

Designing Interactive Systems 2

Lecture 2: Window System Architecture

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/dis2

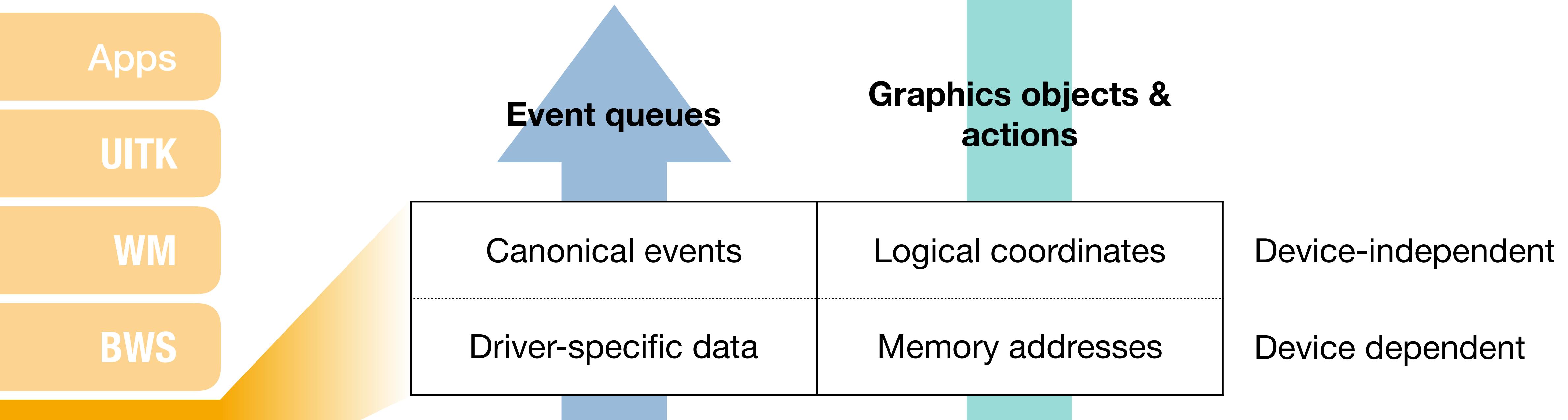


RWTHAACHEN
UNIVERSITY

CHAPTER 4

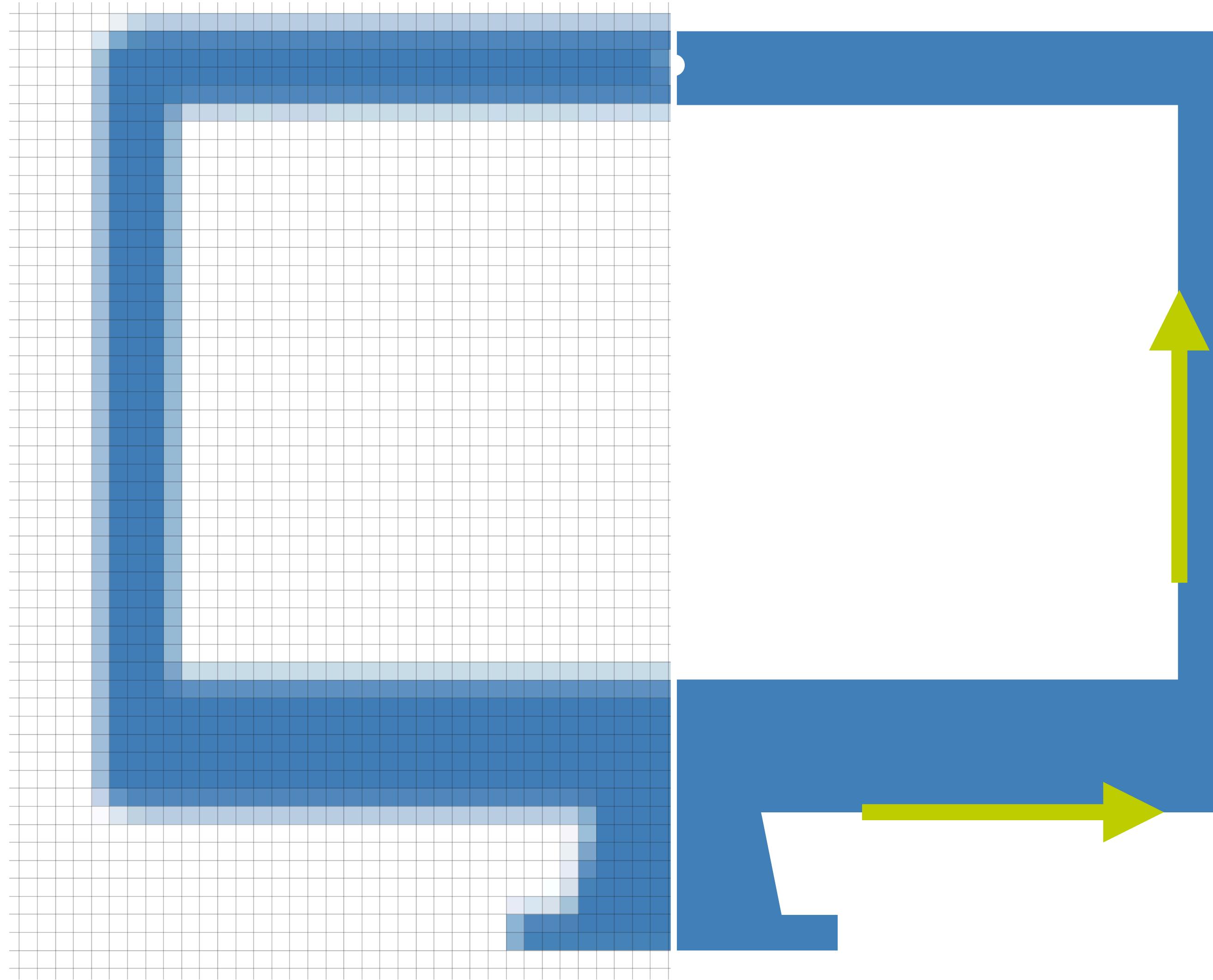
Graphics & Event Library

Graphics & Event Library



Graphics Models

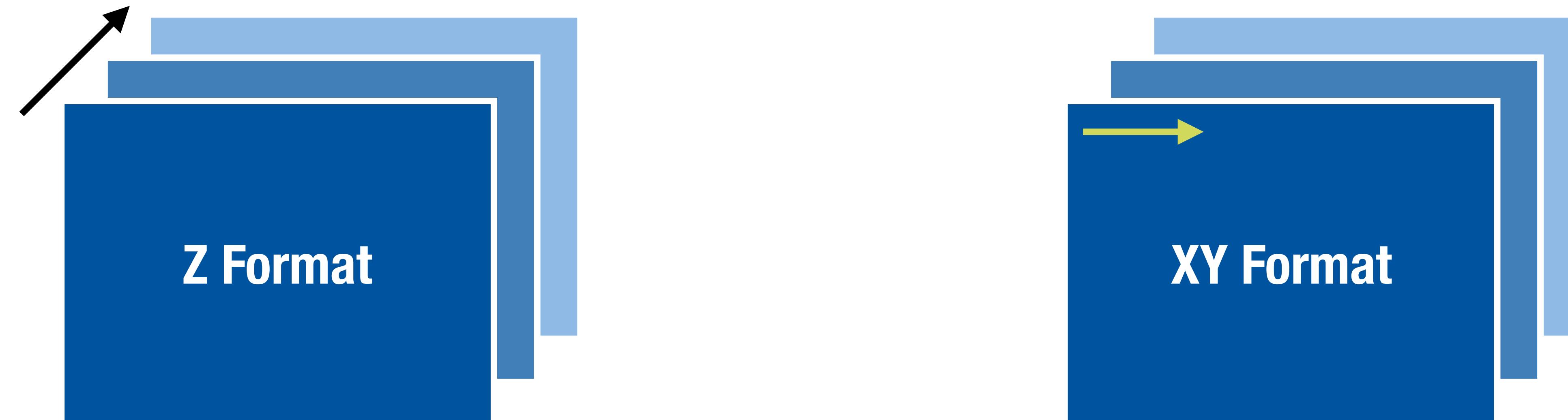
RasterOp Model



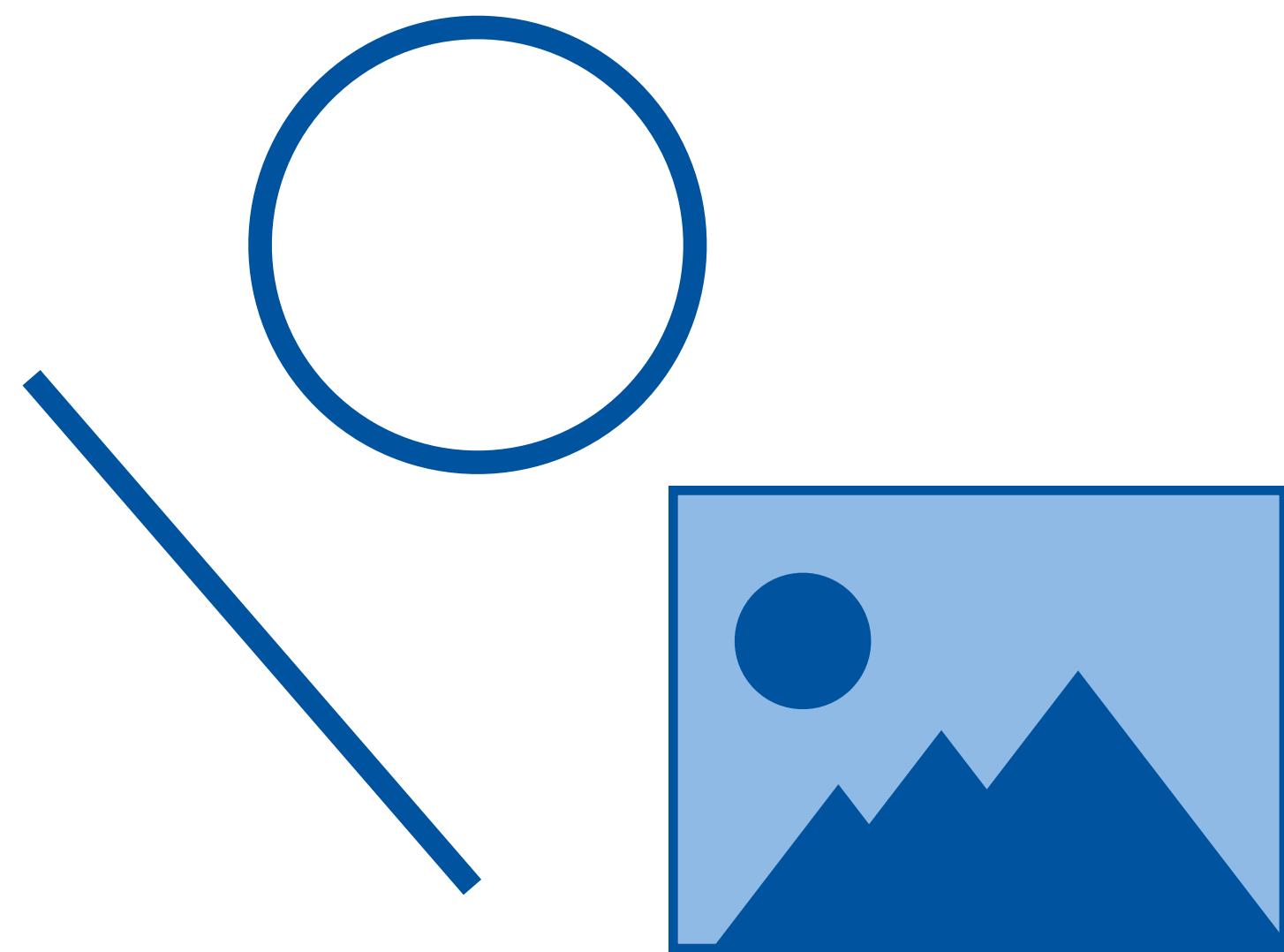
Vector Model

Graphics Library Objects: Canvas

- A canvas is a memory area with a coordinate system and memory-to-pixel mapping
- Different formats



Graphics Library Objects: Output Objects



Elementary Objects

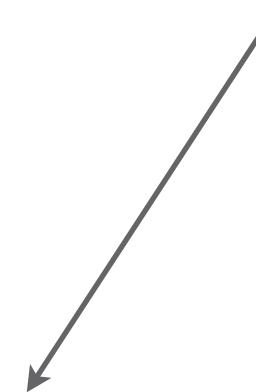


Complex Objects

Graphics Library Objects: Graphics Context

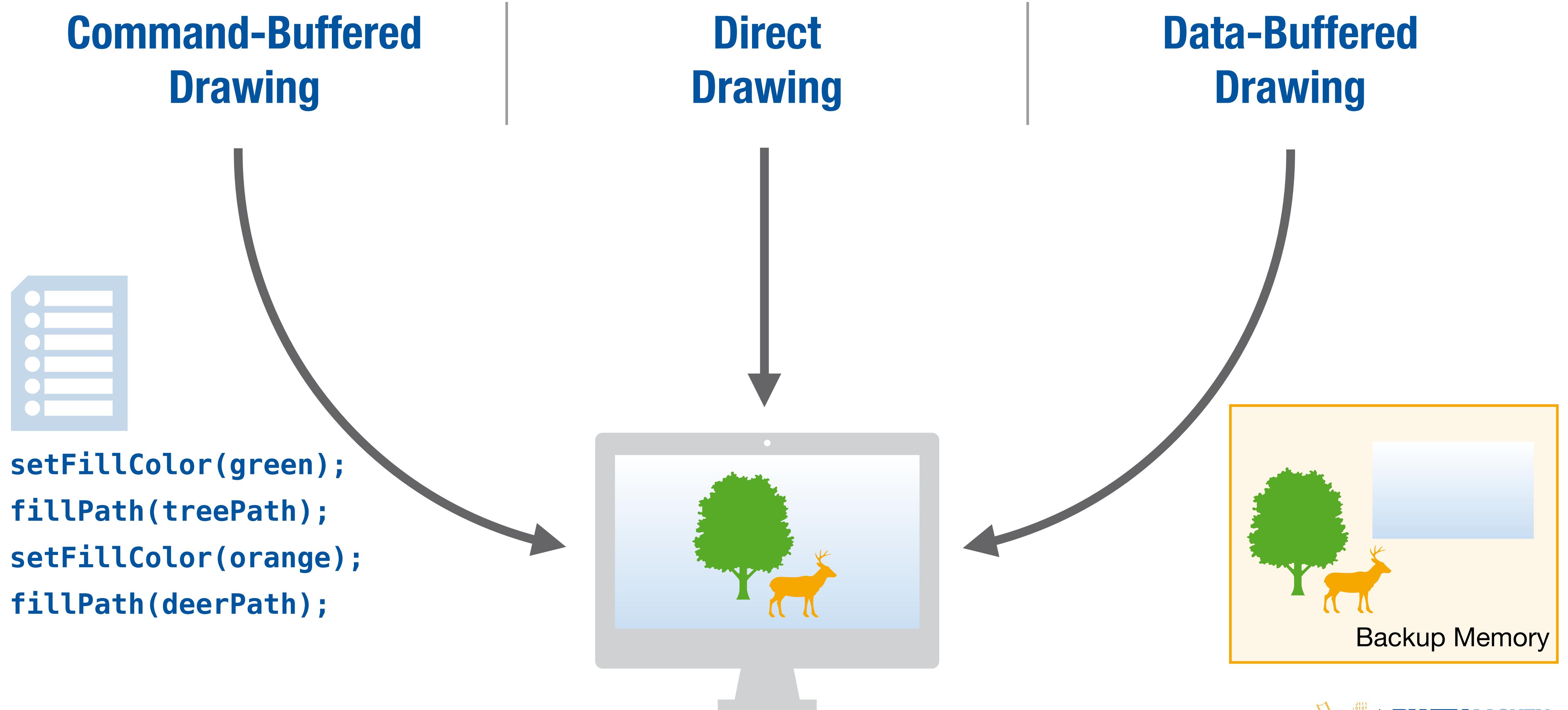
- State of the (virtual) graphics processor
- Goal: **reduce parameters to pass when calling graphics operations**

Attribute	Value
Font	Gill Sans
Font size	24 pt
Font color	(0,0,0)
Line width	2 px
...	...



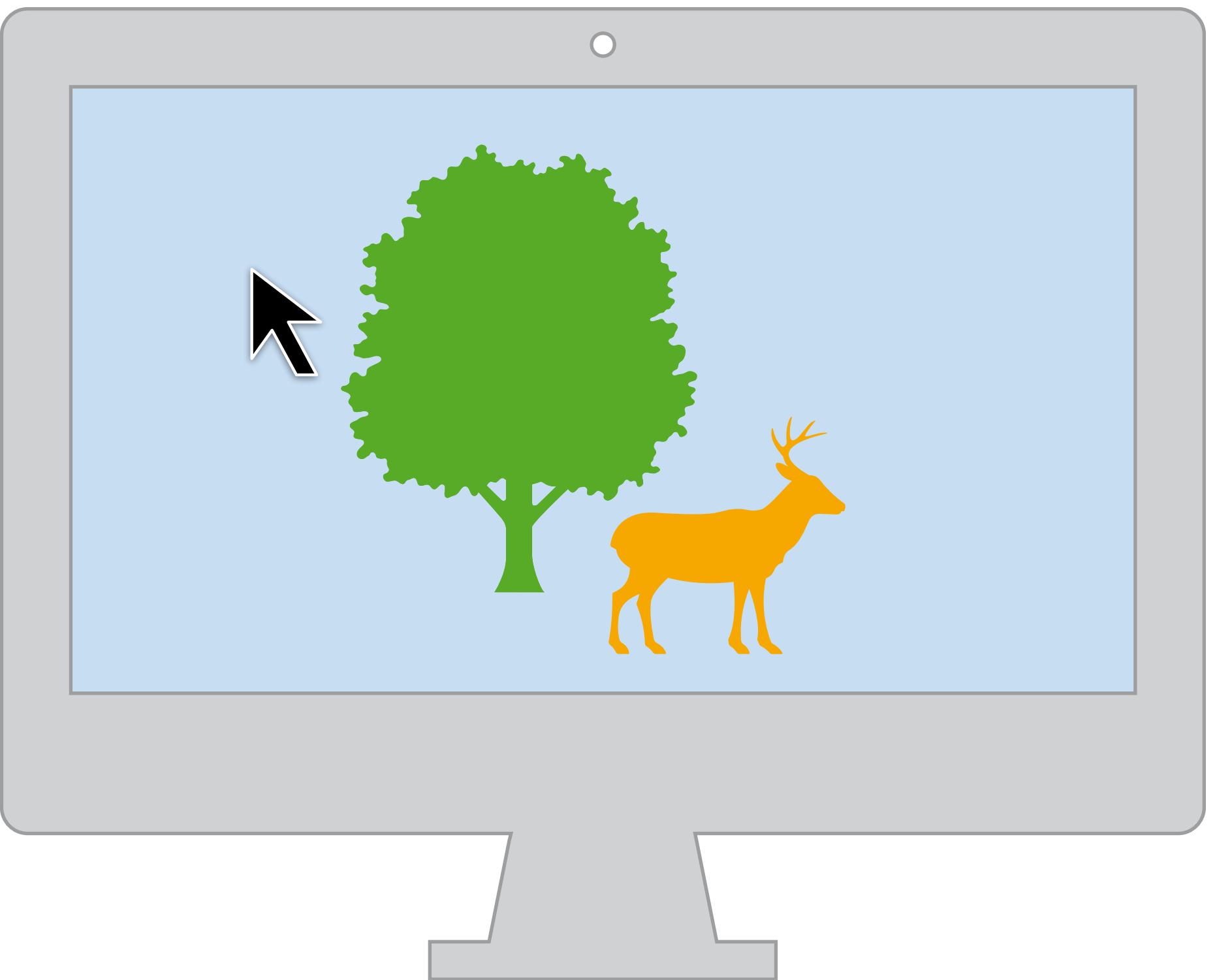
```
drawString(x, y, "Turtle");
```

Graphics Library: Drawing



Damage Repair

- Mouse cursor is always drawn by GEL (performance)
- Restoring contents that were occluded by mouse cursor needed



Event Library

Canonical Mouse Event Queue

Event	
Type	Movement
Value	(0,2)
Timestamp	13:37:12.203

Event	
Type	Mouse Down
Value	Left
Timestamp	13:37:12.271

Event	
Type	Mouse Up
Value	Left
Timestamp	13:37:12.289

Event	
Type	Movement
Value	(12,2)
Timestamp	13:37:12.416

Mouse Driver Event Queue

Event	
Button 1	0
Button 2	0
Button 3	0
X	0
Y	1
Wheel	0
Timestamp	31267

Event	
Button 1	1
Button 2	0
Button 3	0
X	0
Y	0
Wheel	0
Timestamp	31335

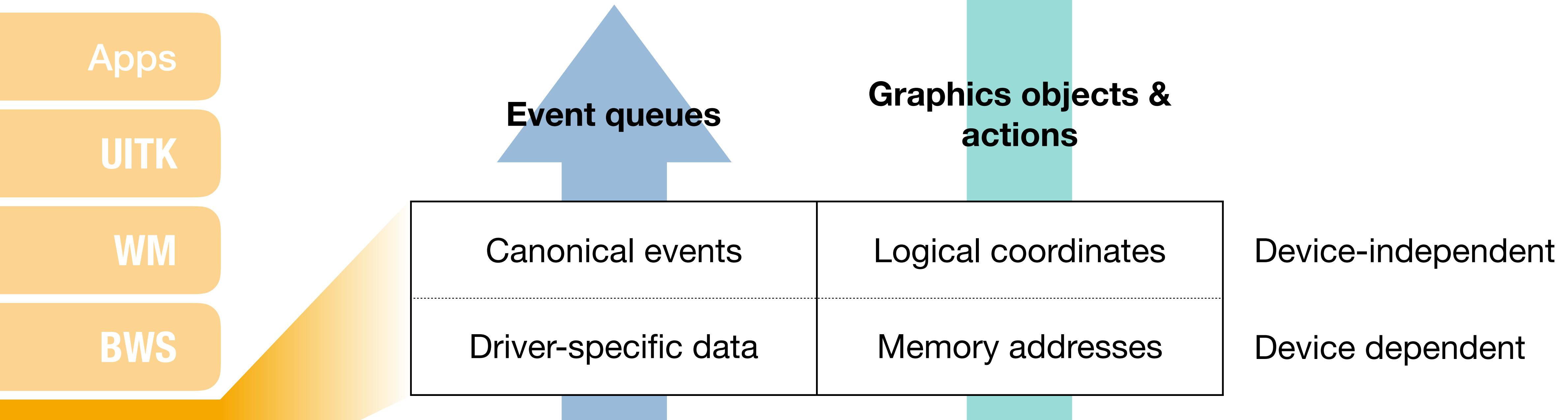
Event	
Button 1	0
Button 2	0
Button 3	0
X	0
Y	0
Wheel	0
Timestamp	31421

Event	
Button 1	0
Button 2	0
Button 3	0
X	6
Y	1
Wheel	0
Timestamp	31634

How Extensible is the GEL?

