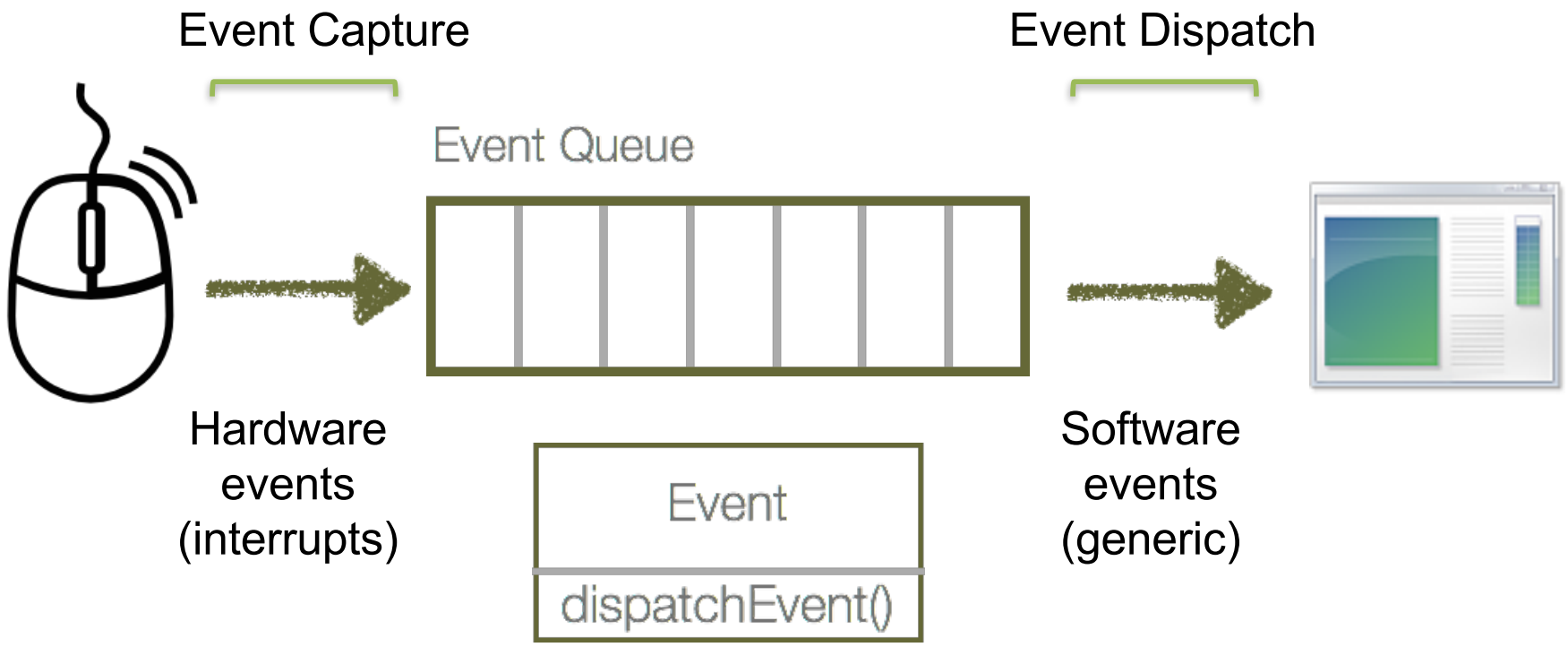
Event Dispatch

Dispatching events to windows and widgets.

Event capture, processing and dispatch.



