Platform-Based Development: Android Programming – Permissions and Notifications

BS UNI studies, Spring 2018/2019

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

Android Security

- Applications are sandboxed
  - Each app has its user ID and group ID that is unique for the device
  - Allocated resources for the user ID
- Only the most basic functionalities are available to an application, other functionalities have to be explicitly asked for

Permissions

- An app requests permissions to access:
  - User data (e.g. contacts)
  - Some cost-sensitive APIs (e.g. send SMS)
  - Some system resources (e.g. camera)
- Permissions are declared in AndroidManifest.xml
  - Predefined Strings from Manifest.permission
  - <user-permission> in Manifest indicates a request:

```
<uses-permission android:name=
“android.permission.ACCESS_FINE_LOCATION”/>
```

Permissions Types

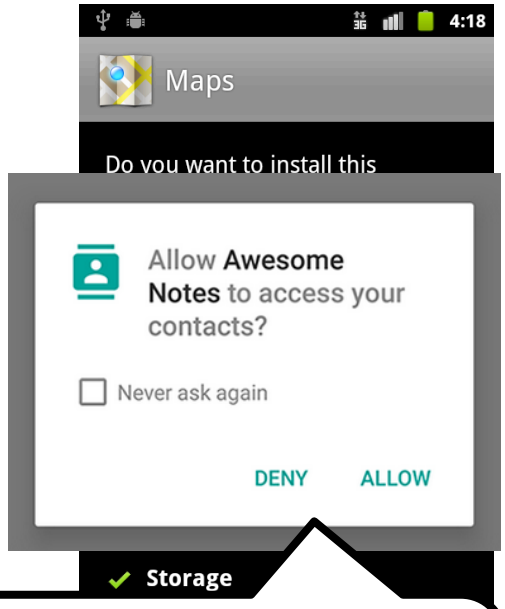
- Normal permissions
  - FLASHLIGHT, VIBRATE, BLUETOOTH, etc.
  - The OS will automatically grant these permissions to your app

  http://developer.android.com/guide/topics/security/normal-permissions.html
- Dangerous permissions
  - READ_CONTACTS, SEND_SMS, ACCESS_COARSE_LOCATION, WRITE_EXTERNAL_CONTACTS, etc.
  - The OS will explicitly ask the user to grant access for dangerous permissions

Handling Permissions

- Prior to Marshmallow (API 23):
  - Users must accept permissions before the app starts
- Marshmallow (API 23):
  - Permissions must be granted at the runtime, when the functionality is about to be used
  - Ask straight away, educate up front, ask in context, educate in context

See a short explanation by Google devs:
www.youtube.com/watch?v=iZqDdvhTZj0



Note that users can disable permissions at a later point

Custom Permissions

- Define your own permission if the app performs a privileged/dangerous function, and you don't want just any app to be able to launch yours
- In AndroidManifest.XML under <permission>
- Example:

```
<permission android:name="com.testpackage.mypermission"
            android:label="my_permission"
            android:protectionLevel="dangerous"/>
```

- Any app that wants to launch yours must request "com.testpackage.mypermission"

Component Permissions

- Permissions can be defined for individual components restricting which other component can call them
  - Activities
    - Which components can call startActivity()
  - Services
    - Which components can start or bind to a Service
  - BroadcastReceivers
    - Which components can receive and send broadcasts
  - ContentProviders
    - Which components can read and write content provider data

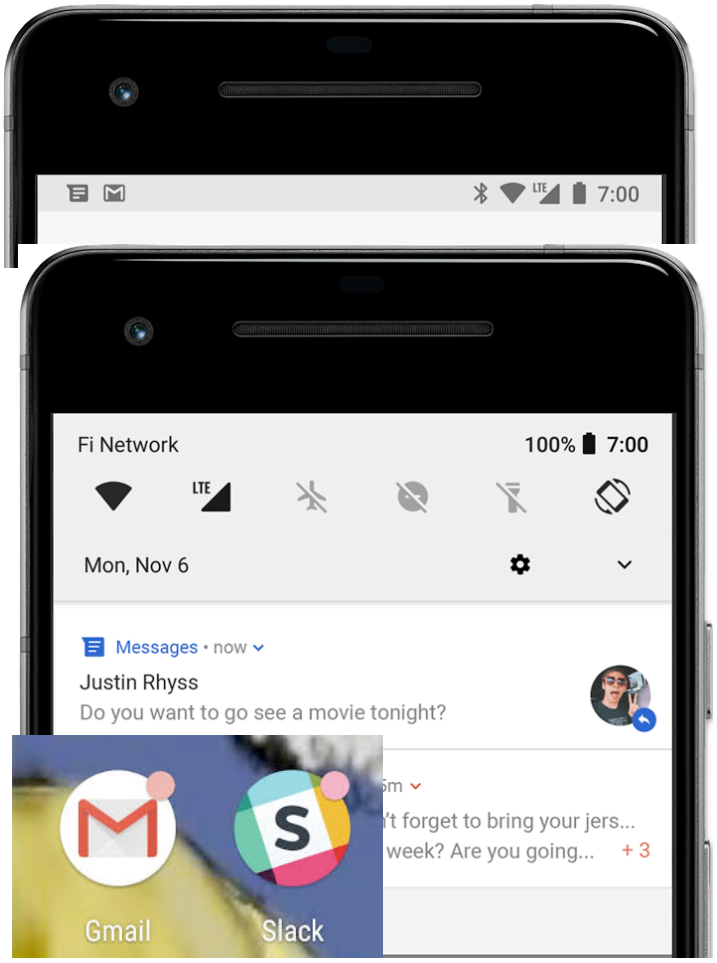
Permissions Design

- Do not request permissions unless you really need them
  - E.g. do you need to write data to an external storage (file) or can you keep the information in SharedPrefs?
- Show immediate benefit of granting a permission
  - E.g. your ToDo list app requires location info – show the user how she can make location-based reminders
- Use Intents to call other apps in case you don't need to handle the functionality within your app:
  - E.g. call a camera app, rather than requesting the camera permission for your app

Notifications

Notifications

- Allows the app to initiate contact with the user
- Shown in the Status Bar (Notification Bar)
  - Custom icons
  - Alerts: sound, vibration, LED flashing
  - Notification drawer
  - On a locked screen
  - As a “badge”



https://developer.android.com/guide/topics/ui/notifiers/notifications

Notification Implementation

- Manage notifications via NotificationManager (or better yet via NotificationManagerCompat)
  - Post notifications with notify()
  - Clear notifications with cancel() and cancelAll()
- Build a notification via Notification.Builder

```
notification = new Notification.Builder(this)
        .setContentTitle(notificationTitle)
        .setContentText(notificationText)
       .setSmallIcon(R.drawable.icon)
        .setContentIntent(pendingIntent).build();
```

- Set a notification channel (in API 26 and above)

Notification Design

- Notifications are the only way for your app to initiate interaction with the user
  - Can be a core functionality of your app (e.g. Facebook messenger)
  - Use to show new/important information to the user
  - Use to increase user engagement
  - Use to get feedback from the user
- Do not overuse and minimise interruptions

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it"
Mark Weiser, 1991

Platform-Based Development: Background Processing

BS UNI studies, Spring 2018/2019

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

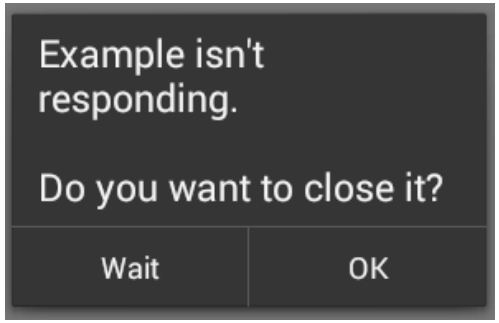
Concurrency in Android

- Java Threads
  - The most general method
- Service
  - Runs in the background, however, by default on the same thread as the UI
- IntentService
  - Background work on a separate thread, to contact the UI use local broadcast
- AsyncTask
  - Background work on a separate thread, but with a tight integration with the UI

Threads

- A UI (Main) Thread is created and started when an application is launched
  - Listens for events on UI components
  - Loops infinitely
- Your code (by default) runs on the UI Thread
- Intensive work (database access, networking) can prevent the UI thread from processing UI interaction tasks

There are no separate threads for components



Threads

- Instead, run heavy/slow operations in background threads
- Background thread processing supported by:
  - Threads + Handlers
  - IntentService
  - AsyncTask
  - Thread + Service
- Different abstractions for different goals, e.g.:
  - A music player that runs in the background
  - An online social network post button

Thread and Handlers

- Java Threads + a Handler that enables communication among the threads
- A straightforward solution:
  - Create a worker Thread
  - Put an infinite loop in it and listen for new tasks
  - De-queue the tasks, for each task:
    - Execute
    - Report results back to the UI Thread via a Handler
  - Break the loop to kill the thread

Thread and Handlers Example

Looper, Message Queue, Handler

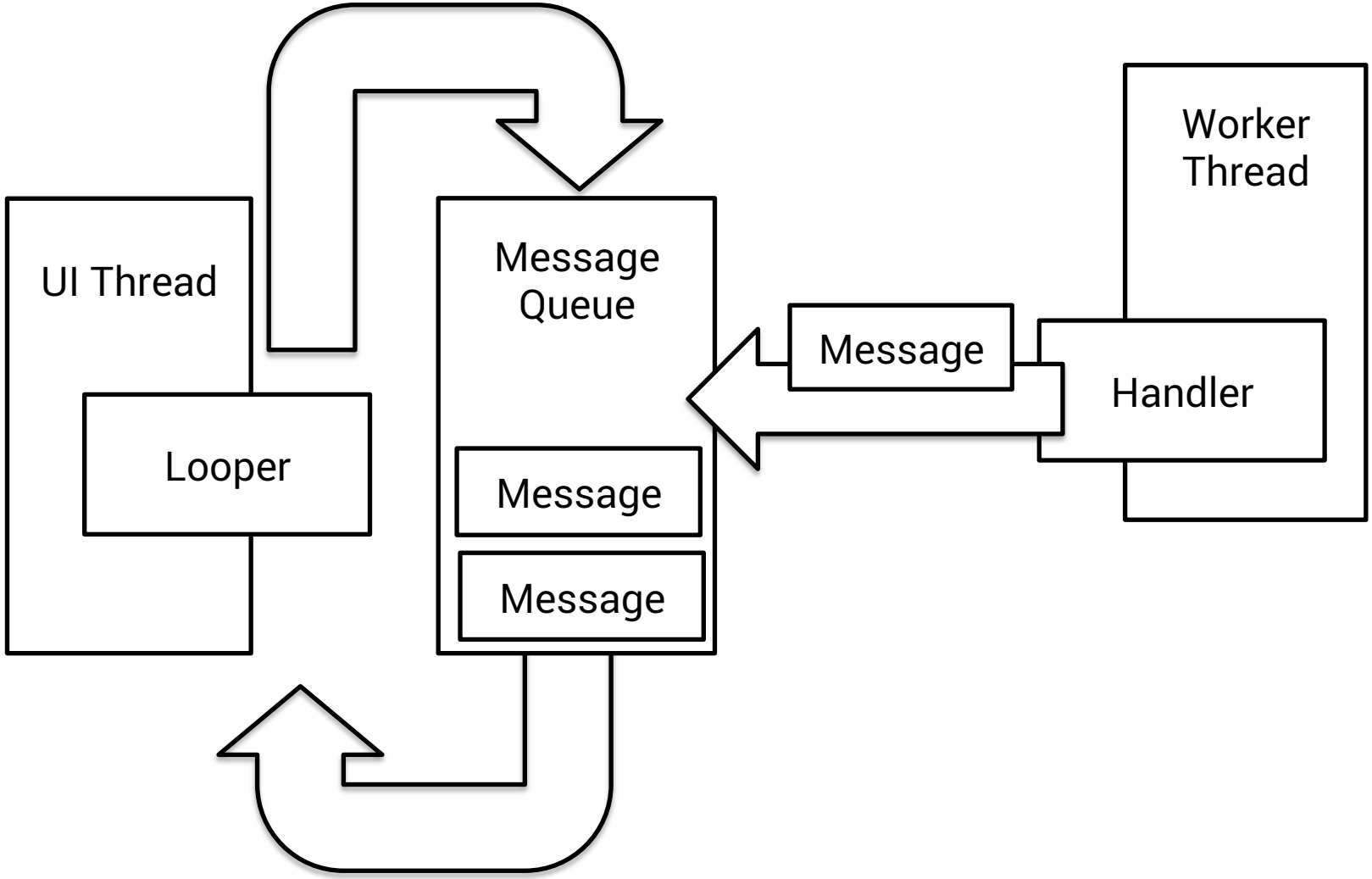
- Looper keeps the Thread alive in an infinite loop
  - Automatically created for the UI Thread
  - For custom threads, create it yourself or use HandlerThread
    - `Looper.prepare();`
    - `Looper.loop();`
    - `Looper.quit();`
- MessageQueue holds Messages/Runnables for a Thread
  - Message – for passing data to a thread
  - Runnable – a task that is executed when the thread is free (or after a predefined delay)

Looper, Message Queue, Handler

- Handler
  - Associated with a particular Thread (e.g. UI Thread)
  - Allows you to send Messages/Runnables to the MessageQueue and process them
  - Post a Message/Runnable to be handled immediately via post(Runnable r) method or after a certain delay via postDelayed(Runnable r, int msDelay)

```
new Handler(Looper.getMainLooper())
    .post(new Runnable() {
          @Override
          public void run() {
            // this will run in the main thread
          }
});
```

Thread and Handlers



HandlerThread

- A Thread that has a Looper
- Example use:
  - Instantiate a HandlerThread
  - Attach a Handler to your thread

```
HandlerThread handlerThread =
      new HandlerThread("MyHandlerThread");
handlerThread.start();
Looper looper = handlerThread.getLooper();
Handler handler = new Handler(looper);
```

Call .quit() to shut the thread down

HandlerThread Example

Services

- Activities run on the UI (main) thread and have a UI attached (layout)
  - Processing-heavy functions on the main thread impact the responsiveness
- Services can run on either the Main or separate threads and do not have a UI attached
  - Run outside UI, for long-running operations
- Services are often more convenient than custom Threads for tasks than need to be “independent” and run even when the Activity is destroyed

Background and Foreground Service

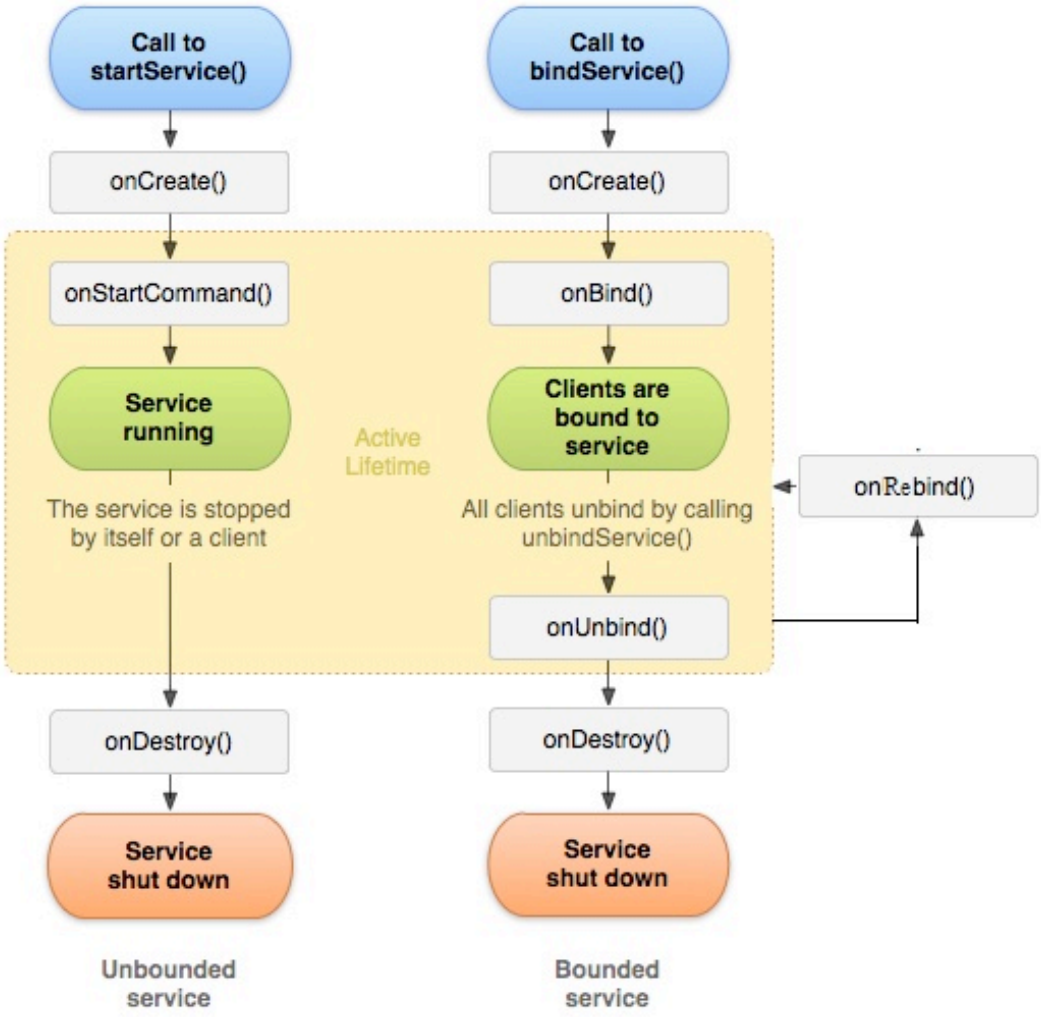
- Background Service
  - For actions that do not have to be noticed by the user (e.g. sensing a user's physical activity)
- Foreground Service
  - For actions that the user needs be aware of and should the control of (e.g. a music player app)
  - A foreground service must show a notification in the notification bar

Starting/Stopping a Service

- Services can be created:
  - Explicitly using Context.startService()
  - Implicitly, if not already running, when a client requests connection to a Service via Context.bindService()
- Services can be stopped:
  - From within the Service with stopSelf()
  - From another component with Context.stopService()

Services

- Multiple startService calls do not nest – you only have one service; however, onStartCommand() will be called repeatedly
- Service will be stopped only once with Context.stopService() or stopSelf()



Services – Bound

- Bound Services – like servers in a client-server paradigm
- Services started through binding, do not call onStartCommand()
- Return IBinder object from onBind(Intent) so that connected clients can call the Service
- The service remains running as long as the connection is established

Broadcast

- Messages sent from other components of your app, other apps or from the Android system
- Messages are wrapped in Intents

```
Intent intent = new Intent();
intent.setAction(ACTION);
intent.putExtra(STOP_SERVICE_BROADCAST_KEY, RQS_STOP_SERVICE);
sendBroadcast(intent);
```

- Send broadcasts
  - System sends certain broadcasts when an event happens, e.g. ACTION_BOOT_COMPLETED
  - Send custom broadcasts via sendBroadcast()

Broadcast

- Broadcasts are captured in an app/component if a BroadcastReceiver is registered in the code:
  - Create a BroadcastReceiver and impl. onReceive()

```
public class NotifyServiceReceiver extends BroadcastReceiver{

    @Override
    public void onReceive(Context arg0, Intent arg1) {
                …
        }
    }
}
```

  - Register for receiving certain kinds of Intents

```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(ACTION);
registerReceiver(notifyServiceReceiver, intentFilter);
```

Broadcast

- Broadcasts are captured in an app/component if a BroadcastReceiver is registered in AndroidManifest.XML and onReceive() is implemented in the code:

```
<receiver android:name=".MyBroadcastReceiver" android:exported="true">
 <intent-filter>
   <action android:name="android.intent.action.BOOT_COMPLETED"/>
   <action android:name="android.intent.action.INPUT_METHOD_CHANGED"/>
 </intent-filter>
</receiver>
```

```
public class MyBroadcastReceiver extends BroadcastReceiver {
     @Override
     public void onReceive(Context context, Intent intent) {
….
```

