PV239 Windows 10
Cvičení 3

Mgr. David Gešvindr
MVP: Azure | MCSD: Windows Store | MCSE: Data Platform | MCT | MSP
gesvindr@mail.muni.cz
@gesvindr

Jak začít s návrhem aplikace

- Seznamte se s UI design guidelines pro Windows 10
  - UI by mělo být v souladu s UI systému – uživatelé nemají rádi nekonzistence

- **Představte si svého uživatele**
  - **K čemu mu vaše aplikace bude?**
  - **Proč ji bude používat?**
  - **Jak ji bude nejčastěji používat**

- Identifikujte nejdůležitější scénáře použití
- Udělejte je co **nejjednodušší**

Dependecy Injection

- Návrhový vzor, kdy třída přenechává zodpovědnost za poskytnutí závislostí externímu kódu

- Např.: Třída načítající data z webové služby by si sama mohla vytvořit instanci webového klienta, ale místo toho nechá klienta vytvořit někoho externího
- Implementace třídy pracuje pouze s rozhraním popisujícím webového klienta

```
public interface IHttpClient
{
    Task<string> GetStringAsync(Uri uri);
}
```

Dependecy Injection

- Třída si uloží předanou závislost v konstruktoru jako svůj stav

```
public class ServiceEventProvider
{
    IHttpClient httpClient;

    public ServiceEventProvider(IHttpClient httpClient)
    {
        this.httpClient = httpClient;
    }

    // Při volání webové služby se používá injektovaný
    // httpClient

}
```

Výhody Dependency Injection

- Přehlednější správa závislostí tříd mezi sebou
- Jednoduchá změna závislostí s jistotou, že se nic nerozbije
- Nezávislý vývoj více jasně popsaných komponent současně
- Dobře testovatelný kód s pomocí unit testů
  - Možnost v testu vložit jinou závislost, která nebude pracovat se skutečnými daty, ale jen testovacími daty jejich výsledek půjde porovnat

Dependency Injection Container

- Pomocný framework, který umožňuje jednoduše spravovat závislosti

```
// Při startu aplikace zaregistrujeme závislosti
UnityContainer container = new UnityContainer();
container.RegisterType<IHttpClient, PlatformSpecific.HttpClient>();

// Pokud potřebujeme instanci závislosti, DI kontejner ji vyrobí
IHttpClient client = container.Resolve<IHttpClient>();
```

- Na UWP je podporován např. DI container Unity:
- https://github.com/unitycontainer/unity

Návrhový vzor MVVM

- Doporučuje se používat návrhový vzor

Model – View – ViewModel

- **Model**
  - Reprezentuje business vrstvu aplikace zcela nezávislou na uživatelském rozhraní
  - Aplikační logika pro získání a uložení dat
  - Datové entity

Návrhový vzor MVVM

- **ViewModel**
  - **ViewModel** obsahuje data získaná z modelu, která budou zobrazena v uživatelském rozhraní
  - Může provádět transformaci dat, která je specifická pro dané zobrazení
  - Obsahuje příkazy, které mohou být spuštěny z uživatelského rozhraní, obsluha dané operace může být předána modelu
  - ViewModel může být kompozitní
    - Více ViewModelů spojených dohromady
    - Např. ViewModel hlavní obrazovky rozdělený na více „fragmentů“

Návrhový vzor MVVM

- **View**
  - Je tvořen uživatelským rozhraním definovaným v XAML
  - K danému View je vytvořena instance ViewModelu, který uchovává všechna data a stav daného UI
  - Propojení View a ViewModelu je realizováno pomocí **Data Bindingu**
    - Je to vazba, která popisuje, která data vzít z ViewModelu (libovolného objektu) a jak je zobrazit v kterém ovládacím prvku
    - Je možné realizovat i databinding příkazů
  - View neobsahuje žádnou aplikační logiku s výjimkou obsluhy událostí měnících pouze zobrazení

