Model-View-Controller and Event Driven UI

CS160: User Interfaces

John Canny

Includes slides based on those of James Landay & Jeffrey Heer

Review

Persuasion and Games

Rhetoric

- Ethos, Logos, Pathos

The psychology of persuasion

- 9 principles and application to games

Persuasion and interactivity

- Dialogic presentation of information is more persuasive.

Topics

2D graphics fundamentals

Interactive application programming

- Component Model
- Event-Driven User Interfaces

Model-View-Controller

- Architecture for interactive components
- Why do we need it?
- Changing the display

2-D Computer Graphics

Models for images

- Strokes, pixels, regions

Coordinate systems

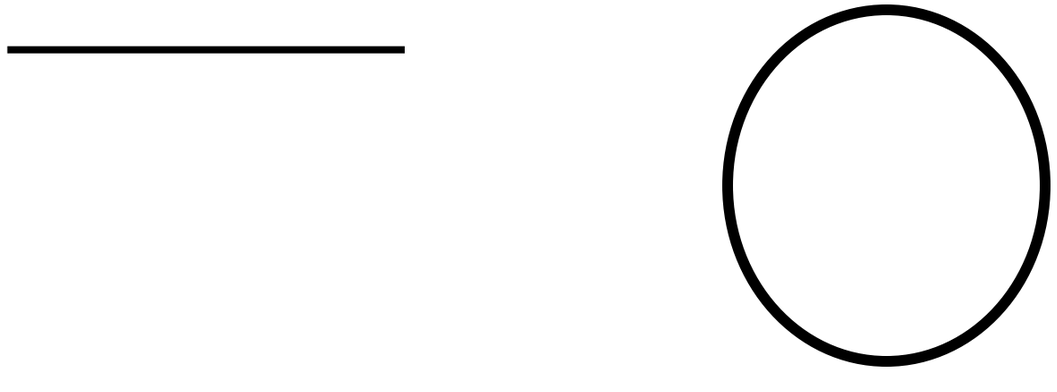
- Device, physical

Canvas

Drawing

- Paths, shapes, text

Stroke Model



Describe image as strokes (w/ color/thickness)

- `Line ((10, 4), (17,4), thick 2, red)`
- `Circle (( 19, 13), radius 3, thick 3, white)`

Maps to early vector displays & plotters

Most UI toolkits have stroked objects

- arcs, ellipses, rounded rectangles, etc.

Problems with Stroke Model?

How would you represent with strokes?
Solution?

Pixel Model

Break-up complex images into discrete "pixels" & store color for each

Resolution

- Spatial: number of rows by columns
- e.g., 1280 x 1024 is a good monitor display
- Quality laser printer: 10200 x 13200 (1200 dpi)
- Image depth (i.e., number of bits per pixel)
- Several styles... 8-bit, 24-bit, 32-bit

Image Depth

Bit map - 1 bit/pixel (on/off)

- B&W screens or print-outs

Image Depth (cont.)

Gray scale - 2-8 bits/pixel

Full color - 24 bits/pixel

- 8 bits per primary color (Red, Green, Blue)

Image Depth (cont.)

Full color – 32 bits/pixel

- Usually just 24-bit color (used for efficiency)
- Extra 8-bits are optional – can be used for "alpha" (transparency)

Color mapped - 8 bits/pixel

- Store index @ pixel - map into table w/ 24 bits
- Cuts space & computation
- Problem????

Image Depth (cont.)

- Jpeg image of blue sky

Image Depth (cont.)

- Blue sky with limited image depth

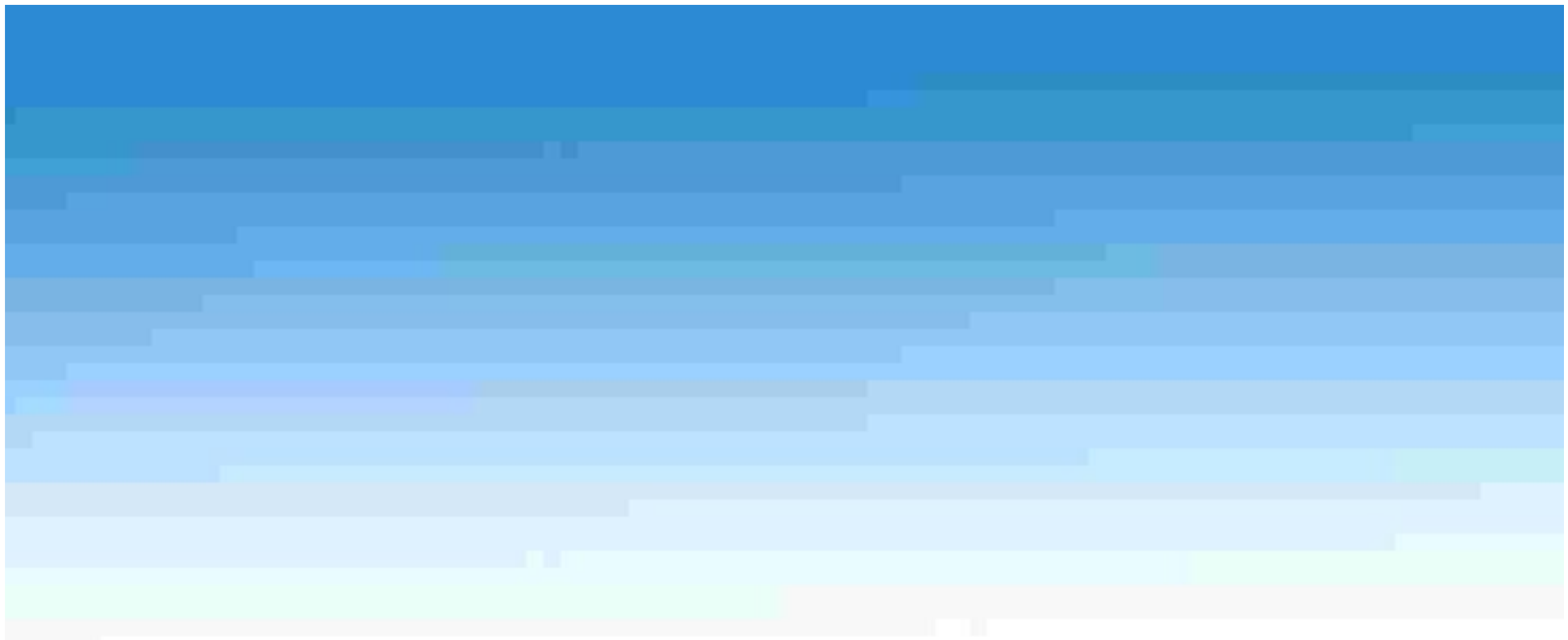


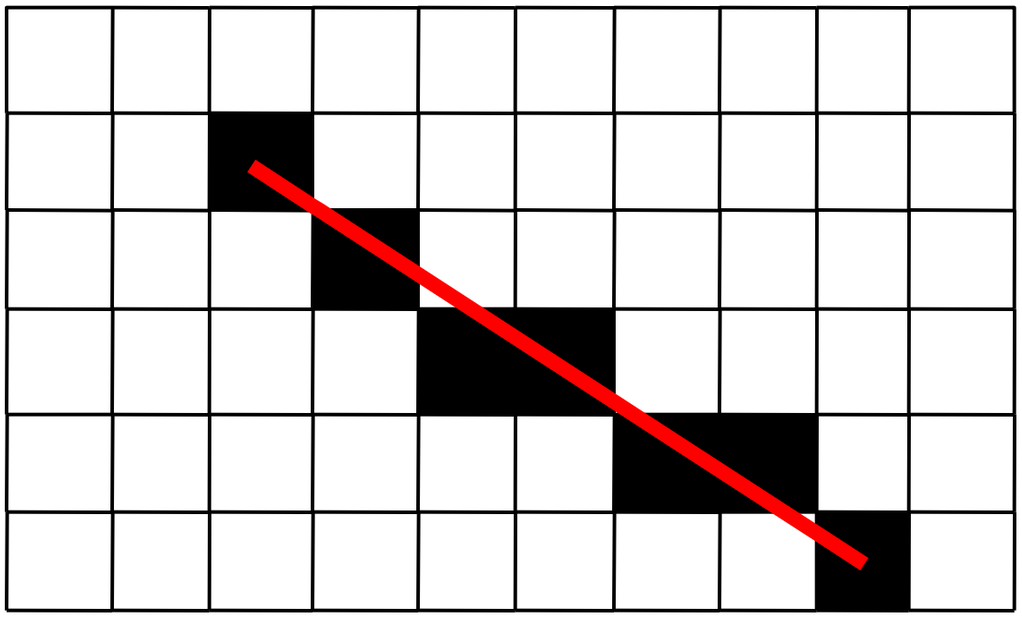
Image compression

Lossless formats: BMP, TIFF

JPEG: based on image spectral analysis. Best for natural images.

GIF/PNG: Best for line art and synthetic images.

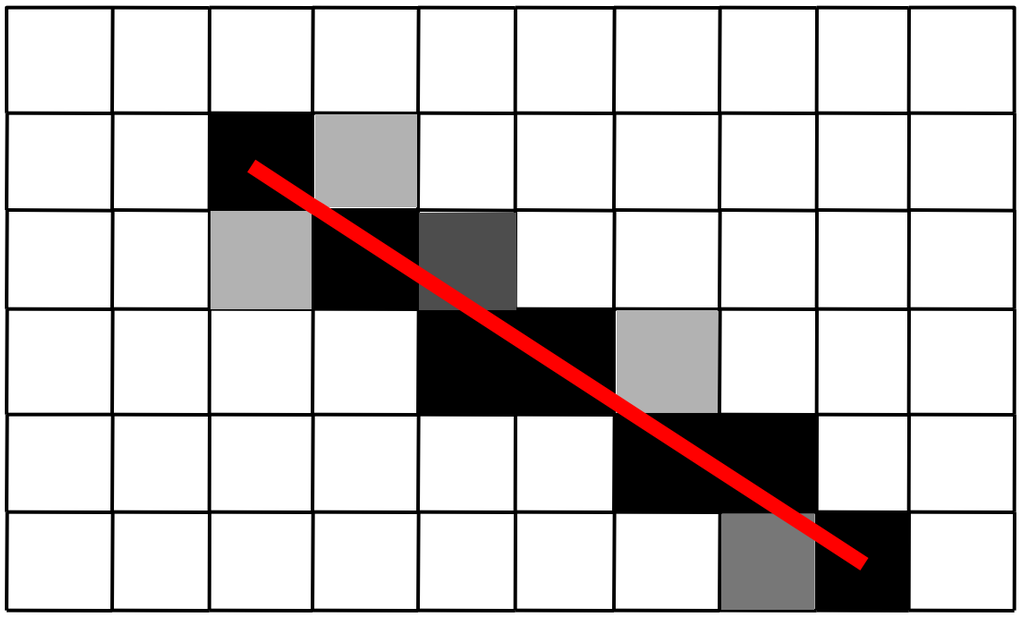
Aliasing



Smooth objects (e.g., lines) appear jagged since resolution is too low

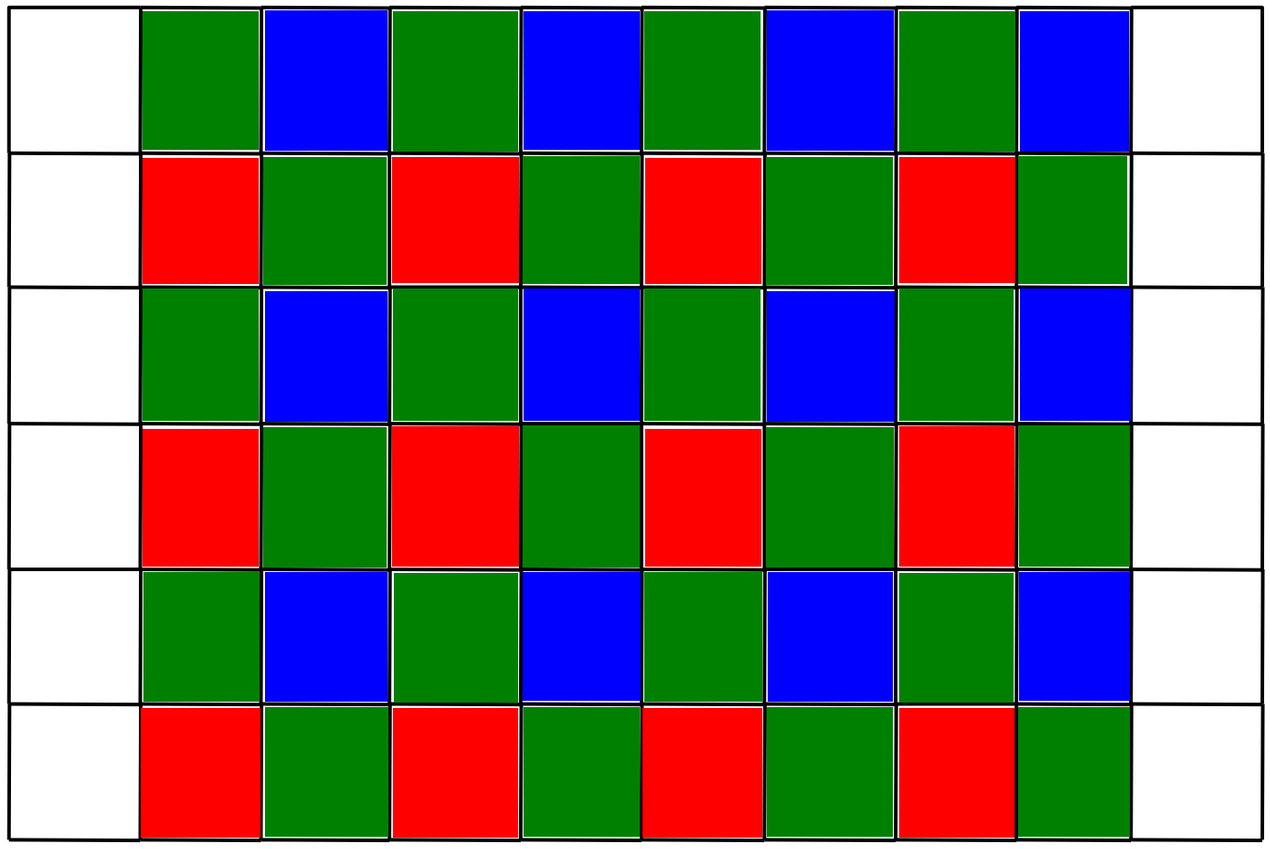
Antialiasing - fill-in some jagged places w/ gray scale or primary colors

Anti-Aliasing



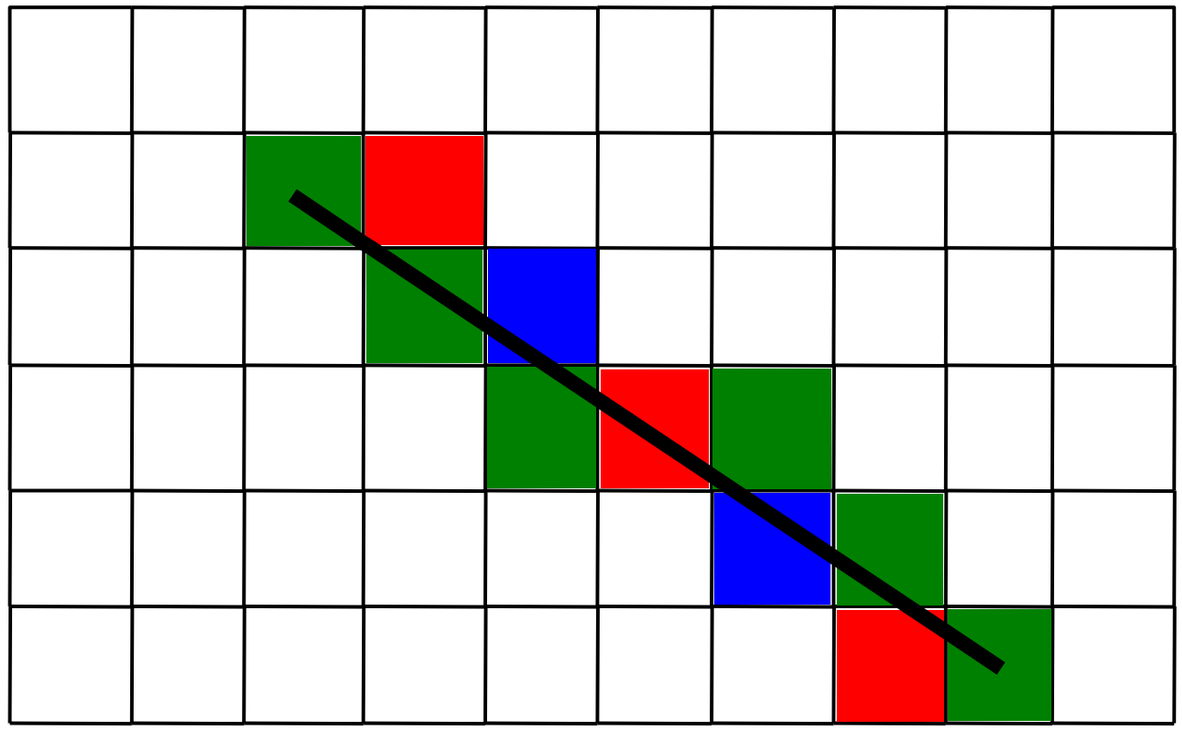
- Pixels colored in proportion to relative amount of line that crosses them.
- Equivalently, draw the line in B/W at finer resolution and then color each pixel in proportion to number of colored sub-pixels.

Cleartype



The pixel matrix for a laptop or LCD screen.

