# CSC309: Introduction to Web Programming

## Lecture 5

*Wael Aboulsaadat*

# Command-line driven vs. event-driven

Command-line model (e.g., UNIX shell, DOS)

- Interaction controlled by system
- User queried when input is needed

Event-driven model (e.g., GUIs)

- Interaction controlled by the user
- System waits for user actions and then reacts
- More complicated programming and architecture

# Events

User input is modeled as “events” that must be handled by the system

Examples?

- Mouse

  button down, button up, button clicked, entered, exited, moved, dragged

- Keyboard

  key down, key up, key pressed

- Window

  movement, resizing

# Anatomy of an Event

An event encapsulates the information needed for handlers to react to the input

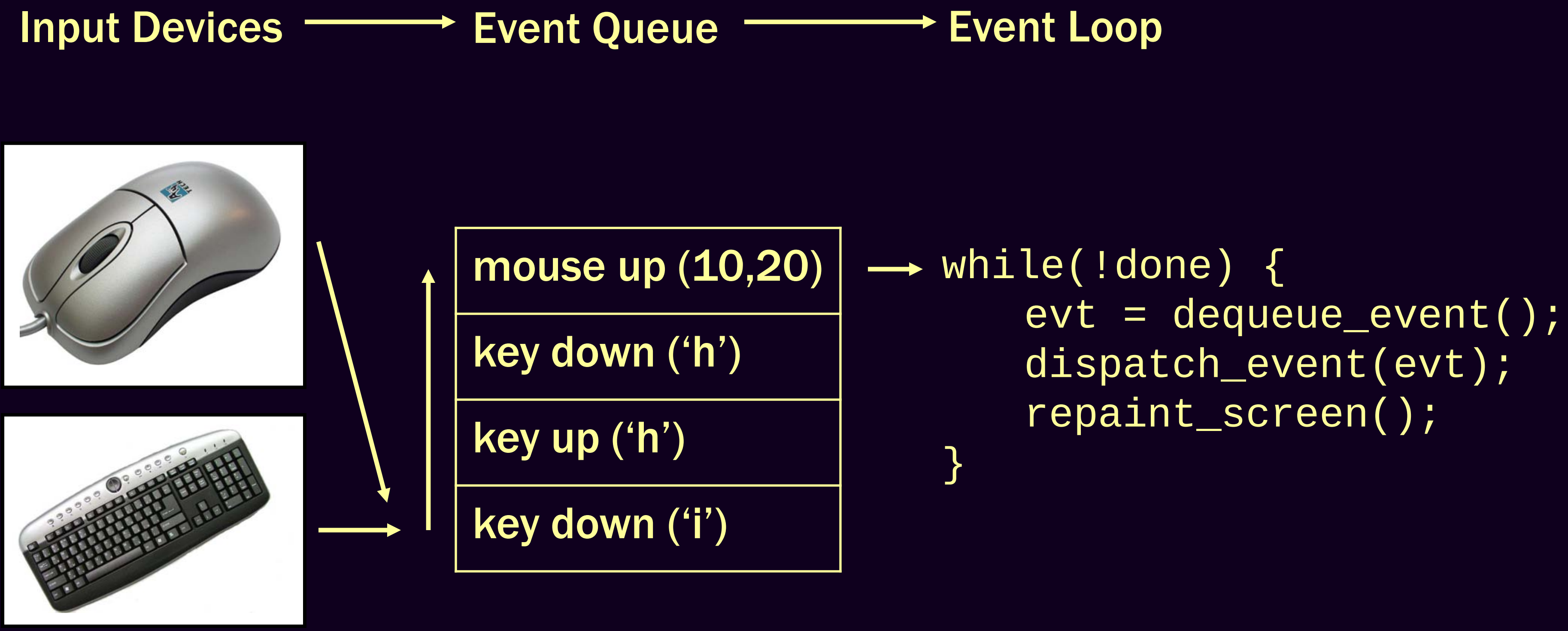
- Event type (mouse button down, key up, etc.)
- Event target (component in which event occurred)
- Timestamp
- Modifiers (Ctrl, Shift, Alt, etc.)
- Type-specific content
  - Mouse: x,y coordinates, # clicks
  - Keyboard: key code

