8.2 – Introduzione a CMake

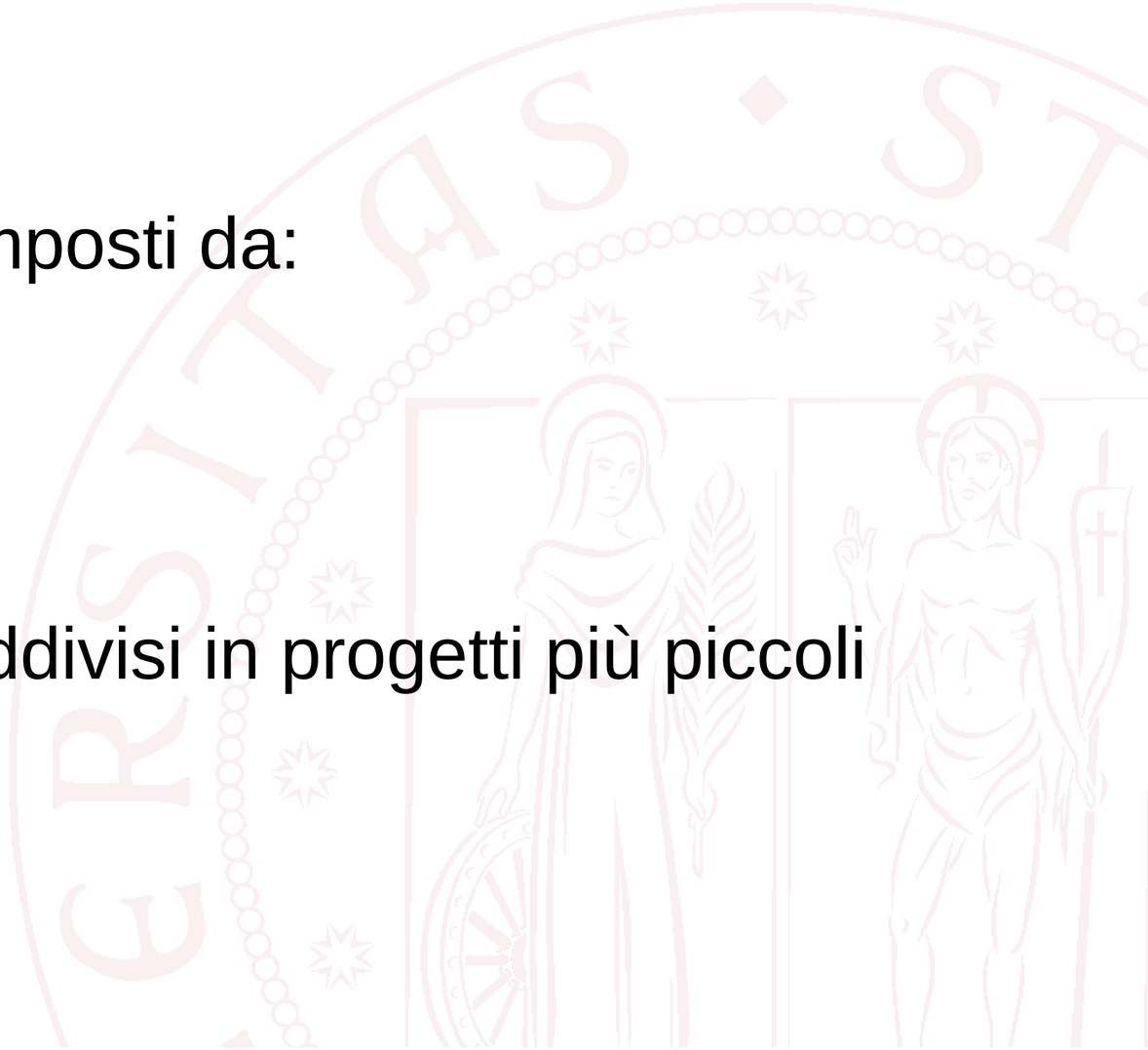
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- Descrivere un progetto software
- Sistemi di building
- Cos'è CMake
- Vantaggi
- Utilizzo lato utente
- Utilizzo lato sviluppatore

