Uniwersytet Wrocławski

Biblioteki w C++

Zbigniew Koza
Wydział Fizyki i Astronomii

C++/C to metajęzyki wyspecjalizowane w obsłudze bibliotek

- W czystym C/C++ naprawdę niewiele można zrobić (w rozsądnym czasie)
- Popularność tych języków bierze się z łatwości tworzenia w nich i używania bibliotek
- Siła C/C++ leży w dostępności wysokiej jakości bibliotek rozwijanych od blisko 50 lat

Dwa wyzwania

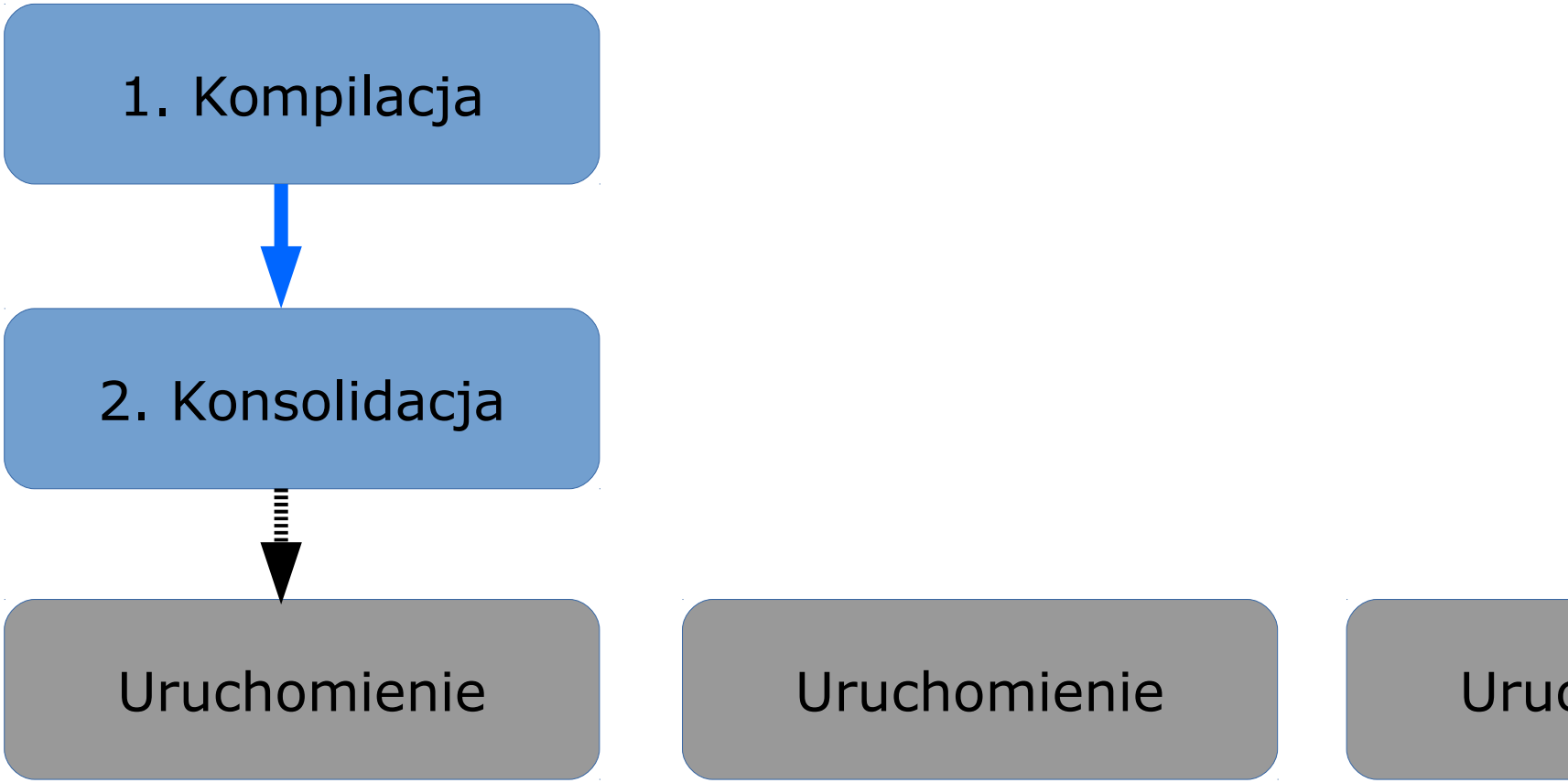
- Jak używać biblioteki?

Poziom operacyjny/ zawodowy

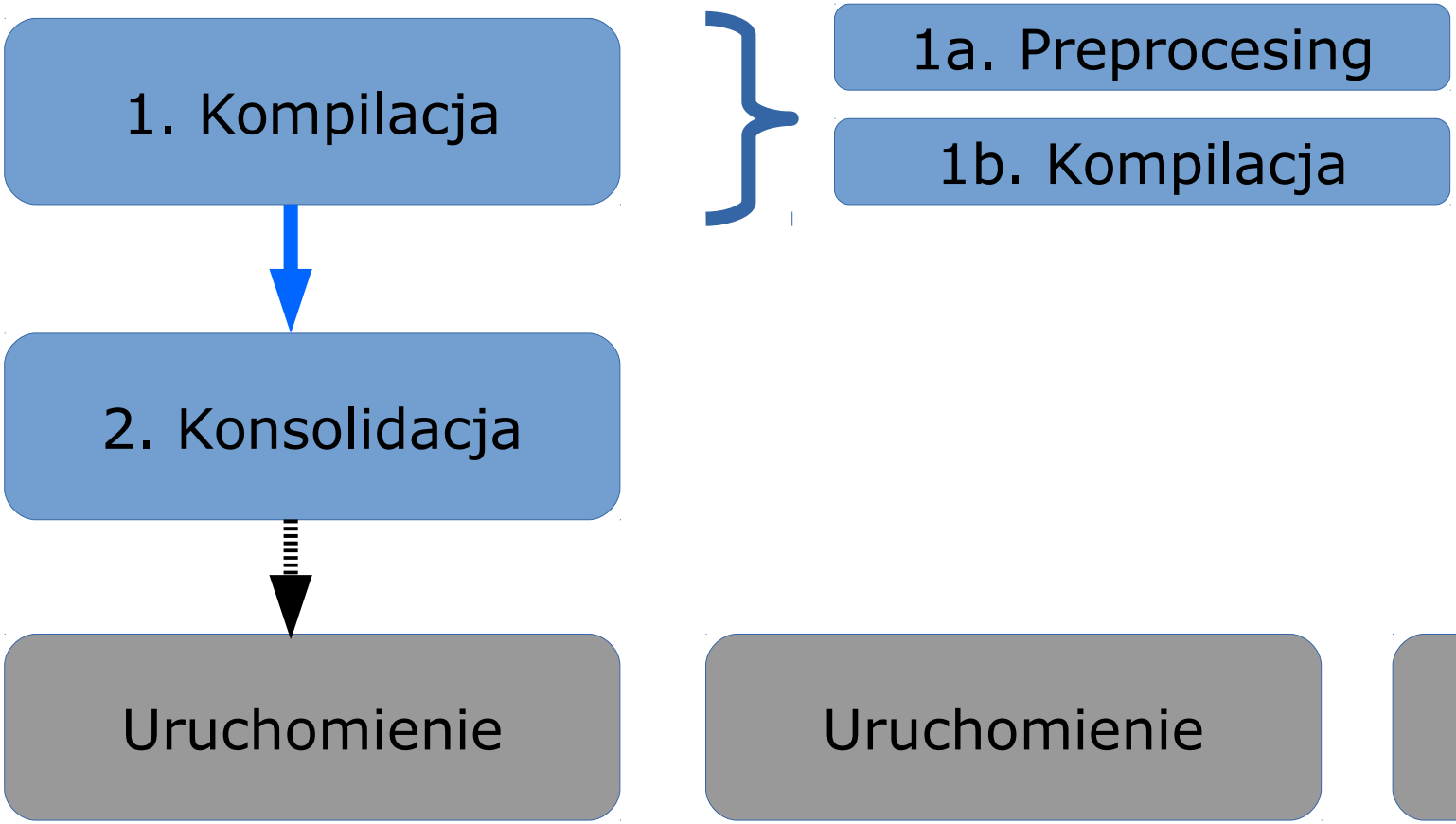
- Jak tworzyć biblioteki?

