Προγραμματισμός ΙΙ (ECE116)

#11 το λειτουργικό σύστημα

Σπύρος Λάλης
lalis@uth.gr

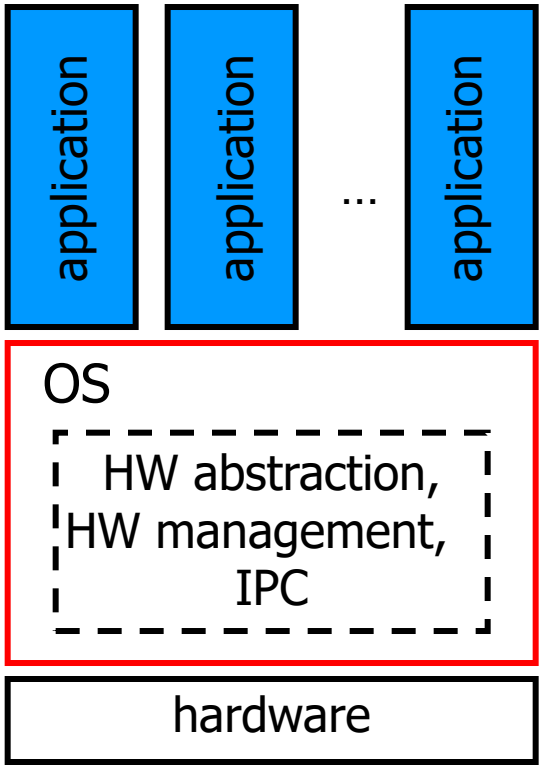
Συστήματα υπολογιστών

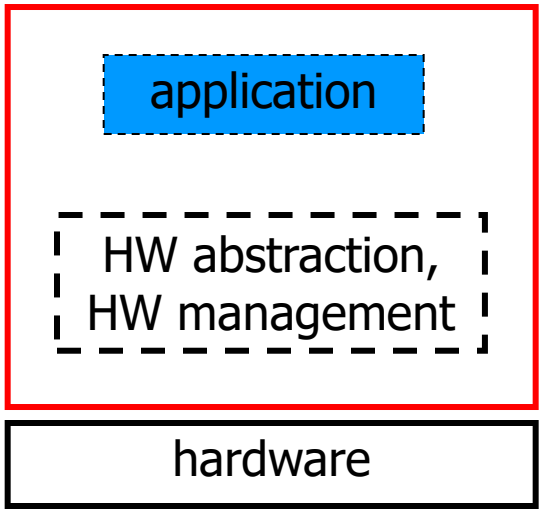
- Ειδικού σκοπού
  - συστήματα για μια ή περισσότερες συγκεκριμένες εφαρμογές
  - οι εφαρμογές είναι γνωστές εκ των προτέρων
  - περιορισμένοι υπολογιστικοί / αποθηκευτικοί πόροι
  - δεν τίθεται θέμα υποστήριξης πολλών χρηστών

- Γενικής χρήστης
  - υπάρχουν πολλές (ανεξάρτητες) εφαρμογές
    που μπορεί να εκτελούνται ταυτόχρονα μεταξύ τους
  - μπορεί να δίνεται πρόσβαση σε πολλούς χρήστες
    που (γενικά) δεν εμπιστεύονται ο ένας τον άλλο
  - ο αριθμός και το είδος των εφαρμογών είναι άγνωστος (ανοιχτός)
    και μπορεί να αλλάξει δυναμικά, ακόμα και την ώρα της εκτελεσης