Descrivere un progetto

- Un progetto software è spesso composto da molti file sorgente (`.h`, `.hpp`, `.cpp`)
- Progetti complessi possono essere composti da:
  - Uno o più eseguibili
  - Una o più librerie
- Progetti così grandi **devono** essere suddivisi in progetti più piccoli



Descrivere un progetto

- I soli sorgenti non sono sufficienti!
- È necessario sapere come deve essere organizzata la compilazione
- Target di compilazione
  - Creare un eseguibile
  - Creare una libreria
- In entrambi i casi
  - Lista dei `.cpp` che devono essere compilati assieme
  - Lista delle librerie che devono essere linkate

Descrivere un progetto

- Come avete descritto i vostri progetti fino ad ora?

Descrivere un progetto

- Descrizione tramite comando di compilazione

  - Poco pratico: necessario riscrivere tutto a ogni compilazione
  - Ricompilazione completa

- Descrizione tramite progetto dell'IDE
  - Dipende dall'IDE in uso, poco generale

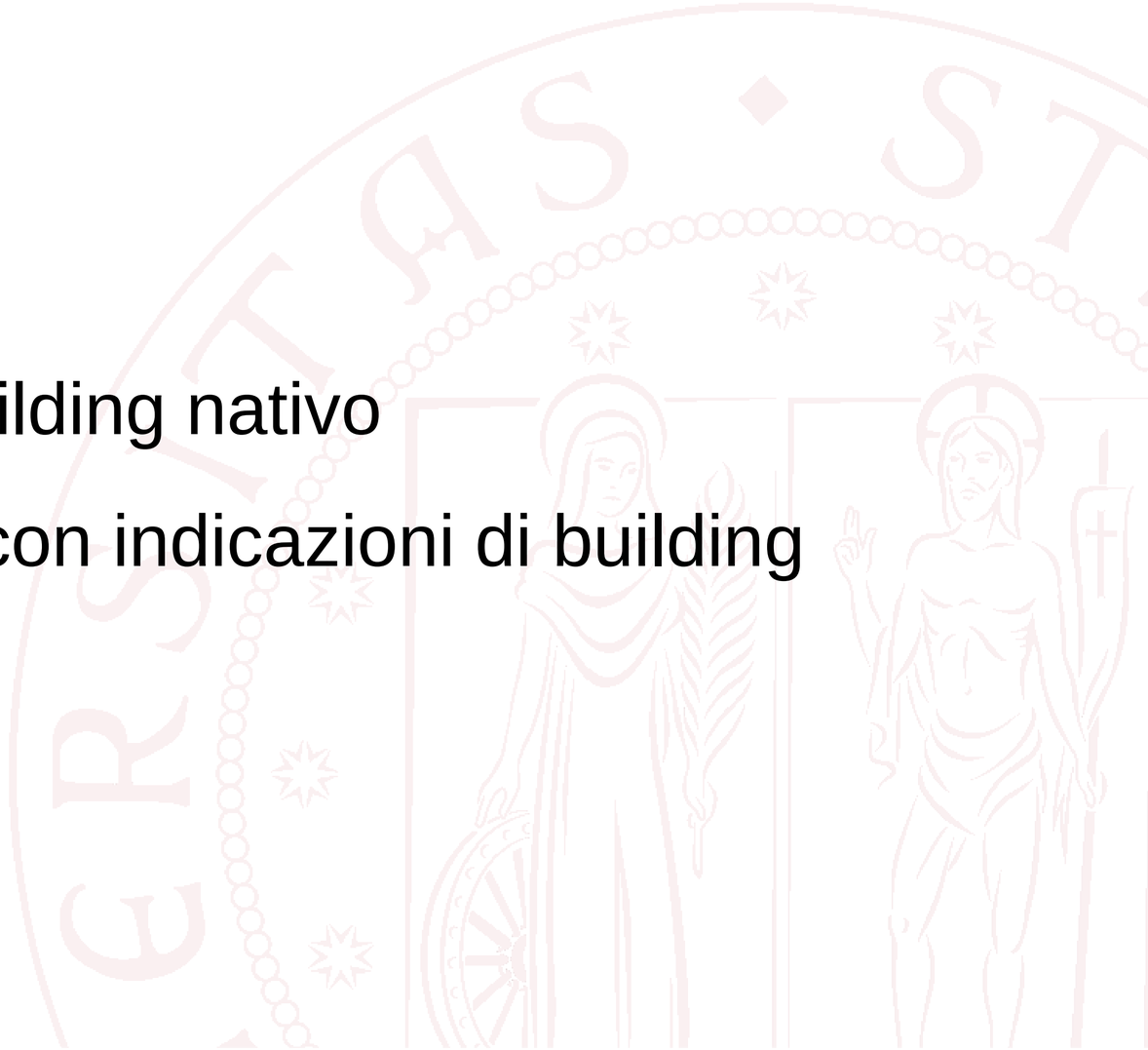
- Sarebbe molto utile avere a disposizione un **sistema di building** standard

