

CSCI0320

Introduction to Software Engineering

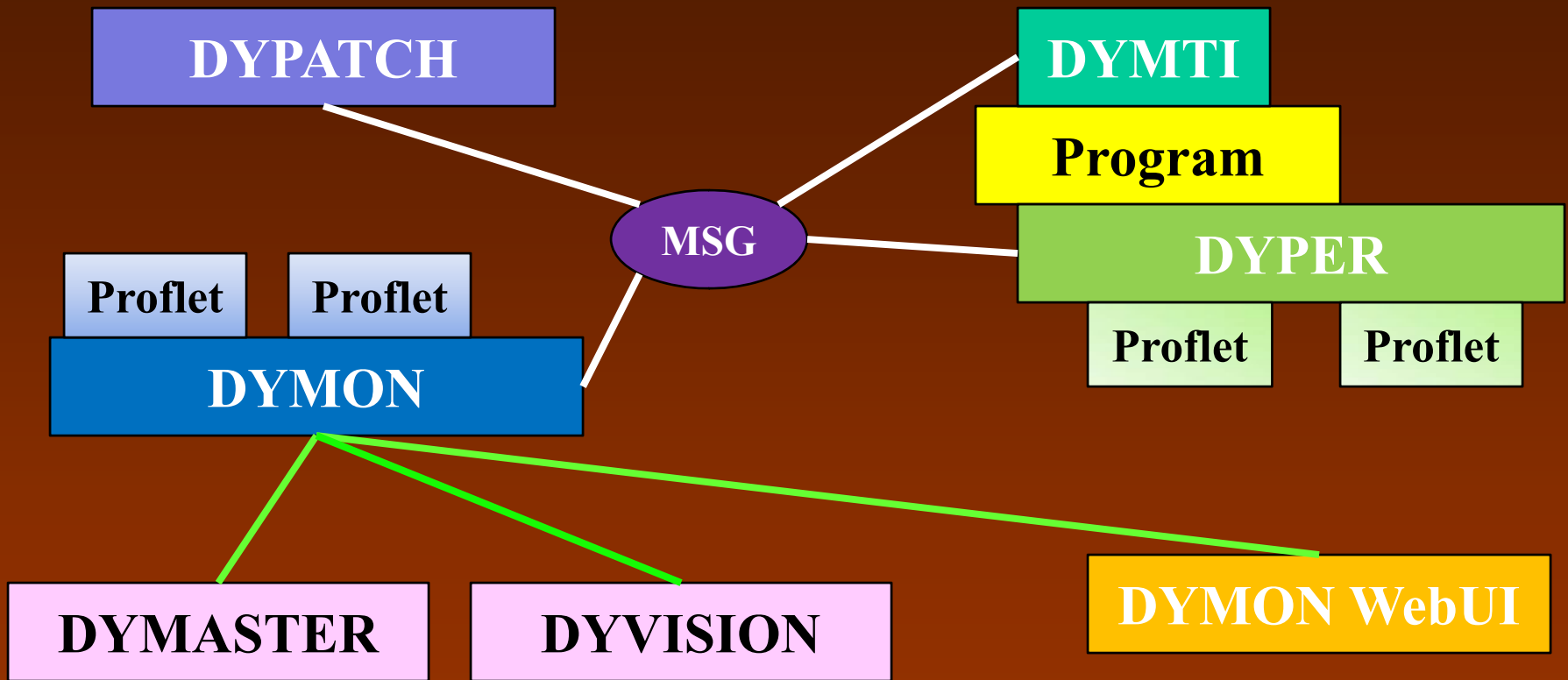
Lecture 10

Design and User Interfaces

Concrete Example

- **DYVISE**
 - This is a performance monitoring tool designed to work with long-running server systems that need to be monitored while in production mode.
 - Variety of different performance metrics
 - Extensible
 - Demand-driven
- **DEMO**

Basic Architecture



Client-Server

- **DYMON acts as a server**
 - **DYMASTER is a client**
 - **DYVISION is a client**
 - **Web interface is a client**
 - **Well defined message interface**
 - **<Command> <args...>**

