Προγραμματισμός II (ECE116)

#4 ξεχωριστή μετάφραση & διασύνδεση (separate compilation & linking)

Σπύρος Λάλης
lalis@uth.gr

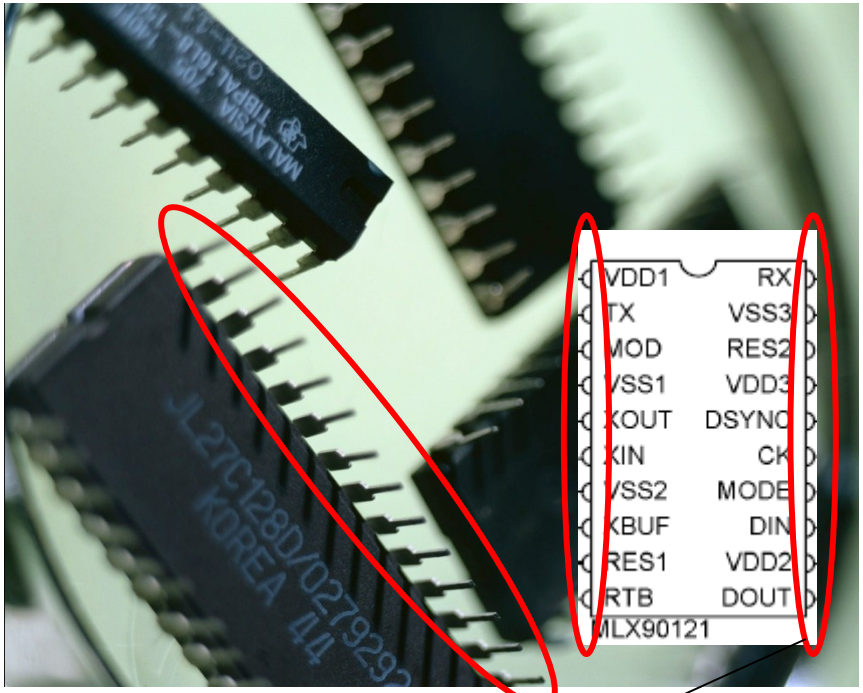
Χρήση λογισμικού που ήδη υπάρχει

- Τα πολύπλοκα συστήματα αναπτύσσονται σταδιακά, «χτίζοντας» πάνω σε **υπάρχουσα** λειτουργικότητα

- Όταν αυτή παρέχεται από ένα κομμάτι υλικού, χρησιμοποιούμε μια **φυσική διασύνδεση**
  - physical interface, π.χ., pins, cables, sockets, …
- Όταν αυτή παρέχεται από ένα κομμάτι λογισμικού, χρησιμοποιούμε μια **διασύνδεση προγραμματισμού**
  - (application) programming interface

- Σε κάθε περίπτωση, πρέπει να γνωρίζουμε και να τηρούμε τις **προδιαγραφές** χρήσης
  - πρέπει να διαβάσουμε τα αντίστοιχα manuals

hardware components



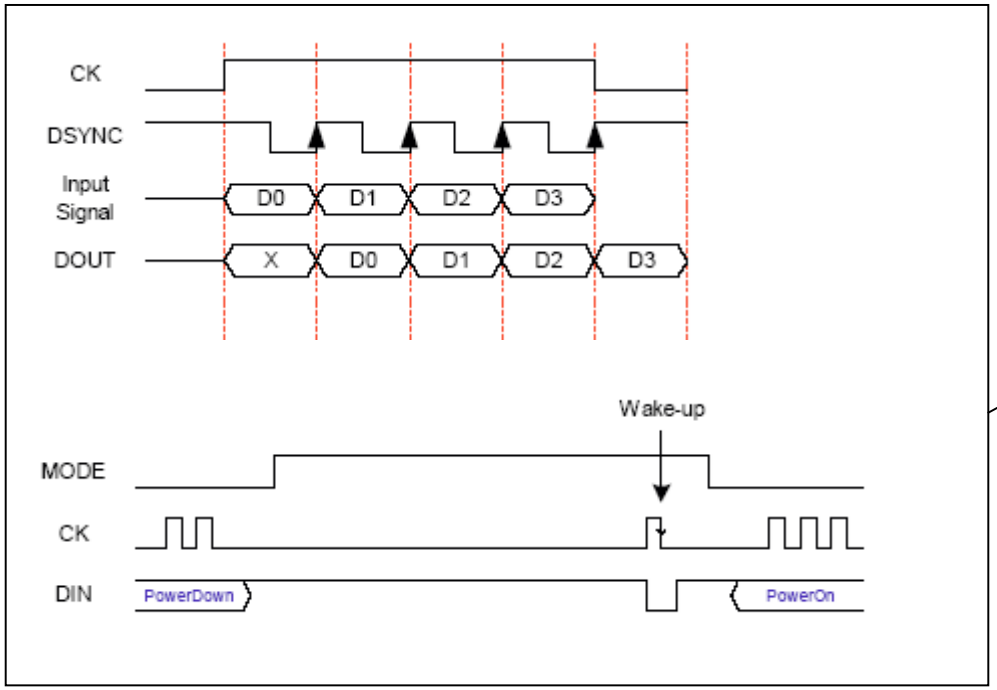
hardware interface

software components

```
0A1F68CED90B3020D85F1397C
5D90A3120D35F1197FFFF0131
D28CEB95F3420C85C11100
…
…
…
00B19D5711C210AA00B5A911F
F2F8D9C5B3B20B851A0AC10
```

```
void rand_init(int seed);
int rand_generate();
```

software interface



hardware documentation

```
void rand_init(int seed);
/* initialize random number
   generator with a seed;
   call once, before calling
   rnd_generate */

int rand_generate(void);
/* returns next random number */
```

software documentation

Η λογική του «μαύρου κουτιού»

