Event Handling, Application Framework

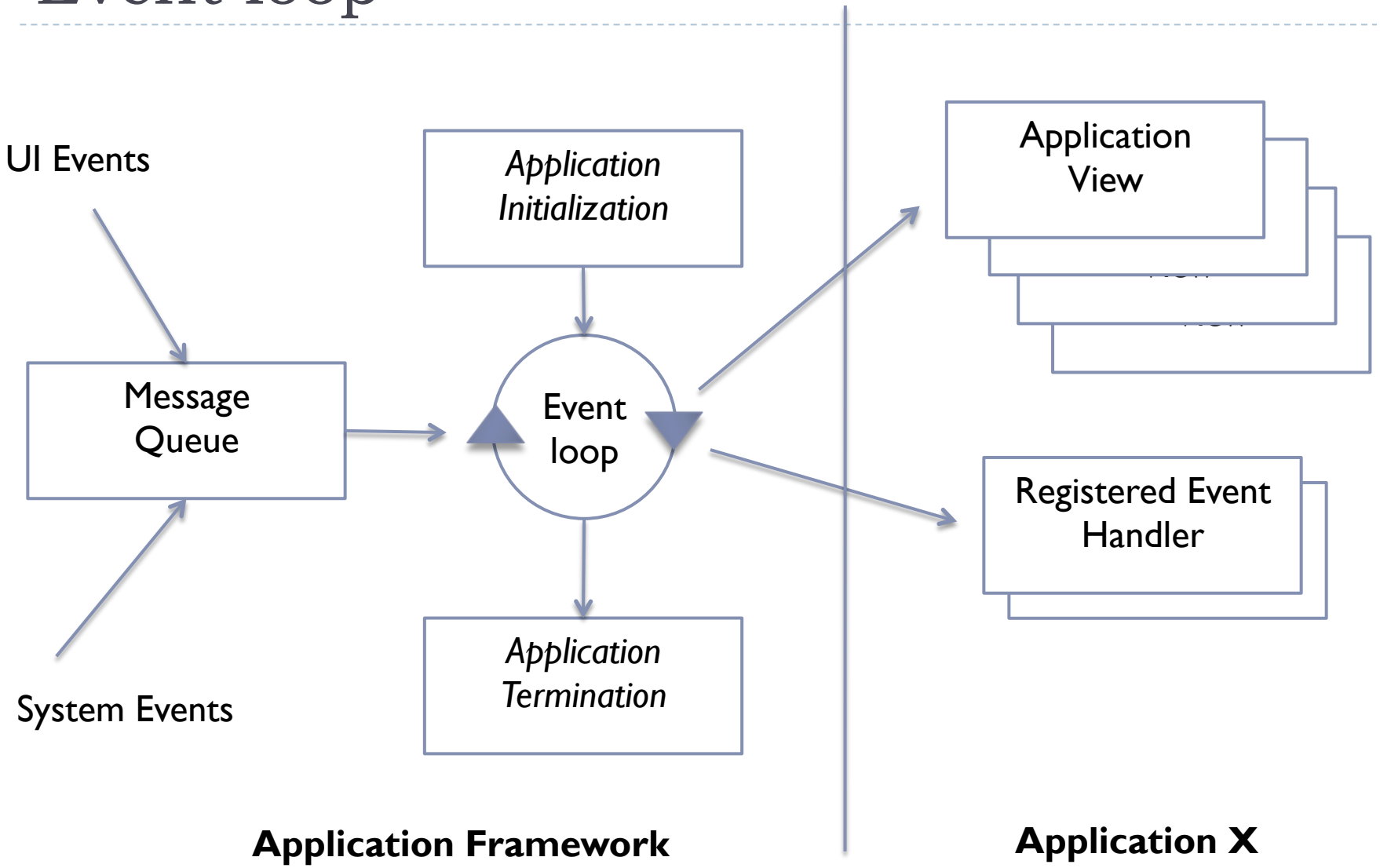
Embedded Software Development Environments* Spring 2013

