3.2 – Librerie e linking

Libro di testo:

- Capitolo 2.4, 2.5

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- File header
- Genesi di una libreria
- Il ruolo del linker
- Linking statico e dinamico

File header

- Molte funzioni che usiamo non sono scritte da noi
  - Es: `sqrt()`, operatore `<<` su `cout`
- Serve uno strumento per importare funzioni scritte da altri (librerie)
- Un file header è un **insieme di dichiarazioni**

  (alcune delle quali definizioni) di entità
  - Tali entità sono utilizzabili da chi include il file header
  - Es: `math.h` dichiara `double sqrt(double x);`
- Questo è il file header visto dalla prospettiva dell'utente

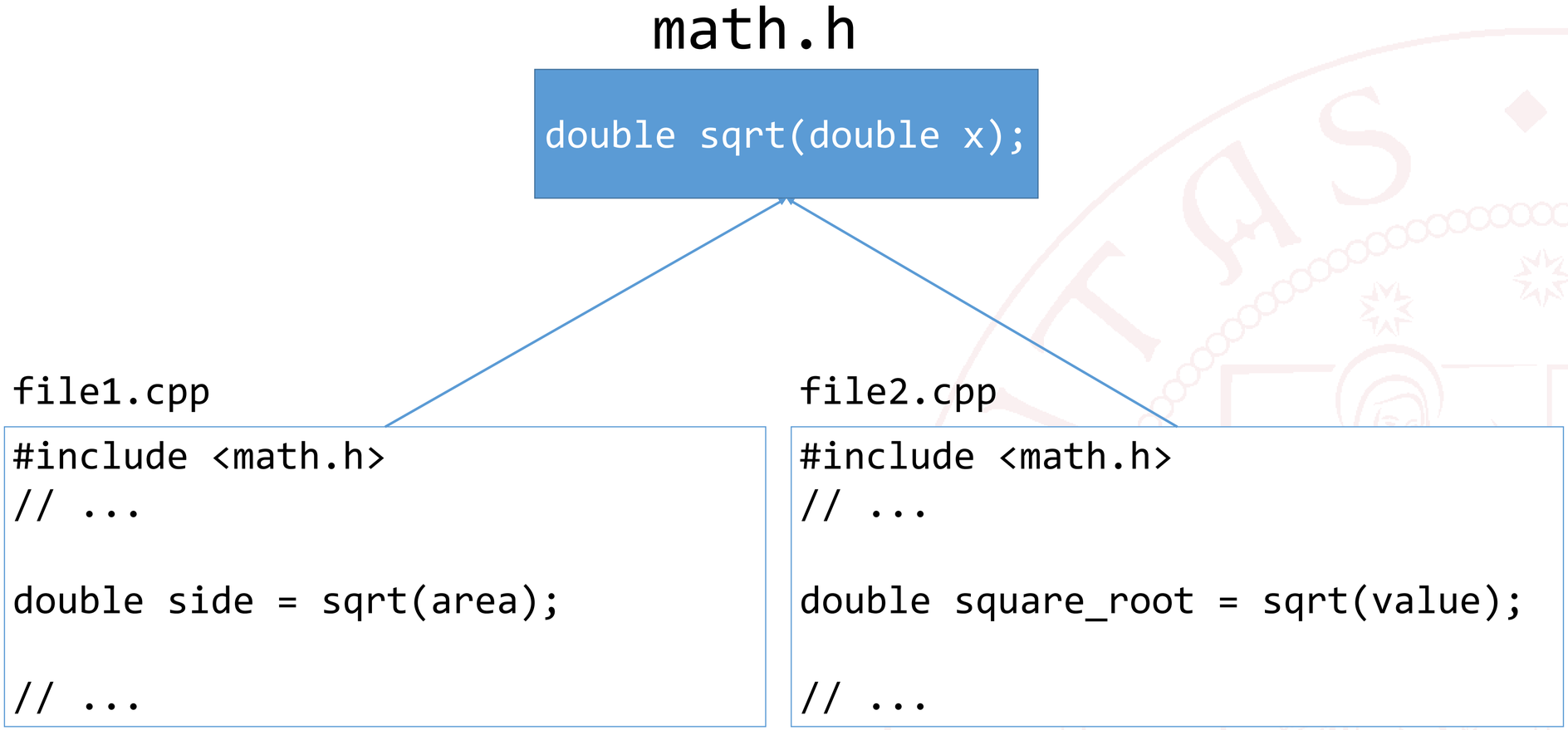
File header

- Due meccanismi di inclusione:
  - Header di sistema, inclusi con < >
    - Es: `#include <iostream>`
  - Header definiti dall'utente, inclusi con " "
    - Es: `#include "my_header.h"`

      `#include "opencv2/core.h"`
- Differenza nel path di ricerca del file header
  - path di sistema (dipende dal sistema) vs. path locale + path di sistema

File header

- Esempio di uso dell'header lato utente



Funzioni, header, compilazione

Funzioni in C++

main.cpp

```
int f(int i);

int main(void)
{
  int i = 0;

  i = f(i);

  return 0;
}

int f(int i)
{
  return i + 2;
}
```

Dichiarazione

Chiamata

Definizione

Progetti software in file multipli

- Progetti grandi prevedono:
  - Molte dichiarazioni
  - Molte definizioni
- Necessità:
  - Raggruppare e spostare altrove le dichiarazioni
  - Raggruppare e spostare altrove le definizioni
- Questo porta a:
  - Avere più di un file sorgente
  - Creare header
- Questo è il file header visto dalla prospettiva del progettista

Progetti software in file multipli

- Come possiamo distribuire un SW in molti file?

main.cpp

```
int f(int i);

int main(void)
{
  int i = 0;

  i = f(i);

  return 0;
}
```

Dichiarazione → Header file (`my_func.h`)

Chiamata

```
int f(int i)
{
  return i + 2;
}
```

Definizione → File sorgente di libreria (`my_func.cpp`)

Progetti software in file multipli

- Come possiamo distribuire un SW in molti file?

main.cpp

```
int f(int i);

int main(void)
{
  int i = 0;

  i = f(i);

  return 0;
}
```

Dichiarazione

Chiamata

```
int f(int i)
{
  return i + 2;
}
```

Definizione

Libreria

my_func.h

```
int f(int i);
```

Eseguibile

main.cpp

```
#include "my_func.h"

int main(void) {
  int i = 0;
  i = f(i);
  return 0;
}
```

