ανάλογο status
- Το σήμα μπορεί να συλληφθεί, οπότε εκτελείται συνάρτηση *χειρισμού* του (**signal()**, **sigaction()**)
  - π.χ., όταν πατηθεί Ctrl-C, σώσε το ανοιχτό αρχείο
- Τα SIGSTOP και SIGKILL *δεν* πιάνονται
- Αναξιόπιστος μηχανισμός επικοινωνίας
  - Ασύγχρονος, με *race conditions*
- Το POSIX προβλέπει αξιόπιστα σήματα (**sigprocmask()**, **sigsuspend()**)

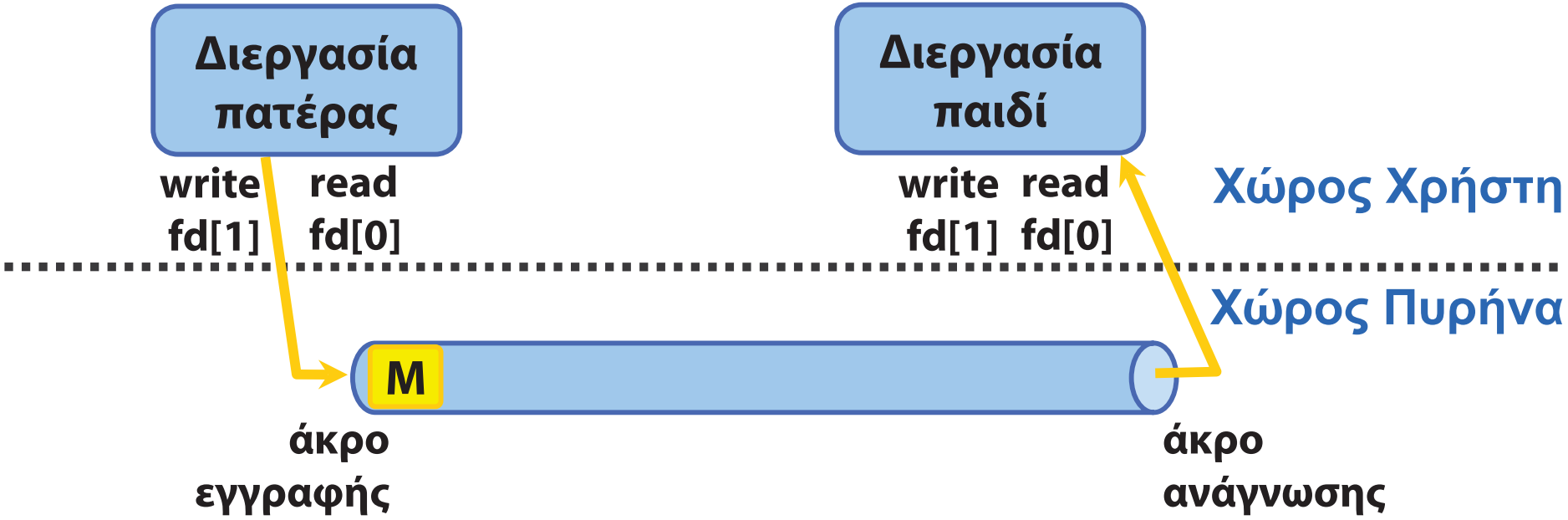
Σωληνώσεις στο UNIX (1)



- Ένας από τους βασικότερους μηχανισμούς στο UNIX
- Μονόδρομη μεταφορά δεδομένων
- Από το άκρο *εγγραφής* στο άκρο *ανάγνωσης*
  - Δημιουργία με **pipe()**, επικοινωνία με **write()** και **read()**
  - Αν η σωλήνωση είναι άδεια; → η read() *μπλοκάρει*

```
int fd[2];
int num1, num2;
pipe(fd);
write(fd[1], &num1, sizeof(num1));
read(fd[0], &num2, sizeof(num2));
```