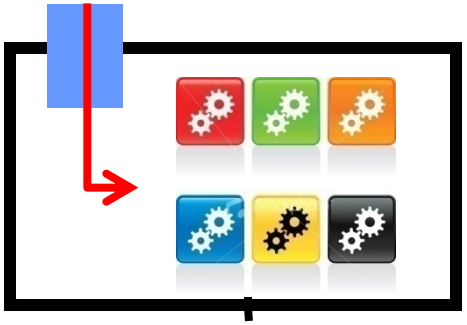
- Συνήθως μας ενδιαφέρει η λειτουργικότητα που θέλουμε να εκμεταλλευτούμε/χρησιμοποιήσουμε
  - όχι οι «εσωτερικές» λεπτομέρειες υλοποίησης της
  - αν και αυτές πάντα ενδιαφέρουν τους μηχανικούς ☺

- **Black box principle**

1. ξέρουμε πως πρέπει να χρησιμοποιήσουμε κάτι
2. αλλά **δεν** χρειάζεται να γνωρίζουμε το πώς «δουλεύει»

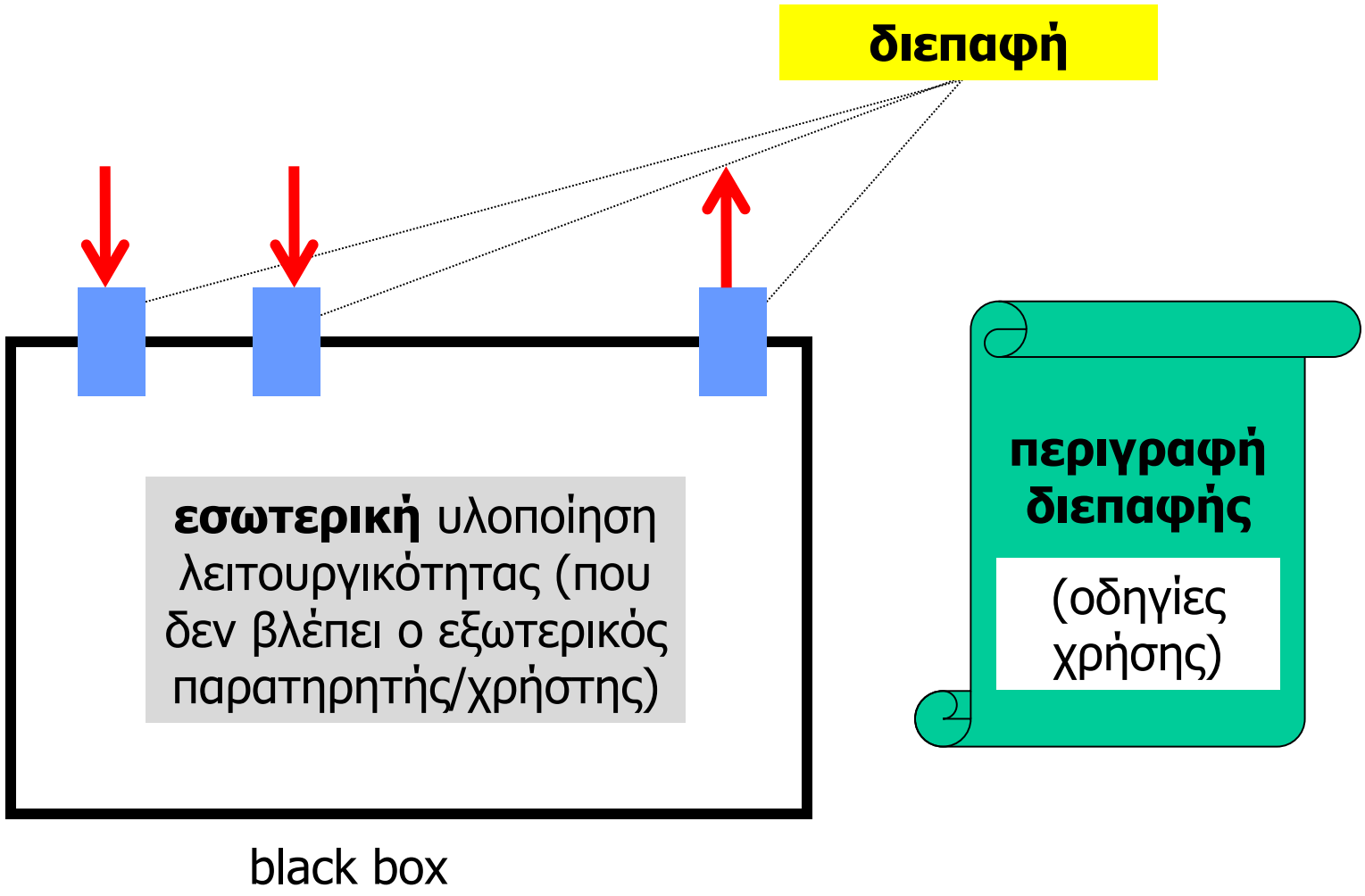
- Βασική αρχή ανάπτυξης πολύπλοκων συστημάτων
- Βασική αρχή πρακτικά για **όλα** τα τεχνολογικά προϊόντα που χρησιμοποιούμε σε καθημερινή βάση

