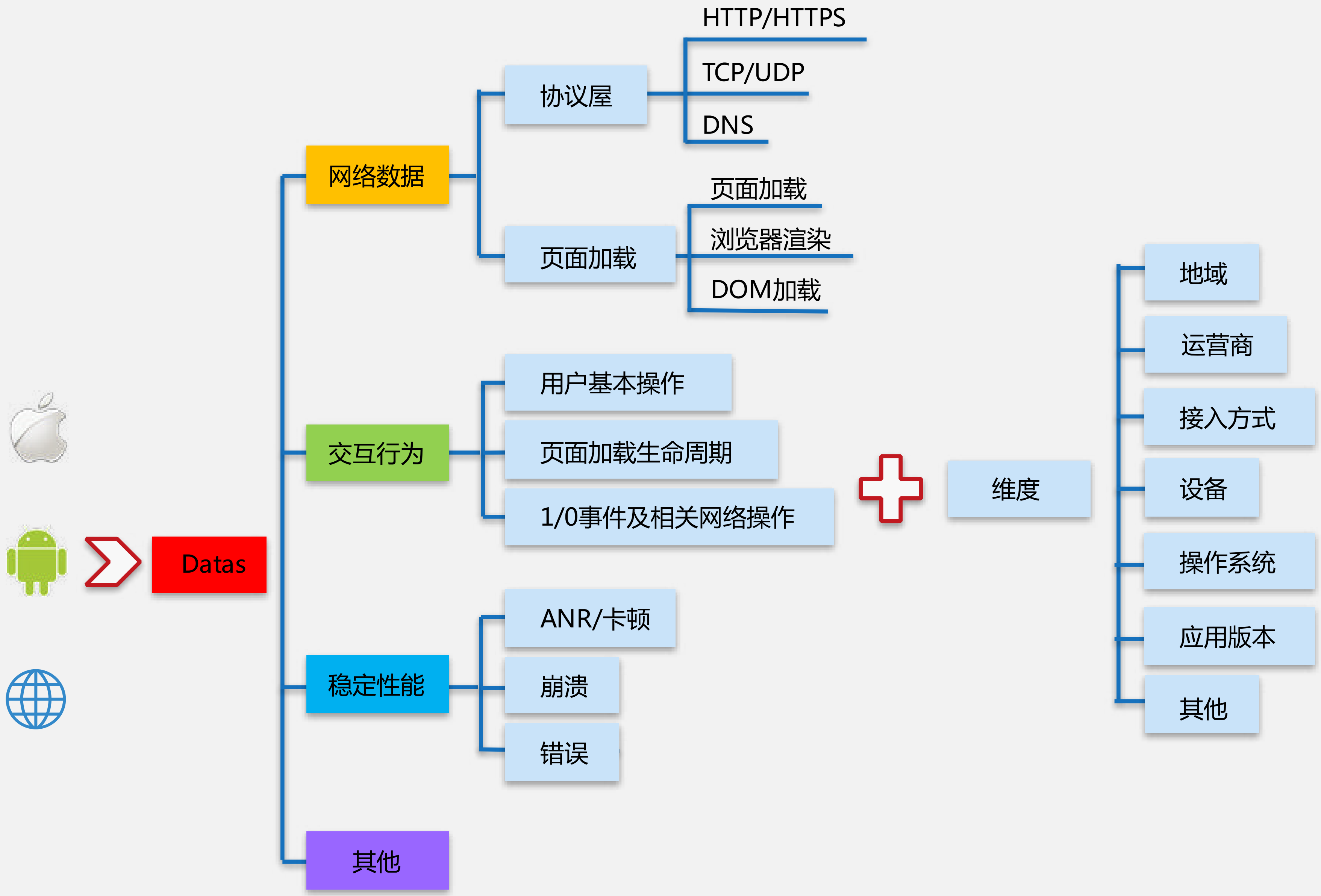
APMCon

移动端APM产品研发技能

江赛

APM概况

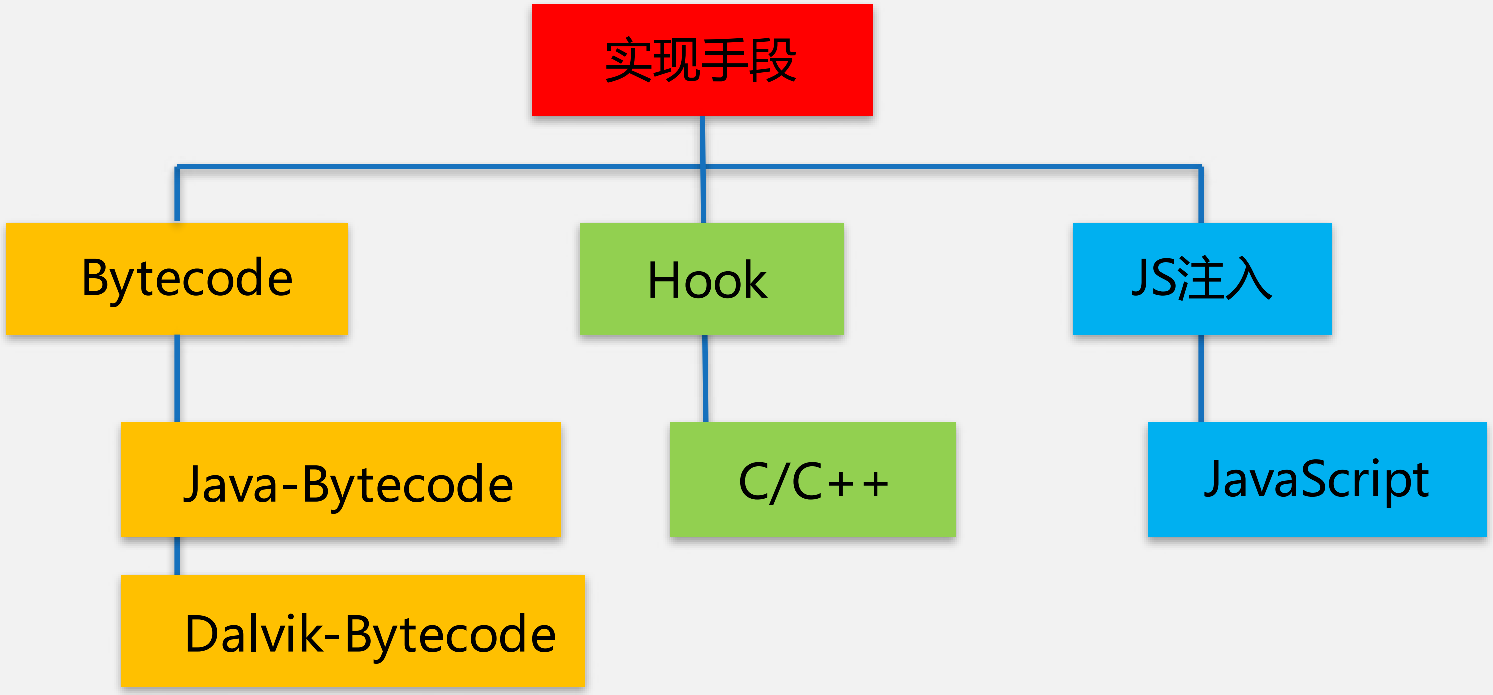


Datas
网络数据
协议屋
HTTP/HTTPS
TCP/UDP
DNS
页面加载
页面加载
浏览器渲染
DOM加载
交互行为
用户基本操作
页面加载生命周期
1/0事件及相关网络操作
稳定性能
ANR/卡顿
崩溃
错误
其他
维度
地域
运营商
接入方式
设备
操作系统
应用版本
其他

