Uniwersytet Wrocławski

Rafał Lewandków

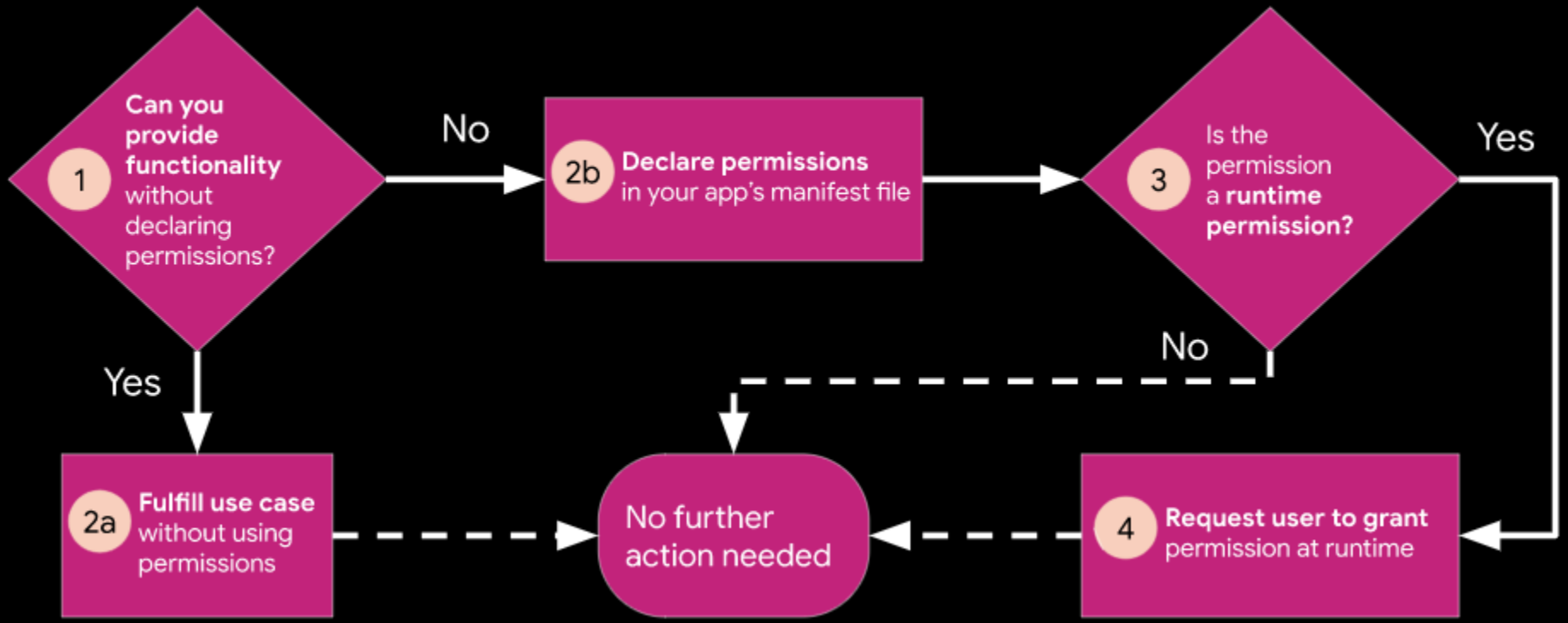
PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 7

- Upoważnienia
- Powiadomienia
- Widget

Uniwersytet Wrocławski

Upoważnienia



Uniwersytet Wrocławski

Upoważnienia

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

Uniwersytet Wrocławski

Upoważnienia

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

```
private val requestPermissionLauncher =
    registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            Log.i("Permission: ", "Granted")
        } else {
            Log.i("Permission: ", "Denied")
        }
    }
```

Uniwersytet Wrocławski

Upoważnienia

- upoważnienie jest już nadane - wyświetlimy tylko prostą wiadomość

```
findViewById<Button>(R.id.cameraButton).setOnClickListener {
    when {ContextCompat.checkSelfPermission(
        this, Manifest.permission.CAMERA) ==
        PackageManager.PERMISSION_GRANTED -> {
            Toast.makeText(this, "Upoważnienie nadane", Toast.LENGTH_SHORT).show()
        }
```

Uniwersytet Wrocławski

Upoważnienia

- upoważnienie jest już nadane - wyświetlimy tylko prostą wiadomość

```
findViewById<Button>(R.id.cameraButton).setOnClickListener {
    when {ContextCompat.checkSelfPermission(
        this, Manifest.permission.CAMERA) ==
        PackageManager.PERMISSION_GRANTED -> {
            Toast.makeText(this, "Upoważnienie nadane", Toast.LENGTH_SHORT).show()
        }
```

