
Google™
11
I/O

# Evading Pirates and Stopping Vampires
## Using License Server, In App Billing, and App Engine

Dan Galpin and Trevor Johns
May 11, 2011

http://goo.gl/Q8SR7
#Android

# Pirates

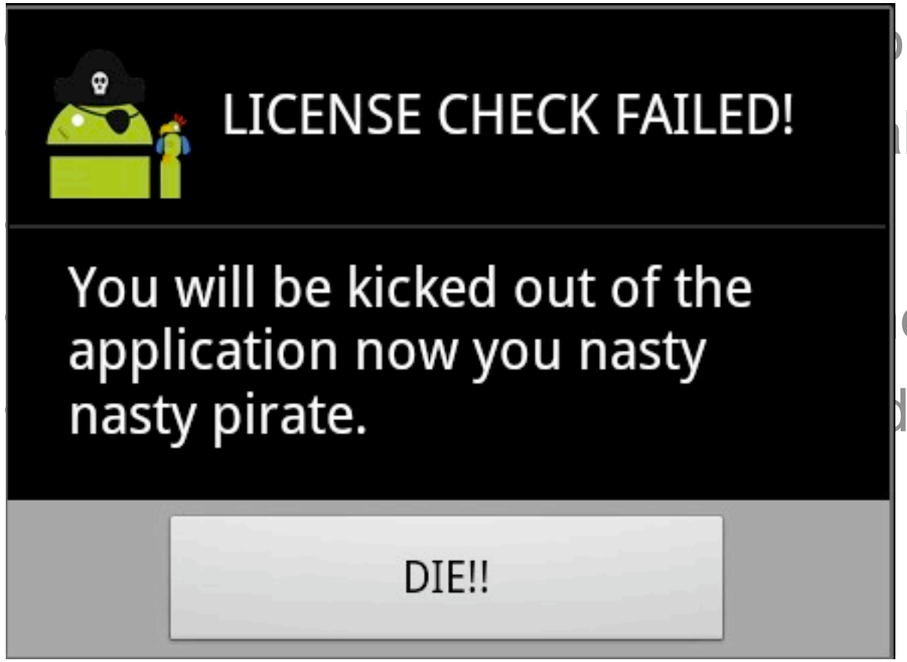
- Piracy is a reality on all platforms
- Dedicated pirates cannot be stopped

# Casual Pirates

Casual Pirates are potential customers.

- Make piracy inconvenient or challenging
- Use it as a marketing tool
- Allow for limited unlicensed gameplay
- Leverage new monetization models around IAP and Ads

# Casual Pirates




omers.

llenging

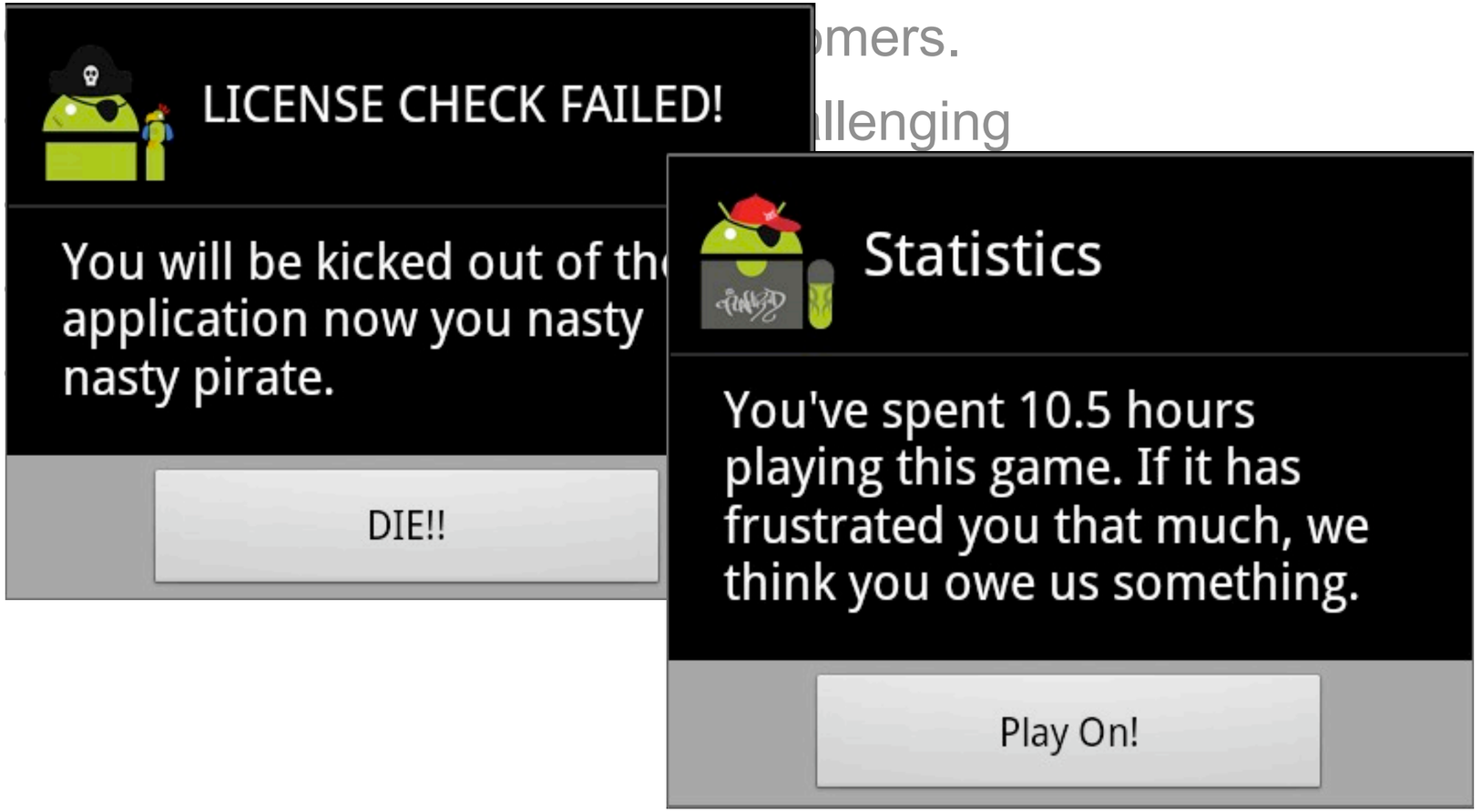
eplay

dels around IAP

# Casual Pirates

omers.

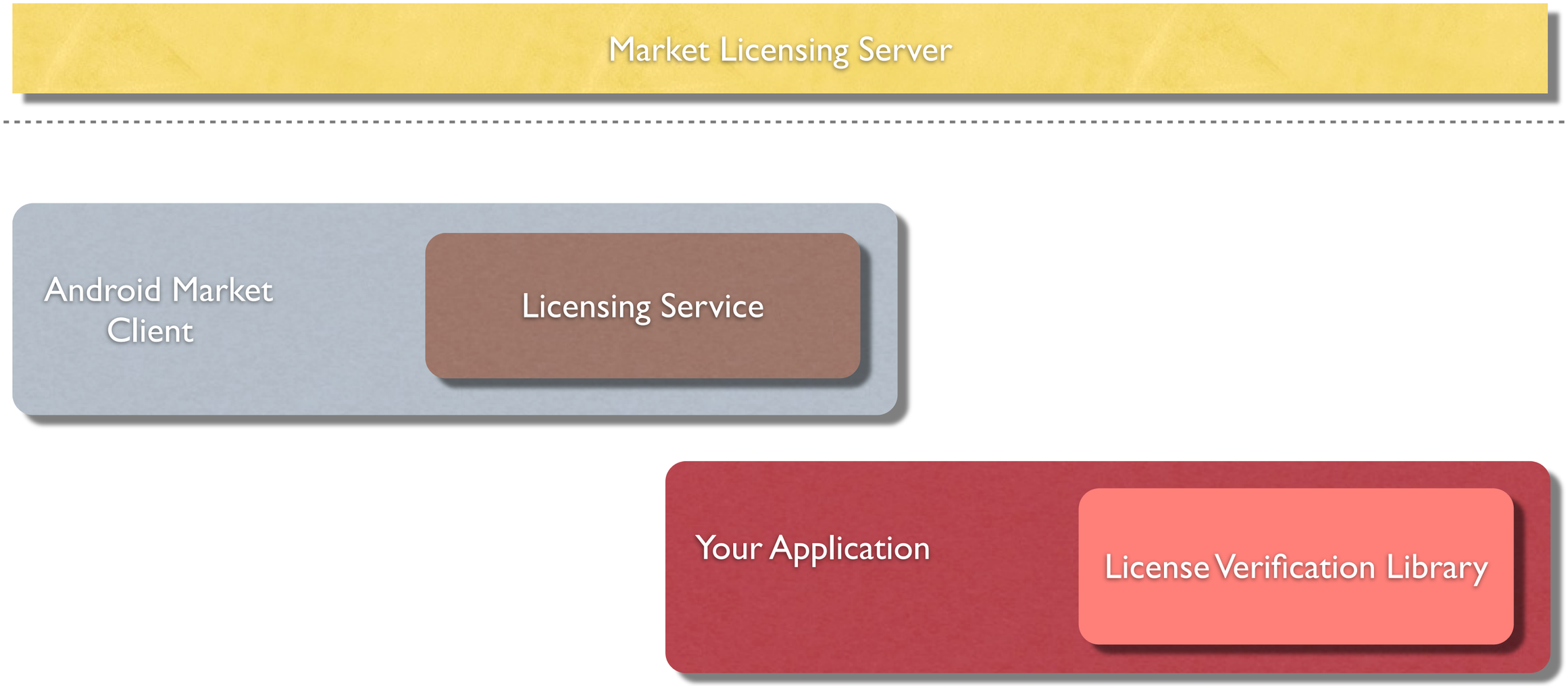
llenging



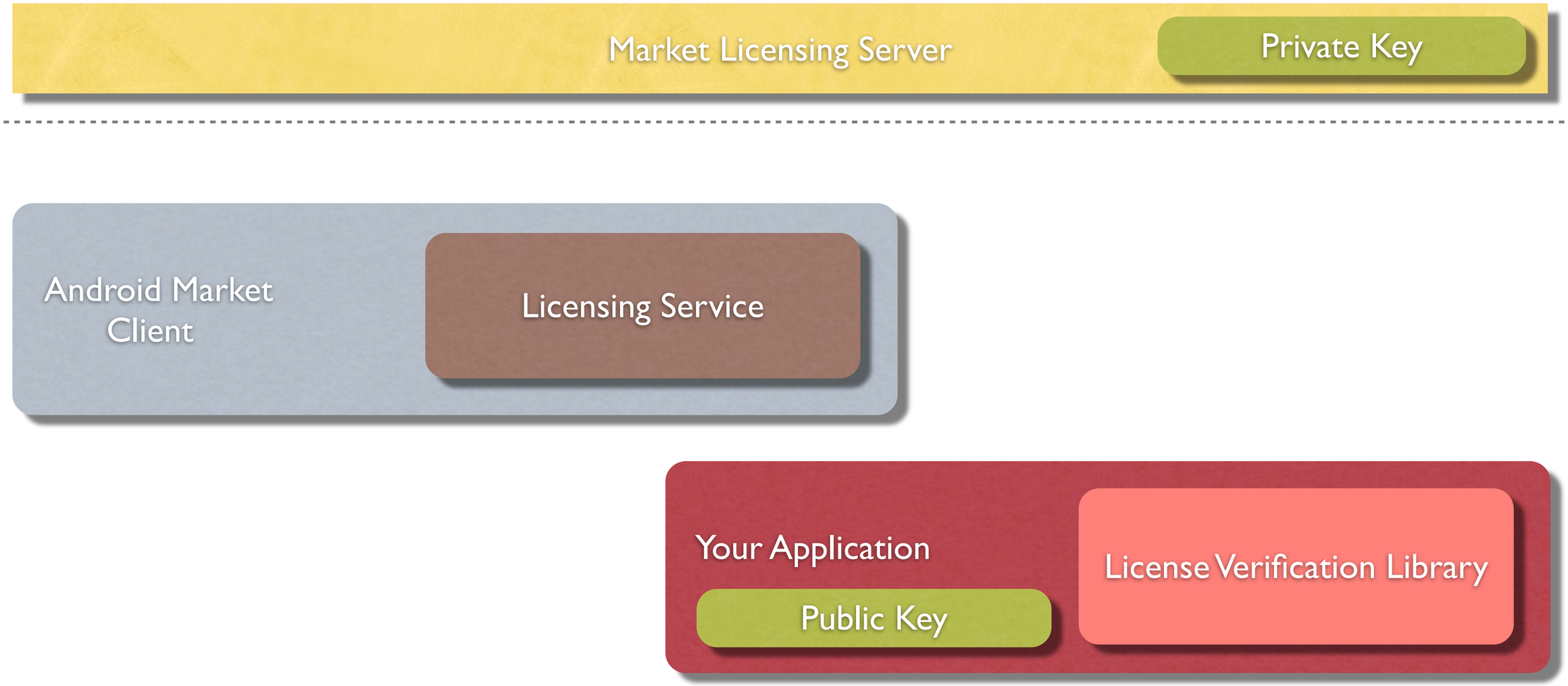

# License Verification: Your First Defense

- Google provides the License Verification Library
- Application determines how to enforce the policy
- Frequency of checks is managed by the client application policy
- Private key is stored with License Server,  public key is in application to verify signature
- Supported on Android 1.5 devices or above that have Android Market

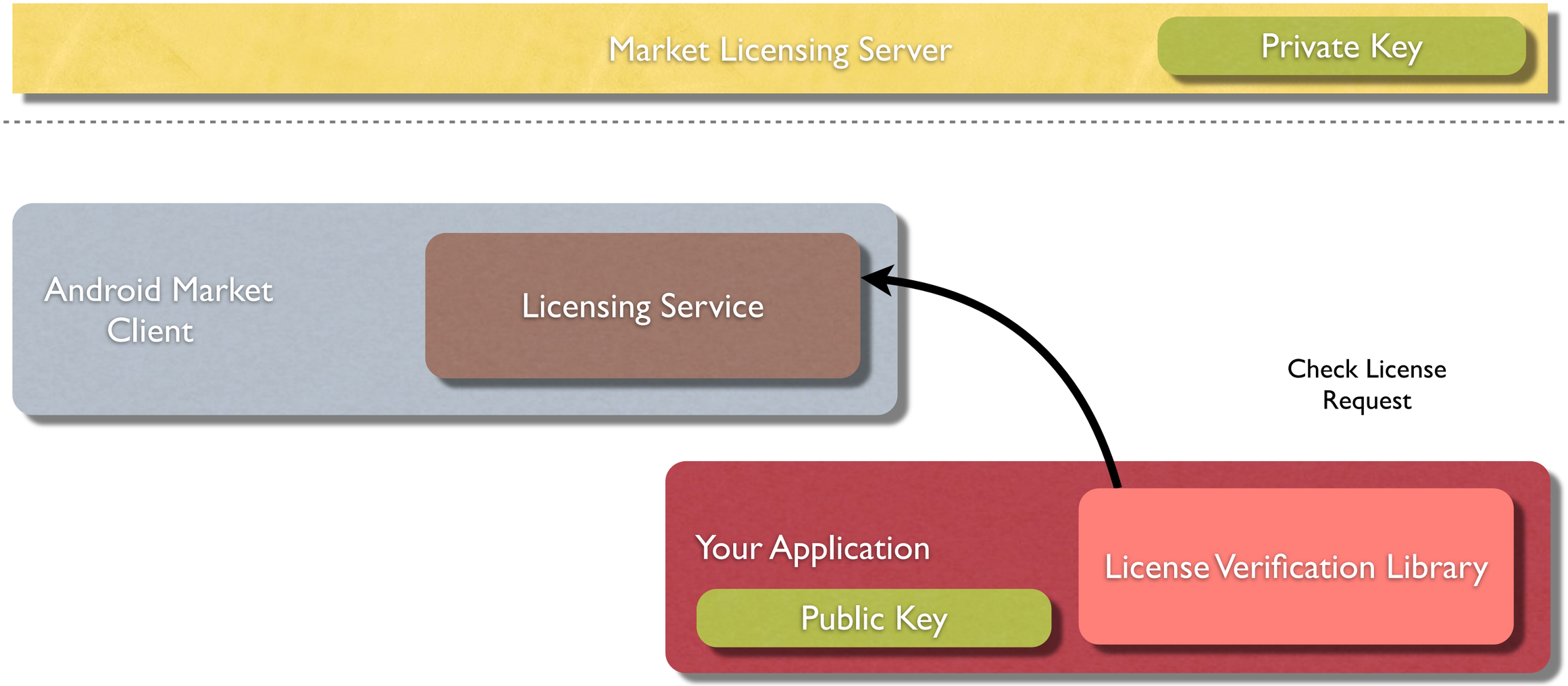
# Android Market Licensing - Client Verification



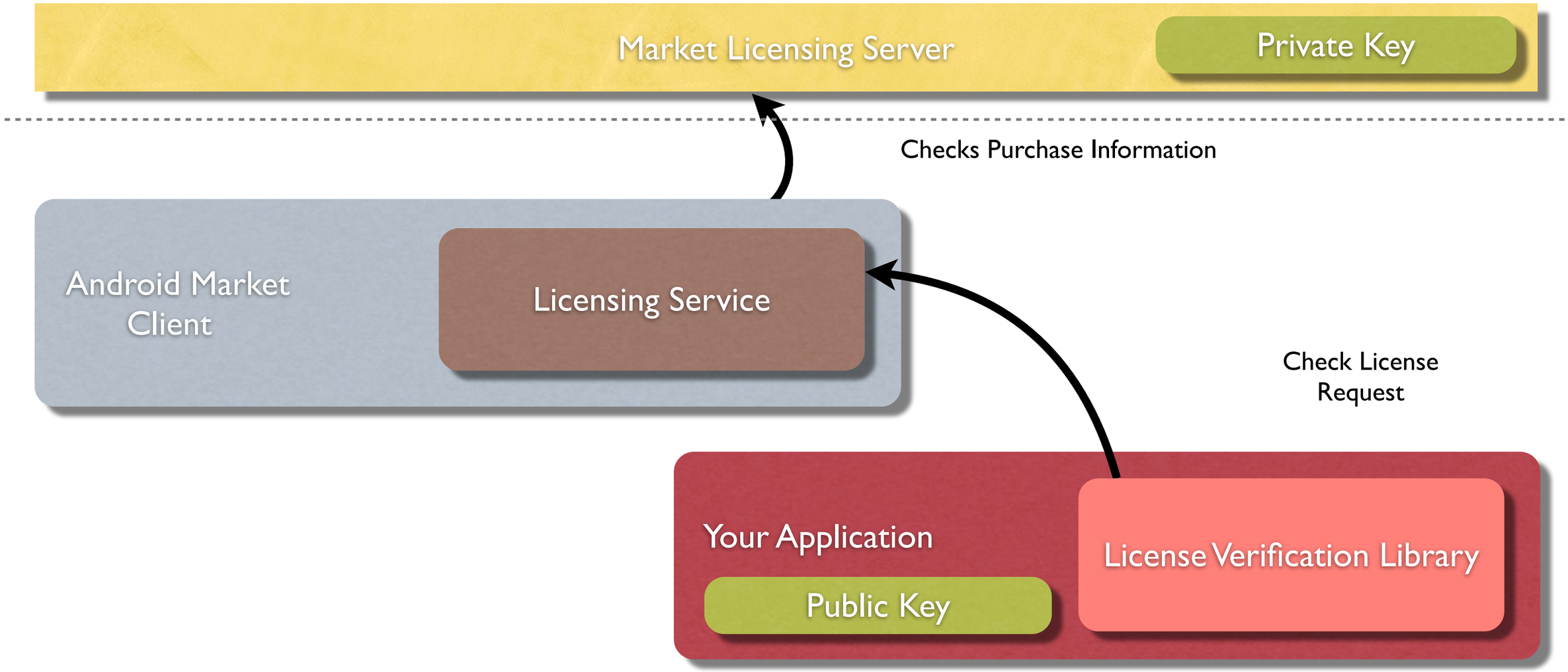
# Android Market Licensing - Client Verification



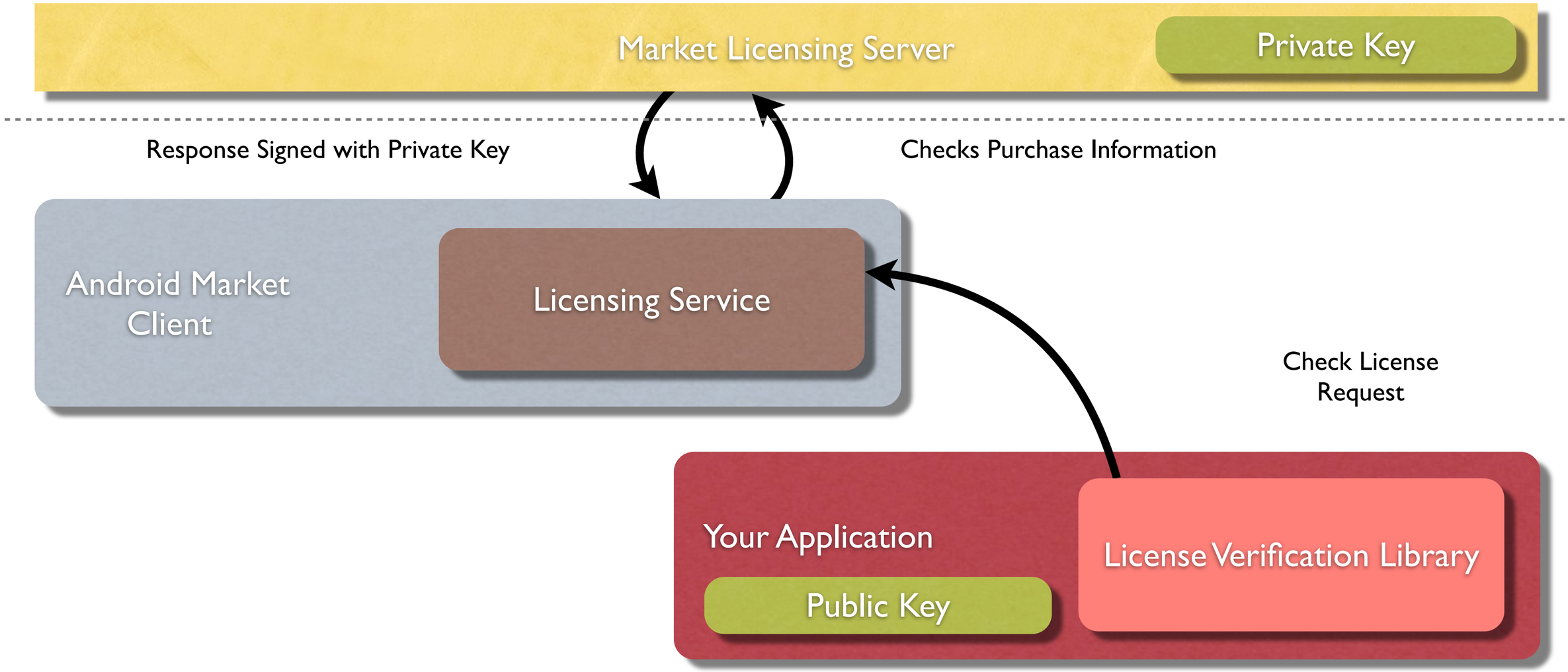
# Android Market Licensing - Client Verification



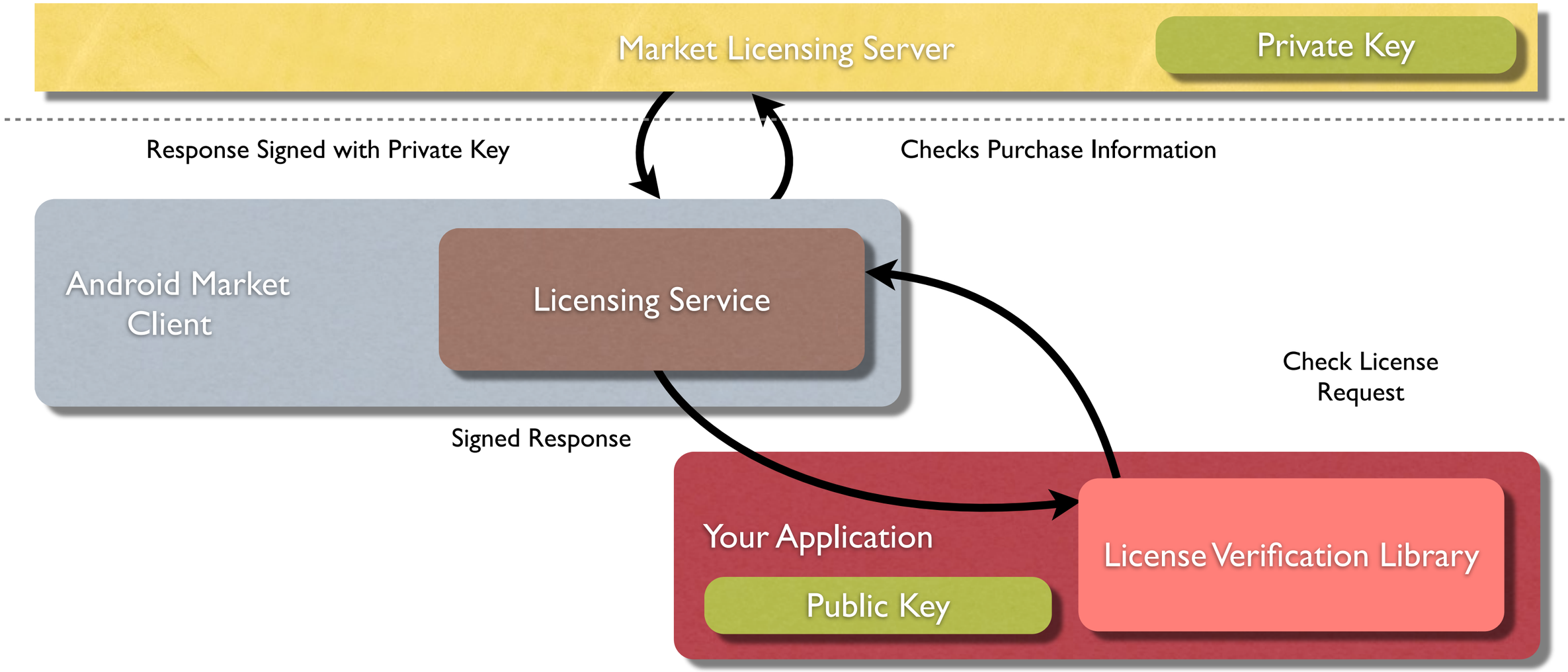
# Android Market Licensing - Client Verification



# Android Market Licensing - Client Verification



# Android Market Licensing - Client Verification



# Android Market Licensing - Client Attack


- Use tools to disassemble Dalvik/native code
- Alter the response from the library to ignore the server result and always return “licensed”
- Reassemble the application package
- Sign the package with an alternate signature

# LVL - Strengthening our Defenses

- Use an obfuscator
  http://proguard.sourceforge.net
  http://code.google.com/p/android-proguard-commandline/
- Modify the LVL code
  - Invocation of License Check and response-handling
  - Core LVL logic
  - Entry/Exit Points
- Make the application tamper-resistant

# LVL - Invocation/Response Handling

- Don’t invoke in a non-obfuscated function such as onCreate
  - Consider in a background thread
- Allow for a limited amount of gameplay when unlicensed
  - Consider delaying the result of a license failure
- Implement a policy that allows for multiple background retries if the network or server are unavailable before failing
  - Networks don’t always work even when the system thinks they should
- Invoke another activity rather than a dialog to inform the user of a validation failure

# LVL - Core Logic

## Original Sample Code

```
public void verify(PublicKey publicKey, int responseCode, String signedData, String
signature) {
    // ... Response validation code omitted for brevity ...
    switch (responseCode) {
        // In bytecode, LICENSED will be converted to the
        // constant 0x0
        case LICENSED:
        // NOT_LICENSED will be converted to the constant 0x1
        case NOT_LICENSED:
            handleResponse(LicenseResponse.NOT_LICENSED, data);
            break;
        // ... Extra response codes also removed for brevity ...
  }
}
```

## Example Updated Code

```
public void verify(PublicKey publicKey, int responseCode, String signedData, String
signature) {
  // ... Response validation code omitted for brevity …
  java.util.zip.CRC32 crc32 = new java.util.zip.CRC32();
  crc32.update(responseCode);
  int transformedResponseCode = crc32.getValue();

  // crc32(LICENSED) == 3523407757
  if (transformedResponseCode == 3523407757) {
    LicenseResponse limiterResponse =
        mDeviceLimiter.isDeviceAllowed(userId);
    handleResponse(limiterResponse, data);
  }
  // ... put unrelated application code here ...
  // crc32(NOT_LICENSED) == 2768625435
  if (transformedResponseCode == 2768625435) {
    userIsntLicensed();
  }
}
```

LVL Core Logic

Orig
publ
sign
/
s
}
}

## Example Updated Code

```
public void verify(PublicKey publicKey, int responseCode, String
signature) {
  // ... Response validation code omitted for brevity …
  java.util.zip.CRC32 crc32 = new java.util.zip.CRC32();
  crc32.update(responseCode);
  int transformedResponseCode = crc32.getValue();

  // crc32(LICENSED) == 3523407757
  if (transformedResponseCode == 3523407757) {
    LicenseResponse limiterResponse =
        mDeviceLimiter.isDeviceAllowed(userId);
    handleResponse(limiterResponse, data);
  }
  // ... put unrelated application code here ...
  // crc32(NOT_LICENSED) == 2768625435
  if (transformedResponseCode == 2768625435) {
    userIsntLicensed();
  }
}
```

Compute response code value using a hash function

ing

# LVL Core Logic

Orig
publ
sign
/
s
}
}

## Example Updated Code

```
public void verify(PublicKey publicKey, int responseCode, String
signature) {
  // ... Response validation code omitted for brevity …
  java.util.zip.CRC32 crc32 = new java.util.zip.CRC32();
  crc32.update(responseCode);
  int transformedResponseCode = crc32.getValue();

  // crc32(LICENSED) == 3523407757
  if (transformedResponseCode == 3523407757) {
    LicenseResponse limiterResponse =
        mDeviceLimiter.isDeviceAllowed(userId);
    handleResponse(limiterResponse, data);
  }
  // ... put unrelated application code here ...
  // crc32(NOT_LICENSED) == 2768625435
  if (transformedResponseCode == 2768625435) {
    userIsntLicensed();
  }
}
```

Compute response code value using a hash function

Use separate if statements separated by unrelated application code

ing

# LVL - Tamper Resistance

- Check that your application signature matches

```
Signature sigs[] = getPackageManager().getPackageInfo(getPackageName(),
PackageManager.GET_SIGNATURES).signatures;
if ( sigs[0].hashCode() == mySignatureHash )
```

# LVL - Tamper Resistance

- Check that your application signature matches

```
Signature sigs[] = getPackageManager().getPackageInfo(getPackageName(),
PackageManager.GET_SIGNATURES).signatures;
if ( sigs[0].hashCode() == mySignatureHash )
```

- Make sure your application is not debuggable

```
boolean isDebuggable =  ( 0 != ( getApplicationInfo().flags &=
ApplicationInfo.FLAG_DEBUGGABLE ) );
```

# LVL - Tamper Resistance

- Check that your application signature matches

```
Signature sigs[] = getPackageManager().getPackageInfo(getPackageName(),
PackageManager.GET_SIGNATURES).signatures;
if ( sigs[0].hashCode() == mySignatureHash )
```

- Make sure your application is not debuggable

```
boolean isDebuggable =  ( 0 != ( getApplicationInfo().flags &=
ApplicationInfo.FLAG_DEBUGGABLE ) );
```

- CRC code files and compare

```
zf = new ZipFile(getApplicationInfo().sourceDir);
          ZipEntry ze = zf.getEntry("classes.dex");
          if ( null != ze ) {
              ze.getCrc();
          }
```

# LVL - Tamper Resistance

- Check that your application certificate matches

```
public Certificate[] GetApplicationCertificates() {
    InputStream is = null;
    try {
        JarFile jf = new JarFile(getApplicationInfo().sourceDir);
        JarEntry je = jf.getJarEntry("classes.dex");
        is = jf.getInputStream(je);
        System.out.println("Got InputStream");
        while ((is.read()) != -1) {}  // whole stream is read
        return je.getCertificates();
    } catch (Exception e) {
    } finally {
        if ( null != is) {
            try {  is.close(); }
           catch (IOException e) { e.printStackTrace(); }
        }
    }
    return null;
}
```

# LVL - Tamper Resistance - Adding Reflection

- Our previous example, which gets the hash of the first signature

```
Signature sigs[] = getPackageManager().getPackageInfo(getPackageName(),
PackageManager.GET_SIGNATURES).signatures;
if ( sigs[0].hashCode() == mySignatureHash )
```

# LVL - Tamper Resistance - Adding Reflection

- Our previous example, which gets the hash of the first signature

```
Signature sigs[] = getPackageManager().getPackageInfo(getPackageName(),
PackageManager.GET_SIGNATURES).signatures;
if ( sigs[0].hashCode() == mySignatureHash )
```

- Using reflection

```
String getPM = new String(Base64.decode("Z2V0UGFja2FnZU1hbmFnZXI=\n", 0));
String getPI = new String(Base64.decode("Z2V0UGFja2FnZUluZm8=\n", 0));
Method gpmMethod = getClass().getMethod(getPM);
Object gpmObject = gpmMethod.invoke(this);
Method gpiMethod = gpmObject.getClass().getMethod(getPI, String.class, int.class);
Object pi = gpiMethod.invoke(gpmObject, getPackageName(), PackageManager.GET_SIGNATURES);
if (((PackageInfo)pi).signatures[0].hashCode() == mySignatureHash)
```

# LVL - Tamper Resistance - Adding JNI

```
jclass activityClass = (*pJNIEnv)->GetObjectClass(pJNIEnv, activityObject);
jmethodID getPackageManagerMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass, "getPackageManager", "()Landroid/
content/pm/PackageManager;");
jmethodID getPackageNameMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass, "getPackageName", "()Ljava/lang/
String;");
jobject packageManagerObject = (*pJNIEnv)->CallObjectMethod(pJNIEnv, activityObject, getPackageManagerMid);
jclass packageManagerClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/content/pm/PackageManager");
jmethodID getPackageInfoMid = (*pJNIEnv)->GetMethodID(pJNIEnv, packageManagerClass, "getPackageInfo", "(Ljava/
lang/String;I)Landroid/content/pm/PackageInfo;");
jstring packageNameString = (*pJNIEnv)->CallObjectMethod(pJNIEnv, activityObject, getPackageNameMid);
jfieldID getSignaturesFid = (*pJNIEnv)->GetStaticFieldID(pJNIEnv, packageManagerClass, "GET_SIGNATURES", "I");
jint GET_SIGNATURES = (*pJNIEnv)->GetStaticIntField(pJNIEnv, packageManagerClass, getSignaturesFid);
jobject packageInfoObject = (*pJNIEnv)->CallObjectMethod(pJNIEnv, packageManagerObject, getPackageInfoMid,
packageNameString, GET_SIGNATURES);
jclass packageInfoClass = (*pJNIEnv)->GetObjectClass(pJNIEnv, packageInfoObject);
jfieldID signatureFid = (*pJNIEnv)->GetFieldID(pJNIEnv, packageInfoClass,
          "signatures", "[Landroid/content/pm/Signature;");
jobject signatureArrayObject = (*pJNIEnv)->GetObjectField(pJNIEnv, packageInfoObject, signatureFid);
jint len = (*pJNIEnv)->GetArrayLength(pJNIEnv, signatureArrayObject);
jobject signatureObject = (*pJNIEnv)->GetObjectArrayElement(pJNIEnv, signatureArrayObject, 0);
jclass signatureClass = (*pJNIEnv)->GetObjectClass(pJNIEnv, signatureObject);
jmethodID hashCodeMid = (*pJNIEnv)->GetMethodID(pJNIEnv, signatureClass, "hashCode", "()I");
if ( (*pJNIEnv)->CallIntMethod(pJNIEnv, signatureObject, hashCodeMid) == mySignatureHash )
```

# LVL - Tamper Resistance - Adding JNI

```
jclass activityClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/app/Activity");
jmethodID getPackageManagerMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass, "getPackageManager",
"()Landroid/content/pm/PackageManager;");
jmethodID getPackageNameMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass, "getPackageName",
"()Ljava/lang/String;");
jclass packageManagerClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/content/pm/PackageManager");
jmethodID getPackageInfoMid = (*pJNIEnv)->GetMethodID(pJNIEnv, packageManagerClass, "getPackageInfo",
"(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;");
jfieldID getSignaturesFid = (*pJNIEnv)->GetStaticFieldID(pJNIEnv, packageManagerClass, "GET_SIGNATURES",
"I");
jint GET_SIGNATURES = (*pJNIEnv)->GetStaticIntField(pJNIEnv, packageManagerClass, getSignaturesFid);
jclass packageInfoClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/content/pm/PackageInfo");
jfieldID signatureFid = (*pJNIEnv)->GetFieldID(pJNIEnv, packageInfoClass,
         "signatures", "[Landroid/content/pm/Signature;");
jclass signatureClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/content/pm/Signature");
jmethodID hashCodeMid = (*pJNIEnv)->GetMethodID(pJNIEnv, signatureClass, "hashCode", "()I");
jint len = (*pJNIEnv)->GetArrayLength(pJNIEnv, signatureArrayObject);
jobject signatureObject = (*pJNIEnv)->GetObjectArrayElement(pJNIEnv, signatureArrayObject, 0);
jclass signatureClass = (*pJNIEnv)->GetObjectClass(pJNIEnv, signatureObject);
jmethodID hashCodeMid = (*pJNIEnv)->GetMethodID(pJNIEnv, signatureClass, "hashCode", "()I");
if ( (*pJNIEnv)->CallIntMethod(pJNIEnv, signatureObject, hashCodeMid) == mySignatureHash )
```

# LVL - Tamper Resistance - Adding JNI

```
jclass activityClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/app/Activity");
jmethodID getPackageManagerMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass, "getPackageManager",
"()Landroid/content/pm/PackageManager;");
jmethodID getPackageNameMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass, "getPackageName",
"()Ljava/lang/String;");
jclass packageManagerClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/content/pm/PackageManager");
jmethodID getPackageInfoMid = (*pJNIEnv)->GetMethodID(pJNIEnv, packageManagerClass, "getPackageInfo",
"(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;");
jfieldID getSignaturesFid = (*pJNIEnv)->GetStaticFieldID(pJNIEnv, packageManagerClass, "GET_SIGNATURES",
"I");
jint GET_SIGNATURES = (*pJNIEnv)->GetStaticIntField(pJNIEnv, packageManagerClass, getSignaturesFid);
jclass packageInfoClass = (*pJNIEnv)->FindClass(pJNIEnv, "android/content/pm/PackageInfo");
jfieldID signatureFid = (*pJNIEnv)->GetFieldID(pJNIEnv, packageInfoClass,
        "signatures", "[Landroid/content/pm/Signature;");
```

```
jobject packageManagerObject = (*pJNIEnv)->CallObjectMethod(pJNIEnv, activityObject, getPackageManagerMid);
jstring packageNameString = (*pJNIEnv)->CallObjectMethod(pJNIEnv, activityObject, getPackageNameMid);
jobject packageInfoObject = (*pJNIEnv)->CallObjectMethod(pJNIEnv, packageManagerObject, getPackageInfoMid,
packageNameString, GET_SIGNATURES);
jobject signatureArrayObject = (*pJNIEnv)->GetObjectField(pJNIEnv, packageInfoObject, signatureFid);
jint len = (*pJNIEnv)->GetArrayLength(pJNIEnv, signatureArrayObject);
jobject signatureObject = (*pJNIEnv)->GetObjectArrayElement(pJNIEnv, signatureArrayObject, 0);
if ( (*pJNIEnv)->CallIntMethod(pJNIEnv, signatureObject, hashCodeMid) == mySignatureHash )
```

# LVL - Tamper Resistance - Obfuscating

- The latest version of the Android tools has built-in support for  Proguard and the LVL
  http://developer.android.com/guide/developing/tools/proguard.html
- With one minor change to the LVL, you can obfuscate much better (remove ILicensingService from the Proguard cfg)

# LVL - Tamper Resistance - Obfuscating

- The latest version of the Android tools has built-in support for  Proguard and the LVL
  http://developer.android.com/guide/developing/tools/proguard.html
- With one minor change to the LVL, you can obfuscate much better (remove ILicensingService from the Proguard cfg)

**Change:**

```
boolean bindResult = mContext.bindService(
 new Intent(ILicensingService.class.getName()),
```

**To:**

```
boolean bindResult = mContext.bindService(
 new Intent("com.android.vending.licensing.ILicensingService")
```

# LVL - Tamper Resistance - Obfuscating

- The latest version of the Android tools has built-in support for Proguard and the LVL
  http://developer.android.com/guide/developing/tools/proguard.html
- With one minor change to the LVL, you can obfuscate much better (remove ILicensingService from the Proguard cfg)

**Change:**

```
boolean bindR
 new Intent(I
```

**To:**

```
boolean bindResult = mContext.bindService(
 new Intent("com.android.vending.licensing.ILicensingService")
```

Consider performing a transform on “com.android.vending.licensing.ILicensingService” to make the code less obvious.

