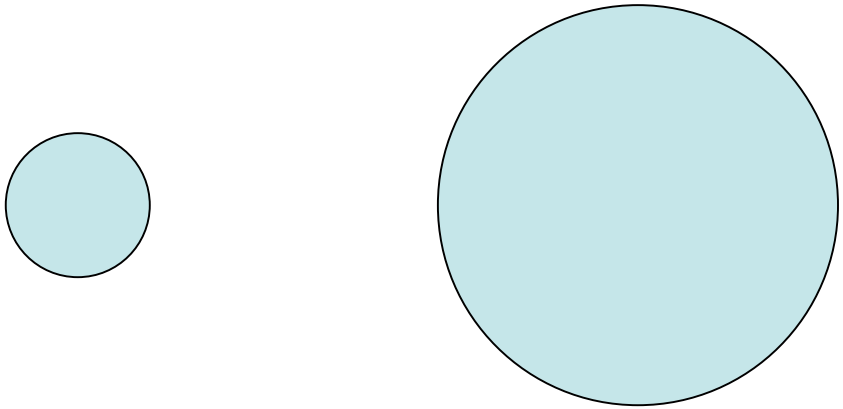
Tools, Layers, and Basic Organization of UI Software

Quick Review

- Fitts' Law
  - Larger targets easier to hit than smaller targets
  - Greater distance means longer time to hit target

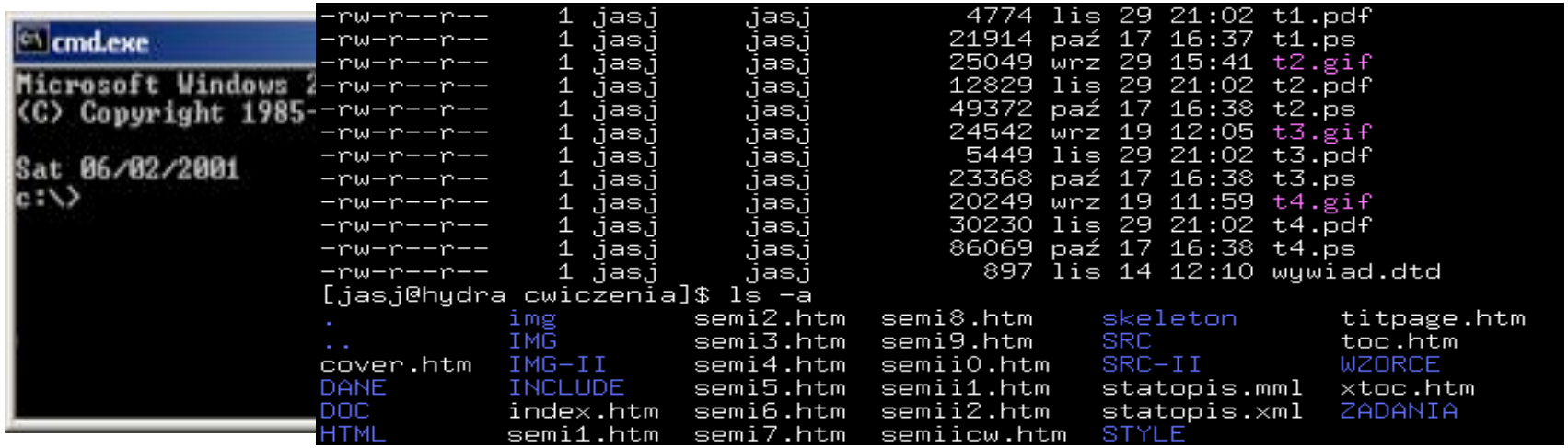


Outline

- Basic organization of user interfaces
  - 30,000 foot view of how user interfaces work
  - Most of the course will be examining the details
- User Interface Layers and Tools
- Building a UI in Java Swing

Sequential Programs

- Computer in charge, prompts for input
  - command-line prompts (DOS, UNIX)



- User waits on the program
  - program tells user it's ready for more input
  - then user enters more input
- Most intro programming courses teach this style

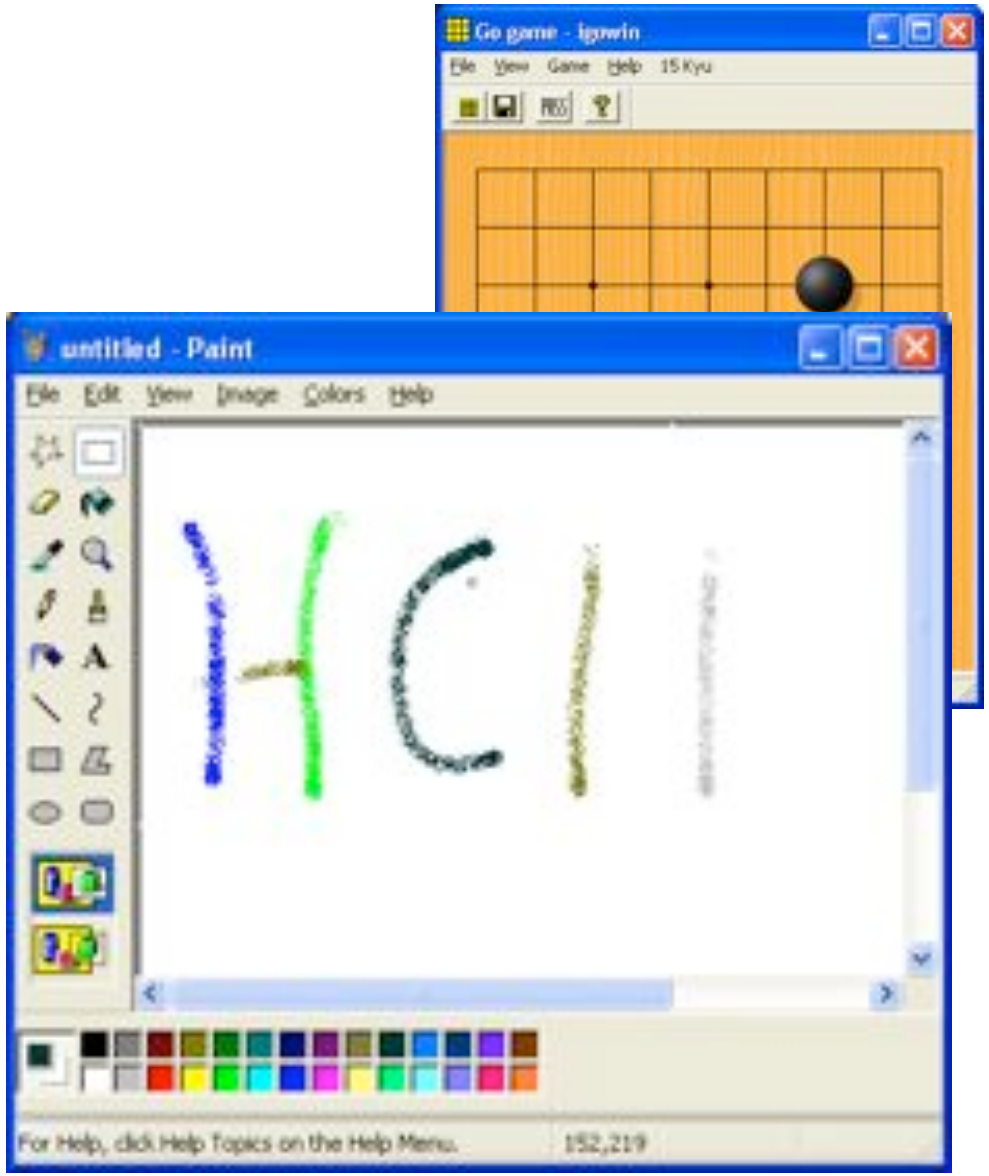
Sequential Programs (cont.)

- Imagine trying to edit and print a document
  - Issue command to modify document
  - View document
  - Issue next command, view again…
- This doesn't work well for highly-interactive apps
  - Blocks on input, system controls everything
    - Need to handle <u>any</u> legal user input at any time
  - Limits kind of inputs
    - Hard to model mouse input with linear I/O
  - Output and input often coupled
    - Ex. You can see a button AND can press on it
    - Ex. You can see text AND edit it

Two Basic Paradigms for Interaction

- Computer in charge vs…
- User in charge
  - Can click anywhere
  - Interact with any window
  - Freedom and control

- A different style of programming needed for GUIs to work



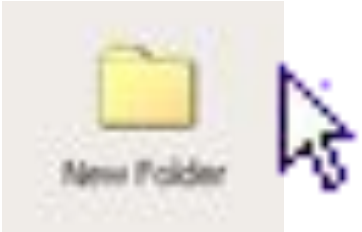
Key Idea #1

Event-Driven Programming

- Can’t tell in advance where user will direct input
  - Thus, can’t have synchronous input
  - Need to support asynchronous input

- **All** input from human to computer is done via events
  - mouse button ‘left’ went down
  - item ‘New Folder’ is being dragged
  - keyboard button ‘A’ was hit
  - keyboard focus event
  - …

Event-Driven Programming



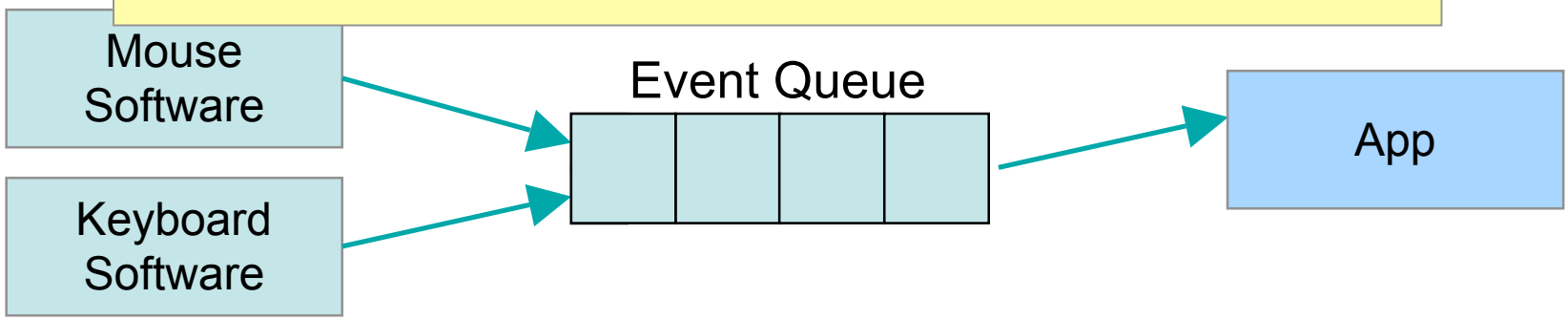
Icon –> Mouse entered
–> Mouse moved
–> Mouse moved
–> Mouse moved
–> Mouse pressed

–> Mouse dragged
–> Mouse dragged
–> …

Event-Driven Programming

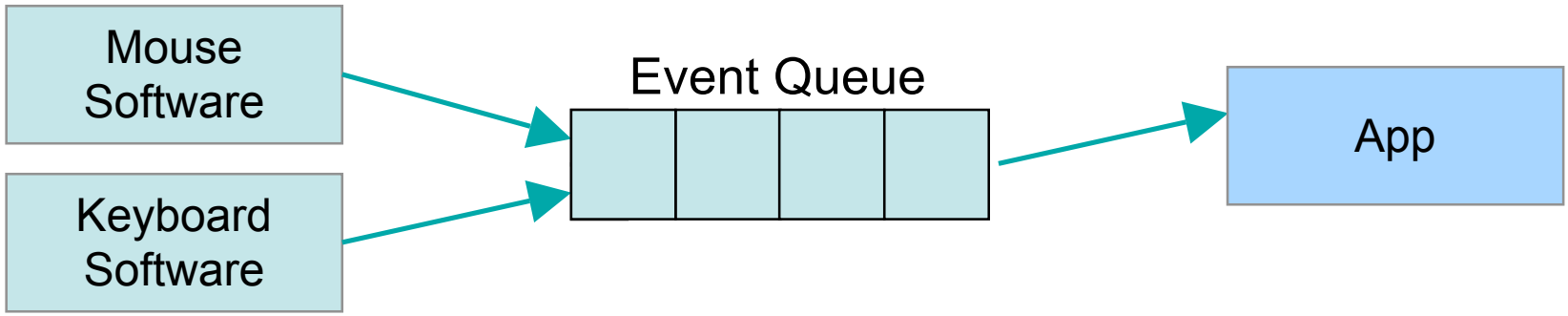
- All generated events go to a single *event queue*
  - Provided by operating system or GUI toolkit
  - Why have an event queue?
    - Why add a level of indirection?
    - Why not send event immediately to application?

Find someone in class you don't already know
Discuss for 4-5 minutes and come up with ideas



Event-Driven Programming

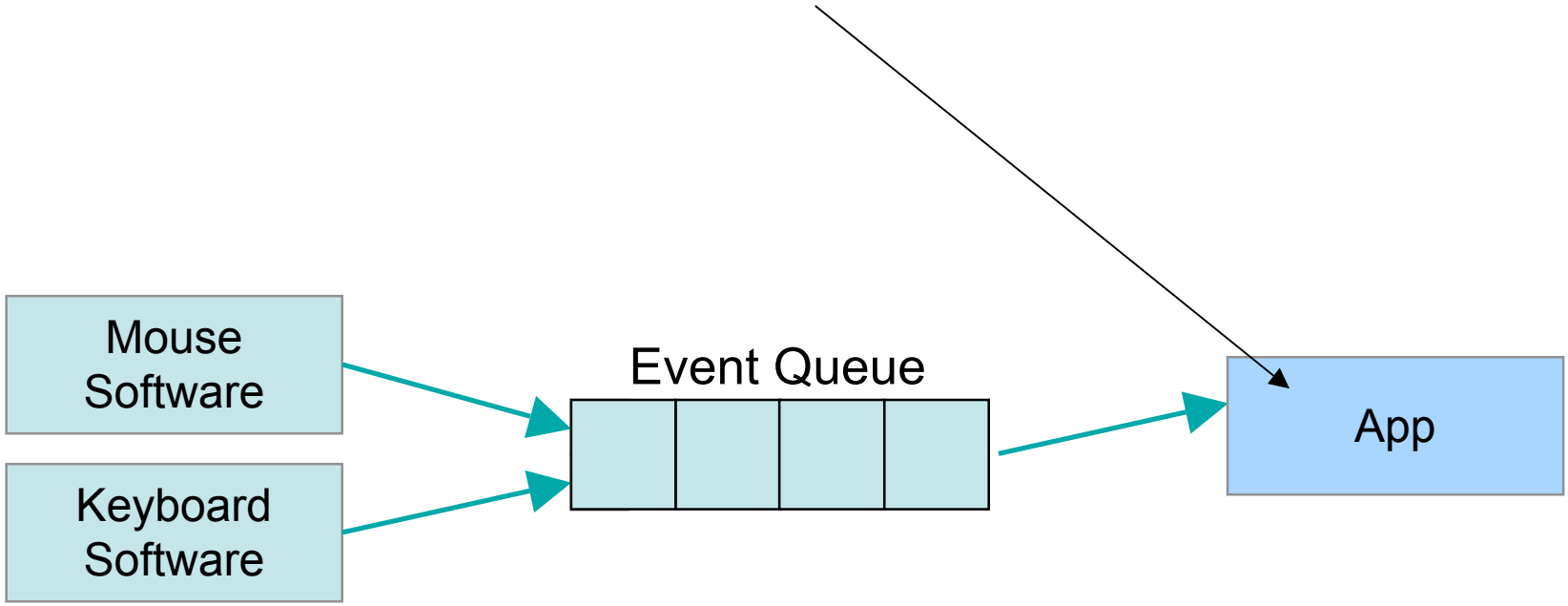
- All generated events go to a single *event queue*
  - Provided by operating system or GUI toolkit
  - Makes sure events are not accidentally dropped
  - Ensures that events are handled in the order they occurred
  - Hides specifics of input from apps
  - Easier to add new input devices
  - Easier to debug (if necessary)
  - Can do pre-processing of events (coalesce mouse events)



