Designing Interactive Systems 2

Lecture 7: Windows

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

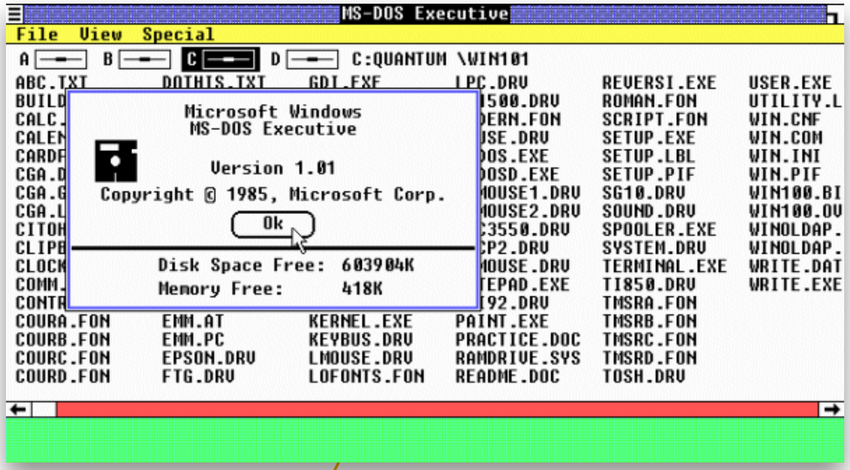
hci.rwth-aachen.de/dis2

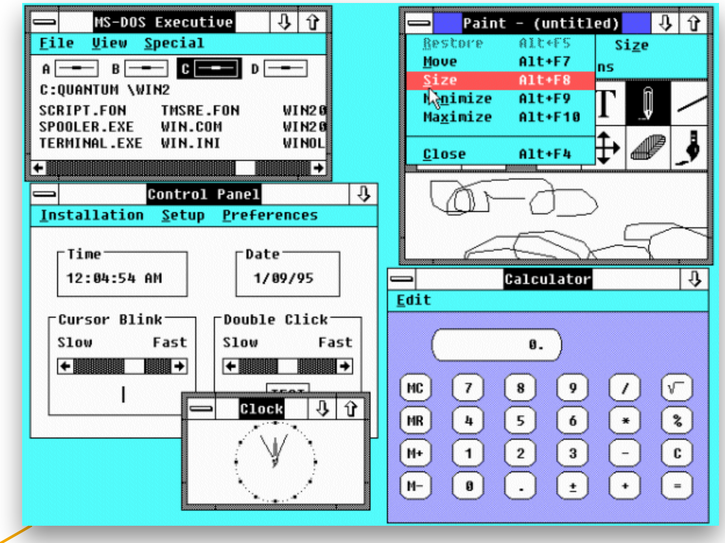
RWTH AACHEN UNIVERSITY

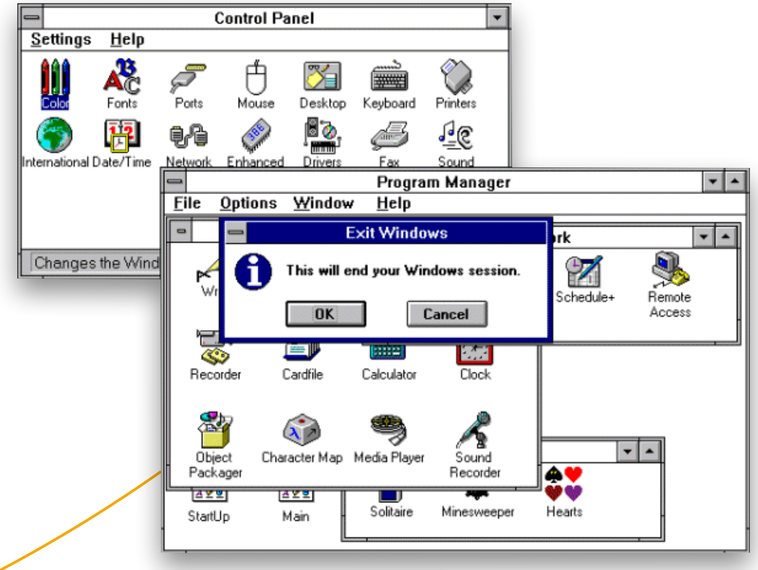
CHAPTER 19

Windows History

Windows 1.0 – 3.1

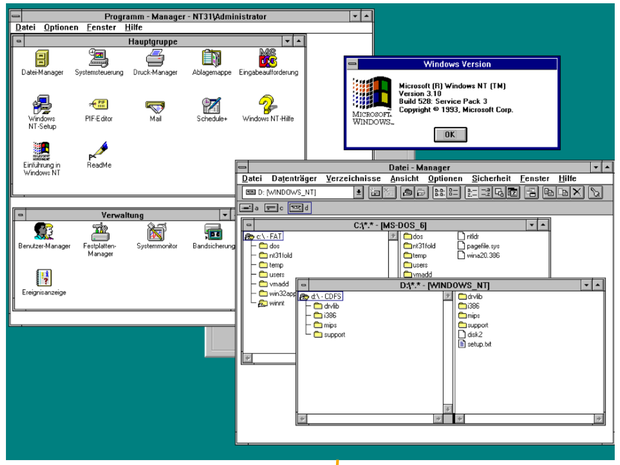






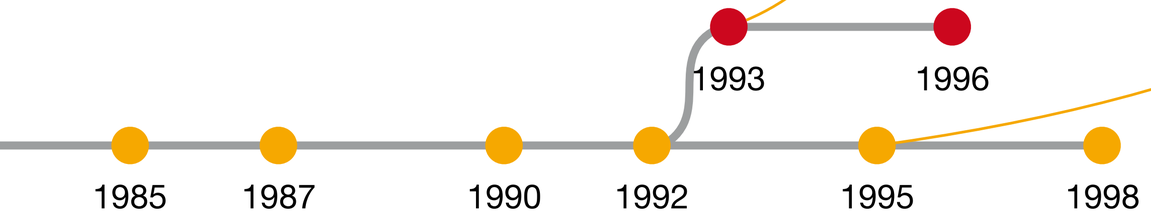
1985 1987 1990 1992

The Origins of Windows NT & Windows 95

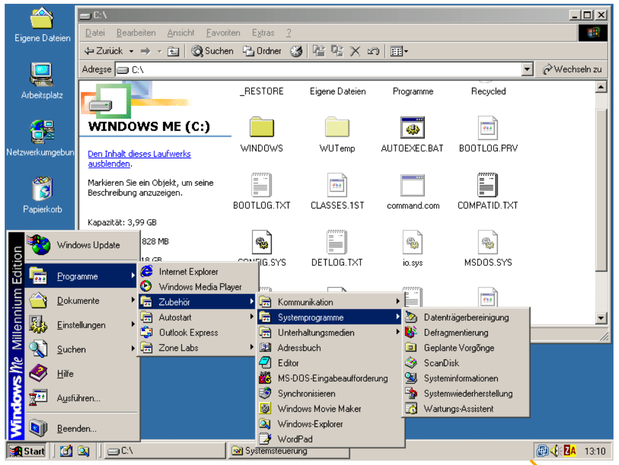


Windows NT 3.1

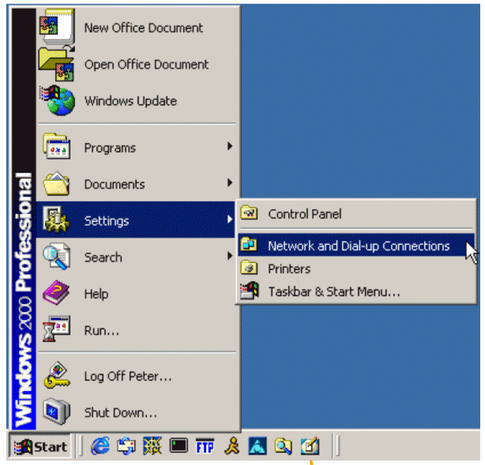
Windows 95



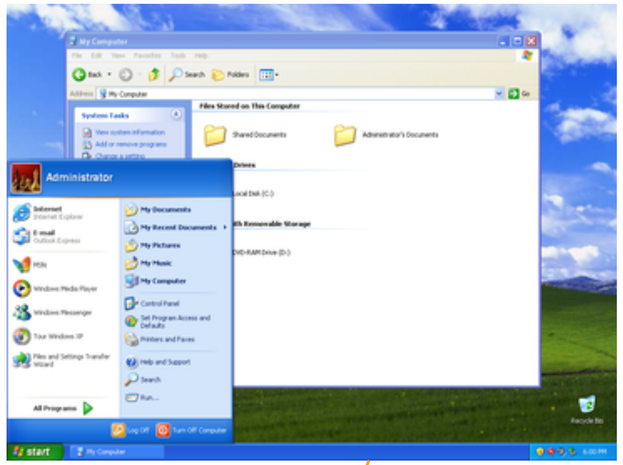
Windows 2000 – XP



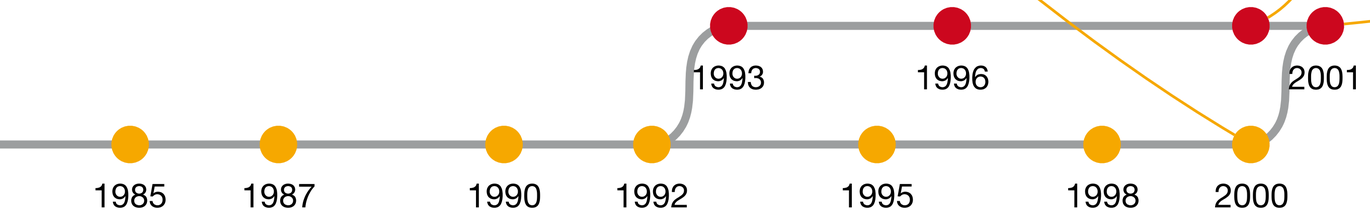
Windows ME



Windows 2000

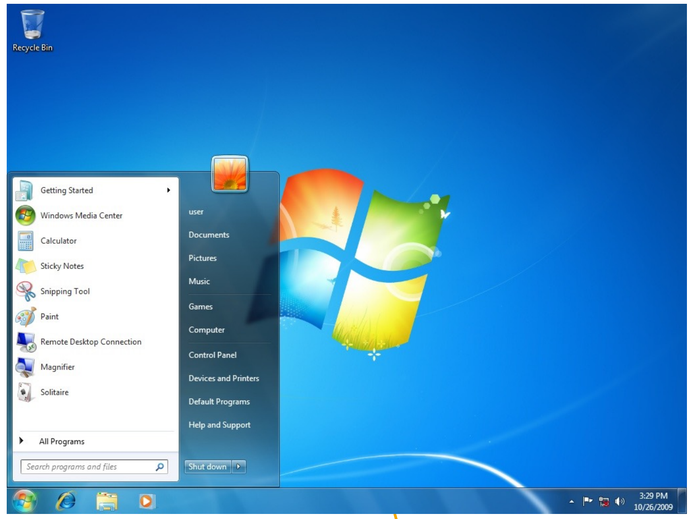


Windows XP

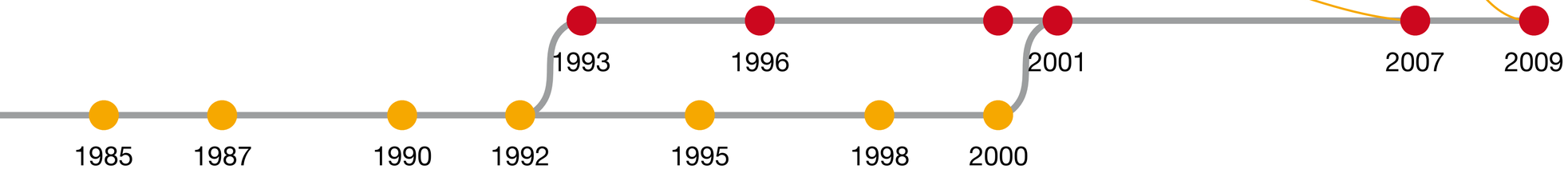


Windows Vista & 7

Windows Vista



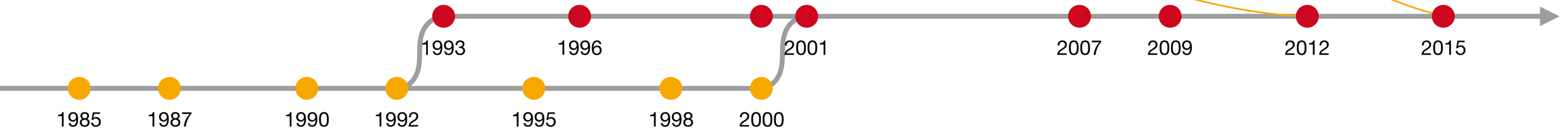
Windows 7

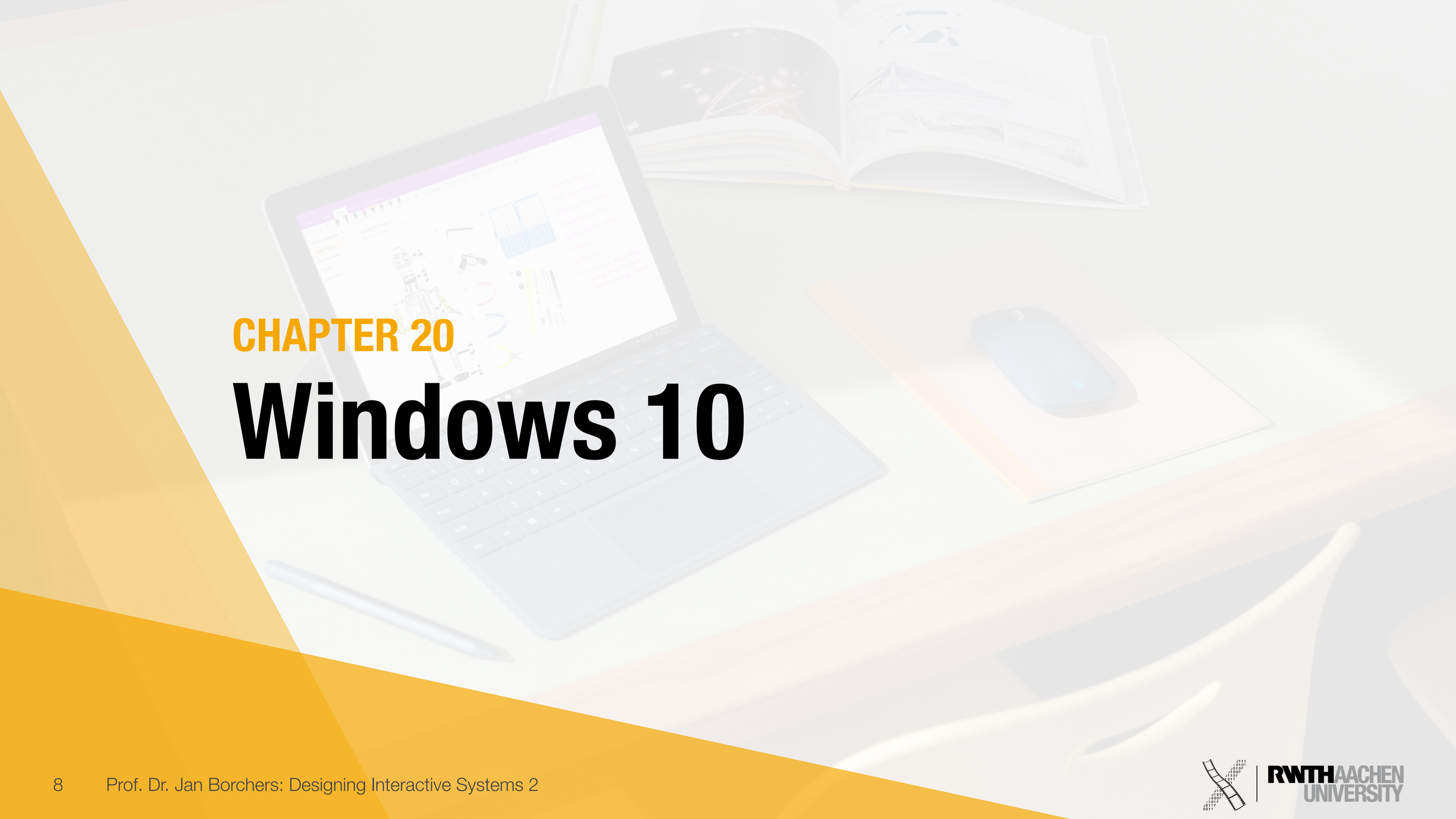


Windows 8 & Windows 10

Windows 8

Windows 10





CHAPTER 20

Windows 10

Windows 10

- Unified platform for different devices
  PC, tablet, Surface Hub, Xbox, HoloLens, IoT
- Intelligent assistant **Cortana**