Antti Juustila, Henrik Hedberg
Department of Information Processing Science, University of Oulu, Finland

Event handling basics

- Applications conform to an Application framework
  - Apps receive events from the framework and handle them
  - Defines the architecture of the app – often MVC
- Framework and system services
  - Take care of interacting with the physical devices
    - Touch screen, keyboard, buttons, system clock, network, sensors…
  - Route events to active application
    - UI events (touch, keyboard) to active view
    - Other system events to initiator / registered listener
- Applications / app elements register to events
  - Events put on the app event queue
  - Framework reads events from the queue, pass forward
  - Application implements callback methods, which are called by the framework / system as events happen
  - Usually single threaded

Event loop


UI Events
Application Initialization
Application View
Message Queue
Event loop
Registered Event Handler
Application Termination
System Events
Application Framework
Application X

Implications to Apps

- No (visible) main()
  - Study the app framework, implement necessary classes
  - Study the app/view initialization sequence, implement necessary initialization callbacks
  - Study necessary events to handle & implement event handlers as needed
  - Study which events are available and if you want your app to register to those events → handle the events
  - Do not try to pass app framework, cooperate with it
  - ”Any event may arrive to application at any time” (almost)
- Single threaded
  - Event handlers must not take longer than ~20 ms to complete!
    - Long running tasks must be executed in a separate thread
  - Worker threads must not try to access UI objects directly
    - Pass data and events to your classes in the UI thread, use locks/ semaphores

Android App Framework

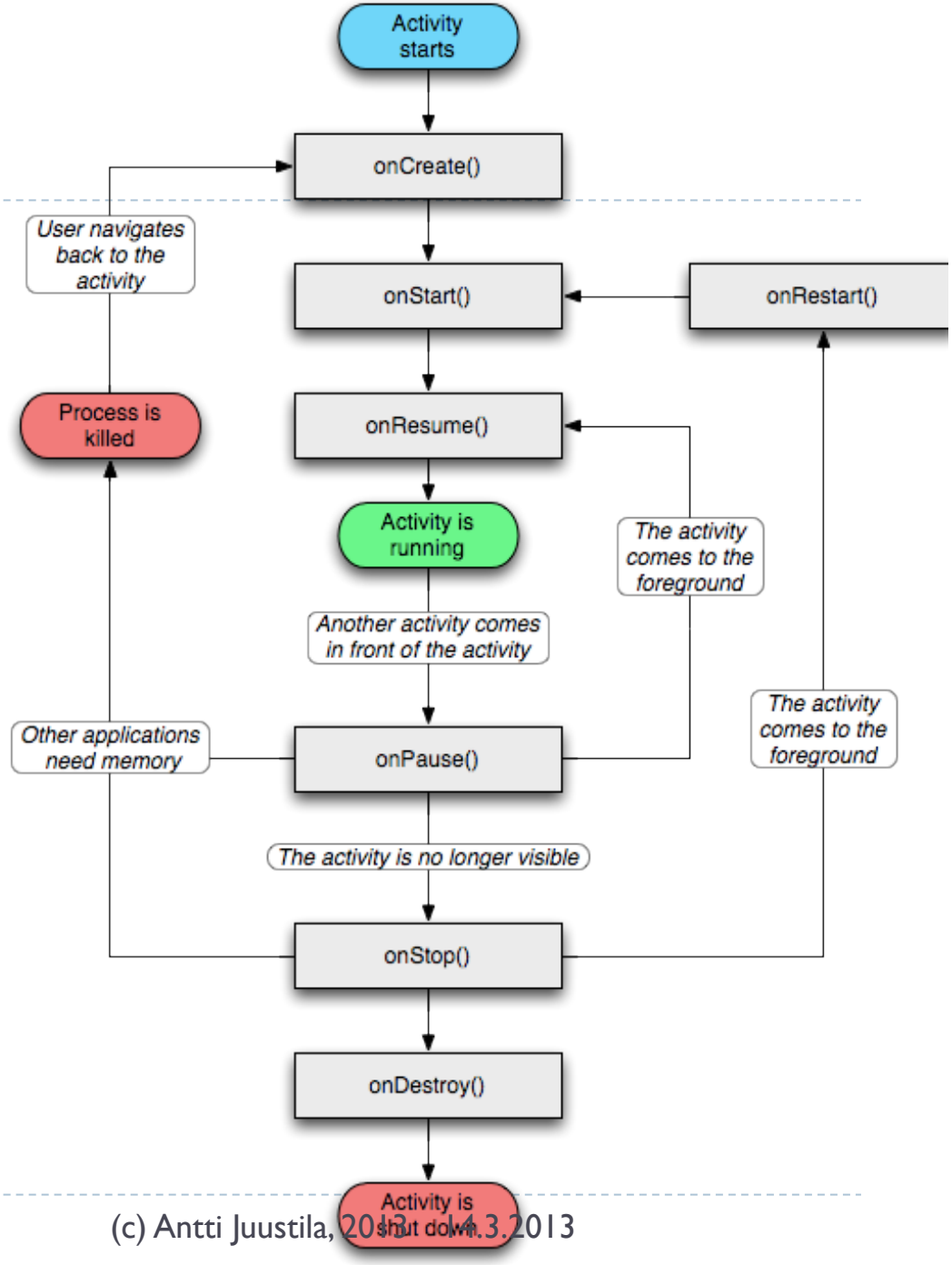
- Basics
  - Android is based on Linux
  - Each app has own process and a Linux user ID
  - Permissions based on user ID
  - Each process has its own virtual machine
- Basic application components
  - **Activity** – a screen with a user interface
  - **Service** – performing long-running background operations or work for remote processes
  - **Content provider** – manages a set of shared data
  - **Broadcast receiver** – respond to system wide broadcasted announcements
  - **Intent** – asynchronous message object that activates the forementioned components

