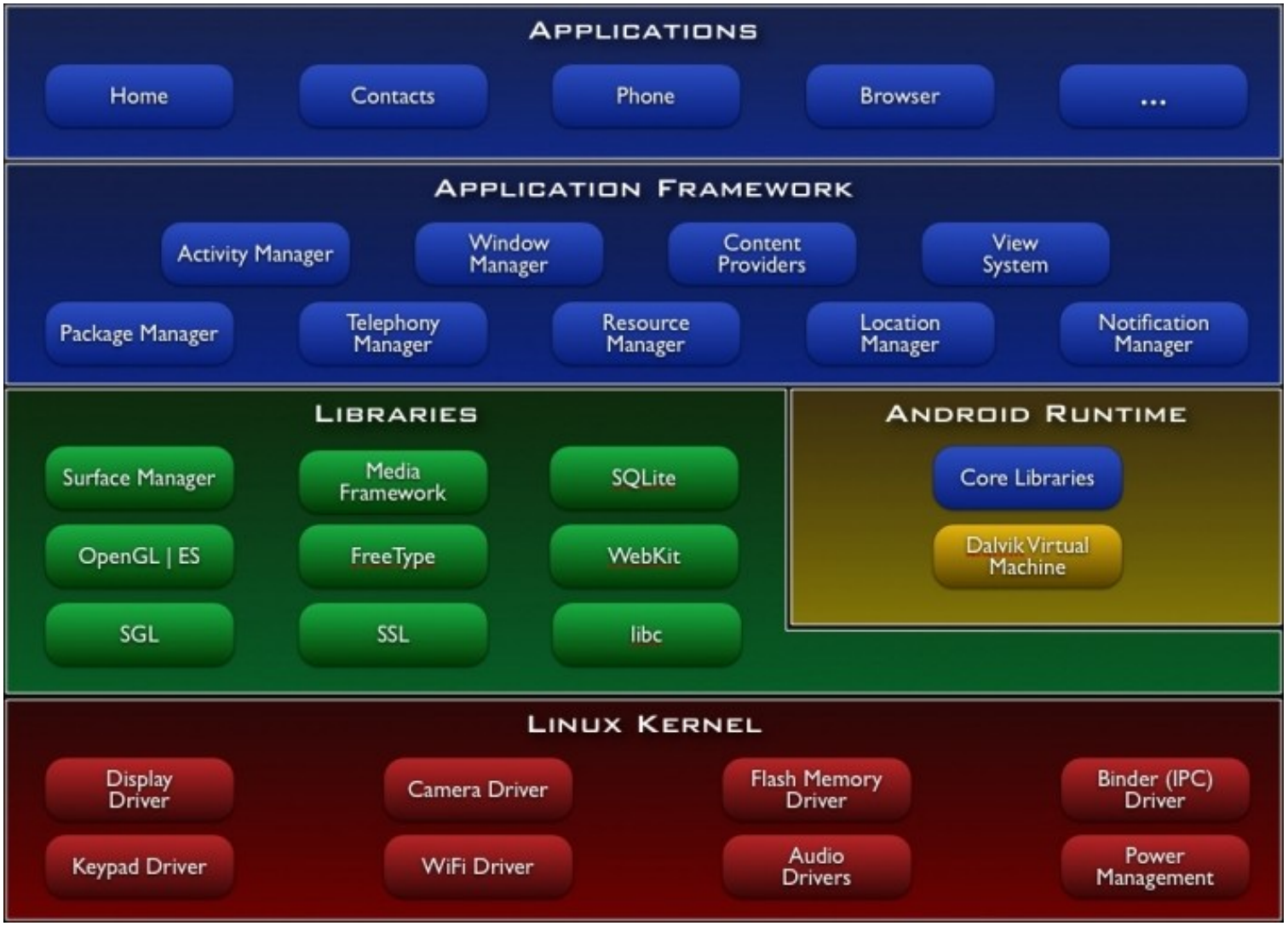
Lezione 2

Programmazione Android

- Architettura di un sistema Android
  - Kernel
  - La macchina virtuale
    - Dalvik e ART
  - Librerie e Framework
- Struttura di un'applicazione Android
  - In sviluppo, in deployment, in esecuzione
- Il sistema delle risorse
  - Architettura generale, risorse alternative
- AndroidManifest.xml

Un po' di architettura

The big picture



Studieremo a fondo

Studieremo per diletto

Largamente invisibili

Utile da sapere

Il Kernel

- Alla base di tutto c'è un **kernel Linux**
- Si tratta di un kernel completo, con tutte le primitive UNIX a cui siamo abituati
  - Processi, gestione della memoria, IPC, thread
  - Filesystem, utenti, diritti
  - Librerie, shell, comandi
  - Driver (sotto forma di moduli) per vari device
    - Tipicamente, quelli presenti in ogni particolare dispositivo
    - SD card, reti, telefonia, sensori, ecc.

Applicazioni native

- È possibile scrivere applicazioni che chiamano direttamente il kernel
  - Direttamente (via syscall) o via librerie (es., stdio.h)
  - Il codice deve essere compilato per il particolare processore in uso su un certo telefonino
    - In genere, ARM – ma non forzosamente
  - Deve poi essere “impacchettato” in un formato specificato per la distribuzione/installazione
    - Non troppo diverso dai vari .rpm, .deb, e simili
- Sconsigliato! (noi lo vedremo in fondo)

Dalvik & ART

- La stragrande maggioranza delle applicazioni “gira su” una **macchina virtuale**: **Dalvik**
- Analoga alla JVM con importanti differenze
  - Basata su registri (non su stack)
  - Set di istruzioni ottimizzato per risparmiare memoria e aumentare la velocità di esecuzione
  - Formato dei file eseguibili ottimizzato per risparmiare memoria
  - Eseguibile da più processi con una sola istanza
    - Tutto codice **rientrante** – sharing del codice di Dalvik via mmap()
  - **Non** sotto il controllo di Oracle (storica causa legale)

Dalvik & ART

- Due meccanismi di esecuzione
  - Android 1 - Android 4.3: **Dalvik**
  - Android 4.4: **Dalvik** per default, con opzione **ART**
  - Android 5 – in poi: **ART**
- Differenze:
  - ART pre-compila a install-time, non interpreta
  - Più lenta l'installazione, più veloce l'esecuzione
  - Largamente **invisibile** a programmatore e utente
- Stesso bytecode!

Dalvik & ART

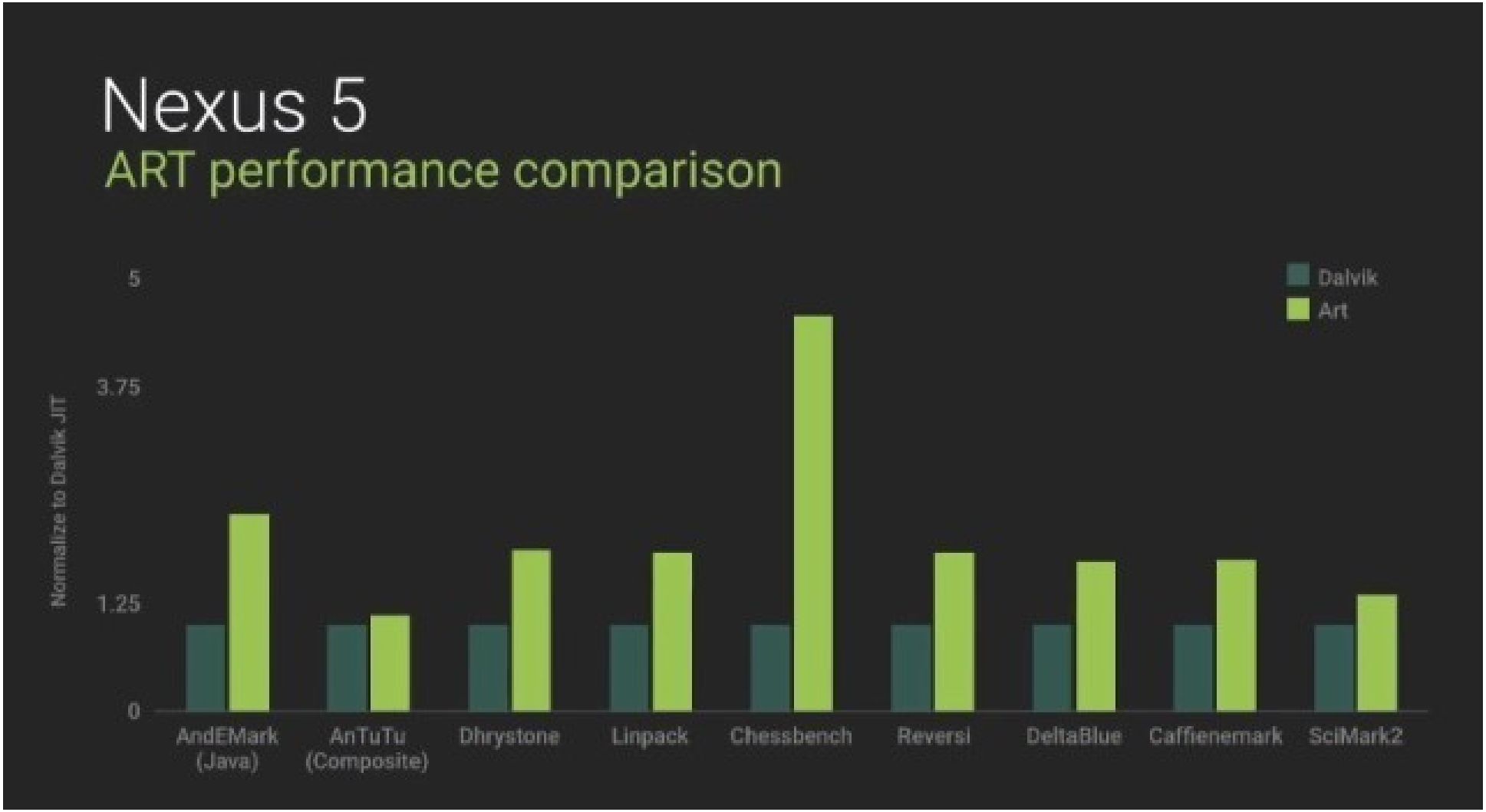
- Due meccanismi di esecuzione
  - Android 1 - Android 4.3: **Dal**
  - Android 4.4: **Dalvik** per def
  - Android 5 – in poi: **ART**
- Differenze:
  - ART pre-compila a install-time, non interpreta
  - Più lenta l'installazione, più veloce l'esecuzione
  - Largamente **invisibile** a programmatore e utente
- Stesso bytecode!

