CMSC434

Introduction to Human-Computer Interaction

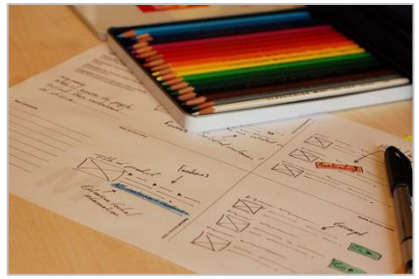
Week 10 | Lecture 16 | Mar 29, 2016

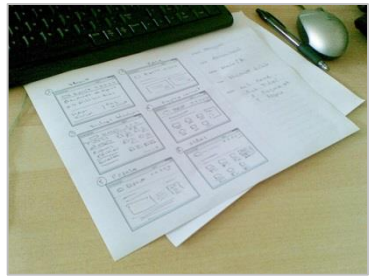
Engineering Interfaces II

Jon Froehlich

@jonfroehlich











Assignments

Use the Calendar feature in Canvas to see all upcoming assignments and deadlines

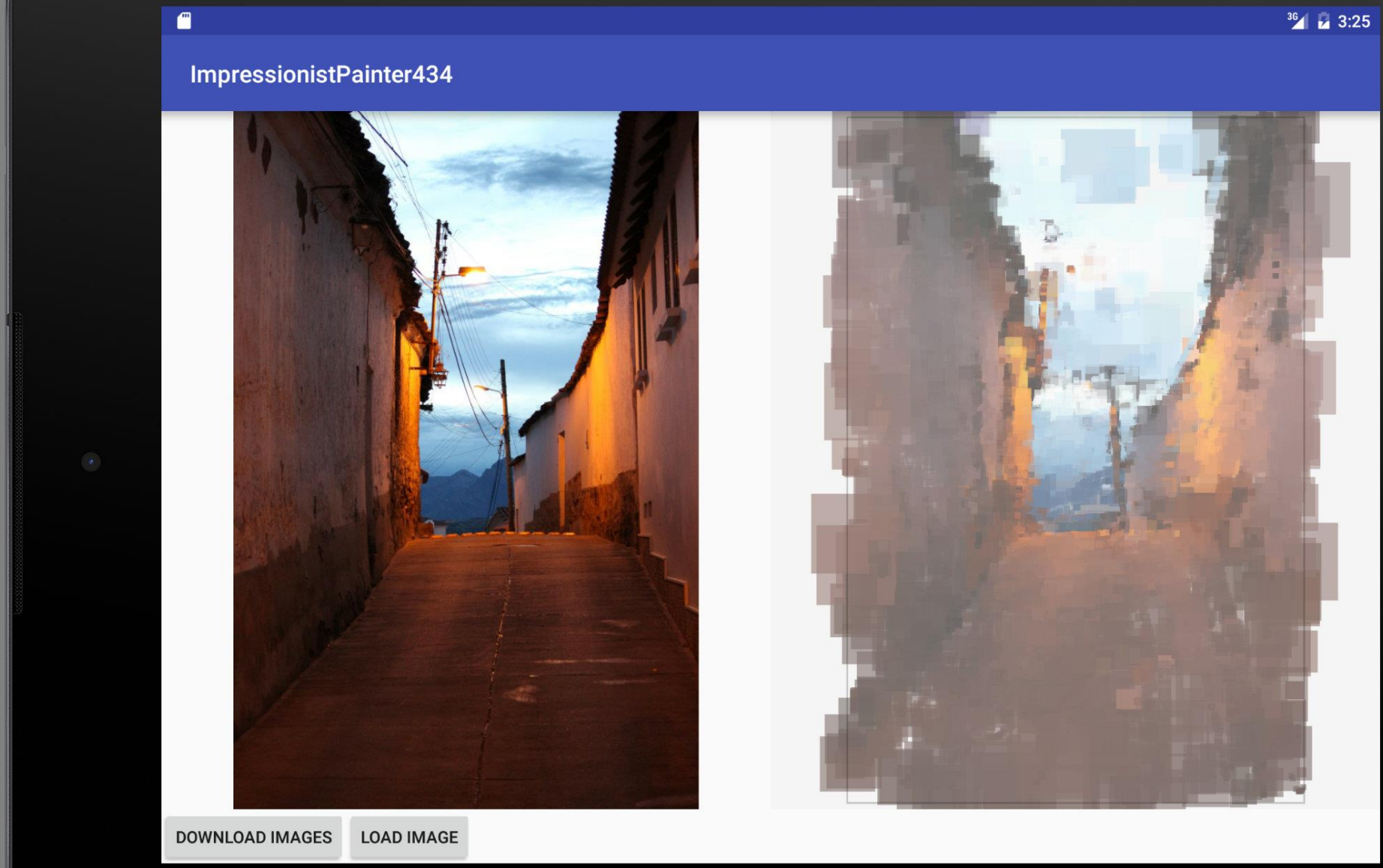
TA04 Mid-Fi Prototypes: Due March 31th

Midterm: Thursday, April 7th

- Will cover the readings, content from homework assignments, and lectures

IA08 Android Doodle Prototype: Due April 5 & 8

- Come to class with initial prototype on April 5th. You must upload a screenshot + code to Canvas on April 5th
- Submit final version April 8th.
- Same deliverables as before except you must also submit your 1-3 images of your favorite artwork made with your app. You should include the raw image + impressionist image for each submission. Matt and I will go through these, find our favorite top ~10 and post them for you to vote on as a class.



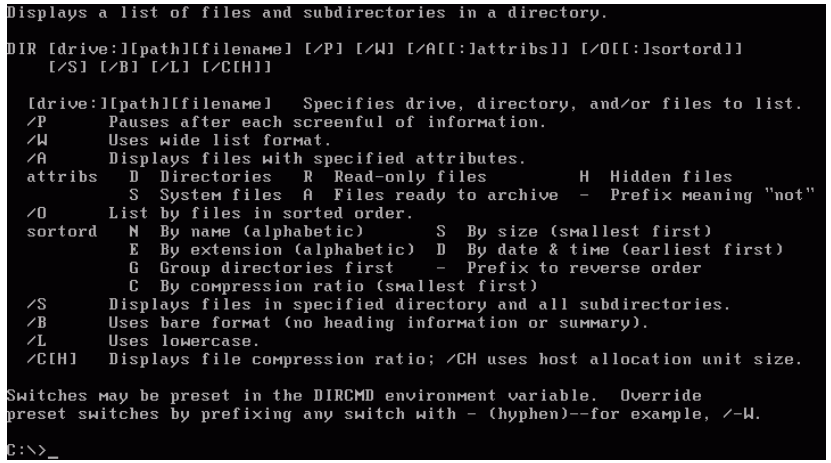
Video Demo on Assignment Website

Members Settings

Video Demo

This video demo illustrates the overall functionality of this assignment and some example brush types. The video is not comprehensive and does not include all required features (*e.g.*, the save feature is not shown).

Evolution of User Interfaces

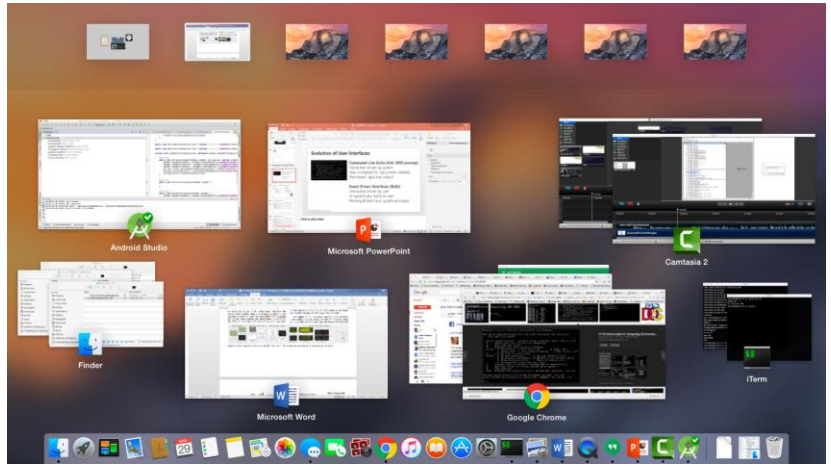


Command Line (Unix shell, DOS prompt)

Interaction driven by system

User prompted for input when needed

Text-based input and output



Event-Driven Interfaces (GUIs)

Interaction driven by user

UI constantly waiting for input events

Pointing & text input, graphical output

Based on: Jeff Heer, UW HCID 520, 02-UIToolkits.pdf

Procedural vs. Event-Driven Programming

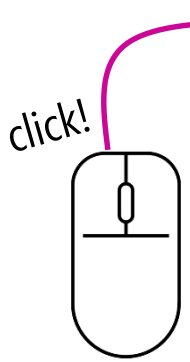
Procedural

Code is executed in sequential order

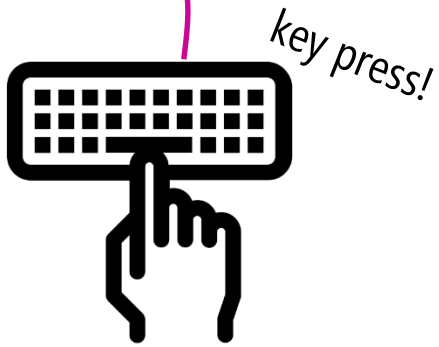
```
Statement 1
Statement 2
Statement 3
…
Statement N
```

Event-Driven

Code is executed based upon events




```
Method 1
Method 2
Method 3
…
Method 4
```



Procedural Programming Example

```
#include <stdio.h>

int main( )
{
  char str[100];

  printf("How old are you?");
  gets( str );

  printf( "\nYou entered: ");
  puts( str );
  ...
```

Control flow. Execution starts at main() and executes sequentially, branching with if, for, and while statements and method calls.

User input. When we need user input, we call read() on the console stream and wait (blocks) until the user types something, then return

Procedural Programming Example

```
#include <stdio.h>

int main( )
{
  char str[100];

  printf("How old ar
  gets( str );

  printf( "\nYou entered: ");
  puts( str );
  ...
```

Execution literally blocks until the user types in a string & hits '\n'

Control flow. Execution starts at main() and executes sequentially, branching with if, for, and while statements and method calls.

stream and wait (blocks) until the user types something, then return

Procedural Programming Example

```c
#include <stdio.h>

int main( )
{
  char str[100];

  printf("How old ar
  gets( str );

  printf( "\nYou entered: ");
  puts( str );
  ...
```

Control flow. Execution starts at main() and executes sequentially, branching with if, for, and while statements and method calls.

Execution literally blocks until the user types in a string & hits '\n'

stream and wait (blocks) until the user types something, then return

How do we respond to input **from > 1 source** (*e.g.*, keyboard & mouse?)

What if user wants to interact in a more **dynamic order**?

Unlike MS-DOS-based applications, Windows-based applications are **event-driven**. They do **not make explicit function calls** to obtain input (such as C run-time library calls). Instead, Windows-based applications **wait for the system** to pass input to them.

Windows and Messages
Official Windows Developer Documentation

Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Event-Driven Programming

```
int WinMain( )
{
  ... initialization code ...
  ... setup and show GUI ...

  // Enter event Loop
  while(true){
    Event e = GetEvent();
    DispatchEvent(e);
  }
}
```

Control flow. Program waits for user input events. OS routes user input events to program, which are processed in an event queue.

Message loop. Continuously waits for events to process in a message queue.

User input. When a user moves the mouse over the program window or presses a key on the keyboard when this window is in focus, the OS sends the event to this program, where it is processed by the message loop.

Event-Driven Programming

```
int WinMain( )
{
   ... initialization code ...
   ... setup and show GUI ...

   // Enter event Loop
   while(true){
      Event e = GetEvent();
      DispatchEvent(e);
   }
}
```

Control flow. Program waits for user input events. OS routes user input events to program, which are processed in an event queue.

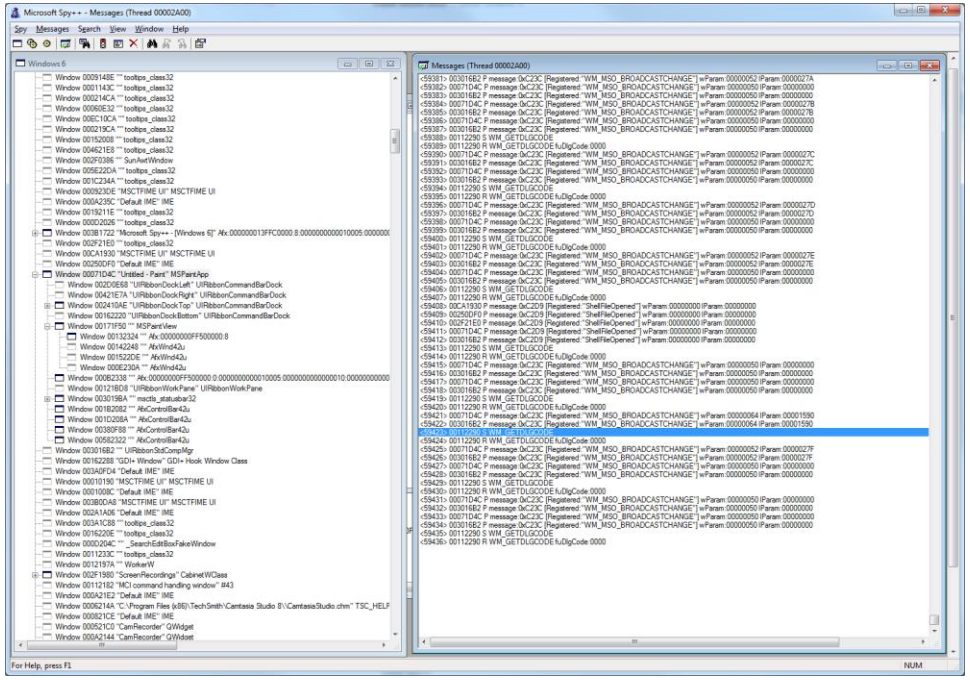
Message loop. Continuously waits for events to process in a message queue.

User input. When a user moves the mouse over the program window or presses a key on the keyboard when this window is in focus, the OS sends where it is loop.

This message loop—the core part of any UI program—is often hidden from the typical UI developer. It's setup by the UI toolkit/framework that you use.

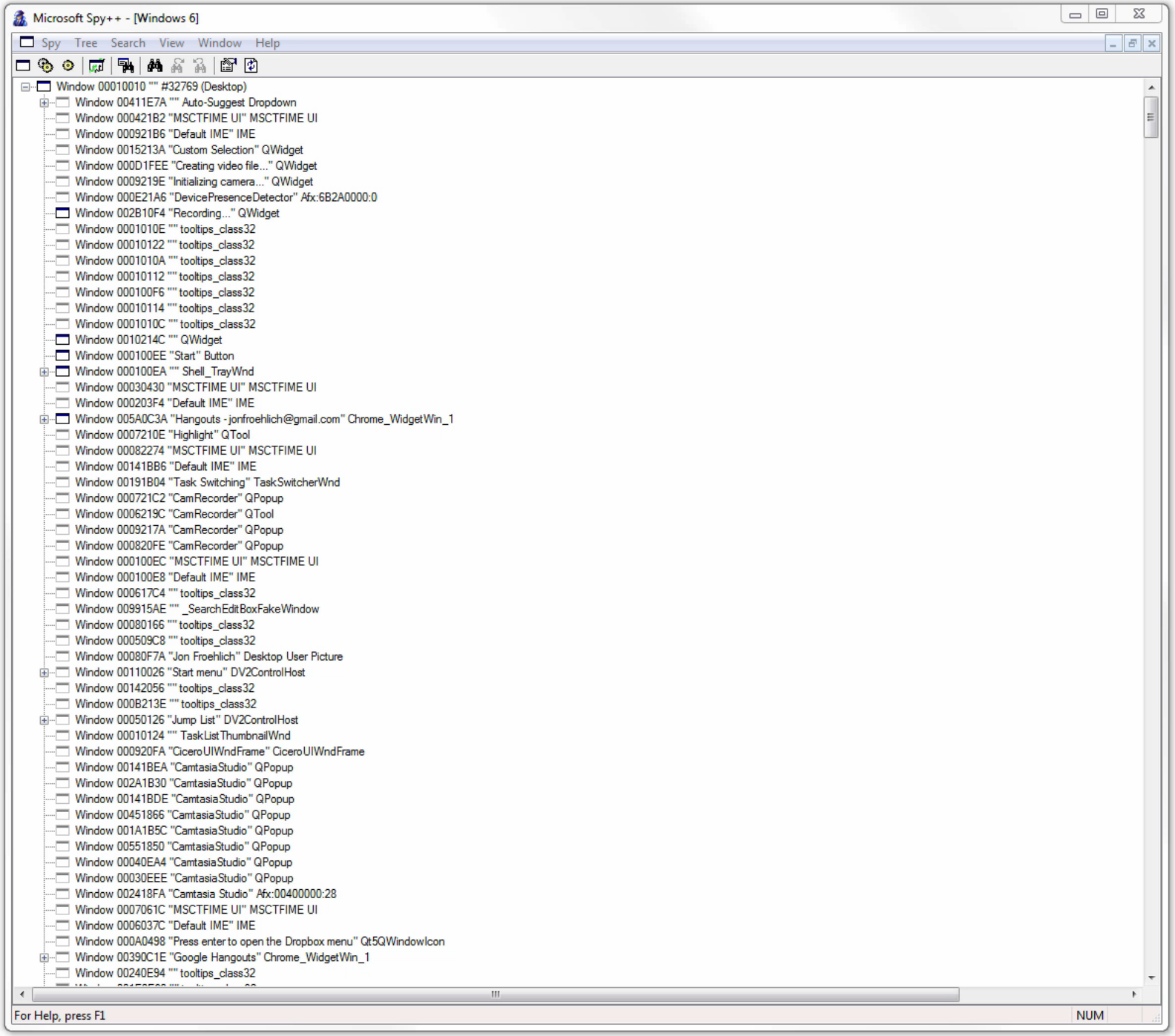
Windows Event Messaging Example

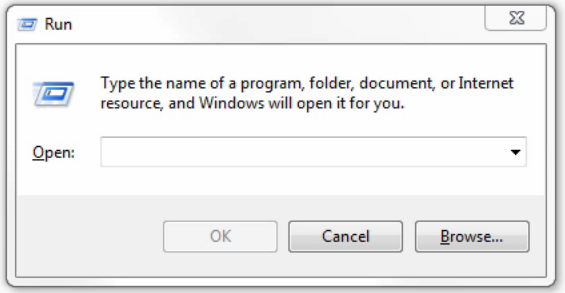
On the next slide, I'll play a video of the Spy++ tool for sniffing Windows event messages



Microsoft Spy++ Tool

https://msdn.microsoft.com/en-us/library/dd460756.aspx


Microsoft Spy++ - [Windows 6]
Spy Tree Search View Window Help
Window 00010010 "" #32769 (Desktop)
Window 00411E7A "" Auto-Suggest Dropdown
Window 000421B2 "MSCTFIME UI" MSCTFIME UI
Window 000921B6 "Default IME" IME
Window 0015213A "Custom Selection" QWidget
Window 000D1FEE "Creating video file..." QWidget
Window 0009219E "Initializing camera..." QWidget
Window 000E21A6 "DevicePresenceDetector" Afx:6B2A0000:0
Window 002B10F4 "Recording..." QWidget
Window 0001010E "" tooltips_class32
Window 00010122 "" tooltips_class32
Window 0001010A "" tooltips_class32
Window 00010112 "" tooltips_class32
Window 000100F6 "" tooltips_class32
Window 00010114 "" tooltips_class32
Window 0001010C "" tooltips_class32
Window 0010214C "" QWidget
Window 000100EE "Start" Button
Window 000100EA "" Shell_TrayWnd
Window 00030430 "MSCTFIME UI" MSCTFIME UI
Window 000203F4 "Default IME" IME
Window 005A0C3A "Hangouts - jonfroehlich@gmail.com" Chrome_WidgetWin_1
Window 0007210E "Highlight" QTool
Window 00082274 "MSCTFIME UI" MSCTFIME UI
Window 00141BB6 "Default IME" IME
Window 00191B04 "Task Switching" TaskSwitcherWnd
Window 000721C2 "CamRecorder" QPopup
Window 0006219C "CamRecorder" QTool
Window 0009217A "CamRecorder" QPopup
Window 000820FE "CamRecorder" QPopup
Window 000100EC "MSCTFIME UI" MSCTFIME UI
Window 000100E8 "Default IME" IME
Window 000617C4 "" tooltips_class32
Window 009915AE "" _SearchEditBoxFakeWindow
Window 00080166 "" tooltips_class32
Window 000509C8 "" tooltips_class32
Window 00080F7A "Jon Froehlich" Desktop User Picture
Window 00110026 "Start menu" DV2ControlHost
Window 00142056 "" tooltips_class32
Window 000B213E "" tooltips_class32
Window 00050126 "Jump List" DV2ControlHost
Window 00010124 "" TaskListThumbnailWnd
Window 000920FA "CiceroUIWndFrame" CiceroUIWndFrame
Window 00141BEA "CamtasiaStudio" QPopup
Window 002A1B30 "CamtasiaStudio" QPopup
Window 00141BDE "CamtasiaStudio" QPopup
Window 00451866 "CamtasiaStudio" QPopup
Window 001A1B5C "CamtasiaStudio" QPopup
Window 00551850 "CamtasiaStudio" QPopup
Window 00040EA4 "CamtasiaStudio" QPopup
Window 00030EEE "CamtasiaStudio" QPopup
Window 002418FA "Camtasia Studio" Afx:00400000:28
Window 0007061C "MSCTFIME UI" MSCTFIME UI
Window 0006037C "Default IME" IME
Window 000A0498 "Press enter to open the Dropbox menu" Qt5QWindowIcon
Window 00390C1E "Google Hangouts" Chrome_WidgetWin_1
Window 00240E94 "" tooltips_class32
For Help, press F1
NUM

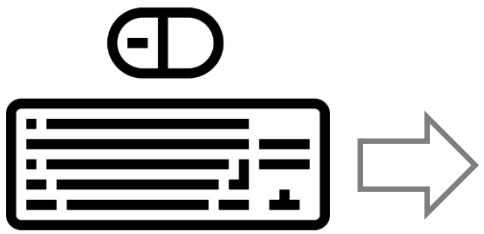

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
OK
Cancel
Browse...

