C2DM

Cloud to Device Messaging

Push!

Instant notifications on Android through C2DM

Karol Kuczmarski

In this talk

- What are push notifications and how do they work
- Differences between Android and iOS
- Client implementation of Android C2DM
- Tips & tricks

Push notifications

What and why?

Push notifications

- Received „instantly” by the device
  - Sent by some remote server, relayed via **push server**
- Contain small amount of information
  - More like wake-up call rather than actual message
- Handled mostly by the device OS
  - Long-lived, high timeout connection to the push server
  - Exposed by system API to apps running on the device

Benefits

- Avoid constant polling of the remote server
  - Saves bandwith and reduces battery use
- Receive notifications outside of synch. cycles
  - User can be notified immediately
- Handle events when application is not running
  - Platform-specific restrictions apply

Applications

- Asynchronous messaging
  - Most notably e-mail
- Social network updates
  - Facebook, Twitter, Foursquare, …
- Prospective searching
  - RSS readers, news from specialized sources, …
- Games
  - Especially multiplayer and/or time-dependant games
- Software updates

- …and many more

Push notifications

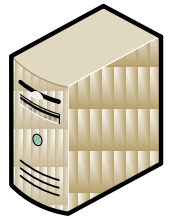
How?

Pushes… how do they work?

Mobile device

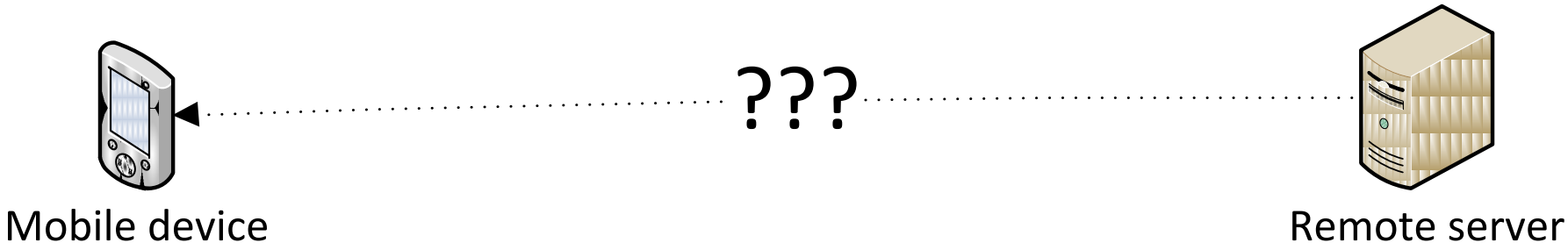
Pushes… how do they work?

Mobile device

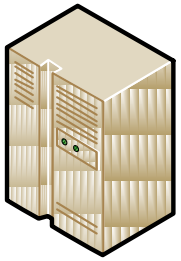


Remote server

Pushes… how do they work?

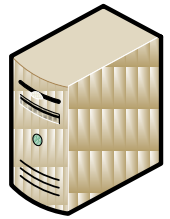


Pushes… how do they work?



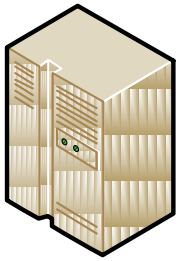
Push server

Mobile device



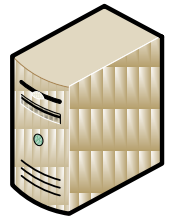
Remote server

Pushes… how do they work?