my_func.cpp

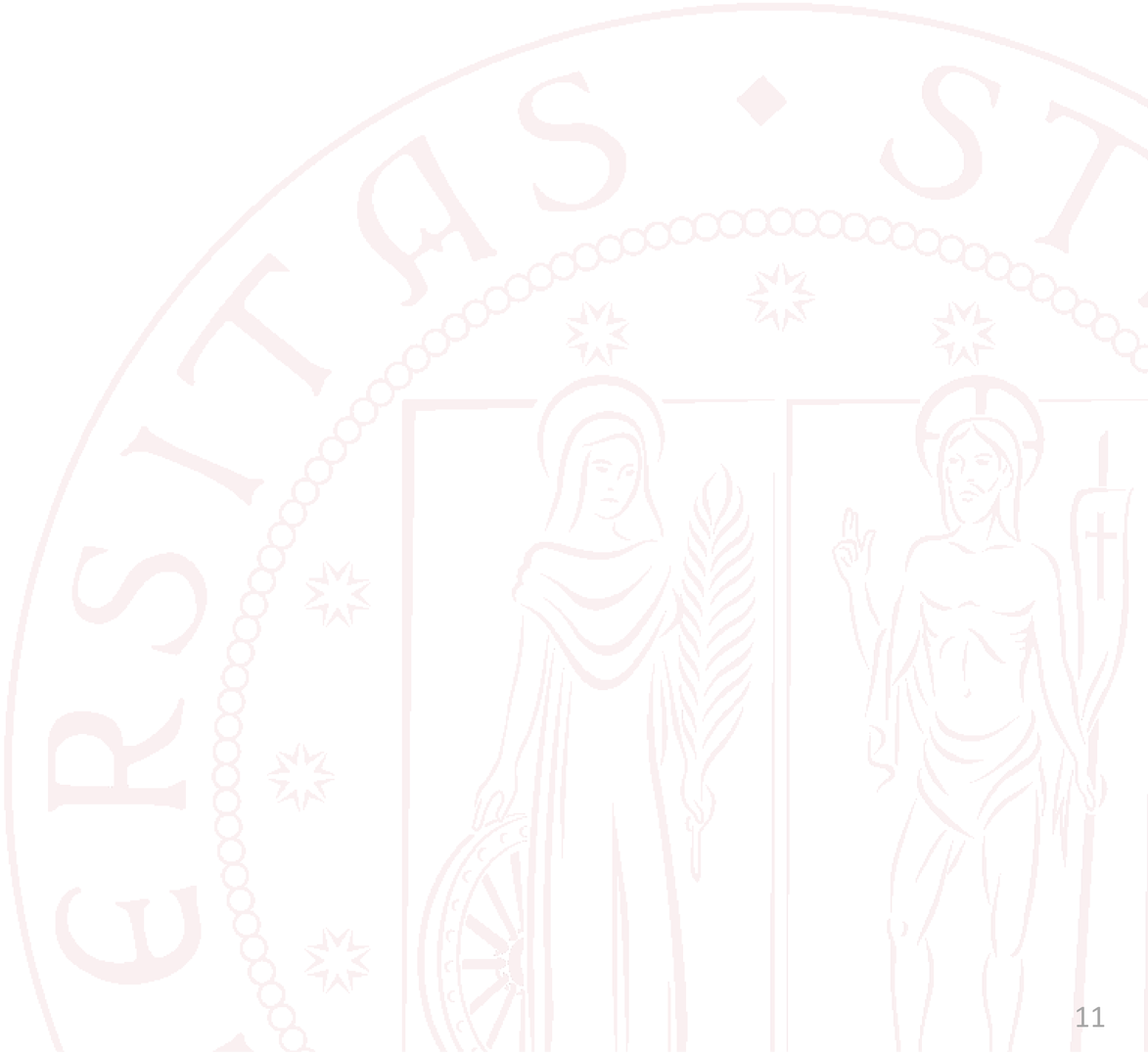
```
#include "my_func.h"

int f(int i) {
  return i + 2;
}
```

