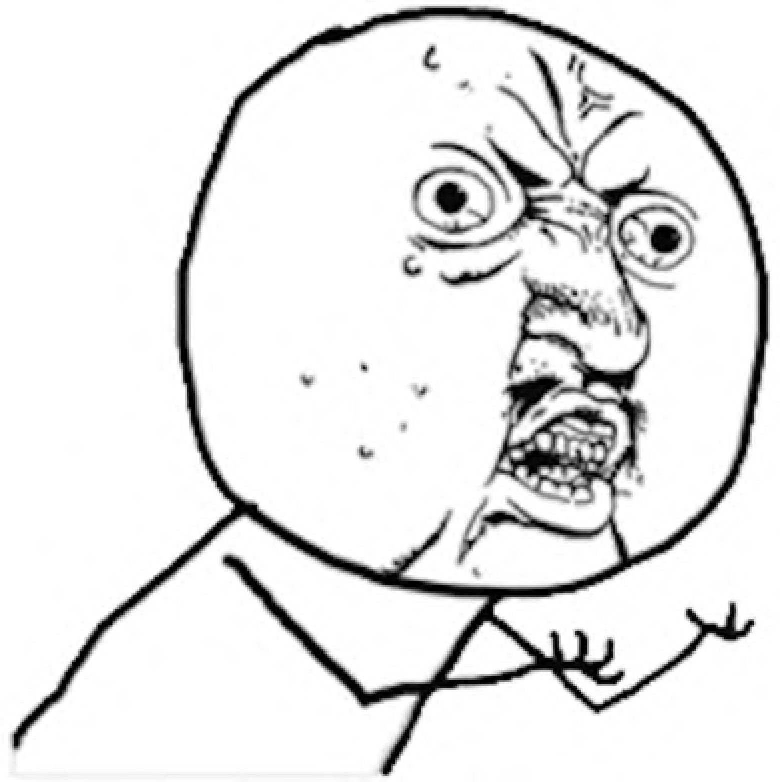
手淘iOS性能优化探索

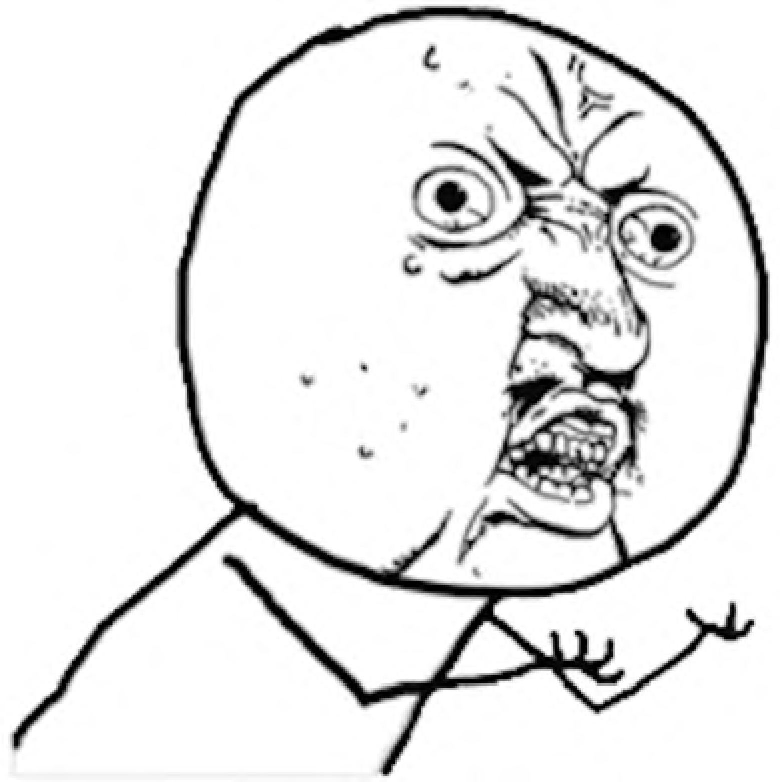
手淘基础架构 — 方颖（叁省）

淘宝网
Taobao.com

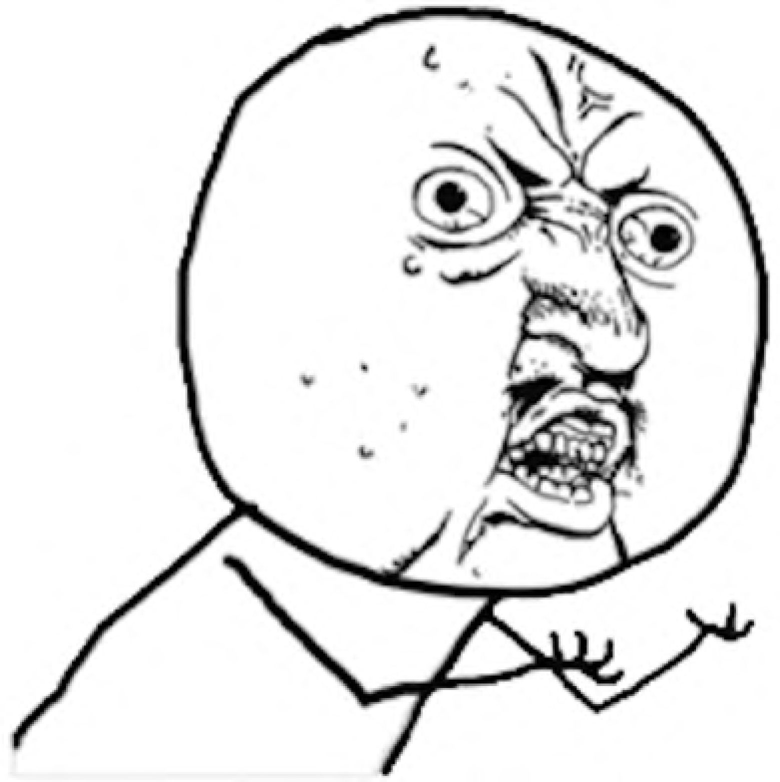
我们面对的每一天~~



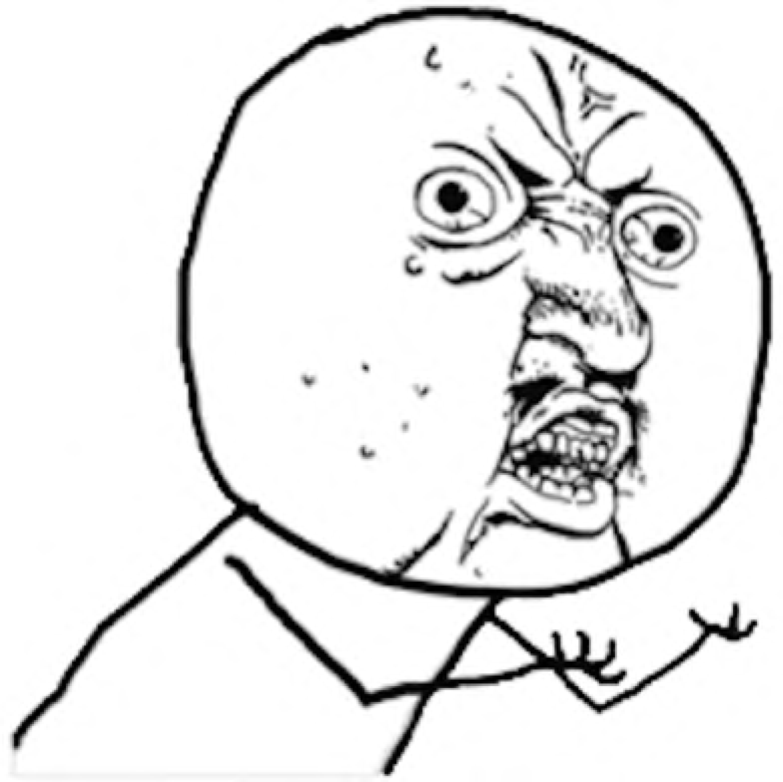
页面加载这么慢
都不知道吗~~~



怎么一个版本一个版
本性能越来越差了~~~



线上页面加载数据怎么这么卡~~~
有很多客户投诉，页面划不动啊~~~



启动不达标，不达标~~~

XXX~~~
XXX~~~
XXX~~~
~~~
XX~~~
~~~

不知道！

一脸懵逼~~~

性能面对的问题

研发流程如何自动防止性能衰退

线上问题如何排查

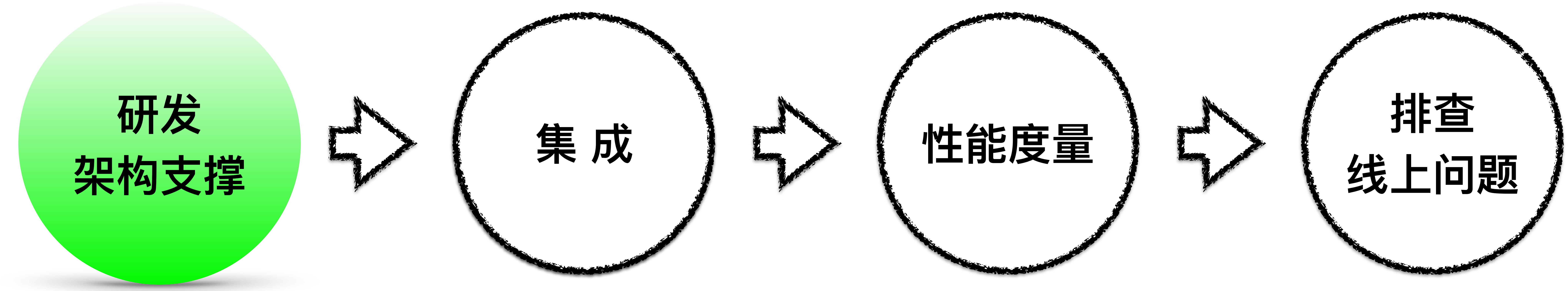
性能如何度量

性能监控是否能真实反映用户体验

版本质量如何保证

架构如何支撑性能

技术上从研发流程角度的思考



- App启动器

研发架构沉淀 — App启动器设计目标

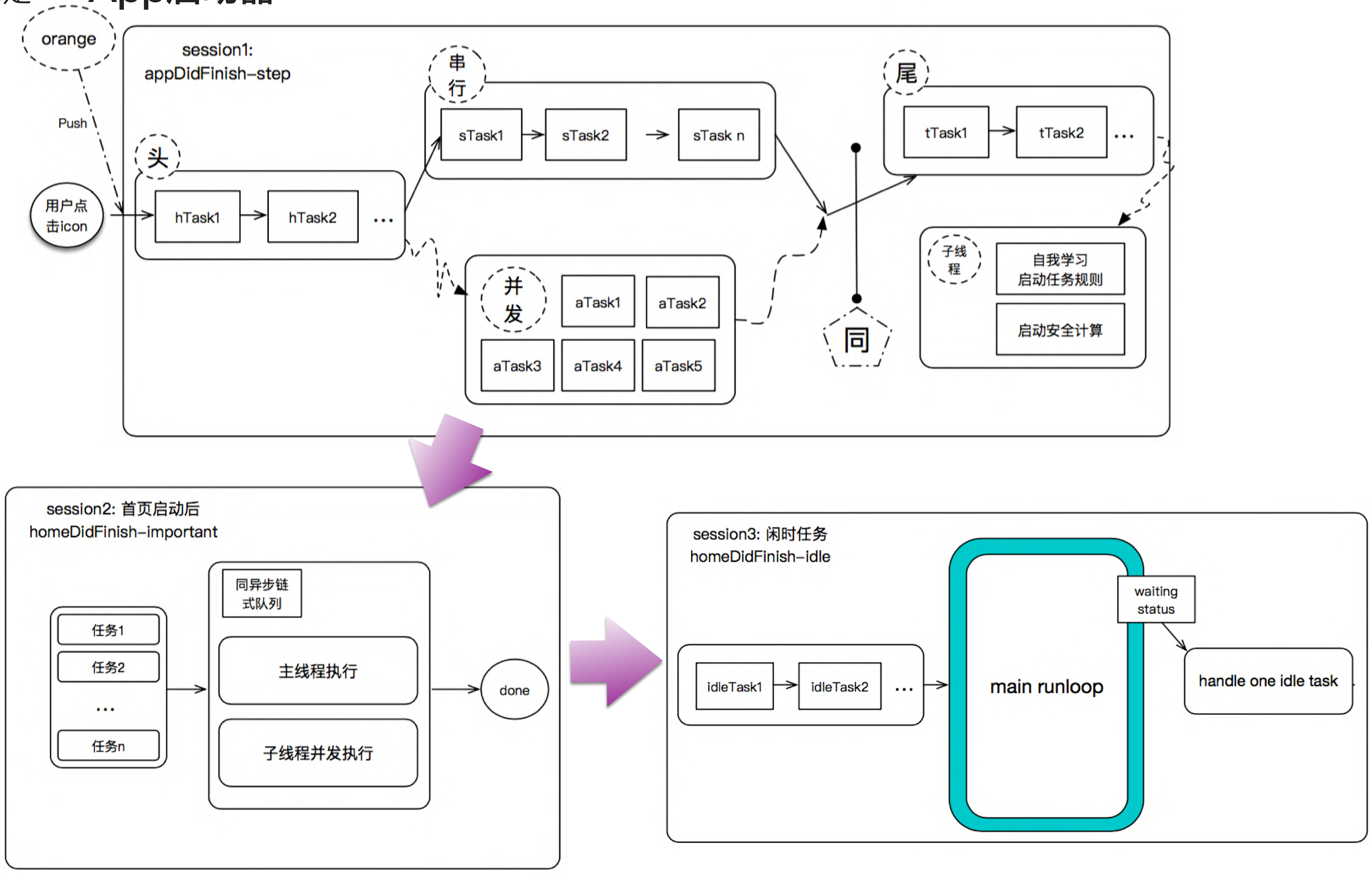
问题篇：

1. 启动过程任务数量多，并且复杂/凌乱
2. 版本持续迭代，启动任务任意增加，维护成本高
3. 启动性能不宜被管控，未知任务容易导致性能下降
4. 稳定性容易受到挑战，任意加入启动逻辑，可能造出闪退
5. 启动过程业务逻辑严重耦合

设计目标：

1. 采取配置信息，解决耦合问题，达到启动任务松耦合
2. 启动任务细粒子化，通过高并发设计、自学习最佳任务顺序，提高启动性能
3. 启动任务服务端可配置，保证线上问题服务端控制解决
4. 启动任务严格管控，业务任务接入需要接受审核

研发架构沉淀 — App启动器



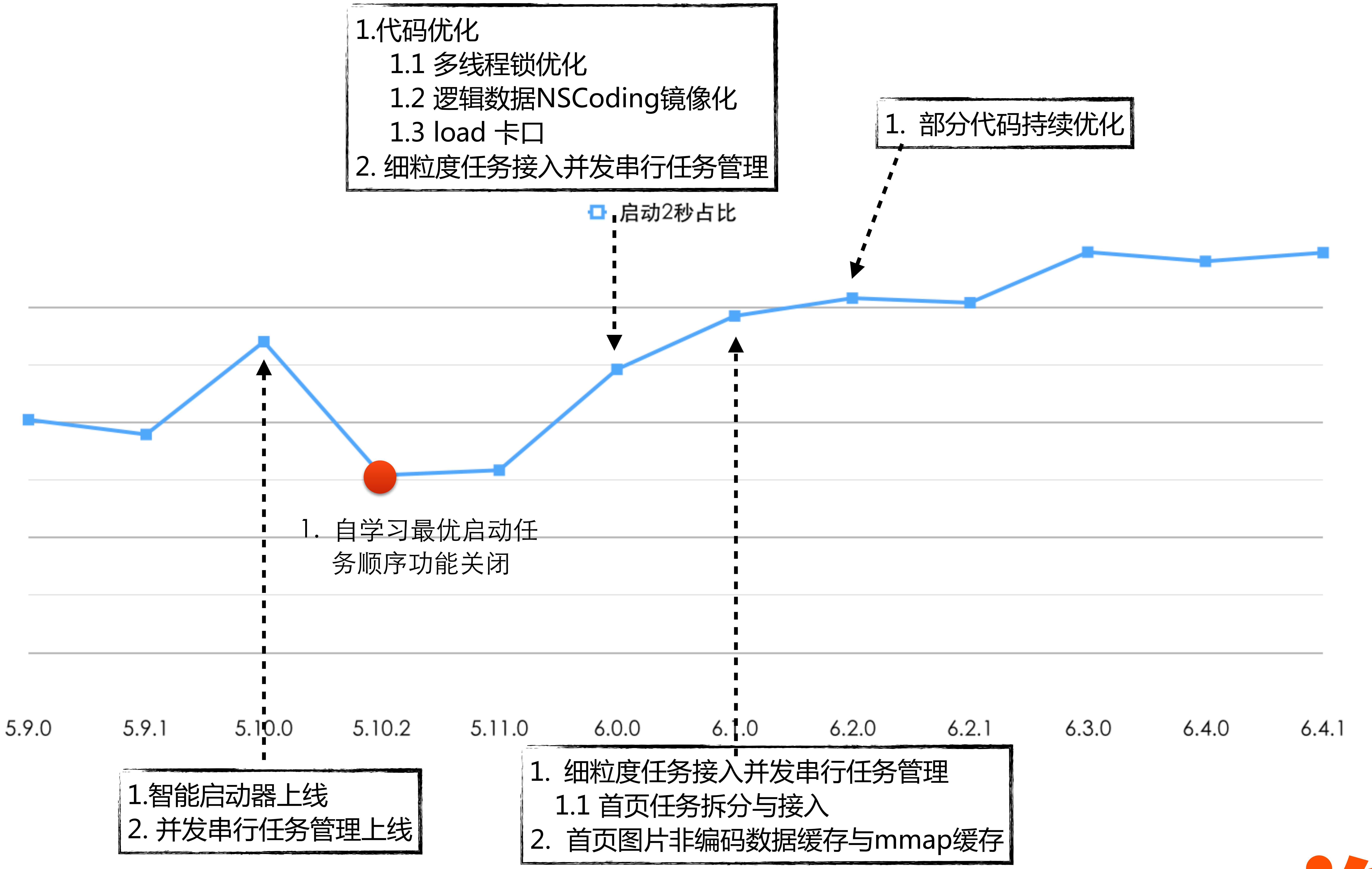
研发架构沉淀 — App启动器的效果

横轴:

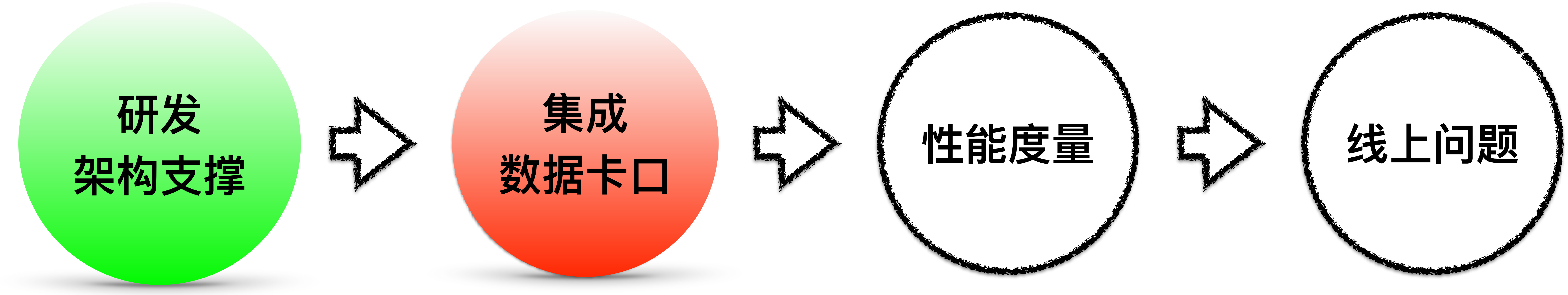
手淘版本号

纵轴:

启动耗时2秒用户占比



技术上从研发流程角度的思考



- API使用卡口
- 性能维度卡口

集成数据卡口 — **非最佳使用API卡口案例**

developer 在main之前的可以进行的处理

Load消耗具体时间

285.0ms	2.9%	0.0	▼call_load_methods libobjc.A.dylib
162.0ms	1.6%	0.0	[redacted]obao4iPhone
34.0ms	0.3%	0.0	[redacted] load] Taobao4iPhone
15.0ms	0.1%	1.0	[redacted]ad] Taobao4iPhone
12.0ms	0.1%	1.0	▶[redacted]r load] Taobao4iPhone
10.0ms	0.1%	1.0	[redacted]] Taobao4iPhone
6.0ms	0.0%	0.0	[redacted] Taobao4iPhone
5.0ms	0.0%	0.0	▶xmlInitParser libxml2.2.dylib

集成数据卡口 — **非最佳使用API卡口案例**

```objc
#import "TestFramework.h"

