.nl. jug

J-Fall

31 oktober 2012 - Hart van Holland

Android & OSGi

Angelo van der Sijpt

luminis
Conversing worlds

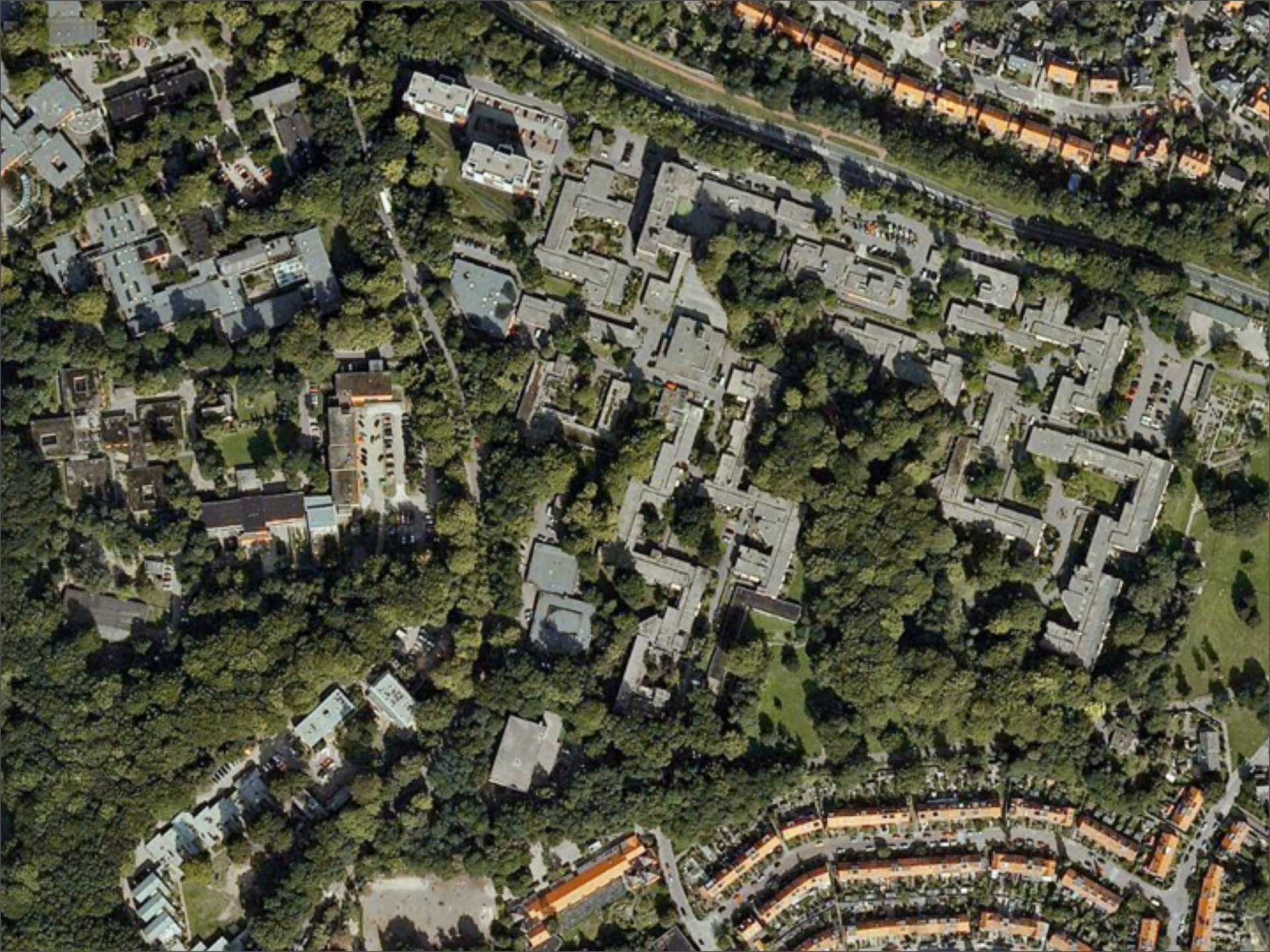
http://www.flickr.com/photos/politikill/3478232525/

$ whoami

- Angelo van der Sijpt
- Software engineer at Luminis Arnhem
- Committer for Apache ACE
- Buzzwords: Java, OSGi, Agile

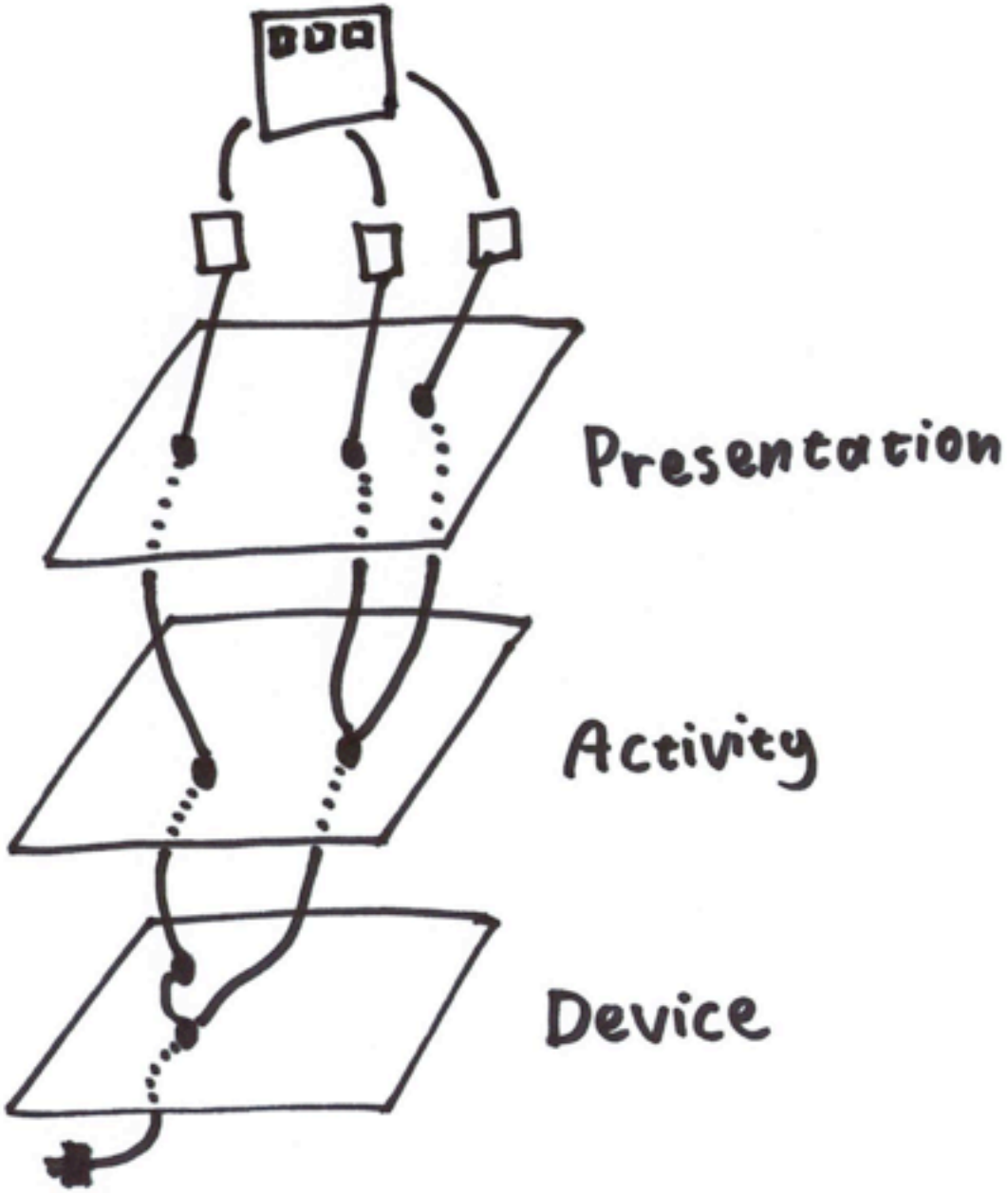
luminis
Conversing worlds

http://retro.nrc.nl/W2/Lab/Profiel/Charitas/televisie.html




rtd
...maakt méér mogelijk


Hoofdscherm
Instellingen
Lamp
Lamp
terug
Verlichti
Klimaat
Deuren
Telefoon
Lift
Stapsnel heid: 0.7 sec
Sneller
Langza
0.5 sec
0.7 sec
TV
Radio
Rolstoel
Fokus
Spraak: uit
Aan
Uit
Instellin
KAKU test on
KAKU test off
Lamp
Geluiden: aan
Aan
Uit
11:03
Hoofdscherm
Licht
Verlichting
Terug
Woonkamer 1
Woonkamer 2
Woonkamer 3
Slaapkamer 1
Slaapkamer 2
Hal
11:03