# Event Handlers

Events are dispatched to components

- Application developers can specify code to be executed when the event occurs (callbacks)
- Built-in components will have code to handle most keyboard and mouse events
  - Buttons handle mouse up/down to change graphic
  - Text boxes update their contents on key press
- Built-in components often generate new “high-level” events from combinations of low-level events
  - Text boxes generate “change” events when contents changes and focus is lost
  - Sliders generate “change” events when thumb is dragged

# Event Loop



Input Devices → Event Queue → Event Loop

mouse up (10,20)

key down (‘h’)

key up (‘h’)

key down (‘i’)

```
while(!done) {
    evt = dequeue_event();
    dispatch_event(evt);
    repaint_screen();
}
```

Exists in every application

Usually handled for you by UI framework

# Event Loop

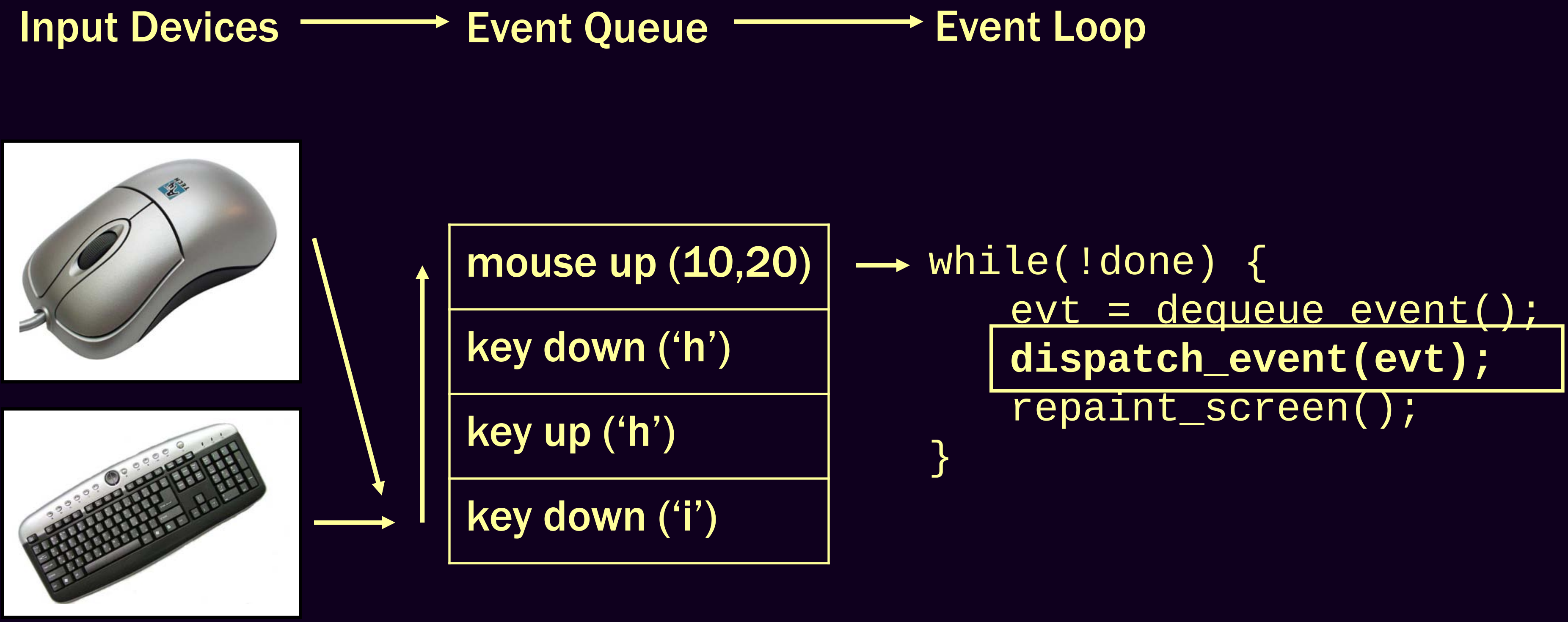
Input Devices → Event Queue → Event Loop

| |
|---|
| mouse up (10,20) |
| key down (‘h’) |
| key up (‘h’) |
| key down (‘i’) |

```
while(!done) {
    evt = dequeue_event();
    dispatch_event(evt);
    repaint_screen();
}
```