- **Action Center** for notifications
  and quick settings
- Microsoft Edge web browser with annotations
- DirectX 12, Xbox streaming

Desktop Window Manager (DWM)

- **Task View** and virtual desktops
- **Timeline** shows recently used websites, files, apps
- Hardware accelerated GUI drawing
- **Desktop composition** feature allows visual effects as windows are drawn into an off-screen buffer

Modern UI

- Large variety of devices:
  Windows has to work with different input types

- **Desktop Mode**
  optimized for mouse input, windows, smaller controls

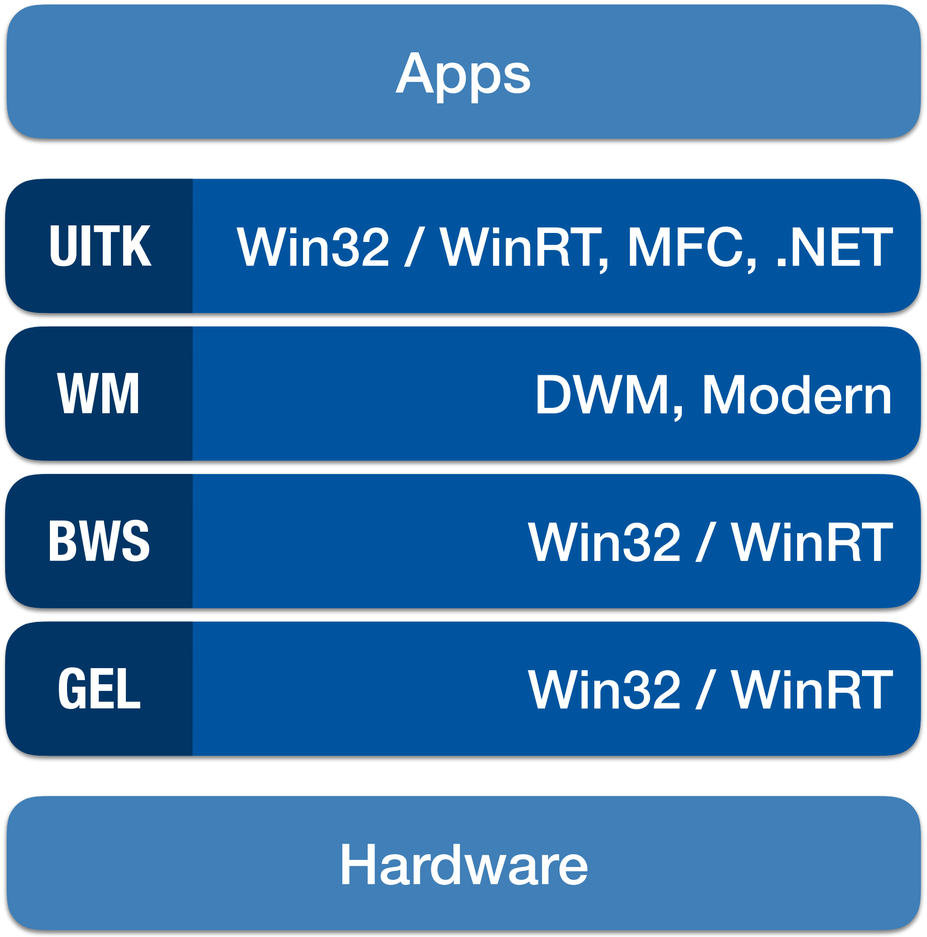
- **Tablet Mode**
  gestures, horizontal scrolling, fullscreen apps, back button in task bar

- Live Tiles take functionality of desktop and task bar icons, gadgets

Windows API

- **Base Services**
  File system, processes, threads
- **Advanced Services**
  Registry, system shutdown
- **Graphics Device Interface** (GDI)
  Graphical output to screens, printers
- **User Interface**
  UI widgets and mouse / keyboard input
- **Common Dialog Box Library**
  e.g., for color and fonts
- **Common Control Library** Status/ progress bars, toolbars, tabs
- **Windows Shell**

