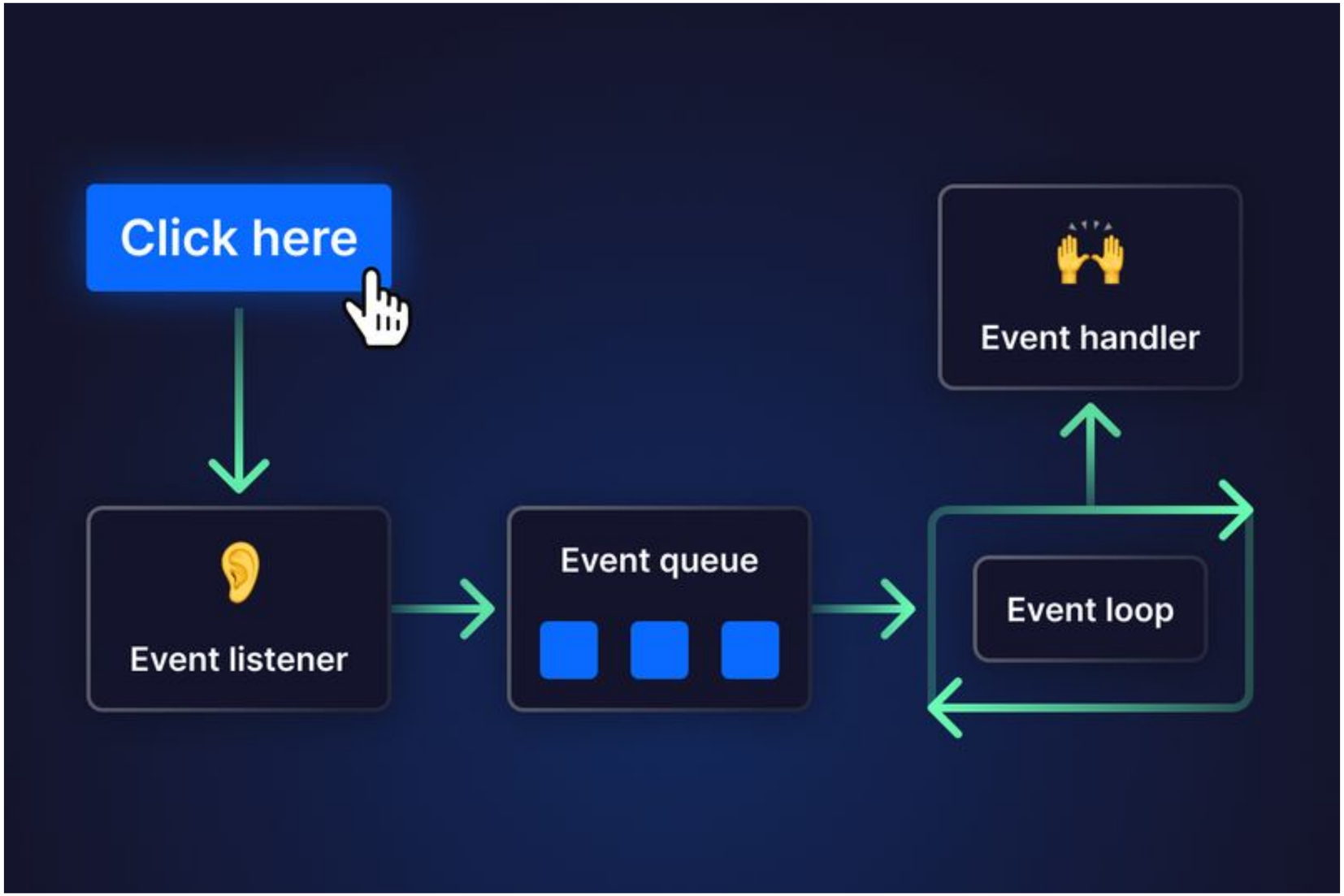
Event Driven Programming

Event Driven Programming



Evento

- Segnale inviato dall'utente al programma
- Asincrono: può arrivare in qualunque momento
- Caratterizzato da:
  - Quale dispositivo proviene (es.: tastiera, mouse)
  - Parametri (es: quale tasto, schiacciato/rilasciato, modificatori)