- Most systems: Not accessible to application developer
- GEL **as library**: extensible only with access to source code (X11)
- GEL access **via interpreted language**: extensible at runtime (NeWS)
 - NeWS example: Download PostScript code into GEL to draw triangles, gridlines, patterns,...

Graphics & Event Library



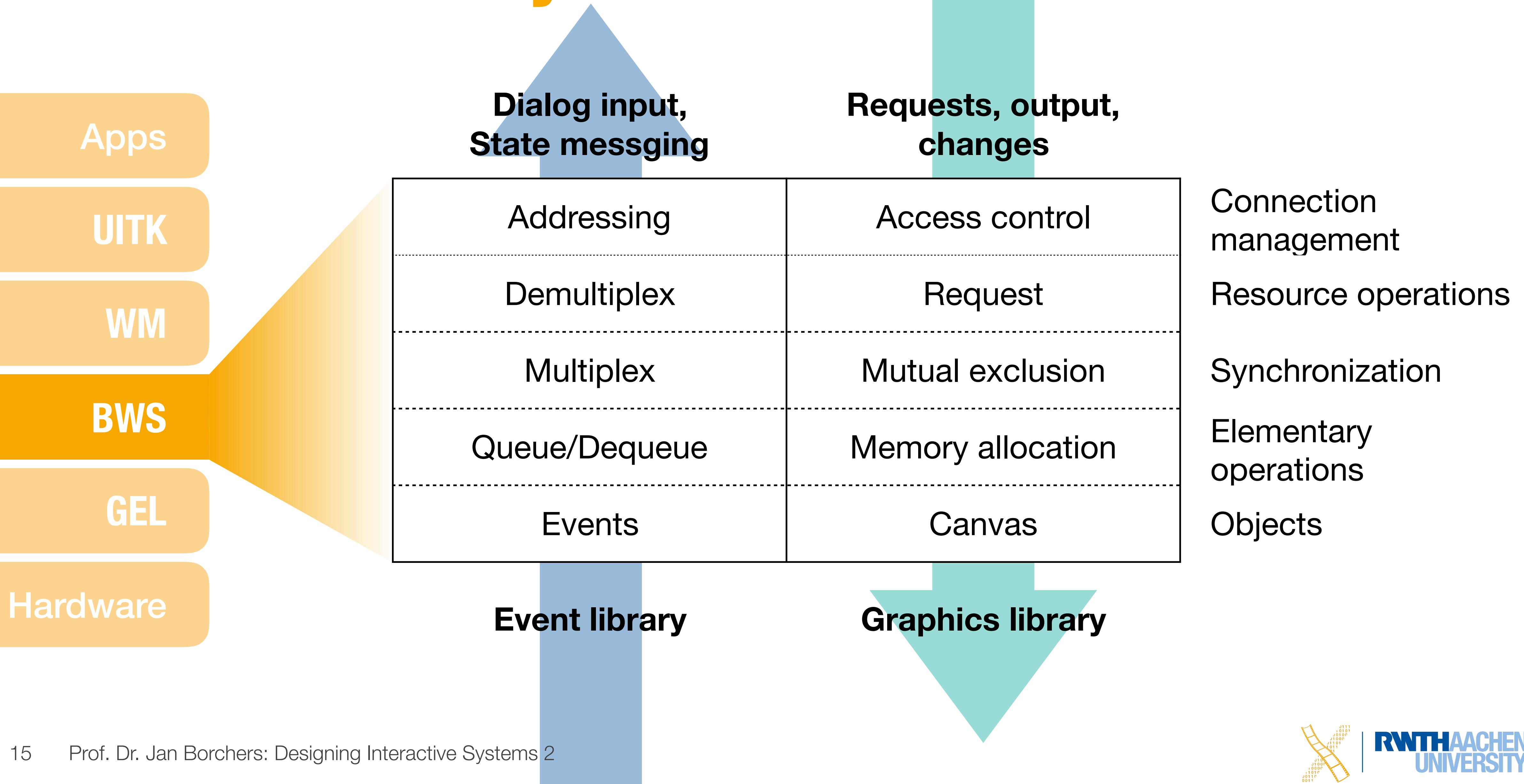
CHAPTER 5

Base Window System

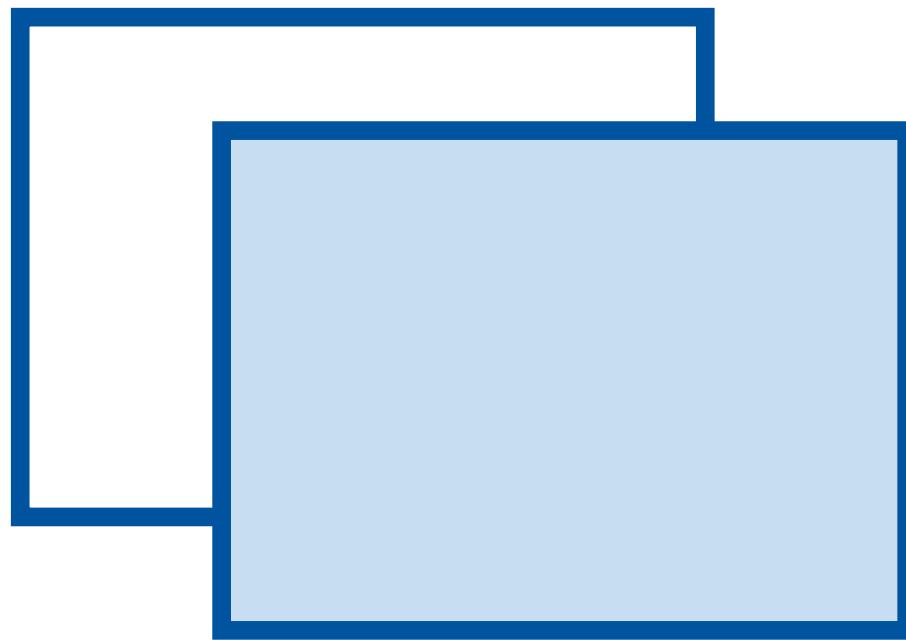
Base Window System

- **Core** of the Window System
 - Provides **WS-wide data structures** and operations
 - Manages **shared resources** to ensure consistency
- **Base window:** logical canvases that include the on-screen windows and / or widgets
- In general:
1 WS – k terminals, m applications, n objects per application (windows, fonts)

Base Window System



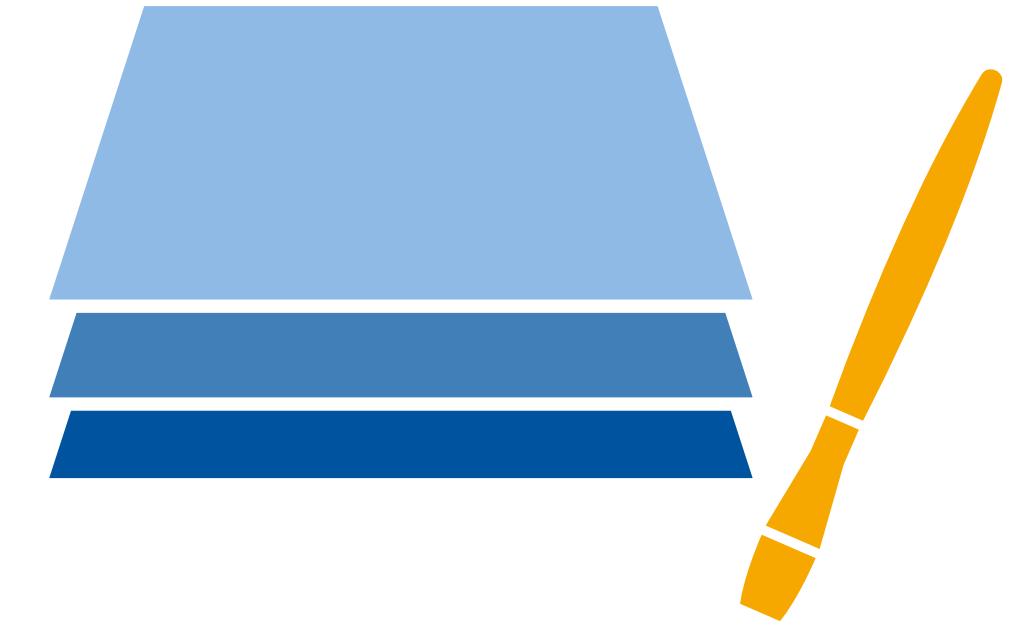
Base Window System: Objects



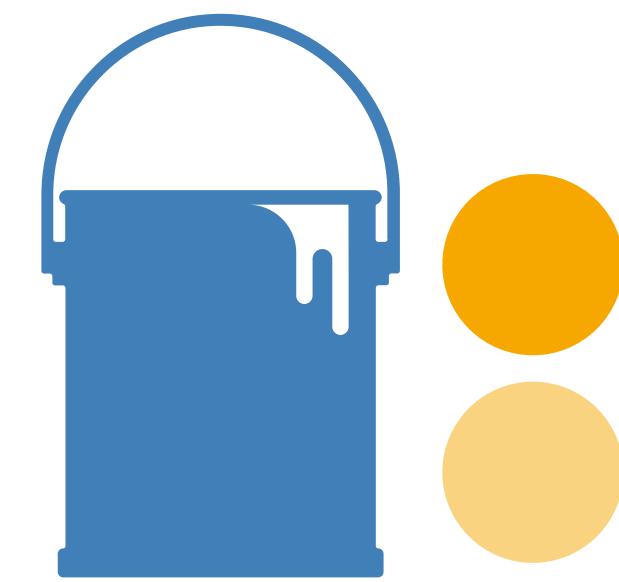
Window & Canvas



Events



Graphics Context

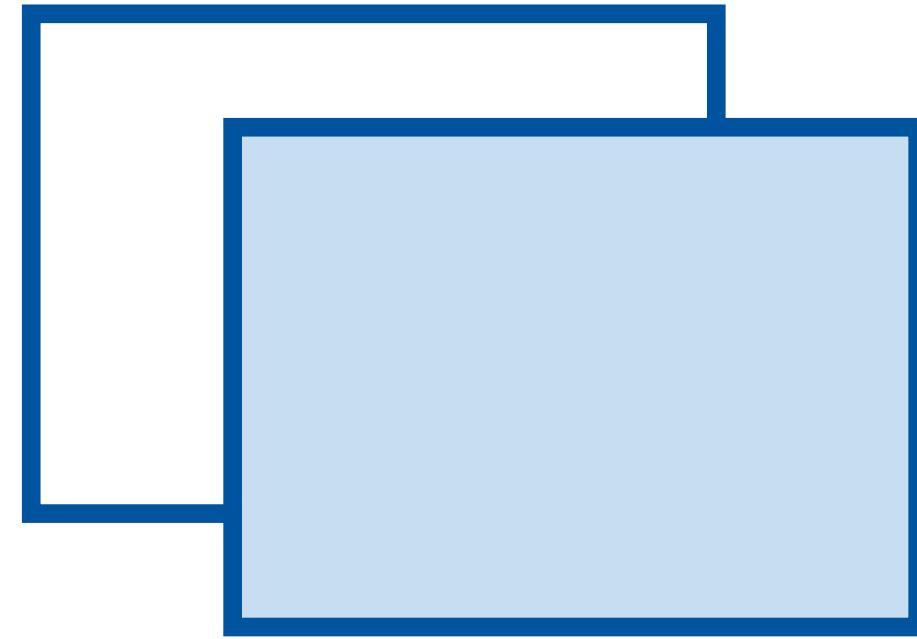


Color Tables



Fonts

Base Window System: Objects



Components

- Owning application
- Using applications
- Size, depth, border, origin
- State variables (only for windows)

Window & Canvas

Operations

- Drawing in local coordinate system
- State changes

Base Window System: Objects



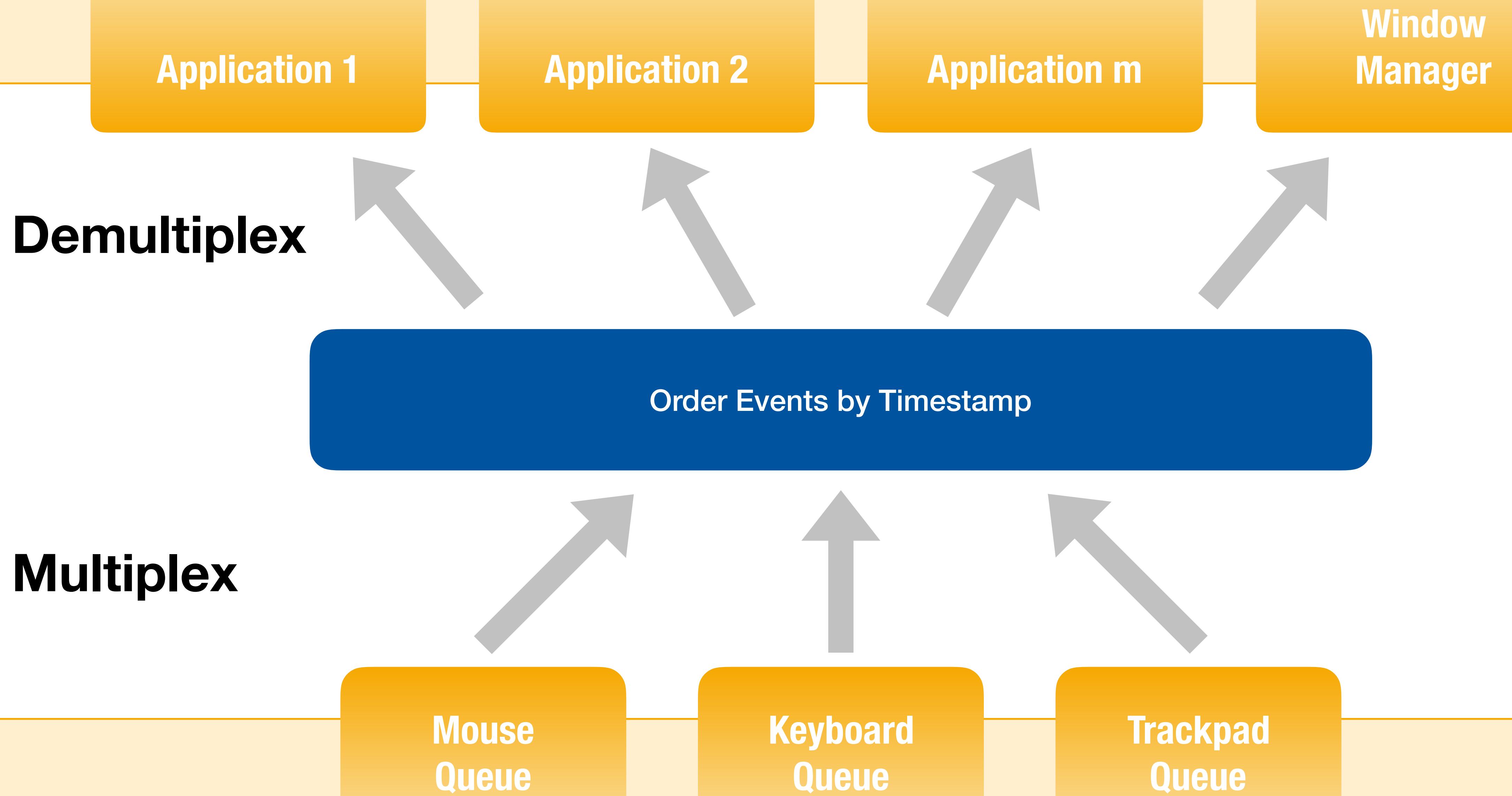
Events

Components

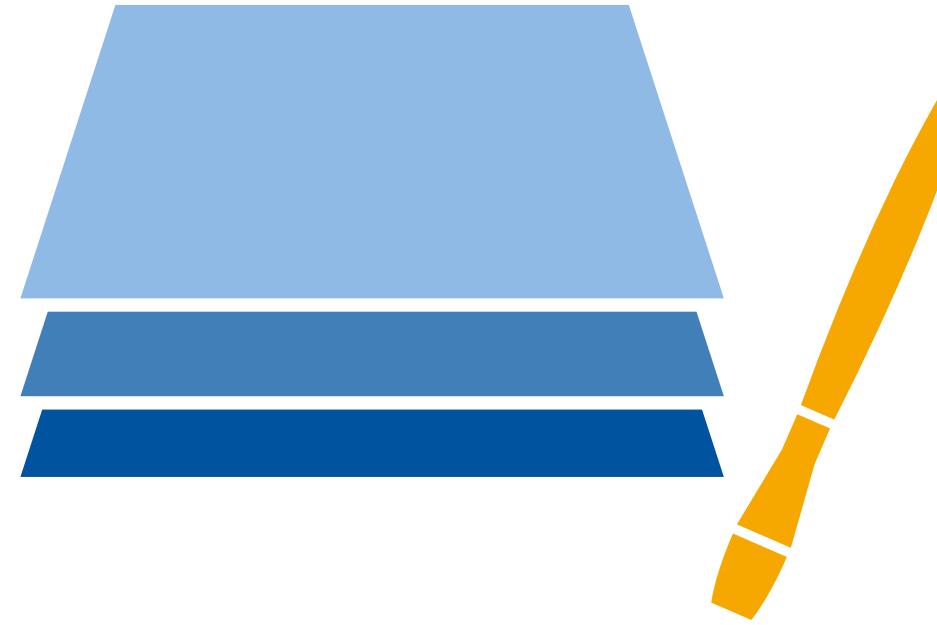
- Event type
- Timestamp
- Type-specific data
- Location
- Window
- Application

Operations

- Read, write and filter events



Base Window System: Objects

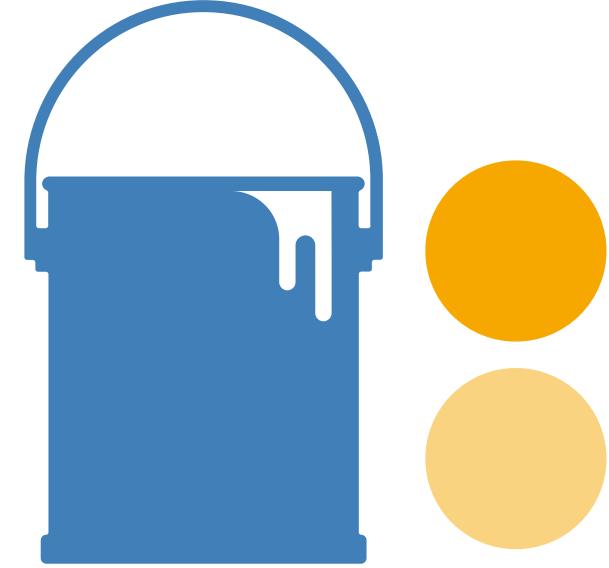


Graphics Context

Components

- Owner app, user apps
- Graphics attributes (line thickness, color index, copy function,...)
- Text attributes (color, skew, direction, copy function,...)
- Color table reference

Base Window System: Objects



Color Tables

Components

- Owner app, user apps
- Data fields for each color entry
- RGB, HSV, YIQ,...

