Dalvik VM

Karthik Dantu and Steve Ko

Administrivia

- Assignment 3 is out, due in 1 ½ weeks.
- Please come on time.

Today: Dalvik VM

- Why?
  - Core of the Android runtime
- I’m not an expert on compiler/runtime, so this will be just an overview.
- Resources:
  - AOSP
  - https://dl.google.com/googleio/2010/android-jit-compiler-androids-dalvik-vm.pdf
  - http://fiona.dmcs.pl/podyplomowe_smtm/smob3/Presentation-Of-Dalvik-VM-Internals.pdf
  - http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf

General Java Execution

- Java compiler: Java code $\rightarrow$ Java compiler $\rightarrow$ .class files (per class bytecode)
  - The compiler generates machine-independent bytecode, not machine-specific binary.
- Java VM/Runtime: Load .class files $\rightarrow$ execute bytecode

Questions

- What is the bytecode format?
- How to execute bytecode?
- (Tangential) How to manage memory?

Bytecode

- What is bytecode?
  - Machine-neutral ISA
- Many popular languages/runtimes use this
  - Java, Python, OCaml, LLVM, etc.

Dalvik Executable (.dex)

- Bytecode format for Dalvik
- Register-based format
  - Java bytecode is stack-based.
- .dex generation
  - First pass: One .class file for each and every class
  - dx tool combines .class files into one .dex file.
  - Primarily for memory saving

.class file (from Java)

- Magic number, version info for Java
- Constant pool
- Super class
- Access flags (public, private, ...)
- Interfaces
- Fields
  - Name and type
  - Access flags (public, private, static, ...)
- Methods
  - Name and signature (argument and return types)
  - Access flags (public, private, static, ...)
  - Bytecode
  - Exception tables
- Other stuff (source file, line number table, ...)

Example

```
class Foo {
   public static void main(String[] args) {
      System.out.println("Hello world");
   }
}
```

Q) How many entries in the constant pool?

Example

```
class Foo {
   public static void main(String[] args) {
      System.out.println("Hello world");
   }
}
```

Q) How many entries in the constant pool?

A) 33

```
1)CONSTANT_Methodref[10](class_index = 6, name_and_type_index = 20)
2)CONSTANT_Fieldref[9](class_index = 21, name_and_type_index = 22)
3)CONSTANT_String[8](string_index = 23)
4)CONSTANT_Methodref[10](class_index = 24, name_and_type_index = 25)
5)CONSTANT_Class[7](name_index = 26)
6)CONSTANT_Class[7](name_index = 27)
7)CONSTANT_Utf8[1]("<init>")
8)CONSTANT_Utf8[1]("()V")
9)CONSTANT_Utf8[1]("Code")
10)CONSTANT_Utf8[1]("LineNumberTable")
11)CONSTANT_Utf8[1]("LocalVariableTable")
12)CONSTANT_Utf8[1]("this")
13)CONSTANT_Utf8[1]("LFoo;")
14)CONSTANT_Utf8[1]("main")
15)CONSTANT_Utf8[1]("([Ljava/lang/String;)V")
16)CONSTANT_Utf8[1]("args")
17)CONSTANT_Utf8[1]("[Ljava/lang/String;")
18)CONSTANT_Utf8[1]("SourceFile")
19)CONSTANT_Utf8[1]("Foo.java")
20)CONSTANT_NameAndType[12](name_index = 7, signature_index = 8)
21)CONSTANT_Class[7](name_index = 28)
22)CONSTANT_NameAndType[12](name_index = 29, signature_index = 30)
23)CONSTANT_Utf8[1]("Hello world")
24)CONSTANT_Class[7](name_index = 31)
25)CONSTANT_NameAndType[12](name_index = 32, signature_index = 33)
26)CONSTANT_Utf8[1]("Foo")
27)CONSTANT_Utf8[1]("java/lang/Object")
28)CONSTANT_Utf8[1]("java/lang/System")
29)CONSTANT_Utf8[1]("out")
30)CONSTANT_Utf8[1]("Ljava/io/PrintStream;")
31)CONSTANT_Utf8[1]("java/io/PrintStream")
32)CONSTANT_Utf8[1]("println")
33)CONSTANT_Utf8[1]("(Ljava/lang/String;)V")
```

