Window Systems

Goals: Virtual Devices

- Virtual display abstraction
- Multiplex physical input devices
- Simulated or higher level "devices"
- Limited resource management

Goals: Interface Uniformity (UI and API)

- UI
  - Provides consistent “look and feel”
- API
  - Provides virtual device abstraction
  - Performs low level ops

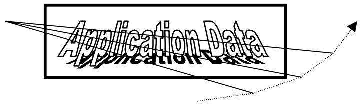
2 Views of the Window System

- User Interface
- Application Interface
  - Imaging model
  - Input model

Imaging Model

- Typically close to the hardware
  - Raster
  - Positive integer coords
- Primitives put values in frame buffer
  - Explicit bit pattern
- Low level view creates some problems

Problem #1: Dynamics e.g. Rubber-banding a line



- Line drawn with standard techniques
- How to "undraw" when moved?

Solutions

1) Redraw

2) “Save-unders”

Solutions

3) Bit manipulations

- Colors are bits in FB
- Don’t draw with line color, XOR with it
  - L ^ P =
- XOR again to “undraw”
  - L ^ P ^ P =
- Problems?

Solutions

4) Simulate bit planes

Problem #2: Color map (CLUT)

- Small # hardware color maps
  - share between windows/apps
  - each would like full power provided

- No real good solutions.
  - use 24 bits of color!
  - Other suggestions?

Solutions

1) dynamically switch color maps

2) no (direct) color map access

3) sharing schemes

Higher Level Imaging Models e.g. Postscript-based

- e.g. NeWS & NeXT [AKA OpenStep]
- Real valued coordinate system
- Support for full transformations
  - e.g., scale & rotate
- Richer primitives
  - e.g., Curves
- Stencil and paint model

Higher Level Imaging Models

- Advantages
  - Resolution/device independent
  - Can support full transformations
- Disadvantages
  - Slower
  - Harder to implement
  - For opaque model, some effects hard (XOR)

Other issues in imaging model

- Exposing/hiding facts of overlap
- Hierarchy
  - Windows within windows

Input models

- Handling input devices tedious
  - Want an abstraction for input
    - As disks, etc. are to file systems
- The "uniform event" input model
  - An event is a record of an input action
  - Events placed in a queue
    - processed asynchronously
  - “producer/consumer” between system/user

An event record

- What caused the event
  - e.g., left mouse button went down
- Where was the mouse
- When did the event occur
- Value associated with device action
  - e.g. ascii value of key, position of knob
- Additional Context
  - e.g. modifiers