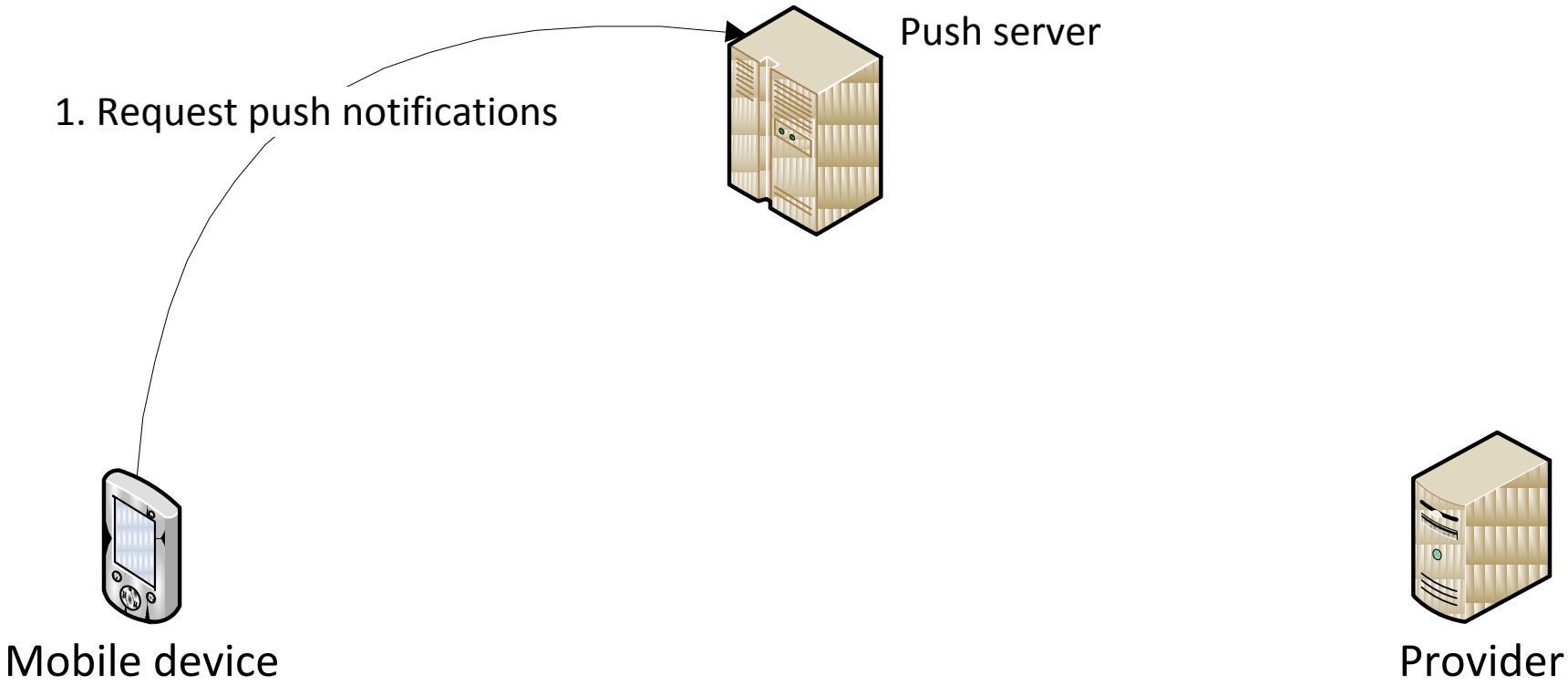
Push server

Mobile device

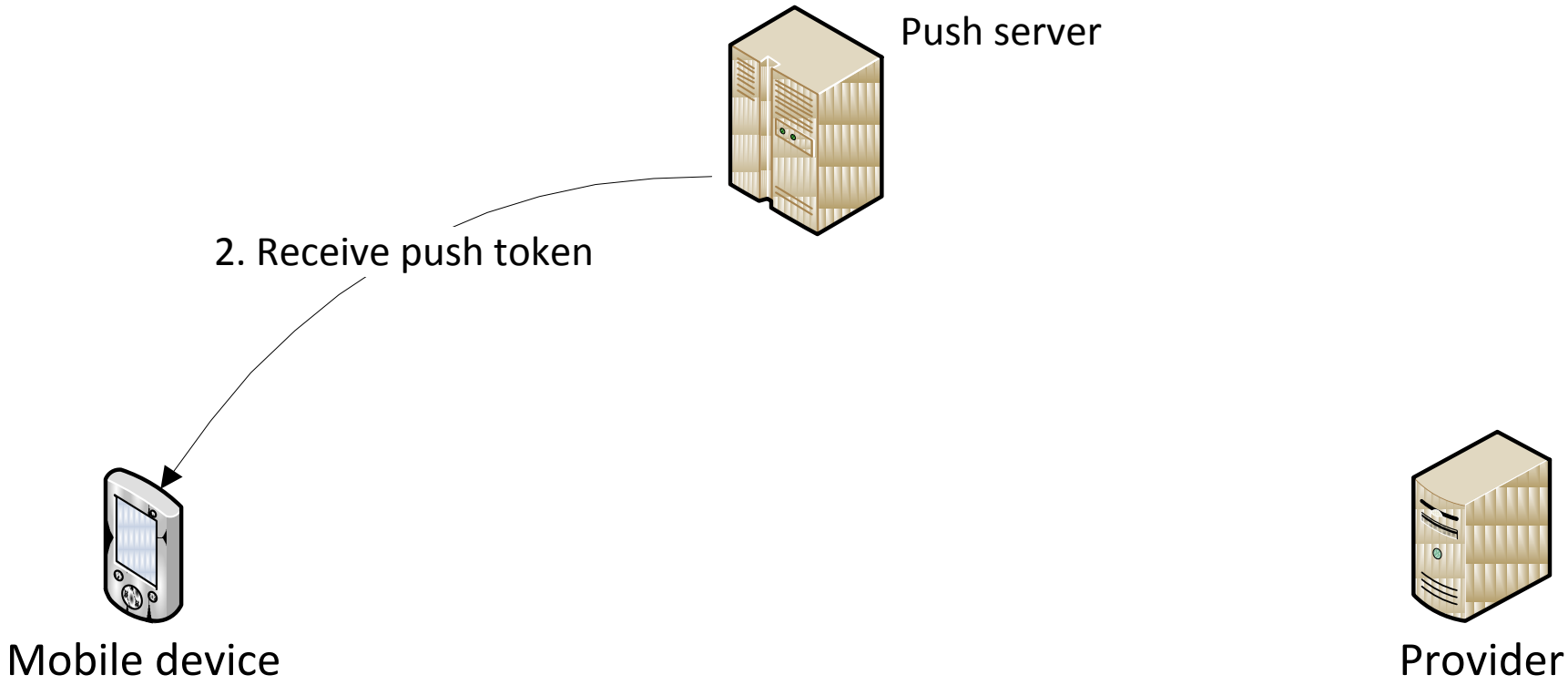


