Snívajú používatelia o mobilných androidoch?

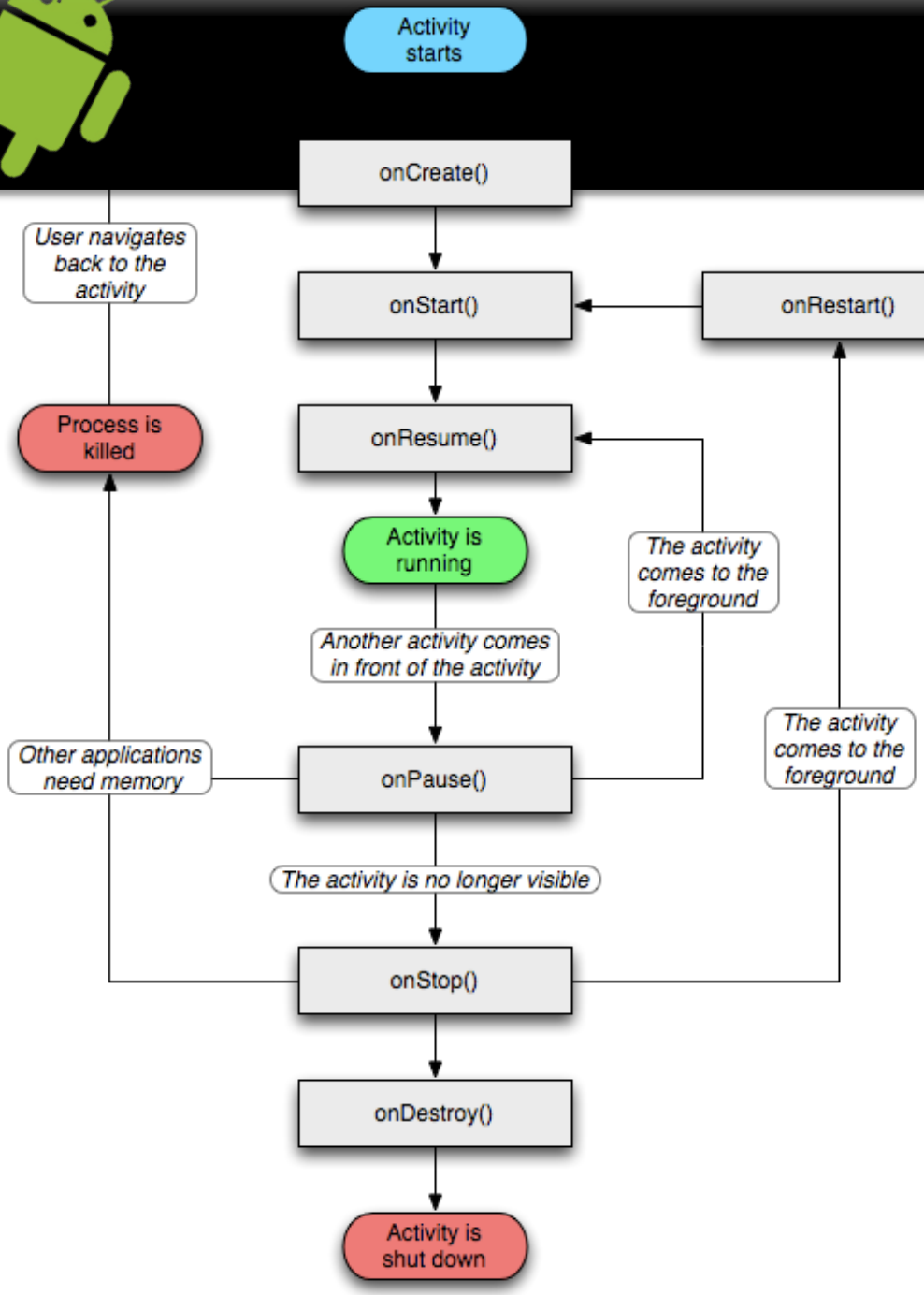
platforma pre mobilné aplikácie

Róbert Novotný
robert.novotny@upjs.sk

23. 3. 2011

ANDROID

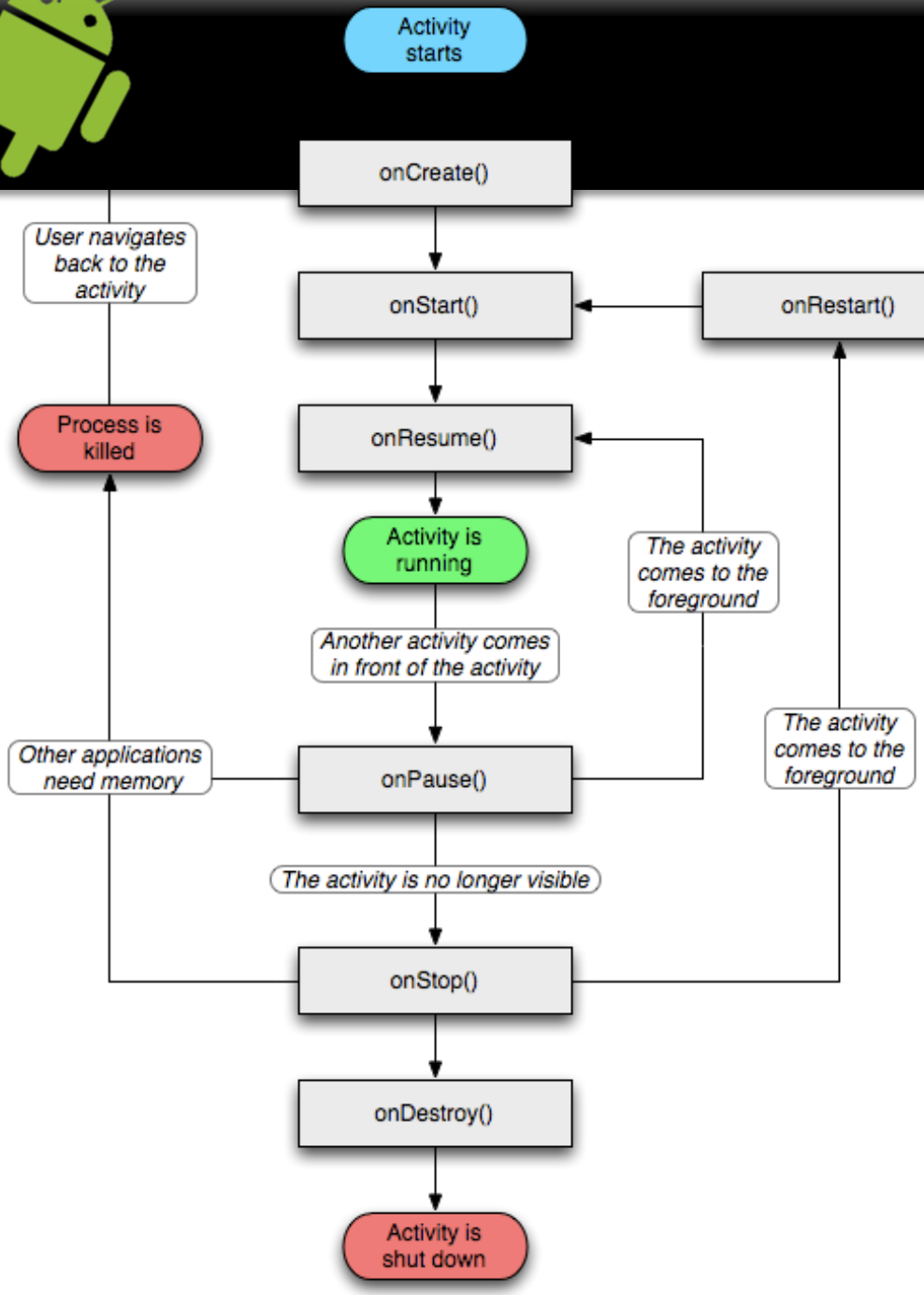
Životný cyklus aktivity



- každá z fáz životného cyklu aktivity je namapovaná na konkrétnu metódu
- prekrývaním metód reagujeme na príslušnú fázu