Activities

- A specific, separate functionality from the user's point of view
- User's task often performed with several activities
  - Activities can be in different applications → and in different processes
- Activity is
  - A class in the API, which…
  - …encapsulates an operation (send mail, set up game,…),
  - …which runs in the application's process, and…
  - …usually has a view (window, UI), and
  - …an execution context

http://developer.android.com/guide/topics/fundamentals/activities.html

Activity lifecycle



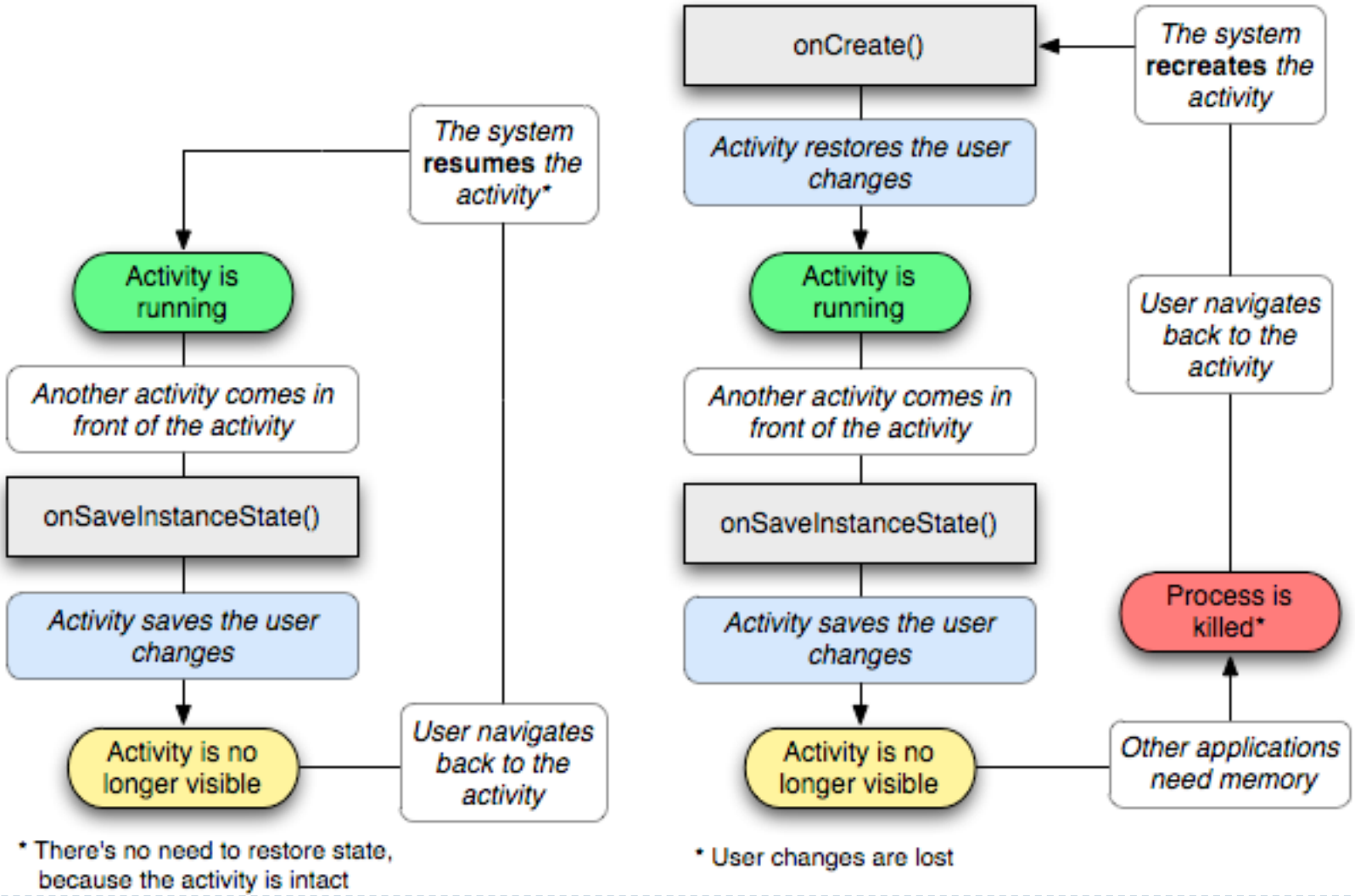
Activity callbacks

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();   // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();  // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();  // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();  // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();  // The activity is about to be destroyed.
    }
}
```

Flow of events

- User performs a task, switching from an activity to another
  - One or several activities can be in different applications
- Need to save the state of the activity
  - Especially if system shuts the application down
  - State can then be restored
- System maintains a Back Stack
  - A stack data structure
  - Used in navigating back to previous activities in the task flow