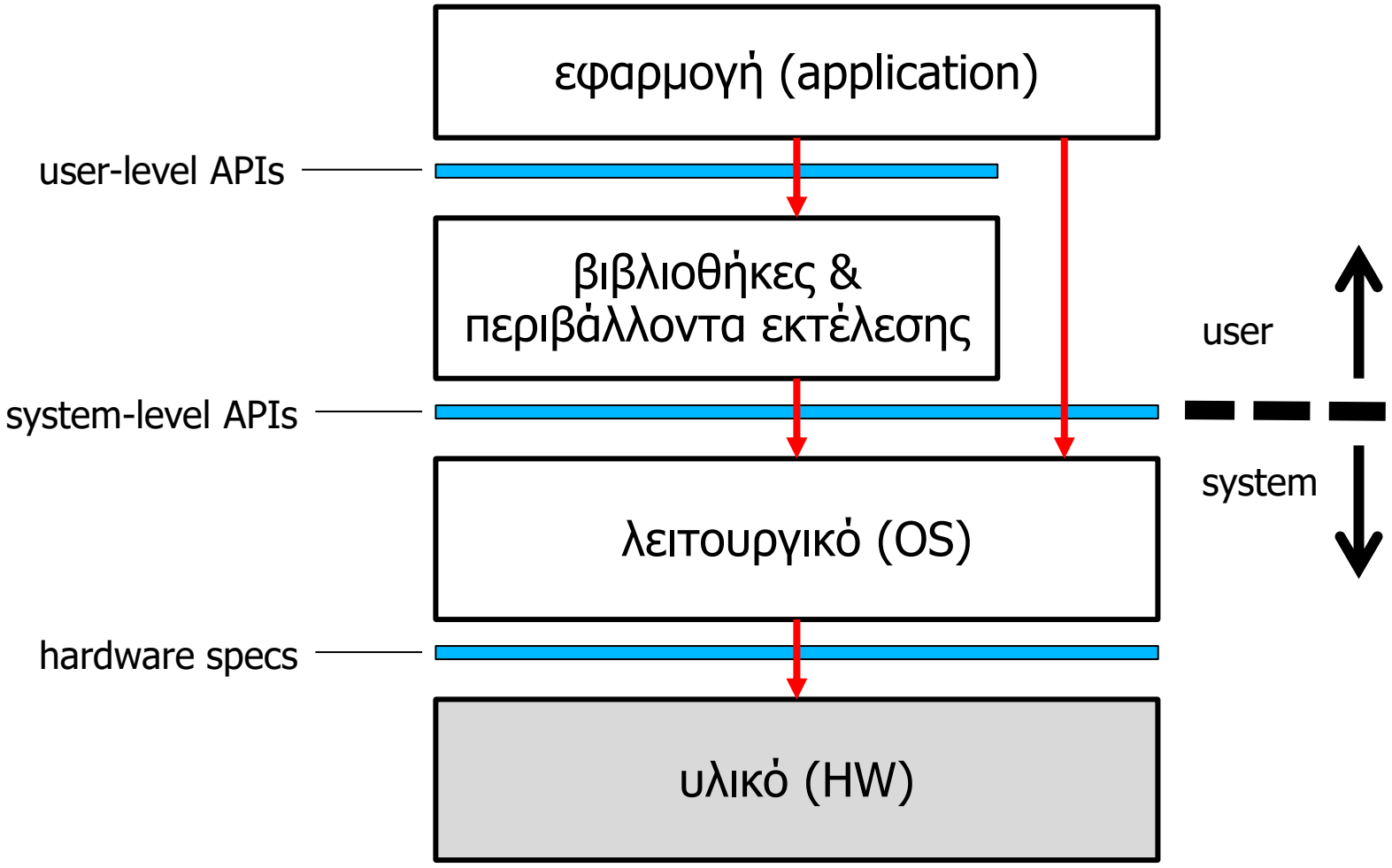
συστήματα γενικής χρήσης

συστήματα ειδικού σκοπού





Ενδεικτική στοίβα λογισμικού ενός Η/Υ



Βασικοί ρόλοι και μηχανισμοί ΛΣ

- Αφαίρεση (abstraction) του υλικού
  - απόκρυψη λεπτομερειών υλοποίησης, ανεξαρτησία από υλικό
- Διαχείριση πόρων και συσκευών του συστήματος
  - επεξεργαστής, μνήμη, δίσκος, δίκτυο, λοιπά περιφερειακά

- Υποστήριξη εκτέλεσης προγραμμάτων εφαρμογής
- Υποστήριξη επικοινωνίας προγραμμάτων
- Υποστήριξη αποθήκευσης δεδομένων

- Κατανομή των πόρων του συστήματος
- Ελεγχόμενη πρόσβαση σε πόρους του συστήματος

Διαχωρισμός σύστημα - χρήστης

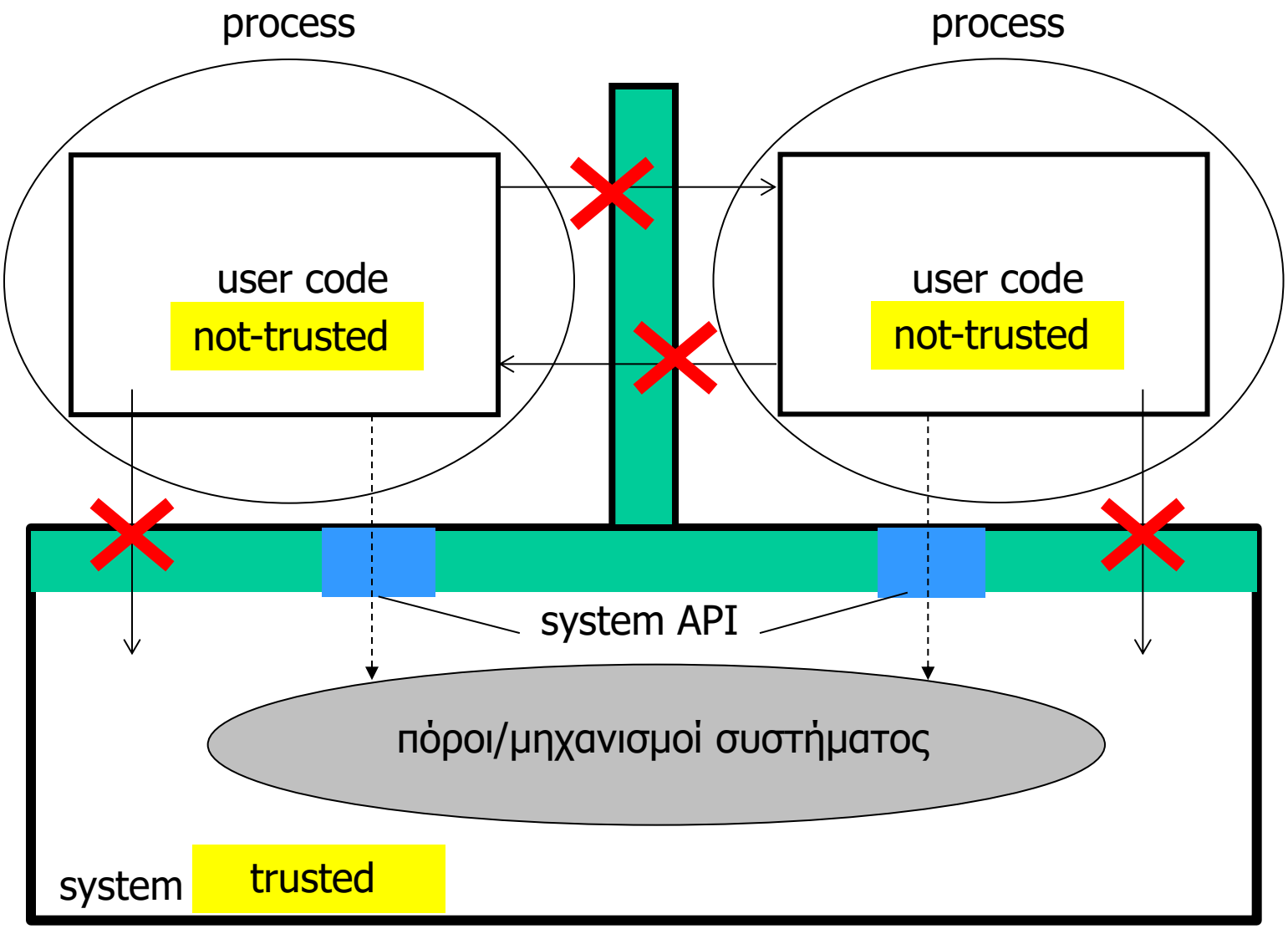
- Η ασφάλεια στους Η/Υ βασίζεται στον **διαχωρισμό** μεταξύ του κώδικα του συστήματος / χρήστη
- Κώδικας **συστήματος**
  - θεωρείται **αξιόπιστος** (χωρίς αυτό να ισχύει πάντα)
  - υλοποιεί τις βασικές λειτουργίες του συστήματος
  - γράφεται από εξειδικευμένους/έμπιστους προγραμματιστές
  - αλλάζει σπάνια και με ελεγχόμενο τρόπο
- Κώδικας **χρήστη**
  - θεωρείται **αναξιόπιστος** – δυνητικά «κακόβουλος»
  - υλοποιεί λειτουργικότητα της εφαρμογής
  - μπορεί να γραφτεί από οποιονδήποτε
  - μπορεί να έχει πολλά και διάφορα λάθη
  - αλλάζει συχνά και χωρίς κανέναν έλεγχο

