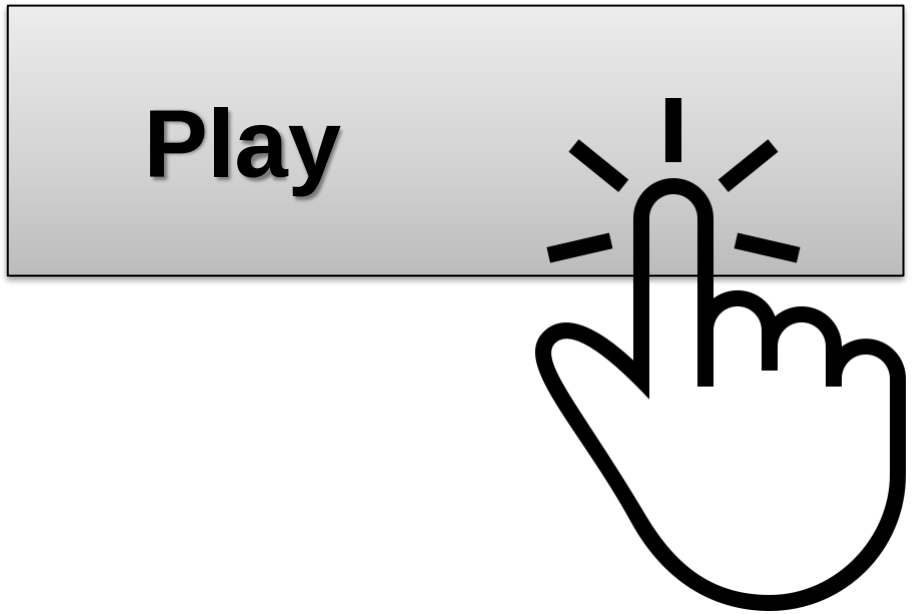
# CPSC 427
# Video Game Programming

## IO and the Observer Pattern

Helge Rhodin

# Recap: Motivation: Object selection

- ***Point inside object boundary?***



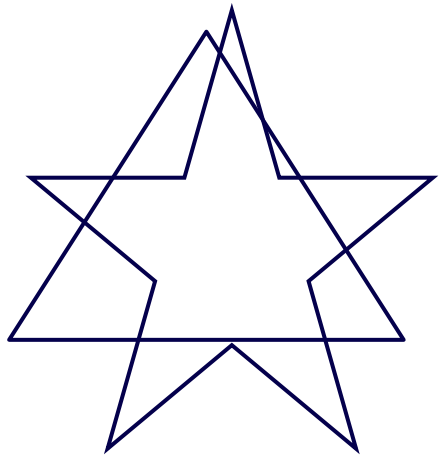
# Motivation: Bullet trajectories

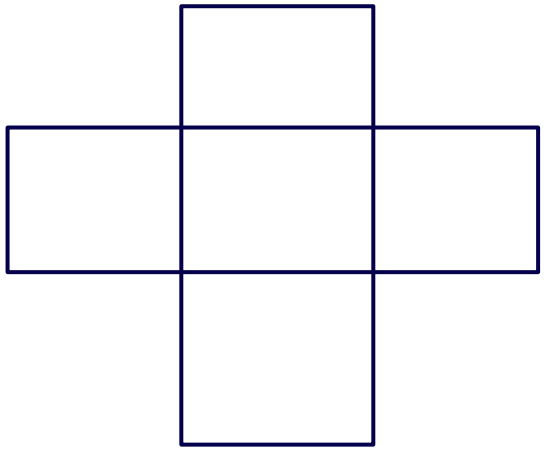
- ***Line-object or point-object intersection?***

**https://forum.unity.com/threads/2d-platformer-shooting.365971/**

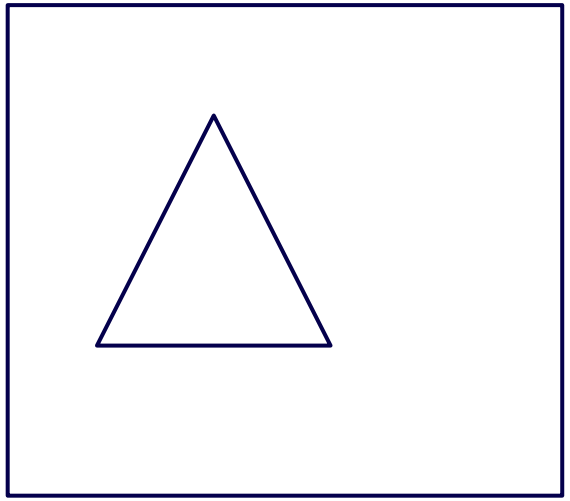
# Collision Configurations?

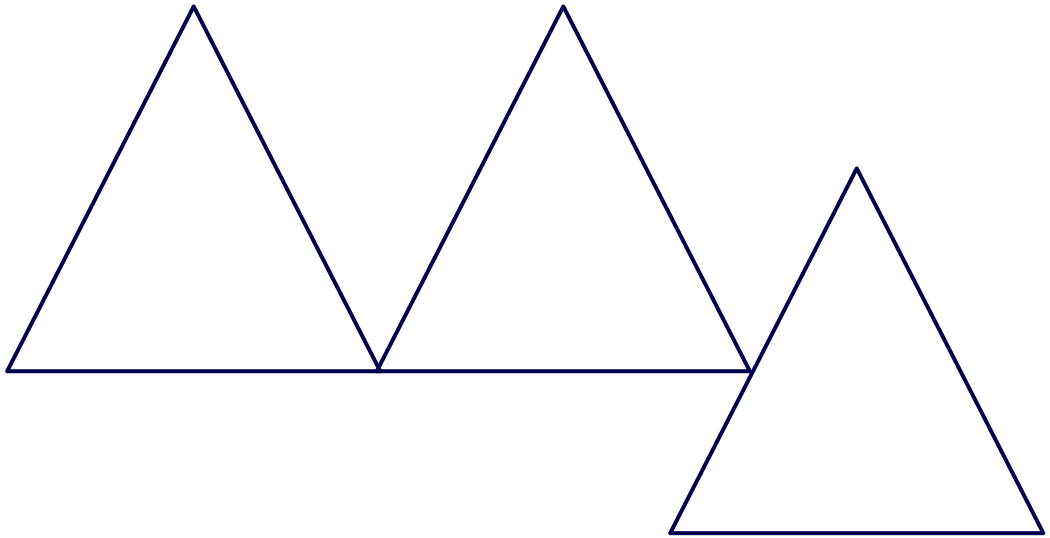
- Segment/Segment Intersection
  - *Point **on** Segment*





- Polygon inside polygon





# Inside Test?

- How to test if one poly is inside another?