Casualmente, ART produce codice nativo, non bytecode... ulteriore assicurazione contro le cause di Oracle!

Dalvik & ART

- Sia Dalvik che ART sono entrambi rilevanti
  - Dalvik, perché è il target della toolchain di compilazione per il 99% delle app
  - ART, perché Google già da tempo sviluppa e supporta soltanto questa
    - Più veloce in esecuzione
    - Migliore gestione della *garbage collection*
      - Meno pause, grafica più fluida
    - Maggiore integrazione con profiling e debugging
    - Minor consumo di energia
      - Carica della batteria dura più a lungo

Dalvik & ART


Nexus 5
ART performance comparison
Dalvik
Art
Normalize to Dalvik JIT
5
3.75
1.25
0
AndEMark (Java)
AnTuTu (Composite)
Dhrystone
Linpack
Chessbench
Reversi
DeltaBlue
Caffienemark
SciMark2

Dalvik & ART

- Dalvik
  - **javac**: .java → .class
  - **dx**: .class → .dex
  - **dexopt**: .dex → .odex
  - **libdvm.so**: .odex → run
    - Interpretato + JIT
- ART
  - **javac**: .java → .class
  - **dx**: .class → .dex
  - **dex2oat**: .dex → ELF
  - **libart.so**: ELF → run
    - Esecuzione nativa con un po' di runtime

Alternativamente, si può programmare in Kotlin anziché in Java.

In ogni caso, il compilatore produce file .class che poi seguono la stessa strada di quelli prodotti da javac.

compile-time
install-time
run-time

Linux, VM, e sicurezza

- Ogni App viene eseguita dal kernel Linux
  - **In un processo separato**
    - Che esegue Dalvik che esegue il bytecode dell'app
    - Controllo dei **permessi** di accesso alle risorse logiche fatto dalla VM (i permessi sono concessi dall'utente-umano)
  - **Con uno user ID distinto**
    - Tutti i file creati dall'applicazione appartengono al “suo” user ID; altre applicazioni non possono accedere alla “sua” directory, né leggere i “suoi” file
    - È possibile che applicazioni *amiche* condividano processo e user ID – occorre però che siano firmate dallo stesso autore
    - Controllo dei **diritti** di accesso alle risorse fisiche fatto dal kernel (i diritti **non** sono controllati dall'utente-umano)

Linux, VM, e sicurezza

- Risultato complessivo
  - Notevole grado di **separazione** e **isolamento** delle Applicazioni
    - Ci può sempre essere un buco di sicurezza non patchato nel kernel Linux, ma l'uso dello stesso kernel usato per tutte le altre applicazioni rende l'eventualità remota
  - Android è un sistema piuttosto **sicuro**, ma...
    - Sono sempre possibili exploit basati sull'**ingegneria sociale**
      - **Una App convince l'utente darle particolari permessi**
      - **La App usa poi questi permessi per uno scopo diverso da quello pubblicizzato**
    - Molto più difficili gli attacchi veri

Linux, VM, e sicurezza

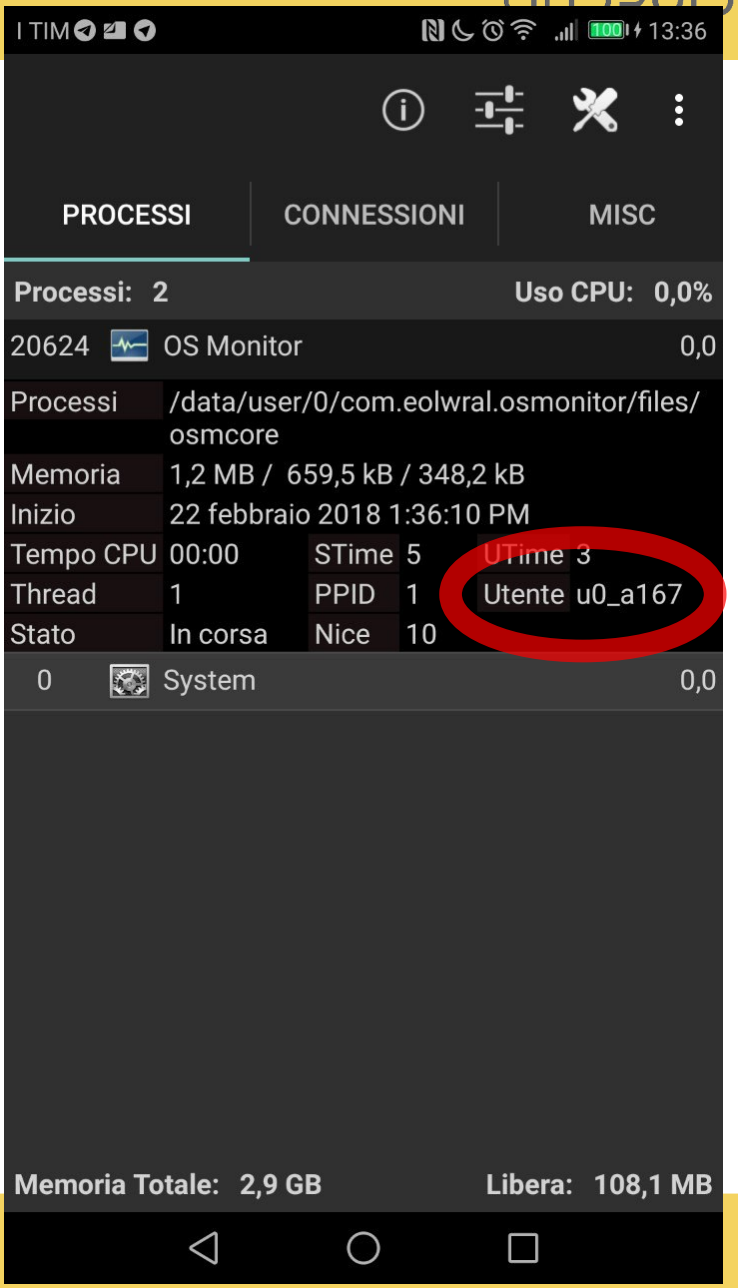
- In ambiente UNIX tradizionale, abbiamo
  - Le applicazioni (comandi) appartengono al sistema
  - Più utenti (umani) usano il sistema
    - Mutuamente malfidati
  - Il dominio di protezione è l'**utente**
- Su Android invece
  - C'è un solo utente umano, fidato ma inaffidabile
  - Le applicazioni sono mutuamente malfidate
  - Il dominio di protezione è l'**applicazione**

Linux, VM, e sicurezza

- I veri utenti (nel senso UNIX) sono **i programmatori delle varie App**
- Il proprietario del telefonino è (nel senso UNIX) quasi un **device** :-)
  - Non ha un suo userID
  - Non è proprietario di nessun file nel file system
  - Non è titolare di nessun processo
  - Non ha nemmeno un login e una password!
- L'utente non fa niente, se non tramite App!

Linux, VM, e sicurezza

- Android supporta
  - Più utenti (incluso un utente “guest”)
  - Più account per ogni utente
  - Più applicazioni per ogni utente
- Durante l’esecuzione, ogni processo ha uno user ID (UNIX) dato dal *prodotto cartesiano* delle ID dell’utente e dell’app



