

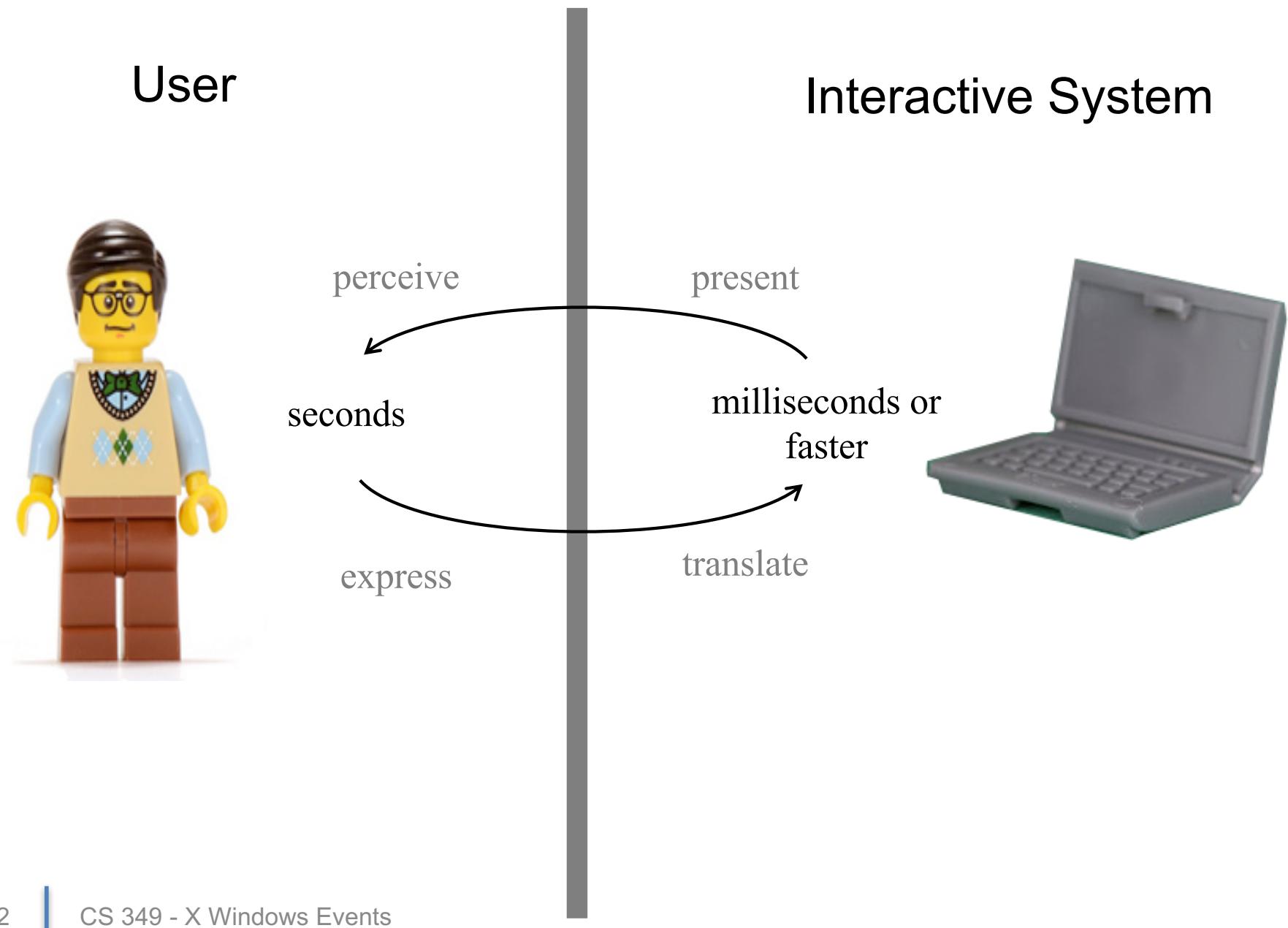
Events Demystified

Events and the Event Loop

Animation

Double Buffering

Human vs. System



Objective

- User interface architectures need to be able to accept input from real-world devices, and map to actions within a system.
 - Support the transformation of input into commands (e.g. “Ctrl” key)
 - We want a general, reusable architecture.