APM实现

APMCon

为了减少开发者的工作量，采用了自动埋点技术



APM实现-Bytecode

APMCon

一、从Java源代码到Dalvik Bytecode

APM实现-Bytecode

APMCon

.java -------------> .class ----------> .dex

javac dx

.java <------------- .class <------------ .dex

JD-GUI dex2jar

APM实现-Bytecode

Example Java source: Foo.java

```
 class Foo {

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

         public int method(int i1, int i2) {
                int i3 = i1 * i2;
                return i3 * 2;
        }
}
```

APM实现-Bytecode

```
$ javac Foo.java
$ javap -v Foo

 public int method(int, int);
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=4, args_size=3
         0: iload_1
         1: iload_2
         2: imul
         3: istore_3
         4: iload_3
         5: iconst_2
         6: imul
         7: ireturn
      LineNumberTable:
        line 6: 0
        line 7: 4
```

Stack

Before	**After**
value1	result
value2	…
…	…

（imul指令对栈的操作）

APM实现-Bytecode

```
$ dx --dex --output=Foo.dex Foo.class
$ dexdump -d Foo.dex


 Virtual methods   -
   #0              : (in LFoo;)
     name          : 'method'
     type          : '(II)I'
     access        : 0x0001 (PUBLIC)
     code          -
     registers     : 4
     ins           : 3
     outs          : 0
     insns size    : 5 16-bit code units
00018c:                                        |[00018c] Foo.method:(II)I
00019c: 9200 0203                              |0000: mul-int v0, v2, v3
0001a0: da00 0002                              |0002: mul-int/lit8 v0, v0, #int 2 // #02
0001a4: 0f00                                   |0004: return v0
     catches       : (none)
     positions     :
       0x0000 line=6
       0x0002 line=7
     locals        :
       0x0000 - 0x0005 reg=1 this LFoo;
```

9200 0203

92: mul-int

binop vAA, vBB, vCC

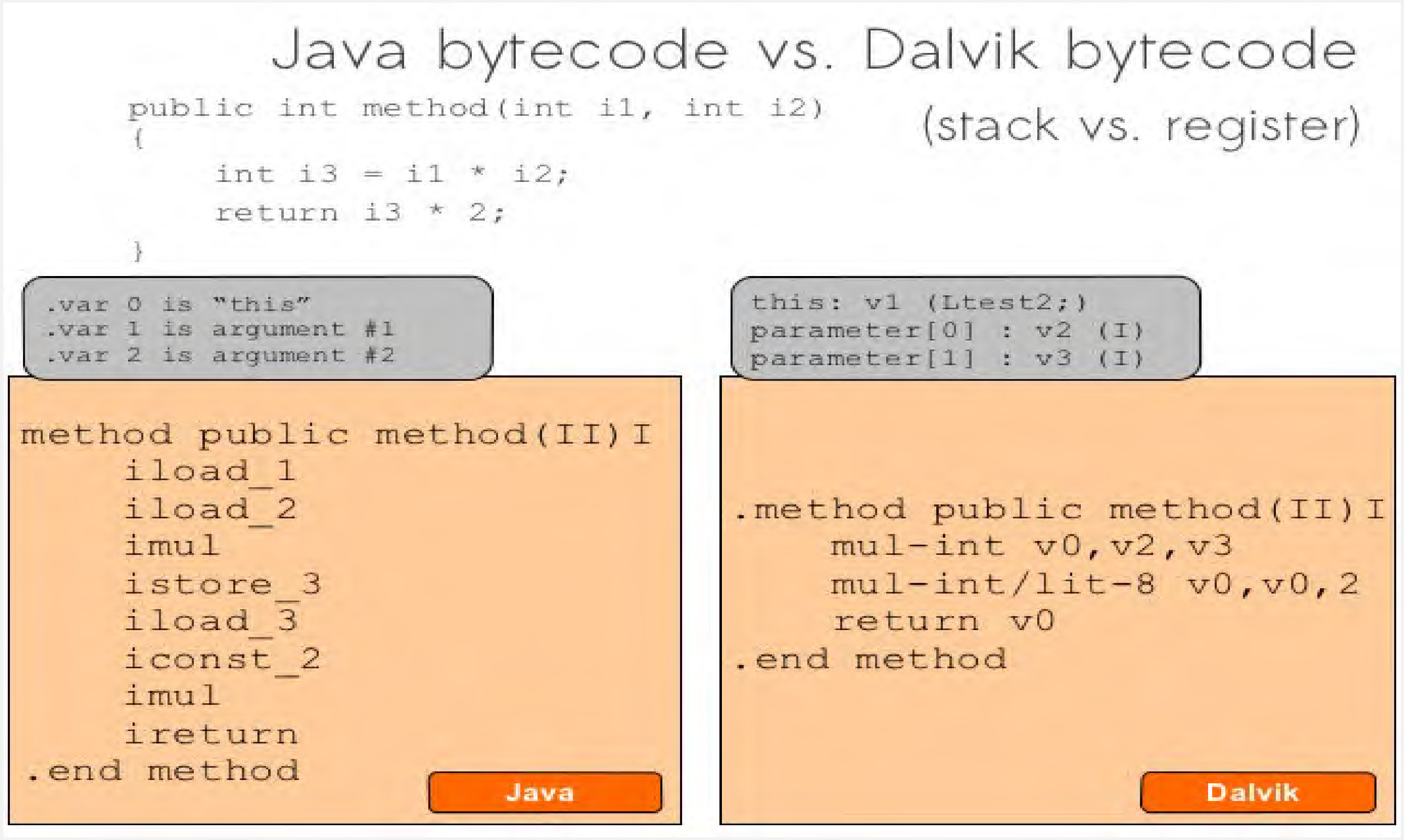
00: v0 (destination register)

