EVENT HANDLING

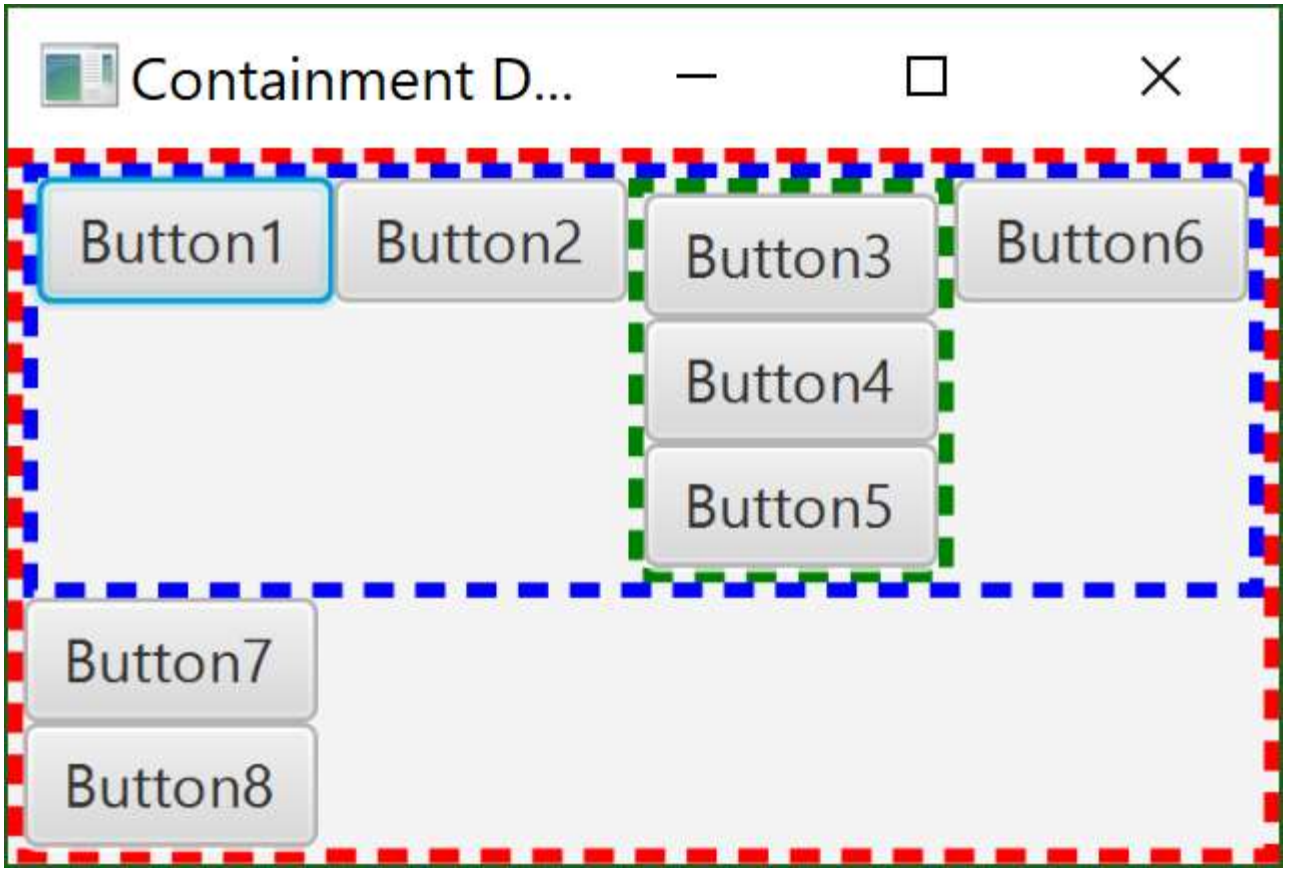
CMPT 381

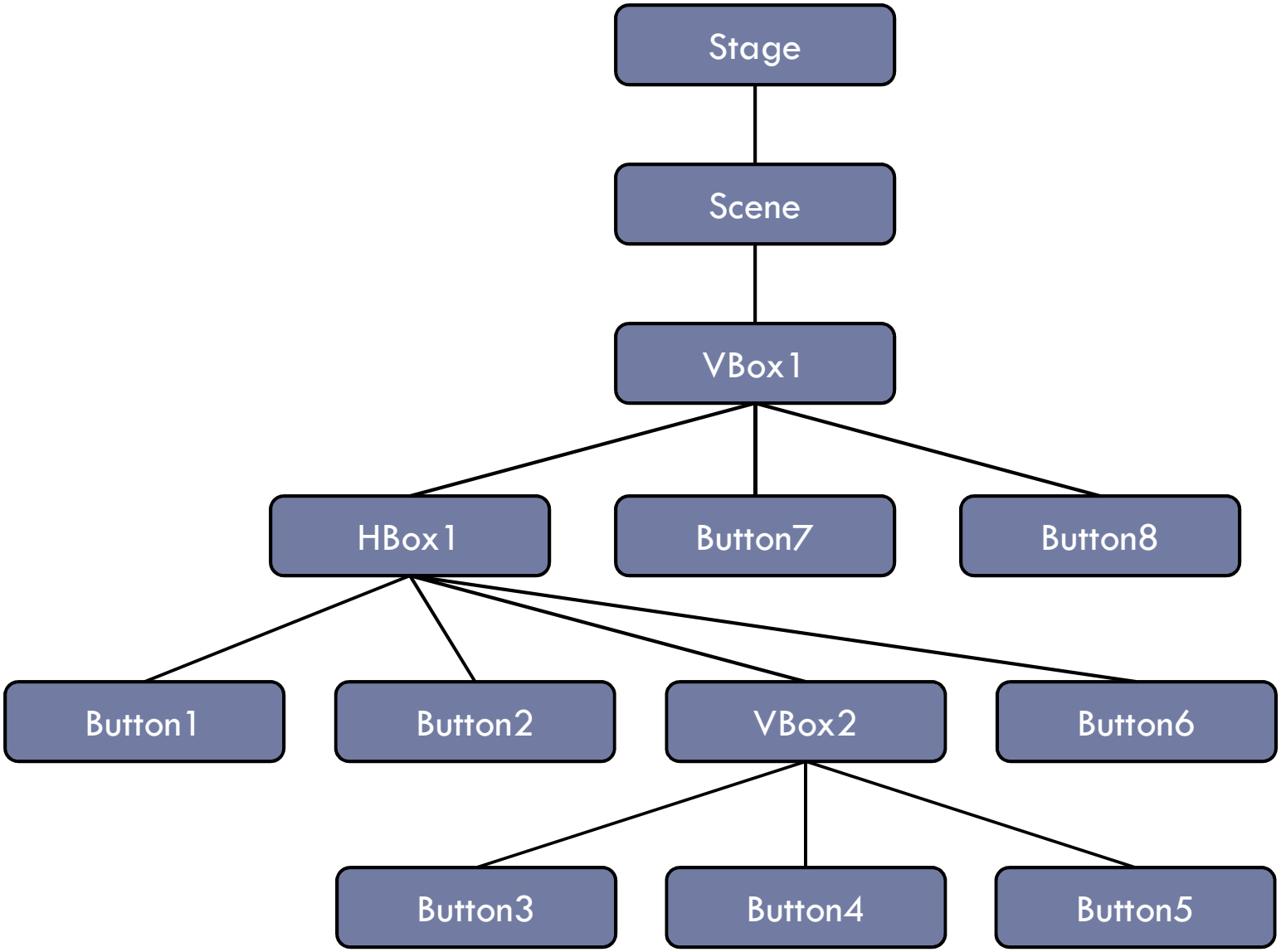
Overview

- Review of containment hierarchies
- Event handling
- Event responsibilities for window systems

