JavaCro'13

Android development
Best Practices

Ivan Gavran, Calyx d.o.o.

Overview

- Android 101
- Adaptive UI
- Offline support
- Networking tips
- Handling bitmaps efficiently
- Conclusion

Android 101

- Application Components
  - Activities/Fragments
  - Service
  - Content Providers
  - Broadcast Receivers
- Component Activation
  - Intents
  - Intent Filters

Android 101

- The Manifest File
  - Component declaration
  - Component capabilities
  - Application Requirements
- Application Resources
  - Images, audios, videos
  - Layouts
  - Strings, dimensions...

The Good Parts

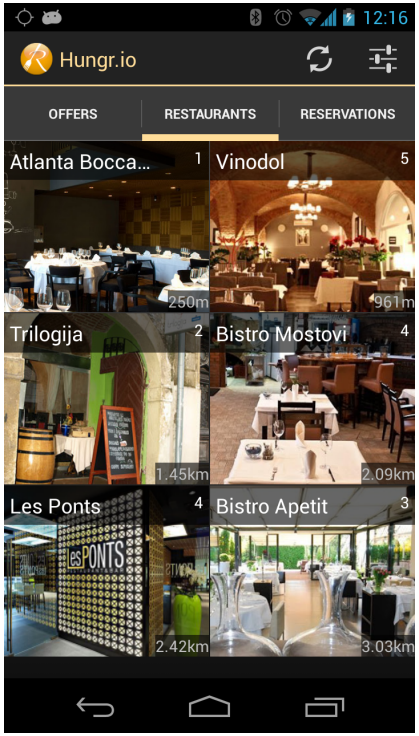
- Separation of concerns
  - Sources
  - Resources
  - App configuration
- Inter-process communication
  - Intents, Content Providers
- Built in services
  - Location, Notification, Connectivity...

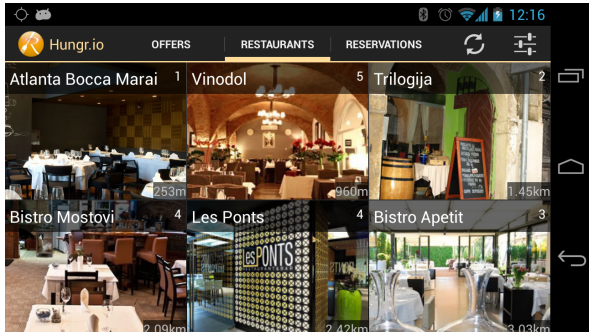
Case study

- Android app backed by a REST Service
- Requirements:
  - Adaptive UI
  - Offline support
  - Efficiency

Lesson 1:
Define adaptive UI

Adaptive Actionbar



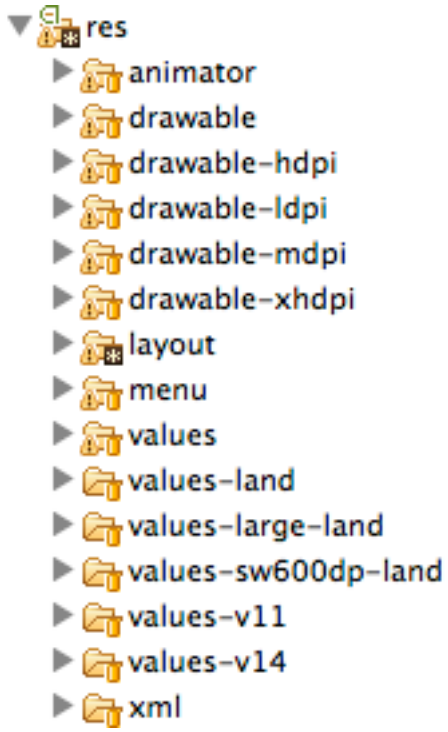


Adaptive GridView

Adaptive Actionbar

Adaptive Layout

Power of resources



res/values/layouts.xml

```
<resources>
  <item name="restaurant_layout" type="layout">
    @layout/restaurant_details
  </item>
  <bool name="has_two_panes">false</bool>
</resources>
```