- upoważnienie zostało już odrzucone

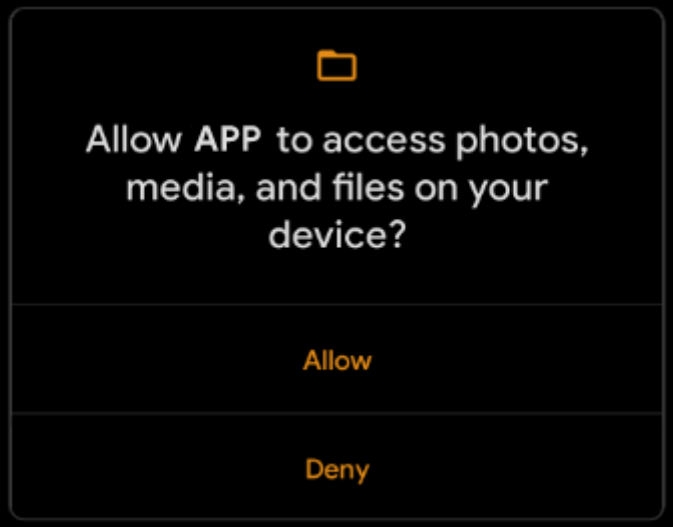
```
ActivityCompat.shouldShowRequestPermissionRationale(
                    this,
                    Manifest.permission.CAMERA) -> {
                        showMessageOKCancel("Wymagane upoważnienie")
                    }
```

Uniwersytet Wrocławski

Upoważnienia

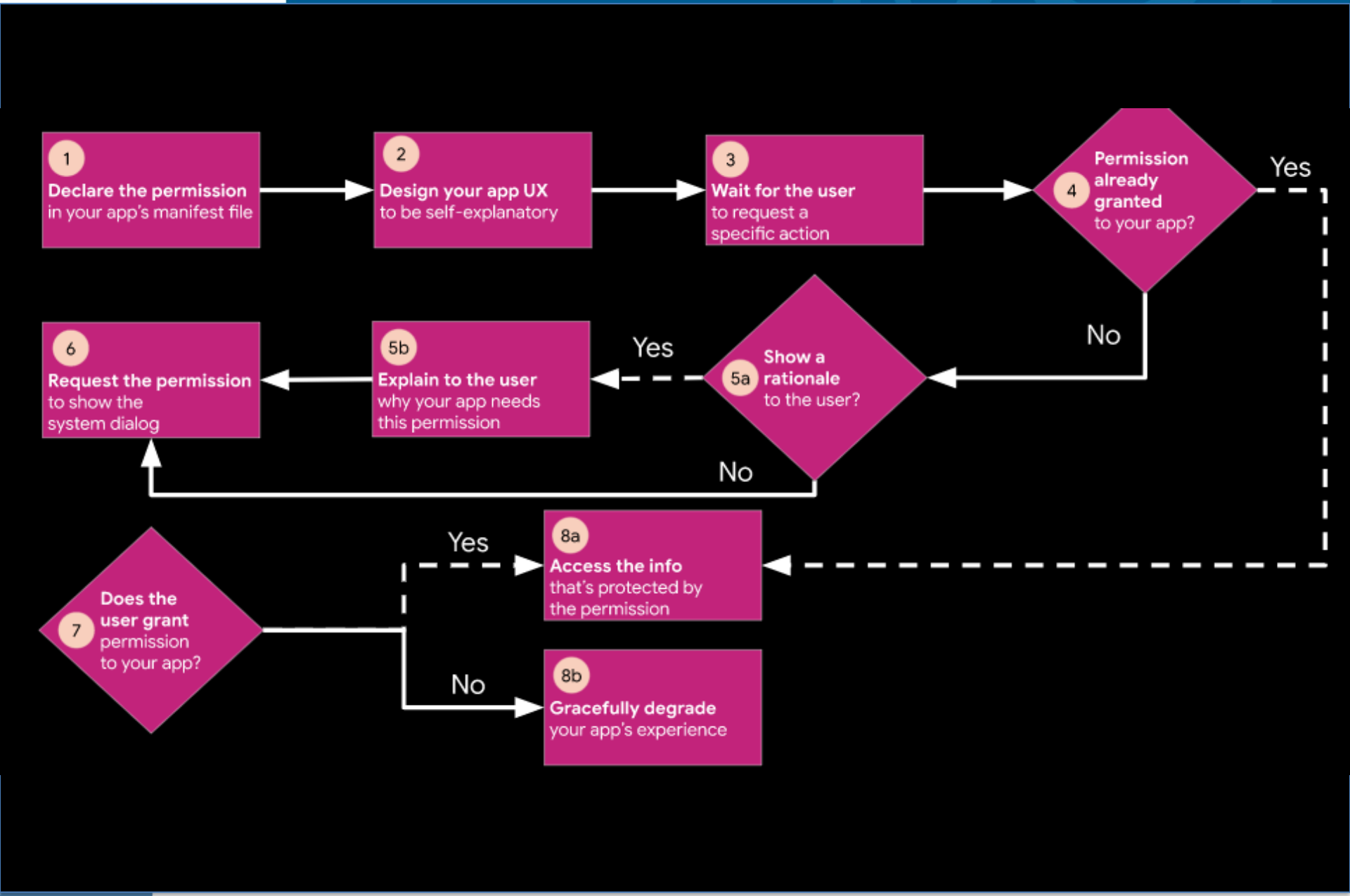
- upoważnienie nie jest nadane

```
else -> {requestPermissionLauncher.launch(Manifest.permission.CAMERA)}
```