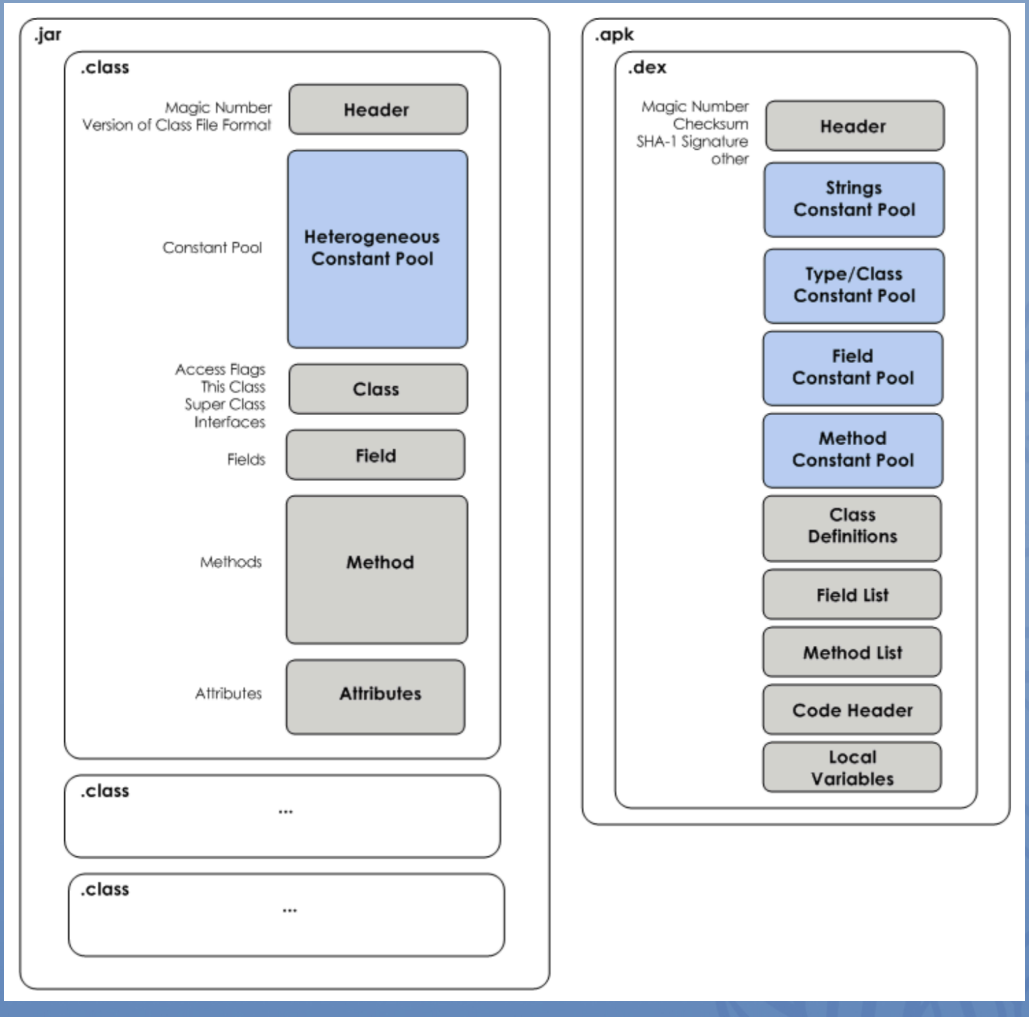
Java Bytecode Assembly

- Popular one is called Jasmin

```
.method foo()V
    .limit locals 1
    ; declare variable 0 as an "int Count;"
    ; whose scope is the code between Label1 and Label2
    .var 0 is Count I from Label1 to Label2

  Label1:
    bipush 10
    istore_0
  Label2:
    return
  .end method
```

Dalvik Executable (.dex)



Decompiling .dex

- Smali
  - https://code.google.com/p/smali/
- Soot
  - http://www.sable.mcgill.ca/soot/

Questions

- What is the bytecode format?
- How to execute bytecode?
- (Tangential) How to manage memory?

