G54MDP
Mobile Device Programming

Lecture 3 – Mobile Phone Architecture / Android Internals

Mobile Device Characteristics

- CPU ~1Gz
- GPU
- Memory
  - RAM 128MB-1GB
  - Flash Storage 16-64GB, internal / external via SD card
- Communications
  - Telephony
  - WiFi
  - Bluetooth
  - NFC
- Screen
- Audio
- User input
- Battery

Mobile CPUs

- Almost all Mobile Devices use ARM CPUs
- Originated from Acorn computers in the late-1980s
- Acorn RISC Machine
- Spun out from Acorn in early-1990s
- Now, Advanced RISC machine

ARM CPU

- 32-bit CPU (64 bit iPhone)
  - Multiple instruction sets
  - Older phones made use of Jazelle DBX / embedded systems
    - Run Java bytecode directly, dedicated chips
- Sold as a design, not a physical device
  - Great for SoC usage
  - Chip manufactures can add ARM CPU to bespoke SoC vs buying a chip from Intel
- Aims
  - Fast and efficient
    - Good for mobile applications
    - Best use of battery – more instructions = more battery use
  - High code density
    - Less moving stuff around, best use of space

ARM CPU

- RISC-design (Reduced Instruction Set Chip)
- Only includes simple instructions that can be executed in one clock cycle
- Remove instructions you might expect in x86
  - e.g. divide instructions
  - If you need to divide, you roll your own in software
  - Or rather, the compiler does it for you

ARM CPU

Register names

r0	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	r14	r15

- 16 registers
- Load/Store architecture
- Each instruction is 32-bits long
- Makes decoding easy / efficient
  - Vs x86 variable instruction length
- But means loading a constant into a register can be tricky

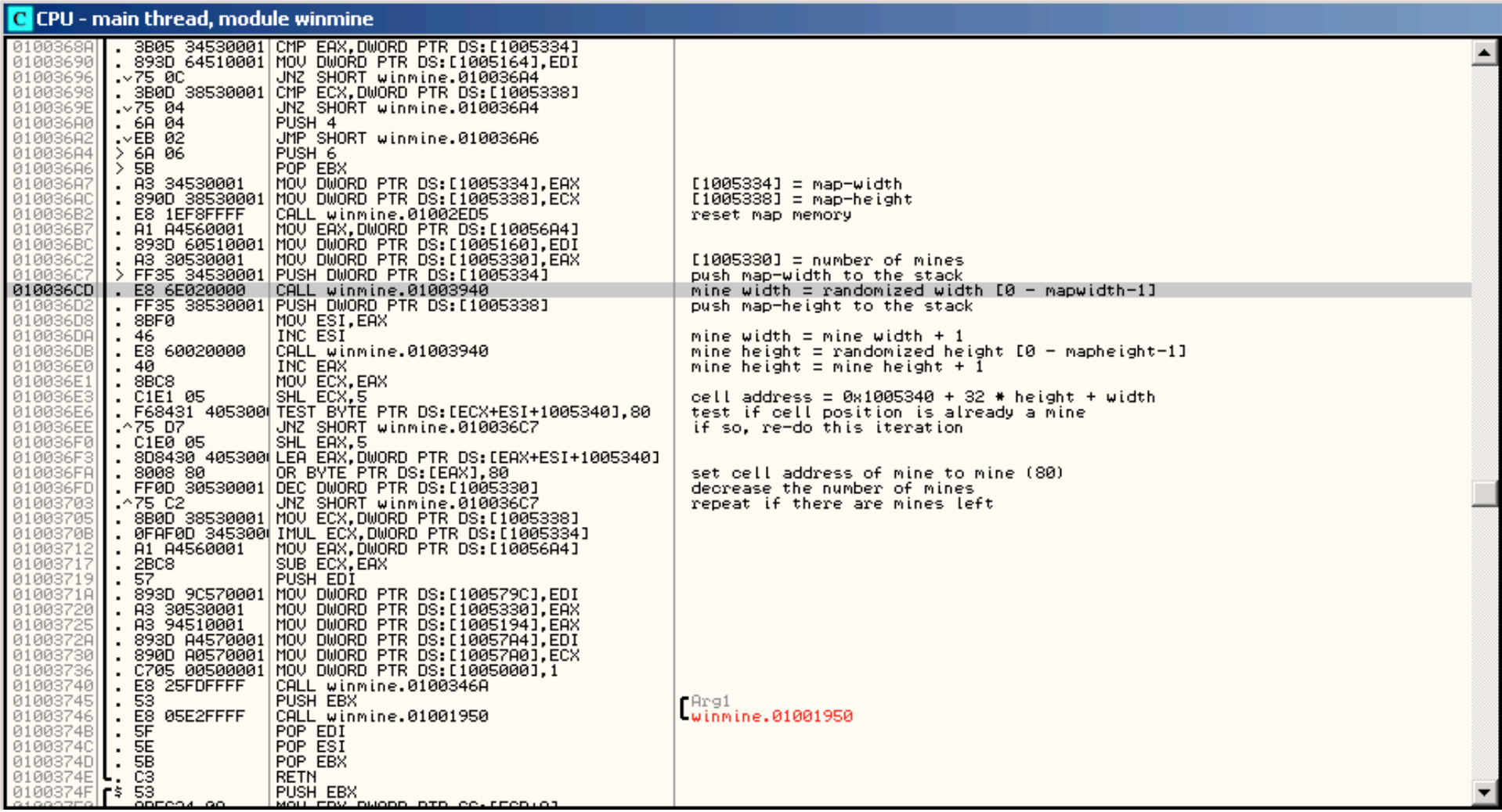
```
.data

msg:
   .ascii      "Hello, ARM!\n"
len = . - msg

.text

.globl _start
_start:
   mov     %r0, $1     /* fd -> stdout */
   ldr     %r1, =msg   /* buf -> msg */
   ldr     %r2, =len   /* count -> len(msg) */
   mov     %r7, $4     /* write is syscall #4 */
   swi     $0          /* invoke syscall */
   mov     %r0, $0     /* status -> 0 */
   mov     %r7, $1     /* exit is syscall #1 */
   swi     $0          /* invoke syscall */
```

