Introduction à la compilation séparée

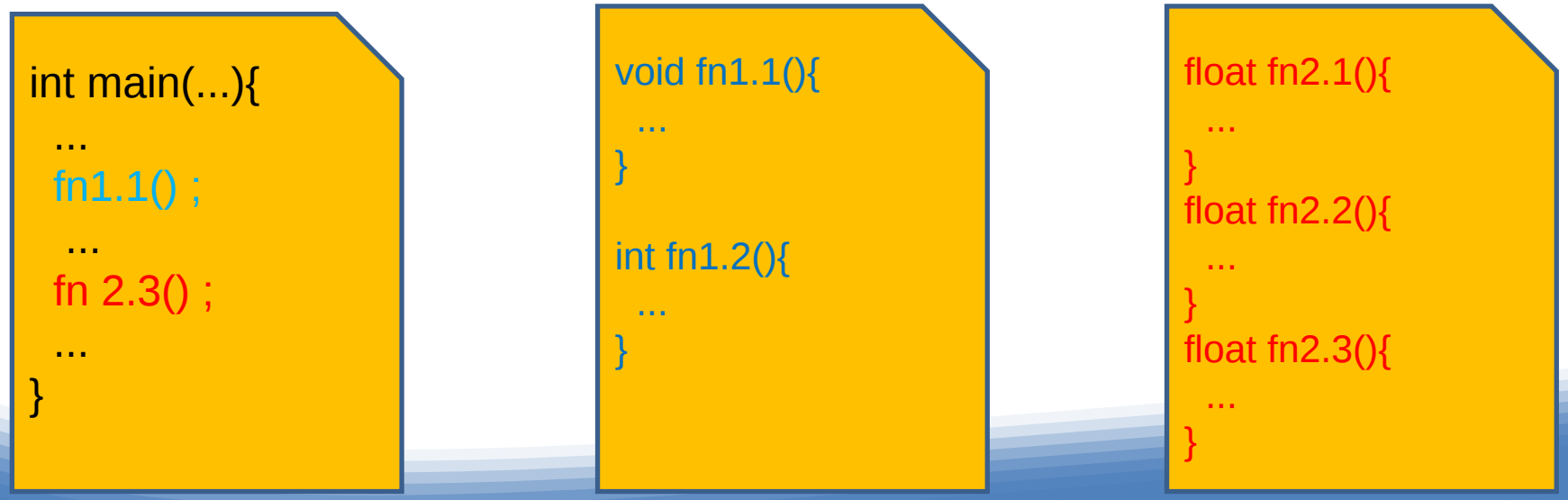
Licence 2 Informatique
C. Renaud
Année 2023-2024

Problématique

- Développement dans un seul fichier
  - Pratique en cas d’application de petite taille
  - Complexe pour application de grande taille
    - Fichier volumineux
    - Contenu peu lisible
    - Code peu réutilisable
    - Difficulté de maintenance
    - Compilation longue

Une solution

- Séparer le code en plusieurs fichiers
  - Un fichier contient des fonctions ayant un lien entre elles



fichier principal

fichier de fonctions 1

fichier de fonctions 2

Exemple

- Application
  - Recherche du plus court chemin dans un graphe pondéré

```
int main(...){
  ...
  creermatrice(...) ;
   ...
  rechercheCheminMinimum(...) ;
  ...
}
```

tp5.cpp

```
void creerMatrice(..){
  ...
}

...
```

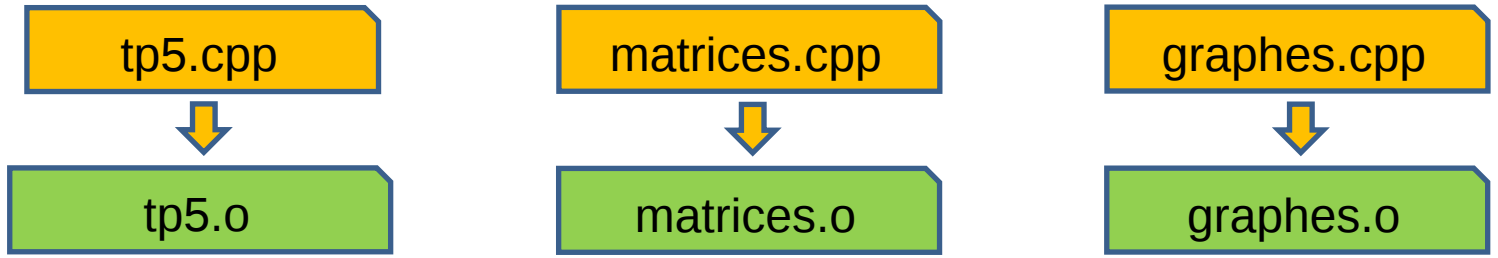
matrices.cpp

```
...

void rechercheCheminMinimum(){
  ...
}

...
```

