



# Chaine de compilation

---

L'exemple du langage C

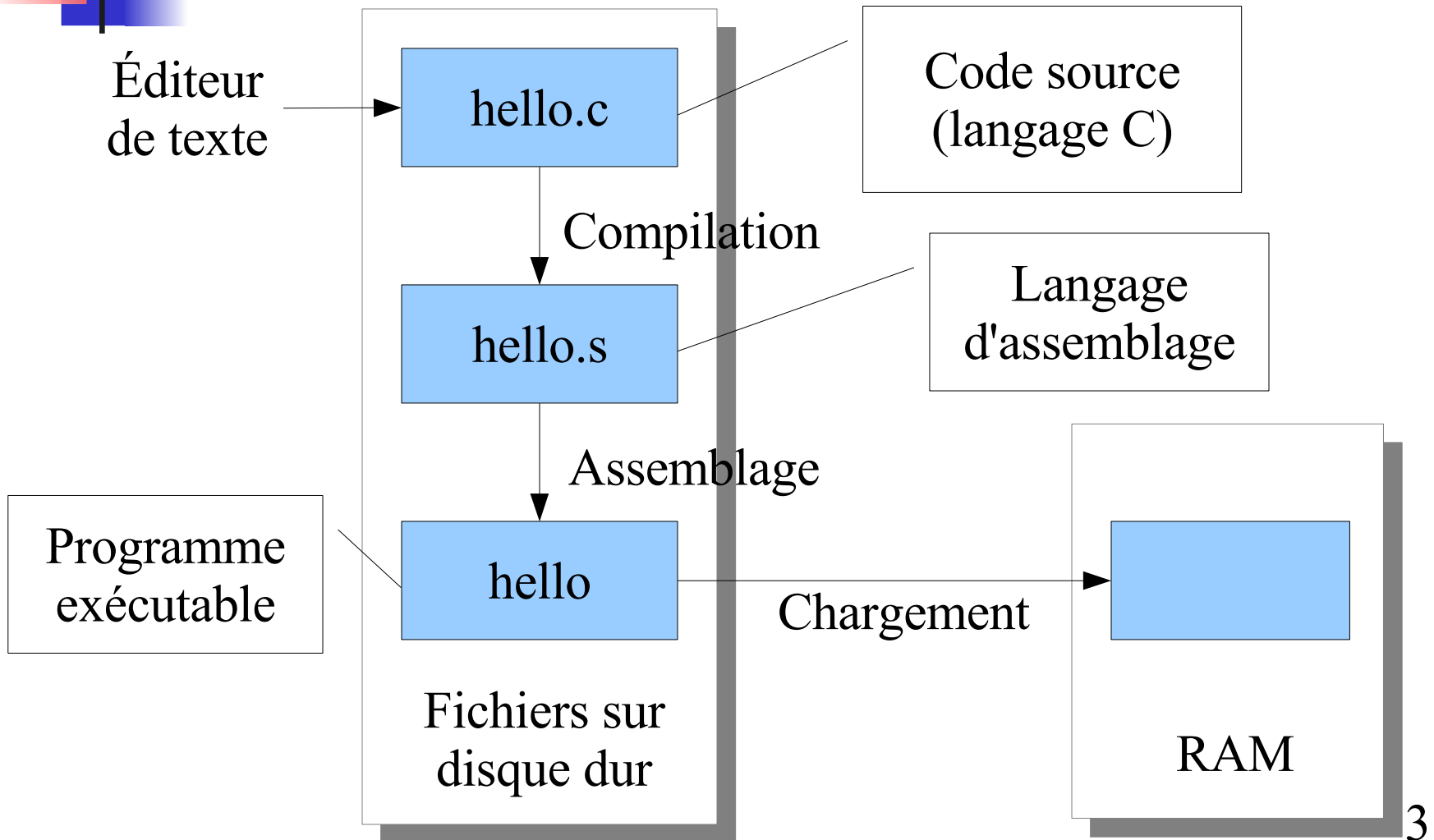


# Compilation d'un fichier

---

- Code source (langage C)
  - Lisible par un humain
  - portable
- Code machine (binaire) :
  - exécutable par la machine
- Intermédiaire : Assembleur
  - Peu agréable pour l'humain, mais lisible
  - Dépendant de la machine cible