Σωληνώσεις στο UNIX (2)



- pipe(fd);
- fork();
- … ο πατέρας κλείνει το άκρο ανάγνωσης
- …το παιδί κλείνει το άκρο εγγραφής

Νήματα - Σύνοψη

- Ορισμός νημάτων, σχέση με διεργασίες
- Πλεονεκτήματα πολυνηματισμού
- Μοντέλα πολυνηματισμού
  - Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- Βιβλιοθήκες και APIs πολυνηματισμού
  - POSIX Threads, Win32, Java Threads

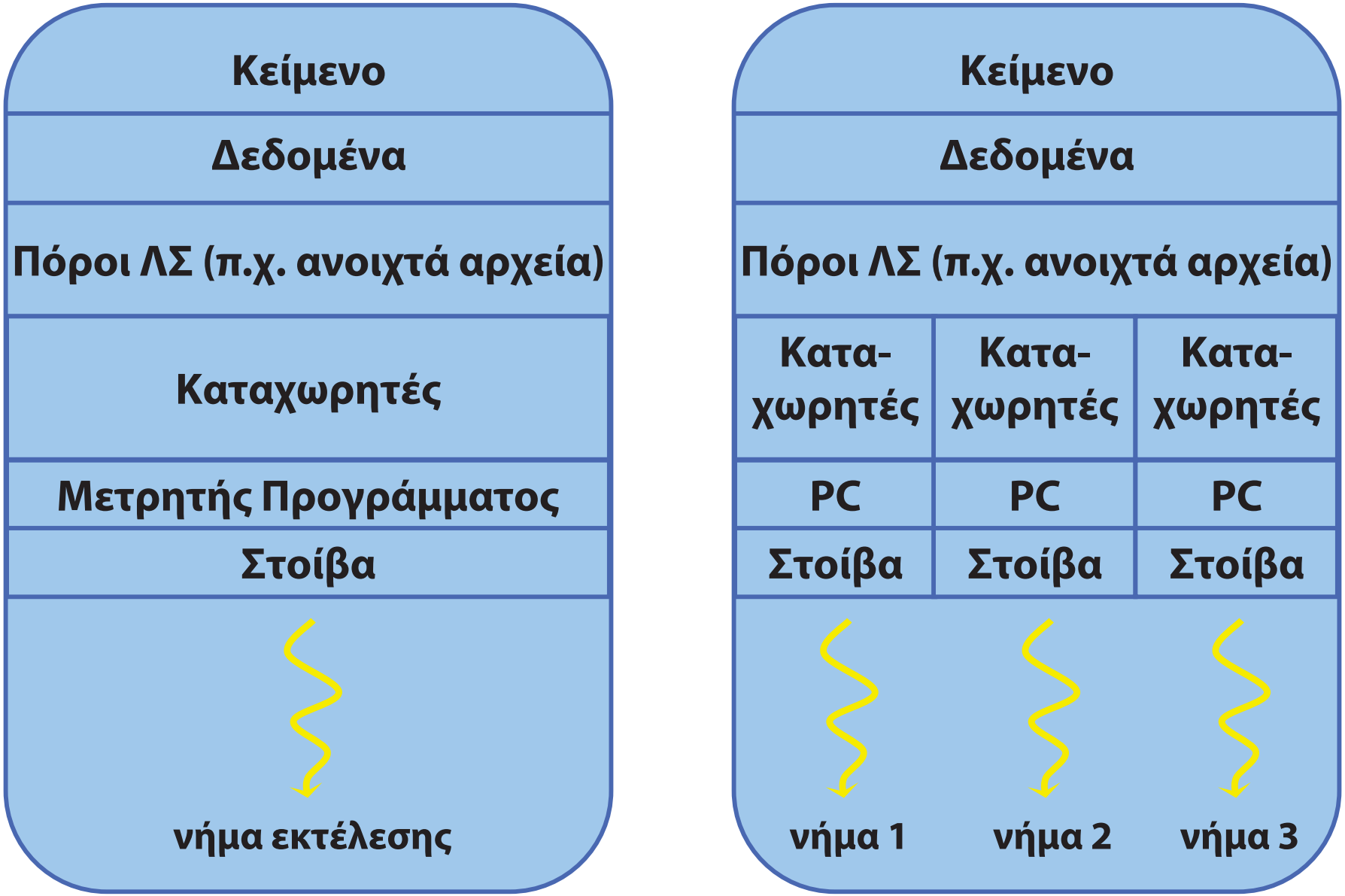
Νήματα - Σύνοψη

- Ορισμός νημάτων, σχέση με διεργασίες
- Πλεονεκτήματα πολυνηματισμού
- Μοντέλα πολυνηματισμού
  - Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- Βιβλιοθήκες και APIs πολυνηματισμού
  - POSIX Threads, Win32, Java Threads

Νήματα – Threads

- Νήματα: χωριστές ροές εκτέλεσης μέσα στην ίδια διεργασία
  - Κατάσταση διεργασίας = αρχιτεκτονική κατάσταση + πόροι ΛΣ
  - Χωριστή αρχιτεκτονική κατάσταση για κάθε νήμα
    - Μετρητής Προγράμματος, Καταχωρητές CPU, Στοίβα
  - Όλα τα νήματα ζουν μέσα στην ίδια διεργασία
    - Μοιράζονται τους πόρους του ΛΣ