Event-Driven Programming

- Note: most event queues running on its own thread
  - This will have interesting implications later on

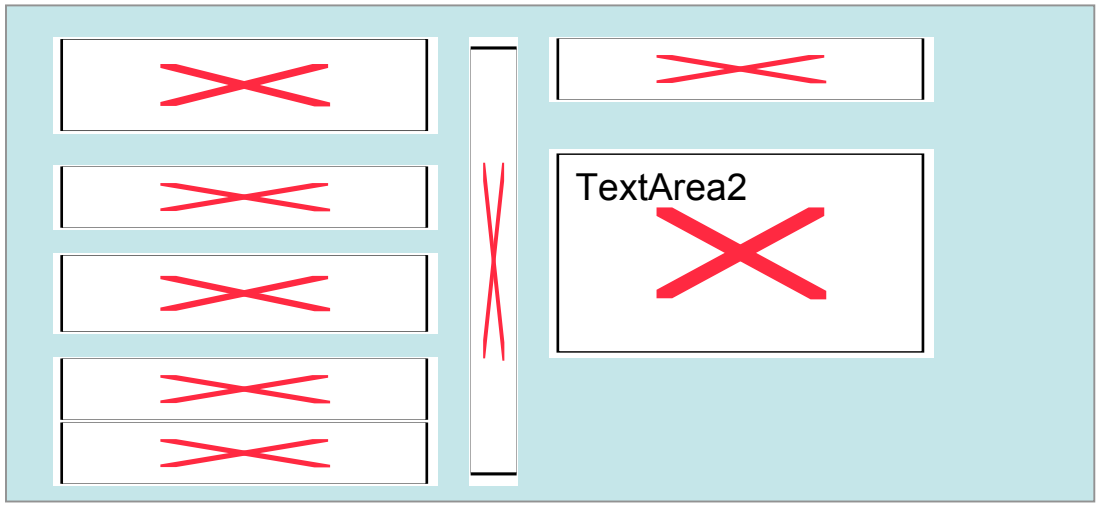
- Now how is the app structured?

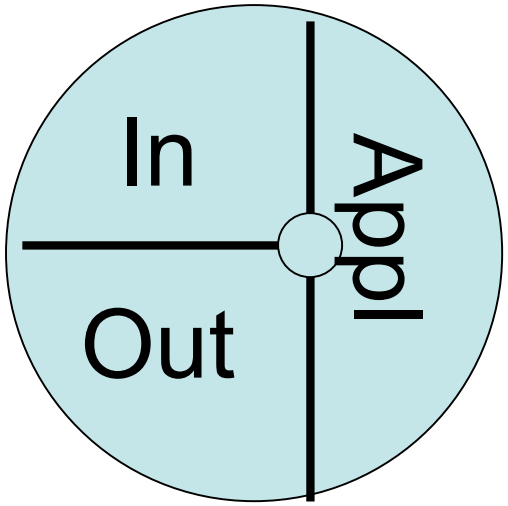


Key Idea #2

Key Idea #2: Object-based organization

- There is an object for every screen element
  - (Plus some hidden ones too)

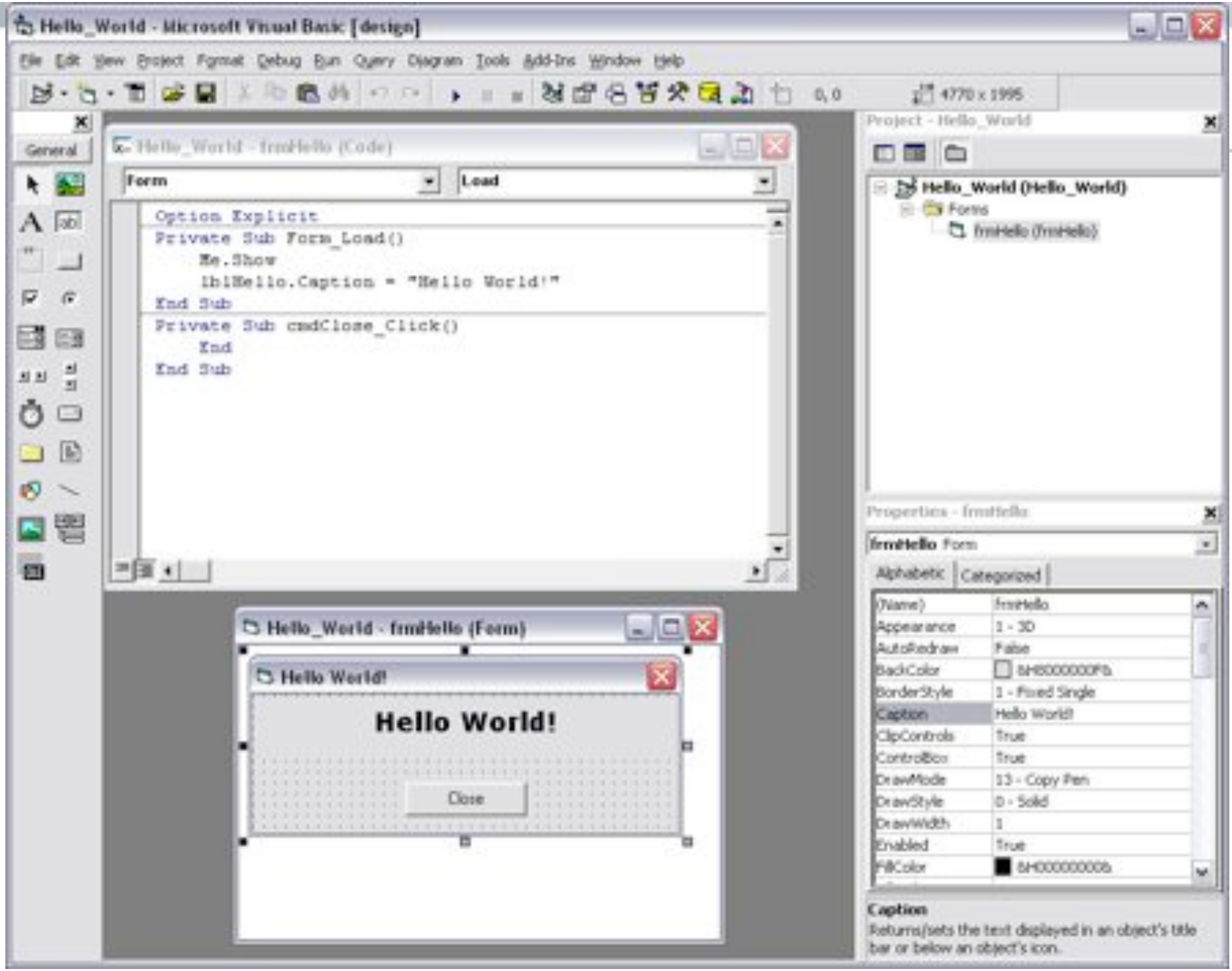




- Each object has its own behaviors and states
  - Can draw itself
  - Might contain internal state

Key Idea #2: Object-based organization

- Widgets, controls, components, interactors
  - Highly reusable interactive components
- Programming with widgets now consists of:
  - selecting an appropriate widget for a particular task
  - positioning that widget in a window
  - modifying widget properties to control how it looks and feels
  - adding the right behavior to that widget


Hello_World - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Hello_World - frmHello (Code)
Form
Load
Option Explicit
Private Sub Form_Load()
Me.Show
lblHello.Caption = "Hello World!"
End Sub
Private Sub cmdClose_Click()
End
End Sub
Hello_World - frmHello (Form)
Hello World!
Hello World!
Close
Project - Hello_World
Hello_World (Hello_World)
Forms
frmHello (frmHello)
Properties - frmHello
frmHello Form
Alphabetic
Categorized
(Name) frmHello
Appearance 1 - 3D
AutoRedraw False
BackColor
BorderStyle 1 - Fixed Single
Caption Hello World!
ClipControls True
ControlBox True
DrawMode 13 - Copy Pen
DrawStyle 0 - Solid
DrawWidth 1
Enabled True
FillColor
Caption
Returns/sets the text displayed in an object's title bar or below an object's icon.

Key Idea #2: Object-based organization

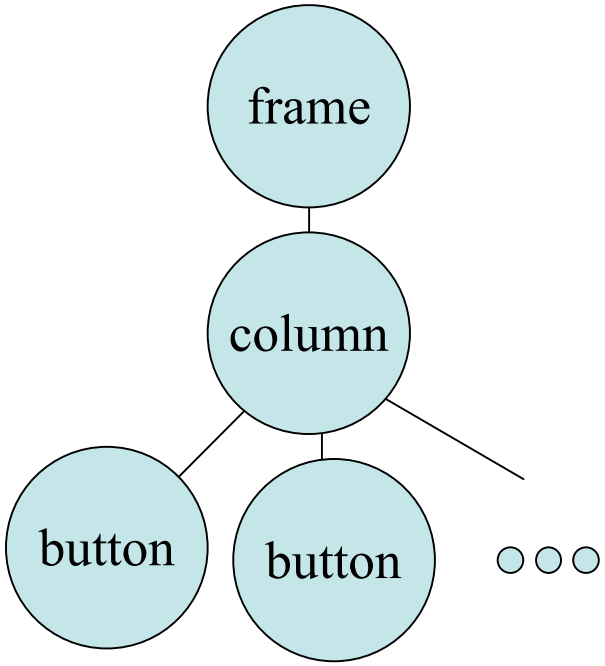
- GUIs and objects have strong natural relationship
  - GUIs led to many advances in OOP
  - GUIs helped propel OOP into mainstream

- Widgets only describe individual components
  - How to organize entire windows?

Key Idea #3

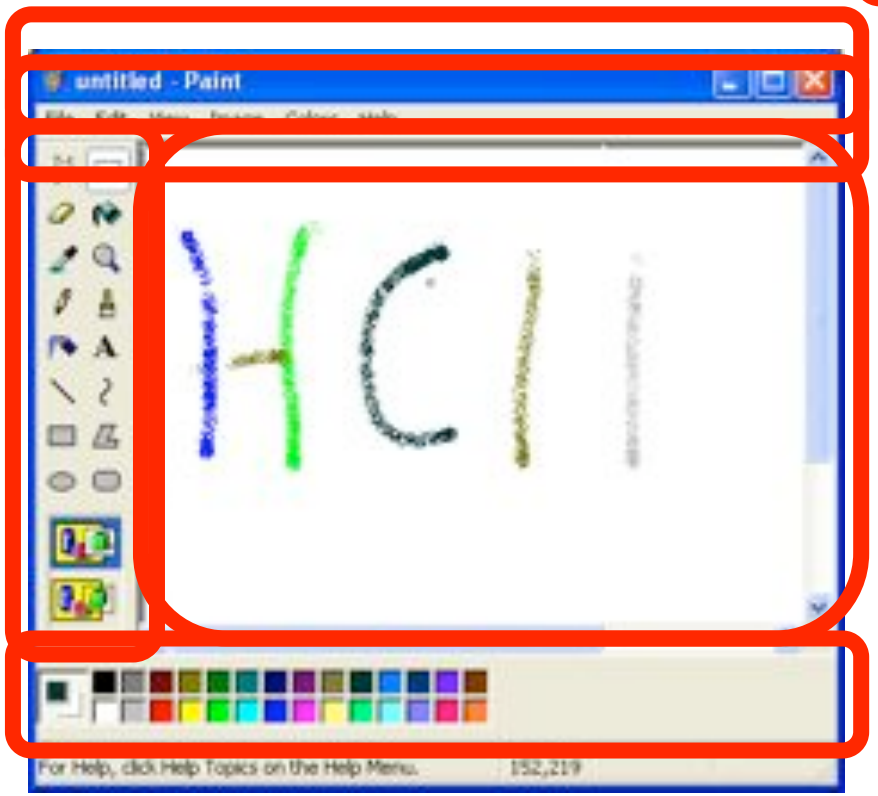
Key Idea #3: Component Trees

- Widgets are organized hierarchically
  - Normally reflecting spatial containment relationships
- Everything is done through this tree
  - Build an interface == build a tree
  - Change an interface == change a tree
  - (Note that HTML is like this too)
  - (Similar to scenegraph in graphics)
- Also several alternative names
  - Interactor trees, Component trees, etc



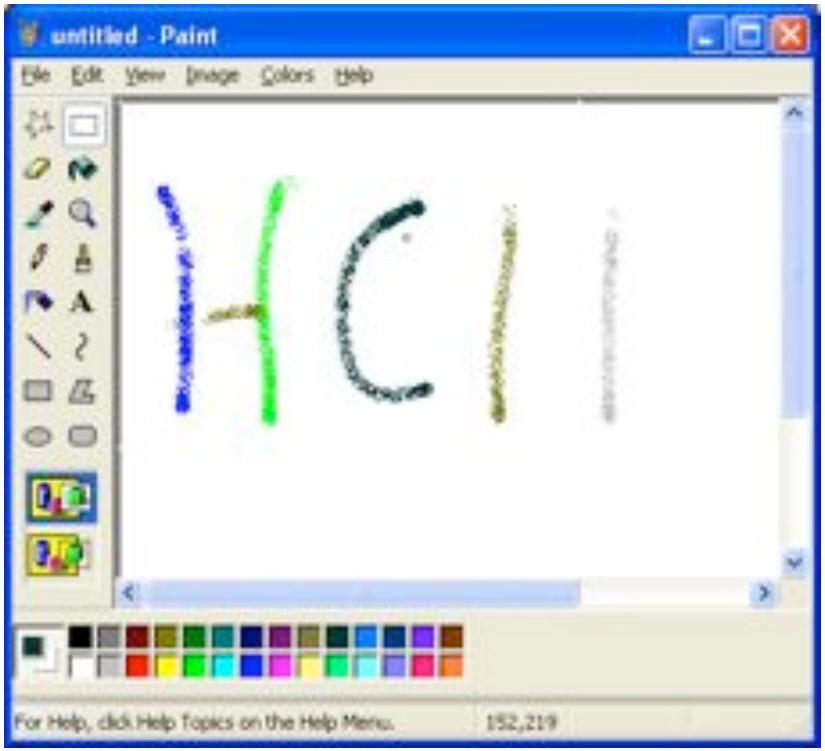
Component Tree Rough Example

Component Tree Rough Example



- “untitled - Paint” window
  - Menu Bar
    - File
    - Edit
    - …
  - Paint Toolbar
    - 
    - …
  - Color Palette
  - Scrollable Panel
    - Horizontal Scrollbar
    - Vertical Scrollbar
    - Drawing Canvas
  - …

Component Tree Rough Example



“untitled - Paint” window

- Menu Bar
  - File
  - Edit
  - …
- Paint Toolbar
  - 
  - …
- Color Palette
- Scrollable Panel
  - Horizontal Scrollbar
  - Vertical Scrollbar
  - Drawing Canvas
- …