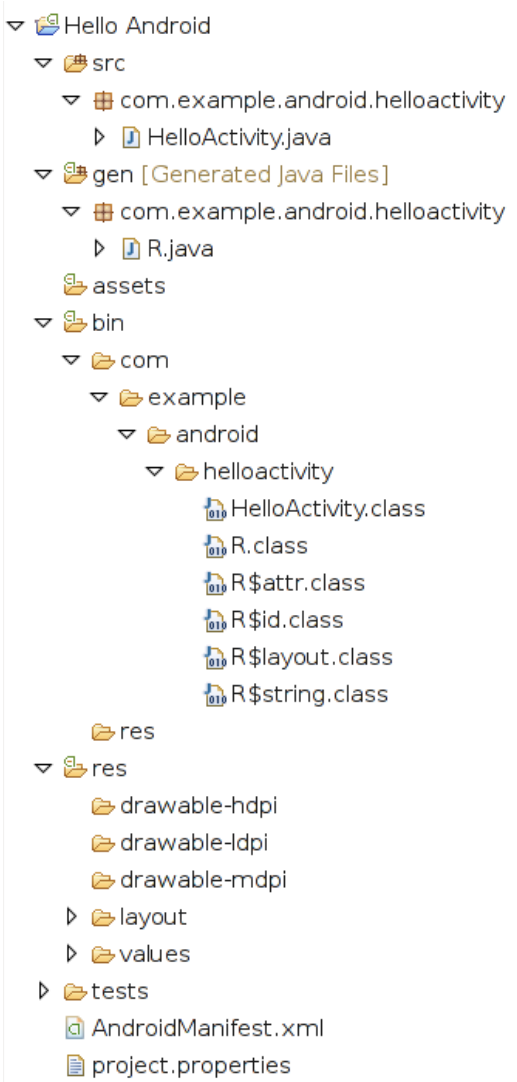
Struttura di un'applicazione Android

Struttura di un'applicazione

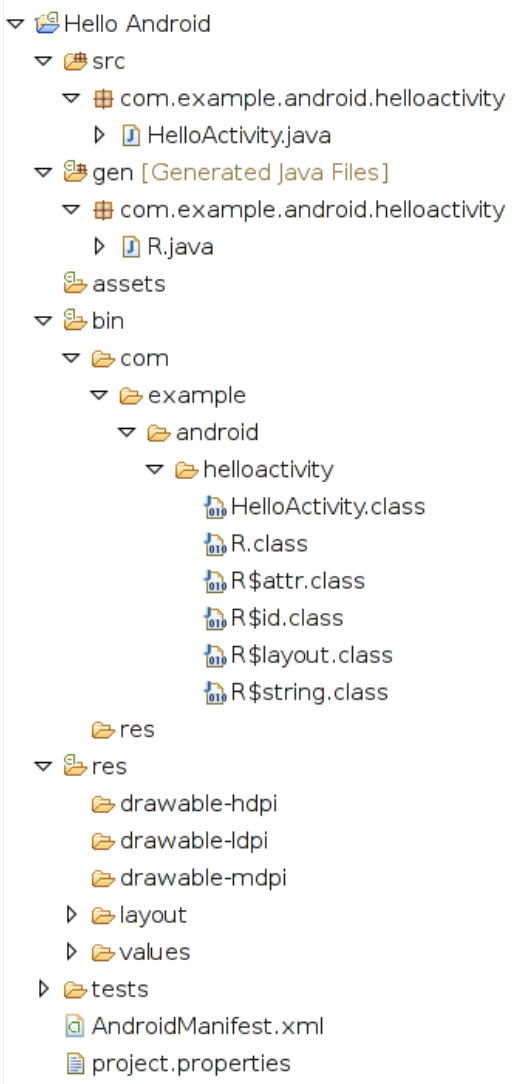
- Un'applicazione può assumere diverse forme nel corso della sua vita
  - In **sviluppo**: layout su disco (progetto)
  - In **deployment**: formato dei file .apk
  - In **esecuzione**: struttura in memoria
- Le varie forme sono legate da tre processi
  - **Build**: sorgente → .apk
  - **Deploy**: .apk (su un market/ecc.) → .apk (sul device)
  - **Run**: .apk → processo in memoria

Struttura di un progetto

- Un'applicazione in sviluppo è chiamata **progetto**
- Il wizard di creazione di un nuovo progetto Android dell'ADT crea lo scheletro per noi
- Solo alcune directory sono di interesse per lo sviluppatore
  - Altre vengono generate automaticamente



Struttura di un progetto (su Eclipse)

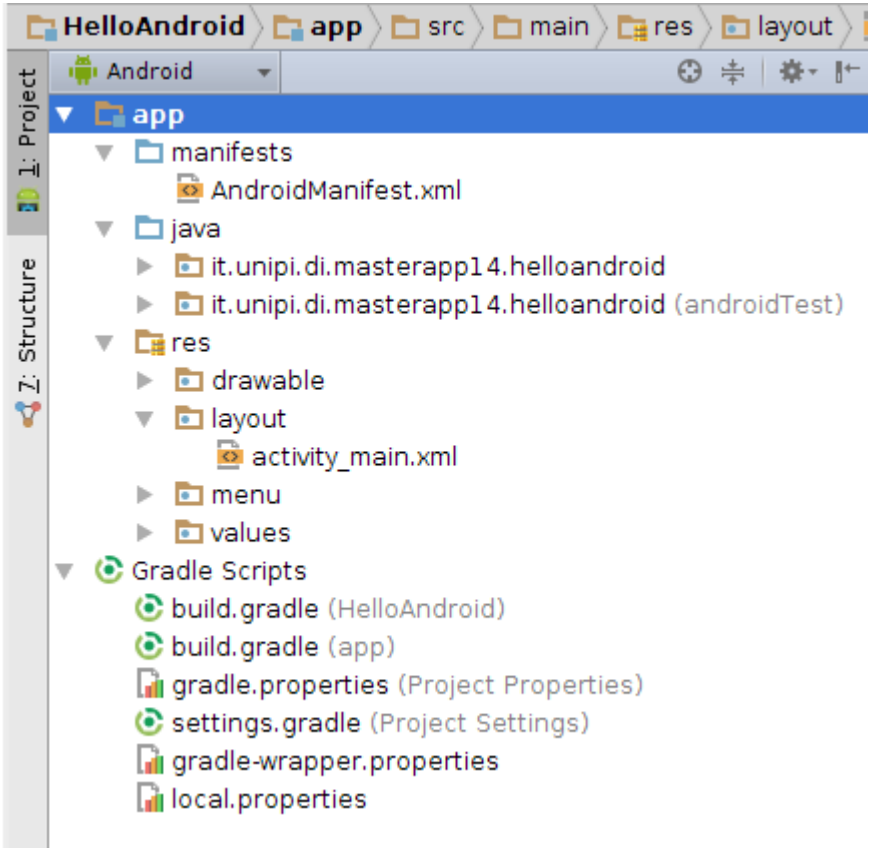


- **src** – sorgenti (.java, a volte .aidl o altri linguaggi)
- **gen** – sorgenti generati automaticamente (R.java)
- **assets** – file arbitrari, aggiunti al .apk
- **bin** – risultato della compilazione (risorse, .dex, .apk, …)
- **res** – risorse note al runtime Android
  - **animator** – animazioni basate su proprietà
  - **anim** – animazioni basate su intercalazione
  - **color** – colori
  - **drawable** – immagini raster o vettoriali
  - **layout** – descrizioni del layout della UI
  - **menu** – menu usati dall'applicazione
  - **raw** – file arbitrari (alternativa ad assets)
  - **values** – costanti (stringa, interi, array, ecc.)
  - **xml** – file XML arbitrari (incluse configurazioni)
- **libs** – librerie native custom
- **AndroidManifest.xml** – metadati sull'applicazione
- Altri **.properties**, **.cfg**, **.xml** – configurazioni varie

Lista parziale
(solo le più comuni)

Struttura di un progetto (su Android Studio)

- **manifests**
  - **AndroidManifest.xml** – metadati sull'applicazione
- **java** – sorgenti (.java, .kt, e unit test associati)
- **assets** – file arbitrari, aggiunti al .apk
- **bin** – risultato della compilazione (risorse, .dex, .apk, …)
- **res** – risorse note al runtime Android
  - **animator** – animazioni basate su proprietà
  - **anim** – animazioni basate su intercalazione
  - **color** – colori
  - **drawable** – immagini raster o vettoriali
  - **layout** – descrizioni del layout della UI
  - **menu** – menu usati dall'applicazione
  - **raw** – file arbitrari (alternativa ad assets)
  - **values** – costanti (stringa, interi, array, ecc.)
  - **xml** – file XML arbitrari (incluse configurazioni)
- **libs** – librerie native custom
- **Gradle Scripts** – configurazioni di build
- Altri **.properties**, **.cfg**, **.xml** – configurazioni varie

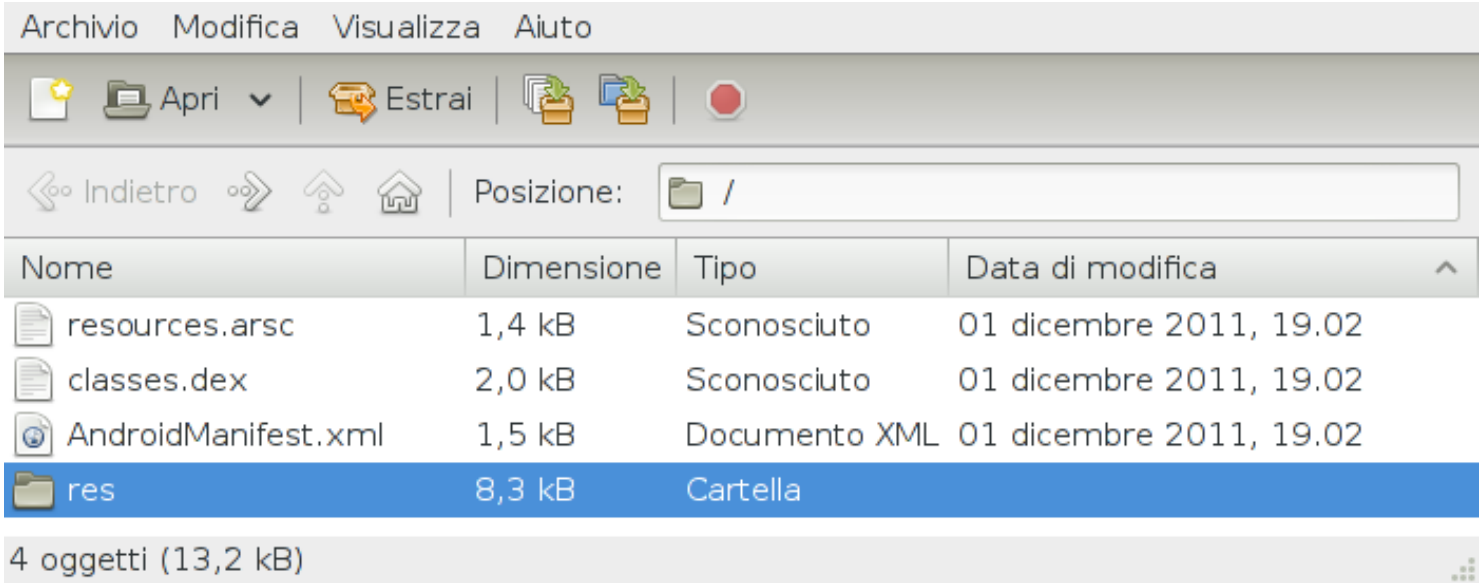


Struttura di un progetto

- La struttura che abbiamo visto è quella tipica di un'**applicazione**
- Esistono altre due forme di progetto Android:
  - Una **libreria** contiene componenti destinati ad essere usati da altre applicazioni
    - In questo modo, i membri di una famiglia di applicazioni correlate possono condividere componenti
    - Ogni libreria può essere usata da varie applicazioni; ogni applicazione può usare varie librerie
  - Un **progetto test** contiene codice usato per il testing di un'altra applicazione
- Li vedremo a tempo debito (tempo permettendo!)