How do user input events (*e.g.*, keypresses, mouse clicks) get from the hardware, into the operating system, and eventually processed by an application?

How do user input events (*e.g.*, keypresses, mouse clicks) get from the hardware, into the operating system, and eventually processed by an application?

A: Let's take a look! This example is specifically from MS Windows, but it's similar across all modern operating systems.

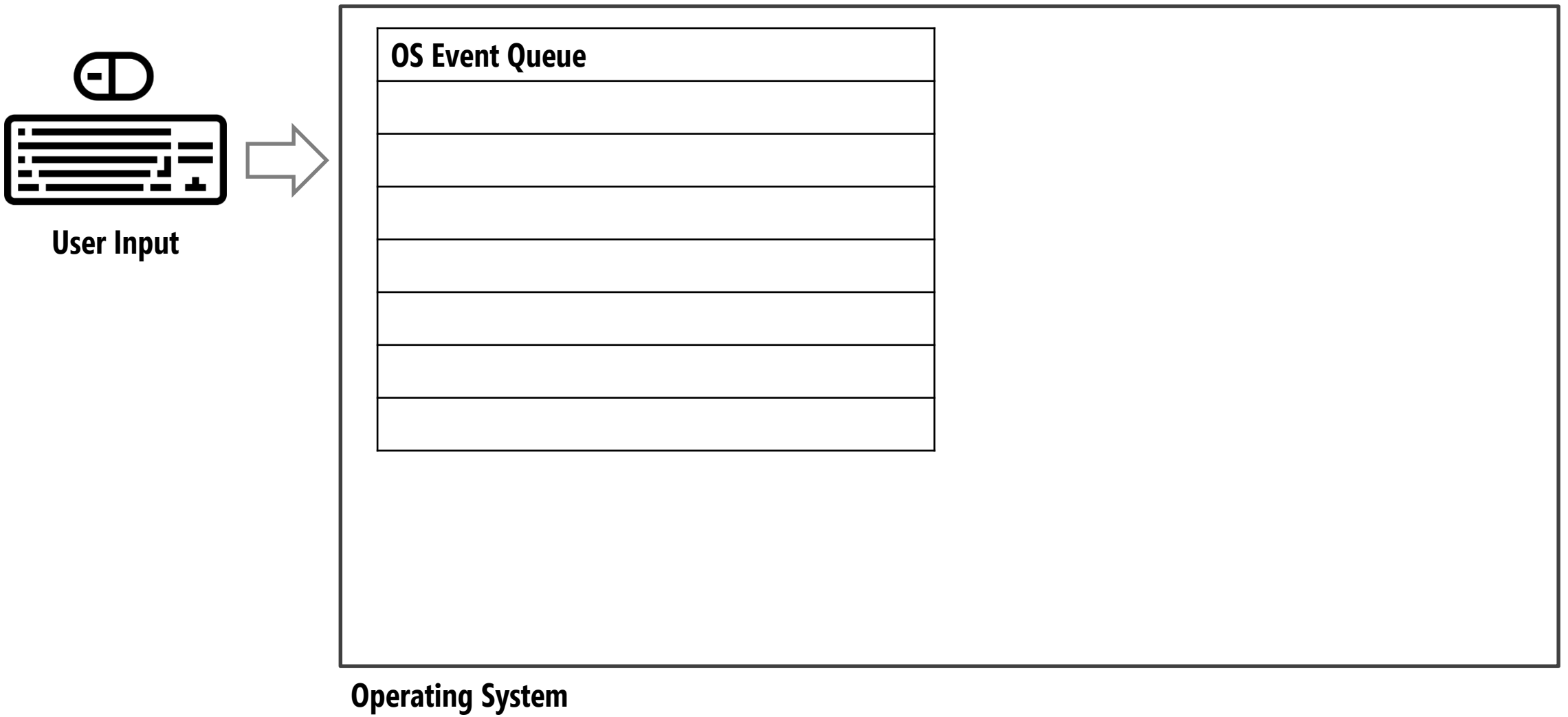
Event Processing Diagram



User Input

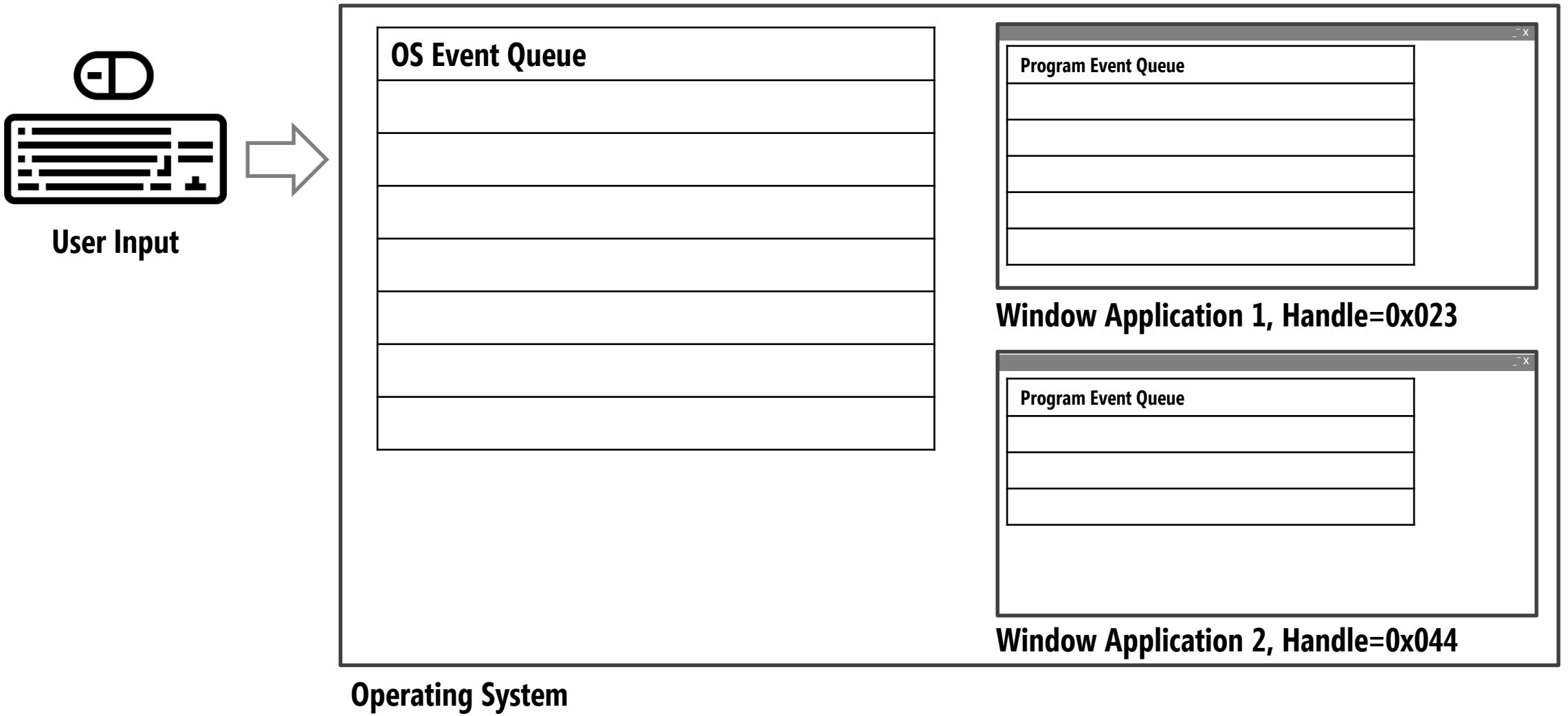
Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Event Processing Diagram



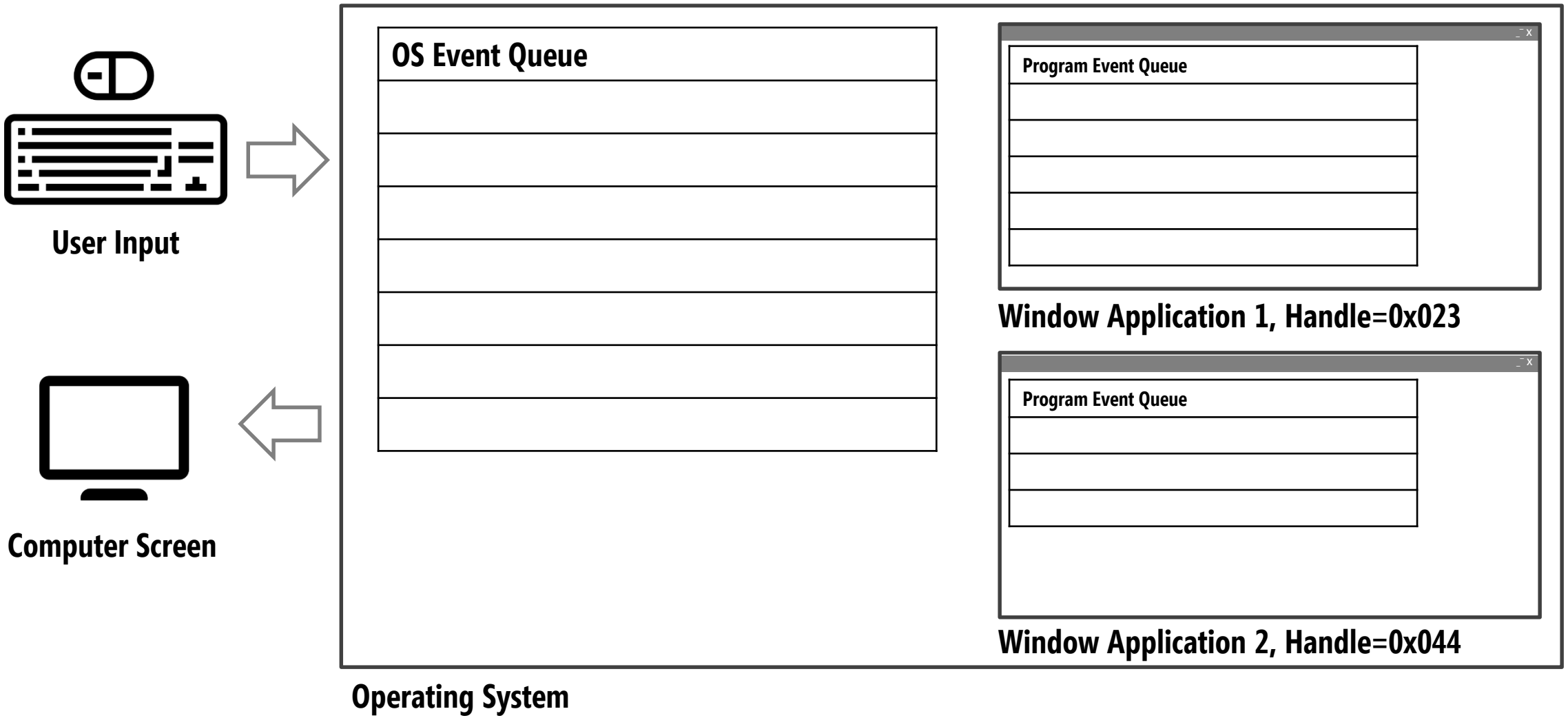
Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Event Processing Diagram



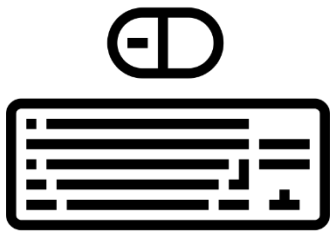
Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Event Processing Diagram



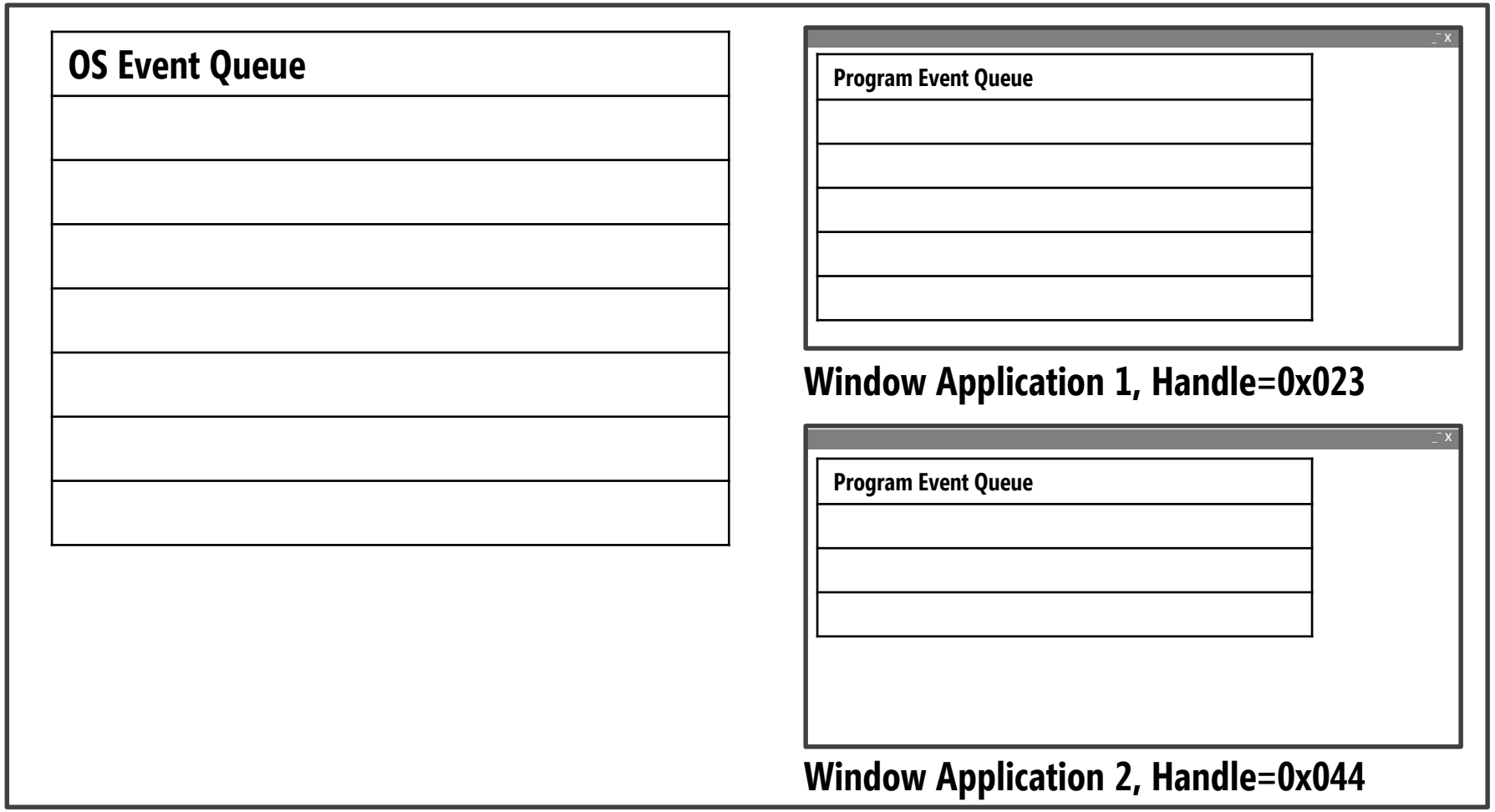
Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Whenever the user moves the mouse, clicks the mouse button, or types on the keyboard, the device driver for the respective device converts the input into messages and places them into the **system message queue**



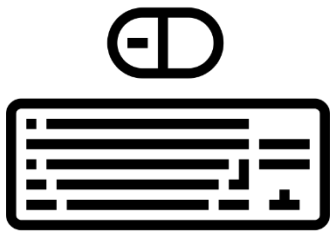
User Input

Computer Screen



Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Whenever the user moves the mouse, clicks the mouse button, or types on the keyboard, the device driver for the respective device converts the input into messages and places them into the **system message queue**



User Input

Computer Screen

OS Event Queue
WM_MOUSEMOVE, 0x023, [data]
WM_LBUTTONDOWN, 0x023, [data]
WM_LBUTTONUP, 0x023, [data]
WM_MOUSEMOVE, 0x023, [data]
WM_MOUSELEAVE, 0x023, [data]
WM_MOUSEMOVE, 0x044, [data]