Blocks until an event arrives

# Event Loop



Input Devices → Event Queue → Event Loop

mouse up (10,20)

key down (‘h’)

key up (‘h’)

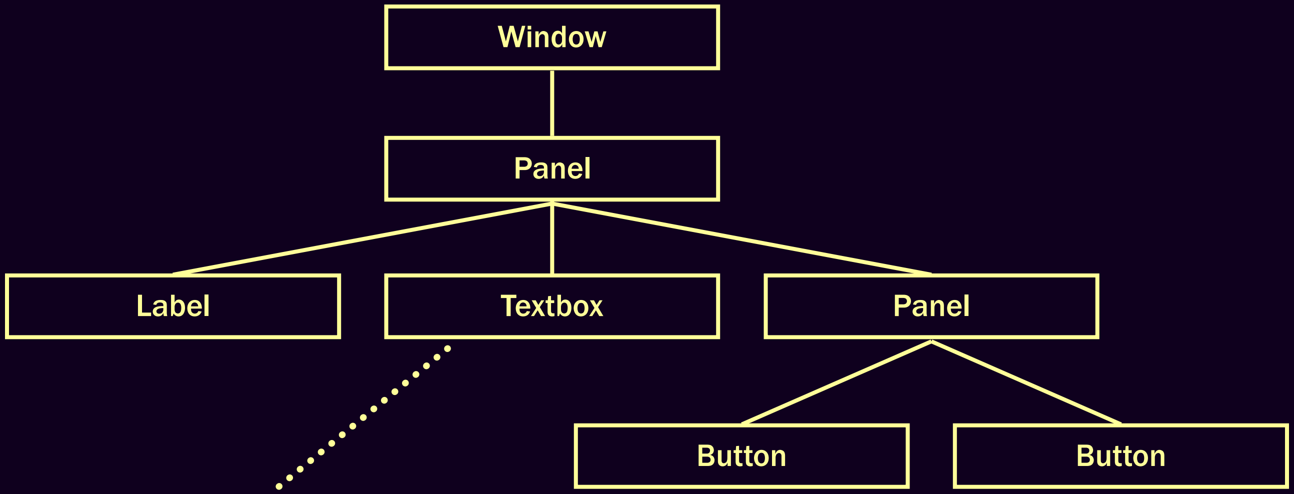
key down (‘i’)

```
while(!done) {
    evt = dequeue_event();
    dispatch_event(evt);
    repaint_screen();
}
```