Sistemi di building

- Sistema di building: gestore del processo di compilazione
- Esistono vari sistemi di building
  - Es: Make, Ninja, ...
  - Spesso dipendenti dalla piattaforma in cui lavorano

CMake: intro

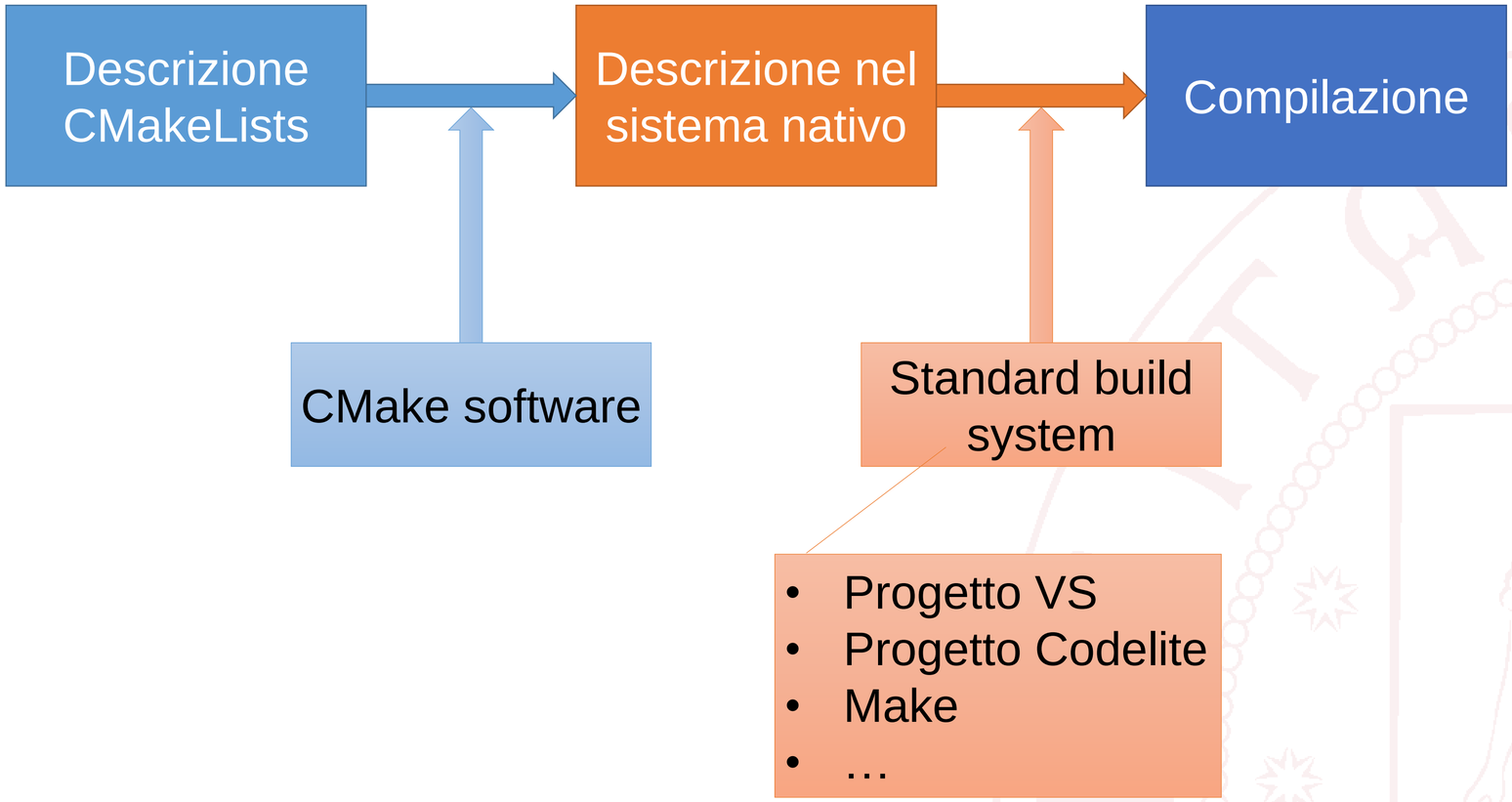
- Meta-sistema di building standardizzato
- Indipendente da:
  - Compilatore
  - Piattaforma
- Usato in abbinamento al sistema di building nativo
- Utile per distribuire il codice sorgente con indicazioni di building
- Supportato da molti IDE
  - Incluso Codelite