Publish-Subscribe

- **External publish subscribe mechanism**
 - Using XML-based messages (MSG)
 - Clients register patterns with MSG
 - With callbacks
 - Clients send messages to MSG
 - These get forwarded to all matching handlers
 - Handlers can send a reply to the message
- **Clients**
 - DYMOM, DYMTI, DYPER, DYPATCH
 - All communications between these
- **Well-defined interface**
 - Show definition file

Look Inside DYMON

CONTROL

ATTACH

SERVER

PROCESS

**PATCH
TRACKER**

AGENT

DETAILING

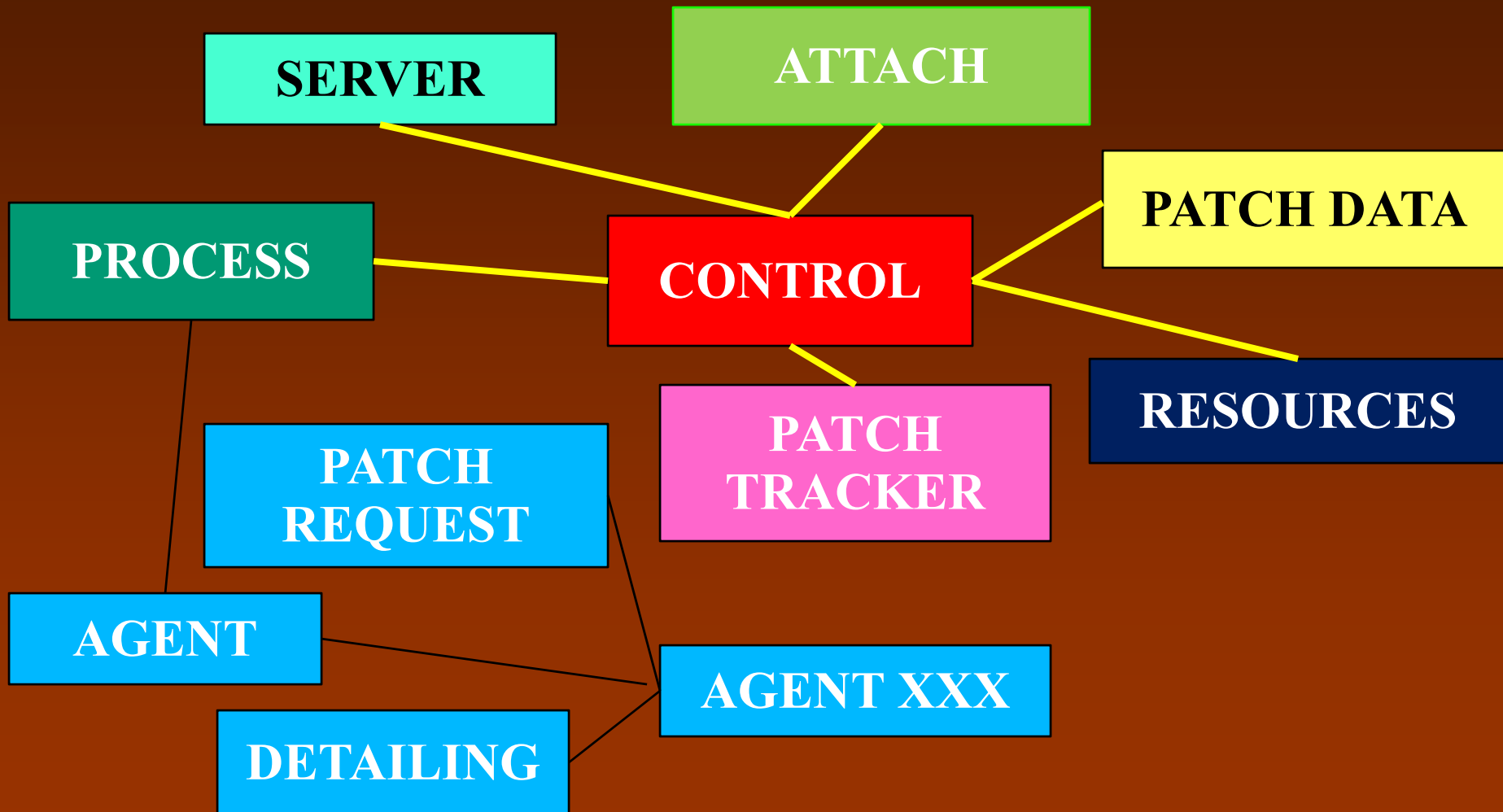
PATCH DATA

**PATCH
REQUEST**

RESOURCES

AGENT XXX

CORE + EXTENSIONS



Separation of Concerns

- **Want to have multiple agents**
 - **Each independent**
 - **Easy to add**
- **System is designed to make this doable**
 - **Abstract interface to an agent**
 - **Generic support classes for the agent**
 - **Multiple agents can then be added**