- Use inside test for point(s)

- How?
  - *Convex Polygon*
    - Same side WRT to line (all sides)
  - *Non-Convex*
    - Subdivide= triangulate
    - How?
    - Shoot rays (beware of corners and special cases)

# Explicit functions

- $y = f(x)$
- E.g. $y = a x + b$
- Single y value for each x
- Useful for?
  - *Terrain*
  - *“height field” geometry*

# Parametric Functions

- 2D: x and y are functions of a parameter value t
- 3D: x, y, and z are functions of a parameter value t

$$C(t) := \begin{pmatrix} P_y^0 \\ P_x^0 \end{pmatrix} t + \begin{pmatrix} P_y^1 \\ P_x^1 \end{pmatrix} (1-t)$$

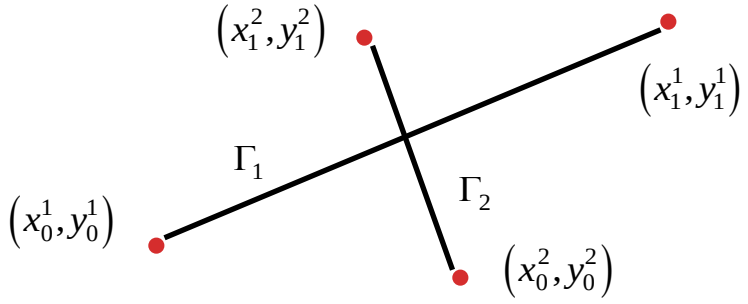
**Line (segment)**

$$C(t) := \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}$$

**Circle (arc)**

- Depends on parameter range t1 < t < t2

# Line-Line Intersection




$$G_1 = \begin{cases} x^1(t) = x_0^1 + \left(x_1^1 - x_0^1\right)t \\ y^1(t) = y_0^1 + \left(y_1^1 - y_0^1\right)t \end{cases} \quad t \in [0,1] \qquad G_2 = \begin{cases} x^2(r) = x_0^2 + \left(x_1^2 - x_0^2\right)r \\ y^2(r) = y_0^2 + \left(y_1^2 - y_0^2\right)r \end{cases} \quad r \in [0,1]$$

**Intersection: *x* & *y* values equal in both representations - two linear equations in two unknowns (*r*,*t*)**

$$x_0^1 + \left(x_1^1 - x_0^1\right)t = x_0^2 + \left(x_1^2 - x_0^2\right)r$$
$$y_0^1 + \left(y_1^1 - y_0^1\right)t = y_0^2 + \left(y_1^2 - y_0^2\right)r$$

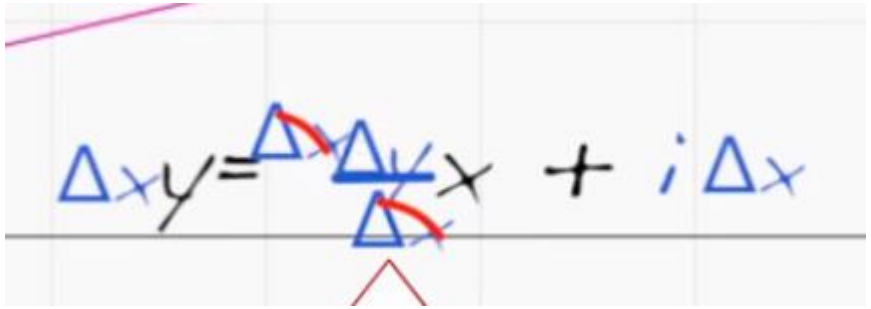
**Question: What is the meaning if the solution gives *r*,*t* < 0 or *r*,*t* >1 ?**

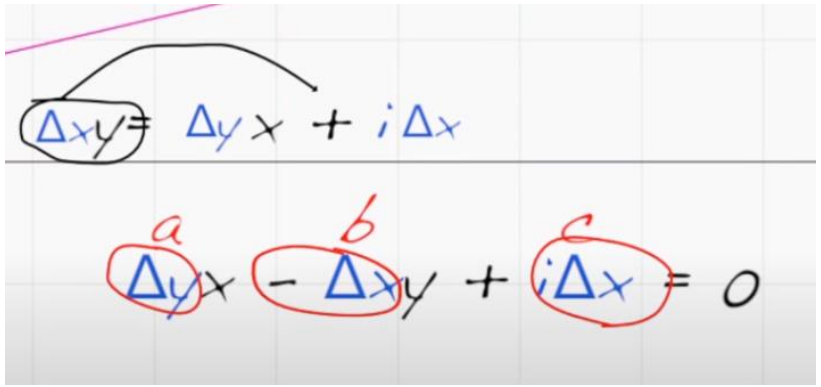
# Implicit Line

**Explicit: y = m x + b** **Implicit:** $Ax + By + C = 0$

$y = dy/dx\ x + b$
$dx\ y = dy\ x + dx\ b$
$0 = dy\ x - dx\ y + dx\ b$

$\Rightarrow A = dy,\ B = -dx,\ C = dx\ b$



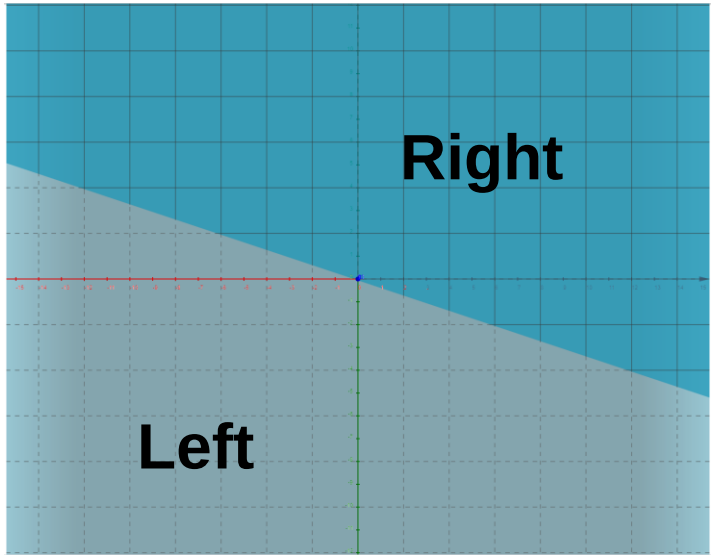


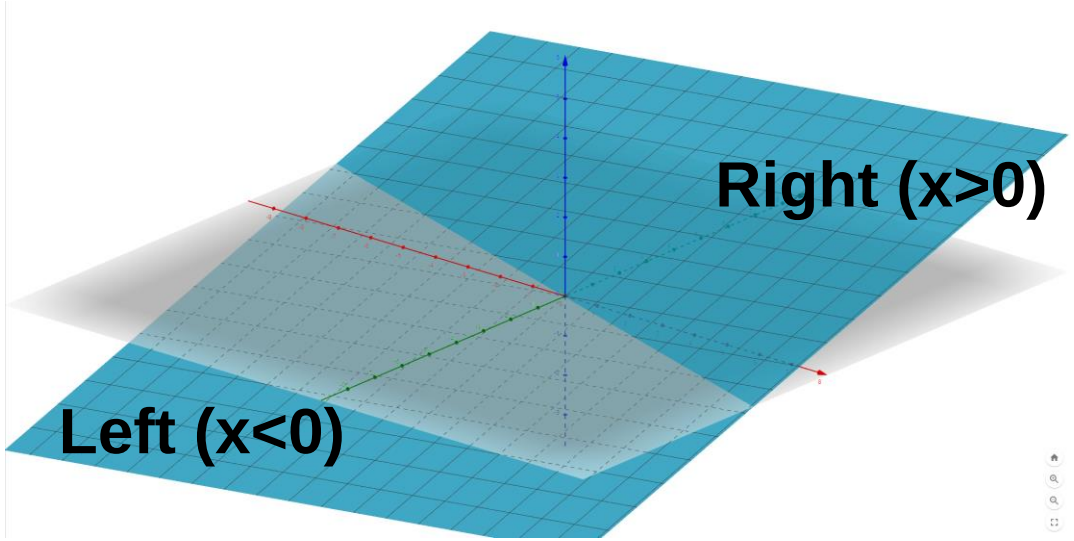
*Example*
$y = -1/3\ x + 0$
$dx = -1,\ dy = 3,$
$A = 3,\quad B = 1,\quad C = 0$
**=> 1 x + 3 y = 0**

