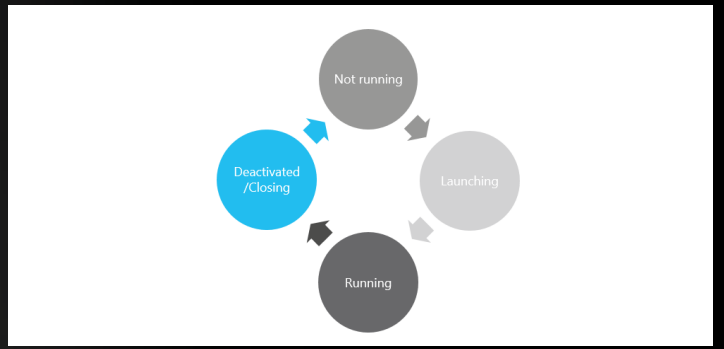
Application Lifecycle



Windows Phone

Agenda

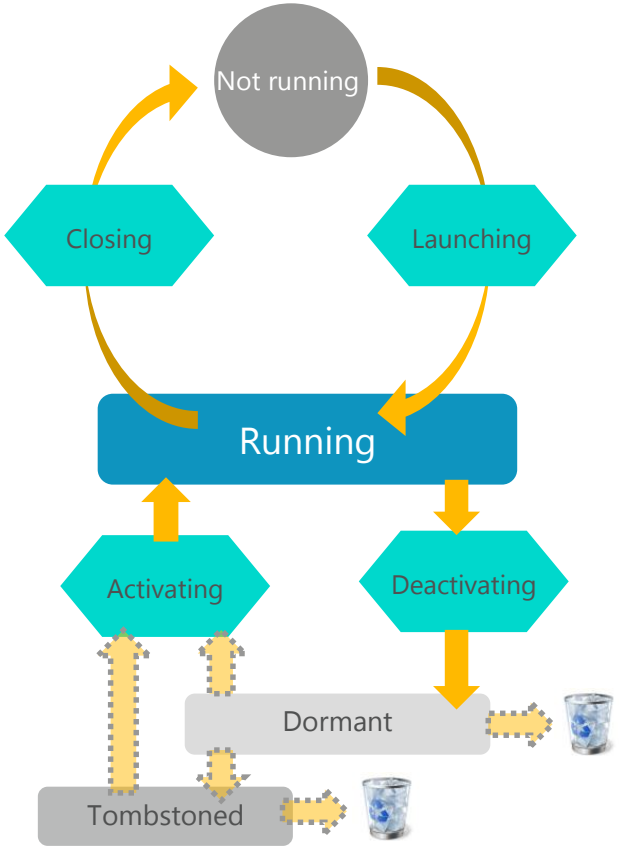
In This Session…

- Windows Phone 8 program lifecycle
  - Launching and Closing
  - Deactivating and Activating
  - Dormant and Tombstoned applications
  - Simulation Dashboard
- Idle Detection on Windows Phone
  - Detecting Obscured events
- Fast Application Resume
- Lifecycle design
- Page Navigation and the Back Stack

Windows Phone Program Lifecycle

Windows Phone Application Lifecycle

- Windows Phone apps transition between different application states
  - Apps are launched from Start Screen icon, apps menu or from deep link
  - User may close apps
  - The OS will suspend your app if it loses focus
    Suspended apps may be tombstoned
  - Apps may be reactivated from a suspended state
- When the user starts a new instance of your app, any suspended instance is discarded
  - In Windows Phone 8.0, you can enable Fast Application Resume to relaunch the suspended instance



Application Lifecycle Events

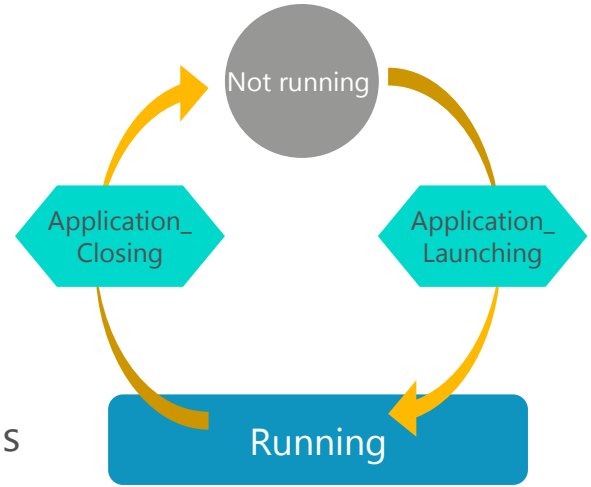
```
// Code to execute when the application is launching (eg, from Start)
// This code will not execute when the application is reactivated
private void Application_Launching(object sender, LaunchingEventArgs e)
{
}
```

- The Windows Phone application environment notifies an application about lifecycle events through a set of events
- In the project templates, the events are already wired up in `App.xaml` and the event handler methods are in the `App.xaml.cs` file
- Initially the event handler methods are empty

Demo 1: Launching and Closing

Launching and Closing – what we have seen

- When a Windows Phone 8 application is started the Application_Launching event handler method is called
  - Good place to load content from backing store
- When a Windows Phone 8 application is ended the Application_Closing event handler method is called
  - Good place to save content in backing store
- The debugger will keep running even when your application is "stopped"
  - This makes it easier to debug start and stop behaviour
  - But you need to stop it yourself to work on your code..