Ελεγχόμενη πρόσβαση

- Χρειάζονται μηχανισμοί **ελέγχου/ασφάλειας**
  - για την **προστασία του λειτουργικού** από τα προγράμματα / εφαρμογές που εκτελούνται πάνω από αυτό
  - για την **προστασία των εφαρμογών** μεταξύ τους

- Οι αλληλεπιδράσεις ανάμεσα στις εφαρμογές και το λειτουργικό σύστημα γίνονται με **ασφαλή τρόπο**

- Μέσω της **διεπαφής/διασύνδεσης συστήματος** (system interface)
- Αποτελείται από τις λεγόμενες **κλήσεις συστήματος**

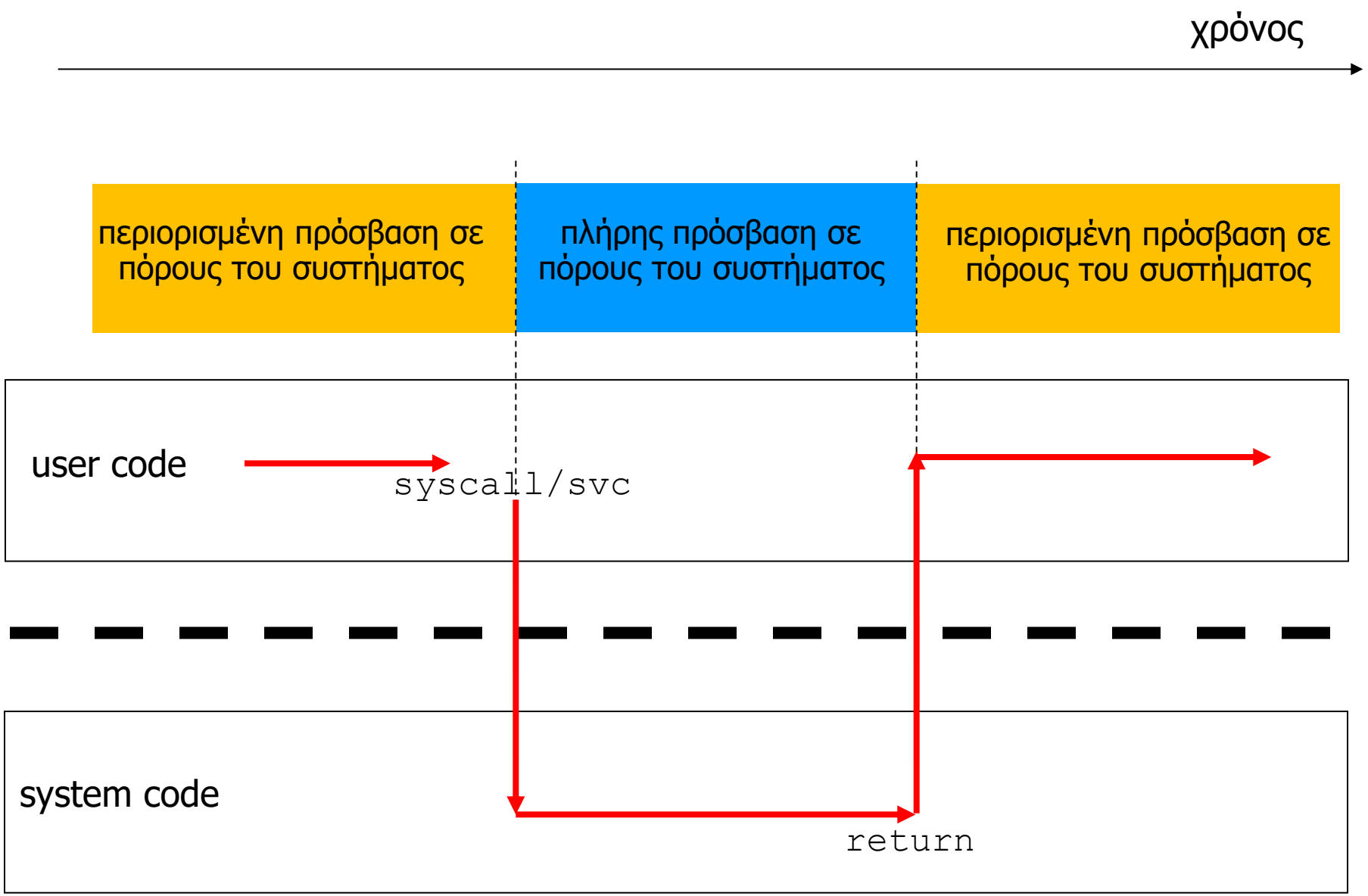

process
process
user code
not-trusted
user code
not-trusted
system API
πόροι/μηχανισμοί συστήματος
system
trusted

