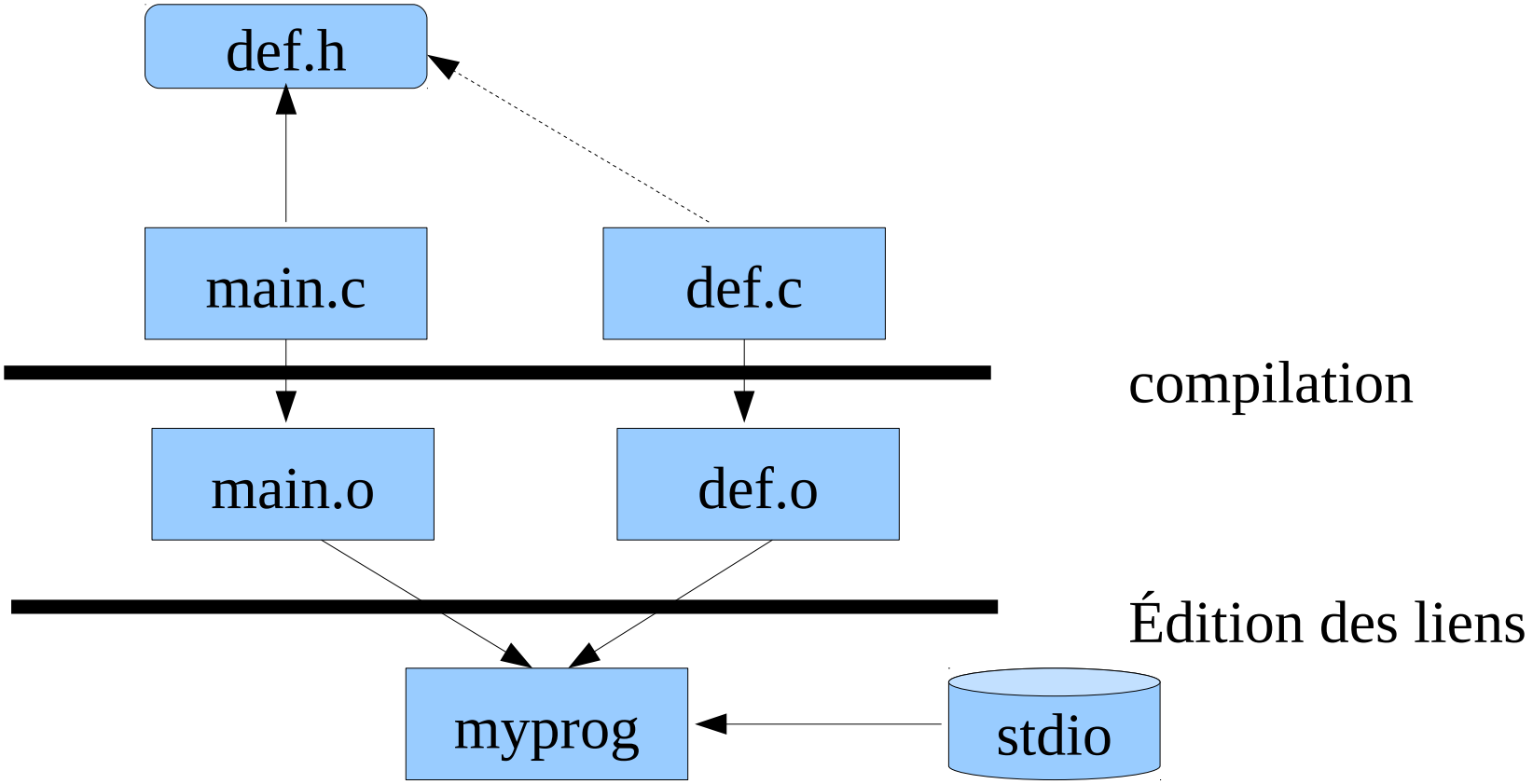
Compilation séparée

Compilation séparée

- La compilation séparée permet de **fragmenter** un grand programme en des parties appelées **modules** qui peuvent être compilées indépendamment les unes des autres.

- **La modularité** est basée sur un découpage en fichiers
  - **programme** = ensemble de fichiers (modules)
  - **fichier** = liste de déclarations
  - **déclaration** = déclarations de types
    - » déclaration de variables & définitions
    - » déclaration de fonctions & définitions
    - » directives du pré-processeur

Exemple



Compilation et édition des liens :

```
% gcc -c def.c
% gcc main.c def.o -o myprog
```

Notion de visibilité

- La compilation séparée des différentes parties d'un programme implique le respect de certaines règles dites **Règles de visibilité**.