BroadcastReceiver

- Receive events announced by other components
- Events announced via Intents
  - Not the same Intent as the one starting an Activity: this one remains in the background
- Events can be announced within your app or publicly to every app on the phone
  - Announce via sendBroadcast()
- Events captured if the receiver is registered:
  - onReceiverRegistered() and then onReceive()

Service, Notification, BroadcastReceiver Example

Note on Foreground Services

- Likely to see increased use
  - Google aims to minimize background processing
  - FS for immediate guaranteed tasks, such as mobile payments, apps for unlocking garages, etc.
- In API 26 and above
  - Starting a foreground service should be done with:
  - startForegroundService() – a promise that it will go to foreground and show a notification
    followed by
  - startForeground() – the actual notification is shown

Usually in the Service's onCreate

IntentService

- A Service that
  - Runs on a separate thread
  - Queues up requests and processes them one by one
- Suitable for long running one-off tasks when we don't want to affect the UI responsiveness
- IntentService survives Activity lifecycle changes
- Called using explicit Intent
- Starts on demand, stops when it runs out of work

IntentService

- Define in AndroidManifest.XML

```
<service
    android:name=".FetchAddressIntentService"
    android:exported="false"/>
```

- Extend the class in your Java code

```
public class FetchAddressIntentService extends
IntentService {
```

Invoking IntentService

- Create an explicit Intent for your IntentService
- Use startService() to start the IntentService
- Add additional data if needed with the extra field

Handling Results – from IntentService to Activity (1)

- BroadcastReceiver in your Activity
  - Subclass BroadcastReceiver, implement onReceive
  - Register the receiver for a particular action for times when you would like to handle IntentService results (usually when your Activity is in the foreground)
- Broadcast from your IntentService
  - sendBroadcast() from your IS using the same Intent action as the above

Handling Results – from IntentService to Activity (2)

- ResultReceiver in your Activity
  - Subclass ResultReceiver, implement onReceiveResult

```
class AddressResultReceiver extends ResultReceiver {
    public AddressResultReceiver(Handler handler) {
        super(handler);
    }

    @Override
    protected void onReceiveResult(int resultCode,
                                   Bundle resultData) {…}
```

  - Pass ResultReceiver through Intent when starting IS

```
Intent intent = new Intent(this, FetchAddressIntentService.class);
intent.putExtra(Constants.RECEIVER, mResultReceiver);
intent.putExtra(Constants.LOCATION_DATA_EXTRA, mLastLocation);
startService(intent);
```

Handling Results – from IntentService to Activity (2)

- Set ResultReceiver result
  - IntentService sends results to ResultReceiver in a Bundle with send() method

```
Bundle bundle = new Bundle();
bundle.putString(Constants.RESULT_DATA_KEY, message);
mReceiver.send(resultCode, bundle);
```

- Example
  - Display location address

    http://developer.android.com/training/location/display-address.html

IntentService Example

University *of Ljubljana*
Faculty *of Computer and Information Science*

Based on:
https://www.vogella.com/tutorials/AndroidServices/article.html

AsyncTask

- For short, more interactive tasks
- Runs on a separate worker thread, but keeps a link with the main UI thread via:
  - onPreExecute
  - onProgressUpdate
  - onPostExecute

AsyncTask <?, ?, ?>
"?" param types for input, progress, output

- Define what we want to do in the background in:
  - doInBackground
- Start with YourTask().execute()

https://github.com/googlesamples/android-play-location/tree/master/ActivityRecognition

AsyncTask Example

```
private class PostTask extends AsyncTask<String, Integer, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        ProgressBar bar=(ProgressBar)findViewById(R.id.progressBar);
        bar.setVisibility(View.VISIBLE);
        bar.setProgress(0);
    }

 @Override
 protected String doInBackground(String... params) {
     String url=params[0];
     for (int i = 0; i <= 10; i += 1) {
         try {
             Thread.sleep(1000);
         } catch (InterruptedException e) {
             e.printStackTrace();
         }
         publishProgress(i);
     }
     return "All Done!";
 }
```

Just before the task starts

This is done in the background, and the status is communicated via publishProgress()

AsyncTask Example

```
@Override
protected void onProgressUpdate(Integer... values) {
    super.onProgressUpdate(values);
    ProgressBar bar=(ProgressBar)findViewById(R.id.progressBar);
    bar.setVisibility(View.VISIBLE);
    bar.setProgress(values[0]);
}
```

Connects with the UI thread

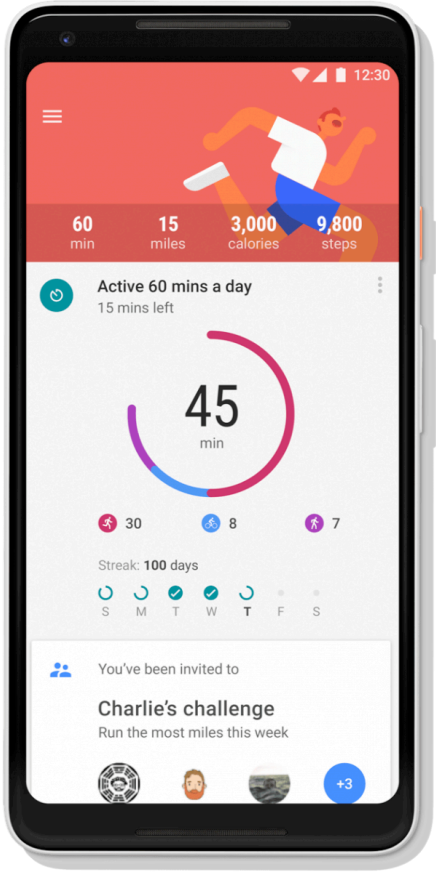
```
@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);
    ProgressBar bar=(ProgressBar)findViewById(R.id.progressBar);
    bar.setVisibility(View.GONE);
    TextView text = (TextView) findViewById(R.id.status);
    text.setText(R.string.after);
}
```

Immediately after the task is finished

AsyncTask Example

Periodic/Occasional Task Scheduling

- Numerous situations in which we require occasional processing:
  - Tracking physical activity throughout a day – e.g. Google Fit
    - Sampling sensors periodically
  - Synchronizing data with the server
    - Send data periodically, when there is WiFi connectivity
  - Reminding a user when in a particular location
    - Geofenced reminder



Periodic/Occasional Task Scheduling

- Limited battery capacity is the main issue in mobile computing
- Long and frequent background processing is the main reason for inefficient energy use:
  - Users are often unaware of background processes and their intentions, cannot easily shut them down
  - Processes consume computational and memory resources
  - Processes prevent a device from going to a low-power mode

Periodic/Occasional Task Scheduling

- Android's general direction is towards limited and controlled background processing
- In the old days (API<19):
  - schedule a periodic job to be executed every 15 mins
- Today:
  - schedule a job and Android will aggregate jobs of all apps, schedule them for a particular time slot (that you have no control off), if the app is used only rarely it might have to wait for 24 hours, and forget about getting location updates more than a few times per hour (if in background), getting notified when there is connectivity, etc.

Tools for Periodic/Occasional Task Scheduling

- Wake lock
- Foreground Service
- AlarmManager
- WorkManager (JobScheduler++)
- DownloadManager
- SyncAdapter

Wake Lock

- App prevents the phone from going to a low-power sleep mode
- Needs a special permission

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

- Acquire a wake lock

```
PowerManager powerManager = (PowerManager)
                            getSystemService(POWER_SERVICE);
WakeLock wakeLock = powerManager
                    .newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
                                 "MyApp::MyWakelockTag");
wakeLock.acquire();
```

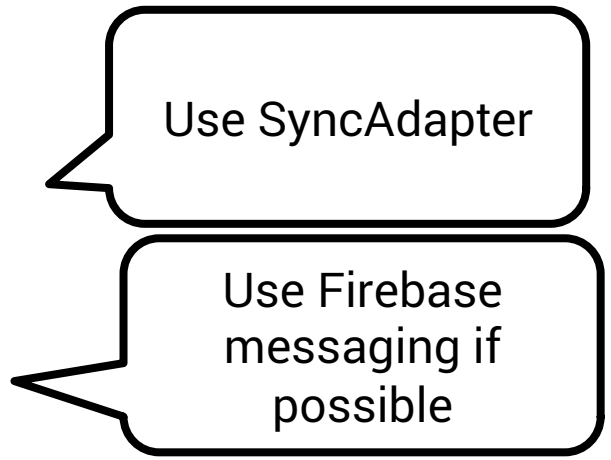
- Release: `wakelock.release()`

This does not prevent the screen from going dark!

(use FLAG_KEEP_SCREEN_ON)

AlarmManager

- Running periodic operations at specified times or with a specified time interval
- Use when you need your tasks done at (almost) exact times between them
- Do not use for:
  - Periodic backups to the server
  - Checking for new notifications/ messages from the server



AlarmManager

- Alarm types (exactness):
  - Inexact – Android will decide how to group alarms coming from multiple apps in order to optimize energy use
  - Exact – Your alarm will be executed at the prescribed time, unless the device is “sleeping”
  - Exact while idle - Your alarm will be executed at the prescribed time, even if the device is “sleeping”
- Alarm types (clock):
  - RTC – real time clock
  - ELAPSED_REALTIME – time since booted

AlarmManager

- Using AlarmManager
  - Create a BroadcastReceiver that manages the task you wish to perform when the alarm is ready
  - Set alarm
    - Define the type (exact/inexact, one off/repeating, RTC/ELAPSED)
    - Define the starting time
    - Define the repeating interval (optionally)
    - Supply Intent that starts the above BroadcastReceiver
  - Alarms can be cancelled

AlarmManager

- Restoring alarms when the device is rebooted
  - Acquire the necessary permission

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

  - Create a receiver

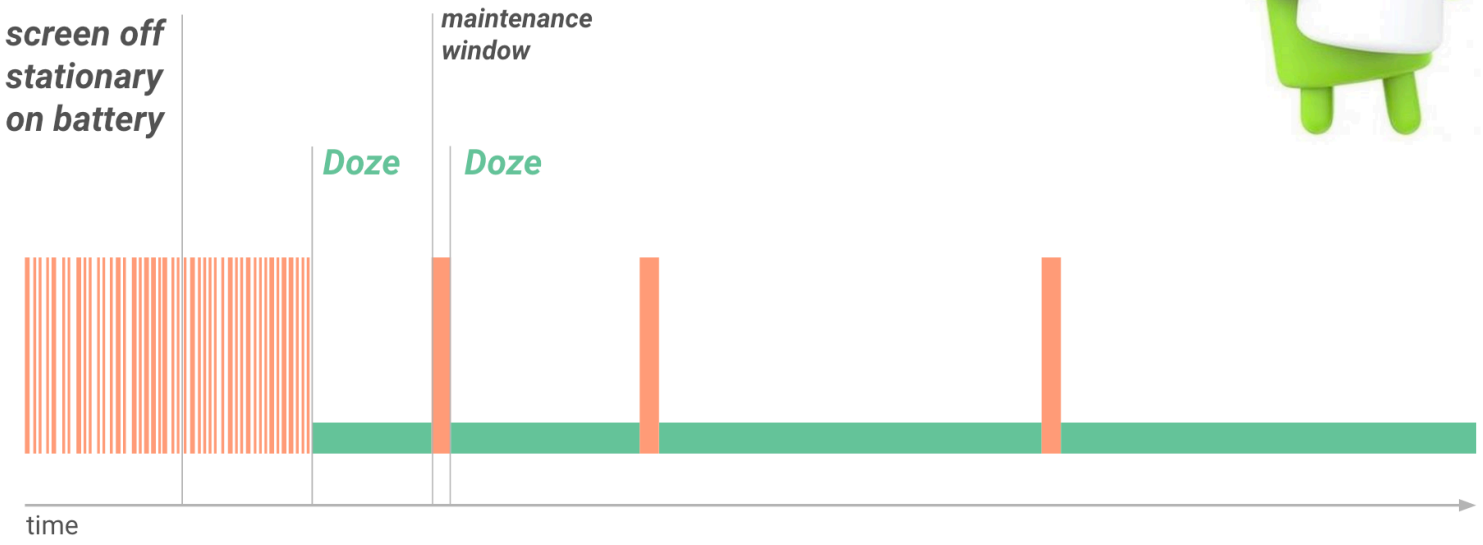
```
public class SampleBootReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction()
            .equals("android.intent.action.BOOT_COMPLETED")) {
            // Set the alarm here.
         }
```

  - Register in the manifest

```
<receiver android:name=".SampleBootReceiver”>
 <intent-filter>
   <action android:name="android.intent.action.BOOT_COMPLETED"></action>
 </intent-filter>
</receiver>
```

Doze Mode

- If a device is not charging nor actively used, it enters Doze Mode



Doze Mode

- The system sleeps most of the time
- Periodic maintenance periods when it wakes up and performs tasks from the backlog
- During the sleep time:
  - Wake locks ignored
  - Network access suspended
  - AlarmManager doesn't work
  - No WiFi scanning
  - Jobs not scheduled (see WorkManager)

unless setAndAllowWhileIdle() or setExactAndAllowWhileIdle()

Doze Mode

- To program with Doze Mode in mind, use
  - Firebase cloud messaging (FCM) for communication apps – a single connection is established
  - Use WorkManager for scheduling jobs
  - Request to be exempt from Doze
    - Can acquire partial wake lock
    - Requires a special permission

      `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS`
- To test how your apps will behave when Doze Mode is active:
  - Force a device/emulator to idle mode


```
adb shell dumpsys deviceidle force-idle
```

WorkManager

- Idea:
  - Guaranteed execution
  - Constraint-aware execution
  - Respect system restrictions
  - Work without GooglePlayServices
- Implementation:
  - Part of Android Jetpack (introduced in 2018)
    - Add as a dependency to your app
  - Backwards compatible
    - Uses JobScheduler for newer APIs
    - Uses AlarmManager for older APIs

WorkManager

- Worker – a unit of work

```
public class UploadWorker extends Worker {

    public UploadWorker(
        @NonNull Context context,
        @NonNull WorkerParameters params) {
        super(context, params);
    }

    @Override
    public Result doWork() {
      // Do the work here, e.g. upload
      uploadImages()

      // Indicate whether the task finished successfully
      return Result.success()
    }
}
```

By default runs on a background thread

WorkManager

- WorkRequest – set constraints, types of execution for your work, e.g.

```
Constraints constraints = new Constraints.Builder()
    .setRequiresDeviceIdle(true)
    .setRequiresCharging(true)
     .build();

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

WorkManager

- Running tasks

