

Communication Internet

Fabien Dutoit

SYM – Systèmes mobiles

Particularités

- *La communication avec un serveur d'applications passe le plus souvent par HTTP*
- *HTTP encourage un modèle de communications **synchrone** (requête-réponse)*
- *Ce modèle n'est pas toujours adapté à l'application, surtout dans les cas différents d'une simple interrogation du serveur*

Particularités

- *Bloquer une application en raison d'une transaction est une pratique à décourager*
- *Dans le cas particulier d'applications mobiles, l'environnement dynamique rend cette pratique encore plus défavorable*
- *Il faut donc clairement isoler ce genre de transactions de l'application*

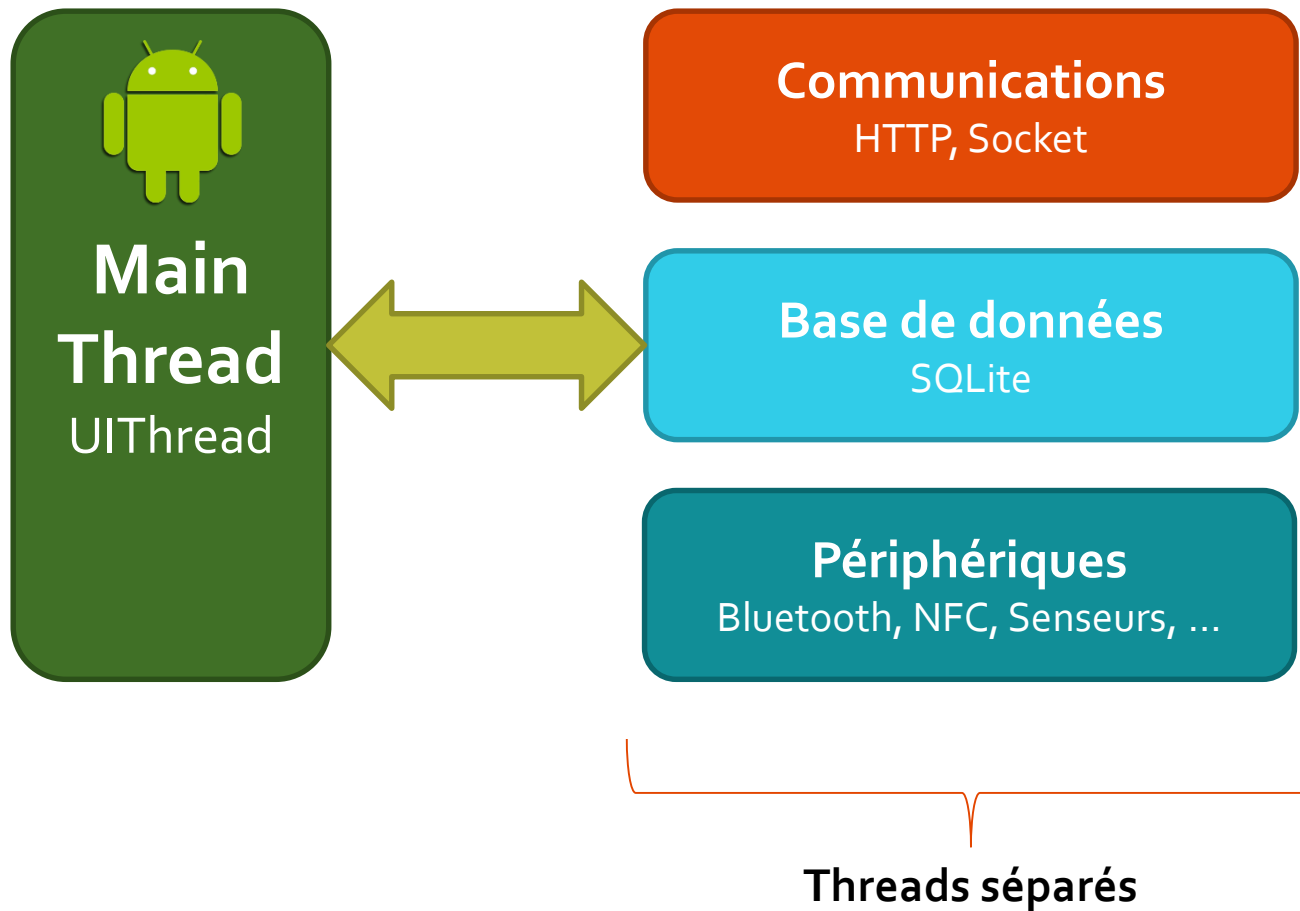
Particularités

- *Internet n'implémente pas de couche de session générique (couche 5)*
- *Si un service de transfert fiable (RTS) est requis, l'application devra l'implémenter au-dessus de HTTP*
- *Un service de ce type comporte la notion d'indépendance du cycle de vie de l'application*

Particularités

- *Chaque plateforme implémente son propre modèle de délégation; ainsi, il n'est pas sans autre possible d'accéder aux threads de communication sous iOS*
- *Dans le cas d'Android, la plateforme offre de nombreuses possibilités rendant la programmation plus flexible, mais aussi plus complexe*

Application Android



Concurrence

- *Les threads délégués à la gestion de la périphérie doivent communiquer avec le thread principal (UIThread)*
- *Pour des raisons évidentes de concurrence, l'interaction directe n'est pas possible*
- *Il faut donc introduire des mécanismes de communication avec le thread principal*
- *Le modèle de programmation devient asynchrone*

Listeners

- *L'interface `EventListener` de `java.util` propose une formalisation basique du pattern «Listener»*
- *On peut la dériver et définir les méthodes nécessaires :*

```
public interface CommunicationEventListener extends EventListener {  
    public boolean handleServerResponse(String response);  
}
```


Listeners

- *Le modèle ne définit pas le mode de propagation de l'évènement lorsqu'il y a plusieurs Listeners. Nous allons définir notre propre mode de propagation*
- *En revanche, il suggère que l'on puisse ajouter ou retirer des Listeners de manière dynamique, au besoin*

Listeners

// The event listener

```
private List<CommunicationEventListener> theListeners = new LinkedList<>();
```

*// The LISTENER pattern requires that a listener registers himself as such
// to the service. In such a simple example, it could be an implicit reference;
// but let's do things according to good practices !
// The removeCommunicationEventListener() method, although prescribed
// by the pattern hasn't been implemented. It's straightforward !*

```
public void addCommunicationEventListener(CommunicationEventListener listener){  
    if(!theListeners.contains(listener))  
        theListeners.add(listener);  
}
```

Listeners

Lors de l'apparition de l'évènement, il suffit d'invoquer tous les listeners successivement avec les paramètres de l'évènement :

```
for (CommunicationEventListener cel : theListeners) {  
    if (cel.handleServerResponse(answer)) break;  
}
```

On notera que la propagation s'arrête au premier listener retournant true

Implémentation

```
public boolean handleServerResponse(final String response) {  
    this.runOnUiThread(new Runnable() {  
        @Override  
        public void run() {  
            // Update UI widget  
            responseText.setText(response);  
        }  
    });  
    /*  
     * Message has been handled...  
     */  
    return true;  
}
```

L'activité

Listeners

- *Si l'on souhaite n'avoir qu'un seul Listener on nommera la méthode d'enregistrement: `set<XXX>Listener(Listener l)`*
- *On a vu tout à l'heure que le premier listener à valider l'événement stoppait la propagation, selon l'ordre d'enregistrement. On peut imaginer des scénarios différents, par exemple avec des priorités*
- *On trouvera aussi des cas où tous les Listeners enregistrés doivent être notifiés*

Message passing

- *Une méthode alternative est le passage de message; dans le cadre Android, ceci se réalise à l'aide de la classe Handler*
- *Un handler est associé au thread dans lequel il a été créé, et peut recevoir des messages de la part d'autres threads*
- *Il faut en général redéfinir la méthode handleMessage() du Handler*

Implémentation

```
/*
 * Alternate way to interact with a view :
 */
private Handler handler;
private CheckBox answerReceived;

answerReceived = findViewById(R.id.resp_sent);

//A faire dans l'activité ! Le Handler doit être instancié dans l'UIThread
handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        Bundle b = msg.getData();
        boolean flag = b.getBoolean("DATA_RECEIVED");
        answerReceived.setChecked(flag); //GUI
    }
};
```

Utilisation

```
/*  
 * On pourrait aussi écrire new Message()  
 * mais obtainMessage() réutilise un ancien message  
 * ce qui est plus efficace  
 */
```

```
Message msg = handler.obtainMessage();
```

```
Bundle b = new Bundle();  
b.putBoolean("DATA_RECEIVED", true);  
b.putString("DATA", response);  
msg.setData(b);  
handler.sendMessage(msg);
```


Remarques – Memory Leak

- *Avec le code vu précédemment, on crée une sous-classe anonyme de Handler depuis une instance de l'activité*

```
handler = new Handler() {  
    @Override  
    public void handleMessage(Message msg) {  
        ...  
    }  
};
```

- *L'instance de l'activité ne pourra pas être «garbage collectée» avant que tous les threads référençant ce Handler ne soient terminés*

Memory Leak – Solution

- *On doit créer une sous-classe **static** avec une **WeakReference** vers l'instance de l'activité*

```
public class SampleActivity extends Activity {  
    private static class MyHandler extends Handler {  
        private final WeakReference<SampleActivity> mActivity;  
  
        public MyHandler(SampleActivity activity) {  
            mActivity = new WeakReference<>(activity);  
        }  
  
        @Override  
        public void handleMessage(Message msg) {  
            SampleActivity activity = mActivity.get();  
            if (activity != null) {  
                // traitement du message  
            }  
        }  
    }  
    // code normal de l'activité  
}
```

Notifications

- *Un système complètement asynchrone fonctionne également si la réponse parvient après la terminaison de l'application*
- *Android définit le service de notifications à cet effet*
- *Une notification apparaît comme une icône s'affichant au sommet de l'écran et que l'utilisateur peut demander à consulter*

Utilisation

```
//we display a notification
private void notifyMessageReceived(final String answer) {

    NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, "Channel Id");
    mBuilder.setSmallIcon(android.R.drawable.stat_notify_sync_noanim); //icone
    mBuilder.setContentTitle("My notification"); //titre
    mBuilder.setContentText(answer); //message
    mBuilder.setAutoCancel(true); //se ferme automatiquement au premier clic

    Intent resultIntent = new Intent(this, MonActivite.class); //on souhaite ouvrir notre activité
    //on veut remettre notre application dans une état d'utilisation correct
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(MonActivite.class); // utilise l'activité parent du manifest /!
    stackBuilder.addNextIntent(resultIntent);
    PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(
        0, PendingIntent.FLAG_UPDATE_CURRENT
    );
    mBuilder.setContentIntent(resultPendingIntent);

    NotificationManager mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    int mId = 123; // mId allows you to update the notification later on
    mNotificationManager.notify(mId, mBuilder.build());
}
```