Most of the work happens here

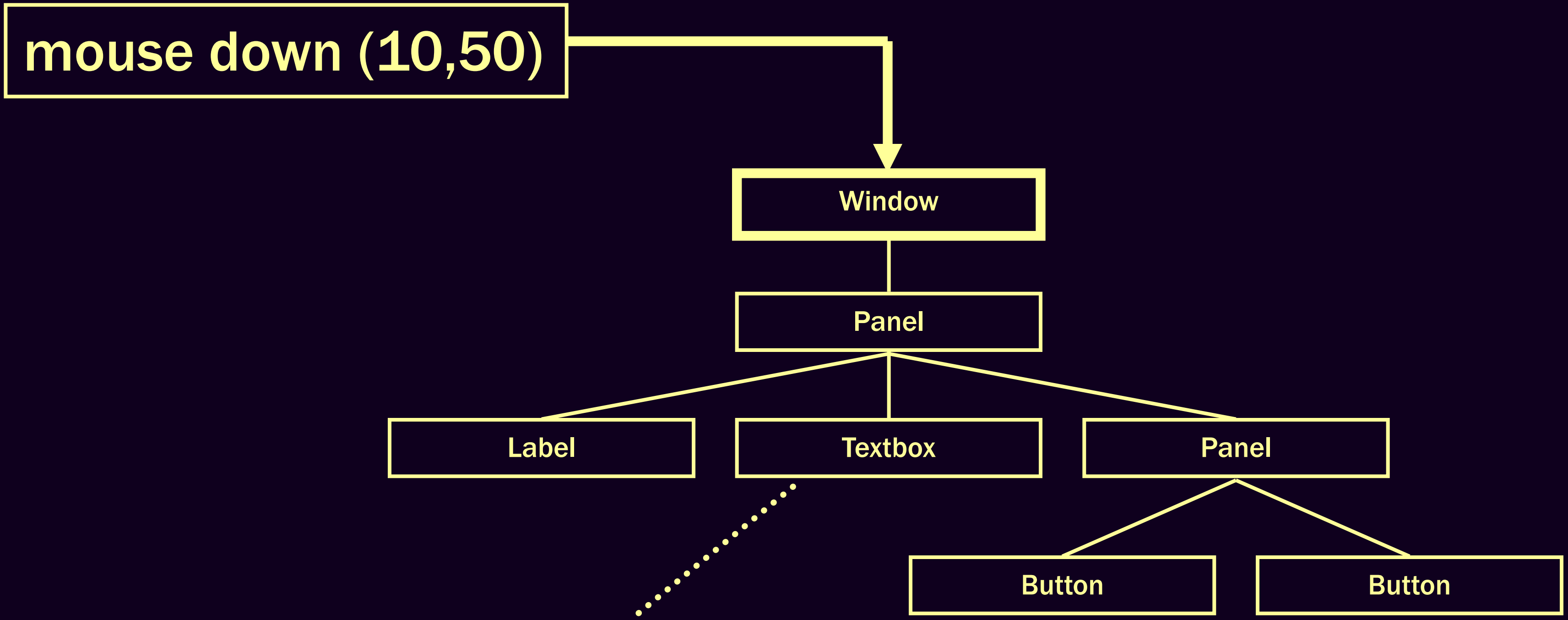
# Dispatching Events

**mouse down (10,50)**



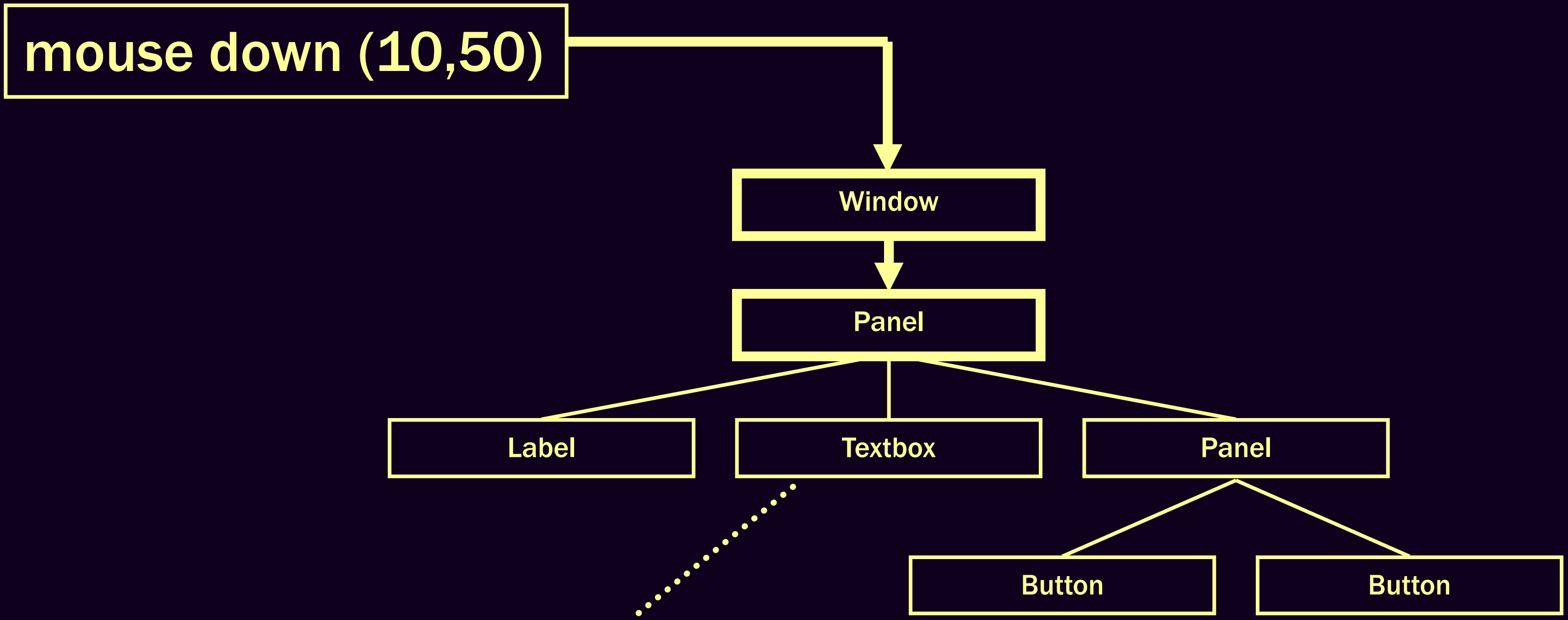
```
function onMouseDown(evt) {
    // do something...
}
```