Notes on Component Trees

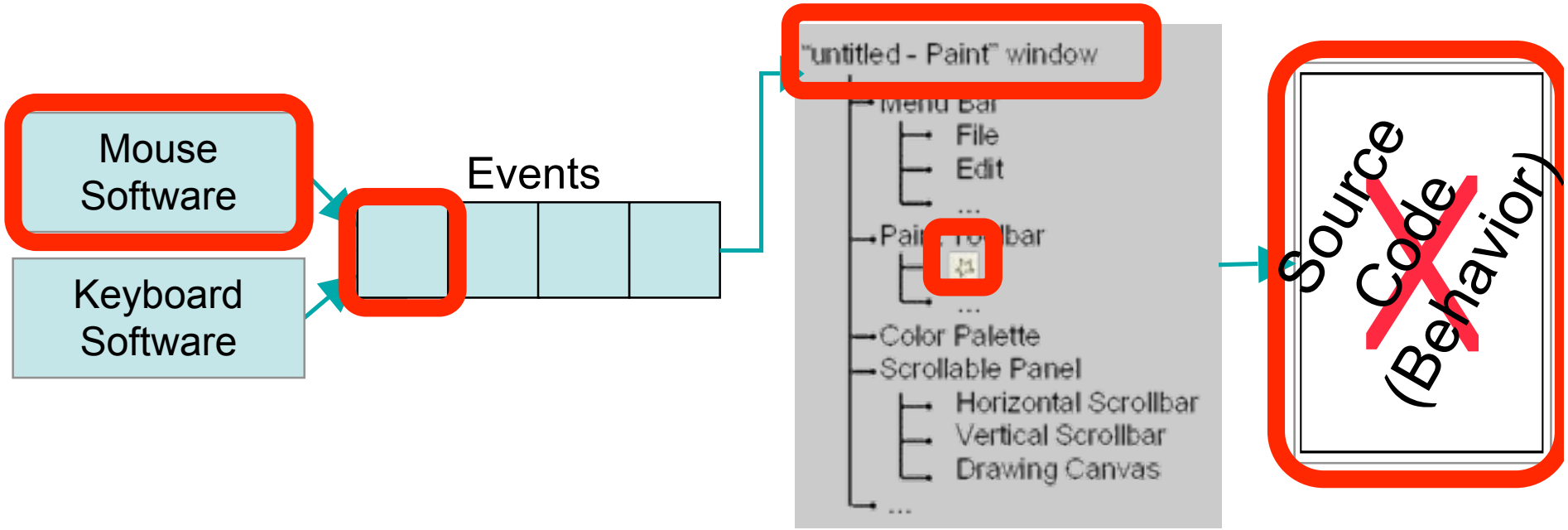
- Tree makes it easy to do certain things
  - Move a parent node, moves all of its children too
- Often, you do not interact with tree directly
  - No explicit notion of this tree in many GUIs
  - Indirect interaction with tree, `addChild(Component)`
  - Less true for HTML (tree is highly exposed as DOM)

Recap

- Event-driven programming
  - All user input handled as events
  - Events stored in event queue before sent off to app
- Widgets
  - Buttons, checkboxes, text input fields, etc
- Component trees
  - Windows can be represented internally as trees

Tying it All Together

```
while (app is running) {
        get next event
        send event to right widget (dispatch)
}
```

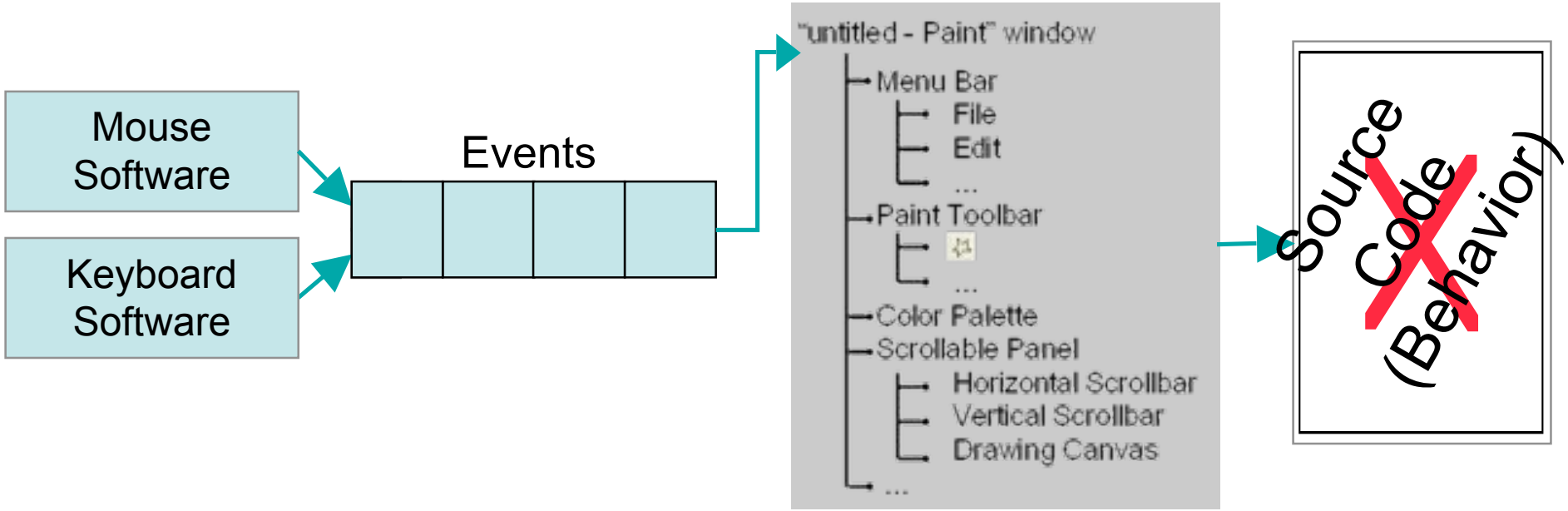


Tying it All Together

```
while (app is running) {
        get next event
        send event to right widget (dispatch)
}
```

- Event loop
  - Typically provided for you by most GUI toolkits
    - Java, MFC
  - Sometimes has to be done manually
    - Palm Pilot, Win32

Tying it All Together



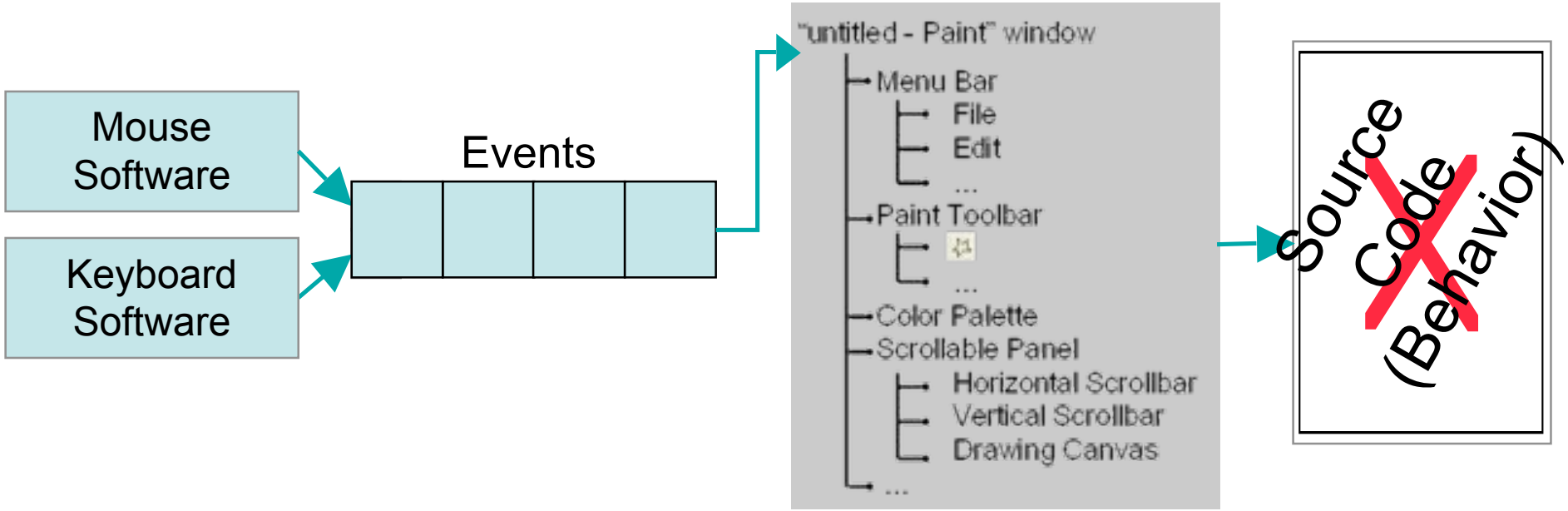
These are done automatically for you

- Low-level input
- Event queue
- Widget management (click, move, draw)
- Tree data structure

You do this

- What widgets to use
- Layout of widgets
- Behavior when used
- Data model

Tying it All Together



Some implicit design constraints here:

- One person
- One computer
- One output
- Optimized for keyboard and mouse (ex. no speech)
- May need modified approaches in future

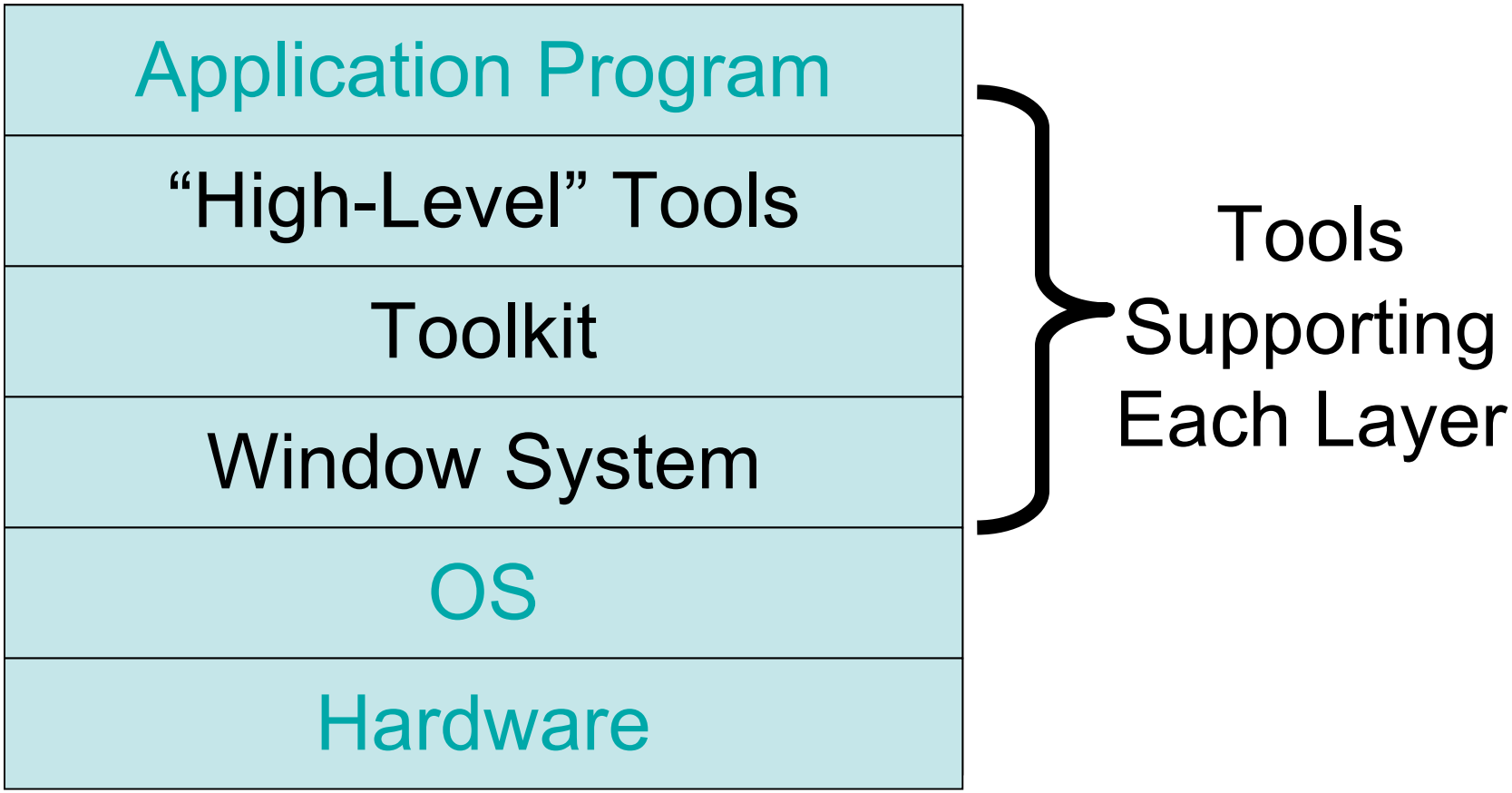
Administrivia

- Everyone visited the home page?
- Questions on first assignment?
- Anyone interested in doing a summer internship at Hewlett-Packard research labs in the UK?

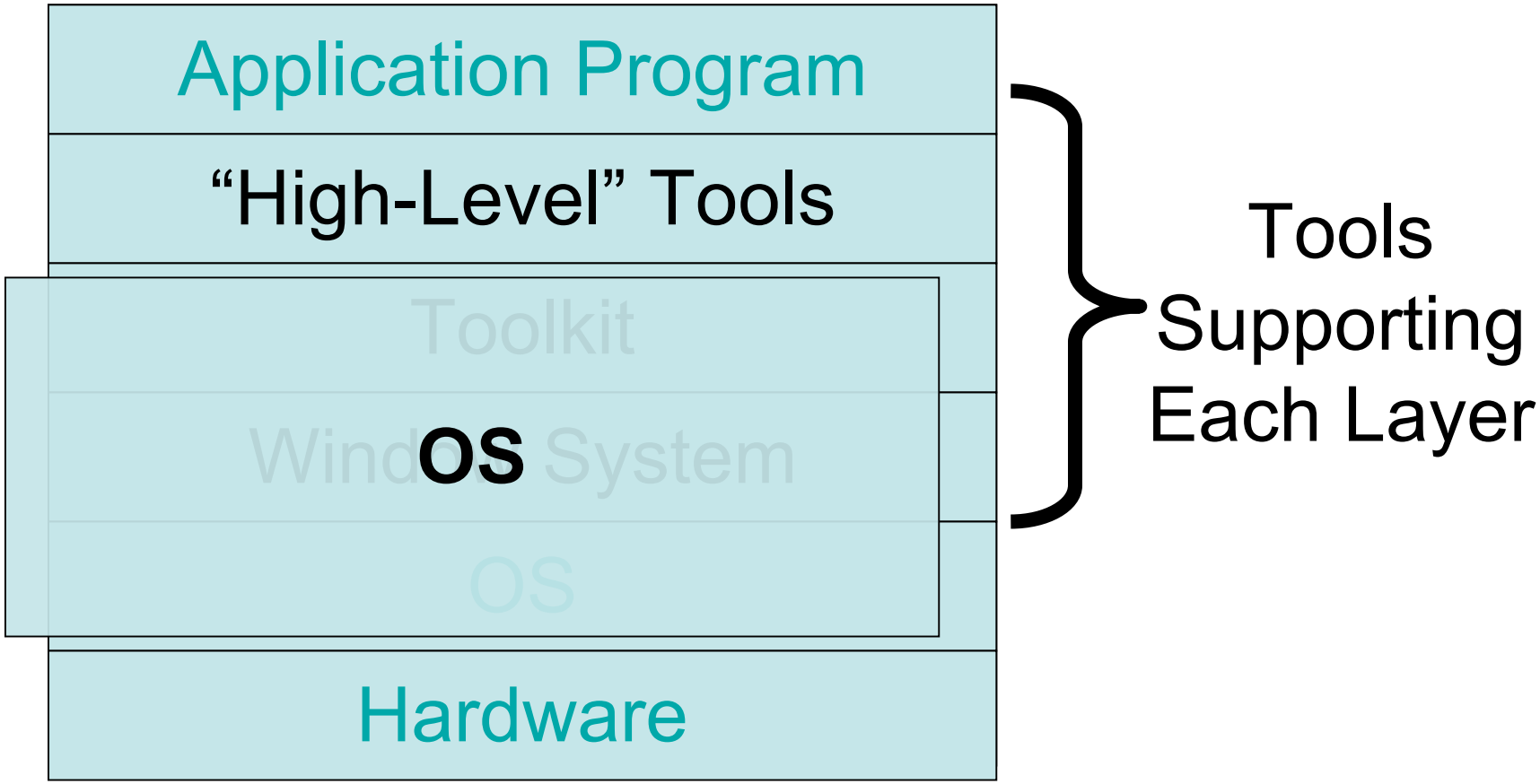
Help For Implementing Systems?