Event-Driven Programming

- “Event-driven programming is a paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.” -- Wikipedia

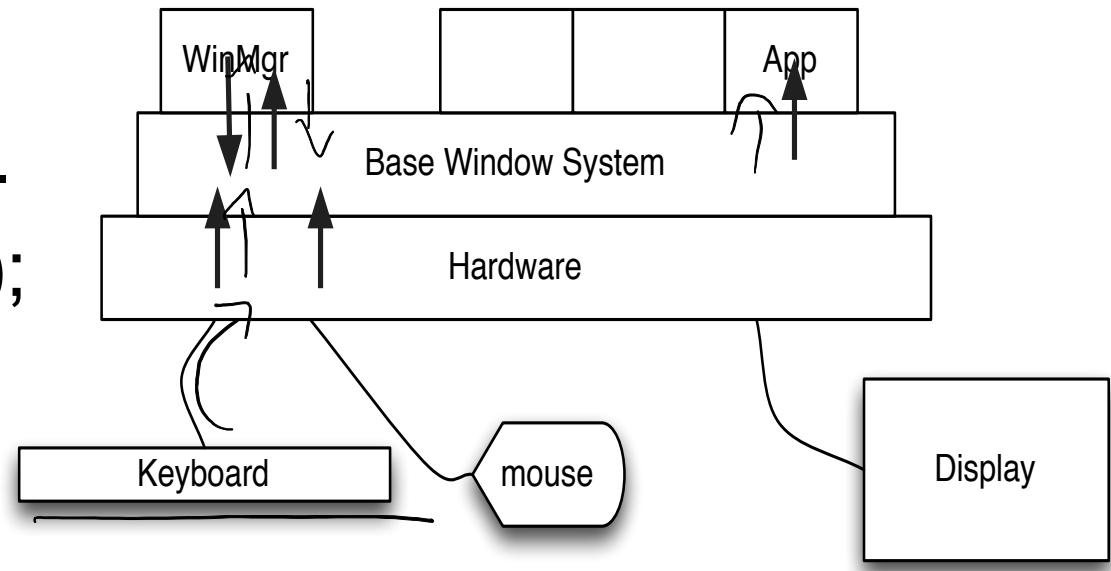
An **event** is a message to notify an application that something happened. Examples include:

- Keyboard (key press, key release)
- Pointer events (button press, button release)
- Input focus (gained, lost)
- Sensor or timer events

Role of the GUI Toolkit, BWS, WM

1. Collect event information
2. Put relevant information in a known structure
3. Order the events by time
4. Decide which application and window should get the event
5. Deliver the event

Some events come from the BWS (user-driven via hardware); some from the window manager.



Role of the Programmer

- Indicate what events you are interested in to BWS/WM, to ensure that events are delivered.
- Write code to:
 - Receive and interpret that event
 - Update program content due to event
 - Redraw display to communicate to user what changed
- In modern languages (Java, C#, Javascript) the process of registering for events and handling events is simplified
 - Java – Listener model
 - C# -- Delegate model
 - Javascript – Looks like a Java/C# hybrid, but is not
 - Apparently simpler, actually worse. See
 - <http://www.quirksmode.org/js/introevents.html>

- In X, the application programmer has to manually handle all event processing
 - Applications get the next event using:
 - `XNextEvent(Display* d, XEvent* e)`
 - Gets & removes the next event in the queue.
 - If empty, it blocks until another event arrives.
 - Can avoid blocking by checking if events available using:
 - `XPending(Display* d)`
 - Query number of events in queue, never blocks.

```
window = XCreateSimpleWindow(...);
XSelectInput(display, window, PointerMotionMask | KeyPressMask);
XMapRaised(display, window);
XFlush(display);

XEvent event; // save the event here
while( true ) { // loop until 'exit' event
    XNextEvent( display, &event ); // wait for next event
    switch( event.type ) {
        case MotionNotify: // mouse movement
            cout << event.xmotion.x << "," << event.xmotion.y << endl;
            break;
        case KeyPress: // any keypress
            exit(0);
            break;
        // Handle other kinds of events
    }
    repaint( ... ); // call my repaint function
}
XCloseDisplay(display);
```

blocks

bad

Selecting Events to “listen to”

- Don’t always need all of the events. (Why?)

```
// Tell the BWS which input events you want.  
XSelectInput( xinfo.display, xinfo.window,  
    ButtonPressMask | KeyPressMask |  
    ExposureMask | ButtonMotionMask );
```