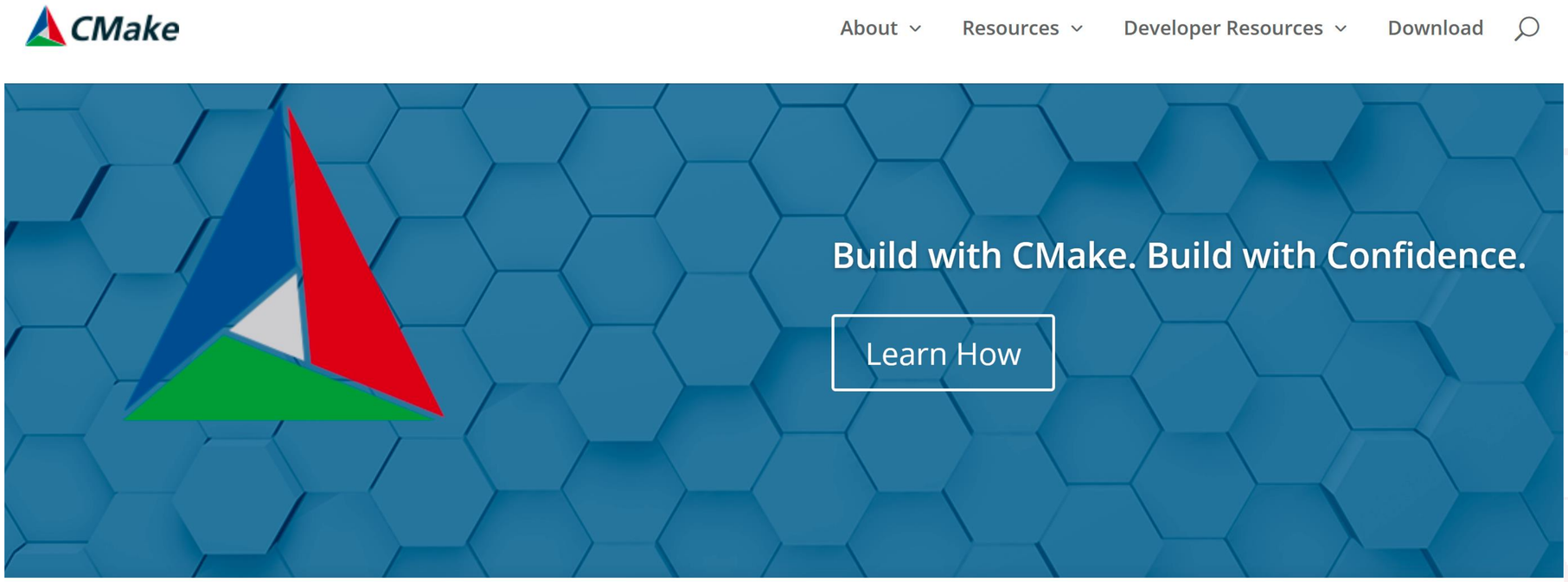
CMake: intro

- È un sistema alternativo per descrivere un progetto
- Definisce:
  - Target di compilazione ed elenco sorgenti
  - Eseguibile
  - Libreria
  - Librerie da linkare