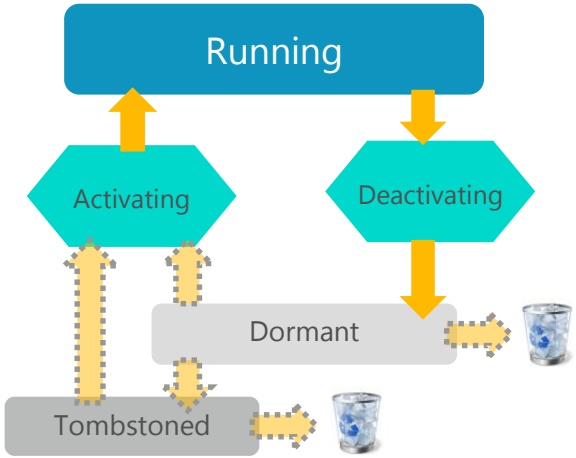
Application Deactivation and Reactivation

- If you are from a Windows desktop background this behavior will make perfect sense
- Windows desktop applications can subscribe to events that are fired when the program starts and ends
- However, Windows Phone applications operate on a smartphone and the OS imposes restrictions designed to save battery life and ensure a good end-user experience
  - At any give time only one applications is in the foreground
  - Users may deactivate their applications and reactivate them later
- Applications have to deal with being activated and deactivated

Demo 2: Deactivating and Activating

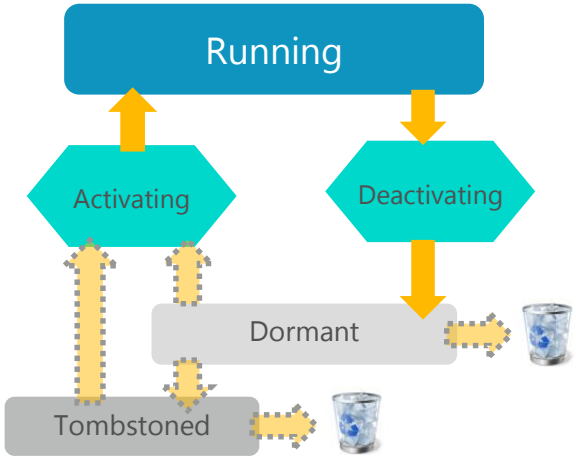
Dormant applications

- The user can make your application dormant at any time
  - They could launch another program from the start screen or the Programs menu
  - The Application_Deactivated method is called in this situation
- External events may also make your application dormant
  - If a phone call is received when your program is running it will be made dormant
  - If the lock screen shows after a period of inactivity
- A user may return to your application and resume it at a later time