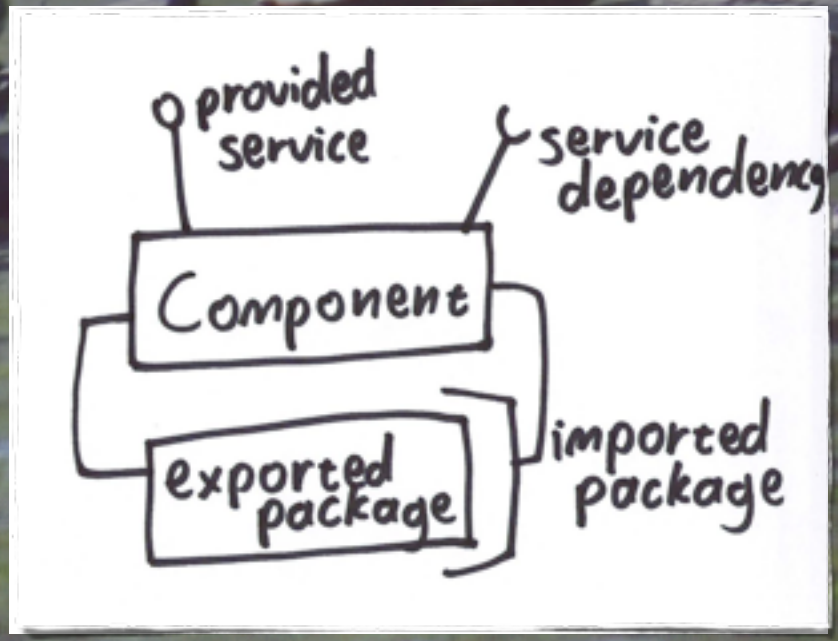

Presentation
Activity
Device

http://www.flickr.com/photos/keturn/4599069063/

http://www.flickr.com/photos/keturn/4599069063/

http://www.flickr.com/photos/matthileo/6912691494/

Services all the way down



I use the Apache Felix Dependency manager for this

http://www.flickr.com/photos/matthileo/6912691494/

Android & OSGi

- Android ~= Java

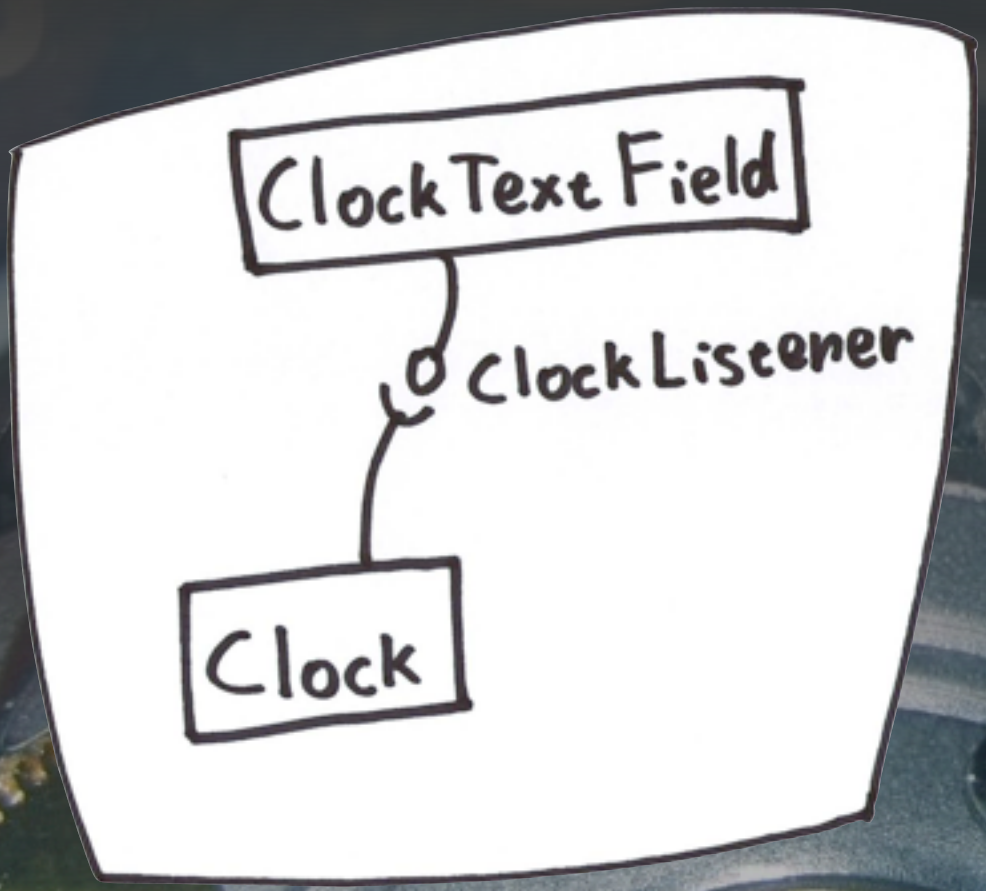
- https://github.com/angelos/AndroidAndOsgi

- Eclipse
- Android tooling
- BNDTools

Basic Framework

- Android project
- Felix Framework in `libs` directory
- Build framework in `onCreate`

```
// BasicFramework.java

// Create and start a framework
Map<String, Object> config = new HashMap<String, Object>();
String cache = "/data/data/" +
    getApplicationContext().getPackageName() + "/cache";
config.put(Constants.FRAMEWORK_STORAGE, cache);
Felix felix = new Felix(config);
felix.start();

// Get to the bundle context
BundleContext context = felix.getBundleContext();

// Print something that shows we have a framework
Log.d("JFall2012",
       context.getProperty(Constants.FRAMEWORK_VENDOR)
       );
```



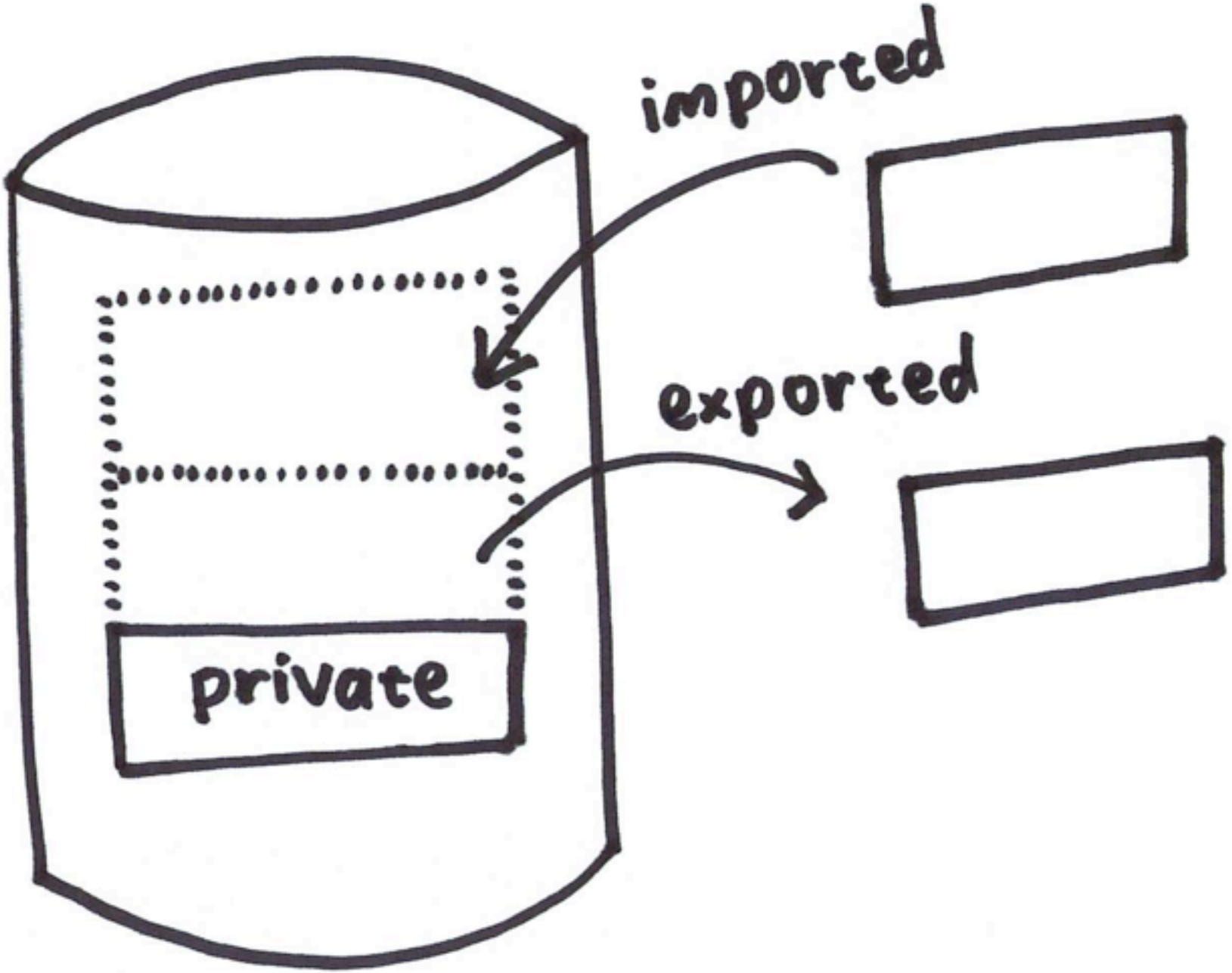

```
public interface ClockListener {
    public void tick(String time);
}
```