The event dispatch stage includes:

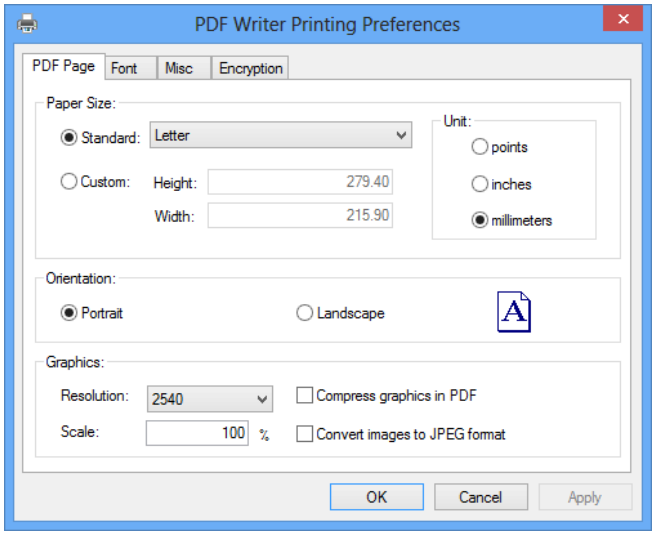
1. **Event dispatch**: Getting events to correct widget
2. **Event handling**: Running code for an event
3. **Notifying view and windowing system**: Model-View-Controller, and view notification.

Recall: Event Loop

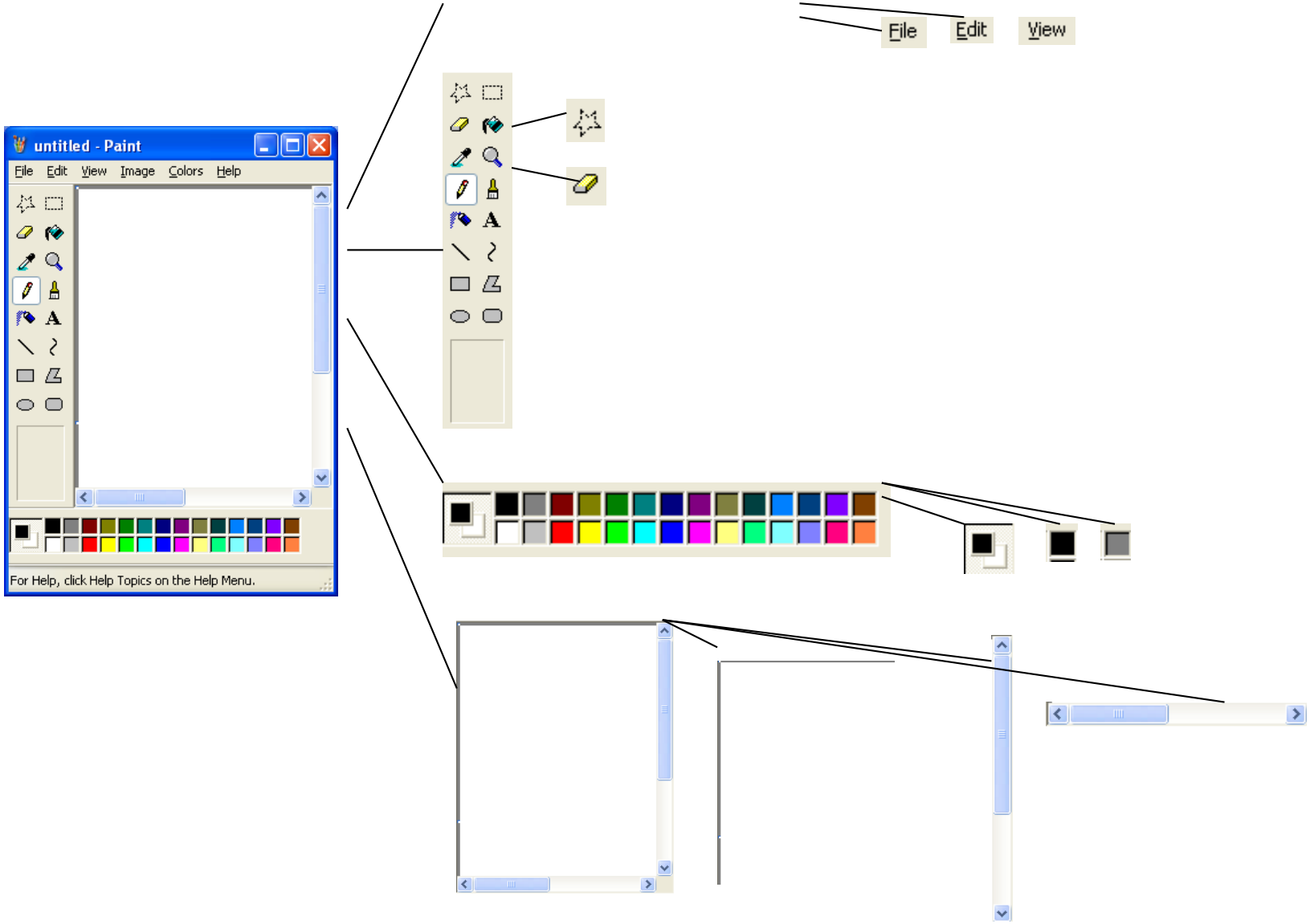
- The **event loop** is the primary mechanism for event dispatch within an application.
  - The event loop can be managed by application (XLib), or the toolkit (JVM).
  - The event loop iterate through all events in the event queue, and pushes them in order to the appropriate application.
  - The application needs to determine which component should process the event.
  - Widgets (i.e. components) are often the final target of events dispatched from the event loop
- Once delivered to a widget, it still needs to interpret what any of the input ***means*** in the proper context and react appropriately.