```
WorkManager.getInstance().enqueue(uploadWorkRequest);
```

Platform-Based Development: Mobile Sensing and Machine Learning

BS UNI studies, Spring 2018/2019

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

Course Admin

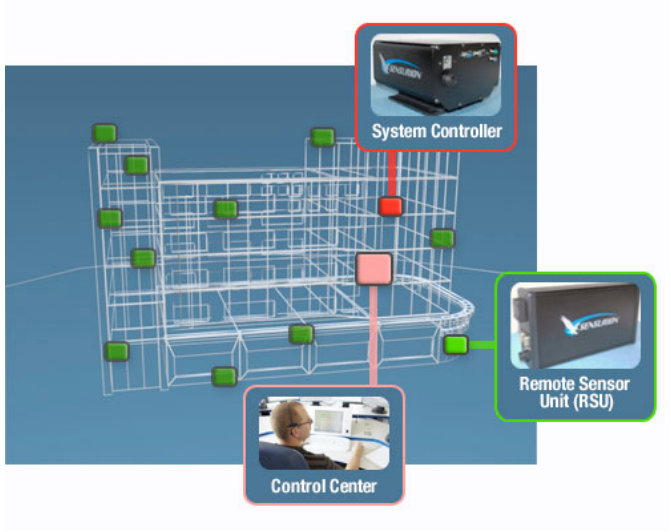
- Sprint #2 consultation labs
  - Tuesday 5pm @ P19
  - Wednesday 5pm @ P19
  - Post questions on Slack #consultations
- Automated testing environment
  - Updated instructions on Ucilnica
  - Certain tests disabled
  - New testing cutoff times:
    - noon
    - 8pm
    - midnight

Course Admin

- Code from the lectures:
  - https://bitbucket.org/veljkop/
- Code from the labs
  - Coming soon!
- No lectures next week!
  - You still have labs at regular lab slots
- No labs the week after!
  - You still have the lectures at the regular slot
- No more homework
  - You still have sprints
- Please fill out the class survey on Ucilnica

Mobile Phone Sensing

- Environment sensing
  - Fixed indoor sensors



- Specialised mobile sensing solutions (early 2000s):
  - Sociometer
  - MSP

Mobile Phone Sensing

- Phone manufacturers never intended their devices to act as general purpose sensing devices
- Sensing components used to improve the interaction with the phone:
  - Accelerometer to trigger screen rotation
  - Gyroscope for playing games
  - Microphone for making calls
  - Camera for taking conventional photos

Mobile Phone Sensing

- Phone sensing requires a significant engineering effort:
  - Frequent sampling with what was supposed to be an occasionally used feature
    - Accuracy problems
    - Battery lifetime
    - Processing overhead
- Android is trying to lower the sensing overhead:
  - E.g. Google Play Services for location updates
- Manufacturers start viewing sensors as a central component of their platforms

Smartphone Sensors

Accelerometer
Magnetometer
GPS
Light
Camera
Barometer
Gyroscope
Proximity
Microphone

WiFi
Bluetooth
GSM
NFC
Touch screen
Thermometer
Humidity sensor

Pros and Cons of Mobile Sensing

- Pros
  - Personalised –suited for sensing human activities
  - Low cost of deployment and maintenance (millions of users where each user charges their own phone)
- Cons
  - General purpose hardware, often inaccurate sensing of the target phenomena
  - Multi-tasking OS. Main purpose of the device is to support other applications
  - Apps could get uninstalled

Applications of Mobile Sensing

- Individual sensing:
  - Fitness applications
  - Behaviour intervention applications
- Group/community sensing:
  - Sense common group activities and help achieving group goals, environmental sensing
- Urban-scale sensing:
  - Large scale sensing - a large number of people have the same application installed; e.g. tracking speed of disease across the country

Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles,Tanzeem Choudhury, Andrew T. Campbell.A Survey of Mobile Phone Sensing. IEEE Communications Magazine. September 2010.

Properties We Can Infer

- Physical activity (running, walking, sitting)
  - Accelerometer
- Transport mode (bicycle, car, train)
  - Accelerometer, GPS, WiFi
- Surroundings, context (party, shopping mall)
  - Microphone, camera, Bluetooth
- Human voice (speaker recognition, stress)
  - Microphone
- Many other things:
  - Emotion, depression, sociability, etc.

Phone as a Societal Sensor

SOCIAL SCIENCE

Computational Social Science

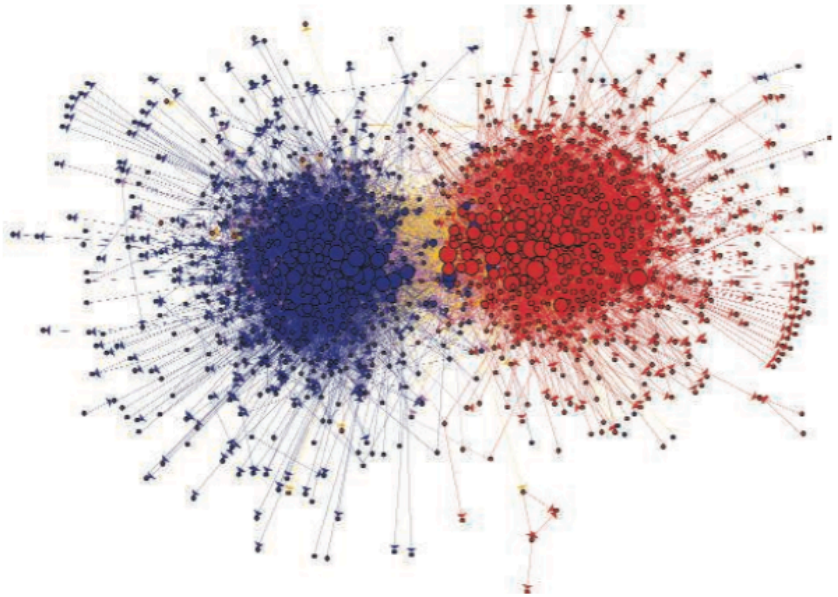
A field is emerging that leverages the capacity to collect and analyze data at a scale that may reveal patterns of individual and group behaviors.

David Lazer,[1] Alex Pentland,[2] Lada Adamic,[3] Sinan Aral,[2,4] Albert-László Barabási,[5] Devon Brewer,[6] Nicholas Christakis,[1] Noshir Contractor,[7] James Fowler,[8] Myron Gutmann,[3] Tony Jebara,[9] Gary King,[1] Michael Macy,[10] Deb Roy,[2] Marshall Van Alstyne[2,11]

We live life in the network. We check our e-mails regularly, make mobile phone calls from almost any location, swipe transit cards to use public transportation, and make purchases with credit cards. Our movements in public places may be captured by video cameras, and our medical records stored as digital files. We may post blog entries accessible to anyone, or maintain friendships through online social networks. Each of these transactions leaves digital traces that can be compiled into comprehensive pictures of both individual and group behavior, with the potential to transform our understanding of our lives, organizations, and societies.

The capacity to collect and analyze massive amounts of data has transformed such fields as biology and physics. But the emergence of a data-driven "computational social science" has been much slower. Leading journals in economics, sociology, and political science show little evidence of this field. But computational social science is occurring—in Internet companies such as Google and Yahoo, and in government agencies such as the U.S. National Security Agency. Computational social science could become the exclusive domain of private companies and government agencies. Alternatively, there might emerge a privileged set of academic researchers presiding over private data from which they produce papers that cannot be critiqued or replicated. Neither scenario will serve the long-term public interest of accumulating, verifying, and disseminating knowledge.