Simplicity

- **The interfaces are kept simple**
 - Few messages actually handled (< 10)
 - Agent interface is 5 or so methods
- **Complexity is hidden**
 - Scheduling algorithm is inside Process
 - Actual code patching is inside DYPATCH

Iterative Design

- **This is not the first design**
 - Detailing and patch request were split
 - Patch data was added as support
 - Attach and server were split from control
- **This is not the final design**
 - Process is too complex and should be divided
 - Patch Tracker and process are too intertwined

Interface-Based Design

- **DYMON is too simple for this**
 - Single package, single class
- **However remember the search tool**
 - Look for code that meets some test cases

Starting Interfaces

S6Request

S6SolutionSet

S6Factory

S6Solution

S6KeySearch

S6Engine

S6Fragment

S6Transform

S6Language

Implementation

- **Show S6 common directory**
- **Note Core+Extensions design again**

Extensions not Pure

- **Data-access objects passed around**
 - **S6Request**
 - **S6Solution**
- **These provide access methods**
 - **Get information**
- **Creation and manipulation still thru core**
 - **Engine creates requests**
 - **Engine creates solutions**
- **Dual interfaces for these objects**
 - **One for use in the core**
 - **One for use by other components**

User Interfaces

- **Are complex**
 - Often 50-80% of the code of a system
 - Code can be quite intricate
 - Hardware dependent, OS dependent
- **Are important**
 - Make or break a system
- **Are often done wrong**
 - Examples of bad interfaces
 - Examples of good interfaces

User Interface Design

- **Steps**
 - **Understand** the problem
 - **Draw pictures** of the interface
 - Experiment
 - **Understand** the toolkit
 - **Write a prototype**
 - Experiment
 - **Test and rewrite**

Understand The Problem

- **Know the user(s)**
 - Level of user
 - What they are used to, what they expect
 - What you can assume they know or don't
 - Accessibility (Target suit)
 - Internationalization
- **Know the problem domain**
 - Vocabulary, what is important, what to stress
- **Know the context**
 - Frequency of use of each command/component
 - Common versus uncommon tasks
 - Consequences of mistakes
 - Constraints (screen space, interaction time, ...)

Draw Pictures

- **Pictures of the interface**
 - General overview of possible layouts
 - Pictures telling a story (scenario/use case)
 - Sequence showing interactions
- **Work with users to get these right**
 - Iterate, try different approaches, etc.
- **Integrate multiple stories**
 - In a consistent way
 - Taking into account priorities, frequency
- **Get a good understanding of the UI**

Understand the Toolkit

- Know what is **easy/hard**, **doable/undoable**
 - Dual sliders in swing
 - Editable combo box in html
- Depends on target platform
 - Swing for Java
 - Applets, HTML, flex, ... for browser
 - GWT or dynamic HTML (ajax)
 - Qt for C++
- Know best way to use it

Prototype

- **Map the picture to widgets**
 - Pictures with explicit widgets
 - Don't invent new widgets
 - Think of screen and window sizes
- **Implement something early**
 - Without the rest of the application
 - Don't be afraid to rewrite it
 - Multiple times
 - Evaluate it early and continually
 - With real users if possible
- **Treat it as a test case for the system**

