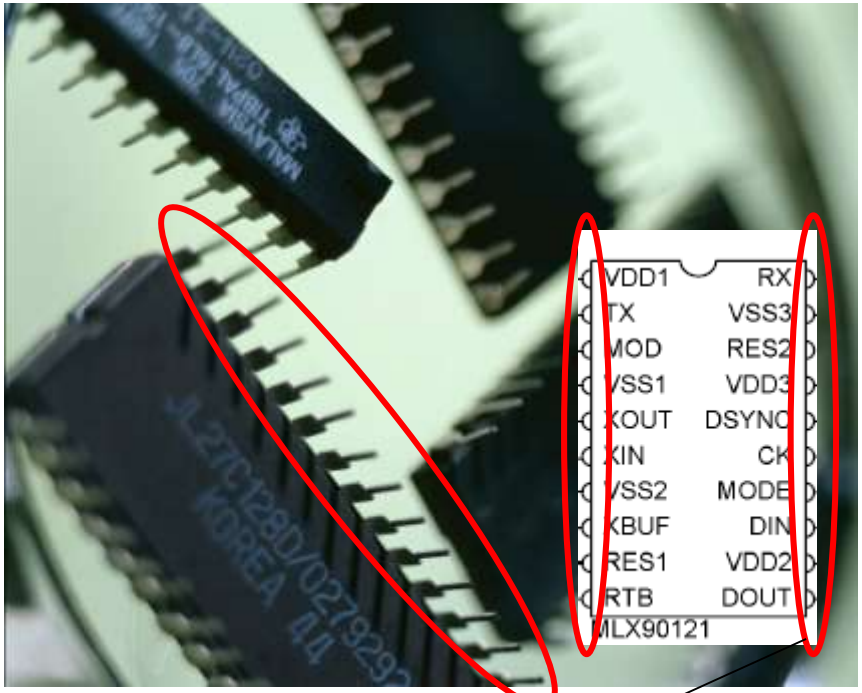
Μεταγλώττιση και σύνδεση πολλαπλών αρχείων κώδικα

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
- Όταν αυτή παρέχεται από λογισμικό, 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

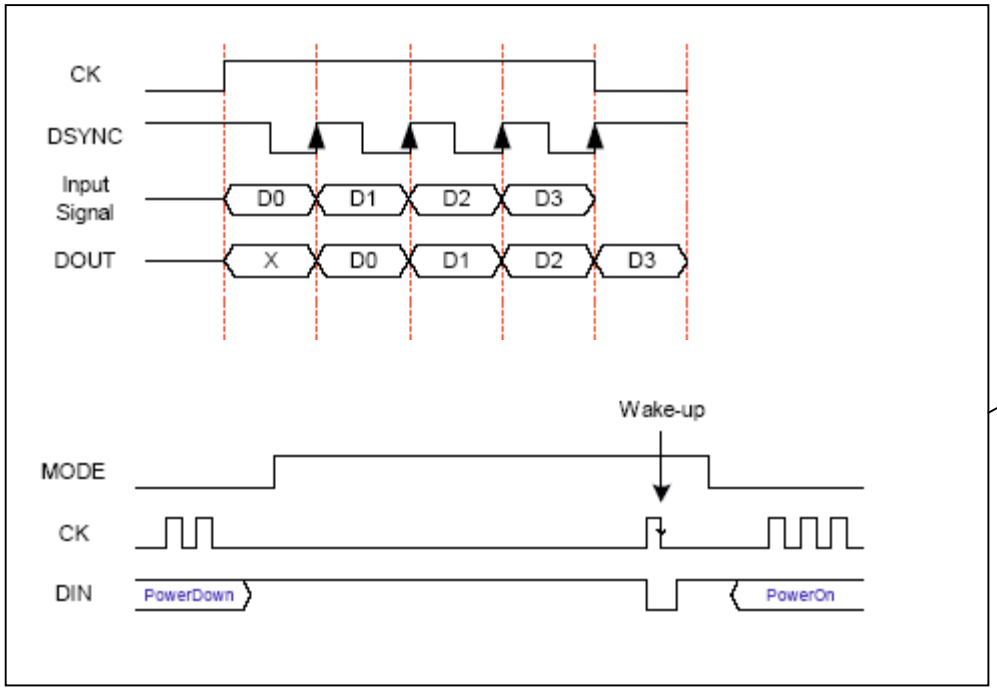



```
0A1F68CED90B3020D85F1397C
5D90A3120D35F1197FFFF0131
D28CEB95F3420C85C11100
…
00B19D5711C210AA00B5A911F
F2F8D9C5B3B20B851A0AC10
```

```
void init(int);
int rnd();
```

