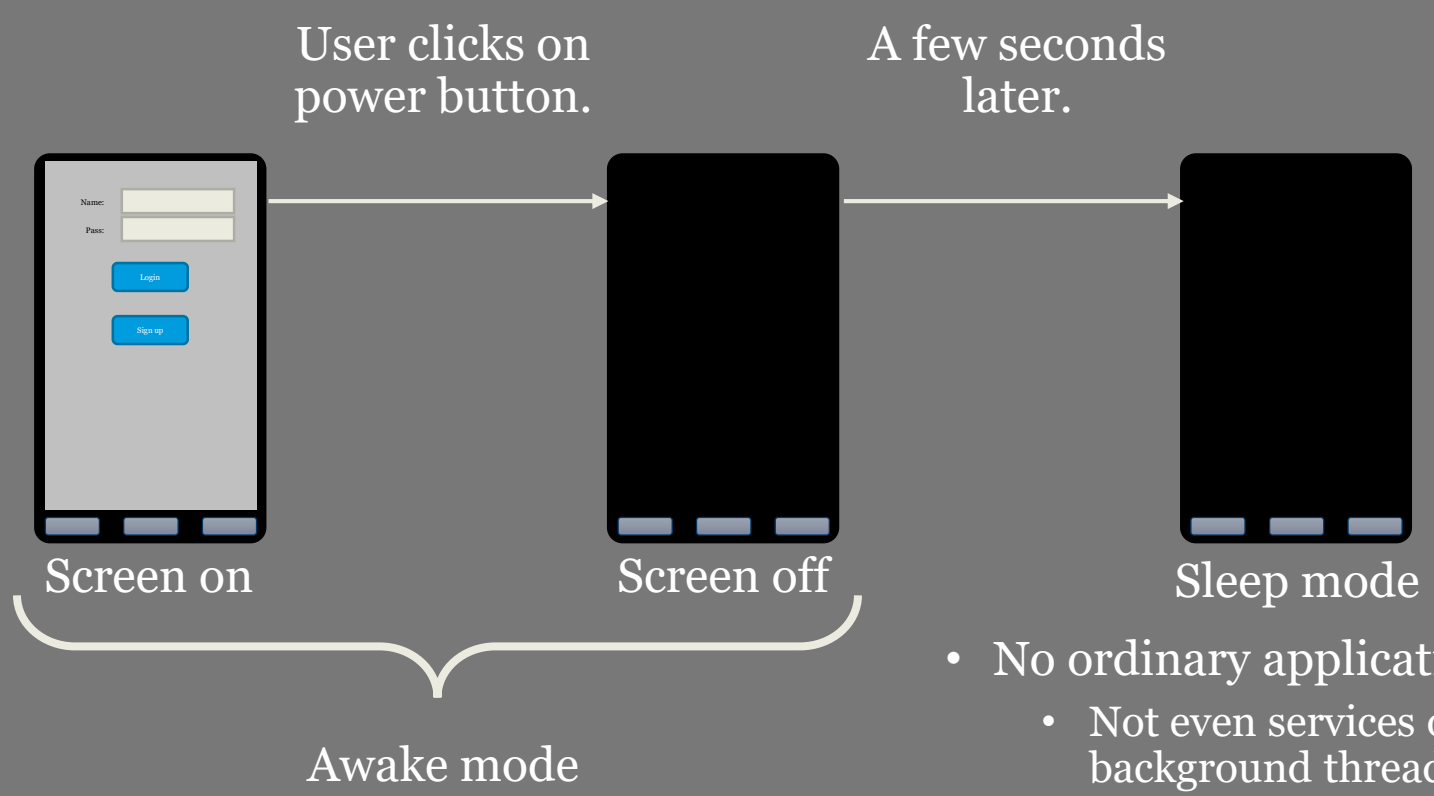
JÖNKÖPING UNIVERSITY

School of Engineering

ANDROID SCHEDULING

Peter Larsson-Green
Jönköping University
Spring 2020

SAVING BATTERY



- No ordinary applications run.
  - Not even services or background threads.
  - Phone calls & SMS messages can wake up the device
  - Alarms can wake up the device.

TIME IN ANDROID

Can be changed by the user!

Real-time clock

- Elapsed time since the Unix epoch (1 January 1970).

```
long now = Calendar.getInstance().getTimeInMillis();
```

Is resetting on reboot!

Elapsed real-time

- Elapsed time since the device booted.

```
long now = SystemClock.elapsedRealtime();
```

ONE-SHOT ALARMS

```
AlarmManager alarmManager = (AlarmManager)
            aContext.getSystemService(Context.ALARM_SERVICE);
```

```
alarmManager.set(
  AlarmManager.XXX,
  SystemClock.elapsedRealtime() + 5*1000,
  thePendingIntent
);
```

```
ELAPSED_REALTIME
ELAPSED_REALTIME_WAKEUP
RTC
RTC_WAKEUP
```

ONE-SHOT ALARMS

```
alarmManager.set(...);
```

Up to Android 4.3: is exact.