parcours.cpp

Principe de la compilation

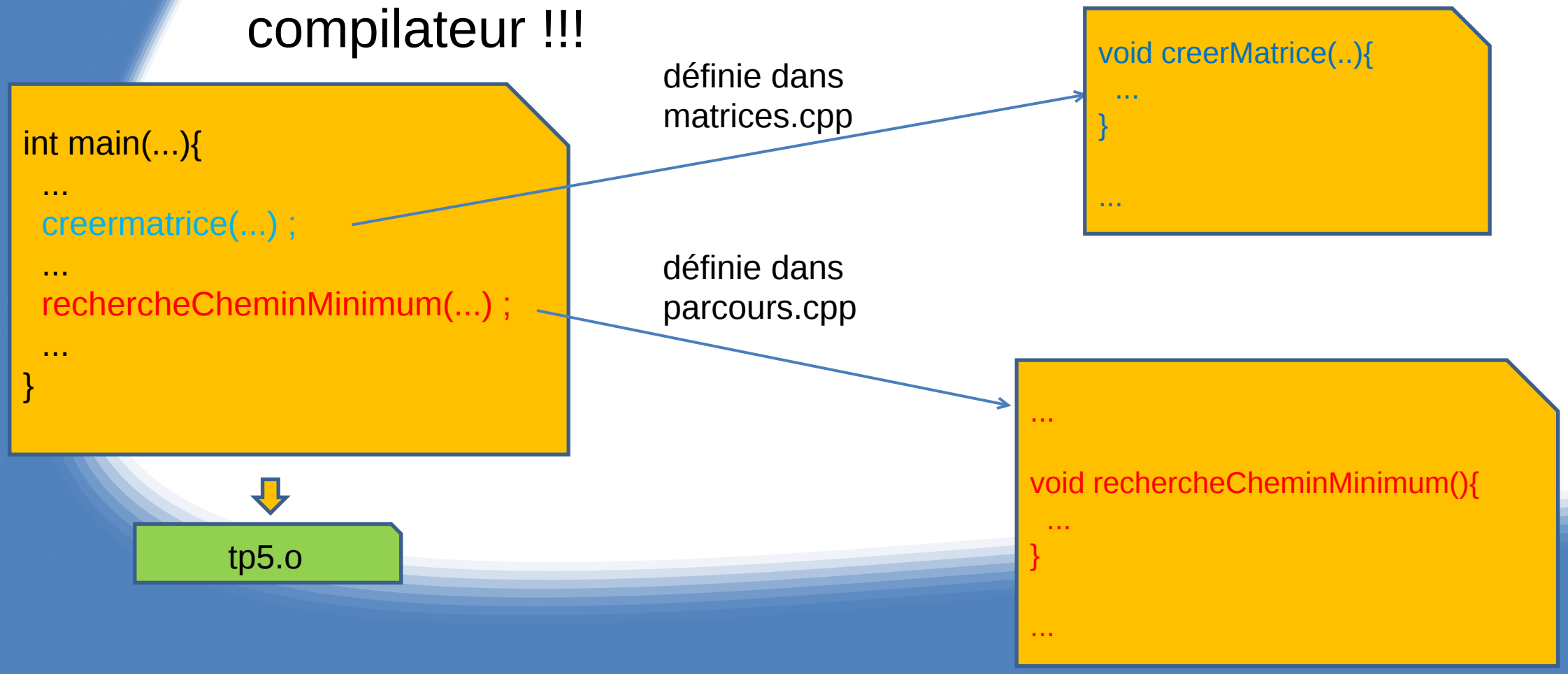
- On compile chaque fichier séparément
  - On génère un fichier « objet »



- On réunit chaque fichier « objet » en un fichier exécutable
  - Étape d'édition de liens