Operations

- Provide default color
- Find close replacements of colors
(fault tolerance)

Base Window System: Objects

Components

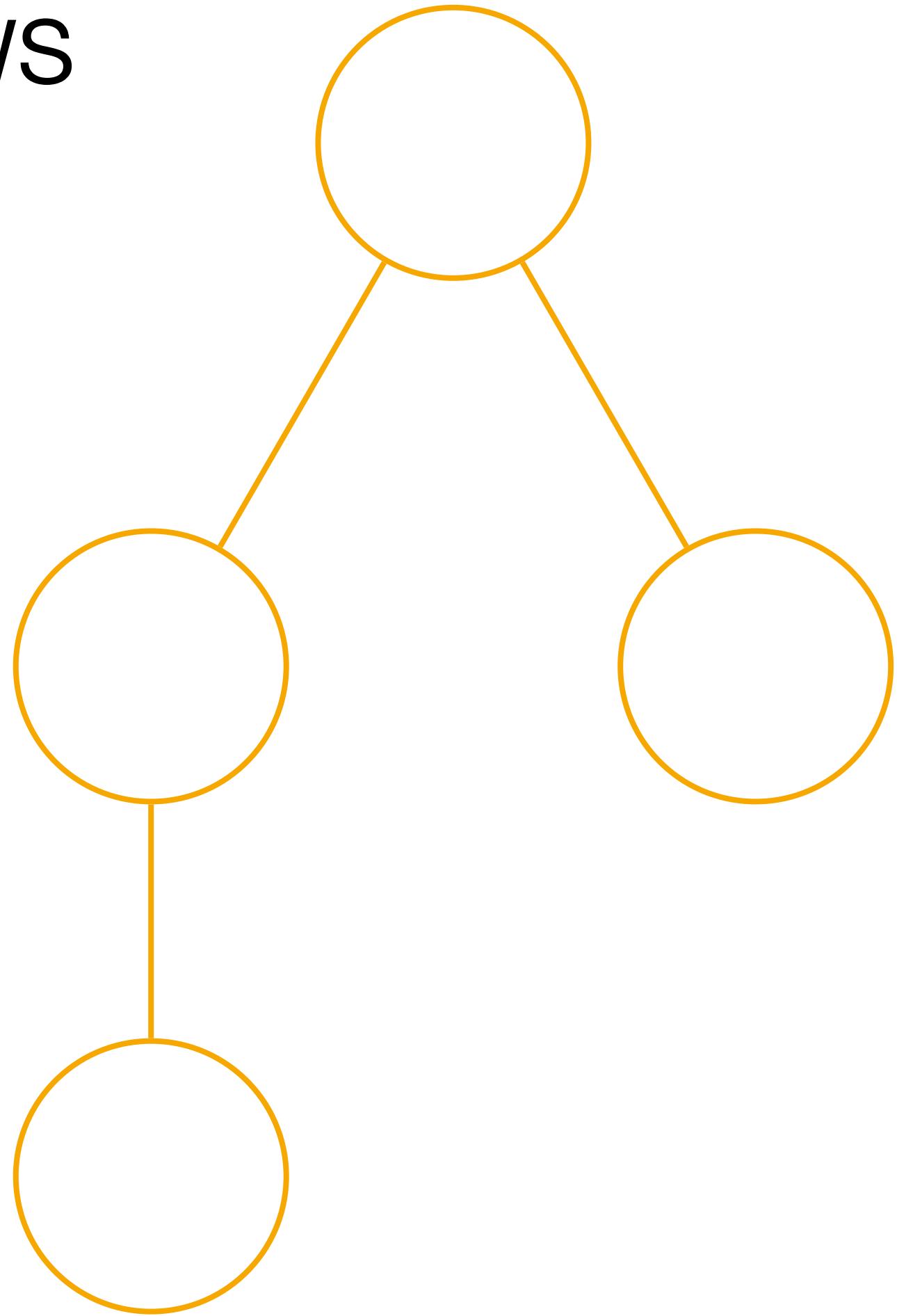
- Owner app, user apps
- Name, measurements
(font size, kerning, ligatures,...)
- One data field per character

f1

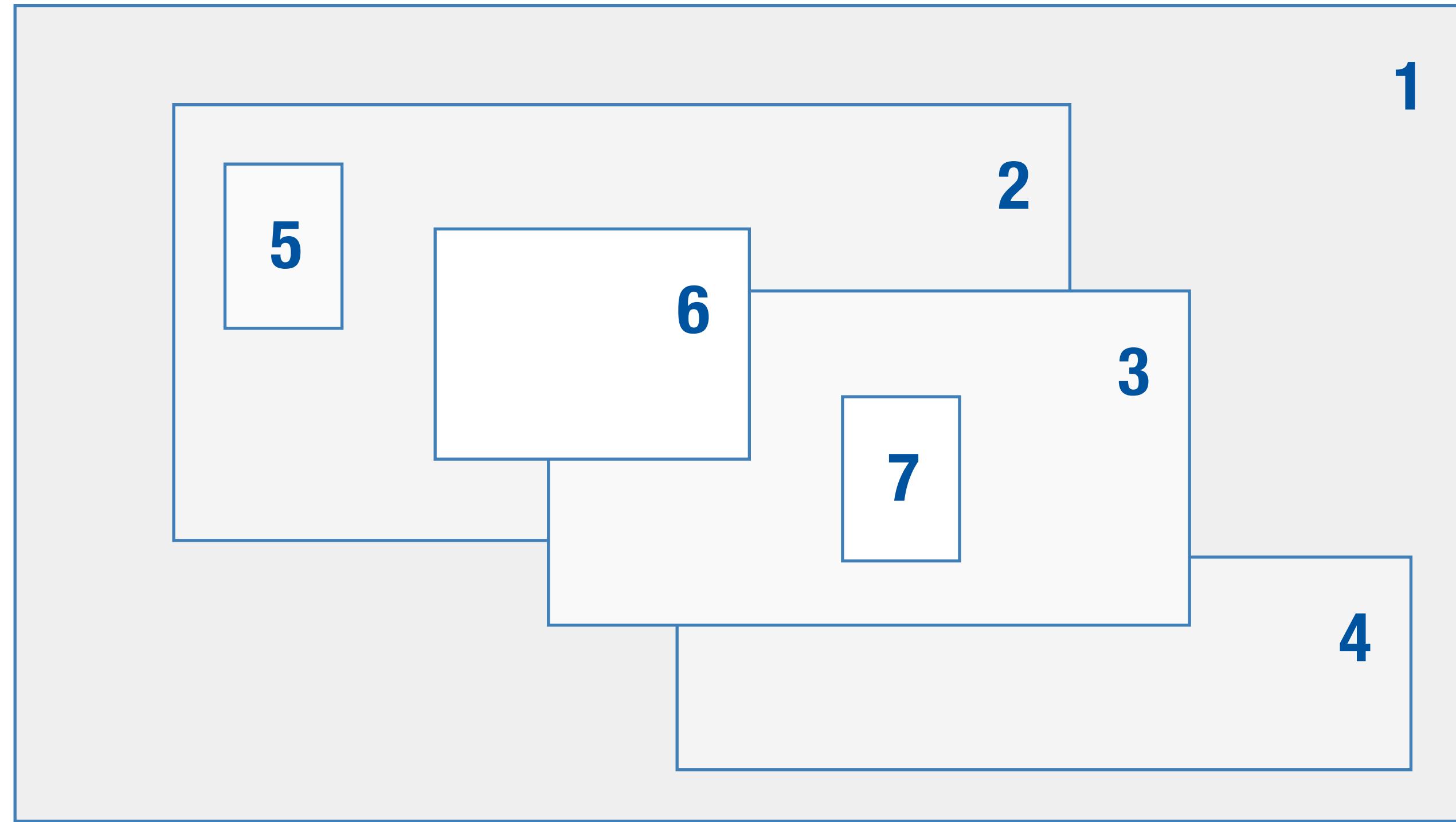
Fonts

Base Window Management

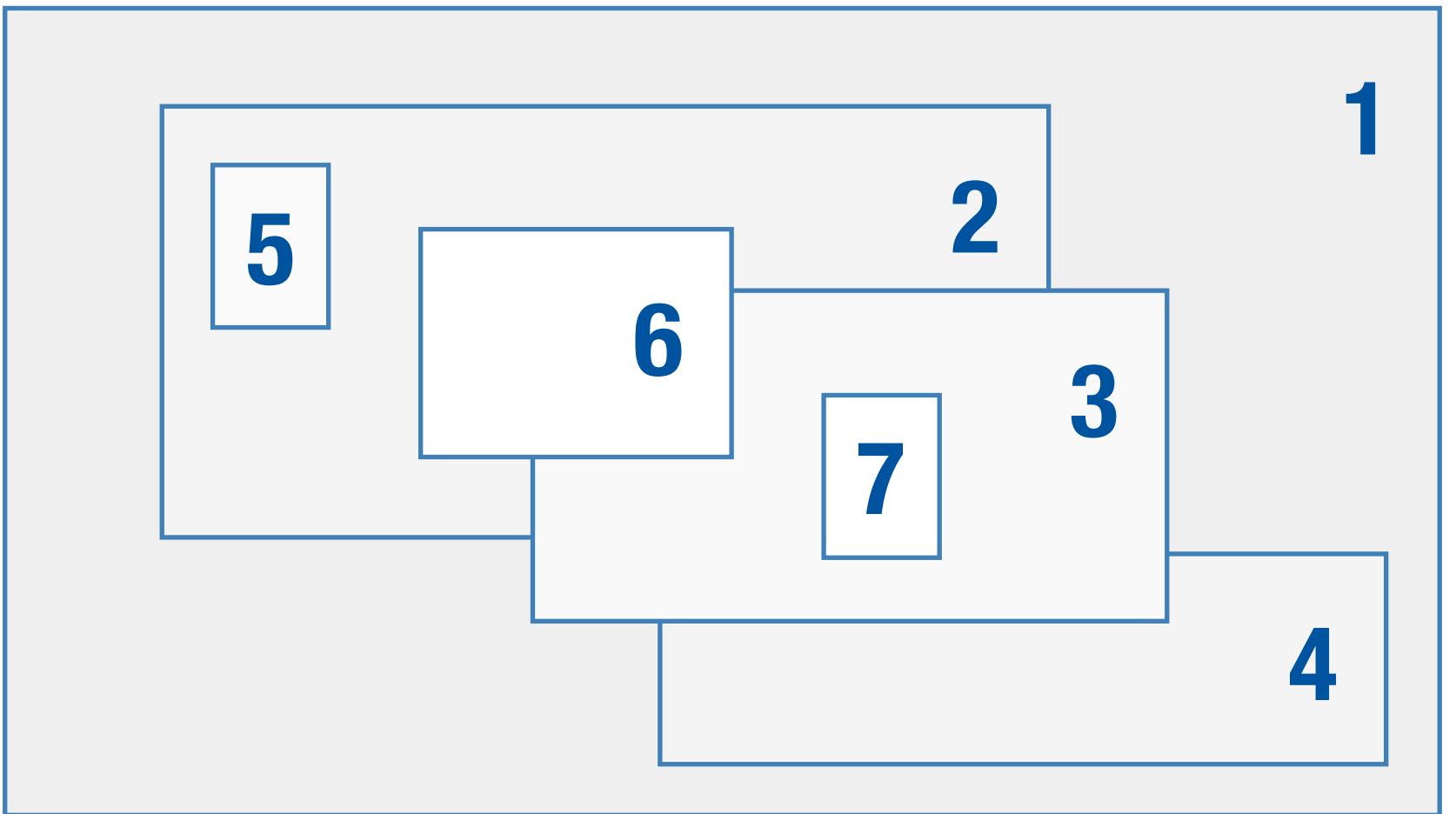
- Trees are used to manage the window collection in the BWS
 - All child windows are inside their parent window
 - Simplifies event routing or setting visibility



Base Window Management



Base Window Management



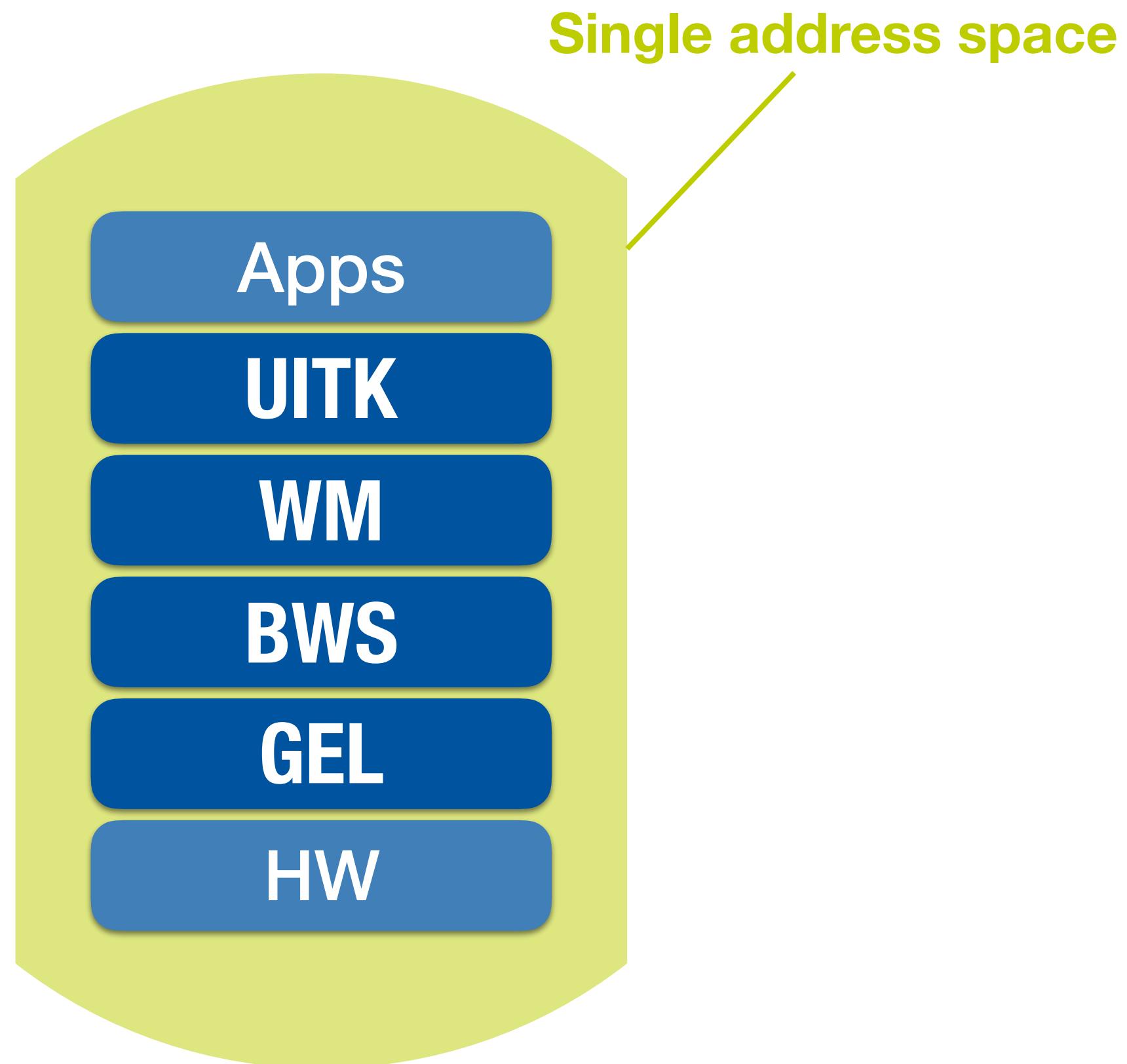
Base Window System: Shared Resources

- Reasons for sharing resources: Scarcity, collaboration
- **Problems:** Competition, consistency

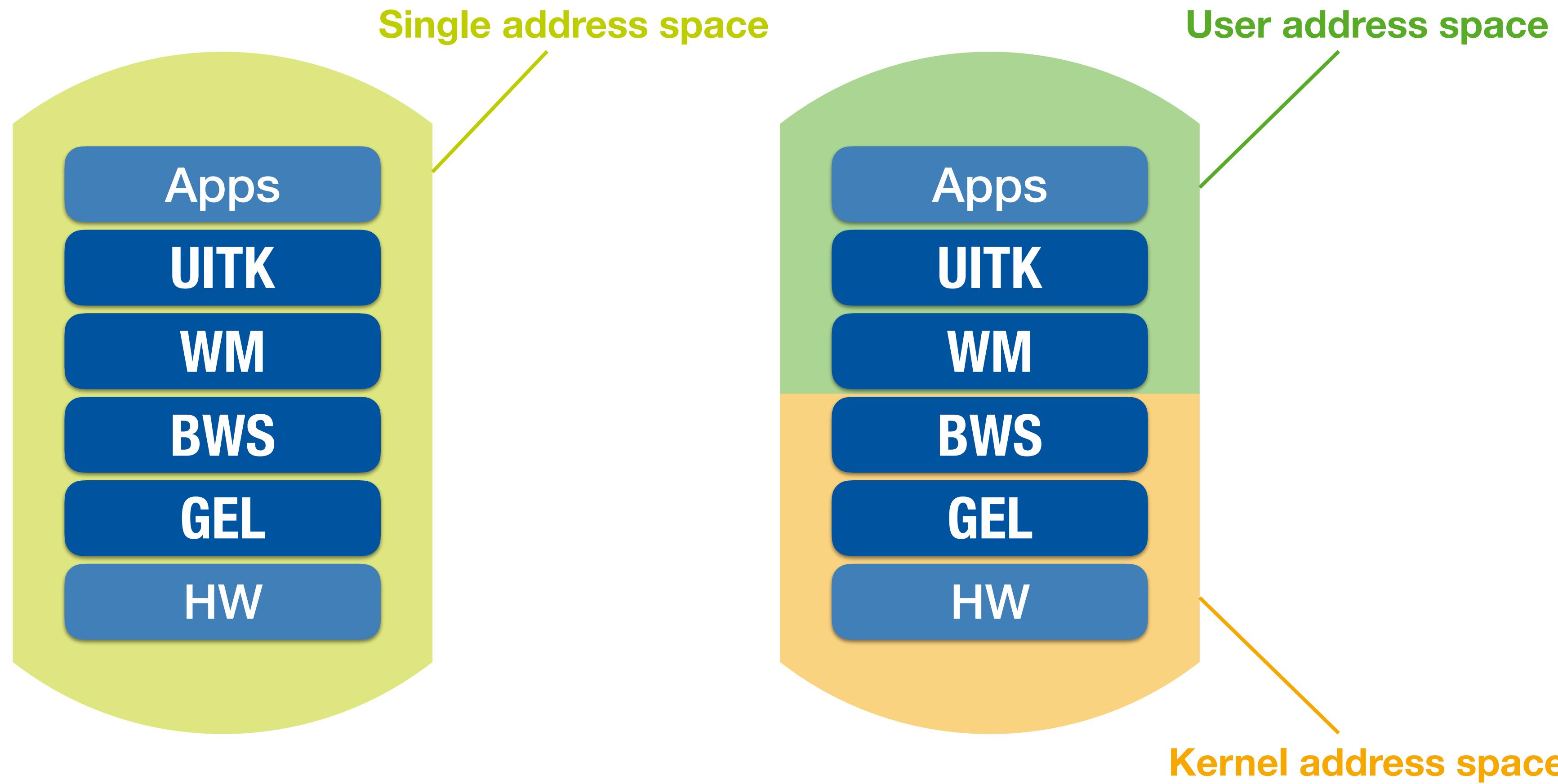
Base Window System: Shared Resources

- Reasons for sharing resources: Scarcity, collaboration
- **Problems:** Competition, consistency
- **Synchronization** needed
 - At BWS entrance
 - Or on individual objects