Utilisation

//we display a notification

```
private void notifyMessageReceived(final String answer) {
```

Apparence

```
    NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, "Channel Id");  
    mBuilder.setSmallIcon(android.R.drawable.stat_notify_sync_noanim); //icone  
    mBuilder.setContentTitle("My notification"); //titre  
    mBuilder.setContentText(answer); //message  
    mBuilder.setAutoCancel(true); //se ferme automatiquement au premier clic
```

```
    Intent resultIntent = new Intent(this, MonActivite.class); //on souhaite ouvrir notre activité  
    //on veut remettre notre application dans une état d'utilisation correct
```

```
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
```

```
    stackBuilder.addParentStack(MonActivite.class); // utilise l'activité parent du manifest /!\
```

```
    stackBuilder.addNextIntent(resultIntent);
```

```
    PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(  
        0, PendingIntent.FLAG_UPDATE_CURRENT
```

```
    );
```

```
    mBuilder.setContentIntent(resultPendingIntent);
```

```
    NotificationManager mNotificationManager =
```

```
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
```

```
    int mId = 123; // mId allows you to update the notification later on
```

```
    mNotificationManager.notify(mId, mBuilder.build());
```

```
}
```

Utilisation

```
//we display a notification
private void notifyMessageReceived(final String answer) {

    NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, "Channel Id");
    mBuilder.setSmallIcon(android.R.drawable.stat_notify_sync_noanim); //icone
    mBuilder.setContentTitle("My notification"); //titre
    mBuilder.setContentText(answer); //message
    mBuilder.setAutoCancel(true); //se ferme automatiquement au premier clic

    Intent resultIntent = new Intent(this, MonActivite.class); //on souhaite ouvrir notre activité
    //on veut remettre notre application dans une état d'utilisation correct
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(MonActivite.class); // utilise l'activité parent du manifest !
    stackBuilder.addNextIntent(resultIntent);
    PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(
        0, PendingIntent.FLAG_UPDATE_CURRENT
    );
    mBuilder.setContentIntent(resultPendingIntent);

    NotificationManager mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    int mId = 123; // mId allows you to update the notification later on
    mNotificationManager.notify(mId, mBuilder.build());
}
```

Configuration de l'Intent permettant d'ouvrir
notre application pour traiter la notification

Utilisation

```
//we display a notification
private void notifyMessageReceived(final String answer) {

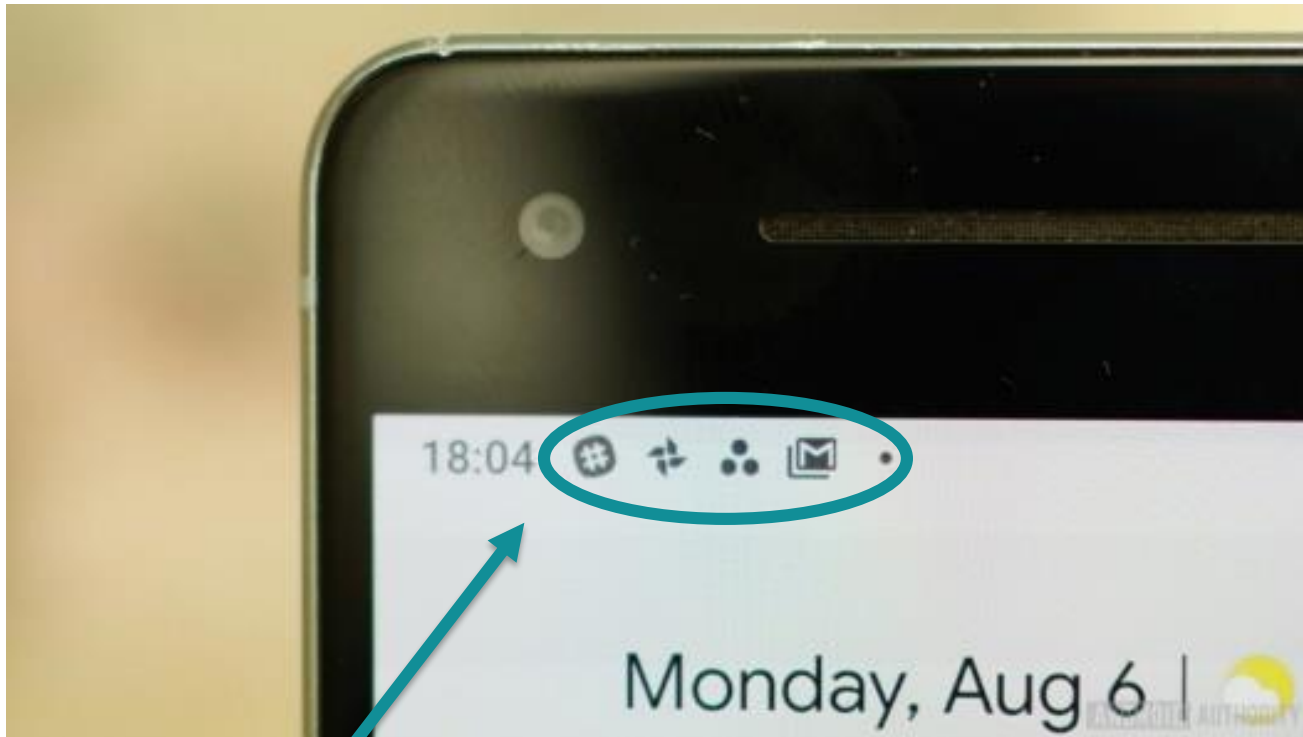
    NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, "Channel Id");
    mBuilder.setSmallIcon(android.R.drawable.stat_notify_sync_noanim); //icone
    mBuilder.setContentTitle("My notification"); //titre
    mBuilder.setContentText(answer); //message
    mBuilder.setAutoCancel(true); //se ferme automatiquement au premier clic

    Intent resultIntent = new Intent(this, MonActivite.class); //on souhaite ouvrir notre activité
    //on veut remettre notre application dans une état d'utilisation correct
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(MonActivite.class); // utilise l'activité parent du manifest /!\
    stackBuilder.addNextIntent(resultIntent);
    PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(
        0, PendingIntent.FLAG_UPDATE_CURRENT
    );
    mBuilder.setContentIntent(resultPendingIntent);

    NotificationManager mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    int mId = 123; // mId allows you to update the notification later on
    mNotificationManager.notify(mId, mBuilder.build());
}
```

Affichage

Effet visuel

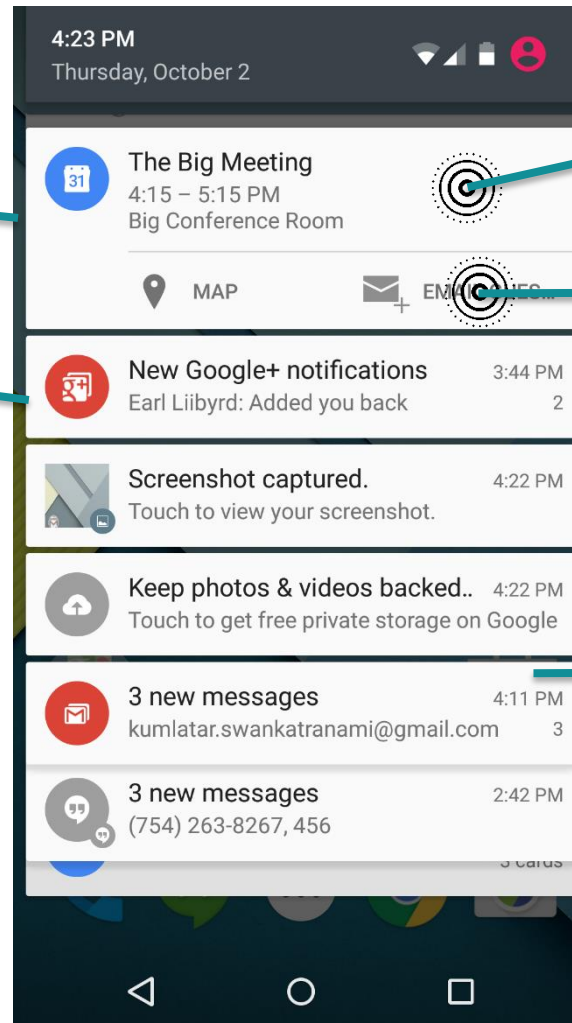


Notifications

Effet visuel

Notification étendue

Notification réduite



Action
Intent par défaut

Actions spécifiques
Différents *Intents*

Les notifications
similaires sont
regroupées

Sérialisation

- *Transmettre des objets à travers une connexion implique la sérialisation des objets*
- *En général seuls des objets implémentant l'interface `Serializable` ou l'interface `Externalizable` peuvent être transmis sur une connexion*
- *L'interface `Parcelable` d'Android n'est pas équivalente, bien qu'elle corresponde à un objectif similaire*

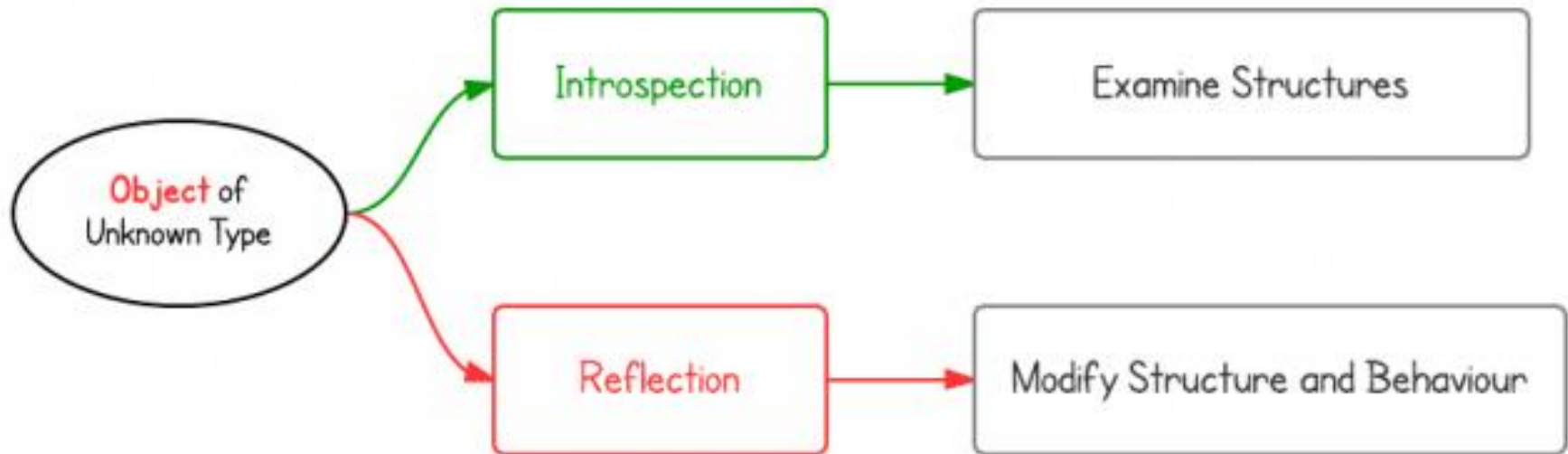
Sérialisation

- *Si on utilise une connexion HTTP, on va généralement sérialiser les objets sous forme textuelle. Il existe principalement deux technologies permettant de remplir cet objectif :*
 - *XML : Verbeux, lisible, partiellement auto-documenté, peut être validé par des outils tels que XSchema ou DTD*
 - *JSON : Compact, concis, ne peut pas encore être validé automatiquement/nativement*

