

# Allegro5 Tutorial

The background of the slide features a light gray grid pattern. Overlaid on this grid are numerous water droplets of various sizes, some with soft shadows, giving a sense of depth and texture.

The background of the slide features a light gray gradient with a subtle, large-scale circular pattern. Numerous realistic water droplets of various sizes are scattered across the surface, with some concentrated in the top-left and bottom-right corners. The text "Before we start..." is centered in a black, sans-serif font.

Before we start...

# Announcements

- You should finish installing and setting up Allegro5 on your own computer and practice the tasks before Hackathon.
- Hackathon (grading: 3%)
  - 12/19 (Sunday) 09:00-12:00 (Prof. Hu's class/Prof. Yang's class)
- Final Project Demo (grading: 17%)
  - 01/17, 18 (Mon, Tue), details will be announced one week ahead

# Announcements

- For the materials, please refer to:
- [TunchinKao/Allegro5Template \(github.com\)](https://github.com/TunchinKao/Allegro5Template)

TunchinKao / Allegro5Template Public

forked from j3soon/Allegro5Template

<> Code Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 1 tag

Go to file Add file Code

This branch is 5 commits ahead of j3soon:master.

TunchinKao feat : "Add comment on BernoulliTrail"

Exercises	Add Assets for Exercise
Final_Codes	feat : "Add comment on BernoulliTrail"
docs	Update README for 2020, remove
.gitattributes	Add .gitignore and .gitattributes
.gitignore	Add .gitignore and .gitattributes

Clone ?

HTTPS SSH GitHub CLI

git@github.com:TunchinKao/Allegro5Template

Use a password-protected SSH key.

Open with GitHub Desktop

Open with Visual Studio

Download ZIP



# A new data type - bool

- A kind of data type that can only be true(1) or false(0).
- Implemented in C++, C#, Java (boolean), Python, ...
- Allegro5 has defined its own bool data type.
- No need to include stdbool.h.

```
bool is_SR_handsome = true;  
if (is_SR_handsome) {...}
```

# Outline

- Introduction
- Display & draw image
- Events (display, timer, keyboard, mouse)
- Tips on debugging
- Tasks
- References & Tutorials

# Outline

- Introduction
- Display & draw image
- Events (display, timer, keyboard, mouse)
- Tips on debugging
- Tasks
- References & Tutorials

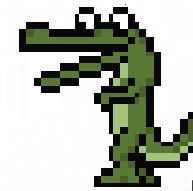
# Allegro

## (Atari Low-LEvel Game ROutines)

- Atari Low-Level Game Routines
- A software library written in C for video game development.
- Initially released in early 1990.



# Allegro5



- A cross-platform library mainly aims at video game and multimedia programming.
- Supported on Windows, Linux, Mac OSX, iPhone and Android.
- User-friendly, intuitive C API usable from C++ and many other languages.
- Hardware accelerated bitmap and graphical primitive drawing support. (via OpenGL or Direct3D)

# Outline

- Introduction
- Display & draw image
- Events (display, timer, keyboard, mouse)
- Tips on debugging
- Tasks
- References & Tutorials

# Display (Window)

```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```

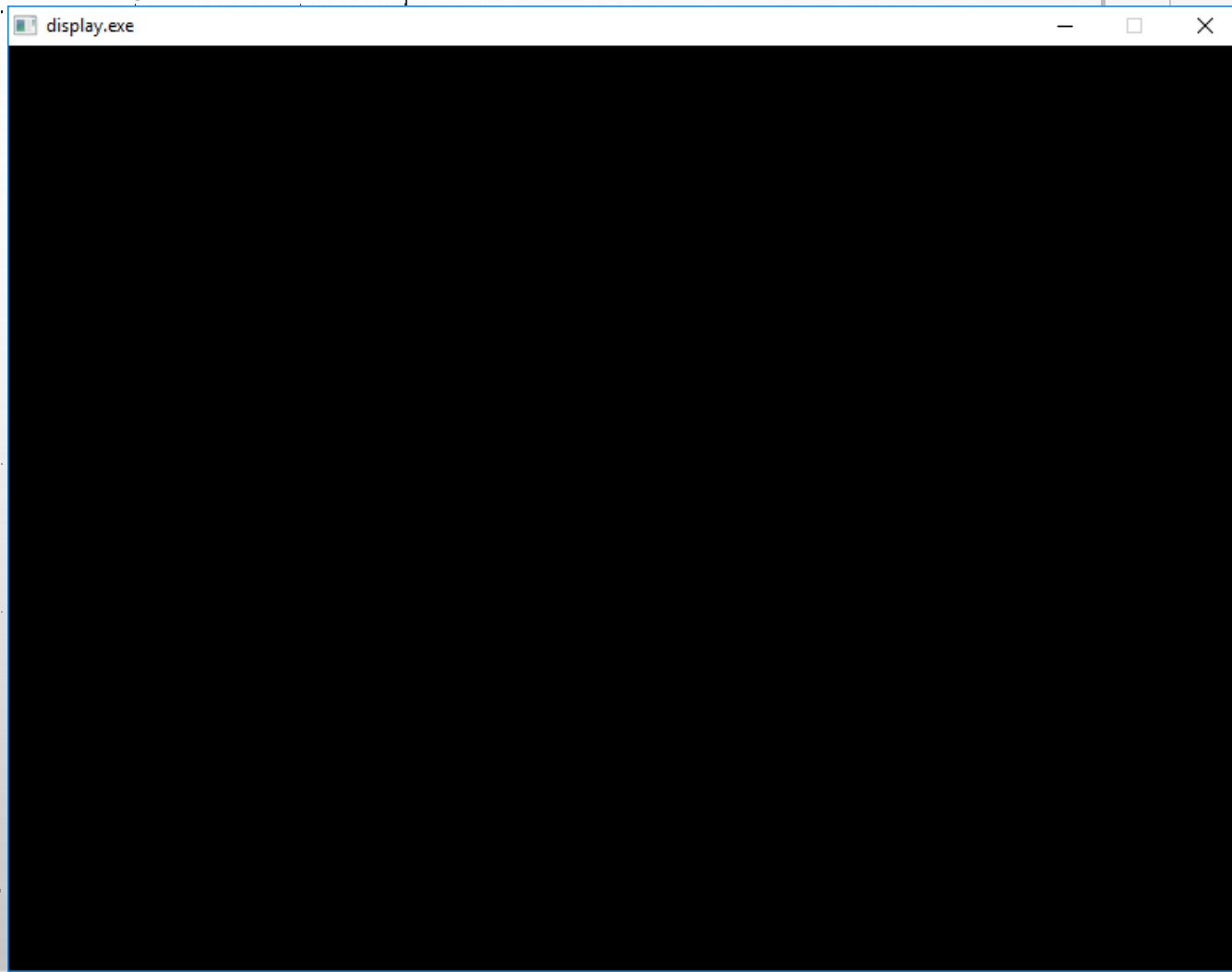
# Display (Window)

```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    → al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



# Display (Window)

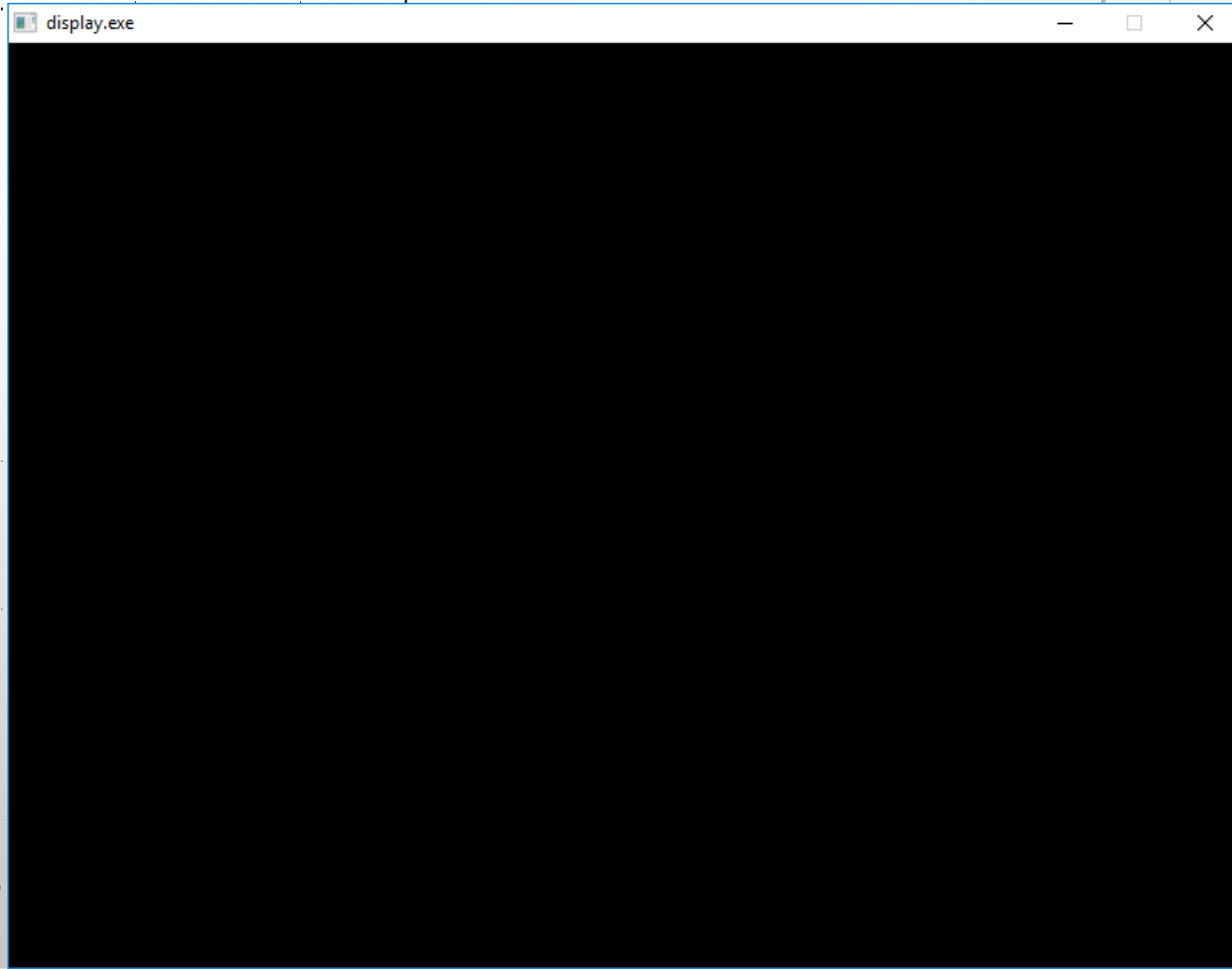
```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ➔ ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```





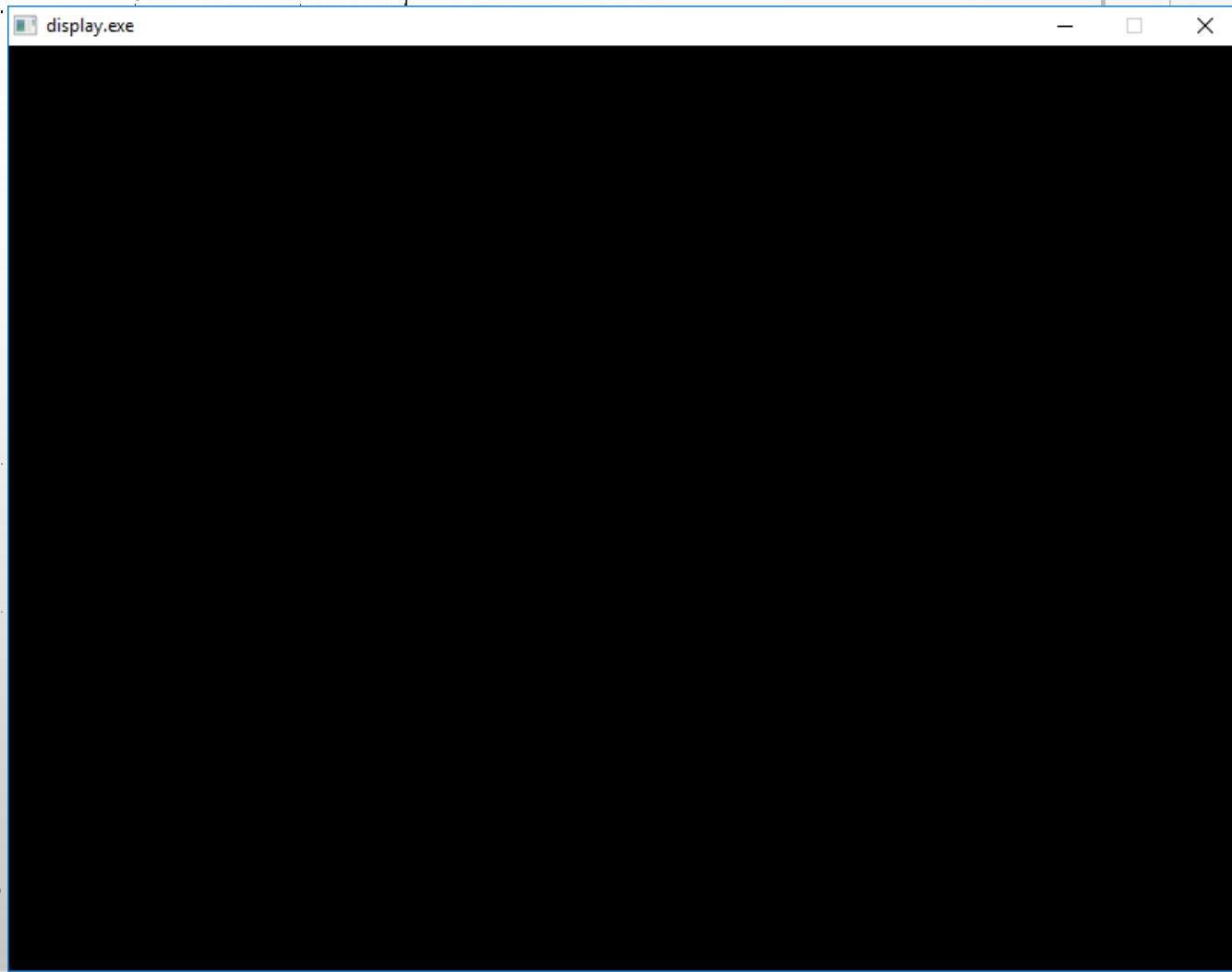
# Display (Window)