  Ces règles s'appliquent aux noms de variables et de fonctions.

- Lors de la phase de compilation, le compilateur doit connaître le type et l'adresse de toute variable ou fonction qu'il rencontre.

- **Règle fondamentale** : *toute variable ou fonction doit être déclarée avant d'être utilisée.*

Visibilité de noms

```
int a ;
void f1(void)
{ long a ;
  a = 50 ;
}
void f2(void)
{
 a = 10 ;
}
Void f3(float a)
{
 a = 10 ;
}
Void f4(int c)
{
 a = 10 ;
}
```

Exemple de règles de visibilité :

1. la fonction f4 peut appeler les fonctions f4, f3, f2, et f1 ;
2. la fonction f3 peut appeler les fonctions f3, f2, et f1 ;
3. la fonction f2 peut appeler la fonction f2, et f1 ;
4. la fonction f1 ne peut que s'appeler elle-même ;

Les noms des variables locales peuvent masquer les celles des variables globales :

a) Modification locale de variable locale "a" dans f1() :
   - L'affectation a = 50 réalisée dans f1() modifie la variable locale et non la variable globale ;

b) Modification globale de la variable "a" dans f2() :
   - L'affection a = 10 réalisée dans la fonction f2() modifie la variable globale ;

c) Modification du paramètre local "a" dans f3() :
   - L'affectation a = 10 réalisée dans f3() modifie l'argument local de type float et non la variable globale.

Extension de la visibilité de noms

- Il est possible de déclarer une variable ou une fonction **externe,** afin de l'utiliser dans un module différent de celui où elle est définie.

- Dans le cas d'une variable, le compilateur accepte son utilisation et fait les contrôles de cohérence sur son type.

- Le compilateur demande à l'éditeur de liens de retrouver la variable dans les autres modules. Ces déclarations se placent aussi bien au niveau global qu'à l'intérieur des blocs.

- Une telle variable doit être définie dans un autre fichier du programme et tous les fichiers doivent être associés par **une édition de liens**, sinon cette dernière se termine avec des références non résolues et le fichier binaire exécutable n'est pas produit.

Visibilité des variables entre modules

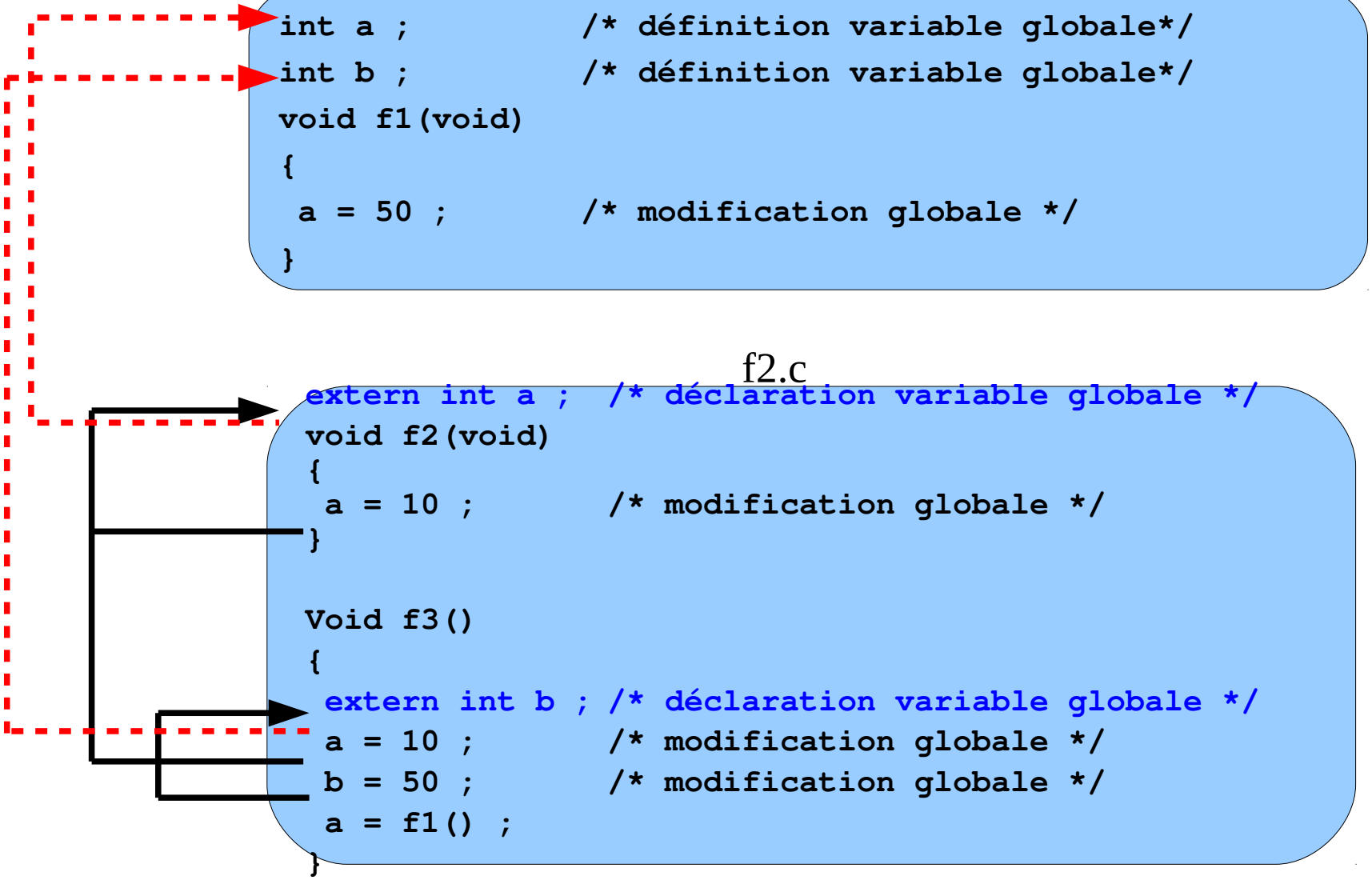
f1.c

```
int a ;          /* définition variable globale*/
int b ;          /* définition variable globale*/
void f1(void)
{
 a = 50 ;        /* modification globale */
}
```