Operating System

Program Event Queue

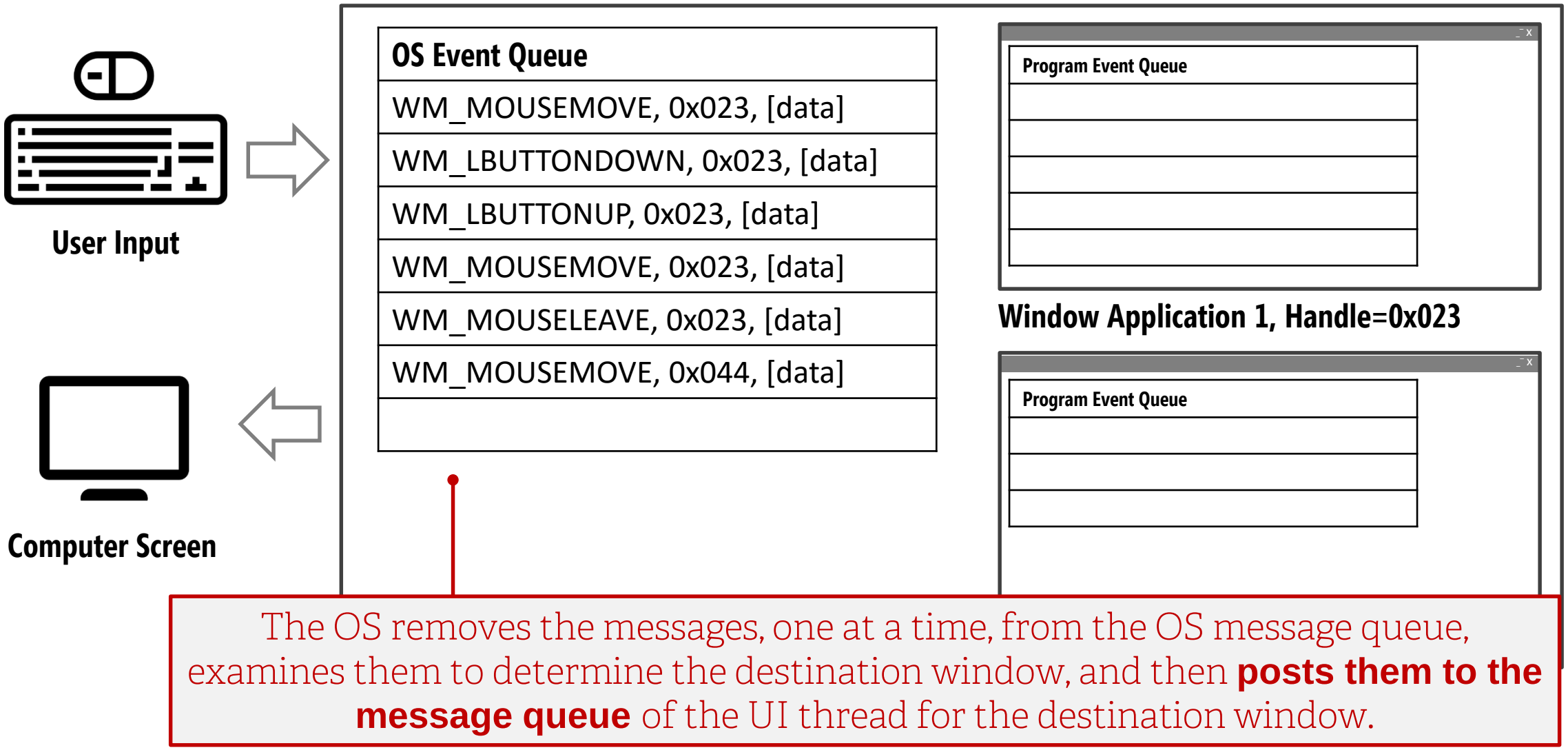
Window Application 1, Handle=0x023

Program Event Queue

Window Application 2, Handle=0x044

Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

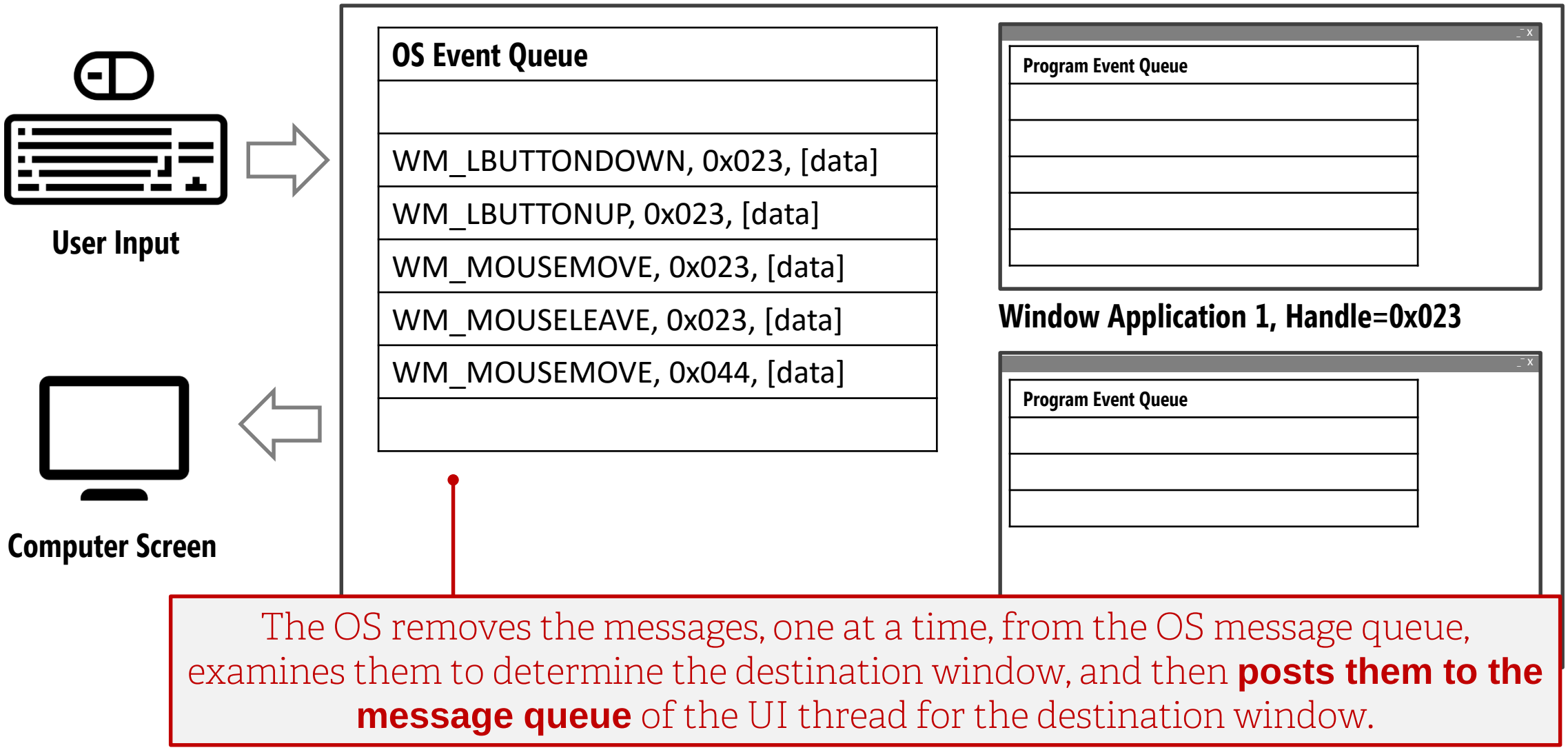
Event Processing Diagram



The OS removes the messages, one at a time, from the OS message queue, examines them to determine the destination window, and then **posts them to the message queue** of the UI thread for the destination window.

Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

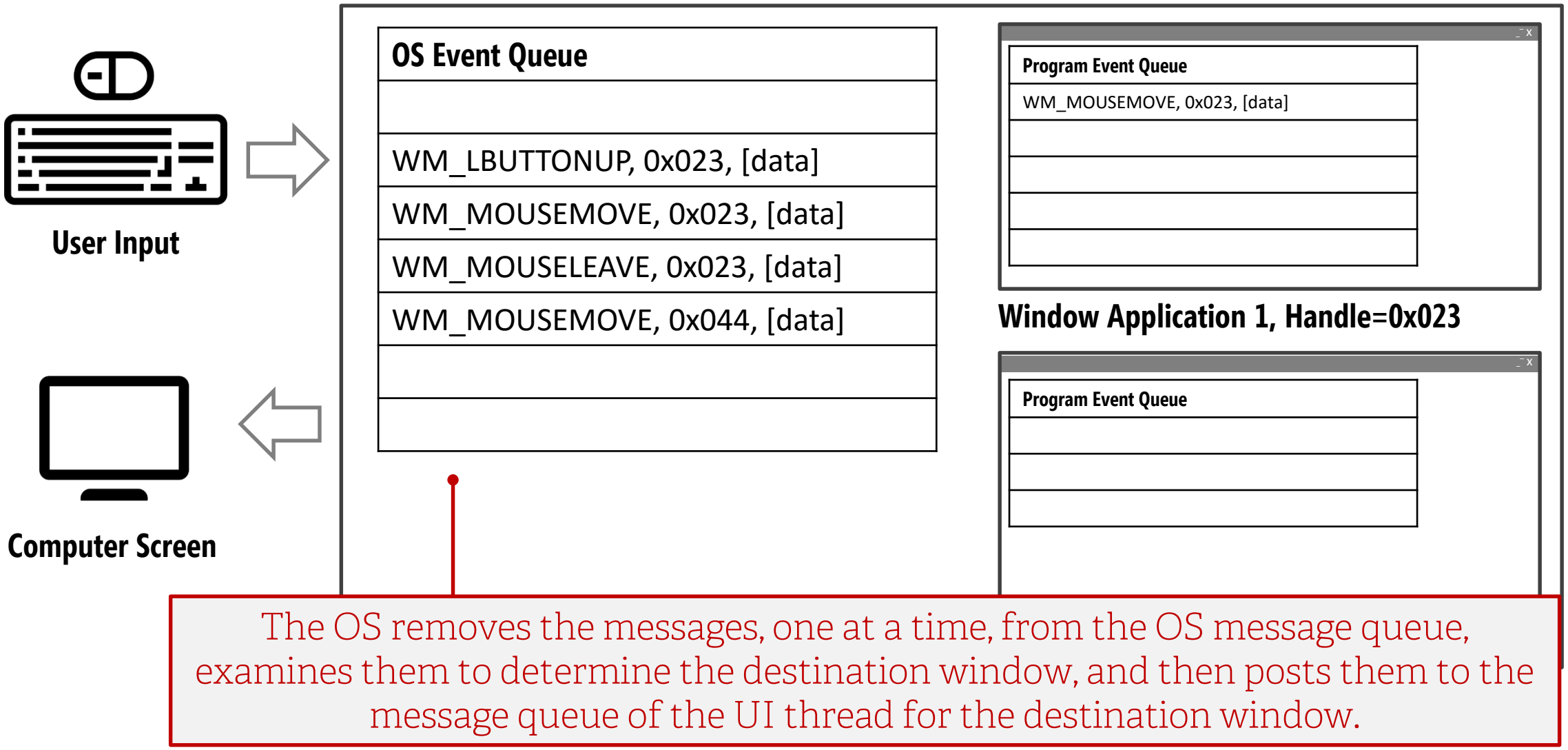
Event Processing Diagram



The OS removes the messages, one at a time, from the OS message queue, examines them to determine the destination window, and then **posts them to the message queue** of the UI thread for the destination window.

Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

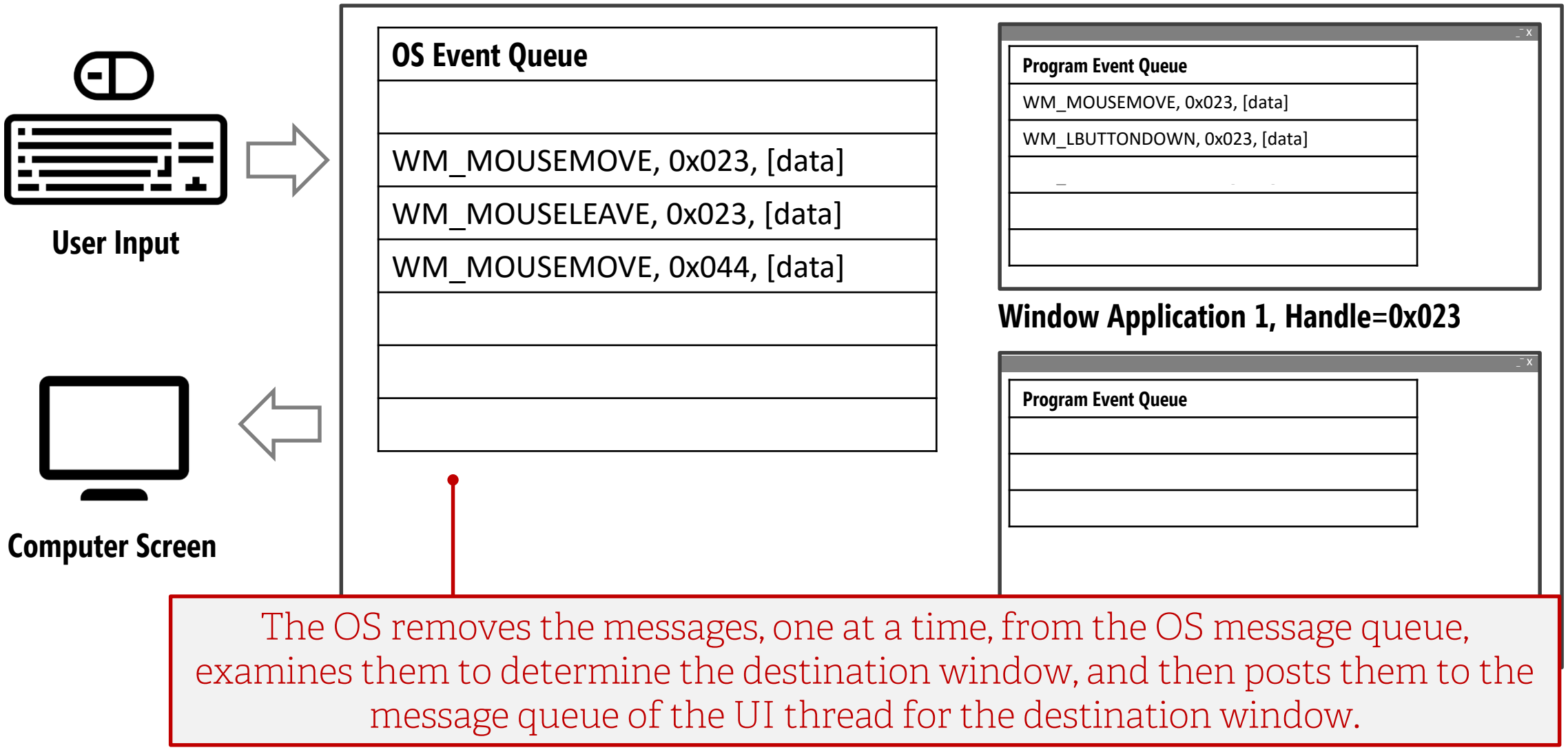
Event Processing Diagram



The OS removes the messages, one at a time, from the OS message queue, examines them to determine the destination window, and then posts them to the message queue of the UI thread for the destination window.

Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

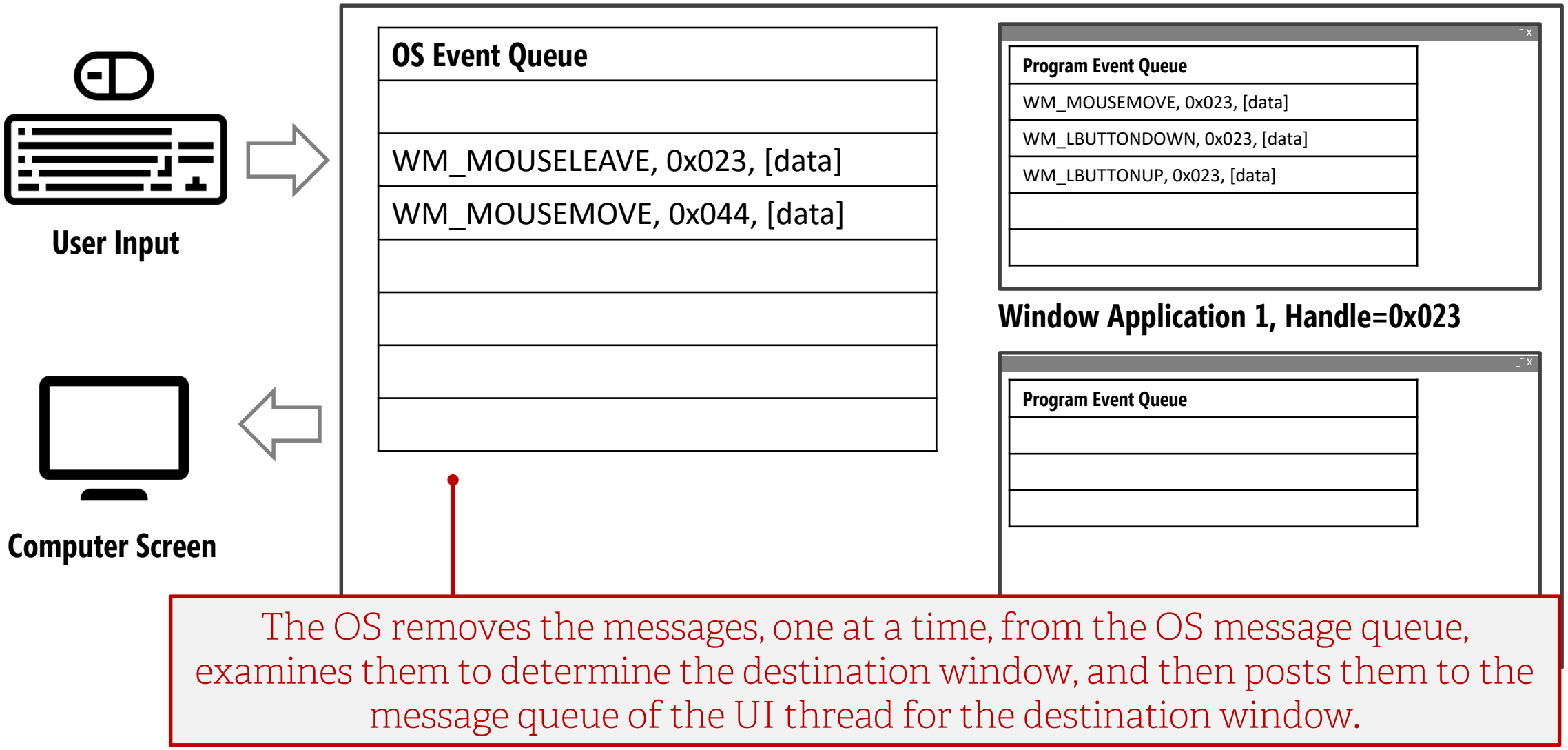
Event Processing Diagram



The OS removes the messages, one at a time, from the OS message queue, examines them to determine the destination window, and then posts them to the message queue of the UI thread for the destination window.

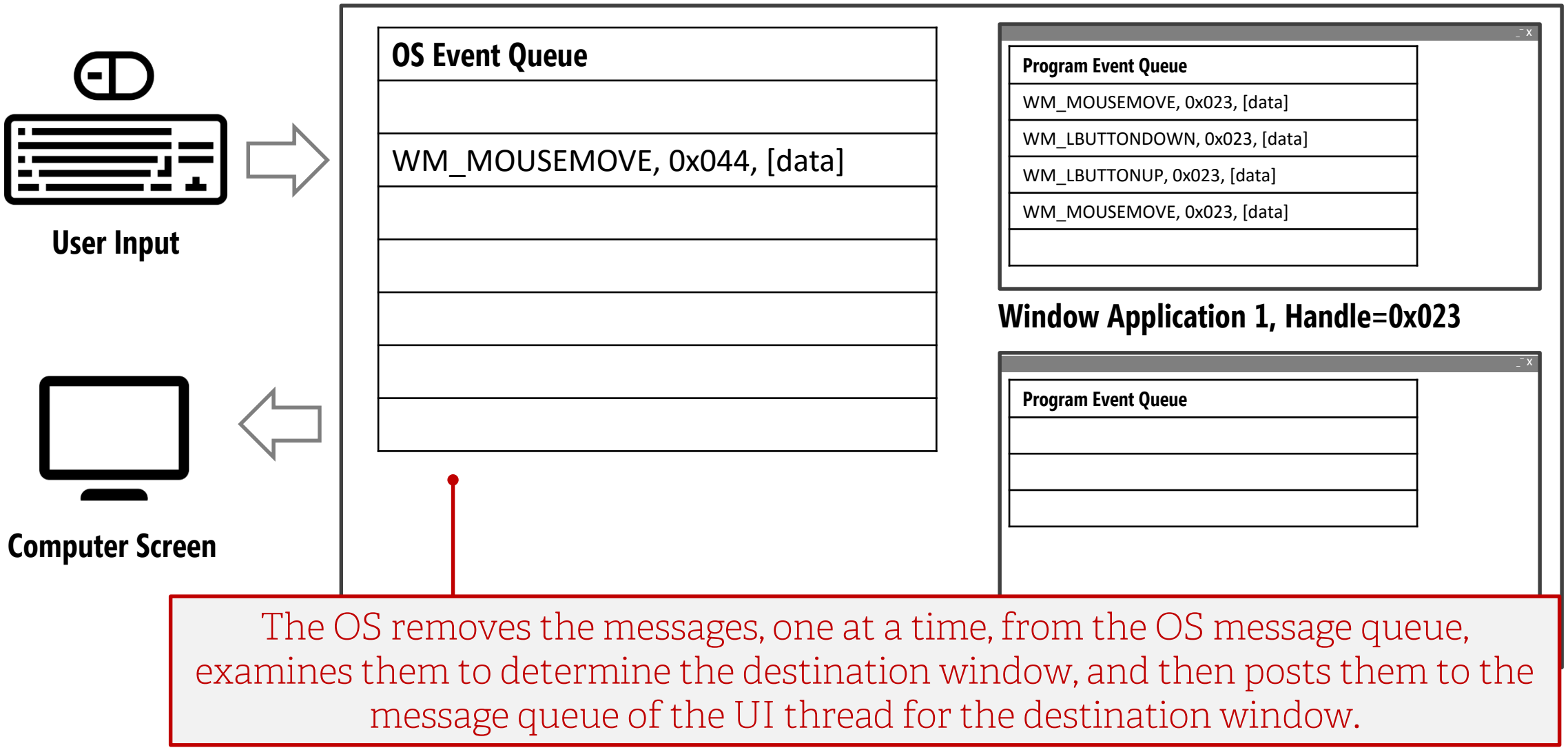
Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Event Processing Diagram



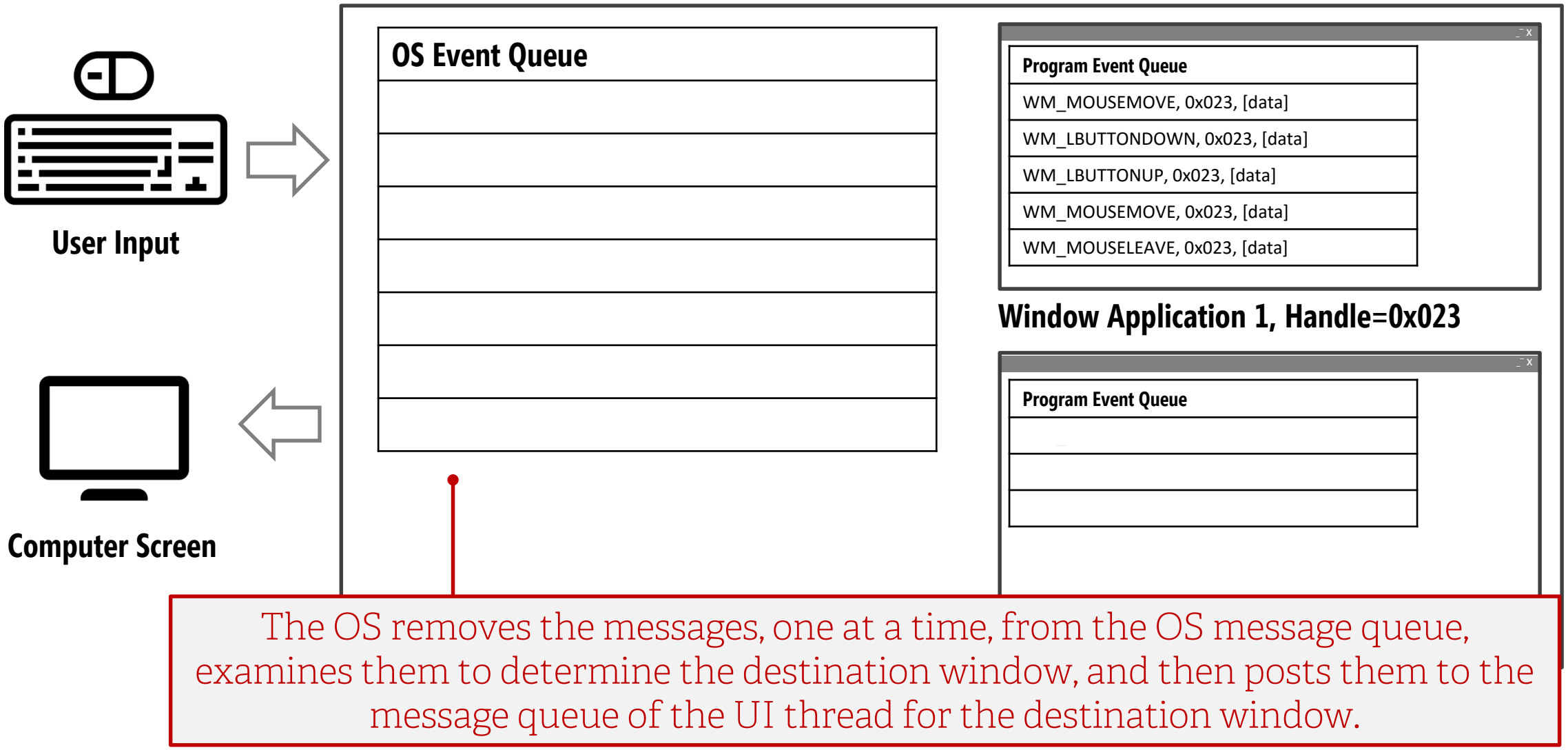
Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Event Processing Diagram



Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

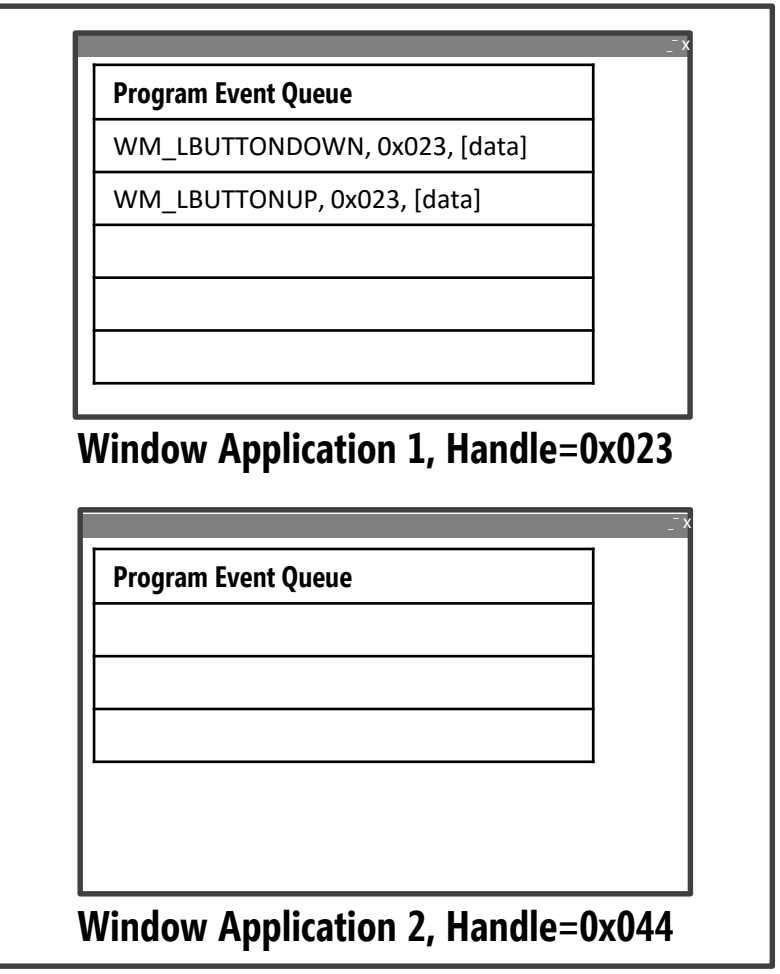
Event Processing Diagram



The OS removes the messages, one at a time, from the OS message queue, examines them to determine the destination window, and then posts them to the message queue of the UI thread for the destination window.

Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

Event Processing Queue

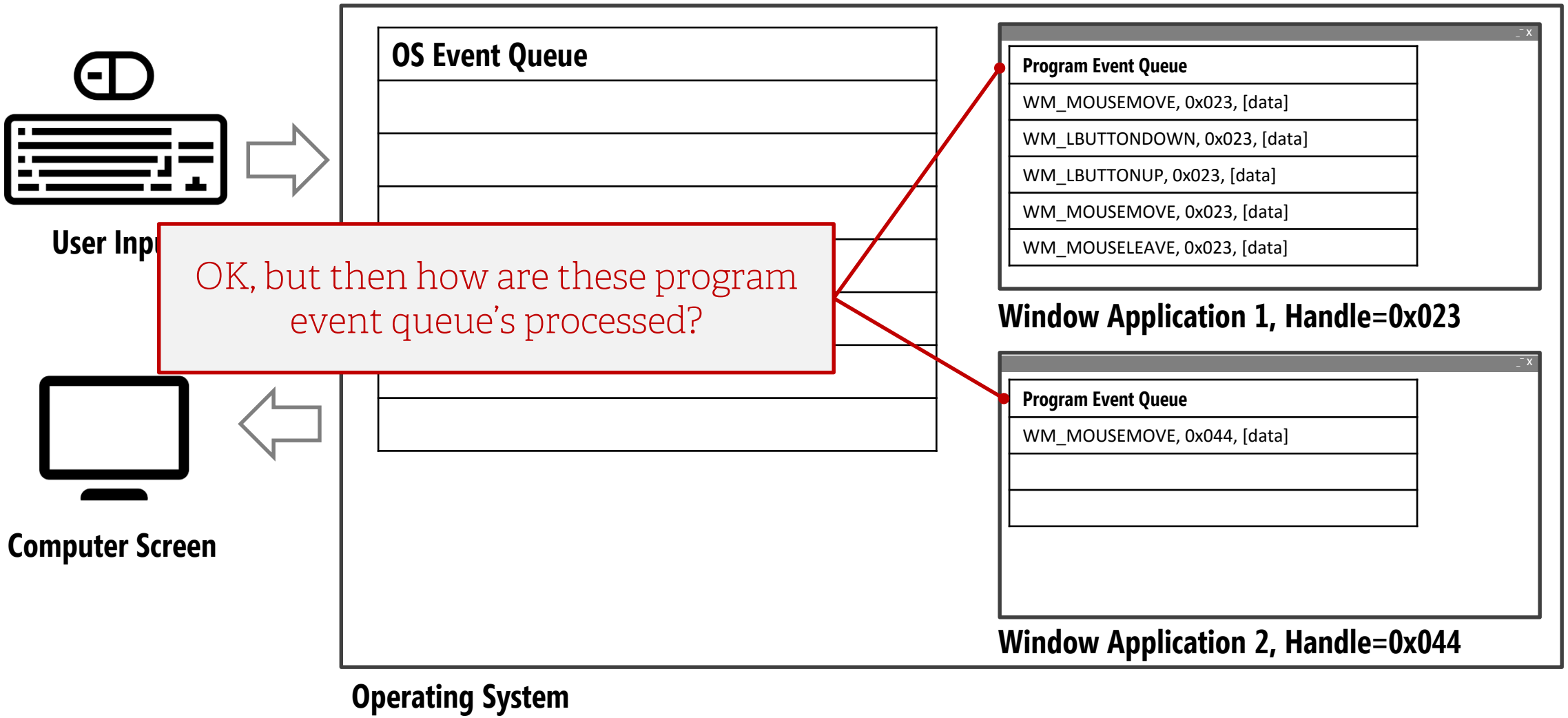


OS always posts messages to the end of a program's message queue (*i.e.*, FIFO).

The program message queue processes the event messages in FIFO

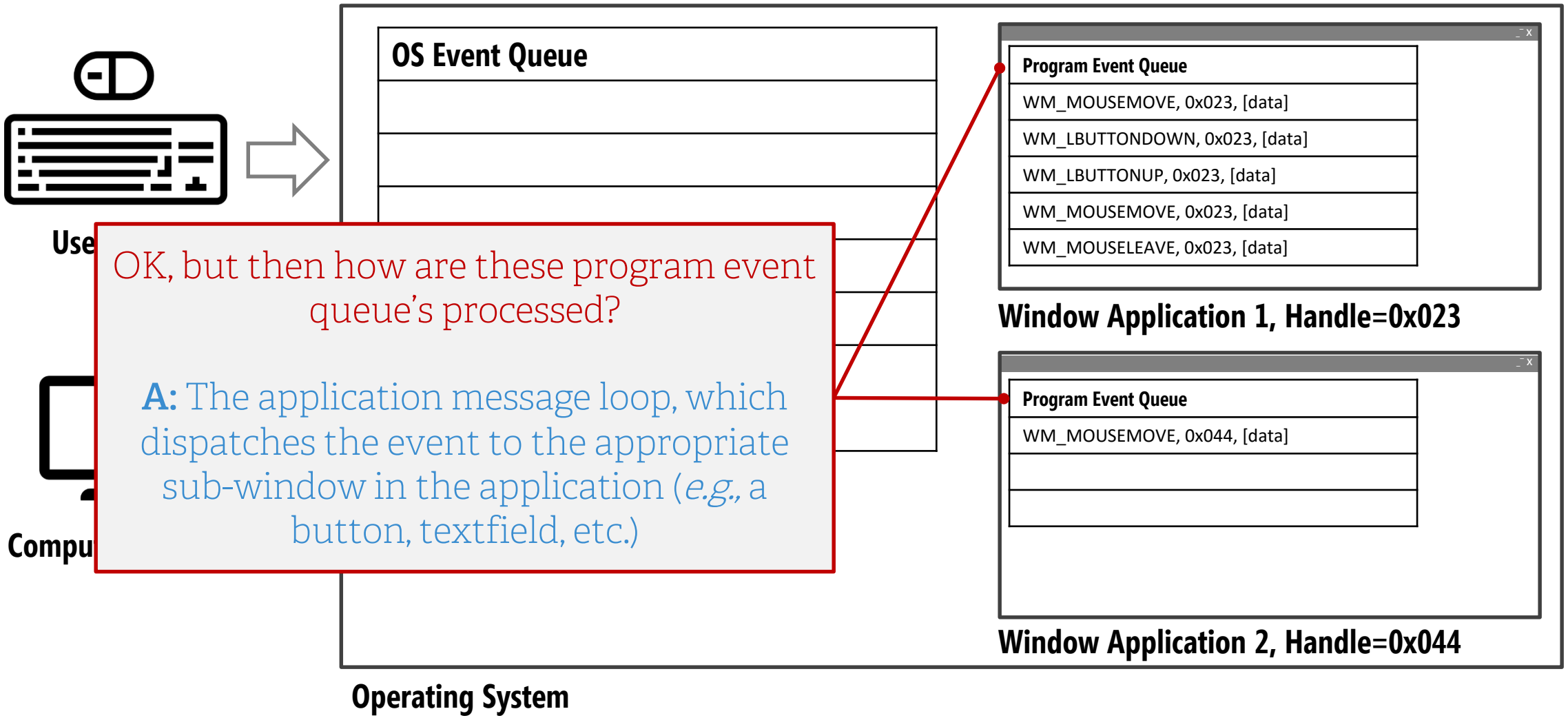
However, some events receive special handling. For example, on Windows, WM_PAINT messages are only processed when the queue contains no other messages. In addition, for efficiency, multiple WM_PAINT messages in the queue are consolidated into one.

Event Processing Diagram



Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

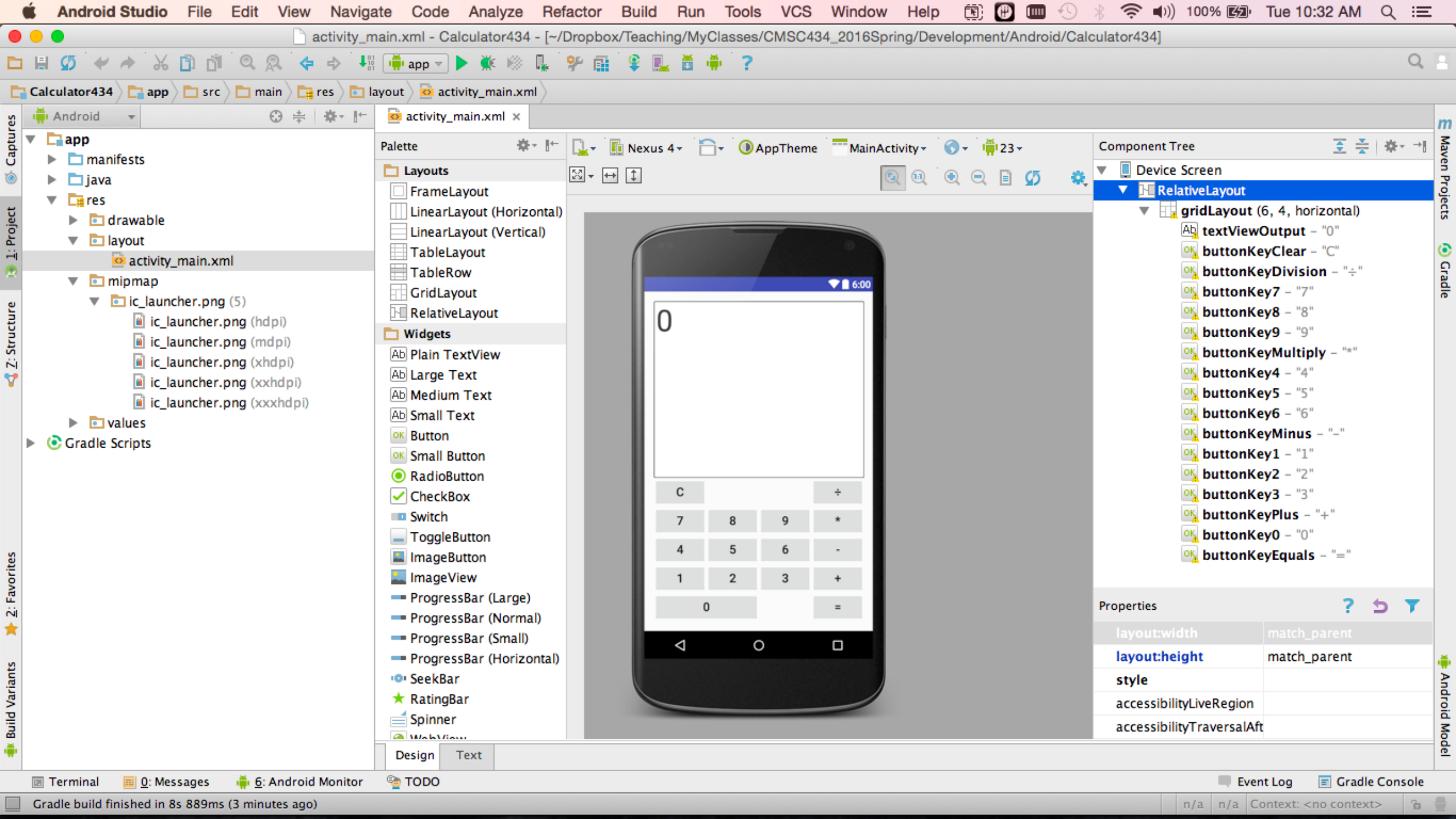
Event Processing Diagram

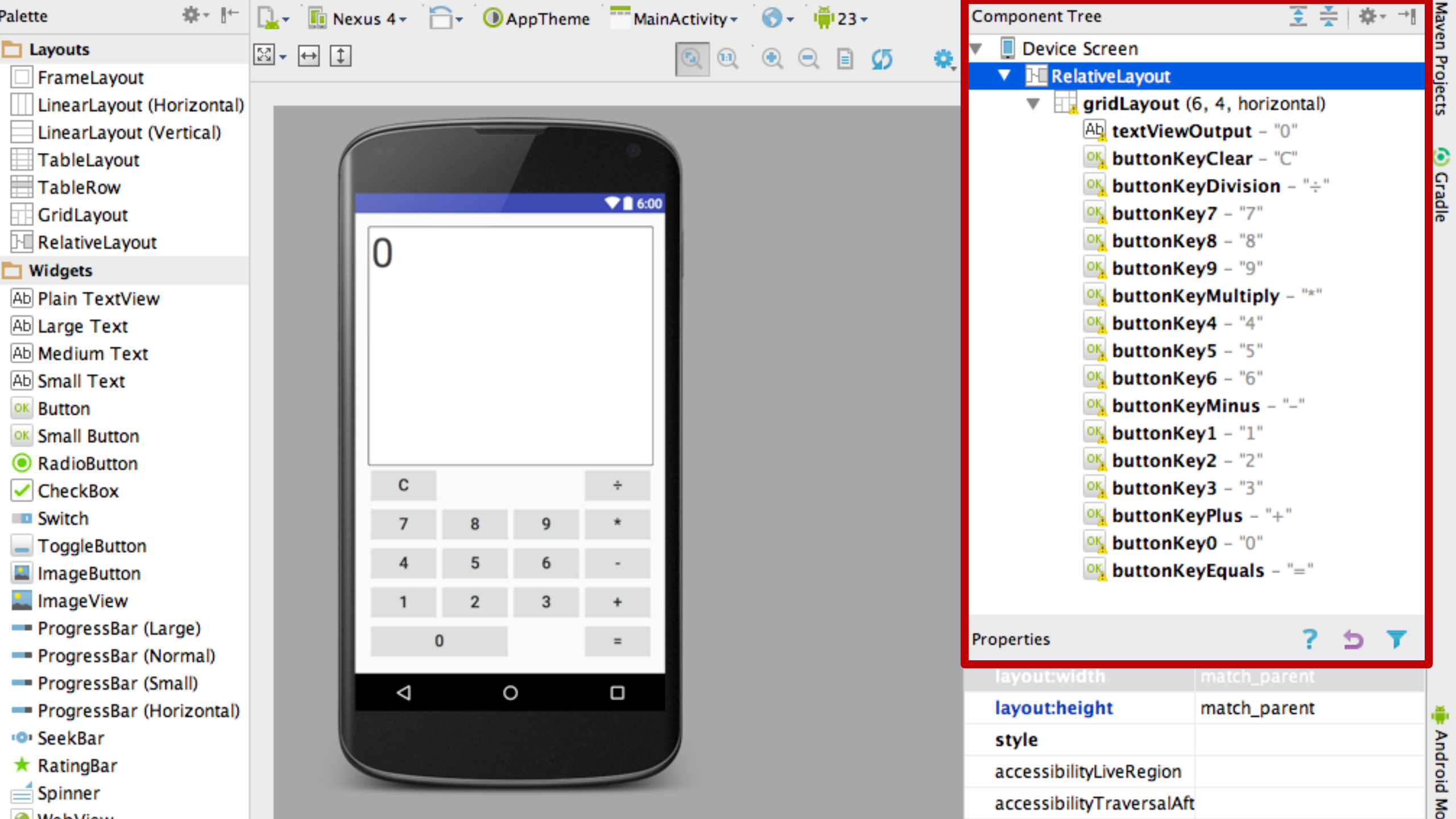


Source: Windows Message Queue, https://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx

To understand this part, we need to return to the hierarchical nature of UI windows—**Window Trees**.