Event listener e coda

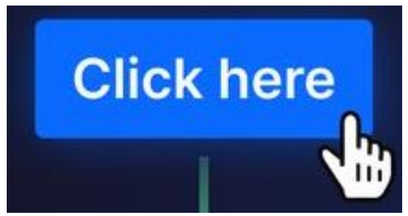
- Event listener:
  - Fa parte del sistema operativo
  - Processo che resta in ascolto, in attesa di eventi
  - Quando arriva un evento, lo aggiunge in coda
- Coda degli eventi:
  - mantiene e consuma gli eventi in ordine di arrivo (FIFO)
  - fa da ponte tra il sistema operativo e il programma
  - ogni programma ha la sua coda

Event loop e Event handler

- Fanno parte del programma
- Event loop:
  - Processo / ciclo che controlla continuamente lo stato della coda
  - Se trova un evento lo estrae e lo passa all'event handler
- Event handler:
  - Reagisce a un evento svolgendo le operazioni correlate al verificarsi dell'evento stesso
  - Le operazioni dipendono da: stato del programma, quale evento si è verificato, parametri che caratterizzano l'evento

Esempio

- Faccio click col mouse su un bottone virtuale (area blu)



- L’event listener cattura l’evento e lo mette in coda

- L’evento non sa di provenire da un bottone virtuale!

Esempio

- Informazioni relative all'evento:
  - proviene dal mouse
  - quale bottone del mouse è stato schiacciato
  - in che posizione dello schermo stava il cursore del mouse quando l'evento è avvenuto
  - eventuali modificatori (SHIFT, CTRL, ALT, ...)

Esempio

- L’event loop si accorge che è arrivato un nuovo evento e lo estrae dalla coda

- Riconosce l’evento in base alle sue caratteristiche e lo passa all’event handler opportuno

- L’event loop si basa su una mappa che associa ogni tipo di evento a un determinato event handler

Esempio

- L’event handler sa come reagire all’evento:
  - vede che il click del mouse è avvenuto in una determinata zona di schermo (quella occupata dal bottone virtuale)
  - consulta lo stato del programma per sapere cosa c’è in quella zona
  - capisce che in tale posizione c’è un determinato bottone virtuale
  - svolge le operazioni previste dal programma per la pressione di tale bottone
  - questo può modificare lo stato del programma e/o avere effetti collaterali