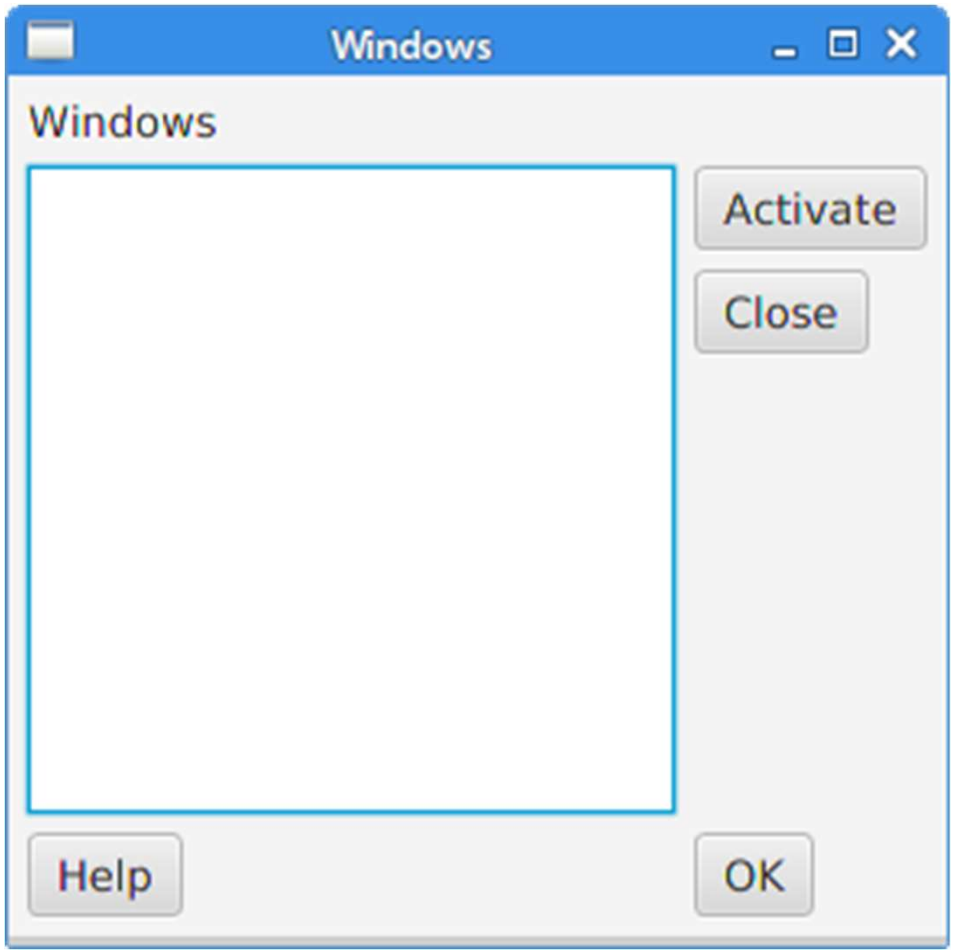
Containment Hierarchies

Draw the containment hierarchy:




Stage
Scene
VBox1
HBox1
Button7
Button8
Button1
Button2
VBox2
Button6
Button3
Button4
Button5

Draw the containment hierarchy:



Containment hierarchies

- Group widgets together (into container widgets)
- Control layout
- Control event routing

Event Handling

Event-Based GUI Programming

- Communication from user to computer via events
- An *input event* is something "interesting" that happens in an input channel
  - Mouse button down
  - Item dragged
  - Keyboard button pressed
  - Mouse wheel turned
  - Others for other types of input devices
- Also *pseudoevents* that are not direct input events

New devices mean new events

- Johnny Lee: Wiimote hacks
  - http://blog.ted.com/2008/04/11/wii_remote_hack/
- Wiimote library for Java
  - http://motej.sourceforge.net/examples.html
- Kinect SDK for Java
  - http://research.dwi.ufl.edu/ufdw/j4k/

Input Event Architecture

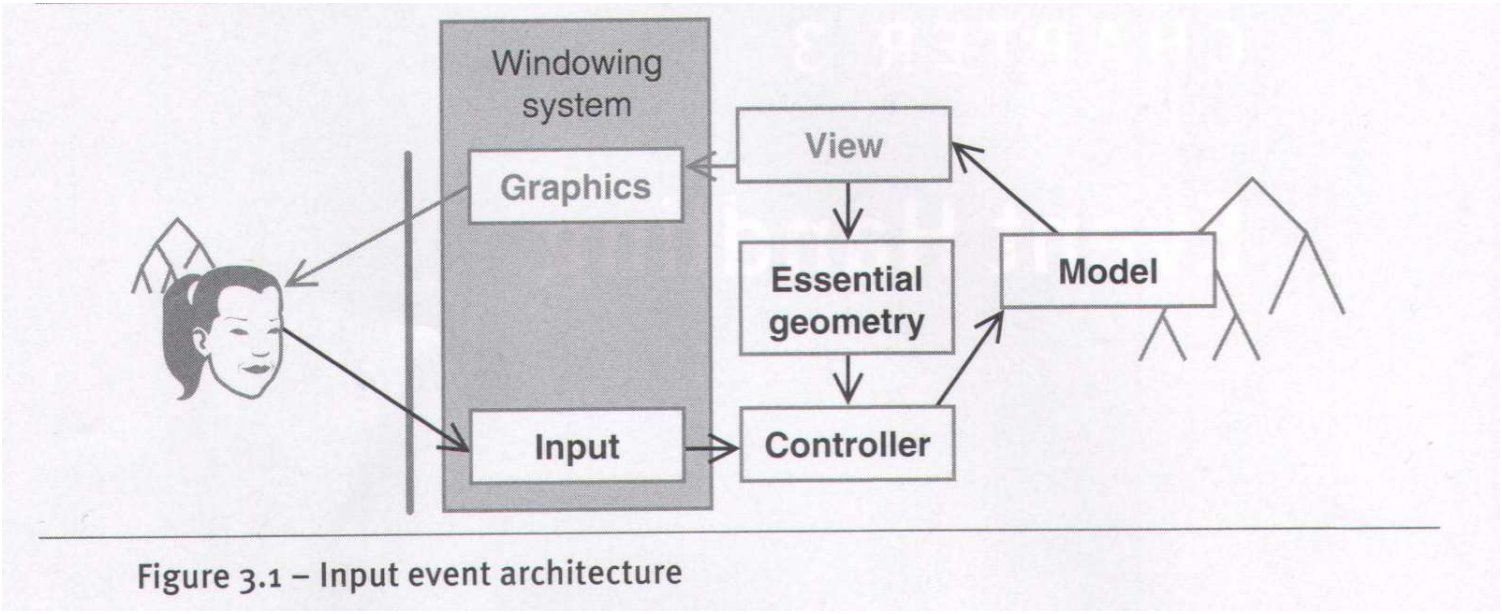


Figure 3.1 – Input event architecture

Types of events

- Device events
  - e.g. mouse button 1 down, wheel turned
- Filtered / high-level events
  - e.g. enter, leave, scroll
- Window events
  - e.g. resize window, close window
- Programmer-defined events
  - e.g. message arrived, calculation finished

Types of events

- Device events
  - e.g. mouse button 1 down, wheel turned
- Filtered / high-level events
  - e.g. enter, leave, scroll
- Window events
  - e.g. resize window, close window
- Programmer-defined events
  - e.g. message arrived, calculation finished

"Pseudoevents"

Three main issues

- Dispatching/receiving events
- Event must trigger code
- Notifying the view of any model changes