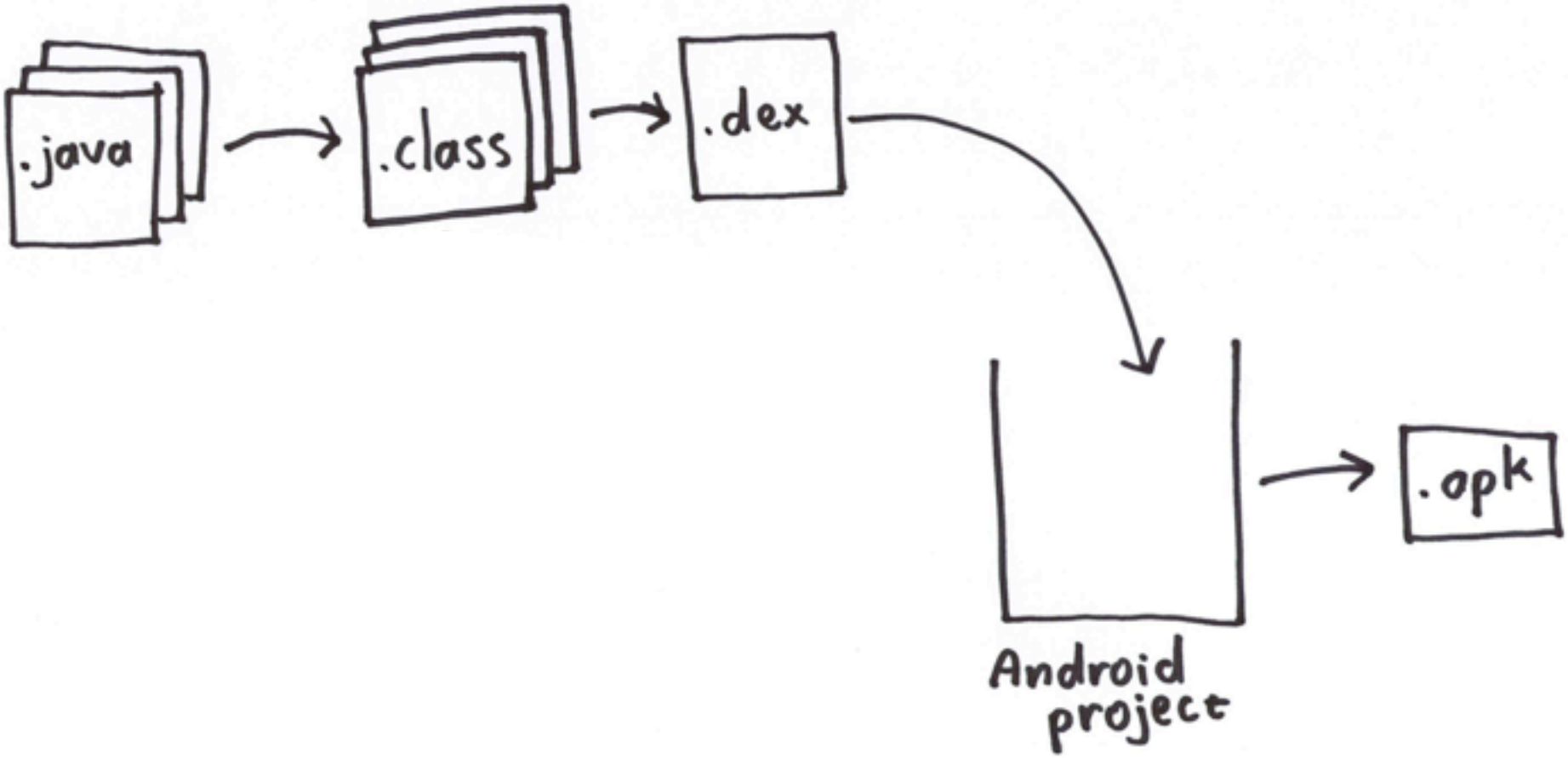
Set up project

- BNDTools projects for bundles
- Create bundle: export interface

- Get bundles into Android project
- Use the view, and register it

http://www.flickr.com/photos/bdesham/2432400623/

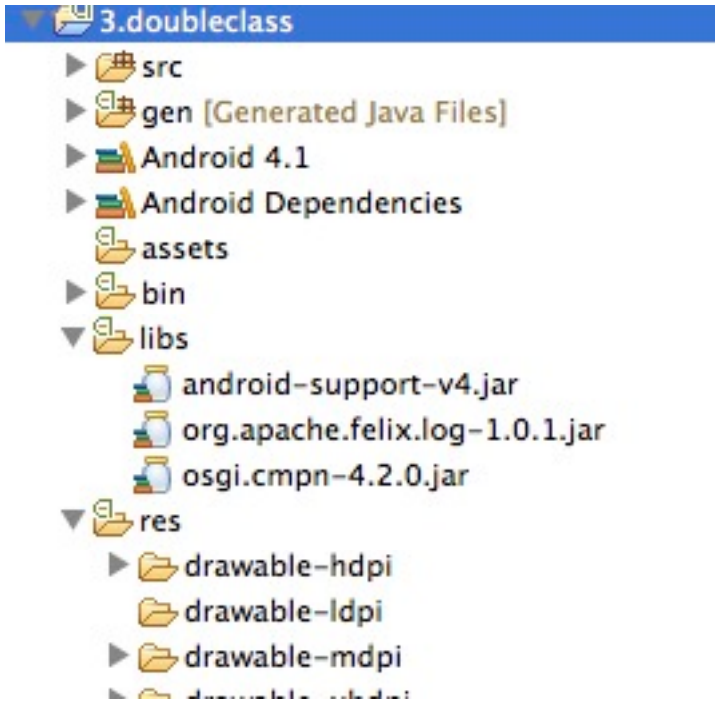

imported
exported
private

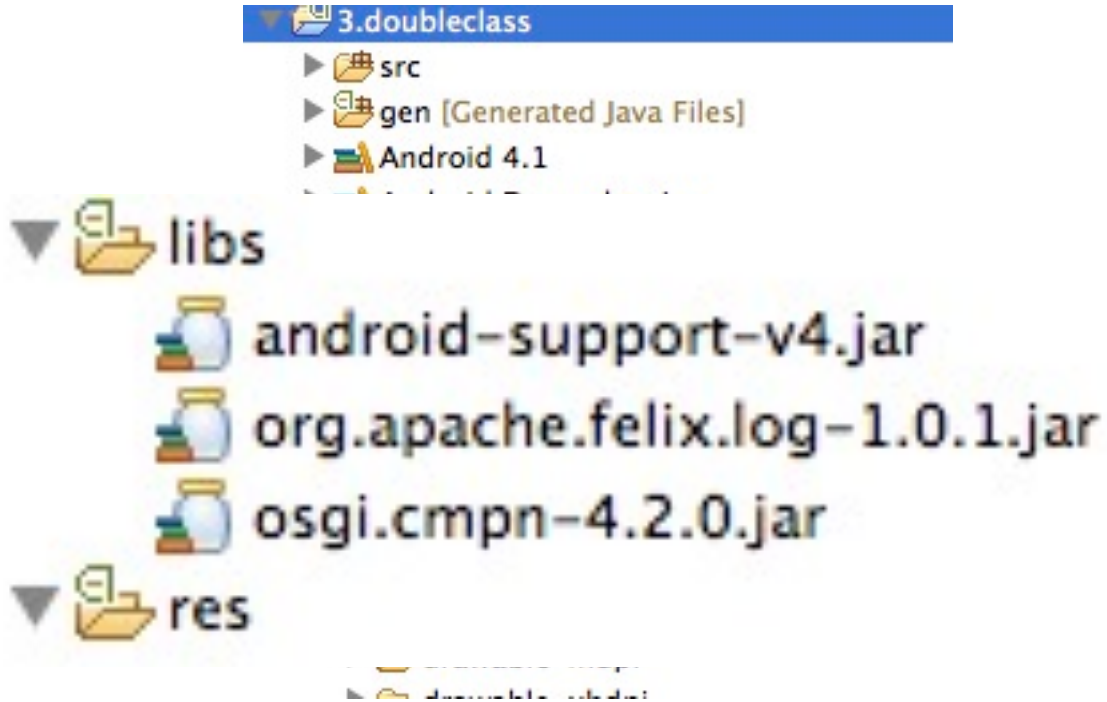

.java
.class
.dex
Android project
.apk

Duplicated classes on the classpath?

Compendium

Felix LogService

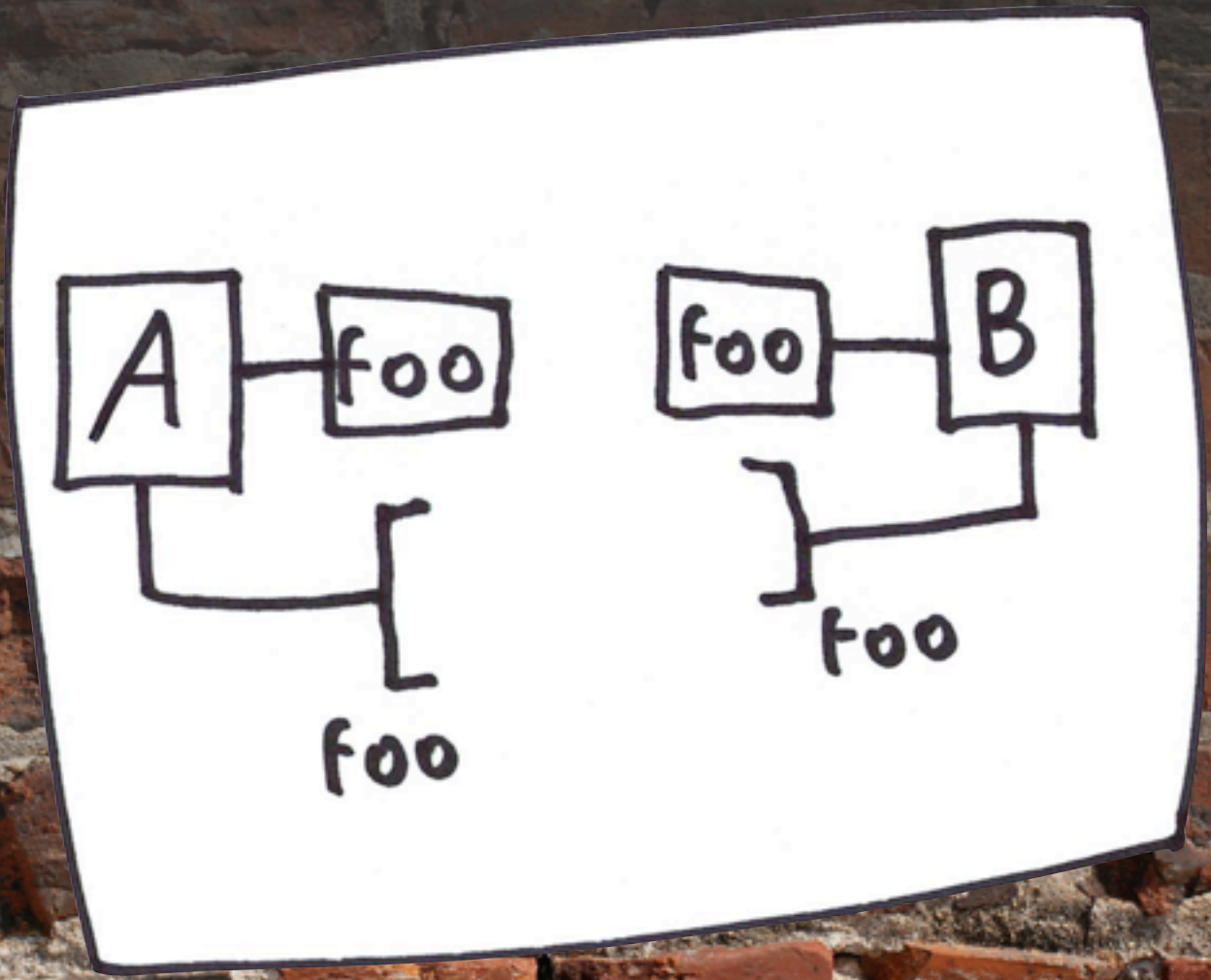

3.doubleclass
src
gen [Generated Java Files]
Android 4.1
Android Dependencies
assets
bin
libs
android-support-v4.jar
org.apache.felix.log-1.0.1.jar
osgi.cmpn-4.2.0.jar
res
drawable-hdpi
drawable-ldpi
drawable-mdpi