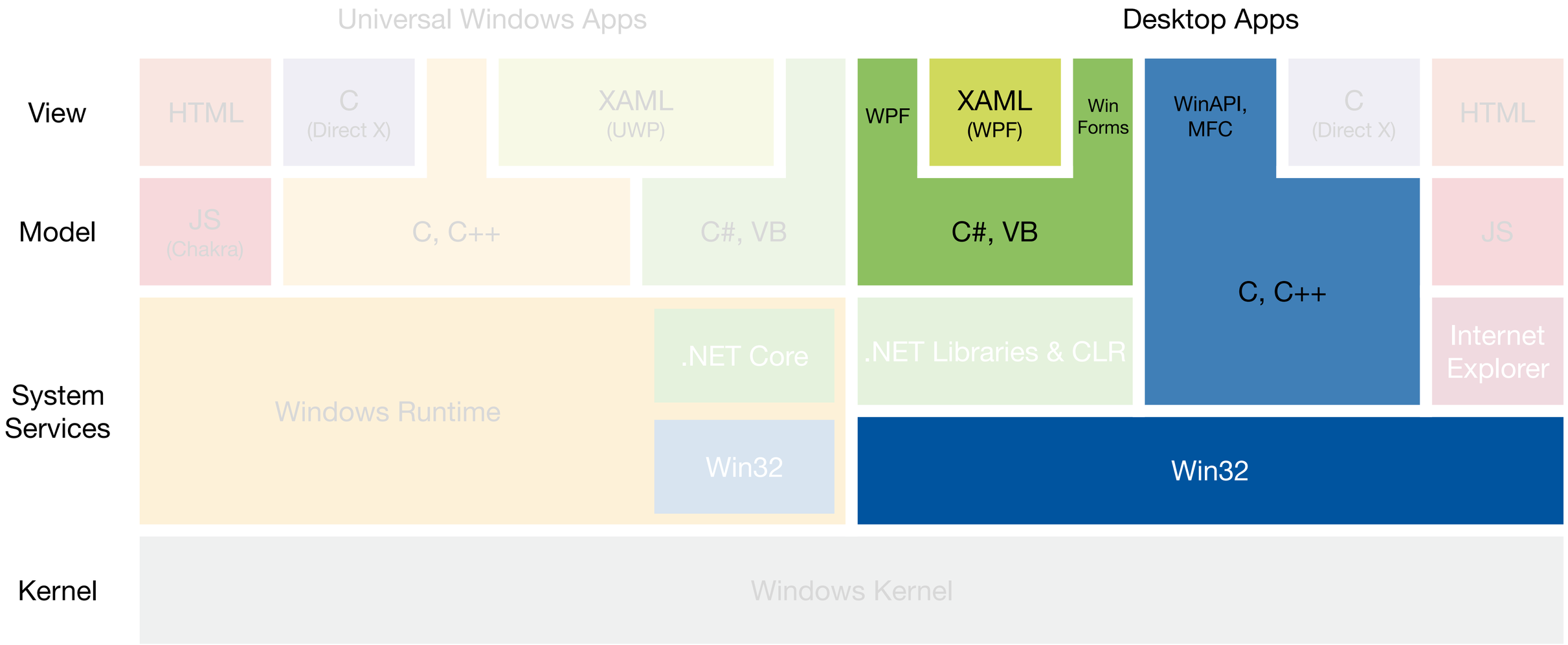
Windows: Four Layer Model



CHAPTER 21

Windows Desktop Apps

Windows Application Architecture



Hello, Windows API

```
#define PROG_NAME "Win32 Hello World"

HWND hWnd = NULL;
HANDLE hThread = NULL;

unsigned int __stdcall thread_main(void*) {
    MessageBox(NULL, "hello, world", PROG_NAME, MB_OK | MB_TOPMOST);
    hThread = NULL;
    PostMessage(hWnd, WM_CLOSE, 0, 0);
    return 0;
}

HANDLE start_thread() {
    unsigned int id;
    hThread = (HANDLE)_beginthreadex(NULL, 0, thread_main, NULL, 0, &id);
    if (hThread == NULL) {
        // error
    }
    return hThread;
}

static LRESULT CALLBACK win_proc(HWND hwnd, UINT msg, WPARAM wp, LPARAM lp)
{
    switch (msg) {
        case WM_CREATE:
            hWnd = hwnd;
            if (start_thread() == NULL) {
                PostMessage(hwnd, WM_CLOSE, 0, 0);
            }
            return 0;
        case WM_CLOSE:
            if (hThread != NULL) {
                WaitForSingleObject(hThread, INFINITE);
                CloseHandle(hThread);
            }
            DestroyWindow(hwnd);
            return 0;
        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;
        default:
            return DefWindowProc(hwnd, msg, wp, lp);
    }
}

int WINAPI WinMain(HINSTANCE hi, HINSTANCE hp, LPSTR cmdline, int cmdshow)
{
    if (!hp) {
        WNDCLASS wc;
        wc.style = 0;
        wc.lpfnWndProc = win_proc;
        wc.cbClsExtra = 0;
        wc.cbWndExtra = 0;
        wc.hInstance = hi;
        wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
        wc.lpszMenuName = NULL;
        wc.lpszClassName = PROG_NAME;
        if (! RegisterClass(&wc)) {
            // error
            return 0; }
    }
    HWND wnd = CreateWindow(PROG_NAME, PROG_NAME,
                            WS_POPUP, 0, 0, 0, 0, NULL, NULL, hi, NULL);
    if (wnd == NULL) {
        // error
        return 0;
    }
    ShowWindow(wnd, SW_SHOW);
    UpdateWindow(wnd);
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}
```

Microsoft Foundation Classes (MFC)

- Introduced in 1992 for 16-bit Windows application development
- Library that wraps Windows API in C++ classes
- Objects for predefined windows and controls (button, text box, etc.)
- Still updated for backwards compatibility
  - e.g., added new UI controls, such as ribbons and docking panes
  - remains being used

Hello, MFC

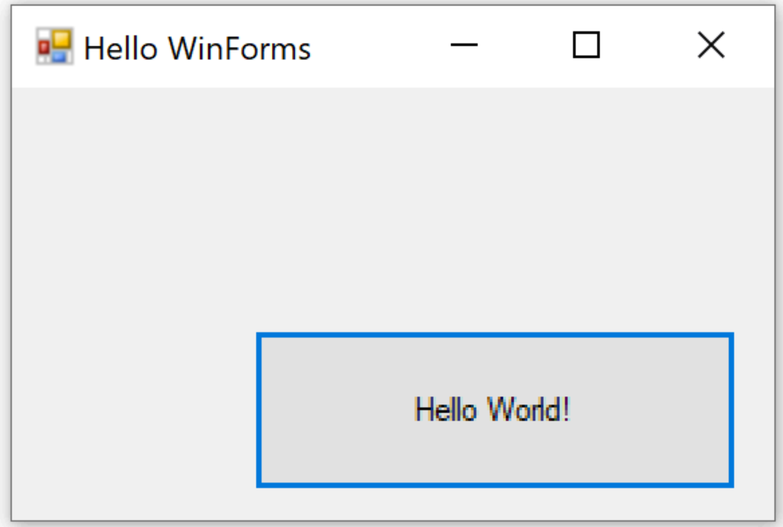
```
class HelloApplication : public CWinApp {
public:
    virtual BOOL InitInstance();
};
HelloApplication HelloApp;

#define BUTTON_ID 1001

class HelloWindow : public CFrameWnd {
public:
    HelloWindow();
protected:
    afx_msg void OnClicked();
    DECLARE_MESSAGE_MAP();
    CButton *m_pHelloButton;
};

BOOL HelloApplication::InitInstance() {
    m_pMainWnd = new HelloWindow();
    m_pMainWnd->ShowWindow(m_nCmdShow);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}
BEGIN_MESSAGE_MAP(HelloWindow, CFrameWnd)
ON_BN_CLICKED(BUTTON_ID, OnClicked)
END_MESSAGE_MAP()

HelloWindow::HelloWindow() {
    Create(NULL, _T("Hello MFC"),
WS_OVERLAPPEDWINDOW, CRect(0,0,300,160));
    m_pHelloButton = new CButton();
    m_pHelloButton->Create(_T("Hello World!"),
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
CRect(50,20,230,80), this, BUTTON_ID);
}

void HelloWindow::OnClicked() {
    PostMessage(WM_CLOSE);
}
```

Windows Forms

- Introduced in 2002 as part of the .NET framework
- Wrapper over Windows user interface libraries and GDI+
- Offers native controls and some layout management
- Position of widgets defined at design time:
  Issues dealing with different system fonts and varying aspect ratios
- Since 2018 open source

Hello, Windows Forms

```
public class MyForm : Form {
    private Button button = new Button();

    MyForm() {
        this.Size = new Size(300,200);
        this.Text = "Hello WinForms";

        button.Text = "Hello World!”;
        button.SetBounds(90,90,180,60);
        button.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;
        EventHandler handler = new EventHandler(buttonClicked);
        button.Click += handler;

        this.Controls.Add(button);
    }

    private void buttonClicked(object sender, EventArgs e) {
        Application.Exit();
    }

    public static void Main(string[] args) {
        Application.Run(new MyForm());
    }
}
```

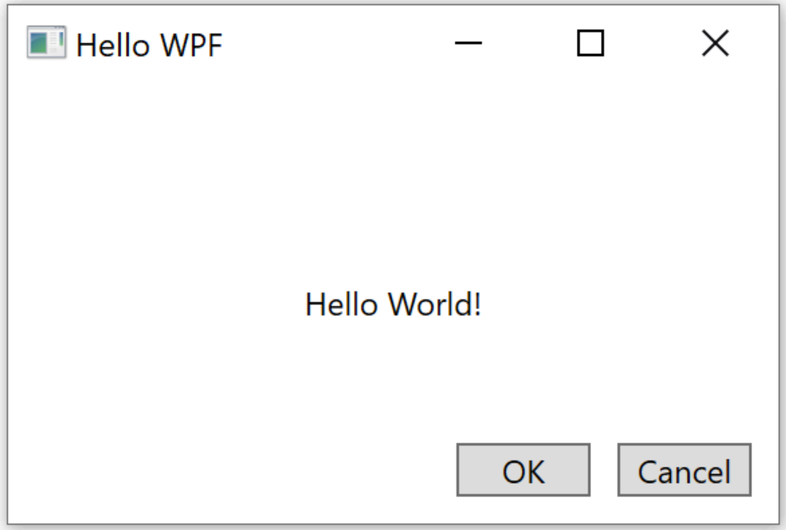


Windows Presentation Foundation (WPF)

- Released in 2006 as part of .NET 3
- Rendering based on DirectX
  allows faster screen refreshes, animations, effects
- Scaling based on layout managers
- Design-centric framework including UIDS
  - Use XAML to separate UI from business logic
  - Data binding
- Multitouch support as of Windows 7

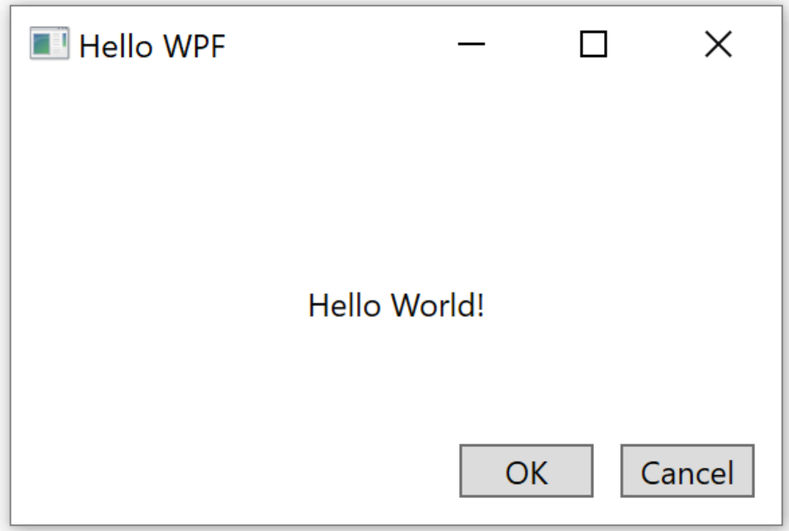
Hello, WPF

```
<Window Title="Hello WPF" Height="200" Width="300">
    <Grid>
        <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
                   Text="Hello World!" x:Name="label"/>
        <StackPanel HorizontalAlignment="Right" VerticalAlignment="Bottom" Width="Auto"
                    Margin="10,0,10,10" Orientation="Horizontal">
            <Button Content="OK" Padding="16,0" Click="Callback"/>
            <Button Content="Cancel" Margin="10,0,0,0" Padding="8,0"/>
        </StackPanel>
    </Grid>
</Window>
```