User Interface Toolkits

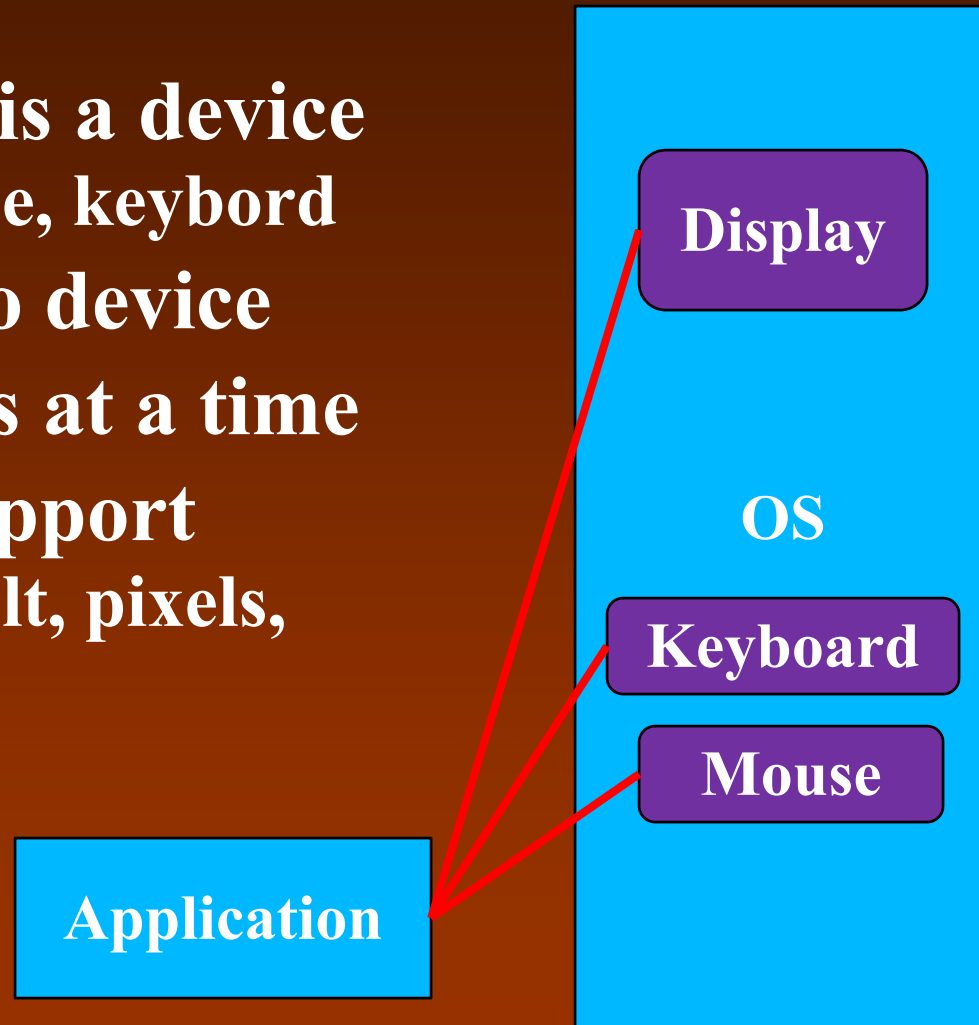
- **How do user interfaces work?**
 - **What happens when the user types or clicks?**
 - At the system level
 - **How should a user interface interact?**
 - With the user
 - With the application
 - **What type of user interface should I use?**
 - Swing, web-based
 - Inside or outside application
- **What do you need to know to build UIs**

Early User Interfaces

- **Command line interfaces**
 - Seem primitive today
 - Still very useful for composable tools
 - Good for scripting
 - Command languages