Widget Layout

- In complex applications, widgets are often laid out in a hierarchy
  - We call this hierarchy an **interactor tree**.
  - Container classes and low-level components are nested together.
  - Dispatching an event to the correct widget (that can handle the event) means *traversing this tree*.
- Question:
  - Which window?
  - Which widget receives it?



Managing Graphical Complexity



PaintDemo / PaintDemo.java

Lightweight vs. Heavyweight

Heavyweight widgets

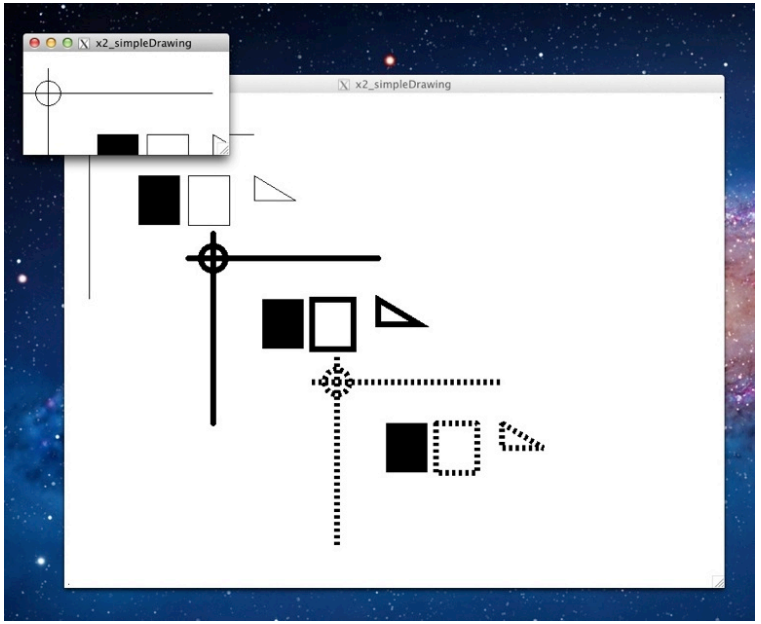
- ***Widget toolkit wraps native-OS widgets***
- BWS/OS provides a hierarchical "windowing" system for *all* widgets across all applications, and treats a widget essentially as a window
- This allows the BWS to dispatch events to a *specific widget (and not just the top-level window).*
- *Examples: nested X Windows, Java's AWT, standard HTML form widgets, Windows MFC*

