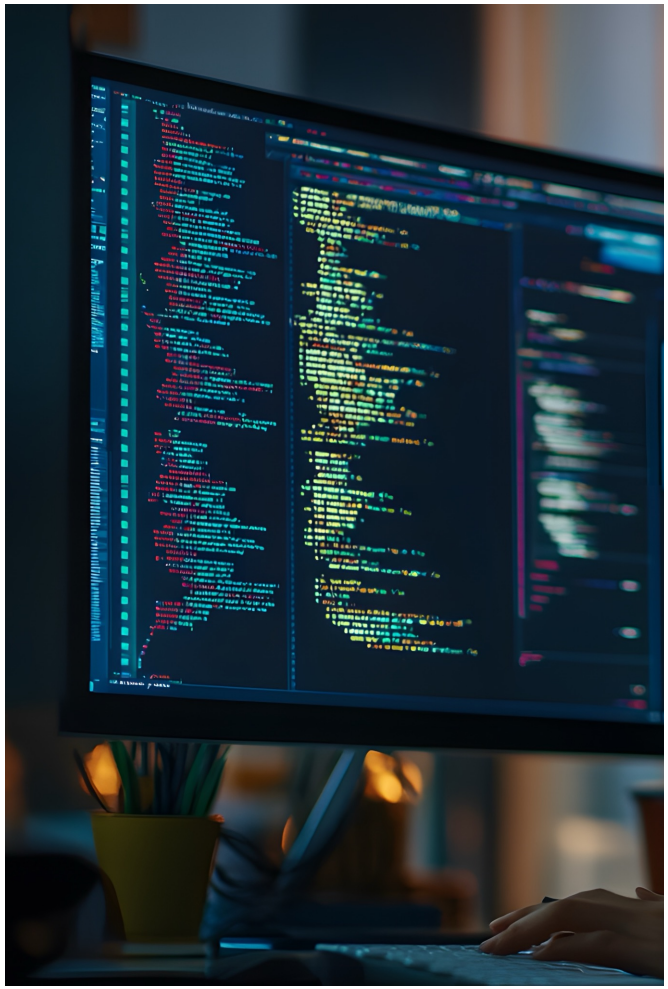
Programmation Orientée Projet
COM-112(a)

Semaine 6

Rafael Pires
rafael.pires@epfl.ch

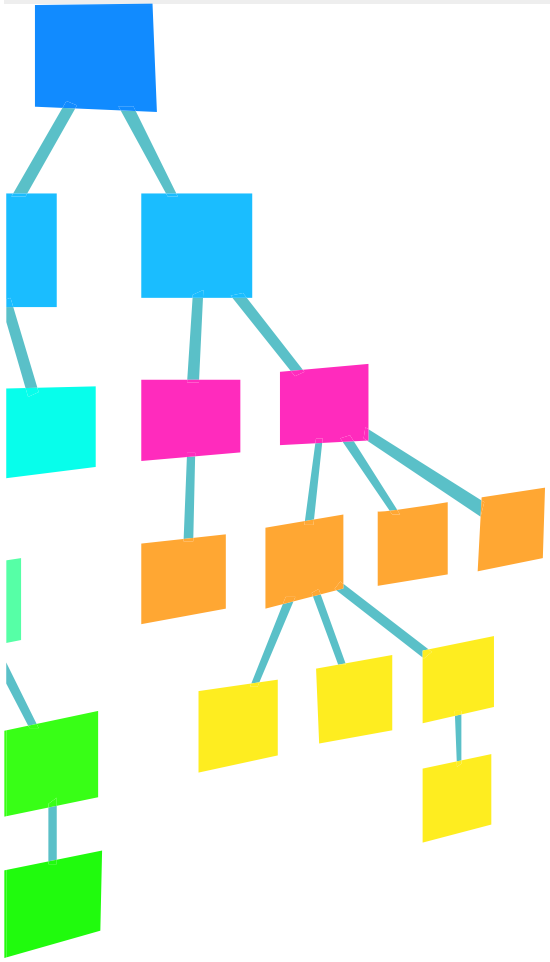
Lausanne, 27.03.2026

PoP 06

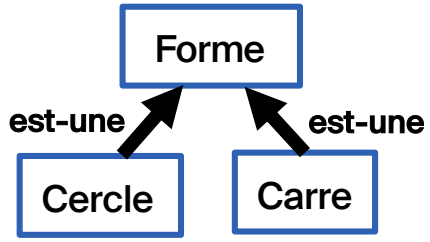


- **MOOC :**
  - Polymorphisme
- **Projet :**
  - Gtkmm4
  - Programmation par événement

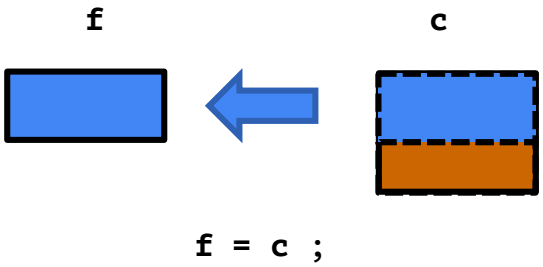
Héritage : Rappel - downslicing



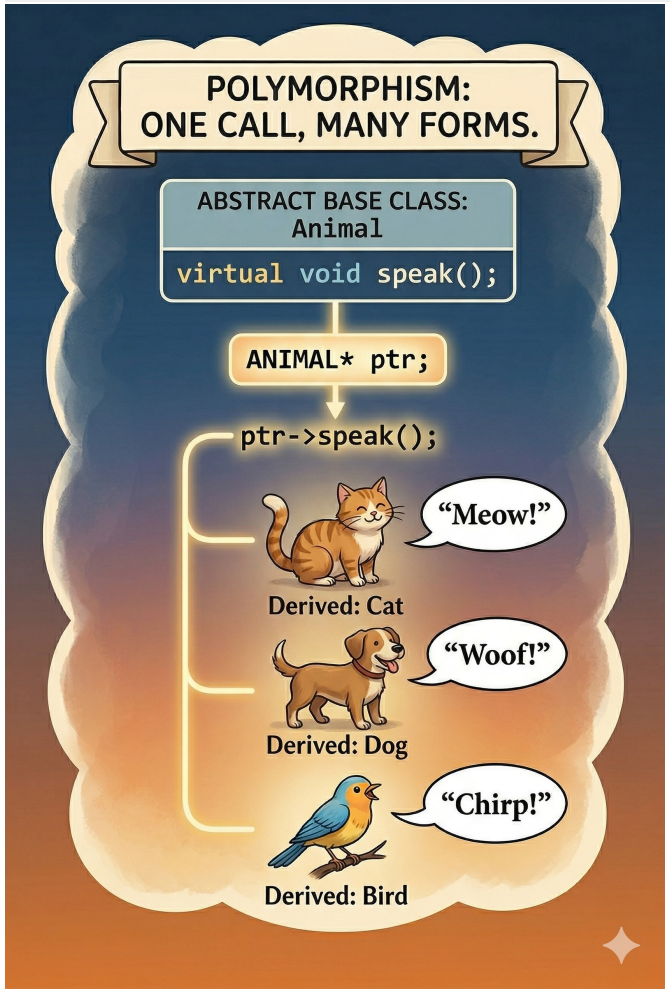
- L'affectation d'une instance d'une classe dérivée à une instance d'une classe parente se traduit par la perte des attributs de la classe dérivée (object downslicing).



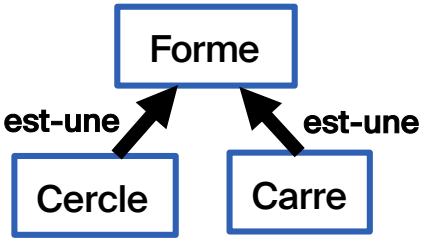

```
Forme f;
Cercle c;
f = c;
```



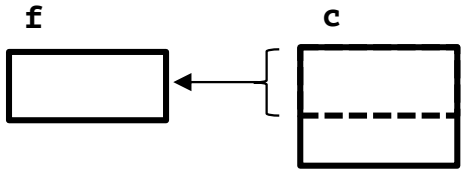
Polymorphisme : pointeur et résolution dynamique



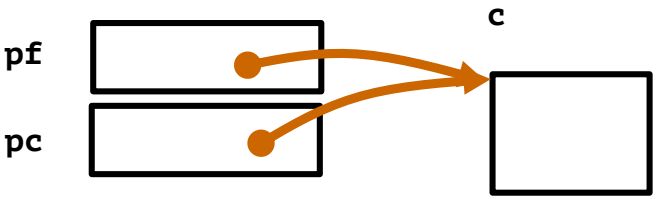
- L'affectation d'un pointeur d'une instance d'une classe dérivée à un pointeur de la classe parente **conserve** le lien avec la variable de la classe dérivée (l'adresse reste la même)



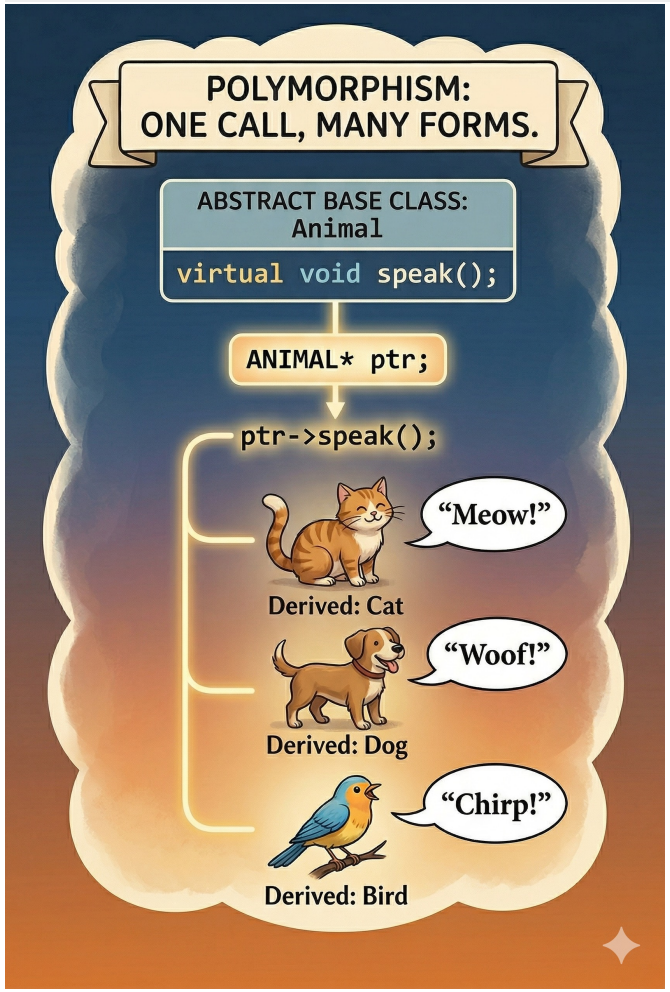

```
Forme f;
Cercle c;
f = c;
```




```
Forme* pf(nullptr);
Cercle* pc(&c);
pf = pc;
```