@implementation TestFramework


+ (void)load {
    NSLog(@"TestFramework");
}

@end
```

```
yingfang:Documents fangying$ nm libTestFramework.a

libTestFramework.a(TestFramework.o):
0000000000000040 t +[TestFramework load]
                 U _NSLog
                 U _OBJC_CLASS_$_NSObject
00000000000001a8 S _OBJC_CLASS_$_TestFramework
                 U _OBJC_METACLASS_$_NSObject
0000000000000180 S _OBJC_METACLASS_$_TestFramework
                 U ___CFConstantStringClassReference
                 U __objc_empty_cache
0000000000000020 T _after_main
0000000000000000 T _before_main
                 U _printf
00000000000000d0 s l_OBJC_$_CLASS_METHODS_TestFramework
0000000000000138 s l_OBJC_CLASS_RO_$_TestFramework
00000000000000f0 s l_OBJC_METACLASS_RO_$_TestFramework
```

```objc
#import <Foundation/Foundation.h>

__attribute__((constructor)) void before_main() {
    printf("before main\n");
}

__attribute__((destructor)) void after_main() {
    printf("after main\n");
}

@interface TestFramework : NSObject

@end
```

```
Section
  sectname __mod_init_func
   segname __DATA
      addr 0x00000000000001e0
      size 0x0000000000000008
    offset 3016
     align 2^3 (8)
    reloff 5368
    nreloc 1
     flags 0x00000009
 reserved1 0
 reserved2 0
Section
  sectname __mod_term_func
   segname __DATA
      addr 0x00000000000001e8
      size 0x0000000000000008
    offset 3024
     align 2^3 (8)
    reloff 5376
    nreloc 1
     flags 0x0000000a
 reserved1 0
 reserved2 0
```