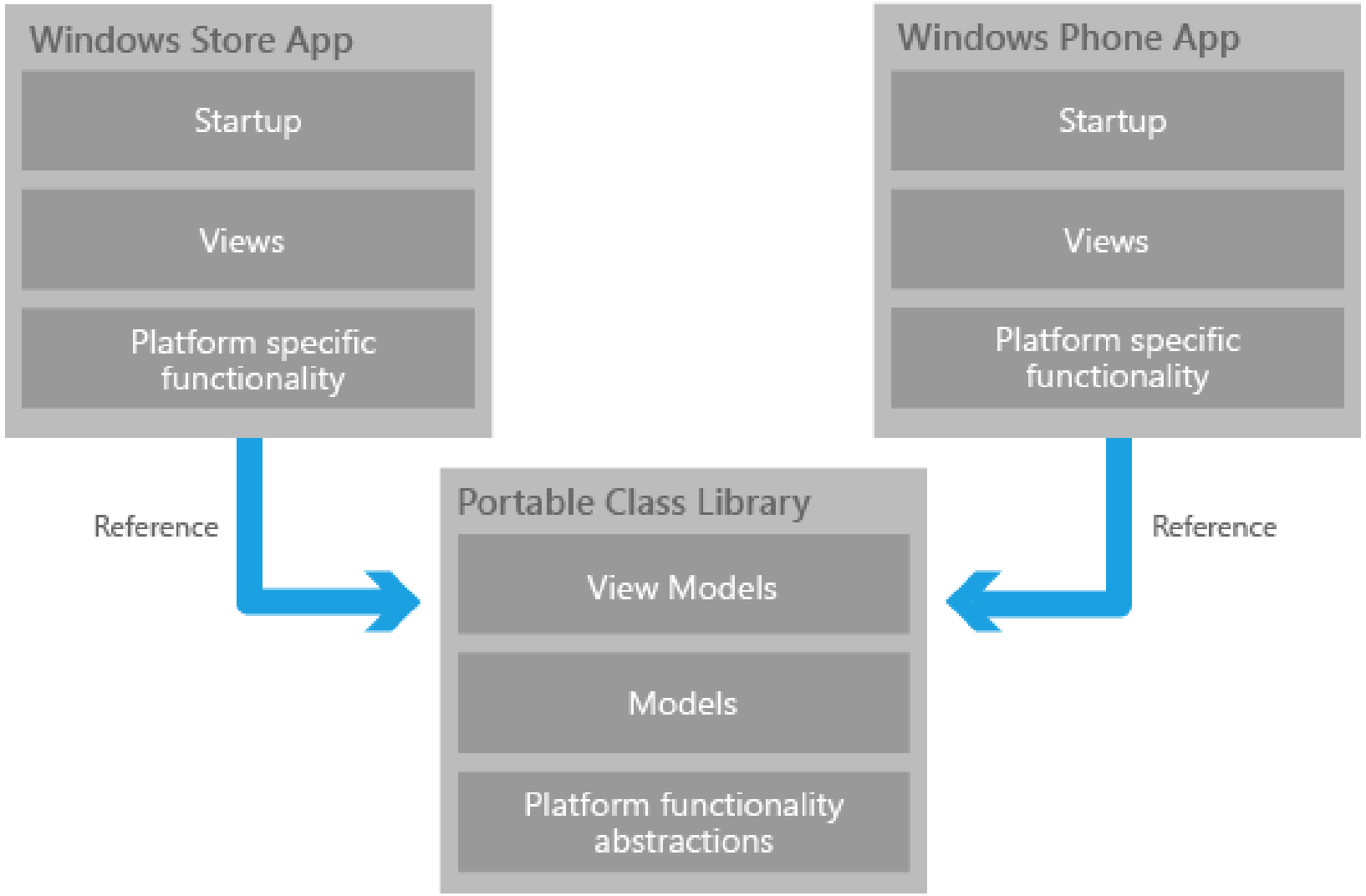
Sdílení kódu mezi více platformami

- V případě Windows Universal application díky jednotnosti API může být vrstva model plně závislá na platformě a celá aplikace může být tvořena jedním projektem