- Just showed a dataflow perspective
  - What talks to what
  - How things are wired together
- Need way of cleanly organizing what we talked about
  - Understandable, modularized
- Look at the same concepts from layered perspective
  - What layers are there?
  - What are the responsibilities of each layer?

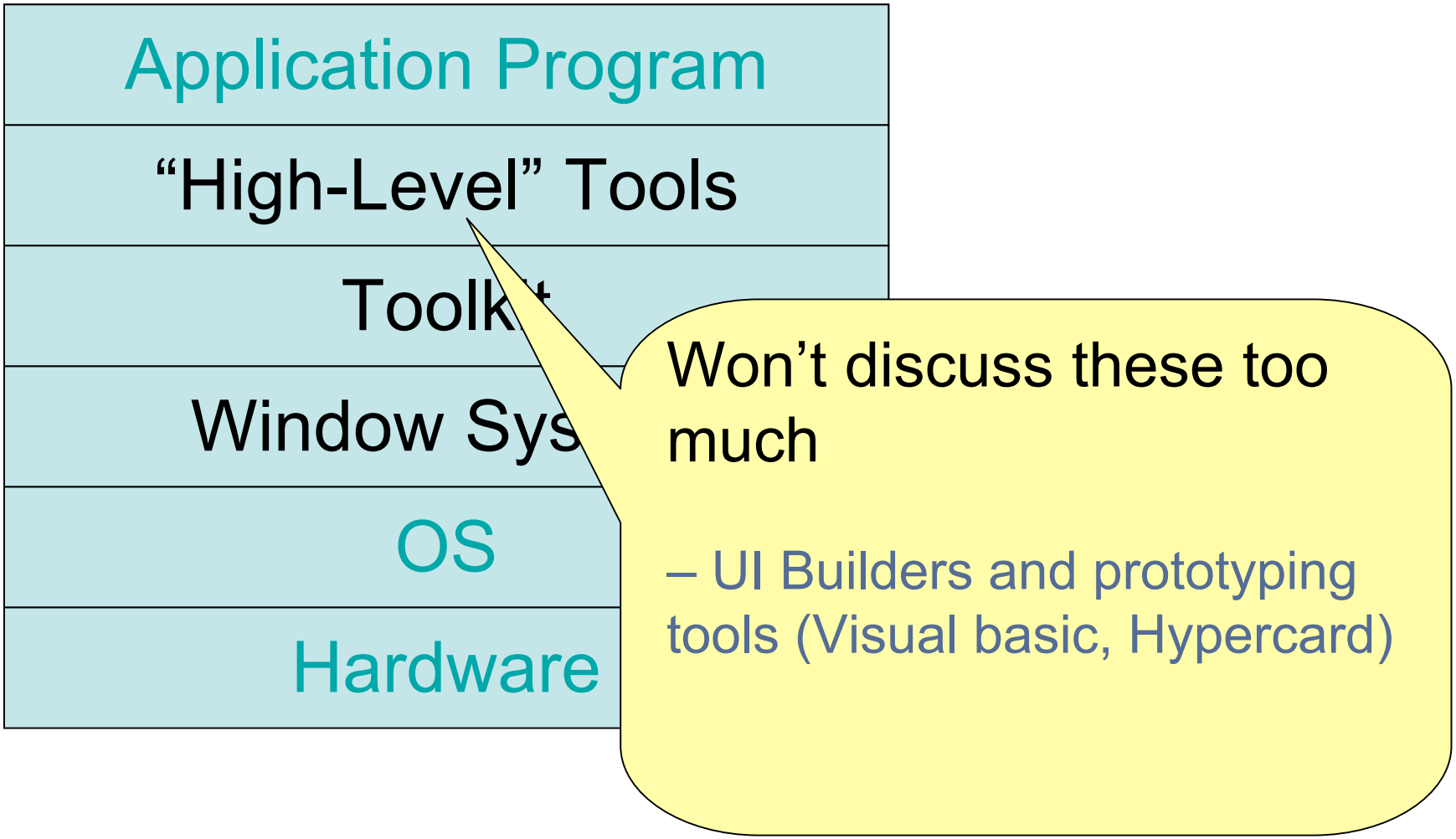
Layers of UI Software



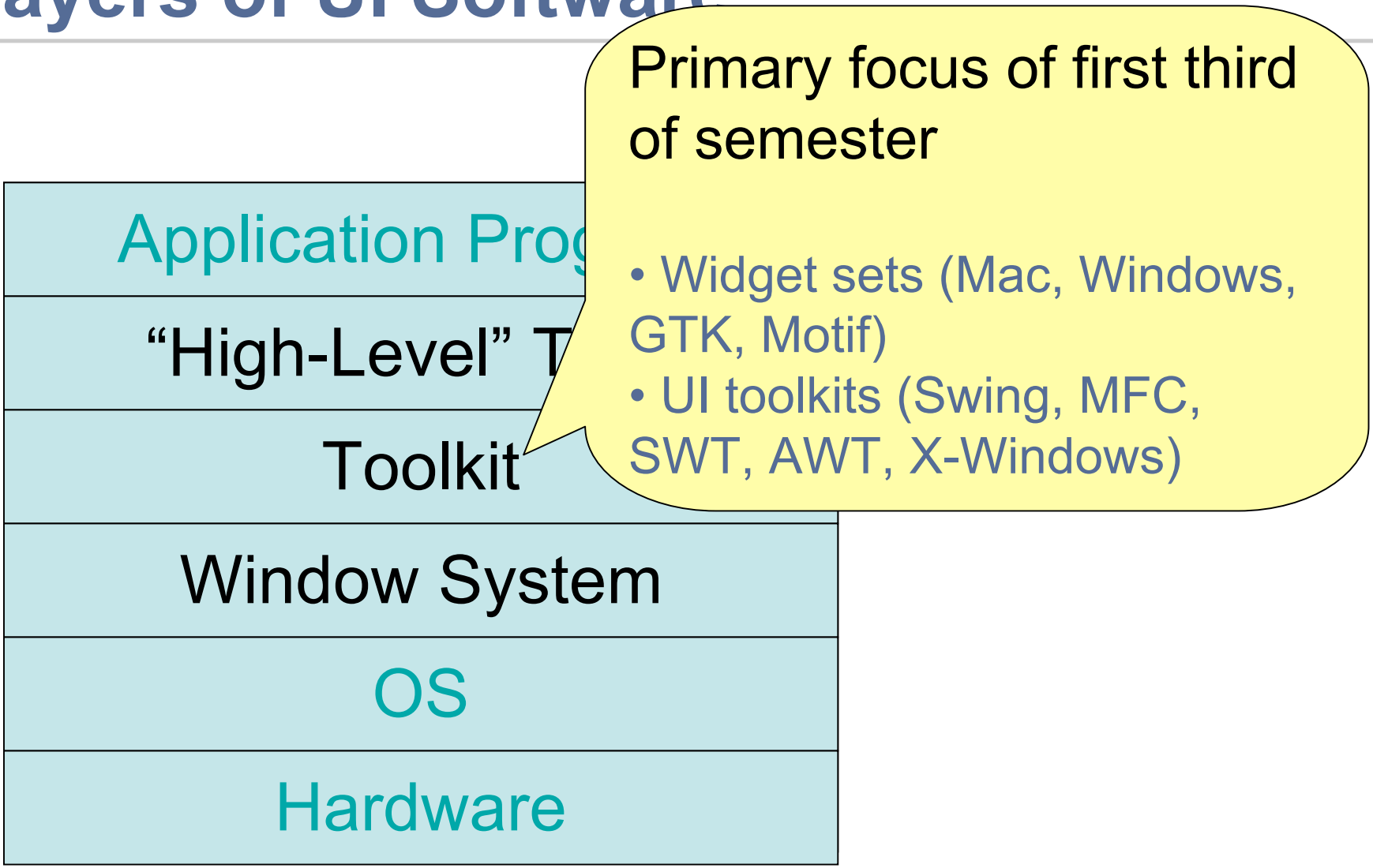
Layers of UI Software (Commercial)



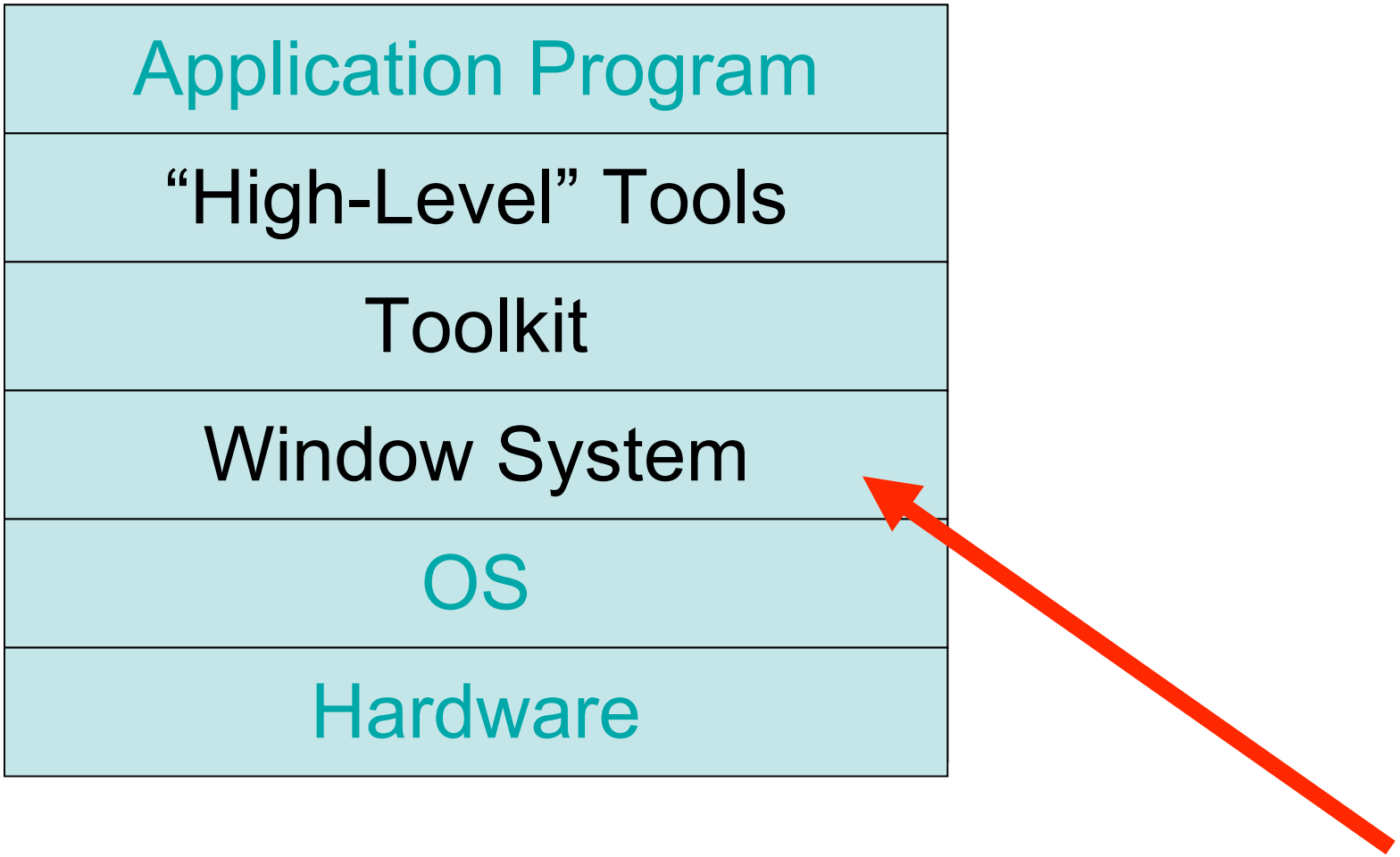
Layers of UI Software



Layers of UI Software

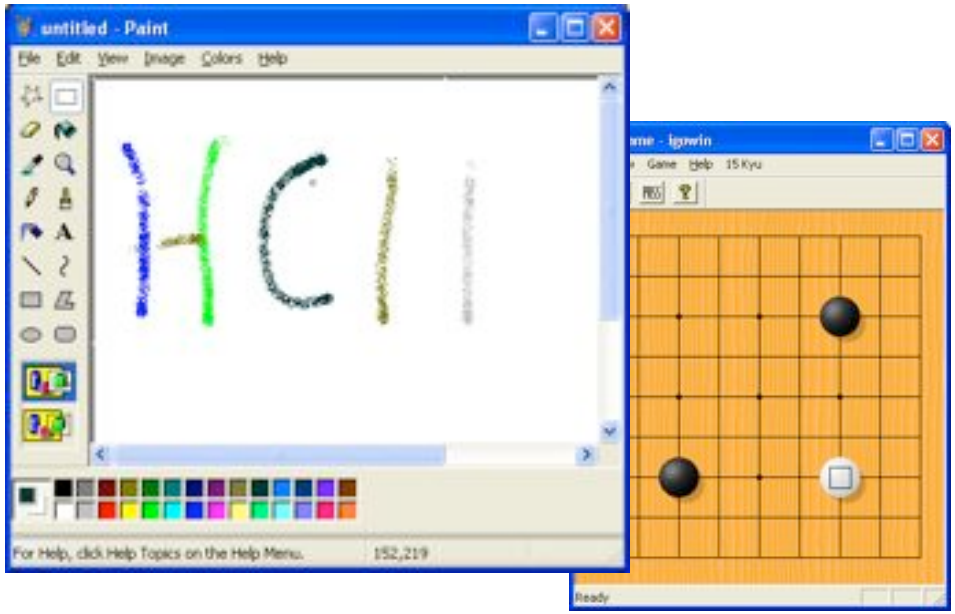


Layers of UI Software



Window Systems

- Provides a virtual device abstraction
  - Each program can act as if it has a complete control over screen & input
  - Window system manages and controls multiple contexts, logically separated, but implemented together
    - Analogous to OS multiplexing of CPU and memory