# Implicit Line – left or right?

***Implicit line in 2D <-> Explicit plane in 3D***

***$0.1\,x + 0.3\,y = 0$ <-> $f(x,y) = 0.1\,x + 0.3\,y$***

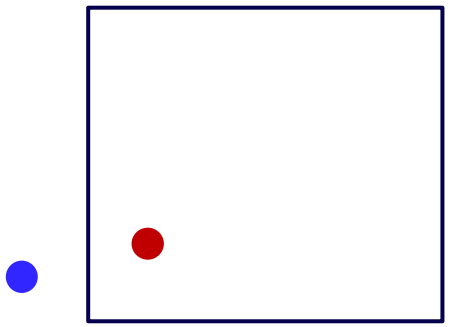



https://www.geogebra.org/3d

# Point vs Line (Ray)

- Point $P=(P_x,P_y)$
- Use implicit equation to determine coincidence & side
  - *Implicit* $A\,x + B\,y + C = 0$
  - *Solve 2 equations in 2 unknowns (third equation: set* $A^2+B^2=1$*)*
  - *On:* $A\,P_x + B\,P_y + C=0$
  - *Use same orientation to get consistent left/right orientation for inside test for lines defining CONVEX polygon*
    - Same sign implies inside
    - *Eg.* ***ALL*** $AP_x + B\,P_y + C < 0$



# Self-study:
# Winding number algorithm

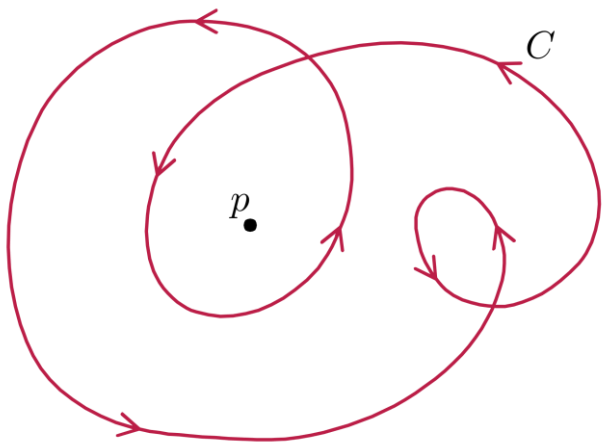
Point in polygon?

- If the winding number is non-zero

- How to compute the winding number?
  - http://geomalgorithms.com/a03-_inclusion.html

Winding number:

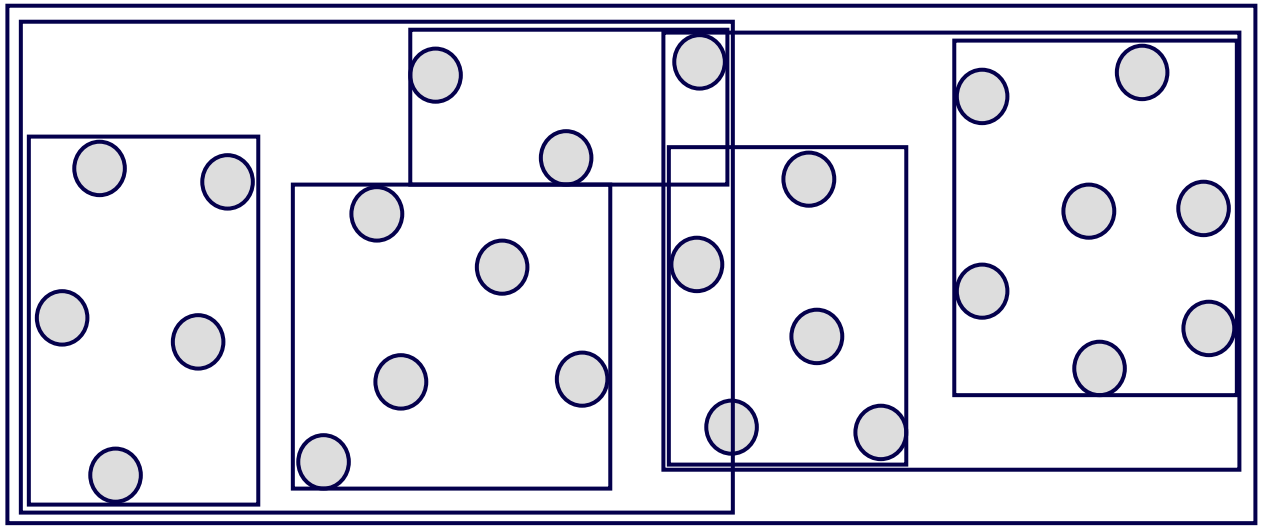
- the number of times that curve travels counterclockwise around the point
- negative if clockwise



# Hierarchical Bounding Volumes

## *Bound Bounding Volumes:*

- Use (hierarchical) bounding volumes for groups of objects

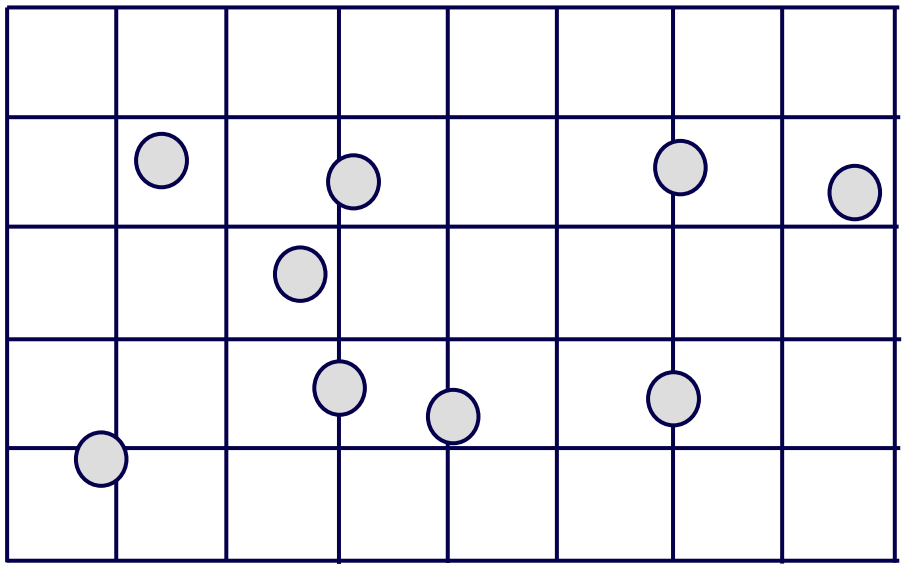


- Challenge: dynamic data…
  - *Need to update hierarchy efficiently*

# Creating a Regular Grid

***Steps:***

- Find bounding box of scene
- Choose grid resolution in x, y, z
- Insert objects
- Objects that overlap multiple cells get referenced by all cells they overlap