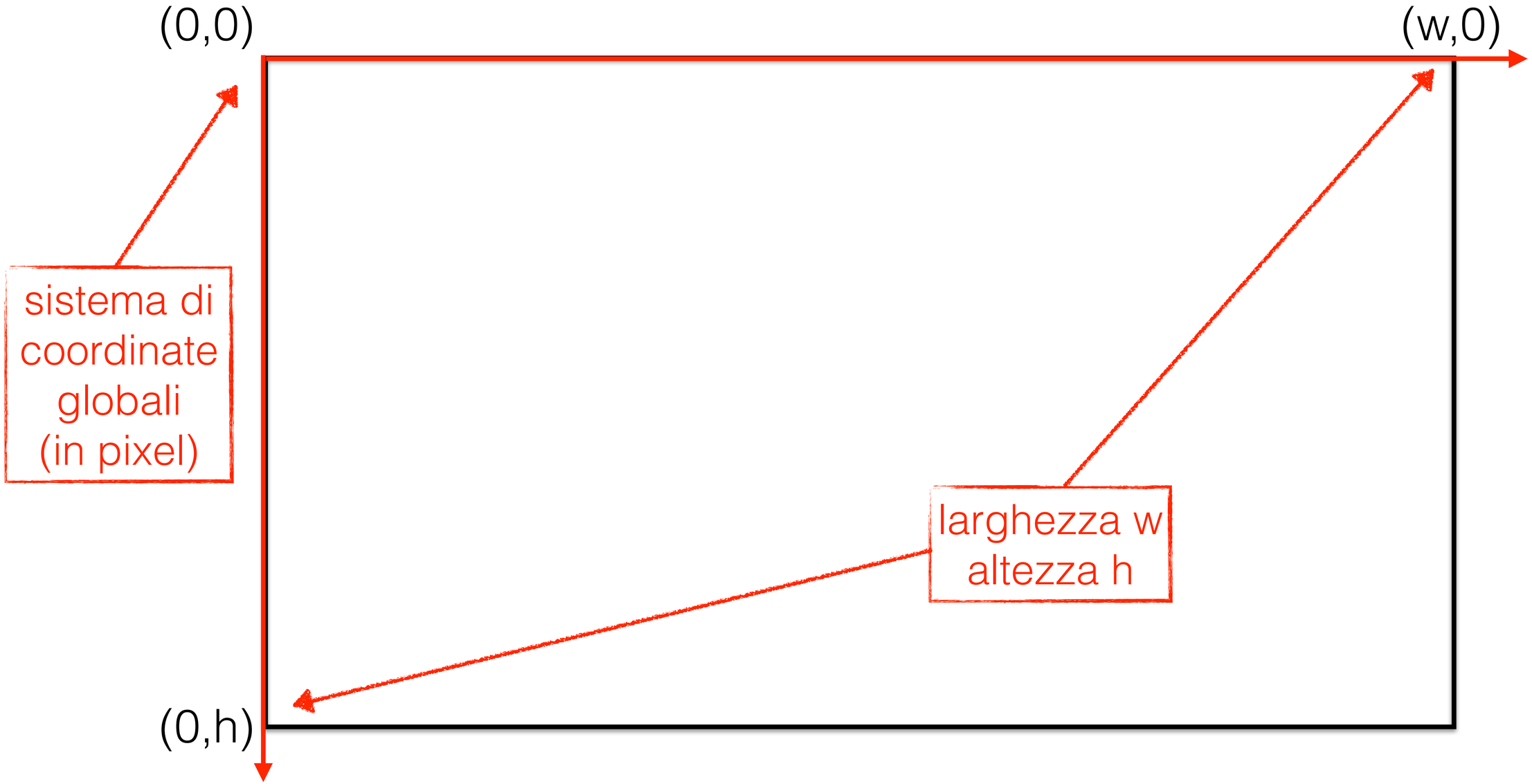
Window Management System

- Parte del sistema operativo
- Implementa l'ambiente "desktop"
- Si frappone fra le applicazioni (dotate di interfaccia utente) e i dispositivi di I/O per gestire l'interazione:
  - Input: eventi dall'utente verso l'applicazione
  - Output: feedback (testo e grafica) dall'applicazione verso l'utente
- Gestisce **tutte** le finestre di **tutte** le applicazioni (che ne hanno)