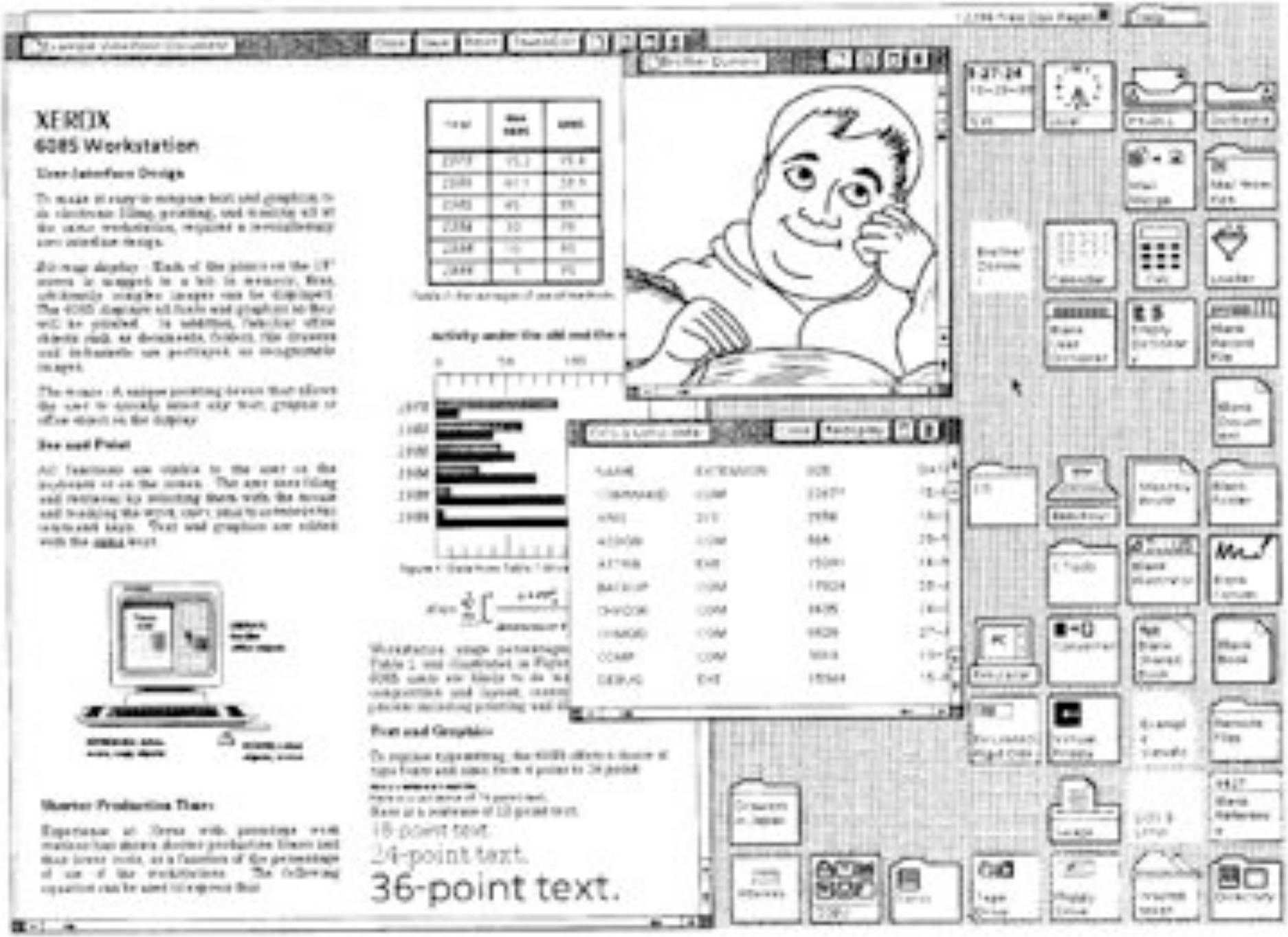


Window Managers (History)

- Multiple (tiled) windows in research systems of 1960's: NLS, etc.

- Overlapping in Alan Kay's thesis (1969)
- Smalltalk (1974) at Xerox PARC

Window Managers (History)


XEROX
6085 Workstation
See and Point
Shorter Production Times
Text and Graphics
18-point text.
24-point text.
36-point text.

Window Managers (History)

- Successful because multiple windows help users manage scarce resources
  - Screen space and input devices
  - Attention of users
  - Affordances for reminding and finding other work

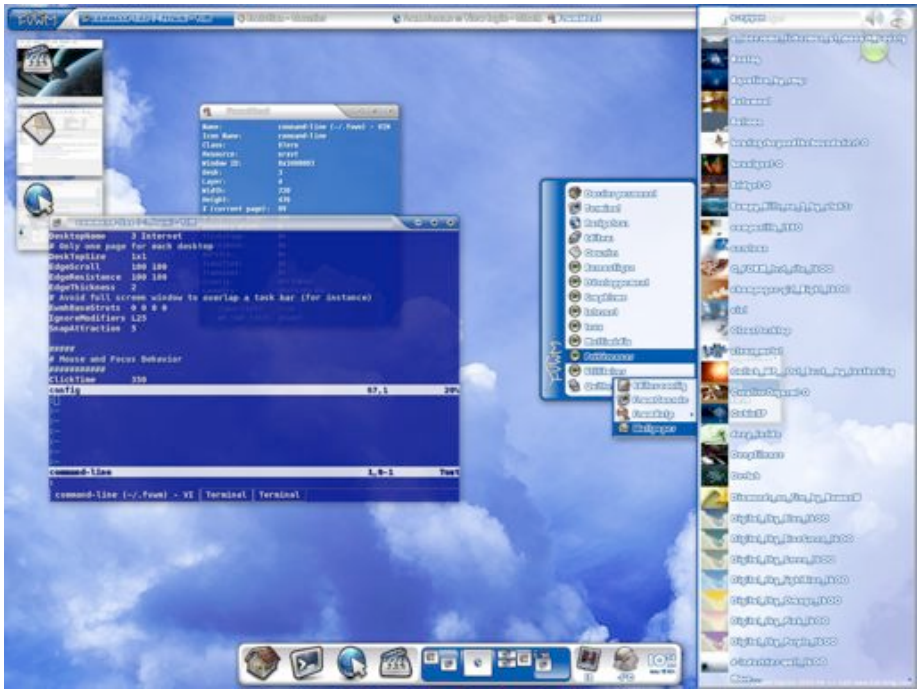
Windows, Components

- "Window Manager"
  - User interface to the windows themselves
  - Decorations on windows, overall look and feel
  - Mouse and keyboard commands to control windows
  - **Mechanics of the windows themselves (higher level)**

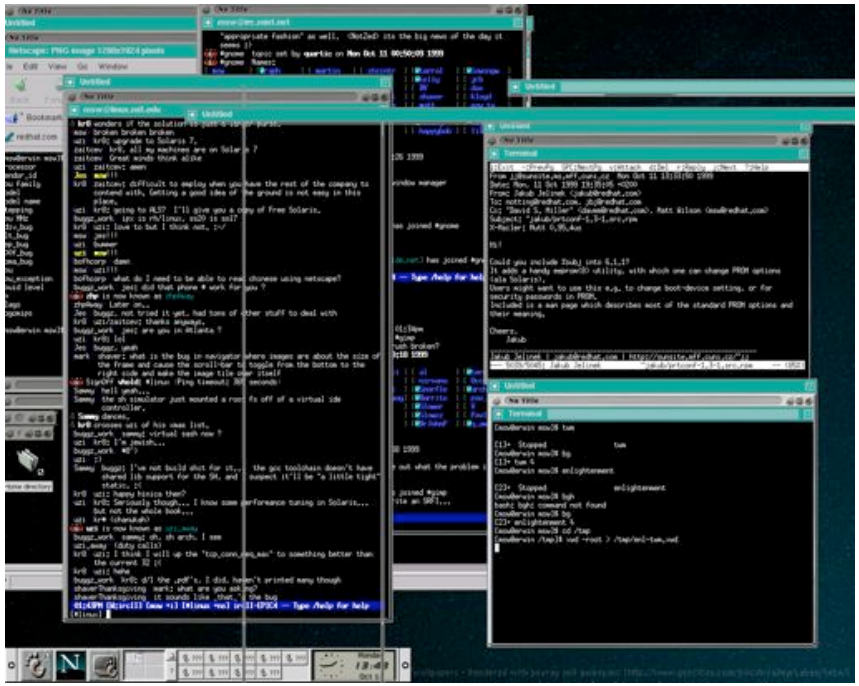
- "Window System"
  - Programming interface
  - Output graphics to a window
  - Input from mouse and keyboard to appropriate component
  - **Everything inside a window (lower level)**

Windows, cont.

- Different Window Managers on same Window System
  - fvwm, twm, Enlightenment, Motif, etc on top of X-windows
  - Allows diversity and user preference



fvwm

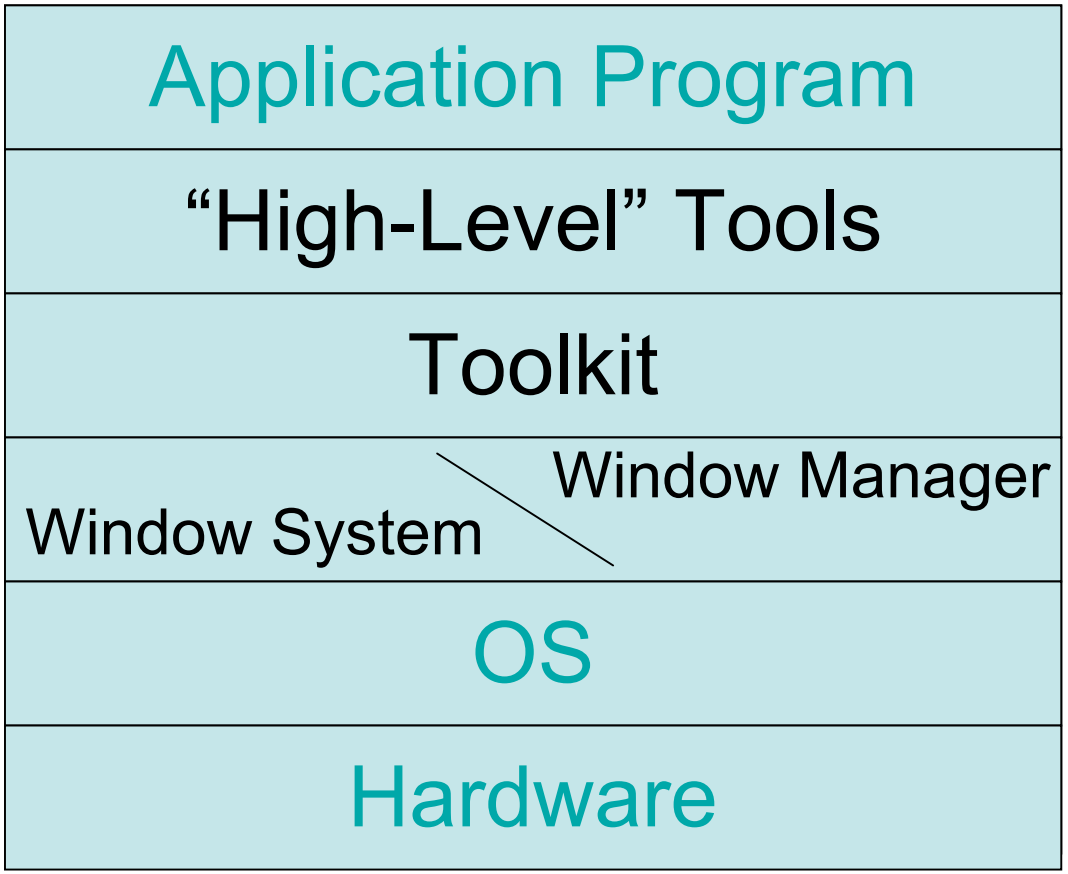


twm

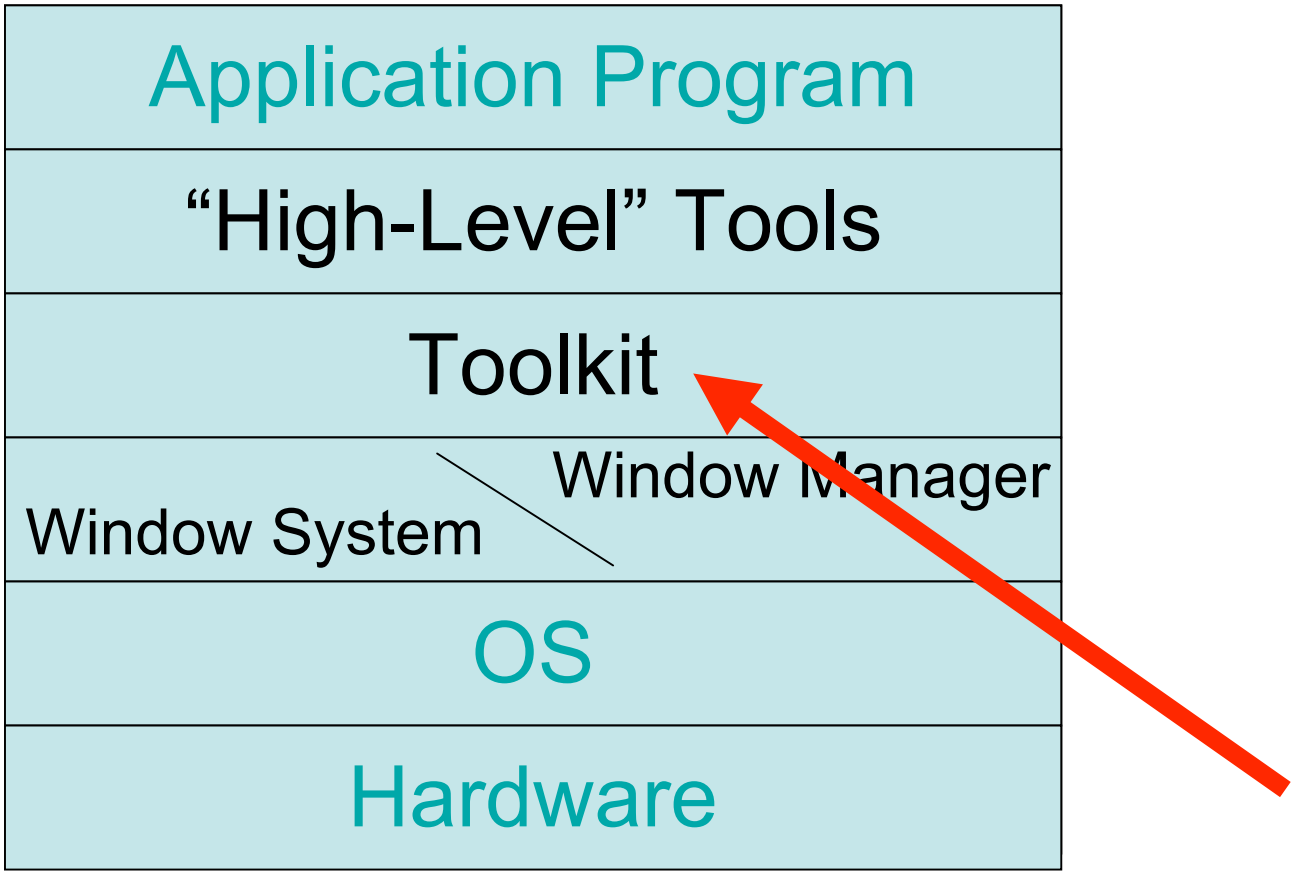
Windows, cont.