```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ➔ ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



# Display (Window)

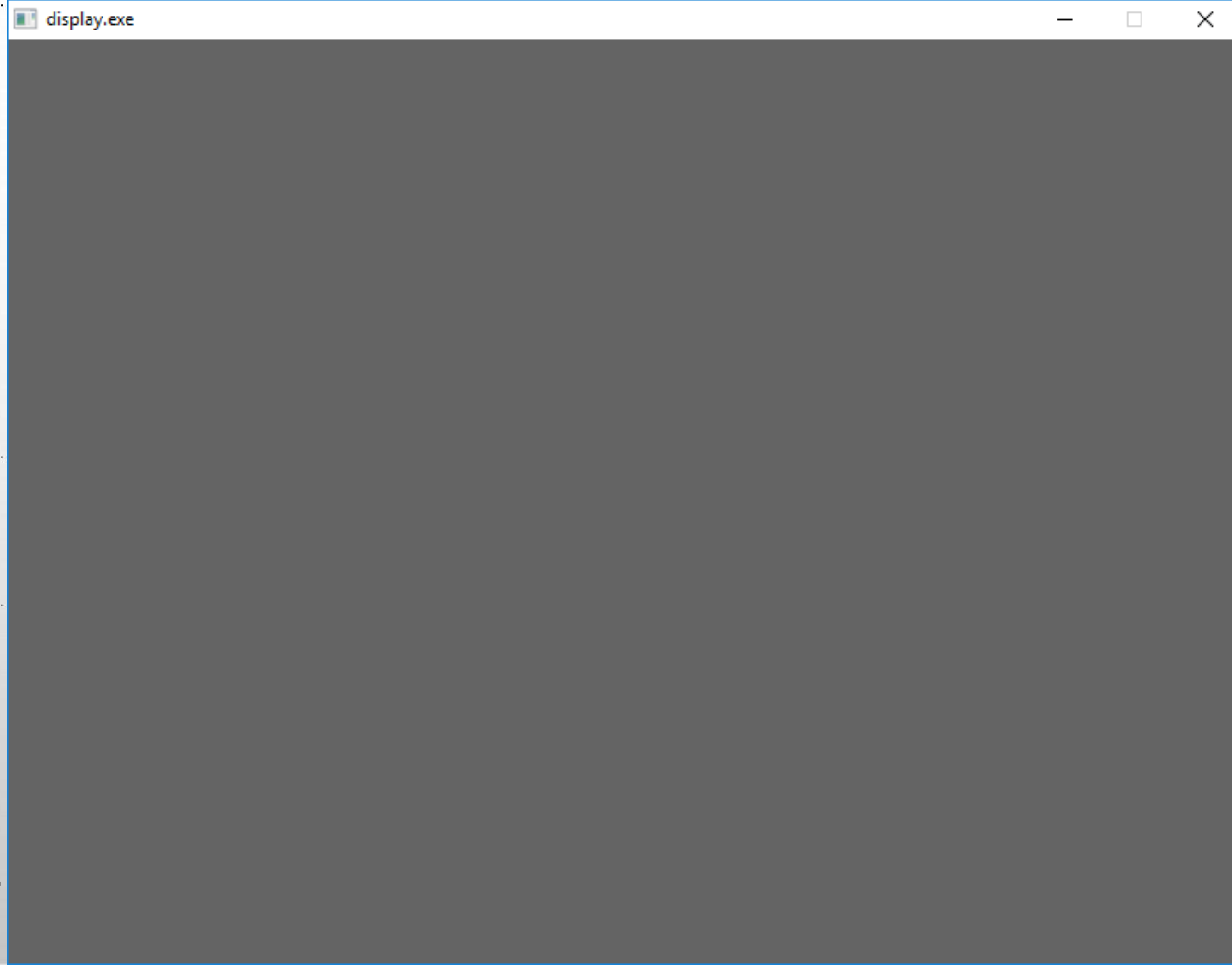
```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    ➔ al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



Buffer:

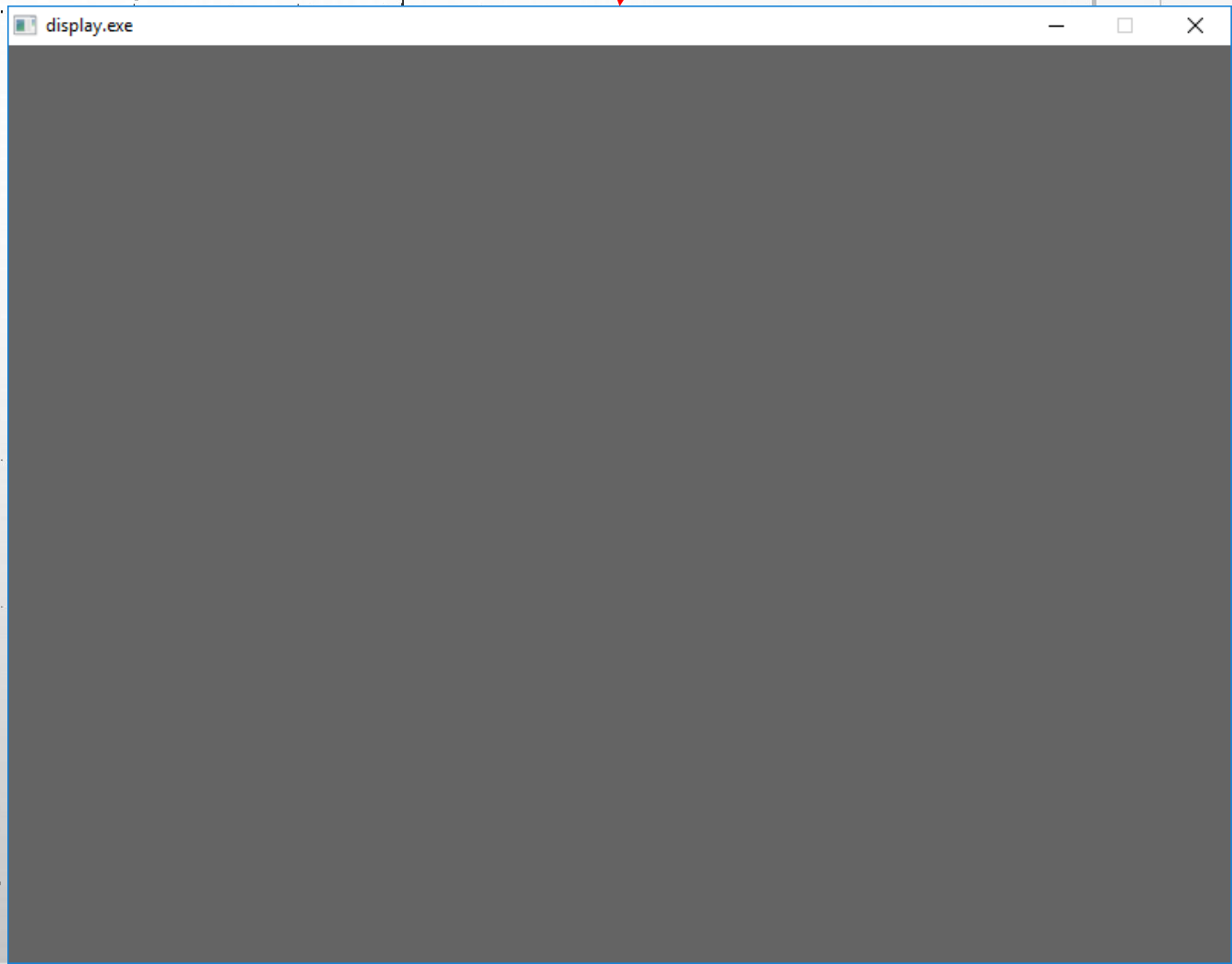
# Display (Window)

```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    → al_flip_display();
    al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



# Display (Window)

```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    ➡ al_rest(5.0);
    al_destroy_display(display);
    return 0;
}
```



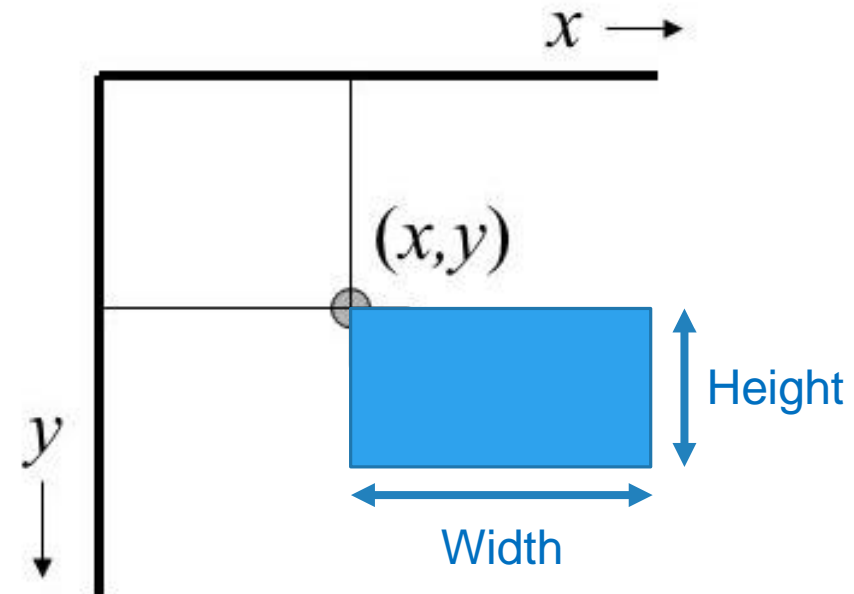
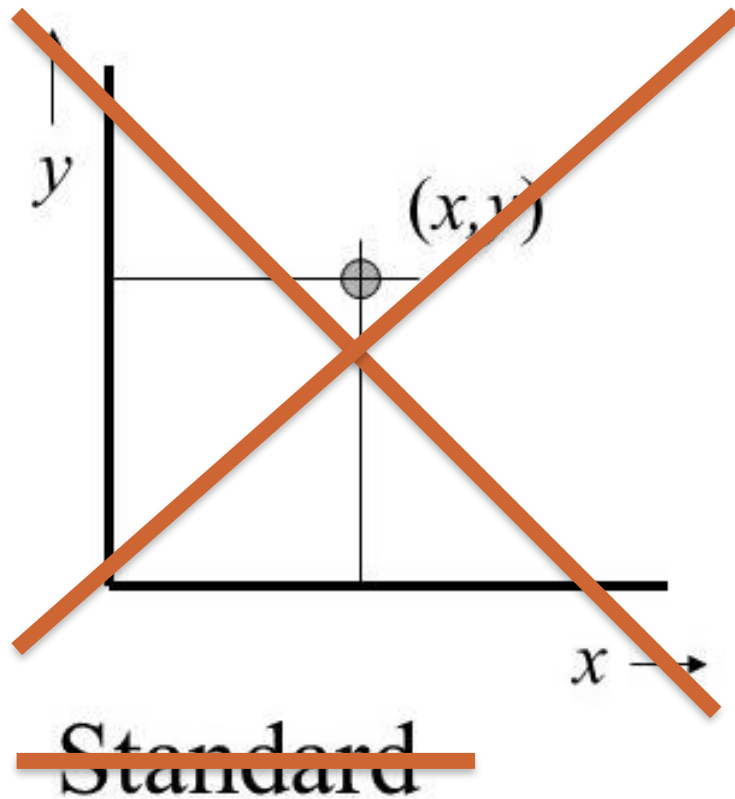
# Display (Window)

```
#include <allegro5/allegro.h>
int main(int argc, char **argv) {
    al_init();
    ALLEGRO_DISPLAY* display =
        al_create_display(800, 600);
    al_clear_to_color(
        al_map_rgb(100, 100, 100));
    al_flip_display();
    al_rest(5.0);
    ➡ al_destroy_display(display);
    return 0;
}
```



# Coordinates on Display

2D computer graphics often have the origin in the top left corner and the y-axis down the screen.



