Le Langage C++
et
La Programmation Objet

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Organisation

- 4 Cours Magistraux de 2h
- 3 séances sur machines (TPs) de 4h

- Evaluation
  - Un examen écrit (50% note finale)
  - Une remise en fin de trimestre (50% note finale)
    - Compte-rendu
    - Code source

Organisation

Page du cours et des TDs:

http://manao.inria.fr/perso/~pac/cpp.php

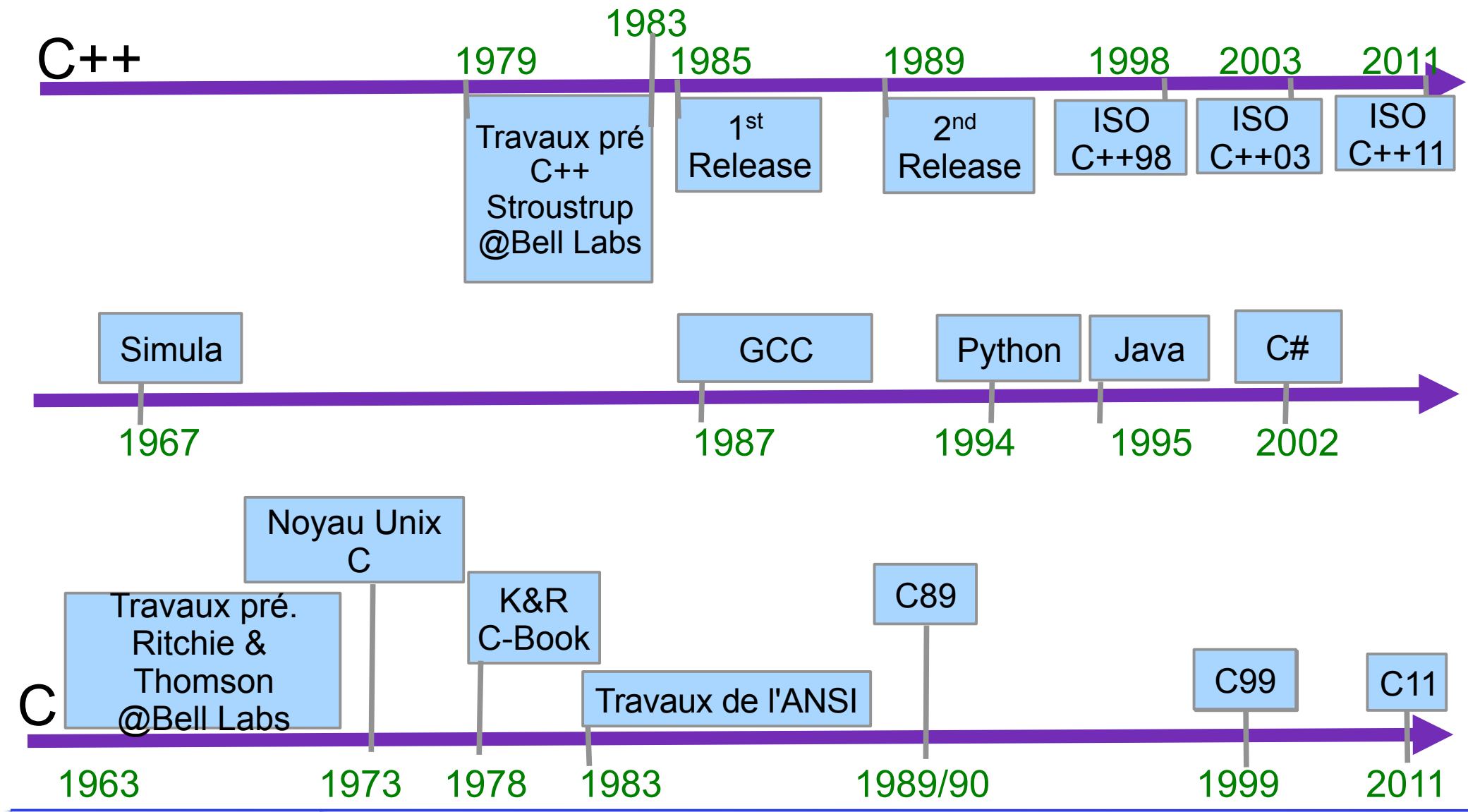
➔ A consulter Régulièrement !

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

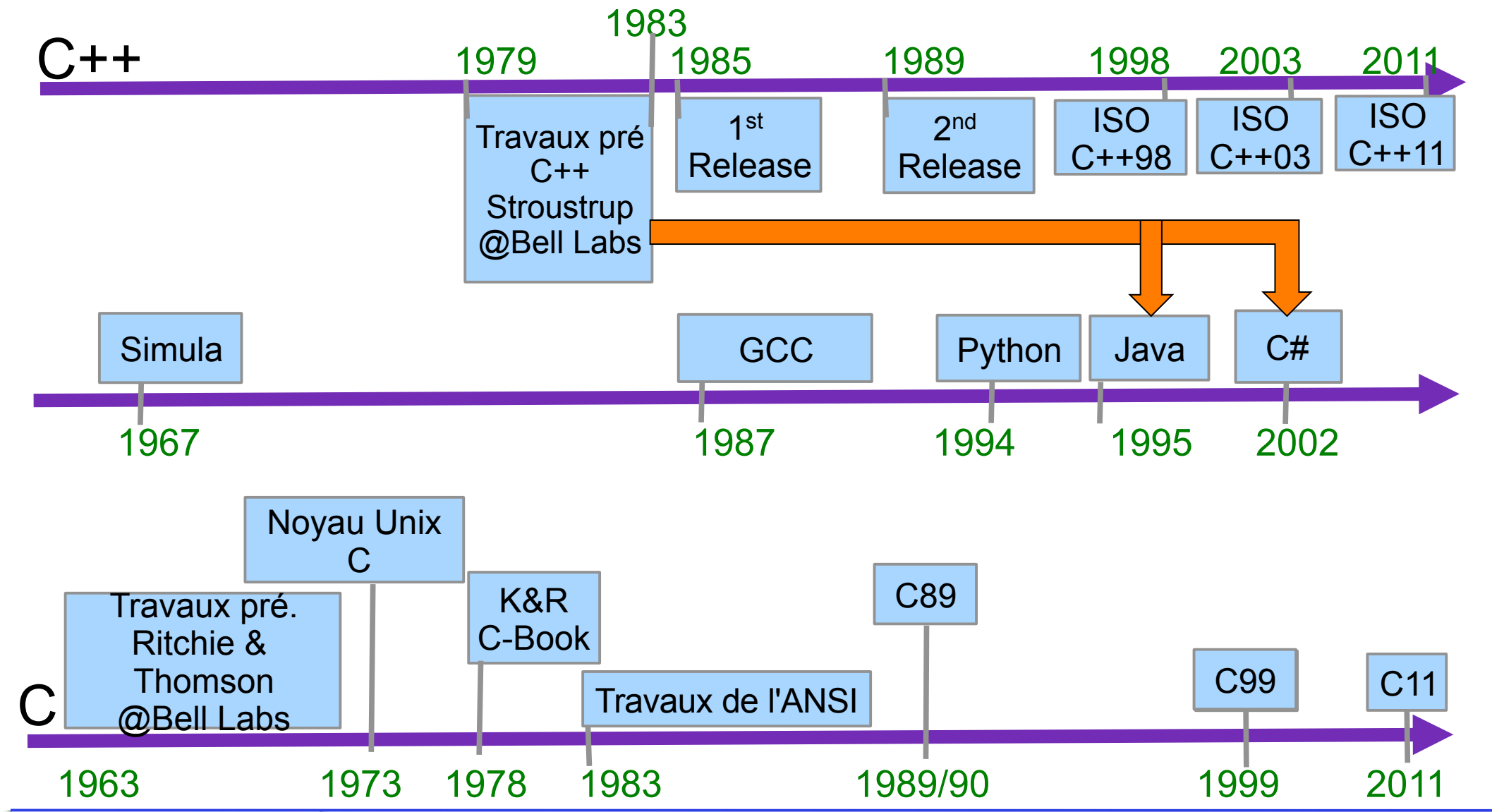
- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

Historique C, C++, Java et C#, Python

Historique C, C++, Java et C#, Python



Historique C, C++, Java et C#, Python



INSTITUT d'OPTIQUE GRADUATE SCHOOL

Design du C++

- Syntaxe épurée du C

  → Conserver le savoir-faire != compatibilité du code

- Améliorer la productivité du progammeur
  - Simplifier le langage à la place du compilateur
- Code aussi portable et efficace que le C
  - « Zero-overhead » principle
- Liberté : responsabiliser le programmeur
  - e.g., gestion de la mémoire

Design du C++

- Limitations historique du C
  - Assembleur de haut niveau != langage de prog. impératif
  - L'absence
    - de typage des macros
    - de passage par adresse
  - La rarification des noms d'identificateurs

Caratéristiques générales du C++

- Langage Compilé
- Syntaxe ala C
  - Transition facilitée des développeurs
- Typage statique
- C++
  - N'est pas une extension objet du C
  - Un programme C ne compile pas forcément en C++
  - rmes en appliquant la norme ISO.

Caratéristiques générales du C++

- « Grande Comptabilité » avec le C
  - Conversion librairies C pour utilisation depuis C++

- Gestion des erreurs avec des exceptions
- Multiples paradigmes de programmation
- Multiplateformes en appliquant la norme ISO.

Caractéristiques générales du C++

Paradigmes de programmation :

- Impérative
  - haut/bas niveau
- Programmation par objet O
- Générique (templates)
- Meta-templates....