Hello, WPF

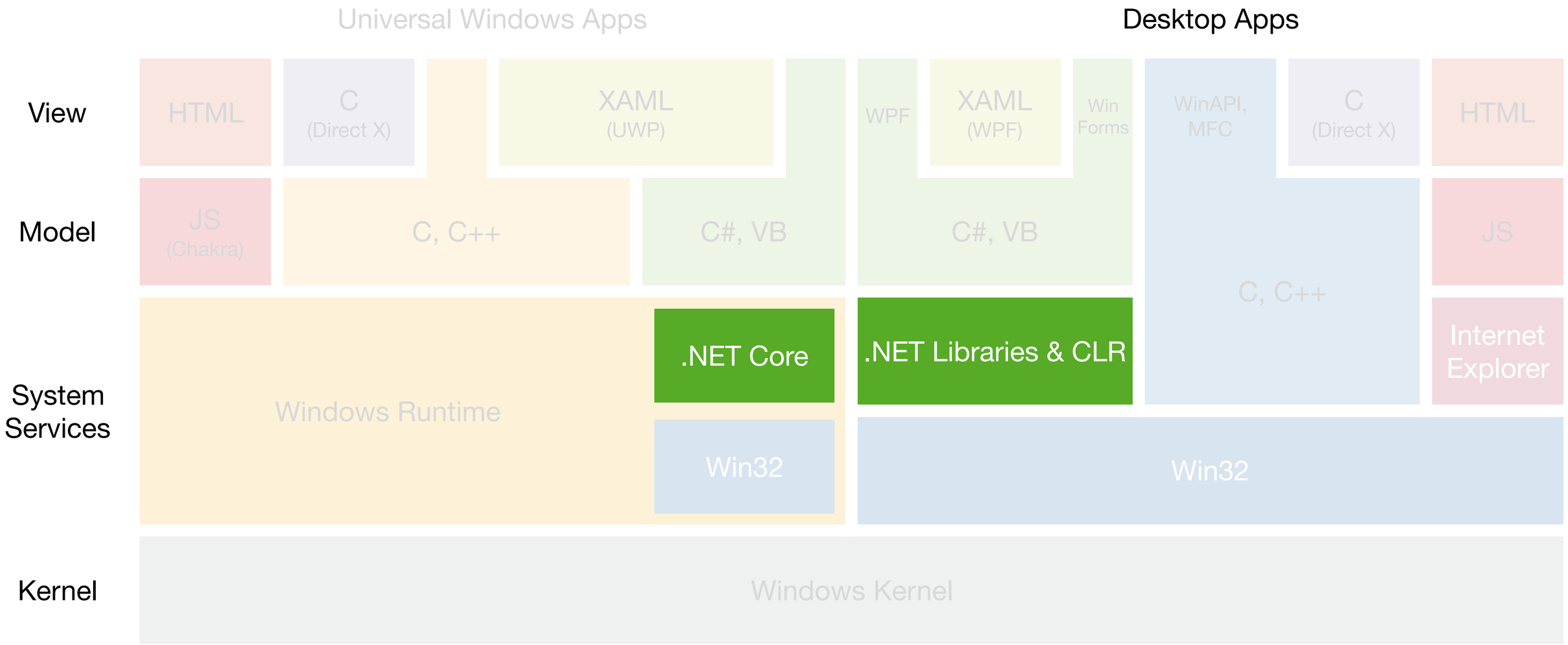
```
public partial class MainWindow : Window
{
   public MainWindow()
   {
      InitializeComponent();
   }

   private void Callback(object sender, RoutedEventArgs e)
   {
      this.label.Text = "Clicked";
   }
}
```



CHAPTER 22

.NET

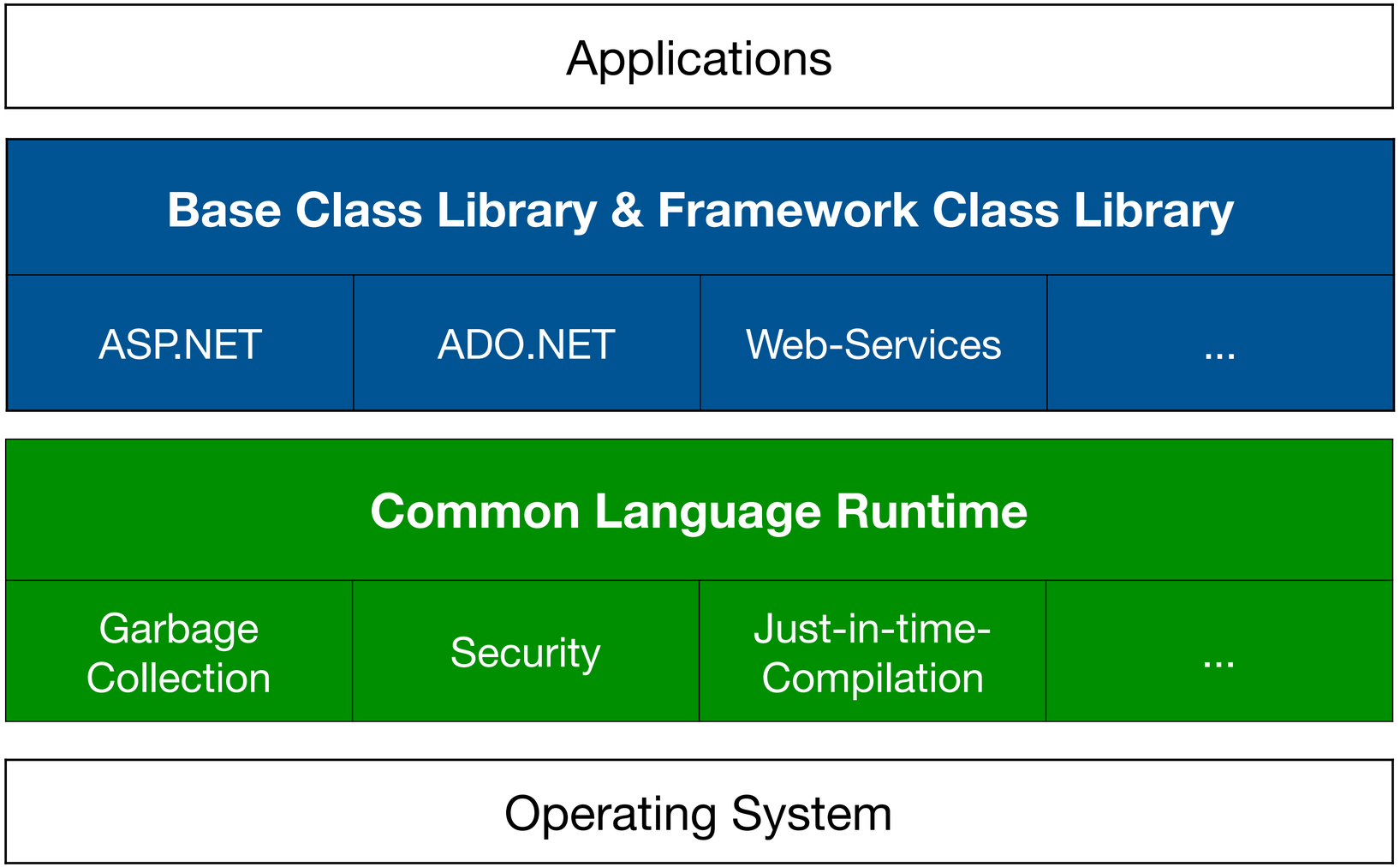
Windows Application Architecture



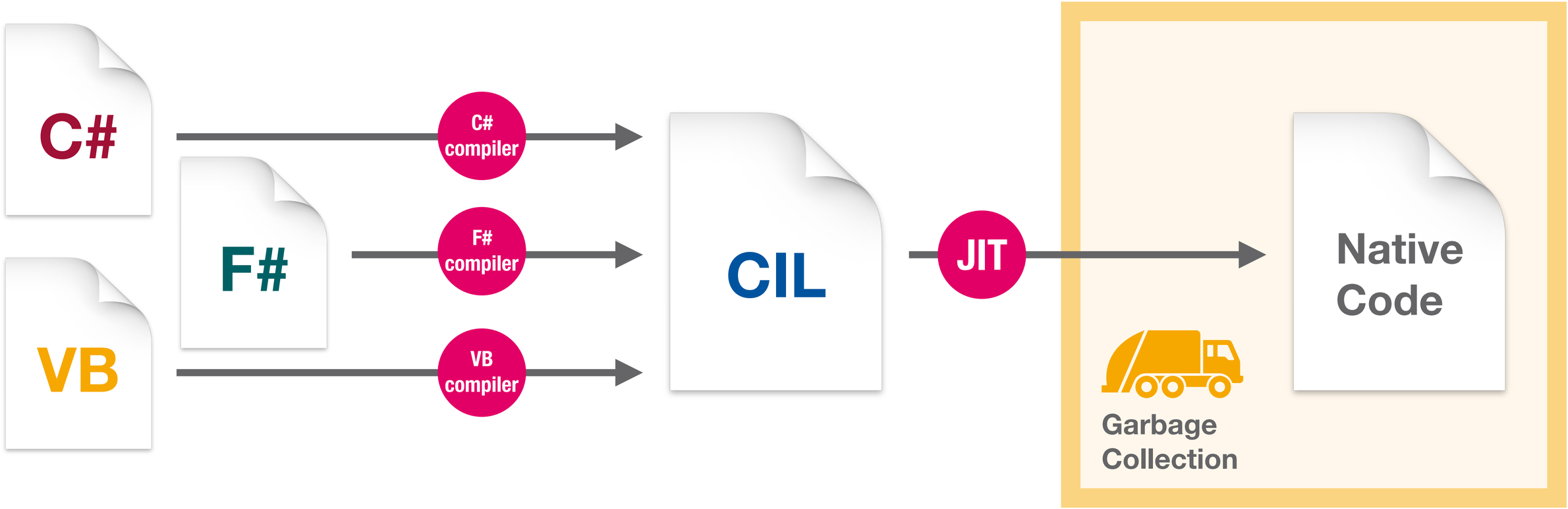
Microsoft .NET Framework

- Software platform developed by Microsoft
- Layer on top of Windows (and other systems)
- Framework Class Library – UI, data access, web application development, …
- Common Language Runtime
- Language interoperability
- Vision: Join all existing software systems and platforms
- June 2016: .NET Core 1.0 for Windows, Mac, Linux