Polymorphisme : résolution dynamique

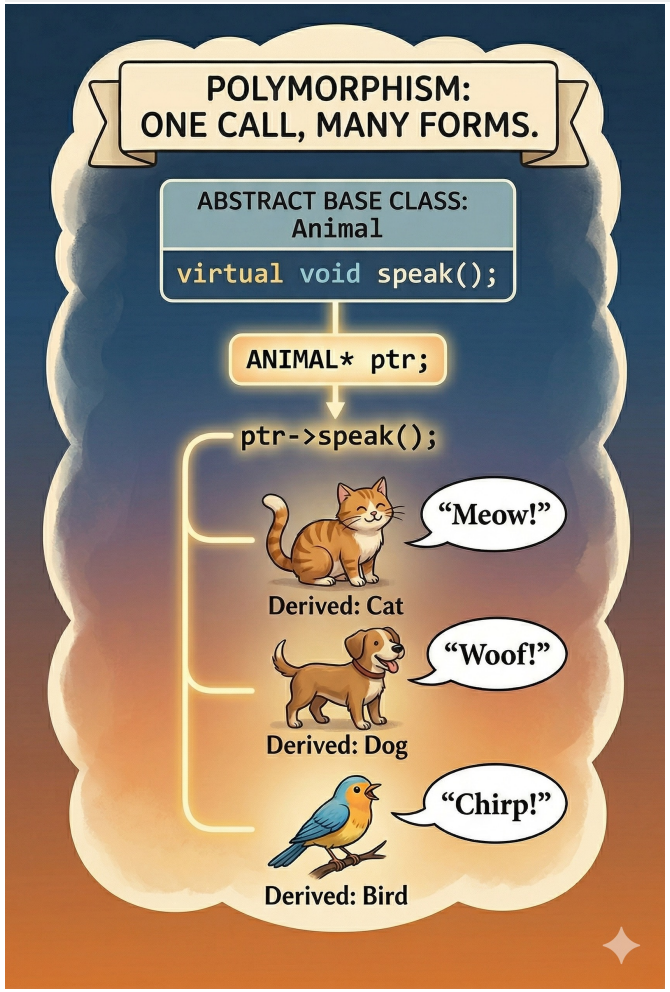



```
char c;
std::cin >> c;

Forme *pf(nullptr);
if (c == 'c') pf = new Cercle;
else pf = new Forme;

pf->description();
```

Si on se place dans un scénario de polymorphisme, il est impossible pour le compilateur de savoir quelle méthode `description()` doit être appelée au moment de la compilation

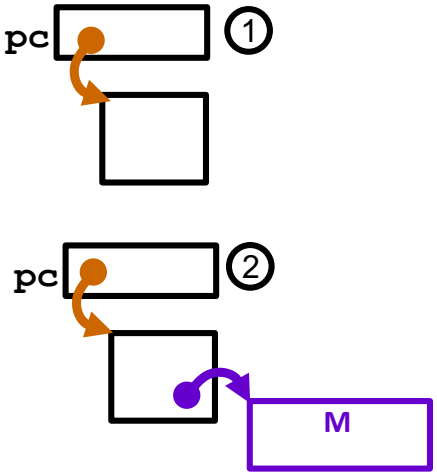
- Par défaut c'est la résolution **statique** qui prime (basée sur le **type**)
- La résolution **dynamique** est obtenue pour les methodes `virtual`
- Il n'est pas obligatoire mais recommandé d'ajouter `override` : demande au compilateur de **vérifier** qu'il y a bien une méthode `virtual` de même prototype dans une classe parente

Polymorphisme : destructeur `virtual` ?

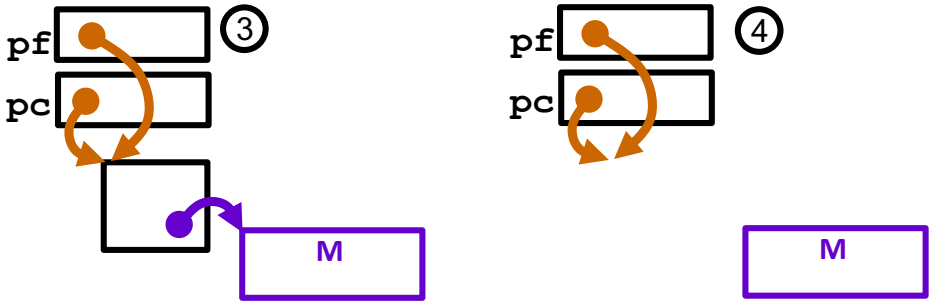



```
① Cercle* pc(new Cercle);
② pc->alloc();
③ Forme* pf(pc);
④ delete pf;
```

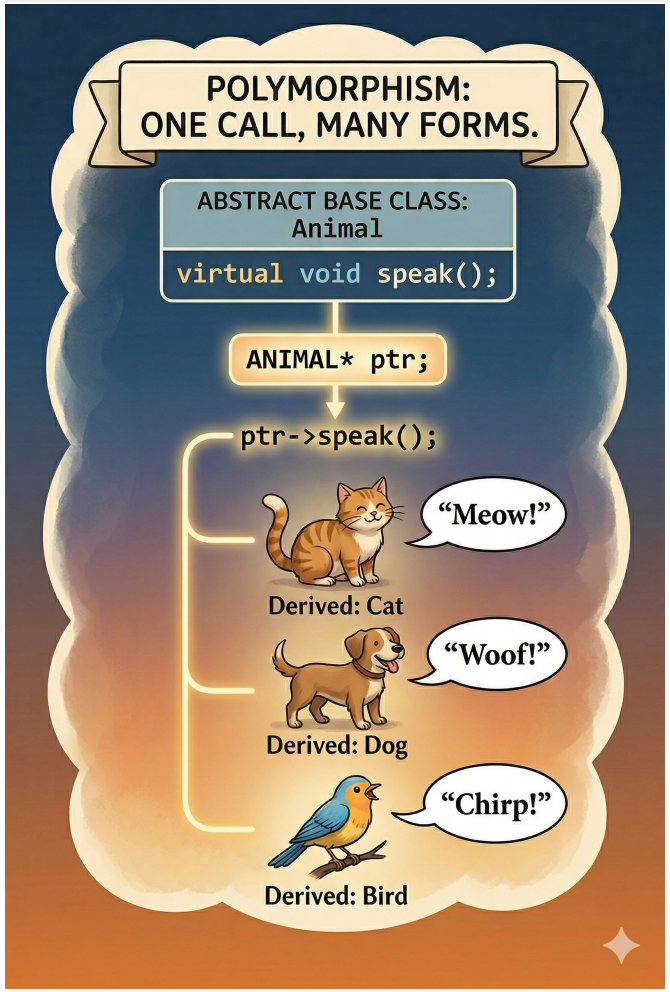
supposons que la méthode `alloc()` réalise une allocation dynamique **d'un bloc de mémoire M** dont l'adresse est mémorisée dans un attribut.



Si le destructeur de Forme n'est **PAS** `virtual` alors le destructeur de Cercle n'est PAS appelé et donc **le bloc M** n'est pas libéré : fuite de mémoire.



Polymorphisme : Quiz 1




```cpp
class Forme {
public:
  void description() const;
};

void Forme::description() const {
  cout << "Ceci est une forme !" << endl;
}

class Cercle : public Forme {
public:
  void description() const;
};

void Cercle::description() const {
  cout << "Ceci est un cercle !" << endl;
}

void affichageDesc(Forme& f) {
  f.description();
}
```

```cpp
int main() {
  Forme f;
  Cercle c;
  affichageDesc(f);
  affichageDesc(c);
  return 0;
}
```

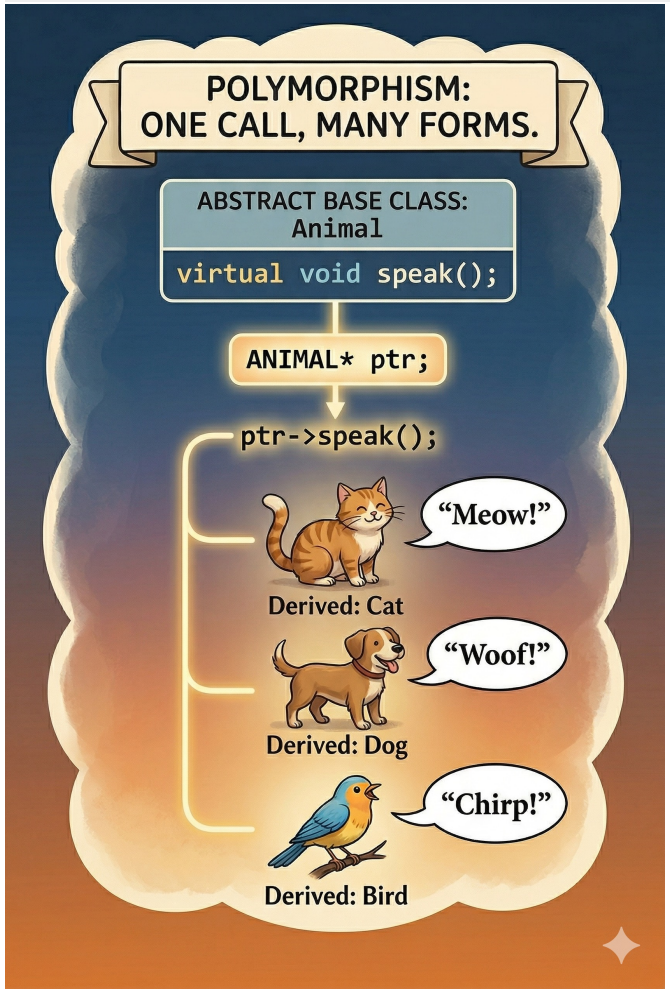
Ce code compile ; il affiche 2 lignes :

A: **Ceci est une forme !**
   Ceci est un cercle !

