Lezione 24

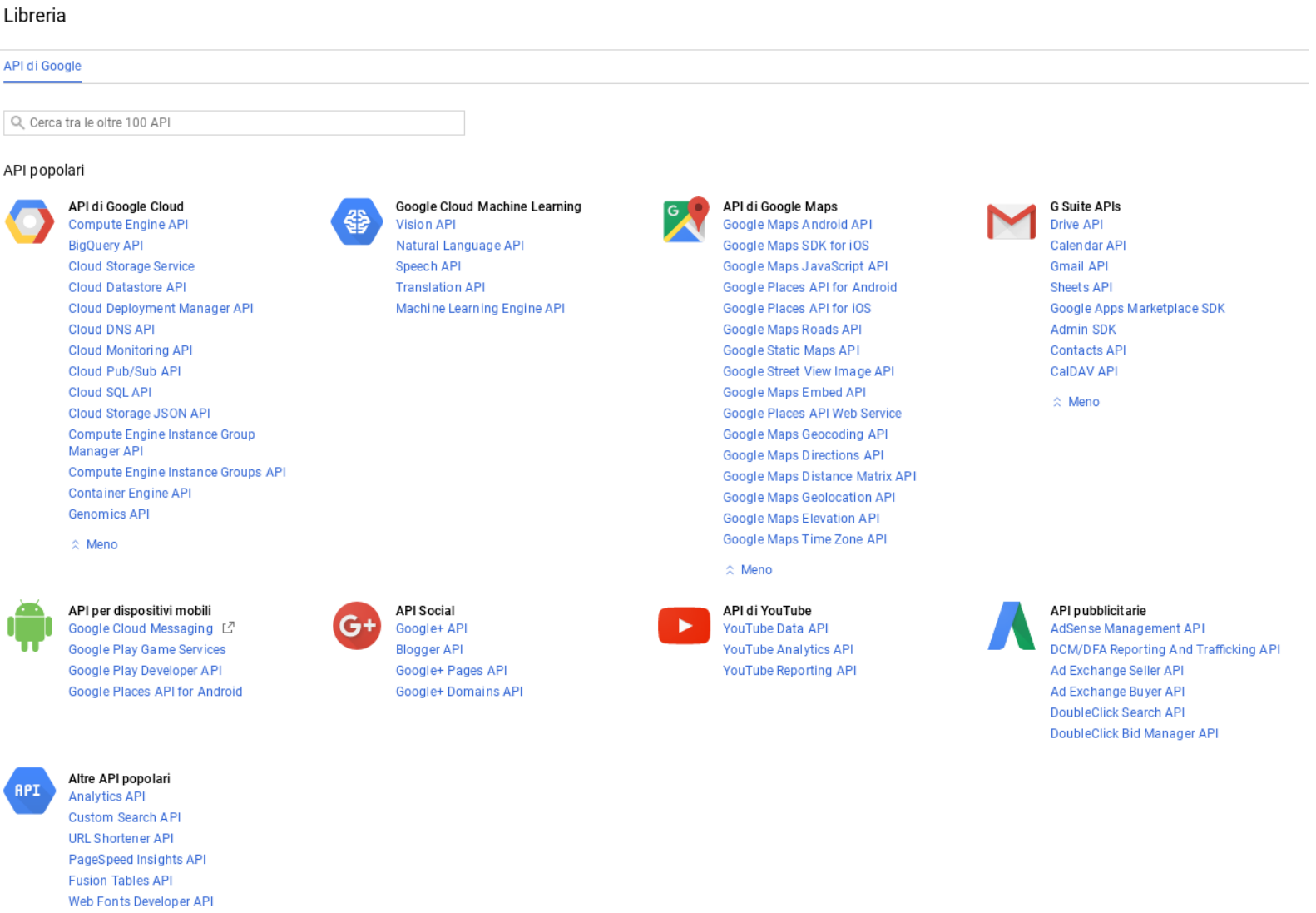
Programmazione Android

- Accesso ai servizi Google
  - Google APIs for Android
- Kotlin e Android

Google APIs

Google APIs

- Oltre 100 APIs
- Ciascuna con dozzine o centinaia di metodi
- Così tanti servizi da richiedere un motore di ricerca interno con una sezione delle API più popolari!
- Marketplace di APIs a pagamento



API Explorer

https://developers.google.com/apis-explorer

Google

Search for services, methods, and recent requests... Loading...