διασύνδεση

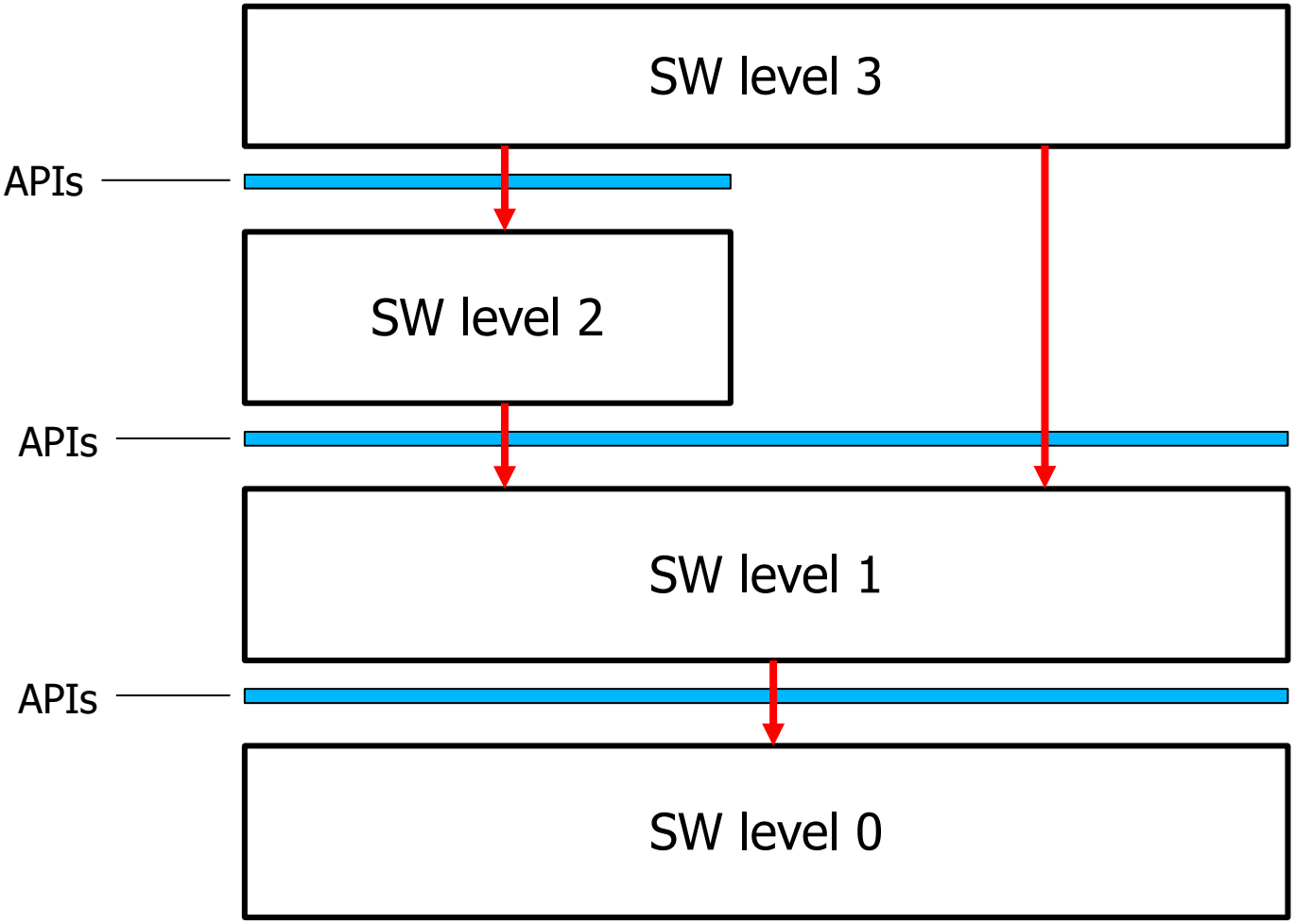


black box

Οδηγίες χρήσης:
οι τροχοί στρίβουν προς την κατεύθυνση που στρίβεις το τιμόνι


διεπαφή
εσωτερική υλοποίηση λειτουργικότητας (που δεν βλέπει ο εξωτερικός παρατηρητής/χρήστης)
περιγραφή διεπαφής
(οδηγίες χρήσης)
black box