Κλήσεις συστήματος

- **Ειδικές** κλήσεις συνάρτησης
  1. Η κατάσταση του επεξεργαστή αλλάζει από user σε system mode
  2. Άλμα στον κώδικα του λειτουργικού
  3. Εκτέλεση κώδικα λειτουργικού
  4. Επιστροφή στον κώδικα του χρήστη
  5. Η κατάσταση του επεξεργαστή επανέρχεται σε user mode

- Η πρόσβαση στις εσωτερικές λειτουργίες και τους πόρους του ΛΣ επιτρέπεται **μόνο** σε system mode

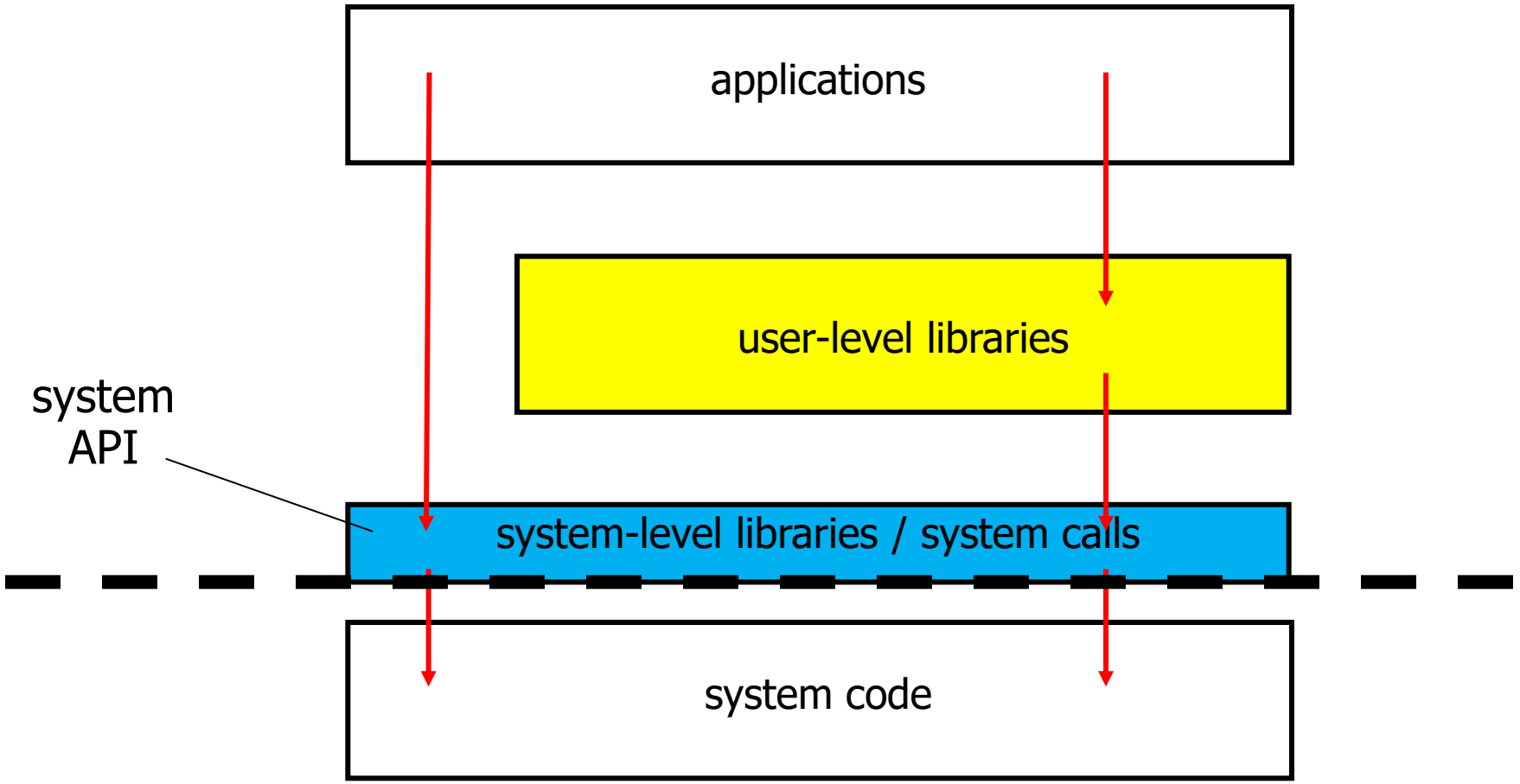
- Το άλμα στον κώδικα του λειτουργικού με αλλαγή από user σε system mode γίνεται μέσω **ειδικής εντολής** του επεξεργαστή: supervisor call – svc