Poziom ekspercki

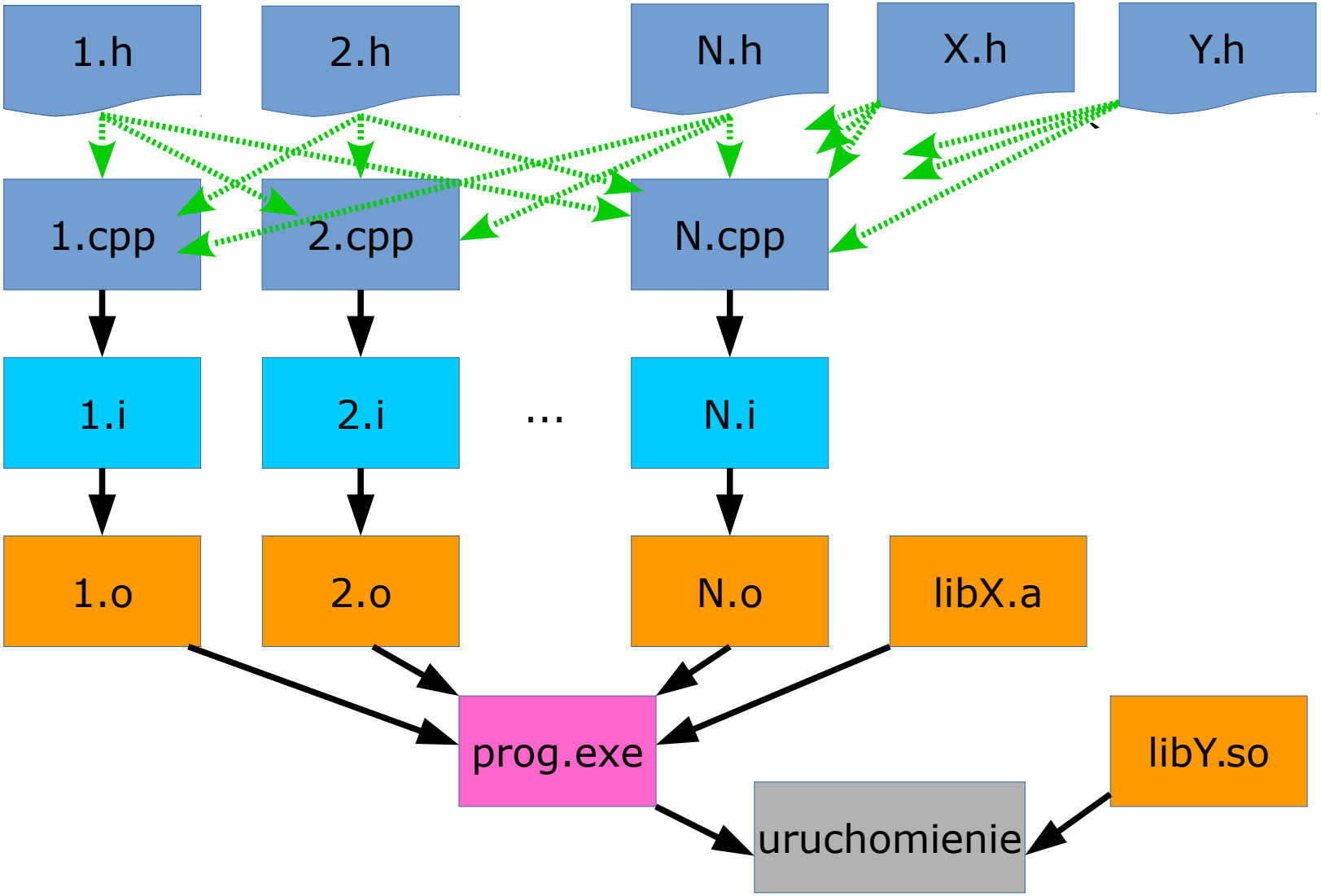
Główne etapy kompilacji programu



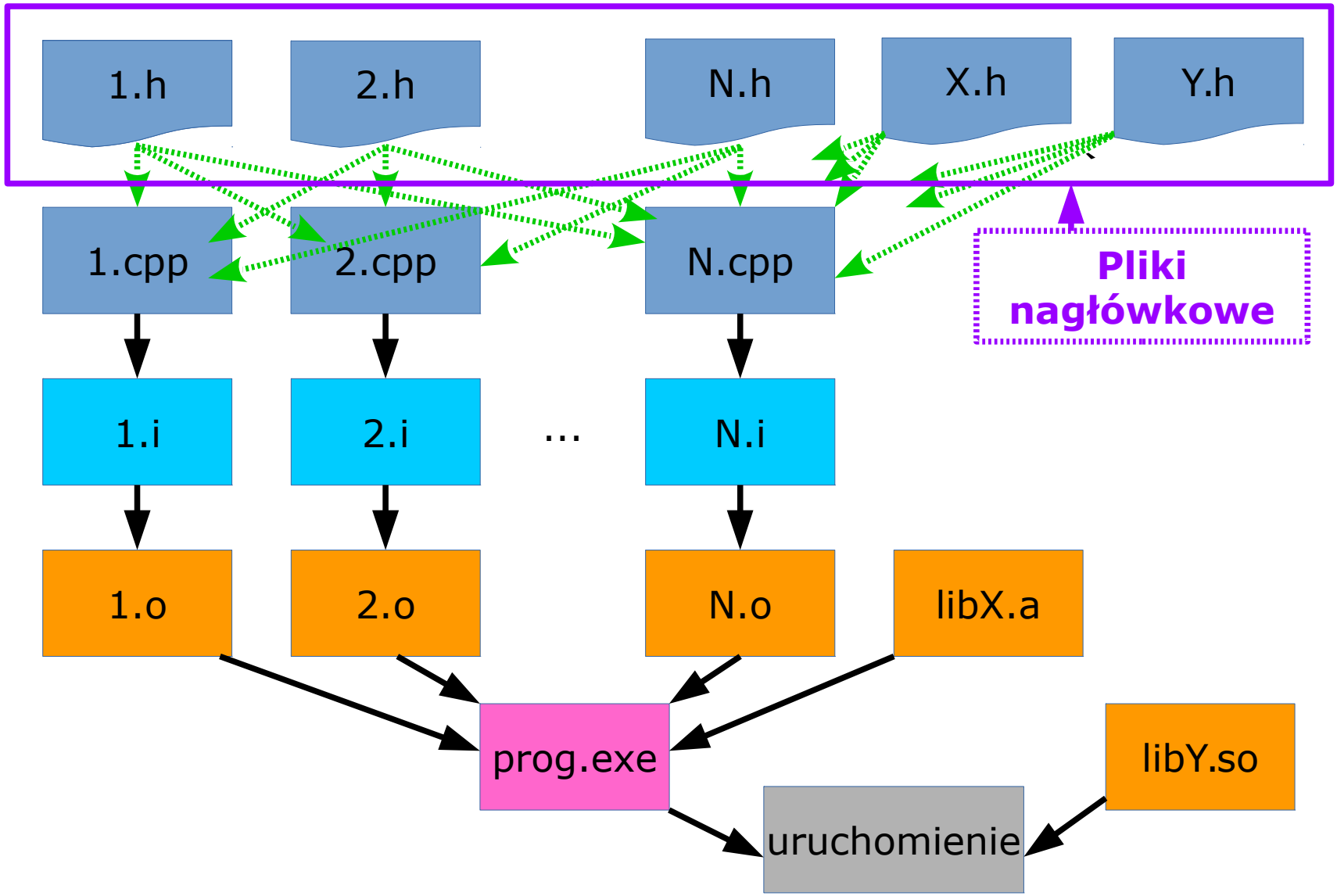