3.doubleclass
src
gen [Generated Java Files]
Android 4.1
libs
android-support-v4.jar
org.apache.felix.log-1.0.1.jar
osgi.cmpn-4.2.0.jar
res

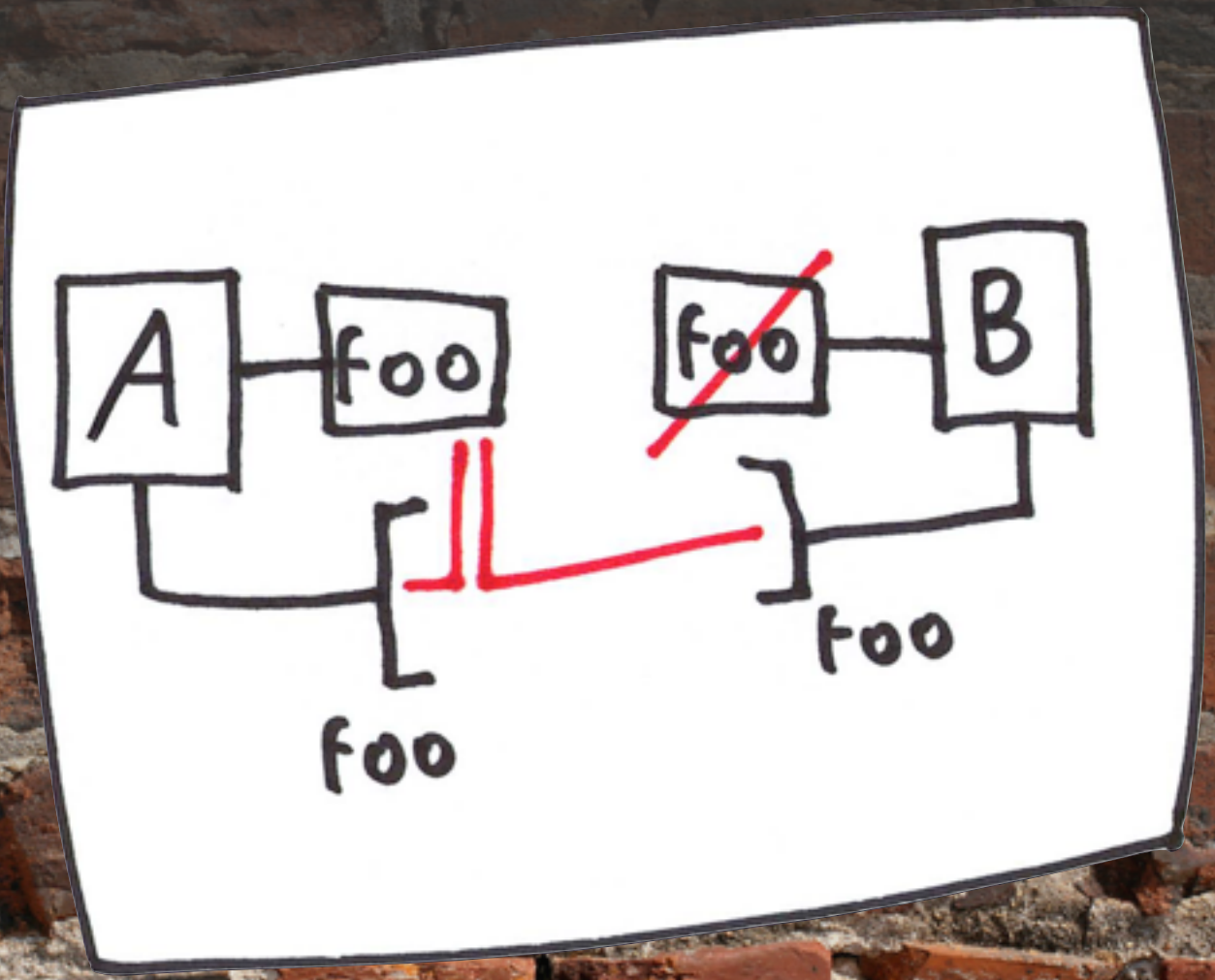
```
[2012-10-30 21:29:21 - 3.doubleclass] Dx
UNEXPECTED TOP-LEVEL EXCEPTION:
java.lang.IllegalArgumentException: already added: Lorg/
osgi/service/log/LogEntry;
[2012-10-30 21:29:21 - 3.doubleclass] Dx    at
com.android.dx.dex.file.ClassDefsSection.add(ClassDefsSecti
on.java:123)
[2012-10-30 21:29:21 - 3.doubleclass] Dx    at
com.android.dx.dex.file.DexFile.add(DexFile.java:163)
[2012-10-30 21:29:21 - 3.doubleclass] Dx    at
com.android.dx.command.dexer.Main.processClass(Main.java:
486)

...

[2012-10-30 21:29:21 - 3.doubleclass] Dx 1 error; aborting
[2012-10-30 21:29:21 - 3.doubleclass] Conversion to Dalvik
format failed with error 1
```

```
[2012-10-30 21:29:21 - 3.doubleclass] Dx
UNEXPECTED TOP-LEVEL EXCEPTION:
java.lang.IllegalArgumentException: already added: Lorg/
osgi/service/log/LogEntry;
[2012-10-30 21:29:21 - 3.doubleclass] Dx    at
com.android.dx.dex.file.ClassDefsSection.add(ClassDefsSecti
on.java:123)
[2012-10-30 21:29:21 - 3.doubleclass] Dx    at
com.android.dx.dex.file.DexFile.add(DexFile.java:163)
[2012-10-30 21:29:21 - 3.doubleclass] Dx    at
com.android.dx.command.dexer.Main.processClass(Main.java:
486)

...

[2012-10-30 21:29:21 - 3.doubleclass] Dx 1 error; aborting
[2012-10-30 21:29:21 - 3.doubleclass] Conversion to Dalvik
format failed with error 1
```


felix.getBundleContext().installBundle("my.jar",
getAssets().open("raw/my.jar"));
http://www.flickr.com/photos/grungetextures/4107730906/



http://www.flickr.com/photos/grungetextures/4107730906/



http://www.flickr.com/photos/grungetextures/4107730906/

Dalvik

- Optimized for ARM processors
- Register based i.s.o. stack based
- Uses pre-optimization
  - Compile time
  - Install time
  - Run time

38

This is Ben, one of the engineers working on the JIT @ Google. When Bill and I started on this project, the goal was to deliver a working JIT as soon as possible with minimal impact to resource contention (eg memory footprint, CPU hijacked by the compiler thread) so that it can run on low-end devices as well. Therefore we used a very primitive trace based model. That is, the compilation entity passed to the JIT compiler is a basic block, sometimes as short as a single instruction. Such traces will be stitched together at runtime through a technique called chaining so that the interpreter and code cache lookup won't be invoked often. To some degree the major source of speedup comes from eliminating the repeated interpreter parsing overhead on frequently executed code paths.

That said, we do have quite a few local optimizations implemented with the Froyo JIT:

- Register allocation (8 registers for v5te target since the JIT produces Thumb code / 16 registers for v7)
- Scheduling (eg redundant ld/st elimination for Dalvik registers, load hoisting, store sinking)
- Redundant null check elimination (if such redundancy can be found in a basic block).
- Loop formation and optimization for simple counted loops (ie no side-exit in the loop body). For such loops, array accesses based on extended induction variables are optimized so that null and range checks are only performed in the loop prologue.
- One entry inline cache per virtual callsite w/ dynamic patching at runtime.
- Peephole optimizations like power-reduction on literal operands for mul/div.

In Gingerbread we added simple inlining for getters/setters. Since the underlying JIT frontend is still simple trace based, if the callee has branches in there it won't be inlined. But the inline cache mechanism is implemented so that virtual getters/setters can be inlined without problems.