Sérialisation

- *La notion de sérialisation est liée à la capacité d'un environnement d'exécution de se livrer à l'introspection de code. C'est pourquoi le langage Java offre plus de facilités que, par exemple, le langage C*
- *Aussi bien pour XML que pour JSON, des outils très performants ont été définis pour Java, et sont donc utilisables sur Android*

Sérialisation - L'introspection en Java



Sérialisation - L'introspection en Java

```
package org.wikibooks.fr;

public class Livre
{
    String titre;
    int nb_pages;

    public Livre(String _titre, int _nb_pages) {
        this.titre = _titre;
        this.nb_pages = _nb_pages;
    }

    public int getNombreDeFeuilles(int pages_par_feuille)
    {
        return (nb_pages+pages_par_feuille-1)/pages_par_feuille;
    }
}
```

Sérialisation - L'introspection en Java

```
// Accès à la classe Livre
```

```
Class c = Class.forName("org.wikibooks.fr.Livre");
```

```
// Obtenir le constructeur (String, int)
```

```
Constructor constr = c.getConstructor(String.class, int.class);
```

```
// -> new Livre("Programmation Java", 120);
```

```
Object o = constr.newInstance("Programmation Java", 120);
```

```
// Obtenir la méthode getNombreDeFeuilles(int)
```

```
Method method = c.getMethod("getNombreDeFeuilles", int.class);
```

```
// -> o.getNombreDeFeuilles(2);
```

```
int nb_feuilles = (int)method.invoke(o, 2);
```

Sérialisation

- *La sérialisation textuelle n'est pas toujours adaptée aux valeurs numériques, par exemple:*
 - *4294967295 est codé sur 32 bits soit 4 Bytes*
 - en json {"v":4294967295} → 16 car. soit 16 Bytes*
 - en xml <v>4294967295</v> → 17 car. soit 17 Bytes*
 - *false est un booléen, donc codé sur 1 bit*
 - en json {"v":false} → 11 car. soit 11 Bytes*
 - en xml <v>false</v> → 12 car. soit 12 Bytes*

Sérialisation - Alternative à JSON/XML

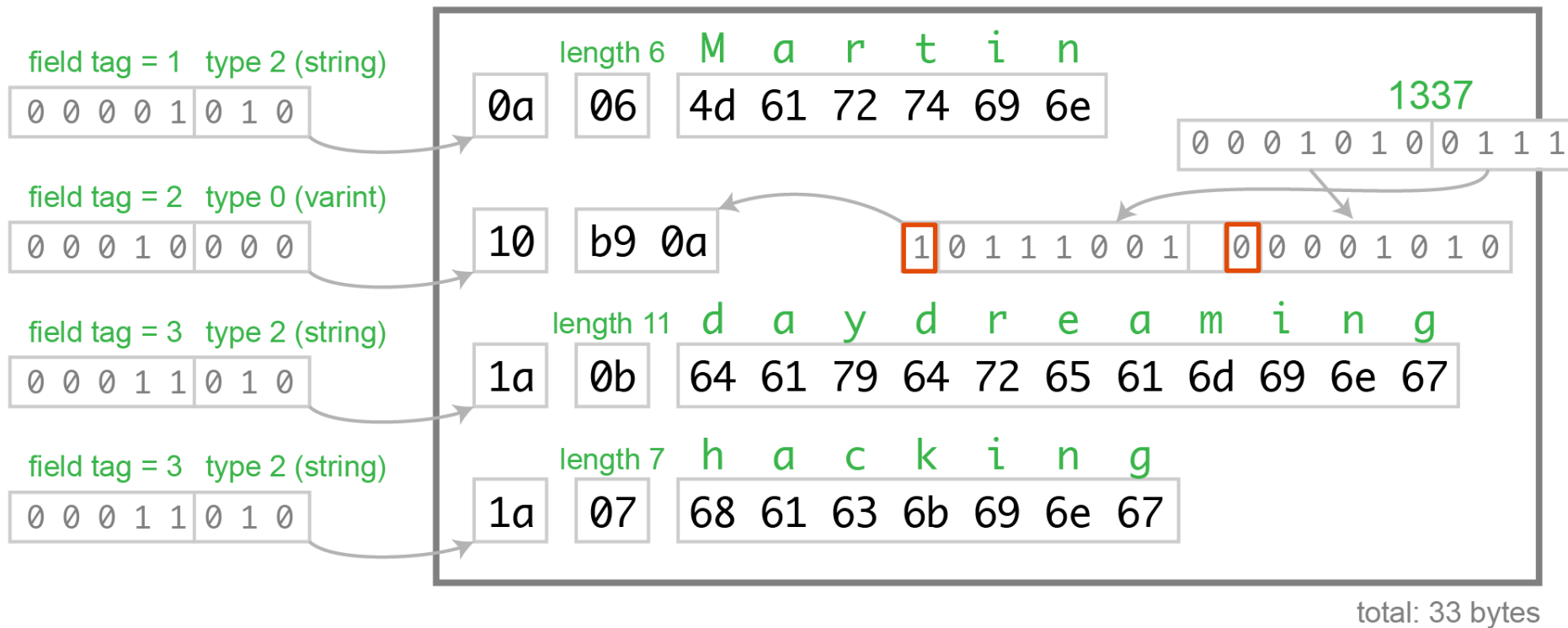
- *Protocol Buffers – format de sérialisation basé sur un langage de description d'interface développé par Google*
- *Outils officiels de génération de code pour C++, Java, GO, C#, Ruby et Python*
- *Outils tiers pour PHP, Swift, Scala, etc.*

Depuis la version 3

```
message Person {  
    required string user_name          = 1;  
    optional int64  favourite_number = 2;  
    repeated string interests          = 3;  
}
```

Sérialisation - protobuf

Protocol Buffers



Sérialisation - protobuf

.proto Type	Notes	C++ Type	Java Type
double		double	double
float		float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long
uint32	Uses variable-length encoding.	uint32	int ^[1]
uint64	Uses variable-length encoding.	uint64	long ^[1]
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2 ²⁸ .	uint32	int ^[1]
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2 ⁵⁶ .	uint64	long ^[1]
sfixed32	Always four bytes.	int32	int
sfixed64	Always eight bytes.	int64	long
bool		bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String
bytes	May contain any arbitrary sequence of bytes.	string	ByteString


Services Web

- *Sérialiser des objets est nécessaire pour consommer des services WEB. Google défend une approche RESTful, alors que Apple et Microsoft encouragent plutôt une approche RPC / SOAP sans pour autant forcément mettre à disposition des outils permettant d'automatiser leur intégration*
- *Une application mobile est de ce fait souvent constituée d'un client riche sur le mobile, et d'un ensemble de services nécessitant une authentification*


Services Web – XML-RPC

- *RPC (Remote Procedure Call) est un protocole réseau permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'applications*
- *XML-RPC est une variante utilisant un format XML standardisé (DTD) pour transmettre les paramètres et résultats de la méthode distante appelée*

POST /my_rpc



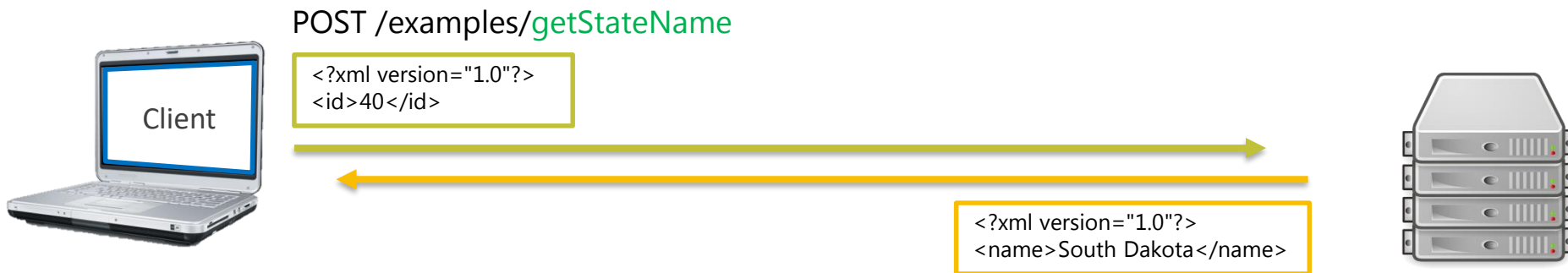
```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```



```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

Services Web – XML-RPC

- Une des plus grosse critique de XML-RPC est que l'utilisation des structures XML n'apporte rien de plus par rapport à l'utilisation de simple xml si ce n'est d'alourdir les requêtes, par exemple en reprenant l'exemple précédent:



- XML-RPC est 4 fois plus verbeux qu'un simple XML
- On rencontre donc très souvent des solutions «maisons»
- JSON-RPC est son équivalent utilisant le format JSON

Services Web – SOAP

- *Très semblable à XML-RPC bien que plus complexe (utilisation de XMLSchema et namespaces) et relativement lourd*
- *Le principal avantage est qu'il est possible de décrire les services et types de données offerts par un service web SOAP dans un fichier WSDL (Web Services Description Language). Des outils permettent de créer le code client et/ou serveur automatiquement à partir de sa définition WSDL dans différents langages (pas d'outils suffisamment mature pour Android)*