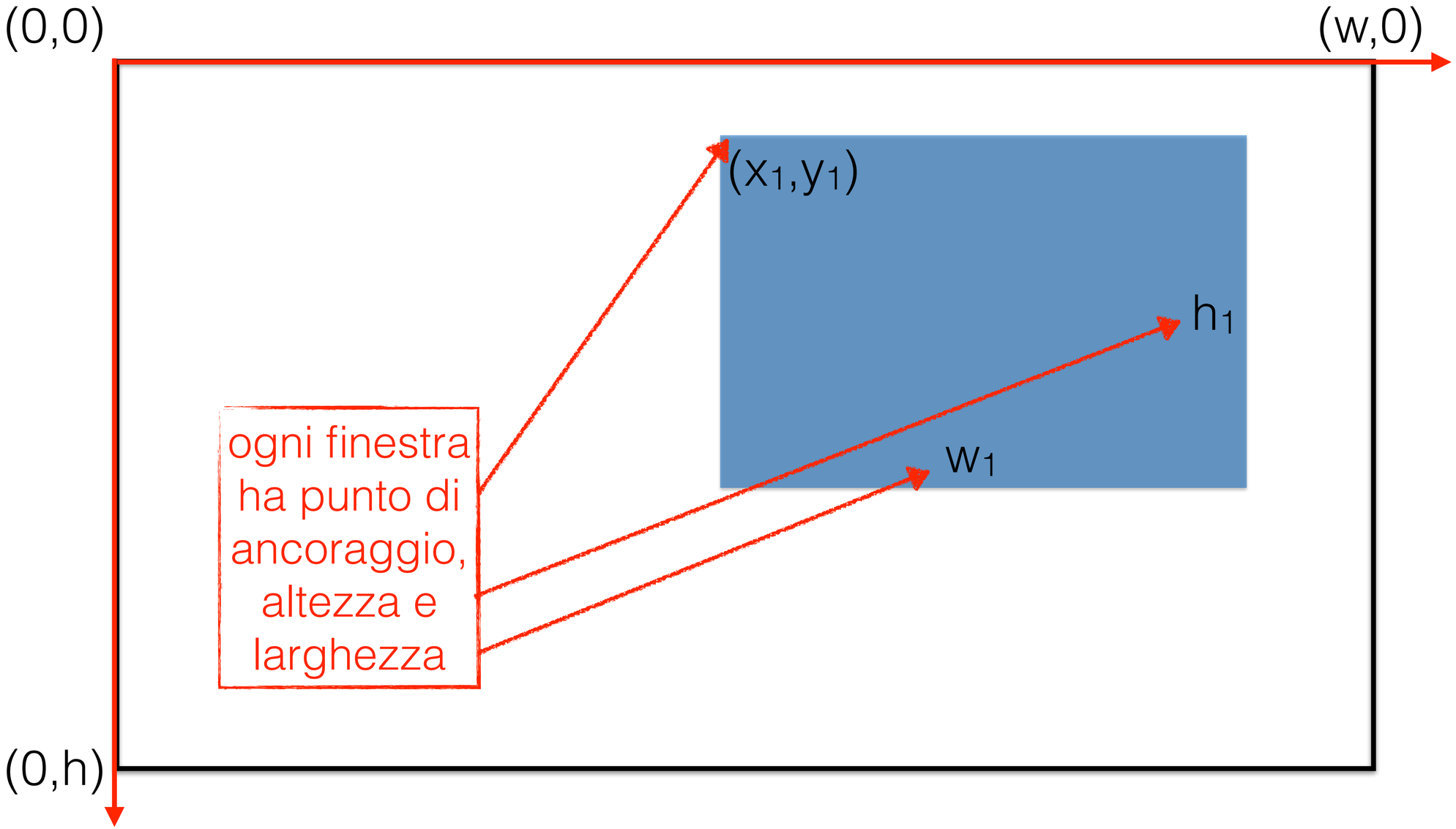
Risorse e richieste

- Un’applicazione interattiva ha bisogno di risorse per poter
  - produrre output (grafica - anche il testo è grafica)
  - ricevere input (eventi)
- Risorse: finestre, font, tavolozza colori, bitmap, pixmap, ...
- Richiesta: atto con cui l’applicazione chiede una risorsa al WMS
- Una volta ottenuta, una risorsa è accessibile tramite un identificatore