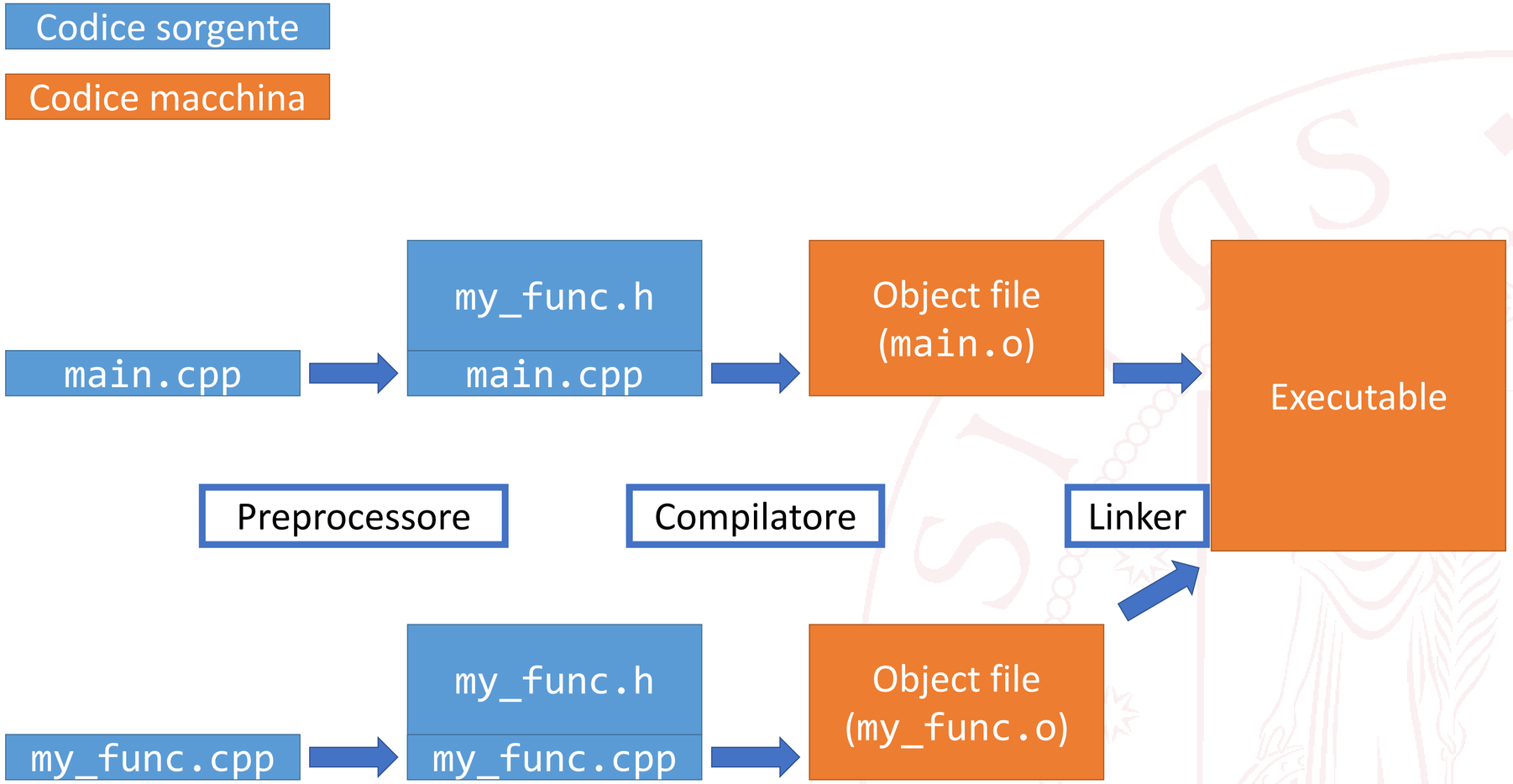
C’è un errore! Chi lo trova?

Progetti software in file multipli

- Come compilare un progetto composto da molti file?
- Preprocessore, compilatore, linker!
  - Ora agiscono su più file
- Vediamo in dettaglio come…



Da sorgente a eseguibile



Librerie esterne

- Caso precedente: un progetto diviso in più file
  - In questo caso la libreria era parte del progetto stesso, ed è stata compilata con esso
- Tuttavia, le librerie possono anche essere fornite da terzi
- Lo stesso meccanismo funziona quando la libreria e il codice che la usa appartengono a progetti diversi
  - Spesso non sono forniti i file .cpp, ma file pre-compilati

