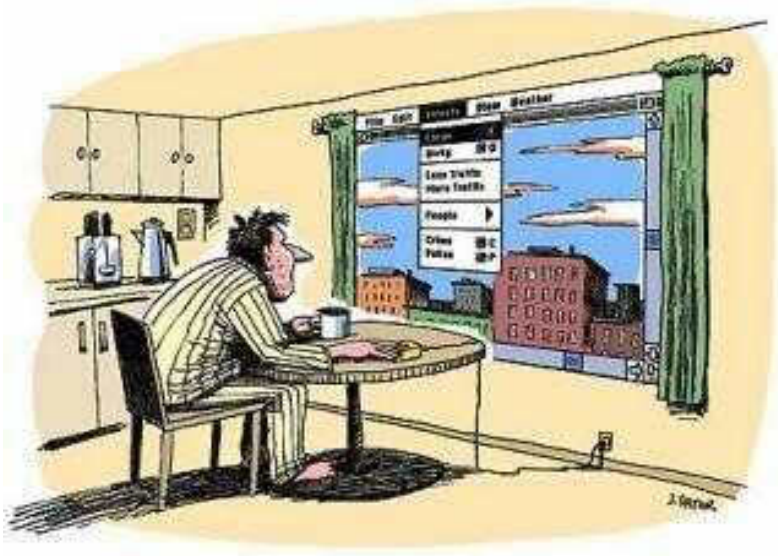
Graphical User Interface Technology

Credits:
Stanford HCI,
Pressman



Instructor: Peter Baumann

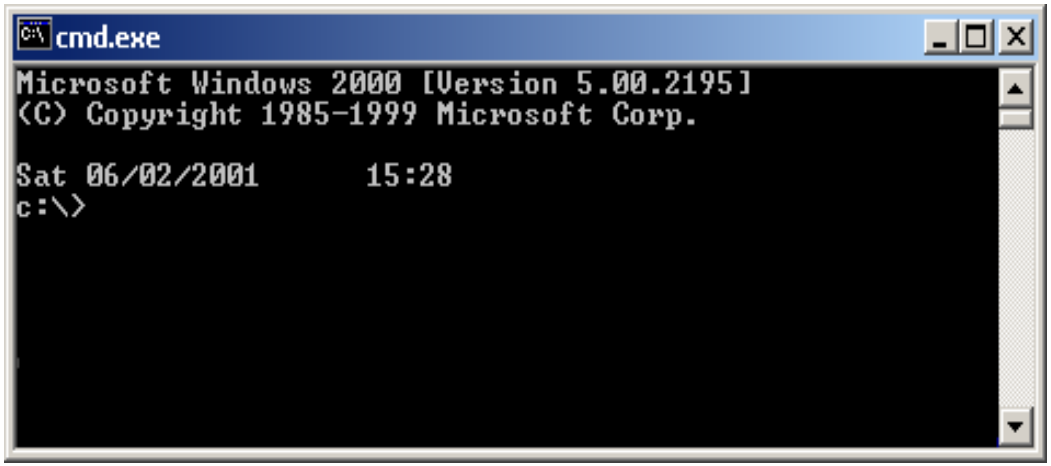
email: p.baumann@jacobs-university.de

tel: -3178

office: room 88, Research 1

Sequential Programs

- Program takes control, prompts for input
  - command-line prompts (DOS, UNIX)