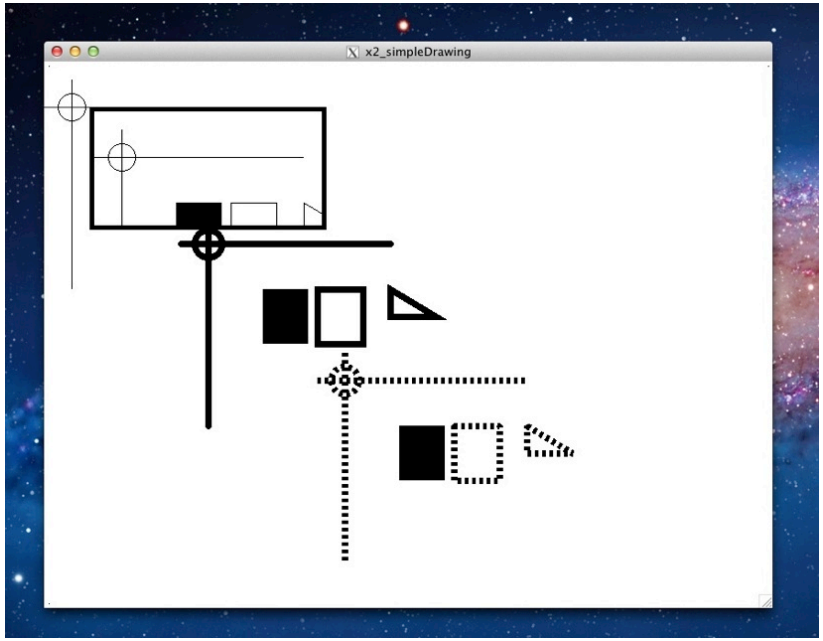
Lightweight widgets

- ***The widget toolkit draws its own widgets*** and is responsible for mapping incoming events to widgets
- BWS/OS provides a top-level window only, and can only dispatch to the window (NOT the widget)
- *Examples: Java Swing, JQuery UI, Windows WPF*

X11 Example: Heavyweight Widgets

```
xInfo1.display = display;
xInfo1.screen = screen;
initX(argc, argv, xInfo1, DefaultRootWindow(
display) 100, 100, 800, 600);

xInfo2.display = display;
xInfo2.screen = screen;
initX(argc, argv, xInfo2, DefaultRootWindow(
display) 50, 50, 300, 200);
```