res/values-land/layouts.xml

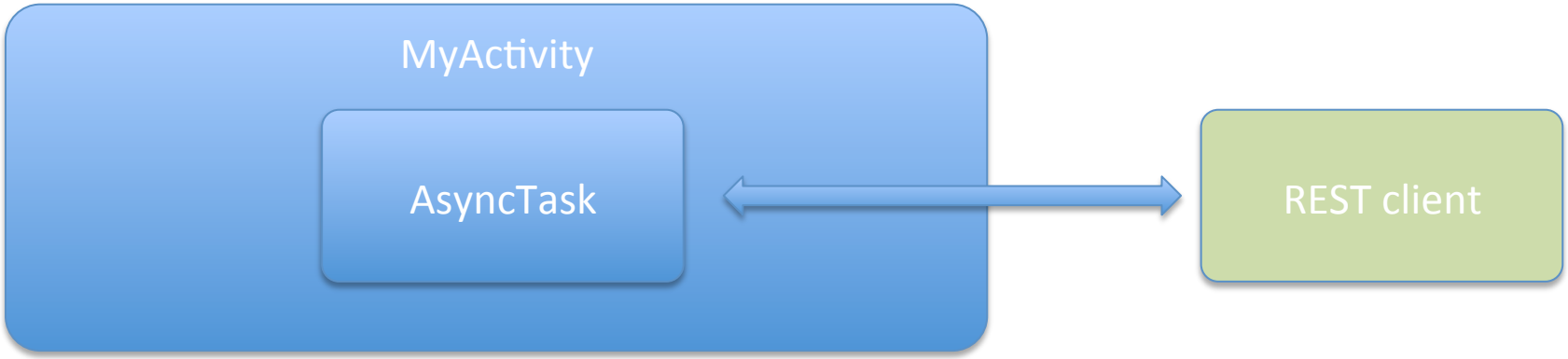
```
<resources>
  <item name="restaurant_layout" type="layout">
    @layout/restaurant_details_land
  </item>
  <bool name="has_two_panes">true</bool>
</resources>
```

Activity initialization

```
public class MyActivity extends FragmentActivity {

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //Set content view based on layout defined
    // for current screen configuration
    setContentView(R.layout.activity_restaurant_layout);

    // or read any other value...
    if(getResources().getBoolean(R.bool.has_two_panes)){
        //... we are in landscape
    }
}
```