    - Κοινή μνήμη, κοινά ανοιχτά αρχεία, κοινά δικαιώματα
    - Άμεση πρόσβαση σε κοινά δεδομένα

Πολυνηματική Διεργασία



Νήματα - Σύνοψη

- Ορισμός νημάτων, σχέση με διεργασίες
- Πλεονεκτήματα πολυνηματισμού
- Μοντέλα πολυνηματισμού
  - Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- Βιβλιοθήκες και APIs πολυνηματισμού
  - POSIX Threads, Win32, Java Threads

Νήματα – Γιατί;

- Αποκρισιμότητα
  - Διαφορετικά νήματα κάνουν διαφορετικά πράγματα, π.χ. νήμα αλληλεπίδρασης με χρήστη και νήματα υπολογισμού
- Οικονομία πόρων
  - Μικρότερο κόστος δημιουργίας/καταστροφής νημάτων
  - Ένα νήμα θέλει λιγότερους πόρους ΛΣ από μια διεργασία
- Παράλληλος υπολογισμός
  - Μικρό κόστος επικοινωνίας ανάμεσα σε νήματα
  - Χρονοδρομολόγηση νημάτων προσαρμοσμένη στην εφαρμογή
- Ευκολότερος προγραμματισμός
  - Πολυνηματικοί εξυπηρετητές
    - Κάθε νήμα έχει τη δική του κατάσταση → νήμα ανά πελάτη
    - *Ασύγχρονη* λειτουργία με *σύγχρονες* κλήσεις συστήματος
    - Κάθε νήμα αναστέλλεται ανεξάρτητα από τα υπόλοιπα