Główne etapy kompilacji programu



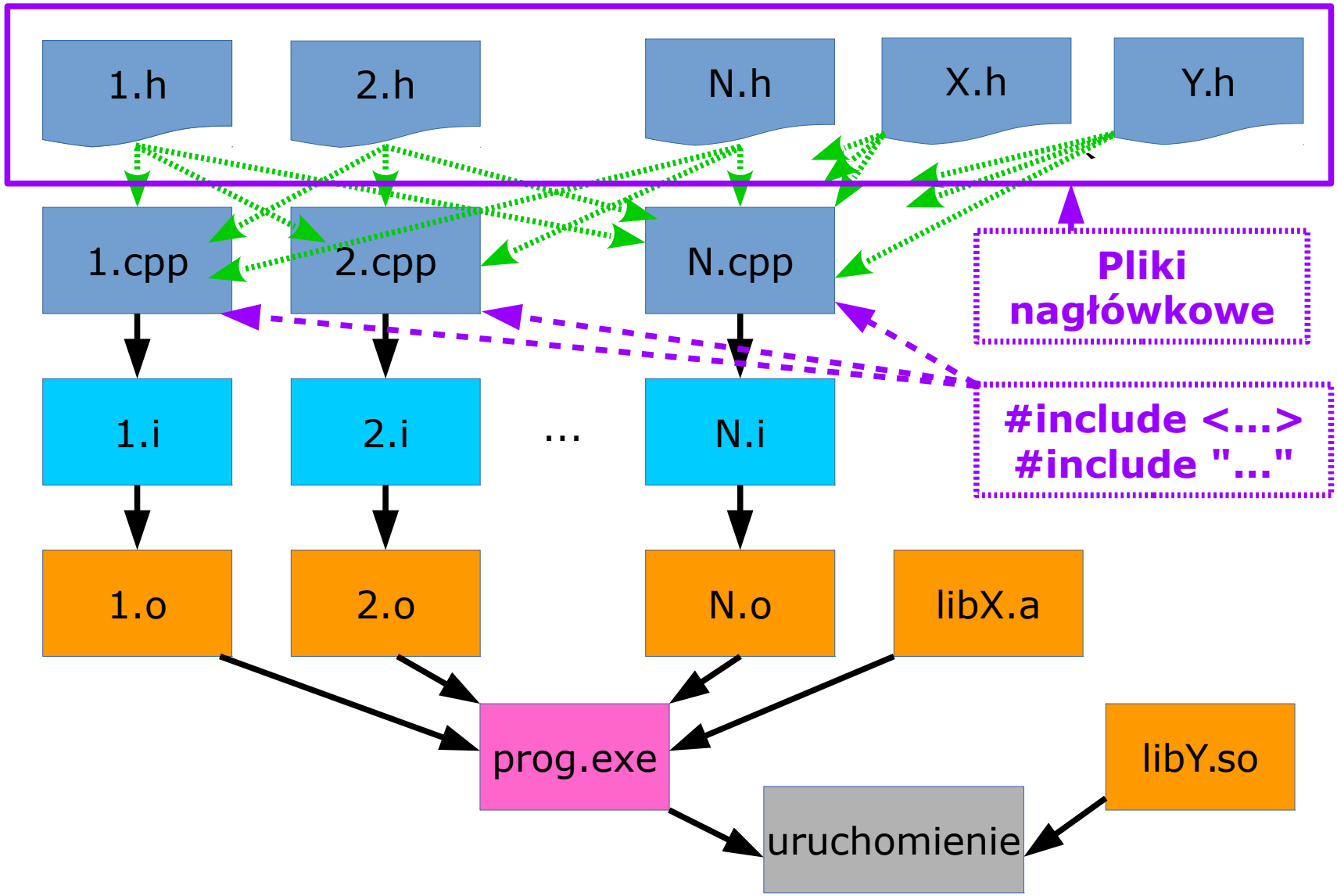
Jak przebiega proces kompilacji?