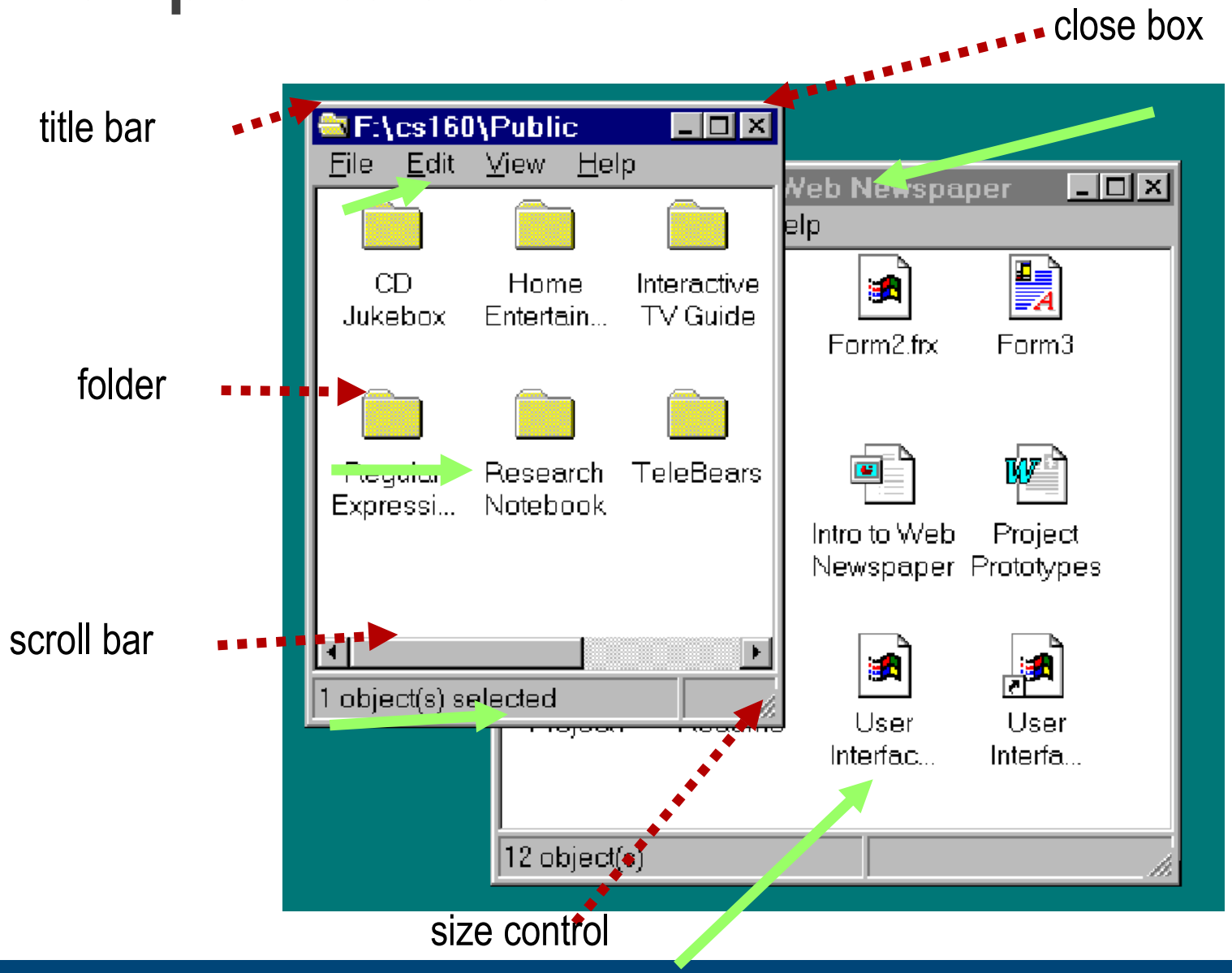


- user waits on the program
  - program tells user it's ready for more input
  - user enters more input

Sequential Programs (cont.)

- General flow of interaction
  - *Prompt user for input*
  - *Program reads in a line of text*
  - *Program runs for a while (user waits)*
  - *Maybe some output*
  - *Loop back to beginning*
- But how do you model the many actions a user can take?
  - for example, a word processor?
  - printing, editing, inserting, whenever user wants
  - sequential doesn't work as well for graphical and for highly-interactive apps

Example Interactions



GUI History

- forms generator source:
  - 1985
  - From memory, not exact

```
+----------------------------------------+
|                                        |
|   Central Donation Form                |
|                                        |
|   Your name:     $$$$$$$$$$$$$$$$$     |
|   Bank account: ############           |
|                                        |
|   Your donation: ########.## EUR       |
|                                        |
+----------------------------------------+
$ char
# numeric

comment plausi checks:
@1 not empty
@2 >0
@3 >1.0
```

Modern GUI Systems

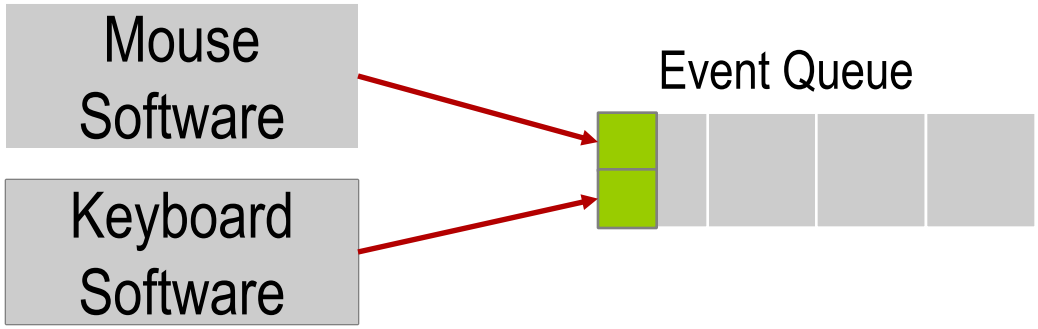
- Three concepts:
  - Event-driven programming
  - Widgets
  - Interactor Tree
- Describes how most GUIs work
  - Closest to Java
  - But similar to Windows, Mac, Palm Pilot, ...