02: v2 (first resource register)

03: v3 (second ...)

APM实现-Bytecode

APMCon



Java bytecode vs. Dalvik bytecode

(stack vs. register)

```
public int method(int i1, int i2)
{
      int i3 = i1 * i2;
      return i3 * 2;
}
```

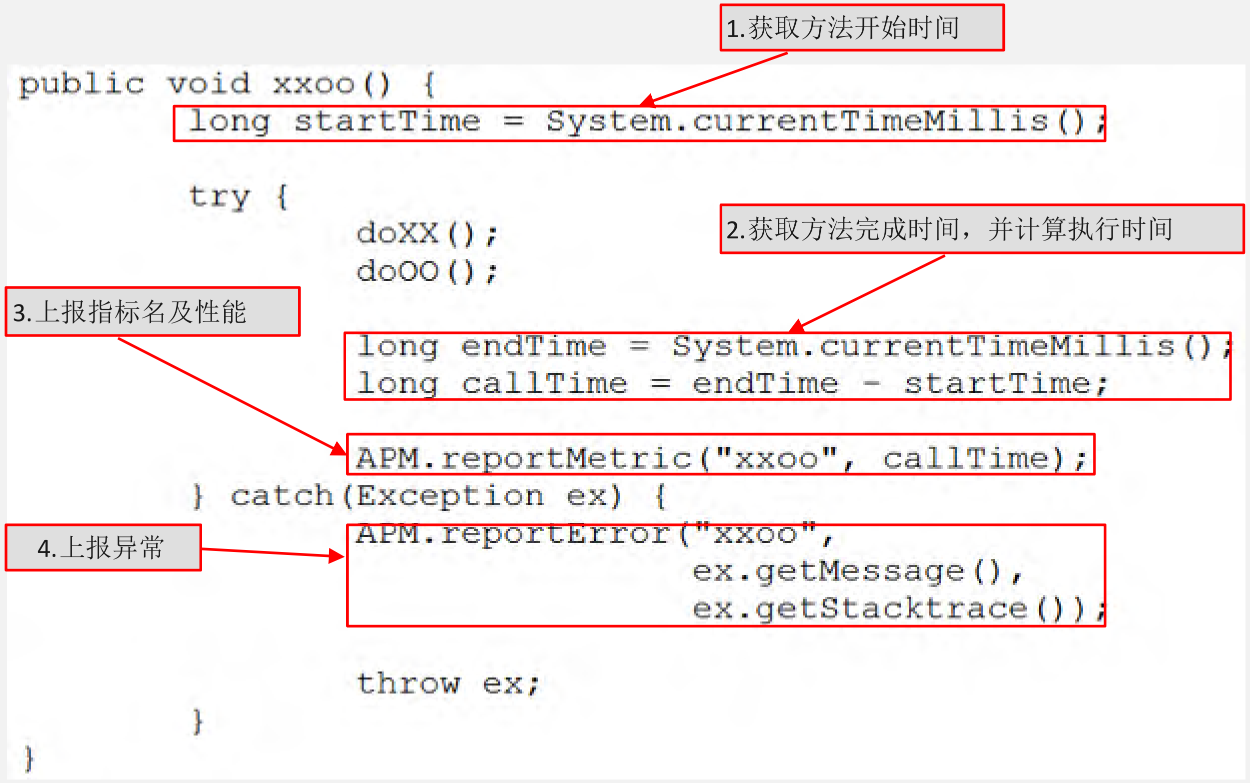
```
.var 0 is "this"
.var 1 is argument #1
.var 2 is argument #2
```

```
method public method(II)I
    iload_1
    iload_2
    imul
    istore_3
    iload_3
    iconst_2
    imul
    ireturn
.end method
```

```
this: v1 (Ltest2;)
parameter[0] : v2 (I)
parameter[1] : v3 (I)
```