Η πολυκατοικία / στοίβα του λογισμικού

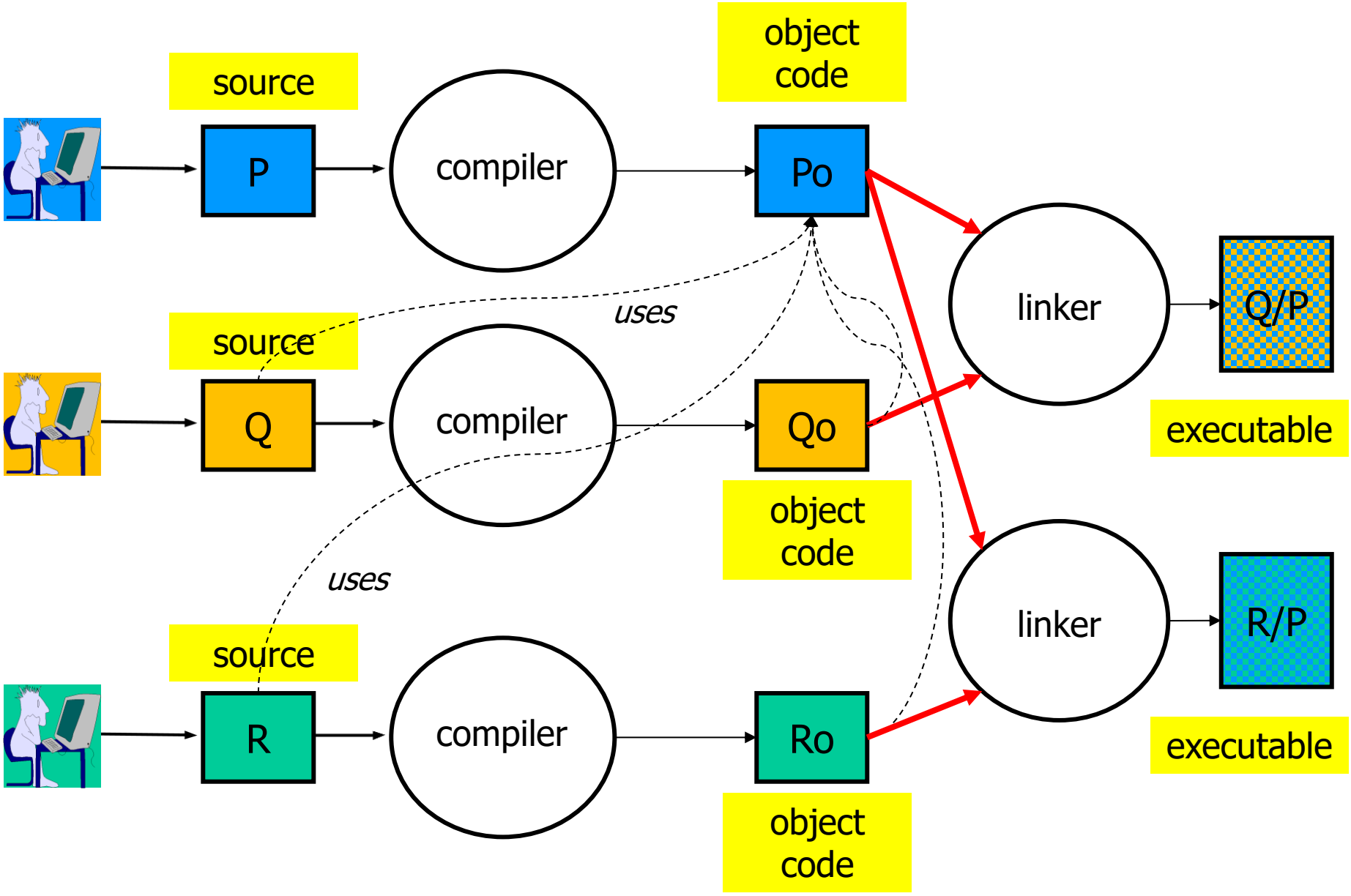


Επαναχρησιμοποίηση λογισμικού

- Το μεγάλο «στοίχημα» στην βιομηχανία λογισμικού
  - δεν χρειάζεται να ξαναγράψουμε κώδικα που έχει ήδη γραφτεί

- Προσέγγιση Α: χρησιμοποιούμε **πηγαίο** κώδικα
  - αντιγραφή μέσα στον δικό μας κώδικα
  - μετάφραση μαζί με τον δικό μας κώδικα

- Προσέγγιση Β: χρησιμοποιούμε **μεταφρασμένο** κώδικα
  - απαιτείται **μηχανισμός αναφοράς** στα αντικείμενα/λειτουργίες που εισάγει/υλοποιεί ο μεταφρασμένος κώδικας
  - απαιτείται **μηχανισμός σύνδεσης** του δικού μας κώδικα με τον μεταφρασμένο κώδικα


source
object code
P
compiler
Po
linker
Q/P
executable
uses
source
Q
compiler
Qo
object code
uses
linker
R/P
executable
source
R
compiler
Ro
object code

Ανάπτυξη τμημάτων λογισμικού

- Σχεδίαση **προγραμματιστικής διεπαφής**
  - πως πρέπει να χρησιμοποιηθεί αυτό που θα υλοποιήσουμε
  - τύποι, σταθερές, macros, μεταβλητές, συναρτήσεις
- Συγγραφή **manual**
  - τεκμηρίωση και οδηγίες χρήσης για όλα τα παραπάνω

- Κατασκευή **υλοποίησης**
  - κώδικας που υλοποιεί την λειτουργικότητα
  - όπως ορίζει η διεπαφή / περιγράφει το manual

- Μπορεί να μεταφραστεί **ξεχωριστά** (γιατί;)
  - θα διασυνδεθεί (αργότερα) με τον κώδικα των προγραμμάτων που χρησιμοποιούν την συγκεκριμένη διεπαφή

Προγραμματιστική διεπαφή στην C

- Ορίζεται μέσω των **header files**
  - macros, types, variables, function prototypes

- Η υλοποίηση γίνεται σε **ξεχωριστό** αρχείο
  - κώδικας που υλοποιεί την λειτουργικότητα
  - μπορεί να μεταφραστεί **ξεχωριστά**

- Οποιοσδήποτε άλλος κώδικας μπορεί να κάνει `#include` το header file
  - για να αναφερθεί στα αντικείμενα που ορίζονται εκεί
  - χωρίς να απαιτείται γνώση (**ούτε καν ύπαρξη!**) της υλοποίησης

Διασύνδεση (linking)

- Η μετάφραση με βάση τις δηλώσεις ενός header file, παράγει αναφορές σε **εξωτερικά αντικείμενα**
  - μεταβλητές και συναρτήσεις

- Για να παραχθεί ο τελικός εκτελέσιμος κώδικας, απαιτείται μια επιπλέον **διαδικασία ανεύρεσης και ενσωμάτωσης** των αντίστοιχων αντικειμένων
  - που είναι μέρος της (ήδη μεταφρασμένης) υλοποίησης

- Αυτή η διαδικασία ονομάζεται **διασύνδεση**
  - όταν η διασύνδεση ολοκληρωθεί επιτυχώς (δεν υπάρχουν αναφορές σε άγνωστα αντικείμενα), και υπάρχει η κυρίως συνάρτηση `main`, παράγεται το **τελικό εκτελέσιμο**

