UI Software Organization

3

(based on CS4470/6456 slides by
Keith Edwards)

The user interface

- Generally want to think of the "UI" as only one component of the system
  - Deals with the user
  - Separate from the "functional core" (AKA, the "app")

- Feels different in 3D than traditional 2D interfaces right now
  - Less "well accepted" metaphors
  - Less integration with OS/system

- Many commonalities between 2D and 3D, however
  - Significant conceptual overlap between this course and 2D class (CS4470: Intro to UI Software / CS6456: Principles of UI Software)

Separation of Concerns

- There are good software engineering reasons to do this
  - Keep UI code separate from app code
  - Isolate changes
  - More modular implementation
  - Different expertise needed
  - Don't want to iterate the whole thing

- Without strong separation, app developer must do everything
  - Many 3D libraries (e.g., three.js) provide little separation/abstraction
  - No consistency across apps

In practice, very hard to do...

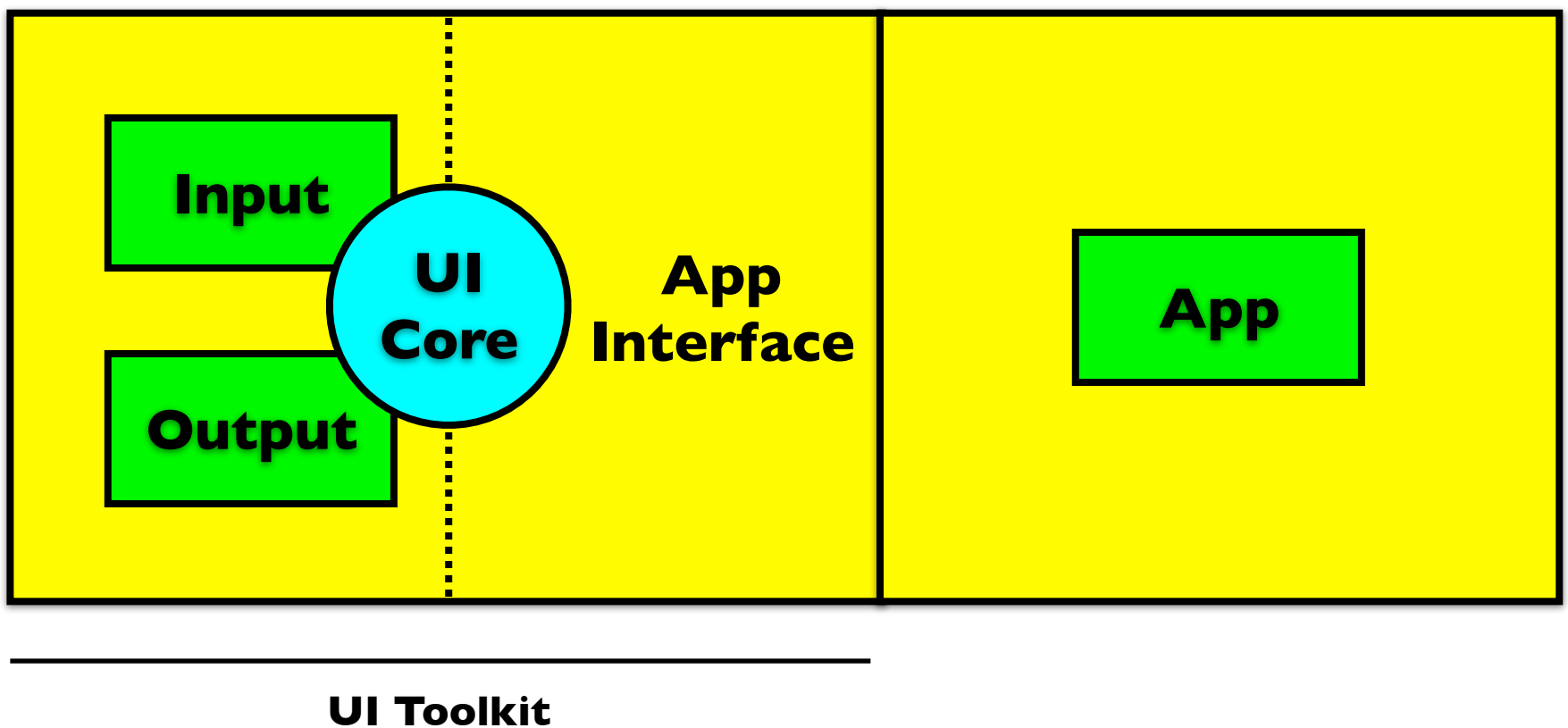
- More and more interactive programs are tightly coupled to the UI
  - Not just 3D; touch, desktop, etc
  - Programs structured around UI concepts/flow
  - UI structure "sneaks into" application
- Not always bad...
  - Tight coupling can offer better feedback/performance