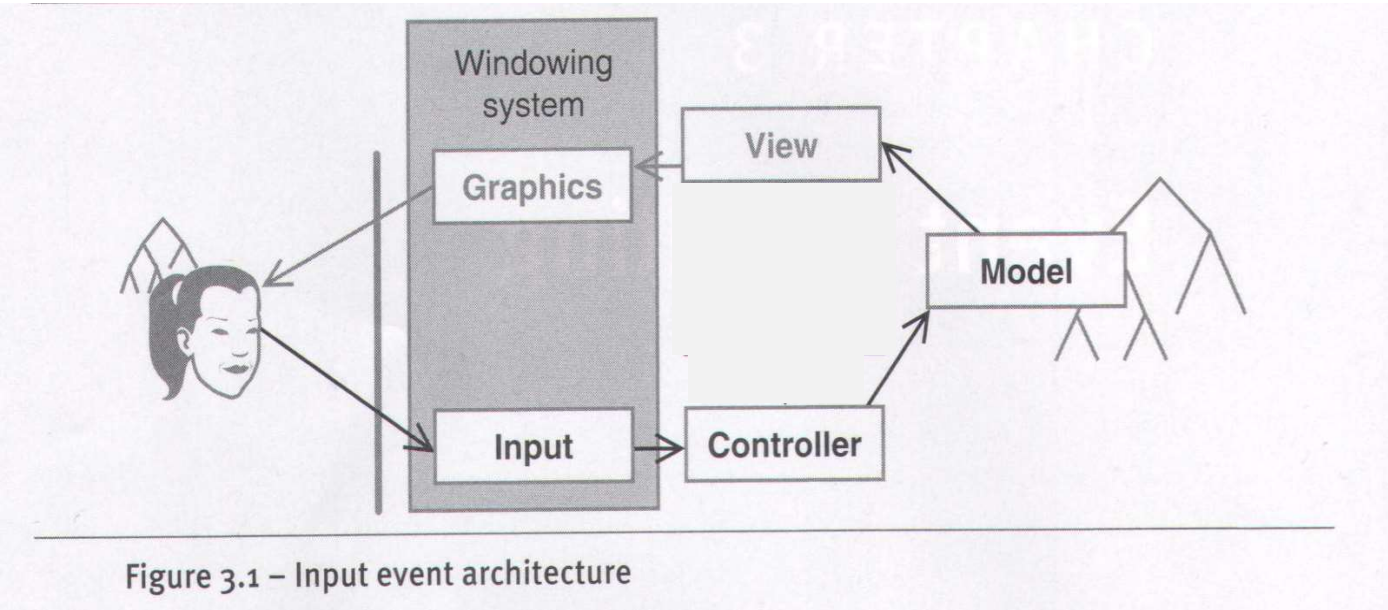


Figure 3.1 – Input event architecture

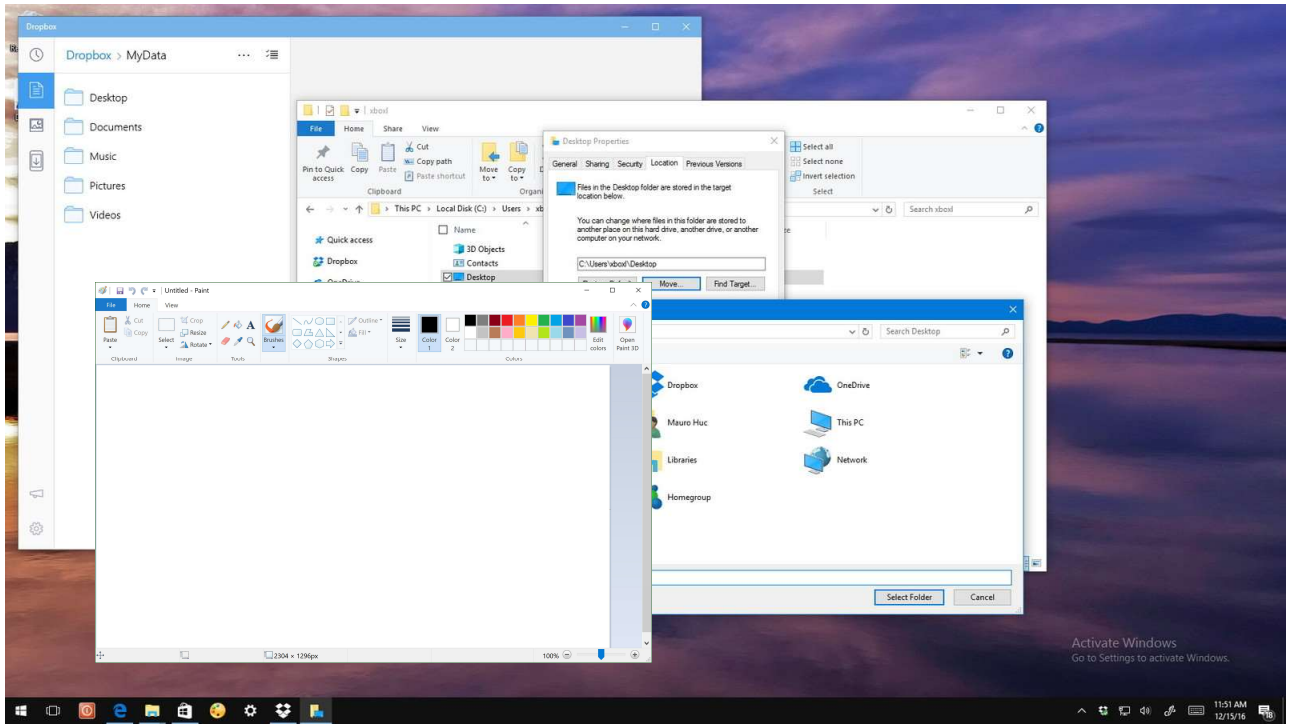
Dispatching Events

Window Trees

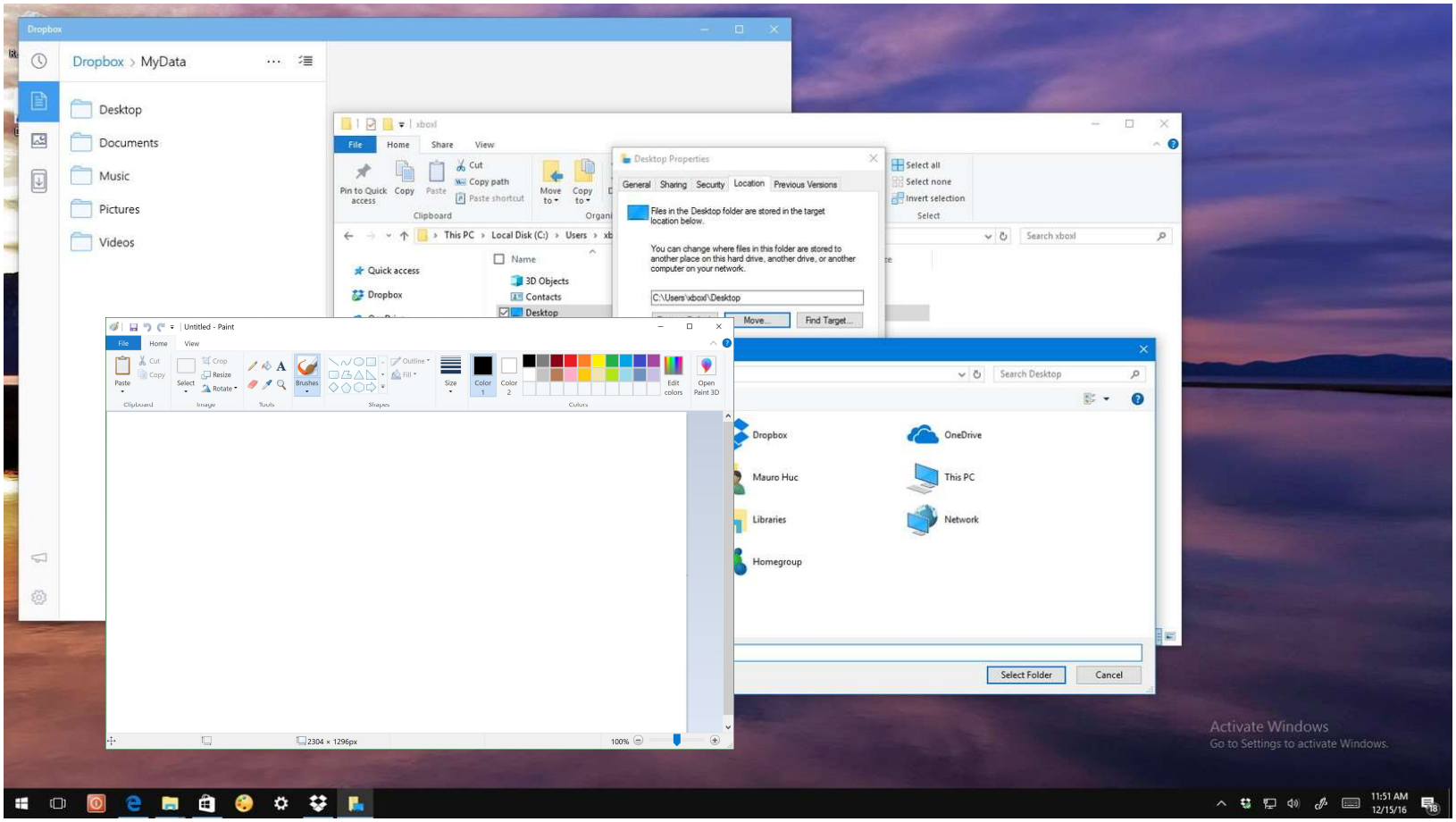
A colour box is clicked on the tool palette in the Paint application - where should the event go?