Screen (output, input)

# Image (Bitmap / Picture)

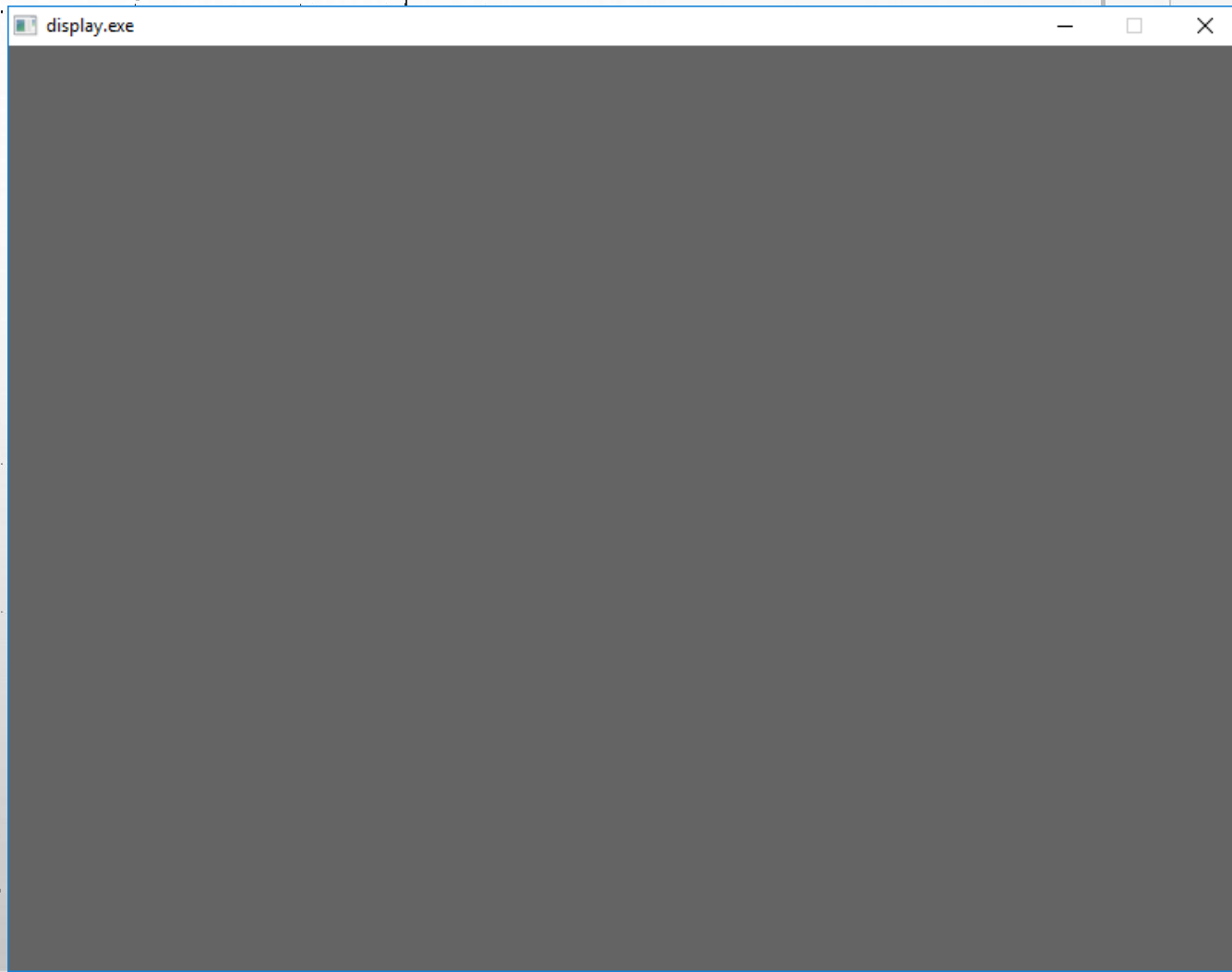
```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap("save.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

# Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    ➔ al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap("save.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

# Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ➔ ALLEGRO_BITMAP* img =
        al_load_bitmap("save.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```



img:



r/Jokes  
u/voracread • 7h

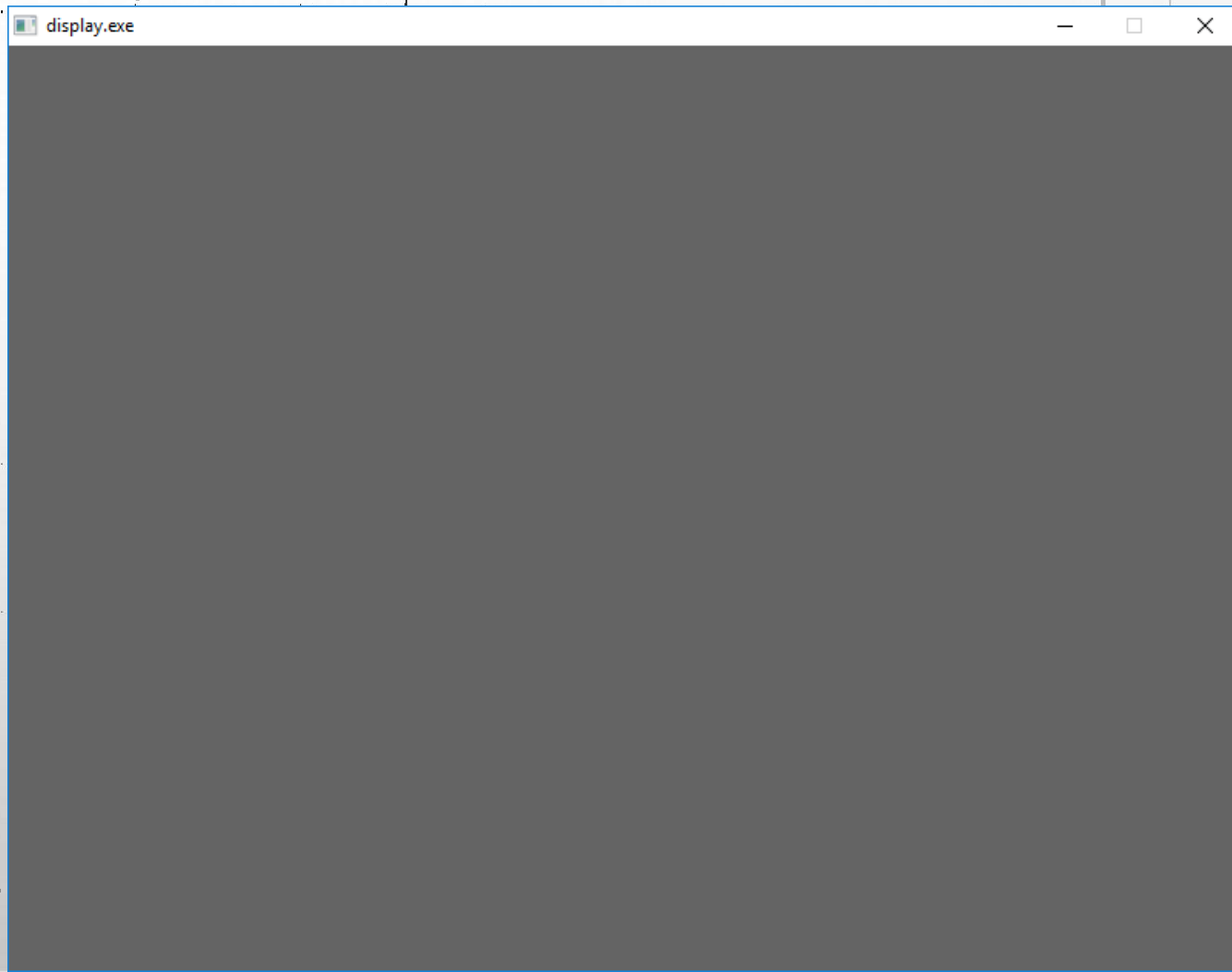
Me : God save me...

God : as jpg or png???

Buffer:

# Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap("save.png");
    ➔ al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```



img:

 r/Jokes  
u/voracread • 7h  
Me : God save me...  
God : as jpg or png???

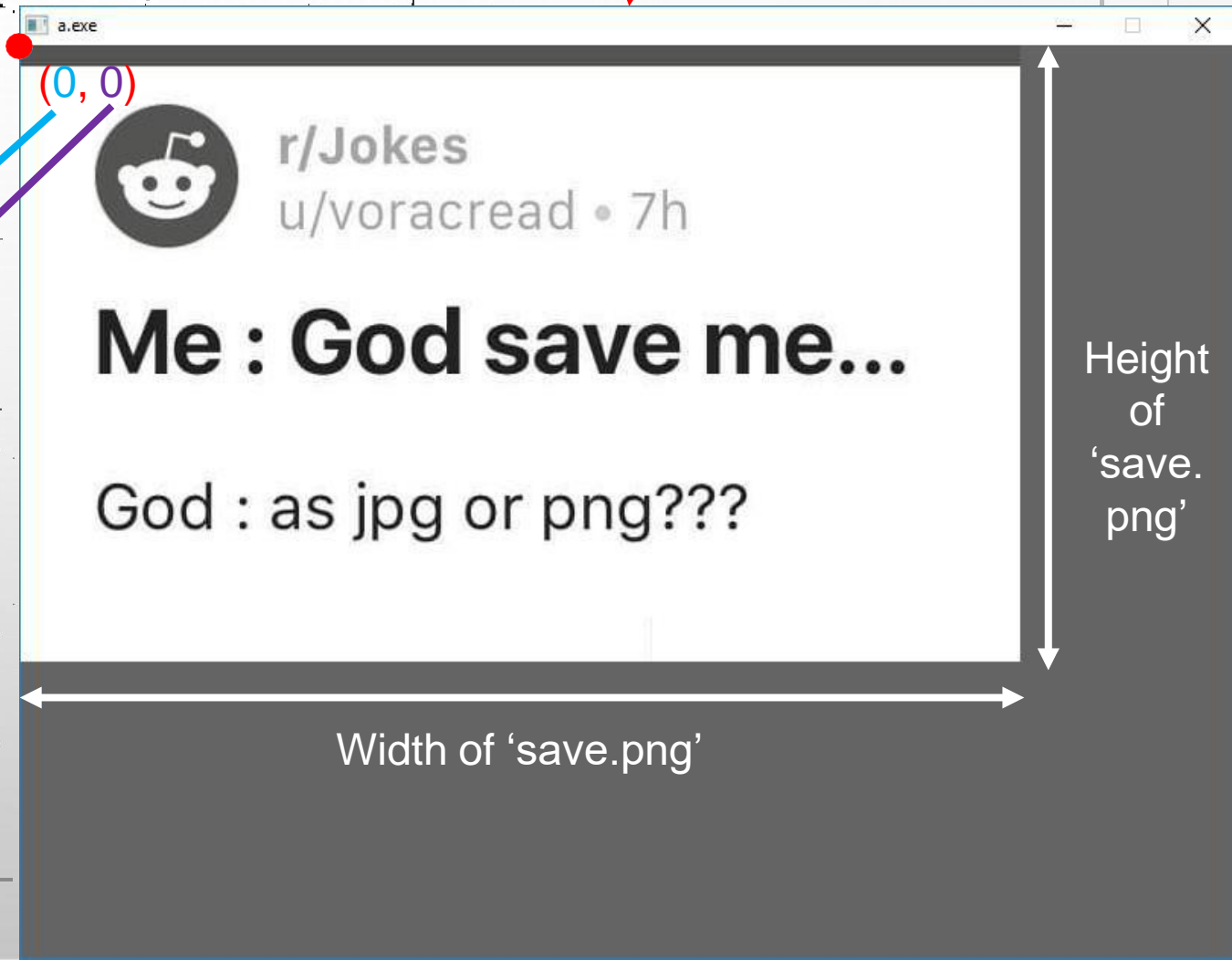
Buffer:

 r/Jokes  
u/voracread • 7h  
Me : God save me...  
God : as jpg or png???



# Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap("save.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```



img:

r/Jokes  
u/voracread • 7h  
**Me : God save me...**  
God : as jpg or png???

Buffer:

r/Jokes  
u/voracread • 7h  
**Me : God save me...**  
God : as jpg or png???

Height of  
'save.  
png'

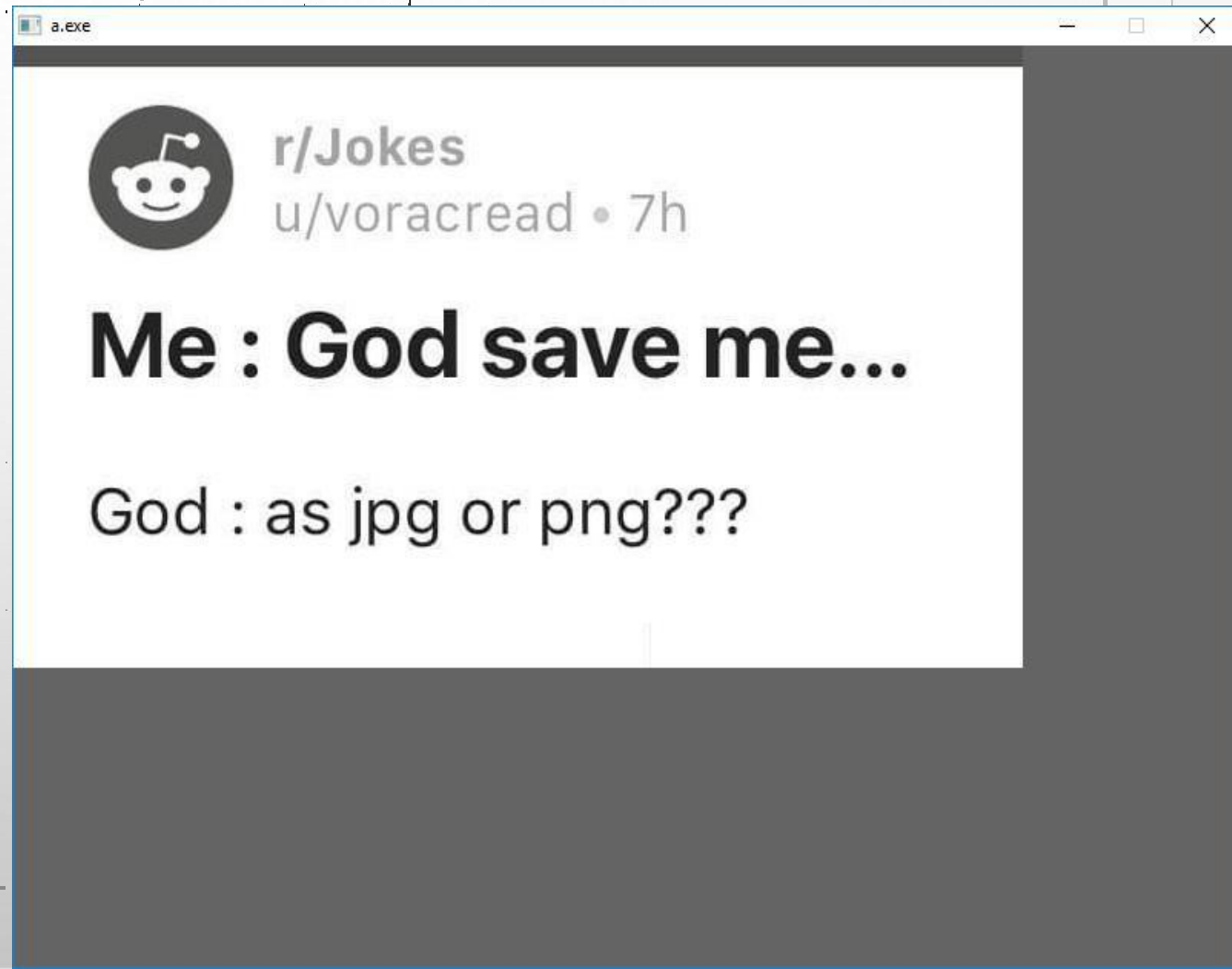
Width of 'save.png'

Buffer:



# Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap("save.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    al_destroy_bitmap(img);
    //...
    return 0;
}
```