APIs Explorer

Services

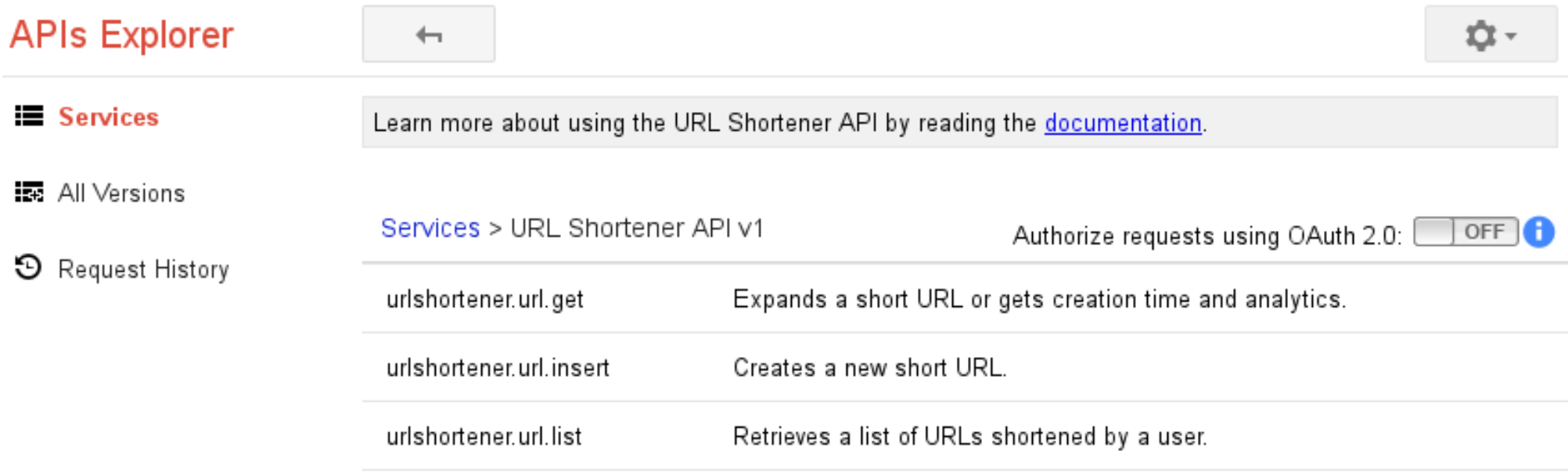
All Versions

Request History

Accelerated Mobile Pages (AMP) URL API	v1	This API contains a single method, batchGet. Call this method to retrieve the AMP URL (and equivalent AMP Cache URL) for given public URL(s).
Ad Exchange Buyer API	v1.4	Accesses your bidding-account information, submits creatives for validation, finds available direct deals, and retrieves performance reports.
Ad Exchange Buyer API II	v2beta1	Accesses the latest features for managing Ad Exchange accounts and Real-Time Bidding configurations.
Ad Exchange Seller API	v2.0	Accesses the inventory of Ad Exchange seller users and generates reports.
Admin Reports API	reports_v1	Fetches reports for the administrators of G Suite customers about the usage, collaboration, security, and risk for their users.
AdSense Host API	v4.1	Limited Availability Generates performance reports, generates ad codes, and provides publisher management capabilities for AdSense Hosts.
AdSense Management API	v1.4	Accesses AdSense publishers' inventory and generates performance reports.
APIs Discovery Service	v1	Provides information about other Google APIs, such as what APIs are available, the resource, and method details for each API.
BigQuery API	v2	A data platform for customers to create, manage, share and query data.
BigQuery Data Transfer Service API	v1	Transfers data from partner SaaS applications to Google BigQuery on a scheduled, managed basis.
Blogger API	v3	Limited Availability API for access to the data within Blogger.
Books API	v1	Searches for books and manages your Google Books library.
Calendar API	v3	Manipulates events and other calendar data.
Cloud Monitoring API	v2beta2	Accesses Google Cloud Monitoring data.
Cloud Source Repositories API	v1	Access source code repositories hosted by Google.
Cloud Spanner API	v1	Cloud Spanner is a managed, mission-critical, globally consistent and scalable relational database service.
Cloud SQL Administration API	v1beta4	Creates and configures Cloud SQL instances, which provide fully-managed MySQL databases.
Cloud Storage JSON API	v1	Stores and retrieves potentially large, immutable data objects.

API Explorer

https://developers.google.com/apis-explorer

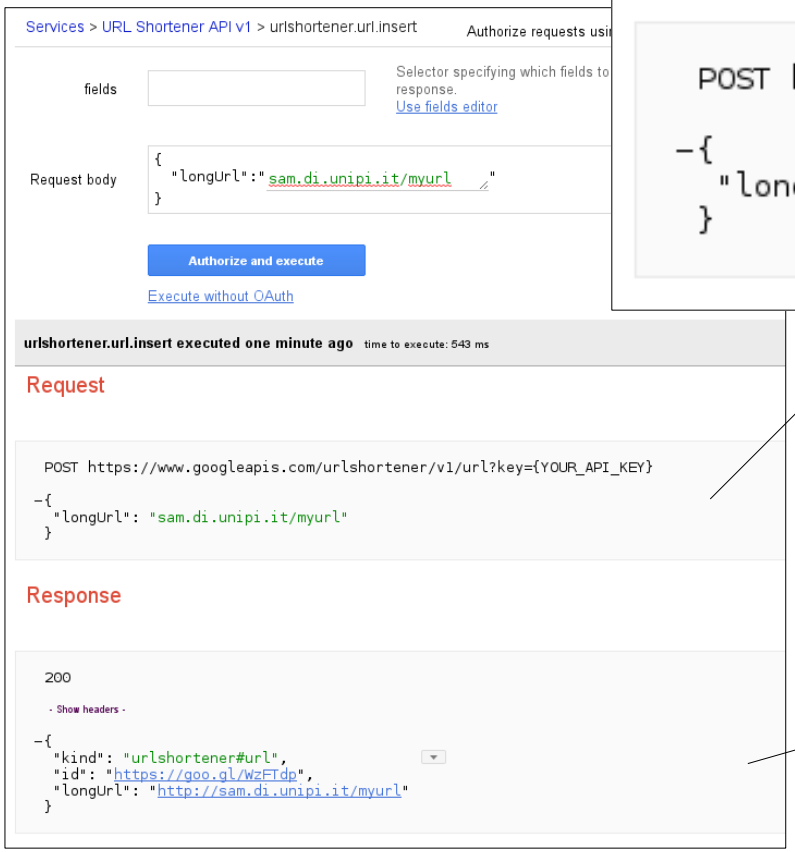


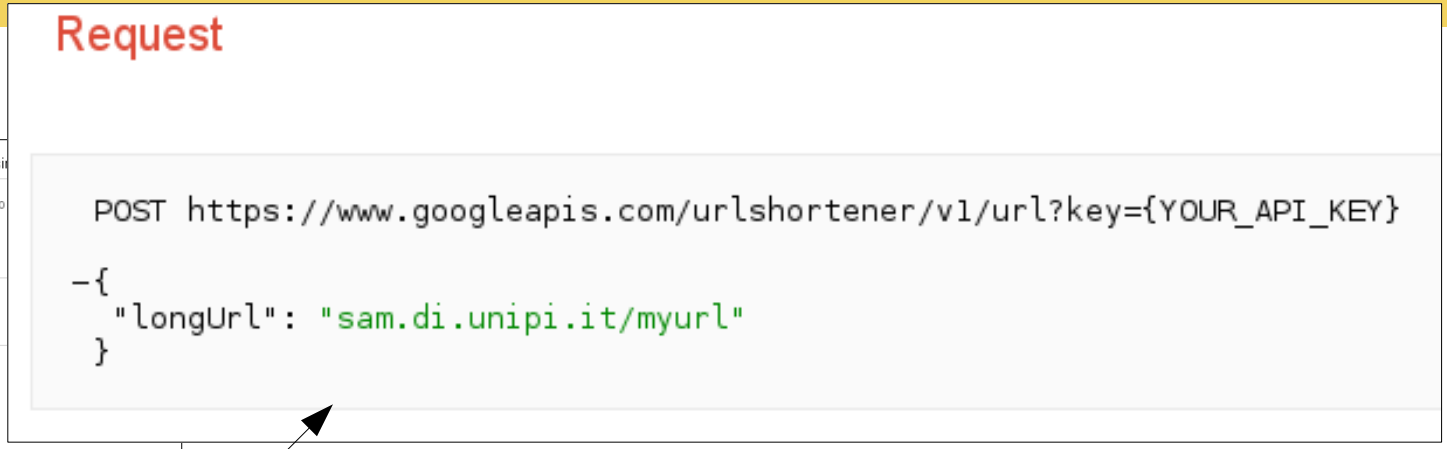
Ogni API offre un insieme di metodi che possono essere chiamati in stile REST.

Ogni chiamata incluse una **richiesta** (HttpRequest), tipicamente con un *payload* JSON che fornisce gli argomenti.