We are currently working on enlarging the compilation scope beyond a simple trace so that the compiler has a larger window for code analysis and optimization. Stay tuned.

answered **Feb 8 '11 at 7:20**

Ben
396 ● 2 ● 2

38

This is Ben, one of the engineers working on the JIT @ Google. When Bill and I started on this project, the goal was to deliver a working JIT as soon as possible with minimal impact to resource contention (eg memory footprint, CPU hijacked by the compiler thread) so that it can run on low-end devices as well. Therefore we used a very primitive trace based model. That is, the compilation entity passed to the JIT compiler is a basic block, sometimes as short as a single instruction. Such traces will be stitched together at runtime through a

That said, we do have quite a few local optimizations implemented wit

- Register allocation (8 registers for v5te target since the JIT produ
- Scheduling (eg redundant ld/st elimination for Dalvik registers, loa
- Redundant null check elimination (if such redundancy can be fou
- Loop formation and optimization for simple counted loops (ie no s loops, array accesses based on extended induction variables are checks are only performed in the loop prologue.
- One entry inline cache per virtual callsite w/ dynamic patching at
- Peephole optimizations like power-reduction on literal operands f

larger window for code analysis and optimization. Stay tuned.

share | edit | flag

answered Feb 8 '11 at 7:20

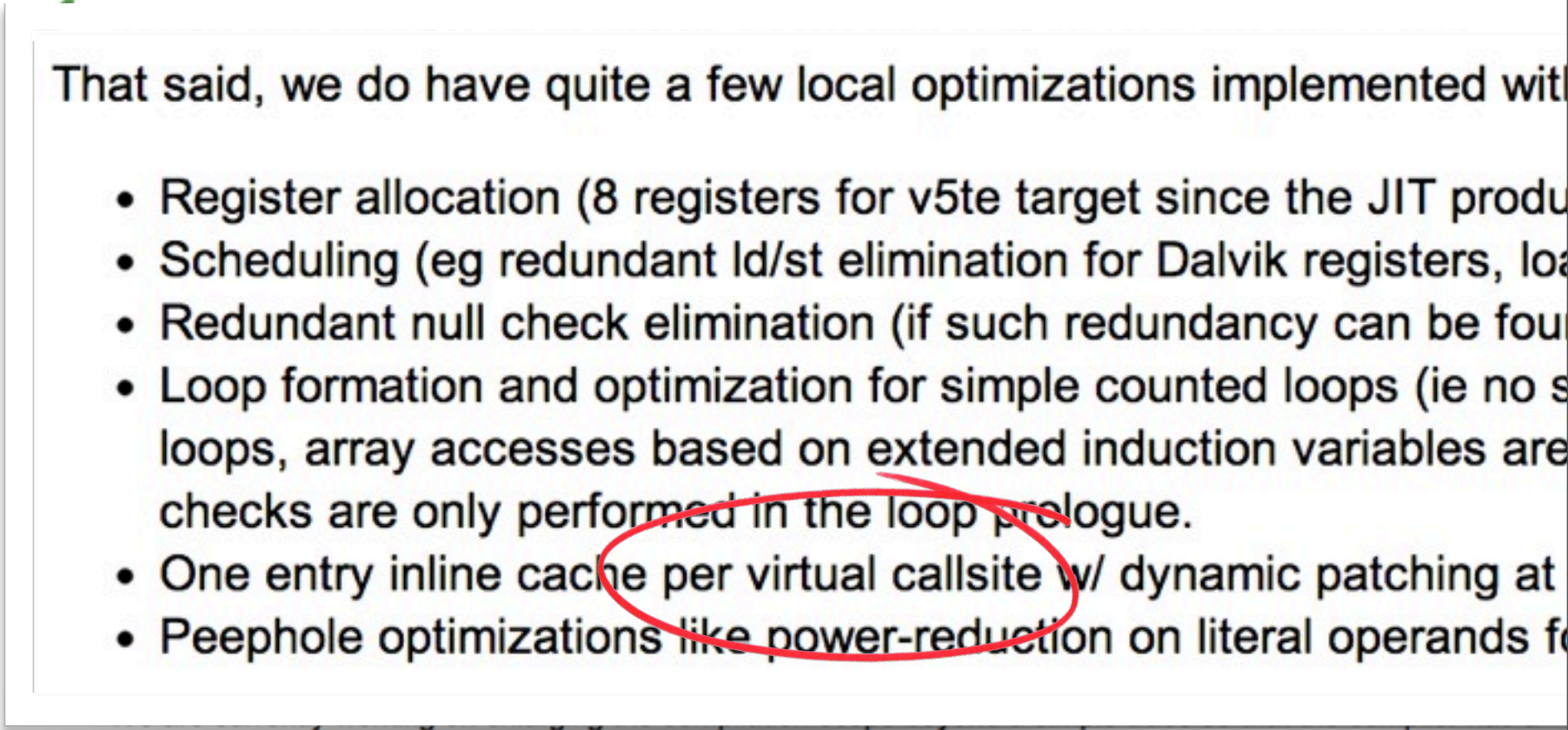
Ben

396 ● 2 ● 2

http://stackoverflow.com/questions/4912695/what-optimizations-can-i-expect-from-dalvik-and-the-android-toolchain

38

This is Ben, one of the engineers working on the JIT @ Google. When Bill and I started on this project, the goal was to deliver a working JIT as soon as possible with minimal impact to resource contention (eg memory footprint, CPU hijacked by the compiler thread) so that it can run on low-end devices as well. Therefore we used a very primitive trace based model. That is, the compilation entity passed to the JIT compiler is a basic block, sometimes as short as a single instruction. Such traces will be stitched together at runtime through a



larger window for code analysis and optimization. Stay tuned.

share | edit | flag

answered Feb 8 '11 at 7:20

Ben

396 ● 2 ● 2

http://stackoverflow.com/questions/4912695/what-optimizations-can-i-expect-from-dalvik-and-the-android-toolchain

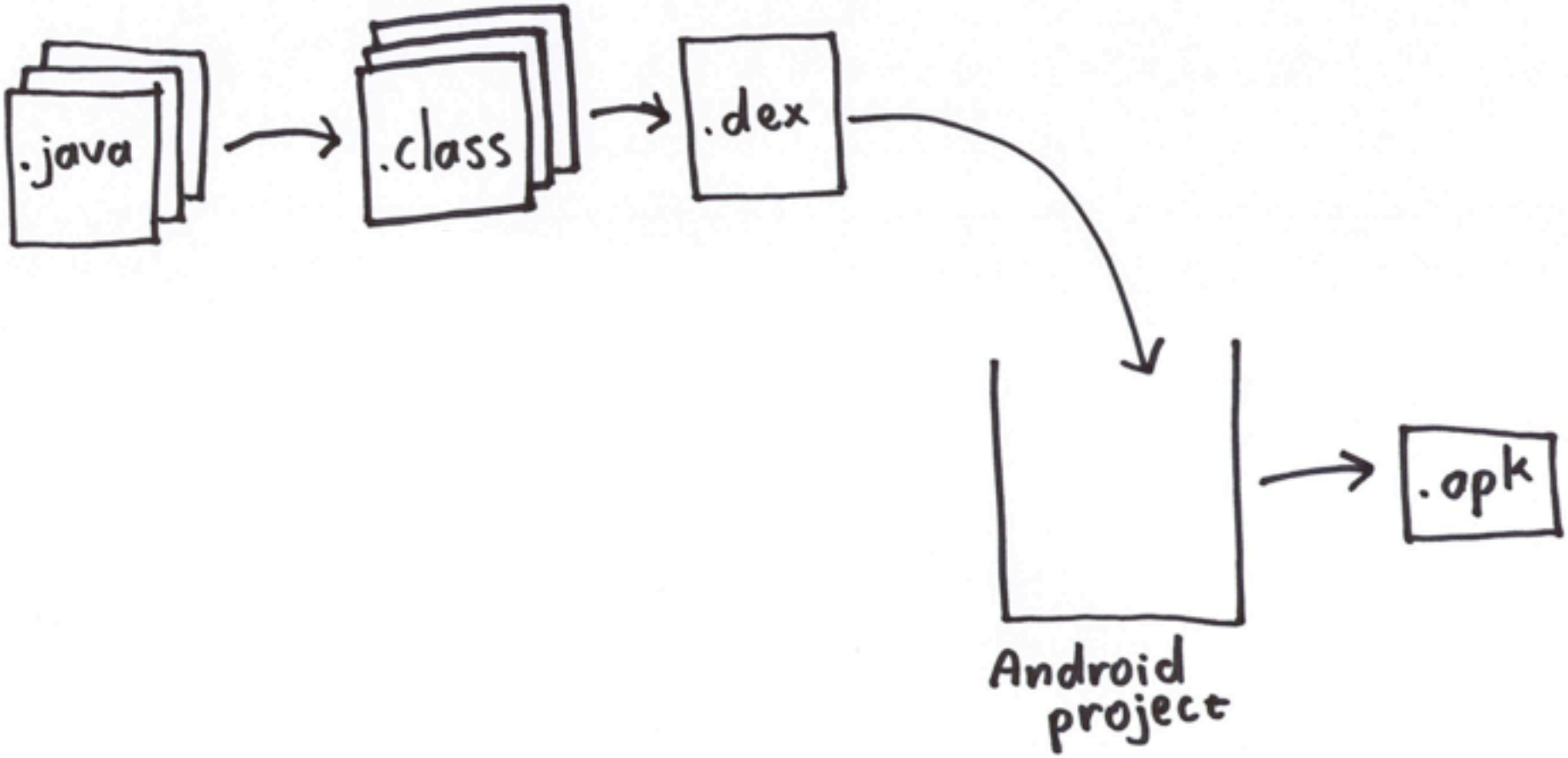
So?