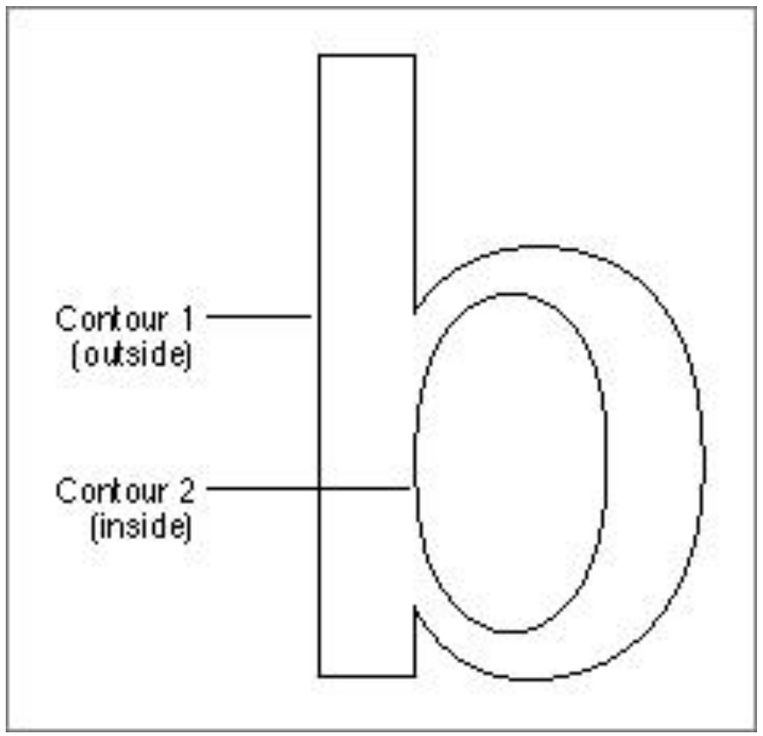
Cleartype



Use sub-pixel color pixels as though they were gray pixels (can cause color anomalies).

Outline Fonts

Used by both Postscript & TrueType



Boundary is represented with splines, and can be scaled to any size.

Vector formats

Vector graphics increasingly popular as the rendering platform to support many device types:

- Flash
- SVG (Scalable Vector Graphics) and XML format
- XAML (eXtensible Application Markup Language) and WPF (Windows Presentation Foundation), the heart of Vista.
- VML (Microsoft) in Powerpoint/Internet Explorer.

Interactive Application Programming

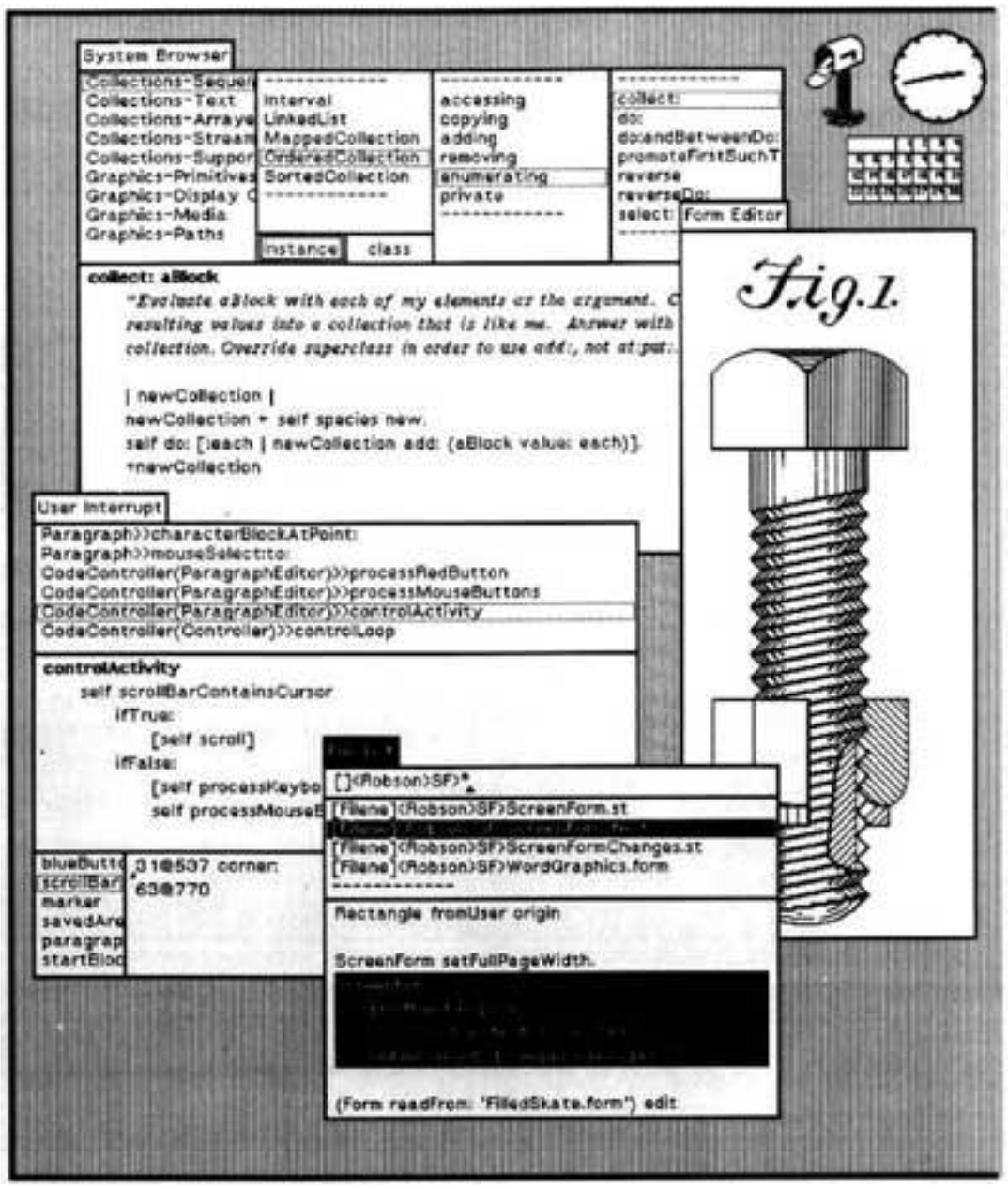
In the beginning...

```
bash-2.05b$ pwd
/home/dstone
bash-2.05b$ cd /usr/portage/app-shells/bash
bash-2.05b$ ls -al
total 68
drwxr-xr-x    3 root root  4096 May 14 12:05 .
drwxr-xr-x   26 root root  4096 May 17 02:36 ..
-rw-r--r--    1 root root 13710 May  3 22:35 ChangeLog
-rw-r--r--    1 root root  2924 May 14 12:05 Manifest
-rw-r--r--    1 root root  3720 May 14 12:05 bash-2.05b-r11.ebuild
-rw-r--r--    1 root root  3516 May  2 20:05 bash-2.05b-r9.ebuild
-rw-r--r--    1 root root  5083 May  3 22:35 bash-3.0-r11.ebuild
-rw-r--r--    1 root root  4038 May 14 12:05 bash-3.0-r7.ebuild
-rw-r--r--    1 root root  3931 May 14 12:05 bash-3.0-r8.ebuild
-rw-r--r--    1 root root  4267 Mar 29 21:11 bash-3.0-r9.ebuild
drwxr-xr-x    2 root root  4096 May  3 22:35 files
-rw-r--r--    1 root root   164 Dec 29  2003 metadata.xml
bash-2.05b$ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
</pkgmetadata>
bash-2.05b$ sudo /etc/init.d/bluetooth status
Password:
 * status:  stopped
bash-2.05b$ ping -q -c1 en.wikipedia.org
PING rr.chtpa.wikimedia.org (207.142.131.247) 56(84) bytes of data.

--- rr.chtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 112.076/112.076/112.076/0.000 ms
bash-2.05b$ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1               /mnt/usbkey
/dev/sda2               /mnt/ipod
bash-2.05b$ date
Wed May 25 11:36:56 PDT 2005
bash-2.05b$ lsmod
Module                  Size  Used by
joydev                  8256  0
ipw2200               175112  0
ieee80211              44228  1 ipw2200
ieee80211_crypt         4872  2 ipw2200,ieee80211
e1000                  84468  0
bash-2.05b$
```

http://www.cryptonomicon.com/beginning.html

The Xerox Alto (1973)



Event-Driven UIs

Old model (e.g., UNIX shell, DOS)

- Interaction controlled by system, user queried for input when needed by system

Event-Driven Interfaces (e.g., GUIs)

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

Component/Widget Model

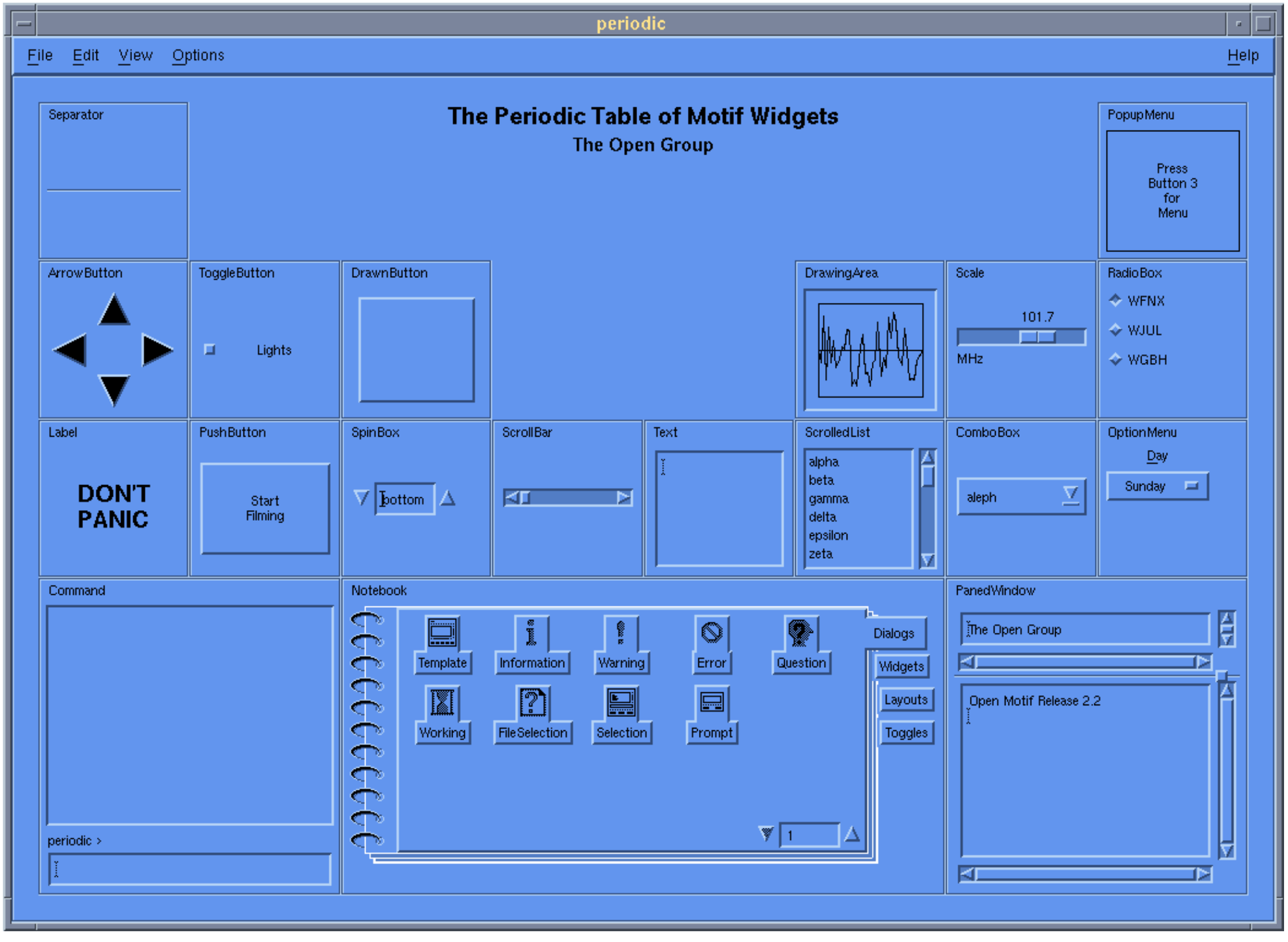
- Encapsulation and organization of interactive components ("widgets")
  - Typically using a class hierarchy with a top-level "Component" type implementing basic bounds management, and event processing
- Drawn using underlying 2D graphics library
- Input event processing and handling
  - Typically mouse and keyboard events
- Bounds management (damage/redraw)
  - Only redraw areas in need of updating

What are Some Examples of Components?

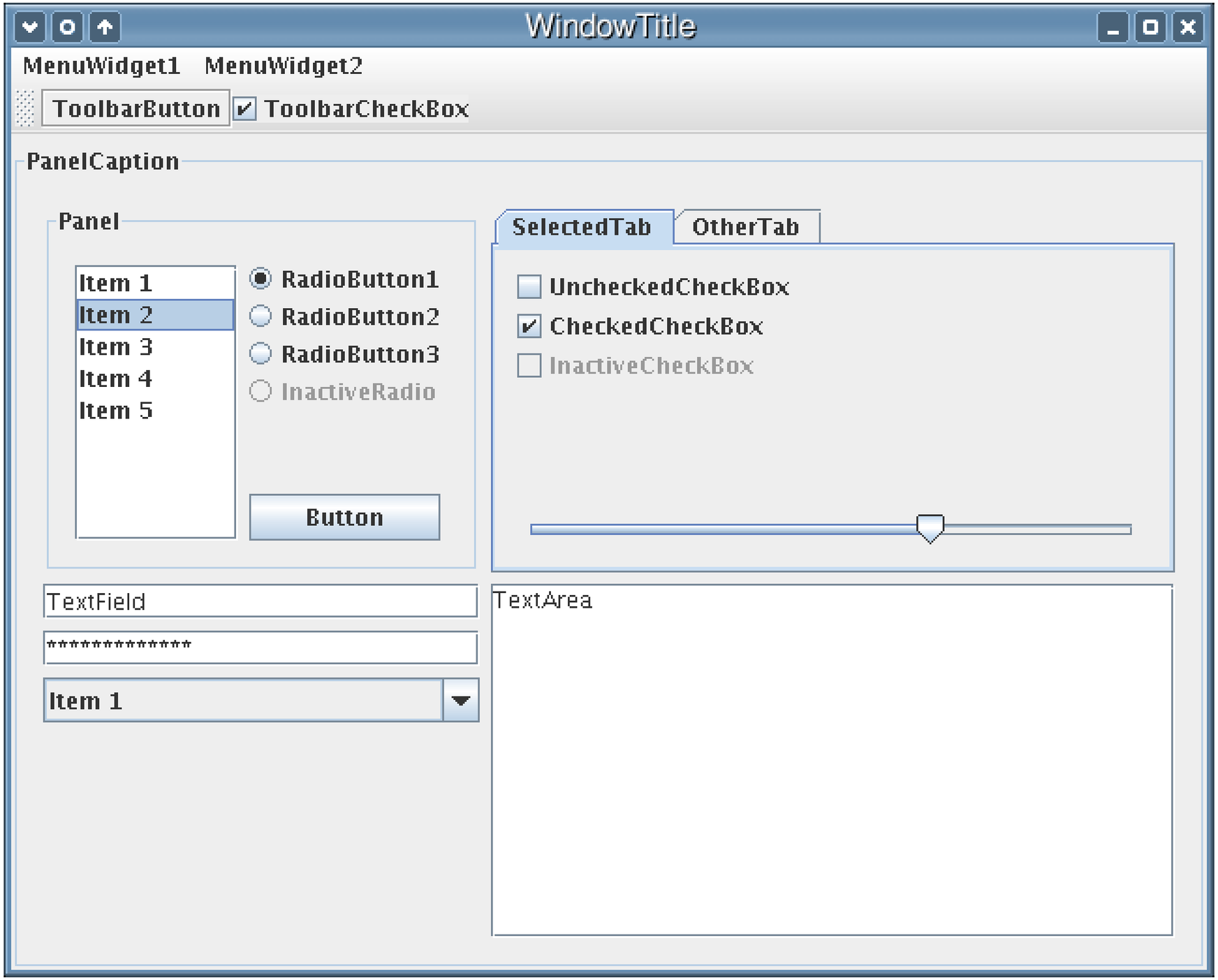
What are Some Examples of Components?

-Windows
-Layout panels
-Drawing panes
-Buttons
-Sliders
-Scrollbars
-Images
-Dropdown boxes
-Toolbars
-Menus
-Dialogue Boxes
-Progress indicators
-Video
-Icons
-Links
-Checkboxes
-Radio buttons
-Etc.