Contenuti di un .apk

- Il **build** di un'applicazione Android produce un file in formato .apk
- Un .apk è una specializzazione di un .jar
  - Un .jar è una specializzazione di uno .zip



Contenuti di un .apk

- **resources.arsc** – file binario contenente la tabella che mappa ID a risorse
- **classes.dex** – tutti i .class dell'applicazione, convertiti in DEX, e unificati in un solo file
- **AndroidManifest.xml** – manifesto dell'applicazione
- **res/*** - file delle risorse
- **META-INF/*** - contiene i certificati pubblici (chiavi)
  - (solo per le applicazioni firmate)

Contenuti di un .apk

- **META-INF/MANIFEST.MF** contiene
  - Informazioni di versionamento
  - Un *digest* (checksum) dei contenuti di ciascun file
- **META-INF/CERT.*** – certificati RSA
- È necessario creare un proprio certificato per il deploy!
  - Uno di debug viene creato da ADT

```
Manifest-Version: 1.0
Created-By: 1.0 (Android)

Name: res/layout/main.xml
SHA1-Digest: 8syRBinqeui2pRMGibKHakq1Lew=

Name: AndroidManifest.xml
SHA1-Digest: q3nOaBAUplDlAalDCNaDnPQ0ELU=

...
```

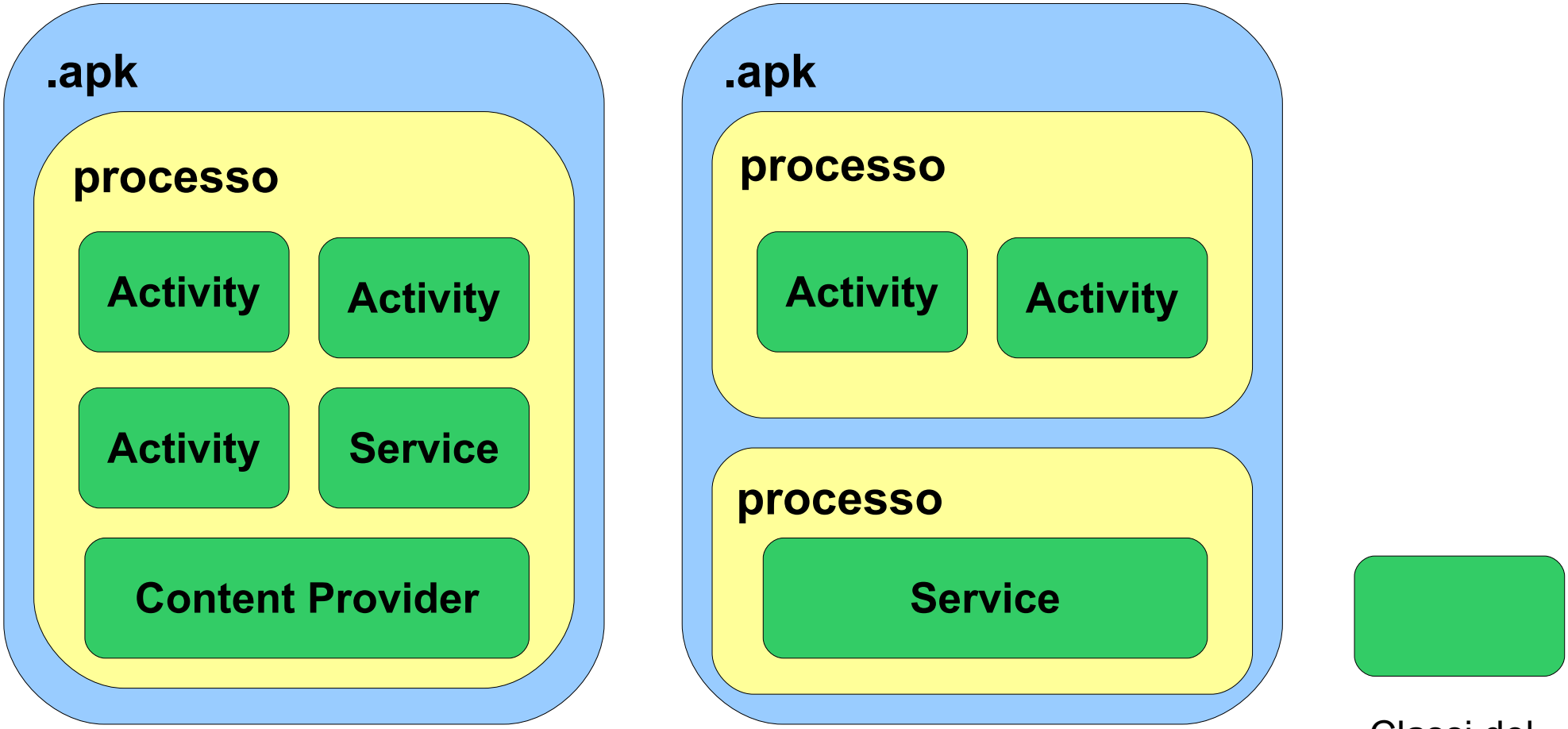
Contenuti di un .apk

- Un .apk è dunque un archivio contenente tutti i componenti di un'applicazione
  - **Auto-descrittivo** (grazie ai manifesti)
  - **Compatto** (grazie alla compressione)
  - **Affidabile** (grazie alla firma digitale)
    - Non è possibile aprire un .apk, sostituire alcuni componenti e re-impacchettarlo – la firma non sarebbe valida!
  - Facilmente **distribuibile** (un file unico)
  - Facilmente **installabile** (niente “setup.exe”)
    - In /sys/app (pre-installed) o /data/app (user-installed)

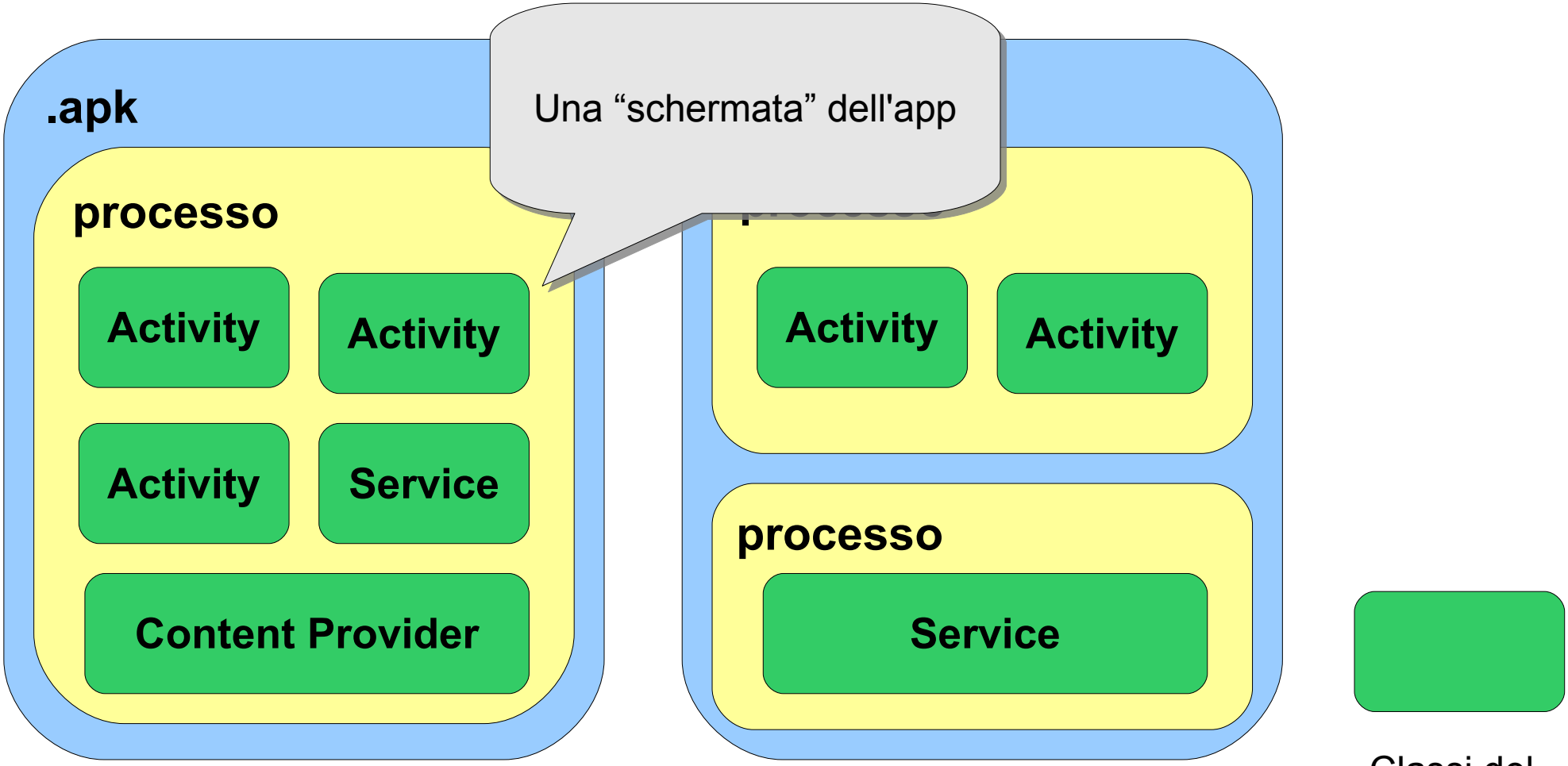
Struttura in memoria

- Una volta caricata in memoria, una applicazione è distinta in **componenti**
- Di norma, 1 app = 1 processo = 1 VM = 1 thread
  - Possibili variazioni: app multi-threaded, più app “amiche” in un solo processo, ecc.
- Il flusso di lavoro dell'utente (*task*) è fatto però spesso di componenti appartenenti ad applicazioni diverse, eseguiti da processi diversi
- Android **incoraggia** la condivisione (sicura) fra applicazioni
  - di dati
  - di componenti = funzionalità