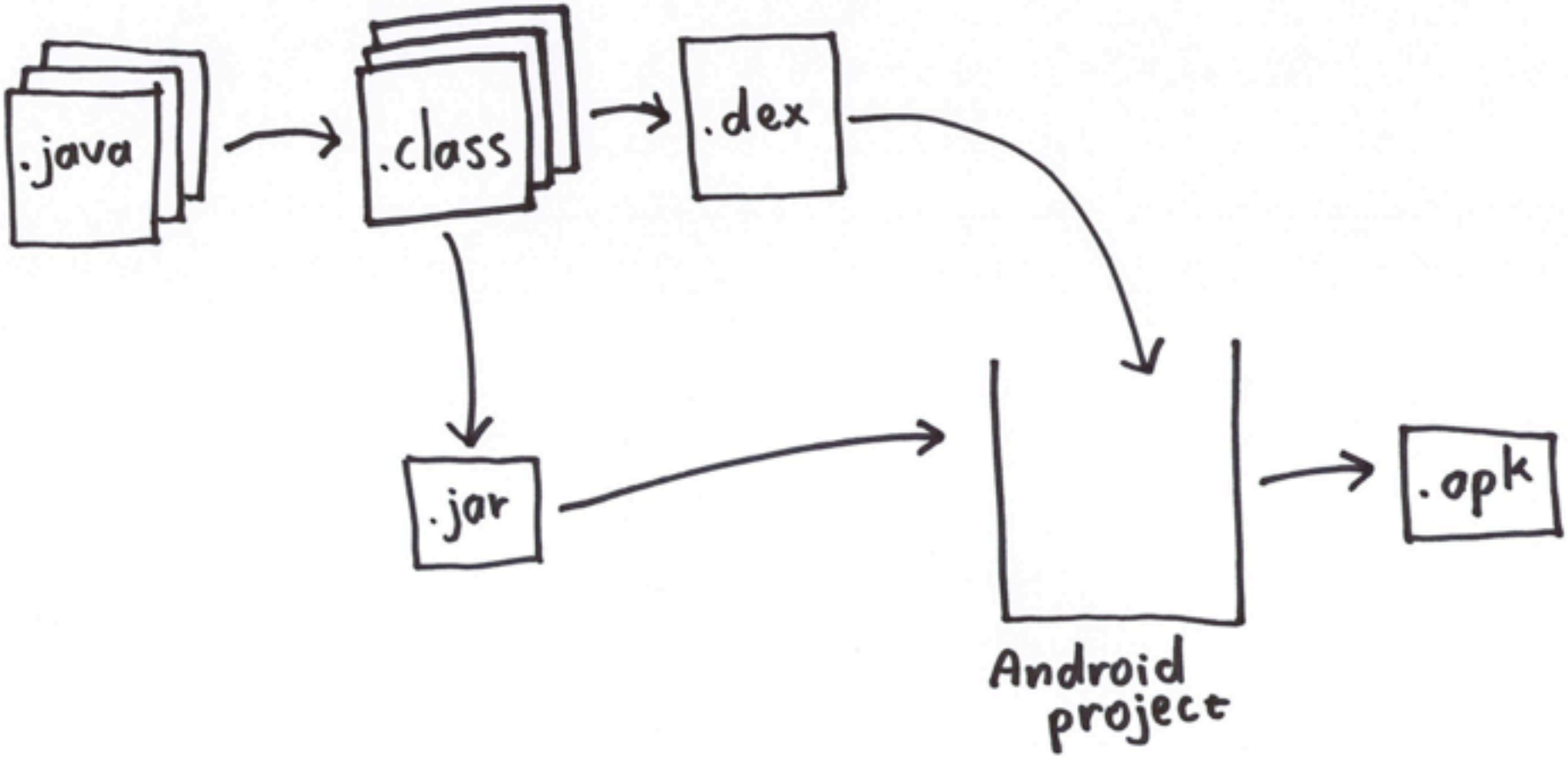
- Wishlist
  - Use service registry
  - No double classes
  - No over-optimization
    - but don’t kill all optimization

1. Strict separation

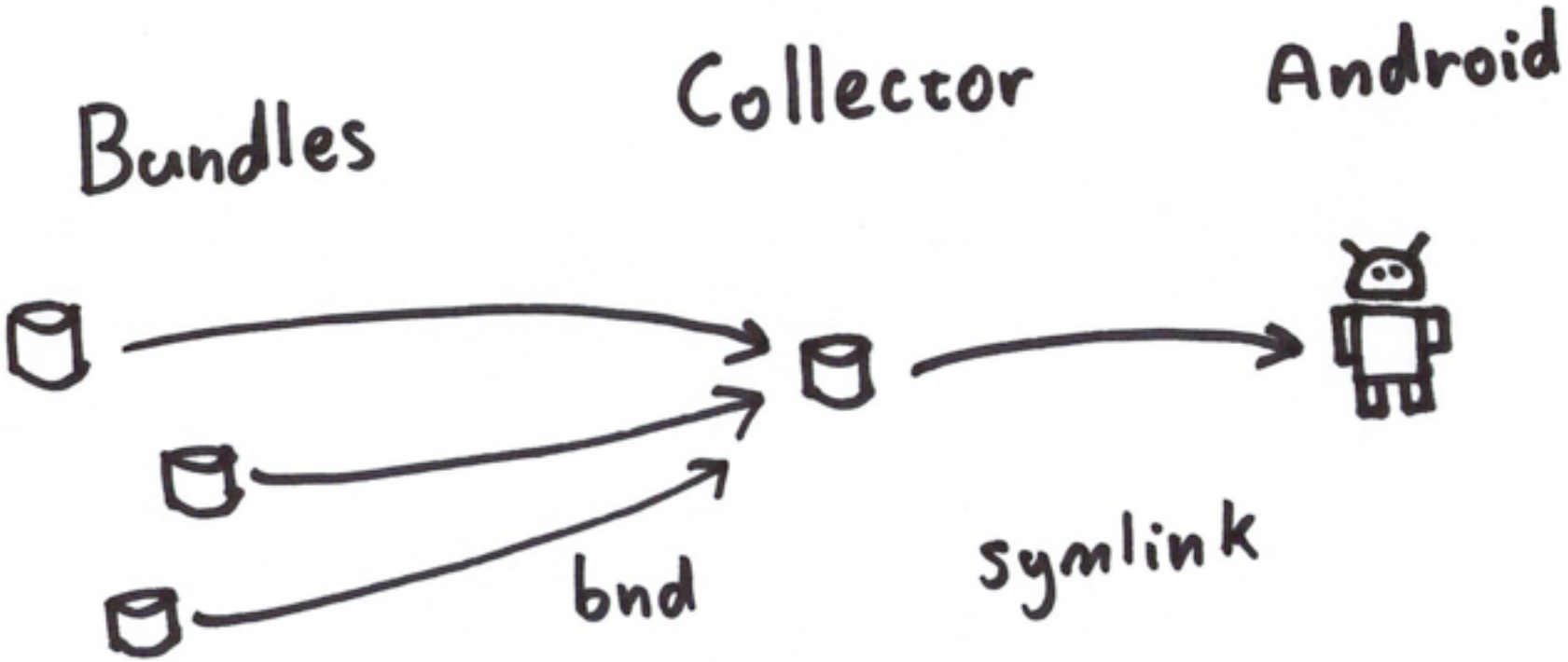
- Each class in exactly one bundle

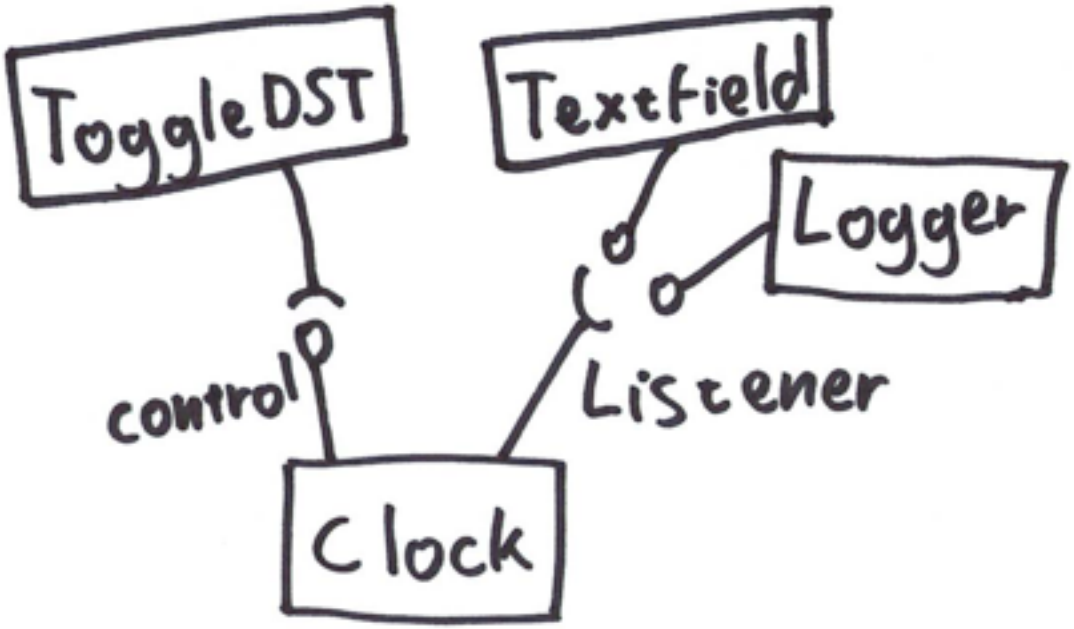
- How about no
  - removes our freedom of division of the class space
  - needs processing of third party bundles