Lesson 2:
Use Loaders for loading data in an activity or fragment

The naive approach

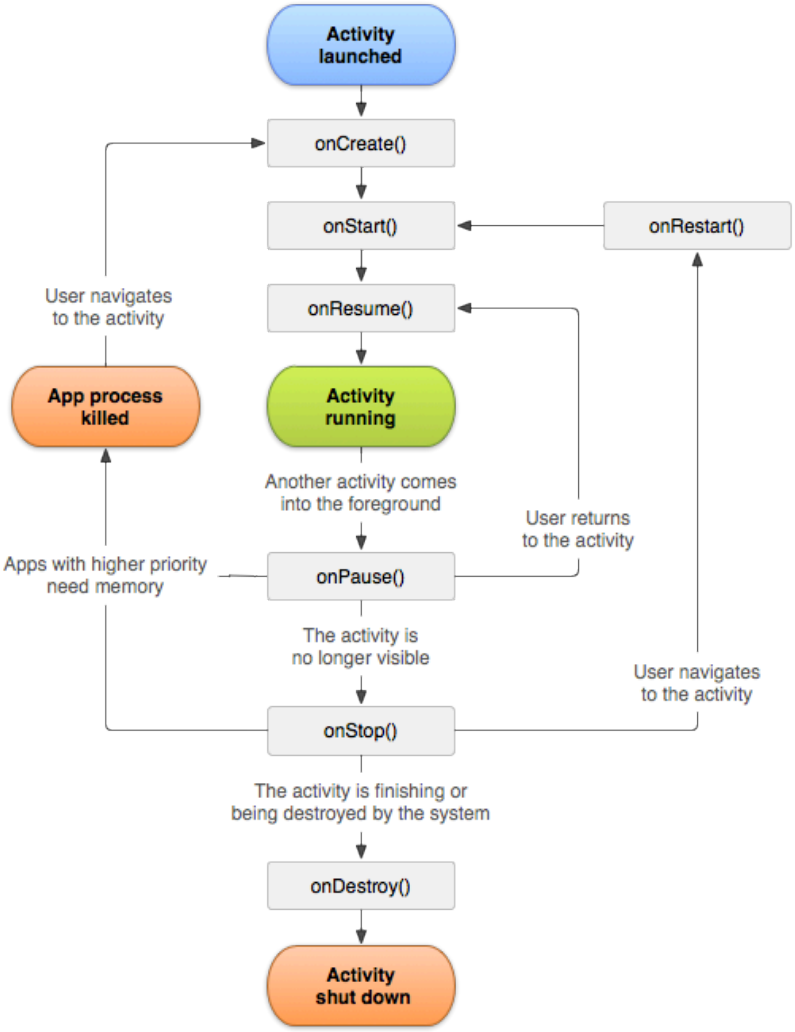


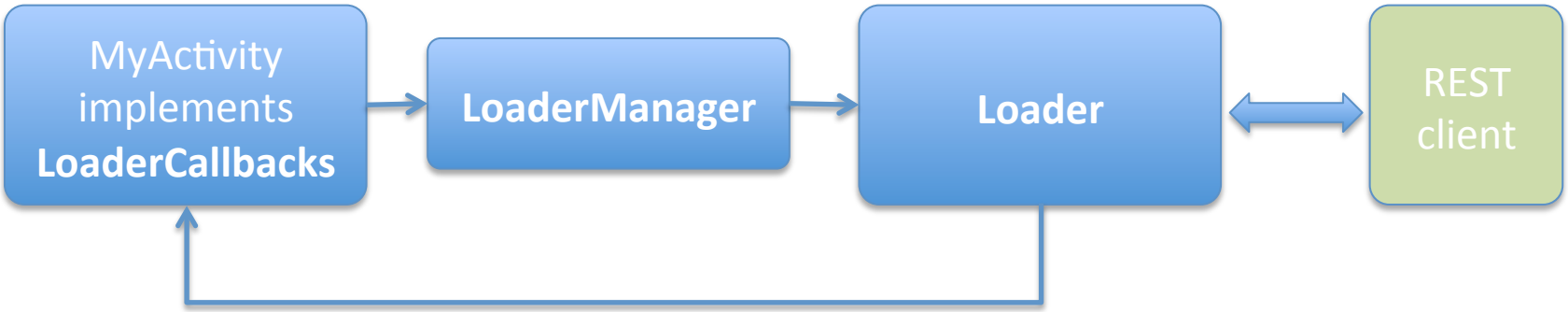

```
class MyActivity extends Activity {

  @Override
  public void onCreate(...) {
    new MyAsyncTask().execute(...);
  }

}
```

```
class MyAsyncTask extends
    AsyncTask<Pars, Progress, Res>{

  void onPreExecute(Pars...){
  // executed on UI thread
  }
  Res doInBackground(){
  // executed on background thread
  }
  void onPostExecute(Res){
  // executed on UI thread
  }
}
```

Problems with AsyncTasks

- Activity life cycle
- Memory Leaks





Loader API

```
android.app.LoaderManager

android.app.LoaderManager.LoaderCallbacks

android.content.Loader<D>
    ↳  android.content.AsyncTaskLoader<D>
          ↳  android.content.CursorLoader
```

Loader initialization

```
public class MyActivity extends FragmentActivity
implements LoaderManager.LoaderCallbacks<DomainModel> {

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //... activity setup code

    getLoaderManager().initLoader (0, null, this);
}
```

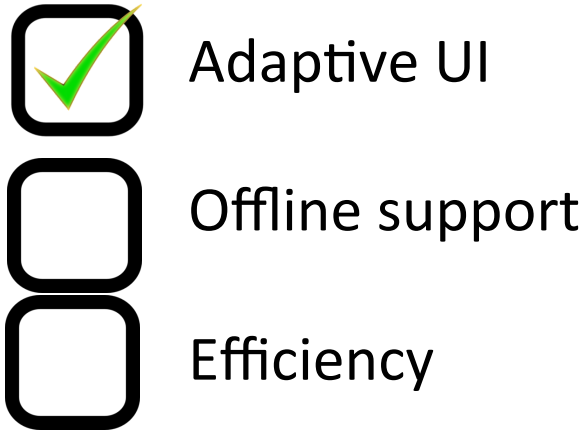
LoaderCallbacks interface implementation

```
public class MyActivity extends FragmentActivity
implements LoaderManager.LoaderCallbacks<DomainModel> {

//....
public Loader<DomainModel> onCreateLoader(int id, Bundle args) {
    return new MyCursorLoader(this);
}

public void onLoadFinished(Loader<DomainModel> loader,
    DomainModel data) {

    mAdapter.swapCursor(data);
}

public void onLoaderReset(Loader<DomainModel> loader) {
    mAdapter.swapCursor(null);
}
```