B: **Ceci est une forme !**
   Ceci est une forme !

C: **Ceci est un cercle !**
   Ceci est un cercle !

Polymorphisme : Quiz 2




```cpp
class Forme {
public:
  virtual void description() const;
};

void Forme::description() const {
  cout << "Ceci est une forme !" << endl;
}

class Cercle : public Forme {
public:
  void description() const;
};

void Cercle::description() const {
  cout << "Ceci est un cercle !" << endl;
}

void affichageDesc(Forme& f) {
  f.description();
}
```

```cpp
int main() {
  Forme f;
  Cercle c;
  affichageDesc(f);
  affichageDesc(c);
  return 0;
}
```

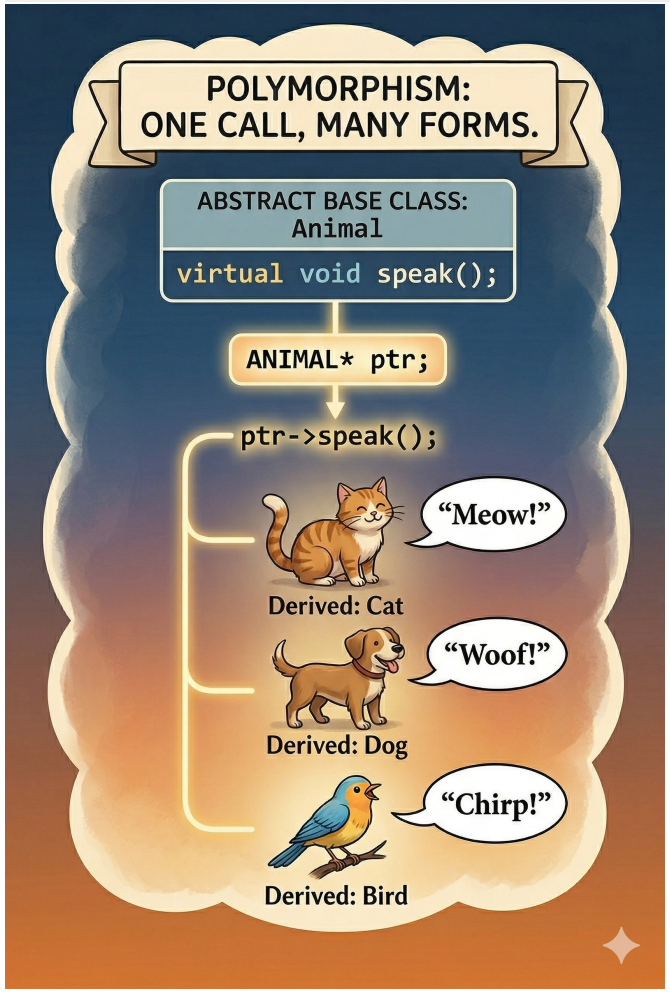
Ce code compile ; il affiche 2 lignes :

A: **Ceci est une forme !**
   Ceci est un cercle !

B: **Ceci est une forme !**
   Ceci est une forme !

C: **Ceci est un cercle !**
   Ceci est un cercle !

Polymorphisme : Quiz 3




```cpp
class Forme {
public:
  virtual void description() const;
};

void Forme::description() const {
  cout << "Ceci est une forme !" << endl;
}

class Cercle : public Forme {
public:
  void description() const;
};

void Cercle::description() const {
  cout << "Ceci est un cercle !" << endl;
}

void affichageDesc(Forme f) {
  f.description();
}
```

```cpp
int main() {
  Forme f;
  Cercle c;
  affichageDesc(f);
  affichageDesc(c);
  return 0;
}
```

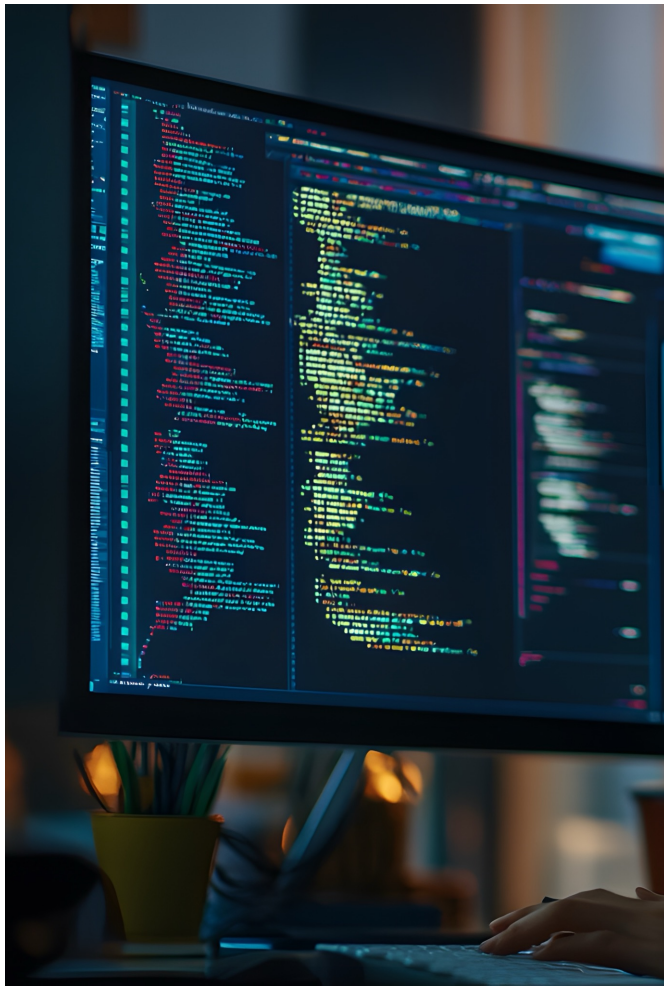
Ce code compile ; il affiche 2 lignes :

A: **Ceci est une forme !**
Ceci est un cercle !

B: **Ceci est une forme !**
Ceci est une forme !

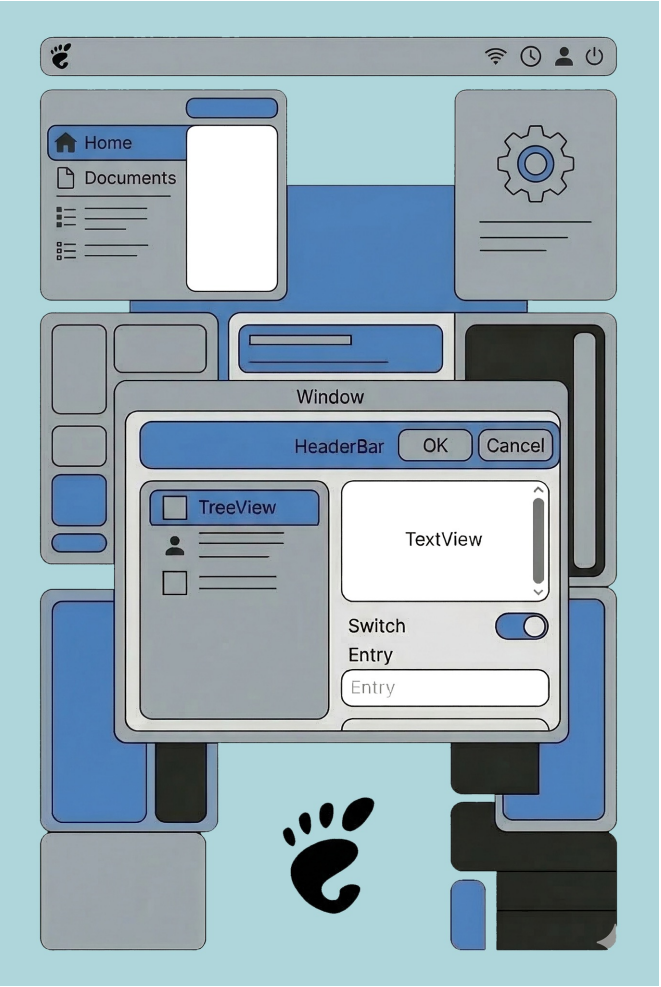
C: **Ceci est un cercle !**
Ceci est un cercle !

PoP 06



- **MOOC :**
  - Polymorphisme
- **Projet :**
  - Gtkmm4
  - Programmation par événement

Architecture MVC et GTKmm4



projet

À partir du rendu2

gui
Gestion du dialogue
Préparation affichage

CONTROLE

MODELE
Modélisation de la simulation indépendamment du choix de la bibliothèque d'affichage graphique.