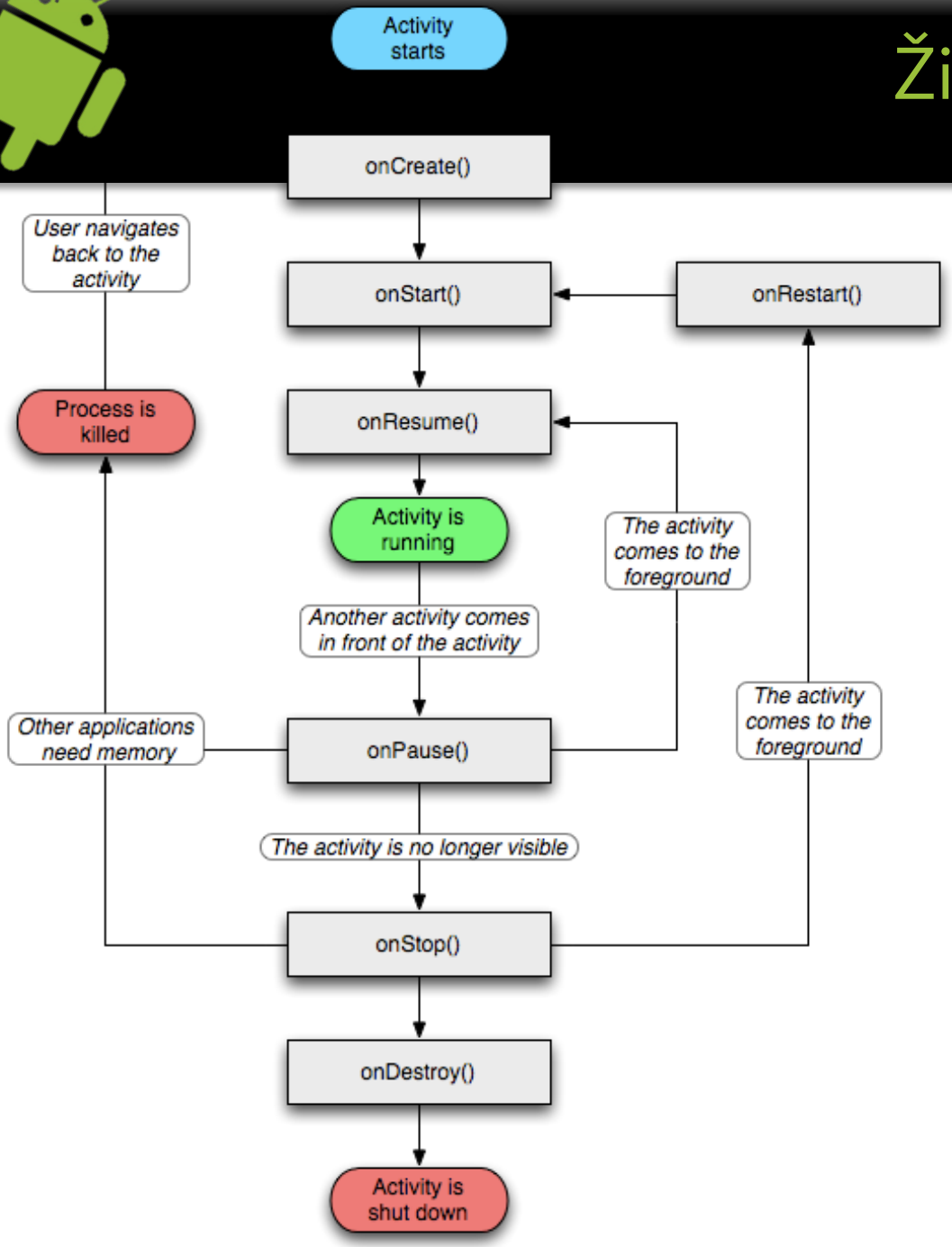
ANDROID

Životný cyklus aktivity



- onCreate()
  - začína životnosť
  - možnosť nastaviť globálny stav
- onStart()
  - aktivita sa stáva viditeľnou
  - volaná viackrát
    - aktivita sa môže skrývať a zobrazovať

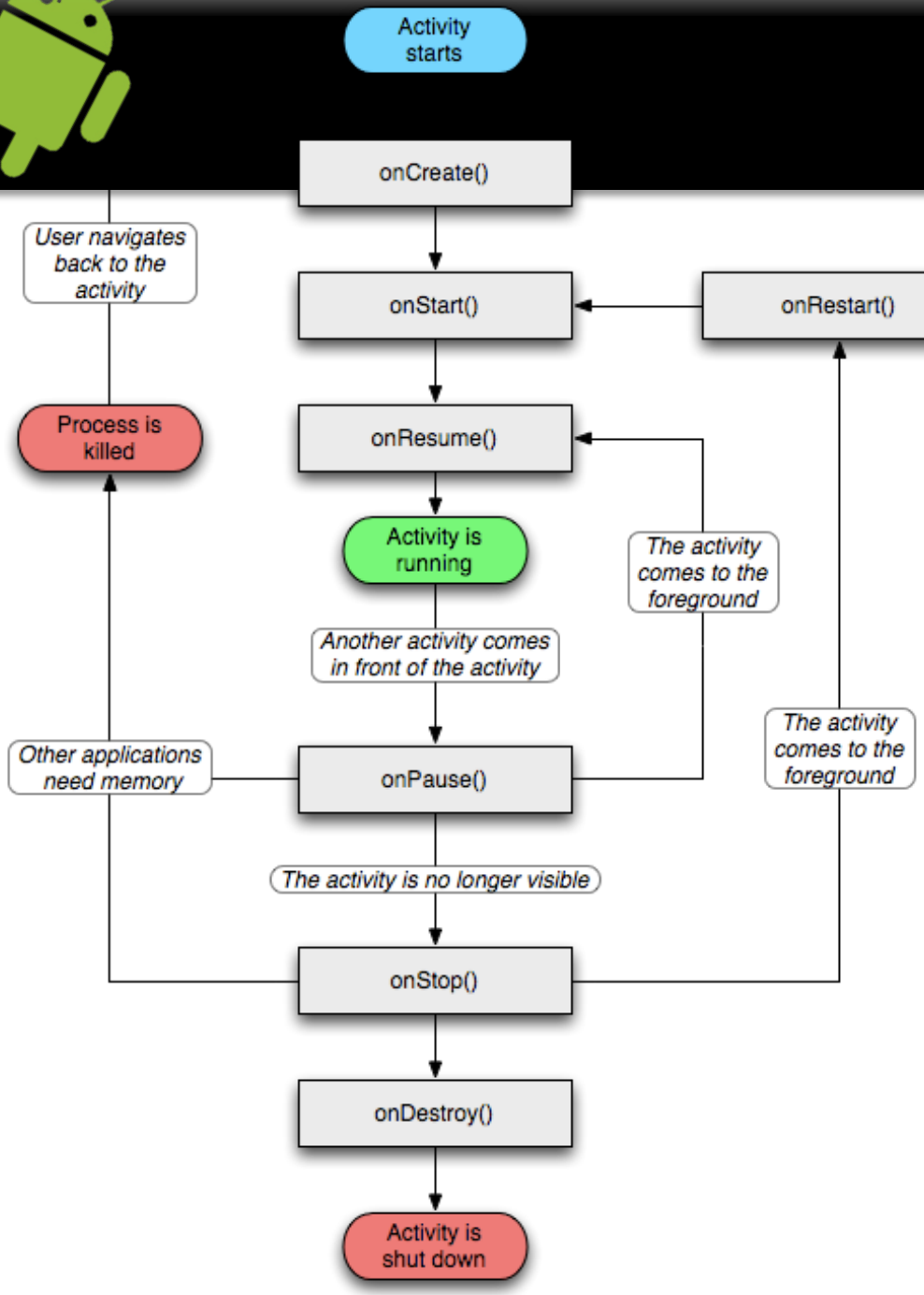
Životný cyklus aktivity



- **onResume()**
  - aktivita môže interagovať s používateľom
  - vhodné miesto na otváranie exkluzívnych prostriedkov
- **onStart()**
  - aktivita sa stáva viditeľnou
  - volaná viackrát
    - aktivita sa môže skrývať a zobrazovať