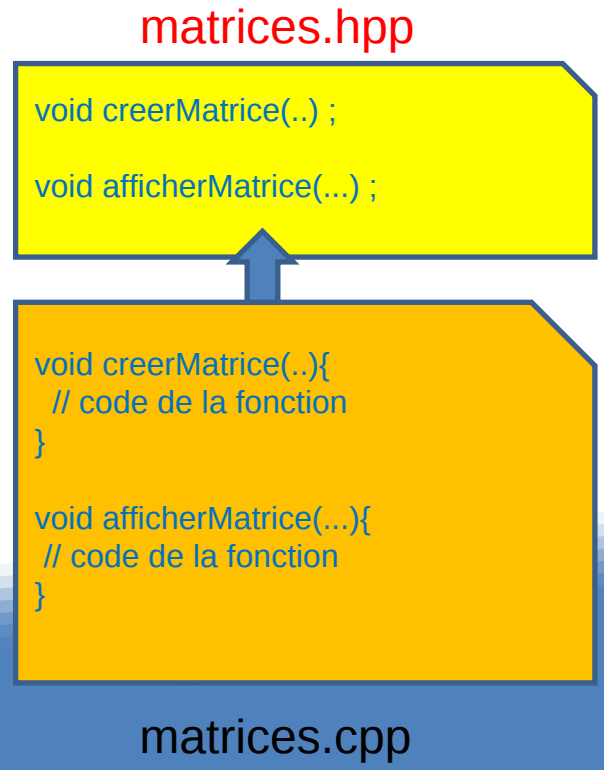
Problème

- Les fonctions extérieures ne sont pas connues du compilateur !!!



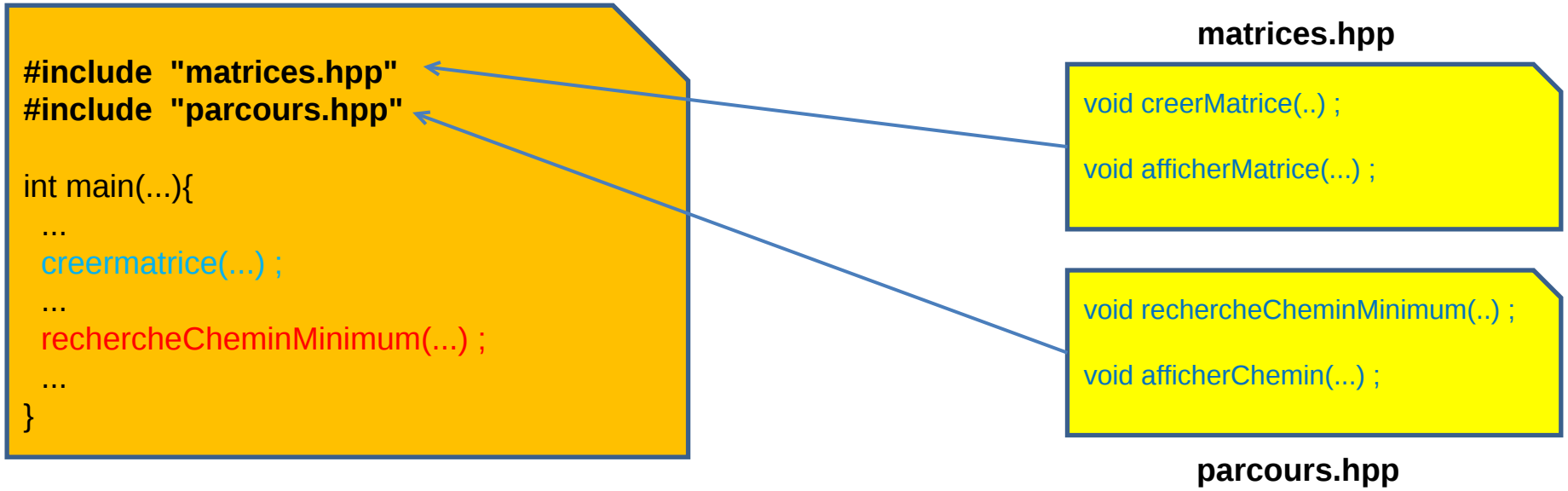
Résolution (1)

- Pour chaque module
  - (sauf le module principal)
  - Création d'un fichier contenant la définition des fonctions « exportées » par le module
  - Par convention :
    - Même nom que le module
    - Extension .hpp



Résolution (2)

- Dans le module principal
  - Inclure les fichiers « d'export » (.hpp) nécessaires



- Remarque
  - Idem si un module a besoin d'un fonction définie dans un autre module

Résolution (3)