Separation of concerns a central theme of UI organization

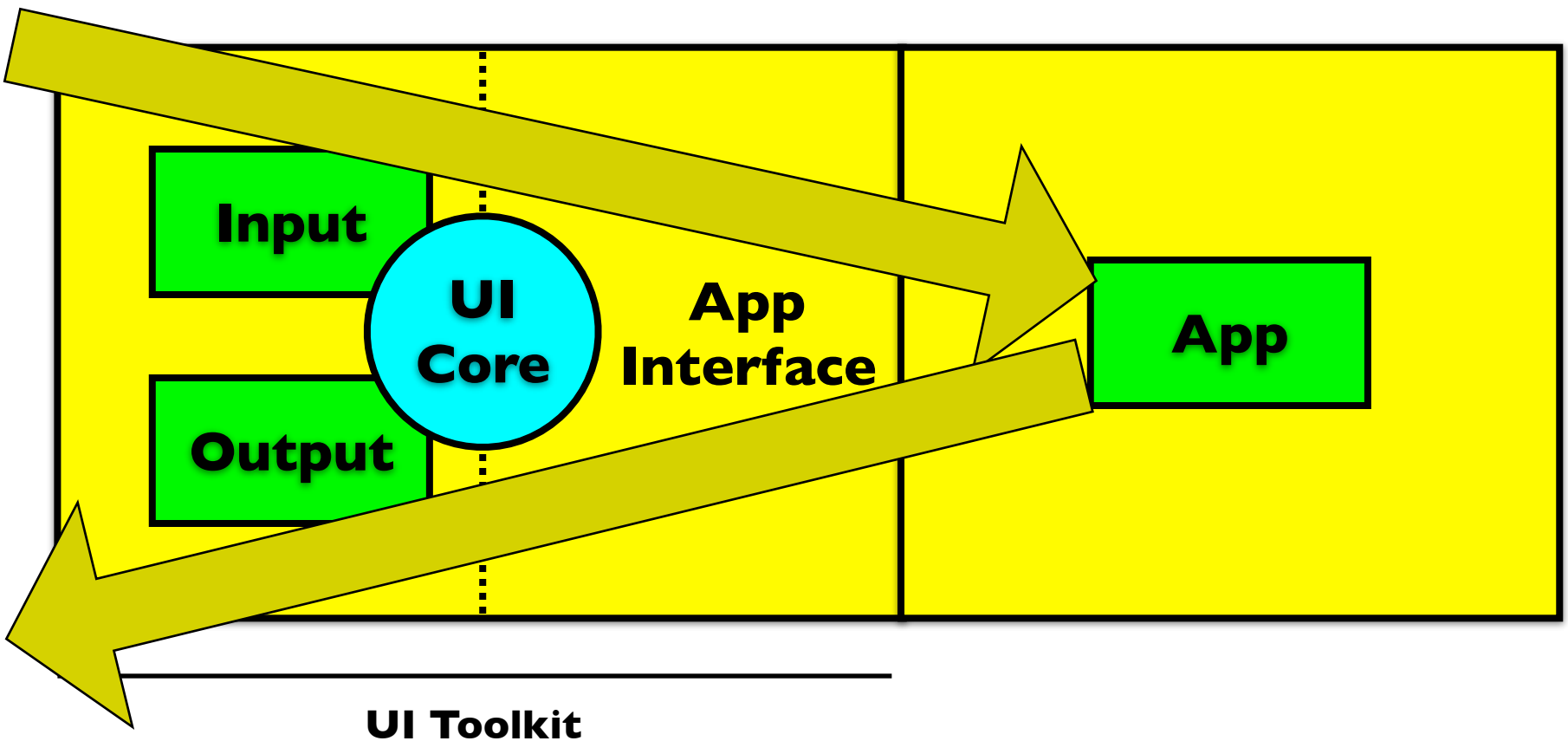
- A continual challenge
- A continual tension and tradeoff