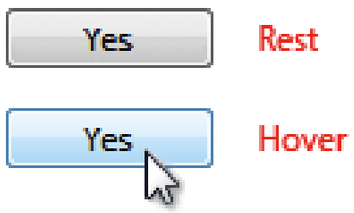
Periodic Table of Motif Widgets



Java Swing Widgets



Windows Vista/.Net Widgets

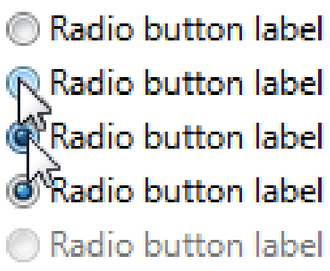


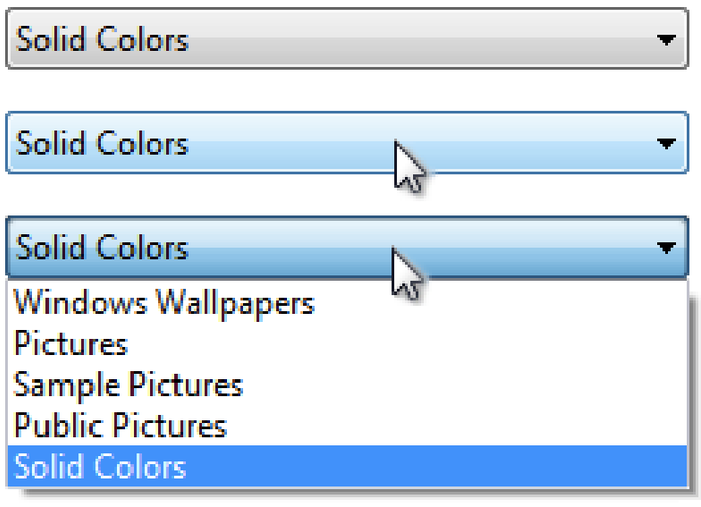
Rest

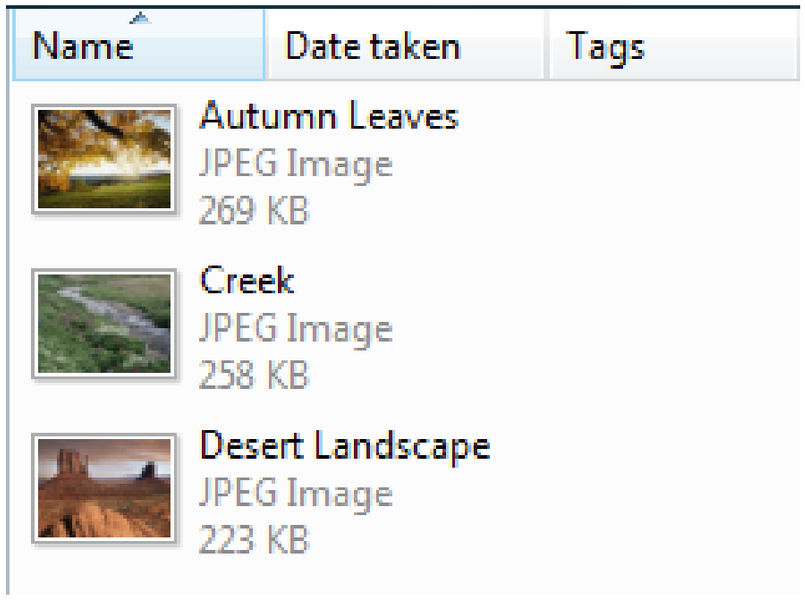
Hover

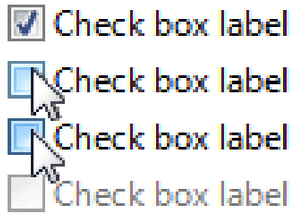
Click to do something

Click to do something









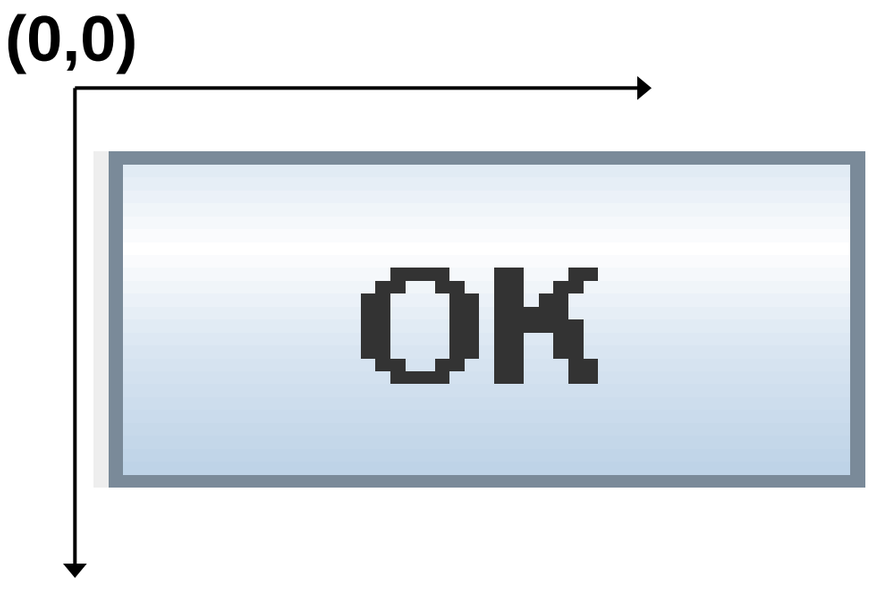
User Interface Components

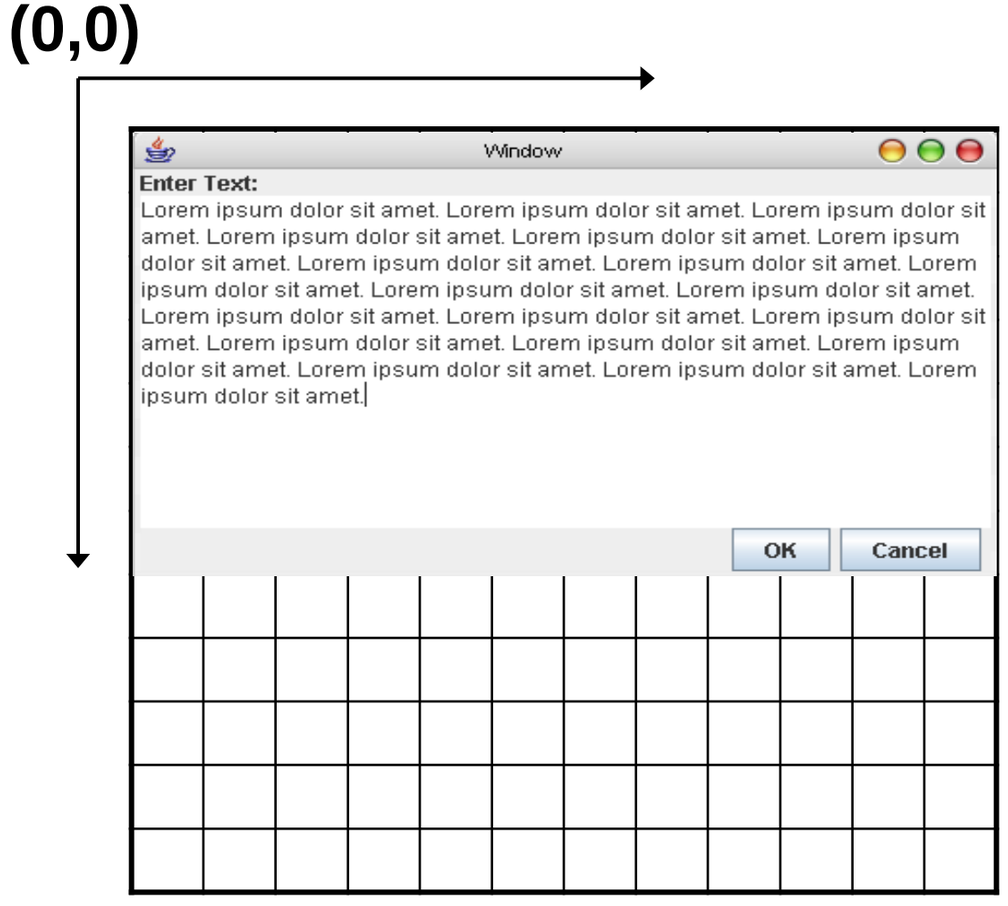
- Each component is an object with
  - Bounding box
  - Paint method for drawing itself
    - Drawn in the component's coordinate system
  - Callbacks to process input events
    - Mouse clicks, typed keys


```
public void paint(Graphics g) {
   g.fillRect(…); // interior
   g.drawString(…); // label
   g.drawRect(…); // outline
}
```

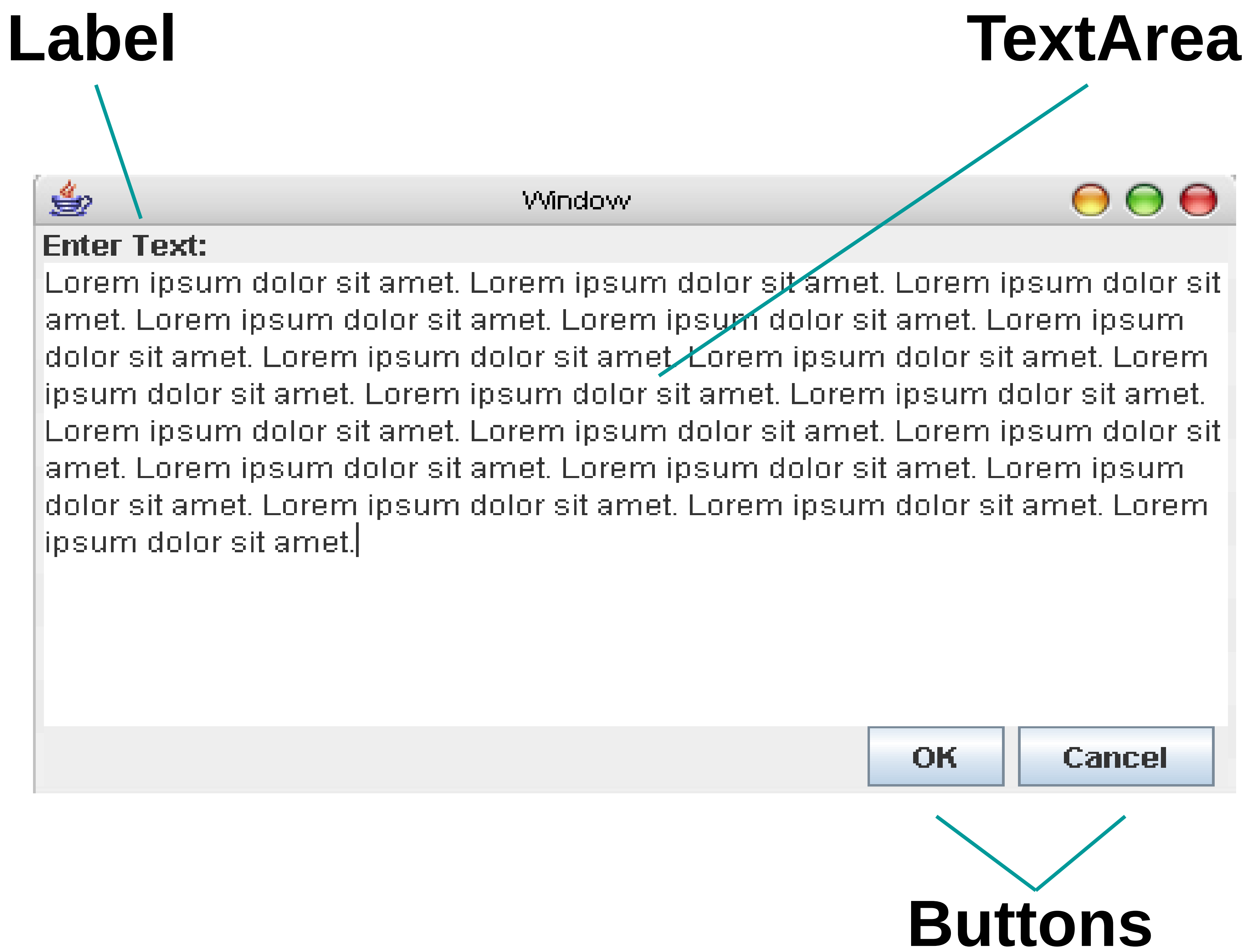
2D Graphics Model

- Every component is a clipped drawing canvas with a coordinate system
  - Origin typically at top-left, increasing down and to the right
  - Units depend on the output medium (e.g., pixels for screen)
  - Rendering methods
    - Draw, fill shapes
    - Draw text strings
    - Draw images



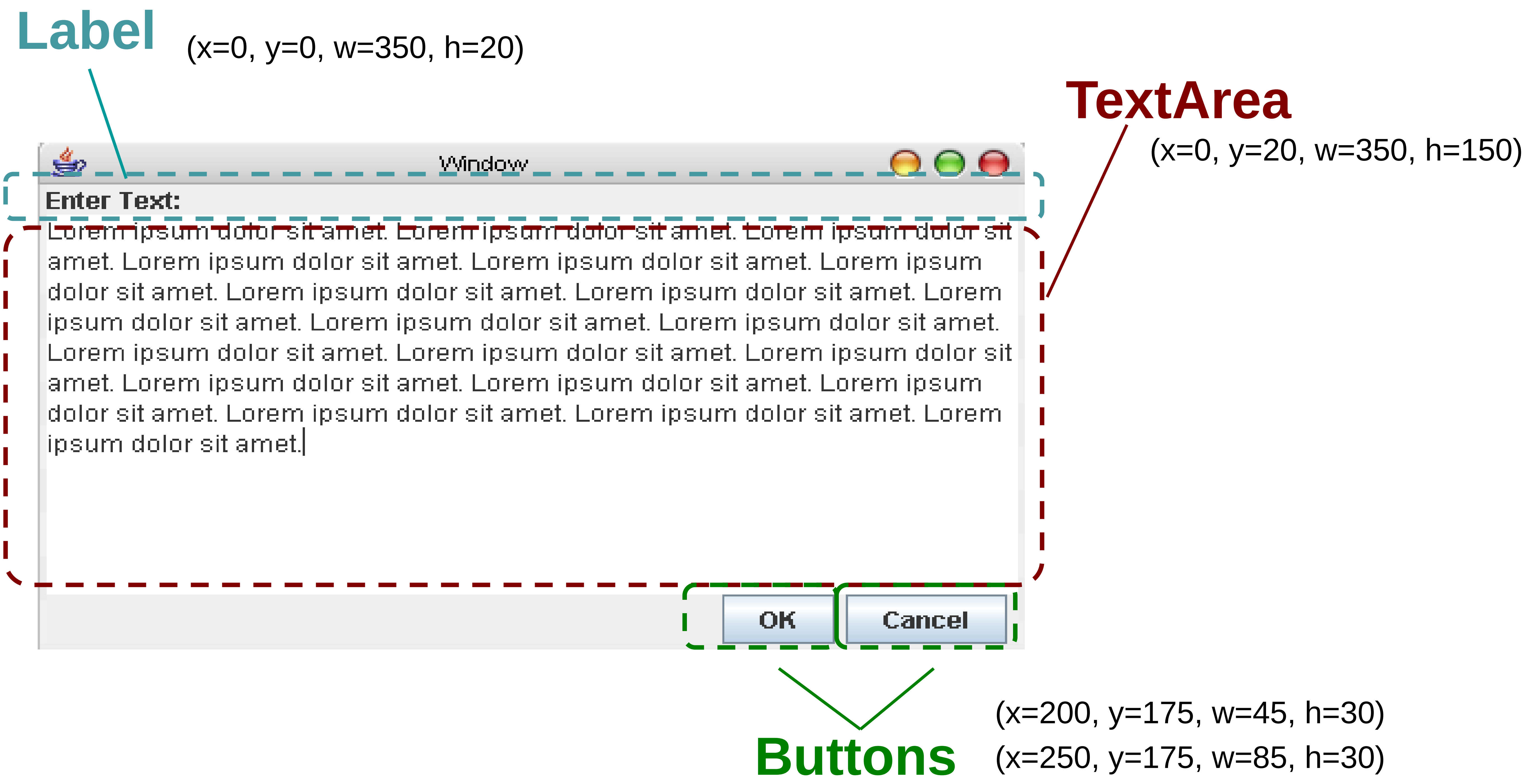


Composing a User Interface



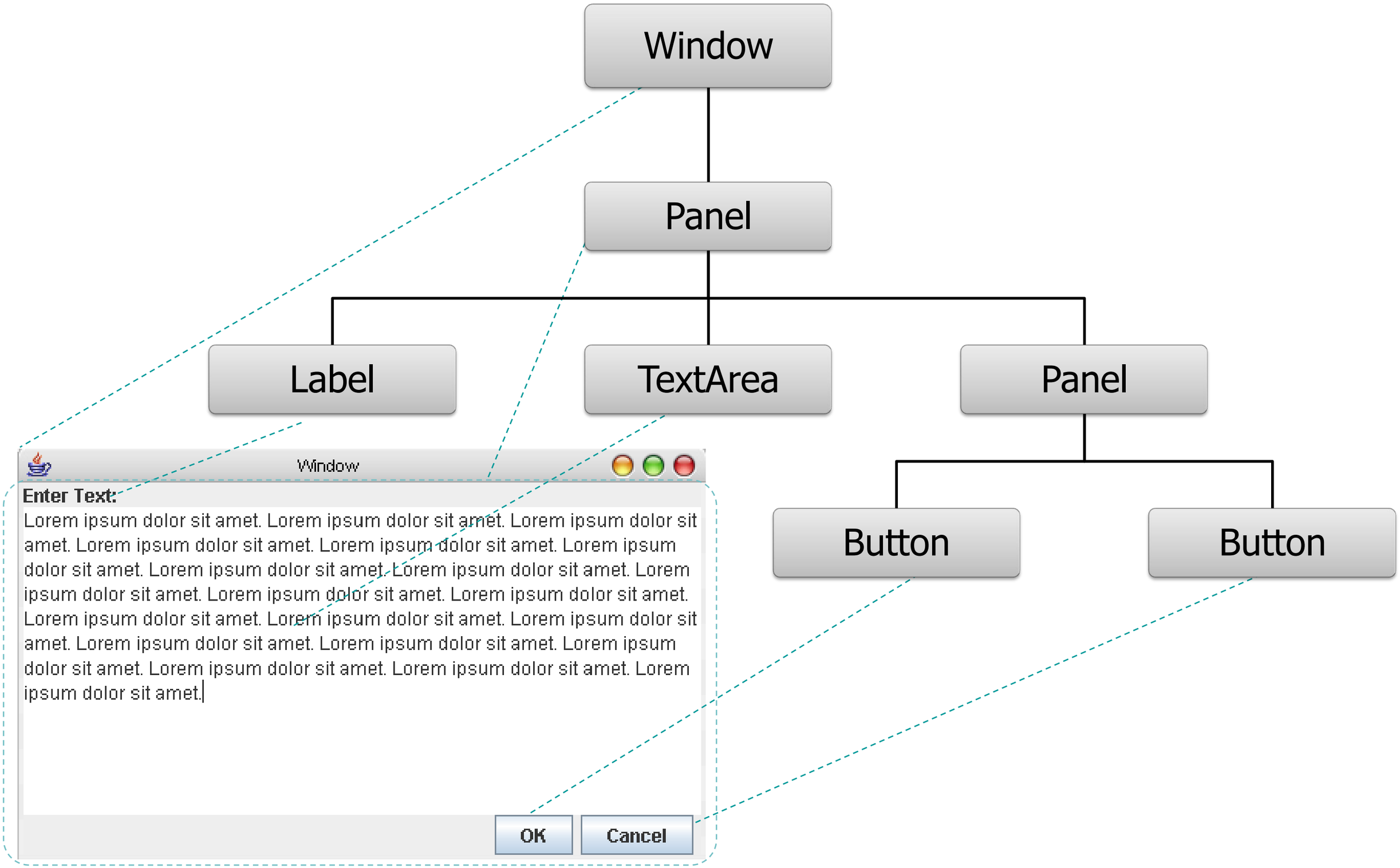
How might we instruct the computer to generate this layout?

Absolute Layout



But this is inflexible and doesn't scale or resize well.