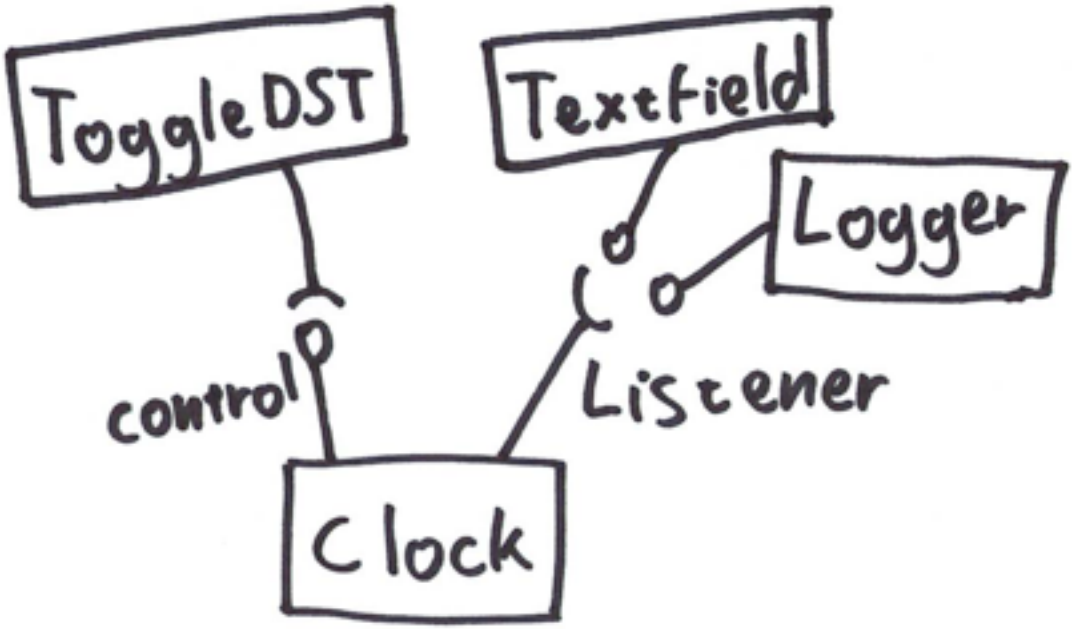

.java
.class
.dex
Android project
.apk

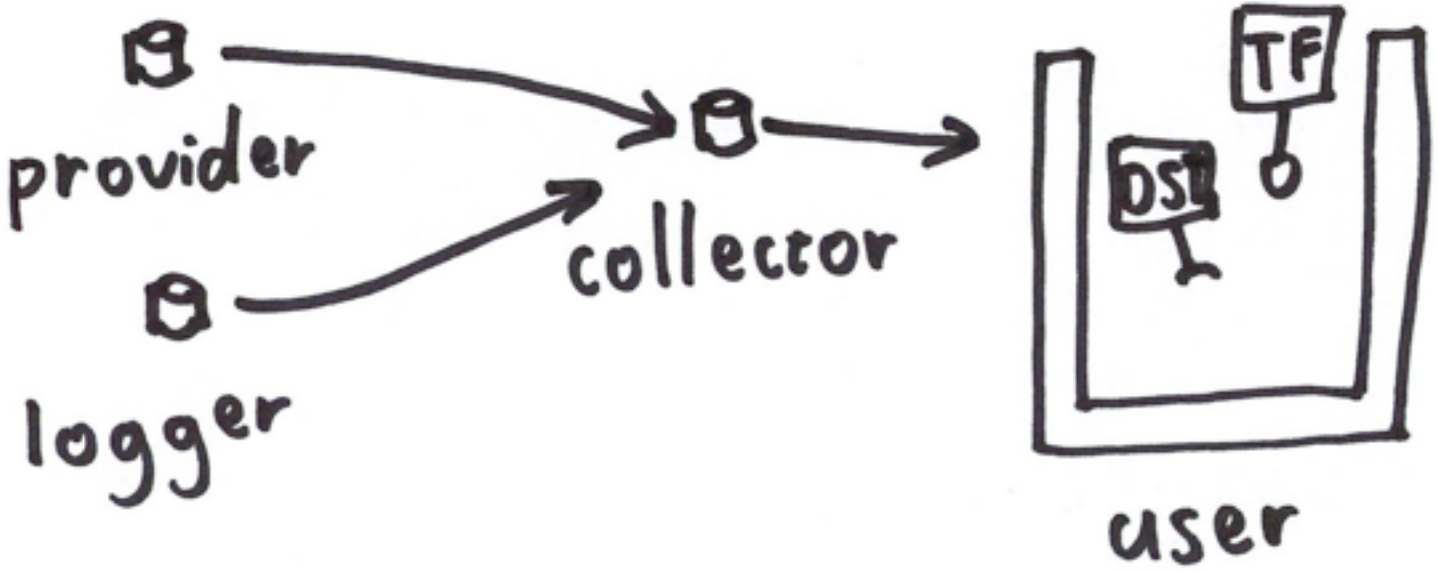

.java
.class
.dex
.jar
Android project
.apk

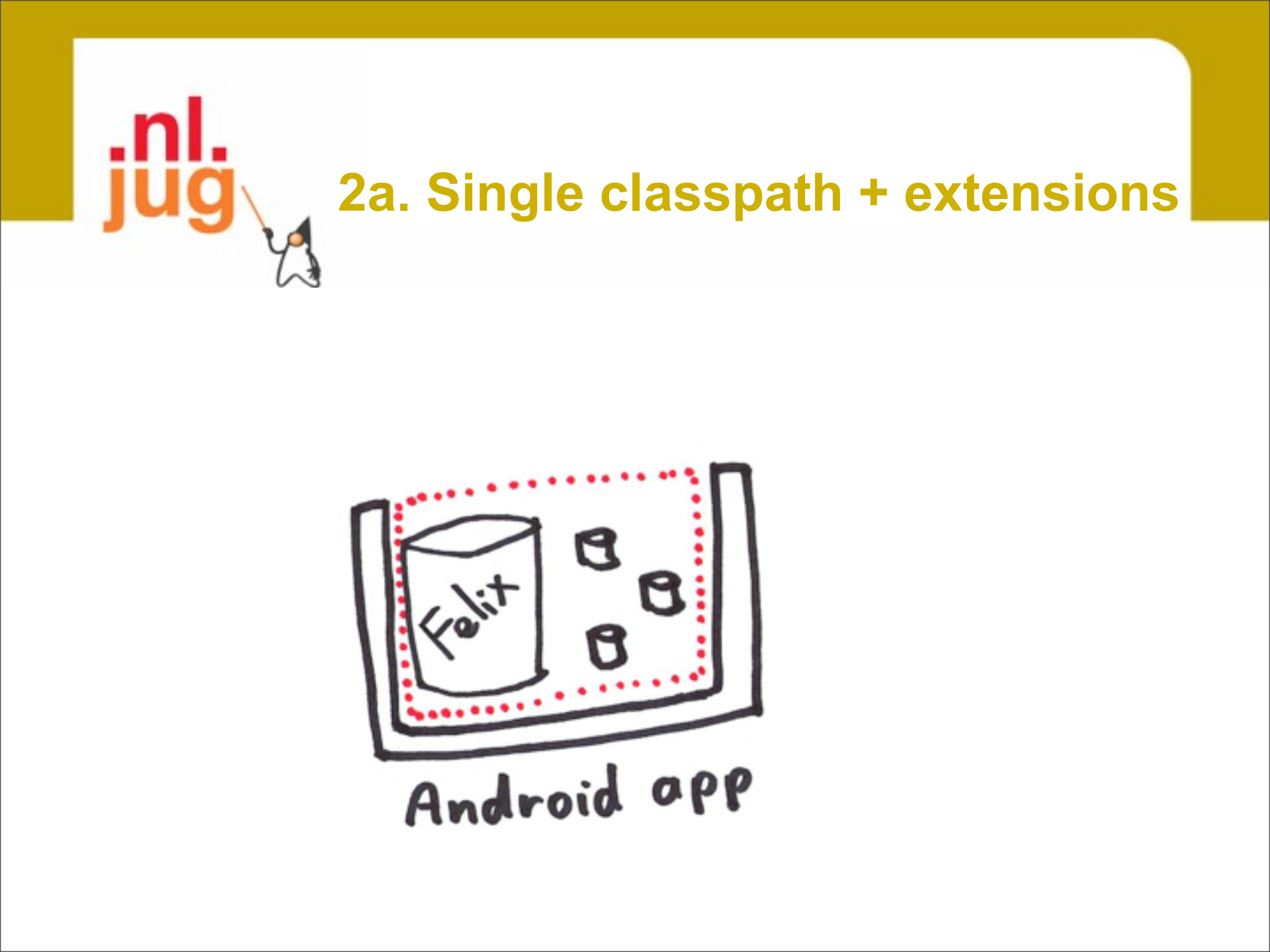
2. Single classpath



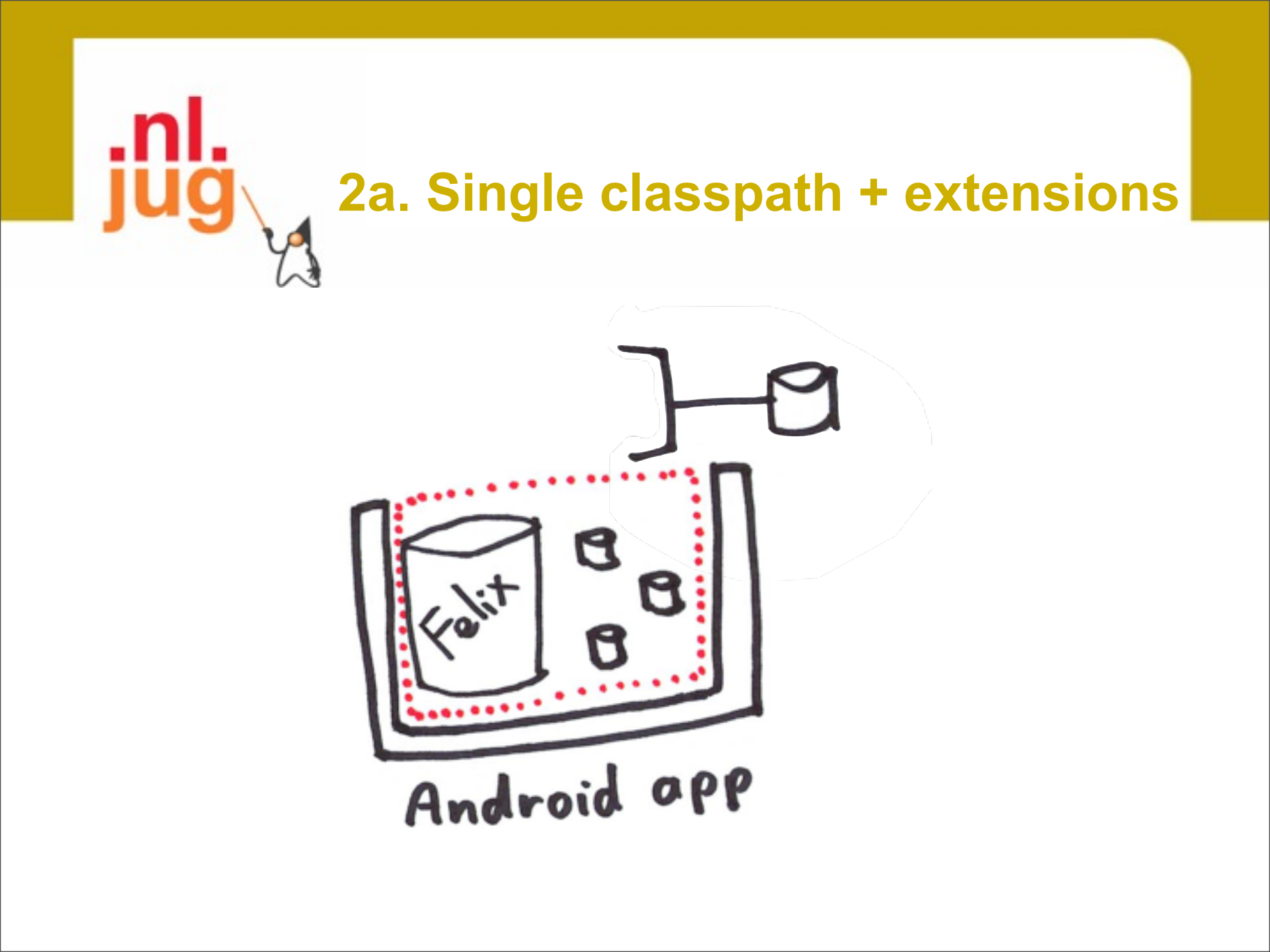

ToggleDST
Textfield
Logger
control
Listener
Clock


ToggleDST
Textfield
Logger
control
Listener
Clock


provider
logger
collector
TF
DST
user



2a. Single classpath + extensions



2a. Single classpath + extensions

2a. Single classpath + extensions

2a. Single classpath + extensions

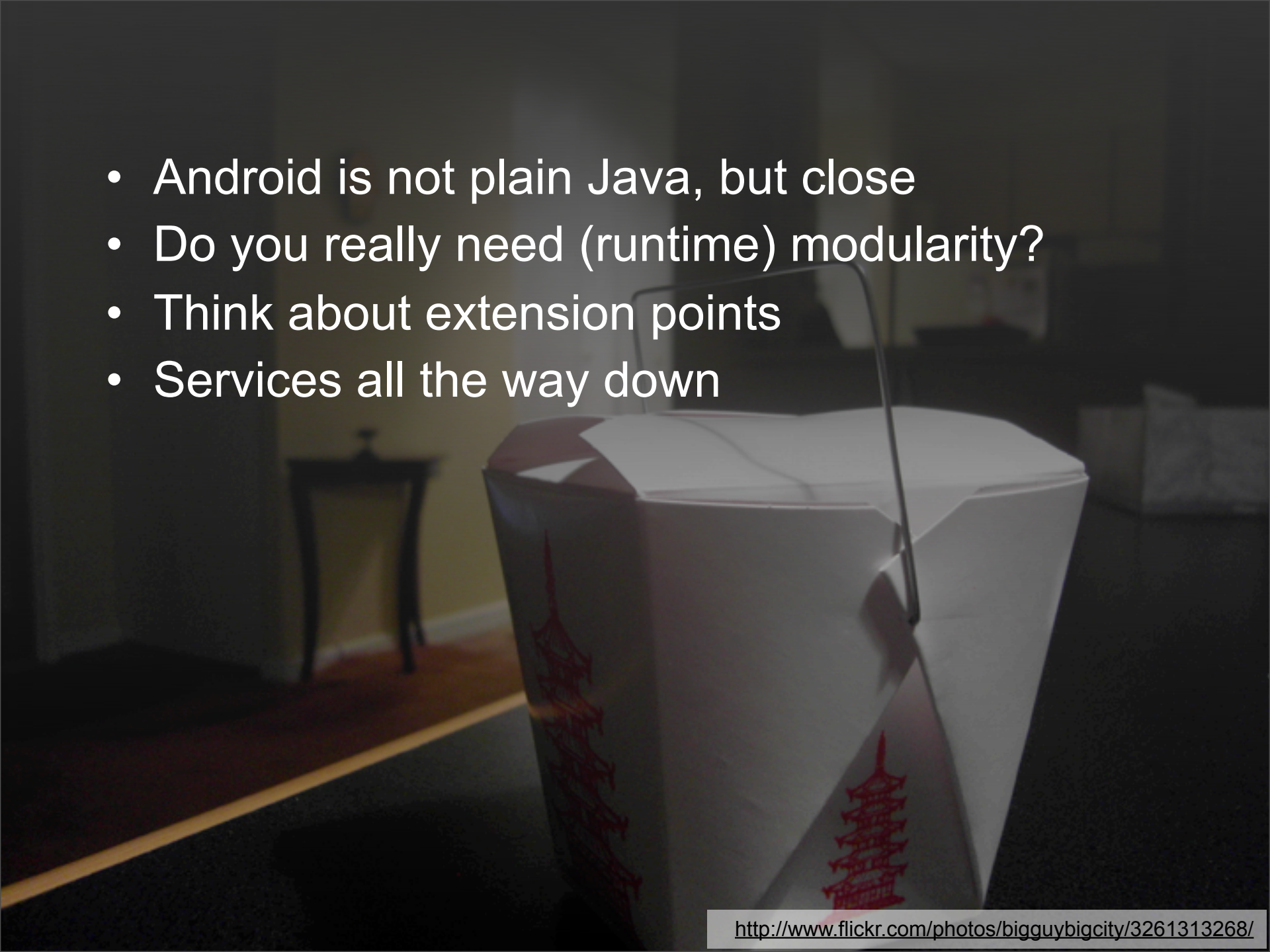

```
config.put(Constants.FRAMEWORK_SYSTEMPACKAGES_EXTRA, "...");
```


PojoSR
http://code.google.com/p/pojosr/

Yay, OSGi! Now what?