- Different Window Managers on same Window System
  - fvwm, twm, Enlightenment, Motif, etc on top of X-windows
  - Allows diversity and user preference
- Different Window System on same hardware
  - SunTools, X, NeWS on Unix machines
  - Different programming models for developing GUI apps
- Many systems combine Window System and Window Manager
  - SunTools, Macintosh Quartz Compositor, MS Windows, NEXTSTEP

Layers of UI Software

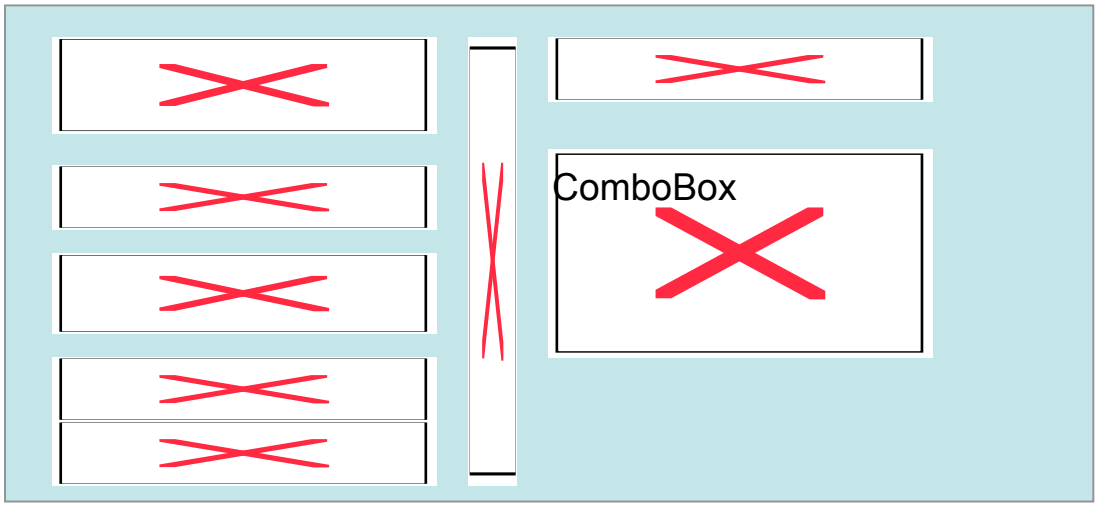


Layers of UI Software



Toolkits

- Recall: widgets are graphical objects that can be manipulated by users to input values
  - Menus, scroll bars, text entry fields, buttons, etc.



- Toolkits are libraries of widgets
  - Motif, GTK+, Qt, AWT, Swing, SWT, Cocoa, MFC
  - Used directly only by programmers

Toolkit Advantages

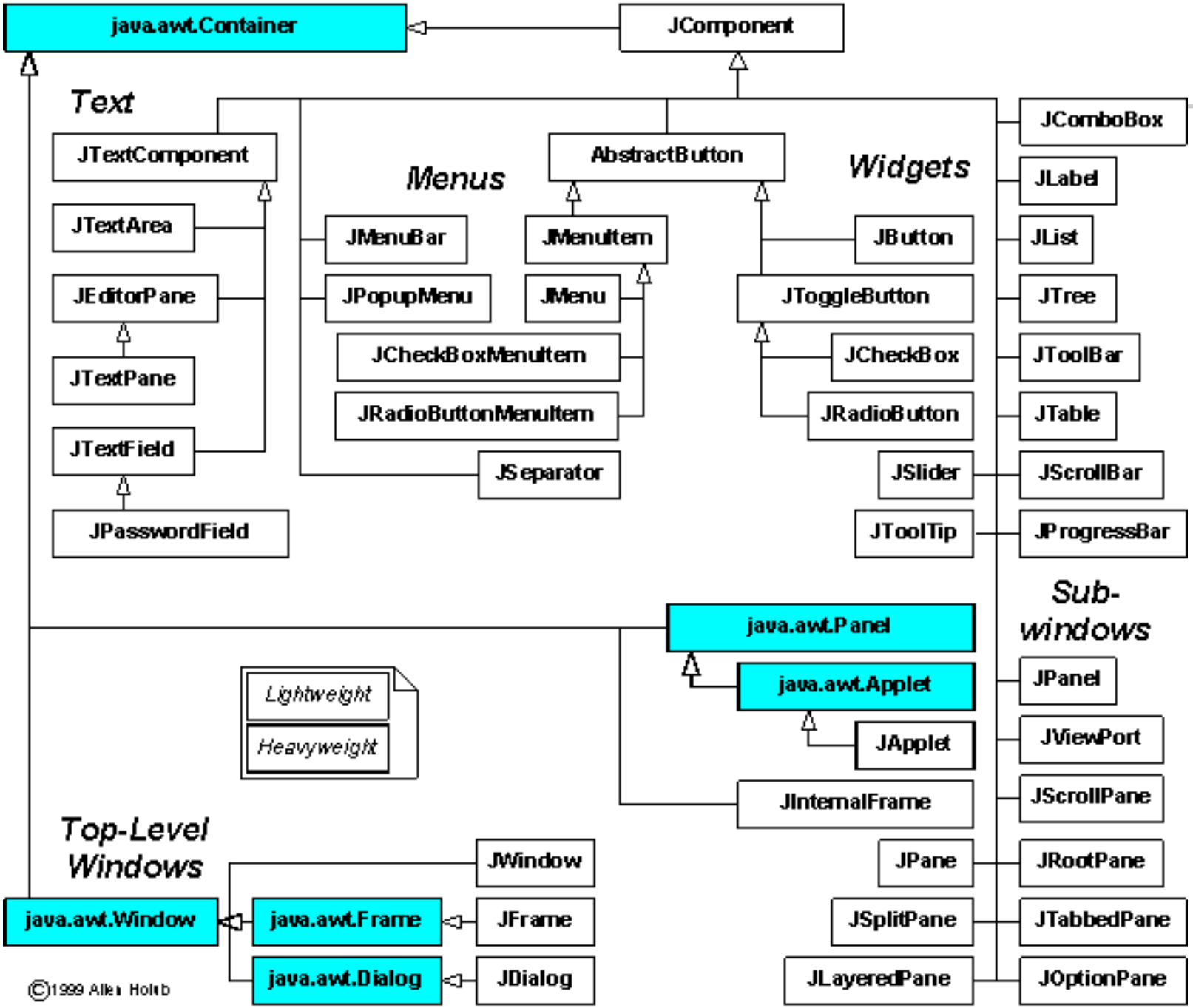
- Consistent Look and Feel
  - Key insight of Macintosh toolbox
  - Path of least resistance was to be consistent

Toolkit Advantages

- Consistent Look and Feel
  - Key insight of Macintosh toolbox
  - Path of least resistance was to be consistent
- Structured the programming task
  - Choose what widget, choose placement, choose properties, link to behavior
- Re-use of code
  - Lot less work to use toolkit library than to recreate the wheel
  - Lot less bugs too

But...

- Can be hard to use:
  - Very large libraries
    - Can end up as a complicated mess
    - Very large manuals
  - No help with when and how to call what


java.awt.Container
JComponent
Text
JTextComponent
JTextArea
JEditorPane
JTextPane
JTextField
JPasswordField
Menus
JMenuBar
JPopupMenu
JCheckBoxMenuItem
JRadioButtonMenuItem
JSeparator
AbstractButton
JMenuItem
JMenu
Widgets
JButton
JToggleButton
JCheckBox
JRadioButton
JSlider
JToolTip
JComboBox
JLabel
JList
JTree
JToolBar
JTable
JScrollBar
JProgressBar
Sub-windows
java.awt.Panel
java.awt.Applet
JApplet
JInternalFrame
JPanel
JViewPort
JScrollPane
JPane
JRootPane
JSplitPane
JTabbedPane
JLayeredPane
JOptionPane
Lightweight
Heavyweight
Top-Level Windows
JWindow
java.awt.Window
java.awt.Frame
JFrame
java.awt.Dialog
JDialog
©1999 Allen Holub

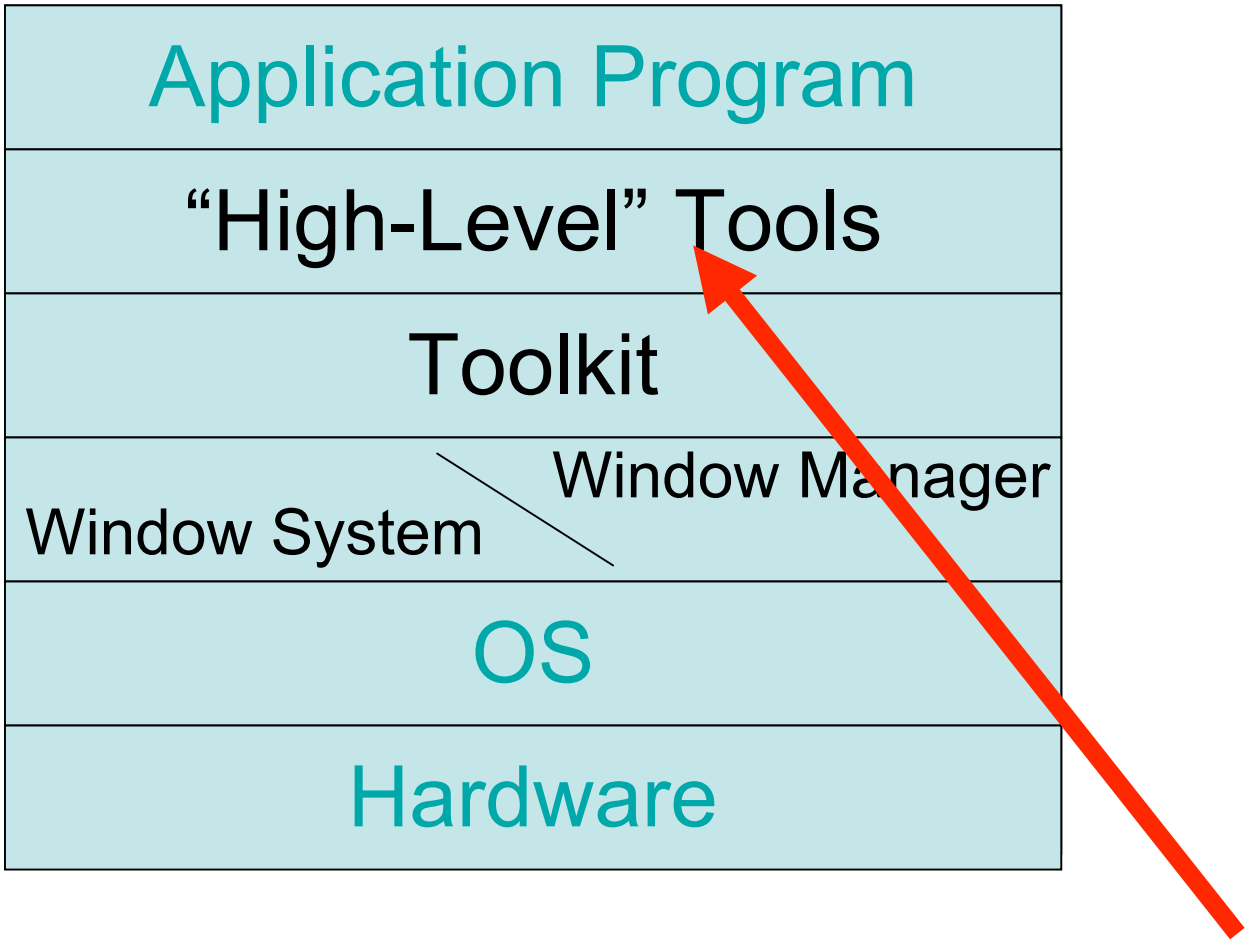
Microsoft Foundation Class Library Version 6.0

- **CObject**
  - **Application Architecture**
  - **CCmdTarget**
    - CWinThread
      - CWinApp
        - COleControlModule
          - user application
    - CDocTemplate
      - CSingleDocTemplate
      - CMultiDocTemplate
    - COleObjectFactory
      - COleTemplateServer
    - COleDataSource
    - COleDropSource
    - COleDropTarget
    - COleMessageFilter
    - CConnectionPoint
    - CDocument
      - COleDocument
        - COleLinkingDoc
          - COleServerDoc
            - CRichEditDoc
      - user documents
    - CDocItem
      - COleClientItem
        - COleDocObjectItem
        - CRichEditCntrItem
        - user client items
      - COleServerItem
        - CDocObjectServerItem
        - user server items
    - CDocObjectServer
    - user objects
    - **Exceptions**
      - CException
        - CArchiveException
        - CDaoException
        - CDBException
        - CFileException
        - CInternetException
        - CMemoryException
        - CNotSupportedException
        - COleException
        - COleDispatchException
        - CResourceException
        - CUserException
    - **File Services**
      - CFile
        - CMemFile
          - CSharedFile
        - COleStreamFile
          - CMonikerFile
            - CAsyncMonikerFile
              - CDataPathProperty
                - CCachedDataPathProperty
        - CSocketFile
        - CStdioFile
          - CInternetFile
            - CGopherFile
            - CHttpFile
      - CRecentFileList
    - **Window Support**
    - **CWnd**
      - **Frame Windows**
        - CFrameWnd
          - CMDIChildWnd
            - user MDI windows
          - CMDIFrameWnd
            - user MDI workspaces
          - CMiniFrameWnd
          - user SDI windows
          - COleIPFrameWnd
        - CSplitterWnd
      - **Control Bars**
        - CControlBar
          - CDialogBar
          - COleResizeBar
          - CReBar
          - CStatusBar
          - CToolBar
      - **Property Sheets**
        - CPropertySheet
          - CPropertySheetEx
      - **Dialog Boxes**
        - CDialog
          - CCommonDialog
            - CColorDialog
            - CFileDialog
            - CFindReplaceDialog
            - CFontDialog
            - COleDialog
              - COleBusyDialog
              - COleChangeIconDialog
              - COleChangeSourceDialog
              - COleConvertDialog
              - COleInsertDialog
              - COleLinksDialog
                - COleUpdateDialog
              - COlePasteSpecialDialog
              - COlePropertiesDialog
            - CPageSetupDialog
            - CPrintDialog
          - COlePropertyPage
          - CPropertyPage
            - CPropertyPageEX
          - user dialog boxes
      - **Views**
        - CView
          - CCtrlView
            - CEditView
            - CListView
            - CRichEditView
            - CTreeView
          - CScrollView
            - user scroll views
            - CFormView
              - user form views
              - CDaoRecordView
              - CHtmlView
              - COleDBRecordView
              - CRecordView
                - user record views
      - **Controls**
        - CAnimateCtrl
        - CButton
          - CBitmapButton
        - CComboBox
          - CComboBoxEx
        - CDateTimeCtrl
        - CEdit
        - CHeaderCtrl
        - CHotKeyCtrl
        - CIPAddressCtrl
        - CListBox
          - CCheckListBox
          - CDragListBox
        - CListCtrl
        - CMonthCalCtrl
        - COleControl
        - CProgressCtrl
        - CReBarCtrl
        - CRichEditCtrl
        - CScrollBar
        - CSliderCtrl
        - CSpinButtonCtrl
        - CStatic
        - CStatusBarCtrl
        - CTabCtrl
        - CToolBarCtrl
        - CToolTipCtrl
        - CTreeCtrl
  - **Graphical Drawing**
    - CDC
      - CClientDC
      - CMetaFileDC
      - CPaintDC
      - CWindowDC
  - **Control Support**
    - CDockState
    - CImageList
  - **Graphical Drawing Objects**
    - CGdiObject
      - CBitmap
      - CBrush
      - CFont
      - CPalette
      - CPen
      - CRgn
  - **Menus**
    - CMenu
  - **Command Line**
    - CCommandLineInfo
  - **ODBC Database Support**
    - CDatabase
    - CRecordset
      - user recordsets
    - CLongBinary
  - **DAO Database Support**
    - CDaoDatabase
    - CDaoQueryDef
    - CDaoRecordset
    - CDaoTableDef
    - CDaoWorkspace
  - **Synchronization**
    - CSyncObject
      - CCriticalSection
      - CEvent
      - CMutex
      - CSemaphore
  - **Windows Sockets**
    - CAsyncSocket
      - CSocket
  - **Arrays**
    - CArray (template)
    - CByteArray
    - CDWordArray
    - CObArray
    - CPtrArray
    - CStringArray
    - CUIntArray
    - CWordArray
    - arrays of user types
  - **Lists**
    - CList (template)
    - CPtrList
    - CObList
    - CStringList
    - lists of user types
  - **Maps**
    - CMap (template)
    - CMapWordToPtr
    - CMapPtrToWord
    - CMapPtrToPtr
    - CMapWordToOb
    - CMapStringToPtr
    - CMapStringToOb
    - CMapStringToString
    - maps of user types
  - **Internet Services**
    - CInternetSession
    - CInternetConnection
      - CFtpConnection
      - CGopherConnection
      - CHttpConnection
    - CFileFind
      - CFtpFileFind
      - CGopherFileFind
    - CGopherLocator