Containment Hierarchy

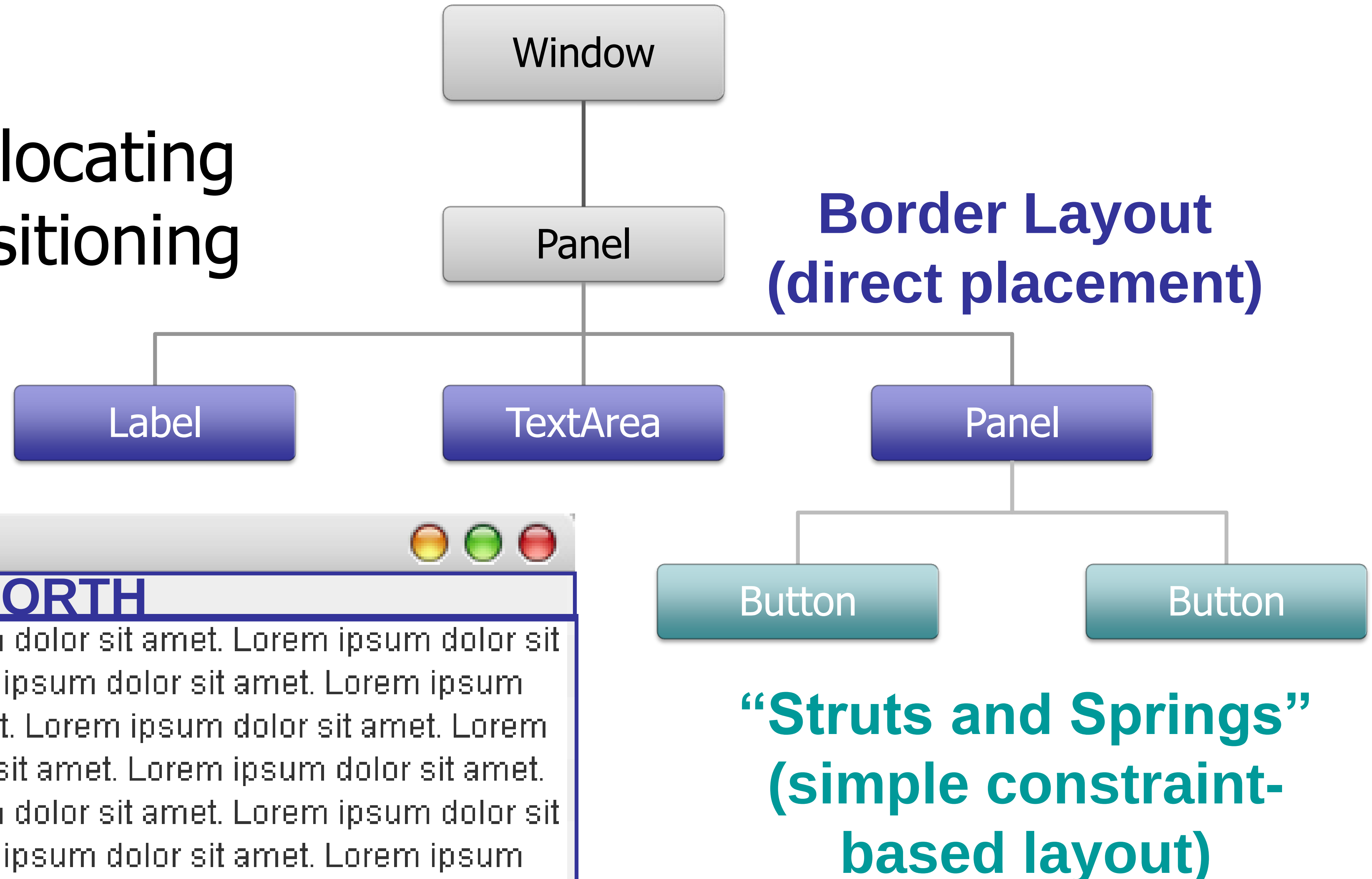


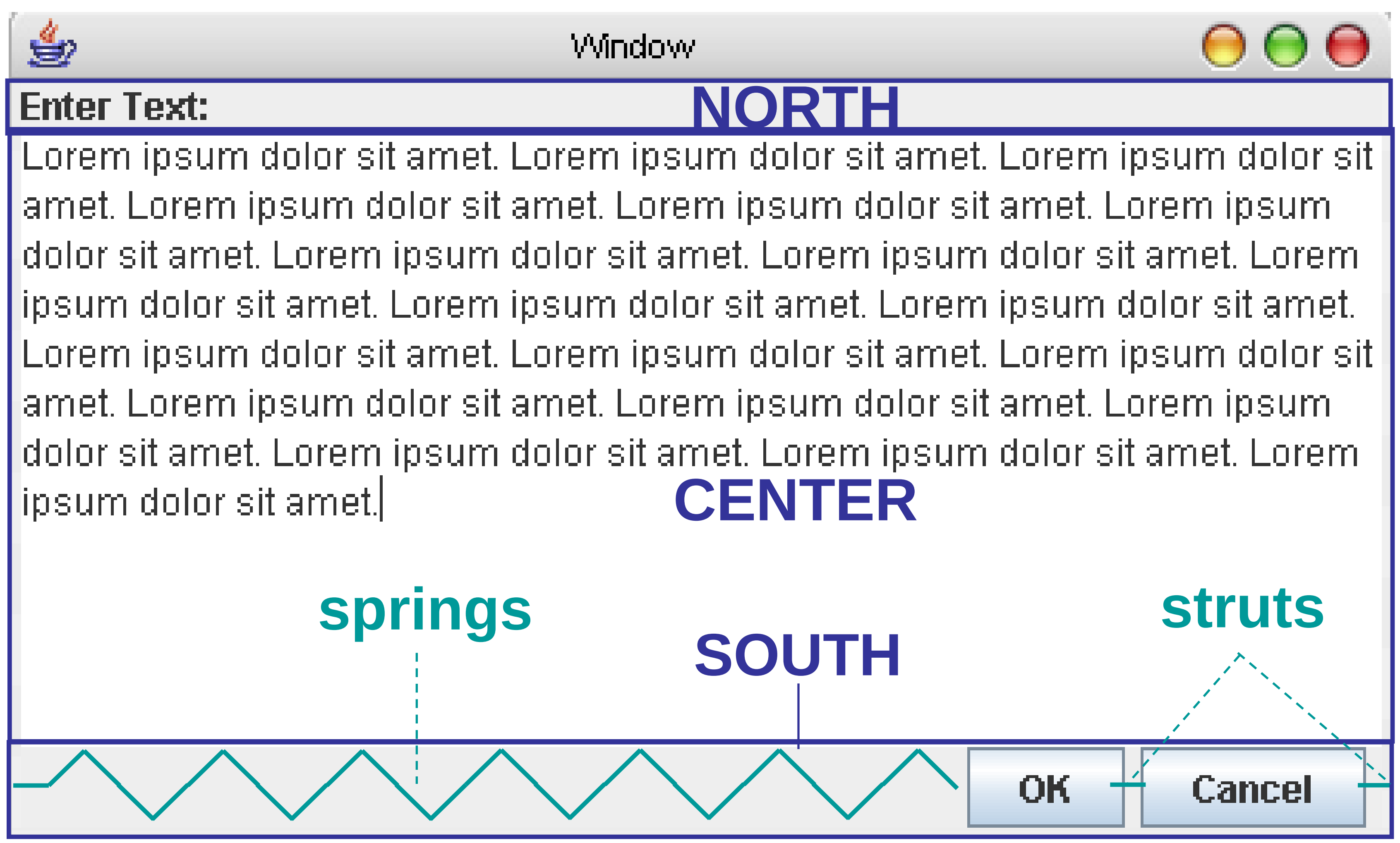
Enter Text:

Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet.

Component Layout

- Each container is responsible for allocating space for and positioning its contents





Events

Events

User input is modeled as "events" that must be handled by the system and applications.

Examples?

- Mouse input (and touch, pen, etc.)
  - Mouse entered, exited, moved, clicked, dragged
  - Inferred events: double-clicks, gestures
- Keyboard (key down, key up)
- Sensor inputs
- Window movement, resizing

Anatomy of an Event

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

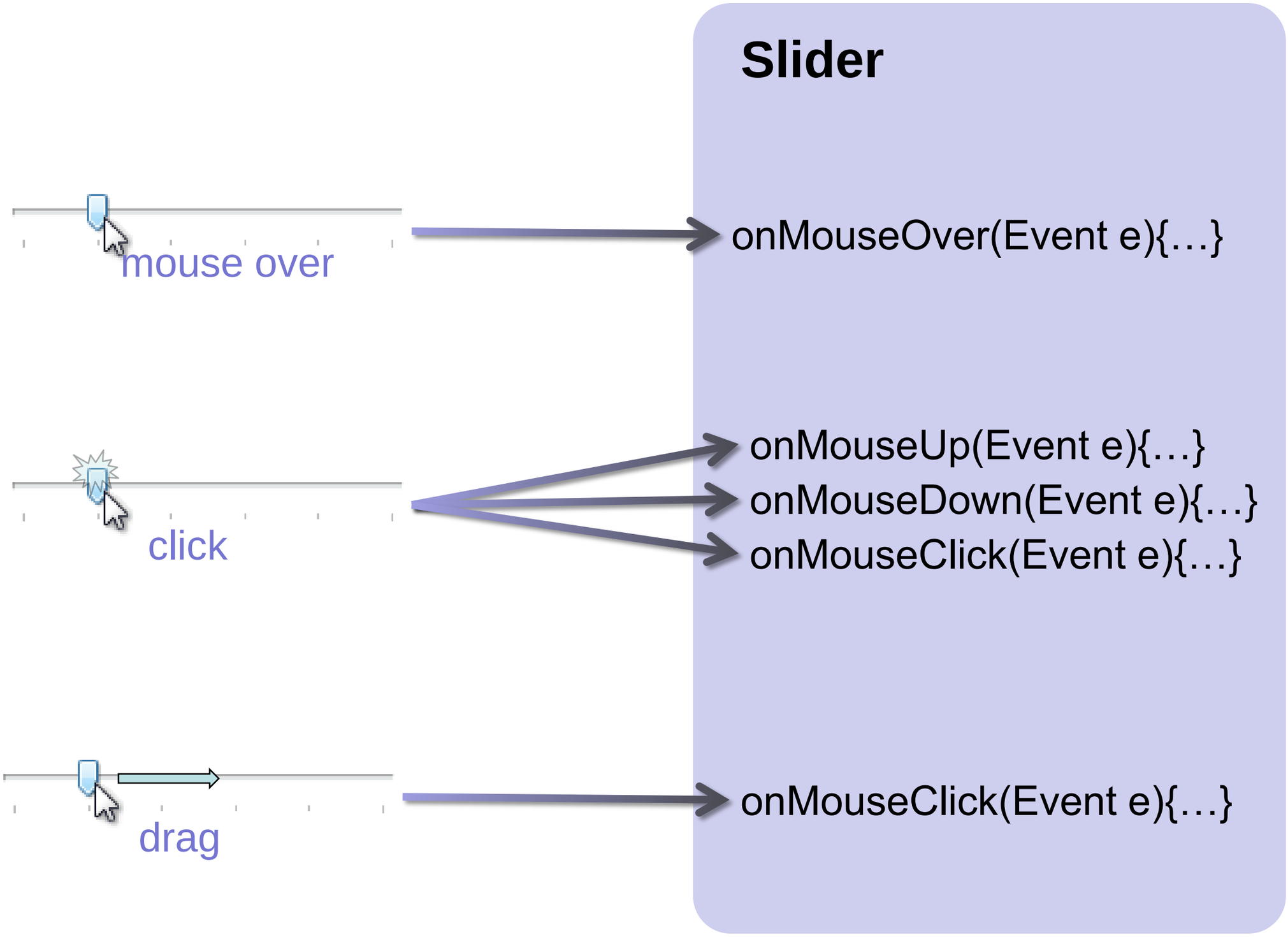
- Event Type (mouse moved, key down, etc)
- Event Source (the input component)
- Timestamp (when did event occur)
- Modifiers (Ctrl, Shift, Alt, etc)
- Event Content
  - Mouse: x,y coordinates, button pressed, # clicks
  - Keyboard: which key was pressed

Events

Level of abstraction may vary. Consider:

- **Mouse down** vs. **double click** vs. **drag**
- **Pen move** vs. **gesture**

Callbacks


Slider
mouse over
onMouseOver(Event e){…}
click
onMouseUp(Event e){…}
onMouseDown(Event e){…}
onMouseClick(Event e){…}
drag
onMouseClick(Event e){…}

Event Dispatch Loop

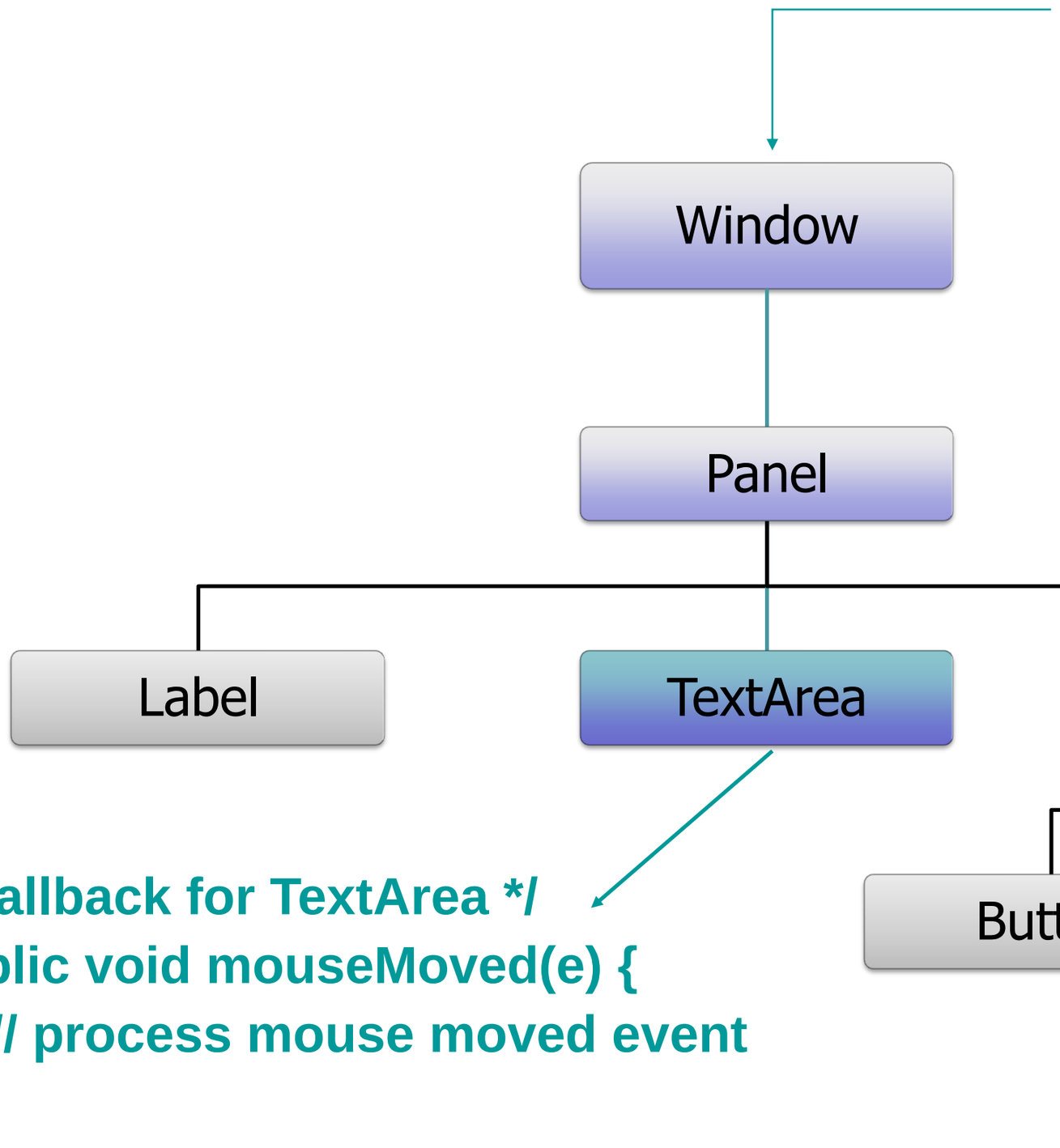
Mouse moved (t_0,x,y)

Event Queue
- Queue of input events

Event Loop (runs in dedicated thread)
- Remove next event from queue
- Determine event type
- Find proper component(s)
- Invoke callbacks on components
- Repeat, or wait until event arrives

Component
- Invoked callback method
- Update application state
- Request repaint, if needed

Event Dispatch



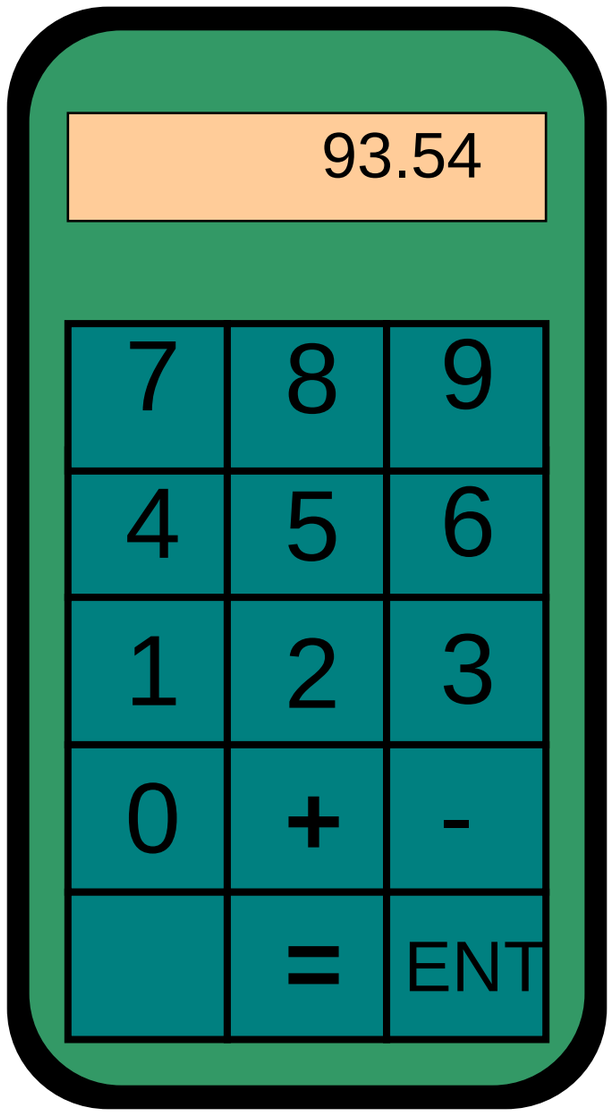
Event Queue

- **Mouse moved (t_0,x,y)**
- **Mouse pressed (t_1,x,y,1)**
- **Mouse dragged (t_2,x,y,1)**
- **Key typed (t_3, 'F1')**
- **…**