- Defined masks: NoEventMask, KeyPressMask, KeyReleaseMask, ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, ..., ButtonMotionMask, KeymapStateMask, ExposureMask, VisibilityChangeMask, ...
- See
 - <http://www.tronche.com/gui/x/xlib/events/types.html>
 - <http://www.tronche.com/gui/x/xlib/events/mask.html>

Event Structure: Union

- X uses a C union

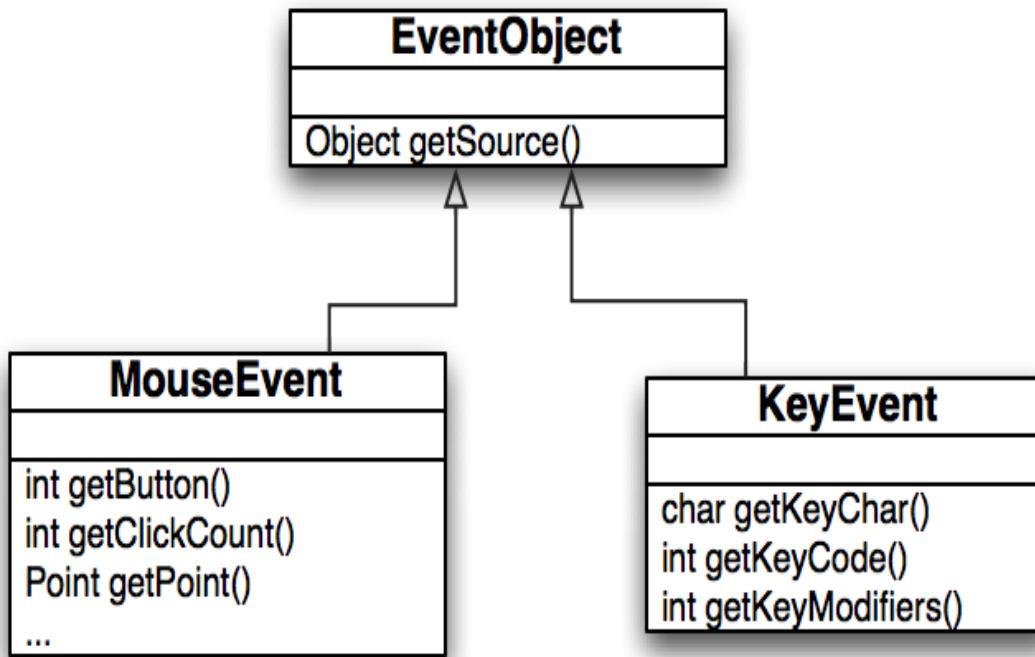
```
typedef union {  
    int type;  
    XKeyEvent xkey;  
    XButtonEvent xbutton;  
    XMotionEvent xmotion;  
    // etc. ...  
}
```

- Each structure contains at least the following

```
typedef struct {  
    int type;  
    unsigned long serial; // sequential #  
    Display* display; // display event was read from  
    Window window; // window which event is relative to  
} X__Event
```

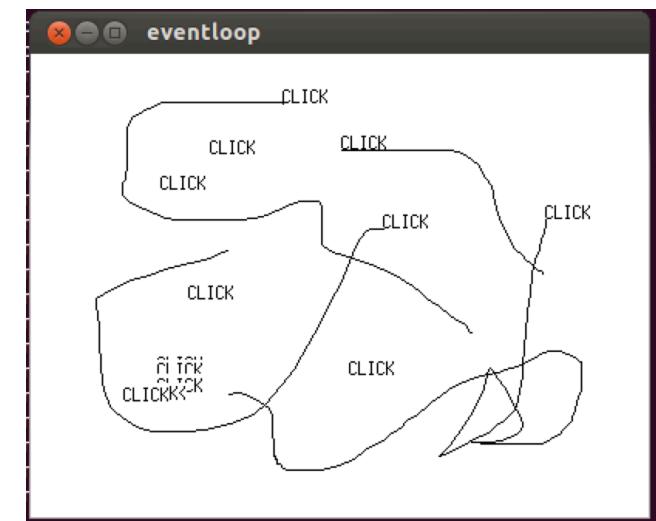
Java Event Structure: Inheritance

- Java (and C# and C++.NET and VB and Objective C and Swift and ...) uses an inheritance hierarchy
- Each subclass contains additional information, as required (not shown)