hardware interface

software interface



hardware documentation

```
void init(int seed);
/* initialize random number
   generator with a seed;
   call once, before rnd */

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

software documentation

Διεπαφή (προγραμματισμού) λογισμικού

- Αποτελείται (κυρίως) από δηλώσεις τύπων, σταθερών, μεταβλητών και συναρτήσεων
- Κάθε συνάρτηση χρησιμεύει για ένα συγκεκριμένο σκοπό, π.χ. την εκτέλεση κάποιας λειτουργικότητας
- Η **σημασία / κωδικοποίηση** των παραμέτρων και λειτουργικότητα των συναρτήσεων ορίζεται μέσω κατάλληλων προδιαγραφών (τεκμηρίωση)
- Για να χρησιμοποιήσουμε ένα τμήμα λογισμικού
  - πρέπει να το συνδέσουμε κατάλληλα με το πρόγραμμα μας
  - πρέπει να καλούμε τις συναρτήσεις σύμφωνα με τις προδιαγραφές που έχουν οριστεί

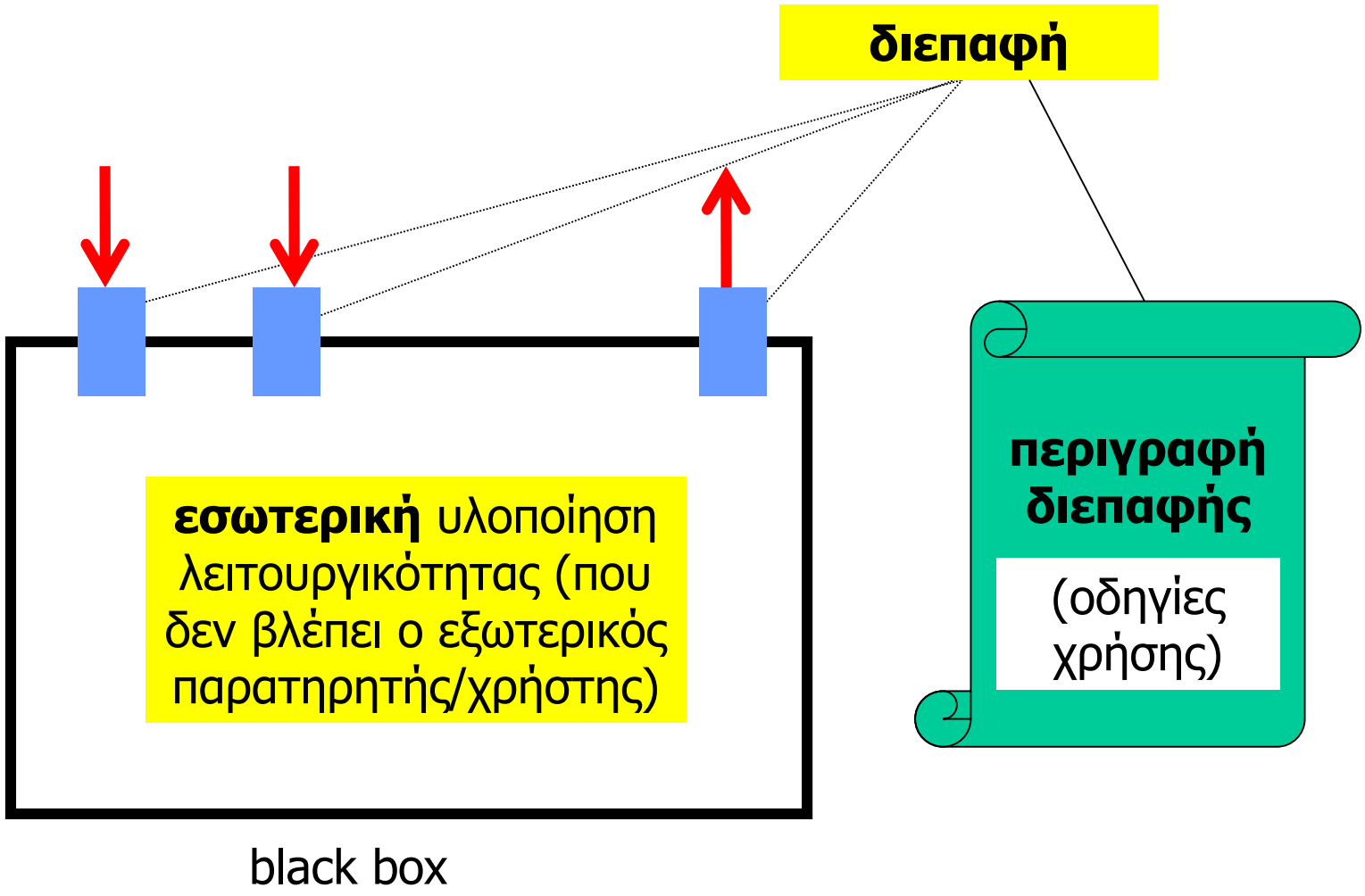
Η λογική του «μαύρου κουτιού»

- 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:**
  - ξέρουμε πως μπορούμε/πρέπει να το χρησιμοποιήσουμε
  - **χωρίς** να χρειάζεται να γνωρίζουμε το πώς «δουλεύει»

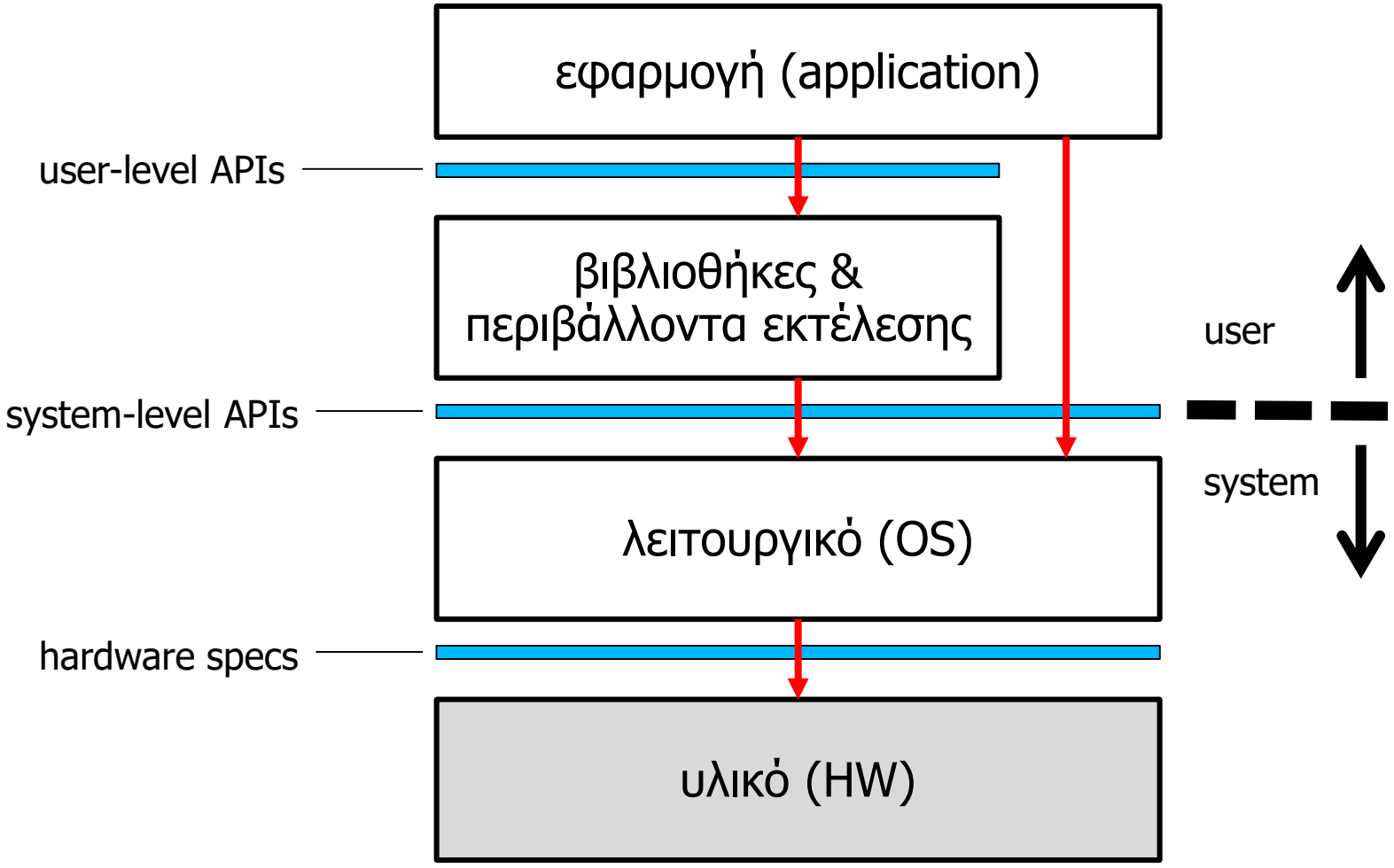
- Βασική αρχή ανάπτυξης πολύπλοκων συστημάτων
- Βασική αρχή πρακτικά για **όλα** τα τεχνολογικά προϊόντα που χρησιμοποιούμε σε καθημερινή βάση

διασύνδεση

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


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

Η πολυκατοικία / στοίβα του λογισμικού

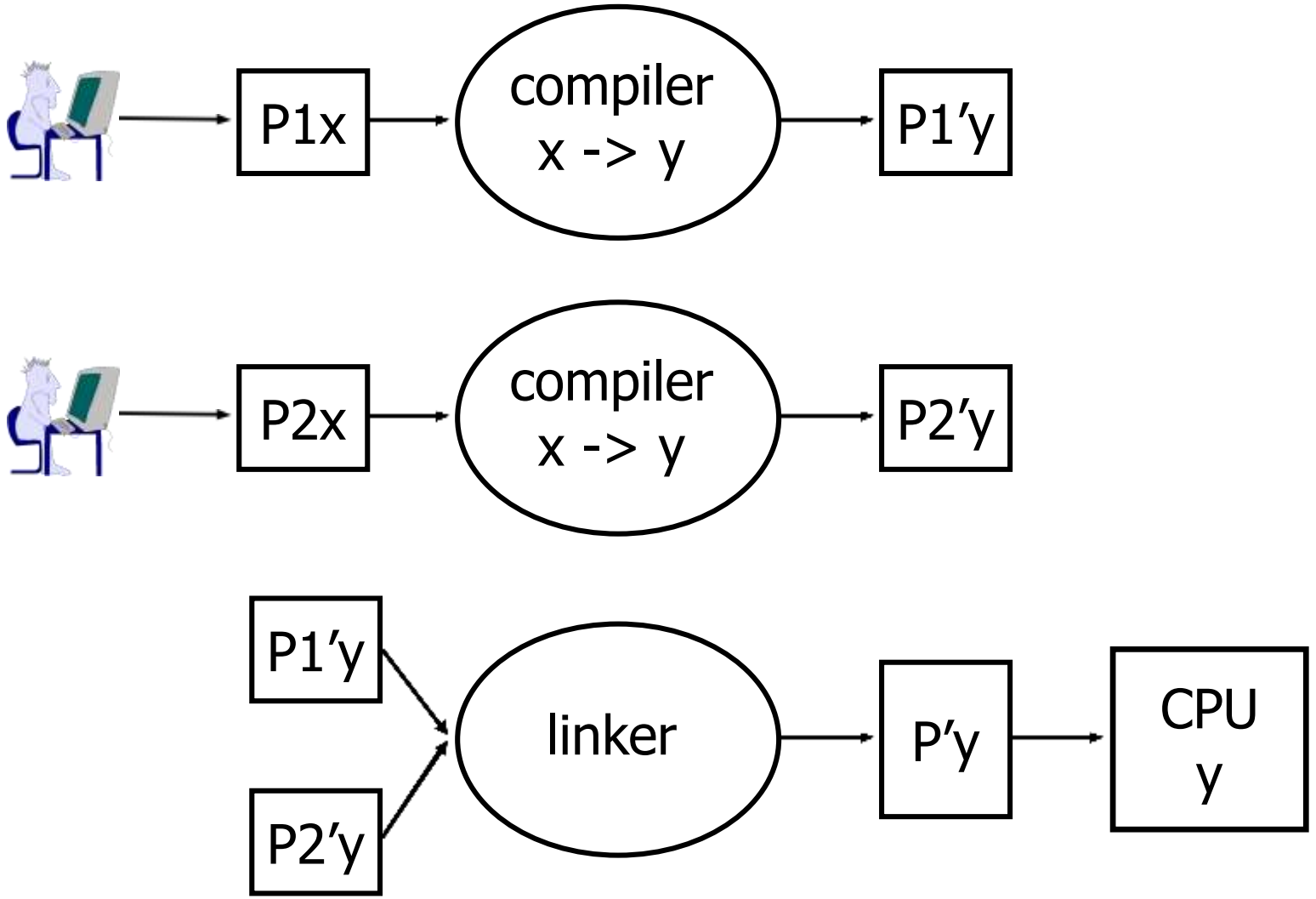


Επαναχρησιμοποίηση λογισμικού

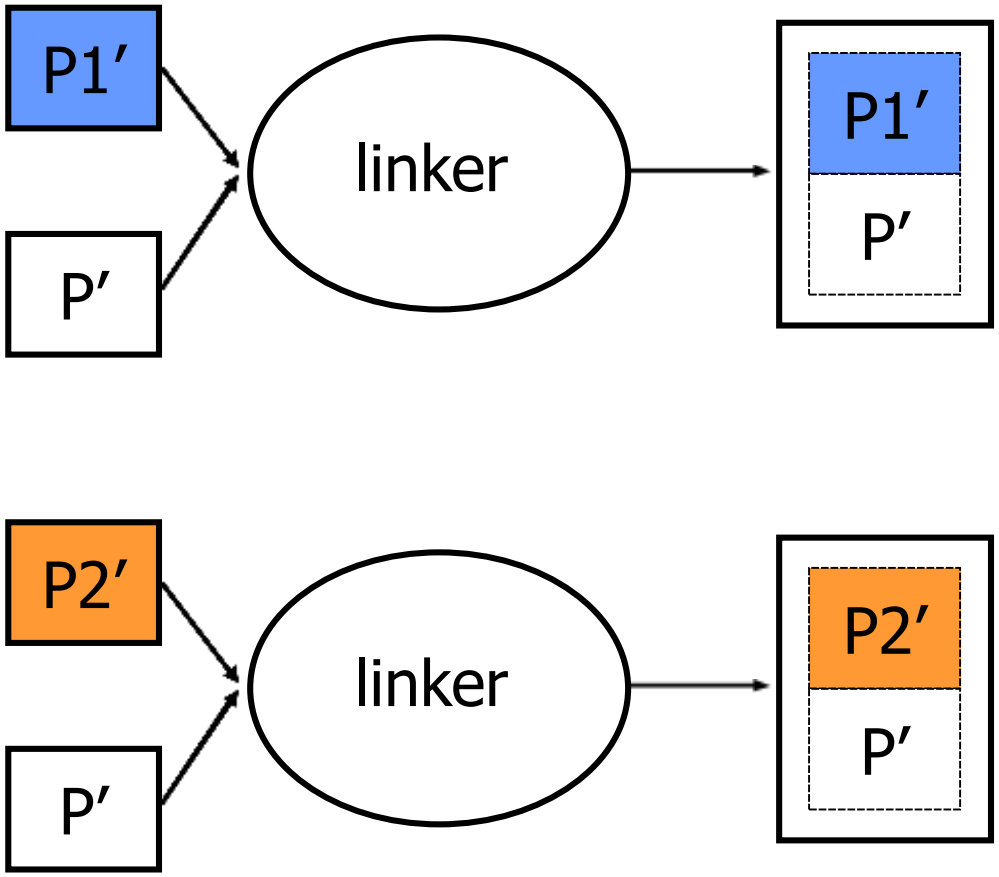
- Το μεγάλο «στοίχημα» στην βιομηχανία λογισμικού
- Ιδανικά, δεν χρειάζεται να ξαναγράψουμε κώδικα που έχει ήδη γραφτεί (από κάποιους άλλους)

- Προσέγγιση Α: αντιγράφουμε τον πηγαίο κώδικα, κάνοντας ίσως προσαρμογές για τις ανάγκες μας

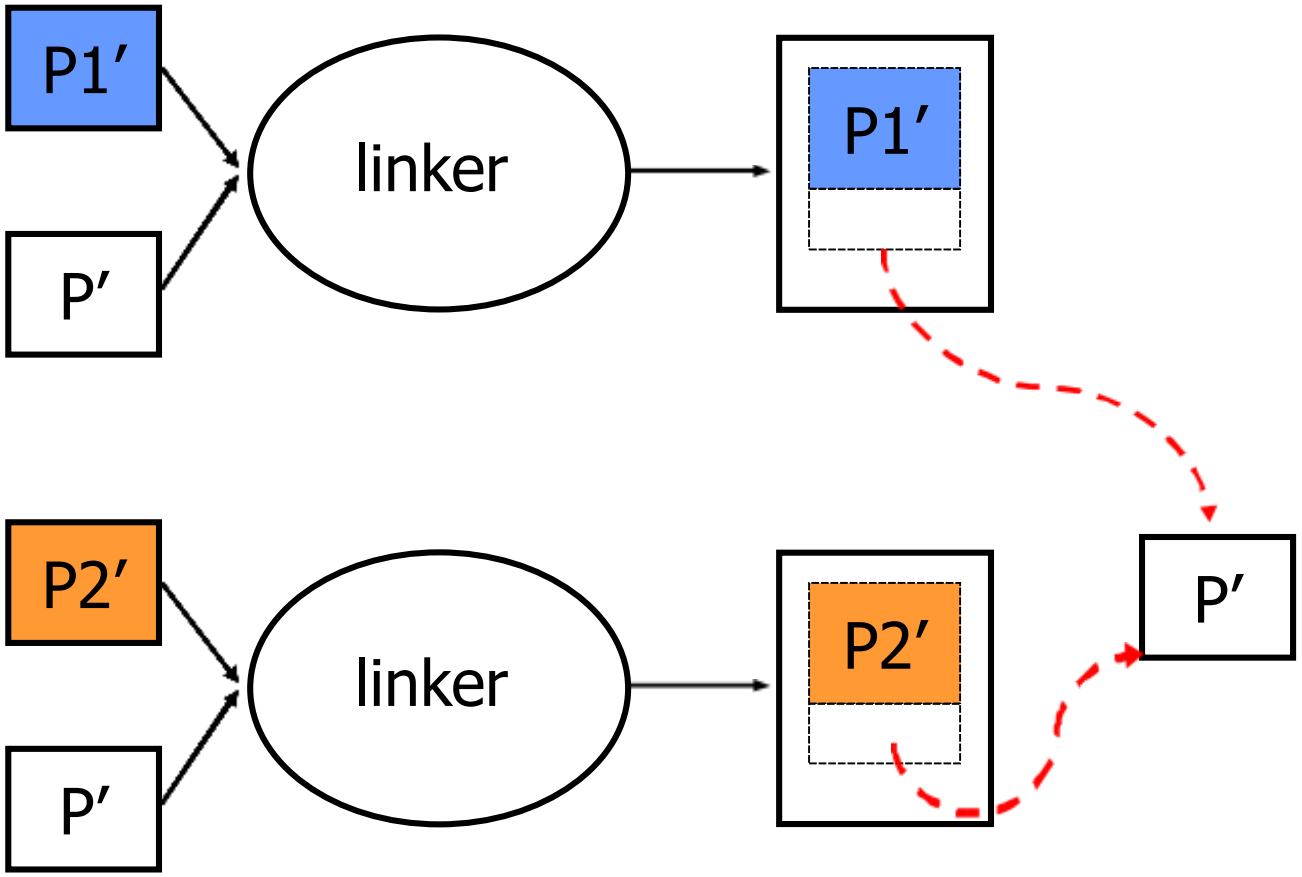
- Προσέγγιση Β: καλούμε τον εκτελέσιμο κώδικα, σύμφωνα με τις οδηγίες χρήσης
  - απαιτεί **μηχανισμό σύνδεσης** του δικού μας κώδικα με τον κώδικα που έχει ήδη γραφτεί/μεταφραστεί