What value might a computational social science—based in an open academic environment—offer society, by enhancing understanding of individuals and collectives? What are the



Data from the blogosphere. Shown is a link structure within a community of political blogs (from 2004), where red nodes indicate conservative blogs, and blue liberal. Orange links go from liberal to conservative, and purple ones from conservative to liberal. The size of each blog reflects the number of other blogs that link to it. [Reproduced from (*8*) with permission from the Association for Computing Machinery]

[1]Harvard University, Cambridge, MA, USA. [2]Massachusetts Institute of Technology, Cambridge, MA, USA. [3]University of Michigan, Ann Arbor, MI, USA. [4]New York University, New York, NY, USA. [5]Northeastern University, Boston, MA, USA. [6]Interdisciplinary Scientific Research, Seattle, WA, USA. [7]Northwestern University, Evanston, IL, USA. [8]University of California–San Diego, La Jolla, CA, USA. [9]Columbia University, New York, NY, USA [10]Cornell University, Ithaca, NY, USA. [11]Boston University, Boston, MA, USA. E-mail: david_lazer@harvard.edu. Complete affiliations are listed in the supporting online material.

StudentLife (2014)

- Study aims to answer the following questions with mobile sensing:
  - why do students some burn out, drop classes, do poorly, even drop out of college, when others excel?
  - what is the impact of stress, mood, workload, sociability, sleep and mental health on academic performance?
  - is there a set of behavioral trends or signature to the semester?

Rui Wang et al. "StudentLife: assessing mental health, academic performance and behavioral trends of college students using smartphones." In *ACM UbiComp'14*

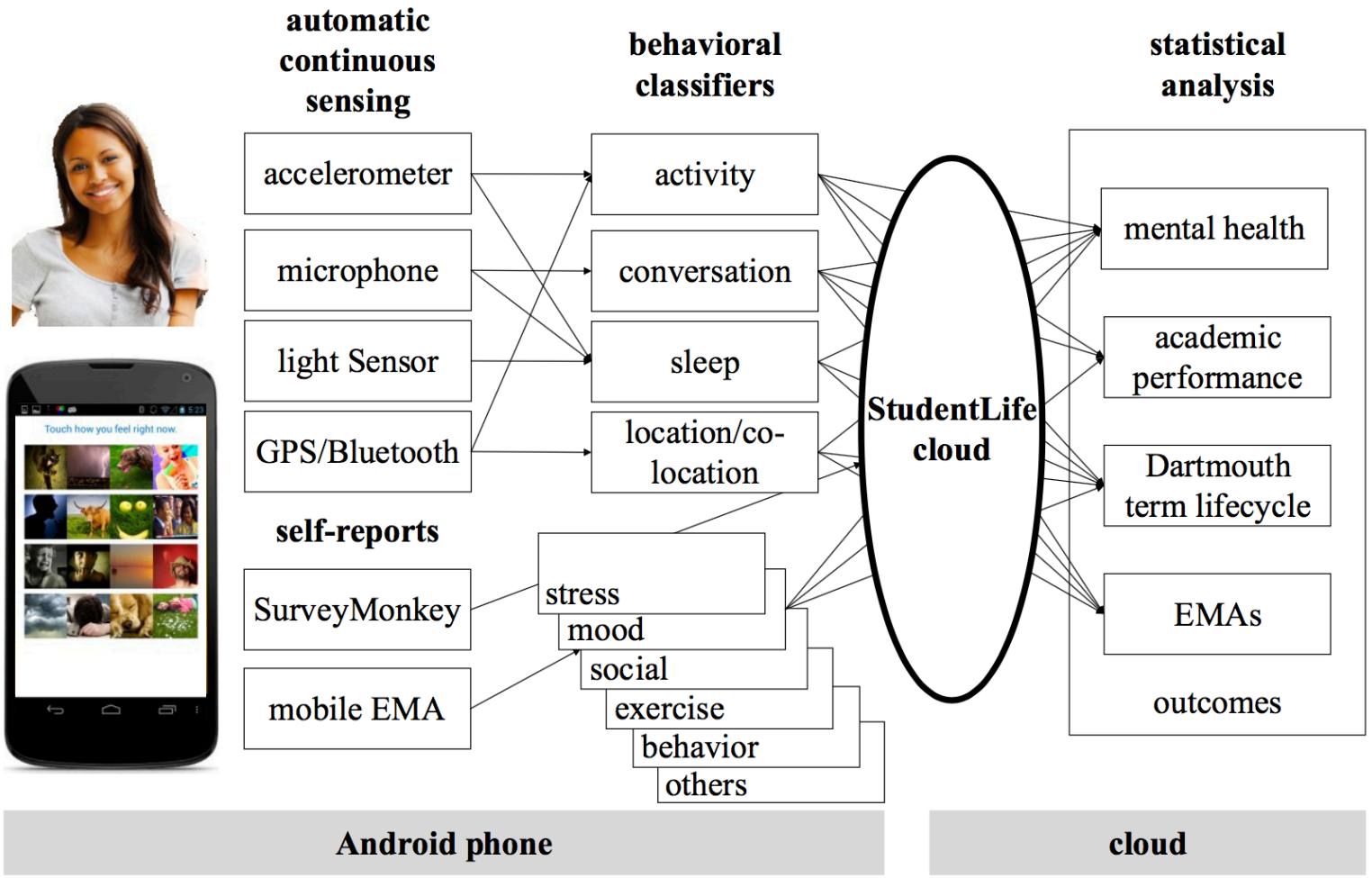
StudentLife (2014)

- Those who work with students subjectively know that there is a cycle in a semester, but no objective data is available

StudentLife Study

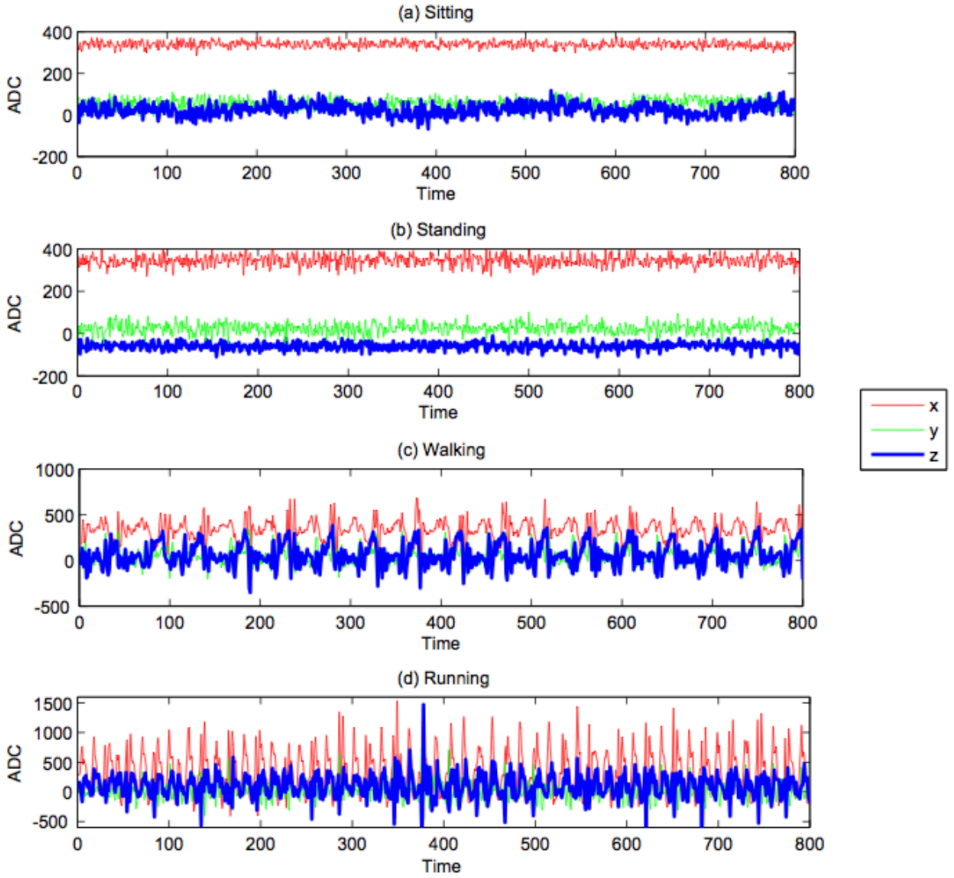
- Automated mobile sensing
  - Many aspects of human behaviour can be inferred with the help of sensing and machine learning: physical activity, location, collocation with other people, even sleep duration
- Experience sampling method (ESM or EMA)
  - Self-reflecting questions about emotions, thoughts
- Study details:
  - 48 students (10 female, 38 male, all CS, 23 Caucasians, 23 Asians and 2 African-Americans)
  - 10 weeks

StudentLife – Sensing System



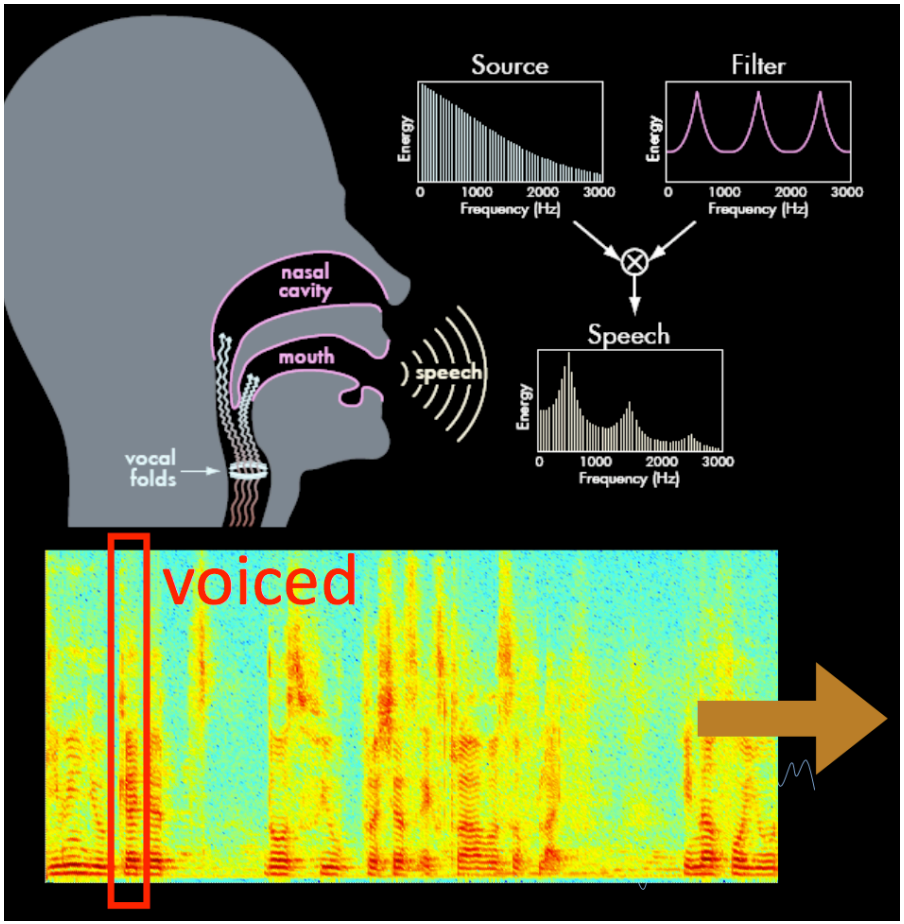
StudentLife Classifiers

- Physical activity
  - A classifier that uses accelerometer data



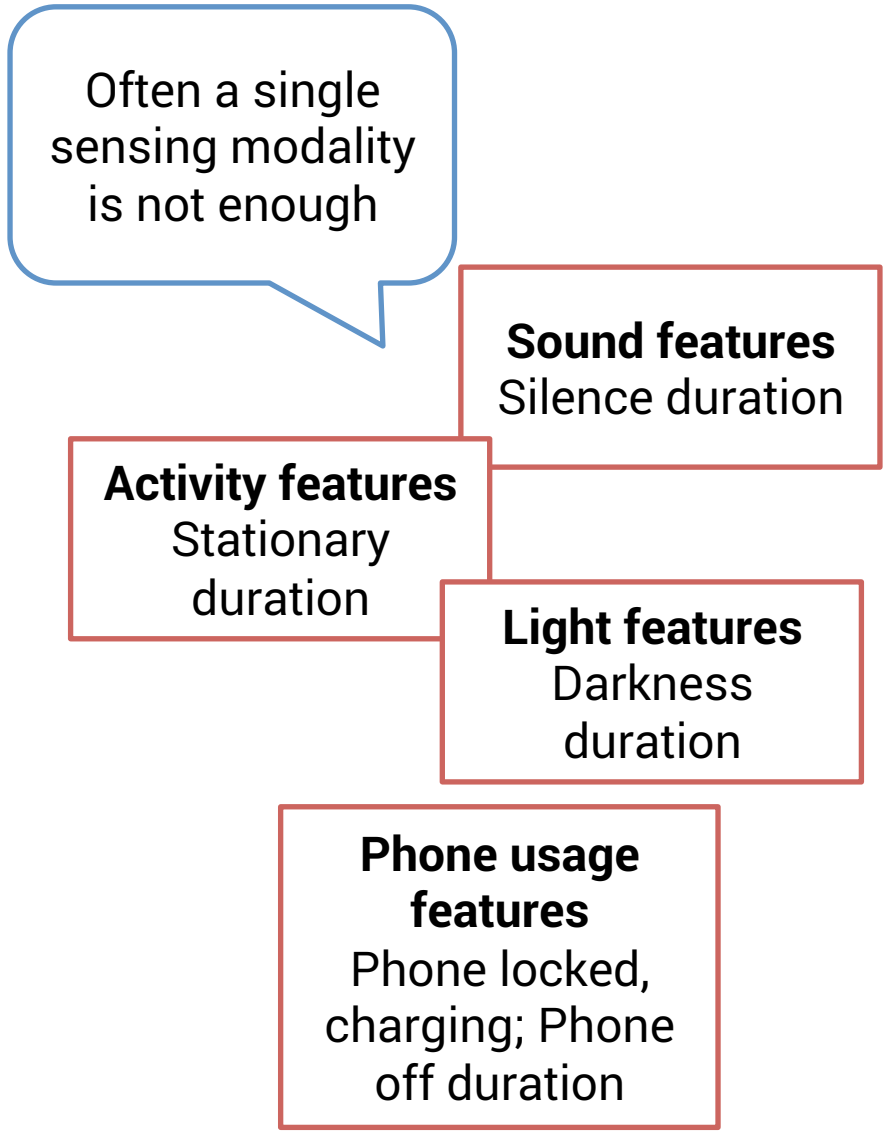
StudentLife Classifiers

- Physical activity
- Face-to-face conversation duration and frequency
  - Detect voiced segments in microphone data, recognise if multiple people are present (ensure that lectures are not counted as conversations)



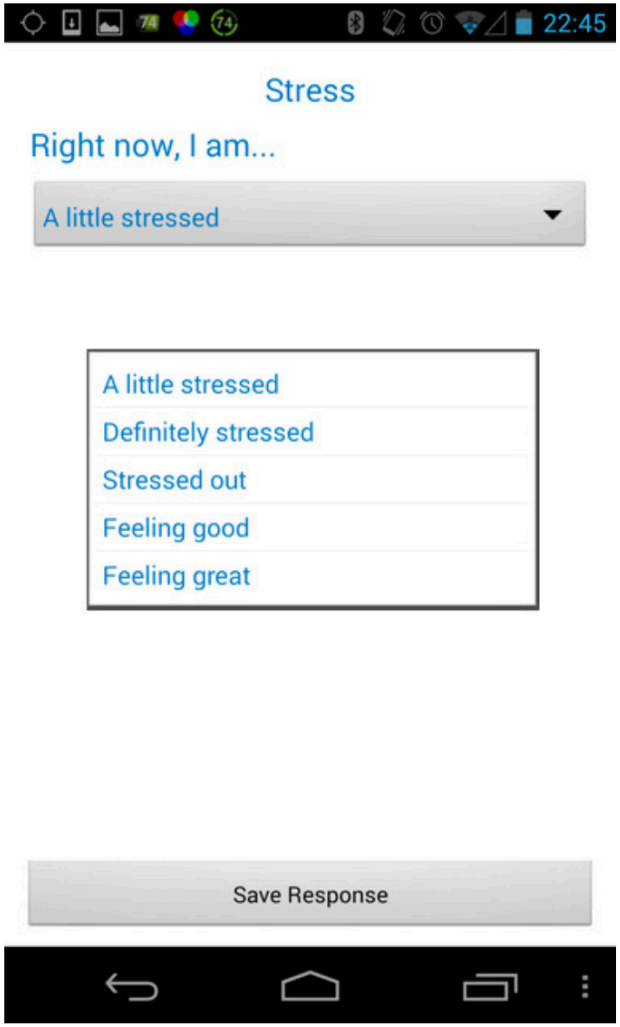
StudentLife Classifiers

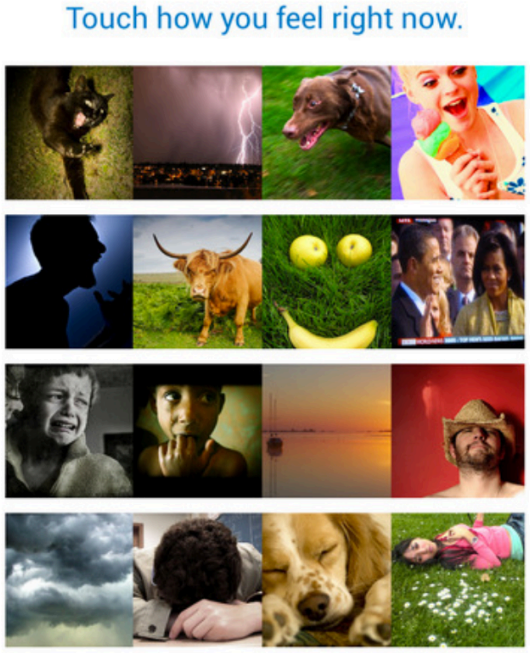
- Physical activity
- Face-to-face conversation duration and frequency
- Sleep duration
  - A linear regression model that takes a variety of features into account



StudentLife ESM

- Experience sampling method (ESM or EMA)
  - Questions about stress, sleep
  - Photographic affect meter



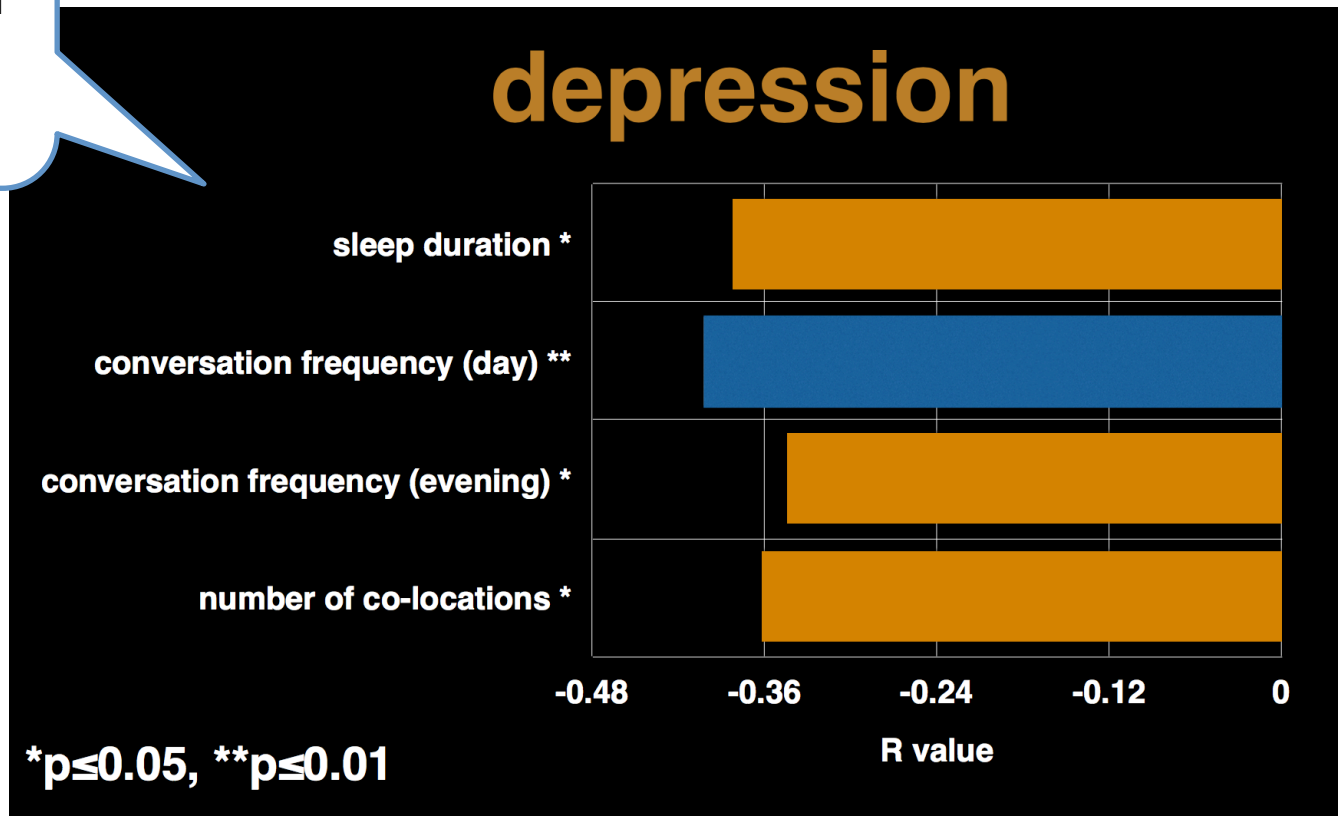


StudentLife Dataset

- 53 GB of data, 32,000 EMAs, 48 surveys, interviews
- Passive sensor data from phone
  - activity, sleep, face-to-face conversation frequency/ duration, indoor and outdoor mobility, location, distance travelled, co- location, light, app usage, calendar, call logs
- ESM data
  - PAM (affect), behavioral, class, campus events, social events, sleep quality, exercise, comments, mood
- Pre-post surveys from Survey Monkey
  - stress, personality, mental and physical health, loneliness
- Grade transcripts, classes information, dining, etc.

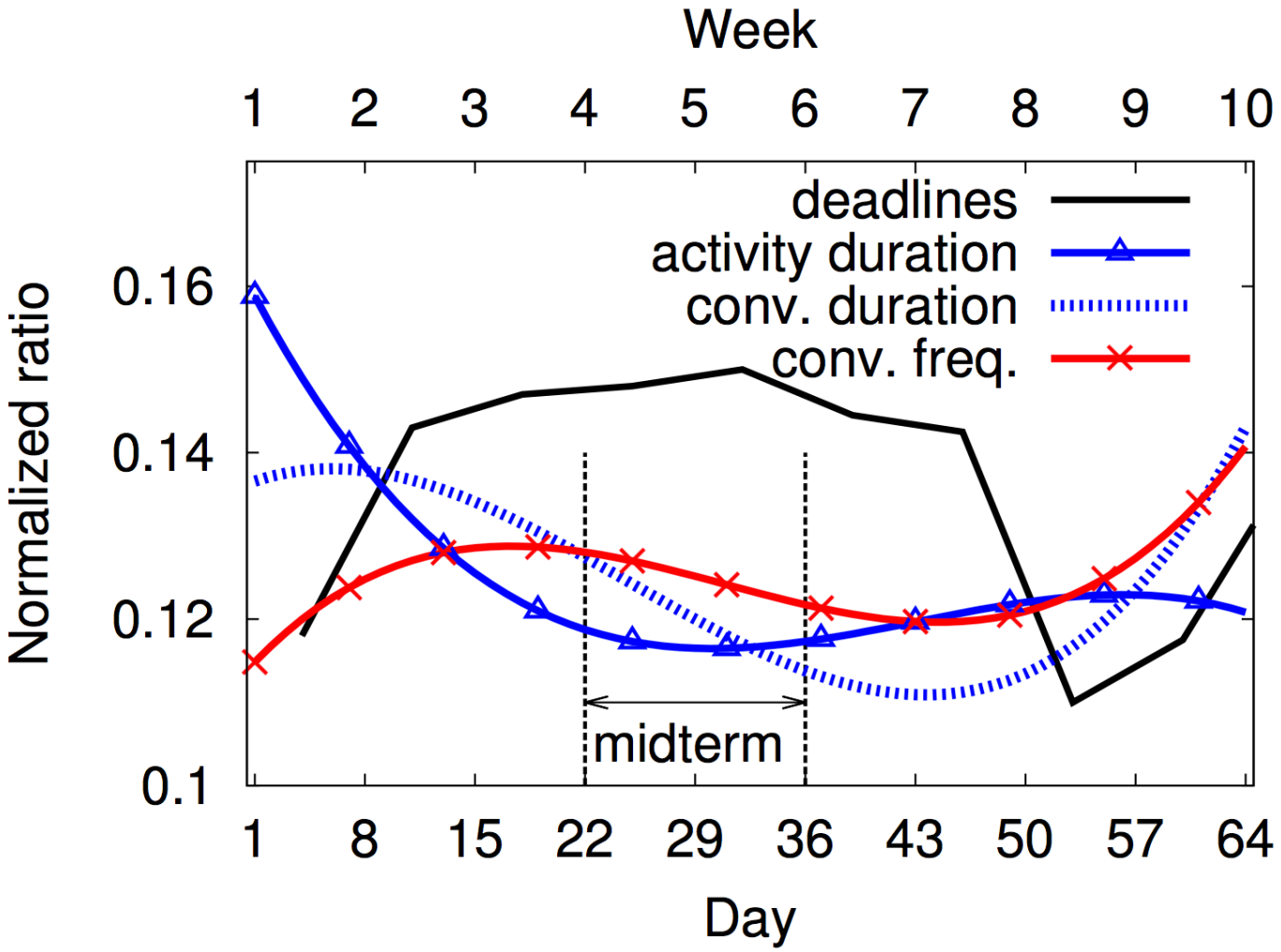
StudentLife (Some) Findings

Mobile sensing data can be used to automatically infer depression



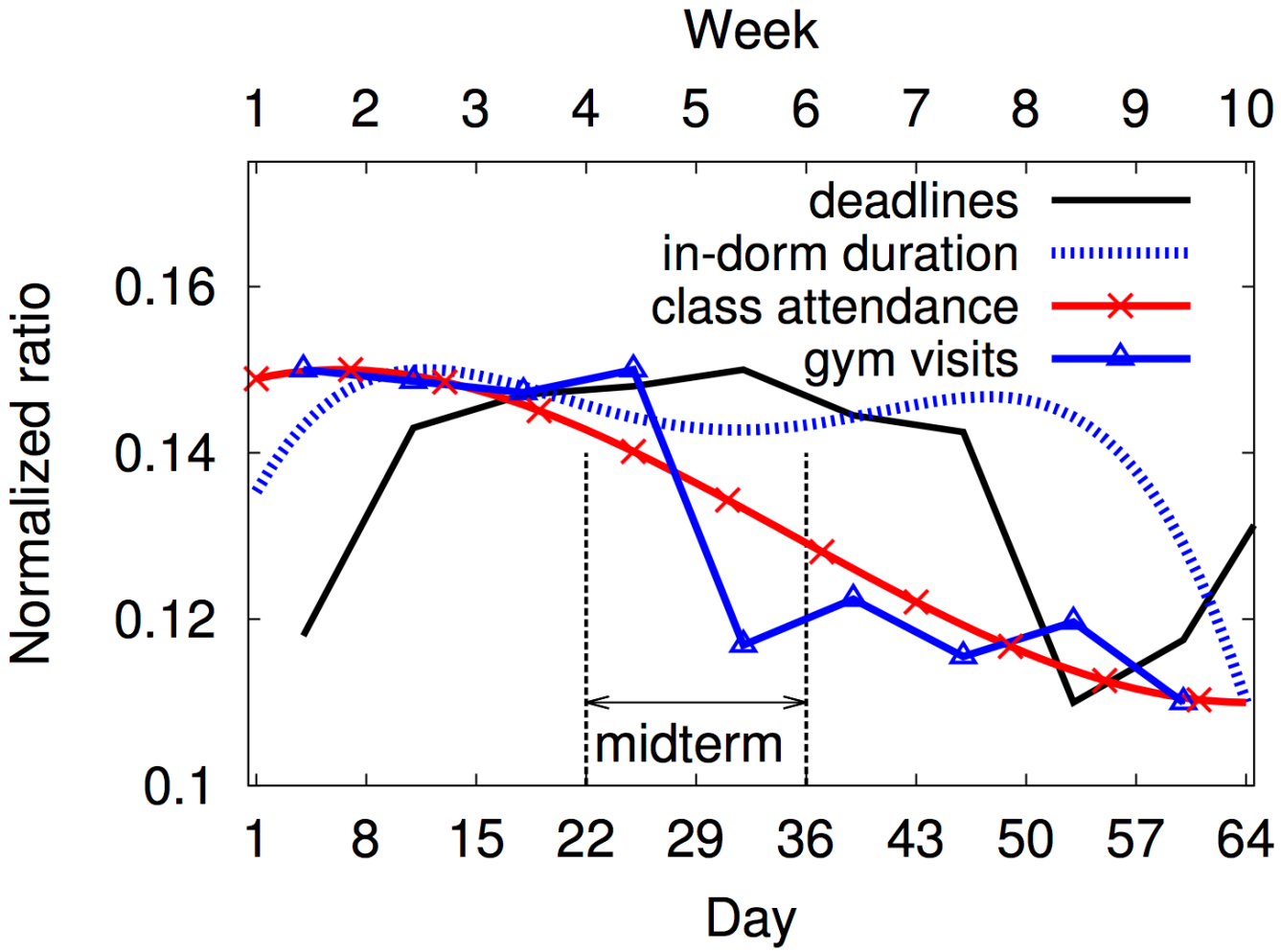
University *of Ljubljana*
Faculty *of Computer and Information Science*

StudentLife (Some) Findings



University *of Ljubljana*
Faculty *of Computer and Information Science*

StudentLife (Some) Findings



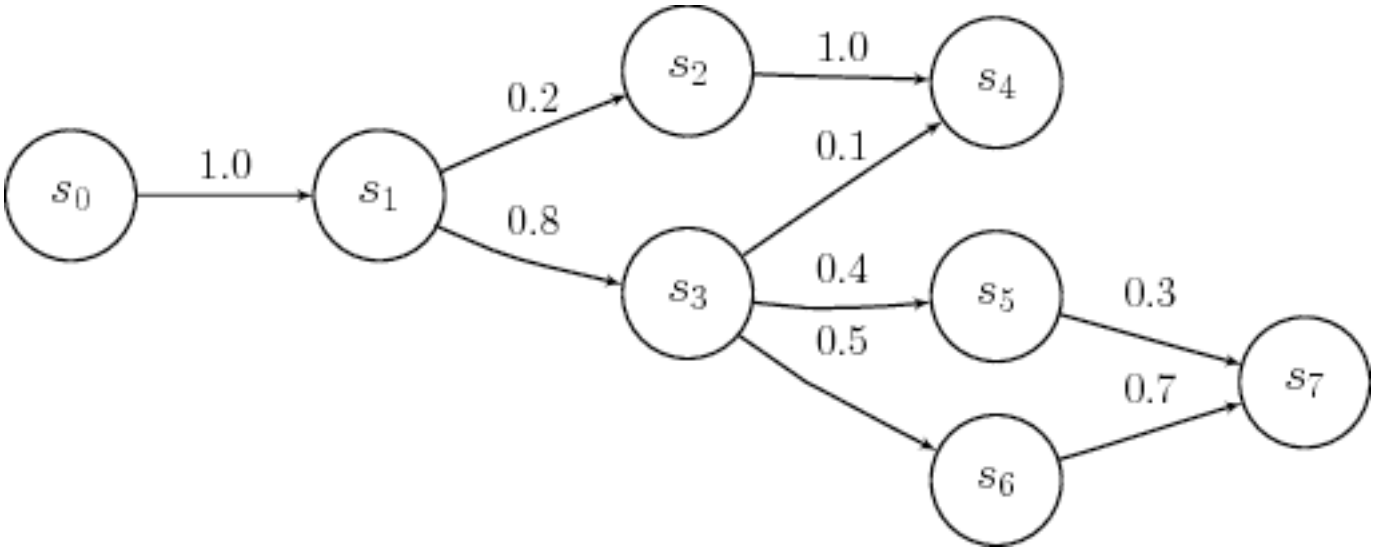
So, it’s all about learning from the data, but how do I do that?

Learning from Sensor Data

- Machine learning algorithms used for:
  - Obtaining high-level inferences from raw sensor data

Learning from Sensor Data

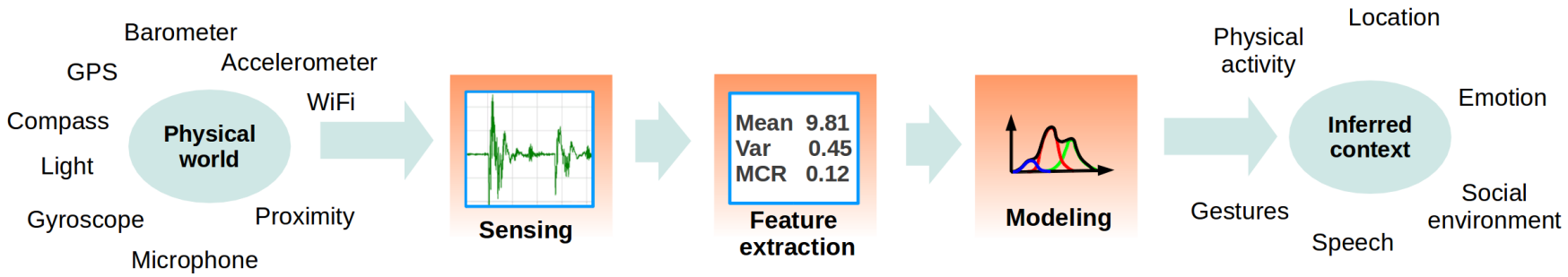
- Machine learning algorithms used for:
  - Obtaining high-level inferences from raw sensor data
  - Predicting a user's context (e.g. mobility prediction)



Learning from Sensor Data

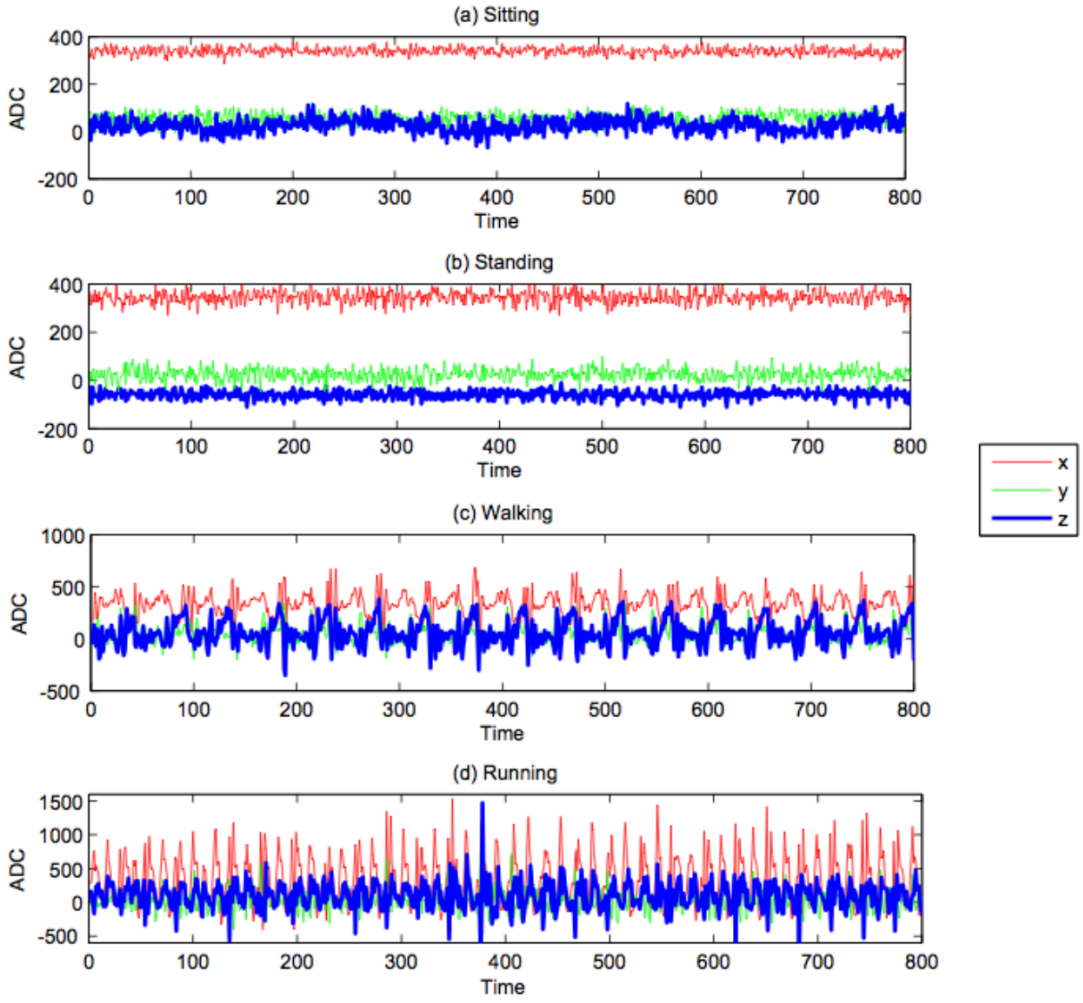
- Machine learning algorithms used for:
  - Obtaining high-level inferences from raw sensor data