集成数据卡口 — **非最佳使用API卡口案例**　　如何去统计和定位App load函数的耗时?

```
#import "TestFramework.h"

@implementation TestFramework


+ (void)load {
    NSLog(@"TestFramework");
}

@end
```

```
void initializeMainExecutable() {
    // record that we've reached this step
    gLinkContext.startedInitializingMainExecutable = true;
    // run initialzers for any inserted dylibs
    ImageLoader::InitializerTimingList initializerTimes[sAllImages.
    initializerTimes[0].count = 0;
    const size_t rootCount = sImageRoots.size();
    if ( rootCount > 1 ) {
        for(size_t i=1; i < rootCount; ++i) {
            sImageRoots[i]->runInitializers(gLinkContext, initializerTimes[0]);
        }
    }
    // run initializers for main executable and everything it bring
    sMainExecutable->runInitializers(gLinkContext, initializerTimes
    // register cxa_atexit() handler to run static terminators in a
    if ( gLibSystemHelpers != NULL )
        (*gLibSystemHelpers->cxa_atexit)(&runAllStaticTerminators, NULL, NULL);
    // dump info if requested
    if ( sEnv.DYLD_PRINT_STATISTICS )
        ImageLoaderMachO::printStatistics((unsigned int)sAllImages.size(), initializerTimes[0]);
}
```

初始化依赖动态库

初始化mainExec静态库

集成数据卡口 — **非最佳使用API卡口案例**　　如何去统计和定位App load函数的耗时?

```
void ImageLoader::recursiveInitialization(const LinkContext& context, mach_port_t this_thread,
                                          InitializerTimingList& timingInfo, UninitedUpwards& uninitUps)
{
    ...

    if ( fState < dyld_image_state_dependents_initialized-1 ) {

        ...

            context.notifySingle(dyld_image_state_dependents_initialized, this);

            // initialize this image
            bool hasInitializers = this->doInitialization(context);

            // let anyone know we finished initializing this image
            fState = dyld_image_state_initialized;
            oldState = fState;
            context.notifySingle(dyld_image_state_initialized, this);

            ...
    }
    ...
}
```

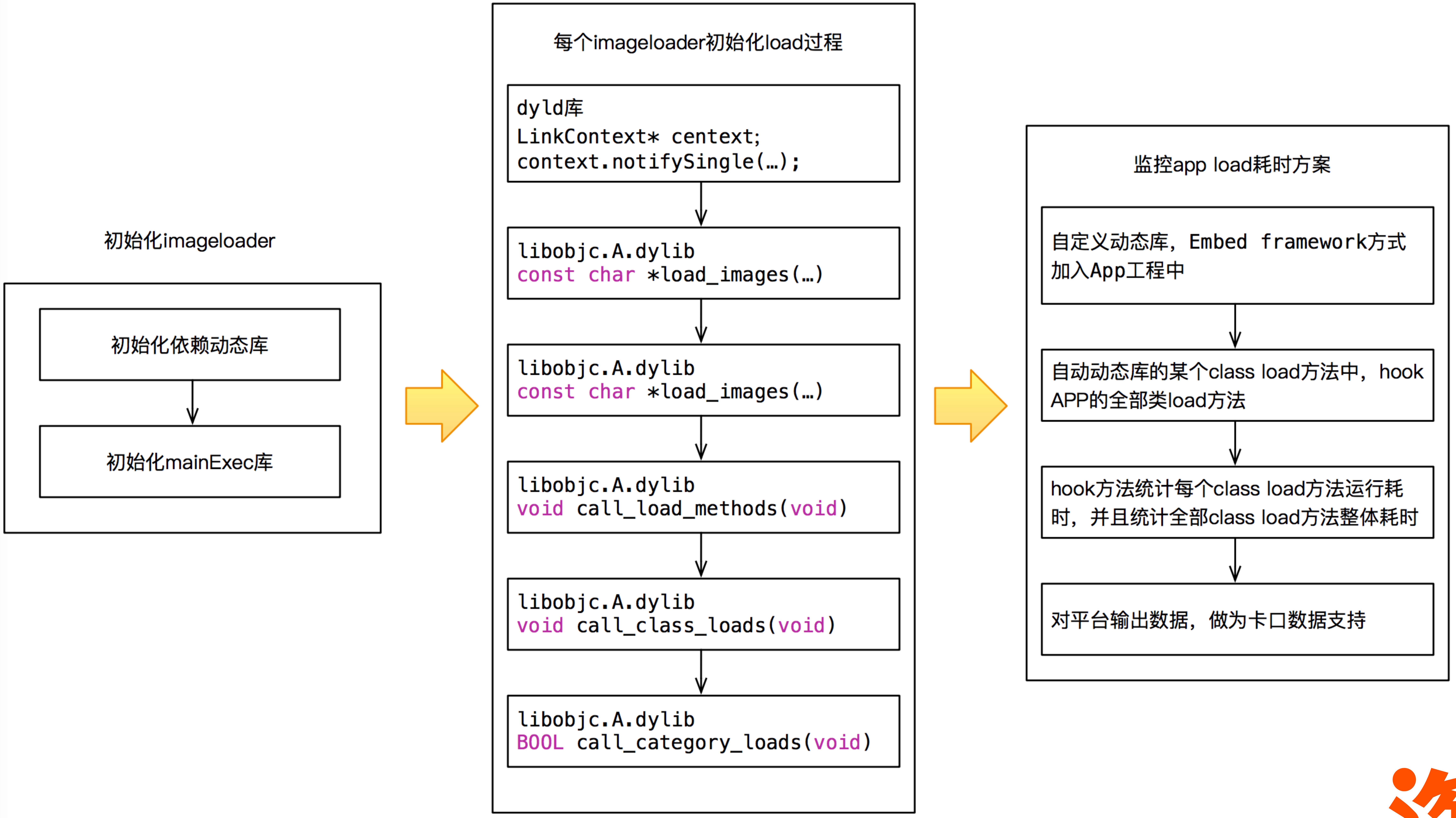
内部执行 load 方法

```
const char *
load_images(image_state state, uint32_t infoCount,
            const struct dyld_image_info infoList[])
{
    BOOL found;

    recursive_mutex_lock(&loadMethodLock);

    // Discover load methods
    rwlock_write(&runtimeLock);
    found = load_images_nolock(state, infoCount, infoList);
    rwlock_unlock_write(&runtimeLock);

    // Call +load methods (without runtimeLock - re-entrant)
    if (found) {
        call_load_methods();
    }

    recursive_mutex_unlock(&loadMethodLock);

    return NULL;
}
```

```
static void notifySingle(dyld_image_states state, const ImageLoader* image)
{
    //dyld::log("notifySingle(state=%d, image=%s)\n", state, image->getPath());
    std::vector<dyld_image_state_change_handler>* handlers = stateToHandlers(state, sSingleHandlers);
    if ( handlers != NULL ) {
        dyld_image_info info;
        info.imageLoadAddress   = image->machHeader();
        info.imageFilePath      = image->getRealPath();
        info.imageFileModDate   = image->lastModified();
        for (std::vector<dyld_image_state_change_handler>::iterator it = handlers->begin(); it != handlers->end(); ++it) {
            const char* result = (*it)(state, 1, &info);
            if ( (result != NULL) && (state == dyld_image_state_mapped) ) {
                //fprintf(stderr, "  image rejected by handler=%p\n", *it);
                // make copy of thrown string so that later catch clauses can free it
                const char* str = strdup(result);
                throw str;
            }
        }
    }
}
```

综合结论：动态库load方法调用，早与主二进制所有load方法调用

集成数据卡口 — **非最佳使用API卡口案例**　如何去统计和定位App load函数的耗时?



集成数据卡口 — **非最佳使用API卡口案例** 监控C++静态对象构造函数和__attribute__((constructor))的耗时?

```
void ImageLoaderMachO::doModInitFunctions(const LinkContext& context)
{
    if ( fHasInitializers ) {
        ...
        for (uint32_t i = 0; i < cmd_count; ++i) {
            if ( cmd->cmd == LC_SEGMENT_COMMAND ) {
                const struct macho_segment_command* seg = (struct macho_segment_command*)cmd;
                const struct macho_section* const sectionsStart = (struct macho_section*)((char*)seg + sizeof(struct macho_segment_command));
                const struct macho_section* const sectionsEnd = &sectionsStart[seg->nsects];
                for (const struct macho_section* sect=sectionsStart; sect < sectionsEnd; ++sect) {
                    const uint8_t type = sect->flags & SECTION_TYPE;
                    if ( type == S_MOD_INIT_FUNC_POINTERS ) {
                        Initializer* inits = (Initializer*)(sect->addr + fSlide);
                        const size_t count = sect->size / sizeof(uintptr_t);
                        for (size_t i=0; i < count; ++i) {
                            Initializer func = inits[i];
                            ...
                            func(context.argc, context.argv, context.envp, context.apple, &context.programVars);
                        }
                    }
                }
            }
            cmd = (const struct load_command*)(((char*)cmd)+cmd->cmdsize);
        }
    }
}
```

TYPE==S_MOD_INIT_FUNC_POINTERS
对应Mach-O文件中具体的section段

```
        ; Section __mod_init_func
        ; Range: [0x100098b78; 0x100098bb0[ (56 bytes)
        ; File offset : [625528; 625584[ (56 bytes)
        ; Flags: 0x9
        ;   S_MOD_INIT_FUNC_POINTERS

0000000100098b78         dq         0x0000000100060490
0000000100098b80         dq         0x0000000100063e88
0000000100098b88         dq         0x000000010007b7c8
0000000100098b90         dq         0x000000010007e524
0000000100098b98         dq         0x000000010007e554
0000000100098ba0         dq         0x000000010007e57c
0000000100098ba8         dq         0x000000010007e670
```

集成数据卡口 — **非最佳使用API卡口案例** 监控C++静态对象构造函数和__attribute__((constructor))的耗时?

实验1

```
class clsA {
public:
    clsA();
    int a;
};
clsA::clsA() {
    a = 10;
}
static clsA objA;
clsA objB;
```

_TEXT段

```
                        ; Section __mod_init_func
                        ; Range: [0x260; 0x268[ (8 bytes)
                        ; File offset : [3232; 3240[ (8 bytes)
                        ; Flags: 0x9
                        ;   S_MOD_INIT_FUNC_POINTERS

                              ltmp11:
0000000000000260                   dq         0x00000000000000cc

                    _
                  ltmp1:
000000000000007     sub        sp, sp, #0x20                    ; CODE XREF=__GLOBAL__sub_I_TestFramework.mm+20
00000000000008      stp        x29, x30, [sp, #0x10]
0000000000000084    add        x29, sp, #0x10
0000000000000088    adrp       x8, #0x0
000000000000008c    add        x0, x8, #0xc48
0000000000000090    bl         __ZN4clsAC1Ev                    ; clsA::clsA()
0000000000000094    str        x0, [sp, #0x8]
0000000000000098    ldp        x29, x30, [sp, #0x10]
000000000000009c    add        sp, sp, #0x20
00000000000000a0    ret
                  ___cxx_global_var_init.1:
00000000000000a     sub        sp, sp, #0x20                    ; CODE XREF=__GLOBAL__sub_I_TestFramework.mm+24
00000000000000a8    stp        x29, x30, [sp, #0x10]
00000000000000ac    add        x29, sp, #0x10
00000000000000b0    adrp       x8, #0x0
00000000000000b4    add        x0, x8, #0xc4c
00000000000000b8    bl         __ZN4clsAC1Ev                    ; clsA::clsA()
00000000000000bc    str        x0, [sp, #0x8]
00000000000000c0    ldp        x29, x30, [sp, #0x10]
00000000000000c4    add        sp, sp, #0x20
00000000000000c8    ret
                  __GLOBAL__sub_I_TestFramework.mm:
00000000000000cc    sub        sp, sp, #0x20
00000000000000d0    stp        x29, x30, [sp, #0x10]
00000000000000d4    add        x29, sp, #0x10
00000000000000d8    bl         _objc_autoreleasePoolPush
00000000000000dc    str        x0, [sp, #0x8]
00000000000000e0    bl         ltmp1
00000000000000e4    bl         ___cxx_global_var_init.1
00000000000000e8    ldr        x0, [sp, #0x8]
00000000000000ec    bl         _objc_autoreleasePoolPop
00000000000000f0    ldp        x29, x30, [sp, #0x10]
00000000000000f4    add        sp, sp, #0x20
00000000000000f8    ret
```

集成数据卡口 — **非最佳使用API卡口案例** 监控C++静态对象构造函数和__attribute_((constructor))的耗时?