Classes Not Derived from CObject

- **Internet Server API**
  - CHtmlStream
  - CHttpFilter
  - CHttpFilterContext
  - CHttpServer
  - CHttpServerContext
- **Run-time Object Model Support**
  - CArchive
  - CDumpContext
  - CRuntimeClass
- **Simple Value Types**
  - CPoint
  - CRect
  - CSize
  - CString
  - CTime
  - CTimeSpan
- **Structures**
  - CCreateContext
  - CMemoryState
  - COleSafeArray
  - CPrintInfo
- **Support Classes**
  - CCmdUI
    - COleCmdUI
  - CDaoFieldExchange
  - CDataExchange
  - CDBVariant
  - CFieldExchange
  - COleDataObject
  - COleDispatchDriver
  - CPropExchange
  - CRectTracker
  - CWaitCursor
- **Typed Template Collections**
  - CTypedPtrArray
  - CTypedPtrList
  - CTypedPtrMap
- **OLE Type Wrappers**
  - CFontHolder
  - CPictureHolder
- **OLE Automation Types**
  - COleCurrency
  - COleDateTime
  - COleDateTimeSpan
  - COleVariant
- **Synchronization**
  - CMultiLock
  - CSingleLock

Layers of UI Software

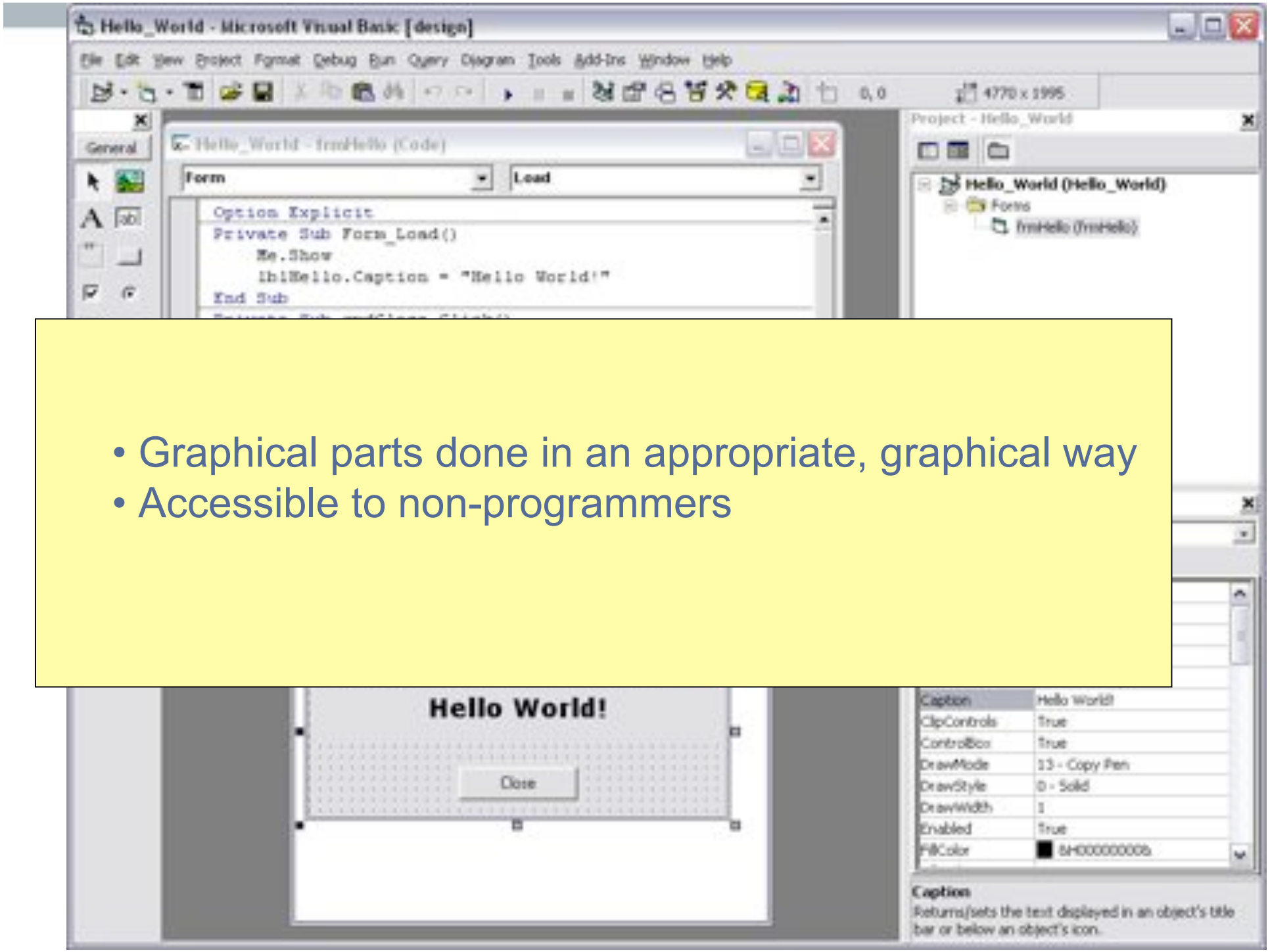


Higher Level Tools

- Toolkits hard to use, higher-level support is helpful
  - Graphical layout tools
  - Higher-level frameworks
  - Older tools called “User Interface Management Systems”
- Successful research ⇒ industry

Graphical / Interactive Tools

- Create parts of UI by visually laying out components
  - Examples: Menulay (1983), Trillium (1986), Jean-Marie Hullot from INRIA to NeXT
  - Now: "**Interface Builders"**, Visual Basic's layout editor, resource editors, "constructors"



- Graphical parts done in an appropriate, graphical way
- Accessible to non-programmers

Component Architectures

- Example of framework at higher level than widgets

- Create apps from loosely coupled *components* which are separately developed and compiled
  - Example: drawing component handles picture inside a document
  - Example: embed a web browser in your app

- Invented by Andrew research project at CMU (1988)
- Old: OLE, OpenDoc, Visual Basic Controls (VBX Controls), ActiveX, CORBA
- Current: COM, Java Beans

Observation #1

- Many common themes to what we discussed today

- Lower barriers to entry
  - Really hard to program GUIs, create a framework to simplify
  - Only programmers can create, create interface builders
- Increase expressiveness
- Raise levels of abstraction
  - More examples of this next class
- Raise level of complexity that can be managed
  - Components, re-use of code, frameworks

Observation #2

- Evolution of web highly similar to what we just described
  - Lots of Javascript / AJAX Toolkits coming out
    - Yahoo UI, Dojo, Rico, Prototype, …
  - Web "components" coming out too
    - Trivial to embed YouTube video on your web page
    - Trivial to embed Google map
    - What's next? Embed Google office?
      - Live spreadsheets? Live graphs?
      - Connect their events together via GUI editor?

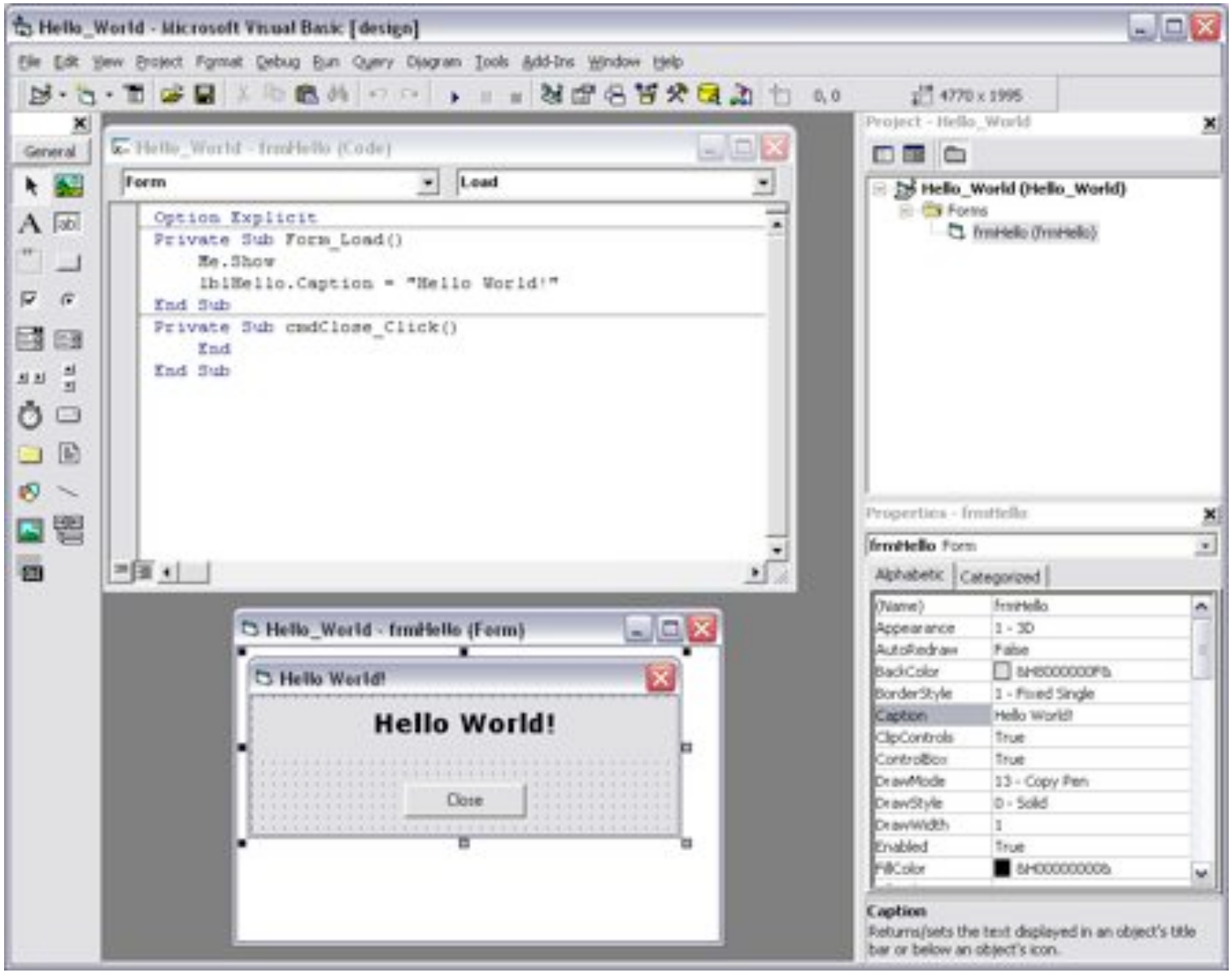
- Could be room for interesting final project here
  - Take an old idea from GUI world and apply to Web
  - Make it easy to create highly attractive and usable site

Subtle Influence of Tools

- Be aware of the path of least resistance
- Tools have Whorfian effects
  - Change the way you think
  - Change what is possible
  - ➔Change what you design

Subtle Influence of Tools

Subtle Influence of Tools



Summary

- High-level overview of how user interfaces work
- Dataflow perspective
  - Widgets
  - Component Tree
  - Events
- Layered perspective
- Rest of course will be the details

Java Swing

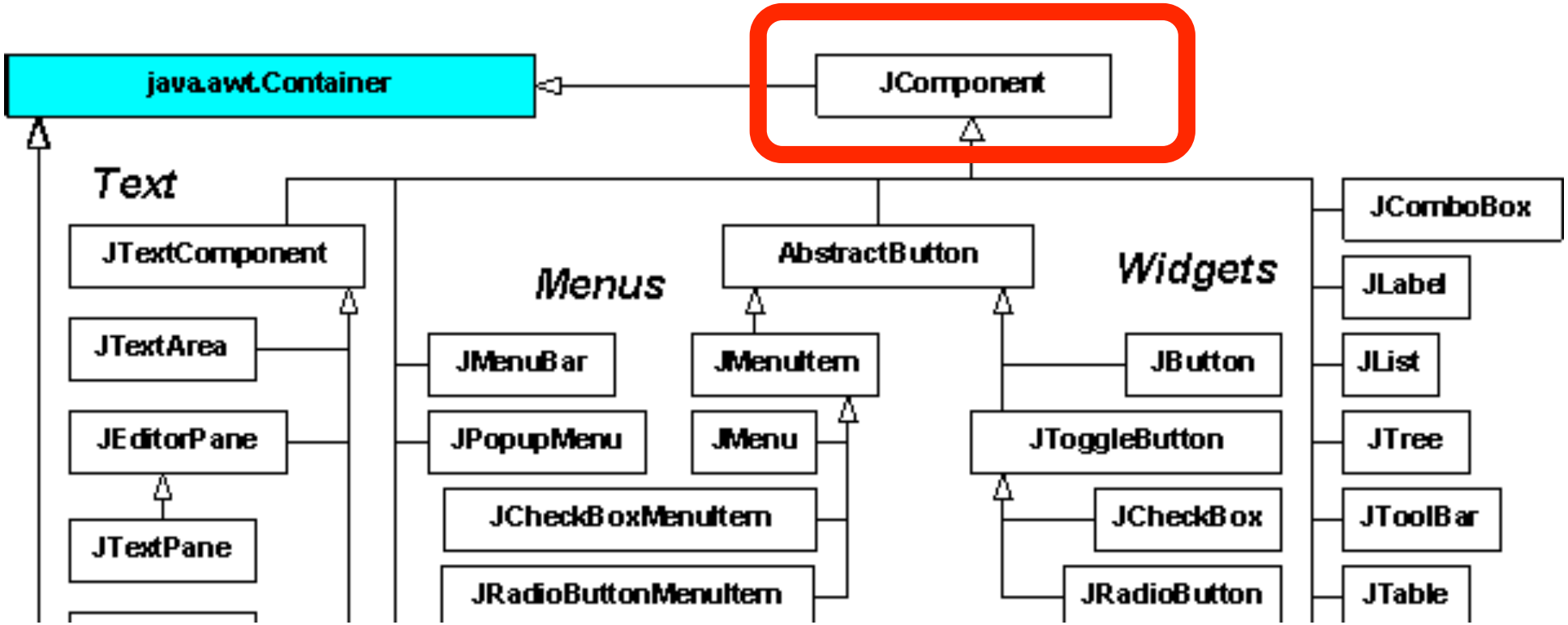
(1 minute break)