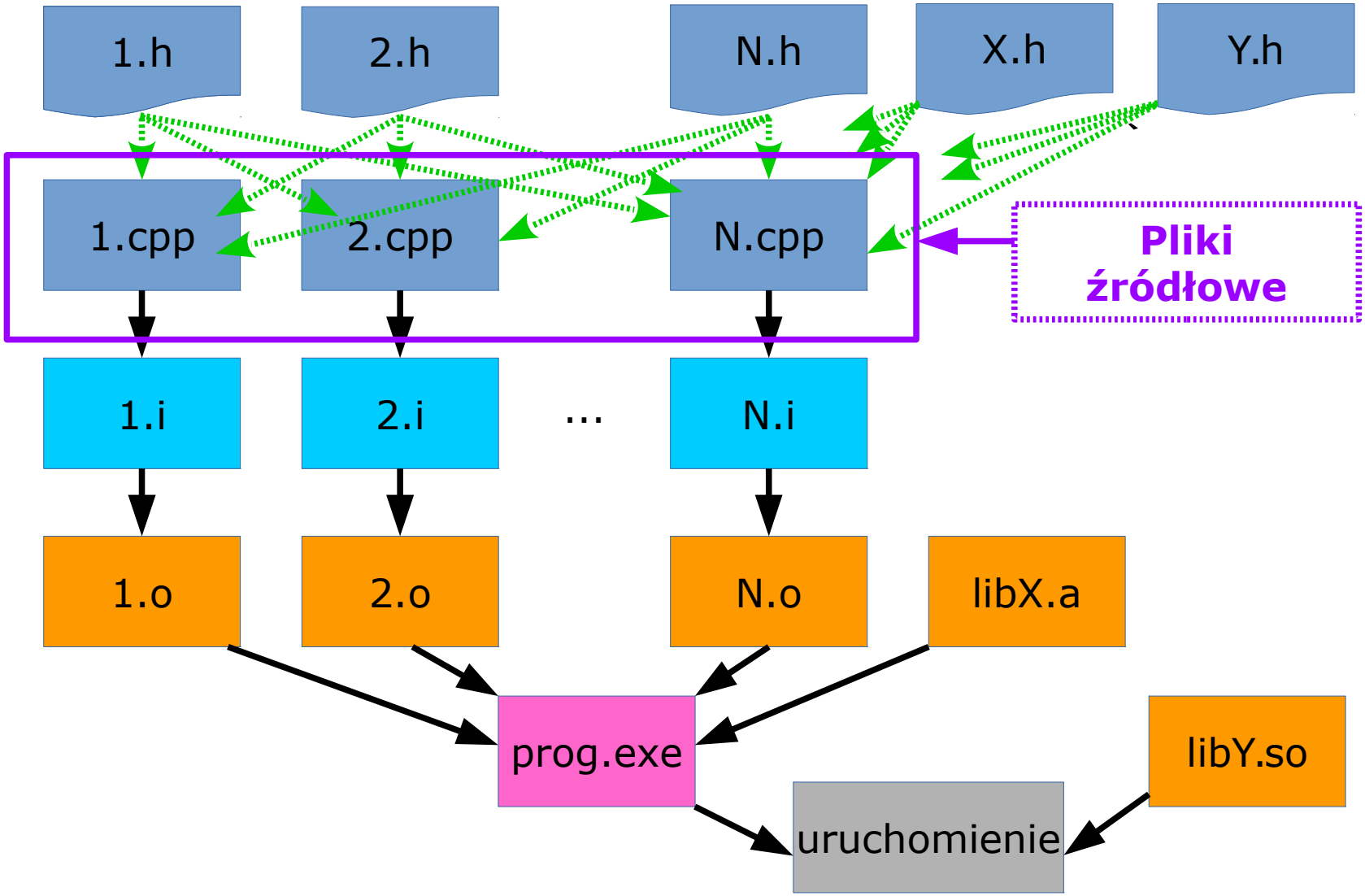
Pliki nagłówkowe



Pliki nagłówkowe są włączane...



Do plików źródłowych



Co i po co jest w plikach nagłówkowych?

- C/C++ o programy z (dość) silną typizacją danych
- Zanim cokolwiek użyjesz, musisz zadeklarować **typ** tego czegoś
- Obiekt o raz zdefiniowanym typie nie może już go zmienić → to pomaga uzyskać efektywny kod i pomaga eliminować błędy

Co i po co jest w plikach nagłówkowych?

- Dlatego zanim użyjemy cokolwiek z biblioteki zewnętrznej, musimy to coś zadeklarować.
- Deklaracje umieszcza się właśnie w plikach nagłówkowych
- Pliki nagłówkowe włącza się do programu makrem preprocesora

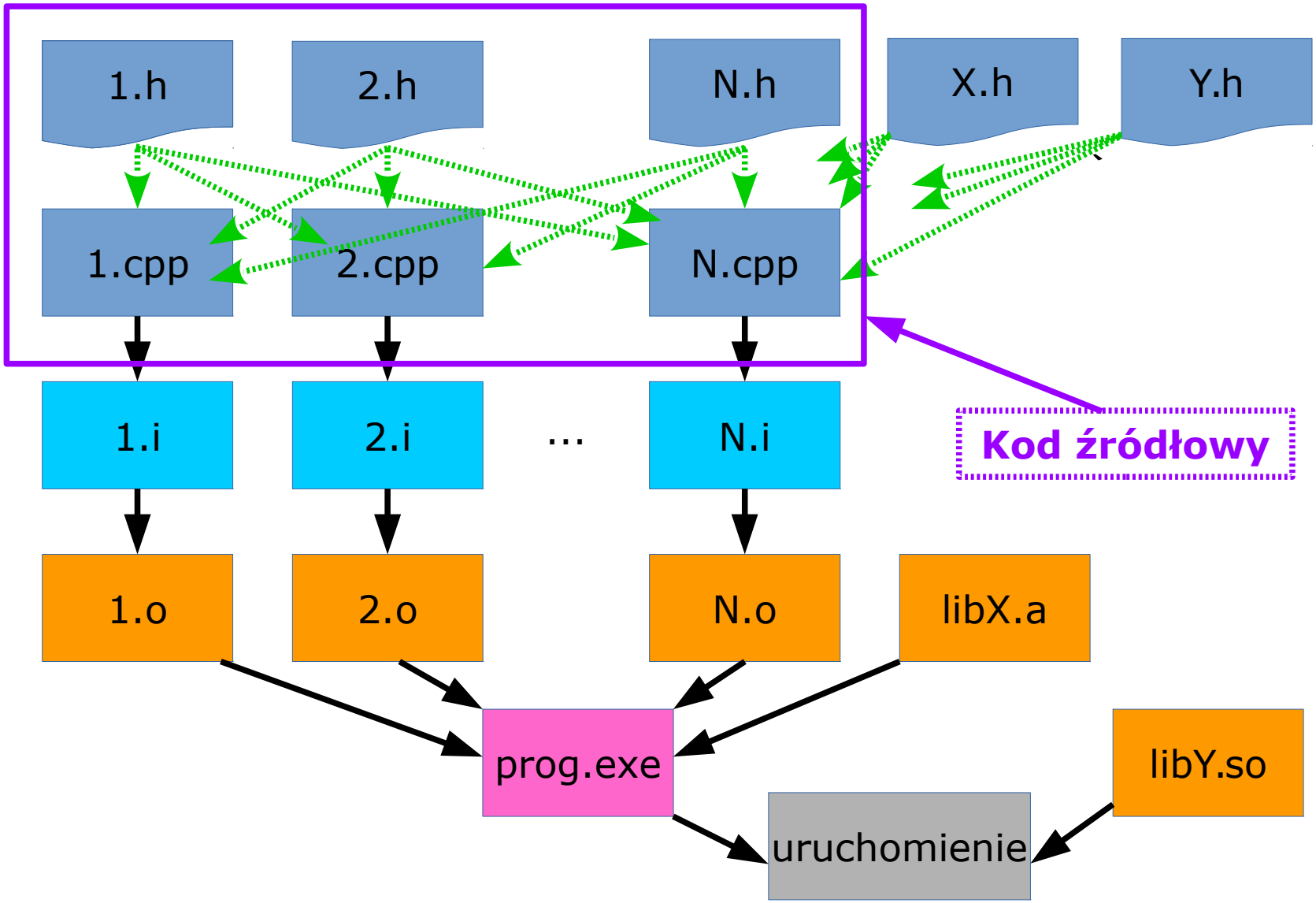
  #include <…> lub

  #include "plik"