.NET Architecture



Common Language Runtime (CLR)



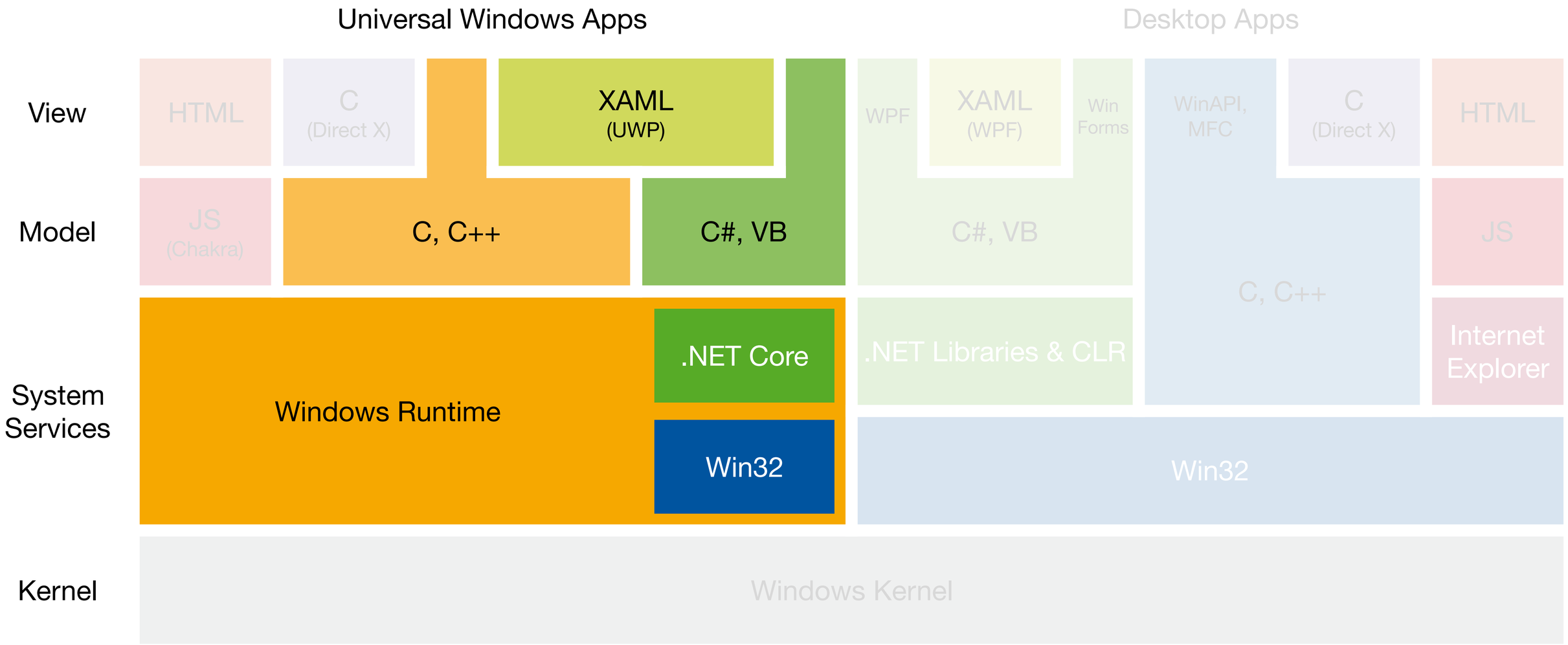
.NET Core

- Goal:
  Modernizing .NET to achieve cross-platform support and increased modularity
- Used to create device, cloud and IoT applications
- Open source
- **.NET Standard** is the base set of APIs that are shared with .NET
- Static compilation of Windows Store apps with **.NET Native** bundles required portions of .NET with an app

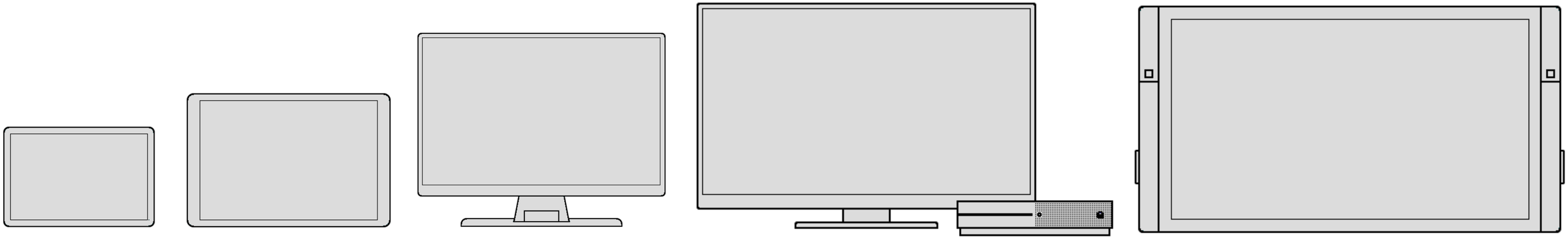
CHAPTER 23

Universal Windows Apps

Windows Application Architecture

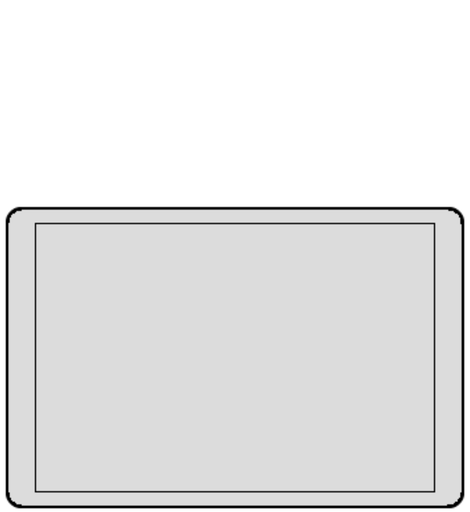


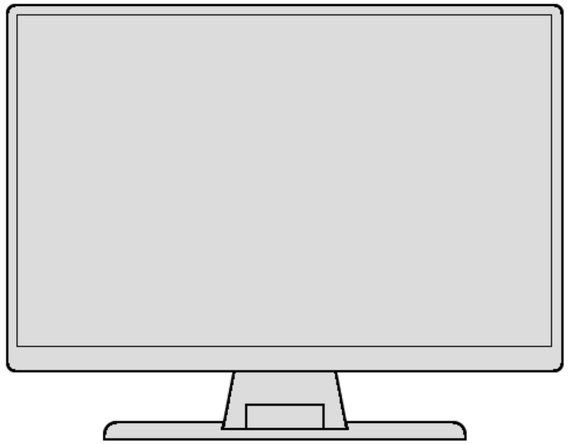
Universal Windows Platform

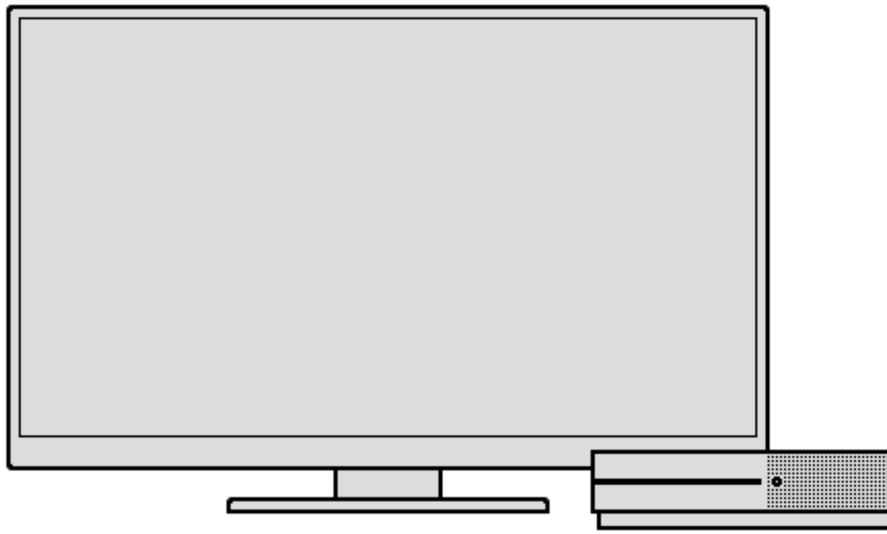


IoT devices | Tablets | PCs and laptops | Xbox and TV | Surface Hub

Universal Windows Platform







Tablets and 2-in-1s

7” to 13.3” and greater

80% used by the owner

touch, stylus, (keyboard) input

PCs and laptops

13” and greater

shared, one user at a time

keyboard and mouse input

Xbox and TV

24” and up

shared among several people

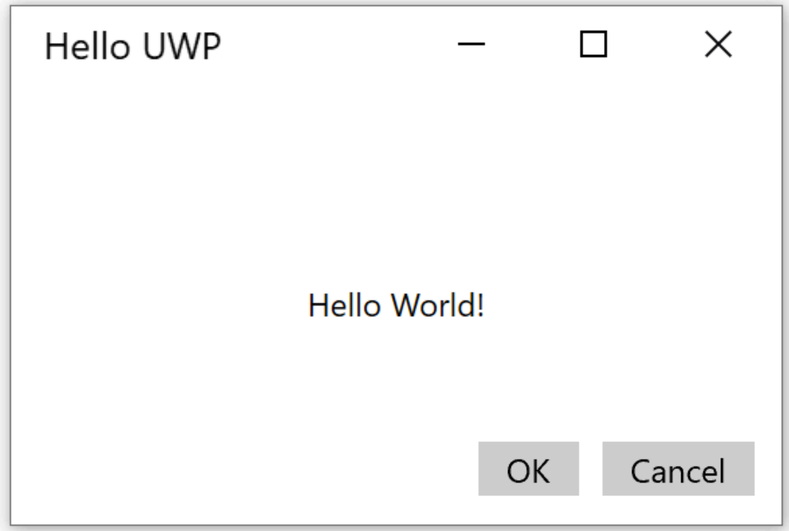
gamepad and remote

Windows Runtime (WinRT)

- Successor of Win32 API, first introduced with Windows 8
- Apps are sandboxed and target x86 and ARM processors
- Same WinRT **across all devices** allows to share code
- Designed to handle touch input, asynchronous calls, different screen sizes
- UWP apps support XAML for UI layout
- Enhanced support for multimedia and UI scaling