- Liste finale des fichiers

matrices.hpp

```
void creerMatrice(..) ;

void afficherMatrice(...) ;
```

parcours.hpp

```
void rechercheCheminMinimum(..) ;

void afficherChemin(...) ;
```

```
#include  "matrices.hpp"
#include  "parcours.hpp"

int main(...){
  ...
  creermatrice(...) ;
  ...
  rechercheCheminMinimum(...) ;
  ...
}
```

```
void creerMatrice(..){
  ...
}

void afficherMatrice(...){
...
}
...
```

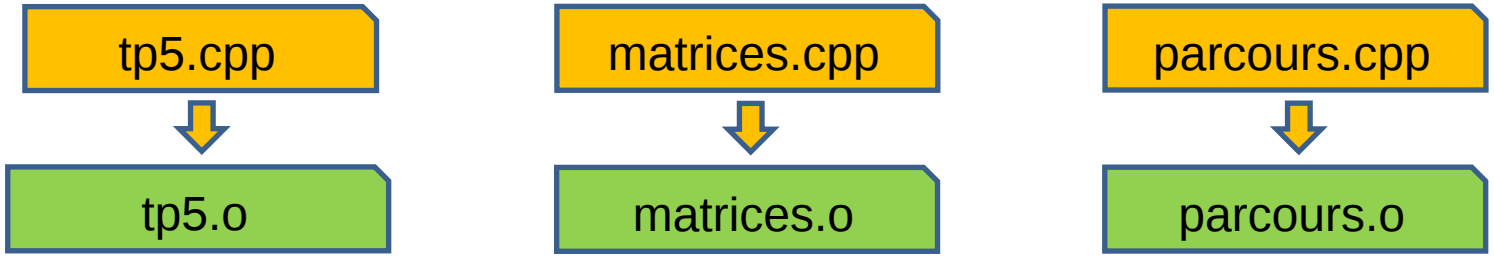
matrices.cpp

```
...
void rechercheCheminMinimum(...){
  ...
}

void afficherChemin(...){
  ...
}
...
```

parcours.cpp

Compilation (1)

- On compile chaque fichier séparément
  - On génère un fichier « objet »




```
g++  -c  tp5.cpp
g++  -c  matrices.cpp
g++  -c  parcours.cpp
```

option demandant la génération d'un fichier « objet »

Compilation (2)

- On réunit chaque fichier « objet » en un fichier exécutable
  - Étape d'édition de liens


```
g++    tp5.o  matrices.o  parcours.o  -o  tp5
```

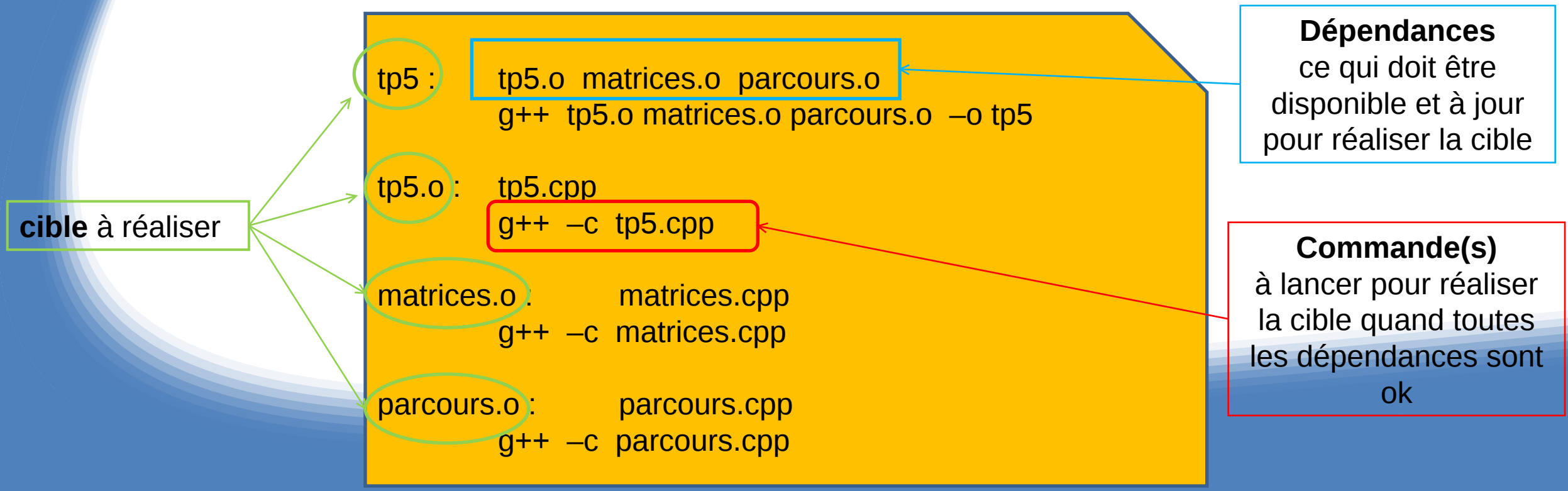
option demandant la génération d'un fichier exécutable à partir des fichiers objets fournis.