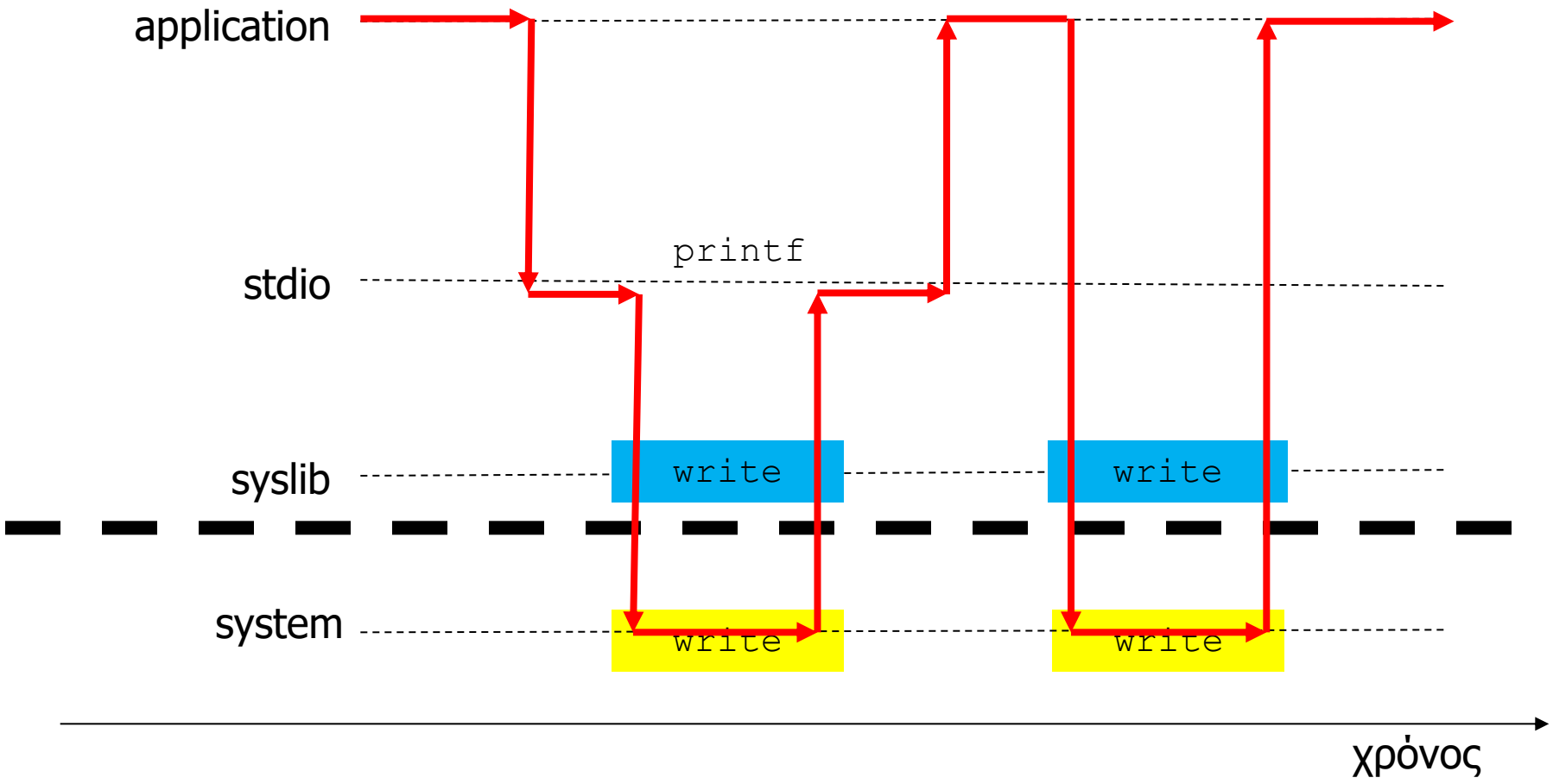

χρόνος
περιορισμένη πρόσβαση σε πόρους του συστήματος
πλήρης πρόσβαση σε πόρους του συστήματος
περιορισμένη πρόσβαση σε πόρους του συστήματος
user code
syscall/svc
system code
return

Απόκρυψη κλήσεων συστήματος

- Οι συμβάσεις για τις κλήσεις συστήματος μπορεί να είναι αρκετά περίπλοκες
  - το πέρασμα παραμέτρων / η επιστροφή των αποτελεσμάτων μπορεί να εξαρτάται από την αρχιτεκτονική του επεξεργαστή
  - το άλμα στον κώδικα του λειτουργικού γίνεται μέσω ειδικής εντολής του επεξεργαστή
  - εύκολα γίνονται προγραμματιστικά λάθη

- Οι κλήσεις συστήματος συνήθως «περιτυλίγονται» ώστε να έχουν την μορφή οικείων συναρτήσεων, π.χ., με βάση την σύνταξη/συμβάσεις της C
  - τα περιτυλίγματα ορίζονται σε ειδικές βιβλιοθήκες
  - system-level libraries (glibc – GNU C library)


applications
user-level libraries
system
API
system-level libraries / system calls
system code