Dormant applications

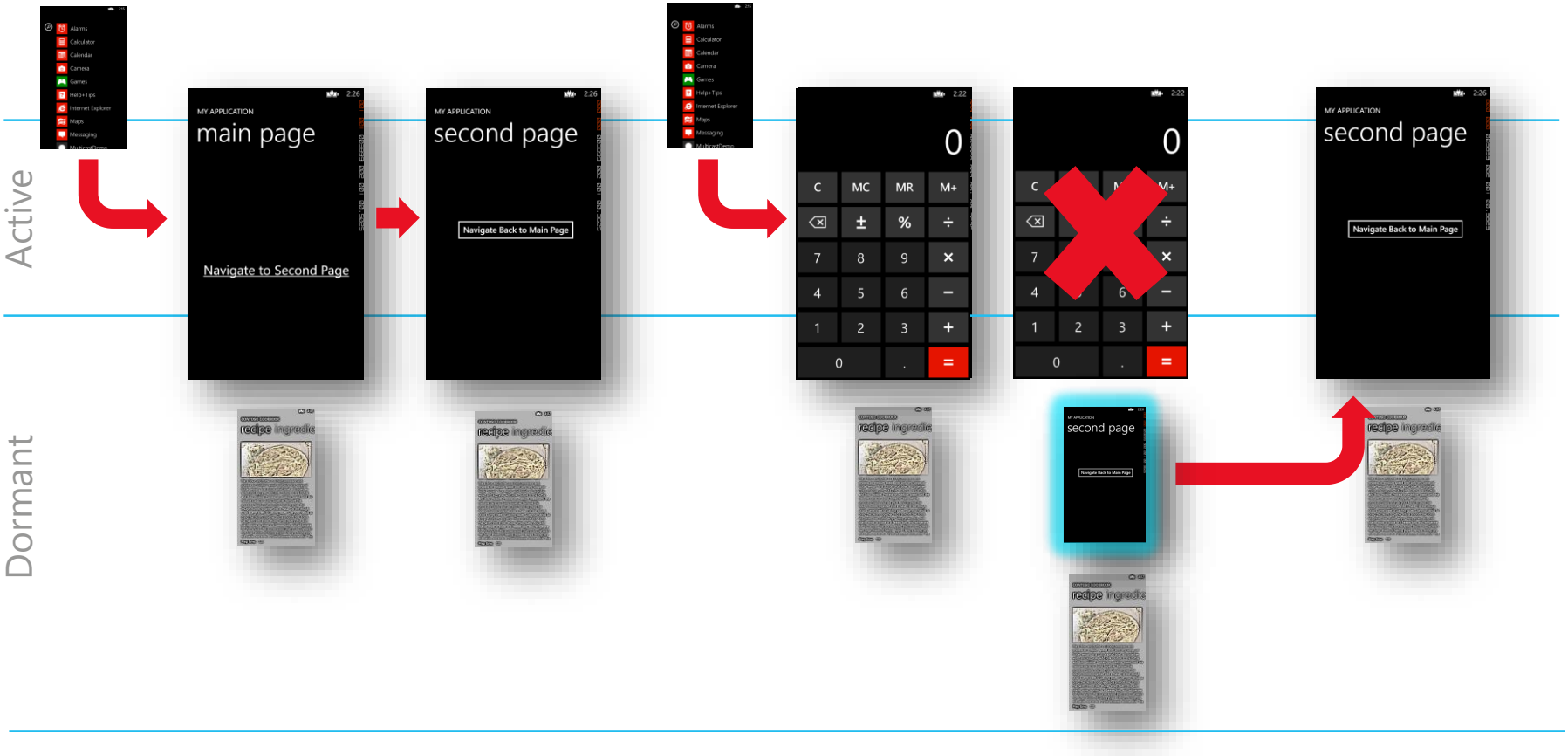
- The user can make your application dormant at any time
  - They could launch another program from the start screen or the Programs menu
  - The Application_Deactivated method is called in this situation
- External events may also make your application dormant
  - If a phone call is received when your program is running it will be made dormant
  - If the lock screen shows after a period of inactivity
- A user may return to your application and resume it at a later time



There is no guarantee that your application will ever be resumed if it is made dormant

Dealing with Dormant

- When your application is made dormant it must do as much data persistence as if it was being closed
  - Because Application_Deactivated and Application_Closing will mean the same thing if the user never comes back to your program
- Your application can run for up to 5 seconds to perform this tidying up
  - After that it will be forcibly removed from memory
- When an application is resumed from dormant it will automatically resume at the page where it was deactivated
  - This behaviour is managed by the operating system
  - All objects are still in memory exactly as they were at the moment the app was suspended