# LVL - Tamper Resistance - NDK

- Put checks in C/C++ code

```
JNIEXPORT jlong JNICALL getClassesCRC (JNIEnv * pJNIEnv, jobject activityObject) {
  jlong retvalue = -1;
  jclass activityClass = (*pJNIEnv)->GetObjectClass(pJNIEnv, activityObject);
  jmethodID getApplicationInfoMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass,
     "getApplicationInfo", "()Landroid/content/pm/ApplicationInfo;");
  jobject applicationInfo = (*pJNIEnv)->CallObjectMethod(pJNIEnv, activityObject, getApplicationInfoMid);
  jfieldID pathField = (*pJNIEnv)->GetFieldID(pJNIEnv, (*pJNIEnv)->GetObjectClass(pJNIEnv, applicationInfo),
    "sourceDir", "Ljava/lang/String;");
  jobject strApplicationPath = (*pJNIEnv)->GetObjectField(pJNIEnv, applicationInfo, pathField);
  const jbyte * applicationPath = (*pJNIEnv)->GetStringUTFChars(pJNIEnv, strApplicationPath, NULL);
  unzFile uf =  unzOpen(applicationPath);
  if (unzLocateFile(uf,"classes.dex",CASESENSITIVITY)==UNZ_OK) {
    unz_file_info * fileInfo = malloc(sizeof(unz_file_info));
  if ( UNZ_OK == unzGetCurrentFileInfo (uf, fileInfo, NULL, 0, NULL, 0, NULL, 0) ) {
    retvalue = fileInfo->crc;
    }
  }
  unzClose(uf);
  (*pJNIEnv)->ReleaseStringUTFChars(pJNIEnv, strApplicationPath, applicationPath);
  return retvalue;
}
```

# LVL - Tamper Resistance - NDK

```
JNIEXPORT jlong JNICALL getClassesCRC (JNIEnv * pJNIEnv, jobject activityObject) {
  jlong retvalue = -1;
  jclass activityClass = (*pJNIEnv)->GetObjectClass(pJNIEnv, activityObject);
  jmethodID getApplicationInfoMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass,
     "getApplicationInfo", "()Landroid/content/pm/ApplicationInfo;");
  jobject applicationInfo =
    (*pJNIEnv)->CallObjectMethod(pJNIEnv, activityObject, getApplicationInfoMid);
  jfieldID pathField = (*pJNIEnv)->GetFieldID(pJNIEnv,
    (*pJNIEnv)->GetObjectClass(pJNIEnv, applicationInfo),
     "sourceDir", "Ljava/lang/String;");
  jobject strApplicationPath =
     (*pJNIEnv)->GetObjectField(pJNIEnv, applicationInfo, pathField);
  const jbyte * applicationPath =
     (*pJNIEnv)->GetStringUTFChars(pJNIEnv, strApplicationPath, NULL);
    }
  }
  unzClose(uf);
  (*pJNIEnv)->ReleaseStringUTFChars(pJNIEnv, strApplicationPath, applicationPath);
  return retvalue;
}
```

# LVL - Tamper Resistance - NDK

```
JNIEXPORT jlong JNICALL getClassesCRC (JNIEnv * pJNIEnv, jobject activityObject) {
  jlong retvalue = -1;
  jclass activityClass = (*pJNIEnv)->GetObjectClass(pJNIEnv, activityObject);
  jmethodID getApplicationInfoMid = (*pJNIEnv)->GetMethodID(pJNIEnv, activityClass,
     "getApplicationInfo", "()Landroid/content/pm/ApplicationInfo;");
  jobject applicationInfo =
    (*pJNIEnv)->C
  jfieldID pathFi
    (*pJNIEnv)->G
     "sourceDir",
  jobject strAppl
     (*pJNIEnv)->
  const jbyte * a
     (*pJNIEnv)->
    }
  }
  unzClose(uf);
  (*pJNIEnv)->Relea
  return retvalue;
}
```

```
  unzFile uf =  unzOpen(applicationPath);
  if (unzLocateFile(uf,"classes.dex",CASESENSITIVITY)==UNZ_OK) {
    unz_file_info * fileInfo = malloc(sizeof(unz_file_info));
  if ( UNZ_OK == unzGetCurrentFileInfo
    (uf, fileInfo, NULL, 0, NULL, 0, NULL, 0) ) {
       retvalue = fileInfo->crc;
    }
  }
  unzClose(uf);
  (*pJNIEnv)->ReleaseStringUTFChars(pJNIEnv,
     strApplicationPath, applicationPath);
  return retvalue;
}
```

# LVL - Tamper Resistance - NDK

- Get native signatures from class file with javap -s
  - `javap -s -private com.android.gl2jni.GL2JNILib`

```
static JNINativeMethod methods[] = {
  { "init", "(II)V", (void*) nativeGameInit },
  { "step", "()V", (void*) nativeStep }
};
```

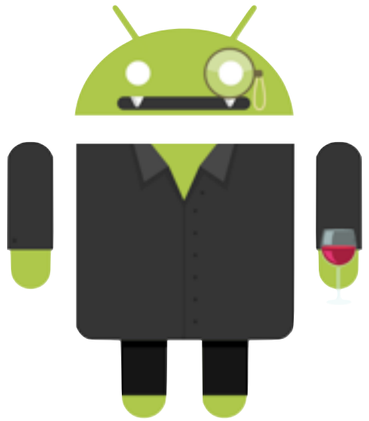
```
  char* className = "com/android/gl2jni/GL2JNILib";
  jclass clazz = env->FindClass(className);
  if (clazz == NULL) {
    __android_log_print(ANDROID_LOG_ERROR, "AwesomeGame",
      "Native registration unable to find class '%s'\n", className);
  return JNI_FALSE;
  }
  if (env->RegisterNatives(clazz, gMethods, numMethods) < 0) {
    __android_log_print(ANDROID_LOG_ERROR, "AwesomeGame",
      "RegisterNatives failed for '%s'\n", className);
  return JNI_FALSE;
  }
```

# LVL - Tamper Resistance - Advanced Ideas

- Store LVL and other application binaries as encrypted resources. Decrypt to the filesystem and use the class loader to call the LVL
- Make direct calls to the Binder interface, eliminating the entire RPC shell class

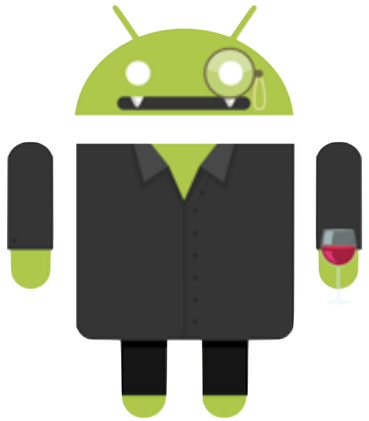
# When Pirates Become Vampires

# When Pirates Become Vampires

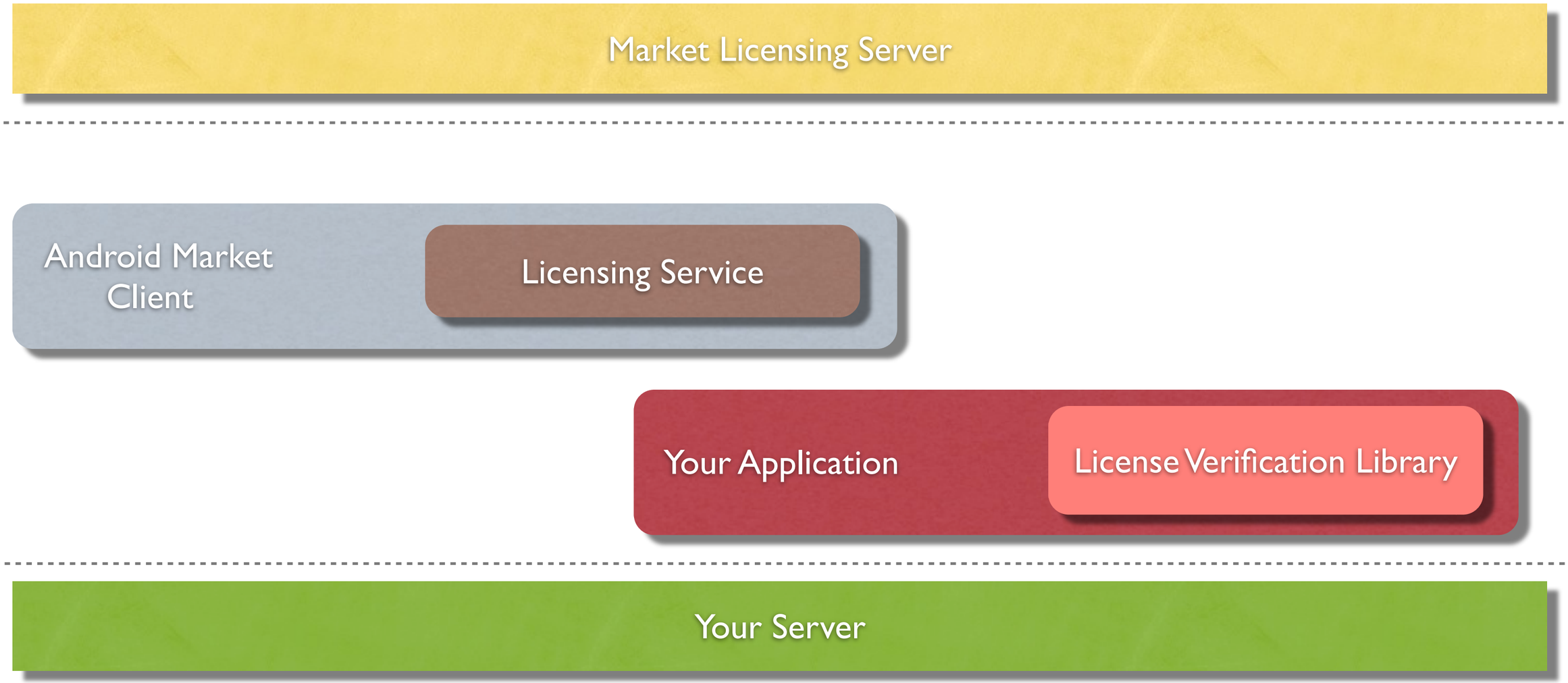


# When Pirates Become Vampires

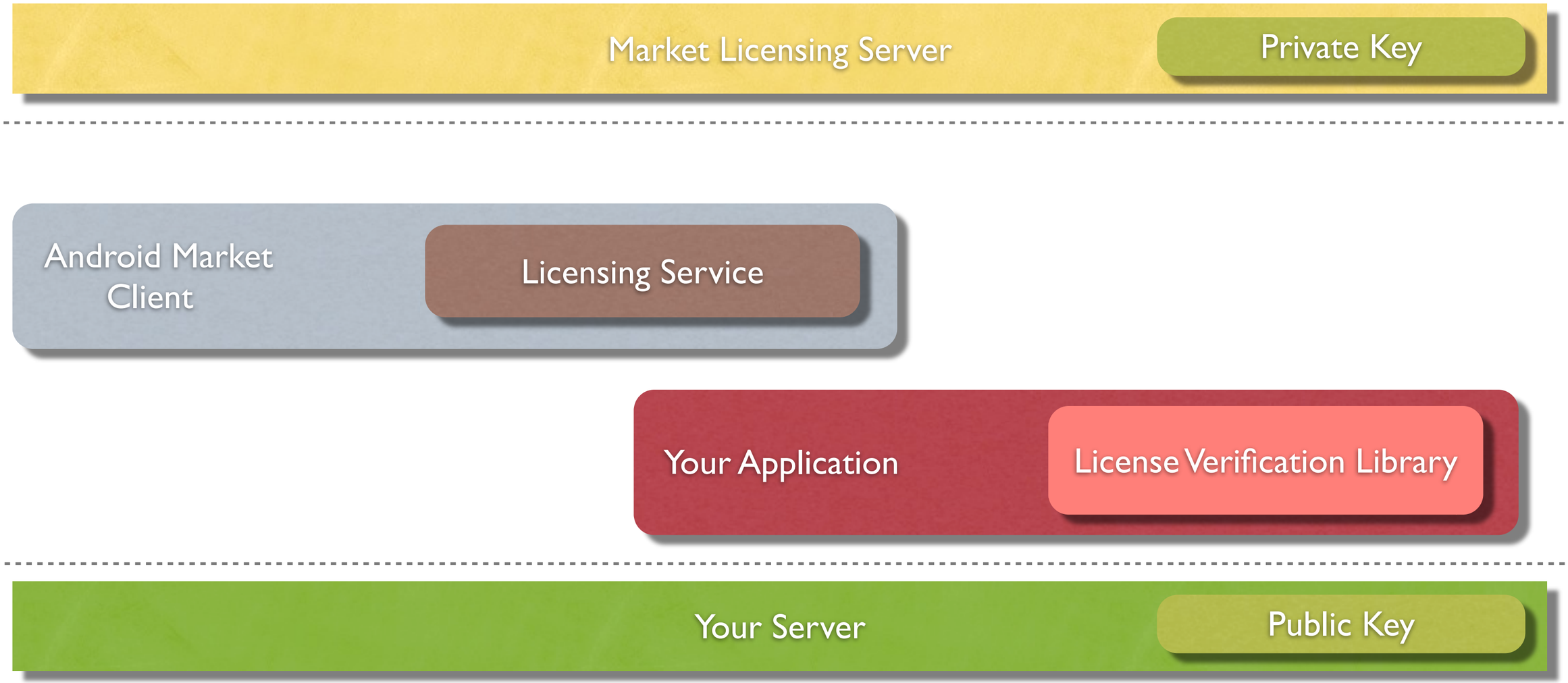
- Games often have server components
- Android games often have asset downloads
- Running an unlicensed title can turn a pirate into a vampire that feeds off your bandwidth
- Since you have a server component, you can stop vampires with server-side validation



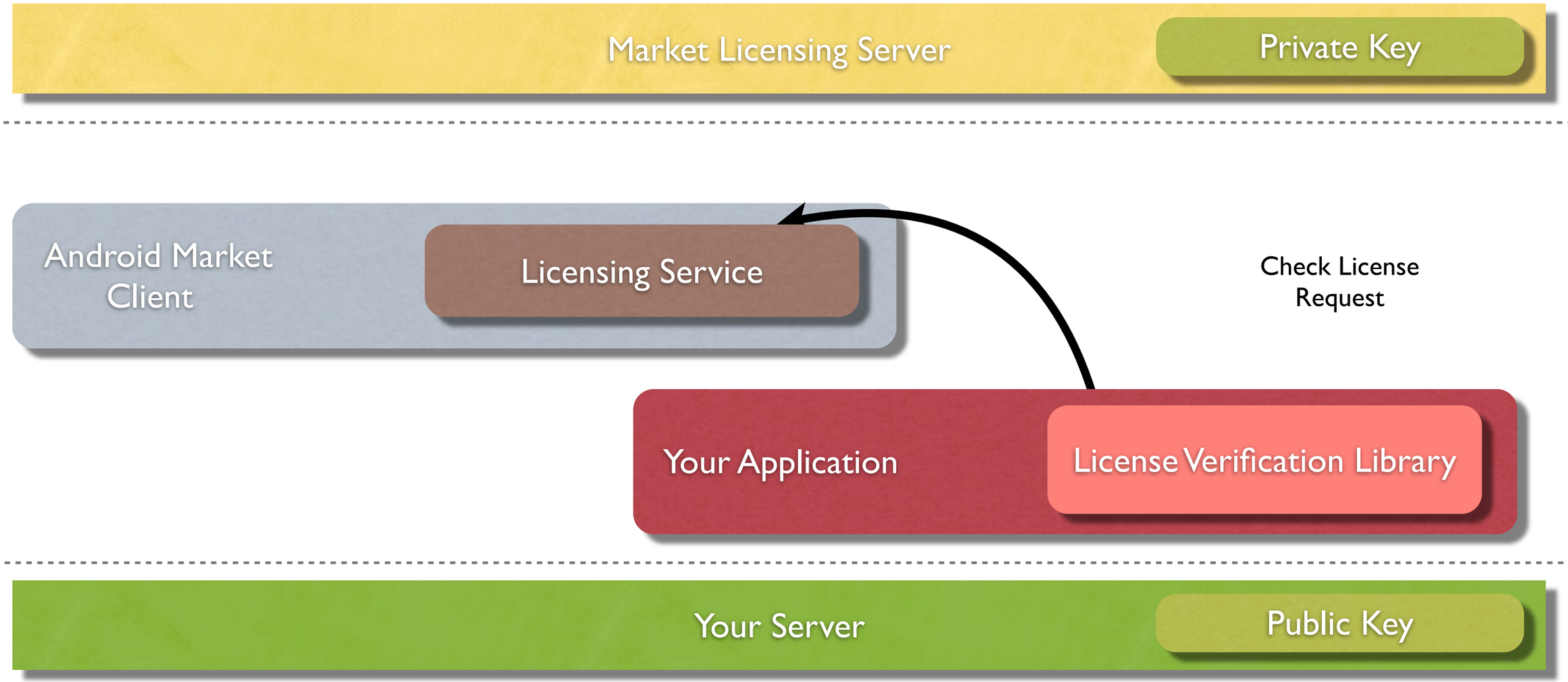
# Android Market Licensing - Server Validation



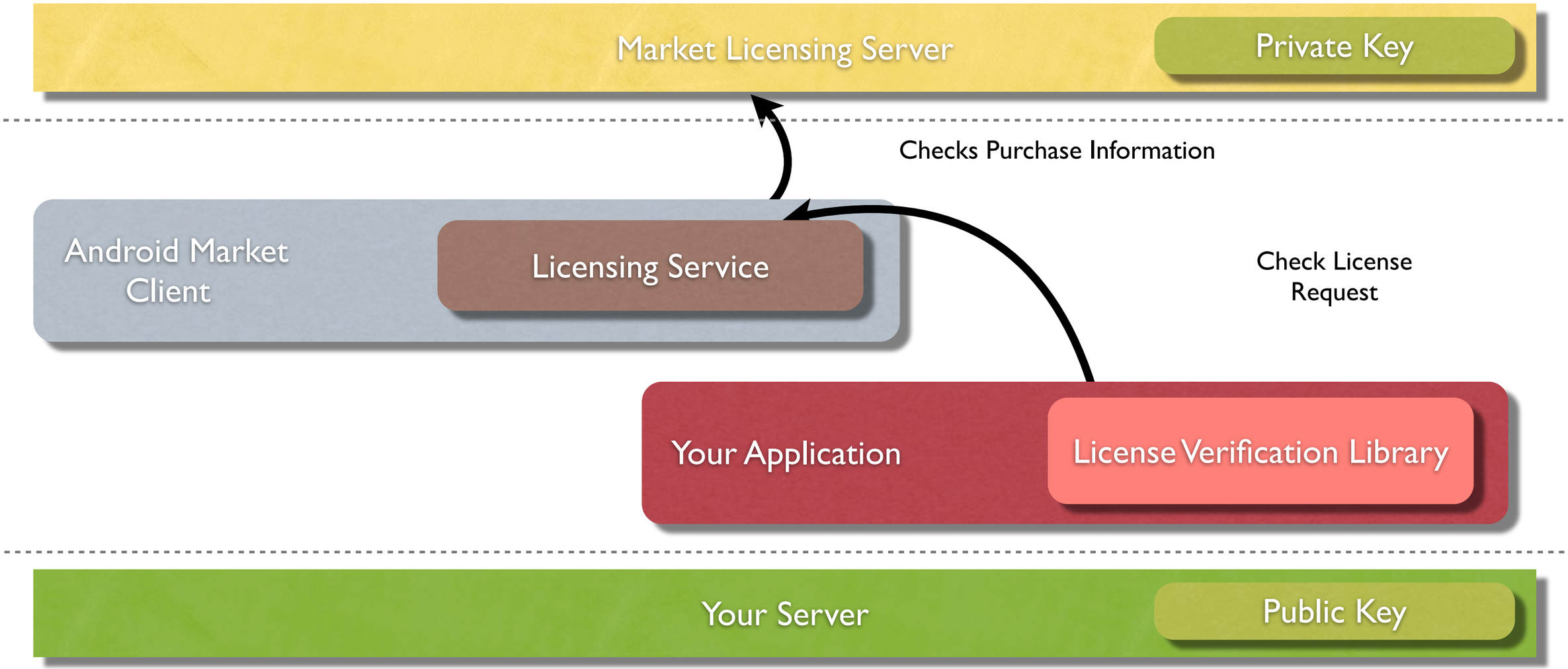
# Android Market Licensing - Server Validation



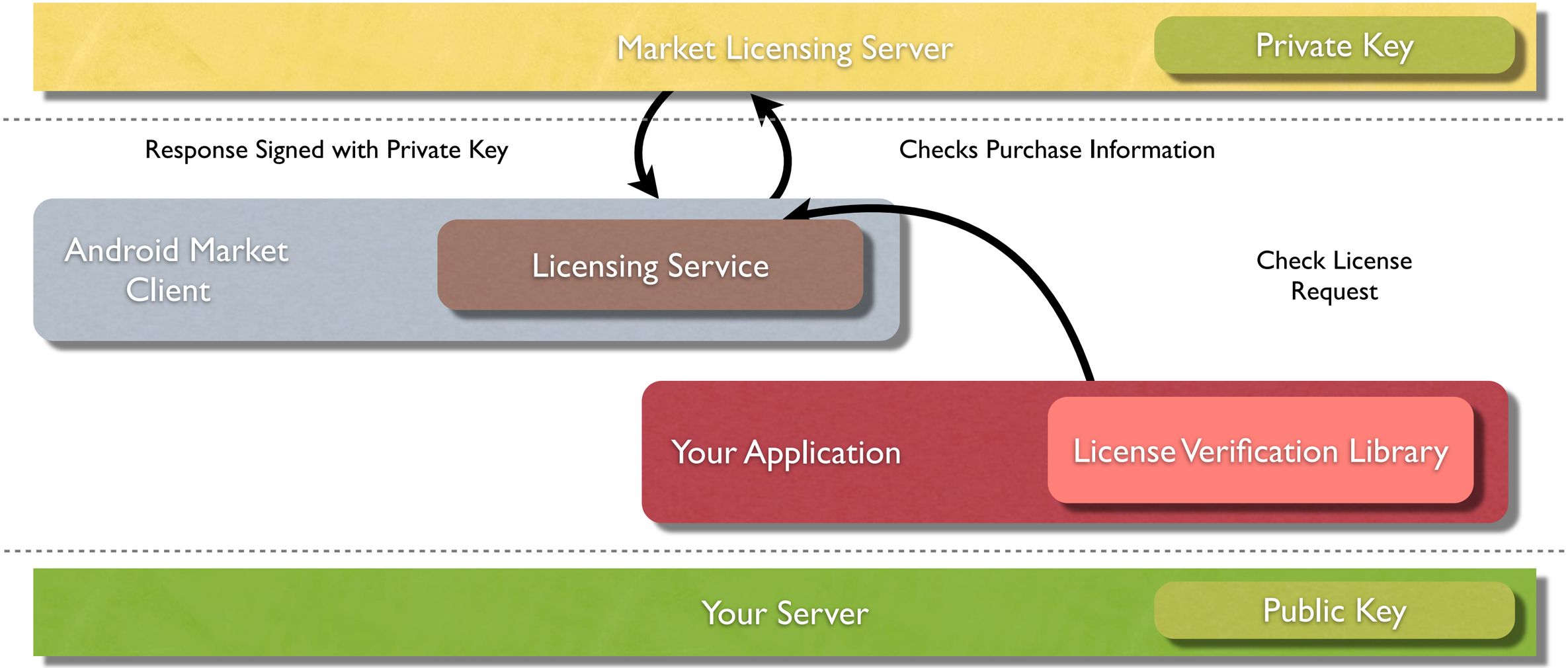
# Android Market Licensing - Server Validation



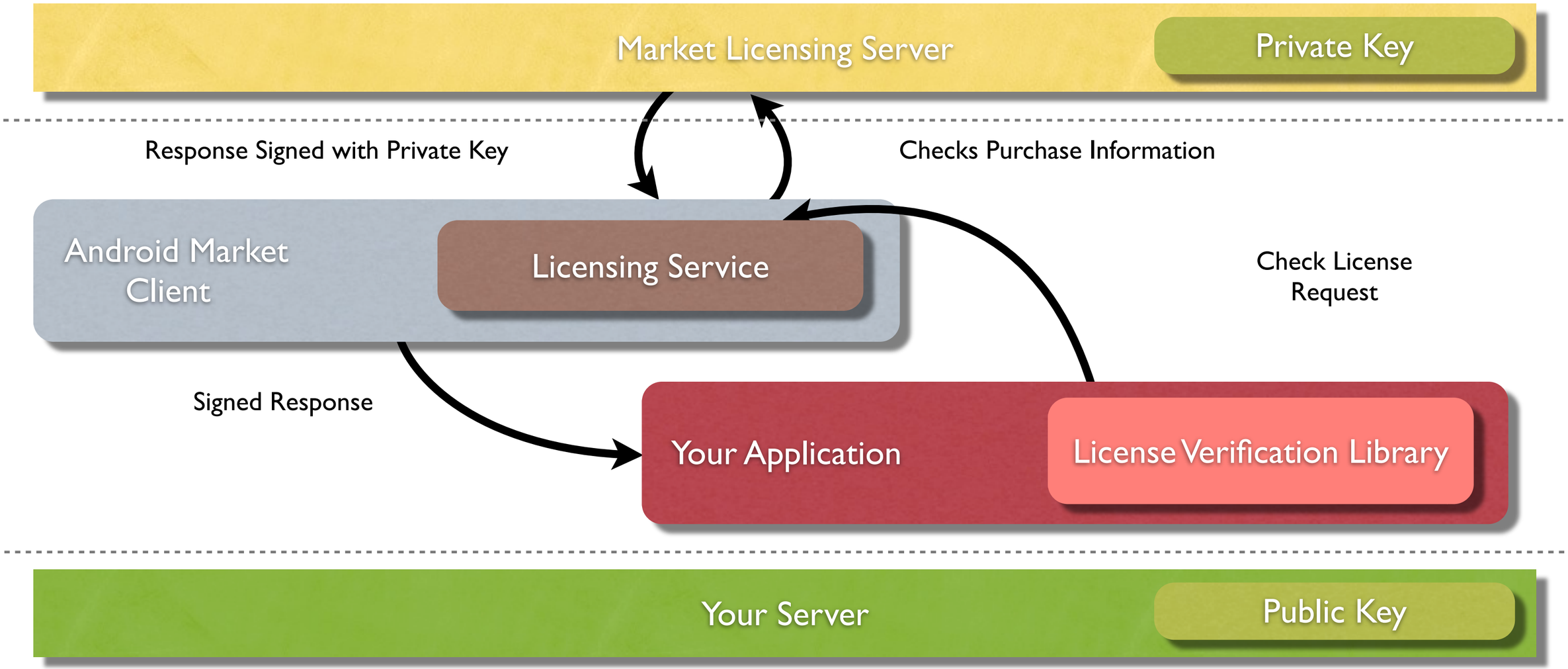
# Android Market Licensing - Server Validation



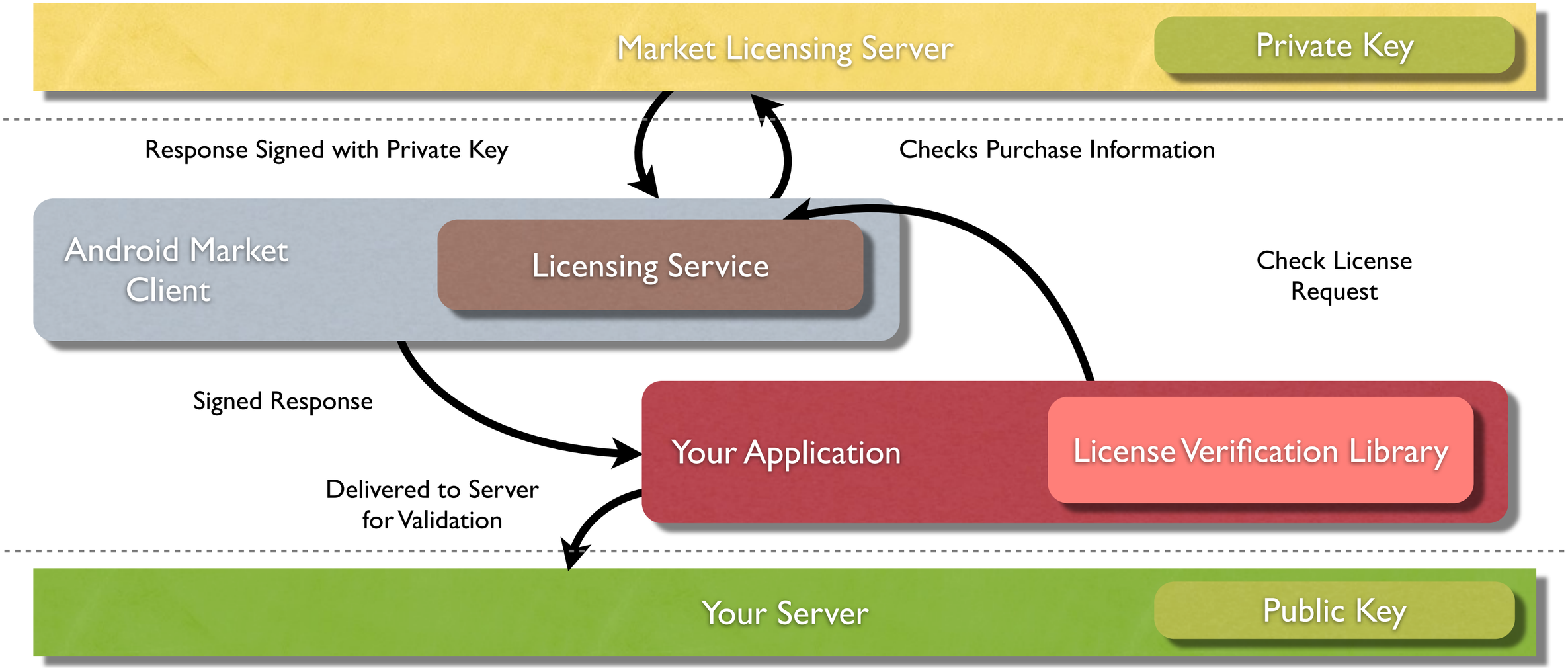
# Android Market Licensing - Server Validation



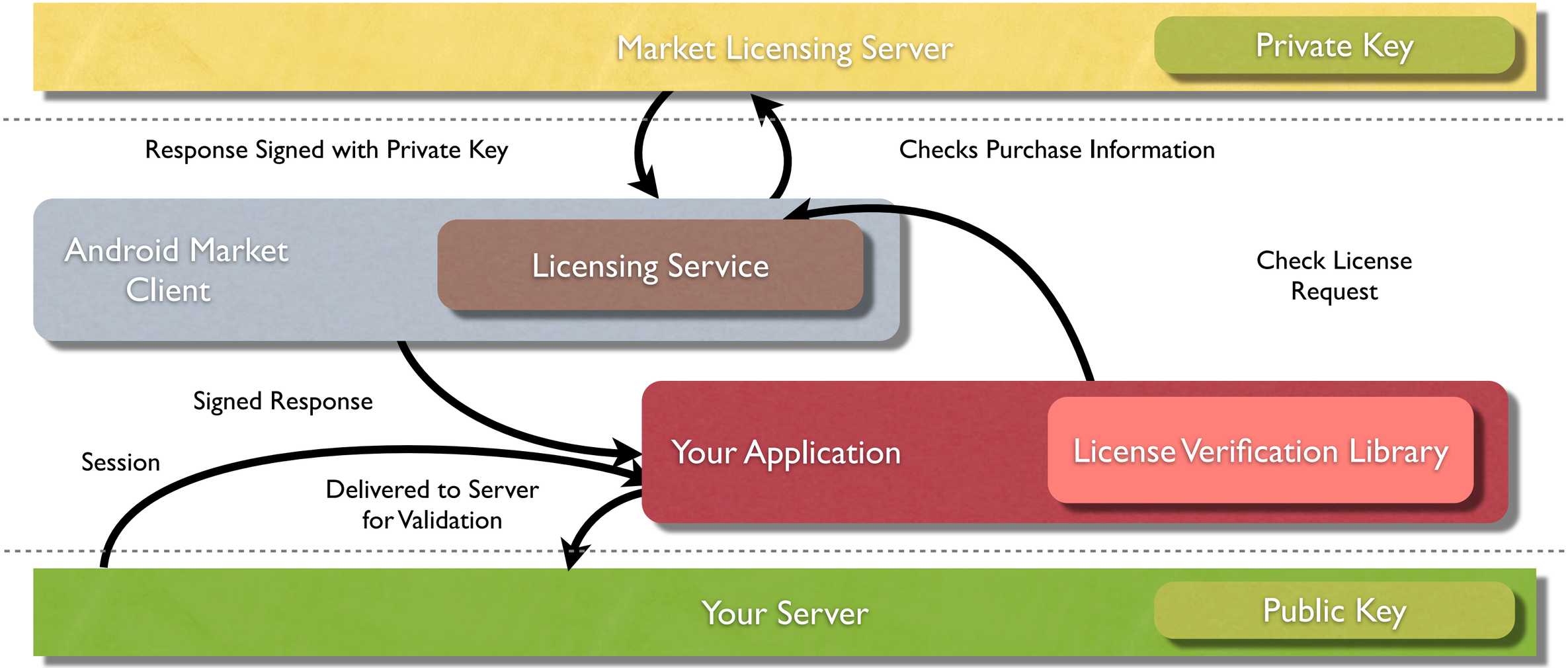
# Android Market Licensing - Server Validation