Early Graphical Interfaces

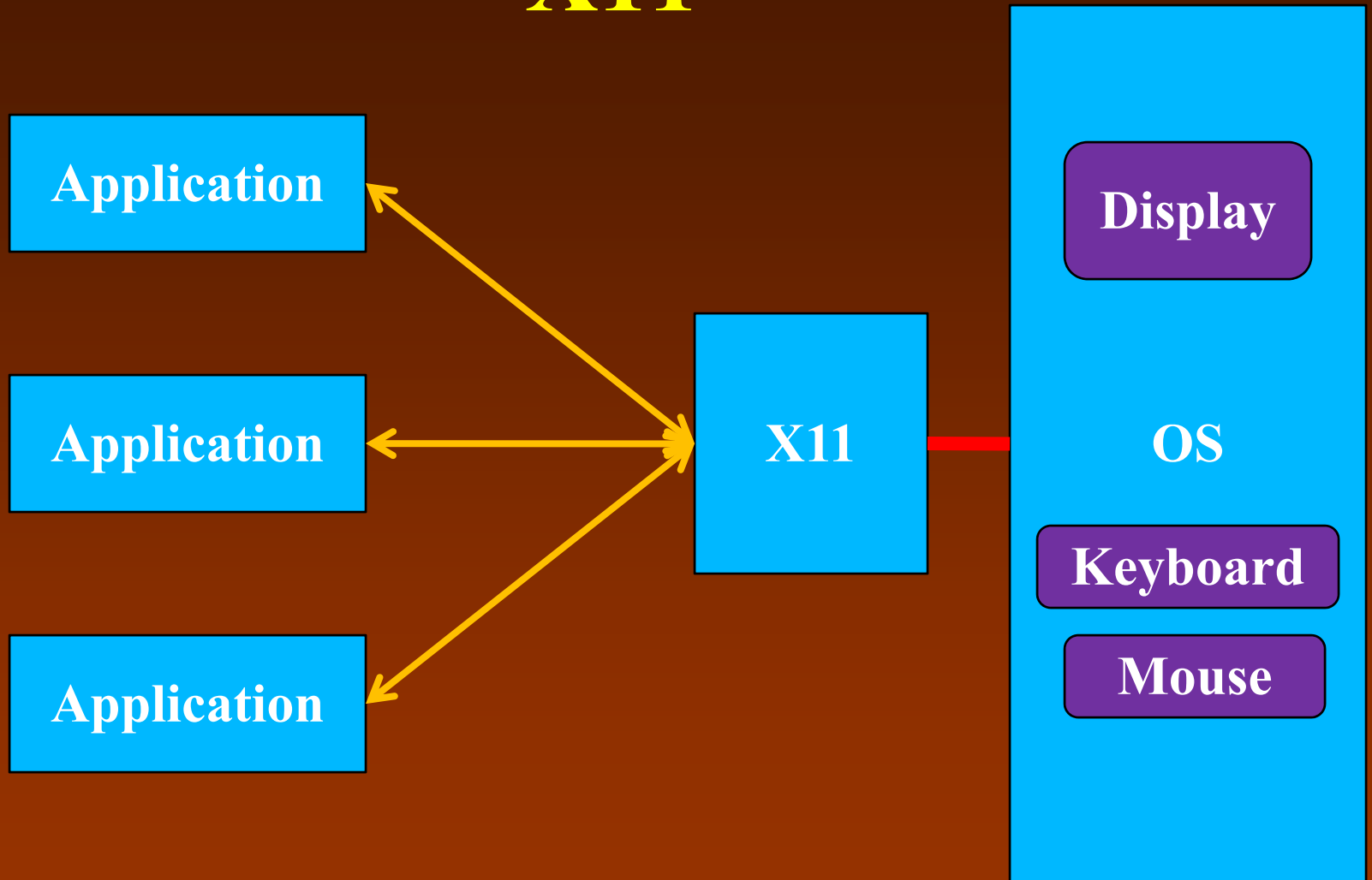
- **The screen is a device**
 - As is mouse, keyboard
- **App talks to device**
- **One process at a time**
- **Minimal support**
 - Lines, bitblt, pixels, characters



Windowing Systems

- Provide software support for windows
- Early systems
 - **Smalltalk, Suntools**
 - Can still address the whole display
 - Process must cooperate
 - Library tried to limit things, not always correctly
 - Additional components for drawing
- **Current** windowing systems
 - Separate graphics functionality
 - Provide a display/window **abstractions**
 - Provide additional drawing **primitives**
 - Images, fills, polygons, transforms

X11



Windowing Variations

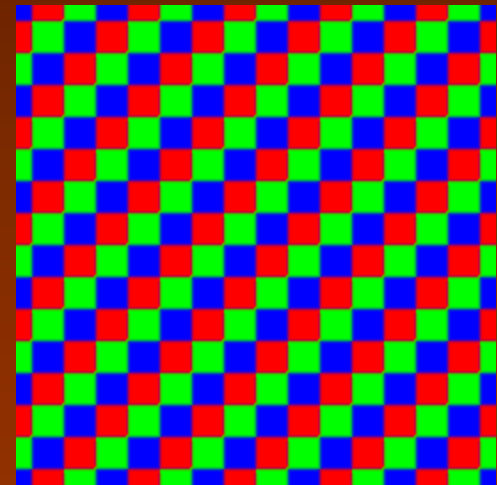
- **Sun's NeWS**
 - **Postscript-based programming language**
 - User sent postscript to the display
 - Display would execute postscript programs
 - **Language included events**
 - Constructs for waiting for/processing events
- **Display Postscript (NeXT)**
 - **Similar, slightly different abstraction**
- **Mac OS/X (Cocoa, carbon)**
 - **Display PDF**