application
stdio
printf
syslib
write
write
system
write
write
χρόνος

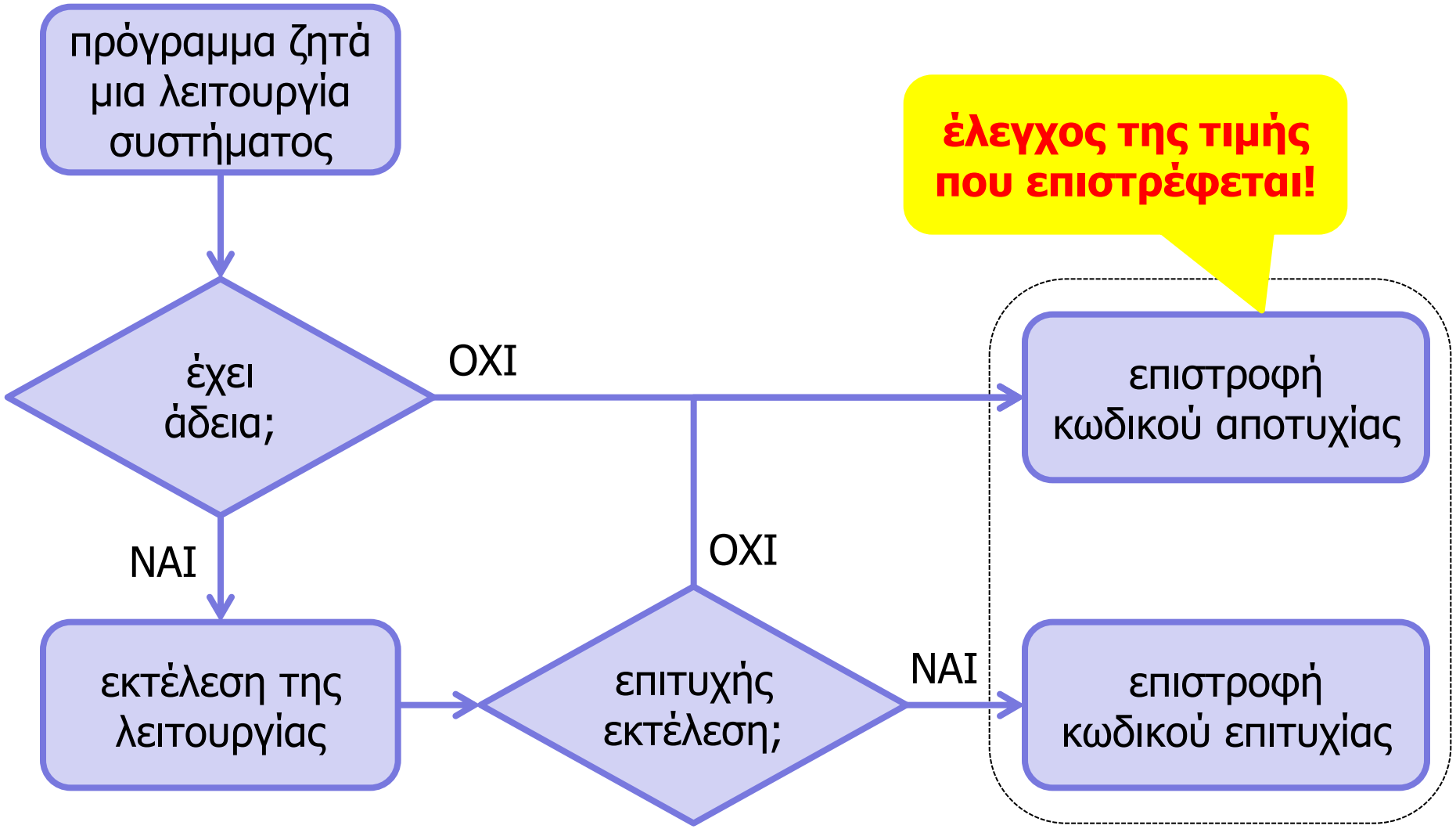
Πως πραγματοποιούμε κλήσεις συστήματος;

- Μέσω της **ειδικής/αντίστοιχης** συνάρτησης περιτυλίγματος (special/dedicated wrapper function)
- Μέσω της **γενικής** συνάρτησης `syscall` για την εκτέλεση **οποιασδήποτε** κλήσης συστήματος
  - δέχεται ως πρώτη παράμετρο τον **αριθμό/κωδικό** της επιθυμητής κλήσης συστήματος (και μετά τις παραμέτρους που απαιτεί η κλήση συστήματος)
  - οι αριθμοί/κωδικοί για τις κλήσεις συστήματος μπορεί να αναζητηθούν μέσω της εντολής `ausyscall`

Παράδειγμα: γράψιμο σε περιγραφέα αρχείου

- Καλούμε την συνάρτηση περιτυλίγματος `write`
- Καλούμε την συνάρτηση `syscall` με αριθμό κλήσης συστήματος `1` (για `write`)

```c
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>

int main(int argc, char *argv[]){
  char str[] = "Hello world!\n";
  int res;

  // indirect system call via stdio library
  printf("%s",str);

  // system call via glibc wrapper function
  res = write(STDOUT_FILENO,str, 13);
  printf("Return value is: %d\n", res);

  // system call via generic syscall function
  res = syscall(SYS_write, STDOUT_FILENO, str, 13);
  printf("Return value is: %d\n", res);

  return(0);
}
```


πρόγραμμα ζητά μια λειτουργία συστήματος
έχει άδεια;
ΟΧΙ
ΝΑΙ
εκτέλεση της λειτουργίας
επιτυχής εκτέλεση;
ΟΧΙ
ΝΑΙ
επιστροφή κωδικού αποτυχίας
επιστροφή κωδικού επιτυχίας
έλεγχος της τιμής που επιστρέφεται!