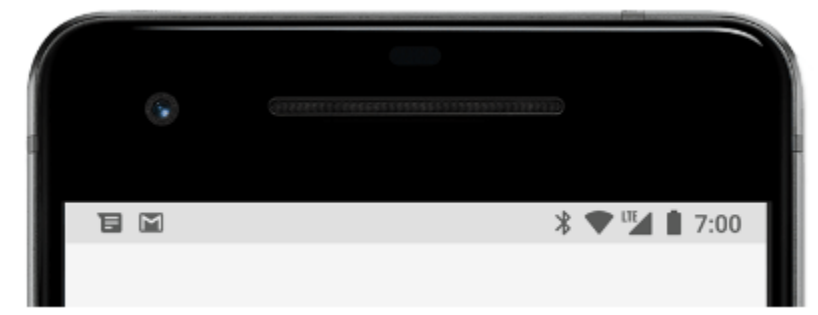


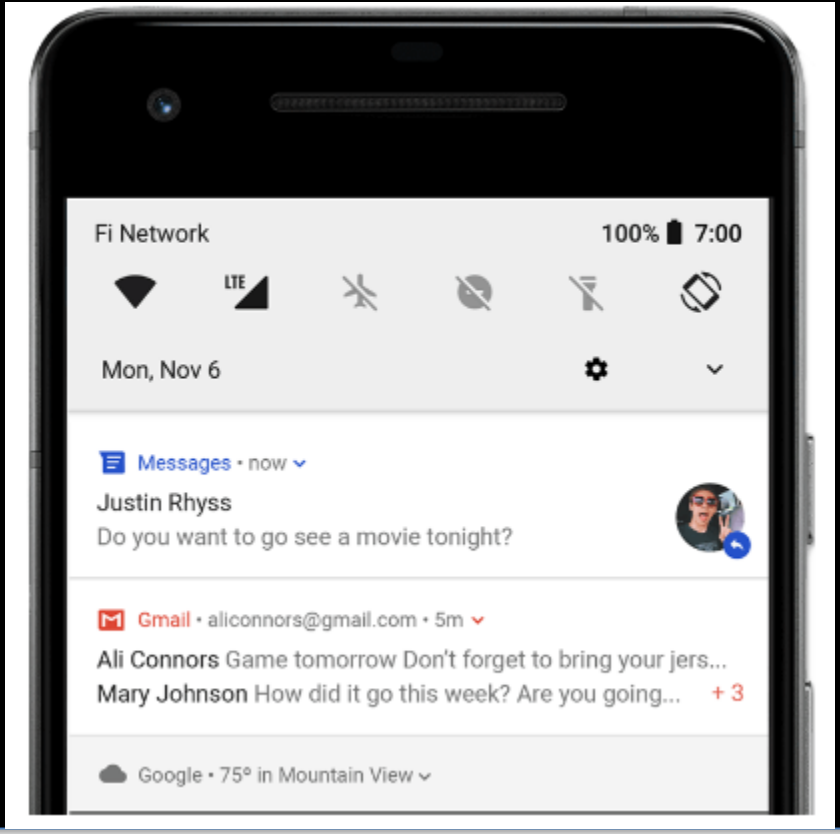
Uniwersytet Wrocławski

Upoważnienia

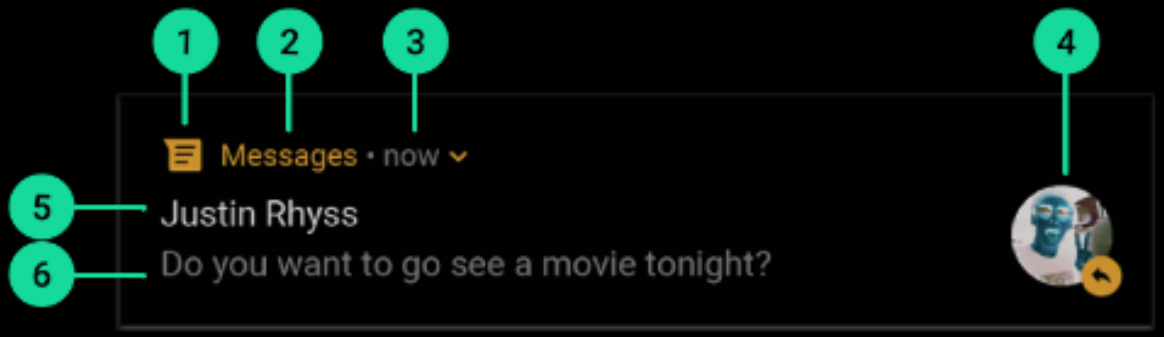


Powiadomienia



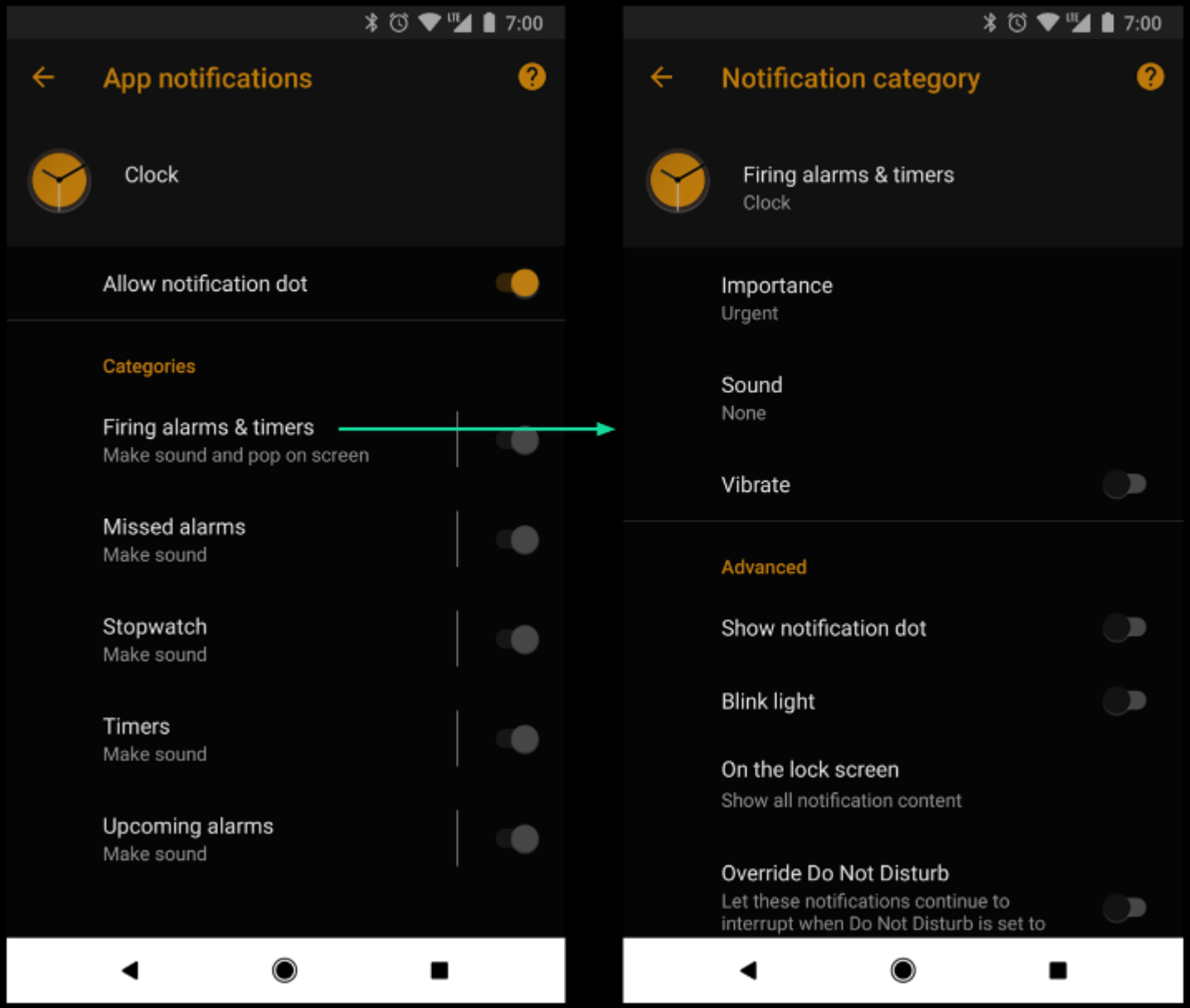


Powiadomienia

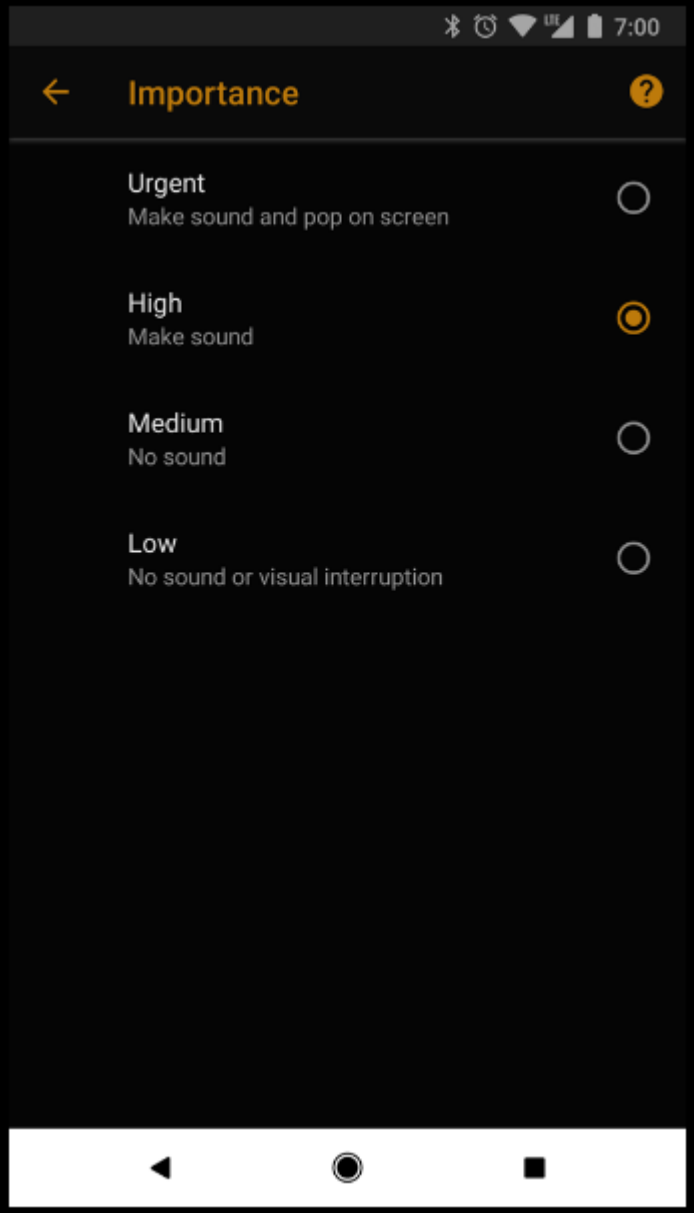


1. Mała ikona – wymagne - setSmallIcon()
2. Nazwa aplikacji – dostarczane przez system
3. Czas – dostarczane przez system
4. Duża ikona – opcjonalne – setLargeIcon()
5. Tytuł powiadomienia – opcjonalne – setContentTitle()
6. Tekst powiadomienia – opcjonalne – setContentText()

Powiadomienia



Powiadomienia



Powiadomienia

- identyfikator kanału
- nazwę kanału
- identyfikator powiadomienia

```
private val channelID = "channel_id"
private val channelName = "channel_name"
private val notificationId = 1
```

- `description` - zawierającą wymagany opis
- `importance` - zawierającą informację o istotności powiadomienia

```
private fun createNotificationChannel() {
    val descriptionText = "powiadomienie"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
```

Uniwersytet Wrocławski

Powiadomienia

```
private fun createNotificationChannel() {
