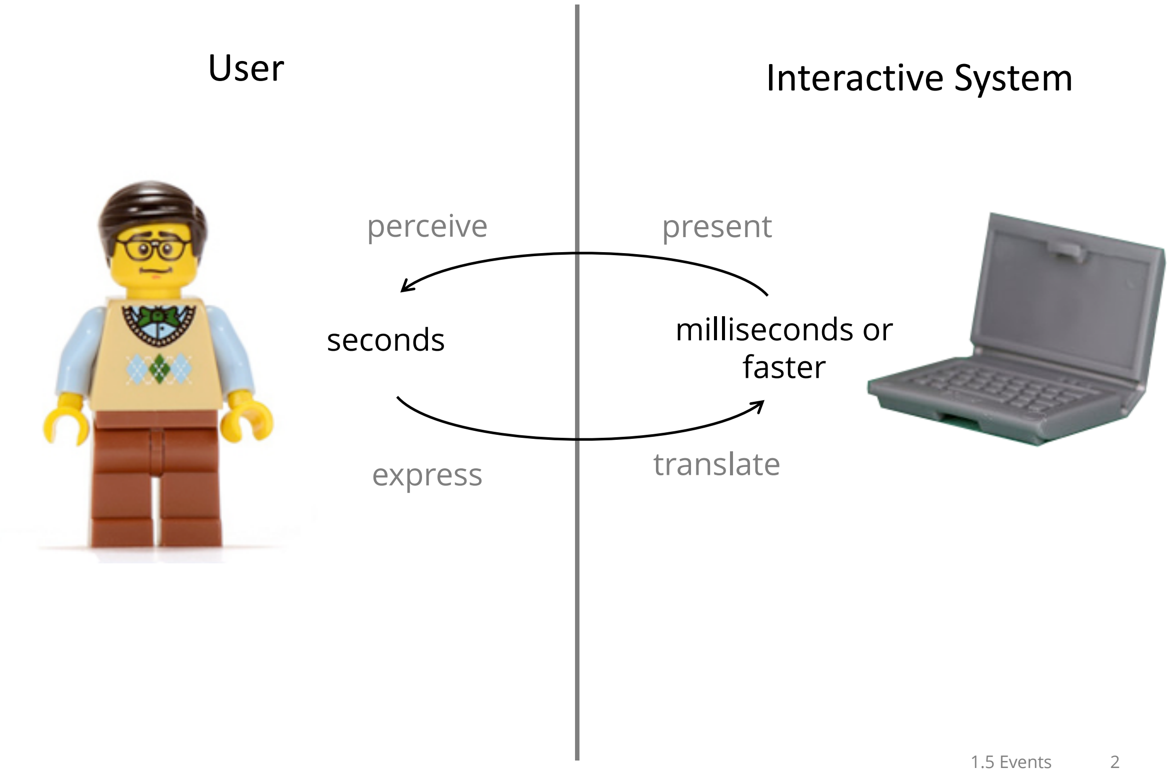
Events

Events and the Event Loop

Animation

Double Buffering

Human vs. System



Event Driven Programming

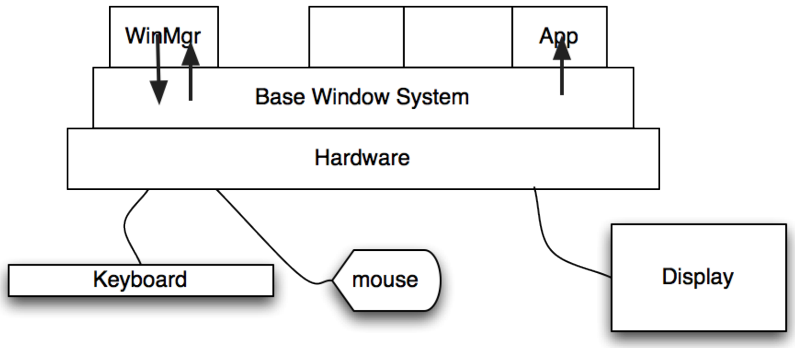
- Nothing happens unless the something else happens:
  - user presses a key, moves the mouse, …
  - window is resized, closed, covered …
  - certain time passes
  - (file changes, network connection, database updates, order arrives, sensor is triggered, …)
- Write code to:
  1. Register to receive events
  2. Receive and interpret those types of events
  3. Update program content based on event
  4. Redraw the display (provide feedback) to communicate to user what changed