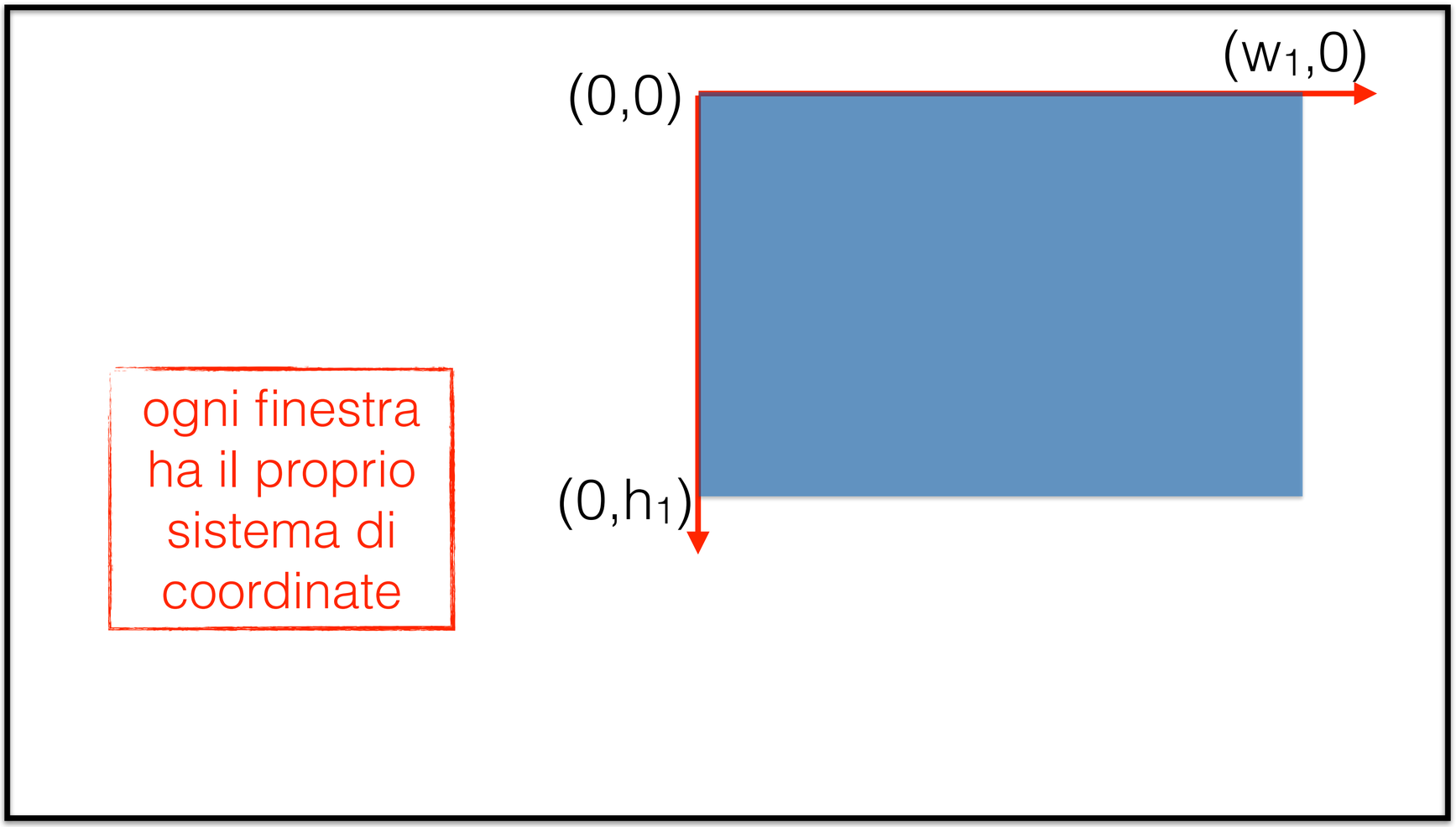
Coordinate schermo



Finestre

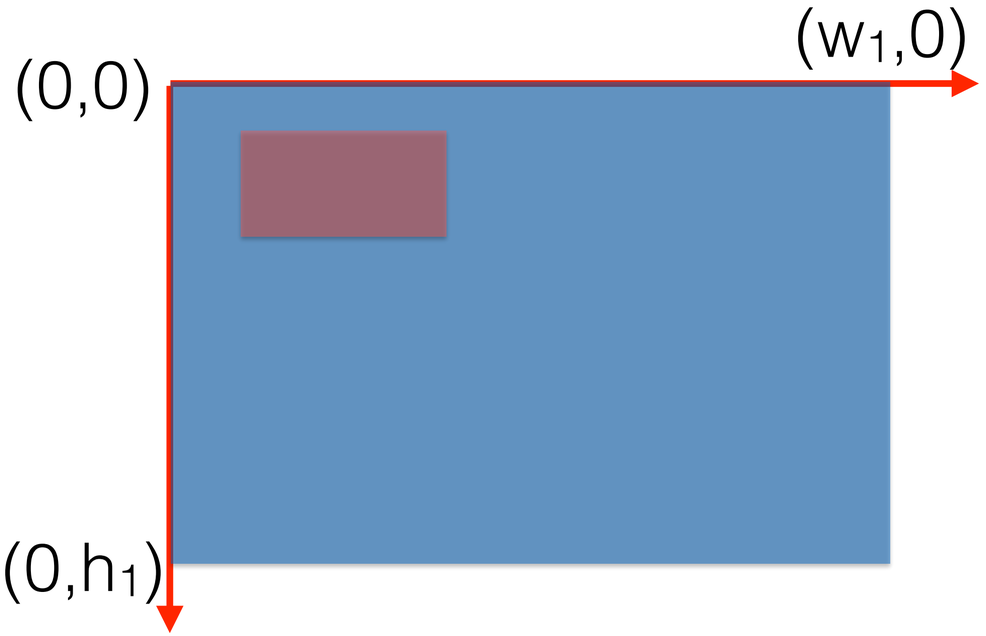

(0,0)
(w,0)
(x1,y1)
h1
w1
ogni finestra ha punto di ancoraggio, altezza e larghezza
(0,h)