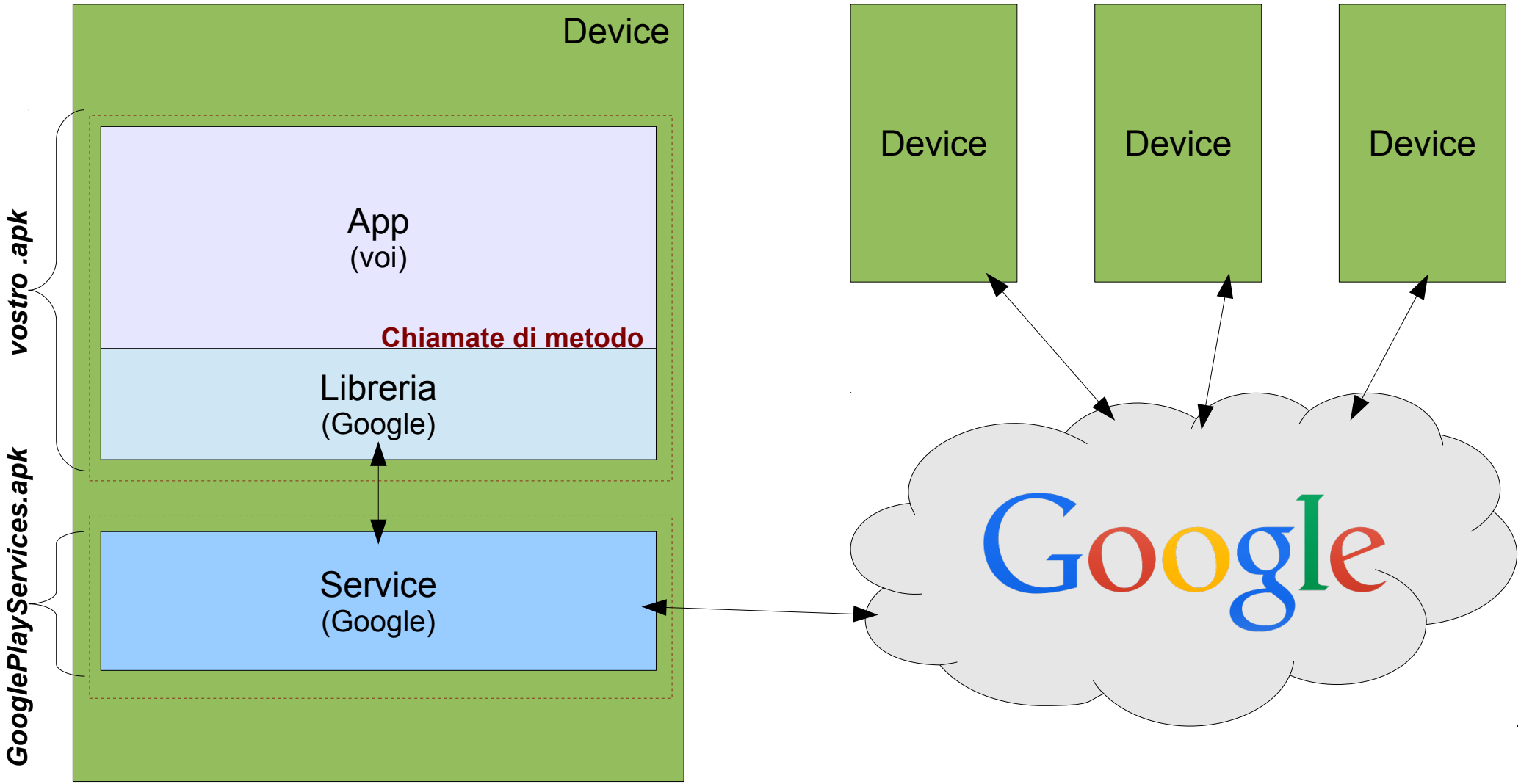
La **risposta** è una coppia <*codice*,*corpo*> in cui il *codice* è un error code HTTP (200=ok, 404=forbidden, ecc.), mentre il *corpo* è un oggetto JSON i cui campi rappresentanto il risultato della chiamata.