Παράδειγμα: Τηλεφωνικός κατάλογος

- Κώδικας που παρουσιάστηκε σε προηγούμενη διάλεξη
- Φροντίσαμε να είναι **ήδη** καλά δομημένος
- Καθορισμένες λειτουργίες που υποστηρίζονται μέσα από αντίστοιχες συναρτήσεις, με τεκμηρίωση για την χρήση τους
- Ιδανικές προϋποθέσεις για την δημιουργία ενός ξεχωριστού/ανεξάρτητου συστατικού λογισμικού
  - που μπορεί να χρησιμοποιηθεί στην συνέχεια από **πολλά** και **διαφορετικά** προγράμματα «πελάτες»

```
typedef struct {
  char name[64];  /* name of person */
  char phone[64]; /* phone as string for flexibility */
} entry_t;

typedef struct {
  entry* entries; /* dynamic table holding the entries */
  int size;       /* current size of the table */
} phonebook_t;

void phonebook_init(phonebook_t *pb);
int phonebook_add(phonebook_t *pb, const entry_t *e);
int phonebook_find(const phonebook_t *pb,
                   const char *name, entry_t *e);
int phonebook_rmv(phonebook_t *pb, const char *name);
void phonebook_clear(phonebook_t *pb);
```

phonebook.h

```
#ifndef __PHONEBOOK_H
#define __PHONEBOOK_H

typedef struct {
  char name[64];  /* name of person */
  char phone[64]; /* phone as string for flexibility */
} entry_t;

typedef struct {
  entry* entries; /* dynamic table holding the entries */
  int size;       /* current size of the table */
} phonebook_t;

extern void phonebook_init(phonebook_t *pb);
extern int phonebook_add(phonebook_t *pb, const entry_t *e);
extern int phonebook_find(const phonebook_t *pb,
                          const char *name, entry_t *e);
extern int phonebook_rmv(phonebook_t *pb, const char *name);
extern void phonebook_clear(phonebook_t *pb);

#endif
```

phonebook.c

```
#include "phonebook.h"

void phonebook_init(phonebook_t *pb) {
  …
}


int phonebook_add(phonebook_t *pb, const entry_t *e) {
  …
}

int phonebook_find(const phonebook_t *pb,
                   const char *name, entry_ *e) {
  …
}

int phonebook_rmv(phonebook_t *pb, const char *name) {
  …
}

void phonebook_clear(phonebook_t *pb) {
  …
}
```

main.c

```
#include "phonebook.h"
…

int main(int argc, char *argv[]) {
  …
  entry_t e;
  phonebook_t *pb;

  phonebook_init(&pb);
  …
  phonebook_add(&pb, &e);
  …
  phonebook_find(&pb, e.name, &e);
  …
  phonebook_rmv(&pb, e.name);
  …
  phonebook_add(&pb, &e);
  …
  phonebook_find(&pb, e.name, &e);
  …
  phonebook_clear(&pb);
  …
  return(0);
}
```

Μετάφραση και χρήση συστατικού

- Μεταγλώττιση κώδικα συστατικού και δημιουργία object file
  - `gcc -Wall -c phonebook.c`
- Μεταγλώττιση κώδικα «πελάτη» που χρησιμοποιεί τις λειτουργίες του συστατικού και δημιουργία object file
  - `gcc -Wall -c main.c`
- Διασύνδεση object files και δημιουργία εκτελέσιμου, που περιλαμβάνει τον κώδικα του συστατικού
  - `gcc -Wall main.o phonebook.o –o main`

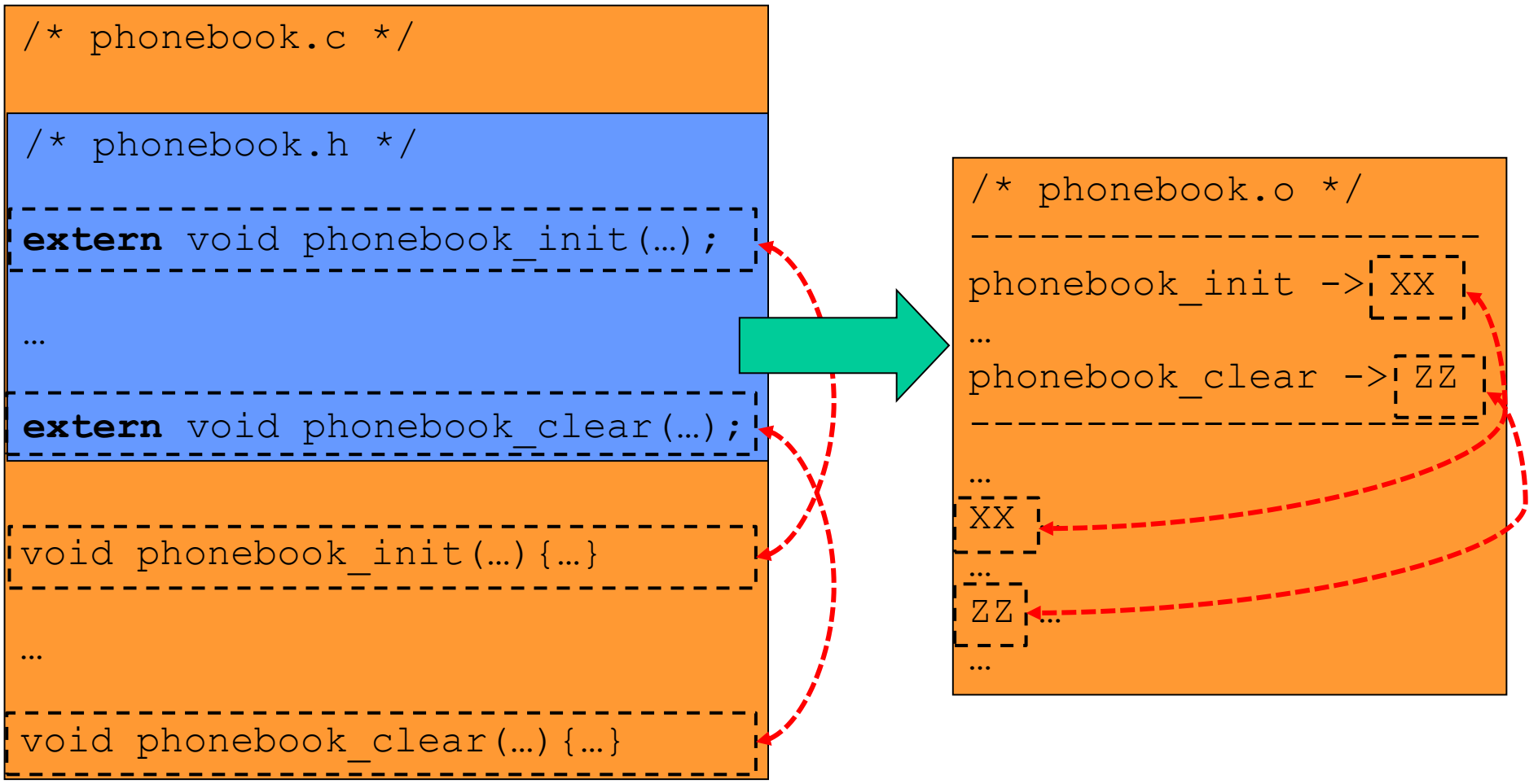
```c
/* phonebook.h */

extern void phonebook_init(…);
extern int phonebook_add(…);
extern int phonebook_find(…);
extern int phonebook_rmv(…);
extern void phonebook_clear(…);
```