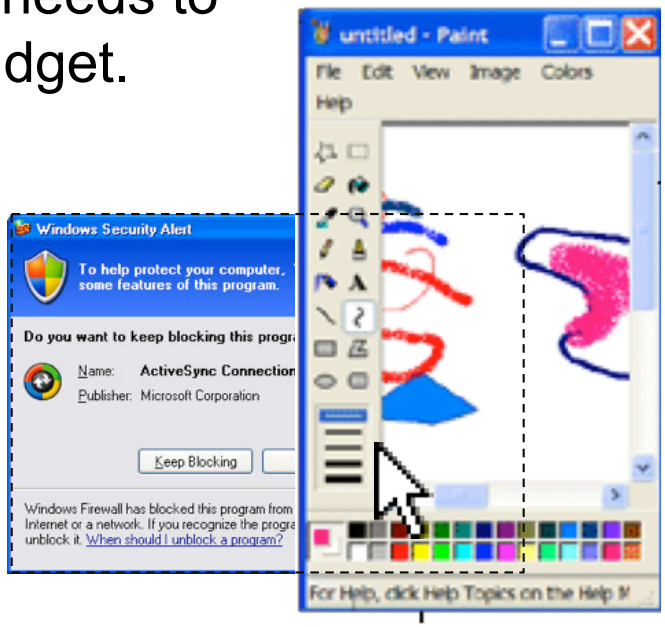


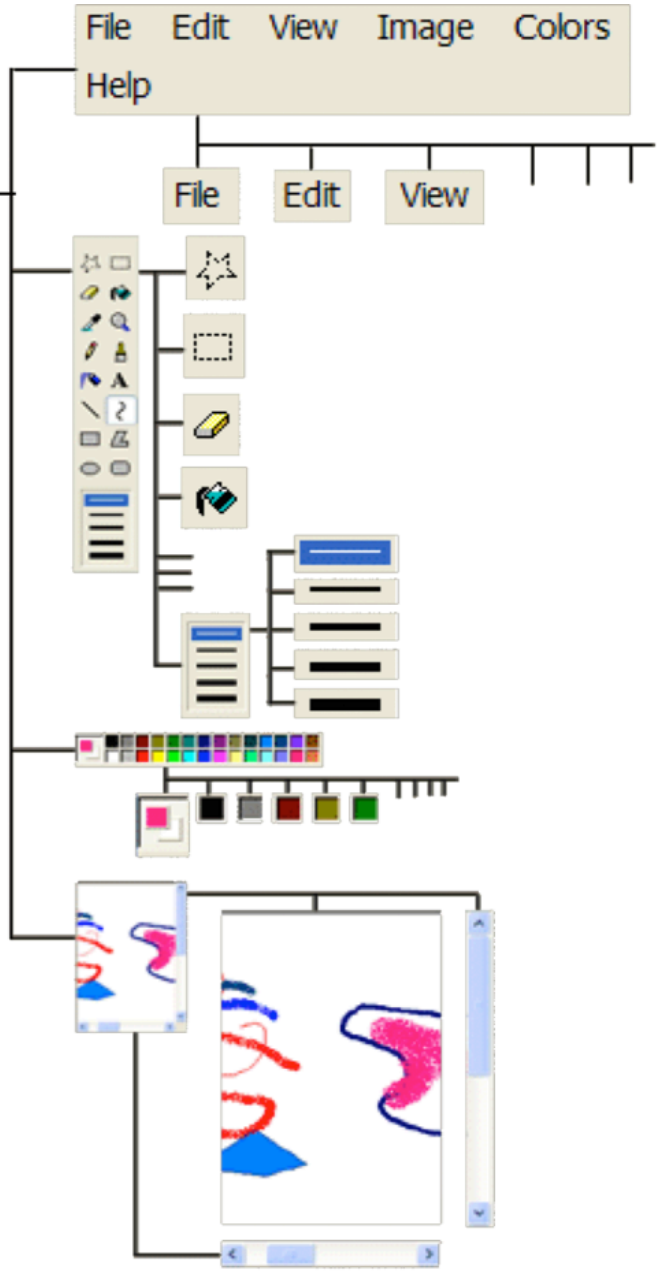

```
xInfo1.display = display;
xInfo1.screen = screen;
initX(argc, argv, xInfo1, DefaultRootWindow(
display), 100, 100, 800, 600);

xInfo2.display = display;
xInfo2.screen = screen;
initX(argc, argv, xInfo2, xInfo1.window,
            50, 50, 300, 200);
```

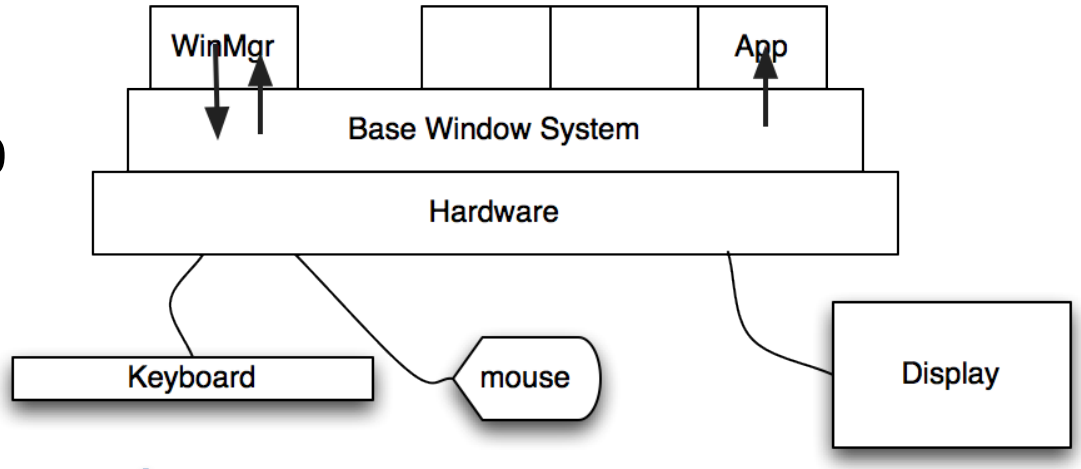
Example of a window being embedded as a widget. The BWS can dispatch events directly to a widget in a heavyweight toolkit.