Loader implementation

```
public class MyLoader extends AsyncTaskLoader<DomainModel> {

    public MyCursorLoader (Context context) {
        super(context);
    }
    @Override
    protected void onStartLoading() {
        forceLoad();
    }
    @Override
    public DomainModel loadInBackground() {
        return RestClient.get().getDomainModel();
    }
    @Override
    protected void onReset() {
        super.onReset();
    }
```

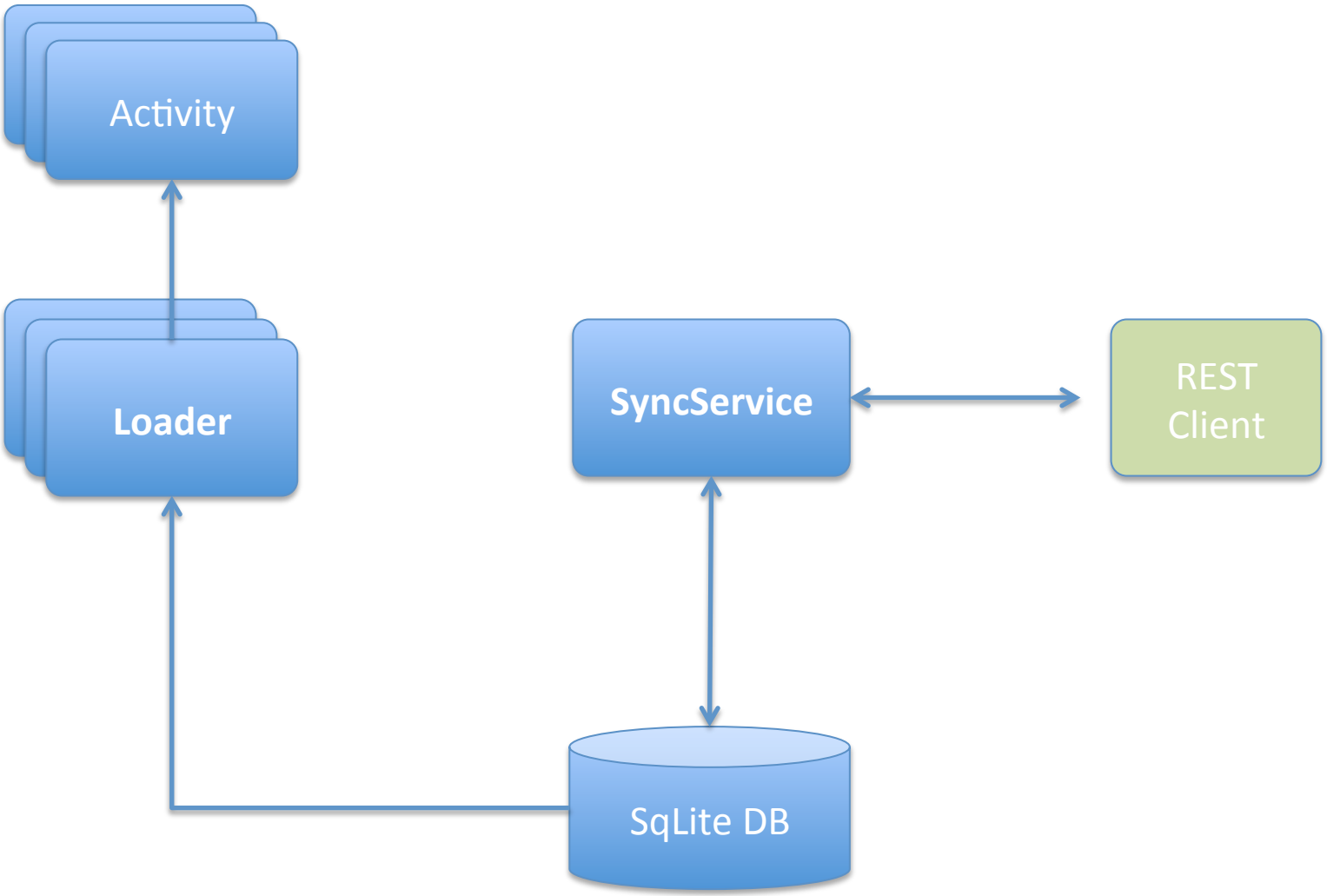
Goals



- Adaptive UI
- Offline support
- Efficiency

Lesson 3:
Local Database and Sync Service

Application architecture



Synchronization IntentService

```
public class SyncService extends IntentService {

    public SyncService() {
        super("DataSyncService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        String action = intent.getAction();

        // get optional sync parameters from the intent

        doSync(...);
    }
```

How to refresh data in the application

- Start sync service
  - Just start service from any part of the application
- Restart Loaders
  - implement BroadcastReceiver in each Loader
- Refresh UI
  - triggered by Loaders!

SyncIntentReciever

```
public class MyCursorLoader extends AsyncTaskLoader<Cursor> {

    //...

    private static class SyncIntentReceiver extends BroadcastReceiver {
        final MyReservationsCursorLoader mLoader;
        public SyncIntentReceiver(MyReservationsCursorLoader loader) {
            mLoader = loader;
            IntentFilter filter = new IntentFilter(
                    MyAppConstants.ACTION_SYNC_FINISHED);
            mLoader.getContext().registerReceiver(this, filter);
        }

        @Override
        public void onReceive(Context context, Intent intent) {
            mLoader.onContentChanged();
        }
    }
```

SyncIntentReceiver

```
public class MyCursorLoader extends AsyncTaskLoader<Cursor> {
    private SyncIntentReceiver mObserver;
    @Override
    protected void onStartLoading() {
        if (mObserver == null) {
            mObserver = new SyncIntentReceiver(this);
        }
        forceLoad();
    }
    @Override
    protected void onReset() {
        super.onReset();
        if (mObserver != null) {
            getContext().unregisterReceiver(mObserver);
            mObserver = null;
        }
    }
    private static class SyncIntentReceiver extends BroadcastReceiver {
        //...
    }
}
```

Result

- Loose coupling of application and synchronization logic
- Easy implementation of application specific sync strategies
  - Update triggered by app user
  - Updates based on changes on the application server
  - Location based updates
  - ... *just start the Sync Service*...