1. Window system decides which application

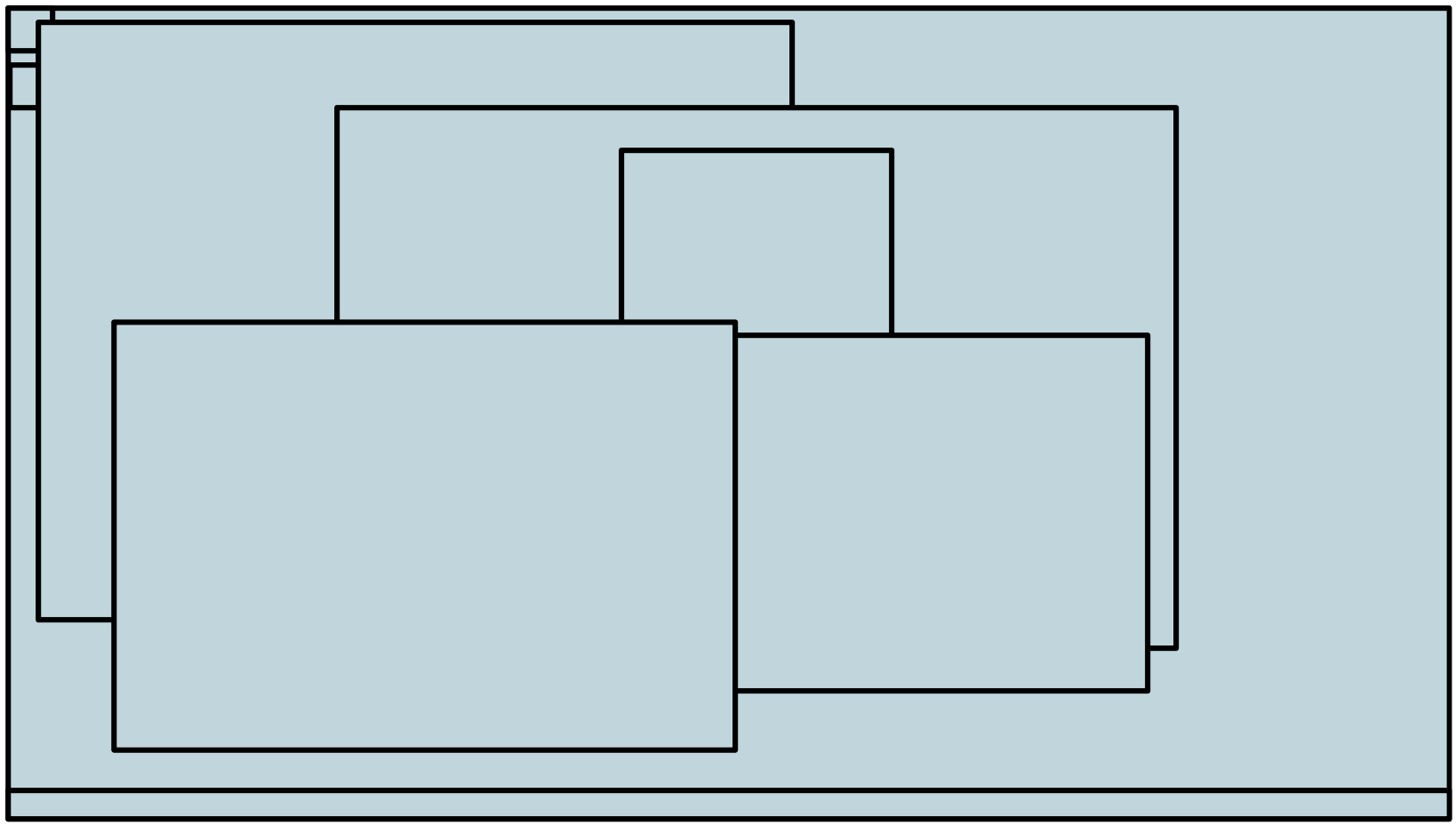
2. Application decides which widget



At the Window Manager level



WM: map of regions to applications



At the application level

Application window tree = containment hierarchy

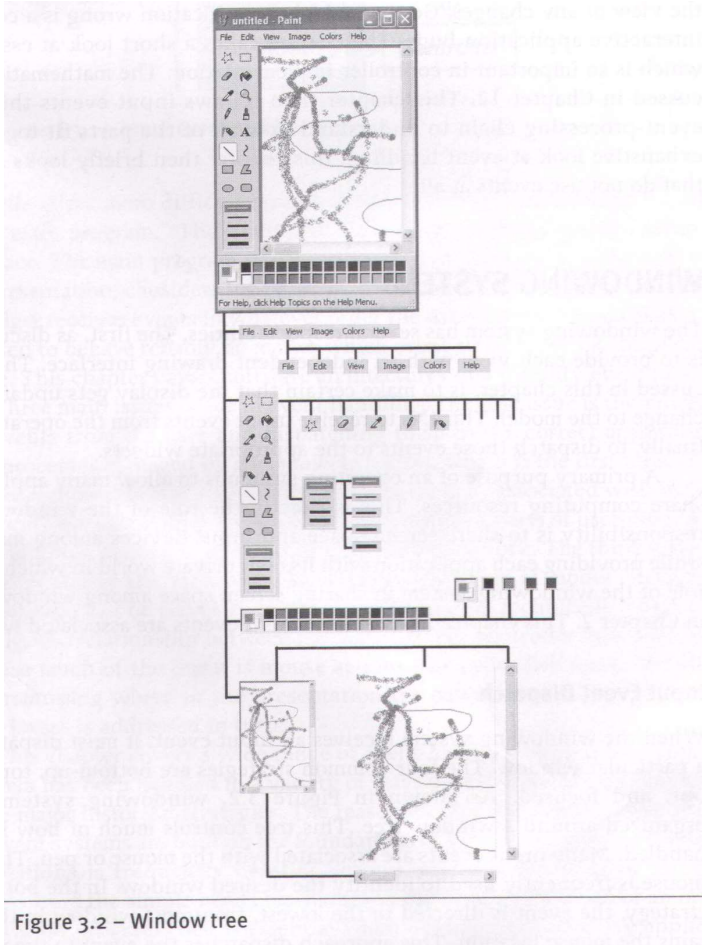


Figure 3.2 – Window tree

Approaches for Directing Events

- Bottom-up
- Top-down
- Bubble-out
- Focus-based

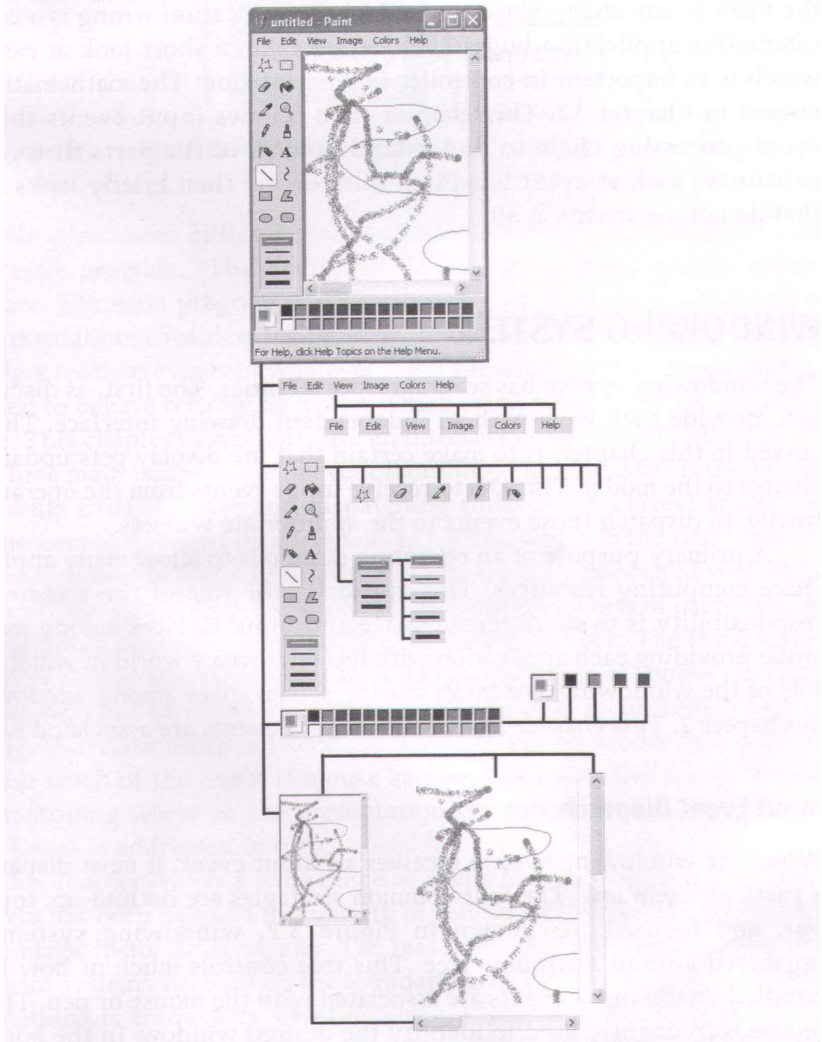


Figure 3.2 – Window tree

Bottom-up

- Event directed to lowest front-most window or widget in the tree
- If not needed, passed up

Top-down

- Event passed recursively down until a widget can use it
  - e.g., from top-level window, to container widget, to visible widget (e.g., a button)

