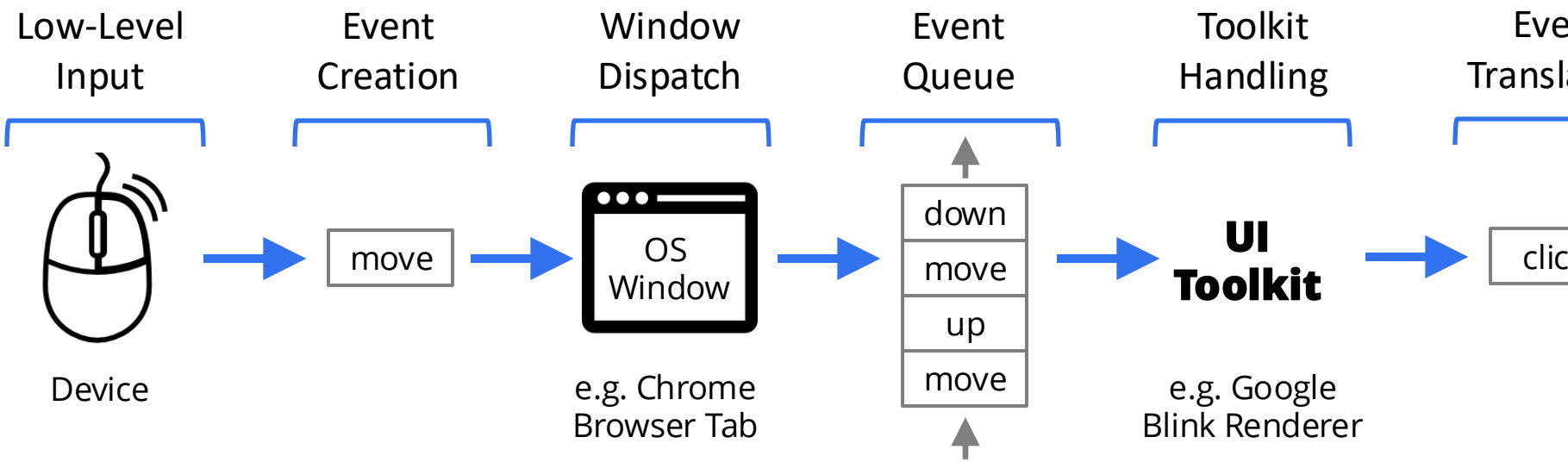
Event Dispatch

- Dispatch
- Binding
- Propagation

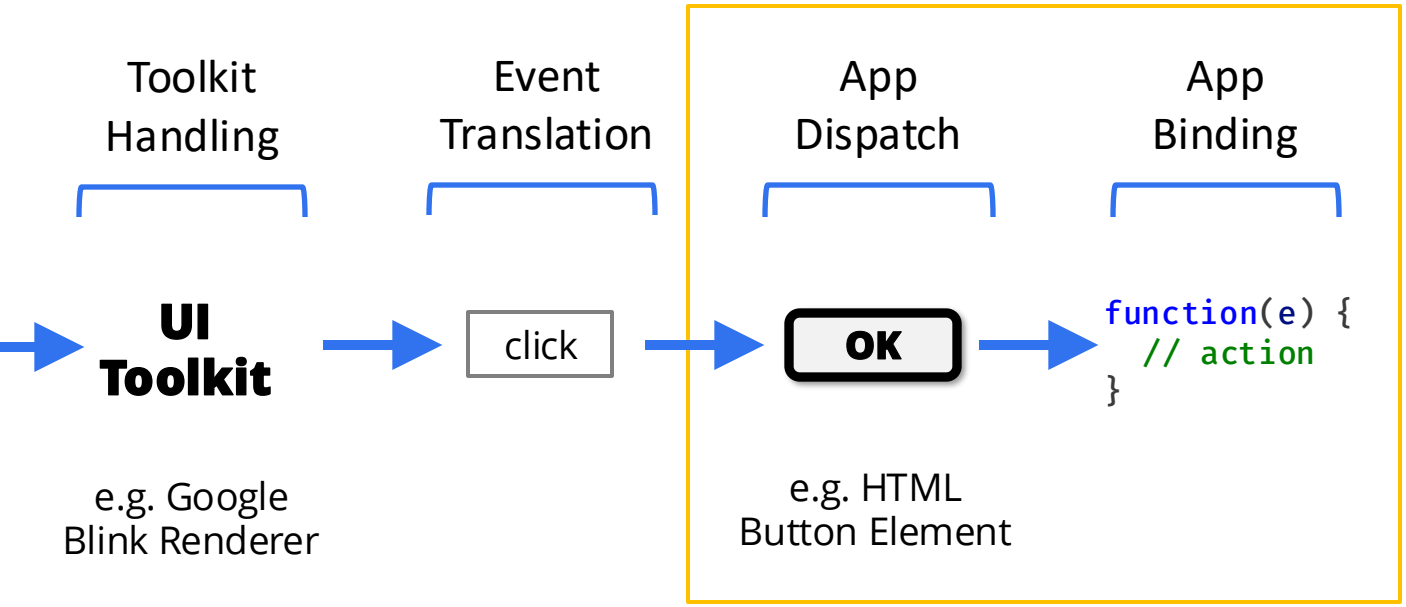
Event Pipeline So Far

review from Input Events lecture



App Dispatch and App Binding

- UI Toolkit events need to **trigger specific code** in the app



App Dispatch and Event Binding … so far

- A simple central **dispatch** method:
  - all events are handled in a single app function, e.g.