Reactivating from Dormant

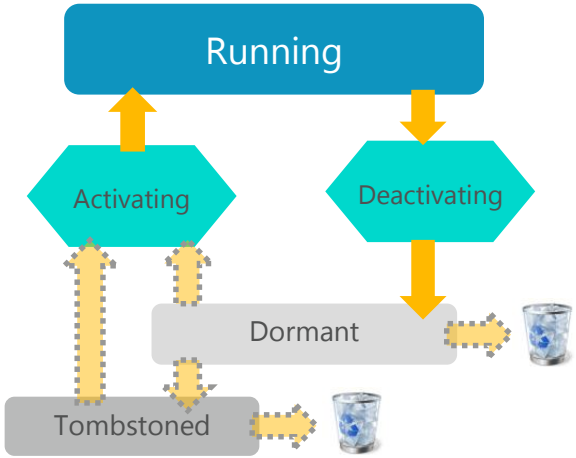
Windows Phone

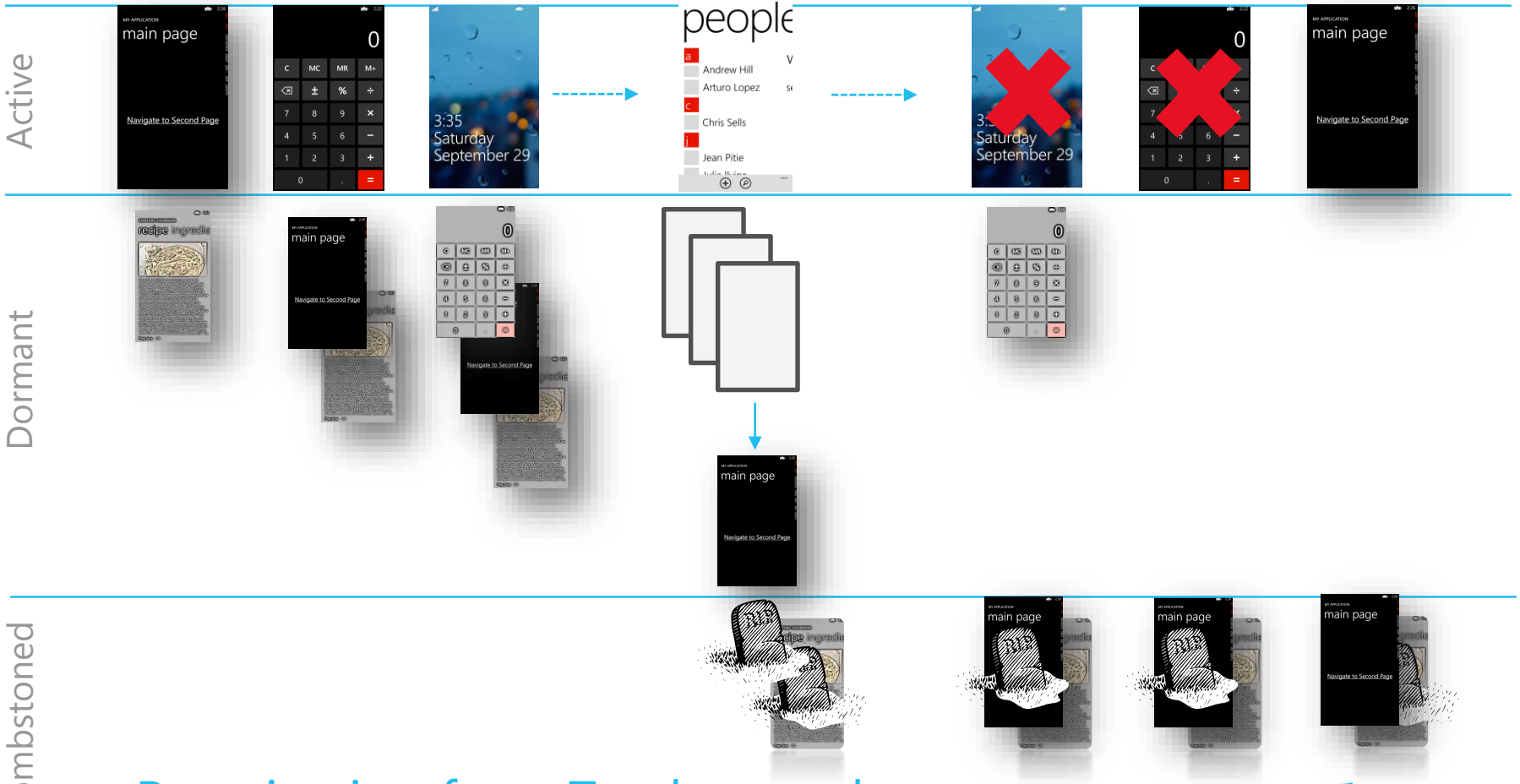
Resuming an Application – Fast Application Switching

- The user can resume a suspended application by ‘unwinding’ the application history stack by continually pressing the ‘Back’ key to close applications higher up the history stack, until the suspended application is reached
- The Back button also has a “long press” behaviour
- This allows the user to select the active application from the stack of interrupted ones
- Screenshots of all the applications in the stack are displayed and the user can select the one that they want
- The application is brought directly into the foreground if it is selected by the user
- On Windows Phone 8, the app history stack can hold up to 8 applications

From Dormant to Tombstoned

- An application will be held dormant in memory alongside other applications
  - If the operating system becomes short of memory it will discard the cached state of the oldest dormant application
  - This process is called “Tombstoning”
- The page navigation history and a special cache called the state dictionaries are maintained for a tombstoned application
- When a dormant application is resumed the application resumes running just where it left off
- When a tombstoned application is resumed, it restarts at the correct page but all application state has been lost – you must reload it
- An application can determine which state it is being activated from




Active
Dormant
Tombstoned
main page
Navigate to Second Page
people
Andrew Hill
Arturo Lopez
Chris Sells
Jean Pitie
3:35
Saturday
September 29
0
Reactivating from Tombstoned
Windows Phone

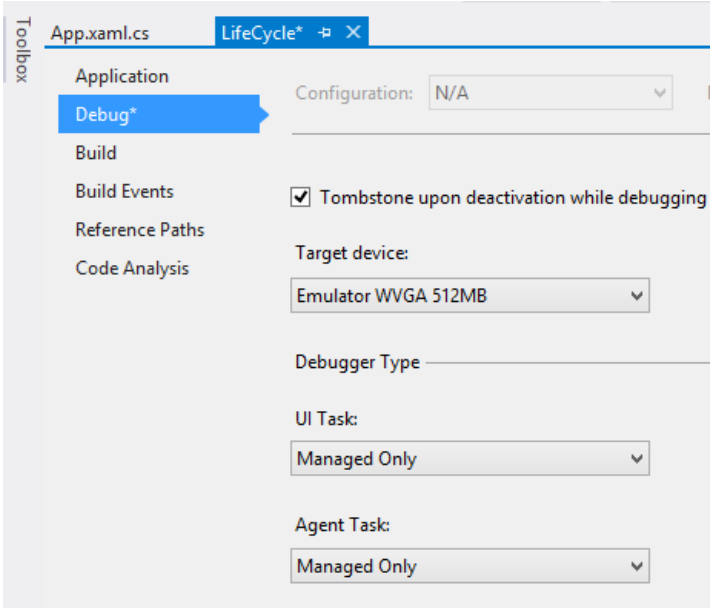
Resumed from Dormant or Tombstoned?

```
private void Application_Activated(object sender, ActivatedEventArgs e)
{
    if (e.IsApplicationInstancePreserved)
    {
        // Dormant - objects in memory intact
    }
    else
    {
        // Tombstoned - need to reload
    }
}
```