Event-Driven Programming

- Instead of the user waiting on program, program waits on the user
- All communication from user to computer is done via “events”
  - “mouse button went down”
  - “item is being dragged”
  - “keyboard button was hit”
- Events have:
  - type of event
  - mouse position or character key + modifiers
  - *...plus possible additional, application-dependent information*

Event-Driven Programming

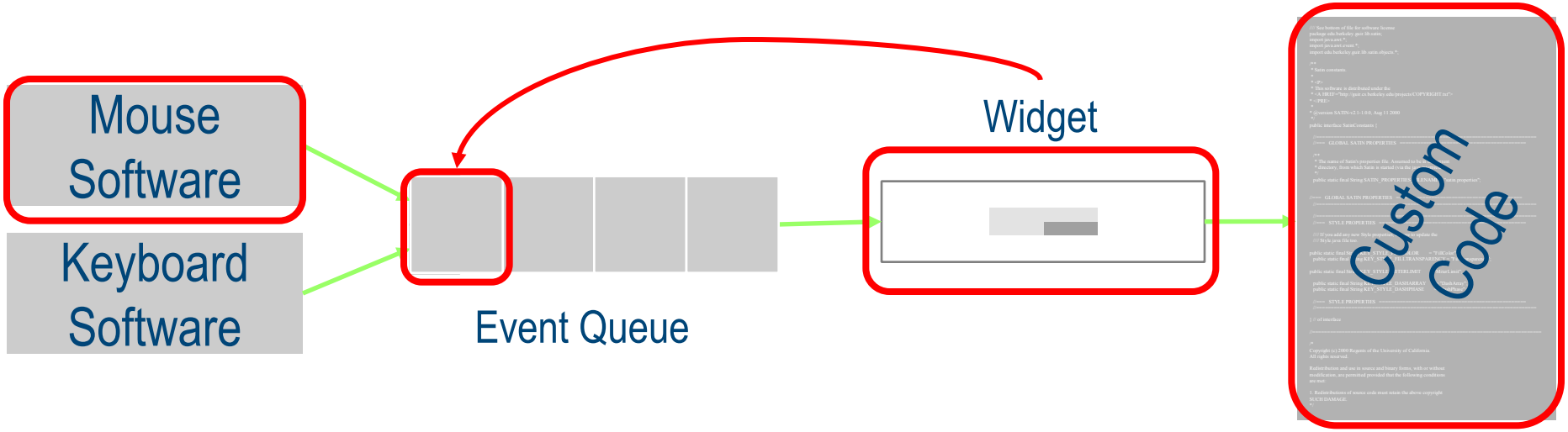
- All events generated go to a *single* event queue
  - provided by operating system
  - ensures that events are handled in the order they occurred
  - hides specifics of input from apps



Widgets

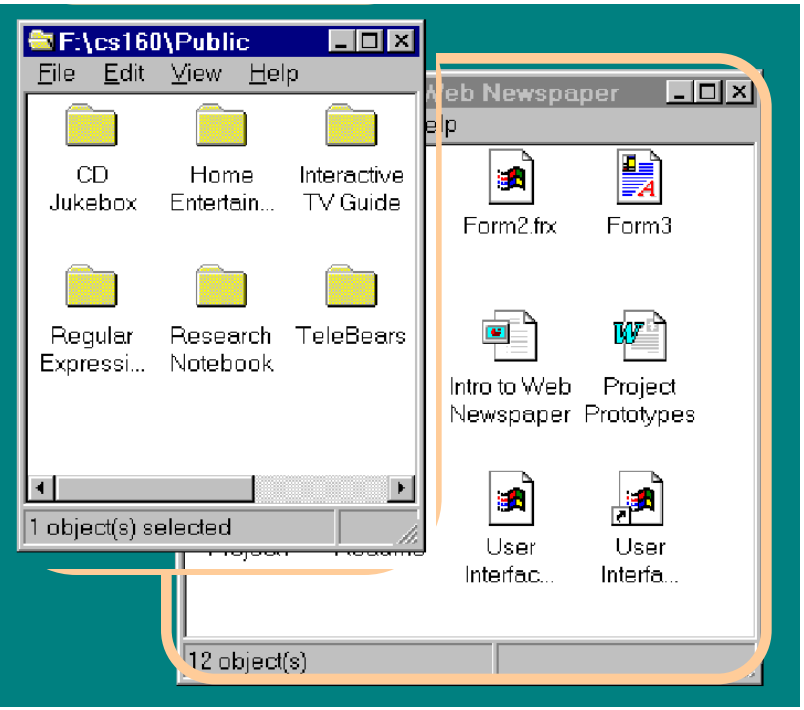
- Widget = (reusable) interactive object
  - “window gadget”
- Widget tasks:
- Handle certain events:
  - widgets say what events they are interested in
  - event queue sends events to the “right” widget
- Update appearance
  - e.g. button up / button down
- Generate some new events, eg
  - “button pressed”
  - “window closing”
  - “text changed”
- But these events are sent to interested listeners instead
  - custom code goes there