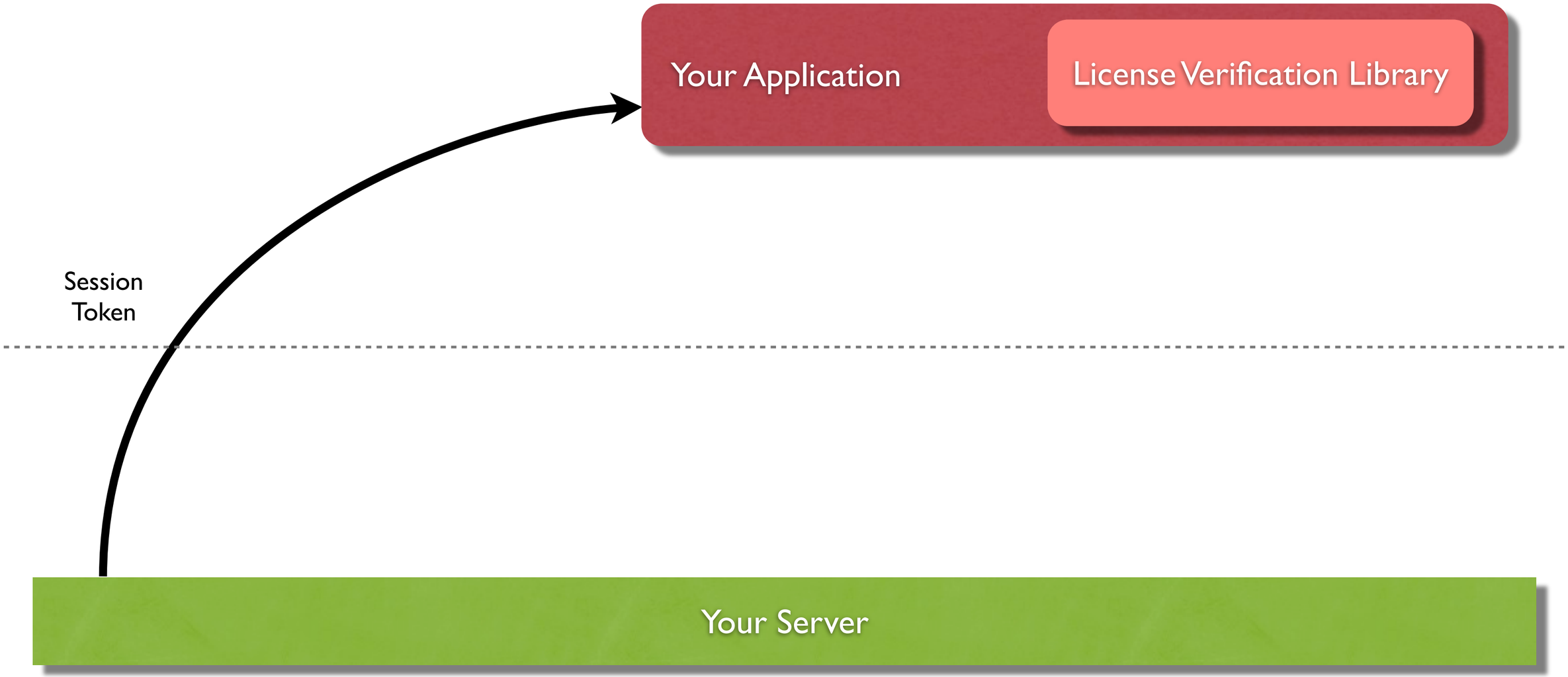
# Android Market Licensing - Server Validation



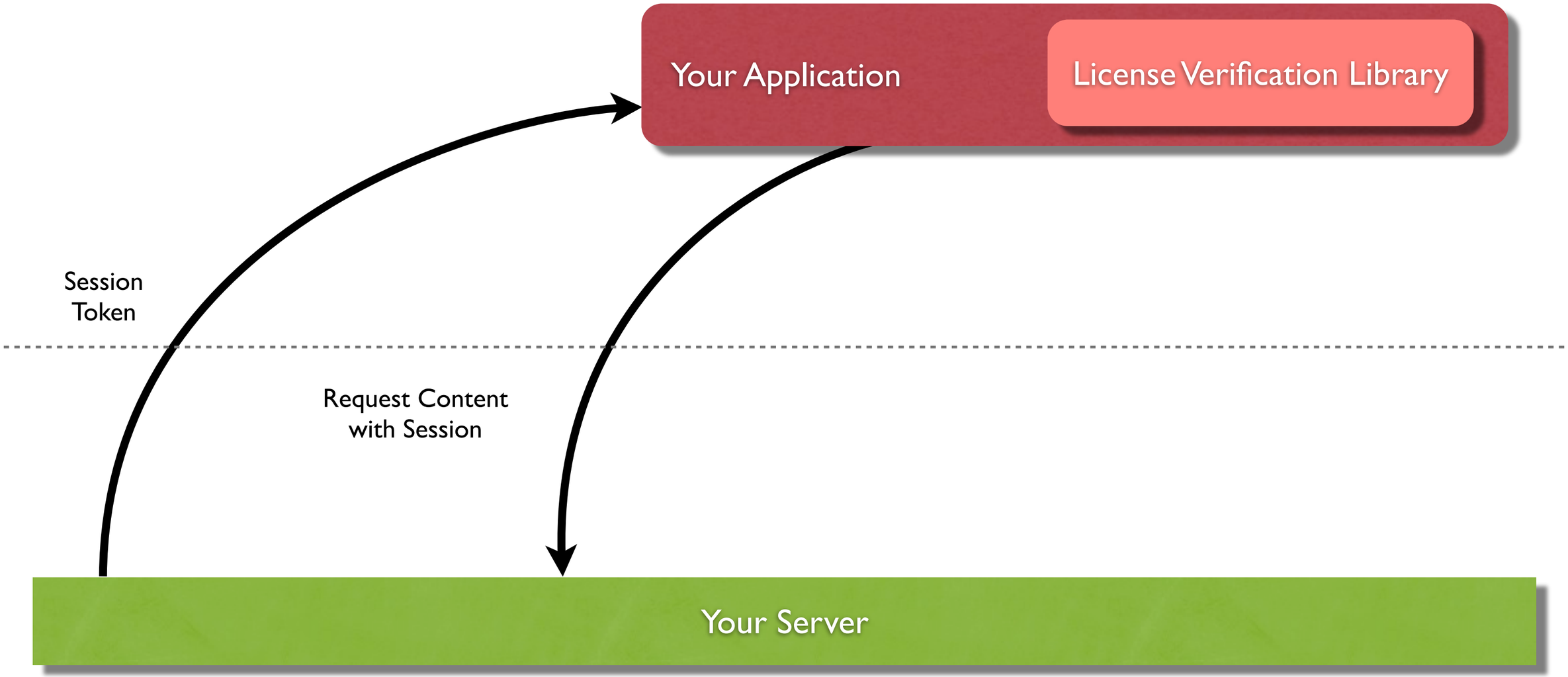
# Android Market Licensing - Server Validation



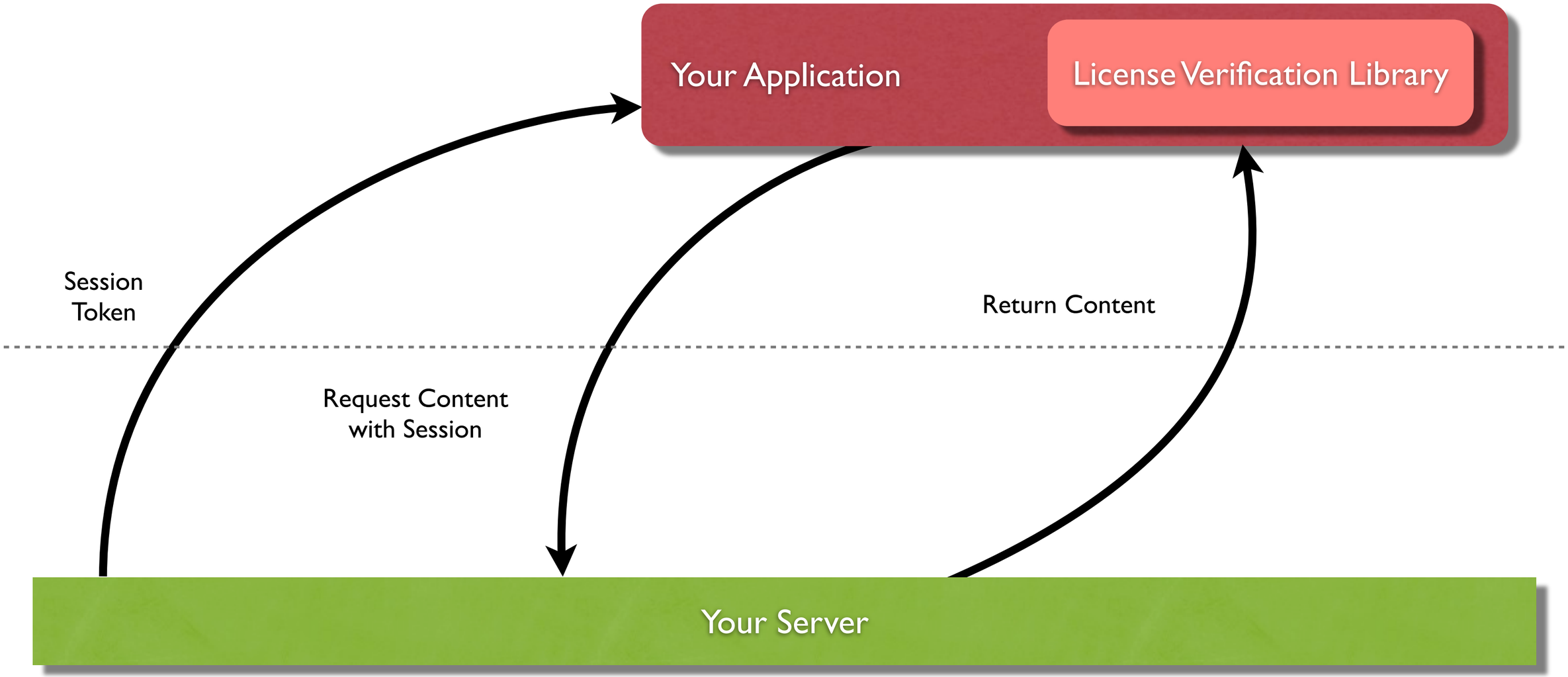
# Android Market Licensing - Server Validation



# Android Market Licensing - Server Validation



# Android Market Licensing - Server Validation



# Server-Side LVL - Replay Attacks

# Server-Side LVL - Replay Attacks

## Rely on the Nonce

- Generating the nonce on your server simplifies the server, as only nonces “in progress” must be tracked - but this adds an extra round trip to your server
- Generating the nonce on the client means the server must store all nonces

# Change the Monetization Model

- Consider a free version of the game that can be upgraded
- Use game mechanics that trade money for time
- Provide add-on content that extends play

# Android Market In-App Billing

- Purchase Virtual Goods (Managed Items)
- Purchase Consumables (Unmanaged Items)
- Client-side or Server-side Validation

# Android Market In-App Billing

- Purchase Virtual Goods (Managed Items)
- Purchase Consumables (Unmanaged Items)
- Client-side or Server-side Validation

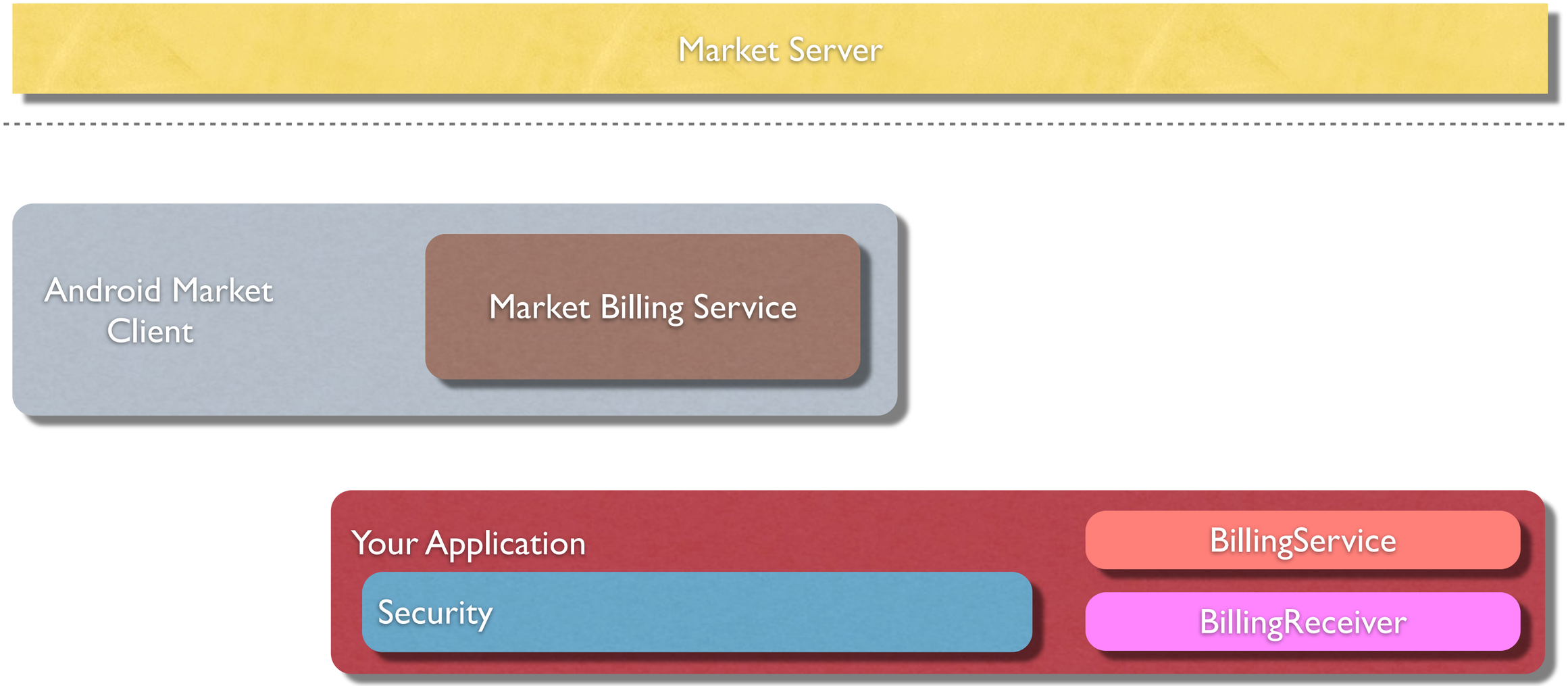
# In-App Billing - Managed Items

- SKU’s in Market - like your application
- Can only be purchased once
- Applications can ask to replay all managed item purchases

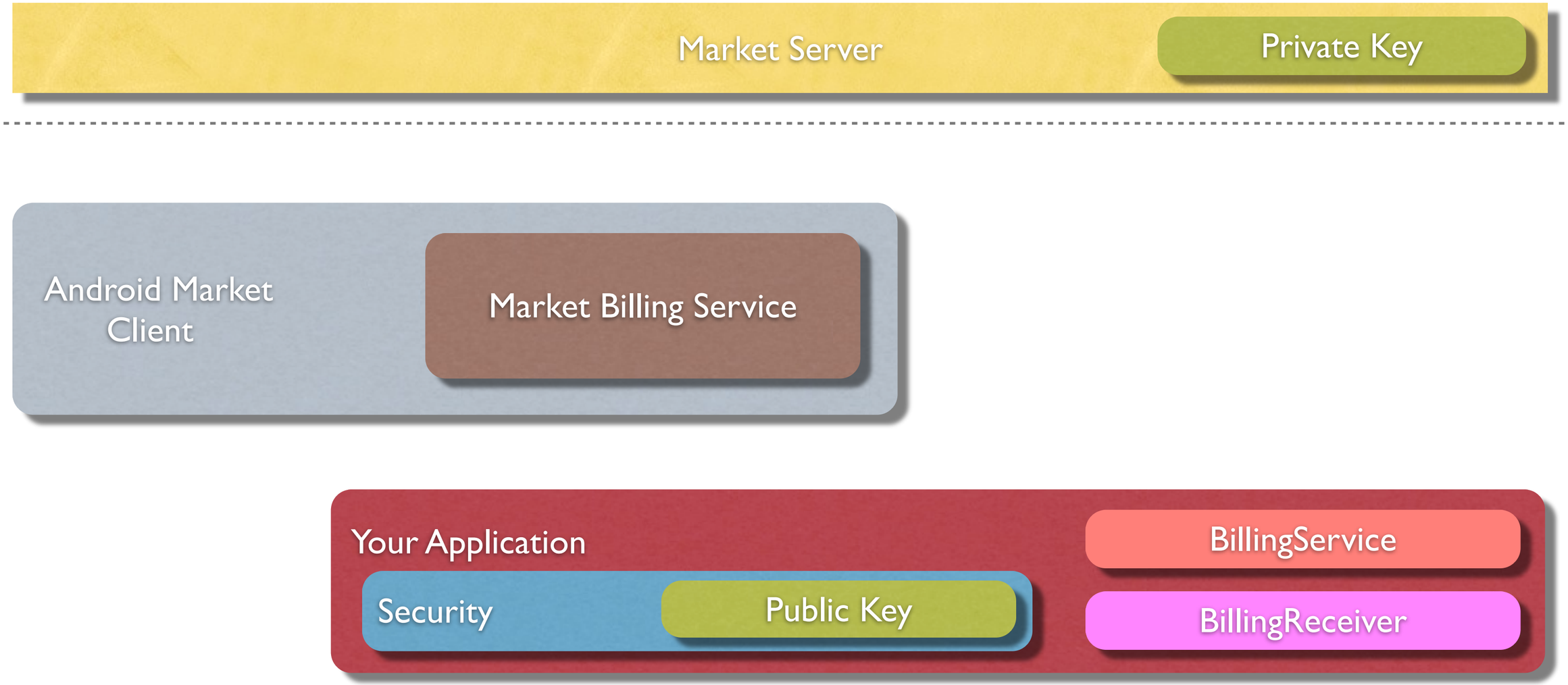
# In-App Billing - Unmanaged Items

- SKU’s in Market - like your application
- Can be purchased multiple times
- Applications cannot ask for a replay