- Real separation of UI from application is almost a lost cause

Conceptual Overview of the UI



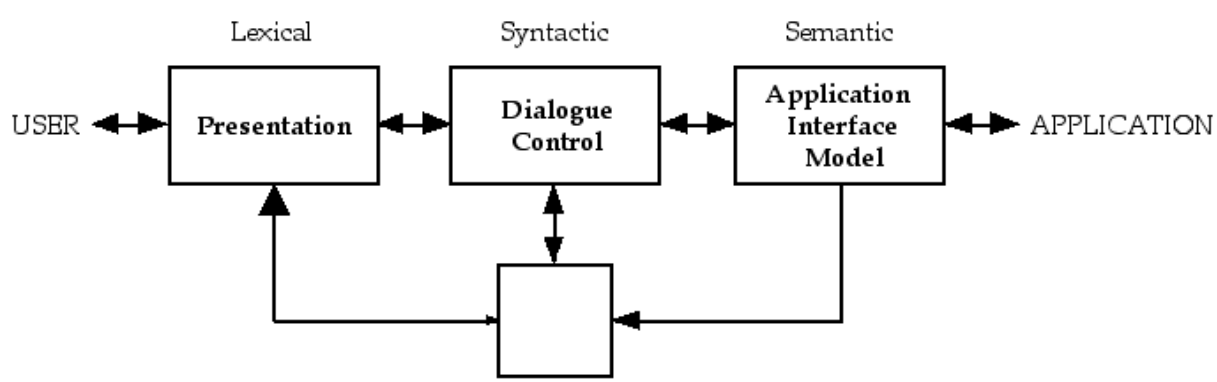
Basic UI Flow



How would you architect this?

- Tempting to architect systems around these boxes
  - One module for input, one for output, etc.
  - Has been tried ("Seeheim model")
    - Lots of research and development pursuing UIMSes in 80's and 90's https://en.wikipedia.org/wiki/User_interface_management_system
  - Didn't work well

Seeheim model



Why "Big Box" architectures don't work well

- Modern ("direct manipulation") interfaces tend to be collections of quasi-independent agents
  - Each **interactor** ("object of interest" on the screen) is separable
  - Example: an on-screen button (in 2D or 3D)
    - Produces "button-like" output
    - Acts on input in a "button-like" way
    - Etc.

Has lead to object-based architectures in 2D Systems

- 2D Interactor classes are organized into a hierarchy of super/subclasses
  - Each class represents a specific **type** of interactor (a button, or a scrollbar, or a window, for instance)
  - Typically a top-level "root interactor" class that describes basic interactor capabilities, and some intermediate classes that inherit from it for common behaviors.
  - Common methods for stuff like drawing, handling input, which are overridden in specific subclasses
- Leaf-node classes for the things you actually see on the screen (buttons, scrollbars, etc.)
  - These provide **specific** implementations of the common methods
    - Drawing output—rendering button-like appearance on the screen
    - Handling input—what happens when the user clicks the button