Code Review: eventloop.cpp

- XSelectInput: defines events to process
- XNextEvent: used in event loop to get the next event to process
- Events
 - KeyPress and XLookupString
 - character vs. scan codes
 - Expose and resizing the window
 - MotionNotify on mouse move



Java Event Handling

- Java simplifies much of this
 - Events are transmitted to onscreen objects without programmer intervention
 - The event loop is hidden and managed by the JVM (i.e. you don't need to explicitly manage it)
 - To handle events, tell onscreen objects what you're interested in and what to do

Opening a Window in Java

```
import javax.swing.*;  
  
public class TestWindow extends JFrame{  
  
    public TestWindow(){  
        this.setTitle("My Window");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
  
    public static void main (String args[]){  
        TestWindow myWindow = new TestWindow();  
        myWindow.setSize(400, 300);  
        myWindow.setVisible(true);  
    }  
}
```

X Windows: Events + Drawing

Animation

Double Buffering

- A simulation of movement created by displaying a series of pictures, or frames.
 - Animation: *drawing frames at a consistent rate.*
- Goals:
 - Move things around on the screen
 - Repaint 24 - 60 times per second (frames-per-second, frame rate, or “FPS”)
 - Make sure events are handled on timely basis
 - Don’t use more CPU than necessary
 - Easily understood code



Respond to Events (non-blocking)

```
while( true ) {
    if (XPending(display) > 0) {      // pending events?
        XNextEvent(display, &event ); // yes, process them
        switch( event.type ) {
            // handle event cases here ...
        }
    }
    handleAnimation(xinfo);          // update animation objects
    repaint(xinfo);                // repaint the frame
}
```

This doesn't block, but it doesn't work very well either...

Key to animation: managing time

```
while( true ) {
    if (XPending(display) > 0) {      // pending events?
        XNextEvent(display, &event ); // yes, process them
        switch( event.type ) {
            // handle event cases here ...
        }
    }
    handleAnimation(xinfo);          // update animation objects
    repaint(xinfo);                // repaint the frame

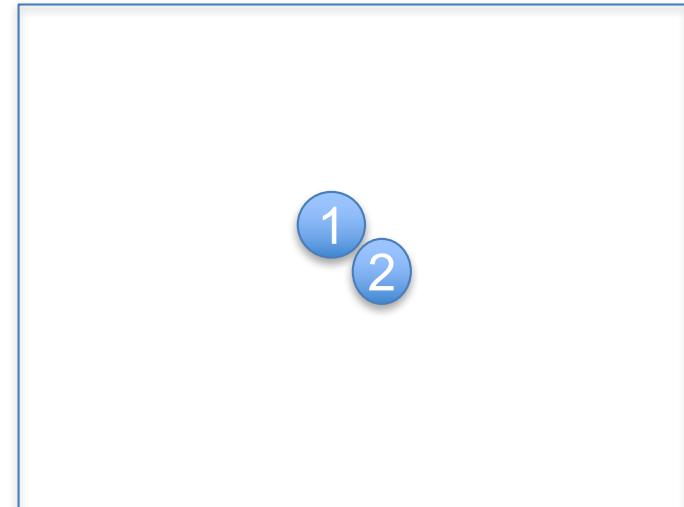
    // sleep for about 1/30 second
    const int FPS = 30;
    usleep(1000000/FPS);
}
```

← added

Much better, but still
not great... can
overwhelm CPU.