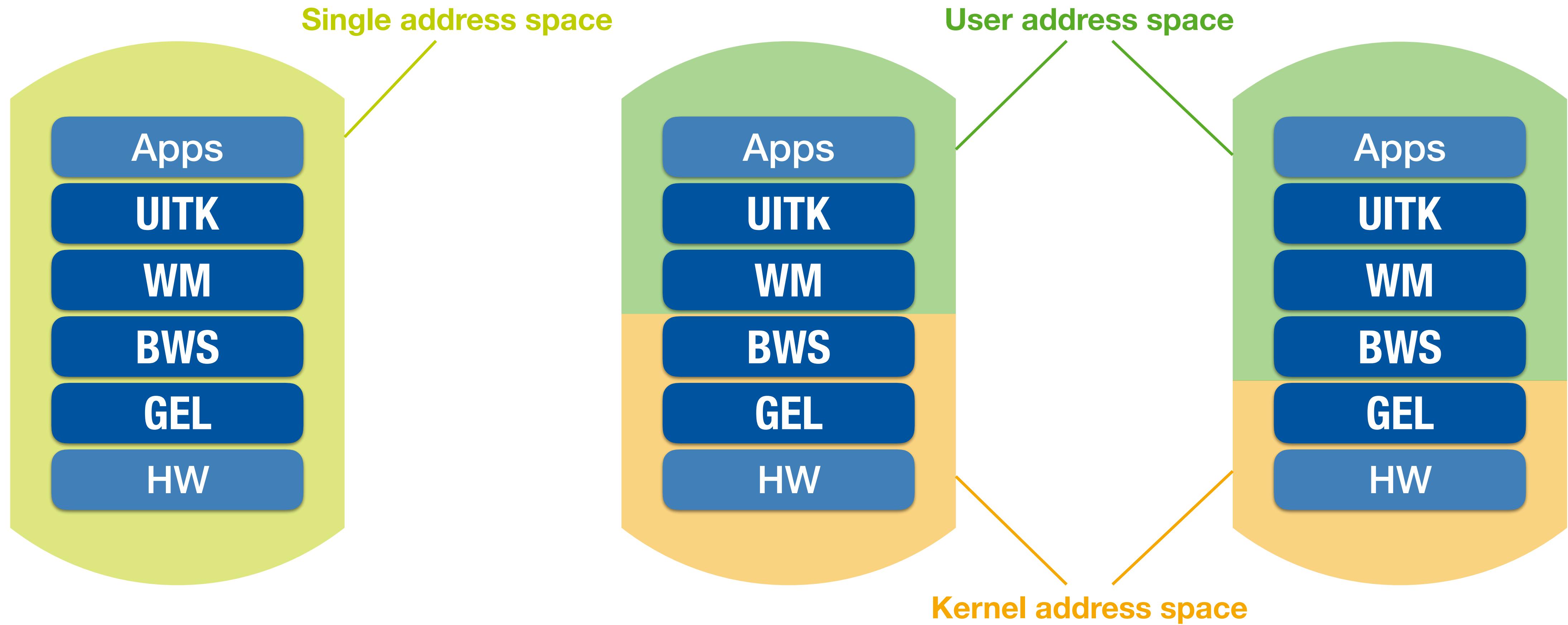
Base Window System: OS Integration



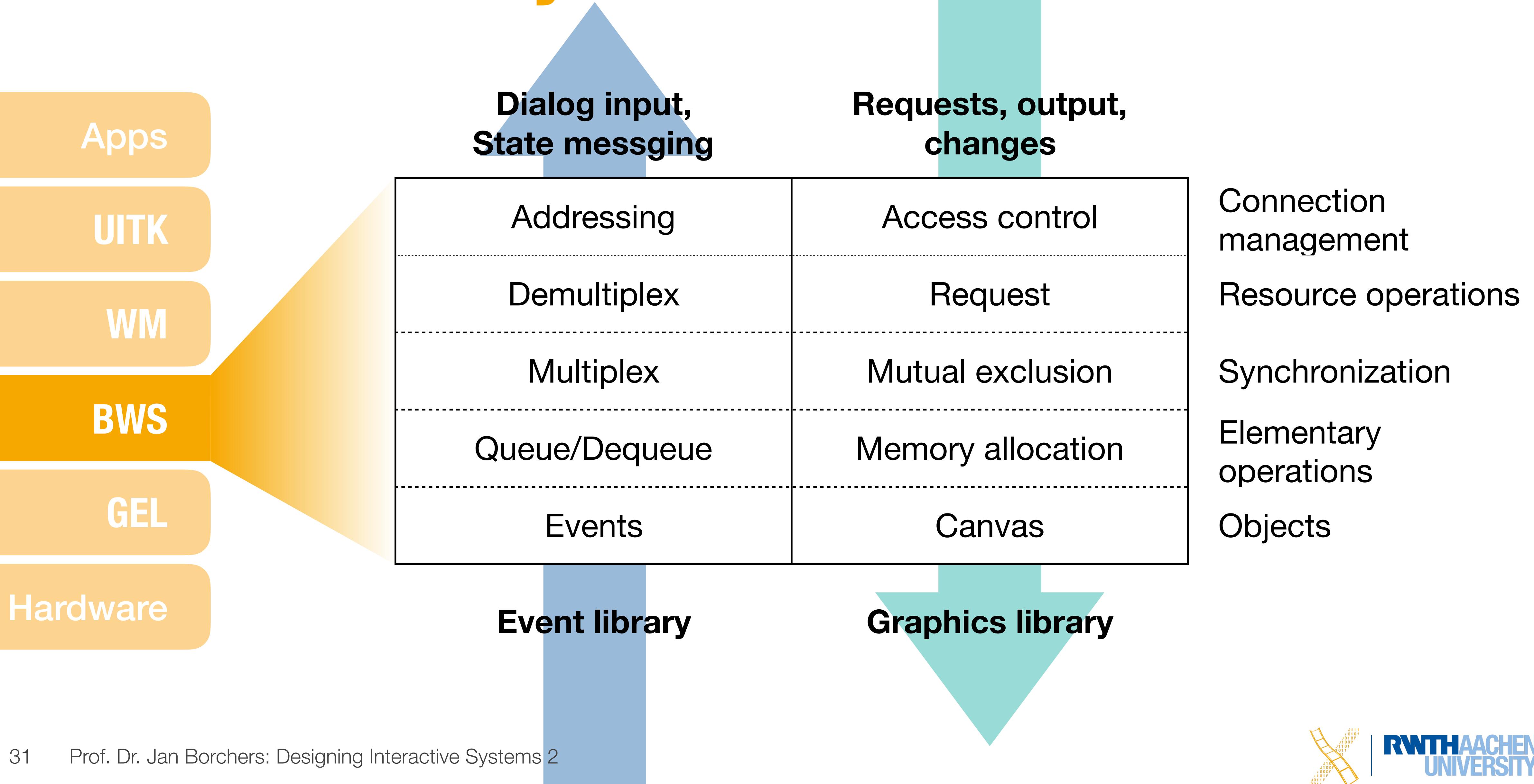
Base Window System: OS Integration



Base Window System: OS Integration



Base Window System



CHAPTER 6

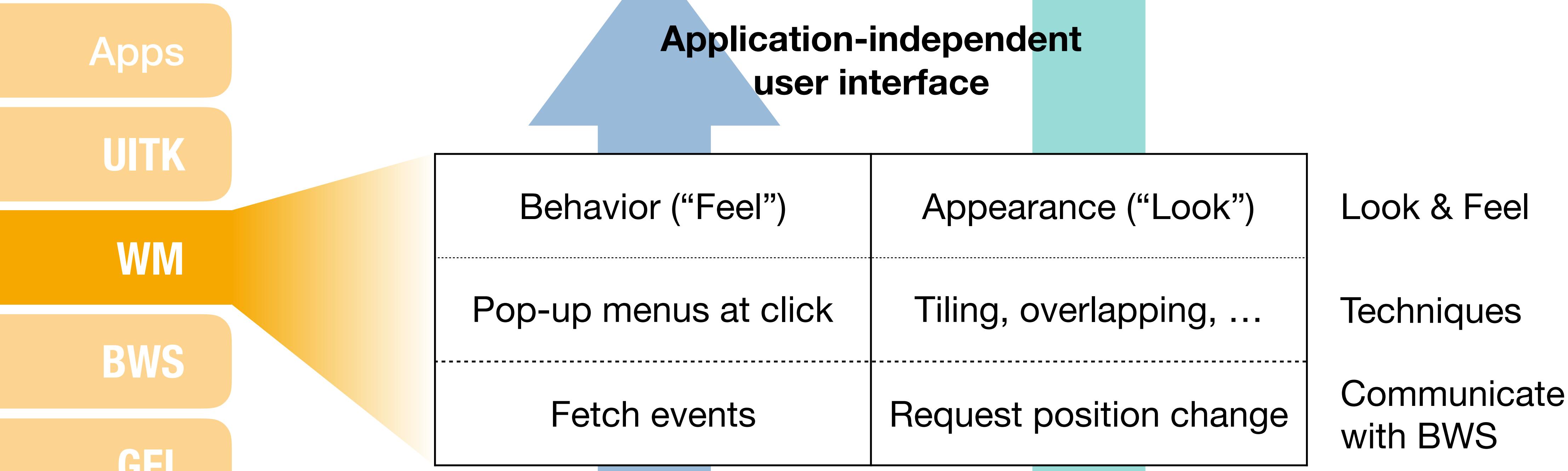
Window Manager

Window Manager

- Allow users to control windows
- Provide Look & Feel for interaction with the WS



Window Manager



Window Manager

Screen Management

- What is rendered where on screen?
- Where is empty space? Which apps are iconified?

Window Manager

Screen Management

- What is rendered where on screen?
- Where is empty space? Which apps are iconified?

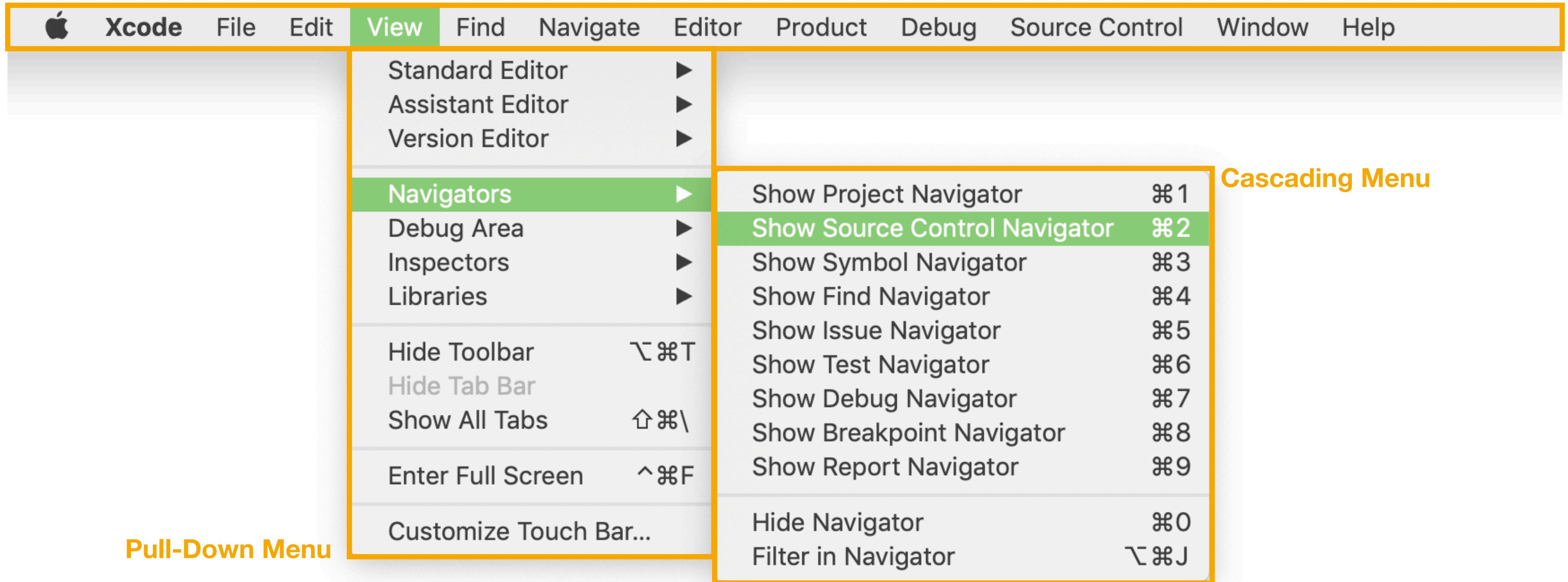
Session Management

- Provide consistent ways to perform standard tasks
- Move window, start app, iconify window