From Android 4.4 and on: is inexact

(the time is minimum).

```
alarmManager.setExact(...);
```

Added in Android 4.4.

REPEATING ALARMS

```
AlarmManager alarmManager = (AlarmManager)
            aContext.getSystemService(Context.ALARM_SERVICE);
```

```
alarmManager.setRepeating(
   AlarmManager.ELAPSED_REALTIME_WAKEUP,
   SystemClock.elapsedRealtime() + 5*1000,
   1000*60,
   thePendingIntent
);
```

First time.

Interval.
At least one minute from Android 5.1 and on.

REPEATING ALARMS

```
AlarmManager alarmManager = (AlarmManager)
            aContext.getSystemService(Context.ALARM_SERVICE);
```

```
alarmManager.setInexactRepeating(
   AlarmManager.ELAPSED_REALTIME_WAKEUP,
   SystemClock.elapsedRealtime() + 5*1000,
   AlarmManager.INTERVAL_XXX,
   thePendingIntent
);
```

Min first time.

```
INTERVAL_FIFTEN_MINUTES
INTERVAL_HALF_HOUR
INTERVAL_HOUR
INTERVAL_HALF_DAY
INTERVAL_DAY
```

REPEATING ALARMS

```
alarmManager.setRepeating();
```

Up to Android 4.3: is exact.

From Android 4.4 and on:
is identical to
`setInexactRepeating`.

```
alarmManager.setInexactRepeating();
```

Is inexact.

CANCELING ALARMS

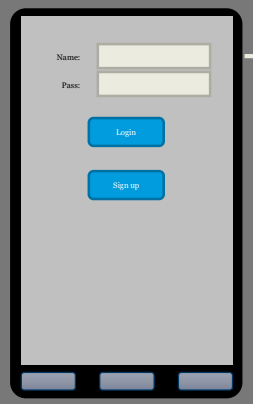
```
alarmManager.cancel(theAlarmIntent);
```

SAVING BATTERY

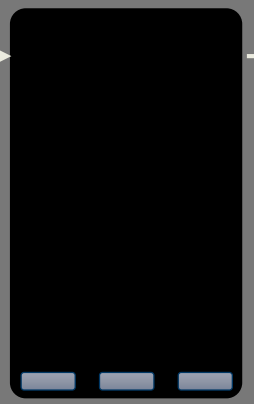
User clicks on power button.

A few seconds later.

~Device been still for a while and not charging.

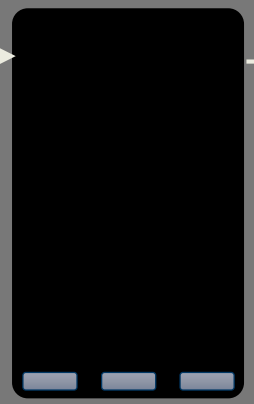


Screen on



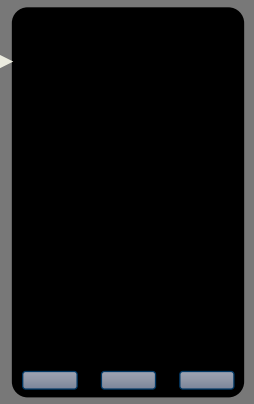
Screen off

Awake mode



Sleep mode

- No ordinary applications run.
  - Not even services or background threads.
  - Phone calls & SMS messages can wake up the device
  - Alarms can wake up the device.