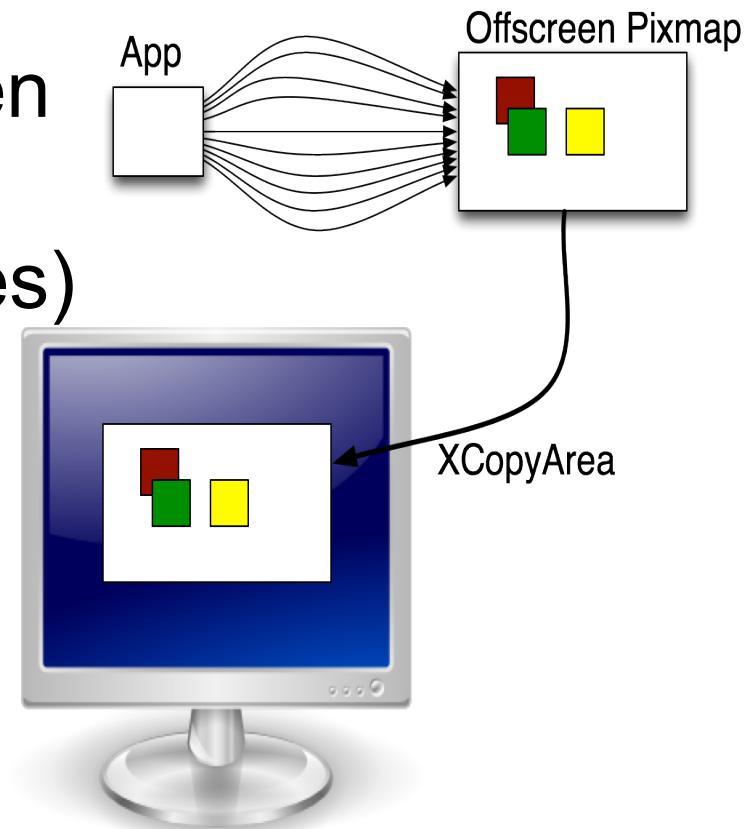
Screen Flicker During Drawing

- Imagine you are animating two balls on the display in a game of breakout
- As the balls get close to each other, you notice that one seems to flicker
 - Why?
- If you paint the balls sequentially, then you might clear a rectangle of the first ball's position, paint the first ball in the new position, then clear rect for the second ball and paint the second ball
- Ball 1 has a bite out of it
 - Or may even vanish ...
 - Until next update



Fix: Double Buffering

- Flickering: when an intermediate image is on the display
 - e.g.: Clear, then redraw strategies
- Solution:
 - Create an off screen image buffer
 - Draw to the buffer
 - Copy the buffer to the screen as quickly as possible (hopefully between refreshes)
- In X manual
- In ... MOST ... other situations, it is done for you



Double Buffering

```
// create off screen buffer
xinfo.pixmap = XCreatePixmap(xinfo.display,
xinfo.window,
    width, height, depth); // size and *depth* of pixmap

// draw into the buffer
// note that a window and a pixmap are “drawables”
XFillRectangle(xinfo.display, xinfo.pixmap,
    xinfo.gc[0], 0, 0, width, height);

// copy buffer to window
XCopyArea(xinfo.display, xinfo.pixmap, xinfo.window,
xinfo.gc[0],
    0, 0, width, height, // pixmap region to copy
    0, 0); // top left corner of pixmap in window

XFlush( xinfo.display );
```

Java Code: set Double Buffering

- In java swing, double buffering is built into the JComponent class (which is a base class for all Swing components, so they inherit this functionality)
- Two methods:
 - `public boolean isDoubleBuffered();`
 - `public void setDoubleBuffered(boolean o);`
- Double-buffering enabled by default !
 - If set in top-level container, all subcomponents will be double buffered regardless of individual settings
 - Why?

Understanding Modern GUI Toolkits



Widgets

- GUI Toolkits contain sets of widgets that you can use to build applications, and enable event delivery to widgets
- So ... what is a widget?
- A widget (also graphical control element or control) is an element of interaction in a graphical user interface (GUI), such as a button or a scroll bar. Controls are software components that a computer user interacts with through direct manipulation to read or edit information about an application. User interface libraries contain a collection of widgets and the logic to render these.
 - Wikipedia (AKA the font of all knowledge)