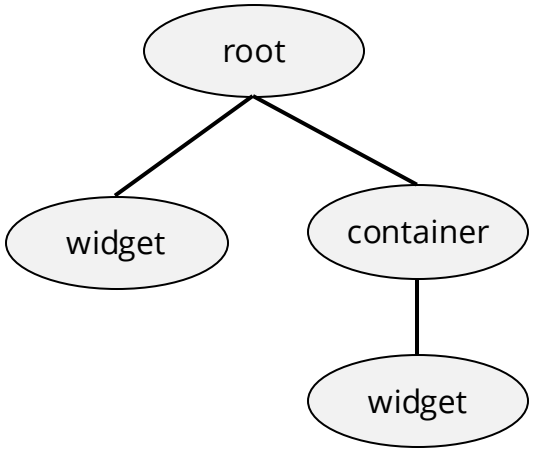
```
function handleEvent(e: SKEvent) { ... }
```

- A simple switch statement **binding** method:
  - app code to handle each event is in event handing function, e.g.

```
switch (e.type) {
   case "mousemove":
   // app code here for mousemove
   break;
   case "click":
   // app code here for click
   break;
   ...
```

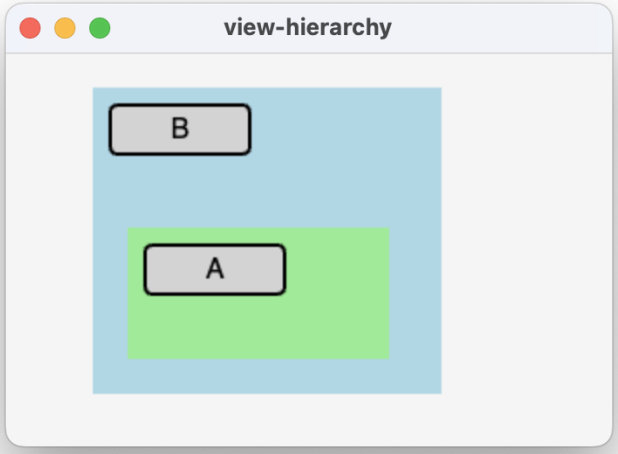
View Hierarchy

- UI toolkits typically organize widgets into a *tree*
  - only one root element
  - need **container widgets** for non-leaf nodes
  - child order dictates draw order (e.g. draw left-to-right)



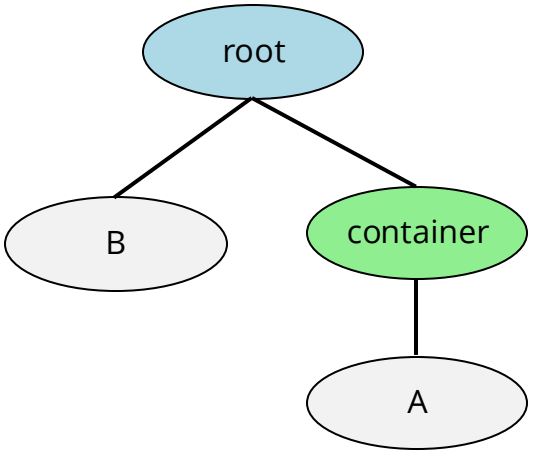
view-hierarchy

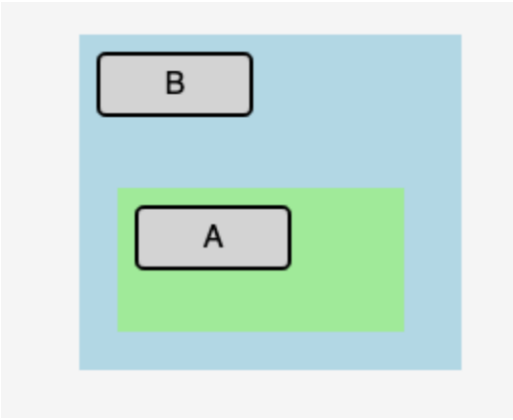
- Using SimpleKit imperative-mode
  - Using SimpleKit widgets
  - Console warning since demo uses setSKDrawCallback
  - (will show correct way near end of this lecture)
- Show how order of adding children changes display
  - How should events be “sent” to “overlapping” widgets?