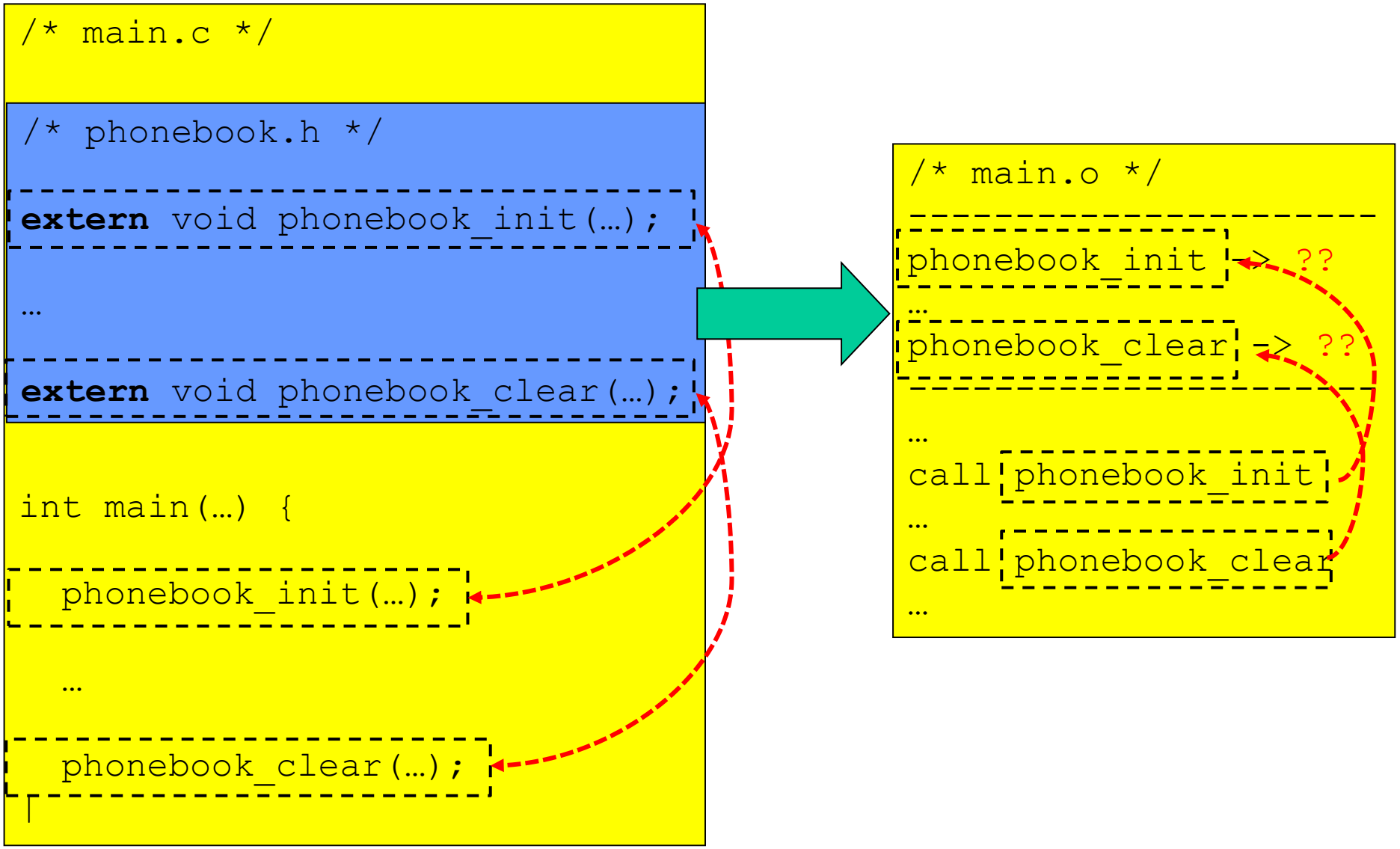
υλοποιείται από

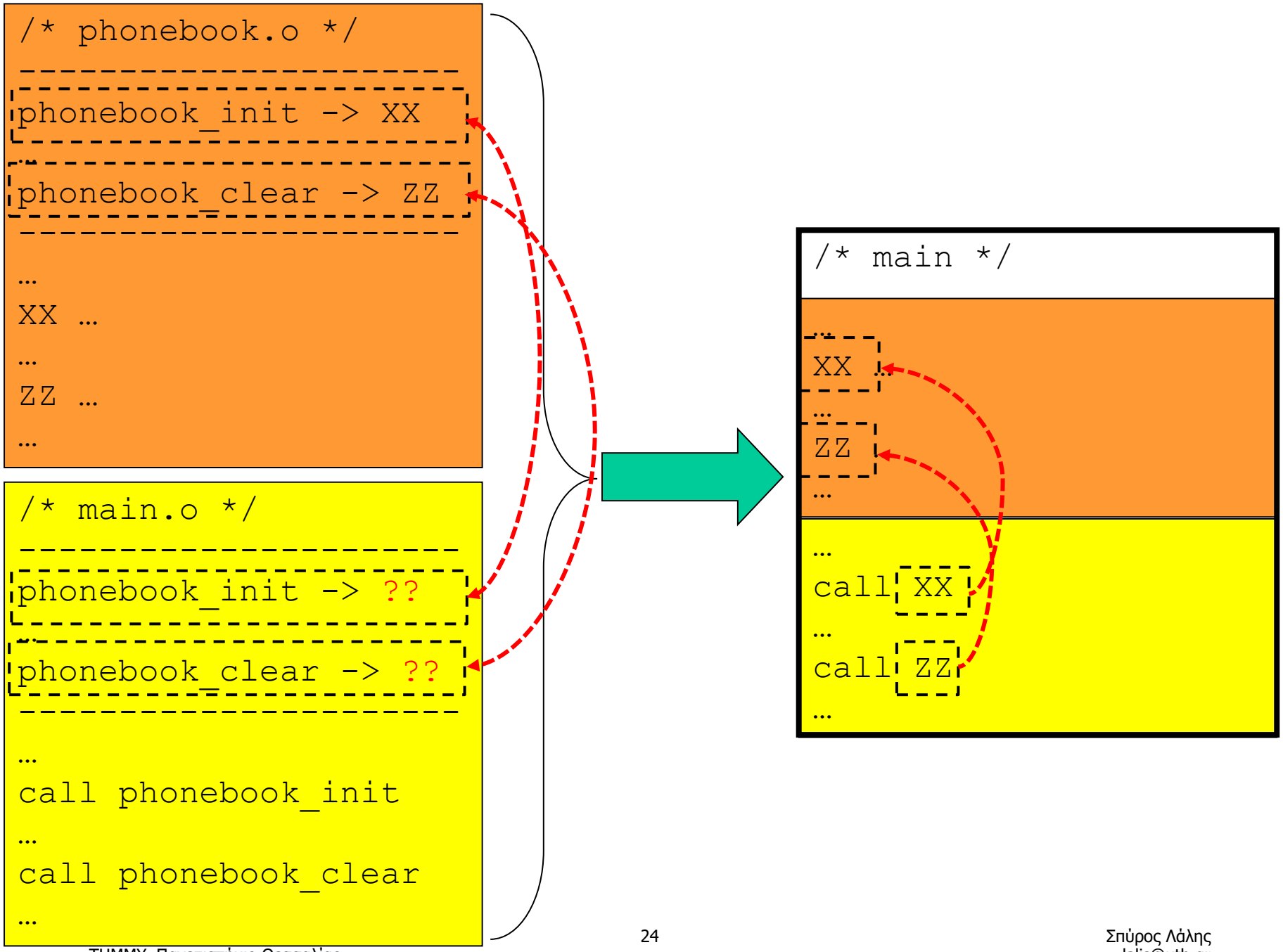
χρησιμοποιείται από

```c
/* phonebook.c */

#include "phonebook.h"

void phonebook_init(…){…}

…

void phonebook_clear(…){…}
```

```c
/* main.c */

#include "phonebook.h"

int main(int argc, char *argv[]) {

  phonebook_init(…);

  …

  phonebook_clear(…);
}
```


/* phonebook.c */
/* phonebook.h */
extern void phonebook_init(…);
…
extern void phonebook_clear(…);
void phonebook_init(…){…}
…
void phonebook_clear(…){…}
/* phonebook.o */
phonebook_init -> XX
…
phonebook_clear -> ZZ
…
XX
…
ZZ
…


/* main.c */
/* phonebook.h */
extern void phonebook_init(…);
…
extern void phonebook_clear(…);
int main(…) {
phonebook_init(…);
…
phonebook_clear(…);
/* main.o */
phonebook_init -> ??
…
phonebook_clear -> ??
…