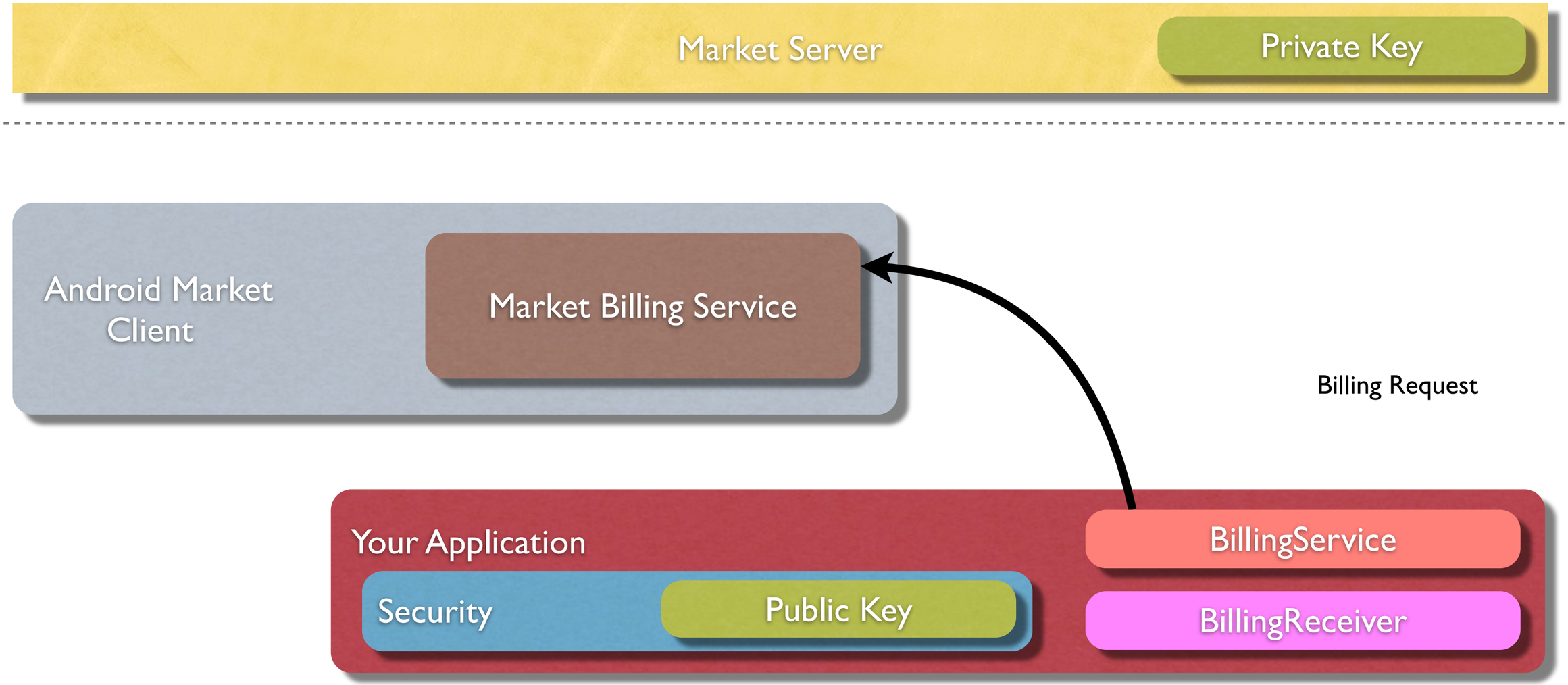
# In-App Billing - Client Version



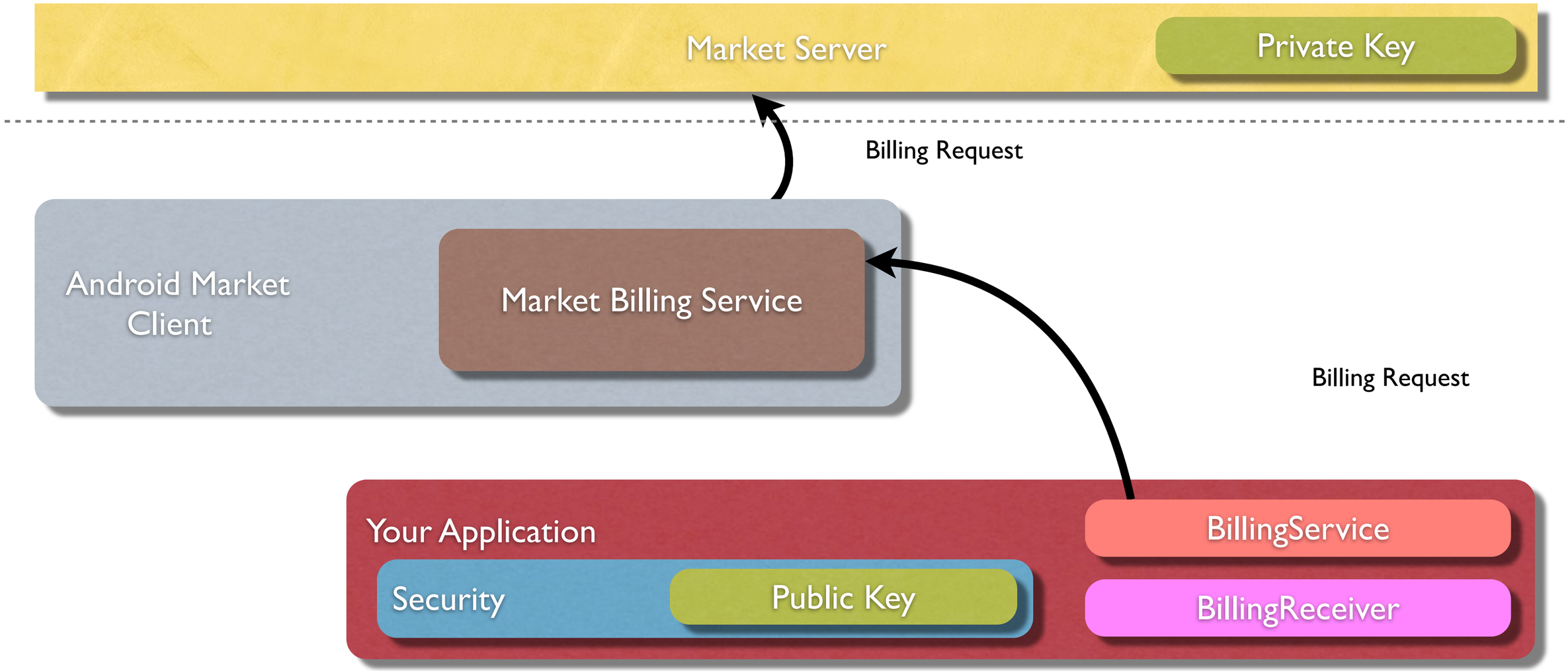
# In-App Billing - Client Version



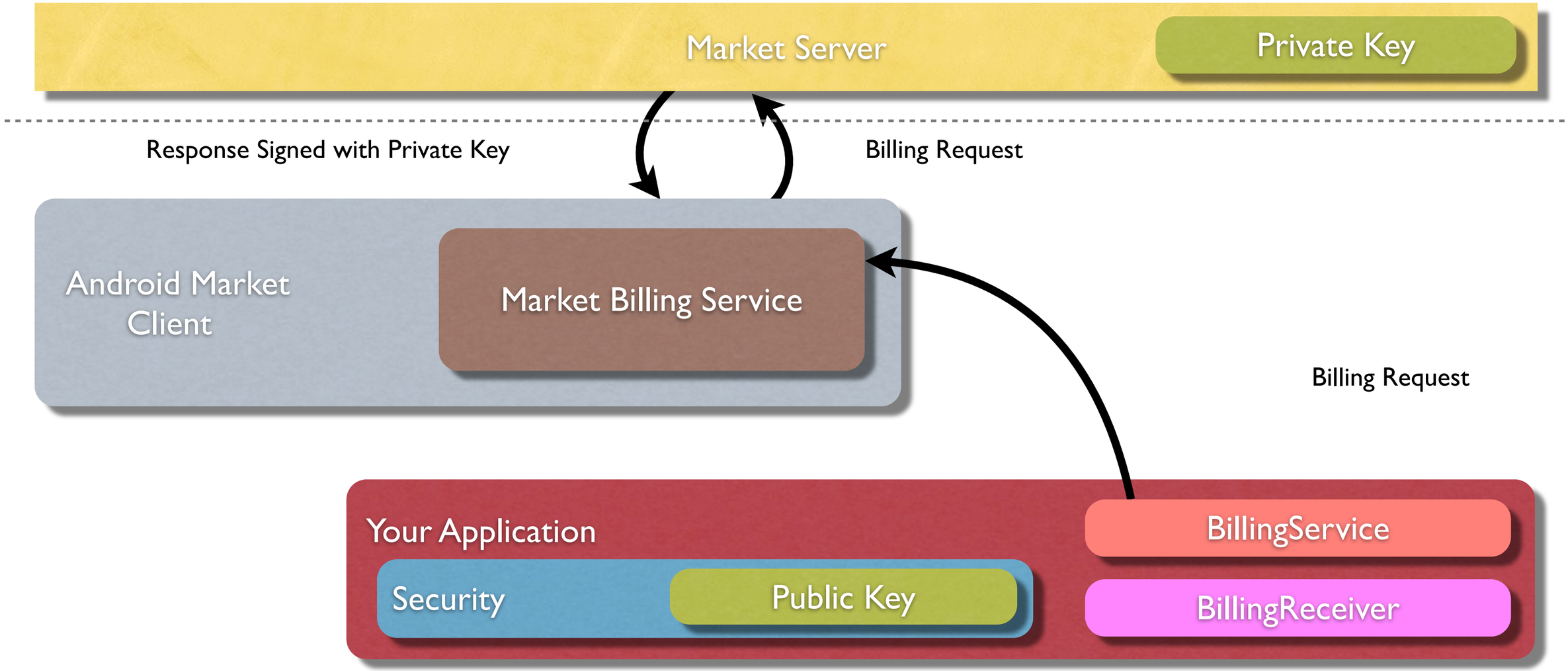
# In-App Billing - Client Version



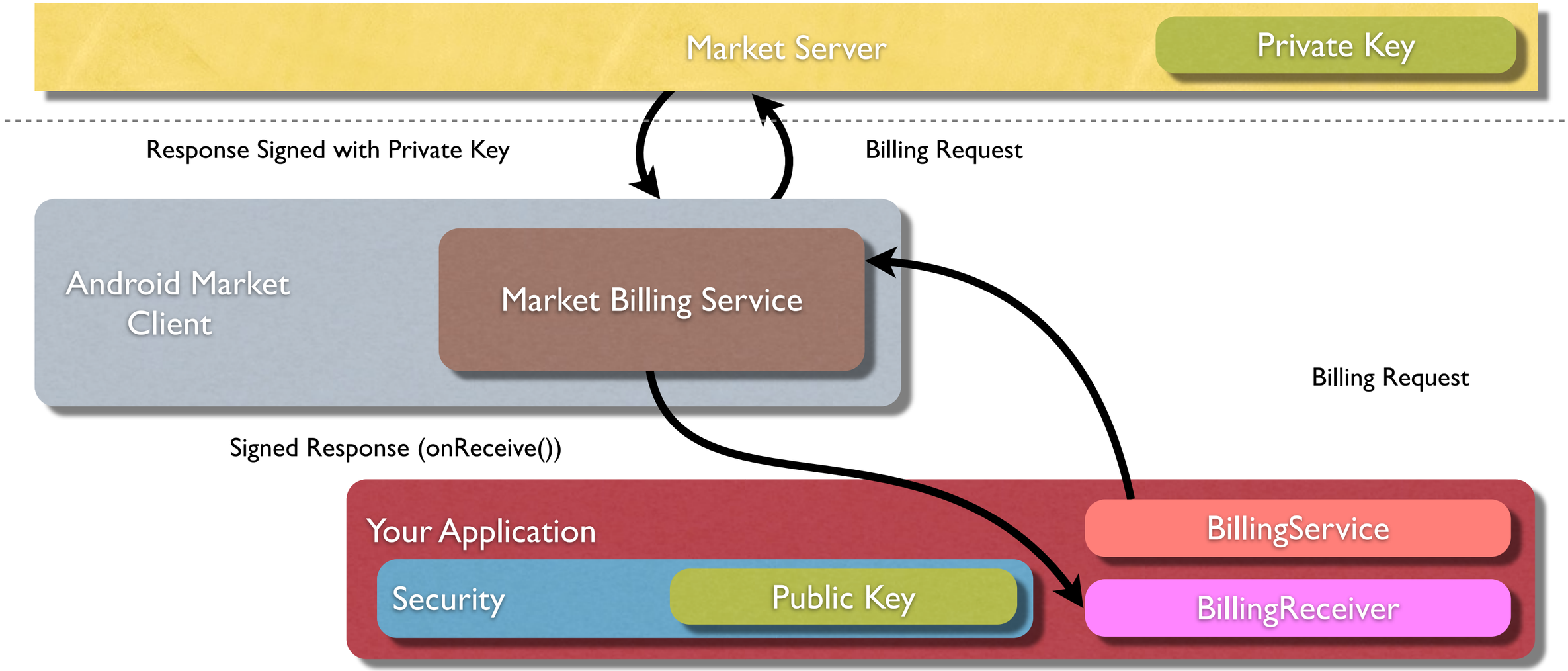
# In-App Billing - Client Version



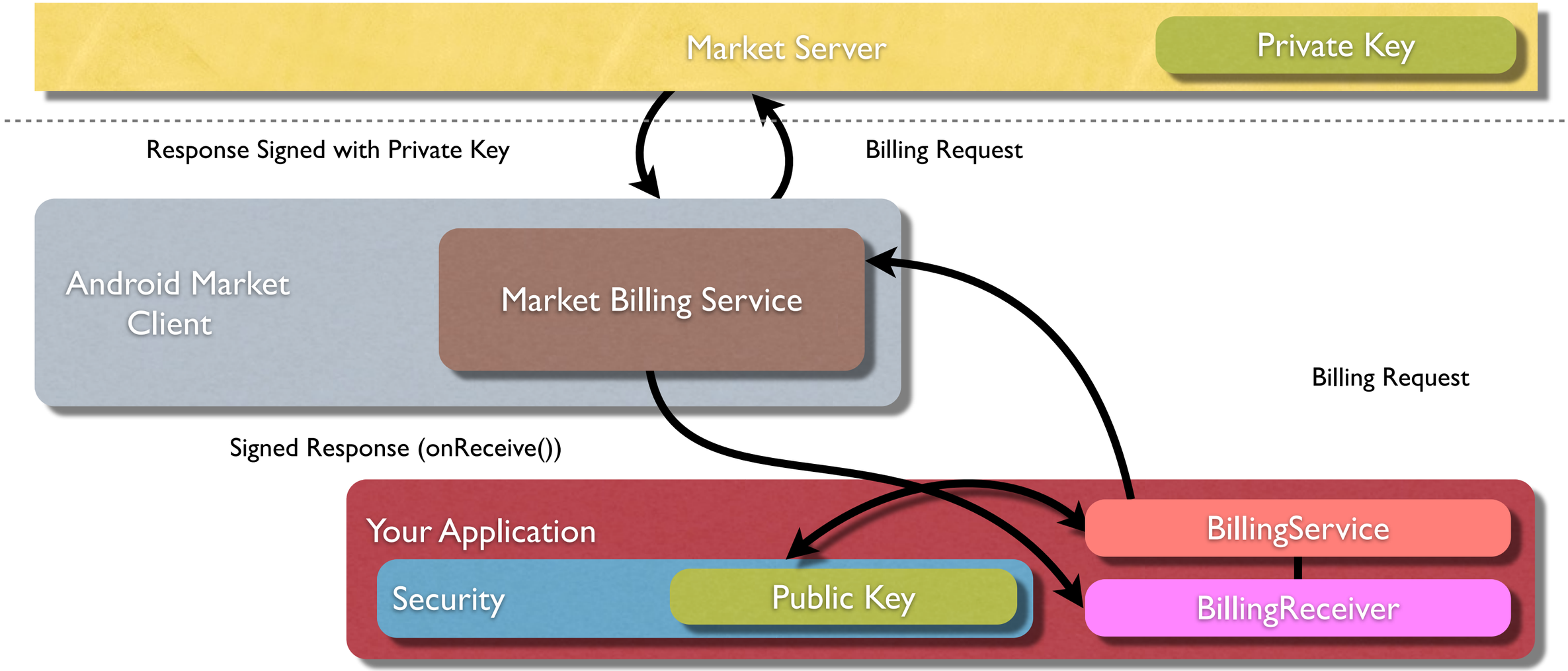
# In-App Billing - Client Version



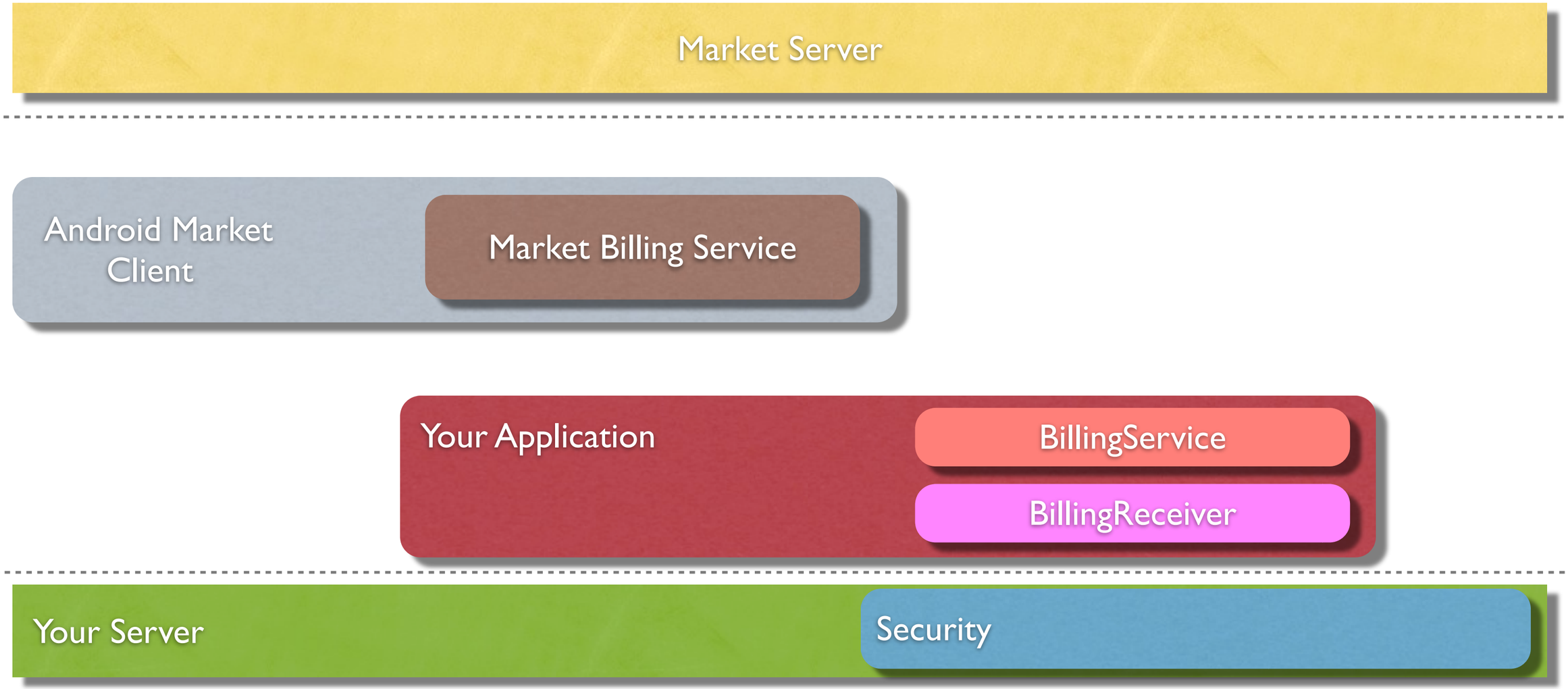
# In-App Billing - Client Version



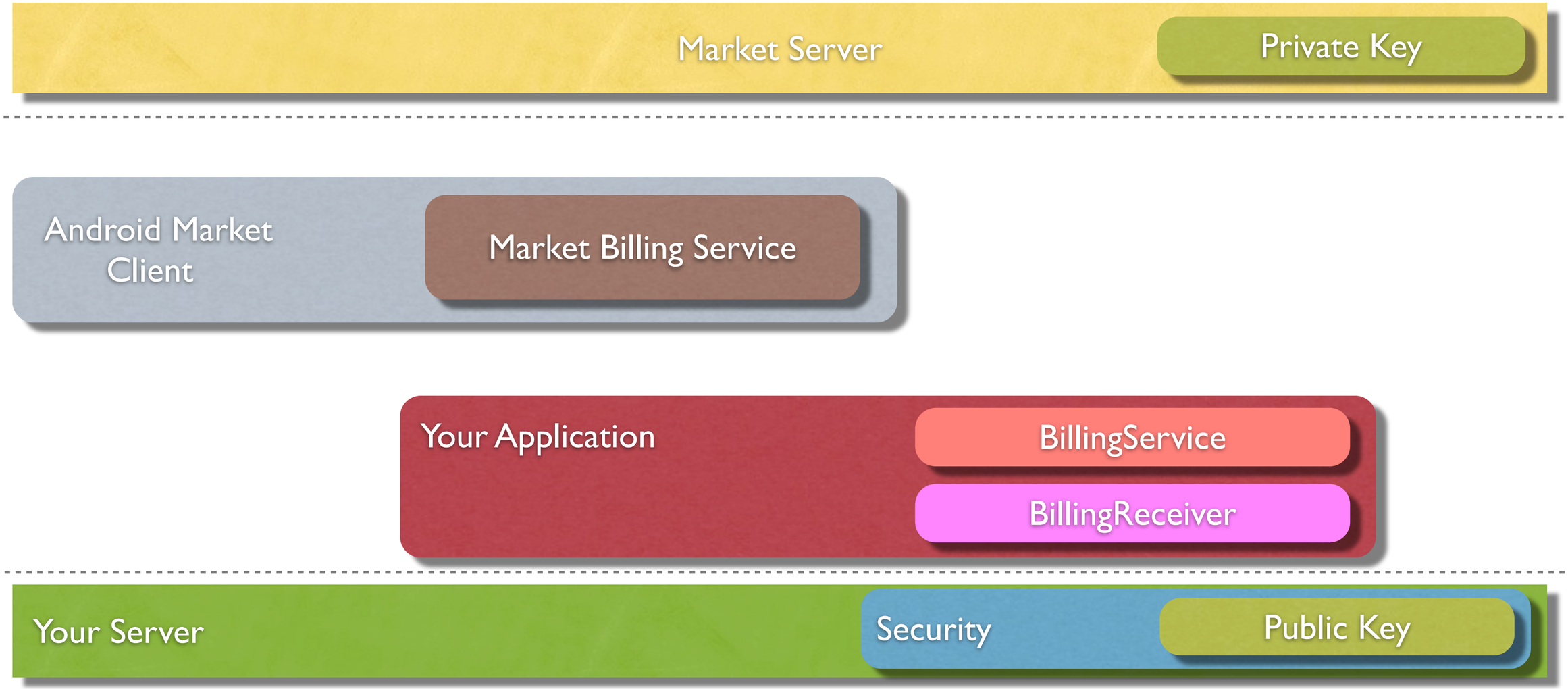
# In-App Billing - Client Version



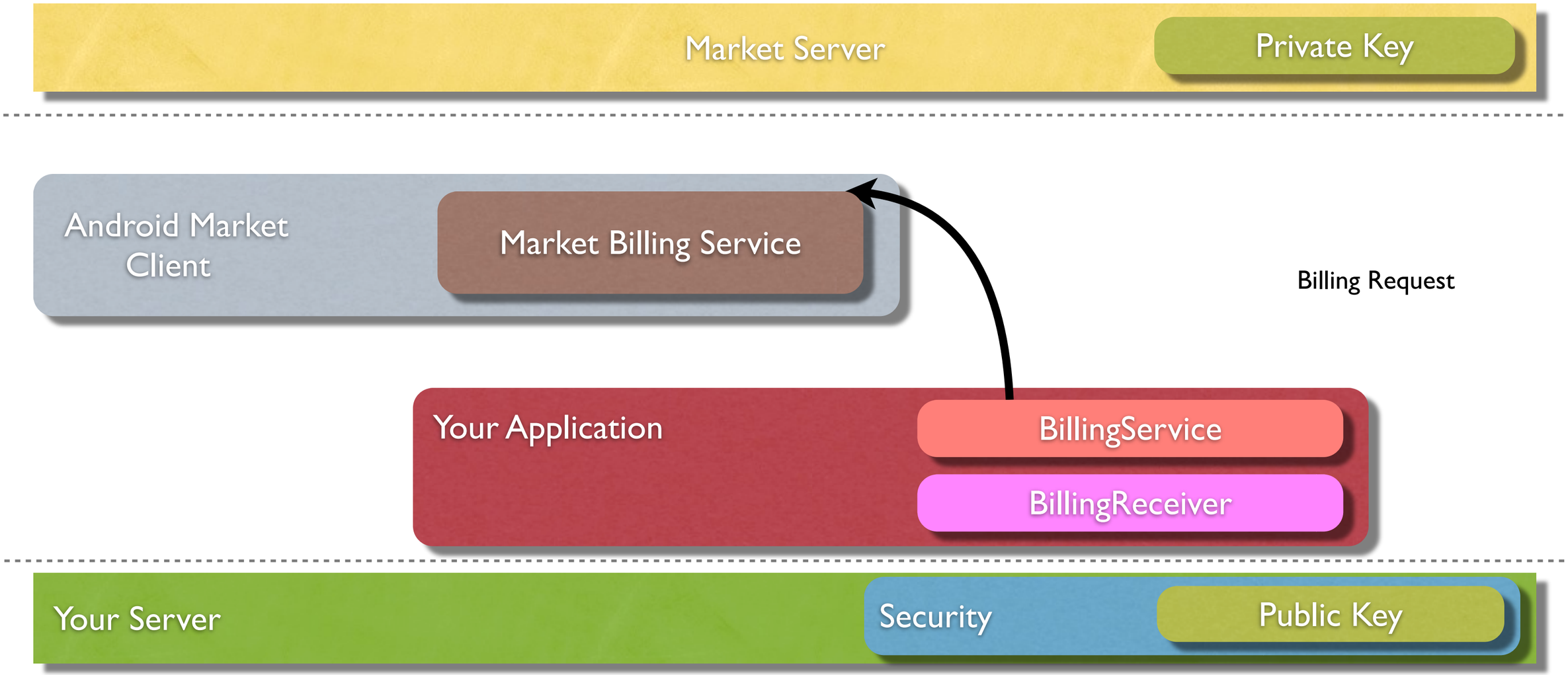
# In-App Billing - Server Version



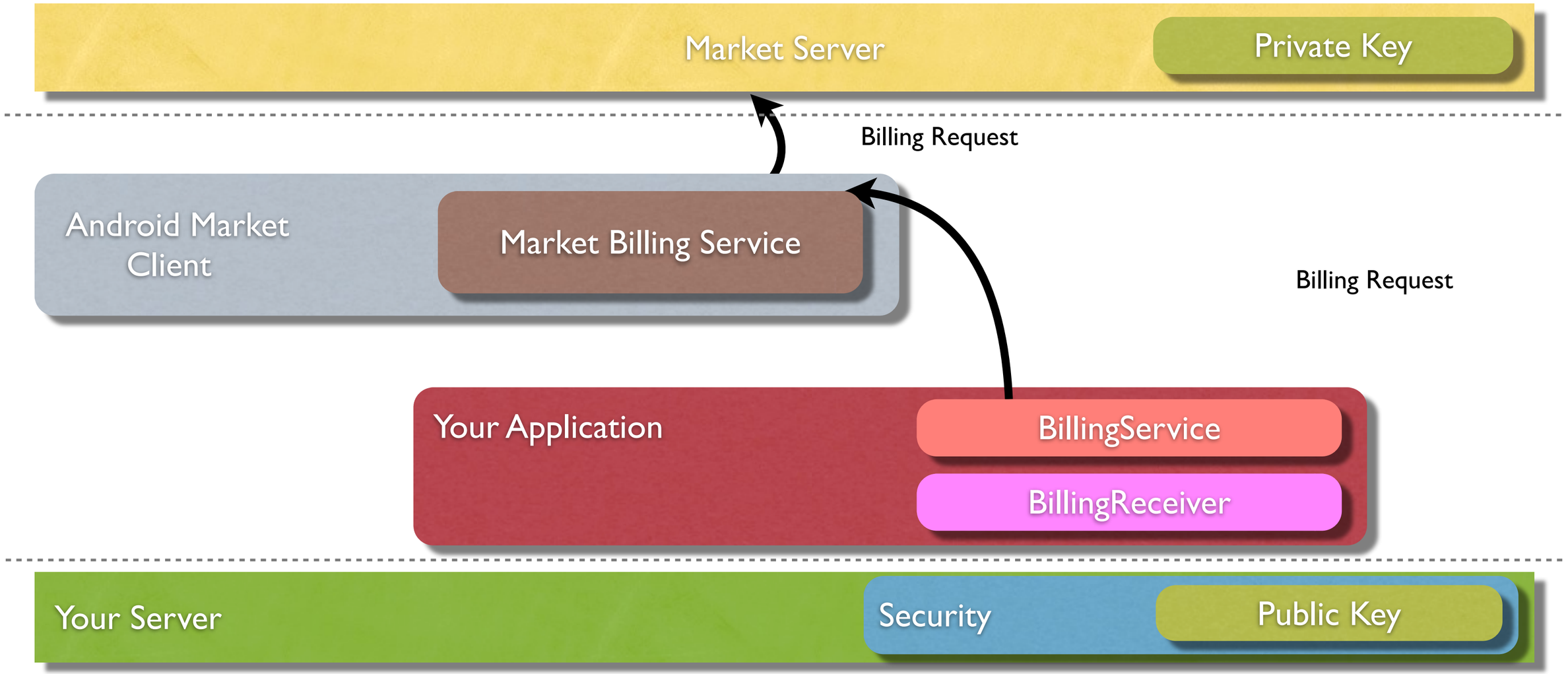
# In-App Billing - Server Version



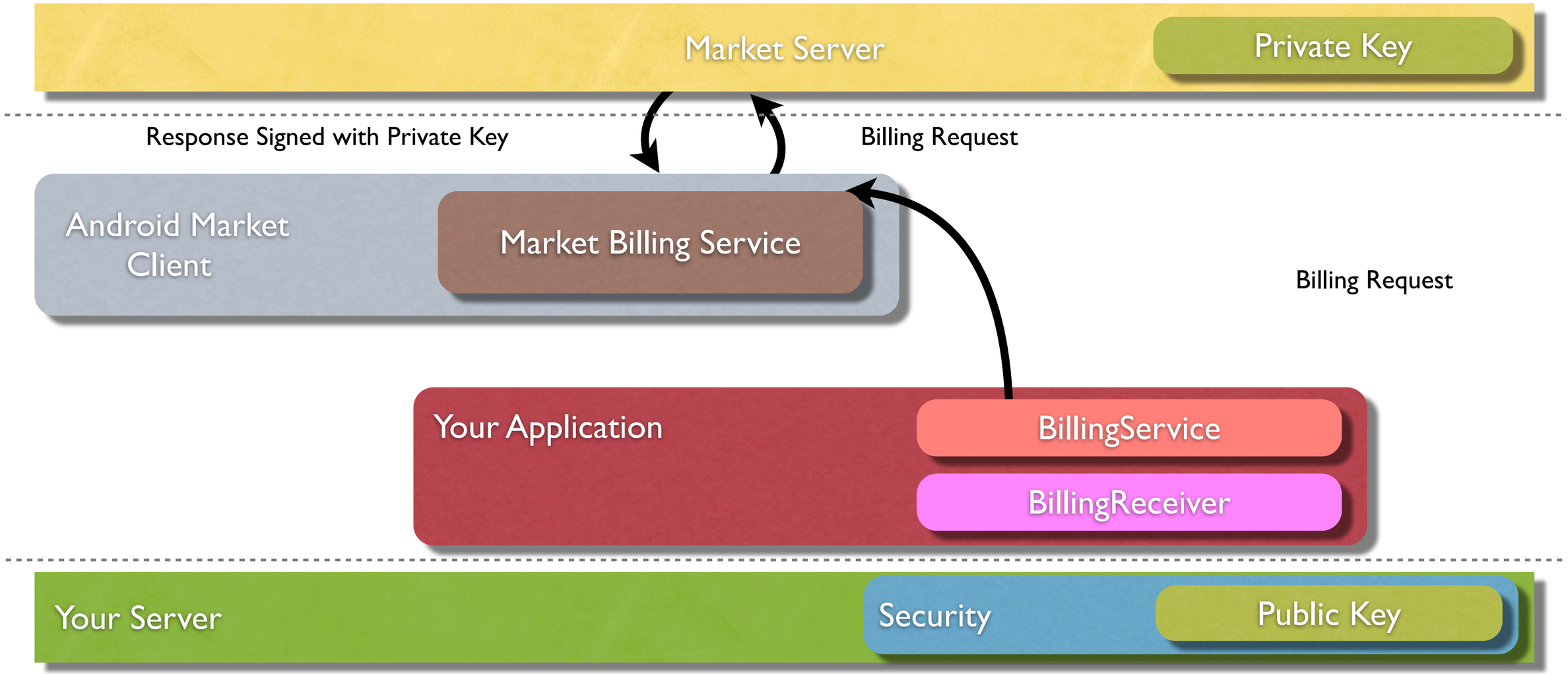
# In-App Billing - Server Version



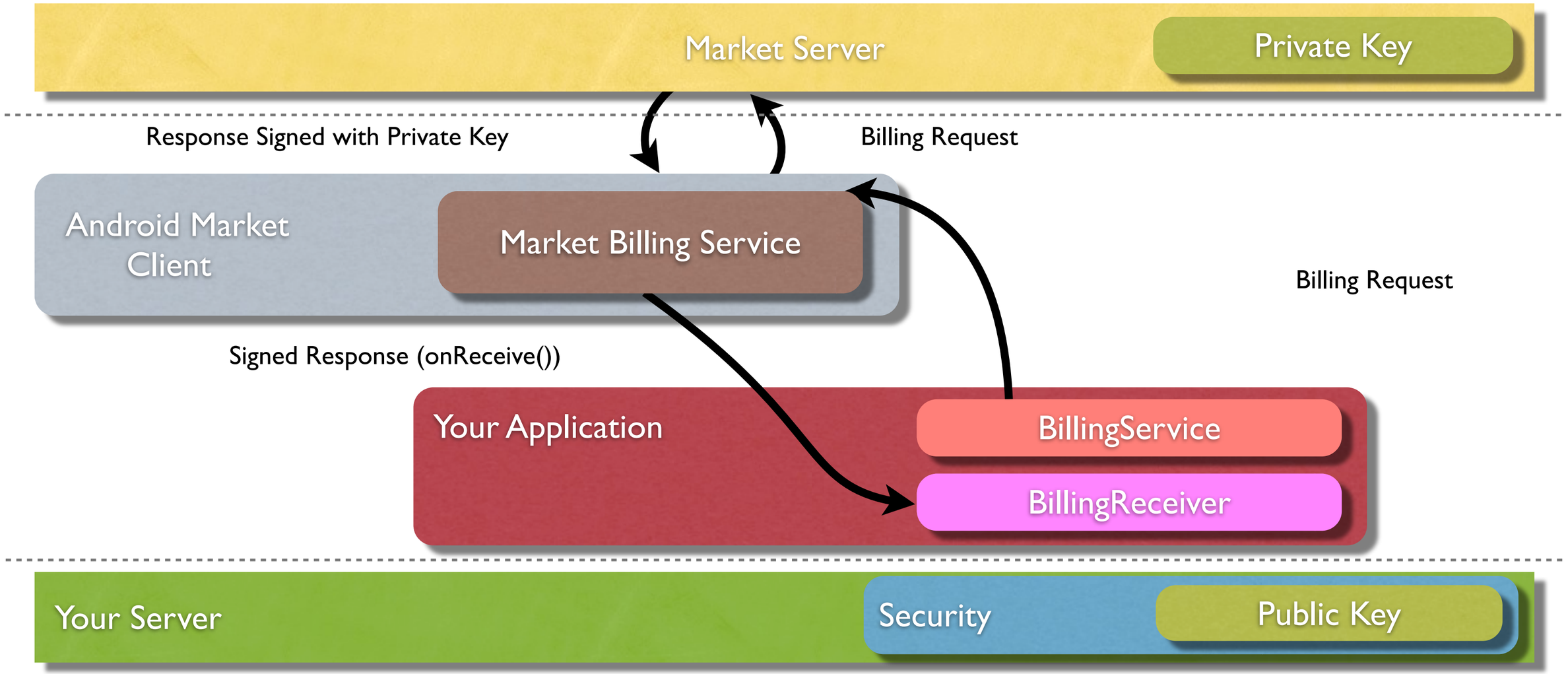
# In-App Billing - Server Version



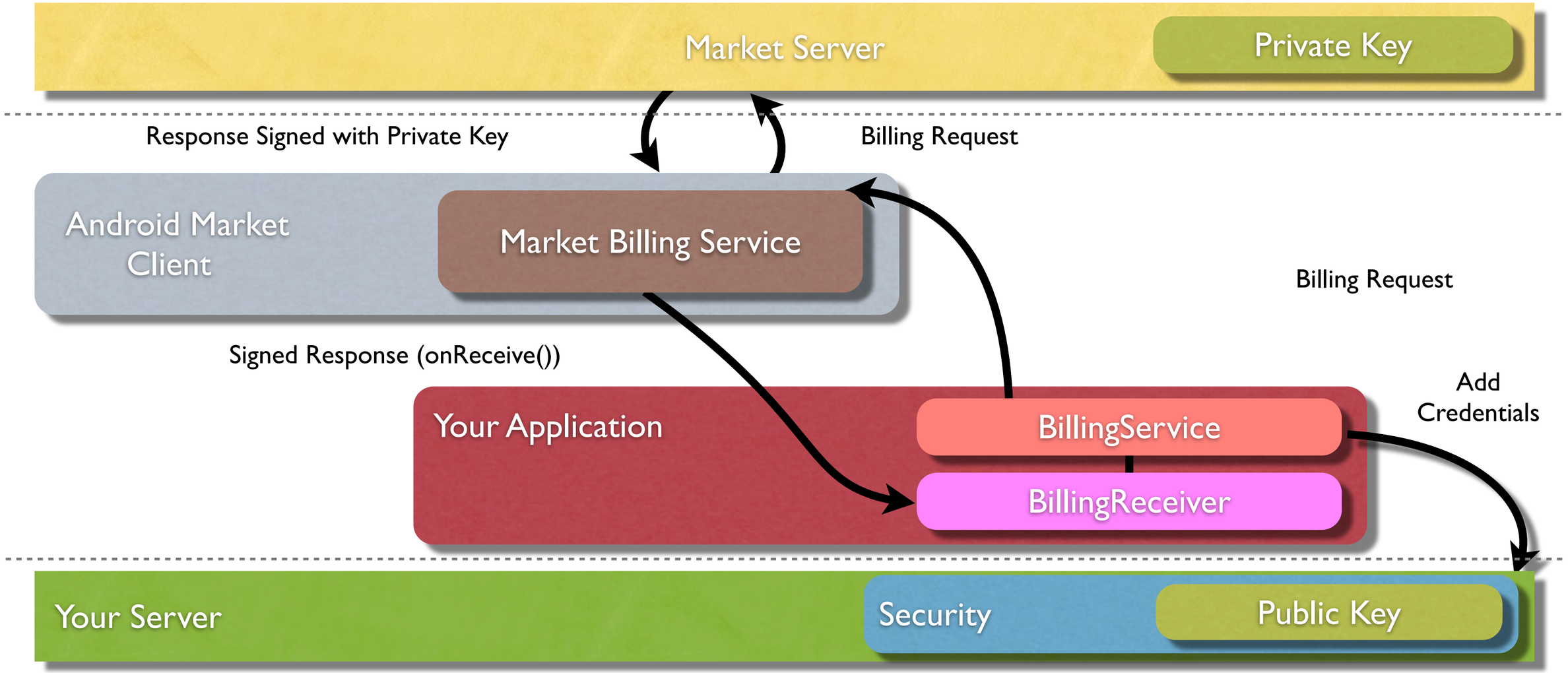
# In-App Billing - Server Version



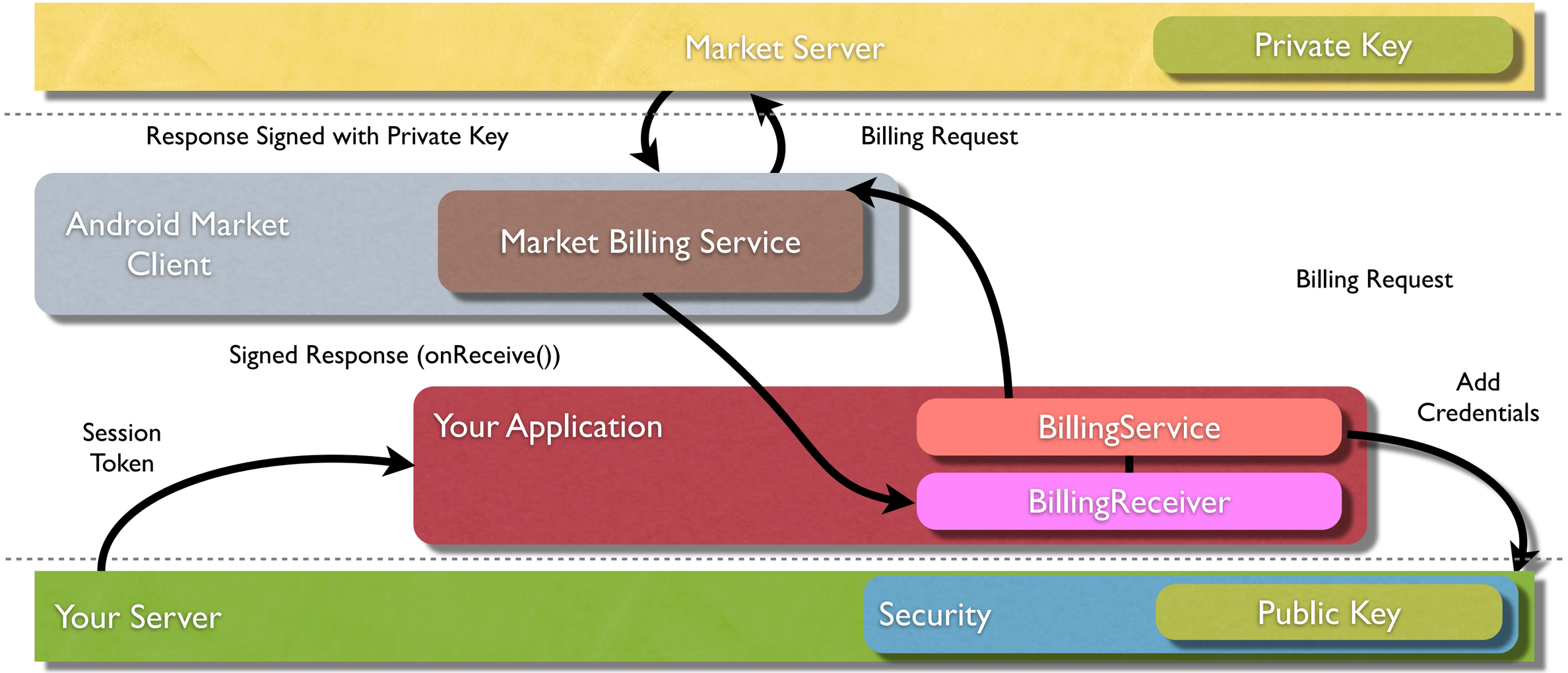
# In-App Billing - Server Version



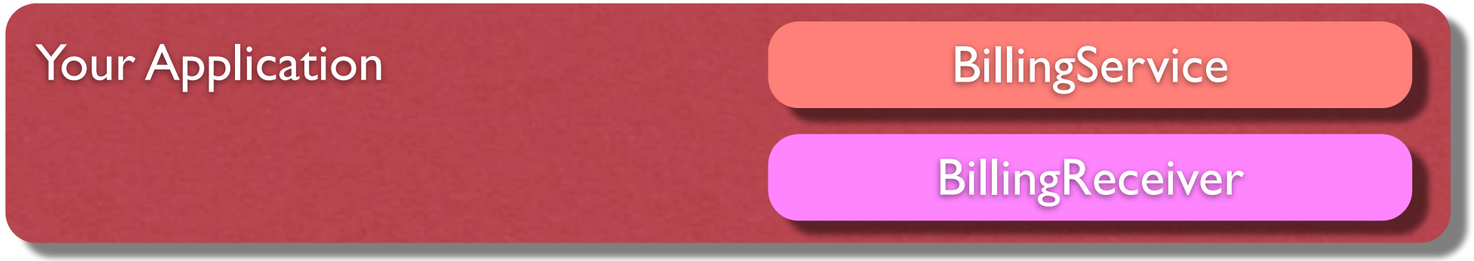
# In-App Billing - Server Version



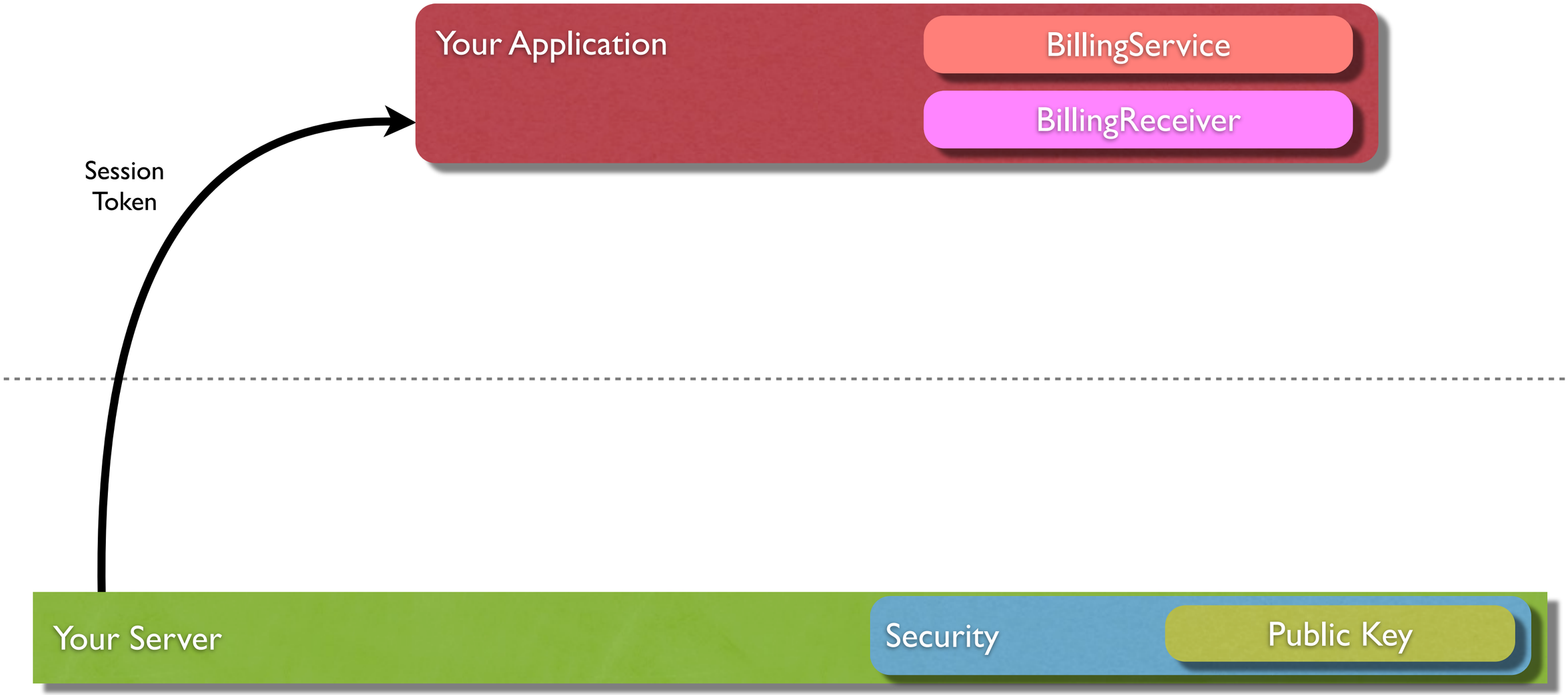