- Gestito tramite file di testo: CMakeLists.txt

CMake: processo



CMake homepage

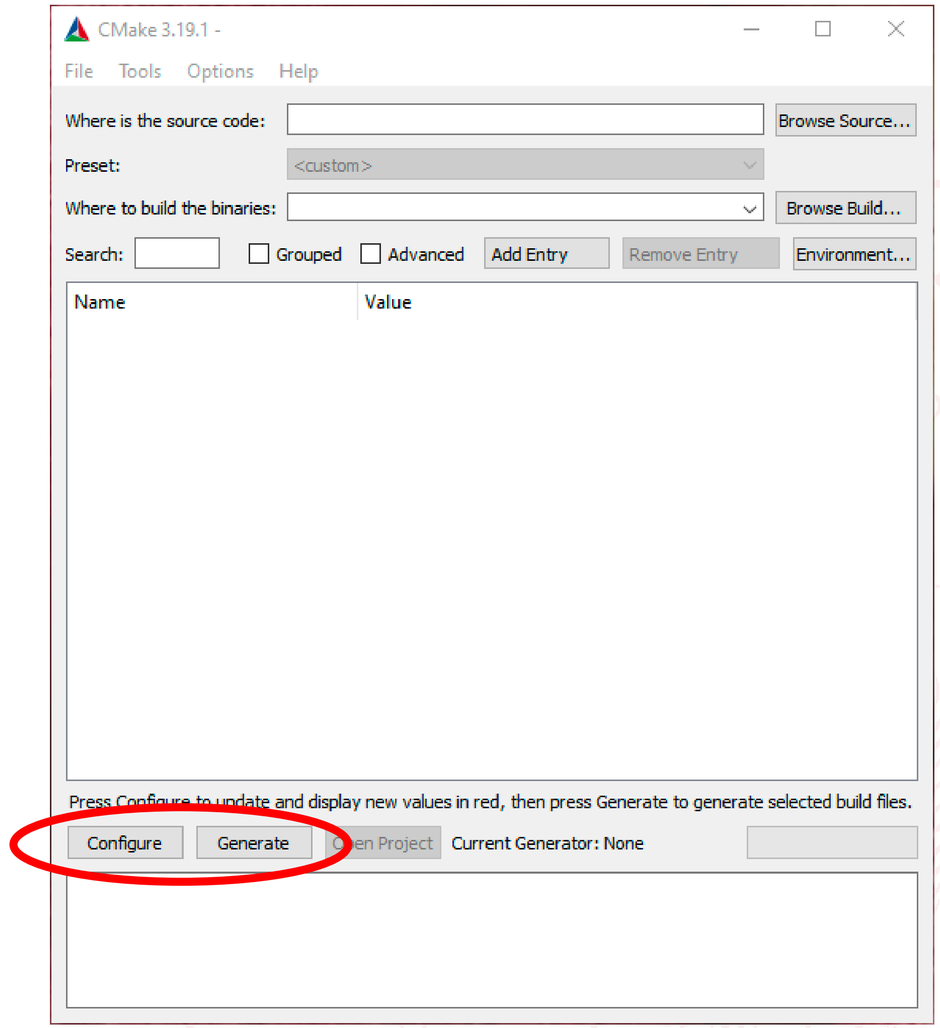


Processo lato utente

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Processo lato utente: il software