→ Langage « hybride »

Langage et librairie standard C vs C++

LANGAGE C++
Core Language
Syntaxe & Sémantique

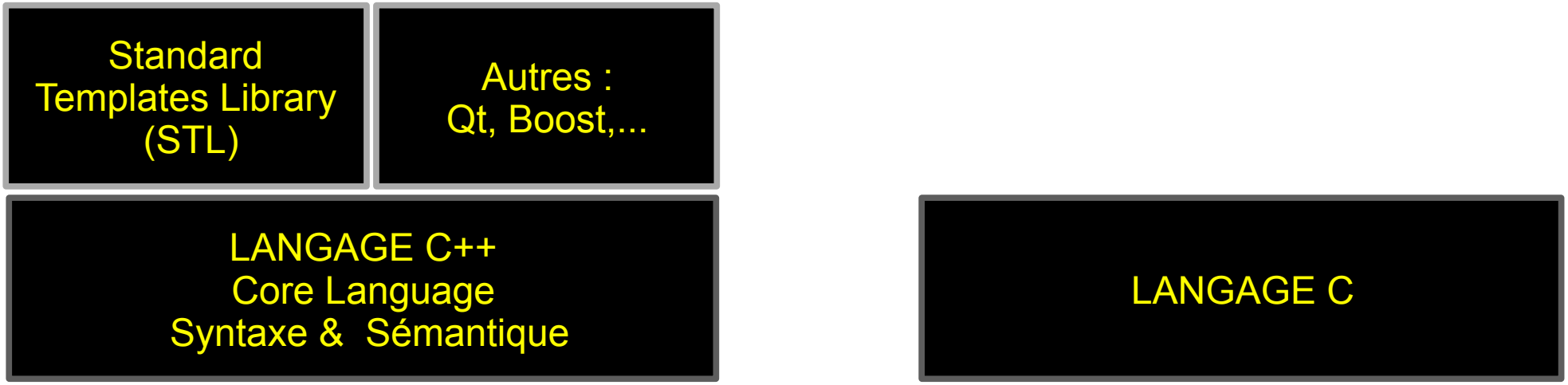
Langage et librairie standard C vs C++

LANGAGE C++
Core Language
Syntaxe & Sémantique

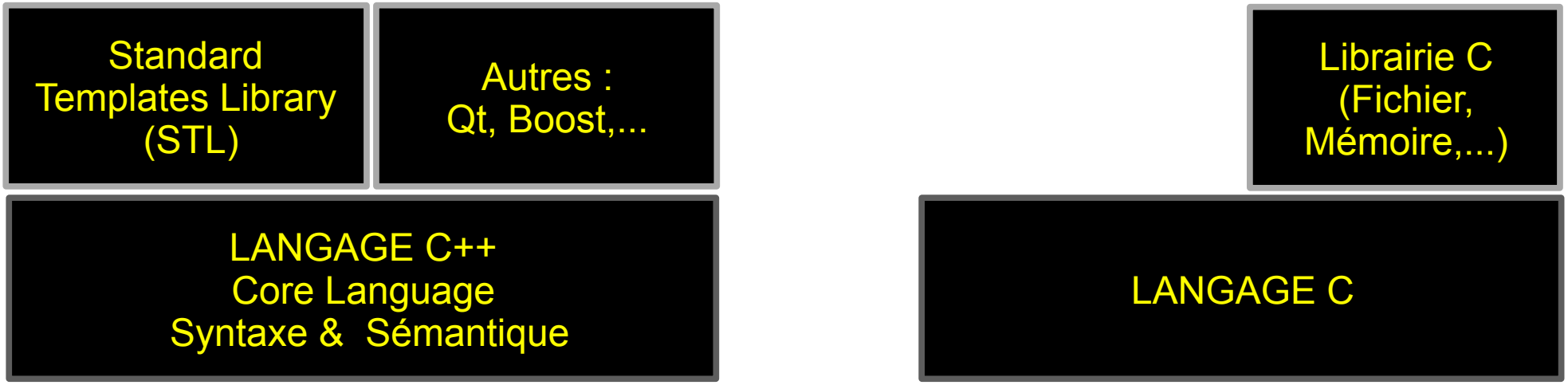
LANGAGE C

Langage et librairie standard C vs C++

Langage et librairie standard C vs C++

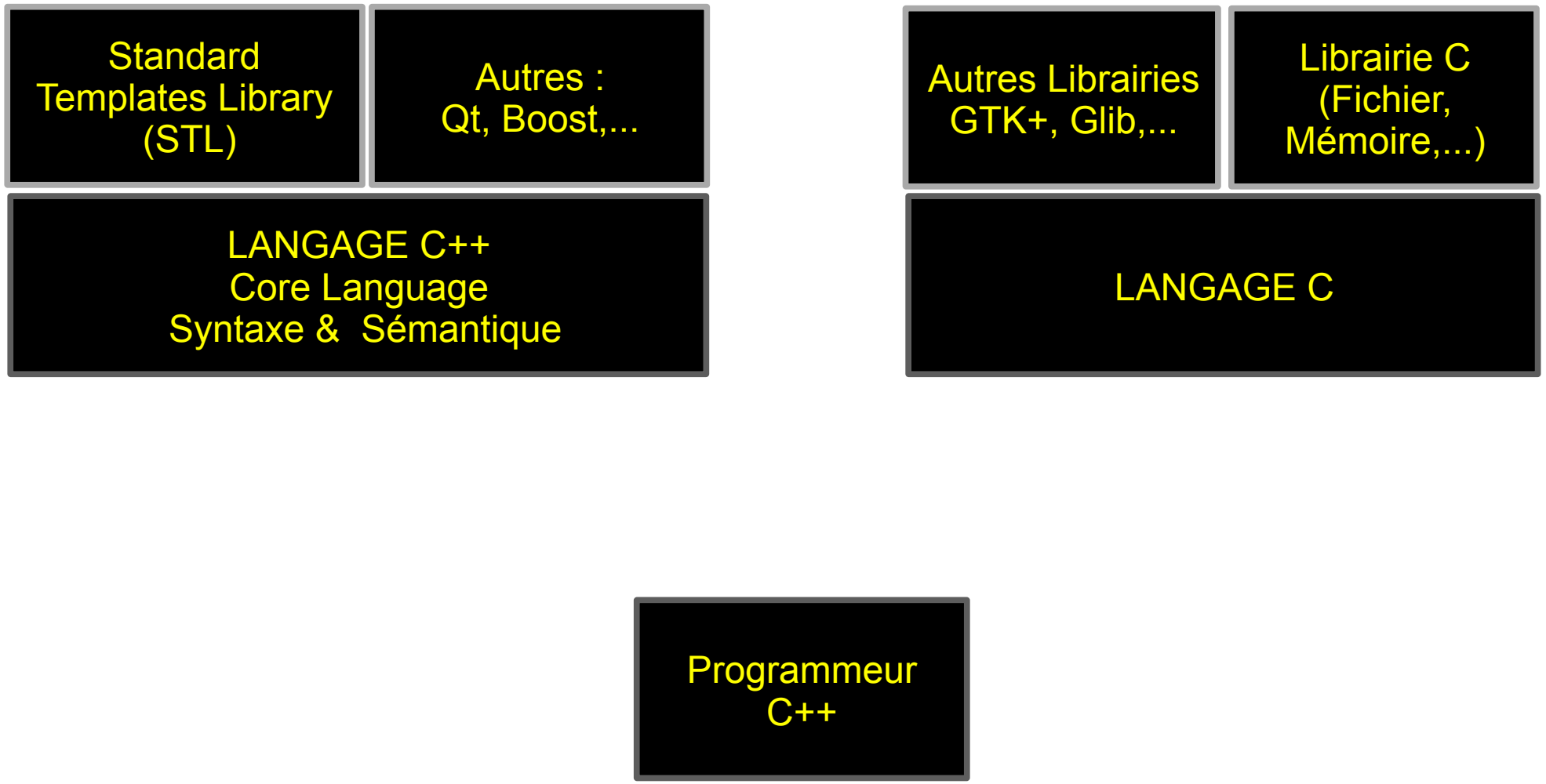


Langage et librairie standard C vs C++

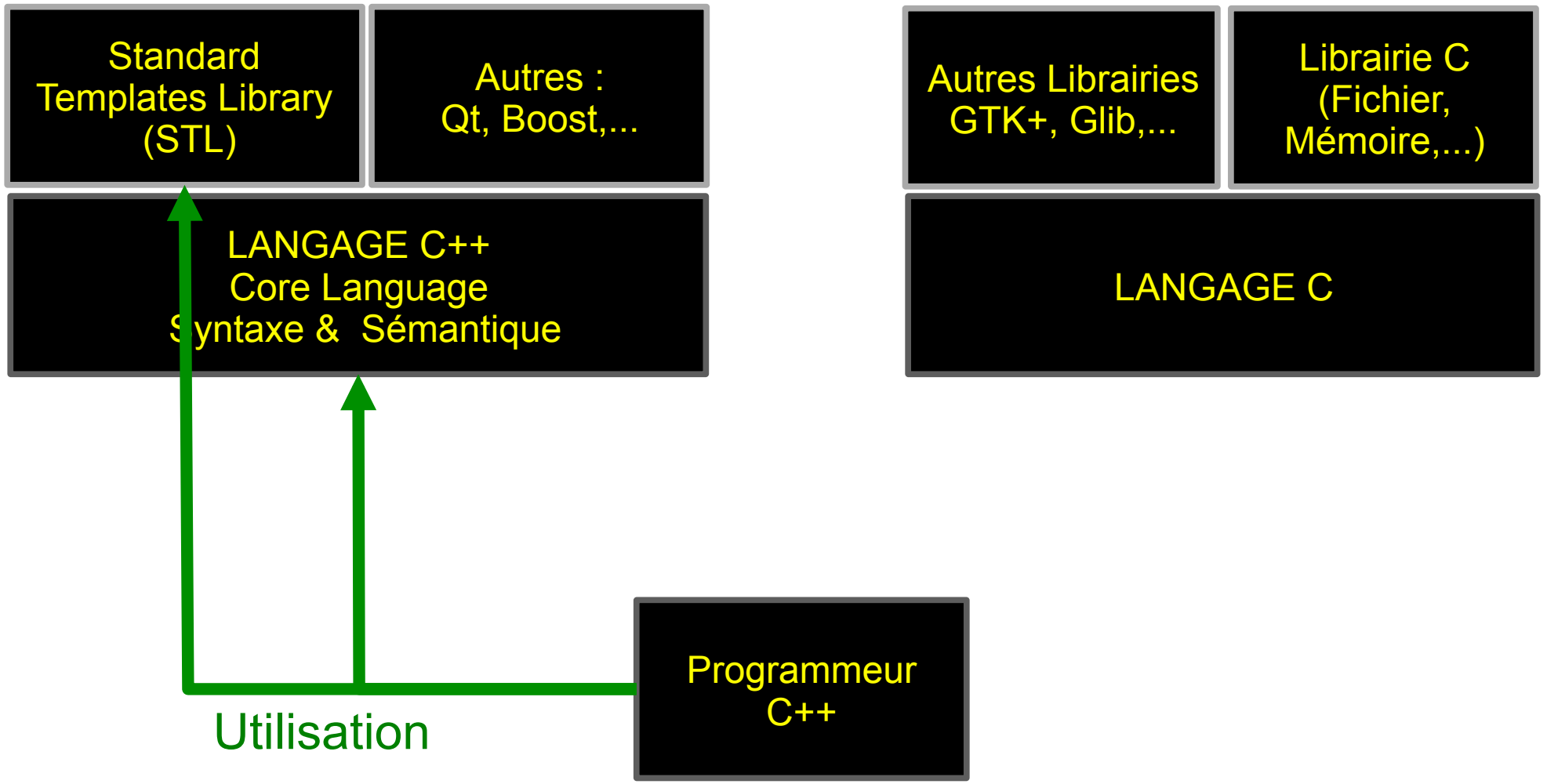


Langage et librairie standard C vs C++

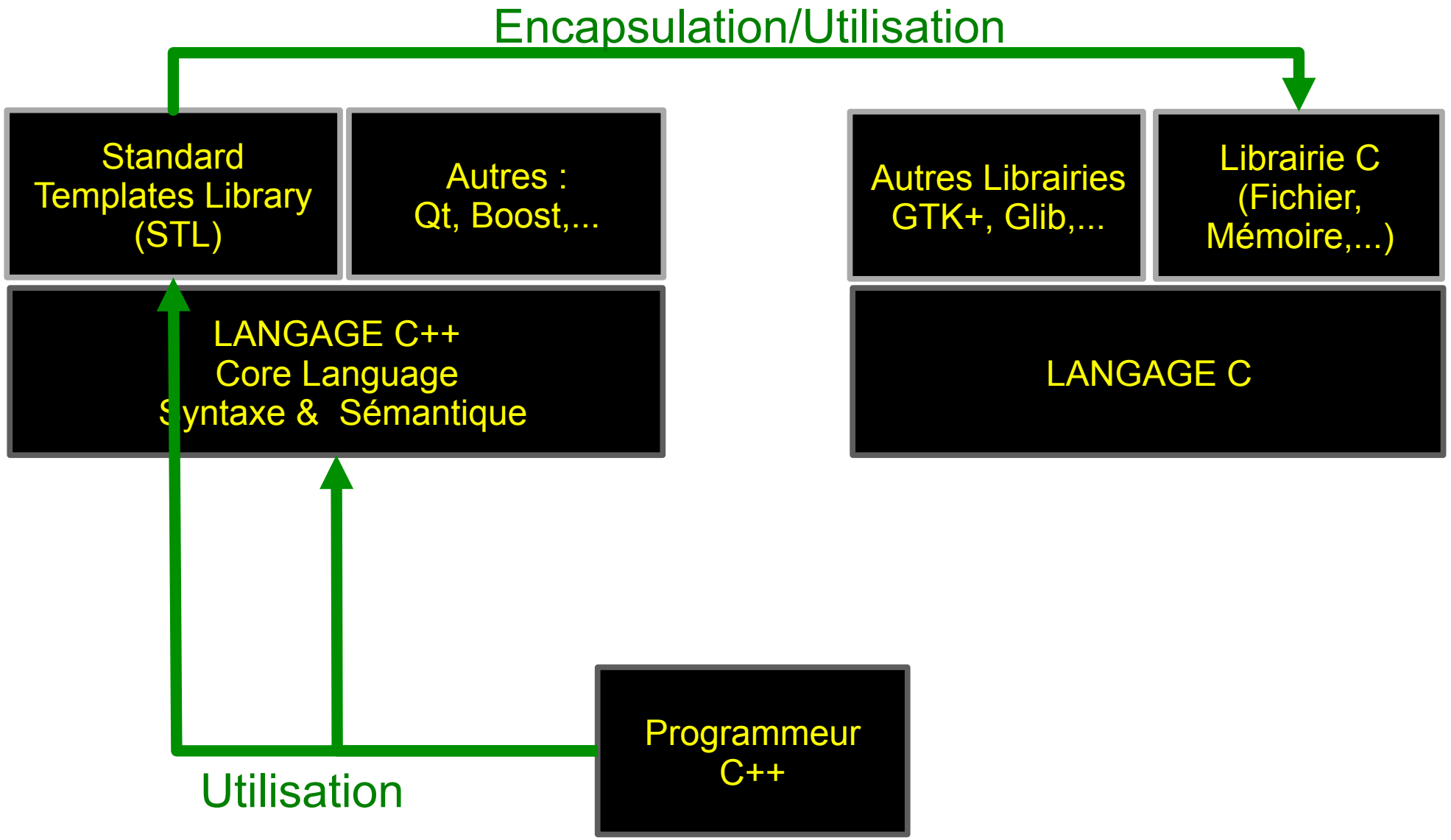
Langage et librairie standard C vs C++



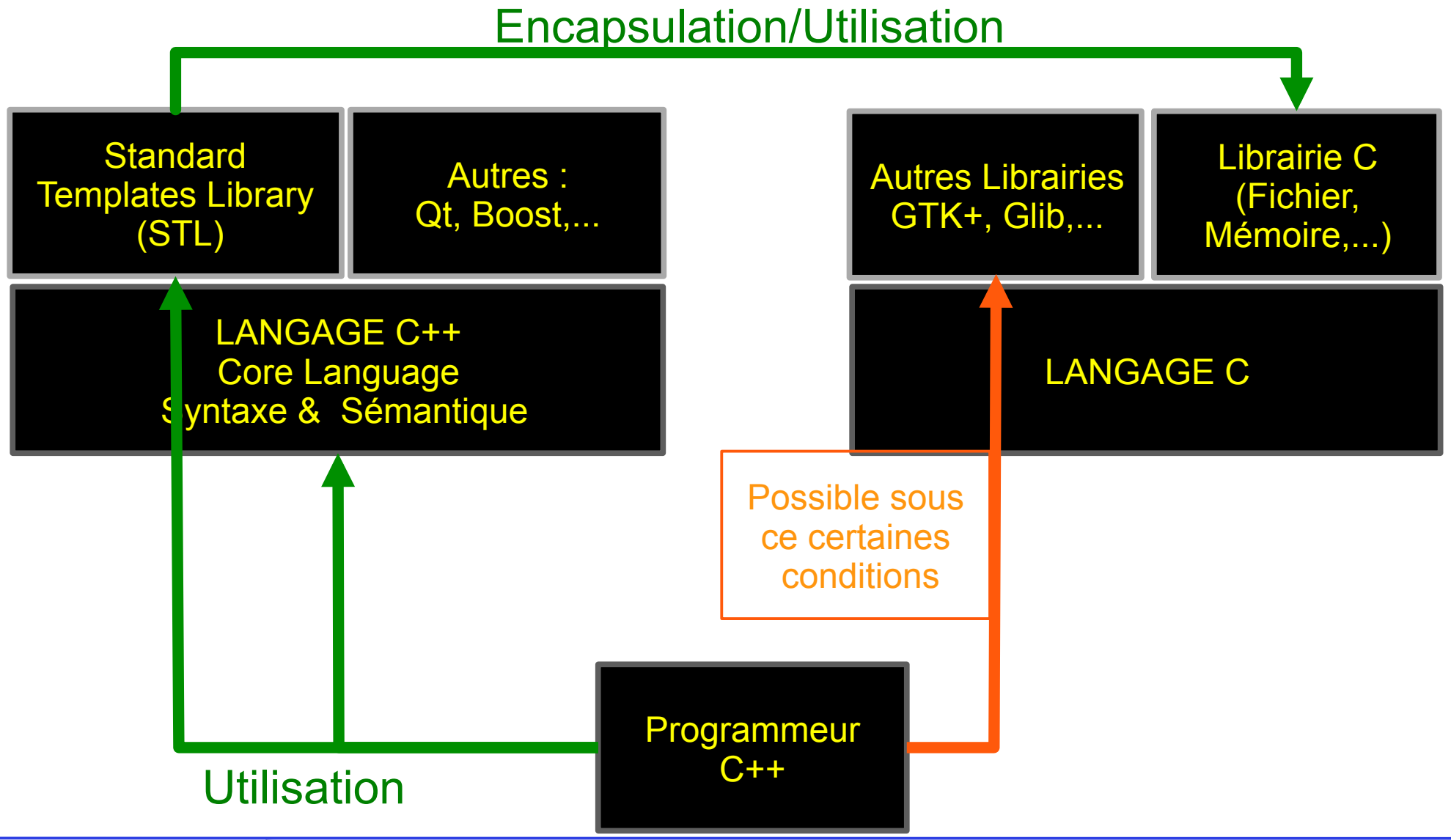
Langage et librairie standard C vs C++



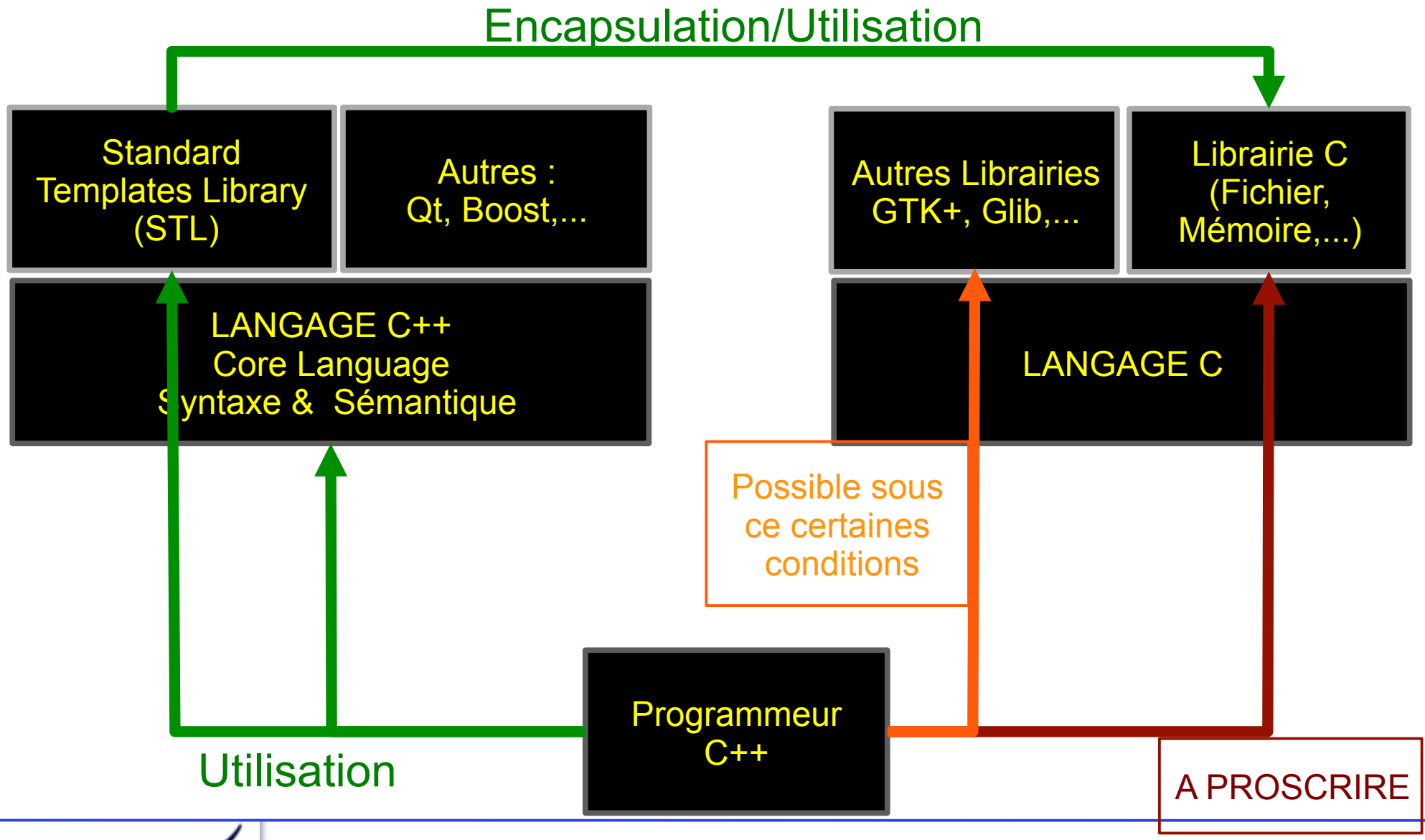
Langage et librairie standard C vs C++



Langage et librairie standard C vs C++



Langage et librairie standard C vs C++



Standardisations/Normes du C++

- C++98
  - 1998 : ISO/IEC 14882:1998
- C++03
  - 2003 : ISO/IEC 14882:2003
- C++TR1
  - 2007 : ISO/IEC TR 19768:2007
- C++11 (avant C++0x)
  - 2011 : ISO/IEC 14882:2011
- C++ 2014
  - ISO/IEC 14882:2014

Objectifs du cours et des TD

- Maîtrise des concepts clés