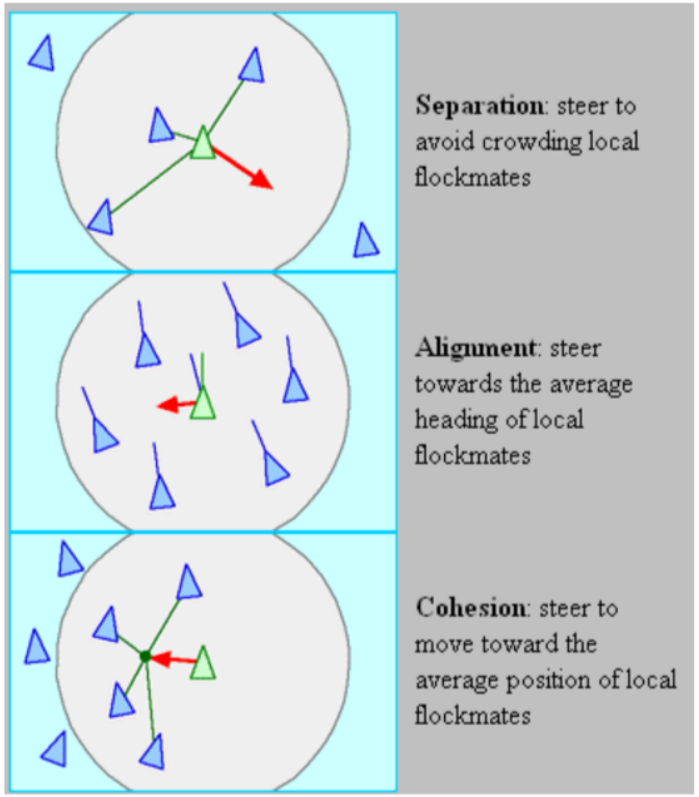
# Basic Particle Simulation (first try)

Forces only $\vec{F} = ma$

$$d_t = t_{i+1} - t_i$$
$$\vec{v}_{i+1} = \vec{v}(t_i) + (\vec{F}(t_i)/m)d_t$$
$$\vec{p}_{i+1} = \vec{p}(t_i) + \vec{v}(t_{i+1})d_t$$

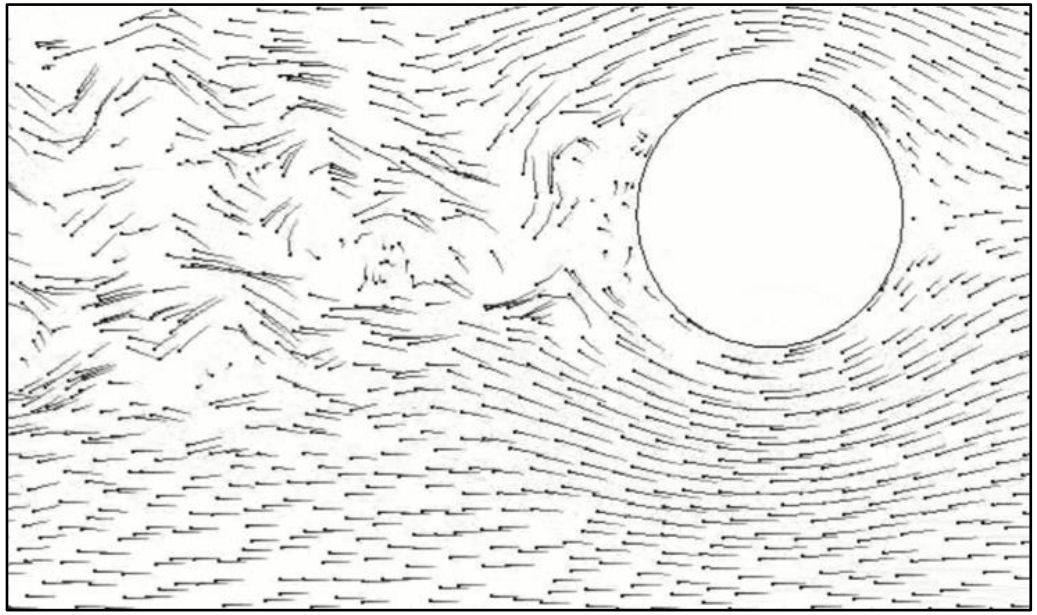
# Proxy Forces

- **Behavior forces:**
  **flocking birds, schooling fish, etc.**
  **[“Boids”, Craig Reynolds, SIGGRAPH 1987]**



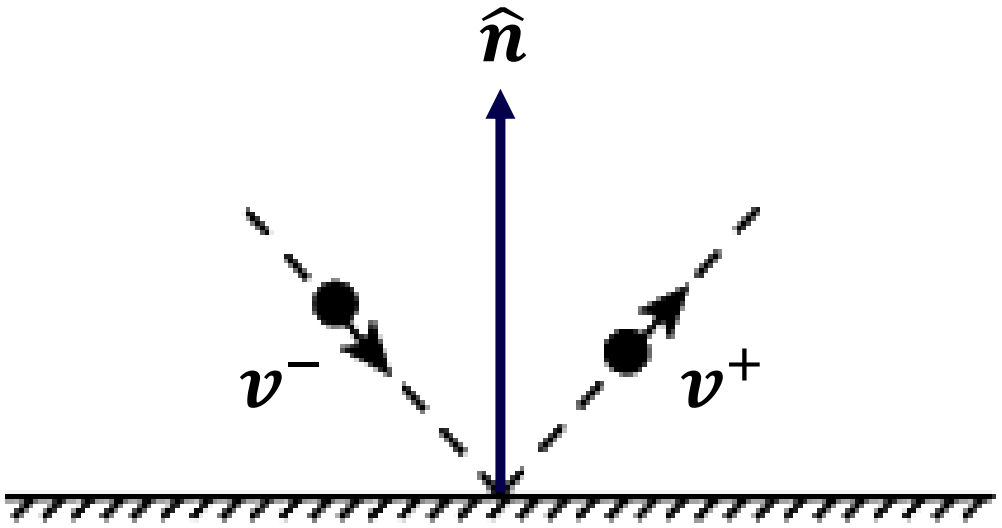

Courtesy of Craig W. Reynolds. Used with permission.