Provider

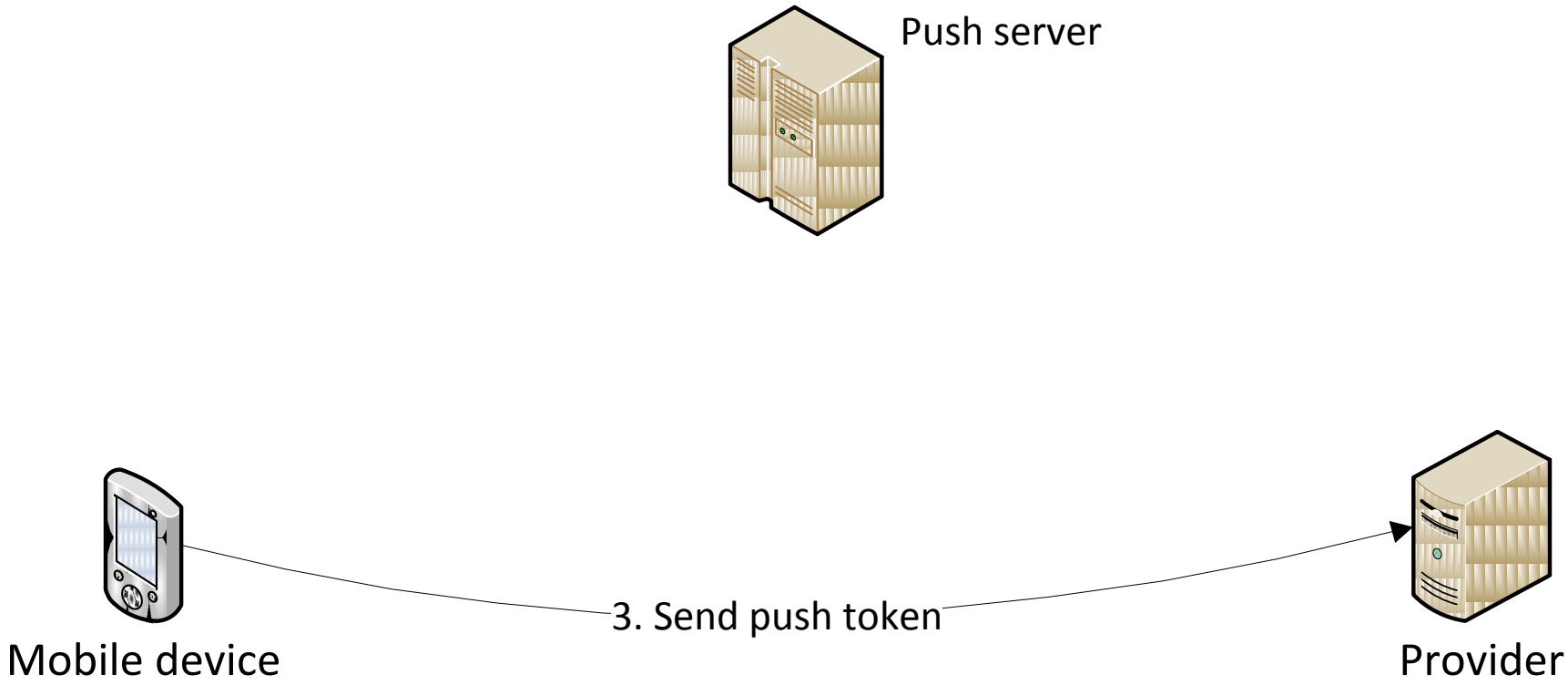
Pushes… how do they work?