实验2

```
__attribute__((constructor)) void before_main0() {
    printf("before main\n");
}
__attribute__((constructor)) void before_main1() {
    printf("before main\n");
}
```

```
; Section __mod_init_func
; Range: [0x2b8; 0x2d0[ (24 bytes)
; File offset : [3320; 3344[ (24 bytes)
; Flags: 0x9
;   S_MOD_INIT_FUNC_POINTERS

                        ltmp11:
00000000000002b8        dq          0x0000000000000000     1
00000000000002c0        dq      2   0x0000000000000028
00000000000002c8        dq          0x000000000000011c
```

```
                     1  ltmp0:
0000000000000000            sub         sp, sp, #0x20
0000000000000004            stp         x29, x30, [sp, #0x10]
0000000000000008            add         x29, sp, #0x10
000000000000000c            adrp        x0, #0x0
0000000000000010            add         x0, x0, #0x14c
0000000000000014            bl          _printf
0000000000000018            stur        w0, [x29, #-0x4]
000000000000001c            ldp         x29, x30, [sp, #0x10]
0000000000000020            add         sp, sp, #0x20
0000000000000024            ret
                        ; endp
                     2  __Z12before_main1v:           // before_main1()
0000000000000028            sub         sp, sp, #0x20
000000000000002c            stp         x29, x30, [sp, #0x10]
0000000000000030            add         x29, sp, #0x10
0000000000000034            adrp        x0, #0x0
0000000000000038            add         x0, x0, #0x14c
000000000000003c            bl          _printf
0000000000000040            stur        w0, [x29, #-0x4]
0000000000000044            ldp         x29, x30, [sp, #0x10]
0000000000000048            add         sp, sp, #0x20
000000000000004c            ret
```

实验结论:

- __attribute__((constructor))修饰函数的个数与section __mod_init_func端funciton pointers个数一致
- C++静态对象构造函数虽然无法统计出每个具体函数的耗时，但是可以统计出具体对应某个.o中全部静态对象构造函数的耗时
- Section __mod_init_func是在**DATA段**（该段可以动态修改），function pointers指向的区域是**TEXT段**（该段无权限修改）

集成数据卡口 — **非最佳使用API卡口案例** 监控C++静态对象构造函数和__attribute_((constructor))的耗时?

```
; Section __mod_init_func
; Range: [0x2b8; 0x2d0[ (24 bytes)
; File offset : [3320; 3344[ (24 bytes)
; Flags: 0x9
;   S_MOD_INIT_FUNC_POINTERS

                    ltmp11:
00000000000002b8        dq      0x0000000000000000
00000000000002c0        dq      0x0000000000000028
00000000000002c8        dq      0x000000000000011c
```

1. 全部函数指针都替换为hook函数地址
2. 原始函数地址记录到全局数组

```
typedef void (*aliPerformanceInitFuncOrigInitializer)(int argc,
                                        const char* argv[],
                                        const char* envp[],
                                        const char* apple[],
                                        const AliPerformancePremainProgramVars* vars);
```

监控C++静态对象构造函数和__attribute_((constructor))耗时方案

class load 方法获取__DATA段 section __mod_init_func地址

↓

遍历S_MOD_INIT_FUNC_POINTERS 函数指针

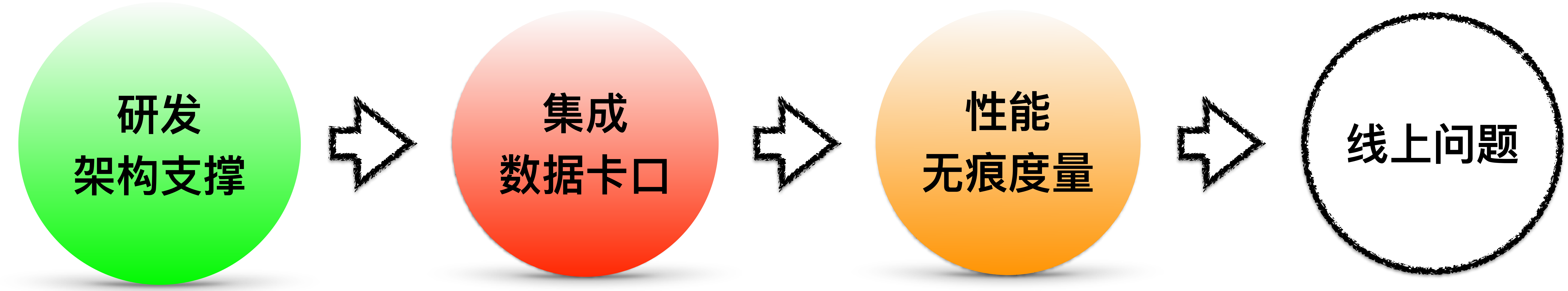
↓

hook过程：
A. 用hook函数指针替换全部的__mod_init_func函数指针
B. 把原始函数地址记录到一个全局数组
C. hook函数中根据执行index，从全局数组中获取真实的原始函数地址执行
D. 统计原始函数耗时

↓

对平台输出数据，做为卡口数据支持

技术上从研发流程角度的思考



- **启动&页面加载监控**
- FPS监控
- 内存监控
- CPU监控

无痕性能度量SDK — **常见页面加载耗时度量**

图1：App启动过程

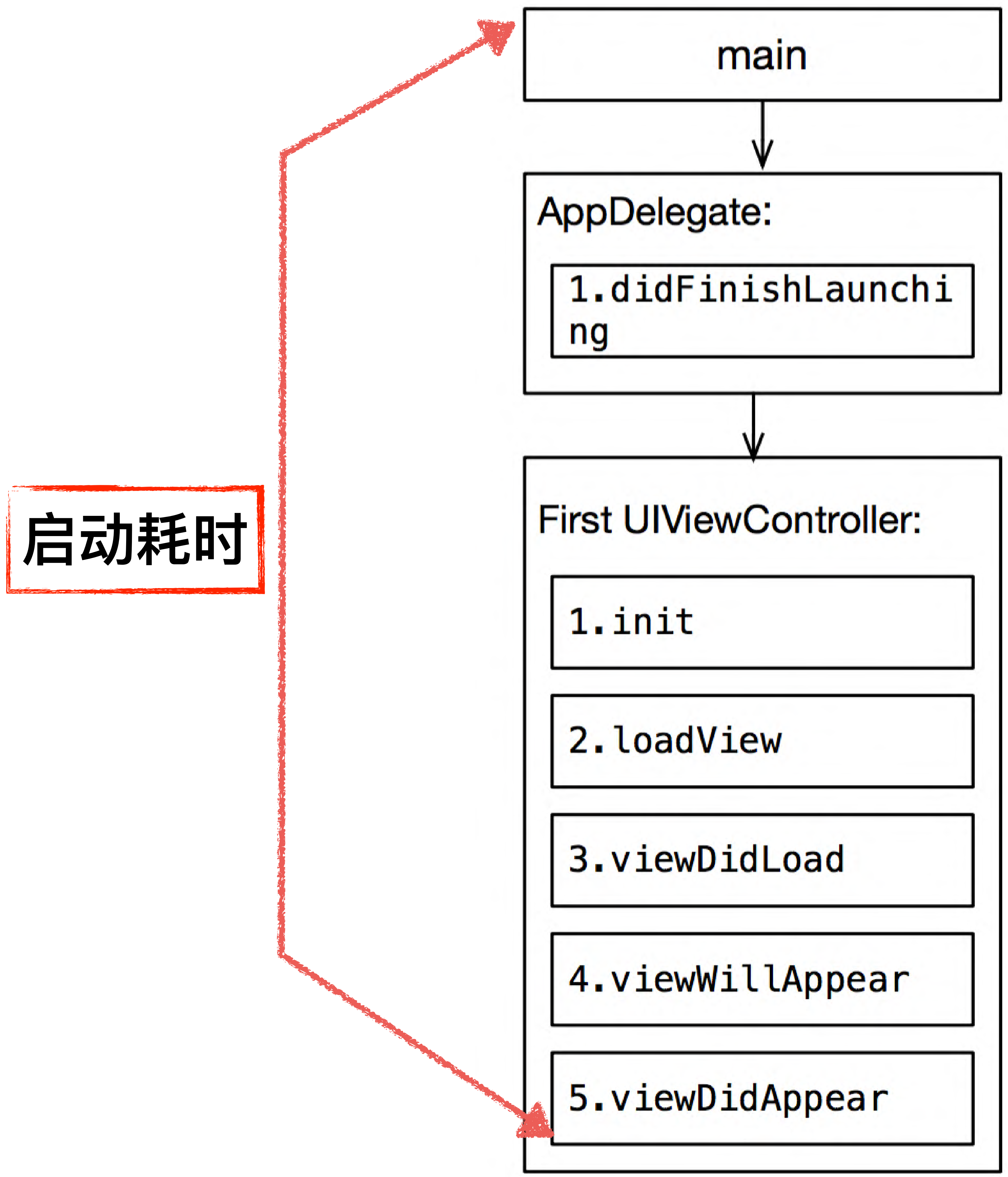
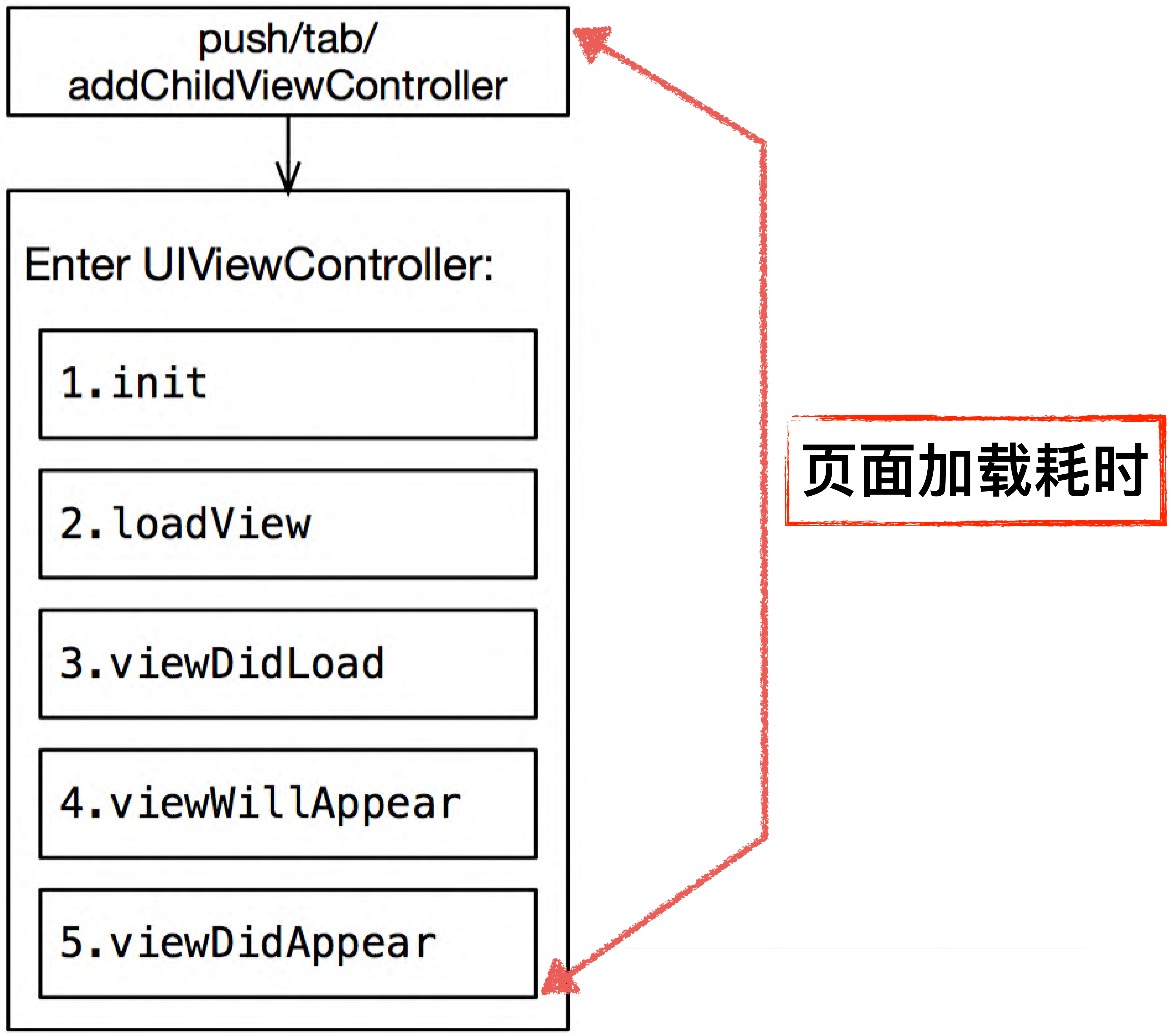