User Interface Toolkits

- **Library on top of the windowing system**
 - **Object-oriented** view of the interface
 - Provide an **interaction model**
 - Provide a **drawing model**
- **Many tool kits available**
 - **Java: AWT, Swing, SWT**
 - **C++: Qt**
 - **Windows: MFC, WPF**
 - **Unix: Xt, Motif**
 - **Mac: Cocoa, Carbon**
 - **Others: Tcl/Tk**
- **Web interface thru HTML is similar**
 - **GWT**

Drawing Basics

- **Device Coordinates -- Pixels**
 - What actually exists on the display
 - 0,0 in upper left (Why?)
 - Pixels can be 1,8,24,32 bits
- **Provide finest control**
 - Make the UI look exact
- **Most toolsets**
 - Provide access to this level



Coordinate-Based Drawing

- **2D Coordinate spaces**
 - User can define center and span
 - Floating point coordinates
 - How to map a coordinate to pixels
 - Anti-aliasing
 - Mapped to pixels
 - Using a linear transformation
 - Java2D
- **Issues**
 - Scaling
 - Text

Linear Transformations

- **Matrix-vector multiplication**

$$\begin{bmatrix} X & Y & 1 \end{bmatrix} * \begin{bmatrix} A & B & 0 \\ C & D & 0 \\ E & F & 1 \end{bmatrix} = \begin{bmatrix} XA+YC+E & XB+YD+F & 1 \end{bmatrix}$$

- **Used to map coordinates to pixels**
 - Original (x,y) are the coordinates
 - Resultant (x,y) are the pixels

Translation

- If matrix is

1	0	0
0	1	0
U	V	1

- What happens to (x,y)
 - $(x+u,y+v)$
- Moved everything by (u,v)
- If you want to set the origin in the middle

Scaling

- If matrix is

A	0	0
0	B	0
0	0	1

- What happens to (x,y)
 - (Ax,By)
- Everything is scaled
- Used to map $(0-1,0-1)$ to arbitrary window
- What happens if B is negative?
 - Reflections
 - Map y coordinate so up is positive

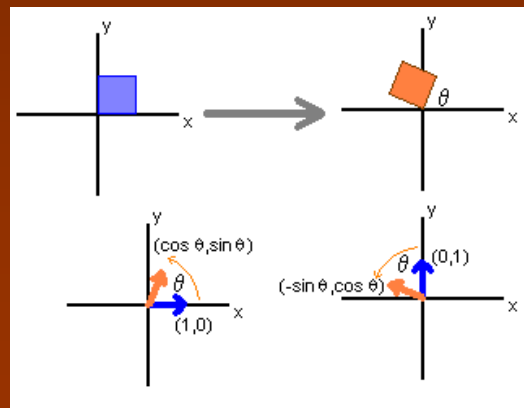
Rotation

- If matrix is

$\cos \theta$	$-\sin \theta$	0
$\sin \theta$	$\cos \theta$	0
0	0	1

– What happens to (x,y)

- $(x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta)$
- This is a rotation around the origin of θ degrees



Matrices Compose

- Can do multiple operations at once
 - Multiply the corresponding matrices
 - Consider 200x200 pixel space
 - Want origin in middle, -1 to +1 in x and y
 - Want y going up