P1x
compiler
x -> y
P1'y
P2x
compiler
x -> y
P2'y
P1'y
P2'y
linker
P'y
CPU
y

Στατική διασύνδεση



Δυναμική διασύνδεση



Ανάπτυξη τμημάτων λογισμικού

- Σχεδίαση προγραμματιστικής **διεπαφής**: αποφασίζουμε το πως θα χρησιμοποιηθεί (προγραμματιστικά) η λειτουργικότητα

- Κατασκευή **header file**: ορισμός της διεπαφής με την μορφή ενός header file που θα χρησιμοποιηθεί (κυρίως) από τα προγράμματα «πελάτες»

- Κατασκευή **υλοποίησης**: σε ξεχωριστό αρχείο υλοποιούμε τη λειτουργικότητα, όπως ορίζεται στο header file, που μεταφράζεται για να δημιουργηθεί το τμήμα κώδικα που θα συνδεθεί (αργότερα) με τον κώδικα από τα προγράμματα «πελάτες»

Header files στην C

- Οι διεπαφές προγραμματισμού στην C περιγράφονται μέσω **header files**, που περιέχουν
  - ορισμούς τύπων δεδομένων
  - δηλώσεις σταθερών και καθολικών μεταβλητών
  - δηλώσεις συναρτήσεων
- Οι δηλώσεις μεταβλητών και συναρτήσεων πρέπει να έχουν τον προσδιορισμό `extern`
  - υποδεικνύει στον μεταγλωττιστή ότι οι αντίστοιχες υλοποιήσεις τους **δεν** βρίσκονται στο ίδιο αρχείο
- Κάνοντας `#include` το header file, το πρόγραμμα μπορεί να χρησιμοποιεί τύπους, μεταβλητές και συναρτήσεις ενός ξεχωριστού τμήματος λογισμικού
  - τον πηγαίο κώδικα του οποίου μπορεί να μην γνωρίζουμε

Διασύνδεση

- Όταν ένα πρόγραμμα μεταφράζεται με βάση τις δηλώσεις ενός header file, ο κώδικας που παράγεται περιέχει αναφορές σε **εξωτερικά αντικείμενα** (μεταβλητές και συναρτήσεις)
- Για να παραχθεί ο τελικός εκτελέσιμος κώδικας, απαιτείται μια επιπλέον διαδικασία **ενσωμάτωσης** των ορισμών και υλοποιήσεων τους
  - που βρίσκονται σε αρχείο που έχει ήδη μεταφραστεί
- Αυτή η διαδικασία ονομάζεται **διασύνδεση** (linking)
- Αφού ο μεταφρασμένος κώδικας συνδεθεί με όλα τα υπόλοιπα μεταφρασμένα τμήματα που παρέχουν τους ορισμούς / υλοποιούν των εξωτερικών μεταβλητών / συναρτήσεων, δημιουργείται το **τελικό εκτελέσιμο**