- Services, services, services
- Compendium services
  - Config Admin for configuration (i.e., screen setup)
  - Remote Service Admin for remoting
- Apache ACE for configurability

http://www.flickr.com/photos/bigguybigcity/3261313268/


• Android is not plain Java, but close
• Do you really need (runtime) modularity?
• Think about extension points
• Services all the way down
http://www.flickr.com/photos/bigguybigcity/3261313268/

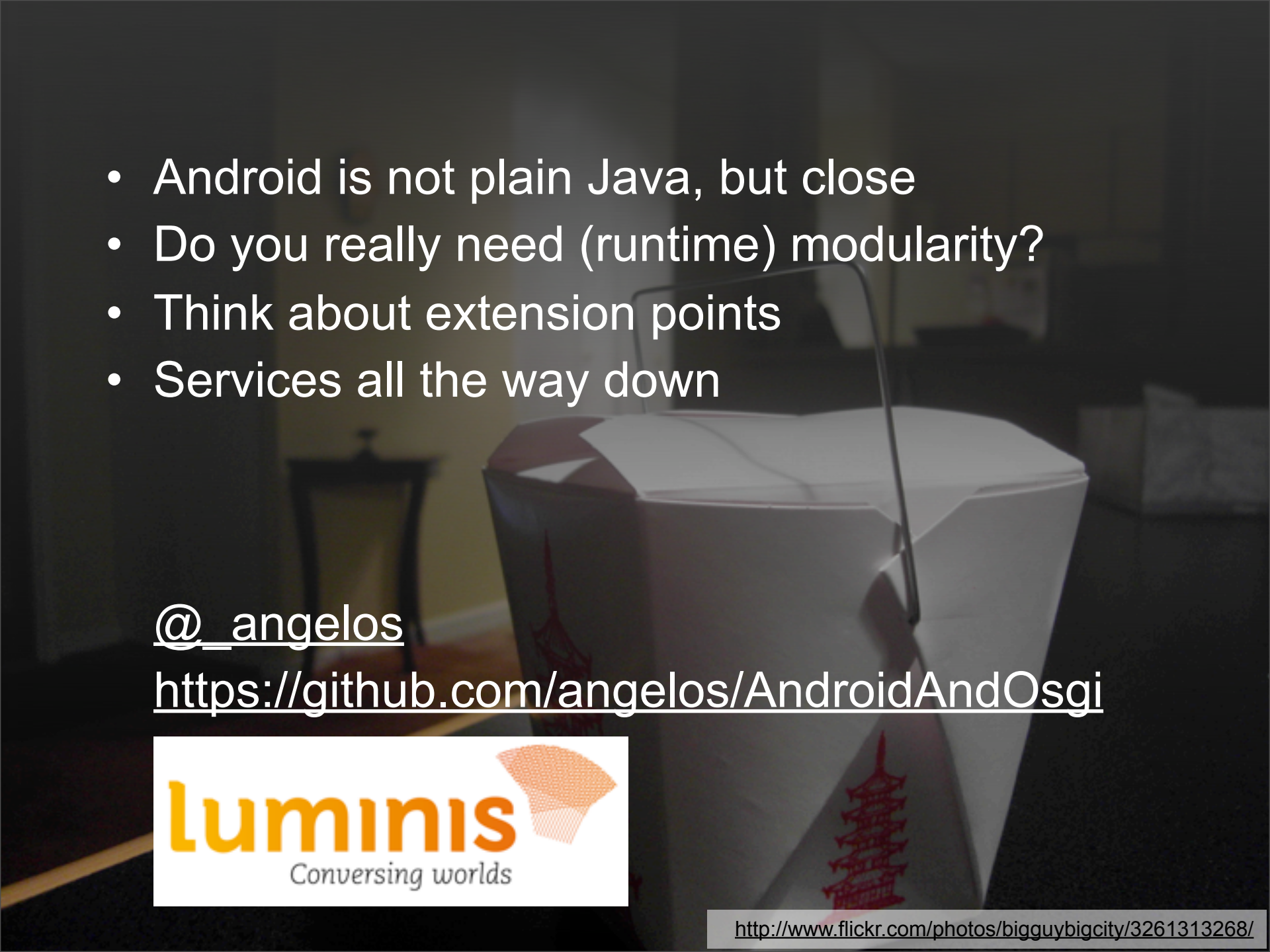
- Android is not plain Java, but close
- Do you really need (runtime) modularity?
- Think about extension points
- Services all the way down

@_angelos

https://github.com/angelos/AndroidAndOsgi

luminis

Conversing worlds



http://www.flickr.com/photos/bigguybigcity/3261313268/