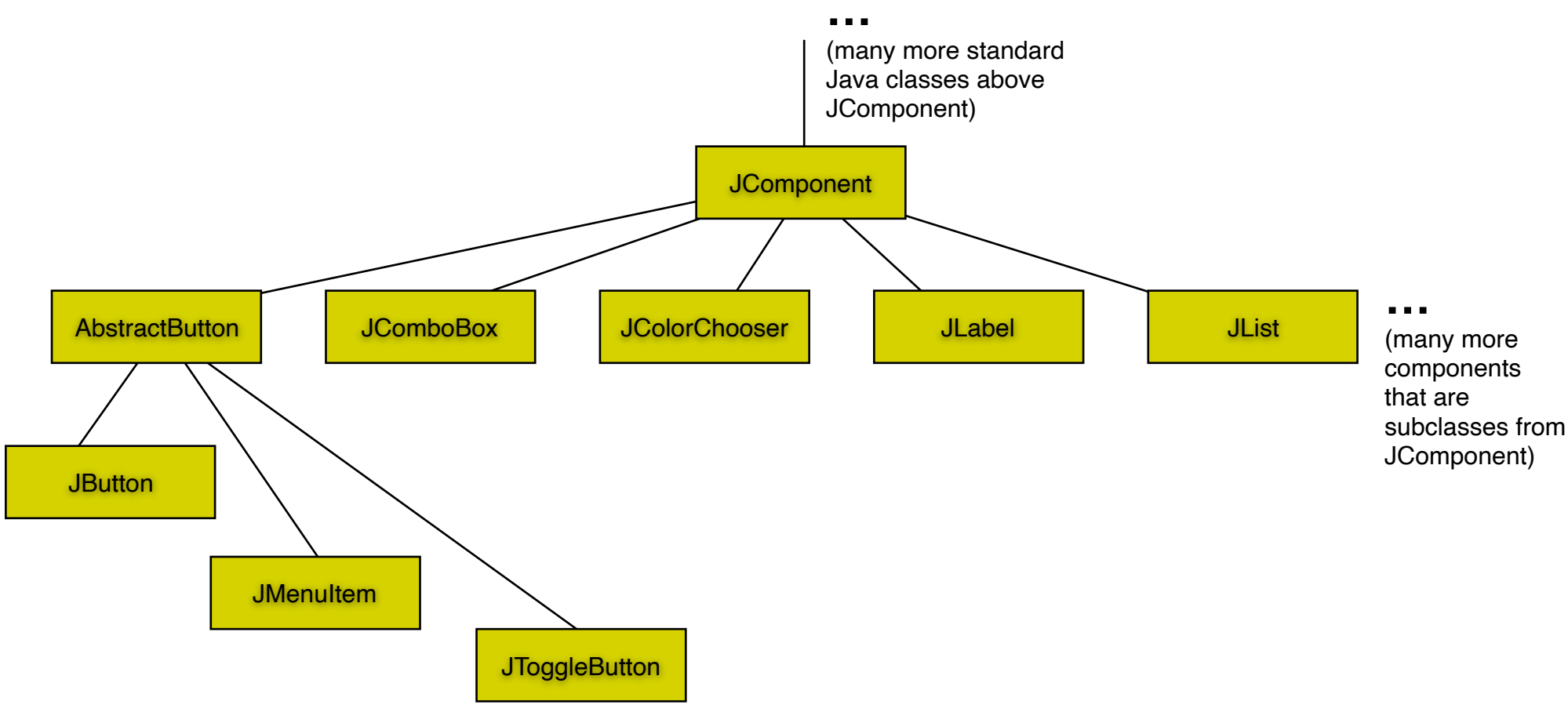
Has lead to object-based architectures in 2D Systems (2)

- Each on-screen interactor corresponds to an object instance
  - You instantiate a new object from a selected class, to create each on-screen interactor
  - E.g., each button in your UI corresponds to a separate Button instance
- You will probably have many of these, depending on how complex your UI is.

Has lead to object-based architectures in 2D Systems (3)

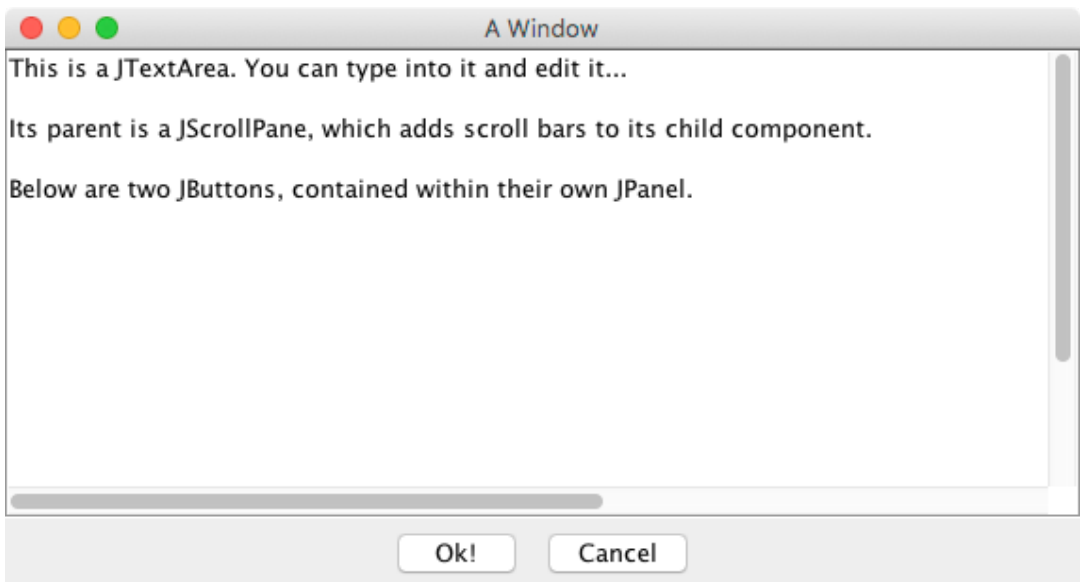
- Finally, these instances are organized hierarchically at runtime
  - You take these instances (each representing a particular interactor in your UI) and assemble them into a big tree.
  - The tree represents the spatial containment relationships in your UI's layout
    - Child interactors are contained within their parents — provides you a way to group interactors together
  - Non-leaf nodes in the tree are containers (interactors that are designed to hold other interactors). Leaf nodes are simple things like buttons, scroll bars, labels, et.
- Most of these trees are a bit more complex than you might initially imagine
  - Sometimes, additional (even multiple levels) of containers needed in order to get the layout you want.
- NOTE that this interactor tree, created at runtime, is different than class hierarchy created at development time