call phonebook_init
…
call phonebook_clear
…


/* phonebook.o */
phonebook_init -> XX
phonebook_clear -> ZZ
…
XX …
…
ZZ …
…
/* main.o */
phonebook_init -> ??
phonebook_clear -> ??
…
call phonebook_init
…
call phonebook_clear
…
/* main */
…
XX …
…
ZZ
…
…
call XX
…
call ZZ
…

Γιατί ξεχωριστή μετάφραση;

- Όταν ο κώδικας μεγαλώσει ή/και περιλαμβάνει μεγάλο αριθμό συστατικών, η διαδικασία της μετάφρασης μπορεί να έχει **σημαντικό κόστος**
  - χρόνος του προγραμματιστή, χρήση πόρων του υπολογιστή
- Η ξεχωριστή μετάφραση **αποφεύγει** την άσκοπη σπατάλη πόρων
  - **δεν** μεταφράζεται ξανά κώδικας που δεν έχει αλλάξει

- Όμως πρέπει να κρατάμε «λογαριασμό» για το **που έγιναν αλλαγές** και τις **εξαρτήσεις** ανάμεσα σε διαφορετικά κομμάτια κώδικα / αρχεία
  - πολύ εύκολο να γίνουν λάθη παραλήψεις
  - βλέπε make …