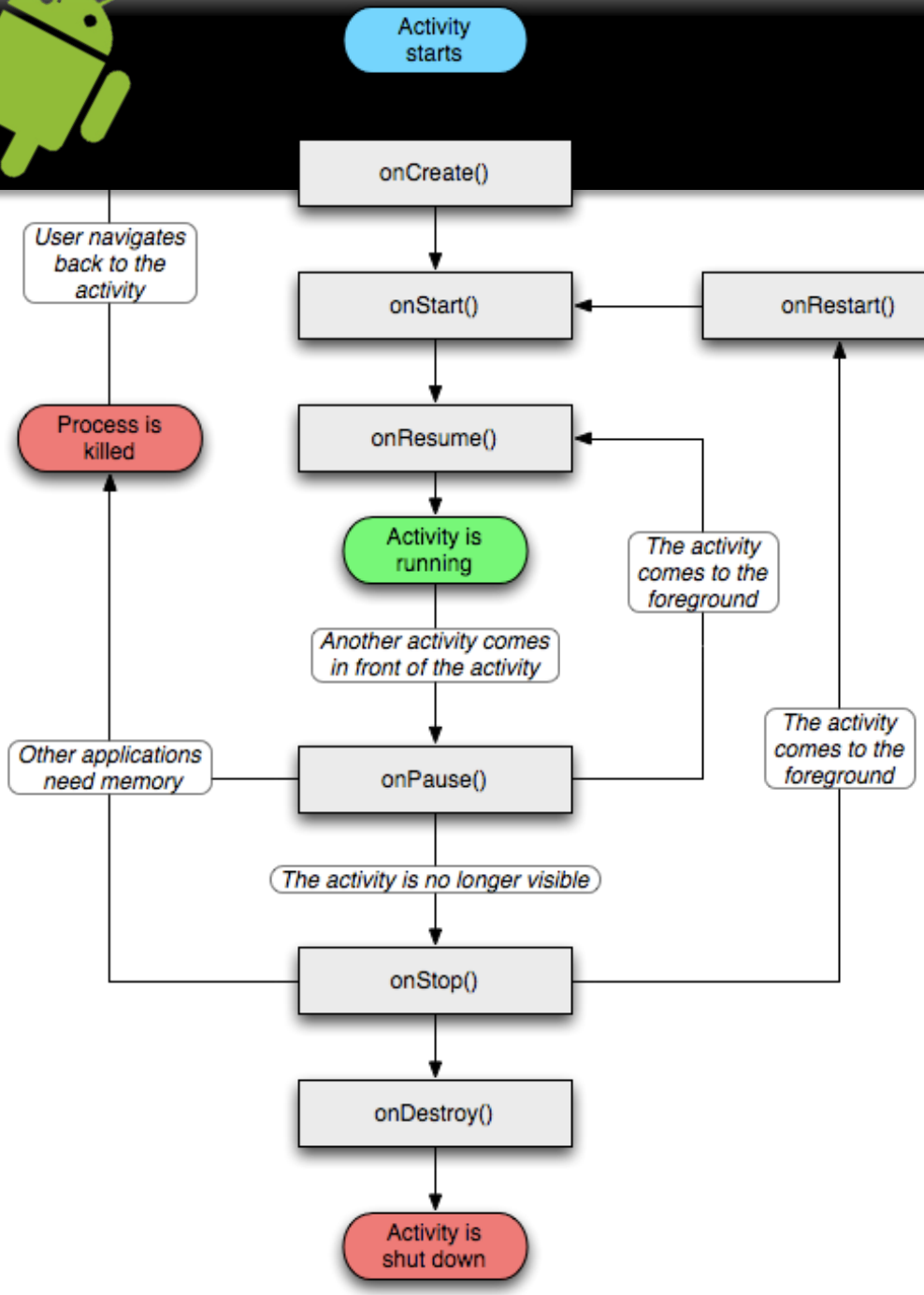
ANDROID

Životný cyklus aktivity



- onPause()
  - aktivita odchádza na pozadie, pretože bude prekrytá inou aktivitou
  - ideálne miesto na ukladanie stavu do databázy
  - má byť rýchla, keďže inak sa spomalí spúšťanie novej aktivity