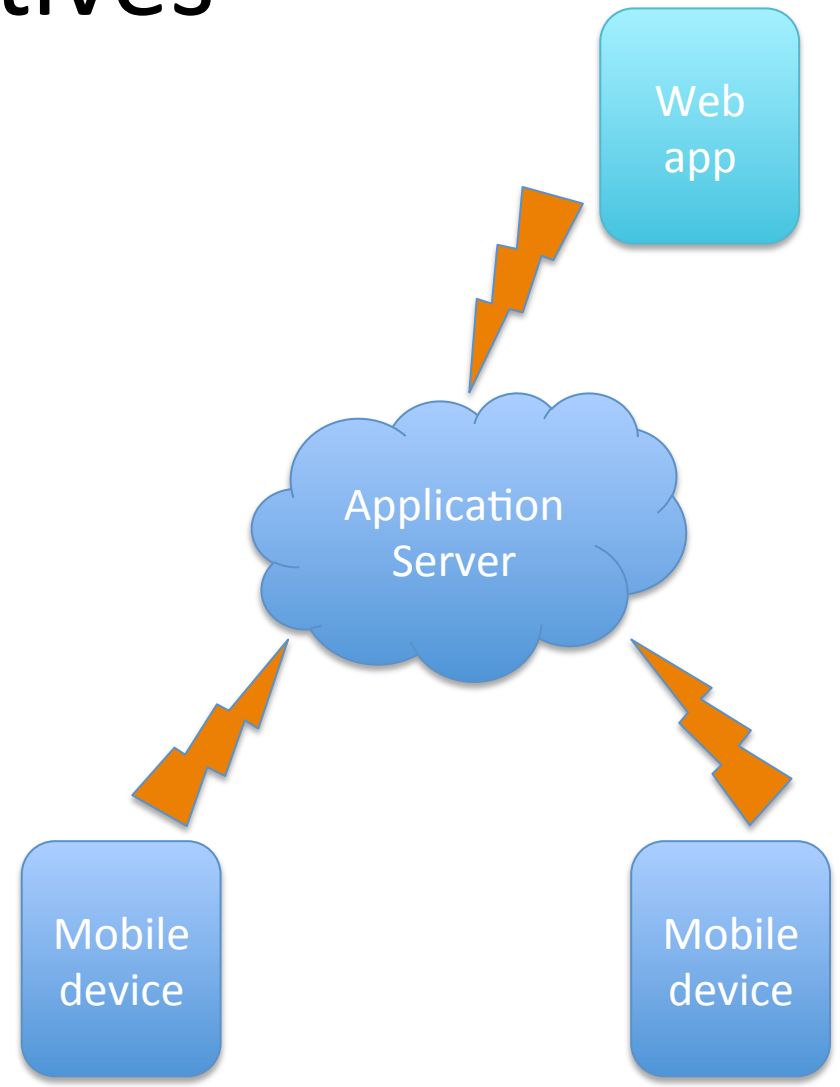
Goals

- Adaptive UI
- Offline support
- Efficiency

Lesson 4:
Keep mobile data in sync with the application server

Objectives