Un module générique indépendant **tools** gère des formes élémentaires, point, vecteur, cercle ; ce module est complété pour dessiner les formes à partir du rendu2

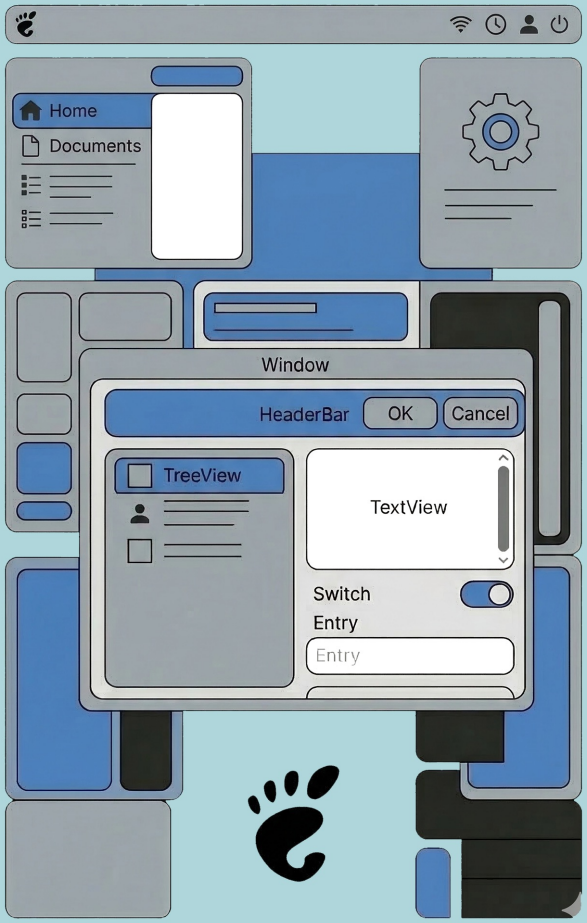
À partir du rendu2

À partir du rendu2

VISUALISATION
Un module **graphic** rassemble les dépendances GTKmm pour dessiner les entités définies dans le module **tools** : point, segment, cercle, style vide ou plein, avec une table prédéfinie des couleurs

GTKmm

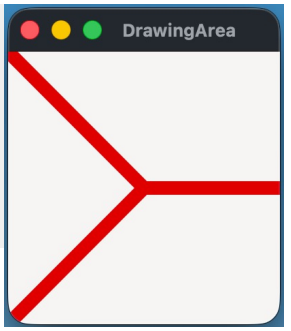
Architecture MVC et GTKmm4 : Problème



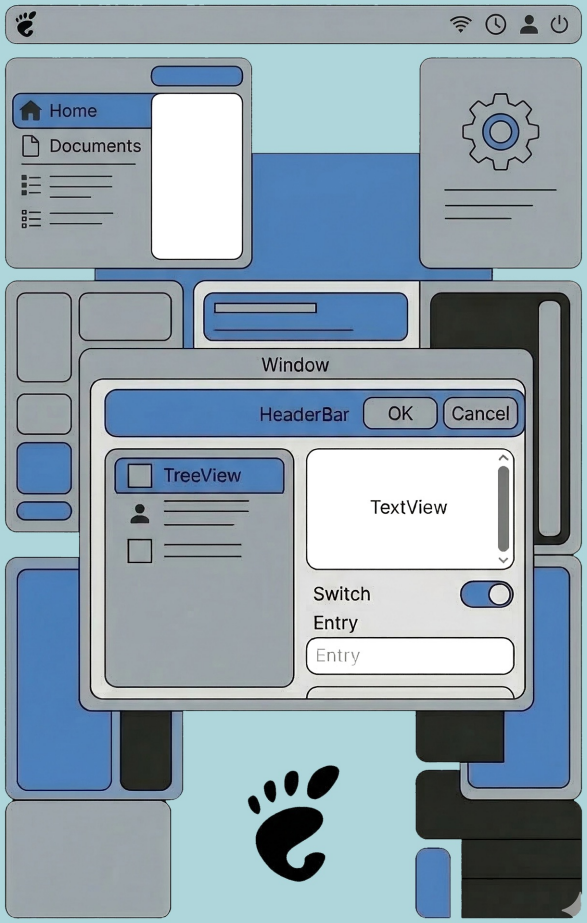
Tous les appels définissant les attributs du dessin et effectuant le tracé ont besoin du pointeur Cairo::Contex

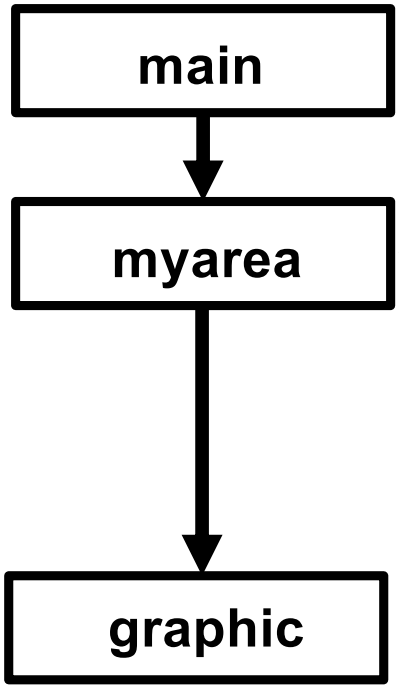
```
#ifndef GTKMM_EXAMPLE_MYAREA_H
#define GTKMM_EXAMPLE_MYAREA_H


#include <gtkmm/drawingarea.h>


class MyArea : public Gtk::DrawingArea {
public:
  MyArea();
  virtual ~MyArea();


protected:
  //Override default signal handler:
  void on_draw(const
      Cairo::RefPtr<Cairo::Context>& cr,
      int width, int height);
};
#endif // GTKMM_EXAMPLE_MYAREA_H
```

```
#include "myarea.h"
#include <cairomm/context.h>

MyArea::MyArea(){}
MyArea::~MyArea(){}

void MyArea::on_draw(const
                  Cairo::RefPtr<Cairo::Context>& cr,
                  int width, int height) {
  // coordinates for the center of the GTKmm window
  int xc, yc;
  xc = width / 2;
  yc = height / 2;

  cr->set_line_width(10.0);
  cr->set_source_rgb(0.8, 0.0, 0.0);
  cr->move_to(0, 0);
  cr->line_to(xc, yc);
  cr->line_to(0, height);
  cr->move_to(xc, yc);
  cr->line_to(width, yc);
  cr->stroke();
}
```



Architecture MVC et GTKmm4 : Solution

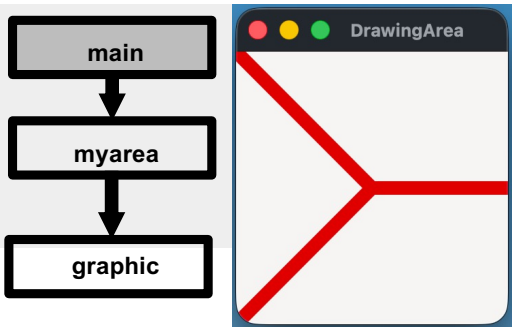


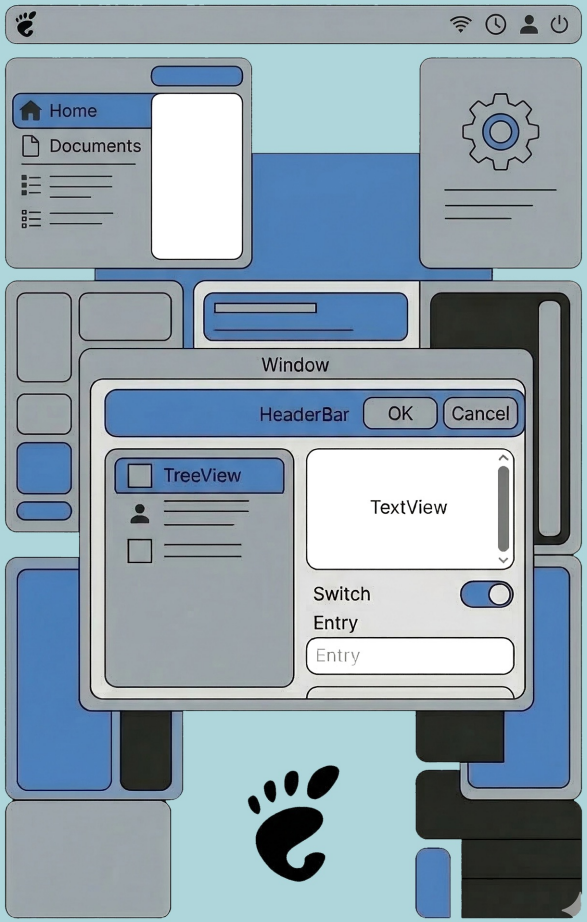


Module équivalent au module **gui** du projet:
- inclut l'interface complète du module graphic `graphic_gui.h`
- utilise la fonction qui initialise le pointeur sur `Cairo::Context cr`
- appelle ensuite une fonction de dessin de **graphic** sans avoir besoin de passer `Cairo::Context cr` en paramètre

Module **graphic** :
- offre une interface *complète* `graphic_gui.h`
- offre une seconde interface *partielle* `graphic.h` (projet: pour **tools**)
- Met à jour le pointeur `ptcr` sur `Cairo::Context cr`
- Utilise le pointeur `ptcr` pour toutes les commandes de dessin

Architecture MVC et GTKmm4 : Solution

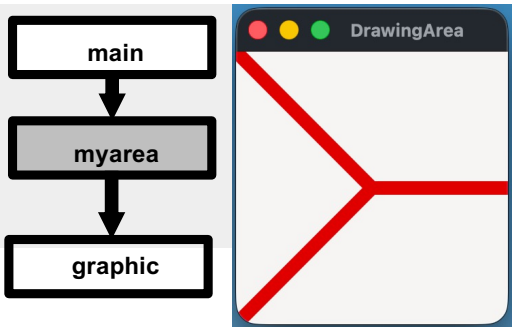


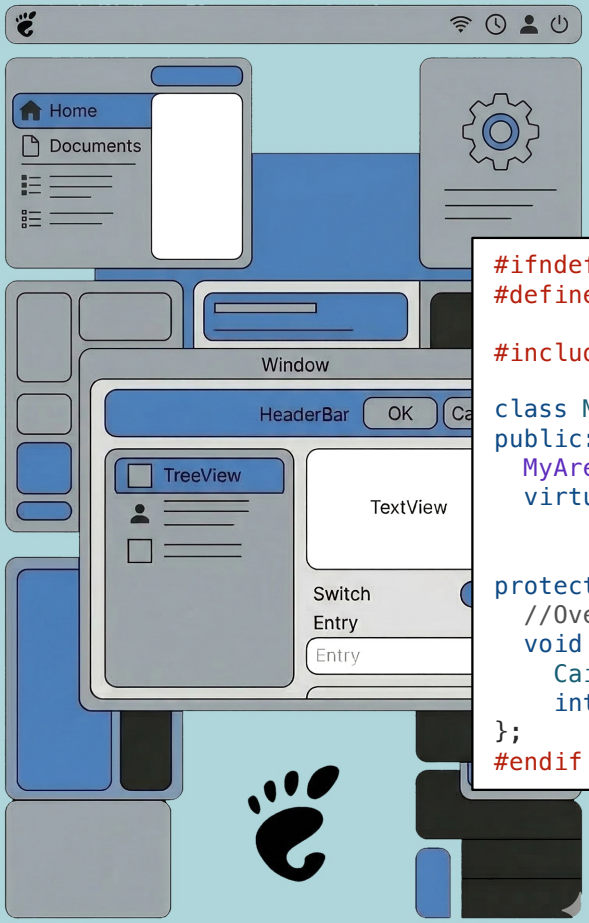


main.cc

```
#include "myarea.h"
#include <gtkmm/application.h>
#include <gtkmm/window.h>

class ExampleWindow : public Gtk::Window {
public:
  ExampleWindow();

protected:
  MyArea m_area;
};

ExampleWindow::ExampleWindow() {
  set_title("DrawingArea");
  set_child(m_area);
}

int main(int argc, char** argv) {
  auto app = Gtk::Application::create();
  return app->make_window_and_run<ExampleWindow>(argc, argv);
}
```