- You can also check a flag to see if a the application is being resumed from dormant or tombstoned state

Debugging Tombstoning

- You can force an application to be “tombstoned” (removed from memory) when it is made dormant by setting this in Visual Studio
- You should do this as part of your testing regime
- You can also use the Simulation Dashboard to show the lock screen on the emulator which will also cause your application to be made dormant



Demo 3: Dormant vs Tombstoned

States and Tombstones

- When an application is resumed from Dormant, it resumes exactly where it left off
  - All objects and their state is still in memory
  - You may need to run some logic to reset time-dependent code or networking calls
- When an application is resumed from Tombstoned, it resumes on the same page where it left off but all in-memory objects and the state of controls has been lost
  - Persistent data will have been restored, but what about transient data or "work in progress" at the time the app was suspended?
  - This is what the state dictionaries are for, to store transient data
  - State dictionaries are held by the system for suspended applications
- When a new instance of an application is launched the state dictionaries are empty
  - If there is a previous instance of the application that is suspended, standard behaviour is that the suspended instance – along with its state dictionaries - is discarded

The Application State dictionary

```
PhoneApplicationService.Current.State["Url"] = "www.robmiles.com";
```

- Transient data for a tombstoned application may be stored in the application state dictionary
- This can be set in the `Application_Deactivated` method and then restored to memory when the application is activated from tombstoned
- This means that the `Application_Deactivated` method has two things to do:
  - Store persistent data in case the application is never activated again
  - Optionally store transient data in the application state dictionary in case the user comes back to the application from a tombstoned state

Saving Transient Data on Deactivation

```
protected override void OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
{
   base.OnNavigatedFrom(e);
   if (e.NavigationMode != System.Windows.Navigation.NavigationMode.Back
     && e.NavigationMode != System.Windows.Navigation.NavigationMode.Forward)
   {
      // If we are exiting the page because we've navigated back or forward,
      // no need to save transient data, because this page is complete.
      // Otherwise, we're being deactivated, so save transient data
      // in case we get tombstoned
      this.State["incompleteEntry"] = this.logTextBox.Text;
   }
}
```

- Use the Page State dictionary to store transient data at page scope such as data the user has entered but not saved yet
- Page and Application State are string-value dictionaries held in memory by the system but which is not retained over reboots

Restoring Transient Data on Reactivation

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    // If the State dictionary contains our transient data,
    // we're being reactivated so restore the transient data
    if (this.State.ContainsKey("incompleteEntry"))
    {
        this.logTextBox.Text = (string)this.State["incompleteEntry"];
    }
}
```

- In OnNavigatedTo, if the State dictionary contains the value stored by OnNavigatedFrom, then you know that the page is displaying because the application is being reactivated

Demo 4: Using State Dictionaries

Running Under Lock Screen on Windows Phone

Windows Phone Idle Detection

- The Windows Phone operating system will detect when an application is idle
  - When the phone enters idle state the lock screen is engaged
  - The user can configure the timeout value, or even turn it off
- When an application is determined to be idle it will be Deactivated and then Activated when the user unlocks the phone again
- An application disable this behaviour, so that it is able to run underneath the lock screen

Windows Phone Idle Detection

- The Windows Phone operating system will detect when an application is idle
  - When the phone enters idle state the lock screen is engaged
  - The user can configure the timeout value, or even turn it off
- When an application is determined to be idle it will be Deactivated and then Activated when the user unlocks the phone again
- An application disable this behaviour, so that it is able to run underneath the lock screen

This is strong magic. It gives you the means to create a “battery flattener” which users will hate. Use it with care, read the guidance in the documentation and follow the checklists.

Disabling Idle Detection