100	0	0
0	100	0
0	0	1

1	0	0
0	-1	0
0	0	1

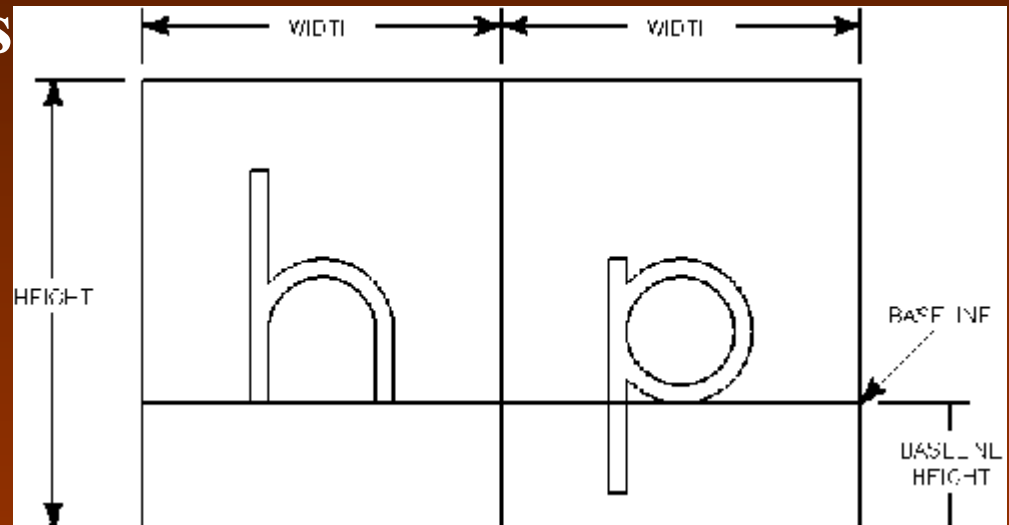
1	0	0
0	1	0
100	100	1

100	0	0
0	-100	0
100	100	1

- Map
 - $(0,0) \Rightarrow (100,100)$, $(-1,-1) \Rightarrow (0,200)$
 - $(1,1) \Rightarrow (200,0)$

Drawing Elements

- **Fonts**
 - Java Font class
 - Define characters
 - Baseline
 - Height
 - Offset
 - Size



Drawing Elements

- **Images**
 - Off-screen pictures
 - Used for icons
 - Can be colored in the image
 - Or colored at drawing time
 - Depth options
- Can be user-created as well as file-based
 - Draw into offscreen bitmap or pixmap

Drawing Elements

- **Colors**
 - **Red-green-blue (RGB)** color model
 - Standard for displays
 - 0-255 for each
 - #rrggbb format
 - 0-1 for each
 - **RGBA** color model
 - A value is transparency
 - Blended with what was there before
 - **HSV** color model
 - Useful for information visualization
 - Many other models as well

Layout Management

- **Control the assembly in a composite**
 - Handle window size changes
- **Simple layouts**
 - Grids, sequential, box
- **Complex layouts**
 - Gridbag layout
 - Spring layout
- **Browser: tables, style-sheets**
- **Most difficult part of setting up a UI**
 - Lots of lines of code to do very little
 - Especially handling resize

User Interface Programming

- **Inverted control structure**
 - Normally your program has control
 - In a UI application, the UI has control
 - Calls your program as needed
 - Takes a little getting used to
- **Two phase process**
 - Set up the interface by defining widgets
 - Register callbacks for actions
 - Start the user interface
 - Handle events as callbacks

Callback Handling

- **Callback method passed Event object**
 - User registers callback object
- **How to handle multiple events**
 - Multiple callback objects
 - Callback object looks at event
- **Callback hints**
 - **Avoid anonymous classes** (used nested ones)
 - Callbacks should **do very little work**
 - Use **modal dialogs** if more input needed
 - Always let user abort the operation
 - **Swing uses its own thread (but only one)**

Drag and Drop

- **Basic Concept**
 - **User clicks** on one item
 - **Drags** the mouse to another item
 - Then **lets go**
- **Actual implementation**
 - Is quite messy
 - Involves a number of classes
 - Is difficult to get right
- **What are the complexities?**

Drag and Drop Issues

- **Click Down**
 - What is the “item”
 - Can the item be dragged
 - What is the actual thing being moved
 - Might depend on where it is dropped
 - Might be in different application
- **Drag**
 - What should the cursor be
 - Should it change over a drop site
 - Should the target change when drag over
 - Drag between processes

Drag and Drop Issues

- **Letting Go**
 - Is the drop site valid for this item
 - How to interpret the data based on target
 - What if the target isn't valid
 - Handling copy versus move
 - Drag site might need to take action
- **How does Java handle all of this?**

Java Drag and Drop

- **Controller** class
- Registering **sources**
- Registering **sinks**
- Handling drag and drop
 - **Representing** the data
 - **Negotiation** of data between source and sink
 - **Callbacks** for source and sinks