Αποτυχία κλήσης συστήματος

- Η τιμή που επιστρέφει μια κλήση συστήματος πρέπει **οπωσδήποτε(!!!)** να ελέγχεται
- Σηματοδοτεί (μόνο) το **αν** η κλήση πέτυχε ή όχι
  - **δεν** φανερώνει τον λόγο της αποτυχίας

- Ο **λόγος αποτυχίας** αποθηκεύεται, ως κωδικός, στην καθολική μεταβλητή συστήματος `errno`
  - οι κωδικοί λαθών ορίζονται στο αρχείο `errno.h`
  - η τιμή της `errno` συνήθως ανανεώνεται **μόνο** σε περίπτωση αποτυχίας μια κλήσης συστήματος – γιατί;
- Η συνάρτηση `strerror` επιστρέφει το μήνυμα που αντιστοιχεί στην τρέχουσα τιμή της `errno`
  - η συνάρτηση `perror` εκτυπώνει αυτό το μήνυμα

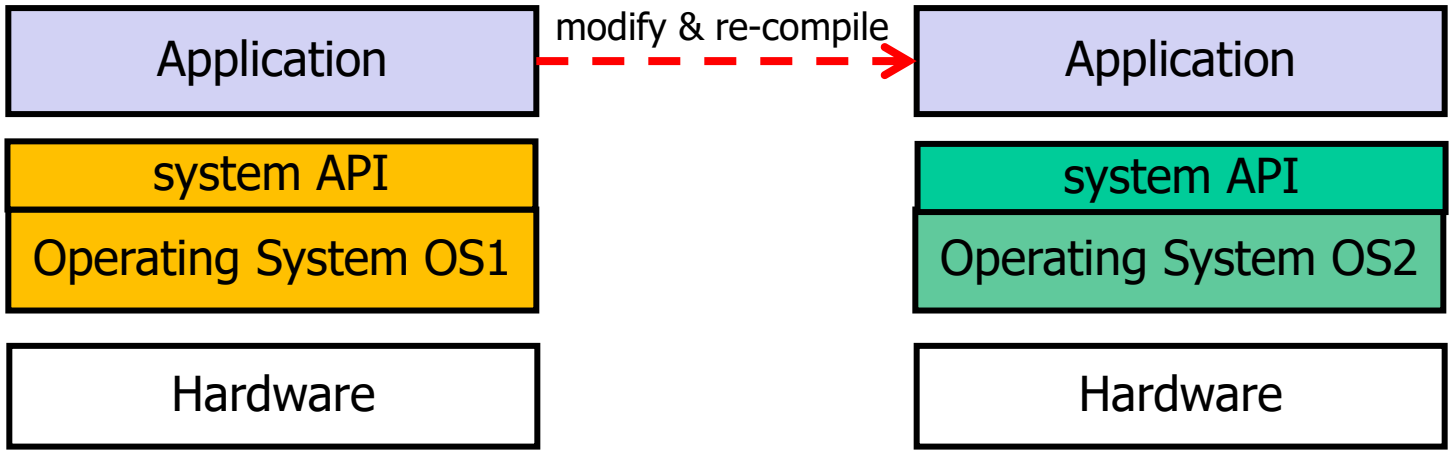
Πρέπει να διαβάζετε το manual

RTFM!

Το πρόβλημα της φορητότητας του κώδικα (code portability)

- Γενικά είναι επιθυμητό ο κώδικας που γράφουμε να μπορεί να εκτελεστεί σε **διαφορετικούς** Η/Υ
  - διαφορετικό υλικό/επεξεργαστές, διαφορετικά λειτουργικά
- **Φορητός κώδικας:** κώδικας που εκτελείται **σωστά** σε διαφορετικά μηχανήματα (και περιβάλλοντα) **χωρίς καμία αλλαγή!**
  - σημαντικό για την διάδοση/χρήση του λογισμικού
- Όσο πιο κοντά στο λειτουργικό σύστημα βρίσκεται ο κώδικας τόσο λιγότερο μεταφέρσιμος είναι
  - μπορεί όμως να εκμεταλλευτεί ειδικές λειτουργίες
- Υπάρχουν **προδιαγραφές** λειτουργιών επιπέδου συστήματος **ανεξάρτητες** λειτουργικού συστήματος
  - POSIX (Portable Operating System Interface for Unix)

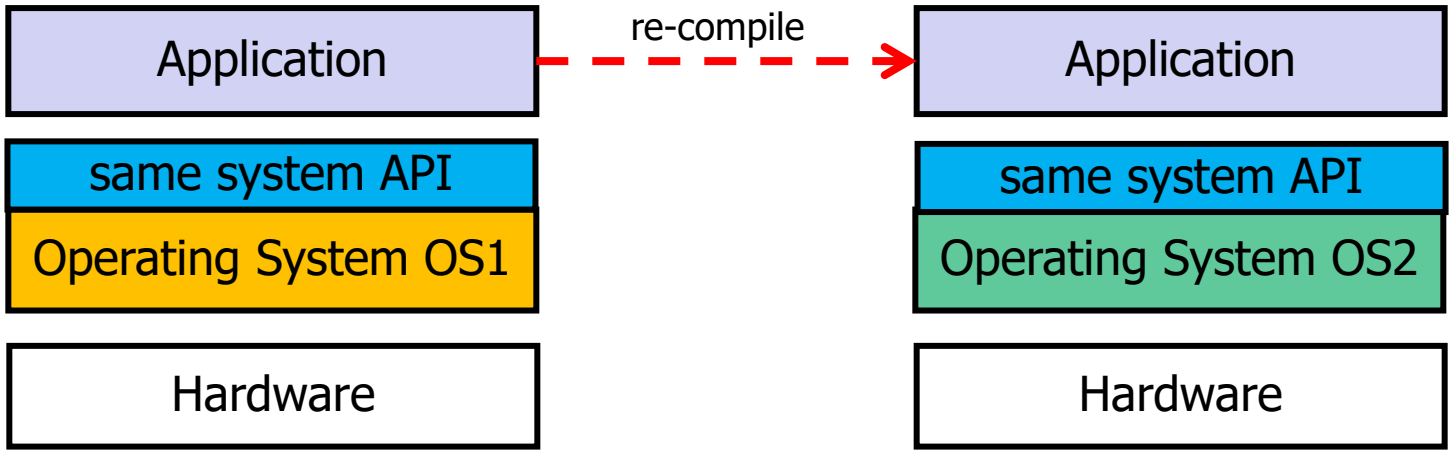
Μεταφορά σε διαφορετικό λειτουργικό σύστημα με διαφορετική διεπαφή προγραμματισμού