- **Fluids**
  **[“Curl Noise for Procedural Fluid Flow”**
  **R. Bridson, J. Hourihan, M. Nordenstam,**
  **Proc. SIGGRAPH 2007]**



# Particle-Plane Collisions

- **More formally…**
  - ***Apply an impulse of magnitude j***
    - Inversely proportional to mass of particle
  - ***In direction of normal***



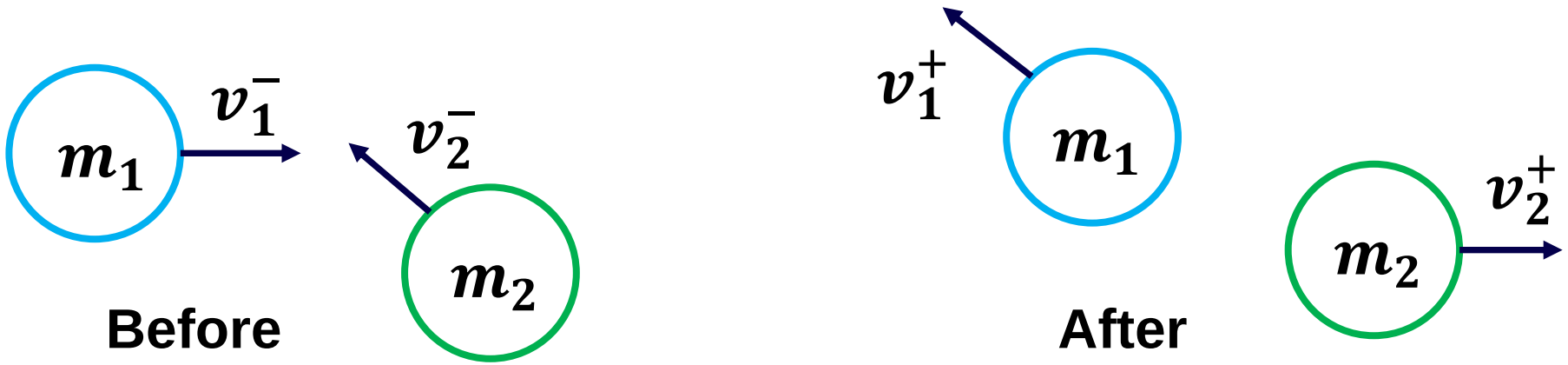

$$j = (1 + \epsilon)m$$

$$\vec{j} = j\,\hat{n}$$

$$v^+ = \frac{\vec{j}}{m} + v^-$$

# Particle-Particle Collisions (radius=0)

- **Particle-particle frictionless elastic impulse response**




- **Momentum is preserved**

$$m_1 v_1^- + m_2 v_2^- = m_1 v_1^+ + m_2 v_2^+$$

- **Kinetic energy is preserved**

$$\tfrac{1}{2} m_1 {v_1^-}^2 + \tfrac{1}{2} m_2 {v_2^-}^2 = \tfrac{1}{2} m_1 {v_1^+}^2 + \tfrac{1}{2} m_2 {v_2^+}^2$$

# CPSC 427
# Video Game Programming

## IO and the Observer Pattern

Helge Rhodin

# Mainloop

*a1.cpp*

***int main(int argc, char* argv[]) {***

...

## 2. Mainloop:

# Mainloop

*a1.cpp*

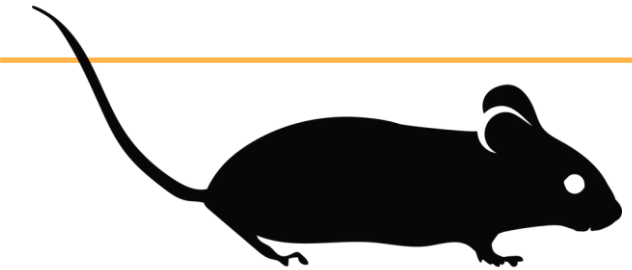
***int main(int argc, char* argv[]) {***

**…**

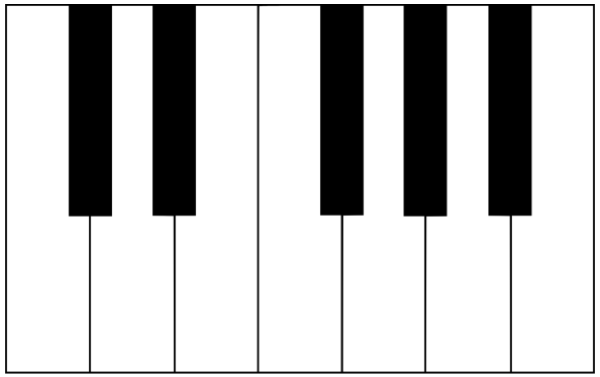
## 2. Mainloop:

```
while (!world.is_over()) {
```

# Event Processing

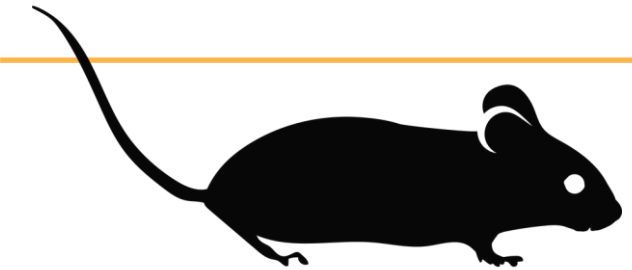


Mouse event,
Keyboard event,
etc.

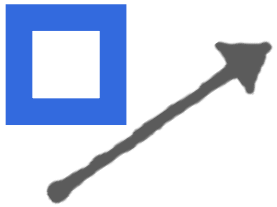


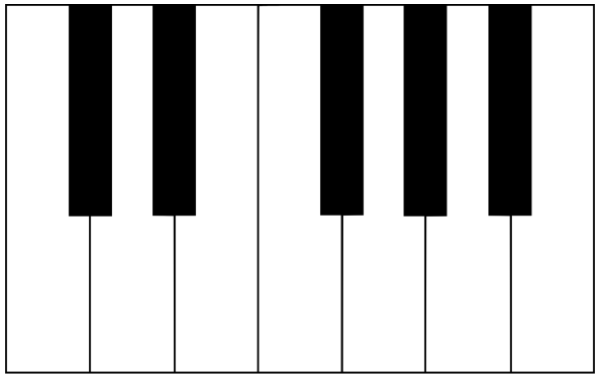
Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

# Event Processing: Event Queuing



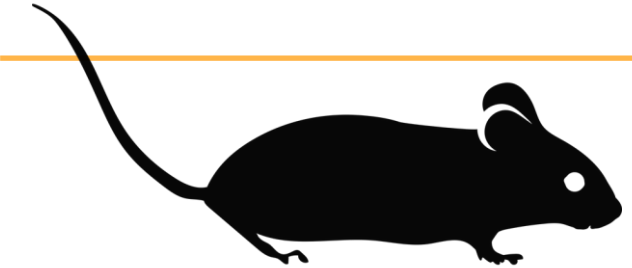
**Mouse event,**
**Keyboard event,**
**etc.**