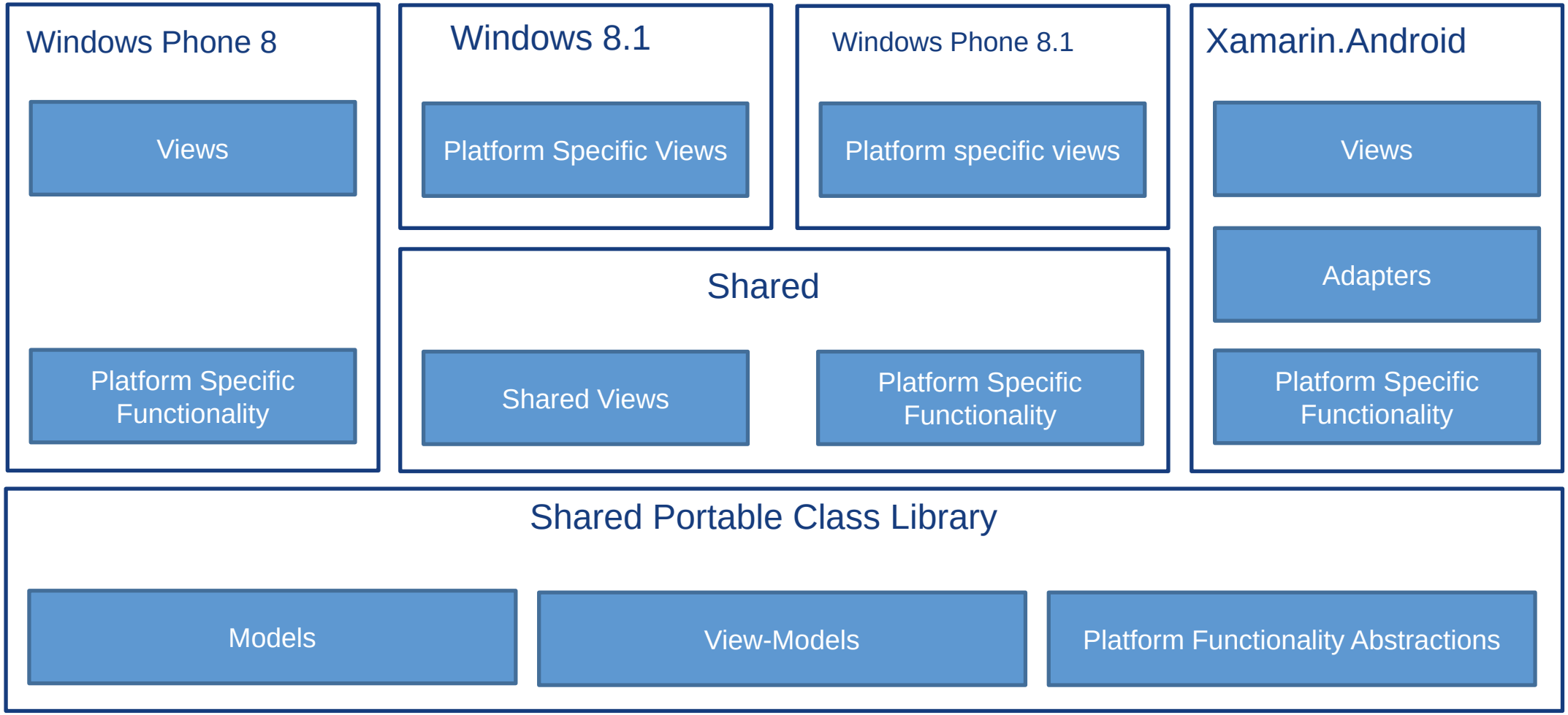
- **Nešlo by ale vrstvu model a view-model udělat platformě nezávislou a znovupoužít ji i na jiné platformě?**

- Výhody:
  - Další úspory při vývoji aplikace na Android a iOS s využitím nástrojů Xamarin (70-85% sdílení kódu)