  - Référence (C++)
  - *Constness* (C++)
  - Programmation Objet (C++ mais commun à Java, C#,…)

- Initiation en C++
  - Standard Templates Library (STL)
    - Conteneurs
    - Algorithmes
  - Templates
  - Design Patterns

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

Le processus de création d'un programme

- Programmation / Développement

  → Programmeur 1

- Traduction en langage machine

  → Compilateur + Editeur de lien (linker)

- Test du programmeur

  → Programmeur 1 ou autre personne

- Déploiement / Installation

  → Programmeur 1 ou 2

Le processus de création d'un programme

- Programmation / Développement

  → Programmeur 1

- Traduction en langage machine

  → Compilateur + Editeur de lien (linker)

- Test du programmeur

  → Programmeur 1 ou autre personne

- Déploiement / Installation

  → Programmeur 1 ou 2

Traduction en langage machine

- Langages interprétés (e.g., Scheme, Perl, ...)
  - code source → activité et/ou instructions machines
  - Interpréteur et programme « fits » en mémoire ☹
  - Vitesse d'exécution réduite
  - Débogage facilité
- Langages compilés (e.g., C, C++, …)
  - code source → instructions machines
  - Taille mémoire plus faible pour le code compilé ☺
  - Compilation séparée des fichiers sources

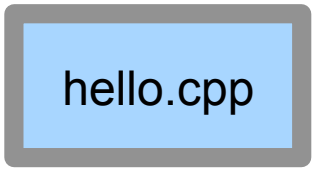
    →Création de librairie

Processus de génération de code machine

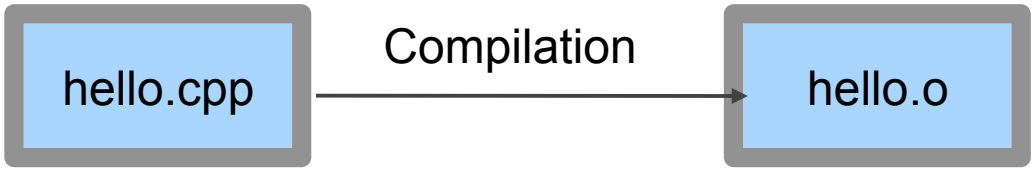
1. Compilation du code source

2. Edition de lien et création du binaire

Compilation et Edition de lien

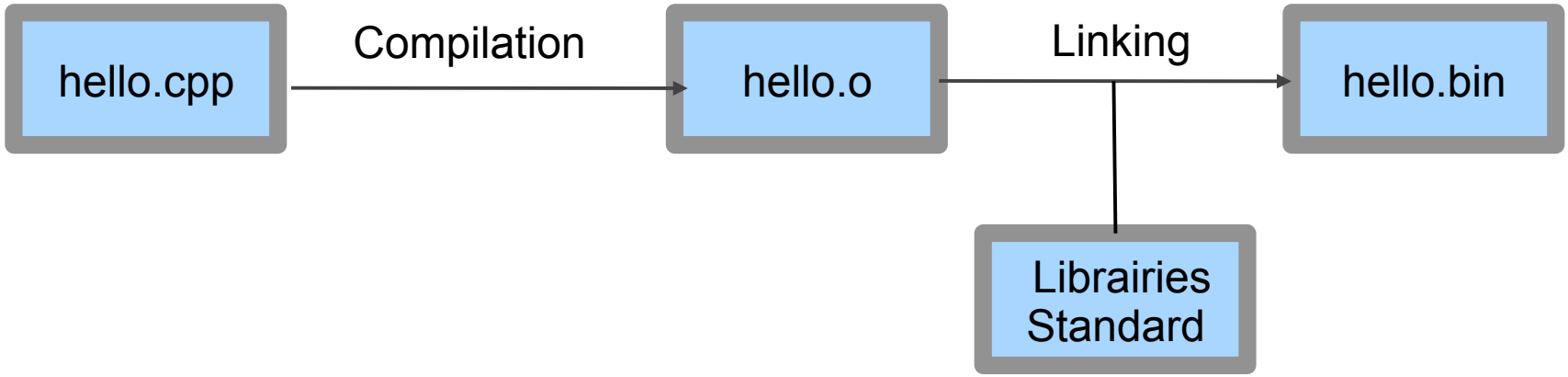


Compilation et Edition de lien



Compilation : production d'instructions machines

Compilation et Edition de lien



Compilation : production d'instructions machines

Edition de lien :

- Vérification que les liens pointent sur des instructions valides
- Ajout de code pour l'éxécution du programme

Librairies

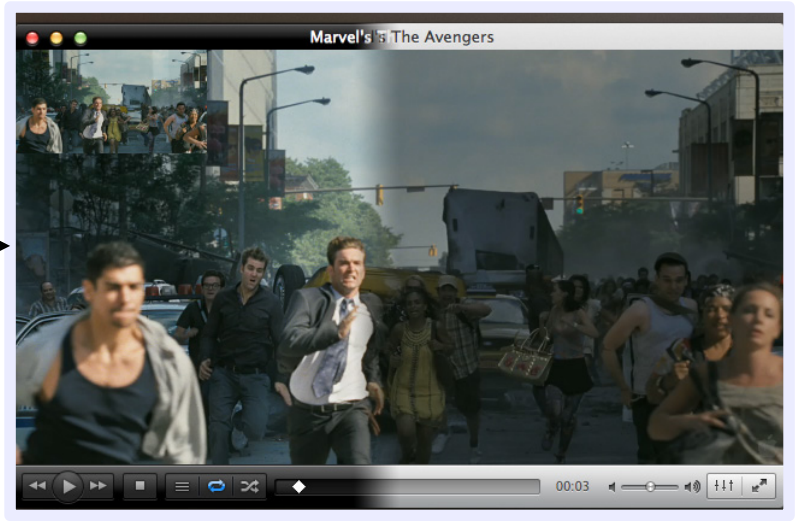
- Librairies
  - ensembles de fonctions/classes/procedures
  - pas de point d'entrée (i.e., main(...)) != application
- Facilité par la compilation séparée
- But: Partager les fonctionnalités développés dans la librairie pour différentes applications
- Deux type de librairies
  - Dynamiques (.so, .dll, .dylib)
  - Statiques (.a, .lib)

Illustration d'utilisation de Librairies

Librairie

- GUI
- XML
- Thread
- ....