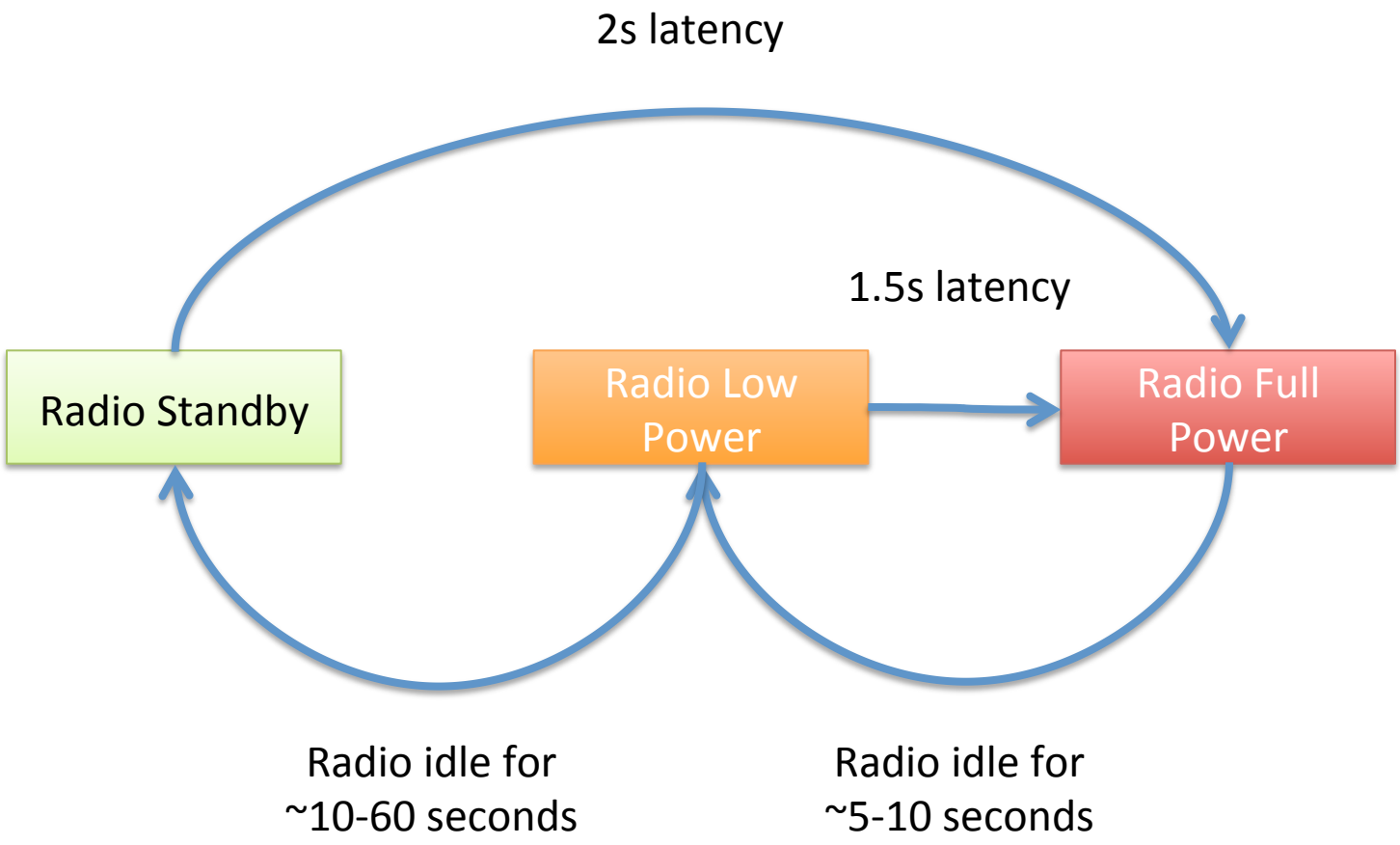
- Be efficient:
  - Data traffic costs
  - Battery life
- Be invisible
  - Don't keep user wait
  - Give users what they want before they have to ask for it