```c
/* foo.h */

extern int i;

extern void boo();
```

υλοποιείται από

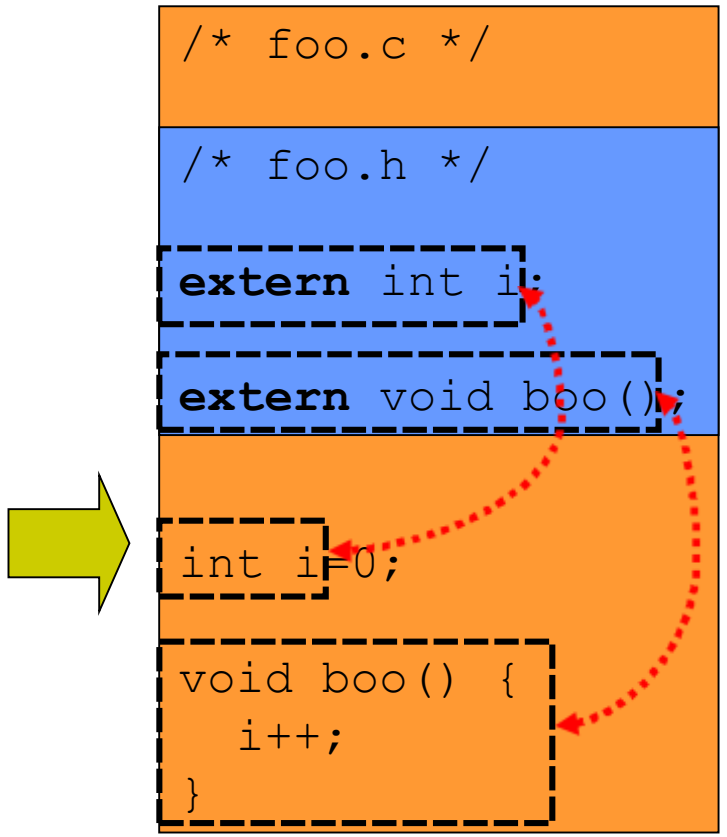
χρησιμοποιείται από

```c
/* foo.c */

#include "foo.h"

int i=0;

void boo() {
  i++;
}
```

```c
/* main.c */

#include "foo.h"

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

  boo();
  i++;
  boo();
}
```

```
/* foo.h */

extern int i;

extern void boo();
```

```
/* foo.c */

#include "foo.h"

int i=0;

void boo() {
  i++;
}
```

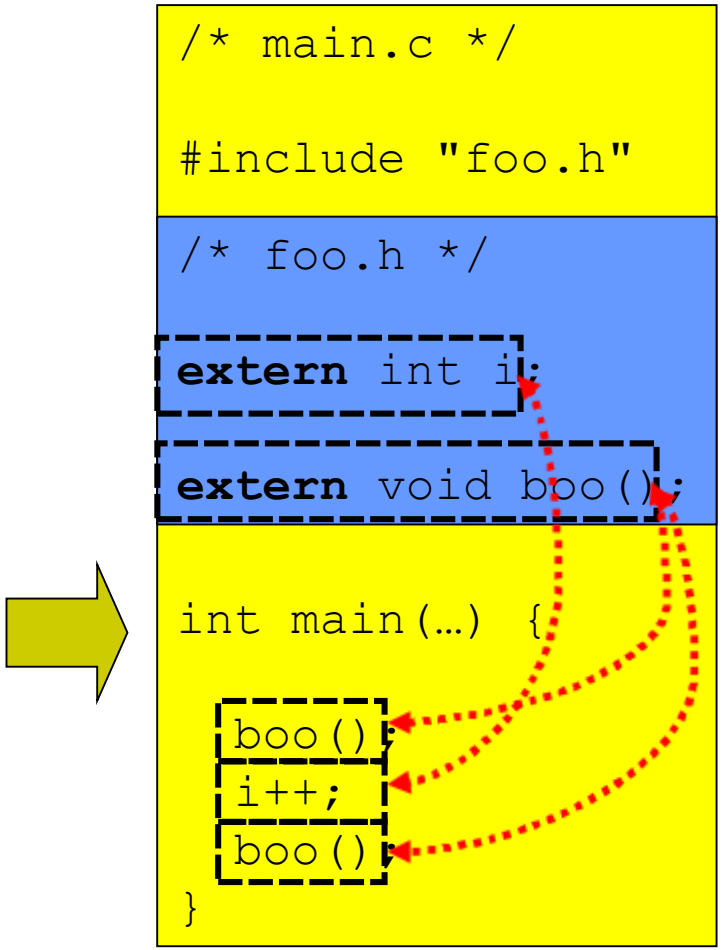



```
/* foo.o */
------------
i -> 0
boo -> 5
------------
0 …
1 …
2 …
3 …
4 …
5 …
6 …
7 …
8 …
```