  - Predicting a user's context (e.g. mobility prediction)
  - Managing sensor sampling, energy usage, data and computation distribution

From Raw Data to High-Level Inferences



- Get high-level inferences from low-level data:
  - Sample low-level data;
  - Extract useful features

From Raw Data to High-Level Inferences



From Raw Data to High-Level Inferences

- Get high-level inferences from low-level data:
  - Sample low-level data;
  - Extract useful features
    - accelerometer mean, variance, peaks
  - Train a classifier with labelled ground truth data
    - samples collected when we know whether a user is walking, sitting, running, etc.
  - Classify – decide the label for newly-seen data

From Raw Data to High-Level Inferences

OK, I'm convinced, but I don't know much about classification, machine learning, etc.

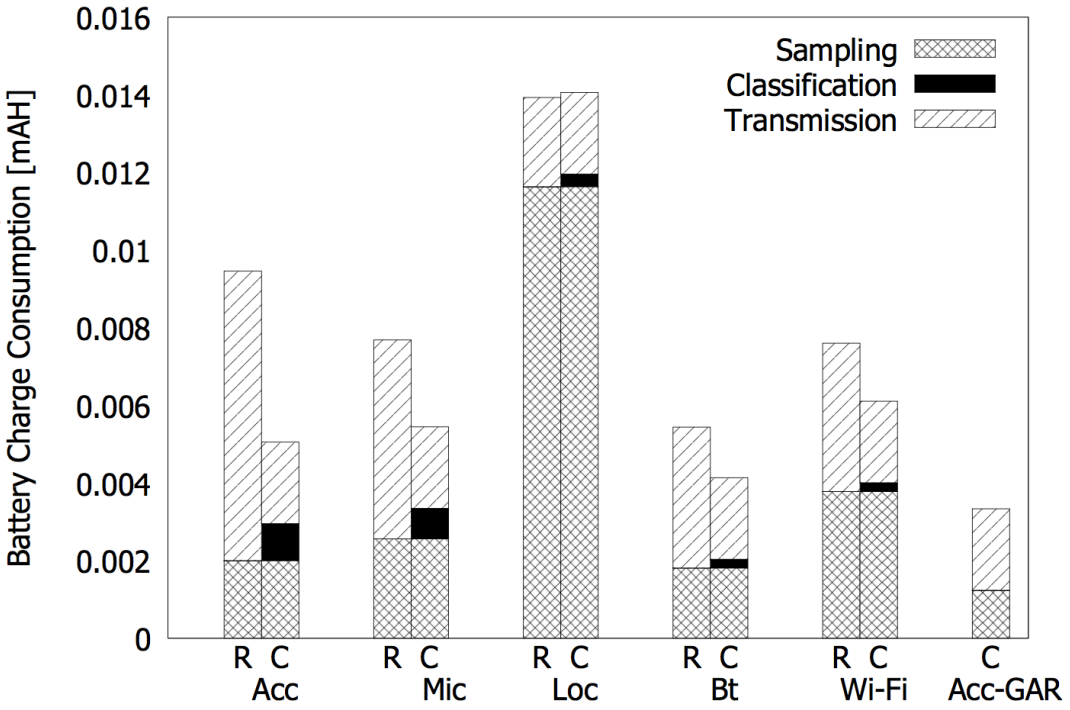
There's a library for that

https://github.com/vpejovic/MachineLearningToolkit/

Battery Charge – A Critical Limitation

- Example measurements for a mobile sensing system on a common smartphone:

Continuous sensing is the quickest way to **negative app store reviews**!



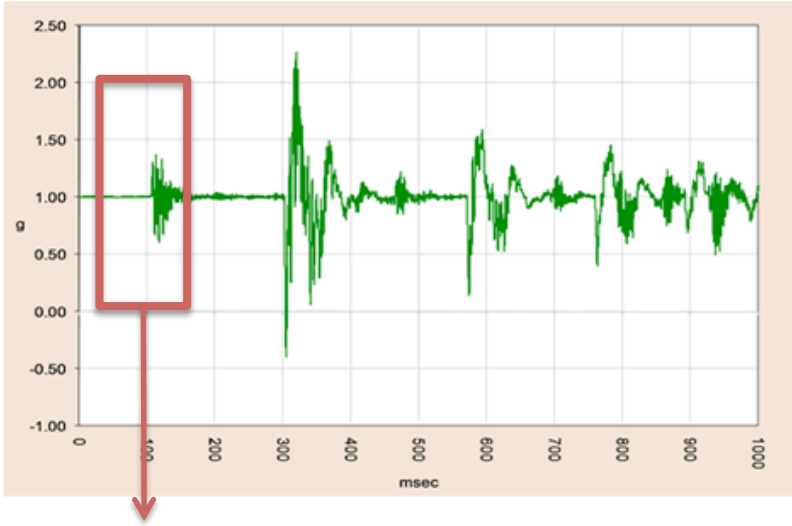
Abhinav Mehrotra et al, SenSocial - A Middleware for Integrating Online Social Networks and Mobile Sensing Data Streams. In *ACM Middleware'14*

Adaptive Sampling

- Frequent sampling depletes phone's resources
- Infrequent sampling may miss interesting events
- Ways to optimise the sampling frequency:
  - Duty cycling
    - Let the device sleep, but adjust the length of time when a device is not sensing according to the distribution of interesting events
  - Hierarchical sensor activation
    - Energy efficient (but perhaps less accurate) sensors are turned on first, if they detect an interesting event, more sophisticated sensors are turned on

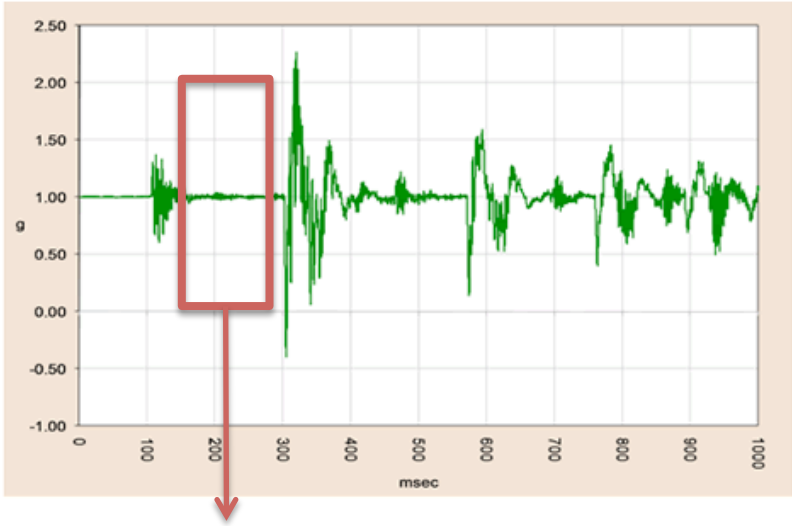
Adaptive Sampling

- No duty cycling:



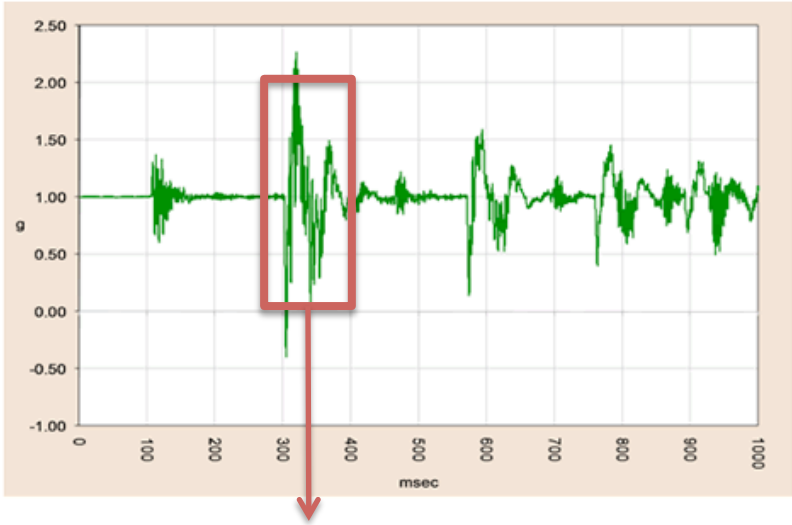
Adaptive Sampling

- No duty cycling:



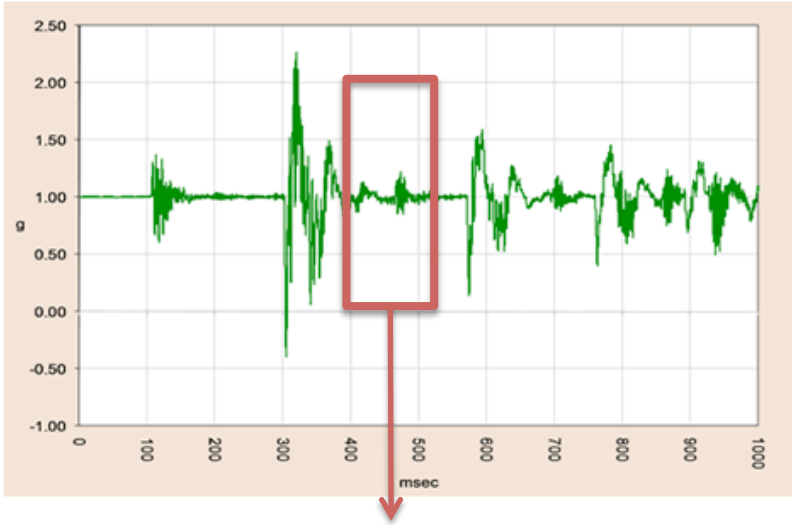
Adaptive Sampling

- No duty cycling:



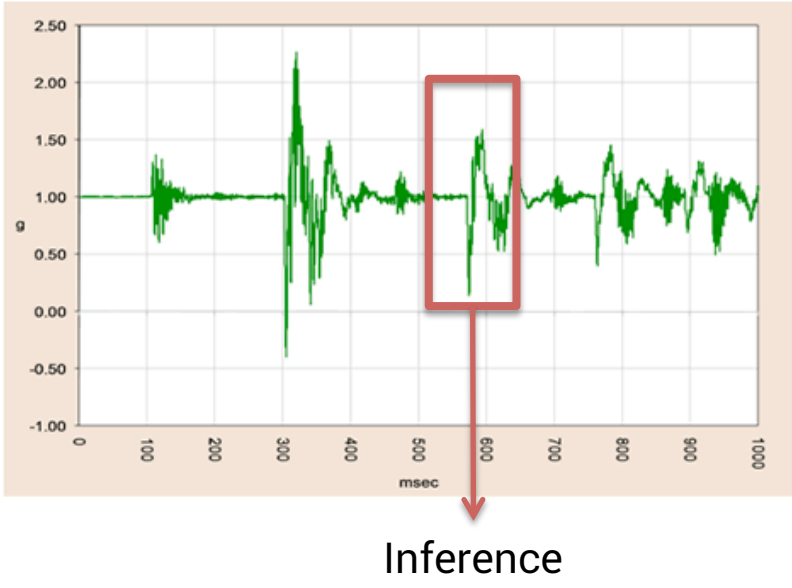
Adaptive Sampling

- No duty cycling:



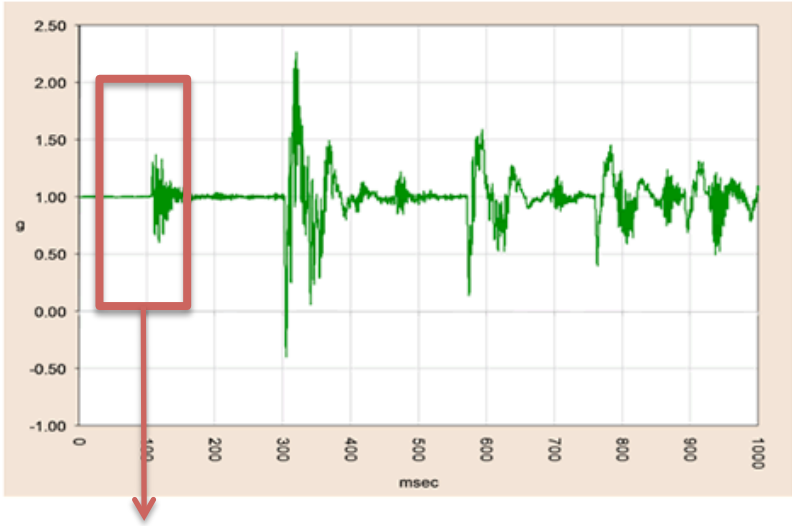
Adaptive Sampling

- No duty cycling:



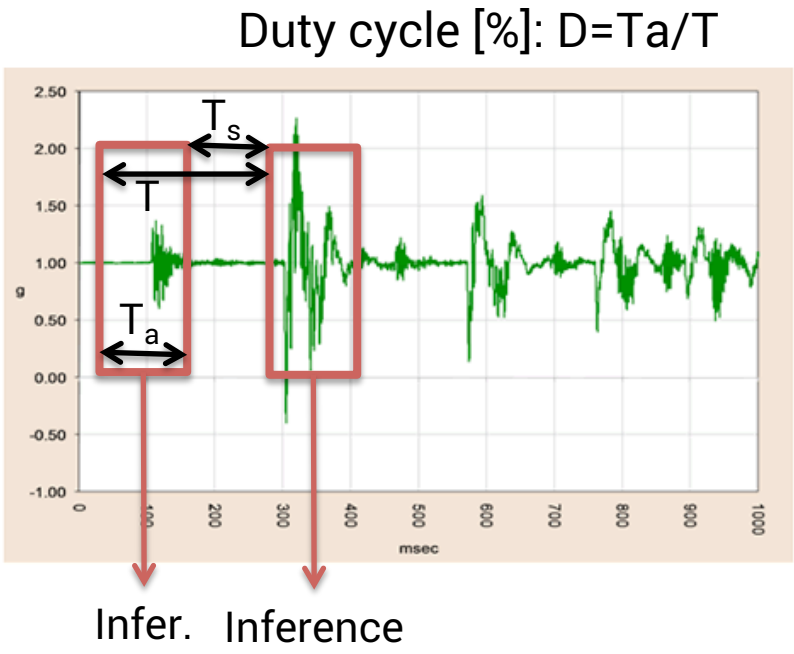
Adaptive Sampling

- Fixed duty cycling:



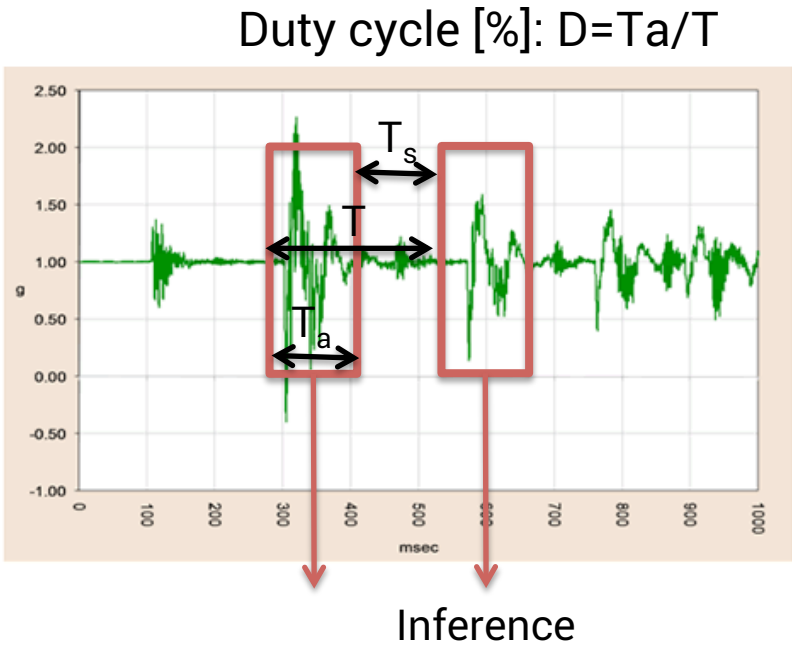
Adaptive Sampling

- Fixed duty cycling:
  - The duty cycle remains the same



Adaptive Sampling

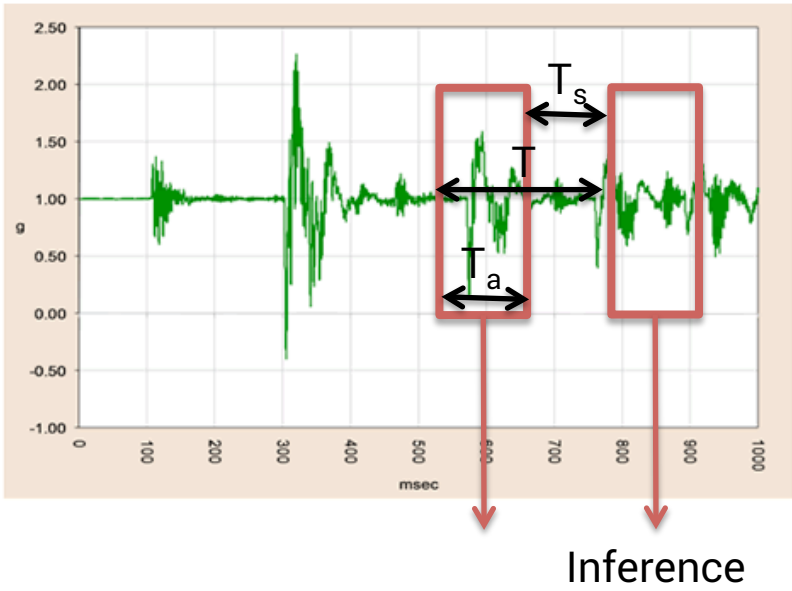
- Fixed duty cycling:
  - The duty cycle remains the same



Adaptive Sampling

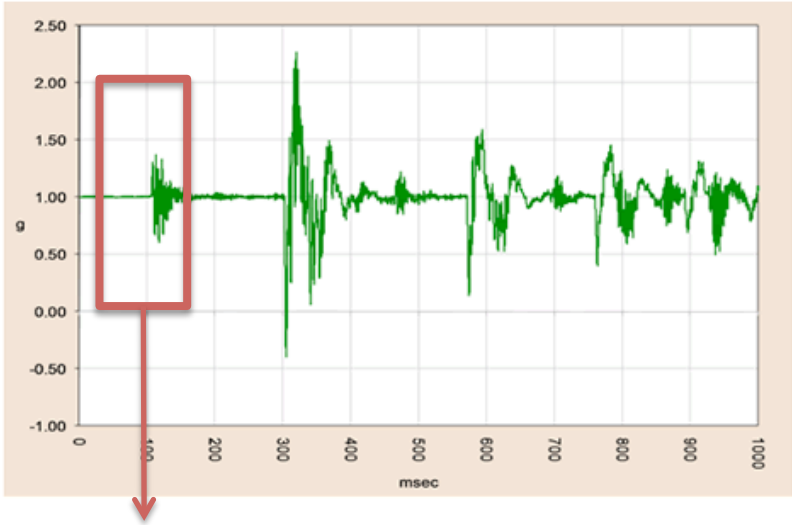
- Fixed duty cycling:
  - The duty cycle remains the same
  - May miss interesting events

Duty cycle [%]: D=Ta/T



Adaptive Sampling

- Adaptive sampling:
  - The sampling schedule varies to adapt to the distribution of interesting events

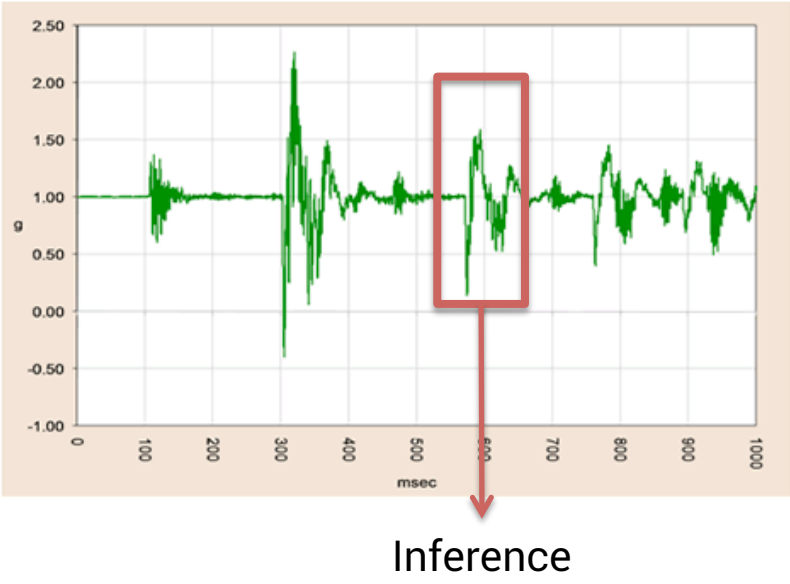


Adaptive Sampling

- Adaptive sampling:
  - The sampling schedule varies to adapt to the distribution of interesting events

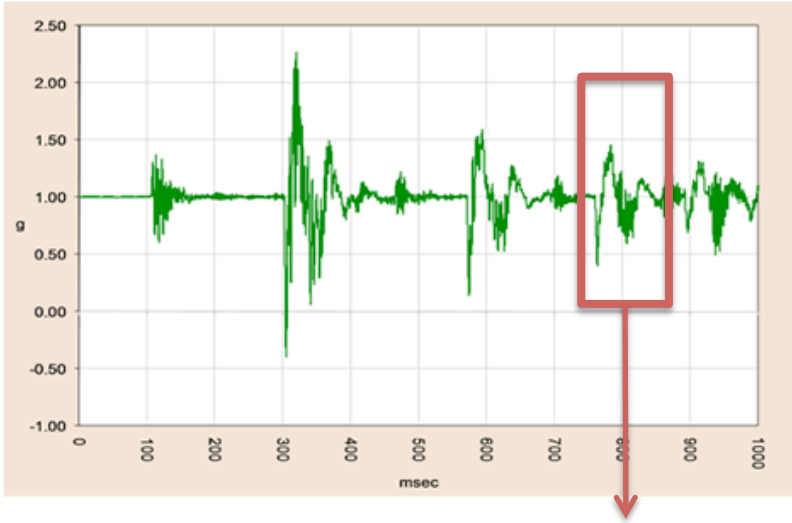
Adaptive Sampling

- Adaptive sampling:
  - The sampling schedule varies to adapt to the distribution of interesting events



Adaptive Sampling

- Adaptive sampling:
  - The sampling schedule varies to adapt to the distribution of interesting events



Adaptive Sampling

- How do we know the distribution of interesting events upfront? We don't!
- However, in reality, interesting events often come in groups:
  - People have conversations, not occasional mutters
  - Users often walk for more than one step
- SociableSense uses the linear reward-inacton algorithm to adapt the probability of sampling

Kiran Rachuri et al, Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *ACM MobiCom'11*

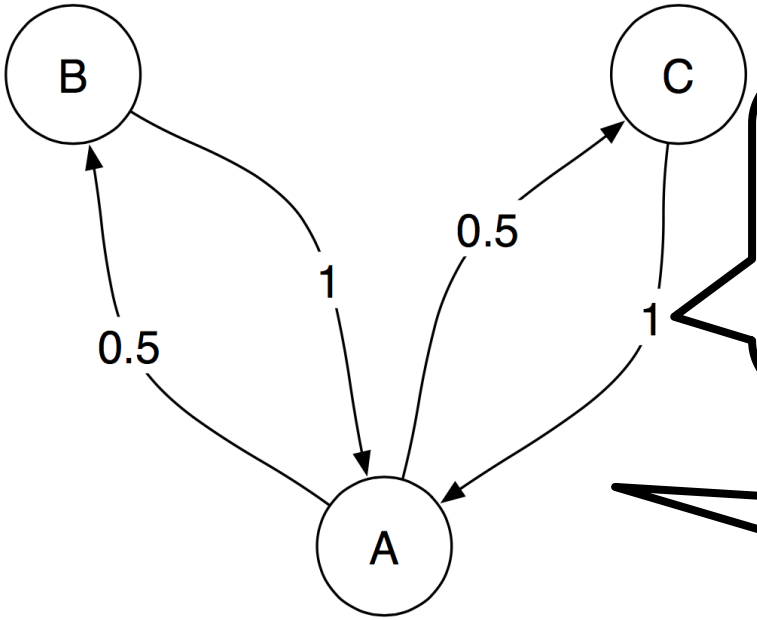
SociableSense (2011)

- Use smartphones to automatically measure social interactions
  - Bluetooth – detect when other users are nearby
  - Microphone – detect conversations
  - Accelerometer – detect movement
- The end goal:
  - Answer questions such as “Do people socialise more in personal office spaces or in common spaces like coffee rooms?”

Measure Sociability

- Sociability – the strength of the user's connection to his/her social group
- Network constraint – quantifies the strength of the node's connectivity
- Draw a weighted social network of users and calculate the network constraint for a given user
  - Lower network constraint – higher strength in terms of connectivity

Measure Sociability



p_{ij}* - The total time spent by *i* with *j divided by **the total time *i* spent with all** users in the network.

Calculate *N* for A, B and C

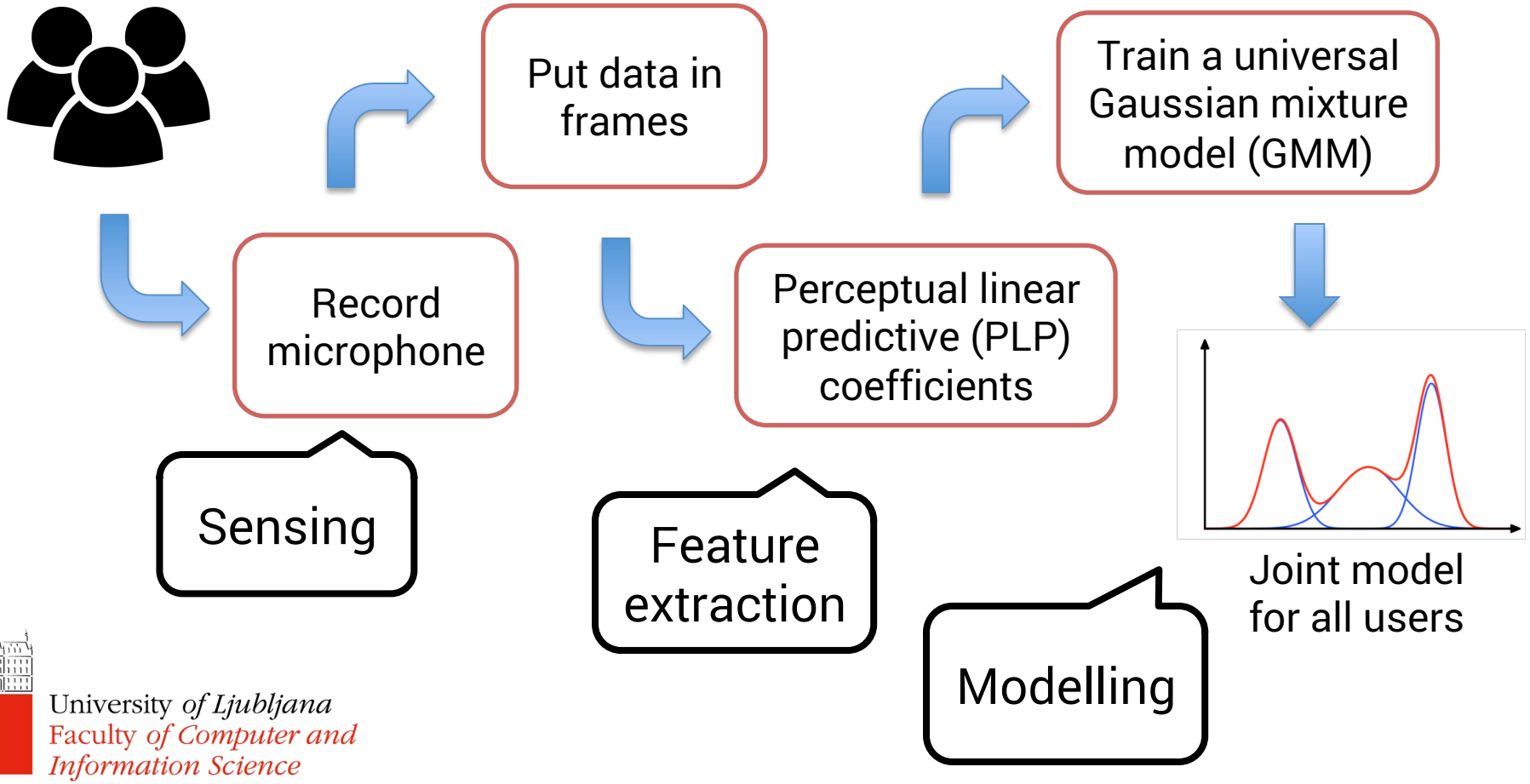
$$N_i = \sum_j (p_{ij} + \sum_q p_{iq} p_{qj})^2, q \neq i, j; j \neq i$$

Measure Sociability

- Estimating “time spent together”
  - Collocation
    - In SociableSense static Bluetooth devices are placed in offices and common rooms of a university building; mobiles that sense the same BT device are collocated
  - Interaction