Session Management

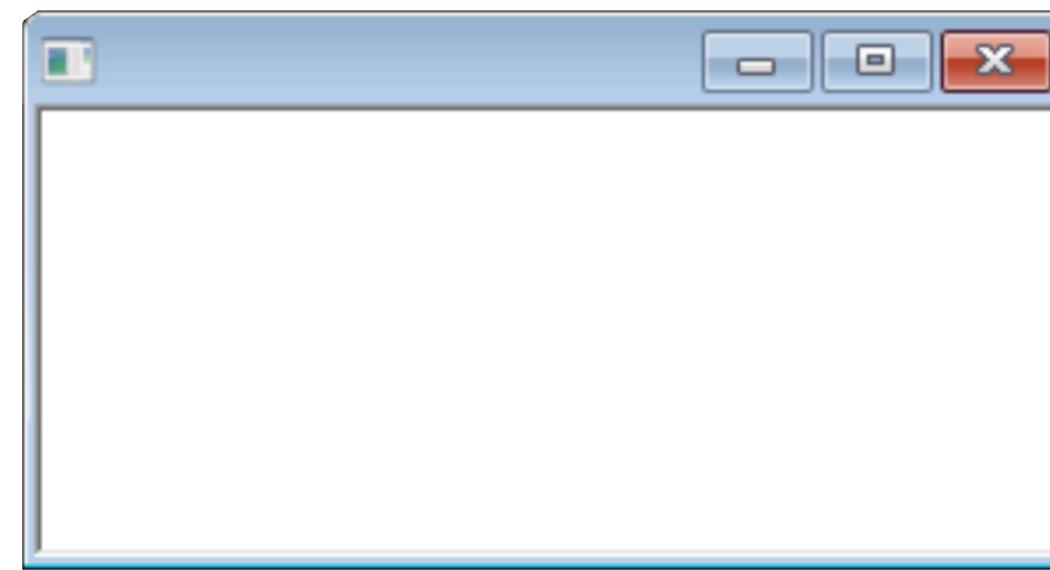
Menu Techniques

Fixed Menu Bar

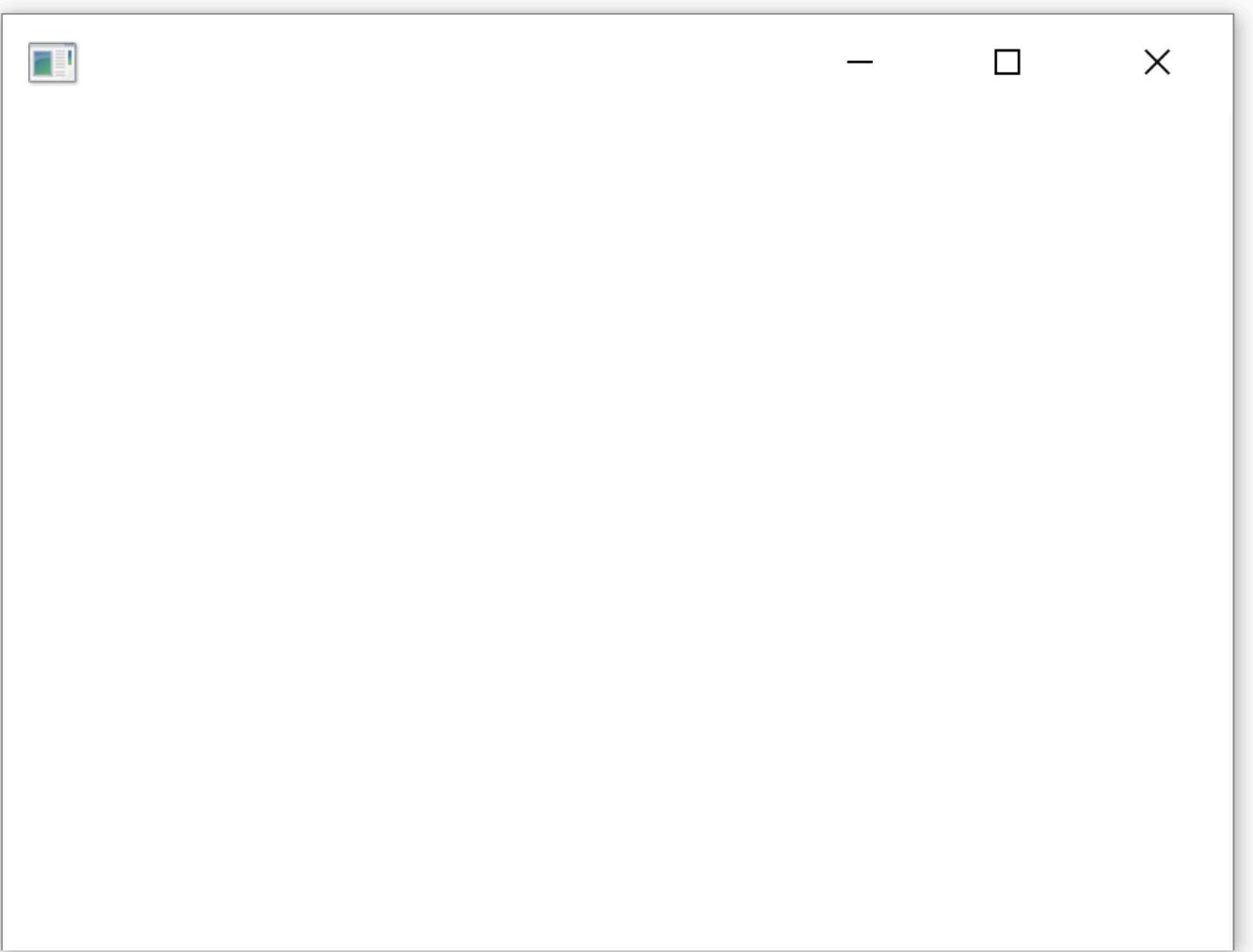


Session Management

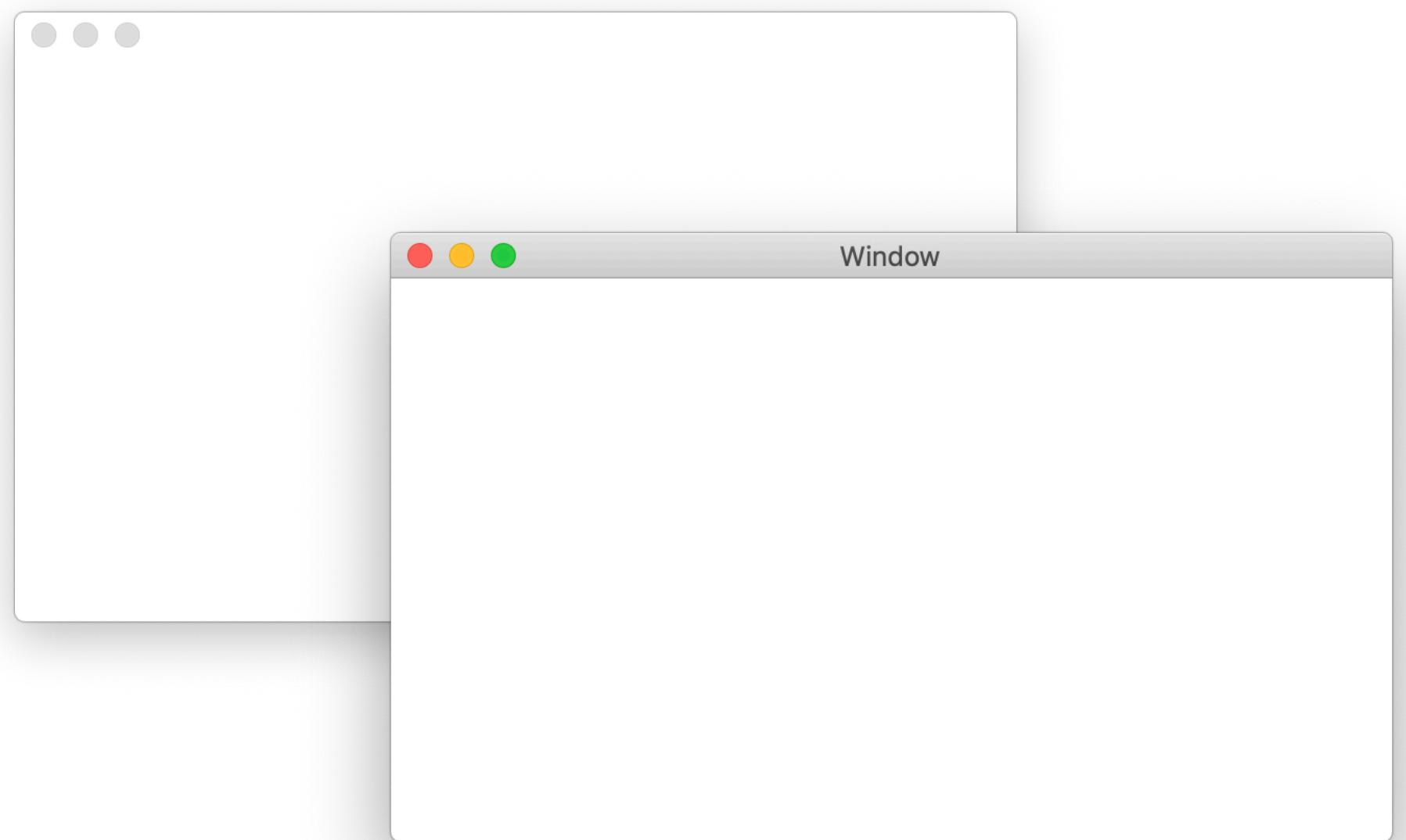
Window Decorations



Windows 7



Windows 10



macOS

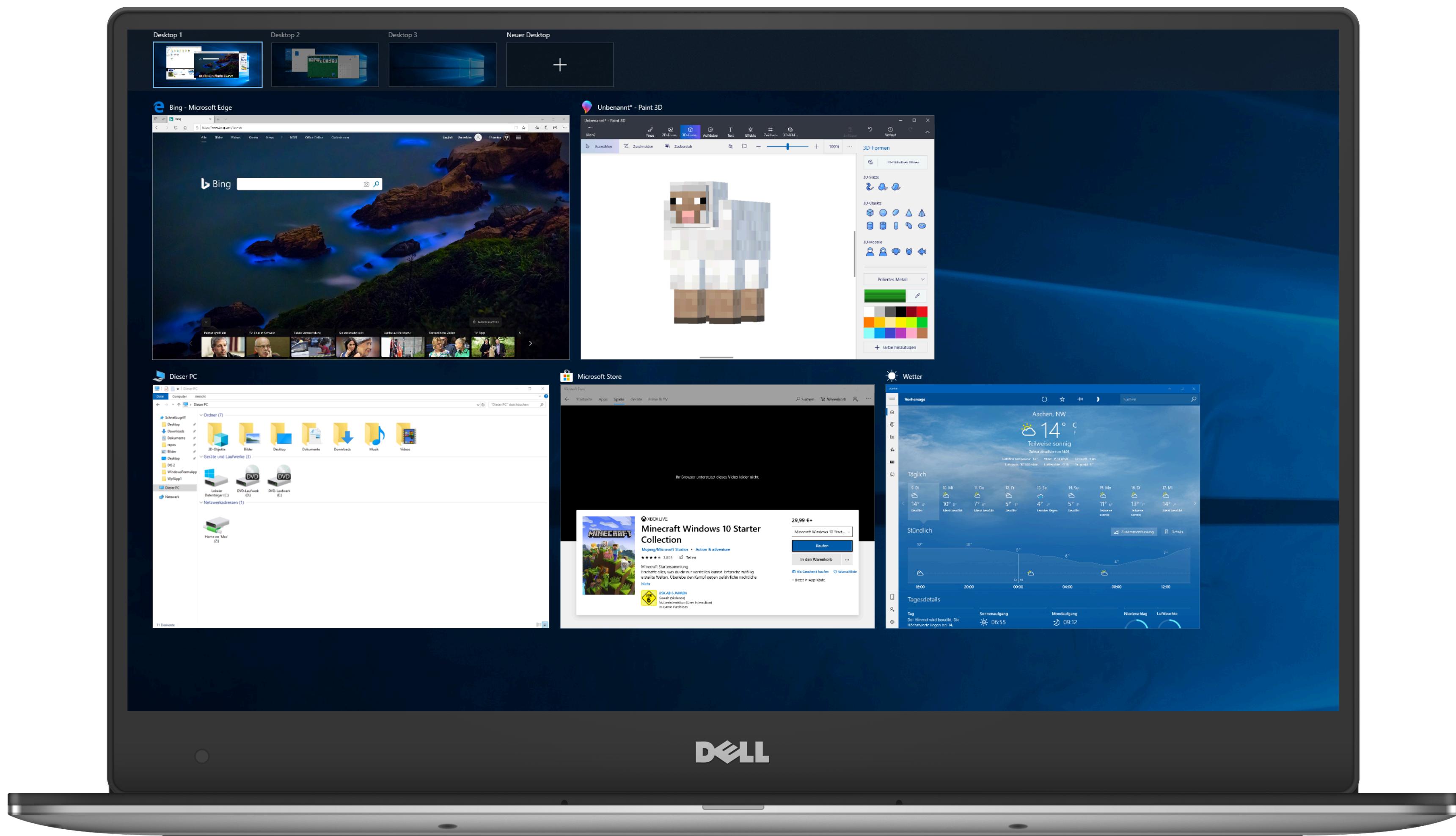
Session Management

Layout Policy



Session Management

Virtual Desktops



Session Management

Direct manipulation

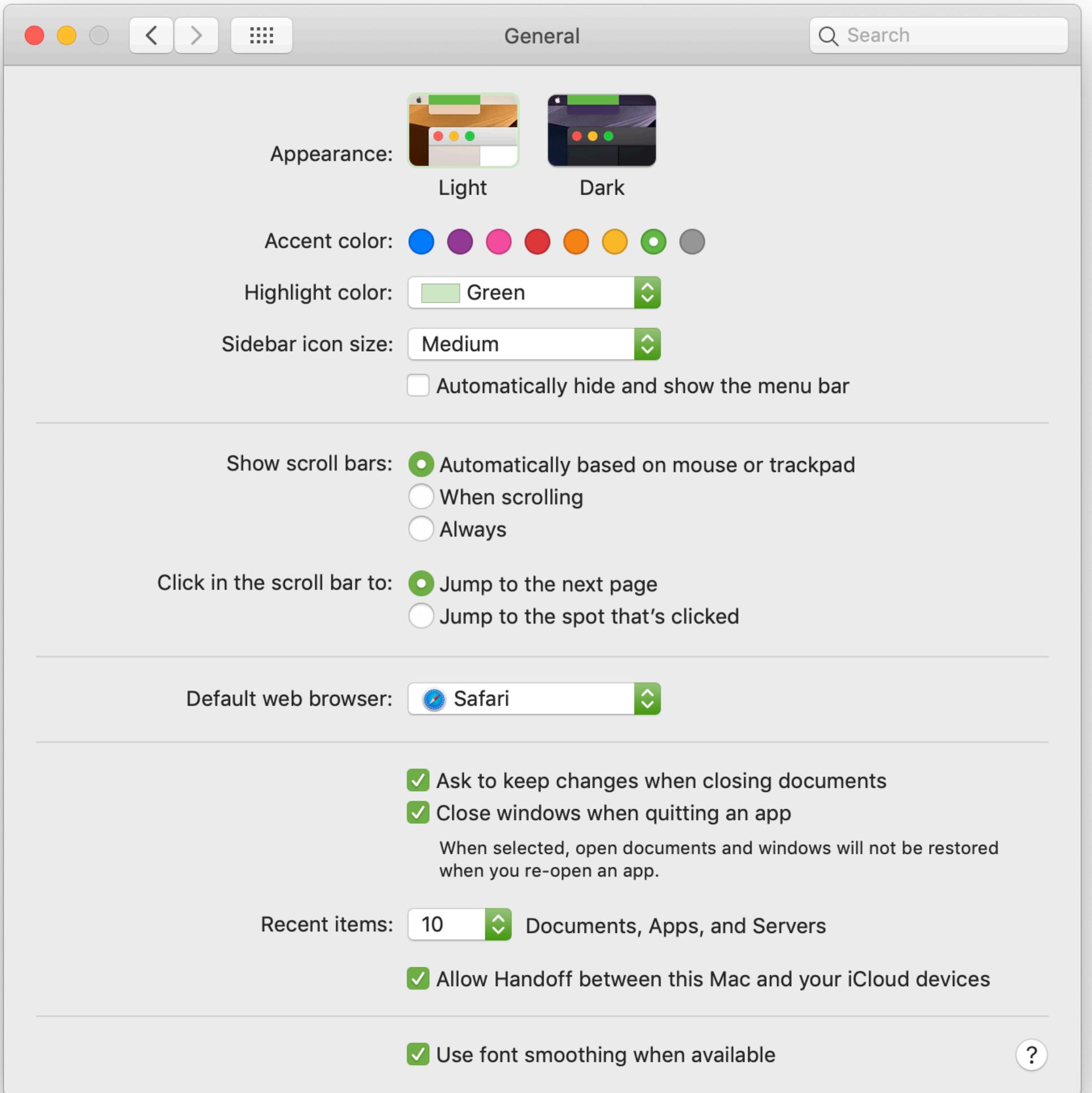
Icon technique

Input focus

Look & Feel

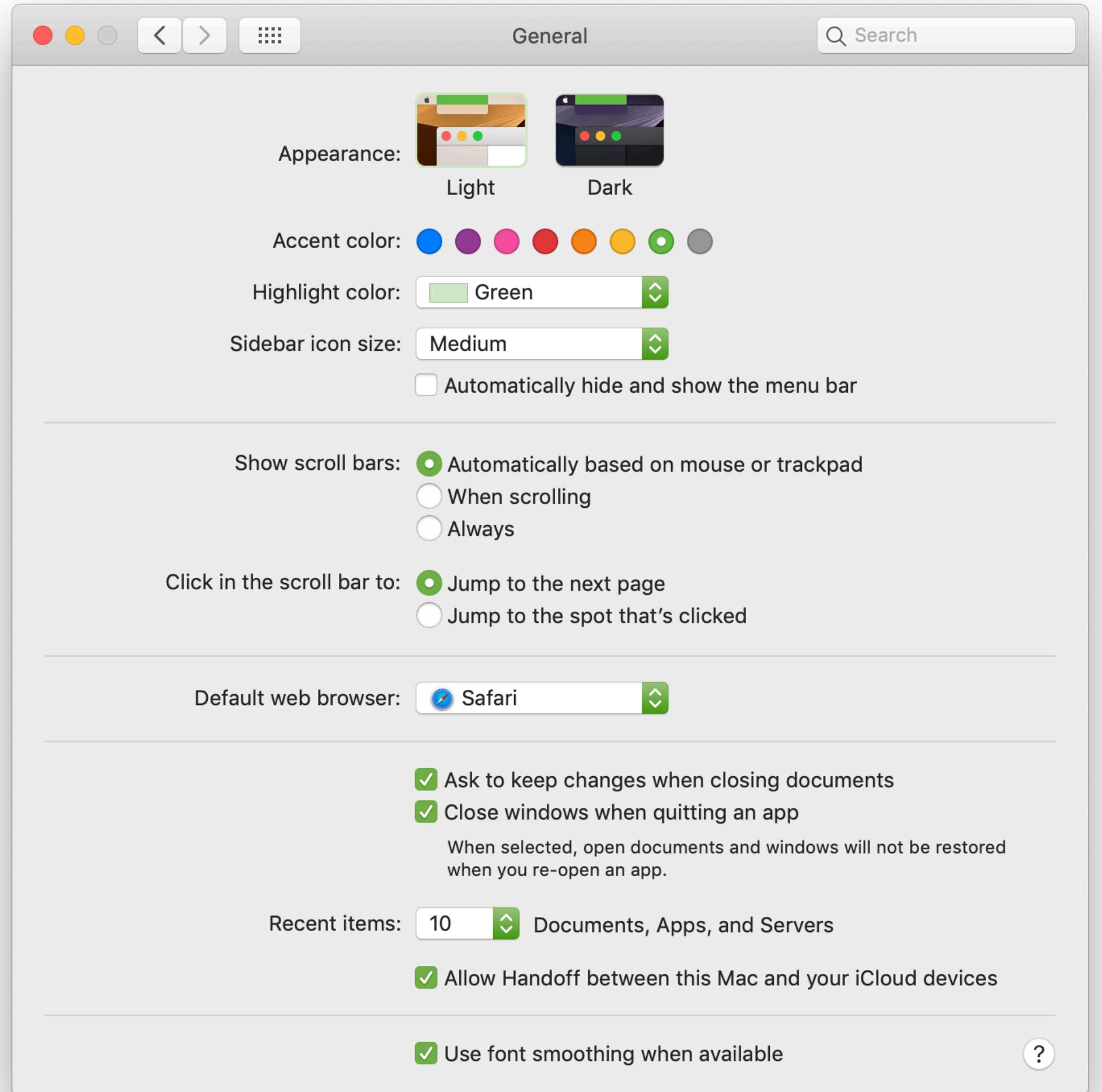
Late Refinement

- WM accompanies the session
 - changing window positions
 - changing app appearance



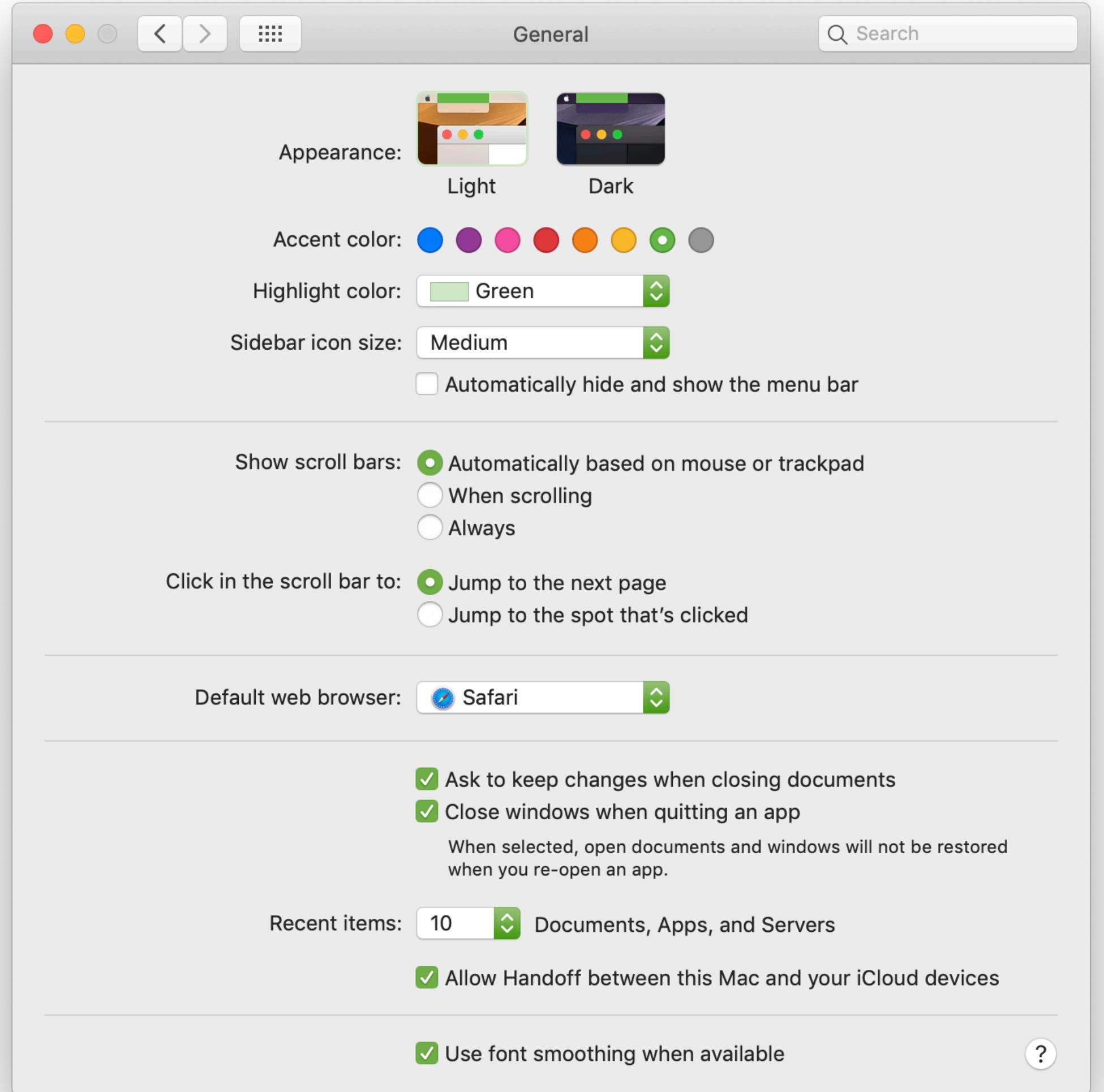
Late Refinement

- WM accompanies the session
 - changing window positions
 - changing app appearance
- Levels of late refinement
 - Per session or application
 - Per launch, user or for all users



Late Refinement

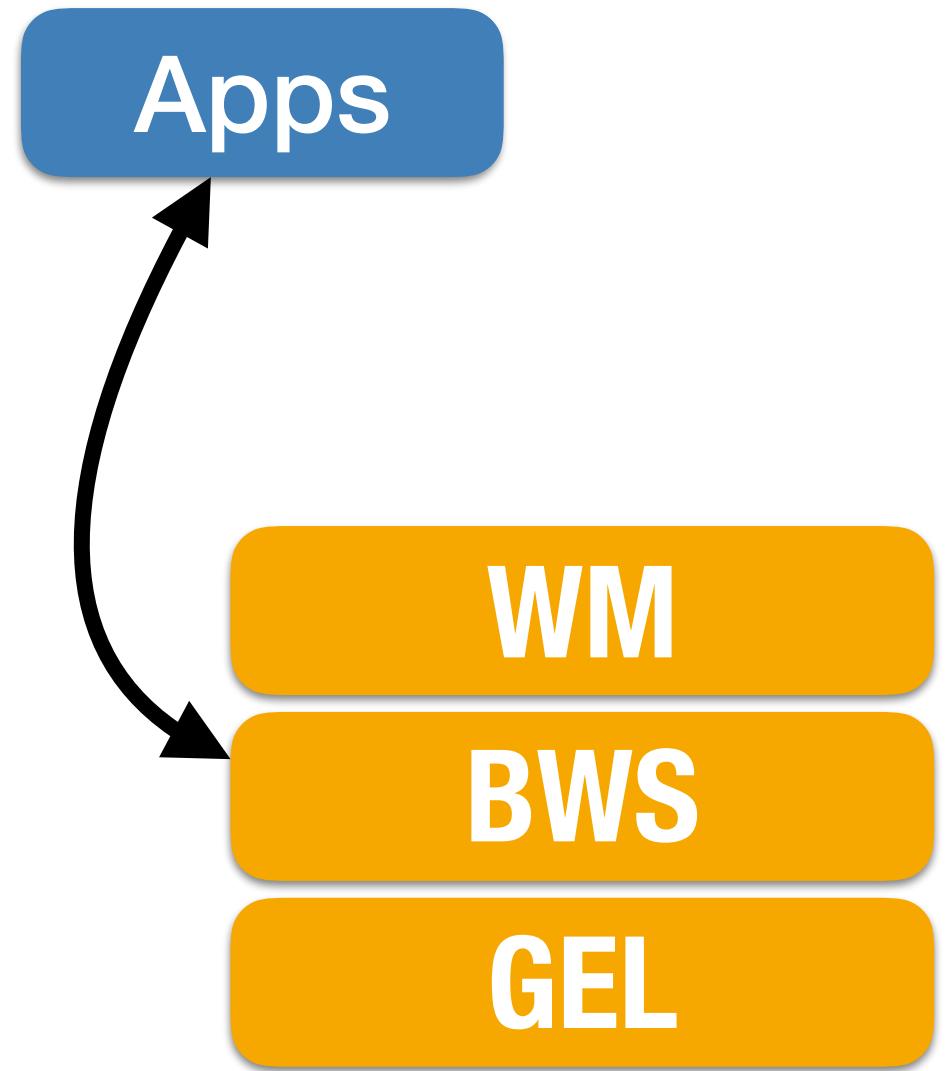
- WM accompanies the session
 - changing window positions
 - changing app appearance
- Levels of late refinement
 - Per session or application
 - Per launch, user or for all users
- Implementation with table files, internal database, or delta technique