```
$ gcc foo.c -c -o foo.o
```

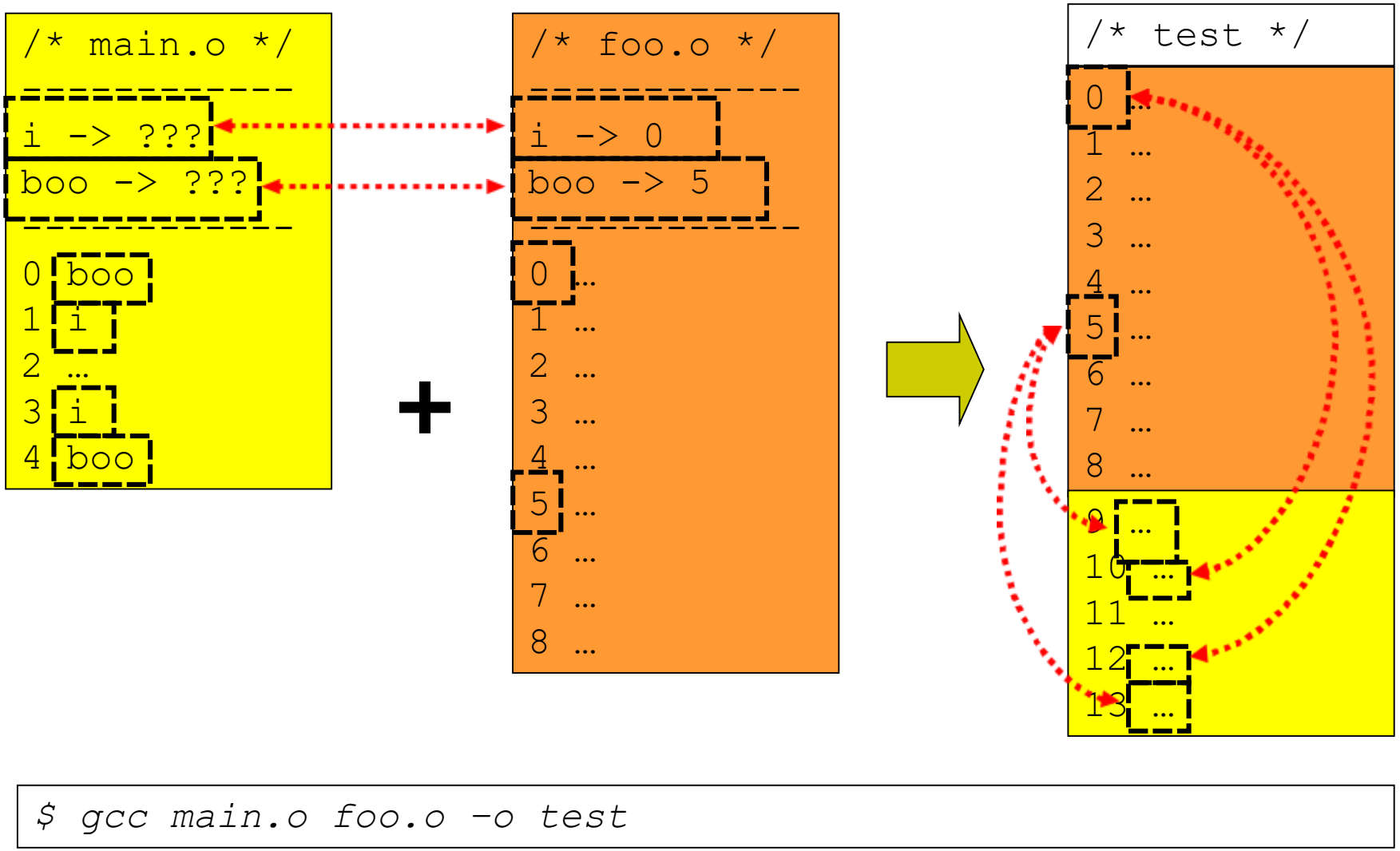
```
/* foo.h */

extern int i;

extern void boo();
```

```
/* main.c */

#include "foo.h"

int main(…) {

  boo();
  i++;
  boo();
}
```




```
/* main.o */
------------
i -> ???
boo -> ???
------------
0 boo
1 i
2 …
3 i
4 boo
```