```
PhoneApplicationService.Current.ApplicationIdleDetectionMode =
                                                IdleDetectionMode.Disabled;
```

- This statement disables idle detection mode for your application
- It will now continue running under the lock screen
- There will be no Activated or Deactivated messages when the phone locks because the application is timed out
- Note that your application will still be deactivated if the users presses the Start button
- Disabling idle detection is not a way that you can run two applications at once

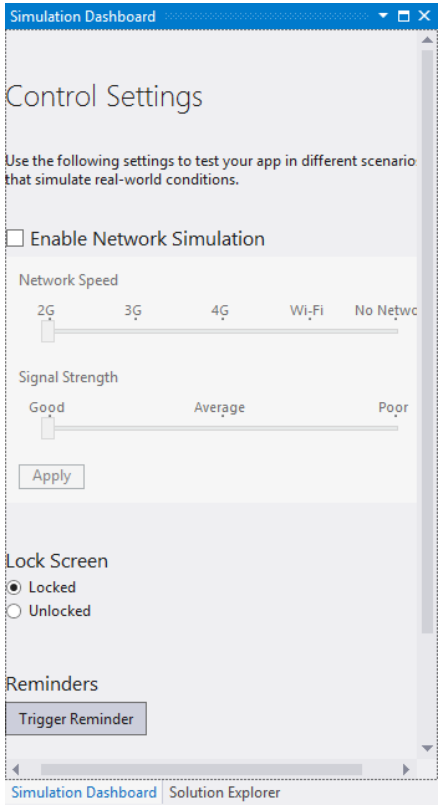
Detecting “Obscured” events

```
App.RootFrame.Obscured += RootFrame_Obscured;

...