# Dispatching Events



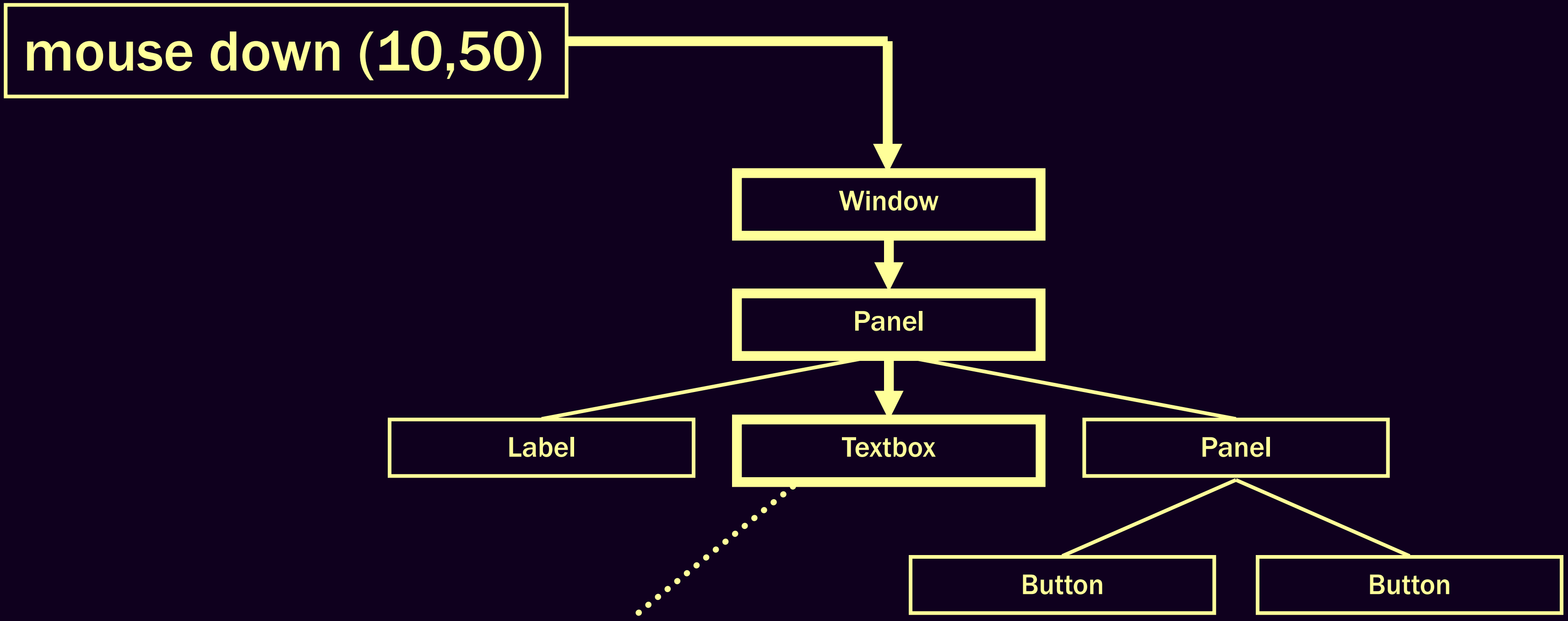
```
function onMouseDown(evt) {
    // do something...
}
```