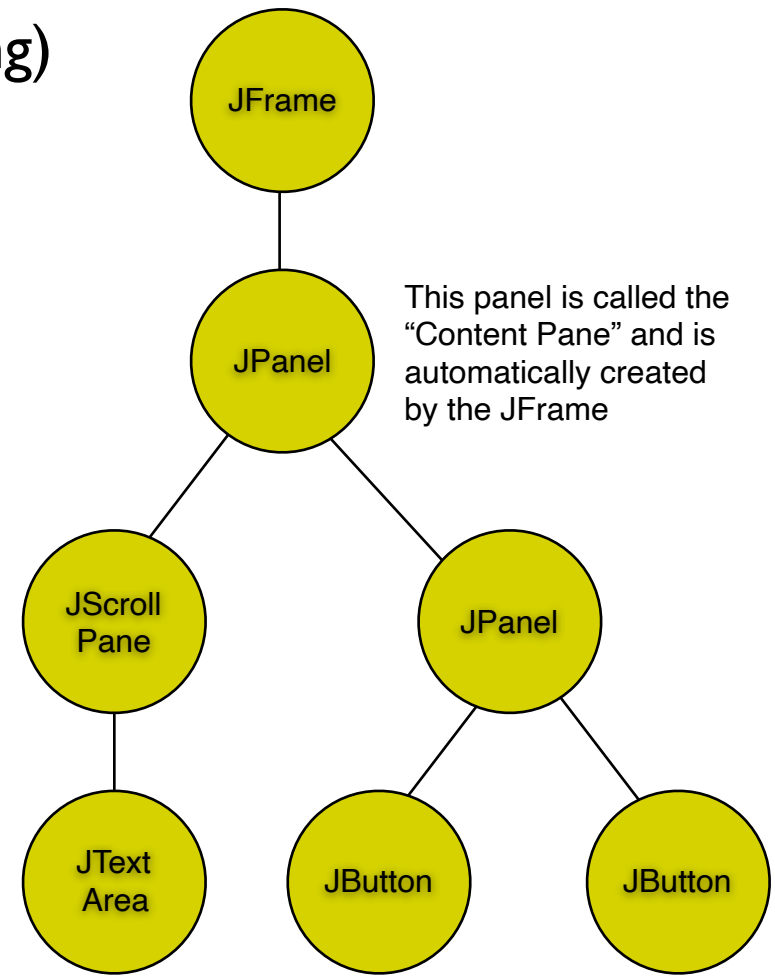
An Example from Swing: Class hierarchy (from javax.swing)



An Example from Swing: Assembling a runtime tree for

- Swing class hierarchy (from javax.swing)





Challenge: maintaining separation of concerns

- Trick is coming up with a separation that works quickly, simply, and extensibly
  - Even a single button may be hopelessly complex (pluggable looks-and-feels anyone?)
  - Needs to be extensible to new interactors
  - What's the right factoring for all this stuff?
- Will see some strategies later
- Basically: common O-O patterns to manage complexity