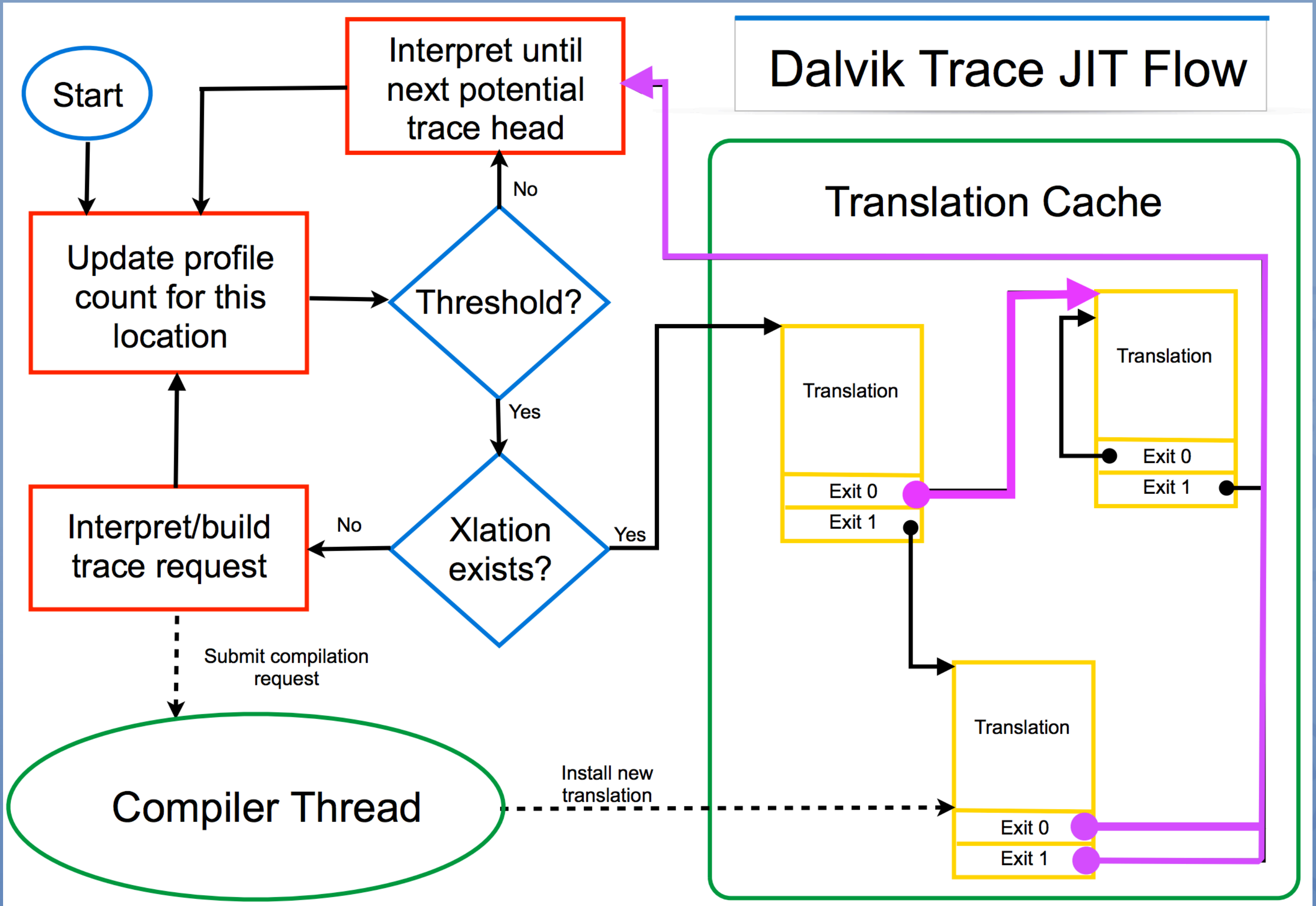
Compile/Runtime Support

- Ahead-of-time compile
  - C/C++, etc.
  - Generating machine-dependent binaries
- (Pure) Interpretation
  - Probably no popular example
  - Interpreting on-the-go
- Just-in-time compile
  - Don’t interpret every time, but generate machine code at runtime and keep it for later.

How to JIT

- A large design space
  - When? Installation, launch, method invoke, etc.
  - What? Everything, instruction, method, etc.
- Dalvik: Trace-based JIT
  - Only compile “hot” paths (typically under 10% of the code)
  - Good for performance/memory footprint, bad for optimization (loses optimization opportunities)
- ART: Installation-time JIT at the method granularity

Dalvik JIT


Dalvik Trace JIT Flow
Start
Interpret until next potential trace head
Update profile count for this location
Threshold?
No
Yes
Xlation exists?
No
Yes
Interpret/build trace request
Submit compilation request
Compiler Thread
Install new translation
Translation Cache
Translation
Exit 0
Exit 1
Translation
Exit 0
Exit 1
Translation
Exit 0
Exit 1