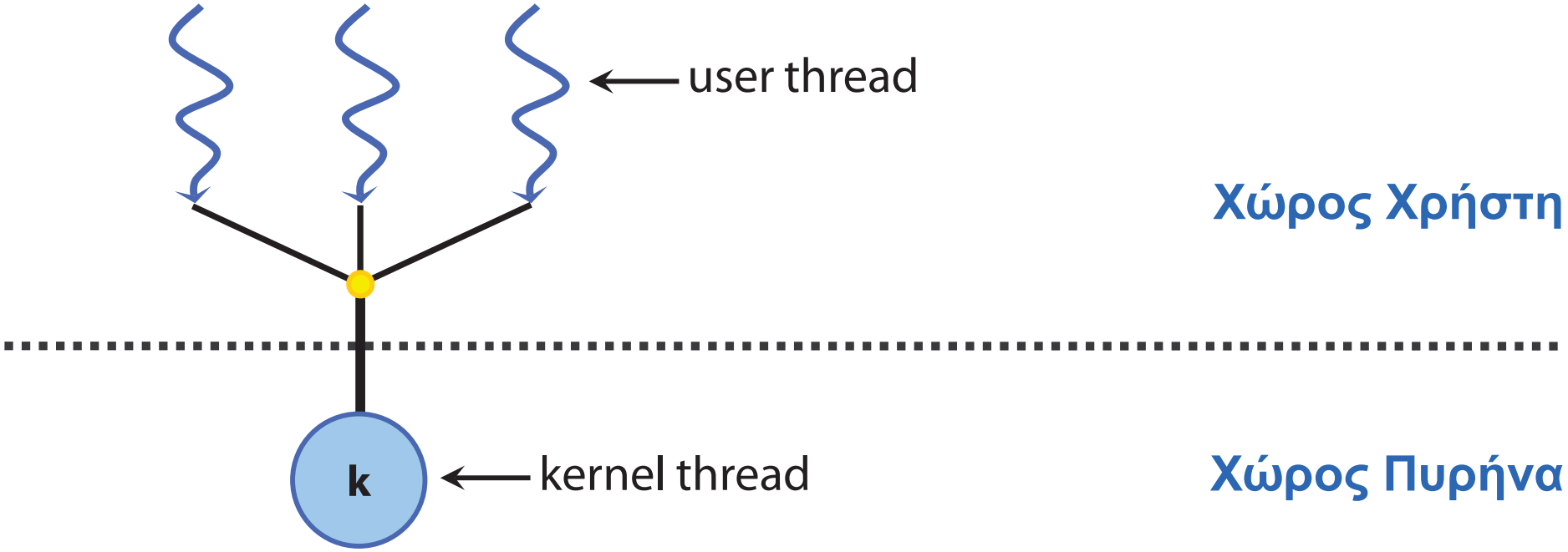
Νήματα - Σύνοψη

- Ορισμός νημάτων, σχέση με διεργασίες
- Πλεονεκτήματα πολυνηματισμού
- Μοντέλα πολυνηματισμού
  - Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- Βιβλιοθήκες και APIs πολυνηματισμού
  - POSIX Threads, Win32, Java Threads