Nesting Windows: Child vs. Sibling Window

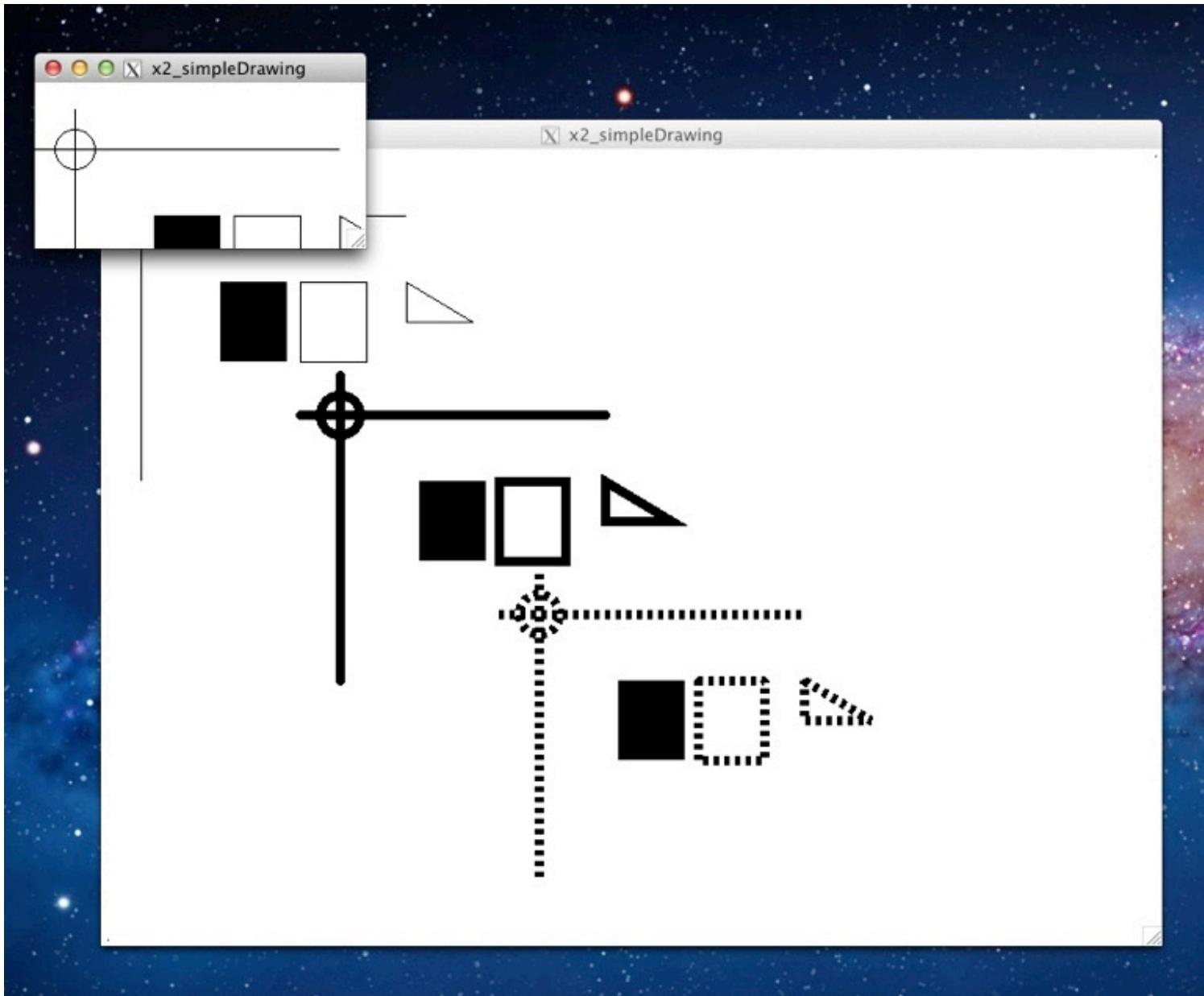
```
xInfo1.display = display;
xInfo1.screen = screen;
initX(argc, argv, xInfo1, DefaultRootWindow( display ),
      100, 100, 800, 600);

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

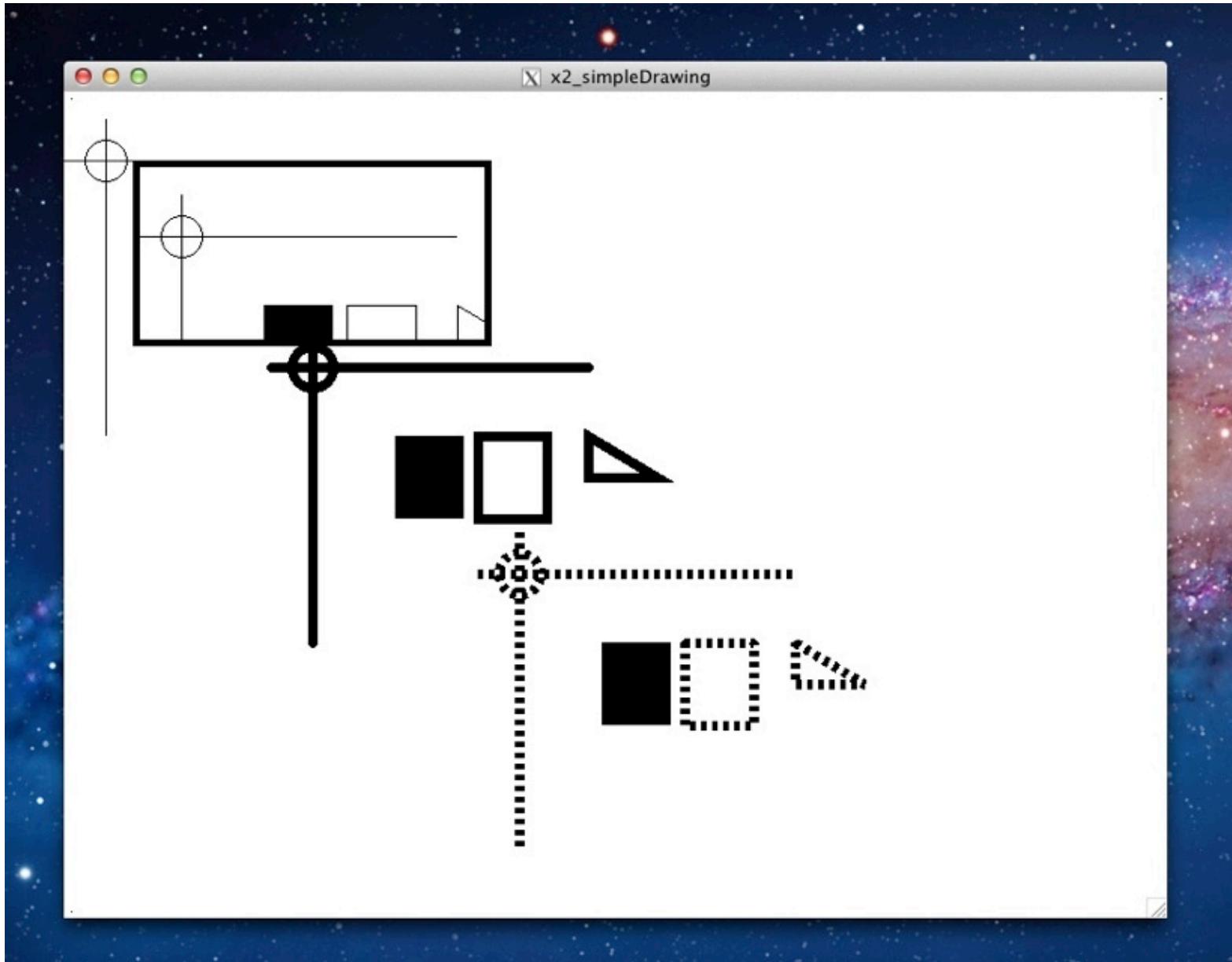
```
xInfo1.display = display;
xInfo1.screen = screen;
initX(argc, argv, xInfo1, DefaultRootWindow( display ),
      100, 100, 800, 600);