# In-App Billing - Server Version



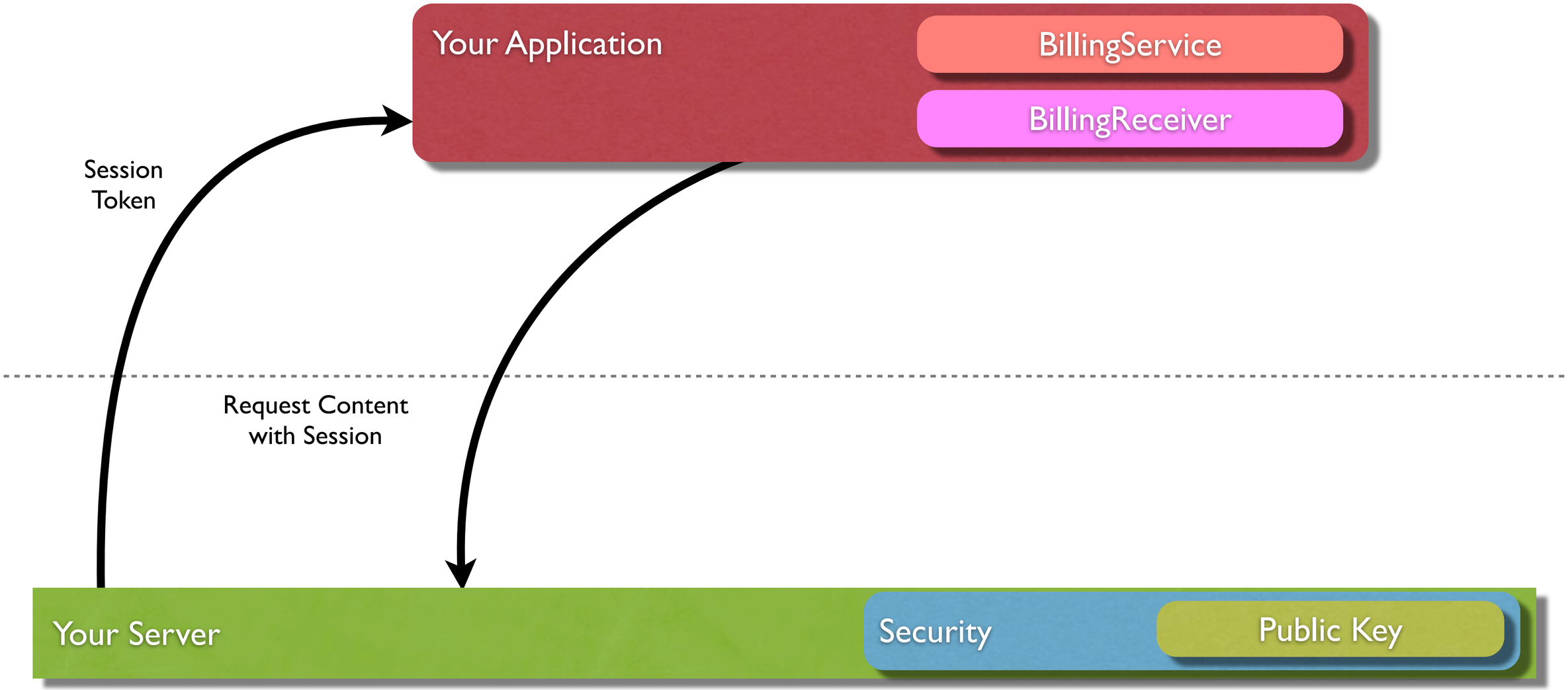
# In-App Billing - Server Version



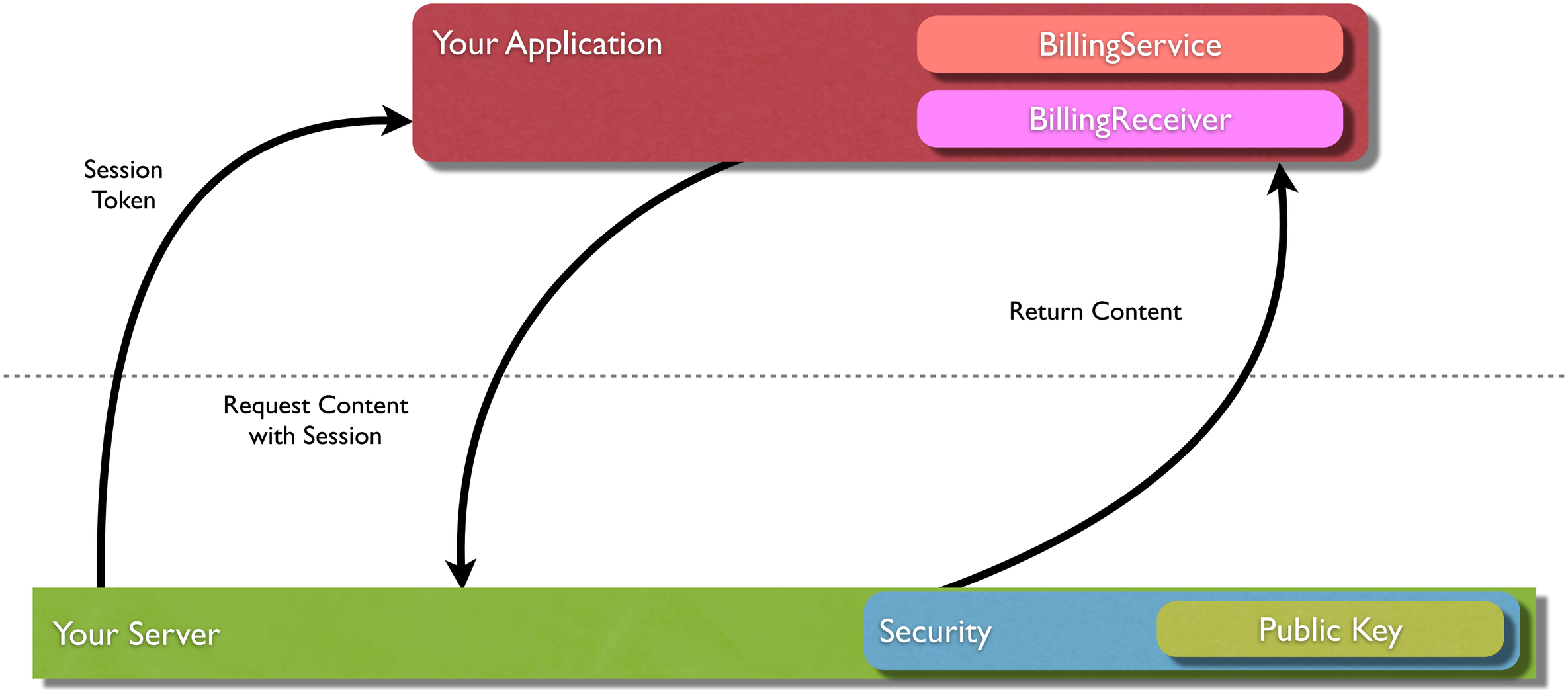

# In-App Billing - Server Version



# In-App Billing - Server Version



# In-App Billing - Server Version

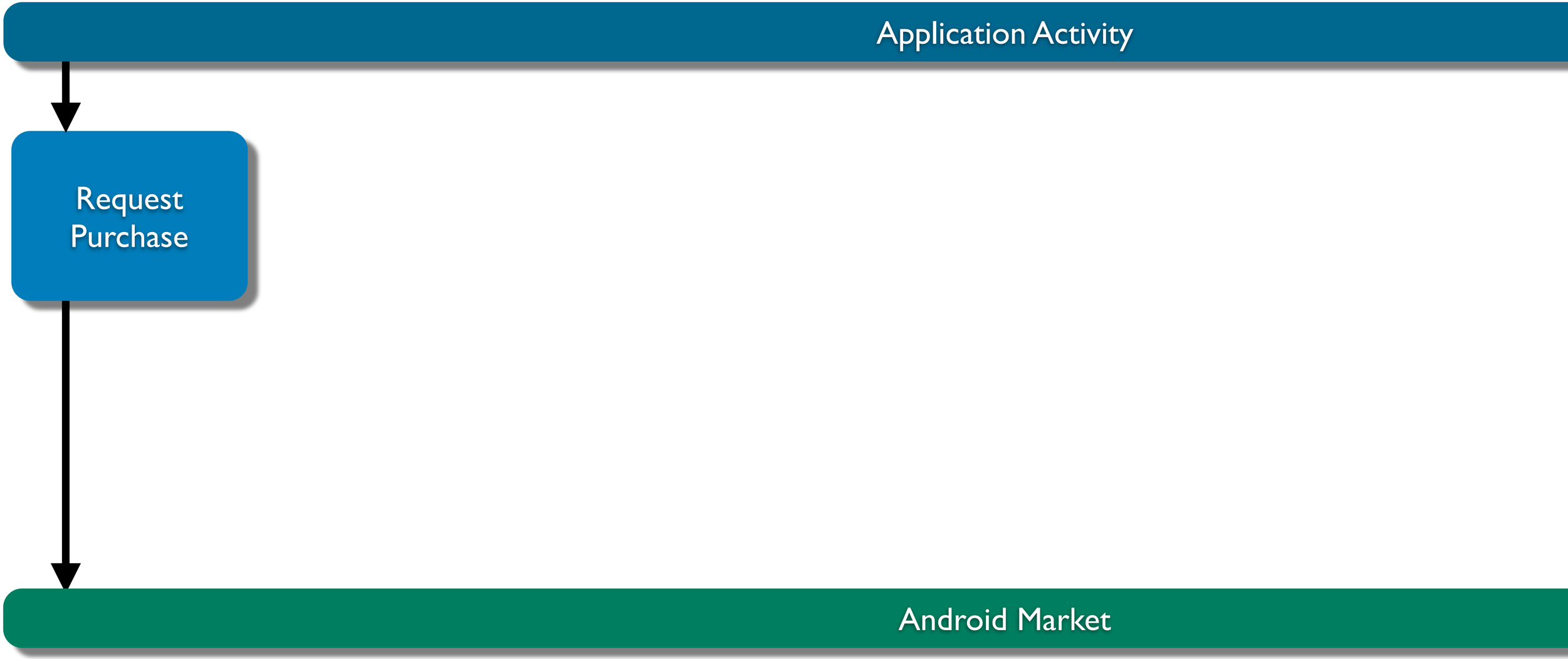


# In-App Billing - Request Purchase

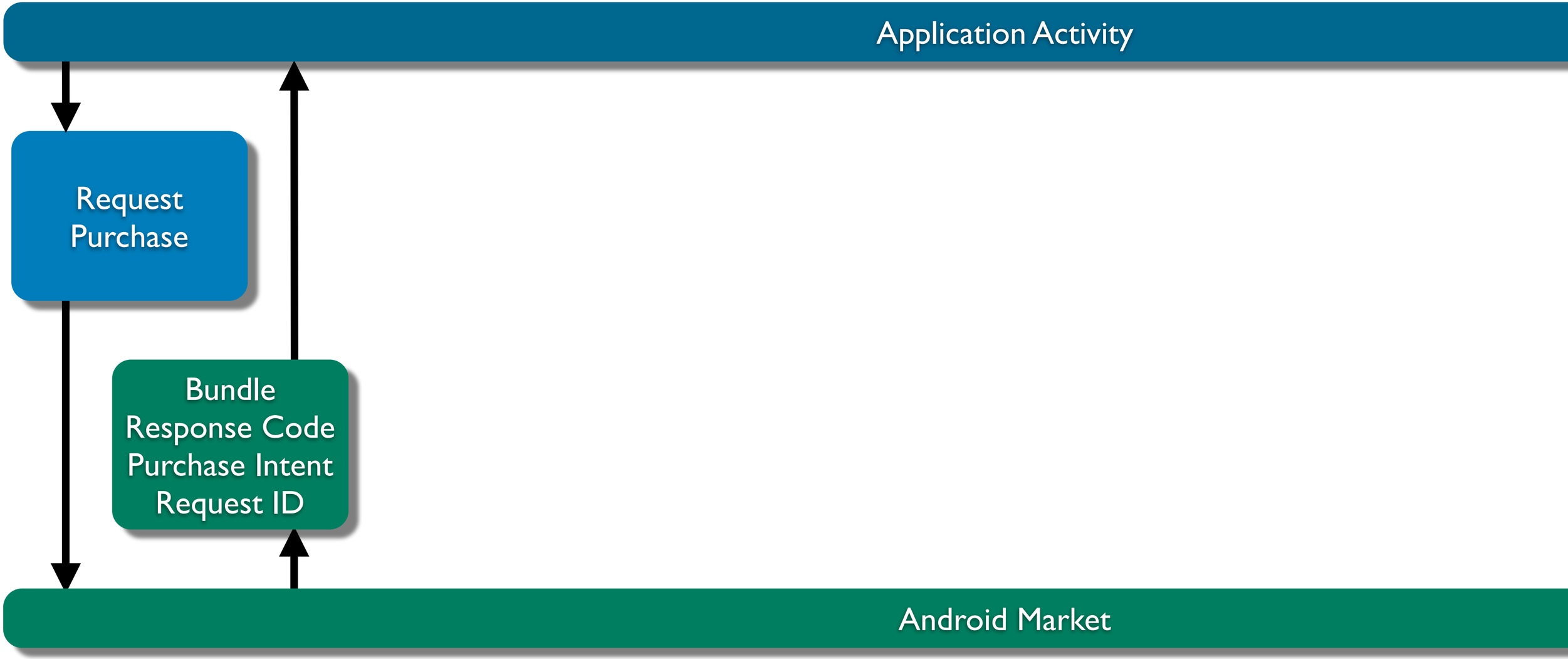
Application Activity

Android Market

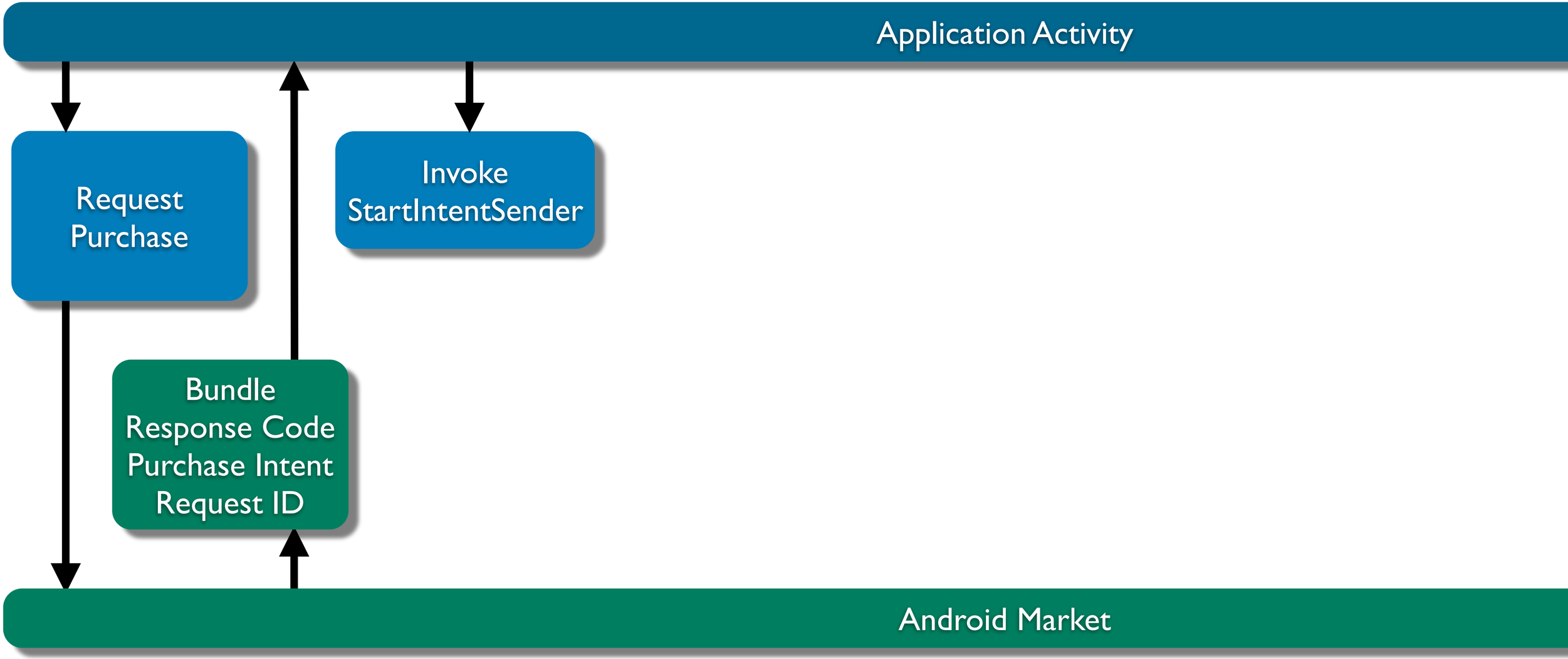
# In-App Billing - Request Purchase



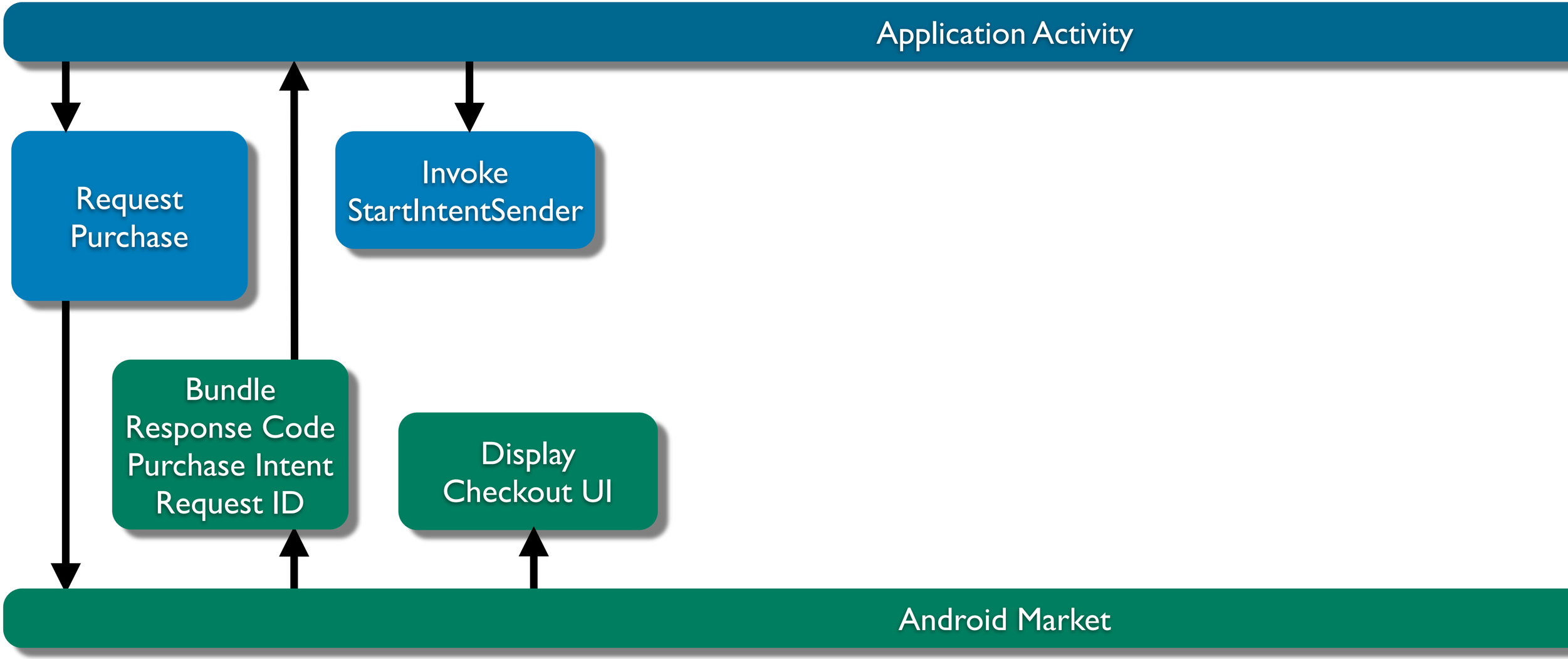
# In-App Billing - Request Purchase



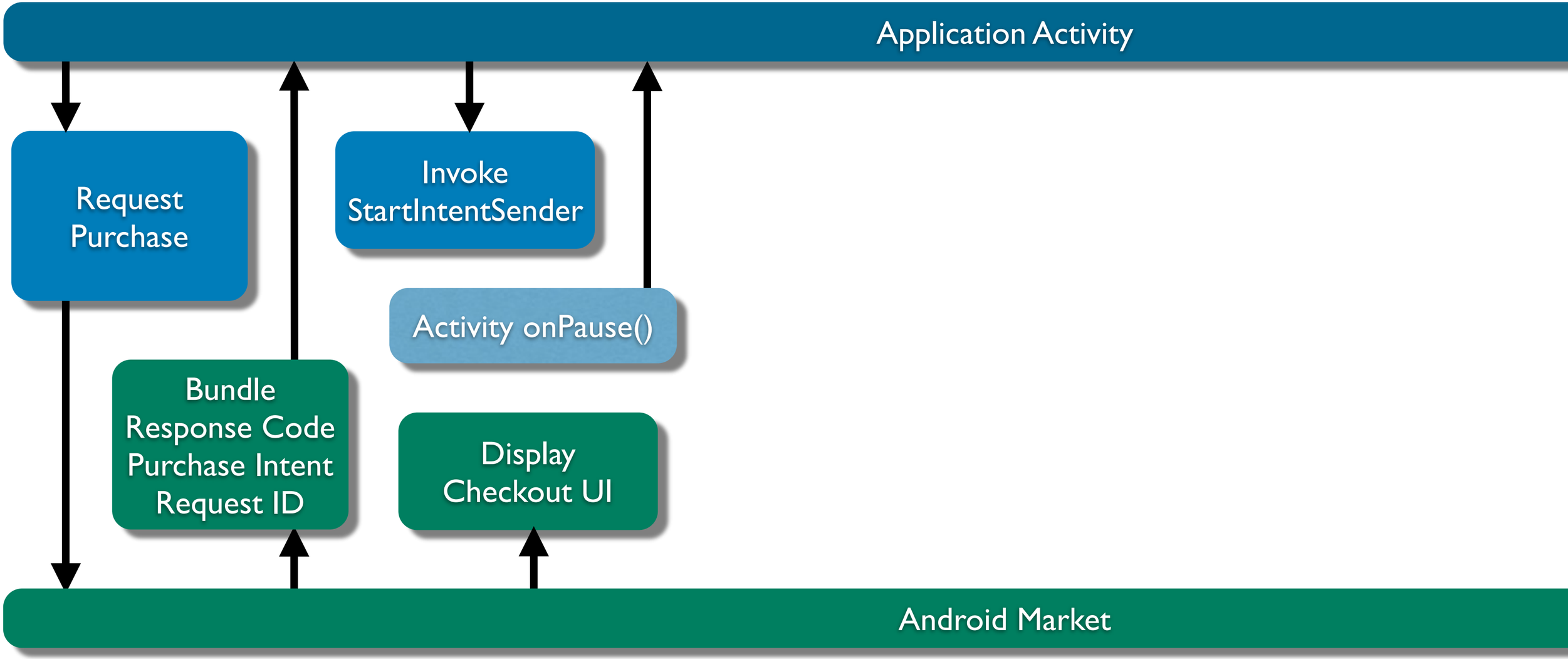
# In-App Billing - Request Purchase



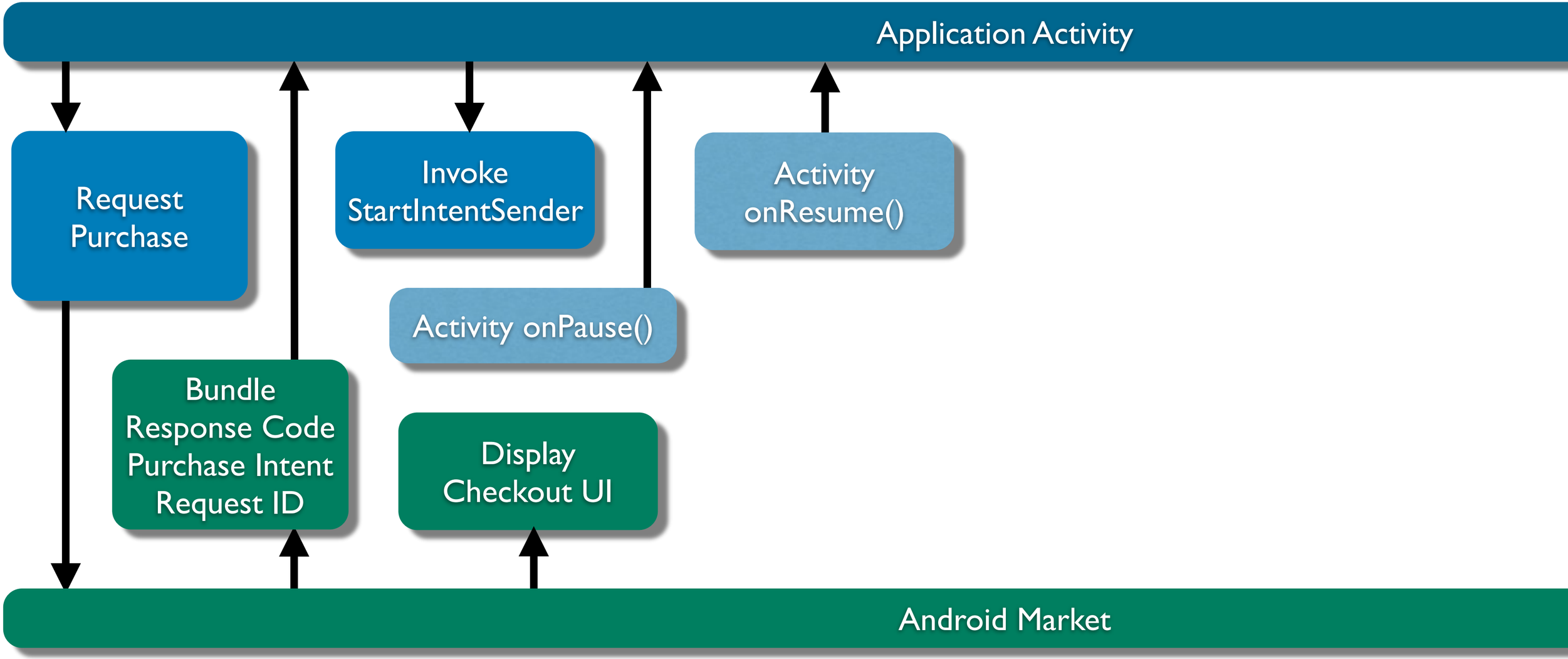
# In-App Billing - Request Purchase



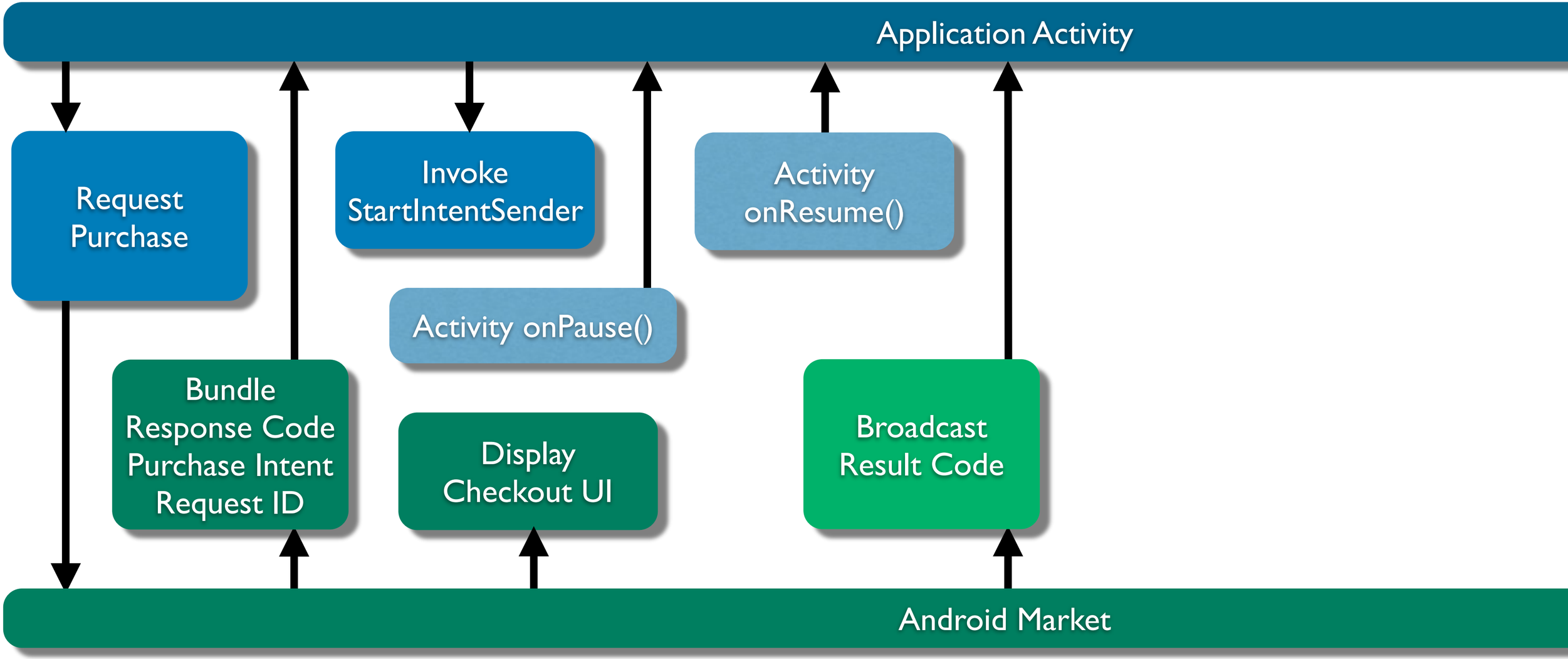
# In-App Billing - Request Purchase



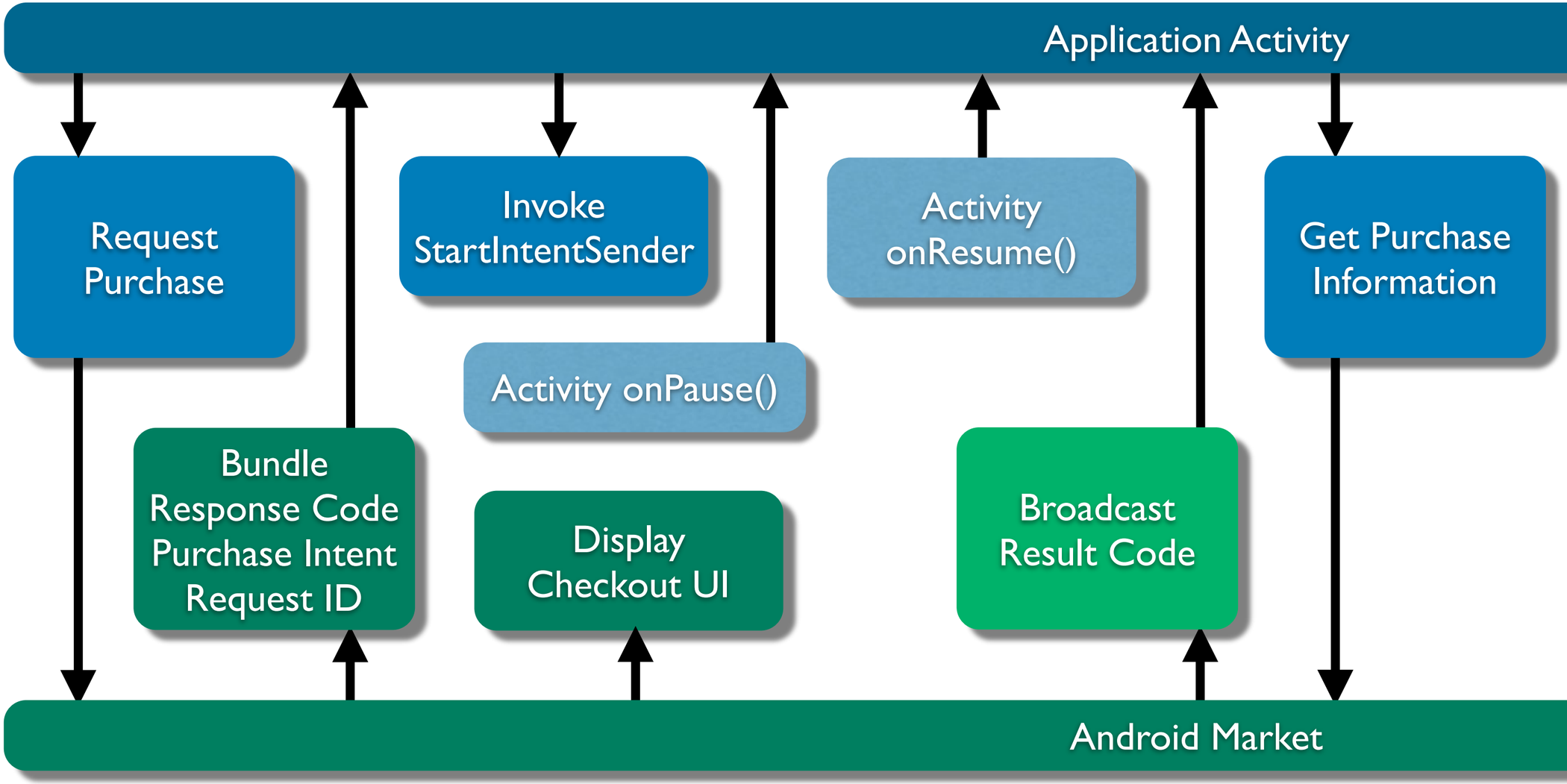
# In-App Billing - Request Purchase



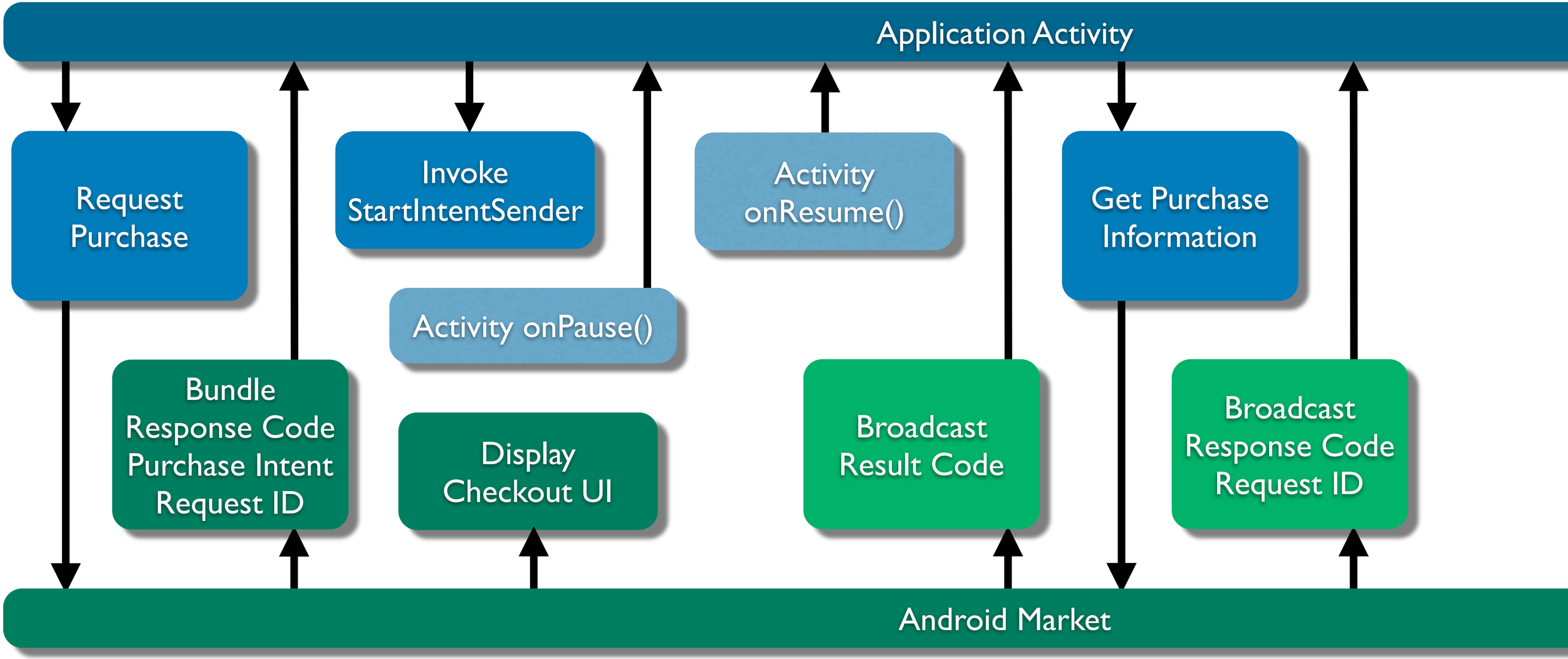
# In-App Billing - Request Purchase



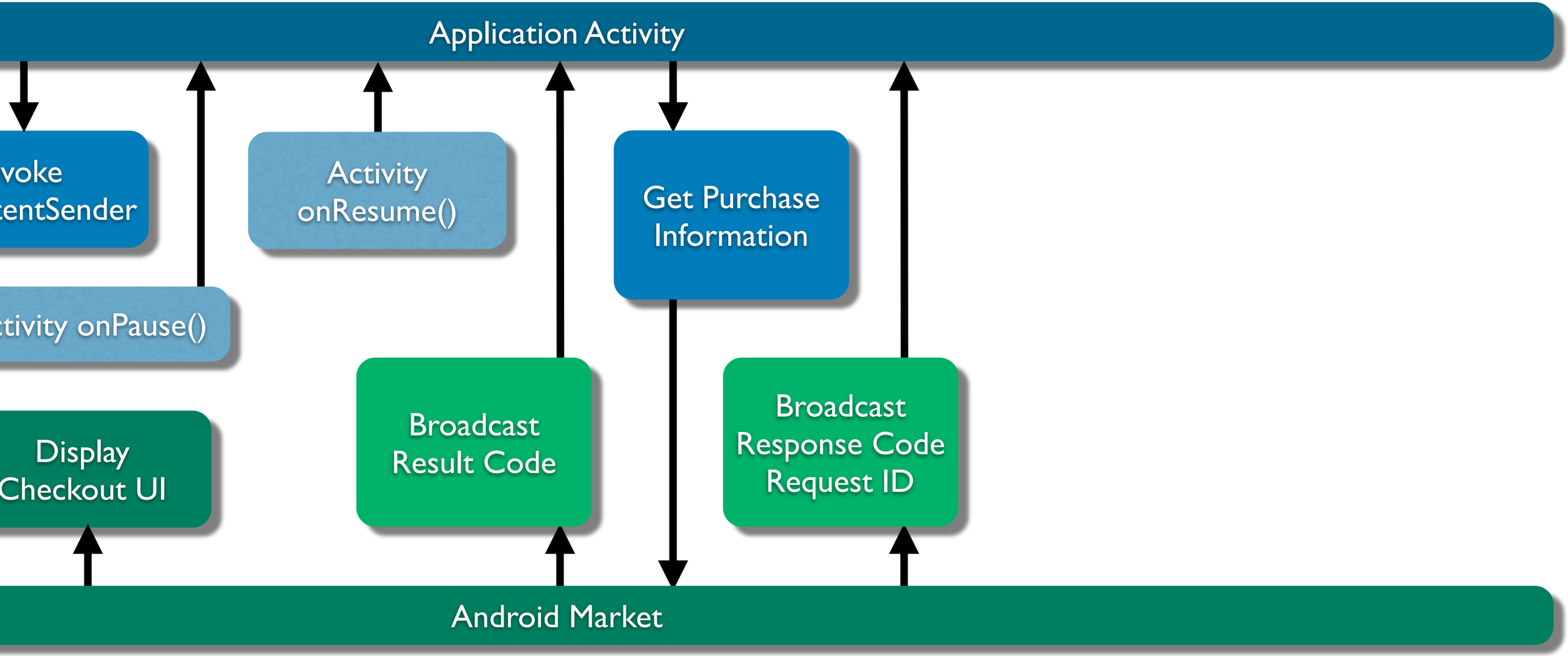
# In-App Billing - Request Purchase



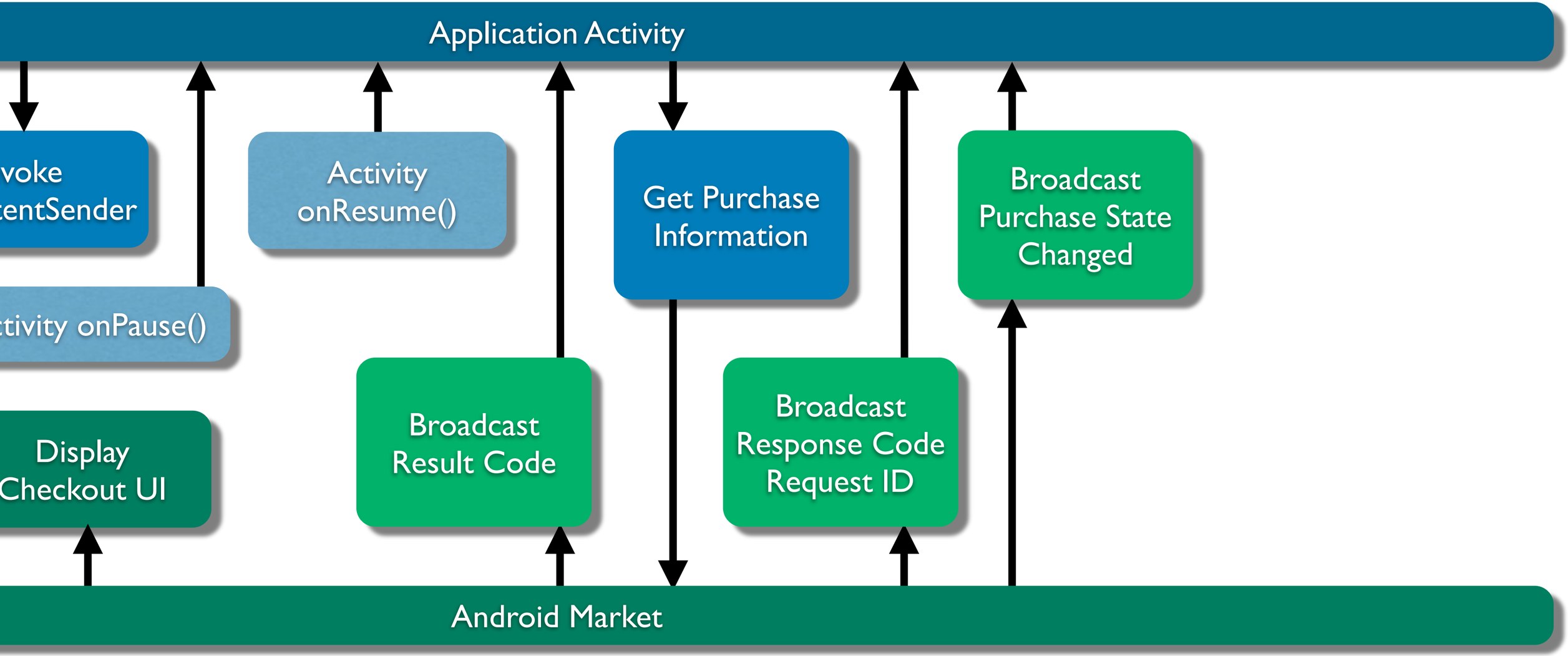
# In-App Billing - Request Purchase



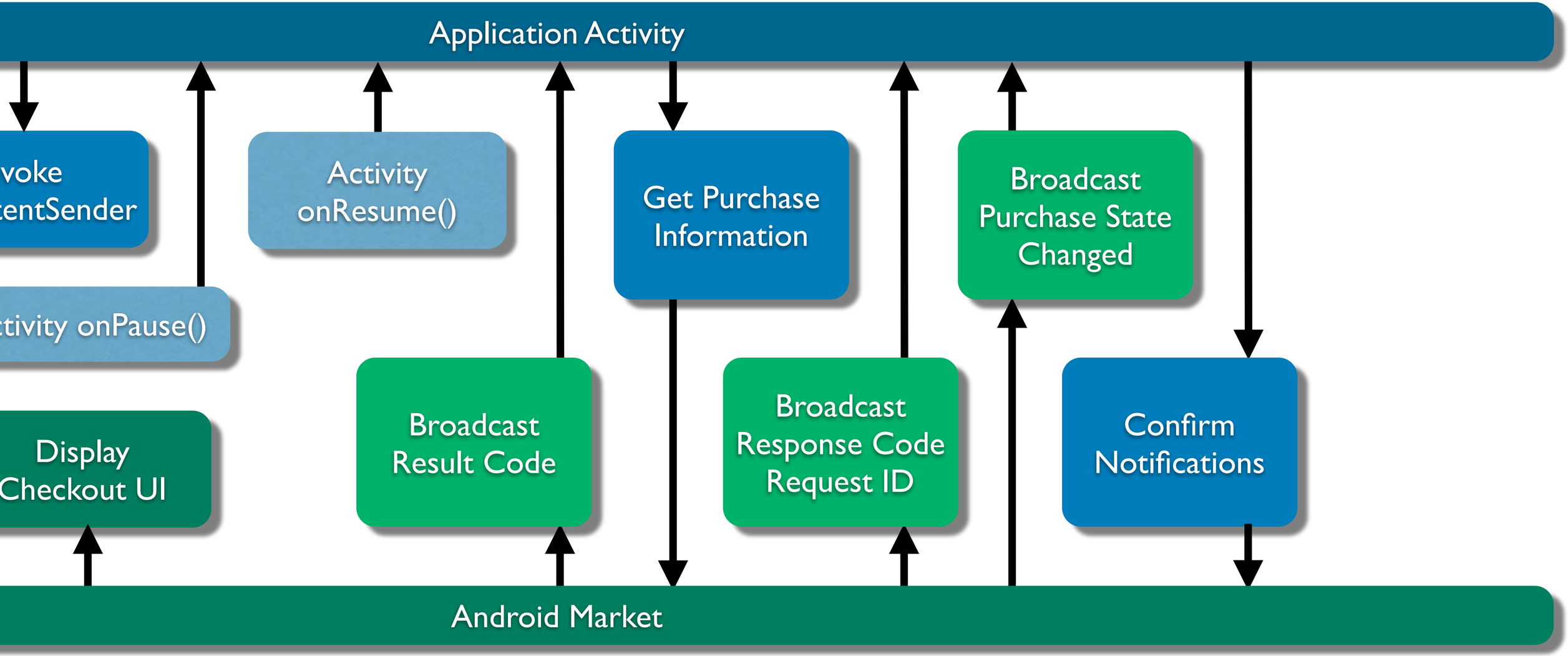