Widget in Action



Interactor Tree

- Decompose interactive objects into a tree

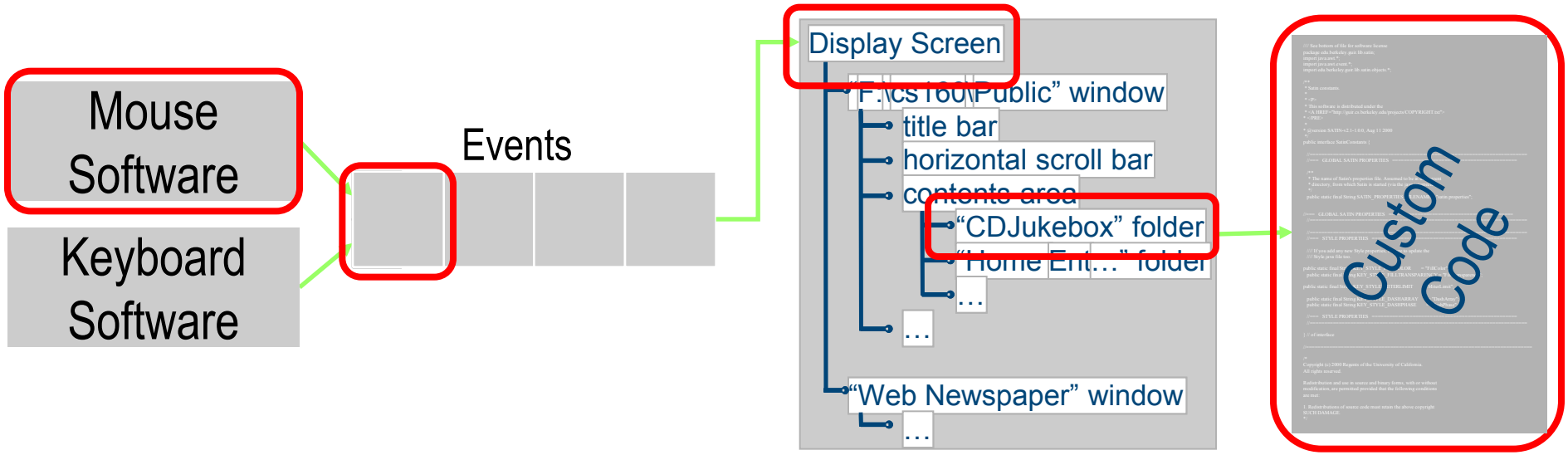


Display Screen

- "F:\cs160\Public" window
  - title bar
  - horizontal scroll bar
  - contents area
    - "CDJukebox" folder
    - "Home Ent…" folder
    - …
  - …
- "Web Newspaper" window
  - …