#include

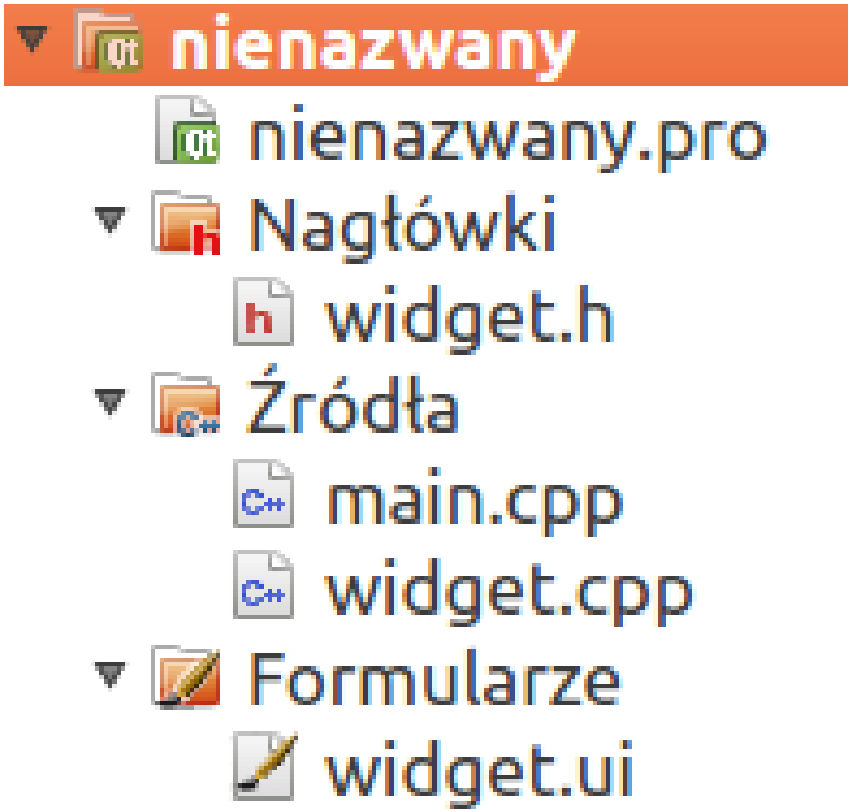
- #include <cmath>
  oznacza: w tym pliku korzystam
  z (zewnętrznej?) biblioteki cmath
- #include "version.h"
  oznacza: w tym pliku korzystam
  z (mojej?) biblioteki/modułu version

Kod źródłowy

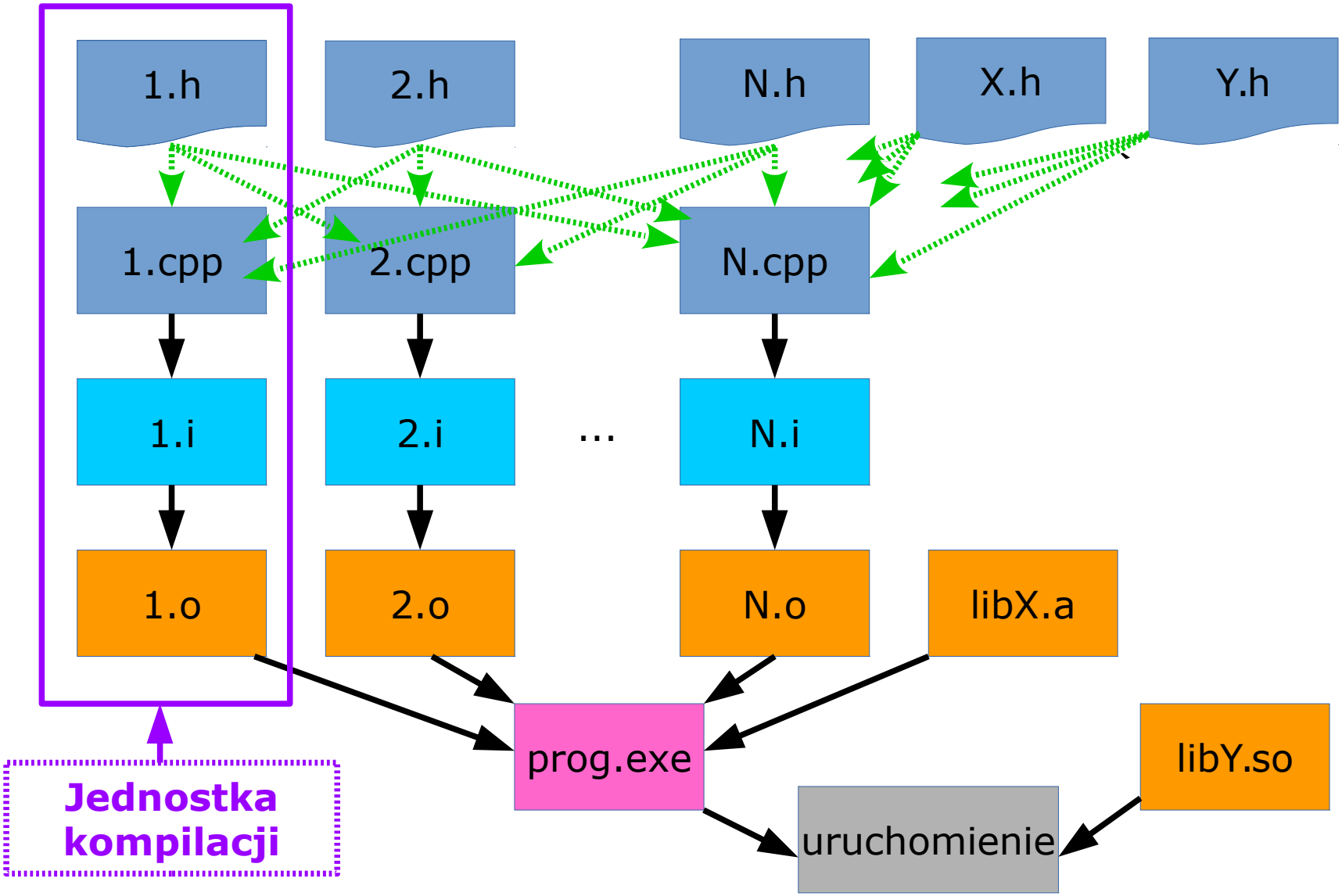


Typowa struktura kodu

- Zwykle pliki źródłowe, nagłówkowych, grafikę itp. umieszcza się w osobnych katalogach



Jednostki kompilacji



Jednostki kompilacji są rozłączne

- Kompilację plików źródłowych można wykonać rozłącznie, niezależnie od siebie
- Jednostka kompilacji = jeden plik *.cpp
- Skoro jednak w plikach źródłowych chcemy korzystać z kodu, zdefiniowanego gdzie indziej, musimy do nich włączyć deklaracje tego zewnętrznego kodu
  → #include <...>