Opcode	Type	Decoded Instruction
`0xE3A00000`	Data Processing	`MOV R0, #0`
`0xEF000002`	Software Interrupt	`SWI 2`
`0xEAFFFFFC`	Branch Instruction	`B -16`
`0xE0810002`	Data Processing	`ADD R0, R1,R2`


CPU - main thread, module winmine
0100368A . 3B05 34530001 CMP EAX,DWORD PTR DS:[1005334]
01003690 . 893D 64510001 MOV DWORD PTR DS:[1005164],EDI
01003696 .v75 0C JNZ SHORT winmine.010036A4
01003698 . 3B0D 38530001 CMP ECX,DWORD PTR DS:[1005338]
0100369E .v75 04 JNZ SHORT winmine.010036A4
010036A0 . 6A 04 PUSH 4
010036A2 .vEB 02 JMP SHORT winmine.010036A6
010036A4 > 6A 06 PUSH 6
010036A6 > 5B POP EBX
010036A7 . A3 34530001 MOV DWORD PTR DS:[1005334],EAX [1005334] = map-width
010036AC . 890D 38530001 MOV DWORD PTR DS:[1005338],ECX [1005338] = map-height
010036B2 . E8 1EF8FFFF CALL winmine.01002ED5 reset map memory
010036B7 . A1 A4560001 MOV EAX,DWORD PTR DS:[10056A4]
010036BC . 893D 60510001 MOV DWORD PTR DS:[1005160],EDI
010036C2 . A3 30530001 MOV DWORD PTR DS:[1005330],EAX [1005330] = number of mines
010036C7 > FF35 34530001 PUSH DWORD PTR DS:[1005334] push map-width to the stack
010036CD . E8 6E020000 CALL winmine.01003940 mine width = randomized width [0 - mapwidth-1]
010036D2 . FF35 38530001 PUSH DWORD PTR DS:[1005338] push map-height to the stack
010036D8 . 8BF0 MOV ESI,EAX
010036DA . 46 INC ESI mine width = mine width + 1
010036DB . E8 60020000 CALL winmine.01003940 mine height = randomized height [0 - mapheight-1]
010036E0 . 40 INC EAX mine height = mine height + 1
010036E1 . 8BC8 MOV ECX,EAX
010036E3 . C1E1 05 SHL ECX,5 cell address = 0x1005340 + 32 * height + width
010036E6 . F68431 405300 TEST BYTE PTR DS:[ECX+ESI+1005340],80 test if cell position is already a mine
010036EE .^75 D7 JNZ SHORT winmine.010036C7 if so, re-do this iteration
010036F0 . C1E0 05 SHL EAX,5
010036F3 . 8D8430 405300 LEA EAX,DWORD PTR DS:[EAX+ESI+1005340]
010036FA . 8008 80 OR BYTE PTR DS:[EAX],80 set cell address of mine to mine (80)
010036FD . FF0D 30530001 DEC DWORD PTR DS:[1005330] decrease the number of mines
01003703 .^75 C2 JNZ SHORT winmine.010036C7 repeat if there are mines left
01003705 . 8B0D 38530001 MOV ECX,DWORD PTR DS:[1005338]
0100370B . 0FAF0D 345300 IMUL ECX,DWORD PTR DS:[1005334]
01003712 . A1 A4560001 MOV EAX,DWORD PTR DS:[10056A4]
01003717 . 2BC8 SUB ECX,EAX
01003719 . 57 PUSH EDI
0100371A . 893D 9C570001 MOV DWORD PTR DS:[100579C],EDI
01003720 . A3 30530001 MOV DWORD PTR DS:[1005330],EAX
01003725 . A3 94510001 MOV DWORD PTR DS:[1005194],EAX
0100372A . 893D A4570001 MOV DWORD PTR DS:[10057A4],EDI
01003730 . 890D A0570001 MOV DWORD PTR DS:[10057A0],ECX
01003736 . C705 00500001 MOV DWORD PTR DS:[1005000],1
01003740 . E8 25FDFFFF CALL winmine.0100346A
01003745 . 53 PUSH EBX Arg1
01003746 . E8 05E2FFFF CALL winmine.01001950 winmine.01001950
0100374B . 5F POP EDI
0100374C . 5E POP ESI
0100374D . 5B POP EBX
0100374E . C3 RETN
0100374F $ 53 PUSH EBX