# Compilation d'un fichier





# Compilation d'un fichier : Les commandes (avec GCC)

---

- Compilation (C vers assembleur) :
  - `gcc -S fichier.c` (crée fichier.s)
- Assemblage (Assembleur vers exécutable) :
  - `gcc fichier.s -o fichier`
- Tout d'un coup (GCC fait toutes toutes les étapes) :
  - `gcc fichier.c -o fichier`

# Programmes utilisant plusieurs fichiers



---

- Motivations :
  - Structurer le programme
    - Exemple paquetages Ada
    - 1 fichier = un ensemble de choses (fonctions, objets, structures, ...) réalisant une fonctionnalité
  - Faciliter le travail à plusieurs
    - Répartition du travail
    - Éviter les conflits (deux personnes modifient le même fichier en même temps)
  - Compilation séparée ...



# Programmes multi-fichiers : Compilation séparée

---

- À chaque modification du programme, ne recompiler “que ce qui est nécessaire” :
  - Recompiler
    - les fichiers modifiés
    - les fichiers qui utilisent quelque chose dont l'interface a été modifiée (structure de données par exemple)
  - Régénérer le programme exécutable (édition de lien)

# Programmes multi-fichiers en langage C



---

- Chaque module peut avoir besoin de savoir « comment utiliser les autres »
  - Séparation implémentation/interface
  - Implémentation : fichier.c
  - Interface : fichier.h
  - Utilisation d'un autre module :  
`#include "fichier.h"`

# Multi-fichiers en C : définition de module

- `bonjour.c` :

```
void dire_bonjour() {  
    printf("Hello\n");  
}
```

- => La fonction est implémentée

- `Bonjour.h` :

```
#ifndef BONJOUR_H  
#define BONJOUR_H  
  
void dire_bonjour();  
  
#endif /* BONJOUR_H */
```