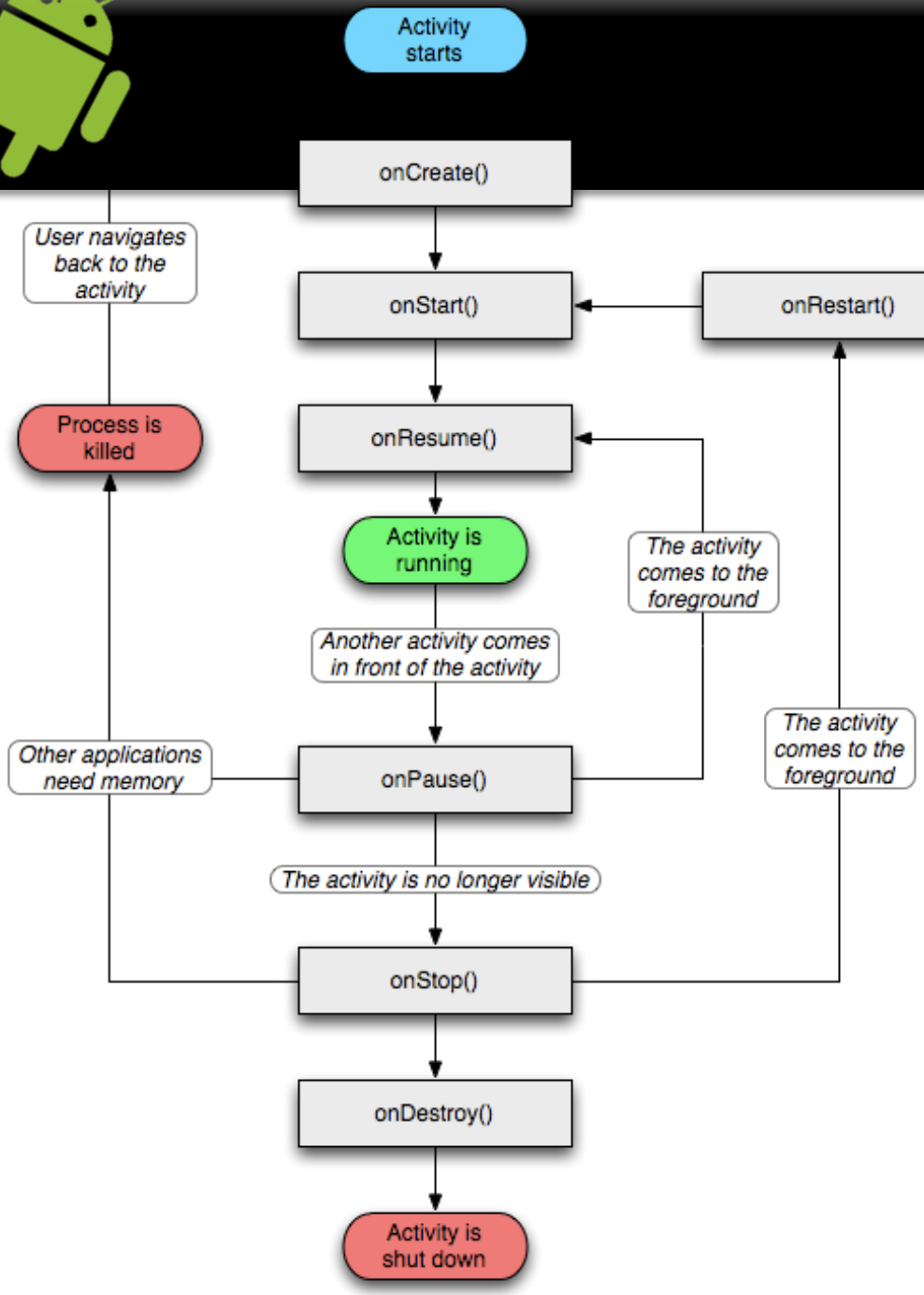
Životný cyklus aktivity



- onStop()
  - aktivita už nie je viditeľná
  - vhodné na deaktiváciu databázových kurzorov
- pozor! ak je nedostatok pamäte, nemusí sa zavolať!