# Dispatching Events



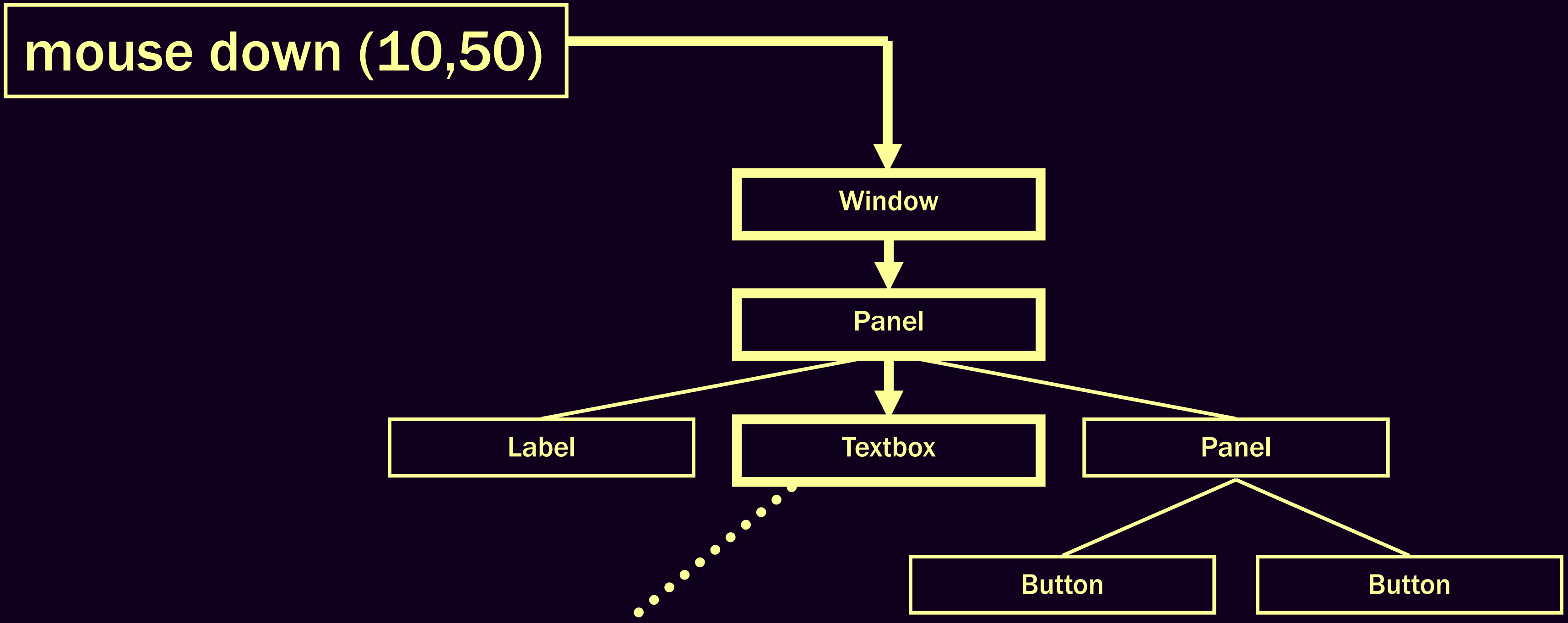
```
function onMouseDown(evt) {
    // do something...
}
```

# Dispatching Events



```
function onMouseDown(evt) {
    // do something...
}
```

# Dispatching Events



```
function onMouseDown(evt) {
    // do something...
}
```

# Events in the Web Browser

Events are dispatched very much like this within the web browser

DOM structure of HTML document is used

Two-stage dispatch process:

- Capture phase

  Event is sent down the tree to target

- Bubbling phase

  Event goes back up the tree to the window