Μοντέλα Πολυνηματισμού

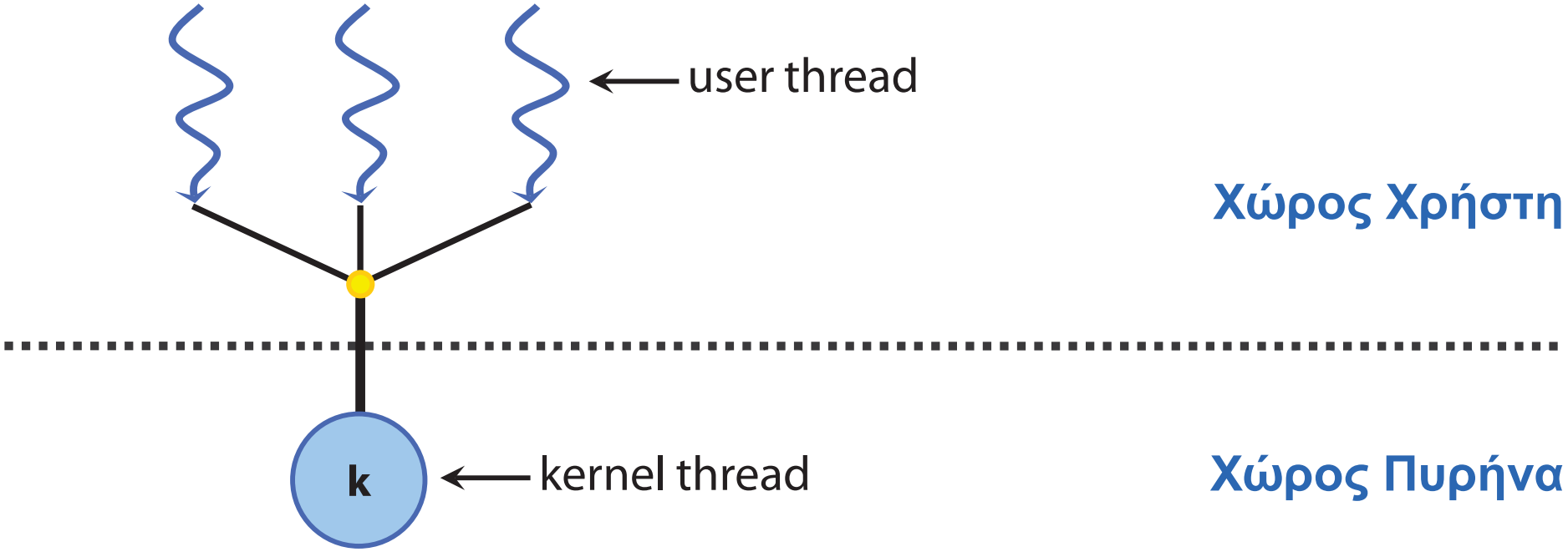
- Προγραμματιστικά μοντέλα και APIs για πολυνηματισμό
  - POSIX Threads, Win32 API, Java threads, OpenMP
- Πώς υλοποιείται η υποστήριξη νημάτων;
  - ➡ Στο χώρο πυρήνα, στο χώρο χρήστη, με συνεργασία και των δύο;
  - ➡ Νήματα χώρου χρήστη – νήματα χώρου πυρήνα
    - Ο χρονοδρομολογητής του ΛΣ βλέπει μόνο νήματα πυρήνα
  - ➡ Πλεονεκτήματα / μειονεκτήματα της κάθε προσέγγισης;

N:1 - Νήματα σε επίπεδο χρήστη (1)



- Δημιουργία νημάτων σε επίπεδο χρήστη
  - Με φορητό τρόπο, πχ. swapcontext(), setjmp/longjmp()
  - Ο πυρήνας δεν γνωρίζει τίποτε για αυτά
  - Η χρονοδρομολόγηση γίνεται από κώδικα χρήστη
    - Είτε με διακοπτό (SIGALRM) είτε συνεργατικά, με μη-διακοπτό τρόπο

N:1 - Νήματα σε επίπεδο χρήστη (2)