Un caso semplice

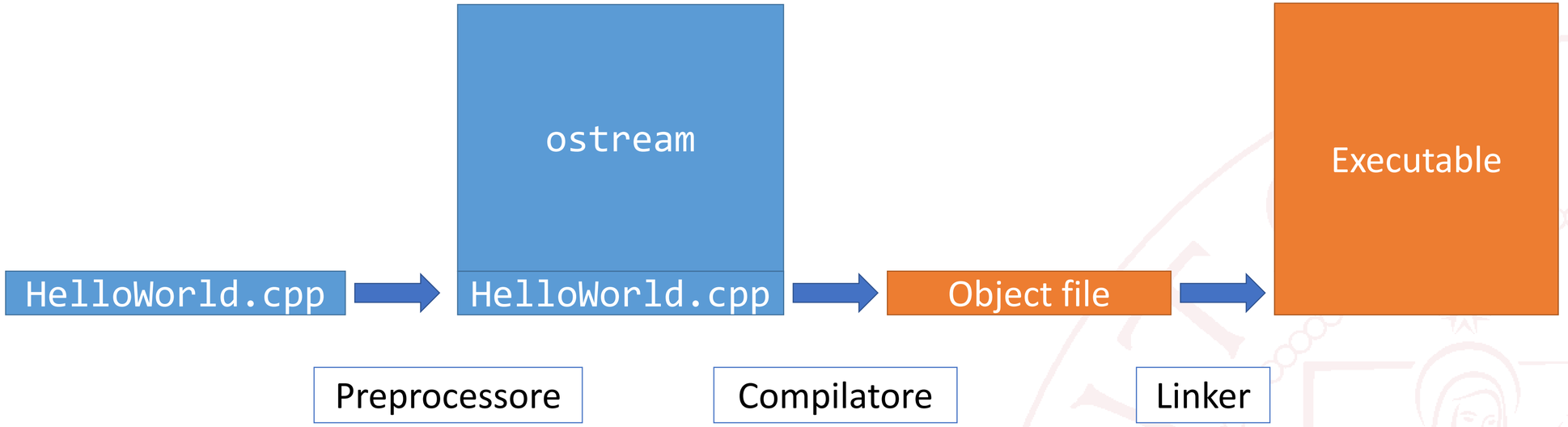
Un solo file sorgente (es., hello world)

```
#include <ostream>

int main(void)
{
  std::cout << "Hello world!\n";

  return 0;
}
```