Architecture MVC et GTKmm4 : Solution





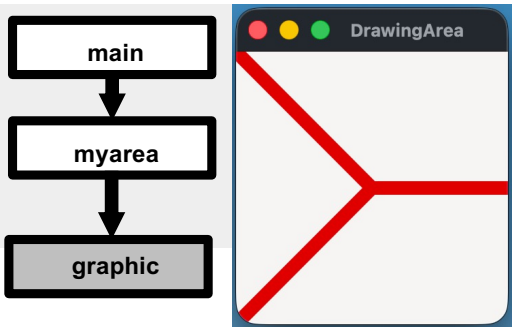
myarea.h

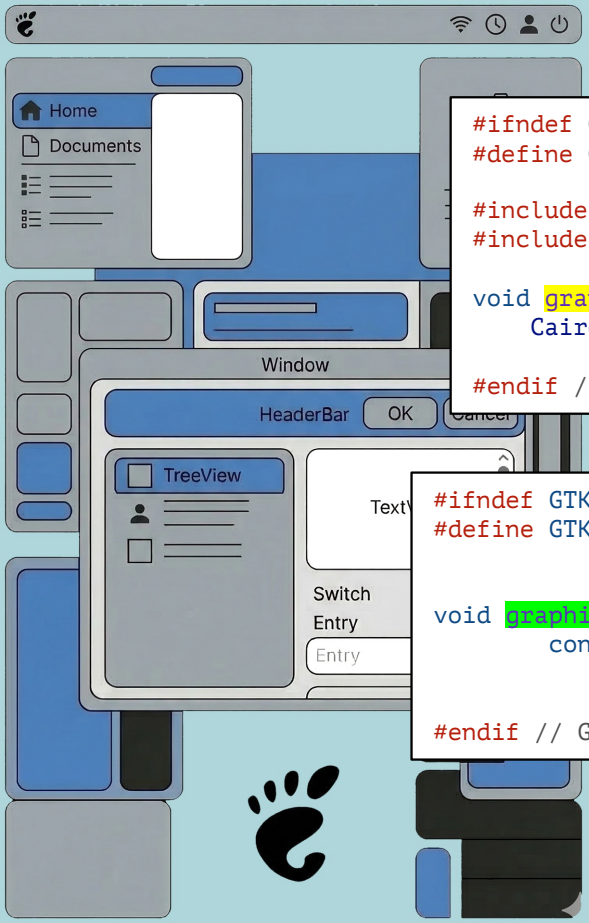
```
#ifndef GTKMM_EXAMPLE_MYAREA_H
#define GTKMM_EXAMPLE_MYAREA_H

#include <gtkmm/drawingarea.h>

class MyArea : public Gtk::DrawingArea {
public:
  MyArea();
  virtual ~MyArea();


protected:
  //Override default signal handler:
  void on_draw(const
    Cairo::RefPtr<Cairo::Context>& cr,
    int width, int height);
};
#endif // GTKMM_EXAMPLE_MYAREA_H
```

myarea.cc

```
#include "myarea.h"
#include "graphic_gui.h"
#include <cairomm/context.h>


MyArea::MyArea(){
  set_draw_func(sigc::mem_fun(*this, &MyArea::on_draw));
}
MyArea::~MyArea(){}


void MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
int width, int height) {
  graphic_set_context(cr);

  // coordinates for the center of the GTKmm window
  int xc, yc;
  xc = width / 2;
  yc = height / 2;

  graphic_draw_shape(width, height,xc,yc);
}
```

Architecture MVC et GTKmm4 : Solution





graphic_gui.h

```
#ifndef GTKMM_EXAMPLE_GRAPHIC_GUI_H
#define GTKMM_EXAMPLE_GRAPHIC_GUI_H

#include <gtkmm/drawingarea.h>
#include "graphic.h"

void graphic_set_context(const
    Cairo::RefPtr<Cairo::Context>& cr);

#endif // GTKMM_EXAMPLE_GRAPHIC_GUI_H
```

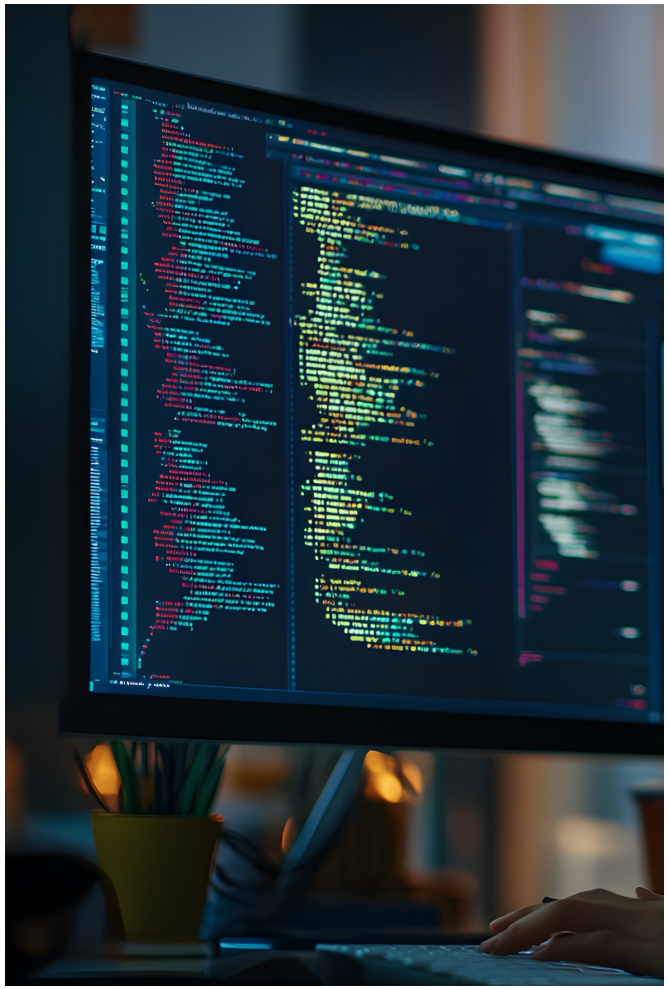
graphic.h

```
#ifndef GTKMM_EXAMPLE_GRAPHIC_H
#define GTKMM_EXAMPLE_GRAPHIC_H


void graphic_draw_shape(const int width,
        const int height, int xc, int yc);


#endif // GTKMM_EXAMPLE_GRAPHIC_H
```

graphic.cc

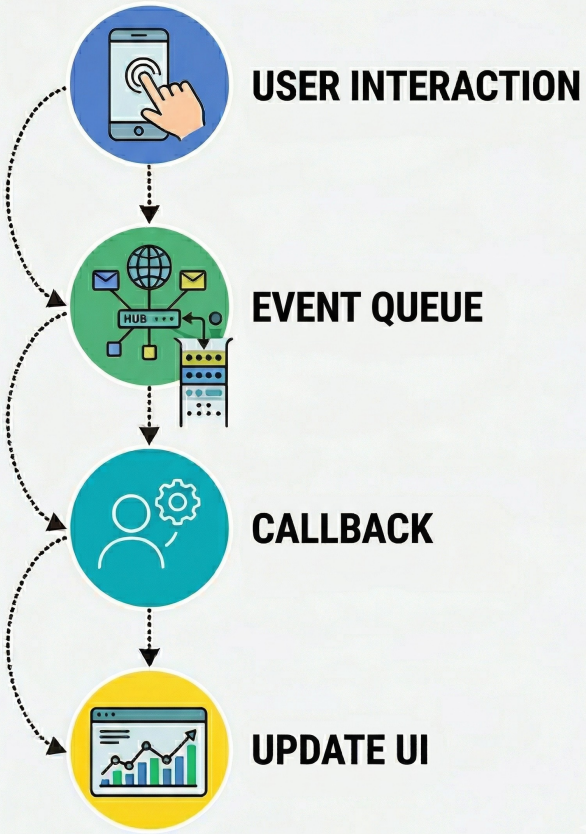
```
#include "graphic_gui.h"

static const Cairo::RefPtr<Cairo::Context>* ptcr(nullptr);

void graphic_set_context(const Cairo::RefPtr<Cairo::Context>& cr) {
  ptcr = &cr;
}

void graphic_draw_shape(const int width, const int height,
                        int xc, int yc) {
  (*ptcr)->set_line_width(10.0);

  // draw red lines out from the center of the window
  (*ptcr)->set_source_rgb(0.8, 0.0, 0.0);
  (*ptcr)->move_to(0, 0);
  (*ptcr)->line_to(xc, yc);
  (*ptcr)->line_to(0, height);
  (*ptcr)->move_to(xc, yc);
  (*ptcr)->line_to(width, yc);
  (*ptcr)->stroke();
}
```

PoP 06



- **MOOC :**
  - Polymorphisme
- **Projet :**
  - Gtkmm4
  - **Programmation par événement**

Lecture non-bloquante

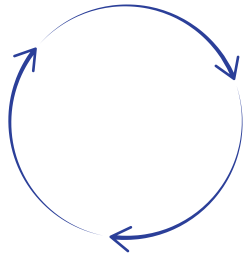


Entrée conversationnelle
= lecture bloquante

Tant qu'on n'a pas validé ce qui est tapé au clavier avec **Enter**, le *buffer* d'entrée est vide :

- il n'y a rien à « lire » pour le programme
- Le programme attend...