f2.c

```
extern int a ;  /* déclaration variable globale */
void f2(void)
{
 a = 10 ;        /* modification globale */
}

Void f3()
{
 extern int b ; /* déclaration variable globale */
 a = 10 ;        /* modification globale */
 b = 50 ;        /* modification globale */
 a = f1() ;
}
```



Edition des liens ----> Compilation ——>

Visibilité des fonctions entre modules

Prg1.c

```
int a ;
int f1(void)    /* définition f1() */
{
 a = 50 ;
 return a ;
}
```

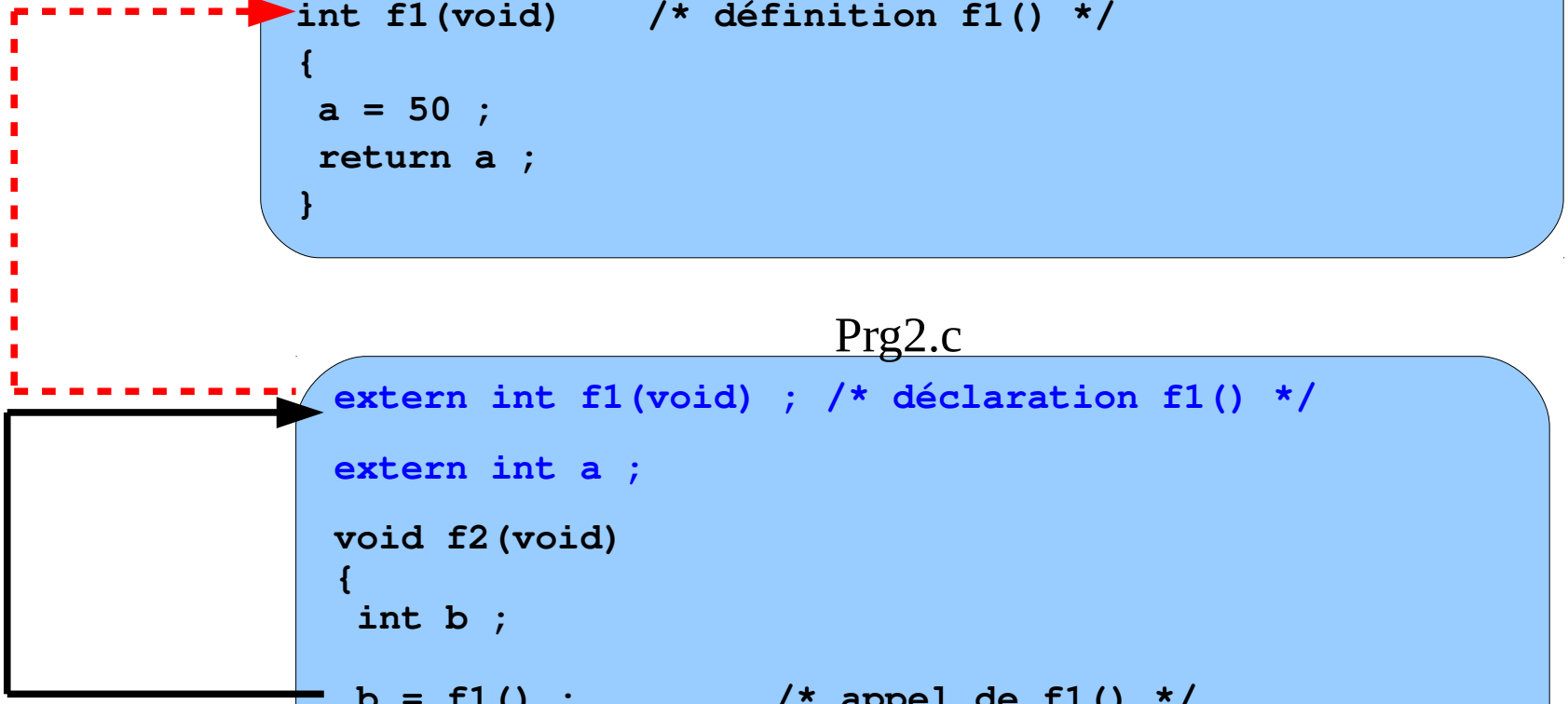
Prg2.c

```
extern int f1(void) ; /* déclaration f1() */

extern int a ;

void f2(void)
{
 int b ;

 b = f1() ;          /* appel de f1() */
}
```



Edition des liens -----> Compilation ——>

Fichiers d'inclusion

- Les fichiers d’inclusion sont destinés à regrouper des déclarations d’objets des types suivants :
  - types non prédéfinis,
  - modèles et noms de structures,
  - types et noms de variables,
  - Prototypes de fonctions.

- Les fichiers d’inclusion contiennent les déclarations des variables et fonctions utilisées par plusieurs modules.

Utilisation de fichiers d'inclusion

```
int f1(void)
{
 int a = 50 ;
 return a ;
}
```

p1.c

```
int f3(void)
{
 int b = 10 ;
 return b ;
}
```

p2.c

```
int c ;
int f1(void) ;
int f3(void) ;
```

def.h

```
include "def.h"      /* inclusion def.h */
int main(void)
{
 a = 10 ;                /* modification globale */
 a = f1() ;
 b = f3() ;
 return 0 ;
}
```