View Hierarchy

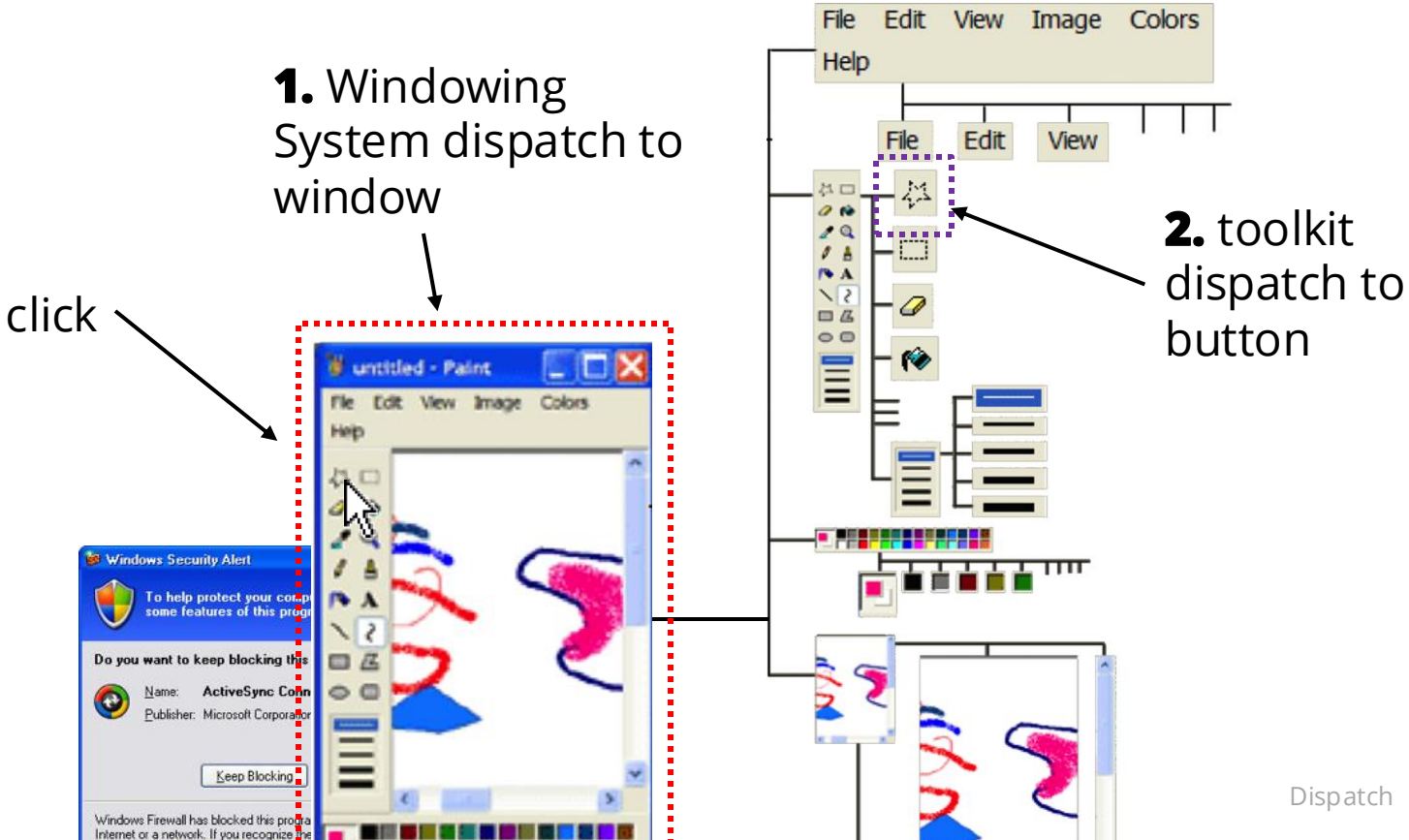
```
const blueContainer = new SKContainer( ... );
blueContainer.fill = "lightblue";

const buttonB = new SKButton( ... );

const greenContainer = new SKContainer( ... );
greenContainer.fill = "lightgreen";

const buttonA = new SKButton( ... );

// build the UI tree
blueContainer.addChild(buttonB);
blueContainer.addChild(greenContainer);
greenContainer.addChild(buttonA);
```





(Event) Dispatch

- In general English:
  - To send off or away with promptness or speed
- In user interface architecture:
  - To route an event to the appropriate widget or code



Event Dispatch Steps

1. Target selection
   - The frontmost widget under the mouse
2. Route construction
   - Path from root to target
3. Propagation
   - capture DOWN from root to target
   - bubble UP from target to root

Route construction and propagation only apply to ***positional dispatch***

Target Selection

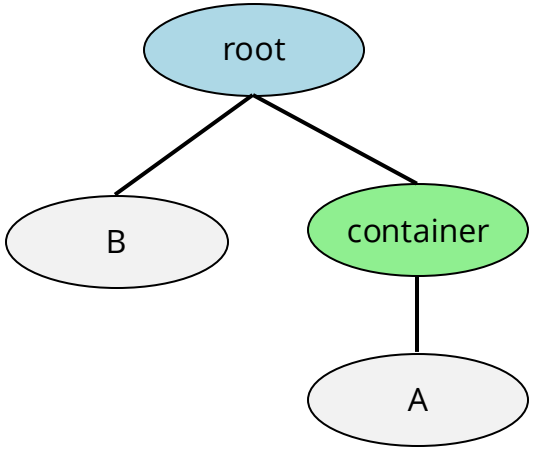
Determined by the type of event:

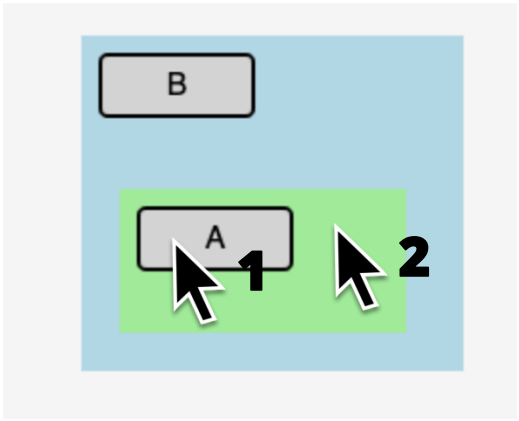
- *mouse event*: target is the widget at the location of the cursor (typically, mousedown target used until mouseup)
  - called **Positional Dispatch**
- *key event*: target is the widget that has focus
  - focus typically assigned with mouse click (could also be code or key like TAB)
  - called **Focus Dispatch**
- *touch events*: target selection may be more complex, e.g.:
  - A *continuous gesture* (like pinch-to-zoom) might select the target at the center point of all touches at gesture start
  - A *swipe* (like swipe right) might select the target at the center of the entire path of all fingers

Target Selection in Positional Dispatch

Target is last widget drawn under mouse

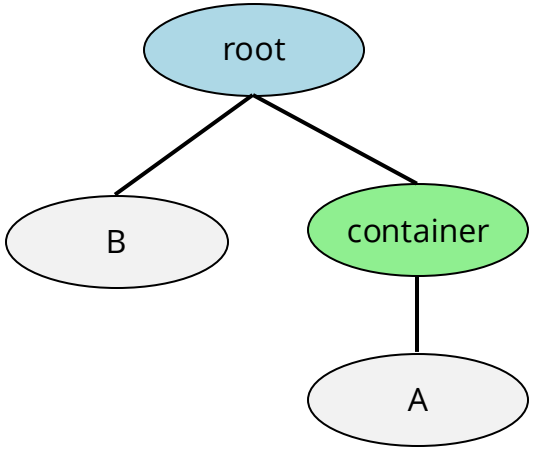
- mouse event at position 1:
  - buttonA
- mouse event at position 2:
  - greenContainer