- **Αλλαγή/προσαρμογή** κώδικα εφαρμογής
  - για να χρησιμοποιεί την νέα διεπαφή προγραμματισμού

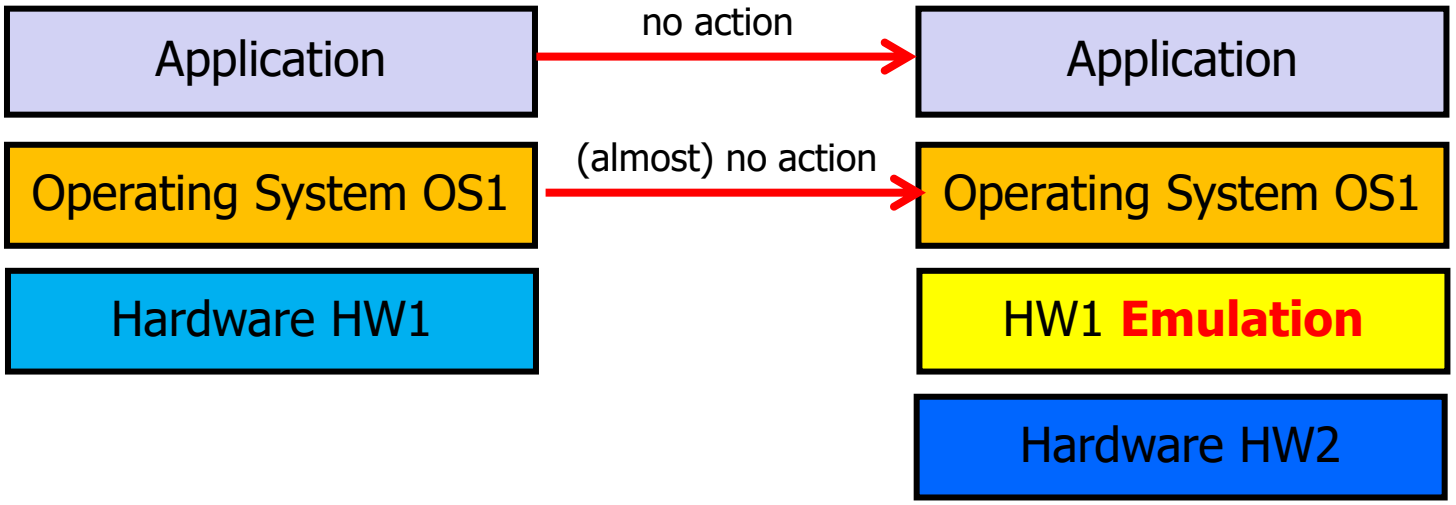
- **Μετάφραση και αποσφαλμάτωση** εφαρμογής

Μεταφορά σε διαφορετικό λειτουργικό σύστημα με ίδια διεπαφή προγραμματισμού



- Μετάφραση κώδικα εφαρμογής
  - δεν χρειάζεται αλλαγή / εκ νέου αποσφαλμάτωση κώδικα
  - η διεπαφή προγραμματισμού του συστήματος παραμένει ίδια (ακόμα και πάνω από διαφορετικά λειτουργικά συστήματα)
- Μια τέτοια προσπάθεια είναι το POSIX
  - Portable System Interface for Unix

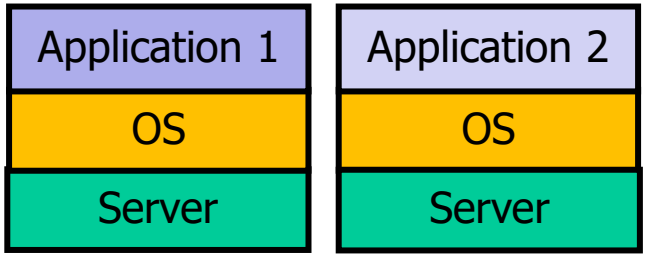
Μεταφορά εφαρμογής ΚΑΙ λειτουργικού συστήματος μέσω εξομοίωσης υλικού



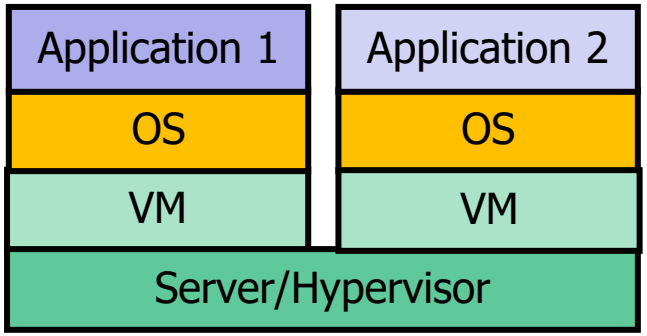
- Η εφαρμογή και το λειτουργικό **δεν** αλλάζουν
- Το λειτουργικό εκτελείται πάνω από ειδικό λογισμικό που **εξομοιώνει** το υλικό για το οποίο έχει σχεδιαστεί/υλοποιηθεί/μεταφραστεί το λειτουργικό

Υποστήριξη πολυμίσθωσης (multitenancy)

- Ξεχωριστοί εξυπηρετητές



- Κοινόχρηστος εξυπηρετητής με ξεχωριστές εικονικές μηχανές (virtual machines)



- Κοινόχρηστος εξυπηρετητής και λειτουργικό με ξεχωριστούς περιέκτες (containers)