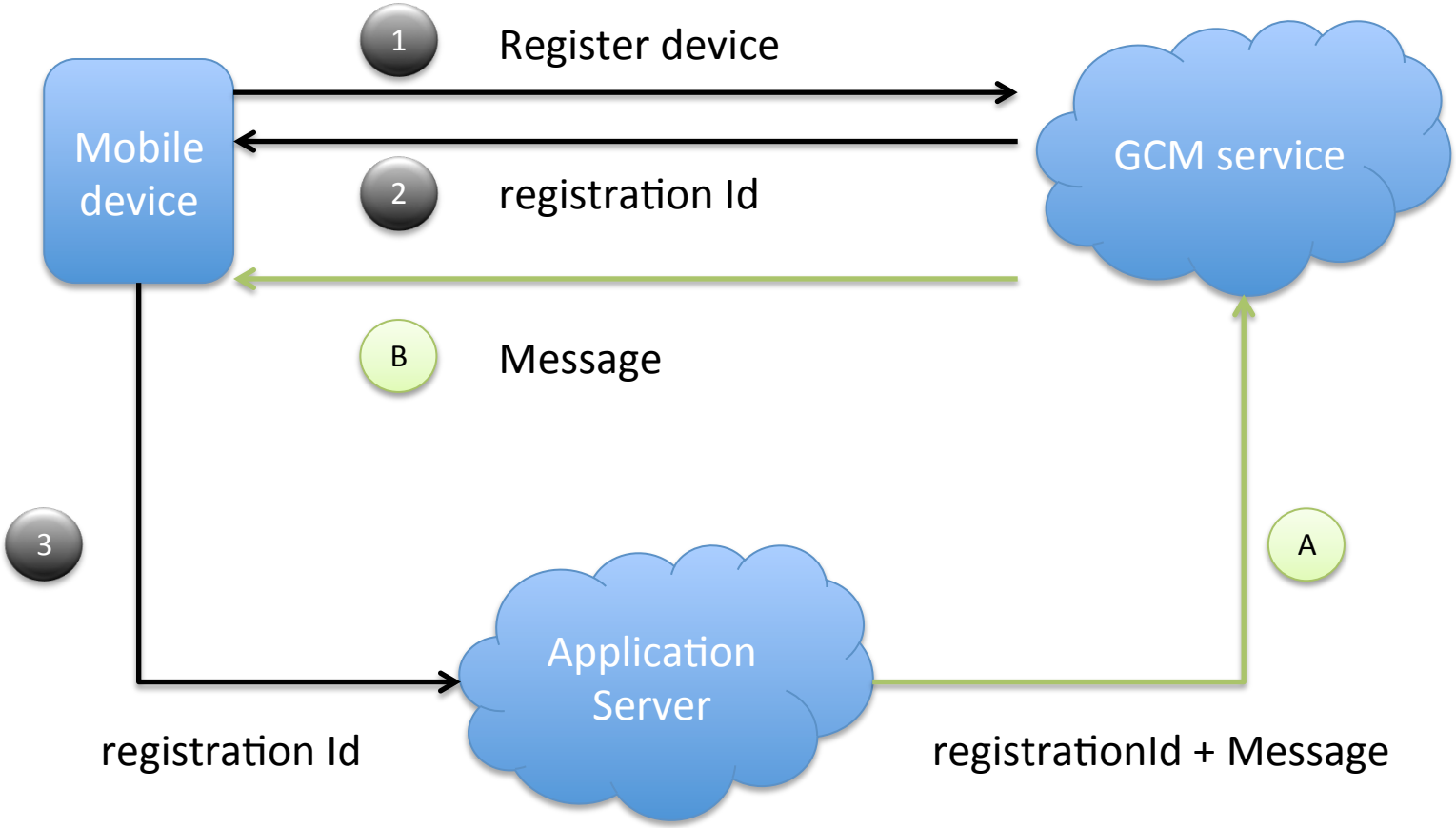
Networking tips

- Detect active network type
  - WIFI vs. cellular data
  - Mobile radio state machine
- Networking over 3G networks
  - Avoid periodical data transfer – don't pull for updates
  - Use greedy loading but don't be too greedy
  - Prefetching, bundling
- Google Cloud Messaging
  - Push data to mobile device

The mobile radio state machine



Google Cloud Messaging



Lesson 5:
Displaying Bitmaps Efficiently

Resize your images on the server.

Load Large Bitmaps Efficiently

Image resolution	Bitmap size (ARGB_8888)
2048*1536	12 MB
512*384	0.75 MB
128*96	0.05

Decode Large Bitmaps

```
public static Bitmap decodeSampledBitmap(InputStream is, int reqWidth, int
reqHeight) {

    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeStream(is, null, options);

    // Calculate inSampleSize
    options.inSampleSize = calculateInSampleSize(
        options.outWidth, options.outHeight, reqWidth, reqHeight);

    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    return BitmapFactory.decodeStream(is, null, options);
}
```

Process Bitmaps off the UI thread.

Processing Bitmaps Off the UI Thread

```
class BitmapLoaderTask extends AsyncTask<String, Void, Bitmap> {
    private final WeakReference<ImageView> imageViewReference;
    public BitmapWorkerTask(ImageView imageView) {
        imageViewReference = new WeakReference<ImageView>(imageView);
    }
    @Override
    protected Bitmap doInBackground(String... params) {
        String url = params[0];
        InputStream is = ImageLoader.load(url);
        return decodeSampledBitmap(is, 100, 100));