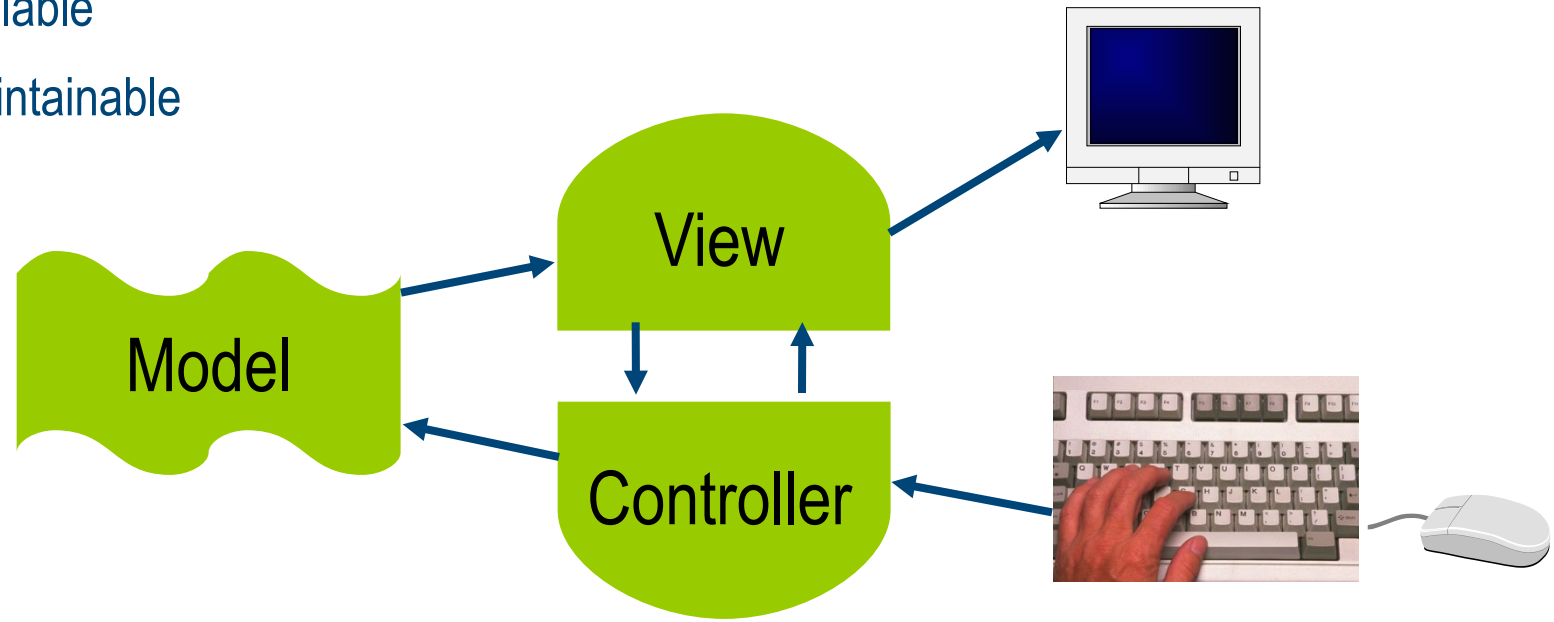
Main Event Loop

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

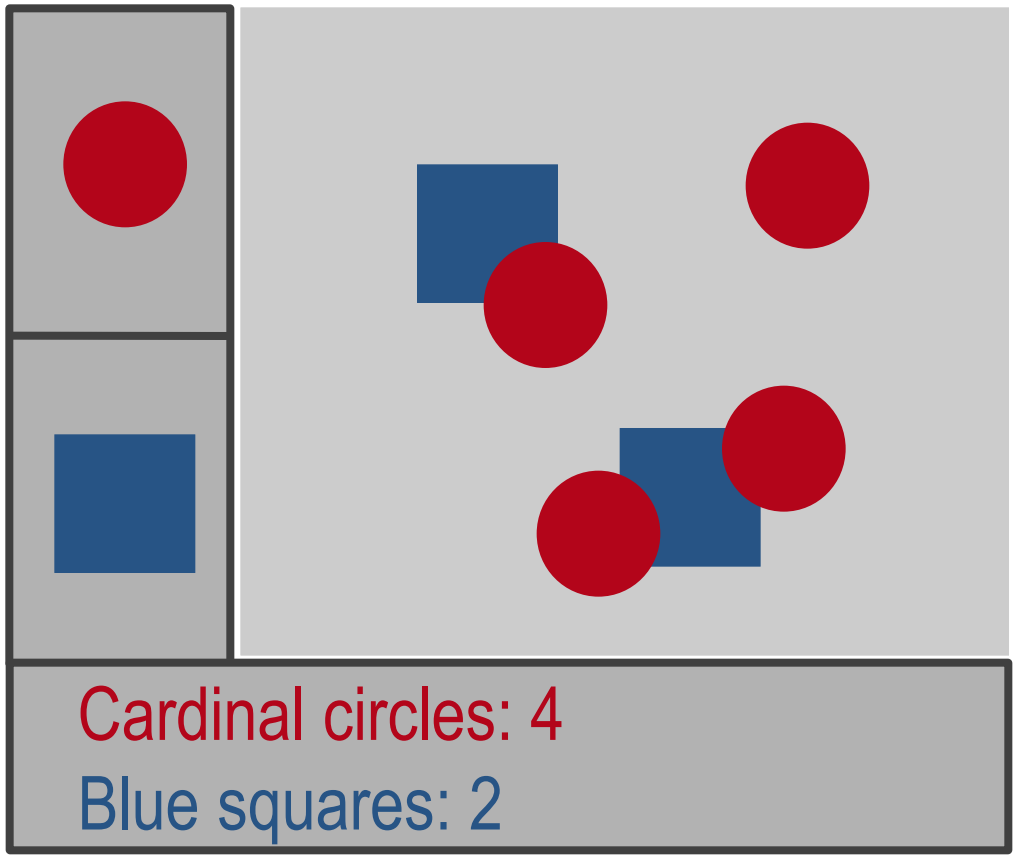


Model-View-Controller

- Architecture for interactive apps
  - introduced by Smalltalk developers at PARC
- Partitions application in a way that is
  - scalable
  - maintainable