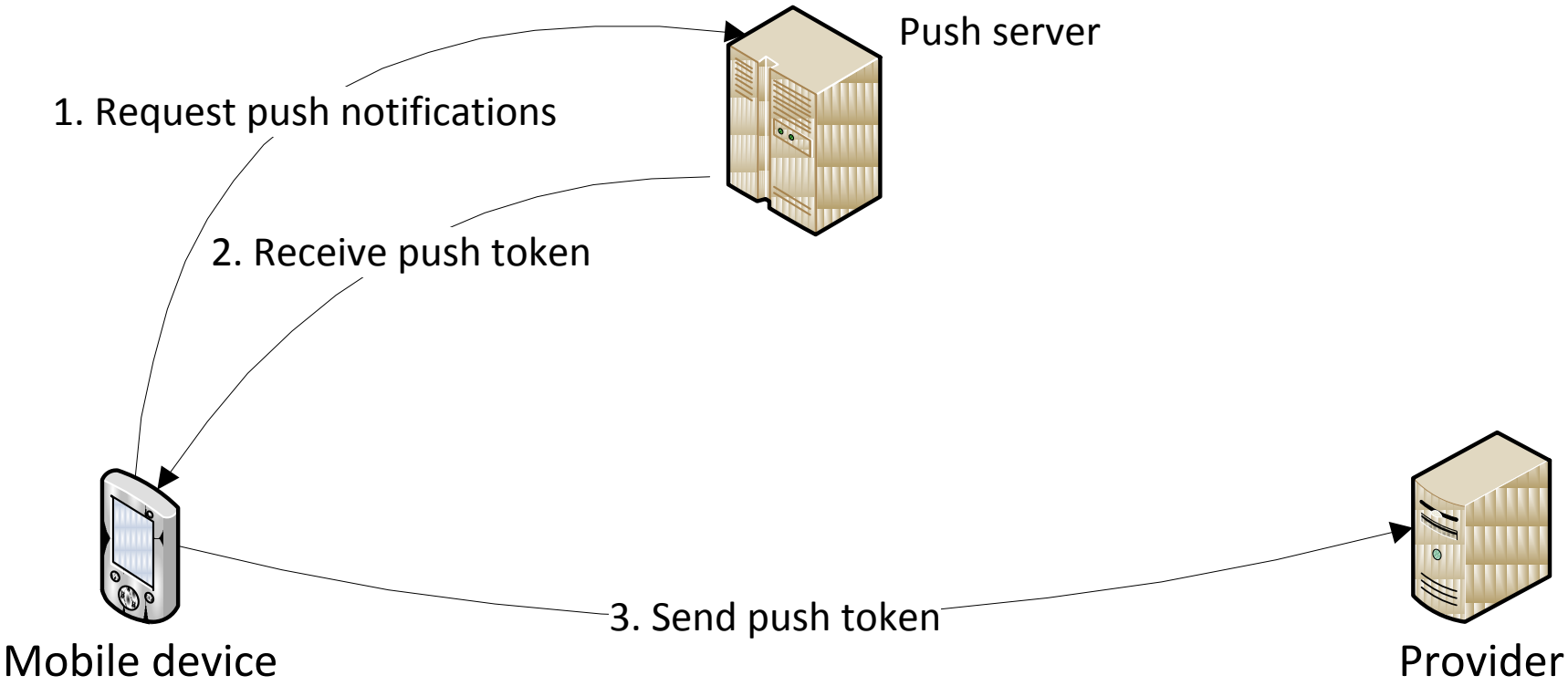
Pushes… how do they work?



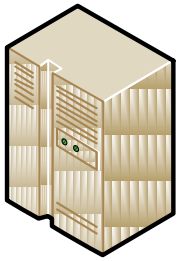
Pushes… how do they work?



Pushes… how do they work?

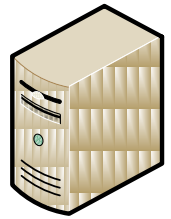


Pushes… how do they work?