Bubble-out

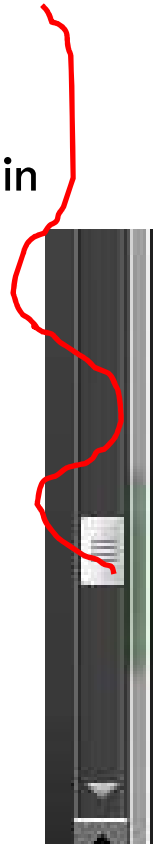
- Some systems don't have nested windows/ widgets
  - e.g., drawing shapes inside other shapes
- Bounding boxes are used to determine if an event has a 'hit'
- Then 'bubble out' passing the event to parents
  - Used in Flash and Javascript/HTML

Focus-based

- Windowing system keeps pointer to correct widget
- Key Focus – focus, unfocus events
  - Use mouse position
    - attach mouse position to all key events and dispatch events in the same way as mouse events
  - Remember a 'focus window'
    - send key events to last window to see mouse-down
    - click-to-type
  - Application control
    - explicitly set keyboard focus
- Mouse Focus
  - Allows us to be sloppy

Focus-based

- Windowing system keeps pointer to correct widget
- Key Focus – focus, unfocus events
  - Use mouse position
    - attach mouse position to all key events and dispatch events in the same way as mouse events
  - Remember a 'focus window'
    - send key events to last window to see mouse-down
    - click-to-type
  - Application control
    - explicitly set keyboard focus
- Mouse Focus
  - Allows us to be sloppy



Device Events

- Most toolkits provide support for common types:
  - Button Events
  - Mouse Movement
  - Keyboard
  - Window Events
  - Other Inputs
    - Multi-user, touch, pressure, speech
    - New input device? Need to extend the UI toolkit
    - https://www.youtube.com/watch?v=PmgoxOHeQO0

Event/Code Binding

Main event loop

Event queues

Event dispatching

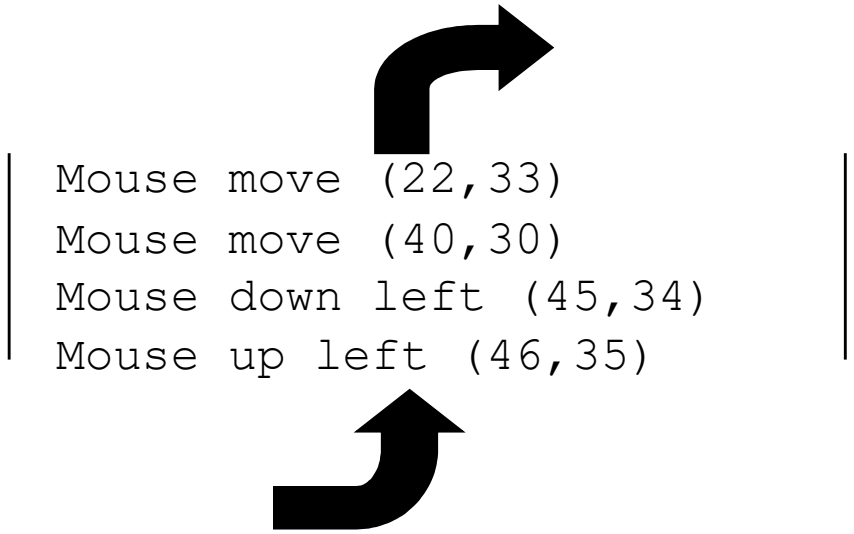
Main event loop

- Simple loop to get and dispatch events
- What layer implements it?

```
Initialization
While (not time to quit) {
    Get next event E
    Dispatch event E
}
```

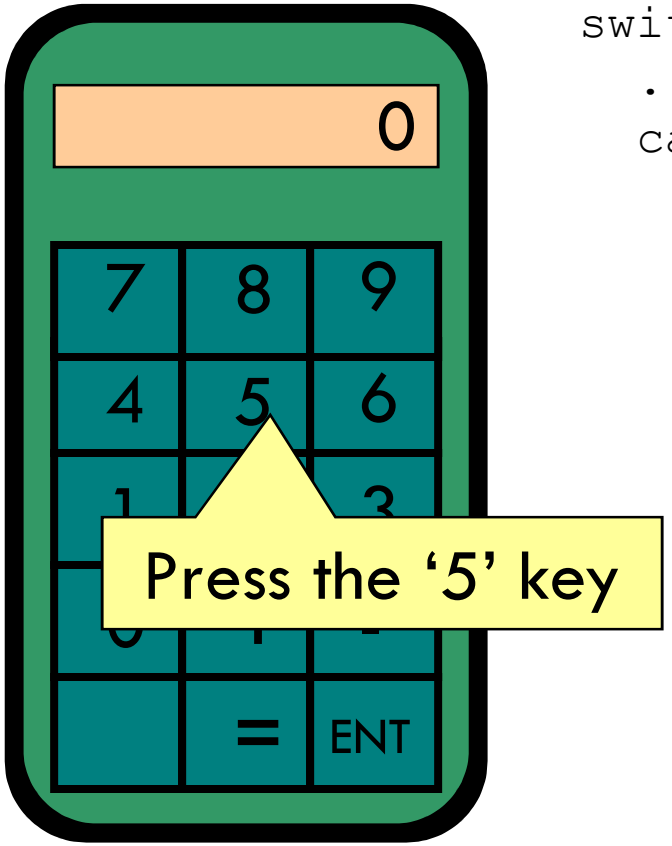
Event Queue

- OS receives device interrupts
- Places events on the queue
- Ensures that events are processed in order
- Different queues for different layers
- At application layer, the main event loop removes them from the queue



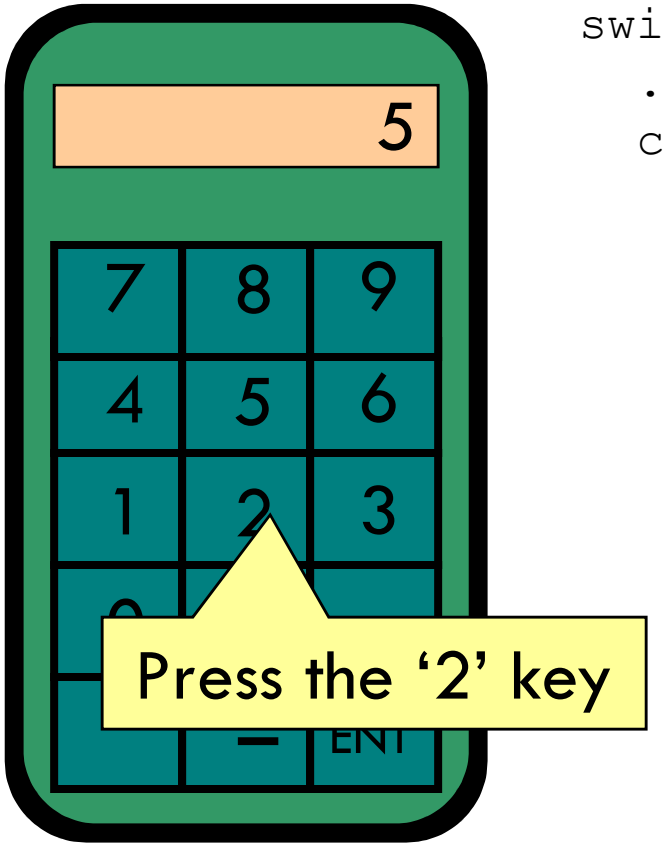
Basic event dispatch

- Application programmer writes dispatch code



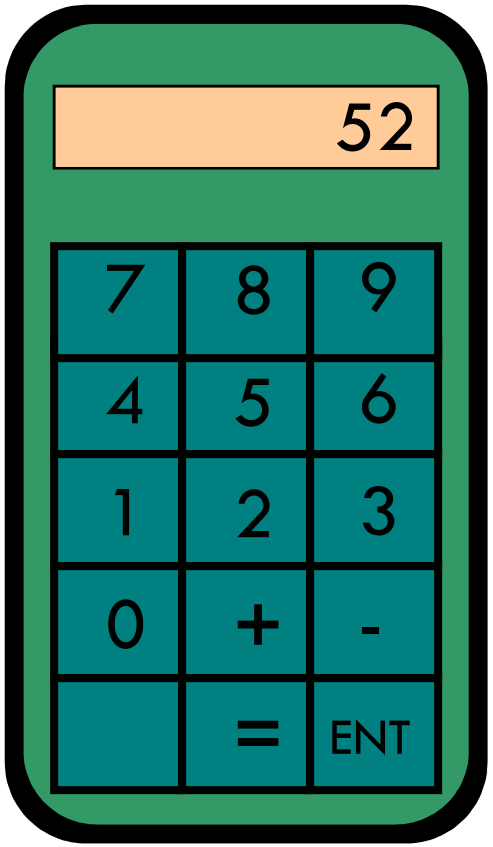

```
Dispatch (event E) {
  switch (E.window) {
    ...
    case FIVE-KEY:
        if (E.type == left-down){
          cur = 5 + 10*cur;
          display (cur);
          last-op = NUMBER;
        }
        ...
```