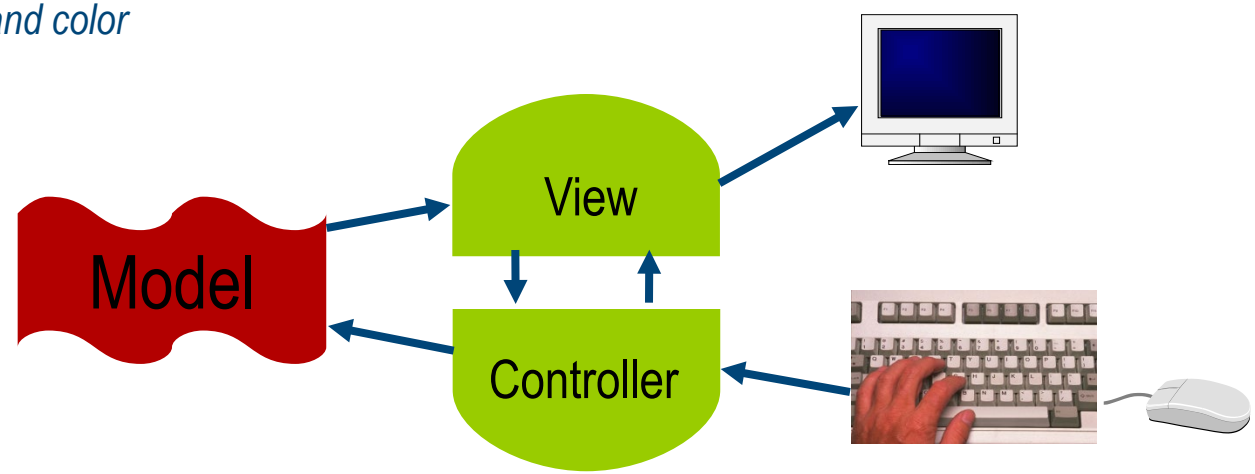


Example Application



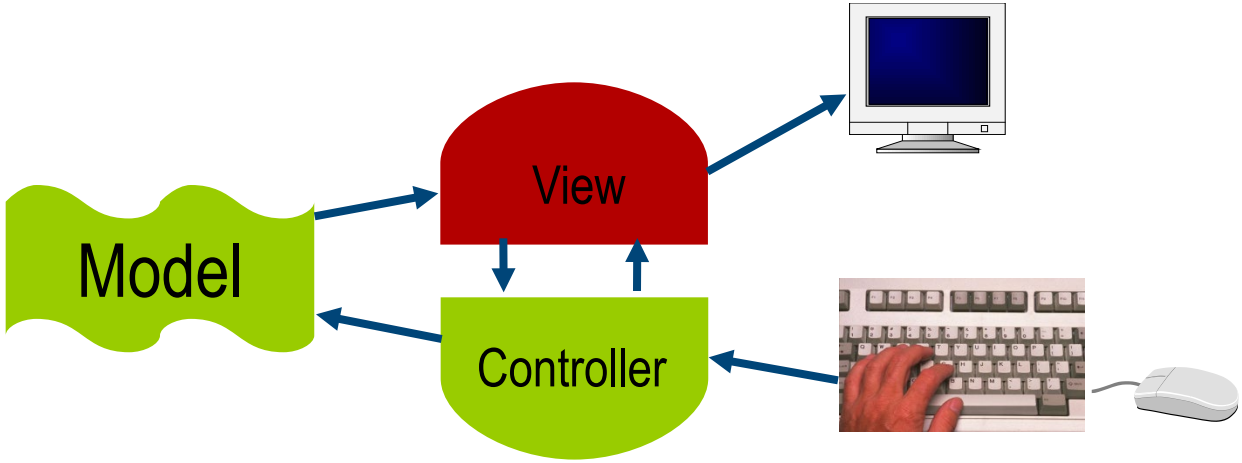
Model

- Model = Information the app is trying to manipulate
- Representation of real world objects
  - circuit for a CAD program
    - *logic gates and wires connecting them*
  - shapes in a drawing program
    - *geometry and color*



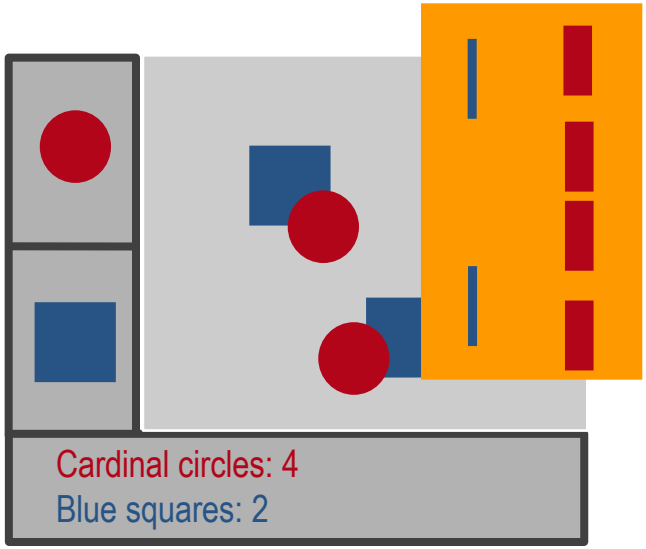
View

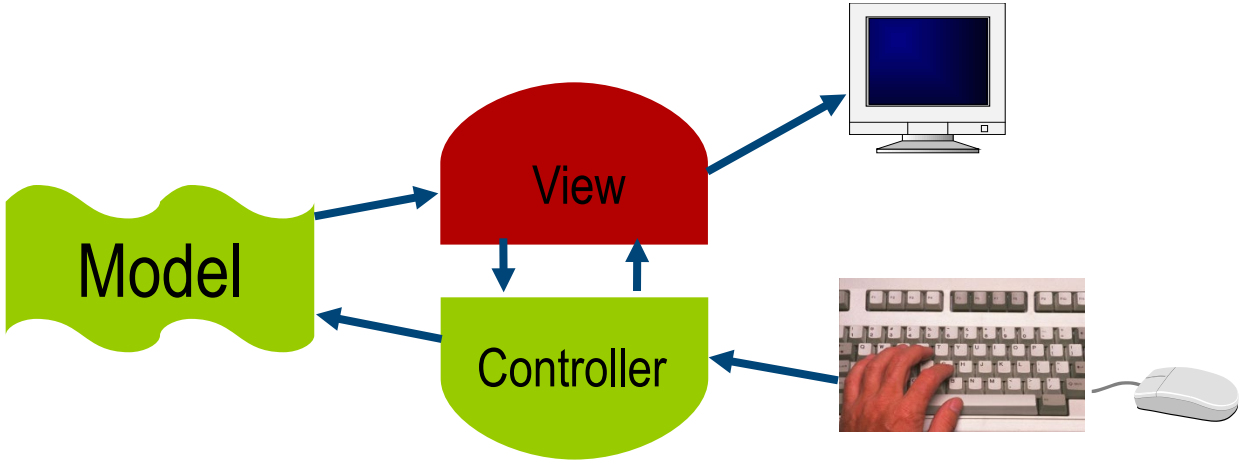
- Implements a visual representation of the model
  - Can generalize to virtually any externally observable action: audio/speech, alarms, …