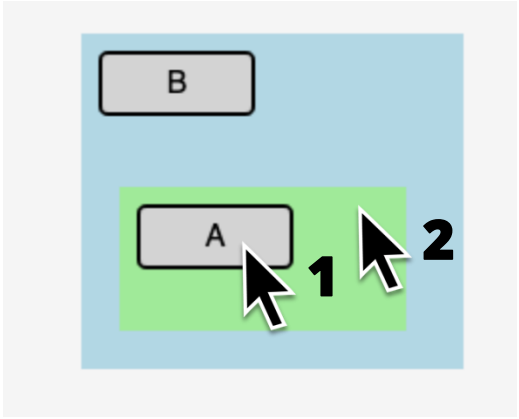




Route Construction in Positional Dispatch

- Route is from *root* to *target*
- mouse event at position 1:
  - blueContainer , greenContainer, buttonA,
- mouse event at position 2:
  - blueContainer , greenContainer





target-route

NOTE: simplified TypeScript

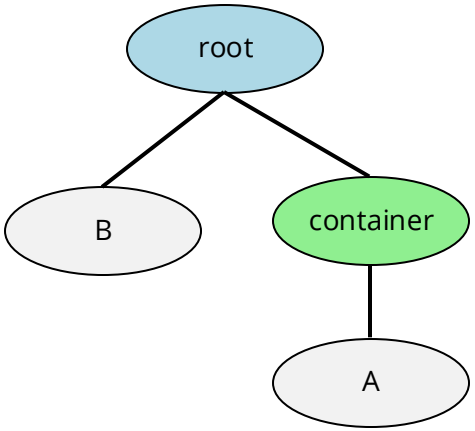
```
// returns list of elements under mouse (from back to front)
function buildTargetRoute(mx, my, SKElement) {
  route = [];
  if (element instanceof SKContainer) {
    element.children.forEach((child) =>
      route.push(
        ...buildTargetRoute(
          mx - element.x,
          my - element.y,
          child
        )
      )
    );
  }

  if (element.hittest(mx, my)) {
    return [element, ...route];
  } else {
    return route;
  }
}
```

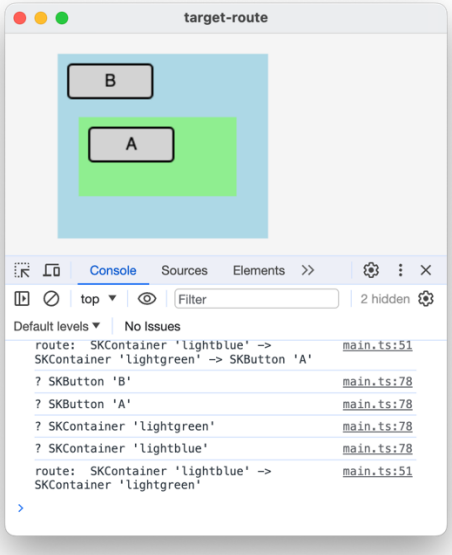
visit children from back to front

Translate to child coordinate system

add this element to route if hit

o.w. just return route

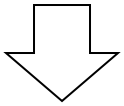




Event Dispatch Steps

1. Target selection
   - The frontmost widget under the mouse
2. Route construction
   - Path from root to target node
3. Propagation
   - capture DOWN from root to target
   - bubble UP from target to root

Route construction and propagation only apply to positional dispatch



Event Binding

hard to demo propagation without this last step

Event Binding

- How to associate events with code?
  - (route to code, send to code, handle with code, ...)
- Design goals of event binding mechanisms:
  - Easy to understand (clear connection between event and code)
  - Easy to implement (binding paradigm or API)
  - Easy to debug (how did this event get here?)
  - Good performance

Global Event Callback

- Used in early Windows
  - each app window registers a WindowProc function (Window Procedure) which is called each time an event is dispatched
  - a switch statement binds event to code
  - (there were over 100 standard events ...)