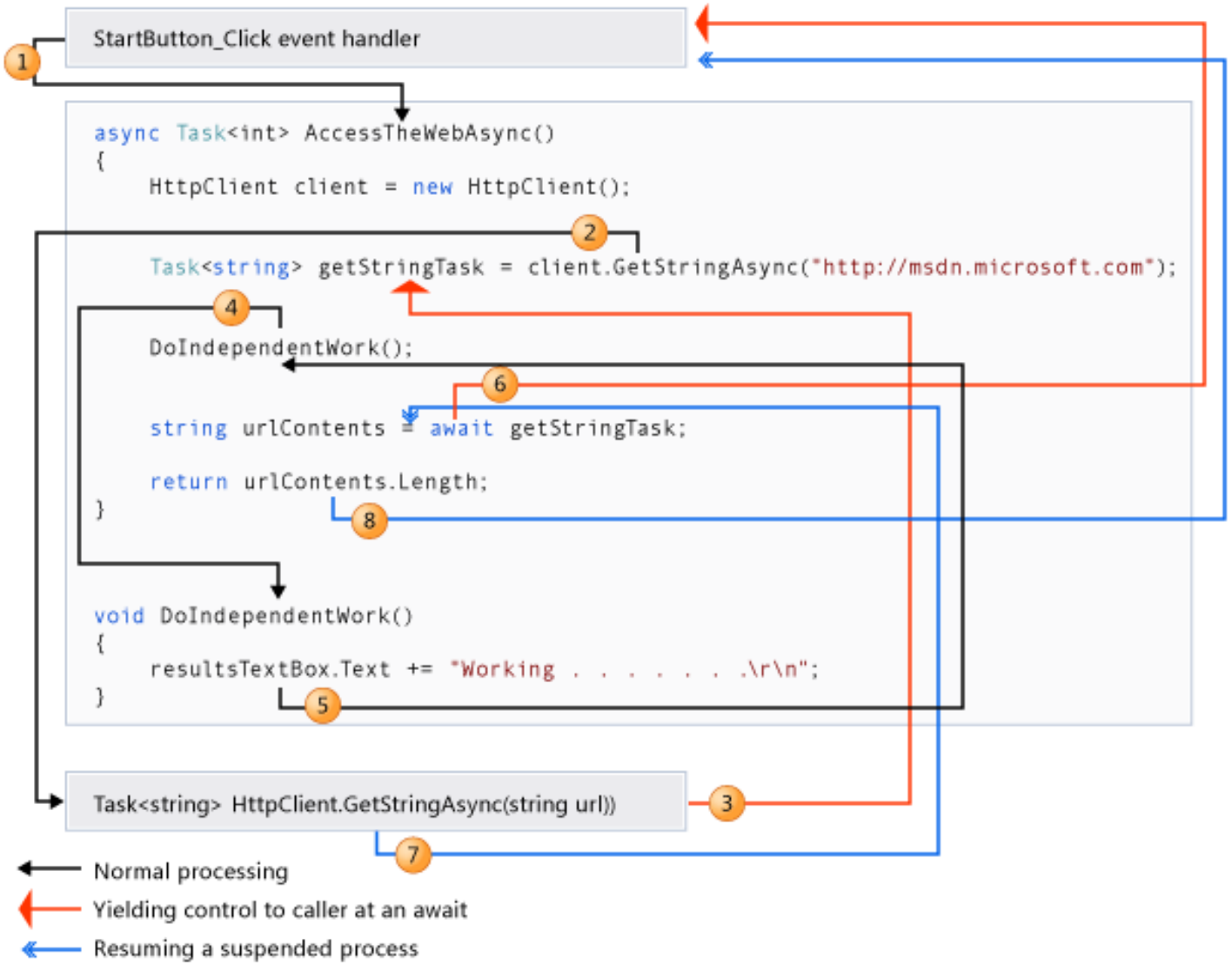
Sdílení kódu mezi více platformami



Sdílení kódu mezi více platformami



Asynchronní zpracování



Komunikace s webovými službami

- Nemalá část aplikací je vyvíjena jako klient pro existující webovou službu
- Výhody oproti webové stránce
  - Optimalizované uživatelské rozhraní
  - Efektivnější datová komunikace
  - Možnost využívat specifická API systému
- Způsob komunikace záleží na technologii, kterou webová služba využívá
- Nejčastěji se využívají protokoly
  - REST
  - OData
  - SOAP

Komunikace přes REST

- Webová služba poskytuje serializovaná data na určené webové adrese
- Např:
  - www.gopas.cz/api/courses
  - www.gopas.cz/api/courses/548