Compilation (3)

- Modification d’un module de code
  - Ce module doit être recompilé
  - Les modules non modifiés ne doivent pas être recompilés
  - L’édition de liens doit être refaite

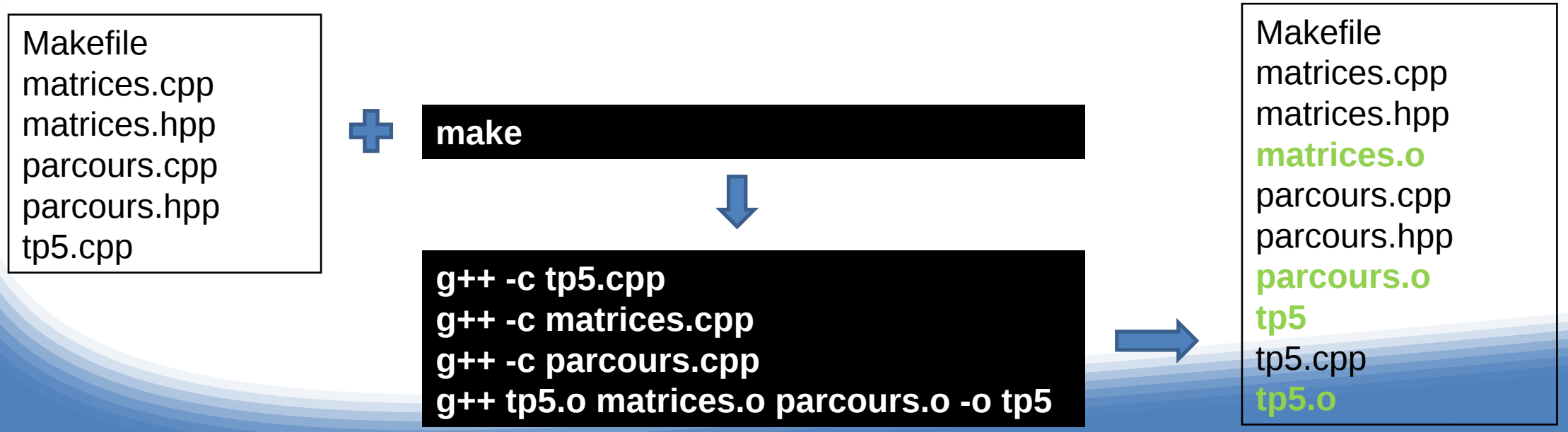
Compilation (4)

- Automatisation du suivi des modifications
  - Utilitaire make
    - Utilise un fichier nommé **Makefile**



Compilation (4)

- Automatisation du suivi des modifications
  - Utilitaire make
    - Lancement initial



Compilation (5)

- Automatisation du suivi des modifications
  - Utilitaire make
    - Lancement après une modification

Makefile
matrices.cpp
matrices.hpp
matrices.o
parcours.cpp
parcours.hpp
parcours.o
tp5
tp5.cpp
tp5.o

modification de parcours.cpp

make

g++ -c parcours.cpp
g++ tp5.o matrices.o parcours.o -o tp5

Makefile
matrices.cpp
matrices.hpp
matrices.o
parcours.cpp
parcours.hpp
parcours.o
tp5
tp5.cpp
tp5.o

Compilation (6)

- Automatisation du suivi des modifications
  - Utilitaire make
    - **ATTENTION** aux tabulations

```
tp5 : tp5.o  matrices.o  parcours.o
	g++  tp5.o matrices.o parcours.o  –o tp5

tp5.o : tp5.cpp
	g++  –c  tp5.cpp

matrices.o : matrices.cpp
	g++  –c  matrices.cpp

parcours.o : parcours.cpp
	g++  –c  parcours.cpp
```