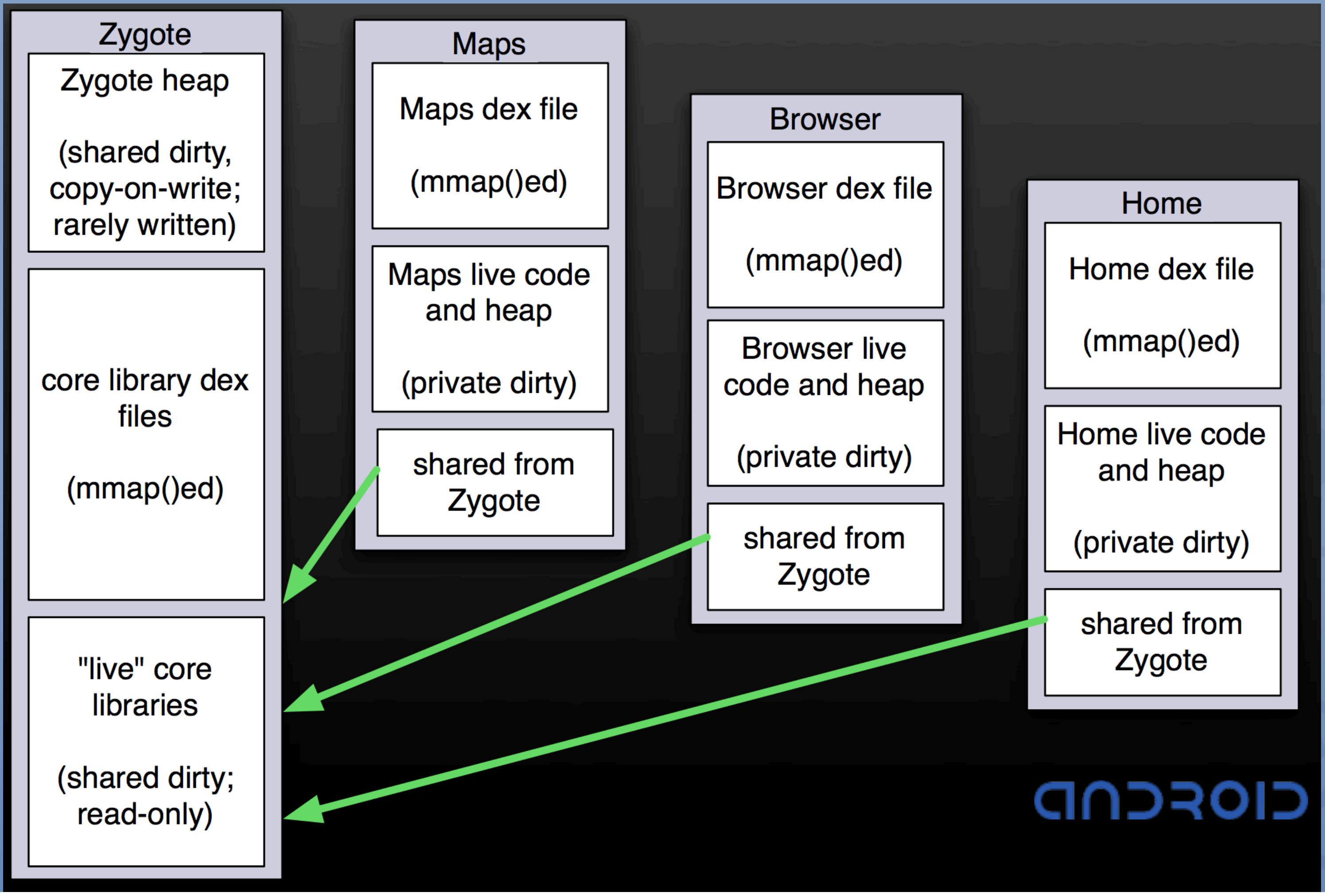
.odex

- During installation, Dalvik performs some optimization on the bytecode itself.
- Dalvik generates .odex file for each app at installation time.
  - Static linking
  - Method in-lining
  - Removal of empty methods
  - Etc.