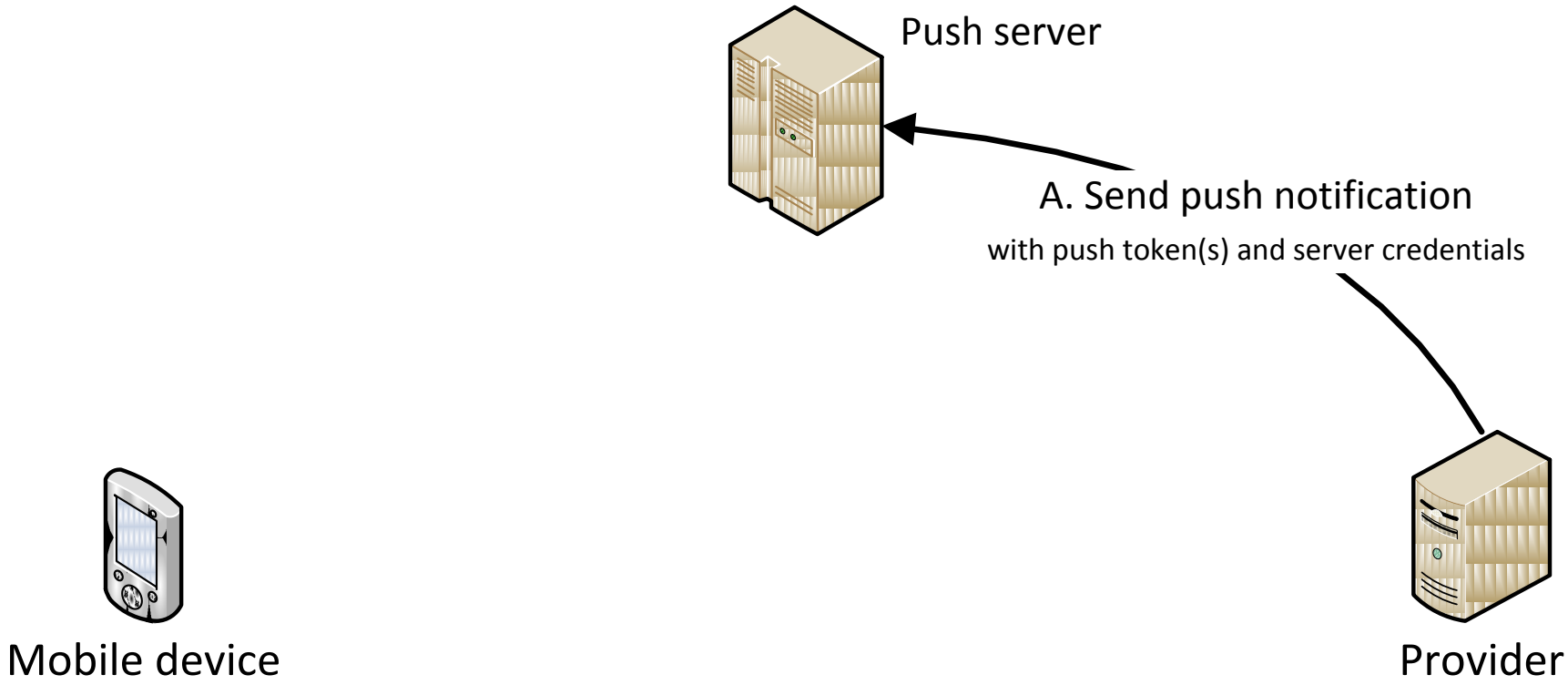
Push server

Mobile device

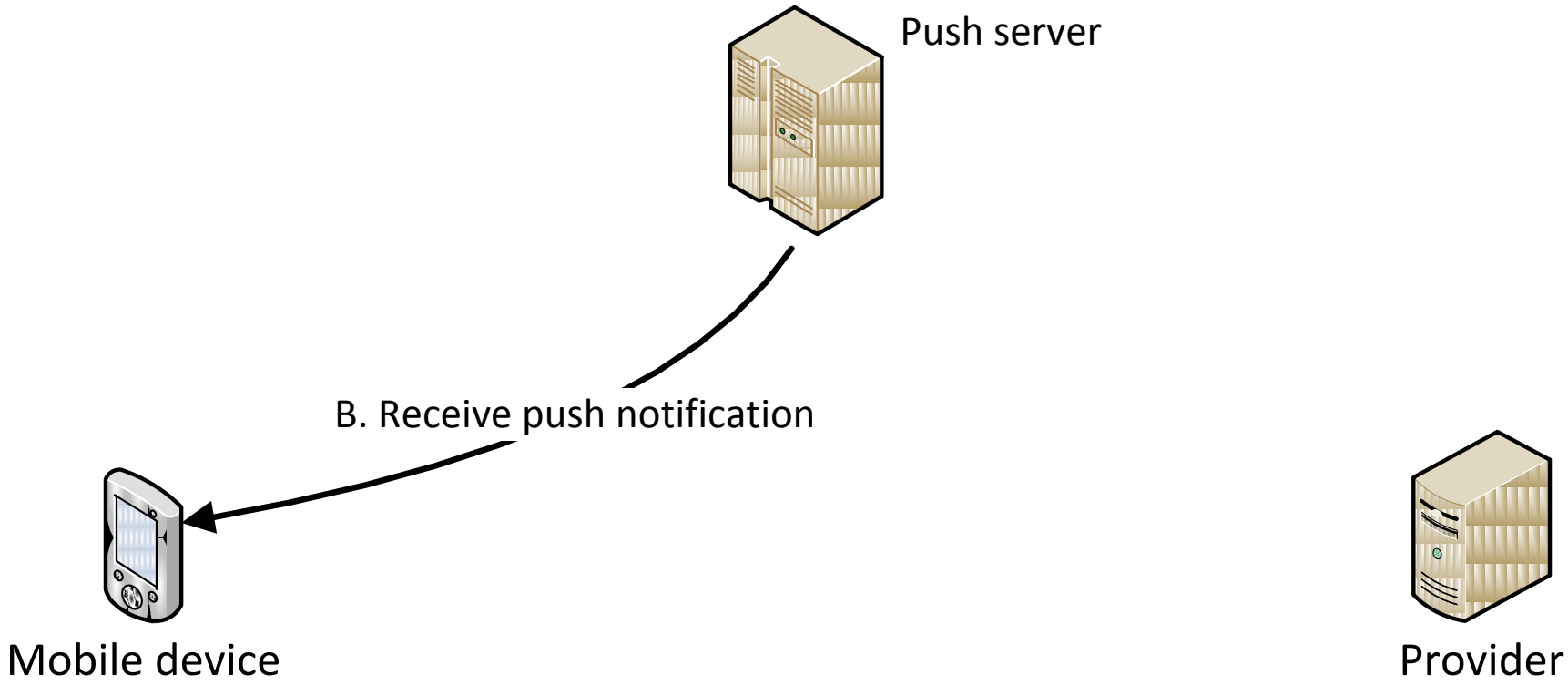