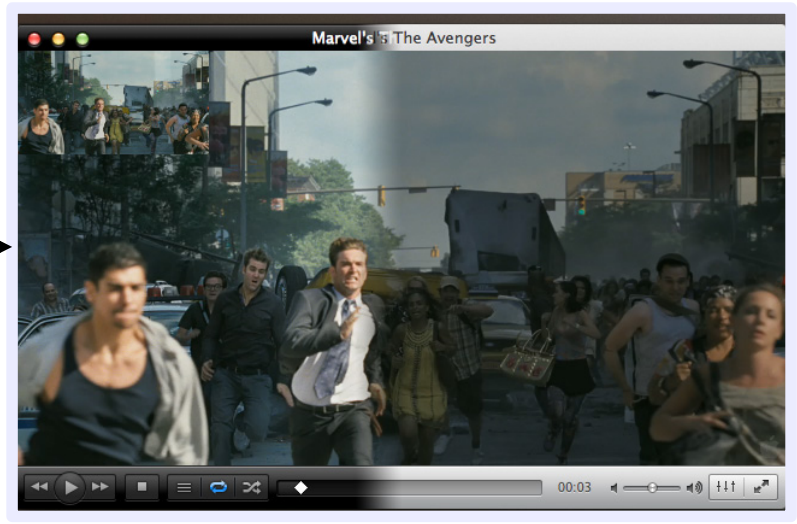
Illustration d'utilisation de Librairies



Librairie

- GUI
- XML
- Thread
- ....

Illustration d'utilisation de Librairies



Librairie

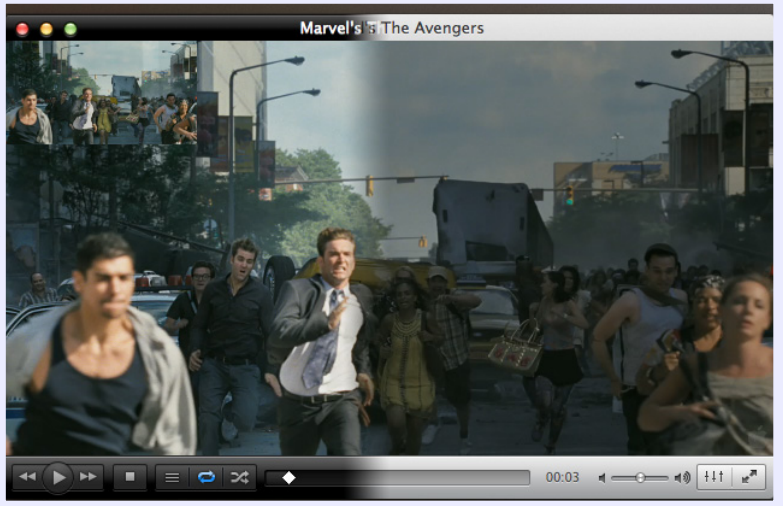
- GUI
- XML
- Thread
- ....

Illustration d'utilisation de Librairies

Librairie

- GUI
- XML
- Thread
- ....

VLC



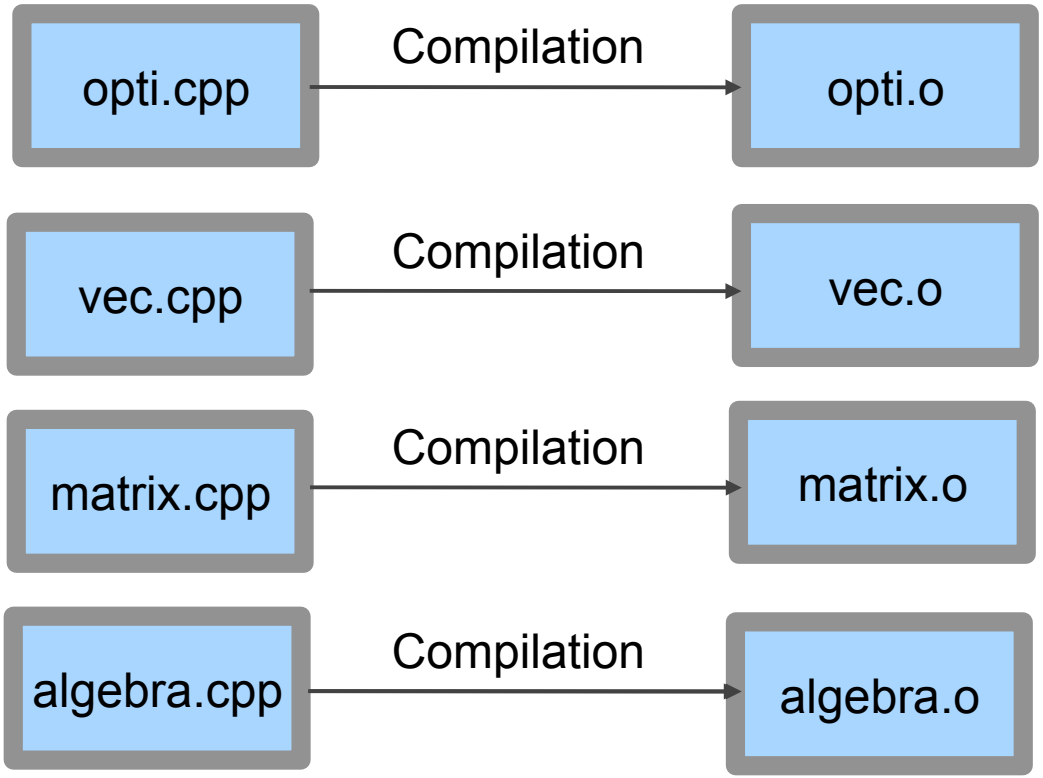
NUKE

La compilation

- Pré-processeur
  - Remplacement de toutes les macros.
- Création de l'arbre représentant le code
  - GCC GENERIC: Language-independent representation(generated by Front Ends)
  - Vérification du type (*type-safe checking*) **statique** en C++
- Passe globale d'optimisation
  - GCC GIMPLE

    ➔ Tuple representation used by Tree SSA optimizers
- Génération des instructions machines (IR)

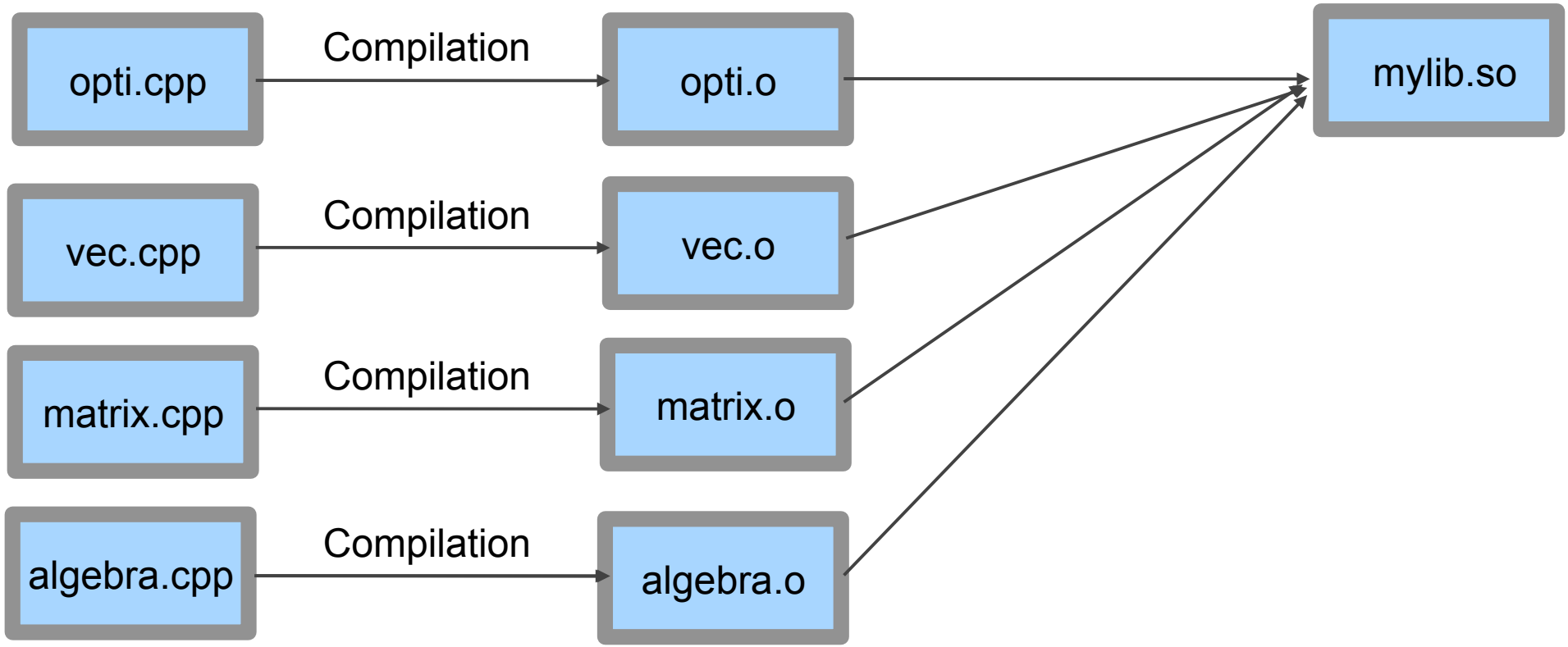
Création de librairies



COMPILATION SEPARÉE :