ANDROID

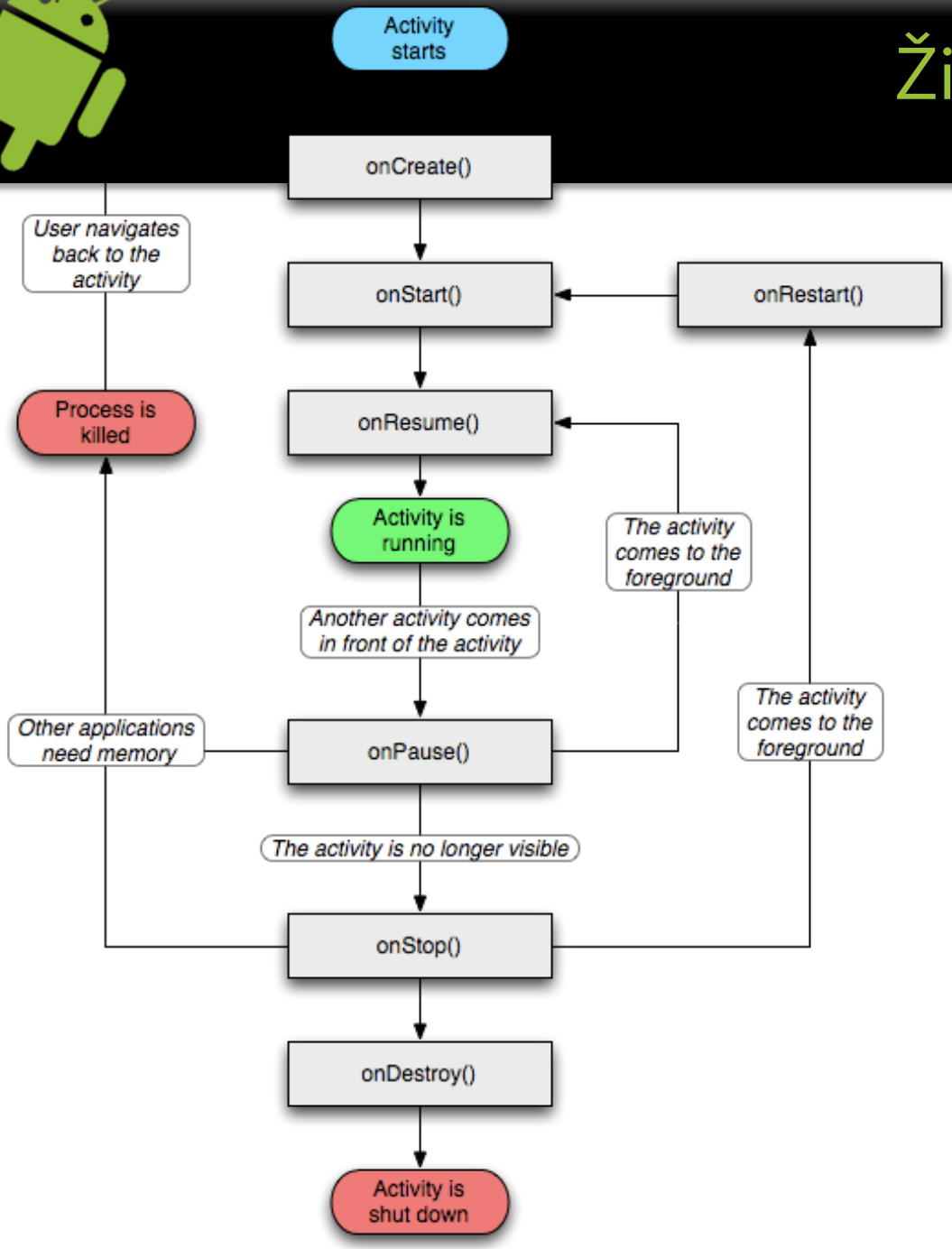
Životný cyklus aktivity



- onDestroy()
  - aktivita končí
  - buď niekto zavolal finish()
  - alebo systém upratuje pamäť
  - pozor! už nemusí byť volaná

ANDROID

Životný cyklus aktivity



- onRestart()
  - aktivita je znovuzobrazená používateľovi
  - vhodné miesto na reaktiváciu kurzorov

ANDROID

Ako ukladať stav?

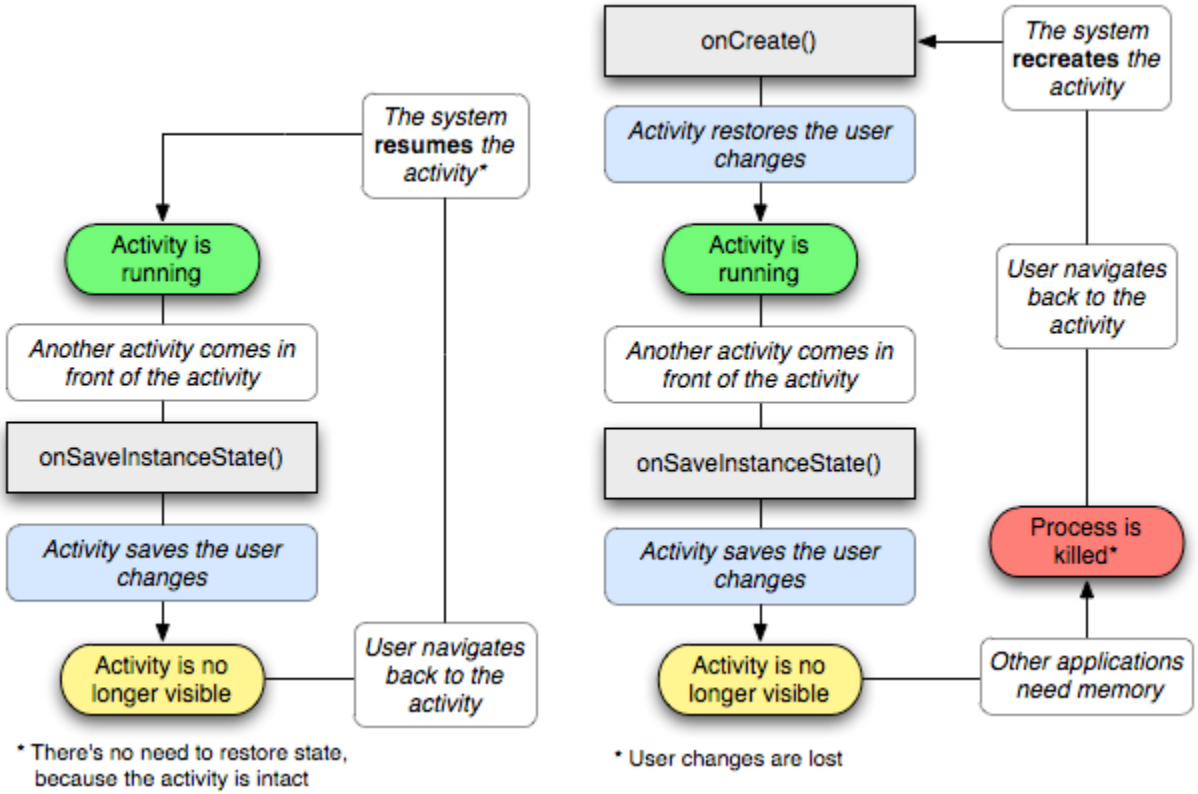
- stav treba ukladať priebežne
- odporúčanie: metóda **onPause()**
  - volaná vo chvíli, keď aplikácia odchádza na pozadie
  - obvykle commitneme stav do databázy/súboru
- špeciálna situácia: aktivita je prekrytá inou aktivitou
  - zavolá sa dvojica **onSaveInstanceInstance()**
  - uloží sa stav používateľského rozhrania
  - do objektu **Bundle** (mapa) vieme voliteľne naukladať dodatočné konfiguračné nastavenia