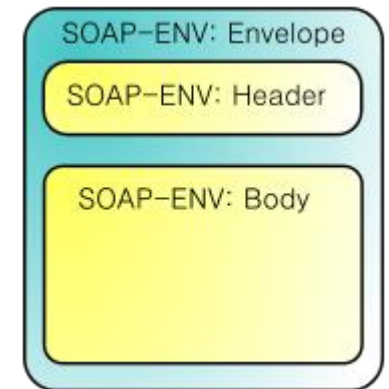


Image: Wikipedia

Services Web – SOAP

WSDL - Exemple

1

```
<wsdl:binding name="AirlineServicesSOAP" type="iis:AirlineService">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="bookFlight">
    <soap:operation soapAction="http://lsir-cis-pc4.epfl.ch:8080/AirlineService/bookFlight"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
```

2

```
<wsdl:operation name="searchFlights">
  <soap:operation soapAction="http://lsir-cis-pc4.epfl.ch:8080/AirlineService/searchFlights"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

</wsdl:binding>
<wsdl:service name="AirlineService">
  <wsdl:documentation></wsdl:documentation>
  <wsdl:port binding="iis:AirlineServicesSOAP" name="AirlineServicesSOAP">
    <soap:address location="http://lsir-cis-pc4.epfl.ch:8080/AirlineService/">
    </wsdl:port>
  </wsdl:service>
```

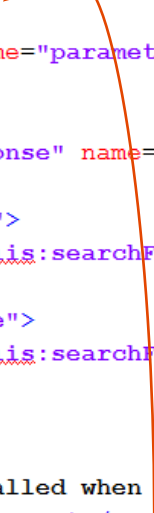

Services Web – SOAP

WSDL - Exemple

```

<wsdl:message name="bookFlightRequest">
  <wsdl:part element="iis:bookFlight" name="parameters"/>
</wsdl:message>
<wsdl:message name="bookFlightResponse">
  <wsdl:part element="iis:bookFlightResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="searchFlightsRequest">
  <wsdl:part name="parameters" element="iis:searchFlights"></wsdl:part>
</wsdl:message>
<wsdl:message name="searchFlightsResponse">
  <wsdl:part name="parameters" element="iis:searchFlightsResponse"></wsdl:part>
</wsdl:message>
1 <wsdl:portType name="AirlineService">
  <wsdl:operation name="bookFlight">
    <wsdl:documentation>This method is called when a customer wants to book a certain flight</wsdl:documentation>
    <wsdl:input message="iis:bookFlightRequest"/>
    <wsdl:output message="iis:bookFlightResponse"/>
  </wsdl:operation>
2 <wsdl:operation name="searchFlights">
    <wsdl:documentation>Method used to search flights given certain constraints</wsdl:documentation>
    <wsdl:input message="iis:searchFlightsRequest"></wsdl:input>
    <wsdl:output message="iis:searchFlightsResponse"></wsdl:output>
  </wsdl:operation>
</wsdl:portType>

```



Services Web – SOAP

WSDL - Exemple

```

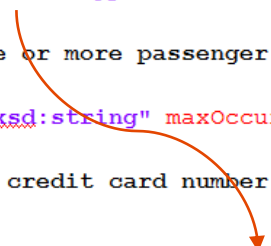
<xsd:element name="bookFlight">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="flightId" type="xsd:string" maxOccurs="1" minOccurs="1">
        <xsd:annotation>
          <xsd:documentation>id of the flight the customer wants to book</xsd:documentation>
        </xsd:annotation></xsd:element>
      <xsd:element name="client" type="iis:clientType" maxOccurs="unbounded" minOccurs="1">
        <xsd:annotation>
          <xsd:documentation>list of one or more passenger(s) for this reservation</xsd:documentation>
        </xsd:annotation></xsd:element>
      <xsd:element name="creditCard" type="xsd:string" maxOccurs="1" minOccurs="1">
        <xsd:annotation>
          <xsd:documentation>Compulsory credit card number used for the payment</xsd:documentation>
        </xsd:annotation></xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:complexType name="clientType">
  <xsd:sequence>
    <xsd:element name="fName" type="xsd:string" maxOccurs="1" minOccurs="1">
      <xsd:annotation>
        <xsd:documentation>first name</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="lName" type="xsd:string" maxOccurs="1" minOccurs="1">
      <xsd:annotation>
        <xsd:documentation>last name</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="contact" type="xsd:string" maxOccurs="1" minOccurs="1">
      <xsd:annotation>
        <xsd:documentation>complete adress</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

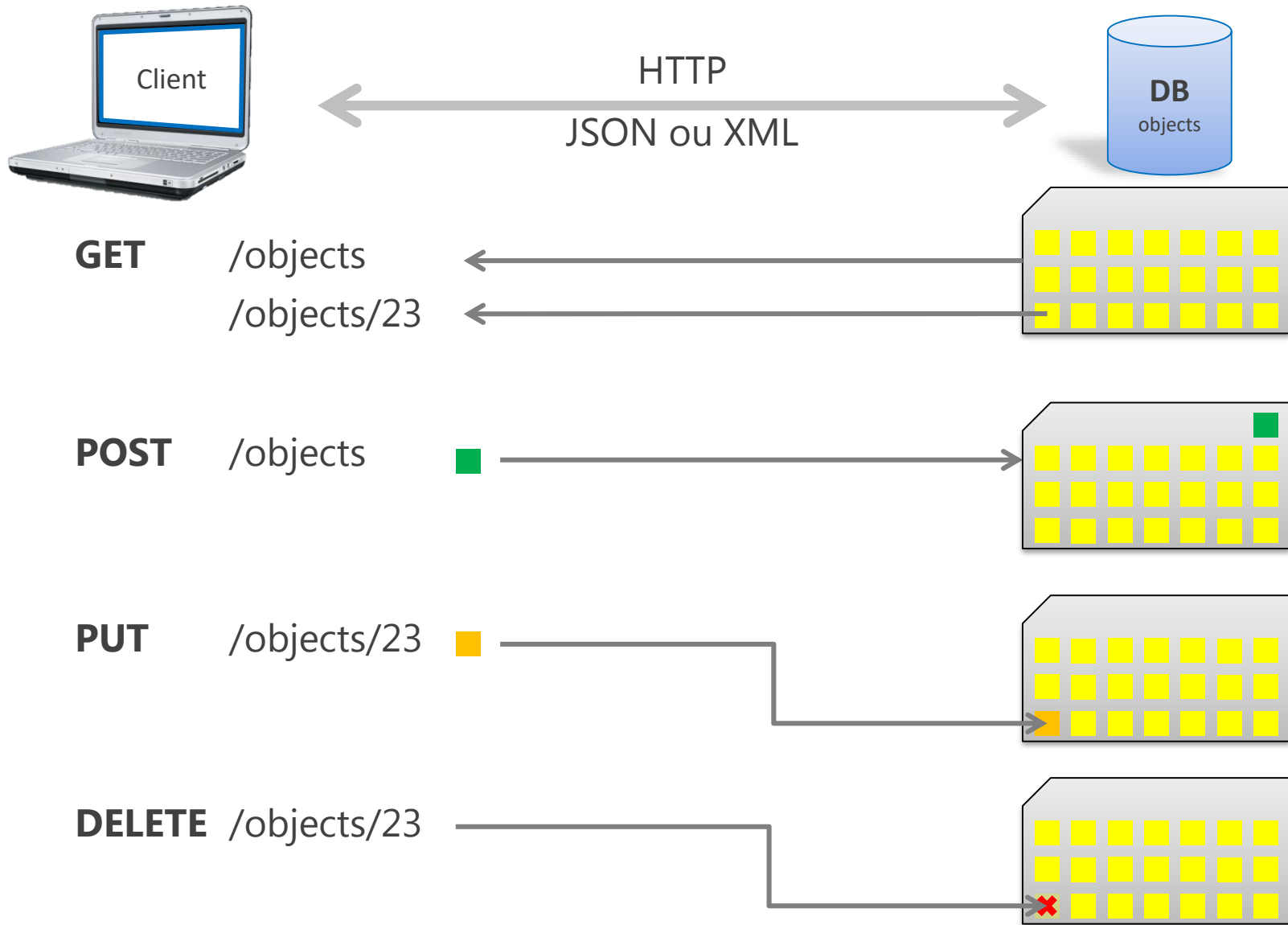
```



Services Web – RESTful

- *REST (representational state transfer) est un style d'architecture avec quelques contraintes:*
 - *Séparation client/serveur*
 - *Sans état*
 - *Mise en cache*
 - *Interface uniforme (chaque ressource possède son URI)*
- *Un service web respectant ces contraintes est appelé RESTful*
- *Principalement utilisé pour accéder à des ressources dans une base de données*
- *Utilisation d'HTTP pour le transport et de XML ou JSON pour la sérialisation des données échangées*

Services Web – RESTful - Exemple



GraphQL

- *GraphQL est un langage de requête et de manipulation développé en interne par Facebook et open-sourcé depuis 2015*
- *Il s'agit d'une alternative aux API REST*
- *Le client peut préciser dans sa requête la structure des données qu'il souhaite récupérer*
 - *Permet d'éviter les problèmes d'under-fetching ou d'over-fetching, ce qui est particulièrement bienvenu dans le monde mobile*