Events Defined

- English:
  - An observable occurrence, often extraordinary occurrence
- User Interface Architecture:
  - A message to notify an application that something happened
- Examples:
  - Keyboard (key press, key release)
  - Pointer Events (button press, button release, motion)
  - Window crossing (mouse enters, leaves)
  - Input focus (gained, lost)
  - Window events (exposure, destroy, minimize)
  - Timer events

Role of the Base Window System

- Collect event information
- Put relevant information in a known structure
- Order the events by time
- Decide which application/window should get event
- Deliver the event
- Some events come from the user via the underlying hardware; some from the window manager.



Receiving Events

- In X Windows, applications get the next event using:

```
XNextEvent(Display* display, XEvent* evt)
```

  – Gets and 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* display)
```

  – Query number of events in queue, never blocks

Simple Event Loop: eventloop.min.cpp

```
// select events
XSelectInput(dis, win, PointerMotionMask | KeyPressMask);
...
XEvent event; // save the event here

while( true ) {  // event loop

   // wait for next event
   XNextEvent( display, &event );

   switch( event.type ) {

     case MotionNotify: // mouse movement event
        // handle here …
        break;

     case KeyPress:  // key press event
        // handle here …
        exit(0); // need a way to exit infinite event loop …
        break;
   }
}
```

Selecting Input Events to “listen to”

```
// Tell the window manager what input events you want.
XSelectInput( display, 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 #
    Bool send_end;    // from SendEvent request?
    Display* display; // display event was read from
    Window window; // window which event is relative to
} X___Event
```

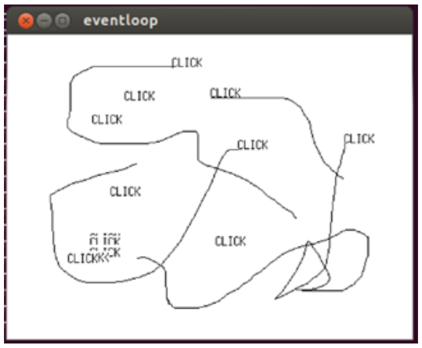
Responding to Events (blocking)

```
while( true ) {

     XNextEvent(display, &event); // wait for next event

     switch(event.type) {
     case Expose:
        // ... handle expose event ...
        cout << event.xexpose.count << endl;
         break;
     case ButtonPress:
        // ... handle button press event ...
        cout << event.xbutton.x << endl;
         break;
     case MotionNotify:
        // ... handle event ...
         cout << event.xmotion.x << endl;
         break;
     }
     repaint( ... ); // call my repaint function
}
```

Code Review: eventloop.cpp

- XSelectInput
- XNextEvent
- event loop
- Notes:

  `KeyPress and XLookupString`
  - character vs. scan codes
  - Uses Displayables

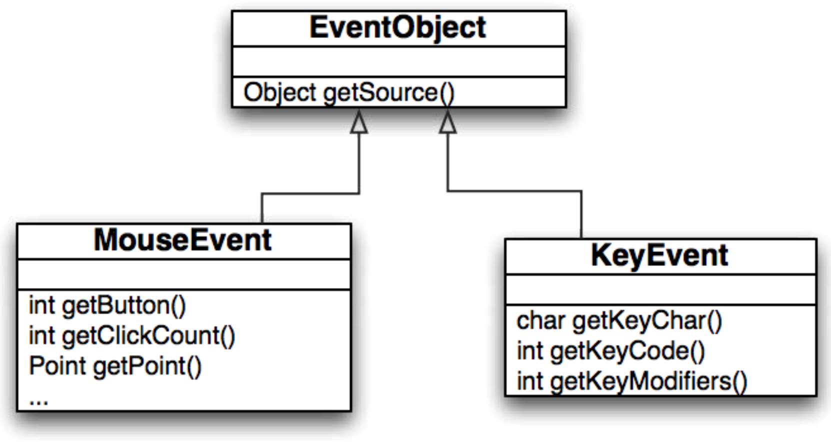


Events in Modern Languages

- The process of registering for, and handling events is simplified
- Examples:
  - Java: listener model
  - C#: delegate model
  - Javascript: (looks like Java/C# hybrid, but is not)
    http://www.quirksmode.org/js/introevents.html

Java Event Structure: Inheritance

- Java uses an inheritance hierarchy
- Each subclass contains additional information, as required



Animation

- A simulation of movement created by displaying a series of pictures, or frames.
- 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 a timely basis
  - Don’t use more CPU than necessary

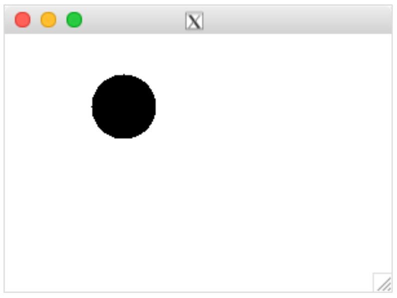
Animation Timing and Responding to Events (non-blocking)