Lightweight: Window Dispatch

BWS needs to select the appropriate application window for dispatch, then the window needs to send to a widget.

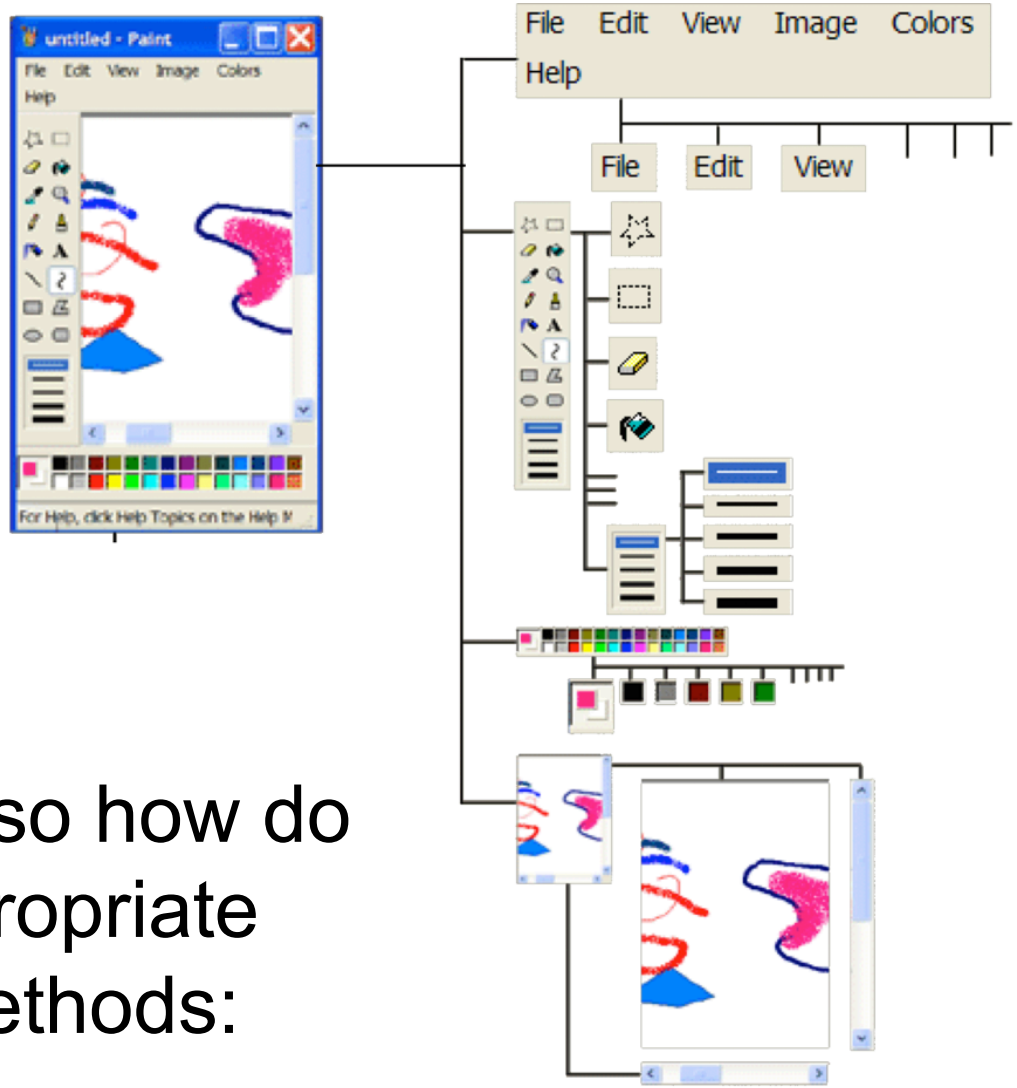






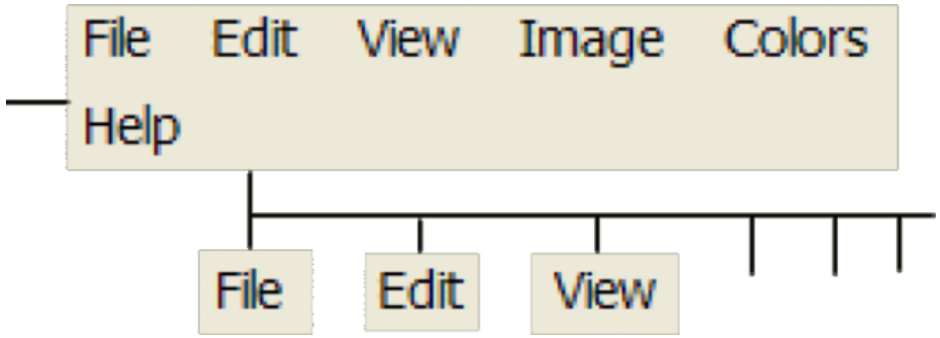
Positional Dispatch

- Send input to the widget under mouse cursor.
- The front-most widget under the cursor should receive the event.
- Widgets can overlap, so how do we determine the appropriate target widget? Two methods:
  - Bottom-up positional dispatch
  - Top-down positional dispatch



Bottom-up Positional Dispatch

- Bottom-Up Positional Dispatch