Load and Store

- Data can only be moved
  - From a register to memory (store)
  - From memory to a register (load)
- Not memory to memory
- Cannot perform operations on memory

```
ADD r0, r1, r2
ADD eax, [0x00000001]
```

Constants

- Constants must fit into the 32-bit instruction width
- Can't therefore be the full 32-bits
- 8-bit + a 4-bit shift — gives a wide range of values
- Also a Move Negated instruction
- If that doesn't work, load from a literal pool

ARM Speed

- Does running at 1GHz means its going to be slow?
- GHz-speed tells us ‘cycles per second’
  - Actual speed depends on how many cycles an instruction takes
  - ARM aims for one-cycle per instruction
- x86 instructions can take many cycles
  - If an ARM instruction takes 1 cycle at 1GHz
  - And an x86 instruction takes 3 cycles at 3GHz
  - Which is faster?
- Speed is not entirely down to clock speed
  - It’s what you do with it

ARM Conditional Execution

- Often want to conditionally execute some code – while loops, for loops etc

```
cmp al,dl
jg label1
```

  - Traditional approach is to execute a compare
  - Branch if the condition is met
  - Branches are 'expensive' (difficult to make parallel efficiently)
- ARM allows any instruction to made conditional
  - Remove the need for an expensive branch
  - Smaller code footprint

Barrel Shifter

- ARM has a barrel shifter that can be used on any instruction
- Shift/Rotate the bits within a binary number

```
MOV r0, r0, LSL
```

  - Multiply r0 by 2 in one operation
  - Divide by 4 using LSR

ARM and Thumb

- Every ARM instruction is 32bits long
  - Can take up a lot of memory
  - Memory accesses take time
  - Can slow things down
- Thumb
  - 16-bit version of the ARM instruction set
  - Variable length instruction set
  - Took the most popular ARM instructions used and encoded them as 16-bit values
  - Why?

ARM and Thumb

- Speed
  - ARM instructions require 32bits to be read every instruction
  - Memory reads take time
  - Not all memory is 32bit wide
    - On smaller RAM width, takes multiple reads to get an instruction
- By making the instructions smaller gain a speed up
  - 8-bit memory two reads per instruction
  - 16-bit memory, one read per instruction
  - 32-bit memory, one read gives two instructions
- iPhone SDK generates Thumb by default

Floating Point

- Thumb doesn't encode FPU instructions
- CPU would have to branch to ARM instructions to execute it
  - This takes time
- FPU heavy code is better compiled to ARM, not Thumb
  - Assuming the device has an FPU!

ARM big.LITTLE

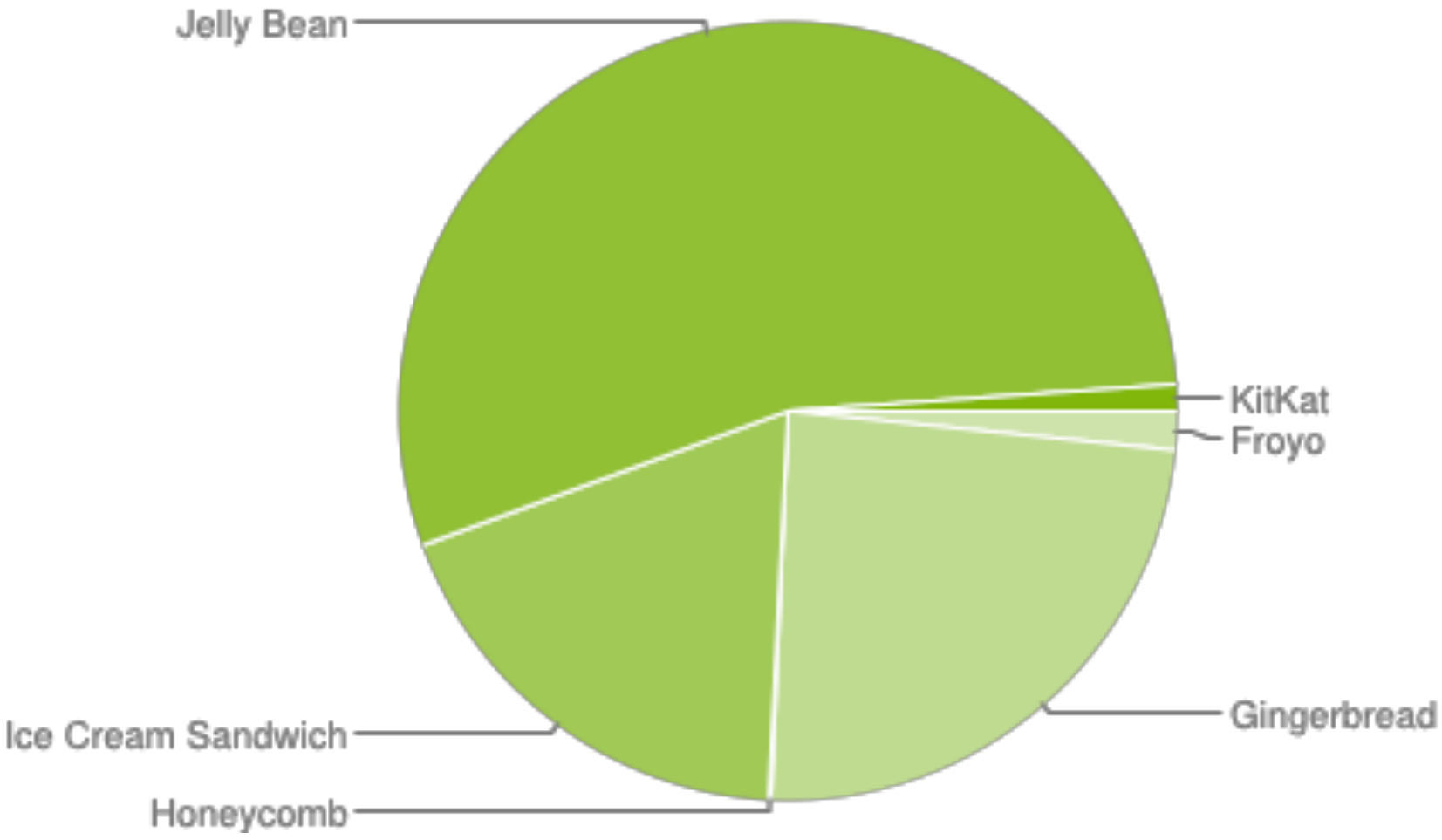
- ARM's latest processor contains two cores
  - 1.8GHz quad core optimised for performance (big core)
  - 1.2GHz quad core optimised for energy efficiency (LITTLE core)
- Two Cores are architecturally consistent
- System can switch between the two as appropriate for the task in hand