# In-App Billing - Request Purchase



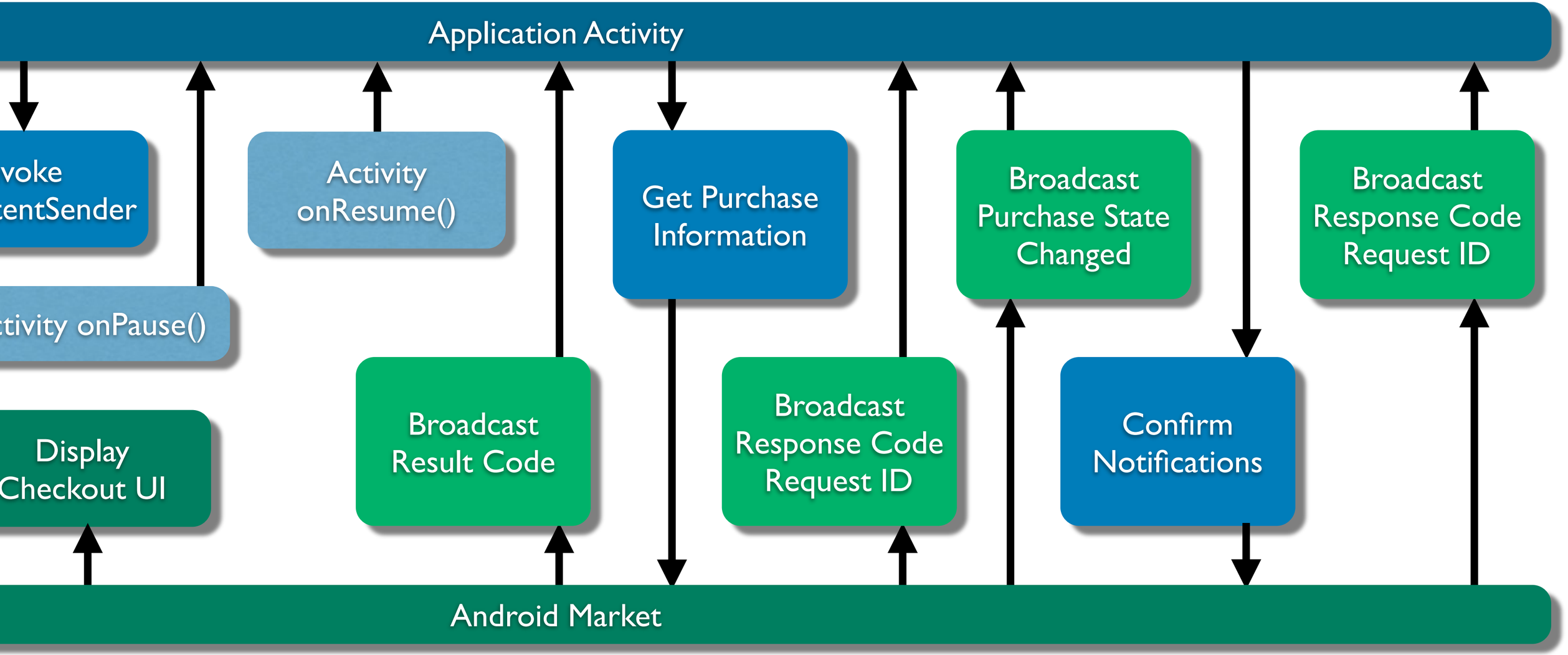
# In-App Billing - Request Purchase



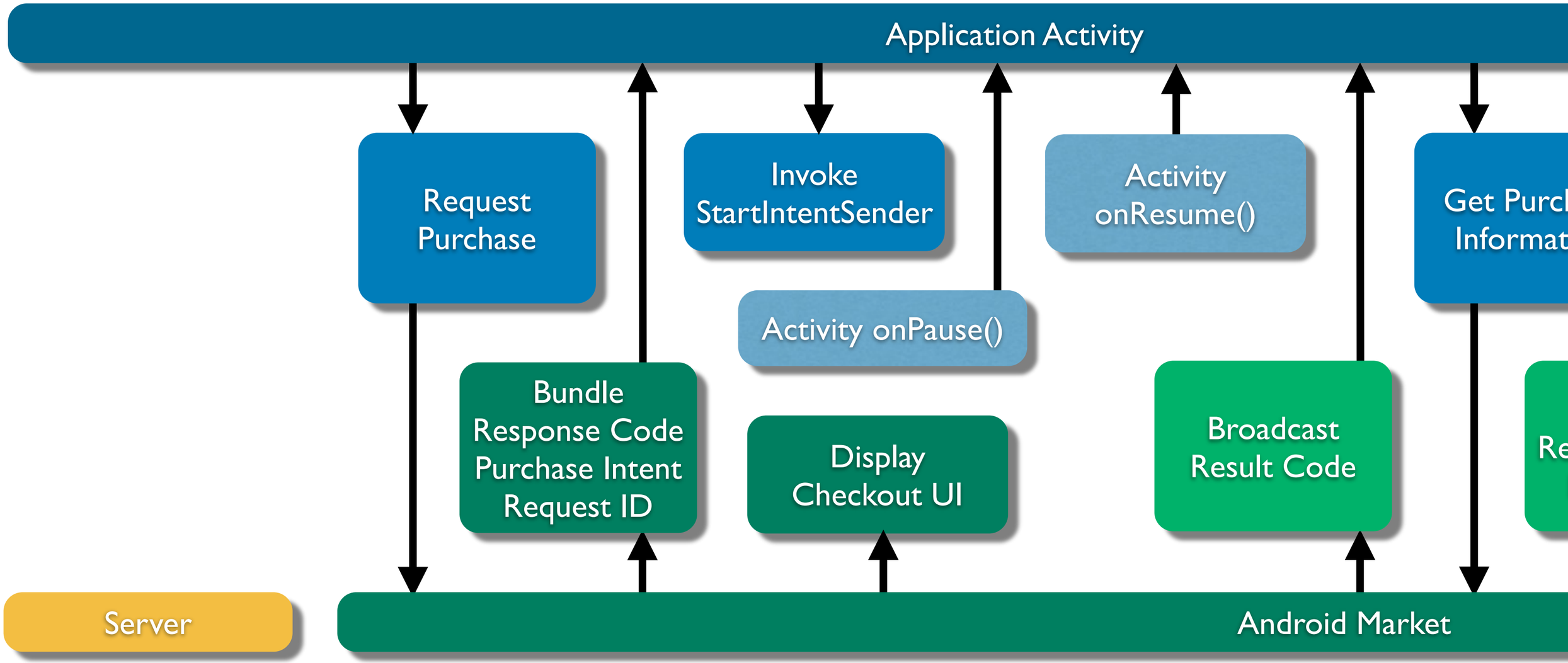
# In-App Billing - Request Purchase



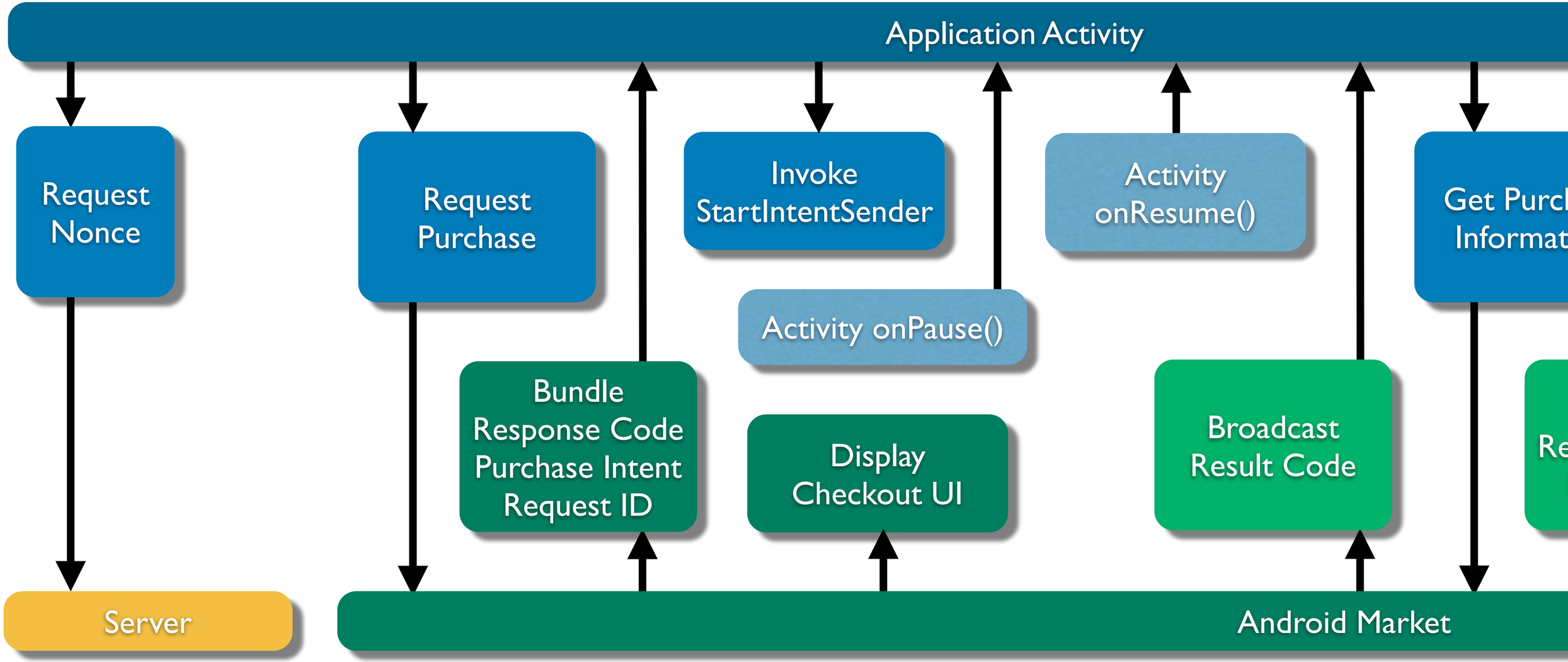
# In-App Billing - Request Purchase



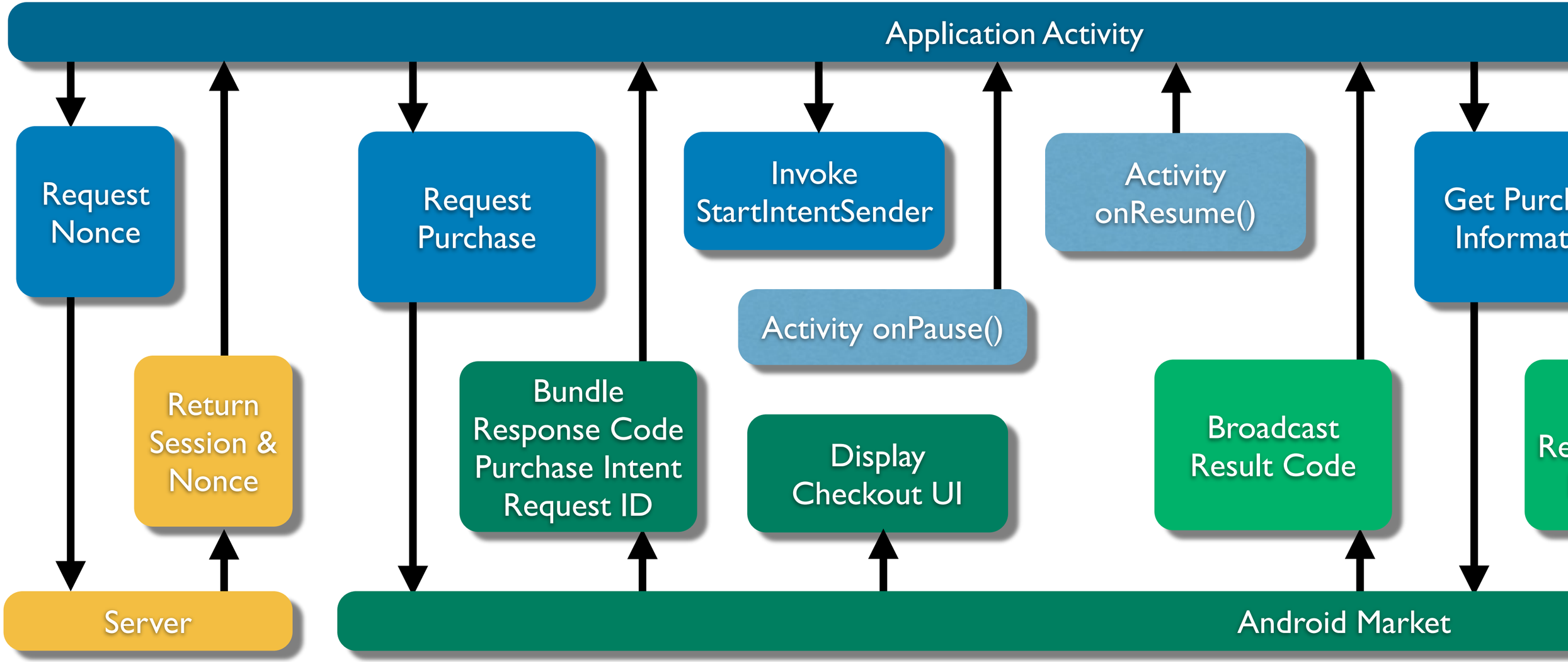
# In-App Billing - Request Purchase with Server



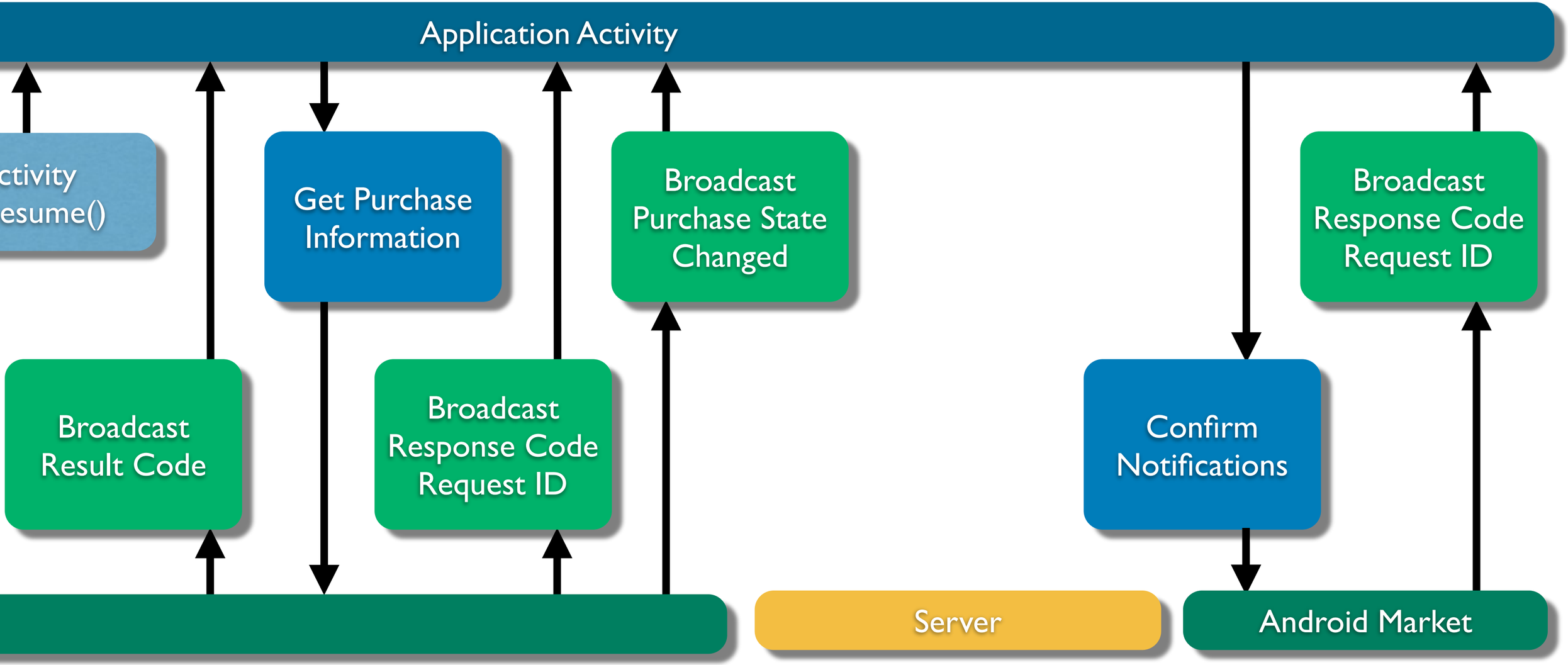
# In-App Billing - Request Purchase with Server



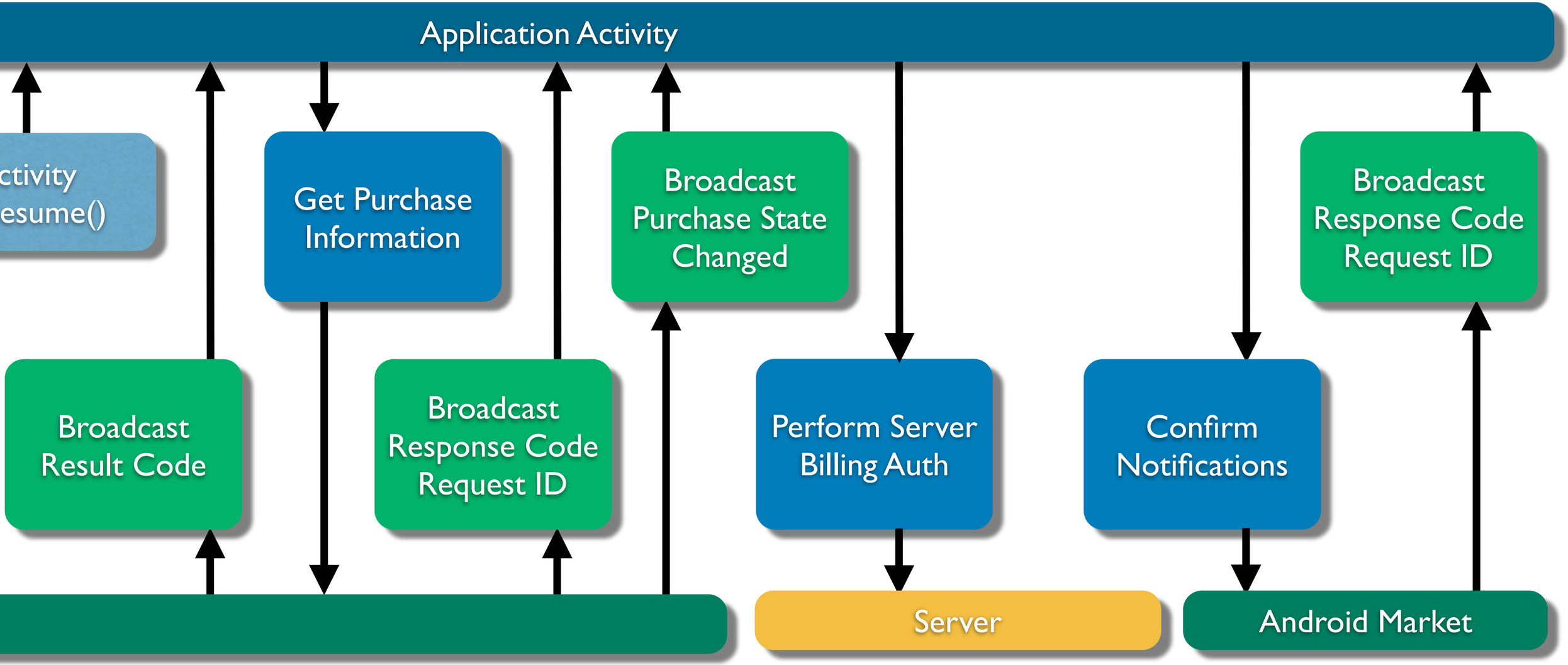
# In-App Billing - Request Purchase with Server



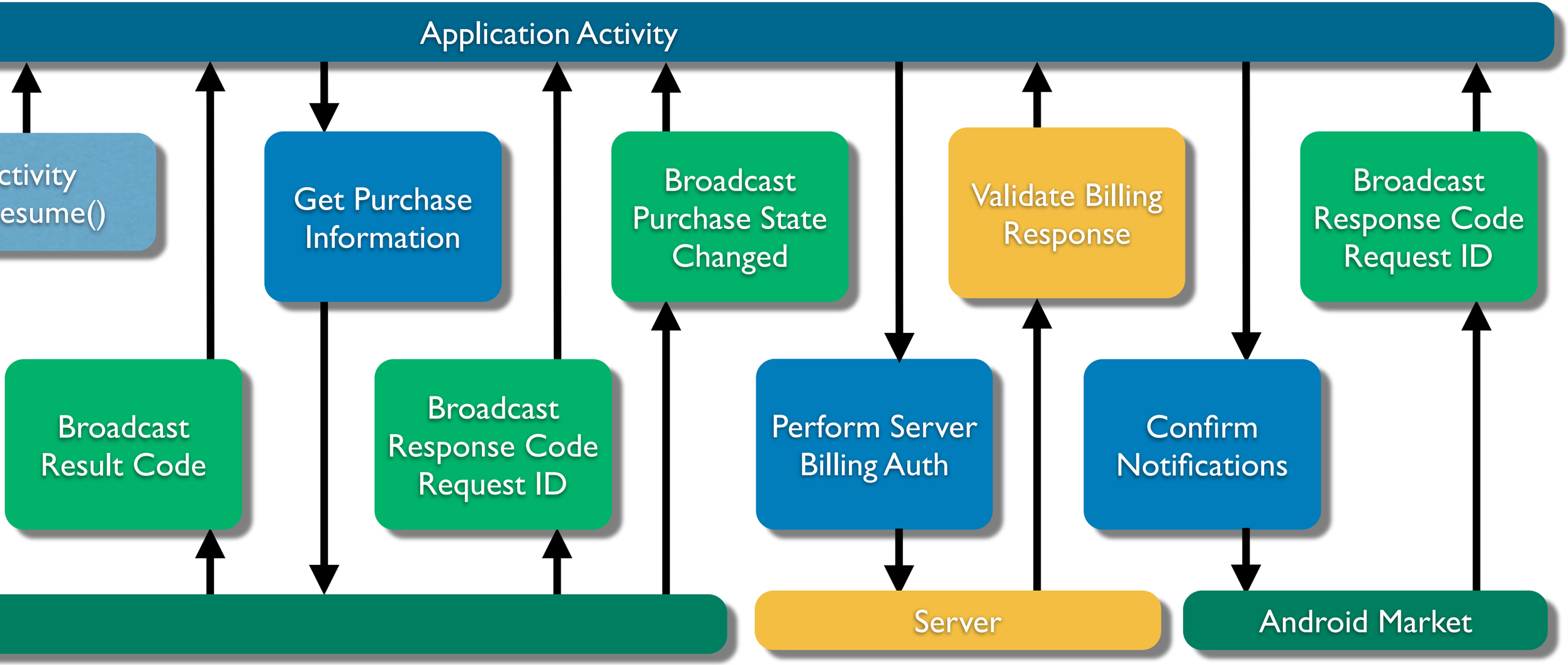
# In-App Billing - Request Purchase with Server



# In-App Billing - Request Purchase with Server



# In-App Billing - Request Purchase with Server

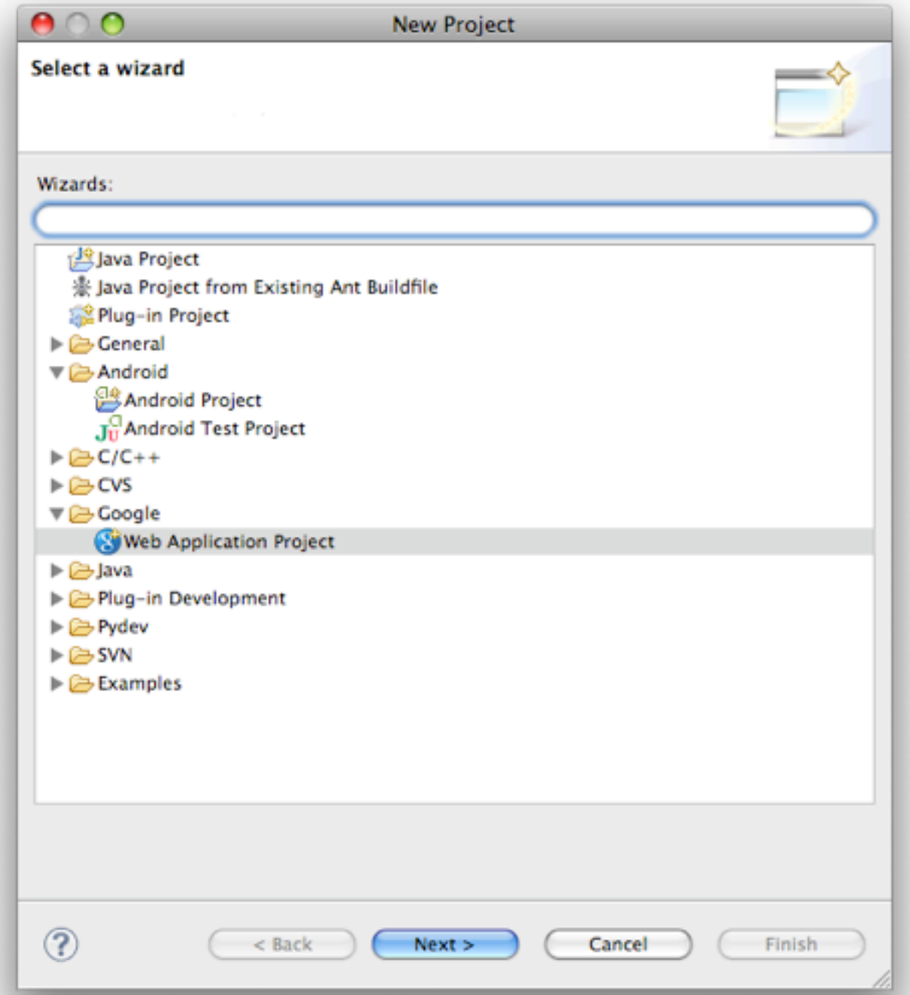


# App Engine can be Your Server

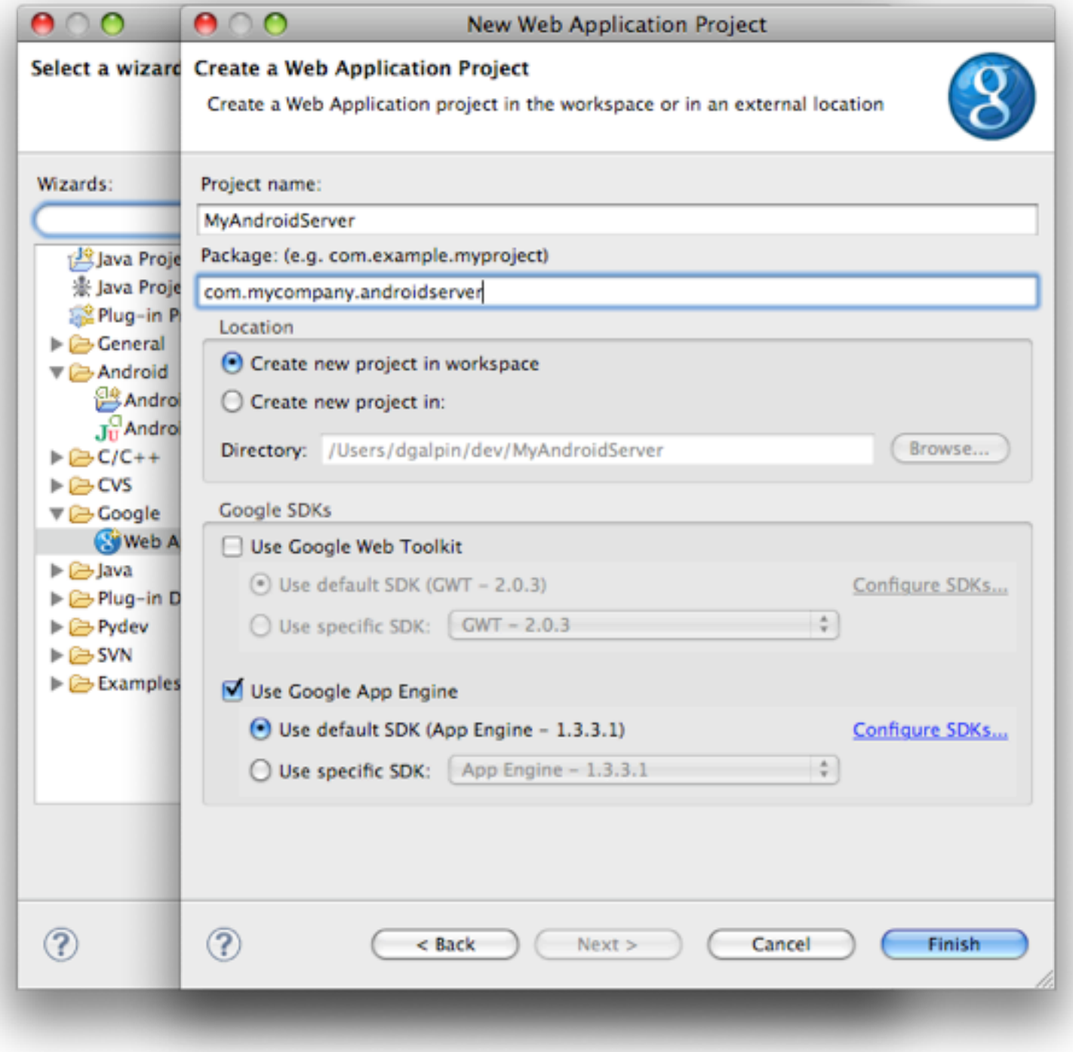
Runs your Java servlet or Python code on our infrastructure

- Validate license responses
- Secure In-App-Billing Transactions
- Serve files from BlobStore
- App Engine is useful as an example.  This can be run easily on any Java-capable server.