- => On signale seulement l'existence de la fonction

# Multi-fichiers en C :

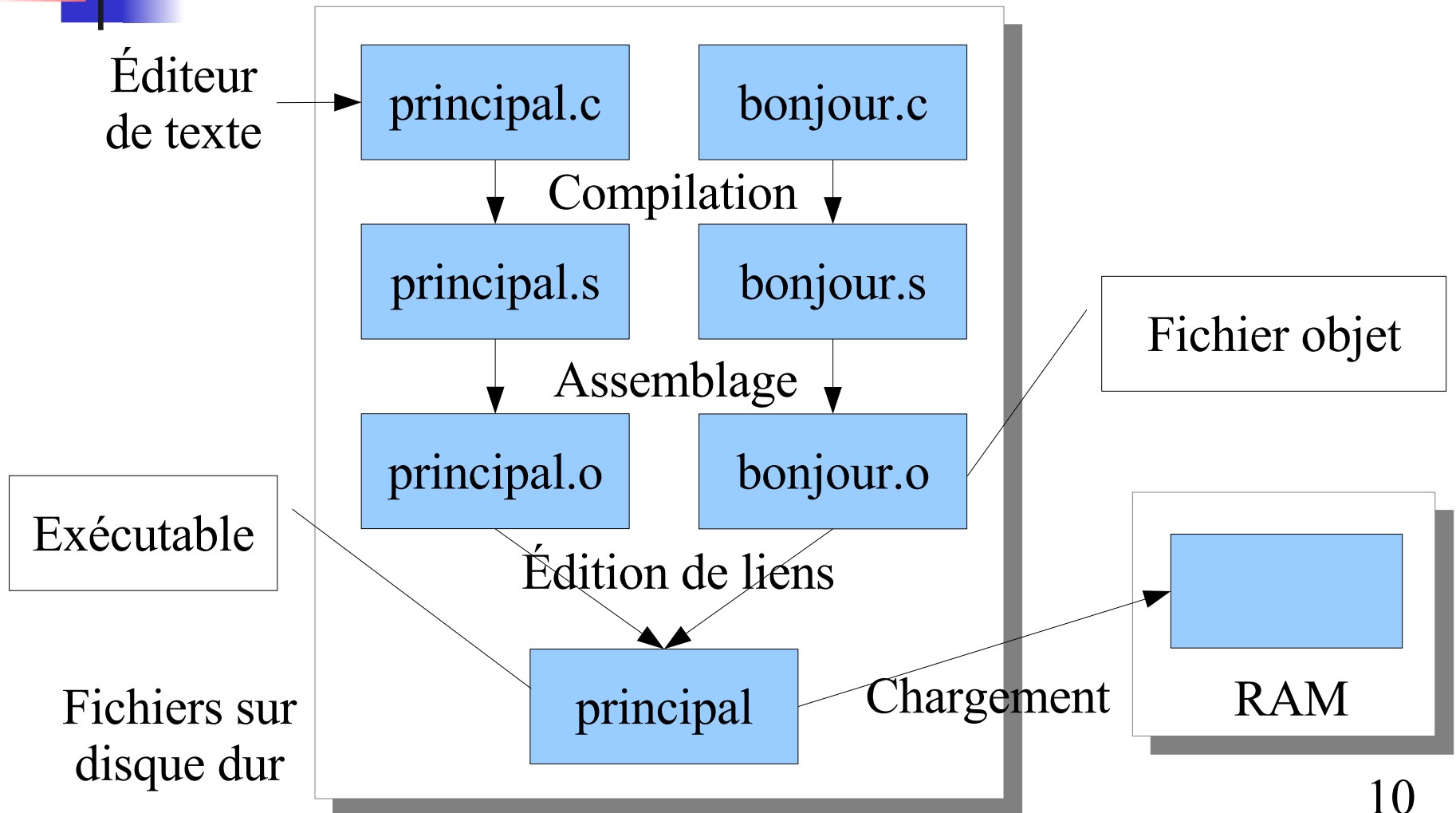
## Utilisation de module

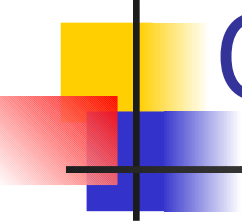
---

- principal.c :

```
#include "bonjour.h"
/* Maintenant, le compilateur sait que
   « dire_bonjour() » existe */
int main () {
    dire_bonjour();
    return 0;
}
```

# Multi-fichier en C : Compilation (principe)





# Multi-fichier en C :

## Compilation (en pratique)

---

- Compilation et assemblage d'un fichier (C vers fichier objet) :
  - `gcc -c fichier.c`
- Édition de lien :
  - `gcc fichier1.o fichier2.o ... -o executable`
- Sur notre exemple :
  - `gcc -c principal.c`
  - `gcc -c bonjour.c`
  - `gcc bonjour.o principal.o -o principal`

# Multi-fichier en C :

## Automatisation avec « make »

---

- « Make » : outil utile pour la compilation
- Utilise le fichier appelé « Makefile » dans le répertoire courant.
- Le Makefile dit à make
  - Quand recompiler un fichier
  - Comment le recompiler
- Makefile = ensemble de règles  
*cible: dépendances*  
*<-- tabulation -->action (commande shell)*
- Une dépendance peut appeler une autre cible



# Exemple de Makefile

---

- Makefile :

```
principal: bonjour.o principal.o
```

```
    gcc bonjour.o principal.o -o principal
```

```
bonjour.o: bonjour.c bonjour.h
```

```
    gcc -c bonjour.c
```

```
principal.o: principal.c bonjour.h
```

```
    gcc -c principal.c
```

- Utilisation :

```
telesun> make principal
```

```
gcc -c bonjour.c
```

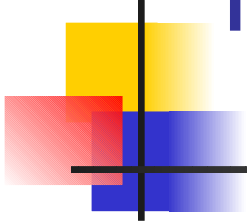
```
gcc -c principal.c
```

```
gcc bonjour.o principal.o -o principal
```

```
telesun> make principal
```

```
make: `principal' is up to date.
```

Fin





# Bibliothèques partagées

---

- Bibliothèque = code exécutable + interface
  - En C, 1 bibliothèque = 1 fichier binaire + 1 ou plusieurs fichiers « .h »
- Plusieurs programmes peuvent utiliser la même bibliothèque (stockée une seule fois sur le disque) :
  - Unix : libXYZ.so (« Shared object »)
  - Windows : XYZ.dll (« Dynamic link library »)