void RootFrame_Obscured(object sender, ObscuredEventArgs e)
{
}
```

- Your application can connect to the Obscured event for the enclosing frame
- This will fire if something happens (Toast message, Lock Screen, Incoming call) that will obscure your frame
  - It also fires when the obscuring item is removed
  - You can use the ObscuredEventArgs value to detect the context of the call

Simulation Dashboard

- The Simulation Dashboard is one of the tools supplied with Visual Studio 2012
- It lets you simulate the environment of your application
- It also lets you lock and unlock the screen of the emulator phone
- This will cause Deactivated and Activated events to fire in your program



Fast Application Resume

Fast Application Resume

- By default a fresh instance of your application is always launched when the user starts a new copy from the programs page or a deep link in a secondary tile or reminder, or via new features such as File and Protocol associations or launching by speech commands
  - This forces all the classes and data to be reloaded and slows down activation
- Windows Phone 8 provides Fast Application Resume, which reactivates a dormant application if the user launches a new copy

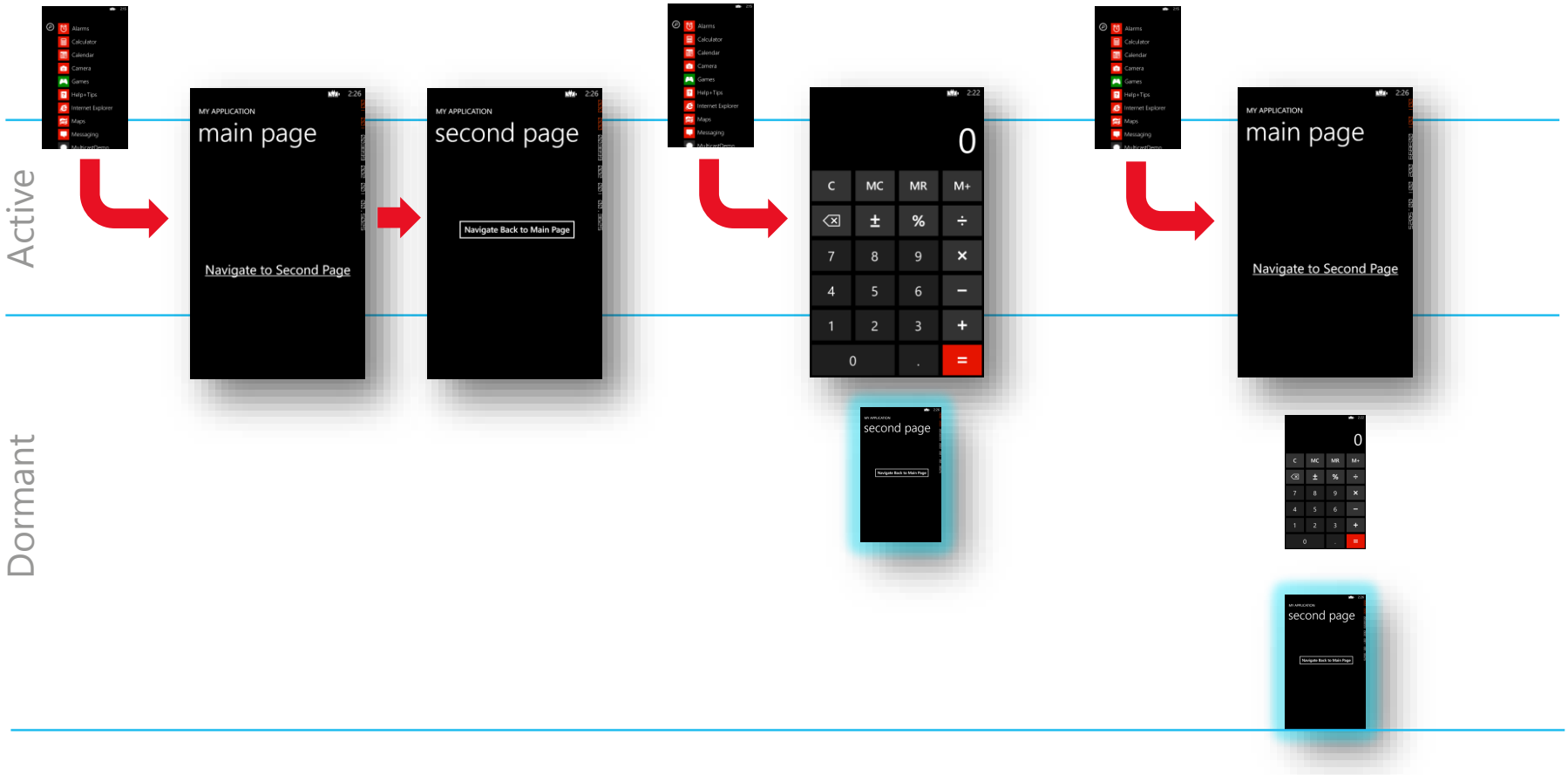
Demo 6: Fast Application Resume

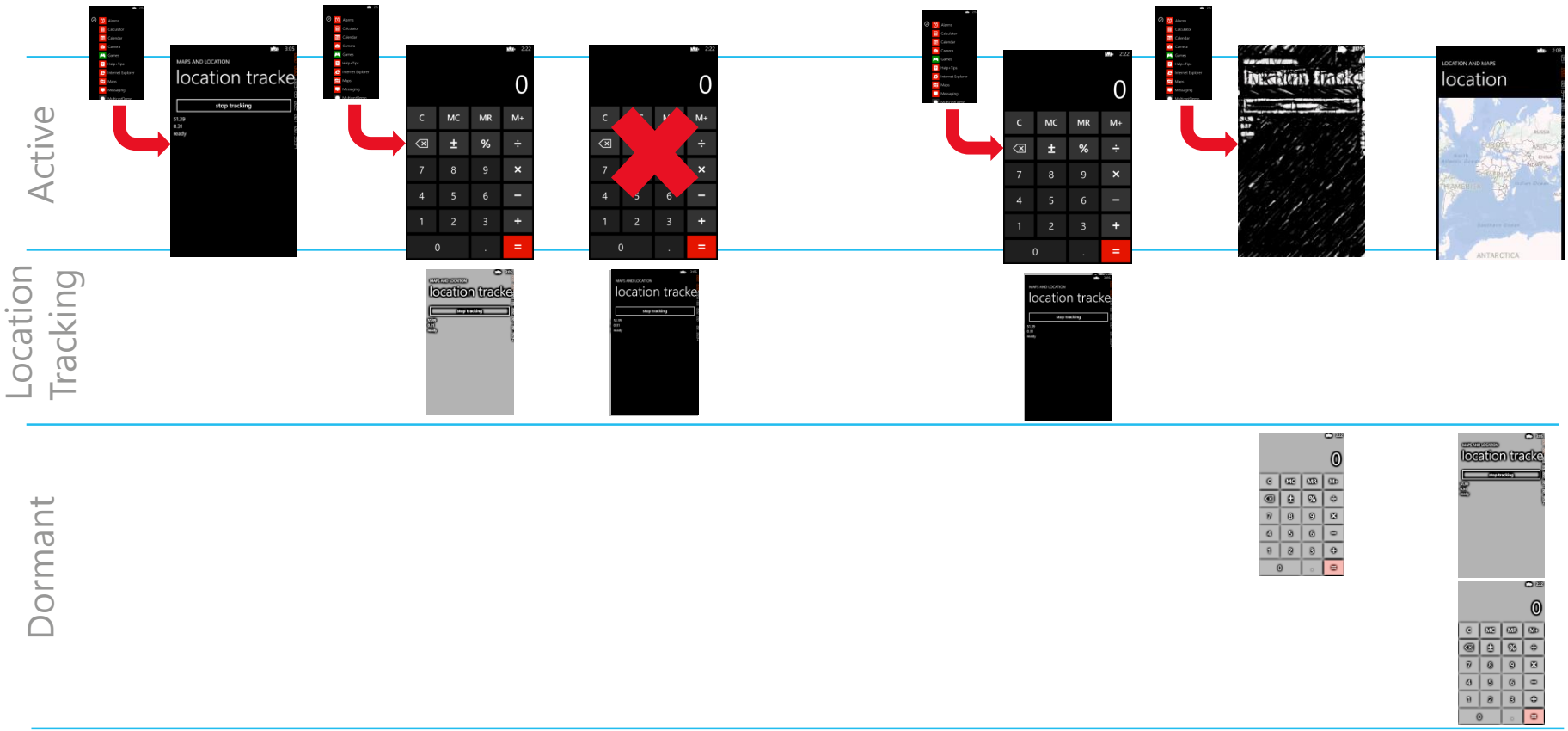
Enabling FAR in **Properties\WMAppManifest.xml**