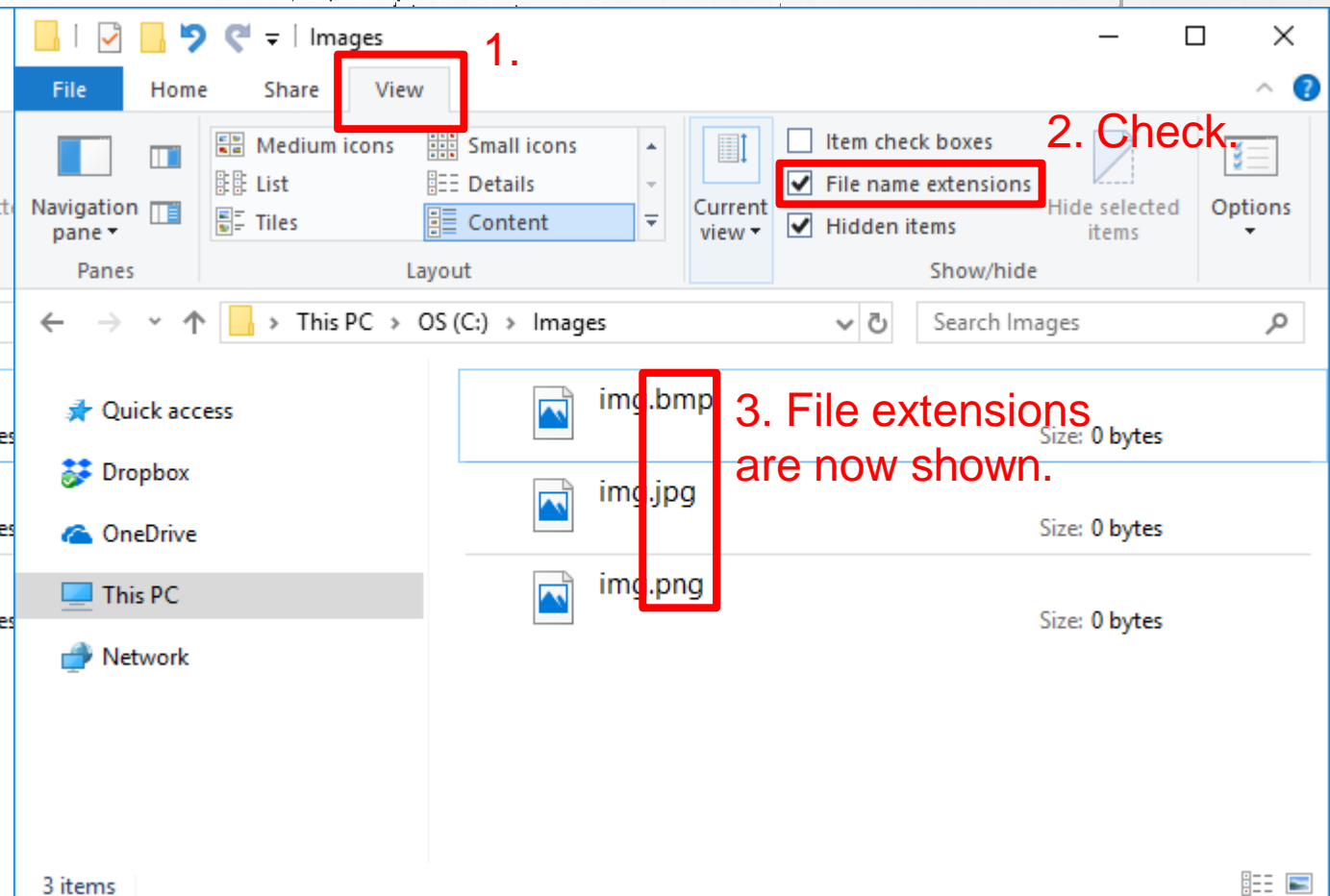
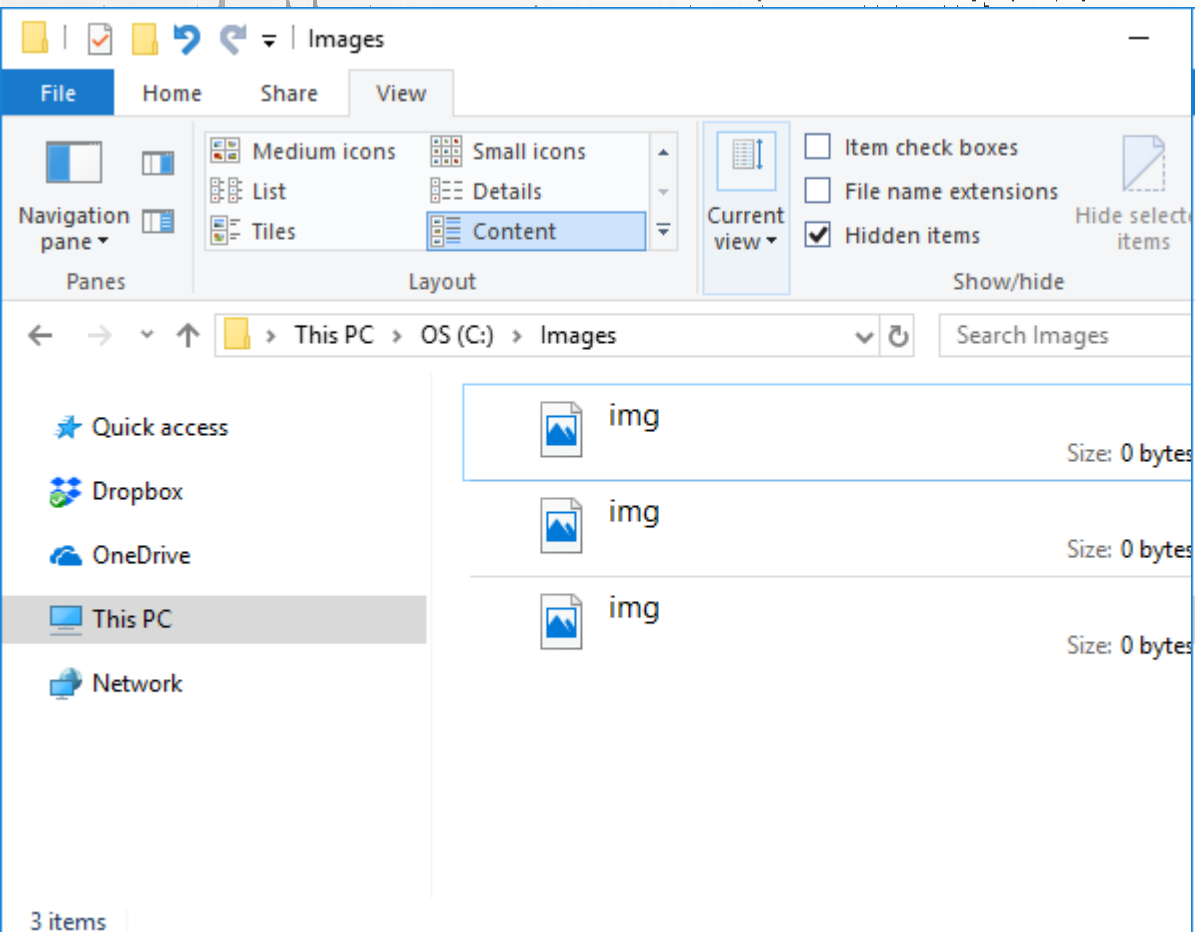


# Image (Bitmap / Picture)

```
#include <allegro5/allegro.h>
#include <allegro5/allegro_image.h>
int main(int argc, char **argv) {
    al_init();
    al_init_image_addon();
    //...
    ALLEGRO_BITMAP* img =
        al_load_bitmap("save.png");
    al_draw_bitmap(img, 0, 0, 0);
    al_flip_display();
    al_rest(5.0);
    ➡ al_destroy_bitmap(img);
    //...
    return 0;
}
```

# File Extensions

Take Windows Explorer as example.