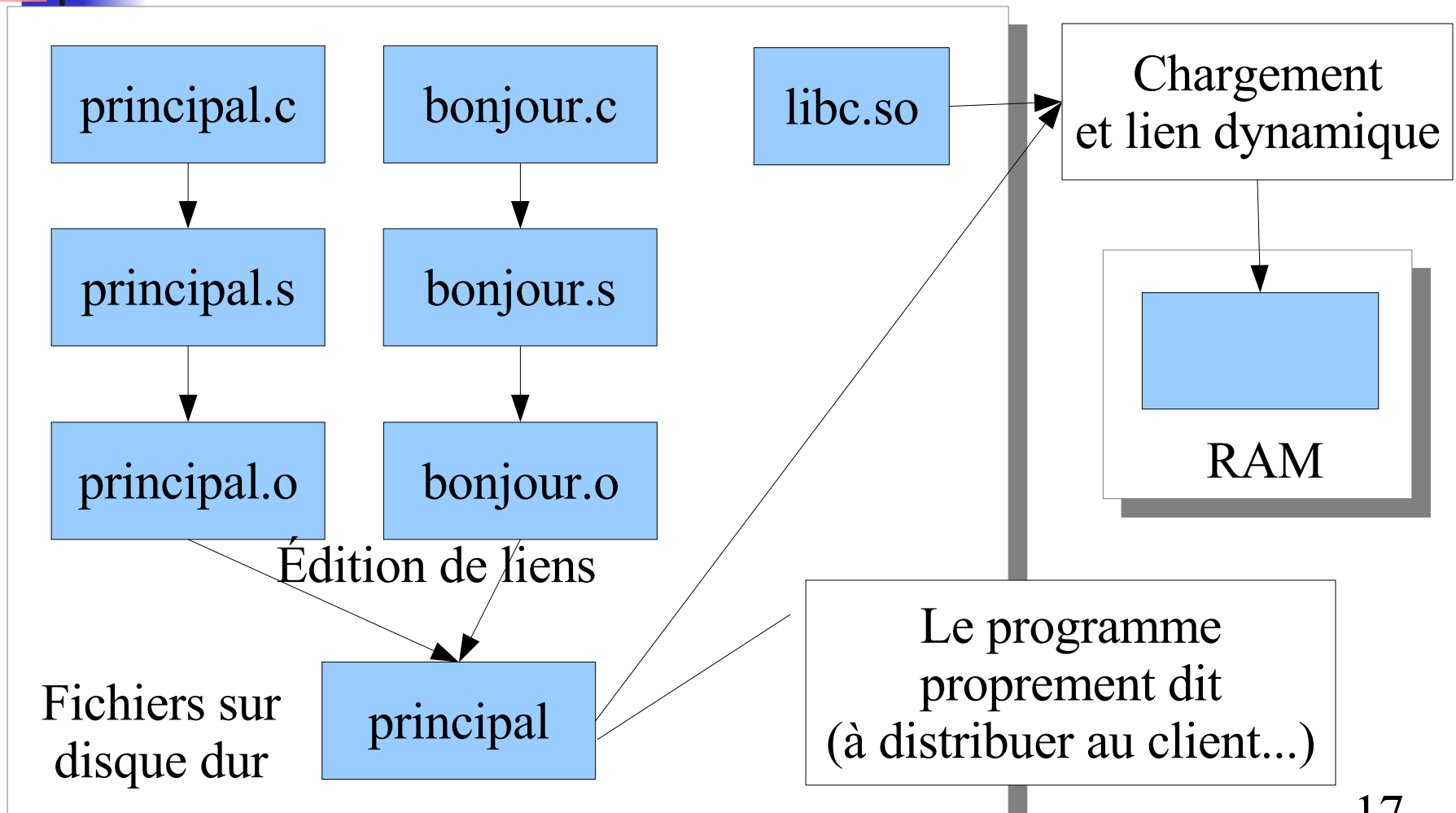
# Utilisation d'une bibliothèque partagée



---

- À la compilation :
  - Le compilateur utilise les fichiers « .h » (exemple : `stdio.h`)
  - Le binaire généré contient des symboles non-définis (exemple : `printf`), et sait dans quelle bibliothèque les trouver
- Au chargement :
  - Le chargeur dynamique va chercher les symboles non définis dans la bibliothèque (exemple : `libc.so`)

# Utilisation d'une bibliothèque partagée





# A propos de ce document ...

---

- Auteur original :
- Matthieu Moy, juin 2007