main.c

Réduction de la visibilité

- Utilité de restreindre l'accès de variables ou de fonctions pour permettre une meilleure utilisation des interfaces entre modules sans permettre l'accès au fonctionnement interne d'un module.

- En langage C, le prédicat **static** permet de masquer des noms de variables ou de fonctions aux autres fichiers du programme.

- Une variable de type **global static** n'est visible que du module qui la déclare. Elle n'est accessible qu'aux fonctions définies après elle dans le même fichier.

- Permet de manipuler des données qui ont le même nom dans plusieurs fichiers avec des définitions différentes dans chacun d'entre-eux.

Réduction de visibilité

```
int a ;                 /* variable globale du programme */
Static int b,c ;        /* variables globales dans prg1.c */
int f1(void)
{
 a = 50 ; b = 10 ;
 return a ;
}
```

prg1.c

```
extern int a, c ;          /* déclaration variable globale*/
extern int f1(), f3() ; /* déclaration fonctions globales*/
```

defs.h

```
 include "defs.h"      /* inclusion defs.h */
 Static int f2()       /* fonction utilisable dans prg3.c*/
 {
  a = 10 ;             /* modification globale */
  c = f1() ;           /* variable du fichier prg3.c*/
  a = f3() ;
 }
 int c ;
 int f3()
 {
  a = 10 ;             /* modification globale */
  c = 10 ;             /* variable du fichier prg3.c*/
  /*b = 50;*/          /* impossible*/
 }
```

prg3.c