Esempio: Url shortener

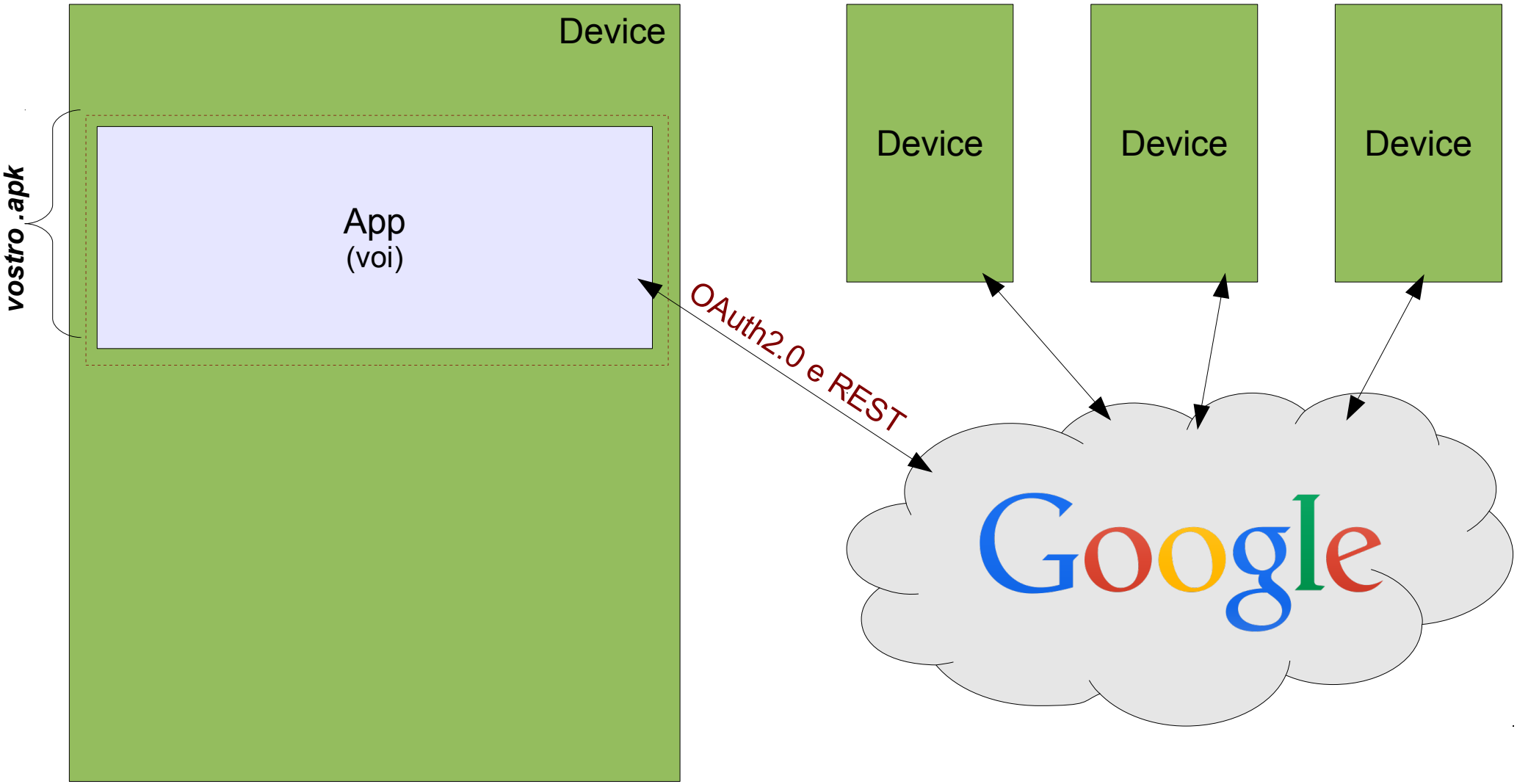




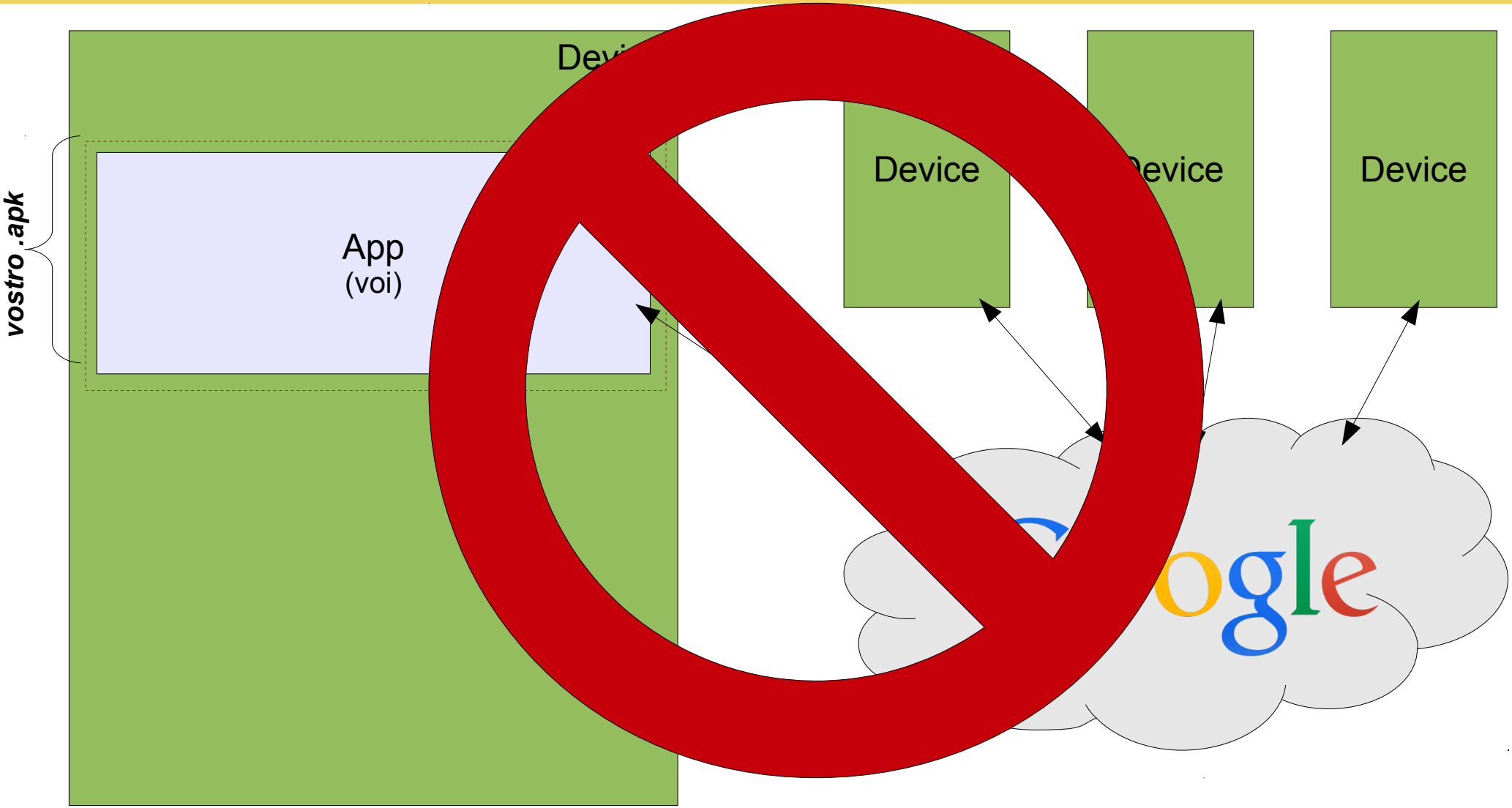
Architettura "Google APIs for Android"



Architettura alternativa



Architettura alternativa



Connessione a Google API

- Usiamo il pattern *Builder*, aggiungendo due interfacce e un gruppo di API:

```
GoogleApiClient gapi = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks() {
        public void onConnected(Bundle hint) {
            // Possiamo usare l'API
        }
        public void onConnectionSuspended(int res) {
            // Accesso all'API sospeso – ma tornerà!

        }
    })
    .addOnConnectionFailedListener(new GoogleApiClient.OnConnectionFailedListener() {
        public void onConnectionFailed(ConnectionResult res) {
                // Connession fallita. Addio mondo crudele!
        }
    })
    .addApi(Wearable.API) // Lista delle Google API a cui vogliamo accedere
    .build();
```

Spesso (ma non sempre) fatto nella onCreate() dell'activity

Connessione a GoogleAPI

- Si può anche, come al solito, fare implementare le interfacce all'Activity in questione, attivare più API insieme, ecc.
  - E magari scrivere di meno!
- Per esempio:

```
gapi = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .addApiIfAvailable(Wearable.API)
    .addApi(Drive.API)
    .addScope(Drive.SCOPE_FILE)
    .build();
```

Wear solo se disponibile.

Poi si può usare gapi.hasConnectedApi(Wearable.API) per scoprire a runtime se l'API è disponibile.

Drive sempre – altrimenti è un errore

Connessione a GoogleAPI

- L'ultimo passo è anche il più semplice!
- Per avviare la connessione alle API selezionate:

```
gapi.connect();
```

- Schemi tipici:
  - Uso solo durante l'attività
    - gapi.connect() nella onStart(), gapi.disconnect() nella onStop()
  - Uso solo in certi casi specifici
    - gapi.connect(); operazione; gapi.disconnect()
  - Uso continuo
    - gapi.connect() nella onCreate(), gapi.disconnect() nella onDestroy()

Connessione a GoogleAPI

- La gestione degli errori invece è una tragedia
  - Dopo una chiamata a connect() può darsi che…
    - Tutto bene: onConnected(hint)
    - Qualcosa storto: onConnectionFailed(res)
- In caso di fallimenti, è possibile che il sistema offra una ***risoluzione*** implicita
  - Esempio: il client non era autenticato
  - Spesso le risoluzioni necessitano di azione dell'utente
  - In ogni caso… possiamo esprimere l'*intenzione* di rimediare!