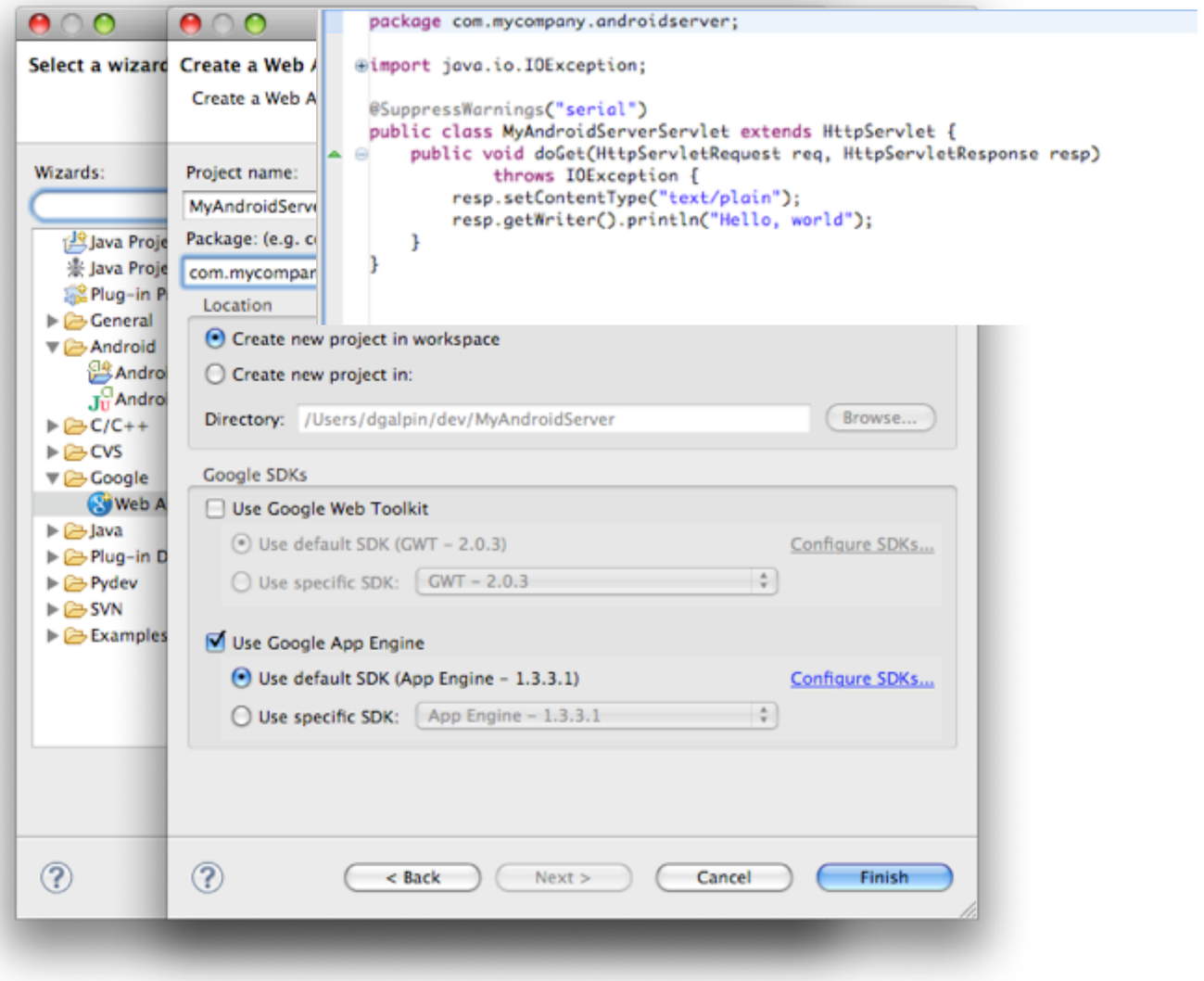
# Creating an AppEngine Project



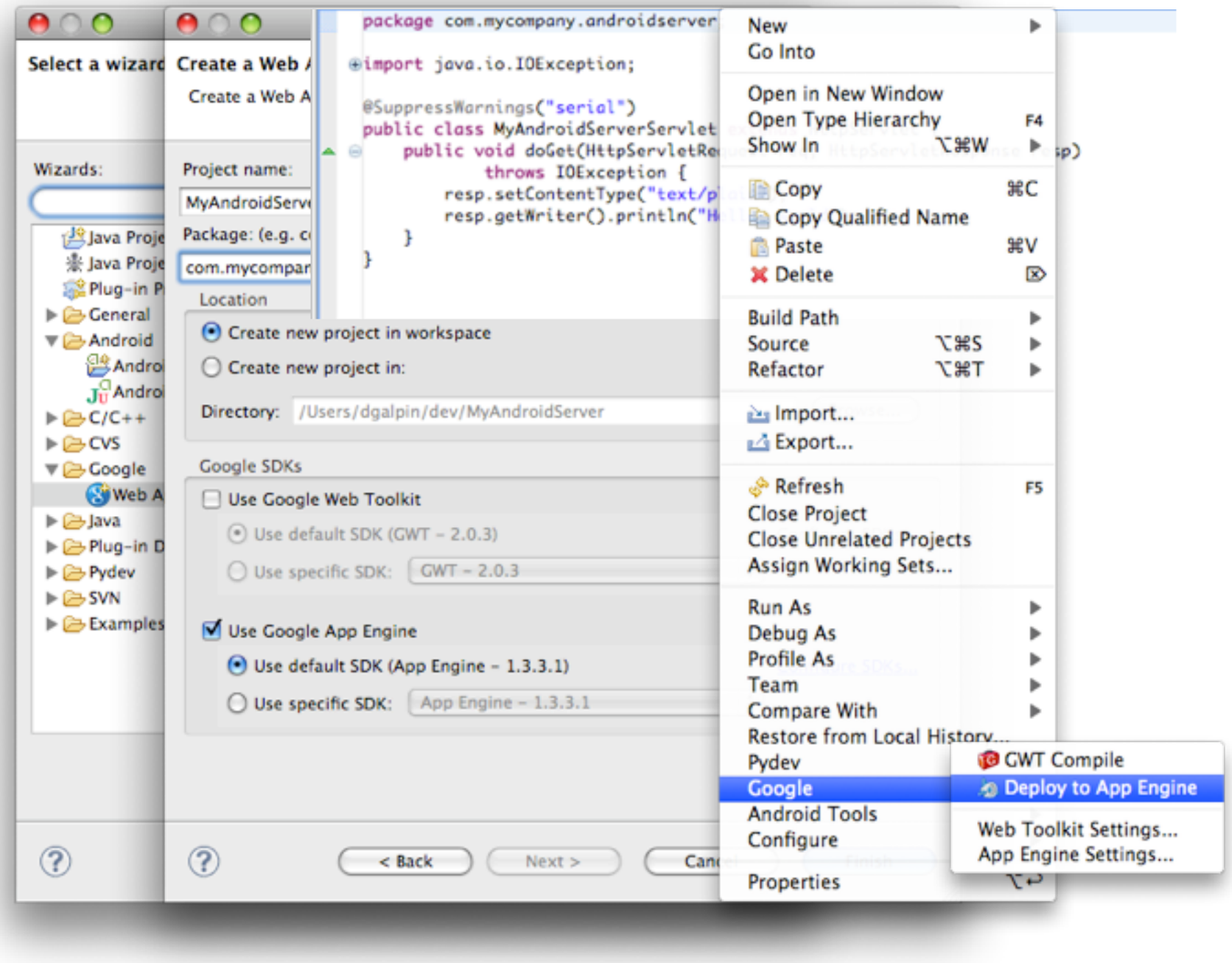
# Creating an AppEngine Project



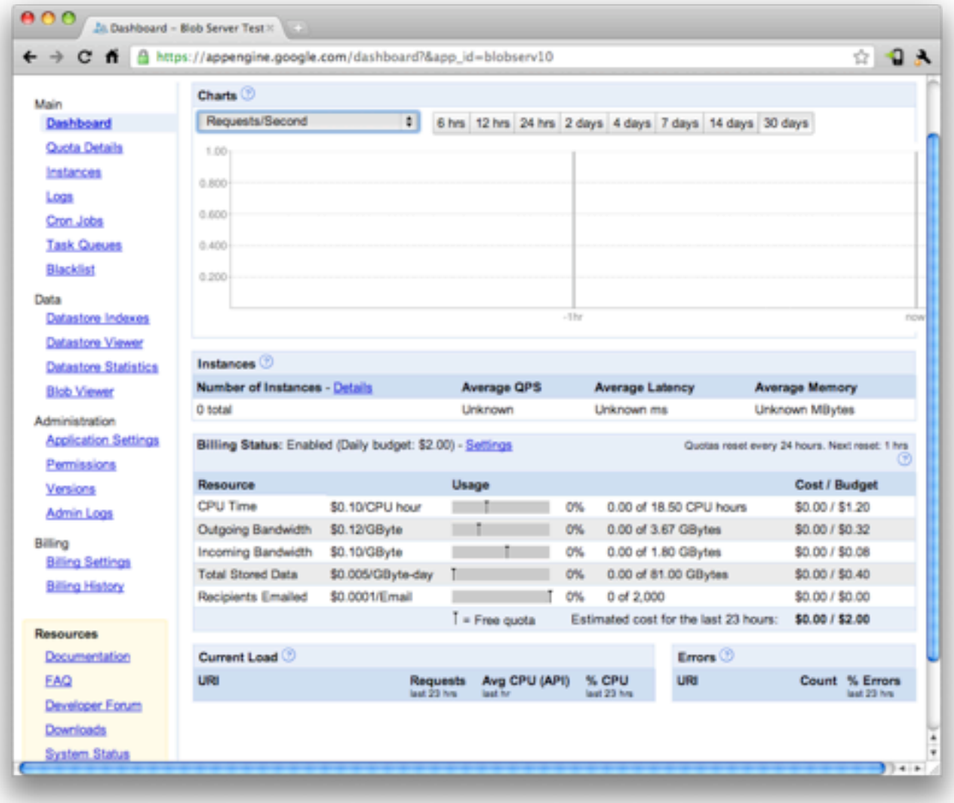
# Creating an AppEngine Project



# Creating an AppEngine Project



# Manage your AppEngine Server



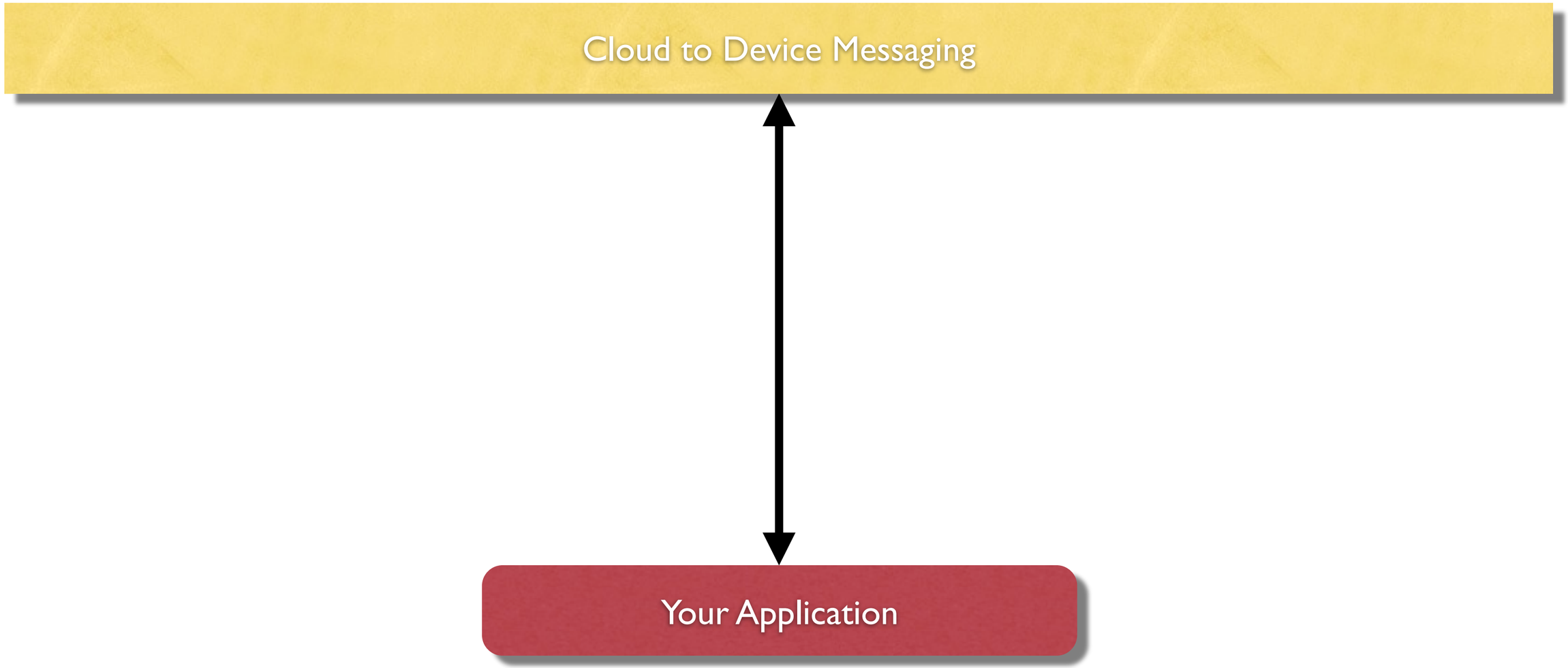
# Server-Side License Validation

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws IOException {
        // Obtain signed license server data as a POST parameter
        String license = req.getParameter("license");
        String signature = req.getParameter("signature");
        String versionCode = req.getParameter("version-code");

        // Extract nonce from session
        HttpSession session = req.getSession(true);
        Object nonceObject = session.getAttribute("nonce");
        session.setAttribute("nonce", null); // Clear nonce so it's only valid once

        int nonce = ((Integer) nonceObject).intValue();

        // Verify signature for license
        LicenseValidator lv = new LicenseValidator(new NullDeviceLimiter(), nonce, mPackageName, versionCode);
        LicenseStatus status = lv.verify(mPublicKey, license, signature);