Basic event dispatch




```
Dispatch (event E) {
  switch (E.window) {
    ...
    case TWO-KEY:
        if (E.type == left-down) {
          cur = 2 + 10*cur;
          display (cur);
          last-op = NUMBER;
        }
        ...
```

Basic event dispatch



Event table dispatch

- Event tables (in the early days…)
  - indexed by event types (integer from 0 - 255)
  - holds pointers to functions that handle each event
  - one table per window

Callback Event Handlers

- Use named events instead of pointers
- Indicate specific code to run
- Tk: the bind command
  - bind associates events with commands
    - bind widget <event-type> command
    - `bind .myCanvas <Button-3> "showPopupMenu %x %y"`
- Tk: widget callbacks
  - certain widgets know they're going to be manipulated
  - they have built-in callbacks for particular actions
    - e.g. "-command" switch for buttons
    - `button .b -text "Test" -command "myProc"`

Object-Oriented Handling

- OO languages naturally handle message passing between independent objects
  - Application code for events can be moved to objects

```
dispatch (event e) {
     //handles the event or passes to child widget
     e.window.handleEvent(e);
}
```

Publish-subscribe dispatch

- Widgets register interest in certain events
  - Publish-subscribe design pattern
  - Publisher: the event producer
    - Note: the toolkit layer itself may be the publisher
  - Subscriber: the interested widget
- When an event occurs:
  - Publisher responsible for notifying all subscribers
- Java: listener interfaces
  - Objects implementing listener interfaces are automatically called when events occur
  - The listener object then calls application code

Swing events and listeners

User action	Listener type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	ActionListener
User presses a mouse button over a component	MouseListener (MouseAdapter)
User moves the mouse over a component	MouseMotionListener (MouseMotionAdapter)
Table or list selection changes	ListSelectionListener
Component gets the keyboard focus	FocusListener
User closes a frame	WindowListener
Component becomes visible	ComponentListener

Swing example

```
JButton helloButton;
ButtonListener listen;  // a listener for button presses

helloButton = new JButton("Hello");
listen = new ButtonListener(); // create the button listener

// assign a command string to the button
helloButton.setActionCommand("hello");
// attach the button listener to the button
helloButton.addActionListener(listen);

class ButtonListener implements ActionListener {
    // actionPerformed is called automatically by the toolkit
    public void actionPerformed(ActionEvent e) {
      String action = e.getActionCommand();
      // handle the command for the hello button
      if (action.equals("hello")) {
           printMessage();
      } ...
```

JavaFX events and handlers

- Primary mechanism: EventHandlers
  - Implementations of the *EventHandler* interface
- Some Listeners still used (for properties e.g., Slider)
- Attaching EventHandlers:
  - Convenience methods (for Nodes with obvious events)
    - Node.setOn*EventType*(…)
    - e.g., myButton.setOnAction(…)
  - Node.addEventFilter(EventType, filter);
  - Node.addEventHandler(EventType, filter);

JavaFX events and handlers

- Node class convenience methods
  - Handle events that *all* Nodes can respond to
- setOnMouseMoved(…)
- setOnMousePressed(…)
- setOnTouchPressed(…)
- setOnZoom(…)
- setOnKeyPressed(…)
  - …and many more
  - docs.oracle.com/javafx/2/api/javafx/scene/Node.html

JavaFX events and handlers

- Specific Node classes have methods for their traditional events:
- Button, ComboBox, Menu, ContextMenu, TextField:
  - setOnAction(…)
- ListView, TableColumn, TreeView
  - setOnEditStart(…), setOnEditCommit(…)
- Window
  - setOnCloseRequest(…), setOnHidden(…)

JavaFX events and handlers

- Widgets with properties can attach change listeners
  - E.g., Slider valueProperty:

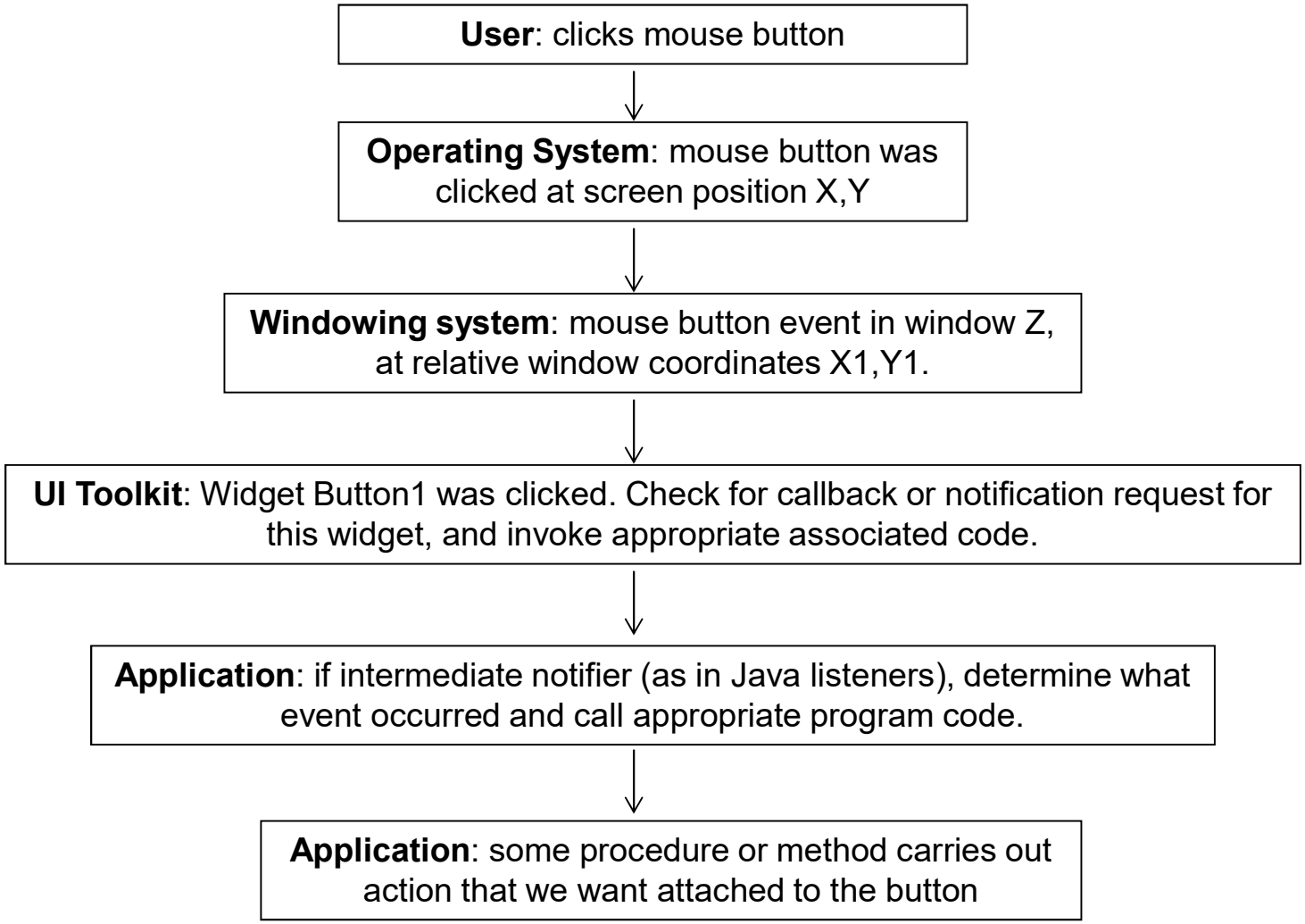
```
// Handle Slider value change events.
mySlider.valueProperty().addListener((observable, oldValue, newValue) -> {
    System.out.println("Slider Value Changed (newValue: " + newValue.intValue() + ")");
});
```

  - E.g., TextField textProperty:

```
// Handle TextField text changes.
myTextField.textProperty().addListener((observable, oldValue, newValue) -> {
    System.out.println("TextField Text Changed (newValue: " + newValue + ")");
});
```

  - note difference to TextField.setOnAction(…)