- Pro stažení dat se využívá protokol HTTP metoda **GET**
- Data se nejčastěji serializují do formátu
  - JSON
  - XML

Komunikace přes REST

- Jsou definovány operace, které vycházejí z HTTP metod
  - **GET** – stažení dat
  - **POST** – vložení dat
  - **PUT** – aktualizace dat
  - **DELETE** – odstranění dat

- V aplikaci je pro implementaci využit *HttpClient*. Který posílá HTTP požadavky na server
- Obsah a zpracování požadavků je v režii vývojáře

Použití HttpClient

- Umožňuje vytvořit, odeslat a zpracovat HTTP požadavek
- Hlavní metody:
  - GetStringAsync
  - GetAsync

```
try
{
    // Create a New HttpClient object.
    HttpClient client = new HttpClient();
    // Connect to the data source asynchronously and get the data.
    string responseBody = await client.GetStringAsync("http://www.microsoft.com/enus/news/rss/rssfeed.aspx");
    // Process data
}
catch(HttpRequestException ex)
{
    // Process exception
}
```

Úložiště aplikačních dat

- Každá aplikace má izolované úložiště dat plně pod správou systému
- Vzniká s instalací aplikace, je odstraněno s aplikací
- Obsahuje následující druhy úložiště:
  - **ApplicationData.Current.LocalFolder**
    - Trvalé úložiště dat na daném zařízení
  - **ApplicationData.Current.TemporaryFolder**
    - Dočasné úložiště dat
    - Systém může data kdykoliv odstranit
  - **ApplicationData.Current.RoamingFolder**
    - Úložiště synchronizované mezi zařízeními s danou aplikací
    - ApplicationData.RoamingStorageQuota;

Uložení aplikačních dat

- Přístup je možný i přes URI adresy

```
StorageFile file = await StorageFile.GetFileFromApplicationUriAsync(
    new Uri("msappdata:///local/file.txt"));
```

- Lokální úložiště:
  - ms-appdata:///local/
- Dočasné úložiště
  - ms-appdata:///temporary/
- Synchronizované úložiště
  - msappdata:///roaming/
- Instalační adresář
  - ms-appx:///

```
Windows.Storage.StorageFolder installedLocation =
    Windows.ApplicationModel.Package.Current.InstalledLocation;
```

Práce s obsahem souborů

- Použití pomocné třídy **Windows.Storage.FileIO**
- Pracuje se souborem jako celkem
  - Vhodné pro malé soubory

- Čtení dat
  - IAsyncOperation<IList> ReadLinesAsync(IStorageFile), IAsyncOperation<string> ReadTextAsync(IStorageFile)
- Zápis dat
  - WriteLinesAsync(IStorageFile, IEnumerable<string>)
  - WriteTextAsync(IStorageFile, String)
  - WriteBytesAsync(IStorageFile, byte[])