```
$ gcc main.c -c -o main.o
```




```
$ gcc main.o foo.o -o test
```

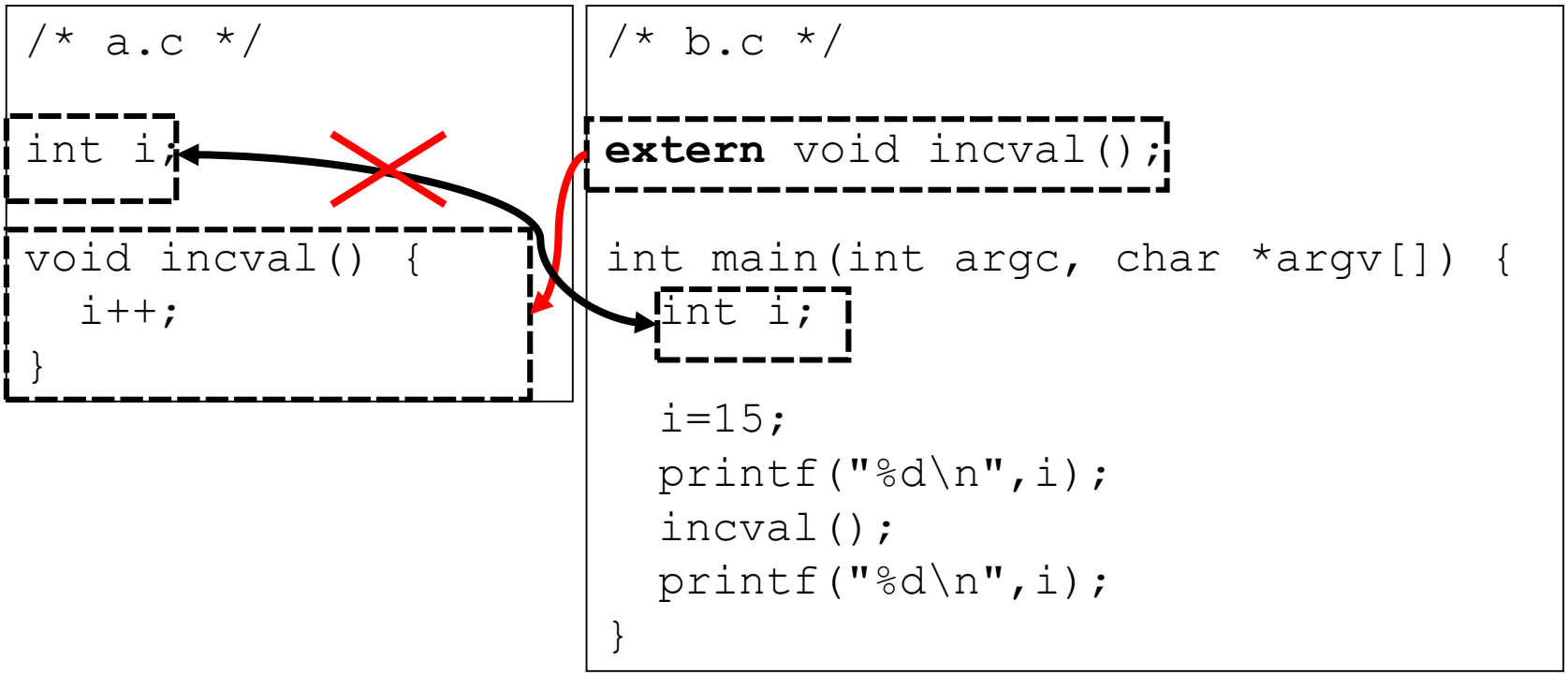
Περισσότερα για το `extern`

- Ο προσδιορισμός `extern` χρησιμοποιείται για τις δηλώσεις εξωτερικών μεταβλητών / συναρτήσεων, που ορίζονται / υλοποιούνται σε ένα άλλο αρχείο
- Η παράλειψη του `extern` σε δήλωση συνάρτησης (που δεν υλοποιείται τοπικά) οδηγεί σε **αναζήτηση** για **«ταιριαστή»** υλοποίηση της συνάρτησης στα αρχεία με τα οποία γίνεται σύνδεση του κώδικα
- Η παράλειψη του `extern` σε δήλωση (καθολικής) μεταβλητής **δεν** εγγυάται τοπικότητα
- Αν μια άλλη καθολική μεταβλητή δηλώνεται σε άλλο αρχείο με το ίδιο όνομα, κατά την διασύνδεση οι αναφορές σε αυτό το όνομα θα αφορούν την **ίδια** θέση μνήμης

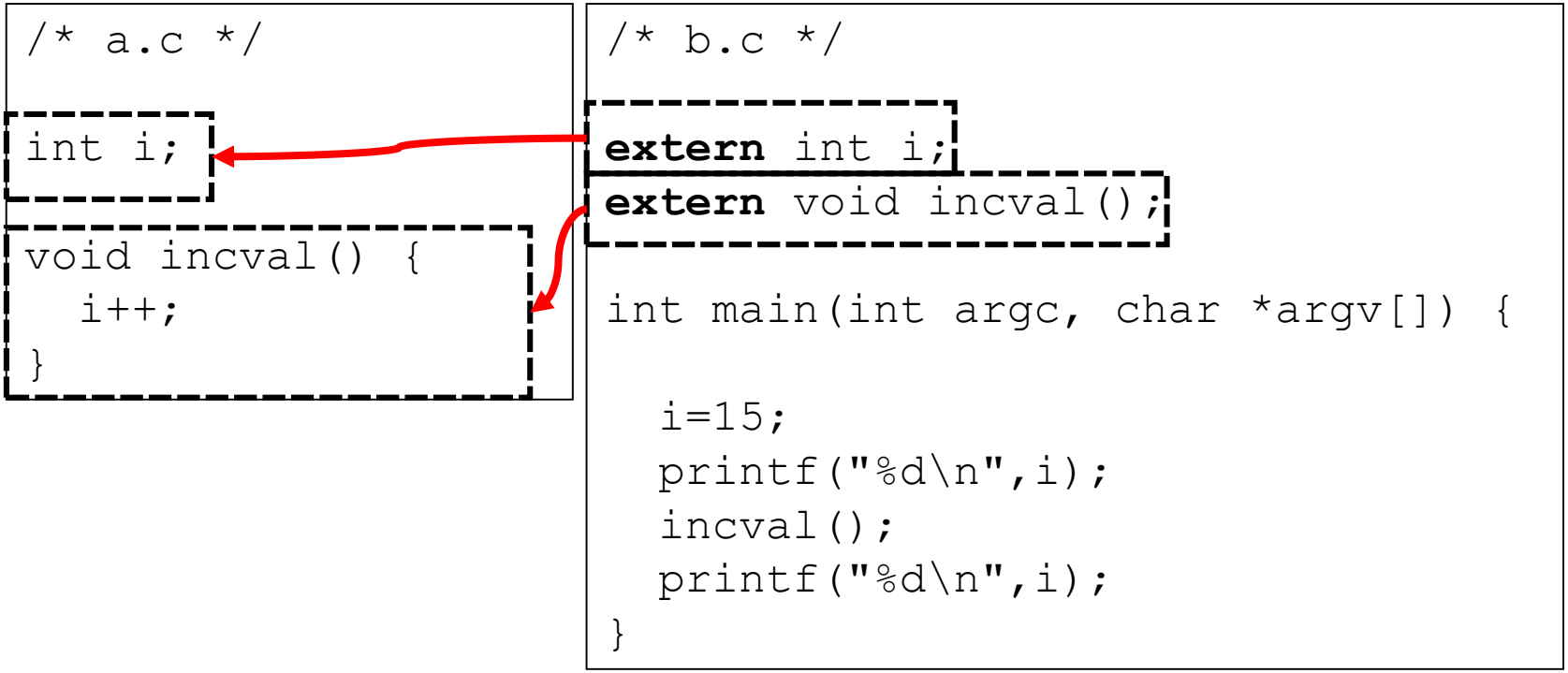
```c
/* a.c */

int i;

void incval() {
  i++;
}
```

```c
/* b.c */

extern void incval();

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

  i=15;
  printf("%d\n",i);
  incval();
  printf("%d\n",i);
}
```

```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
$
$./test
15
16
$
```




```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
$
$./test
15
15
$
```




```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
$
$./test
15
16
$
```

```
/* a.c */

extern int i;

void incval() {
  i++;
}
```

```
/* b.c */

int i;
extern void incval();

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

  i=15;
  printf("%d\n",i);
  incval();
  printf("%d\n",i);
}
```

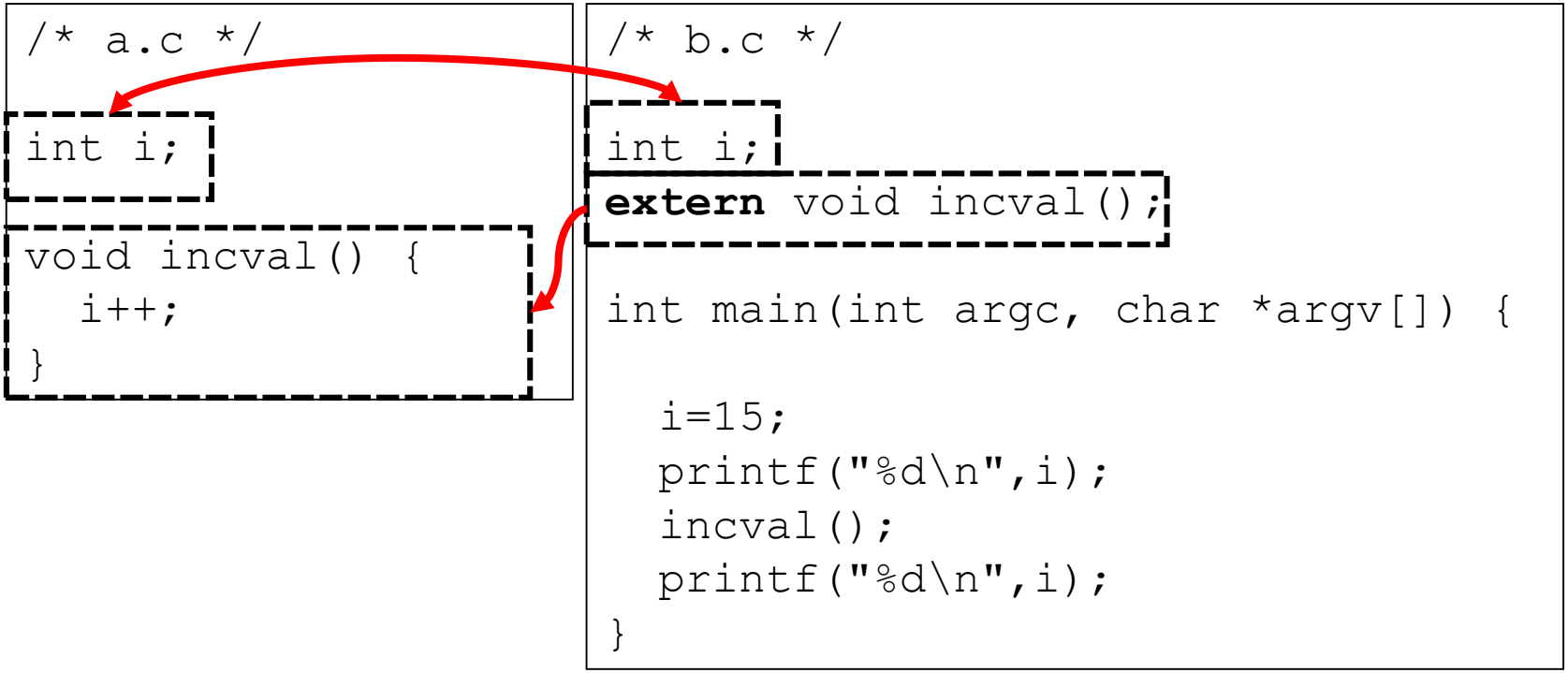
```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
$
$./test
15
16
$
```