    val descriptionText = "powiadomienie"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel("channel_id", channelName, importance).apply {
        description = descriptionText
    }
```

Uniwersytet Wrocławski

Powiadomienia

```
private fun createNotificationChannel() {
    val descriptionText = "powiadomienie"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel("channel_id", channelName, importance).apply {
        description = descriptionText
    }
```

Kolejnym krokiem będzie dodanie kanału do `NotificationManager`

```
    val notificationManager: NotificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)
```

Uniwersytet Wrocławski

Powiadomienia

```
private fun createNotificationChannel() {
    val descriptionText = "powiadomienie"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel("channel_id", channelName, importance).apply {
        description = descriptionText
    }
    val notificationManager: NotificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)
```

Po naciśnięciu powiadomienia tutaj chcemy przejść na nową instancję `MainActivity` - musimy ustawić odpowiednie flagi

- `FLAG_ACTIVITY_NEW_TASK` - aktywność zostanie ustawiona na tasku aktualnego stosu
- `FLAG_ACTIVITY_CLEAR_TASK` - sprawia że task powiązany z tą aktywnością zostanie wyczyszczony przed startem aktywności

```
val intent = Intent(this, MainActivity::class.java).apply {
        flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    }
```

Uniwersytet Wrocławski

Powiadomienia

```
private fun createNotificationChannel() {
    val descriptionText = "powiadomienie"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel("channel_id", channelName, importance).apply {
        description = descriptionText
    }
    val notificationManager: NotificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)
    val intent = Intent(this, MainActivity::class.java).apply {
        flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    }
```

Kolejnym krokiem będzie utworzenie `PendingIntent` - zawiera opis intentu i wykonywanej przez niego akcji. Przyjmuje cztery argumenty

- `context`
- `requestCode` - unikalny kod dla tego intentu - tutaj ustawimy 0
- `Intent`
- `flags` - tutaj wykorzystamy `PendingIntent.FLAG_IMMUTABLE` - opisuje `PendingIntent` i jest wykorzystywana do jego identyfikacji

```
val pendingIntent: PendingIntent =
        PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_IMMUTABLE)
```