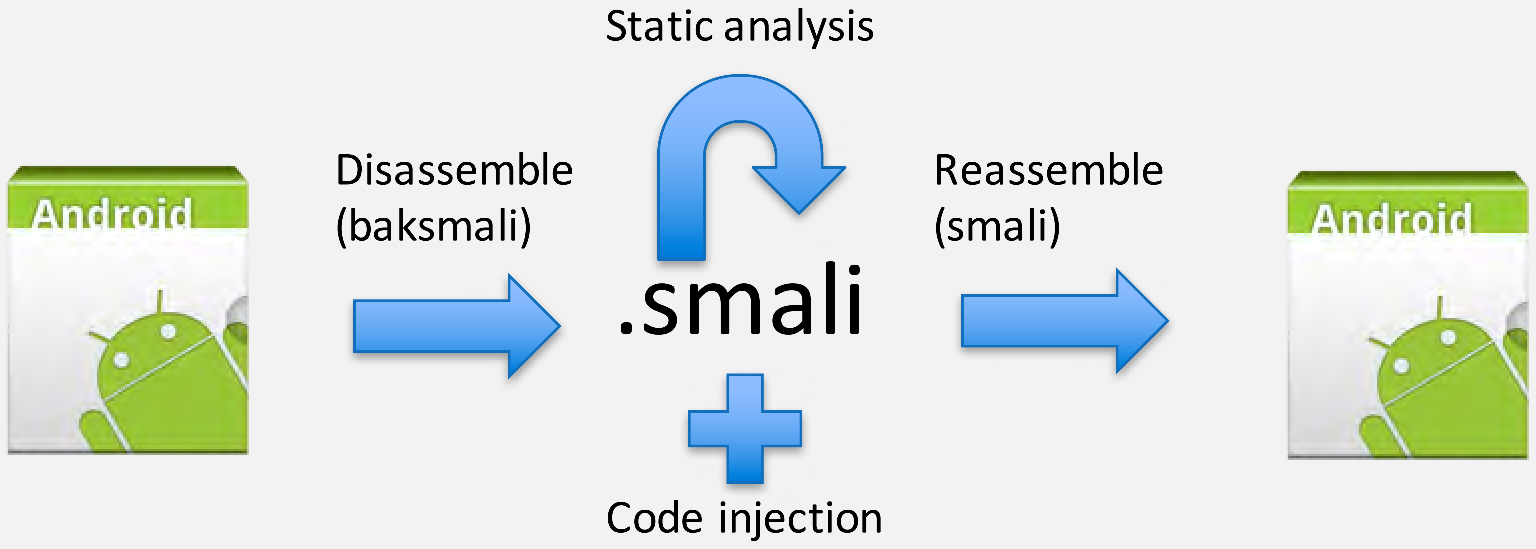
```
.method public method(II)I
    mul-int v0,v2,v3
    mul-int/lit-8 v0,v0,2
    return v0
.end method
```