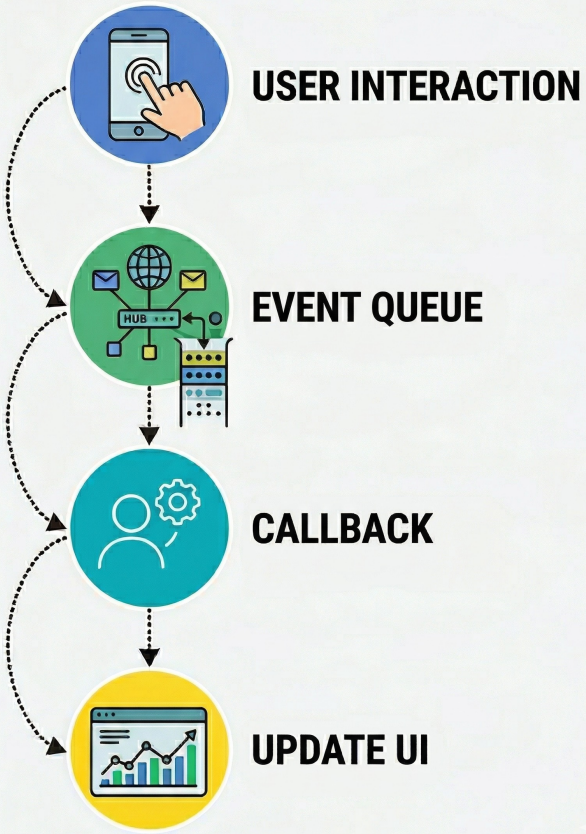
Programmation par événements
= lecture non-bloquante

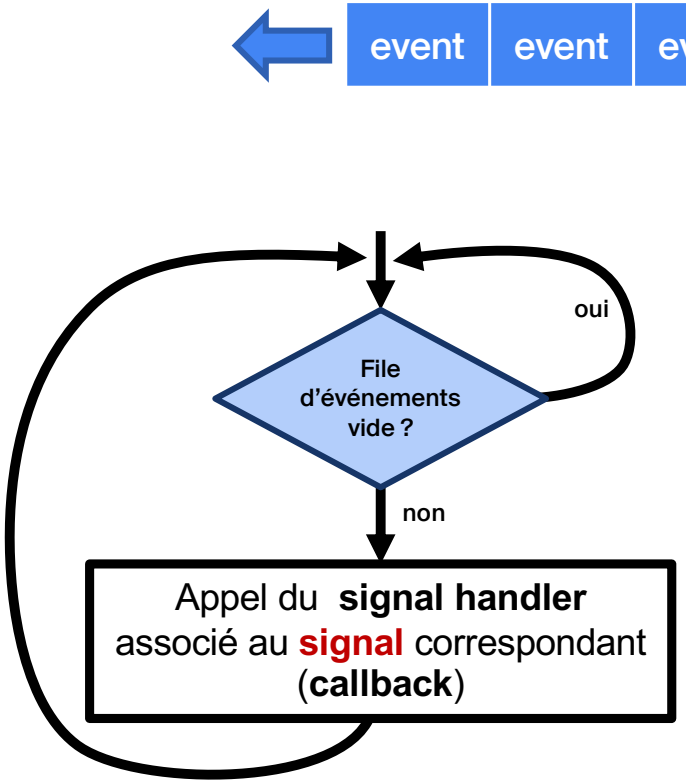


La méthode `run` gère une boucle infinie de traitement des *événements*.

```
int main(…) {
    auto app = Gtk::Application::create();
    return app->make_window_and_run<…>(…);
}
```

Lecture non-bloquante

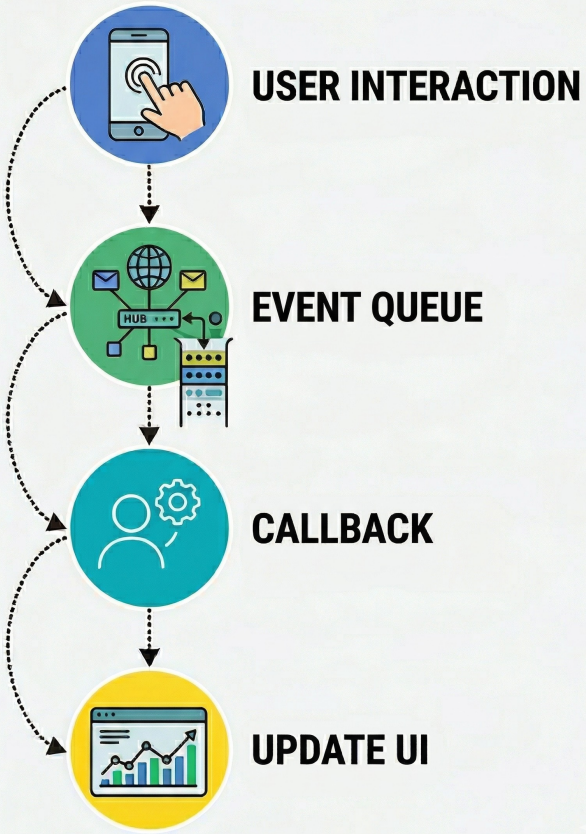




Ajout d'events dans la file d'attente selon les actions de la personne qui utilise le programme

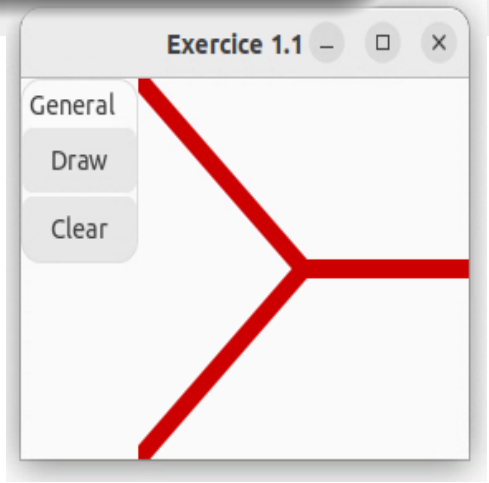
Ex : appuyer sur un `Button` produit `signal_clicked`

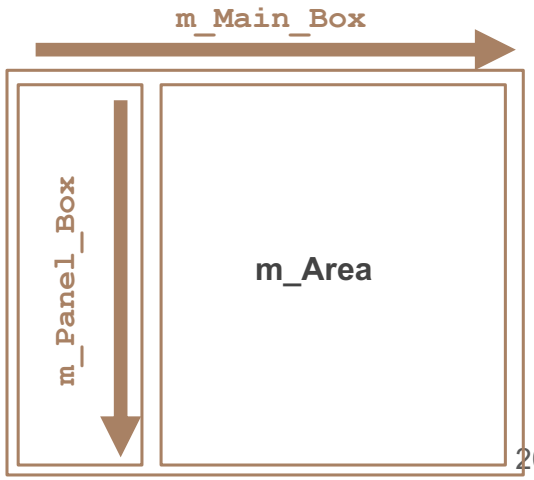
GTKmm4 : Buttons et événements



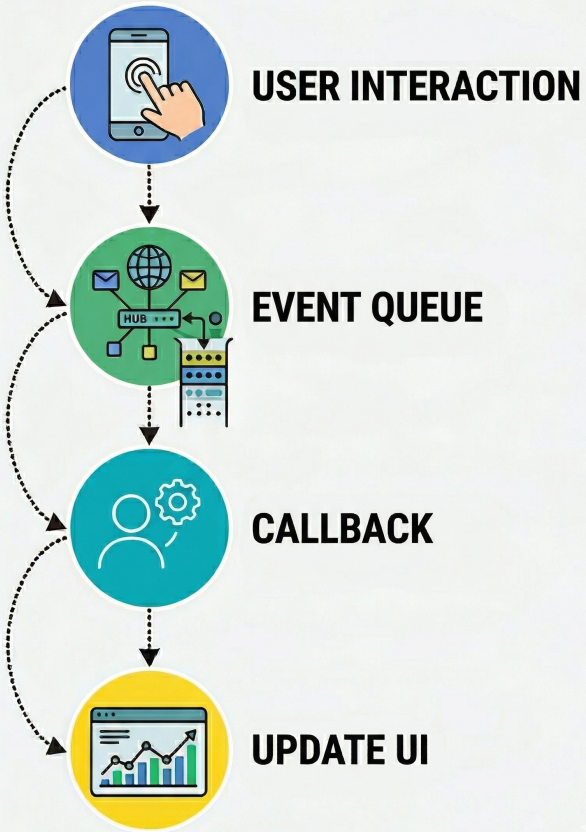
myevent.h

```
class MyEvent : public Gtk::Window
{
public:
  MyEvent();

private:
  // GUI layout
  Gtk::Box m_Main_Box;
  Gtk::Box m_Panel_Box;
  Gtk::Box m_Buttons_Box;
  Gtk::Frame m_Panel_Frame;
  Gtk::Button m_Button_Draw;
  Gtk::Button m_Button_Clear;
  Gtk::DrawingArea m_Area;

  bool draw ; // current drawing state

  //Button Signal handlers:
  void on_button_clicked_clear();
  void on_button_clicked_draw();

  // DrawingArea signal handler:
  void on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
               int width, int height);
};
```