Provider

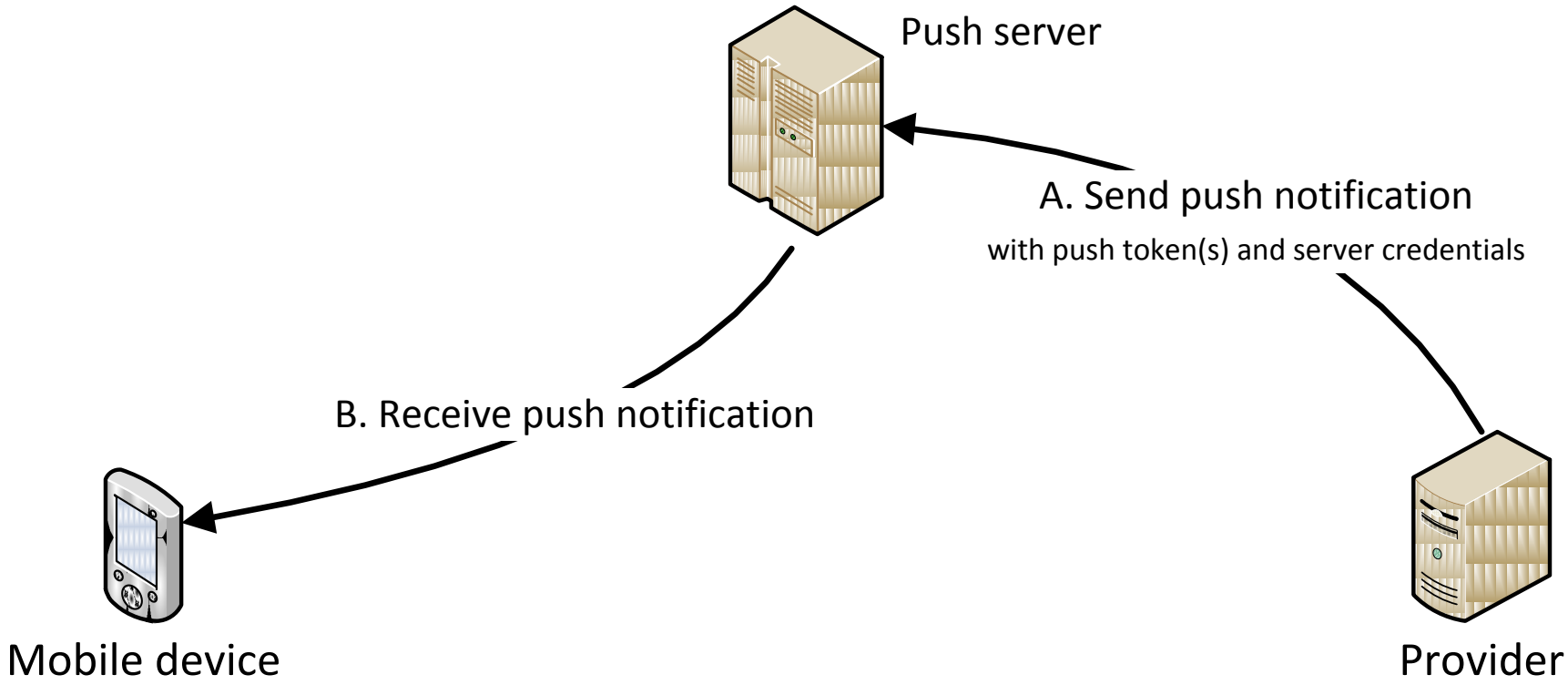
Pushes… how do they work?