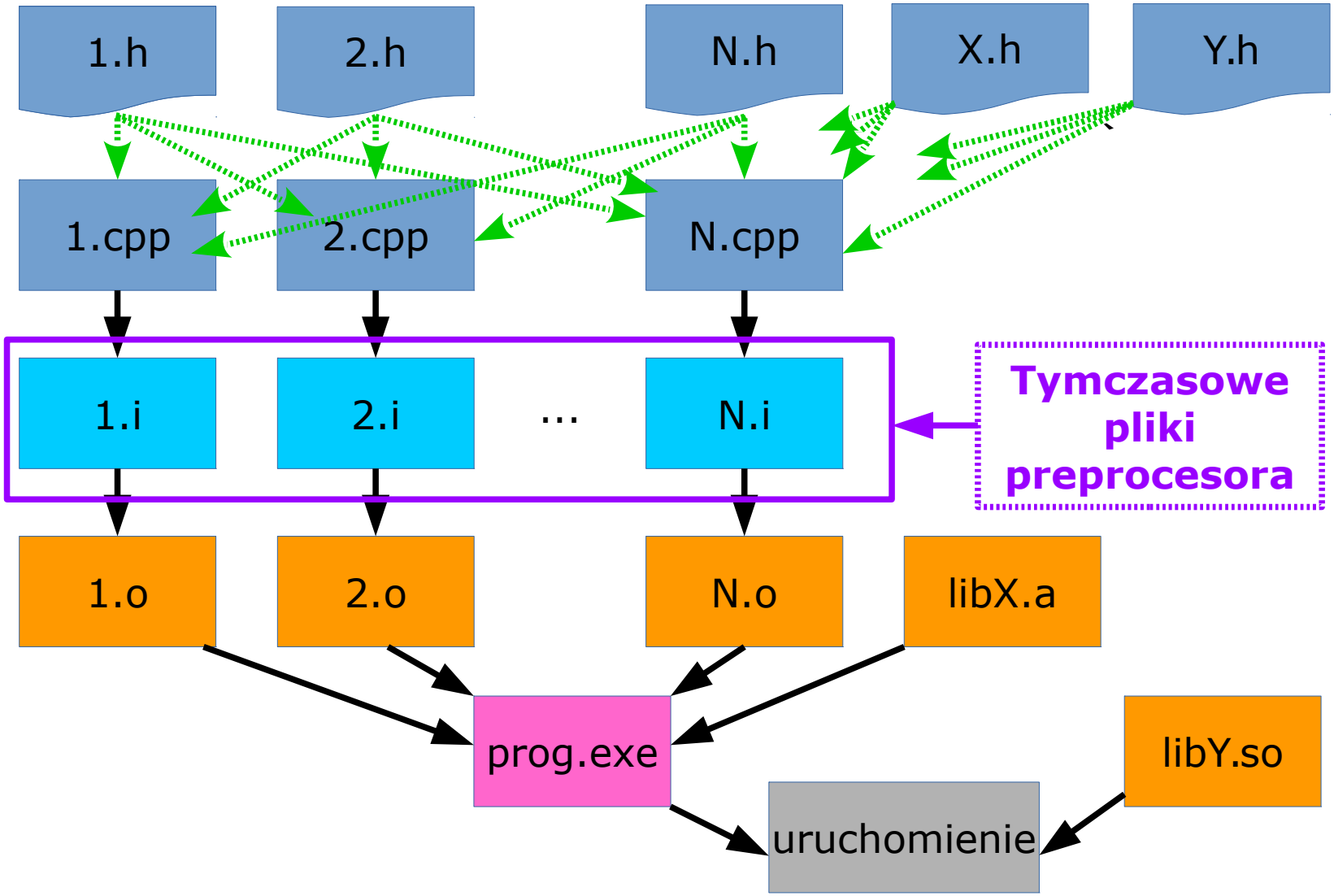
Jednostki kompilacji są niezależne

- Program podzielony na wiele plików można kompilować wieloma różnymi wersjami kompilatora, nawet różnymi kompilatorami, w różnym czasie i rożnych miejscach (jak inaczej tworzyć biblioteki?)
- Kompilator, kompilując plik 1.cpp nie zajrzy do treści żadnego pliku, który nie jest włączany do 1.cpp makrem #include

Dwóch ich zawsze jest...

- Praktycznie każdemu plikowi *.cpp towarzyszy plik *.h (czasem kilka…)
- Częsty wyjątek: main.cpp
- Część plików nagłówkowych może nie mieć towarzyszącego im pliku źródłowego (np. prosty version.h, ale też całkiem skomplikowane pliki)

Pliki *.h = interfejsy



-E

```cpp
#include <iostream>

int main()
{
  std::cout << "Witaj, świecie!\n";
}
```