    - Speech recognition/speaker identification is used to infer if two users are interacting

Speaker Identification

- Speech model for all users:



Speaker Identification

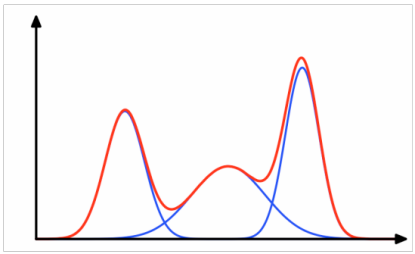
- Refine for a single user:

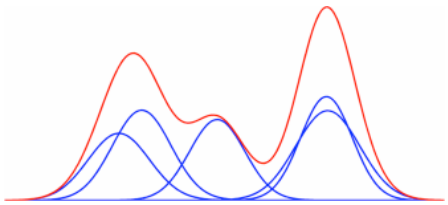
Record microphone

Put data in frames

Perceptual linear predictive (PLP) coefficients

Maximum A Posteriori (MAP) adaptation of the joint model

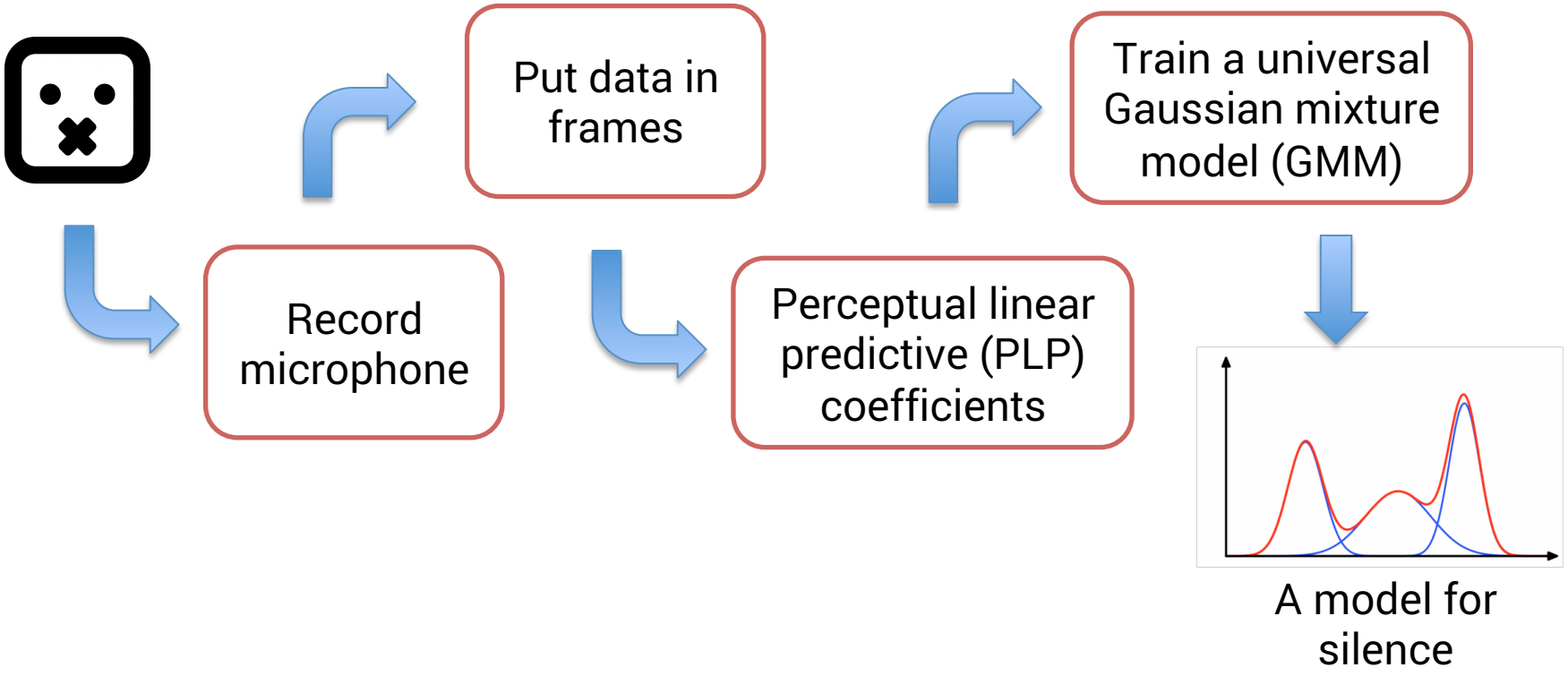




A single user model

Speaker Identification

- Speech model for silence:



Speaker Identification

- Identifying a speaker:

Record microphone

Put data in frames

Perceptual linear predictive (PLP) coefficients

Compare the likelihood of the sample sequence being generated by the joint voiced model vs generated by the silence model

If more likely to be voiced, compare with individual user models, and select the model that most likely generated the sequence

Linear Reward-Inaction Algorithm

- Action – sensing
- Probability of action – *p*
- Success – interesting event sensed
- Failure – interesting event not sensed
- Adaptation:
  - If successful, increase the probability: $p = p + \alpha(1 - p)$
  - If unsuccessful, decrease the probability: $p = p - \alpha p$

Linear Reward-Inaction Algorithm

- Missed events vs energy savings
  - Bluetooth sampling example

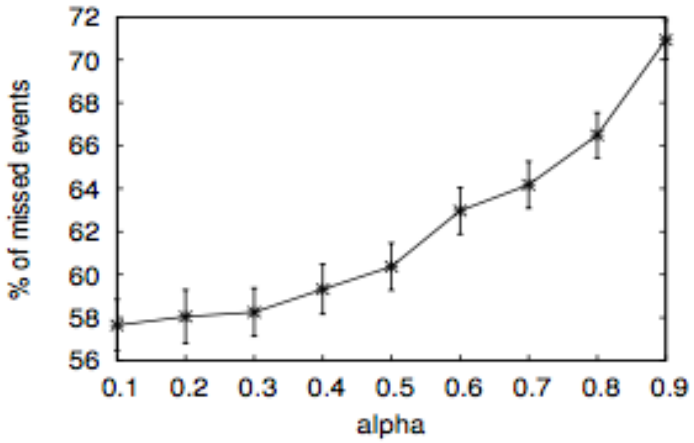


Figure 4: Accuracy (% of missed events) vs alpha for Bluetooth.

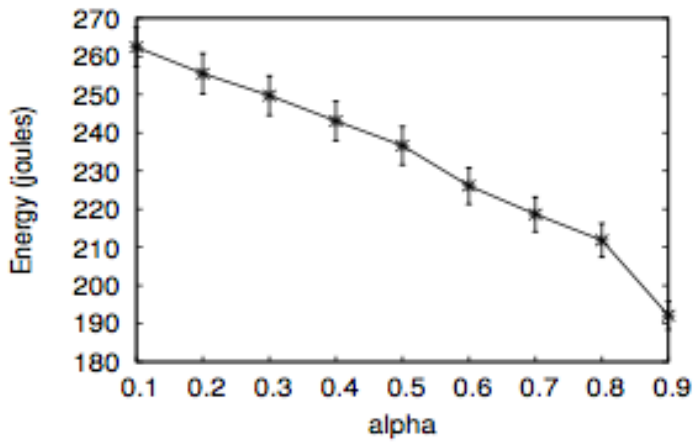
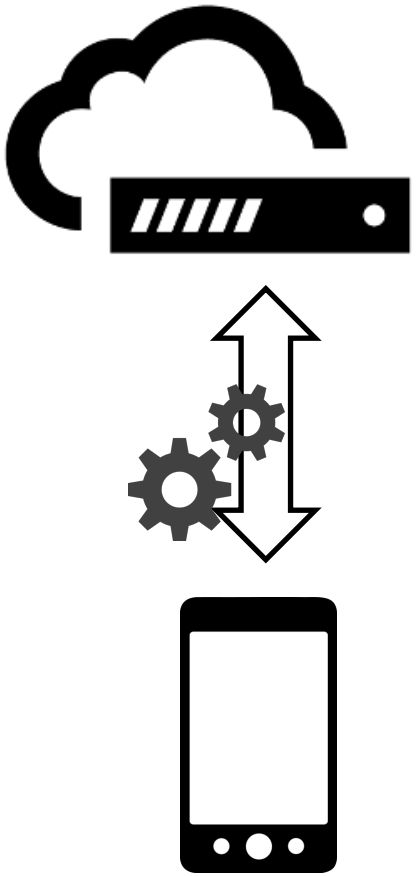


Figure 5: Energy consumption per hour vs alpha for Bluetooth.

Computation Distribution

- Processing (classifying) sensor data:
  - On the phone: may be faster than on a remote server, drains battery
  - In the cloud: phone-cloud transfer may incur data charges, global view, more computing resources available
  - Some parts of a task may be processed on the phone, some on the cloud
- SociableSense uses multi-criteria decision theory to decide where to execute parts of data processing task



Computation Distribution

- Configurations – each of n parts of a single task can be processed either in the cloud or on the phone: 2^n possible configurations
- Utility functions for configuration *i*:

$$u_{e_i} = \frac{e_{\min} - e_i}{e_i}$$ Energy

$$u_{l_i} = \frac{l_{\min} - l_i}{l_i}$$ Latency

$$u_{d_i} = \frac{d_{\min} - d_i}{d_i}$$ Data

- Combined utility function for configuration *i*:

$$u_{c_i} = w_e u_{e_i} + w_l u_{l_i} + w_d u_{d_i}$$

Computation Distribution

- Uc_i is calculated for each configuration and the one with the highest utility is selected
- Weights (w_e, w_d, w_i) can be adjusted to save energy, lower latency or min. data plan usage
- Example: speaker identification (two subtasks)

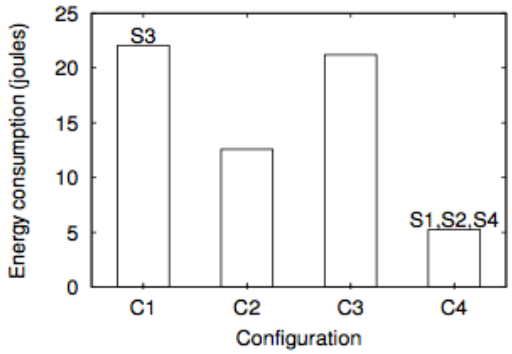


Figure 13: Energy consumption for processing the speaker identification task.

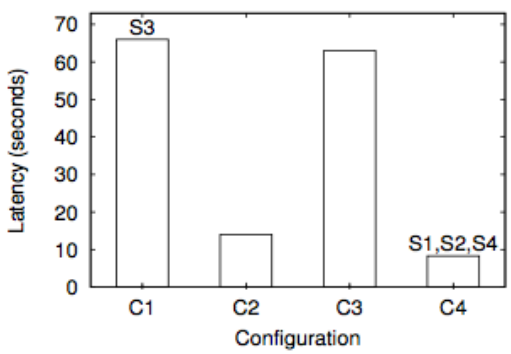


Figure 14: Latency or delay for processing the speaker identification task.

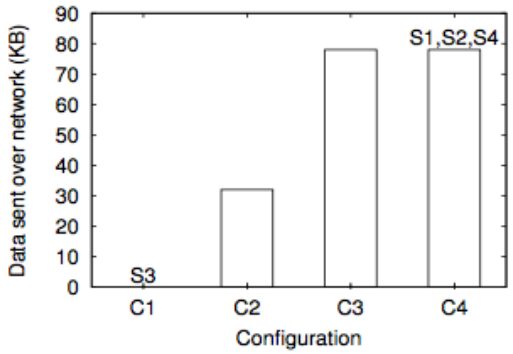
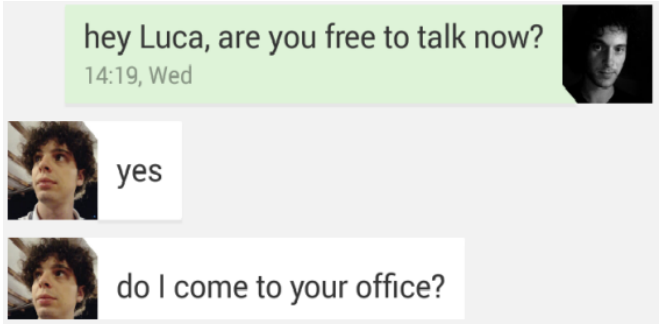
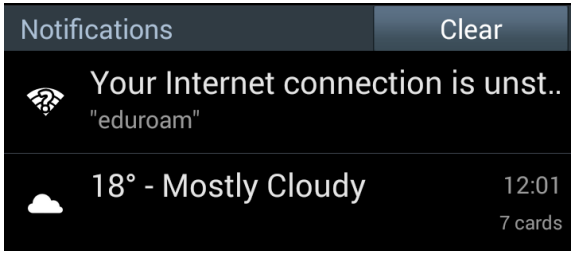


Figure 15: Data sent over the network for processing the speaker identification task.

InterruptMe (2014)

- Mobile phone is the most direct point of contact, our lives are increasingly interactive
- Mobile notifications arrive at wrong moments, interfering with our lifestyle, interrupting ongoing tasks



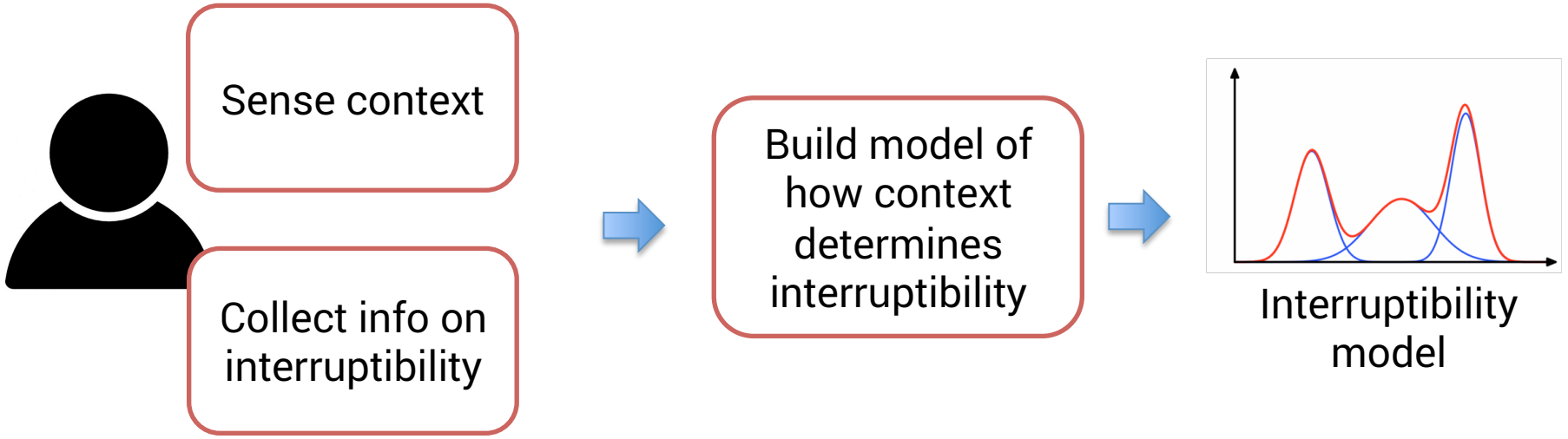


InterruptMe

- Goal: design and develop a system for intelligent notification scheduling on a mobile device
- Hypothesis: a user's context (location, movement, physical activity, time of day, day of week) determines whether a user is interruptible or not
- Opportunity: smartphone sensors can provide the above context information

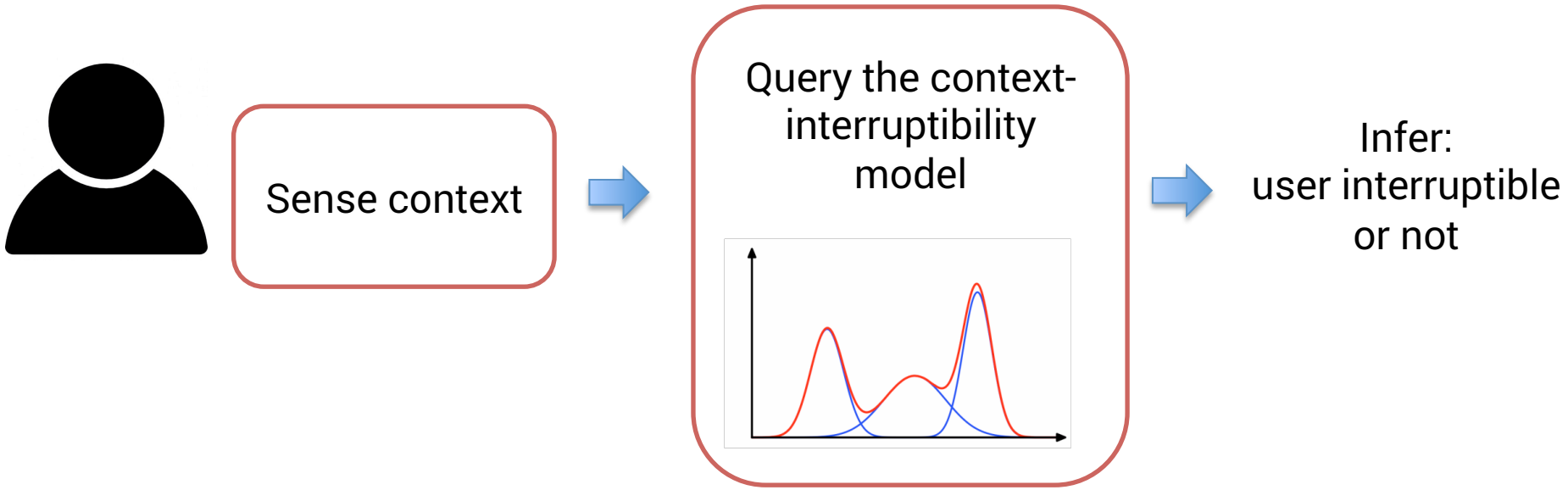
InterruptMe

- Idea:
  - Training period: construct context-interruptibility model



InterruptMe

- Idea:
  - Test period: use context-interruptibility model



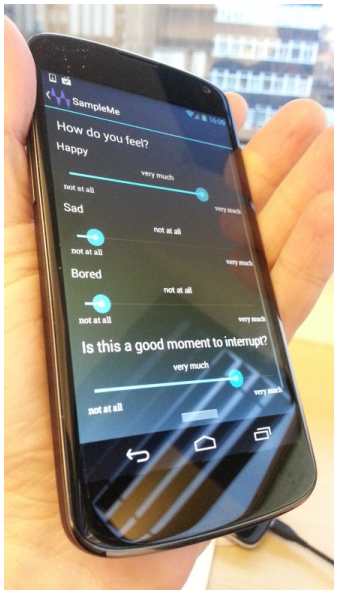
Sensing Interruptibility

- Smartphone sensors give us the context





- But how do we get the info on interruptibility?
  - SampleMe: Android mobile sensing app
    - Send notifications
    - Record sensor data
    - Record user reaction
    - 20 users, two weeks, eight notifications per day

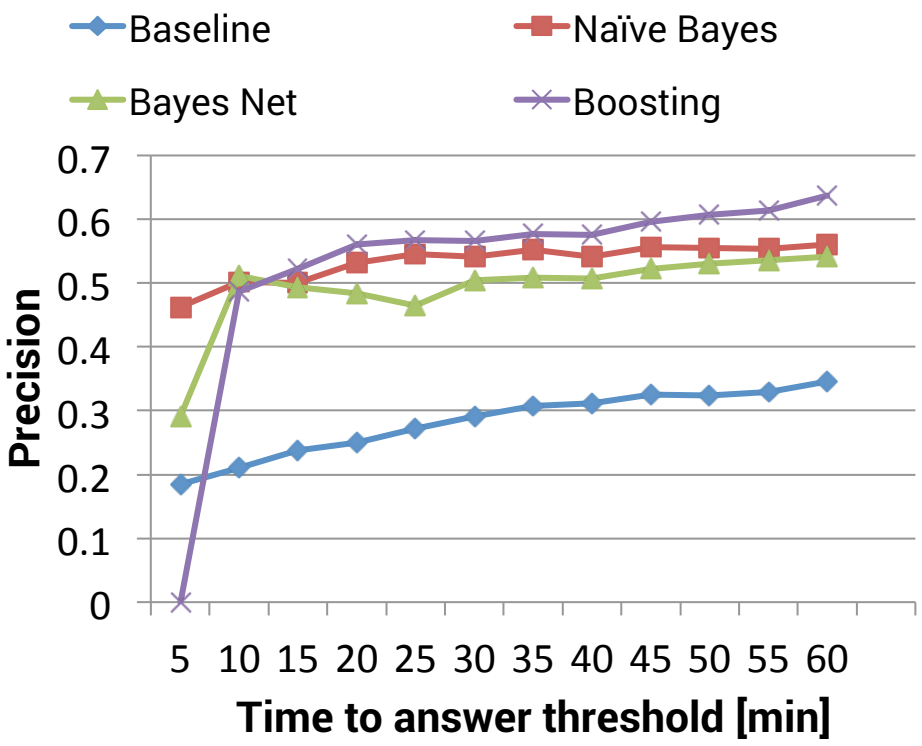


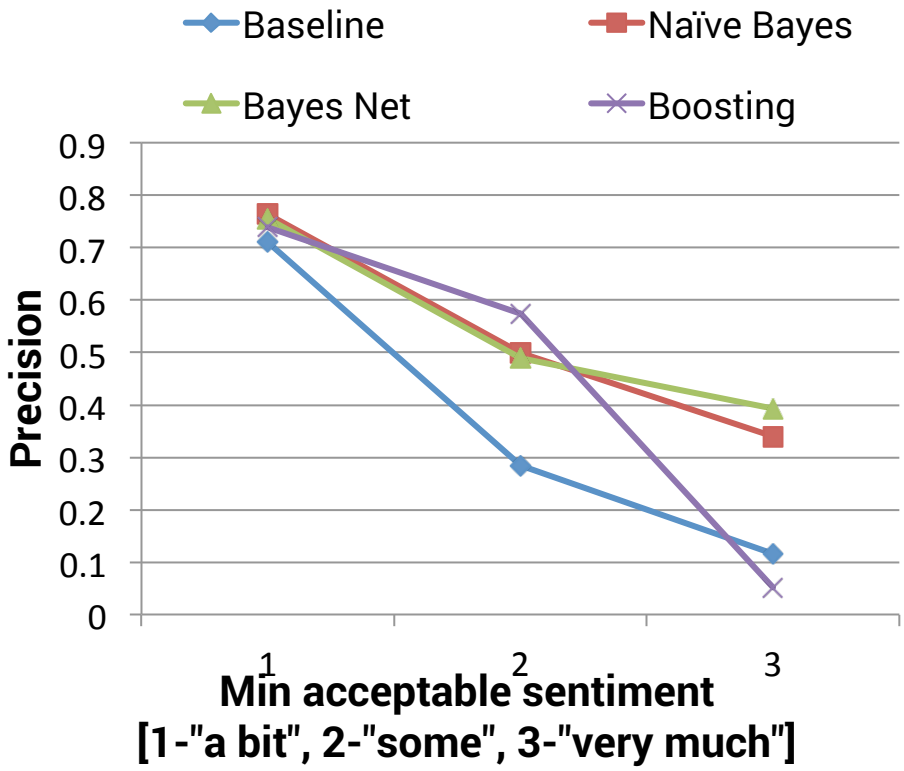
Building a model

- Define “interruptible”
  - Example: answers to a message within 15 minutes
  - Example: answers “this is a good moment to interrupt”
- Select a classifier:
  - Batch vs online
    - Batch is trained once
    - Online can be updated as new information comes in
  - Complexity, type of learning:
    - Naïve Bayes
    - Bayes Net
    - SVM
    - etc.

Building a model

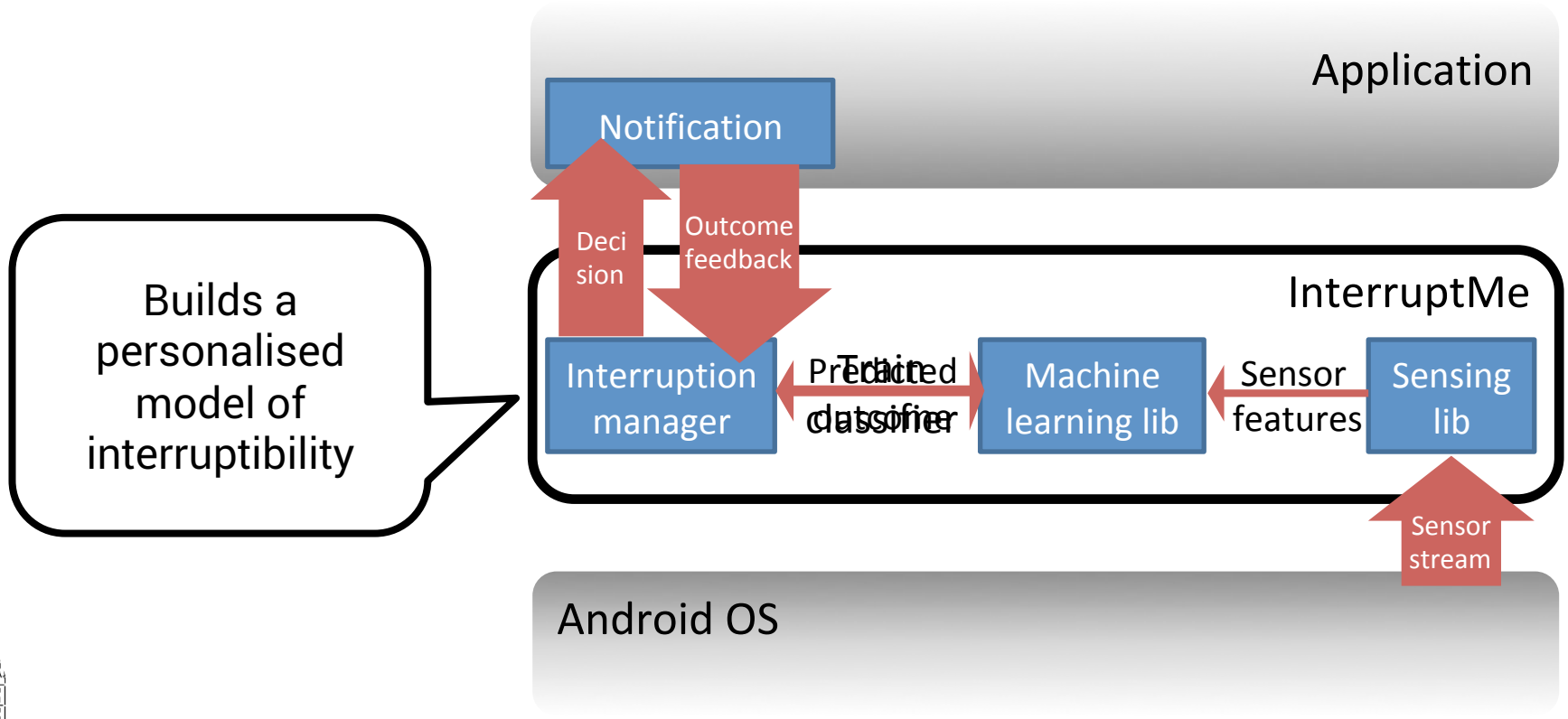
- Test a few models in WEKA:





Implement in Android

- InterruptMe – notification management library



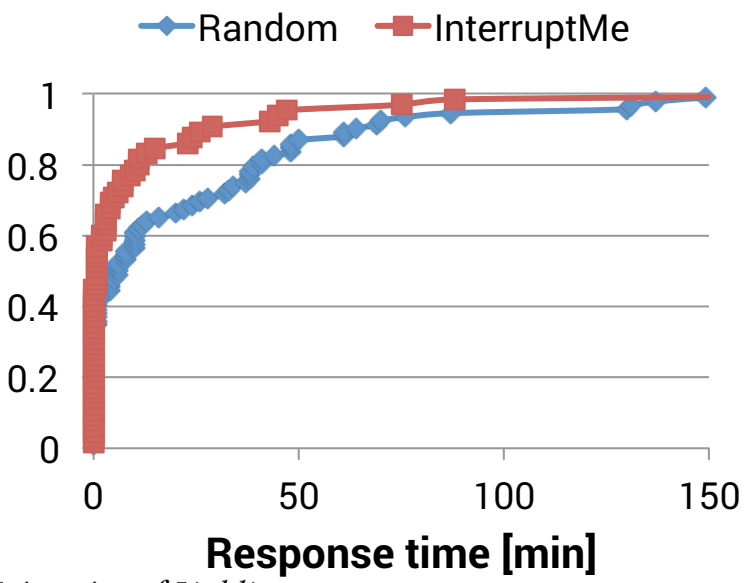
InterruptMe

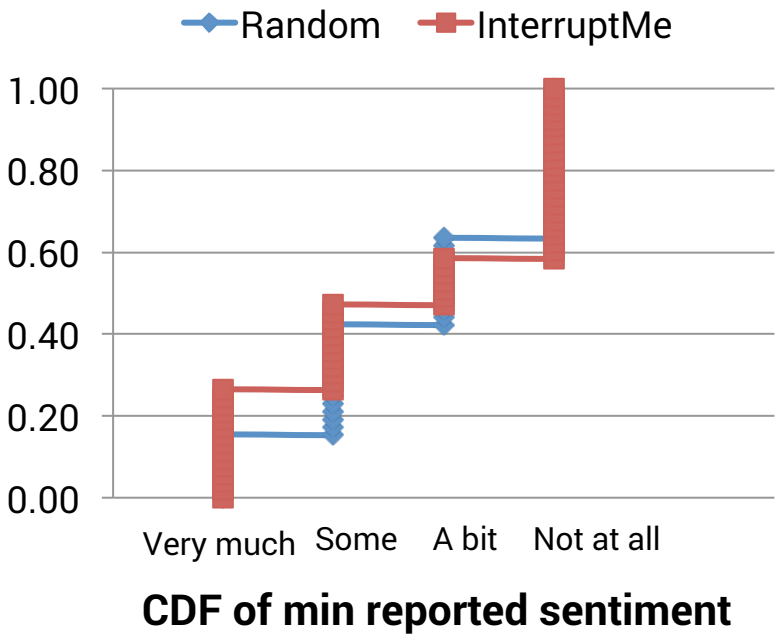
```
public class NotificationService extends Service implements IntelligentTriggerReceiver {