Connessione a GoogleAPI

- Esempio

```
public void onConnectionFailed(ConnectionResult res) {
    if (risincorso) {
        // Stiamo già cercando di risolvere l'errore
        return;
    } else if (res.hasResolution()) {
        try {
            risincorso = true;
            res.startResolutionForResult(this, REQUEST_RESOLVE_ERROR);
        } catch (SendIntentException e) {
            // Il tentativo di risoluzione ha causato errore. Riproviamo.
            gapi.connect();
        }
    } else {
        // Niente da fare, dialog d'errore all'utente
        risincorso = true;
        showErrorDialog(res.getErrorCode());
    }
}
```

GooglePlayServicesUtil.getErrorDialog(errorCode, this.getActivity(), REQUEST_RESOLVE_ERROR);

Connessione a GoogleAPI

- Se abbiamo avviato un tentativo di risoluzione, verrà chiamato più avanti il nostro onActivityResult() con RESULT_OK
  - via startResolutionForResult()
  - via GooglePlayServicesUtil.getErrorDialog()
- A quel punto, vale la pena di riprovare la connect()
- In caso contrario… siamo alla disperazione, errore permanente!

GoogleAPIs

- Alcune delle API disponibili:
  - Search – è Google!
  - Ads – pubblicità e simili
  - Analytics – analisi di traffico
  - AppInvite – gestione beta tester
  - Cast – Chromecast e simili
  - Auth e Identity – identità degli utenti
  - Drive – storage condiviso
  - Fit – informazioni sull'attività fisica
  - Games – aspetti sociali dei giochi
  - Location, Maps, Panorama e Nearby – geocose
  - Plus – accesso a G+
  - Wallet – pagamenti elettronici
  - Wear – dispositivi indossabili

GoogleAPIs

- Trovate un elenco completo di API a https://developers.google.com/products/
- Alcune richiedono particolari relazioni commerciali, o che registriate la vostra applicazione sulla Developer Console, o non sono rilevanti per Android

Operazioni su GoogleAPI

- Le operazioni su GoogleAPI sono per loro natura **asincrone** e **fallibili**
  - Per forza: passano da rete, sotto c'è REST
- Tutte le chiamate vengono fatte tramite metodi statici di classi di libreria corrispondenti alle API
  - Hanno come parametro `gapi`
  - In molti casi, restituiscono un PendingResult
  - Sul PendingResult si può agire in tre modi
    - Registrando una **callback** da chiamare quando il risultato è pronto
    - **Sospendendo** il thread in attesa che il risultato sia pronto
    - **Cancellando** l'operazione (e testando se è stata cancellata)

Operazioni su GoogleAPI
Callback

- Come di consueto...
  - Si chiama **setResultCallback**(ResultCallback<R> rc) sul PendingResult
  - Quando l'operazione sarà conclusa, il risultato (di tipo R) verrà passato a rc
  - R estende Result
    - Al momento, sono definite 71 sottoclassi di Result nella libreria, con i diversi risultati specifici di diverse chiamate
    - Result fornisce di suo solo getStatus(), la quale restituisce un'oggetto Status
      - Status a sua volta ha un codice di successo (successo / interrotto / fallito / cancellato), una stringa che rappresenta un messaggio di stato, e un PendingIntent

Operazioni su GoogleAPI
Callback

- Esempio:

```
private void loadFile(String filename) {
    Query q = new Query.Builder()
        .addFilter(Filters.eq(SearchableField.TITLE, filename))
        .build();
    Drive.DriveApi.query(gapi, q)
        .setResultCallback(new ResultCallback() {
            public void onResult(DriveApi.MetadataBufferResult r) {
                MetadataBuffer mdb=r.getMetaDataBuffer();
                if (mdb!=null) {
                    for (Metadata md: mdb) {
                        int size=md.getFileSize();
                        String desc=md.getDescription();
                        ...
                    }
                    mdb.release();
                }
            }
    });
}
```

Di setResultCallback() esiste anche la versione con timeout: scaduto il tempo indicato, la chiamata si considera fallita.

Operazioni su GoogleAPI
Sospensione

- Sul PendingResult si chiama la **await**()
- Il thread chiamante viene bloccato finché il risultato non è disponibile
  - Quindi: **mai** chiamare sul thread UI!
- La await() restituisce il risultato
  - Di tipo R, sottoclasse di Result
  - Lo stesso oggetto che sarebbe passato a onResult()

Di await() esiste anche la versione con timeout: scaduto il tempo indicato, la chiamata si considera fallita.

Operazioni su GoogleAPI
Cancellazione

- Su un PendingResult non ancora completato si può chiamare la **cancel**()
  - Se era stata impostata una callback, questa viene chiamata con un parametro Result
  - Se un thread era bloccato su una await(), viene risvegliato e si ritorna un Result
- In entrambi i casi, il Result conterrà come status code l'indicazione dell'interruzione
- Il metodo **isCanceled**() di PendingResult indica se il PendingResult è stato cancellato o meno

Esempio: WearAPI

- Una volta ottenuto il `gapi` per Wear.API, si può fare accesso a cinque sotto-API, ciascuna espressa da una interfaccia Java
  - Replicazione e sincronizzazione
    - **DataApi** – legge e scrive DataItem e Asset, che vengono sincronizzati automaticamente fra device
  - Invio di messaggi fra nodi di una rete
    - **MessageApi** – invio di messaggi fra nodi
    - **NodeApi** – informazioni sui nodi disponibili sulla rete
    - **ChannelApi** – creazione di canali fra nodi
    - **CapabilityApi** – scopre le caratteristiche dei nodi sulla rete

DataAPI

- L'idea di fondo della DataAPI è di fornire uno **spazio di memorizzazione** virtuale, condiviso fra più dispositivi
  - Ogni volta che uno dei dispositivi modifica un elemento, il nuovo elemento viene replicato (sincronizzato) sugli altri dispositivi
    - Opzionalmente, l'applicazione viene notificata
  - Il sistema gestisce caching, buffering, trasmissione, ritrasmissione, perdite temporanee di connessione, ecc.
    - Approccio “eventually consistent”, largamente trasparente

DataAPI

DataItem A

DataItem B

Asset B

DataItem A

DataItem B

Asset B

DataItem A

DataItem B

Asset B

Da GooglePlayServices 7.3 (Maggio 2015), si possono connettere più Wearable allo stesso Mobile

DataItem

- Un **DataItem** è una unità di sincronizzazione
  - Immaginate: un file nello storage condiviso
- Dotato di:
  - Un **nome** simbolico (stringa nel formato dei pathname assoluti UNIX: “/voti/conteggio”)
  - Un **contenuto** arbitrario (array di byte)
    - Limite massimo: 100Kb
    - Può essere gestito a mano, oppure creato tramite una **DataMap** che lo incapsula in un bundle chiavi-valori

DataItem
Approccio diretto

- Se volete gestire il contenuto a mano…
- Creazione/scrittura
  - pdr=PutDataRequest.create(path).setData(buffer)
  - pr=DataApi.putDataItem(gapi,pdr)
  - pr sarà un PendingResult<DataApi.DataItemResult>
- Un DataItem è identificato da una URI
  - Accessibile con DataItem.getUri()
  - wear://*nodo*/*path* o wear:/*path*