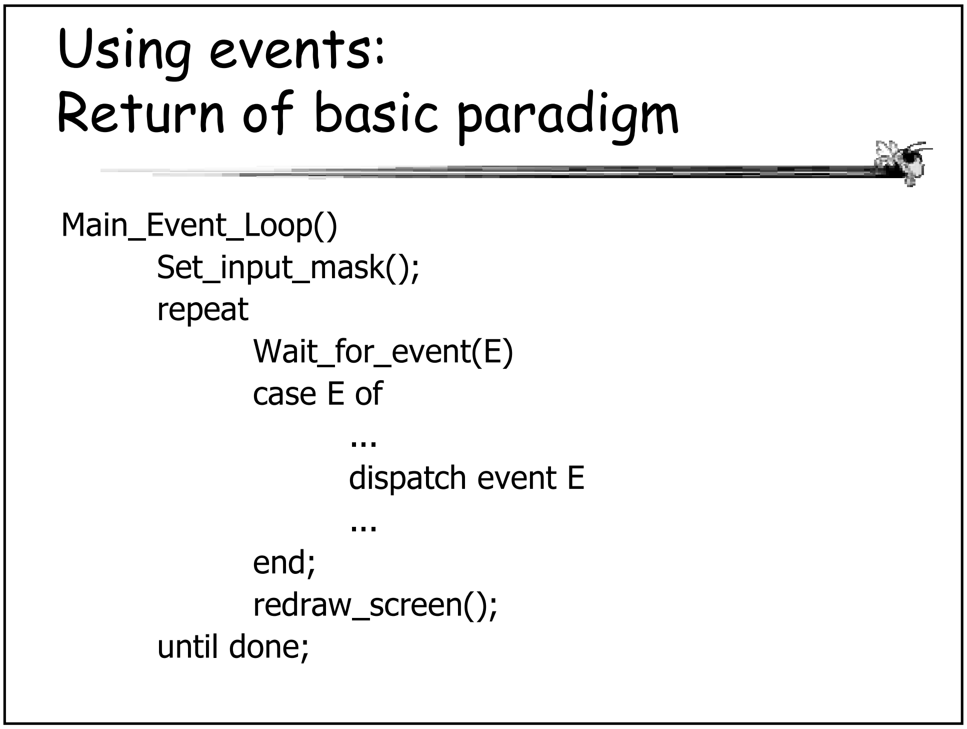
Example

- The Java 1.2 MouseEvent

http://java.sun.com/products/jdk/1.2/docs/api/java.awt.event.MouseEvent.html

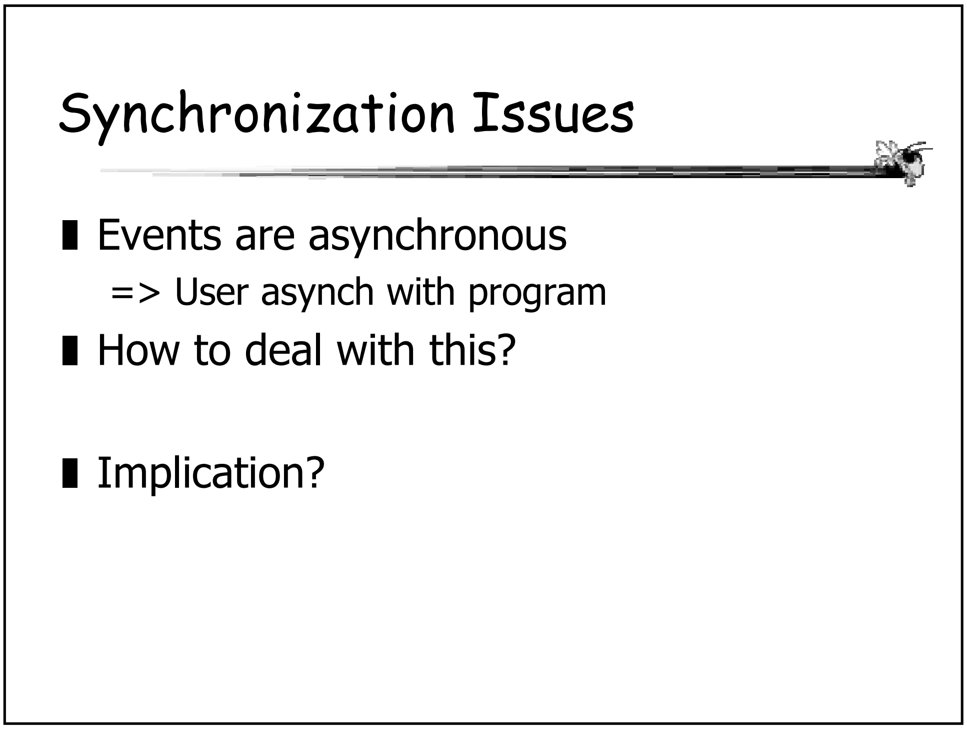
Higher level events

- Not a simple input device action
- e.g.



Using events: Return of basic paradigm

```
Main_Event_Loop()
        Set_input_mask();
        repeat
                Wait_for_event(E)
                case E of
                        ...
                        dispatch event E
                        ...
                end;
                redraw_screen();
        until done;
```



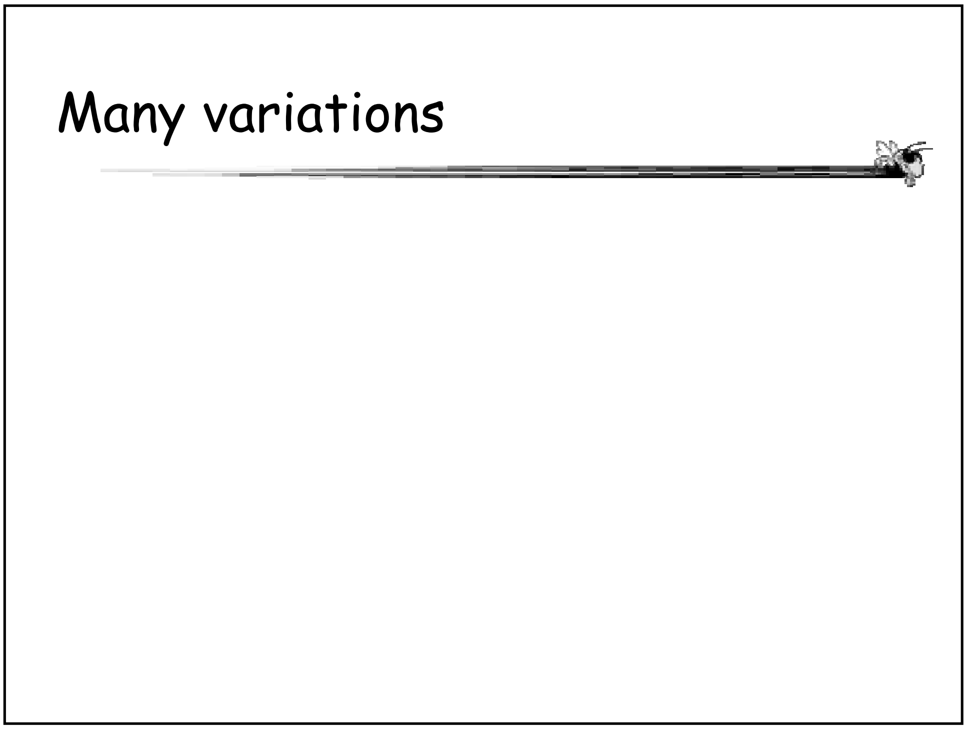
Synchronization Issues

- Events are asynchronous
  - => User asynch with program
- How to deal with this?
- Implication?

Example: drawing overshoot

Synchronization Issues

- Separate queues don't help
- Each thread needs one unified queue
  - Can be one per thread



Many variations