  - Can be configured from default behaviour, if you have a good reason (and test the usability)

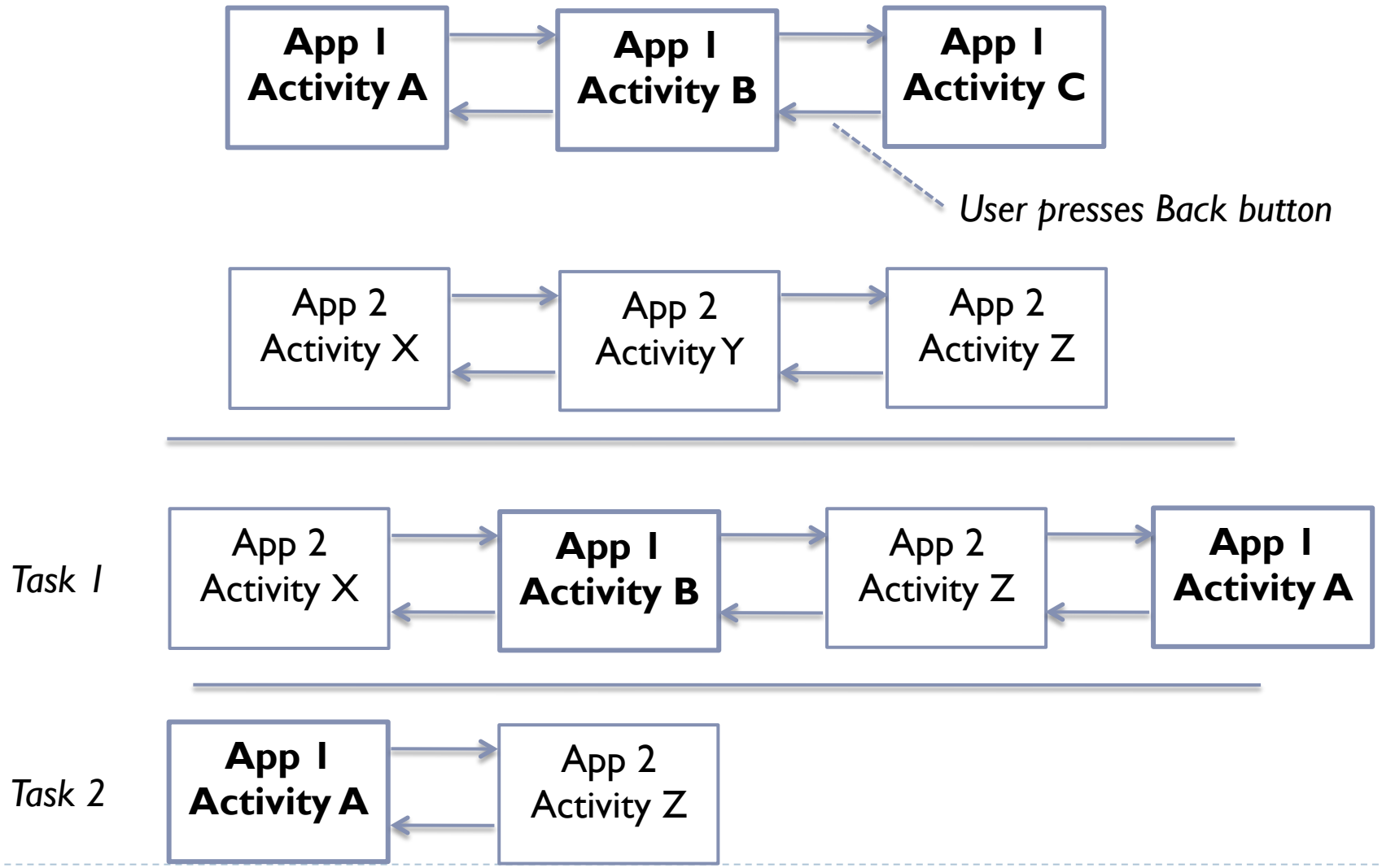
Flow of events in state store/restore



onPause & onSaveInstanceState

- onPause
  - Called as part of the normal life cycle of an app
- onSaveInstanceState
  - Called only by the system, when it thinks it is necessary to save the state before destroying the activity
    - As part of moving through the Back Stack, for example
  - If called, this method will occur before onStop()
  - There are no guarantees about whether it will occur before or after onPause().

Activities, Tasks and Back Stack



Starting Activities with Intents

```
// Activity in your own application
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

```
// Activity in- or outside your application, using data from your application
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

http://developer.android.com/guide/topics/intents/intents-filters.html

Retrieving Data Passed to Activity

```
public class ImageViewActivity extends Activity {

    private Bitmap image;

    public void onCreate(Bundle bundle) {
    super.onCreate(bundle);

    setContentView(R.layout.view_image);

    String imagePath = getIntent().getExtras().getString(TellAStory.MEDIA_PATH);
…
```

Starting an activity, expecting result