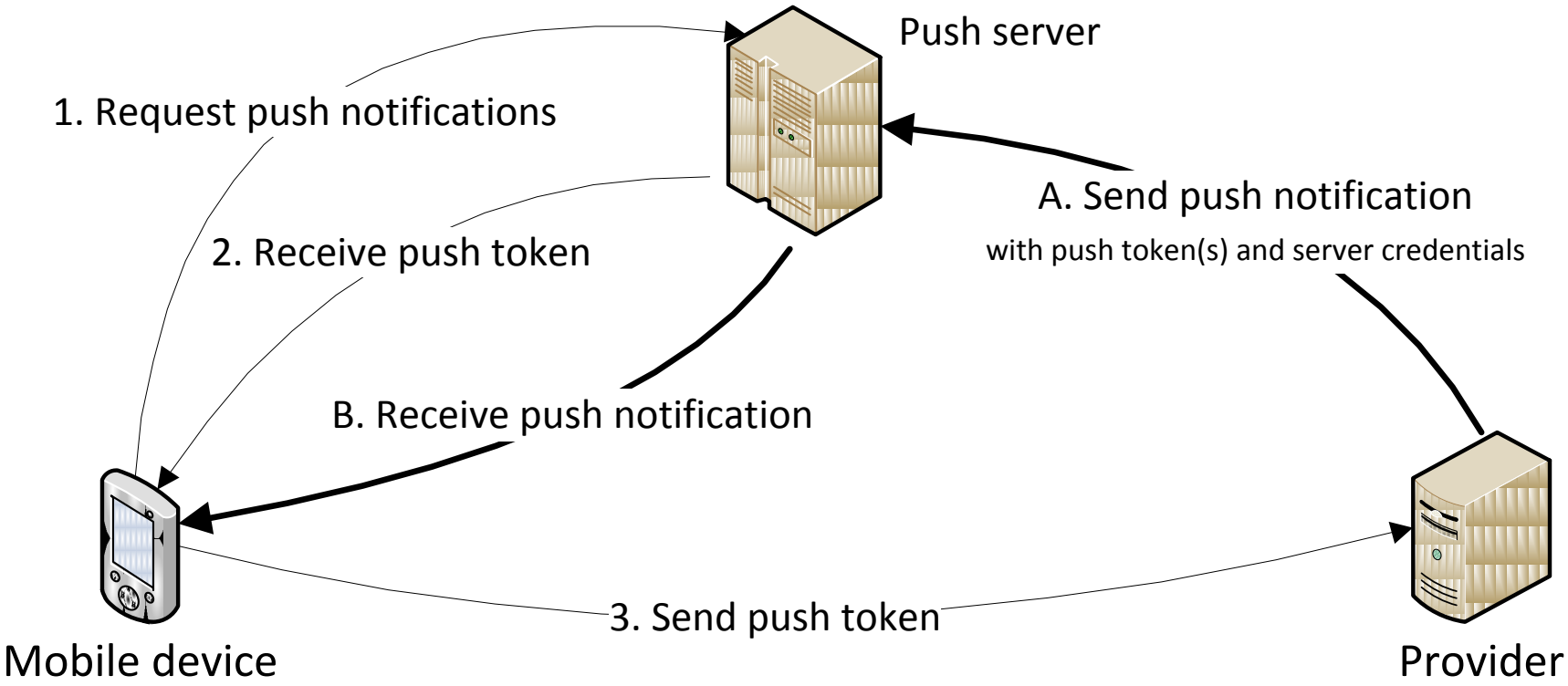
Pushes… how do they work?



Pushes… how do they work?



Pushes… how do they work?



Comparison between iOS and Android

iOS (Apple Push Notifications)

- Since iOS 3.0 (2009)
  - On Mac OS X since Lion
- Payload up to 256 bytes
- Binary protocol for providers
- JSON format
- Handled by system if app is not running
  - Alert, badge or sound
- Configurable via system settings

Android (C2DM)

- Since 2.2 (2010), beta
- Payload up to 1024 bytes
- HTTP protocol for providers
- Any format
- App can be started if not already running
- Configurable via app settings (if any)

Android Cloud to Device Messaging

Implementing a client

Overview

- API consists of sending and recieving Intents
  - Pushes themselves are broadcast Intents
- Push tokens are refreshed after some time
  - App receives new one automatically
- Security is based on custom application's permission
  - Prevents receiving pushes directed to different app

Step 0: service & application setup

- Using C2DM requires registration
  - Need to provide app's package name and Google Account which will be used to send pushes
  - Requests usually take few days to process

- App manifest must include appropriate persmissions

```
<permission android:name="com.example.myapp.permission.C2D_MESSAGE"
android:protectionLevel="signature" />
<uses-permission android:name="com.example.myapp.permission.C2D_MESSAGE" />

<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />

<uses-permission android:name="android.permission.INTERNET" />
```