```
while( true ) {

  if (XPending(display) > 0) { // any events pending?
     XNextEvent(display, &event ); // yes, process them
     switch( event.type ) {
       // handle event cases here ...
     }
  }

  // now() is a helper function I made
  unsigned long end = now(); // time in microseconds

  if (end - lastRepaint > 1000000/FPS) { // repaint at FPS
     handleAnimation(xinfo); // update animation objects
     repaint(xinfo); // my repaint
     lastRepaint = now(); // remember when the paint happened
  }

  // IMPORTANT: sleep for a bit to let other processes work
  if (XPending(xinfo.display) == 0) {
     usleep(1000000 / FPS - (end - lastRepaint));
  }
}
```

get current time in microseconds: now()

```
#include <sys/time.h>

// get microseconds
unsigned long now() {
       timeval tv;
       gettimeofday(&tv, NULL);
       return tv.tv_sec * 1000000 + tv.tv_usec;
}
```

Code Review: animation.min.cpp



- Highlights:

```
XClearWindow(display, window);
ballPos.x += ballDir.x;
```

- Experiments to try:
  1. Resize the window.
  2. Comment out this:

     ```
     XClearWindow(display, window);
     ```

  3. Comment out this (and closing bracket):

     ```
     if (XPending(display) > 0) {
        XNextEvent( display, &event );
     ```

     and try clicking mouse

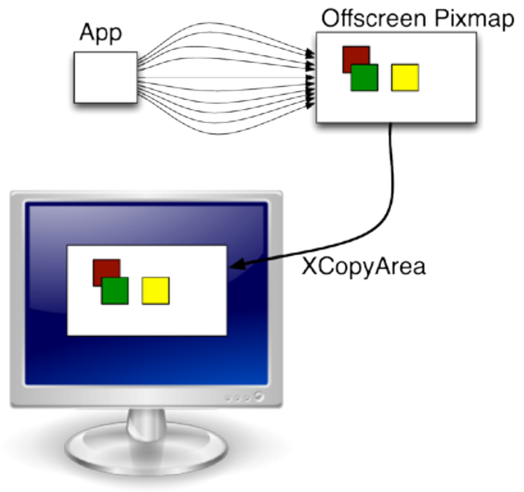
  4. Comment out this:

     ```
     if (XPending(display) == 0) {
        usleep(1000000/FPS-(end-lastRepaint));
     }
     ```

     and look at CPU usage

Double Buffering

- Flickering when an intermediate image is on the display
- Solution:
  - Create an off screen image buffer
  - Draw to the buffer
  - Fast copy the buffer to the screen



Double Buffering: doublebuffer.cpp

```
// 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 );
```

Painting Advice

- Keep it simple
  - Clear the window and redraw everything each frame
  - Use advanced methods (e.g. selective clearing, clipping) only if you really need them for performance
- Don’t repaint too often
  - remember framerate of display (60 FPS)
  - consider adding single “someChanged” bool flag
- Don’t flush too often
  - remember display framerate usually 60 FPS

Summary

- Events (definition, structure, selecting, etc)
- Blocking vs Non-Blocking Event Loop
- Animation
- Double Buffering