Uniwersytet Wrocławski

Powiadomienia

```
private fun createNotificationChannel() {
    val descriptionText = "powiadomienie"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
    val channel = NotificationChannel("channel_id", channelName, importance).apply {
        description = descriptionText
    }
    val notificationManager: NotificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)
    val intent = Intent(this, MainActivity::class.java).apply {
        flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
    }
    val pendingIntent: PendingIntent =
        PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_IMMUTABLE)
```

Uniwersytet Wrocławski

Powiadomienia

Następnie tworzymy samo powiadomienie za pomocą klasy `NotificationCompat.Builder`

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, channelID)
            .setSmallIcon(R.drawable.ic_baseline_notifications_24)
            .setContentTitle("Powiadomienie")
            .setContentText("Treść powiadomienia")
            .setStyle(
                    new NotificationCompat.BigTextStyle()
                            .bigText("Dalszy tekst powiadomienia")
            )
            .setContentIntent(pendingIntent)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Uniwersytet Wrocławski

Powiadomienia

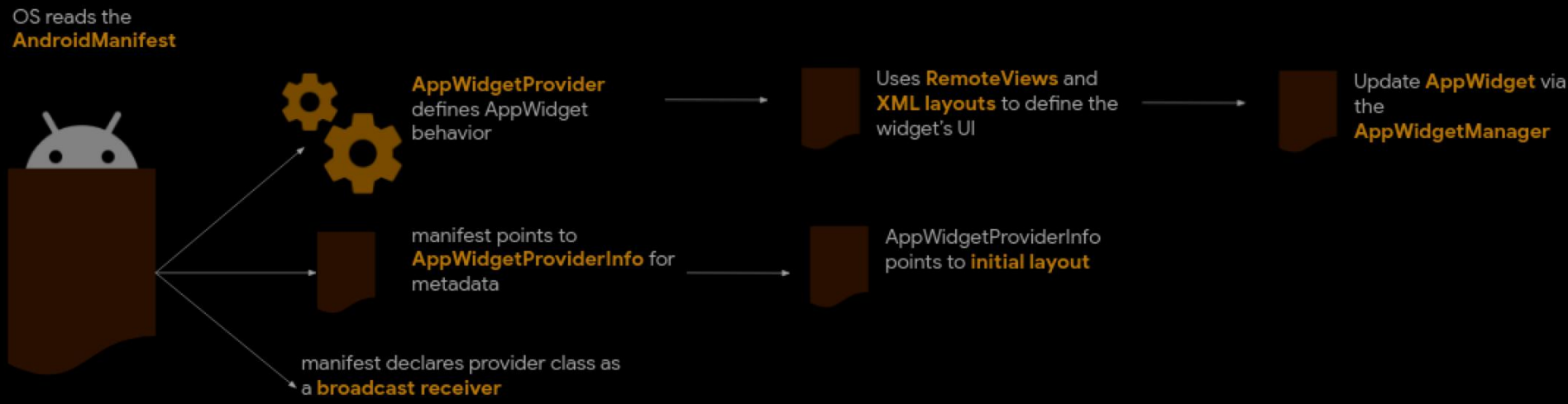
Następnie tworzymy samo powiadomienie za pomocą klasy `NotificationCompat.Builder`

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, channelID)
            .setSmallIcon(R.drawable.ic_baseline_notifications_24)
            .setContentTitle("Powiadomienie")
            .setContentText("Treść powiadomienia")
            .setStyle(
                    new NotificationCompat.BigTextStyle()
                            .bigText("Dalszy tekst powiadomienia")
            )
            .setContentIntent(pendingIntent)
            .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Aby powiadomienie pojawiło się na pasku statusu musimy wywołać metodę `notify` klasy `NotificationManagerCompat`

```
NotificationManagerCompat notificationCompat =
        NotificationManagerCompat.from(this);
    notificationCompat.notify(notificationId, builder.build());
```

Widget


OS reads the
AndroidManifest
AppWidgetProvider
defines AppWidget
behavior
Uses RemoteViews and
XML layouts to define the
widget's UI
Update AppWidget via
the
AppWidgetManager
manifest points to
AppWidgetProviderInfo for
metadata
AppWidgetProviderInfo
points to initial layout
manifest declares provider class as
a broadcast receiver