1.cpp

```
g++ -E 1.cpp -o 1.i
wc -l 1.i
```

preprocessing

zliczenie liczby wierszy

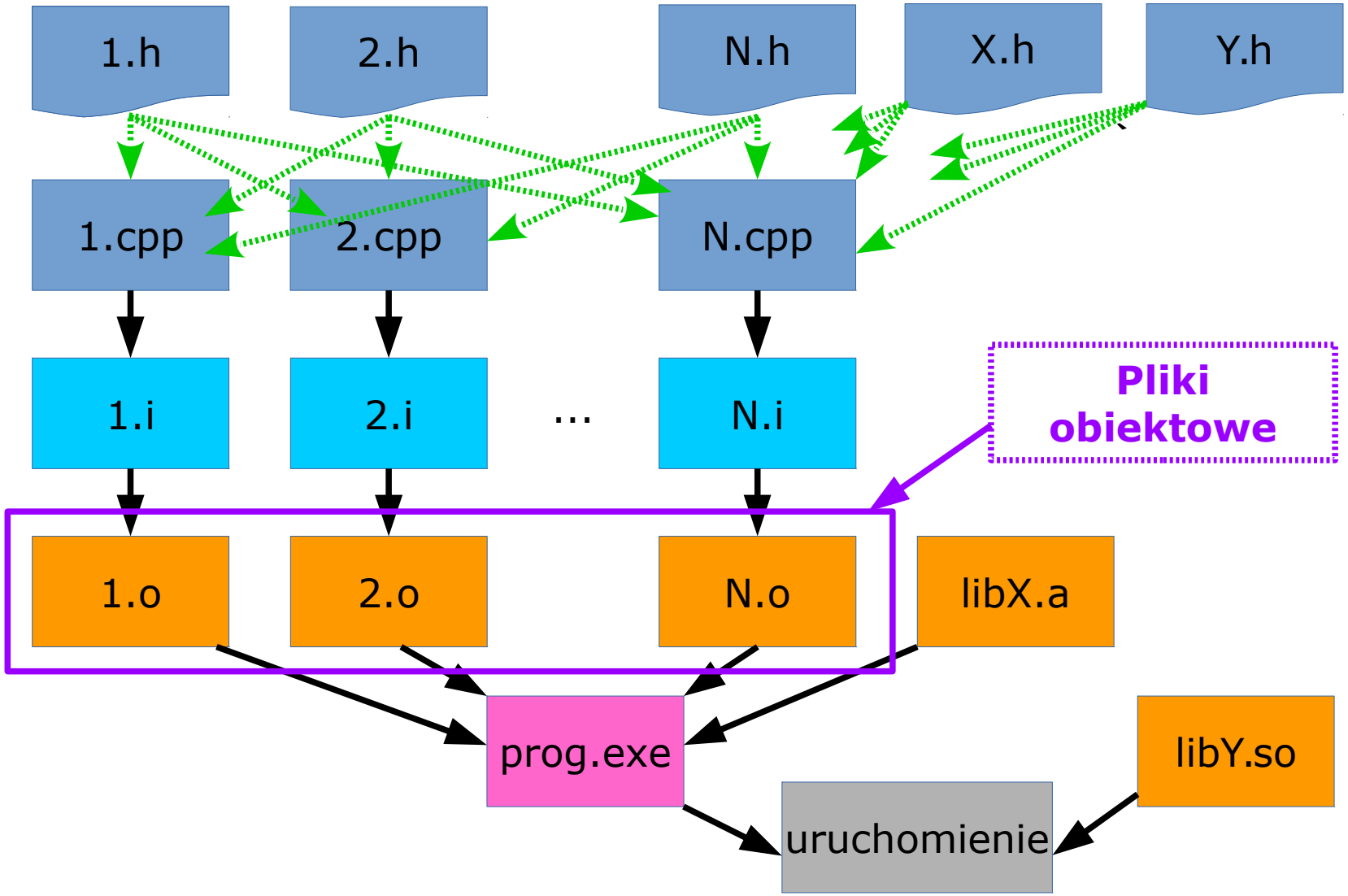
```
26779
```

wynik

Opcja -E

- Opcja -E (kompilator gcc) zatrzymuje kompilację na etapie przetworzenia pliku źródłowego przez kompilator
- Użyj jej, jeśli chcesz sprawdzić, jak interpretowane są w Twoim programie makra preprocesora: co naprawdę na wejściu dostaje kompilator?

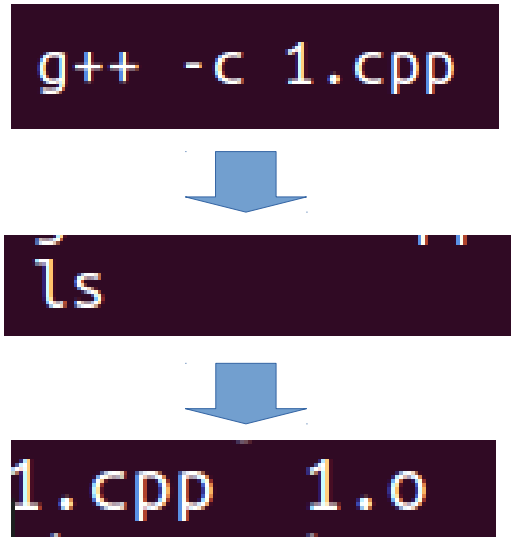
Pliki *.o = skompilowane moduły



*.o, *.obj

- Efektem kompilacji pliku źródłowego w jednostce kompilacji jest plik obiektowy
- Zwykle ma rozszerzenie *.o (linux) lub *.obj (Windows)
- Domyślnie nie jest na trwałe zapisywany na dysku
- Jest to oczywiście plik binarny!!!

Opcja -c

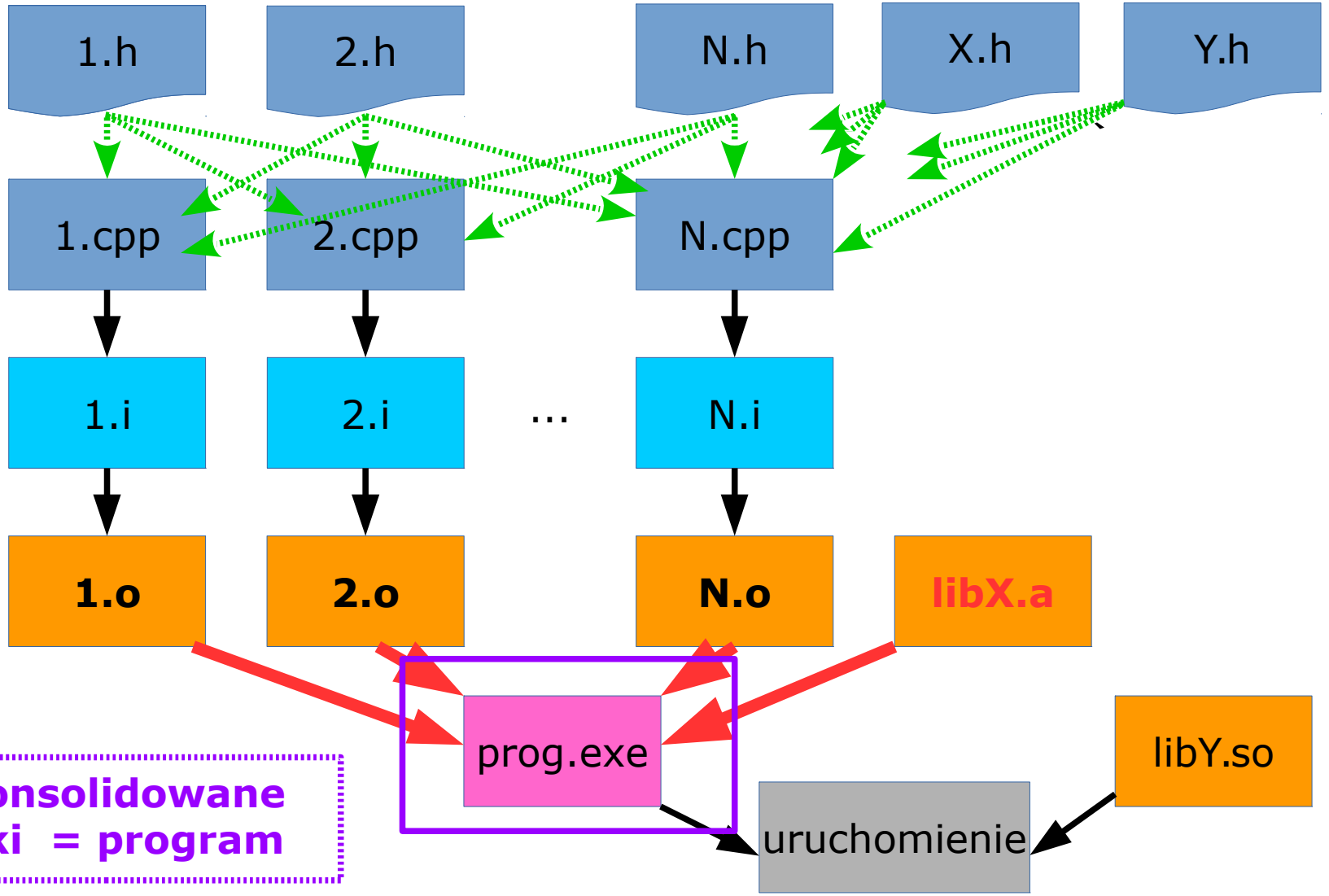


- Opcja -c powoduje przerwanie kompilacji z chwilą wygenerowania pliku obiektowego

Kompilacja zakończona...