Βιβλιοθήκη

- Τμήμα λογισμικού **ευρύτερης χρησιμότητας**
  - έχει σχεδιαστεί με σκοπό να διευκολύνουν την ανάπτυξη πολλών **διαφορετικών** εφαρμογών
- Αναπτύσσεται ως ανεξάρτητο τμήμα λογισμικού
  - με ξεχωριστή μετάφραση
- Χρησιμοποιείται χωρίς (καλή) γνώση για την εσωτερική υλοποίηση – αρχή του black box!
  - π.χ. χρησιμοποιούμε τα αντικείμενα που ορίζονται στο `stdio.h` χωρίς να ξέρουμε (ακριβώς) τι κάνει «μέσα της» η βιβλιοθήκη
- Η διαθέσιμη λειτουργικότητα και κανόνες χρήσης της προγραμματιστικής διεπαφής της βιβλιοθήκης πρέπει να **περιγράφεται κατάλληλα**
  - βλέπε manuals

Δημιουργία στατικής βιβλιοθήκης

- Σχεδίαση της προγραμματιστικής διεπαφής
- Δημιουργία του αντίστοιχου header file
  - π.χ., **`mylib`**`.h`
- Υλοποίηση της λειτουργικότητας σε αρχείο κώδικα
  - π.χ., **`mylib`**`.c`
- Μεταγλώττιση και δημιουργία object file
  - π.χ., `gcc -Wall -c` **`mylib`**`.c -o` **`mylib`**`.o`
- Δημιουργία βιβλιοθήκης
  - π.χ., `ar rcs lib`**`mylib`**`.a` **`mylib`**`.o`
  - το όνομα πρέπει να αρχίζει με `lib` και να έχει κατάληξη `.a`
  - το όνομα δεν χρειάζεται να έχει σχέση με το όνομα του αρχείου στο οποίο έχει αποθηκευτεί ο κώδικας

Χρήση στατικής βιβλιοθήκης

- Συμπεριλαμβάνουμε το header file της βιβλιοθήκης
  - `#include "mylib.h"` αν βρίσκεται στον κατάλογο που βρίσκεται το αρχείο με τον κώδικα που γράφουμε
  - `#include <mylib.h>` αν βρίσκεται στον κατάλογο βιβλιοθηκών του συστήματος
- Χρήση των συναρτήσεων της βιβλιοθήκης
  - με βάση το header file / τις περιγραφές του manual
- Μεταγλώττιση και σύνδεση με την βιβλιοθήκη
  - π.χ. `gcc myprog.c -o myprog -lmylib -L.`
  - στην επιλογή διασύνδεσης `-l` παραλείπουμε από το όνομα της βιβλιοθήκης το πρόθεμα `lib` και την κατάληξη `.a`
  - η τοποθεσία του κώδικα της βιβλιοθήκης δίνεται αμέσως μετά την επιλογή `-L` (π.χ. το " . " είναι ο τρέχον κατάλογος)

Περισσότερα για το `extern`

- Δηλώνει **ρητά** ότι ένα αντικείμενο του προγράμματος (μεταβλητή/συνάρτηση) υπάρχει/υλοποιείται **αλλού**
  - σε άλλο αρχείο
- Κατά την διασύνδεση, γίνεται αναζήτηση για ένα ταιριαστό/συμβατό **εξωτερικό αντικείμενο**
  - από άλλο αρχείο που συμπεριλαμβάνεται στην διασύνδεση
- Αν δεν βρεθεί, προκύπτει **σφάλμα** διασύνδεσης

Προσοχή!

- Η δήλωση/χρήση μιας συνάρτησης που **δεν** έχει τοπική υλοποίηση, οδηγεί **αυτόματα** σε διασύνδεση με **όποια** ταιριαστή υλοποίηση βρεθεί αλλού
  - ακόμα και αυτή **δεν** έχει δηλωθεί ως `extern`

- Καθολικές μεταβλητές που έχουν το ίδιο όνομα αλλά βρίσκονται σε ξεχωριστά αρχεία/κώδικες, **ταυτίζονται** αυτόματα κατά την διασύνδεση
  - οι αναφορές που γίνονται από των κώδικα κάθε αρχείου καταλήγουν να αφορούν την **ίδια** θέση μνήμης
  - παρότι ο προγραμματιστής ίσως **δεν** έχει τέτοια πρόθεση

Περισσότερα για το `static`

- Πρέπει να αποφεύγονται «αυτόματες» διασυνδέσεις
  - όταν δεν υπάρχει τέτοια πρόθεση από τον προγραμματιστή

- Η επιθυμητή **τοπικότητα / απομόνωση** επιτυγχάνεται με τον προσδιορισμό `static`

- Η εμβέλεια καθολικών μεταβλητών και συναρτήσεων **περιορίζεται** «στο πλαίσιο» του κώδικα/αρχείου που δηλώνονται / υλοποιούνται
- Αυτό **εμποδίζει** οποιαδήποτε (τυχαία) διασύνδεση με κώδικα που βρίσκεται σε άλλο αρχείο

Πρόβλημα ...

- Αν ένας κώδικας Α ορίζει καθολικές μεταβλητές / υλοποιεί συναρτήσεις **με σκοπό** αυτές **να χρησιμοποιηθούν** μέσα από οποιοδήποτε **άλλο κώδικα**, αυτές δεν θα δηλωθούν ως `static`
- Όταν ο κώδικας Α συνδεθεί με ένα άλλο κώδικα Β, οι μεταβλητές / συναρτήσεις του Α είναι διαθέσιμες για σύνδεση με αντίστοιχες (άμεσες ή έμμεσες) εξωτερικές δηλώσεις που υπάρχουν στον Β
- Αν στον κώδικα Β έχουν γίνει λάθος δηλώσεις ή έχουν ξεχαστεί δηλώσεις, μπορεί να γίνει αυτόματα διασύνδεση που να οδηγήσει σε **λάθος** αποτέλεσμα
  - από τα πιο άσχημα bugs ...