Uniwersytet Wrocławski

Widget

```
class WidgetProvider : AppWidgetProvider() {}
```

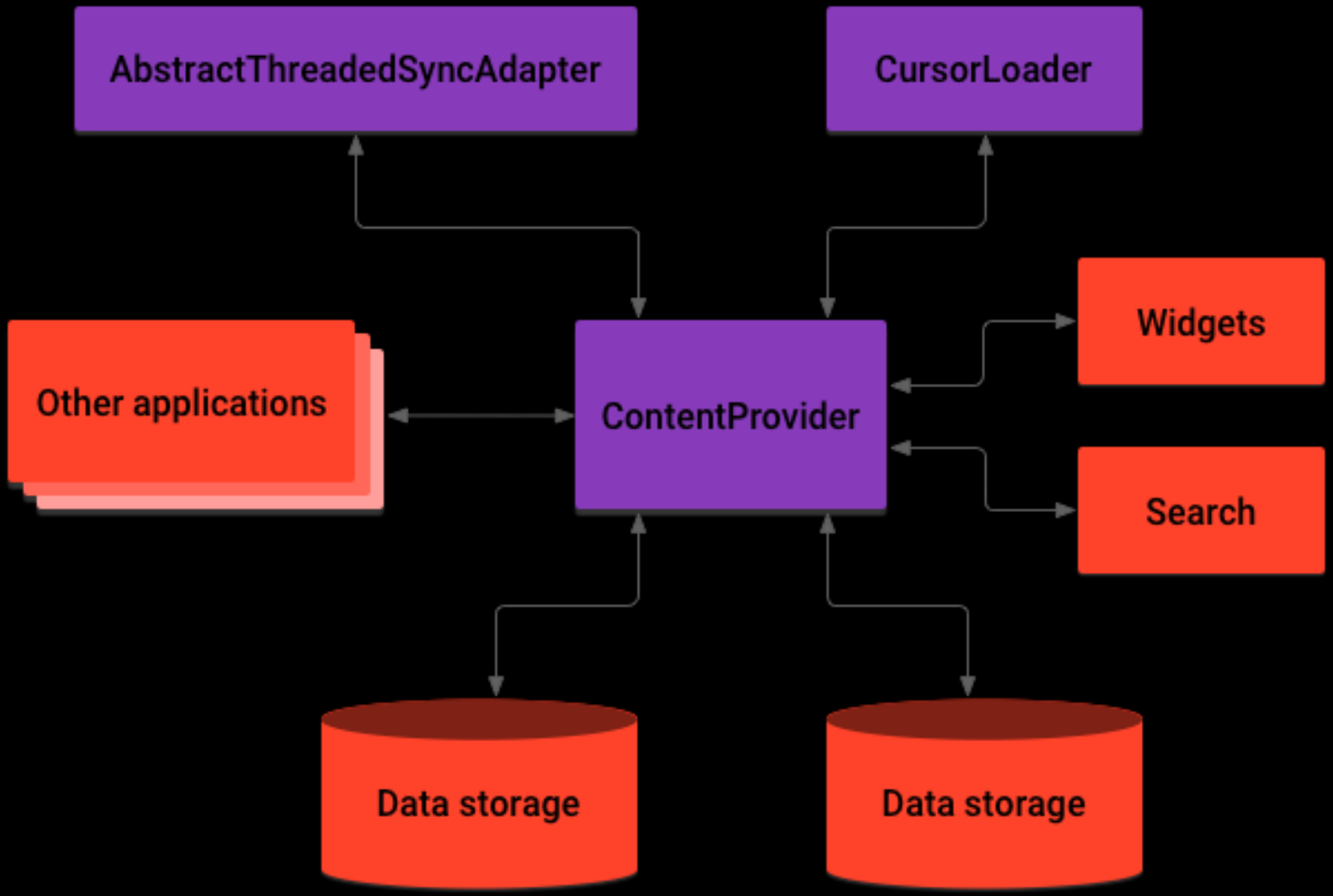
Uniwersytet Wrocławski

Widget

```
class WidgetProvider : AppWidgetProvider() {}
```

- **onUpdate** – wywoływana przy aktualizacji – aktualizacja odbywa się co określony czas
- **onAppWidgetOptionsChanged** – wywoływana przy umieszczeniu widgetu na ekranie i za każdym razem przy zmianie rozmiaru
- **onDeleted(Context, int[])** - wywoływana przy usuwaniu widgetu
- **onEnabled(Context), onDisabled(Context)** – wywoływana gdy instancja widgetu jest umieszczona na ekranie po raz pierwszy/usunięta
- **onReceive(Context, Intent)** – wywoływana przy każdym broadcascie i przed każdym wywołaniem metod zwrotnych

ContentProvider



Udostępnia dane zewnętrznym aplikacjom

Uniwersytet Wrocławski

AppWidgetProvider

```
class WidgetProvider : AppWidgetProvider() {}
```

```
override fun onUpdate(
    context: Context?,
    appWidgetManager: AppWidgetManager?,
    appWidgetIds: IntArray?
) {
    super.onUpdate(context, appWidgetManager, appWidgetIds)
}
```

- **appWidgetManager –** widget działa na innym procesie – umożliwia komunikację
- **appWidgetIds –** unikalne identyfikatory wszystkich instancji widgetu

Uniwersytet Wrocławski

AppWidgetProvider

```
override fun onUpdate(
    context: Context?,
    appWidgetManager: AppWidgetManager?,
    appWidgetIds: IntArray?
) {
    super.onUpdate(context, appWidgetManager, appWidgetIds)
    for (appWidgetId in appWidgetIds!!) {
        val intent = Intent(context, MainActivity::class.java)
        val pendingIntent = PendingIntent.getActivity(
            context,
            0,
            intent,
            PendingIntent.FLAG_IMMUTABLE
        )

        val views = RemoteViews(
            context.packageName,
            R.layout.app_widget_layout
        )
        views.setOnClickPendingIntent(R.id.widgetButton, pendingIntent)
        appWidgetManager!!.updateAppWidget(appWidgetId, views)
    }
}
```