- minimise les besoins en mémoire
- optimisée sur machine multi-coeur

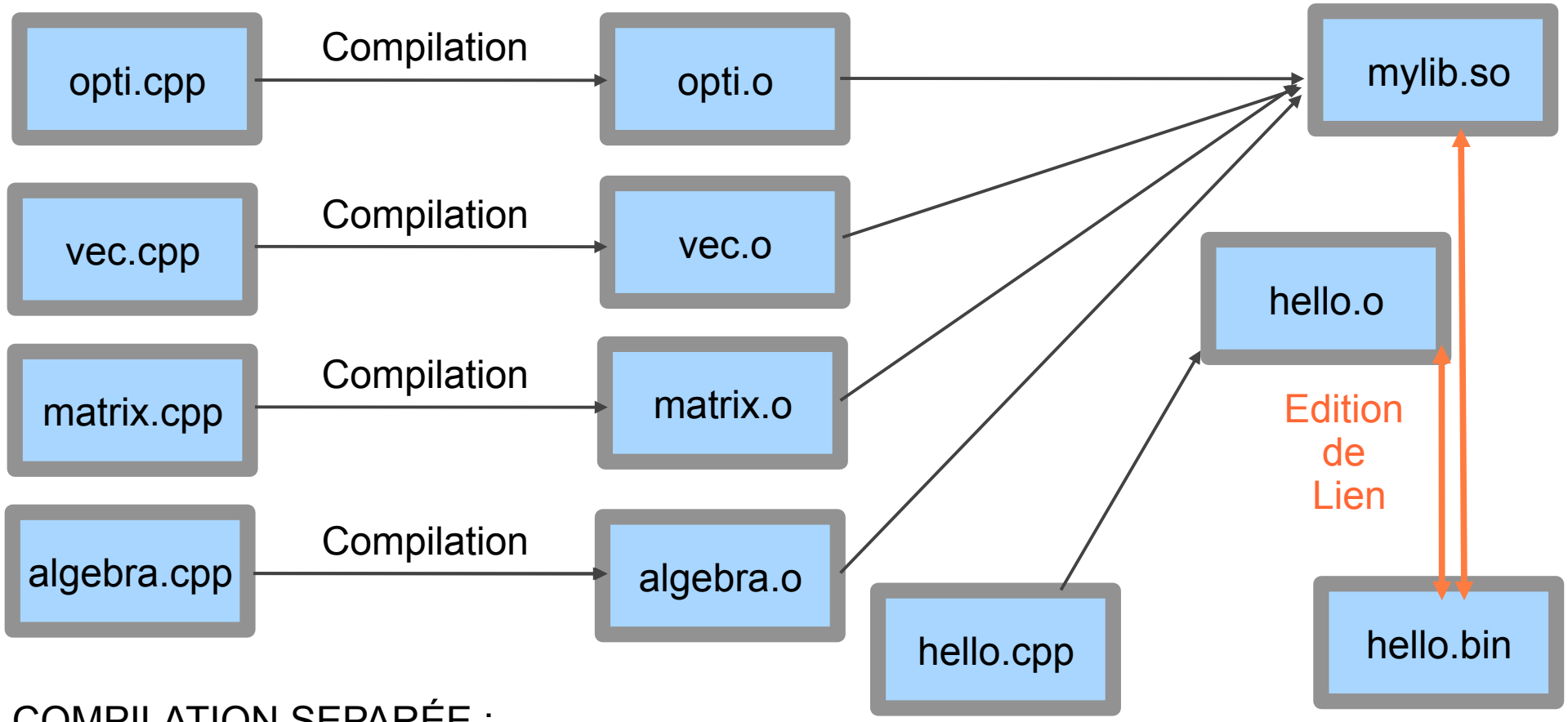
Création de librairies



COMPILATION SEPARÉE :

- minimise les besoins en mémoire
- optimisée sur machine multi-coeur

Création de librairies



COMPILATION SEPARÉE :

- minimise les besoins en mémoire
- optimisée sur machine multi-coeur

Déclaration vs Définition

- Déclaration :
  - Introduction du nom de l'identifiant au compilateur
  - Identifiant = nom de variables,classes, fonctions,...
  - Nécessaire pour la compilation de fichier séparé
- Définition :
  - Allocation d'espace mémoire pour un identifiant
  - Une définition peut être aussi une déclaration
- *One-Definition-Rule* :
  - Déclarations multiples MAIS une seule définition !

Déclaration et définition de fonction

functions.hpp

```
// Déclaration de func1
int
func1(int, int);

// Déclaration de func2
int
func2(int width, int length);
```

functions.cpp

```
// Définition

//Implémentation de func2

int
func2( int width, int length)
{
 return 2 * (width+length) ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Déclarations de fonctions

En C++

```
//  Déclaration de func2
int func2() ;
```

En C

```
//  Déclaration de func2
int func2() ;
```

Combien d'arguments pour func2 ?

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Déclarations

En C++

```
//  Déclaration de func2
int func2() ;
```

En C

```
//  Déclaration de func2
int func2() ;
```

Combien d'arguments pour func2 ?

N'importe quel
nombre
ET
type

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Déclarations

En C++

```
//  Déclaration de func2
int func2() ;
```

En C

```
//  Déclaration de func2
int func2() ;
```

Combien d'arguments pour func2 ?

AUCUN

N'importe quel
nombre
ET
type

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Déclarations de variables

vars.hpp

```
int a ;
```

Déclarations de variables

vars.hpp

```
int a ; // != Déclaration. Déclaration ET Définition
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Déclarations de variables

vars.hpp

```
int a ; // != Déclaration. Déclaration ET Définition

extern int a ; // == Déclaration
```

Remarques :

- `extern` est optionel pour la déclaration de fonction
- `extern` rend le code plus (+) lisible
- `extern` « promet » une définition ailleurs dans ou hors du fichier

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Exemple de déclarations et définitions

exs.cpp

```
extern int i;
float b ;
extern float f(float) ;
int i ;
int h(int x) { return x+1 ;}

float f(float x) { return x*x;}
int main(int argc, char** argv)
{
  i = 2 ;
  b = -3.14159f ;
  f(b) ;
  int foo = h(i) ;
  return EXIT_SUCCESS ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Exemple de déclarations et définitions

exs.cpp

```
extern int i;                                   //Déclaration sans définition
float b ;
extern float f(float) ;
int i ;
int h(int x) { return x+1 ;}

float f(float x) { return x*x;}
int main(int argc, char** argv)
{
  i = 2 ;
  b = -3.14159f ;
  f(b) ;
  int foo = h(i) ;
  return EXIT_SUCCESS ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Exemple de déclarations et définitions

exs.cpp

```
extern int i;                                   //Déclaration sans définition
float b ;                                       //Déclaration et définition
extern float f(float) ;
int i ;
int h(int x) { return x+1 ;}

float f(float x) { return x*x;}
int main(int argc, char** argv)
{
  i = 2 ;
  b = -3.14159f ;
  f(b) ;
  int foo = h(i) ;
  return EXIT_SUCCESS ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Exemple de déclarations et définitions

exs.cpp

```
extern int i;                          //Déclaration sans définition
float b ;                              //Déclaration et définition
extern float f(float) ;                //Déclaration de fonction
int i ;
int h(int x) { return x+1 ;}

float f(float x) { return x*x;}
int main(int argc, char** argv)
{
  i = 2 ;
  b = -3.14159f ;
  f(b) ;
  int foo = h(i) ;
  return EXIT_SUCCESS ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Exemple de déclarations et définitions

exs.cpp

```
extern int i;                                   //Déclaration sans définition
float b ;                                       //Déclaration et définition
extern float f(float) ;                         //Déclaration de fonction
int i ;                                         //Déclaration et définition
int h(int x) { return x+1 ;}

float f(float x) { return x*x;}
int main(int argc, char** argv)
{
  i = 2 ;
  b = -3.14159f ;
  f(b) ;
  int foo = h(i) ;
  return EXIT_SUCCESS ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Déclarations de variables

vars.hpp

```
int a ; // != Déclaration. Déclaration ET Définition

extern int a ; // == Déclaration
```

Remarques :

- `extern` est optionnel pour la déclaration de fonction
- `extern` rend le code plus (+) lisible

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Compilation et *linking* sous Unix

- Compilation: g++ toto.cpp -c
- Edition de lien:
  - ld hello.o -o hello.exe -lcrt1.10.6.o -lstdc++ -lSystem -lgcc
- En pratique (2 en 1):
  - g++ toto.c -o toto.exe

- Librairies partagée et statique vues en TD

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

Premier exemple en C++

```
1: #include <iostream>

2: int main(int argc, char** argv )
   {

3:     std::cout << " HOWDY !!! " << std::endl;

4:     return 0;
   }
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Inclusion des fichiers en-tête (1/2)

```
// Header type <...>
#include <iostream>                    // Header lib STL (C++)
#include < iostream >                  // Ne marchera PAS
```

La syntaxe <...>

- Utiliser pour la librairie standard
- Abstraction du système de fichier (nom, répertoire) de l'OS

- Le préprocesseur se charge :
  - De trouver quel fichier correspond au nom (iostream ici)
  - De remplacer la directive d'inclusion par le contenu du fichier trouvé (copy & paste)

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Inclusion des fichiers en-tête (2/2)

```
// Header type "…"
#include "myvect.hpp"              // Depuis dir. courant
#include "math/myvect.hpp"         // Chemin vers rep. math
```

La syntaxe "..."