Life cycle of a mouse-click event



Connecting events in JavaFX (1)

- Instance of an EventHandler class

```
class ButtonHandler implements EventHandler {
    public void handle(Event event) {
        System.out.println("Hello");
    }
}
myHandler = new ButtonHandler;
btn.setOnAction(myHandler);
```

- Anonymous inner class

```
btn.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
      System.out.println("Hello");
  }
});
```

Connecting events in JavaFX (2)

- Lambda expression

```
btn.setOnAction(e -> System.out.println("Hello"));
```

- Method reference

```
btn.setOnAction(this::handleButtonClick);
```

- Also through FXML (later)

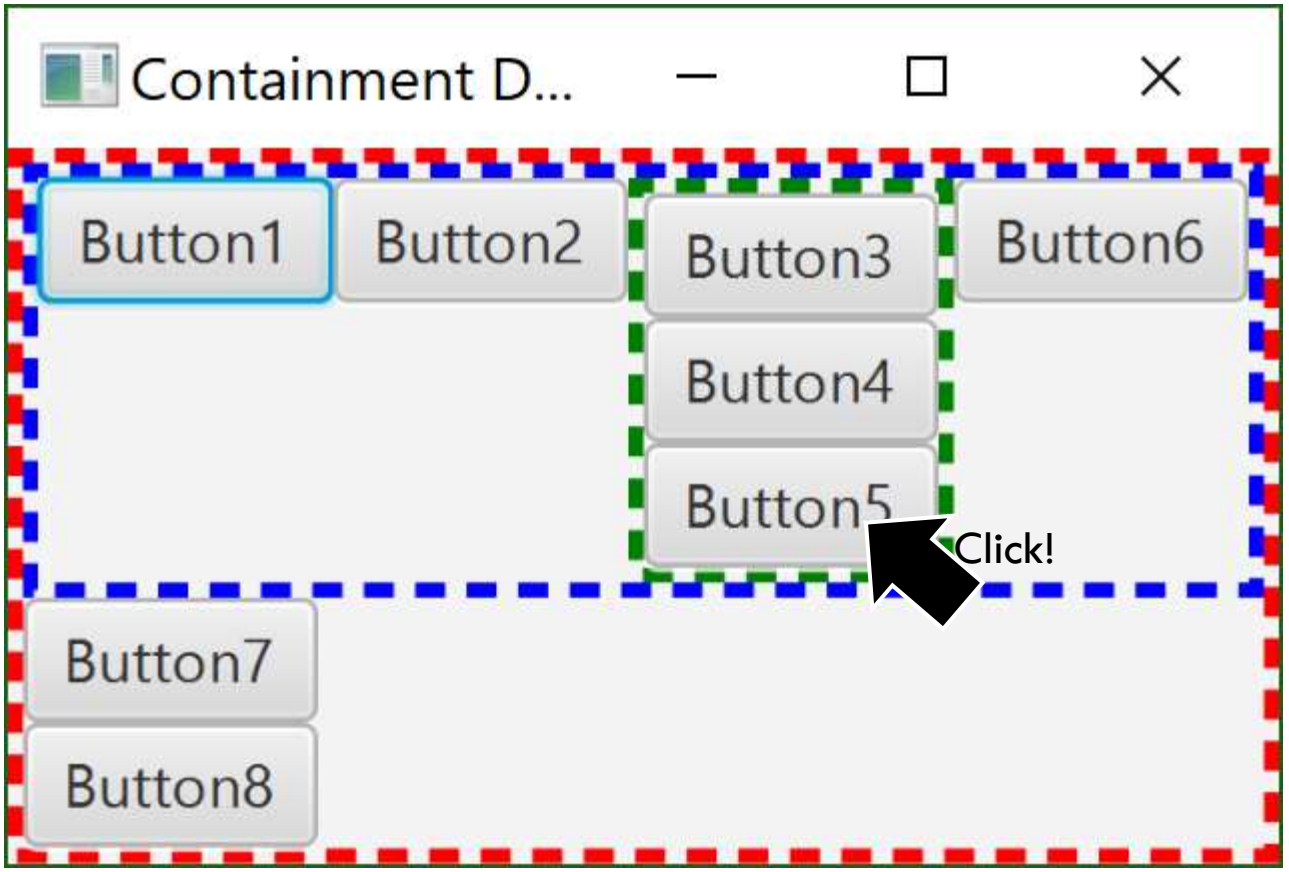
Method references ≈ C# Delegates

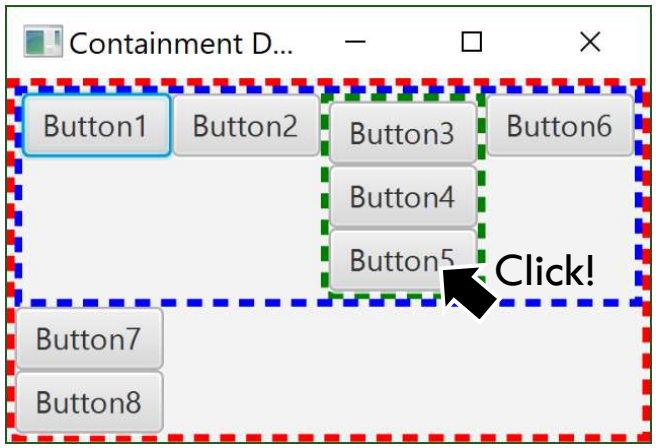
- Similar to callbacks / function pointers
- Assign a method to be called when event occurs
- += and -= operators to add/remove delegates