图2：页面加载过程



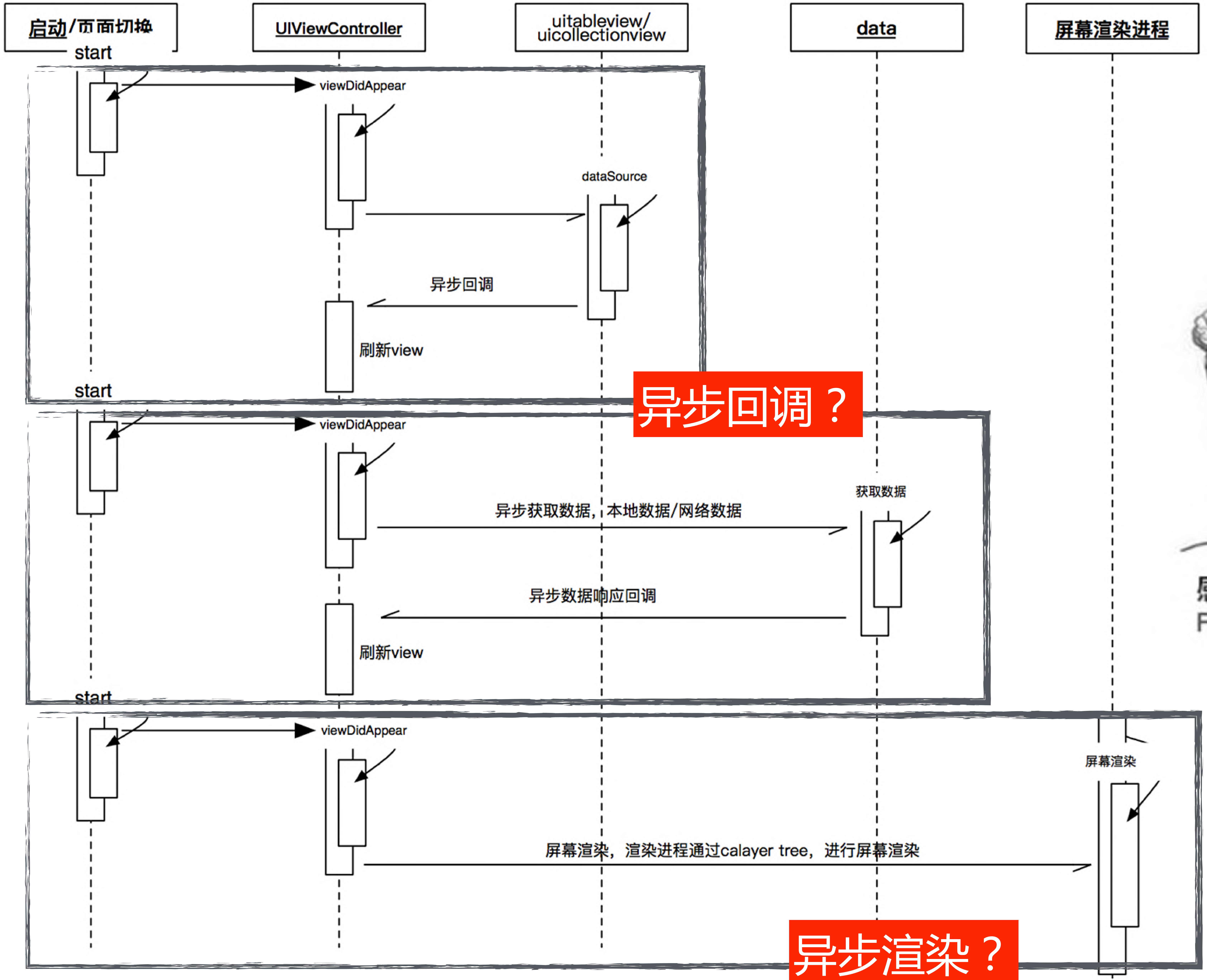
快看！
Duang
Duang~~

无痕性能度量SDK — main函数前系统做了啥?

app启动起点：动态framework的 class load方法调用时机

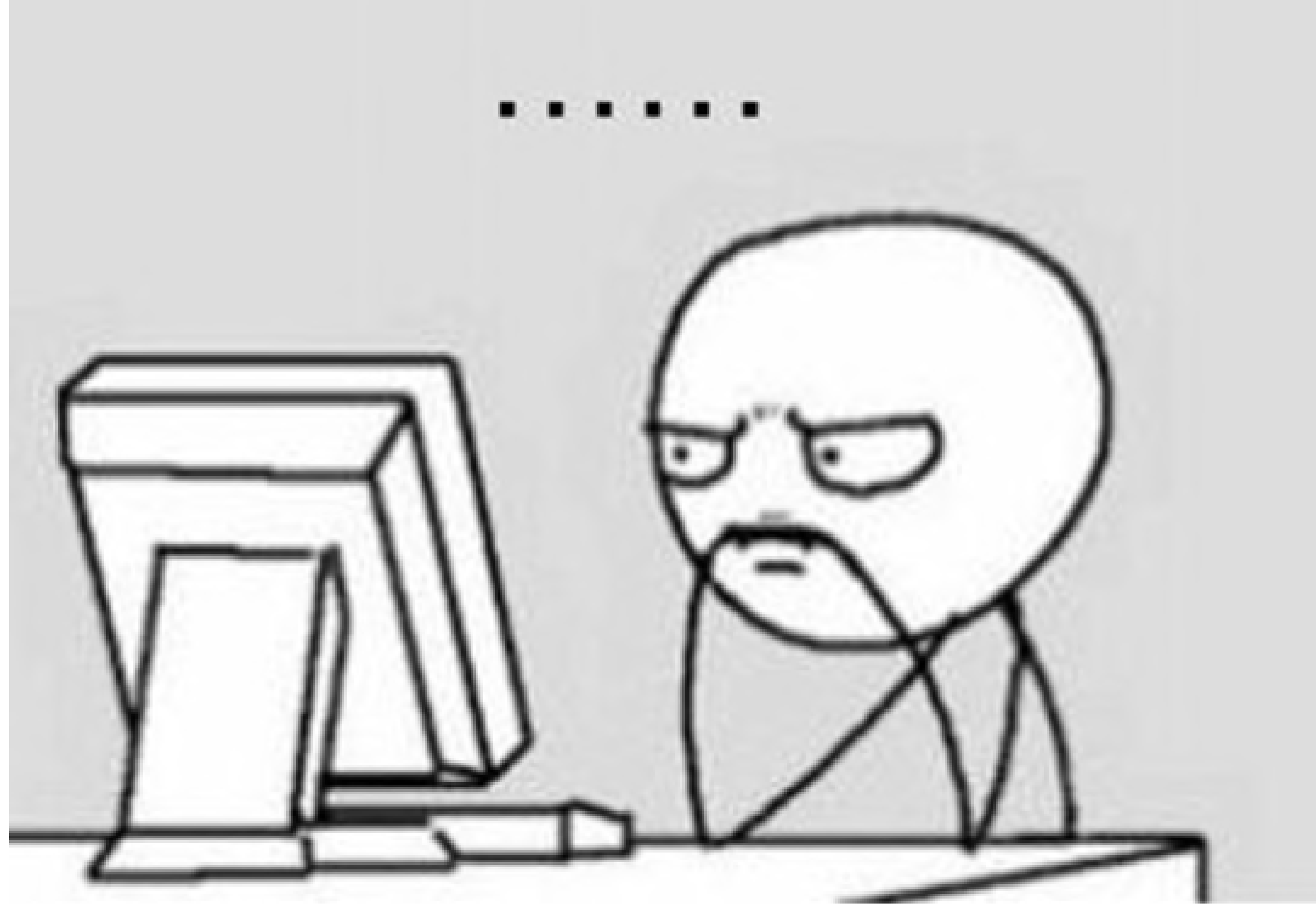
间				
	9%	0.0	▼call_load_methods	libobjc.A.dylib
162.0ms	1.6%	0.0	[redacted]	obao4iPhone
34.0ms	0.3%	0.0	[redacted] load]	Taobao4iPhone
15.0ms	0.1%	1.0	[redacted]ad]	Taobao4iPhone
12.0ms	0.1%	1.0	▶[redacted]r load]	Taobao4iPhone
10.0ms	0.1%	1.0	[redacted]]	Taobao4iPhone
6.0ms	0.0%	0.0	[redacted]	Taobao4iPhone
5.0ms	0.0%	0.0	▶xmlInitParser	libxml2.2.dylib

无痕性能度量SDK — viewDidAppear后页面展示了吗?