Late Refinement: macOS Login Window

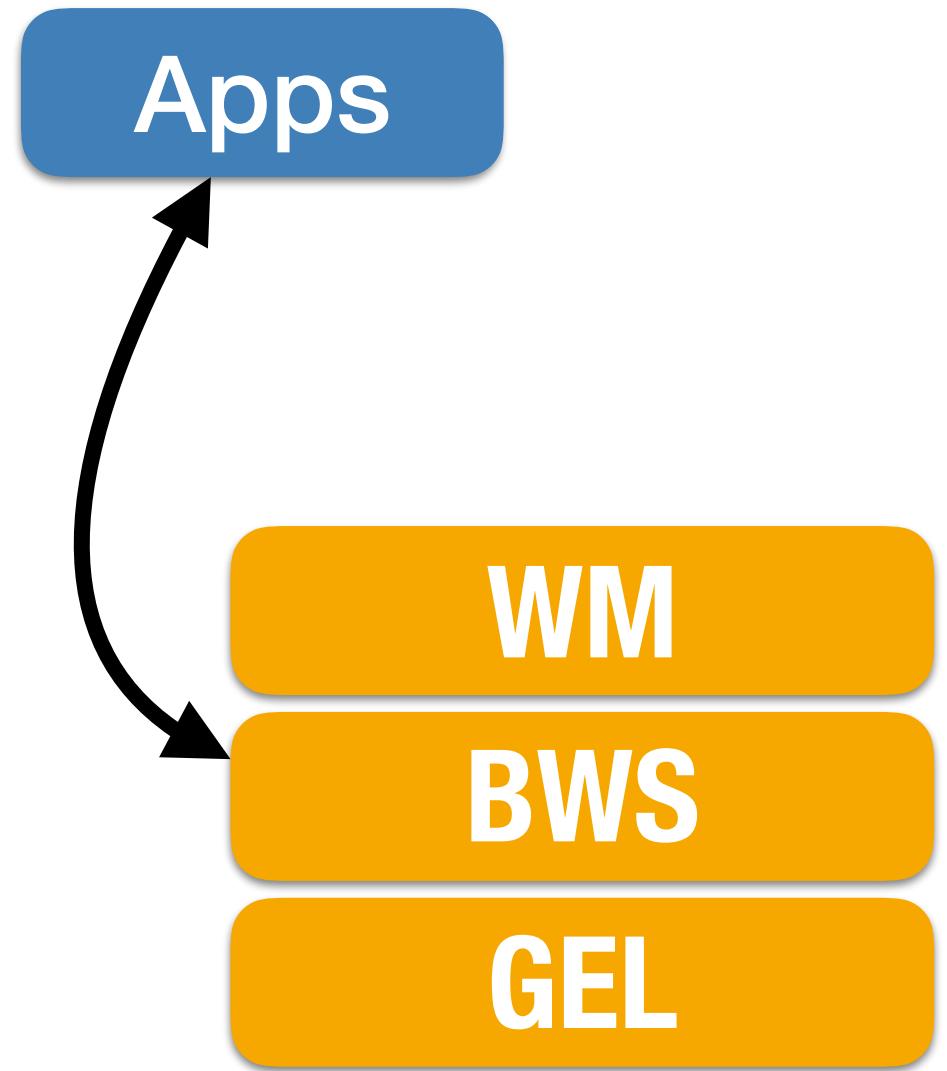
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple/DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>GuestEnabled</key>
    <false/>
    <key>lastUser</key>
    <string>loggedIn</string>
    <key>lastUserName</key>
    <string>borchers</string>
    <key>retriesUntilHint</key>
    <integer>3</integer>
</dict>
</plist>
```

Window Manager: Location

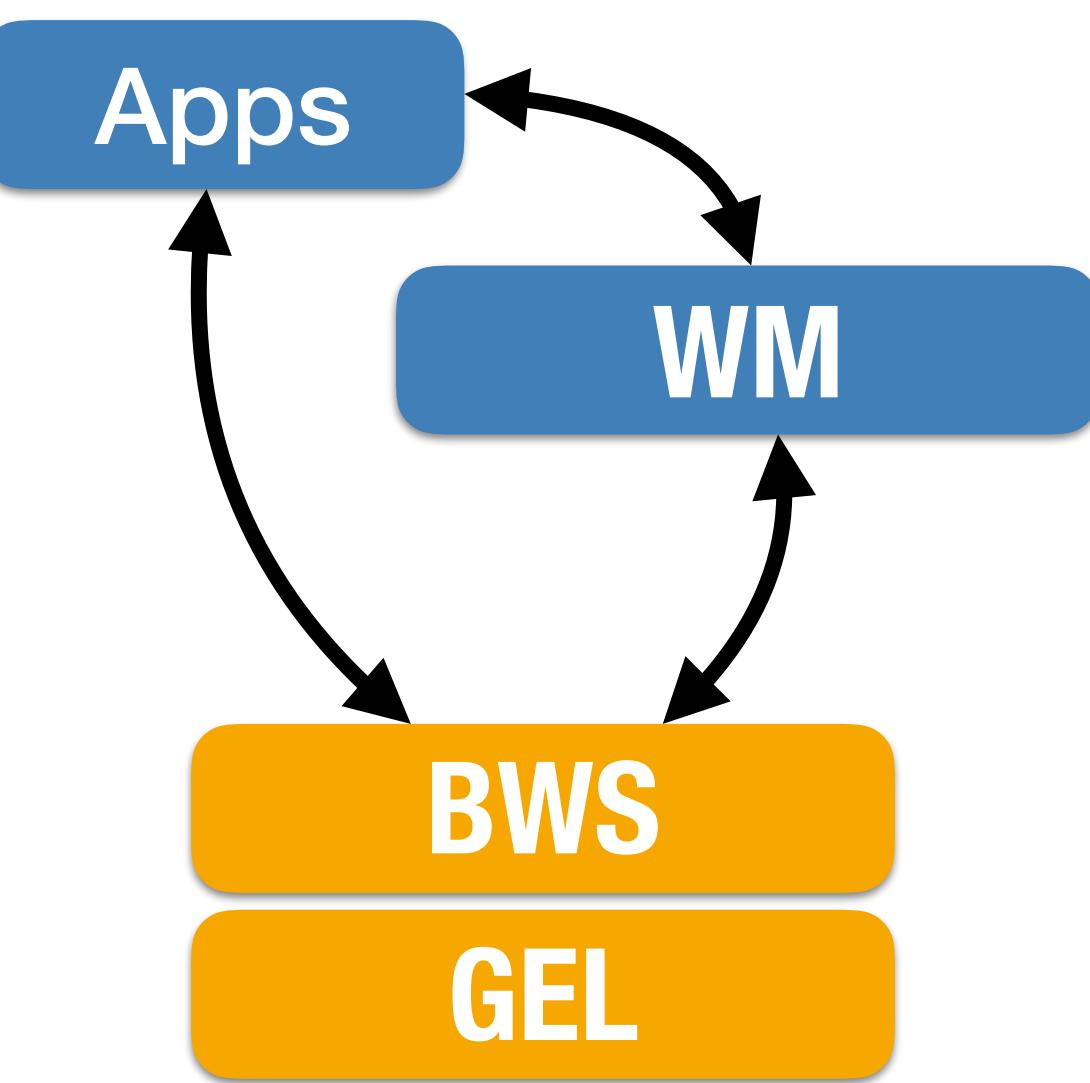


Upper part of BWS

Window Manager: Location

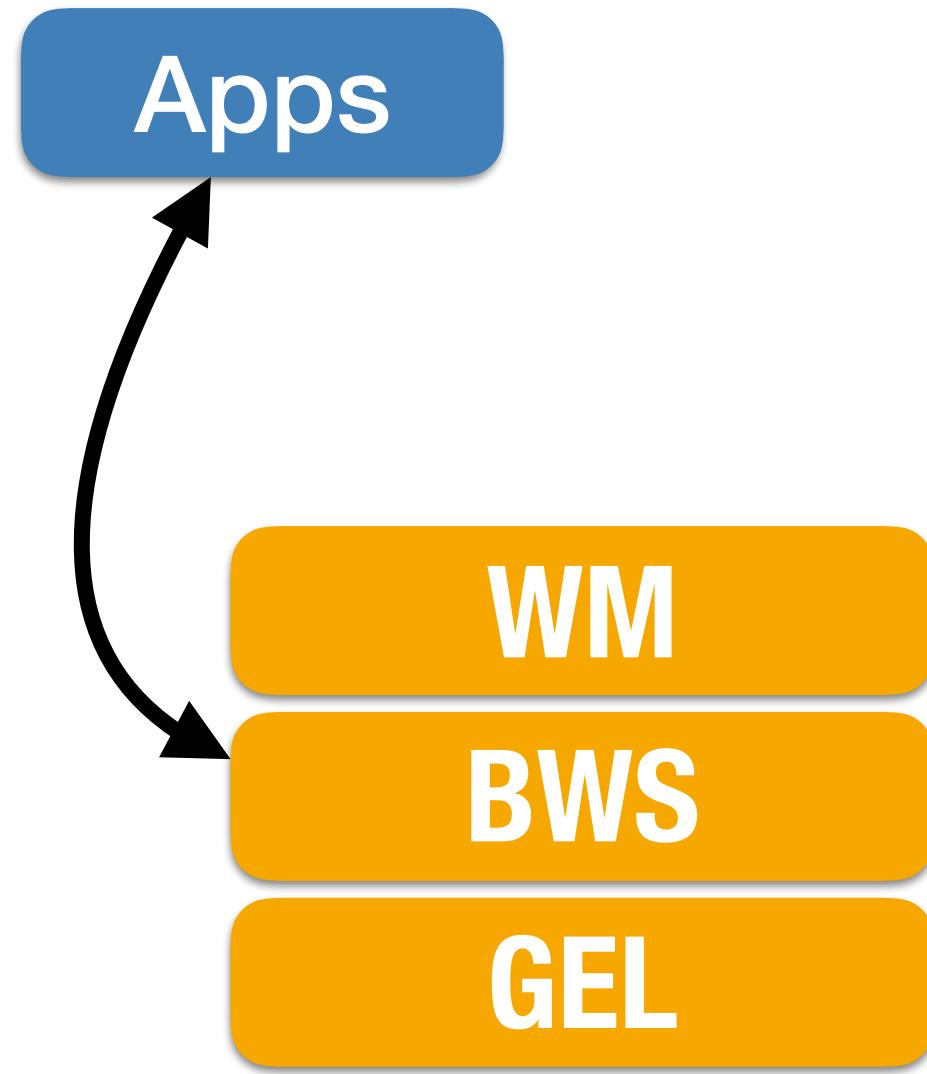


Upper part of BWS

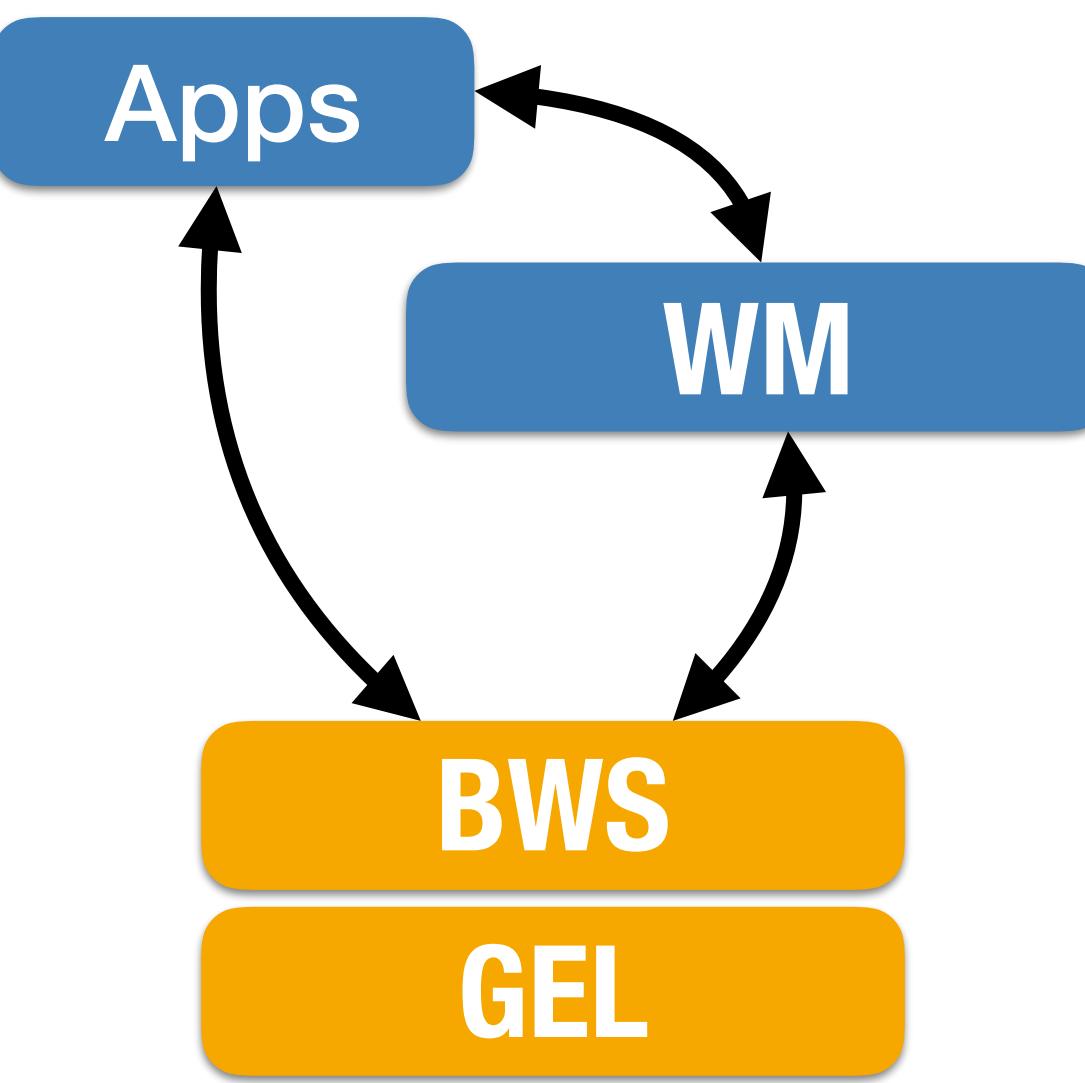


Separate server

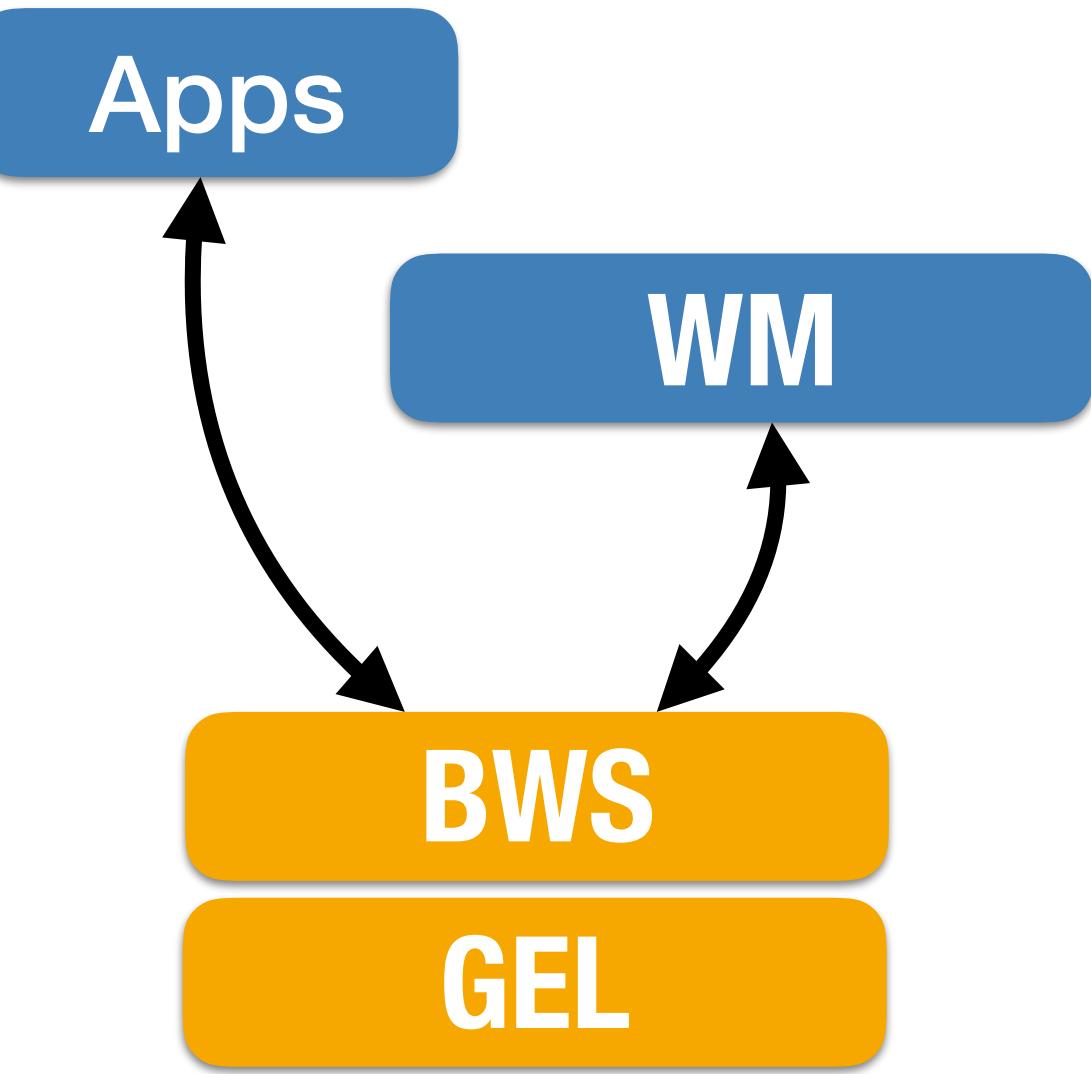
Window Manager: Location



Upper part of BWS



Separate server

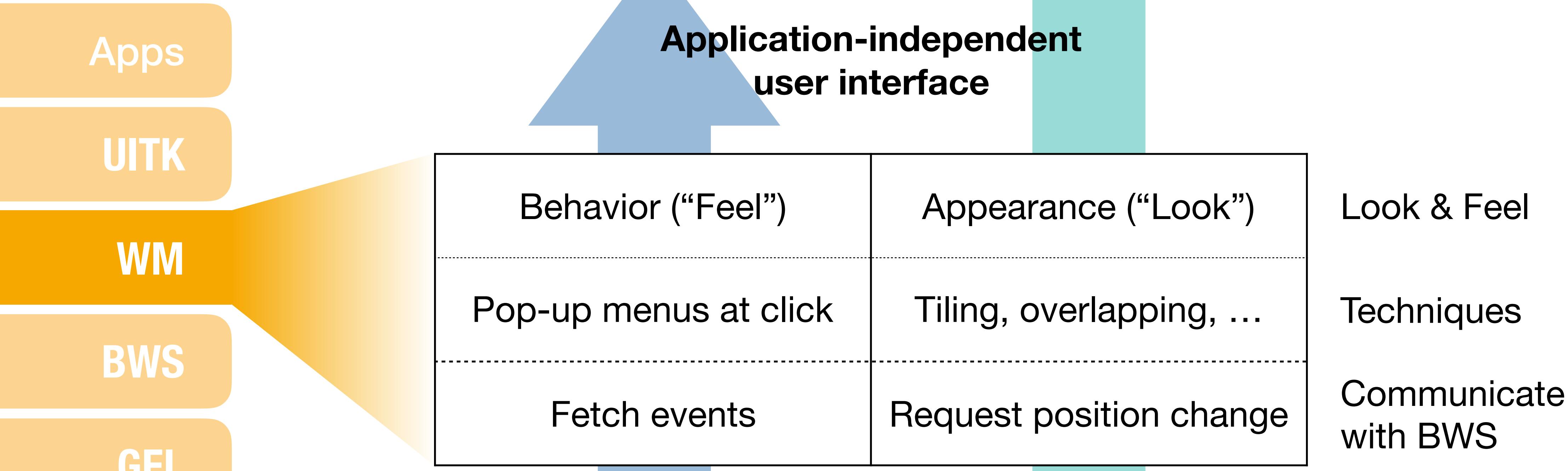


Separate user process

Window Manager: Conclusions

- WM leads from system- to **user-centered view** of WS
- Provide different levels of **consistency**
- Accompanies user during **session**
- Potentially **exchangeable**
- WM requires UI Toolkit to implement same **Look & Feel** across applications