View

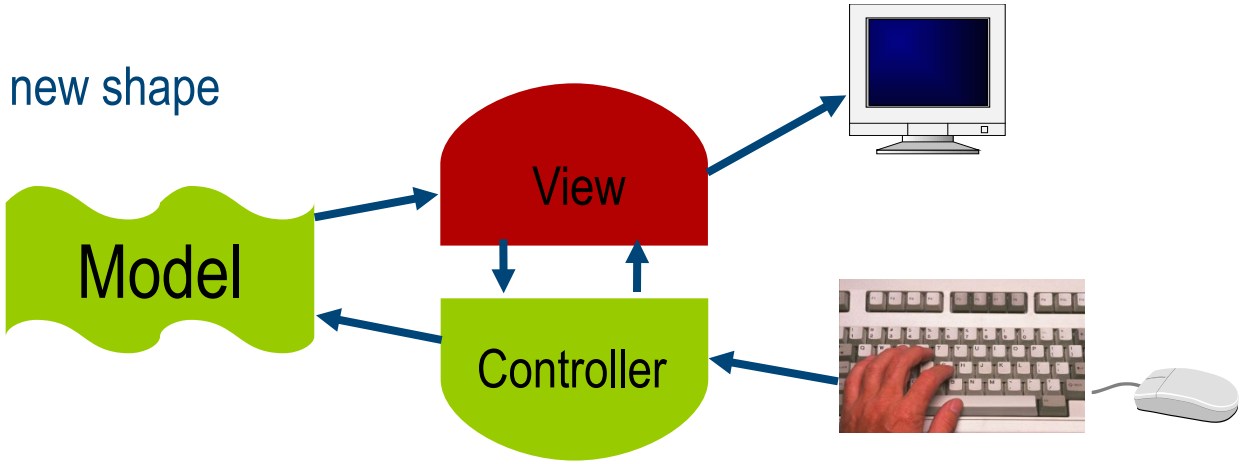
- Implements a visual display of the model
- May have multiple views
  - e.g., shape view and numerical view





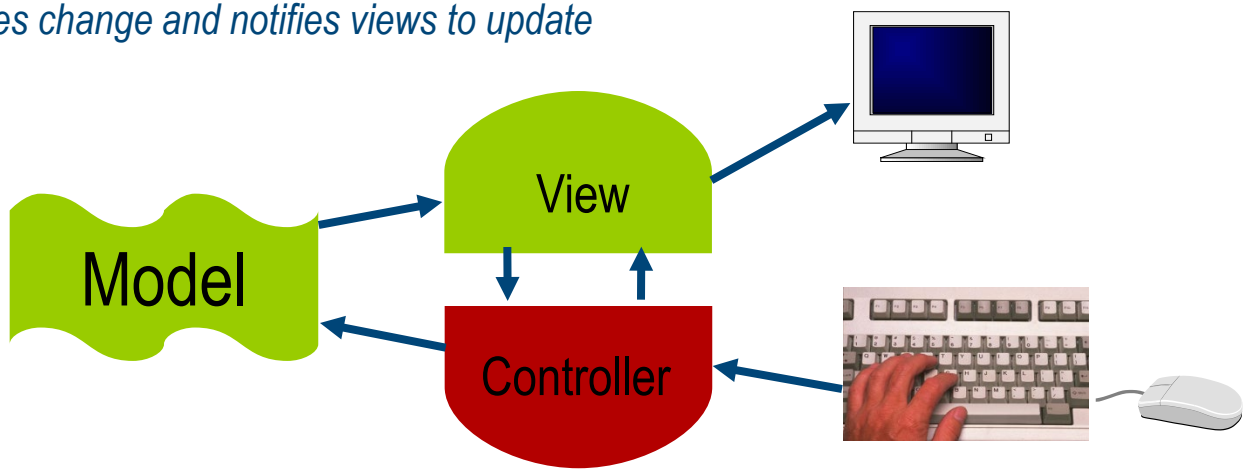
View

- Implements a visual display of the model
- May have multiple views
  - e.g., shape view and numerical view
- Any time the model is changed, each view must be notified so that it can change later
  - e.g., adding a new shape
  - *pattern?*



Controller

- Receives all input events from the user
- Decides what they mean and what to do
  - communicates with view to determine which objects are being manipulated (e.g., selection)
  - calls model methods to make changes on objects
    - *model makes change and notifies views to update*