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

multiwindow.cpp: Sibling Window

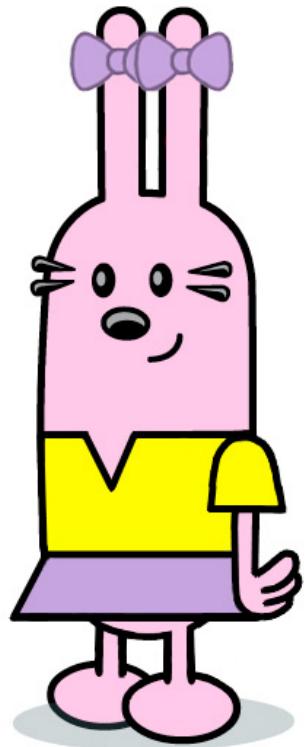


multiwindow.cpp: Child Window



Widget

- Child windows are essentially widgets
 - Widget is a generic name for parts of an interface that have their own behavior: buttons, progress bars, sliders, drop-down menus, spinners, file dialog boxes, ...
 - widgets also called “components”, “controls”
 - Can have their own appearance
 - Receive and interpret their own events
 - Put into libraries (toolkits) for reuse



Widget from *Wow Wow Wubbzy*

- Each operating system implements a set of native widgets
 - What you would expect: buttons, checkboxes, combo boxes, etc., etc.
 - But each OS's native widget set had idiosyncrasies
- Java's cross-platform goal required a decision:
 - AWT, lowest common denominator
 - Swing, roll your own for the JVM.
 - SWT, platform specific
- Pluses and minuses to each approach.
 - What are they?

PopQuiz

- Why Java?
- Why did we talk about XWindows at all?

- Events (structure, selecting)
- Event loops (timing)
- Animation
- Double Buffering