```
<Tasks>
    <DefaultTask  Name ="_default" NavigationPage="MainPage.xaml"/>
</Tasks>
```

```
<Tasks>
    <DefaultTask  Name ="_default" NavigationPage="MainPage.xaml"
                  ActivationPolicy="Resume">
</Tasks>
```

- This is how FAR is selected
  - You have to edit the file by hand, there is no GUI for this

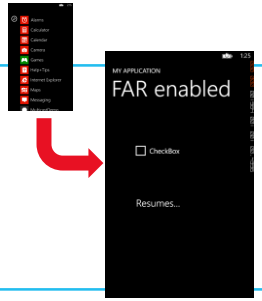

Active
Dormant
MY APPLICATION
main page
Navigate to Second Page
MY APPLICATION
second page
Navigate Back to Main Page
0
C
MC
MR
M+
7
8
9
4
5
6
1
2
3
Standard App Relaunch Behavior
Windows Phone



‘True’ Background Location Tracking Behavior

Windows Phone

Active



Location Tracking

Dormant

FAR-enabled App Behavior

Windows Phone

Sounds Great! So Why Don't I Use FAR With All My Apps?

- You need to be very careful on protecting the user experience with respect to page navigation
- Deep Link activation
  - On non FAR-enabled apps, if you launch an app to 'Page2', then 'Page2' is the only page in the page backstack, and pressing Back closes the app
  - For FAR-enabled apps, if you launch an app to 'Page2', but there was a suspended instance containing a page history backstack of 'MainPage', 'Page1', 'Page2', 'Page3' then the newly launched app will end up with a backstack of 'MainPage', 'Page1', 'Page2', 'Page3', 'Page2'
- What should the user experience be here?

App List/Primary Tile Behavior of FAR-enabled Apps

- Similarly, the user expectation when launching an app from an application tile on the Start Screen or from the Apps List is that the app will launch at the home page of the app
  - This is what will happen for non FAR-enabled apps
  - This is what will happen even for FAR-enabled apps if there is no suspended instance at the time the app is launched
- Since your app can now resume, what should the user experience be?
  - Should you always launch on the home page to match previous behaviour of Windows Phone apps?
  - Or should you resume at the page the user was last on?
  - If you resume at the last page the user was on, think about how the user will navigate to the home page; good idea to put a home icon on the app bar of the page the user lands on

Detecting Resume in a FAR-enabled App

- When an app enabled for Fast App Resume (aka FAR) is relaunched from the App List or a Deep Link:
  - Page on the top of the backstack gets navigated from with **NavigationMode.Reset**
  - Next, the page in the new launch url will be navigated to with **NavigationMode.New**
  - You can decide whether to unload the page backstack or not, or maybe cancel the navigation to the new page

Clearing Previously Launched Pages on Fast App Resume

Add Logic to App.Xaml.cs to Check for Reset Navigation (Behavior Already Implemented in Project Templates)

```
private void InitializePhoneApplication()
{
    ...
    // Handle reset requests for clearing the backstack
    RootFrame.Navigated += CheckForResetNavigation;

    ...
}

private void CheckForResetNavigation(object sender, NavigationEventArgs e)
{
    // If the app has received a 'reset' navigation, then we need to check
    // on the next navigation to see if the page stack should be reset
    if (e.NavigationMode == NavigationMode.Reset)
        RootFrame.Navigated += ClearBackStackAfterReset;
}
```

Clearing Previously Launched Pages on Fast App Resume

Add Logic to App.Xaml.cs to Check for Reset Navigation (Behavior Already Implemented in Project Templates)

```
private void InitializePhoneApplication()
{
    ...
    // Handle reset requests for clearing the backstack
    RootFrame.Navigated += CheckForResetNavigation;

    ...
}

private void CheckForResetNavigation(object sender, NavigationEventArgs e)
{
    // If the app has received a 'reset' navigation, then we need to check
    // on the next navigation to see if the page stack should be reset
    if (e.NavigationMode == NavigationMode.Reset)
        RootFrame.Navigated += ClearBackStackAfterReset;
}

private void ClearBackStackAfterReset(object sender, NavigationEventArgs e)
```

Review

- Programs can be Launched (from new) or Activated (from Dormant or Tombstoned)
- The operating system provides a state storage object and navigates to the correct page when activating an application that has been running, but you have to repopulate forms
- Programs can run behind the Lock Screen by disabling Idle Timeout – but this is potentially dangerous (and apps must deal with the Obscured event in this situation)
- The system maintains a Back Stack for application navigation. This must be managed to present the best user experience when applications are started in different ways
- Applications are normally launched anew each time they are started from the Programs page, but it is now possible to restart a suspended instance using Fast Application Resume
- YOU MUST PLAN YOUR LAUNCHING AND BACK STACK BEHAVIOUR FROM THE START

Windows Phone

The information herein is for informational purposes only an represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

Microsoft