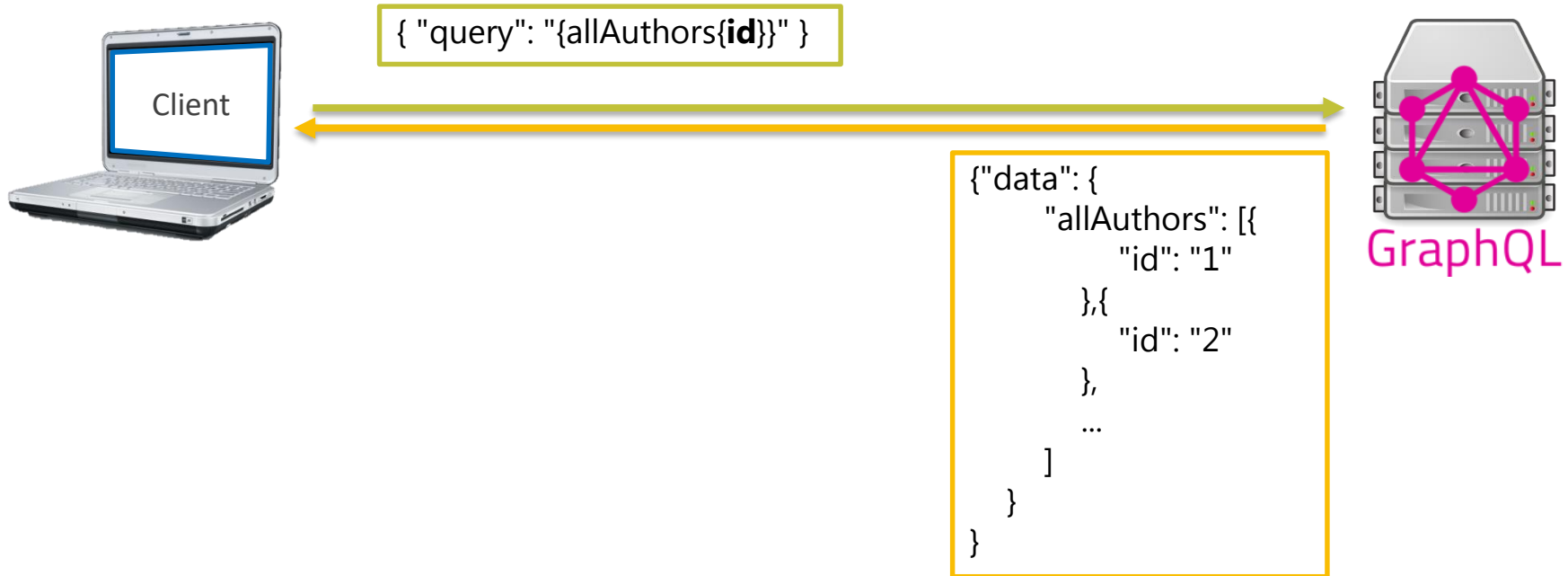
GraphQL – Exemple

- *Un serveur de test GraphQL a été mis en place pour le laboratoire 2, voici le schéma de la base de données*



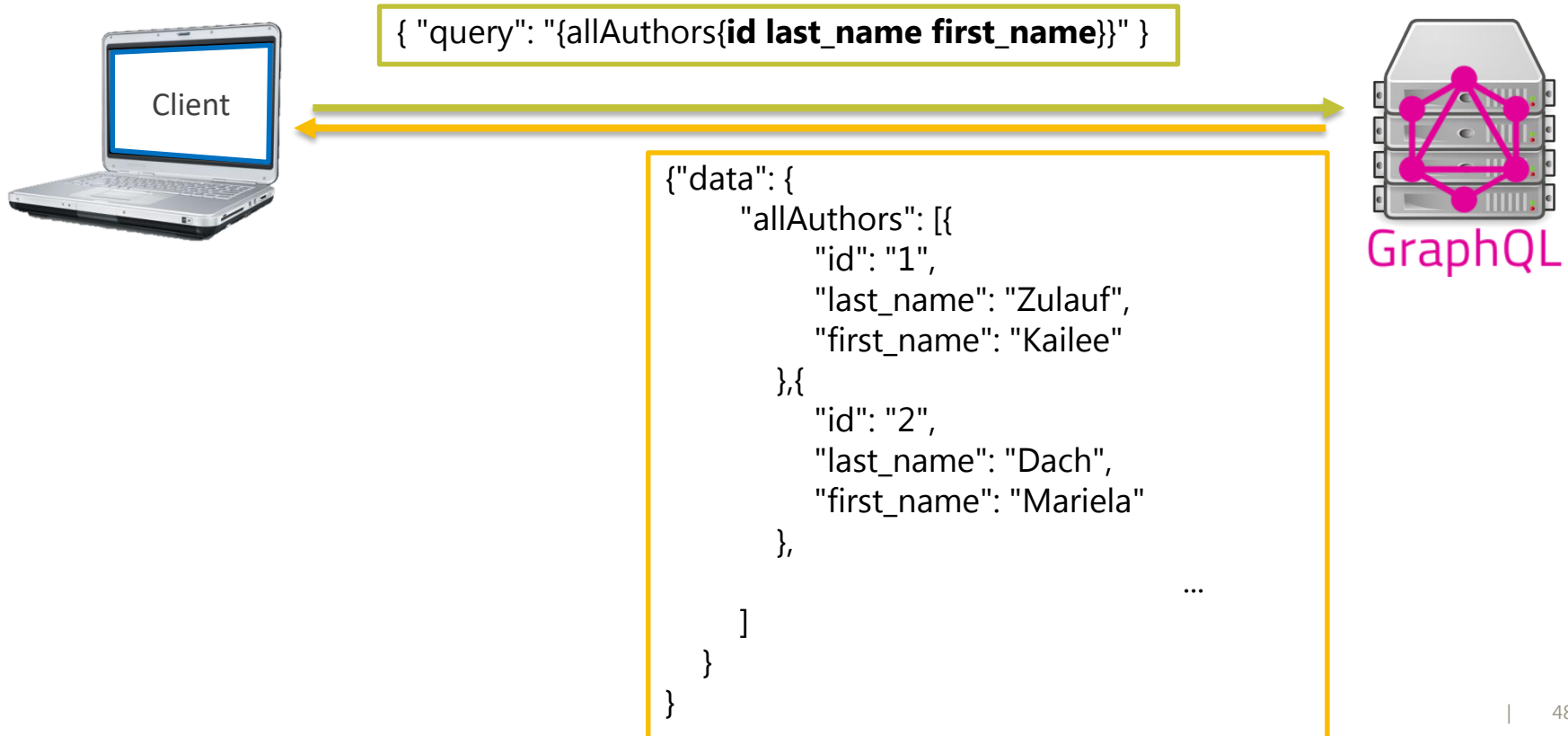
GraphQL – Exemple

- *La requête suivante permet de récupérer tous les auteurs*



GraphQL – Exemple

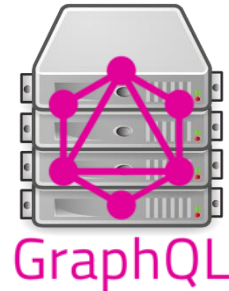
- *La requête suivante permet de récupérer tous les auteurs*



GraphQL – Exemple



```
{ "query": "{allAuthors{id posts{id title}}}" }
```



```
{ "data": {  
  "allAuthors": [  
    {  
      "id": "1",  
      "posts": [{  
        "id": "1",  
        "title": "Sequi unde corrupti tenetur id."  
      }],  
      "id": "1001",  
      "title": "Perferendis ... commodi labore totam quia."  
    },  
    {  
      "id": "2001",  
      "title": "A animi non laudantium doloribus in."  
    },  
    {  
      "id": "3001",  
      "title": "Aut ... laborum recusandae."  
    }  
  ]  
},  
  {  
    "id": "2",  
    "posts": [{  
      "id": "2",  
      "title": "Sit quia ... molestias consequatur."  
    }]  
  },  
  ...  
]  
}
```

Sérialisation XML

- *Nombreux outils disponibles pour Java :*
 - *XStreams (<https://github.com/codehaus/xstream>)*
 - *JDOM (<http://www.jdom.org/>)*
 - *JAXP, JAXB (<https://jaxb.java.net/>)*
 - *XmlEncoder, XmlDecoder*
(<http://docs.oracle.com/javase/7/docs/api/java/beans/XMLEncoder.html>)
- *En C++, la librairie DOM est peu pratique, les outils SOAP de Microsoft offrent des alternatives puissantes dans la galaxie .NET*

Sérialisation XML

- *XML propose une formalisation très stricte des données; les outils de conversion XML – Java sont cohérents, et parfois contraignants*
- *Les contraintes imposées ont un effet positif dans le sens où la sérialisation est plus sûre et peut être validée*
- *XML est un langage verbeux; le protocole de transmission s'en trouve quelque peu alourdi*
- *Les parsers XML sont parfois d'une utilisation un peu complexe (parsing par événements de type SAX)*

Sérialisation JSON

- *JSON est un sous-ensemble de Javascript*
- *Google a développé une librairie très performante appelée GSON pour l'utilisation de JSON en Java*
(<https://code.google.com/p/google-gson/>)
- *Jansson est une librairie utilisable pour le langage C*
(<https://github.com/akheron/jansson>)
- *JsonCpp est utilisable pour des applications en C++*
(<https://github.com/open-source-parsers/jsoncpp>)

Sérialisation JSON

- *Lors de la conversion d'un objet JSON en instance Java et vice-versa, il n'y a pas de contrôle de cohérence*
- *Des champs manquants ou non renseignés ne seront pas signalés comme tels*
- *Si une librairie comme GSON facilite grandement les opérations, il reste important de bien maîtriser les opérations de sérialisation et désérialisation (format des dates et des types spéciaux)*
- *Voir <http://www.javacreed.com/gson-deserialiser-example/>*

XML ou JSON ?

- *Le fait que XML permette une validation le rend plus sûr du point de vue du serveur, au prix d'un parsing plus complexe. DTD ou XML Schema*
- *Si un composant de votre architecture est basé sur JavaScript, un échange en JSON sera certainement plus simple à mettre en œuvre*
 - *JSON est un sous-ensemble de JavaScript; si les données doivent être exploitées aussi avec du code JavaScript, alors JSON est à favoriser*
 - *XML est défavorable dans le monde Javascript et dans le monde PHP*
- *XML permet aussi l'utilisation d'outils comme XPath ou XSL*

Evolution du protocole

- *Un point important à prendre en compte lors de la définition d'un protocole est son évolution future*
- *Lorsque vous développez une application, il y aura très certainement une version 2.0 avec de nouvelles fonctionnalités*
- *Une solution est de refuser l'accès aux anciennes versions, faut-il encore que cela soit prévu dans la première version du protocole...*

Evolution du protocole

- *Certains parsers, par exemple Xstream, refuseront de dé-sérialiser une instance de classe différente (par exemple: nouvelle variable). Cela nécessiterait de garder, sur le serveur, l'historique des modèles POJO et de switcher en fonction de la version, très vite ingérable...*
- *D'autres parsers sont beaucoup plus permissifs, par exemple Gson. Ici c'est votre code traitant les données qui devra être assez flexible pour gérer les différentes versions, par exemple avec des variables manquantes*

Objets volumineux

- *Certains objets peuvent, après sérialisation, représenter un flux d'informations très volumineux*
- *La transmission sous forme textuelle, même en JSON, peut impliquer une transmission de plusieurs Mo*
- *Si par malchance, la connexion disponible est lente ou intermittente, l'application peut dysfonctionner*

Objets volumineux

- *Une fois sérialisée, une instance d'objet peut être compressée par un algorithme approprié (DEFLATE, par exemple)*
- *L'envoi de requêtes POST dont le contenu est compressé ne fait pas partie des standards HTTP, c'est possible de le faire mais nécessite la maîtrise du développement du client et du serveur*
- *Certain type de fichiers (jpeg, mp3, png, etc.) sont déjà compressés, il est donc inutile de les compresser à nouveau*
- *L'utilisation de Base64 pour joindre un média dans un document json ou xml est rarement la solution à privilégier...*