Umożliwia pokazanie layoutu na innym procesie

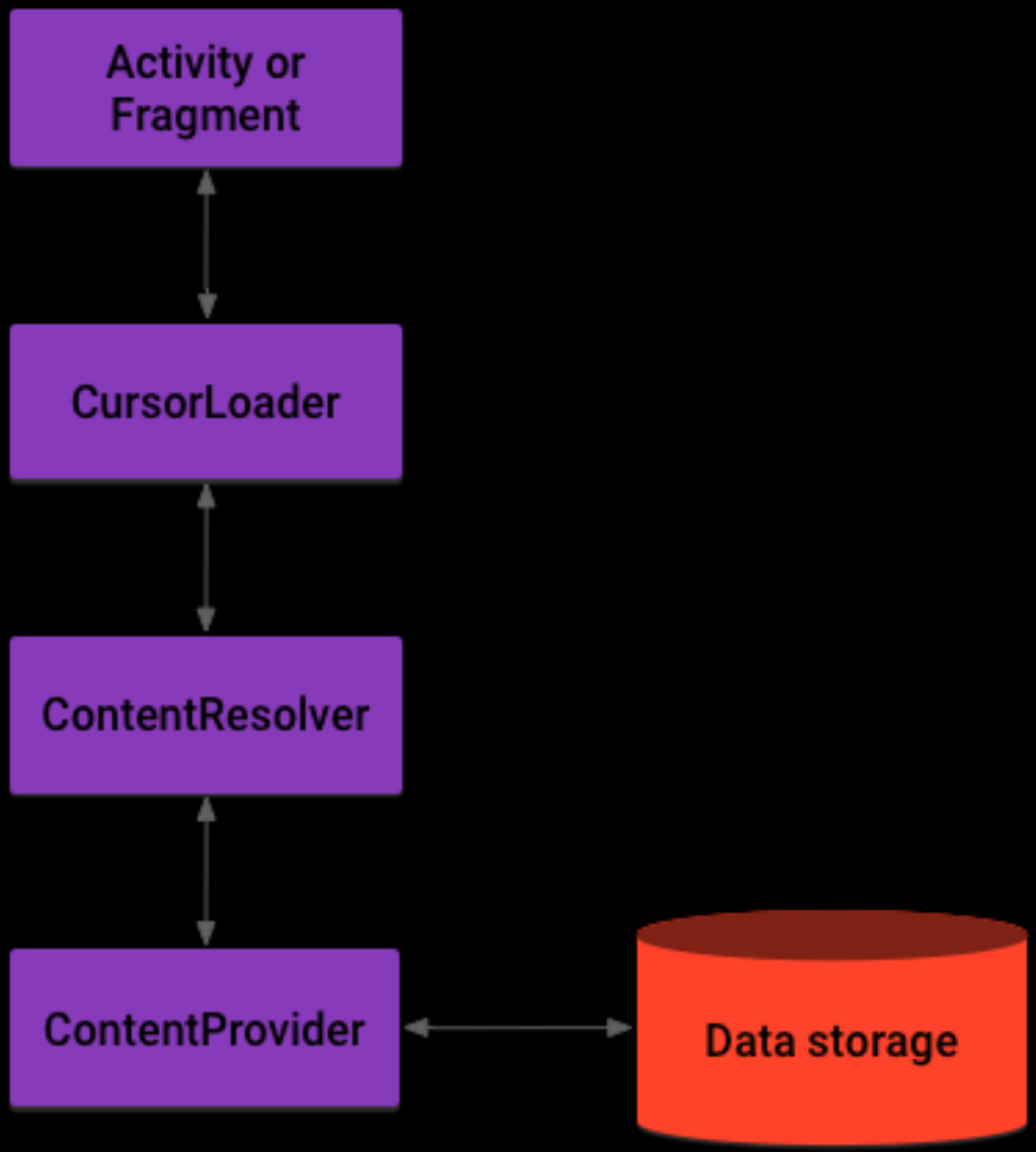
Uniwersytet Wrocławski

AppWidgetProviderInfo

```
xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
   android:initialLayout="@layout/app_widget_layout"
    android:minHeight="40dp"
    android:minWidth="110dp"
    android:minResizeWidth="40dp"
    android:resizeMode="vertical|horizontal"
    android:updatePeriodMillis="36000000"
    android:widgetCategory="home_screen">

appwidget-provider>
```

ContentProvider



BrodcastReceiver

```
<receiver android:name=".WidgetProvider"
    android:exported="false">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE"/>
    intent-filter>
    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/app_widget_info"/>
receiver>
```