Copia su un nodo specifico

Il dato in sé, su tutti i nodi

DataItem
Approccio diretto

- Per leggere il valore corrente di un DataItem
  - pr = DataApi.getDataItem(gapi, uri) oppure
    pr = DataApi.getDataItems(gapi)
  - pr sarà un PendingResult<DataItemBuffer>
  - Potete usare getCount() e get(idx) sul DataItemBuffer per enumerare i DataItem
  - Sul DataItem possiamo finalmente invocare getData() per recuperare il buffer di byte memorizzato

DataItem
Approccio con DataMap

- Per evitare di gestire a mano il buffer di dati, si può passare da una *ulteriore* classe detta DataMap
- pdmr = PutDataMapRequest.create(path);
- dm = pdmr.getDataMap();
- dm.put*Tipo*(chiave, valore); …
- pdr = pdmr.asPutDataRequest();
- Da qui si procede come prima per le PutDataRequest

DataItem
Approccio con DataMap

- Esempio: condividere nome ed età della nonna

```
private void condividi(String nome, int età) {
    PutDataMapRequest pdmr = PutDataMapRequest.create("/nonna");
    DataMap dm = pdrm.getDataMap();
    dm.putString(K_NOME, nome);
    dm.putInt(K_ETA, età);
    PutDataRequest pdr = pdmr.asPutDataRequest();
    PendingResult<DataApi.DataItemResult> pr =
                Wearable.DataApi.putDataItem(gapi, pdr);
}
```

- pr.await() (o una callback) per essere avvisati del termine dell'operazione
- Ma la sincronizzazione potrebbe avvenire dopo!

DataItem
Approccio con DataMap

- Per leggere il valore corrente
  - Si recupera il DataItem come visto in precedenza
  - Si ottiene il DataMap corrispondente con
    ```
    DataMap dm = DataMapItem.fromDataItem(di).getDataMap();
    ```
  - Da dm si estraggono i dati con i soliti metodi “stile Bundle”
    - dm.get*Tipo*(chiave)
- DataMap offre anche metodi per convertire da Bundle a DataMap e viceversa

Gli Asset

- A un DataItem può essere associato zero o più **Asset**
- Blocchi di dati di grandi dimensioni
  - Non hanno il limite di 100Kb
- Vengono trasferiti insieme ai DataItem
  - Però il sistema adotta politiche di caching più aggressive per evitare di sovraccaricare la connessione BlueTooth
- Acceduti tramite file descriptor

Gli Asset

- Per costruire un Asset, si usa uno dei vari metodi statici create...() della classe Asset
  - createFromBytes(byte[] buffer)
  - createFromFd(ParcelFileDescriptor fd)
  - createFromRef(String digest)
  - createFromUri(Uri uri)
- Ottenuto l'Asset, lo si aggiunge al DataItem con una chiamata a putAsset(nome, asset)
  - Nome sarà il nome simbolico dell'asset dentro il DataItem

Gli Asset
Esempio: un'immagine

- Approccio diretto

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.avatar);
ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
bitmap.compress(Bitmap.CompressFormat.PNG, 100, byteStream);
Asset asset = Asset.createFromBytes(byteStream.toByteArray());
PutDataRequest pdr = PutDataRequest.create("/poster");
pdr.putAsset("locandina", asset);
PendingResult<DataApi.DataItemResult> pr = Wearable.DataApi.putDataItem(gapi, pdr);
```

- Approccio con DataMap

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.avatar);
ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
bitmap.compress(Bitmap.CompressFormat.PNG, 100, byteStream);
Asset asset = Asset.createFromBytes(byteStream.toByteArray());
PutDataMapRequest pdmr = PutDataMapRequest.create("/poster");
pdmr.getDataMap().putAsset("locandina", asset)
PutDataRequest pdr = pdmr.asPutDataRequest();
PendingResult<DataApi.DataItemResult> pr = Wearable.DataApi.putDataItem(gapi, pdr);
```

Gli Asset

- Una volta recuperato il DataItem, si possono estrarre gli Asset con getAsset()
  - Chiamato su DataItem nel caso manuale
    - Con il nome appropriato
  - Chiamato su DataMap nel caso si usi DataMap
    - Con la chiave appropriata
- Finalmente, dall'Asset così recuperato possiamo estrarre i dati chiamando
  - getUri() – sotto forma di URI
  - getFd() – sotto forma di ParcelFileDescriptor

DataAPI: notifiche

- Naturalmente, nella maggior parte dei casi vorremo essere **notificati** quando una versione aggiornata di un DataItem è disponibile!
  - Perché un altro nodo ha modificato la sua copia
- Come al solito, possiamo registrare un Listener che verrà invocato quando necessario
  - L'interfaccia da implementare è DataApi.DataListener
  - Pattern solito: la si può fare implementare all'Activity

DataAPI: notifiche

- Solito schema
  - DataApi.registerListener(gapi,listener)
  - DataApi.removeListener(gapi,listener)
- Il listener ha un solo metodo:
  - public void onDataChanged(DataEventBuffer evb)
- evb offre i metodi getCount() e get(*idx*) per scorrere una sequenza di DataEvent de
- Finalmente, de offre
  - getDataItem() che restituisce il DataItem modificato
  - getType() che descrive il tipo di modifica (update o cancellazione)

DataAPI: ogni cosa al suo posto!

- Tipicamente, si fa tutto nella Activity:
  - onCreate() → costruzione di gapi
  - onResume() → gapi.connect()
  - onConnected() → addListener(gapi, this)
  - onPause() → removeListener(gapi,this)
gapi.disconnect()
  - onDataChanged() → lettura DataItem aggiornati
- La scrittura di DataItem invece dipende strettamente dalla logica dell'app

DataAPI: esempio

```
public class TestDataAPI extends Activity implements
        DataApi.DataListener,
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener {

    private static final String K_NOME = "nome";
    private static final String K_ETA = "eta";
    private GoogleApiClient gapi;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        gapi = new GoogleApiClient.Builder(this)
                .addApi(Wearable.API)
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .build();
    }
```

DataAPI: esempio

```
protected void onResume() {
    super.onStart();
    gapi.connect();
}

public void onConnected(Bundle hint) {
    Wearable.DataApi.addListener(gapi, this);
}

protected void onPause() {
    super.onPause();
    Wearable.DataApi.removeListener(gapi, this);
    gapi.disconnect();
}
```

DataAPI: esempio

```
    public void onDataChanged(DataEventBuffer de) {
        for (DataEvent e : de) {
            if (e.getType() == DataEvent.TYPE_CHANGED) {
                // DataItem aggiornato
                DataItem di = e.getDataItem();
                if (di.getUri().getPath().compareTo("/nonna") == 0) {
                    DataMap dm = DataMapItem.fromDataItem(di).getDataMap();
                    nome = dm.getString(K_NOME);
                    età = dm.getInt(K_ETA);
                    // qui possiamo aggiornare la nonna
                }
            } else if (e.getType() == DataEvent.TYPE_DELETED) {
                // la nonna non c'è più...
            }
        }
    }

    // onConnectionSuspended(), onConnectionFailed() come già visto
}
```