- Liberamente scaricabile
- Include un'interfaccia grafica
- Due attività fondamentali:
  - Configure
  - Generate



Lato utente

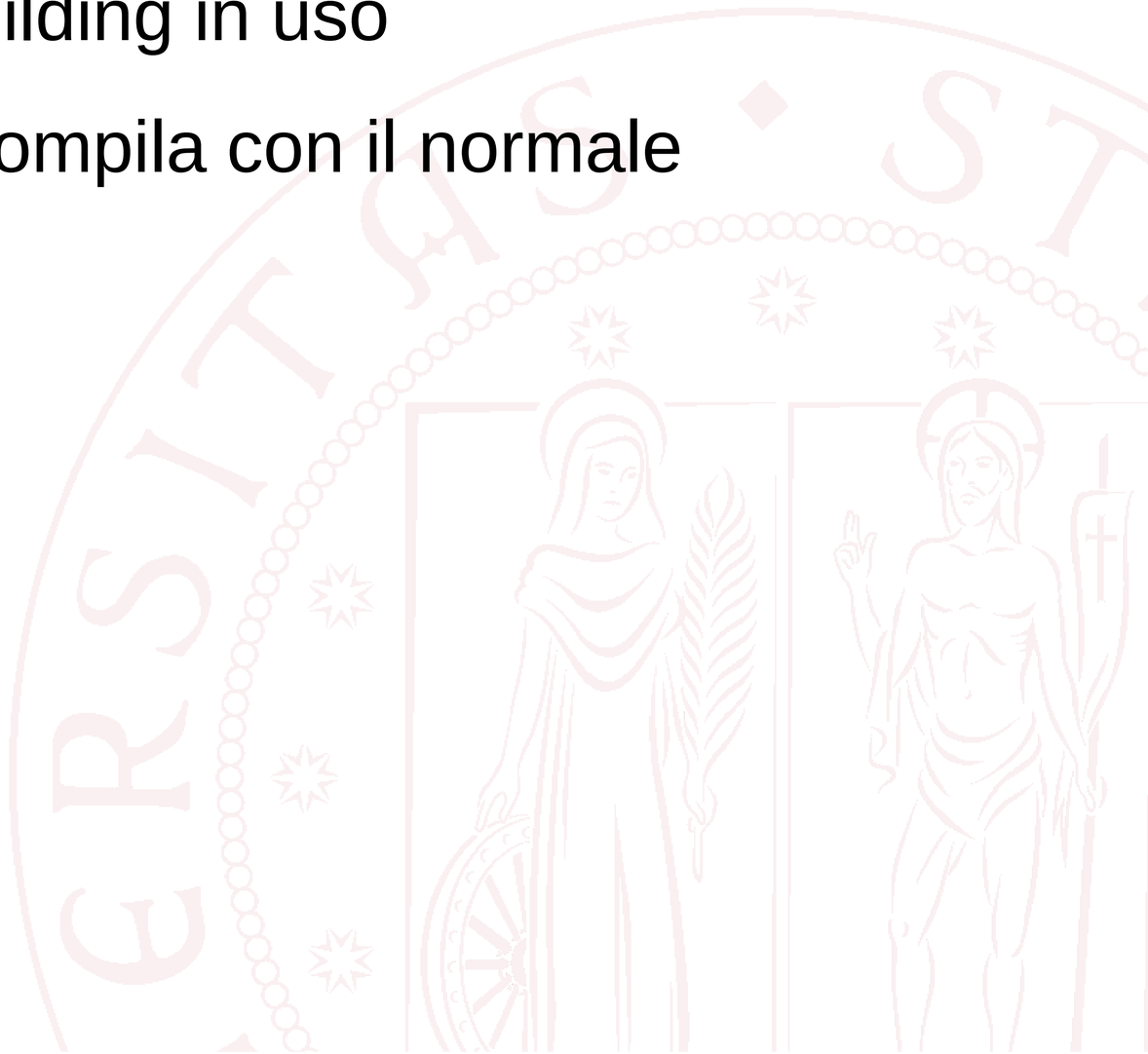
- Come utenti possiamo trovare un progetto gestito con CMake
- Scaricare un sorgente gestito con CMake permette di avere:
  - I file sorgenti del progetto
  - La descrizione del progetto
  - Un sistema che compila il progetto in autonomia
  - Un sistema per gestire la compilazione dei moduli
  - Parti del progetto possono essere compilate o meno a seconda delle impostazioni