Applications embarquées

- *Dans le cas de développements mobiles embarqués non associés à une plateforme comme Android ou iOS, des sérialisateurs (XML et JSON) sont disponibles sous forme de code source*
- *Souvent, le service est implémenté dans un environnement d'exécution différent que celui du client (JavaEE - C, PHP - Java, etc...)*
- *Dans ce dernier cas, les règles plus strictes d'encodage de XML peuvent s'avérer utiles*

Applications embarquées

- *L'Internet des objets constitue dans beaucoup de cas aussi un domaine d'applications mobiles*
- *Les implants (pacemakers, défibrillateurs, implants neurologiques) constituent des cas particuliers de systèmes mobiles assez sensibles !*
- *Ces implants sont de plus en plus souvent connectés pour des raisons évidentes de diagnostic, de surveillance, de statistiques et d'analyse*

Applications embarquées

- *Traditionnellement, les implants étaient connectés par un protocole point-à-point, relativement sûr puisqu'en vue directe*
- *Les implants modernes se connectent à des serveurs distants par les réseaux mobiles ou des passerelles Bluetooth*
- *Les applications embarquées sont donc de plus en plus concernées par les problèmes liés à la sérialisation et à la transmission*

Considérations de sécurité



HEALTH NEWS FEBRUARY 20, 2018 / 10:18 PM / 7 MONTHS AGO

Pacemakers, defibrillators are potentially hackable

Hacking risk leads to recall of 500,000 pacemakers due to patient death fears

FDA overseeing crucial firmware update in US to patch security holes and prevent hijacking of pacemakers implanted in half a million people

The Guardian

- *La première conclusion à tirer est que toute liaison avec un service web devrait être au moins sécurisée par ~~SSL~~ / TLS*
- *La deuxième est que ce n'est probablement pas toujours suffisant*

Considérations de sécurité

- *TLS permet de chiffrer la connexion et de valider l'identité du serveur*
- *L'utilisation de certificats SSL auto-signés n'est pas mauvaise en soit mais il faut absolument valider le certificat côté client (risque de MITM)*
- *Bien souvent on tombe sur des tutoriels sur le net indiquant comment accepter les certificats auto-signés (soit tous les certificats... aussi ceux auto-signés par les autres)*
- *Il est possible de distribuer notre certificat dans l'application pour vérifier au runtime l'identité du serveur (certificate pinning)*

Considérations de sécurité

- *Dans la majeure partie des cas TLS n'identifie pas le client, il y a toujours échange de credentials / token, c'est un point faible*
<http://www.7xter.com/2015/07/intercepting-android-ssl-traffic.html>
ou
<https://www.frida.re/>
- *Une piste est l'utilisation de clés privées sur le client mobile permettant de signer le contenu envoyé. On ne peut pas inclure la clé directement dans l'application car elle serait facilement récupérable. Cela nécessite de générer ou insérer la clé après installation, de faire un appairage avec le serveur / compte de l'utilisateur, et bien sûr de stocker la clé...*

Et le serveur ?

Y a-t-il une technologie plus pertinente qu'une autre pour implémenter un serveur pour applications mobiles ?

- *PHP est encore la technologie la plus répandue et la plus facile à héberger*
- *JavaEE est plus performant et offre plus de possibilités (multithreading, ORM, JMS, Spring, etc.). L'écosystème est éprouvé (Serveurs d'applications, IDE, etc.)*
- *Node.js évolue très rapidement. Il propose des solutions temps-réel (WebSocket) et le multithreading arrive*

PUSH

- *HTTP est un protocole applicatif qui implique une connexion établie par le client vers le serveur*
- *Une connexion inverse n'est pas aisée à implémenter*
- *Apple et Microsoft implémentent une forme de PUSH email en conservant une connexion semi-permanente*
- *Blackberry utilise les protocoles de téléphonie (ce qui implique la participation de l'opérateur)*

PUSH

- *De nombreuses applications nécessitent des notifications spontanées de la part d'un serveur*
- *Les solutions dépendent fortement de la plateforme considérée*
- *On peut citer quelques pistes à envisager, sachant par ailleurs que le serveur devra éventuellement s'adapter au client (problème d'authentification et d'identification non trivial)*

Connexions actives

- *Lorsque le trafic de messages est important du serveur vers le client, on peut implémenter un schéma basé sur des connexions actives en permanence (ex: jeux vidéo)*
- *Ceci implique souvent des techniques (généralement sous-jacentes) d'Active Poll pour conserver la connexion lors de pertes de liaison (handover, perte de réseau)*

Connexions actives

- *Disposant d'une connexion active, le serveur peut contacter le client de manière spontanée*
- *Plusieurs implémentations possibles:*
 - *Sockets TCP/IP*
 - *Websockets (ex: Socket.io)*
 - *HTTP/2 (Standard datant de mai 2015)*

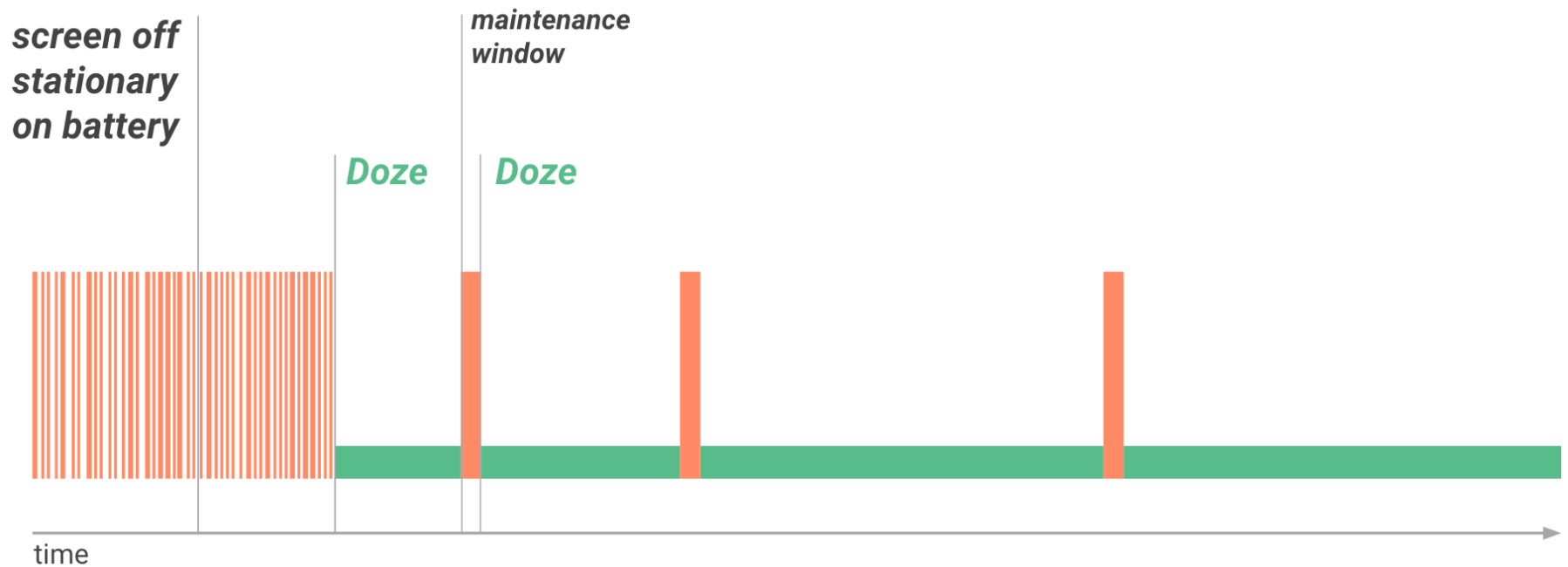
Active Poll

- *Le client ouvre périodiquement une connexion vers le serveur qui va utiliser la réponse pour transmettre l'information au client*
- *Si cette information implique une réponse du client, une nouvelle requête est émise sur la connexion existante, avec une réponse associée*
- *L'Active Poll est intéressant si les interrogations restent relativement rares*

Active Poll

- *Par souci d'économie de batterie, la plupart des plateformes implémentent à présent des politiques strictes pour les tâches de fond (ex: Android Doze – Api 23 et App Standby Buckets – API 28)*
- *Le smartphone est mis en veille profonde, les fonctionnalités les plus énergivores du smartphone seront purement et simplement désactivées, comme par exemple le balayage des réseaux Wi-Fi, la synchronisation et l'accès au réseau, recherche de mises à jour etc.*
- *Le smartphone sort de sa veille profonde uniquement lors de fenêtres de maintenance*

Active Poll - Android Doze



Active Poll - App Standby Buckets – API 28

- *A partir d'Android 9 (API-28), le système va classer les applications dans quatre catégories en fonction de leur utilisation:*
 - *Active bucket*
L'application est en cours d'utilisation
 - *Working set bucket*
L'application n'est pas en cours d'utilisation mais celle-ci est très souvent utilisée (~quotidiennement)
 - *Frequent bucket*
Application utilisée régulièrement, mais pas forcément tous les jours
 - *Rare bucket*
Application peu utilisée
 - *Never bucket*
Application jamais utilisée

Active Poll - Doze + App Standby Buckets

Setting	Jobs *	Alarms †	Network ‡	Firebase Cloud Messaging §
User Restricts Background Activity				
Restrictions enabled:	Never	Never	Never	Messages discarded in Android P+ starting January 2019
Doze				
Doze active:	Deferred to window	Regular alarms: Deferred to window While-idle alarms: Deferred up to 9 minutes	Deferred to window	High priority: No restriction Normal priority: Deferred to window
App Standby Buckets (by bucket)				
Active:	No restriction	No restriction	No restriction	No restriction
Working set:	Deferred up to 2 hours	Deferred up to 6 minutes	No restriction	No restriction
Frequent:	Deferred up to 8 hours	Deferred up to 30 minutes	No restriction	High priority: 10/day
Rare:	Deferred up to 24 hours	Deferred up to 2 hours	Deferred up to 24 hours	High priority: 5/day

Active Poll

- *Dans le monde Java, il existe plusieurs outils pour travailler avec les Threads: ExecutorService*
 - *Gestion d'un pool de threads*
 - *File d'attente, gestion des priorités*
 - *Tâches périodiques: `scheduleAtFixedRate()`*
- *Ces outils sont bien entendu aussi disponibles et utilisables sur Android*
 - *Pas adaptés pour de l'active poll sur mobile*

Active Poll

Sur Android on privilégiera les outils spécifiques à la plateforme:

- *AlarmManager (API 1)*
 - *Permet d'enregistrer des alarmes systèmes qui réveilleront notre application par un Intent*
 - *Dès l'API 19, on ne peut plus fixer une alarme précise, le système profitera d'une fenêtre de maintenance pour lancer toutes les alarmes en attentes*
 - *Les alarmes enregistrées sont perdues en cas de redémarrage du téléphone. Il faut donc détecter l'intent système ACTION_BOOT_COMPLETED pour recréer les alarmes au démarrage*
 - *Le système peut retomber en veille profonde avant que notre tâche ne soit terminée...*
 - *L'alarme peut être lancée à un moment inopportun par rapport à ce que l'on souhaite réaliser (absence de réseau, pas en charge, etc.)*

Active Poll

- *JobScheduler (API 21)*
 - *Les tâches enregistrées survivent à un redémarrage*
 - *Le système reste actif tant que notre tâche n'est pas terminée*
 - *Critères d'exécution (connexion possible, WiFi disponible, téléphone en charge, etc.)*
 - *Politique de back-off en cas d'échec de la tâche*
 - *Minimum 15 minutes entre deux exécutions planifiées*

Active Poll

```
JobInfo.Builder b =  
    new JobInfo.Builder(ID, new ComponentName(this, MyJobService.class));  
  
PersistableBundle pb = new PersistableBundle();  
pb.putString("MY_PARAM", "value");  
b.setExtras(pb);  
  
b.setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY);  
b.setPeriodic(AlarmManager.INTERVAL_FIFTEEN_MINUTES);  
b.setPersisted(true);  
b.setRequiresCharging(false);  
b.setRequiresDeviceIdle(false);  
b.setBackoffCriteria(10000L, JobInfo.BACKOFF_POLICY_EXPONENTIAL);
```

Active Poll

- *En résumé, JobScheduler est à privilégier pour réaliser de l'active poll sur Android*
- *Pas de support des versions < 21 (15% sept. 2018)*
→ *librairie Firebase JobDispatcher*
- *Pas possible d'avoir une période de moins de 15 minutes*
→ *librairie Firebase Cloud Messaging*

Android JetPack - WorkManager

- *WorkManager est disponible depuis peu dans le pack de librairies Android Jetpack*
- *Celle-ci propose une API unifiant l'utilisation de AlarmManager, Firebase JobDispatcher et JobScheduler*

```
// Create a Constraints object that defines when the task should run
Constraints myConstraints = new Constraints.Builder()
    .setRequiresDeviceIdle(true)
    .setRequiresCharging(true)
    .build();

// ...then create a OneTimeWorkRequest that uses those constraints
OneTimeWorkRequest compressionWork =
    new OneTimeWorkRequest.Builder(CompressWorker.class)
        .setConstraints(myConstraints)
        .build();
```


Lazy Poll

- *Dans ce cas de figure, le serveur stocke les informations qu'il veut faire parvenir au client*
- *Lorsqu'il reçoit un REQUEST authentifié du client, il profite de la RESPONSE pour transmettre lesdites informations (multiplexage éventuel nécessaire)*
- *Cette technique peut être associée avec l'Active Poll*

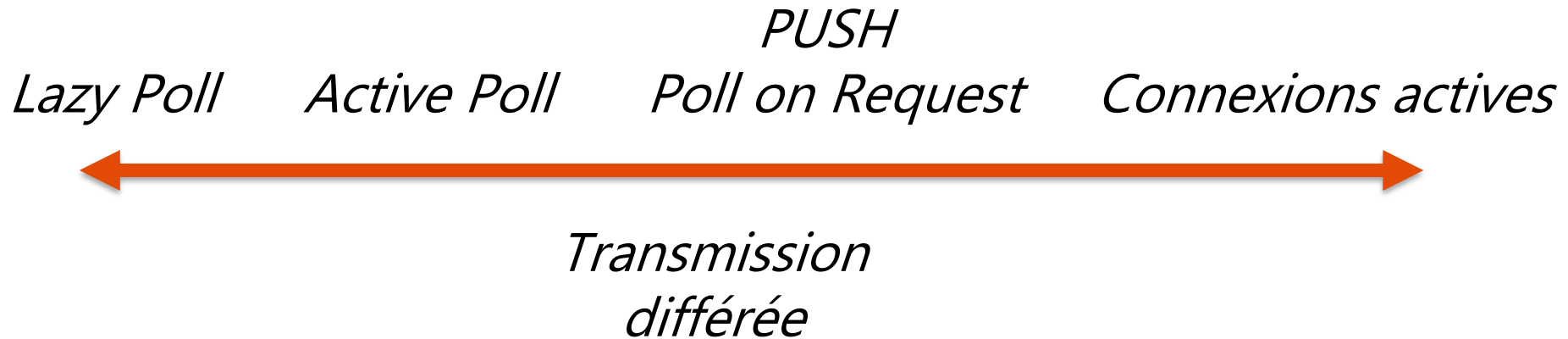
Poll on Request

- *Dans le cas de clients reliés au réseau téléphonique, et selon la plateforme, il est possible d'atteindre le client par un protocole de téléphonie adéquat*
- *En réponse, le client peut alors ouvrir une connexion vers le serveur, et on se retrouve dans le cas de figure de l'Active Poll*
- *Blackberry implémentait son service de Push e-mail sur cette base ; ceci nécessite une extension du protocole de téléphonie, donc l'accord des opérateurs*

Poll on Request

- *Google propose un service de messagerie cloud permettant l'envoi message push:
<https://firebase.google.com/docs/cloud-messaging/>*
- *Il est possible d'envoyer un message à un utilisateur particulier, un groupe d'utilisateurs ou tous les utilisateurs*
- *Le payload des messages étant très limité, habituellement on l'utilise uniquement pour notifier l'application qu'elle a des données à venir récupérer sur le serveur*
- *Cette technique nécessite l'utilisation des Google Play Service et n'est donc pas disponible sur tous les appareils. Selon le type d'application, l'utilisation d'un service Google n'est pas adaptée*

Résumé



Autres techniques complémentaires:

- *Multiplexage*
- *Cache*
- *Information look-ahead*

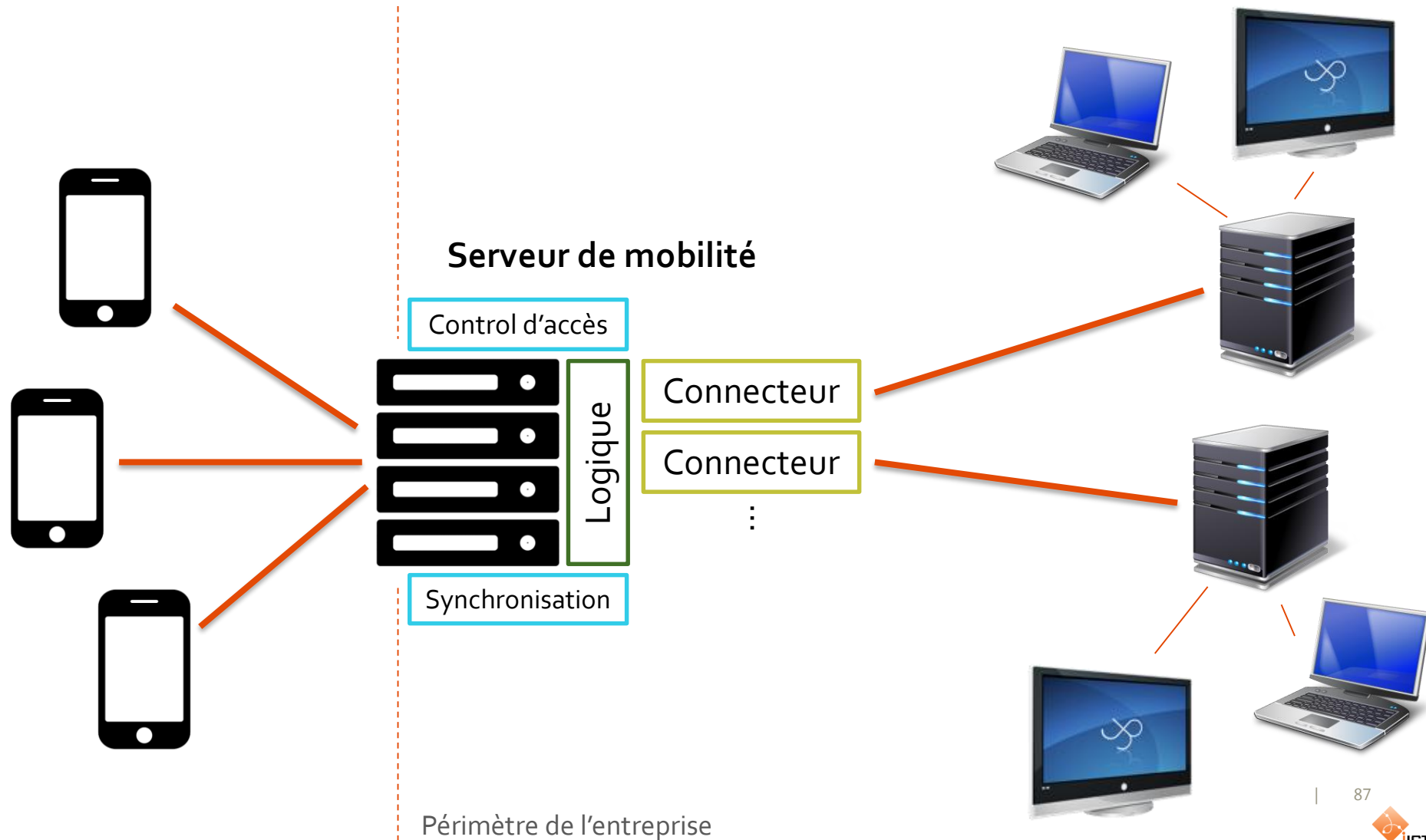
Serveur de mobilité

- *Les entreprises utilisent toujours d'anciens services ou applications qui ne sont pas adaptés au mobile*
- *Devoir réécrire ces anciennes applications pour utiliser des standards de communication plus modernes a un coût important*
- *En général on va vouloir «cacher» ces services existants derrière un serveur de mobilité*
- *Ce serveur de mobilité permettra aussi de simplifier et d'unifier (et donc de renforcer) les problèmes de sécurité*