        session.setAttribute("status", status);
        if (status == LicenseStatus.LICENSED) { resp.getOutputStream().print("ok"); }
        else if (status == LicenseStatus.NOT_LICENSED) { resp.sendError(403, "User not licensed"); }
        else if (status == LicenseStatus.INVALID_SIGNATURE) { resp.sendError(400, "Invalid signature"); }
        else { resp.sendError(500, "Error decoding license data"); }
    }
```

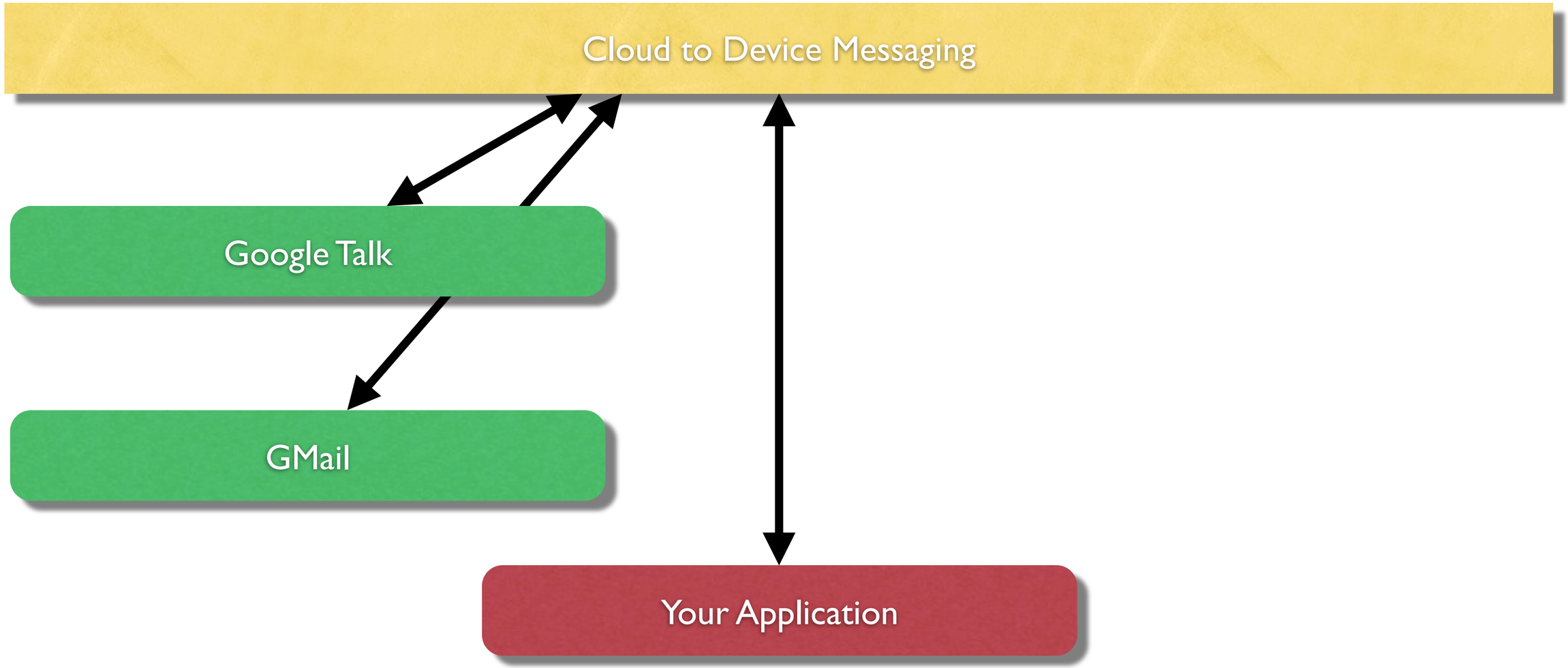
# Android Cloud Services

Your Application

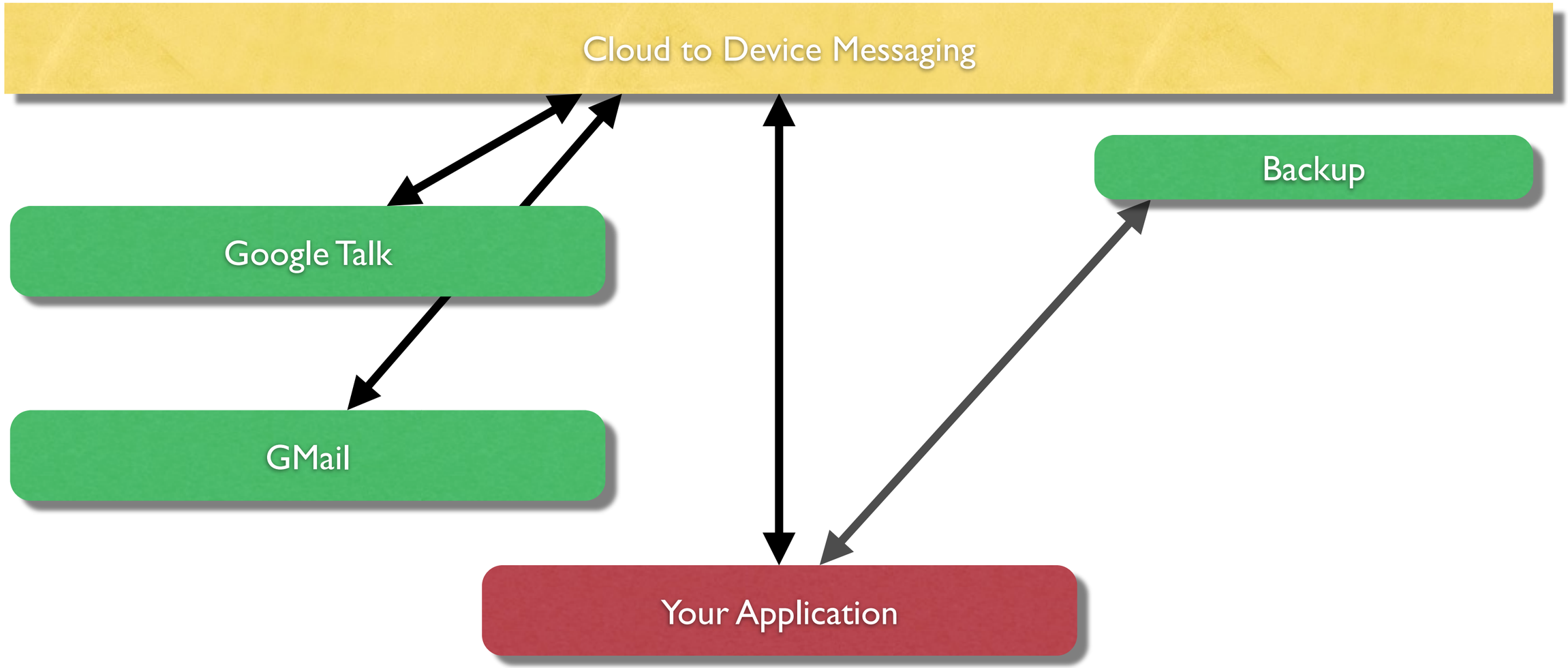
# Android Cloud Services



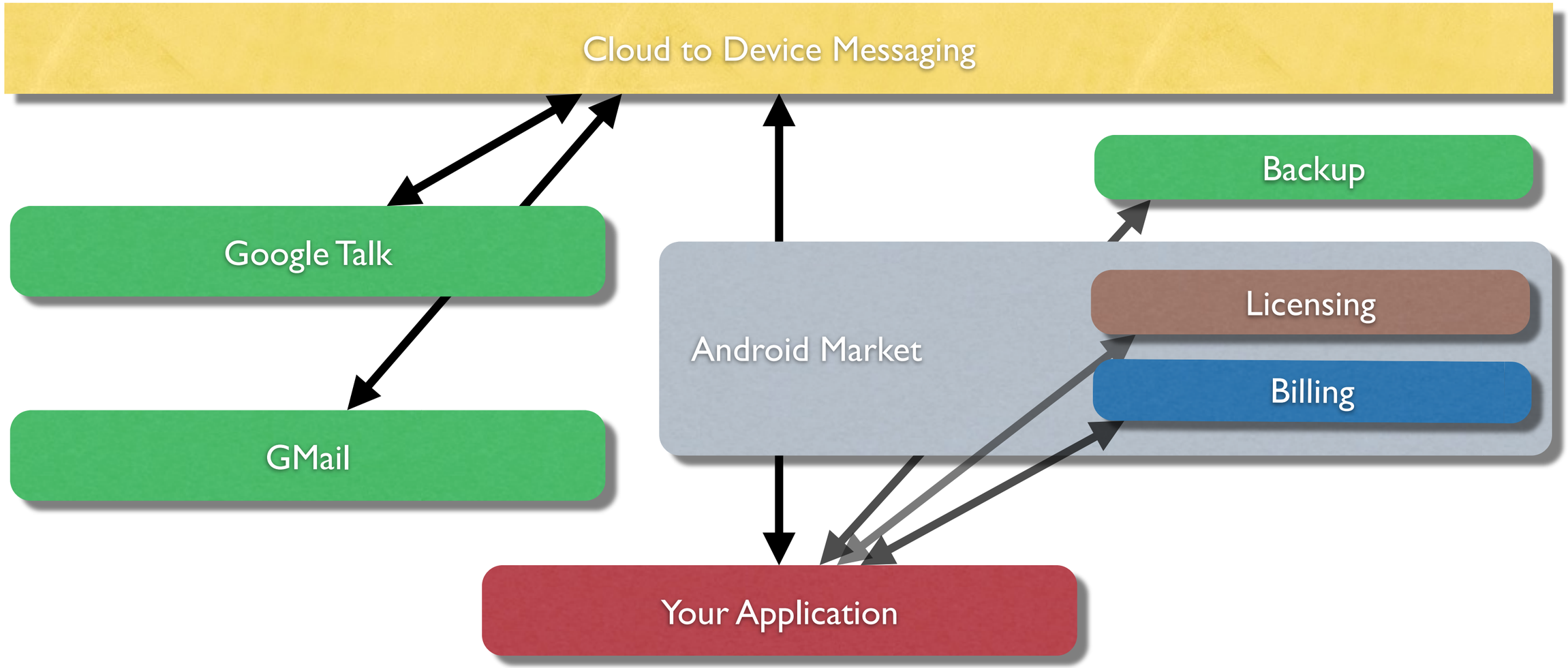
# Android Cloud Services



# Android Cloud Services



# Android Cloud Services



# Cloud to Device Messaging

- Sends lightweight messages to Android applications
- Queues and delivers messages to the target device
- Launches the application if it is not running
- Leverages the network connection being used for Google services

# Cloud to Device Messaging Basic Flows

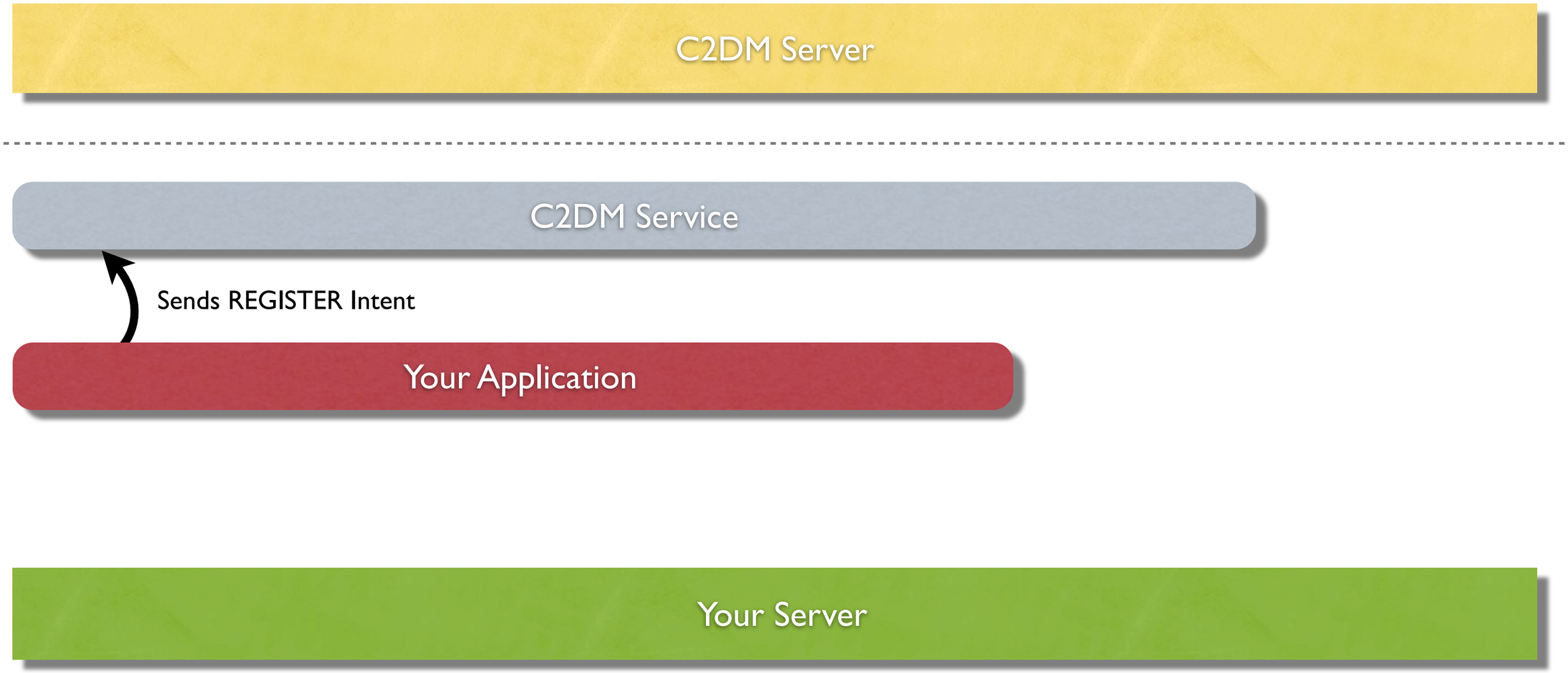
C2DM Server

C2DM Service

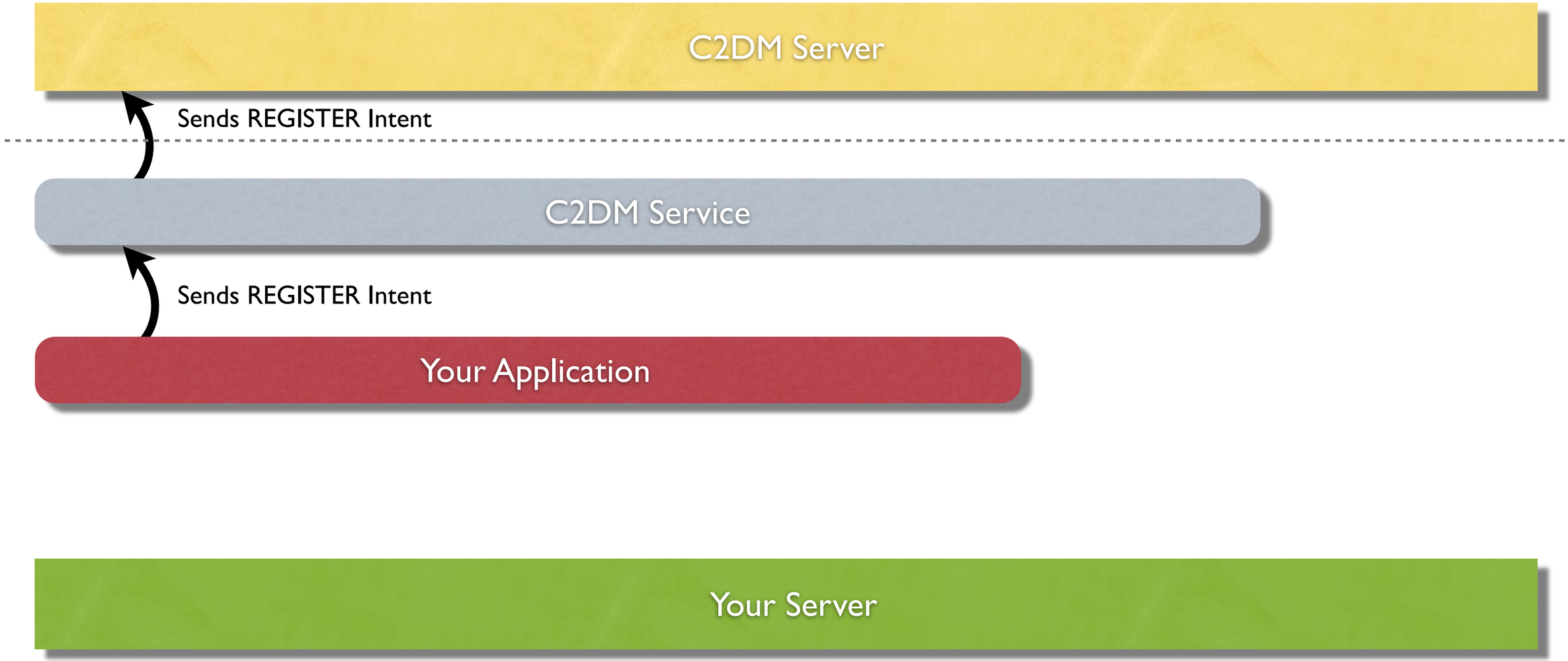
Your Application

Your Server

# Cloud to Device Messaging Basic Flows

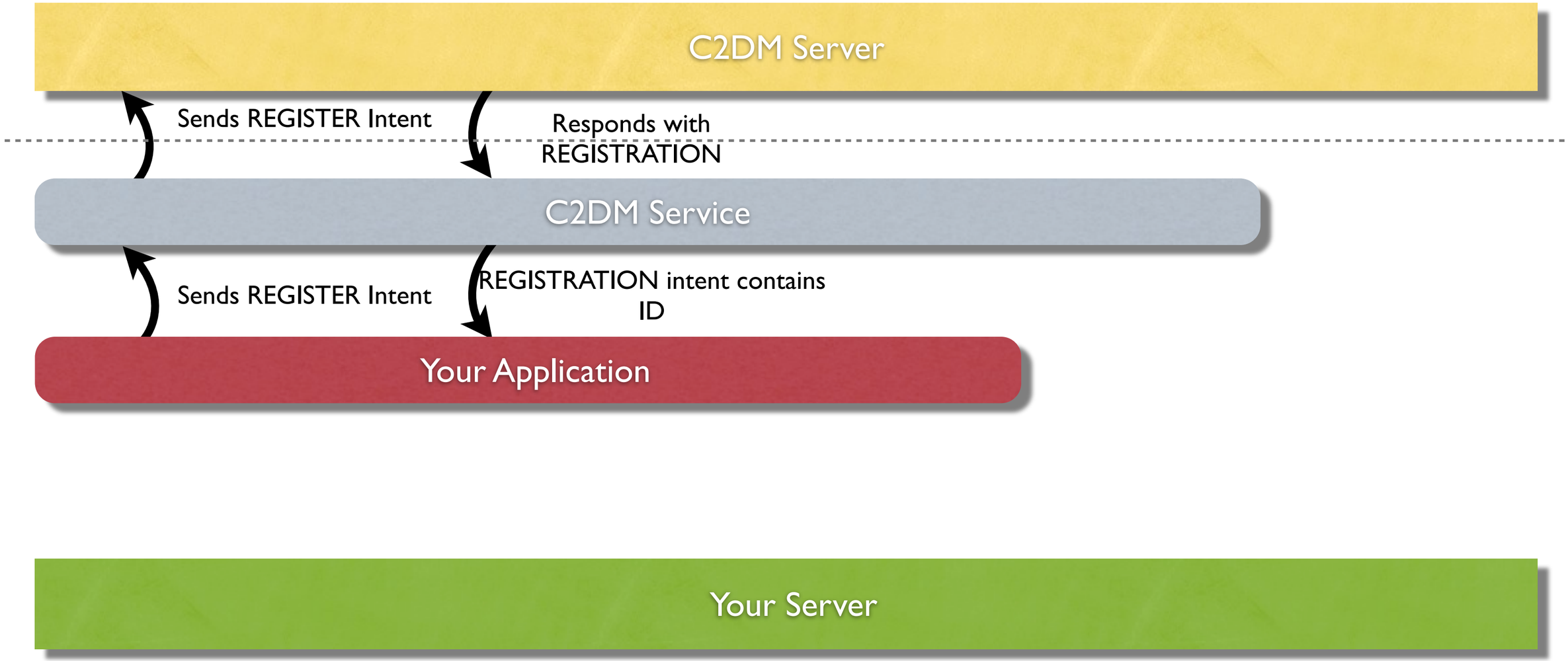


# Cloud to Device Messaging Basic Flows

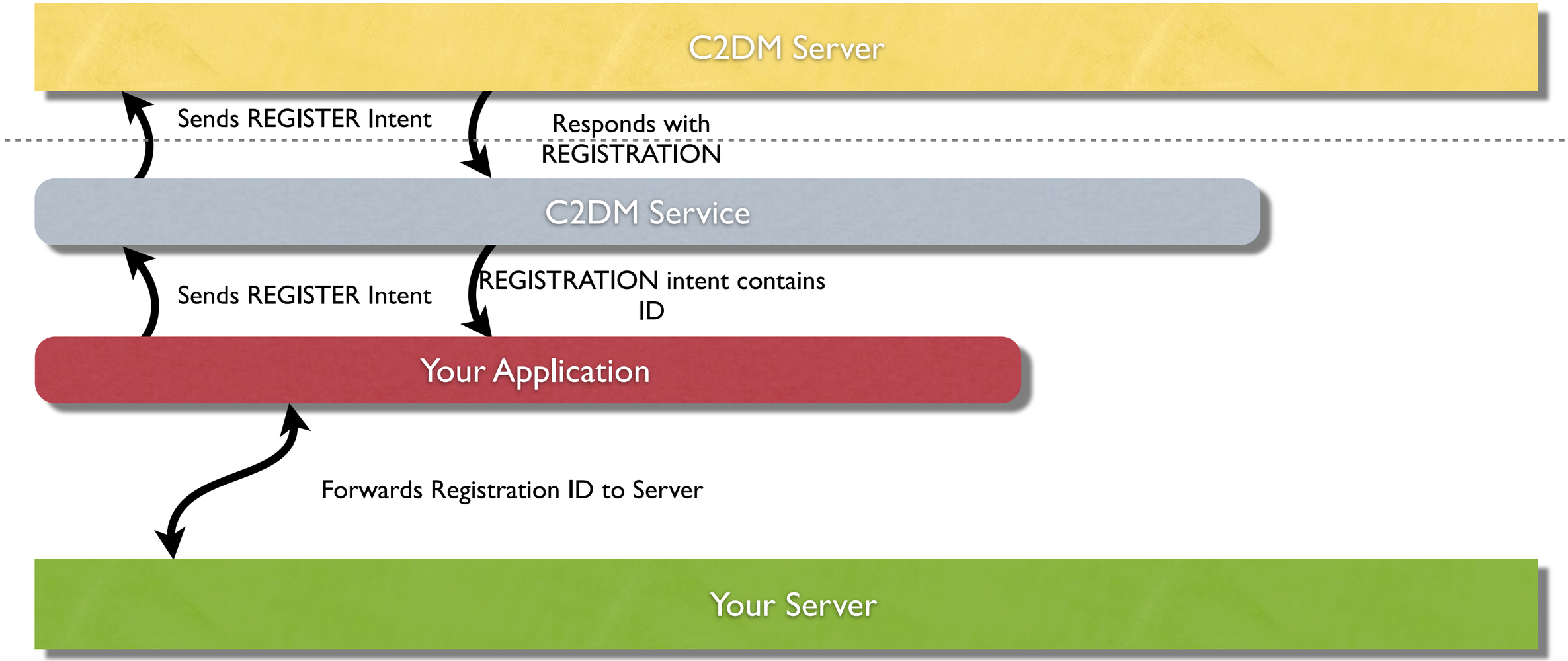


# Cloud to Device Messaging Basic Flows

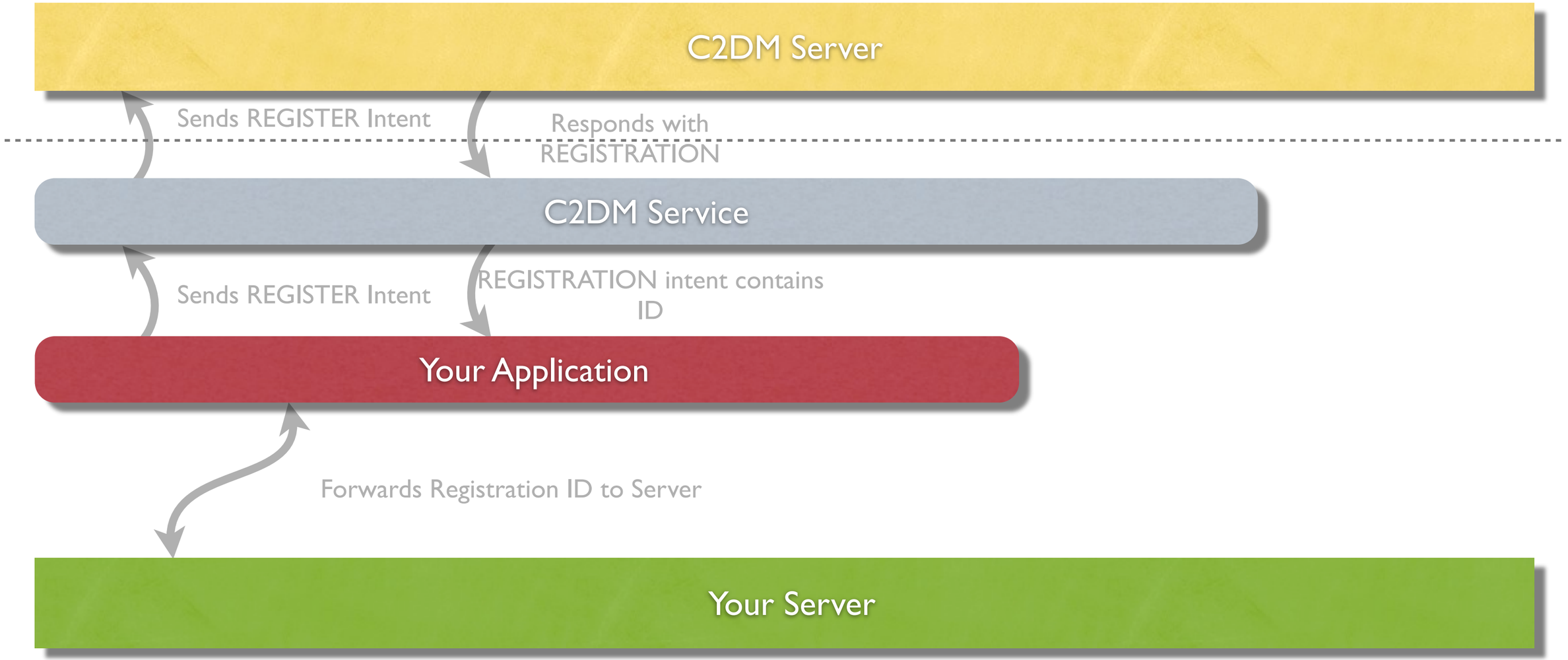
# Cloud to Device Messaging Basic Flows



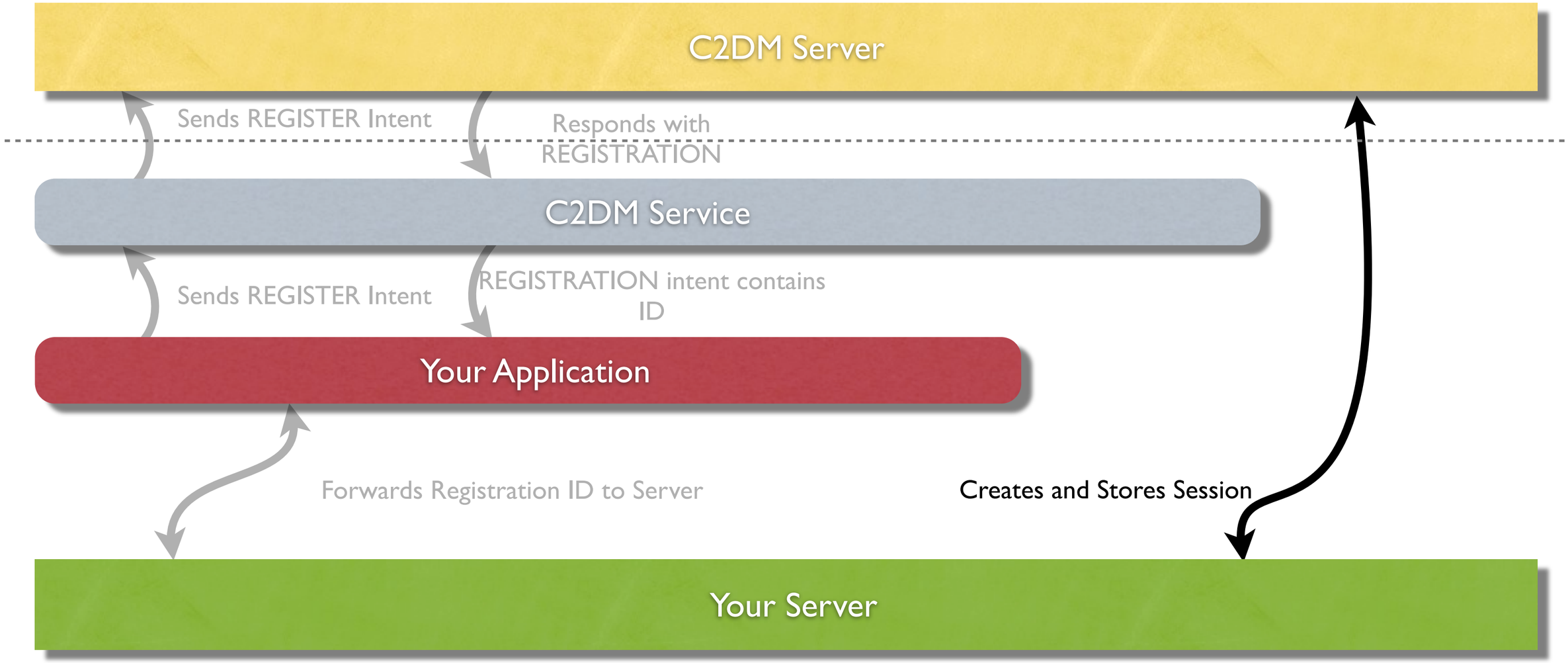
# Cloud to Device Messaging Basic Flows



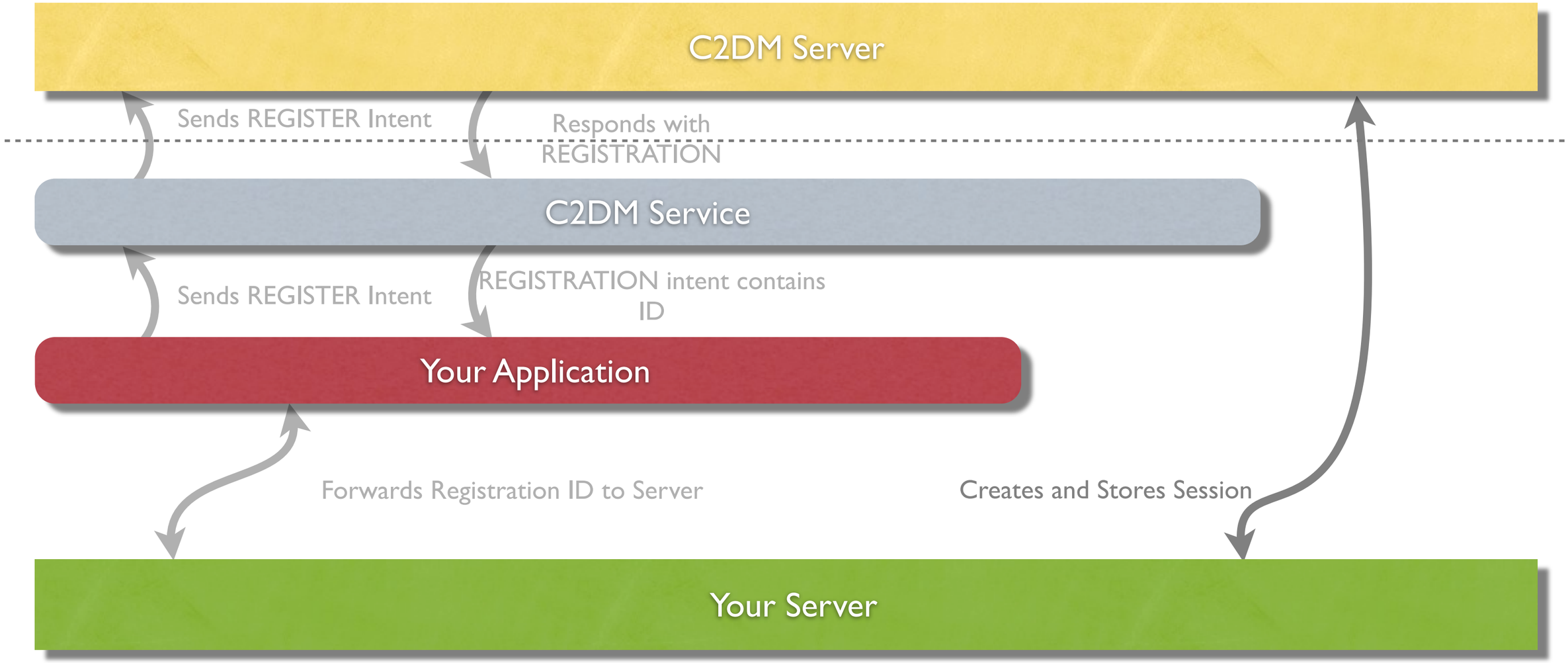
# Cloud to Device Messaging Basic Flows



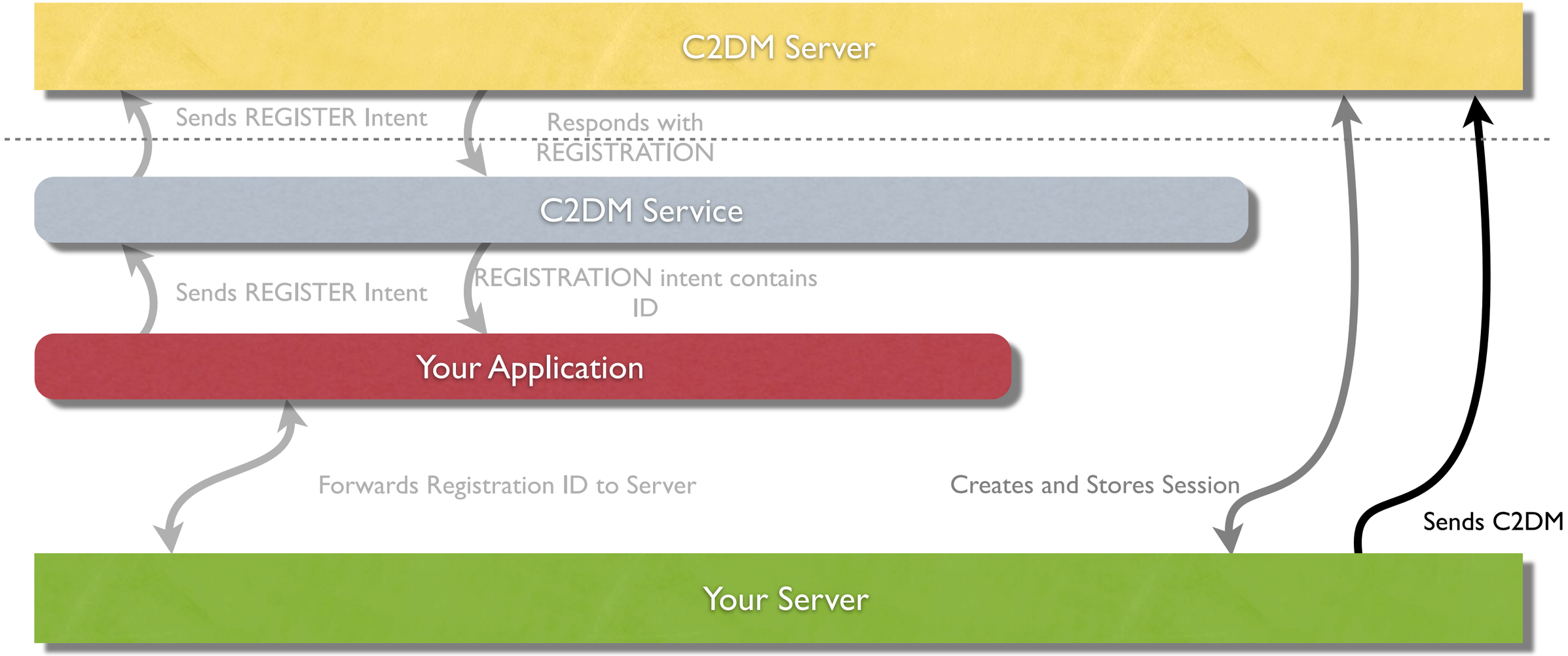
# Cloud to Device Messaging Basic Flows



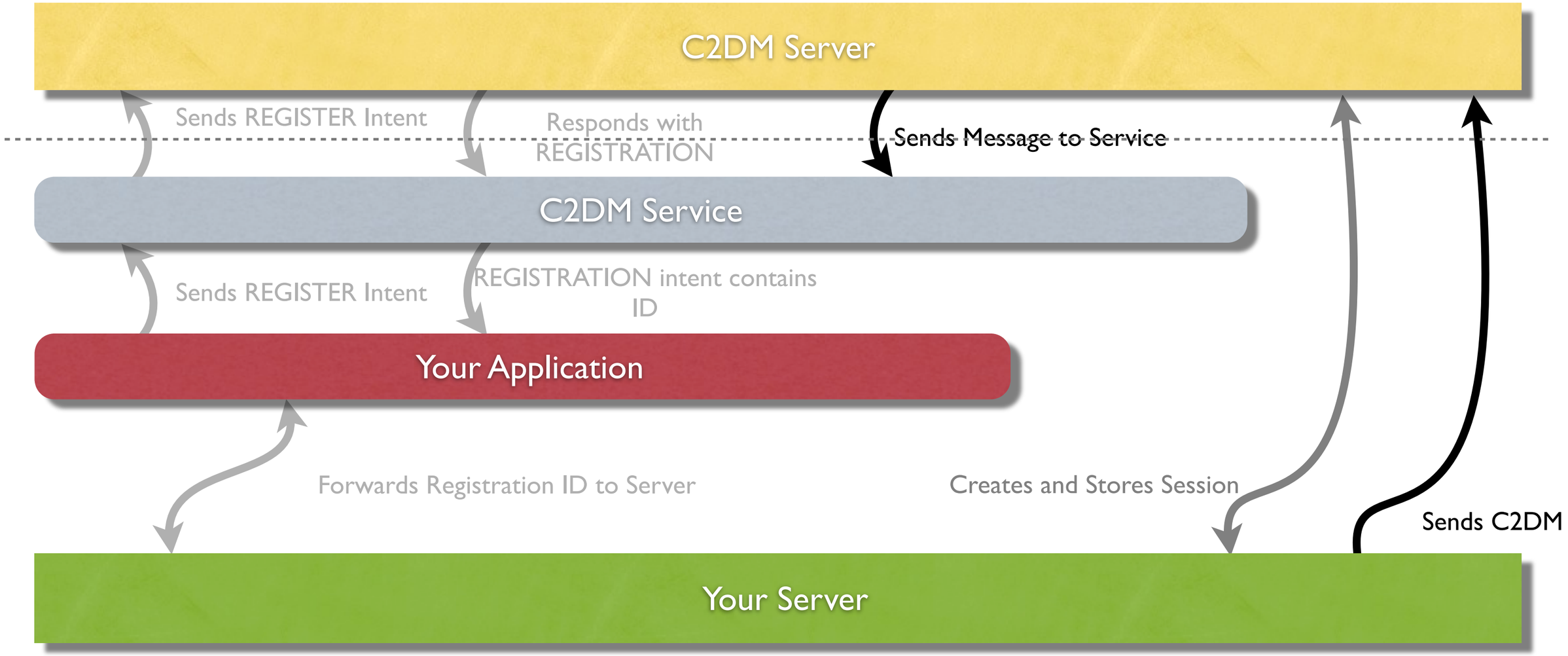
# Cloud to Device Messaging Basic Flows



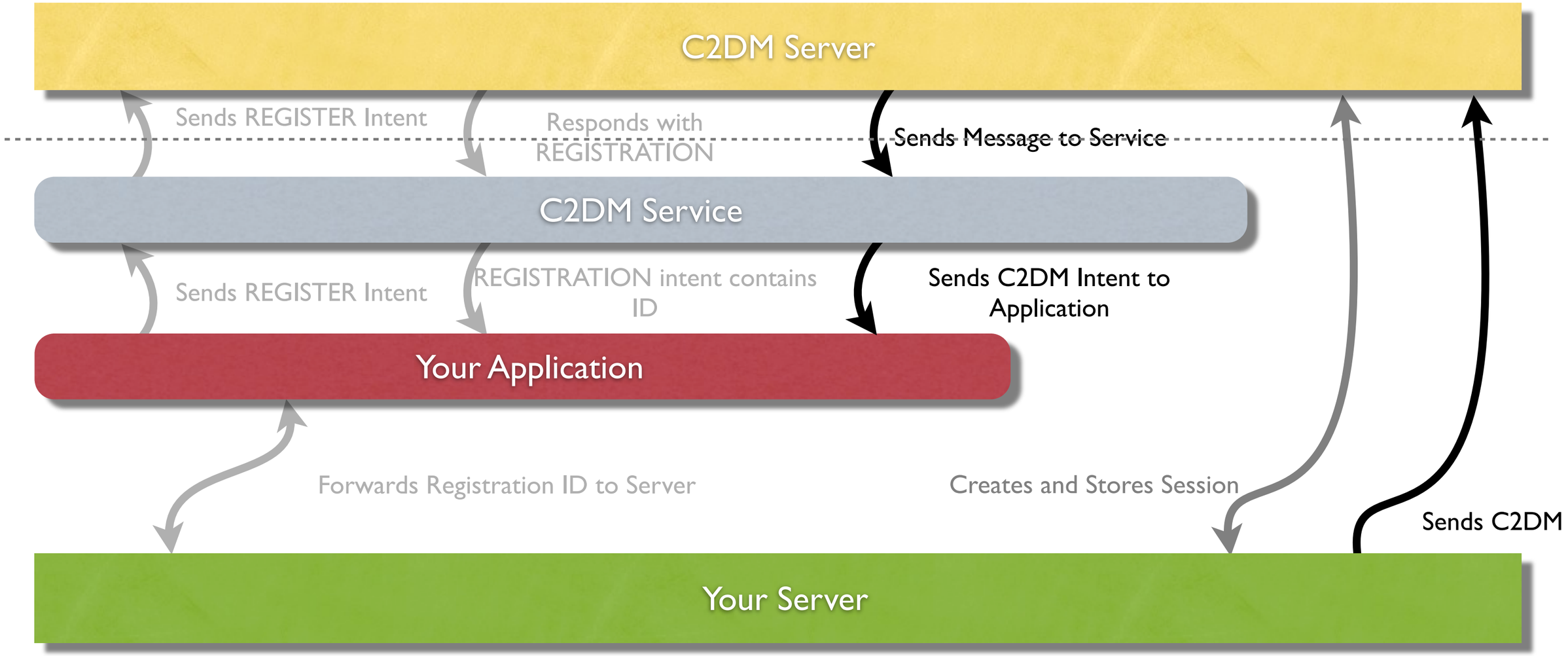
# Cloud to Device Messaging Basic Flows



# Cloud to Device Messaging Basic Flows



# Cloud to Device Messaging Basic Flows



# Application Data Backup

- Copies application data to cloud
- Backup is requested by application
- Automatically restored when application is installed
- Application Registers itself with Backup Service (server) and Backup Manager (client)

# Android Cloud Backup - Backing Up

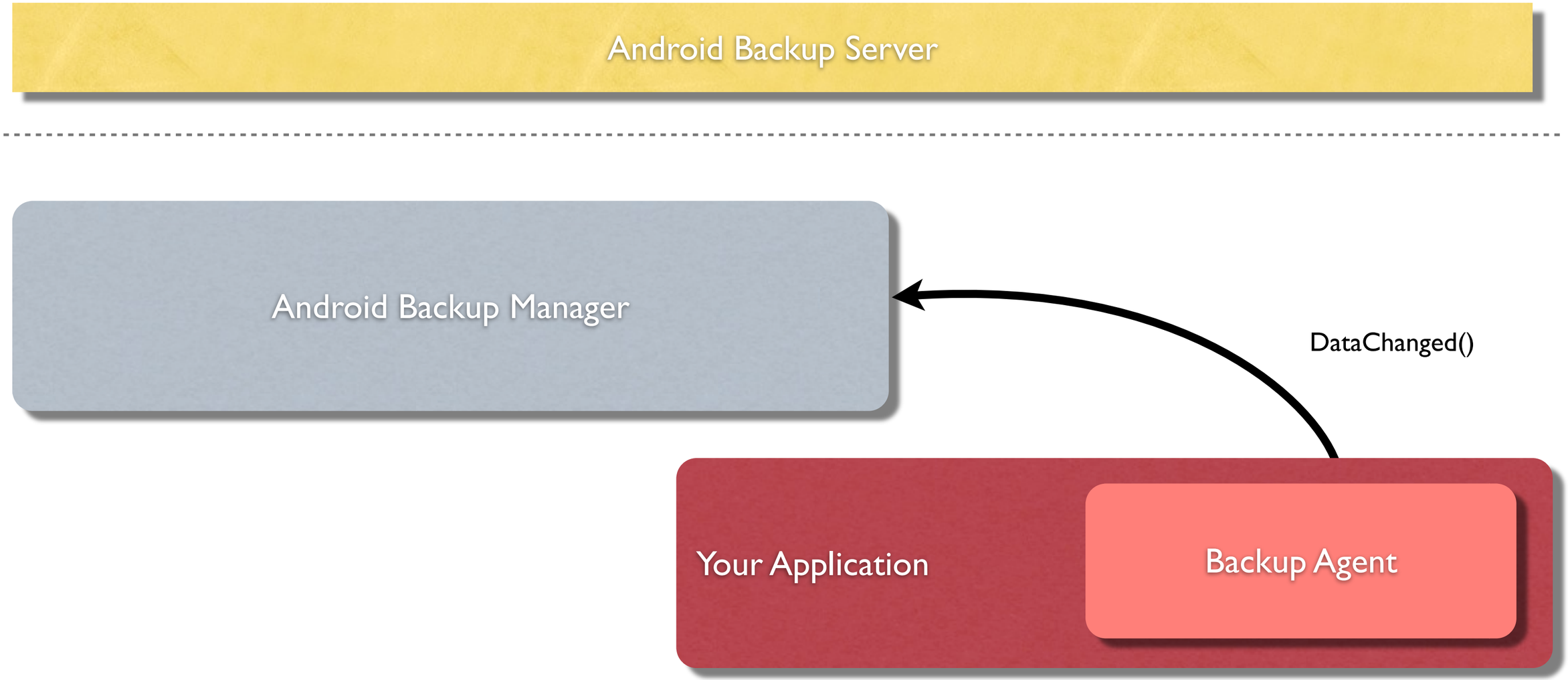
Android Backup Server

Android Backup Manager

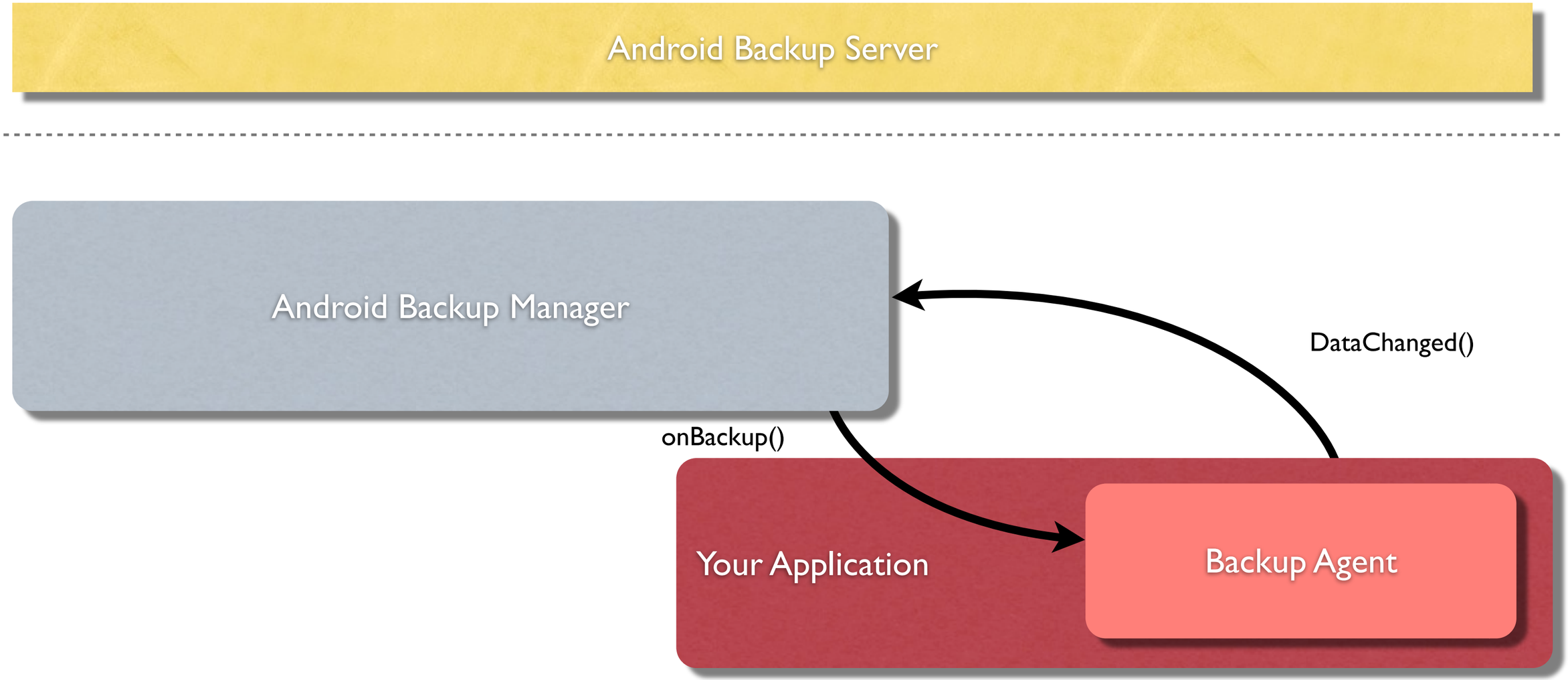
Your Application

Backup Agent

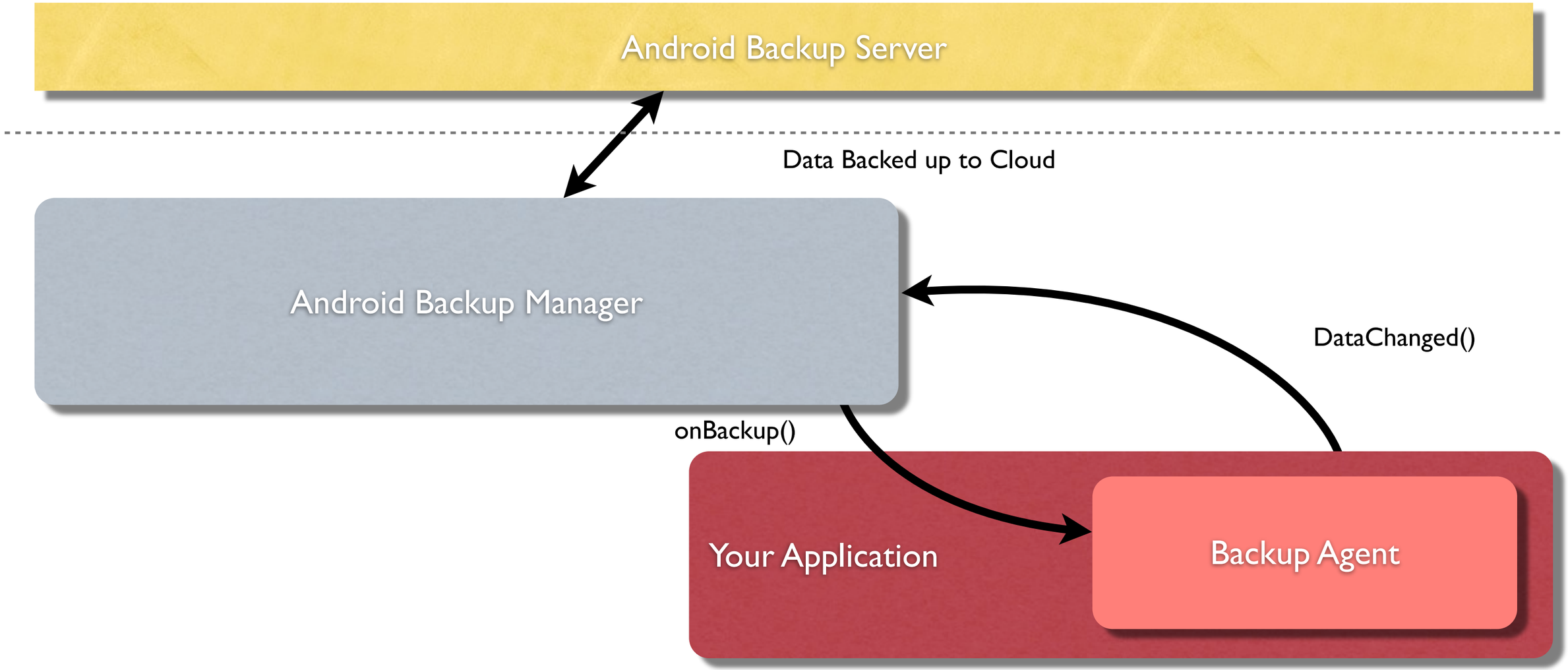
# Android Cloud Backup - Backing Up



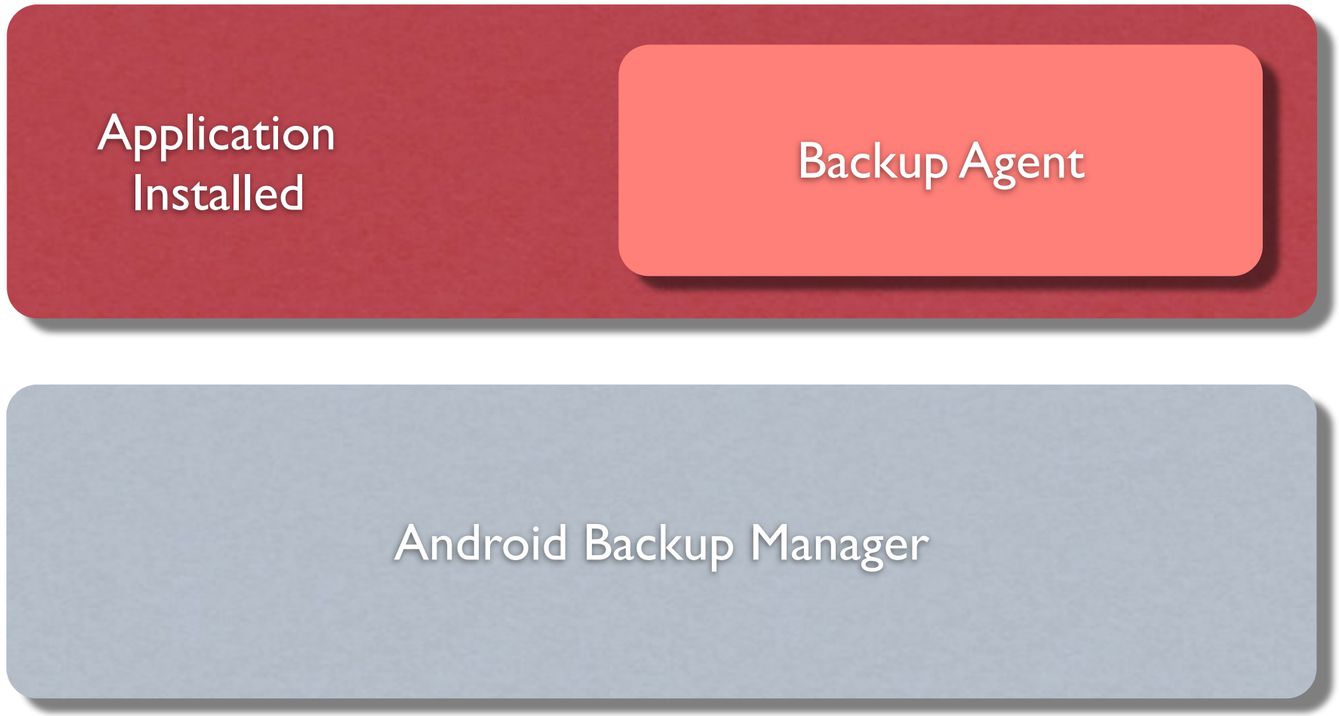
# Android Cloud Backup - Backing Up



# Android Cloud Backup - Backing Up

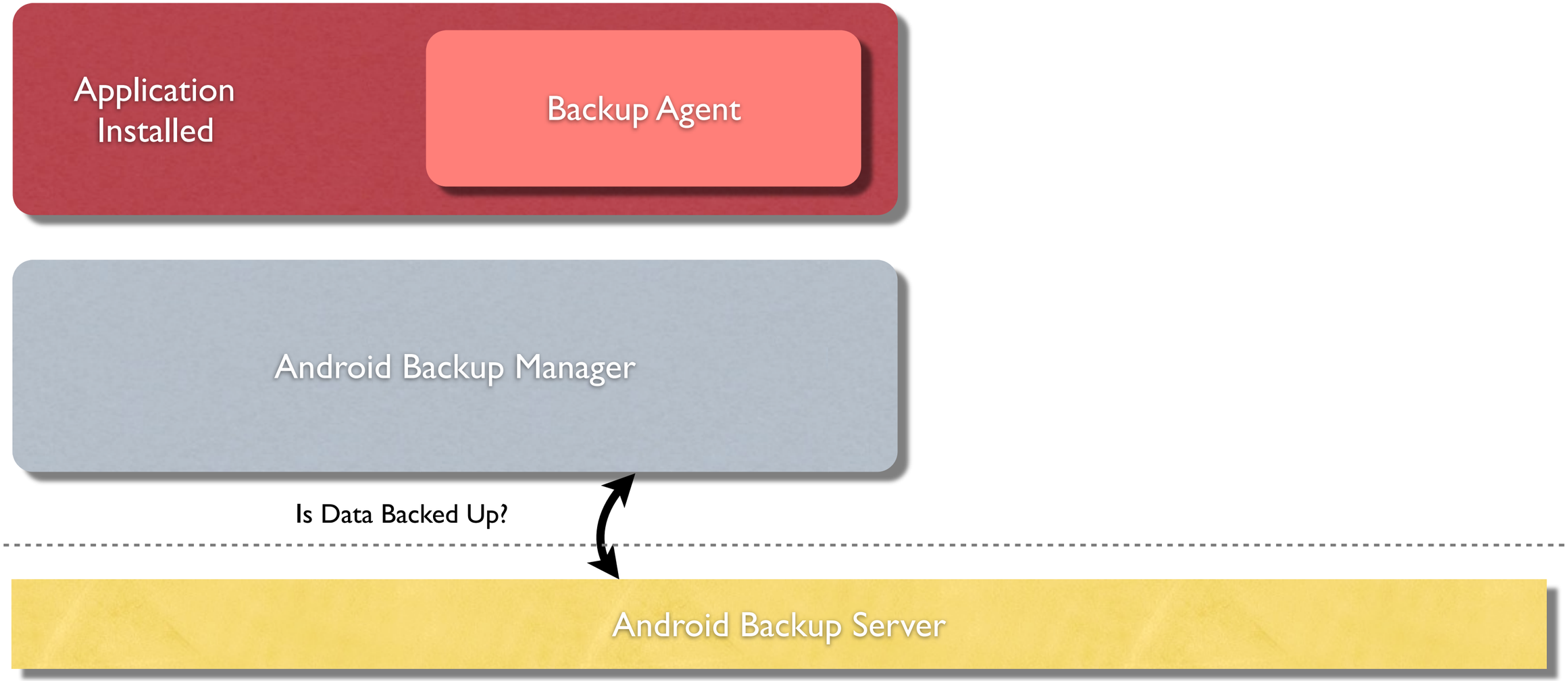


# Android Cloud Backup - Restoring

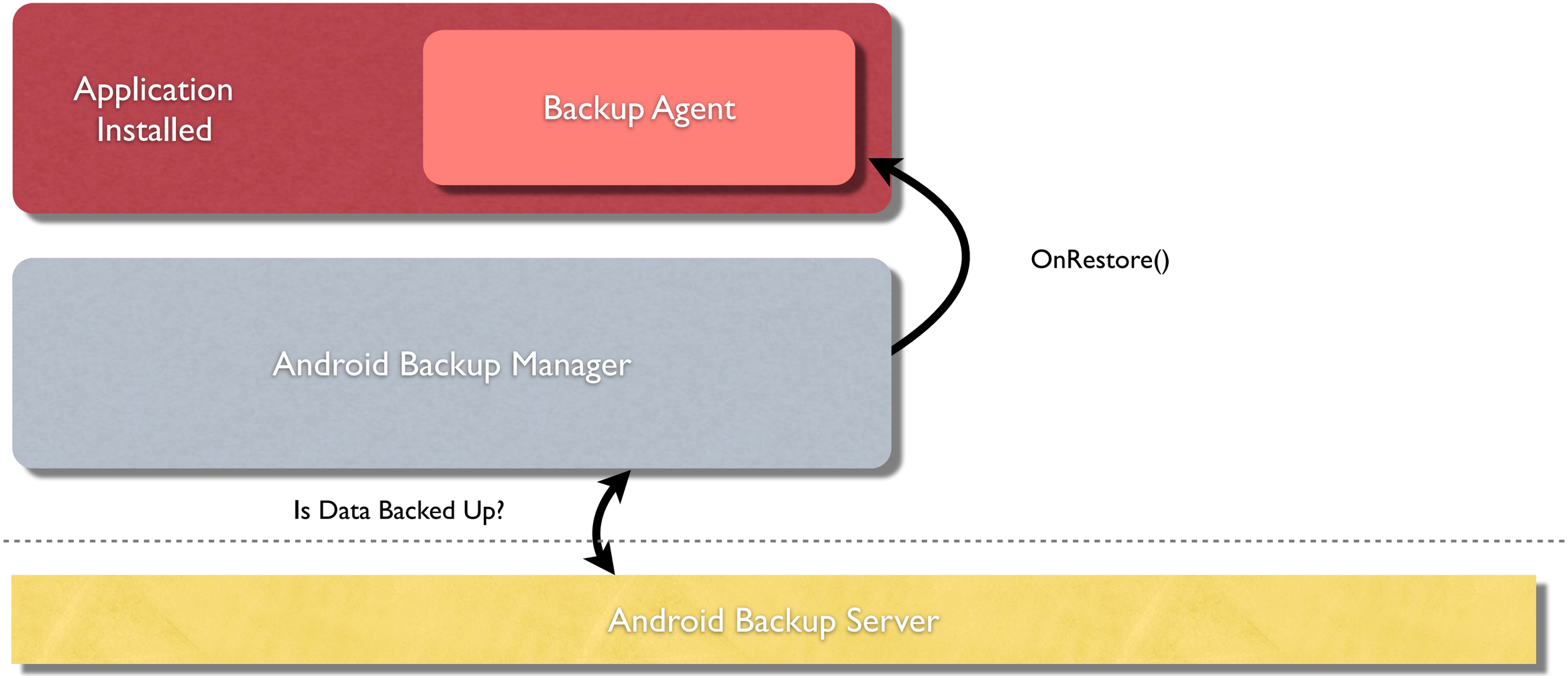



Android Backup Server

# Android Cloud Backup - Restoring



# Android Cloud Backup - Restoring



# Like the Androids in this Presentation? Androidify Yourself

http://www.androidify.com

# Questions?

http://goo.gl/Q8SR7
#Android