Android

- An operating system for mobile phones
- Purchased by Google in 2005
- Open (sort of)
  - Open source / Apache license eventually
  - (Bootloaders / rooting)
- Leverages existing technology
  - Linux (but not really Linux)
    - Linux kernel, custom middleware
  - Java (but not really Java)
- A different programming model

Android versions

- Several versions in the wild
- Android 1.5/1.6 – generally not seen anymore
- Android 2.2/2.3 - phones/cheap tablets
- Android 3.x (Honeycomb) - Tablets only
- Android 4.0 (Ice Cream Sandwich 15)
- Android 4.4 (KitKat 19)
- Forks (e.g. Amazon Kindle Fire)

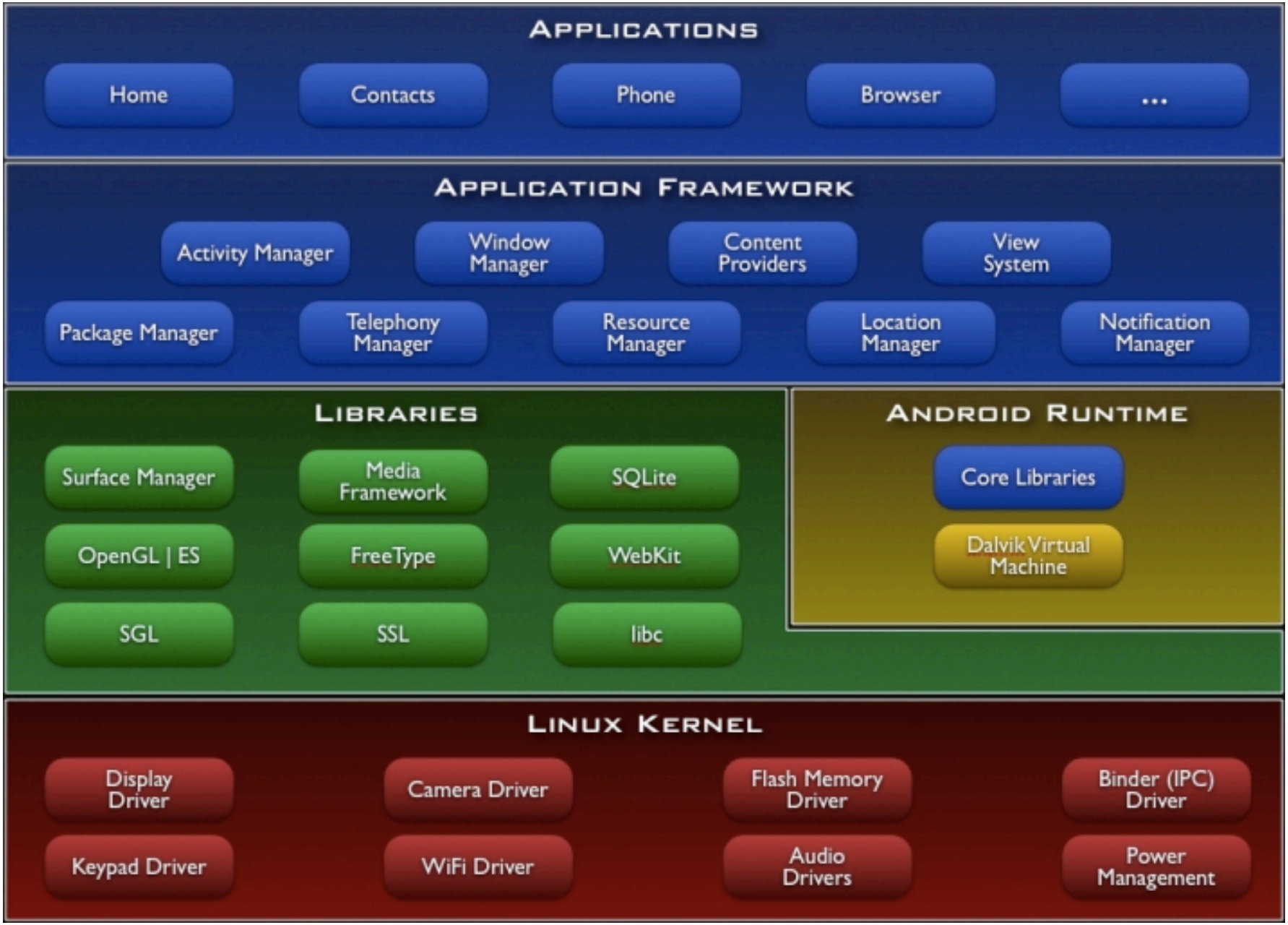

Jelly Bean
KitKat
Froyo
Gingerbread
Ice Cream Sandwich
Honeycomb

Android Compatibility

- Claims to be forwards / backwards compatible
  - An application built against 1.5 should work on the newest 4.4 device
  - Some support for backwards compatibility
    - The newest features may have fallback equivalents in older APIs
    - Obviously cannot use an API that does not exist
- The Android logo is CC licensed
- Can only call a device an “Android phone” if it passes compatibility tests / supports the API
  - “Android” the brand licensed to OMA members
    - Open Mobile Alliance

Android

- A software stack for mobile devices
  - Tablets, phones
- Operating system kernel
- Standard middleware
  - Android library support
- Key applications / user interfaces
  - Vendor specific modifications


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

```
1
2  import java.lang.System;
3
4  public class HelloWorld
5  {
6      public static void main(String args[])
7      {
8          System.out.println("hello world");
9      }
10 }
```

```
Code:
 0:   iconst_2
 1:   istore_1
 2:   iload_1
 3:   sipush  1000
 6:   if_icmpge       44
 9:   iconst_2
 10:  istore_2
 11:  iload_2
 12:  iload_1
 13:  if_icmpge       31
 16:  iload_1
 17:  iload_2
 18:  irem             # remainder
 19:  ifne    25
 22:  goto    38
 25:  iinc    2, 1
 28:  goto    11
 31:  getstatic       #84; //Field java/lang/System.out:Ljava/io/PrintStream;
 34:  iload_1
 35:  invokevirtual   #85; //Method java/io/PrintStream.println:(I)V
 38:  iinc    1, 1
 41:  goto    2
 44:  return
```

Android Apps

- Applications are written in Java
  - But run on Google's own VM — Dalvik
  - Uses its own bytecode (DEX) format
- Code compiled using standard Java tools then convert to DEX format
  - Multiple class files in a single .dex file
- Code, data and resource files packed into a .apk file
  - Classes
  - Configuration
  - Resources