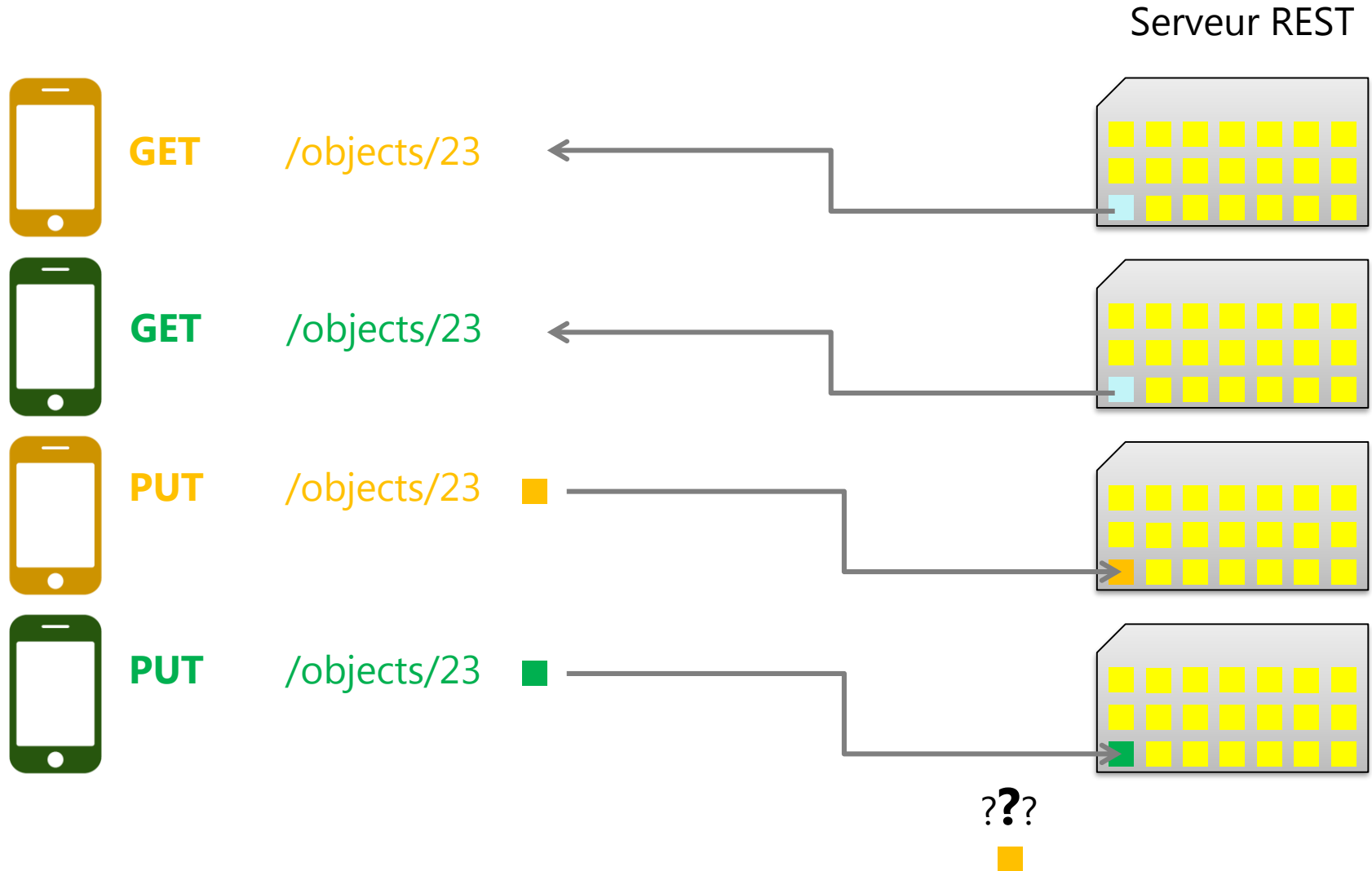
Serveur de mobilité

- *Un des grand challenge des applications mobiles est de pouvoir travailler sur les données en l'absence de réseau:*
 - *L'accès aux données est un problème qui peut être résolu par une mise en cache*
 - *La synchronisation des données modifiées lorsque le réseau est de nouveau disponible n'est pas triviale (résolution des éventuels conflits)*

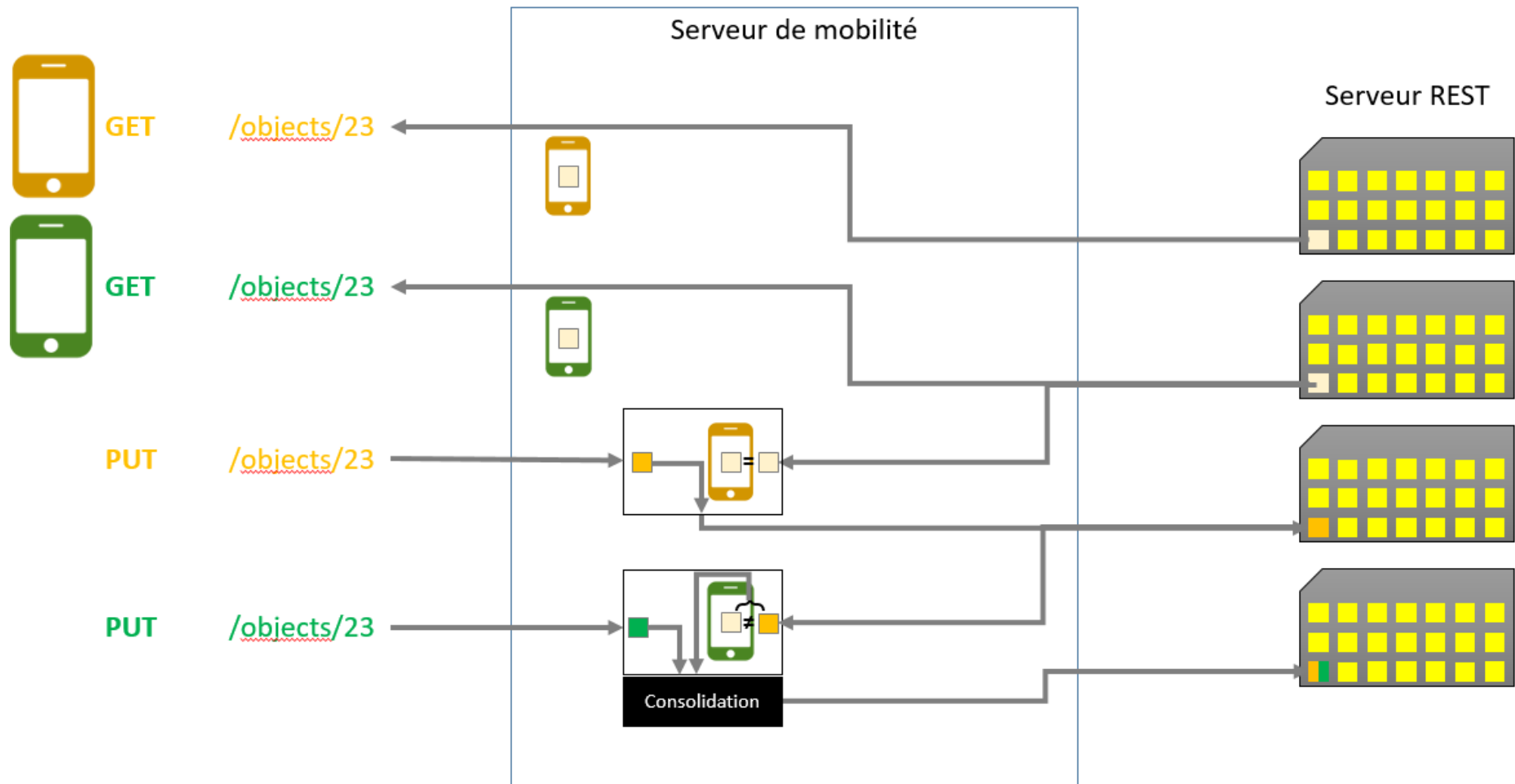
Serveur de mobilité



Serveur de mobilité - Synchronisation

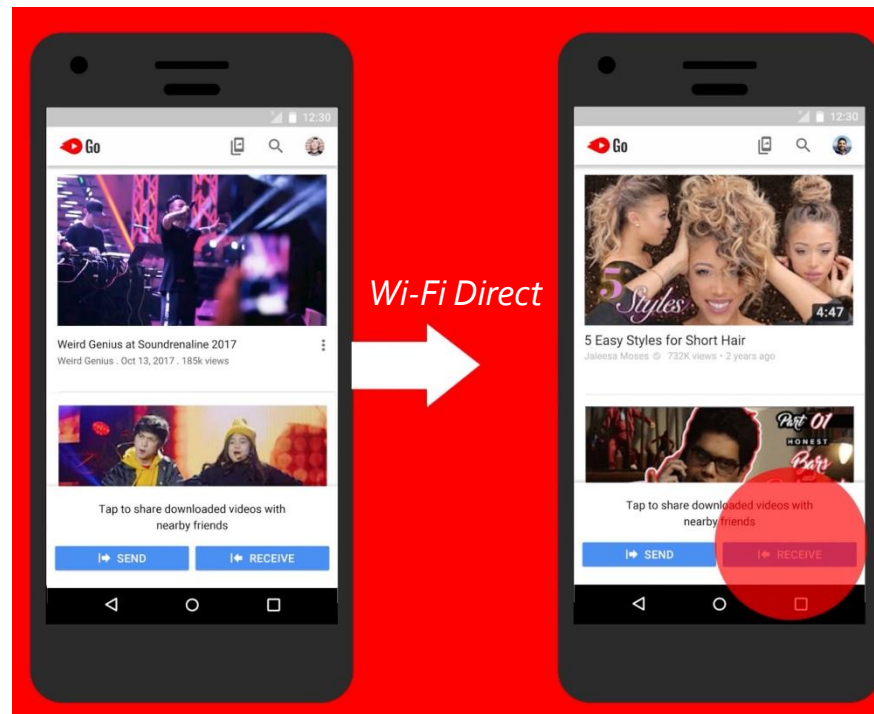


Serveur de mobilité - Synchronisation



Alternatives – Android Go

- *Version d'Android destinée aux périphériques d'entrée de gamme*
- *Les applications de base sont plus légères*
- *L'échange de contenu p2p est encouragé*



Exercice

- *Problématique*
Nous avons une base de données contenant un grand nombre d'anciennes cartes postales géo-localisées (image + métadonnées). Nous souhaitons réaliser une application mobile grand public permettant de consulter les cartes postales disponibles dans sa région (géolocalisation).
- *Quel type de communication est le plus adapté (requêtes séparées, transmission différée, push, active poll, etc.) ?*
- *Vous devez définir le protocole (API) qui permettra la consultation de ces cartes postales depuis votre application mobile. Quelles seront les méthodes, les entrées/sorties et sous quel format les données (images et métadonnées) seront échangées ?*