    }
    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (imageViewReference != null && bitmap != null) {
            final ImageView imageView = imageViewReference.get();
            if (imageView != null) {
                imageView.setImageBitmap(bitmap);
            }
        }
    }
}
```

Caching Bitmaps

Caching Bitmaps using LruCache

```
private LruCache<String, Bitmap> mMemoryCache;

@Override
protected void onCreate(Bundle savedInstanceState) {

    final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);

    // Use 1/8th of the available memory for this memory cache.
    final int cacheSize = maxMemory / 8;

    mMemoryCache = new LruCache<String, Bitmap>(cacheSize) {
        @Override
        protected int sizeOf(String key, Bitmap bitmap) {
            // The cache size will be measured in kilobytes rather than
            // number of items.
            return bitmap.getByteCount() / 1024;
        }
    };
    ...
}
```

Retaining Cached Bitmaps between orientation changes

```
class RetFragment extends Fragment {

    private static final String TAG = "RetFragment";
    public LruCache<String, Bitmap> mRetainedCache;

    public static RetFragment getInstance(FragmentManager fm) {
        RetFragment fragment = (RetFragment) fm.findFragmentByTag(TAG);
        if (fragment == null) {
            fragment = new RetFragment();
        }
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }
}
```

Goals


Adaptive UI


Offline support


Efficiency

Conclusion

- Support different devices
- Be invisible
- Be efficient
- Be reliable