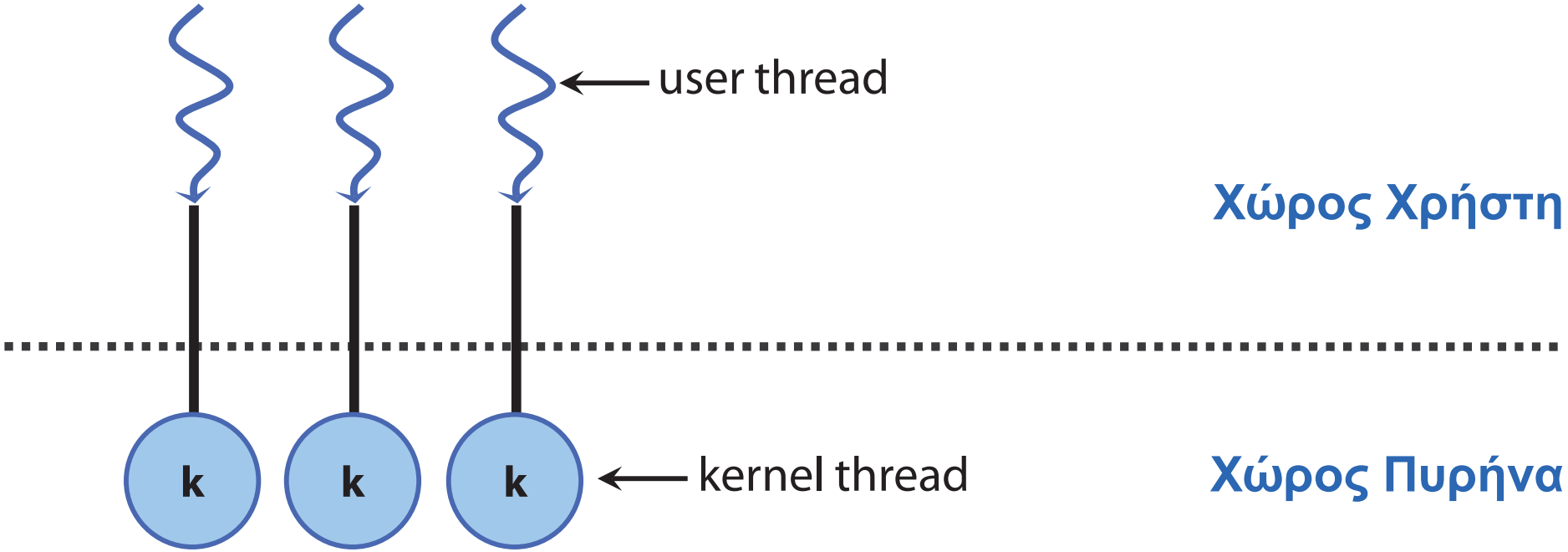


- Μικρό κόστος διαχείρισης νημάτων
- Χρονοδρομολόγηση προσαρμοσμένη στην εφαρμογή
- Τι γίνεται με κλήσεις συστήματος που προκαλούν αναστολή της εκτέλεσης;
  - ➡ Αν ένα νήμα μπλοκάρει σε read(), όλη η διεργασία μπλοκάρει

Παράδειγμα υλοποίησης N:1 - GNU Pth

- Πολυνηματική βιβλιοθήκη σε επίπεδο χρήστη, με υποστήριξη και του Pthreads API
- Μη-διακοπτός καταμερισμός χρόνου (non-preemptive multitasking)
- Δεν χρειάζεται υποστήριξη από τον πυρήνα
  - Διαχείριση νημάτων με κλήσεις POSIX
    - makecontext() / swapcontext(), setjmp()/longjmp()
- Φορητότητα, μικρό κόστος διαχείρισης νημάτων
- Προσαρμοσμένη σε πολυνηματικούς εξυπηρετητές, όχι παράλληλο υπολογισμό
- Προβλήματα: διαχείριση κλήσεων συστήματος, υποστήριξη πολυπεξεργαστών;
  - "On the other hand, it cannot benefit from the existence of multiprocessors, because for this, kernel support would be needed. In practice, this is no problem, because multiprocessor systems are rare " [GNU Pth manual]