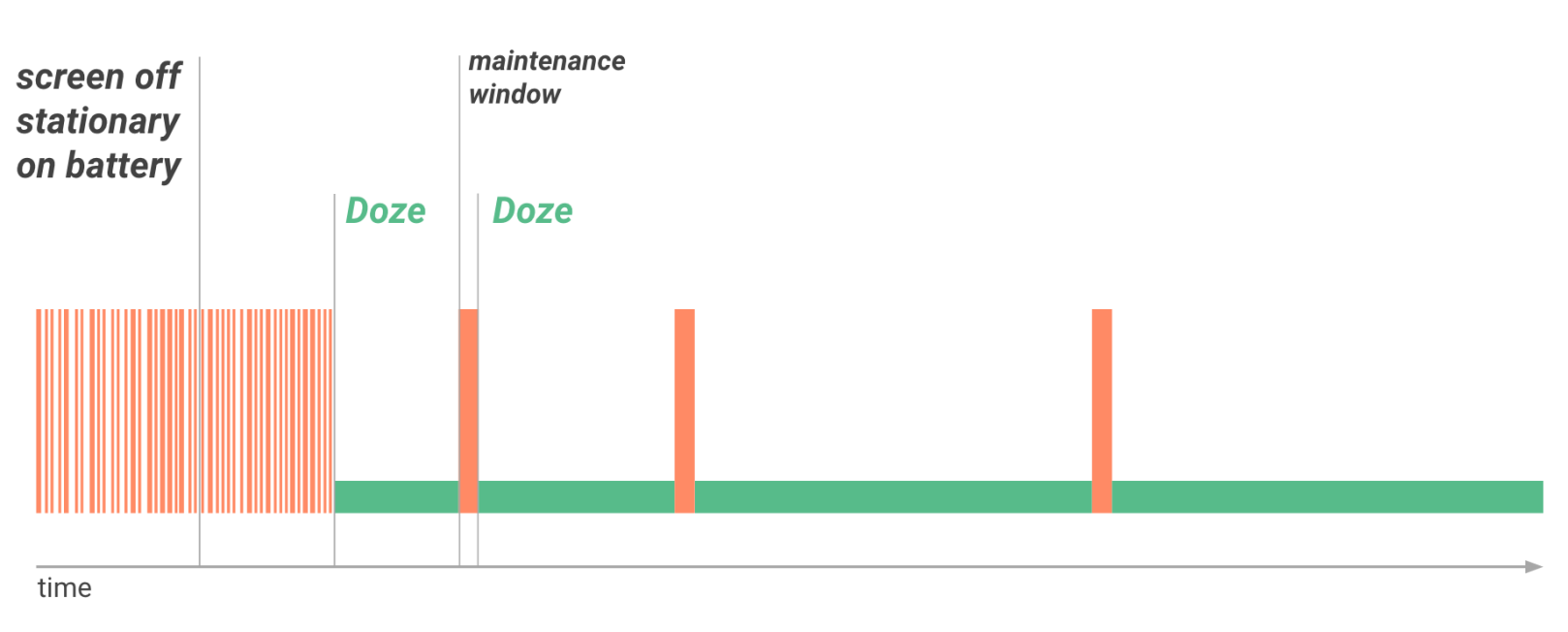
Doze mode

- Introduced in Android 6.
- Scheduled alarms won't run as scheduled.

HANDLING DOZE MODE

Previously mentioned alarms are never fired in Doze mode.

- The device will leave Doze mode every now and then to fire these.



https://developer.android.com/training/monitoring-device-state/doze-standby.html

HANDLING DOZE MODE

Want an alarm to go off in Doze mode?

- Use `setAndAllowWhileIdle(...)` (from API level 23)
- Use `setExactAndAllowWhileIdle(...)` (from API level 23)

But these may at most fire 1 alarm each 9 minutes.

And must complete within a few seconds or obtain an awake lock.

- Requires the permission `android.permission.WAKE_LOCK`.

Another option:

- Use `setAlarmClock(...)` (from API level 21)

JOB SCHEDULER

A better version of the Alarm Manager.

- Introduced in Android 5.0.
- Improvements:
  - Scheduled alarms can survive reboots.
  - Job scheduler alarms can keep the device awake.
  - Only fire off alarm under certain conditions (e.g. internet access).
- Implemented as a service.

CREATING THE JOB SERVICE

```
public class MyJobService extends JobService{
  @Override
  public boolean onStartJob(JobParameters params){
     return false;
  }
  @Override
  public boolean onStopJob(JobParameters params){
     return false;
  }
}
```

CREATING THE JOB SERVICE

```
<manifest package="the.package" ...>
  <application ...>
    <service
      android:name=".MyJobService"
      android:permission="android.permission.BIND_JOB_SERVICE"
    />
  </application>
</manifest>
```

Never granted to ordinary apps.

CREATING A JOB

```
int jobId = 31;
ComponentName jobService = new ComponentName(aContext,
                                             MyJobService.class);
JobInfo.Builder builder = new JobInfo.Builder(jobId, jobService);
builder.setPersisted(true);
builder.setRequiresCharging(true);
builder.setMinimumLatency(1000*60);
// builder.setPeriodic(1000*60);
builder.setExtras(new PersistableBundle());
builder.setRequiredNetworkType(JobInfo.XXX);
JobInfo theJob = builder.build();
```

Requires `RECEIVE_BOOT_COMPLETED` permission.

```
NETWORK_TYPE_NONE
NETWORK_TYPE_ANY
NETWORK_TYPE_UNMETERED
```

SCHEDULING JOBS

```
JobScheduler jobScheduler = (JobScheduler)
        aContext.getSystemService(Context.JOB_SCHEDULER_SERVICE);
jobScheduler.schedule(theJob);
```

Canceling a job:

```
jobScheduler.cancel(jobId);
```

IMPLEMENTING THE JOB SERVICE

```
public boolean onStartJob(JobParameters params){
  int jobId = params.getJobId();
  PersistableBundle extras = params.getExtras();
  return false;
}
```

We are done (no background thread is running).

If you return `true`:

```
boolean needsRescheduled = false;
theJobScheduler.jobFinished(params, needsRescheduled);
```

THE WORKMANAGER

Many changes, staying backward compatible is hard ☹

`WorkManager` part of the support library:

- Use `AlarmManager` for exact timing.
- Use `WorkManager` (or `JobScheduler`) otherwise.