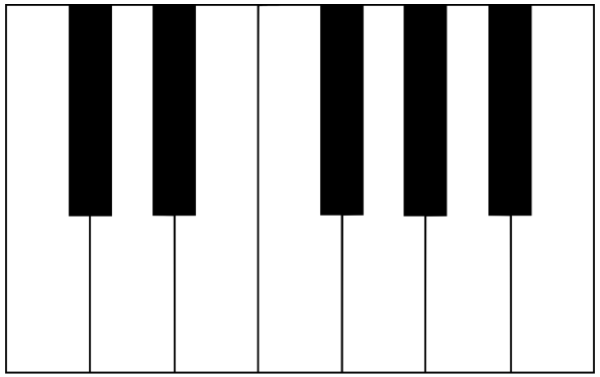



Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

# Event Processing: Event Polling

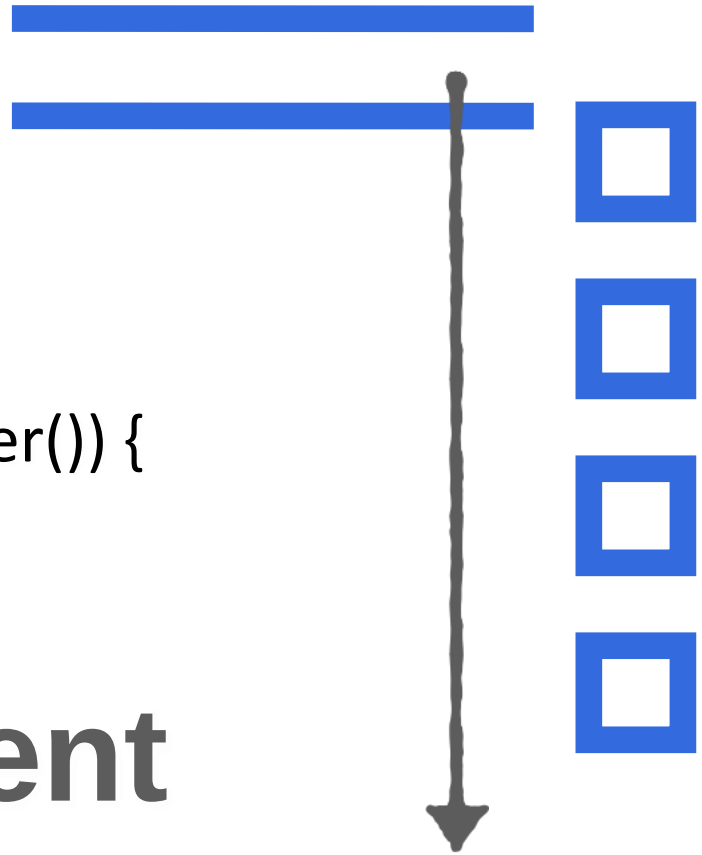


Mouse event,
Keyboard event,
etc.



```
while (!world.is_over()) {
```

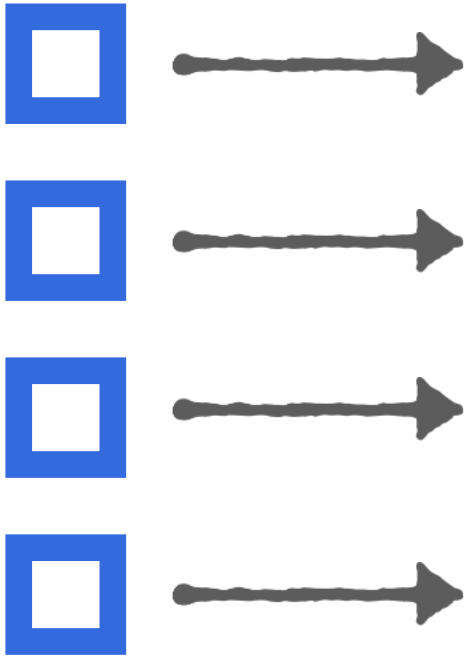
Event queue



## 2.1 Event processing

Credits:
https://pixabay.com/en/mouse-mouse-silhouette-lab-mouse-2814846/
https://svgsilh.com/image/25711.html

# Event Processing: Event Callback



## GLFW calls corresponding callbacks:

- `void World::on_key(GLFWwindow*, int key, int, int action, int mod)`

  —-> world.cpp:L369

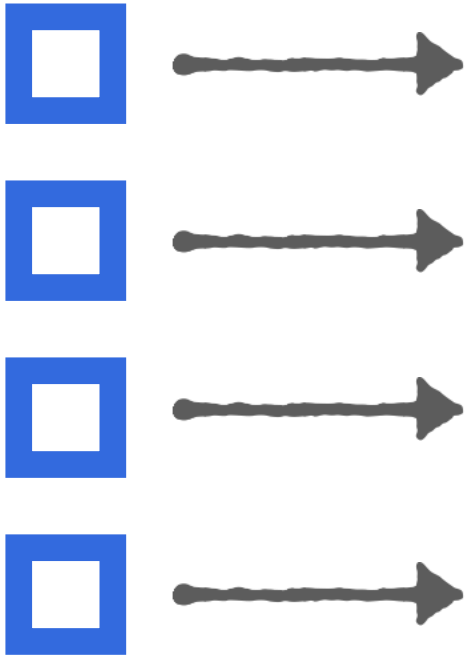
  You need to set salmon motion here.

- `void World::on_mouse_move(GLFWwindow* window, double xpos, double ypos)`

  —-> world.cpp:L399

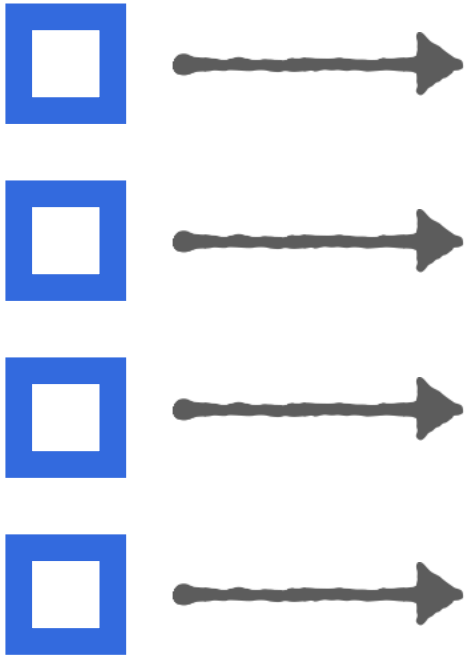
  You need to fill this function to set salmon rotation.

# Event Processing: Event Callback



**How does GLFW know which callback to call?**

# Event Processing: Event Callback



**How does GLFW know which callback to call?**

**—-> Registered in initialization:**

```
world.init(…)
```

```
glfwSetKeyCallback
glfwSetCursorPosCallback
```

# Mainloop

*a1.cpp*

***int main(int argc, char* argv[]) {***

**…**

## 2. Mainloop:

```
while (!world.is_over()) {
```

# glfwPollEvents

- ***Asynchronous?***
- ***Reference:***
  ***https://www.glfw.org/docs/3.0/group__window.html#ga37bd57223967b4211d60ca1a0bf3c832***
  - "This function processes only those events that have already been received and then returns immediately. Processing events will cause the window and input callbacks associated with those events to be called."
    - synchronous!