Ako ukladať stav?

- ak je aktivita A prekrytá inou aktivitou...
- ...a zrazu dôjde pamäť...
- ...systém odstrelí aktivitu A
- ak používateľ ukončí novú aktivitu, má sa zjaviť aktivita A
- tá je však už zničená!
- nastanú metódy onCreate() atď.

Čo so stavom aplikácie?

Ako ukladať stav?



Ako ukladať stav?

- ak je aktivita A prekrytá inou aktivitou...
- ...a zrazu dôjde pamäť...
- ...systém odstrelí aktivitu A
- ak používateľ ukončí novú aktivitu, má sa zjaviť aktivita A
- tá je však už zničená!
- musí sa vytvoriť a spustiť nanovo
- prejsť celou úvodnou inicializačnou fázou

onSaveInstanceState(Bundle)

onCreate(Bundle)

Odovzdávanie informácii cez bundle

- cez Bundle možno udržiavať stav, ktorý má
  - prežiť prekrytie aktivity inou aktivitou
  - a zároveň prežiť prípadné odstrelenie prekrytej aktivity
- v **onSaveInstanceState()** uložíme stav do Bundle
- v **onCreate()** si vyzdvihneme z Bundlu stav
  - pozor! Bundle tu môže byť **null**
  - to v prípade, že aktivita sa spúšťa klasickým spôsobom
- typické komponenty si ukladajú stav automaticky

Odovzdávanie informácii cez bundle

- pozor ale!
  - onSaveInstanceState() nie je súčasť životného cyklu aktivity!
  - môže, ale nemusí byť zavolaná
  - ak sa volá, tak sa volá pred **onStop()**
  - môže sa volať pred **onPause()**
- je to divné, ale dáva to zmysel
  - nie pri každom ničení aktivity treba ukladať stav
  - napr. ak používateľ explicitne finish()uje aktivitu tlačidlom Back

Zmena konfiguračného stavu

- v prípade, že sa zmení konfigurácia za behu, prejde celý cyklus
- vrátane onSaveInstanceState() a onCreate() s použitým bundlom
- notorický prípad:
  - otočenie mobilu

Ďalšie možnosti ukladania dát

- shared preferences
  - mapa pre primitívne hodnoty
    - boolean, float, int, long, string, množina stringov
- úložisko v pamäti zariadenia
  - získame FileOutputStream / FileInputStream
- externé úložisko [SD karta]
- SQL databáza SQLite
- sieťové spojenie

Ako ukladať stav?

- stav treba ukladať priebežne
- odporúčanie: metóda **onPause()**
  - volaná vo chvíli, keď aplikácia odchádza na pozadie
  - obvykle commitneme stav do databázy/súboru

SharedPreferences

- *de facto* mapa pre fixné dátové typy

```
SharedPreferences settings = getPreferences();
boolean silent = settings.getBoolean("silentMode", false);
```

- získame privátne úložisko pre aktuálnu aktivitu
- alternatívne môžeme získať zdieľané úložisko
  - môžeme uviesť meno
  - a rozsah oprávnení

```
SharedPreferences settings
  =  getSharedPreferences("prefs", MODE_WORLD_READABLE);
```

SharedPreferences

- na ukladanie slúži editor

```
SharedPreferences settings = ...
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("silentMode", mSilentMode);
editor.commit();
```

- ukladanie realizujeme v onPause() / onStop()
- nezabudnime na commit()!
- obnovenie dát typicky v onCreate().

Prístup k relačnej databáze

- k dispozícii je SQL databáza SQLite
- základné triedy:

SQLiteOpenHelper

- pomocná trieda pre správu životného cyklu databázy
  - vytváranie
  - upgrade

SQLiteDatabase

- metódy pre dopytovanie
- aktualizáciu
- transakcie

Cursor

- analógia ResultSet-u
- iteruje cez výsledok,
- ukazuje na aktuálny riadok
- umožňuje vyťahovať hodnoty

SQLiteOpenHelper

```
public class DatabaseOpenHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DatabaseOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

SQLiteOpenHelper

- Náš DatabaseOpenHelper vytvoríme konštruktorom
- parametrom je kontext
  - typicky inštancia aktivity, v rámci ktorej používame databázu
- Inštanciu databázy SQLiteDatabase získame
  - getReadableDatabase(): databáza určená pre čítanie
  - getWritableDatabase(): pre čítanie a zápis