```
/* a.c */

extern int i;

void incval() {
  i++;
}
```

```
/* b.c */

extern int i;
extern void incval();

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

  i=15;
  printf("%d\n",i);
  incval();
  printf("%d\n",i);
}
```

```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
undefined reference to i
$
```

όμως

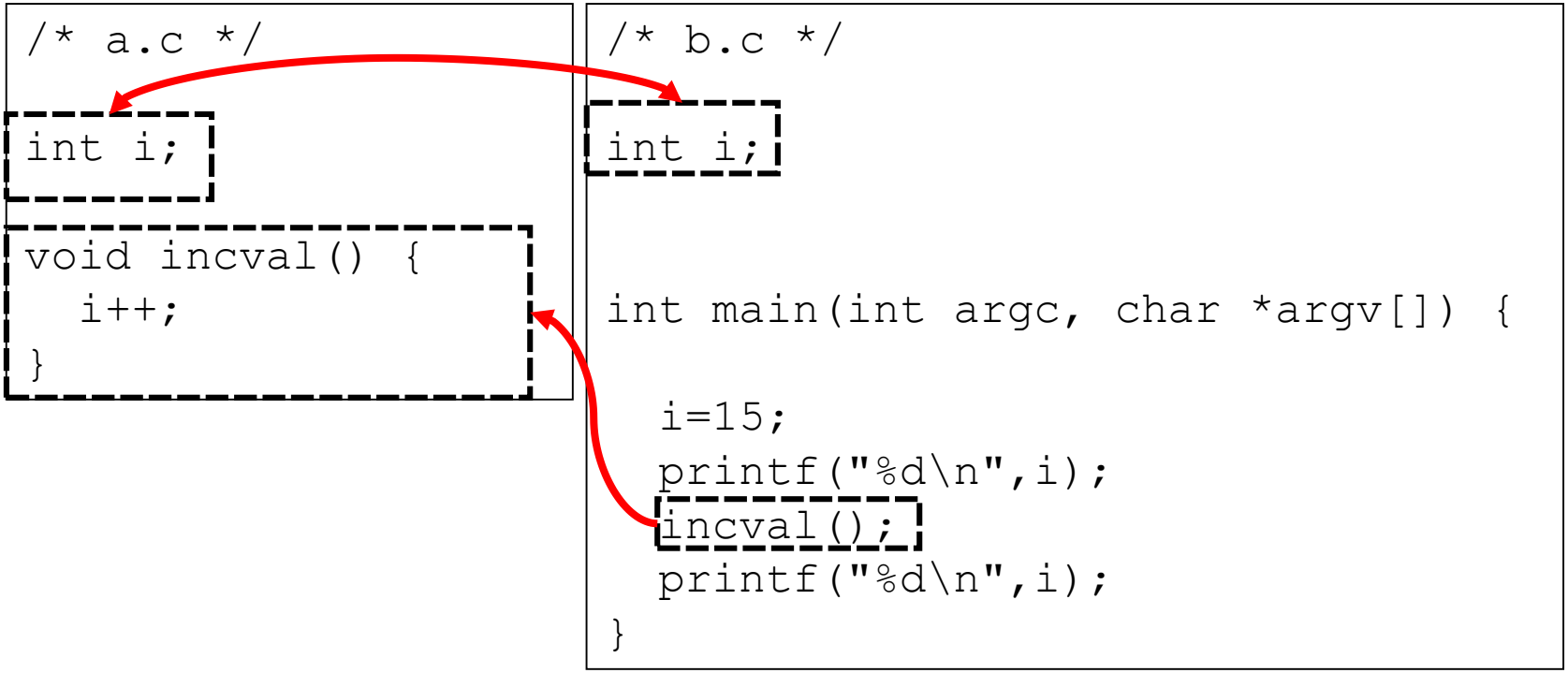



```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
15
16
$
```

```c
/* a.c */

int i;

void incval() {
  i++;
}
```

```c
/* b.c */

int i;
void incval();