```
private void pickContact() {
    // Create an intent to "pick" a contact, as defined by the content provider URI
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
    startActivityForResult(intent, PICK_CONTACT_REQUEST);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // If the request went well (OK) and the request was PICK_CONTACT_REQUEST
    if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {
        // Perform a query to the contact's content provider for the contact's name
        Cursor cursor = getContentResolver().query(data.getData(),
        new String[] {Contacts.DISPLAY_NAME}, null, null, null);
        if (cursor.moveToFirst()) { // True if the cursor is not empty
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);
            String name = cursor.getString(columnIndex);
            // Do something with the selected contact's name...
        }
    }
}
```

Input Events

- In the focus – the View class
- Can subclass it and override the event handling methods
  - Buttons and other controls are also views
  - Not pratical to subclass all ui object classes when using the controls ”as they are”
    - Extending and/or changing the functionality of the control is a different issue
- Use Event Listeners
  - And the callback methods

UI Event Listeners

View.OnClickListener.onClick() – User either touches the item, or focuses upon the item and presses the suitable "enter" key or the trackball.

View.OnLongClickListener. onLongClick() – User either touches and holds the item, or focuses upon the item and presses and holds the suitable "enter" key or presses and holds down on the trackball.

View.OnFocusChangeListener.onFocusChange() – User navigates onto or away from the item, using the navigation-keys or trackball.

View.OnKeyListener.onKey() – User is focused on the item and presses or releases a key on the device.

View.OnTouchListener.onTouch() – User performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

View.OnCreateContextMenuListener.onCreateContextMenu() – A Context Menu is being built (as the result of a sustained "long click").

Using Event Listeners – Typical Example

```
public class ExampleActivity extends Activity implements OnClickListener {
    Button checkButton;
    protected void onCreate(Bundle savedValues) {
        ...
        checkButton = (Button)findViewById(R.id.corky);
        checkButton.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
    // do something when the button (v) is clicked
      if (v == checkButton) { …  // Or: if (view.getId() == R.id.corky)
      }
    }
    ...
}
```

Event listener – anonymous handler

```
EditText e = (EditText) findViewById(R.id.value_numeric_editor);
e.setOnEditorActionListener(new OnEditorActionListener() {
   @Override
   public boolean onEditorAction(TextView arg0, int arg1, KeyEvent arg2) {
      if (arg1 == EditorInfo.IME_ACTION_DONE || arg1 == EditorInfo.IME_NULL) {
         String newText = arg0.getText().toString();
         checkAndSetNumericValue(newText);
      }
      return true;
   }});
```

Event Listeners – Return Values

- Some callbacks return a value, which must be handled
  - *onLongClick()* – Return true to indicate that you have handled the event and it should stop here. Return false if you have not handled it and/or the event should continue to any other on-click listeners.
  - *onKey()* – Return true to indicate that you have handled the event and it should stop here; return false if you have not handled it and/or the event should continue to any other on-key listeners.
  - *onTouch()* – This event can have multiple actions that follow each other. If you return false when the down action event is received, you indicate that you have not consumed the event and are also not interested in subsequent actions from this event. Thus, you will not be called for any other actions within the event, such as a finger gesture, or the eventual up action event.

Focus

- Some events are handled to the view who has the focus
  - Usually, keyboard events are like this
- Focus related methods
  - isFocusable() – can the view take focus
  - setFocusable() – change if can take focus or not
  - …InTouchMode(), special for touch mode behaviour
- Focus movement
  - By touching or by navigation key(s)
  - Framework has an algorithm which calculates where focus should go next

Focus

- Changing the way focus movement behaves

```
<LinearLayout
    android:orientation="vertical"
    ... >
  <Button android:id="@+id/top"
          android:nextFocusUp="@+id/bottom"
          ... />
  <Button android:id="@+id/bottom"
          android:nextFocusDown="@+id/top"
          ... />
</LinearLayout>
```

Focus

- To request a particular View to take focus
  - call requestFocus()
- To listen for focus events
  - Get notified when a View receives or looses focus
  - use onFocusChange()
  - Common in situations where
    - Input has to be validated or
    - State / contents of a control influence on the state of other control(s)