(queues and dispatches incoming events in a dedicated thread)

```
/* callback for TextArea */
public void mouseMoved(e) {
    // process mouse moved event
}
```

Interactor Tree



- Display Screen
  - Outer Win [*black*]
    - Inner Win [*green*]
      - Result Win [*tan*]
        - Result String
      - Keypad [*Teal*]
        - = button
        - **-** button
        - **+** button
        - **0** button

Model-View-Controller Architecture

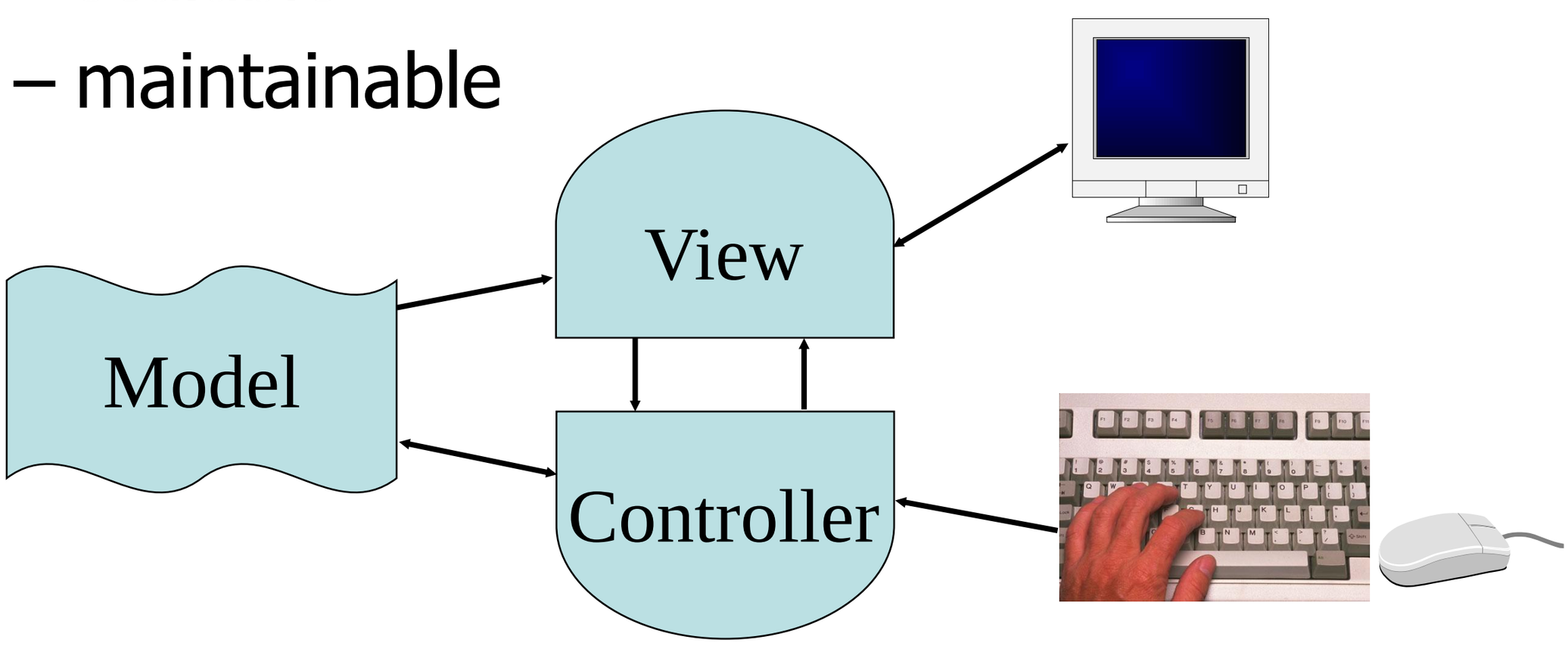
Model-View-Controller

Architecture for interactive apps

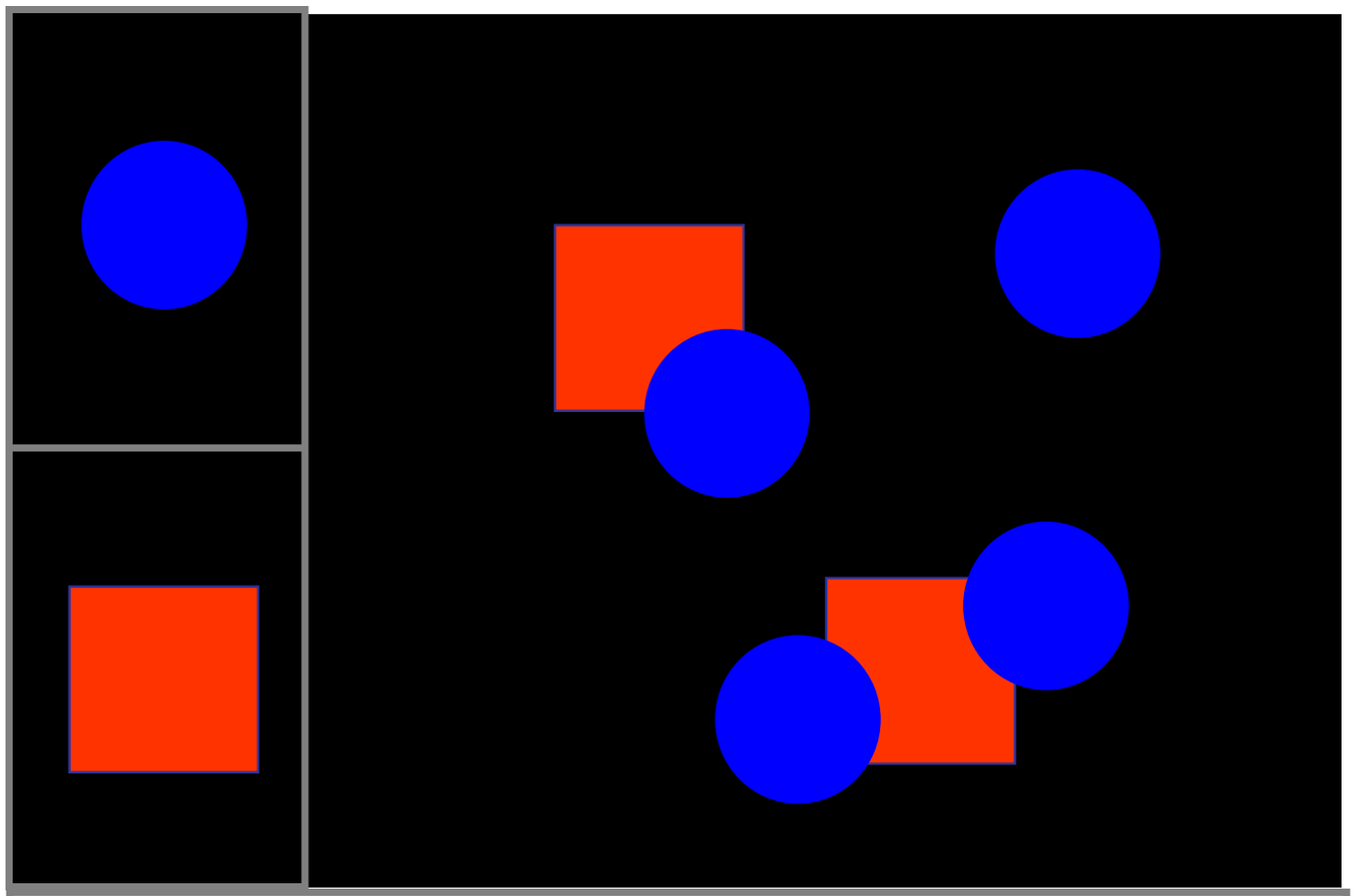
- introduced by Smalltalk developers at PARC

Partitions application in a way that is

- scalable
- maintainable



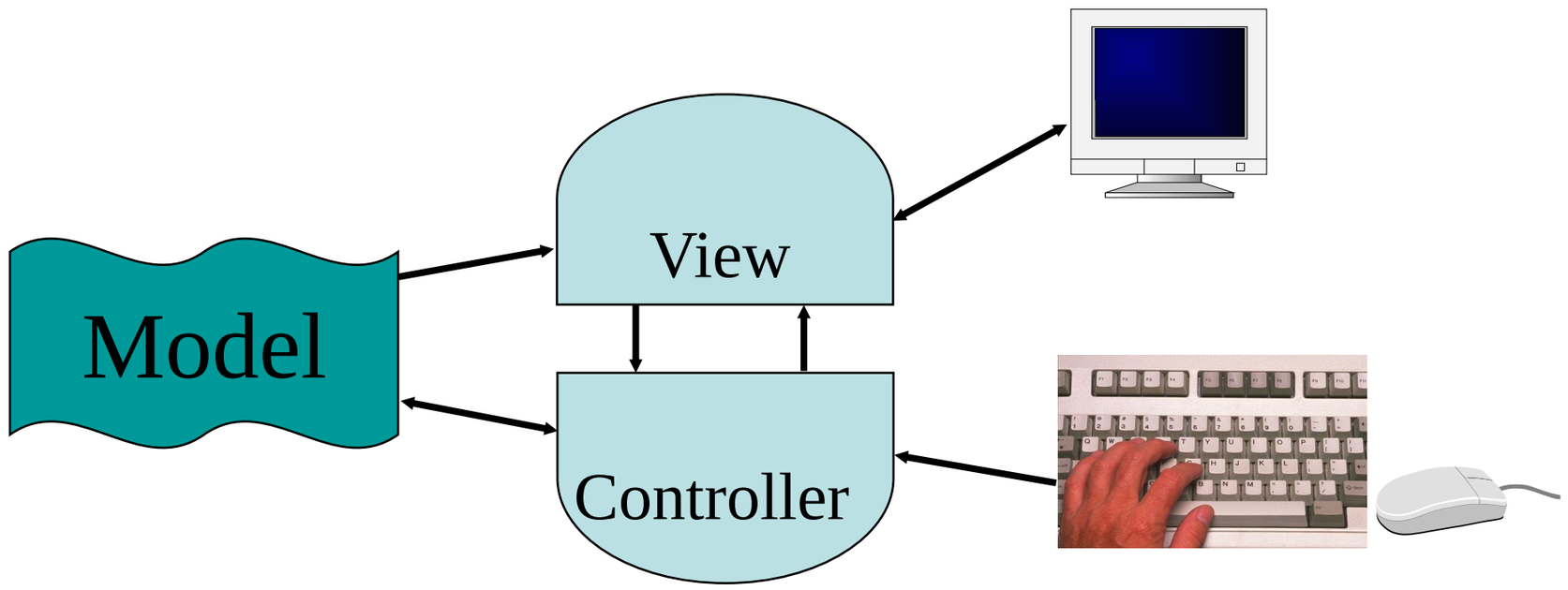
Example Application



Blue circles: 4

Cardinal squares: 2

Model

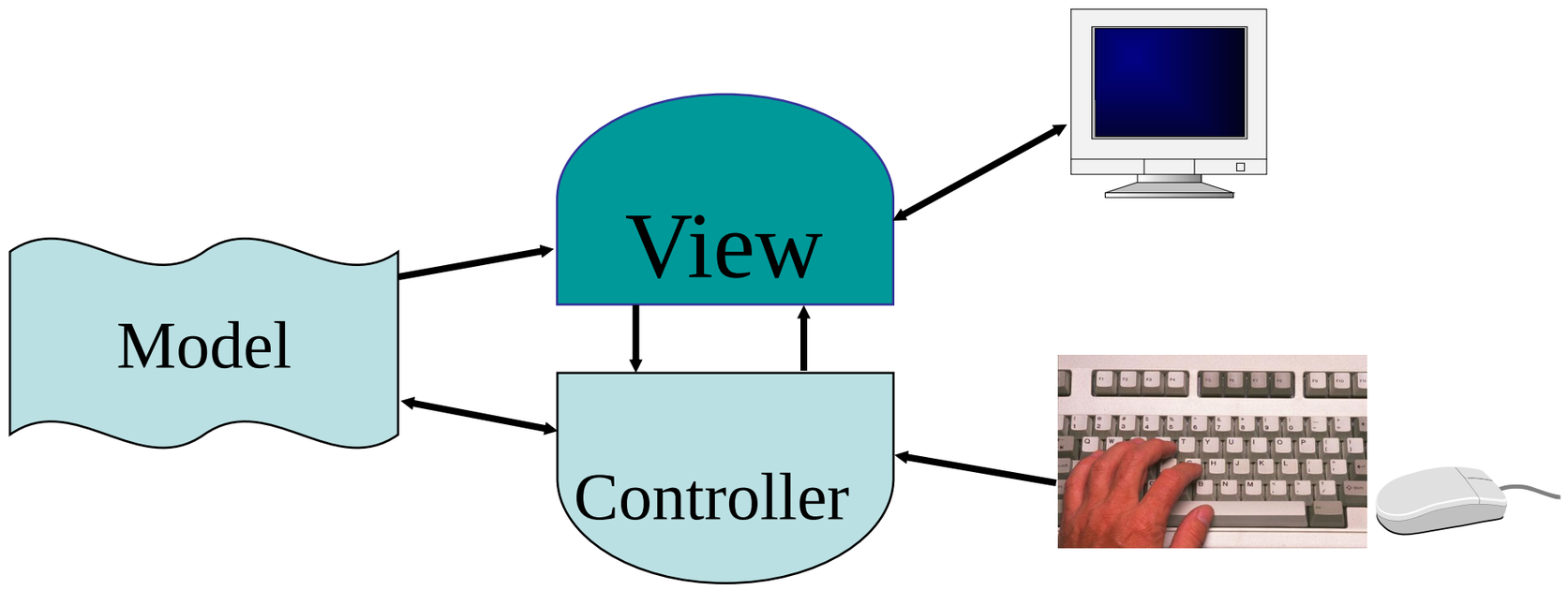


Information the app is trying to manipulate

Representation of real world objects

- circuit for a CAD program
  - logic gates and wires connecting them
- shapes in a drawing program
  - geometry and color