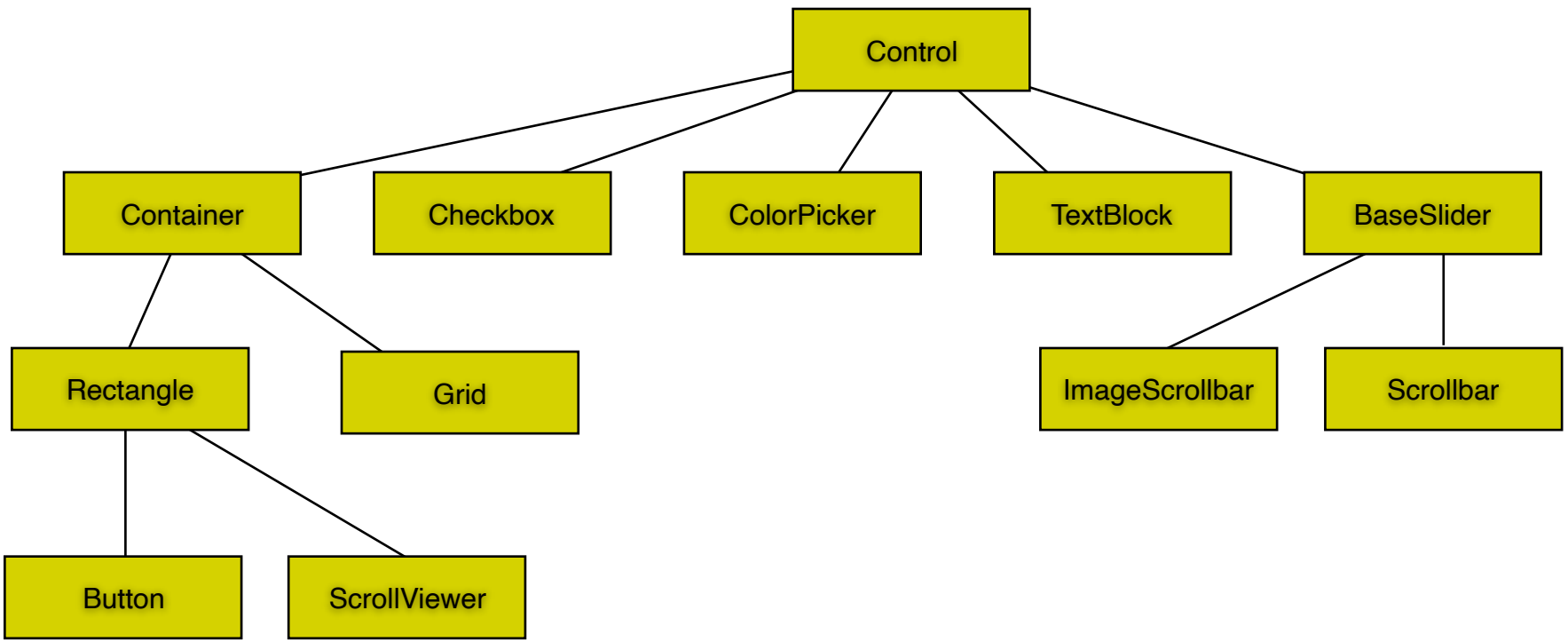
Similar Structure for 3D

- Babylon GUI classes are organized into a hierarchy of super/subclasses
  - Each class represents a specific **type** of interactor (a button, or a scrollbar, or a slider, for instance)
  - A top-level "root" class that is 2D or 3D
    - 2D GUI is either Fullscreen (one/scene) or mapped as a texture on a mesh
    - 3D GUI roots are positioned in 3D in the scene
- Leaf-node classes for the things you actually see on the screen (buttons, scrollbars, etc.)
  - These provide **specific** implementations of the common methods
    - Drawing output—rendering button-like appearance on the screen
    - Handling input—what happens when the user clicks the button
- GUI classes are separate from the content
  - Can use 3D objects as part of GUI controls, but app elements are not "in" the GUI hierarchy

Similar Structure for 3D (2)

- As with 2D, these instances are organized hierarchically at runtime
  - You take these instances (each representing a particular interactor in your UI) and assemble them into a collection of trees
    - Separate tree for each 3D menu in the application
    - The tree represents the spatial containment relationships in your UI's layout
    - Child interactors in 2D DynamicTextures are contained within their parents — provides you a way to group interactors together (like 2D, clipping can be overridden, within the area of the texture)
  - Non-leaf nodes in the tree are containers (interactors that are designed to hold other interactors). Leaf nodes are simple things like buttons, scroll bars, labels, etc.
- As with 2D, this interactor tree, created at runtime, is different than class hierarchy created at development time