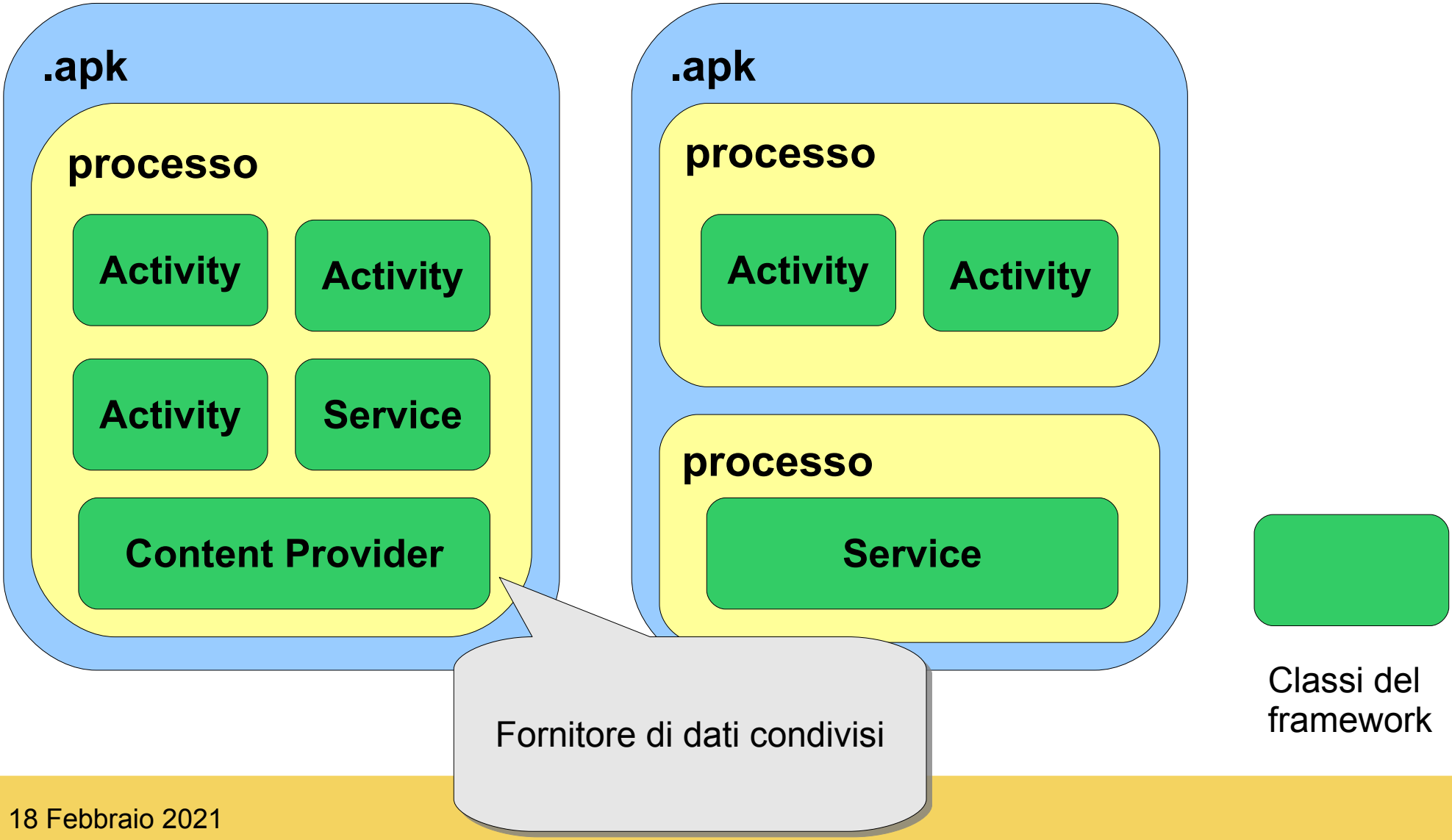
Struttura in memoria



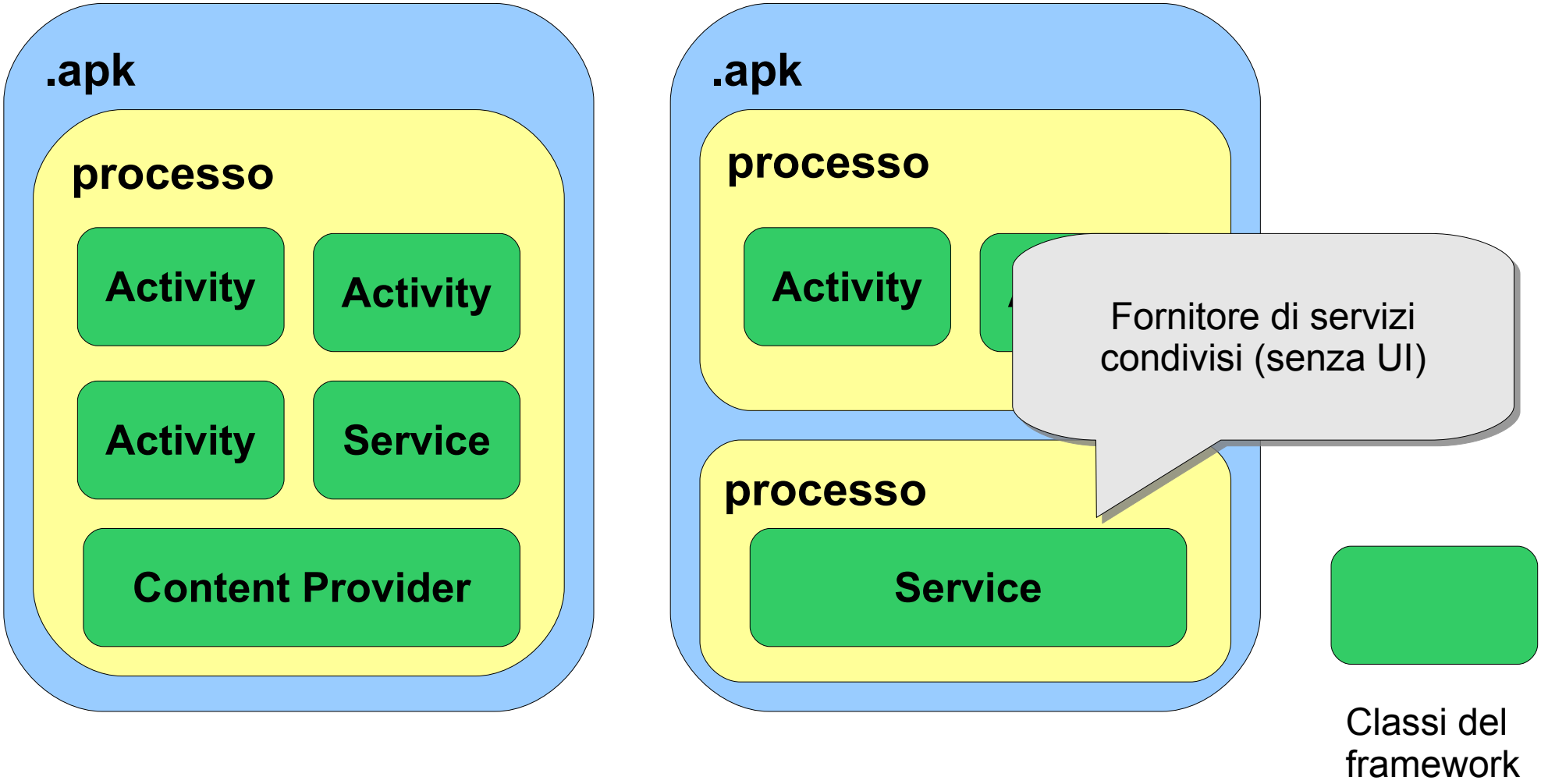
Struttura in memoria



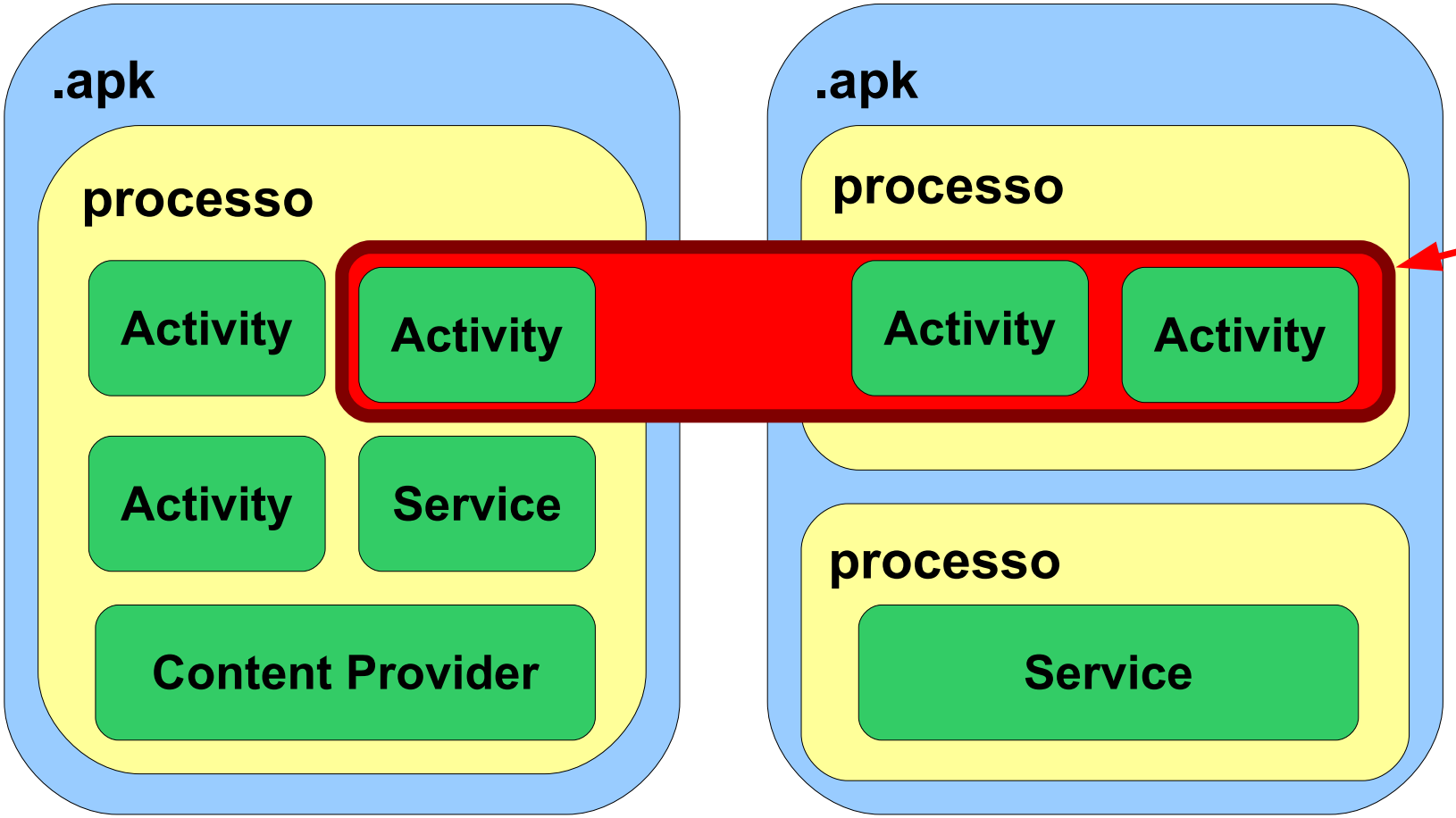
Struttura in memoria



Struttura in memoria



Struttura in memoria



Struttura in memoria

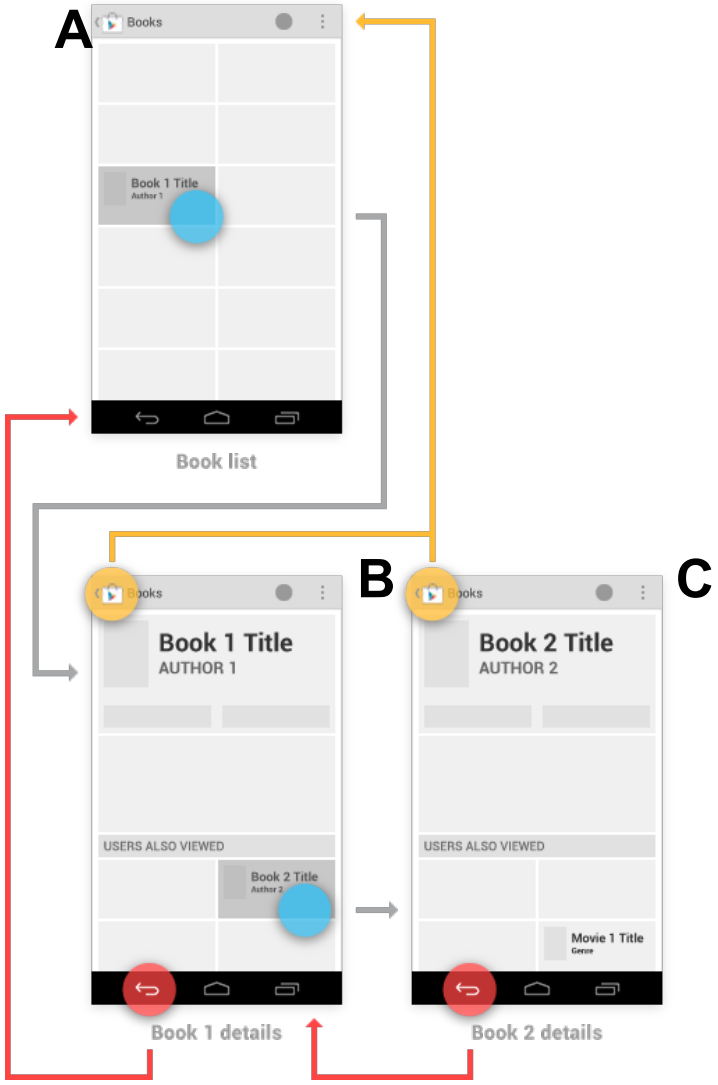
- Due visioni diverse
  - I **programmatori** percepiscono come “app” un .apk
  - Gli **utenti** percepiscono come “app” un task
- Per garantire un'esperienza utente gradevole, occorre che l'integrazione sia *seamless*
  - Assenza di cesure visuali / comportamentali
  - Uso di **stili**, **temi**, **stock icon**, style guide
  - Consistenza *fra* applicazioni più importante della consistenza *dentro* le applicazioni (o fra piattaforme)

Esempio (negativo)

Navigazione – primi cenni

- L'ordine temporale di passaggio fra un task e l'altro (anche di app diverse) costruisce uno stack
- Il tasto **back** serve a risalire lo stack
  - tasto fisico o virtuale
  - “torna alla schermata in cui ero prima”