Práce se streamy dat

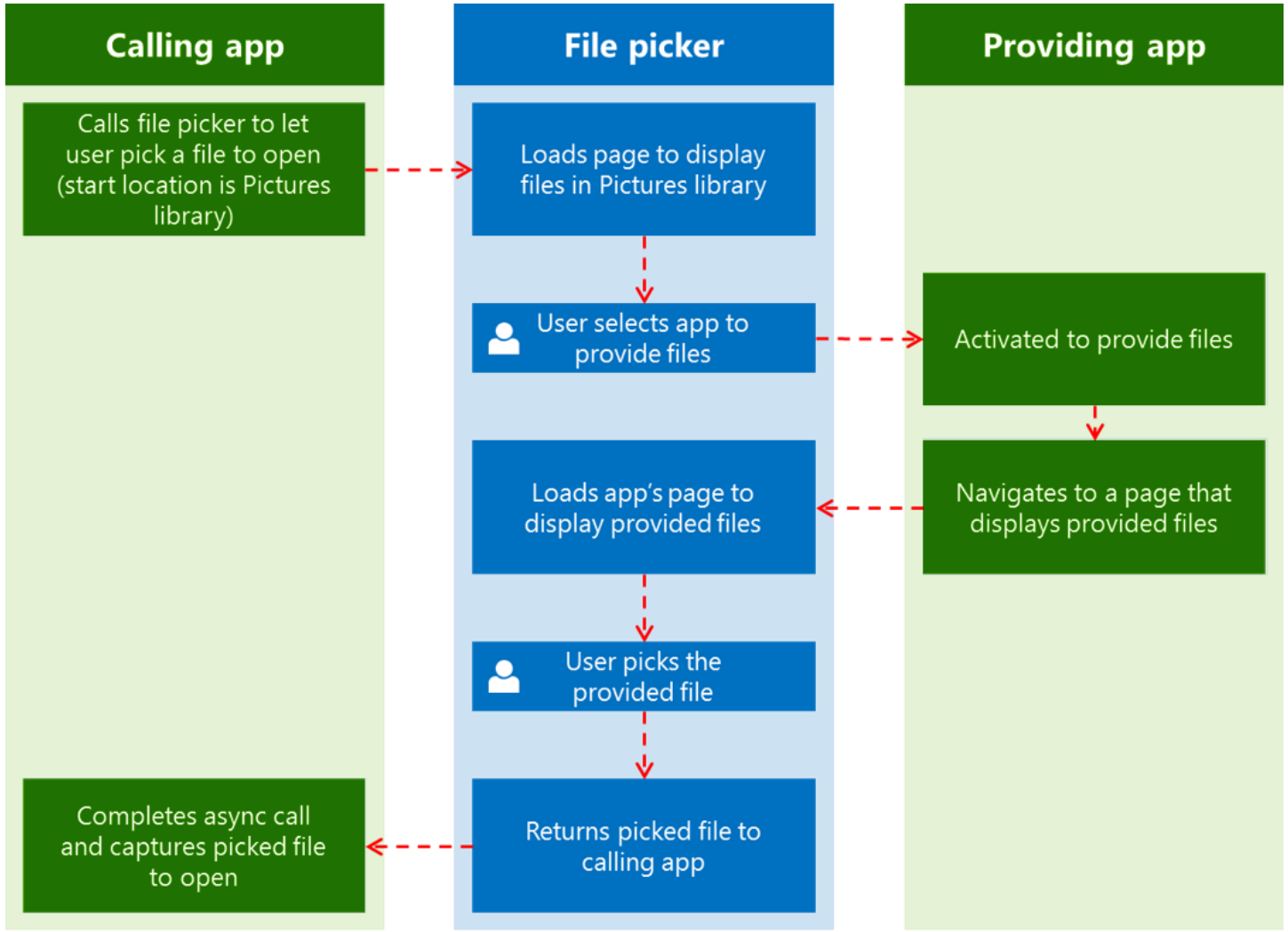
- Soubor se otevře, ale data čteme nebo zapisujeme dle potřeby po částech

```
StorageFolder storageFolder = KnownFolders.DocumentsLibrary;
StorageFile stockFile = await storageFolder.CreateFileAsync("stockData.txt",
    CreationCollisionOption.OpenIfExists);

var stream = await stockFile.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);

using (var outputStream = stream.GetOutputStreamAt(0))
{
    DataWriter dataWriter = new DataWriter(outputStream);
    dataWriter.WriteString("Zápis dat do souboru");
}
```

FilePickers



FilePickers

- Pro výběr souboru pro čtení:
  - FileOpenPicker.PickSingleFileAsync
  - FileOpenPicker.PickMultipleFilesAsync
- Pro výběr souboru pro zápis:
  - FileSavePicker.PickSaveFileAsync
- Pro výběr adresáře:
  - FolderPicker.PickSingleFolderAsync
- Nastavte výchozí složku: SkyDrive, Desktop, ...