View

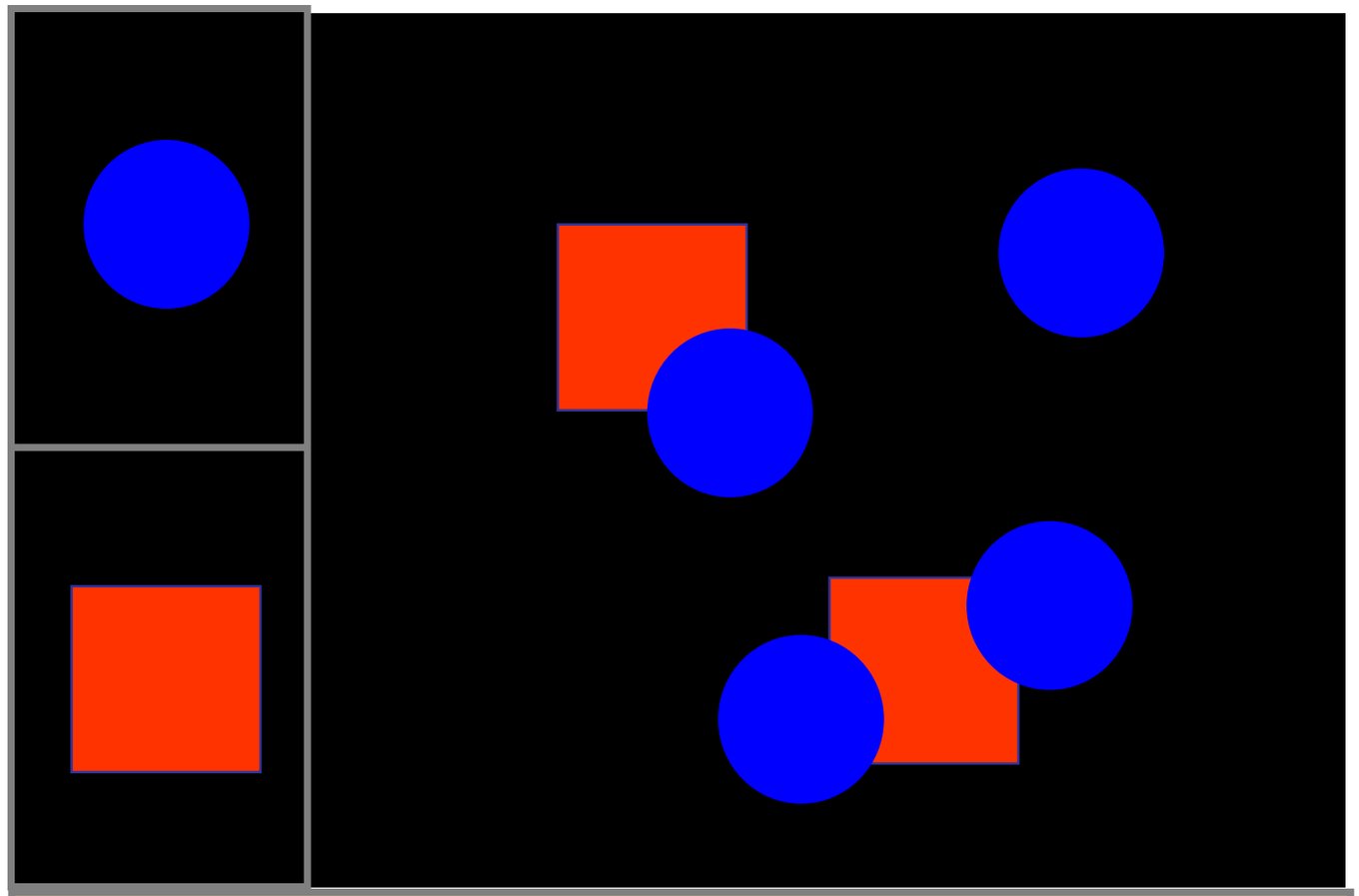


Implements a visual display of the model

May have multiple views

- e.g., shape view and numerical view

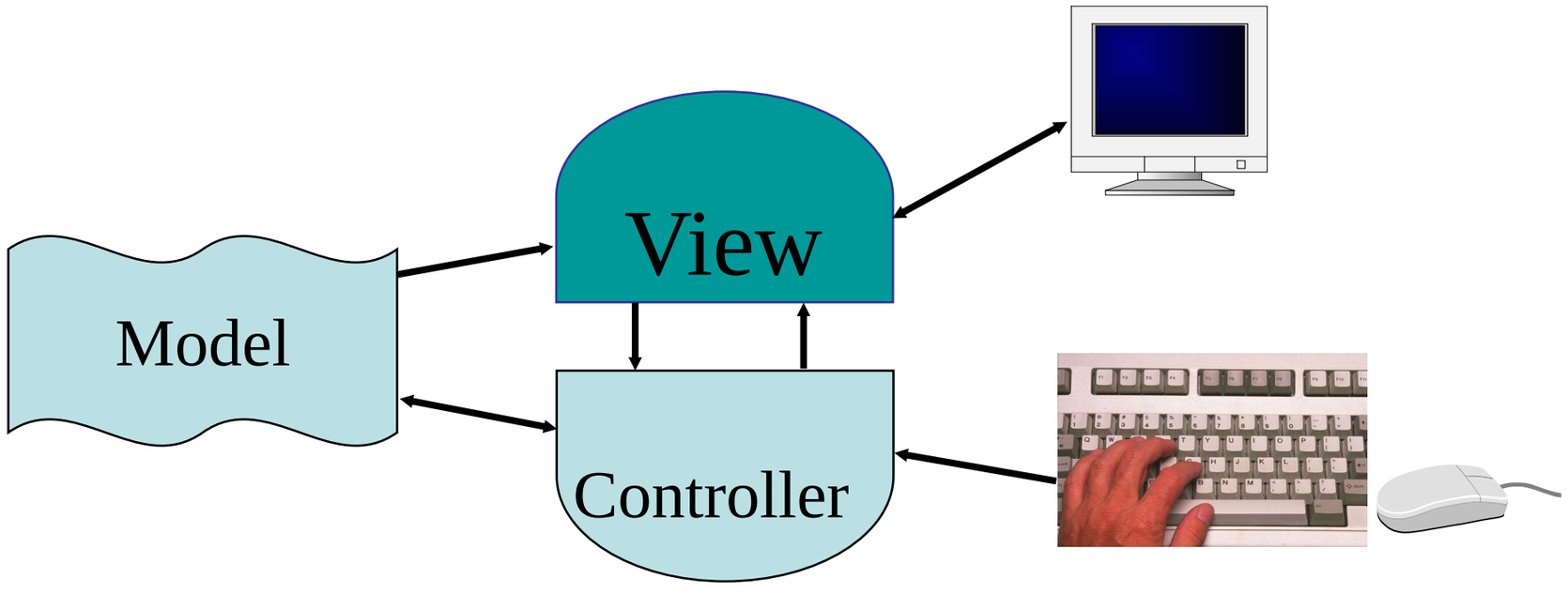
Multiple Views



Blue circles: 4

Cardinal squares: 2

View



Implements a visual display of the model

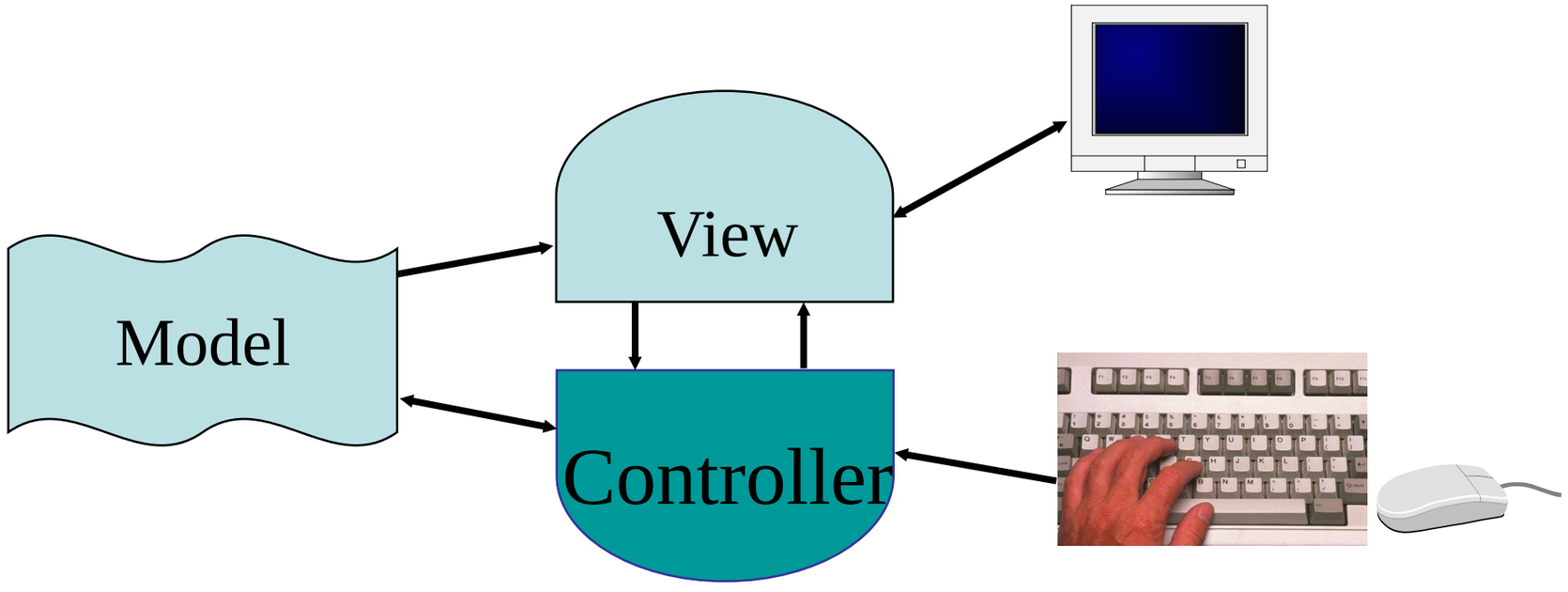
May have multiple views

- e.g., shape view and numerical view

Any time the model is changed, each view must be notified so that it can change *later*

- e.g., adding a new shape

Controller

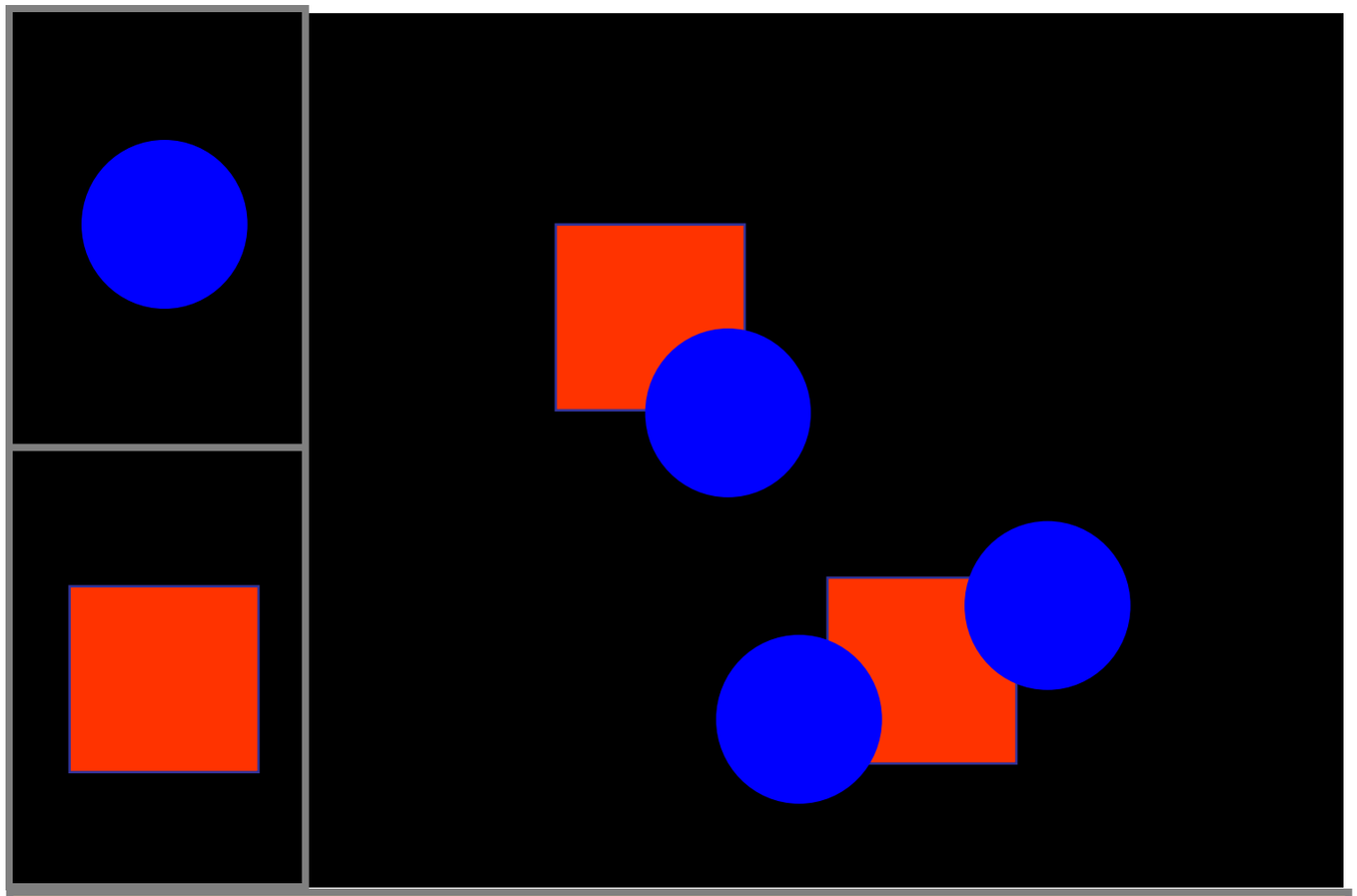


Receives all input events from the user

Decides what they mean and what to do

- communicates with view to determine the objects being manipulated (e.g., selection)
- calls model methods to make changes on objects
  - model makes change and notifies views to update