Java Swing

- Caveat: Swing is a decent toolkit but not organized extremely well, and is messy in some places
  - Big and complicated, “easy” things not always easy
  - (Re)designed by a lot of people
  - Has to work with/within old AWT toolkit which was very badly designed (6 weeks!)
- Used to be only commercially viable toolkit in Java
  - Until IBM’s Standard Widget Toolkit (SWT)
  - SWT is like a much better AWT
- Will go over this again at Friday’s tutorial

Standard object-oriented approach

- Most functions of an interactive object encapsulated in base class JComponent
  - (& AWT super classes above it)
  - Swing interactive objects are all subclasses of this



JComponent defines methods for:

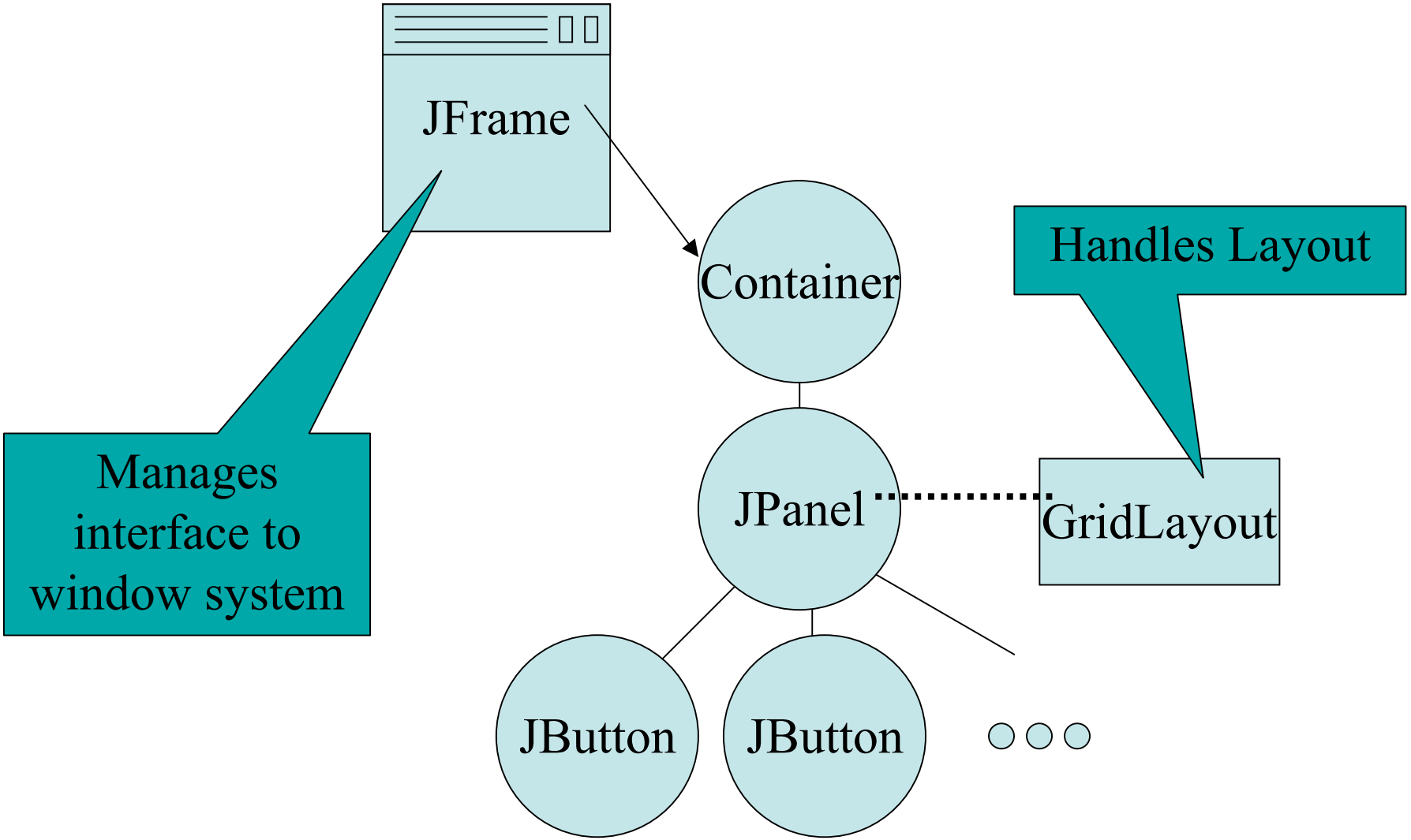
- Each JComponent has methods for:
  - Hierarchy management
  - Geometry management
  - Object status / info management
  - Layout
  - (Re)drawing
  - Damage management
  - Picking
  - Input
  - Actions
  - Localization / internationalization
  - …

Lots of Different Components

- Top-level containers
  - Windows
- General containers
- Basic controls
  - Buttons, checkboxes, etc
- Uneditable Information Displays
- Interactive Information Displays

http://java.sun.com/docs/books/tutorial/uiswing/components/components.html

Swing UIs are a Tree of Components



Hierarchy Management

- JFrame (& super class) API provides methods for tree manipulation
  - `add(), getComponent (), getComponentCount(), getParent(), remove(), removeAll()`, etc...
- Debugging hint: if nothing shows up on the screen
  - check that you added it to the tree
  - check that you added it to the right parent

Geometry Management

- Every component maintains its own geometry
  - E.g., bounding box: `getX(), getY(), getWidth(), getHeight(), getBounds()`
    - x,y is relative to parent
    - i.e., 0,0 is at parent's top-left corner
  - Drawing is relative to top-left corner
    - Each component has own coord system

Object status / information

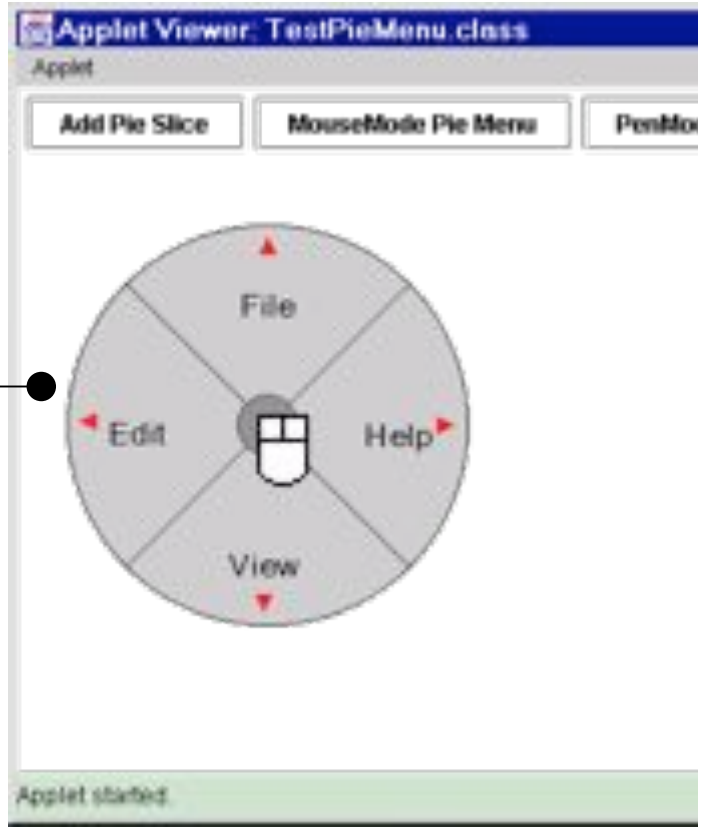
- Each component maintains information about its state
  - `setVisible(), setEnabled()`
- Each component instance keeps its application info
  - `getClientProperty(), putClientProperty()`

Each object handles:

- Layout (later in course…)
- Drawing
  - Each object knows how to (re)create its appearance based on its current state
  - `paint()` `// don't override this`
    - `paintComponent()`
    - `paintBorder()`
    - `paintChildren()`

Each object handles:

- Damage management
  - Tell the system that something about your internal state has changed and your image may not be correct
  - repaint(), revalidate()
- Picking
  - See if a point is “inside” or “outside”
  - contains(x,y)
  - even works for nonlinear widgets



Other parts

- Input (will talk about later…)
- Actions & Application interface
  - Done in terms of sending events to “listeners”
    - Register as a listener to get notifications of when things you are interested in happen
    - P1 - MouseListener

Lots of parts, but…

- … is (mostly) understandable in terms of major tasks we have laid out

- Only have to implement the specialized parts
  - E.g., `paint()`

Let’s build an interface...

```
package Demo631;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Demo1 implements ActionListener {
    public static void main(String[] args) {…}
    public JComponent buildUI() {…}
    public void actionPerformed(ActionEvent e) {…}
}
```

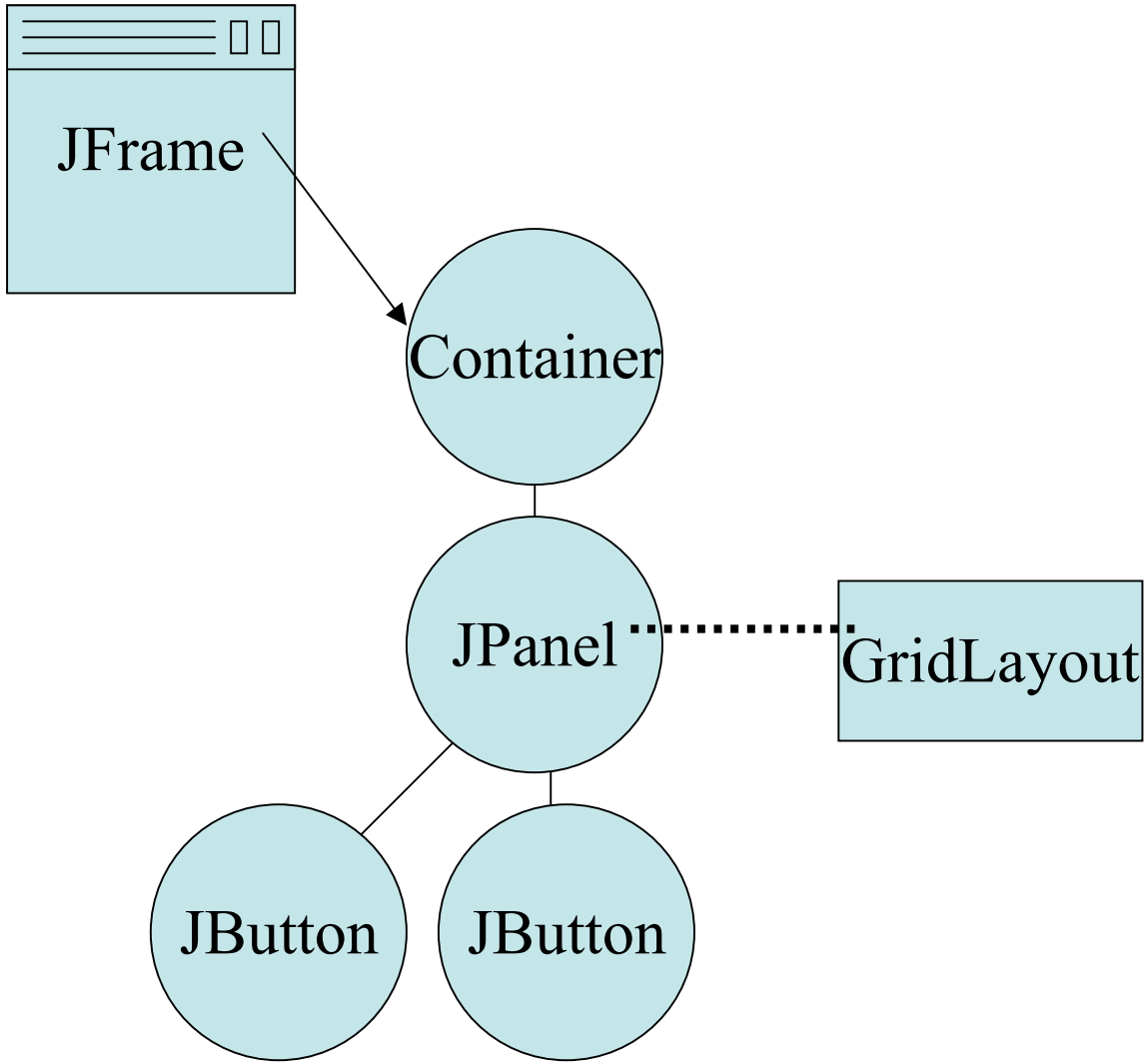
```
public static void main(String[] args)
{
      // instantiate an object of this class
      Demo1 app = new Demo1();

      // create a top level frame and put an interface in it
      JFrame frame = new JFrame("Demo1");
      frame.getContentPane().add(app.buildUI(),

   BorderLayout.CENTER);

      // arrange for it to close, then do layout and make visible
      frame.setDefaultCloseOperation(

   JFrame.EXIT_ON_CLOSE);
      frame.pack();
      frame.setVisible(true);
}
```

```
public JComponent buildUI()
{
      // top level container laid out as a column
      JPanel pane = new JPanel(new GridLayout(0,1));

      // create a button and send its action events to us
      JButton b1 = new JButton("A Button!");
      b1.addActionListener(this);

      // install the button as a child of the pane
      pane.add(b1);
      ...
      return pane;
  }
```

```
// this gets called when we are notified
// of an ActionEvent.  we asked for this via the
// addActionListener() calls above
public void actionPerformed(ActionEvent e)
{
    System.out.println("Action:" +
                                e.getActionCommand());
}
```

What did we build?

What did we build?



An improved way to handle actions

- As shown, action response is separated from component causing it
- Also normally have to have selection logic to pick out which button, etc.
- Can use anonymous inner classes to improve these two things

```
// create a button
JButton b1 = new JButton("A Button!");

// put the response code here using an
// anonymous inner class
b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Top button was pressed...");
    }
});

// install the button as a child of the pane
pane.add(b1);
```

```
// create a button
JButton b1 = new JButton("A Button!");

// put the response code here using an
// anonymous inner class
b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("Top button was pressed...");
    }
});

// install the button as a child of the pane
pane.add(b1);
```

Anonymous Inner Class

```
new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        …
    }
}
```

- Creates an instance of local unnamed subclass of ActionListener() which has actionPerformed() method overridden

Summary

- Very high-level overview of how user interfaces work
  - Widgets
  - Component tree
  - Events
- Rest of course

- Java Tutorial
  - http://java.sun.com/docs/books/tutorial/uiswing/index.html