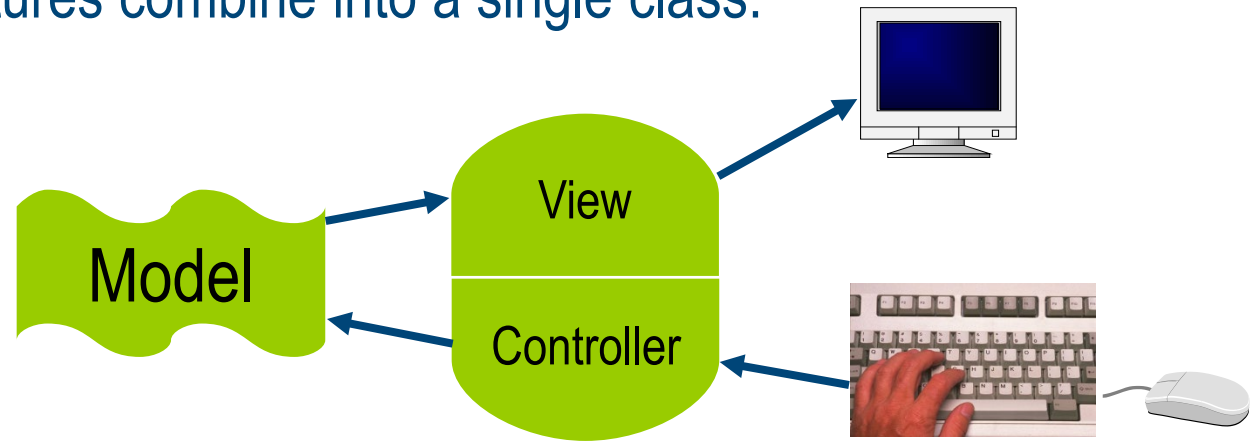
View/Controller Relationship

- View/Controller Relationship =
  "*pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)*"

- Controller must contact view
  to interpret what user events mean
  - e.g., selection

Combining View & Controller

- View and controller are tightly intertwined
  - lots of communication between the two
- Almost always occur in pairs
  - i.e., for each view, need a separate controller
- Many architectures combine into a single class:

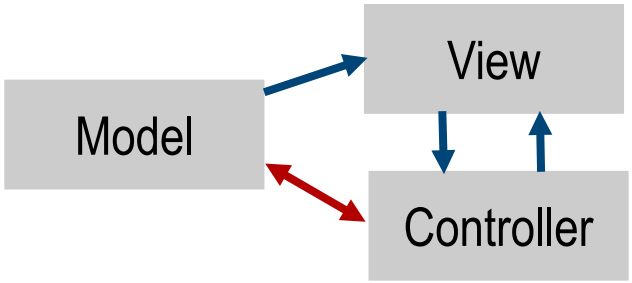


Why MVC?

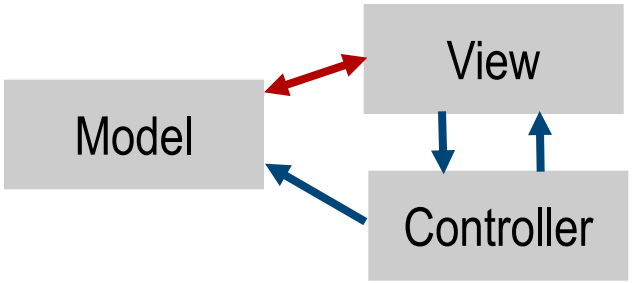
- Combining MVC into one class or using global variables will not scale
  - model may have more than one view
  - …each is different and needs update when model changes
- Separation eases maintenance
  - easy to add a new view later
  - new model info may be needed, but old views still work
  - can change a view later, e.g., draw shapes in 3-d (recall, view handles selection)

Reflections

- MVC is a (complex) design pattern!
  - Made up from mainly from Composite, Observer
  - See http://c2.com/cgi/wiki?ModelViewController for a discussion (after which not exactly everything is clear)
- Variations
  - Stanford HCI:



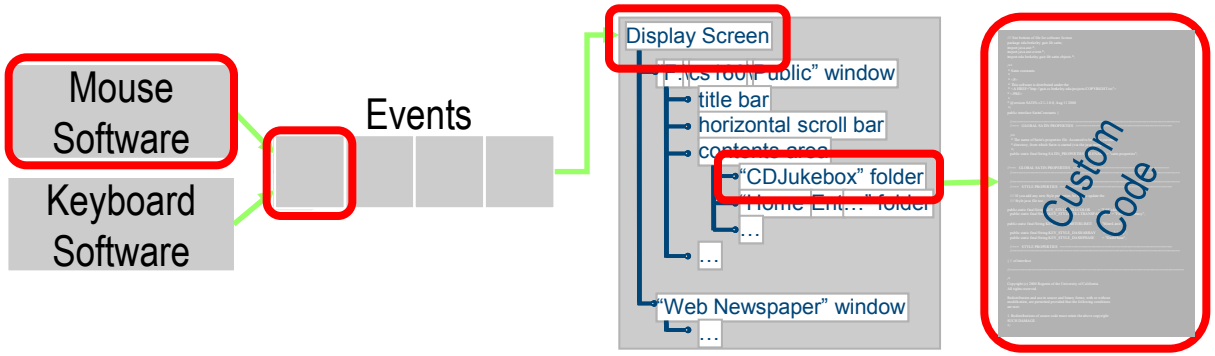
  - http://ootips.org/mvc-pattern.html :



  - ModelDelegate (see link above)

Summary

- Event-driven programming, widgets, event loop



- Model-View-Controller pattern as a GUI paradigm