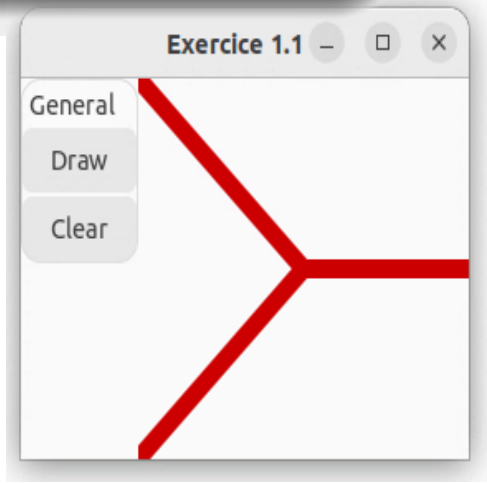


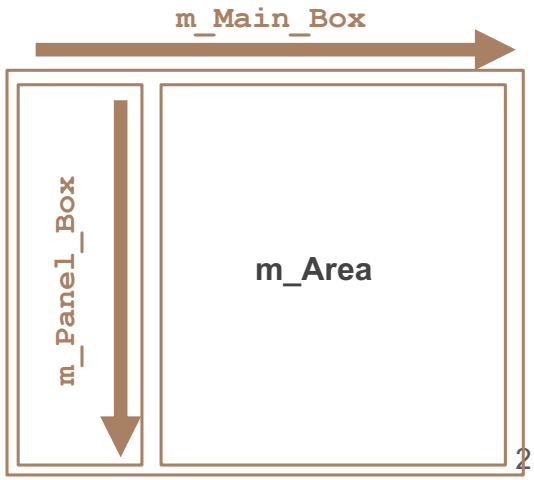
GTKmm4 : Buttons et événements



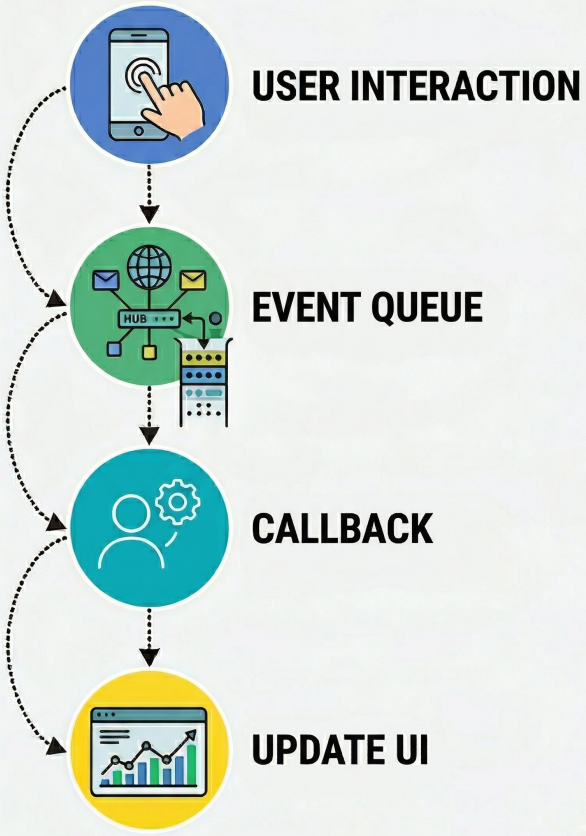
myevent.cc

```
MyEvent::MyEvent():
  m_Main_Box(Gtk::Orientation::HORIZONTAL, 0),
  m_Panel_Box(Gtk::Orientation::VERTICAL, 2),
  m_Buttons_Box(Gtk::Orientation::VERTICAL, 2),
  m_Panel_Frame("General"),
  m_Button_Draw("Draw"),
  m_Button_Clear("Clear"),
  draw(true) {

  // init layout
  set_title("Exercice 1.1");
  set_child(m_Main_Box);

  m_Main_Box.append(m_Panel_Box);
  m_Main_Box.append(m_Area);
  m_Panel_Box.append(m_Panel_Frame);
  m_Panel_Frame.set_child(m_Buttons_Box);
  m_Buttons_Box.append(m_Button_Draw);
  m_Buttons_Box.append(m_Button_Clear);

  // init buttons signal handlers
  m_Button_Clear.signal_clicked().connect(
  sigc::mem_fun(*this, &MyEvent::on_button_clicked_clear));
  m_Button_Draw.signal_clicked().connect(
  sigc::mem_fun(*this, &MyEvent::on_button_clicked_draw));

  // init drawing sub-window parameters
  m_Area.set_content_width(area_side);
  m_Area.set_content_height(area_side);
  m_Area.set_expand();
  m_Area.set_draw_func(sigc::mem_fun(*this, &MyEvent::on_draw));
}
```





GTKmm4 : Buttons et événements

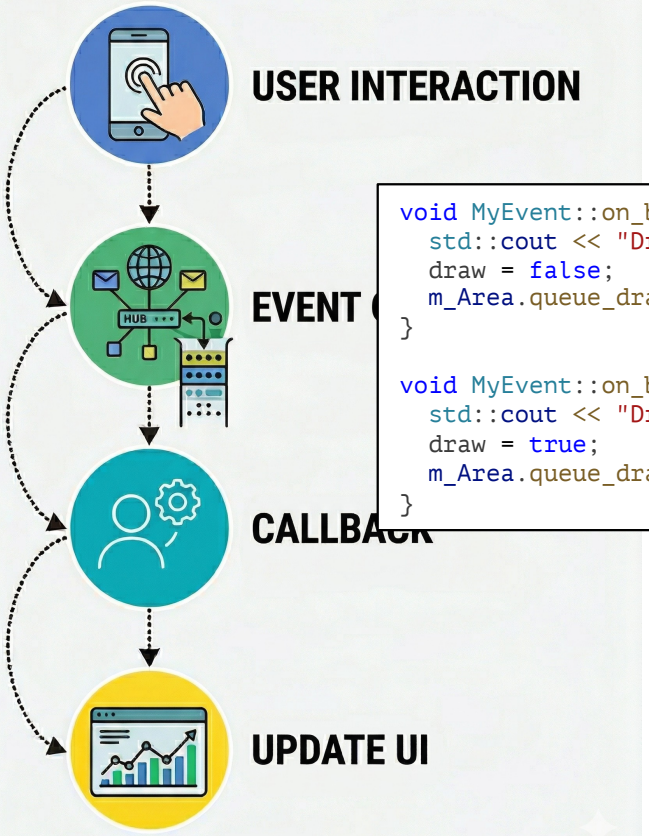


clic! Clear ⇨ Appel du signal handler : `on_button_clicked_clear()` ⇨ 1) draw passe à **false** 2) `queue_draw()`

clic! Draw ⇨ Appel du signal handler : `on_button_clicked_draw()` ⇨ 1) draw passe à ***true*** 2) `queue_draw()`

La méthode `queue_draw()` produit un **event** qui va causer l'appel de `on_draw()`

GTKmm4 : Buttons et événements



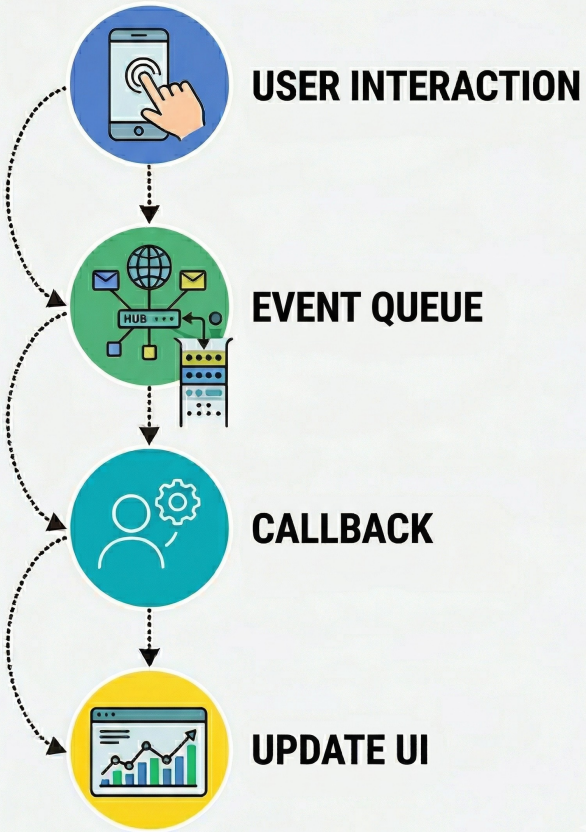
myevent.cc

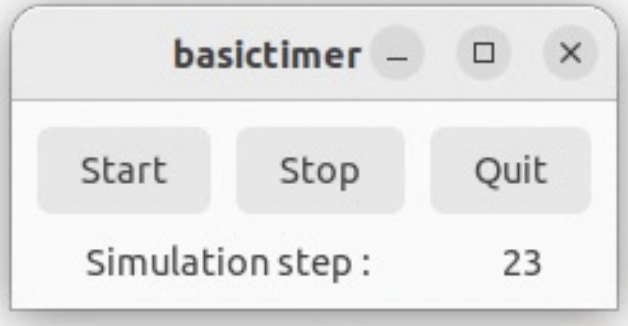
```
void MyEvent::on_button_clicked_clear() {
  std::cout << "Drawing cancelled" << std::endl;
  draw = false;
  m_Area.queue_draw();
}

void MyEvent::on_button_clicked_draw() {
  std::cout << "Drawing activated" << std::endl;
  draw = true;
  m_Area.queue_draw();
}
```

myevent.cc

```
void MyEvent::on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
                      int width, int height) {
  if(draw) {
    // coordinates for the center of the window
    int xc, yc;
    xc = width / 2;
    yc = height / 2;

    cr->set_line_width(10.0);

    // draw red lines out from the center of the window
    cr->set_source_rgb(0.8, 0.0, 0.0);
    cr->move_to(0, 0);
    cr->line_to(xc, yc);
    cr->line_to(0, height);
    cr->move_to(xc, yc);
    cr->line_to(width, yc);
    cr->stroke();
  } else {
    std::cout << "Nothing to draw !" << std::endl;
  }
}
```

GTKmm4 : Timer

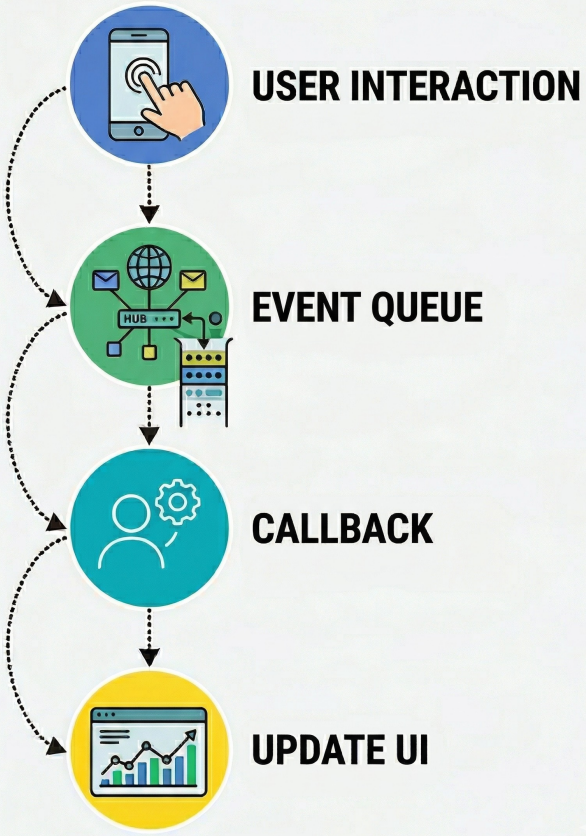




- 2 boutons gérent la demande d'activation et d'arrêt d'un timer à l'aide de 2 attributs d'état `timer_added` et `disconnect`
- l'attribut `timeout_value` mémorise la durée séparant 2 events du timer (exprimée en ms)
- le signal handler du timer qui est activé est :

```
bool on_timeout();
```