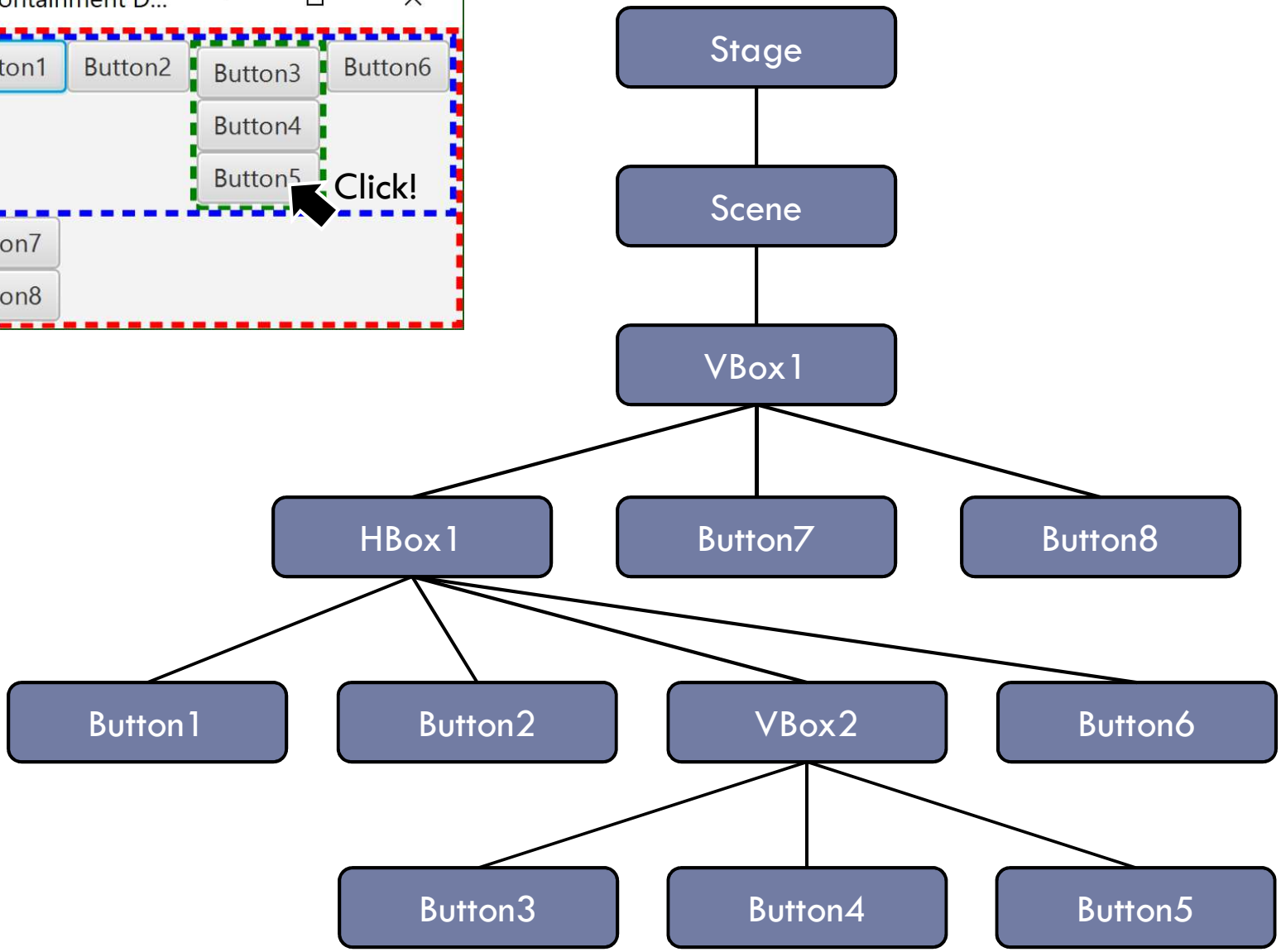
JavaFX event dispatching

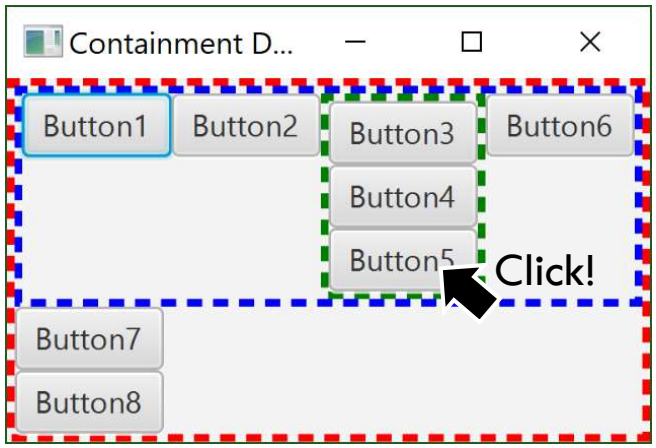
- Two phases: top-down then bottom-up
- Top-down ("event capture")
  - Event *filters*
- Bottom-up ("event bubbling")
  - (note this is different from "bubble-out" in the text)
  - Event *handlers*
- Filters and Handlers use the same EventHandler interface – they are just called at different times

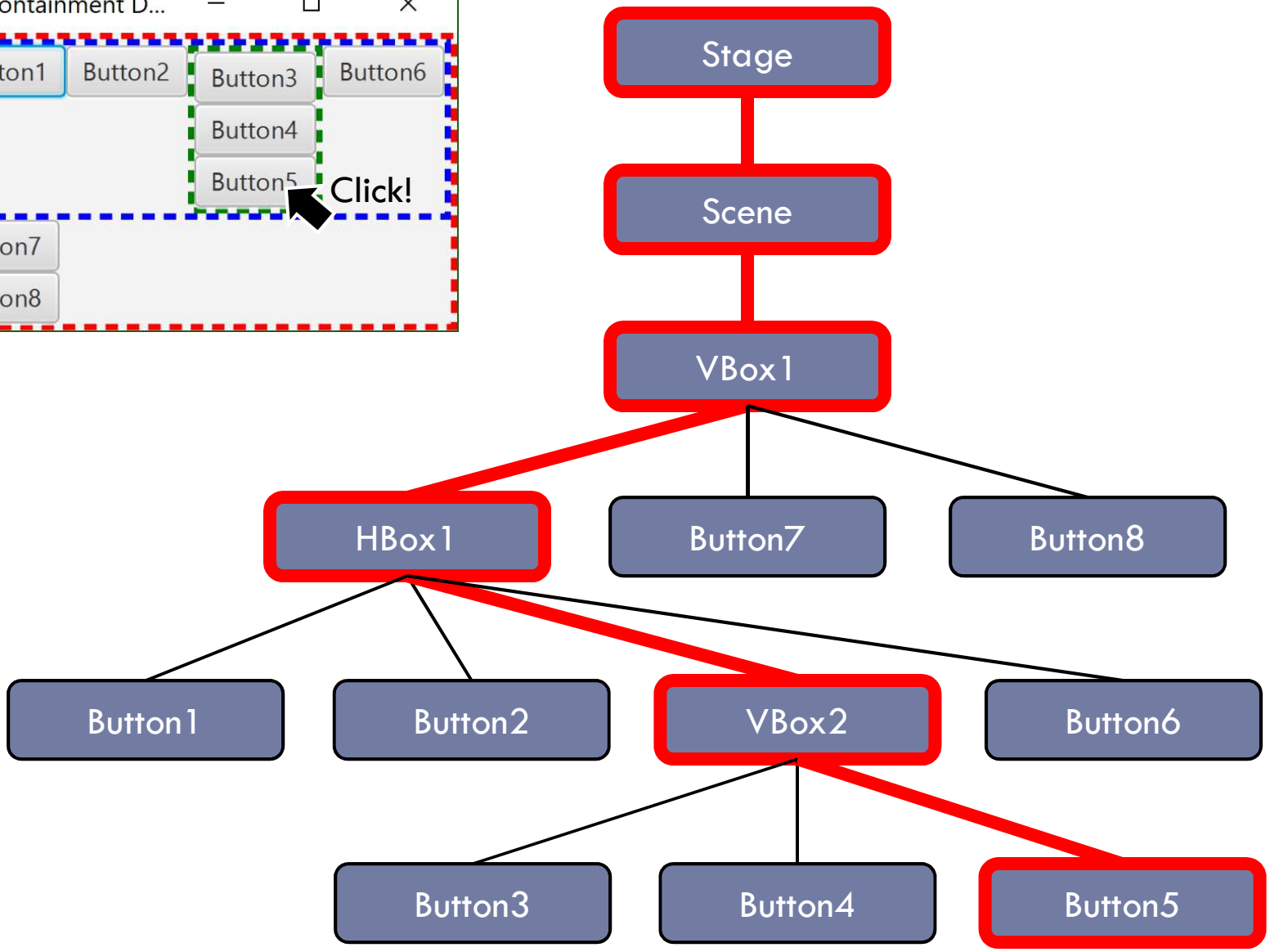
Where are events handled?




Containment D...
Button1
Button2
Button3
Button6
Button4
Button5
Click!
Button7
Button8

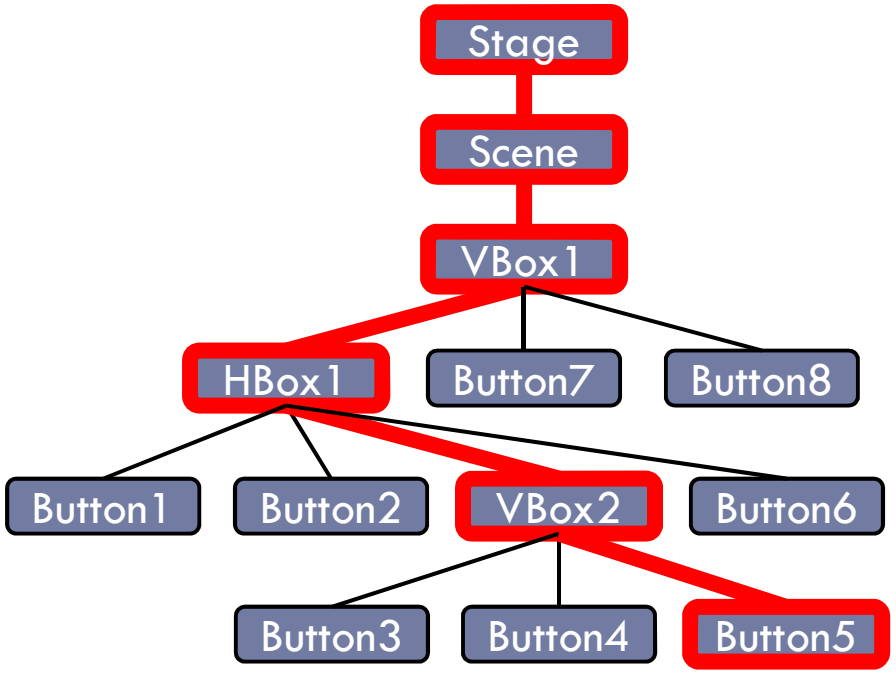

Stage
Scene
VBox1
HBox1
Button7
Button8
Button1
Button2
VBox2
Button6
Button3
Button4
Button5


Containment D...
Button1
Button2
Button3
Button6
Button4
Button5
Click!
Button7
Button8


Stage
Scene
VBox1
HBox1
Button7
Button8
Button1
Button2
VBox2
Button6
Button3
Button4
Button5

Where are events handled?

- At any or all Nodes
- Going downwards, with event filters
- Going back up, with event handlers
- In this UI the event could be handled 12 times
- How to stop further processing?
  - event.consume();



Node.addEventFilter(EventType, filter);
Node.addEventHandler(EventType, filter);