  - Event is first routed to **leaf node widget in the interactor tree** corresponding to location of mouse cursor
  - *Leaf node has the first opportunity to act on that event*
  - The leaf node widget can either:
    1. process the event itself, or
    2. pass the event to its parent
       (who can process it or send to its parent...)



Passing to Parent

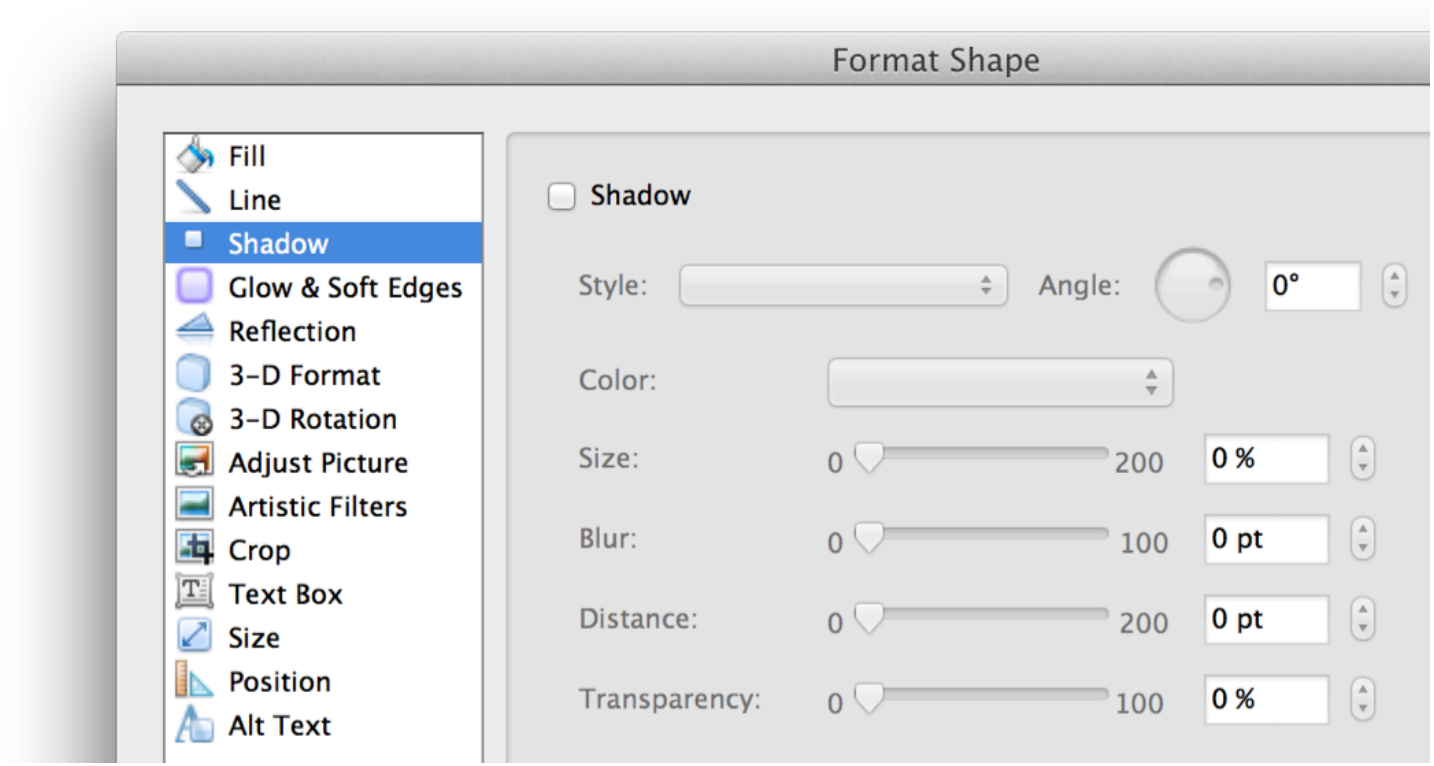
- Why would a widget pass an event to its parent?
  - Example: A palette of colour swatches may implement the colours as buttons. But palette needs to track the currently selected colour. Easiest if the palette deals with the events.