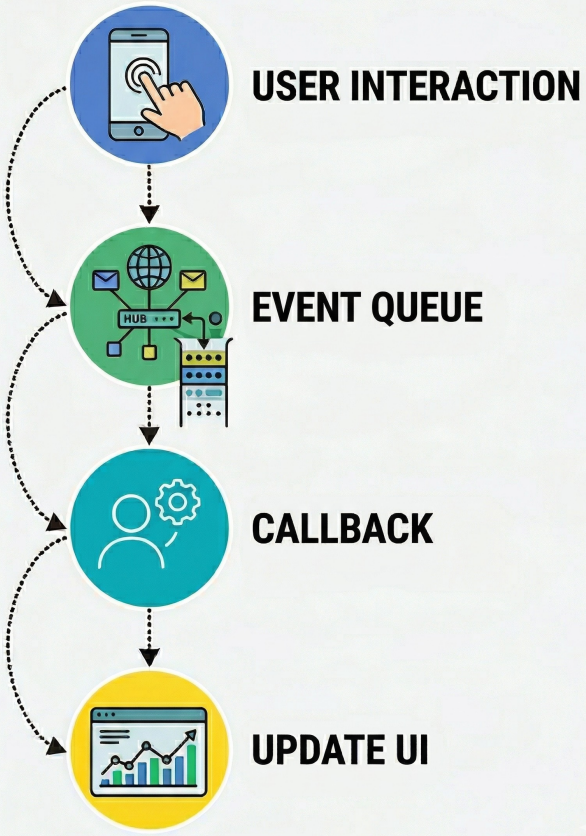
GTKmm4 : Timer



basictimer.cc

```
void BasicTimer::on_button_add_timer() {
  if(not timer_added) {
    sigc::slot<bool()> my_slot = sigc::bind(sigc::mem_fun(*this,
                                            &BasicTimer::on_timeout));
    auto conn = Glib::signal_timeout().connect(my_slot,timeout_value);
    timer_added = true;
    std::cout << "Timer added" << std::endl;
  } else {
    std::cout << "The timer already exists : nothing more is created"
              << std::endl;
  }
}

void BasicTimer::on_button_delete_timer() {
  if(not timer_added) {
    std::cout << "Sorry, there is no active timer at the moment." << std::endl;
  } else {
    std::cout << "manually disconnecting the timer " << std::endl;
    disconnect = true;
    timer_added = false;
  }
}
```

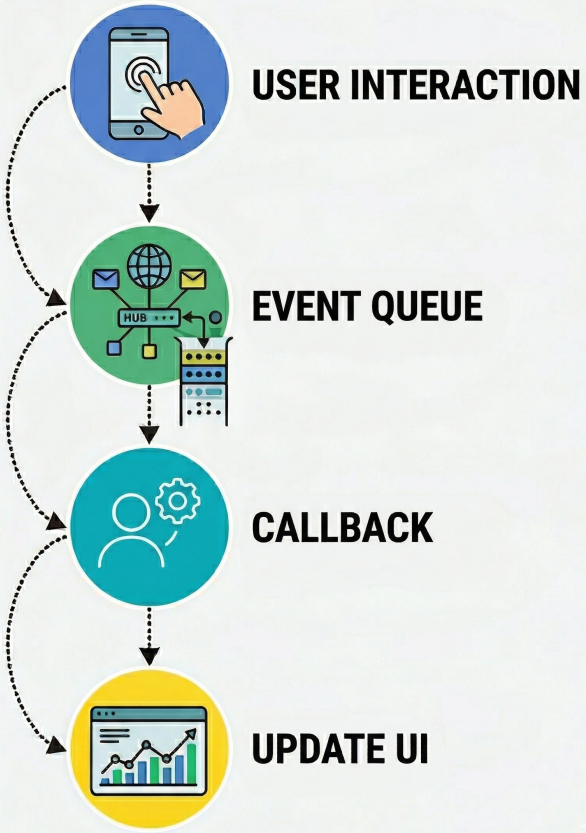
GTKmm4 : Timer

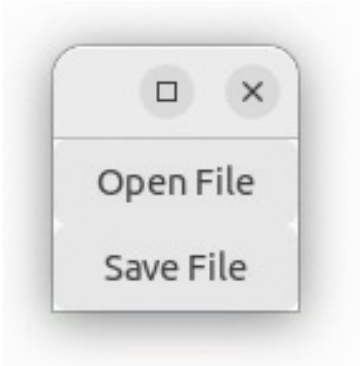


basictimer.cc

```
bool BasicTimer::on_timeout() {
  static unsigned int val(1);

  if(disconnect) {
    disconnect = false; // reset for next time a Timer is created
    return false; // End of Timer
  }

  data_label.set_text(std::to_string(val)); // display the clock
  std::cout << "This is simulation update number : " << val
            << std::endl;
  ++val;
  return true;
}
```

GTKmm4 : File dialog





2 boutons, chacun avec son signal handler, resp.

`on_button_open_clicked()` et

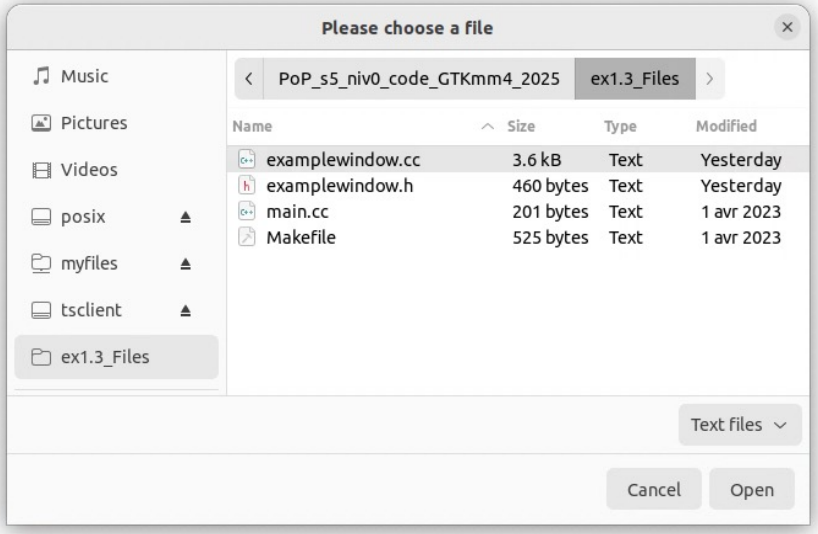
`on_button_save_clicked()`

+ le signal handler de la fenêtre pop-up de dialogue qui servira pour les 2 actions Open et Save:

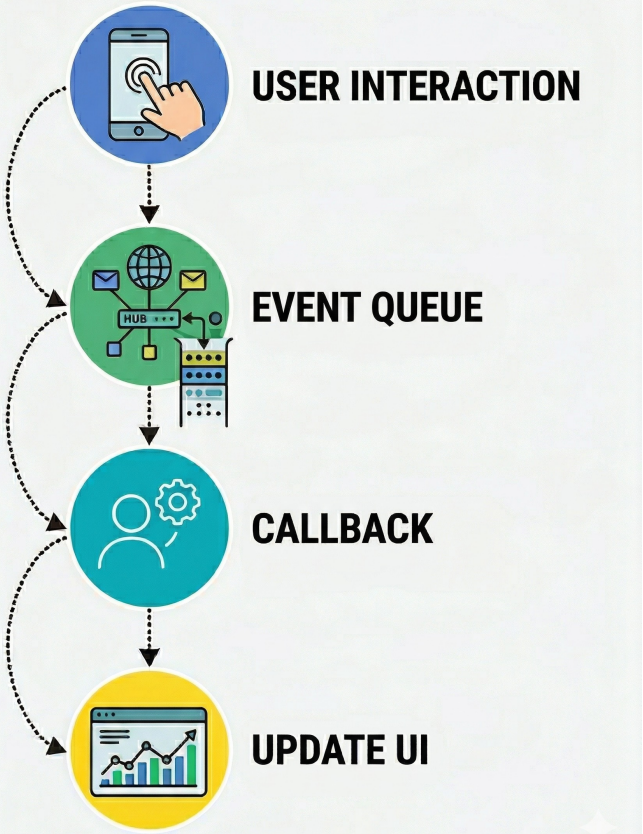
```
void on_file_dialog_response
```

GTKmm4 : File dialog

```
void ExampleWindow::on_button_open_clicked()
{
  auto dialog = new Gtk::FileChooserDialog("Please choose a file",
  Gtk::FileChooser::Action::OPEN);
  dialog->set_transient_for(*this);
  dialog->set_modal(true);
  dialog->signal_response().connect(sigc::bind(
    sigc::mem_fun(*this, &ExampleWindow::on_file_dialog_response), dialog));

  //Add response buttons to the dialog:
  dialog->add_button("_Cancel", Gtk::ResponseType::CANCEL);
  dialog->add_button("_Open", Gtk::ResponseType::OK);

  //Add filters, so that only certain file types can be selected:
  auto filter_text = Gtk::FileFilter::create();
  filter_text->set_name("Text files");
  Filter_text->add_mime_type("text/plain");
  dialog->add_filter(filter_text);
  …
  //Show the dialog and wait for a user response:
  dialog->show();
}
```



GTKmm4 : File dialog




```
void ExampleWindow::on_file_dialog_response(int response_id,
                                            Gtk::FileChooserDialog* dialog) {
  //Handle the response:
  switch (response_id)
  {
  case Gtk::ResponseType::OK:
    {
    std::cout << "Open or Save clicked." << std::endl;
    //Notice that this is a std::string, not a Glib::ustring.
    filename = dialog->get_file()->get_path();
    std::cout << "File selected: " << filename << std::endl;
    break;
    }
  case Gtk::ResponseType::CANCEL:
    {
    std::cout << "Cancel clicked." << std::endl;
    break;
    }
  default:
    {
    std::cout << "Unexpected button clicked." << std::endl;
    break;
    }
  }
  delete dialog;
}
```

rafael.pires@epfl.ch

Merci