Dalvik

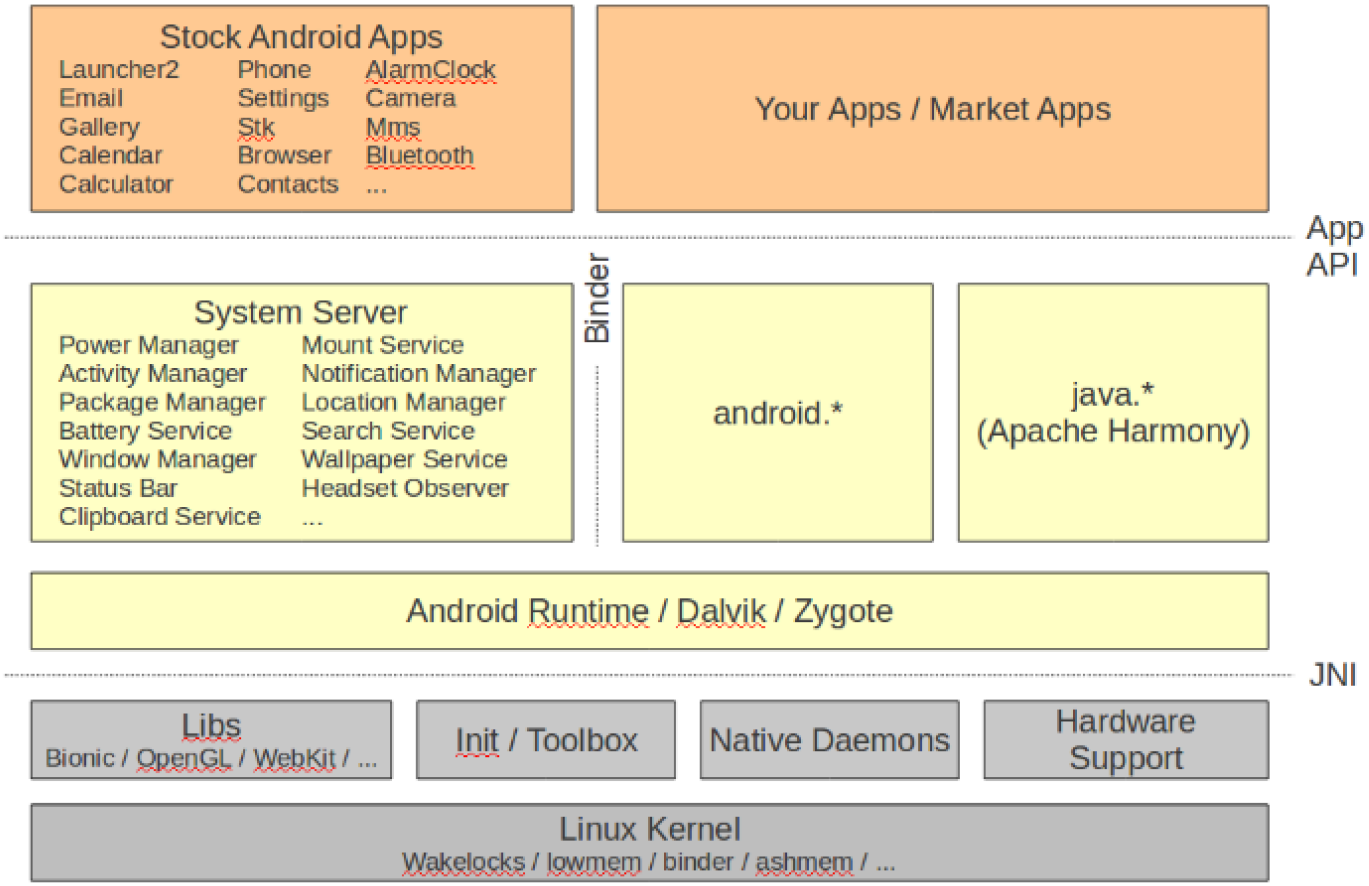
- A clean-room implementation of the JVM
- Sun Java
  - Java language, JDK compiler, JVM, JDK Libraries
  - java.*
- Android Java
  - Java language, JDK compiler, dx, Dalvik, Apache Harmony Libraries
  - Does not align to Java SE, Java ME
    - No AWT, Swing
  - Subset of java.*, android.*
- JVM interprets, executes .class files
- Dalvik interprets .dex files
  - Post-processes .class files
  - Size reduction, various optimisations
- Target slow cpu, no swap, low RAM, battery powered

Android Kernel

- Linux kernel
  - Not the same as a linux distribution
    - Ubuntu et al.
    - No standard libraries, UI etc
  - Bionic
    - GNU libc library variant
    - Legal / efficiency
- Android specific modifications
  - wakelocks – keep the phone awake
  - binder – interprocess communication
  - ashmem – shared memory
  - oom – kills processes when memory is low
  - alarm manager – wakes up the phone when necessary

Android Apps

- Applications are sandboxed
  - A security mechanism for separating running applications and data
  - Each application gets its sandbox in which to play
  - Why?
- Android application sandbox
  - Linux is a multi-user system
    - How many people use your phone at once?
  - Makes use of Linux permissions and security
    - Own process, own VM, own UID
    - Cannot access other application files / data / processes
      - Owner not generally given access to the root user
      - Root can access the entire system, hence “rooting” or “jailbreaking” a phone
    - Root != supervisor mode



SDK, Eclipse, .apk

Manifest:
Perms / SDK ver.

.dex, ddms

NDK, rootfs, initrc, adb

GNU toolchain

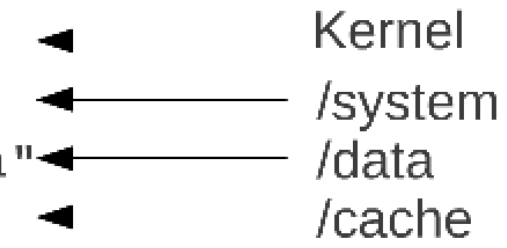
(fastboot)

Android Hardware Support

- Bluetooth - BlueZ
- GPS – Manufacturer provided libgps.so
- Wifi – wpa_supplicant
- Display – Standard framebuffer driver
- Keyboard – Standard input event
- Lights – Manufacturer provided liblights.so
- Audio – Manufacturer provided libaudio.so
- Camera – Manufacturer provided libcamera.so
- Power Management – “wakelocks” kernel patch
- Sensors – Manufacturer provided libsensors.so
- Radio – Manufacturer provided libril.so