DataAPI: notifiche via Service

- In alternativa al sistema dei callback, possiamo estendere **WearableListenerService**
- Si tratta di un servizio di tipo bound
  - A cui si collega sia la vostra applicazione, sia il processo che esegue GooglePlayServices
- Il Service espone vari metodi che potete (dovete) sovrascrivere nella sottoclasse
  - Per il nostro caso:
    onDataChanged(DataEventBuffer de)

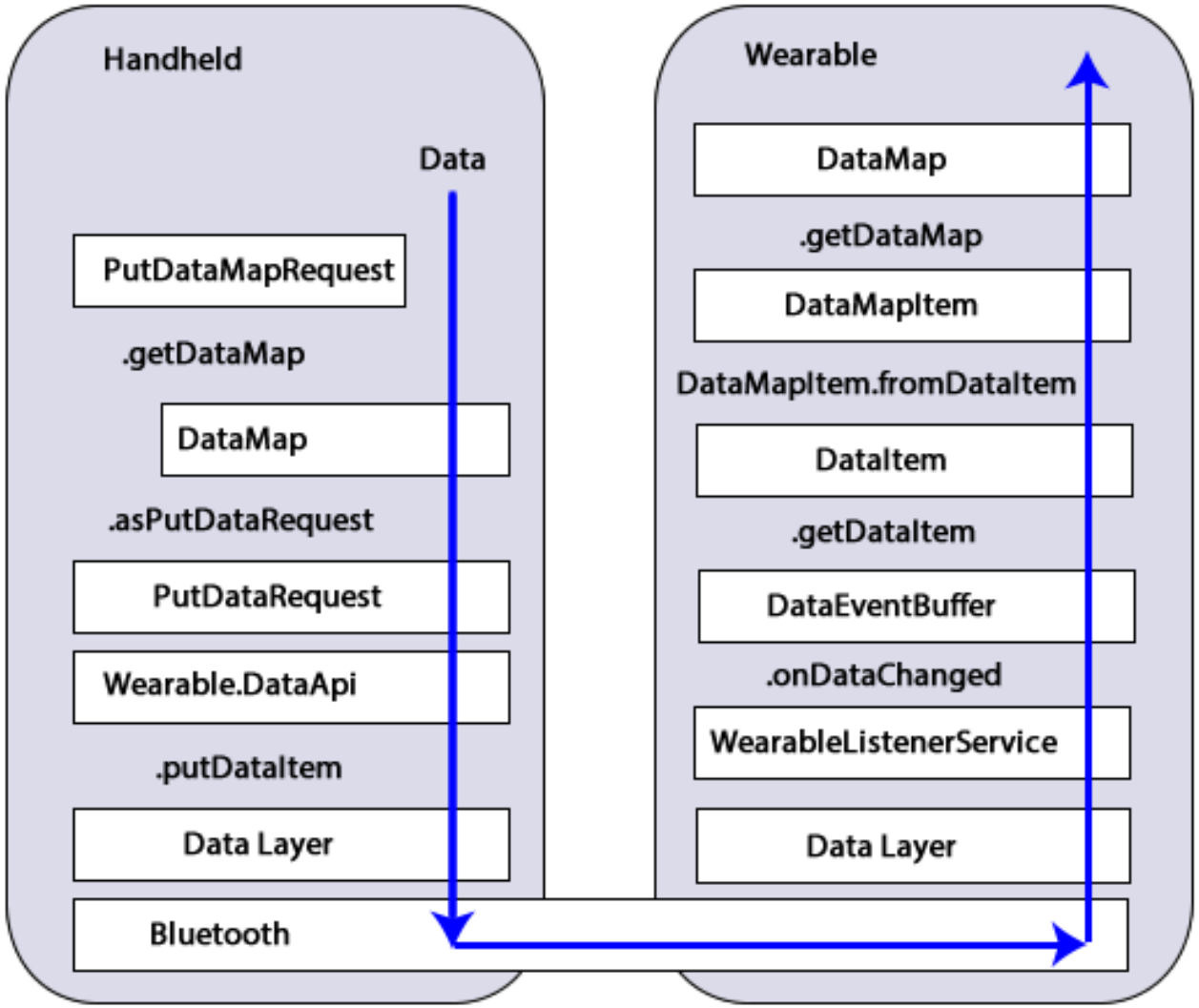
DataAPI: notifiche via Service

- Naturalmente, il Service andrà dichiarato in AndroidManifest.xml
- Deve dichiarare un intent filter che consente a GooglePlayServices di “trovarlo”

```
<service android:name="IlMioWLS">
  <intent-filter>
      <action android:name="com.google.android.gms.wearable.BIND_LISTENER" />
  </intent-filter>
</service>
```

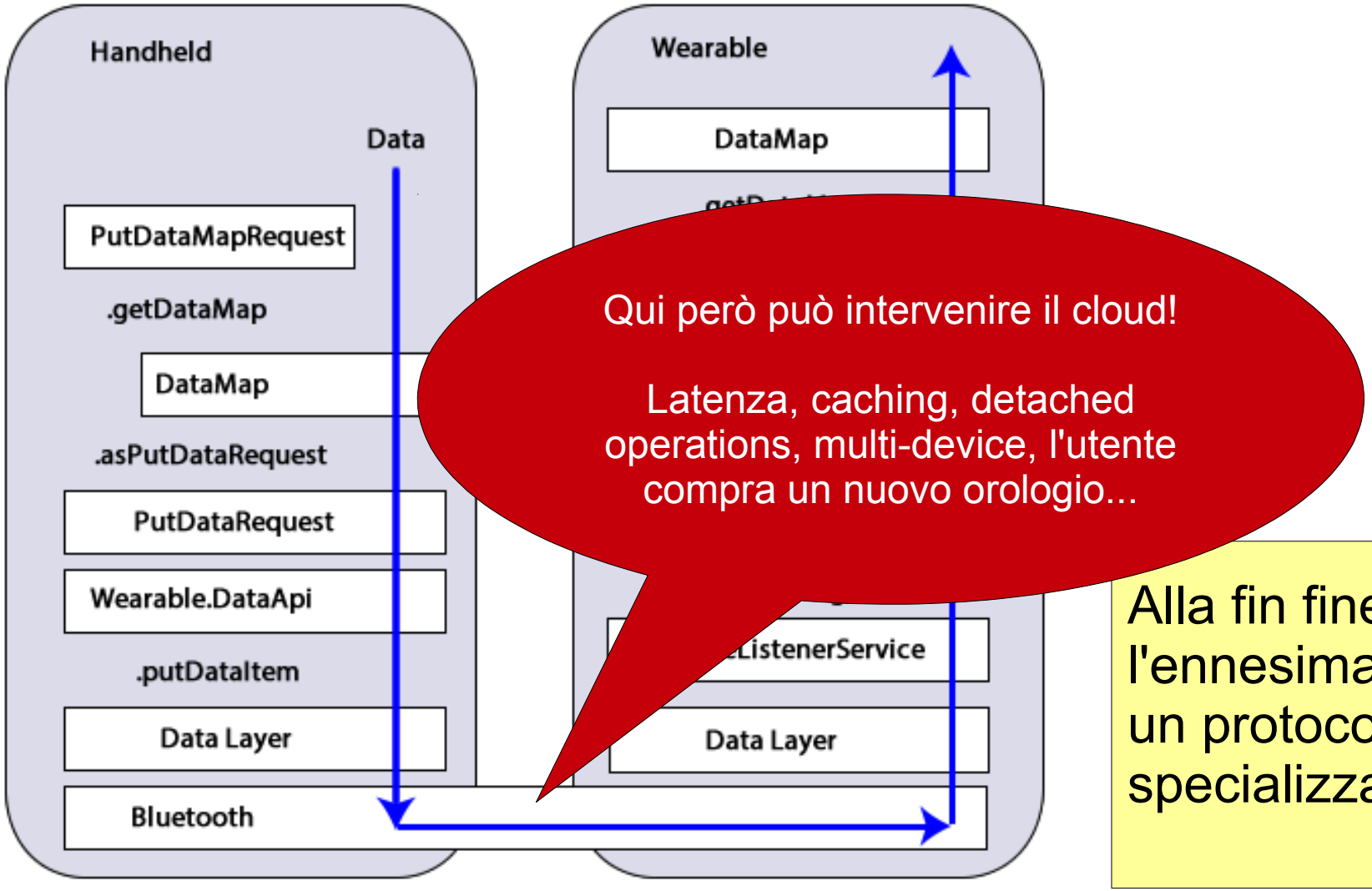
- Nota bene: il ciclo di vita del Service è pur sempre gestito dal sistema
  - Sarà vivo quando ci sono eventi da segnalare, ma Android potrebbe chiuderlo (se non bindato) negli intervalli