FilePickers

```
FileOpenPicker openPicker = new FileOpenPicker();
openPicker.ViewMode = PickerViewMode.Thumbnail;
openPicker.SuggestedStartLocation = PickerLocationId.PicturesLibrary;
openPicker.FileTypeFilter.Add(".gif");
openPicker.FileTypeFilter.Add(".jpeg");
openPicker.FileTypeFilter.Add(".png");
StorageFile file = await openPicker.PickSingleFileAsync();
if (file != null)
{
    // Aplikace má nyní přístup k souboru
}
else
{
    // Operace zrušena
}
```

Uložení nastavení

- Nastavení není nutné ručně ukládat do souborů
- Existuje slovník pro uložení nastavení
  - `Windows.Storage.ApplicationData.Current.LocalSettings`
  - `Windows.Storage.ApplicationData.Current.RoamingSettings`
- Postup pro uložení a načtení nastavení:

```
var localSettings = Windows.Storage.ApplicationData.Current.LocalSettings;

// Uložení nastavení
localSettings.Values["exampleSetting"] = "Hello Windows";

// Načtení nastavení
Object value = localSettings.Values["exampleSetting"];
```

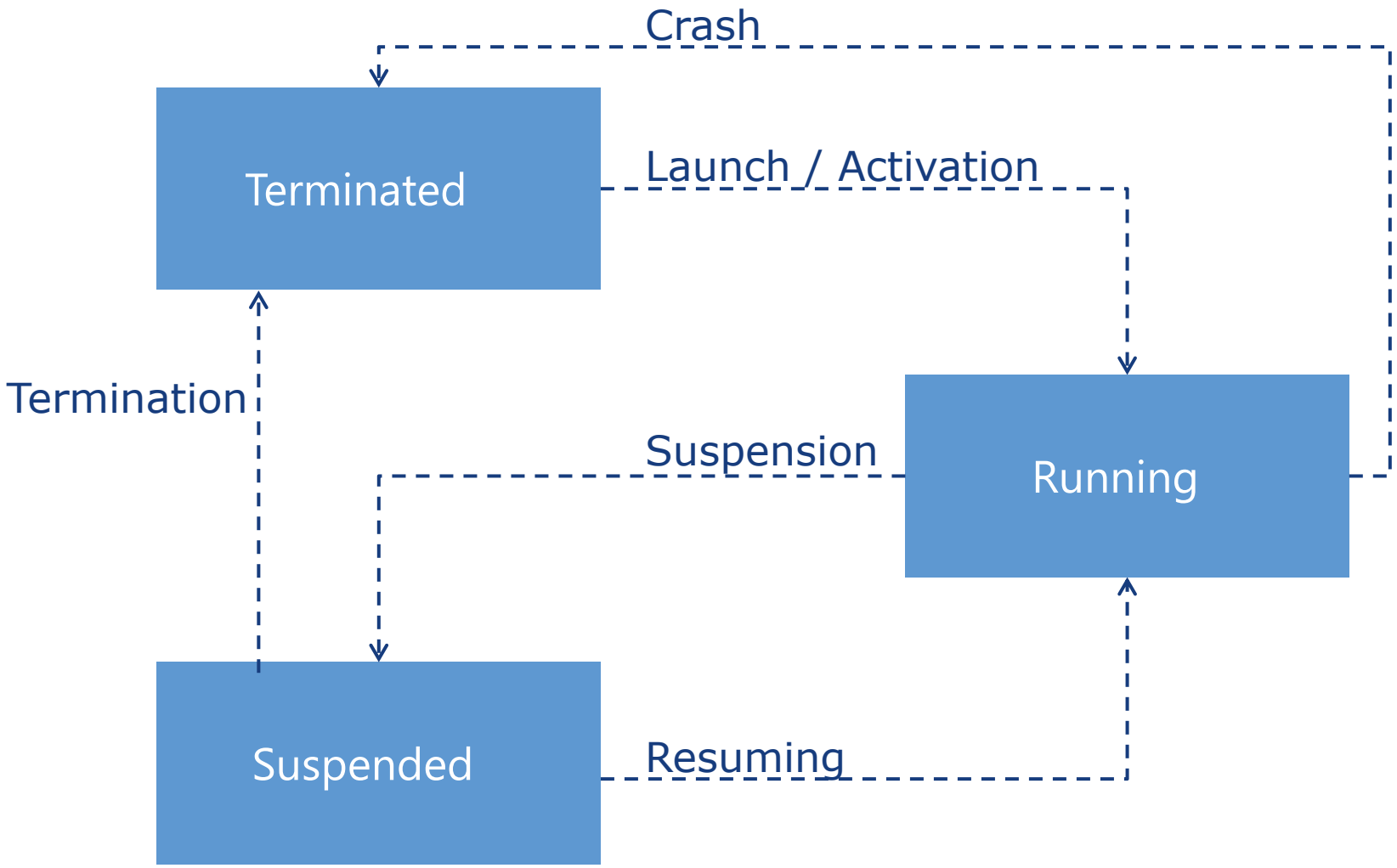
Drag&Drop souborů do aplikace

- Je možné přetáhnout soubor z průzkumníku do aplikace, která jej načte
- Stačí obsloužit událost Drop nad cílovým ovládacím prvkem:

```
private async void RelativePanel_Drop(object sender, DragEventArgs e)
{
    var storageItems = await e.DataView.GetStorageItemsAsync();
    var storageItem = storageItems.First();
    var storageFile = storageItem as StorageFile;

    if (storageFile != null)
    {
        // Zpracujeme soubor
    }
}
```

Kdy je aplikace spuštěna



K čemu dochází při uspání aplikace

- Aplikace není zobrazena v popředí, bude systémem pozastavena
- Je vyvolána událost **suspending,** na kterou může být navázána v aplikaci vlastní obsluha
- Je vhodné provést:
  - Uložení stavu aplikace do persistentní paměti
  - Uvolnění nepotřebných zdrojů
  - Aktivaci procesů na pozadí
- K uspání dochází 5 vteřin po opuštění aplikace
- Maximální doba pro zpracování události **suspending** je **5 vteřin**

Probuzení aplikace

- Probuzení ze stavu **suspended** není třeba řešit, aplikace zůstává v paměti