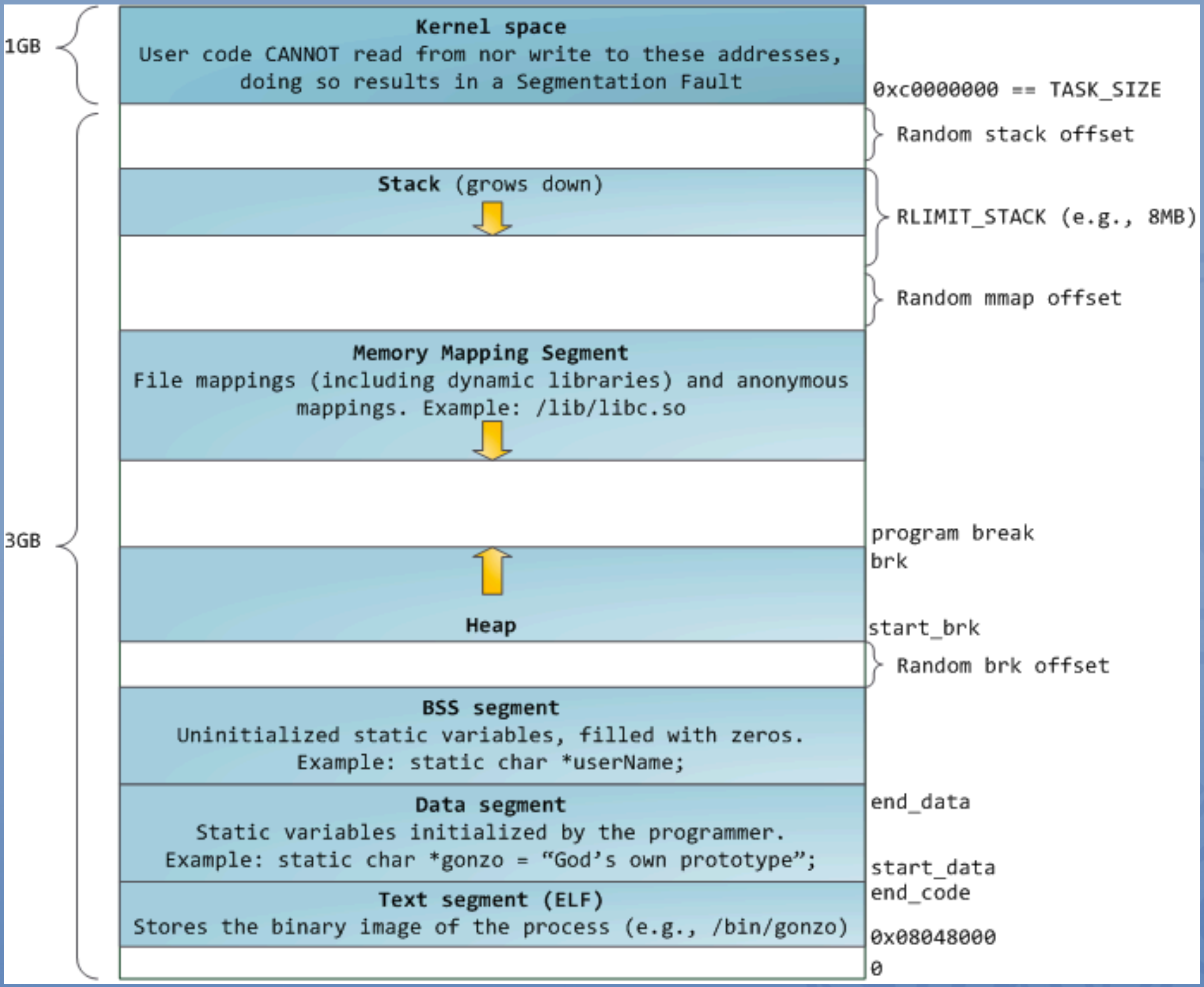
Questions

- What is the bytecode format?
- How to execute bytecode?
- (Tangential) How to manage memory?

Memory Management



Typical Memory Structure


1GB
3GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc0000000 == TASK_SIZE
Random stack offset
Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so
program break
brk
Heap
start_brk
Random brk offset
BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;
Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = "God's own prototype";
end_data
start_data
end_code
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
0x08048000
0

Heap Management in Dalvik

- Used for dynamic memory requests
- In Java, there's no free(), so allocation/deallocation is automatically done by VM.
- GC (Garbage Collection) is used.
  - A stop-all method: Suspends the app execution, scans the whole heap, and runs the GC algorithm

Dalvik GC

- Mark and sweep

```
void mark (Object p) {
    if (!p.marked) {
        p.marked = true;
        for each Object q referenced by p
            mark(q);
    }
}
```

```
void sweep () {
    for each p in heap
        if (p.marked)
            p.marked = false;
        else
            heap.release(p);
}
```

Summary

- What is the bytecode format?
  - Dalvik Executable (.dex)
- How to execute bytecode?
  - Trace-based JIT
- (Tangential) How to manage memory?
  - Sharing with Zygote & GC