Android calls this **Component Tree**

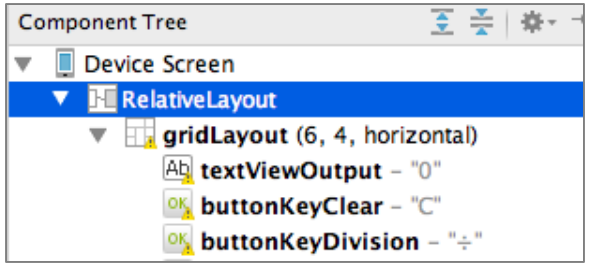

Android Studio File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
activity_main.xml - Calculator434 - [~/Dropbox/Teaching/MyClasses/CMSC434_2016Spring/Development/Android/Calculator434]
Calculator434 app src main res layout activity_main.xml
Android
app
manifests
java
res
drawable
layout
activity_main.xml
mipmap
ic_launcher.png (5)
ic_launcher.png (hdpi)
ic_launcher.png (mdpi)
ic_launcher.png (xhdpi)
ic_launcher.png (xxhdpi)
ic_launcher.png (xxxhdpi)
values
Gradle Scripts
activity_main.xml
Palette
Layouts
FrameLayout
LinearLayout (Horizontal)
LinearLayout (Vertical)
TableLayout
TableRow
GridLayout
RelativeLayout
Widgets
Plain TextView
Large Text
Medium Text
Small Text
Button
Small Button
RadioButton
CheckBox
Switch
ToggleButton
ImageButton
ImageView
ProgressBar (Large)
ProgressBar (Normal)
ProgressBar (Small)
ProgressBar (Horizontal)
SeekBar
RatingBar
Spinner
Nexus 4
AppTheme
MainActivity
23
0
C ÷
7 8 9 *
4 5 6 -
1 2 3 +
0 =
Component Tree
Device Screen
RelativeLayout
gridLayout (6, 4, horizontal)
textViewOutput - "0"
buttonKeyClear - "C"
buttonKeyDivision - "÷"
buttonKey7 - "7"
buttonKey8 - "8"
buttonKey9 - "9"
buttonKeyMultiply - "*"
buttonKey4 - "4"
buttonKey5 - "5"
buttonKey6 - "6"
buttonKeyMinus - "-"
buttonKey1 - "1"
buttonKey2 - "2"
buttonKey3 - "3"
buttonKeyPlus - "+"
buttonKey0 - "0"
buttonKeyEquals - "="
Properties
layout:width match_parent
layout:height match_parent
style
accessibilityLiveRegion
accessibilityTraversalAft
Design Text
Terminal 0: Messages 6: Android Monitor TODO
Event Log Gradle Console
Gradle build finished in 8s 889ms (3 minutes ago)
n/a n/a Context: <no context>


Palette
Layouts
FrameLayout
LinearLayout (Horizontal)
LinearLayout (Vertical)
TableLayout
TableRow
GridLayout
RelativeLayout
Widgets
Plain TextView
Large Text
Medium Text
Small Text
Button
Small Button
RadioButton
CheckBox
Switch
ToggleButton
ImageButton
ImageView
ProgressBar (Large)
ProgressBar (Normal)
ProgressBar (Small)
ProgressBar (Horizontal)
SeekBar
RatingBar
Spinner
Nexus 4
AppTheme
MainActivity
23
6:00
0
C
÷
7
8
9
*
4
5
6
-
1
2
3
+
0
=
Component Tree
Device Screen
RelativeLayout
gridLayout (6, 4, horizontal)
textViewOutput - "0"
buttonKeyClear - "C"
buttonKeyDivision - "÷"
buttonKey7 - "7"
buttonKey8 - "8"
buttonKey9 - "9"
buttonKeyMultiply - "*"
buttonKey4 - "4"
buttonKey5 - "5"
buttonKey6 - "6"
buttonKeyMinus - "-"
buttonKey1 - "1"
buttonKey2 - "2"
buttonKey3 - "3"
buttonKeyPlus - "+"
buttonKey0 - "0"
buttonKeyEquals - "="
Properties
layout:width
match_parent
layout:height
match_parent
style
accessibilityLiveRegion
accessibilityTraversalAft
Maven Projects
Gradle
Android Mo

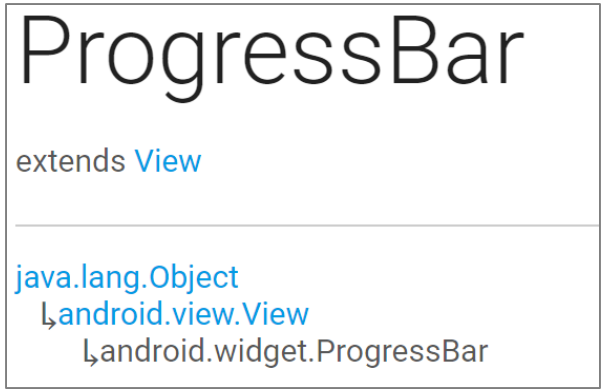
Window Tree vs. UI Object Hierarchy

Remember, the Window Tree is **completely different** from the UI object hierarchy

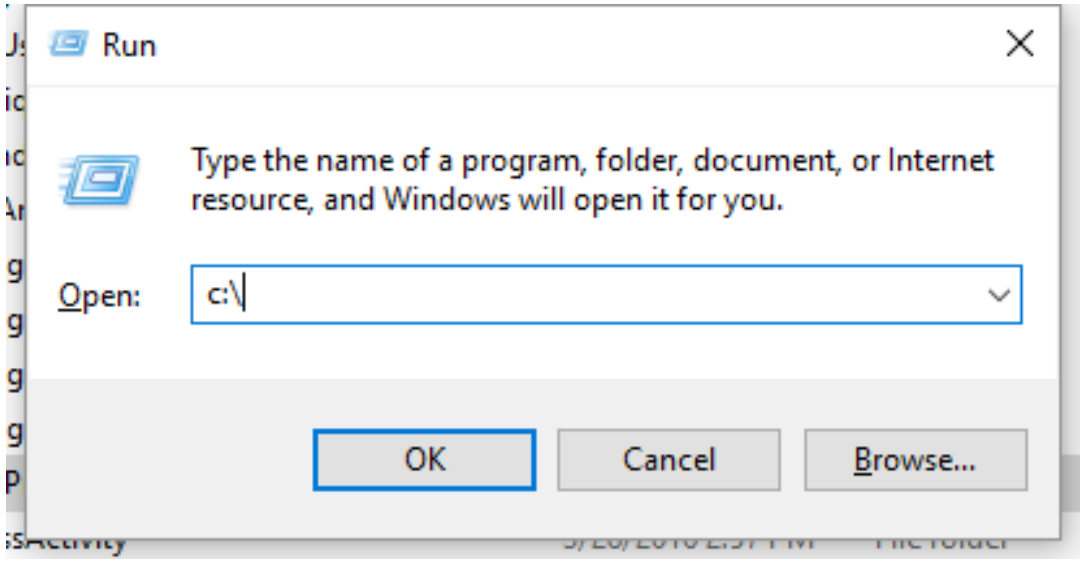
The **Window Tree** describes the relationship between UI components laid out in a window



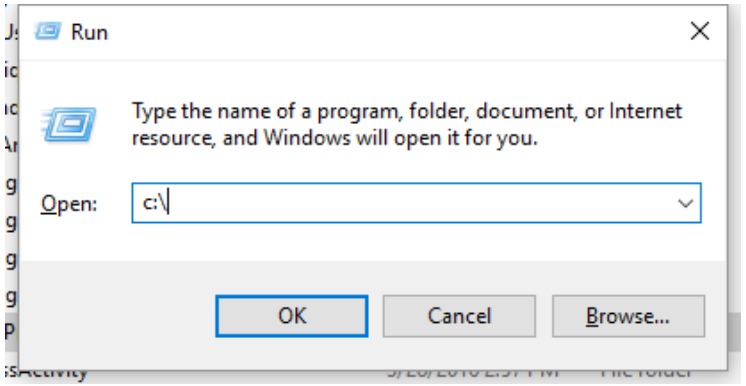
The **UI Object Hierarchy** describes the class hierarchy that UI frameworks use.

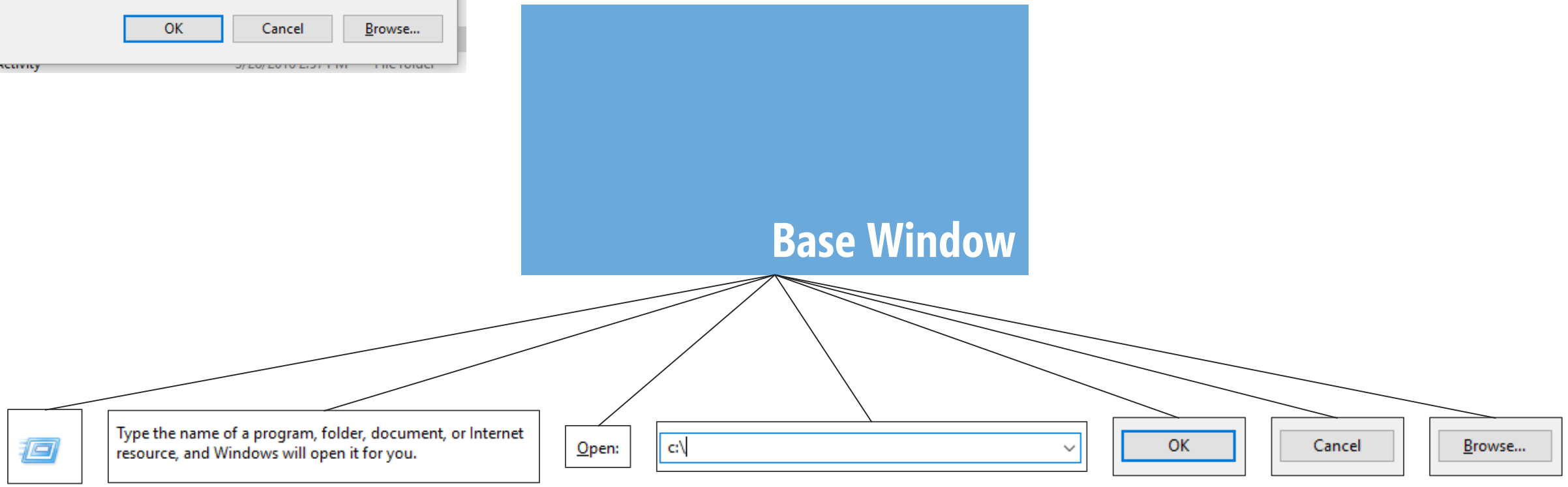


Window Tree Example



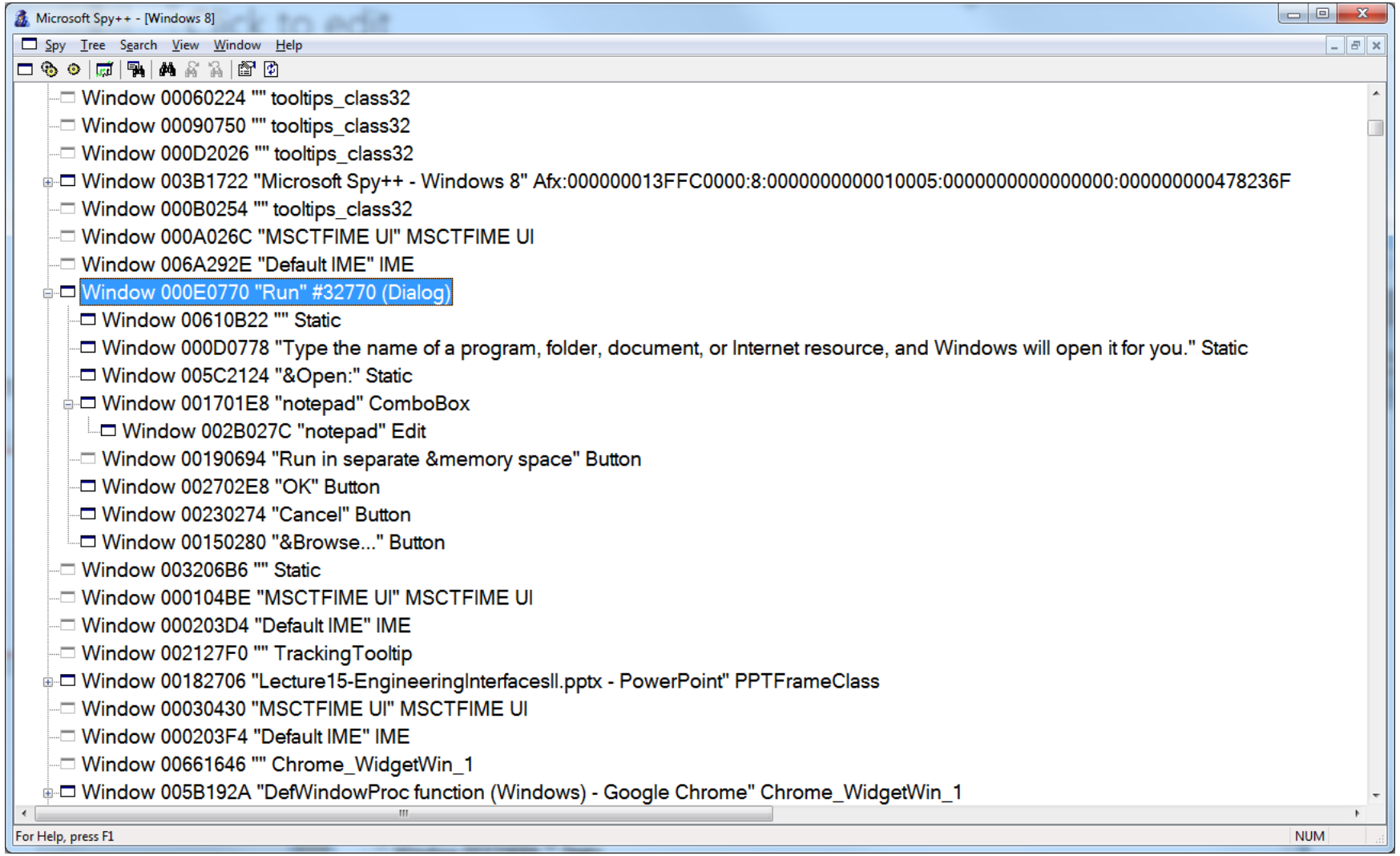
Window Tree Example





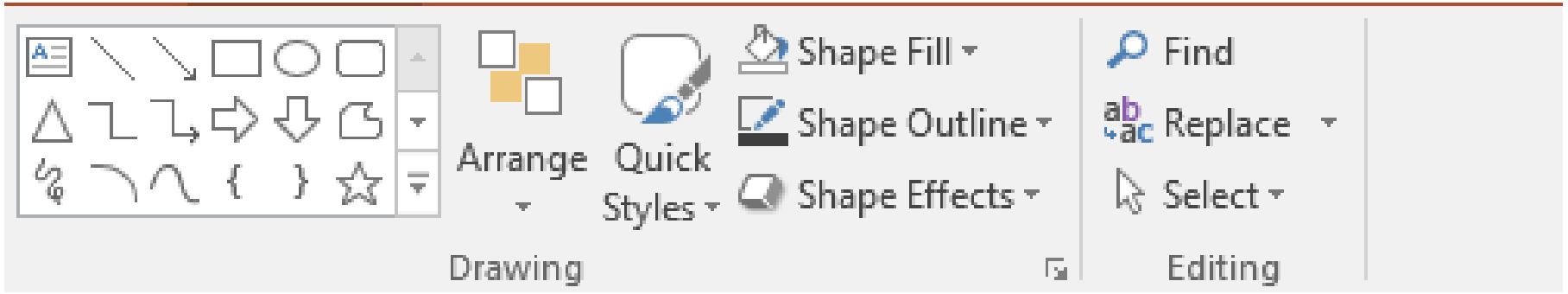
Actual Window Tree for MS Windows Run Dialog