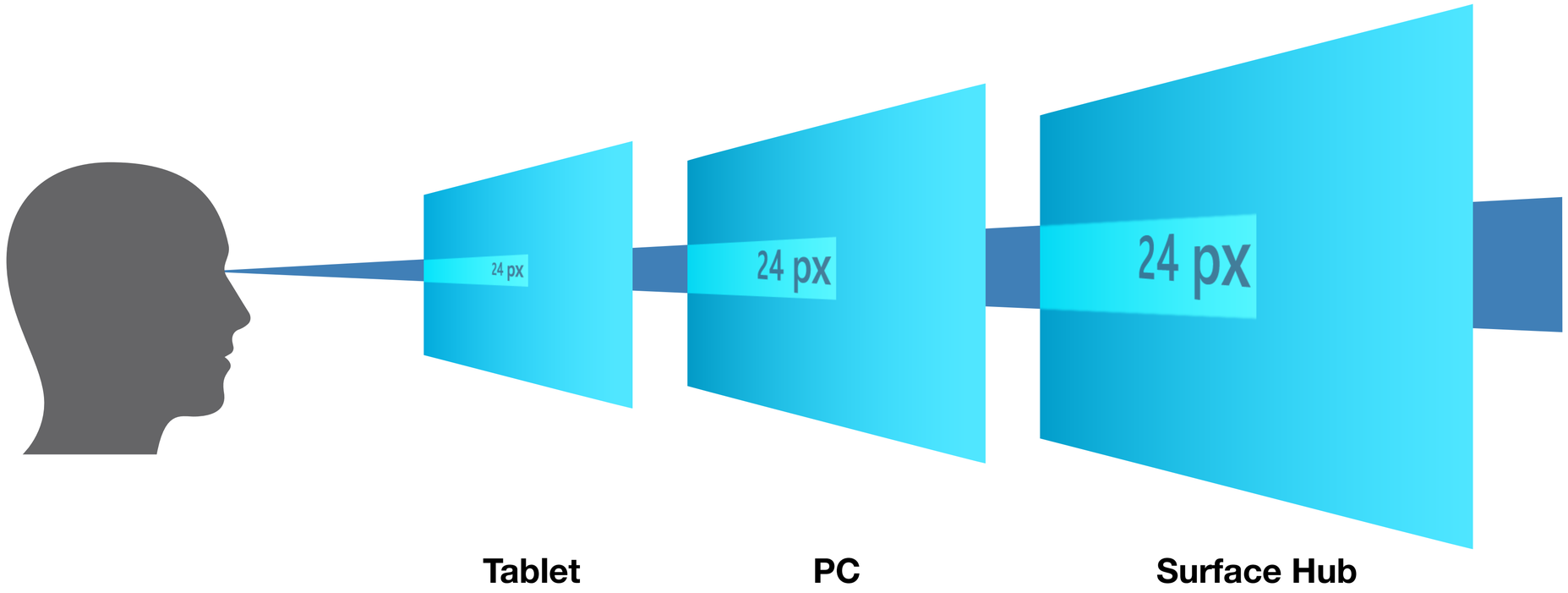
Hello, UWP

```
<Page Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid>
        <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
                   Text="Hello World!" x:Name="label"/>
        <StackPanel HorizontalAlignment="Right" VerticalAlignment="Bottom" Width="Auto"
                    Margin="10,0,10,10" Orientation="Horizontal">
            <Button Content="OK" Padding="10,0" Click="Callback"/>
            <Button Content="Cancel" Margin="10,0,0,0" Padding="10,0"/>
        </StackPanel>
    </Grid>
</Page>
```



Effective Pixels and Scaling

- Content sizes scale depending on the typical user distance for this category
- **Effective pixels** allow developers to design a UI that is legible on all devices

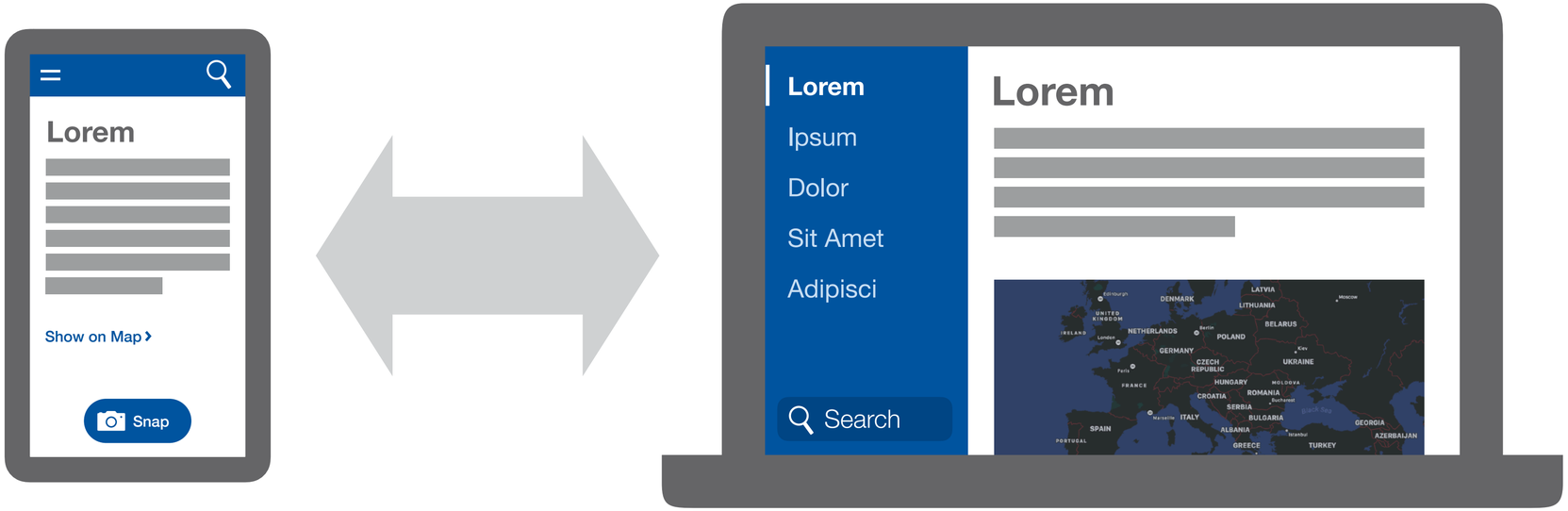


Responsive UIs

- Expand or contract contents to fit the screen
- Reposition, resize, reflow

Adaptive UIs

- Add or remove content based on device family, screen size, …
- Reveal, replace, re-architect



Two Ways to Create Adaptive UIs

- **Different XAML files for different screen sizes**
  - Bad scalability
  - Still can be useful in some scenarios
- **Visual State Triggers and Setters**
  - One XAML for all screen sizes
  - Declare different visual states, triggers select which UI is used

```
<VisualState x:Name="Medium">
  <VisualState.Setters>
    <Setter Target="MyTextBox.FontSize" Value=“24" />
  </VisualState.Setters>

  <VisualState.StateTriggers>
    <AdaptiveTrigger MinWindowWidth="600"/>
  </VisualState.StateTriggers>
</VisualState>

<VisualState x:Name="Small">
  <VisualState.Setters>
    <Setter Target="MyTextBox.FontSize" Value="12" />
  </VisualState.Setters>

  <VisualState.StateTriggers>
    <AdaptiveTrigger MinWindowWidth="0"/>
  </VisualState.StateTriggers>
</VisualState>
```

Adaptive Code

- Not all features are available on all targeted platforms
  But we want one compiled binary for all targets

- Elegant solution: Query that APIs are available on the platform

- All checks are performed at runtime,
  inappropriate blocks will just be skipped

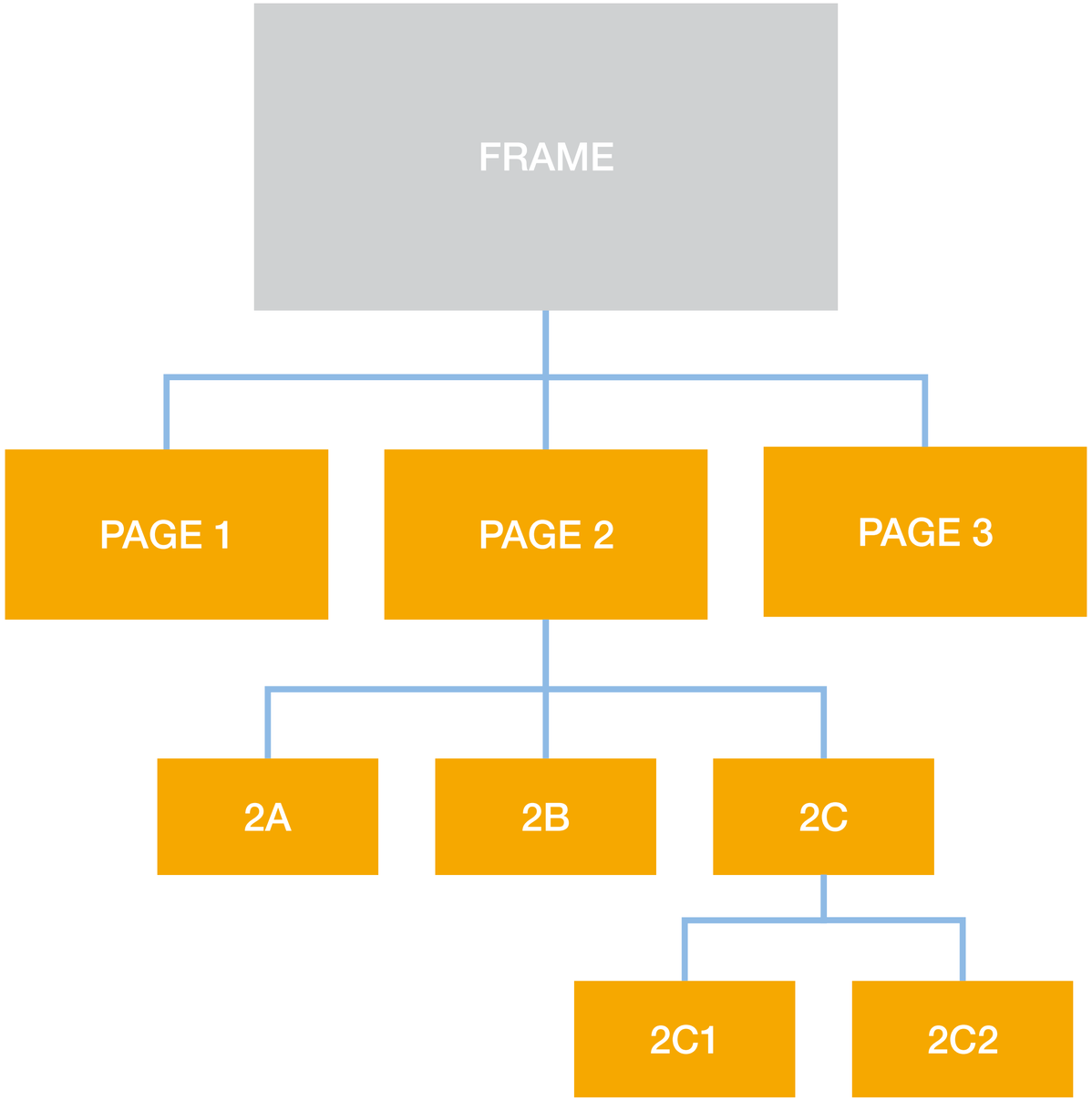
```
var api = "Windows.Phone.UI.Input.HardwareButtons";
if (Windows.Foundation.Metadata.ApiInformation.IsTypePresent(api))
{
  Windows.Phone.UI.Input.HardwareButtons.CameraPressed
        += CameraButtonPressed;
}
```