Finestre

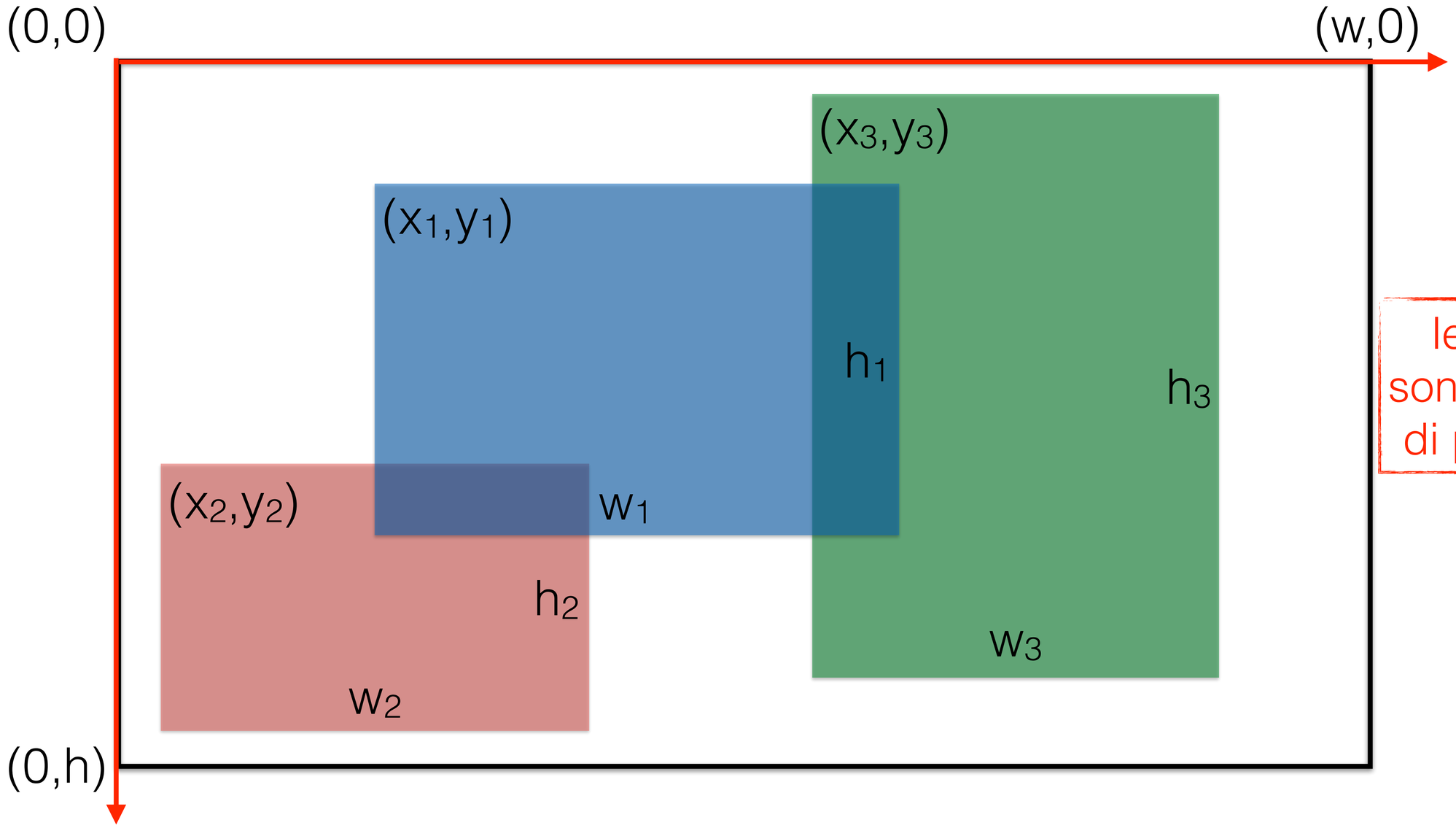

(0,0)
(w1,0)
(0,h1)
ogni finestra ha il proprio sistema di coordinate

Finestre

un’area rettangolare interna alla finestra (es. bottone) ha a sua volta un’origine (rispetto alla finestra), larghezza e altezza



Finestre



le finestre sono in ordine di profondità

Finestre

- Ogni finestra è un’area autonoma di I/O, come uno schermo nello schermo
- Una finestra fa riferimento a un’applicazione
- Un’applicazione può usare una o più finestre (o nessuna)
- Una finestra è caratterizzata da una serie di informazioni:
  - ancoraggio e dimensioni
  - visibile / invisibile
  - caratteristiche grafiche
  - contenuto grafico

Eventi e finestre

- Ogni evento è generato in una finestra
- Una sola finestra ha il **focus** per ogni determinato evento
  - Il focus può essere fissato dalla posizione del cursore del mouse o dal click
  - Ogni evento generato dall'utente si riferisce alla finestra che ha il focus per quel tipo di evento

Eventi

- Tastiera: pressione / rilascio di un tasto
- Mouse:
  - movimento passivo (hover) o con un bottone schiacciato (drag)
  - pressione / rilascio di un bottone (sinistro, centrale, destro)
  - scroll

Eventi

- Finestre:
  - riduzione di finestra a icona / espansione di icona a finestra
  - ridimensionamento di una finestra
  - presa / perdita del focus
  - chiusura

Funzioni callback

- Funzioni che sono “registrate” rispetto al verificarsi di eventi
- Quando si verifica un evento di un certo tipo, il programma manda in esecuzione la callback registrata per quel tipo di evento
- Concetto generale, il modo di implementare le callback dipende dal sistema specifico che si utilizza per mettere in comunicazione l’applicazione col Window Management System

Toolkit

- Libreria ad alto livello per la gestione di finestre ed eventi
- Permette di costruire una Graphical User Interface (GUI) utilizzando costrutti “preconfezionati”:
  - bottoni, cursori, menu, campi testo, maschere, decori
- Noi useremo costrutti a basso livello, per semplicità

- Simple and Fast Multimedia Library
- Libreria C++ multi-piattaforma
- Orientata allo sviluppo di videogame 2D
- Permette la gestione delle risorse e supporta l’Event Driven Programming a basso livello
- https://www.sfml-dev.org/

- Versione SFML 3.0
- Installazione: https://www.sfml-dev.org/tutorials/3.0/
- Scegliete una delle modalità (CMake, Visual Studio, Linux, ecc.)
- Consigliata, unica per la quale posso dare assistenza: CMake