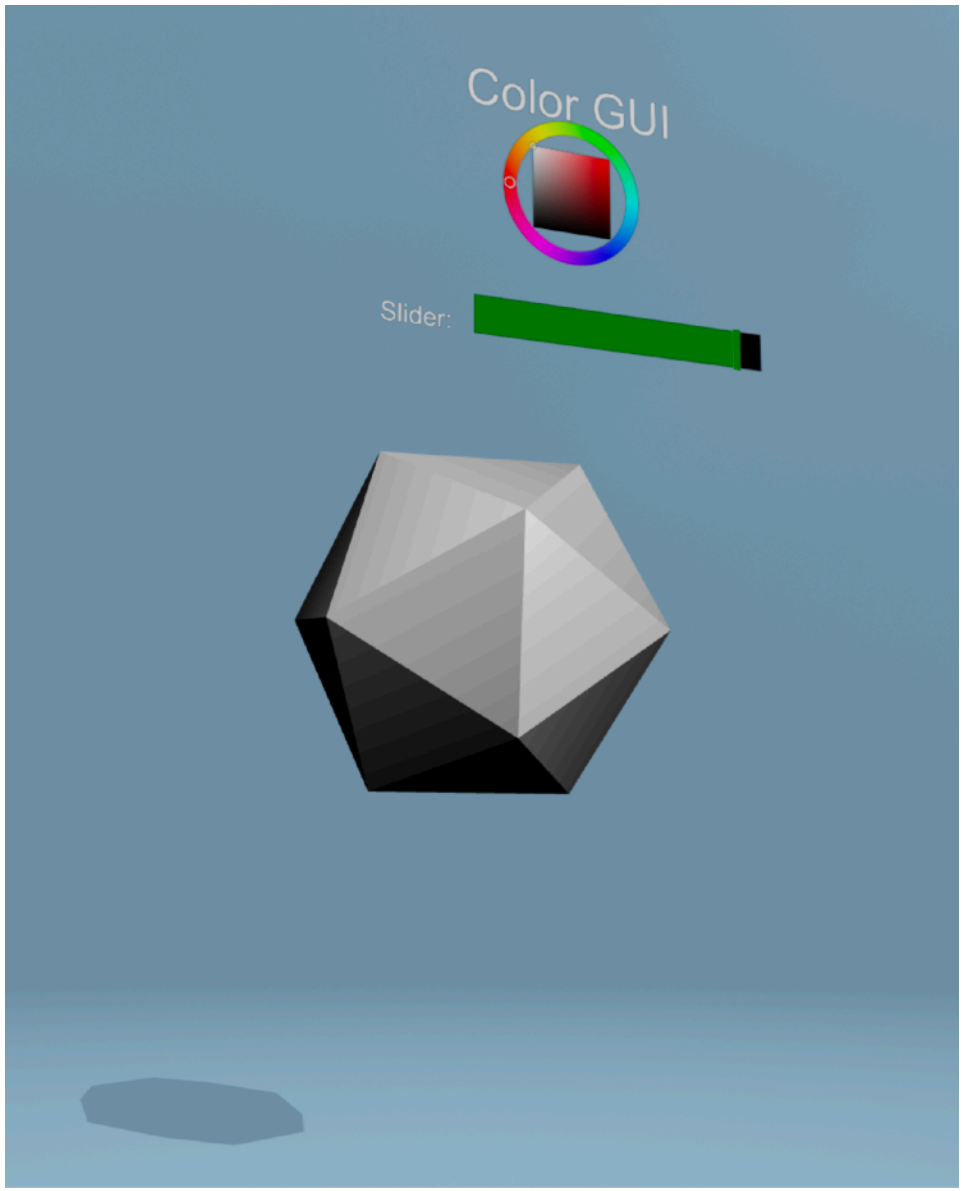
An Example from Babylon: Class hierarchy of GUI 2D