CHAPTER 24

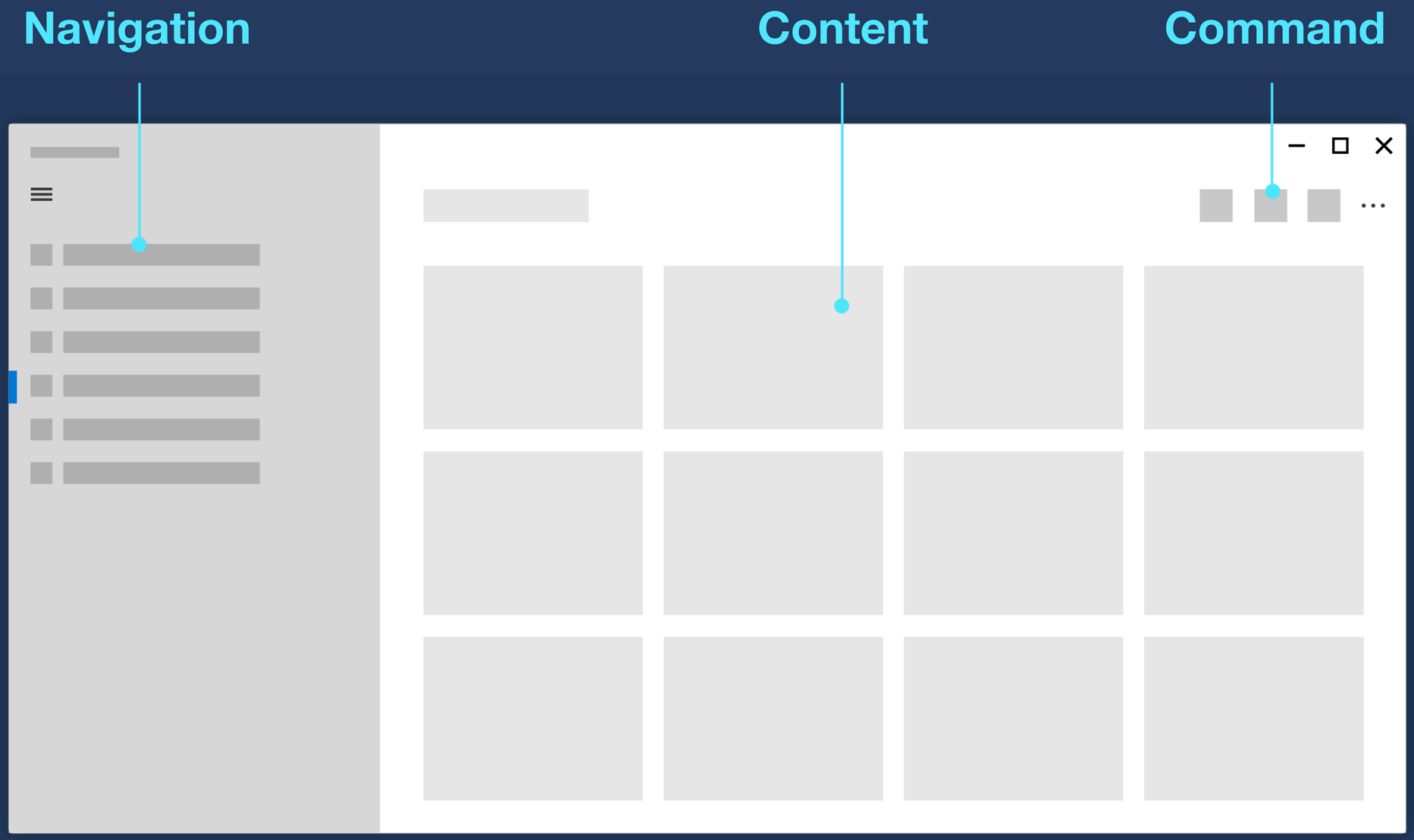
Developing UWP Apps in Visual Studio

Window, Frame, Page

- A UWP is launched in a **window** with a **frame**
- The frame navigates between **pages**
- Some pages contain child pages, e.g. in order to implement a hamburger menu



Page Layout



Switching Pages

- The frame navigates between different pages,
  e.g. as reaction to a click in a list

```
private void ListView_ItemClick(object sender, ItemClickEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;
    var content = e.ClickedItem as DataModelClass;
    rootFrame.Navigate(typeof(OtherPage), content);
}
```

- The new page completely replaces the previous page
  by performing a small animation

Presenting a Back Button

- After switching a page, there is by default no way to go back to the previous one

- You can present a back button in the window's title bar

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility
     = AppViewBackButtonVisibility.Visible;
}
```

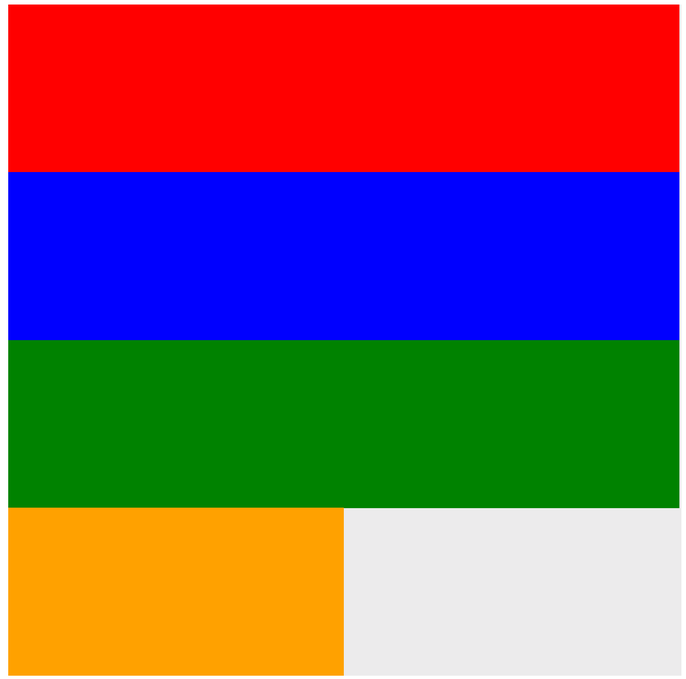
Presenting a Back Button

- You still have to add yourself as handler of the back event and implement a useful callback

```
SystemNavigationManager.GetForCurrentView().BackRequested +=
 App_BackRequested;

private void App_BackRequested(object sender, BackRequestedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame.CanGoBack && e.Handled == false)
    {
        e.Handled = true;
        rootFrame.GoBack();
    }
}
```

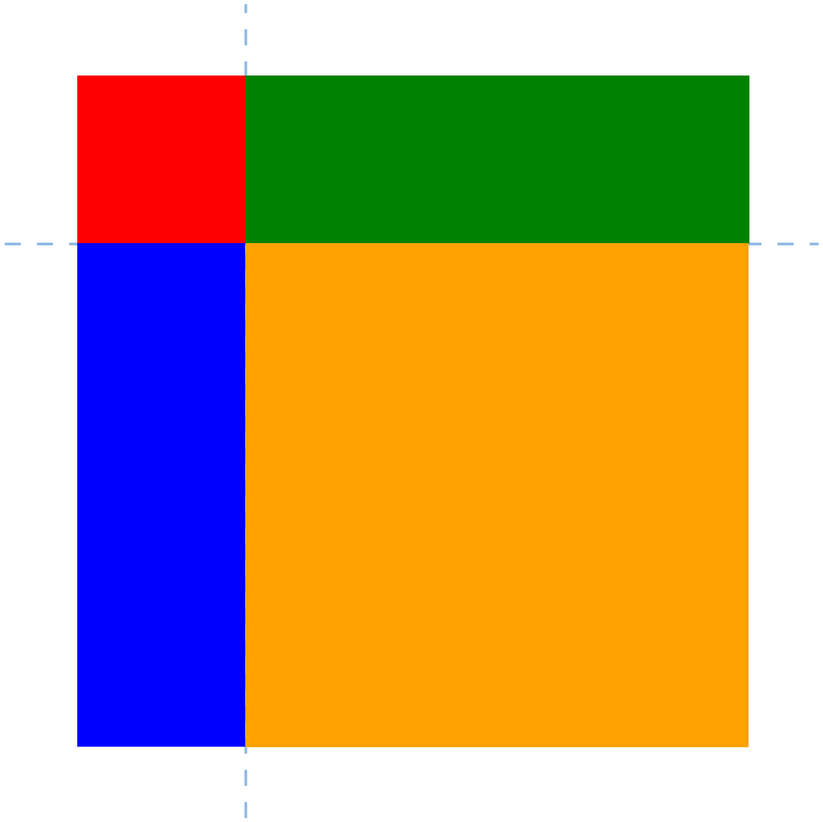
StackPanel

```
<StackPanel Width="176" Height="176">
    <Rectangle Fill="Red"
               Height="44"/>
    <Rectangle Fill="Blue"
               Height="44"/>
    <Rectangle Fill="Green"
               Height="44"/>
    <Rectangle Fill="Orange"
               Height="44"
               Width="88"
               HorizontalAlignment="Left"/>
</StackPanel>
```



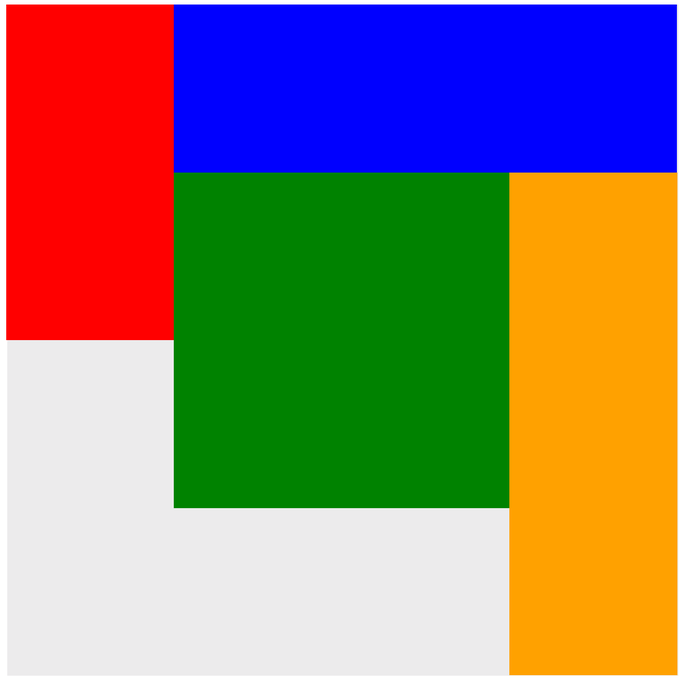
GridPanel