Step 1: Requesting push token

- It is actually called **registration ID**
- Straightforward: just send a simple Intent

```
Intent regItent = new Intent("com.google.android.c2dm.intent.REGISTER");
regIntent.putExtra("app", PendingIntent.getBroadcast(this, 0, new Intent(), 0));
regIntent.putExtra("sender", emailOfSender);
startService(regIntent);
```

- `emailOfSender` is the Google Account used by push provider (the server) – e.g. `c2dm@exampleapp.com`
- C2DM API will automatically (and in the background) contact the push server if needed

Step 2 & 3: Handling push tokens

- Registration ID comes as a broadcast Intent
  - Hence we need a receiver for it

```
<receiver android:name=".C2DMRegistrationReceiver"
android:permission="com.google.android.c2dm.permission.SEND">
   <intent-filter>
      <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
      <category android:name="com.example.myapp" />
   </intent-filter>
</receiver>
```

- It will also receive unregistration events, should we ever request for them

```
public void onReceive(Context context, Intent intent) {
   if (intent.getStringExtra("unregistered") == null) {
      String registrationId = intent.getStringExtra("registration_id");
      sendToPushProvider(registrationId);
   }
}
```

Receiving a message

- Messages also come as broadcasted Intents

```
<receiver android:name=".C2DMMessageReceiver"
android:permission="com.google.android.c2dm.permission.SEND">
   <intent-filter>
      <action android:name="com.google.android.c2dm.intent.RECEIVE" />
      <category android:name="com.example.myapp" />
   </intent-filter>
</receiver>
```

- Payload can be fetched from Intent's extras

```
public void onReceive(Context context, Intent intent) {
   Bundle payload = intent.getExtras();
   int messageCount = Integer.parseInt(payload.getString(C2DM.MESSAGE_COUNT));
   String latestMsg = messageCount == 1 ?
                      Payload.getString(C2DM.LATEST_MESSAGE) : null;
   showNotification(messageCount, latestMsg);
}
```

Android Cloud to Device Messaging

Tips, tricks & caveats

Obtaining registration ID

- You can receive registration ID:
  - Multiple times – in response for every request
  - Again – if previous one is no longer valid
  - Without even requesting it – along with a push message
- You can also (albeit rarely) receive registration error
  - You should then try again with **exponential backoff**

- There are also other possible errors:
  - No Google Account on the phone
  - Wrong credentials on user's Google Account
  - Too many apps on the device using C2DM already

Those *in theory* should be displayed to the user (!).

Obtaining registration ID, part two

- Possible solution: `SharedPreferences` + `AlarmManager`

```
public void onReceive(Context context, Intent intent) { // C2DMRegistrationReceiver
   if (intent.getStringExtra("unregistered") == null) {
      String error = intent.getStringExtra("error");
      if (error == "SERVICE_NOT_AVAILABLE")
         scheduleC2dmRegistration();
      else { /* handle successful registration: delete C2DM_REG_TRY_INTERVAL */ }
   }
}

private void scheduleC2dmRegistration() {
   long interval = 2 * sharedPrefs.getLong(C2DM_REG_TRY_INTERVAL, 500); // ms;
   long nextTryTime = SystemClock.elapsedRealtime() + interval;

   Intent alarmIntent = new Intent(context, RegistrationRetryReceiver.class);
   PendingIntent alarm = PendingIntent.getBroadcast(context, 0, alarmIntent, 0);
   AlarmManager am = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
   am.set(AlarmManager.ELAPSED_REALTIME, nextTryTime, alarm);
}

// RegistrationRetryReceiver simply resends com.google.android.c2dm.intent.REGISTER
```

Sending registration ID to push provider

- Since you can receive registration ID at **any time**, you cannot count on your app actually running
  - Hence you have only short time to process the broadcast
- Uploading registration ID must be done in a **service**
  - In the background via `AsyncTask` or Java threading

```
private void handleRegistrationId(String regId) {
   Intent intent = new Intent(context, PushTokenUploadService.class);
   intent.putExtra(PUSH_TOKEN_EXTRA, regId);
   startService(intent);
}
```

Bonus points for…

- Exponential backoff on push token uploading
  - Same technique – just manage the interval separately
  - Alternative: request the token more often - you'll get the same one anyway
- Retrying registration when connectivity is detected
- Unregistering when you have the opportunity
  - E.g. when user disables C2DM in your app's settings

Related topics

- Actual handling of the messages (app-specific)
  - Status bar notifications
  - „Badges” through widgets
- Implementing the server
  - Storing the push tokens
  - Authenticating to Google push server
  - Sending messages
  - Handling errors (exponential backoff)

- For more information, refer directly to the docs:

  http://code.google.com/android/c2dm/

Thank you!