Finestre in SFML

- **`sf::RenderWindow`** è una classe che fornisce window sulle quali si può disegnare in SFML
- Una window è caratterizzata da:
  - Risorse video: dimensioni in pixel w e h, profondità in bit-per-pixel
  - Titolo (stringa), Stile, Stato (floating o fullscreen)
  - Eventuali informazioni di contesto (per ora non le usiamo)

Finestre in SFML

- Operazioni principali:
  - creazione: l'applicazione chiede al WMS di fornirgli una window
  - close: chiusura definitiva della window e delle sue risorse
  - display: mostrare a schermo il contenuto grafico della window
  - clear: cancellazione del contenuto grafico
  - gestione del focus
  - pollEvent, waitEvent: gestione degli eventi (solo event listener)

Finestre in SFML

- Esempio minimale: main_window0.cpp

```cpp
#include <SFML/Graphics/RenderWindow.hpp>

int main()
{
    auto window = sf::RenderWindow(sf::VideoMode({800u, 600u}), "CMake SFML Project");
    while (window.isOpen())
    {
        while (const std::optional event = window.pollEvent())
        {
            if (event->is<sf::Event::Closed>())
            {
                window.close();
            }
        }
        window.clear();
        window.display();
    }
}
```

Eventi in SFML

- **`sf::Event`** è il tipo generico per qualunque evento
- ci sono poi sotto-tipi per eventi specifici:
  - **`Closed`** evento di chiusura finestra
  - **`KeyPressed/KeyReleased`** è stato premuto/rilasciato un tasto
  - **`MouseButtonPressed / MouseButtonReleased`**
  - **`MouseMoved`**
  - ecc...

Eventi in SFML

- Due modi principali (alternativi) per implementare event loop e event handler

1. Modo classico:
    - L'event loop è un ciclo while che fa *polling* sulla coda degli eventi
    - A ogni ciclo, estrae un evento dalla coda, ne riconosce il sotto-tipo e esegue le operazioni previste per quel tipo di evento
    - L'event handler è implementato direttamente dentro il loop

Eventi in SFML

- Schema del modo classico:
  - si estrae un evento generico **`event`** tramite **`pollEvent`**
  - in una cascata di if, ci si chiede di che tipo di evento si tratta:
    - **`event->is<T>()`** è una funzione Booleana che dice se l'evento è di tipo **`T`**. Si usa se non serve altro che il tipo di evento per trattarlo
    - **`event->getIf<T>()`** è una funzione che restituisce i dati dell'evento se questo è di tipo **`T`**. Si usa se i dati dell'evento sono rilevanti (es: posizione del mouse, tasto premuto)

Eventi in SFML

- Modo classico: main_event_classic_cout.cpp

Eventi in SFML

2. Tramite callback:
   - si definisce una funzione callback per ogni tipo di evento
   - event loop e event handler sono implementati da un'unica chiamata alla funzione **`handleEvents`**
     - riceve come parametri la lista delle callback
     - estrae in successione eventi dalla coda, finché ce ne sono
     - per ogni tipo di evento, applica la callback opportuna

Eventi in SFML

- Schema del modo a callback:
  - si scrivono diverse funzioni void **`handle`** ciascuna con un solo parametro del tipo di evento da trattare
  - si passa il controllo degli eventi alla funzione membro **`handleEvents`** che riceve come parametro un'espressione lambda con template automatico sul tipo di evento:

```
window.handleEvents([](const auto &event)
                        { handle(event); });
```

Eventi in SFML

- Di solito le funzioni handle devono modificare lo **stato del programma**, quindi devono poter accedere alle **variabili di stato**
- Soluzione facile (e sporca): lo stato è una variabile globale
- Soluzione più pulita: lo stato è una variabile `gs` del main e viene passata come ulteriore parametro alle funzioni handle:

```
gs.window.handleEvents([&](const auto &event)
                       { handle(event,gs); });
```

Eventi in SFML

- Modo callback: main_event_callback_cout.cpp