- Z formalnego punktu widzenia kompilacja kończy się po skompilowaniu wszystkich jednostek translacji, czyli kompilacji wszystkich plików źródłowych (*.cpp) do obiektowych (*.o)
- Kolejnym etapem jest **konsolidacja programu**

Pliki *.o = skompilowane moduły



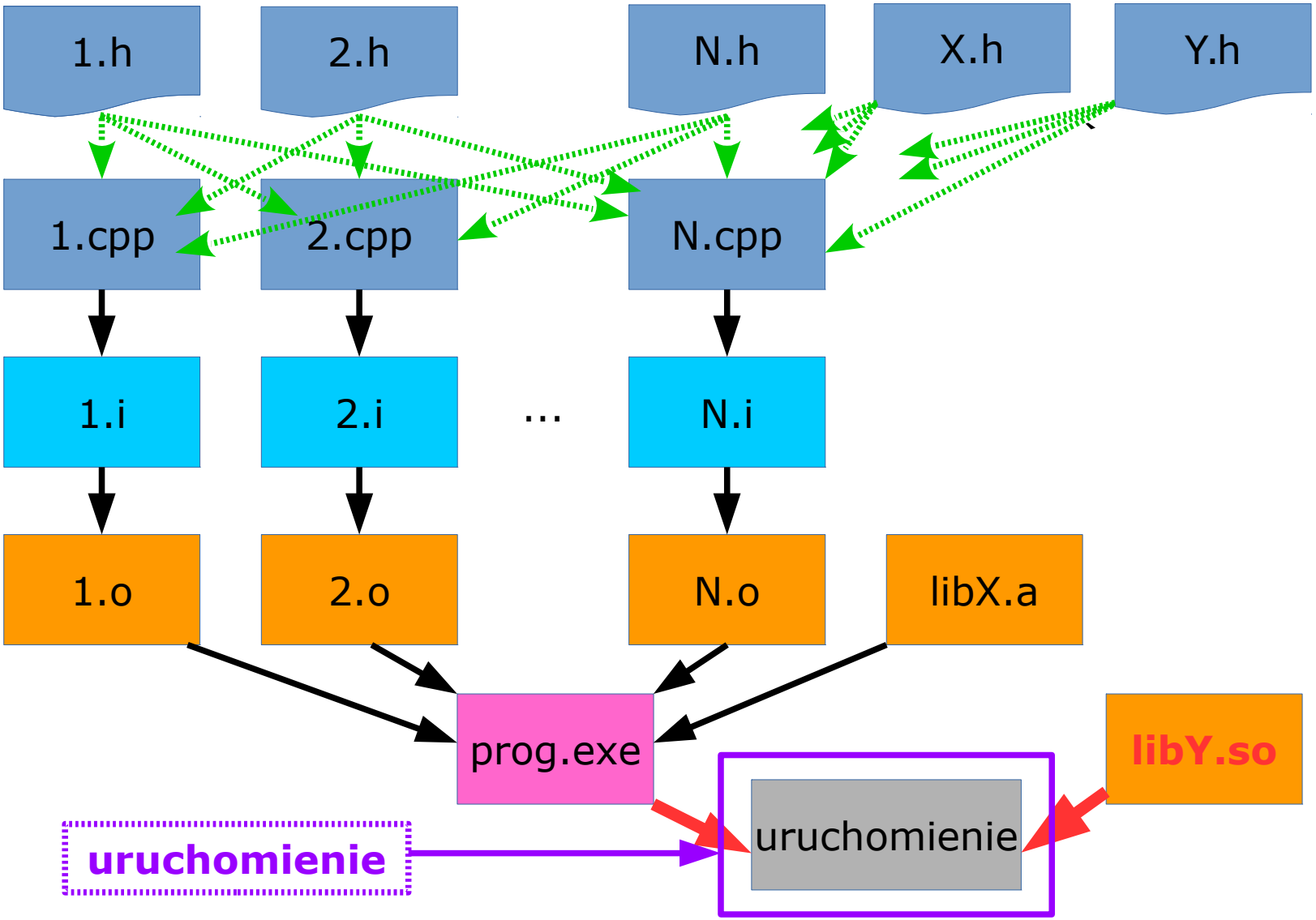
Konsolidacja

- Konsolidację programu przeprowadza konsolidator (ang. *linker*)
- Konsolidacji podlegają pliki obiektowe (*.o) i pliki bibliotek statycznych (*.a)
- Jej efektem jest plik wykonywalny
- Konsolidatora nie powinno obchodzić, w jakich programach napisano kody źródłowe

Konsolidacja jest szybka

- Kompilacja dużych programów może trwać wiele godzin, a ich konsolidacja zwykle nie dłużej niż ok. minuty
- Pozwala to efektywnie rozwijać nawet bardzo duży kod
  - (kompiluje się tylko niedawno zmienione jednostki translacji)

Uruchomienie programu



Program uruchomieniowy

- Nigdy nie wywołałem go bezpośrednio
- Jego celem jest załadowanie obrazu programu do pamięci, połączenie go z bibliotekami współdzielonymi i uruchomienie

Czego potrzebuje Twój program?

- Jeśli twój program podzielony jest na pliki, to potrzebujesz dostarczyć (niemal) każdemu plikowi źródłowemu jego interfejsu w pliku nagłówkowym *.h

Czego potrzebujesz od bibliotek (binarnych)

1) Gdzie są interfejsy (*.h)
2) Gdzie są pliki ze skompilowaną biblioteką statyczną (*.a)
3) lub dynamiczną (*.so)

   Ad 1) -I

   Ad 2) -L

   Ad 3) LD_LIBRARY_PATH

Gdzie są interfejsy?

std::function

Defined in header <functional>

```
template< class >
class function; /* undefined */
```
(since C++11)

```
template< class R, class... Args >
class function<R(Args...)>;
```
(since C++11)

QApplication Class

The QApplication class manages the GUI a

Header: #include <QApplication>

qmake: QT += widgets

Kompilacja i linkowanie

default location of the `gsl` directory is `/usr/local/include/gsl`. A typical compilation command for a source file `example.c` with the GNU C compiler `gcc` is:

```
$ gcc -Wall -I/usr/local/include -c example.c
```

```
$ gcc -L/usr/local/lib example.o -lgsl -lgslcblas -lm
```

```
$ gcc example.o -lgsl -lcblas -lm
```

Wersje alternatywne

Plik .bashrc

export **LD_LIBRARY_PATH=$LD_LIBRARY_PATH:**/opt/My/lib

Biblioteki „czasu kompilacji”

- Szablony C++ umieszcza się w nagłówkach
- Oparte na szablonach biblioteki nie wymagają kompilacji do plików obiektowych
  - „Biblioteki czasu kompilacji”
  - STL, Boost i wiele innych

Co i gdzie? Język C

- Pliki nagłówkowe: **deklaracje**

```
extern int N;
int f(int n);
struct X;
```

- Pliki źródłowe: **definicje**

```
int N = 100;
int f(int n)
{
    return n*n;
}
struct X{
  int i, j;
};
```

Co i gdzie? Język C++

- Pliki nagłówkowe: **deklaracje + definicje funkcji inline**

```
extern int N;
int f(int n);
struct X;
inline int g() {
  return 10;
}
```

- Pliki źródłowe: **definicje**

```
int N = 100;
int f(int n)
{
   return n*n;
}
struct X{
  int i, j;
};
```

Automatyzacja kompilacji

- make + Makefile
  - cmake / ccmake
  - qmake (Qt)
- ninja
- GNU autotools
  - ./configure

- Kompiluj tylko te jednostki translacji, które są niezbędne
- Automatycznie przygotuj różne środowiska kompilacji zależnie od platformy, na którą kompilujesz

Biblioteki statyczne

- TBA

Biblioteki dynamiczne

- TBA