# Others


- Font (Text / String)
- GIF
- Audio (BGM / SFX)
- Video
- ...



# Outline

- Introduction
- Display & draw image
- Events (display, timer, keyboard, mouse)
- Tips on debugging
- Tasks
- References & Tutorials

# Input? (Events?)

- Keyboard (Key down, Key up, ...)
- Mouse (Move, Button down, Button up, ...)
- Joystick
- The close button  (Alt + F4) or maybe Escape key
- Timer (Refresh display)
- Callbacks (Audio / Video finished)

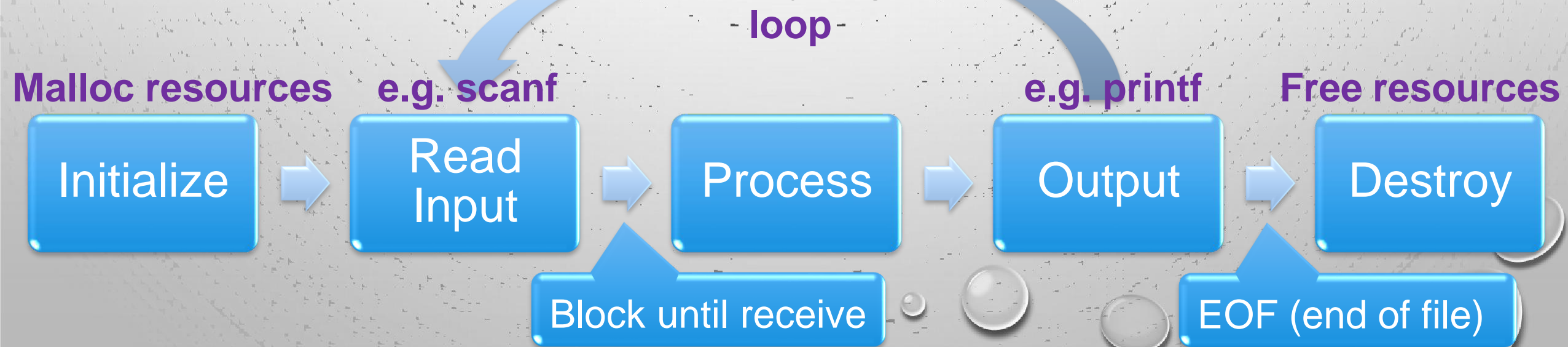
# Program Flow on OJ

- Your codes are sequential.  
(can only execute code in a specific order)
- Most of your codes on online judges:



# Program Flow on OJ

- Your codes are sequential.  
(can only execute code in a specific order)
- Most of your codes on online judges: (with multiple inputs)





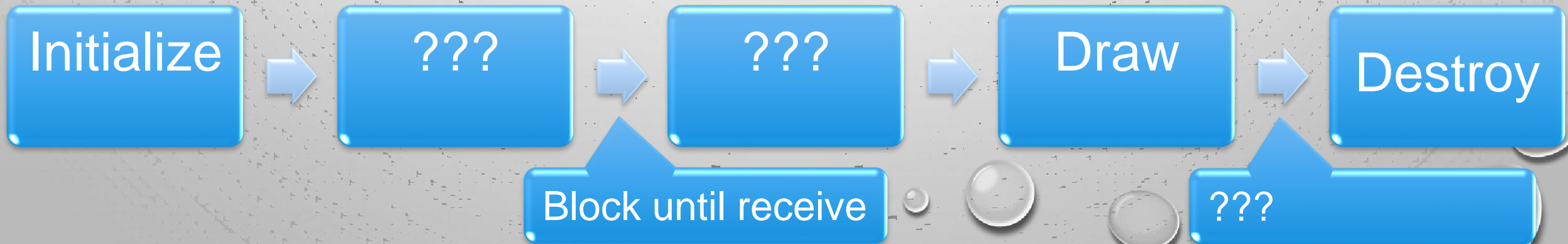
# Program Flow on Allegro5

- Your codes are still sequential.  
(can only execute code in a specific order)
- Initialize → ??? → ??? → Draw → Destroy

Initialize Allegro5, load images, malloc, ...

Update display

Free resources





# Program Flow on Allegro5

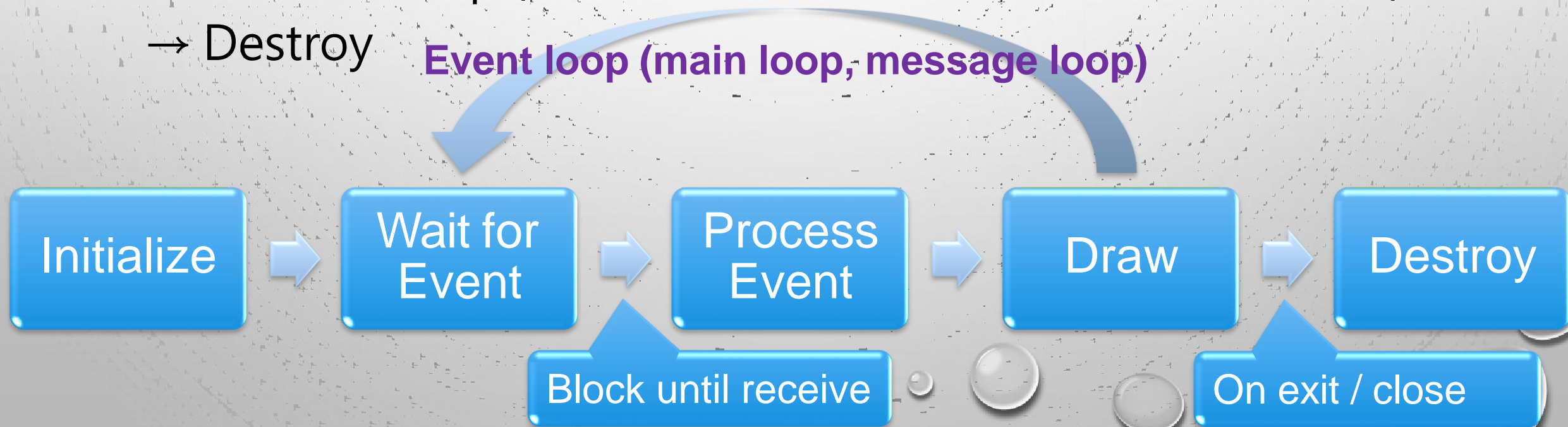
- Your codes are still sequential.
- Initialize → loop (Wait for event → Process event → Draw) → Destroy

e.g. draw signal in a certain rate (FPS (frames per second))  
e.g. keydown, mouse move



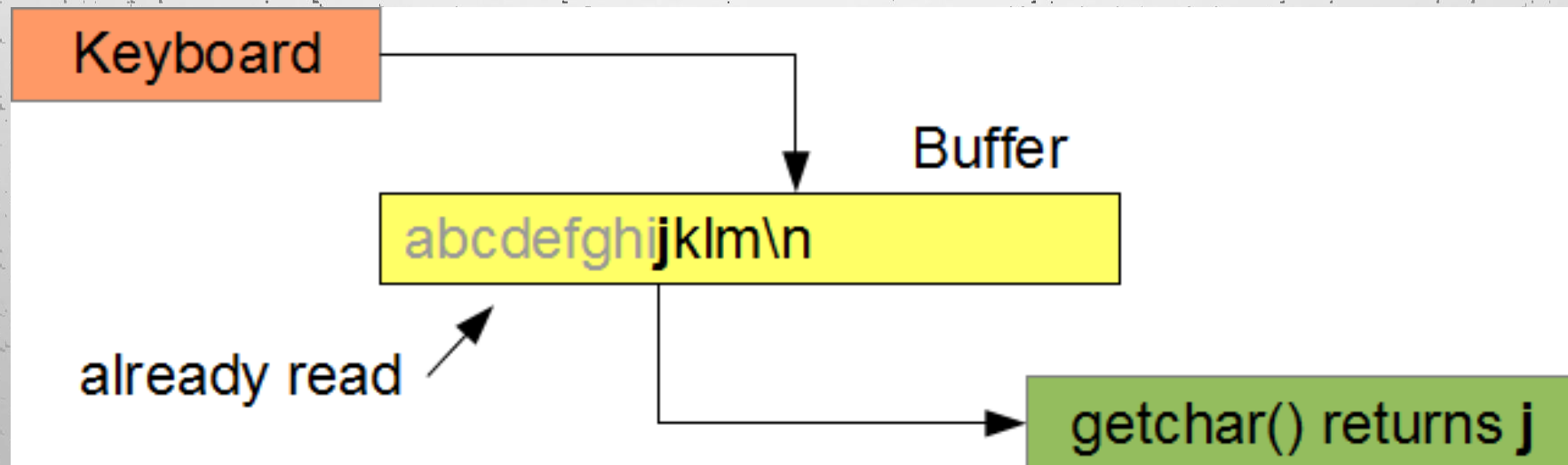
# Program Flow on Allegro5

- Your codes are still sequential.
- Initialize → loop (Wait for event → Process event → Draw) → Destroy



# Buffer used in stdin

- The buffer used in stdin can store the inputs. When the input is read by scanf, getchar, ..., the characters are removed and returned.





# Event Queue (Buffer for events)

- In an event-driven application, there is generally a **main loop** that listens for events, and then triggers a callback function when one of those events is detected.
- Used in Windows, MacOS, ...
- Most event-driven programming environments already provide this main loop.

# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:



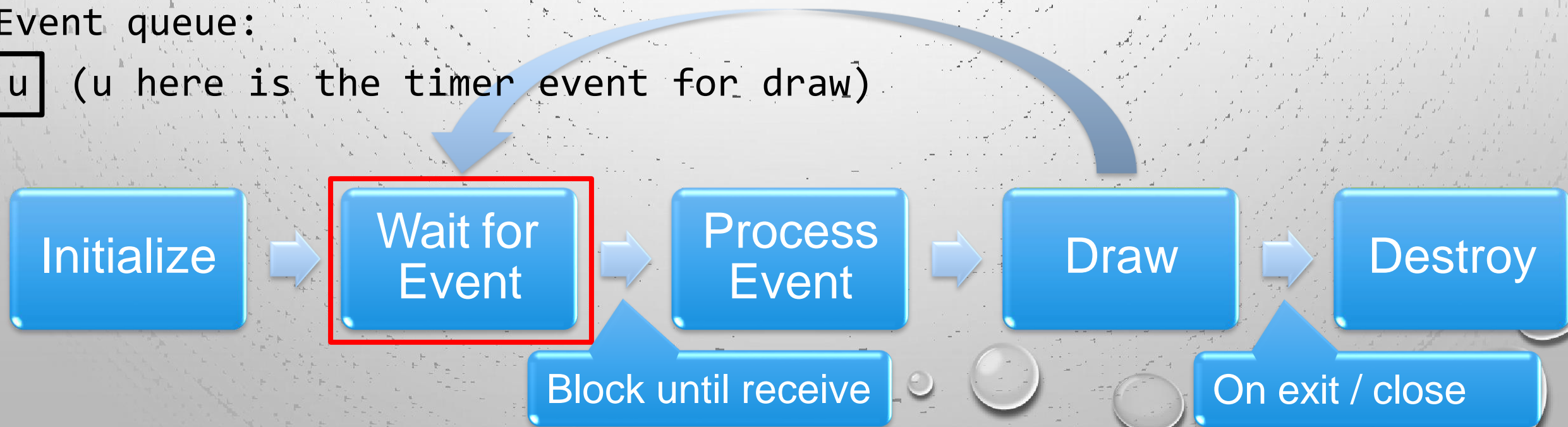


# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:

u (u here is the timer event for draw)



# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

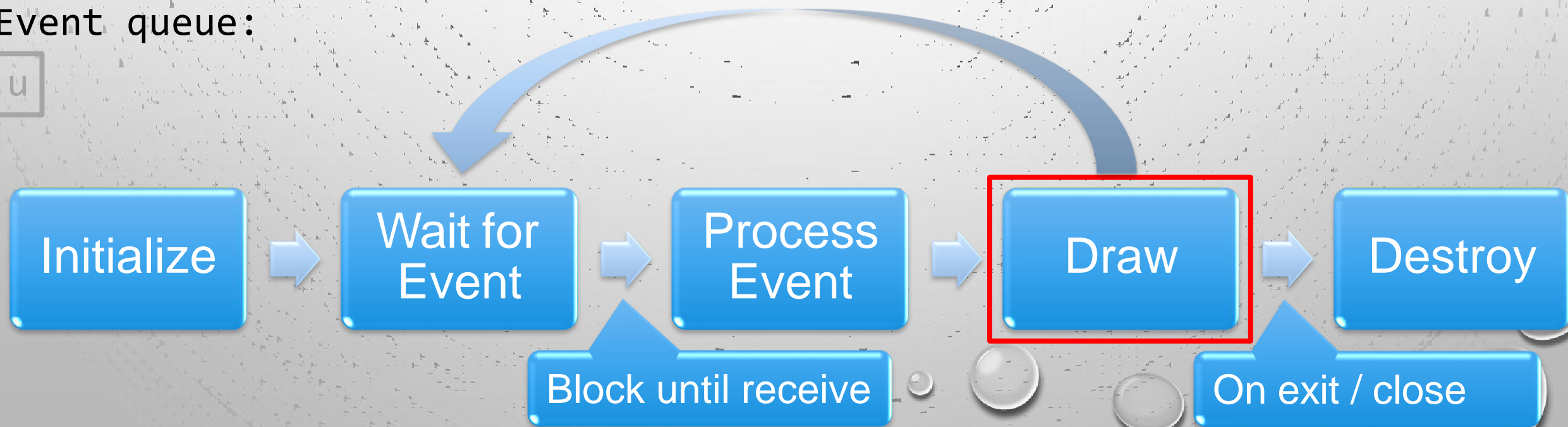
Event queue:



# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:

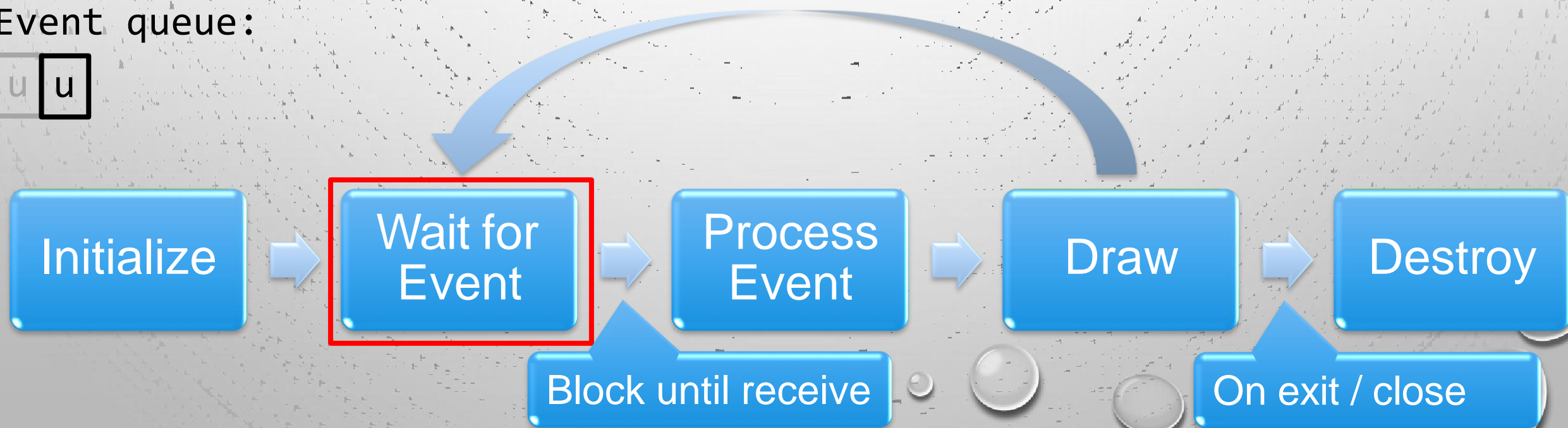




# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:



# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:





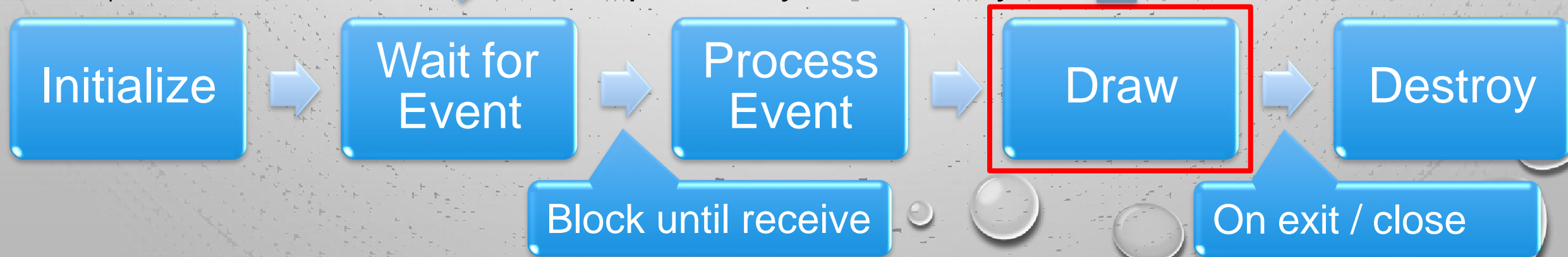
# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:



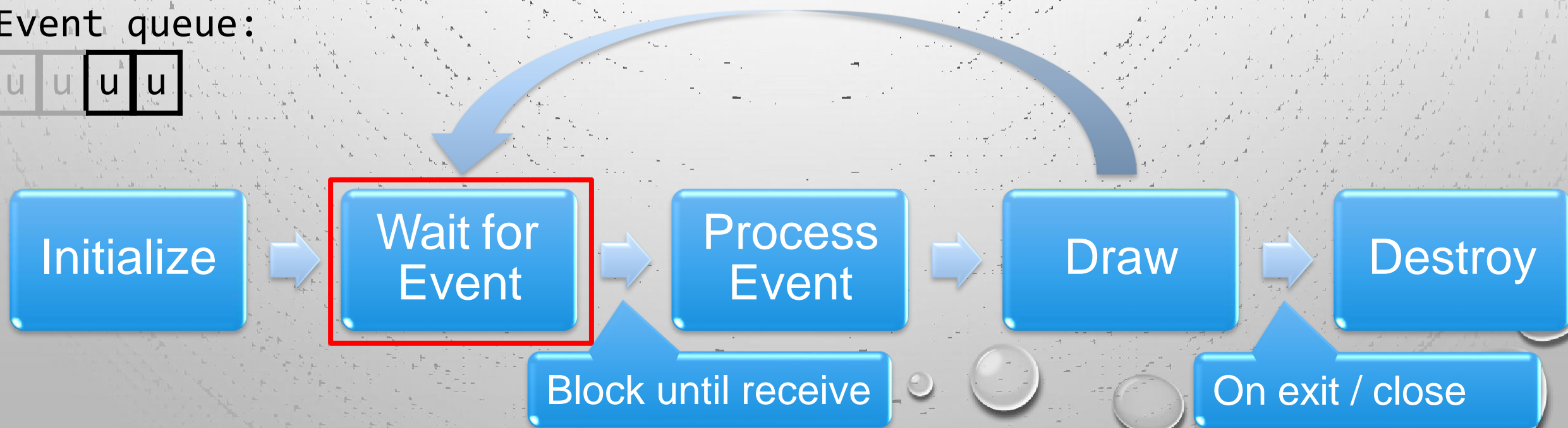
↑ Events are added to event queue asynchronously.



# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

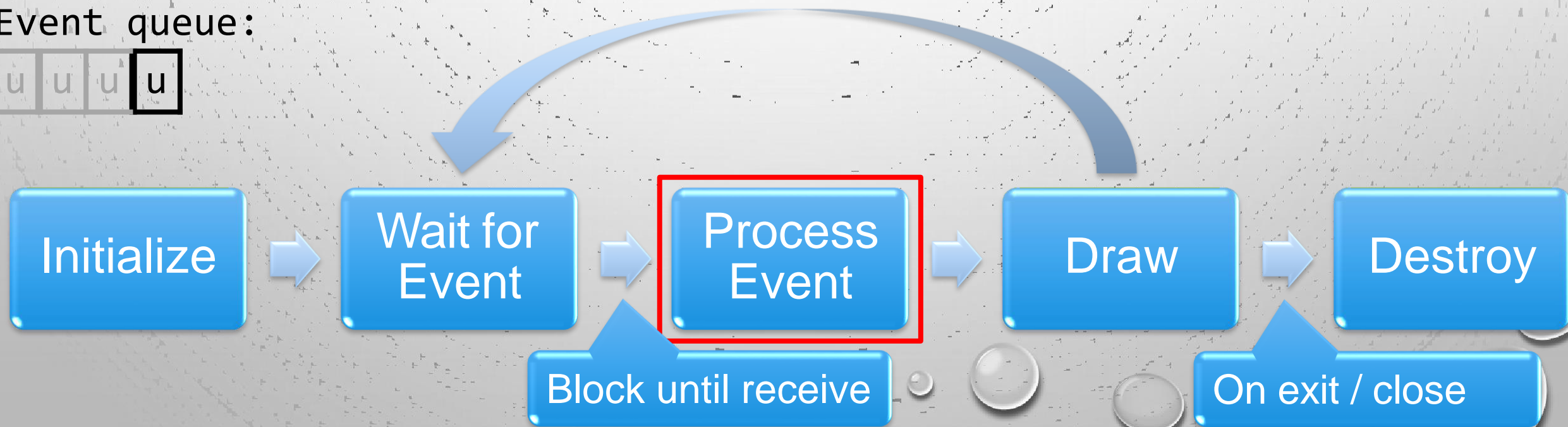
Event queue:



# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:

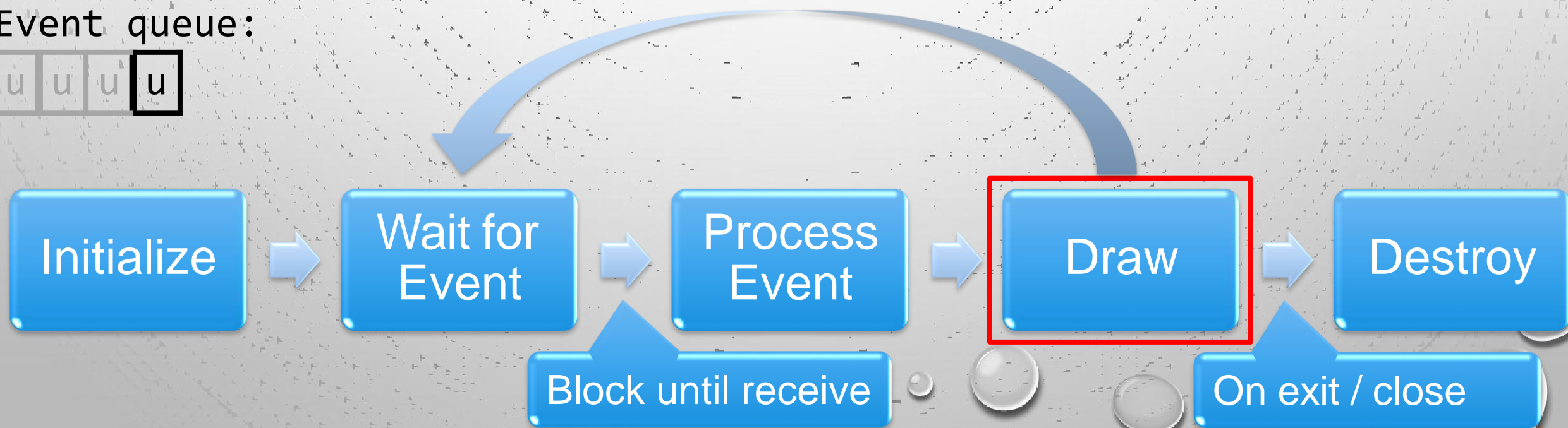




# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

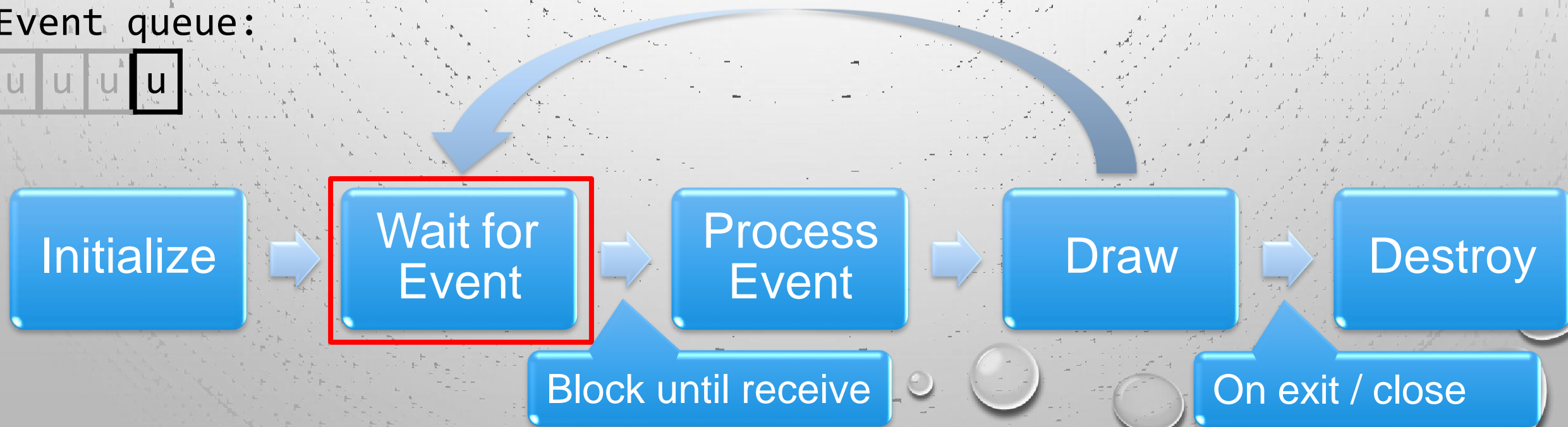
Event queue:



# Event Queue (Buffer for events)

- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:





# Event Queue (Buffer for events)

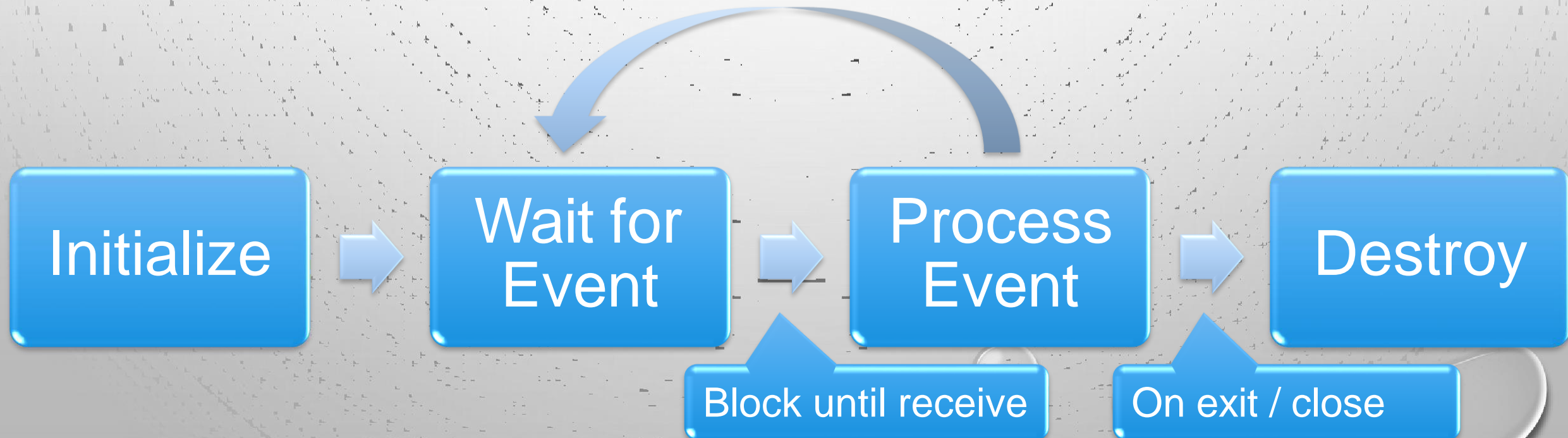
- If we have a timer that ticks for every 10ms, and an update display event is send to the queue when the timer ticks.

Event queue:

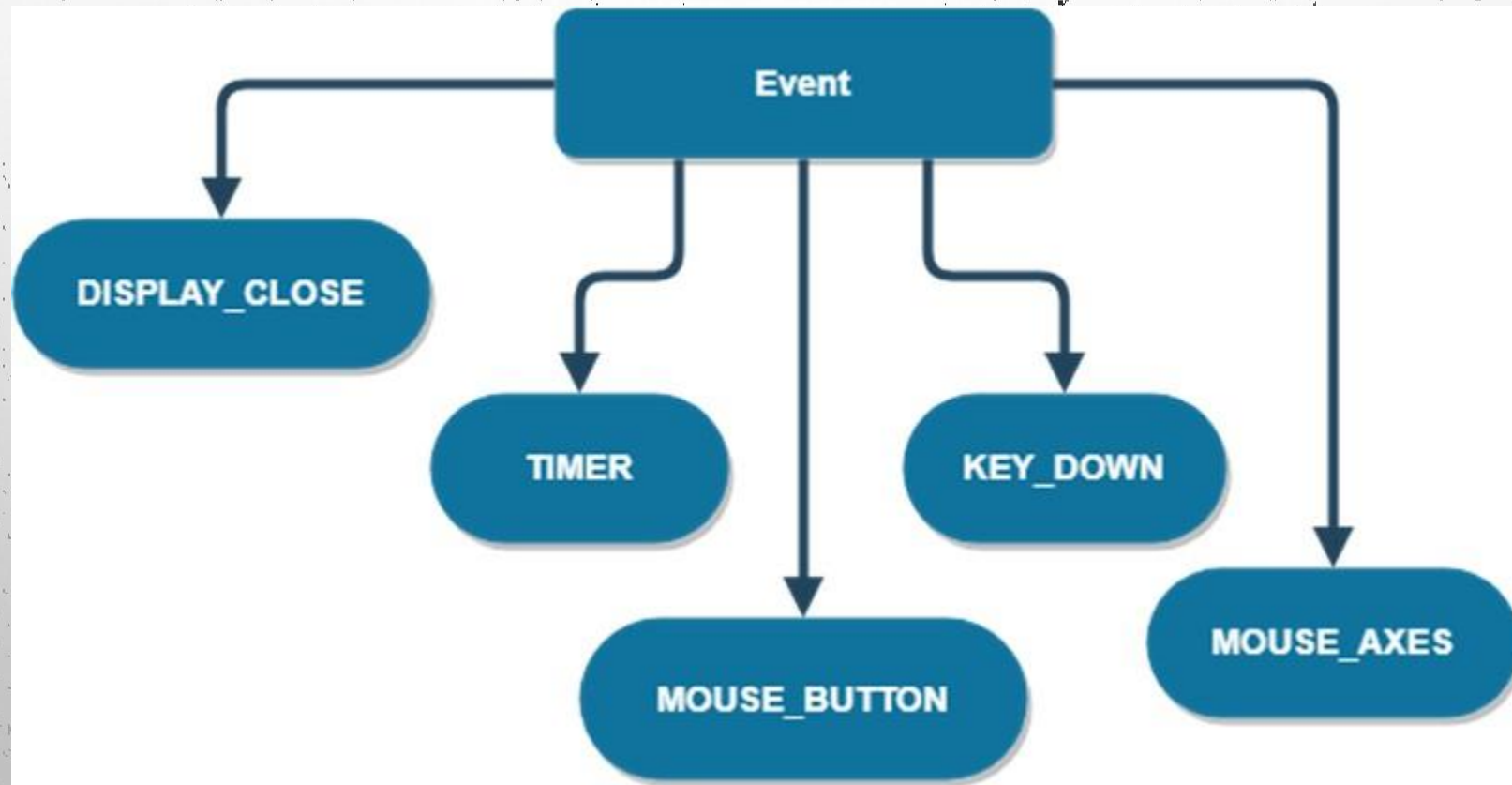


# The Generalized Program Flow

- Process event including draw, keyboard, mouse, ...



# Types of Events



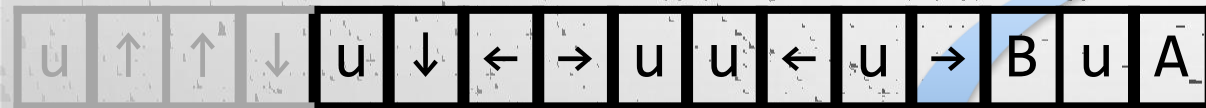


# The Generalized Program Flow

- Process event including draw, keyboard, mouse, ...

Keys pressed: ↑ ↑ ↓ ↓ ← → ← → B

A Event queue:



Initialize

Wait for  
Event

Process  
Event

Destroy

Block until receive

On exit / close

# Event Queue (Buffer for events)

```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer));
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```



# Event Queue (Buffer for events)

```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;

al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer));
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

**Initialize  
variables**

# Event Queue (Buffer for events)

```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;

al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer));
al_register_event_source(game_event_queue, al_get_keyboard_event_source());

while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

Register event sources

# Event Queue (Buffer for events)

```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer));
al_register_event_source(game_event_queue, al_get_keyboard_event_source());

while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

**Main event loop**

# Event Queue (Buffer for events)

```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer));
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event); Wait for new event
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```



# Event Queue (Buffer for events)

```
const int FPS = 30;
ALLEGRO_TIMER* game_update_timer = al_create_timer(1.0f / FPS);
ALLEGRO_EVENT_QUEUE* game_event_queue = al_create_event_queue();
bool done = false;
ALLEGRO_EVENT event;
al_register_event_source(game_event_queue, al_get_timer_event_source(game_update_timer));
al_register_event_source(game_event_queue, al_get_keyboard_event_source());
while (!done) {
    al_wait_for_event(game_event_queue, &event);
    if (event.type == ALLEGRO_EVENT_TIMER && event.timer.source == game_update_timer) {
        // Draw to display.
    } else if (event.type == ALLEGRO_EVENT_KEY_DOWN) {
        // Key pressed.
    } else if (event.type == ALLEGRO_EVENT_KEY_UP) {
        // Key released.
    } //...
}
```

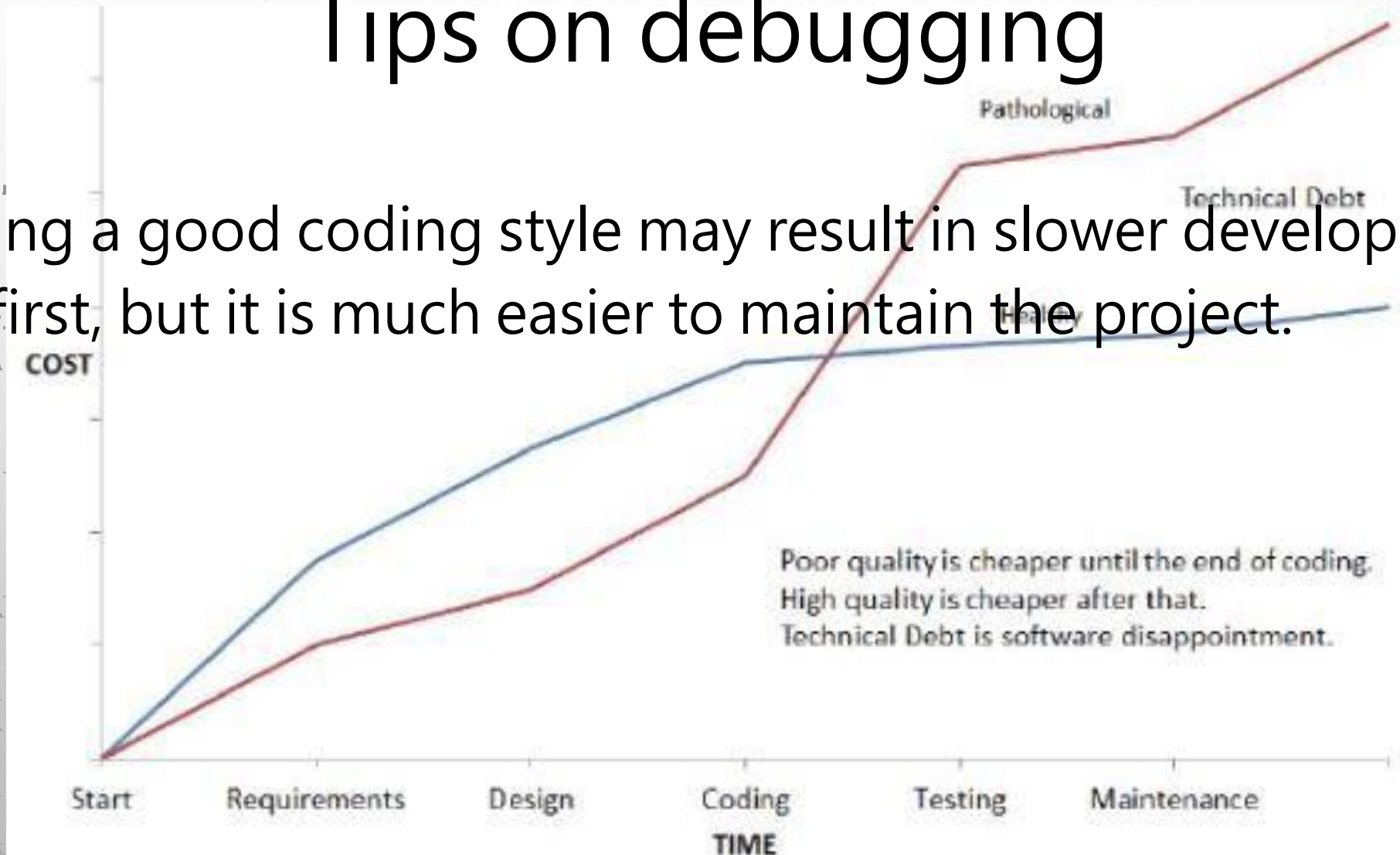
**Process Event**

# Outline

- Introduction
- Display & draw image
- Events (display, timer, keyboard, mouse)
- Tips on debugging
- Tasks
- References & Tutorials

# Tips on debugging

- Using a good coding style may result in slower development at first, but it is much easier to maintain the project.



# Tips on debugging

## (Use helper functions to log to files)

- Can be used just like printf. Both functions will automatically add a newline character at the end and save the logs to file for debugging information if the program crashes.
  - game\_abort – print error message and exit program after 2 secs.
  - game\_log – print logs.
  - LOG\_ENABLED – If not defined, game\_abort and game\_log won't do anything.