- Il tasto **up** serve a risalire la gerarchia logica
  - icona nella Action Bar
  - “vai alla schermata logicamente superiore a questa”
  - introdotto con Android 3.0

Navigazione – esempio



- L'utente è sulla schermata **A**, che è una lista generale
- Su **A** seleziona un elemento, passa in **B** che è una vista di dettaglio
- Su **B** seleziona un elemento correlato, va su **C** che è anch'essa una vista di dettaglio
- In **C**, il tasto **Back** deve portare su **B**, il tasto **Up** invece su **A**
- **Pattern molto comune: master-detail**

Il sistema delle risorse

Due tipi di risorse

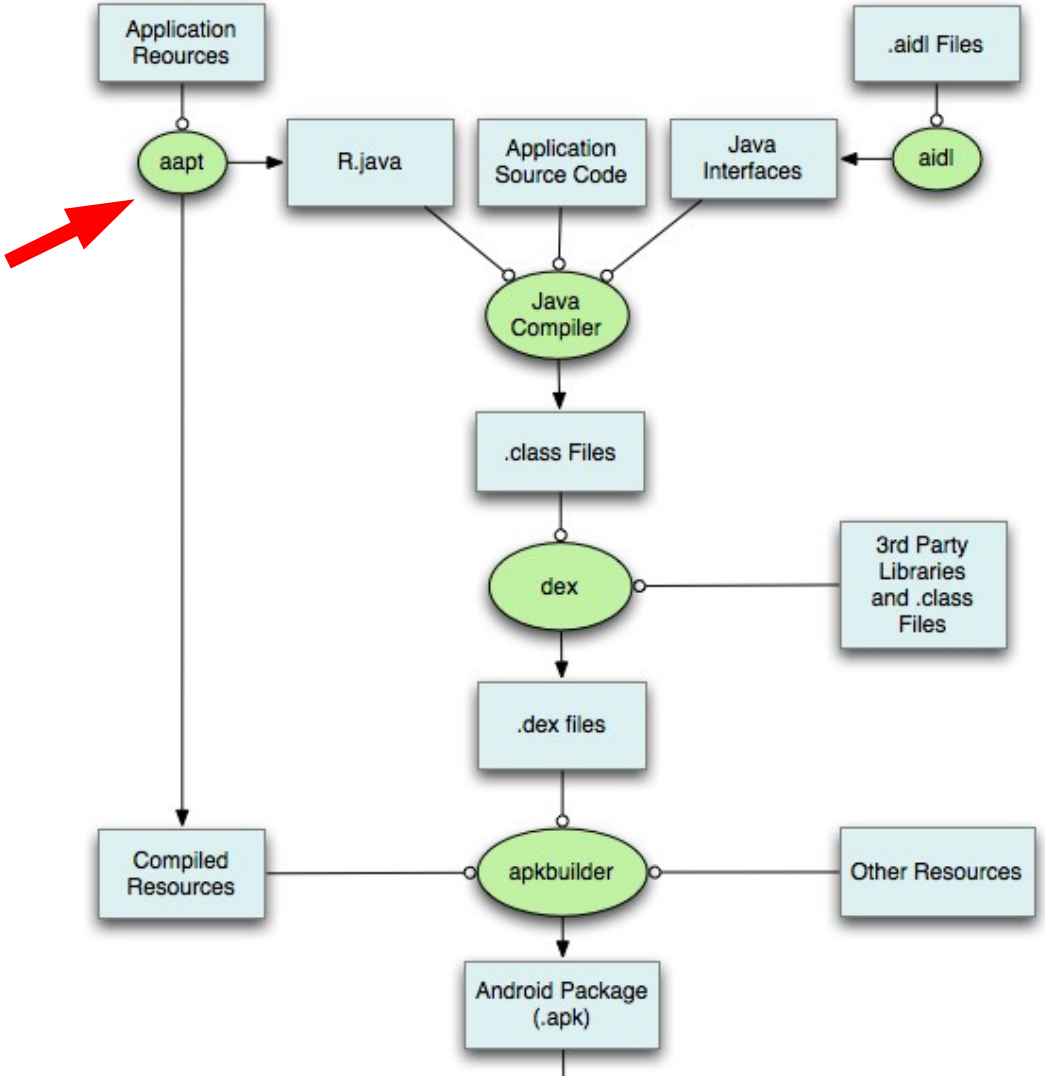
- Le risorse (in generale) sono dati usati dalle applicazioni
- Android distingue due casi:
  - Dati la cui struttura è nota al framework → **resources**
  - Altri dati → **assets**
- Gli asset vengono semplicemente aggiunti all'.apk, e acceduti come normali file
- Le risorse sono invece gestite dal **resource manager** in maniera articolata