APM实现-Bytecode



```
public void xxoo() {
        long startTime = System.currentTimeMillis();

        try {
                doXX();
                doOO();

                long endTime = System.currentTimeMillis();
                long callTime = endTime - startTime;

                APM.reportMetric("xxoo", callTime);
        } catch(Exception ex) {
                APM.reportError("xxoo",
                                ex.getMessage(),
                                ex.getStacktrace());

                throw ex;
        }
}
```

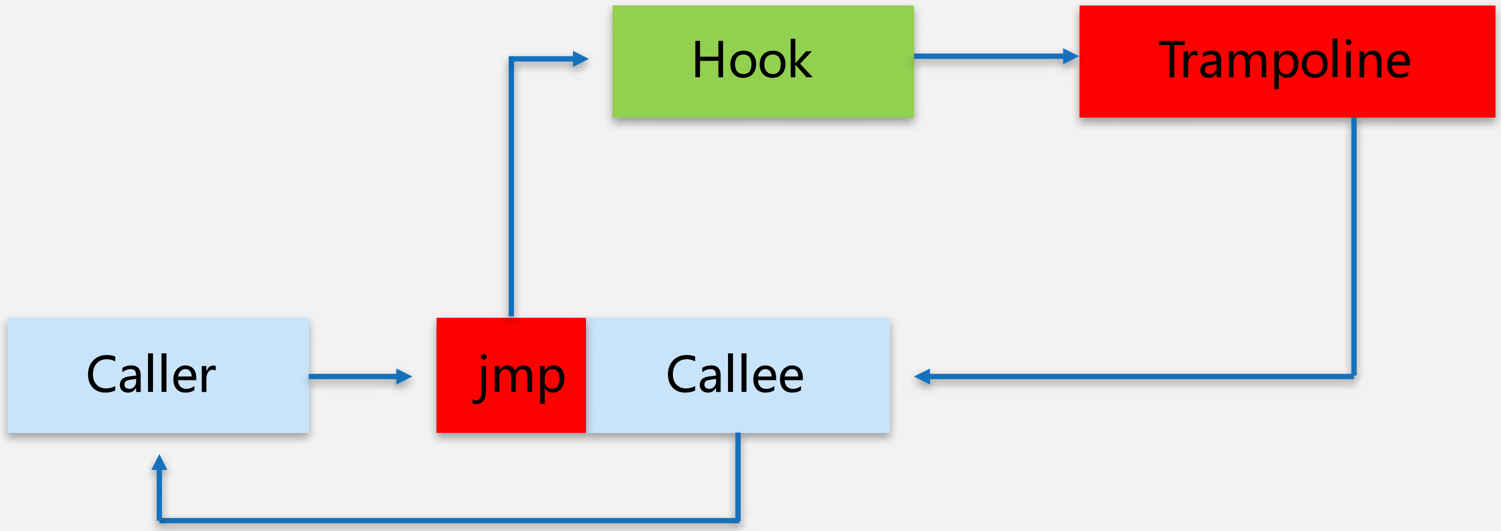
APM实现-Bytecode



APM实现-Bytecode

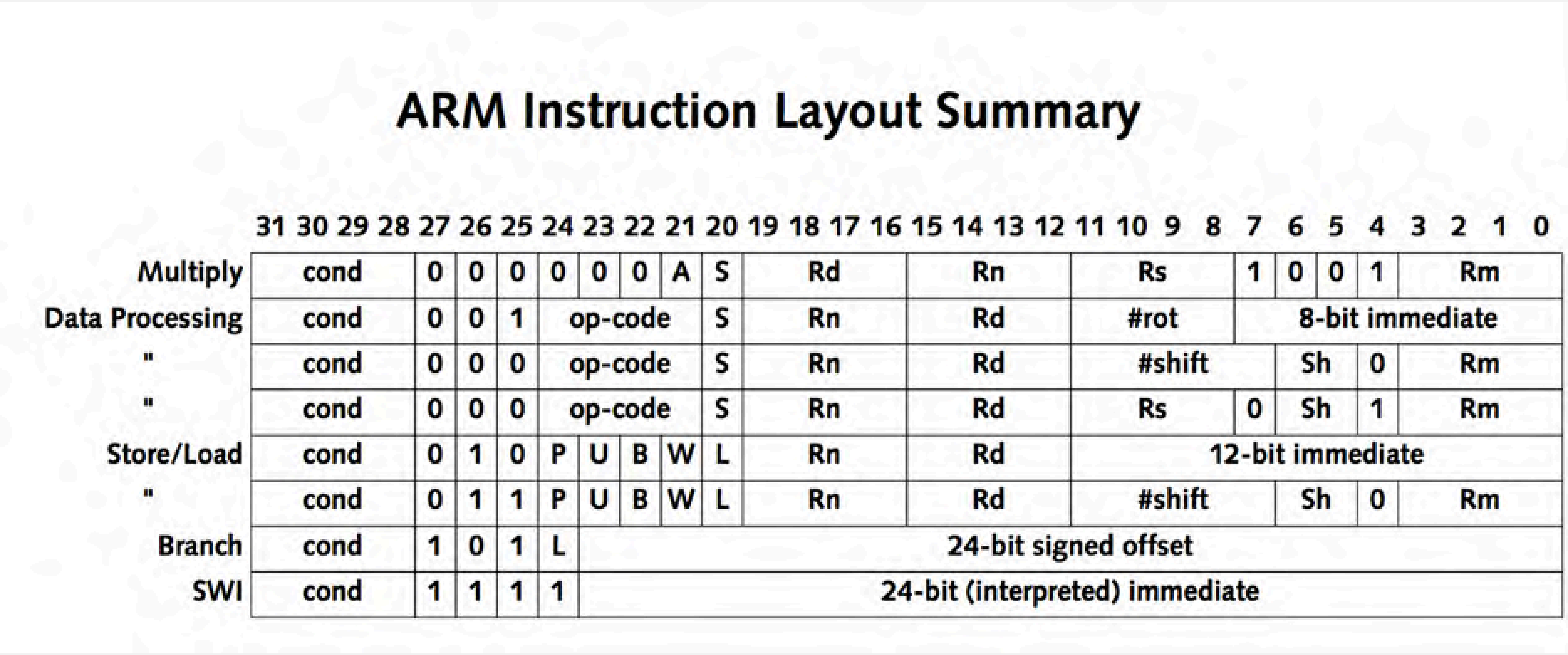
二、native inline hook

APM实现-Hook



Hook function is called,which in turn calls original function

APM实现-Hook



ARM Instruction Layout Summary

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Multiply	cond				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm			
Data Processing	cond				0	0	1	op-code				S	Rn				Rd				#rot				8-bit immediate							
"	cond				0	0	0	op-code				S	Rn				Rd				#shift					Sh		0	Rm			
"	cond				0	0	0	op-code				S	Rn				Rd				Rs				0	Sh		1	Rm			
Store/Load	cond				0	1	0	P	U	B	W	L	Rn				Rd				12-bit immediate											
"	cond				0	1	1	P	U	B	W	L	Rn				Rd				#shift					Sh		0	Rm			
Branch	cond				1	0	1	L	24-bit signed offset																							
SWI	cond				1	1	1	1	24-bit (interpreted) immediate																							