    IntelligentTriggerManager interruptionMngr;

    public void onCreate() {
        interruptionMngr = IntelligentTriggerManager.getTriggerManager(this);
    }

    public void onTriggerReceived(String a_triggerID,  ArrayList<LearnerResultBundle> a_bundles) {
        // … send a notification using NotificationManager
    }
```

```
// … in case user responds to a notification
interruptionMngr.trainLearnerFromFeedback(Constants.MOD_INTERRUPTIBILITY, currentTriggerNo, "yes");
```

Evaluation

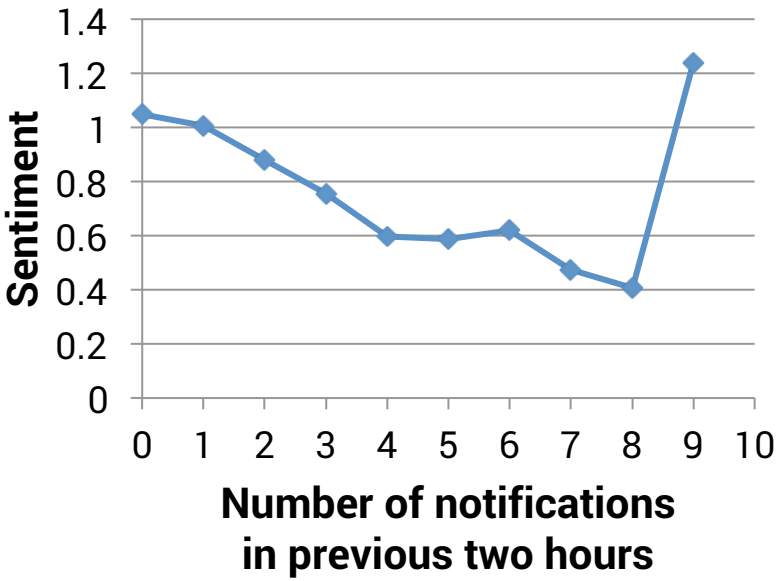
- SampleMe + InterruptMe
  - Context-aware experience sampling
  - N-of-1 trial (random vs intelligent notif triggering)
  - Ten users for one month





InterruptMe – Limitations

- A user's reaction also depends on:
  - The task they are doing at the moment
  - The relationship between the sender and the receiver
  - The content of the notification
- A limited amount of "interruptibility" is available at any moment



Take-Home Messages

- Thanks to sensors, mobile devices can tell us a lot about the user activities, behaviours, even emotions
- Machine learning connects raw sensor data (e.g. light intensity, acceleration values, etc.) and high-level concepts (e.g. sleeping/awake)
- Sensing design is always about trade-offs: get the necessary data with minimum resources
- Do not reinvent things unless you have a good reason to believe your approach is much better

Platform-Based Development: Third Party Libraries

BS UNI studies, Spring 2018/2019

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

Course Admin

- Sprint #3 instructions are out:
  - http://bitbucket.org/veljkop/runsup/
  - All in one .txt (or .md) file
  - Deadline May 12th 23:59
- Sprint #3 main improvements:
  - More sophisticated UI
    - NavigationDrawer and Fragments
  - Persistence in the local database
  - REST API for enabling users to see their data on different devices

Course Admin

- No labs this week!

Third Party Libraries

- Android allows easy integration via implementation command
- Libraries for:
  - Improved UI: Butterknife, MPAndroidChart
  - ORM data access: OrmLite, GreenDAO, Room
  - Easier networking: OkHttp
  - Interacting with backend: Parse (Back4App), Retrofit
  - Innovative data structures: Guava
  - …
- Browse https://android-arsenal.com/

Butterknife

- View-binding library for Android
  - Avoid all those findViewById calls
- Annotations for binding
  - Views:
  - Resources:
  - Listeners:

```
@BindView(R.id.username) EditText edtUsername;
```

```
@BindDrawable(R.drawable.graphic)
Drawable graphic;
```

```
@OnClick(R.id.submit)
public void submit(View view) {
    // TODO submit data to server...
}
```

- Behind the scenes it uses the annotations to create a new class

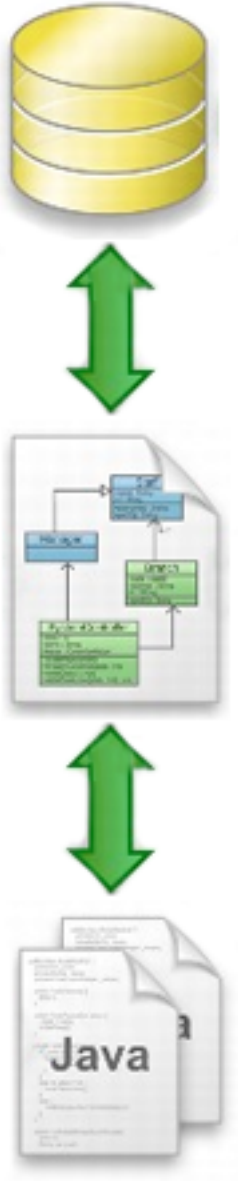
Butter Knife Example

Object-relational Mapping ORM

- Problem:
  - Object-oriented languages work with objects that can be relatively complex
  - (Relational) databases store and manipulate simple scalar values in tables
  - Converting objects to table entries is cumbersome and prone to errors
- Solution
  - Object-Relational Mapping (ORM)

OrmLite

- Data storage
  - A number of relational DBs (MS-SQL, MySQL, Android SQLite)
- Object files
  - Annotated Java models
- Data Access Object (DAO)
  - Interface between the database and Java objects
- Note: this is not another database, but a layer over your SQLite DB!



OrmLite Models

- Use annotations to mark classes that will be persisted

```
@DatabaseTable(tableName = "accounts")
public class Account {

    @DatabaseField(id = true)
    private String name;

    @DatabaseField(canBeNull = false)
    private String password;
    ...
    Account() { // all persisted classes must
      // define a no-arg constructor with at least
      // package visibility
    }
    ...
}
```

OrmLite Models - Annotations

- OrmLite annotations
  - @DatabaseTable: for each Java class you wish to persist
    - **tableName** argument specifies the name of the table that corresponds to the class
  - @DatabaseField: for each field in the class that you wish to persist
    - **columnName** (default: field name normalized)
    - **defaultValue** (default: null)
    - **canBeNull** (default: true)
    - **persisted** (default: true)
    - **unique** (default: false)

OrmLite Models - Annotations

- Managing relations
  - @DatabaseField:
    - **id** (default: false) indicates whether the field is an ID
    - **generatedId** (default: false) tells the database to auto-generate a corresponding id for every row inserted
    - **foreign** (default: false) identifies this field as corresponding to another class that is also stored in the database. The field must not be a primitive type
    - **foreignAutoRefresh** (default: false) automatically refresh the foreign field
    - **foreignAutoCreate** (default: false) automatically create a foreign field (in its table)
    - …

ID is always unique!

OrmLite Models - Annotations

- Managing relations
  - @ForeignCollectionField:
    - Enables one to many relationships
    - **eager** (default: false) a separate query is made immediately and the foreign key items are stored as a list within the collection; otherwise – lazy – accessed only when a method is called on the collection

```
public class Account {
    @ForeignCollectionField(eager = false)
    ForeignCollection<Order> orders;
    …
}
```

OrmLite Models – Data Types

- Persisted data types
  - Standard/Primitive:
    - String, int/Integer, long/Long, float/Float, double/Double, etc.
  - Date/Time:
    - java.util.Date, DateTime, java.sql.Date, java.sql.Timestamp
  - Serializable:
    - You must explicitly set the field type

```
// image is an object that implements Serializable
@DatabaseField(dataType = DataType.SERIALIZABLE)
Image image;
```

OrmLite – Android

- OrmLiteSqliteOpenHelper
  - Extend this class to create and upgrade the database when your application is installed and provide the DAO classes used by your other classes
- OpenHelperManager
  - To manage Helper usage

```
private DatabaseHelper databaseHelper =
null;

@Override
protected void onDestroy() {
    super.onDestroy();
    if (databaseHelper != null) {
        OpenHelperManager.releaseHelper();
        databaseHelper = null;
    }
}

private DBHelper getHelper() {
    if (databaseHelper == null) {
        databaseHelper =

OpenHelperManager.getHelper(this,
DatabaseHelper.class);
    }
    return databaseHelper;
}
```

OrmLite – Android

- OrmLiteConfigUtil
  - Creates a configuration for your database in res/raw/ormlite_config.txt
  - Run it on your local machine, the file is shipped with your application (resource file)

```
public class DatabaseConfigUtil extends OrmLiteConfigUtil {
  private static final Class<?>[] classes = new Class[] {
    SimpleData.class,
  };
  public static void main(String[] args) throws Exception {
    writeConfigFile("ormlite_config.txt", classes);
  }
}
```

OrmLite – Data Access

- Via DAO object

```
Dao<Workout, Long> workoutDao = null;
DatabaseHelper databaseHelper = OpenHelperManager.getHelper(context,
DatabaseHelper.class);
try {
    workoutDao = databaseHelper.workoutDao();
} catch (SQLException e) {
    e.printStackTrace();
}
```

  - Query

```
try {
    QueryBuilder<Workout, Long> workoutBuilder = workoutDao.queryBuilder();
    Where where = workoutBuilder.where();
    where.eq(Workout.colStatus, 1);
    return workoutBuilder.query();
} catch (SQLException e) {
    e.printStackTrace();
}
```

OrmLite – Data Access

- Via DAO object

```
Dao<Workout, Long> workoutDao = null;
DatabaseHelper databaseHelper = OpenHelperManager.getHelper(context,
DatabaseHelper.class);
try {
    workoutDao = databaseHelper.workoutDao();
} catch (SQLException e) {
    e.printStackTrace();
}
```

  - Insert

```
workout = new Workout("Workout", sportActivity);
workout.setUser(user);
try {
    workoutDao.create(workout);
    workout.setTitle("Workout " + Long.toString(getId()));
    // update name
    workoutDao.update(workout);
} catch (SQLException e) {
    e.printStackTrace();
}
```

OrmLite in Android Example

Other ORM Options

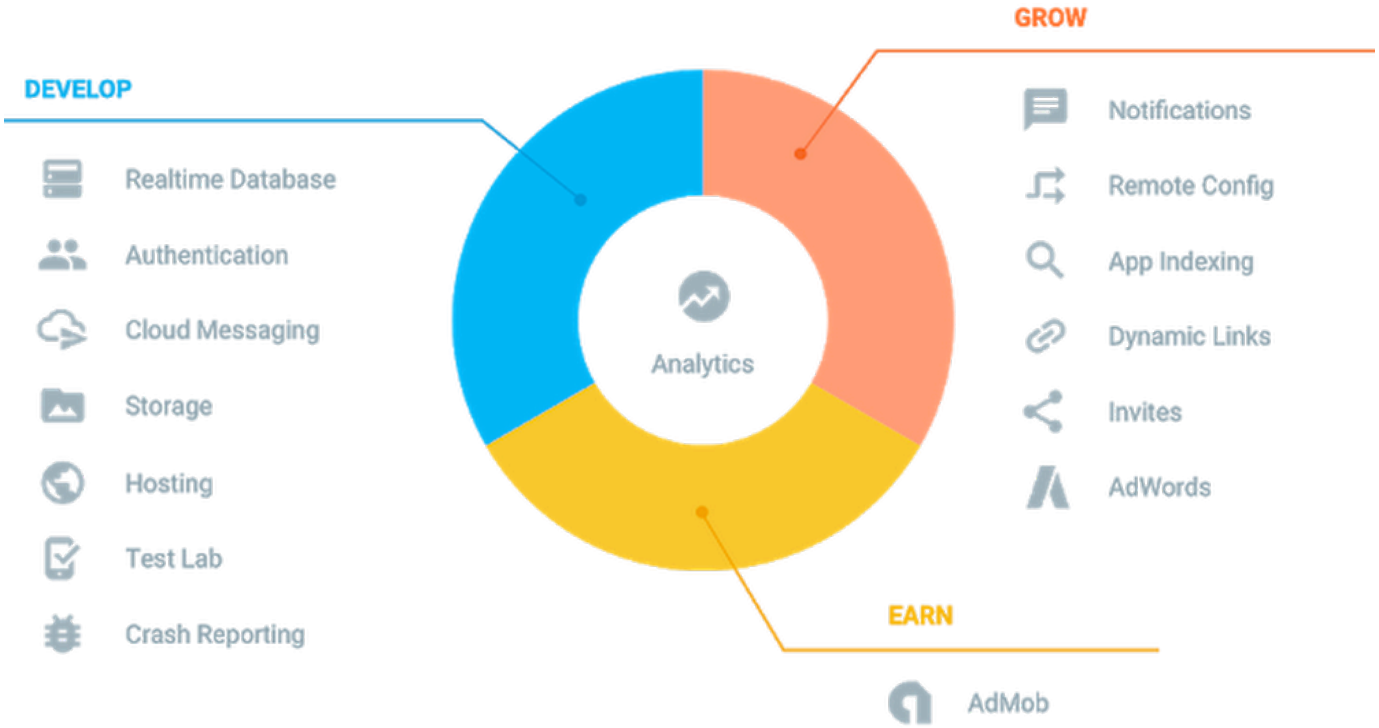
- Room (part of AndroidJetpack)
  - Pros:
    - Optimised to work with recent Android components
    - LiveData – notified when data is changed
    - Does not allow main thread execution
    - SQL queries checked at runtime
  - Cons:
    - Fewer Java methods for querying (compared to OrmLite)
    - Different data types than OrmLite
    - Supports only Android SQLite
- GreenDao
- SugarORM

Backend for Mobile Apps

- Android (or iOS for that matter) do not lock you into a particular backend technology
  - PHP, Node.js, Java Web apps, etc.
  - AWS, Google Cloud Platform, etc.
- Some solution easier to work with than others
  - Firebase
  - Parse Server (Back4App)

Firebase

- Mobile and Web app development platform supported by Google



Firebase

- Mobile and Web app development platform supported by Google
- Great for:
  - Authentication with Google ID (you have to use it)
  - Notifications (think chat-like app)
  - Crashlytics
  - Machine learning support

```
implementation 'com.google.firebase:firebase-core:16.0.8'
```

Parse Server

- Open source backend as a service (BaaS) platform initially developed by Facebook
  - Back4App is a Parse Server hosting platform
- Great for:
  - Building different REST APIs
  - Cron Jobs – schedule server jobs
  - User management (auto emails, social login)
  - Multiple SDKs
    - Including for Android

Back4App

- NoSQL database
- REST API to access data
- Access via HTTP using a number of languages/ platforms
- Different pricing tiers, but the free one is sufficient for prototyping
- Android library

```
implementation "com.github.parse-community.Parse-SDK-Android:parse:1.18.5"
```

Back4App – Create Backend

- Go to back4app.com, log in, and create a new application
  - Manage via a dashboard
  - Add collections (tables)
  - Add custom code
  - Initiate communication (notifications)
- Get the following (and put in your Android app) in order to access the backend:
  - Application ID
  - Client Key

Back4App Example

- At https://bitbucket.org/veljkop/parseexample/

Google Sign In Example

- At https://bitbucket.org/veljkop/googlesigninexample/

Platform-Based Development: Android Programming – Communication

BS UNI studies, Spring 2018/2019

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

Wireless Connectivity

- Smartphone wireless interfaces
  - GSM, 3G, LTE, 5G
  - WiFi
  - Bluetooth
  - NFC
- Understanding tradeoffs

Selecting Connectivity Mode

- NFC
  - Very low power (tags don't even need to be powered)
  - Very short range (~10cm)
  - Low throughput (~400 kbps)
  - Security tags, location-based services
- Bluetooth
  - Low power (~10mW)
  - Short range (~10m)
  - Low throughput (~1 Mbps)
  - Connection with peripherals, wearables (smartwatch)

Selecting Connectivity Mode

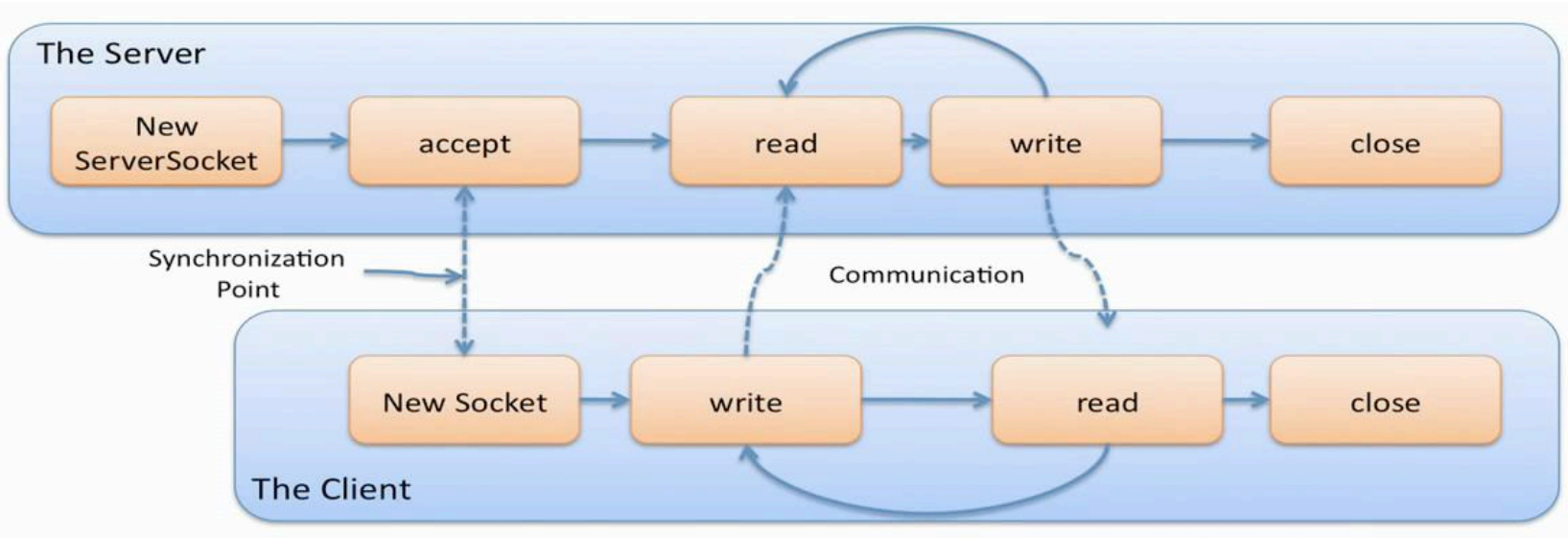
- WiFi
  - Medium power consumption (~100mW)
  - Low range (~100m)
  - High throughput (~100 Mbps)
  - Large downloads, system updates; WiFi is usually unmetered network
- Cellular network (GSM, 3G, LTE, 5G)
  - High power (~200mW)
  - Long range (~1000m)
  - Varying throughput (up to ~1 Gbps with 5G)
  - Ubiquitous connectivity

Android Networking Support

- NfcManager
- BluetoothManager
  - Use it to get BluetoothAdapter
- WifiManager
  - WiFi-specific operations: scans, (dis)associate, etc.
- TelephonyManager
  - Cellular-specific operations: scans, get subscriber and network information
- ConnectivityManager
  - Monitors network connections, manages failovers, notifies when connectivity changes

Networking Abstractions

- You can run any protocol over Android networking stack
- Sockets
  - Standard Java sockets: Socket (TCP) and DatagramSocket (UDP)



Networking Abstractions

- Http(s)URLConnection
  - Connecting to a URL using HTTP(S) protocol
  - Communicate with a REST API
  - Supports secure communication via Transport Layer Security (TLS)
- OkHttp (third-party library)
  - Advanced HTTP Client
  - Pools connections, compresses data, caches content, recovers from network problems, etc.

Networking Abstractions

- Retrofit
  - REST Client for Android
  - Define a model
  - Define possible HTTP operations
  - Define adapter and converter
- Volley
  - Somewhere in between Retrofit and OkHttp
- Glide
  - Media fetching and decoding

Simple Networking Best Practices

- Security
  - Use encrypted communication – HttpsURLConnection
- Run network operations on a separate thread
  - AsyncTask doInBackground for networking requests
  - AsyncTask onPostExecute to process the result
- Data conversion and handling
  - InputStream converted to a target data type (e.g. string, image, whatever you are downloading)
  - Callback interface to report the results to the UI

Http Client Example

https://bitbucket.org/veljkop/httpclientexample

Issues with HttpsUrlConnection + AsyncTask

- Multiple requests are serviced in the First In – First Out fashion
  - But some requests are more important than others!
- Applications often repeatedly issue identical requests
  - Cache!
- Request data can be large
  - Compress!
- AsyncTask can lead to memory leaks

OkHttp Example

https://bitbucket.org/veljkop/okhttpexample