1:1 - Νήματα σε επίπεδο πυρήνα

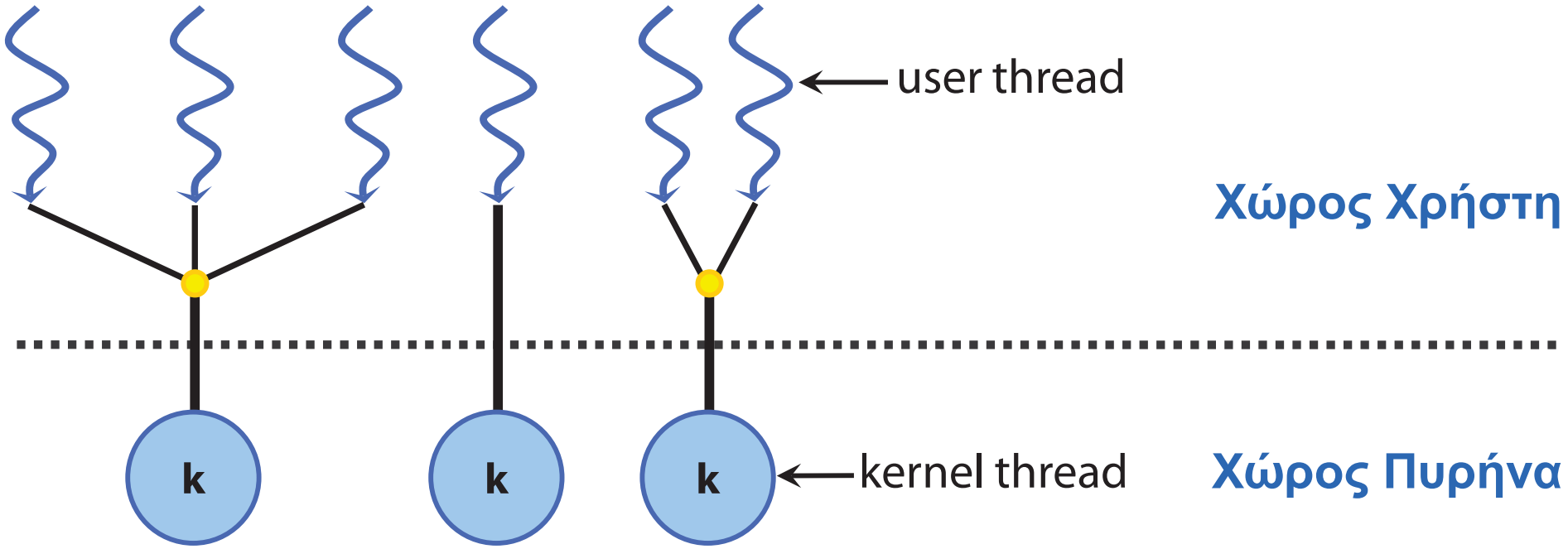


- Υποστήριξη πολυεπεξεργαστών, χωριστά νήματα σε χωριστούς πυρήνες πολυπύρηνων επεξεργαστών
- Ακριβότερη διαχείριση νημάτων
- Απλούστερη υλοποίηση
- Linux NPTL, Win32 σε Windows NT/XP/2k, Solaris 9+