The actual Window Tree of the Run Dialog as observed from MS Spy++

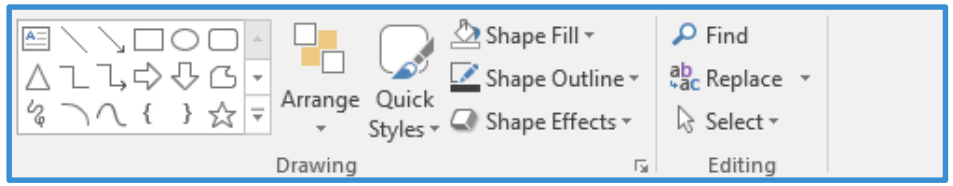


Make Window Tree

Take out your sketchbooks, and make a Window Tree of the following

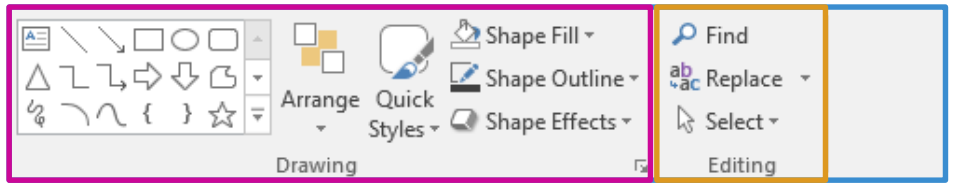


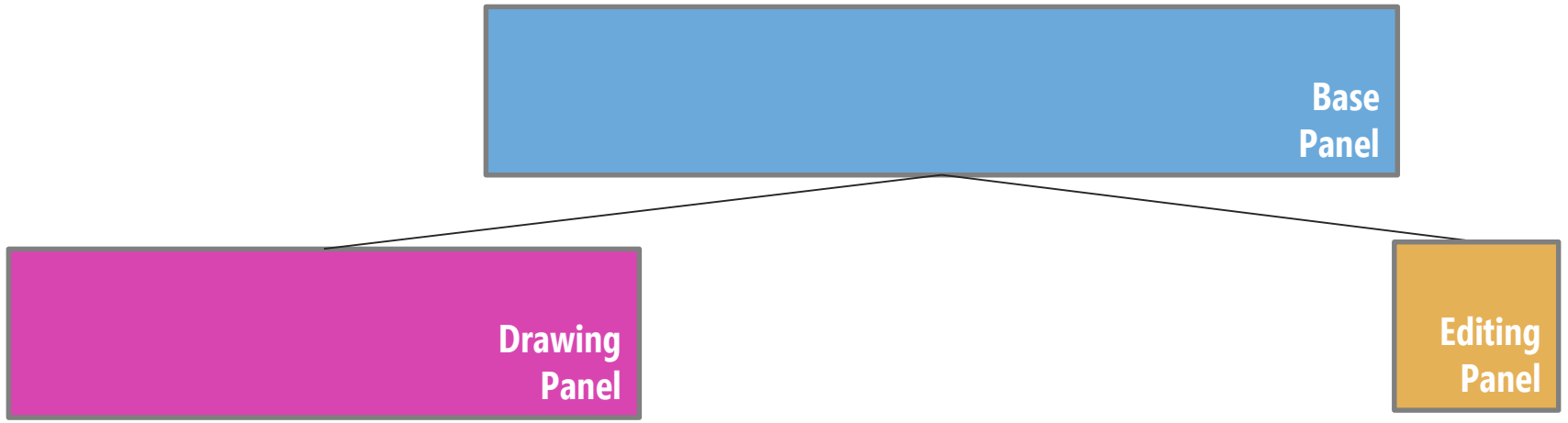
Window Tree Example



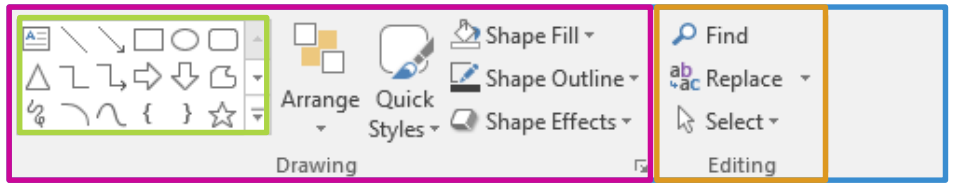
Base Panel

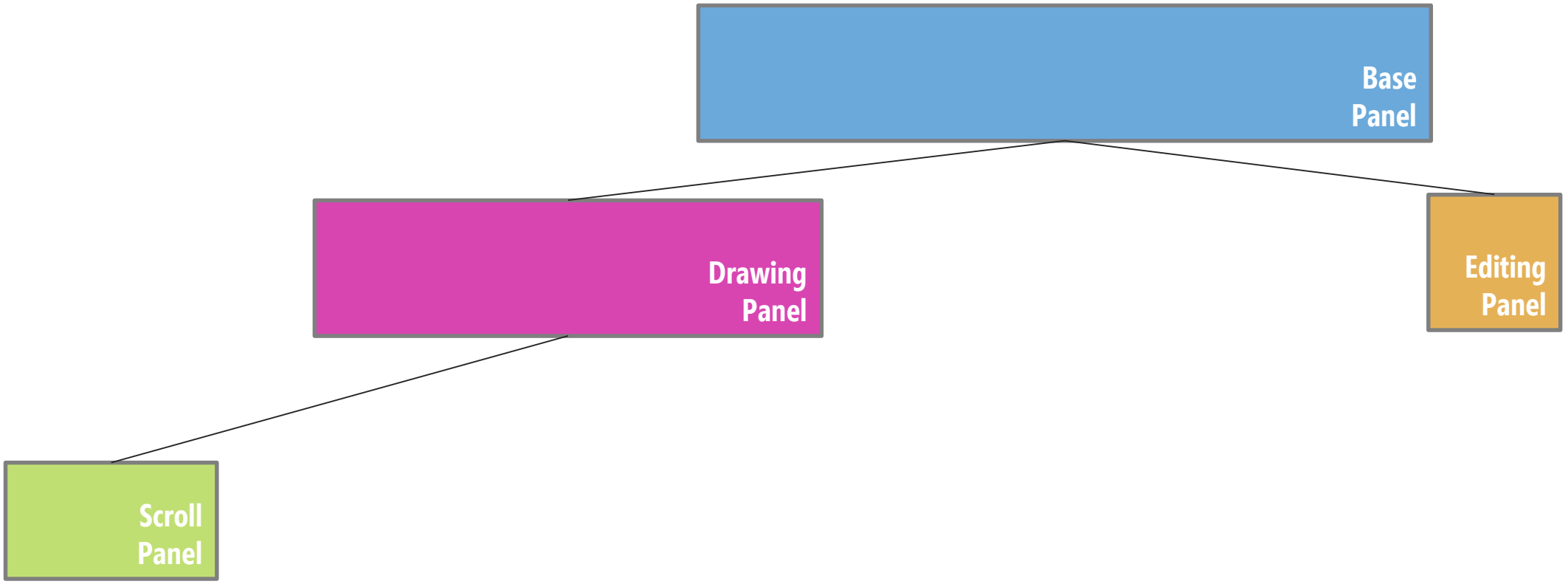
Window Tree Example



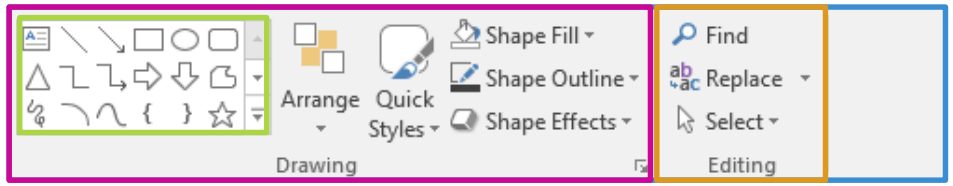


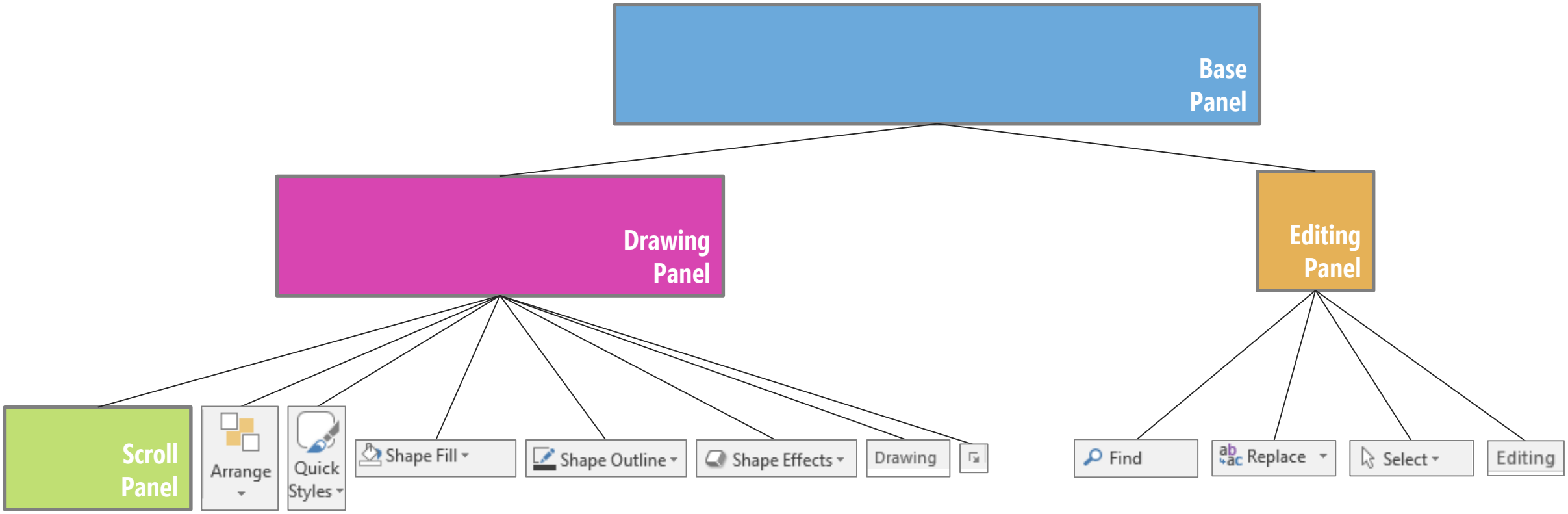
Window Tree Example



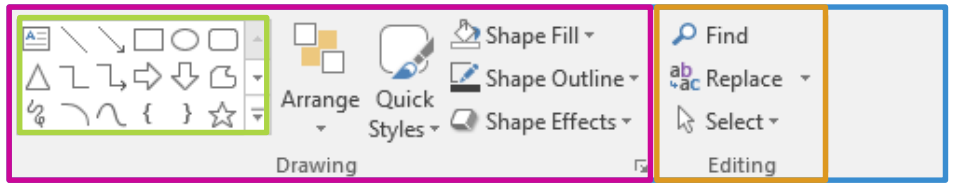


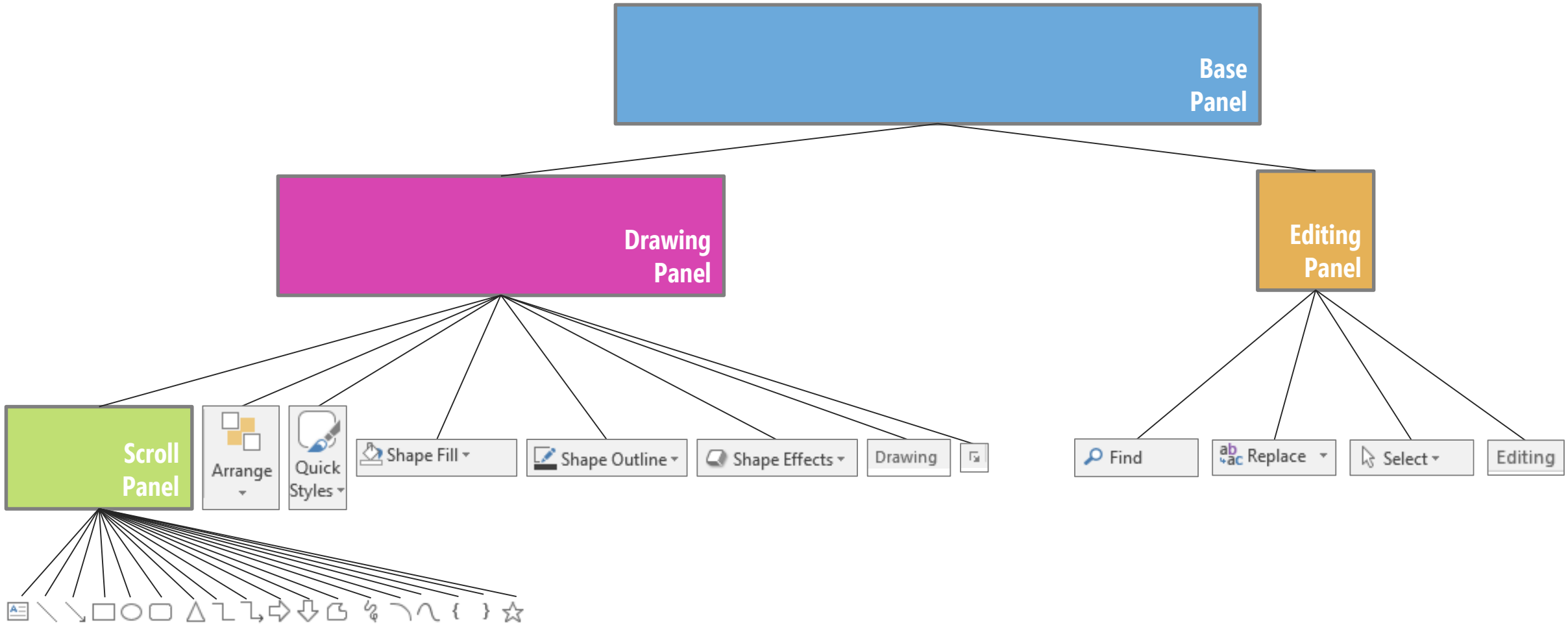
Window Tree Example



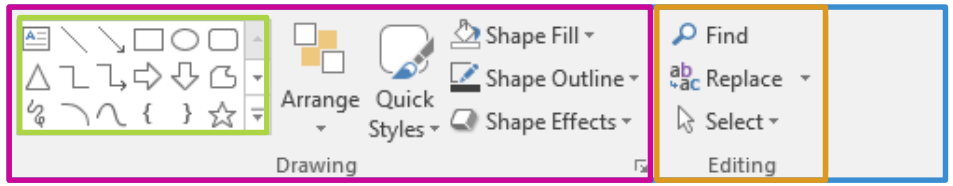


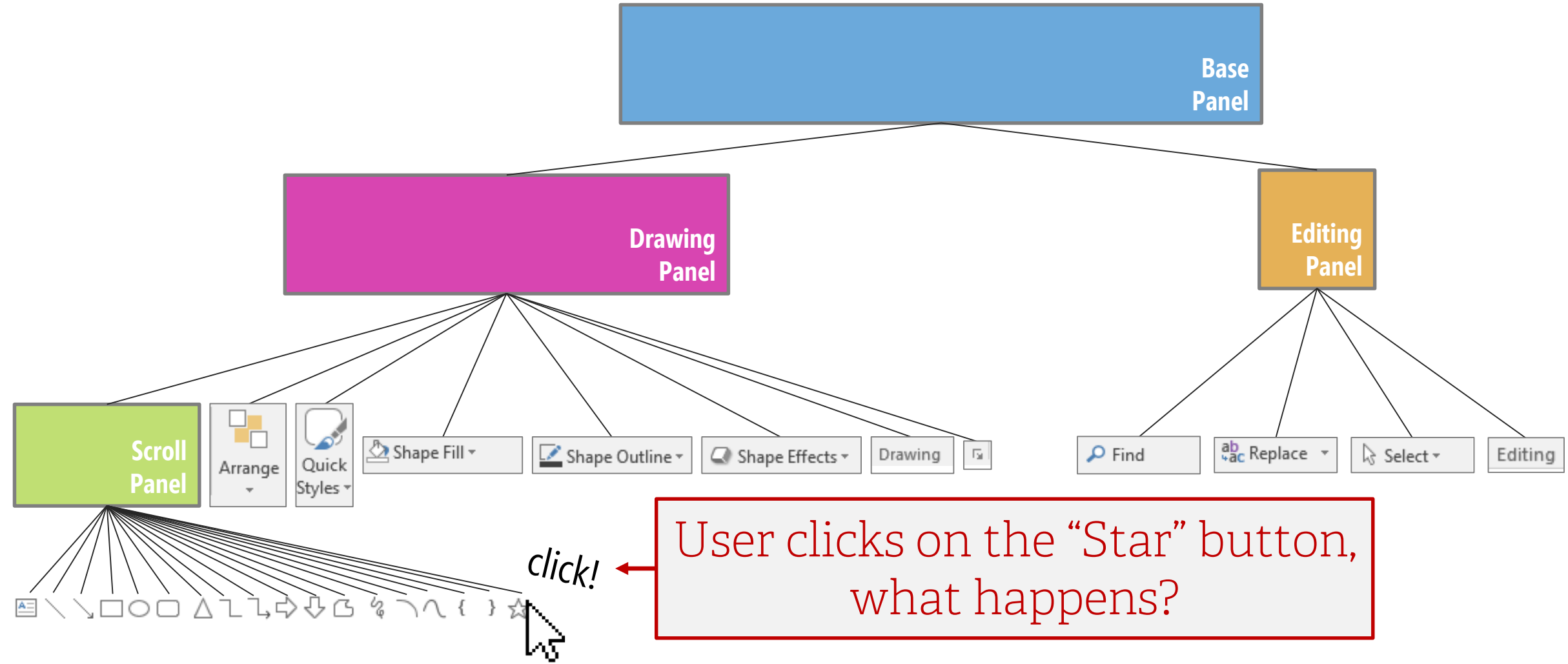
Window Tree Example





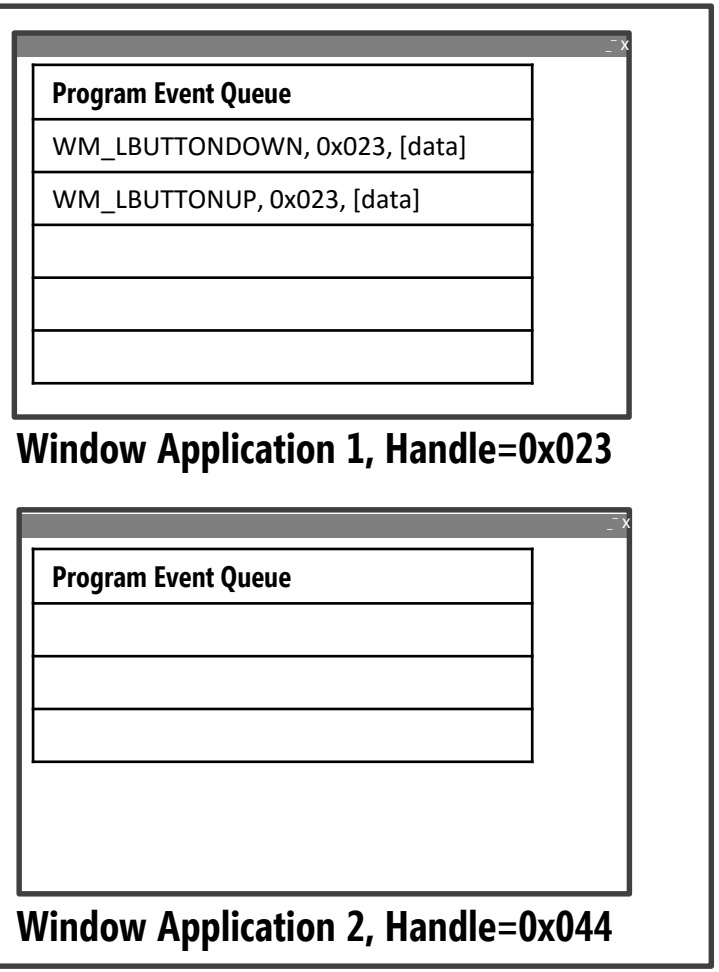
Window Tree Example





User clicks on the "Star" button, what happens?

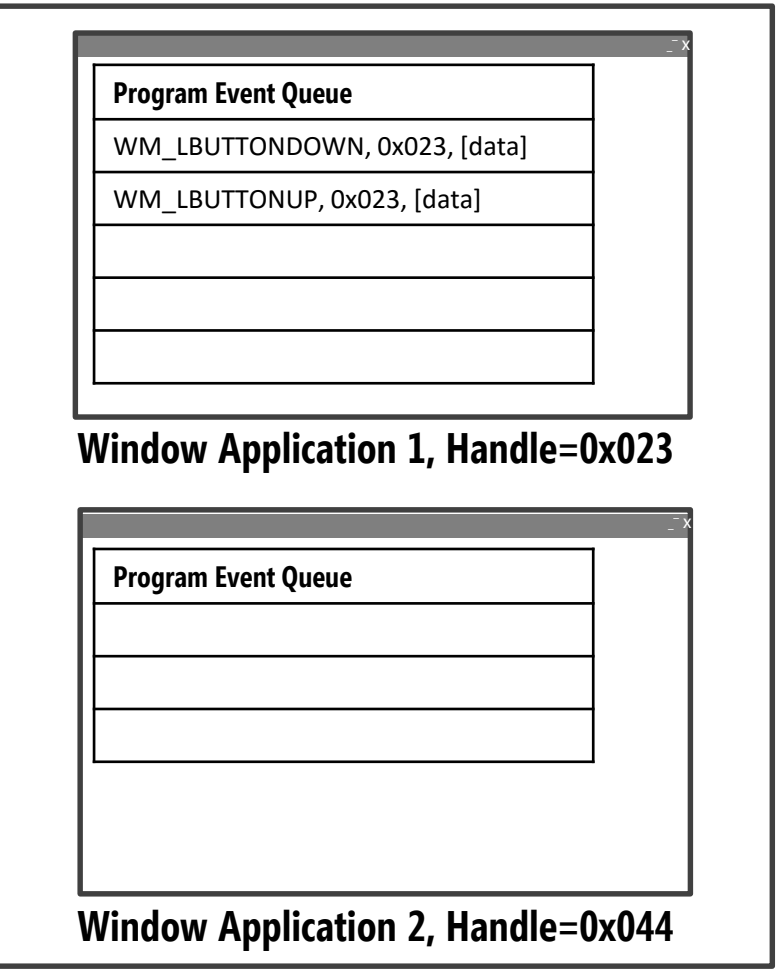
Two Event Routing Approaches



Application processes the event queue in the UI thread message loop, but how does it know which window to send the event to?

1. Sub-window handles. In the Windows OS, every single component drawn to the screen is a "window" and has its own window handle (address), so the application uses that for message routing (as we just saw!).
2. Hit testing. The application uses "hit testing" to check for the top-most window component (z-axis) (which is also the bottom most component in window tree) and routes the message there.

Two Event Routing Approaches



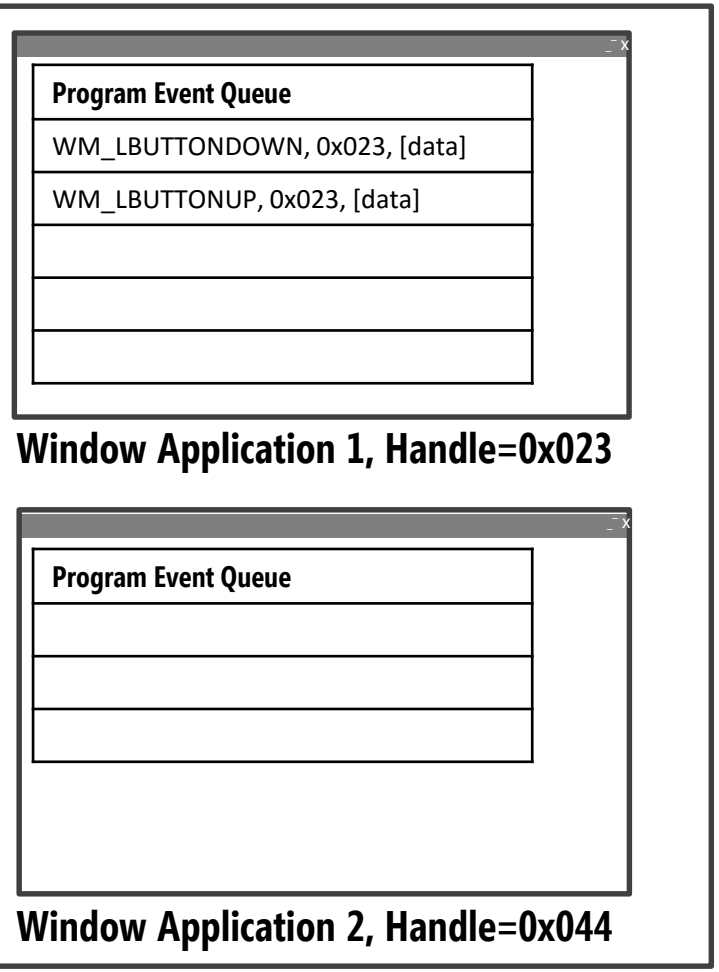
Application processes the event queue in the UI thread message loop, but how does it know which window to send the event to?

1. **Sub-window handles.** In the Windows OS, every single component drawn to the screen is a "window" and has its own window handle (address), so the application uses that for message routing (as we just saw!).

2. **Hit testing.** The application uses "hit testing" to check for the top-most window component (z-axis) (which is also the bottom most component in window tree) and routes the message there.

Regardless of method, are input events processed bottom-up or top-down in Window Tree?

Two Event Routing Approaches

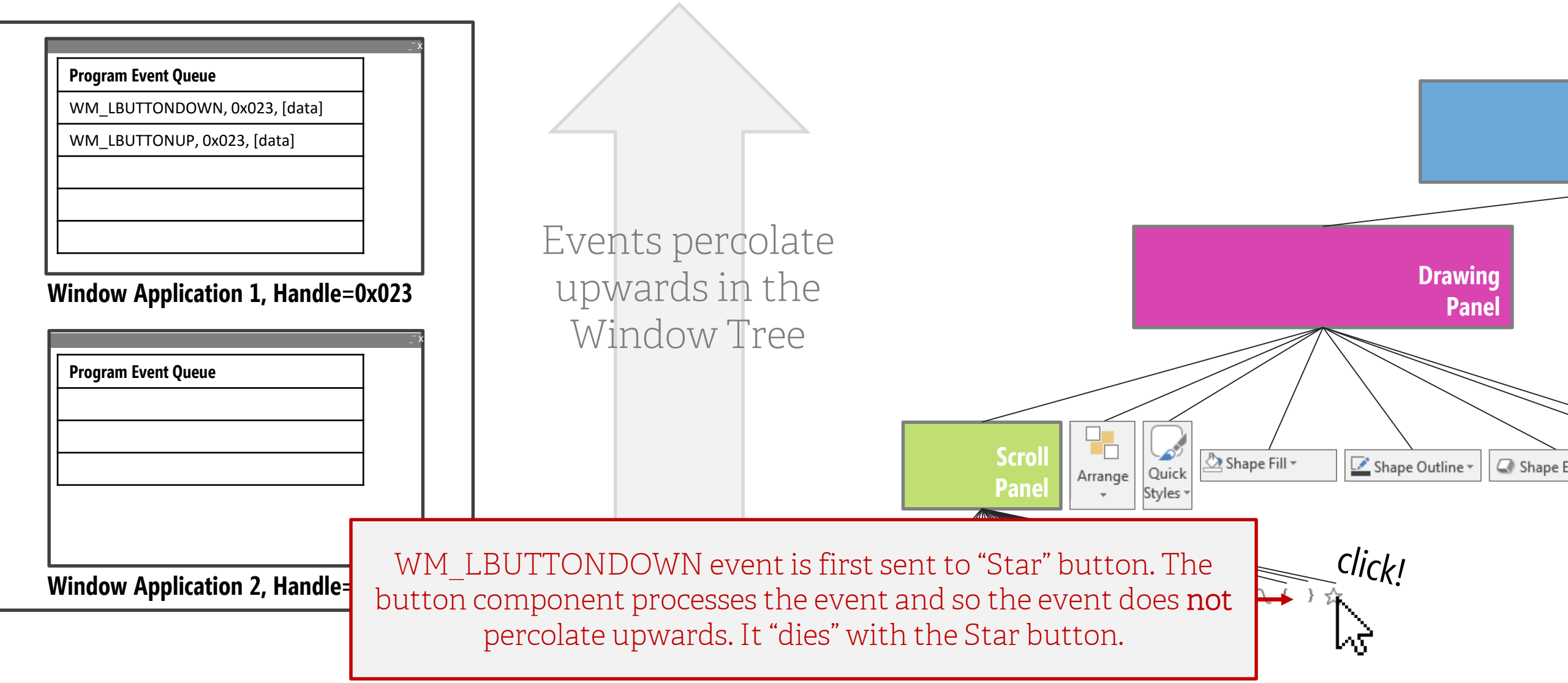


Application processes the event queue in the UI thread message loop, but how does it know which window to send the event to?