```
DatabaseOpenHelper dbHelper = new DatabaseOpenHelper(this);
SQLiteDatabase db = getReadableDatabase();
```

- inštanciu helpera stačí vytvárať raz (v onCreate() aktivity)
- a udržiavať ju ako inštančnú premennú

Kurzory a dopytovanie

```
DatabaseOpenHelper dbHelper = new DatabaseOpenHelper(this);
Cursor cursor = getReadableDatabase().rawQuery(sql, params);
while(cursor.moveToNext()) {
  String lineId = cursor.getString(0);
  String departure = cursor.getString(2);
  String variant = cursor.getString(3);
  …
}
```

- kurzor sa správa ako iterátor cez tabuľku
- metóda moveToNext() ho posúva dopredu
  - na ďalší riadok
- zároveň povie, či je ešte k dispozícii ďalší riadok

Aktualizácia hodnôt

- klasické UPDATE/INSERT sú náchylné na chyby
- útoky SQL injection
- filozofia v Androide:
  - vytvoríme mapu z názvov riadkov do hodnôt
  - zavoláme príslušnú aktualizačnú metódu
- mapa: objekt typu **ContentValues**

```
ContentValues values = new ContentValues();
values.put("ID", 1);
values.put("NAME", "Android");
database.insert("PHONES", null, values);
```

názov tabuľky

null column hack

Null Column Hack

- ak je mapa ContentValues prázdna, databáza nevie vložiť riadok do tabuľky
- v parametri nullColumnHack však vieme uviesť názov nullabilného stĺpca, do ktorého sa vloží NULL, ak je je mapa prázdna

```
ContentValues values = new ContentValues();
values.put("ID", 1);
values.put("NAME", "Android");
database.insert("PHONES", null, values);
```

názov tabuľky

null column hack

Aktualizácia hodnôt

```
ContentValues values = new ContentValues();
values.put("ID", 1);
values.put("NAME", "Android");
database.update("PHONES", values, "ID=1", null);
```

- aktualizácia hodnôt: metóda **update()**
- tretí parameter: podmienka WHERE
- otázniky: zástupné znaky
- štvrtý parameter: hodnoty, ktorými sa nahradia zástupné znaky

Content Providers

- množstvo aplikácií potrebuje sprístupniť svoje dáta
  - kontakty
  - história volaní
  - fotky v galérii
  - ...
- dáta môžu byť na rozličných miestach
  - v SQLite databáze
  - v súborovom systéme
  - ...
- **Content Providers:** cesta ako sprístupniť dáta efektívnou cestou bez ohľadu na spôsob uloženia

Content Providers

- dáta sú chápané tabuľkovo-relačným štýlom
  - tabuľky
  - stĺpce
  - riadky
- každý content provider hovorí, aké „**tabuľky**" chce sprístupniť a aké „**stĺpce**" ponúka
- content provider v systéme je jednoznačne identifikovaný **URI** adresou
  - preddefinovaní provideri sú v balíčku **android.provider**
- k dátam sa následne pristupuje pomocou **kurzora**

ContentResolver

```
ContentResolver resolver = getContentResolver();
```

- v systéme je k dispozícii množstvo Content Providerov
- ale: aplikácie s nimi nepracujú priamo!
- Content Resolver: podľa požadovanej URI automaticky vyhľadá v systéme príslušný content provider a vykoná na ňom príslušnú operáciu

ContentResolver a kontakty

```
Uri providerUri = ContactsContract.Contacts.CONTENT_URI;
ContentResolver resolver = getContentResolver();
Cursor cursor
= resolver.query(providerUri, null, null, null, null);
while(cursor.moveToNext()) {
  Log.i("", cursor.getString(
   cursor.getColumnIndex(
      ContactsContract.Contacts.DISPLAY_NAME)));
}
cursor.close();
```

- metóda **query()**: prakticky SQLoidná metóda
  - potrebujem minimálne URI providera, z ktorého ťahám dáta
  - zvyšné NULLy: default hodnoty

ContentResolver a metóda query()

- parametre:
  - URI providera
  - projekcia: zoznam názvov stĺpcov, ktoré sa majú vrátiť
  - selekcia: WHERE podmienka
    - môže obsahovať otázniky ako zástupné znaky
  - parametre selekcie:
    - hodnoty zástupných znakov
  - parametre triedenia: ORDER BY
    - stĺpce, podľa ktorých triediť

ContentResolver a kontakty

- ako zistiť telefónne číslo?
- dvojfázovo:
  - zistím ID kontaktu (primárny kľúč) z providera **ContactsContract.Contacts.CONTENT_URI**
  - zistím telefónne číslo z providera **ContactsContract.CommonDataKinds.Phone**

ContentResolver a kontakty

```
ContentResolver cr = getContentResolver();
Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
                 null, null, null, null);
if (cur.moveToNext() > 0) {
    String id = cur.getString(
      cur.getColumnIndex(ContactsContract.Contacts._ID));
    Cursor pCur
       = cr.query(
           ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
           null,
           ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ?",
           new String[]{id}, null);
    if(pCur.moveToNext()) {
       Log.i("", pCur.getColumnValue(pCur.getColumnIndex(
           ContactsContract.CommonDataKinds.Phone.NUMBER));
    }
}
```