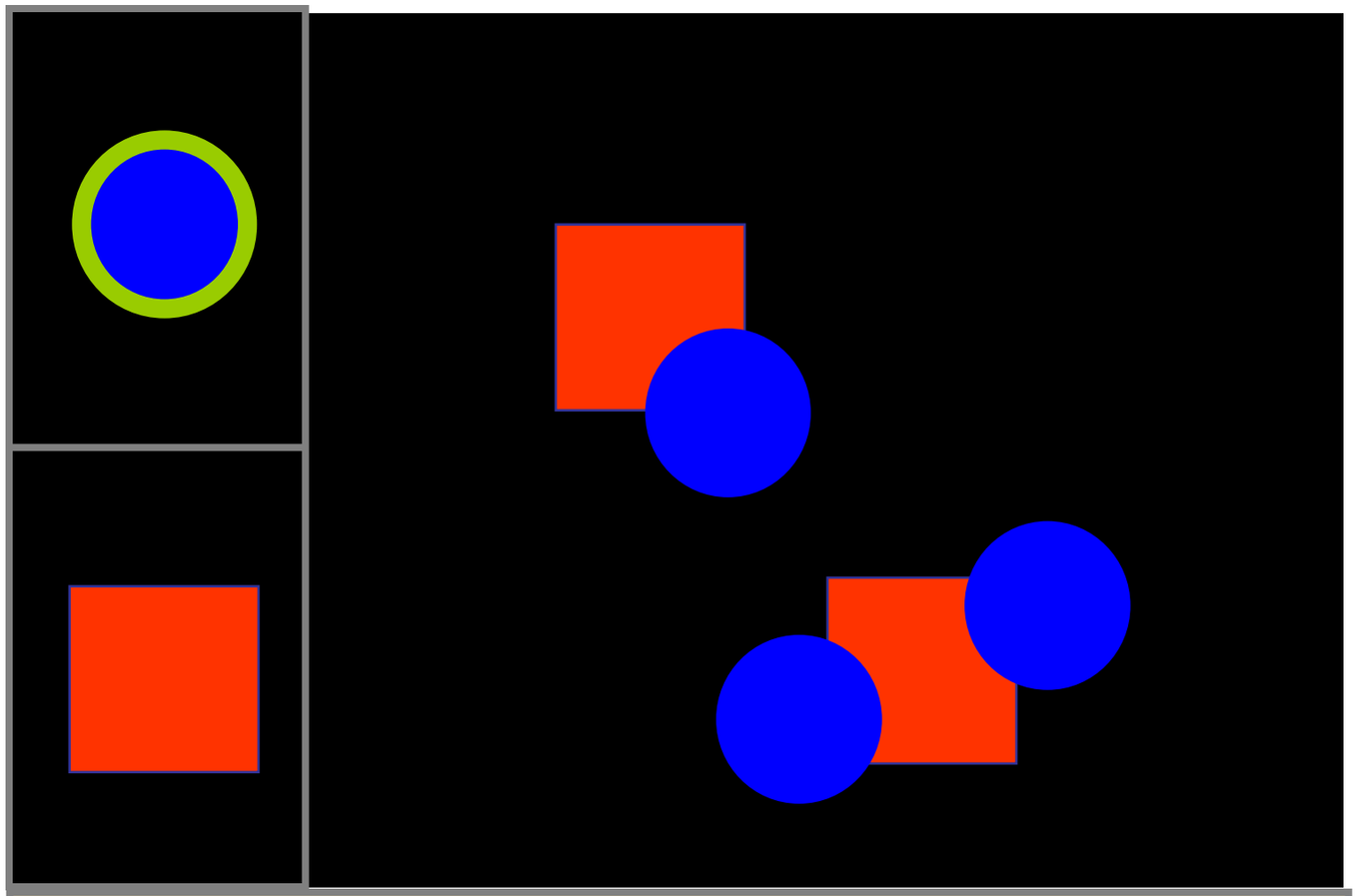
Controller



Blue circles: 3

Cardinal squares: 2

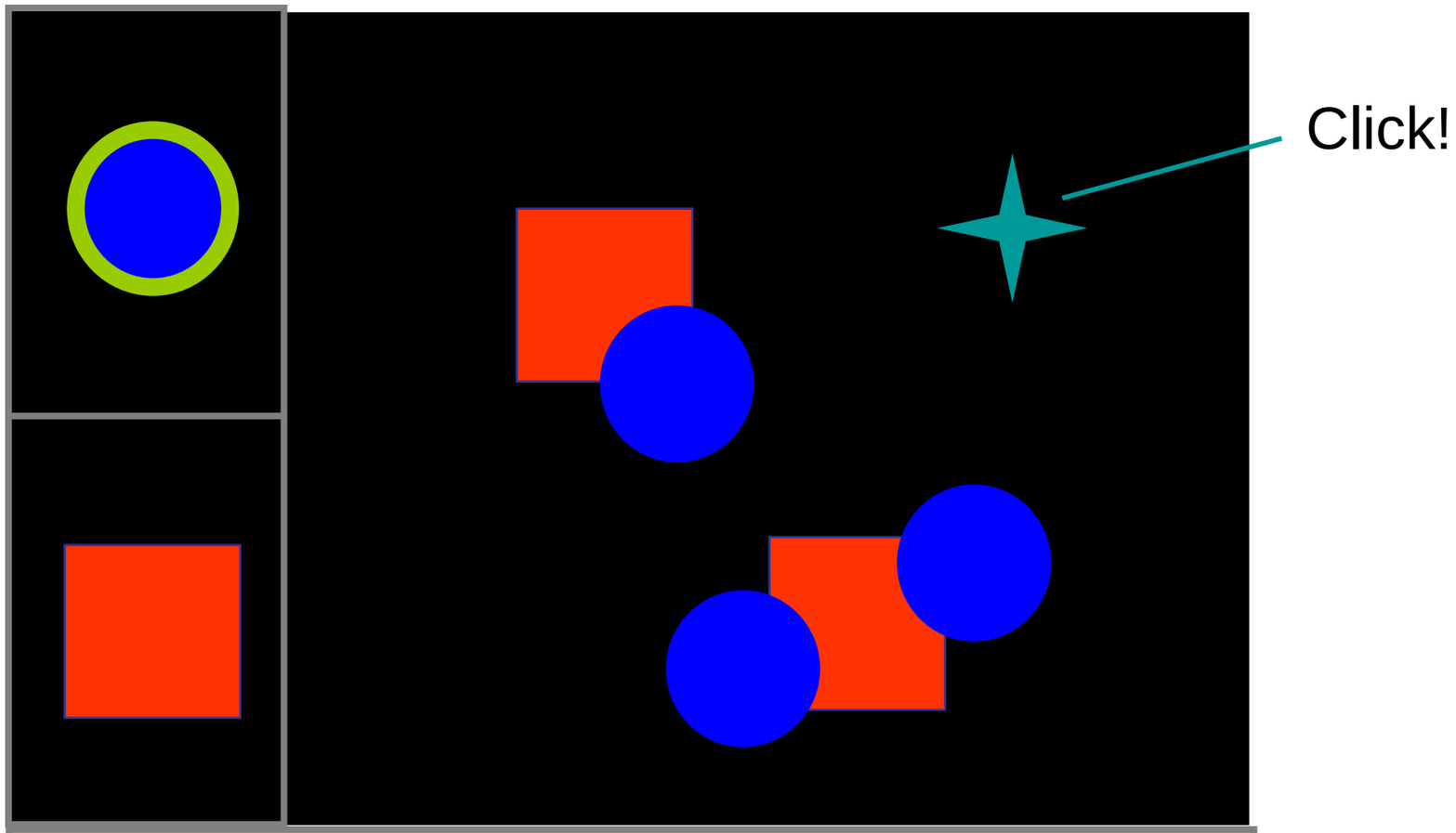
Controller



Blue circles: 3

Cardinal squares: 2

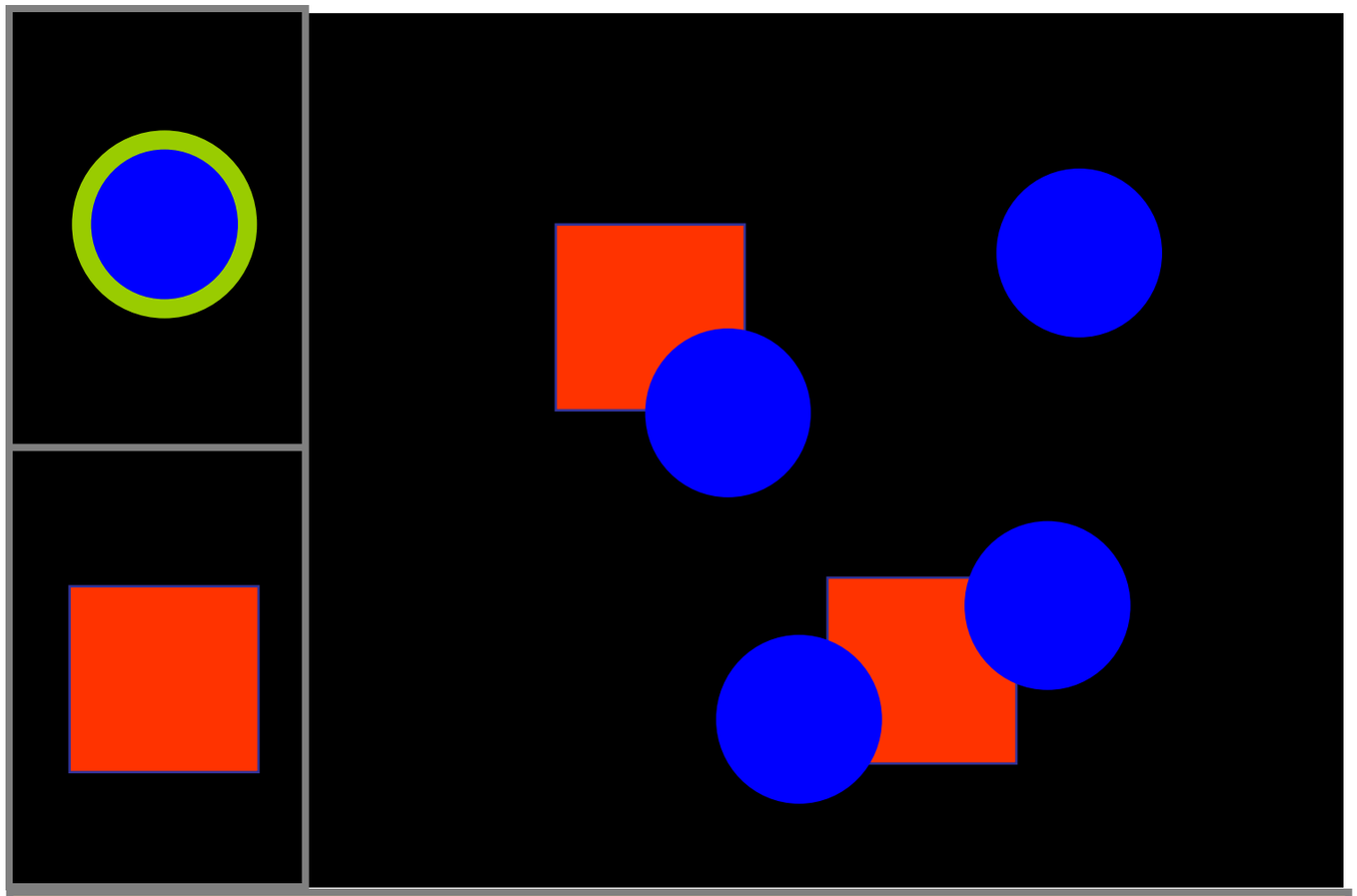
Controller



Blue circles: 3

Cardinal squares: 2

Controller



Blue circles: 4

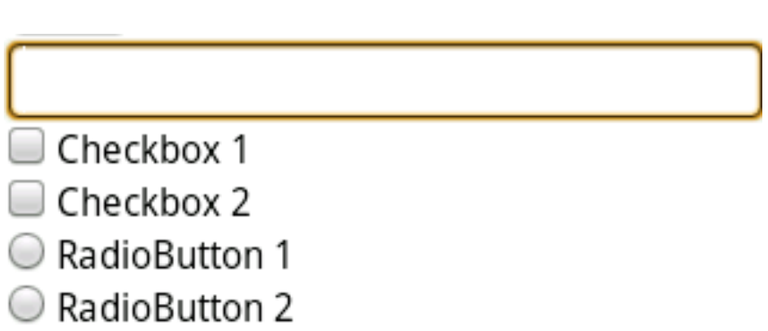
Cardinal squares: 2

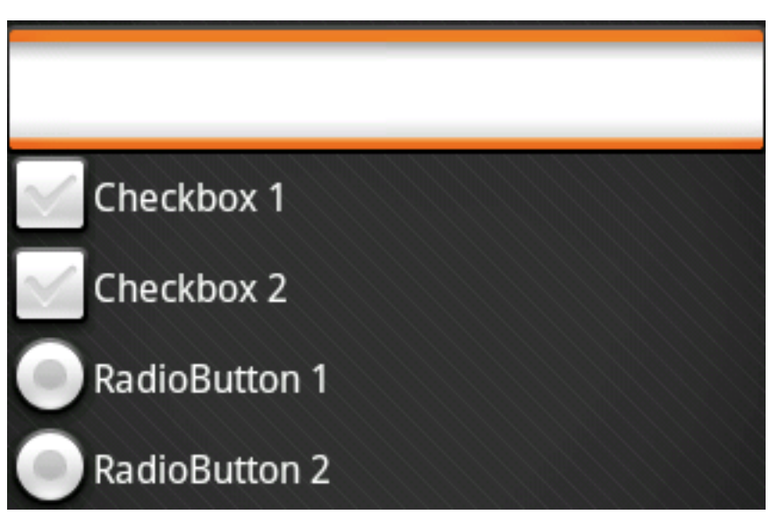
Relationship of View & Controller

"*pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)" –Olsen, Chapter 5.2*

Relationship of View & Controller

"*pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)"*





Controller must contact view to interpret what user events mean (e.g., selection)

Combining View & Controller

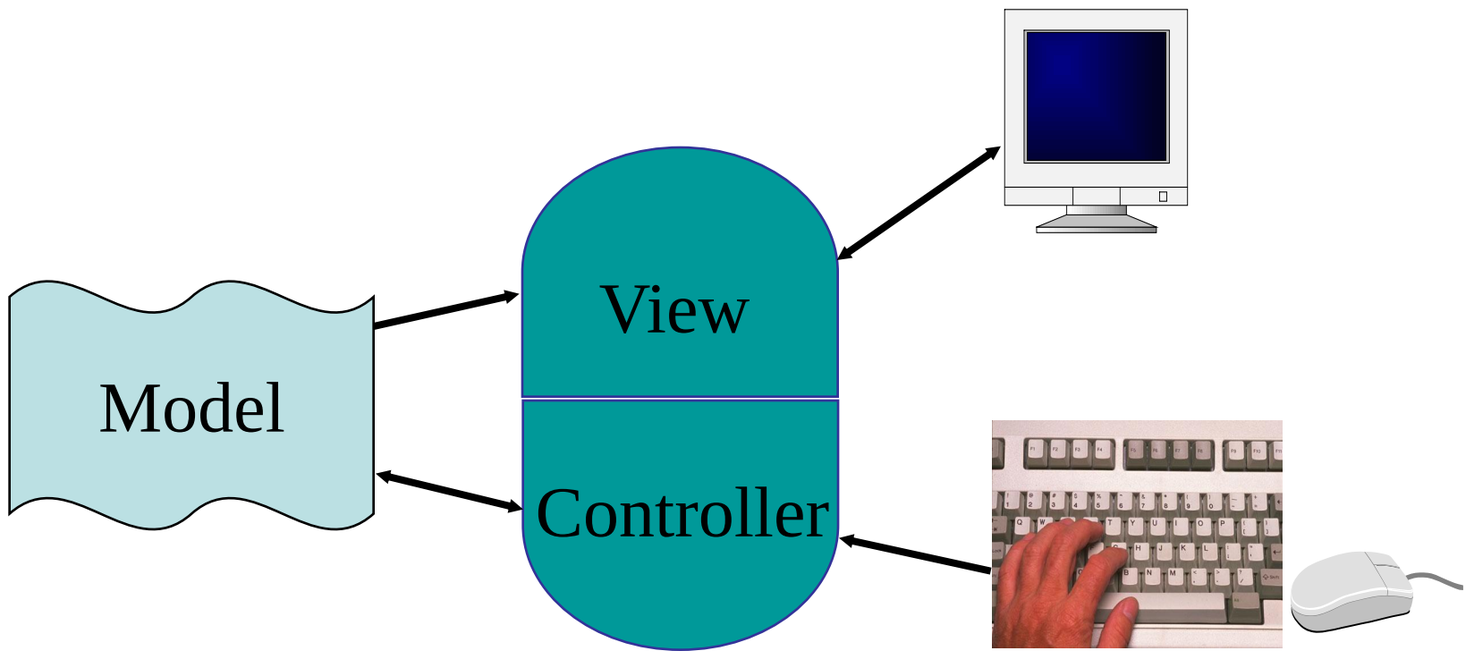
View and controller are tightly intertwined

- lots of communication between the two

Almost always occur in pairs

- i.e., for each view, need a separate controller

Many architectures combine into a single class



Why MVC?

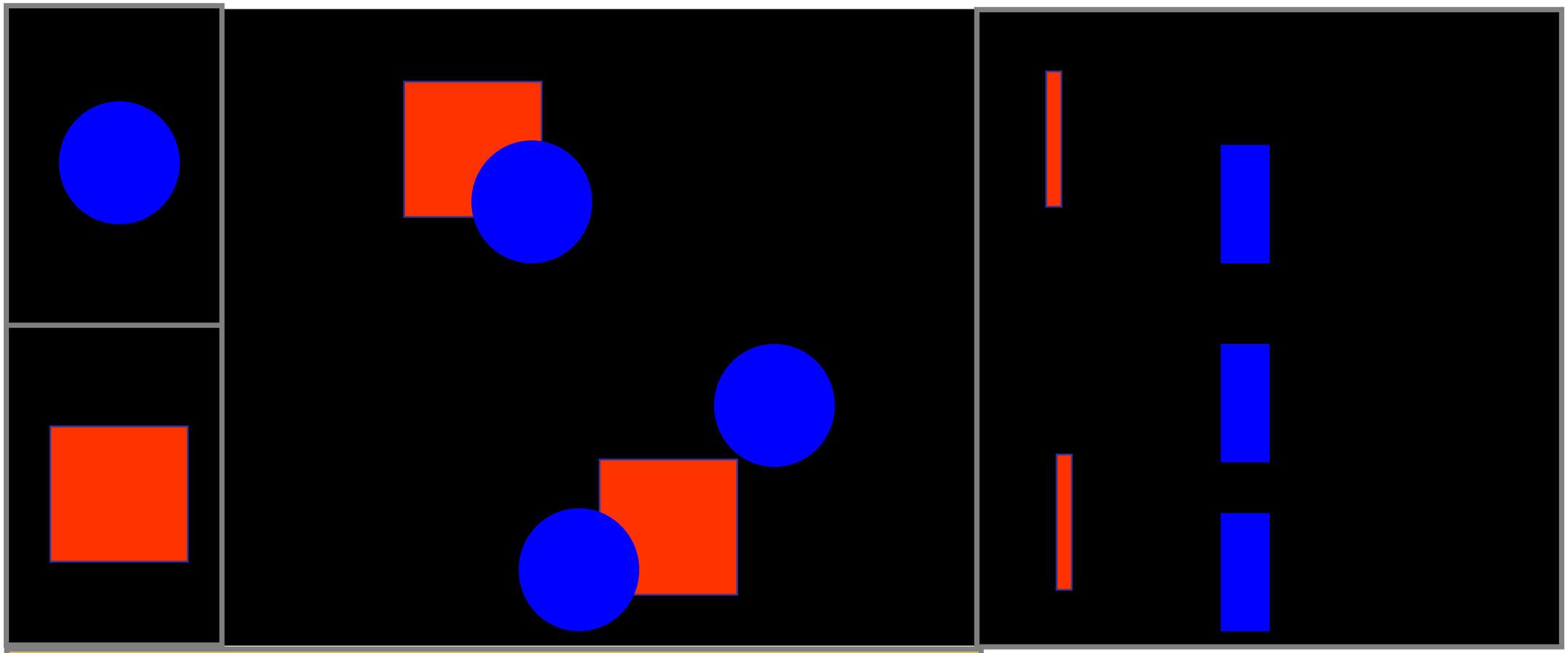
Combining MVC into one class will not scale

- model may have more than one view
  - each is different and needs update when model changes

Separation eases maintenance and extensibility

- easy to add a new view later
- model info can be extended, but old views still work
- can change a view later, e.g., draw shapes in 3-d (recall, view handles selection)
- flexibility of changing input handling when using separate controllers

Adding Views Later



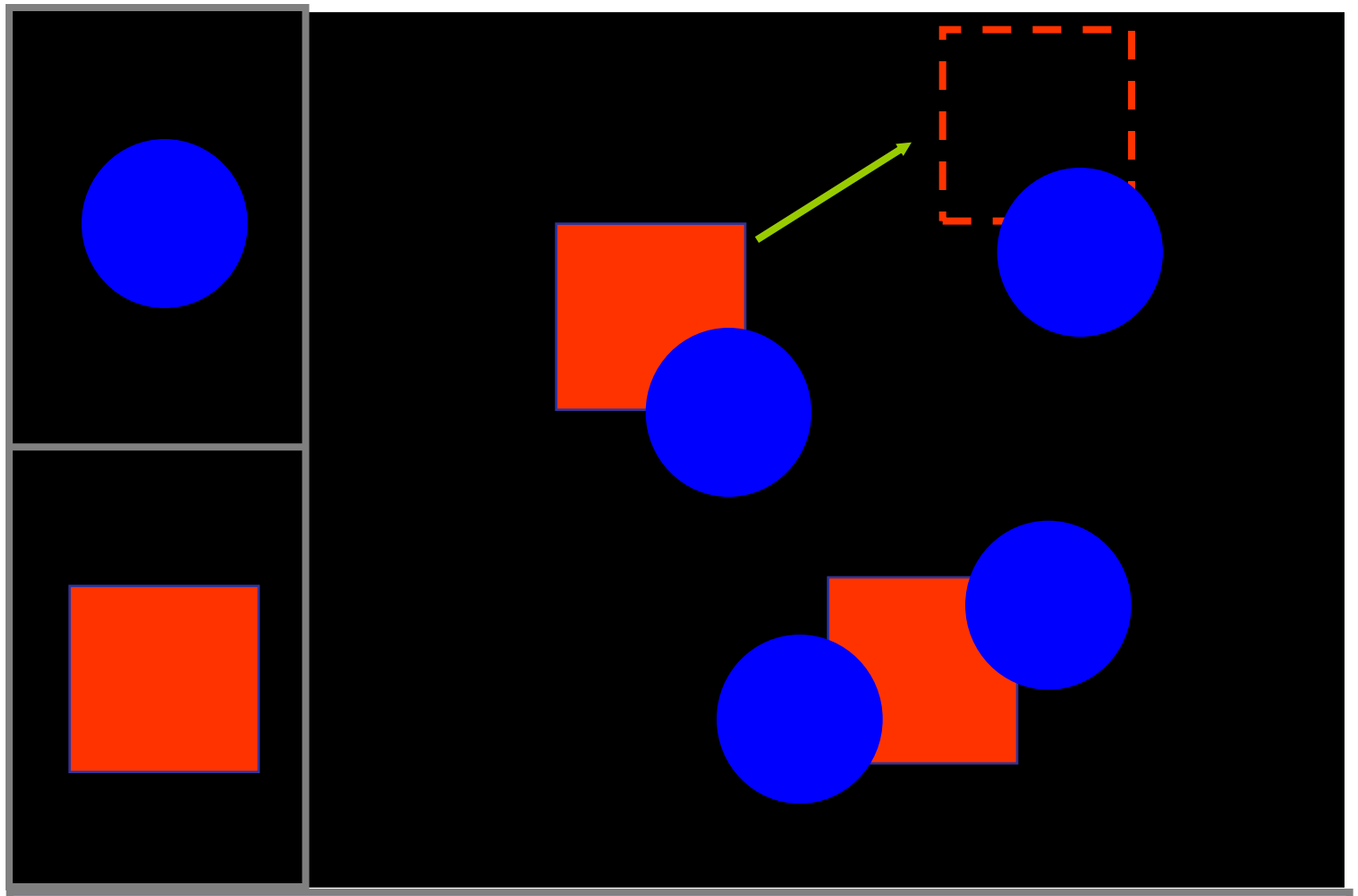
Blue circles: 4

Cardinal squares: 2

Changing the Display

How do we redraw when shape moves?