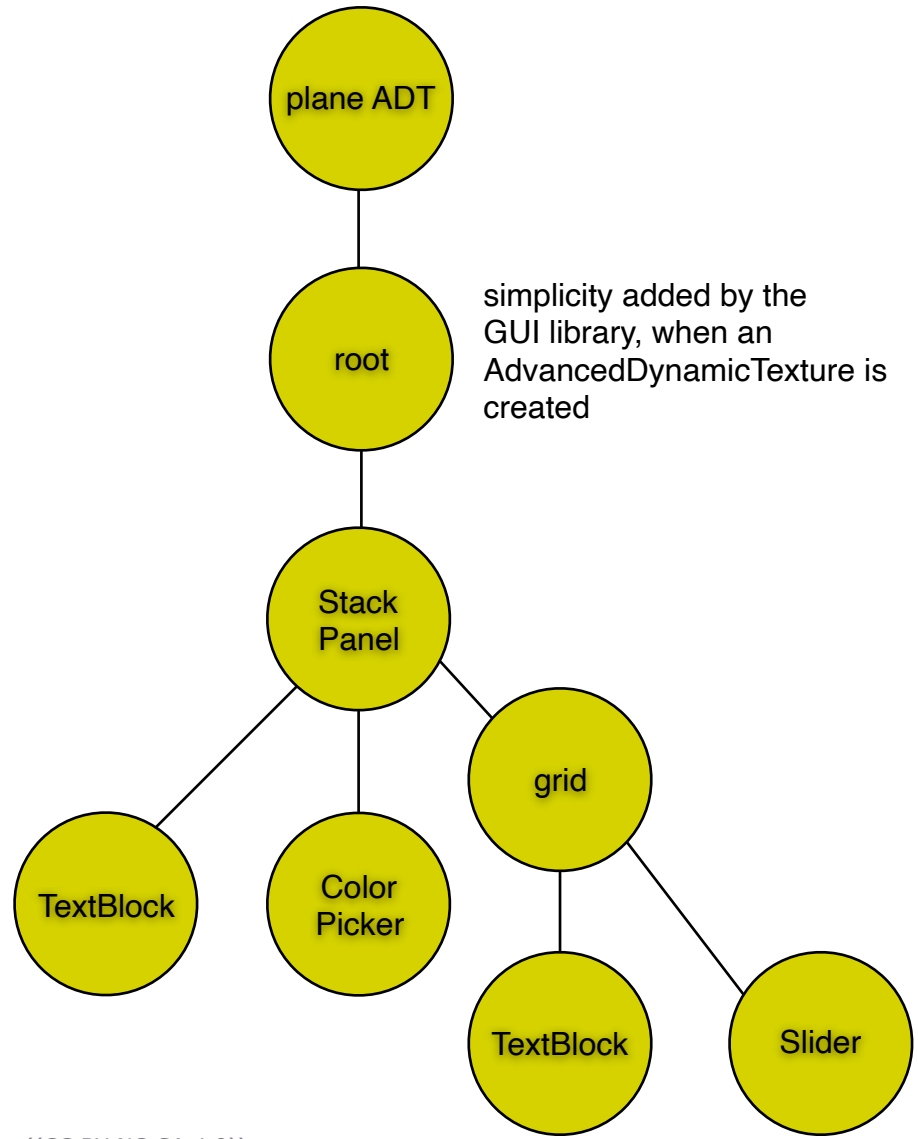


...
(more components that are subclasses from Control and Container)

Babylon GUI sample

https://www.babylonjs-playground.com/#190R08





simplicity added by the GUI library, when an AdvancedDynamicTexture is created

Key Difference is Lack of System Support within App

- All 3D content must be rendered by your application
  - No notion of OS-level windows, dialogs, keyboards, etc.
- Has security implications
  - Application implements everything, application sees everything
    - Passwords, etc
- No platform-wide look and feel, no "updates"
  - Controller models, pointer interactions, common operations like teleport, picking, etc

UI Toolkits

- Much more advanced in 2D than 3D
  - Leverage OS capabilities and commonalities
  - WIMP metaphor and abstractions
- System to provide development-time and runtime support for UIs
  - Core functionality
  - Input & output handling
  - Connecting to the application
- Also: specific interaction techniques
  - Library of interactors
  - Look and feel (sometimes pluggable)

Categories of users similar across 2D and 3D

- Consumer
  - End-user, albeit indirectly
- Programmers
  - Interface designer
  - Application builder
  - Toolkit implementer/maintainer
  - Interactor writer
  - Tool builder
  - Expert end-user (through scripting)

Toolkit functionality in detail (Roadmap of initial topics)

- 2D topics and 3D-on-2D-screens topics mixed together
  - Will revisit some of this again for immersive 3D

- Core functions
  - Hierarchy management
    - Create, maintain, tear down tree of interactor objects
  - Geometry management
    - Dealing with coordinate systems
    - On-screen bounds of interactors
  - Interactor status/information management
    - Is this interactor visible? Is it active?

Toolkit functionality in detail

- Output
  - Layout
    - Establishing the size and position of each object
    - Both initially, and after a resize
  - (Re)drawing
  - Damage management
    - Knowing what needs to be redrawn
  - Localization and customization
    - We won't talk much about this...

Toolkit functionality in detail

- Input
  - Picking in 2D
    - Figuring out what interactors are "under" a given screen point
  - Picking in 3D
    - What intersects with a ray through the scene
      - Revist later with in-depth examination of picking and selection in 3D
  - Event dispatch, translation, handling
    - This is where a lot of the work goes

Toolkit functionality in detail

- Application interface
  - How the UI system connects with application code
    - Callbacks
    - Command objects
    - Undo models
    - ...

Next:
A Whirlwind Intro to Babylon

3