```
#define LOG_ENABLED  
void game_abort(const char* format, ...)  
void game_log(const char* format, ...)
```



# Tips on debugging (Log important events or states)

- Use game\_log every once a while. (kind of like a checkpoint)

```
int main(int argc, char **argv) {  
    allegro5_init();  
    game_log("Allegro5 initialized");  
    game_log("Game begin");  
    game_init();  
    game_log("Game initialized");  
    game_draw(); // Draw the first frame.  
    game_log("Game start event processing loop");  
    game_process_event_loop(); // This call blocks until the game is finished.  
    game_log("Game end");  
    game_destroy();  
    return 0;  
}
```

# Tips on debugging (Always check the return value)

- Check return value of functions and log if they failed. e.g.
  - malloc returns NULL if failed.
  - al\_init, al\_init\_image\_addon, ... returns false if failed.
  - al\_load\_bitmap returns NULL if failed.
    - maybe file doesn't exist, image addon is not initialized, ...
- See the API references for all function calls

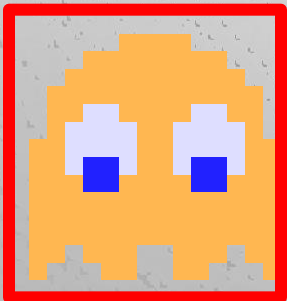
```
if (!al_init())  
    game_abort("failed to initialize allegro");
```

# Tips on debugging (Freeing the resources)

- Free resources that will not be used to avoid memory leaks.
  - malloc vs. free
  - al\_load\_bitmap vs. al\_destroy\_bitmap
- Free the resources when
  - the resources will never be used again, or
  - the program enters another state and the resource will only be used again after some time.
  - the program ends.
- Not necessary on most cases but highly recommended. letting the OS being able to allocate the block of memory to some other processes.

# Tips on debugging (Mark areas by primitive shapes)

- For character hitbox or mouse interaction, we will use collision detection frequently. Draw some primitive shapes above the character's image to indicate the region.
- When releasing the game, just comment out the definition of LOG\_ENABLED, then the primitives will not be drawn.



```
bool debug_mode = false;  
//debugging mode  
if (debug_mode) {  
    draw_hitboxes();  
}
```



# Tips on debugging (Declare constant variables)

- If some constant number is kept being used, declare it as a constant variable for better maintenance.

```
const int FPS = 60;  
const int SCREEN_W = 800;  
const int SCREEN_H = 800;  
const int GAME_TICK_CD = 64;
```

# Tips on debugging (Make duplicate codes into functions)

- e.g. when loading bitmap, there are many duplicated codes.
  - If failed to load bitmap, output failed message and abort.
  - If success, log the success action.

```
// Load bitmap and check if failed.  
ALLEGRO_BITMAP* load_bitmap(const char* filename) {  
    ALLEGRO_BITMAP* bmp = al_load_bitmap(filename);  
    if (bmp == NULL)  
        game_abort("failed to load image: %s", filename);  
    else  
        game_log("loaded image: %s", filename);  
    return bmp;  
}
```

# Tips on debugging

## (Make repeat variable groups into struct)

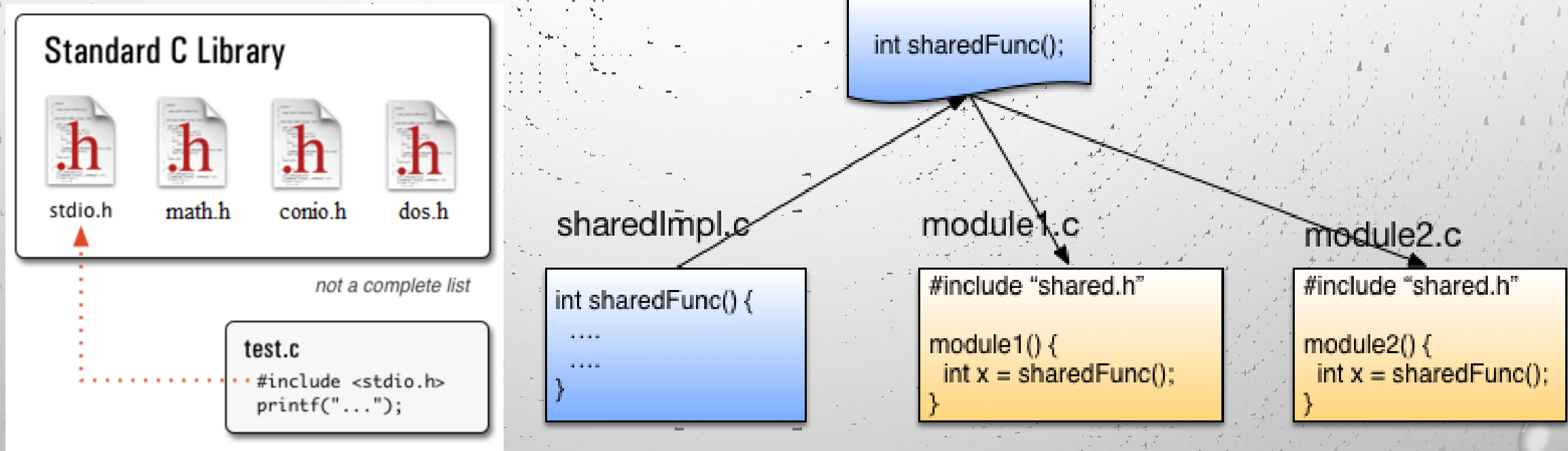
- e.g. objects (both self & enemy & bullets) will usually have the same variable groups.
  - The x, y coordinates on the display.
  - The velocity vx, vy for updating x, y coordinates.
  - Width and height of the object.  
(AABB box collision)
  - Image for drawing the object.
  - More...

```
typedef struct object {  
    Pair_IntInt Coord; //  
    Pair_IntInt Size; // x f  
    Directions facing;  
    Directions preMove;  
    Directions nextTryMove;  
    uint32_t moveCD;  
} object;
```



# Tips on debugging (Store source codes in different files)

- Header (\*.h), Source code (\*.c)



Source: <https://www.quora.com/What-is-a-header-file-and-its-use-in-C-program-Also-tell-me-what-does-function-mean-in-c-programming>

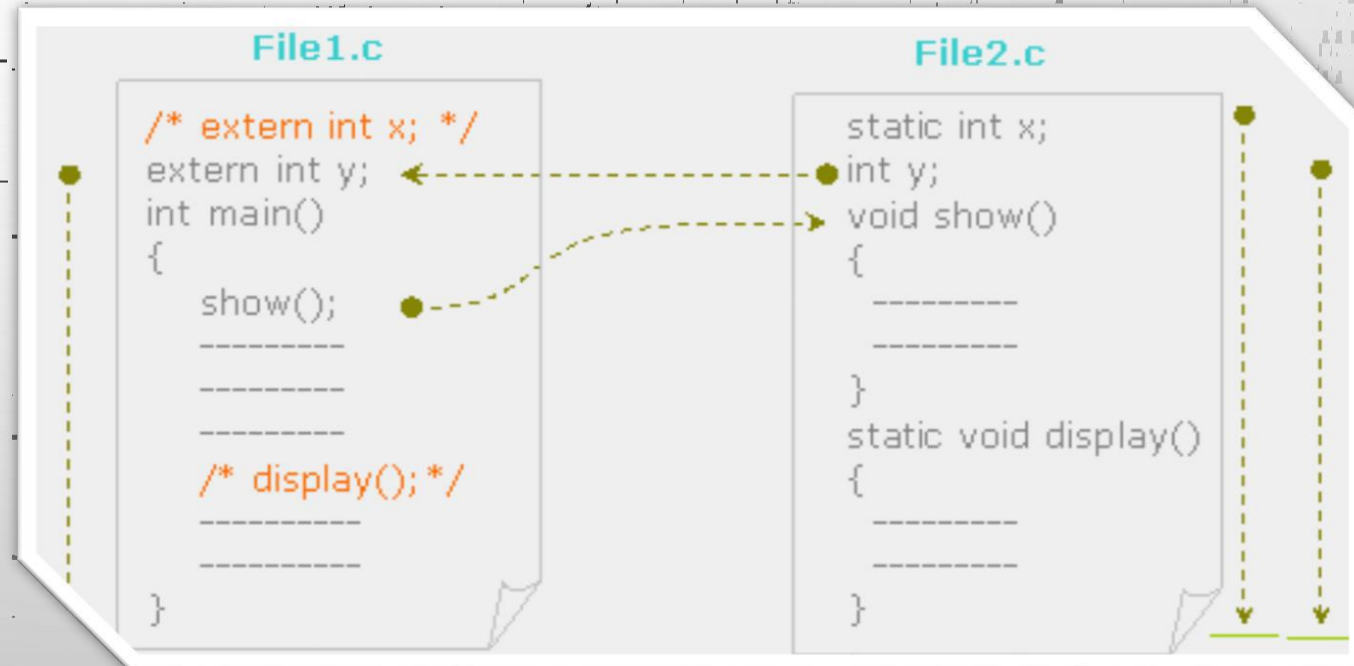
Source: <http://hanxue-it.blogspot.com/2014/04/why-include-c-implementation-code-in.html>



# Tips on debugging

## (Store source codes in different files)

- Extern in (\*.h), make variables exposed to other files that includes the (\*.h) file.
- Static in (\*.c), only visible within the file. Variables or functions with the same name but in different files are considered different.



# Outline

- Introduction
- Display & draw image
- Events (display, timer, keyboard, mouse)
- Tips on debugging
- Tasks
- References & Tutorials

# Tasks (Practice only)

- Task 1 – Blank window.
- Task 2 – Draw images and texts.
- Task 3 – Implement event loop and quit when the close button is clicked.
- Task 4 – Using keyboard.
- Task 5 – Using mouse.

# Outline

- Introduction
- Display & draw image
- Events (display, timer, keyboard, mouse)
- Tips on debugging
- Tasks
- References & Tutorials



# References

- Allegro 5 Wiki  
<https://www.allegro.cc/manual/5/>  
[https://wiki.allegro.cc/index.php?title=Allegro\\_5\\_API\\_Tutorials](https://wiki.allegro.cc/index.php?title=Allegro_5_API_Tutorials)
- Allegro 5 reference manual  
<https://liballeg.org/a5docs/trunk/>
- Allegro5 examples on GitHub  
<https://github.com/liballeg/allegro5/tree/master/examples>

# Tutorials

- C++ Allegro 5 Made Easy  
<https://www.youtube.com/watch?v=IZ2krJ8Ls2A&list=PL6B459AAE1642C8B4>
- 2D Game Development Course  
<http://fixbyproximity.com/2d-game-development-course/>
- Allegro Game Library Tutorial Series  
<https://www.gamefromscratch.com/page/Allegro-Tutorial-Series.aspx>



Questions?