Da sorgente a eseguibile

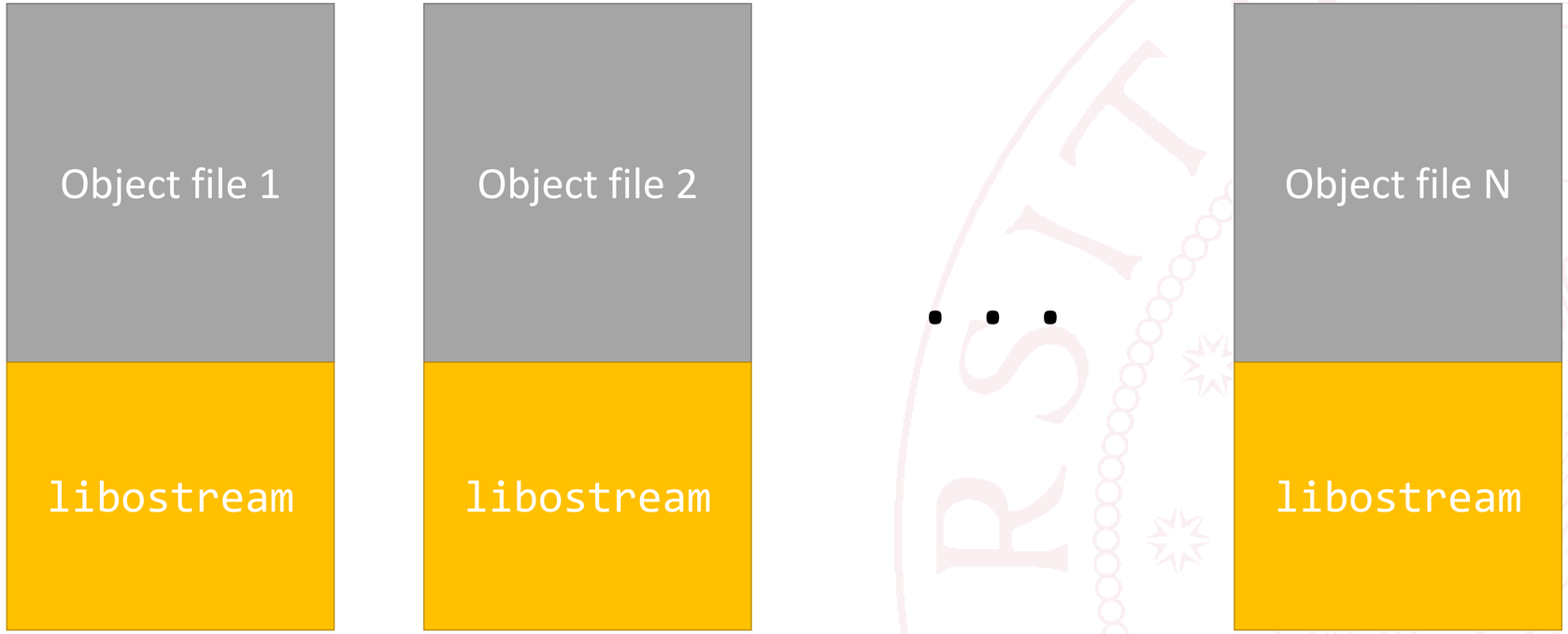


Codice sorgente

Codice macchina

Librerie statiche

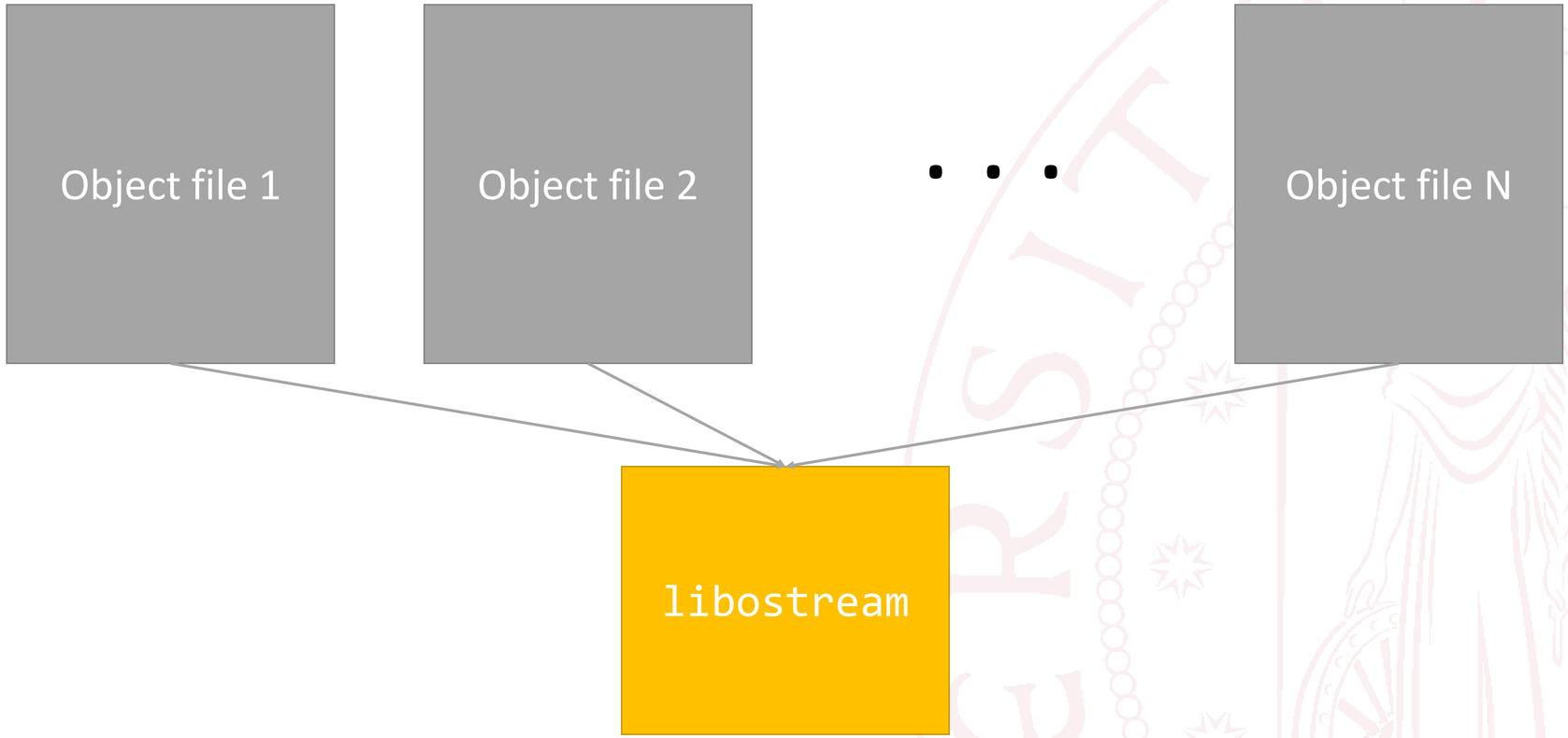
- Quello che abbiamo appena visto è un linking statico (libreria statica)
- Cosa succede se molti programmi linkano la stessa libreria?