```
<Grid Width="176" Height="176">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="3*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Width="44"/>
    <Rectangle Fill="Blue" Grid.Row="1"/>
    <Rectangle Fill="Green" Grid.Column="1"/>
    <Rectangle Fill="Orange" Grid.Row="1" Grid.Column="1"/>
</Grid>
```



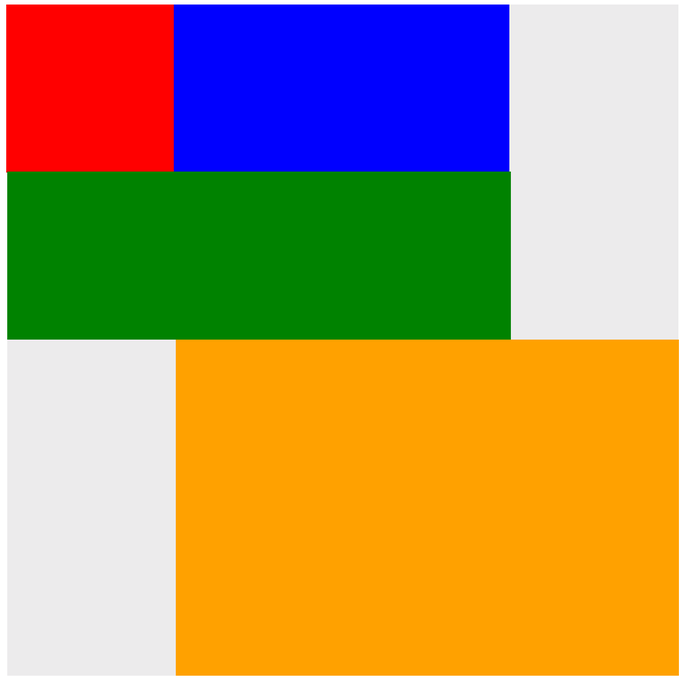
VariableSizedWrapGrid

```
<VariableSizedWrapGrid MaximumRowsOrColumns="4"
                       ItemHeight="44" ItemWidth="44"
                       Orientation="Horizontal">
    <Rectangle Fill="Red"
               VariableSizedWrapGrid.RowSpan="2"/>
    <Rectangle Fill="Blue"
               VariableSizedWrapGrid.ColumnSpan="3"/>
    <Rectangle Fill="Green"
               VariableSizedWrapGrid.RowSpan=“2"
               VariableSizedWrapGrid.ColumnSpan="2"/>
    <Rectangle Fill="Orange"
               VariableSizedWrapGrid.RowSpan="3"/>
</VariableSizedWrapGrid>
```



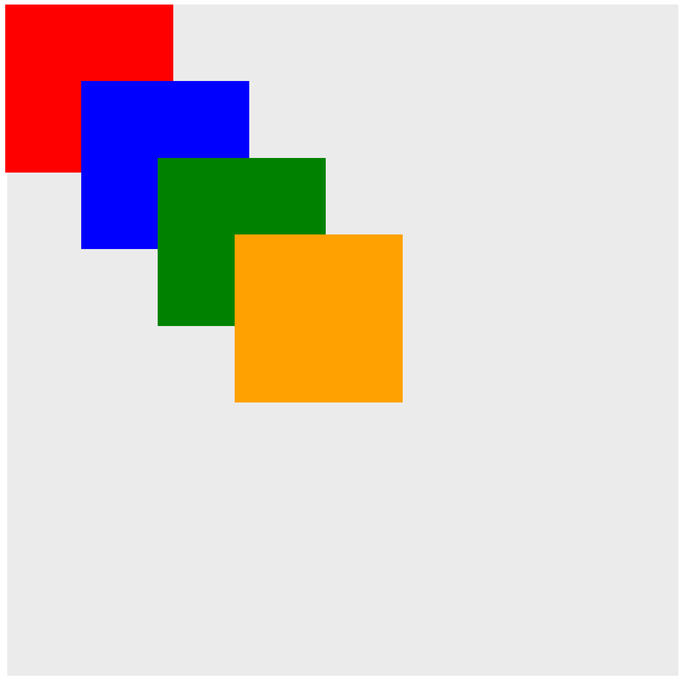
RelativePanel

```
<RelativePanel Width="176" Height="176">
    <Rectangle x:Name="RedRect" Fill="Red"
               Height="44" Width="44"/>
    <Rectangle x:Name="BlueRect" Fill="Blue"
               Height="44" Width="88"
               RelativePanel.RightOf="RedRect" />
    <Rectangle x:Name="GreenRect" Fill="Green"
               Height="44"
               RelativePanel.Below="RedRect"
               RelativePanel.AlignLeftWith="RedRect"
               RelativePanel.AlignRightWith="BlueRect"/>
    <Rectangle Fill="Orange"
               RelativePanel.Below="GreenRect"
               RelativePanel.AlignLeftWith="BlueRect"
               RelativePanel.AlignRightWithPanel="True"
               RelativePanel.AlignBottomWithPanel="True"/>
</RelativePanel>
```

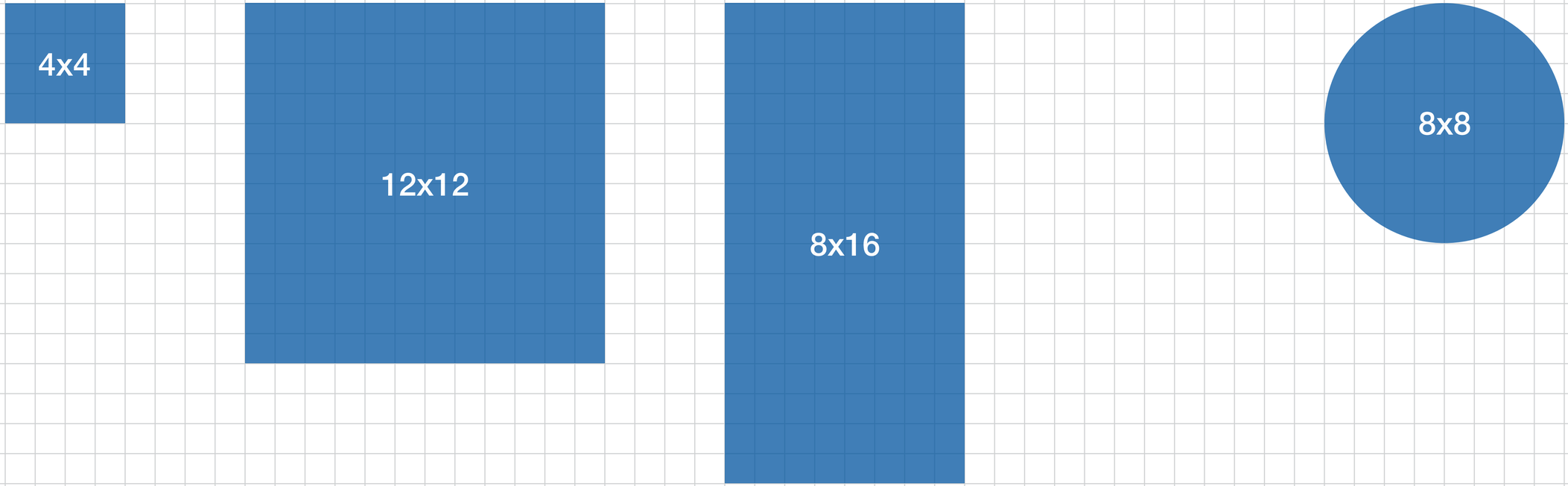


Canvas

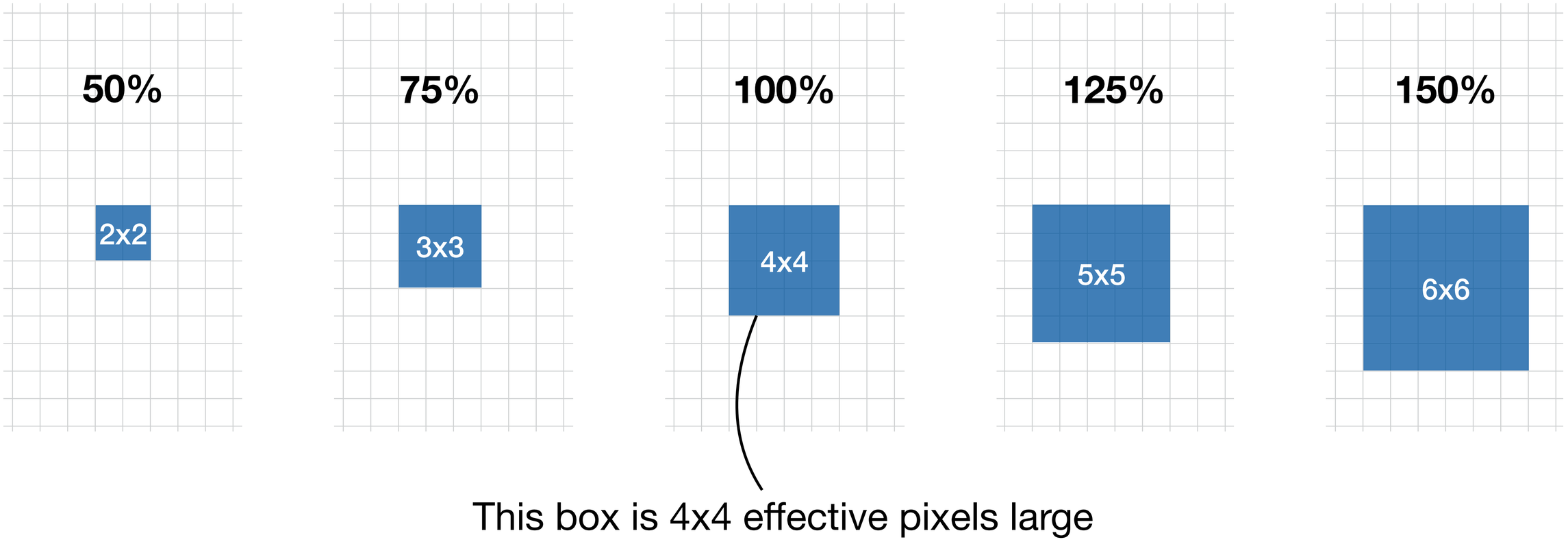
```
<Canvas Width="176" Height="176">
    <Rectangle Fill="Red"
               Height="44" Width="44"/>
    <Rectangle Fill="Blue"
               Height="44" Width="44"
               Canvas.Left="20" Canvas.Top="20"/>
    <Rectangle Fill="Green"
               Height="44" Width="44"
               Canvas.Left="40" Canvas.Top="40"/>
    <Rectangle Fill="Orange"
               Height="44" Width="44"
               Canvas.Left="60" Canvas.Top="60"/>
</Canvas>
```



Multiples of Four



Multiples of Four



Demo