Window Manager



Designing Interactive Systems 2

Lecture 3: Window System Architecture

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/dis2

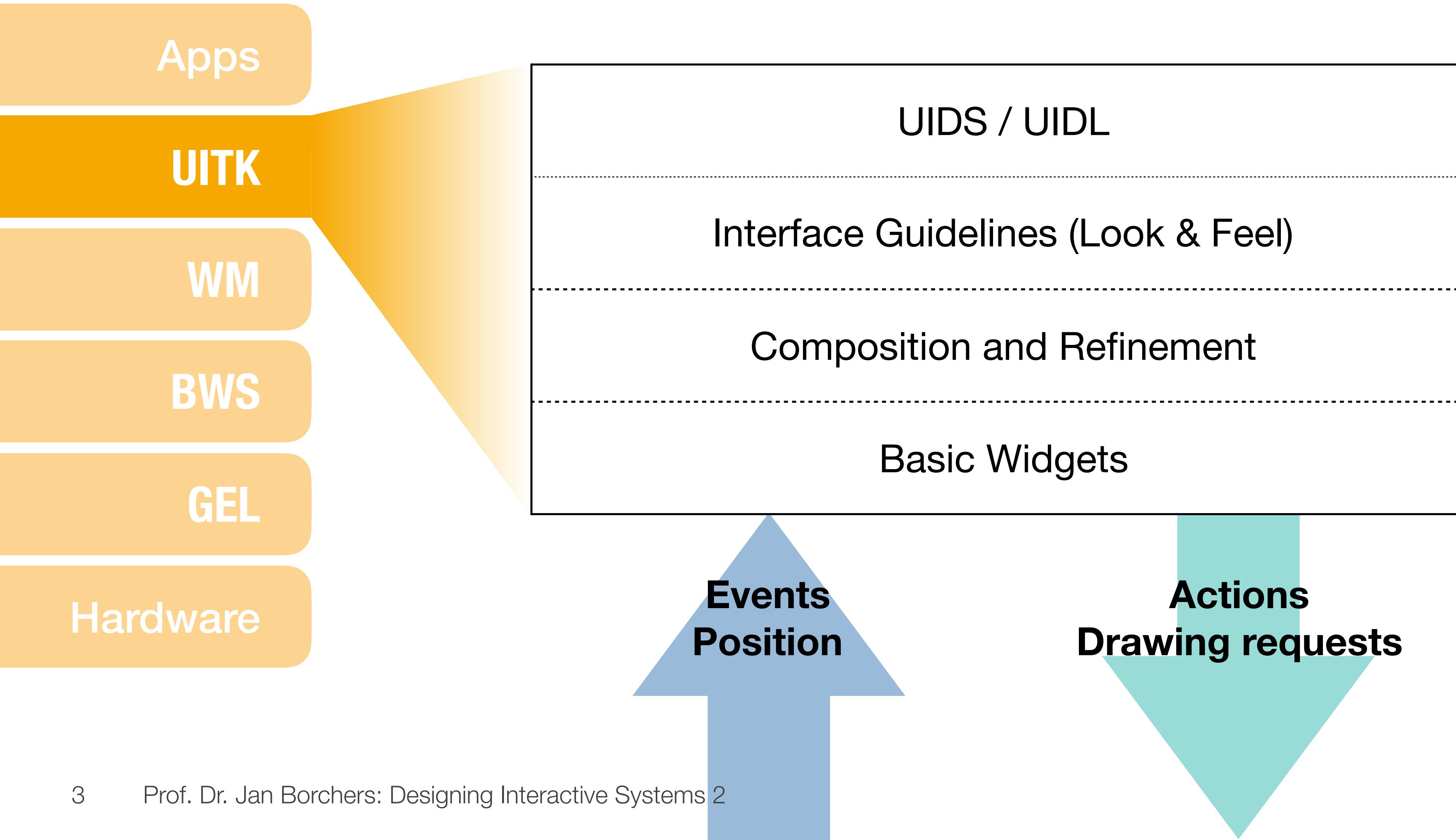


RWTHAACHEN
UNIVERSITY

CHAPTER 7

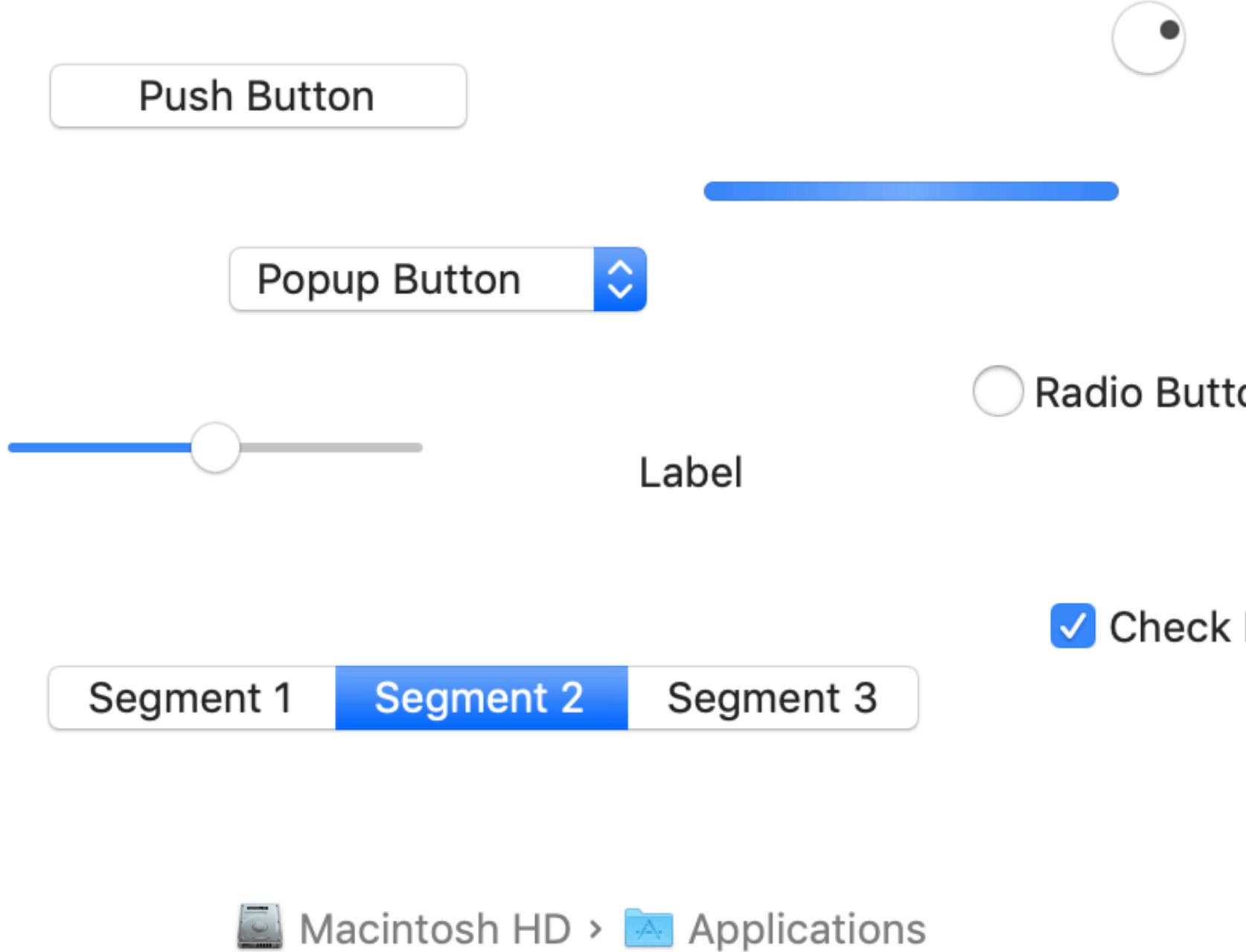
User Interface Toolkit

User Interface Toolkit

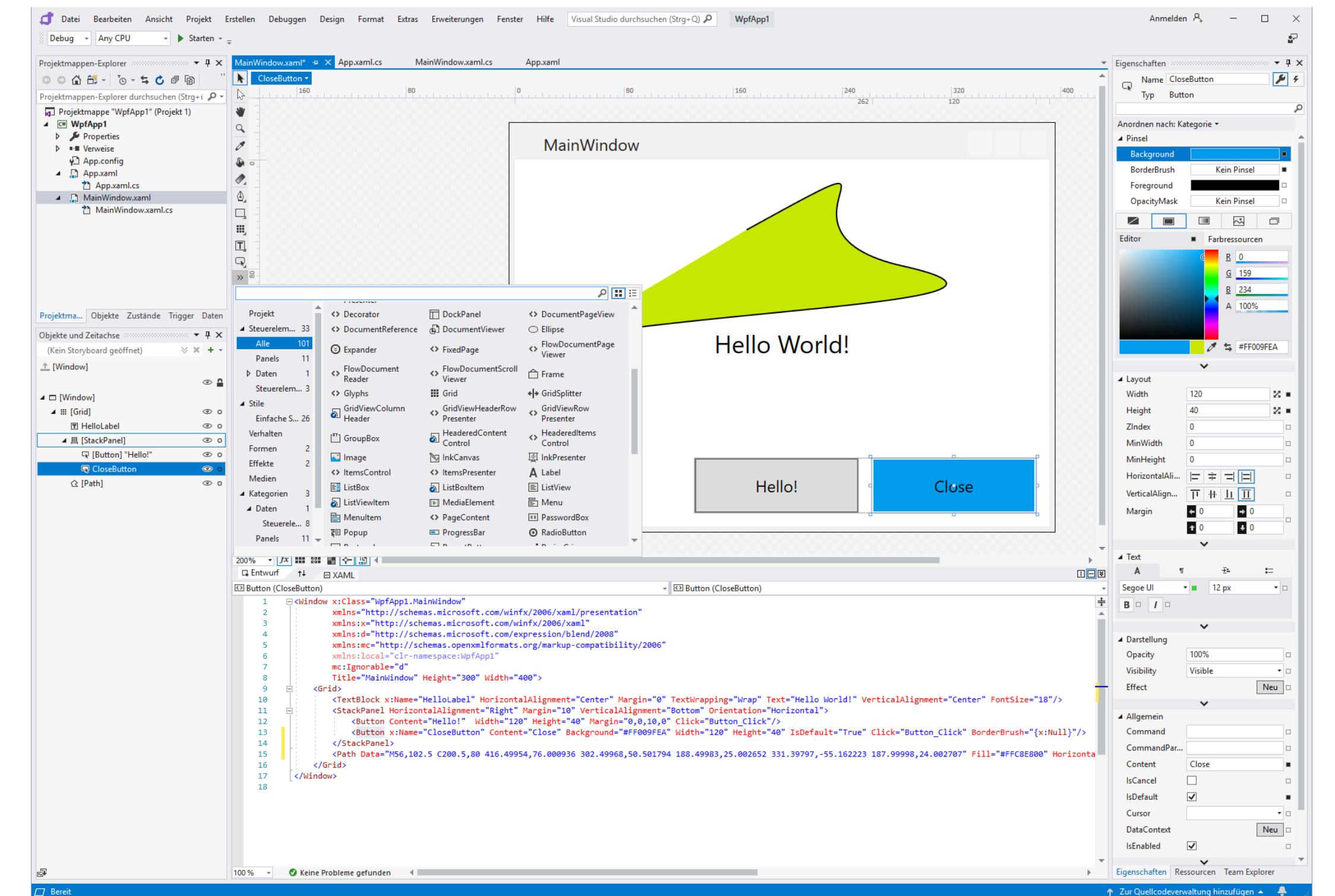


UITK: Components

Widget Set



User Interface Design System



Visual Studio Blend

UITK: Requirements

- Composition
- Reusability
- Communication
- Separation from app logic

UITK: Defining Widgets

Widget :=

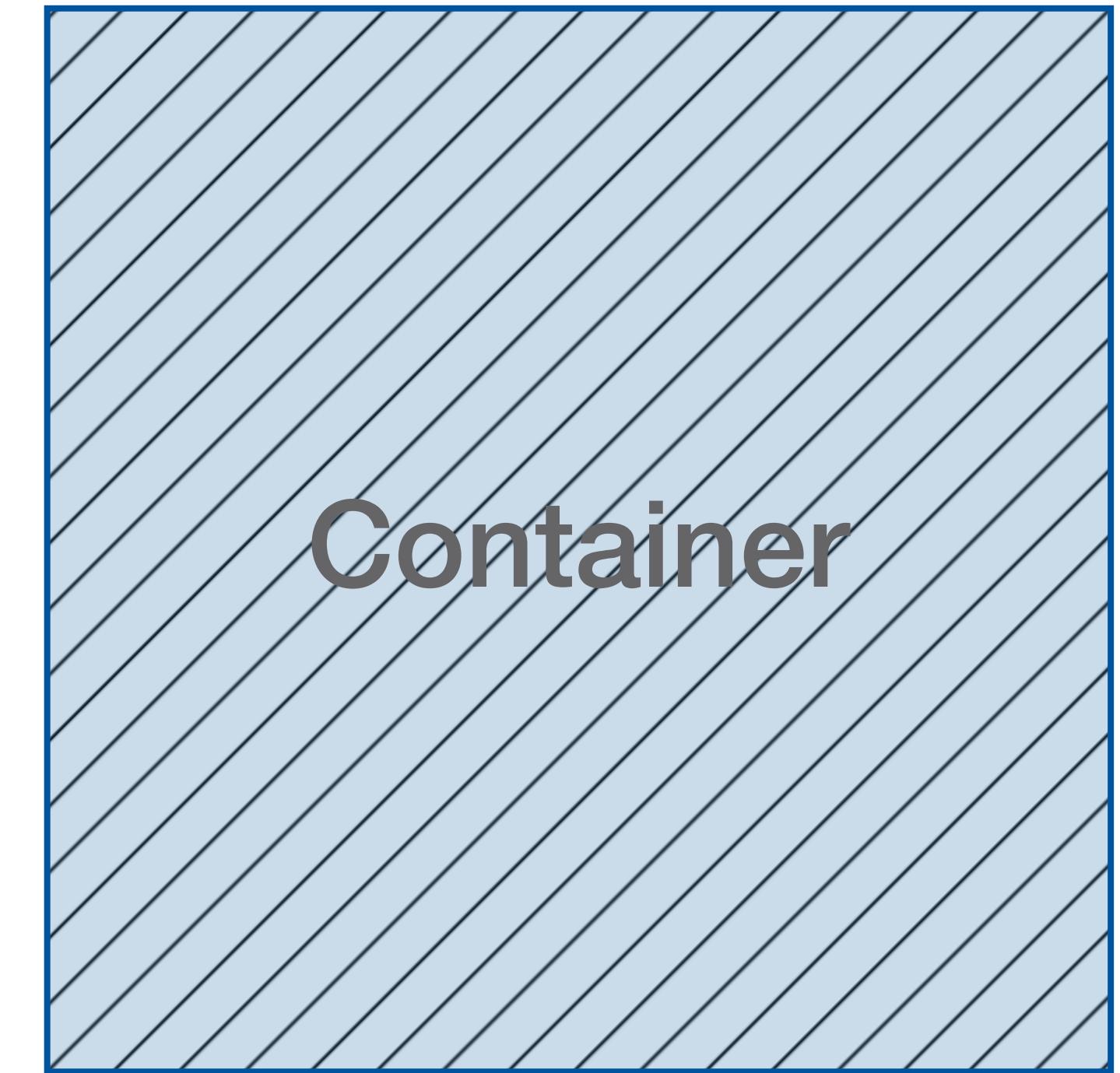
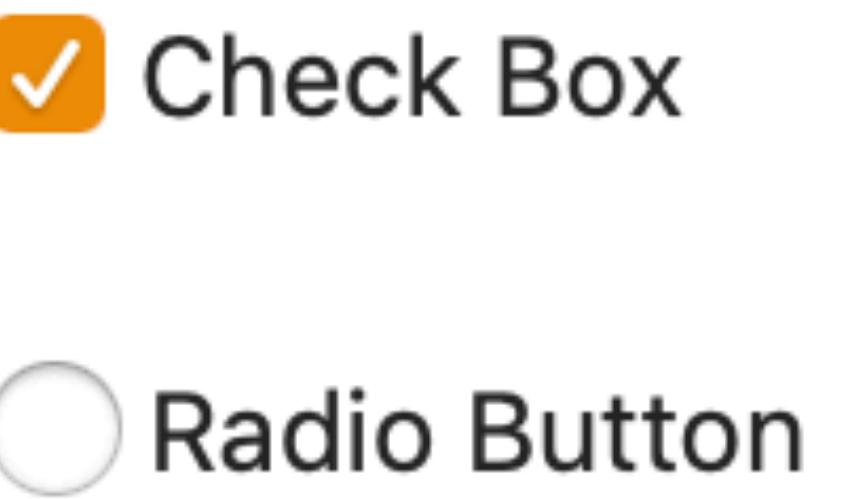
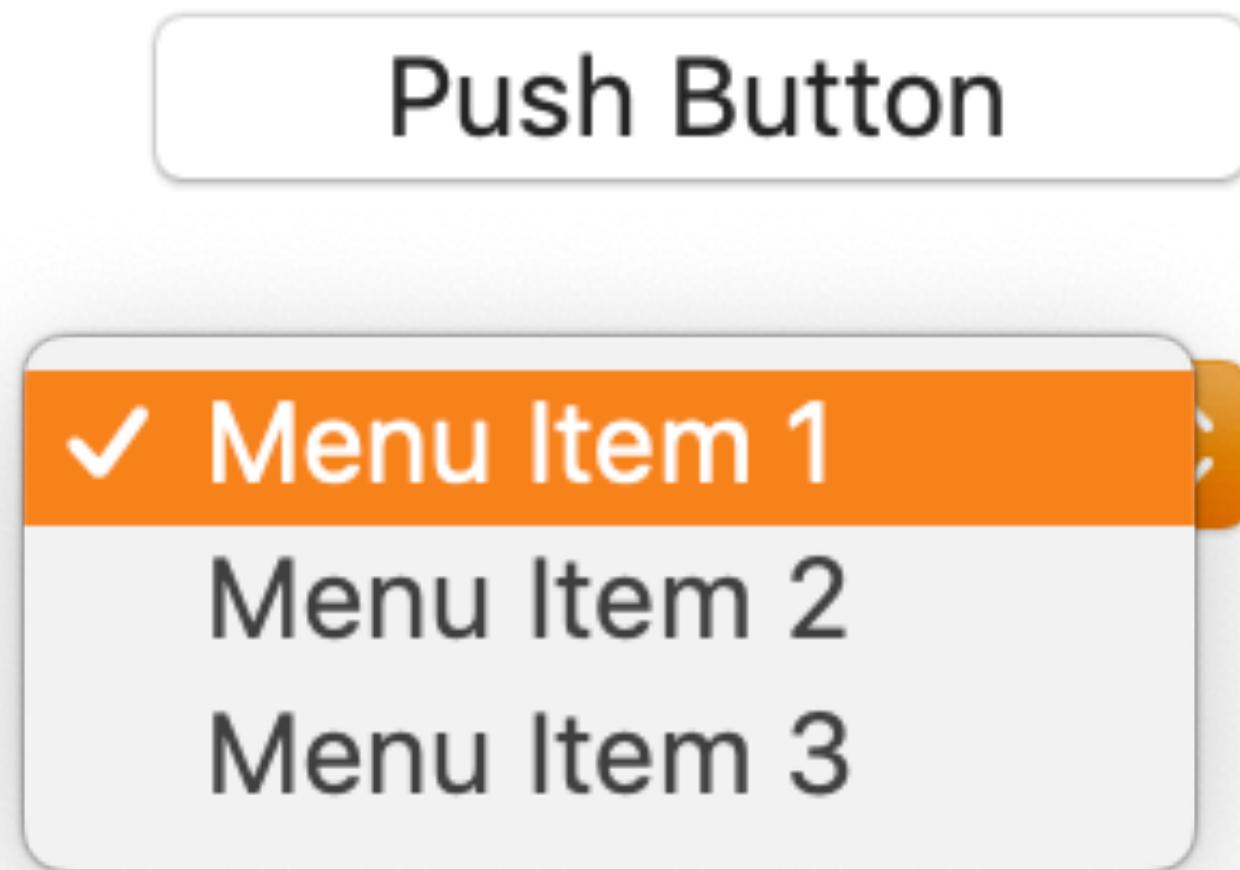
$\langle W=(w_1 \dots w_k), G=(g_1 \dots g_l), A=(a_1 \dots a_m), I=(i_1 \dots i_n) \rangle$

UITK: Defining Widgets

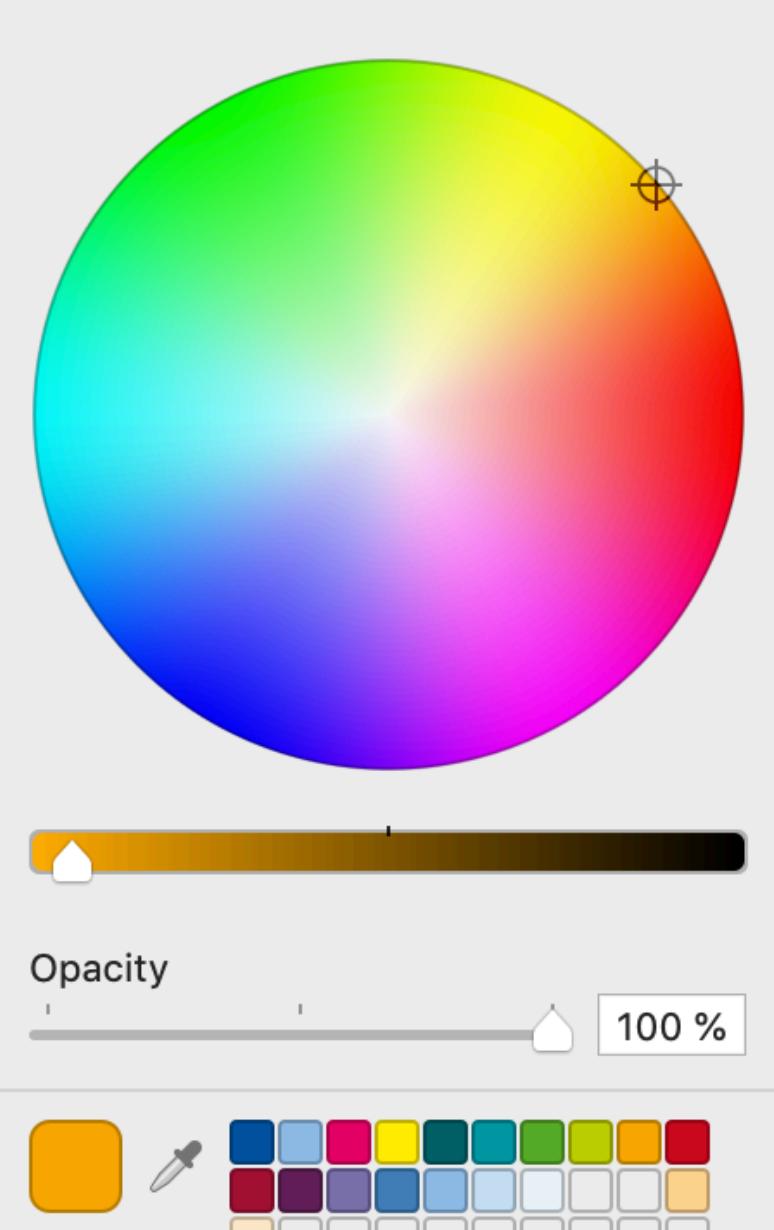
Exercise

What are the typical components $\langle W, G, A, I \rangle$ of a button?

UITK: Basic Widgets



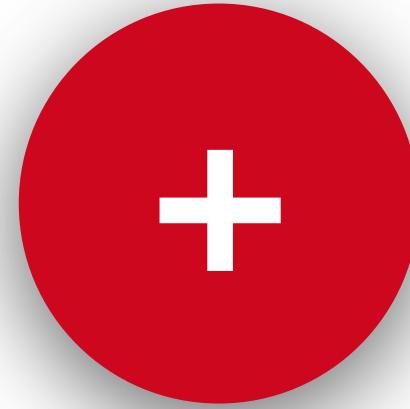
UITK: Creating Complex Widgets



Composition

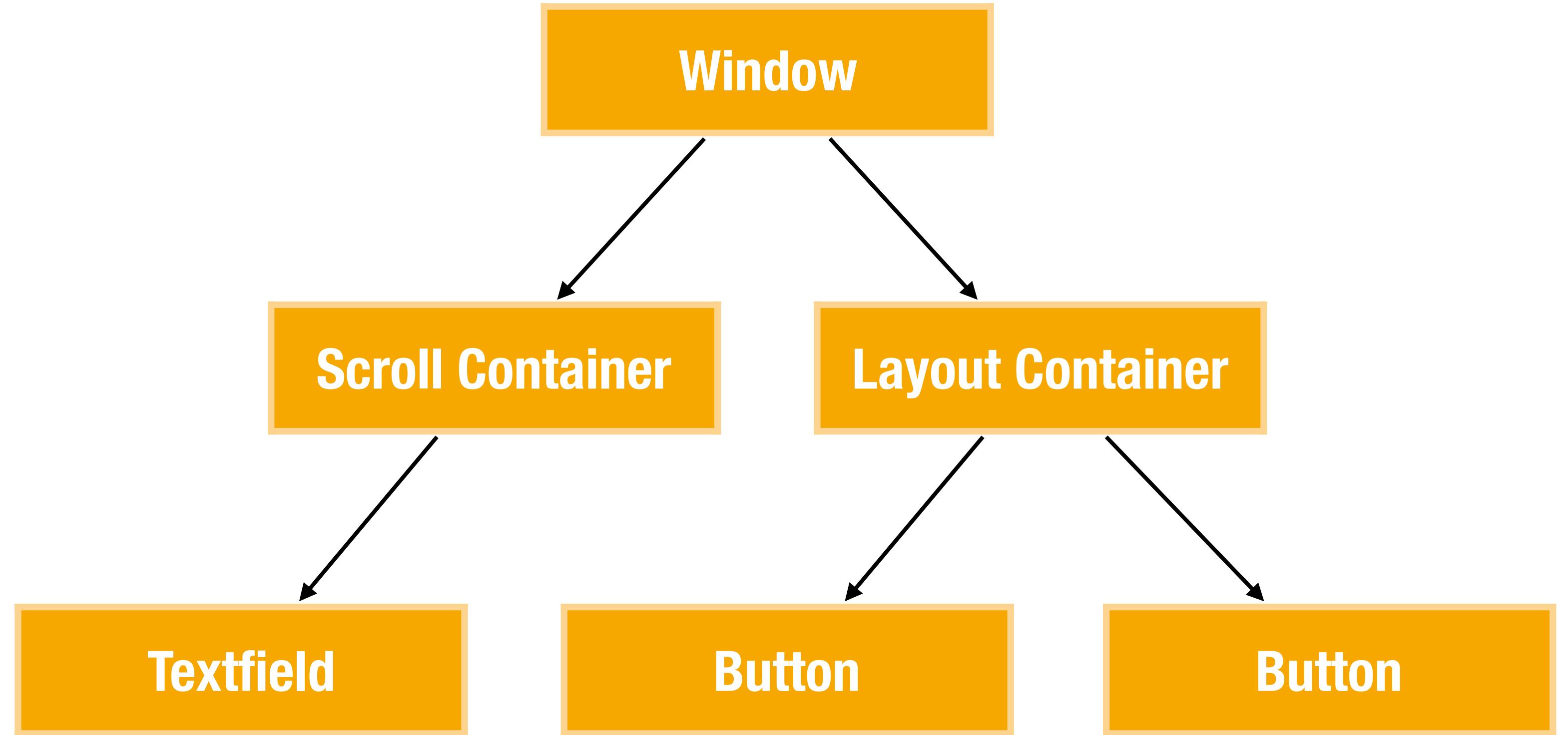