- Le préprocesseur se charge :
  - d'inclure le fichier spécifié à partir du répertoire courant d'appel du compilateur/pré-processeur
  - De remplacer la directive d'inclusion par le contenu du fichier trouvé (copy & paste)

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Les *namespaces*

Problématique :

norm_comp.hpp

```
float normInf( float, float, float );
float normL2( float, float, float );
```

norm.hpp

```
float
normInf( float, float, float);

float
fastNormL2( float, float, float);
```

Les *namespaces*

Problématique :

norm_comp.hpp

```
float normInf( float, float, float );
float normL2( float, float, float );
```

norm.hpp

```
float
normInf( float, float, float);

float
fastNormL2( float, float, float);
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"


int main(int argc, char** argv )
{
   float b=fastNormL2(3.0f,4.0f,5.0f);
   float z = normInf(1.0f,2.0f,-3.0f);

   std::cout << a << " " << b;
   return 0;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Les *namespaces*

Problématique :

norm_comp.hpp

```
float normInf( float, float, float );
float normL2( float, float, float );
```

?

norm.hpp

```
float
normInf( float, float, float);

float
fastNormL2( float, float, float);
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"


int main(int argc, char** argv )
{
    float b=fastNormL2(3.0f,4.0f,5.0f);
    float z = normInf(1.0f,2.0f,-3.0f);

    std::cout << a << " " << b;
    return 0;
}
```

Les *namespaces*

norm_comp.hpp

```
namespace iogs_math {
 float  normInf(float, float, float );
 float  normL2(float, float, float );
}
```

norm.hpp

```
namespace ioa_math {
 float
 normInf(float, float, float);

 float
 fastNormL2(float, float, float);
}
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"



int main(int argc, char** argv )
{
 float b=fastNormL2(3.0f,4.0f,5.0f);

 float z = normInf(1.0f,2.0f,-3.0f);

 std::cout << a << " " << b;
 return 0;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Les *namespaces*

norm_comp.hpp

```
namespace iogs_math {
 float  normInf(float, float, float );
 float  normL2(float, float, float );
}
```

norm.hpp

```
namespace ioa_math {
 float
 normInf(float, float, float);

 float
 fastNormL2(float, float, float);
}
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"

using namespace ioa_math;

int main(int argc, char** argv )
{
 float b=fastNormL2(3.0f,4.0f,5.0f);

 float z = normInf(1.0f,2.0f,-3.0f);

 std::cout << a << " " << b;
 return 0;
}
```

Les *namespaces*

norm_comp.hpp

```
namespace iogs_math {
 float  normInf(float, float, float );
 float  normL2(float, float, float );
}
```

norm.hpp

```
namespace ioa_math {
 float
 normInf(float, float, float);

 float
 fastNormL2(float, float, float);
}
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"

using namespace ioa_math;

int main(int argc, char** argv )
{
 float b=fastNormL2(3.0f,4.0f,5.0f);

 float z = normInf(1.0f,2.0f,-3.0f);

 std::cout << a << " " << b;
 return 0;
}
```

Les *namespaces*

norm_comp.hpp

```
namespace iogs_math {
 float  normInf(float, float, float );
 float  normL2(float, float, float );
}
```

norm.hpp

```
namespace ioa_math {
 float
 normInf(float, float, float);

 float
 fastNormL2(float, float, float);
}
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"

using namespace ioa_math;

int main(int argc, char** argv )
{
 float b=fastNormL2(3.0f,4.0f,5.0f);

 float z = normInf(1.0f,2.0f,-3.0f);

 std::cout << a << " " << b;
 return 0;
}
```

Les *namespaces*

norm_comp.hpp

```
namespace iogs_math {
 float  normInf(float, float, float );
 float  normL2(float, float, float );
}
```

norm.hpp

```
namespace ioa_math {
 float
 normInf(float, float, float);

 float
 fastNormL2(float, float, float);
}
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"

//using namespace ioa_math;

int main(int argc, char** argv )
{
 float b=fastNormL2(3.0f,4.0f,5.0f);

 float z = normInf(1.0f,2.0f,-3.0f);

 std::cout << a << " " << b;
 return 0;
}
```

Les *namespaces*

norm_comp.hpp

```
namespace iogs_math {
 float  normInf(float, float, float );
 float  normL2(float, float, float );
}
```

norm.hpp

```
namespace ioa_math {
 float
 normInf(float, float, float);

 float
 fastNormL2(float, float, float);
}
```

main.cpp

```
#include "norm_comp.hpp"
#include "norm.hpp"

//using namespace ioa_math;

int main(int argc, char** argv )
{
 float b=
    iogs_math::fastNormL2(3.0f,4.0f,5.0f);
 float z =
    ioa_math::normInf(1.0f,2.0f,-3.0f);

 std::cout << a << " " << b;
 return 0;
}
```

Retour sur le premier exemple en C++

```
1: #include <iostream>
2:
3:
4:int main(int argc, char** argv )
5:{
6:   std::cout << " HOWDY !!! " << std::endl;
7:   return 0;
8:}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Retour sur le premier exemple en C++

```
1: #include <iostream>
2:
3: using namespace std;
4:int main(int argc, char** argv )
5:{
6:   cout << " HOWDY !!! " << endl;
7:   return 0;
8:}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Les *namespaces*

- La directive *using namespace*
  - S'applique uniquement à la portée courante
    - Fichier
    - Bloc { }
  - Ne doit PAS s'utiliser dans les fichiers en-tête

  ➔ pollution et effet de bord

- Les blocs { } *namespace* sont utilisées dans les .cpp et les .hpp

Kit de survie : les entrées-sorties

- En ligne de commande
  - Sorties. Avec l'objet cout (console ouput)
  - Entrées. Avec l'objet cin (console input
- Chaîne de caractères (string)
- Pour les fichiers
  - Lecture
  - Ecriture

Kit de survie : sorties formatées

```
1:  #include <iostream> //déclaration des stream (cout,cin,...)
2:  using namespace std;
3:  int main(int argc, char** argv)
4:  {
5:   cout << "nb en decimal" << dec << 15 << endl;
6:   cout << "nb en octal" << oct << 15 << endl;
7:   cout << "nb en hexa" << hex << 15 << endl;
8:   cout << "nb flottant" << 3.14159 << endl;
9:   cout << "caracteres non affiche" << char(27) << endl;
10:  return 0 ;
     }
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Kit de survie : sorties concaténées

```
1: #include <iostream> //déclaration des stream (cout,cin,...)
2: using namespace std;
3: int main(int argc, char** argv)
4: {
5:  cout << " En C++ il est tout a fait possible "
6:           " sur plusieurs lignes tant qu il n y a pas de"
    " caracteres de ponctuation \n " ;
7:
8:  cout << " Si non peut écrire aussi, "
9:       << "caracteres non affiche" << endl;
10: return 0 ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Kit de survie : entrées

```
1: #include <iostream> //déclaration des stream (cout,cin,...)
2: using namespace std;
3: int main(int argc, char** argv)
4: {
5:  int number;
6:  cin >> number;
7:  cout << "nb en octal = 0" << oct << number << endl;
8:  cout << "nb en hexa = 0x" << hex <<number << endl;
9:  return 0 ;
10:}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Kit de survie : string

```
1: #include <string>   // declaration des string
2: #include <iostream> // declaration des stream (cout,cin,...)
3:
4: using namespace std;
5:
6: int main(int argc, char** argv) {
7:  string s1, s2; // String vide
8:
9:  string s3 = "Hello  World !" ;   // Initialisation
10: string s4(". How is it going "); // Initialisation aussi
11:
12: s1  = s3 + " " + s4;             // Concatenation
13: s1 += " today ? " + s2;          // Ajout a la fin
14:
15: cout << s1 + " !" << endl ;
16: return 0 ;
17:}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Kit de survie : les fichiers

```
1: #include <fstream> //stream fichiers (ifstream,ofstream...)
2: #include <string> // objet string
3: #include <iostream> // stream (cout,cin,...)
4: using namespace std;
5: int main(int argc, char** argv) {
6:  ifsteam in("toto.txt");                  // Ouverture pour lecture
7:  ofstream out("toto-c.txt");              // Ouverture pour écriture
8:  string s ;
9:  while( getline(in,s) )                  // Enleve le char newline
10:    out << s << "\n" ;                   // On le remet
11: return 0 ;
12:}
```

Kit de survie : les vecteurs

```
1: #include <fstream> //stream fichiers (ifstream,ofstream...)
2: #include <string> // objet string
3: #include <vector> // objet vecteur
4: using namespace std;
5: int main(int argc, char** argv) {
6:  vector<string> v ;                    // Initialisation v est vide
7:  ifstream in("toto.txt");              // Ouverture pour lecture
8:  string a_line;
9:  while( getline(in,a_line) )           // Lecture ligne par ligne
10:    v.push_back(a_line);                 // Stockage ligne dans v
11: cout << "nombre d element du  vecteur" << v.size() << endl ;
12: return 0 ;
    }
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
    - Bonnes pratiques de programmation
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

Bonnes Pratiques de Programmation

- Séparation des fichiers en-tête et sources
- Rendre le compilateur verbeux
- De la bonne utilisation des macros
- Assertion dans le code
- Outils de compilation séparée (Qmake en TP)
- Conventions de codage (vues TP)

Séparation des fichiers sources et en-tête

- Fichier en-tête
  - .hpp (convention)
  - Déclarations
- Fichier source
  - .cpp (convention)
  - Définitions

Contenu d'un fichier en-tête (.hpp)

- .hpp, le standard autorise:

Named namespaces	`namespace N { /* ... */ }`
Type definitions	`struct Point { int x, y; };`
Template declarations	`template<class T> class Z;`
Template definitions	`template<class T> class V { /* ... */ };`
Function declarations	`extern int strlen(const char*);`
Inline function definitions	`inline char get(char* p) { return *p++; }`
Data declarations	`extern int a;`
Constant definitions	`const float pi = 3.141593;`
Enumerations	`enum Light { red, yellow, green };`
Name declarations	`class Matrix;`
Include directives	`#include <algorithm>`
Macro definitions	`#define VERSION 12`
Conditional compilation directives	`#ifdef __cplusplus`
Comments	`/* check for end of file */`

The C++ Programming Language. B.Stroustrup. 1997.

Contenu d'un fichier en-tête (.hpp)

- .hpp, le standard n'autorise PAS:

Ordinary function definitions	`char get(char* p) { return *p++; }`
Data definitions	`int a;`
Aggregate definitions	`short tbl[] = { 1, 2, 3 };`
Unnamed namespaces	`namespace { /* ... */ }`
Exported template definitions	`export template<class T> f(T t) { /* ... */ }`

The C++ Programming Language. B.Stroustrup. 1997.

Macro guards

Toto.hpp