- "On some platforms, certain callbacks may be called outside of a call to one of the event processing functions."
  - asynchronous! :/

# Workaround?

- ***https://stackoverflow.com/questions/36579771/glfw-key-callback-synchronization***

2

Generally speaking, the way that you should be handling input is to keep a list of keys, and record their last input state.

```
struct key_event {
    int key, code, action, modifiers;
    std::chrono::steady_clock::time_point time_of_event;
}

std::map<int, bool> keys;
std::queue<key_event> unhandled_keys;
void handle_key(GLFWwindow* window, int key, int code, int action, int modifiers) {
    unhandled_keys.emplace_back(key, code, action, modifiers, std::chrono::steady_clock::no
}
```

Then, in the render loop (or you can separate it into a different loop if you're confident with your multithreading + synchronization abilities) you can write code like this:

```
float now = glfwGetTime();
static float last_update = now;
float delta_time = now - last_update;
last_update = now;
handle_input(delta_time);
```

# The Observer Pattern

- ***Gang of Four (GoF)***
  - *Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides*
  - ***Design Patterns: Elements of Reusable Object-Oriented Software*** (1994)

- ***A pattern described by the GoF***
- event-driven
  - *clients register for an event*

**Good ref (object oriented):**
**https://gameprogrammingpatterns.com/observer.html**

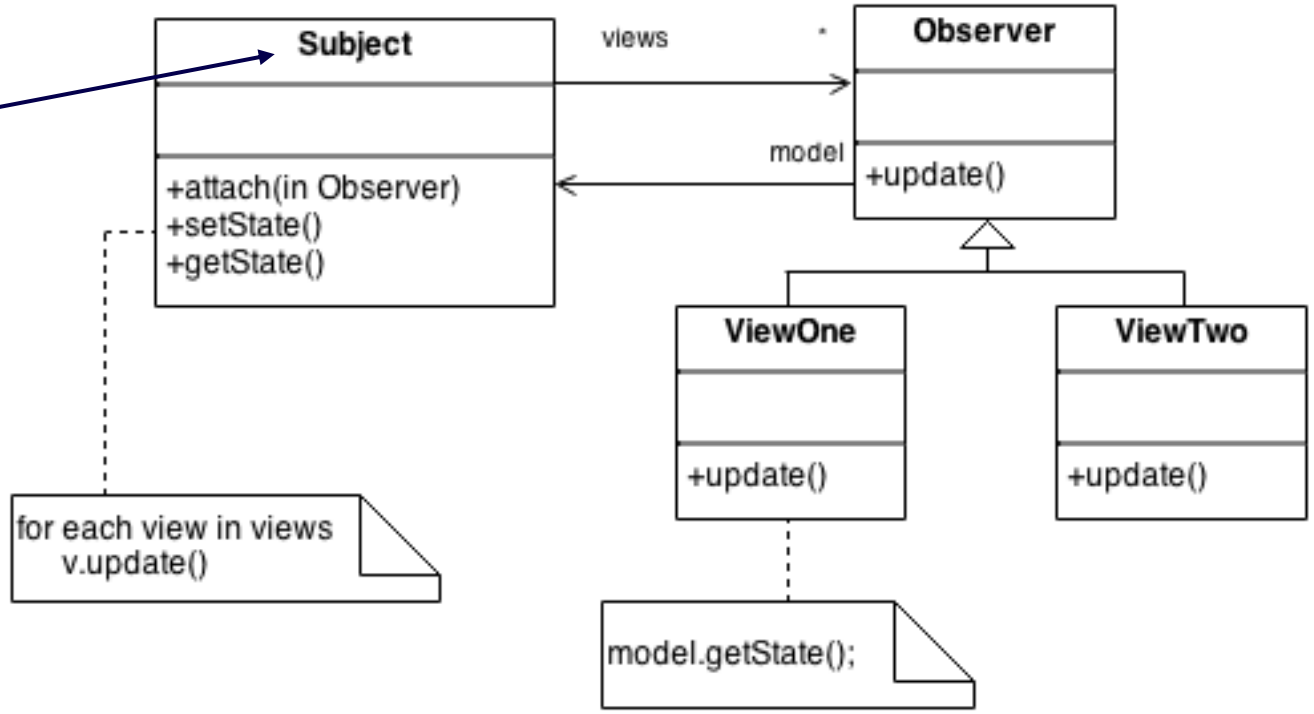
# Use Cases

- ***Rewards***
- ***Communication between systems (in ECS)***
- ***User input***
- ***???***

# Observer Pattern – OOP

- *Define a common interface*
- *All observers inherit from that interface*

**Called Subject by GoF**



# Lambda Functions

***Definition:***

- `auto y = [] (int first, int second) { return first + second; };`

Call: `int z = y(1+3);`

- Infers return type for simple functions (single return statement)
  - otherwise

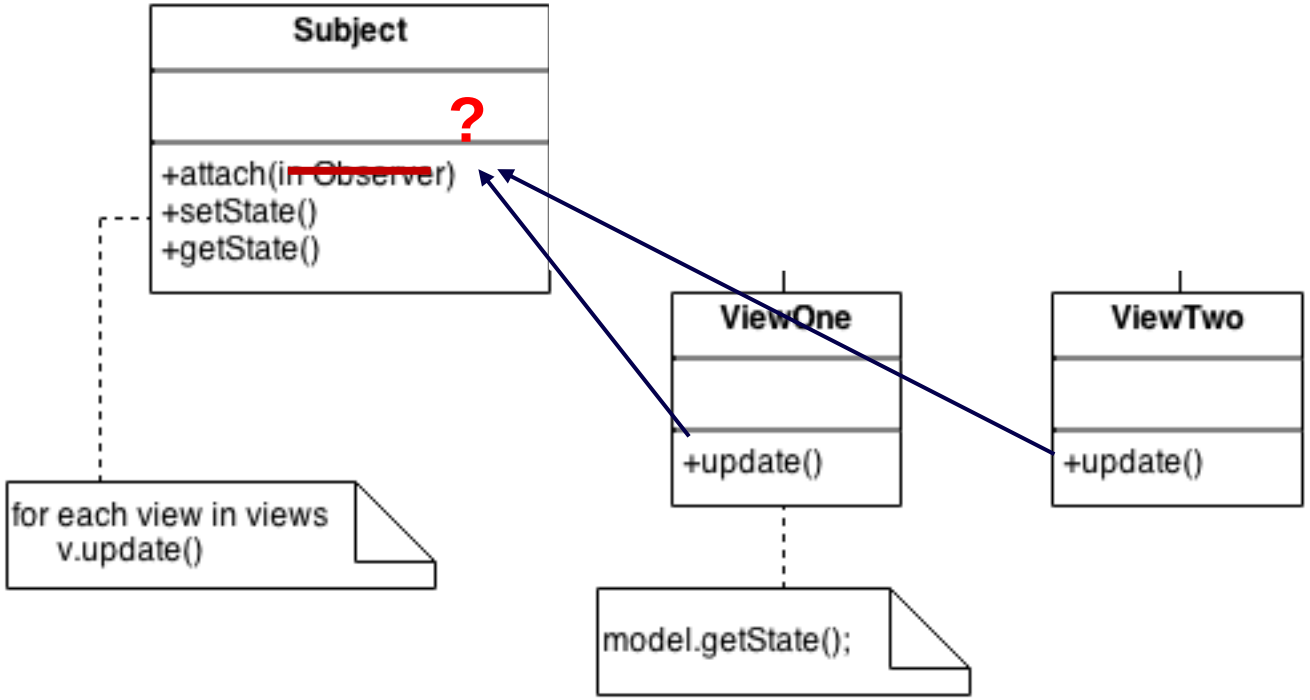
    `auto y = [] (int first, int second) -> int { return first + second; };`
- Can capture variables from the surrounding scope.

```
int scale;
auto y = [] (int first, int second) -> int { return scale*first + second; };

auto y = [&] (int first, int second) -> int { return scale*first + second; };
```