1. Sub-window handles. In the Windows OS, every single component drawn to the screen is a "window" and has its own window handle (address), so the application uses that for message routing (as we just saw!).
2. Hit testing. The application uses "hit testing" to check for the top-most window component (z-axis) (which is also the bottom most component in window tree) and routes the message there.

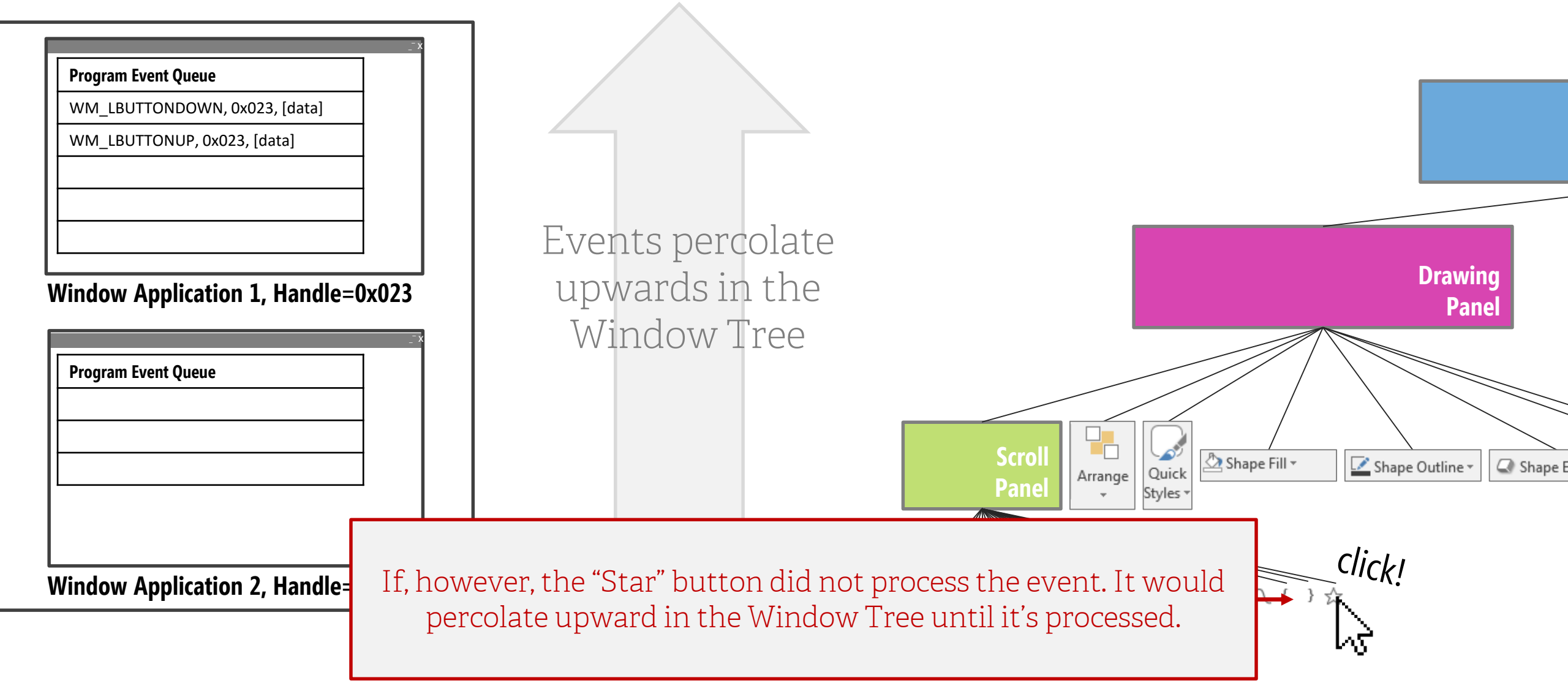
A: Events are always processed bottom-up

Event Processing Diagram

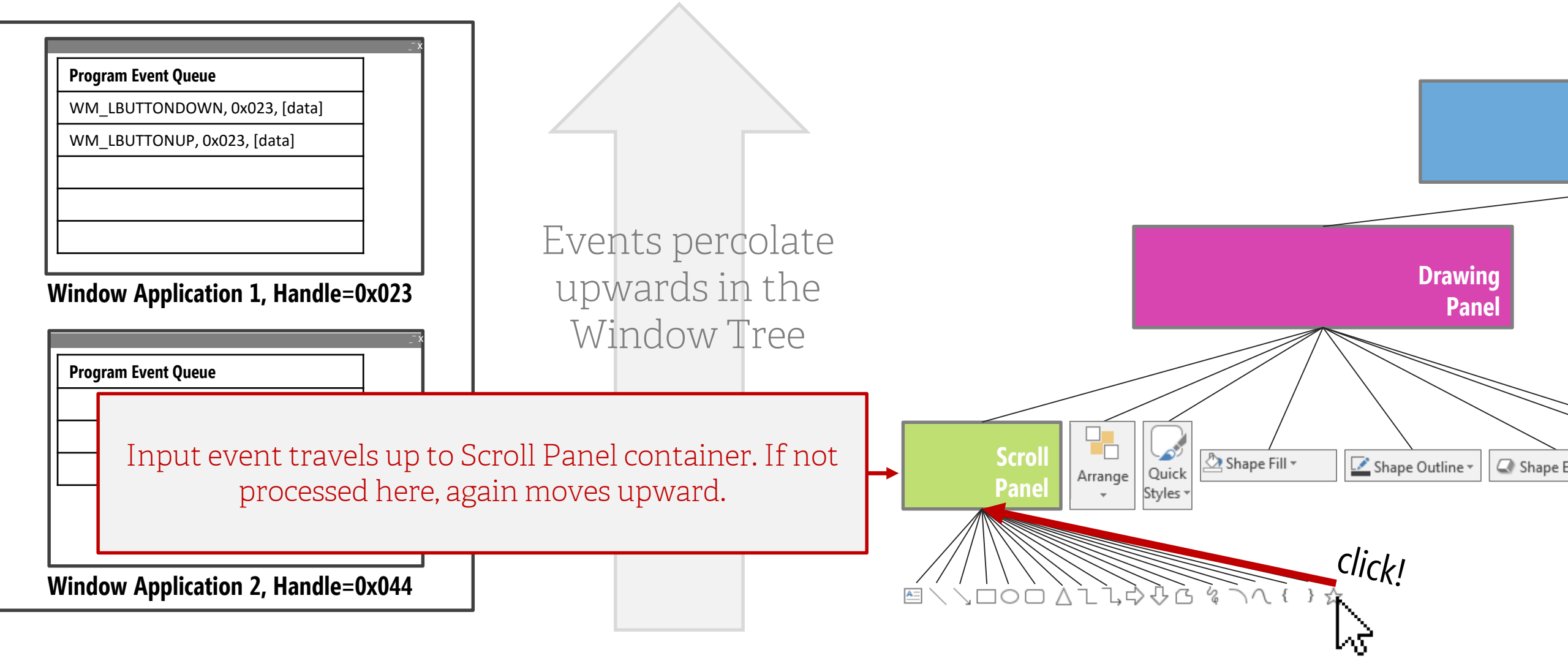


WM_LBUTTONDOWN event is first sent to "Star" button. The button component processes the event and so the event does **not** percolate upwards. It "dies" with the Star button.

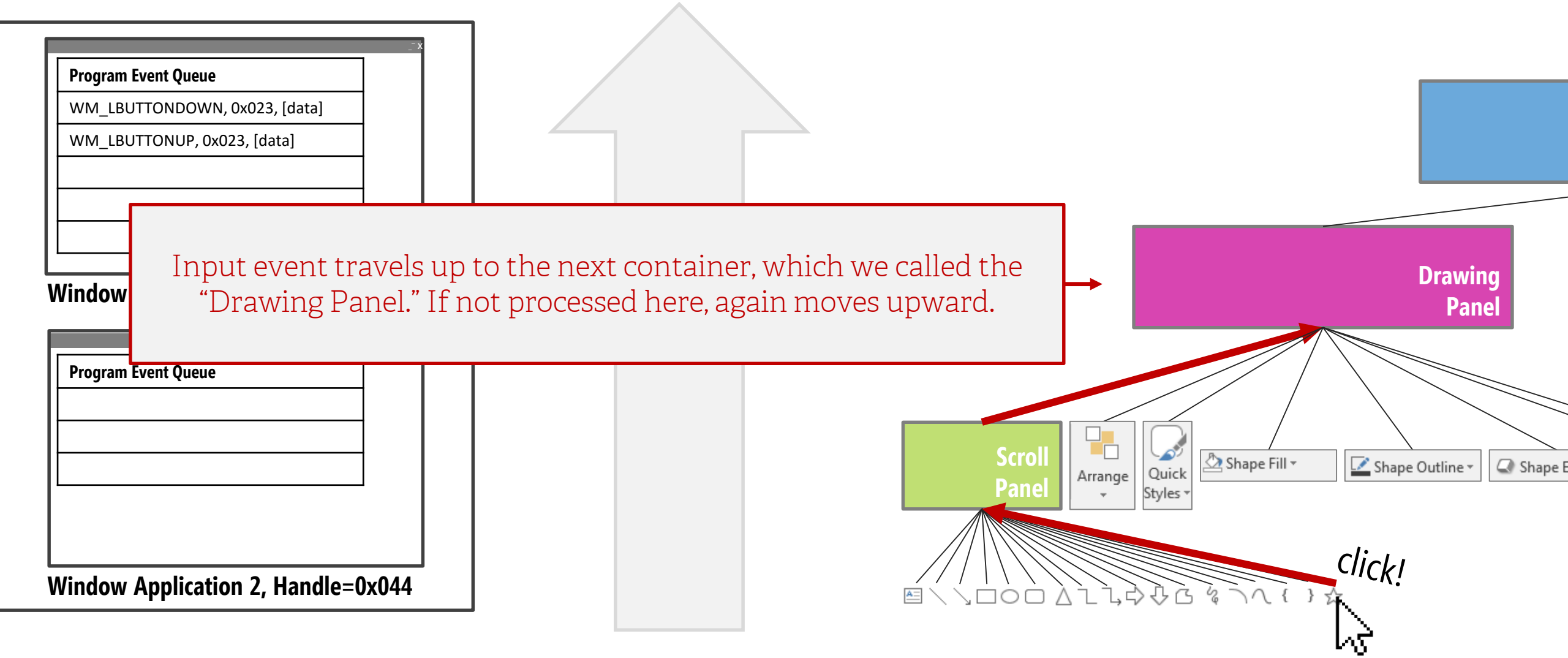
Event Processing Diagram



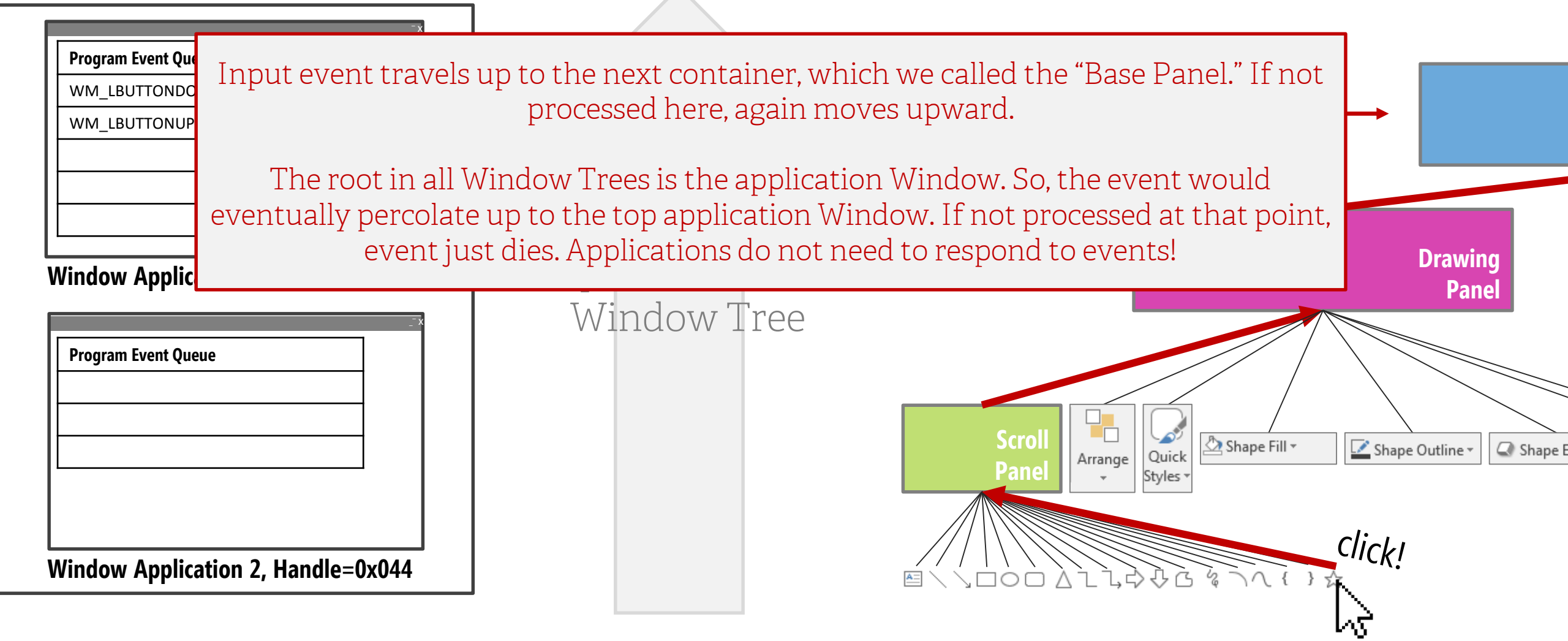
Event Processing Diagram



Event Processing Diagram



Event Processing Diagram



Input event travels up to the next container, which we called the "Base Panel." If not processed here, again moves upward.

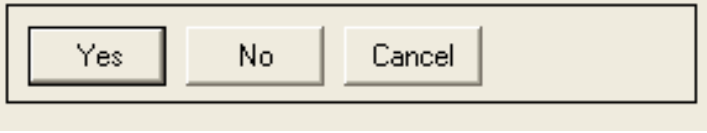
The root in all Window Trees is the application Window. So, the event would eventually percolate up to the top application Window. If not processed at that point, event just dies. Applications do not need to respond to events!

WPF Example

XAML Code

```
<Border Height="50" Width="300"
        BorderBrush="Gray" BorderThickness="1">
  <StackPanel Background="LightGray" Orientation="Horizontal"
              Button.Click="CommonClickHandler">
    <Button Name="YesButton" Width="Auto" >Yes</Button>
    <Button Name="NoButton" Width="Auto" >No</Button>
    <Button Name="CancelButton" Width="Auto" >Cancel</Button>
  </StackPanel>
</Border>
```

Windows UI



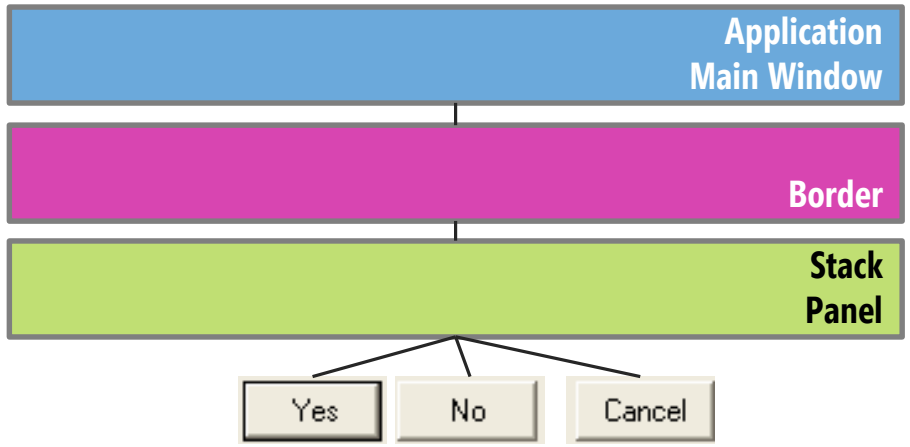
Source: Routed Events, https://msdn.microsoft.com/en-us/library/ms742806(v=vs.100).aspx

WPF Example

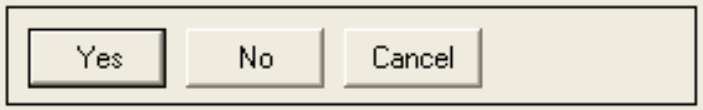
XAML Code

```
<Border Height="50" Width="300"
        BorderBrush="Gray" BorderThickness="1">
  <StackPanel Background="LightGray" Orientation="Horizontal"
              Button.Click="CommonClickHandler">
    <Button Name="YesButton" Width="Auto" >Yes</Button>
    <Button Name="NoButton" Width="Auto" >No</Button>
    <Button Name="CancelButton" Width="Auto" >Cancel</Button>
  </StackPanel>
</Border>
```

Window Tree (WPF calls this an Element Tree)



Windows UI



From the docs: In this simple element tree, the source of a Click event is one of the Button elements. When a Button is clicked, it is the first element with the opportunity to handle the event. If no handler attached to the Button acts on the event, then the event will bubble upwards to the Button parent in the element tree, which is the StackPanel. Again, if the StackPanel does not handle the event, it bubbles up to Border, and then beyond to the root of the element tree (the main Window).

Source: Routed Events, https://msdn.microsoft.com/en-us/library/ms742806(v=vs.100).aspx

Event Processing: Making Sense of Invalidate()

Why don't you call onDraw() directly to paint a window?

Instead, we call Invalidate(), which places a WM_PAINT message into the application's message queue.

It's because of the event-driven nature of UI. We treat a paint event like any other user-oriented event, so we can keep processing user input dynamically.

Mouse Events

When the user moves the mouse, the OS moves a bitmap on the screen called the **mouse cursor**

The mouse cursor contains a **single-pixel point** called the **hot spot**, a point that the OS tracks and recognizes as the **cursor position**

When a mouse event occurs, the **window that contains the hot spot** (typically) receives the **mouse message** resulting from the event

Source: About Mouse Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms645601(v=vs.85).aspx

Example Mouse Messages (Windows)

Move Messages	Description
WM_MOUSEMOVE	The user moves the cursor within the client area
WM_MOUSEHOVER	The cursor hovers over the client area for a certain time
WM_MOUSELEAVE	The cursor leaves the client area

Button Messages	Description
WM_LBUTTONDBLCLK	The left mouse button was double-clicked.
WM_LBUTTONDOWN	The left mouse button was pressed.
WM_LBUTTONUP	The left mouse button was released.
WM_MBUTTONDBLCLK	The middle mouse button was double-clicked.
WM_MBUTTONDOWN	The middle mouse button was pressed.
WM_MBUTTONUP	The middle mouse button was released.
WM_RBUTTONDBLCLK	The right mouse button was double-clicked.
WM_RBUTTONDOWN	The right mouse button was pressed.
WM_RBUTTONUP	The right mouse button was released.
WM_XBUTTONDBLCLK	An X mouse button was double-clicked.
WM_XBUTTONDOWN	An X mouse button was pressed.
WM_XBUTTONUP	An X mouse button was released.