Compilazione delle risorse

- Durante il processo di build, uno dei tool (**aapt –** Android Asset Packaging Tool) processa le risorse come segue:
  - Converte i file XML in res/ in formato binario
  - Genera una tabella di corrispondenza fra ID numerici e offset nel formato binario per ogni risorsa
  - Genera una classe Java (in gen/R.java) contenente tanti campi (final static) quante sono le risorse
    - Nome del campo = nome della risorsa (public static final)
    - Inner class = tipo della risorsa (public static final)
    - Valore = ID numerico della risorsa
  - Compila la classe R (public final), che entra nello spazio dei nomi del progetto

Compilazione delle risorse

- Il processo di build di un'applicazione Android è piuttosto complesso
- L'IDE nasconde (quasi) tutto
- È sempre possibile intervenire manualmente se necessario
  - Toolchain da riga di comando



Compilazione delle risorse - esempio -

- res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hello Android</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

- res/layout/main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Compilazione delle risorse - esempio -

- res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hello Android<
    <string name="hello_world">Hello world
    <string name="action_settings">Setting
</resources>
```

- res/layout/main.xml

```
<RelativeLayout xmlns:android="http://sche
    xmlns:tools="http://schemas.android.co
    android:layout_height="match_parent" a
    android:paddingRight="@dimen/activity_
    android:paddingTop="@dimen/activity_ve
    android:paddingBottom="@dimen/activity

    <TextView android:text="@string/hello_
        android:layout_height="wrap_conten

</RelativeLayout>
```

```
public final class R {
    /* … */

    public static final class string {
        public static final int action_settings=0x7f050000;
        public static final int app_name=0x7f050001;
        public static final int hello_world=0x7f050002;
    }

    public static final class layout {
        public static final int main=0x7f030000;
    }

   /* … */
}
```

Risorse semplici e complesse

- Chiamiamo risorse **semplici** quelle definite da un elemento XML contenuto in un tag <resources>
  - Le risorse semplici sono identificate da tipo+nome
  - Es: `<string name="app_name">Hello Android</string>`
- Chiamiamo risorse **complesse** quelle definite da un intero documento (file) XML
  - Le risorse complesse sono identificate da tipo+basename del file che le contiene
  - Es: /res/layout/main.xml

Default e alternative

- Per ogni risorsa, Android consente di definire un *default* e un numero (anche grande) di *alternative*
- Quando si accede a una risorsa a run-time, viene selezionata l'alternativa più specifica
  - In base all'ambiente o alle condizioni correnti
- Nel codice, si userà sempre un **identificativo di risorsa**; il binding alla risorsa vera è fatto **dinamicamente** dal framework
  - La famosa classe **R** generata automaticamente

Tipi di risorse

- Animation
- Color state list
- Drawable
- Layout
- Menu
- String
- Style
- Bool
- Color
- Dimension
- ID
- Integer
- Integer array
- Typed array
- Raw*
- XML*

* Acceduti tramite API speciali per leggere i dati grezzi

Esempio: definizione

- In res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloAndroidActivity!</string>
    <string name="app_name">Hello Android</string>
</resources>
```

- In gen/*package*/R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY. */
public final class R {
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Esempio: uso

- In res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout … >
    <TextView    android:layout_width="fill_parent"
                 android:layout_height="wrap_content"
                 android:text="@string/hello" />
</LinearLayout>
```

- In HelloAndroidActivity.java

```
public class HelloAndroidActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Accesso alle risorse

- Più formalmente, si può accedere a una risorsa conoscendo:
  - Il suo **package**
    - È opzionale per il “nostro” package
    - Il package “android” raccoglie tutte le risorse *di sistema*
  - Il suo **tipo**
    - Per risorse complesse, dato dalla sottodirectory in res/
    - Per risorse semplici (contenute in un <resources>), dato dal loro tag
  - Il suo **nome**
    - Per risorse complesse, dato dal basename del file
    - Per risorse semplici (contenute in un <resources>), dato dall'attributo android:name di un nodo
- In un file XML: @[*package*:]*tipo*/*nome*
- In Java: [*package*.]R.*tipo*.*nome*

Accesso alle risorse

- ATTENZIONE!
- Il valore di R.*tipo*.*nome* è l'**id numerico** della risorsa, non il suo **valore**
- In particolare, gli ID sono di tipo **int** – non String, Image, etc.
- Molti metodi di classi dei vari framework richiedono un ID
  - es. setContentView()
- Context.getResources() restituisce l'oggetto **Resources** del nostro package

Accesso alle risorse
Metodi di Resources

Context è una superclasse di Activity, quindi spesso basta chiamare getResources()

- boolean getBoolean(int id)
- int getInt(int id)
- float getDimension(int id)
- int [] getIntArray(int id)
- Drawable getDrawable(int id)
- ecc.

- InputStream openRawResource(int id)
- XmlResourceParser getXml(int id)
- AssetManager getAssets()

Accesso alle risorse
Metodi di AssetManager

- String [] list(String path)
- AssetFileDescriptor openFd(String filename)
- InputStream open(String filename)
- XmlResourceParser openXmlResourceParser(String filename)
- ecc.

- Simile a un accesso a file, ma...
  - Le risorse sono compresse, vengono decompresse
  - Il sistema può mantenere una cache (accesso più rapido)

Risorse alternative

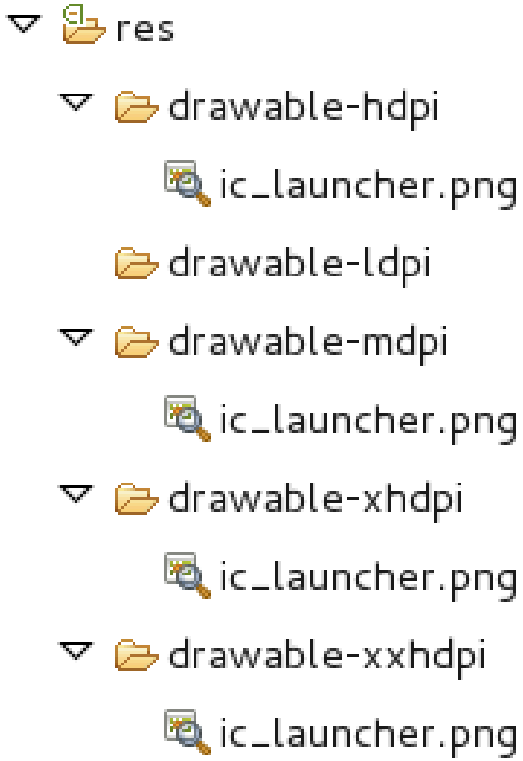
- In moltissimi casi, le risorse devono essere **adattate** all'ambiente
- Alcuni esempi
  - Stringhe per le varie lingue (localizzazione)
  - Colori adattati al paese (es.: bandiere)
  - Layout diversi in base all'*orientazione* del dispositivo :)
  - Icone diverse per carrier diversi (es.: logo)
  - Immagini a risoluzioni diverse
  - ecc.

Risorse alternative

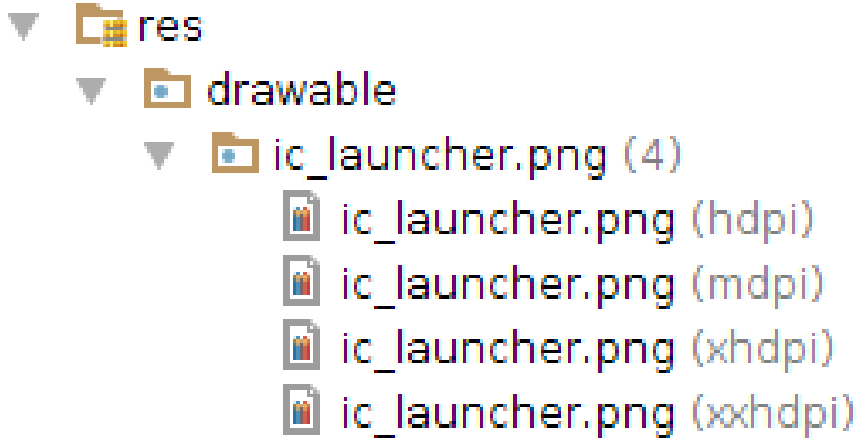
- Si possono definire risorse specializzate tramite l'uso di **qualificatori**
- Nella directory res/, si affiancano alle sotto-directory che già conosciamo (che sono i *default*) delle directory **qualificate**
- Forma: res/*tipo*-***qualificatori***/*file*
- Esempio:
  res/drawable/icon.png
  res/drawable-ldpi/icon.png
  res/drawable-hdpi/icon.png