Quale può essere un’altra soluzione?

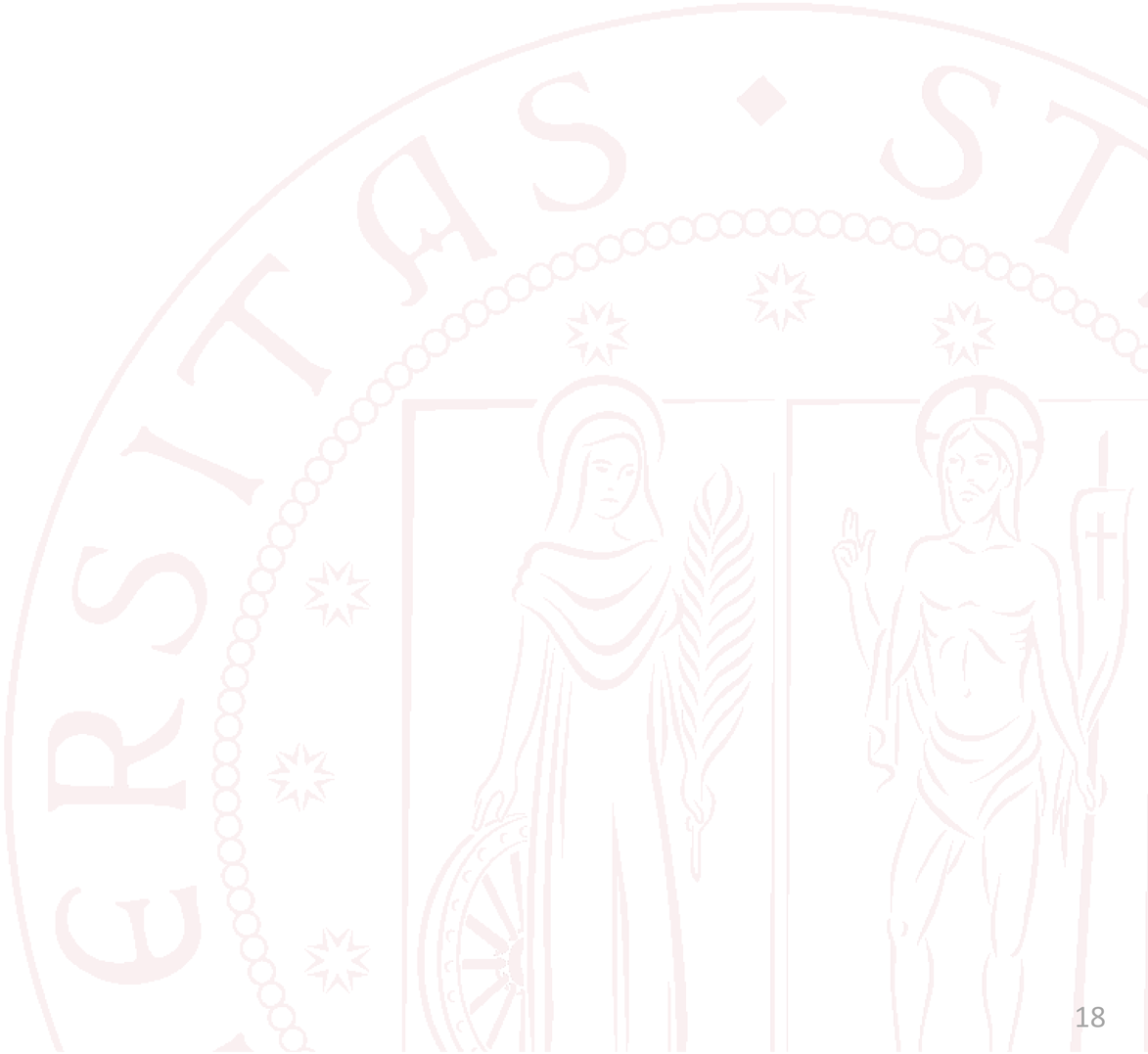
Librerie dinamiche

Una sola copia della libreria vale per tutti



Librerie statiche e dinamiche

- Quali sono i vantaggi e gli svantaggi?



Librerie statiche e dinamiche

- Quali sono i vantaggi e gli svantaggi?

- **Librerie dinamiche**
  - Riduzione dello spazio disco occupato (una sola copia funziona per tutti gli eseguibili)
  - Possono essere ricompilate senza toccare gli eseguibili
  - Chiamate .so (Shared Object) sotto Linux e .dll (Dynamic Linking Library) sotto Windows
- **Librerie statiche**
  - Generano eseguibili che non possono essere spezzati in seguito
  - Sono self-contained
    - Più adatte alla distribuzione di software monolitico

Come creare librerie statiche

```
ltonin@ltonin-laptop$ ls
main.cpp  my_func.cpp  my_func.hpp
ltonin@ltonin-laptop$ mkdir lib
ltonin@ltonin-laptop$ ls
lib main.cpp  my_func.cpp  my_func.hpp
ltonin@ltonin-laptop$ g++ -c my_func.cpp -o my_func.o
ltonin@ltonin-laptop$ ls
lib main.cpp  my_func.cpp  my_func.hpp my_func.o
ltonin@ltonin-laptop$ ar rvs lib/libmy_func_static.a my_func.o
ar: creating lib/libmy_func_static.a
a - my_func.o
ltonin@ltonin-laptop$ ls lib/