Lato utente

- Tipica gestione di un progetto CMake:
  - Lanciare CMake (es., GUI)
  - Selezionare la directory contenente il sorgente
  - Selezionare la directory che conterrà i prodotti della compilazione
  - Standard: `[root sorgente]/build`
- Configure
- Eventuale selezione/deselezione dei moduli
  - Se ci sono modifiche: configure
- Generate

Lato utente

- Generate crea i file per il sistema di building in uso
- Dopo generate si chiude CMake e si compila con il normale flusso di lavoro



Processo lato sviluppatore

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Lato sviluppatore

- Si crea una descrizione del progetto nello standard CMake
  - Scrivere a mano i CMakeLists.txt
  - Usare un IDE per generarli automaticamente
    - Più complicati
    - In genere, non devono essere modificati
      - Possono essere sovrascritti in automatico

CMakeLists

- Fissare una versione minima di CMake
  - 2.8 è solitamente sufficiente

Sets the mimimum required version of CMake.

```
1 cmake_minimum_required(VERSION 3.2 FATAL_ERROR)
```

CMakeLists

- Dare un nome al progetto

Set the name and version; enable languages.

```
1 project(<name> VERSION <version> LANGUAGES CXX)
```

- CMake sets several variables based on project().
- Call to project() must be direct, not through a function/macro/include.
- CMake will add a call to project() if not found on the top level.

CMakeLists

- Selezionare librerie con le dipendenze
  - Non necessario per gli scopi di questo corso

Finds preinstalled dependencies

```
1 find_package(Qt5 REQUIRED COMPONENTS Widgets)
```

- Can set some variables and define imported targets.
- Supports components.

CMakeLists

- Target per la creazione di un eseguibile
  - Equivalente a g++ -o tool –c main.cpp another_file.cpp

Add an executable target.

```
1  add_executable(tool
2    main.cpp
3    another_file.cpp
4    )
```

CMakeLists

- Target per la creazione di una libreria

Add a library target.

```
1  add_library(foo STATIC
2    foo1.cpp
3    foo2.cpp
4    )
```

- Libraries can be `STATIC`, `SHARED`, `MODULE`, or `INTERFACE`.
- Default can be controlled with `BUILD_SHARED_LIBS`.

Esempio completo

```
ltonin@ltonin-laptop:~/Rational/
├── bin
├── build
├── CMakeLists.txt
├── include
│   └── rational.h
├── README.txt
└── src
    ├── main.cpp
    └── rational.cpp
```

Esempio completo

```
cmake_minimum_required(VERSION 2.84 FATAL_ERROR)
set(CMAKE_CXX_STANDARD 11)
project(rational)

include_directories(include)

add_library(${PROJECT_NAME} SHARED
            src/rational.cpp)

add_executable(main bin/main.cpp)

target_link_libraries(main ${PROJECT_NAME})
```

Esempio completo

```
ltonin@ltonin-laptop:~/Rational/$ mkdir build
ltonin@ltonin-laptop:~/Rational/$ cd build
ltonin@ltonin-laptop:~/Rational/$ cmake ..
ltonin@ltonin-laptop:~/Rational/$ make
```

Recap

- Perché dobbiamo descrivere un progetto?
- Soluzione standard: CMake
  - Vantaggi
- Gestione di un semplice progetto tramite CMake
  - Lato utente
  - Lato sviluppatore