Risorse alternative

- Eclipse presente le varianti come cartelle, e le risorse come file



- Android Studio presenta le risorse come cartelle, e le varianti come file



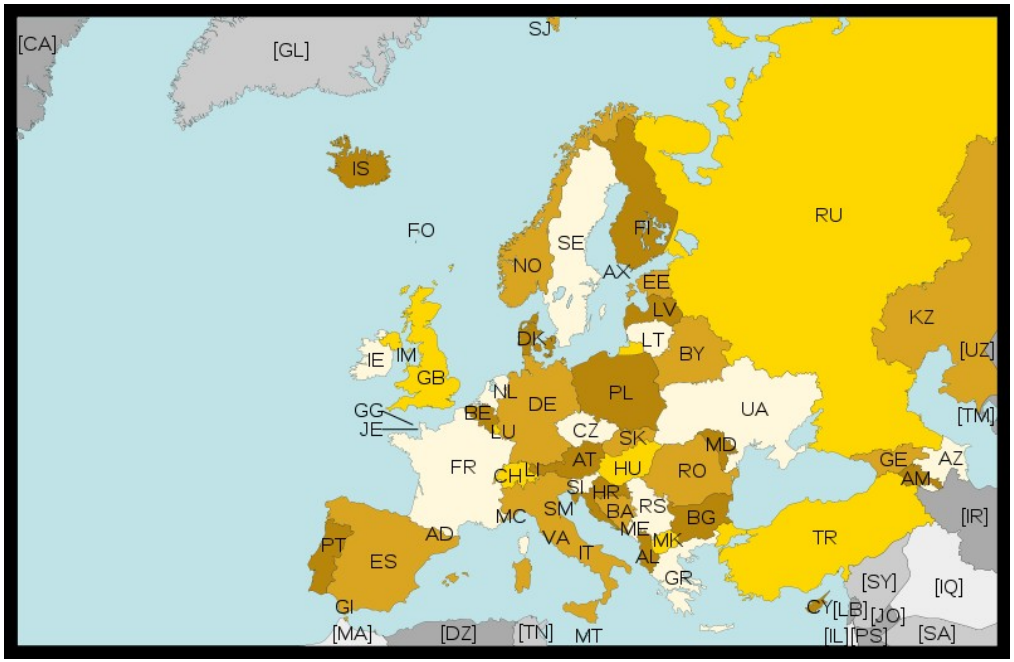
Qualificatori: SIM e rete

- Mobile Country Code / Mobile Network Code
- Formato: -mcc*MCC,* -mnc*MNC*
- -mcc222 = Italia, -mcc222-mnc010 = Vodafone IT

MCC	MNC	Marca	Operatore	Note
222	1	TIM	Telecom Italia SpA	
222	2	Elsacom		Ritirato
222	10	Vodafone	Vodafone Omnitel N.V.	
222	30	RFI	Rete Ferroviaria Italiana	
222	77	IPSE 2000		Ritirato
222	88	Wind	Wind Telecomunicazioni SpA	
222	98	Blu		Ritirato
222	99	3 Italia	Hutchison 3G	

Qualificatori: lingua e regione

- Lingua e regione
- Formato: -*lingua*-r*regione*
- -it = italiano,
  -it-rCH = Canton Ticino,
  -en-rUS = Inglese, USA
  -es-rUS = Spagnolo, USA



- Soliti codici ISO-639 e ISO-3166:
  - Lingue: it, fr, en, es, …
  - Regioni: IT, FR, CA, GB, US, ES, …
  - Le Regioni coincidono (quasi) sempre con i TLD

Qualificatori: direzione di scrittura

- Linguaggi che si scrivono da sinistra a destra o da destra a sinistra
- Supportata da Android 4.2 in poi
- Formato: -direzione
- -ldrtl – right to left
  - es.: arabo, ebraico
- -ldltr – left to right
  - es.: italiano, greco, russo

Qualificatori: schermo

- Dimensione minima dello schermo
  - Lungo l'asse minore, quale che sia
- Formato: -sw*N*dp
- -sw300dp = minimo 300 pixel lungo l'asse minore

- Dimensione dello schermo in larghezza
- Formato: -w*N*dp
- -w720dp = minimo 720 pixel in larghezza

Qualificatori: schermo

- Dimensione dello schermo in altezza
- Formato: -h*N*dp
- -h600dp = minimo 600 pixel in altezza

- Dimensione generale dello schermo
- Formato: -*dimensione*
- -small, -normal, -large, -xlarge

Qualificatori: schermo

- Aspetto dello schermo (rapporto w:h, approx)
- Formato: -*aspetto*
- -long (stile 16:9), -notlong (stile 4:3)

- Orientamento dello schermo
- Formato: -*orientamento*
- -port (portrait), -land (landscape)

Qualificatori: ambiente

- Che tipo di UI è disponibile
- Formato: -*UI*
- -car (installato in dock in auto o parte del cruscotto)
- -desk (in un dock sul tavolo)
- -television (grande schermo, utente seduto lontano)
- -appliance (sistema embedded, senza display)
- -watch (display piccolo, portato al polso)

- Night mode: se è giorno o notte
- Formato -*nightmode*
- -night (è notte), -notnight (è non-notte :-)

Qualificatori: schermo (ancora!)

- Densità del display
- Formato: -*densità*dpi
- Valori ammessi:
  - -ldpi = ~120dpi, -mdpi = ~160dpi, -hdpi = ~240dpi
  - -xdpi = ~320dpi (Retina-style), -xxhdpi = ~480dpi, -xxxhdpi = ~640dpi
  - -tvdpi = ~213dpi (densità delle TV)
  - -nodpi = pixel assoluti, bitmap da non scalare

- Tipo di touch screen
- Formato: -*tipotouch*
- -notouch (eek!), -stylus (penna [obsoleto]), -finger (dito)

Qualificatori: input

- Tastiera
- Formato: -keys*tipo*
- -keysexposed, -keyshidden, -keyssoft
- Formato: -*tipohw*
- -nokeys, -qwerty, -12key

- Tasti “cursore”
- Formato: -nav*tipo*
- -navexposed, -navhidden
- Formato: -*tiponav*
- -nonav, -dpad, -trackball, -wheel

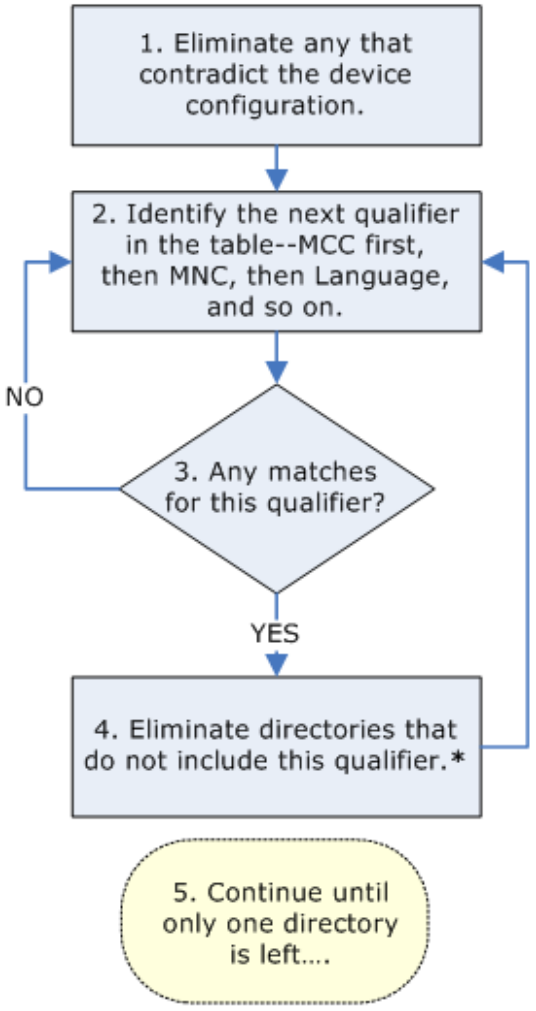
Qualificatori: versione OS

- Versione di Android (API level) corrente
- Formato: -v*versione*
- -v9 (Gingerbread), -v14 (Ice Cream Sandwich)

- Si possono indicare più qualificatori in sequenza
- Tutte le parti sono opzionali, ma quelle presenti **devono** essere nell'ordine in cui le abbiamo presentate!
- Esempio:
  - **res/drawable-es-rVA-night/button.png** → pulsante da usare in Vaticano la notte se la lingua dell'utente è lo spagnolo

Risorse alternative

- **A run-time**, il Resource Manager individua, per ogni risorsa, quale fra le tante alternative usare
- Usa l'algoritmo a lato per cercare il **match migliore**
- Ogni volta che la configurazione corrente cambia, il sistema **riavvia** l'Activity corrente con il nuovo insieme di risorse migliori
  - Salvo che l'Activity gestisca a mano...



* If the qualifier is the screen density, Android selects a "best" match and the process is done.

Risorse alternative Gestione dinamica

- In particolare, un'Activity può:
  - **Ignorare le modifiche**
    - Viene chiusa e riavviata (con le nuove risorse) dal sistema
  - **Salvare e ripristinare**
    - Si dichiara un oggetto che viene preparato dalla “vecchia” Activity e passato alla “nuova”, che lo usa per ripristinare uno stato transiente
  - **Gestire manualmente il cambiamento**
    - L'Activity **non** viene chiusa, ma viene chiamato un suo metodo passando i dettagli della nuova situazione

Richiedere una configurazione

- È possibile specificare nel **manifest** di una app che viene richiesta una particolare configurazione
- Es.: dispositivi di input

```
<uses-configuration
  android:reqFiveWayNav=["true" | "false"]
  android:reqHardKeyboard=["true" | "false"]
  android:reqKeyboardType=["undefined" | "nokeys" | "qwerty" | "twelvekey"]
  android:reqNavigation=["undefined" | "nonav" | "dpad" | "trackball" | "wheel"]
  android:reqTouchScreen=["undefined" | "notouch" | "stylus" | "finger"] />
```

Richiedere una configurazione

- È possibile specificare nel **manifest** di una app che viene richiesta una particolare configurazione
- Es.: caratteristiche dello schermo

```
<supports-screens android:resizeable=["true"| "false"]
                  android:smallScreens=["true" | "false"]
                  android:normalScreens=["true" | "false"]
                  android:largeScreens=["true" | "false"]
                  android:xlargeScreens=["true" | "false"]
                  android:anyDensity=["true" | "false"]
                  android:requiresSmallestWidthDp="integer"
                  android:compatibleWidthLimitDp="integer"
                  android:largestWidthLimitDp="integer"/>
```

Richiedere una configurazione

- È possibile specificare nel **manifest** di una app che viene richiesta una particolare configurazione
- Es.: caratteristiche dello schermo

```
<compatible-screens>
    <screen android:screenSize=["small" | "normal" | "large" | "xlarge"]
            android:screenDensity=["ldpi" | "mdpi" | "hdpi" | "xhdpi"] />
    ...
</compatible-screens>
```

- In generale, sarebbe da evitare
  - Meglio cercare di supportare tutto!

Hello, world!