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg,
                            WPARAM wParam, LPARAM lParam) {
   switch (uMsg) {
      case WM_CLOSE:
         DestroyWindow (hWnd);
         break;
      case WM_SIZE:
         …
      case WM_KEYDOWN:
         …
```

Windows C++ code

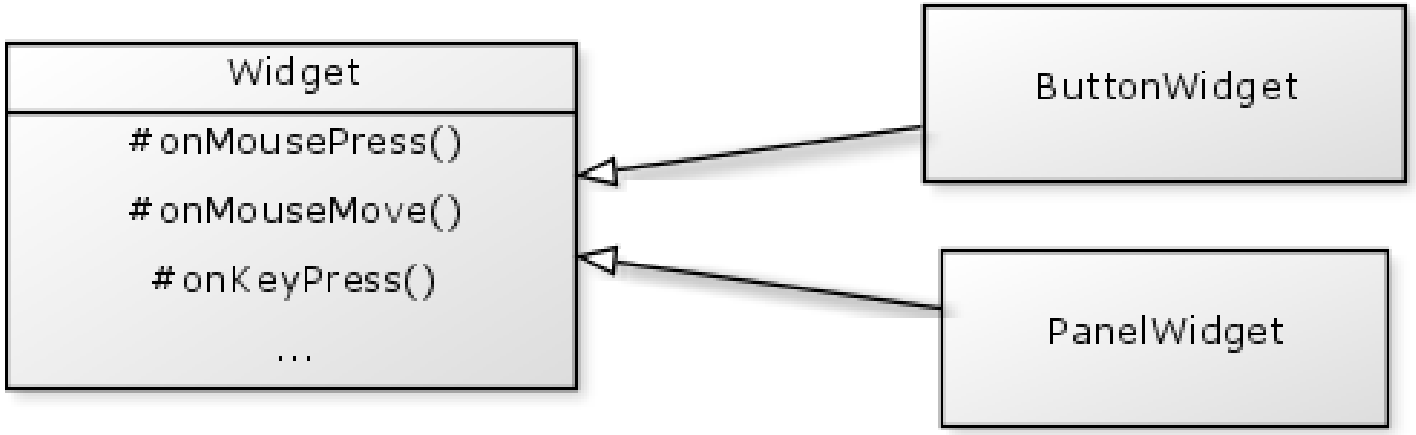
this is basically what SimpleKit does with setSKEventHandler

Global Event Callback Binding Problems

- Difficult to maintain
  - Dozens of different types of events that need to be managed
- Events are not delegated to an object
  - Leads to code where events are handled in callback itself
- Better if widgets handle the events themselves
  - e.g. "click" event on widget is bound *directly* to method on that widget object

Inheritance Binding

- Event is dispatched to a widget base object
  - widget extends from base class with all event handling methods
- Base class can choose specificity of event handling method
  - general event types, e.g. `onMouse, onKeyboard`
  - specific events, e.g. `onMouseMove, onMouseClick`
- Used in Java 1.0



Inheritance Binding Problems

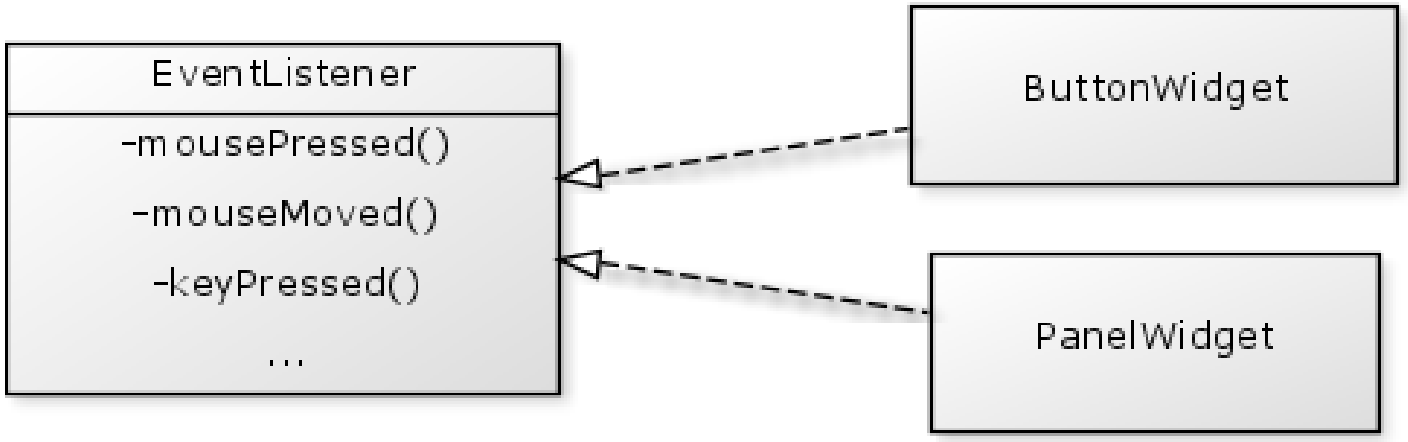
- Multiple event types are processed through each event method
  - still a switch statement, but in the widget

but is more specific switch cases

- No filtering of events might introduce performance issues
  - consider events like `mousemove`: all will be delivered
- If using specific event methods, it doesn't scale well
  - need to modify the base class to add new event types
    e.g. `penButtonPress`, `touchGesture`, ...

Listener Binding

- Define interfaces for *specific event types (or device types)*
  - e.g. MouseListener, MouseMotionListener, KeyListener, …
- Create object that implements interface to handle
  e.g. KeyListener for keyboard events
- When event is dispatched, relevant listener method is called
  e.g. mousePressed, mouseMoved, …
- Used in JavaFX



SimpleKit Binding

- Uses a form of **Inheritance Binding** with **Listener Objects**
- Each SKElement binds events to event handlers
  - toolkit events, like “mousemove” and “keydown”
  - widget events, like “action” when a SKButton is clicked

SimpleKit Binding

- Defines an *event handler function* type

```
type EventHandler = (me: SKEvent) => void;
```

- Defines a *binding route object* to map event type to handler

```
type BindingRoute = {
  type: string; // event type
  handler: EventHandler;
};
```

- SkElement maintains a *table of binding routes*

```
bindingTable: BindingRoute[] = [];
```

- SkElement method to add *event listeners* to a binding table

```
addEventListener(
  type: string,
  handler: EventHandler
) { this.bindingTable.push({ type, handler }); }
```

showing simplified version of code

SimpleKit Binding

- Toolkit calls methods in SKElement with event:

  `handleMouseEvent`

  `handleKeyboardEvent`

- SKElement handles all standard UI Toolkit events

```
handleMouseEvent(me: SKMouseEvent) {
  this.sendEvent(me);
}
```

- SKElement has method to send event to handler (if one exists in binding table)

```
protected sendEvent(e: SKEvent) {
  this.bindingTable.forEach((d) => {
    if (d.type == e.type) { d.handler(e); }
  });
}
```

showing simplified version of code

SimpleKit Binding

- A widget can implement methods to handle toolkit events
- To update widget state:

```
handleMouseEvent(me: SKMouseEvent) {
  switch (me.type) {
    case "mousedown":
      this.state = "down";
    break;
  ...
```

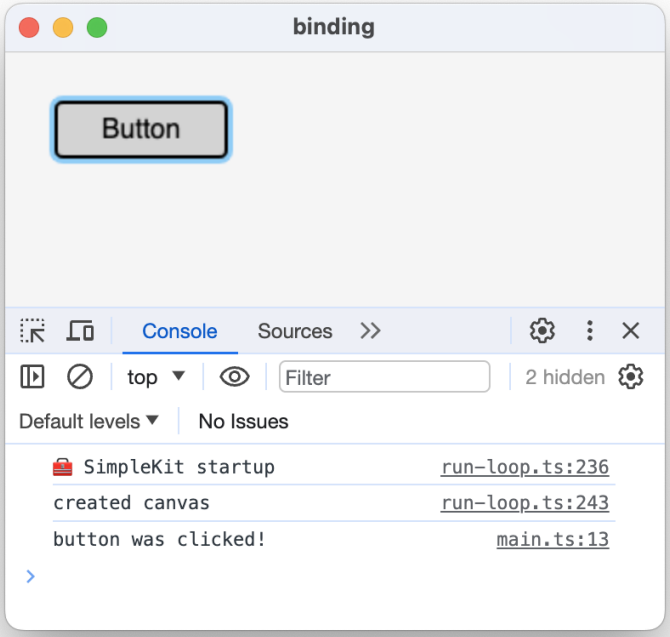
- To send special "widget events":

```
handleMouseEvent(me: SKMouseEvent) {
   switch (me.type) {
     ...
     case "mouseup":
       this.sendEvent({
         source: this,
         timeStamp: me.timeStamp,
         type: "action",
       } as SKEvent);
       break;
     ...
```

showing simplified version of code

binding

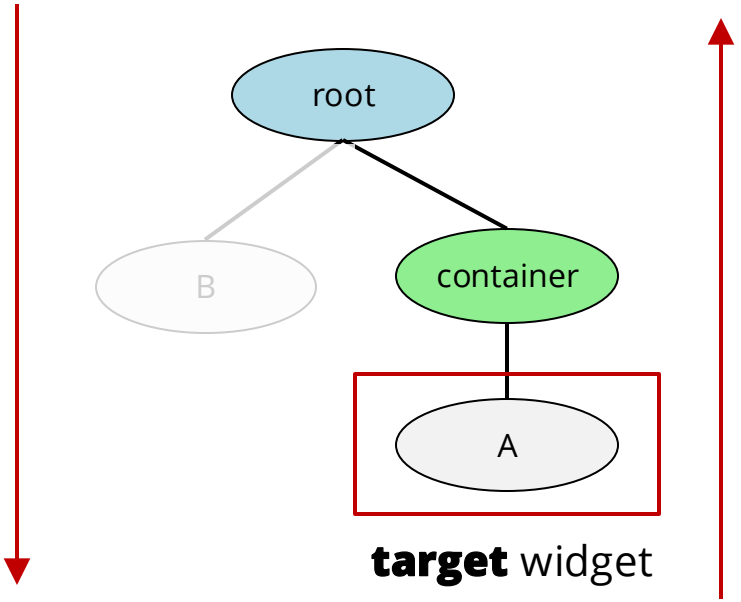
- Simple demo without toolkit dispatch
- In SimpleKit widget code, look at:
  - SKElement.addEventListener
  - button.handleMouseEvent
  - SKElement.dispatch



Event Propagation

- Most UI toolkits support top-down and bottom-up propagation
  - top-down is called **capture**
  - button-up is called **bubbling**
- Any widget in the path can use the event during either pass
- A handler can stop all following propagation
  (i.e. a capture handler can stop rest of capture and bubble phase)

Capture phase walks **down** the tree from the root through each widget until it reaches the target widget



Bubble phase walks **up** the tree starting from the target widget, through each ancestor widget until it reaches the root

https://javascript.info/bubbling-and-capturing
https://docs.oracle.com/javafx/2/events/processing.htm

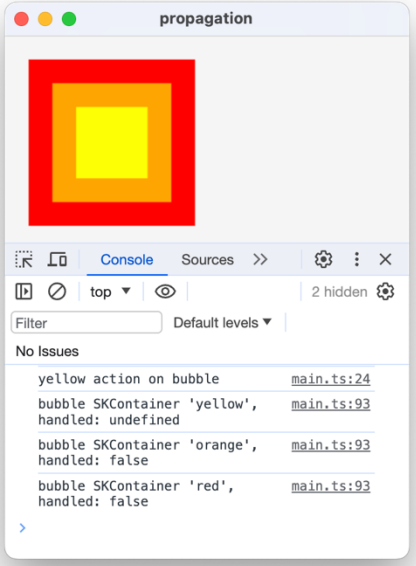
propagation

```
function dispatch(me: SKMouseEvent, root: SKElement) {
  const route = buildTargetRoute(me.x, me.y, root);

  // capture
  const stopPropagation = route.some((element) => {
    return element.handleMouseEventCapture(me);
  });

  if (stopPropagation) return;

  // bubble
  route.reverse().some((element) => {
    return element.handleMouseEvent(me);
  });
}
```

capture binding

returns true to stop propagation

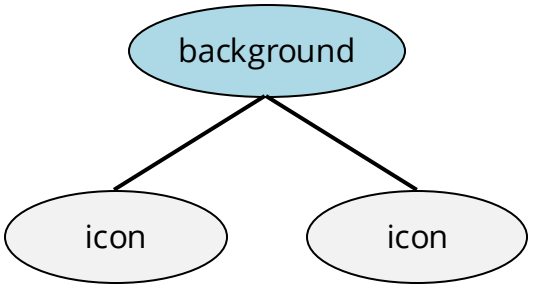
bubble binding



- Demo
  - capture flag for listeners
  - return true in handler to stop propagation

Why stop propagation?

- Prevent some events from bubbling up to “default events”
  - e.g. click on icon selects it, click on background deselects all icons