Παράδειγμα υλοποίησης 1:1 – Linux NPTL

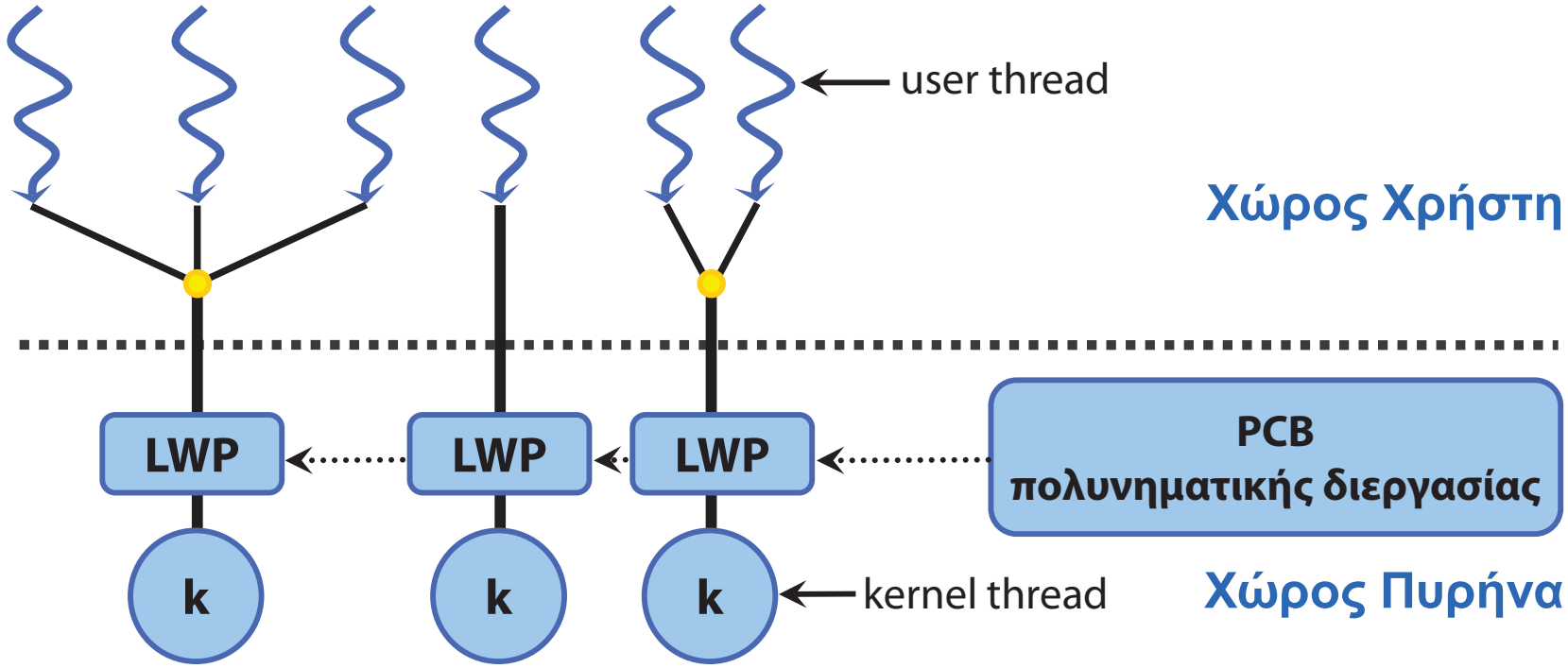
- Linux Native POSIX Threads Library
  - Υλοποίηση της POSIX 1003.1c σε επίπεδο πυρήνα
  - Βασισμένη στο μηχανισμό συγχρονισμού *futex* και την κλήση συστήματος **clone()**
  - Διάφορα clone() flags: CLONE_FILES, CLONE_SIGHAND, CLONE_VM
  - Η fork() υλοποιείται ως clone()
  - Ο πυρήνας βλέπει *tasks*, που μοιράζονται σχεδόν τα πάντα (νήματα) ή τίποτε (διεργασίες)

M:N – Συνδυασμός νημάτων χρήστη/πυρήνα



- Ένας αριθμός από νήματα χρήστη τρέχει πάνω σε περισσότερα του ενός νήματα πυρήνα
- Διεπίπεδη χρονοδρομολόγηση
  - ➡ Πολύπλοκη υλοποίηση
- Solaris έκδοσης < 9

Παράδειγμα υλοποίησης M:N – Solaris



- Μια πολυνηματική διεργασία αποτελείται από πολλές *ελαφρές διεργασίες* – LightWeight Processes (LWPs)
- Ο χρονοδρομολογητής χώρου χρήστη βλέπει εικονικούς επεξεργαστές
- Ο χρονοδρομολογητής χώρου πυρήνα βλέπει νήματα πυρήνα
- Ανάγκη για *upcalls*: Ο πυρήνας ενημερώνει το χώρο χρήστη όταν ένα LWP πρόκειται να μπλοκάρει → εκτέλεση έτοιμου νήματος σε νέο LWP

Νήματα - Σύνοψη

- Ορισμός νημάτων, σχέση με διεργασίες
- Πλεονεκτήματα πολυνηματισμού
- Μοντέλα πολυνηματισμού
  - ➡ Υλοποίηση σε χώρο χρήστη, πυρήνα, συνδυασμού τους
- Βιβλιοθήκες και APIs πολυνηματισμού
  - ➡ POSIX Threads, Win32, Java Threads

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()**