APM实现-Hook

APMCon

```
if ((instruction  & 0xF000000) == 0xA000000) {
        /*is B instruction*/
    address = PC + (SignExtend_30(signed_immed_24)  << 2)
       /*get absolutely address*/
}
```

B指令转换为等效指令

```
LDR PC, [PC, #-4]
0x.....   //Absolutely address
```

APM实现-Hook

- When an inline hook is implemented it will overwrite **the first two instructions** in order to redirect code flow;

  ARM instruction:
  ```
  LDR PC, [PC, #-4]
  addr
  ```

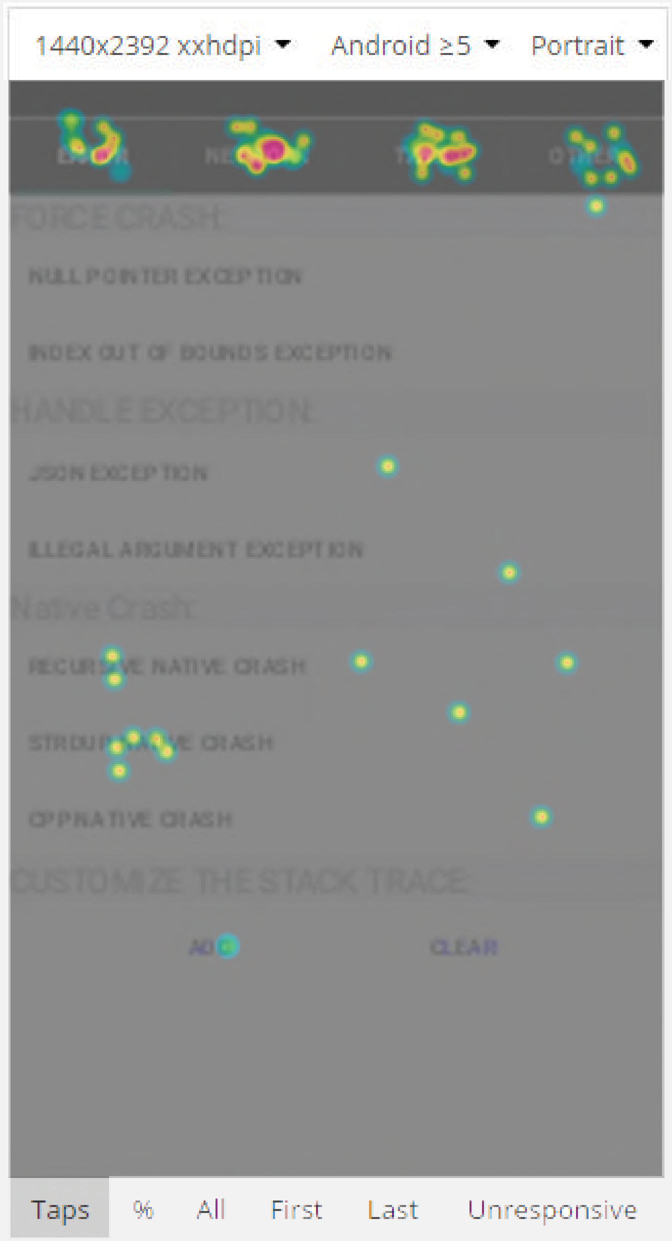
- Fix instruction which is PC-related;

APM产品

APMCon

97.6% Open App

2.4% Menu 1



1440x2392 xxhdpi ▾ Android ≥5 ▾ Portrait ▾

Taps % All First Last Unresponsive

85.7% Quit App

11.9% Crashed

2.4% Menu 1

THANK YOU