```
#ifndef TOTO_CLASS_HPP
#define TOTO_CLASS_HPP


//CODE C++
// DECLARATIONS VIENNENT ICI




#endif
```

Objectifs:

- Faciliter le pre-processing
- Accélérer la compilation

- A partir de C++ 2011:

#pragma once

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Contenu fichier source (.cpp)

Règles de base :

- Ce qu'il n'y a pas dans les .hpp
- L'implémentation = la définition des fonctions et des variables à l'exception de celles définies dans les .hpp (e.g. *inline*)
- Inclusion des .hpp pour avoir accès à la déclaration des fonctions, classes,...

De la bonne utilisation des macros

- Ne pas Utiliser les macros
  - Pour définir des constantes

    → mot clé *const*
  - Pour définir des fonctions

    → mot clé *inline*
- Utiliser les macros pour des constantes de compilation
  - NDEBUG,DEBUG

Options de compilation

- Définir des constantes de compilation
  - g++ -DDEBUGING_MODE hello.cpp -o hello.exe
- Contrôler son code:
  - -Wall
  - -ansi -pedantic
  - -Wunused -Wextra -Wswitch-default
- Spécifier une version du langage
  - -std=c++98

Deboguer son code

- Définir des constantes de compilation
  - g++ -DDEBUGING_MODE hello.cpp -o hello.exe
- Dans le code :

```
#ifdef DEBUGING_MODE
cout <<   "  a=  " << a << endl;
#endif
```

- Pas de constante

➔ Pas de pénalité à l'éxecution

➔ Code éliminé par le pré-processeur

➔ Deux versions d'un programme debug & release

Les assertions

- Proviennent du C.
- Verification d'une condition à l'execution sinon arrêt
- Peuvent être désactivées avec NDEBUG

```
#include <cassert>
int main(int argc, char** argv) {
 int i = 100 ;
 assert( i!= 100 ) ;  // Condition fausse => FIN EXECUTION
}
```

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

- Cours 1
  - Généralités
  - Le processus de création d'un programme
  - Tour d'horizon du C++
  - La programmation impérative en C++
- Cours 2
- Cours 3
- Cours 4

La programmation impérative en C++

- Proche du C pour la syntaxe
  - Declaration des variables
  - Structure de contrôle
  - Boucles
- ! Sémantique parfois différente du C

→ Deux langages différents!

Créer des fonctions (1/2)

```
// Définition.
// Nommage args obligatoire si utilisation dans fct
float translate( float x, float y, float z )
{
  return x + y + z;
}
// Warning du CC si nommage mais non utilisation
float translate( float x, float y, float z )
{
  return x + y;
}
// Absence de nommage possible alors.
float translate( float x, float y, float )
{
  return x + y ;
}
```

Créer des fonctions (2/2)

```
// void  = RIEN
void fooBar(void) ; // eq. void fooBar()


// return n'est PAS optionnel
// quand le type de retour est différent
// de void
int factoriel( int n )
{
  return (n != 0) ? (factoriel(n-1)):(1) ;
}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Utilisation des fonctions STL (C++)

- Trouver dans la documentation le header où est déclaré la fonction

  → www.cppreference.com

- Inclure le *header* correspondant

```
#include <fstream>
```

- Par défaut, le CC et l'éditeur de lien cherche automatiquement dans la librairie standard et connaisse où elle est installée

Utilisation des fonctions de la librairie C

- Trouver dans la documentation le header où est déclaré la fonction

  → www.cppreference.com

- Inclure le *header* correspondant

```
#include <stdlib.h>                    // Forme mauvaise
#include <cstdlib>                     // Forme correcte
```

- Par défaut, le CC et l'éditeur de lien cherche automatiquement dans la librairie standard et connaisse où elle est installée

Type booléen

```
1: #include <iostream>
2: using namespace std;
3: int main(int argc, char** argv) {
4:  bool a = true ;
5:  bool b = false ;
6:  bool c = (a && b ) and (!b or a ) ;
7:  bool d = !b;
8:  bool e = a || b;
9:  cout << " a= " << a << "b = " << b << "c = "<< c
10:       << "d = " << d << " e = " << e << endl;
11: return 0 ;
12:}
```

Type booléen

```
1: #include <iostream>
2: using namespace std;
3: int main(int argc, char** argv) {
4:  bool a = true ;
5:  bool b = false ;
6:  bool c = (a && b ) and (!b or a ) ;
7:  bool d = !b;
8:  bool e = a || b;
9:  cout << " a= " << a << "b = " << b << "c = "<< c
10:       << "d = " << d << " e = " << e << endl;
11: return 0 ;
12: }
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Structure de contrôle if-else

```
1: #include <iostream>
2: using namespace std;
3: int main(int argc, char** argv) {
4:   int i ;
5:   cin >> i ;
6:   if ( i < 5 )  {
7:    cout << "strictement plus petit que 5" << endl ;
8:   } else {
9:     if ( i > 5 ) {
10:      cout << "strictement plus grand que 5" << endl ;
11:    } else {
12:     cout << "egal a 5" << endl ;
13:    }
14:  }
15:  return 0 ;
16: }
```

Boucle for

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
 for ( int i = 0 ; i < 128*argc ; i=i+1 )
 {




  // INSTRUCTIONS
 }

 return 0 ;

}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

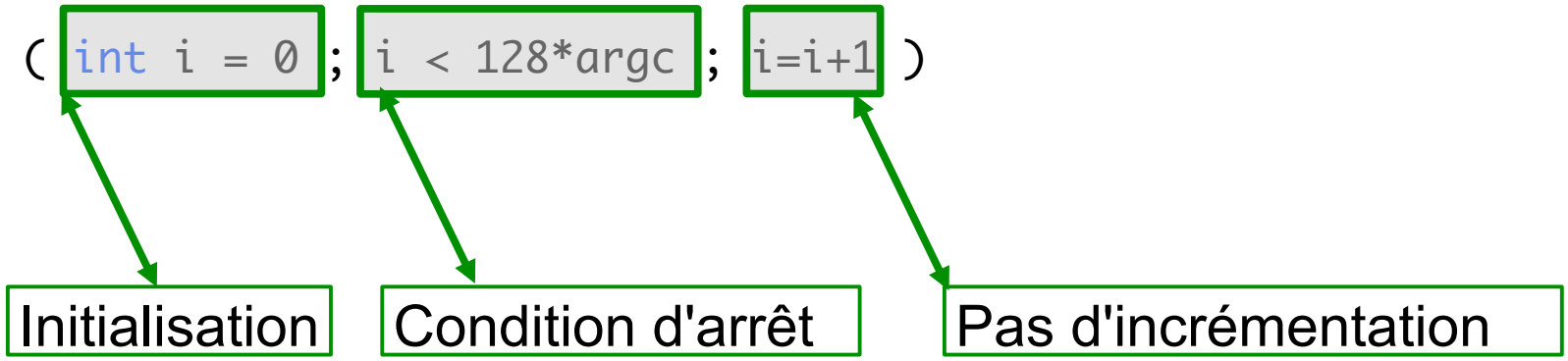
Boucle for

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
 for ( int i = 0 ; i < 128*argc ; i=i+1 )
 {

  // INSTRUCTIONS
 }

 return 0 ;

}
```



Boucle for

```
1: #include <iostream>
2: using namespace std;
3: int main(int argc, char** argv)
4: {
5:  for ( int i = 0 ; i < 128*argc ; i=i+2 )
6:  {
7:     if( i!= 26 )                                  // Devinez/Testez pq!
8:     {
9:       cout << " Value i= " << i
10:           << " char(i) = " << char(i)            // Conversion de type
11:           << endl ;
12:    }
    }
    return 0 ;
   }
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Boucle while

```
1:  #include <iostream>
2:  using namespace std;
3:  int main(int argc, char** argv)
4:  {
5:   int i = 0;
6:   while ( (i != 26) && (i < 128*argc) )
7:   {
8:     cout << " Value i= " << i
9:          << " char(i) = " << char(i)          // Conversion de type
10:         << endl ;
11:    i += 5;
12:  }
     return 0 ;
    }
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Mot-clés *break* et *continue*

```
1:  #include <iostream>
2:  using namespace std;
3:  int main(int argc, char** argv) {
4:   int i = 0;
5:   while ( true )
6:   {
7:     if( i % 2  ) { continue;  }
8:     cout << " Value i= " << i
9:          << " char(i) = " << char(i)          // Conversion de type
10:         << endl ;
11:    i += 5;
12:    if( i > 100 ) { break;}
     }
     return 0 ;
    }
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Introduction aux opérateurs

- Opérateurs classiques :

  +, -, *, \, =, %, &, <<, >>,

- Opérateur = appel « spécial » de fonction
- Règles de précédence
  - Simple pour expression arithmétique (*, \, %, +, -)
  - Complète (cf. annexe)
- Opérateurs d'incrémentation et décrémentation

Introduction aux opérateurs

```
1: #include <iostream>
2: using namespace std;
3: int main(int argc, char** argv)
4: {
5:  int i = 0;
6:  int j = 0 ;
7:  cout << ++i << endl ;              // pre-incrementation
8:  cout << j++ << endl ;              // post-incrementation
9:
10: cout << --i << endl ;              // pre-decrementation
11: cout << j-- << endl ;              // post-decrementation
12:
13: return 0 ;
14:}
```

INSTITUT d'OPTIQUE GRADUATE SCHOOL

Types de données

- Types primitifs (fondamentaux)
  - Très proches du langage C
  - Types signés et non-signés (*unsigned*) != Java
  - Compris intrinsèquement par le compilateur

- Types abstrait
  - Classes créés par le programmeur
  - Appris à la « volée » par le compilateur

Types primitifs

- Norme C++
  - Ne spécifie pas le nombre de bits

  → `sizeof()`
  - Valeurs minimales et maximales que le type
    - `#include <climits>`
    - `#include <cfloat>`
- 4 types de base
  - char, int, float, double

Types primitifs

- 4 types de base
  - char, int, float, double
- bool
  - Valeurs : true or false
  - Conversion du compilateur vers int

  → mauvais usage (historique)

Modificateurs de types primitifs

- 4 modificateurs
  - long, short,unsigned, signed
- Hiérarchie
  - sur les entiers : short int, int, int, long int
  - sur les flottants : float, double, long double
- signed
  - Implicite sur tous les types sauf char

Exercice sur les typres primitifs

- Afficher la taille de tous les types
  - en utilisant l'opérateur sizeof
  - avec et sans modificateurs