DataAPI riassunto



Alla fin fine, è l'ennesima versione di un protocollo RPC, specializzato per i dati

DataAPI riassunto



Qui però può intervenire il cloud!

Latenza, caching, detached operations, multi-device, l'utente compra un nuovo orologio...

Alla fin fine, è l'ennesima versione di un protocollo RPC, specializzato per i dati

Kotlin & Android

Kotlin & Android

- Una settimana fa, Google ha presentato il supporto a **Kotlin** per Android
  - Ancora in fase di test, disponibile solo su Android Studio Canary (early preview)
  - Motivazioni multiple:
    - Linguaggio più moderno e più comodo rispetto a Java
    - Diminuire ulteriormente la dipendenza di Android da tecnologie proprietarie o comunque non controllate da Google (e ridurre il rischio-cause)
      - Come per Dalvik / ART al posto della JVM

Kotlin: sintassi base

- Kotlin è **quasi** una diversa sintassi per Java
  - Notazione dei tipi “stile Pascal”
  - int send(String act) { … } → **fun** send(act: String): Int { … }
  - Tipi nullabili: String? può essere null, String no, String non si sa
    - if (p!=null) p.m(x) → p?.m(x)
    - r=(p!=null?p.f:dflt) → p?.f ?: dflt
    - x=a?.b?.c?.d
  - Distingue le variabili “vere” dalle costanti letterali
    - **var** x:Int – x è una variabile
    - **val** x:Int – x può essere assegnata solo 1 volta
  - Stringhe template:
    - “x vale $x” oppure “il conto fa ${f(x)+3}”

Kotlin: sintassi base

- Kotlin è **quasi** una diversa sintassi per Java
  - Assai migliore **inferenza di tipi**
    - Riduce la necessità di cast, migliora i controlli del compilatore
  - **Data class** per oggetti semplici
    - data class Punti(val giocatore:String, var punti:Int)
    - Crea in automatico costruttori, getter, setter, toString(), ecc.
  - Singleton nel linguaggio
    - object me { val nome="Vincenzo" }

Kotlin: lambda

- Il supporto alle lambda-expressions semplifica la scrittura di listener, runnable, ecc.
  - x -> x*2, oppure x,y -> { f(x); g(x,y); }
  - val sum = { x: Int, y: Int -> x + y }
- Un blocco di codice fra graffe è implicitamente una funzione con un solo parametro *it*
  - list.map { 2*it } // si possono omettere le ()
- Grazie alla type inference, spesso i tipi possono essere omessi

Kotlin: classi sealed e extension

- Le classi in Kotlin sono per default **sealed**
- Le classi destinate a essere sottoclassate devono essere dichiarate **open**
  - open class Padre …
  - class Figlio : Padre …
- Idem per l'override di metodi (e di proprietà)
  - open fun m() {} // nel Padre
  - override fun m() {} // nel Figlio
- Rationale: evitare l'overriding accidentale

Kotlin: classi sealed e extension

- È possibile **aggiungere metodi e proprietà** a classi già esistenti (extension)
  - fun Padre.n(…) {…}
  - In tutto lo scope di visibilità di una simile dichiarazione, gli oggetti di classe Padre avranno un metodo in più (n) rispetto a quelli definiti dalla dichiarazione della classe

Kotlin: delegation

- Un pattern tipico alternativo all'ereditarietà è la **delega**
  - **public class A {…}**
    public class B {
    A a = new A();
    public void m1(args) { a.m1(args); }
    public int m2(args) { a.m2(args); }
    …
    }
  - **class B(a: A): A by a**

Kotlin: coroutines

- Le coroutines sono un modo alternativo (rispetto ai thread) per implementare la concorrenza
- Si definiscono funzioni o lambda con la keyword **suspend** che possono interrompere l'esecuzione e passare il controllo allo scheduler delle coroutine
- Il qualificatore **async** identifica un blocco di codice che può essere eseguito in maniera asincrona
- Meccanismo simile alle async/await in Javascript

Kotlin: Androidismi

- Il compilatore Kotlin in Android Studio ha un po' di plug-in pensati per facilitare la vita al programmatore Android
- Esempi
  - Estrazione degli ID delle View in un layout (*synthetic properties)*
    - String te = ((TextEdit)findViewById(R.id.textedit)).getText();
    - import kotlinx.android.synthetic.main.<*layout*>.*
      var te=textedit.getText();
  - Trasformazione sintattica di codice (o intere classi) da Java a Kotlin
    - … che a volte funziona anche!

Kotlin: Andoidismi

- Es: accesso a SQLite via annotazioni

```
@Table(name="users", database = AppDatabase::class)
class User: BaseModel() {

    @PrimaryKey(autoincrement = true)
    @Column(name = "id")
    var id: Long = 0

    @Column
    var name: String? = null
}
```

O anche:

```
@Table(database = KotlinDatabase::class)
data class User(@PrimaryKey var id: Long = 0, @Column var name: String? = null)
```

Kotlin: Java interop

- Il codice Kotlin è compilato sulla JVM
- Può interagire con classi e librerie Java in maniera trasparente
  - Chiamare Java da Kotlin
  - Chiamare Kotlin da Java
  - Accedere ai campi di oggetti in entrambe le direzioni

Sommario

- In definitiva, Kotlin:
  - È un linguaggio moderno
  - Orientato alla produttività
  - Focalizzato sulla prevenzione degli errori
  - Con customizzazioni per Android