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

  i=15;
  printf("%d\n",i);
  incval();
  printf("%d\n",i);
}
```

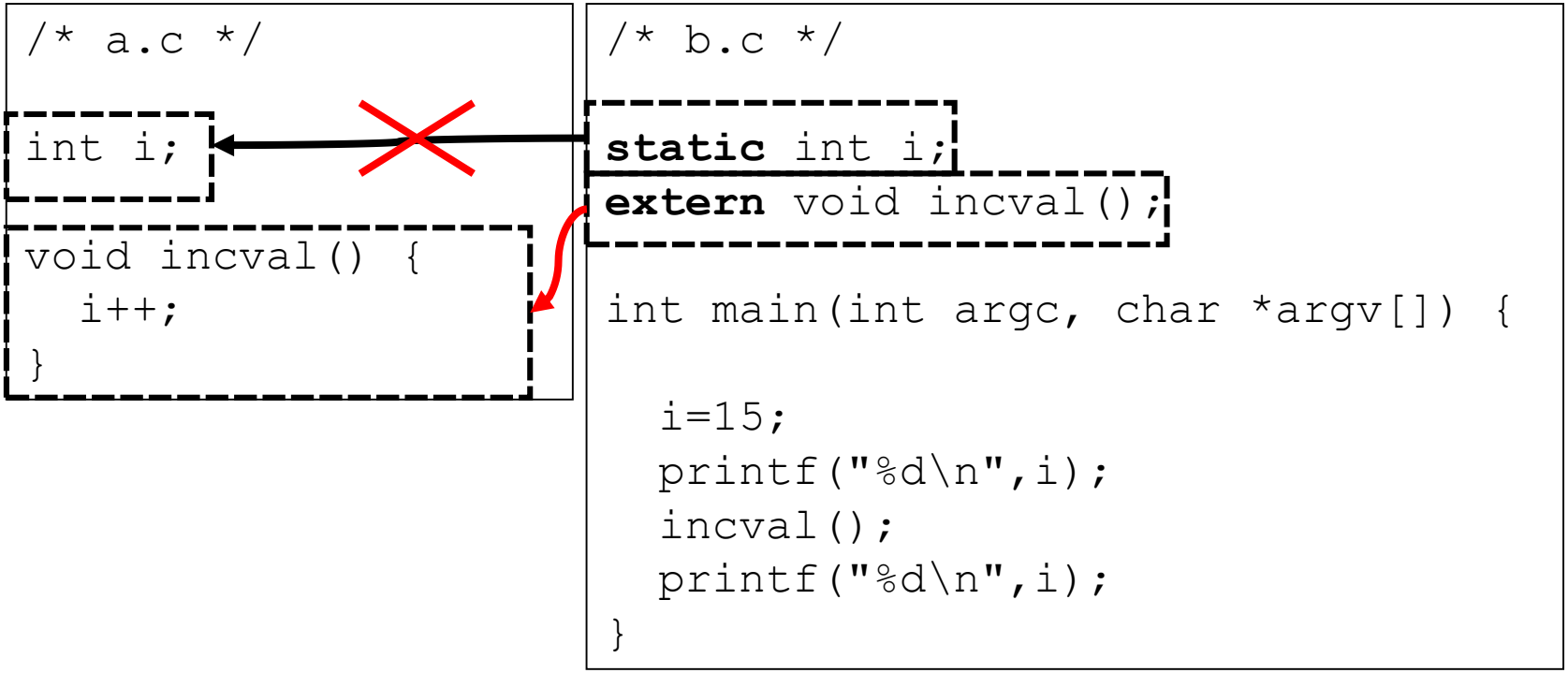
```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
15
16
$
```



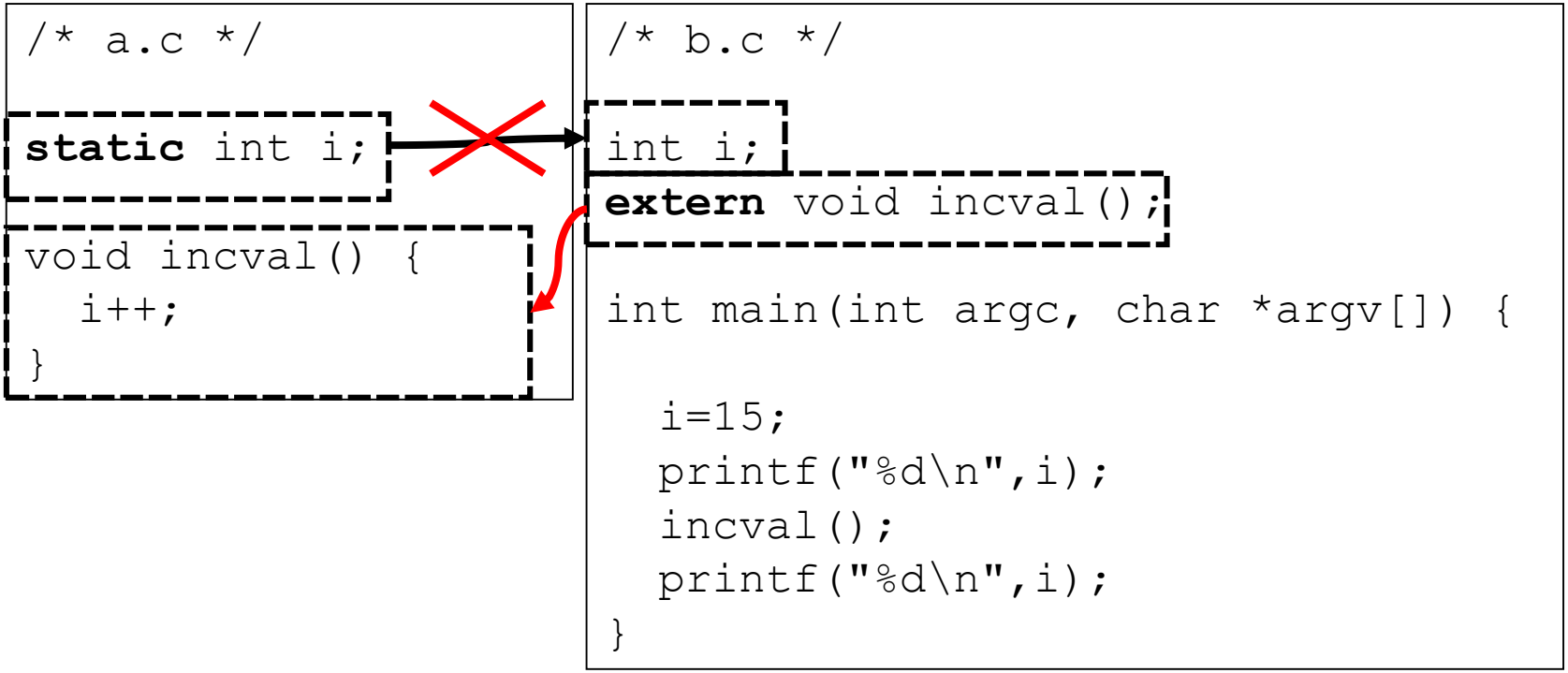

```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
$./test
15
16
$
```

Περισσότερα για το `static`

- Πως αποφεύγουμε μια καθολική μεταβλητή ή/και συνάρτηση που υλοποιούμε αποκλειστικά για τους σκοπούς του τοπικού κώδικα να «προσπελαστεί» (κατά λάθος) μέσα από κώδικα σε άλλο αρχείο;
- Η επιθυμητή **τοπικότητα** επιτυγχάνεται χρησιμοποιώντας τον προσδιορισμό `static`
  - η εμβέλεια των καθολικών μεταβλητών και συναρτήσεων περιορίζεται στο αρχείο που δηλώνονται / υλοποιούνται
- Είναι **αδύνατο** να γίνει αναφορά σε αυτές (είτε επίτηδες είτε κατά λάθος) μέσα από κώδικα που βρίσκεται σε ένα άλλο αρχείο




```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
$
$./test
15
15
$
```




```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
$
$./test
15
15
$
```

```
/* a.c */

static int i;

void incval() {
  i++;
}
```

```
/* b.c */

extern int i;
extern void incval();

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

  i=15;
  printf("%d\n",i);
  incval();
  printf("%d\n",i);
}
```

```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
undefined reference to i
$
```

```c
/* a.c */

int i;

static void incval() {
  i++;
}
```

```c
/* b.c */

extern int i;
extern void incval();

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

  i=15;
  printf("%d\n",i);
  incval();
  printf("%d\n",i);
}
```

```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
undefined reference to incval
$
```

```c
/* a.c */

int i;

static void incval() {
  i++;
}
```

```c
/* b.c */

extern int i;


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

  i=15;
  printf("%d\n",i);
  incval();
  printf("%d\n",i);
}
```

```
$ gcc a.c -c -o a.o
$ gcc b.c -c -o b.o
$ gcc a.o b.o -o test
undefined reference to incval
$
```

Ένα μικρό πρόβλημα ...

- Όταν ένας κώδικας Α ορίζει μεταβλητές και υλοποιεί συναρτήσεις **με σκοπό** αυτές **να χρησιμοποιηθούν** μέσα από οποιοδήποτε **άλλο κώδικα**, τότε αυτές δεν μπορεί να 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 οι εξωτερικές δηλώσεις του Β έχουν προκύψει από **λάθος** (π.χ. παράλειψη προσδιορισμού `static` σε δήλωση καθολικής μεταβλητής / συνάρτησης ή παράλειψη δήλωσης και υλοποίησης συνάρτησης), η διασύνδεση θα οδηγήσει σε **λάθος** αποτέλεσμα

Η ρίζα του προβλήματος

- Αναφορά σε εξωτερικές μεταβλητές / συναρτήσεις γίνεται με βάση έναν **επίπεδο χώρο ονομάτων**, όπου δεν μπορεί να αποκλεισθεί η τυχαία χρήση του ίδιου ονόματος από διαφορετικά τμήματα κώδικα
- Ο προγραμματιστής **δεν** έχει τη δυνατότητα να προσδιορίσει το τμήμα λογισμικού το οποίο θα πρέπει να παρέχει τον ορισμό / υλοποίηση μιας εξωτερικής μεταβλητής / συνάρτησης
- Άλλες γλώσσες **λύνουν** το πρόβλημα, δίνοντας σε κάθε τμήμα λογισμικού ένα **μοναδικό** όνομα με βάση το οποίο άλλα τμήματα κώδικα μπορεί να αναφέρονται (χωρίς πιθανότητα λάθους) σε μεταβλητές και συναρτήσεις που αυτό προσφέρει