Source: About Mouse Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms645601(v=vs.85).aspx

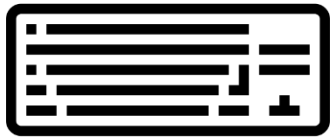
Mouse Message Data (Windows)

[X,Y] coordinate of the cursor hot spot

Flags such as MK_SHIFT (The SHIFT key is down) and MK_CONTROL (The CTRL key is down)

Information about which button is down (legacy Windows supported up to 5-button mice; unsure about Win10)

Source: About Mouse Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms645601(v=vs.85).aspx



Keyboard Input (Windows)

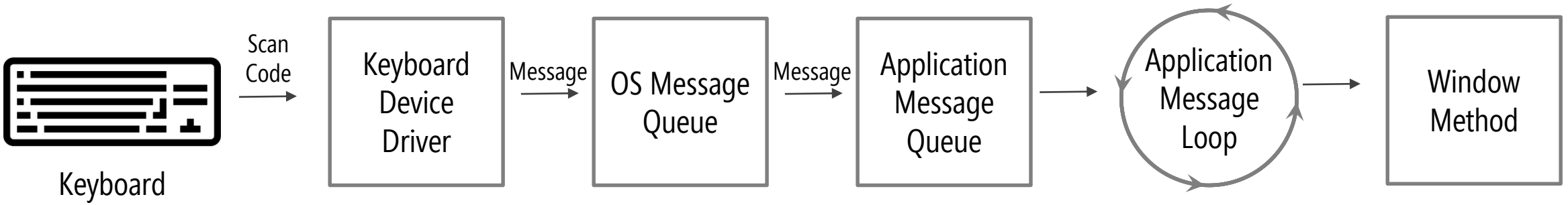
Windows provides device-independent keyboard support for applications by installing a keyboard device driver for the current keyboard

The keyboard device driver receives scan codes from the keyboard, which are sent to the keyboard layout processor (for language dependence), which are then translated into event messages, and posted to the in-focus window in the application.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

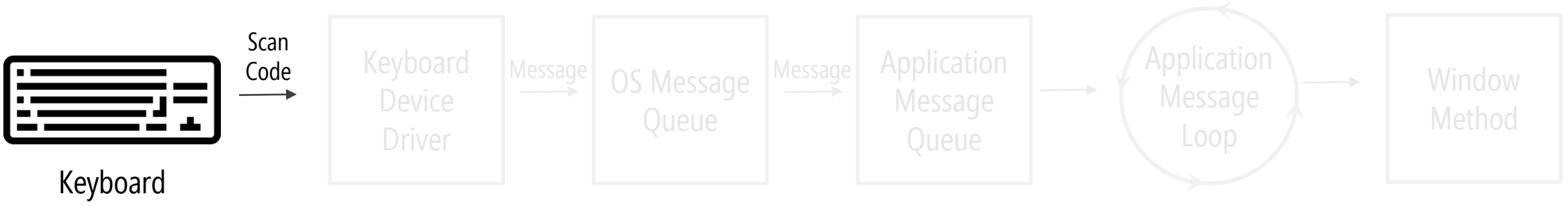
Keyboard Input (MS Windows)

This is exactly like the Event Processing Diagram from before, just a different illustration



Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

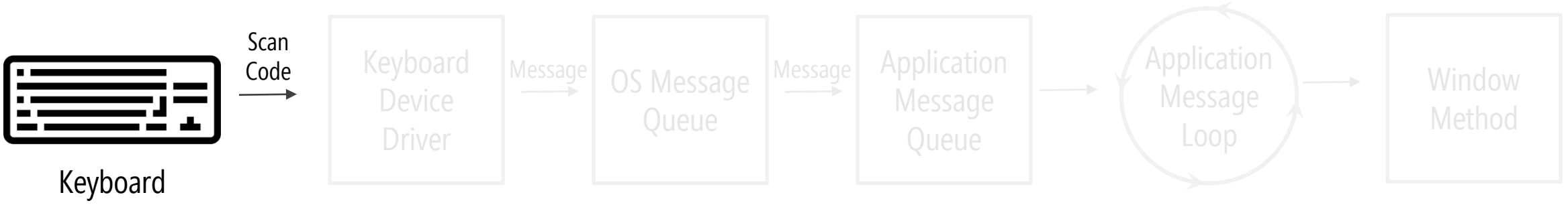
Keyboard Input (MS Windows)



Assigned to each key on a keyboard is a unique value called a **scan code**, a device-dependent identifier for each key on the keyboard.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

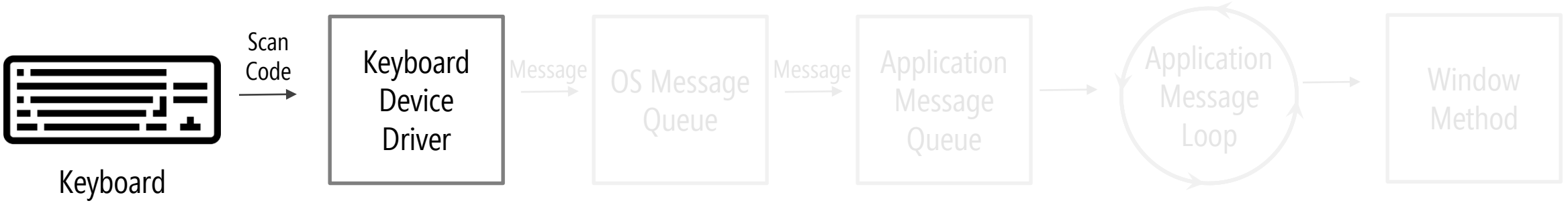
Keyboard Input (MS Windows)



Assigned to each key on a keyboard is a unique value called a **scan code**, a device-dependent identifier for each key on the keyboard. A **keyboard generates two scan codes** when the user types a key—one when the user presses the key, another when the user releases the key.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

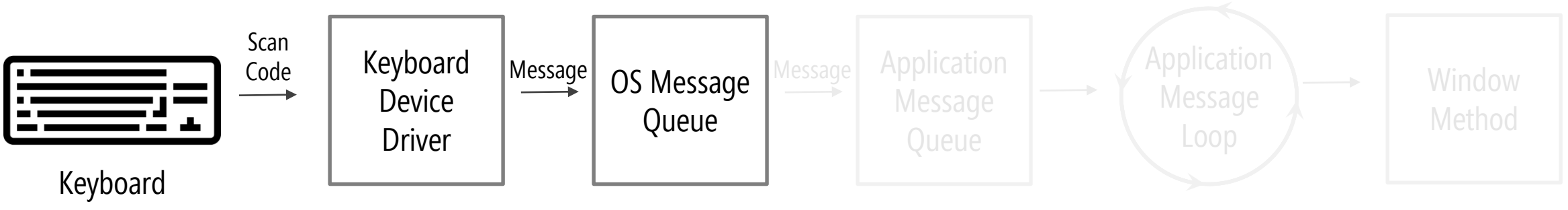
Keyboard Input (MS Windows)



The keyboard device driver analyzes the scan code and **translates it to a virtual keycode**, which is a device-independent value defined by the OS that identifies the key.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

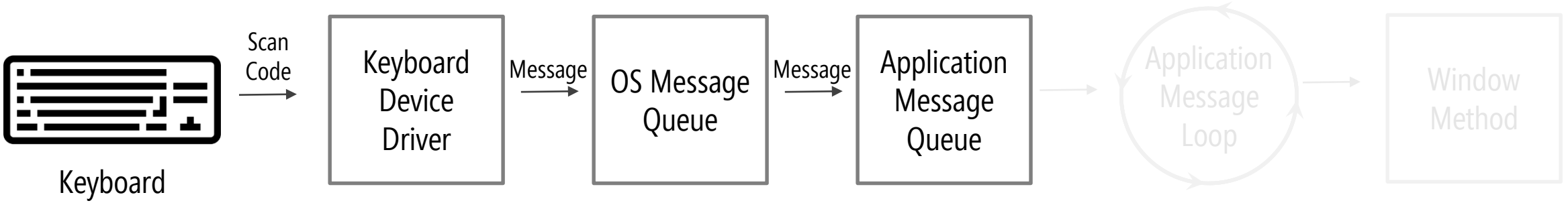
Keyboard Input (MS Windows)



The keyboard device driver analyzes the scan code and **translates it to a virtual keycode**, which is a device-independent value defined by the OS that identifies the key. The **driver then creates an event message** that includes the scan code, the virtual keycode, and other data and places the message into the OS message queue.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

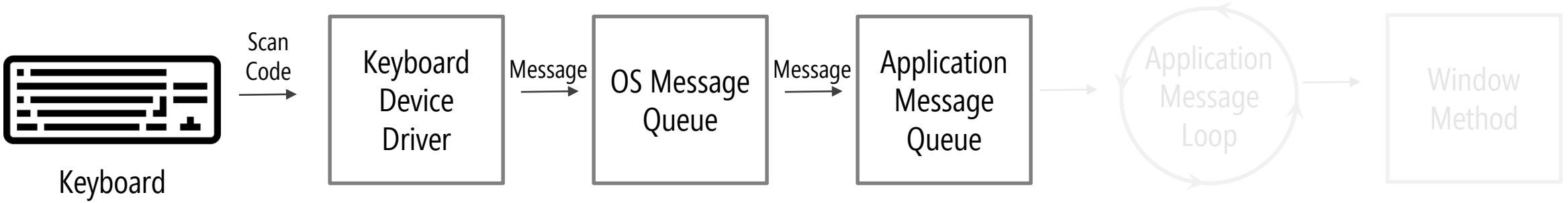
Keyboard Input (MS Windows)



The OS removes the message from the system message queue and posts it to the message queue of the appropriate application.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

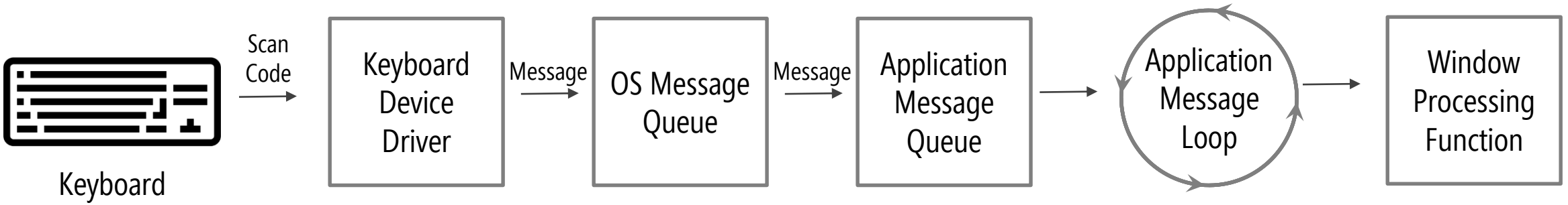
Keyboard Input (MS Windows)



The OS removes the message from the system message queue and posts it to the message queue of the appropriate application. The OS determines which application should receive keyboard events based on 'keyboard focus.' Only one window in the OS can have the keyboard focus at a time.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

Keyboard Input (MS Windows)



Eventually, the thread's message loop removes the message and passes it to the appropriate window for processing.

Source: About Keyboard Input, https://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx

In-Class Work on TA04 Mid-Fi

I want to come around and check-up on teams

Can also ask questions about Android II assignment

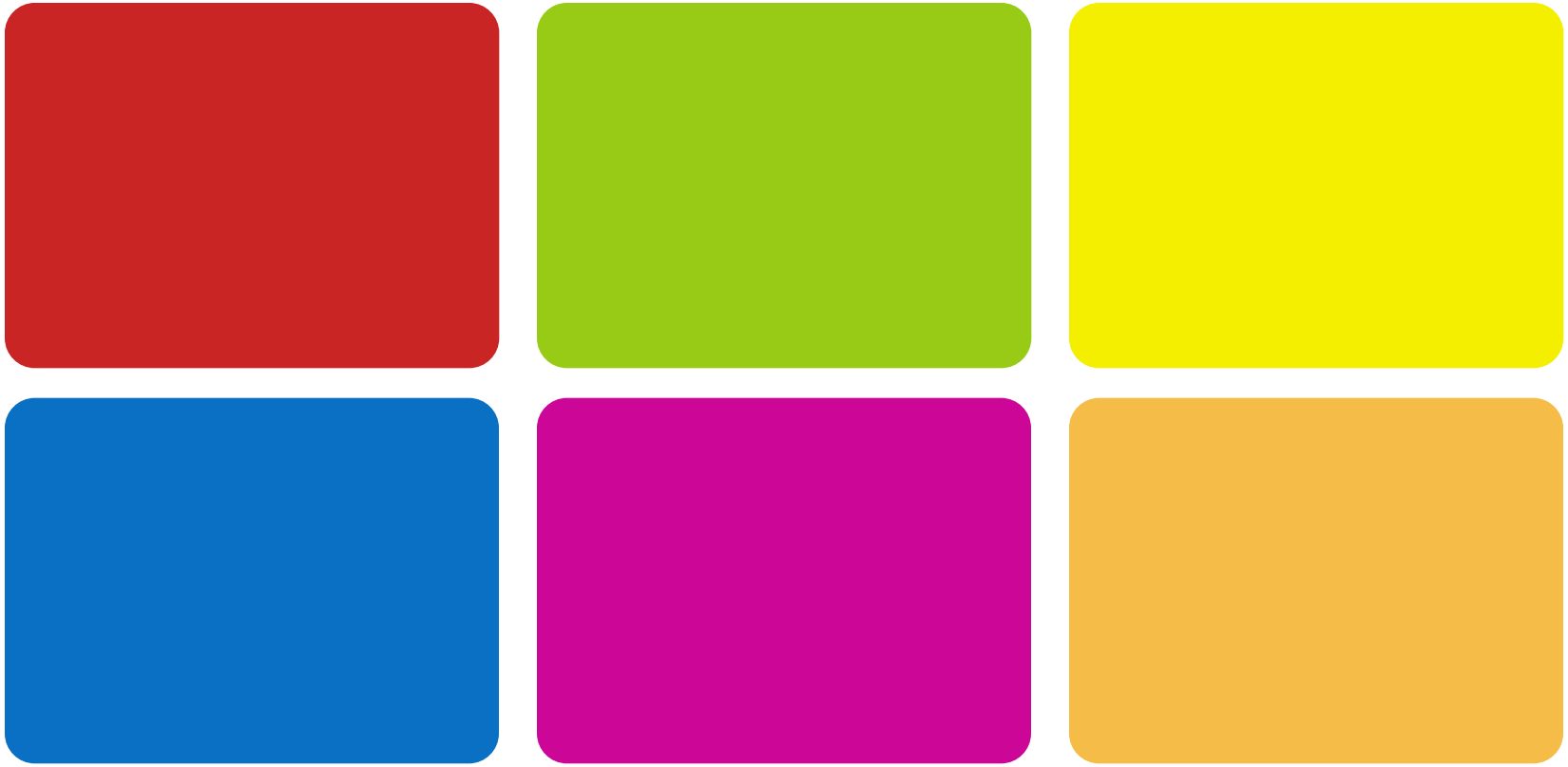
UI Containers and Components

UI Containers

Contain one or more UI components

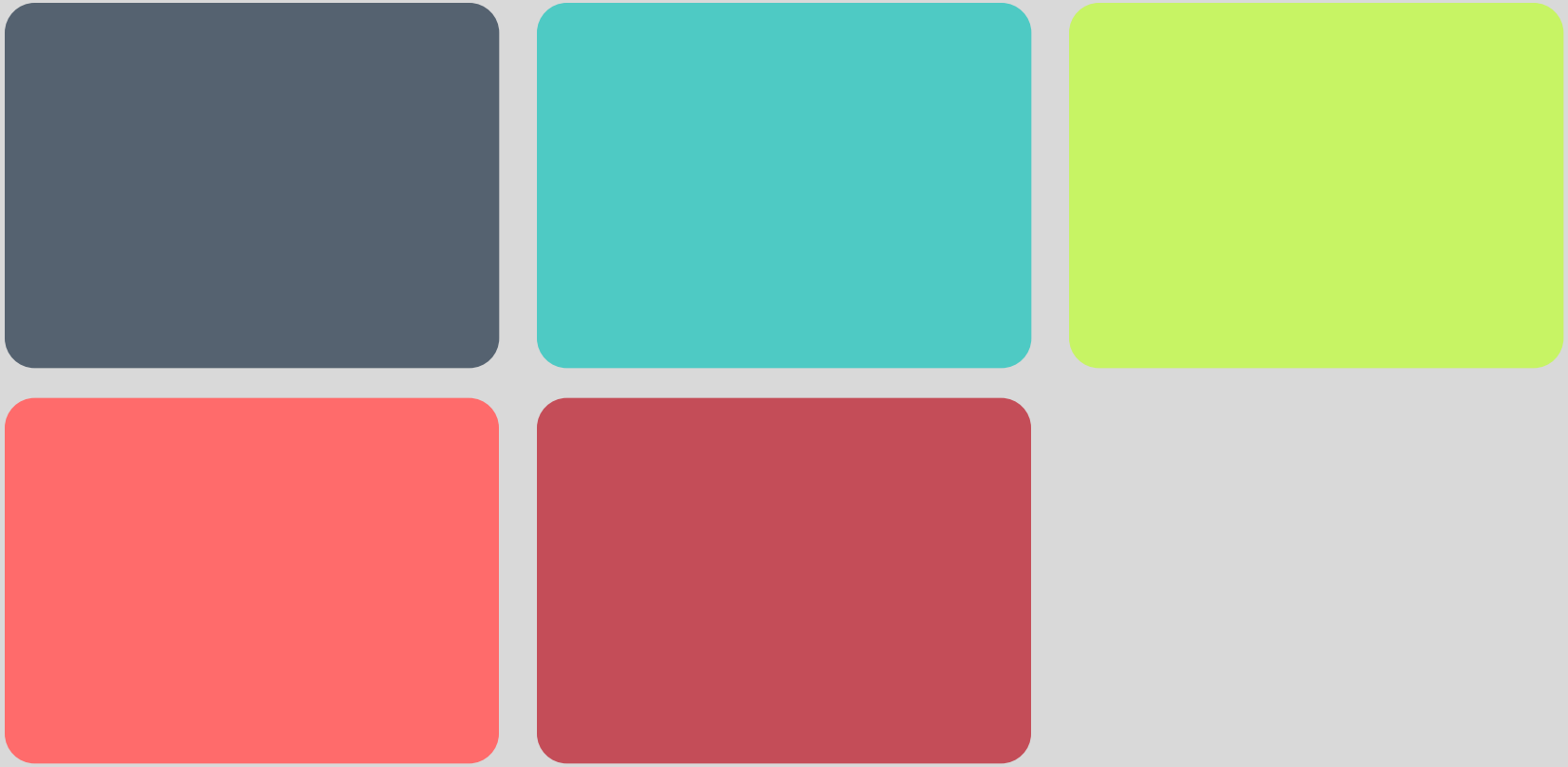
UI Components

Each UI component is a class with a paint method, list of event listeners, and

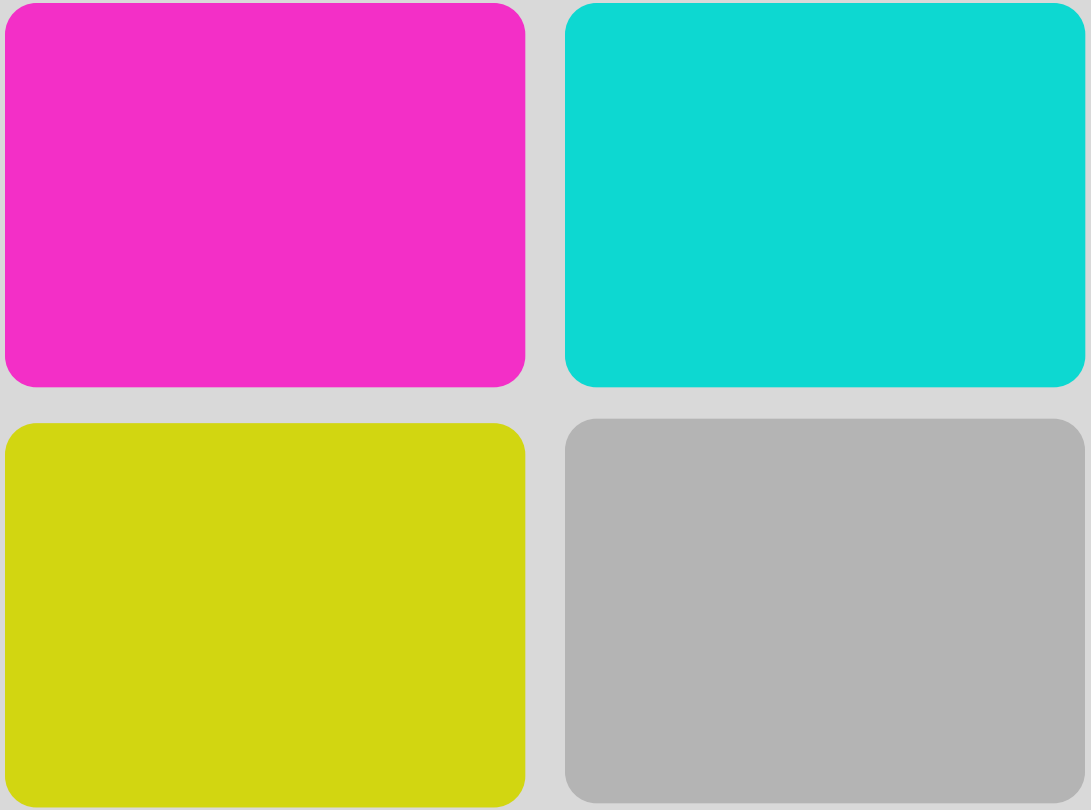


Dark Palette

Light Palette



Light Palette



Light Palette