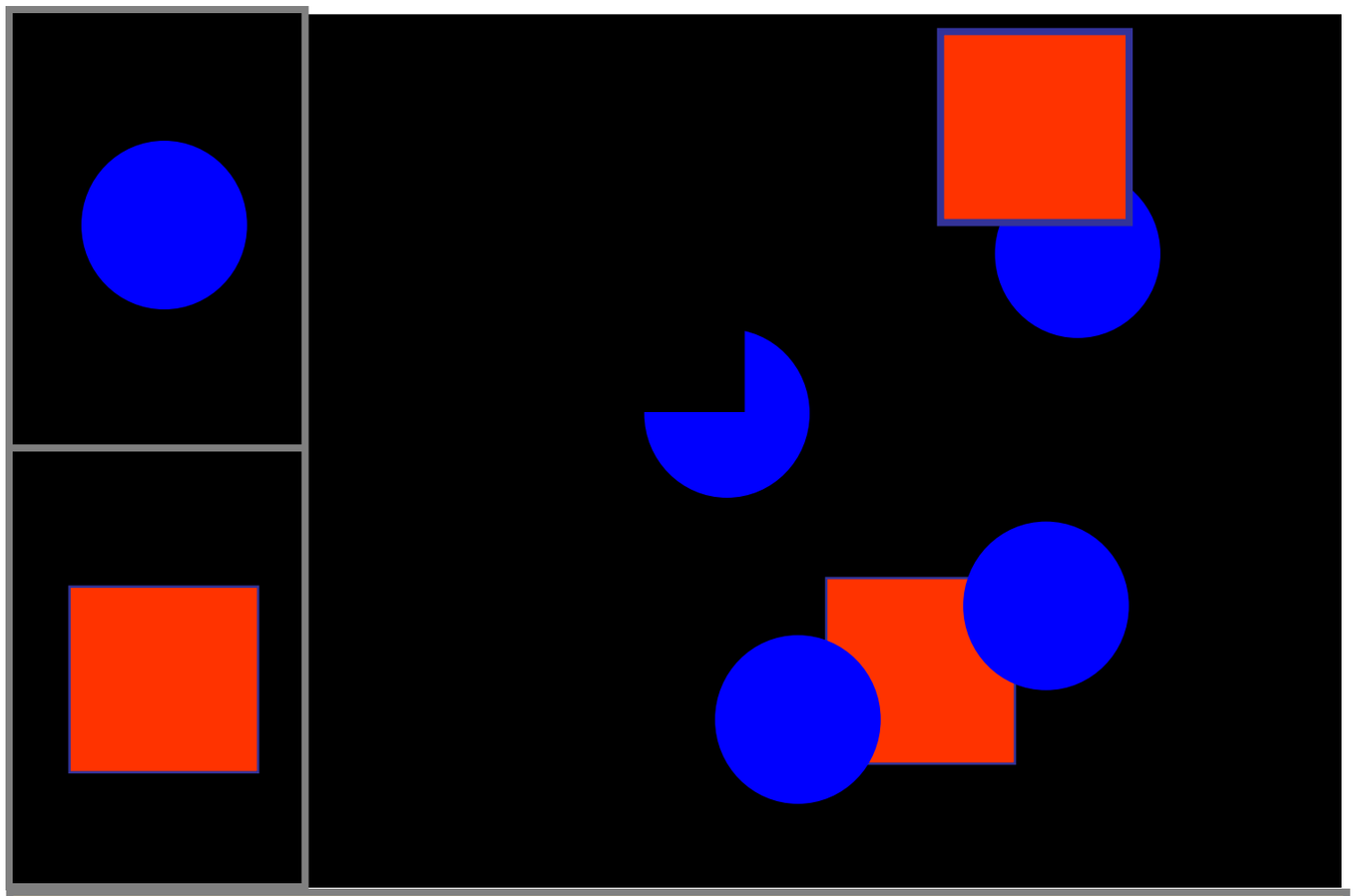
Moving Cardinal Square



Blue circles: 4

Cardinal squares: 2

Erase w/ Background Color and Redraw



Blue circles: 4

Cardinal squares: 2

Changing the Display

Erase and redraw

- using background color to erase fails
- drawing shape in new position loses ordering

Move in model and then redraw view

- change position of shapes in model
- model keeps shapes in a desired order
- tell **all** views to redraw themselves in order
- slow for large / complex drawings
  - flashing! (can solve w/ double buffering)

Damage / Redraw Method

View informs windowing system of areas that need to be updated (i.e., *damaged*)

- does not redraw them at this time…

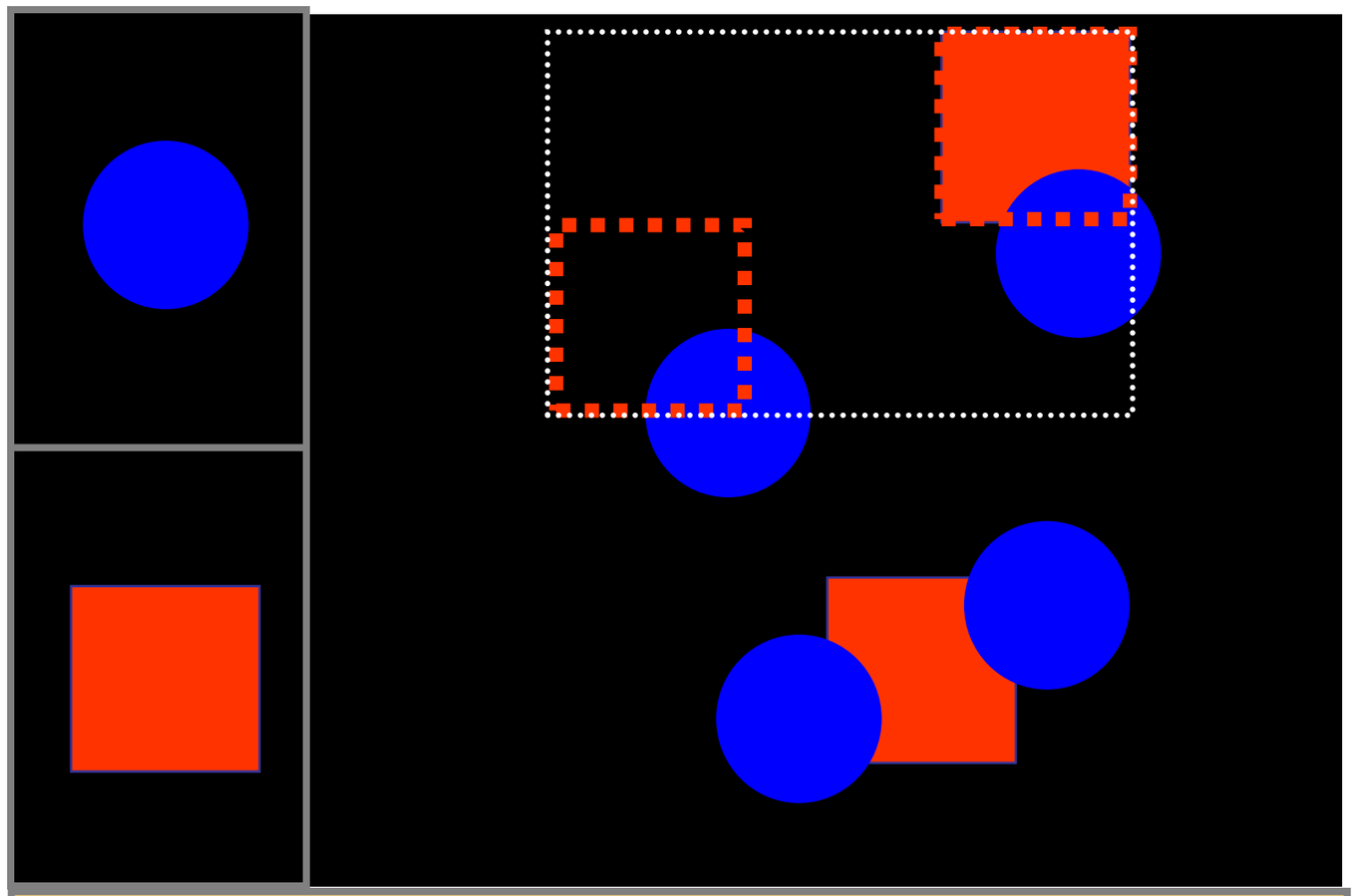
Windowing system

- batches updates
- clips them to *visible* portions of window

Next time waiting for input

- windowing system calls *Repaint* method
  - passes region that needs to be updated

Damage old, Change position in model, Damage new



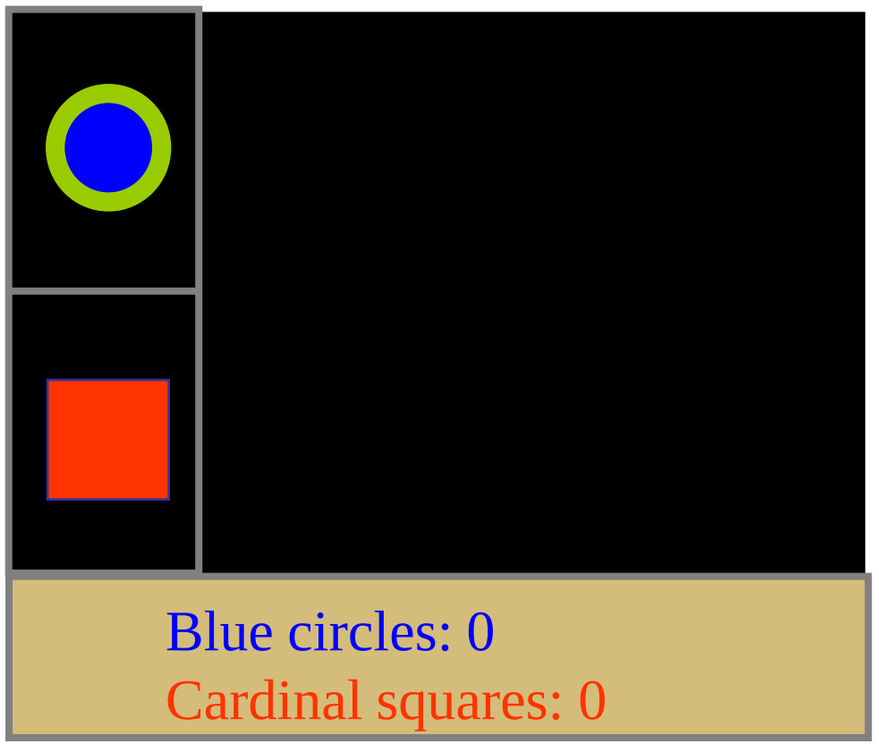
Blue circles: 4

Cardinal squares: 2

Event Flow

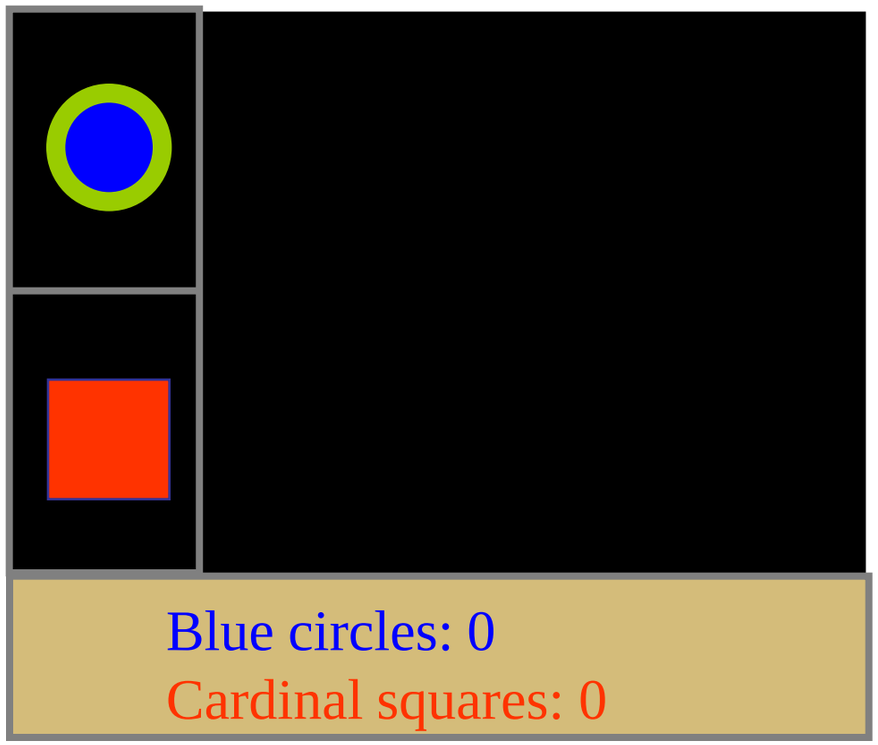
Creating a new shape

Event Flow (cont.)



Assume blue circle selected

Event Flow (cont.)



Press mouse over tentative position

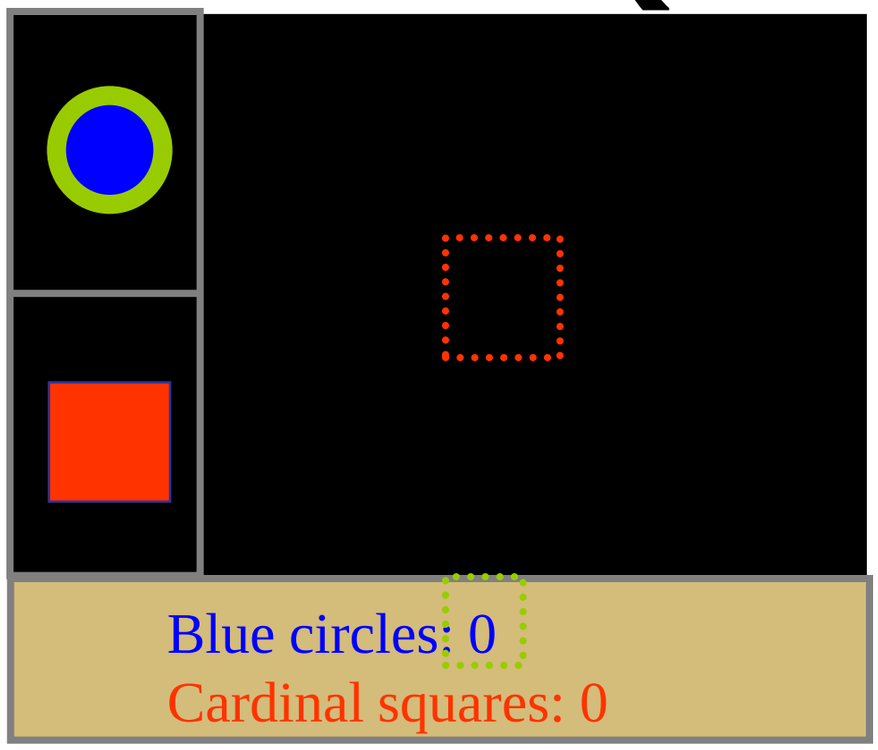
Windowing system identifies proper window for event

Controller for drawing area gets mouse click event

Checks mode and sees “circle”

Calls model’s AddCircle method with new position

Event Flow (cont.)



AddCircle adds new circle to model's list of objects

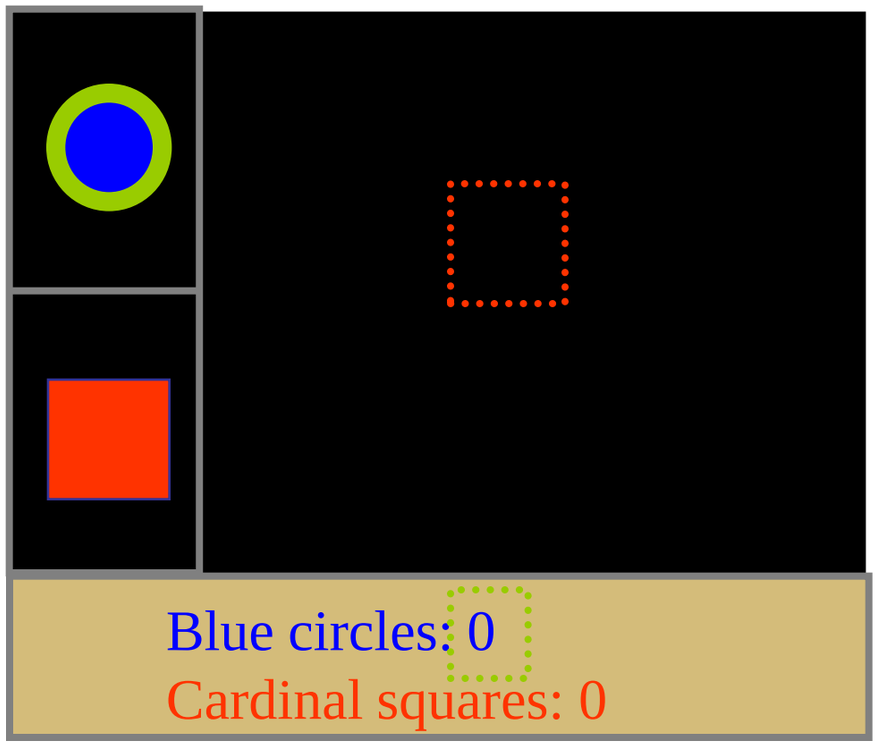
Model then notifies list of views of change

- drawing area view and text summary view

Views notifies windowing system of damage

- both views notify WS without making changes yet!
  - model may override

Event Flow (cont.)



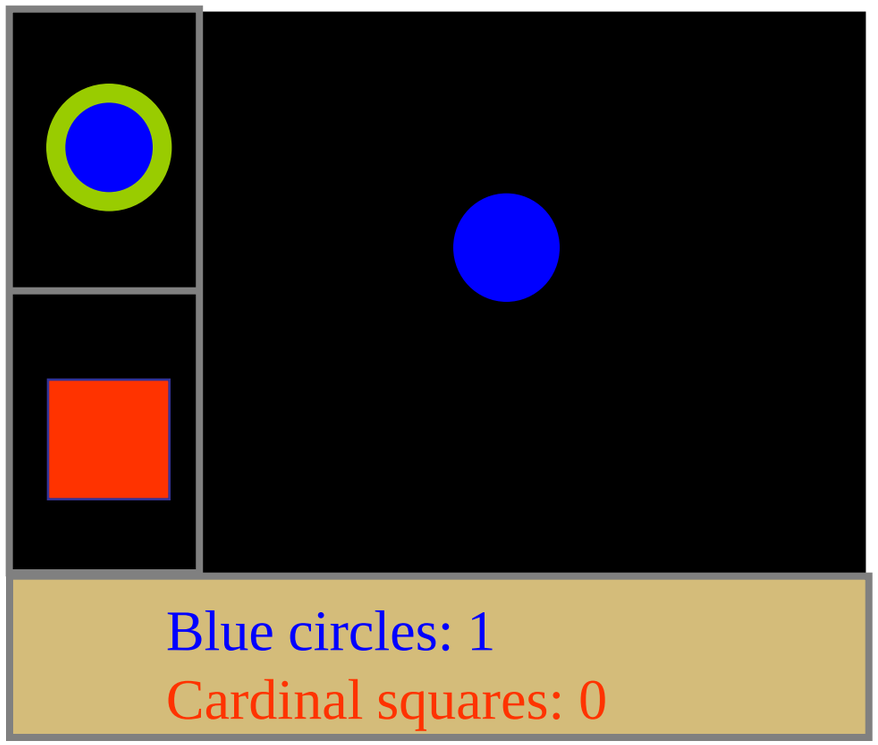
Views return to model, which returns to controller

Controller returns to event handler

Event handler notices damage requests pending and responds

If one of the views was obscured, it would be ignored

Event Flow (cont.)



Event handler calls views' Repaint methods with damaged areas

Views redraw all objects in model that are in damaged area

Dragging at Interactive Speeds

Damage old, move, damage new method may be too slow

- must take less than ~100 ms to be smooth

Solutions

- don't draw object, draw an outline (cartoon)
  - use XOR to erase fast (problems w/ color)
- save portion of frame buffer before dragging
  - draw bitmap rather than redraw the component
- modern hardware often alleviates the problem

Review

2D graphics fundamentals

Event-Driven Interfaces

- Hierarchy of components or widgets
- Input events dispatched to components
- Components process events with callback methods

Model-View-Controller

- Break up a component into
  - **Model** of the data backing the widget(s)
  - **View** determining the look of the widget
  - **Controller** for handling input events
- Provides scalability and extensibility