- Pokud ale systém aplikaci úplně ukončí a uživatel se do ní vrátí, je na nás obnovit stav aplikace, jako by běžela stále
- Rozhoduje hodnota **PreviousExecutionState**:
  - **Terminated** – Ukončena systémem, obnovit stav
  - **ClosedByUser** – Ukončena uživatelem, nový běh
  - **NotRunning** – Nebyla od přihlášení spuštěna, nový běh

Uložení stavu aplikace

- Uložení stavu aplikace je zcela v rukou vývojáře
- Pokud dojde k ukončení aplikace (stav *terminated*) je stav aplikace zcela ztracen
  - Nebyla dlouho použita
  - Systém potřeboval uvolnit paměť
  - Chyba v aplikaci

- Stav je třeba uložit v události **suspending**, protože přechod do stavu terminated už žádnou událost nevyvolá!

Navigační framework

- Přechody mezi stránkami aplikace jsou realizovány pomocí navigačního frameworku
- Navigační framework poskytuje svoji funkcionalitu srze třídu **Frame**

- `Frame.Navigate(typeof(MainPage), txtTextToPass.Text);`

Předávání parametrů mezi stránkami

- Pro otevření stránky a předání parametrů využijte metodu: **Frame.Navigate(PageType, Object)**
  - Je možné předávat pouze primitivní typy
    - String
    - Bool
    - Integer
    - Guid
- Parametr je načtený v cílové stránce v metodě *OnNavigatedTo:*

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    string passedData = e.Parameter as string;
}
```

Události spojené s navigací

- **OnNavigatingFrom**
  - Volána před opuštěním aktuální stránky a načtením nové
  - Možnost zrušit přechod:
    - NavigationCancelEventArgs.Cancel = true;
- **OnNavigatedFrom**
  - Volána po opuštění stránky před jejím uvolněním z paměti
- **OnNavigatedTo**
  - Volána na cílové stránce před zobrazením
  - Možnost načíst předávané parametry

Historie navštívených stránek

- Další metody pro navigaci mezi stránkami:
  - Frame.GoBack, Frame.GoForward
  - Frame.CanGoBack, Frame.CanGoForward

- Navigační Framework podporuje cachování stránek
- Je uložen stav stránky a při návratu na stránku je obnoven
- Třeba povolit v atributu stránky
  - NavigationCacheMode = NavigationCacheMode.Enabled;