```
background.addEventListener("click", (e) => {
  // de-select all icons ...
});

icons.foreach((icon) => {
  icon.addEventListener("click", (e) => {
    // select this icon ...
  });
});
```

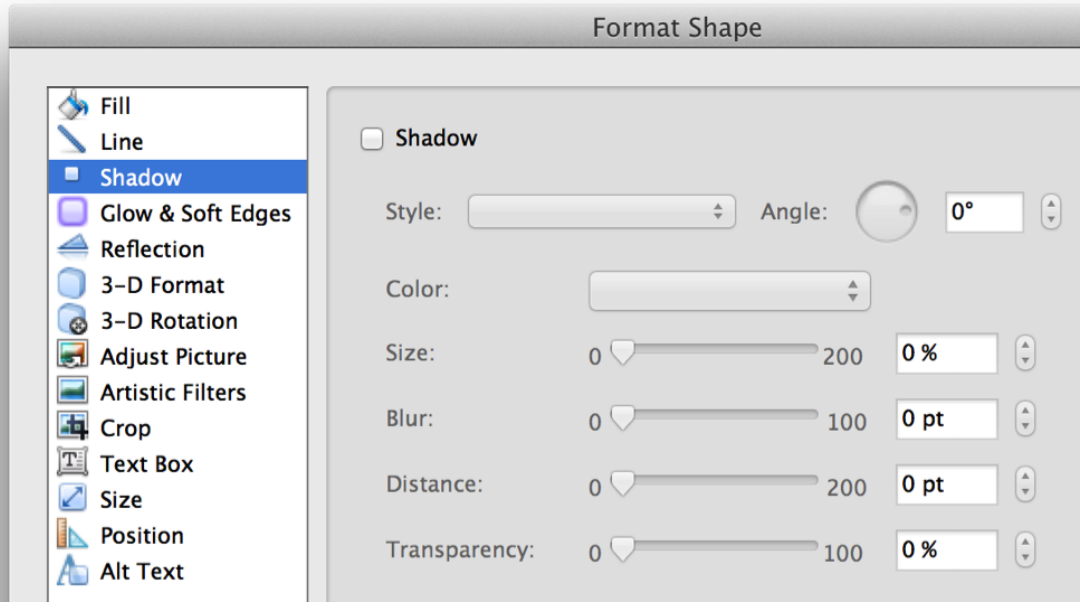
What will happen with this code?

How to fix it?

Why Capture Phase?

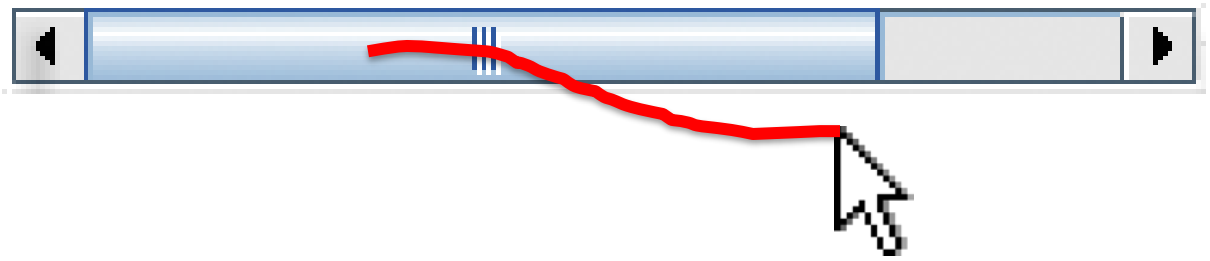
- Events higher up in the widget tree can "filter" events
  - use "handled" flag to stop propagation

Example: a parent wishes to disable all its children.



Positional Dispatch Limitations

- Pure positional dispatch can lead to odd behaviour:
  - Mouse drag starts in a scrollbar, but then moves outside the scrollbar: send the events to the adjacent widget?
  - Mouse press event in one button widget but release is in another: each button gets one of the events?
- Must also consider which widget is “in focus”



Focus Dispatch

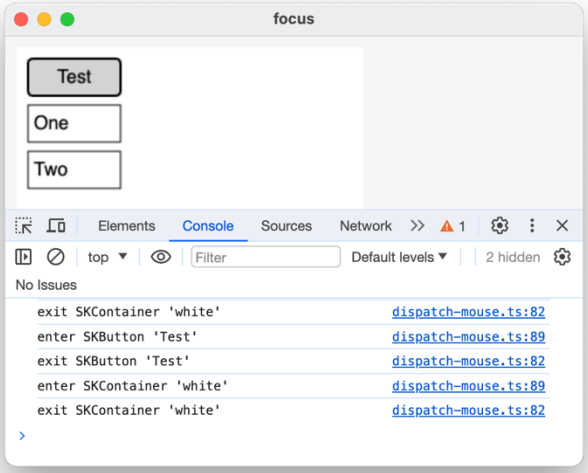
- Events dispatched to widget regardless of mouse cursor position
- Needed for all keyboard and some mouse events:
  - **Keyboard focus**: click on text field, move cursor off, start typing
  - **Mouse focus**: mousedown on button, move off, mouseup (also called “mouse capture”)
- Maximum one keyboard focus and one mouse focus
  - why?
- Need to gain and lose focus at appropriate times

Focus Dispatch *Needs* Positional Dispatch

- A mousedown event sets mouse and keyboard focus to a widget
  - Only text entry widgets should request keyboard focus
  - Any widget could request mouse focus
- UI Toolkits have a dedicated focus managers
  - As part of the dispatch method
- There are other ways to request focus
  - TAB key to navigate a UI without a mouse
    (assumes the UI toolkit defines a “tab order” for widgets)
  - An app can typically request focus itself
    (i.e. pressing ENTER moves keyboard focus to “next” textfield)

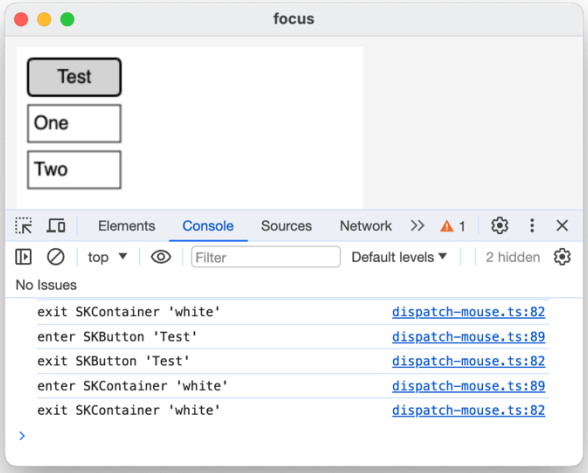
focus (in “dispatch-keyboard.ts”)

- Keyboard dispatch *needs* a `focusedElement`
  - See `keyboardDispatch`
- `requestKeyboardFocus` for elements to request keyboard focus
  - Used in SKTextfield in `handleMouseEvent`