Bootup

```
0x000003860000-0x000003900000 : "misc"
0x000003900000-0x000003e00000 : "recovery"
0x000003e00000-0x000004300000 : "boot"
0x000004300000-0x00000c300000 : "system"
0x00000c300000-0x0000183c0000 : "userdata"
0x0000183c0000-0x00001dd20000 : "cache"
0x00001dd20000-0x00001df20000 : "kpanic"
0x00001df20000-0x00001df60000 : "dinfo"
0x00001df60000-0x00001dfc0000 : "setupdata"
0x00001dfc0000-0x00001e040000 : "splash1"
0x000000300000-0x000001680000 : "modem"
```



Bootup

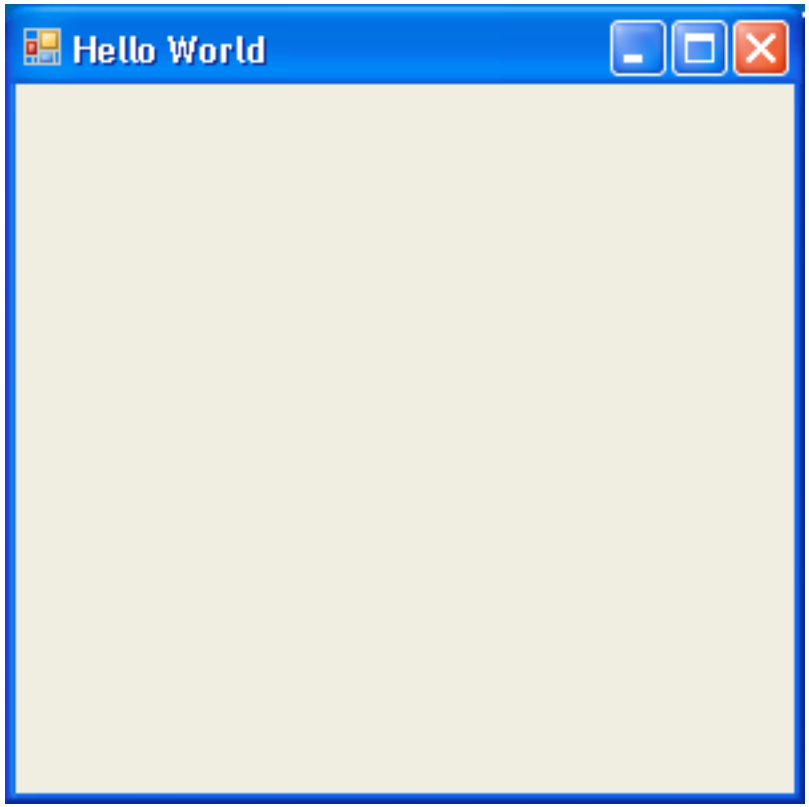
- Initialise the kernel
- Initialise device drivers
- Mount the root file system
- Execution of /init
  - Mount filesystems
  - Setup filesystem permissions
  - Set OOM properties
  - Start daemons
    - adbd
    - servicemanager (binder)
    - vold, netd, rild
    - app_process – Xzygote
    - …