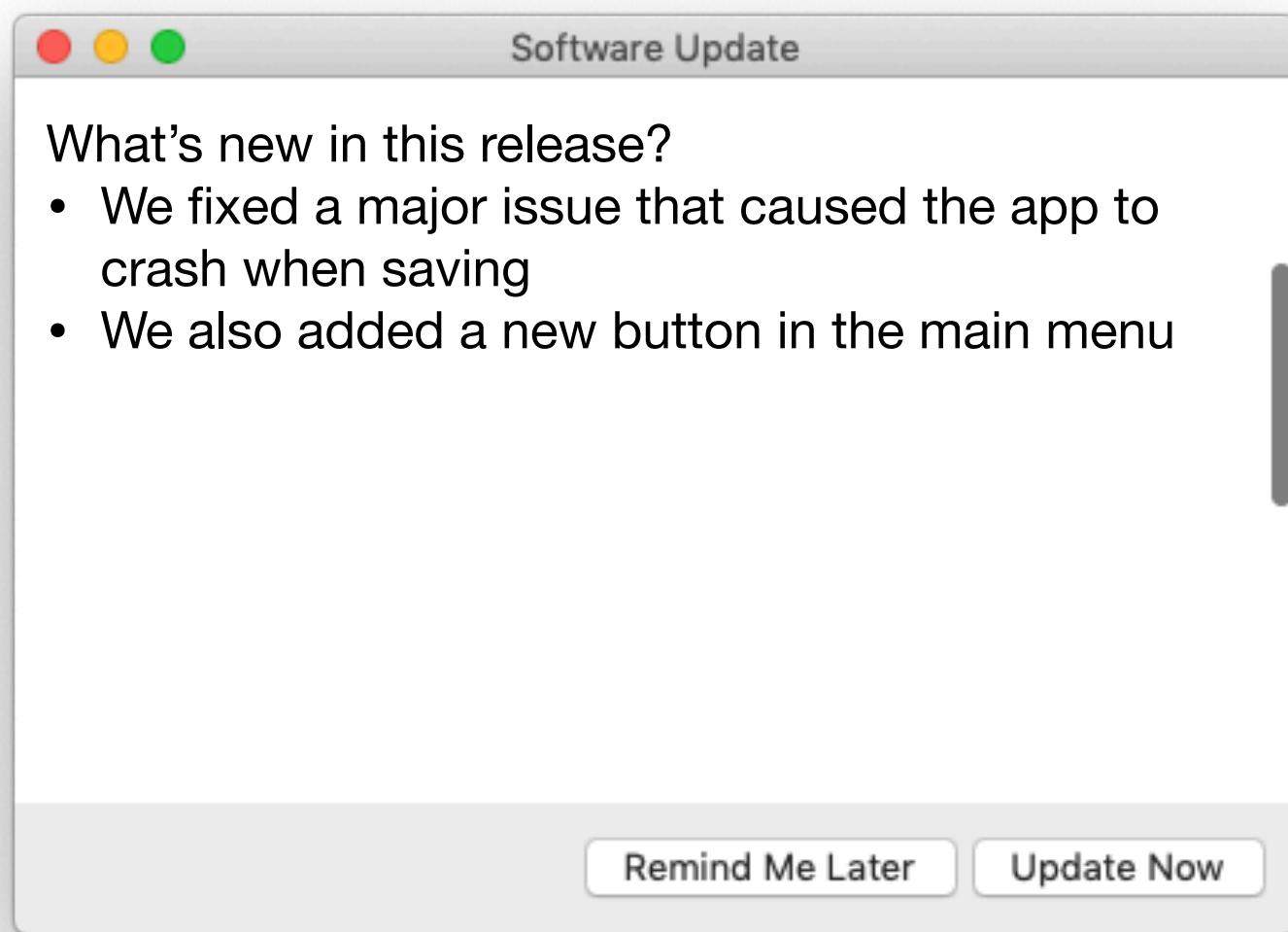


Minimal Button

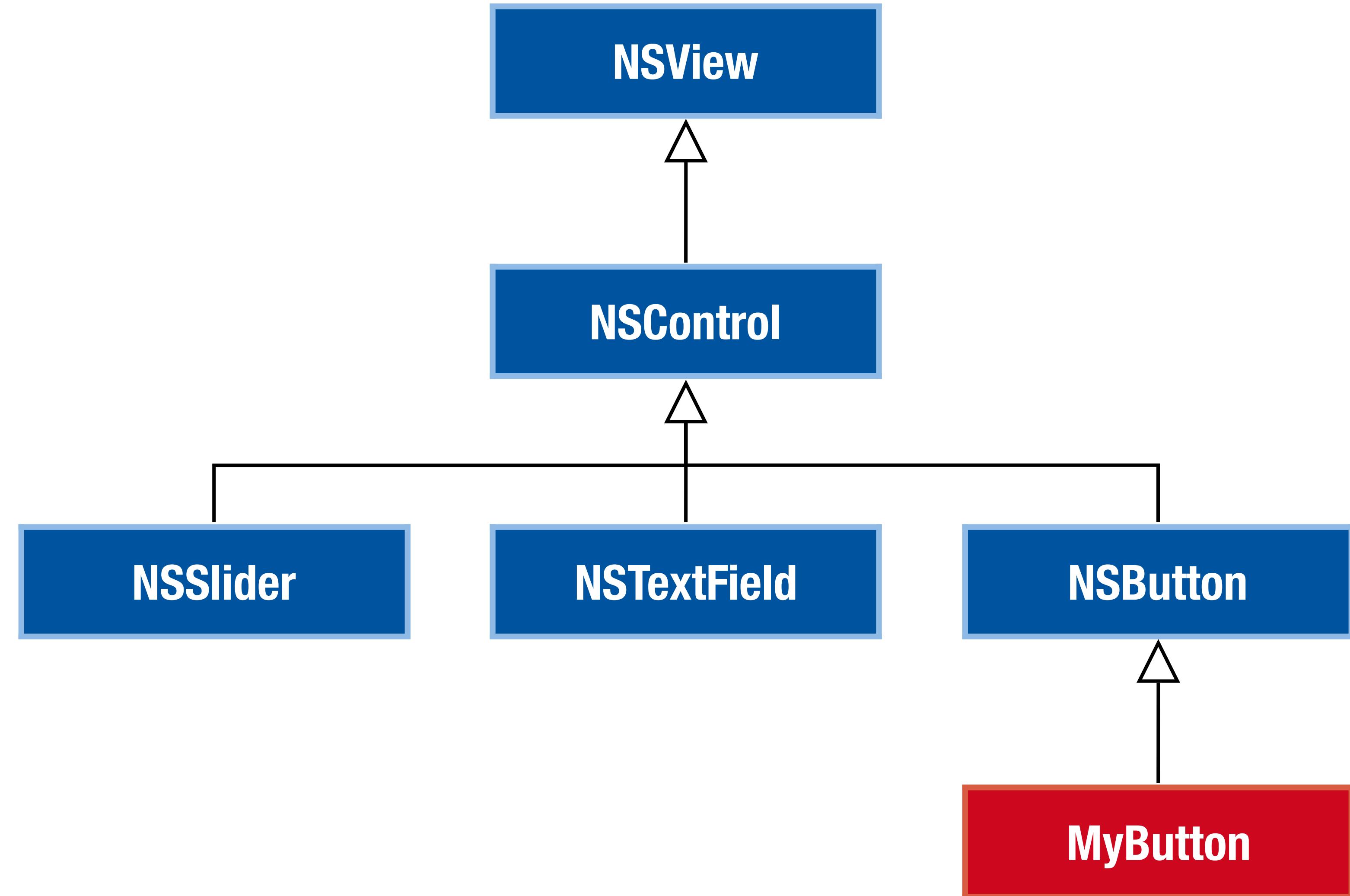


Refinement

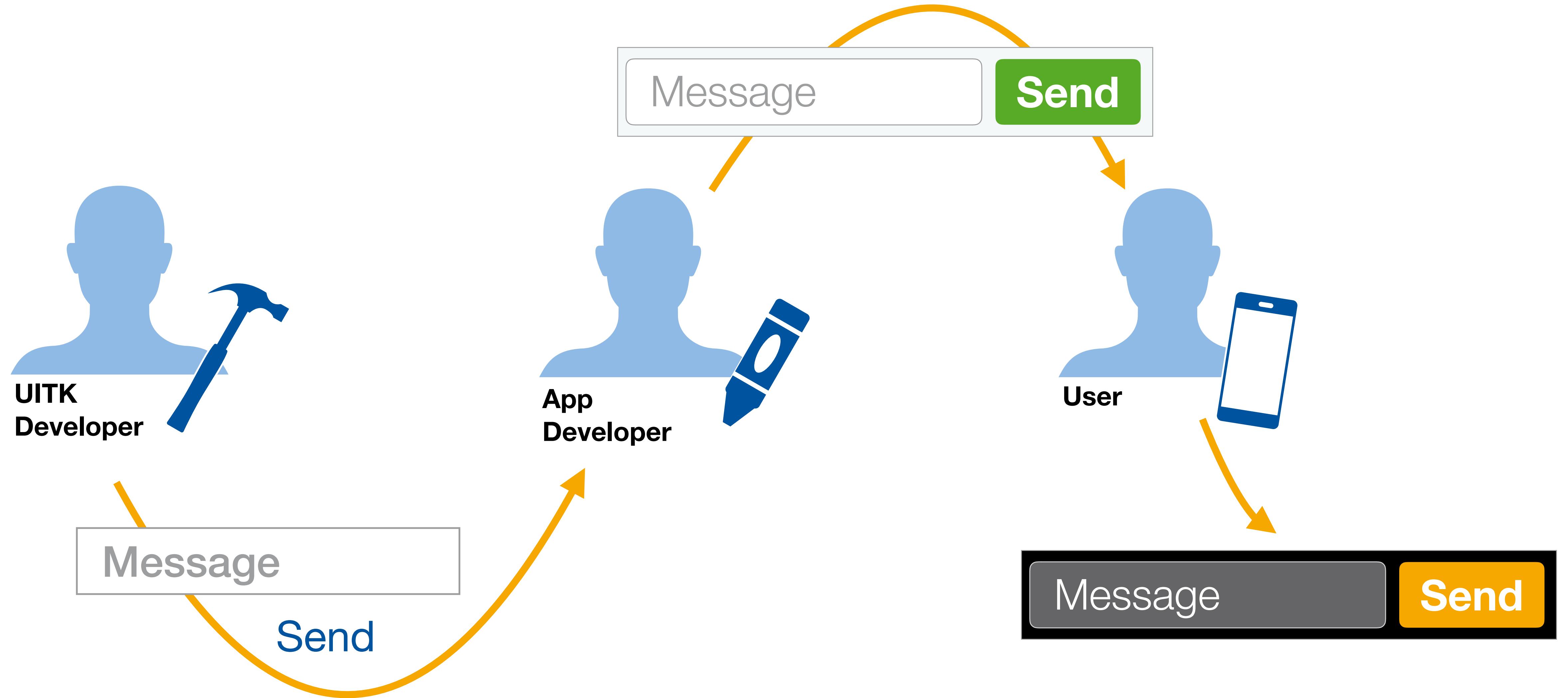
Dynamic Widget Hierarchy



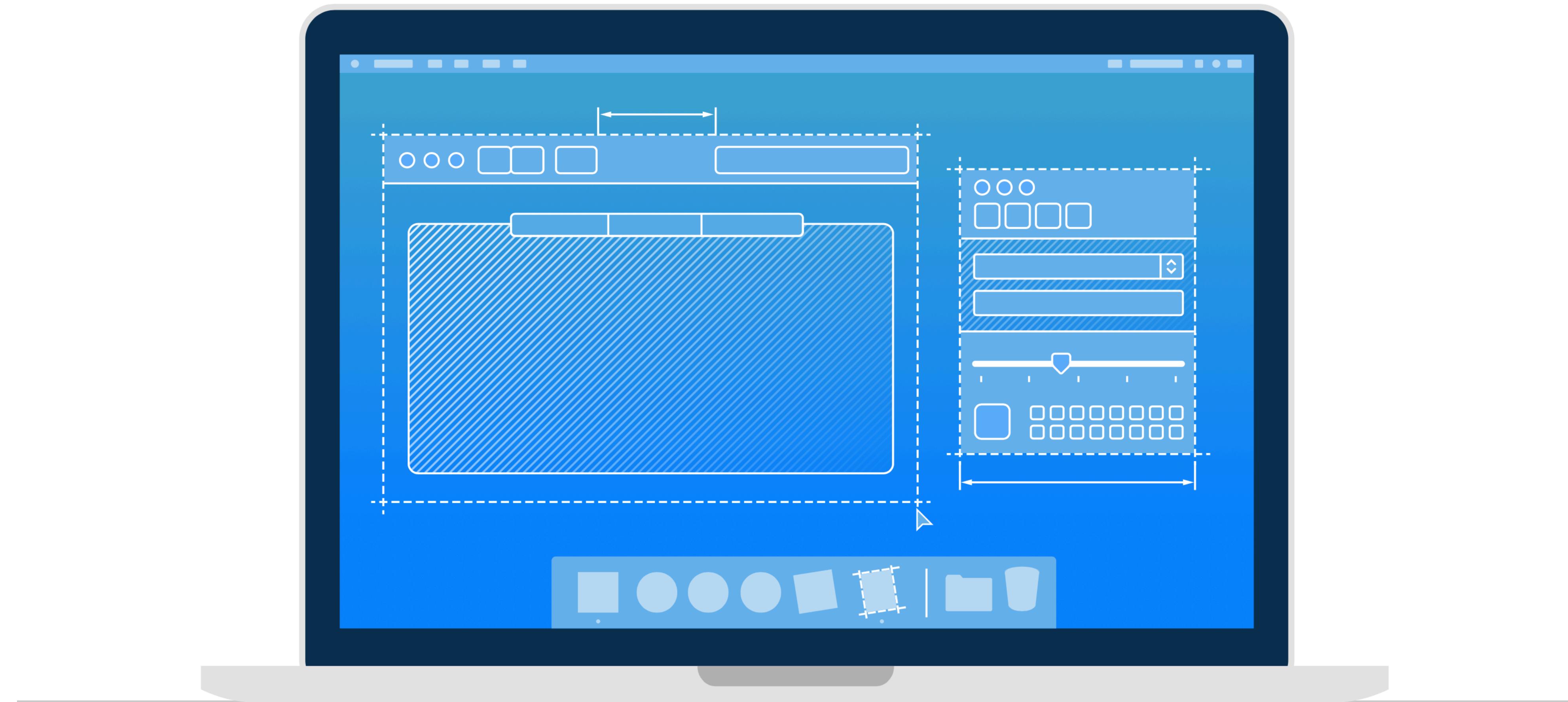
Static Widget Hierarchy



Late Refinement of Widgets



Style Guidelines



Types of UIDs

Language oriented (UIDL)

- Compiler implements style guidelines by checking constructs

Interactive

- Complex drawing programs to define look of UI
- Via lines connecting user input (**I**) to actions (**A**) (allowed by style guide)

Automatic

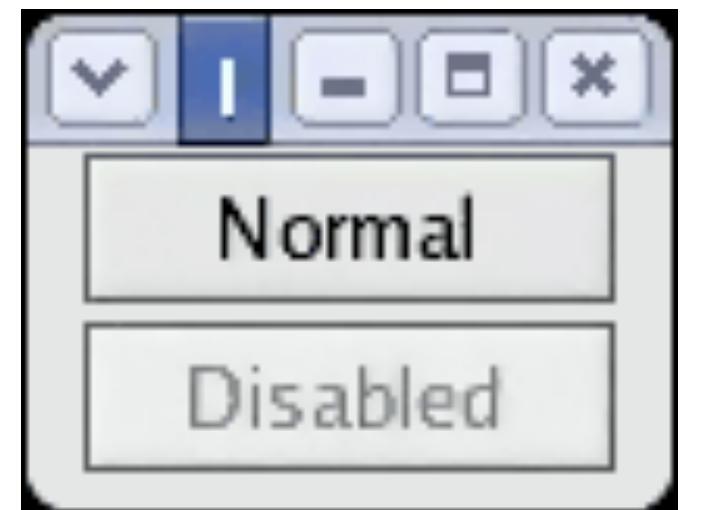
- Create UI automatically from spec of app logic (research)

User Interface Description Languages

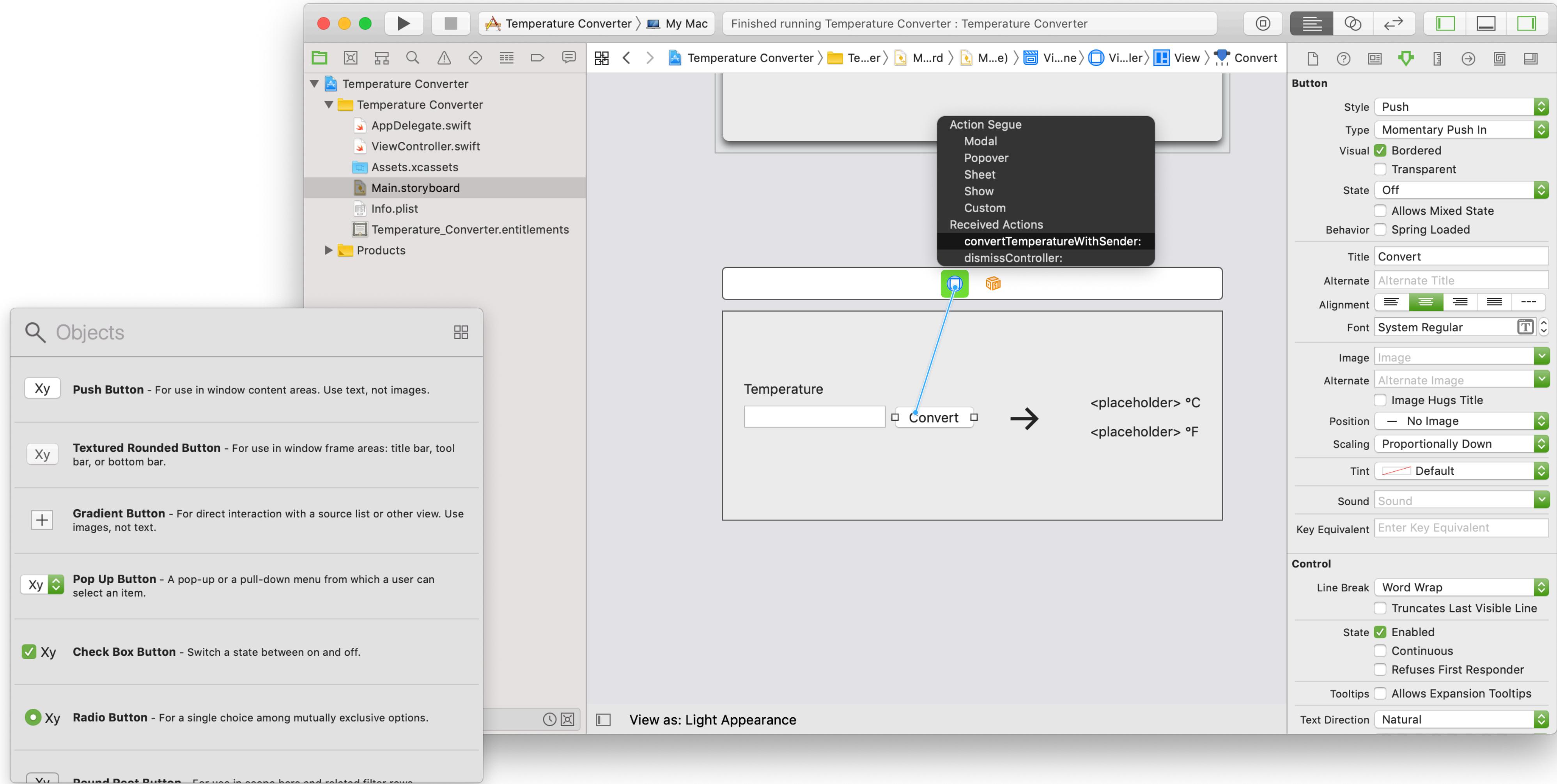
```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window id="findfile-window"
        title="Find Files"
        orient="horizontal"
        xmlns="http://www.mozilla.org/keymaster/gatekeeper/
there.is.only.xul">

    <button label="Normal"/>
    <button label="Disabled" disabled="true"/>

</window>
```



User Interface Design Systems



User Interface Toolkit