无痕性能度量SDK — **用户真正看到页面是啥时候呢?**

如何才能判断屏幕渲染完成？？？



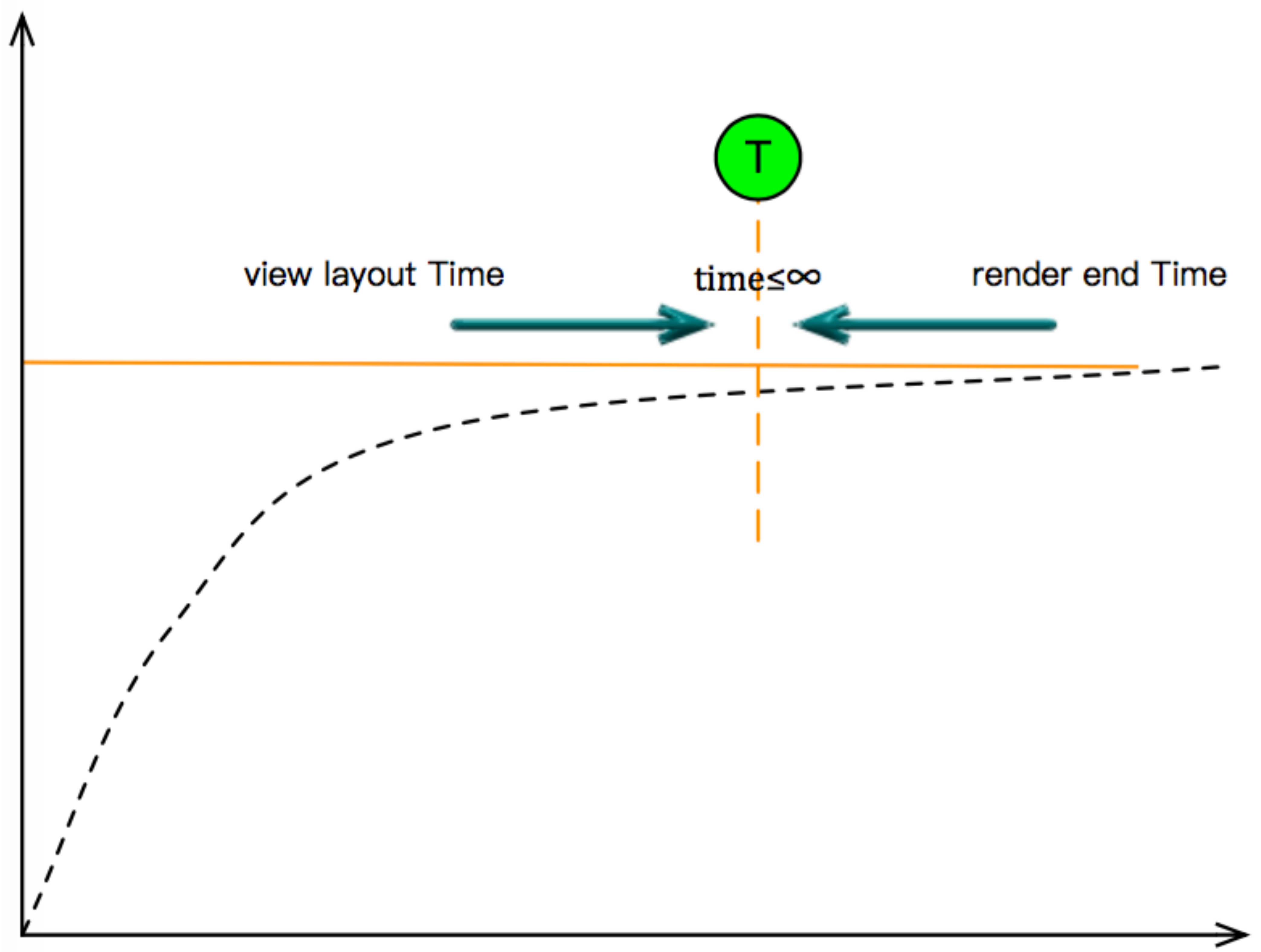
是否能间接获取出屏幕渲染完成时间？？？

无痕性能度量SDK — **基于用户体验页面加载完成度量思路**

理论方法论:

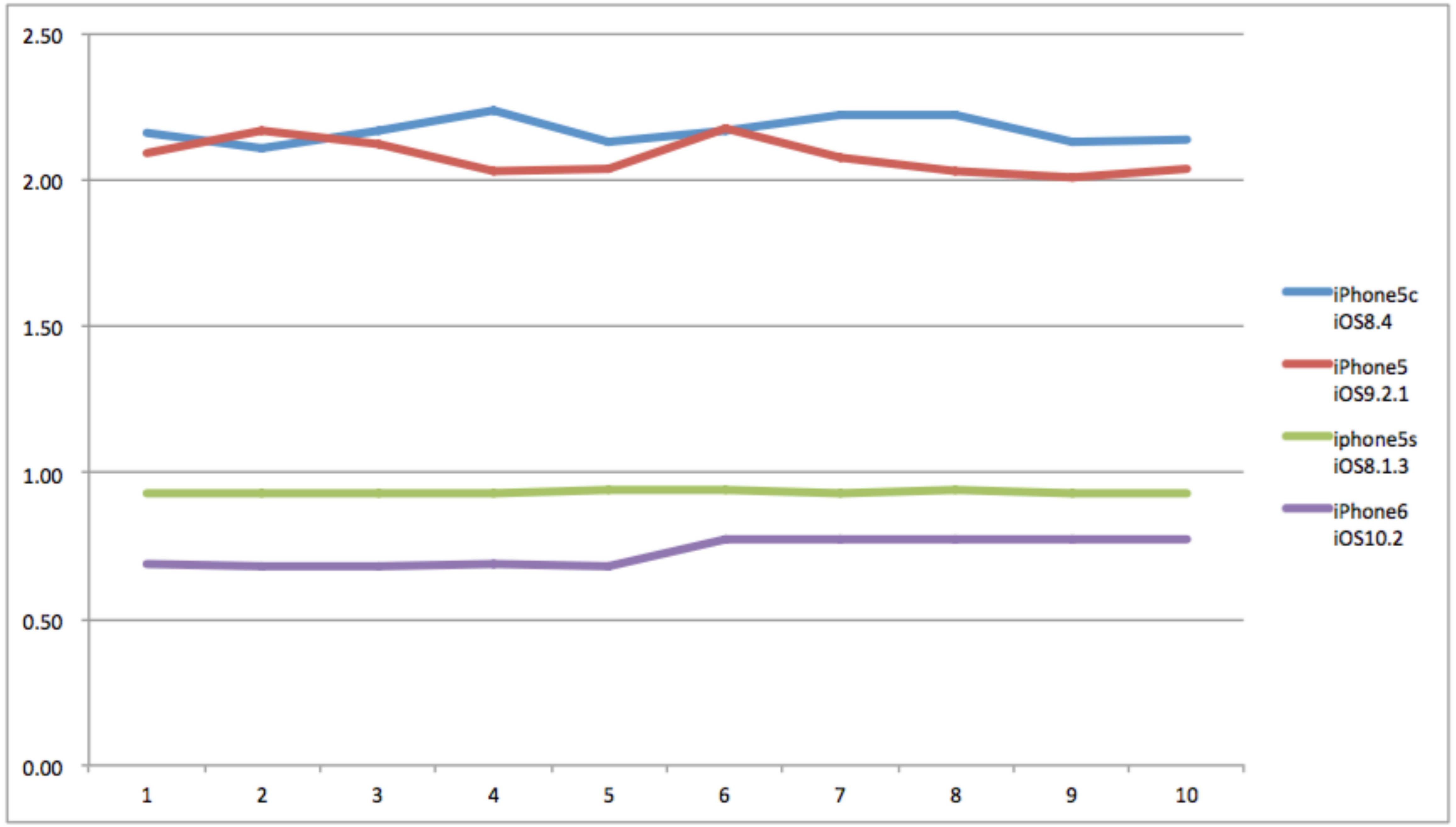
☛ 双向极限逼近法则

T(target) = [Time(forward), Time(backward)] ≤ ∞

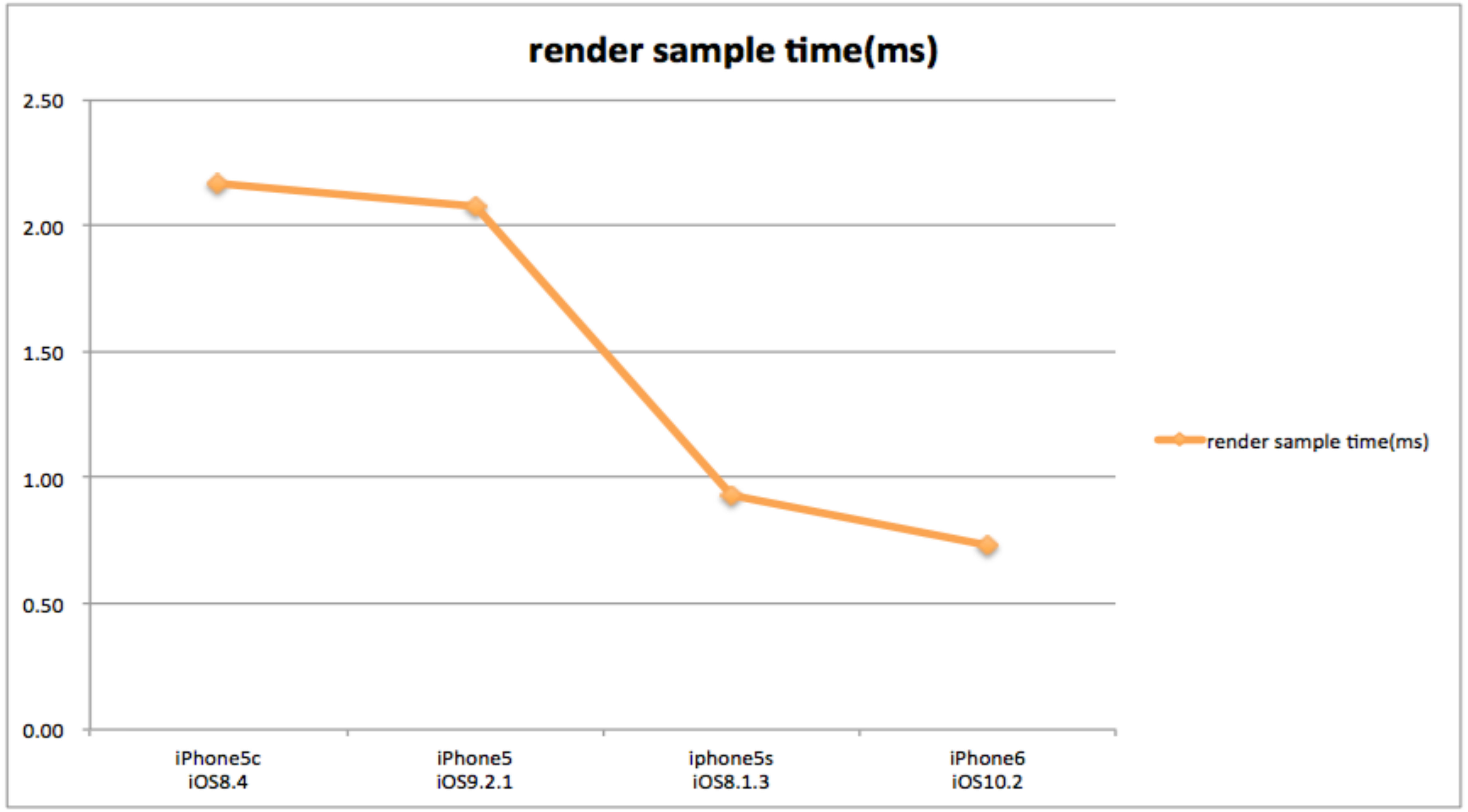


无痕性能度量SDK — **屏幕渲染采样耗时测试数据**

真实实验数据：
☛ 渲染结果采样耗时

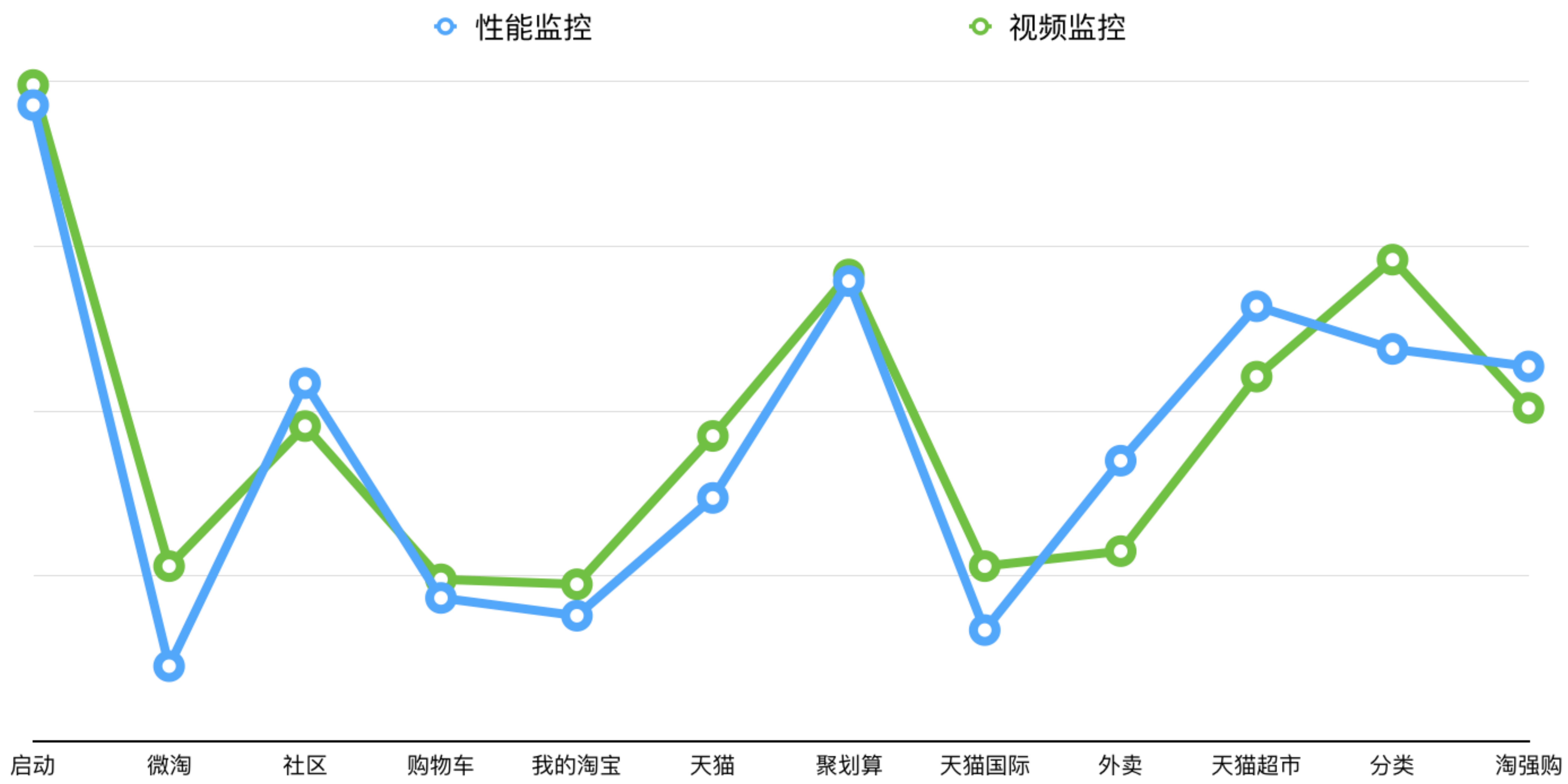


不同机型10次实验数据结果图

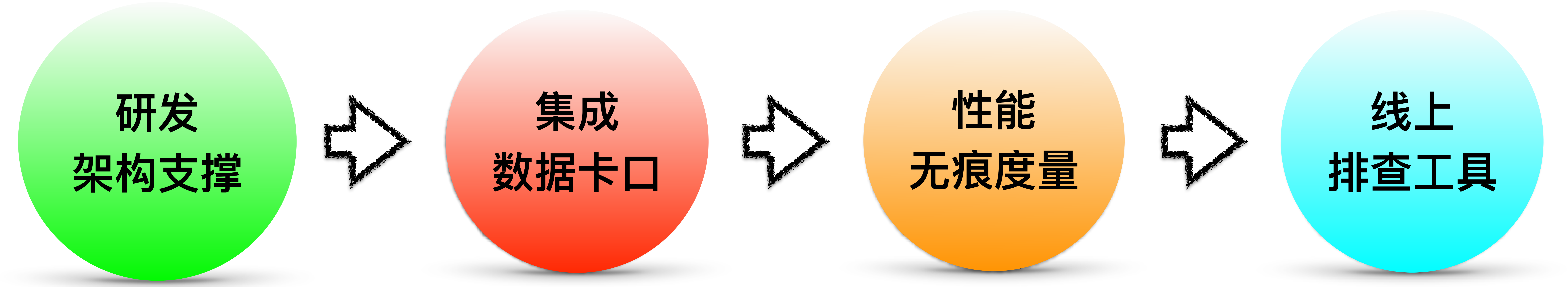


不同机型1次实验数据结果图

无痕性能度量SDK — **基于用户体验页面加载度量方式的测试结果**



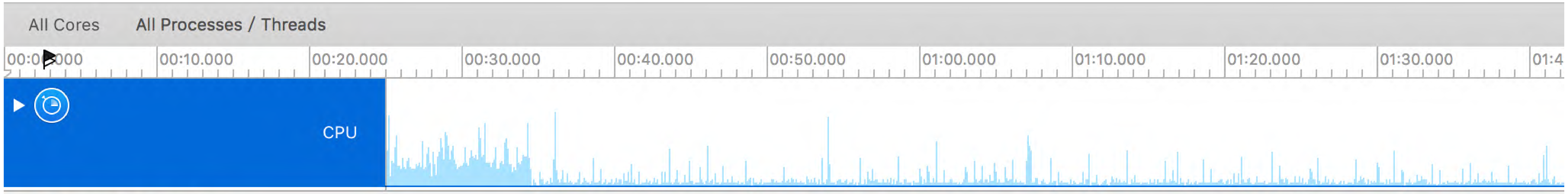
技术上从研发流程角度的思考



- 主线程卡顿监控
- instrument 工具

线上性能排查工具—instrument苹果给开发人员的神器

To a force of English



Details 〉Profile 〉Root

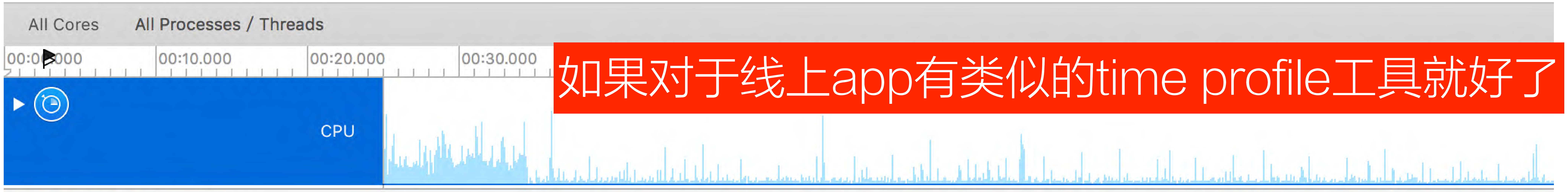
Process

Weight		Self Weight	Symbol Name
3.85 s	100.0%	0 s	▼Taobao4iPhone (922)
2.37 s	61.6%	0 s	▼Main Thread 0xca1d
2.37 s	61.6%	0 s	▼start libdyld.dylib
2.37 s	61.6%	0 s	▼main Taobao4iPhone
2.37 s	61.6%	0 s	▼UIApplicationMain UIKit
2.37 s	61.6%	0 s	▼GSEventRunModal GraphicsServices
2.37 s	61.6%	0 s	▼CFRunLoopRunSpecific CoreFoundation
2.37 s	61.5%	0 s	▶__CFRunLoopRun CoreFoundation
4.00 ms	0.1%	4.00 ms	DYLD-STUB$$-[NSMutableOrderedSet removeObjectsInRange:inSet:] CoreFoundation
1.00 ms	0.0%	1.00 ms	0x1930165f0
433.00 ms	11.2%	0 s	▶_pthread_body 0xcad9
251.00 ms	6.5%	0 s	▶_dispatch_worker_thread3 0xcbbe
193.00 ms	5.0%	0 s	▶_dispatch_worker_thread3 0xcc64
165.00 ms	4.2%	0 s	▶_dispatch_worker_thread3 0xca95
127.00 ms	3.3%	0 s	▶_dispatch_worker_thread3 0xcbe2
100.00 ms	2.5%	0 s	▶_pthread_body 0xcb13
68.00 ms	1.7%	0 s	▶_dispatch_worker_thread3 0xcc27
57.00 ms	1.4%	0 s	▶_dispatch_worker_thread3 0xcab6
20.00 ms	0.5%	0 s	▶easy_io_on_thread_start 0xcaad
15.00 ms	0.3%	0 s	▶_dispatch_worker_thread3 0xcc91
13.00 ms	0.3%	0 s	▶+[NSURLConnection _resourceLoadLoop:] 0xcacf
9.00 ms	0.2%	0 s	▶_dispatch_worker_thread3 0xcaab
8.00 ms	0.2%	0 s	▶_dispatch_mgr_thread 0xca8f
8.00 ms	0.2%	0 s	▶_pthread_body 0xcadd
4.00 ms	0.1%	0 s	▶_dispatch_worker_thread3 0xcb20
2.00 ms	0.0%	0 s	▶loginThreadFunc 0xcb1c
1.00 ms	0.0%	0 s	▶_pthread_body 0xcab0

线上性能排查工具—instrument苹果给开发人员的神器

To a force of English

如果对于线上app有类似的time profile工具就好了



Details 〉Profile 〉Root

Process

Weight		Self Weight	Symbol Name
3.85 s	100.0%	0 s	▼Taobao4iPhone (922)
2.37 s	61.6%	0 s	▼Main Thread 0xca1d
2.37 s	61.6%	0 s	▼start libdyld.dylib
2.37 s	61.6%	0 s	▼main Taobao4iPhone
2.37 s	61.6%	0 s	▼UIApplicationMain UIKit
2.37 s	61.6%	0 s	▼GSEventRunModal GraphicsServices
2.37 s	61.6%	0 s	▼CFRunLoopRunSpecific CoreFoundation
2.37 s	61.5%	0 s	▶__CFRunLoopRun CoreFoundation
4.00 ms	0.1%	4.00 ms	DYLD-STUB$$-[NSMutableOrderedSet removeObjectsInRange:inSet:] CoreFoundation
1.00 ms	0.0%	1.00 ms	0x1930165f0
433.00 ms	11.2%	0 s	▶_pthread_body 0xcad9
251.00 ms	6.5%	0 s	▶_dispatch_worker_thread3 0xcbbe
193.00 ms	5.0%	0 s	▶_dispatch_worker_thread3 0xcc64
165.00 ms	4.2%	0 s	▶_dispatch_worker_thread3 0xca95
127.00 ms	3.3%	0 s	▶_dispatch_worker_thread3 0xcbe2
100.00 ms	2.5%	0 s	▶_pthread_body 0xcb13
68.00 ms	1.7%	0 s	▶_dispatch_worker_thread3 0xcc27
57.00 ms	1.4%	0 s	▶_dispatch_worker_thread3 0xcab6
20.00 ms	0.5%	0 s	▶easy_io_on_thread_start 0xcaad
15.00 ms	0.3%	0 s	▶_dispatch_worker_thread3 0xcc91
13.00 ms	0.3%	0 s	▶+[NSURLConnection _resourceLoadLoop:] 0xcacf
9.00 ms	0.2%	0 s	▶_dispatch_worker_thread3 0xcaab
8.00 ms	0.2%	0 s	▶_dispatch_mgr_thread 0xca8f
8.00 ms	0.2%	0 s	▶_pthread_body 0xcadd
4.00 ms	0.1%	0 s	▶_dispatch_worker_thread3 0xcb20
2.00 ms	0.0%	0 s	▶loginThreadFunc 0xcb1c
1.00 ms	0.0%	0 s	▶_pthread_body 0xcab0