# Observer Pattern – With Functions

- ***function with matching signature instead of class***



# A function that accepts a function

- ***Using std::function***

```
#include<functional>

void LambdaTest (const std::function <void (int)>& f)
{
    ...
}
```

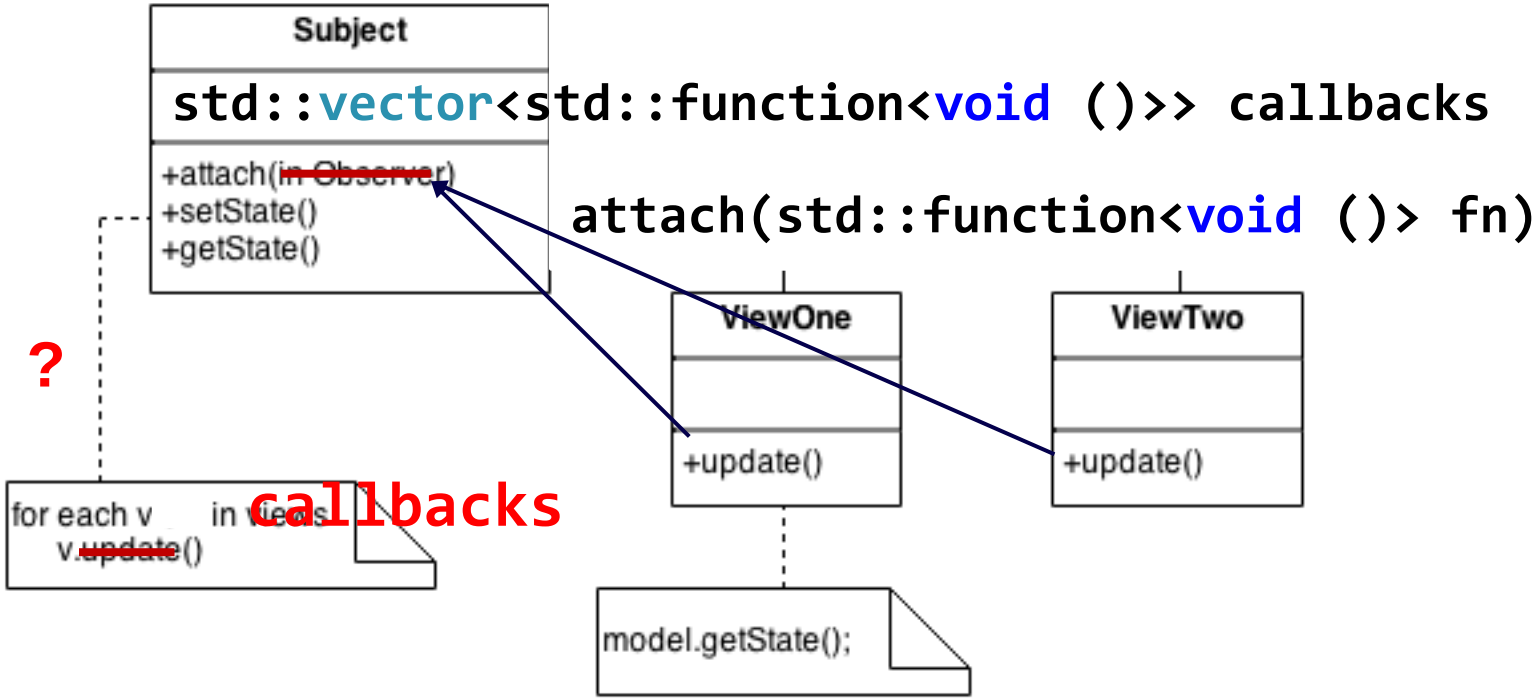
- ***Using templates***

```
template<typename Func>
void LambdaTest(Func f) {
    f(10);
}
```

  - *use templates to accept any argument with an operator()*

# Observer Pattern – With Functions

- ***function with matching signature instead of class***



# IO in our Template

- ***????????***

```
// Setting callbacks to member functions (that's why the redirect is needed)
// Input is handled using GLFW, for more info see
// http://www.glfw.org/docs/latest/input_guide.html
glfwSetWindowUserPointer(window, this);
auto key_redirect = [](GLFWwindow* wnd, int _0, int _1, int _2, int _3) { ((WorldSystem*)glfwGetWindowUserPointer(wnd))->on_key(wnd, _0, _1, _2, _3); };
auto cursor_pos_redirect = [](GLFWwindow* wnd, double _0, double _1) { ((WorldSystem*)glfwGetWindowUserPointer(wnd))->on_mouse_move(wnd, { _0, _1 }); };
glfwSetKeyCallback(window, key_redirect);
glfwSetCursorPosCallback(window, cursor_pos_redirect);
```

- ***Function signature***

```
GLFWAPI GLFWkeyfun glfwSetKeyCallback(GLFWwindow* window, GLFWkeyfun cbfun);

/*! @brief Sets the Unicode character callback.
 *
 *  This function sets the character callback of the specified
 *  called when a Unicode character is input.
 *
 *  The character callback is intended for Unicode text input.
 *  characters, it is keyboard layout dependent, whereas the
 *  [key callback](@ref glfwSetKeyCallback) is not.  Characters
 *  to physical keys, as a key may produce zero, one or more ch
 *  want to know whether a specific physical key was pressed or
 *  the key callback instead.
 *
```

typedef void (*GLFWkeyfun)(GLFWwindow *, int, int, int, int)

The function signature for keyboard key callbacks. This is the function signature for keyboard key callback functions.

**Parameters:**

window The window that received the event.
key The [keyboard key](
scancode The system-specific scancode of the key.
action `GLFW_PRESS`, `GLFW_RELEASE` or `GLFW_REPEAT`.
mods Bit field describing which [modifier keys](

Search Online

# Performance?

- ***Isn't this slow?***