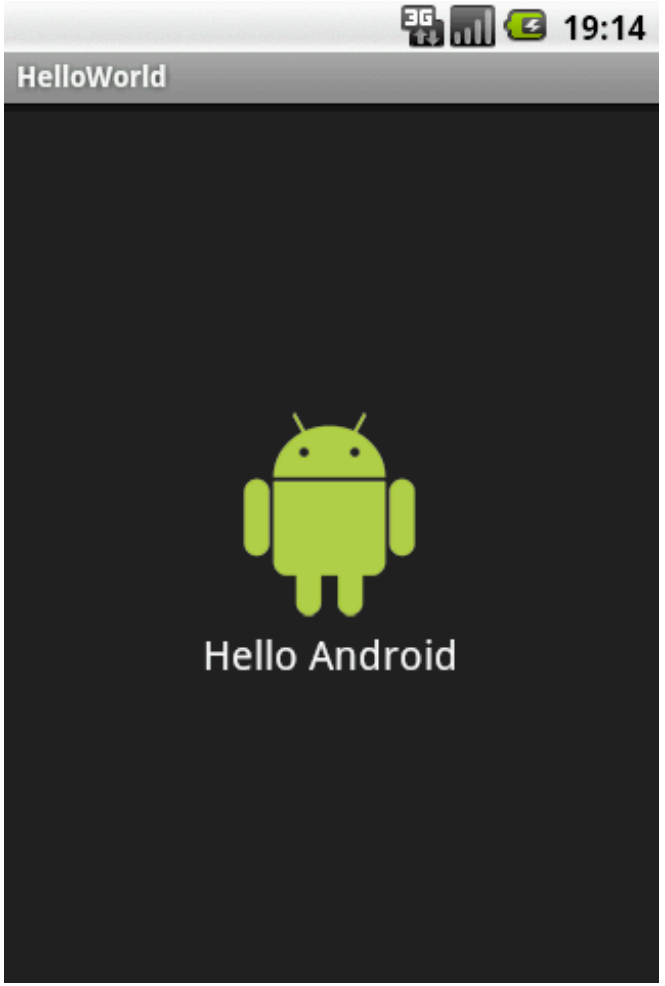
Zygote

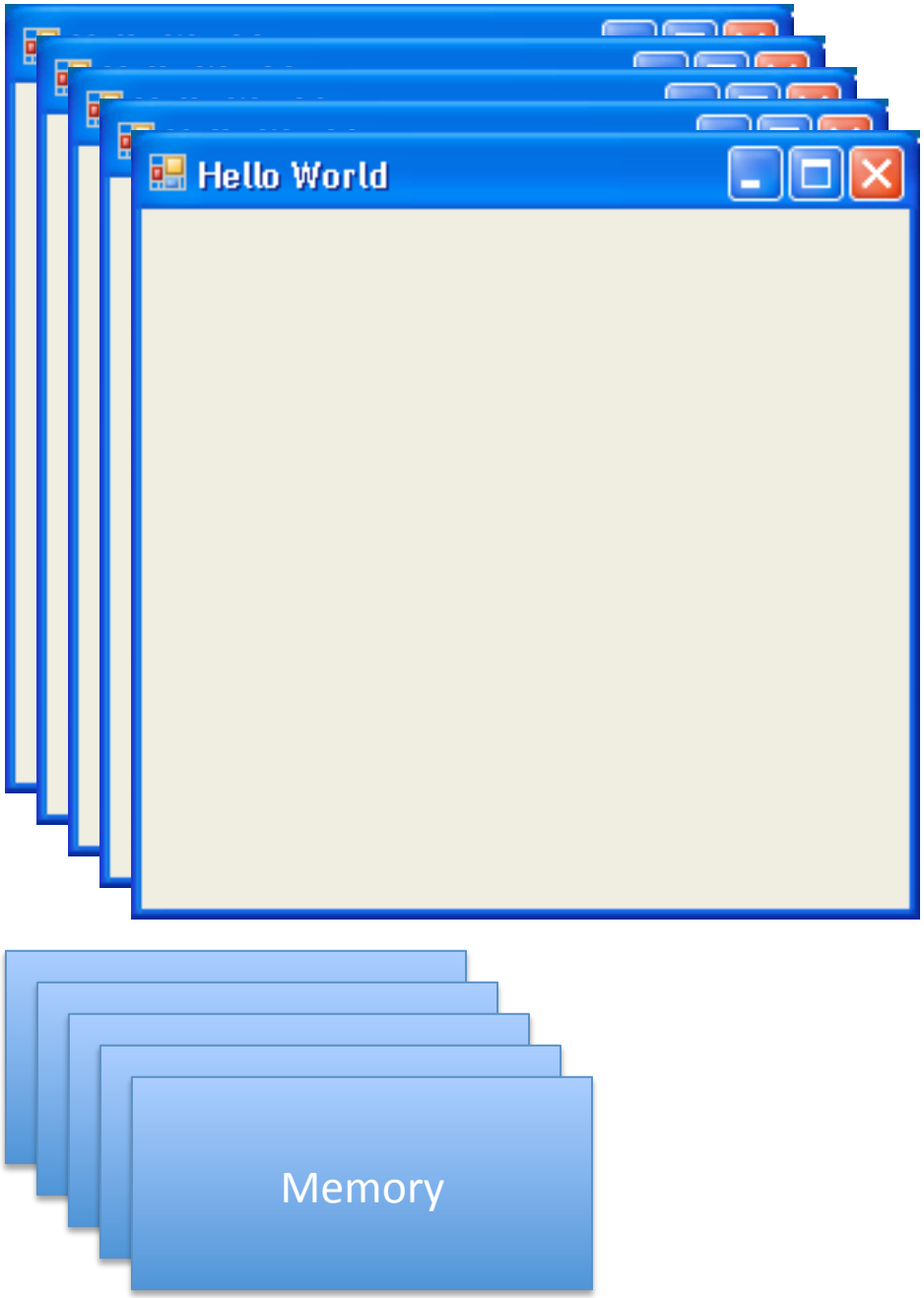
- “The first cell formed when an organism is produced.”
- app_main
  - Runtime.start(“com.android.internal.os.Zygote” …)
  - startVM()
  - Call Zygote’s main()
    - preloadClasses() ← what?
    - startSystemServer()
    - Call SystemServer’s run()
      - Start all system service/managers
      - Start ActivityManager
        - » Send Intent.CATEGORY_HOME
          - Launcher2

Shared Memory

- Load all java.*, android.* classes at boot time
- Initially create a single Dalvik VM process
  - Referencing classes loaded above
- When user runs an application…
  - onClick(Launcher)->startActivity(Activity.java)->Binder->ActivityManagerService->startViaZygote(Process.java)->Socket->Zygote)
  - **fork**
    - Creates a copy of itself in a separate address space
    - Does not copy **memory**, instead refers to original memory until modified
      - **Why?**


Hello World


19:14
HelloWorld
Hello Android


Hello World
Memory
19:14
HelloWorld
Hello Android
Memory