libmy_func_static.a
ltonin@ltonin-laptop$ g++ main.cpp -L./lib -lmy_func_static -o main_static
ltonin@ltonin-laptop$ ls
lib main.cpp  my_func.cpp  my_func.hpp my_func.o main_static
```

Compile but don’t’ link

.a → librerie statiche

-L : path dove cercare la libreria

-l : nome libreria (senza prefisso lib)

Come creare librerie dinamiche

```
ltonin@ltonin-laptop$ ls
main.cpp  my_func.cpp  my_func.hpp
ltonin@ltonin-laptop$ mkdir lib
ltonin@ltonin-laptop$ ls
lib main.cpp  my_func.cpp  my_func.hpp
ltonin@ltonin-laptop$ g++ -c my_func.cpp -o my_func.o
ltonin@ltonin-laptop$ ls
lib main.cpp  my_func.cpp  my_func.hpp my_func.o
ltonin@ltonin-laptop$ g++ -shared –o lib/libmy_func_dynamic.so my_func.o
ltonin@ltonin-laptop$ ls lib/
libmy_func_dynamic.so
ltonin@ltonin-laptop$ g++ main.cpp -L./lib -lmy_func_dynamic -o main_dynamic
ltonin@ltonin-laptop$ ls
lib main.cpp  my_func.cpp  my_func.hpp my_func.o main_dynamic
```

Compile but don’t’ link

.so → librerie dinamiche

-L : path dove cercare la libreria

-l : nome libreria (senza prefisso lib)

Recap

- File header lato utente
- File header lato progettista
- Processo di compilazione
  - Più in dettaglio
  - Con più file
- Librerie statiche e dinamiche