  - `"focusout"` and `"focusout"` event creation and immediate dispatch to the widget’s `handleKeyboardEvent`
- DEMO:
  - debug = true



focus (in “dispatch-mouse.ts”)

- Mouse focus handling in `mouseDispatch` function
  - `focusedElement` module variable
- `requestMouseFocus` for elements to request mouse focus
  - Used by SKButton in handleMouseEvent
- DEMO:
  - with debug = true to see console log



Mouse Enter and Exit Events

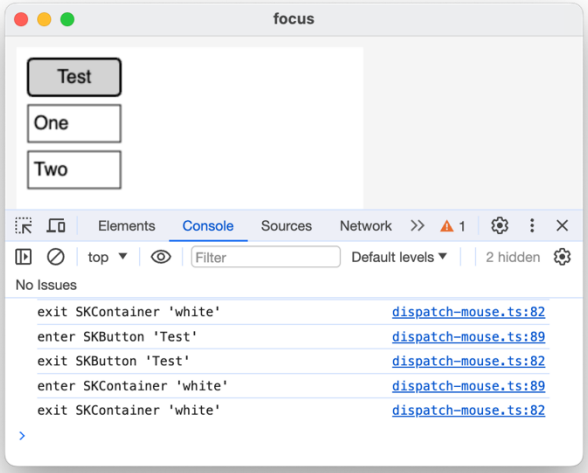
- UI Toolkits generate events when mouse *enters* and *exits* a widget
  - These events are used by widgets for “hover” effects
- Approach:

1. Get the element at very end of the target route (i.e. the front-most widget)
2. If that element wasn't the “last element entered”
   - Send “mouseexit” event to the last element entered
   - Send “mouseenter” event to the element at end of the route

- In practice, a widget can refuse an enter event, then the toolkit will check the penultimate element in the route (and so on)
  - (But SimpleKit always sends enter/exit event to end element)

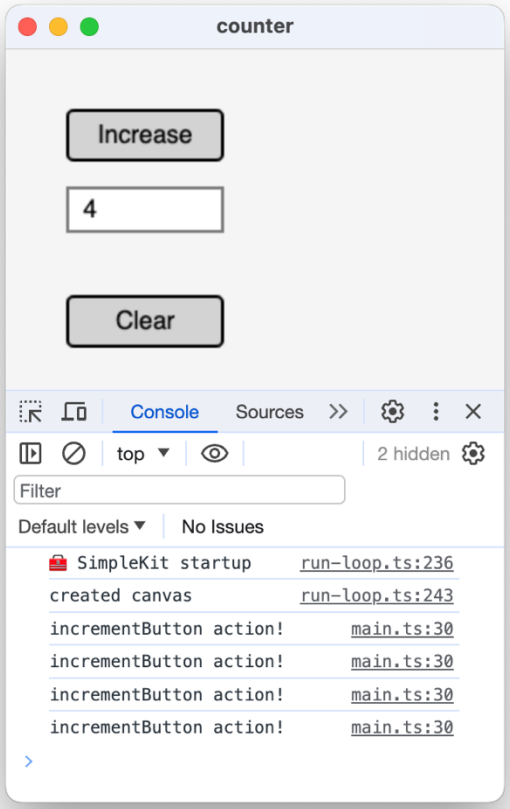
focus (in “dispatch-mouse.ts”)

- Mouse enter/exit handling in `updateEnterExit` function
  - `lastElementEntered` module variable
  - `"mouseenter"` and `"mouseexit"` event creation and immediate dispatch to the widget’s `handleMouseEvent`
- DEMO:
  - with debug = true to see console log



counter

- Building widget tree
- Using skSetRoot
  - no need for skEventHandler or skDrawCallback
- Using eventListeners
  - SKButton increment counter
  - SKButton clear counter
  - SKTextfield to display and edit counter value
- SKTextfield eventListener
  - uses e.source to get reference to widget
  - simple numeric validation
  - parseInt or 0



Exercise

EXERCISE

Make a SimpleKit app to add two numbers

Pressing the “+” SKButton adds the two numbers in two SKTextfields and sets the answer in a result SKLabel

Use x and y properties to “layout” the widgets to look like the picture inside a SKContainer

If you change a number after a result was displayed, the result returns to “= ?”.

Don’t worry about numeric validation, if the textfields aren’t numeric the result is “= NaN”