Top-down Positional Dispatch

- Top-Down Positional Dispatch
  - Event is routed to widget in the **highest-level node** in the interactor tree that contains the mouse cursor
    - Can process the event itself, and/or
    - Can pass it on to a child component
  - Key idea is that highest-level node has first chance at acting on the event
  - Uses:
    - Can create *policies* enforced by the parent
      - For example, stopping events if all children are disabled
    - Supports relatively easy logging of events for later replay

Top-down vs. Bottom-up Dispatch

- The toolkit determines the type of dispatch used (not the programmer!)
  - To end-user, no discernable difference in how the interface behaves
  - Just slightly different implementations



disabled container widget policy

Positional Dispatch Limitations

- Positional dispatch can lead to odd behavior:
- e.g. we normally send keystrokes to scrollbar if the mouse over the scrollbar
  - What if the mouse-drag starts in a scrollbar, but then moves outside the scrollbar: do we continue sending the events to the scrollbar? To the adjacent widget instead?
  - What if the mouse-press event occurs over one button but the release is over another widget: does each widget gets one of the events?
- Sometimes position isn't enough, also need to consider which widget is "in focus"

Focus Dispatch

- Alternate & complementary dispatch mechanism.
  - Events dispatched to widget that has focus, 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**: Mouse down on button, move off, mouse up … also called “mouse capture”
- Maximum one keyboard focus and one mouse focus
- Need to gain and lose focus at appropriate times
  - Transfer focus on mouse down
  - Transfer focus on a tab

Focus Dispatch *Needs* Positional Dispatch

- But if a widget has focus, it should not receive every event:
  - mouse down on another suitable widget should change focus
- Often helpful to have an explicit focus manager in a container widget to manage which widget has the focus.

Accelerator Key Dispatch

- Accelerator keys can by pass focus dispatch
  - Keyboard events dispatched based on **which keys** are pressed
  - Register special keyboard accelerators with specific commands
    - commands are often the target of menu item events
  - The GUI toolkit intercepts accelerators and forwards to the appropriate command handler

Toolkit Summary

- BWS and widget cooperate to dispatch events.
  - Heavyweight toolkits:
    - BWS has visibility into all widgets in the application.
      - BWS can dispatch top-down (application window) or bottom-up (directly to widget).
  - Lightweight toolkits:
    - BWS only has visibility to the application window.
      - BWS can dispatch top-down only (i.e. to the application window).
      - Toolkit dispatches to the widget.
        - » In Java, the JVM manages dispatch.

Dispatch Summary

- Mouse-down events are almost always positional: dispatched to widget under cursor (top-down or bottom-up)
- Other mouse and keyboard events go to widget in focus
- Positional and focus dispatch work together!