线上性能排查工具—**自制instrument的思路**

To a force of English

线上性能排查工具—tbinstrument 淘宝的instrument

To a force of English

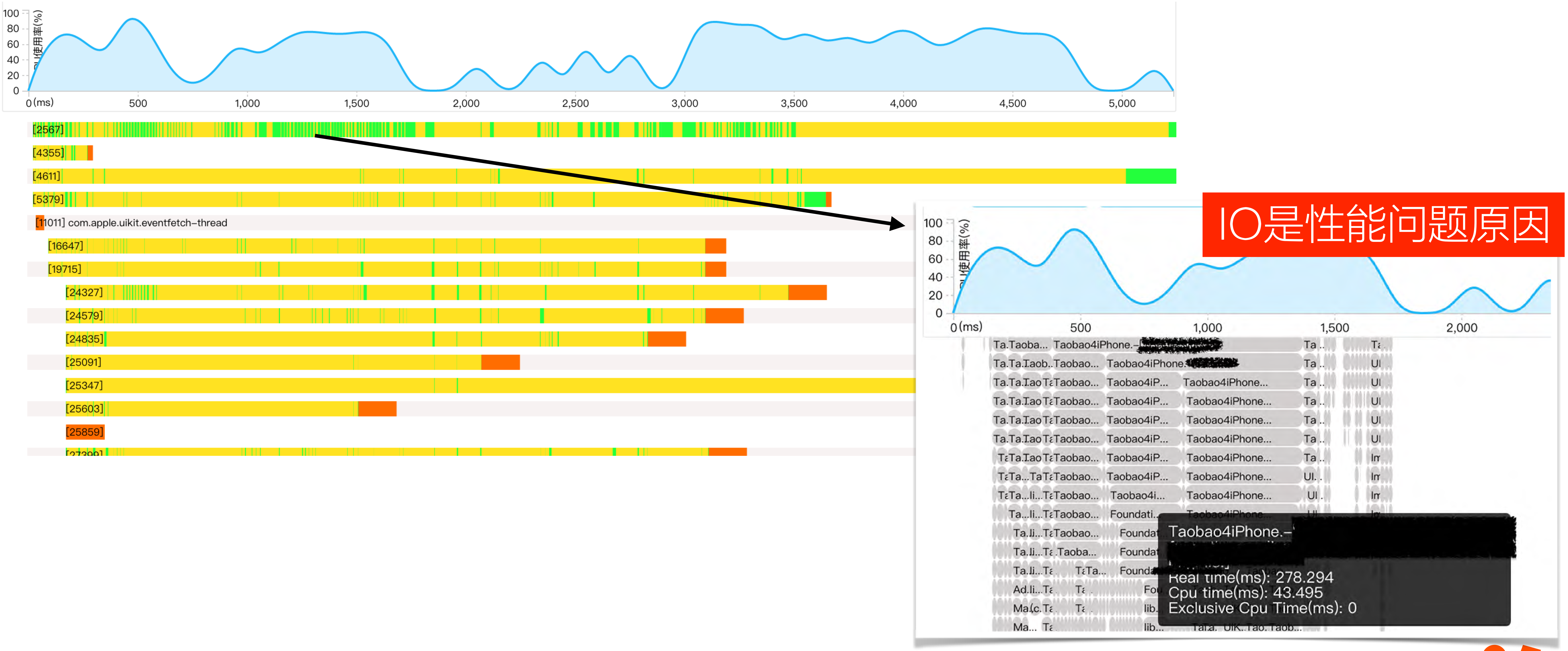
线上性能排查工具—tbinstrument 淘宝的instrument

To a force of English

线上性能排查工具 — tbinstrument 案例

To a force of English

- 排查问题case A



线上性能排查工具 — tbinstrument 案例

To a force of English

- 排查问题case B

线上性能排查工具 — tbinstrument 案例

To a force of English

- 排查问题case B

线上性能排查工具 — tbinstrument 案例

To a force of English

- 排查问题case B　　OC方法消息分发 objc_msgSend内部有**runtimeLock**锁

```
/********************************************
 * id       objc_msgSend(id self,
 *          SEL op,
 *          ...)
 *
 * On entry: a1 is the message receiver,
 *           a2 is the selector
 ********************************************

    ENTRY objc_msgSend
# check whether receiver is nil
    teq     a1, #0
    beq     LMsgSendNilReceiver

# save registers and load receiver's class f
    stmfd   sp!, {a4,v1,r9}
    ldr     v1, [a1, #ISA]

# receiver is non-nil: search the cache
    CacheLookup a2, v1, LMsgSendCacheMiss

# cache hit (imp in ip) and CacheLookup retu
    ldmfd   sp!, {a4,v1,r9}
    bx      ip

# cache miss: go search the method lists
LMsgSendCacheMiss:
    ldmfd   sp!, {a4,v1,r9}
    b   objc_msgSend_uncached

LMsgSendNilReceiver:
    mov     a2, #0
    bx      lr

LMsgSendExit:
    END_ENTRY objc_msgSend
```

```
    STATIC_ENTRY objc_msgSend_uncached

 Push stack frame
    stmfd   sp!, {a1-a4,r7,lr}
    add     r7, sp, #16

 Load class and selector
    ldr a3, [a1, #ISA]        /* class = receiver->isa  */
                              /* selector already in a2 */
                              /* receiver already in a1 */

 Do the lookup
    MI_CALL_EXTERNAL(_class_lookupMethodAndLoadCache3)
    MOVE    ip, a1

 Prep for forwarding, Pop stack frame and call imp
    teq v1, v1          /* set nonstret (eq) */
    ldmfd   sp!, {a1-a4,r7,lr}
    bx  ip

/*************************************************************************/
IMP _class_lookupMethodAndLoadCache3(id obj, SEL sel, Class cls)
{
    return lookUpMethod(cls, sel, YES/*initialize*/, NO/*cache*/, obj);
}
```

```
IMP lookUpMethod(Class cls, SEL sel, BOOL initialize, BOOL cache, id inst)
{
    Class curClass;
    IMP methodPC = NULL;

    // realize, +initialize, and any special early exit
    ...

    // The lock is held to make method-lookup + cache-fill atomic
    // with respect to method addition. Otherwise, a category could
    // be added but ignored indefinitely because the cache was re-filled
    // with the old value after the cache flush on behalf of the category.
 retry:
    lockForMethodLookup();

    //去获取真实的IMP

 done:
    unlockForMethodLookup();

    // paranoia: look for ignored selectors with non-ignored implementations
    assert(!(ignoreSelector(sel) && methodPC != (IMP)&_objc_ignored_method));

    return methodPC;
}


void lockForMethodLookup(void)
{
    rwlock_read(&runtimeLock);
}
void unlockForMethodLookup(void)
{
    rwlock_unlock_read(&runtimeLock);
}
```

runtimeLock 全局锁

线上性能排查工具 — tbinstrument 案例

To a force of English

- 排查问题case B　　OC runtime相关函数内部执行，也有runtimeLock锁

```
IMP imp_implementationWithBlock(id block)
{
    block = Block_copy(block);
    _lock();
    IMP returnIMP = _imp_implementationWithBlockNoCopy(_ar
    _unlock();
    return returnIMP;
}
```

```
static inline void _lock() {
#if __OBJC2__
    rwlock_write(&runtimeLock);
#else
    mutex_lock(&classLock);
#endif
}
```

```
IMP class_replaceMethod(Class cls, SEL name, IMP imp, const char *types)
{
    if (!cls) return NULL;

    rwlock_write(&runtimeLock);
    IMP old = addMethod(newcls(cls), name, imp, types ?: "", YES);
    rwlock_unlock_write(&runtimeLock);
    return old;
}
```

```
void method_exchangeImplementations(Method m1_gen, Method m2_gen)
{
    ...

    rwlock_write(&runtimeLock);
    ...

    IMP m1_imp = m1->imp;
    m1->imp = m2->imp;
    m2->imp = m1_imp;

    ...
    // fixme update monomorphism if necessary

    rwlock_unlock_write(&runtimeLock);
}
```

线上性能排查工具 — tbinstrument 案例

To a force of English

- 排查问题case B

主线程CPU消耗低？？

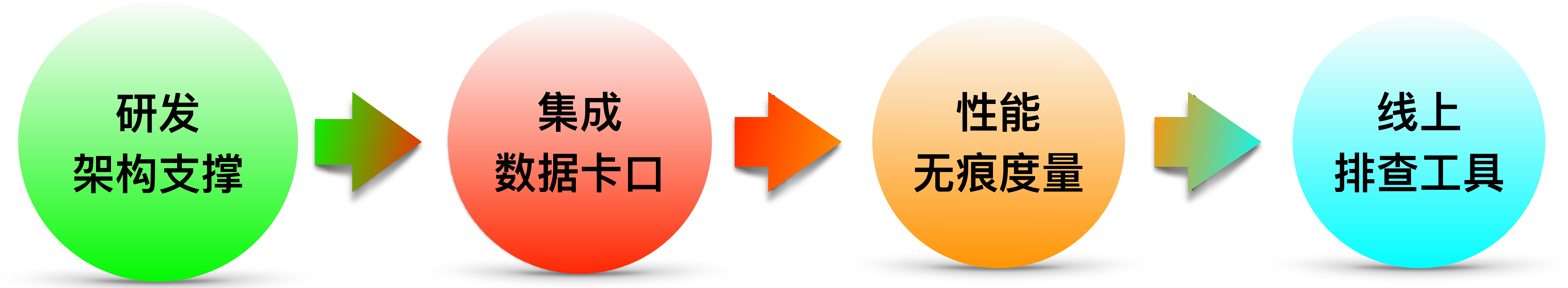
子线程CPU消耗高？？

结论：runtimeLock 全局锁导致性能瓶颈

A. 主线程正常函数调用，运行时oc基于objc_msgSend 消息方法内部遇到runtimeLock锁

B. 子线程oc runtime api频发循环调用，runtime api内部遇到runtimeLock锁

技术上从研发流程角度的思考



我们是一支敢玩、敢想、敢拼、热爱生活的队伍
如果您想翘起地球，我们为您提供支点

fangying.fy@alibaba-inc.com

手淘技术微信公众号 – MTT

Thanks!